THINKING ABOUT IT:
CHANNELS MORE THAN CONNECT THREADS

THEY PROTECT THEM

LECTURE BY ØYVIND TEIG, SIV. ING. NTH (1975)

Version of 1. Feb. 2018 (2)

LECTURE BY ØYVIND TEIG, SIV. ING. NTH (1975)

AUTRONICA @ EMBEDDED SYSTEMS

Version of 1. Feb. 2018 (2)

LECTURE BY ØYVIND TEIG, SIV. ING. NTH (1975)

AUTRONICA @ EMBEDDED SYSTEMS (1976-2017)

Version of 1. Feb. 2018 (2)

LECTURE BY ØYVIND TEIG, SIV. ING. NTH (1975)

AUTRONICA @ EMBEDDED SYSTEMS (1976-2017)
BLOGGING ABOUT CONCURRENCY ETC. (NOW)

Version of 1. Feb. 2018 (2)

LECTURE BY ØYVIND TEIG, SIV. ING. NTH (1975)

AUTRONICA @ EMBEDDED SYSTEMS (1976–2017)
BLOGGING ABOUT CONCURRENCY ETC. (NOW)

INVITED SPEAKER, 1. FEB. 2018 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

Version of 1. Feb. 2018 (2)

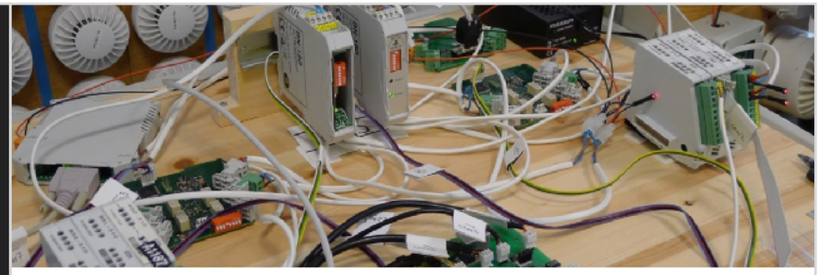
PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG
SENIOR DEVELOPMENT ENGINEER, AUTRONICA

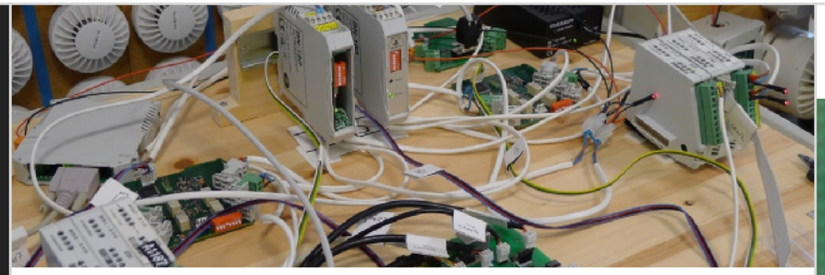
INVITED SPEAKER, 26. APRIL 2016 AT
NTNU, TTK4145 SANNTIDSPROGRAMMERING
(REAL-TIME PROGRAMMING)

Version of 26 April 2016 14.10

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG

SENIOR DEVELOPMENT ENGINEER, AUTRONICA

INVITED SPEAKER, 26. APRIL 2016 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

Version of 26 April 2016 14.10

AUTRONICA FIRE AND SECURITY

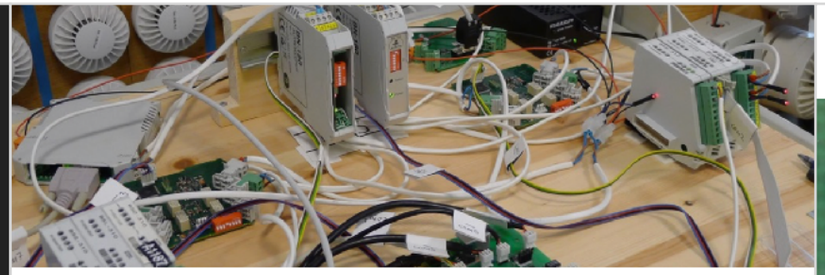
PART OF UTC SINCE 2005

FIRE DETECTION SINCE 1957

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG

SENIOR DEVELOPMENT ENGINEER, AUTRONICA

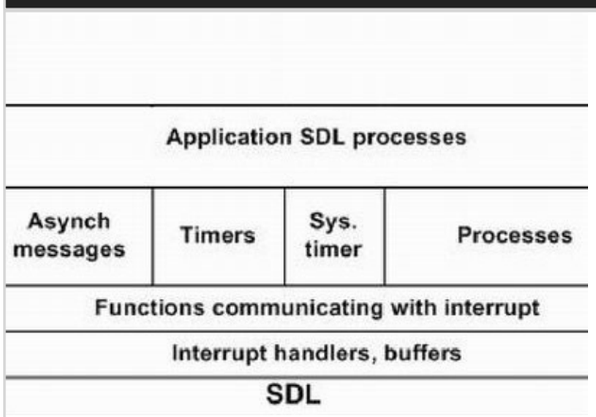
INVITED SPEAKER, 26. APRIL 2016 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

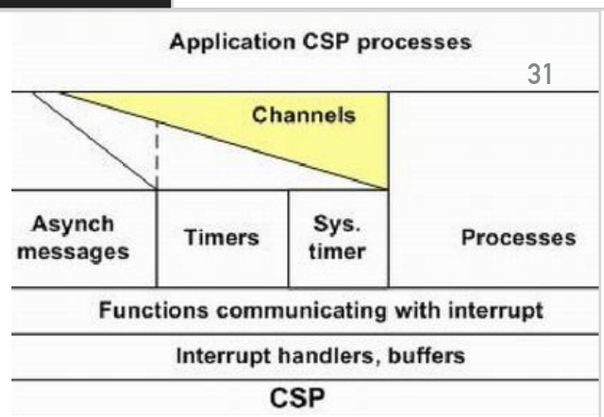
Version of 26 April 2016 14.10

AUTRONICA FIRE AND SECURITY

PART OF UTC SINCE 2005

FIRE DETECTION SINCE 1957





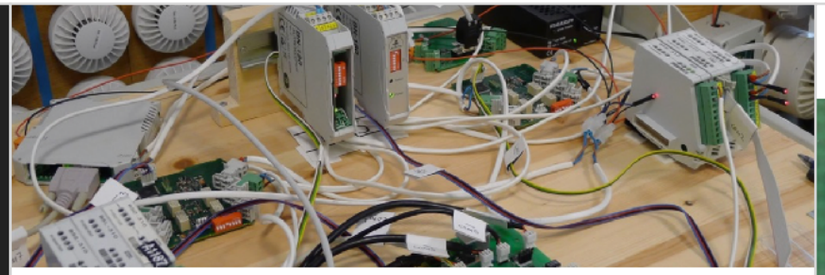
OUR TWO SOLUTIONS

- FSM scheduler: Most of our controllers use this asynchronous SDL-based scheduler
- CHAN_CSP: However: in two of the controller there's synchronous channels on top of it

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG

SENIOR DEVELOPMENT ENGINEER, AUTRONICA

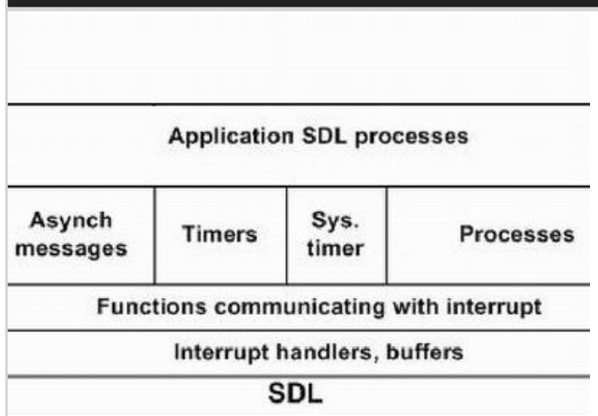
INVITED SPEAKER, 26. APRIL 2016 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

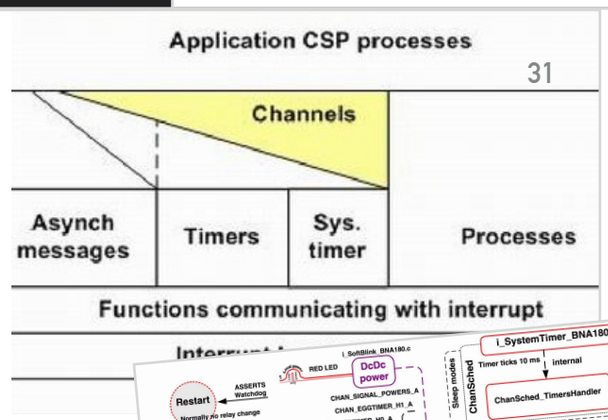
Version of 26 April 2016 14.10

AUTRONICA FIRE AND SECURITY

PART OF UTC SINCE 2005

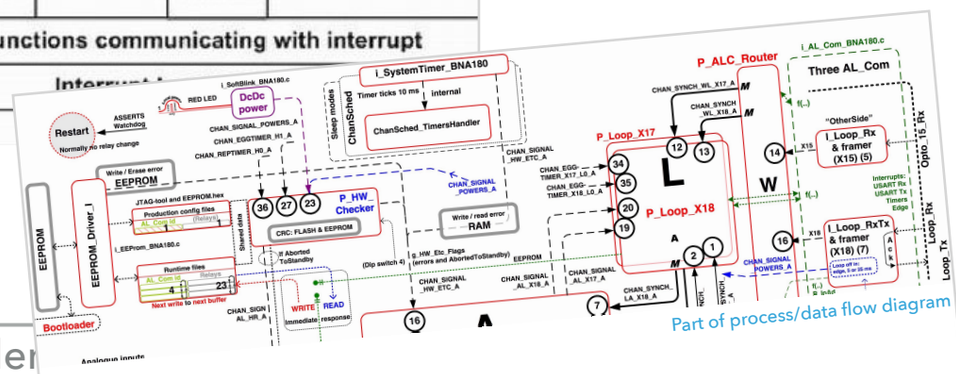
FIRE DETECTION SINCE 1957





OUR TWO SOLUTIONS

- FSM scheduler: Most of our controller
- CHAN_CSP: However: in two of the co
  synchronous channels on top of it



Part of process/data flow diagram

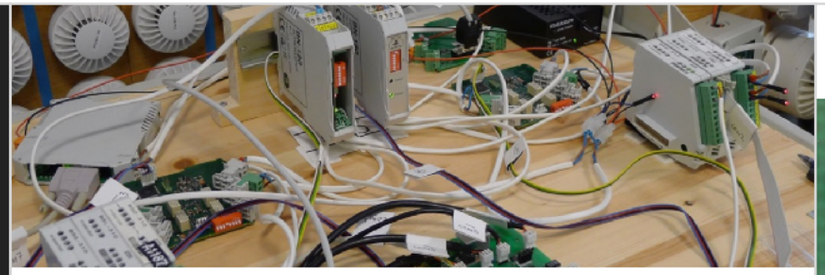
PLUS A THIRD: «CHANSCHED»

- ChanSched: finally in one of the controllers synchronous channels on top of no other runtime («naked»)
- The runtime was more visible to the application code than I thought (later)

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG

SENIOR DEVELOPMENT ENGINEER, AUTRONICA

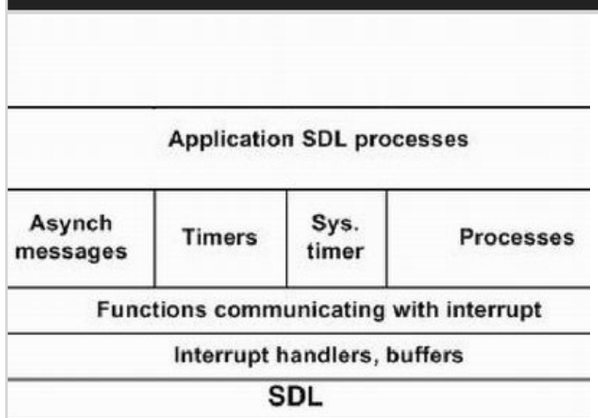
INVITED SPEAKER, 26. APRIL 2016 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

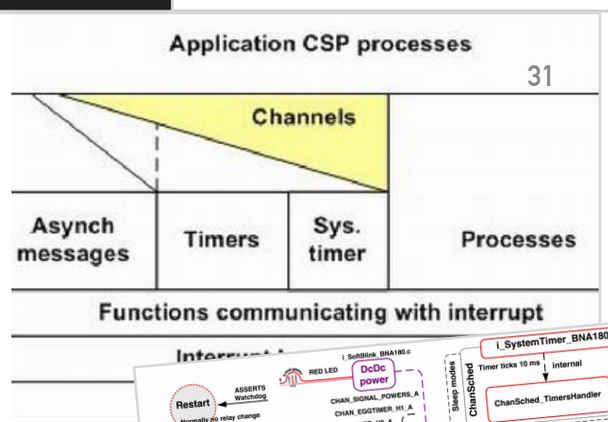
Version of 26 April 2016 14.10

AUTRONICA FIRE AND SECURITY

PART OF UTC SINCE 2005

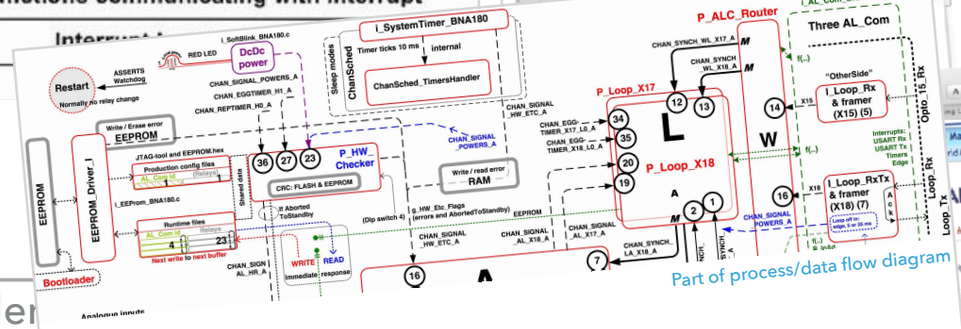
FIRE DETECTION SINCE 1957





OUR TWO SOLUTIONS

- FSM scheduler: Most of our controller
  asynchronous SDL-based scheduler
- CHAN_CSP: However: in two of the co
  synchronous channels on top of it



Part of process/data flow diagram

PLUS A THIRD: «CHANSCHED»

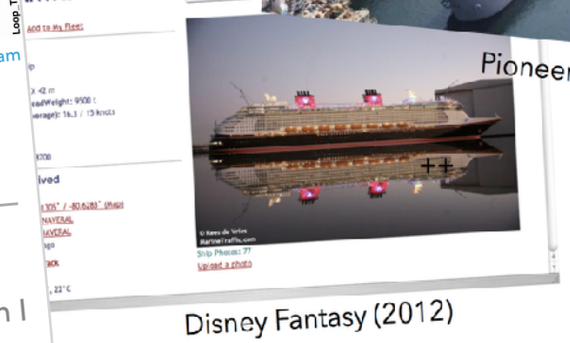
- ChanSched: finally in one of the controllers synchronous channels on top of no other runtime («naked»)
- The runtime was more visible to the application code than I thought (later)

ALL THIS RUNS IN AN AUTRONICA «DUAL SAFETY» COMPONENT 53

«SAFE RETURN TO PORT» (IMO) OR JUST EXTRA SAFETY

Disney Dream (2011)

Pioneering Spirit (2013)



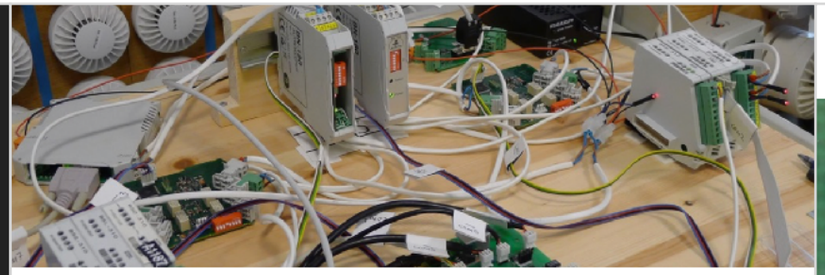
Disney Fantasy (2012)

#2353255)

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG

SENIOR DEVELOPMENT ENGINEER, AUTRONICA

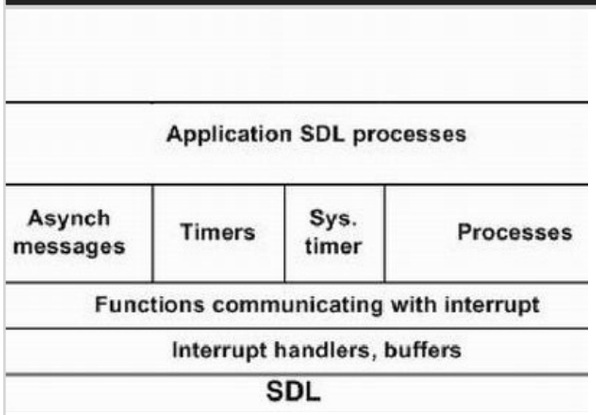
INVITED SPEAKER, 26. APRIL 2016 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

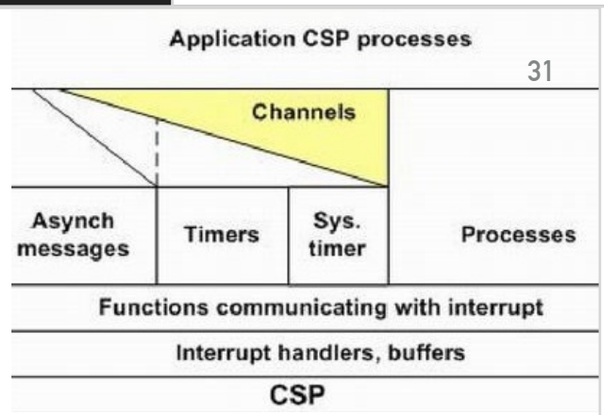
Version of 26 April 2016 14.10

AUTRONICA FIRE AND SECURITY

PART OF UTC SINCE 2005

FIRE DETECTION SINCE 1957



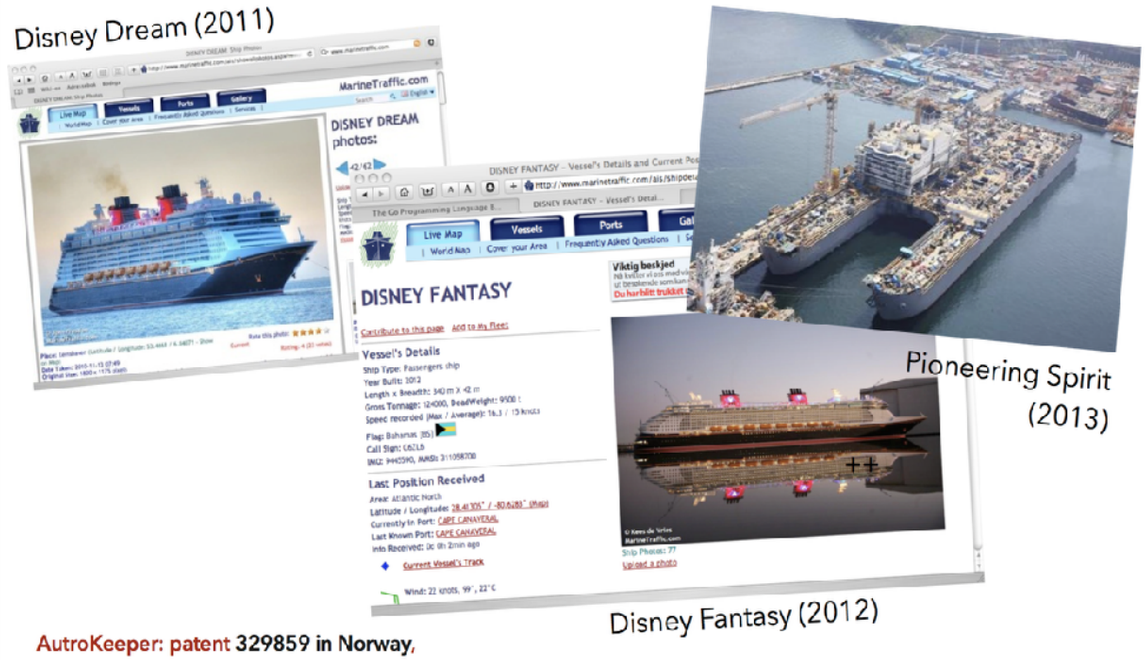


OUR TWO SOLUTIONS

- FSM scheduler: Most of our controllers use this asynchronous SDL-based scheduler
- CHAN_CSP: However: in two of the controller there's synchronous channels on top of it

ALL THIS RUNS IN AN AUTRONICA «DUAL SAFETY» COMPONENT 53

«SAFE RETURN TO PORT» (IMO) OR JUST EXTRA SAFETY



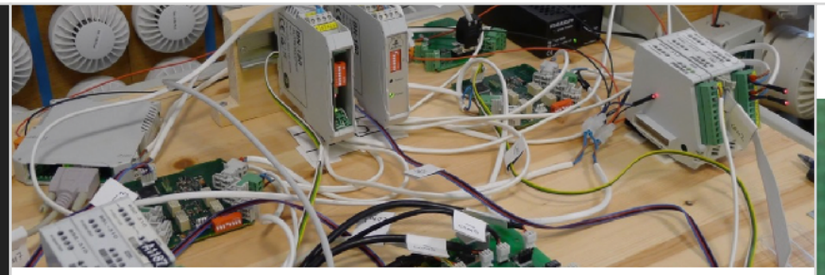
AutroKeeper: patent 329859 in Norway,
PCT/NO2009/000319 international (granted as #2353255)

PREVIOUS LECTURES WERE QUITE DIFFERENT FROM THIS LECTURE

www.teigfam.net/oyvind/pub/pub.html

FROM HARD MICROSECONDS TO SPEEDY YEARS

REAL TIME IN THE INDUSTRY



ØYVIND TEIG

SENIOR DEVELOPMENT ENGINEER, AUTRONICA

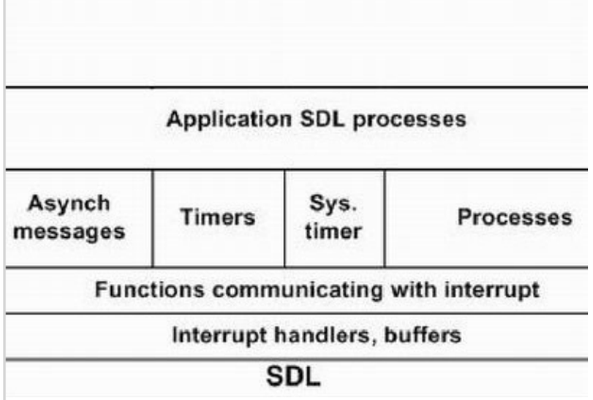
INVITED SPEAKER, 26. APRIL 2016 AT NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

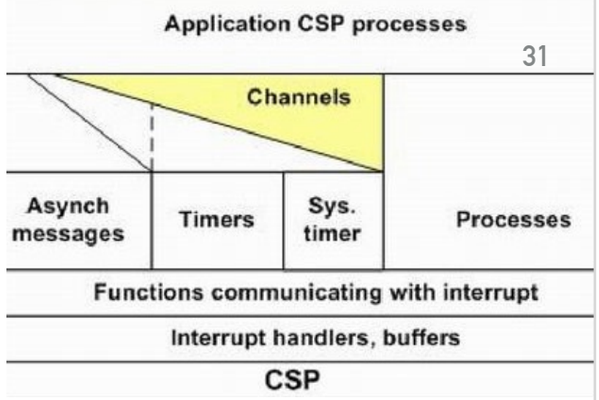
Version of 26 April 2016 14.10

AUTRONICA
FIRE AND SECURITY

PART OF UTC SINCE 2005

FIRE DETECTION SINCE 1957



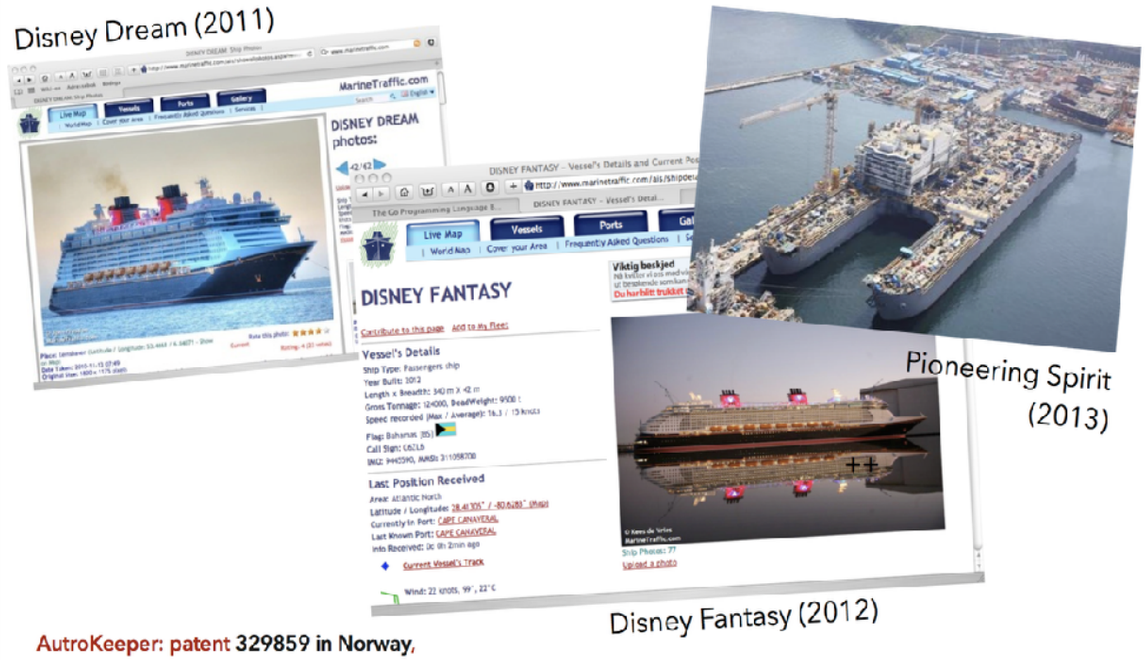


OUR TWO SOLUTIONS

- FSM scheduler: Most of our controllers use this asynchronous SDL-based scheduler
- CHAN_CSP: However: in two of the controller there's synchronous channels on top of it

ALL THIS RUNS IN AN AUTRONICA «DUAL SAFETY» COMPONENT 53

«SAFE RETURN TO PORT» (IMO) OR JUST EXTRA SAFETY



AutroKeeper: patent 329859 in Norway,
PCT/NO2009/000319 international (granted as #2353255)

GOAL

GOAL

- What are channels (and XC «interface»)?

GOAL

- What are channels (and XC «interface»)?
- Why are they more than mere communication channels?

GOAL

- What are channels (and XC «interface»)?
- Why are they more than mere communication channels?
- What problems do they offer a resolution to?

GOAL

- What are channels (and XC «interface»)?
- Why are they more than mere communication channels?
- What problems do they offer a resolution to?
- A little about myself..

GOAL

- What are channels (and XC «interface»)?
- Why are they more than mere communication channels?
- What problems do they offer a resolution to?
- A little about myself..
- ..and my experience over 40+ years in industry

GOAL

- What are channels (and XC «interface»)?
- Why are they more than mere communication channels?
- What problems do they offer a resolution to?
- A little about myself..
- ..and my experience over 40+ years in industry
- (btw: This lecture is on my home page (ref. at the end))

ARDUINO IDE BASICS

ARDUINO IDE BASICS

- «Sketch» is a «project»

ARDUINO IDE BASICS

- «Sketch» is a «project»
- Top level: .ino-files (not main.c)

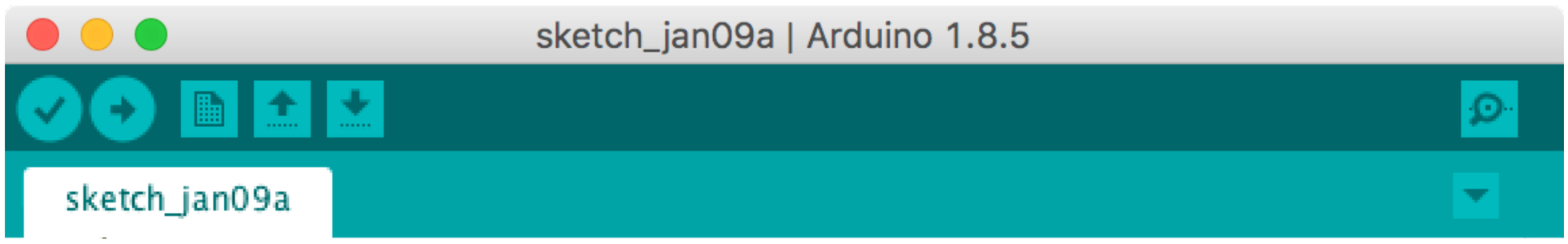
ARDUINO IDE BASICS

- «Sketch» is a «project»
- Top level: .ino-files (not main.c)
- First for Atmel AVR processors

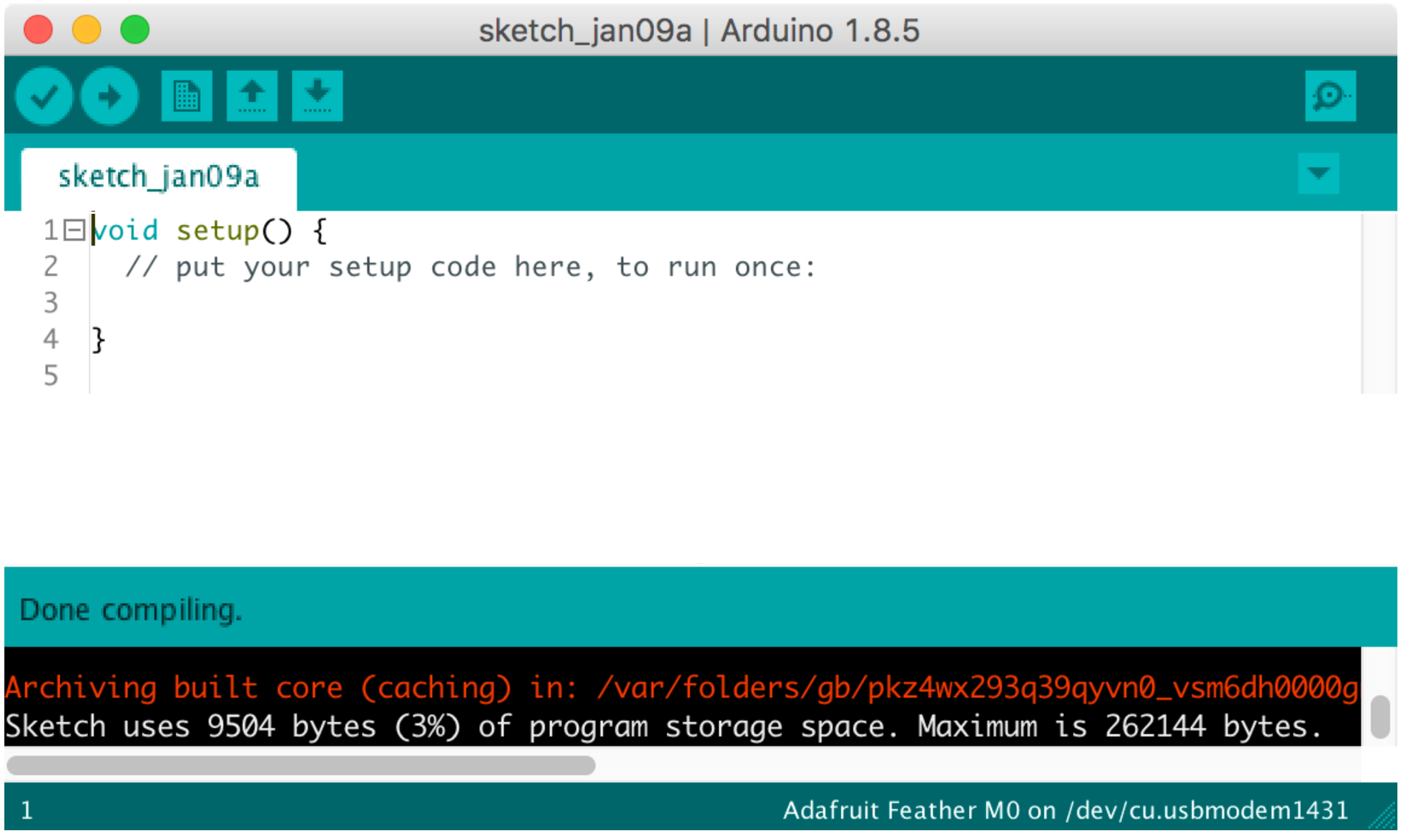
ARDUINO IDE BASICS

- «Sketch» is a «project»
- Top level: .ino-files (not main.c)
- First for Atmel AVR processors
- I have played with Arduino SAMD Boards (32-bits ARM Cortex-M0+)

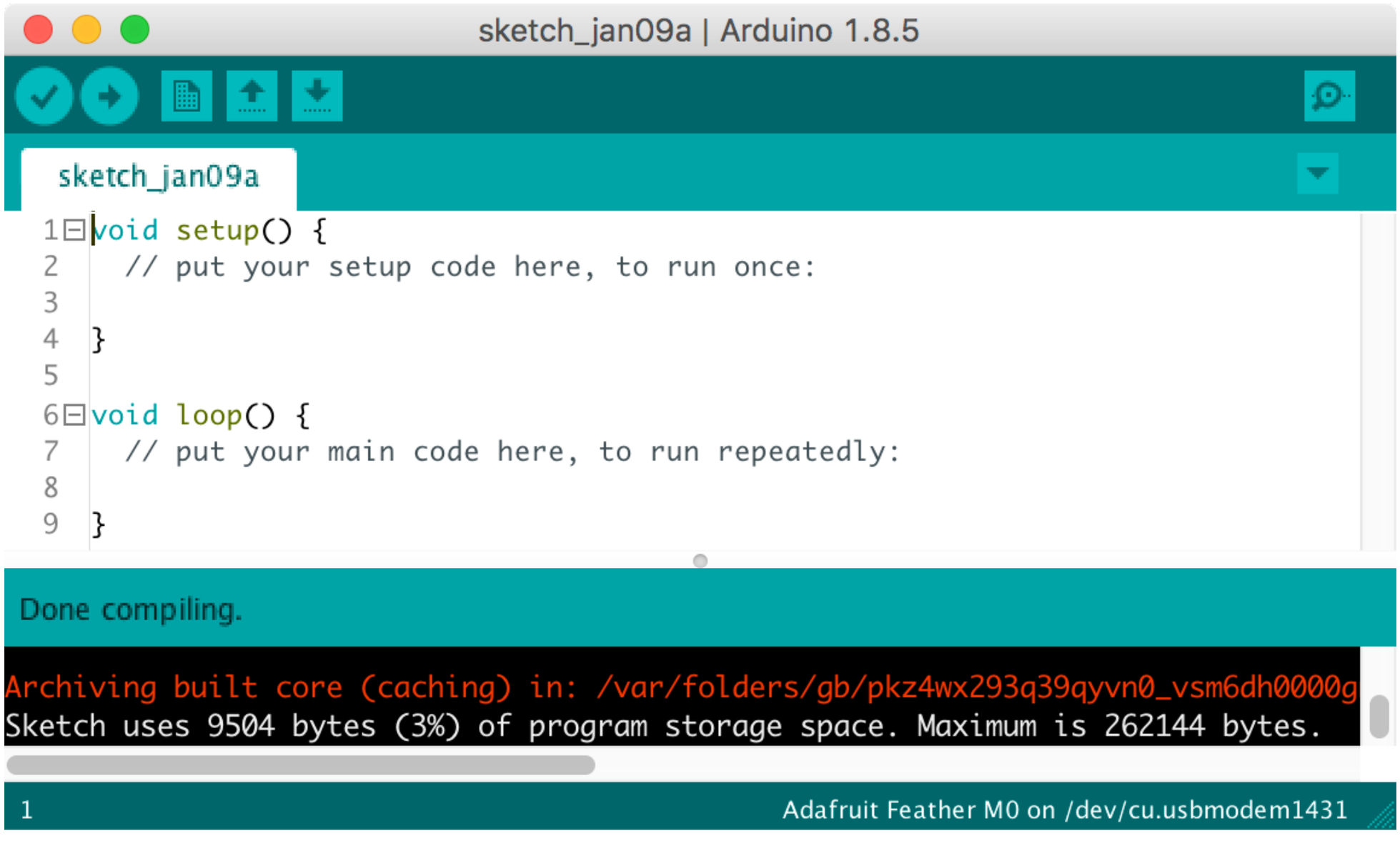
BARE MINIMUM CODE NEEDED



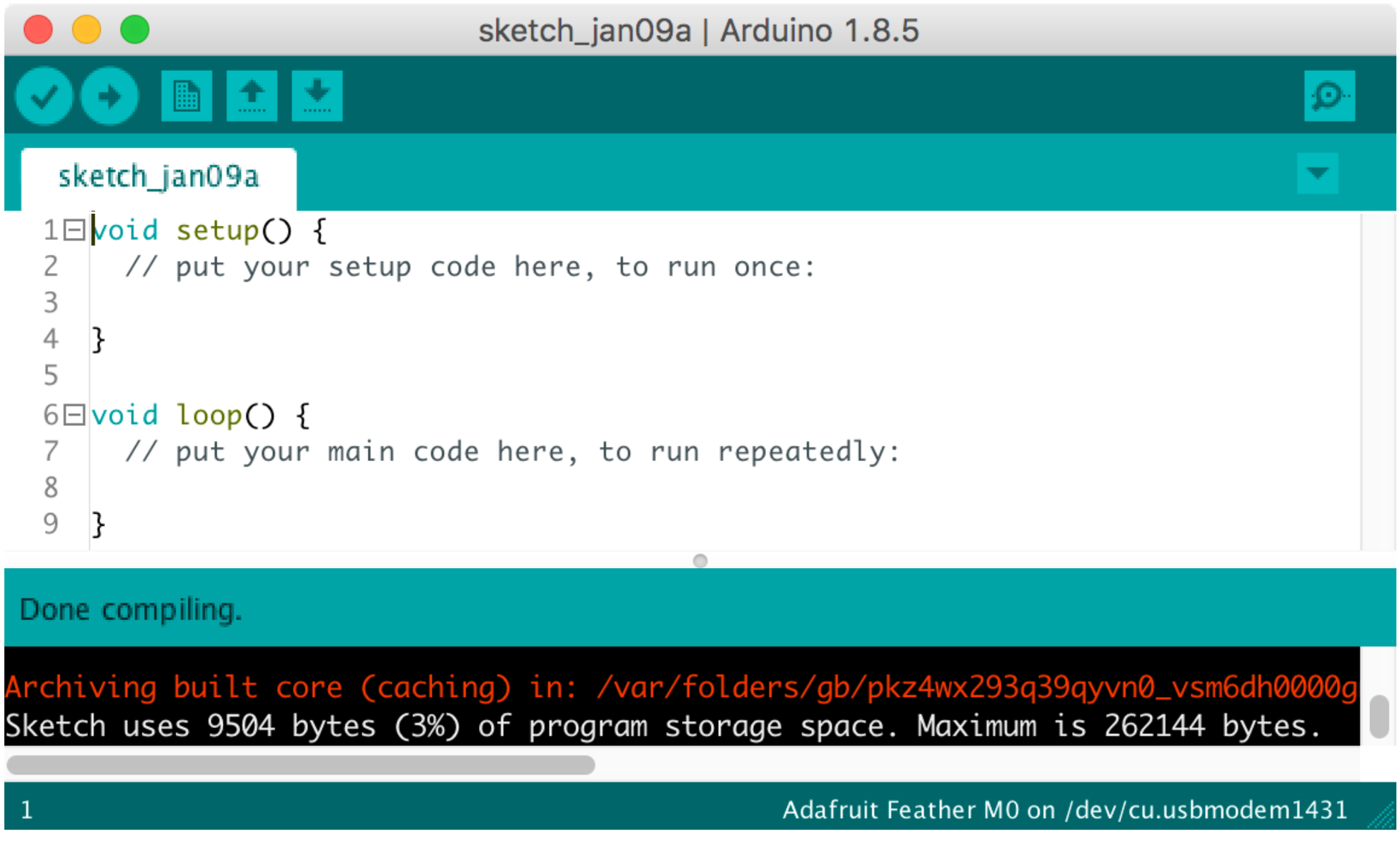
BARE MINIMUM CODE NEEDED



BARE MINIMUM CODE NEEDED

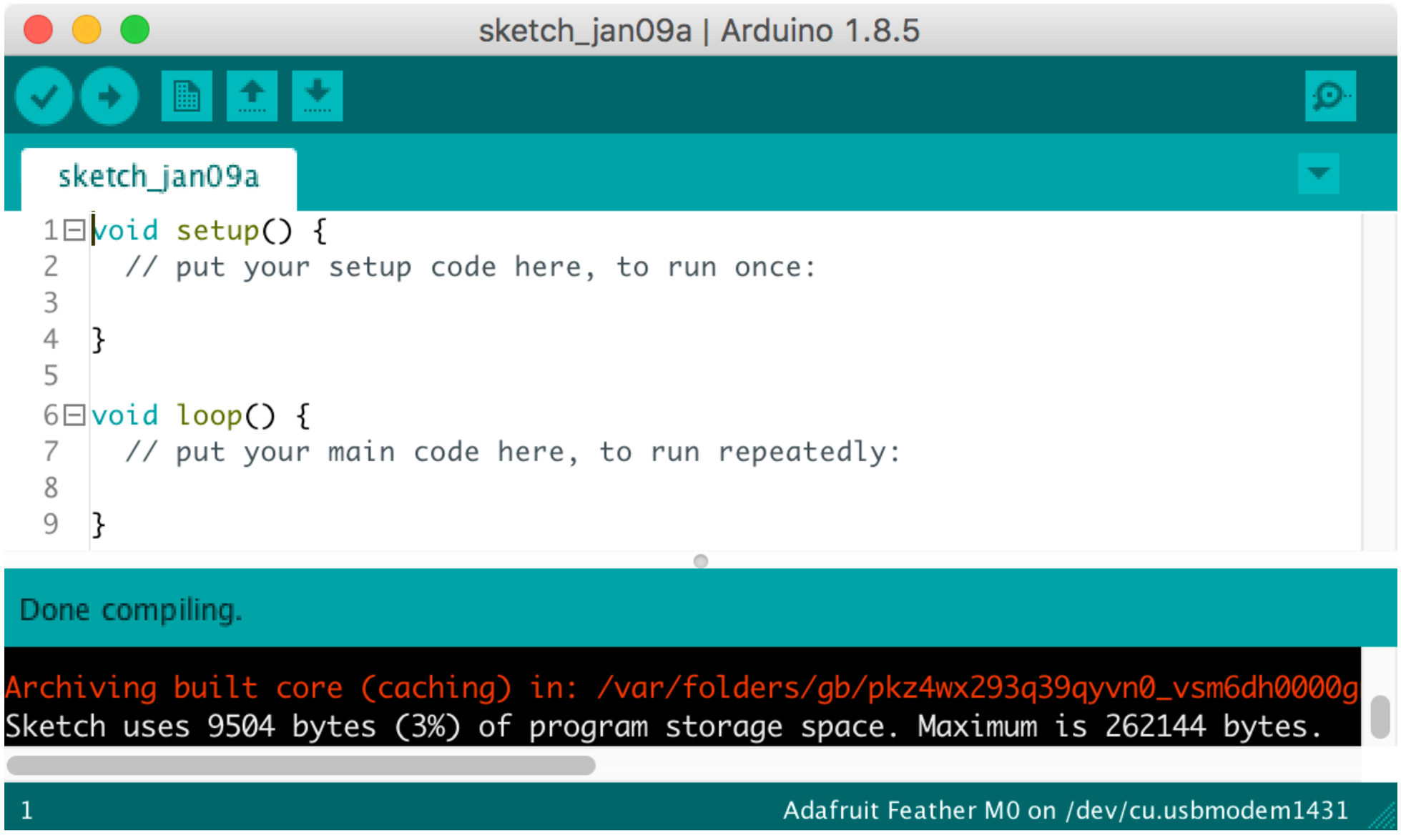


BARE STANDARD CODE NEEDED



https://www.arduino.cc/en/Tutorial/BareMinimum

BARE STANDARD CODE NEEDED



BARE MINIMUM CODE CALLED

BARE MINIMUM CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
    setup();



}
```

BARE MINIMUM CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
    setup();
    for (;;) {


    }
    return 0;
}
```

BARE MINIMUM CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
   setup();
   for (;;) {
      loop();

   }
   return 0;
}
```

BARE MINIMUM CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
   setup();
   for (;;) {
      loop();
      if (serialEventRun) serialEventRun();
   }
   return 0;
}
```

BARE MINIMUM CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
   init();
   initVariant();
#if defined(USBCON)
   USBDevice.attach();
#endif
   setup();
   for (;;) {
      loop();
      if (serialEventRun) serialEventRun();
   }
   return 0;
}
```

BARE STANDARD CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
   init();
   initVariant();
#if defined(USBCON)
   USBDevice.attach();
#endif
   setup();
   for (;;) {
      loop();
      if (serialEventRun) serialEventRun();
   }
   return 0;
}
```

https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/cores/arduino/main.cpp

BARE STANDARD CODE CALLED

```
// main.cpp - Main loop for Arduino sketches

#include <Arduino.h>

int main(void)
{
   init();
   initVariant();
#if defined(USBCON)
   USBDevice.attach();
#endif
   setup();
   for (;;) {
      loop();
      if (serialEventRun) serialEventRun();
   }
   return 0;
}
```

MULTIPLE LOOPS?

MULTIPLE LOOPS?

- «I have a problem. I want to make a car with a motor, front lights and rear lights. I want to run them at the same time but in different loops»

MULTIPLE LOOPS?

- «I have a problem. I want to make a car with a motor, front lights and rear lights. I want to run them at the same time but in different loops»
- «As the others have stated, no you can't have multiple loop functions»

MULTIPLE LOOPS?

- «I have a problem. I want to make a car with a motor, front lights and rear lights. I want to run them at the same time but in different loops»

- «As the others have stated, no you can't have multiple loop functions»

- «What you need to do is modify your approach so that each thing you are trying to do can be done sequentially without blocking (i.e.: remove the delay function usage)»

MULTIPLE LOOPS?

- «I have a problem. I want to make a car with a motor, front lights and rear lights. I want to run them at the same time but in different loops»

- «As the others have stated, no you can't have multiple loop functions»

- «What you need to do is modify your approach so that each thing you are trying to do can be done sequentially without blocking (i.e.: remove the delay function usage)»

- = Concurrency

https://arduino.stackexchange.com/questions/37684/can-i-make-multiple-void-loops-with-arduino-uno

MULTIPLE LOOPS?

- «I have a problem. I want to make a car with a motor, front lights and rear lights. I want to run them at the same time but in different loops»

- «As the others have stated, no you can't have multiple loop functions»

- «What you need to do is modify your approach so that each thing you are trying to do can be done sequentially without blocking (i.e.: remove the delay function usage)»

- = Concurrency

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops
- LED loops do individual blinking

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops
- LED loops do individual blinking
- No general mechanism for communication

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops
- LED loops do individual blinking
- No general mechanism for communication
- No scheme to wait for «resources». So it's busy poll or just a call to set some parameters into the actual loop. Atomicity? Protection?

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops
- LED loops do individual blinking
- No general mechanism for communication
- No scheme to wait for «resources». So it's busy poll or just a call to set some parameters into the actual loop. Atomicity? Protection?
  - I once a system like this, it took a person a year to fix the mess!
    This was between interrupts (more later) and «main» and it was written in assembly

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops
- LED loops do individual blinking
- No general mechanism for communication
- No scheme to wait for «resources». So it's busy poll or just a call to set some parameters into the actual loop. Atomicity? Protection?
  - I once a system like this, it took a person a year to fix the mess!
    This was between interrupts (more later) and «main» and it was written in assembly
- How to send results away?

BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- Motor loop sets off two LED loops
- LED loops do individual blinking
- No general mechanism for communication
- No scheme to wait for «resources». So it's busy poll or just a call to set some parameters into the actual loop. Atomicity? Protection?
  - I once a system like this, it took a person a year to fix the mess!
    This was between interrupts (more later) and «main» and it was written in assembly
- How to send results away?
- It's a start, it works here, but it's not a general problem to design a scheduler by

FINDING SCHEDULERS OR RUNTIME SYSTEMS

FINDING SCHEDULERS OR RUNTIME SYSTEMS

- In Library Manager, search for «scheduler», «task», «thread»

FINDING SCHEDULERS OR RUNTIME SYSTEMS

- In Library Manager, search for «scheduler», «task», «thread»
- Several matches, even one that uses C++11 and the std::thread class

FINDING SCHEDULERS OR RUNTIME SYSTEMS

- In Library Manager, search for «scheduler», «task», «thread»
- Several matches, even one that uses C++11 and the std::thread class
- However

FINDING SCHEDULERS OR RUNTIME SYSTEMS

- In Library Manager, search for «scheduler», «task», «thread»
- Several matches, even one that uses C++11 and the std::thread class
- However
  - As I see it, they are all «toy» examples of regular scheduling of threads with no communication mechanism between them

FINDING SCHEDULERS OR RUNTIME SYSTEMS

- In Library Manager, search for «scheduler», «task», «thread»
- Several matches, even one that uses C++11 and the std::thread class
- However
  - As I see it, they are all «toy» examples of regular scheduling of threads with no communication mechanism between them
  - Beware of «toy» schedulers!

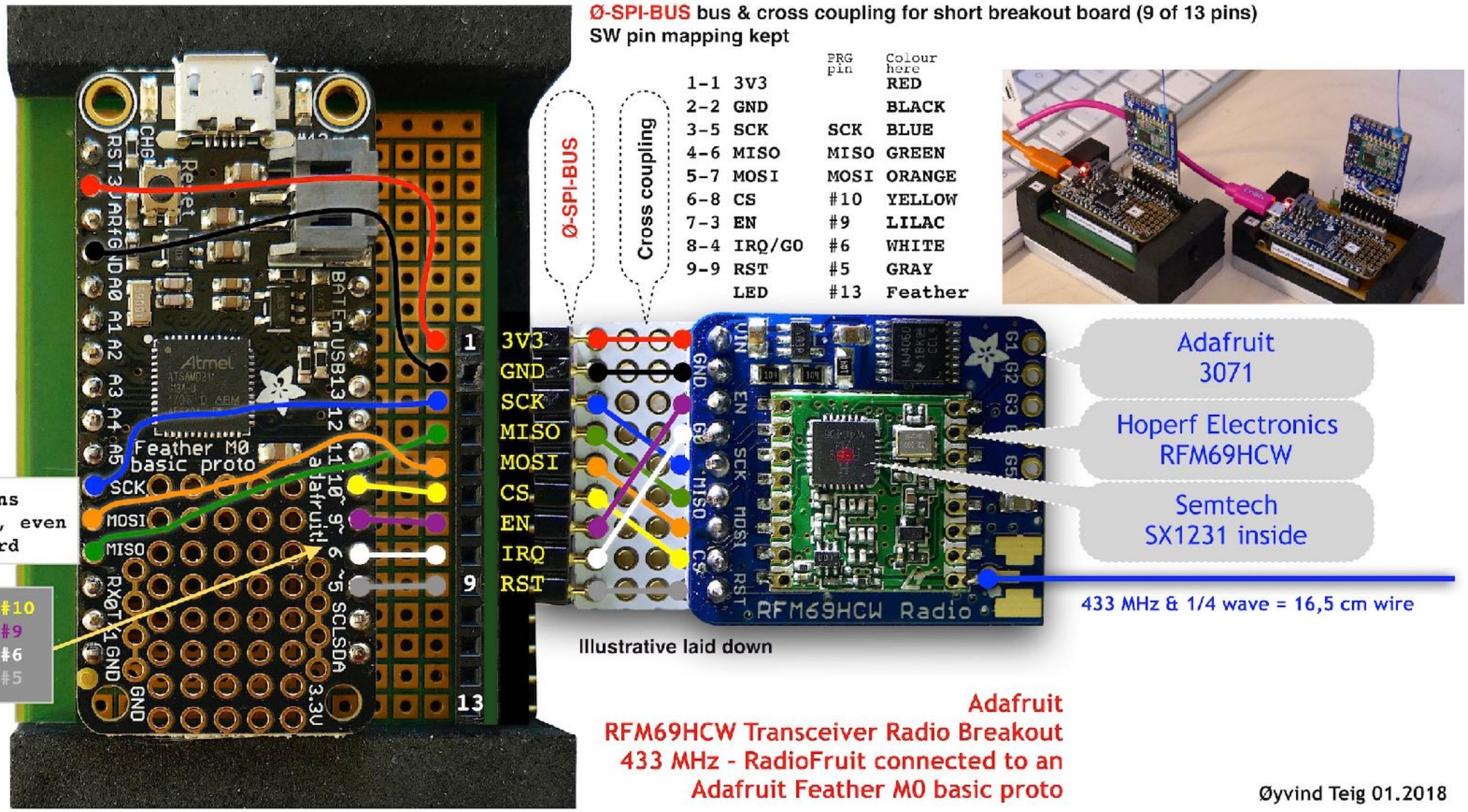
FINDING SCHEDULERS OR RUNTIME SYSTEMS

- In Library Manager, search for «scheduler», «task», «thread»
- Several matches, even one that uses C++11 and the std::thread class
- However
  - As I see it, they are all «toy» examples of regular scheduling of threads with no communication mechanism between them
  - Beware of «toy» schedulers!
- But Arduino is not a toy as such!

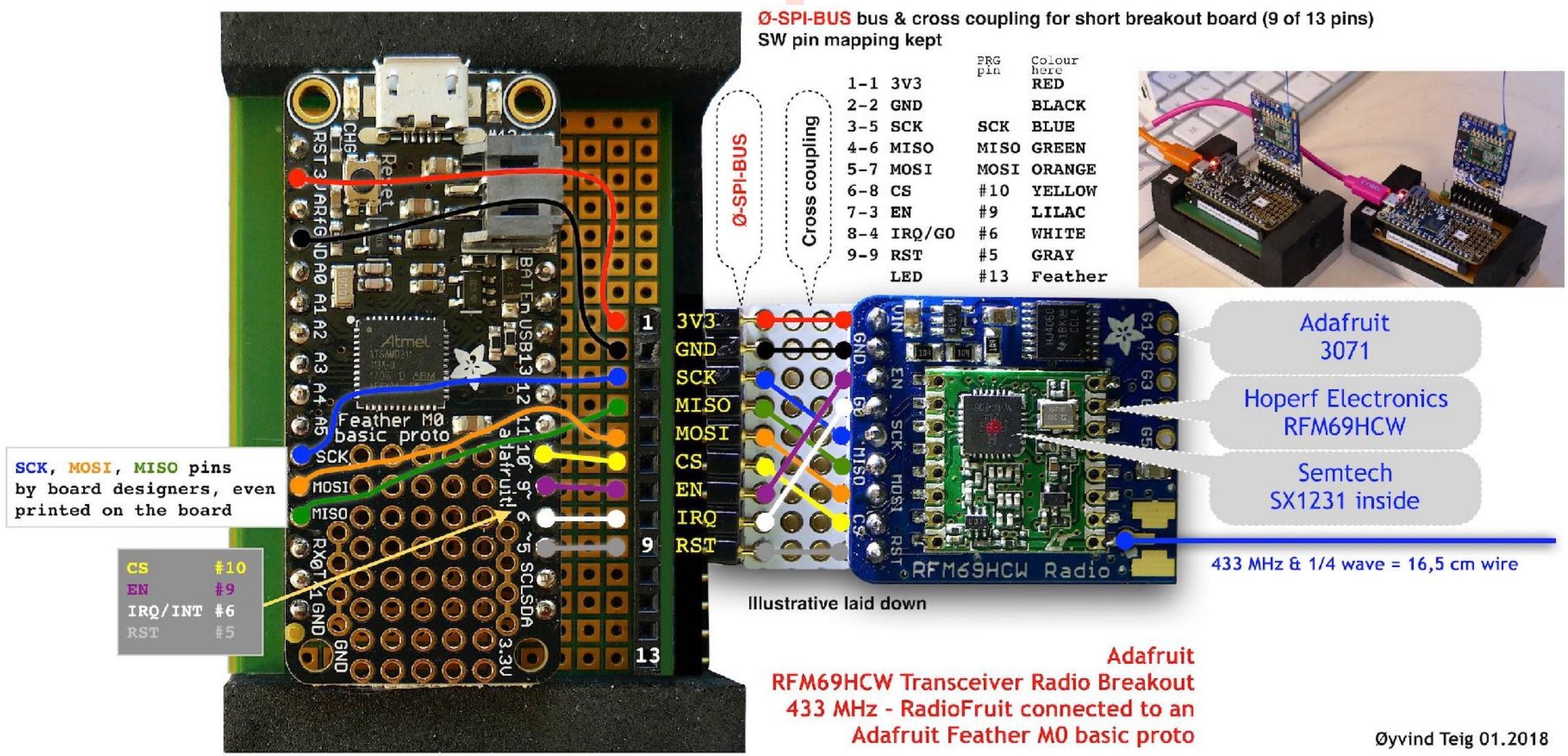
From a blog note

«void loop» ON MY DESK

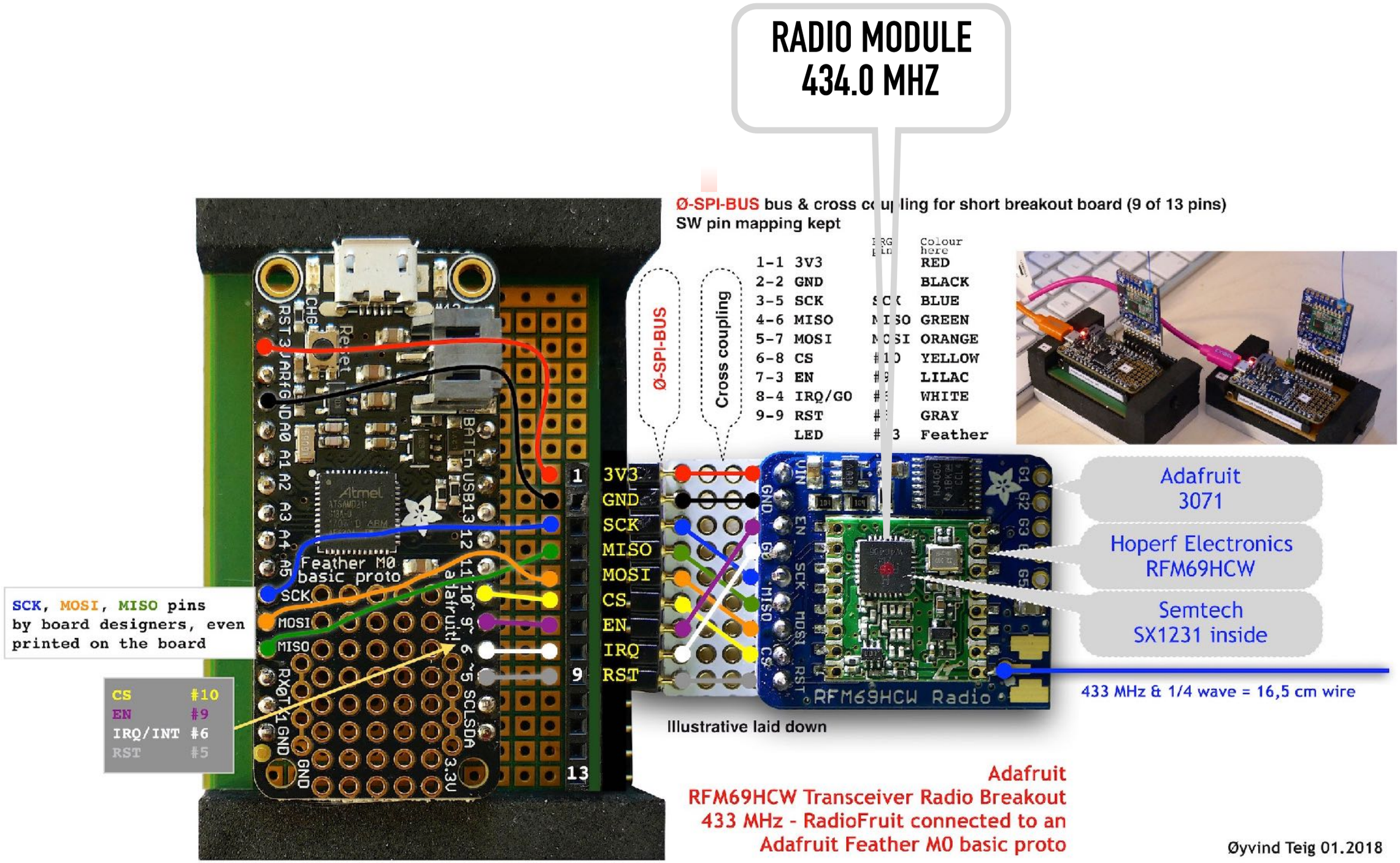
ARDUINO



ARDUINO «void loop» ON MY DESK



ARDUINO «void loop» ON MY DESK

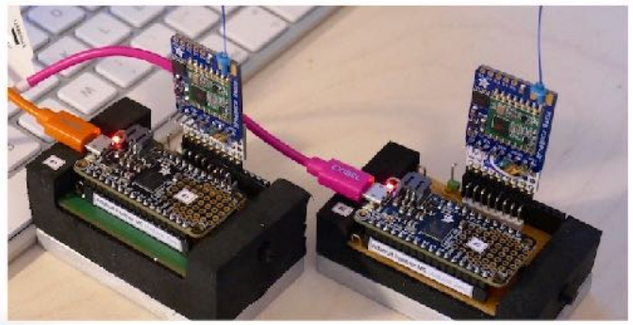


ARDUINO «void loop» ON MY DESK

RADIO MODULE 434.0 MHZ

Ø-SPI-BUS bus & cross coupling for short breakout board (9 of 13 pins)
SW pin mapping kept

		RG pin	Colour here
1-1	3V3		RED
2-2	GND		BLACK
3-5	SCK	SCK	BLUE
4-6	MISO	MISO	GREEN
5-7	MOSI	MOSI	ORANGE
6-8	CS	#10	YELLOW
7-3	EN	#9	LILAC
8-4	IRQ/GO	#6	WHITE
9-9	RST	#5	GRAY
	LED	#13	Feather



Ø-SPI-BUS

Cross coupling

3V3
GND
SCK
MISO
MOSI
CS
EN
IRQ
RST

Adafruit
3071

Hoperf Electronics
RFM69HCW

Semtech
SX1231 inside

433 MHz & 1/4 wave = 16,5 cm wire

Illustrative laid down

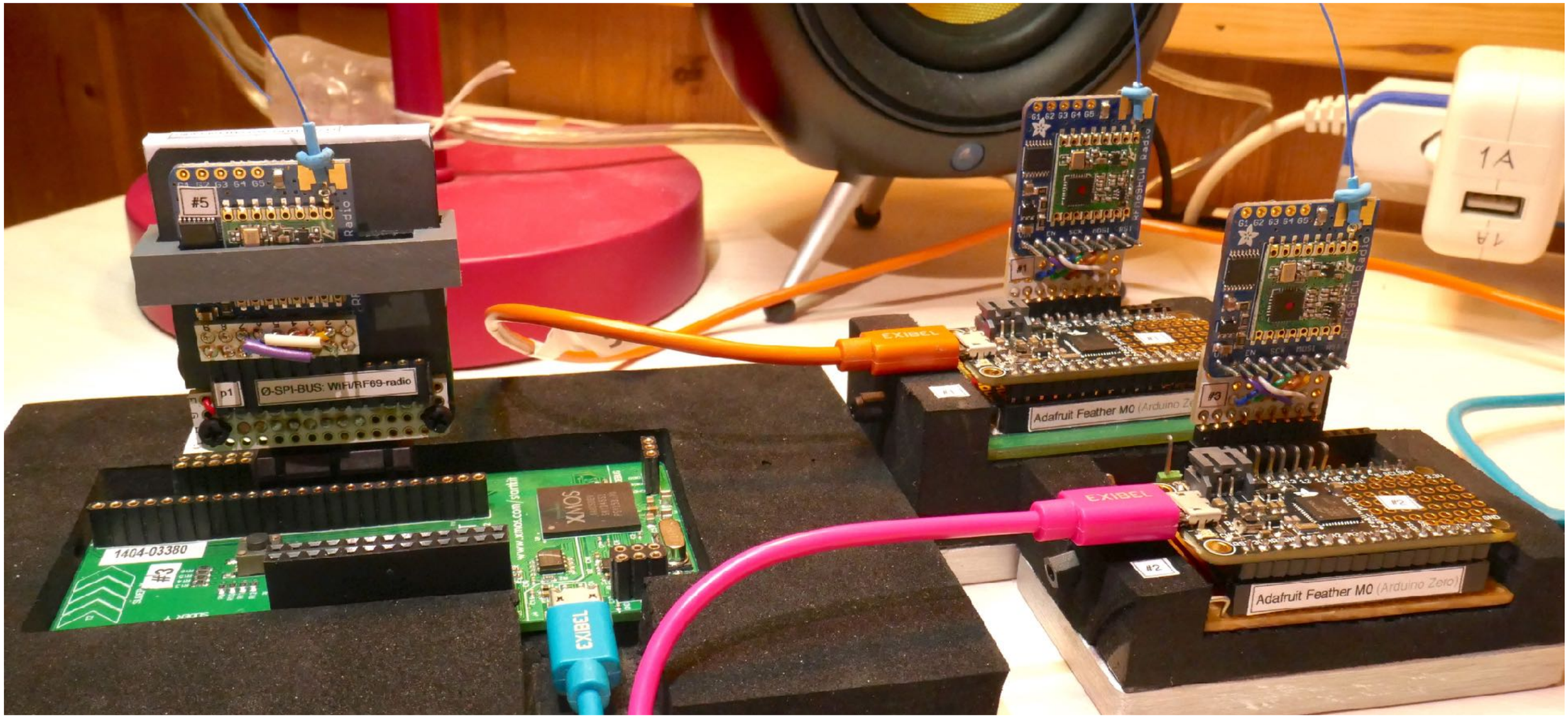
SCK, MOSI, MISO pins
by board designers, even
printed on the board

CS	#10
EN	#9
IRQ/INT	#6
RST	#5

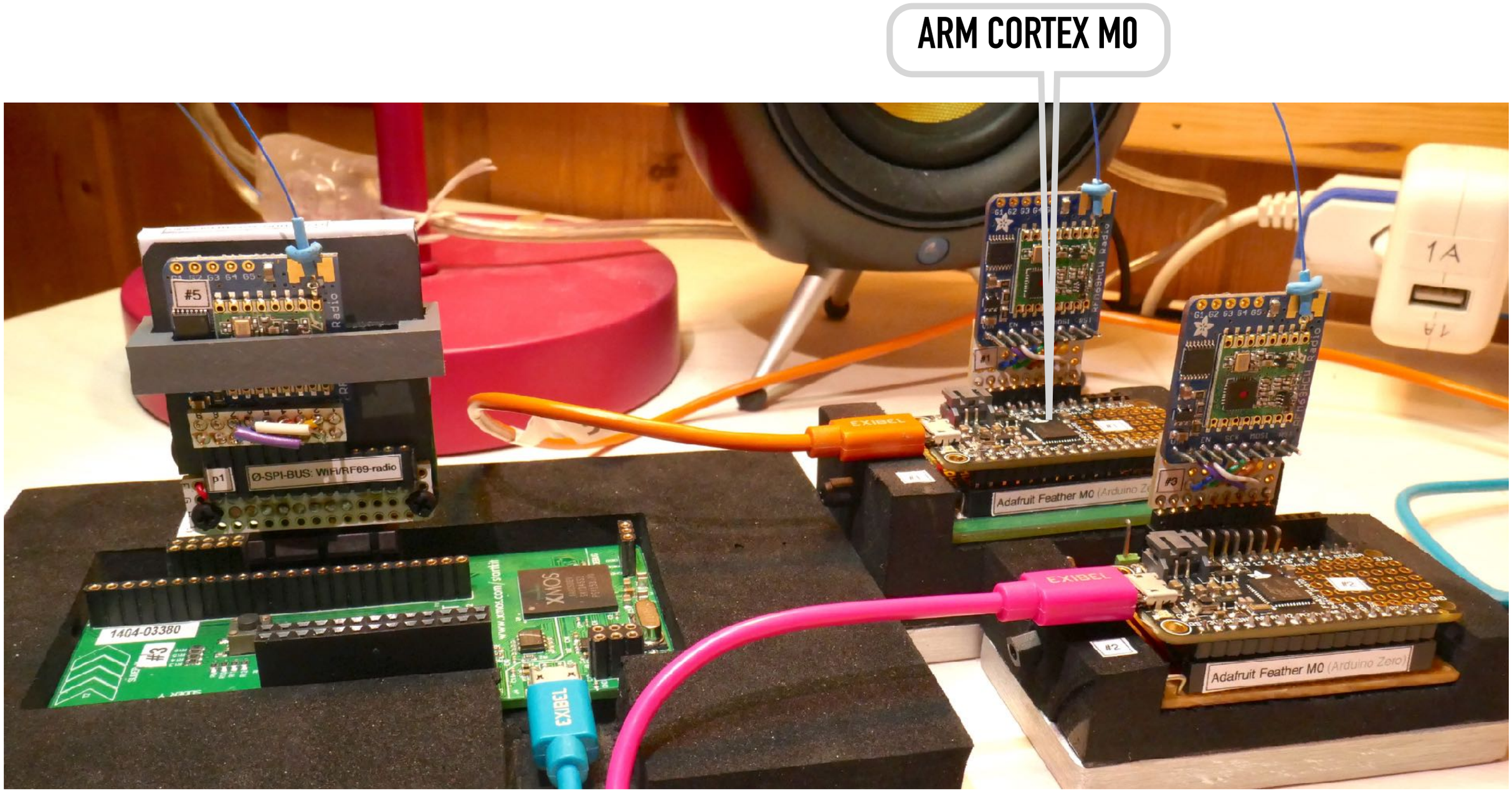
Adafruit
RFM69HCW Transceiver Radio Breakout
433 MHz - RadioFruit connected to an
Adafruit Feather M0 basic proto

Øyvind Teig 01.2018

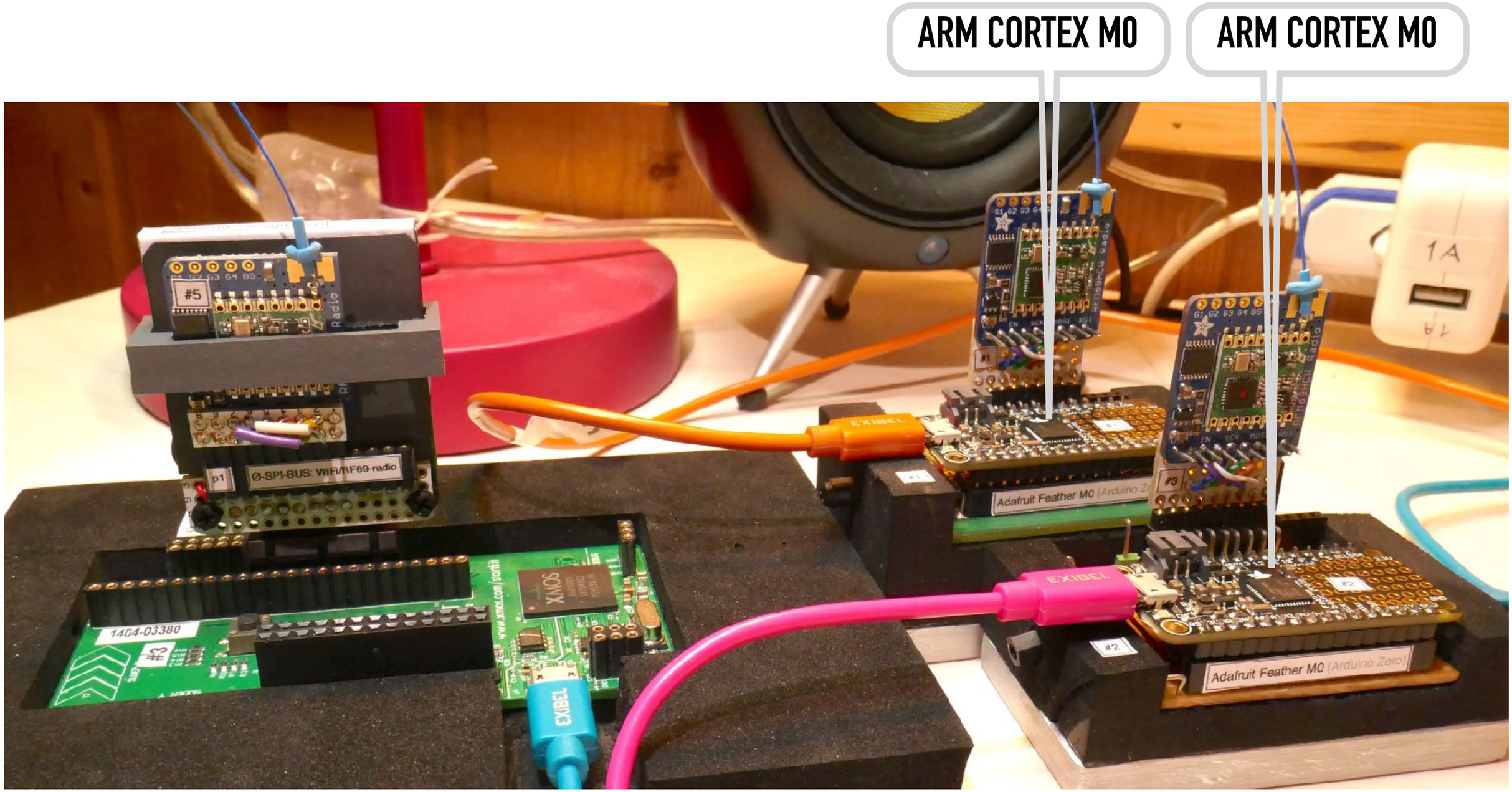
ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD



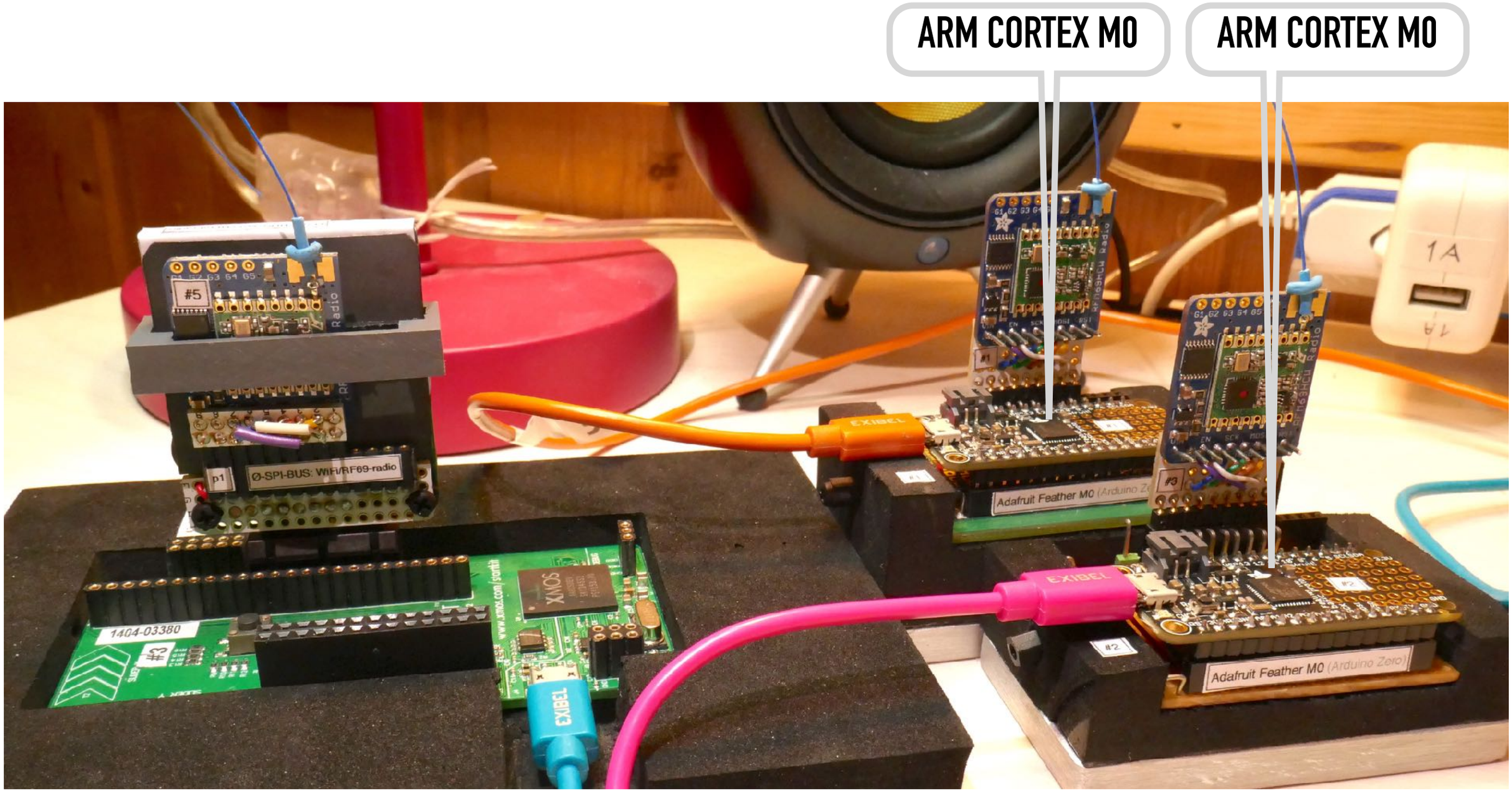
ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD



ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD

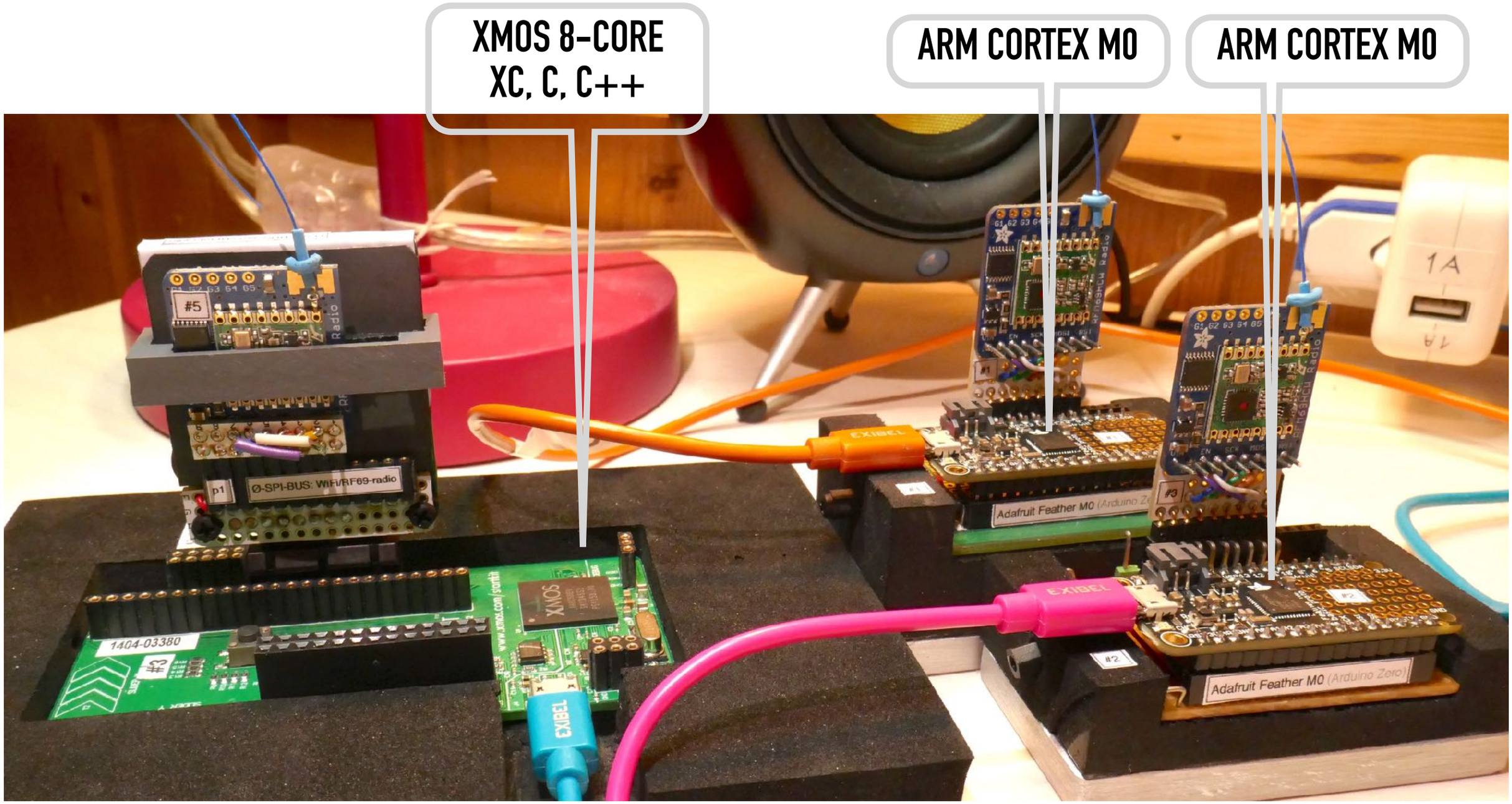


ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD



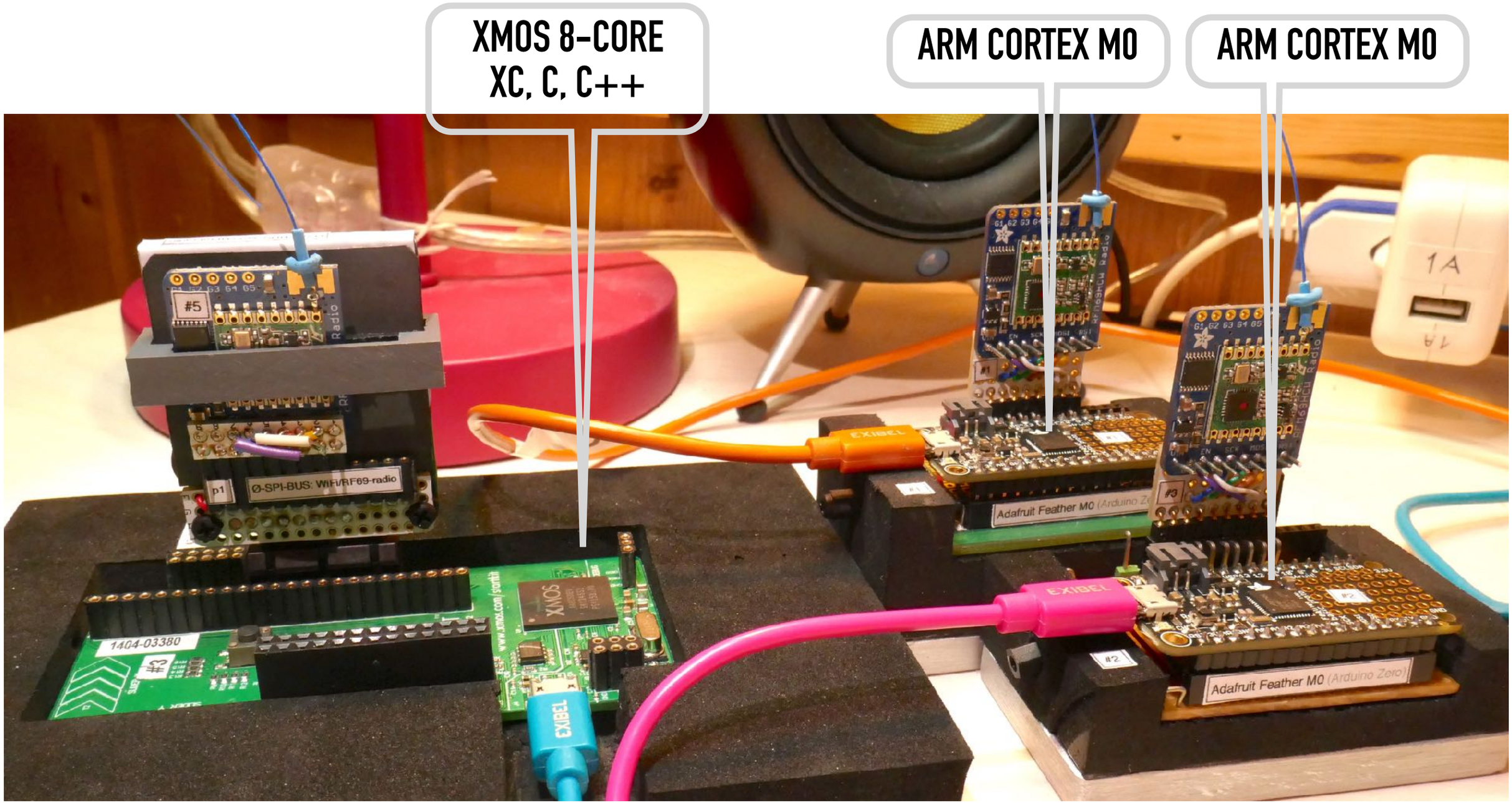
No concurrency

ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD



No concurrency

ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD



Concurrency

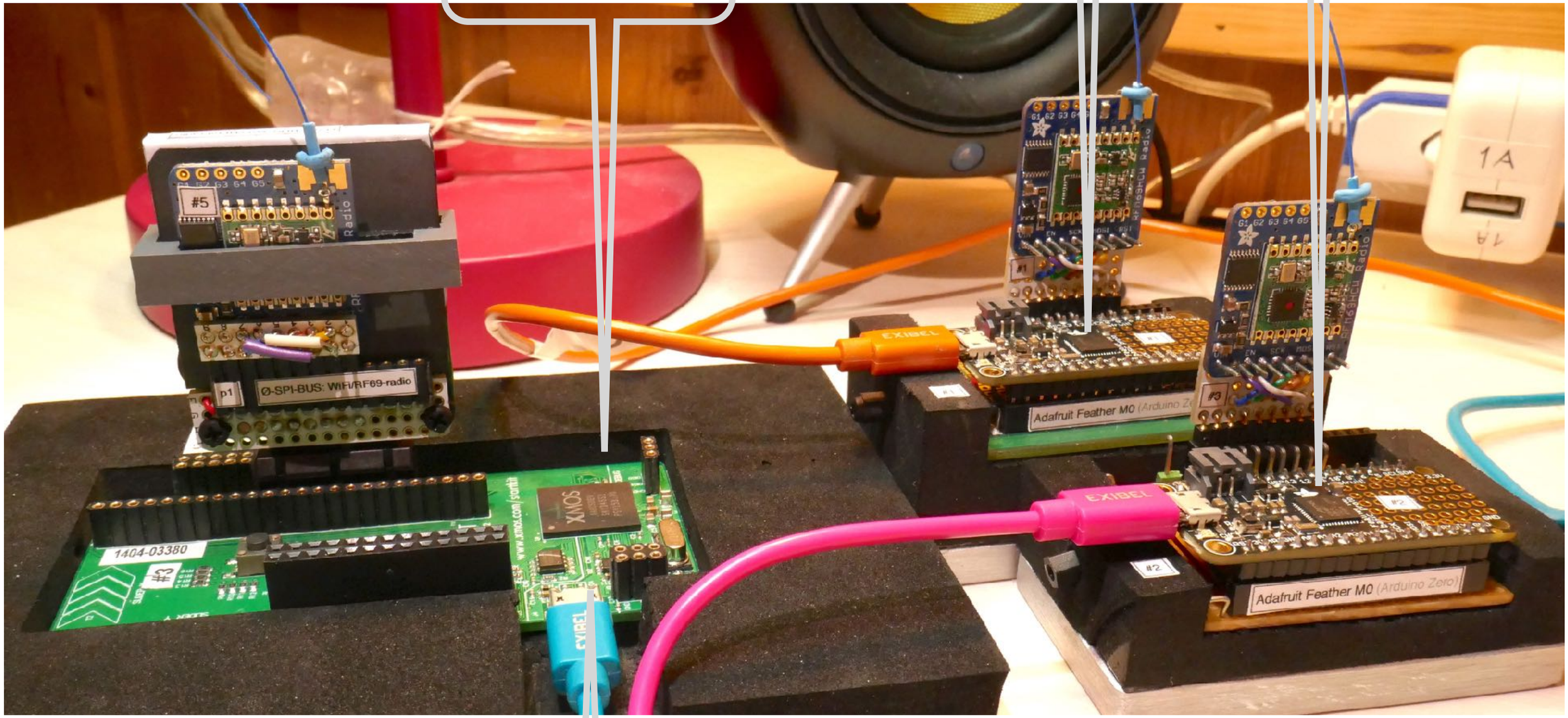
No concurrency

ARDUINO «void loop» ON MY DESK - PLUS AN XMOS BOARD

XMOS 8-CORE
XC, C, C++

ARM CORTEX M0

ARM CORTEX M0



Concurrency

No concurrency

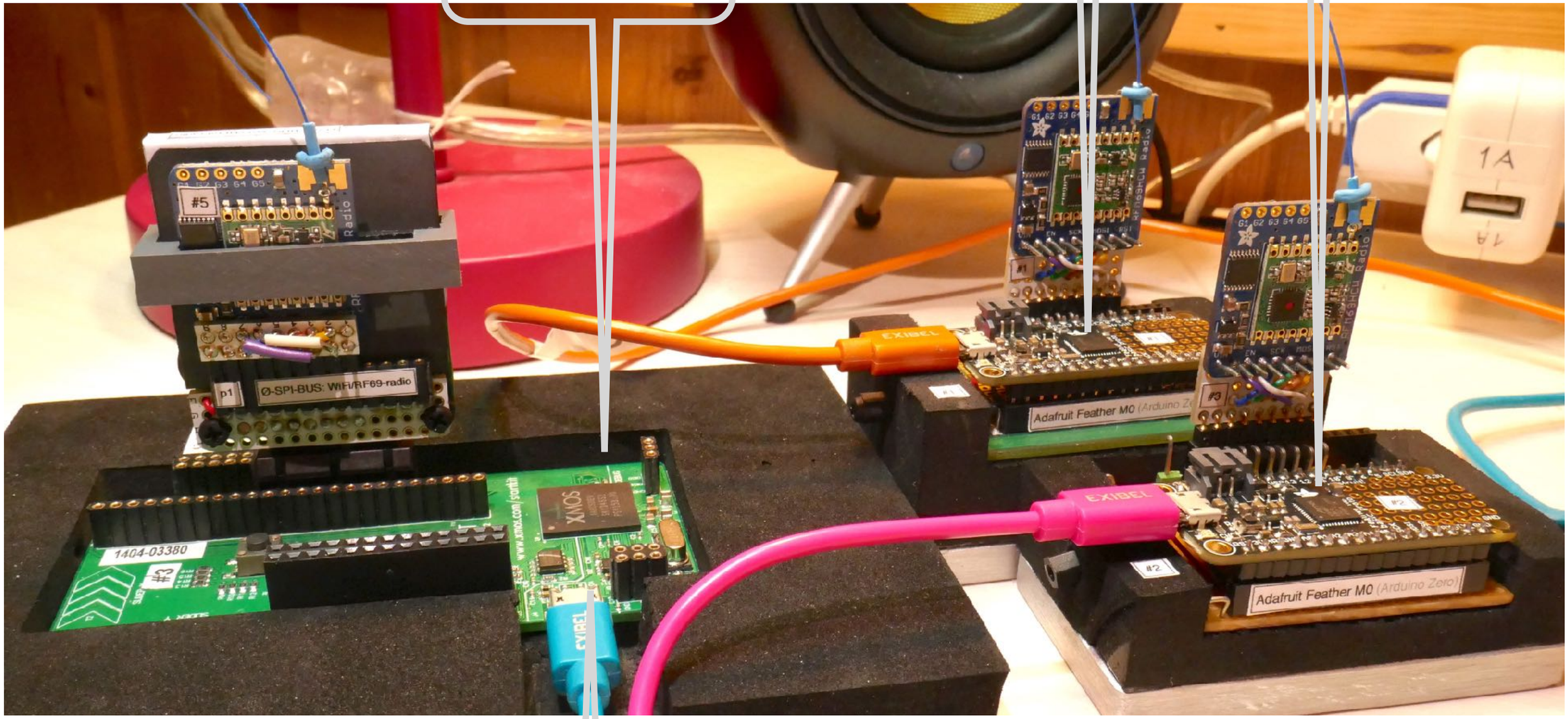
MORE LATER

ARDUINO «`void loop`» ON MY DESK - PLUS AN XMOS BOARD

XMOS 8-CORE
XC, C, C++

ARM CORTEX M0

ARM CORTEX M0



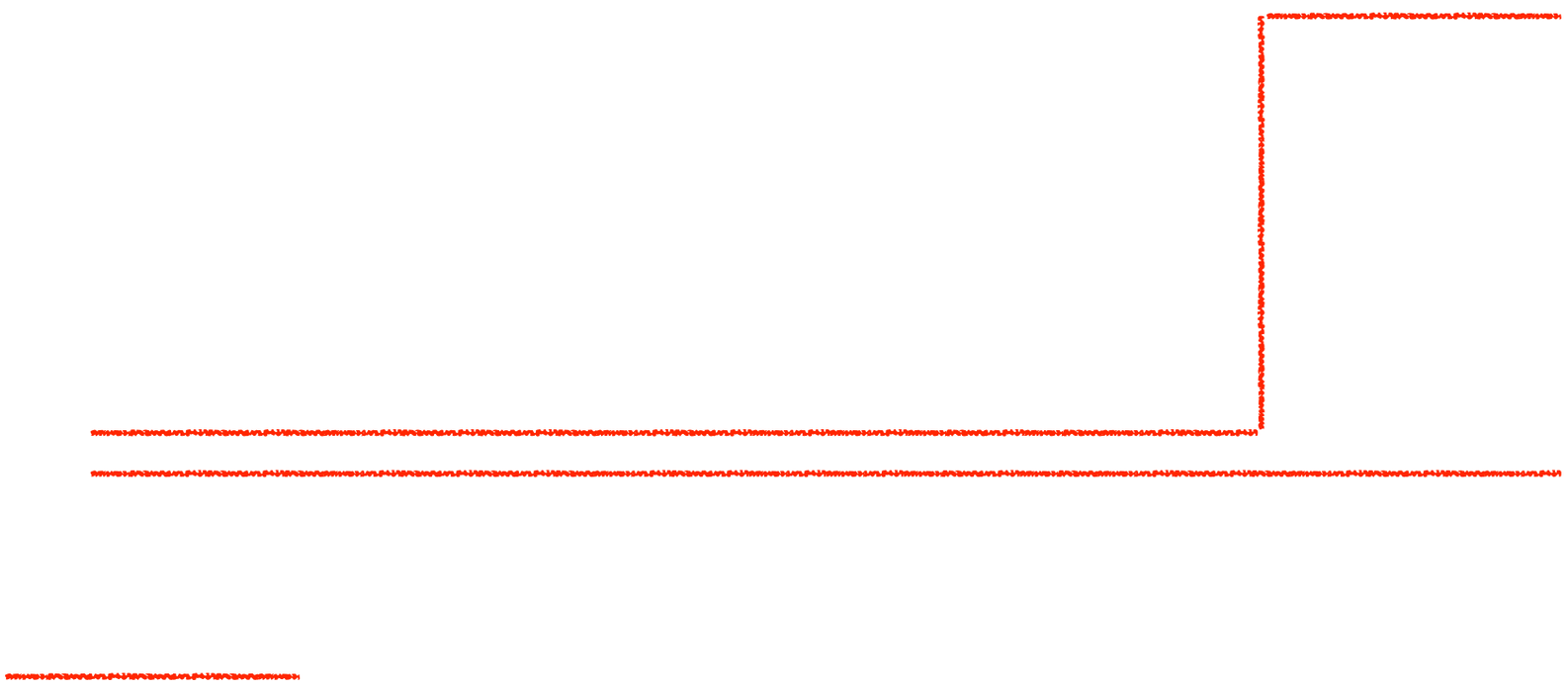
Concurrency

MORE LATER

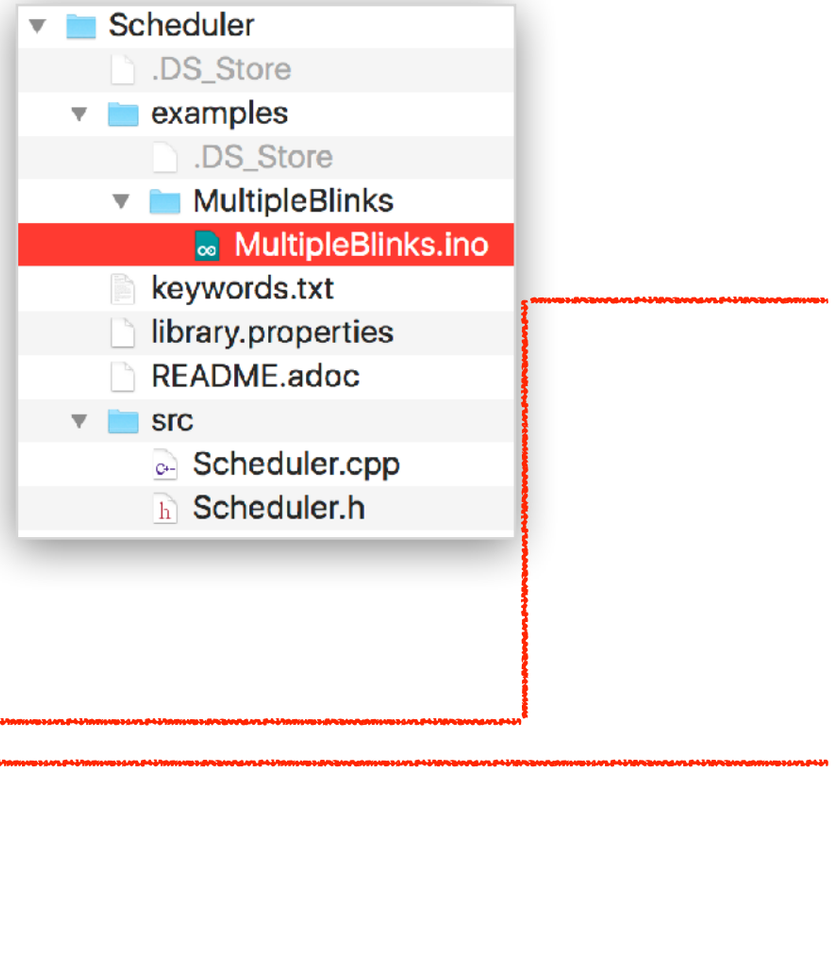
No concurrency

NEXT: `Scheduler`

ARDUINO: `Scheduler` AND THREE `loop()`

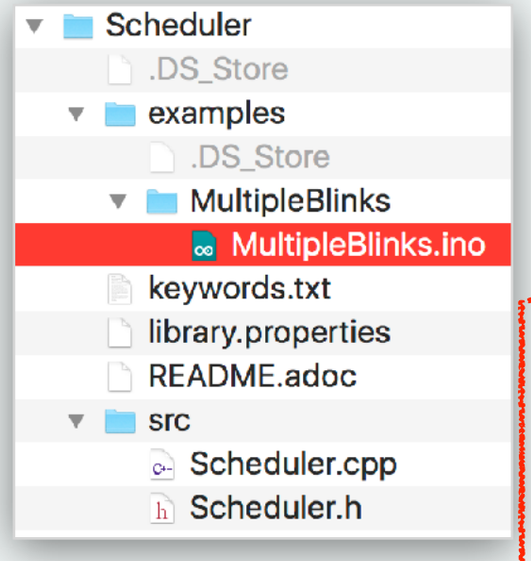


ARDUINO: `Scheduler` AND THREE `loop()`



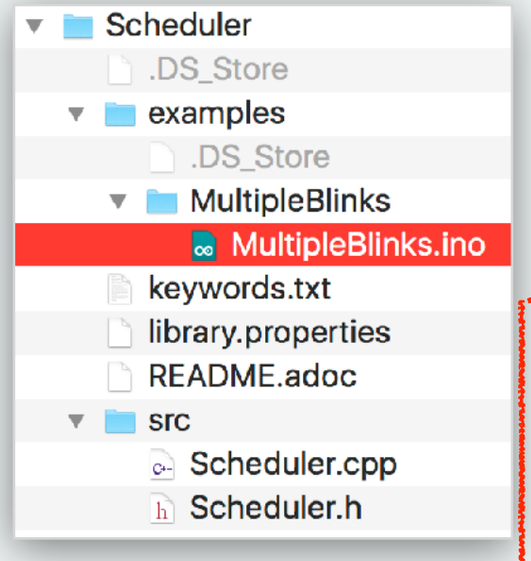
ARDUINO: `Scheduler` AND THREE `loop()`

```
// Include Scheduler since we want to manage multiple tasks.
#include <Scheduler.h>

int led1 = 13;
int led2 = 12;
int led3 = 11;

void setup() {
  Serial.begin(9600);

  // Setup the 3 pins as OUTPUT
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  // Add "loop2" and "loop3" to scheduling.
  // "loop" is always started by default.
  Scheduler.startLoop(loop2);
  Scheduler.startLoop(loop3);
}
```



ARDUINO: `Scheduler` AND THREE `loop()`

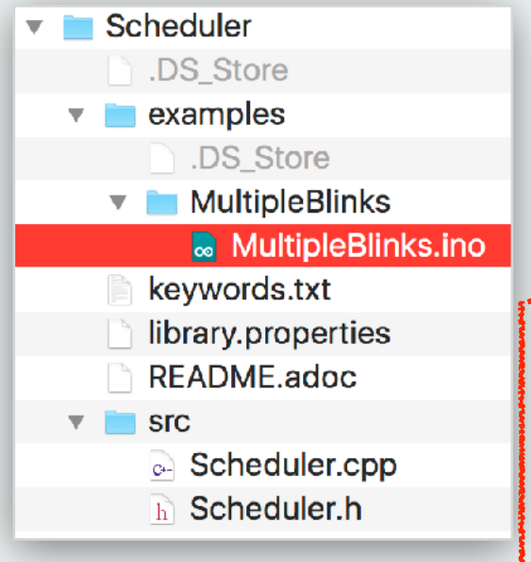
```
// Include Scheduler since we want to manage multiple tasks.
#include <Scheduler.h>

int led1 = 13;
int led2 = 12;
int led3 = 11;

void setup() {
  Serial.begin(9600);

  // Setup the 3 pins as OUTPUT
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  // Add "loop2" and "loop3" to scheduling.
  // "loop" is always started by default.
  Scheduler.startLoop(loop2);
  Scheduler.startLoop(loop3);
}
```




```
// Task no.1: blink LED with 1 second delay.
void loop() {
  digitalWrite(led1, HIGH);

  // IMPORTANT:
  // When multiple tasks are running 'delay' passes control
  // to other tasks while waiting and guarantees they get
  // executed.
  delay(1000);

  digitalWrite(led1, LOW);
  delay(1000);
}
```

ARDUINO: `Scheduler` AND THREE `loop()`

```
// Include Scheduler since we want to manage multiple tasks.
#include <Scheduler.h>

int led1 = 13;
int led2 = 12;
int led3 = 11;

void setup() {
  Serial.begin(9600);

  // Setup the 3 pins as OUTPUT
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  // Add "loop2" and "loop3" to scheduling.
  // "loop" is always started by default.
  Scheduler.startLoop(loop2);
  Scheduler.startLoop(loop3);
}
```

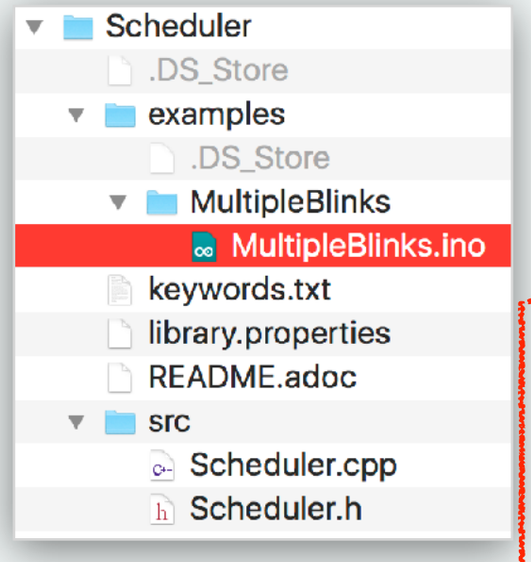



```
// Task no.2: blink LED with 0.1 second delay.
void loop2() {
  digitalWrite(led2, HIGH);
  delay(100);
  digitalWrite(led2, LOW);
  delay(100);
}
```

```
// Task no.1: blink LED with 1 second delay.
void loop() {
  digitalWrite(led1, HIGH);

  // IMPORTANT:
  // When multiple tasks are running 'delay' passes control
  // to other tasks while waiting and guarantees they get
  // executed.
  delay(1000);

  digitalWrite(led1, LOW);
  delay(1000);
}
```

ARDUINO: `Scheduler` AND THREE `loop()`

```
// Include Scheduler since we want to manage multiple tasks.
#include <Scheduler.h>

int led1 = 13;
int led2 = 12;
int led3 = 11;

void setup() {
  Serial.begin(9600);

  // Setup the 3 pins as OUTPUT
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  // Add "loop2" and "loop3" to scheduling.
  // "loop" is always started by default.
  Scheduler.startLoop(loop2);
  Scheduler.startLoop(loop3);
}
```




```
// Task no.1: blink LED with 1 second delay.
void loop() {
  digitalWrite(led1, HIGH);

  // IMPORTANT:
  // When multiple tasks are running 'delay' passes control
  // to other tasks while waiting and guarantees they get
  // executed.
  delay(1000);

  digitalWrite(led1, LOW);
  delay(1000);
}
```

```
// Task no.2: blink LED with 0.1 second delay.
void loop2() {
  digitalWrite(led2, HIGH);
  delay(100);
  digitalWrite(led2, LOW);
  delay(100);
}
```

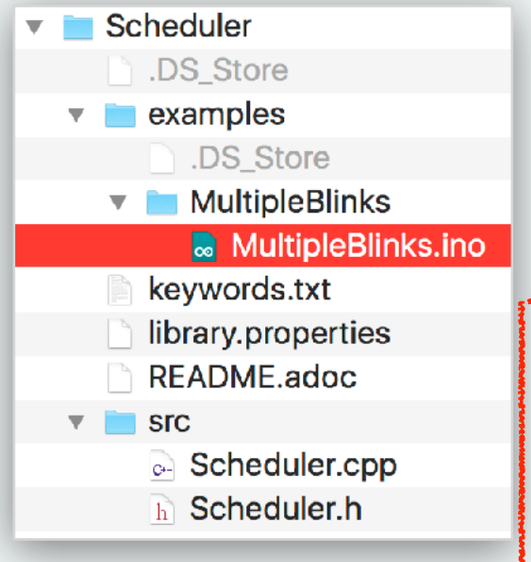
```
// Task no.3: accept commands from Serial port
// '0' turns off LED
// '1' turns on LED
void loop3() {
  if (Serial.available()) {
    char c = Serial.read();
    if (c=='0') {
      digitalWrite(led3, LOW);
      Serial.println("Led turned off!");
    }
    if (c=='1') {
      digitalWrite(led3, HIGH);
      Serial.println("Led turned on!");
    }
  }

  // IMPORTANT:
  // We must call 'yield' at a regular basis to pass
  // control to other tasks.
  yield();
}
```

ARDUINO: `Scheduler` AND THREE `loop()`

https://www.arduino.cc/en/Tutorial/MultipleBlinks

https://www.arduino.cc/en/Reference/Scheduler

```
// Include Scheduler since we want to manage multiple tasks.
#include <Scheduler.h>

int led1 = 13;
int led2 = 12;
int led3 = 11;

void setup() {
  Serial.begin(9600);

  // Setup the 3 pins as OUTPUT
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  // Add "loop2" and "loop3" to scheduling.
  // "loop" is always started by default.
  Scheduler.startLoop(loop2);
  Scheduler.startLoop(loop3);
}
```



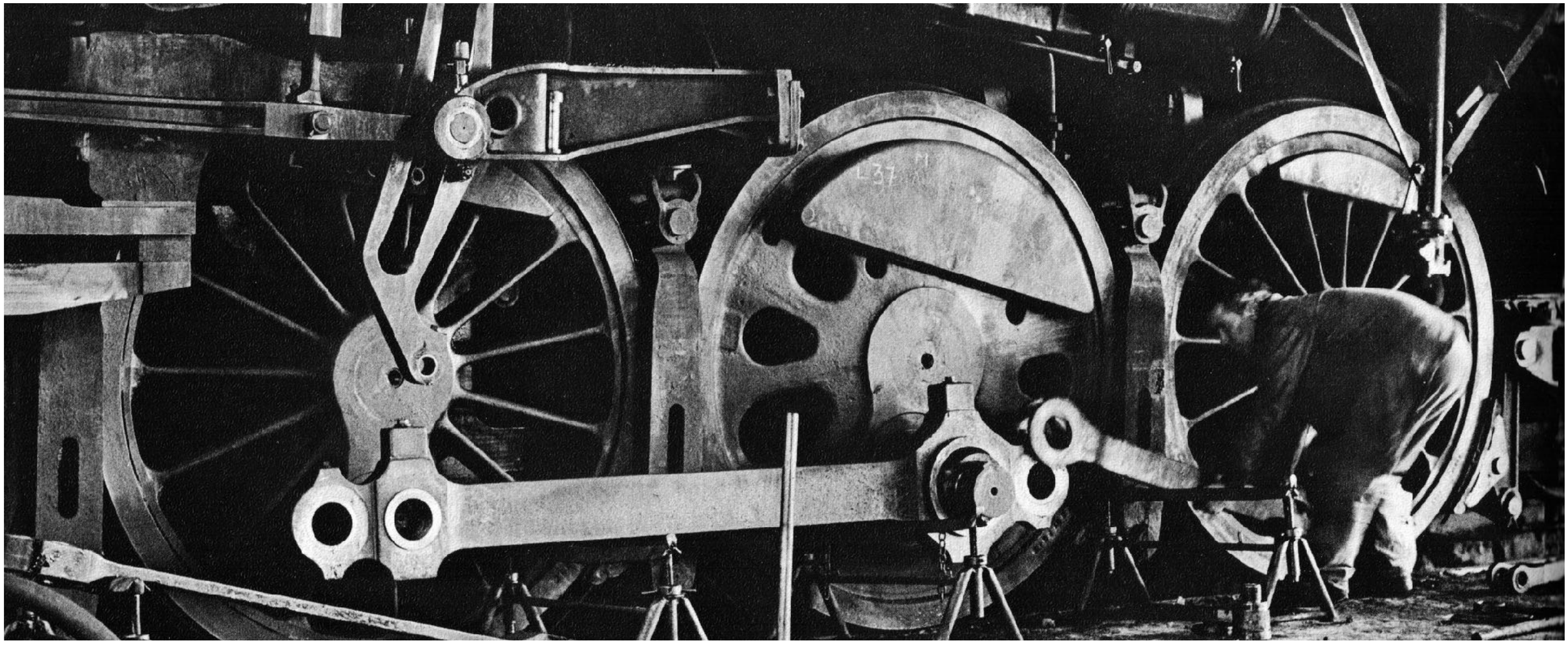

```
// Task no.1: blink LED with 1 second delay.
void loop() {
  digitalWrite(led1, HIGH);

  // IMPORTANT:
  // When multiple tasks are running 'delay' passes control
  // to other tasks while waiting and guarantees they get
  // executed.
  delay(1000);

  digitalWrite(led1, LOW);
  delay(1000);
}
```

```
// Task no.2: blink LED with 0.1 second delay.
void loop2() {
  digitalWrite(led2, HIGH);
  delay(100);
  digitalWrite(led2, LOW);
  delay(100);
}
```

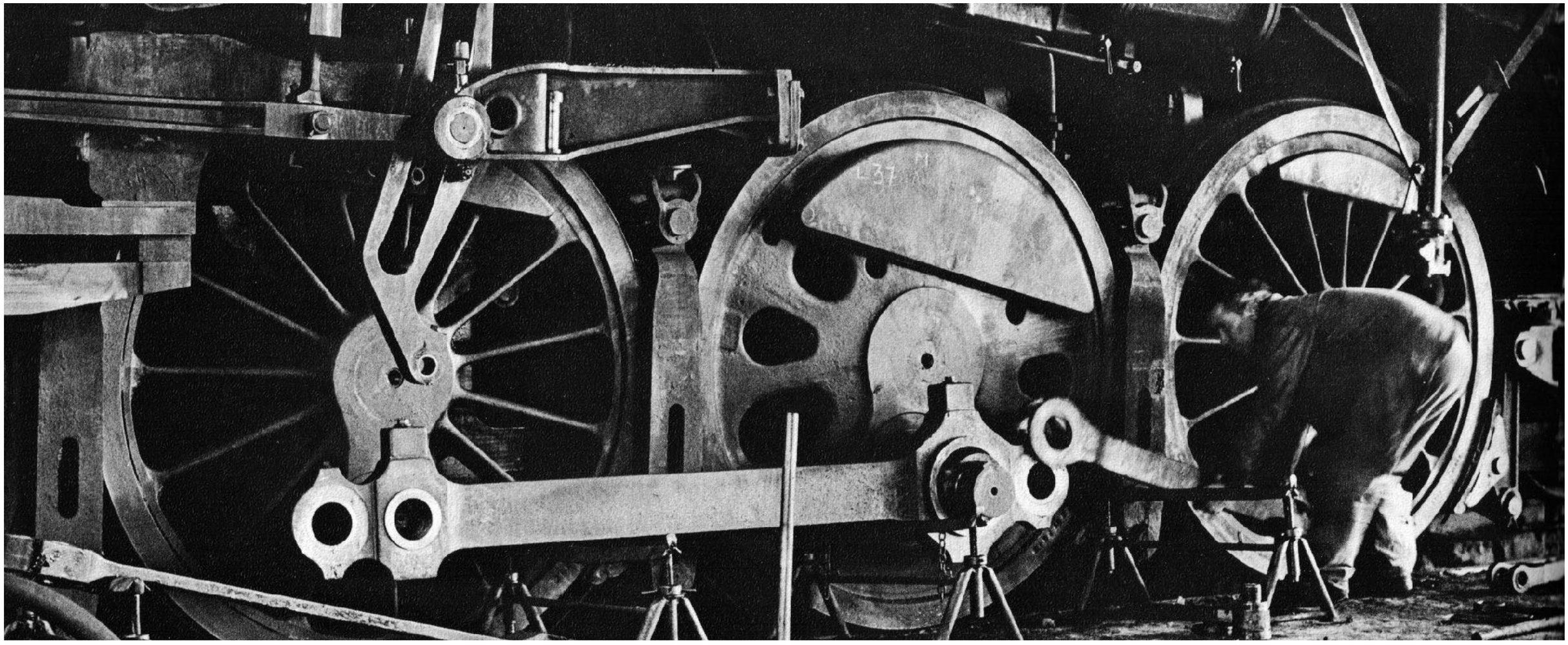
```
// Task no.3: accept commands from Serial port
// '0' turns off LED
// '1' turns on LED
void loop3() {
  if (Serial.available()) {
    char c = Serial.read();
    if (c=='0') {
      digitalWrite(led3, LOW);
      Serial.println("Led turned off!");
    }
    if (c=='1') {
      digitalWrite(led3, HIGH);
      Serial.println("Led turned on!");
    }
  }

  // IMPORTANT:
  // We must call 'yield' at a regular basis to pass
  // control to other tasks.
  yield();
}
```

THE WHEELS MAY TURN, BUT IT MAY SOON END UP LIKE THIS

THE WHEELS MAY TURN, BUT IT MAY SOON END UP LIKE THIS



THE WHEELS MAY TURN, BUT IT MAY SOON END UP LIKE THIS



In **All Trains to Stop** by **Hans Steeneken** (1979)

WHAT ABOUT INTERRUPTS?

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

---

- But could one thread («Driver») initialise an interrupt HW over an init «**channel**», and then sit idly waiting on a return **channel** for the result?

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

---

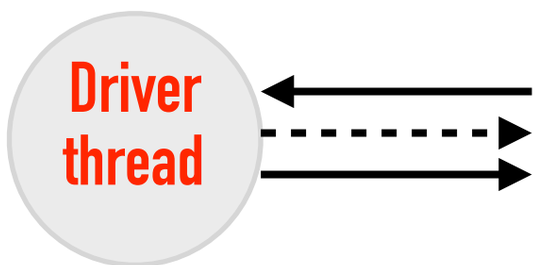
- But could one thread («Driver») initialise an interrupt HW over an init «**channel**», and then sit idly waiting on a return **channel** for the result?
- Provided this thread only did this job «now» and other threads could do their jobs independently?

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

---

- But could one thread («Driver») initialise an interrupt HW over an init «**channel**», and then sit idly waiting on a return **channel** for the result?
- Provided this thread only did this job «now» and other threads could do their jobs independently?



WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

---

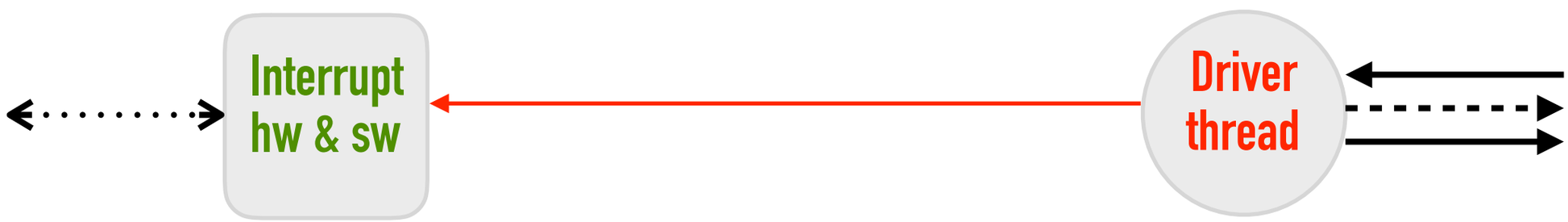
- But could one thread («Driver») initialise an interrupt HW over an init «**channel**», and then sit idly waiting on a return **channel** for the result?
- Provided this thread only did this job «now» and other threads could do their jobs independently?

WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

---

- But could one thread («Driver») initialise an interrupt HW over an init «**channel**», and then sit idly waiting on a return **channel** for the result?
- Provided this thread only did this job «now» and other threads could do their jobs independently?

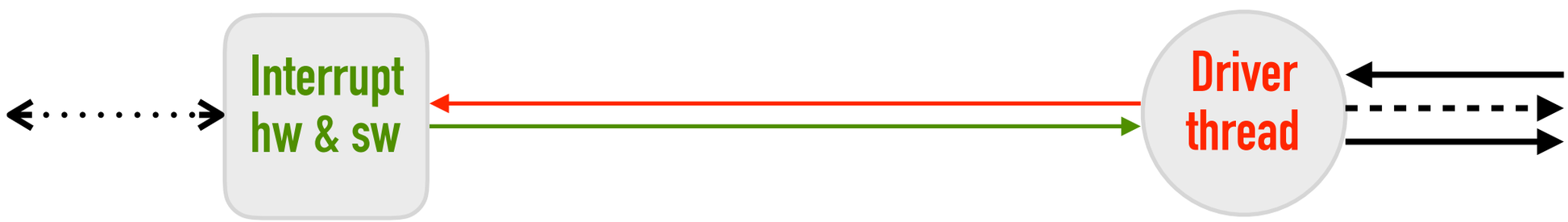


WHAT ABOUT INTERRUPTS?

- You get a lot of concurrency / real-time with interrupts
- After all, the interrupt controller and the HW units (like a USART or TIMER) that mostly deliver data to it, are separate silicon, not stealing (much) cycles from the processor
- Basically, this is all the concurrency that Arduino (AVR, ARM) can offer
- However, an «interrupt thread» («task», «process») (??) does not supply you with general «thread», «task», «process» terms

---

- But could one thread («Driver») initialise an interrupt HW over an init «**channel**», and then sit idly waiting on a return **channel** for the result?
- Provided this thread only did this job «now» and other threads could do their jobs independently?



WHAT ABOUT NOT INTERRUPTS?

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip
  - had the same concept, but also dedicated cores to handle the code (open-source hardware and Spin language) [2]

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip
  - had the same concept, but also dedicated cores to handle the code (open-source hardware and Spin language) [2]
- The XCore multi-core architecture

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip
  - had the same concept, but also dedicated cores to handle the code (open-source hardware and Spin language) [2]
- The XCore multi-core architecture
  - adds a more generalised I/O-pad architecture (edge, timer, etc.) handled in the XC language and intrinsic macros or functions. «Between standard processor and ASIC». I think their deterministic timing guarantee (by compiler and tool) may give full control of interrupt latency [3]

[1] https://en.wikipedia.org/wiki/Transputer

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip
  - had the same concept, but also dedicated cores to handle the code (open-source hardware and Spin language) [2]
- The XCore multi-core architecture
  - adds a more generalised I/O-pad architecture (edge, timer, etc.) handled in the XC language and intrinsic macros or functions. «Between standard processor and ASIC». I think their deterministic timing guarantee (by compiler and tool) may give full control of interrupt latency [3]

[1] https://en.wikipedia.org/wiki/Transputer
[2] https://en.wikipedia.org/wiki/Parallax_Propeller

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip
  - had the same concept, but also dedicated cores to handle the code (open-source hardware and Spin language) [2]
- The XCore multi-core architecture
  - adds a more generalised I/O-pad architecture (edge, timer, etc.) handled in the XC language and intrinsic macros or functions. «Between standard processor and ASIC». I think their deterministic timing guarantee (by compiler and tool) may give full control of interrupt latency [3]

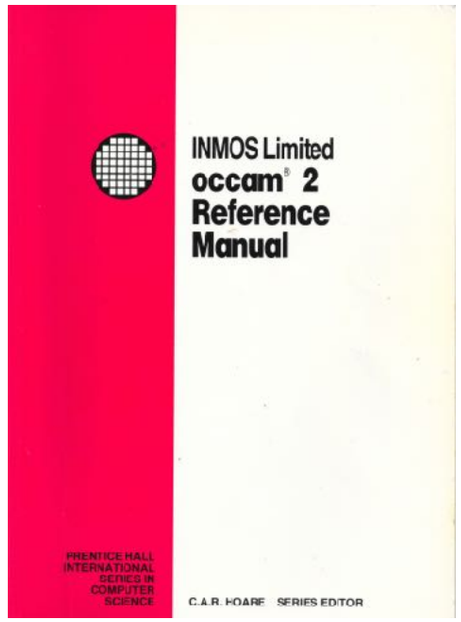
[1] https://en.wikipedia.org/wiki/Transputer
[2] https://en.wikipedia.org/wiki/Parallax_Propeller
[3] https://en.wikipedia.org/wiki/XCore_Architecture

WHAT ABOUT NOT INTERRUPTS?

- Three processors I have come across do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel uP)
  - there was one 'event' line, similar to a conventional processor's interrupt line. Treated as a channel (with no data) in occam, a process could 'input' from the event channel, and proceed only after the event line was asserted [1]
- The Parallax Propeller multi-core chip
  - had the same concept, but also dedicated cores to handle the code (open-source hardware and Spin language) [2]
- The XCore multi-core architecture
  - adds a more generalised I/O-pad architecture (edge, timer, etc.) handled in the XC language and intrinsic macros or functions. «Between standard processor and ASIC». I think their deterministic timing guarantee (by compiler and tool) may give full control of interrupt latency [3]

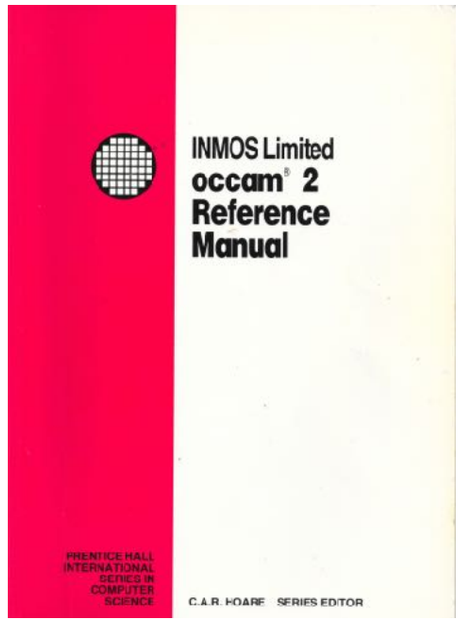
AT NTNU?

AT NTNU?


INMOS Limited
occam® 2
Reference
Manual
PRENTICE HALL
INTERNATIONAL
SERIES IN
COMPUTER
SCIENCE
C.A.R. HOARE SERIES EDITOR

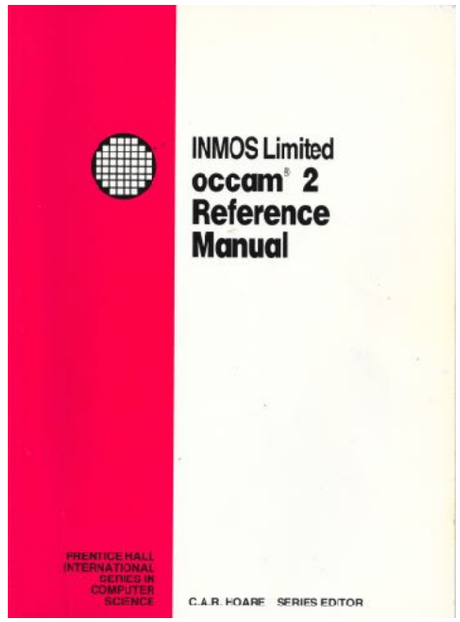
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)



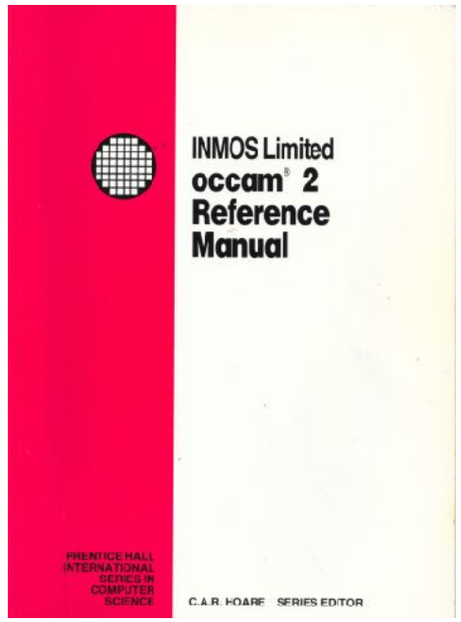
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language



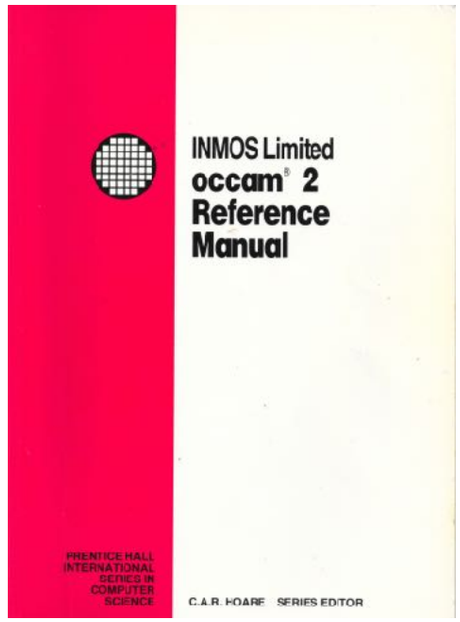
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]



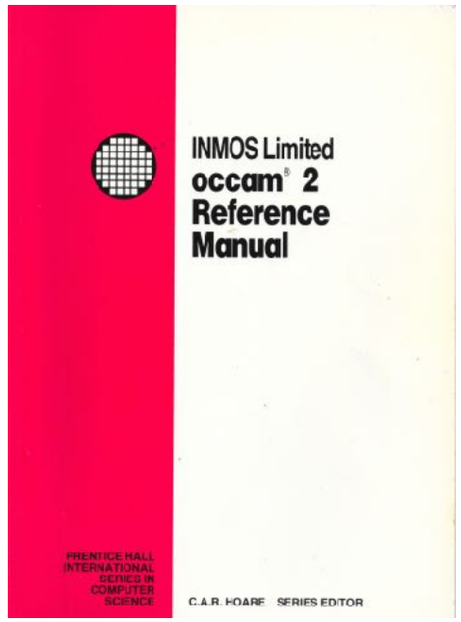
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous



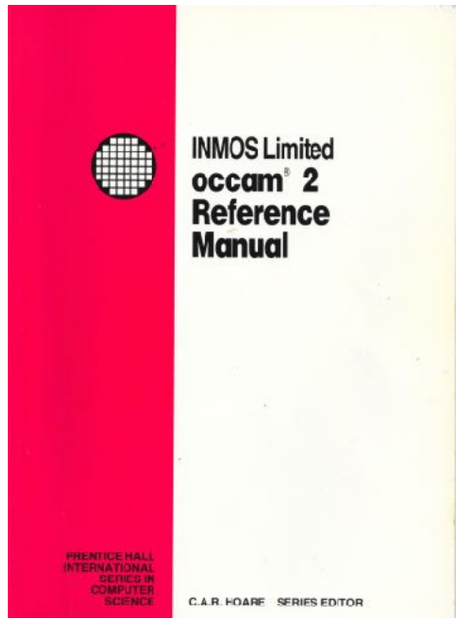
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]



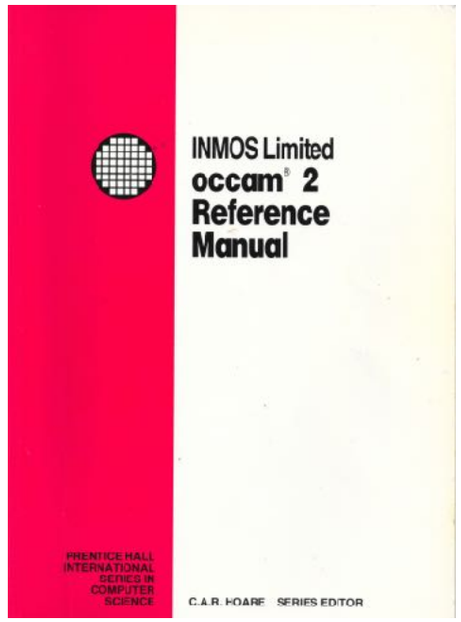
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels



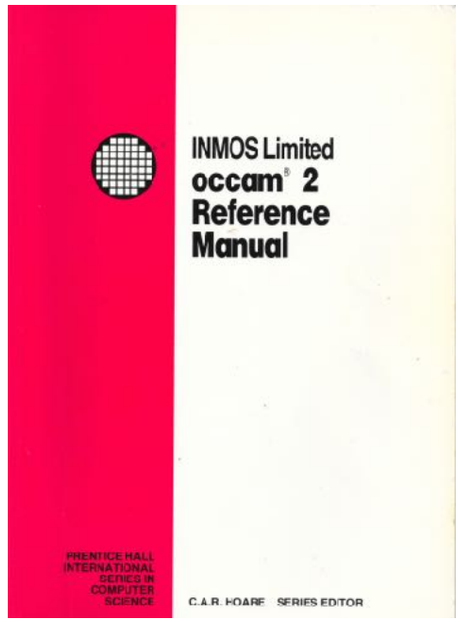
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide



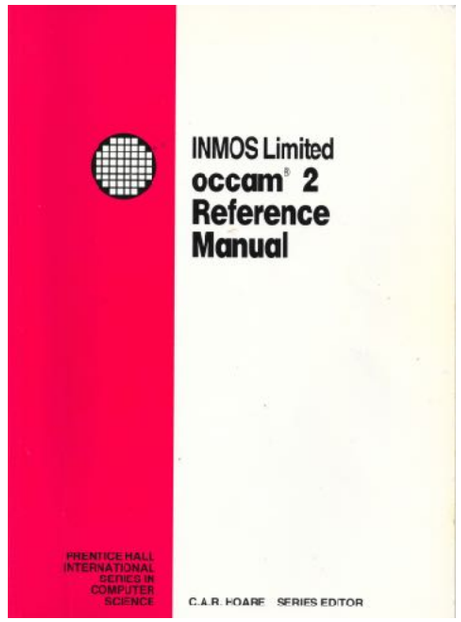
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide



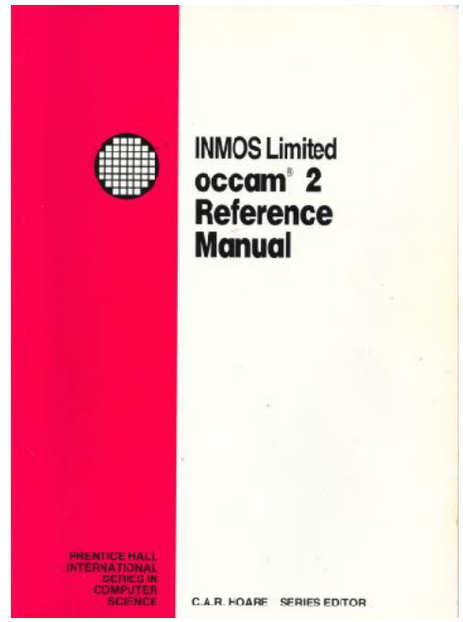
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]



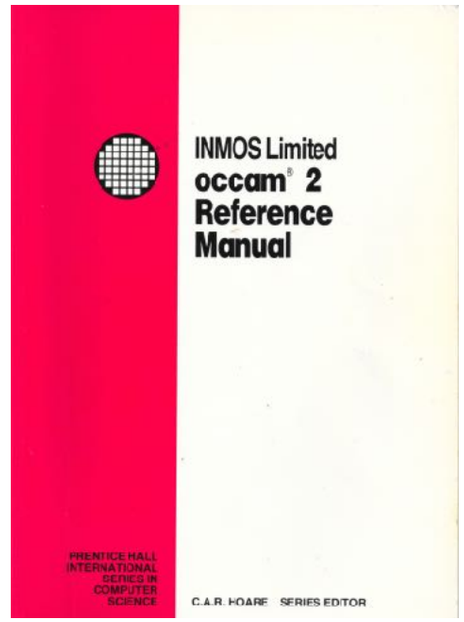
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]
- XC by XMOS on XMOS multi-core processors



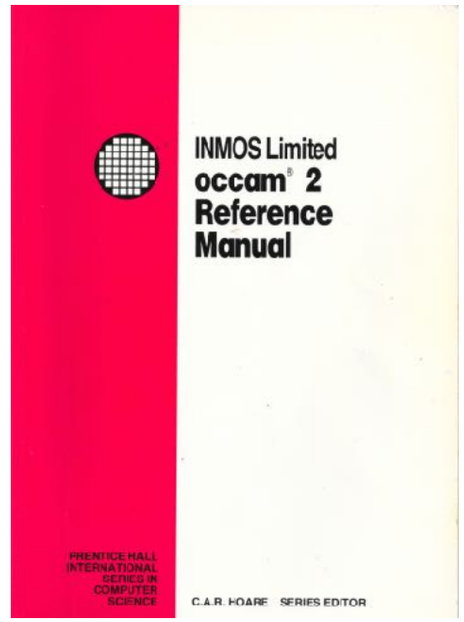
AT NTNU?

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]
- XC by XMOS on XMOS multi-core processors
  - I will show you some here. Has channels and interfaces



AT NTNU?

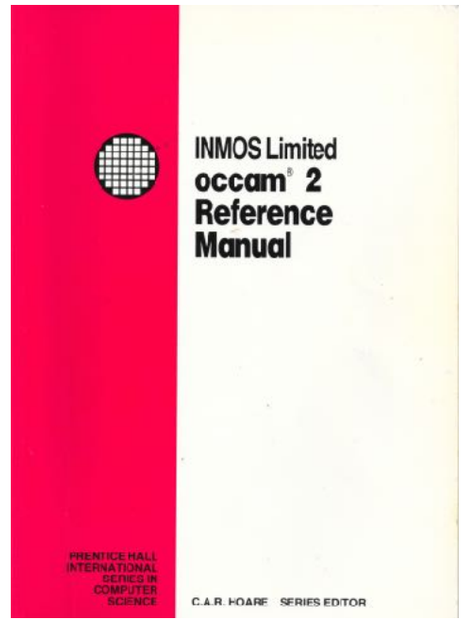
- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]
- XC by XMOS on XMOS multi-core processors
  - I will show you some here. Has channels and interfaces
  - Also based on CSP



AT NTNU?

[1] http://wotug.cs.unlv.edu/generate-program.php?id=1

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]
- XC by XMOS on XMOS multi-core processors
  - I will show you some here. Has channels and interfaces
  - Also based on CSP

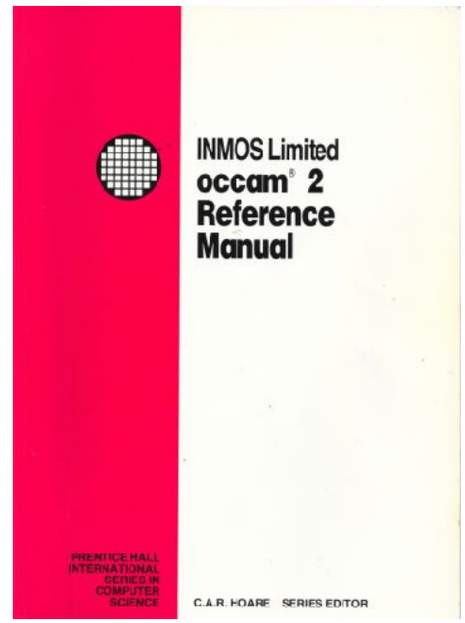


AT NTNU?

[1] http://wotug.cs.unlv.edu/generate-program.php?id=1

[2] https://softwareengineering.stackexchange.com/questions/135104/rendezvous-in-ada

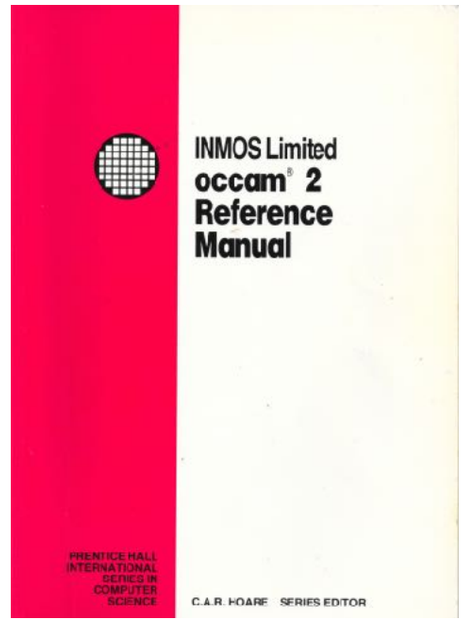
- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]
- XC by XMOS on XMOS multi-core processors
  - I will show you some here. Has channels and interfaces
  - Also based on CSP



AT NTNU?

[1] http://wotug.cs.unlv.edu/generate-program.php?id=1
[2] https://softwareengineering.stackexchange.com/questions/135104/rendezvous-in-ada
[3] https://swtch.com/~rsc/thread/

- occam has (had) channels. Based on CSP (*more later*)
  - Was presented here. Is not used in the industry any more, but occam-pi is used as a research language
    - «Unifying Concurrent Programming and Formal Verification within One Language» by Welch et.al. [1]
- Ada is presented in this course. Has rendezvous
  - Concurrency-part also based on CSP (and more) [2]
- go is presented in this course. Has channels
  - Also concurrency based on CSP. See next slide
  - Read «Bell Labs and CSP Threads». Not invented there (but in the UK) - still impressing [3]
- XC by XMOS on XMOS multi-core processors
  - I will show you some here. Has channels and interfaces
  - Also based on CSP



GO: FREQUENTLY ASKED QUESTIONS (FAQ)

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects.

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

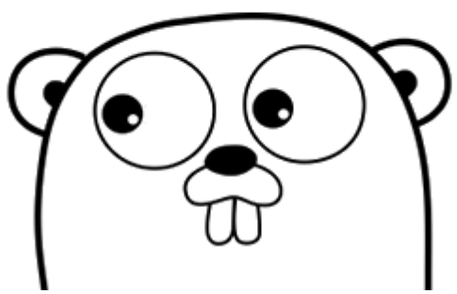
Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus



«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

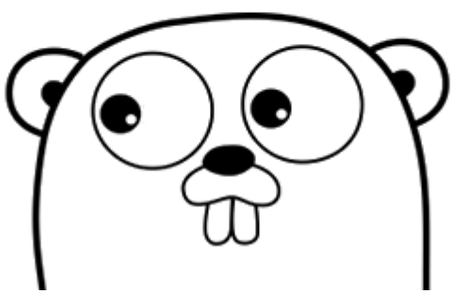
Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus



»

https://golang.org/doc/faq#csp

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

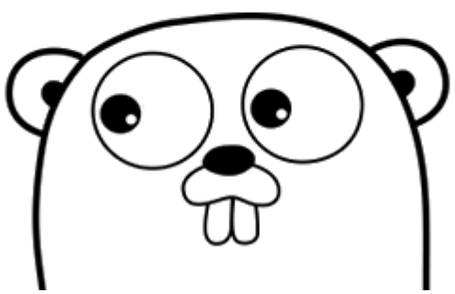
Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus



»

https://golang.org/doc/faq#csp

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

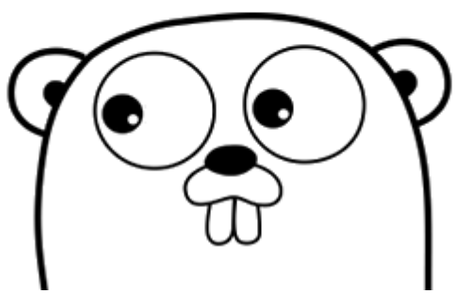
Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus



»

https://golang.org/doc/faq#csp

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

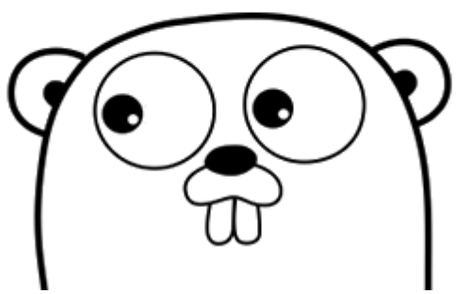
Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus



»

https://golang.org/doc/faq#csp

«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

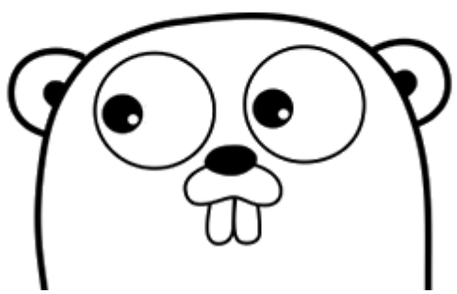
Concurrency and multi-threaded programming have a reputation for difficulty.

We believe this is due partly to complex designs such as pthreads and partly to overemphasis on low-level details such as mutexes, condition variables, and memory barriers.

One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

Occam and Erlang are two well known languages that stem from CSP.

Go's concurrency primitives derive from … notion of channels as first class objects. Pi-calculus



»

https://golang.org/doc/faq#csp

MORE THAN CONNECT THREADS ?

SOME IMPORTANT PROPERTIES

CONCURRENT?

CONCURRENT?

PARALLEL?

CONCURRENT?

PARALLEL?

REAL-TIME?

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core
- Real-time: meeting deadlines

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core
- Real-time: meeting deadlines

- XC is closest to having all properties

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core
- Real-time: meeting deadlines

- XC is closest to having all properties
  - since I guess, if it's parallel then it's concurrent

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core
- Real-time: meeting deadlines

- XC is closest to having all properties
  - since I guess, if it's parallel then it's concurrent
- Ada if «Ravenscar profile» (that removes rendezvous!)

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core
- Real-time: meeting deadlines

- XC is closest to having all properties
  - since I guess, if it's parallel then it's concurrent
- Ada if «Ravenscar profile» (that removes rendezvous!)
- Go is «not real-time»

CONCURRENT? PARALLEL? REAL-TIME?

- Concurrent: tasks scheduled on single-core
- Parallel: multi-core
- Real-time: meeting deadlines

- XC is closest to having all properties
  - since I guess, if it's parallel then it's concurrent
- Ada if «Ravenscar profile» (that removes rendezvous!)
- Go is «not real-time»
- Occam on many transputers and one transputer; different properties. Not really relevant any more, or.. yet(?)

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	+0.58%
18	19	↑	MATLAB	1.653 %	+0.07%
19	13	⇊	Go	1.569 %	−1 %
20	20		PL/SQL	1.429 %	−0 %

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	+0.58%
18	19	↑	MATLAB	1.653 %	+0.07%
19	13	⇊	Go	1.569 %	−1 %
20	20		PL/SQL	1.429 %	−0 %

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

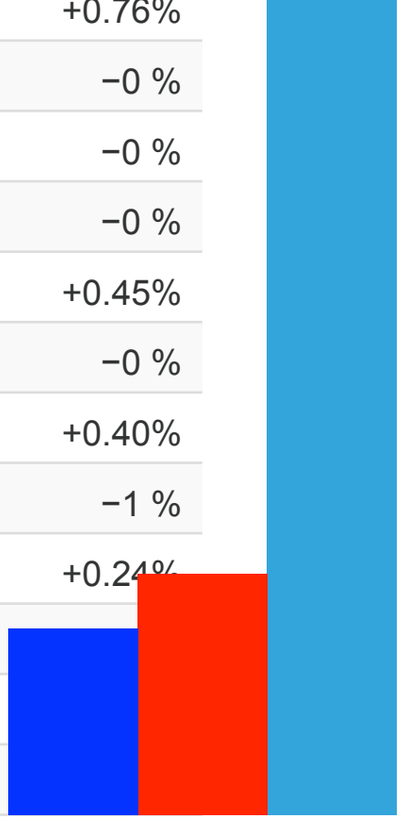
Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	+0.58%
18	19	↑	MATLAB	1.653 %	+0.07%
19	13	⇊	Go	1.569 %	−1%
20	20		PL/SQL	1.429 %	−0 %

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	
18	19	↑	MATLAB	1.653 %	
19	13	⇊	Go	1.569 %	
20	20		PL/SQL	1.429 %	−0 %

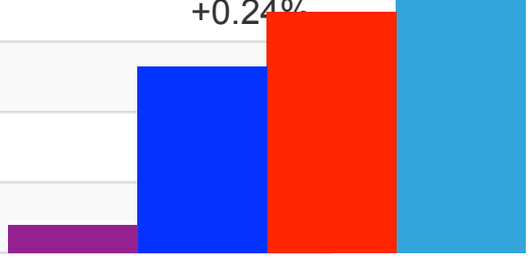


TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	
18	19	↑	MATLAB	1.653 %	
19	13	⇊	Go	1.569 %	
20	20		PL/SQL	1.429 %	−0 %



TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	+0.58%
18	19	↑	MATLAB	1.653 %	+0.07%
19	13	⇊	Go	1.569 %	−1 %
20	20		PL/SQL	1.429 %	−0 %

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1 %
4	5	↑	Python	4.678 %	+1.21%
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.76%
9	10	↑	PHP	2.532 %	−0 %
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0 %
14	15	↑	Visual Basic	2.314 %	+0.40%
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	+0.58%
18	19	↑	MATLAB	1.653 %	+0.07%
19	13	⇊	Go	1.569 %	−1 %
20	20		PL/SQL	1.429 %	−0 %

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1
4	5	↑	Python	4.678 %	+1.2
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.7
9	10	↑	PHP	2.532 %	−0
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0
14	15	↑	Visual Basic	2.314 %	+0.4
15	9	⇊	Assembly language	2.056 %	−1 %
16	18	↑	Objective-C	1.860 %	+0.24%
17	23	⇈	Scratch	1.740 %	+0.5
18	19	↑	MATLAB	1.653 %	+0.0
19	13	⇊	Go	1.569 %	−1
20	20		PL/SQL	1.429 %	−0 %

TIOBE Index for January 2018

January Headline: Programming Language C awarded Language of the Year 2017

https://www.tiobe.com

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215 %	−3 %
2	2		C	11.037 %	+1.69%
3	3		C++	5.603 %	−1
4	5	↑	Python	4.678 %	+1.2
5	4	↓	C#	3.754 %	−0 %
6	7	↑	JavaScript	3.465 %	+0.62%
7	6	↓	Visual Basic .NET	3.261 %	+0.30%
8	16	⇈	R	2.549 %	+0.7
9	10	↑	PHP	2.532 %	−0
10	8	↓	Perl	2.419 %	−0 %
11	12	↑	Ruby	2.406 %	−0 %
12	14	↑	Swift	2.377 %	+0.45%
13	11	↓	Delphi/Object Pascal	2.377 %	−0
	15	↑	Visual Basic	2.314 %	+0.4
	9	⇊	Assembly language	2.056 %	−1 %
16		↑	Objective-C	1.860 %	+0.24%
17		⇈	Scratch	1.740 %	+0.5
18	19		MATLAB	1.653 %	+0.0
19	13		Go	1.569 %	−1
20	20		PL/SQL	1.429 %	−0 %

Who code with chan!

1/7

Showing
a forest
for some trees

Showing
a forest
for some trees

MULTIPLE LOOPS WITH par: XC

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
int main() {
    par {
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    par {
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    par {
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    par {
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if i_spi[1];
    par {




    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if  i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if  i_spi[1];
    par {




    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if  i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if  i_spi[1];
    par {
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if  i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if  i_spi[1];
    par {
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if i_spi[1];
    par {
        on tile[0]:                          installExceptionHandler();
        on tile[0].core[0]: I2C_In_Task      (i_i2c_int);
        on tile[0].core[4]: I2C_Ex_Task      (i_i2c_ext);
        on tile[0]:         Sys_Task         (i_i2c_int[0], i_i2c_ext[0], i_adc_lib[0],
                                              i_heat_light[0], i_heat[0], i_water, c_buts,
                                              i_radio);
        on tile[0].core[0]: Temp_Heater_Task (i_heat, i_i2c_ext[1], i_heat_light[1]);
        on tile[0].core[5]: Temp_Water_Task  (i_water, i_heat[1]);
        on tile[0].core[1]: Button_Task      (BUT_L, but_left,   c_buts[BUT_L]);
        on tile[0].core[1]: Button_Task      (BUT_C, but_center, c_buts[BUT_C]);
        on tile[0].core[1]: Button_Task      (BUT_R, but_right,  c_buts[BUT_R]);
        on tile[0]:         ADC_Task         (i_adc_acq, i_adc_lib, NUM_ADC_DATA);
        on tile[0].core[5]: Port_HL_Task     (i_heat_light);
        on tile[0].core[4]: adc_Task         (i_adc_acq, c_ana, ADC_QUERY);
                            startkit_adc     (c_ana); // XMOS lib
        on tile[0].core[6]: Radio_Task       (i_radio, i_spi);
        on tile[0].core[7]: spi_master       (i_spi, 1, p_sclk, p_mosi, p_miso,
                                              p_ss, 1, clk_spi); // XMOS lib
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if i_spi[1];
    par {
        on tile[0]:                          installExceptionHandler();
        on tile[0].core[0]: I2C_In_Task      (i_i2c_int);
        on tile[0].core[4]: I2C_Ex_Task      (i_i2c_ext);
        on tile[0]:         Sys_Task         (i_i2c_int[0], i_i2c_ext[0], i_adc_lib[0],
                                              i_heat_light[0], i_heat[0], i_water, c_buts,
                                              i_radio);
        on tile[0].core[0]: Temp_Heater_Task (i_heat, i_i2c_ext[1], i_heat_light[1]);
        on tile[0].core[5]: Temp_Water_Task  (i_water, i_heat[1]);
        on tile[0].core[1]: Button_Task      (BUT_L, but_left,   c_buts[BUT_L]);
        on tile[0].core[1]: Button_Task      (BUT_C, but_center, c_buts[BUT_C]);
        on tile[0].core[1]: Button_Task      (BUT_R, but_right,  c_buts[BUT_R]);
        on tile[0]:         ADC_Task         (i_adc_acq, i_adc_lib, NUM_ADC_DATA);
        on tile[0].core[5]: Port_HL_Task     (i_heat_light);
        on tile[0].core[4]: adc_Task         (i_adc_acq, c_ana, ADC_QUERY);
                            startkit_adc     (c_ana); // XMOS lib
        on tile[0].core[6]: Radio_Task       (i_radio, i_spi);
        on tile[0].core[7]: spi_master       (i_spi, 1, p_sclk, p_mosi, p_miso,
                                              p_ss, 1, clk_spi); // XMOS lib
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if  i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if  i_spi[1];
    par {
        on tile[0]:                          installExceptionHandler();
        on tile[0].core[0]: I2C_In_Task      (i_i2c_int);
        on tile[0].core[4]: I2C_Ex_Task      (i_i2c_ext);
        on tile[0]:         Sys_Task         (i_i2c_int[0], i_i2c_ext[0], i_adc_lib[0],
                                              i_heat_light[0], i_heat[0], i_water, c_buts,
                                              i_radio);
        on tile[0].core[0]: Temp_Heater_Task (i_heat, i_i2c_ext[1], i_heat_light[1]);
        on tile[0].core[5]: Temp_Water_Task  (i_water, i_heat[1]);
        on tile[0].core[1]: Button_Task      (BUT_L, but_left,   c_buts[BUT_L]);
        on tile[0].core[1]: Button_Task      (BUT_C, but_center, c_buts[BUT_C]);
        on tile[0].core[1]: Button_Task      (BUT_R, but_right,  c_buts[BUT_R]);
        on tile[0]:         ADC_Task         (i_adc_acq, i_adc_lib, NUM_ADC_DATA);
        on tile[0].core[5]: Port_HL_Task     (i_heat_light);
        on tile[0].core[4]: adc_Task         (i_adc_acq, c_ana, ADC_QUERY);
                            startkit_adc     (c_ana); // XMOS lib
        on tile[0].core[6]: Radio_Task       (i_radio, i_spi);
        on tile[0].core[7]: spi_master       (i_spi, 1, p_sclk, p_mosi, p_miso,
                                              p_ss, 1, clk_spi); // XMOS lib
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //             c_is_channel
    chan           c_buts[NUM_BUTTONS];
    chan           c_ana;
    //             i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if     i_i2c_ext[NUM_I2C_EX];
    i2c_int_if     i_i2c_int[NUM_I2C_IN];
    adc_acq_if     i_adc_acq;
    adc_lib_if     i_adc_lib[NUM_ADC];
    heat_light_if  i_heat_light[NUM_HEAT_LIGHT];
    heat_if        i_heat[NUM_HEAT_CTRL];
    water_if       i_water;
    radio_if       i_radio;
    spi_master_if  i_spi[1];
    par {
        on tile[0]:                          installExceptionHandler();
        on tile[0].core[0]: I2C_In_Task      (i_i2c_int);
        on tile[0].core[4]: I2C_Ex_Task      (i_i2c_ext);
        on tile[0]:         Sys_Task         (i_i2c_int[0], i_i2c_ext[0], i_adc_lib[0],
                                              i_heat_light[0], i_heat[0], i_water, c_buts,
                                              i_radio);
        on tile[0].core[0]: Temp_Heater_Task (i_heat, i_i2c_ext[1], i_heat_light[1]);
        on tile[0].core[5]: Temp_Water_Task  (i_water, i_heat[1]);
        on tile[0].core[1]: Button_Task      (BUT_L, but_left,   c_buts[BUT_L]);
        on tile[0].core[1]: Button_Task      (BUT_C, but_center, c_buts[BUT_C]);
        on tile[0].core[1]: Button_Task      (BUT_R, but_right,  c_buts[BUT_R]);
        on tile[0]:         ADC_Task         (i_adc_acq, i_adc_lib, NUM_ADC_DATA);
        on tile[0].core[5]: Port_HL_Task     (i_heat_light);
        on tile[0].core[4]: adc_Task         (i_adc_acq, c_ana, ADC_QUERY);
                            startkit_adc     (c_ana); // XMOS lib
        on tile[0].core[6]: Radio_Task       (i_radio, i_spi);
        on tile[0].core[7]: spi_master       (i_spi, 1, p_sclk, p_mosi, p_miso,
                                              p_ss, 1, clk_spi); // XMOS lib
    }
    return 0;
}
```

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //            c_is_channel
    chan          c_buts[NUM_BUTTONS];
    chan          c_ana;
    //            i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if    i_i2c_ext[NUM_I2C_EX];
    i2c_int_if    i_i2c_int[NUM_I2C_IN];
    adc_acq_if    i_adc_acq;
    adc_lib_if    i_adc_lib[NUM_ADC];
    heat_light_if i_heat_light[NUM_HEAT_LIGHT];
    heat_if       i_heat[NUM_HEAT_CTRL];
    water_if      i_water;
    radio_if      i_radio;
    spi_master_if i_spi[1];
    par {
        on tile[0]:                          installExceptionHandler();
        on tile[0].core[0]: I2C_In_Task      (i_i2c_int);
        on tile[0].core[4]: I2C_Ex_Task      (i_i2c_ext);
        on tile[0]:         Sys_Task         (i_i2c_int[0], i_i2c_ext[0], i_adc_lib[0],
                                              i_heat_light[0], i_heat[0], i_water, c_buts,
                                              i_radio);
        on tile[0].core[0]: Temp_Heater_Task (i_heat, i_i2c_ext[1], i_heat_light[1]);
        on tile[0].core[5]: Temp_Water_Task  (i_water, i_heat[1]);
        on tile[0].core[1]: Button_Task      (BUT_L, but_left,   c_buts[BUT_L]);
        on tile[0].core[1]: Button_Task      (BUT_C, but_center, c_buts[BUT_C]);
        on tile[0].core[1]: Button_Task      (BUT_R, but_right,  c_buts[BUT_R]);
        on tile[0]:         ADC_Task         (i_adc_acq, i_adc_lib, NUM_ADC_DATA);
        on tile[0].core[5]: Port_HL_Task     (i_heat_light);
        on tile[0].core[4]: adc_Task         (i_adc_acq, c_ana, ADC_QUERY);
                            startkit_adc     (c_ana); // XMOS lib
        on tile[0].core[6]: Radio_Task       (i_radio, i_spi);
        on tile[0].core[7]: spi_master       (i_spi, 1, p_sclk, p_mosi, p_miso,
                                              p_ss, 1, clk_spi); // XMOS lib
    }
    return 0;
}
```

THIS IS PARALLEL

Showing a forest for some trees

MULTIPLE LOOPS WITH par: XC

```
port but_left                = on tile[0]:XS1_PORT_1N;
port but_center              = on tile[0]:XS1_PORT_1O;
port but_right               = on tile[0]:XS1_PORT_1P;
out buffered port:32 p_miso  = XS1_PORT_1A;
out port             p_ss[1] = {XS1_PORT_1B};
out buffered port:22 p_sclk  = XS1_PORT_1C;
out buffered port:32 p_mosi  = XS1_PORT_1D;
clock                clk_spi = XS1_CLKBLK_1;

int main() {
    //            c_is_channel
    chan          c_buts[NUM_BUTTONS];
    chan          c_ana;
    //            i_is_interface, a collection of RPC-type functions with defined roles (none, client, server)
    i2c_ext_if    i_i2c_ext[NUM_I2C_EX];
    i2c_int_if    i_i2c_int[NUM_I2C_IN];
    adc_acq_if    i_adc_acq;
    adc_lib_if    i_adc_lib[NUM_ADC];
    heat_light_if i_heat_light[NUM_HEAT_LIGHT];
    heat_if       i_heat[NUM_HEAT_CTRL];
    water_if      i_water;
    radio_if      i_radio;
    spi_master_if i_spi[1];
    par {
        on tile[0]:                          installExceptionHandler();
        on tile[0].core[0]: I2C_In_Task      (i_i2c_int);
        on tile[0].core[4]: I2C_Ex_Task      (i_i2c_ext);
        on tile[0]:         Sys_Task         (i_i2c_int[0], i_i2c_ext[0], i_adc_lib[0],
                                              i_heat_light[0], i_heat[0], i_water, c_buts,
                                              i_radio);
        on tile[0].core[0]: Temp_Heater_Task (i_heat, i_i2c_ext[1], i_heat_light[1]);
        on tile[0].core[5]: Temp_Water_Task  (i_water, i_heat[1]);
        on tile[0].core[1]: Button_Task      (BUT_L, but_left,   c_buts[BUT_L]);
        on tile[0].core[1]: Button_Task      (BUT_C, but_center, c_buts[BUT_C]);
        on tile[0].core[1]: Button_Task      (BUT_R, but_right,  c_buts[BUT_R]);
        on tile[0]:         ADC_Task         (i_adc_acq, i_adc_lib, NUM_ADC_DATA);
        on tile[0].core[5]: Port_HL_Task     (i_heat_light);
        on tile[0].core[4]: adc_Task         (i_adc_acq, c_ana, ADC_QUERY);
                            startkit_adc     (c_ana); // XMOS lib
        on tile[0].core[6]: Radio_Task       (i_radio, i_spi);
        on tile[0].core[7]: spi_master       (i_spi, 1, p_sclk, p_mosi, p_miso,
                                              p_ss, 1, clk_spi); // XMOS lib
    }
    return 0;
}
```

THIS IS PARALLEL

XC from my aquarium controller and xTIMEcomposer

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future
  - More complicated

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future
  - More complicated
  - Less efficient

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future
  - More complicated
  - Less efficient
  - Easy to compose i.e. when_any

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future
  - More complicated
  - Less efficient
  - Easy to compose i.e. when_any
- Concurrency TS futures are not widely implemented

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future
  - More complicated
  - Less efficient
  - Easy to compose i.e. when_any
- Concurrency TS futures are not widely implemented

TS – Technical Specification

[1] Channels - An Alternative to Callbacks and Futures - John Bandela - CppCon 2016

CHANNELS – AN ALTERNATIVE TO CALLBACKS AND FUTURES

- Channels can be a useful way to think about concurrency
- Callback vs. future
- Callback
  - Conceptually simple
  - Efficient
  - Difficult to compose
- Future
  - More complicated
  - Less efficient
  - Easy to compose i.e. when_any
- Concurrency TS futures are not widely implemented

TS – Technical Specification

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

Google I/O 2012 - Go Concurrency Patterns

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

Google I/O 2012 - Go Concurrency Patterns

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

Google I/O 2012 - Go Concurrency Patterns

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

```
select {
    case v1 := <-c1:
        fmt.Printf("received %v from c1\n", v1)
    case v2 := <-c2:
        fmt.Printf("received %v from c2\n", v1)
    case c3 <- 23:
        fmt.Printf("sent %v to c3\n", 23)
    default:
        fmt.Printf("no one was ready to communicate\n")
    }
```

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

```
select {
    case v1 := <-c1:
        fmt.Printf("received %v from c1\n", v1)
    case v2 := <-c2:
        fmt.Printf("received %v from c2\n", v1)
    case c3 <- 23:
        fmt.Printf("sent %v to c3\n", 23)
    default:
        fmt.Printf("no one was ready to communicate\n")
    }
```

Optional, introduces busy poll, needed *some* times

Watch it!

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

```
select {
    case v1 := <-c1:
        fmt.Printf("received %v from c1\n", v1)
    case v2 := <-c2:
        fmt.Printf("received %v from c2\n", v1)
    case c3 <- 23:
        fmt.Printf("sent %v to c3\n", 23)
    default:
        fmt.Printf("no one was ready to communicate\n")
    }
```

Optional, introduces busy poll, needed *some* times

```
Alternative receives
x, ok       = <-ch
x, ok      := <-ch
var x, ok   = <-ch
var x, ok T = <-ch
```

Watch it!

https://talks.golang.org/2012/concurrency.slide#31

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

Google I/O 2012 - Go Concurrency Patterns

A control structure unique to concurrency.
The reason channels and goroutines are built into the language.

The select statement provides another way to handle multiple channels.
It's like a switch, but each case is a communication:

- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

```
select {
    case v1 := <-c1:
        fmt.Printf("received %v from c1\n", v1)
    case v2 := <-c2:
        fmt.Printf("received %v from c2\n", v1)
    case c3 <- 23:
        fmt.Printf("sent %v to c3\n", 23)
    default:
        fmt.Printf("no one was ready to communicate\n")
    }
```

Optional, introduces busy poll, needed *some* times

```
Alternative receives
                = <-ch
x, ok           := <-ch
x, ok            = <-ch
var x, ok   T = <-ch
var x, ok
```

Autronica

Discussing new runtime scheduler
made at NTH (1981)

Autronica

Discussing new runtime scheduler
made at NTH (1981)

Visiting Whessoe in Newton-Aycliffe (UK)
working with a 16-bits transputer (1995)

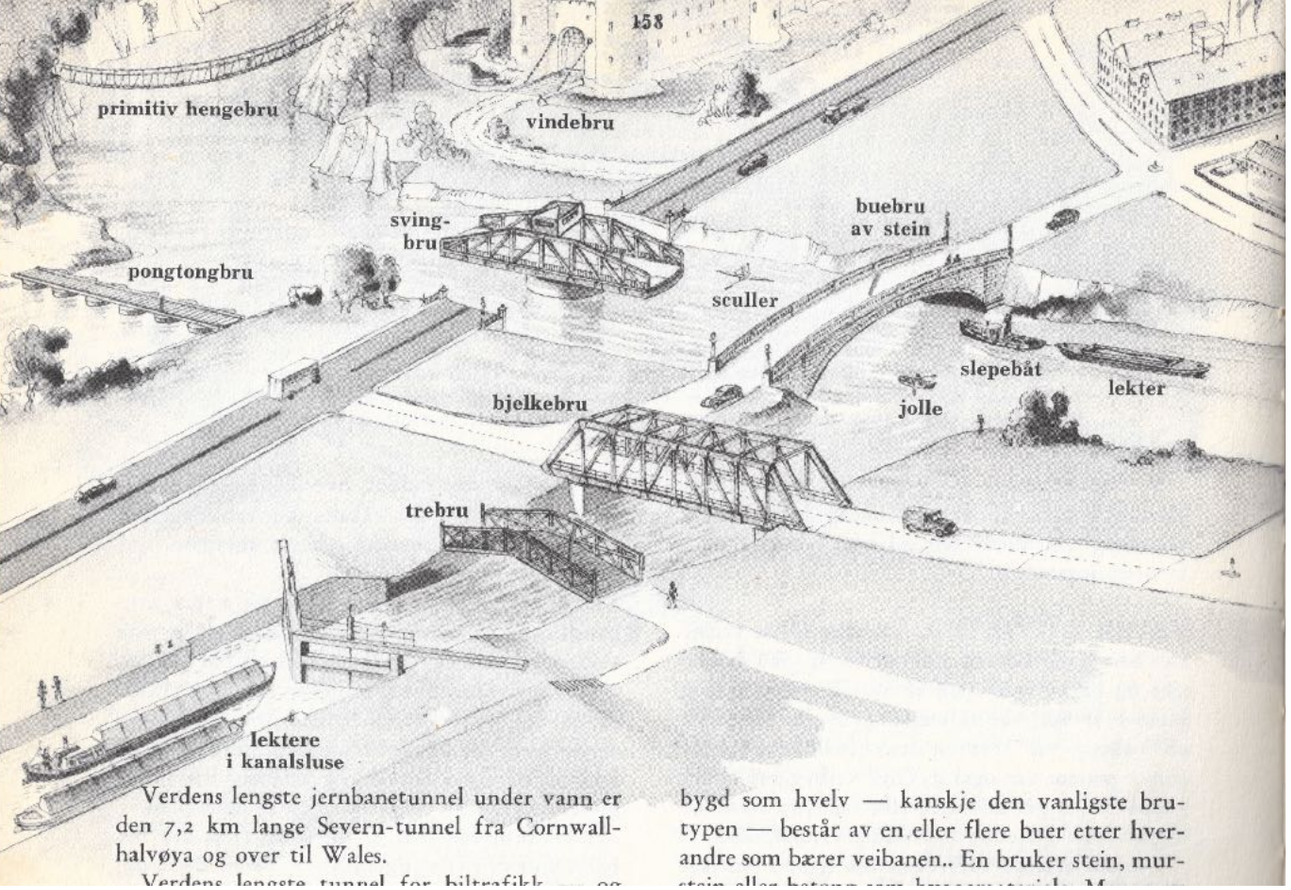
Autronica

Discussing new runtime scheduler made at NTH (1981)

Visiting Whessoe in Newton-Aycliffe (UK) working with a 16-bits transputer (1995)

Starting with C CSP-type schedulers (2002)

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

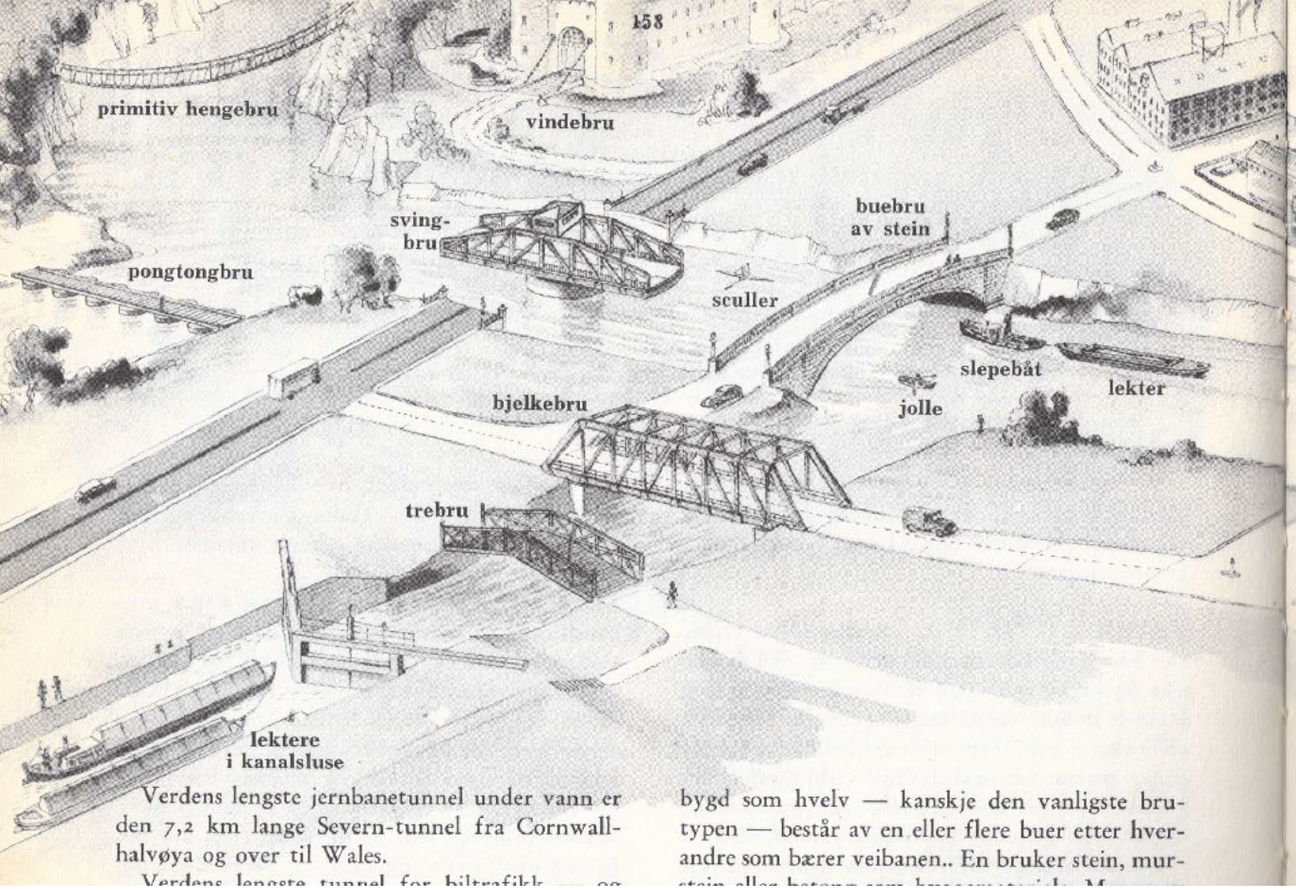


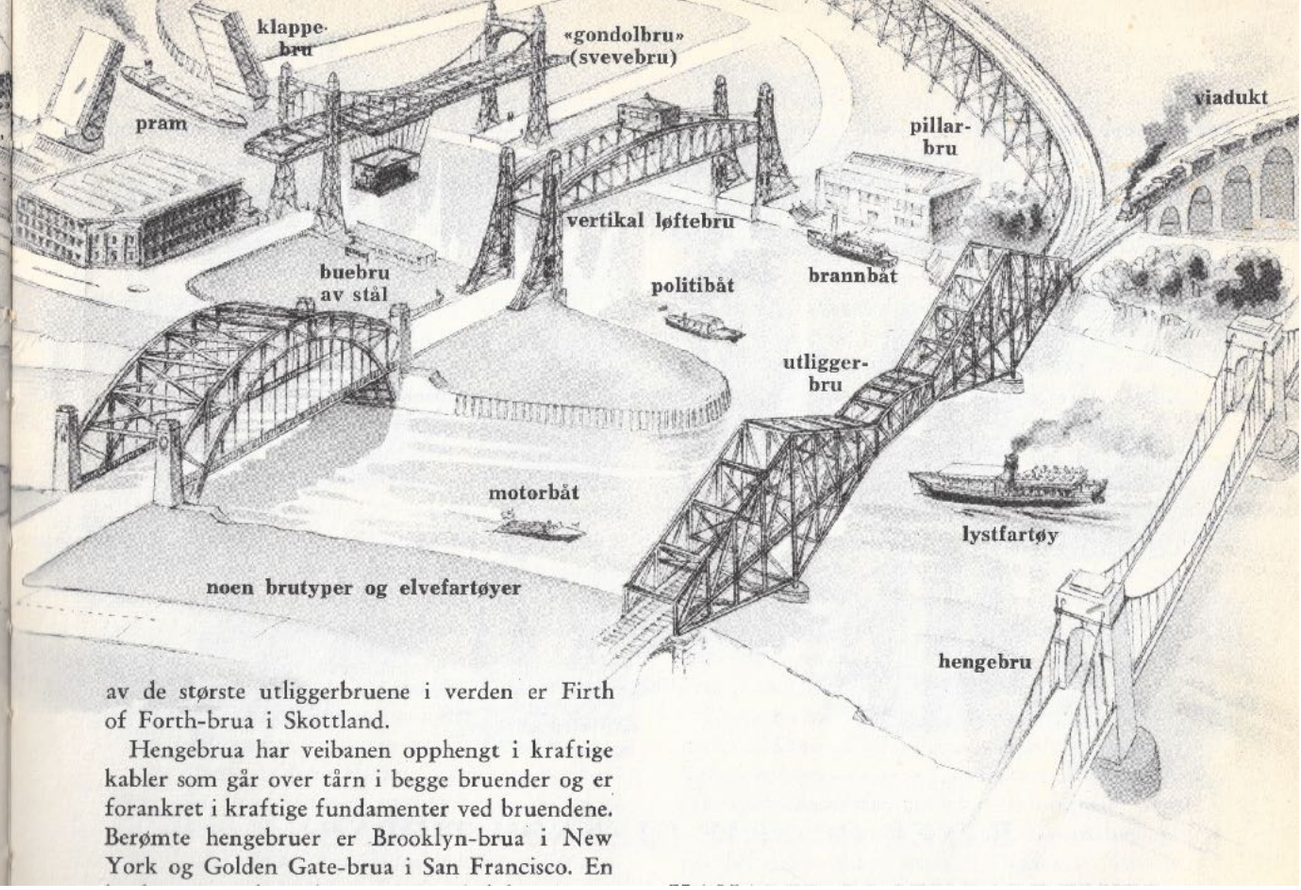
Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

Verdens lengste tunnel for biltrafikk — og

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")





noen brutyper og elvefartøyer

Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

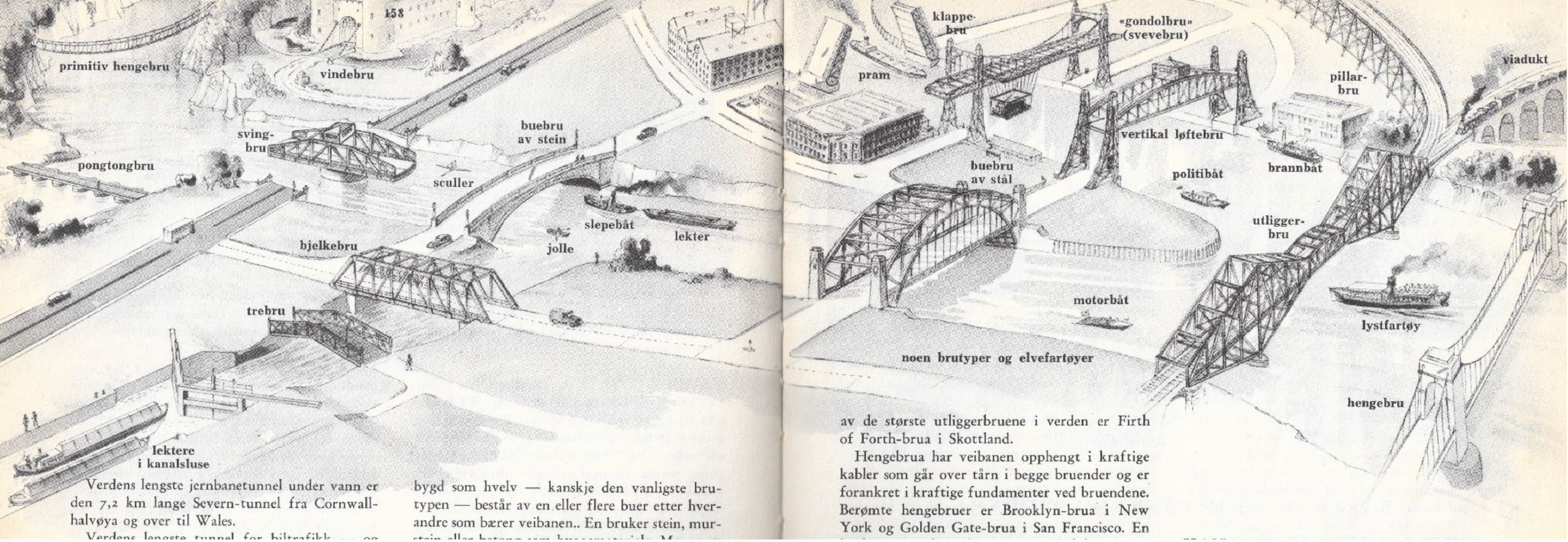
Verdens lengste tunnel for biltrafikk — og

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, murstein eller betong som

av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")



noen brutyper og elvefartøyer

Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

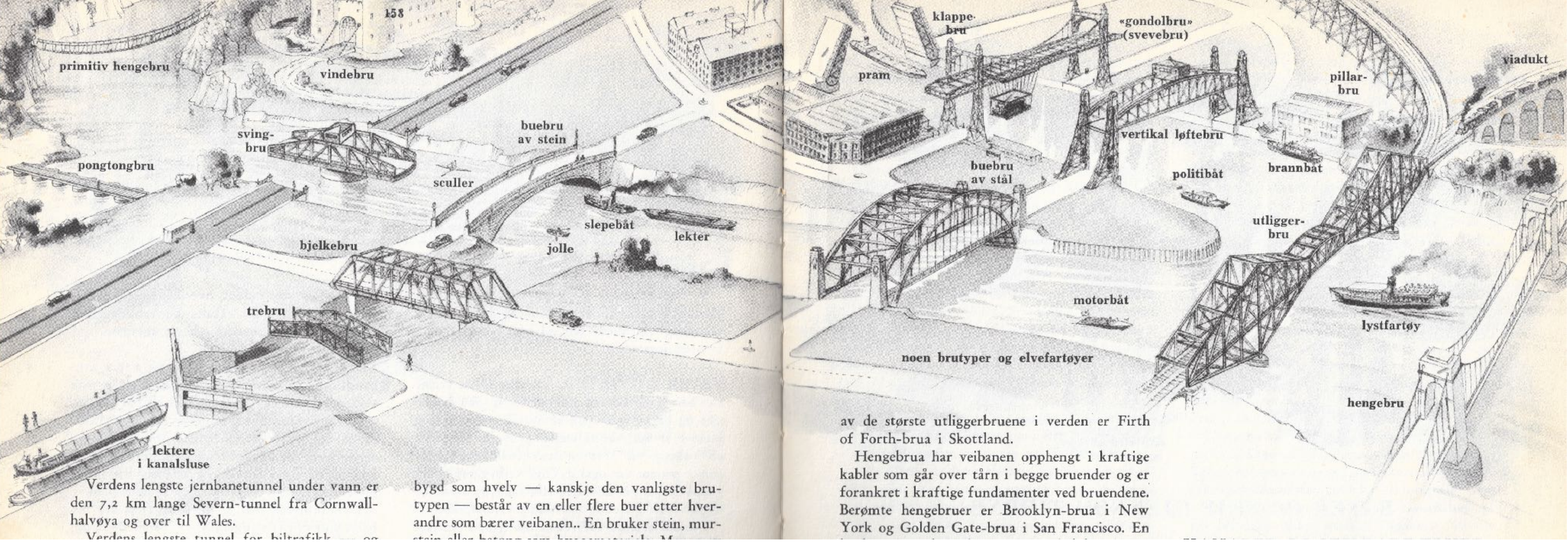
bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

BRIDGING A WORLD



Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

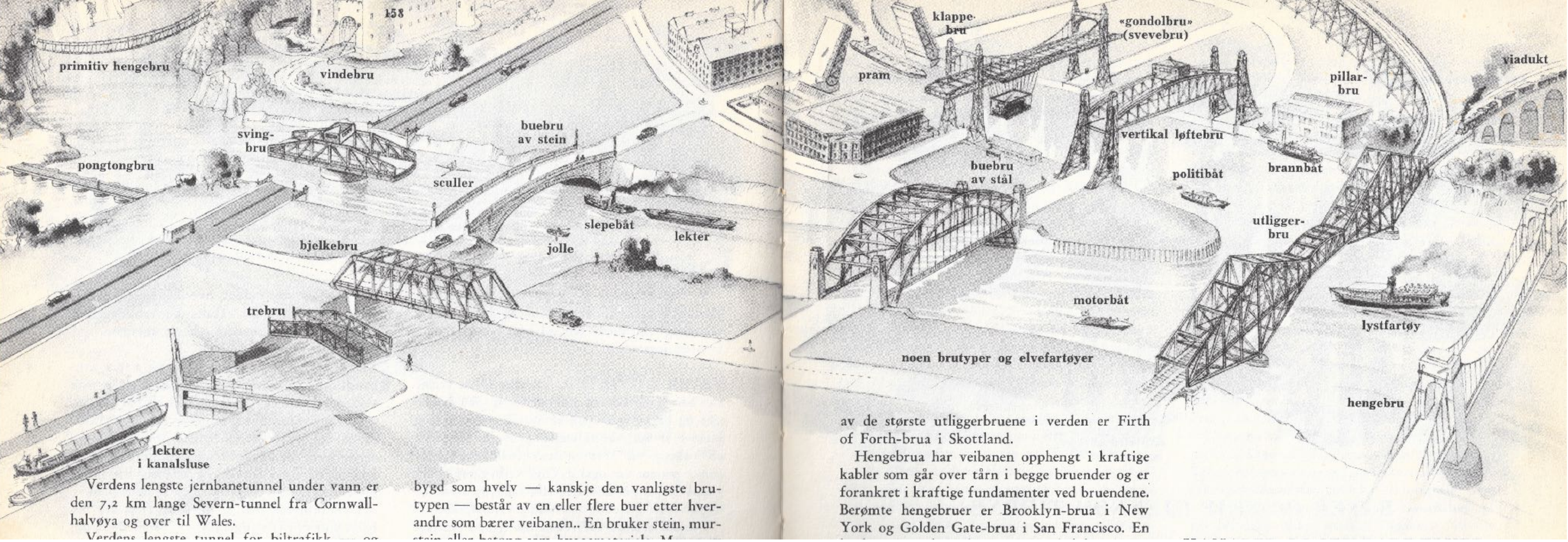
av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

BRIDGING A WORLD

- Some road bridges have access control



Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

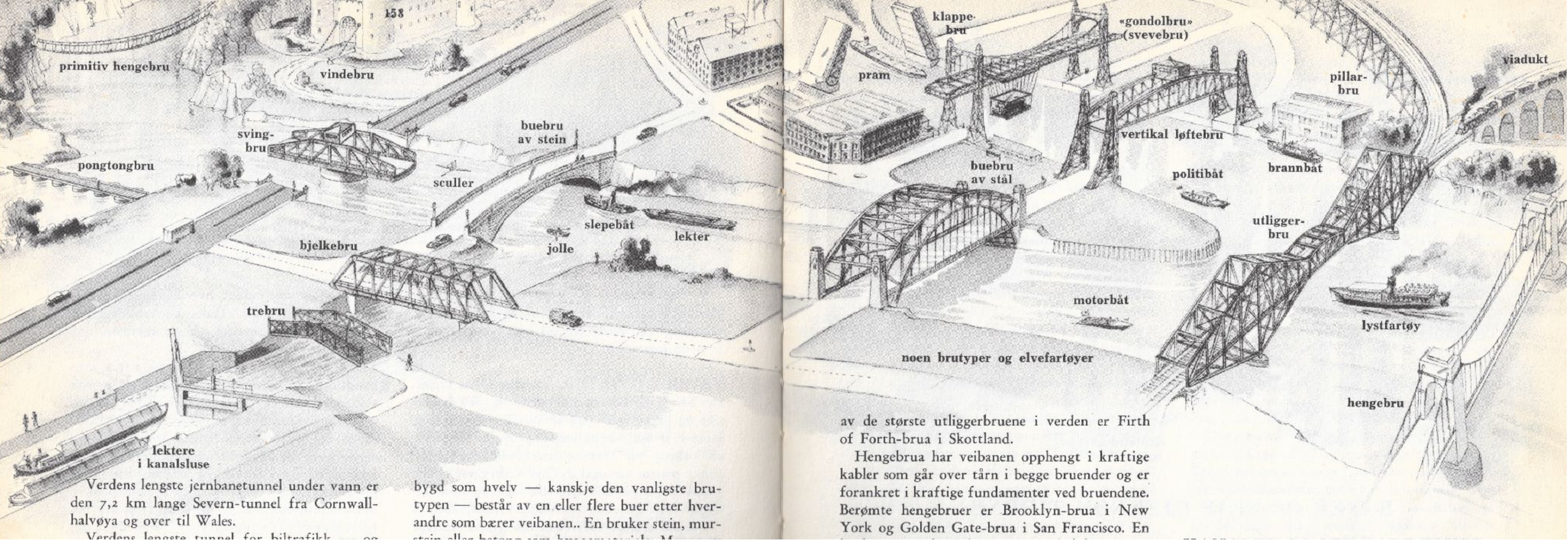
av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

BRIDGING A WORLD

- Some road bridges have access control
- Waiting ships and waiting cars are «orthogonal» (?)



noen brutyper og elvefartøyer

Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

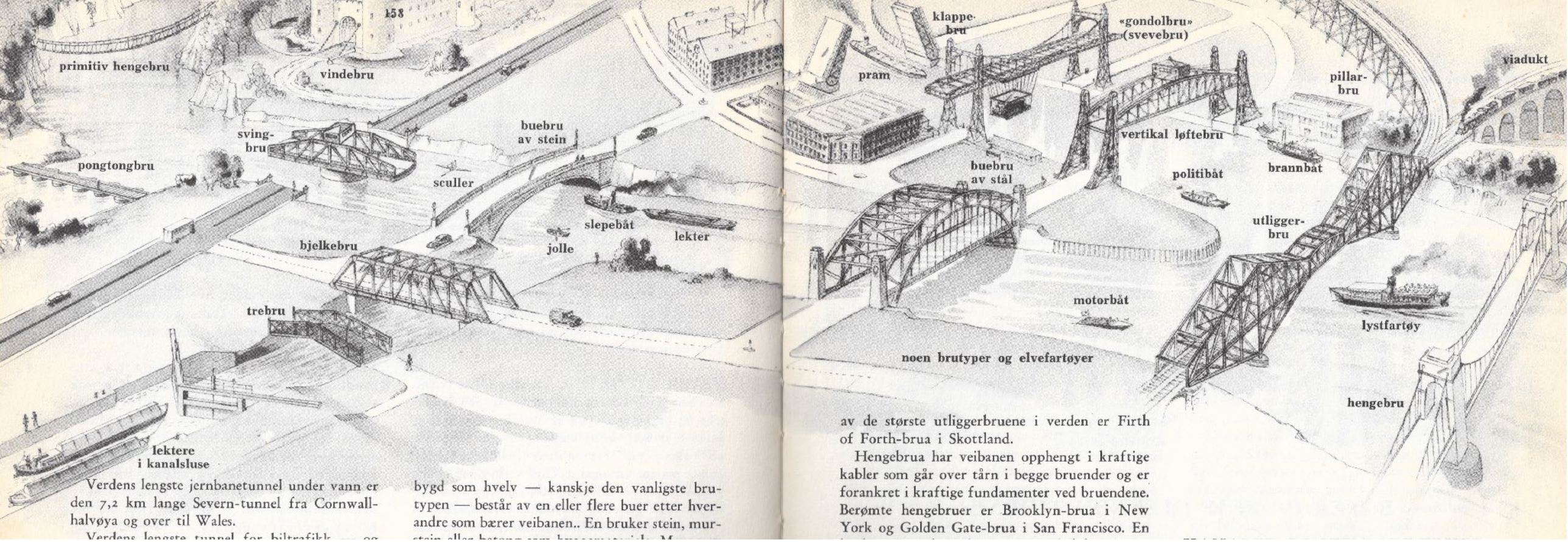
av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

BRIDGING A WORLD

- Some road bridges have access control
- Waiting ships and waiting cars are «orthogonal» (?)
- Some bridges are for cars, some for trains



Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

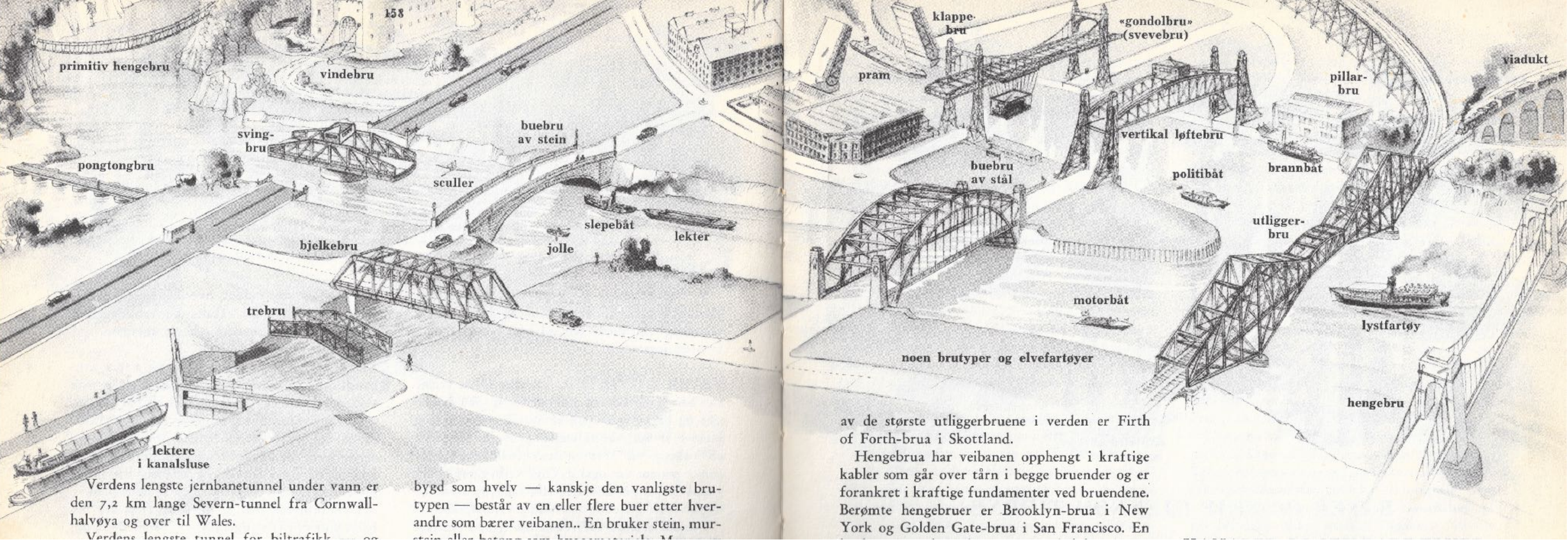
av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

BRIDGING A WORLD

- Some road bridges have access control
- Waiting ships and waiting cars are «orthogonal» (?)
- Some bridges are for cars, some for trains
- Some bridges are tall enough to let most ships through



noen brutyper og elvefartøyer

Verdens lengste jernbanetunnel under vann er den 7,2 km lange Severn-tunnel fra Cornwall-halvøya og over til Wales.

bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen.. En bruker stein, mur-

av de største utliggerbruene i verden er Firth of Forth-brua i Skottland.

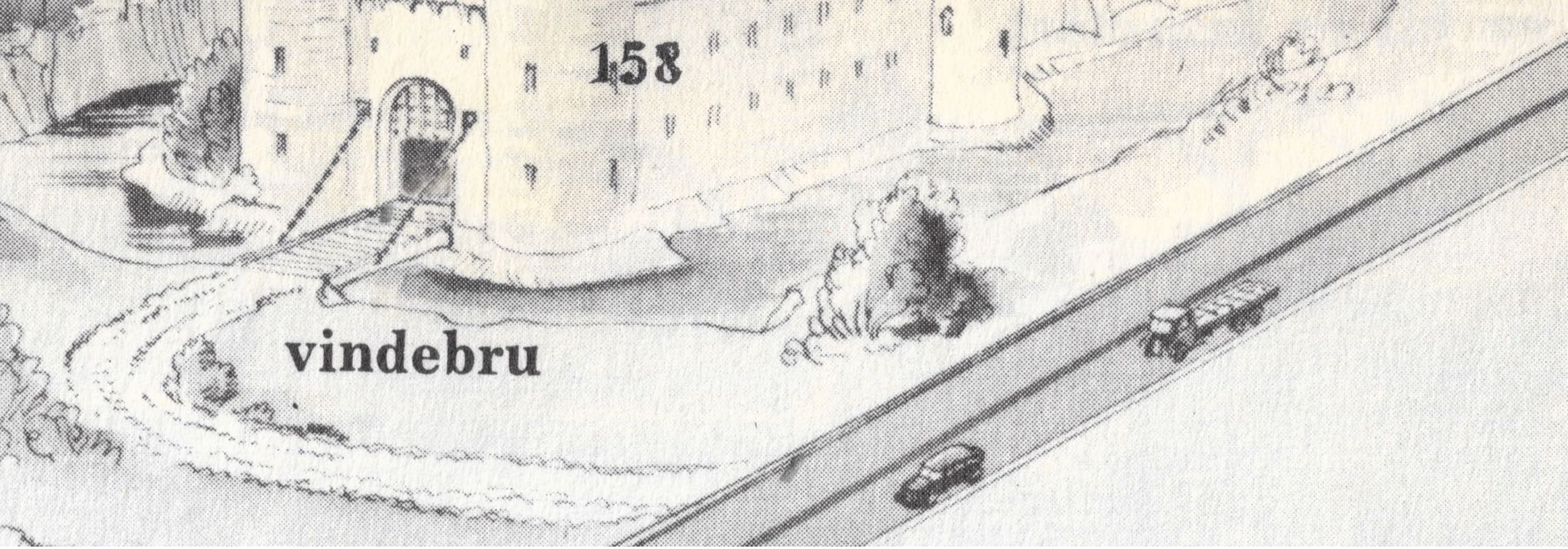
Hengebrua har veibanen opphengt i kraftige kabler som går over tårn i begge bruender og er forankret i kraftige fundamenter ved bruendene. Berømte hengebruer er Brooklyn-brua i New York og Golden Gate-brua i San Francisco. En

"Verden omkring oss", 1955 ("Odhams Encyclopedia for Children")

BRIDGING A WORLD

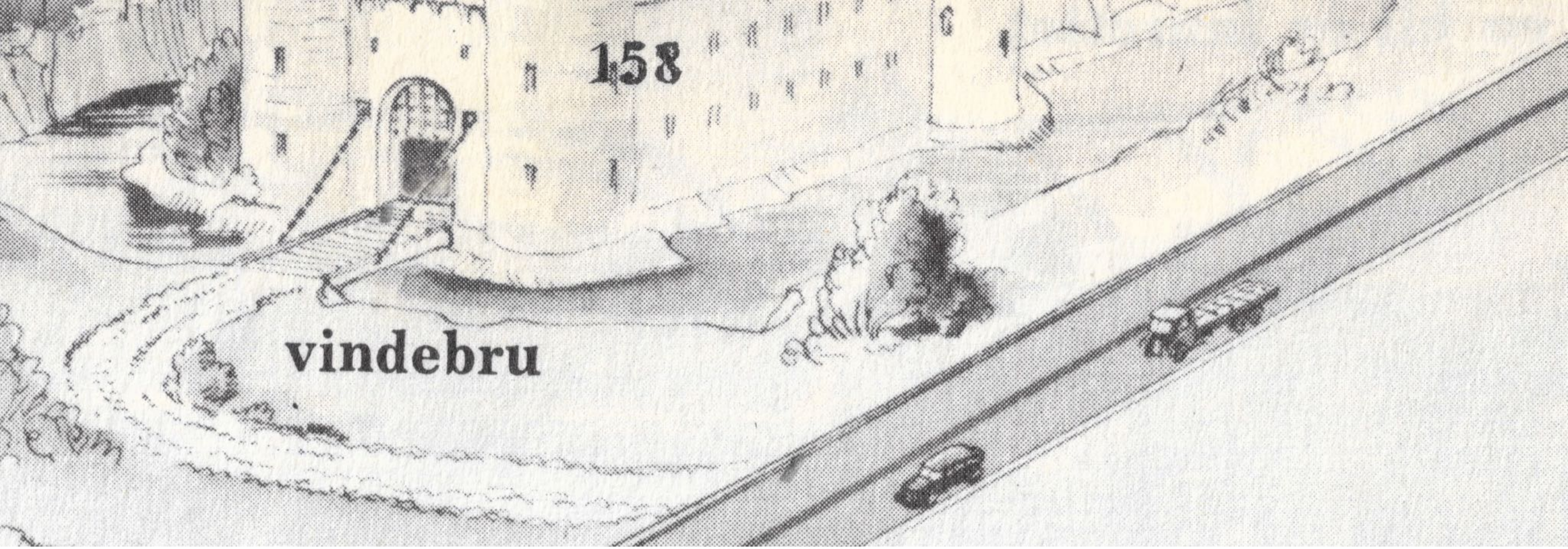
- Some road bridges have access control
- Waiting ships and waiting cars are «orthogonal» (?)
- Some bridges are for cars, some for trains
- Some bridges are tall enough to let most ships through
- Which part of this drawing might most resemble a CSP type system? (Even if CSPm may model everything)

THE CASTLE AND DRAWBRIDGE



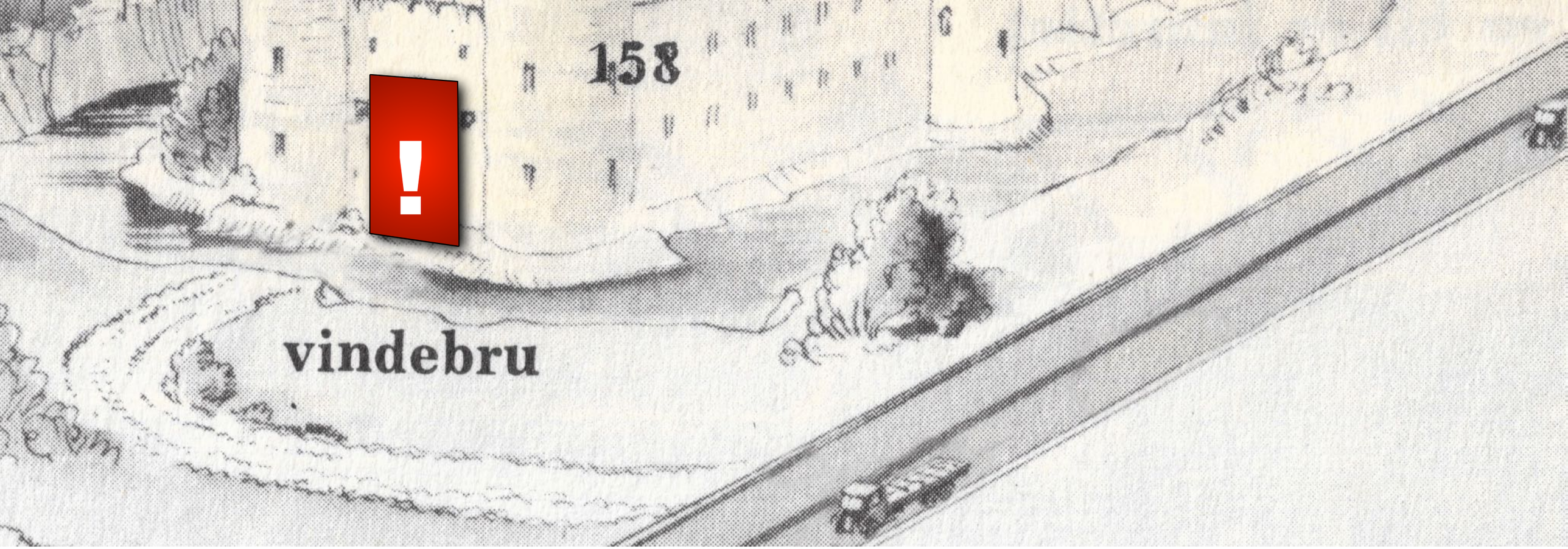
THE CASTLE AND DRAWBRIDGE

- The castle allows all traffic in (ok!)



THE CASTLE AND DRAWBRIDGE

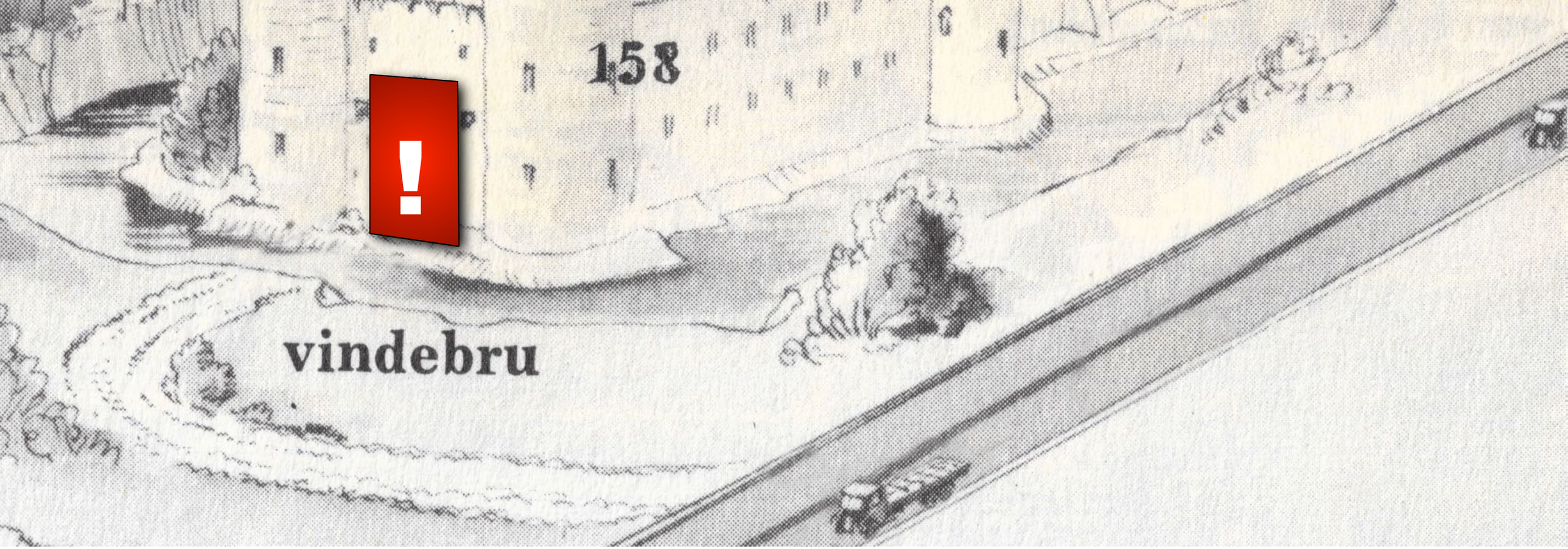
- The castle allows all traffic in (ok!)
- ok, if not disturbed!



THE CASTLE AND DRAWBRIDGE

- The castle allows all traffic in (ok!)
- ok, if not disturbed!

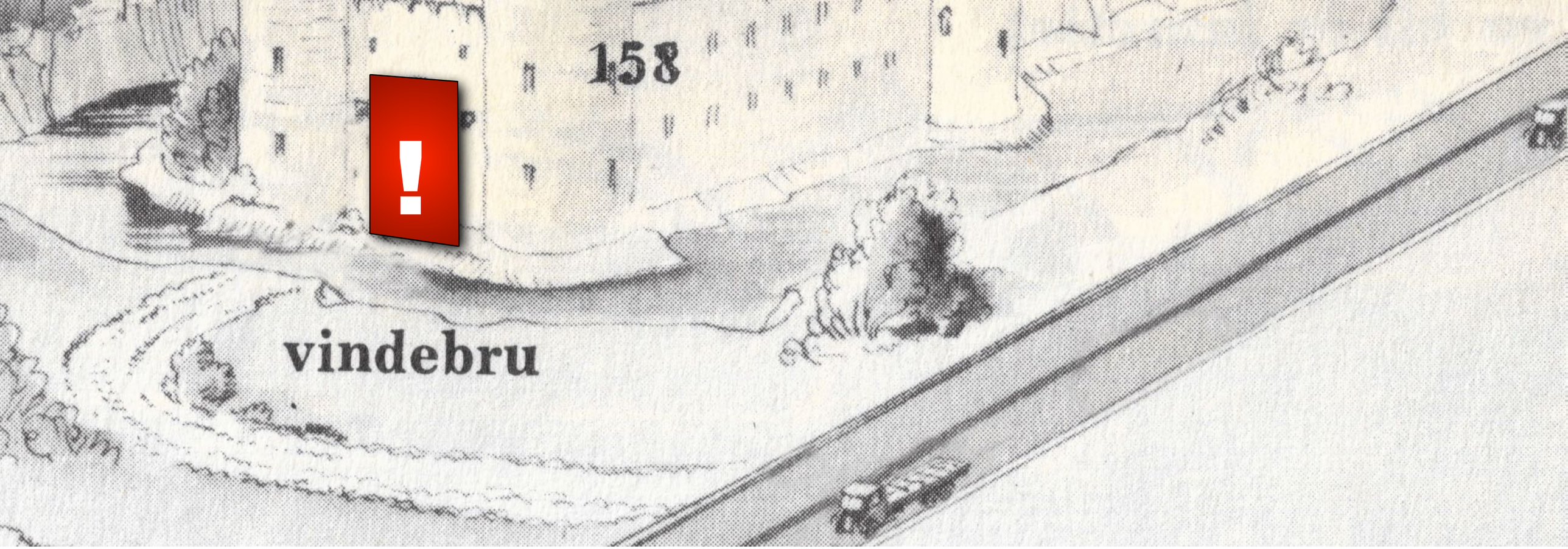
- Now it is protected!



THE CASTLE AND DRAWBRIDGE

- The castle allows all traffic in (ok!)
- ok, if not disturbed!

- Now it is protected!
- Doing something else



THE CASTLE AND DRAWBRIDGE

- The castle allows all traffic in (ok!)
- ok, if not disturbed!

- Now it is protected!
- Doing something else

- I guess that this is the most important page in this lecture!

TERMINOLOGY?

THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

THEY PROTECT THEM

TERMINOLOGY?

THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

THEY PROTECT THEM

TERMINOLOGY?

«DRAWBRIDGES»

THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

THEY PROTECT THEM

TERMINOLOGY?

«DRAWBRIDGES»

«GATES»

THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

THEY PROTECT THEM

TERMINOLOGY?

«DRAWBRIDGES»

«GATES»

THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

guards

THEY PROTECT THEM

TERMINOLOGY?

«DRAWBRIDGES»

«GATES»

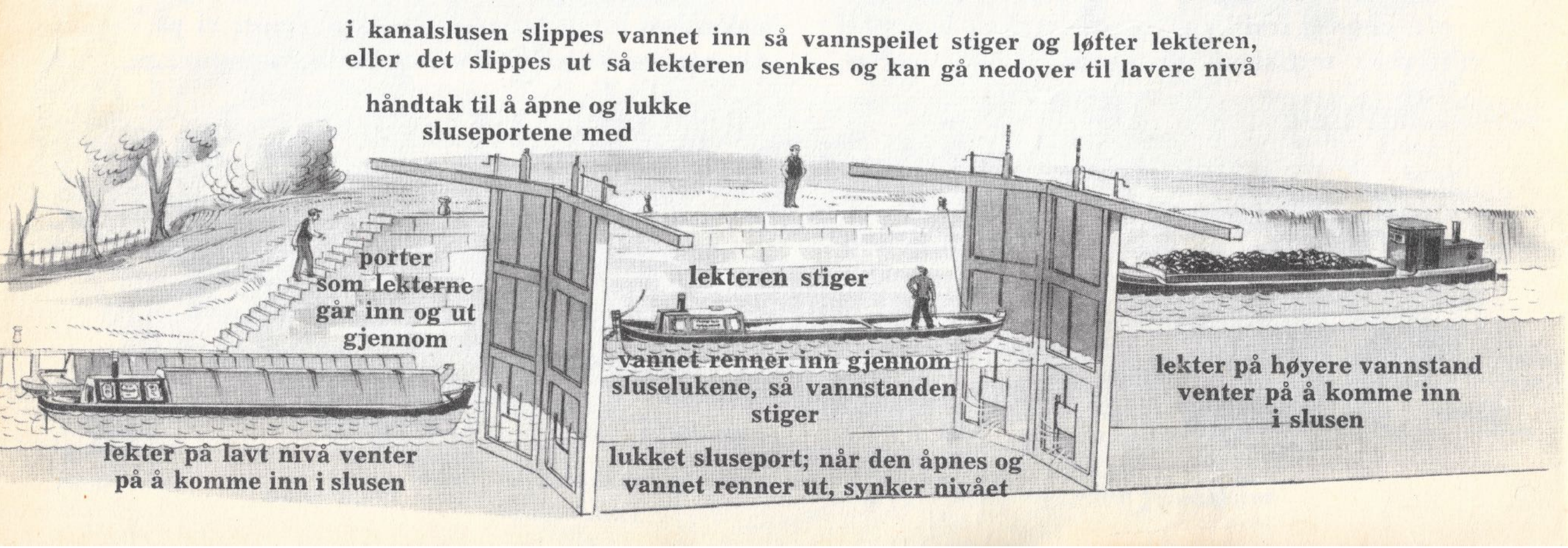
THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

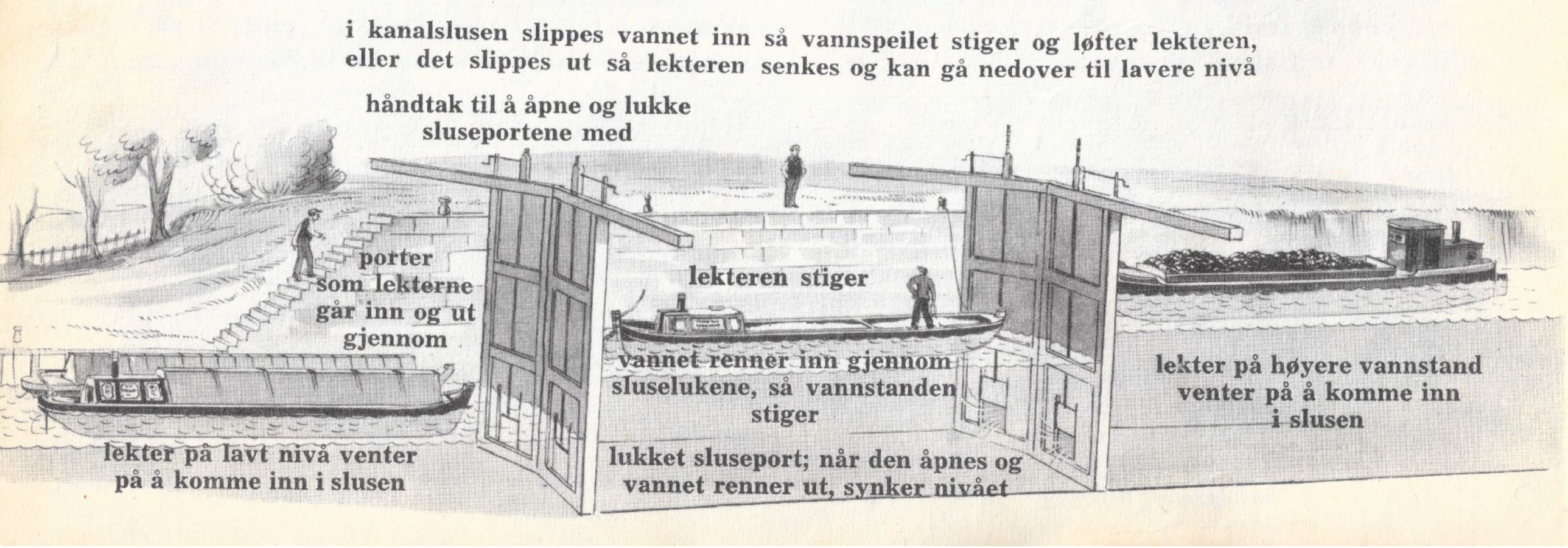
CSP «MODEL»

guards

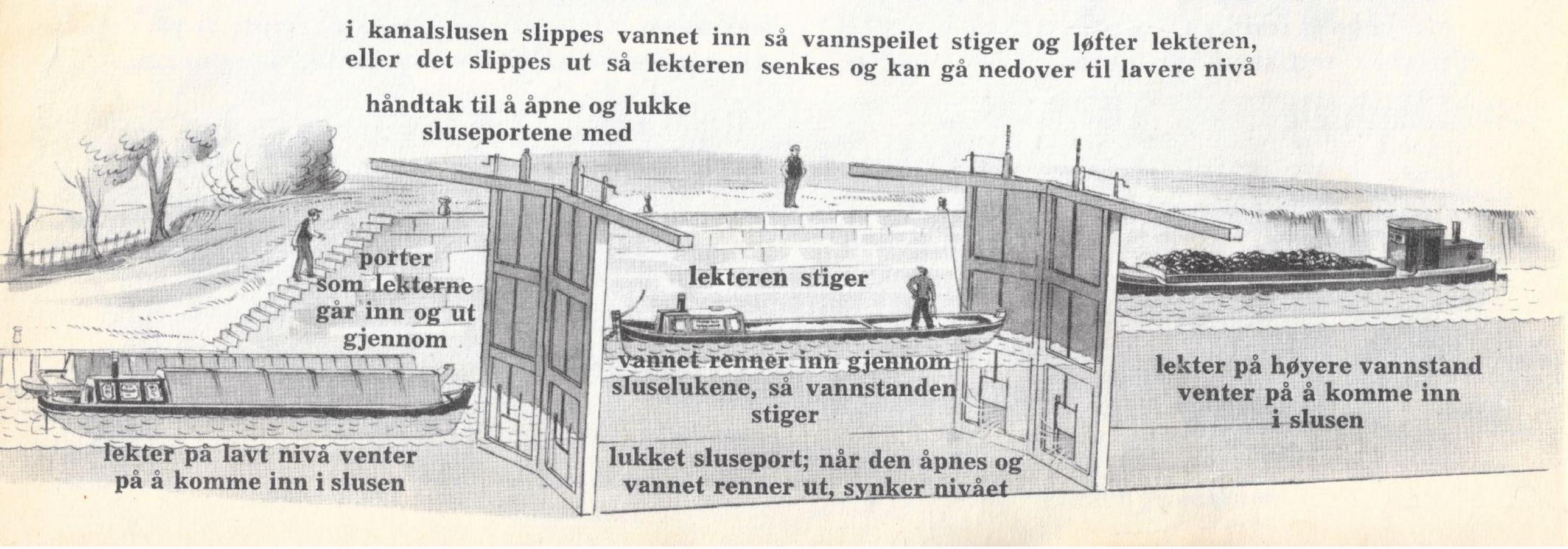
THEY PROTECT THEM



A CHANNEL HAS SEMANTICS

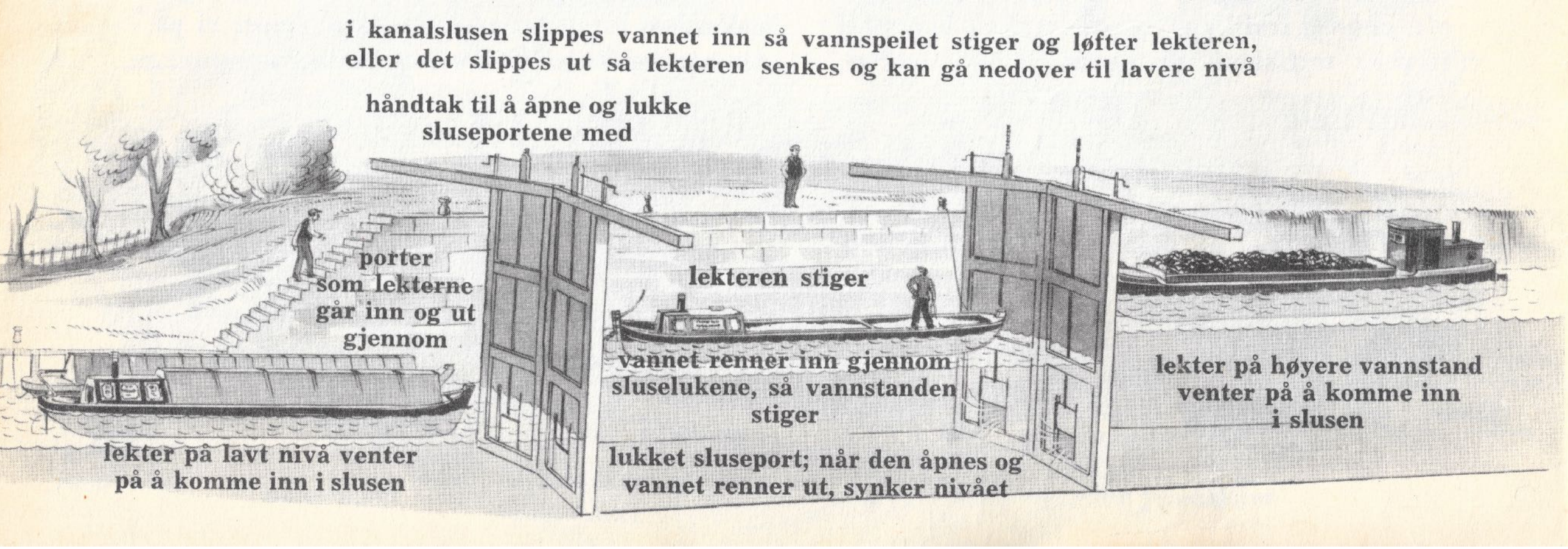


A CANAL LOCK HAS SEMANTICS



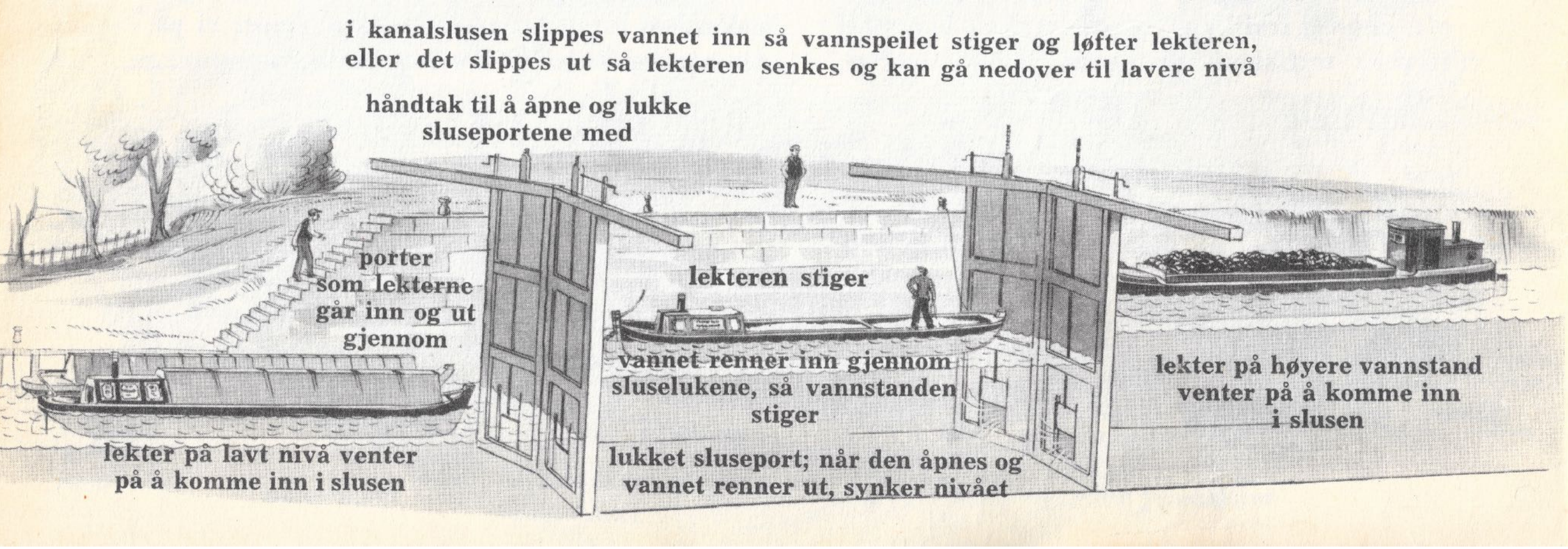
A CANAL LOCK HAS SEMANTICS

- Ship in one direction per turning



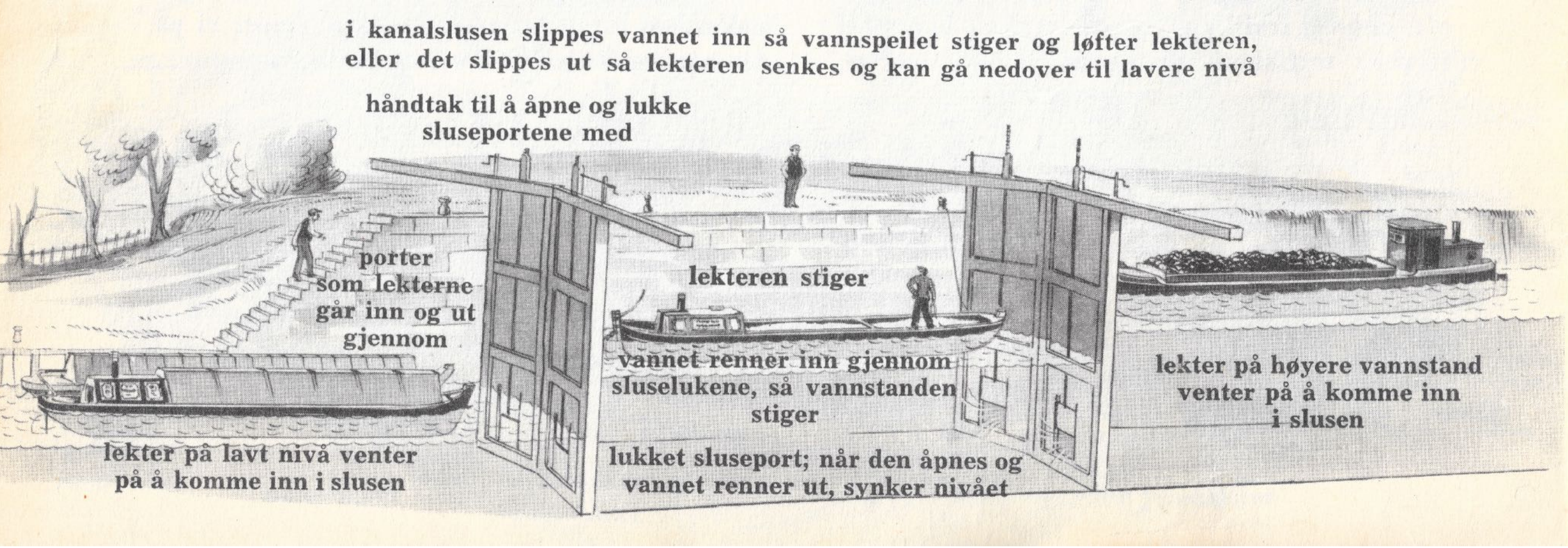
A CANAL LOCK HAS SEMANTICS

- Ship in one direction per turning
- The lock keeper operates it



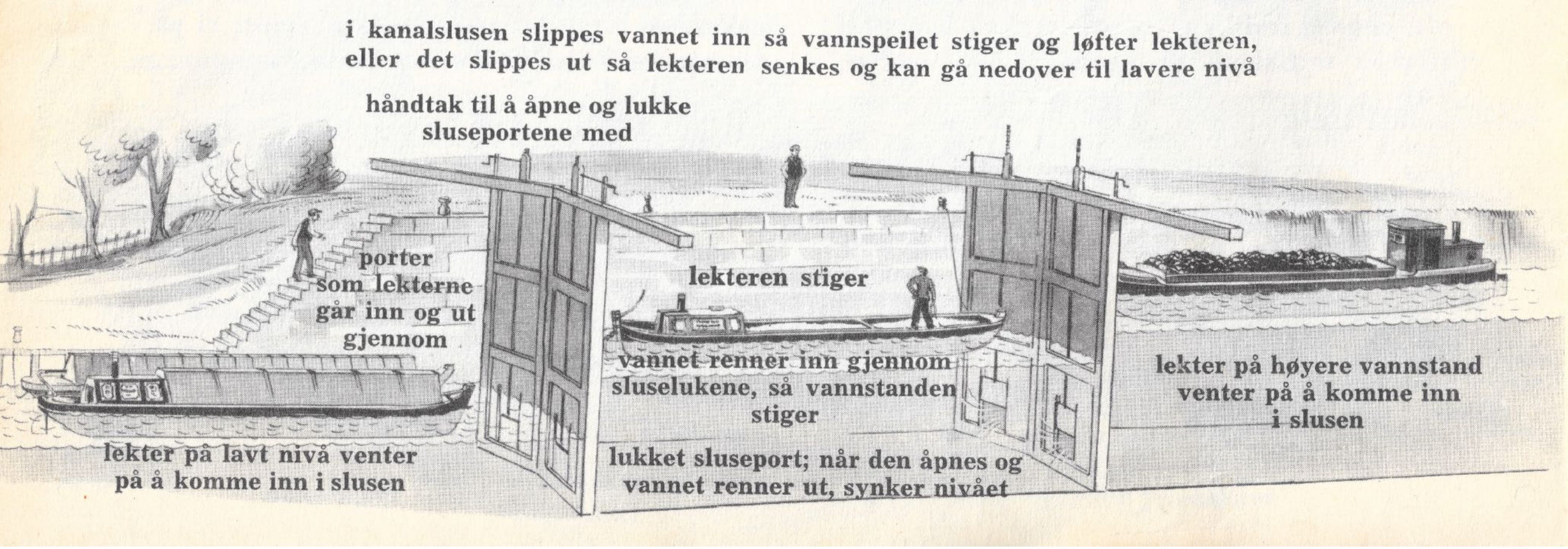
A CANAL LOCK HAS SEMANTICS

- Ship in one direction per turning
- The lock keeper operates it
- It has «states»



A CANAL LOCK HAS SEMANTICS

- Ship in one direction per turning
- The lock keeper operates it
- It has «states»
- Channels, buffers, queues, pipes also have their semantics



A CANAL LOCK HAS SEMANTICS

- Ship in one direction per turning
- The lock keeper operates it
- It has «states»
- Channels, buffers, queues, pipes also have their semantics
- Simplest CSP chan: synchronous, one-way, no buffer

CHANNEL SEMANTICS

A

B

X

CHANNEL SEMANTICS

CHANNEL SEMANTICS

CHANNEL SEMANTICS

CHANNEL SEMANTICS

A

B

X

CHANNEL SEMANTICS

A

B

X

A: run

CHANNEL SEMANTICS

A

B

X

A: run

B: dance

CHANNEL SEMANTICS

A

B

X

A: run

B: dance

CHANNEL SEMANTICS

A

B

X

A: run

B: dance - busy!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

B: dance - busy!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

B: dance - busy!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

CHANNEL SEMANTICS

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

CHANNEL SEMANTICS

A: run

first: have result!

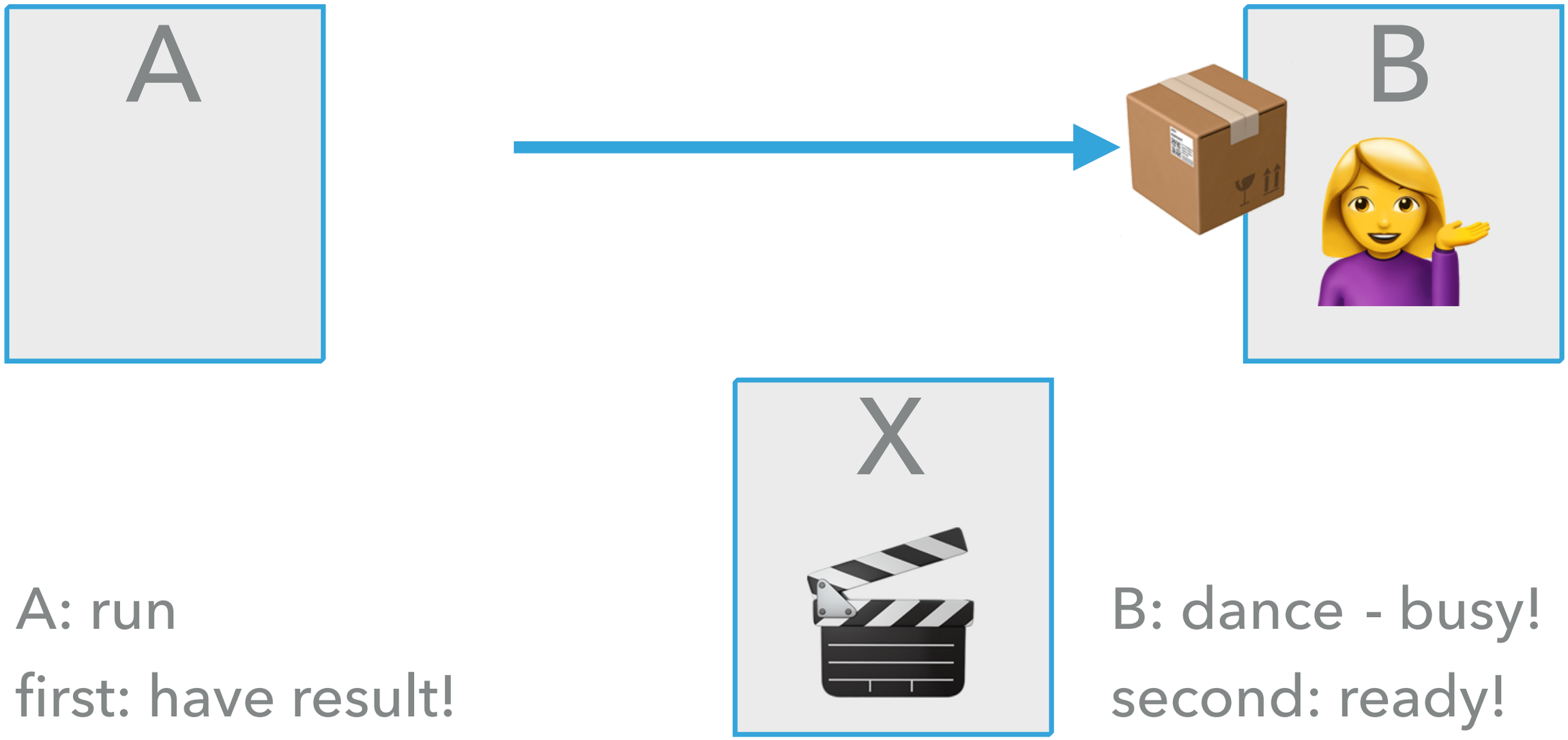
wait/sleep/block

B: dance - busy!

second: ready!

send > receive

CHANNEL SEMANTICS



A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

thanks! paint

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

B: dance - busy!

second: ready!

send > receive

thanks! paint

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

more to do?

send > receive

B: dance - busy!

second: ready!

thanks! paint

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

more to do?

send > receive

B: dance - busy!

second: ready!

thanks! paint

CHANNEL SEMANTICS

A

B

X

A: run

first: have result!

wait/sleep/block

more to do?

send > receive

synchronous
unbuffered

B: dance - busy!

second: ready!

thanks! paint

CHANNEL SEMANTICS

A

B

Has been undisturbed and running all the time!

X

A: run

first: have result!

wait/sleep/block

more to do?

B: dance - busy!

second: ready!

send > receive

synchronous unbuffered

thanks! paint

SAFE MEMCPY, NO POINTERS TO SHARED DATA

SAFE MEMCPY, NO POINTERS TO SHARED DATA

SAFE MEMCPY, NO POINTERS TO SHARED DATA

```
CHAN_OUT (Chan1, ACPtr->Data);
```

Chan state (first, local ptr, length)

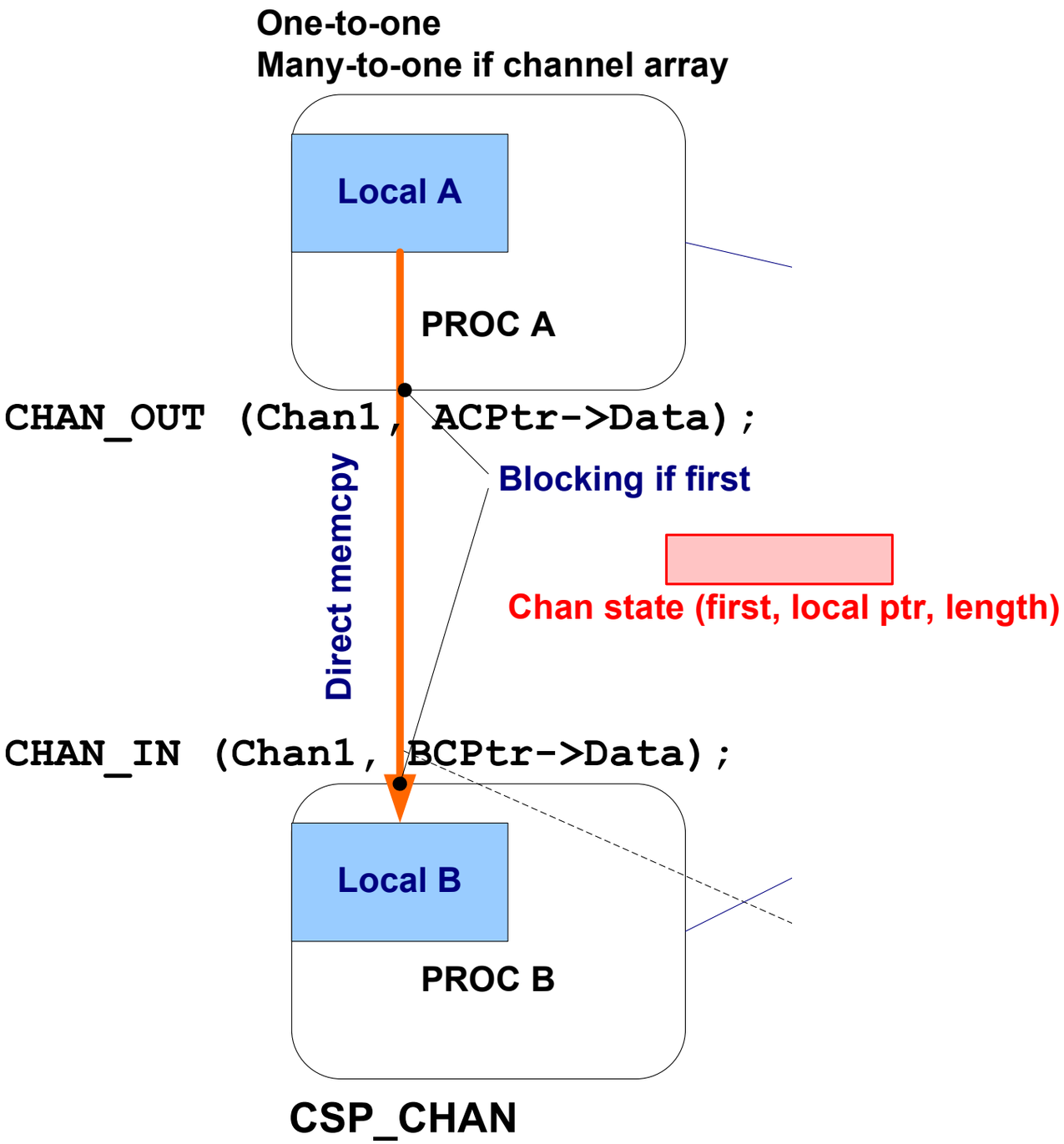
SAFE MEMCPY, NO POINTERS TO SHARED DATA

```
CHAN_OUT (Chan1, ACPtr->Data);
```

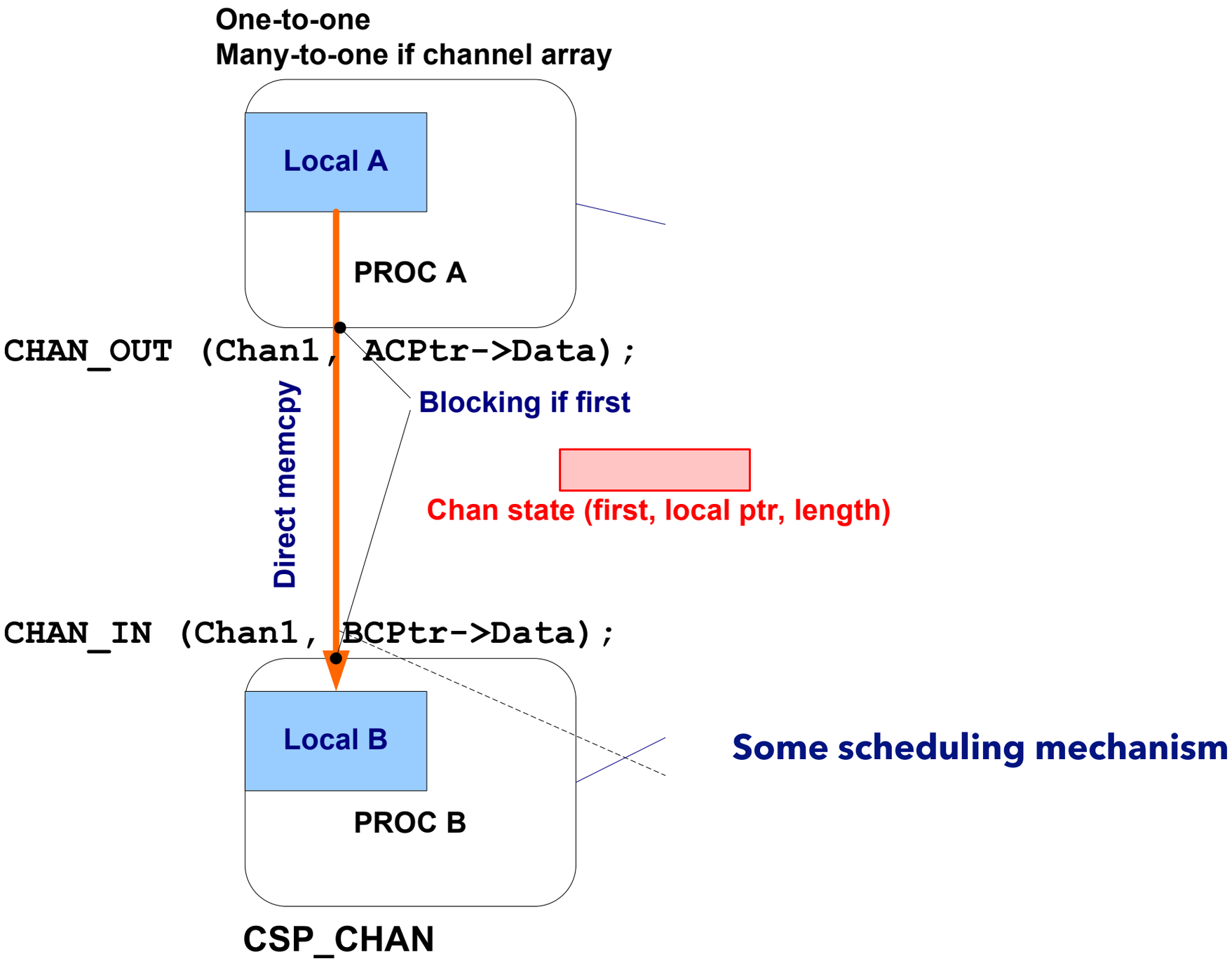
Chan state (first, local ptr, length)

```
CHAN_IN (Chan1, BCPtr->Data);
```

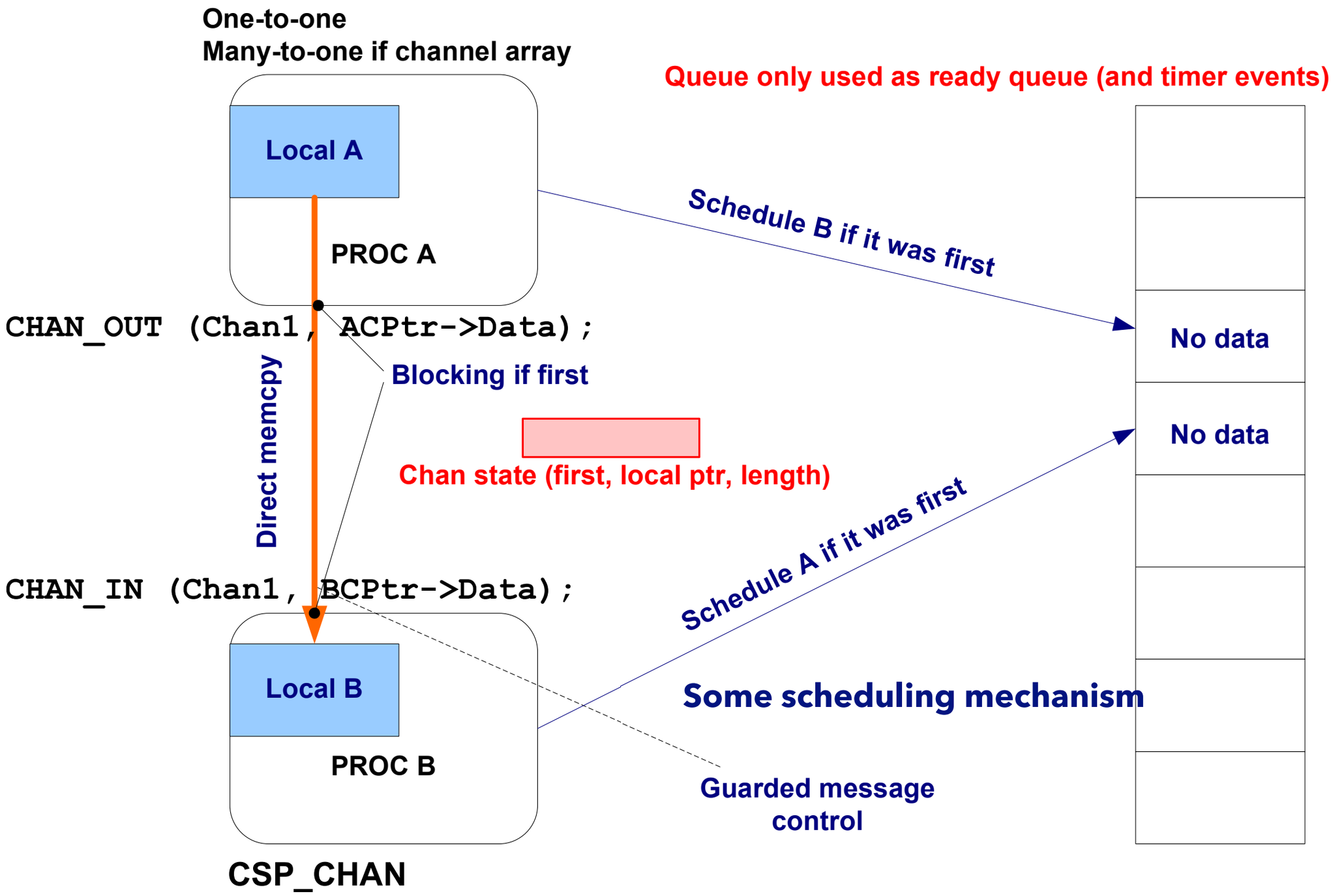
SAFE MEMCPY, NO POINTERS TO SHARED DATA



SAFE MEMCPY, NO POINTERS TO SHARED DATA



SAFE MEMCPY, NO POINTERS TO SHARED DATA



PLAN TO LOSE DATA!

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508
- Standard channel (zero-buffered) just moves data or data ownership

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508
- Standard channel (zero-buffered) just moves data or data ownership
- In Go neither `make(chan int,1)` or `make(chan int)` chans will lose data

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508
- Standard channel (zero-buffered) just moves data or data ownership
- In Go neither `make(chan int,1)` or `make(chan int)` chans will lose data
  - Goroutine will block until ready (or get an «ok/err» if you need to)

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508
- Standard channel (zero-buffered) just moves data or data ownership
- In Go neither `make(chan int,1)` or `make(chan int)` chans will lose data
  - Goroutine will block until ready (or get an «ok/err» if you need to)
- But runtimes/schedulers will, if you use asynch messaging uncritically sooner or later lose data if sender talks too much

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508
- Standard channel (zero-buffered) just moves data or data ownership
- In Go neither `make(chan int,1)` or `make(chan int)` chans will lose data
  - Goroutine will block until ready (or get an «ok/err» if you need to)
- But runtimes/schedulers will, if you use asynch messaging uncritically sooner or later lose data if sender talks too much
  - Buffer full when no more memory: *restart*!😱

I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- Plan to lose data, at application level (=in your control)
  - At «the edges» (retransmit?, error report?)
- More and more applications are «Safety critical»
  - If not necessarily requiring IEC 61508
- Standard channel (zero-buffered) just moves data or data ownership
- In Go neither `make(chan int,1)` or `make(chan int)` chans will lose data
  - Goroutine will block until ready (or get an «ok/err» if you need to)
- But runtimes/schedulers will, if you use asynch messaging uncritically sooner or later lose data if sender talks too much
  - Buffer full when no more memory: *restart*!😱
  - Therefore:

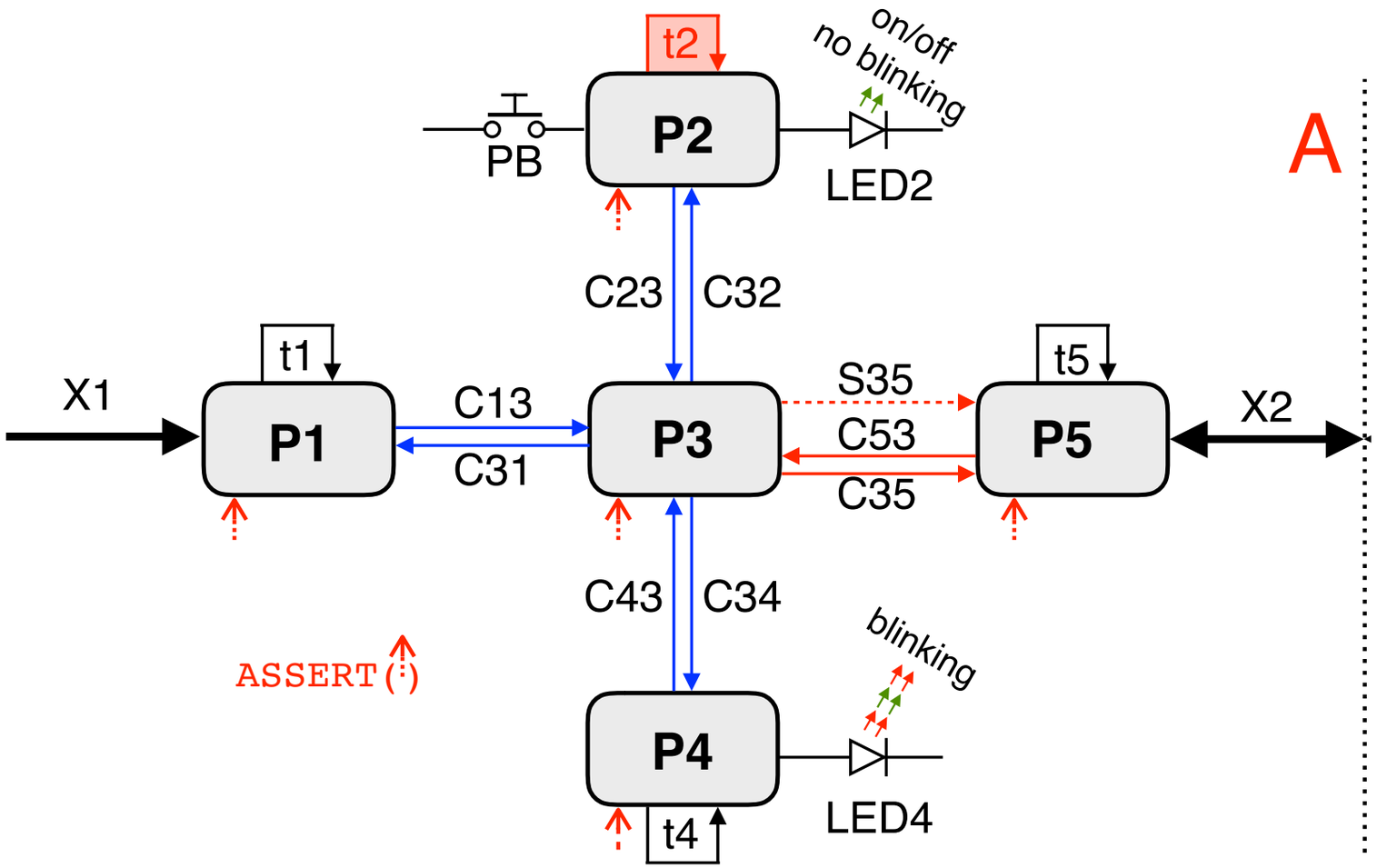
PAUSE!

From a blog note

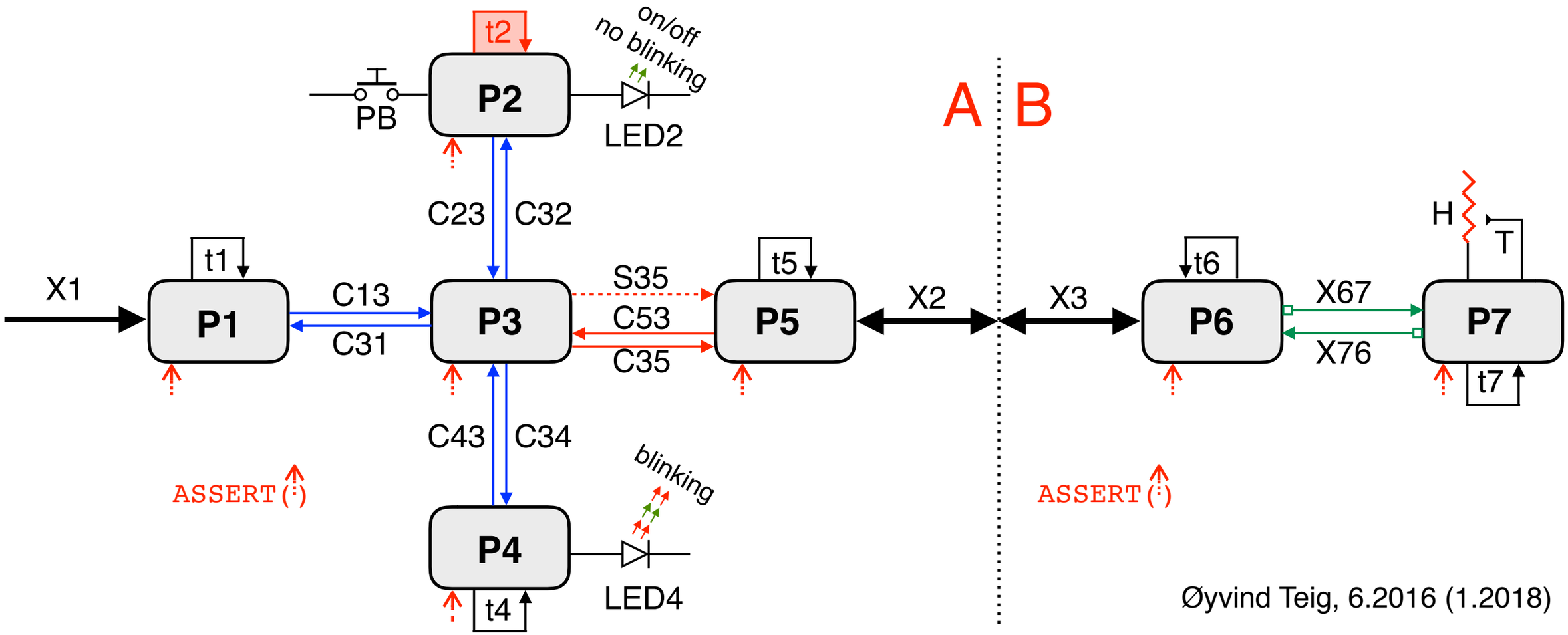
«…-delay/timeout-pollRx» IS NOT A CONTRACT!

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!

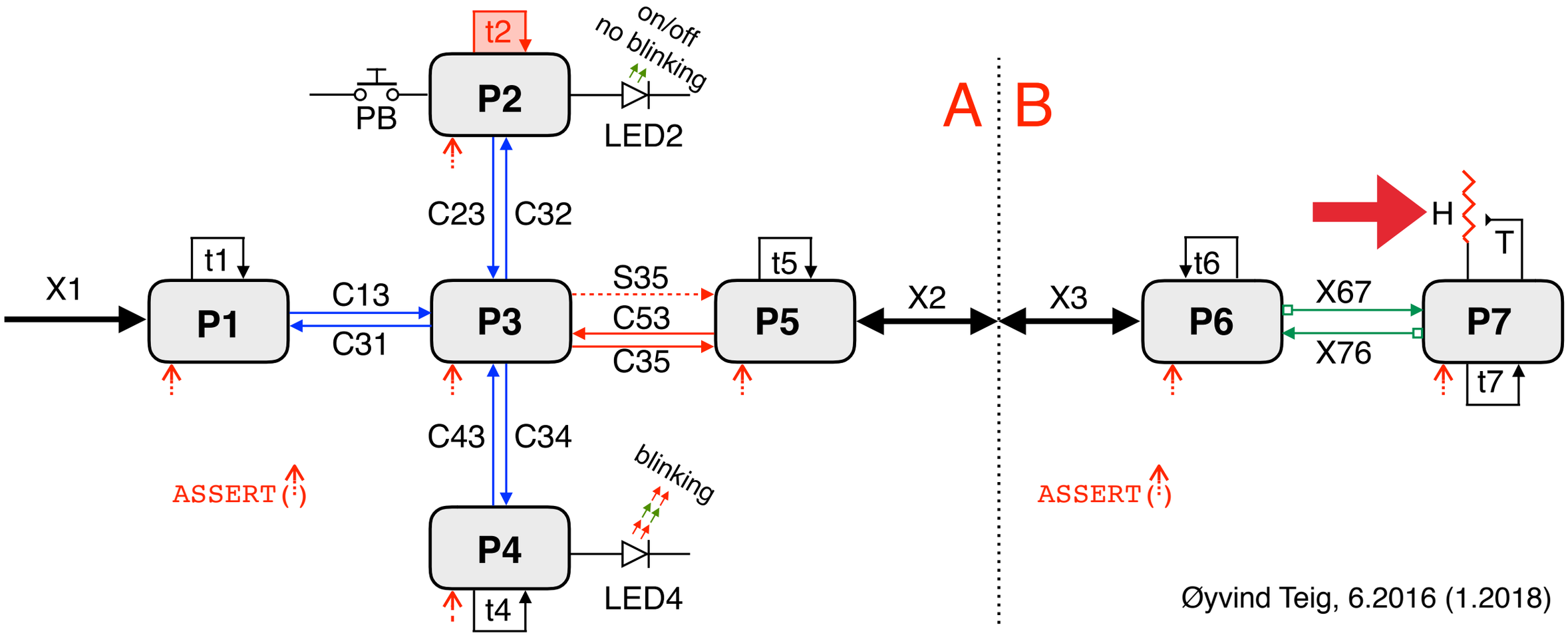


«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



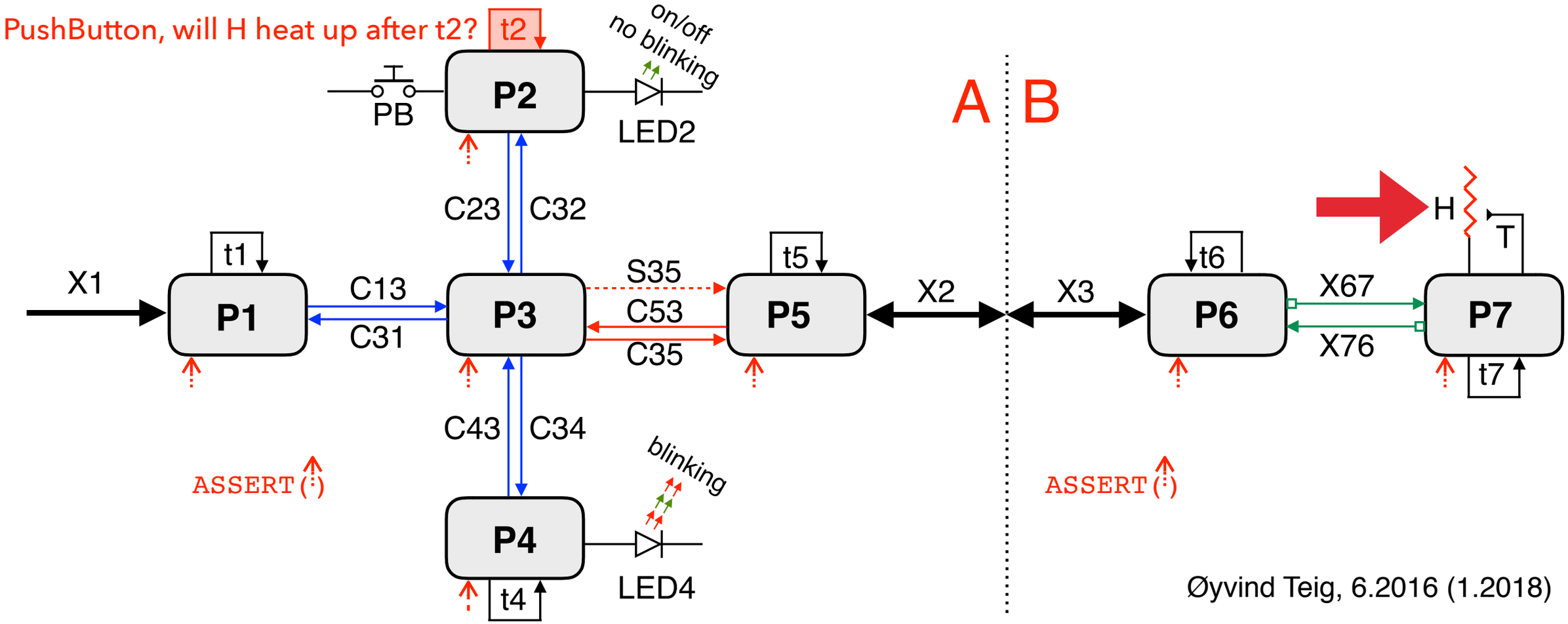
Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



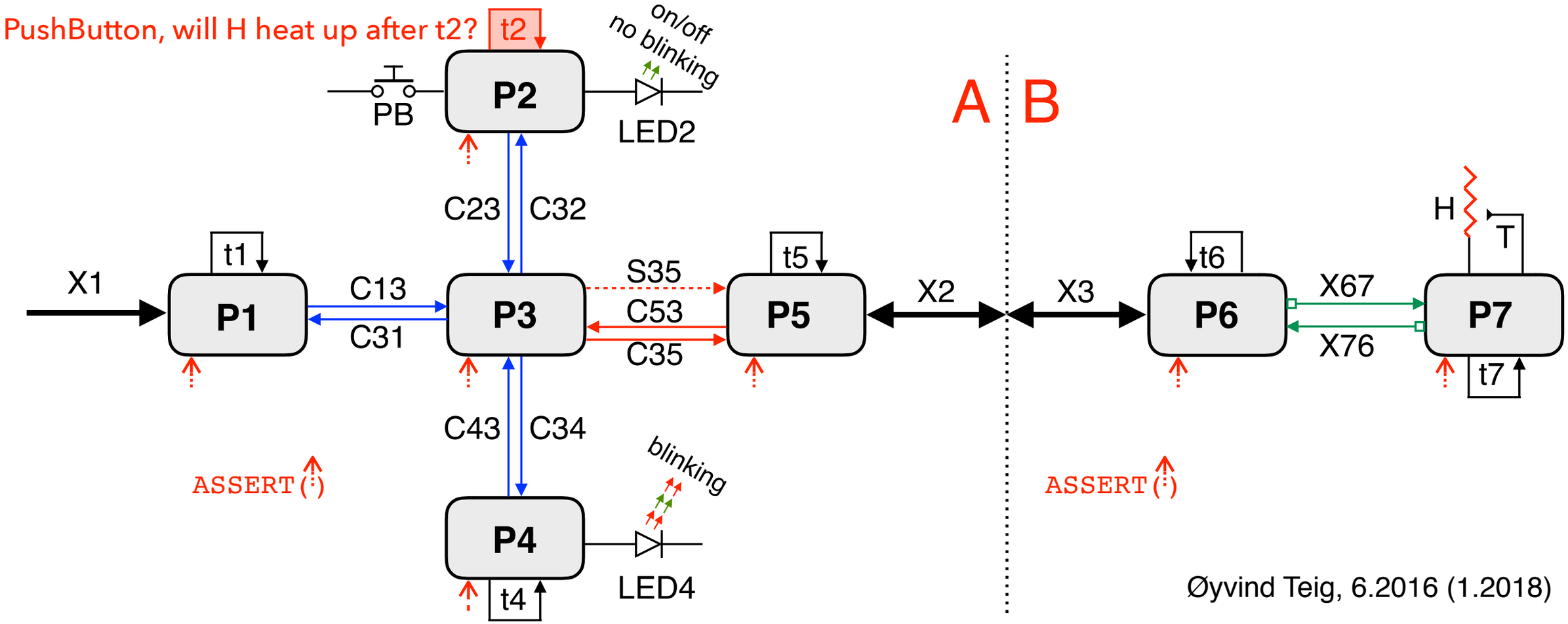
Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



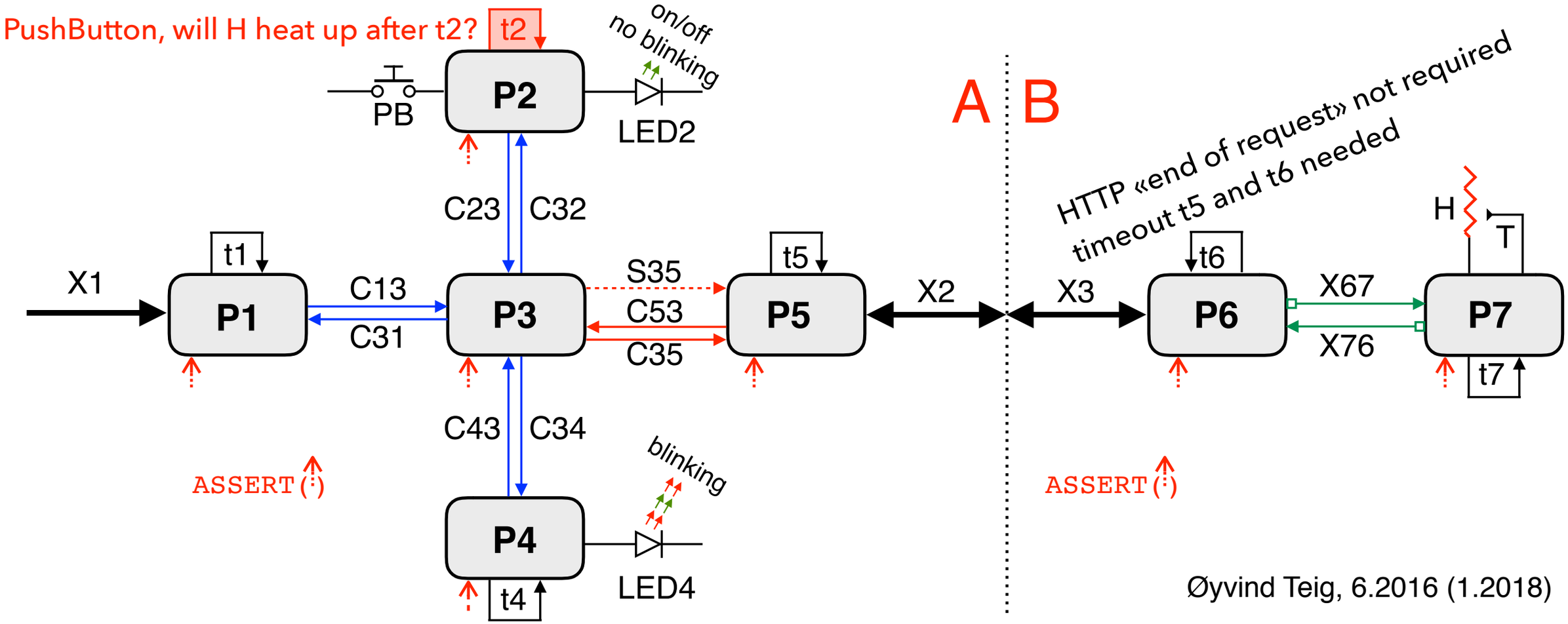
Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



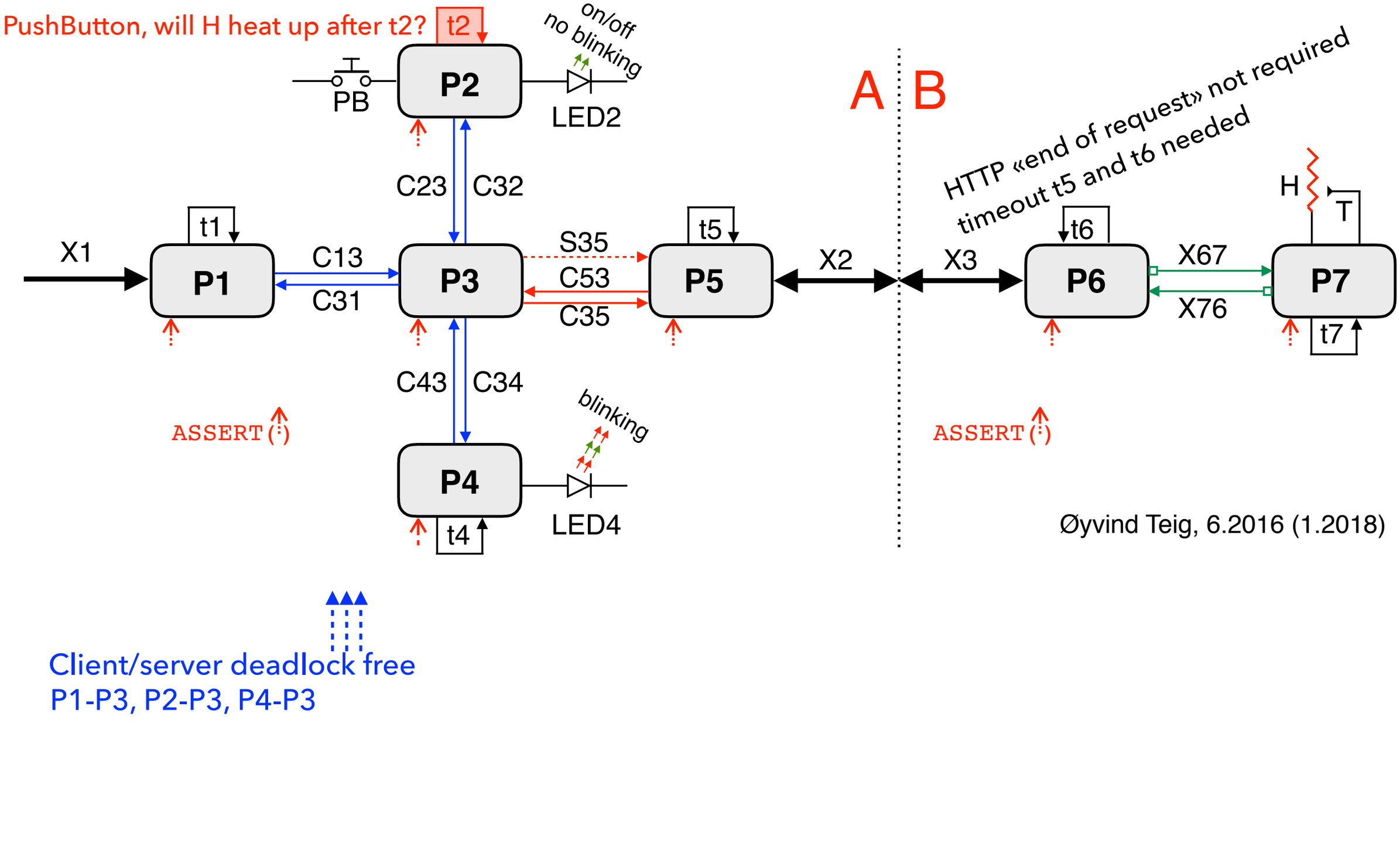
Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



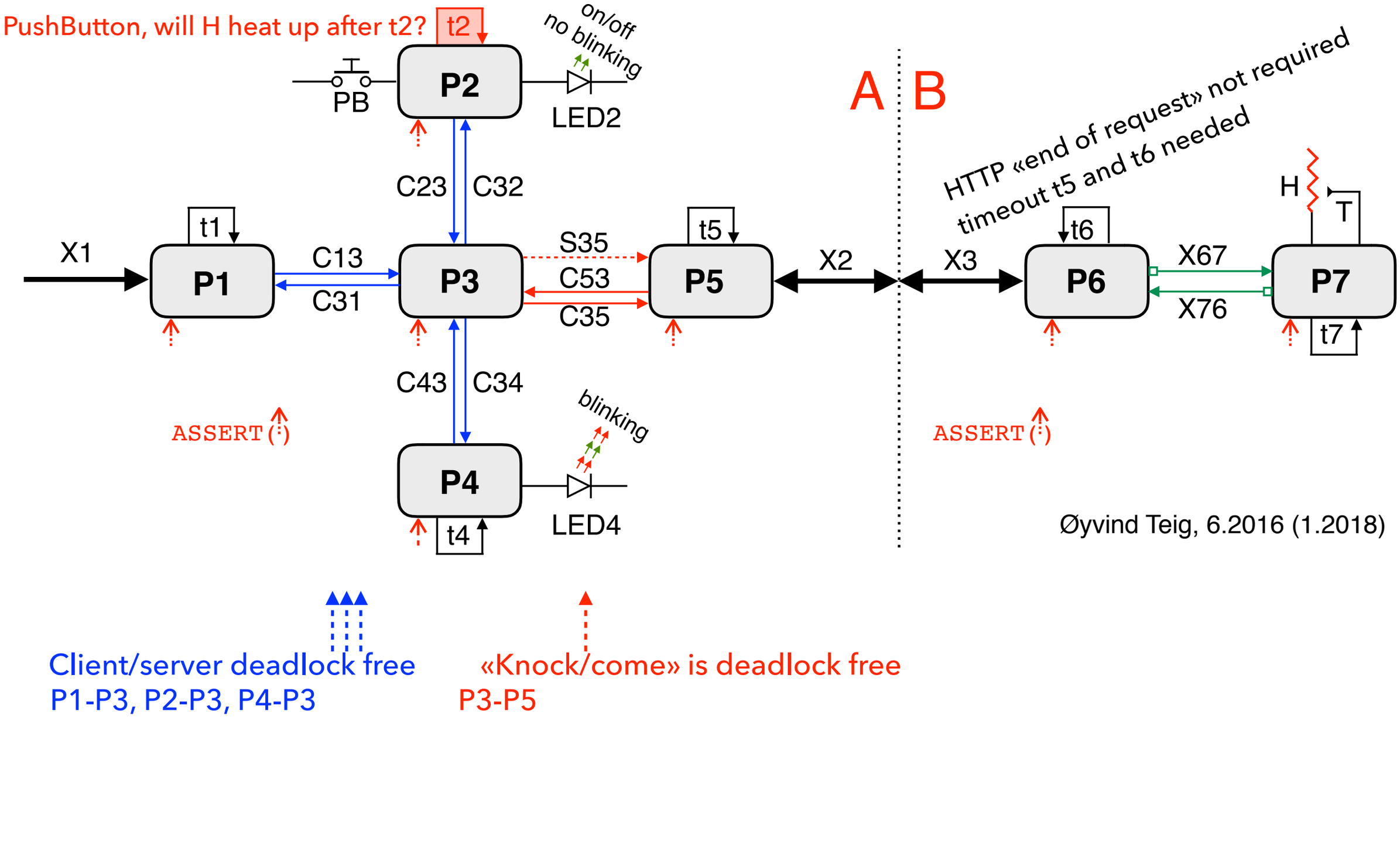
Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



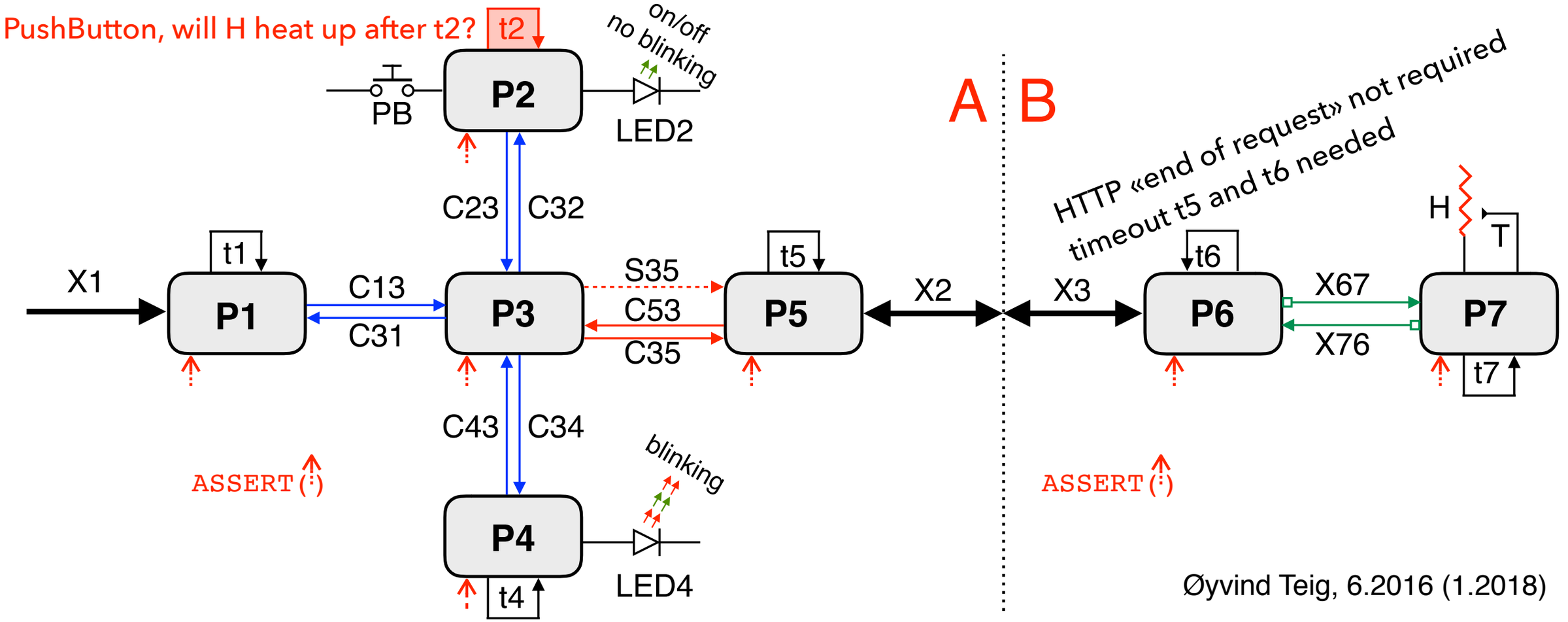
Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



Øyvind Teig, 6.2016 (1.2018)

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!

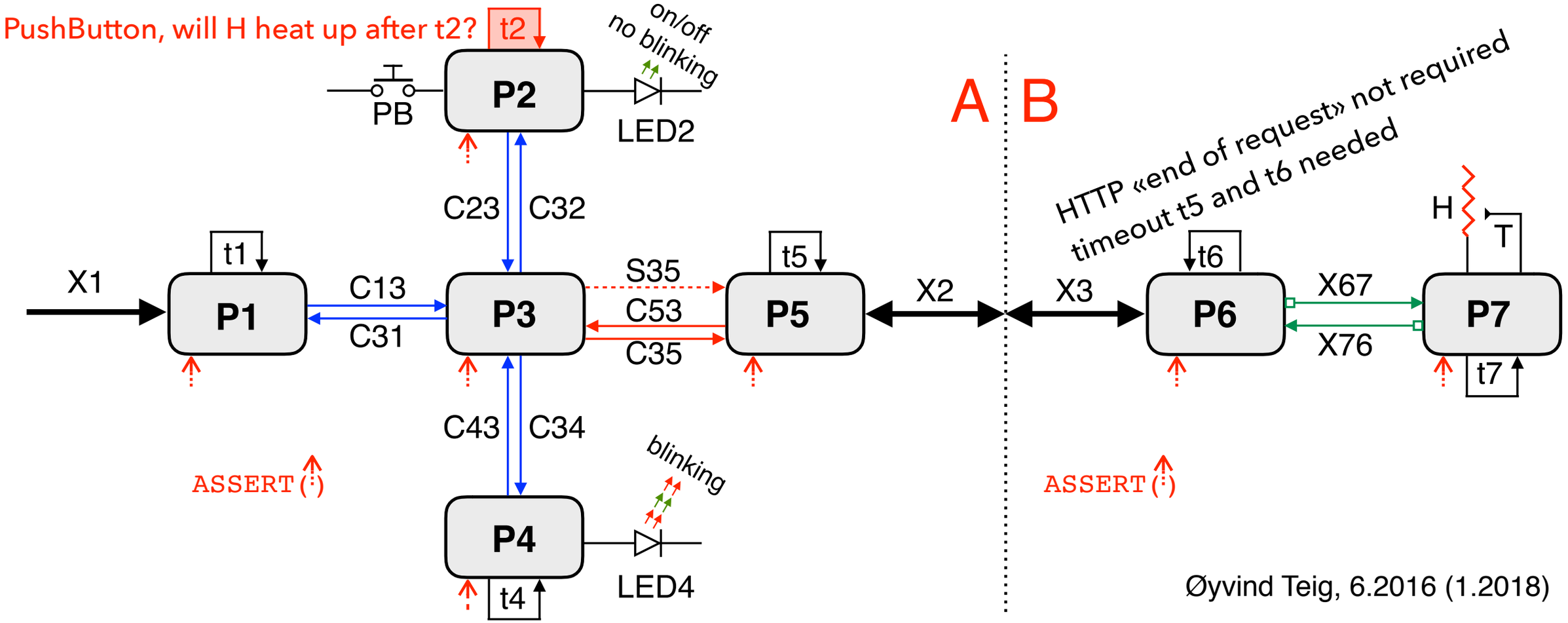


Client/server deadlock free
P1-P3, P2-P3, P4-P3

«Knock/come» is deadlock free
P3-P5

XCHAN is deadlock free [2]
P6-P7

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!



Client/server deadlock free
P1-P3, P2-P3, P4-P3

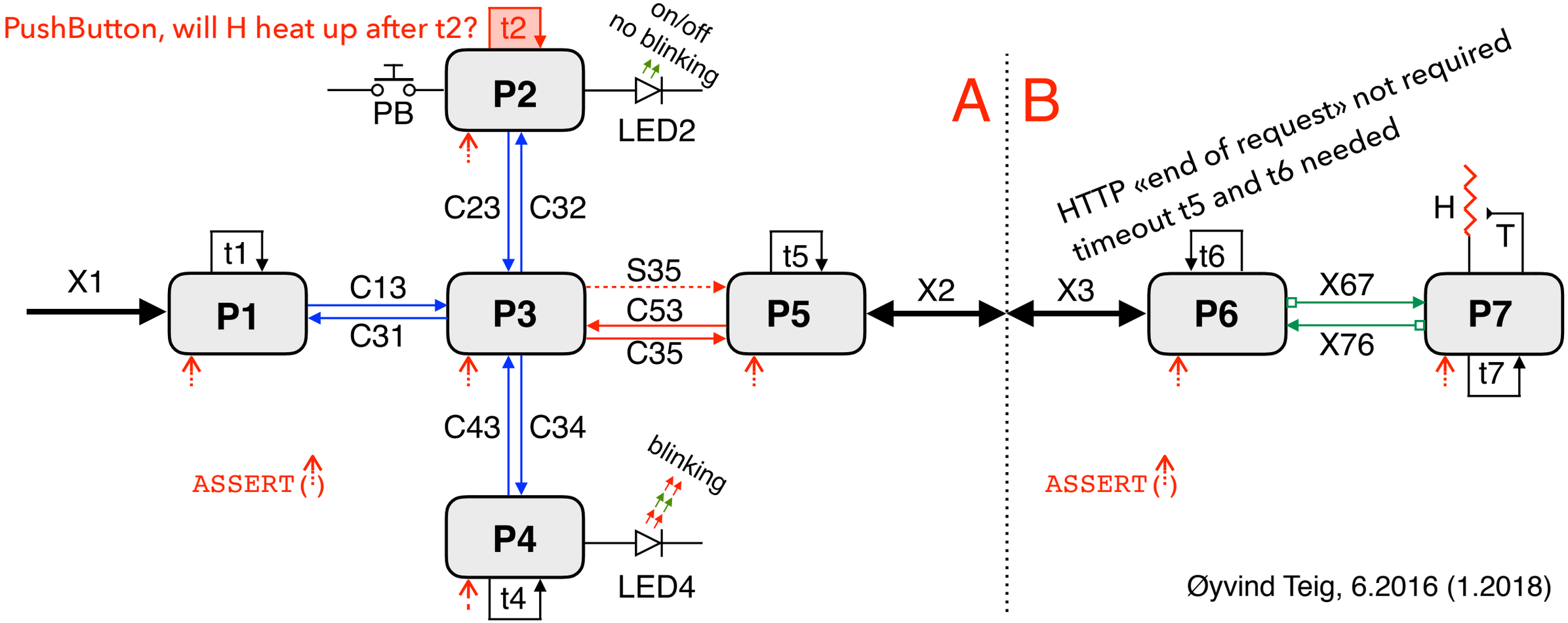
«Knock/come» is deadlock free
P3-P5

XCHAN is deadlock free [2]
P6-P7

No timeout between internal processes! If timeouts: mess guaranteed!

«Tx-delay/timeout-pollRx» IS NOT A CONTRACT!

http://www.teigfam.net/oyvind/home/technology/128-timing-out-design-by-contract-with-a-stopwatch/

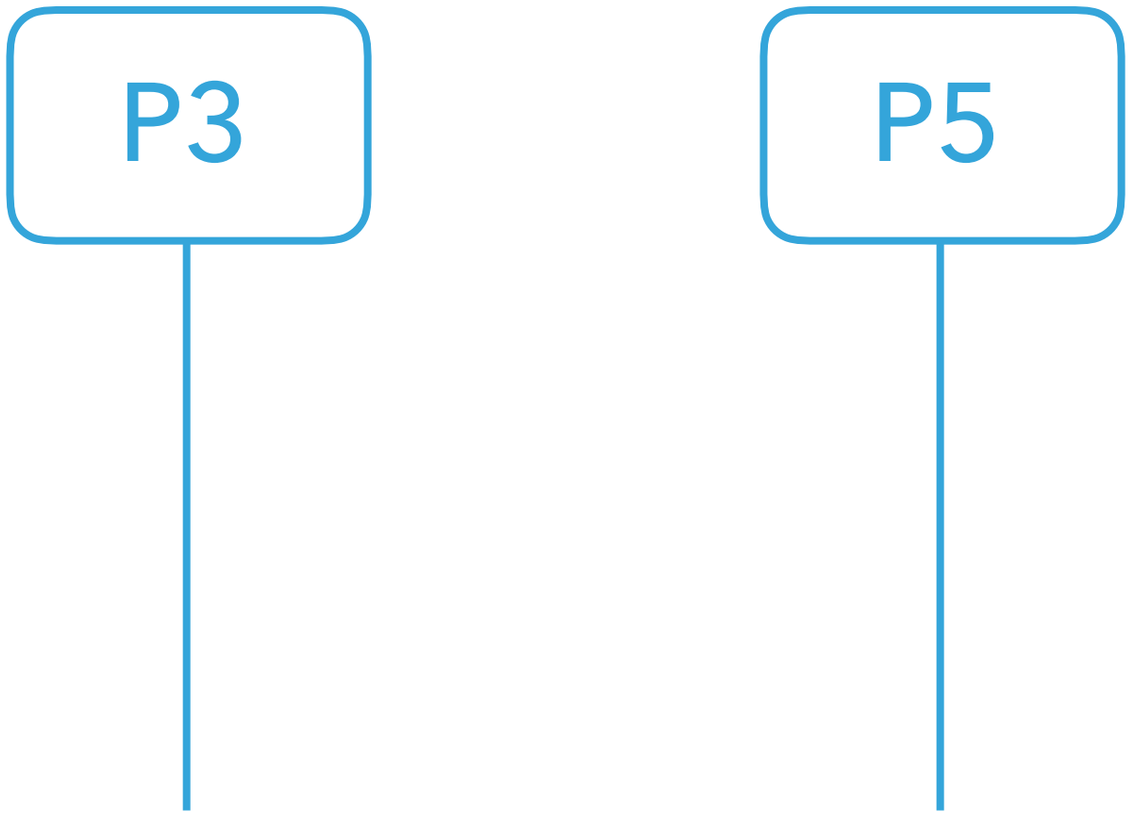


Client/server deadlock free
P1-P3, P2-P3, P4-P3

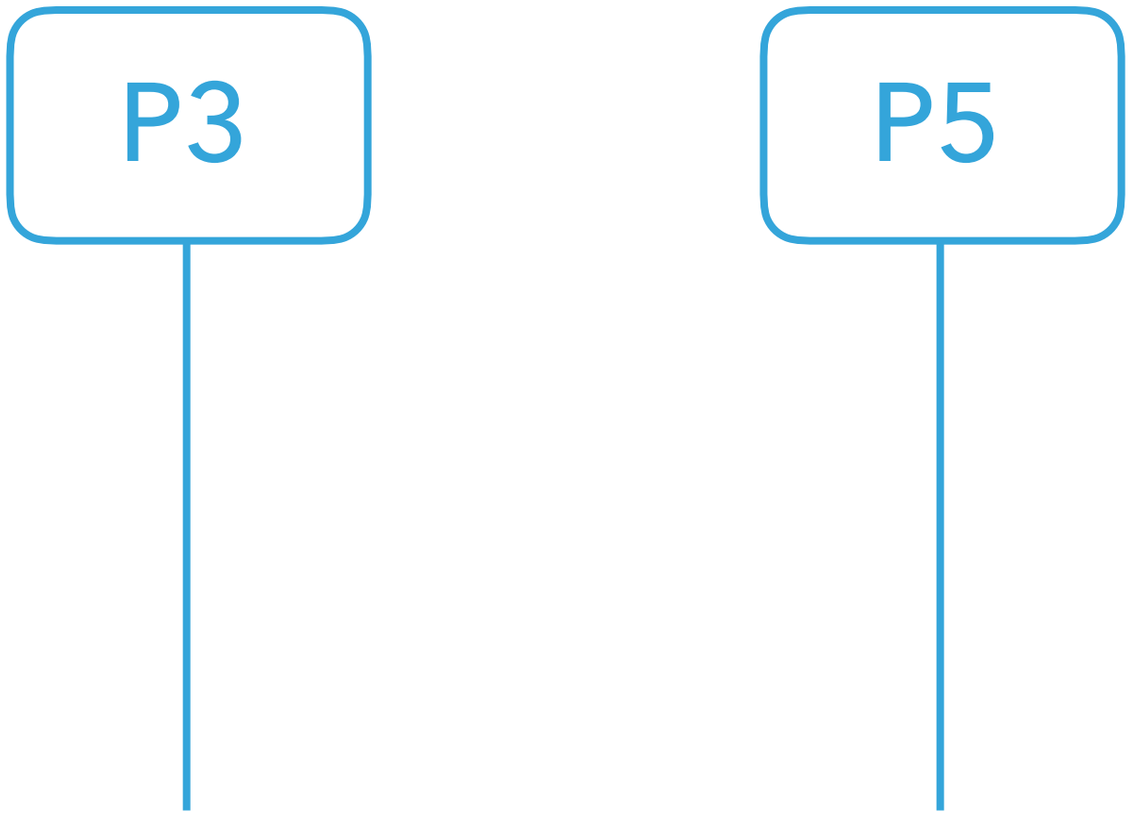
«Knock/come» is deadlock free
P3-P5

XCHAN is deadlock free [2]
P6-P7

No timeout between internal processes! If timeouts: mess guaranteed!

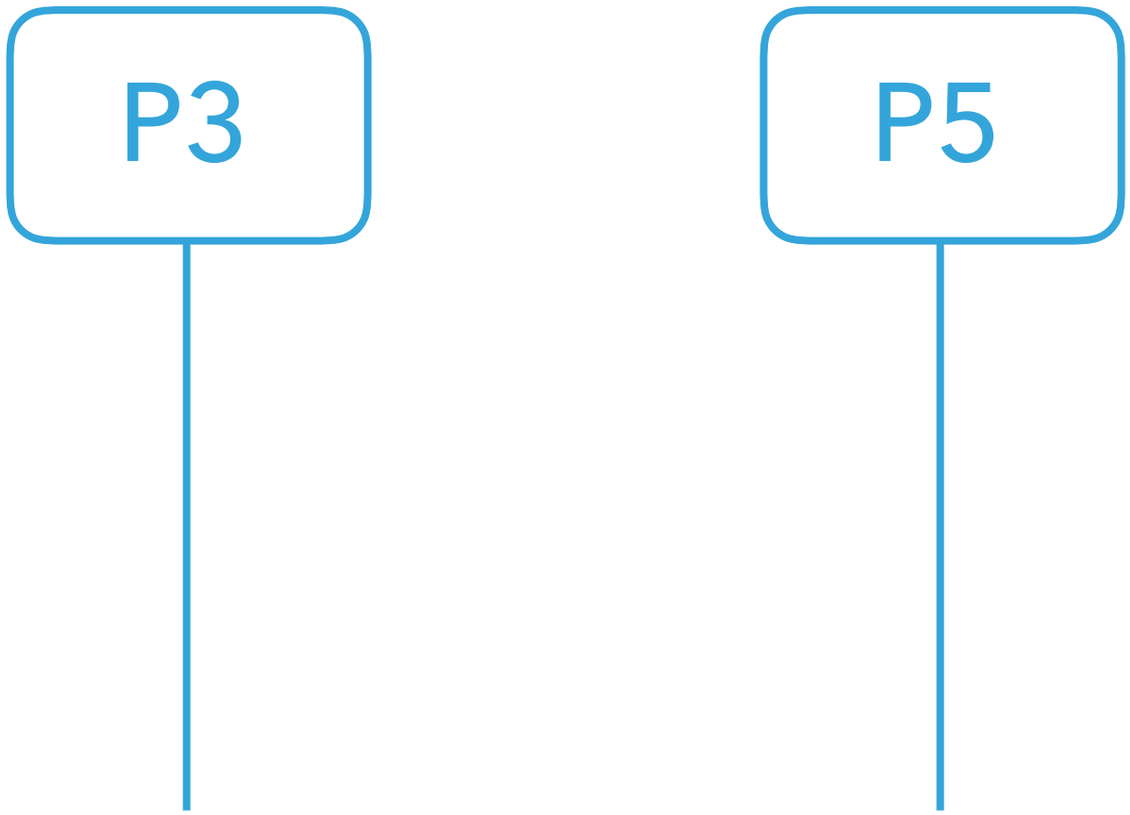


KNOCK-COME, THEN DATA



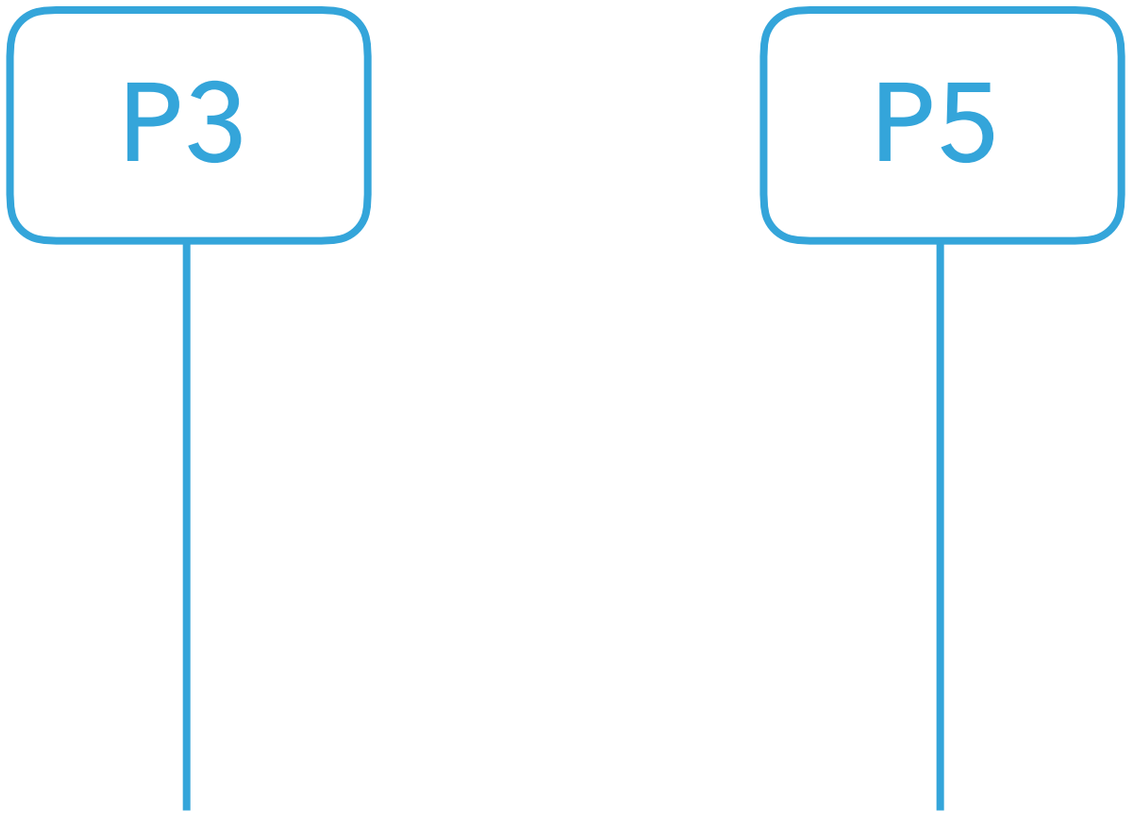
KNOCK-COME, THEN DATA

- Deadlock free communication pattern



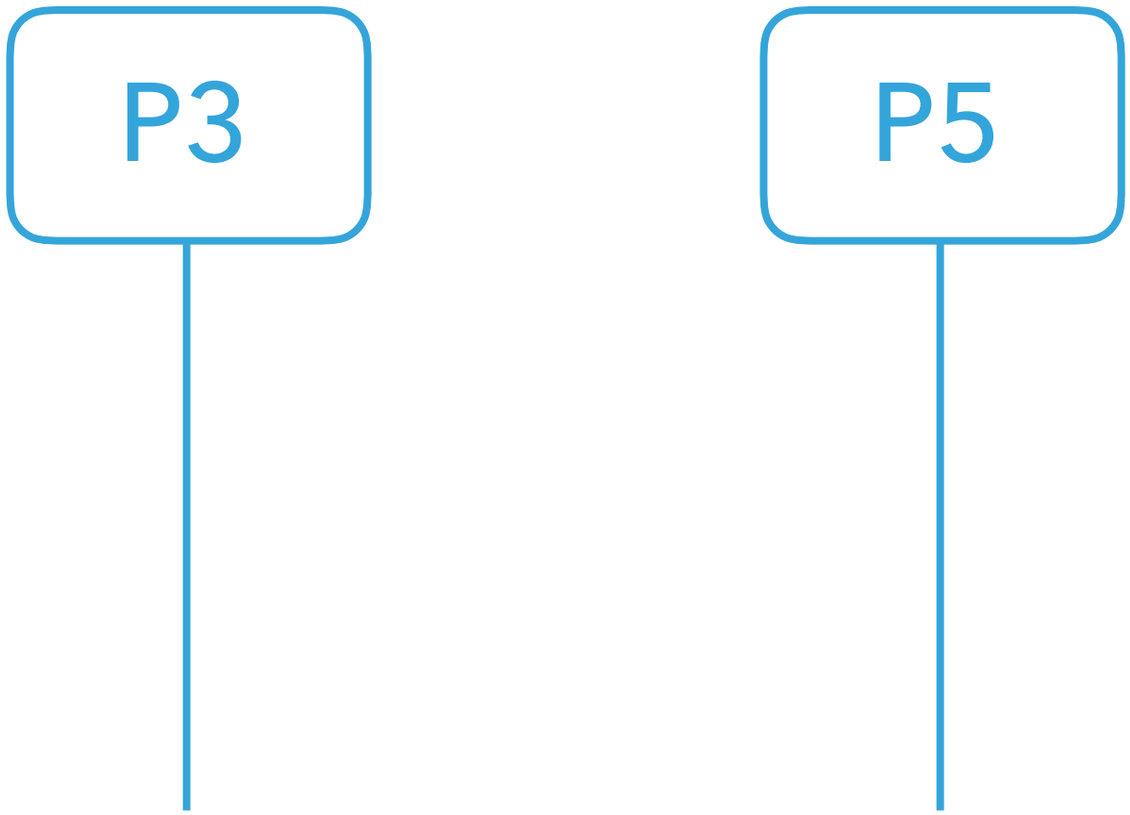
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions



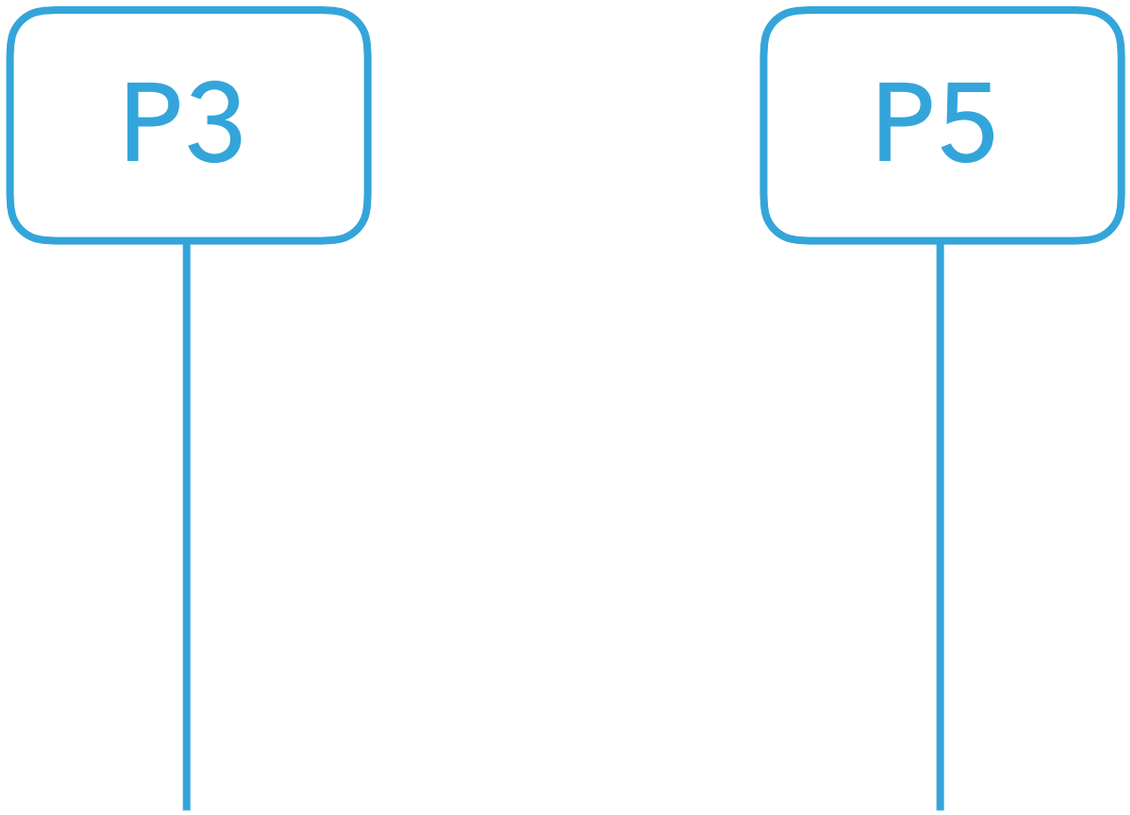
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time



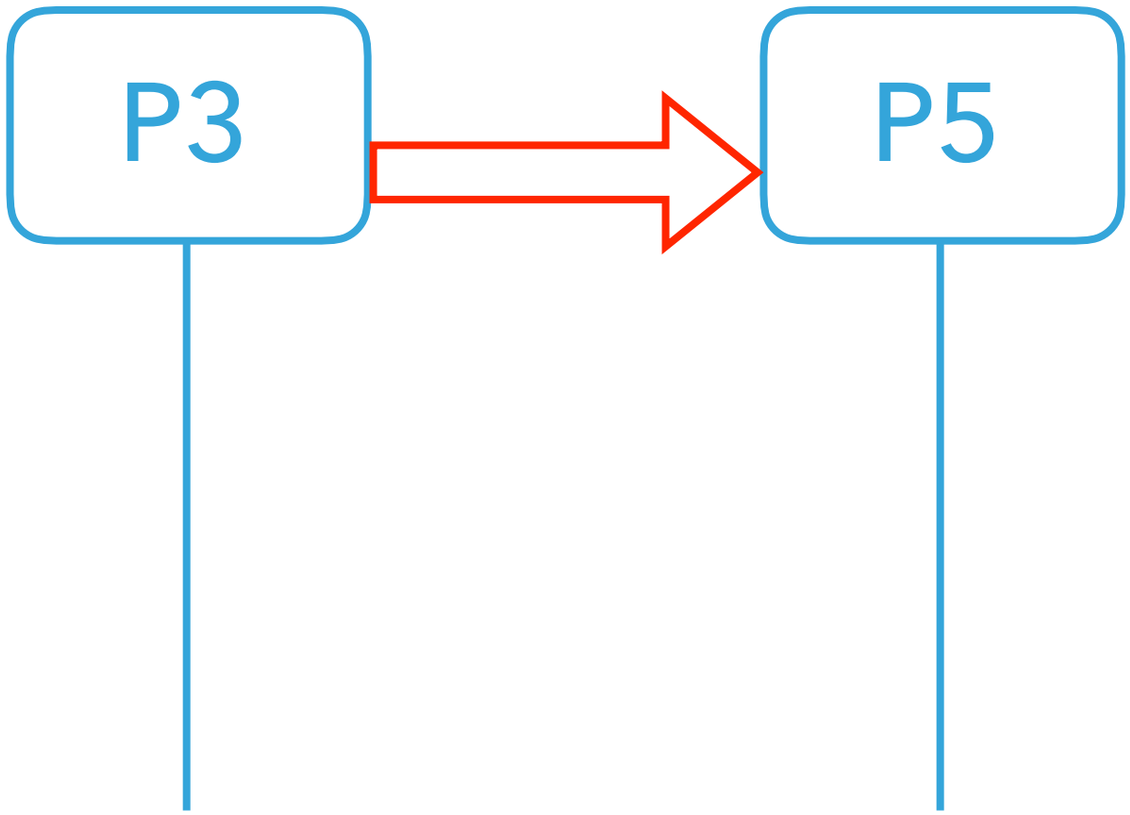
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»



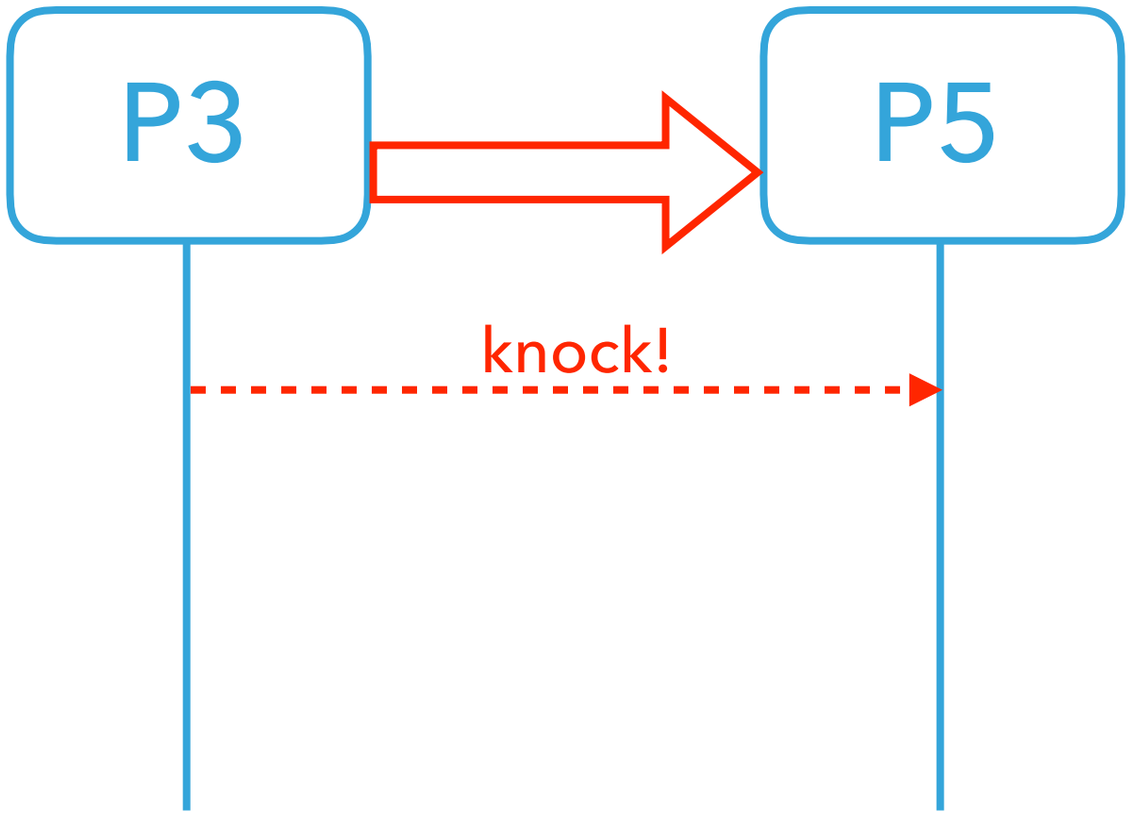
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



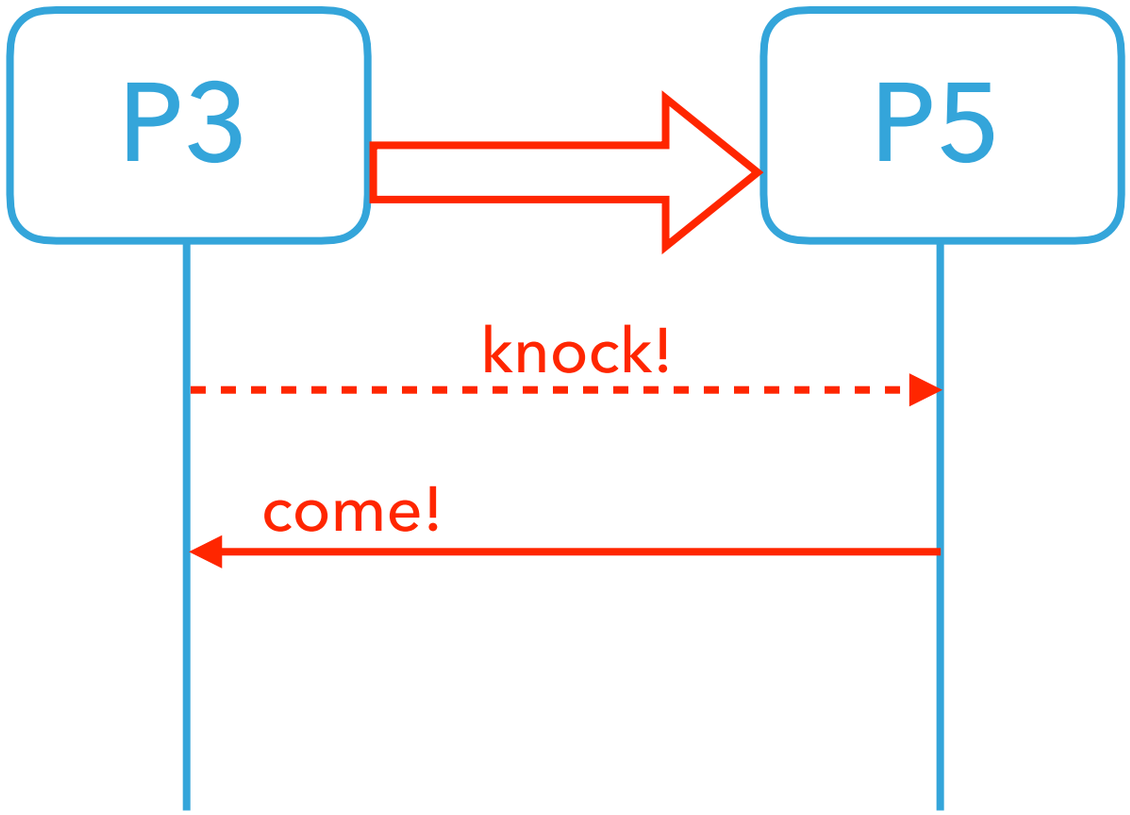
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



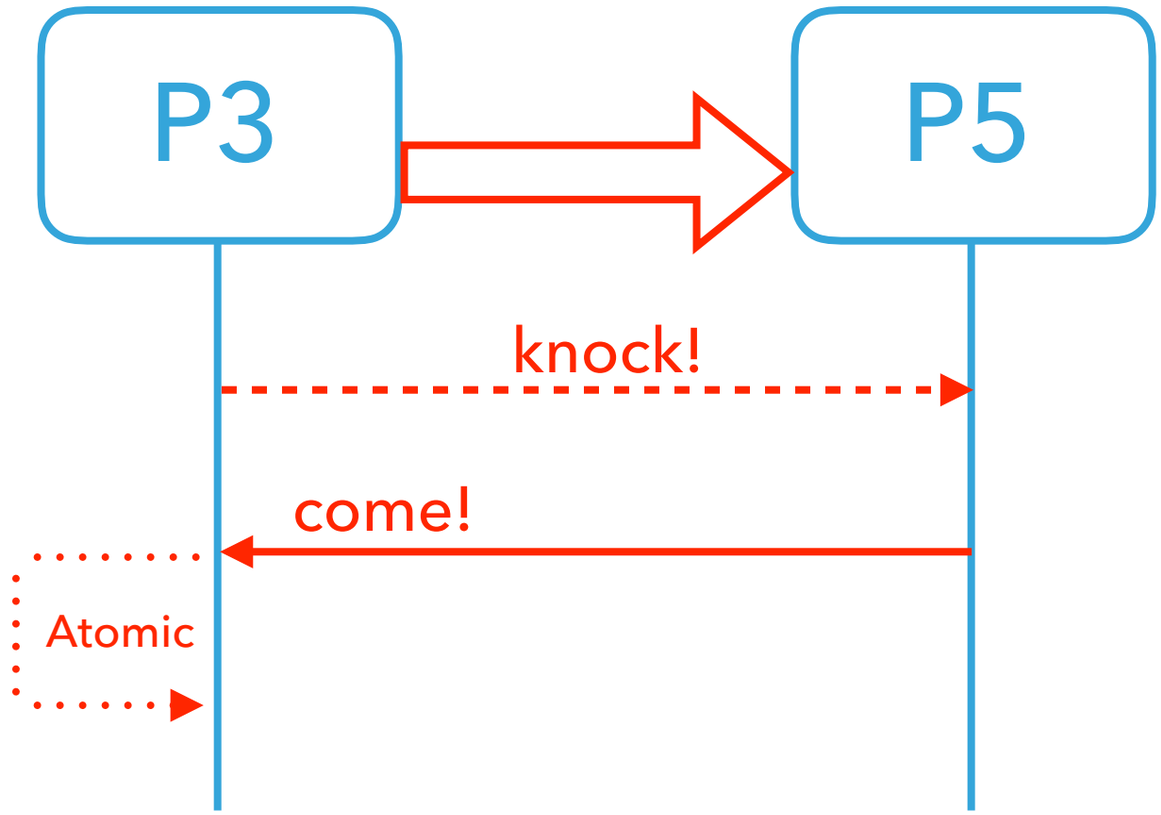
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



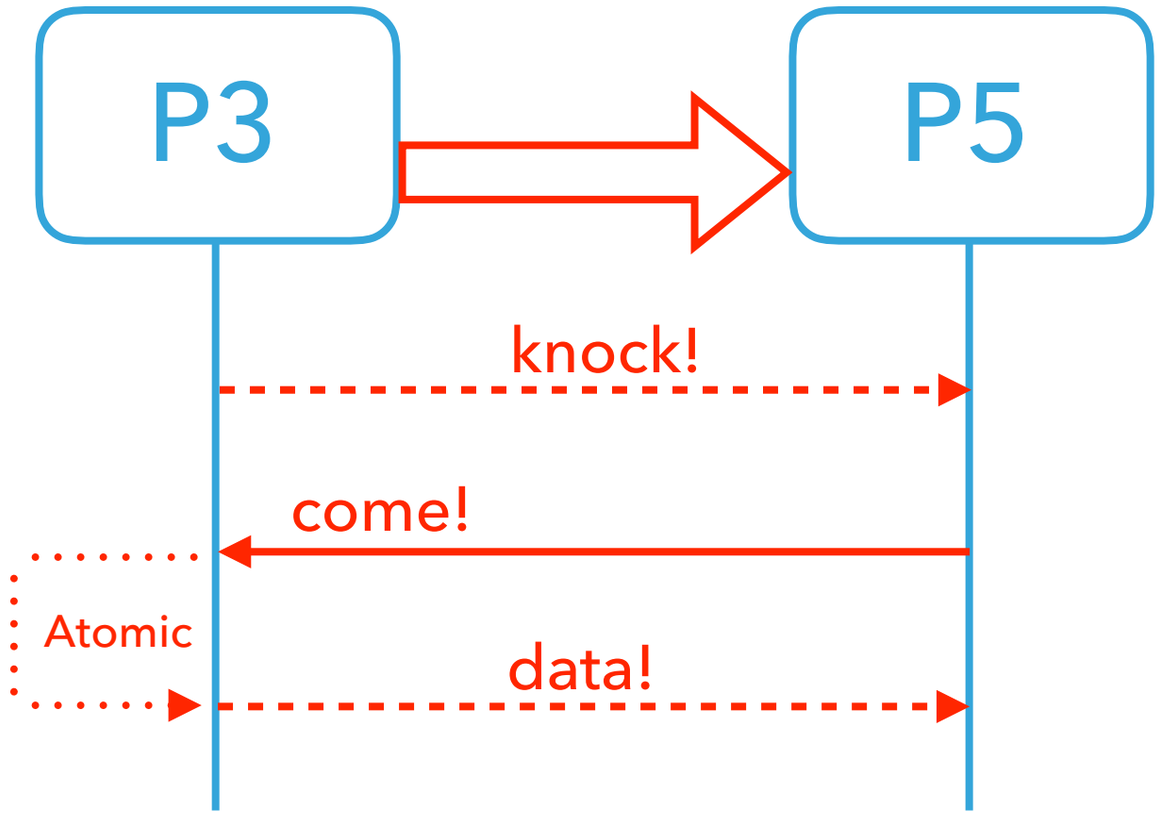
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



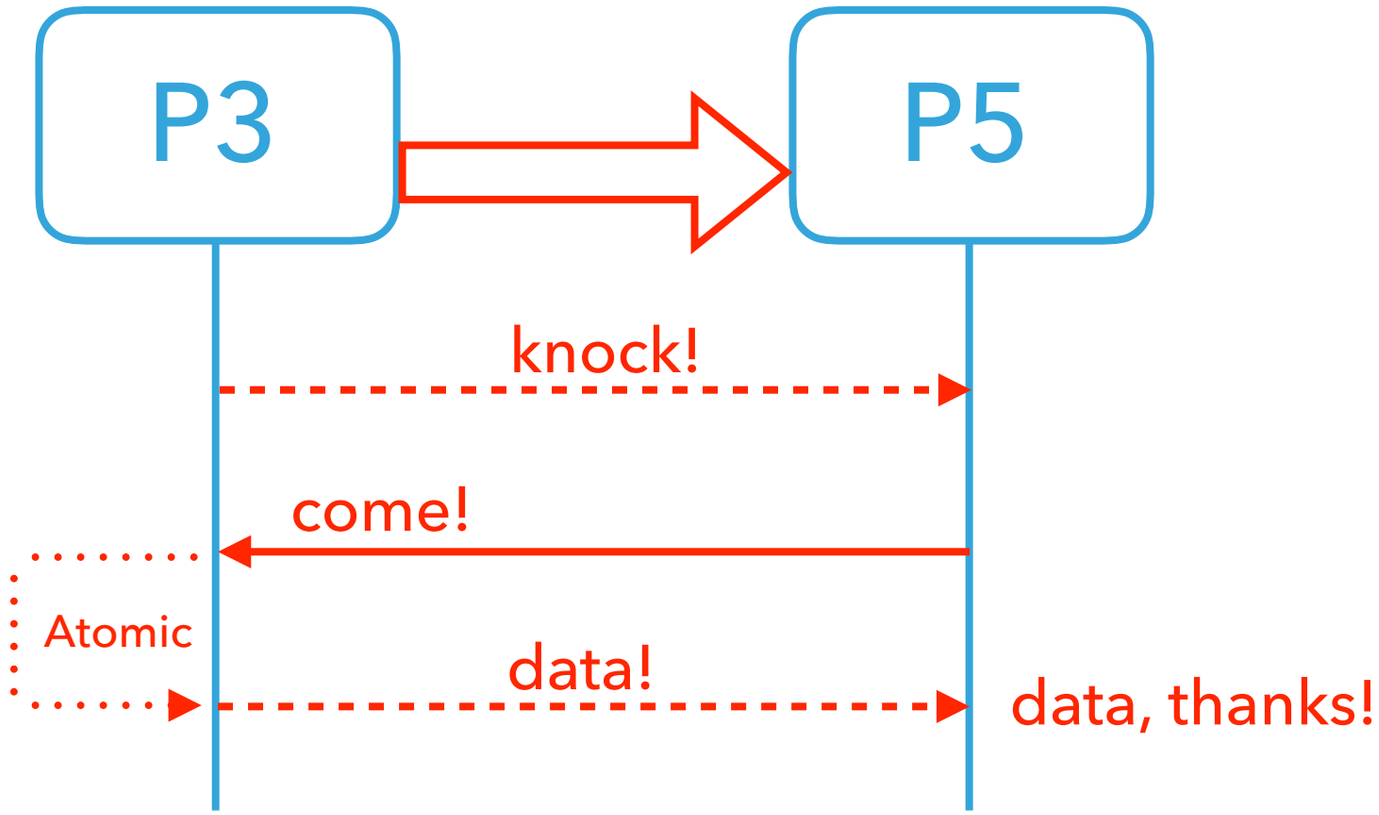
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



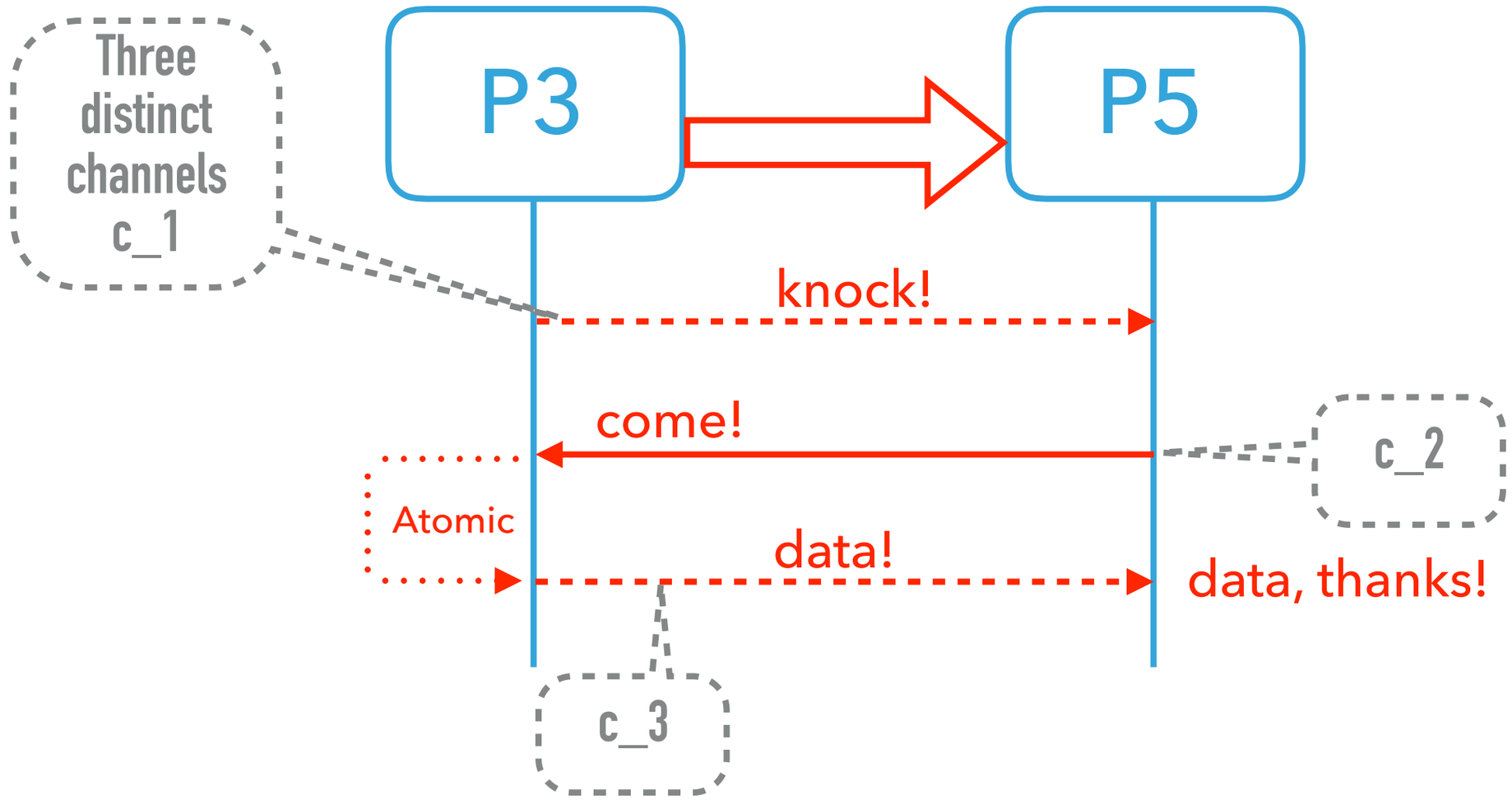
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



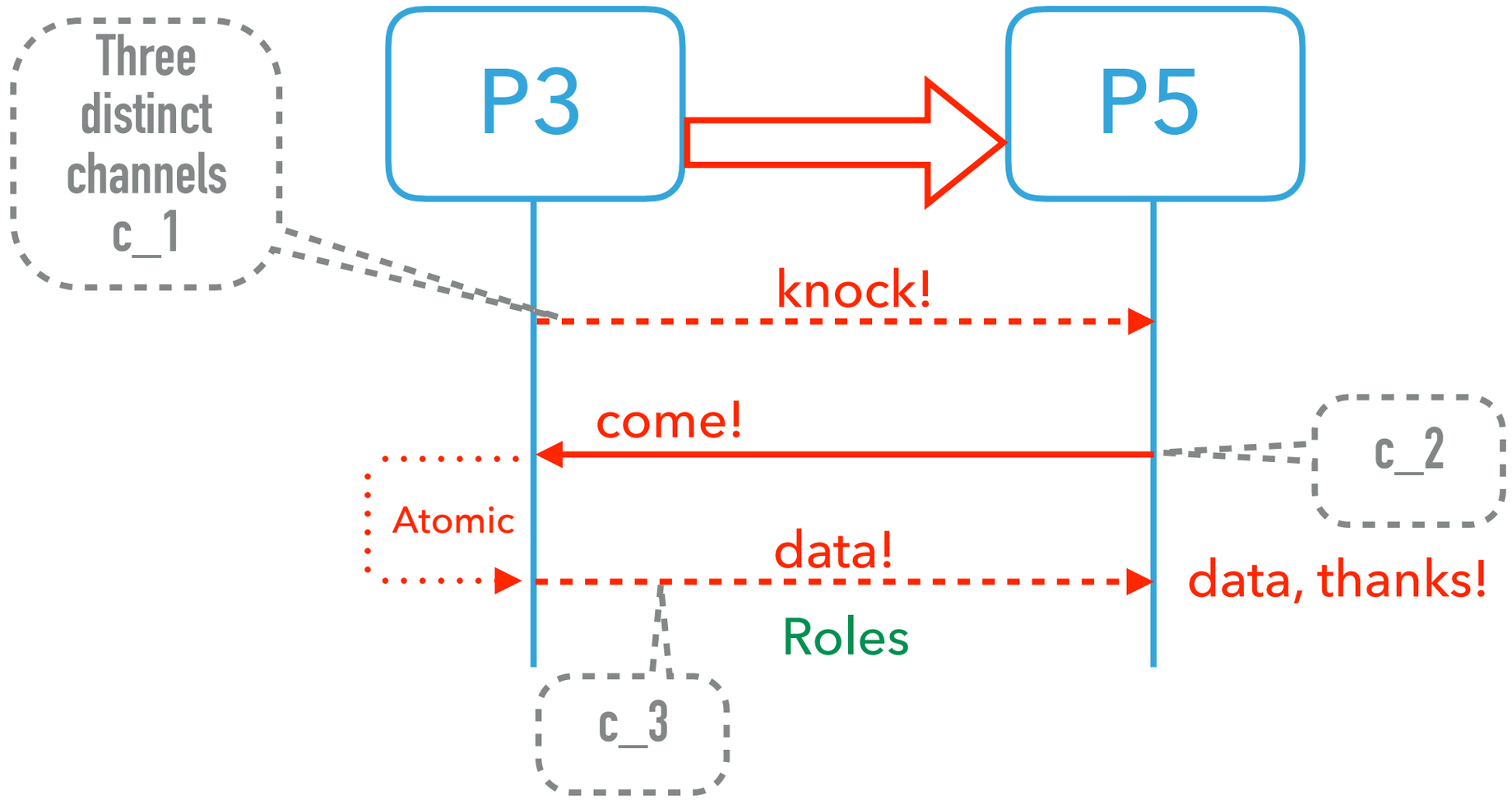
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



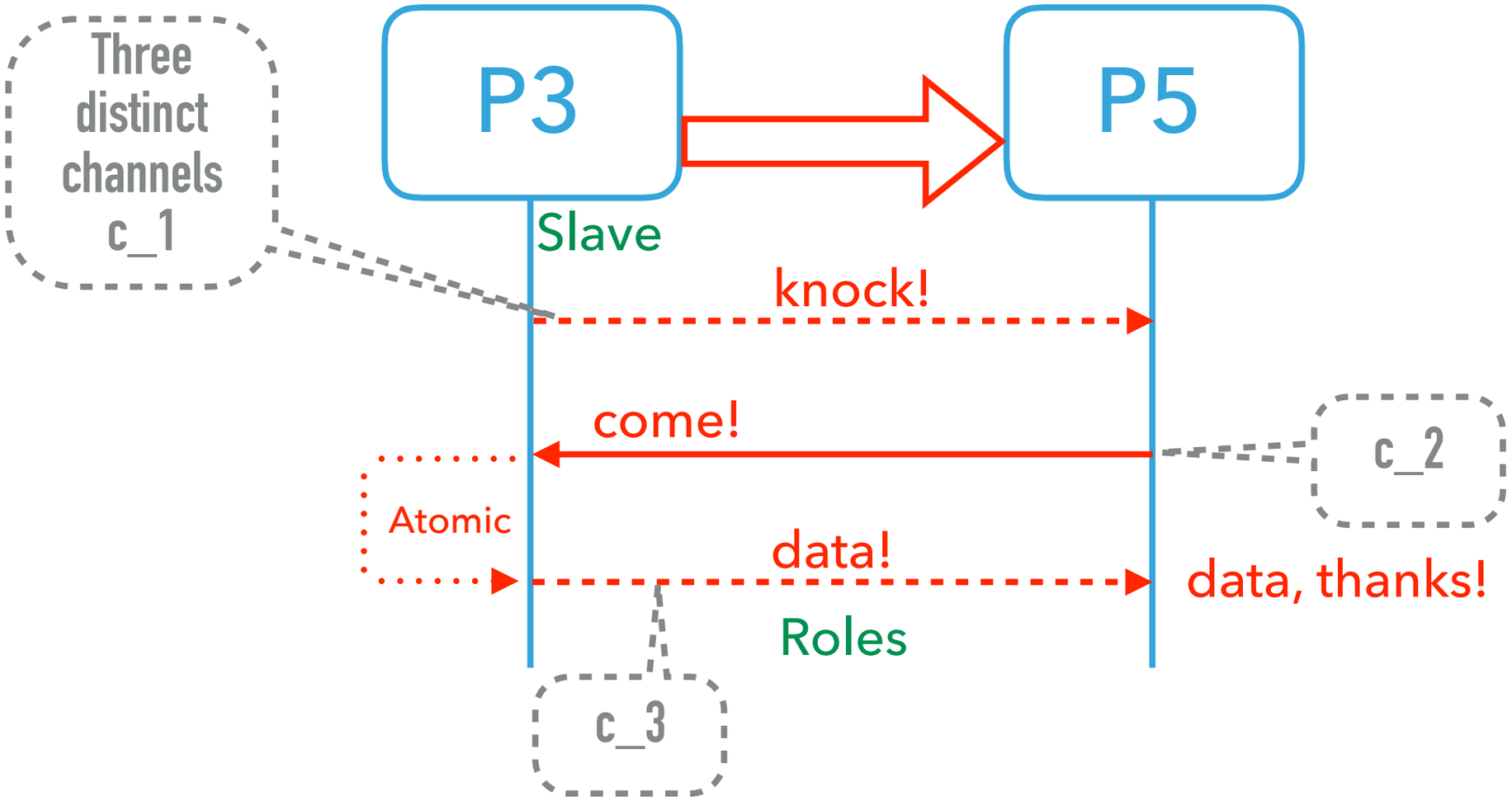
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block



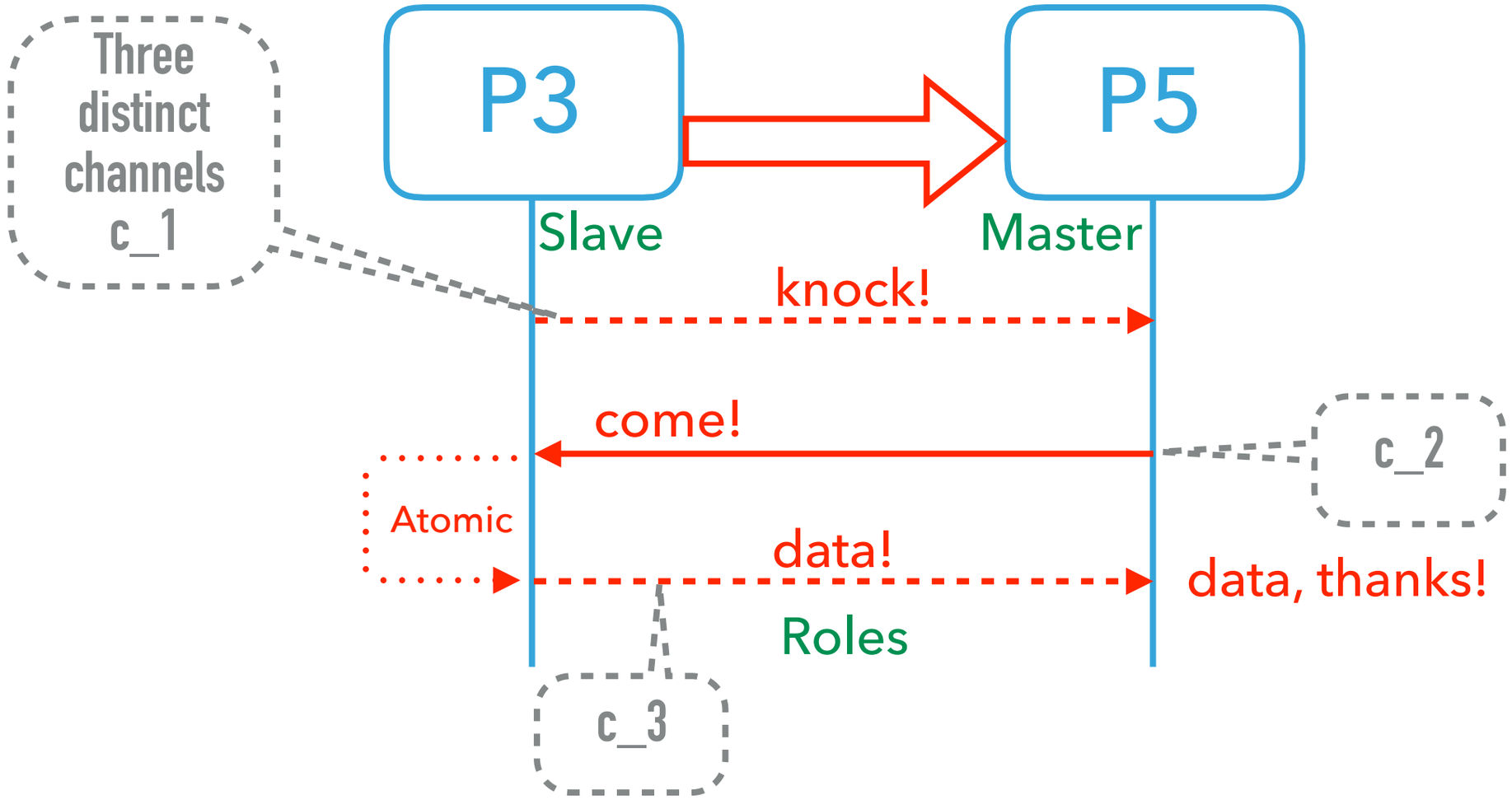
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block



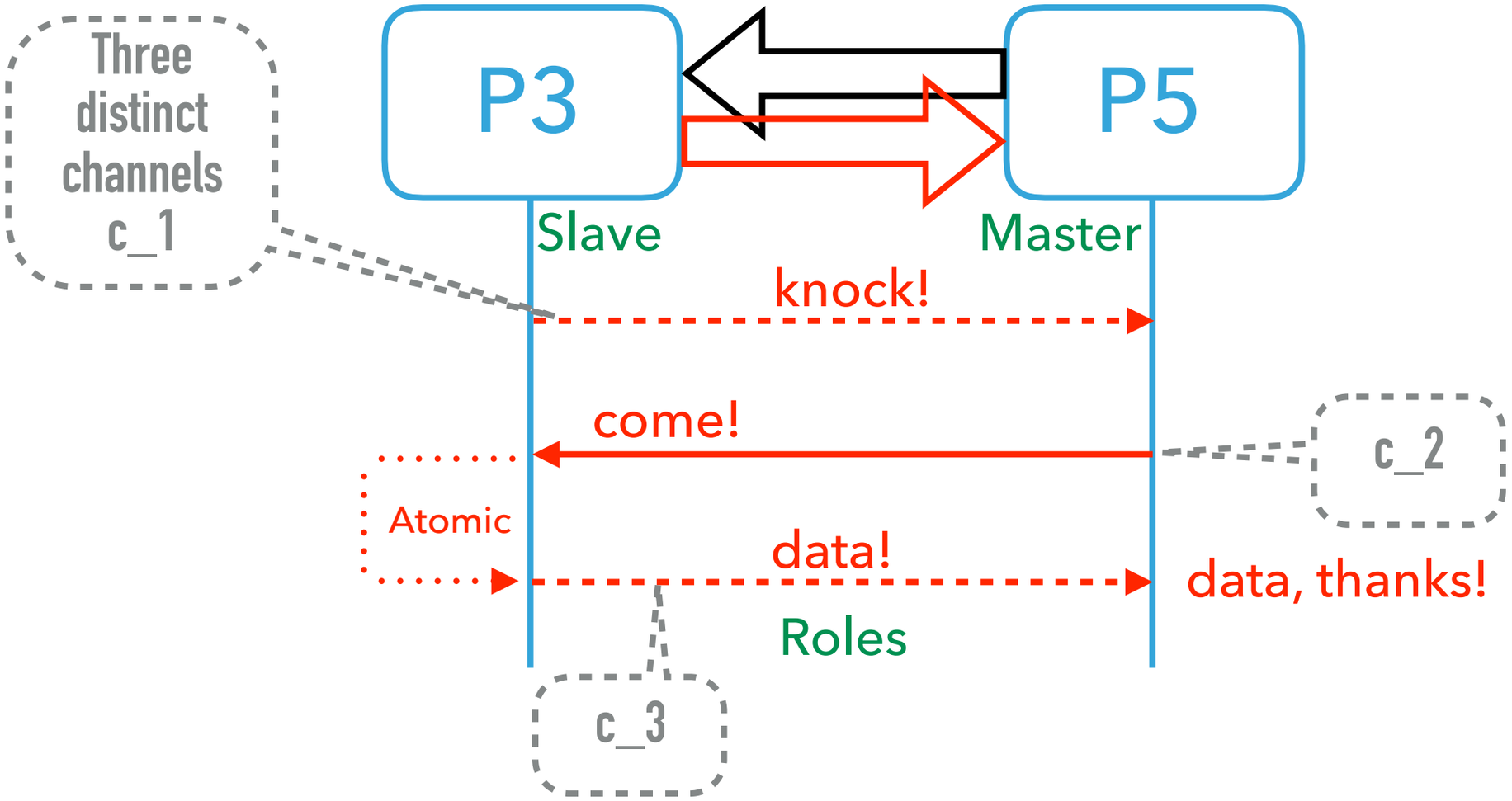
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block



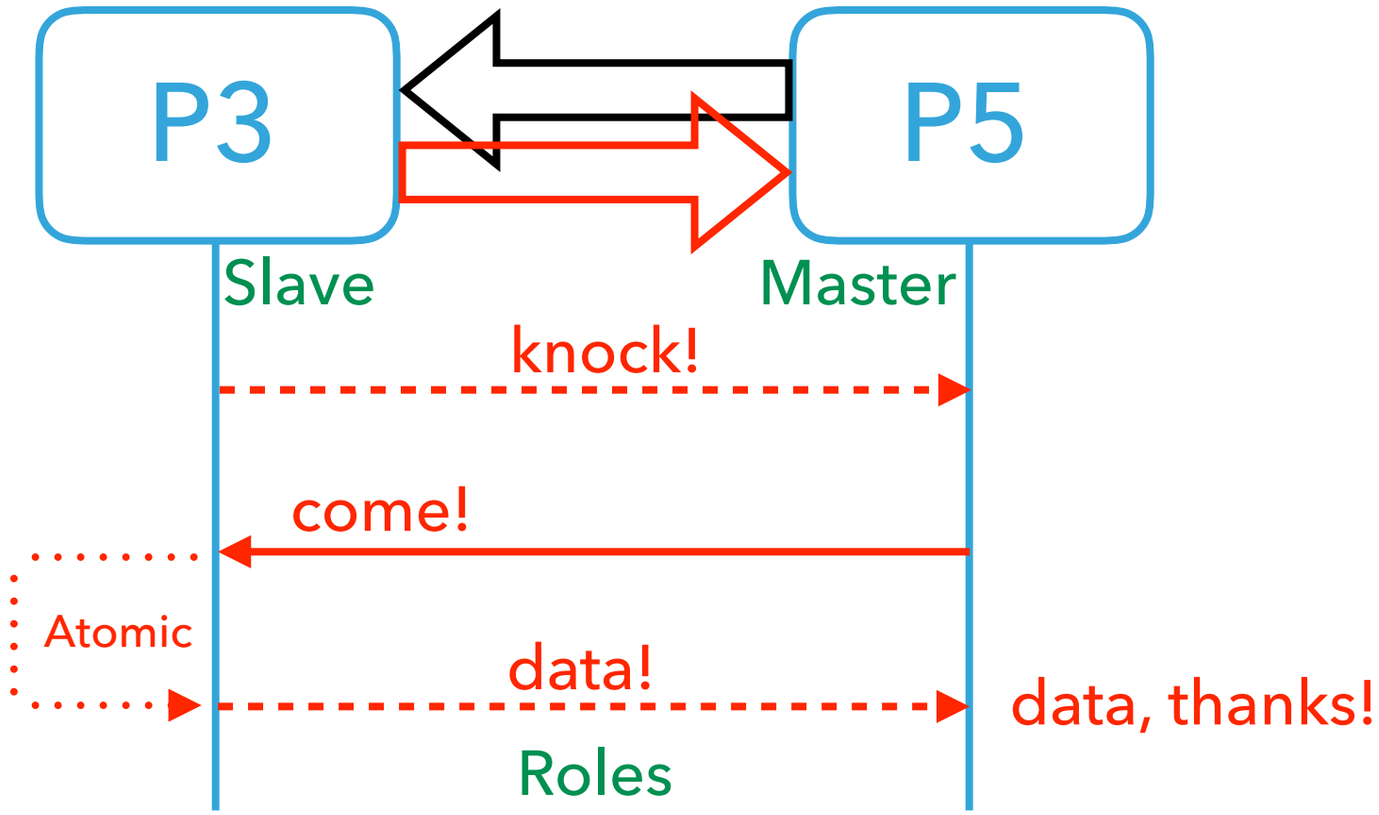
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block



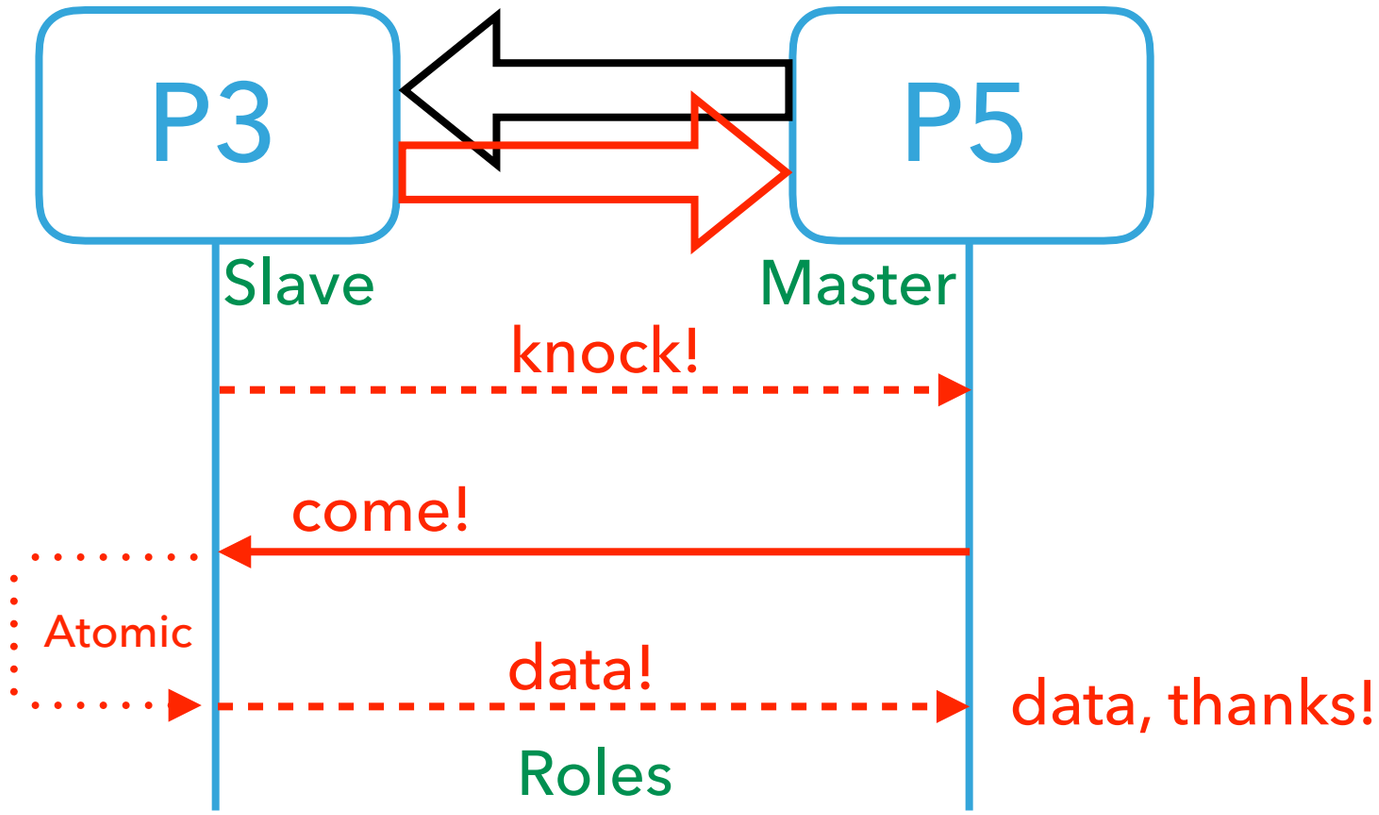
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block

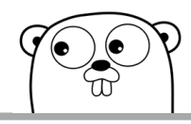


KNOCK-COME, THEN DATA

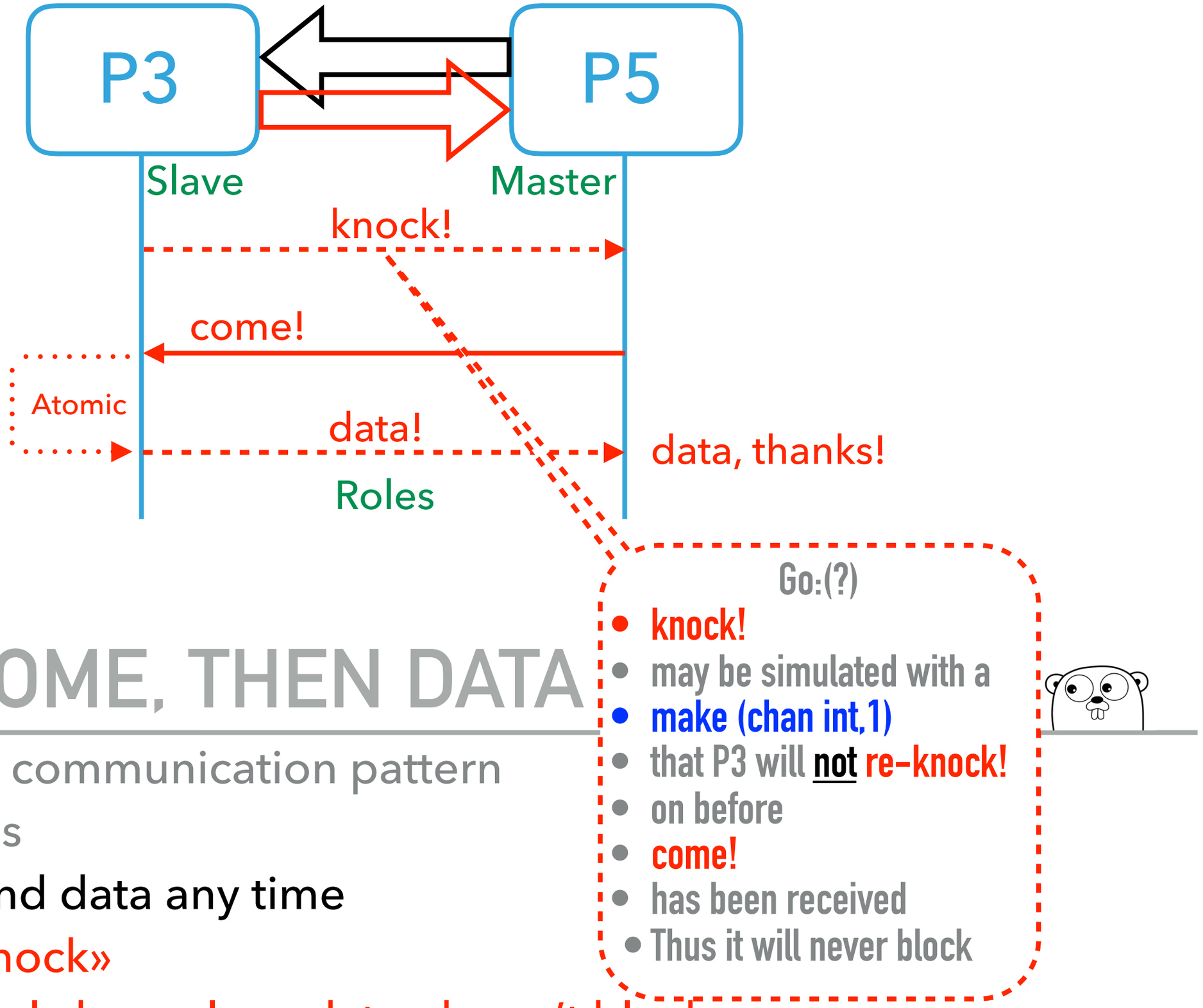
- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



KNOCK-COME, THEN DATA

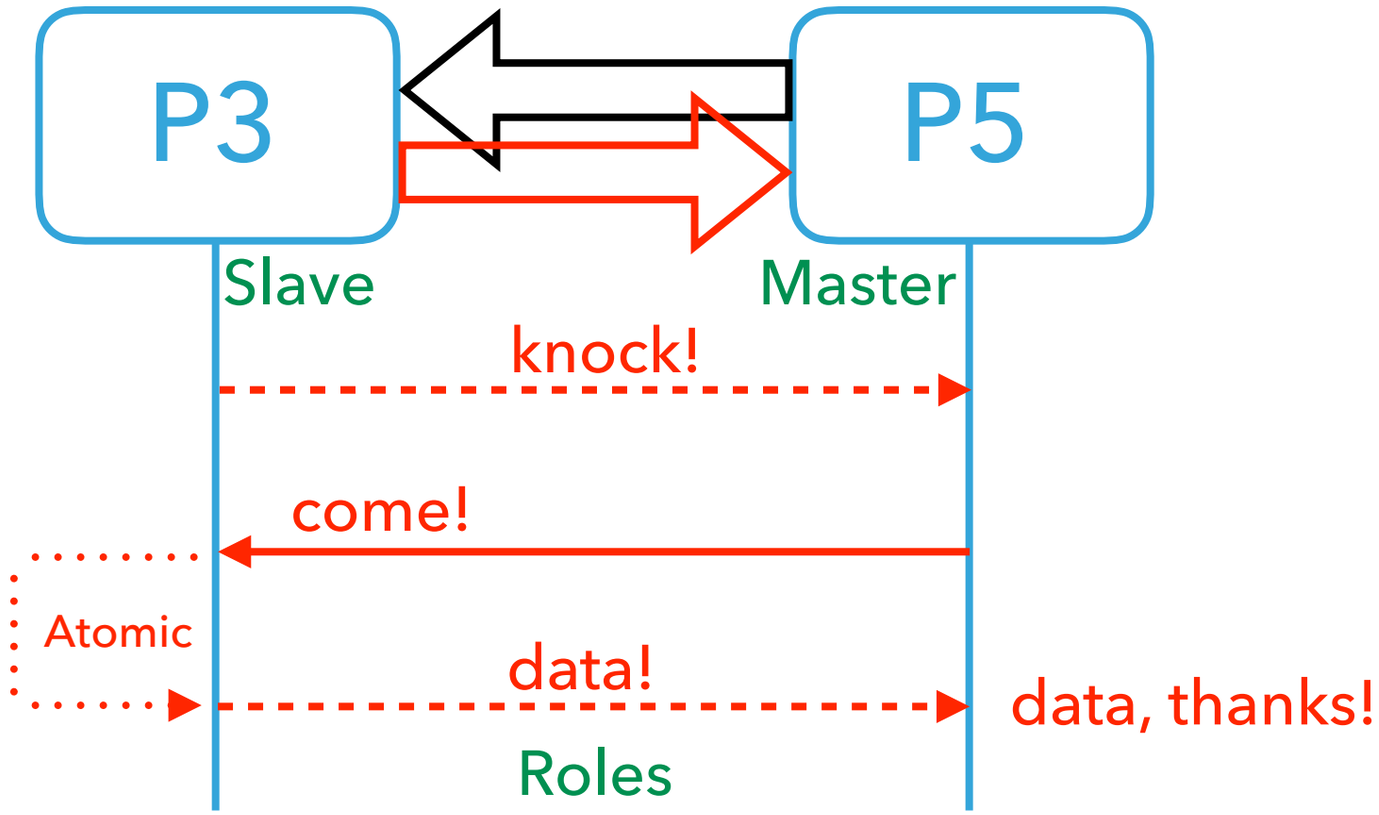


- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block

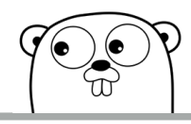


KNOCK-COME, THEN DATA

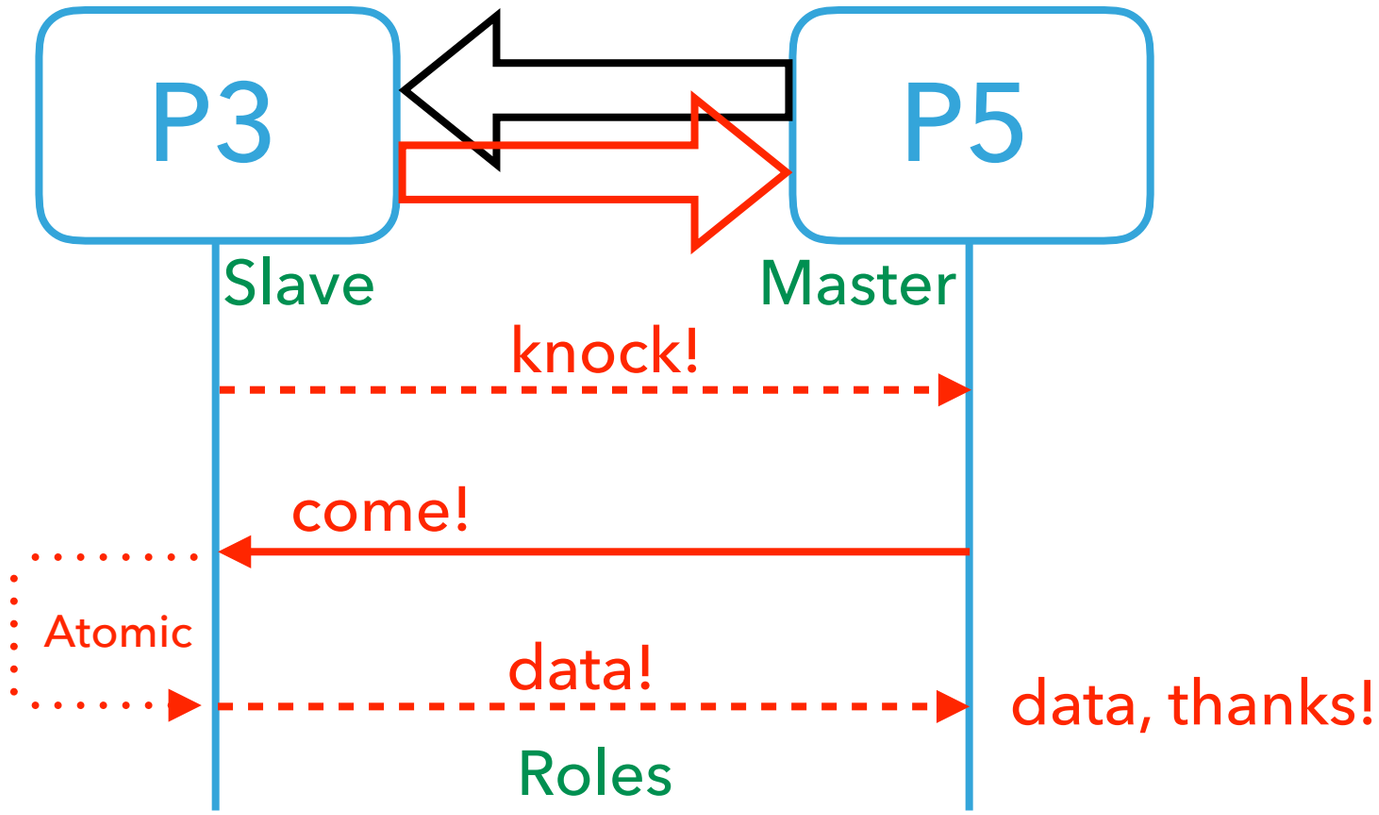
- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block



KNOCK-COME, THEN DATA

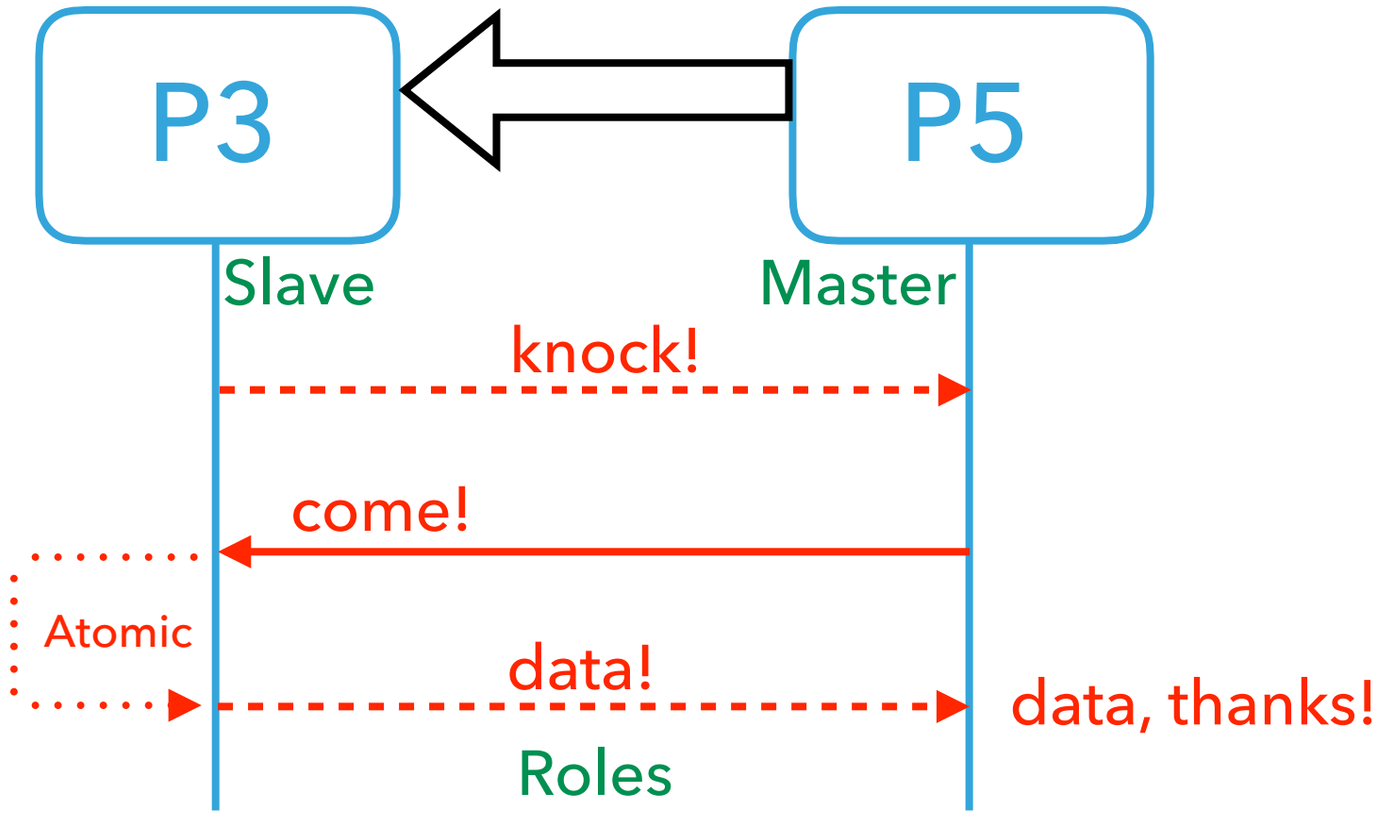


- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block



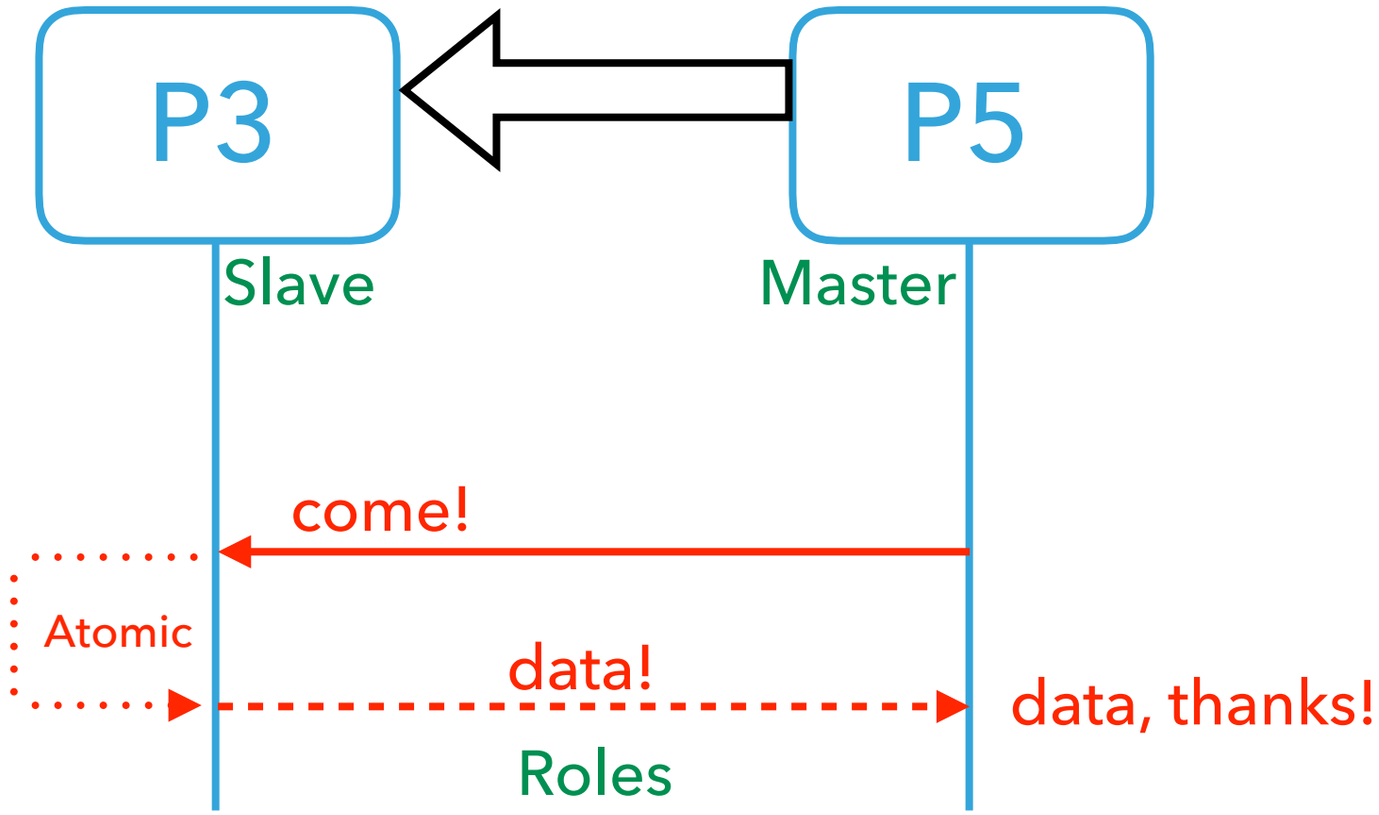
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



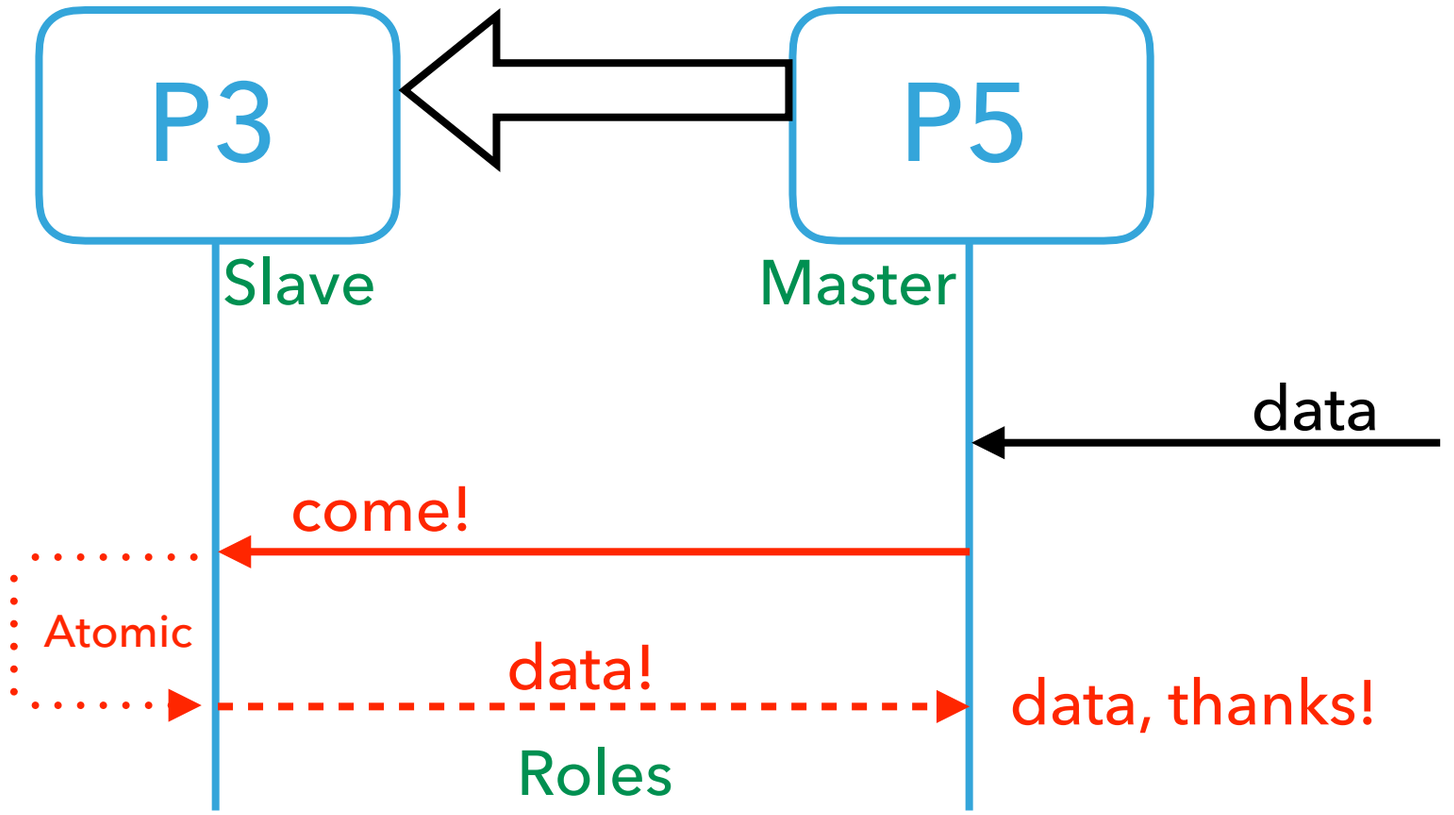
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



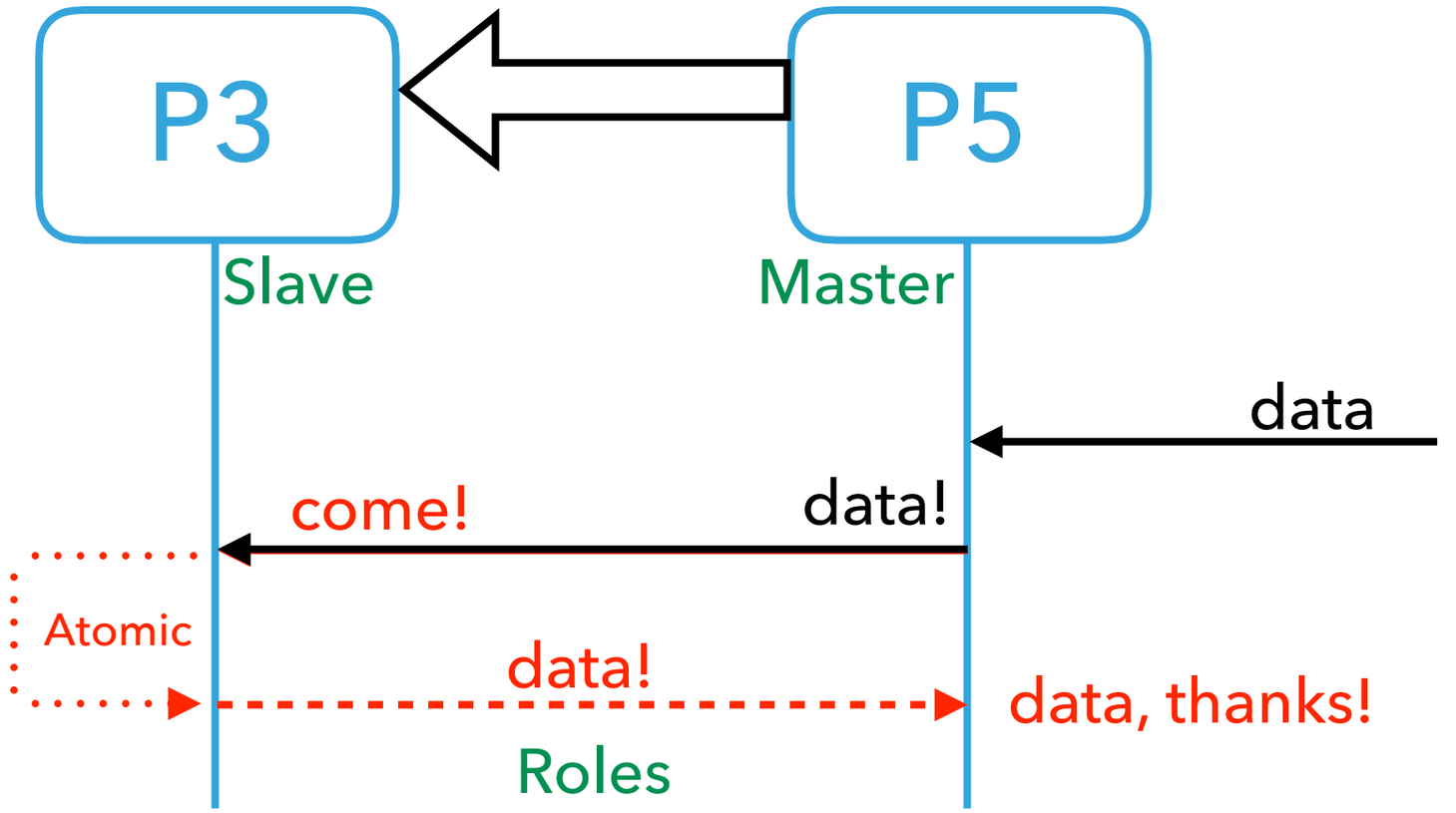
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



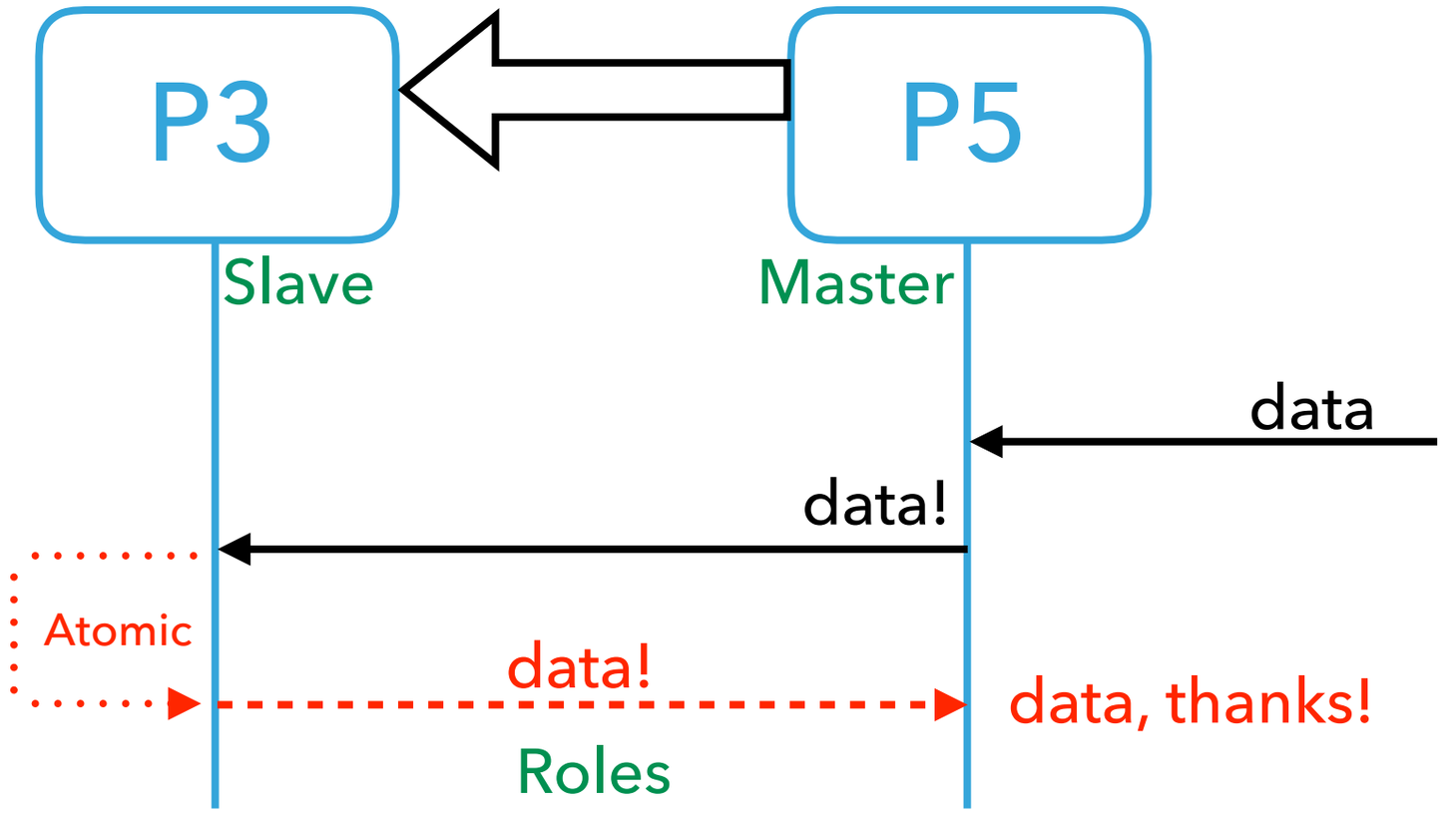
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



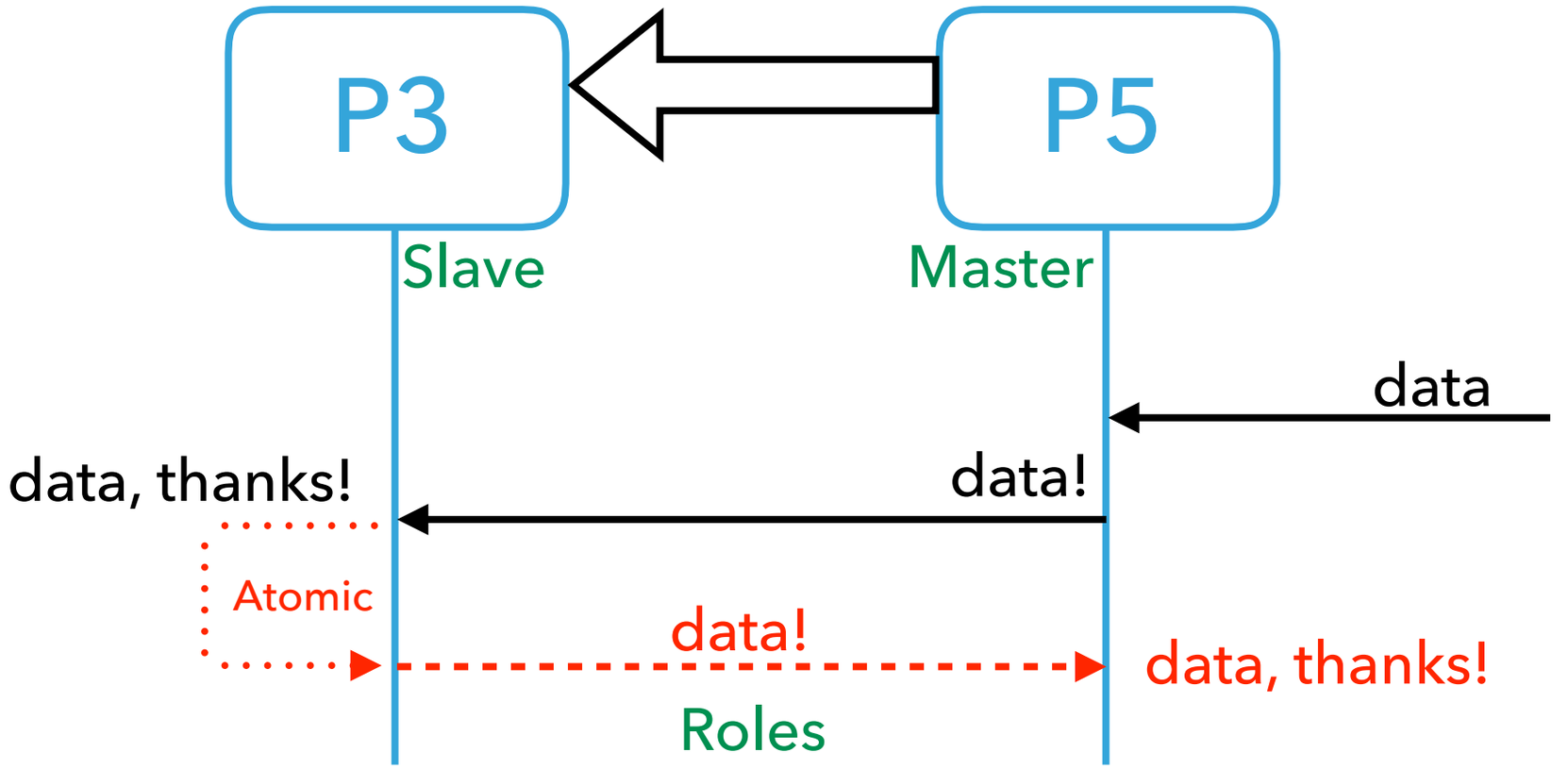
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



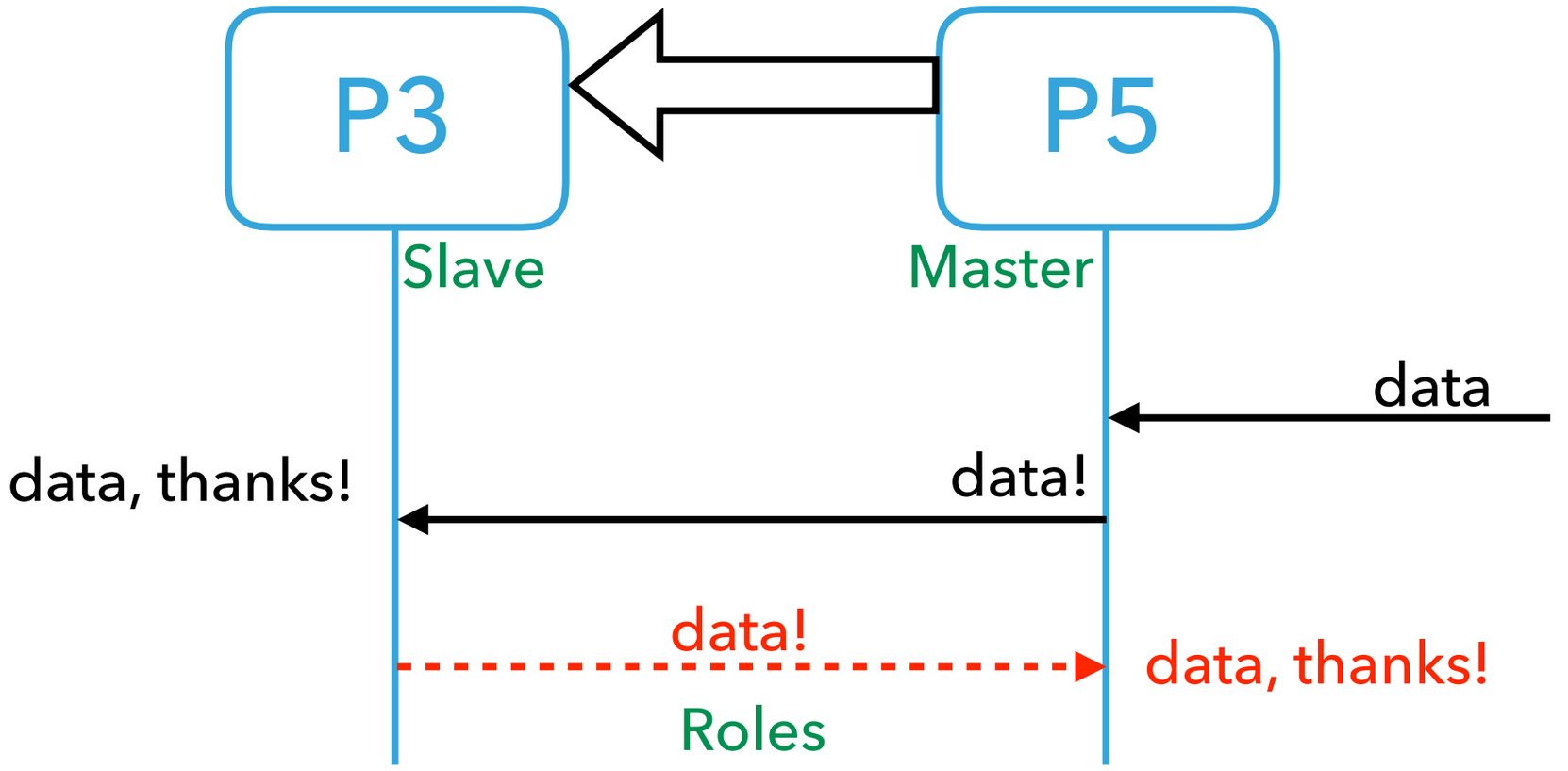
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



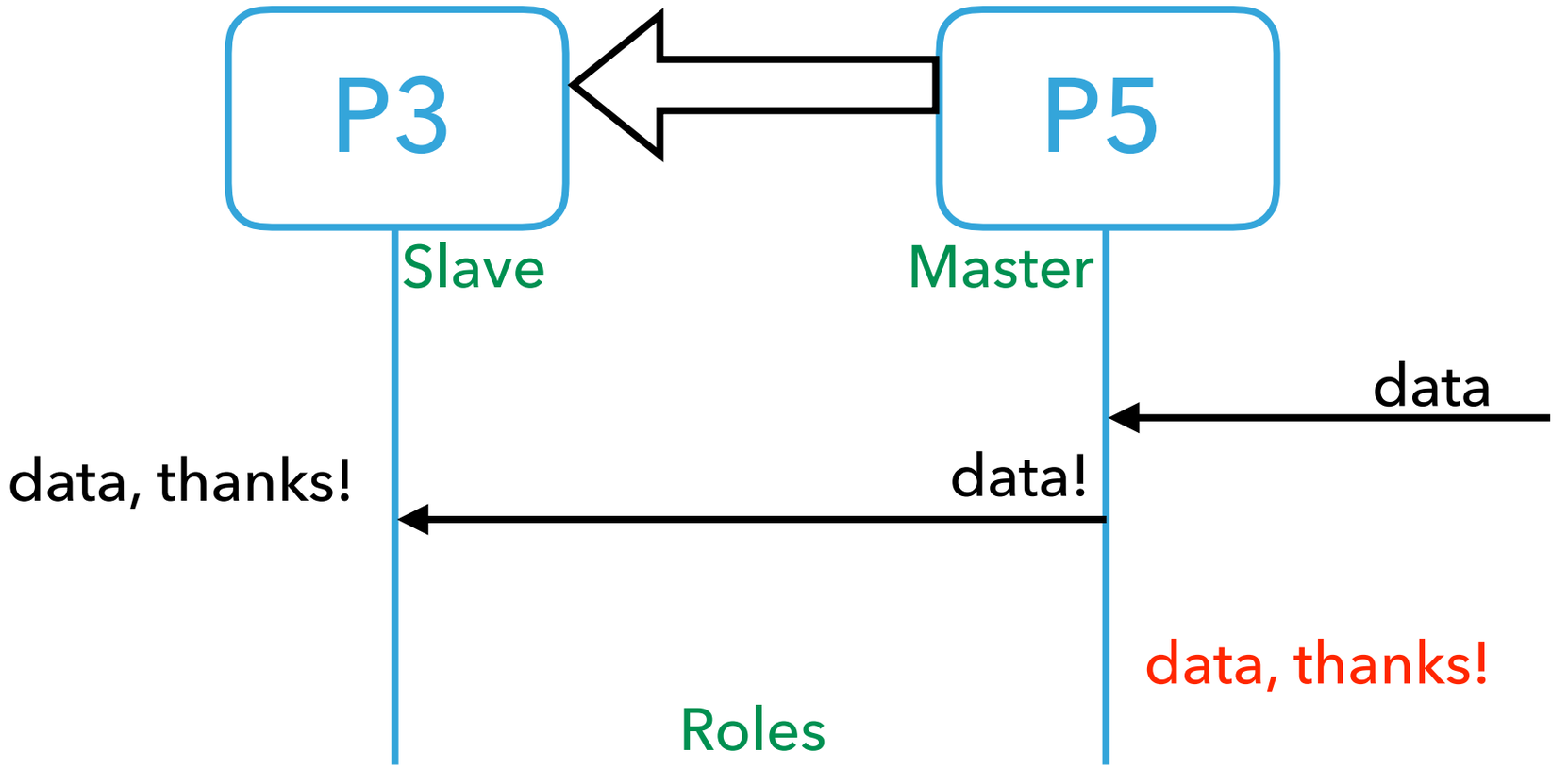
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



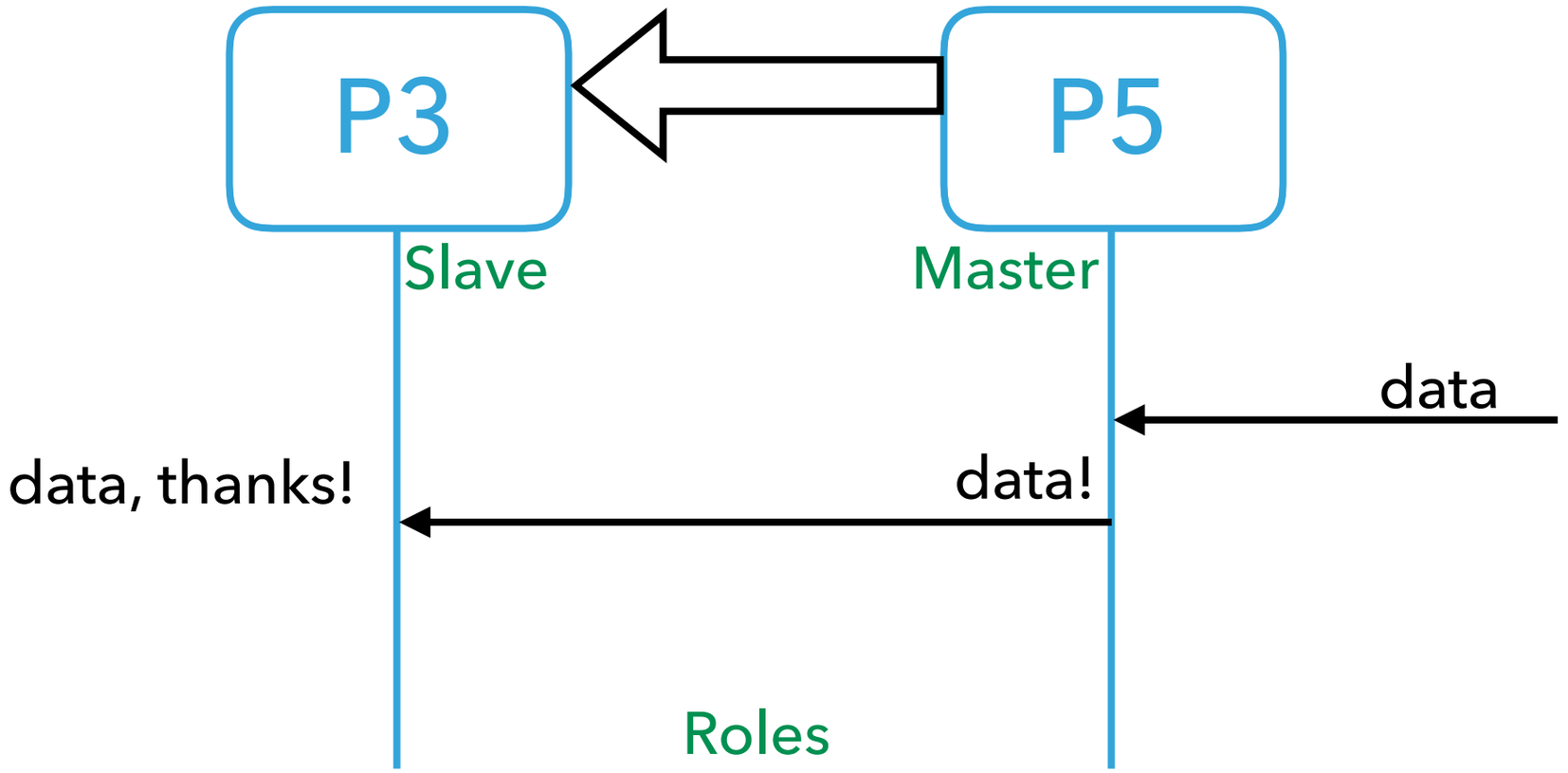
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



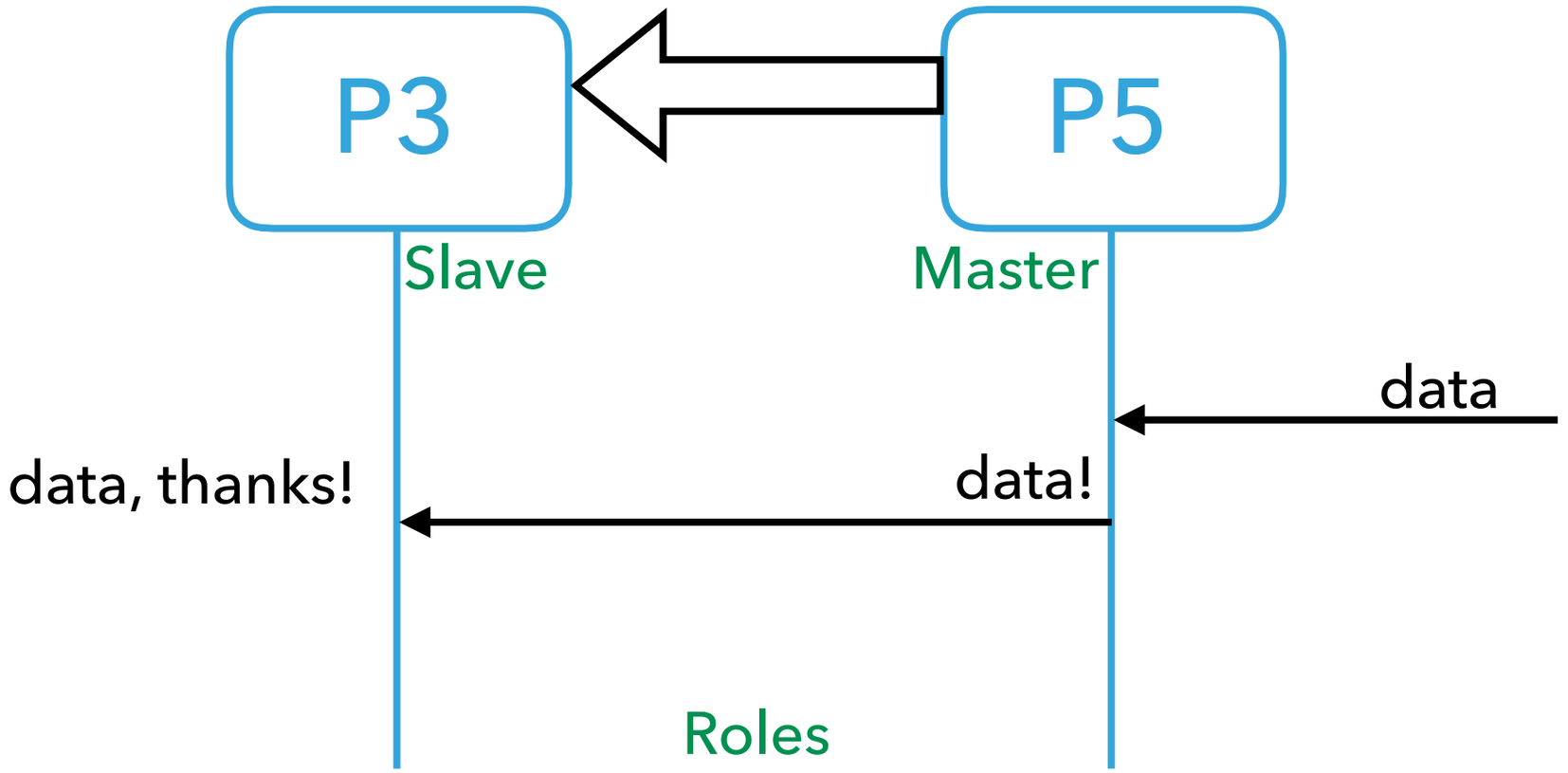
KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn’t block



KNOCK-COME, THEN DATA

- Deadlock free communication pattern
- Both directions
- Master can send data any time
- Slave must «knock»
  - asynch signal channel, no data, doesn't block

oyvteig.blogspot.no/2009/03/009-knock-come-deadlock-free-pattern.html

GUARDS

Go “simulates” a guard if a communication component is `nil`

GUARDS

Go “simulates” a guard if a communication component is `nil`

The Go Playground Run Format Imports Share

```
1 func Server(in <-chan int, out chan<- int) {
2         value := 0     // Declaration and assignment
3         valid := false // --“--
4         for {
18        }
19 }
```

GUARDS

Go “simulates” a guard if a communication component is `nil`

The Go Playground Run Format Imports Share

```
1  func Server(in <-chan int, out chan<- int) {
2          value := 0     // Declaration and assignment
3          valid := false // --“--
4          for {
5                  outc := out // Always use a copy of "out"
11                 select {
12                 case value = <-in: // RECEIVE?
13                         // "Overflow" if valid is already true.
14                         valid = true
15                 case outc <- value: // SEND?
16                         valid = false
17                 }
18         }
19 }
```

GUARDS

Go “simulates” a guard if a communication component is `nil`

The Go Playground Run Format Imports Share

```
1  func Server(in <-chan int, out chan<- int) {
2          value := 0     // Declaration and assignment
3          valid := false // --“--
4          for {
5                  outc := out // Always use a copy of "out"
6                  // If we have no value, then don't attempt
7                  // to send it on the out channel:
8                  if !valid {
9                          outc = nil // Makes input alone in select
10                 }
11                 select {
12                 case value = <-in: // RECEIVE?
13                         // "Overflow" if valid is already true.
14                         valid = true
15                 case outc <- value: // SEND?
16                         valid = false
17                 }
18         }
19 }
```

GUARDS

Go “simulates” a guard if a communication component is `nil`

Referred in http://www.teigfam.net/oyvind/pub/pub_details.html#XCHAN

The Go Playground Run Format Imports Share

```
1  func Server(in <-chan int, out chan<- int) {
2          value := 0     // Declaration and assignment
3          valid := false // --“--
4          for {
5                  outc := out // Always use a copy of "out"
6                  // If we have no value, then don't attempt
7                  // to send it on the out channel:
8                  if !valid {
9                          outc = nil // Makes input alone in select
10                 }
11                 select {
12                 case value = <-in: // RECEIVE?
13                         // "Overflow" if valid is already true.
14                         valid = true
15                 case outc <- value: // SEND?
16                         valid = false
17                 }
18         }
19 }
```

GUARDS

XC has guards built into the language. Plus `interface`

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
```

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
```

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
13 }
```

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
13 }
```

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
10     case (e == 1) => i.g(): {
11         ...
12     } break;
13 }
```

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
10     case (e == 1) => i.g(): {
11         ...
12     } break;
13 }
```

GUARDS

XC has guards built into the language. Plus `interface`

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
10     case (e == 1) => i.g(): {
11         ...
12     } break;
13 }
```

Implemented with channels, states and/or locks by the XC compiler

GUARDS

XC has guards built into the language. Plus `interface`

https://www.xmos.com/published/xmos-programming-guide

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
10     case (e == 1) => i.g(): {
11         ...
12     } break;
13 }
```

Implemented with channels, states and/or locks by the XC compiler

GUARDS

XC has guards built into the language. Plus `interface`

https://www.xmos.com/published/xmos-programming-guide

```
1  interface if1 {
2      void f();
3      [[guarded]] void g(); // this function may be guarded in the program
4  }
5  ..
6  select {
7      case i.f(): {
8          ...
9      } break;
10     case (e == 1) => i.g(): {
11         ...
12     } break;
13 }
```

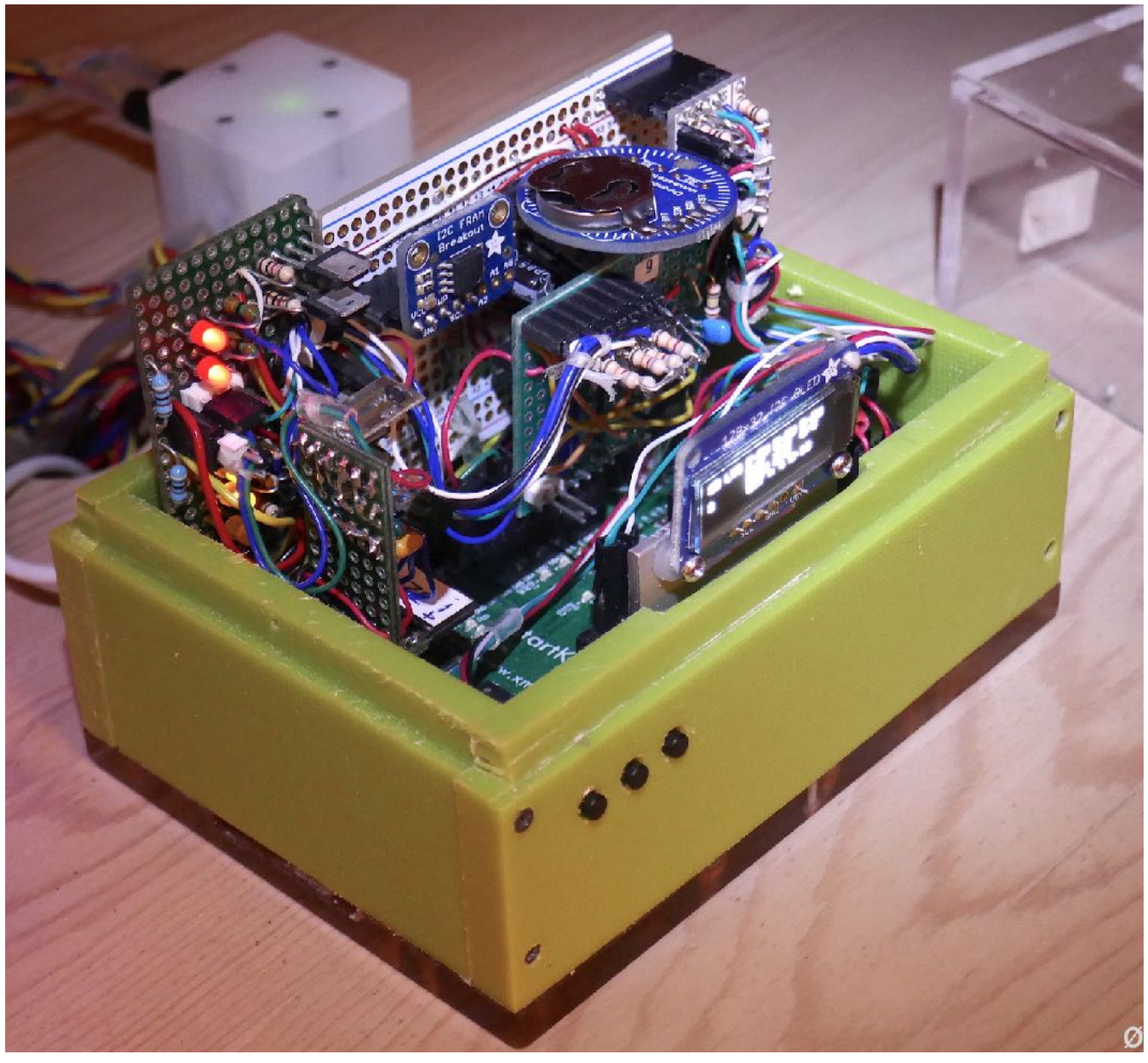
Implemented with channels, states and/or locks by the XC compiler

I use this at home:

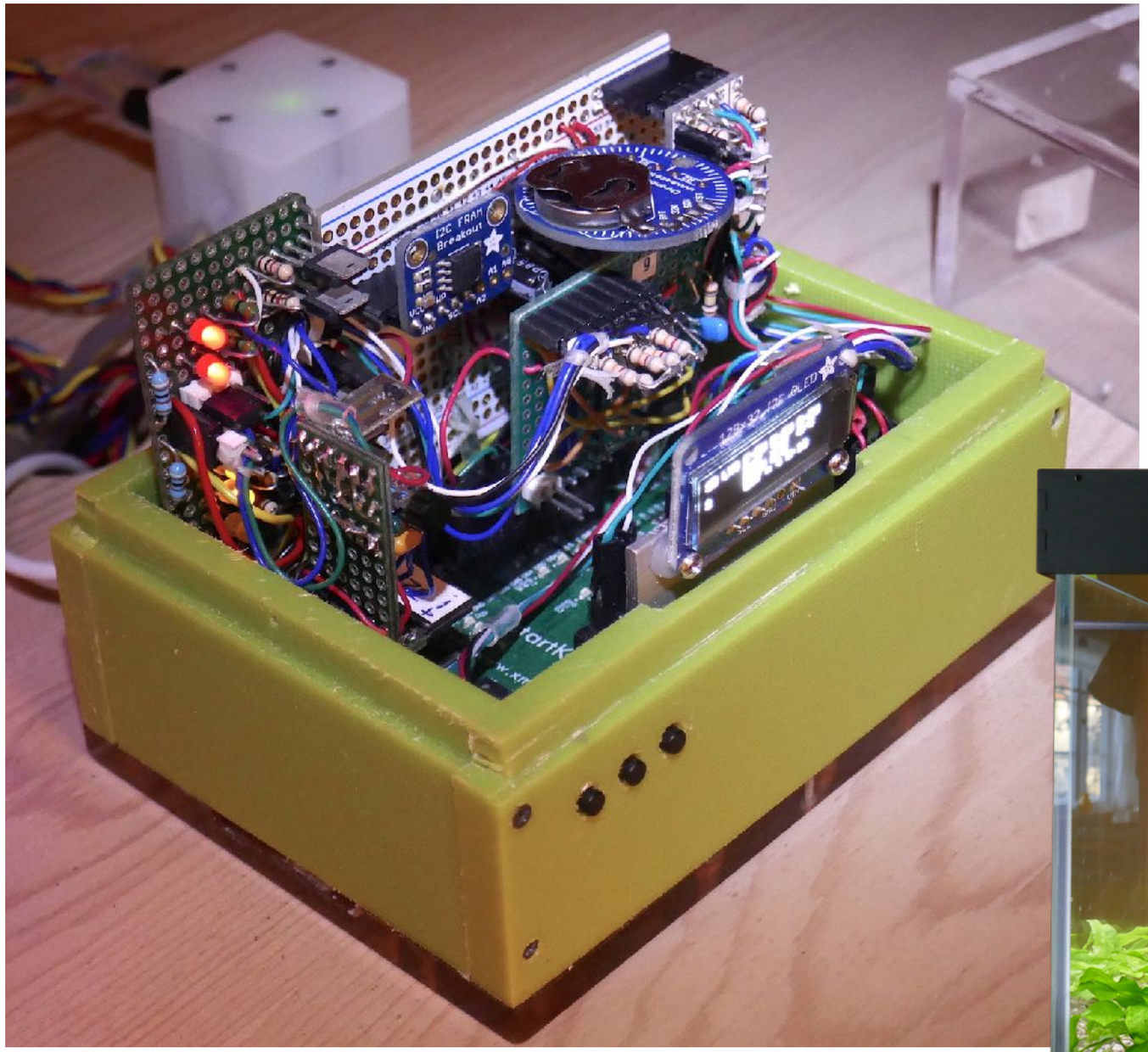
AQUARIUM CONTROL UNIT WITH XMOS `startKIT`, 8 LOGICAL CORES IN `xC`

Aside

AQUARIUM CONTROL UNIT WITH XMOS `startKIT`, 8 LOGICAL CORES IN `xC`



AQUARIUM CONTROL UNIT WITH XMOS `startKIT`, 8 LOGICAL CORES IN `xC`



Showing a forest for some trees 2

KEYWORDS interface, server, client AND slave etc.

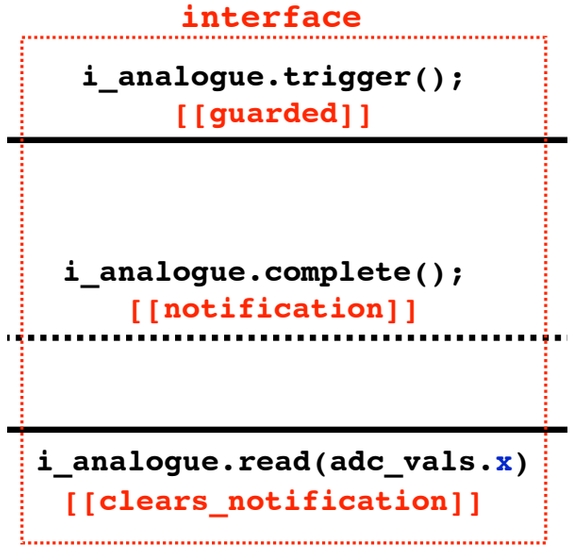
```
typedef interface startkit_adc_if {
  [[guarded]]                void trigger(void);
  [[clears_notification]]    int read(unsigned short
adc_val[4]);
  [[notification]]           slave void complete(void);
} startkit_adc_if;
```

```
interface startkit_adc_if i_analogue;
```

Showing a forest for some trees 2

KEYWORDS interface, server, client AND slave etc.

```
typedef interface startkit_adc_if {
  [[guarded]]                void trigger(void);
  [[clears_notification]]    int read(unsigned short
adc_val[4]);
  [[notification]]           slave void complete(void);
} startkit_adc_if;
```

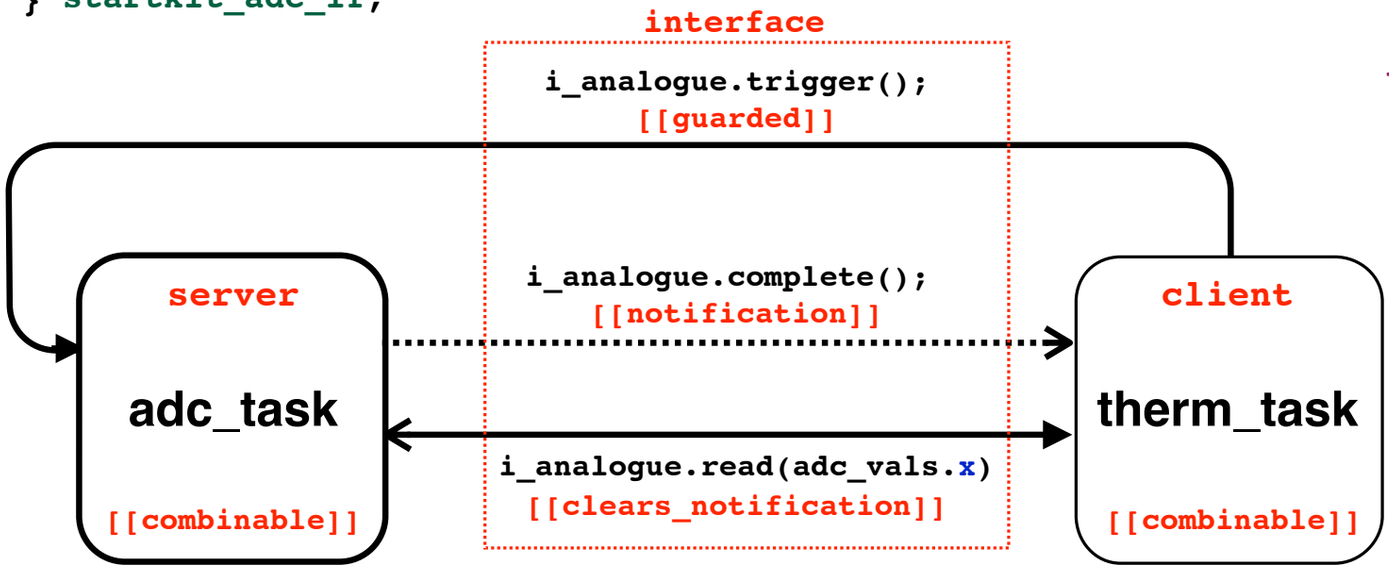



```
interface startkit_adc_if i_analogue;
```

KEYWORDS interface, server, client AND slave etc.

```
typedef interface startkit_adc_if {
  [[guarded]]                 void trigger(void);
  [[clears_notification]]     int read(unsigned short
adc_val[4]);
  [[notification]]            slave void complete(void);
} startkit_adc_if;
```

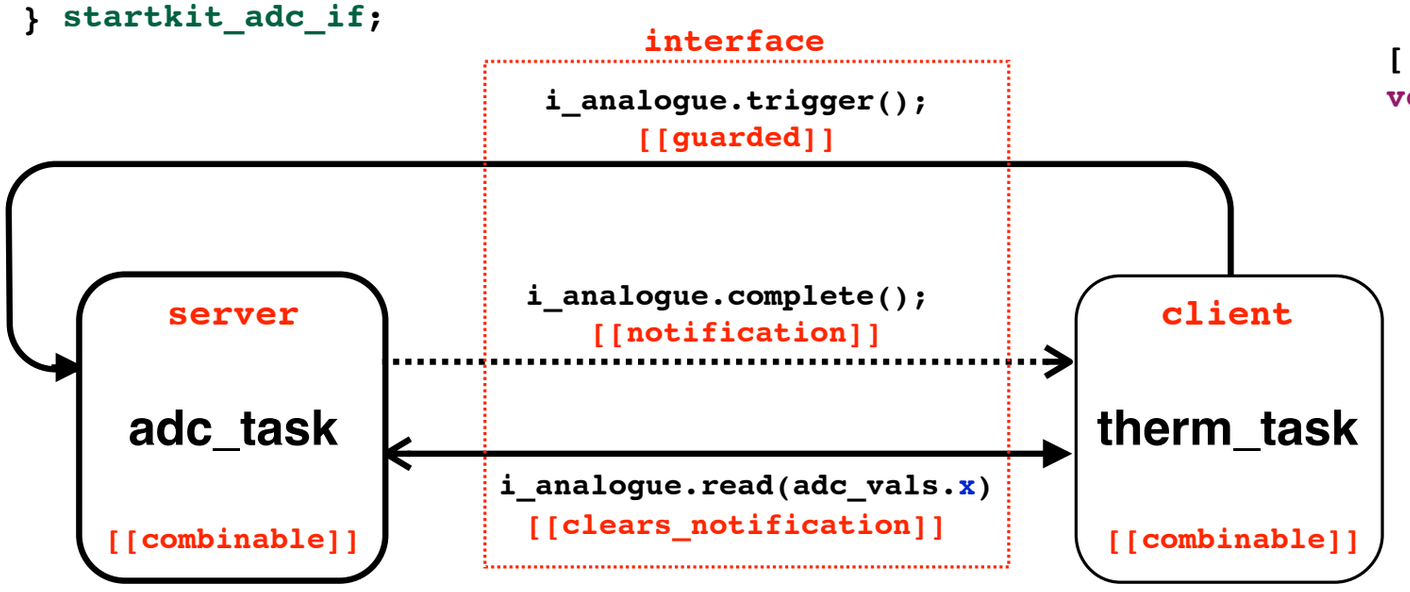
```
interface startkit_adc_if i_analogue;
```



Showing a forest for some trees 2

KEYWORDS interface, server, client AND slave etc.

```
typedef interface startkit_adc_if {
  [[guarded]]                 void trigger(void);
  [[clears_notification]]     int read(unsigned short
adc_val[4]);
  [[notification]]            slave void complete(void);
} startkit_adc_if;
```




```
interface startkit_adc_if i_analogue;
```

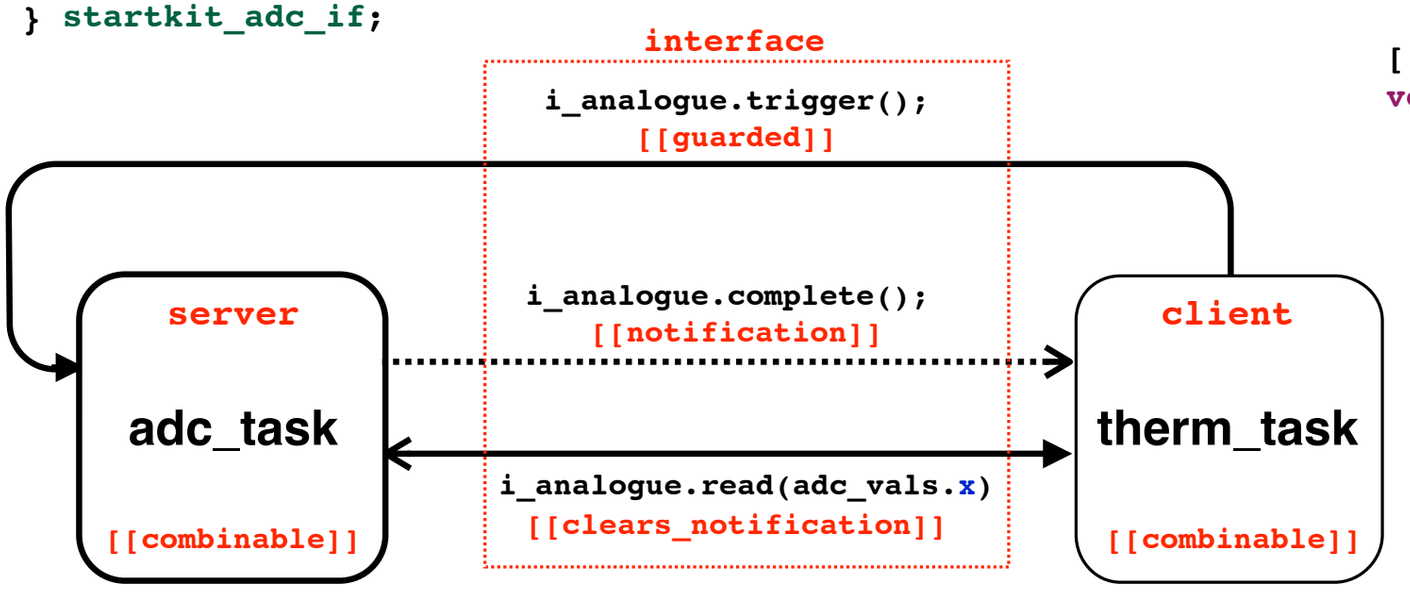
```
[[combinable]]
void therm_task
  // ...
  while(1) {
    select {
      case wait_for_button => c_button_2 :> int x: {
         // ...
         i_analogue.trigger();
         break; }
      case wait_for_adc => i_analogue.complete(): {
         // ...
         if (i_analogue.read(adc_vals.x)) {
           // Use it
         } break;
      }
    }
  }
}
```

Drawing by Øyvind Teig

Showing a forest for some trees 2

KEYWORDS interface, server, client AND slave etc.

```
typedef interface startkit_adc_if {
  [[guarded]]                 void trigger(void);
  [[clears_notification]]     int read(unsigned short
adc_val[4]);
  [[notification]]            slave void complete(void);
} startkit_adc_if;
```



Also has traditional `chan` (untyped)
Guaranteed deterministic real-time response

```
interface startkit_adc_if i_analogue;
```

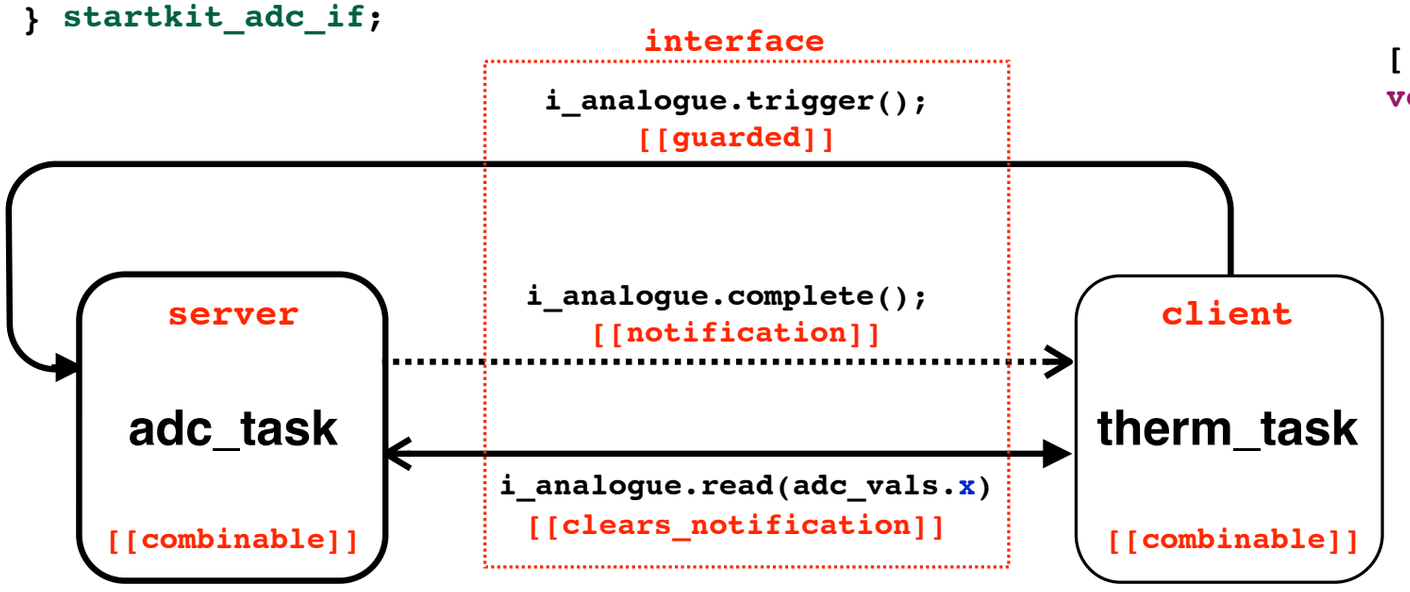
```
[[combinable]]
void therm_task
  // ...
  while(1) {
    select {
      case wait_for_button => c_button_2 :> int x: {
        // ...
        i_analogue.trigger();
        break; }
      case wait_for_adc => i_analogue.complete(): {
        // ...
        if (i_analogue.read(adc_vals.x)) {
          // Use it
        } break;
      }
    }
  }
}
```

Drawing by Øyvind Teig

Showing a forest for some trees 2

KEYWORDS interface, server, client AND slave etc.

```
typedef interface startkit_adc_if {
  [[guarded]]                 void trigger(void);
  [[clears_notification]]     int read(unsigned short
adc_val[4]);
  [[notification]]            slave void complete(void);
} startkit_adc_if;
```



Also has traditional `chan` (untyped)
Guaranteed deterministic real-time response

```
interface startkit_adc_if i_analogue;
```

```
[[combinable]]
void therm_task
  // ...
  while(1) {
    select {
      case wait_for_button => c_button_2 :> int x: {
        // ...
        i_analogue.trigger();
        break; }
      case wait_for_adc => i_analogue.complete(): {
        // ...
        if (i_analogue.read(adc_vals.x)) {
          // Use it
        } break;
      }
    }
  }
}
```

Drawing by Øyvind Teig

This pattern is understood by the compiler and it is deadlock free

occam, too. But it didn't have `interface`

occam, too. But it didn't have `interface`

ALT

occam, too. But it didn't have `interface`

ALT

GUARDS

occam, too. But it didn't have `interface`

```
ALT
  count1 < 100 & c1 ? data
    SEQ
      count1 := count1 + 1
      merged ! data
  
```

occam, too. But it didn't have `interface`

```
ALT
  count1 < 100 & c1 ? data
    SEQ
      count1 := count1 + 1
      merged ! data
  count2 < 100 & c2 ? data
    SEQ
      count2 := count2 + 1
      merged ! data
```

GUARDS

occam, too. But it didn't have `interface`

```
ALT
  count1 < 100 & c1 ? data
    SEQ
      count1 := count1 + 1
      merged ! data
  count2 < 100 & c2 ? data
    SEQ
      count2 := count2 + 1
      merged ! data
status ? request
   SEQ
     out ! count1
     out ! count2
```

GUARDS

occam, too. But it didn't have `interface`

```
ALT
  count1 < 100 & c1 ? data
     SEQ
       count1 := count1 + 1
       merged ! data
  count2 < 100 & c2 ? data
     SEQ
       count2 := count2 + 1
       merged ! data
status ? request
   SEQ
     out ! count1
     out ! count2
```

- Logical and-condition (XC, occam), or nil (Go), or just not include in the select set (next page)

GUARDS

occam, too. But it didn't have `interface`

```
ALT
  count1 < 100 & c1 ? data
     SEQ
       count1 := count1 + 1
       merged ! data
  count2 < 100 & c2 ? data
     SEQ
       count2 := count2 + 1
       merged ! data
status ? request
   SEQ
     out ! count1
     out ! count2
```

- Logical and-condition (XC, occam), or nil (Go), or just not include in the select set (next page)
- Any way gives the wanted effect of «protection»

GUARDS

occam, too. But it didn't have `interface`

https://en.wikipedia.org/wiki/Occam_(programming_language)

```
ALT
  count1 < 100 & c1 ? data
     SEQ
       count1 := count1 + 1
       merged ! data
  count2 < 100 & c2 ? data
     SEQ
       count2 := count2 + 1
       merged ! data
status ? request
   SEQ
     out ! count1
     out ! count2
```

- Logical and-condition (XC, occam), or nil (Go), or just not include in the select set (next page)
- Any way gives the wanted effect of «protection»

GUARDS

PyCSP

GUARDS

PyCSP

- AltSelect

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)
- InputGuard(cin, action=[optional])

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)

- InputGuard(cin, action=[optional])
- OutputGuard(cout, msg=<message>, action=[optional])

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)

- InputGuard(cin, action=[optional])
- OutputGuard(cout, msg=<message>, action=[optional])
- TimeoutGuard(seconds=<s>, action=[optional])

GUARDS

PyCSP

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)

- InputGuard(cin, action=[optional])
- OutputGuard(cout, msg=<message>, action=[optional])
- TimeoutGuard(seconds=<s>, action=[optional])
- SkipGuard(action=[optional])

GUARDS

PyCSP

https://github.com/runefriborg/pycsp/wiki/Getting_Started_With_PyCSP_2

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)

- InputGuard(cin, action=[optional])
- OutputGuard(cout, msg=<message>, action=[optional])
- TimeoutGuard(seconds=<s>, action=[optional])
- SkipGuard(action=[optional])

GUARDS

PyCSP

https://github.com/runefriborg/pycsp/wiki/Getting_Started_With_PyCSP_2

- AltSelect
  - Guards are tested in the order they are given, but final selection may depend on other factors, such as network latency
- PriSelect
  - Guarantees prioritised selection
- FairSelect
  - See next page (It is called fair choice)

- InputGuard(cin, action=[optional])
- OutputGuard(cout, msg=<message>, action=[optional])
- TimeoutGuard(seconds=<s>, action=[optional])
- SkipGuard(action=[optional])

More about «fairness»:

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice
- occam

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice
- occam
  - Pri select does it, because then one can build fairness «by algorithm»

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice
- occam
  - Pri select does it, because then one can build fairness «by algorithm»
- But which is best? Or best suited? Or good enough?

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice
- occam
  - Pri select does it, because then one can build fairness «by algorithm»
- But which is best? Or best suited? Or good enough?
  - They don’t agree!

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice
- occam
  - Pri select does it, because then one can build fairness «by algorithm»
- But which is best? Or best suited? Or good enough?
  - They don’t agree!

«FAIR» CHOICE: REALLY FAIR OR FAIR ENOUGH?

http://www.teigfam.net/oyvind/home/technology/049-nondeterminism/

- PyCSP
  - Performs a fair selection by reordering guards based on previous choices and then executes a PriSelect on the new order of guards
- Go, XC
  - Nondeterministic (pseudo random) choice
- occam
  - Pri select does it, because then one can build fairness «by algorithm»
- But which is best? Or best suited? Or good enough?
  - They don’t agree!

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

- A channels API for Clojure

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

- A channels API for Clojure
- @Java virtual machine and the Common Language Runtime

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

- A channels API for Clojure
  - @Java virtual machine and the Common Language Runtime
- and ClojureScript

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

- A channels API for Clojure
  - @Java virtual machine and the Common Language Runtime
- and ClojureScript
  - JavaScript -> .NET

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

- A channels API for Clojure
  - @Java virtual machine and the Common Language Runtime
- and ClojureScript
  - JavaScript -> .NET
- Real threads. real blocking

Watch it!

Clojure core.async

https://www.infoq.com/presentations/clojure-core-async

- A channels API for Clojure
  - @Java virtual machine and the Common Language Runtime
- and ClojureScript
  - JavaScript -> .NET
- Real threads. real blocking
- Do watch it! The best to understand what this is all about!

Autronica

BS-100 fire panel (1990..)
In-house scheduler and Modula 2

Autronica

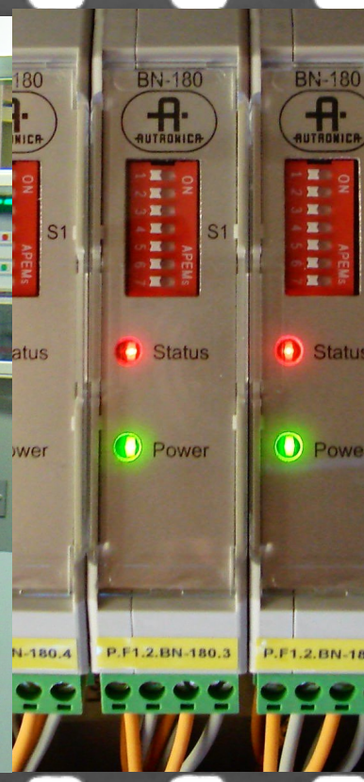
BS-100 fire panel (1990..)
In-house scheduler and Modula 2

Last BS-100 for a ship (2011)
Even in display that scheduler

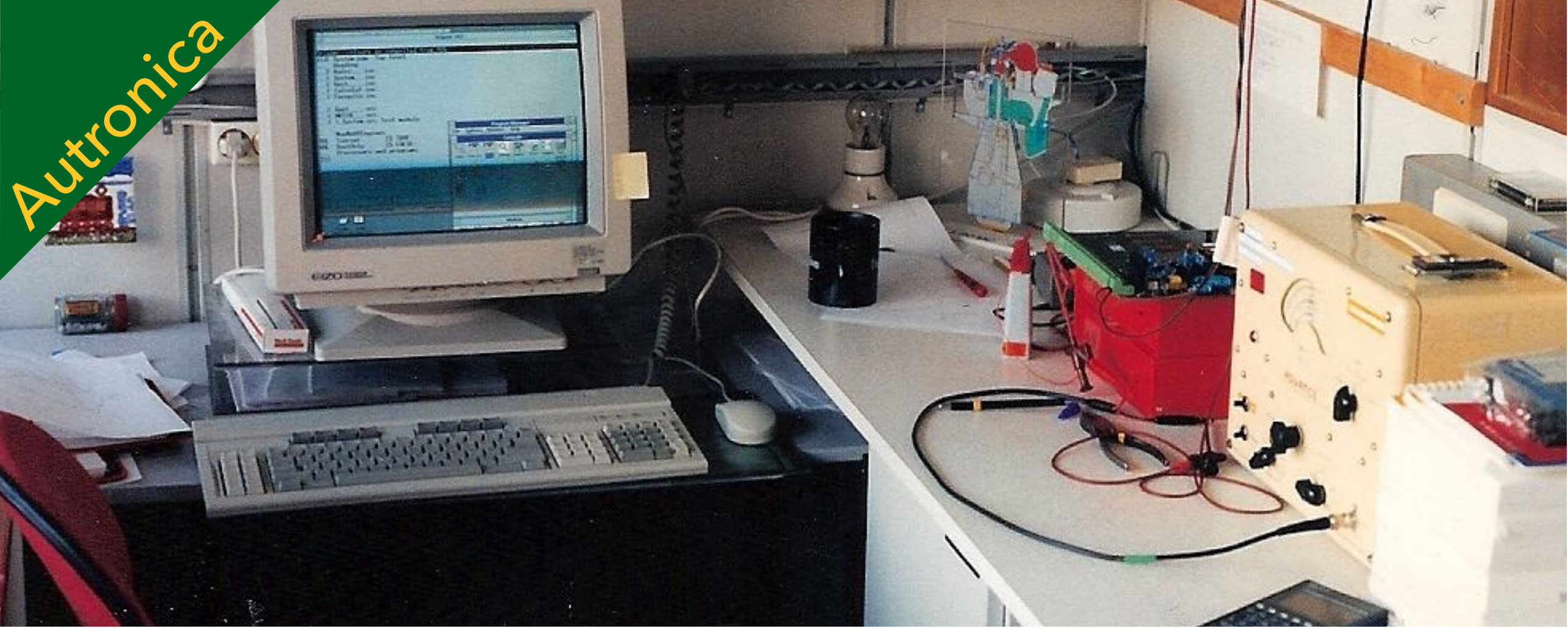
Autronica

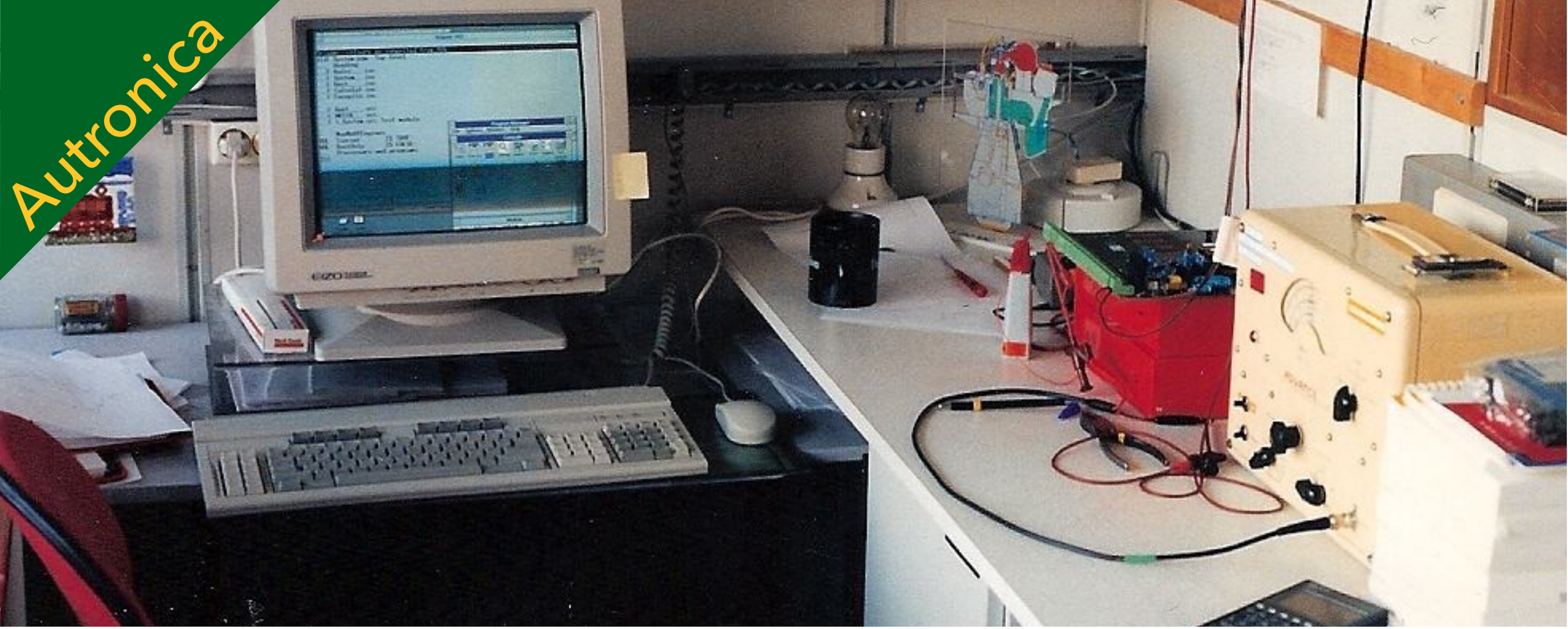
BS-100 fire panel (1990..)
In-house scheduler and Modula 2

Last BS-100 for a ship (2011)
Even in display that scheduler

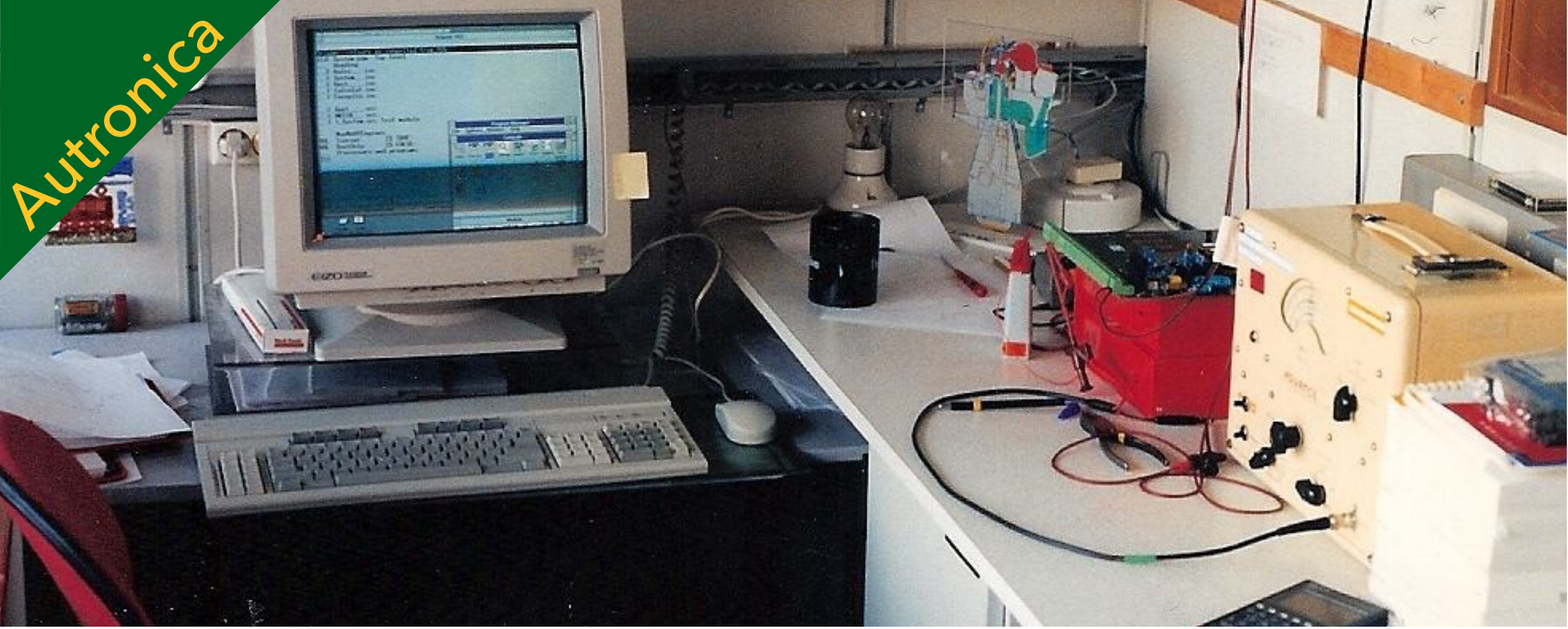


AutroKeeper (2010..)
Chansched scheduler

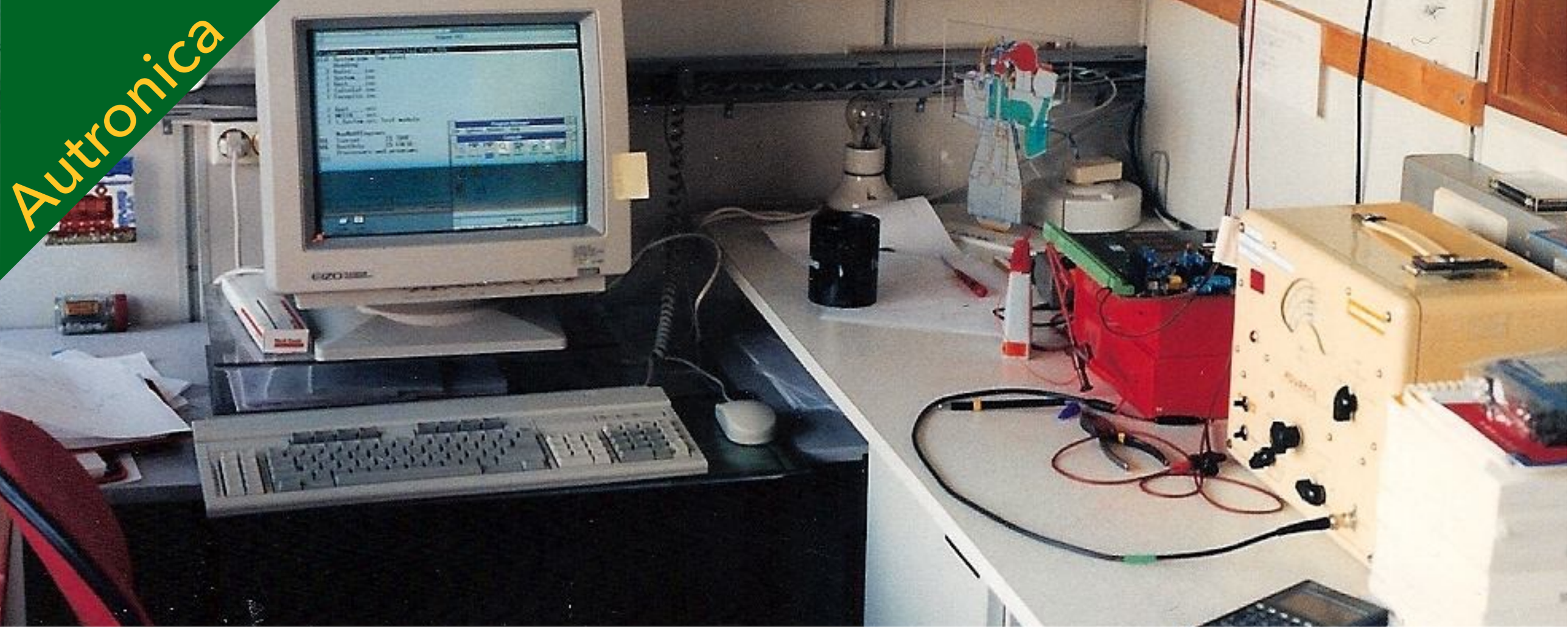

Autronica



1990: OCCAM WITH PROCESS AND CHANNELS.



1990: OCCAM WITH PROCESS AND CHANNELS. SHIP'S ENGINE CONDITION MONITORING (MIP-CALCULATOR: NK-100)



TO ME: NOTHING EVER THE SAME AFTER

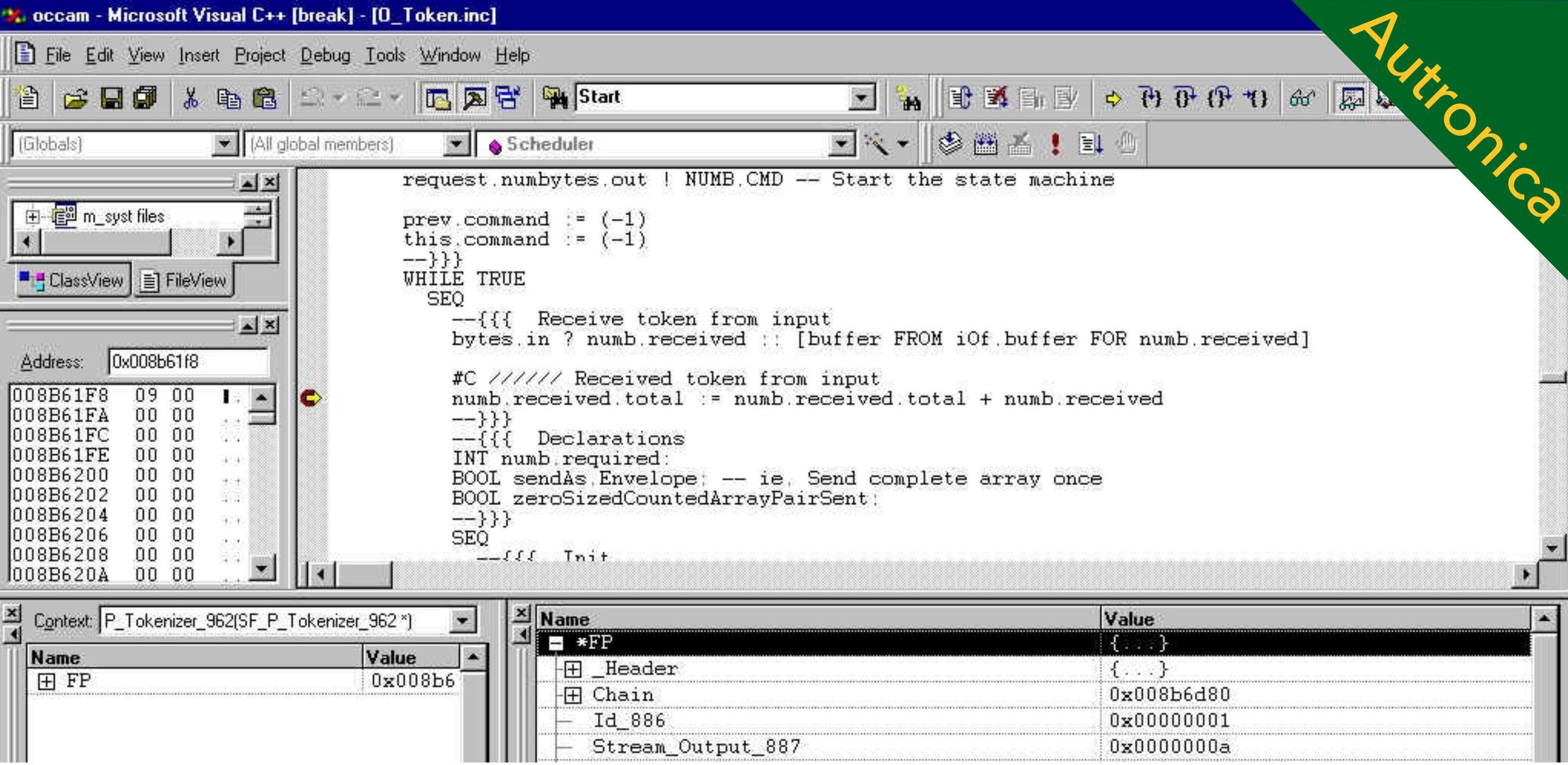
1990: OCCAM WITH PROCESS AND CHANNELS. SHIP'S ENGINE CONDITION MONITORING (MIP-CALCULATOR: NK-100)

Transparent transputer links running in LON industrial network, testing a virtual channel router in my office

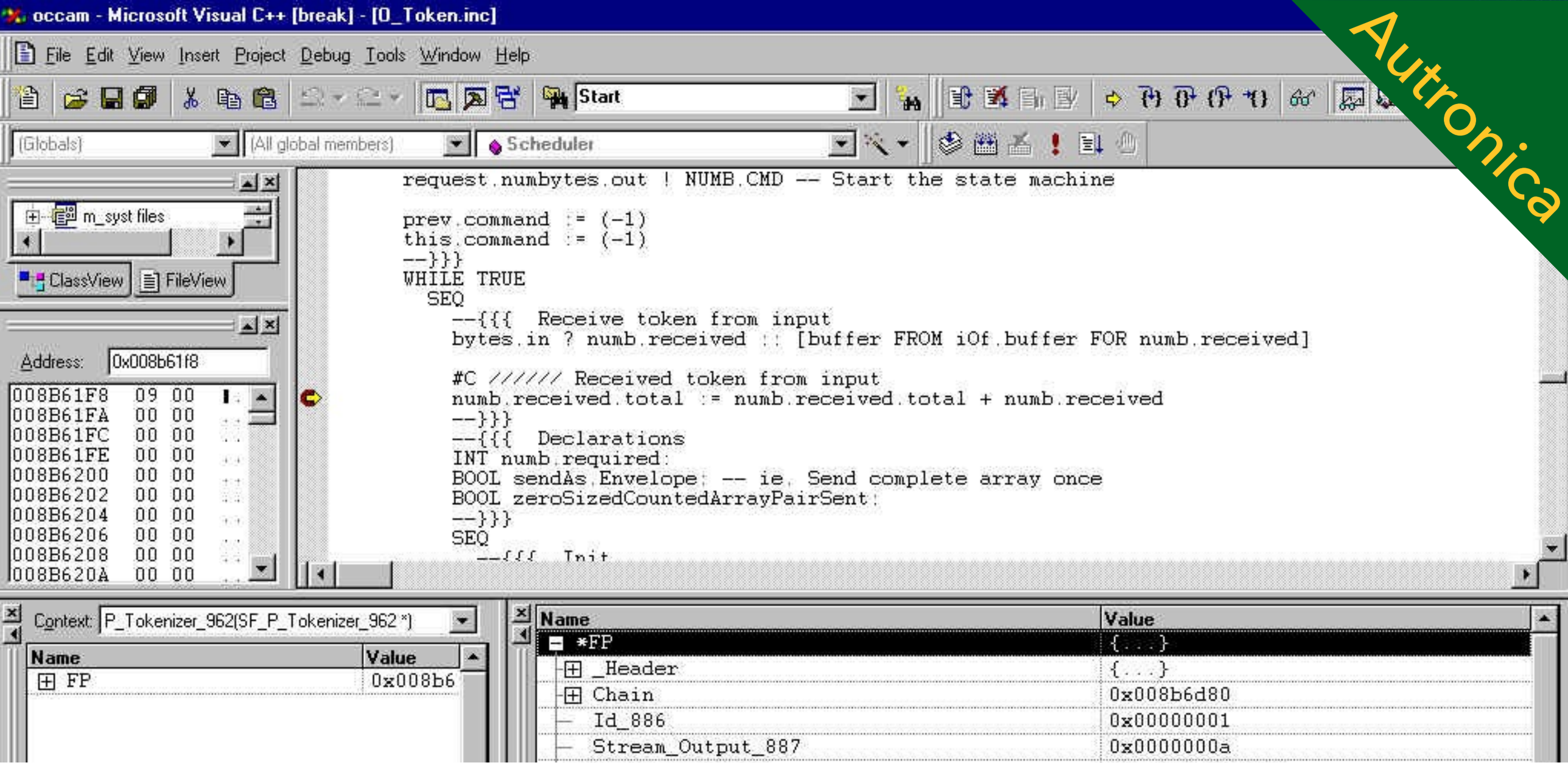
TO ME: NOTHING EVER THE SAME AFTER

1990: OCCAM WITH PROCESS AND CHANNELS. SHIP'S ENGINE CONDITION MONITORING (MIP-CALCULATOR: NK-100)

C? YES: OCCAM TO C: SPOC TOOL

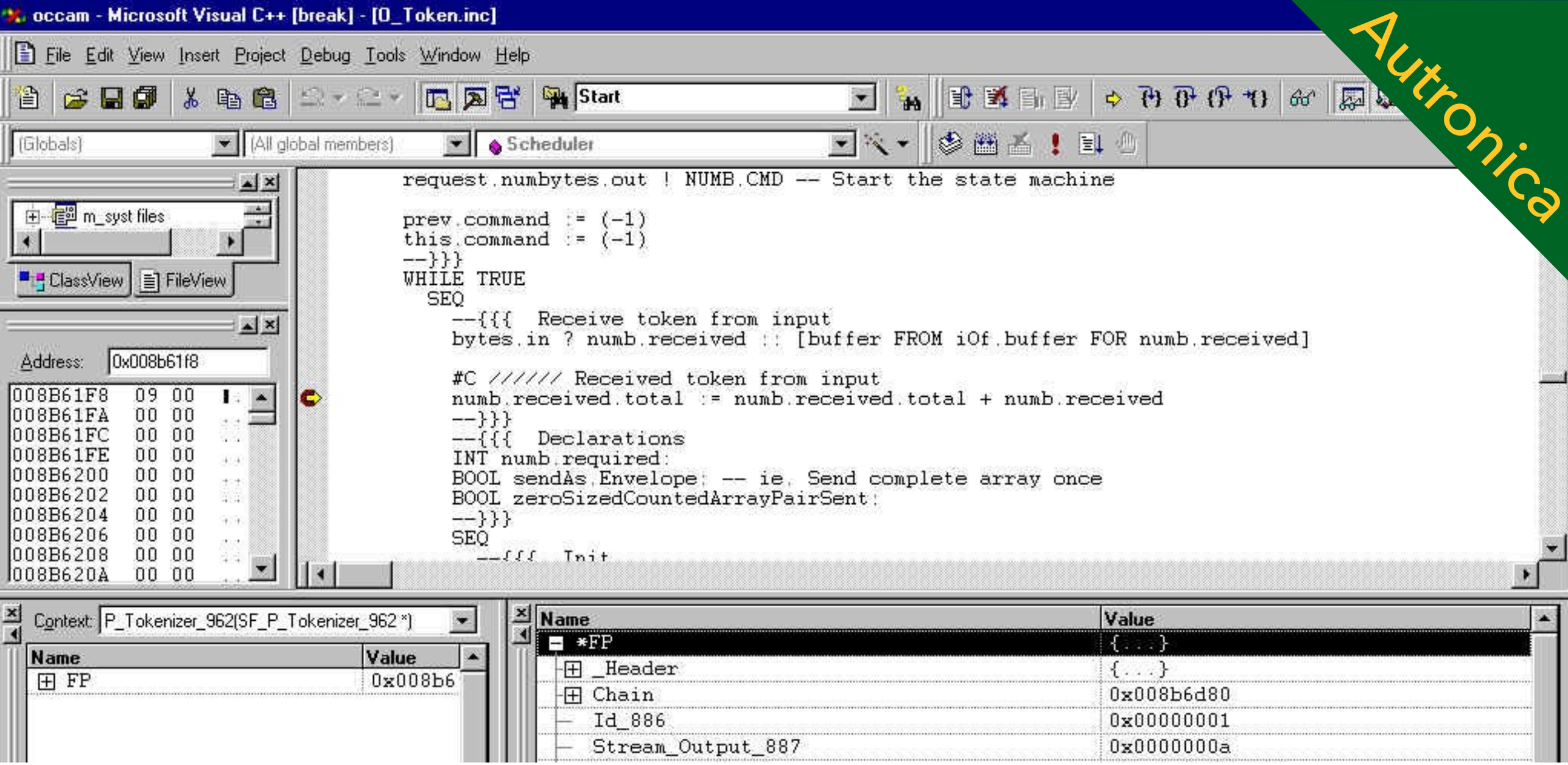


C? YES: OCCAM TO C: SPOC TOOL



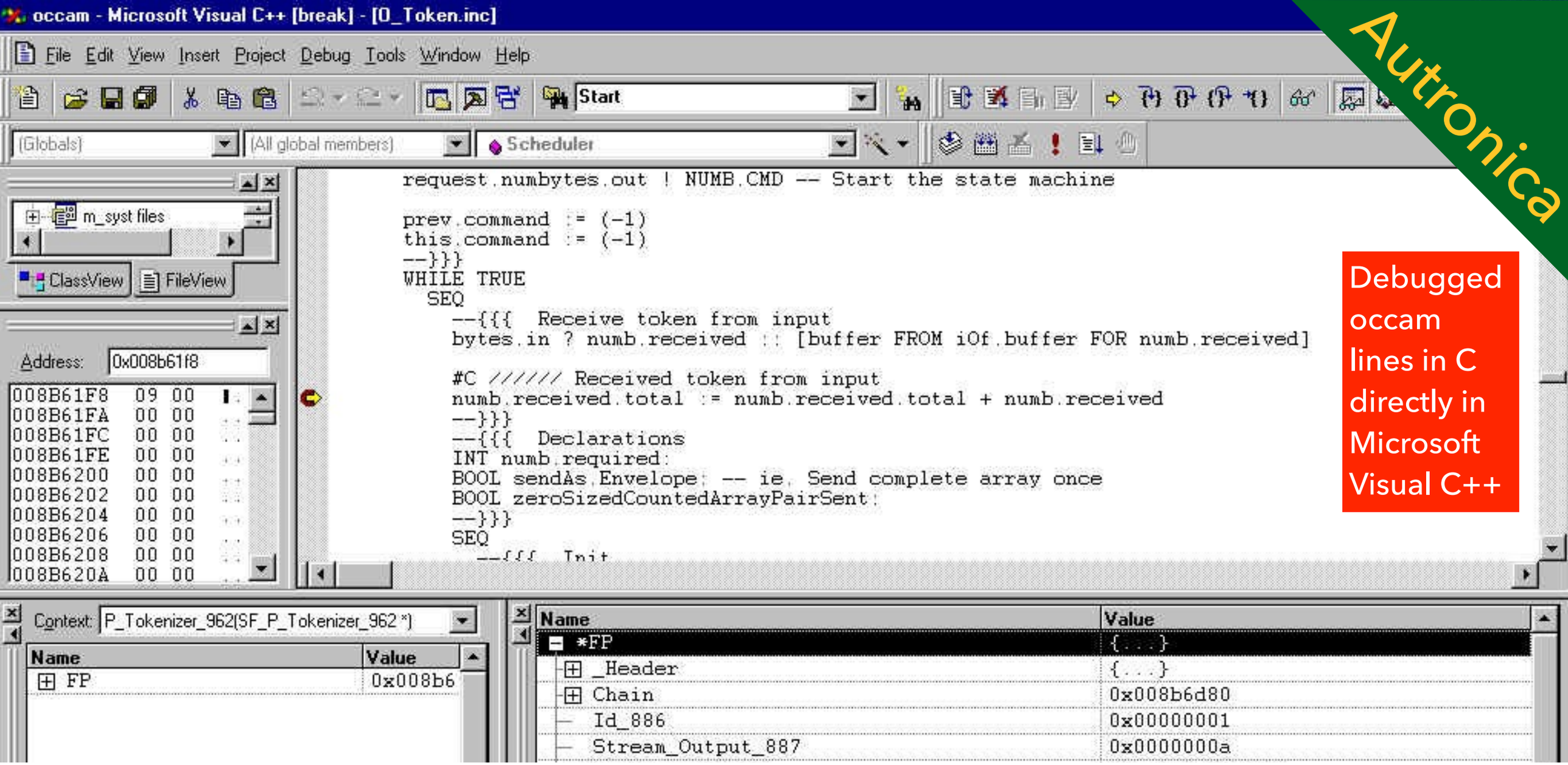
C? YES: OCCAM TO C: SPOC TOOL

1995: OCCAM TO C ON SIGNAL PROCESSOR



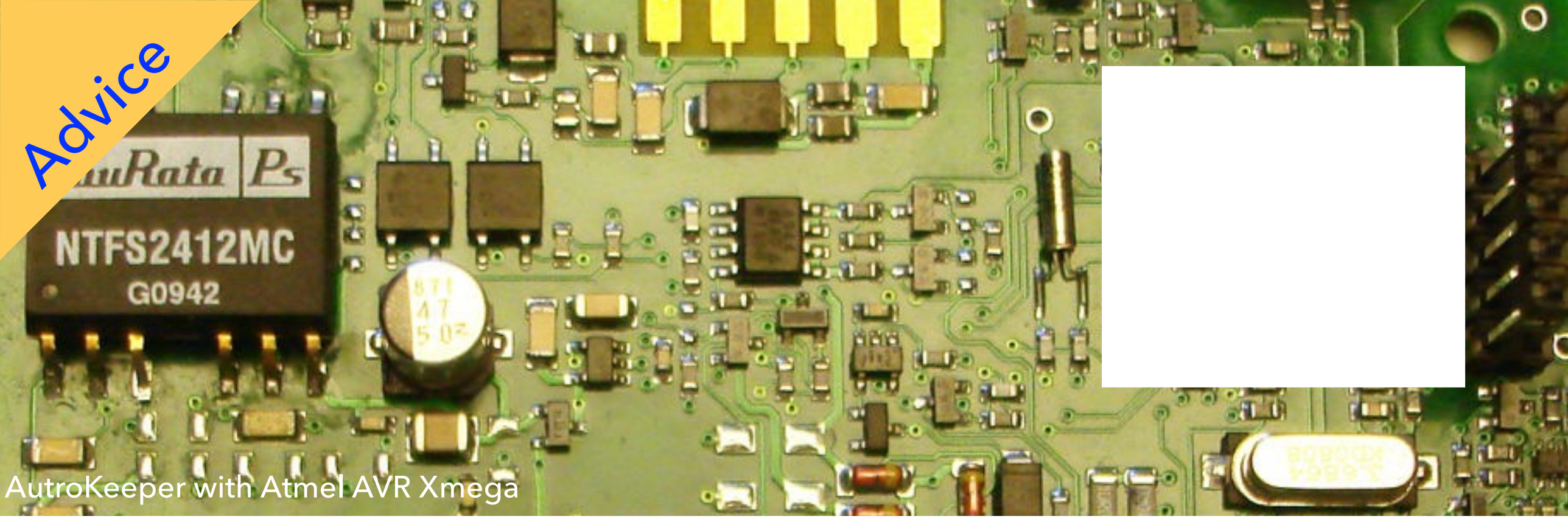
C? YES: OCCAM TO C: SPOC TOOL

1995: OCCAM TO C ON SIGNAL PROCESSOR (MIP-CALCULATOR: NK-200) & NTH DIPLOMA

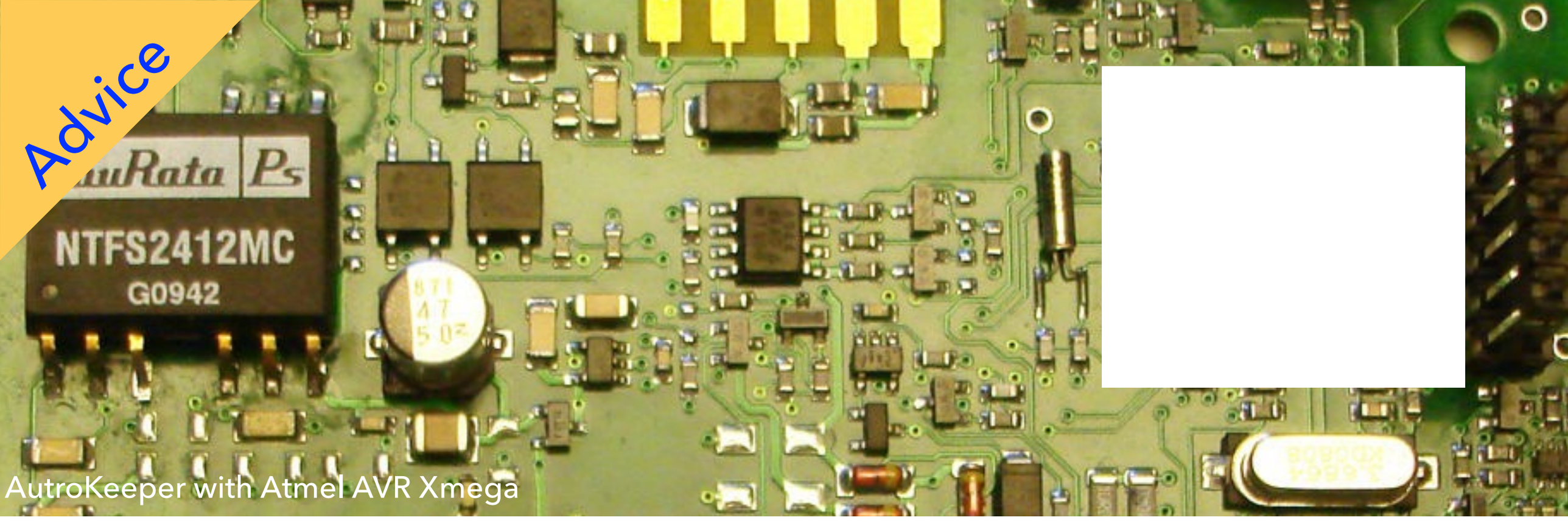


C? YES: OCCAM TO C: SPOC TOOL

1995: OCCAM TO C ON SIGNAL PROCESSOR (MIP-CALCULATOR: NK-200) & NTH DIPLOMA

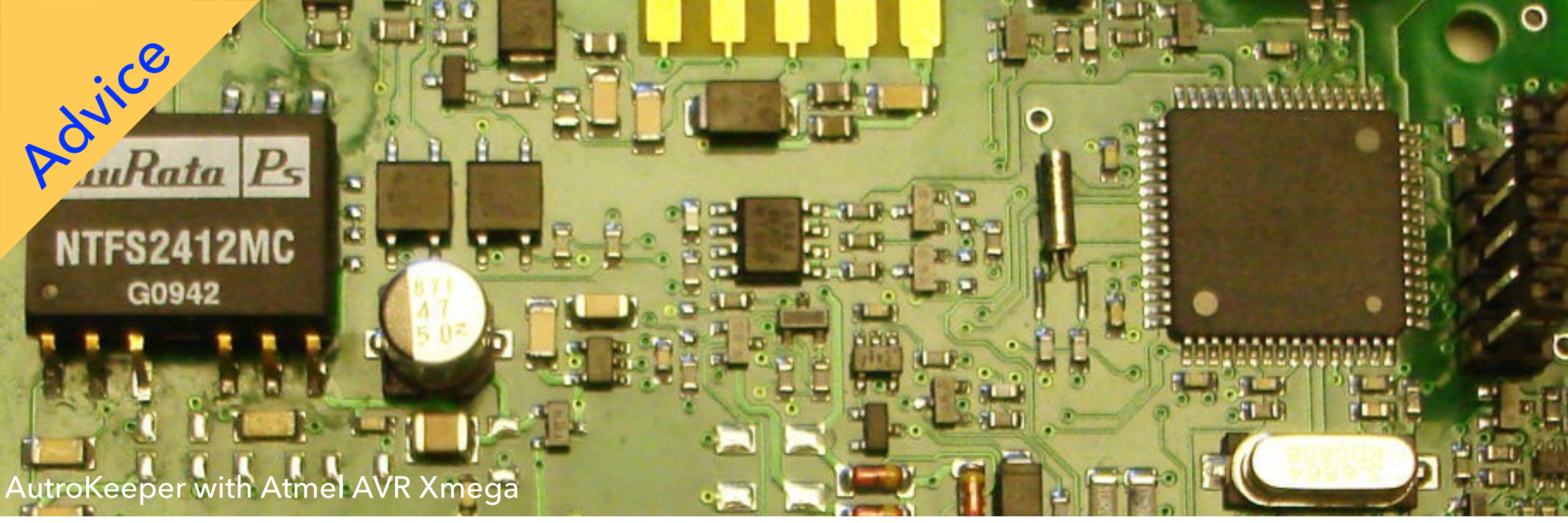


SMALL EMBEDDED SYSTEMS



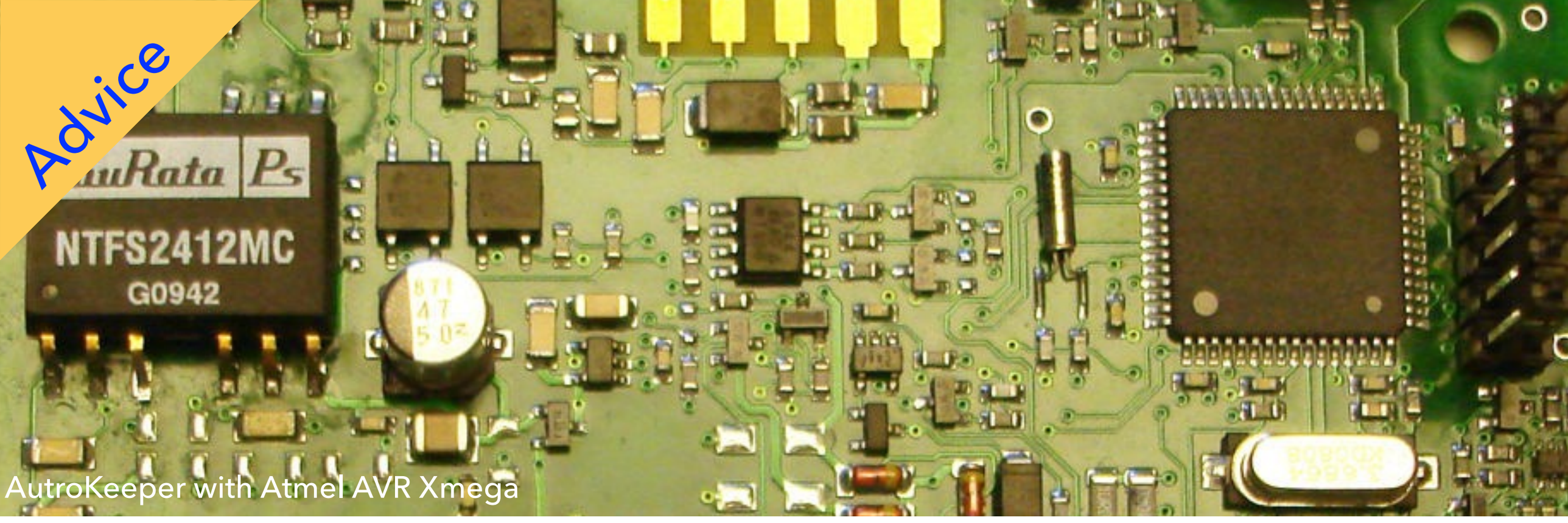
SMALL EMBEDDED SYSTEMS

- Will probably keep C for a long time! We also see C++



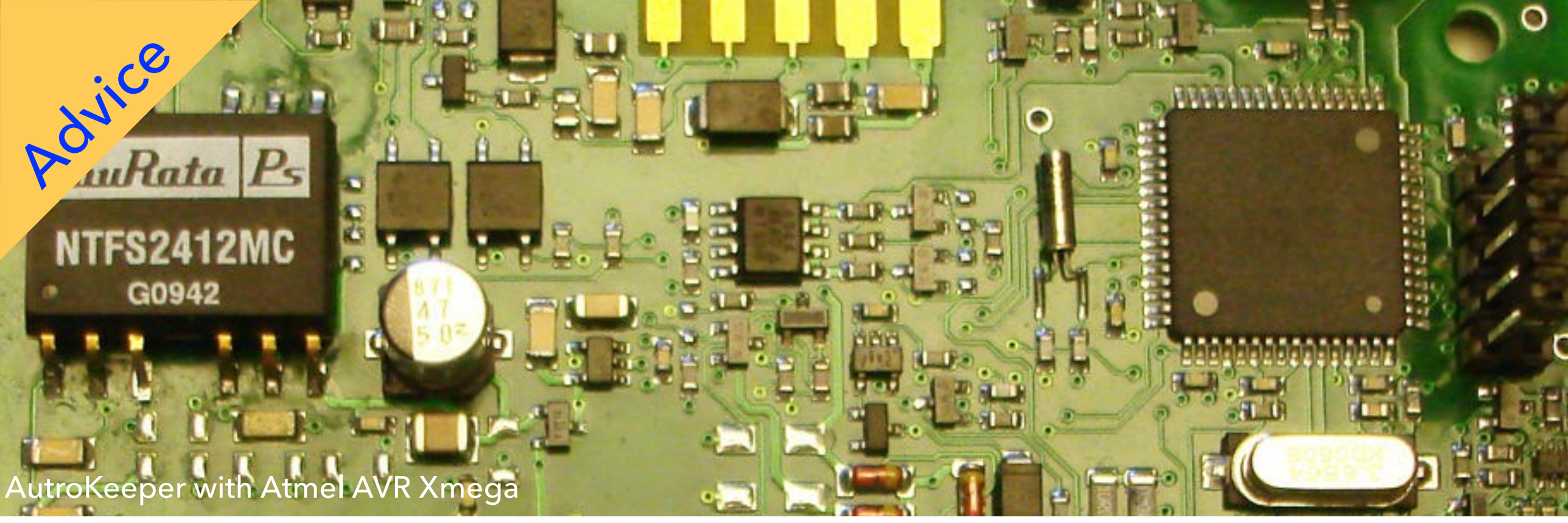
SMALL EMBEDDED SYSTEMS

- Will probably keep C for a long time! We also see C++



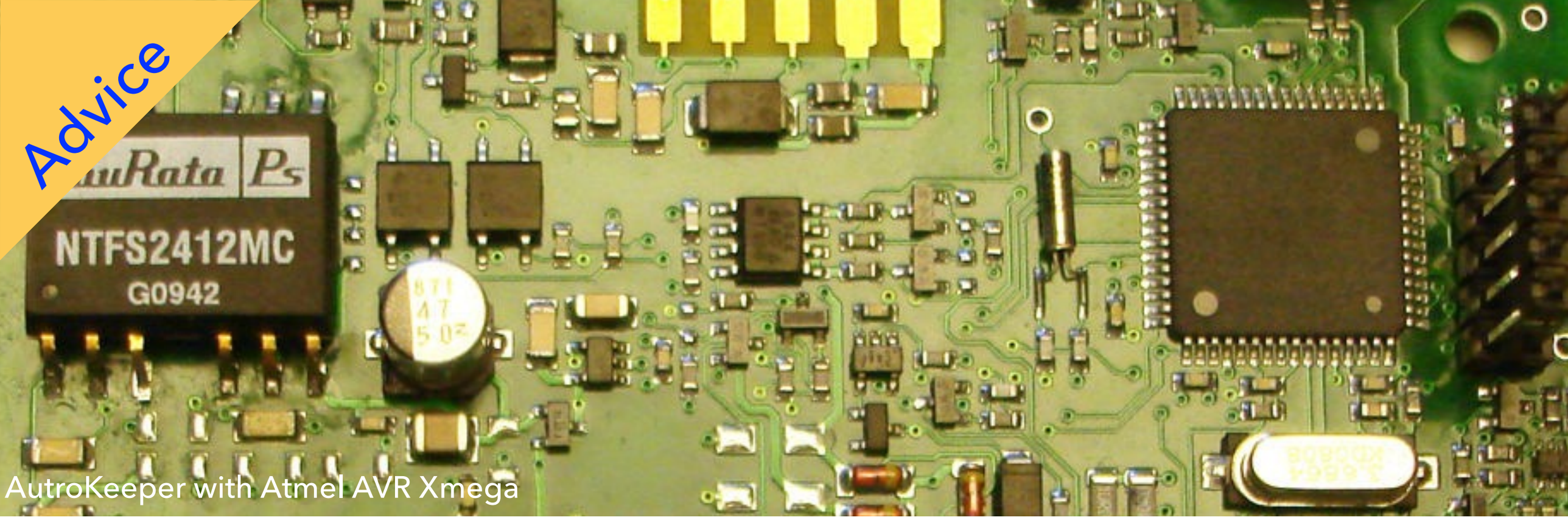
SMALL EMBEDDED SYSTEMS

- Will probably keep C for a long time! We also see C++
- Project managers need to learn about the «Go potential»



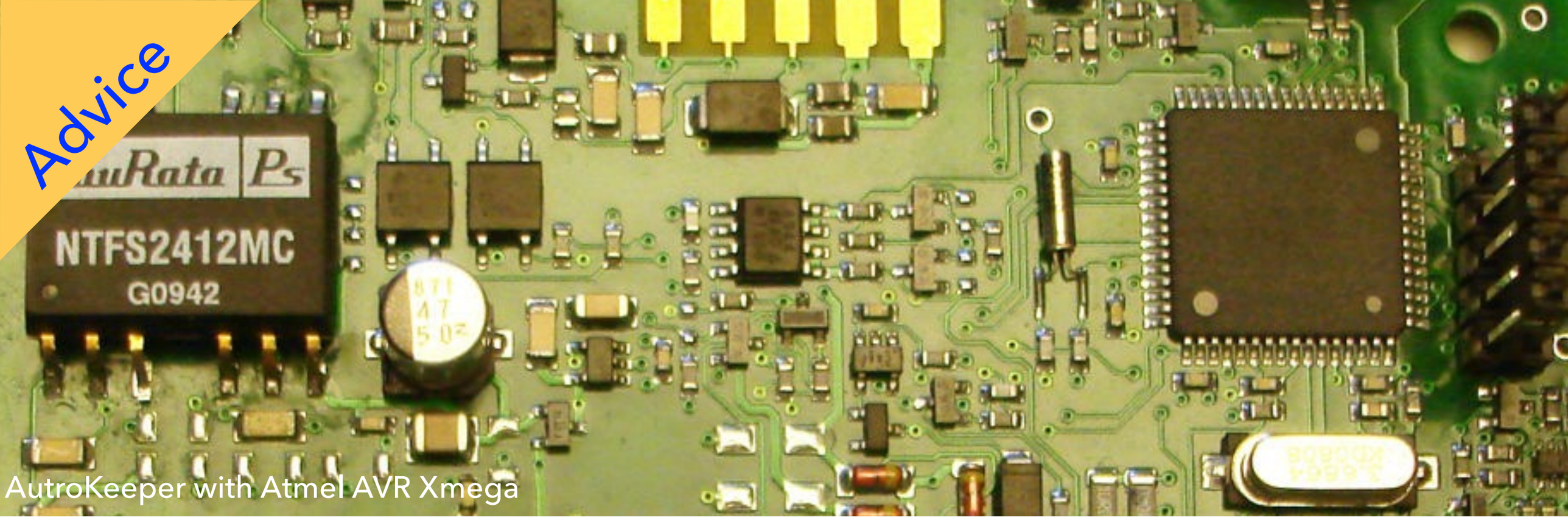
SMALL EMBEDDED SYSTEMS

- Will probably keep C for a long time! We also see C++
- Project managers need to learn about the «Go potential»
- Don’t take over their toolset without adding your knowledge



SMALL EMBEDDED SYSTEMS

- Will probably keep C for a long time! We also see C++
- Project managers need to learn about the «Go potential»
- Don’t take over their toolset without adding your knowledge
  - Like channels and «tight» processes (that protect)



SMALL EMBEDDED SYSTEMS

- Will probably keep C for a long time! We also see C++
- Project managers need to learn about the «Go potential»
- Don't take over their toolset without adding your knowledge
  - Like channels and «tight» processes (that protect)
    - Even if it will be hard to C/C++ schedulers

From a blog note

Which do you mean?

«BLOCKING» EASY TO MISINTERPRET

Which do you mean?

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?

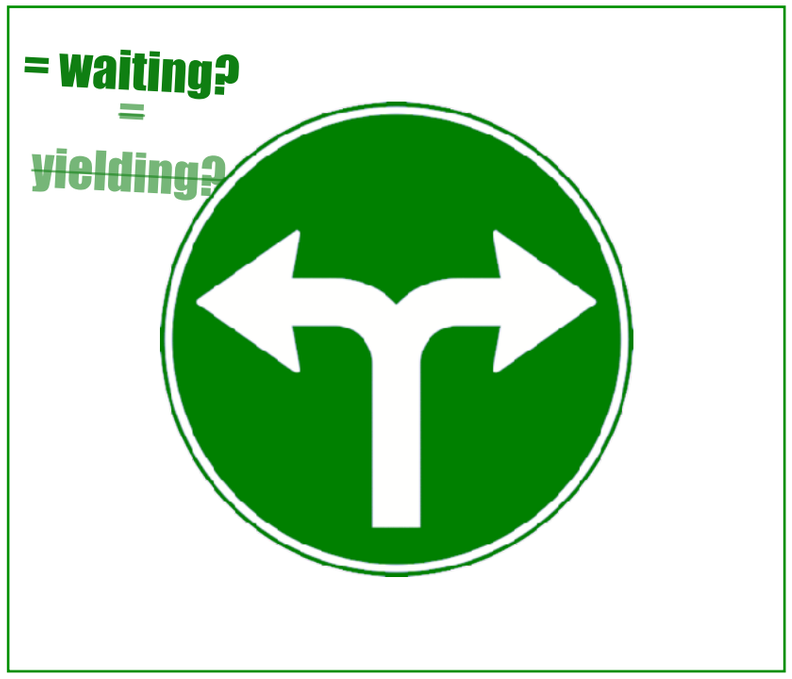
blocking

blocking

blocking

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?



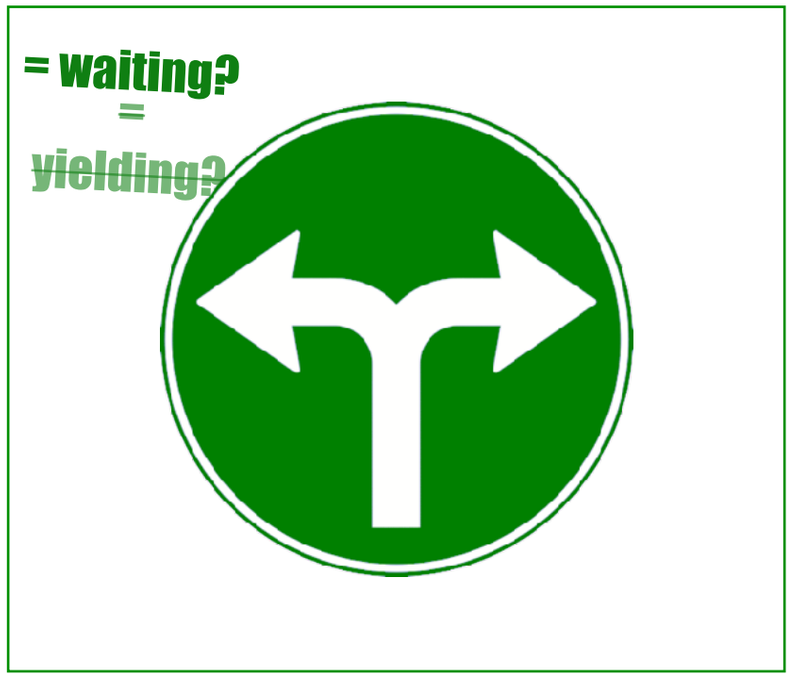
blocking

blocking

blocking

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?



The show goes on with this blocking

blocking

blocking

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?

= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

blocking

blocking

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?

= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

blocking

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?

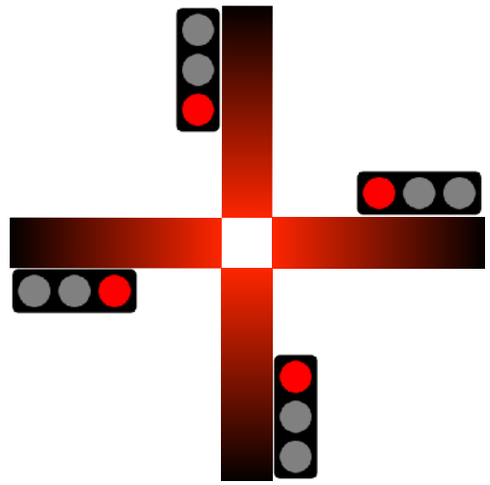
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



blocking

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?

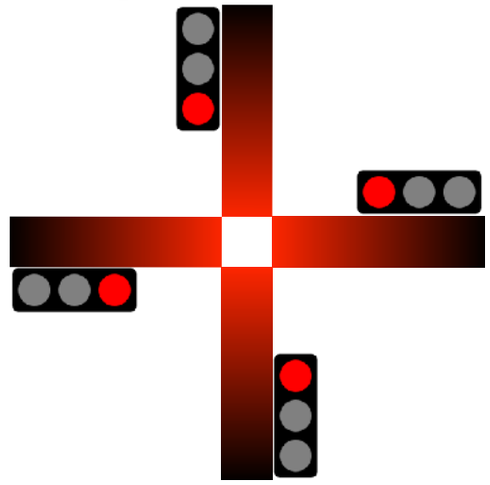
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

Which block ing do you mean?

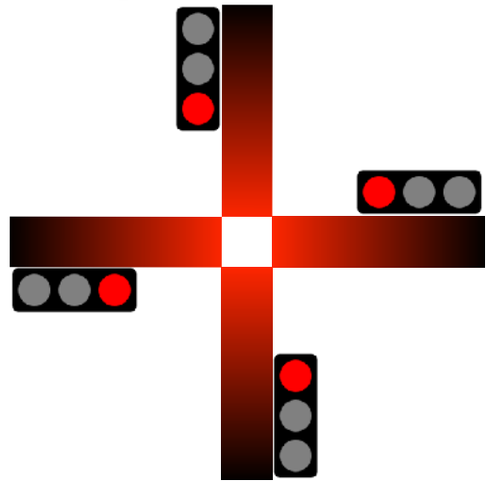
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

- The green channel **blocking** is normal waiting

Which block ing do you mean?

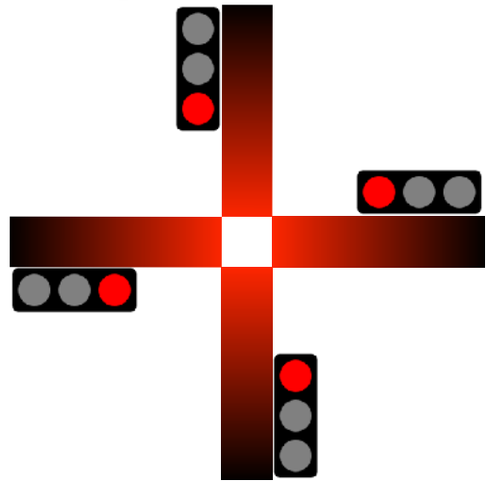
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

- The green channel **blocking** is normal waiting
  - Still called «blocking semantics»

Which block ing do you mean?

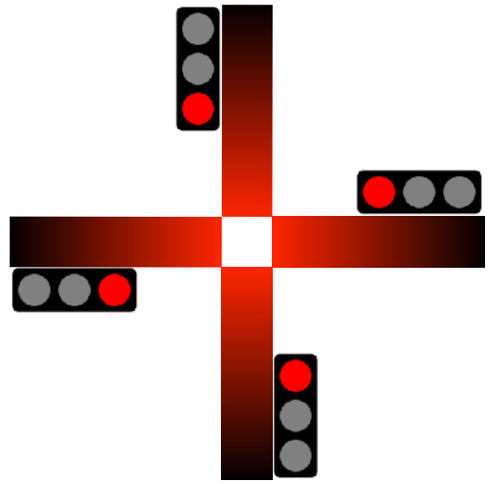
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

- The green channel **blocking** is normal waiting
  - Still called «blocking semantics»
  - We depend on this to make channels «protect» threads!

Which block ing do you mean?

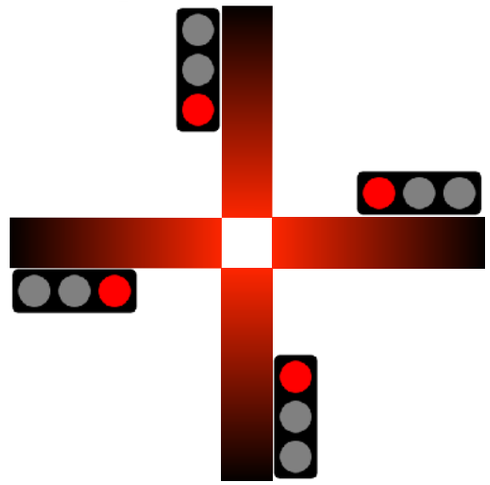
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

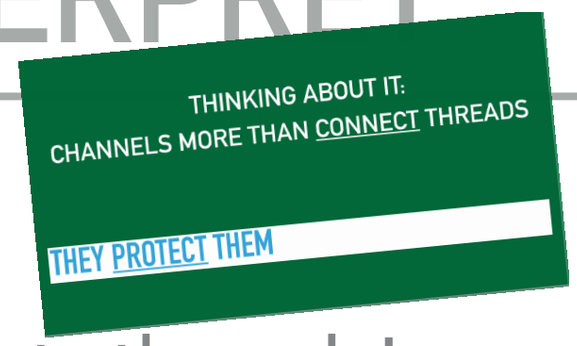
This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET



- The green channel **blocking** is normal waiting
  - Still called «blocking semantics»
  - We depend on this to make channels «protect» threads!

Which block ing do you mean?

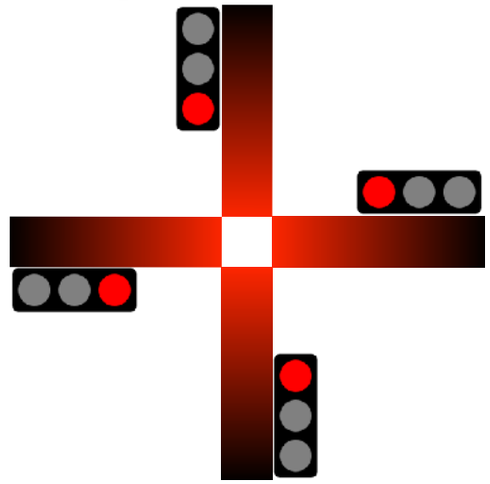
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

- The green channel **blocking** is normal waiting
  - Still called «blocking semantics»
  - We depend on this to make channels «protect» threads!

Which block ing do you mean?

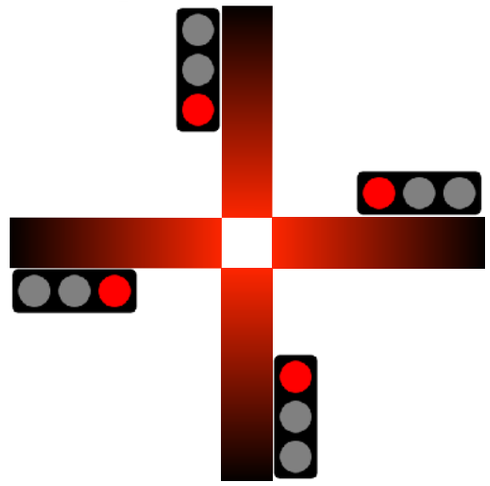
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

- The green channel **blocking** is normal waiting
  - Still called «blocking semantics»
  - We depend on this to make channels «protect» threads!
- The red **blocking** is blocking of others that need to proceed according to specification (too few threads?)

Which block ing do you mean?

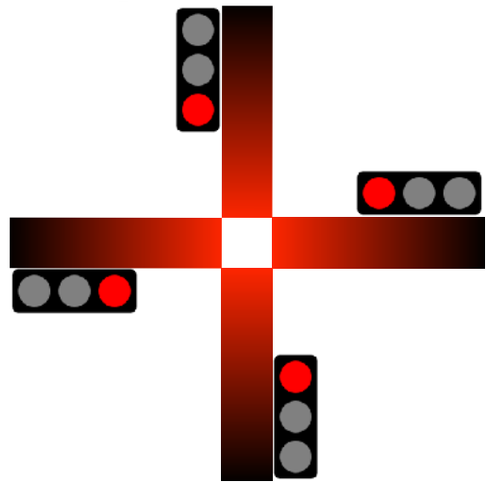
= waiting?
=
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This blocking stops the show

= deadlock!



This blocking stops the world

«BLOCKING» EASY TO MISINTERPRET

- The green channel **blocking** is normal waiting
  - Still called «blocking semantics»
  - We depend on this to make channels «protect» threads!
- The red **blocking** is blocking of others that need to proceed according to specification (too few threads?)
- The black **blocking** is deadlock, pathological, system freeze

THE PROGRAMMING MODEL

THE PROGRAMMING MODEL

- Event loop and callbacks

THE PROGRAMMING MODEL

- Event loop and callbacks
  - Threading often creeps in: problems (shared state, nesting)

THE PROGRAMMING MODEL

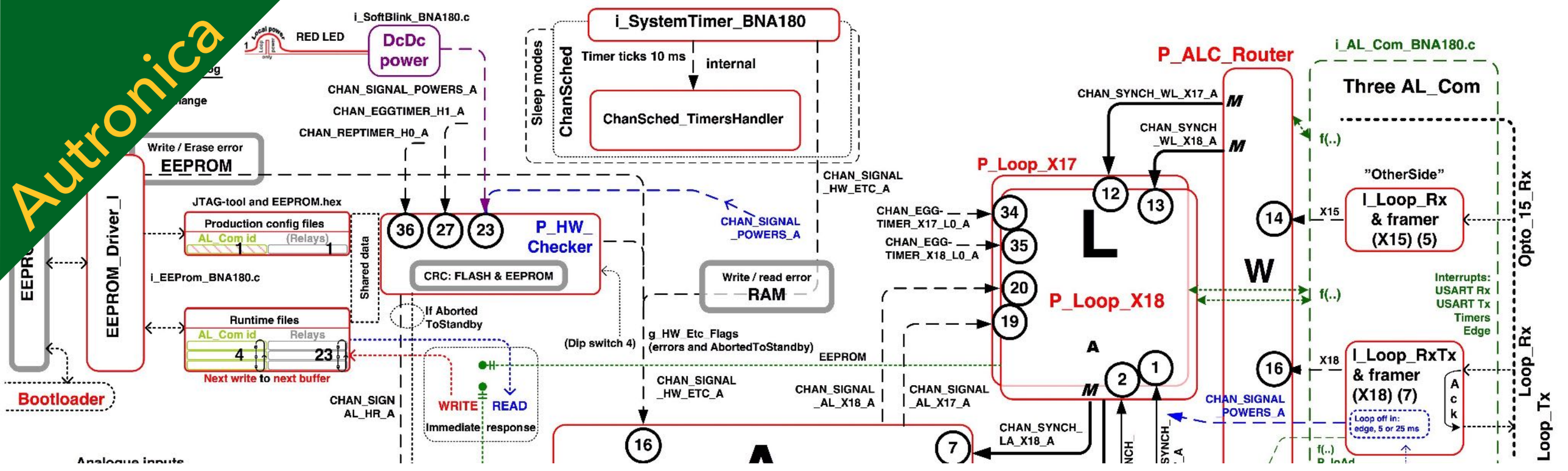
- Event loop and callbacks
  - Threading often creeps in: problems (shared state, nesting)
- Channels and conditional choice (select, alt)

THE PROGRAMMING MODEL

- Event loop and callbacks
  - Threading often creeps in: problems (shared state, nesting)
- Channels and conditional choice (select, alt)
  - In proper processes, concurrency solved

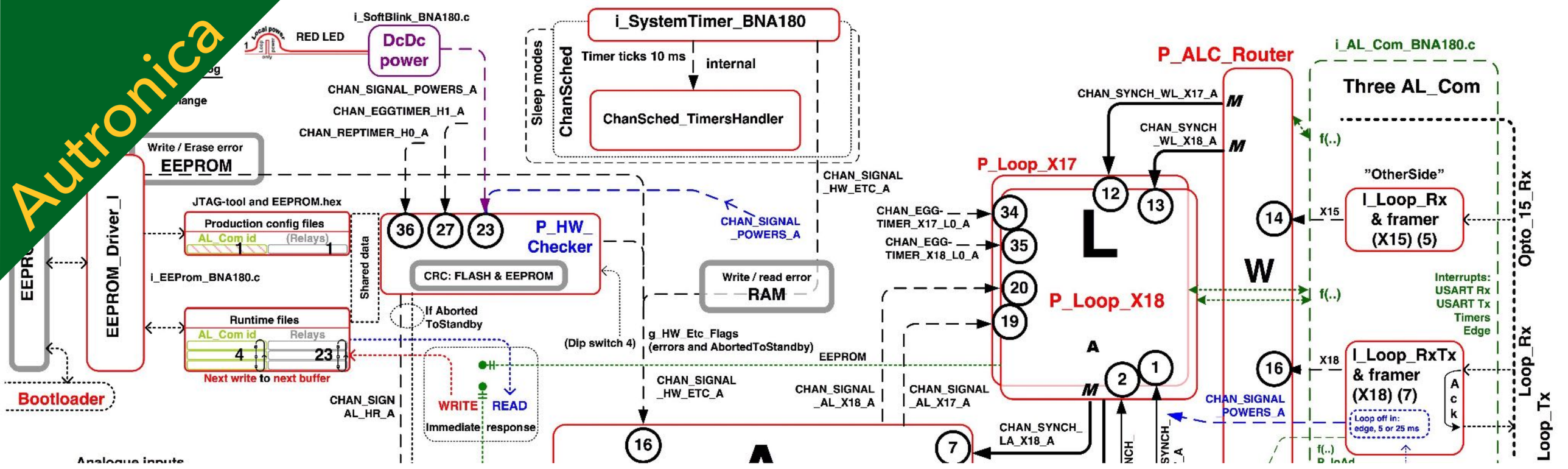
THE PROGRAMMING MODEL

- Event loop and callbacks
  - Threading often creeps in: problems (shared state, nesting)
- Channels and conditional choice (select, alt)
  - In proper processes, concurrency solved
- Connecting channels to event loops and callbacks when that's what you have in a library (like in Closure core.async, see Further reading)



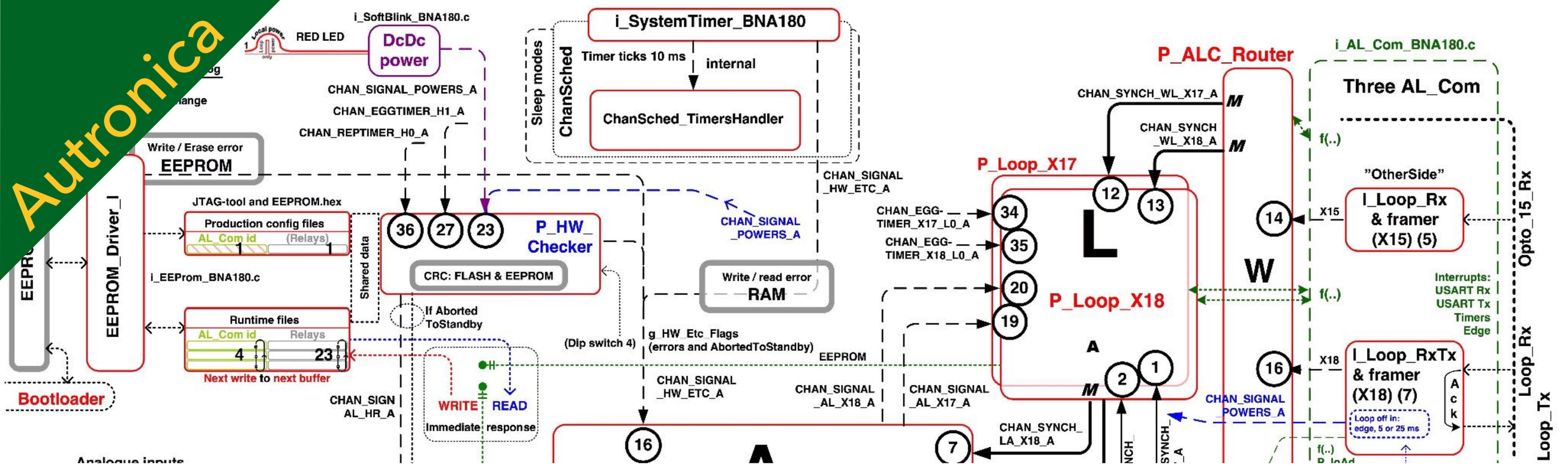
Part of process/data flow diagram

«CHANSCHED»: CSP ON AVR XMEGA



«CHANSCHED»: CSP ON AVR XMEGA

- ChanSched: finally in one of the controllers synchronous channels on top of no other runtime («naked»)



Part of process/data flow diagram

«CHANSCHED»: CSP ON AVR XMEGA

- ChanSched: finally in one of the controllers synchronous channels on top of no other runtime («naked»)
- The runtime was more visible to the application code than I thought (next page)

C CODE ON TOP OF ASYNCH RUNTIME (LEFT) AND NAKED (RIGHT)

C CODE ON TOP OF ASYNCH RUNTIME (LEFT) AND NAKED (RIGHT)

Equal

C CODE ON TOP OF ASYNCH RUNTIME (LEFT) AND NAKED (RIGHT)

Equal

Sync chan comm needs states

C CODE ON TOP OF ASYNCH RUNTIME (LEFT) AND NAKED (RIGHT)

```
void P_Standard_CHAN_CSP (void)
{
  CP_a CP = (CP_a)g_ThisExtPtr; // Application
  switch (CP->State)            // and
                                // communication
                                // state
  {
    case ST_INIT: {/*Init*/ break;}
    case ST_IN:
    {
     CHAN_IN(G_CHAN_IN,CP->Chan_val1);
      CP->State = ST_APPL1;
      break;
    }
    case ST_APPL1:
    {
      // Process val1
      CP->State = ST_OUT;
      break;
    }
    case ST_OUT:
    {
     CHAN_OUT(G_CHAN_OUT,CP->Chan_val1);
      CP->State = ST_IN;
      break;
    }

  }
}
```

Equal

Sync chan comm needs states

C CODE ON TOP OF ASYNCH RUNTIME (LEFT) AND NAKED (RIGHT)

```
void P_Standard_CHAN_CSP (void)
{
  CP_a CP = (CP_a)g_ThisExtPtr; // Application
  switch (CP->State)            // and
                                // communication
                                // state
  {
    case ST_INIT: {/*Init*/ break;}
    case ST_IN:
    {
     CHAN_IN(G_CHAN_IN,CP->Chan_val1);
      CP->State = ST_APPL1;
      break;
    }
    case ST_APPL1:
    {
      // Process val1
      CP->State = ST_OUT;
      break;
    }
    case ST_OUT:
    {
     CHAN_OUT(G_CHAN_OUT,CP->Chan_val1);
      CP->State = ST_IN;
      break;
    }

  }
}
```

Equal

Sync chan comm needs states

Synchronisation points no visible state

C CODE ON TOP OF ASYNCH RUNTIME (LEFT) AND NAKED (RIGHT)

```
void P_Standard_CHAN_CSP (void)
{
  CP_a CP = (CP_a)g_ThisExtPtr; // Application
  switch (CP->State)            // and
                                // communication
                                // state
  {
    case ST_INIT: {/*Init*/ break;}
    case ST_IN:
    {
     CHAN_IN(G_CHAN_IN,CP->Chan_val1);
      CP->State = ST_APPL1;
      break;
    }
    case ST_APPL1:
    {
      // Process val1
      CP->State = ST_OUT;
      break;
    }
    case ST_OUT:
    {
     CHAN_OUT(G_CHAN_OUT,CP->Chan_val1);
      CP->State = ST_IN;
      break;
    }

  }
}
```

Sync chan comm needs states

Equal

```
void P_Extended_ChanSched (void)
{
  CP_a CP = (CP_a)g_ThisExtPtr; // Application
  // Init here                  // state only
  while (TRUE)
  {
    switch (CP->State)
    {
      case ST_MAIN:
      {
        CHAN_IN(G_CHAN_IN,CP->Chan_val2);

        // Process val2

        CHAN_OUT(G_CHAN_OUT,CP->Chan_val2);
        CP->State = ST_MAIN; // option1
        break;
      }
    }
  }
}
```

Synchronisation points no visible state

SAME CODE IN A LIBRARY AND OCCAM

```
void P_libcsp2 (Channel *in, Channel *out)
{
  int val3;
  for(;;)
  {
    ChanInInt (in, &val3);
    // Process val3
    ChanOutInt (out, val3);
  }
}
```

```
PROC P_occam (CHAN OF INT in, out)

  WHILE TRUE
  INT val4:
    SEQ
      in ? val4
      -- Process val4
      out ! val4
:
```

A TYPICAL `ChanSched` PROCESS BODY (OVERVIEW)

A TYPICAL `ChanSched` PROCESS BODY (OVERVIEW)

```
1.  Void P_Prefix (void)                          // extended "Prefix"
2.  {
3.    Prefix_CP_a CP = (Prefix_CP_a)g_CP; // get process Context from Scheduler
4.    PROCTOR_PREFIX()                            // jump table (see Section 2)
5.    ...  some initialisation
6.    SET_EGGTIMER (CHAN_EGGTIMER, LED_Timeout_Tick);
7.    SET_REPTIMER (CHAN_REPTIMER, ADC_TIME_TICKS);
8.    CHAN_OUT (CHAN_DATA_0, Data_0);  // first output
9.    while (TRUE)
10.   {
23.   }
24. }
```

A TYPICAL `ChanSched` PROCESS BODY (OVERVIEW)

```
Void P_Prefix (void)                      // extended “Prefix”
{
  Prefix_CP_a CP = (Prefix_CP_a)g_CP; // get process Context from Scheduler
  PROCTOR_PREFIX()                        // jump table (see Section 2)
  ...  some initialisation
  SET_EGGTIMER (CHAN_EGGTIMER, LED_Timeout_Tick);
  SET_REPTIMER (CHAN_REPTIMER, ADC_TIME_TICKS);
  CHAN_OUT (CHAN_DATA_0, Data_0);  // first output
  while (TRUE)
  {
    ALT();                                // this is the needed ”PRI_ALT”
    ALT_END();
  }
}
```

A TYPICAL ChanSched PROCESS BODY (OVERVIEW)

```
1.  Void P_Prefix (void)                          // extended "Prefix"
2.  {
3.    Prefix_CP_a CP = (Prefix_CP_a)g_CP; // get process Context from Scheduler
4.    PROCTOR_PREFIX()                            // jump table (see Section 2)
5.    ...  some initialisation
6.    SET_EGGTIMER (CHAN_EGGTIMER, LED_Timeout_Tick);
7.    SET_REPTIMER (CHAN_REPTIMER, ADC_TIME_TICKS);
8.    CHAN_OUT (CHAN_DATA_0, Data_0);  // first output
9.    while (TRUE)
10.   {
11.     ALT();                                    // this is the needed "PRI_ALT"
12.       ALT_EGGREPTIMER_IN (CHAN_EGGTIMER);
13.       ALT_EGGREPTIMER_IN (CHAN_REPTIMER);
14.       ALT_SIGNAL_CHAN_IN (CHAN_SIGNAL_AD_READY);
15.       ALT_CHAN_IN        (CHAN_DATA_2, Data_2);
16.       ALT_ALTTIMER_IN    (CHAN_ALTTIMER, TIME_TICKS_100_MSECS);
17.     ALT_END();
23.   }
24. }
```

A TYPICAL ChanSched PROCESS BODY (OVERVIEW)

```
1.  Void P_Prefix (void)                          // extended "Prefix"
2.  {
3.    Prefix_CP_a CP = (Prefix_CP_a)g_CP; // get process Context from Scheduler
4.    PROCTOR_PREFIX()                            // jump table (see Section 2)
5.    ...  some initialisation
6.    SET_EGGTIMER (CHAN_EGGTIMER, LED_Timeout_Tick);
7.    SET_REPTIMER (CHAN_REPTIMER, ADC_TIME_TICKS);
8.    CHAN_OUT (CHAN_DATA_0, Data_0);  // first output
9.    while (TRUE)
10.   {
11.     ALT();                                  // this is the needed "PRI_ALT"
12.       ALT_EGGREPTIMER_IN (CHAN_EGGTIMER);
13.       ALT_EGGREPTIMER_IN (CHAN_REPTIMER);
14.       ALT_SIGNAL_CHAN_IN (CHAN_SIGNAL_AD_READY);
15.       ALT_CHAN_IN        (CHAN_DATA_2, Data_2);
16.       ALT_ALTTIMER_IN    (CHAN_ALTTIMER, TIME_TICKS_100_MSECS);
17.     ALT_END();
18.     switch (g_ThisChannelId)
19.     {
20.       ...  process the guard that has been taken, e.g. CHAN_DATA_2
21.       CHAN_OUT (CHAN_DATA_0, Data_0);
22.     };
23.   }
24. }
```

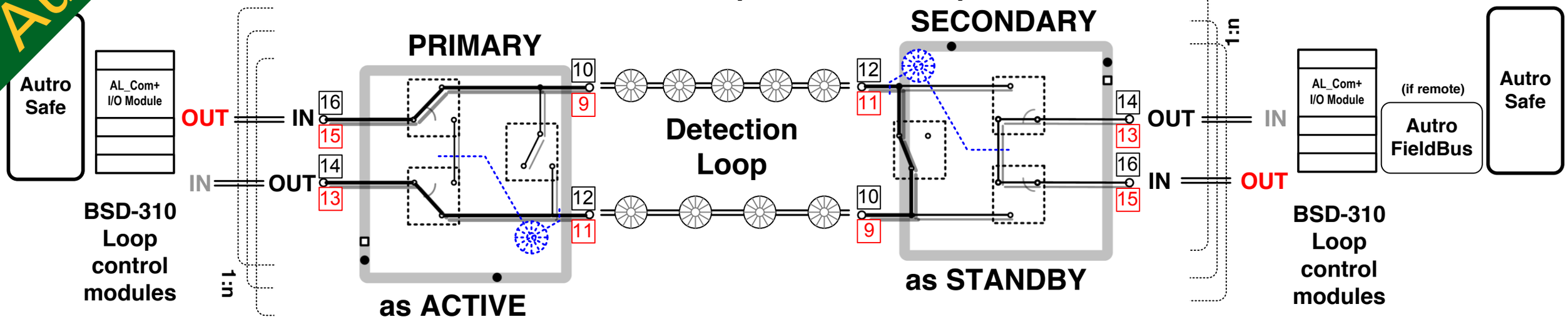
http://www.teigfam.net/oyvind/pub/pub_details.html#NewALT

A TYPICAL `ChanSched` PROCESS BODY (OVERVIEW)

```
1.  Void P_Prefix (void)                          // extended "Prefix"
2.  {
3.    Prefix_CP_a CP = (Prefix_CP_a)g_CP; // get process Context from Scheduler
4.    PROCTOR_PREFIX()                            // jump table (see Section 2)
5.    ...  some initialisation
6.    SET_EGGTIMER (CHAN_EGGTIMER, LED_Timeout_Tick);
7.    SET_REPTIMER (CHAN_REPTIMER, ADC_TIME_TICKS);
8.    CHAN_OUT (CHAN_DATA_0, Data_0);  // first output
9.    while (TRUE)
10.   {
11.     ALT();                                  // this is the needed "PRI_ALT"
12.       ALT_EGGREPTIMER_IN (CHAN_EGGTIMER);
13.       ALT_EGGREPTIMER_IN (CHAN_REPTIMER);
14.       ALT_SIGNAL_CHAN_IN (CHAN_SIGNAL_AD_READY);
15.       ALT_CHAN_IN        (CHAN_DATA_2, Data_2);
16.       ALT_ALTTIMER_IN    (CHAN_ALTTIMER, TIME_TICKS_100_MSECS);
17.     ALT_END();
18.     switch (g_ThisChannelId)
19.     {
20.       ...  process the guard that has been taken, e.g. CHAN_DATA_2
21.       CHAN_OUT (CHAN_DATA_0, Data_0);
22.     };
23.   }
24. }
```

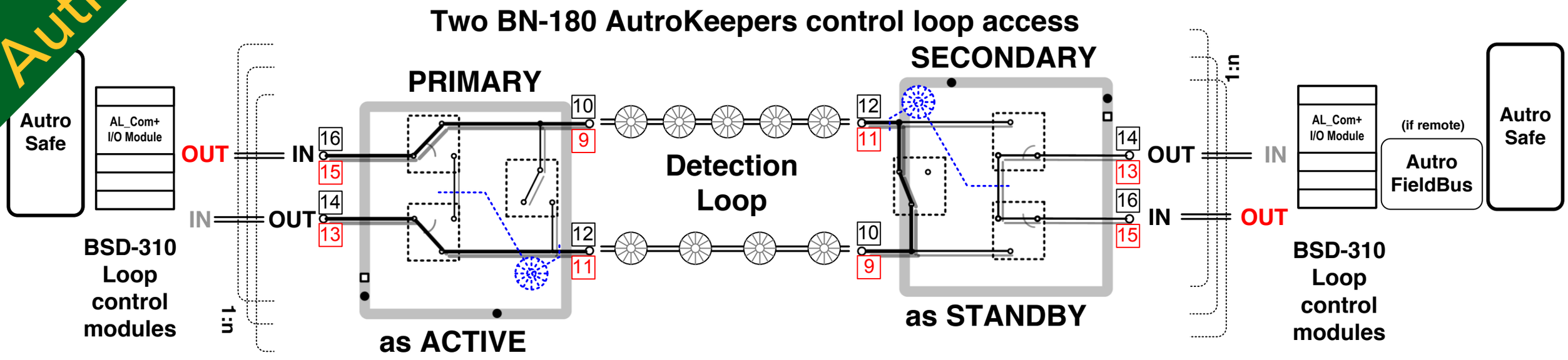
Autronica

Two BN-180 AutroKeepers control loop access



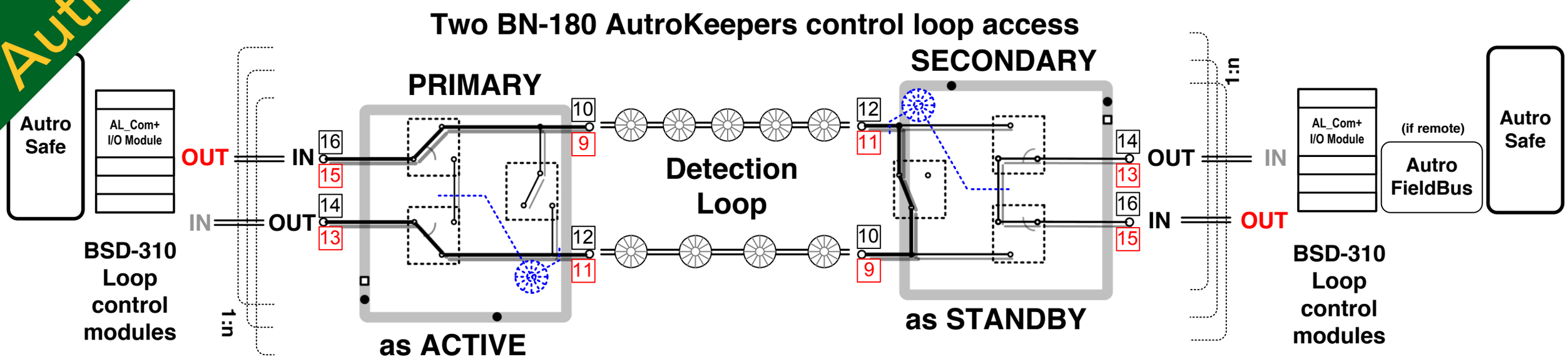
Autronica

Also from real life



Autronica

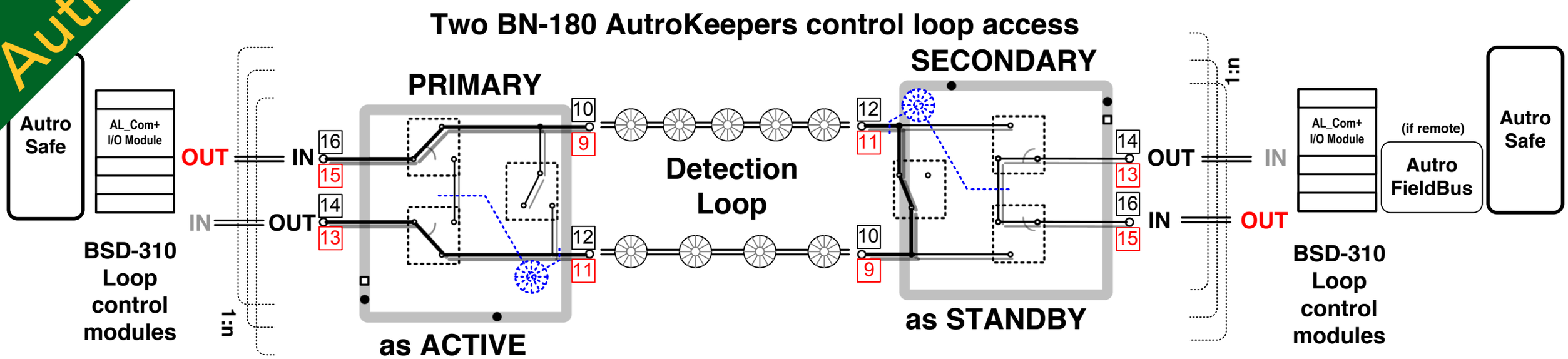
Also from real life



WITH CSP & FDR4, PROMELA & SPIN ETC.

Autronica

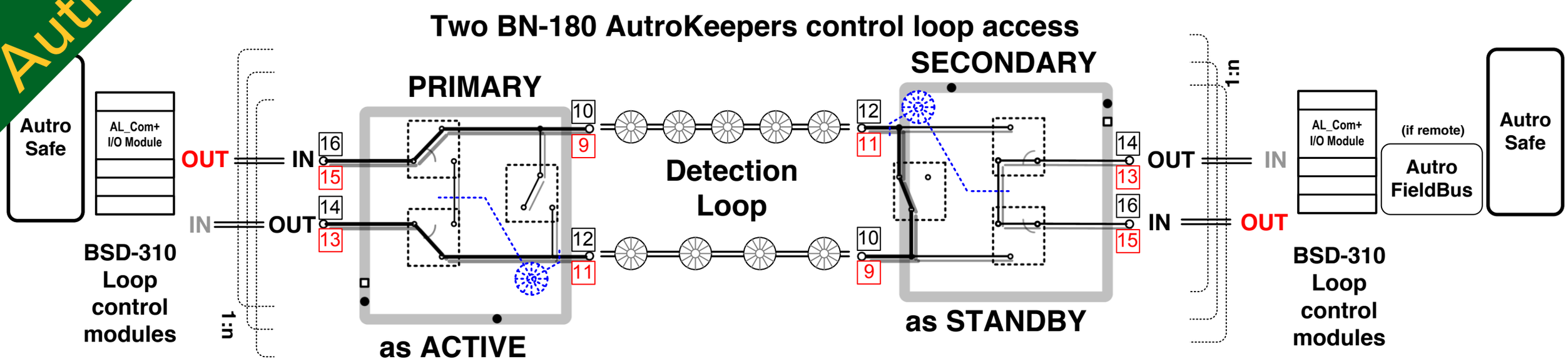
Also from real life



WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

Also from real life



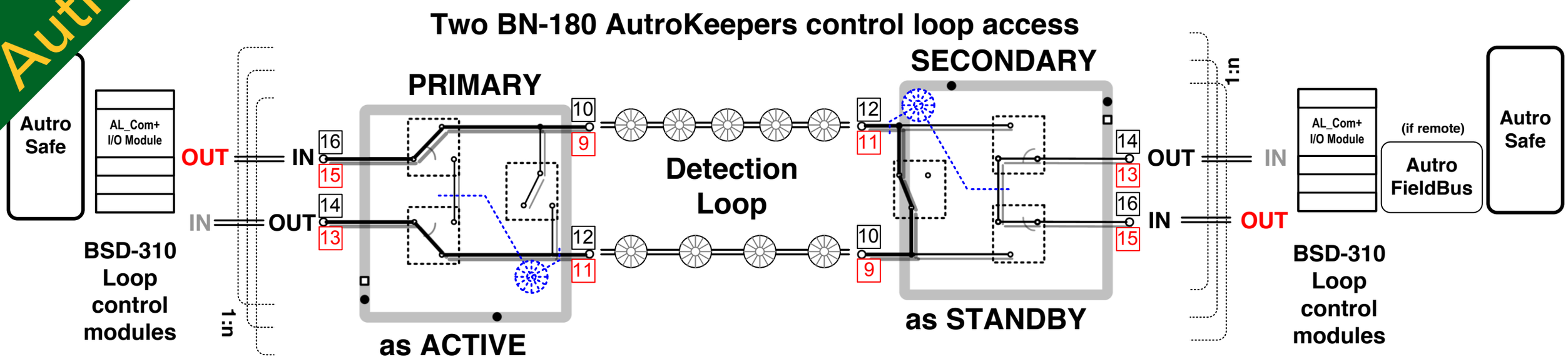
WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles

Autronica

Also from real life



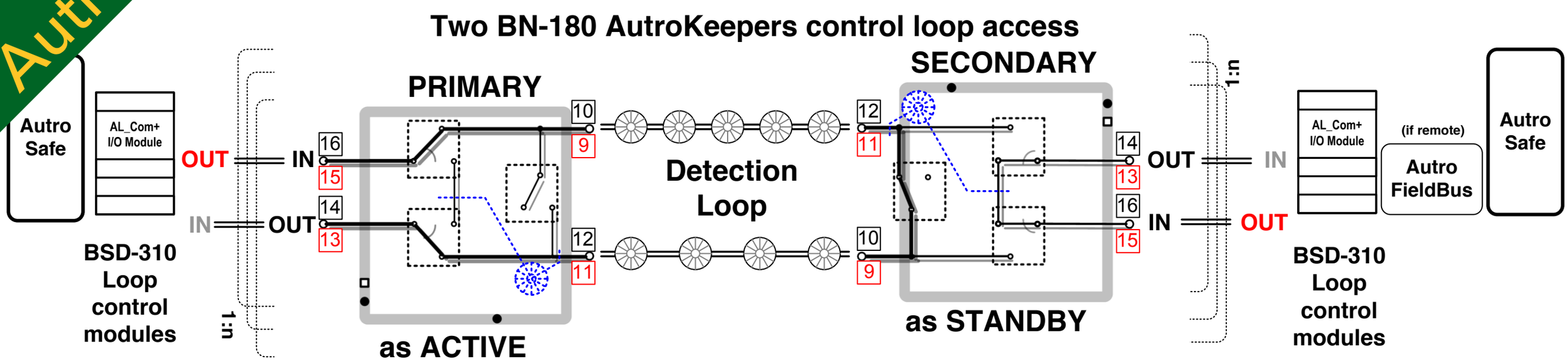
WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop

Autronica

Also from real life

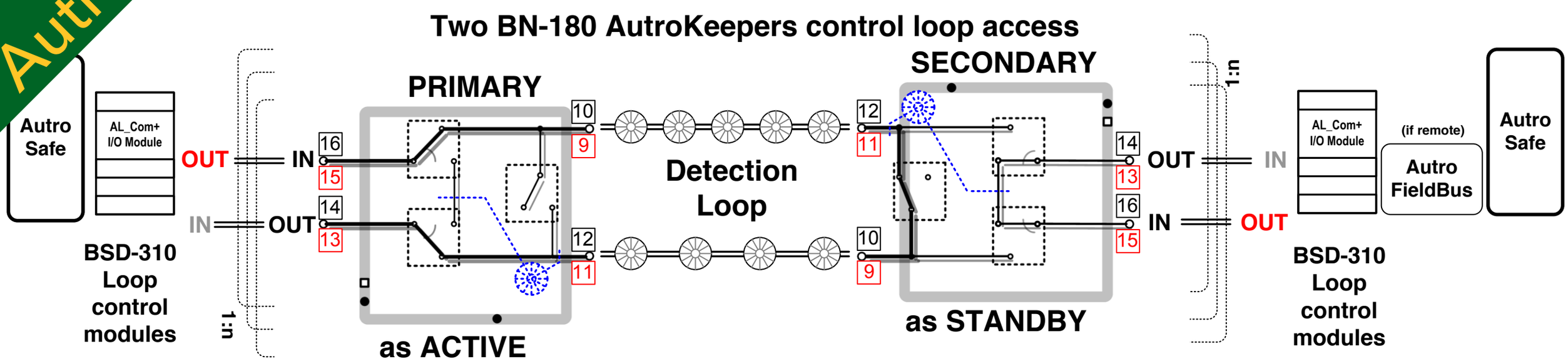


WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop
- Always one side connected

Also from real life



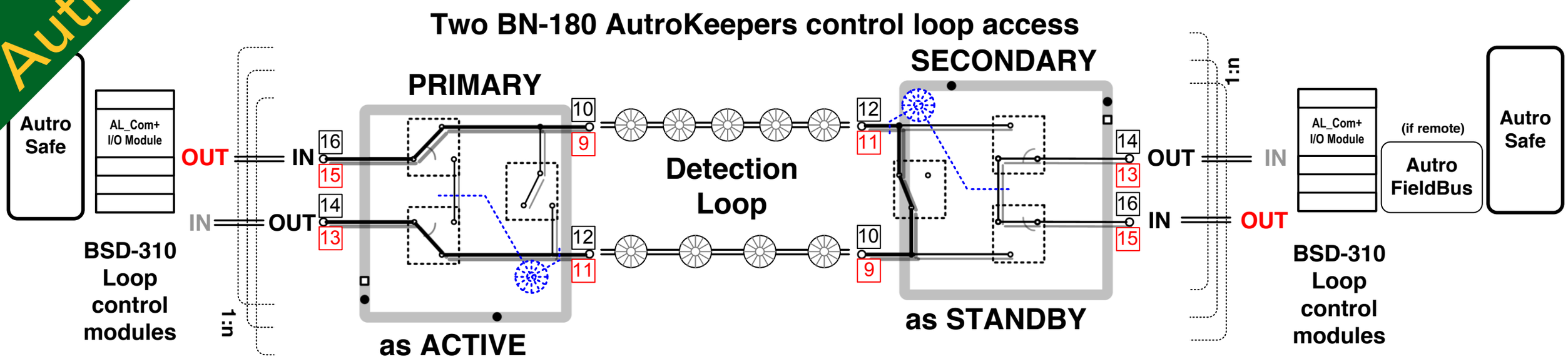
WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop
- Always one side connected
- No oscillations

Autronica

Also from real life



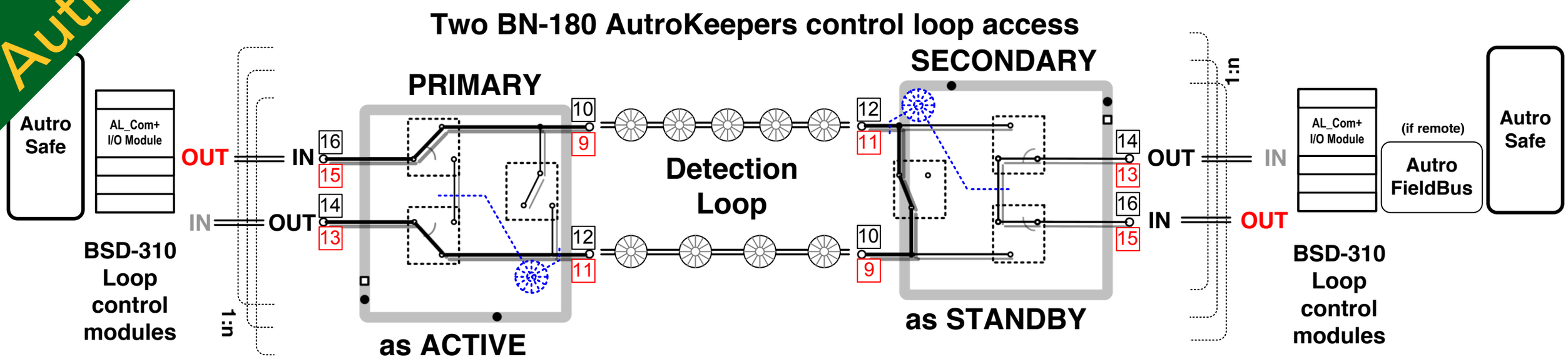
WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop
- Always one side connected
- No oscillations
- Keeps track of the sanity and possibilities of each side

Autronica

Also from real life

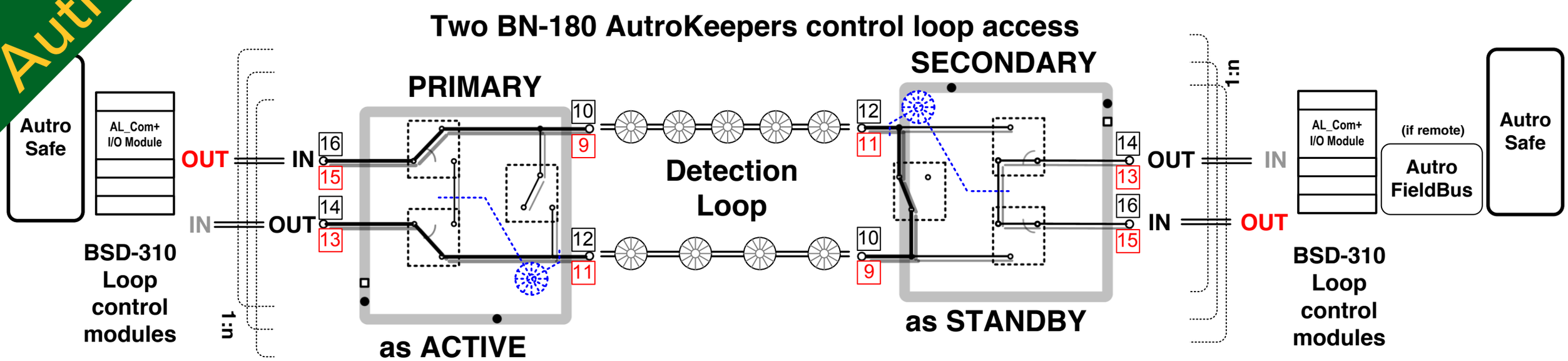


WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop
- Always one side connected
- No oscillations
- Keeps track of the sanity and possibilities of each side
- Switches over in milliseconds when needed

Also from real life



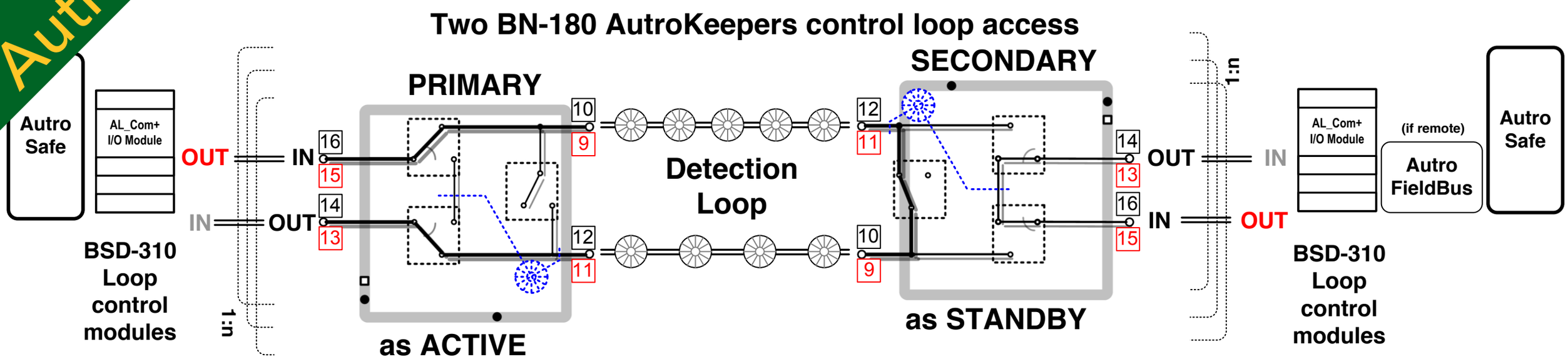
WITH CSP & FDR4, PROMELA & SPIN ETC.

FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop
- Always one side connected
- No oscillations
- Keeps track of the sanity and possibilities of each side
- Switches over in milliseconds when needed
- Formal model gave us roles and protocol elements

Autronica

Also from real life



WITH CSP & FDR4, PROMELA & SPIN ETC.

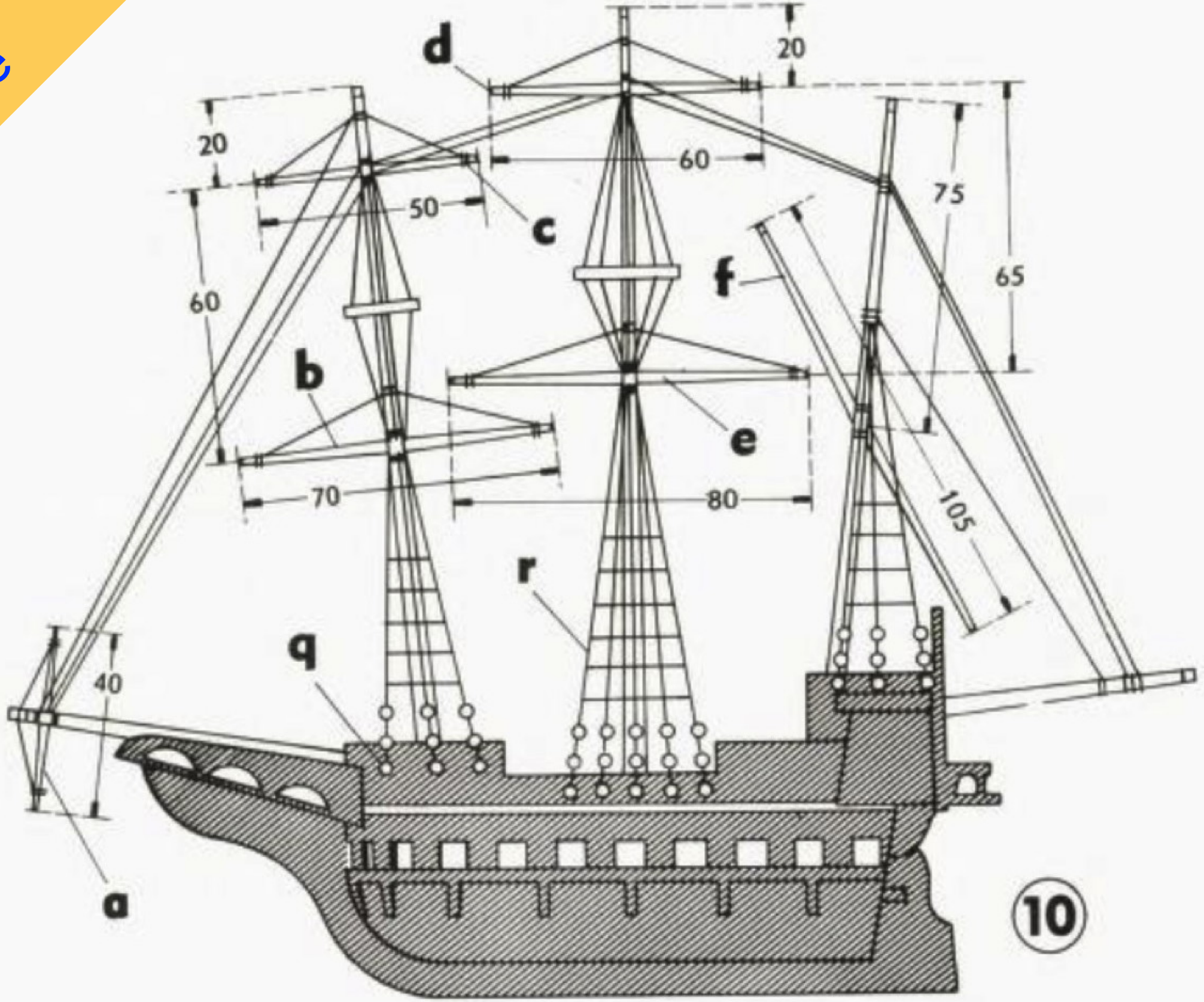
FORMAL MODELING

- Like, modeling of roles
- Safe, not simultaneous dual access of detector loop
- Always one side connected
- No oscillations
- Keeps track of the sanity and possibilities of each side
- Switches over in milliseconds when needed
- Formal model gave us roles and protocol elements

produkt.autronica.no/fileshare/filArkivRoot/produkt/pdf/dokumentasjon/bn180_cn.pdf

Final advice

Final advice



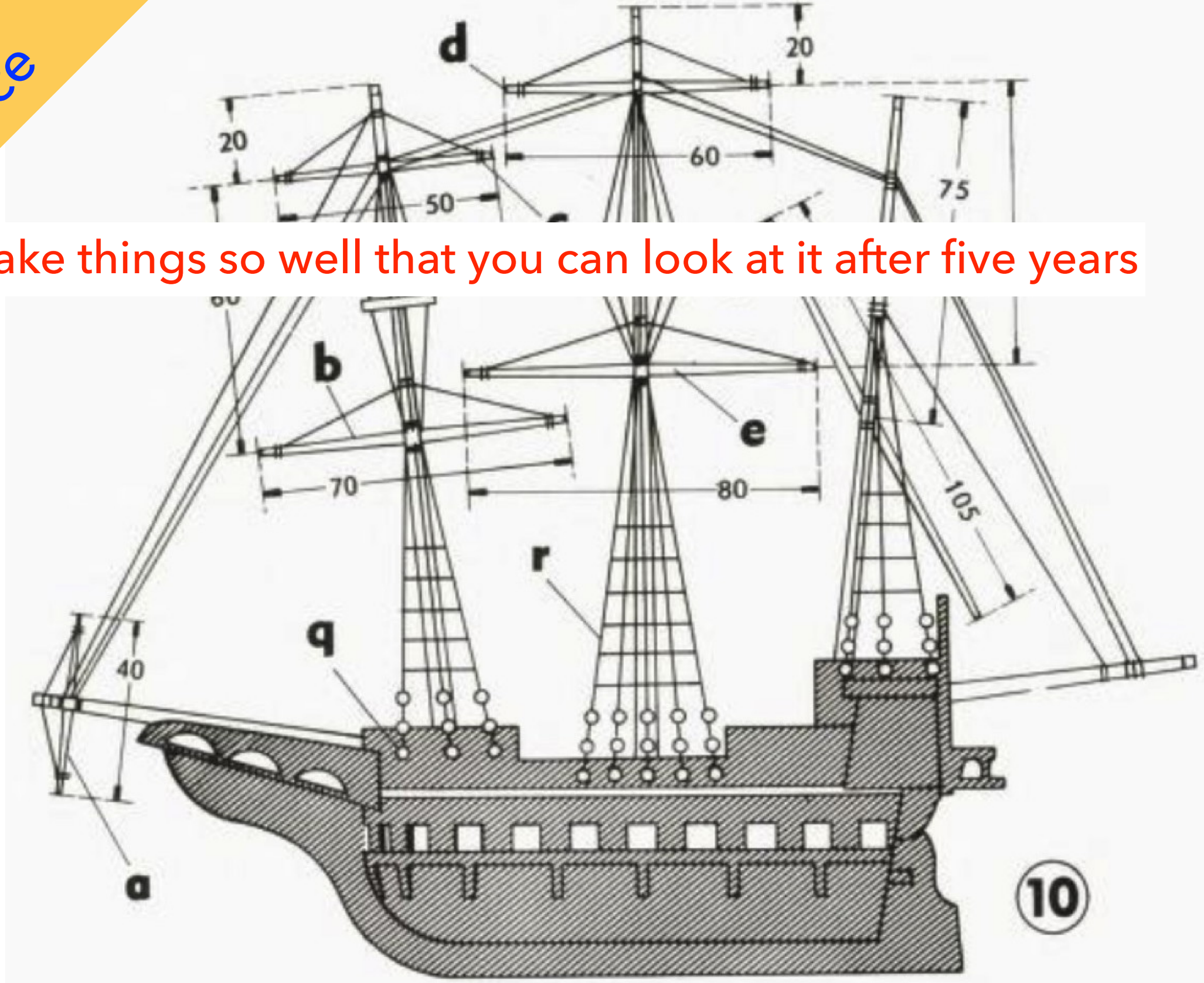
Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice

Make things so well that you can look at it after five years



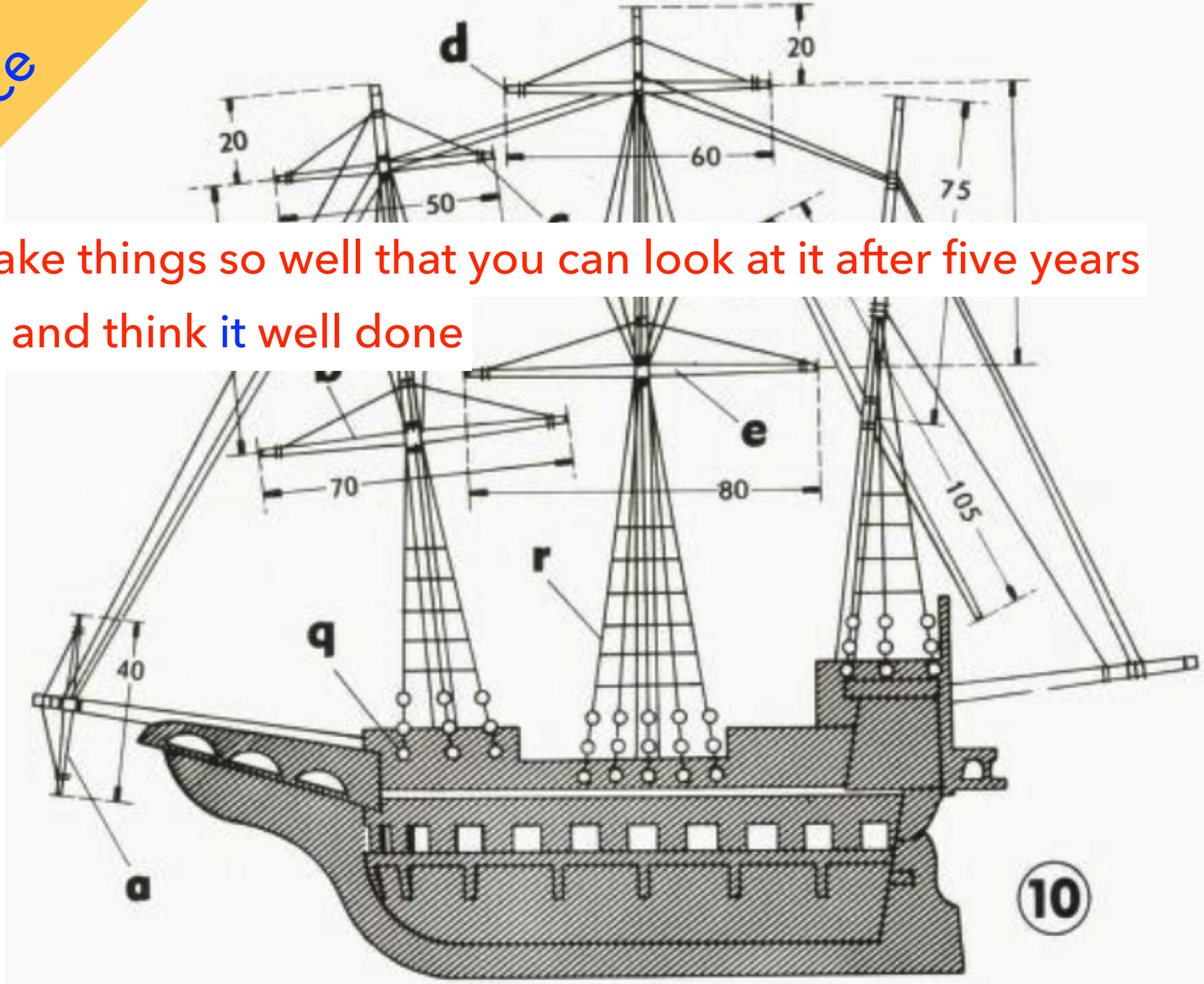
Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice

Make things so well that you can look at it after five years and think it well done



Master, spryd og rær

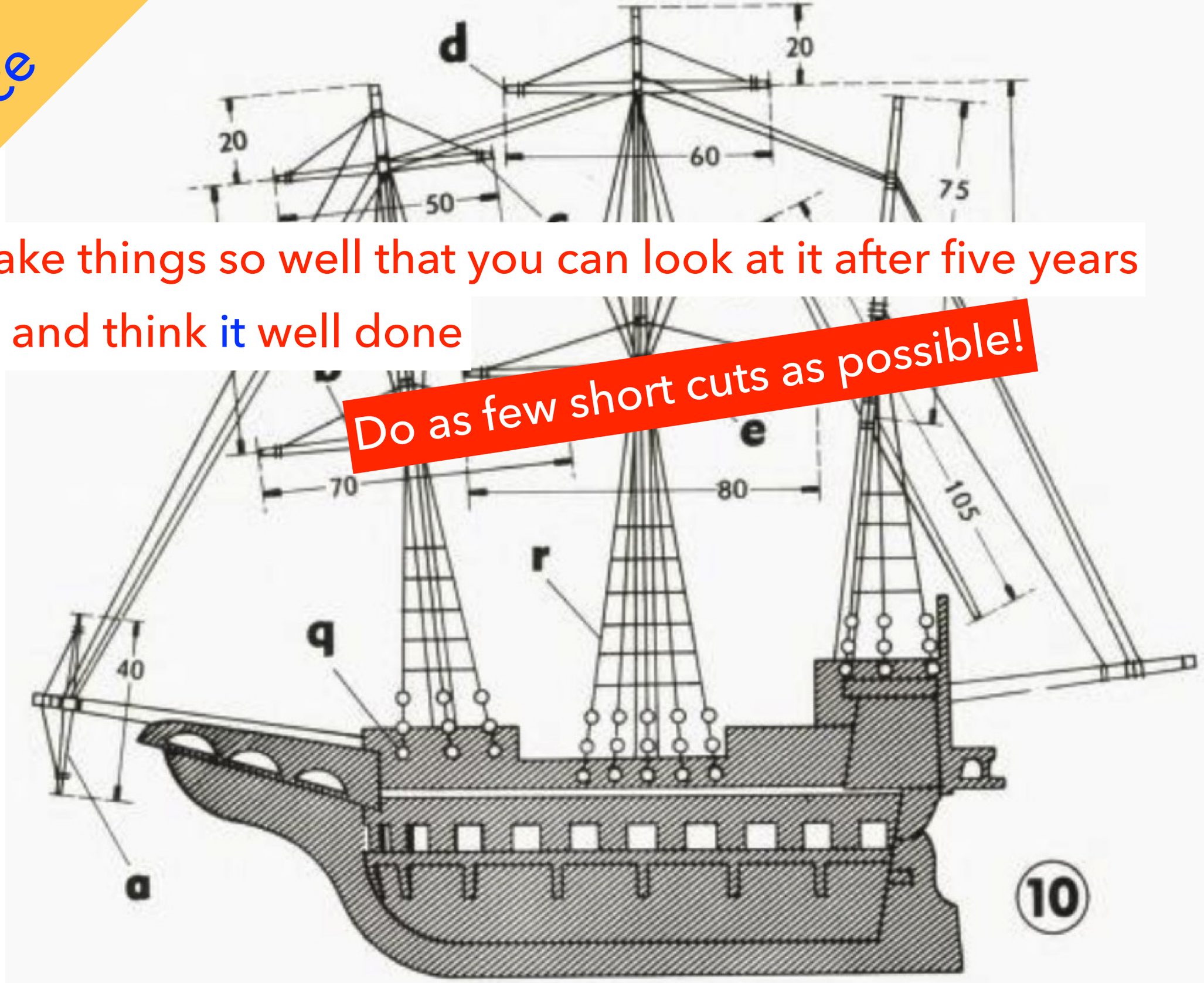
Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice

Make things so well that you can look at it after five years and think it well done

Do as few short cuts as possible!



Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

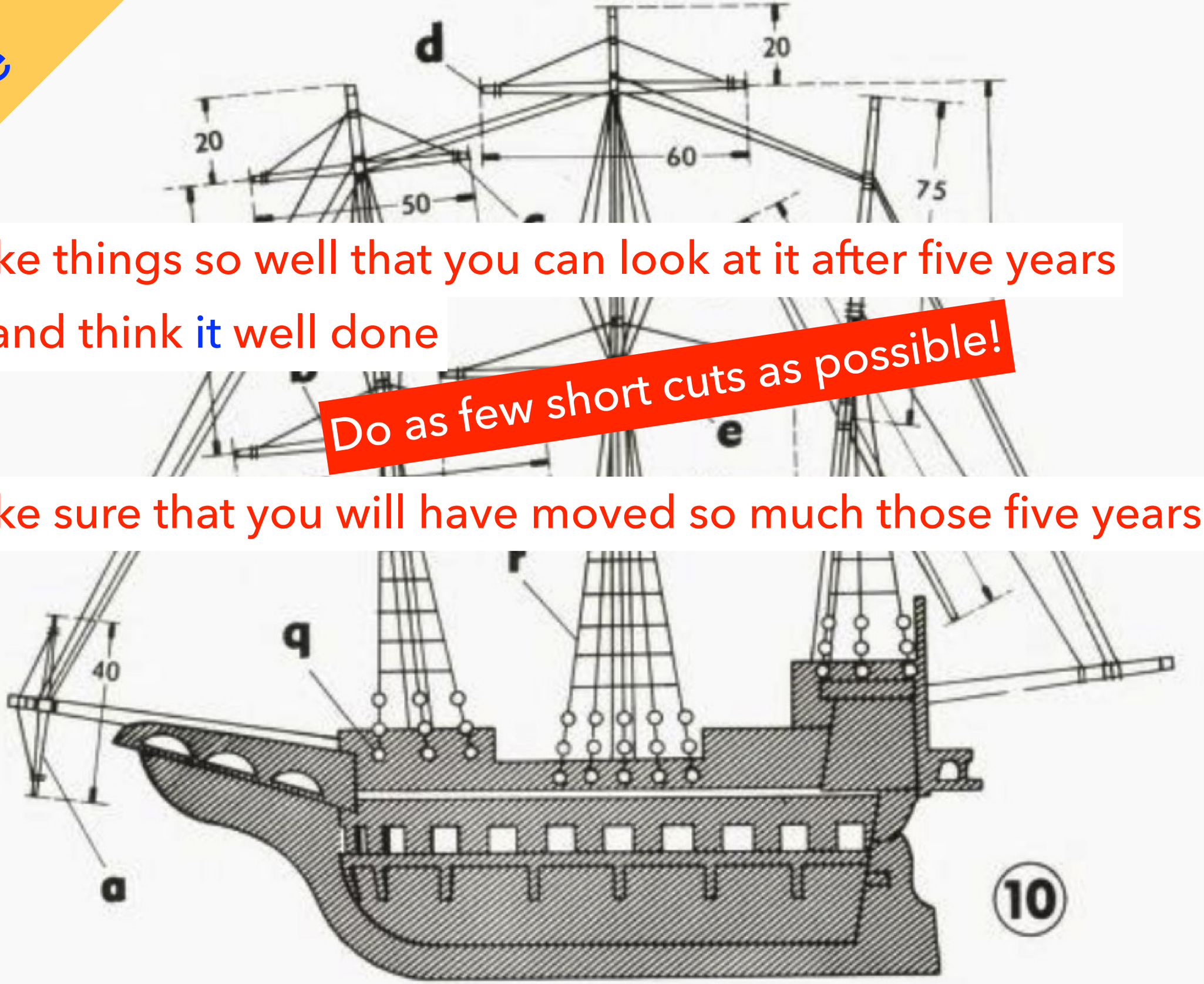
Final advice

Make things so well that you can look at it after five years

and think it well done

Do as few short cuts as possible!

Make sure that you will have moved so much those five years

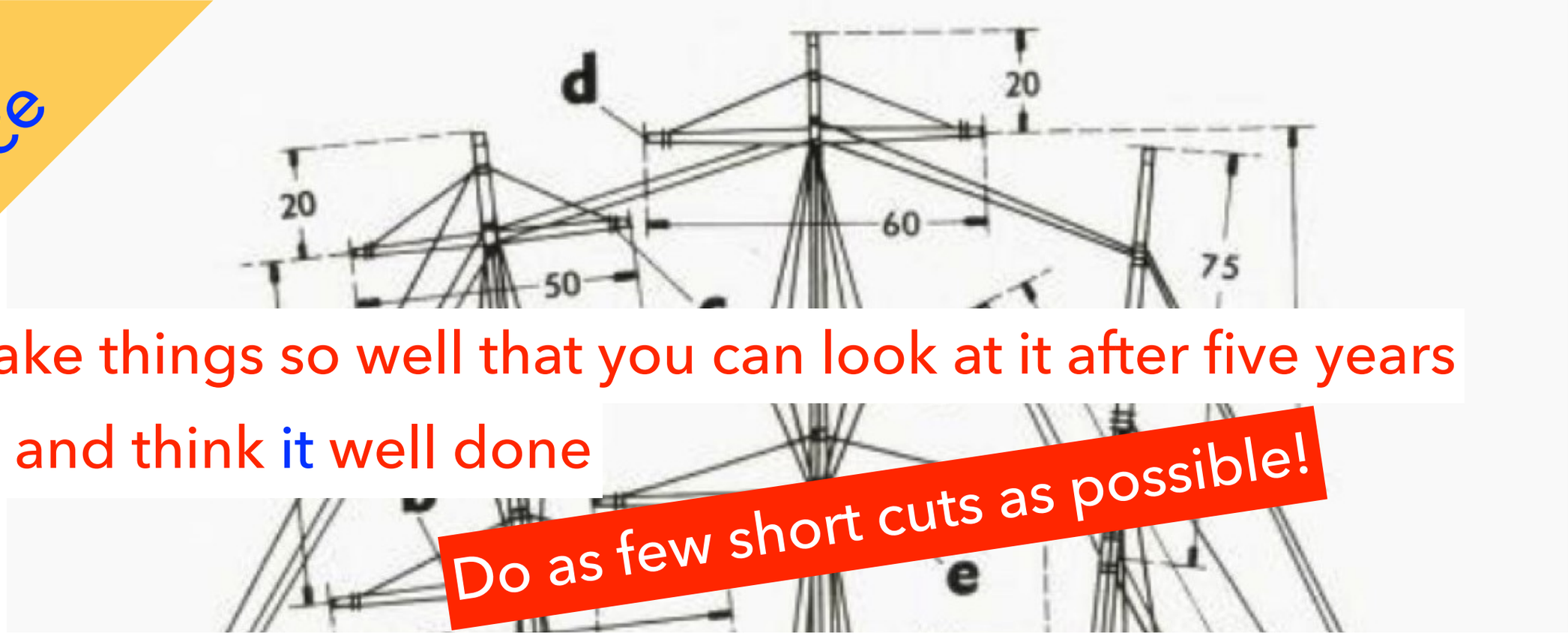


Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice



Make things so well that you can look at it after five years and think it well done

Do as few short cuts as possible!

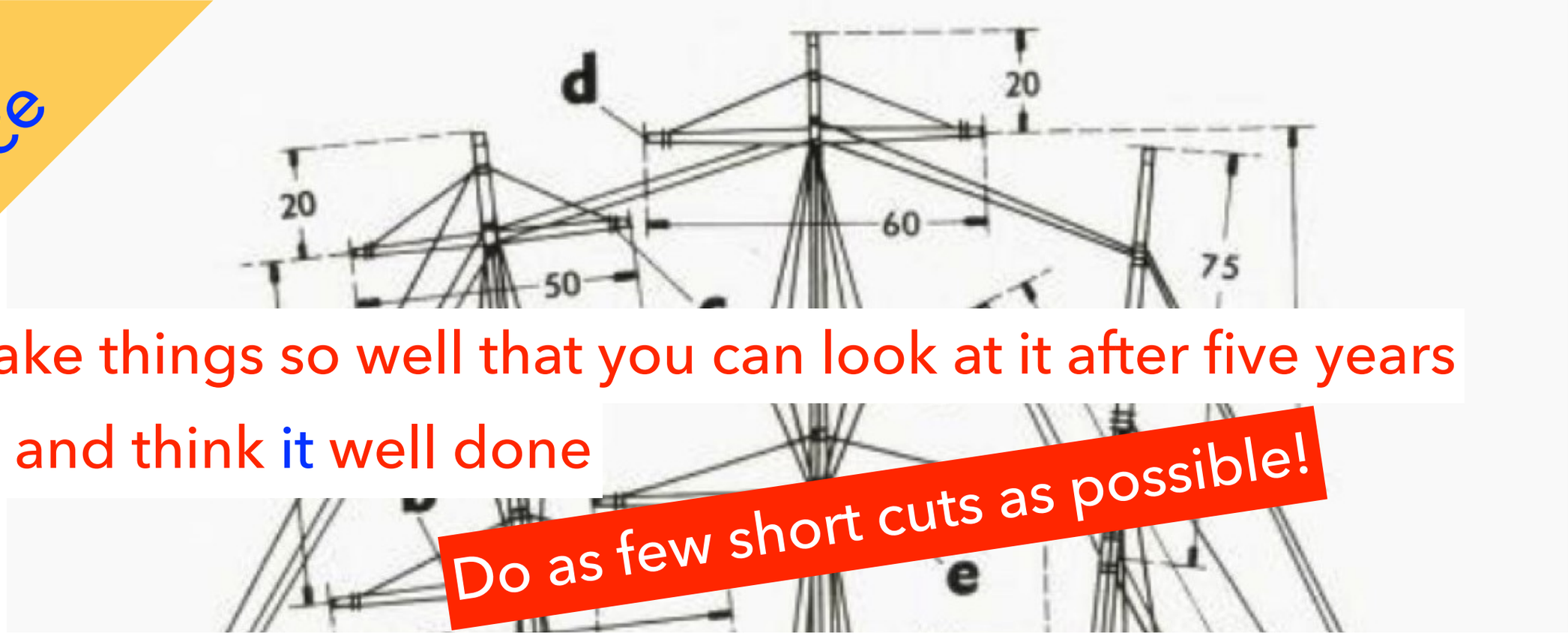
Make sure that you will have moved so much those five years that you wish you could have the time to make it even better now

Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice



Make things so well that you can look at it after five years

and think it well done

Do as few short cuts as possible!

Make sure that you will have moved so much those five years

that you wish you could have the time

to make it even better now

So, if you get into real-time, parallel or concurrent systems

Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice

Make things so well that you can look at it after five years

and think it well done

Do as few short cuts as possible!

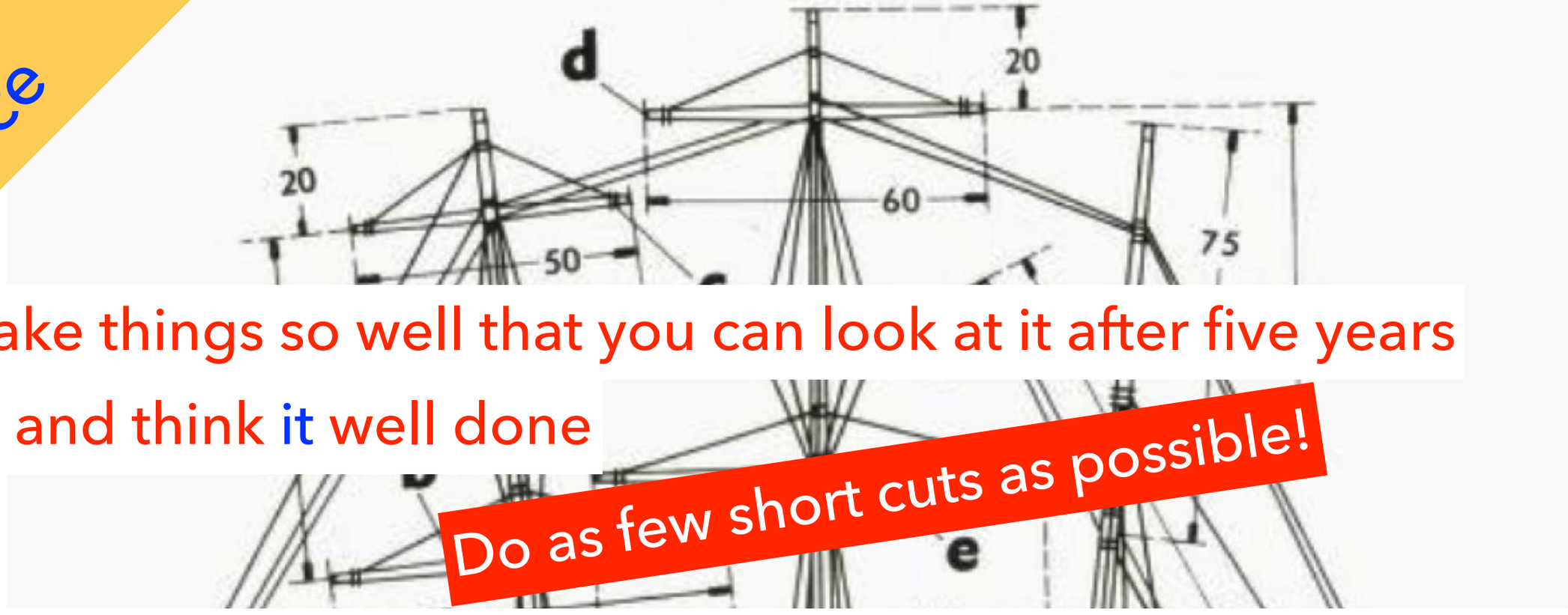
Make sure that you will have moved so much those five years

that you wish you could have the time

to make it even better now

So, if you get into real-time, parallel or concurrent systems

Try to think those five years, ahead

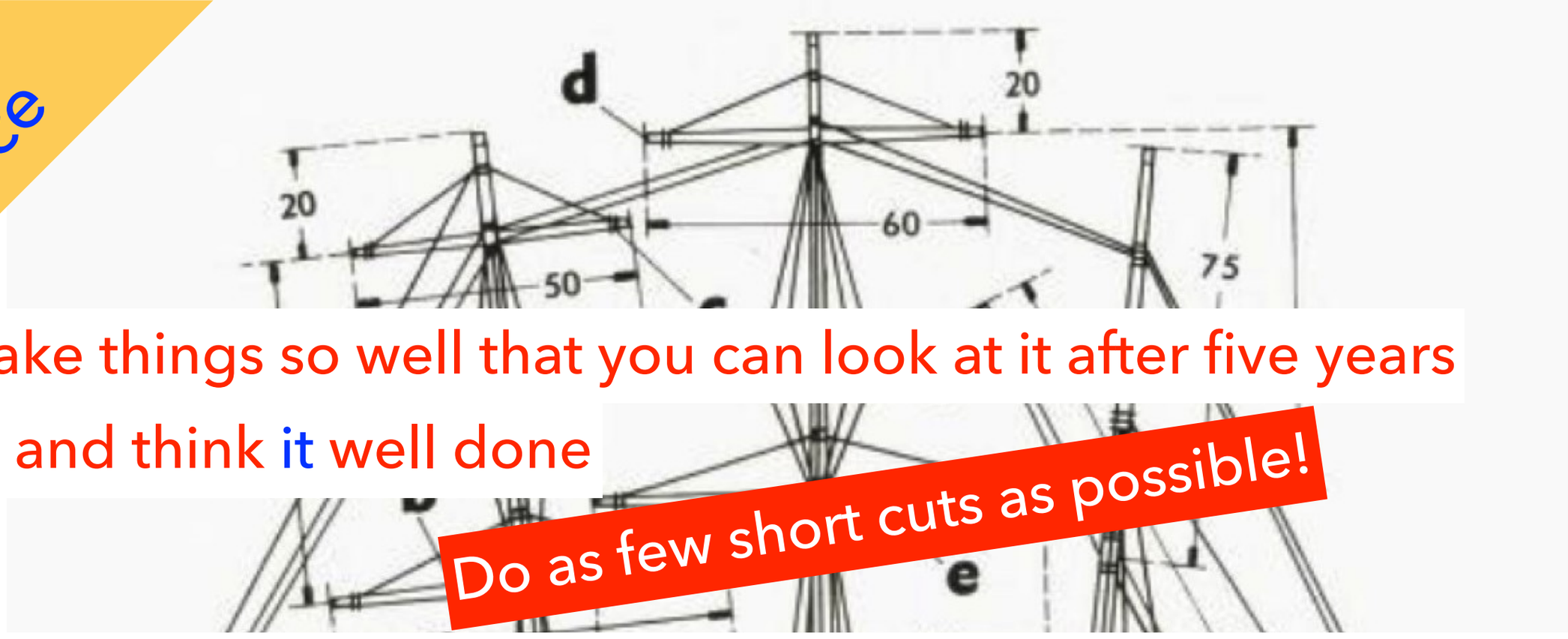


Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

Final advice



Make things so well that you can look at it after five years

and think it well done

Do as few short cuts as possible!

Make sure that you will have moved so much those five years

that you wish you could have the time

to make it even better now

So, if you get into real-time, parallel or concurrent systems

Try to think those five years, ahead Now

10

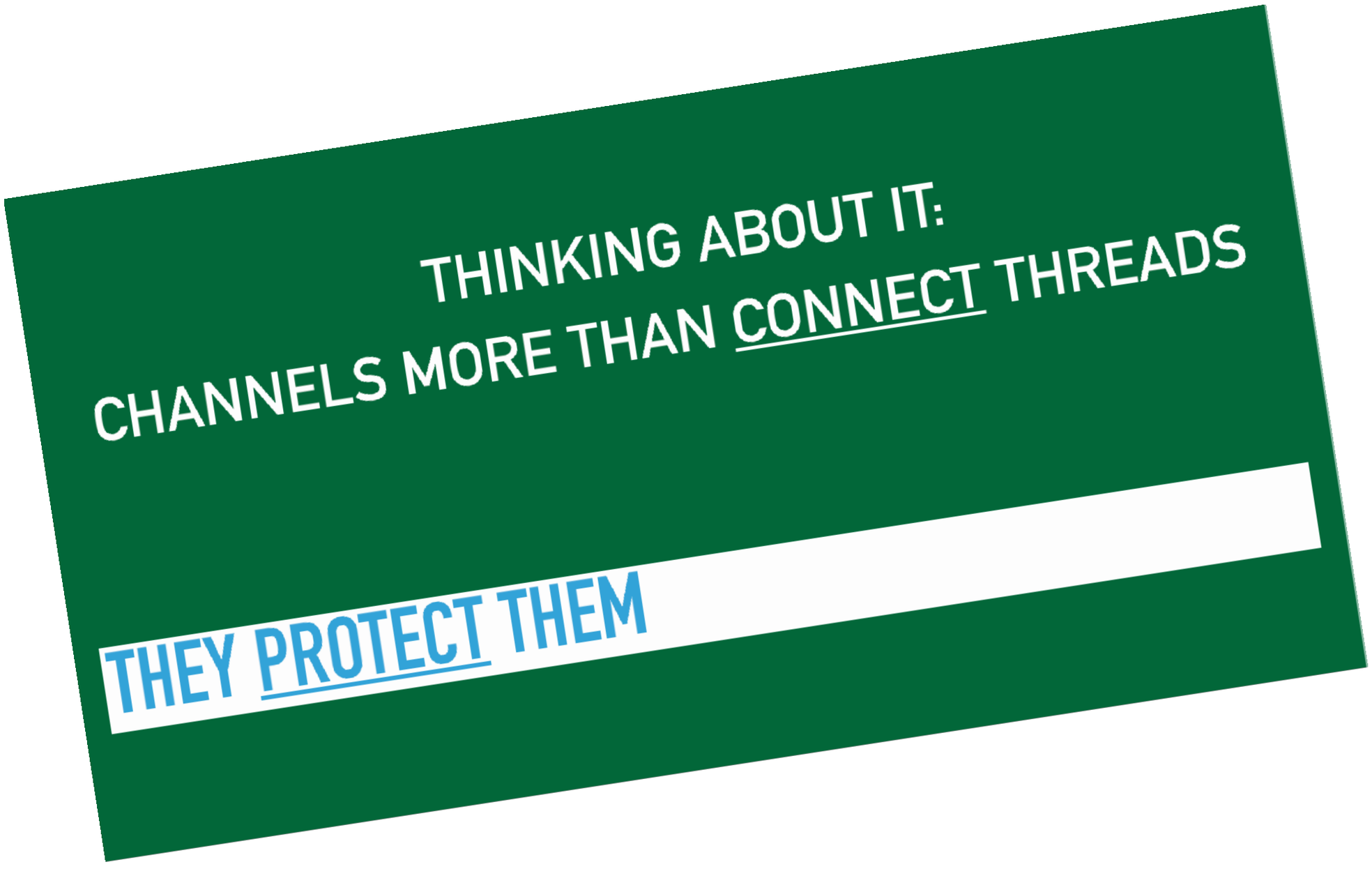
Master, spryd og rær

Master, rær, baug- og akterspryd må lages tynne

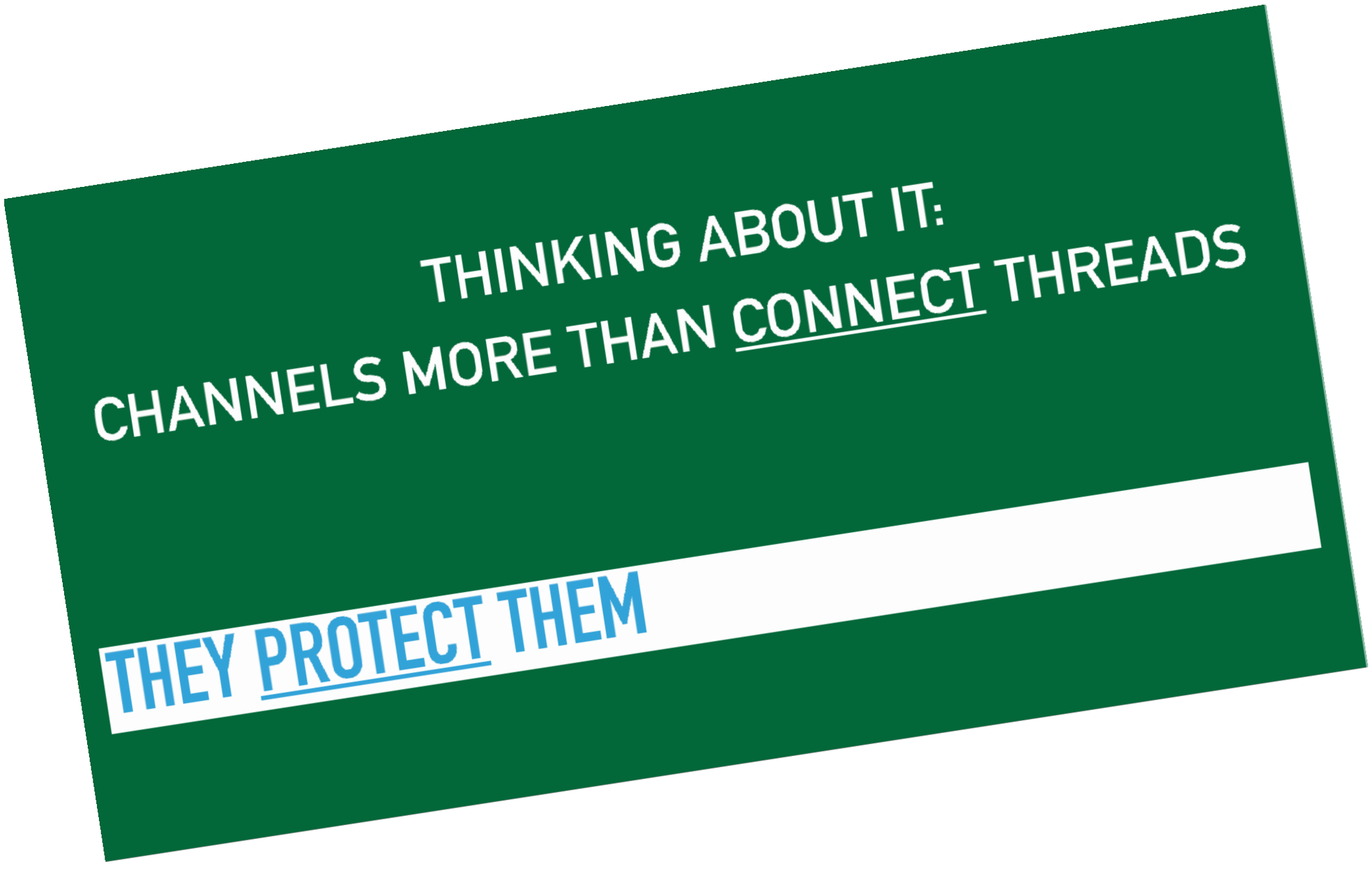
De to runde *mersene* med spor til vantene sages ut av 2 mm kryssfinér efter mønstrene *h* og *i* på side 39. De træs ned på stormast og formast,

THINKING ABOUT IT:
CHANNELS MORE THAN CONNECT THREADS

THEY PROTECT THEM



HOW DO THEY PROTECT THEM?



HOW DO THEY PROTECT THEM?
SUMMARY:

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

- They (and the «process model») help with *reasoning* about the SW architecture

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

- They (and the «process model») help with *reasoning* about the SW architecture
  - At «link layer» (channels)

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

- They (and the «process model») help with *reasoning* about the SW architecture
  - At «link layer» (channels)
  - At «session layer» (interface with client, server etc.)

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

- They (and the «process model») help with *reasoning* about the SW architecture
  - At «link layer» (channels)
  - At «session layer» (interface with client, server etc.)
  - At application layer (talking with another thread's application layer)

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

- They (and the «process model») help with *reasoning* about the SW architecture
  - At «link layer» (channels)
  - At «session layer» (interface with client, server etc.)
  - At application layer (talking with another thread's application layer)
- Keeping local state as consistent as possible!

CHANNELS «PROTECT» THREADS / PROCESSES / TASKS

- They (and the «process model») help with *reasoning* about the SW architecture
  - At «link layer» (channels)
  - At «session layer» (interface with client, server etc.)
  - At application layer (talking with another thread's application layer)
- Keeping local state as consistent as possible!
  - Avoiding, to receive (and send) messages that must be handled «later»

CONTACT INFO ETC.

oyvind.teig@teigfam.net

CONTACT INFO ETC.

oyvind.teig@teigfam.net

- This lecture
  - Standard picture quality, all build steps
    http://www.teigfam.net/oyvind/pub/NTNU_2018/foredrag.pdf
  - Full quality, but each page only once, no build steps (around 70 MB)
    http://www.teigfam.net/oyvind/pub/NTNU_2018/foredrag_full.pdf
- This course
  NTNU, TTK4145 Sanntidsprogrammering (Real-Time Programming) http://www.itk.ntnu.no/fag/TTK4145/information/
- My blog notes
  http://www.teigfam.net/oyvind/home/technology/

RELATED READING, SOME ALREADY REFERENCED..

RELATED READING, SOME ALREADY REFERENCED..

- **Bell Labs and CSP Threads**
  by Russ Cox at https://swtch.com/~rsc/thread/, referred at one of my blog notes: http://www.teigfam.net/oyvind/home/technology/072-pike-sutter-concurrency-vs-concurrency/

RELATED READING, SOME ALREADY REFERENCED..

- **Bell Labs and CSP Threads**
  by Russ Cox at https://swtch.com/~rsc/thread/, referred at one of my blog notes: http://www.teigfam.net/oyvind/home/technology/072-pike-sutter-concurrency-vs-concurrency/

- **Clojure core.async**
  Lecture (45 mins). Rich Hickey explains callback and event loops vs. processes, select and channels at http://www.infoq.com/presentations/clojure-core-async

RELATED READING, SOME ALREADY REFERENCED..

- **Bell Labs and CSP Threads**
  by Russ Cox at https://swtch.com/~rsc/thread/, referred at one of my blog notes: http://www.teigfam.net/oyvind/home/technology/072-pike-sutter-concurrency-vs-concurrency/

- **Clojure core.async**
  Lecture (45 mins). Rich Hickey explains callback and event loops vs. processes, select and channels at http://www.infoq.com/presentations/clojure-core-async

- **New ALT for Application Timers and Synchronisation Point Scheduling**
  CPA-2009. Per Johan Vannebo, Øyvind Teig. Read at http://www.teigfam.net/oyvind/pub/pub_details.html#NewALT. About ChanSched

RELATED READING, SOME ALREADY REFERENCED..

- **Bell Labs and CSP Threads**
  by Russ Cox at https://swtch.com/~rsc/thread/, referred at one of my blog notes: http://www.teigfam.net/oyvind/home/technology/072-pike-sutter-concurrency-vs-concurrency/

- **Clojure core.async**
  Lecture (45 mins). Rich Hickey explains callback and event loops vs. processes, select and channels at http://www.infoq.com/presentations/clojure-core-async

- **New ALT for Application Timers and Synchronisation Point Scheduling**
  CPA-2009. Per Johan Vannebo, Øyvind Teig. Read at http://www.teigfam.net/oyvind/pub/pub_details.html#NewALT. About ChanSched

- Last, but not least:

RELATED READING, SOME ALREADY REFERENCED..

- **Bell Labs and CSP Threads**
  by Russ Cox at https://swtch.com/~rsc/thread/, referred at one of my blog notes: http://www.teigfam.net/oyvind/home/technology/072-pike-sutter-concurrency-vs-concurrency/

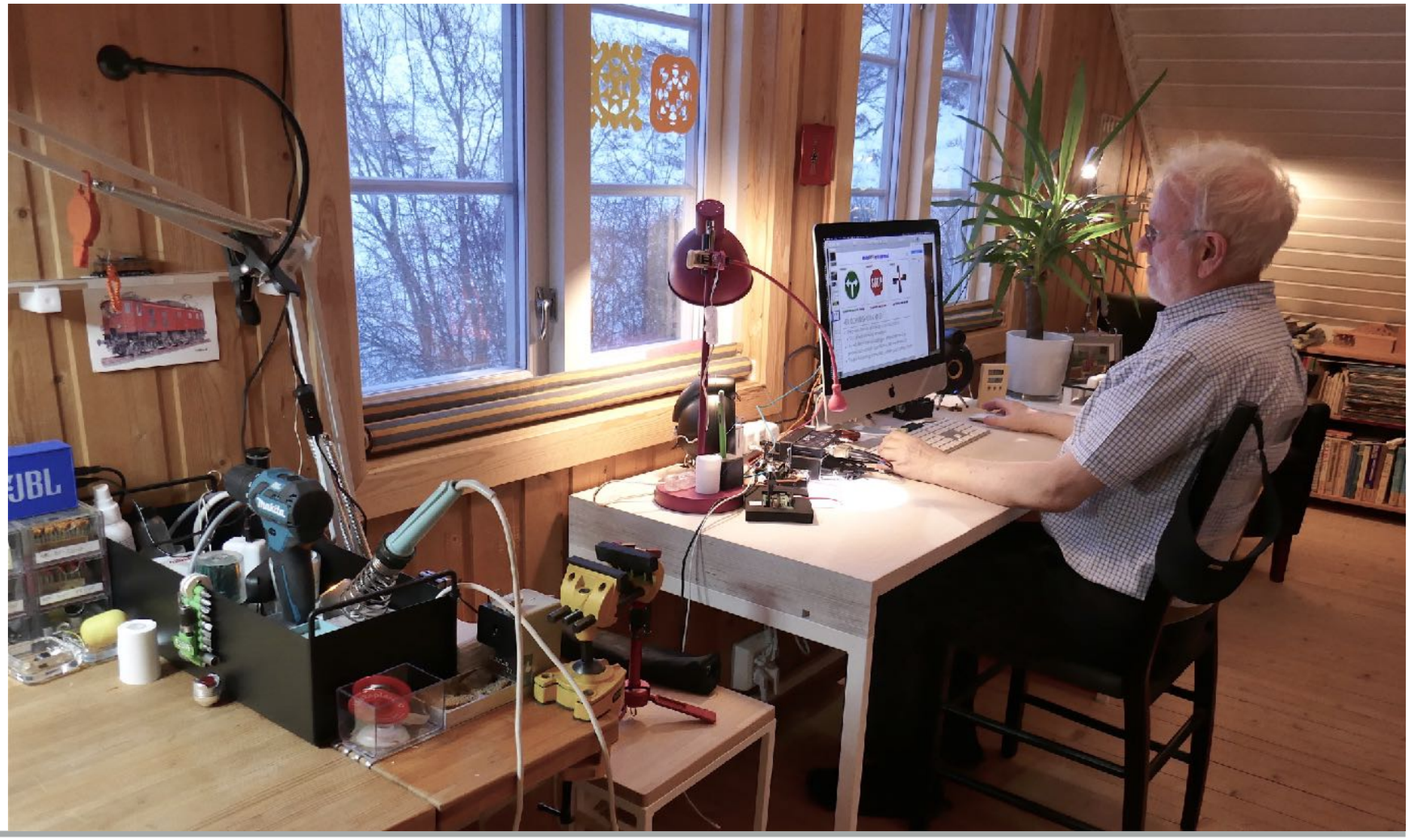
- **Clojure core.async**
  Lecture (45 mins). Rich Hickey explains callback and event loops vs. processes, select and channels at http://www.infoq.com/presentations/clojure-core-async

- **New ALT for Application Timers and Synchronisation Point Scheduling**
  CPA-2009. Per Johan Vannebo, Øyvind Teig. Read at http://www.teigfam.net/oyvind/pub/pub_details.html#NewALT. About ChanSched

- Last, but not least:

  - **ProXC++ - A CSP-inspired Concurrency Library for Modern C++ with Dynamic Multithreading for Multi-Core Architectures** by, Edvard Severin Pettersen. Master thesis, NTNU (2017). Read at https://brage.bibsys.no/xmlui/handle/11250/2453094

Questions?



Questions?

Thank you!