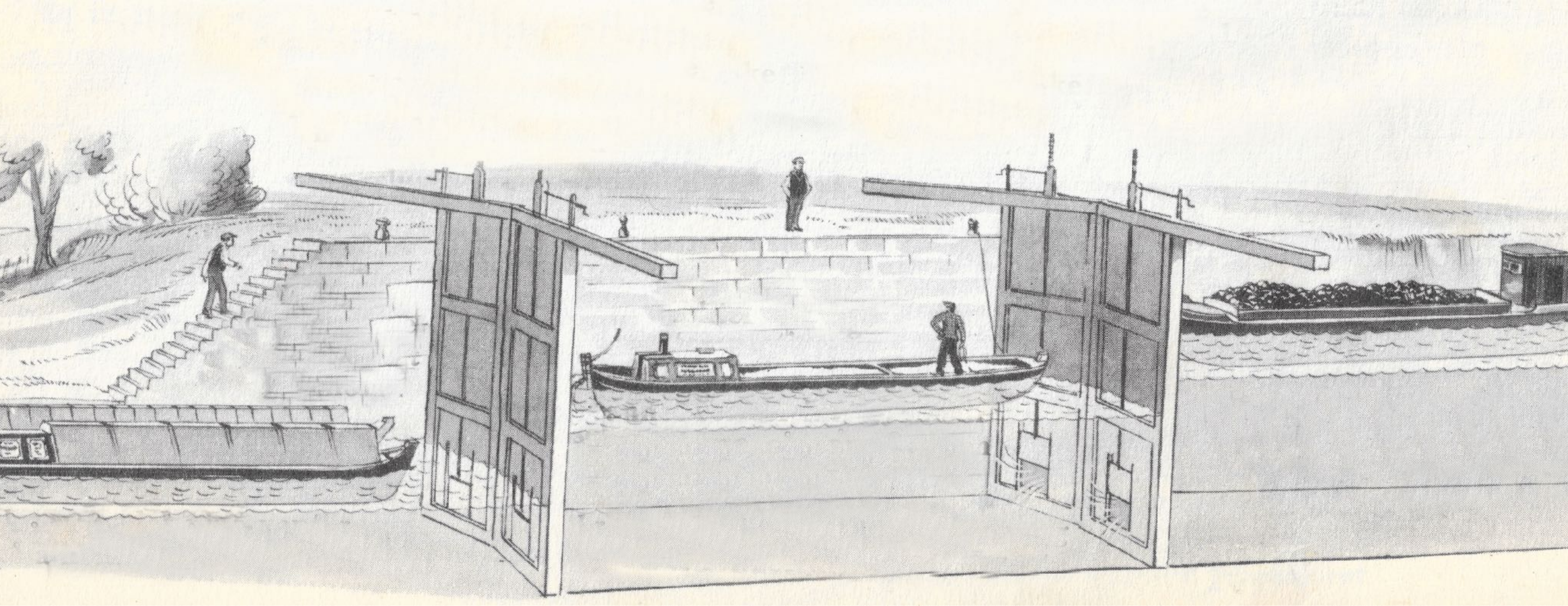
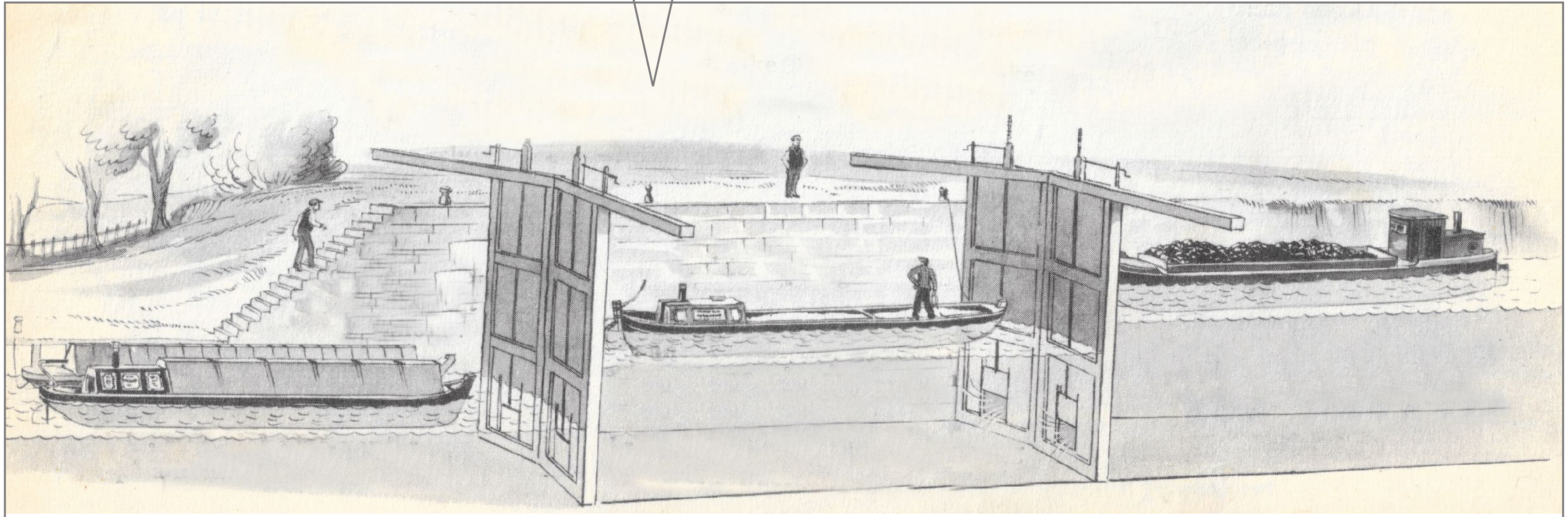


CSP: arriving at the `chan`nel island
From hard microseconds to speedy years. Real time in the industry
Thinking about it: Channels more than connect threads. They protect them

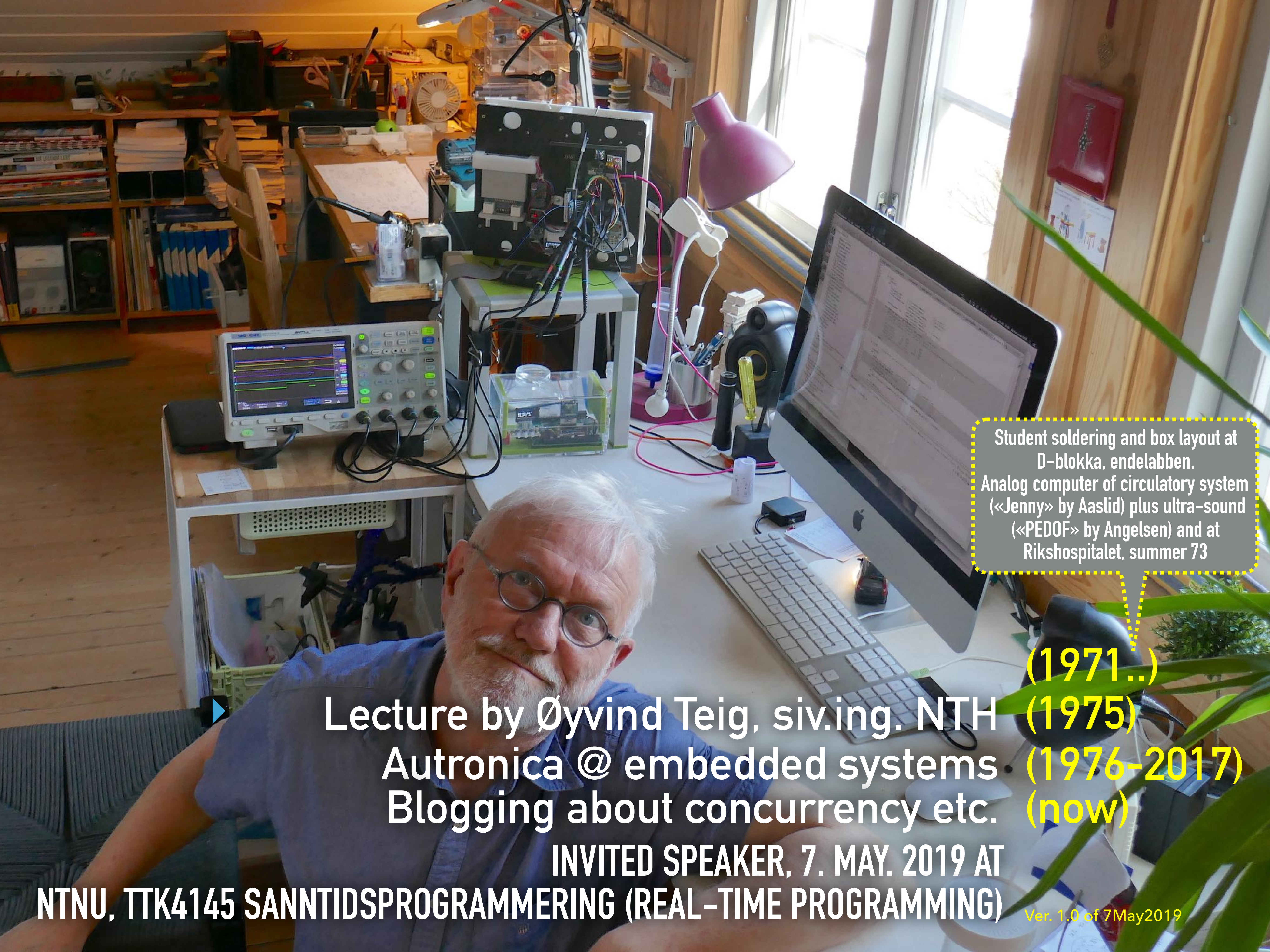
CHANNELING AGAINST THE FLOW

The full, annotated figure from
«Verden omkring oss», 1955 ("Odhams Encyclopedia for Children»)
shown later in this lecture



CSP: arriving at the `chan`nel island
From hard microseconds to speedy years. Real time in the industry
Thinking about it: Channels more than connect threads. They protect them

CHANNELING AGAINST THE FLOW



Student soldering and box layout at D-blokka, endelabben.
Analog computer of circulatory system («Jenny» by Aaslid) plus ultra-sound («PEDOF» by Angelsen) and at Rikshospitalet, summer 73

(1971..)

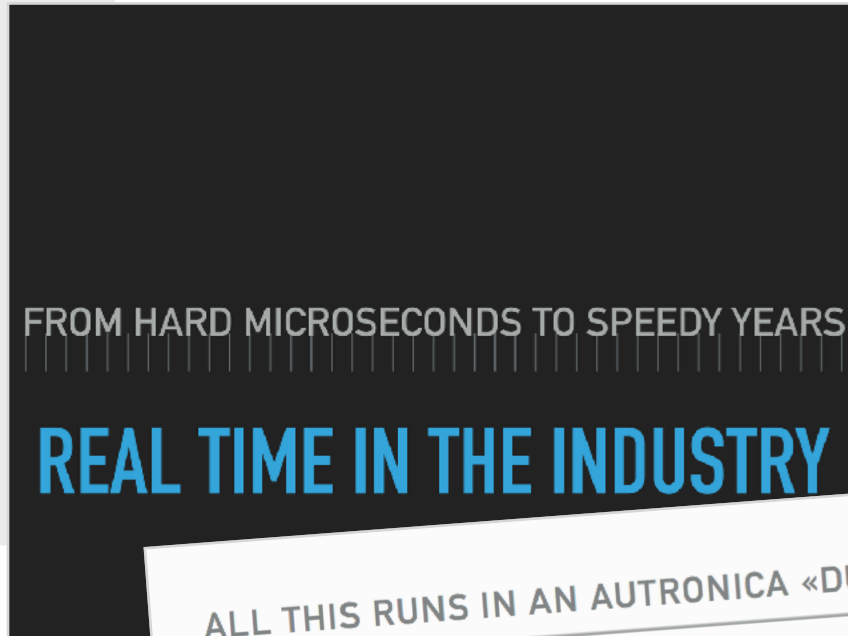
Lecture by Øyvind Teig, siv.ing. NTH (1975)
Autronica @ embedded systems (1976-2017)
Blogging about concurrency etc. (now)

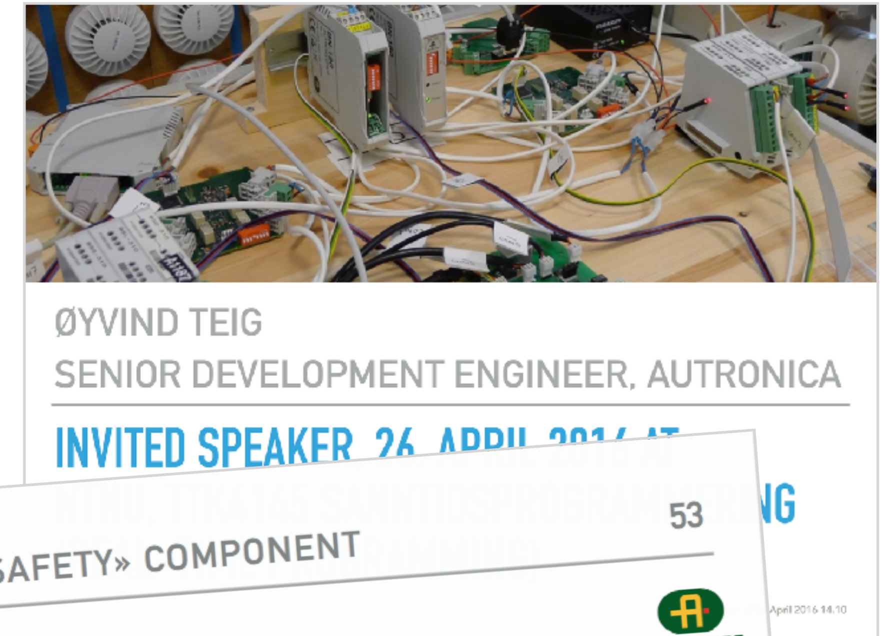
INVITED SPEAKER, 7. MAY. 2019 AT
NTNU, TTK4145 SANNTIDSPROGRAMMERING (REAL-TIME PROGRAMMING)

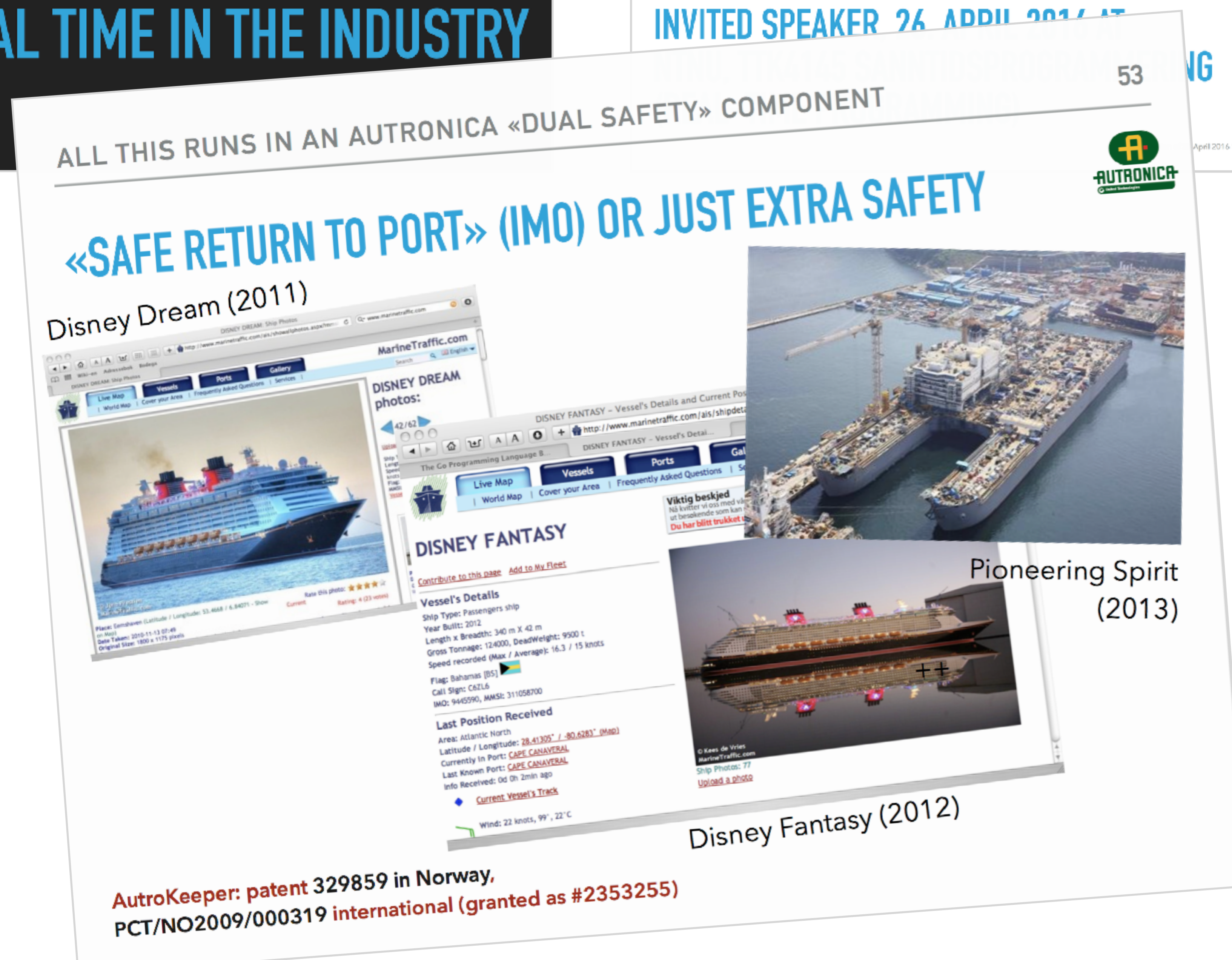
Ver. 1.0 of 7May2019

AS SAID, PREVIOUS LECTURES WERE DIFFERENT FROM THIS

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







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 from more than 40 years in the industry
- (btw: This lecture is on my home page (ref. at the end))
- ..plus some MATTERS #1-4 SINCE LAST YEAR

To be honest: to tell you as much as possible of what I know about concurrency and things

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+)

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

BARE STANDARD CODE NEEDED

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;
}
```

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?

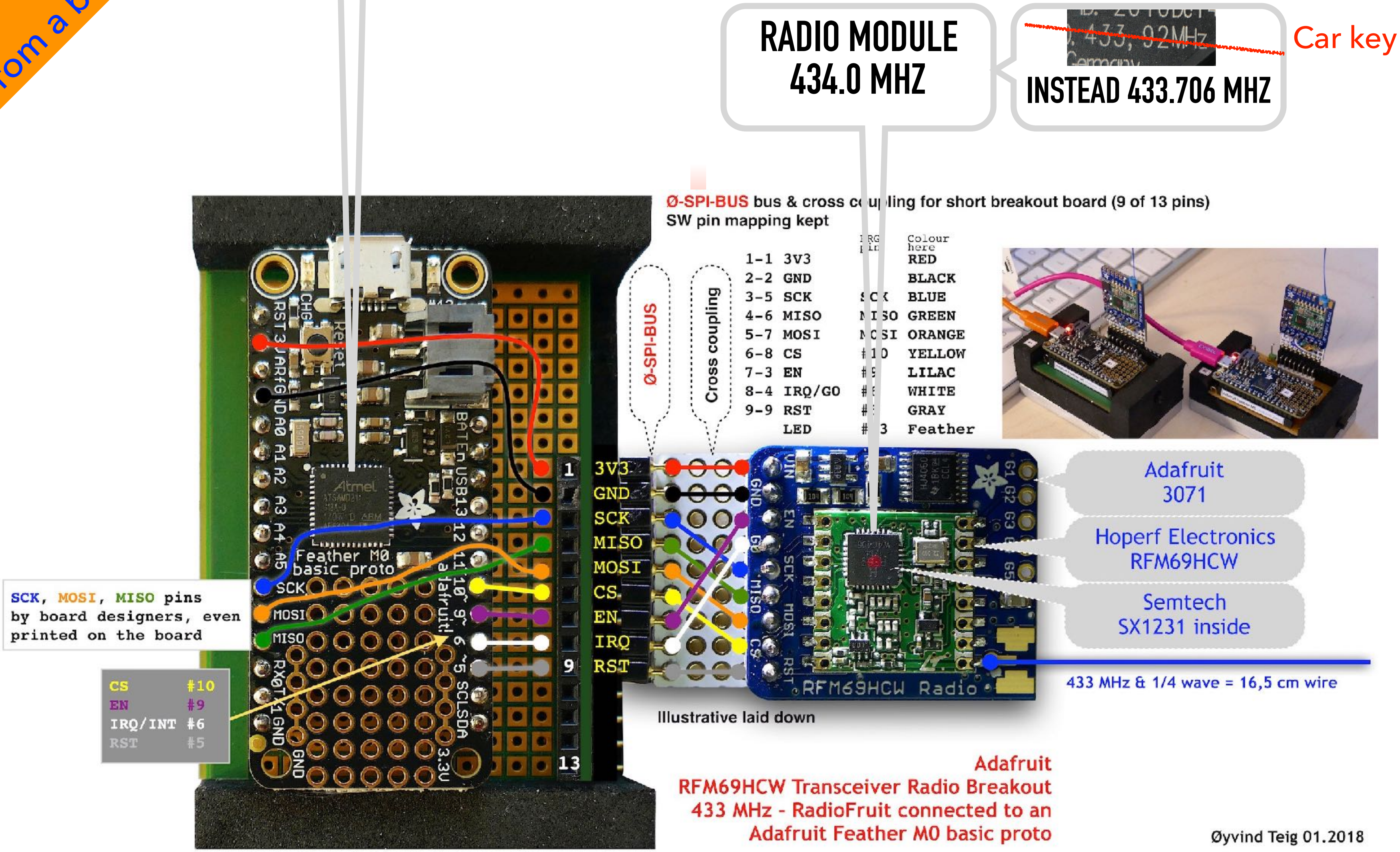
BUT «BLINKING TWO LEDS VIA MOTOR» IS NOT ENOUGH!

- 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 saw a system like this, it took a person a year to fix the mess!
    - The problems were races between interrupts and «main»
- 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

- 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!

«void loop» ON MY DESK

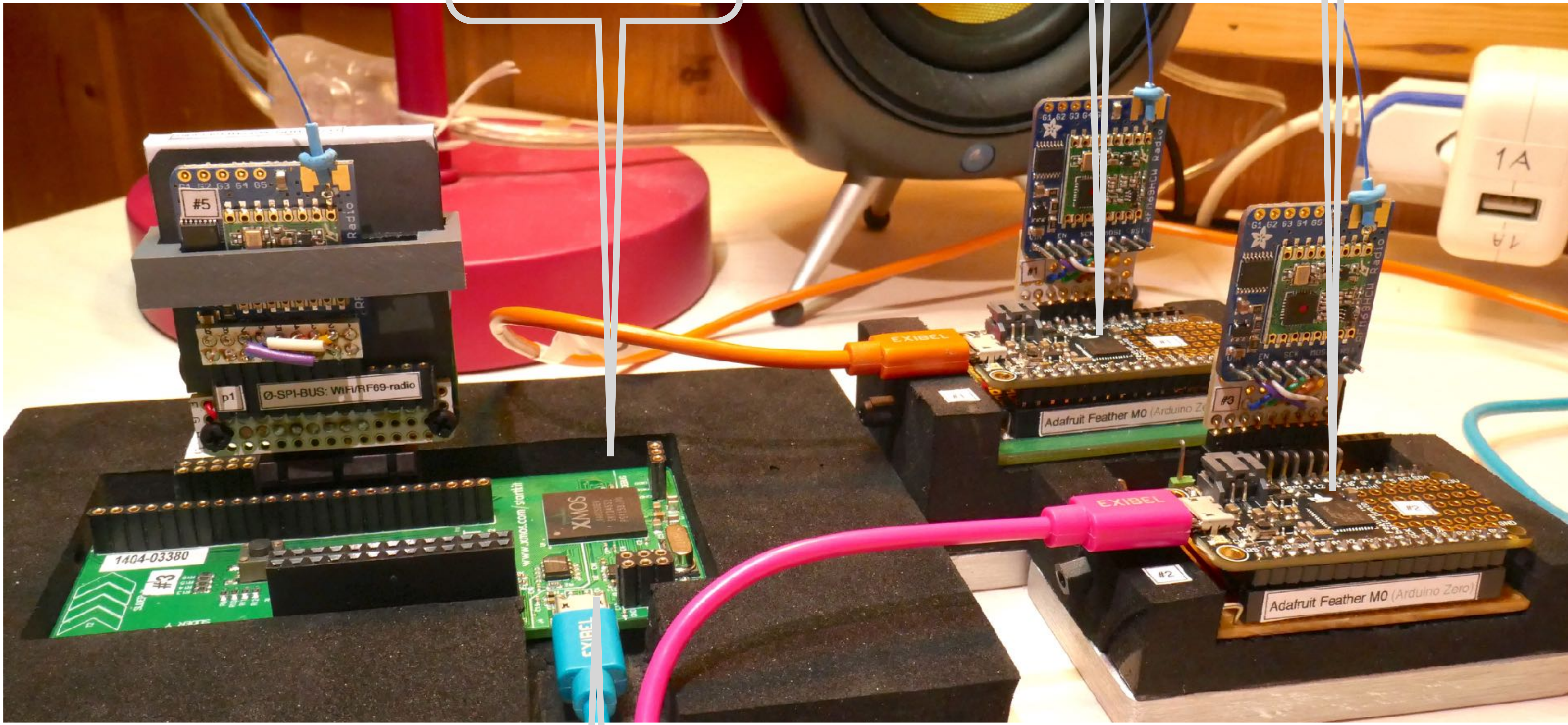


ARDUINO «`void loop`» ON MY DESK - PLUS AN XMOS BOARD

XMOS 8-CORE
XC, C, C++

ARM CORTEX M0

ARM CORTEX M0



Concurrency

No concurrency

MORE LATER

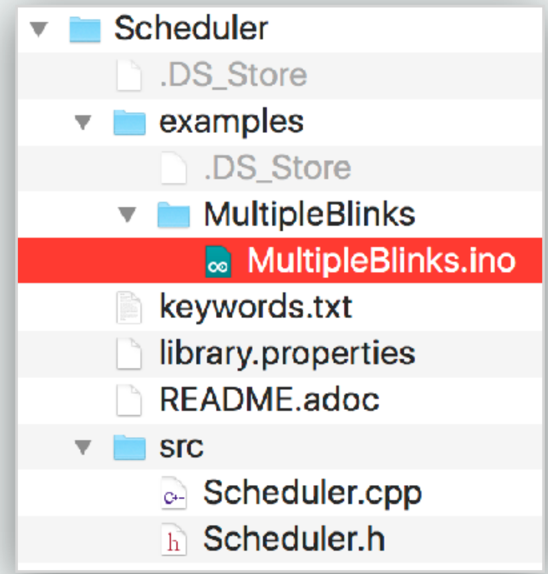
NEXT: `Scheduler`

ARDUINO: Scheduler AND THREE loop()

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

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

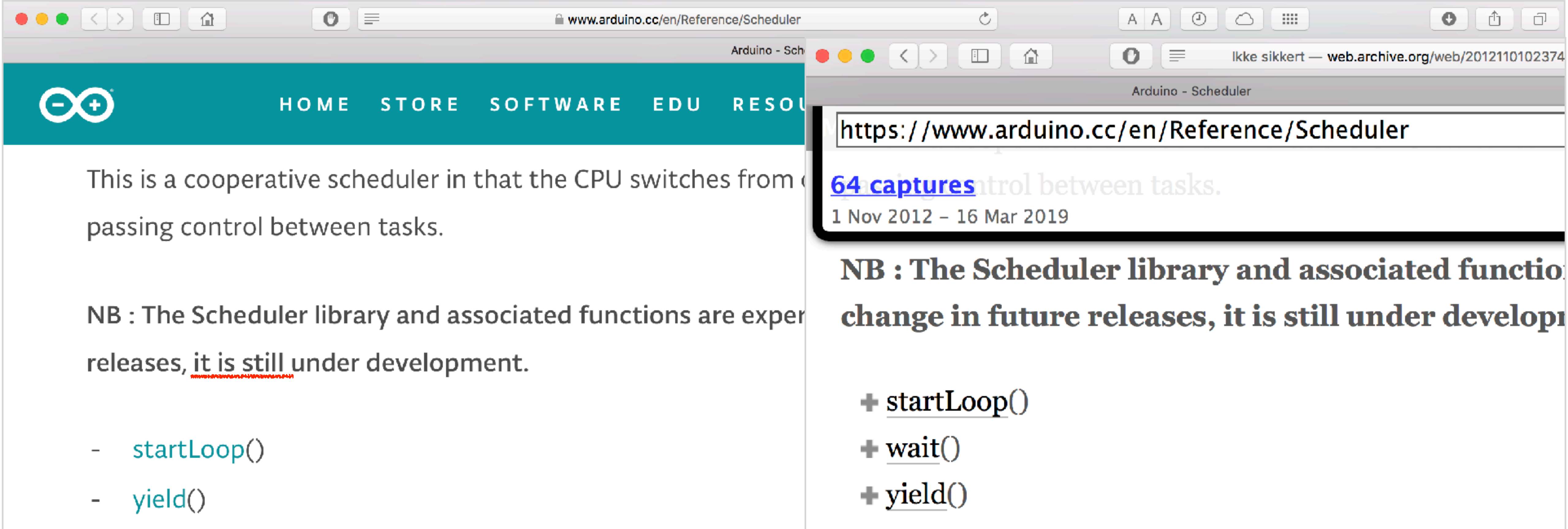
```
// Include Scheduler since we want to manage multiple tasks.
#include <Scheduler.h>

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);
}
```



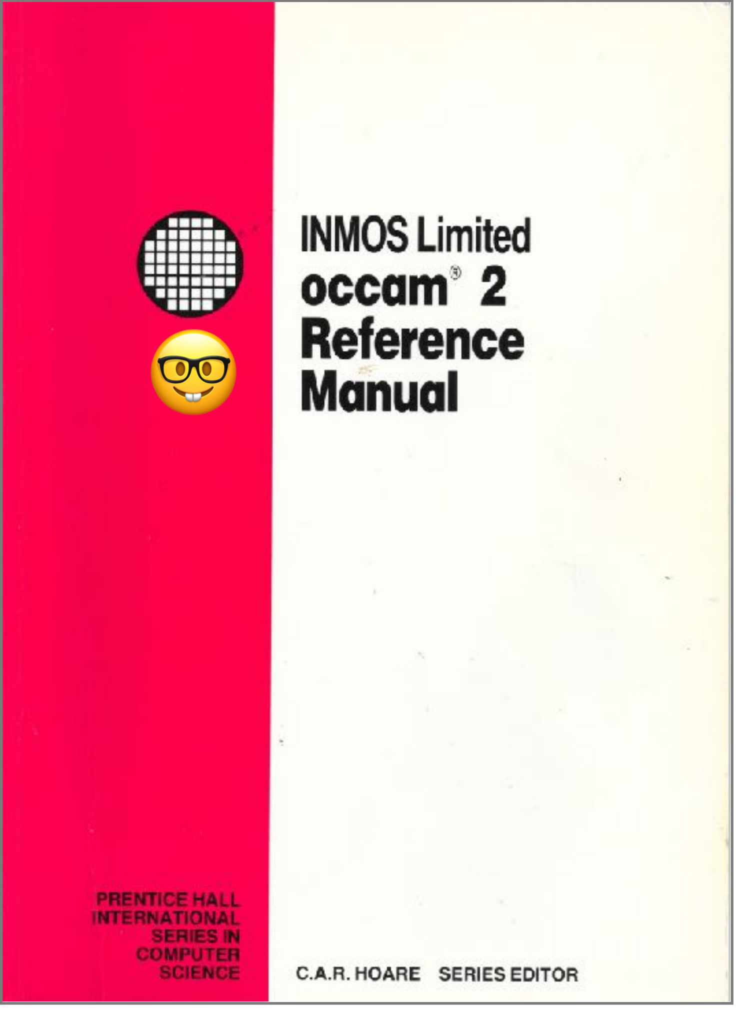

```
// Task no.2: blink LED with 0.1 second delay.
void loop2() {
  digitalWrite(led2, HIGH);
  delay(100);
  digitalWrite(led2, LOW);
  delay(100);
}
```

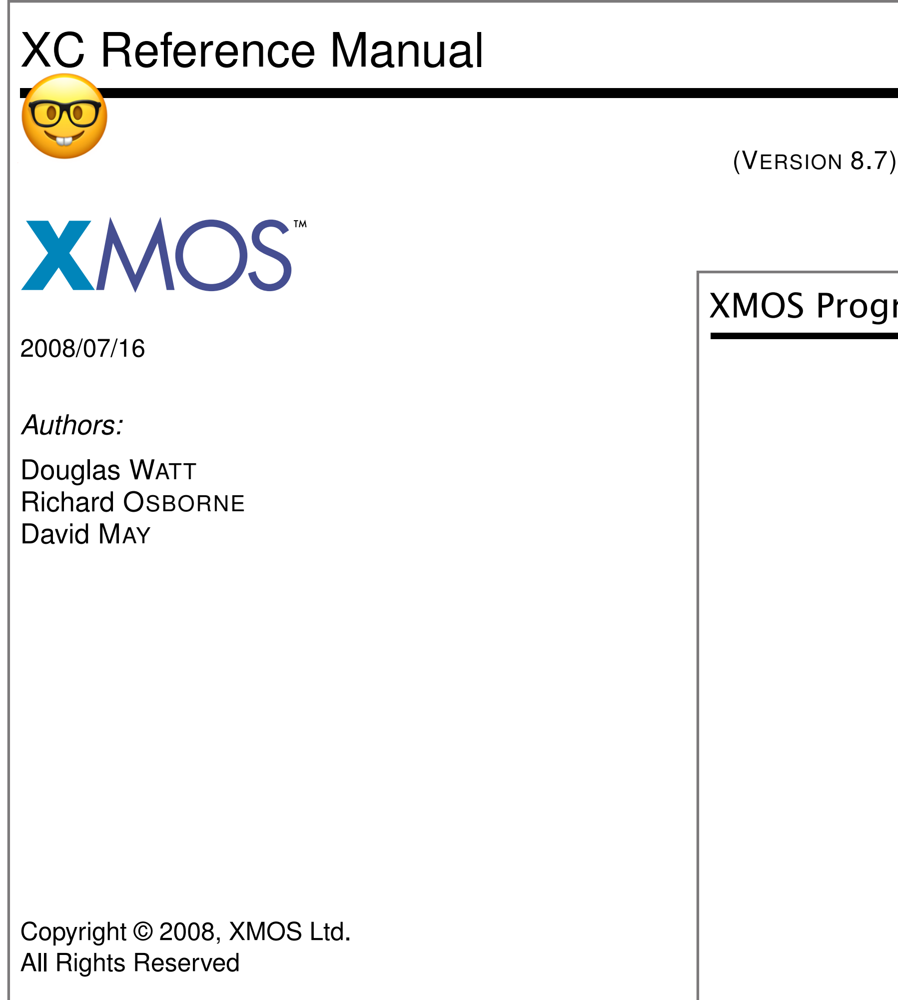
```
// Task no.1: blink LED with 1 second delay.
void loop() {
  digitalWrite(led1, HIGH);

  // 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);
}
```

```
// 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();
}
```



- This is all too usual: concurrency is really not their business: no multi-threading here!
- We often have to use libraries, perhaps even if
  - <threads.h> is an option for C11/C18 compilers
  - and that it supports atomic
- Often home built schedulers
  - Often with a steep learning curve. I would know
- Repeating somehow
  - Solutions would use critical sections (often as disable/enable of interrupt?), semaphores, locks and mutexes only
  - We need time handling, waiting for a set of higher level events with an optional timeout
  - We do not appreciate busy polling
  - etc..




XC Reference Manual

(VERSION 8.7)

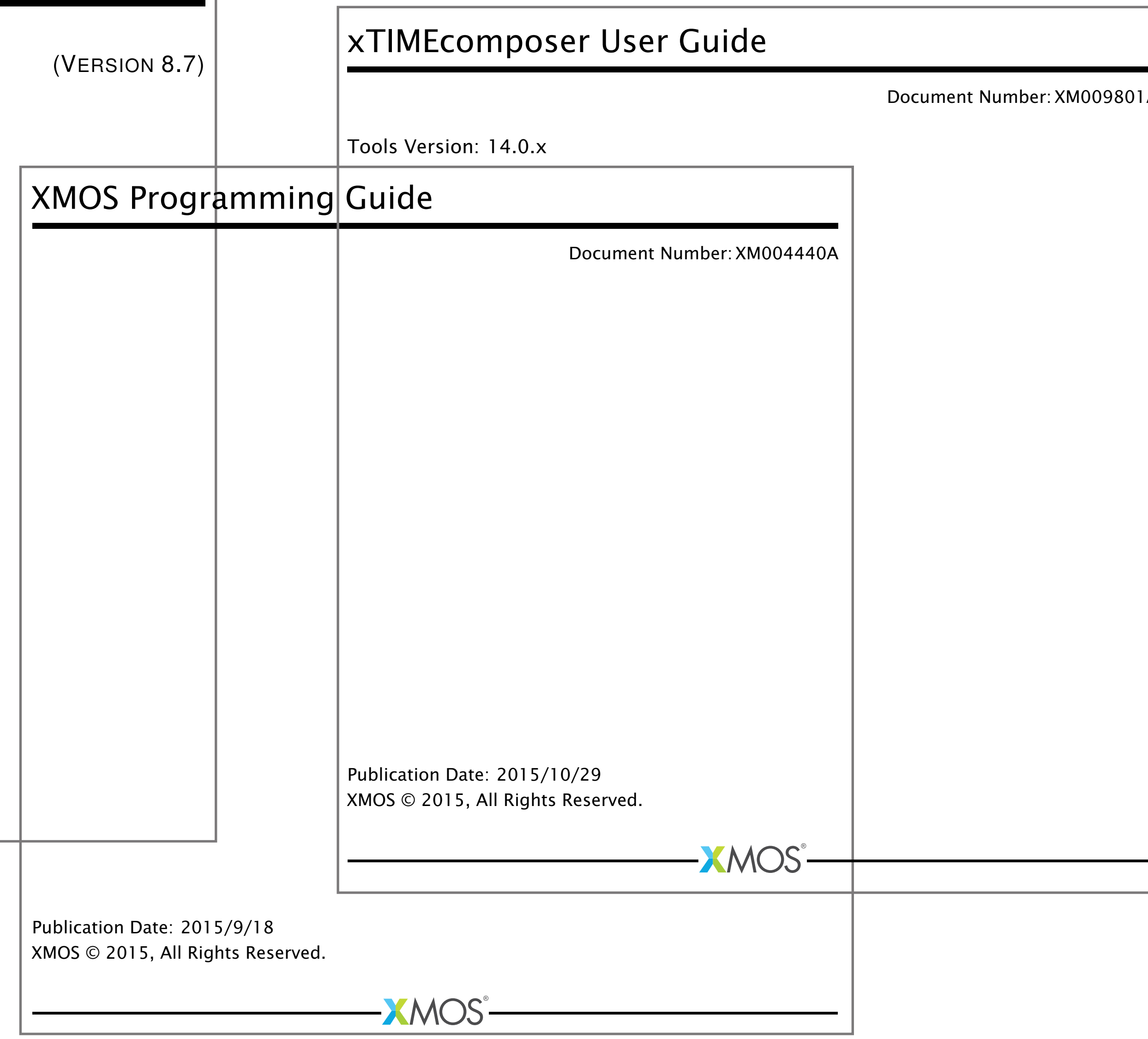
XMOS™

2008/07/16

Authors:

Douglas WATT
Richard OSBORNE
David MAY

Copyright © 2008, XMOS Ltd.
All Rights Reserved


xTIMEcomposer User Guide

Document Number: XM009801A

Tools Version: 14.0.x

Publication Date: 2015/10/29
XMOS © 2015, All Rights Reserved.

XMOS®

XMOS Programming Guide

Document Number: XM004440A

Publication Date: 2015/9/18
XMOS © 2015, All Rights Reserved.

XMOS®

LANGUAGES (THAT I HAVE USED)

- Assembler (1975-1980)
- PL/M (1980-1990)
- Modula-2 (1988-1990)
- MPP-Pascal (1982-1988)
- occam (1990-2001)
- C (2002-2017-(present))
- [Java (1997-2000)]
- [Perl (2002)]
- XC (privately 2012-present)

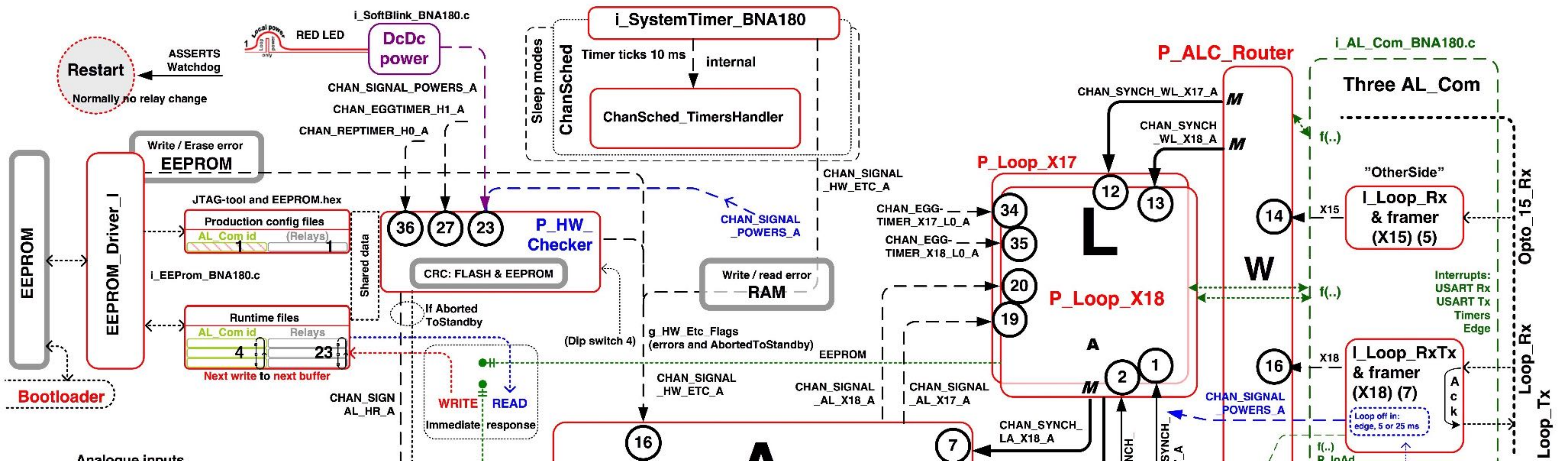
RUNTIMES / SCHEDULERS (THAT I HAVE USED) (1/4)

- Runtime/scheduler is about which process models we have used
  - Rather than all code in a big loop in main
  - More later
- 1978: No runtime system, assembly only
  - *Diesel start/stop for emergency power*
- 1979: MPP Pascal with early «process» term
  - *Protocol conversion, fluid level measurement and fire detection*
- 1980: PL/M with NTH-developed run-time
  - *Ship's machine room monitoring*
- 1982: Assembler with runtime that I developed
  - *Fire detection*

RUNTIMES / SCHEDULERS (THAT I HAVE USED) (2/4)

- 1988: PL/M with runtime that I developed
  - *Fire detection (here?)* 🔭
- 1988: Modula-2 with purchased run-time and coroutines
  - *Fire detection*
- 1990: INMOS transputer with built-in scheduler in HW programmed in occam
  - *Ship's engine monitoring*
- 1995: C with VxWorks os
  - *Fire detection*

RUNTIMES / SCHEDULERS (THAT I HAVE USED) (3/4)



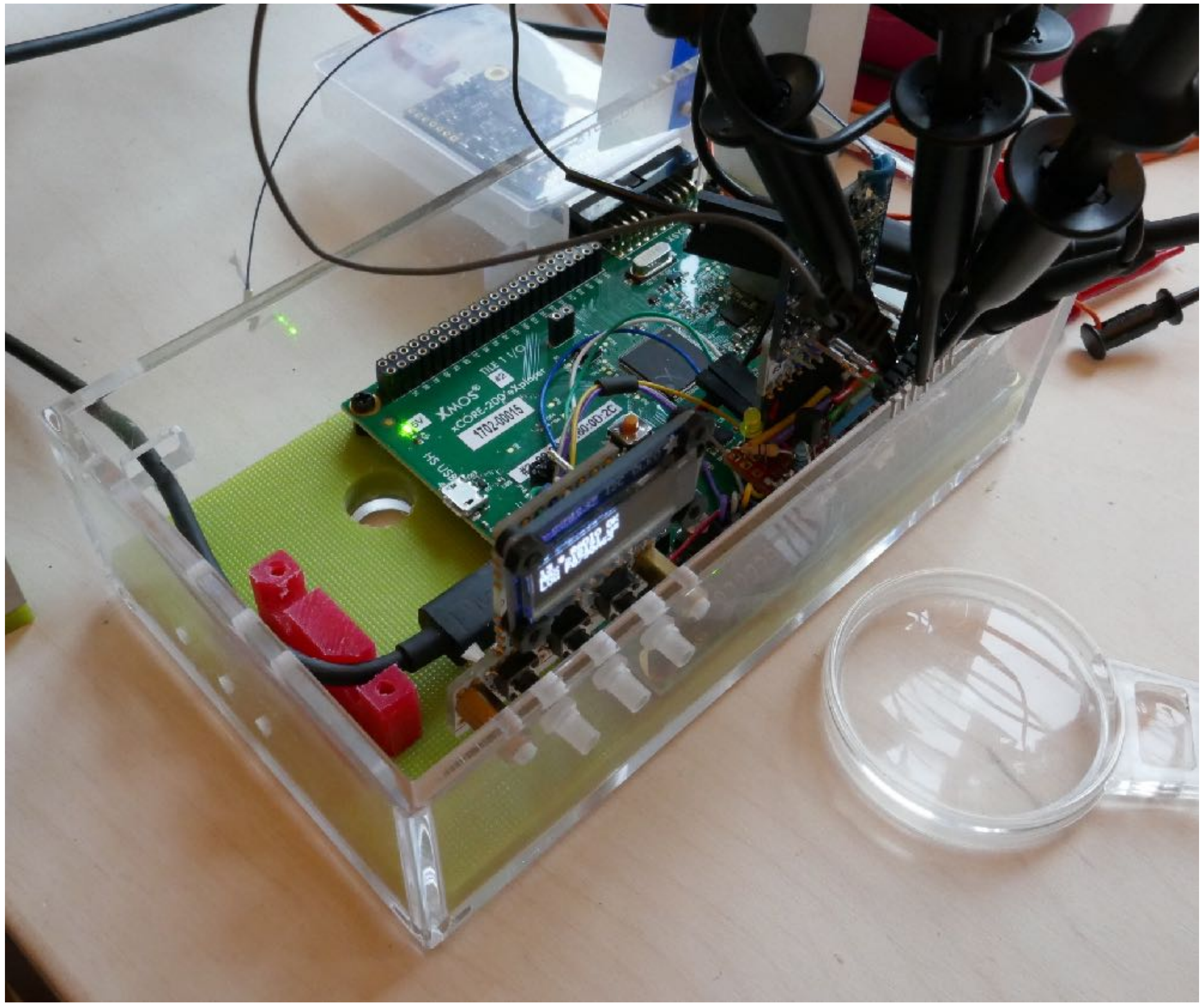
- All below used in fire detection related applications
- State as of June 2017
- 2000: FSM scheduler: Most of our controllers use an
  - asynchronous SDL-based scheduler
- 2006: CHAN_CSP: However: in two of the controller there are
  - synchronous channels on top of the FSM scheduler
- 2010: ChanSched: finally in one of the controllers
  - synchronous channels on top of no other runtime («naked»)

Code examples compared later

RUNTIMES / SCHEDULERS (THAT I HAVE USED) (4/4)

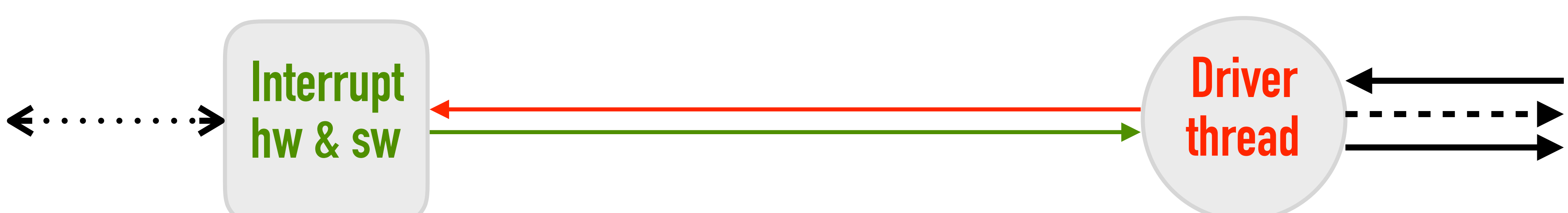
- 2012: XMOS XCore multi-core with built-in scheduler etc. in HW
  - *Blogging and aquarium controller box*
- Programmed in XC
- Also takes C and Cpp (nice for porting)
- Much more later

Aquarium radio client (listener)



WHAT ABOUT (USER LEVEL) 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



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

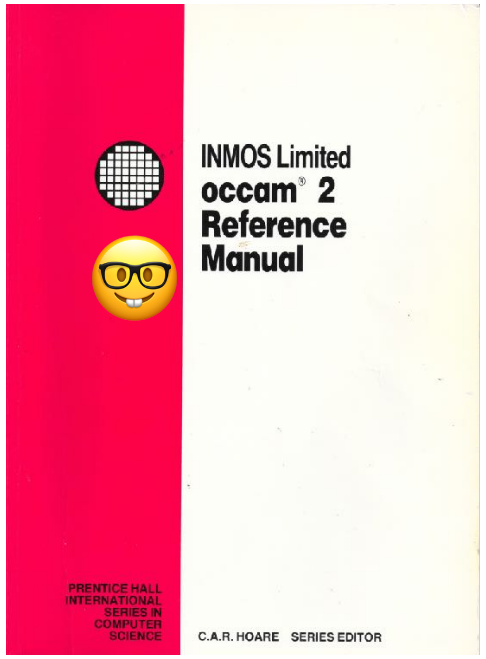
WHAT ABOUT NO (USER LEVEL) INTERRUPTS?

- Two processors I have worked with do not have on board interrupt HW
- With them, dedicated HW may be replaced by dedicated SW
- On the transputer (parallel μP)
  - there was one 'event' line treated as a channel (no data) in occam [1]
- 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 [2]

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 🤓
  - As mentioned, I will show you some here. Has channels and interfaces
  - Also based on CSP

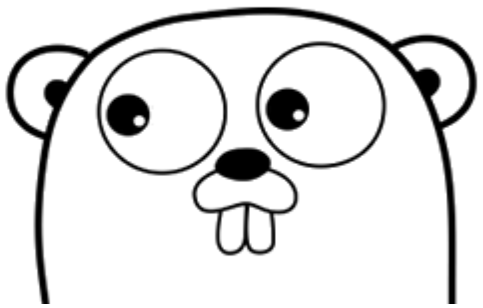


«WHY BUILD CONCURRENCY ON THE IDEAS OF CSP?»

«

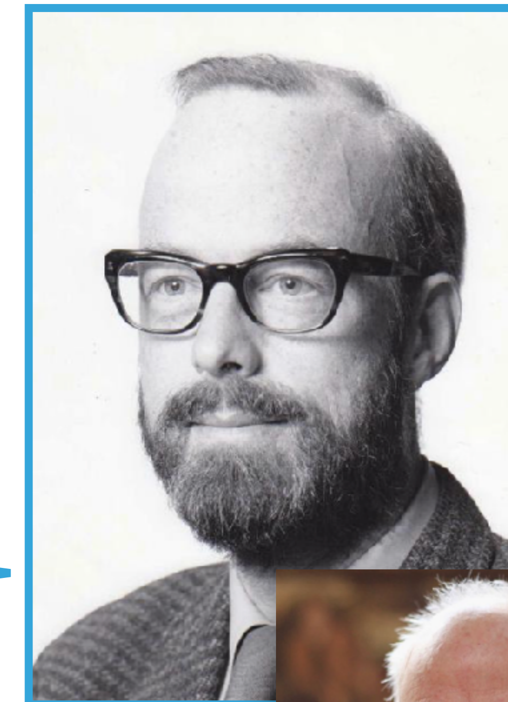
Concurrency and multi-threaded programming have a reputation for difficulty.

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.



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

GO: FREQUENTLY ASKED QUESTIONS (FAQ)

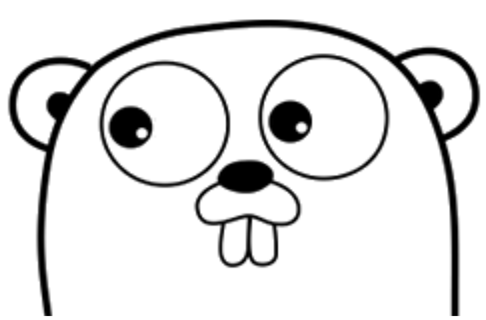


One of the most successful models for providing high-level linguistic support for concurrency comes from Hoare's Communicating Sequential Processes, or CSP.

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?

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» they say
  - Occam on many transputers and one transputer; different properties. Not really relevant any more

Showing a forest for some trees

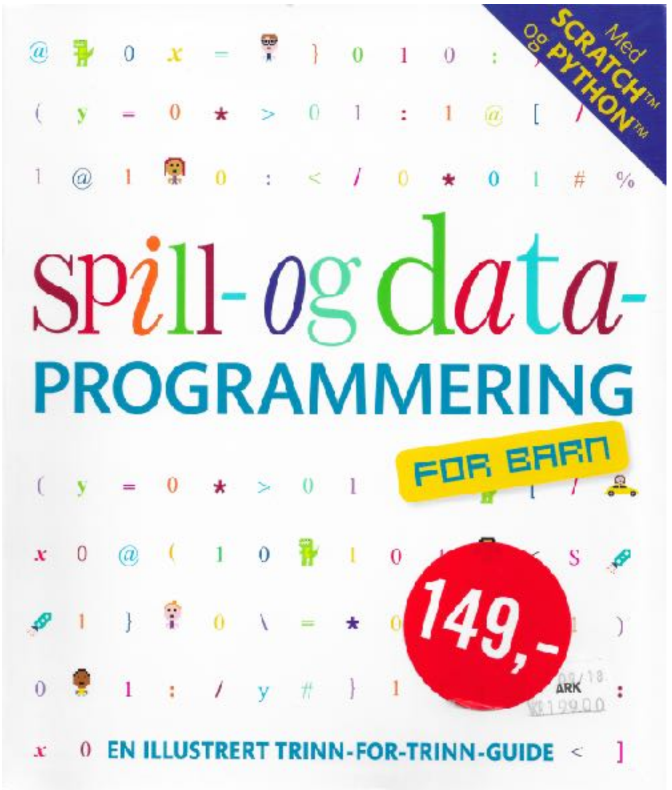
MULTIPLE LOOPS WITH par: XC

```
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;
}
```

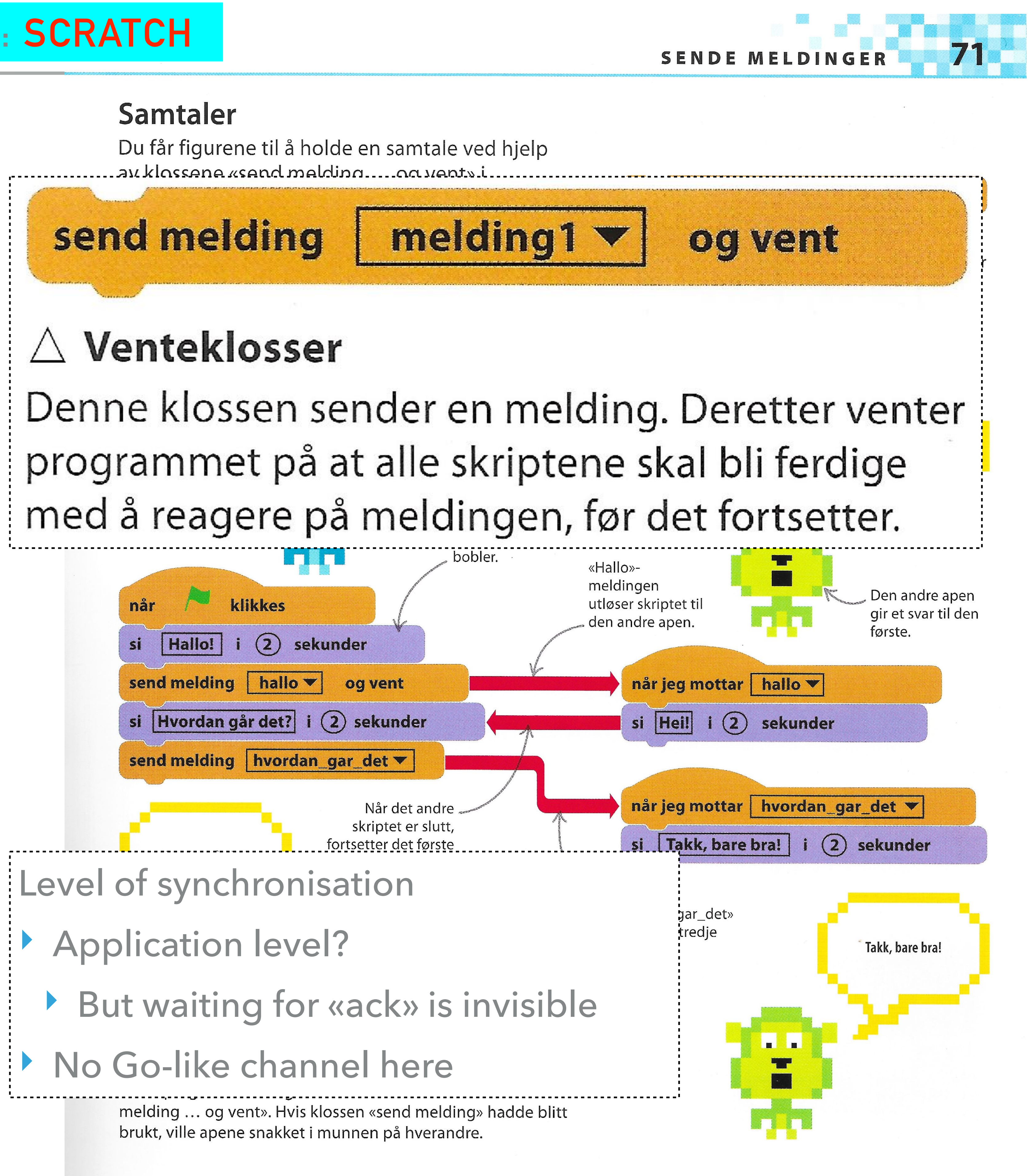
THIS IS PARALLEL

XC from my aquarium controller and xTIMEcomposer

MATTER #1 SINCE LAST YEAR: SCRATCH



Spill-og data-
PROGRAMMERING
Spectrum forlag 2017
ISBN 978-8231611752
→
Computer Coding for Kids
by Carol Vorderman
Dorling Kindersley
ISBN 978-1409347019



Level of synchronisation

- Application level?
  - But waiting for «ack» is invisible
- No Go-like channel here

Using Message Passing to Transfer Data Between Threads
https://doc.rust-lang.org/beta/book/ch16-02-message-passing.html

```
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
    });

    let received = rx.recv().unwrap();
    println!("Got: {}", received);
}
```

MATTER #2 SINCE LAST YEAR: RUST

https://doc.rust-lang.org/std/sync/mpsc/

Structs

IntoIter	An owning iterator over messages on a `Receiver`, created by **Receiver::into_iter**.
Iter	An iterator over messages on a `Receiver`, created by `iter`.
Receiver	The receiving half of Rust's `channel` (or `sync_channel`) type. This half can only be owned by one thread.
RecvError	An error returned from the `recv` function on a `Receiver`.
SendError	An error returned from the `Sender::send` or `SyncSender::send` function on **channel**s.
Sender	The sending-half of Rust's asynchronous `channel` type. This half can only be owned by one thread, but it can be cloned to send to other threads.
SyncSender	The sending-half of Rust's synchronous `sync_channel` type.
TryIter	An iterator that attempts to yield all pending values for a `Receiver`, created by `try_iter`.
Handle	Deprecated Experimental A handle to a receiver which is currently a member of a `Select` set of receivers. This handle is used to keep the receiver in the set as well as interact with the underlying receiver.
Select	Deprecated Experimental The "receiver set" of the select interface. This structure is used to manage a set of receivers which are being selected over.

Disclaimer: I have not coded a line of Scratch or Rust

[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!

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

SELECT (ROB PIKE: «GO CONCURRENCY PATTERNS»)

Rob Pike

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")
    }
```

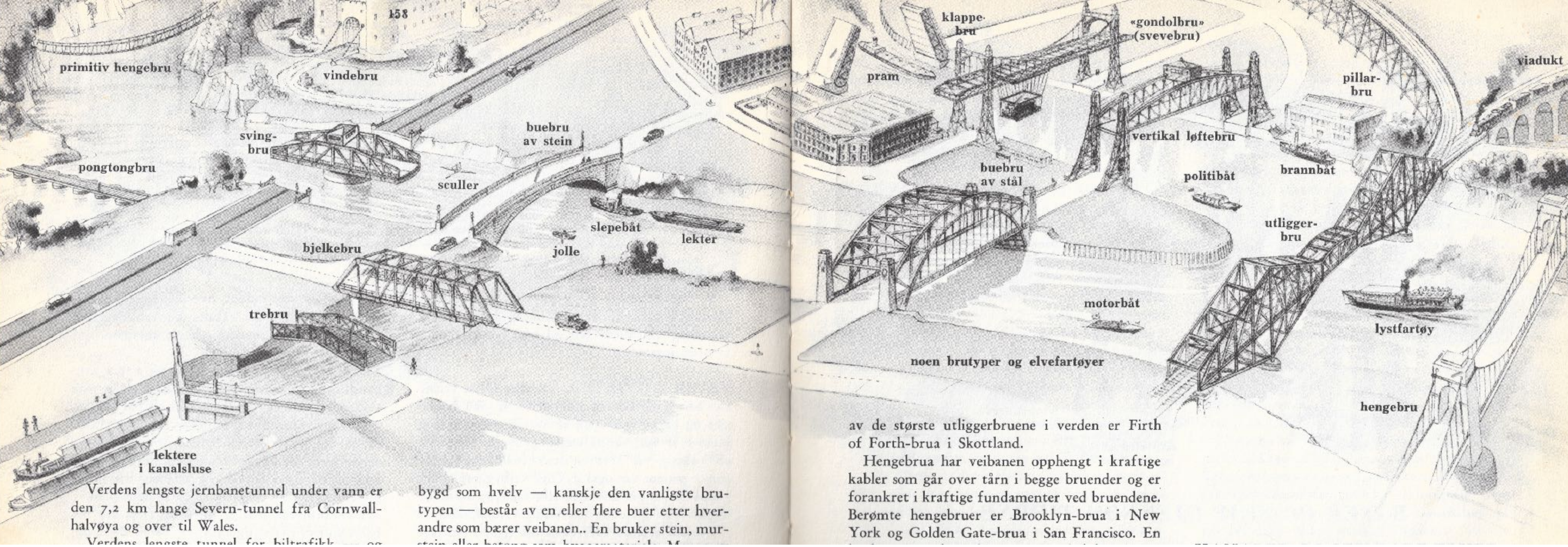
Optional, introduces busy poll, needed *some* times

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

Discussing new runtime scheduler made at NTH (1981)

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

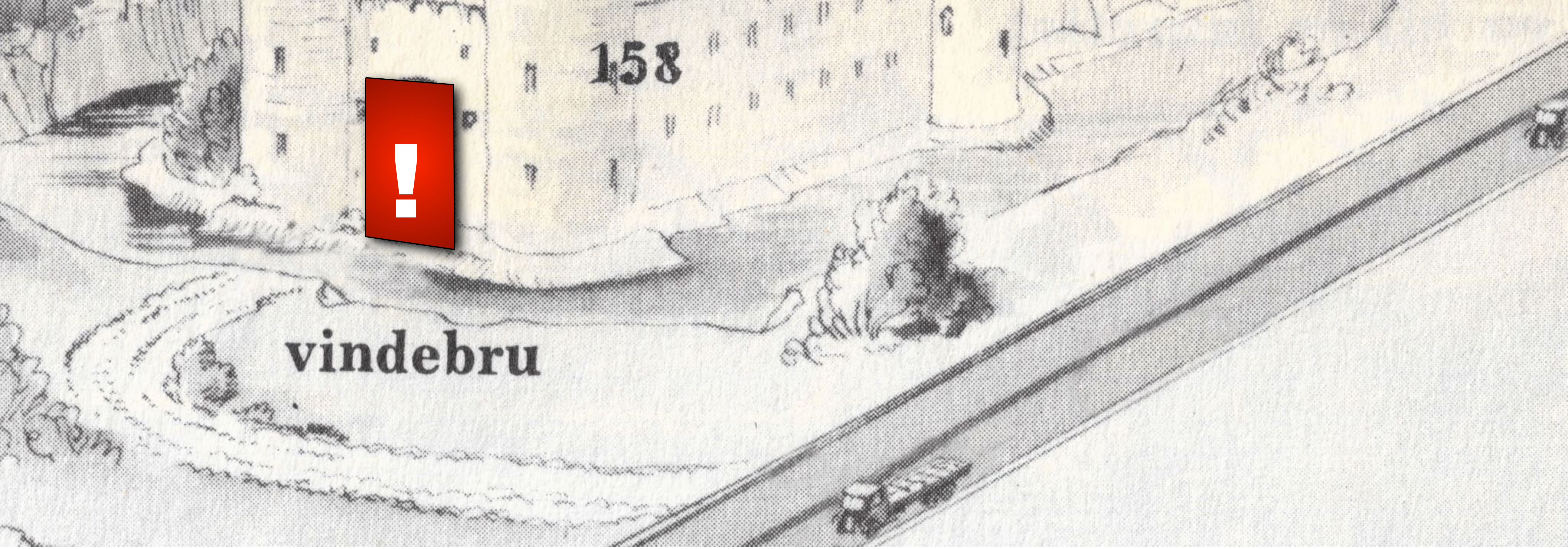
Starting with C CSP-type schedulers (2002)



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

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 allows all traffic in (ok!)
  - ok, if it's not disturbed!
- Now it is protected!
  - Doing something undisturbed in the castle
- I guess that this is the most important page in this lecture!

TERMINOLOGY?

«DRAWBRIDGES»

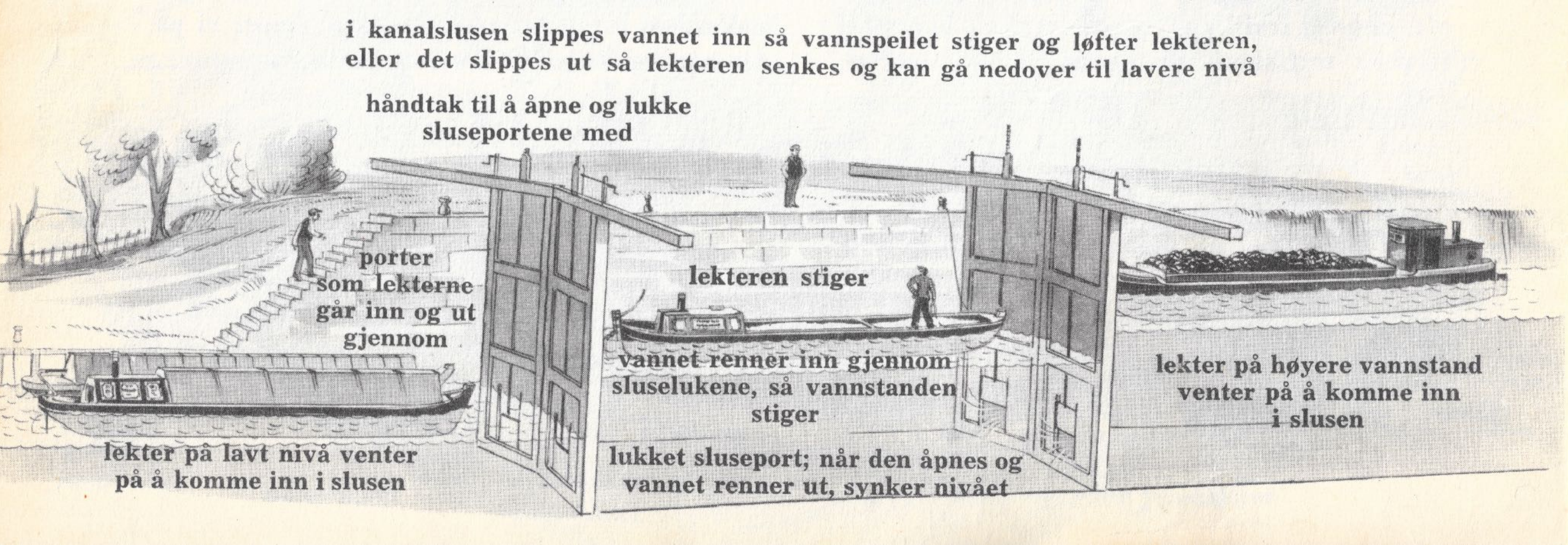
«GATES» THINKING ABOUT IT:

CHANNELS MORE THAN CONNECT THREADS

CSP «MODEL»

guards

THEY PROTECT THEM

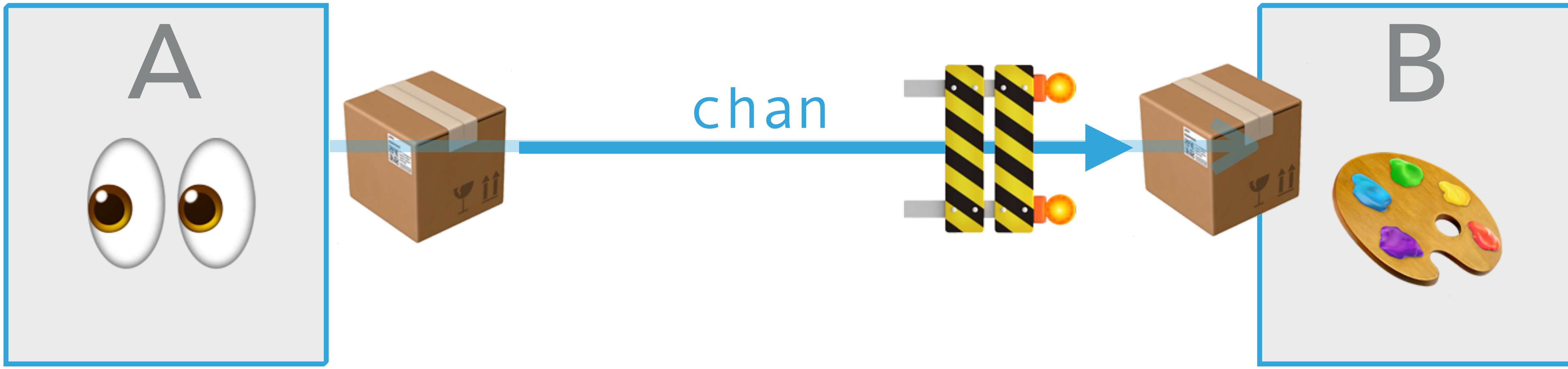


A CANAL LOCK HAS SEMANTICS

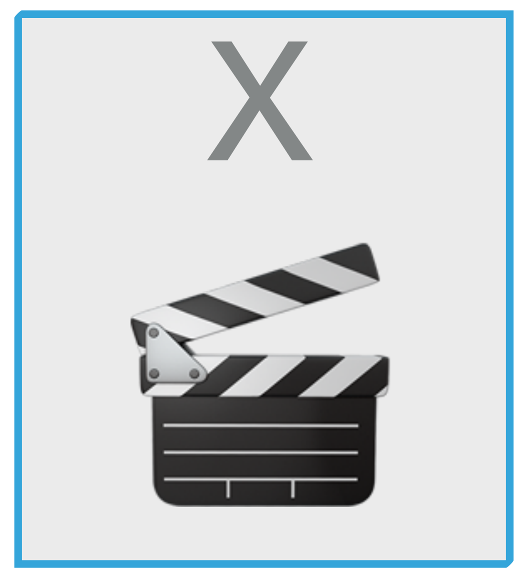
A CHANNEL HAS SEMANTICS

- 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



Has been undisturbed and running all the time!



A: run

first: have result!

B: dance - busy!

second: ready!

wait/sleep/block

send > receive

synchronous
unbuffered

more to do?

thanks! paint

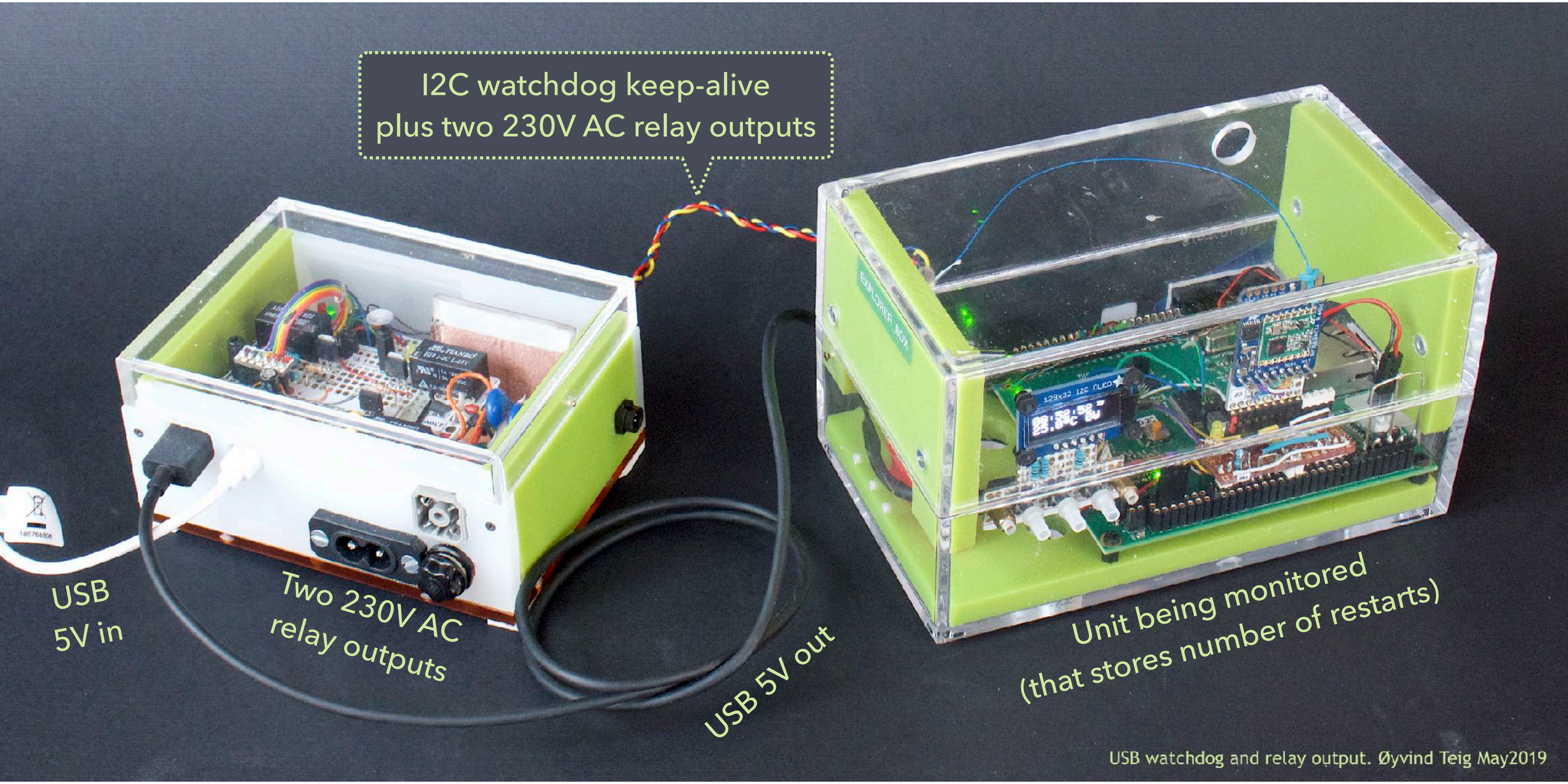
I TALK 🤓 TALK TO YOU, BUT HOW MUCH DID WE LOSE? 😷

- 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?

PAUSE?

MY USB WATCHDOG (AND RELAY OUTPUT) BOX



http://www.teigfam.net/oyvind/home/technology/187-my-usb-watchdog-and-relay-output-box/

CONT?

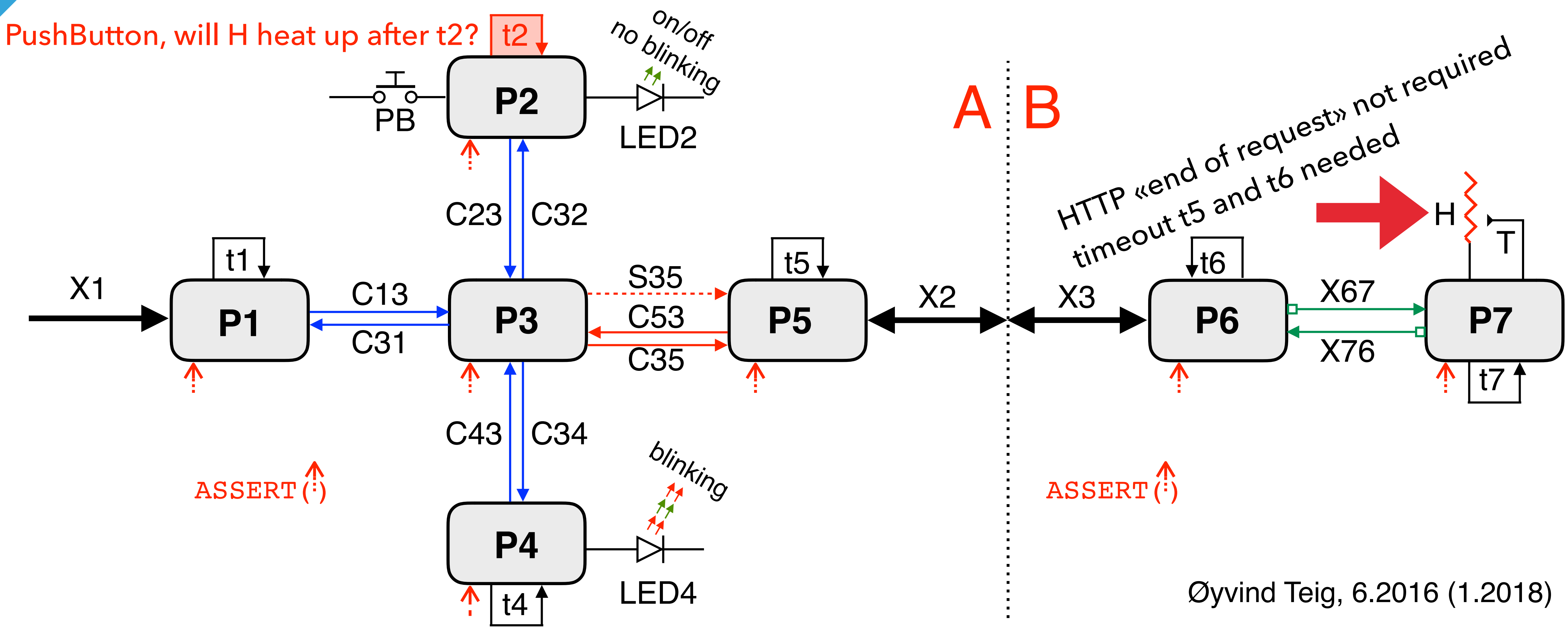
CONT!

From a blog note

AN ADVICE

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

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

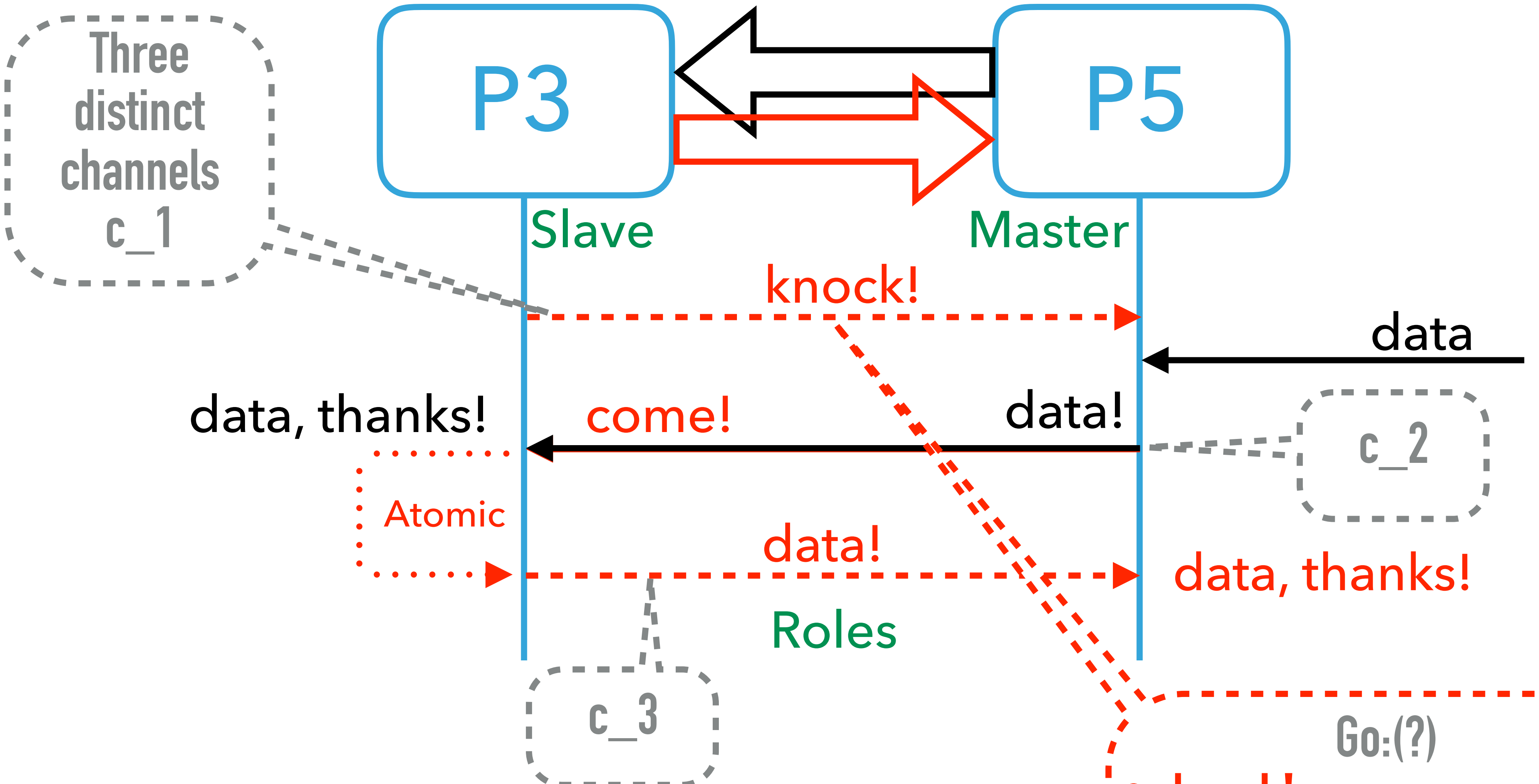


Client/server deadlock free
P1-P3, P2-P3, P4-P3

«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

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

Go:(?)

- knock!
- may be simulated with a
- make (chan int,1)
- that P3 will <u>not</u> re-knock!
- on before
- come!
- has been received
- Thus it will never block

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

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

```go
func Server(in <-chan int, out chan<- int) {
	value := 0     // Declaration and assignment
	valid := false // --"--
	for {
		outc := out // Always use a copy of "out"
		// If we have no value, then don't attempt
		// to send it on the out channel:
		if !valid {
			outc = nil // Makes input alone in select
		}
		select {
		case value = <-in: // RECEIVE?
			// "Overflow" if valid is already true.
			valid = true
		case outc <- value: // SEND?
			valid = false
		}
	}
}
```

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

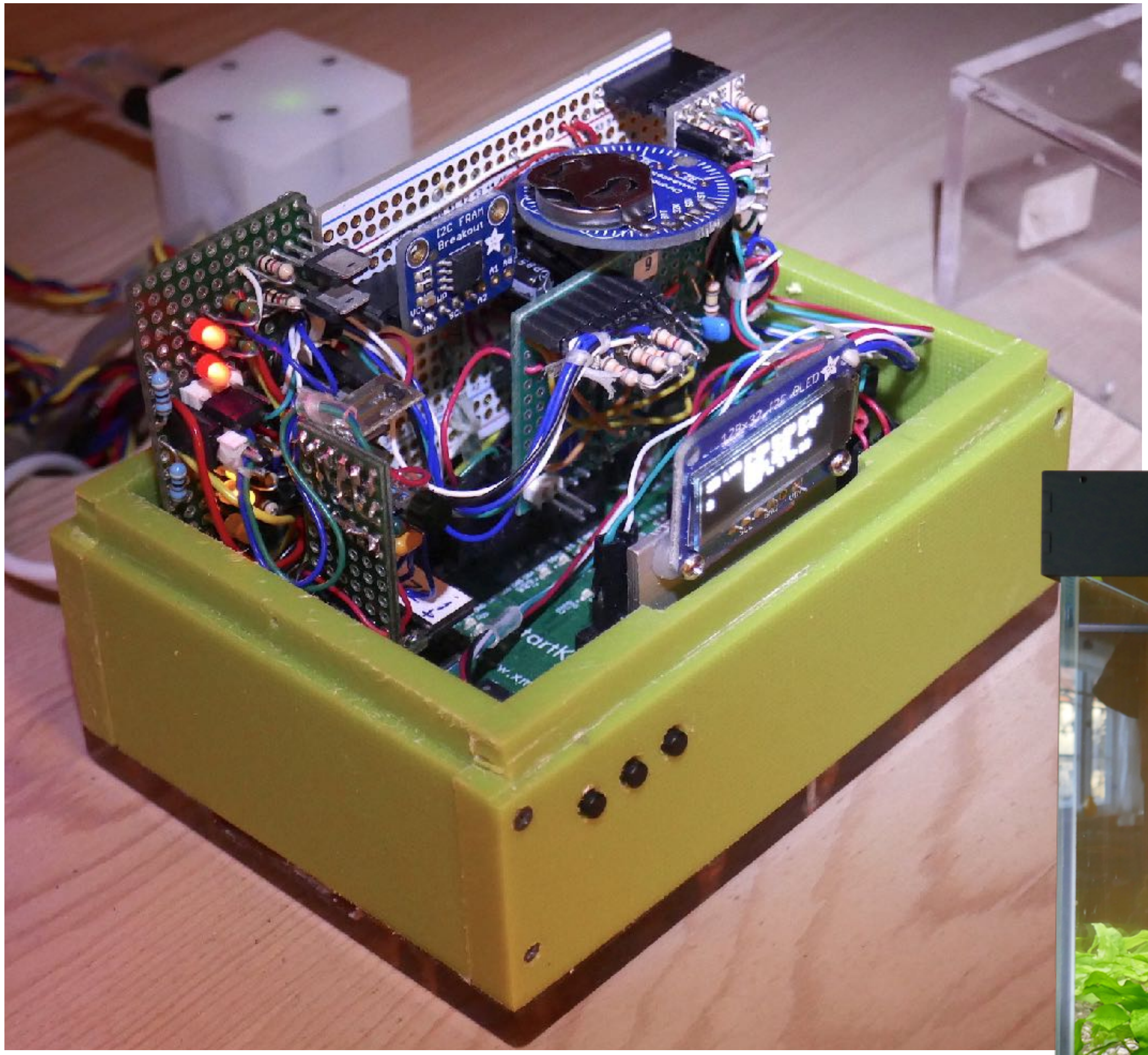
https://www.xmos.com/published/xmos-programming-guide

```
interface if1 {
    void f();
    [[guarded]] void g(); // this function may be guarded in the program
}
..
select {
    case i.f(): {
        ...
    } break;
    case (e == 1) => i.g(): {
        ...
    } break;
}
```

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

As I have already shown, I use this at home:

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



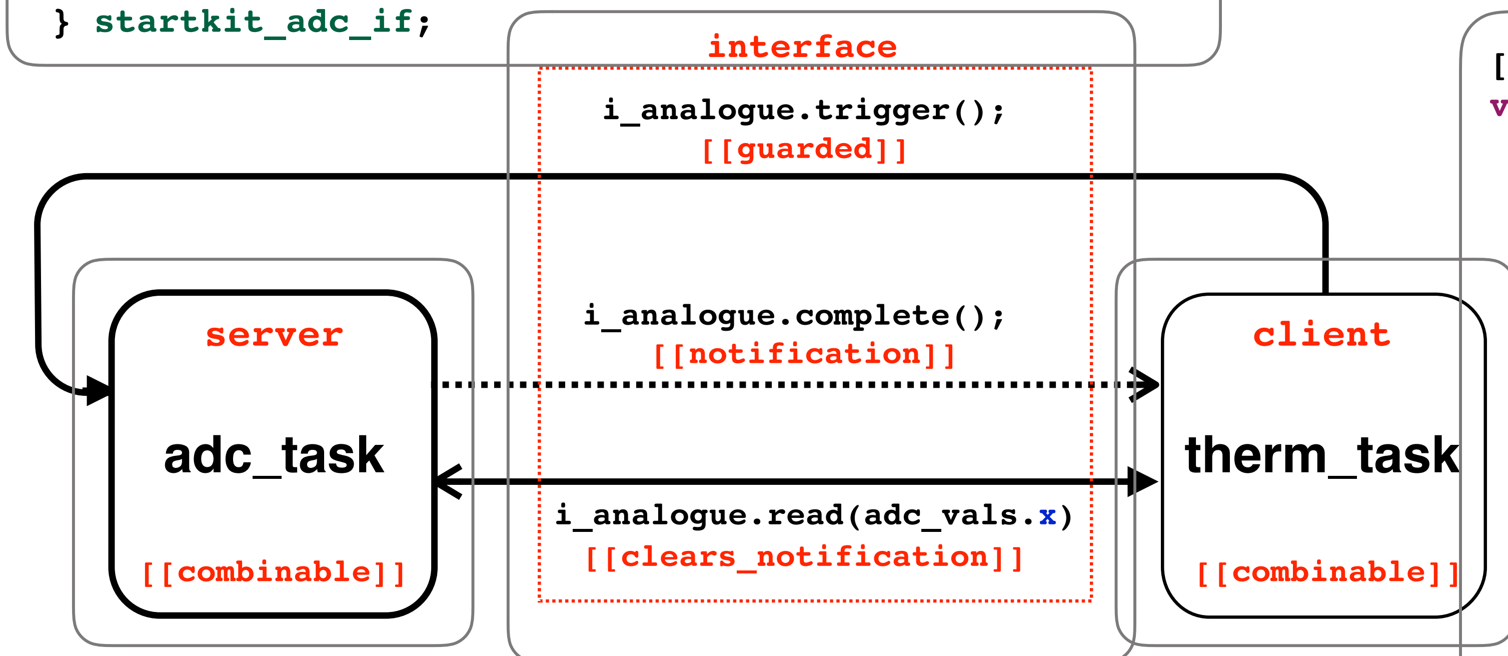
Showing a forest for some trees 2

XMOS xC LANGUAGE FOR THEIR CONTROLLERS. EXTENSION OF C

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

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

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

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»

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?

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
  - Nondeterministic (pseudo random) choice
- XC
  - Nondeterministic (unspecified) choice(?). I have tested it and it seem quite fair
- 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

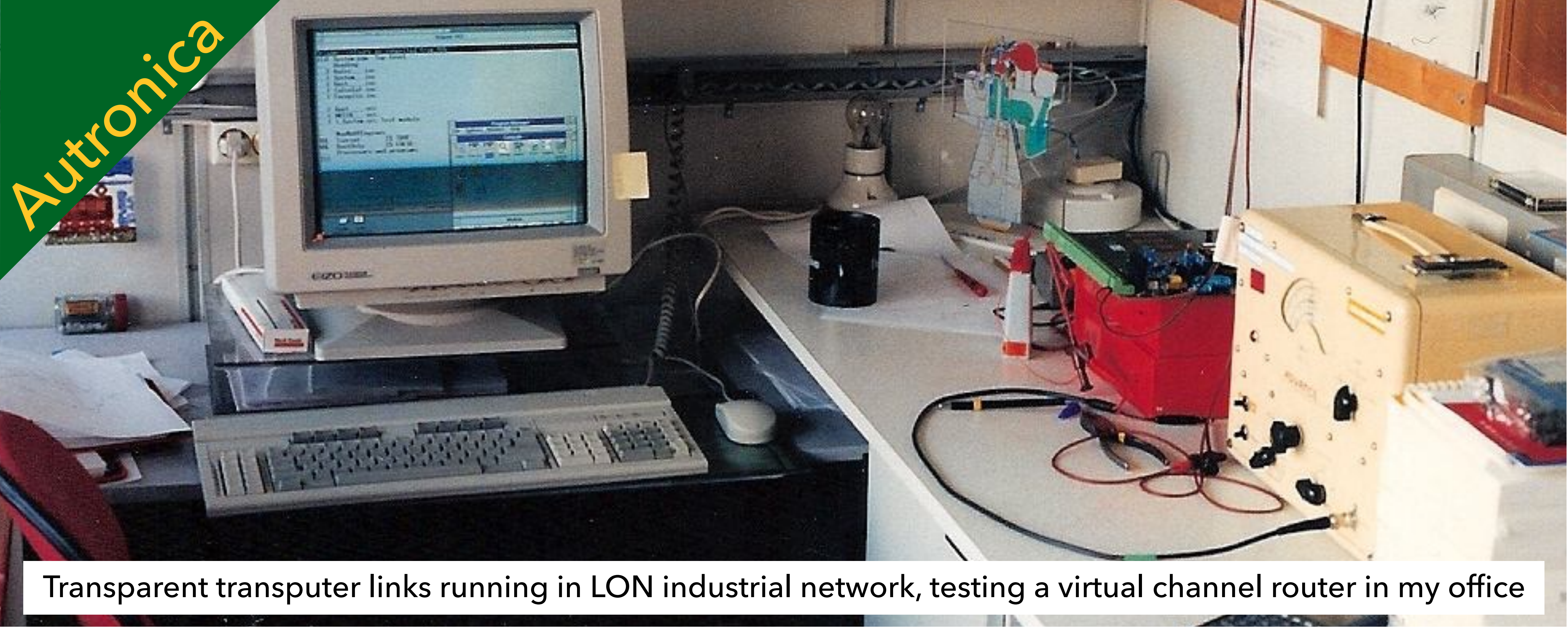
https://www.infoq.com/presentations/clojure-core-async

- 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!

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

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

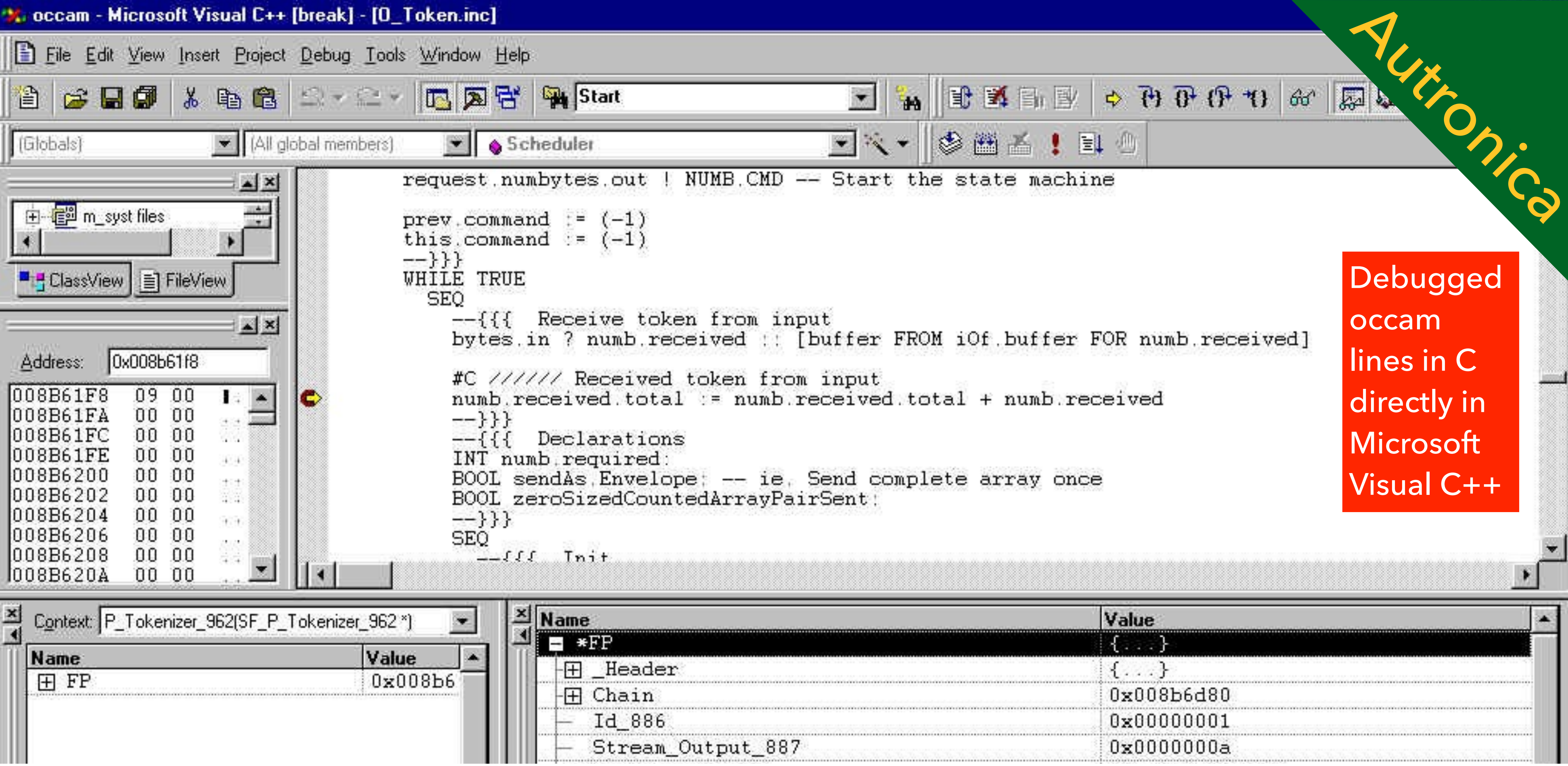
AutroKeeper (2010..)
Chansched scheduler



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

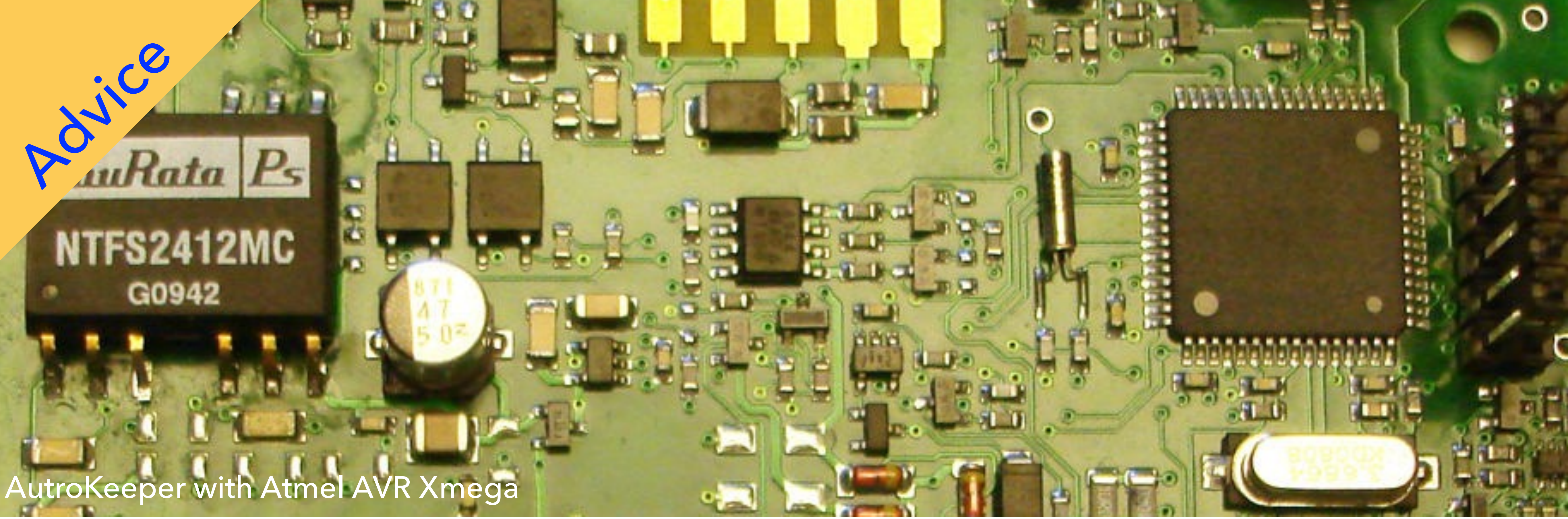
TO ME: NOTHING EVER THE SAME AFTER

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



C? YES: OCCAM TO C: SPOC TOOL

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



AutroKeeper with Atmel AVR Xmega

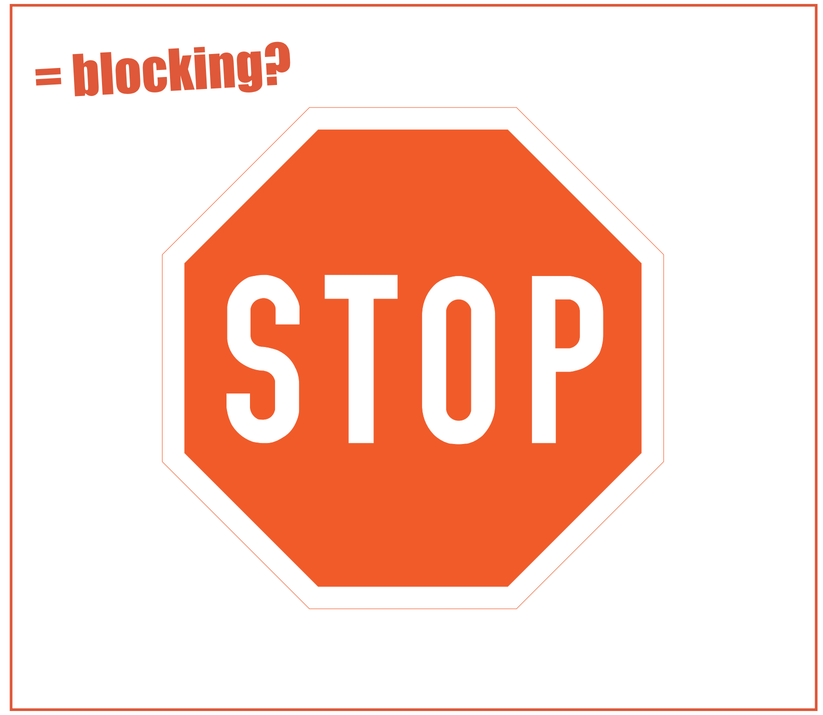
SMALL EMBEDDED SYSTEMS

- 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

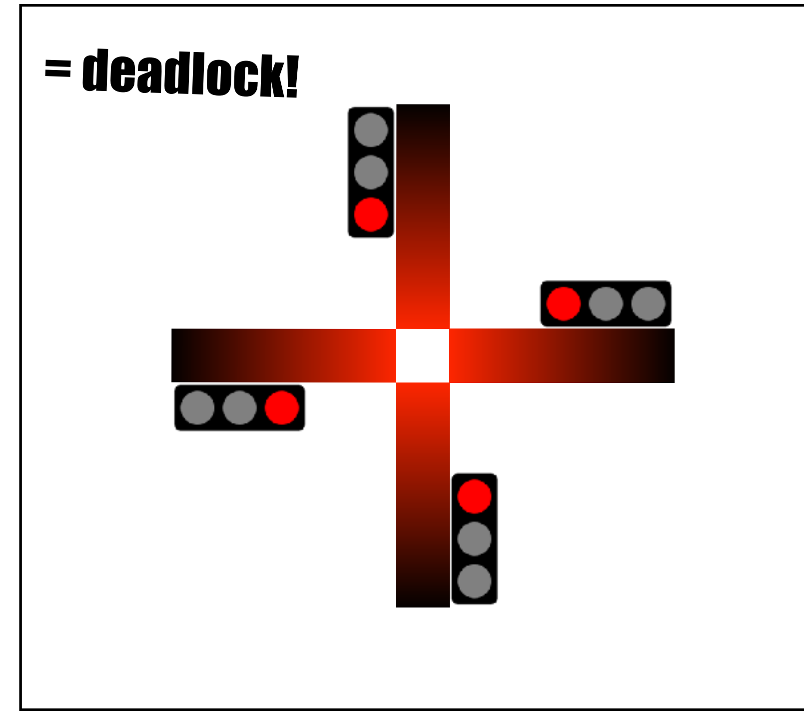
From a blog note

Which block ing do you mean?

The show goes on with this blocking



This blocking stops the show



This blocking stops the world

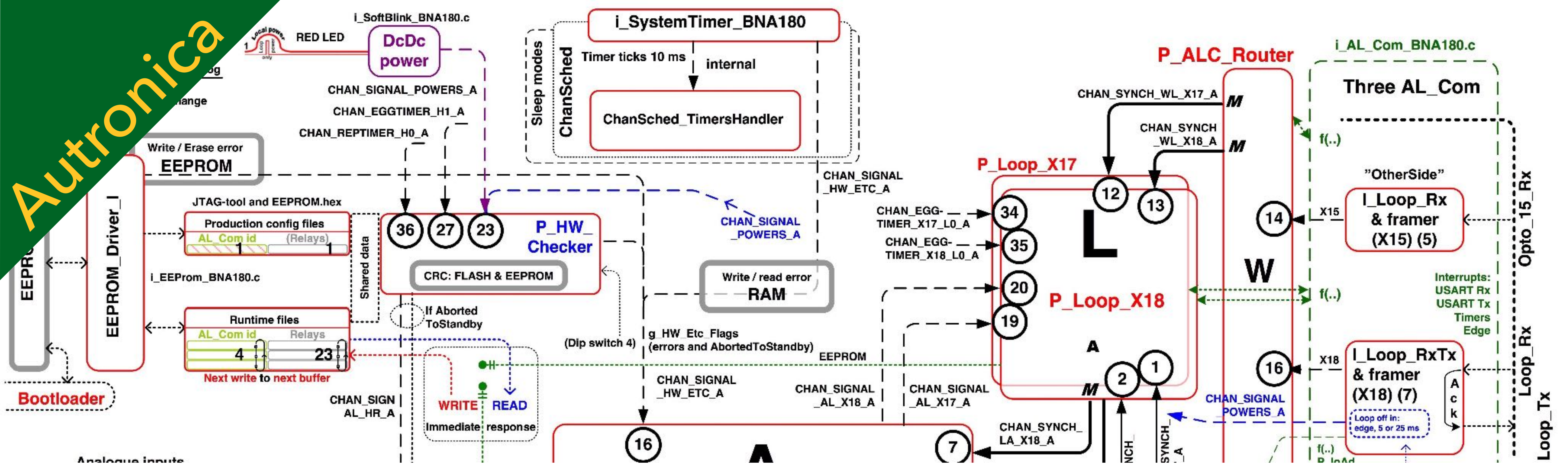
«BLOCKING» EASY TO MISINTERPRET

THINKING ABOUT IT:
CHANNELS MORE THAN CONNECT THREADS
THEY PROTECT THEM

- 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

- 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: 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)

```
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

```
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;
      }
    }
  }
}
```

Sync chan comm needs states

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

:
```

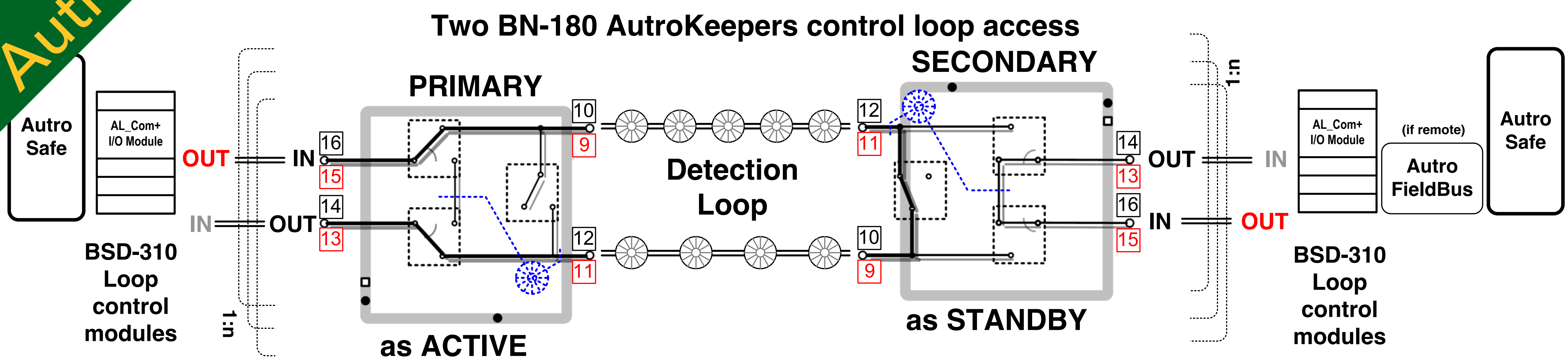
LESS READABLE WHEN PERHAPS:

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

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_EGGREPTIMER_IN (CHAN_EGGTIMER);
      ALT_EGGREPTIMER_IN (CHAN_REPTIMER);
      ALT_SIGNAL_CHAN_IN (CHAN_SIGNAL_AD_READY);
      ALT_CHAN_IN        (CHAN_DATA_2, Data_2);
      ALT_ALTTIMER_IN    (CHAN_ALTTIMER, TIME_TICKS_100_MSECS);
    ALT_END();
    switch (g_ThisChannelId)
    {
      ...  process the guard that has been taken, e.g. CHAN_DATA_2
      CHAN_OUT (CHAN_DATA_0, Data_0);
    };
  }
}
```

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

MATTER #3 SINCE LAST YEAR: XC TASK TYPES

[Unravelling XC concepts [[combine]], [[combinable]], [[distribute]], [[distributable]] and [[distributed(..)]] plus par and on..]

- Task source code not decorated is «normal» task
- Decorated with `[[combinable]]`
  - `= both of the above «asynchronous» interface / channel comms`
- Decorated with `[[distributable]]`
  - `= «synchronous» interface / channel comms`
- Variants: `[[combine]]`, `[[distribute]]`, `[[distributed(..)]]`

MATTER #3 SINCE LAST YEAR: XC TASK TYPES

Task type	Usage
Normal	Tasks run on a logical core and run independently to other tasks. The tasks have predictable running time and can respond very efficiently to external events.
Combinable	Combinable tasks can be combined to have several tasks running on the same logical core. The core swaps context based on cooperative multitasking between the tasks driven by the compiler.
Distributable	Distributable tasks can run over several cores, running when required by the tasks connected to them.

From the XMOS Programming guide

```
interface button_if_t {
    void but (int x);
};

typedef enum {false,true} bool;

[[distributable]] // [[combinable]]
void handle (server interface button_if_t i_but[3]) {
    // int cnt = 0;
    // timer tmr;
    // int   time;
    // bool  timeout = false;
    // tmr :> time;
    while (1) {
        select {
            case i_but[int i].but (int ms) : {
                // Do something
                // timeout = false;
                break;
            }
            // case tmr when timerafter(time) :> void: {
            //     timeout = true;
            //     time += XS1_TIMER_HZ; // One second
            //     break;
            // }
        }
        // cnt++;
    }
}

int main (void) {
    interface button_if_t i_but[3];
    par {
        [[combine]]                  [[combine]]
        par {                        par (int j = 0; j < 3; j++) {
            handle (i_but);              button (i_but[j]);
            button (i_but[0]);       }
            button (i_but[1]);      [[distribute]] // [[combine]]
            button (i_but[2]);       par {
         }                               handle (i_but);
                                     }
    }
    return 0;
}
```

Normal
Combinable
Distributable

XC

1

```
Constraint check for tile[0]:
  Cores available:           8,   used:          4 .  OKAY
  Timers available:         10,   used:          4 .  OKAY
  Chanends available:       32,   used:          6 .  OKAY
  Memory available:      65536,   used:       1464 .  OKAY
    (Stack: 372, Code: 882, Data: 210)
```

2

3

```
Constraint check for tile[0]:
  Cores available:           8,   used:          1 .  OKAY
  Timers available:         10,   used:          1 .  OKAY
  Chanends available:       32,   used:          0 .  OKAY
  Memory available:      65536,   used:       1852 .  OKAY
    (Stack: 404, Code: 1228, Data: 220)
Constraints checks PASSED.
```

4

```
Constraint check for tile[0]:
  Cores available:           8,   used:          1 .  OKAY
  Timers available:         10,   used:          1 .  OKAY
  Chanends available:       32,   used:          0 .  OKAY
  Memory available:      65536,   used:       1756 .  OKAY
    (Stack: 404, Code: 1132, Data: 220)
Constraints checks PASSED.
```

5

```
Constraint check for tile[0]:
  Cores available:           8,   used:          2 .  OKAY
  Timers available:         10,   used:          2 .  OKAY
  Chanends available:       32,   used:          4 .  OKAY
  Memory available:      65536,   used:       1728 .  OKAY
    (Stack: 376, Code: 1090, Data: 262)
Constraints checks PASSED.
```

6 Wrong error message

```
../src/main.xc:366:1: error: distributed statement must be a call
                             to a distributable function
```

Normal
Combinable
Distributable
Elegant but difficult

MY XCORE-200 EXPLORERKIT BOARDS' PROCESSOR

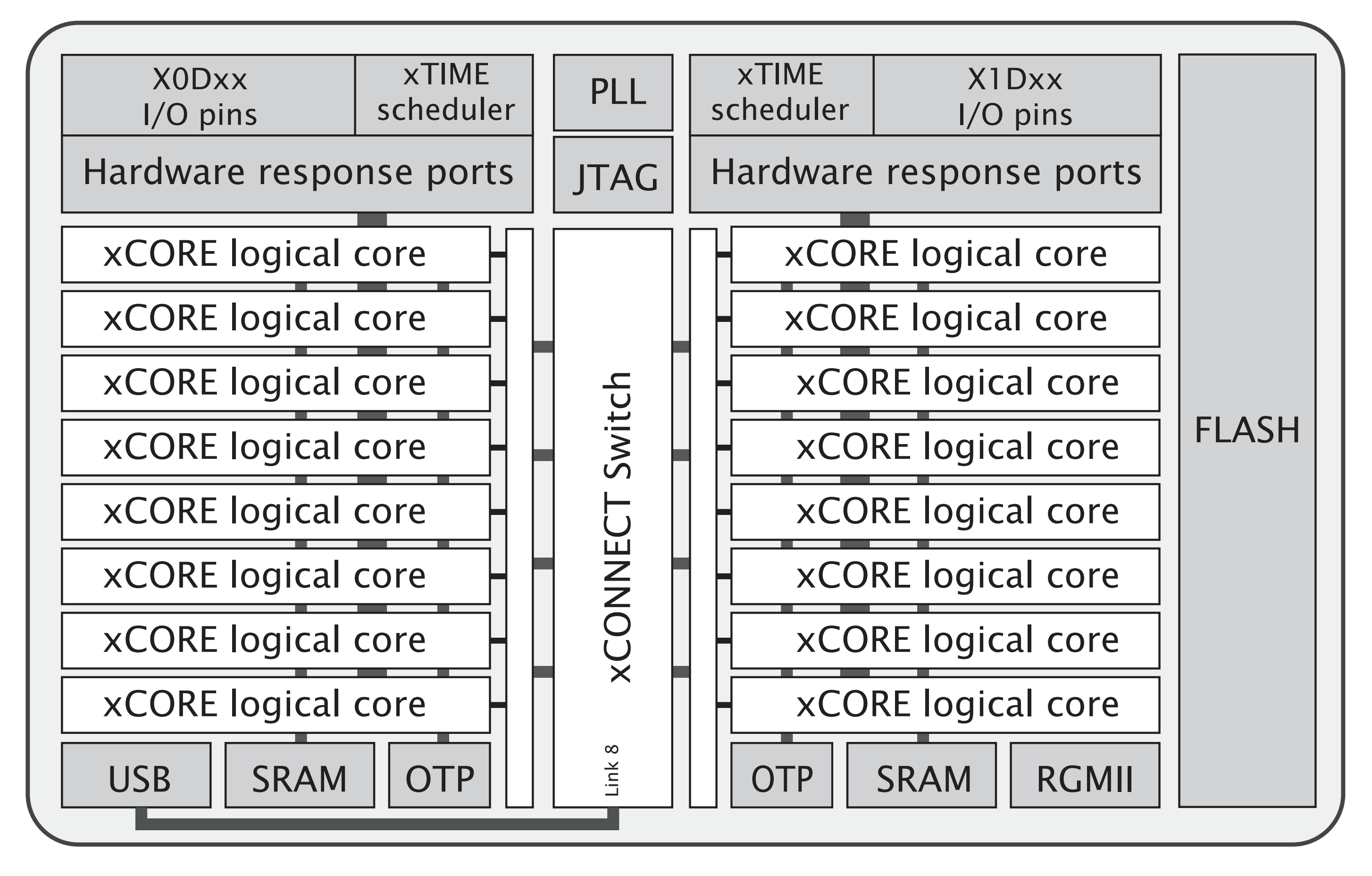


Figure 1: XEF216-512-TQ128 block diagram, from **XEF216-512-TQ128 Datasheet**. 2018/03/23 Document Number: X006990 http://www.xmos.com/download/private/XEF216-512-TQ128-Datasheet%281.15%29.pdf. As used in the xCORE-200 eXplorerKIT.

- 2 tiles (500 MIPS per tile (or dual))
- 8 cores per tile (=«Logical cores»)
- xTIME scheduler. If # cores active:
  - 1-4 cores: 1/4 cycles each
  - 5-8 cores: all cycles shared out
  - Deterministic thread execution
  - Thread safe
  - pragma for some deadlines
- Channels: untyped. Synch or asynch
  - XC `chanends` (32 per tile)
  - Not between tasks on the same core
- XC `interface` (typed and role/session)
  - *May* use `chanends` or locks or sharing of `select` or context (blocks of state data)
- Shared memory & no data bus contention
  - No cache
  - No DMA
  - I/O does not use memory bus
- Also supported/used by XC
  - Locks (4 per tile). Runtime
  - I/O ports
  - Clock blocks (6 per tile)
  - Timers (10 pr tile)

INSIDE THE TOOL CHAIN (FROM AN INSIDER)

- The xCore compiler handles the «lowering of interfaces» onto statically and dynamically allocated channel resources
- Program Content Analysis (optional but on by default) into a pca-file (xml)
- Compilation into Abstract Syntax Tree
  - Specialisation stage using pca-file
  - The XC compiler will generate multiple versions of «interface lowered» code
    - for when the server and client are on different tiles or cores
    - for when the server and client are actually combined
    - for when the server and client are actually distributed
    - for when a server may need to be re-entrant (yielding), due to a possible calling cycle
- The linker runs, linking together the object code, and throwing away unused (non specialised) functions
- In an .s-file there would be duplicate content but with different boiler plating regarding how chanends and blocks of state data (holding chanends) are used

Code example showing scheduling:
http://www.teigfam.net/oyvind/home/technology/165-xc-code-examples/#scheduling

SUMMARY (XC)

- To utilise the HW resources better
  - Cores and channels
- To allow the user to fully code with tasks
  - Not only one per logical core
- These distinctions are really general and could probably be used by many to make multitasking as expensive / affordable as needed only

MATTER #4 SINCE LAST YEAR: TASK TYPES EVEN FOR EMBEDDED ADA?

My blog note 035 mentions the Ravenscar and Jorvik profiles
Leveraging real-time and multitasking Ada capabilities to small microcontrollers in Journal of Systems Architecture (March 2019) by Rivas and Tajero
How Embedded Applications using an RTOS can stay within On-chip Memory Limits by Robert Davis, Nick Merriam, Nigel Tracey at www.realogy.com (2000)

- The Ravenscar profile limits the tasking model quite a lot
  - It is for safety critical systems written in Ada. It basically takes the rendezvous and select statements away and uses protected types and objects instead
  - This opens for schedulability analysis
- The now being worked on Jorvik profile seems to limit the limitations somewhat
- Rivas and Tajero have just recently suggested a task model where the stack is reused. Also starts off with Ravenscar
  - «In this paper we present a new Ada run-time environment that includes a new scheduling policy based on the one-shot task profile that simplifies the implementation of the Ada tasking primitives and allows stack sharing techniques to be applied»
  - Much like [[distributable]]?
  - Also has requirements of code: «we need to restrict the structure of the tasks’ body to the one expected for a one-shot task»
  - The idea seems to stem from a paper from the year 2000 by Davis et al

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!

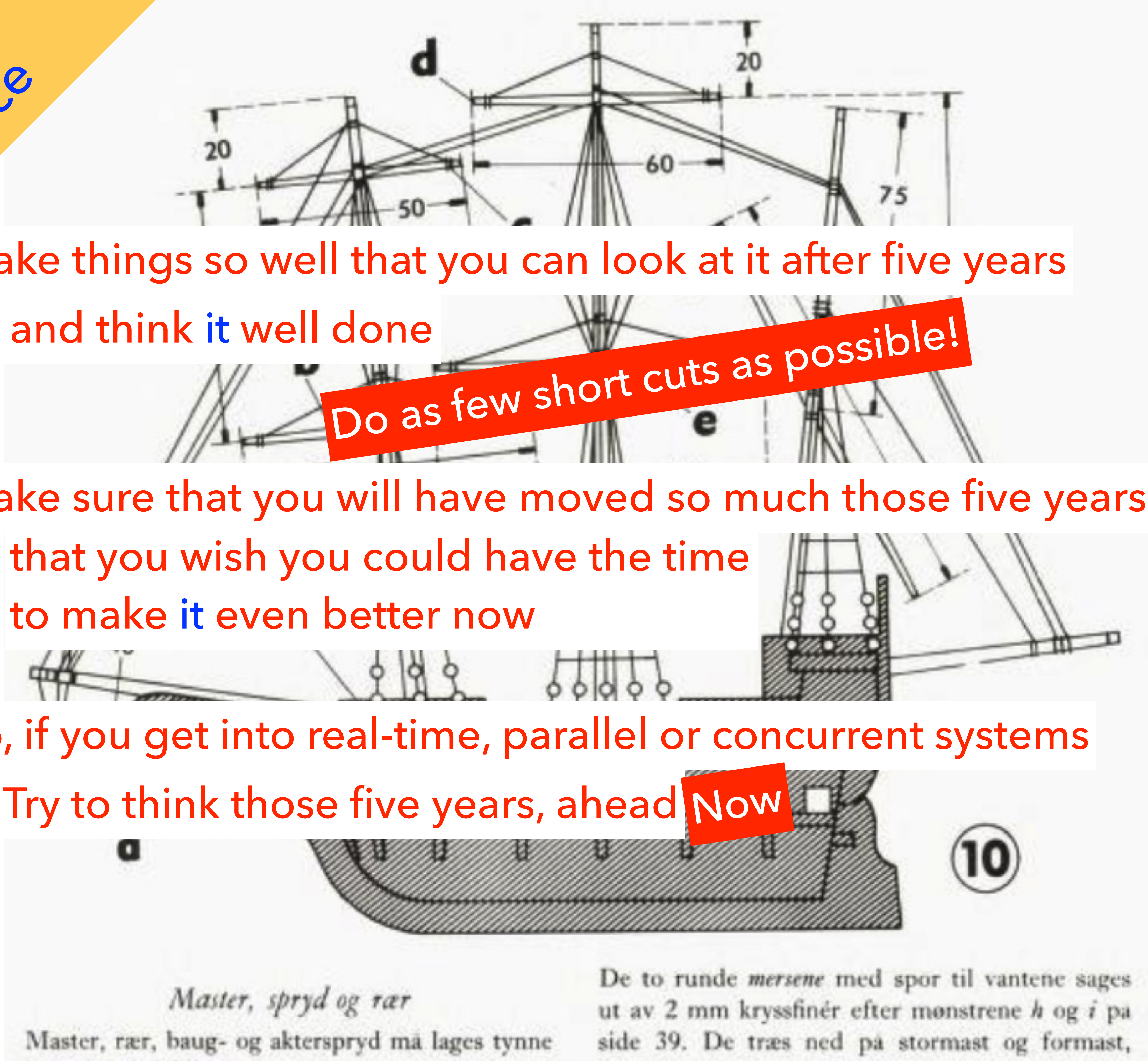
Make sure that you will have moved so much those five years

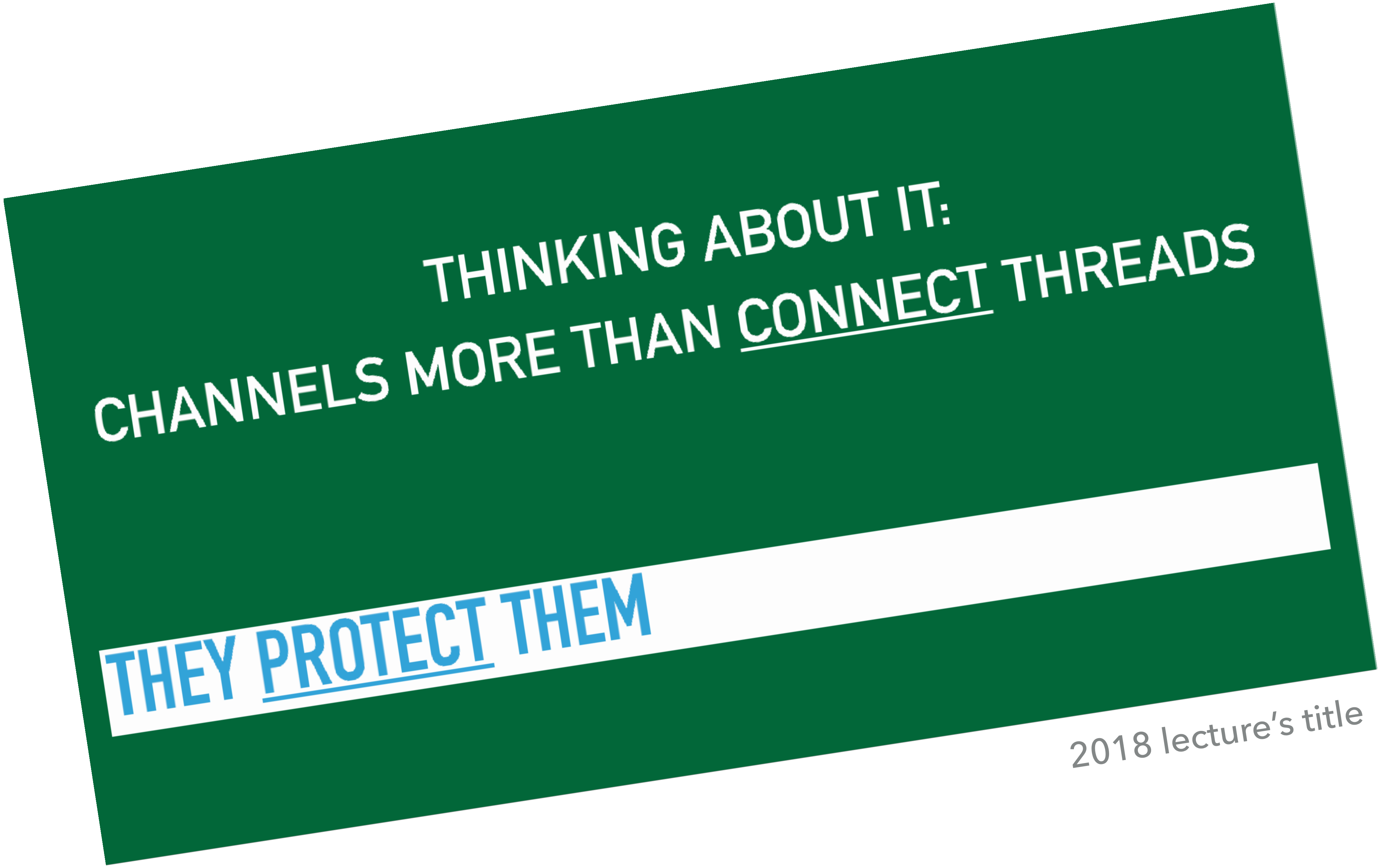
that you wish you could have the time

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





2018 lecture's title

HOW DO THEY PROTECT THEM?
SUMMARY:

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»

CHANNELING AGAINST THE FLOW

WHAT DID I MEAN BY THIS?

- It’s easiest if *you*, your *project* and your *boss* agree to program in Go and need concurrency (goroutines, channels)
- It’s under pressure if you agree on Ada but need the safety critical profile
- It’s utmost difficult if you have an embedded controller and need concurrency. I would know
- Don’t always take the culture «as is». Try challenging it

oyvind.teig@teigfam.net

- This lecture
  - Full quality, each page only once, no build steps (around 76 MB) http://www.teigfam.net/oyvind/pub/NTNU_2019/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..

- **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

(More) questions?

Thank you!