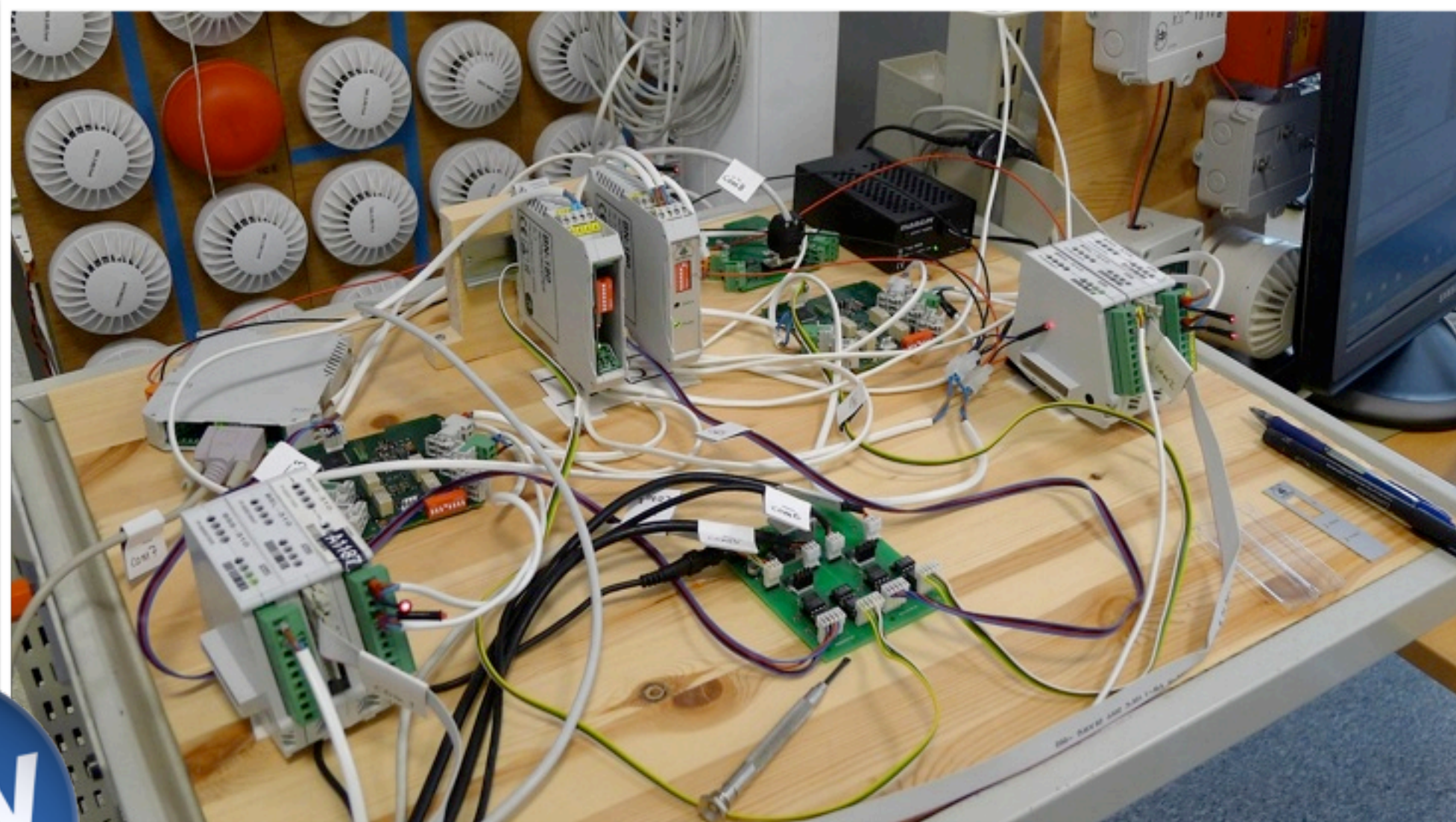


Fra harde μ -sekunder til forte år: sann tid i industrien



Øyvind Teig

senior utviklingsingeniør
Autronica

Gjesteforelesning, 17. april 2012

NTNU, fag TTK 4145 Sanntidsprogrammering (Real-Time Programming)

Autronica Fire and Security (AFS)

- ✓ Ca. 400 ansatte med ca. 25 på utvikling
- ✓ 6-700 mill NOK omsetning
- ✓ På Lade, her i Trondheim
- ✓ Samme sted, og for meg - seks logoer
- ✓ Begynte med "gründerne" i 1957
- ✓ Børs og forskjellige eiere



Autronica Fire and Security (AFS)

Siden 2005

- ✓ "A UTC Fire & Security Company"
United Technologies Corporation
- ✓ Ca. 200.000 ansatte
- ✓ Otis, Pratt & Whitney, Hamilton Sundstrand,
Carrier, Autronica, etc..

"Disclaimer"

Punktene i denne forelesningen står for egen regning!

- Datafaglige erfaring og meninger,
erfart, sett og lest
- Ingen Autronica-sensitiv informasjon
- Autronicas produkter godkjennes etter aktuelle normer

CV

- ✓ NTH, 1975
- ✓ Autronica, 1976-2012 (+studentjobbing der fra 1972)
- ✓ HW og SW (mest SW)
- ✓ Har jobbet med embeddedsystemer hele tida
- ✓ Publisert en del - relativt uvanlig i industrien
- ✓ Hjemmesiden min: <http://www.teigfam.net/oyvind>

«Pappesken» (SW)

- ✓ HW/SW for 30+ år i den pappesken
- ✓ Diverse assembler (78-80)
- ✓ PL/M (80-90)
- ✓ Modula-2 (88-90)
- ✓ MPP-Pascal (82-88)
- ✓ occam (90-01)
- ✓ C (02-nå)
- ✓ Java (97-00)
- ✓ Perl (02)



«Pappesken» (SW)

Øverst til høyre (fra diplommen
1975)

Den papirtapen inneholder
fremdeles programmet!



Sann tid
går alt for fort!

«Pappesken» (SW)

Øverst til høyre (fra diplommen
1975)

Den papirtapen inneholder
fremdeles programmet!



- ✓ Sanntidsprogrammering er gøy!
- ✓ Det kommer til å bli mye mer av det!
- ✓ Inkludert "concurrency"



«Pappesken» (sanntidsmetodikk & produkt)

- ✓ 1978: Ikke noe kjøresystem, ren assembler
Dieselaggregat start/stopp nødstrøm
- ✓ 1979: MPP Pascal med prosessbegrep
Protokollkonvertering, nivåmåling og brannvarsling
- ✓ 1980: PL/M med NTH-utviklet kjøresystem
Maskinromsovervåking
- ✓ 1982: Assembler med jobbsnekret kjøresystem
Brannvarsling
- ✓ 1988: PL/M med jobbsnekret kjøresystem
Brannvarsling

«Pappesken» (sanntidsmetodikk & produkt)

- ✓ 1988: Modula-2 med kjøpt kjøresystem
Brannvarsling
- ✓ 1990: occam med prosessbegrep
Motormålinger og effektberegninger
- ✓ 1995: C med VxWorks opsys/kjøresystem
Brannvarsling
- ✓ 2003-8: C med CSP kanaler oppå SDL kjøresystem
(AutroLooper)
Brannvarsling
- ✓ 2009: "ChanSched" stand-alone kjøresystem, med kollega..
Brannvarsling
- ✓ 2010: ..brukt i patentert enhet (AutroKeeper) i cruiseskip

Lange år

- ✓ OO er fra sekstitallet
 - ❖ I bruk nå, men lite i embedded (blir mer og mer!)
- ✓ Sanntidsmekanismene er fra 60-70+ tallet
 - ❖ Semafor etc. i bruk siden da
- ✓ CSP etc. fra 70-tallet. Lite i bruk (ennå..(?))
- ✓ UML 2.0 er stort og uten aksjonsspråk
- ✓ Java -"synchronized" - det første allment brukte språket som har concurrency-tankegang innebygd..
 - ❖ ..og som er "livsfarlig" å bruke!
 - ❖ ..men som det går an å bygge CSP oppå!

Lange år

- ✓ OO er fra sekstitallet
 - ❖ I bruk nå, men lite i embedded (blir mer og mer!)
- ✓ Sanntidsmekanismene er fra 60-70-tallet
 - ❖ Semafor etc. i bruk siden da
- ✓ CSP etc. fra 70-tallet. Lite i bruk (ennå?)
- ✓ UML 2.0 er stort og uten aksjonsspråk
- ✓ Java - "synchronized" - det første allment brukte språket som har concurrency-tankegang innebygd..
 - ❖ ..og som er "livsfarlig" å bruke!
 - ❖ ..men som det går an å bygge CSP oppå!

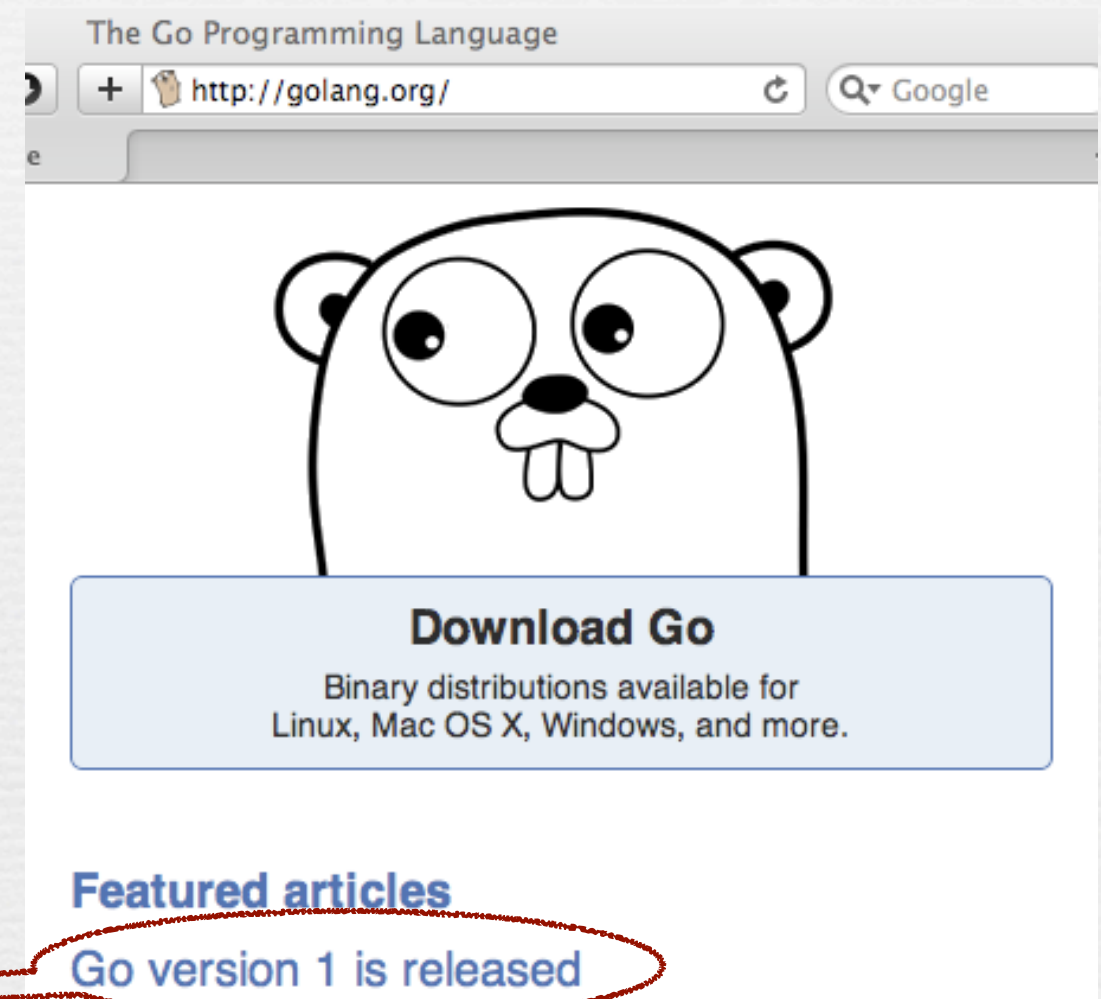
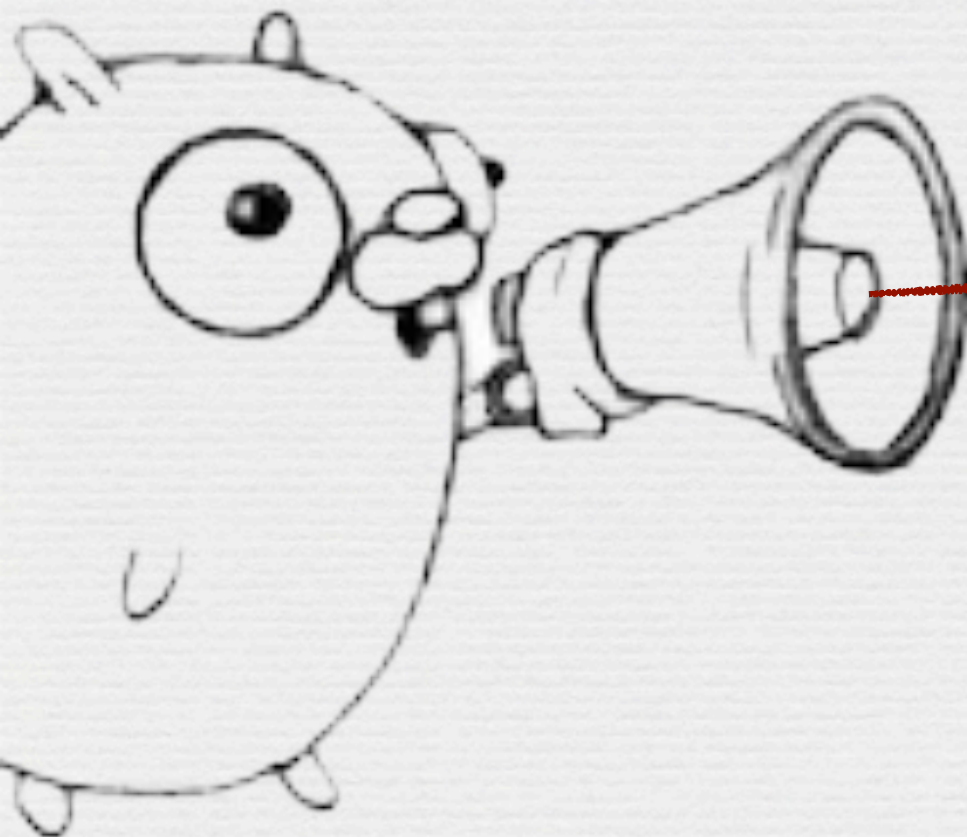
Dette går da ikke fort!

- ✓ OO er fra sekstitallet
 - ❖ I bruk nå, men lite i embedded (blir mer og mer!)
- ✓ Sanntidsmekanismene er fra 60-70+ tallet
 - ❖ Semafor etc. i bruk siden da
- ✓ CSP etc. fra 70-tallet. Lite i bruk (ennå?)
- ✓ UML 2.0 er stort og uten aksjonsspråk
- ✓ Java -"synchronized" - det første allment brukte språket som har concurrency-tankegang innebygd..
 - ❖ ..og som er "livsfarlig" å bruke!
 - ❖ ..men som det går an å bygge CSP oppå!

21 år etter occam (i 2009)

- ✓ **Go** fra Google plukket med seg litt occam!
- ✓ chan, go, select, eksplisitt typekonvertering, array-oppdeling, ikke pekeraritmetikk
- ✓ Scheduling av prosesser på prosessorer i Go som KRoCs kjøresystem for bla.a. occam-pi (fra trådbasseng) (<http://www.cs.kent.ac.uk/pubs/2009/2928/index.html>)
- ✓ Ikke "sikkert" (no «parallel usage rules»)

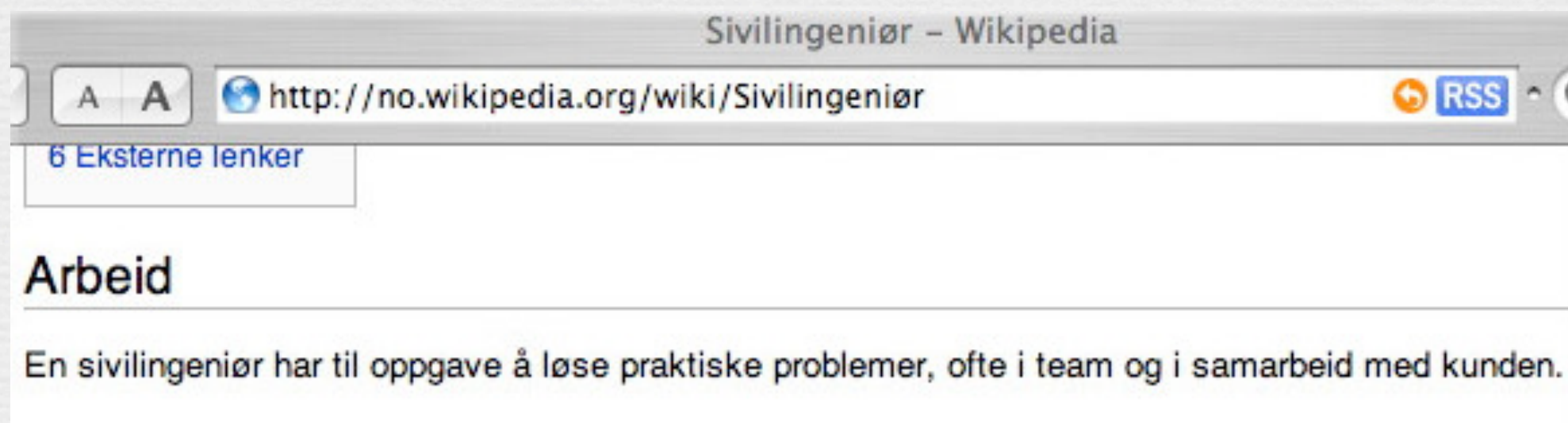
24 år etter occam (i 2012)



28. mars!

Etter forelesningen ønsker jeg

- ✓ At dere skal huske at å jobbe med embedded-systemer, er gøy
 - ➔ ...
- ✓ "En sivilingeniør har til oppgave å løse praktiske problemer, ofte i team og i samarbeid med kunden". (Wikipedia 2007)
- ✓ At dere skal huske å fortelle om det dere har lært i dette faget når dere havner i slike team!



Små embeddedsystemer i industrien

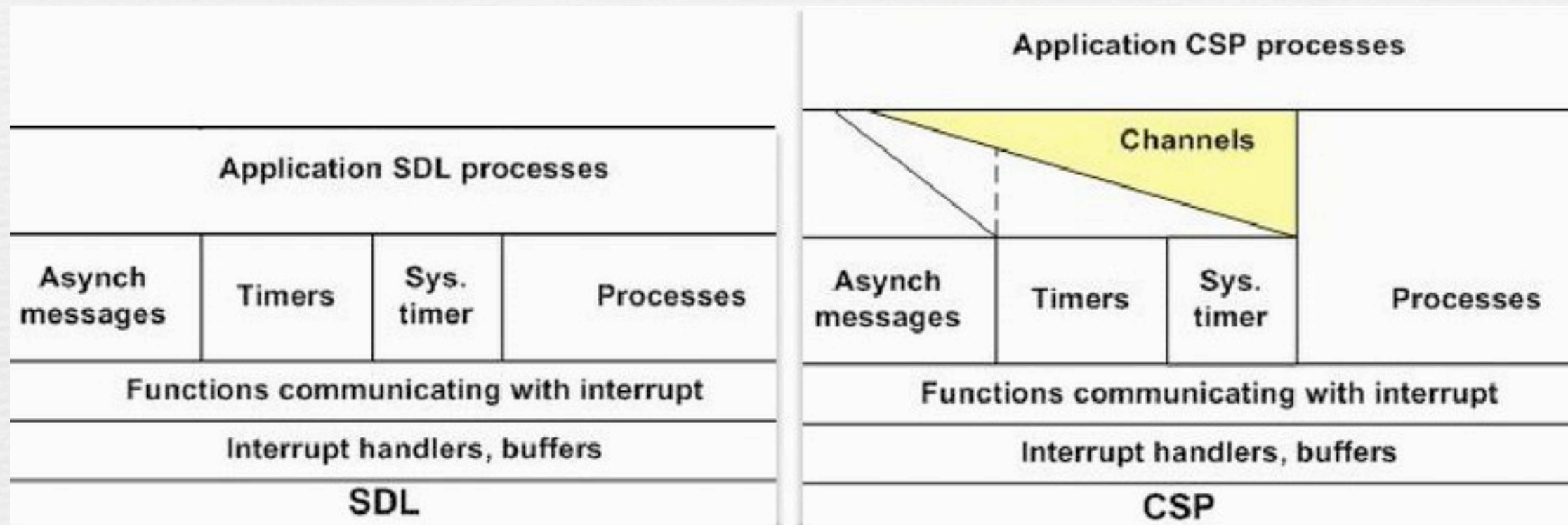
- ✓ Vil nok forholde seg til C en god stund til!
- ✓ Vil nok ha prosjekt- ledere og -deltakere som (bare) kan asynkrone system en god stund til!
- ✓ Ikke overta deres vertøykasse uten å legge synkrone kanaler og tette prosesser oppi:
- ✓ *Sikre* bufrede kanaler (asynkron til full)

Disposisjon

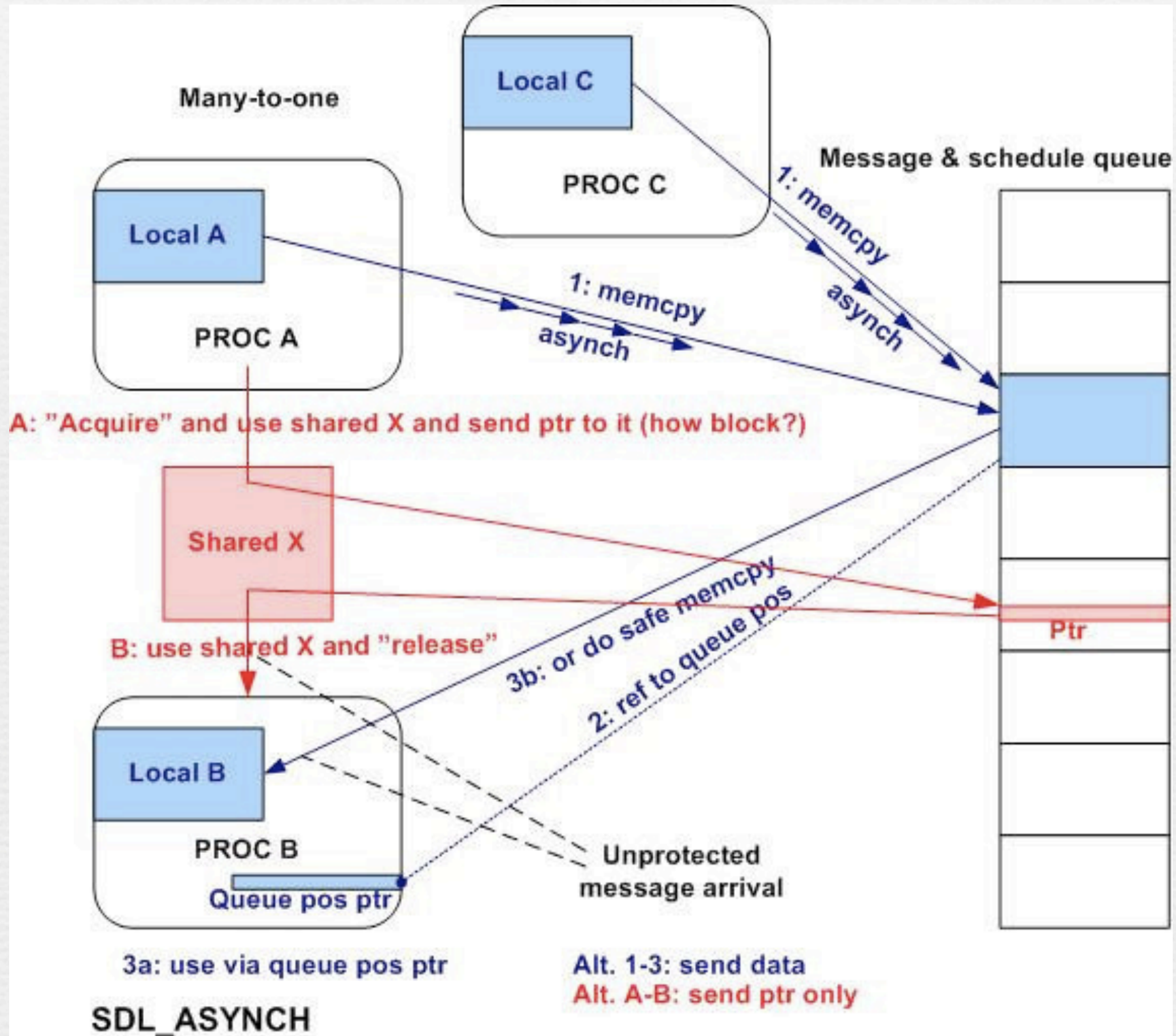
For små (AVR/XMega) system i ANSI C

1. Begynte vi med et asynkront **SDL** kjøresystem (brukes mye!)
2. La så et synkront kanal-lag oppå **CHAN_CSP** (brukes)
3. Har nå lagd et nakent synkront kjøresystem **ChanSched** (brukes)

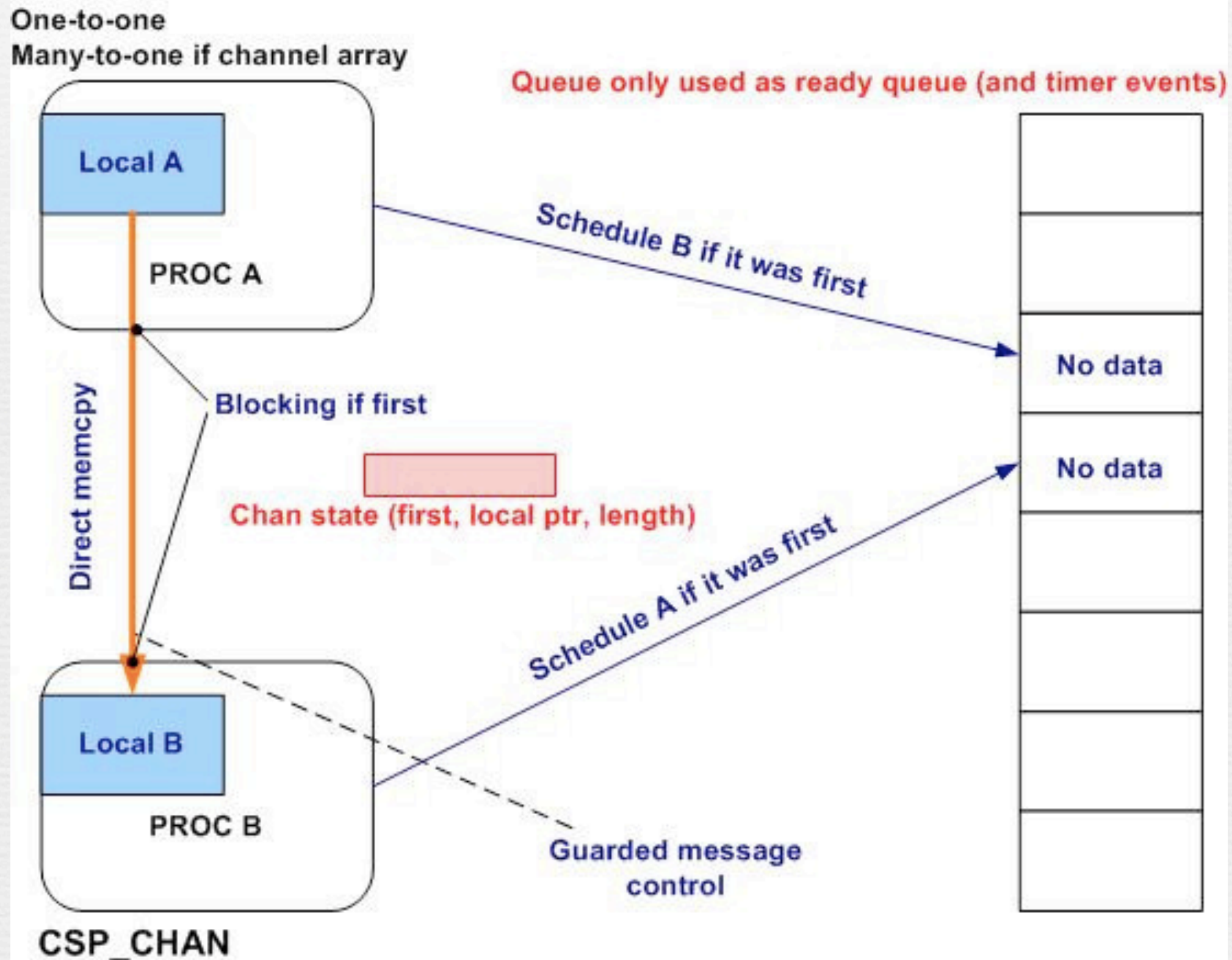
1: CHAN_CSP



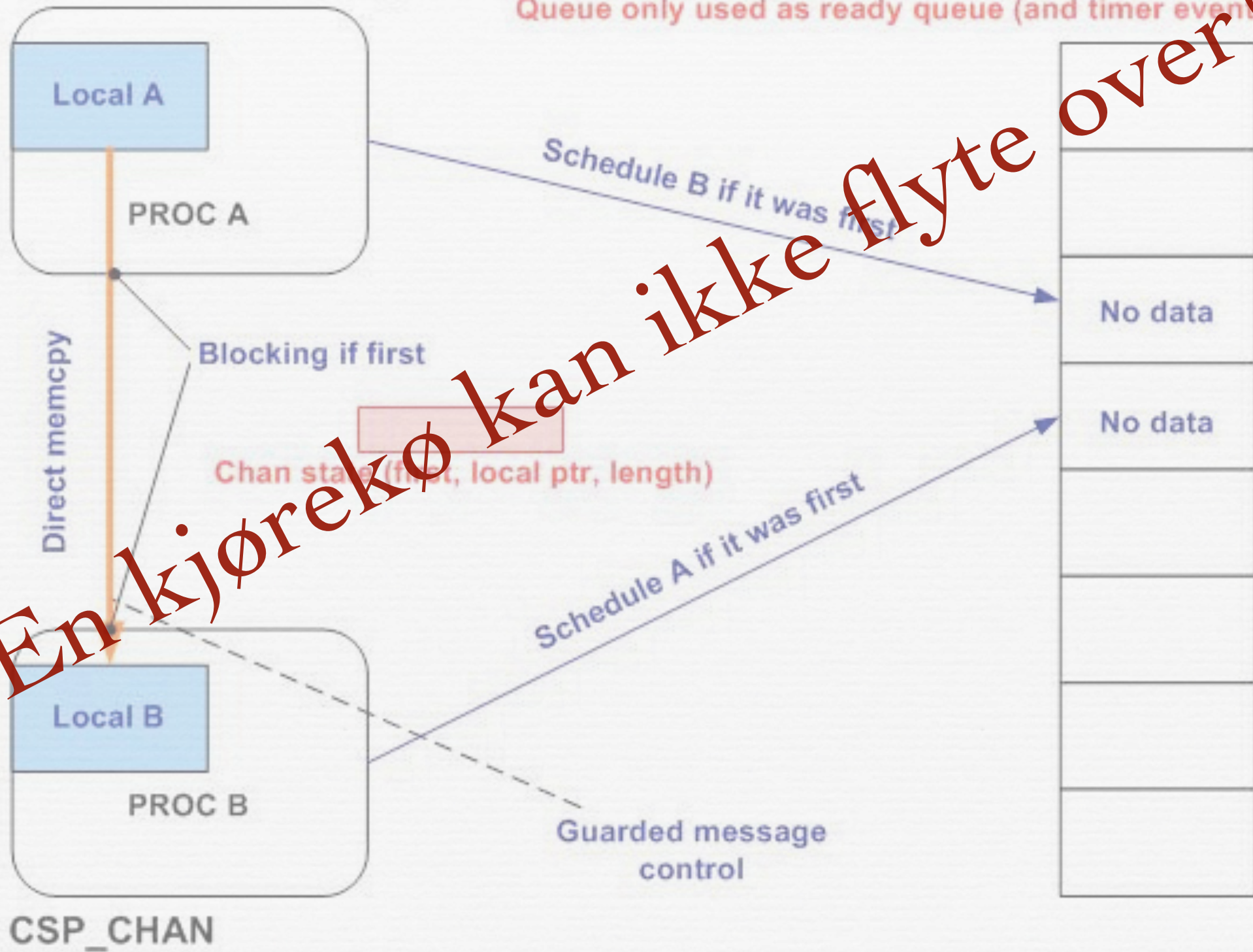
Fra meldingskø...



...til kjørekø



One-to-one
Many-to-one if channel array



Meldingskø

- ✓ Overflyt er naturlig
- ✓ Rask produsent treg konsument
- ✓ Det skal håndteres på applikasjonsnivå

Meldingsko

overflyt

- ✓ Overflyt er naturlig
Men ikke mellom programmer
- ✓ Det skal håndteres på applikasjonsnivå
Og ikke av operativsystemet
- ✓ Meldinger skal kastes bevisst
Men kanskje ikke alle
- ✓ Meldingsflyt skal håndteres bevisst
Så å holde igjen «ved halvgjort» er flott!

"Blokkerende" prosesser

- ✓ Et system som består av blokkerende kommunikasjon får ikke gjort mer enn et system som ikke blokkerer
- ✓ Heller ikke mindre (som nesten alle tror)
- ✓ Jo flere prosesser og dermed "parallell slakket" - desto mer greier et synkront system
- ✓ De fleste funksjoner og prosesser er aktive kort tid om gangen ("run to completion")
- ✓ OBS! Synkron kanaler er ikke implementert i noen busy-poll

2: ChanSched

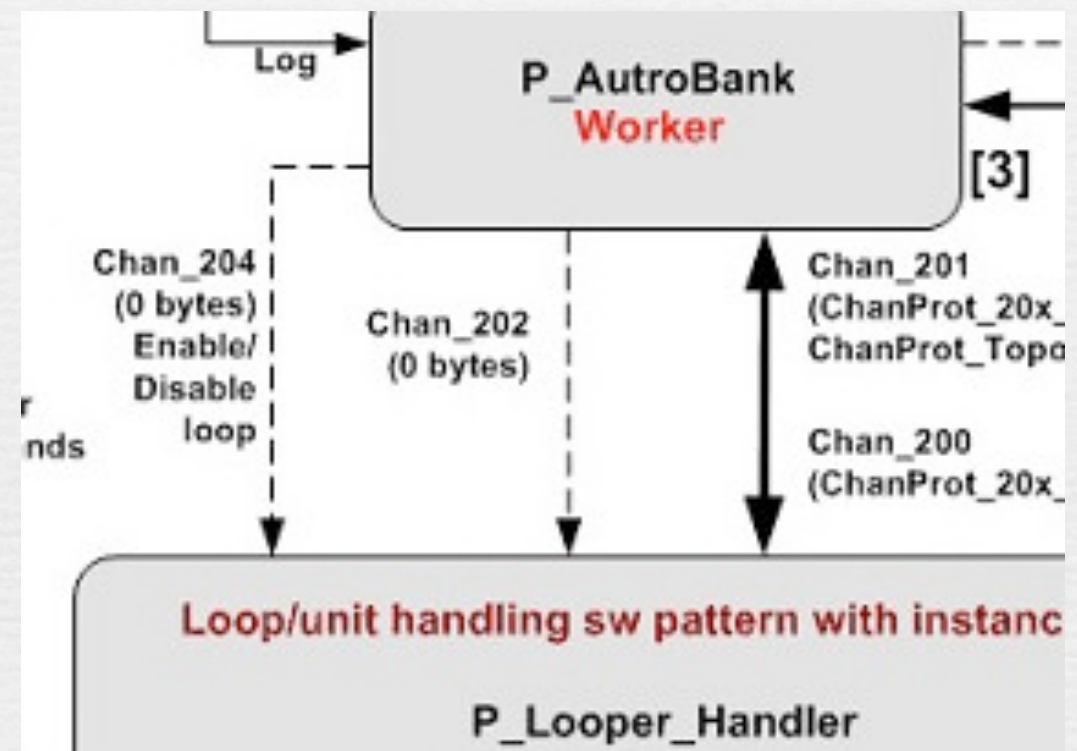
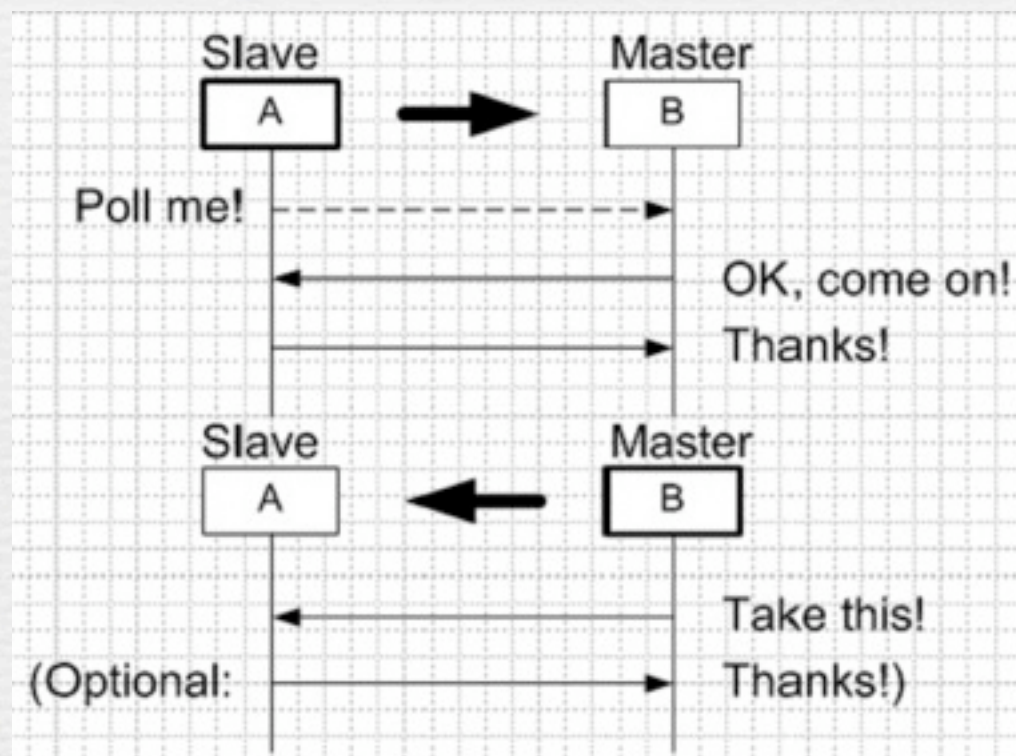
- ✓ Et system som består av blokkerende kommunikasjon får ikke gjort mer enn et system som ikke blokkerer
- ✓ Heller ikke mindre (som nesten alle tror)
- ✓ Jo flere prosesser og dermed "parallell slakkhet" - desto mer greier et synkront system
- ✓ De fleste funksjoner og prosesser er aktive kort tid om gangen ("run to completion")

ChanSched

- ✓ OBS! Synkroner kanaler er ikke implementert i noen busy-poll

Data-komm som alltid funker

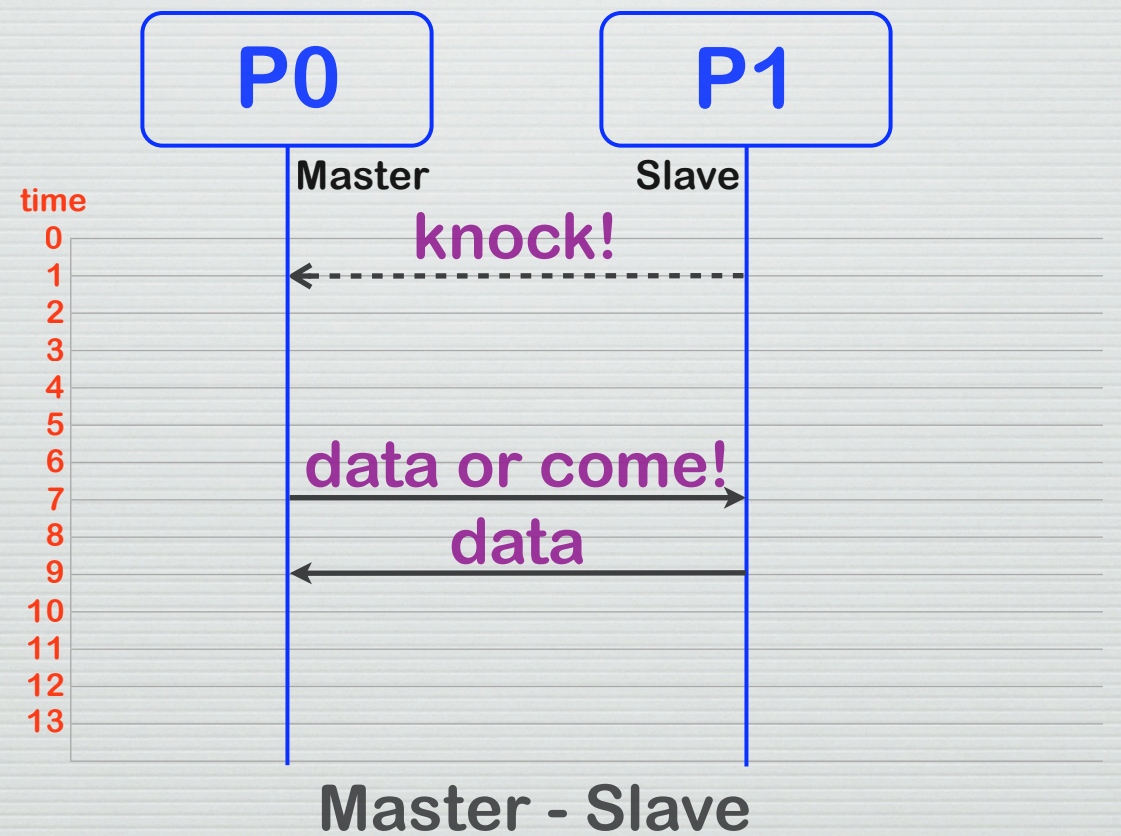
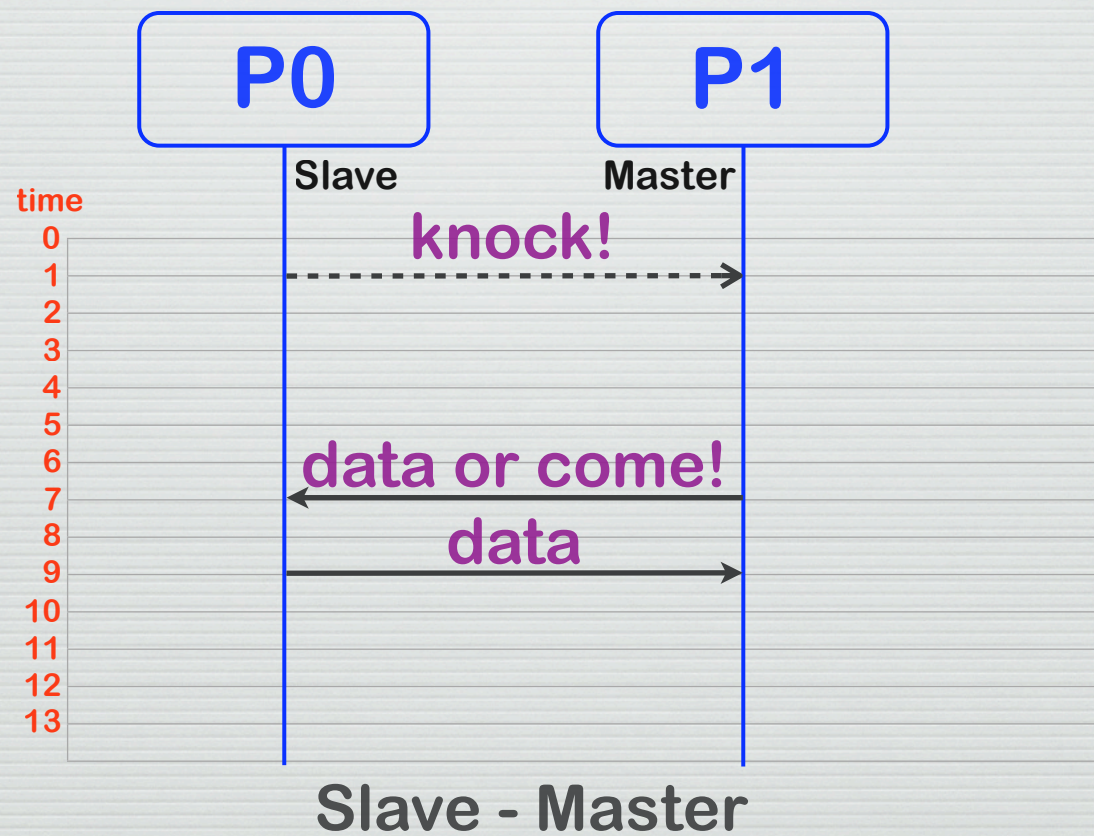
✓ Mønster som garanterer mot vranglås



✓ «Knock-come» verifisert i Promela/Spin

<http://oyvteig.blogspot.com/2009/03/009-knock-come-deadlock-free-pattern.html>

Men hvem har hvilken rolle?



Master-Slave eller omvendt?

Seriell OO modellering

- ✓ Her benytter vi UML
- ✓ Vi benytter Telelogic Rhapsody
 - ❖ Klassediagrammer
 - ❖ Tilstandsmaskiner
 - ❖ Kodegenerering av ramme og skruktur
- ✓ Håndskrevet:
 - ❖ Aksjonskoden
 - ❖ Lim mot resten av systemet
 - ❖ Prosesser via operativsystem så langt

«modellering»

- ✓ ..er nåtidas «ekspertsystem»-ord
- ✓ Men både assembler, C, Java, CSP, occam, Go - og UML er *abstraksjoner*
- ✓ Altså «modeller»
- ✓ Men i varierende grad er de ovenfor «formelle modeller» - dvs. kan være input til et verktøy som gjør verifisering for korrekthet (null til..?)
- ✓ Men disse er: CSP/FDR2, Promela/Spin, LTSA etc. Rask utvikling, følg med!

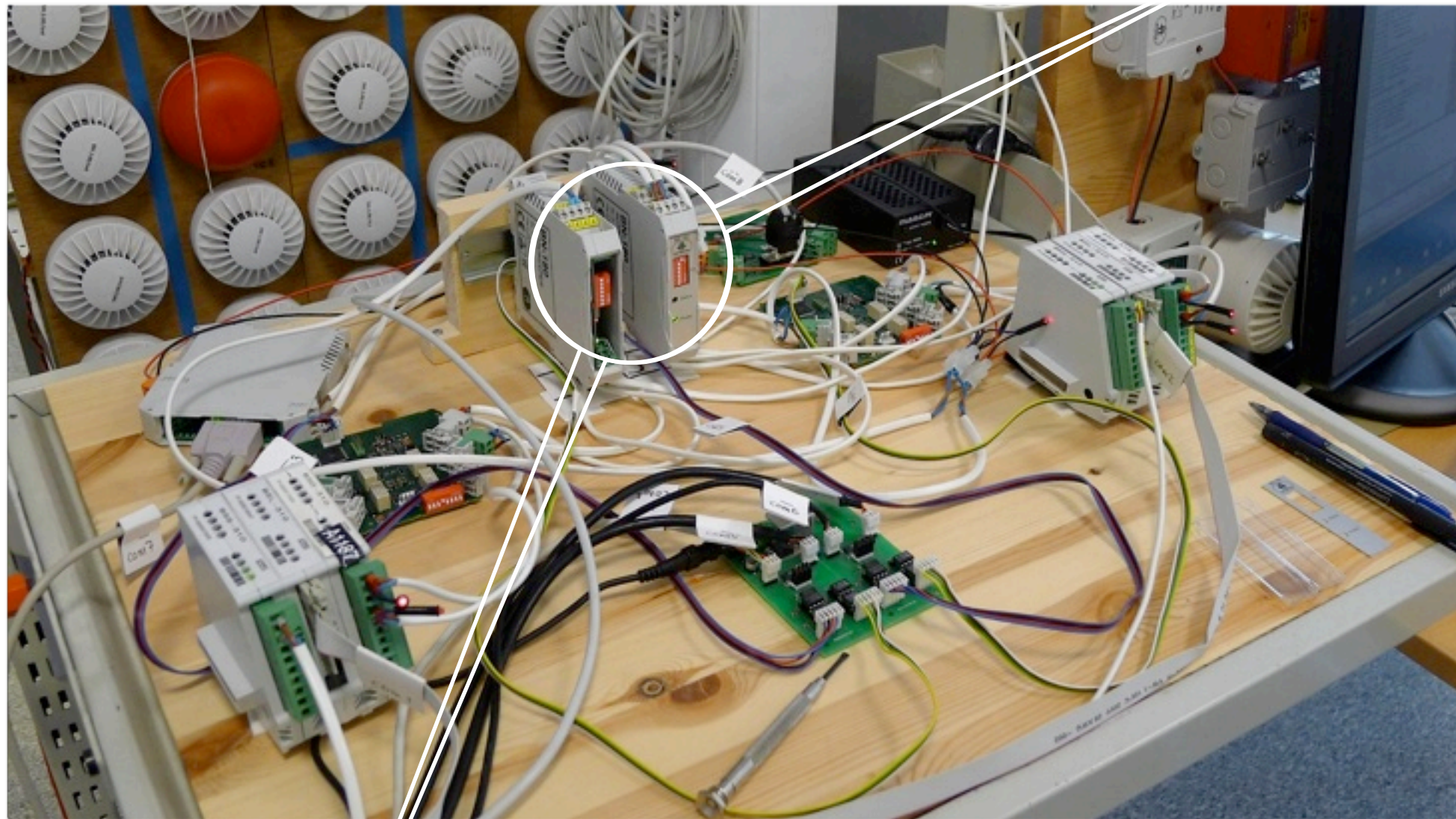
Resten av tida skal jeg vise dere
noe jeg har lært i de siste årene

New ALT for Application Timers and Synchronisation Point Scheduling

(Two excerpts from a small channel based scheduler)

«ChanSched»

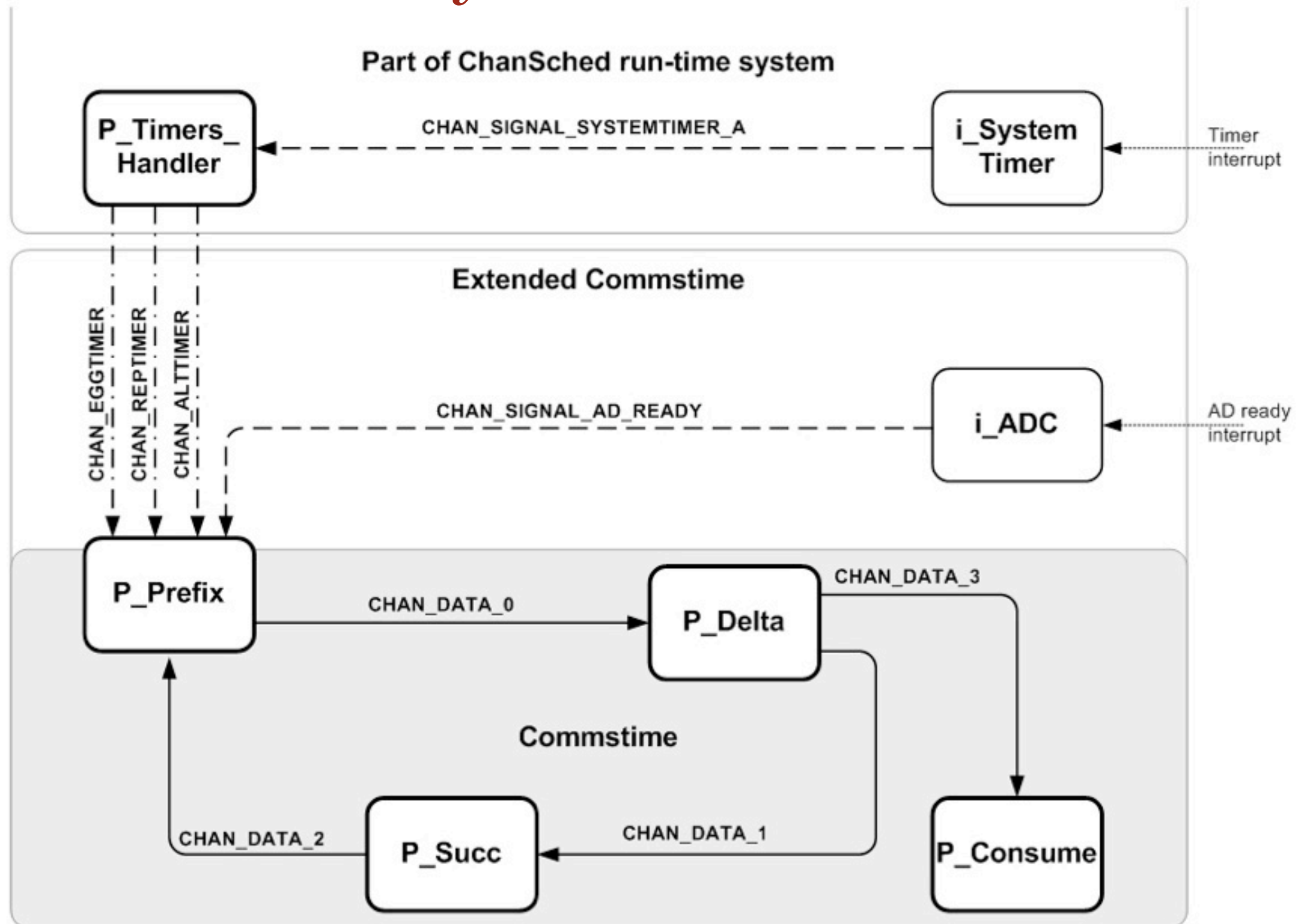
Øyvind TEIG and Per Johan VANNEBO
Autronica Fire and Security, Trondheim, Norway



At *Communicating Process Architectures 2009* (CPA-2009), 1-4 November, 2009, Eindhoven, the Netherlands
<http://www.wotug.org/cpa2009/>

AutroKeeper: patentert 329859 i Norge, søknad PCT/NO2009/000319 utenlands

Testsystem for ChanSched



Prosess/dataflyt for utvidet «commstime»

```

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;
        }
    }
}

```

«En scheduler/
kjøresystem/
operativsystem/
er er ikke så
usynlig som jeg
trodde»

```

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

```

```

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;
            }
        }
    }
}

```

```

PROC P_occam (CHAN OF INT in, out)

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

:

```

```

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;
        }
    }
}

```

Kjøresystem
oppå annet
kjøresystem

```

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

```

```

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
            }
        }
    }
}

```

Kjøresystem
alene gir
eget
prosessbegrep

```

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

```

```

PROC P_occam (CHAN OF INT in, out)

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

:

```

```

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;
        }
    }
}

```

```

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

```

```

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;
            }
        }
    }
}

```

```

PROC P_occam (CHAN OF INT in, out)

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

:

```

En typisk ChanSched prosess

```
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. }
```

"Fjernet" siden i første utgave

```
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, CP->LED_Timeout_Tick);
7.     SET_REPTIMER (CHAN_REPTIMER, ADC_TIME_TICKS);
8.     CHAN_OUT (CHAN_DATA_0, &CP->Data_0, sizeof(CP->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, &CP->Data_2, sizeof (CP->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, &CP->Data_0, sizeof (CP->Data_0));
22.        };
23.    }
24. }
```

Nesten usynlig hopp tilbake til der man sist blokkerte

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

```
64. #define SCHEDULE_AT goto

66. #define CAT(a,b,c,d,e) a##b##c##d##e // Concatenate to f.ex. "SYNCH_8_L"

68. #define SYNCH_LABEL(a,b,c,d,e) CAT(a,b,c,d,e) // Label for Proctor-table

70. #define PROC_DESCCHEDULE_AND_LABEL() \
71.     CP->LineNo = __LINE__; \
72.     return; \
73.     SYNCH_LABEL(SYNCH,_,__LINE__,_,L) :

75. #define CHAN_OUT(chan,dataptr,len) \
76.     if (ChanSched_ChanOut(chan,dataptr,len) == FALSE) \
77.     { \
78.         PROC_DESCCHEDULE_AND_LABEL(); \
79.     } \
80.     g_ThisAltTaken = FALSE

81. #define PROCTOR_PREFIX() \
82.     switch (CP->LineNo) \
83.     { \
84.         case 0: break; \
85.         case 8: SCHEDULE_AT SYNCH_8_L; \
86.         case 17: SCHEDULE_AT SYNCH_17_L; \
87.         case 21: SCHEDULE_AT SYNCH_21_L; \
88.         DEFAULT_EXIT \
89.     }
```

Bare denne er synlig, ref to sider siden

Genereres av verktøy og ligger i egen .h fil

Og dette kjører ombord på

DISNEY DREAM: Ship Photos

MarineTraffic.com

Live Map | Vessels | Ports | Gallery

World Map | Cover your Area | Frequently Asked Questions | Services



DISNEY DREAM photos:

42/62

Upload a photo

Ship Type: Passenger
Length x Breadth: 340m X 36m
Speed recorded (Max / Average): 19.2 / 12.8 knots
Flag: Bahamas [BS] Call Sign: C6YR6
MMSI: 311042900 IMO: 9434254
[Vessel's Details...](#)

Place: Papenburg
Date: 2010-11-07 16:53
© H.Trappmann
Uploaded: 2011-04-02 20:27

Place: Eemshaven (Latitude / Longitude: 53.4668 / 6.84071 - Show on Map)
Date Taken: 2010-11-13 07:49
Original Size: 1800 x 1175 pixels

Rate this photo: ★★★★★
Current Rating: 4 (23 votes)

DISNEY FANTASY

[Contribute to this page](#) [Add to My Fleet](#)

Vessel's Details

Ship Type: Passengers ship
Year Built: 2012
Length x Breadth: 340 m X 42 m
Gross Tonnage: 124000, DeadWeight: 9500 t
Speed recorded (Max / Average): 16.3 / 15 knots
Flag: Bahamas [BS] 
Call Sign: C6ZL6
IMO: 9445590, MMSI: 311058700

Last Position Received

Area: Atlantic North
Latitude / Longitude: [28.41305° / -80.6283° \(Map\)](#)
Currently in Port: [CAPE CANAVERAL](#)
Last Known Port: [CAPE CANAVERAL](#)
Info Received: 0d 0h 2min ago

DISNEY FANTASY - Vessel's Details and Current Position - 9445590 - 311058700

<http://www.marinetraffic.com/ais/shipdetails.aspx?mmsi=311058700>

DISNEY FANTASY - Vessel's Details

Vessels

Ports

Gallery

MarineTraffic.com

Search

English

Viktig beskjed

Nå kvitter vi oss med vårt gamle lager og vi trekker derfor ut besøkende som kan motta gamle lagerprodukter som gave. **Du har blitt trukket ut!** Du kan vinne en av følgende produkter:



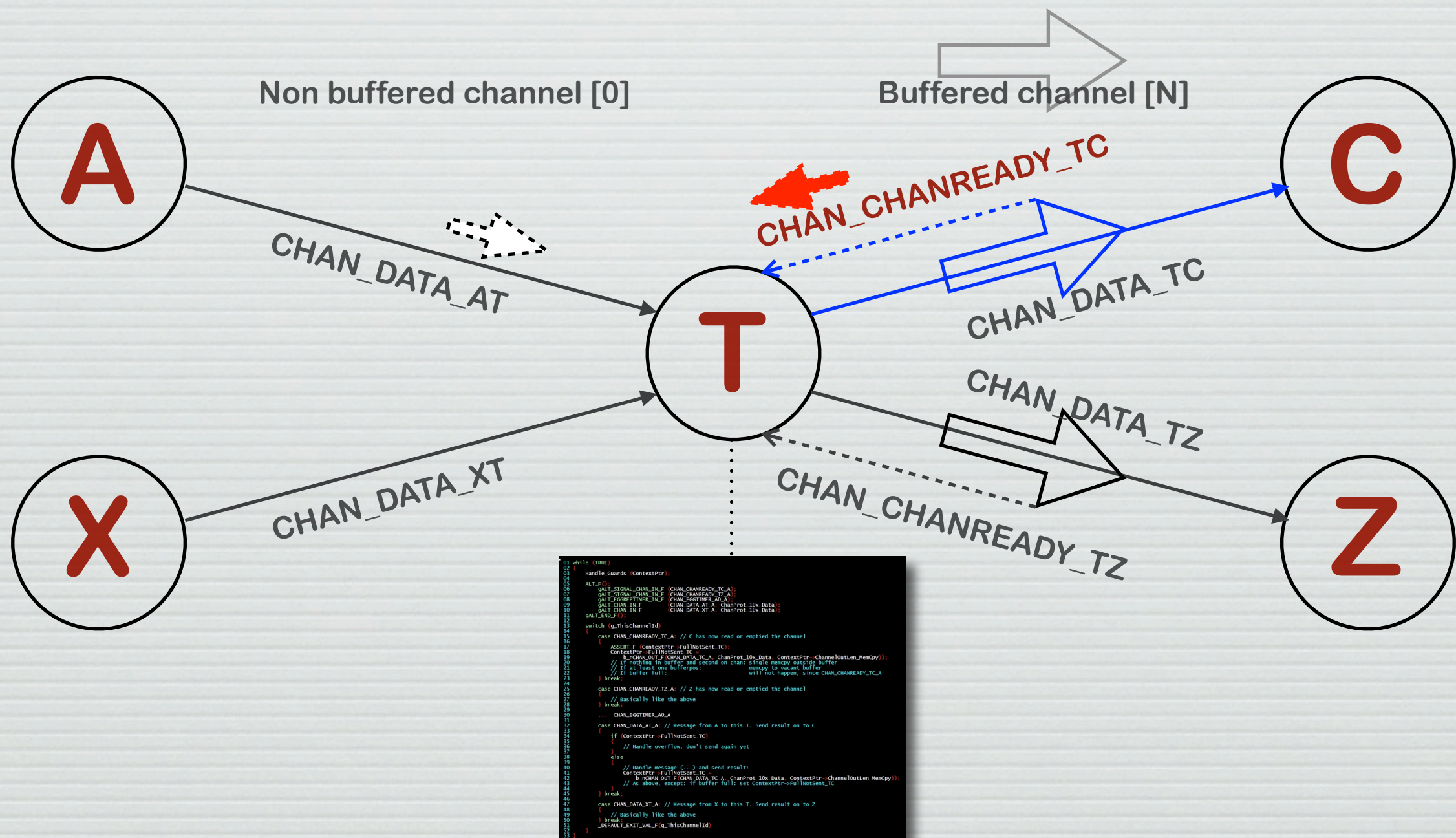
Sjekk

abonnements-
tjeneste hundre
og åtti kr hver andre uke



© Kees de Vries
MarineTraffic.com

ChanSched BufChanSched?



<http://oyvteig.blogspot.com/2011/12/034-output-guard-vs-channel-ready.html>

```

01 while (TRUE)
02 {
03     Handle_Guards (ContextPtr);
04
05     ALT_F();
06     gALT_SIGNAL_CHAN_IN_F (CHAN_CHANREADY_TC_A);
07     gALT_SIGNAL_CHAN_IN_F (CHAN_CHANREADY_TZ_A);
08     gALT_EGGREPTIMER_IN_F (CHAN_EGGTIMER_A0_A);
09     gALT_CHAN_IN_F (CHAN_DATA_AT_A, ChanProt_10x_Data);
10     gALT_CHAN_IN_F (CHAN_DATA_XT_A, ChanProt_10x_Data);
11     gALT_END_F();
12
13     switch (g_ThisChannelId)
14     {
15     case CHAN_CHANREADY_TC_A: // C has now read or emptied the channel
16     {
17         ASSERT_F (ContextPtr->FullNotSent_TC);
18         ContextPtr->FullNotSent_TC =
19             b_nCHAN_OUT_F(CHAN_DATA_TC_A, ChanProt_10x_Data, ContextPtr->ChannelOutLen_MemCpy));
20         // If nothing in buffer and second on chan: single memcpy outside buffer
21         // If at least one bufferpos:                memcpy to vacant buffer
22         // If buffer full:                            will not happen, since CHAN_CHANREADY_TC_A
23     } break;
24
25     case CHAN_CHANREADY_TZ_A: // Z has now read or emptied the channel
26     {
27         // Basically like the above
28     } break;
29
30     ... CHAN_EGGTIMER_A0_A
31
32     case CHAN_DATA_AT_A: // Message from A to this T. Send result on to C
33     {
34         if (ContextPtr->FullNotSent_TC)
35         {
36             // Handle overflow, don't send again yet
37         }
38         else
39         {
40             // Handle message (...) and send result:
41             ContextPtr->FullNotSent_TC =
42                 b_nCHAN_OUT_F(CHAN_DATA_TC_A, ChanProt_10x_Data, ContextPtr->ChannelOutLen_MemCpy));
43             // As above, except: if buffer full: set ContextPtr->FullNotSent_TC
44         }
45     } break;
46
47     case CHAN_DATA_XT_A: // Message from X to this T. Send result on to Z
48     {
49         // Basically like the above
50     } break;
51     _DEFAULT_EXIT_VAL_F(g_ThisChannelId)
52 }
53 }

```

```
ALT_F();  
→ gALT_SIGNAL_CHAN_IN_F {CHAN_CHANREADY_TC_A};  
  gALT_SIGNAL_CHAN_IN_F {CHAN_CHANREADY_TZ_A};  
  gALT_EGGREPTIMER_IN_F {CHAN_EGGTIMER_A0_A};  
→ gALT_CHAN_IN_F {CHAN_DATA_AT_A, ChanProt_10x_Data};  
  gALT_CHAN_IN_F {CHAN_DATA_XT_A, ChanProt_10x_Data};  
gALT_END_F();
```



```
case CHAN_CHANREADY_TC_A: // C has now read or emptied the channel
{
    ASSERT_F (ContextPtr->FullNotSent_TC);
    ContextPtr->FullNotSent_TC =
        b_nCHAN_OUT_F(CHAN_DATA_TC_A, ChanProt_10x_Data, ContextPtr->ChannelOutLen_MemCpy));
    // If nothing in buffer and second on chan: single memcpy outside buffer
    // If at least one bufferpos:          memcpy to vacant buffer
    // If buffer full:                    will not happen, since CHAN_CHANREADY_TC_A
} break;
```

Overflyt håndteres på applikasjonsnivå!



```
case CHAN_DATA_AT_A: // Message from A to this T. Send result on to C
{
    if (ContextPtr->FullNotSent_TC)
    {
        // Handle overflow, don't send again yet
    }
    else
    {
        // Handle message (...) and send result:
        ContextPtr->FullNotSent_TC =
            b_nCHAN_OUT_F(CHAN_DATA_TC_A, ChanProt_10x_Data, ContextPtr->ChannelOutLen_MemCpy));
        // As above, except: if buffer full: set ContextPtr->FullNotSent_TC
    }
} break;
```

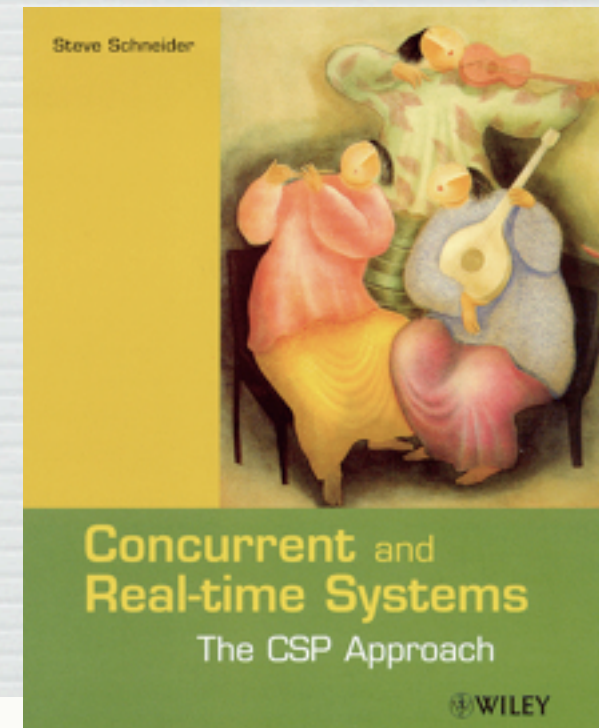
Example 7.4.1 (Buffers) The property of being a buffer of type T is expressible in a process-oriented way, by means of a mutual recursion. Internal choice is used to describe the various possibilities:

Schneider hadde funnet opp dette før meg (og "noe" i unix(?)):

$$\begin{aligned} NBUFF_T(\langle \rangle) &= in?x : T \rightarrow NBUFF_T(\langle x \rangle) \\ NBUFF_T(s \frown \langle y \rangle) &= out!y \rightarrow NBUFF_T(s) \\ &\quad \square (STOP \sqcap in?x : T \rightarrow NBUFF_T(\langle x \rangle \frown s \frown \langle y \rangle)) \end{aligned}$$

The parameter to $NBUFF$ consists of the sequence of messages that the buffer currently contains. If this sequence is the empty sequence $\langle \rangle$, then the buffer is empty and must be ready for input. If the sequence contains some messages, then the buffer must be ready to output the next message required. It is also possible that it will accept further input, but it does not have to. These possibilities are represented by the internal choice between $STOP$ and a further input.

Concurrent and Real-time Systems. The CSP Approach. Steve Schneider



PROCESS-ORIENTED SPECIFICATION **211**

The buffer specification $NBUFF_T = NBUFF_T(\langle \rangle)$ also specifies that the alphabet of the buffer is restricted to its input and output channels. It encapsulates the buffer specification given on page 196:

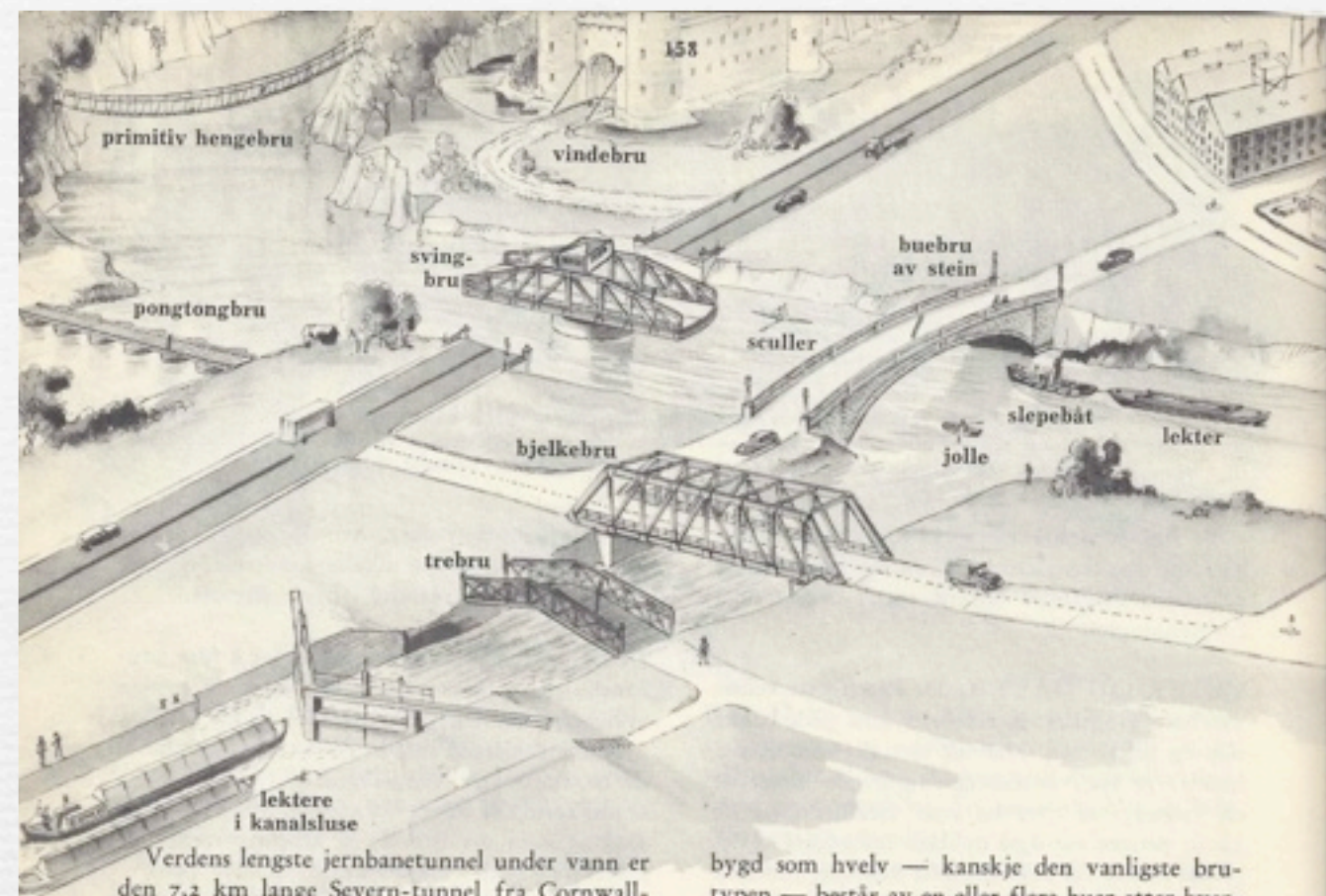
$$NBUFF \sqsubseteq_{SF} IMP \Leftrightarrow IMP \text{ sat } (Buff_T(tr), Buff_{SF}(tr, X))$$

If further events are to be possible (such as a channel which can report on whether or not the buffer is empty), then the appropriate specification will be $NBUFF_T ||| CHAOS_A$, where A are the other possible events. The most general specification, which corresponds to $Buff_T(tr, X)$, is given as follows:

BufChanSched som vi ikke har:

- ✓ Sikker bufret kanal
- ✓ Kanal-klar kanal
- ✓ Synkronisering på kallnivå
 - ★ Send / motta hvor som helst
 - ★ **SPoC**: funksjon->prosess
- ✓ PAR i = 0 for N (instanser)

SPoC = Southampton Portable occam Compiler (occam -> C) som vi brukte i flere produkt på nittitallet



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 den bredeste tunnel som noen gang er bygd — er Merseytunnelen, mellom Liverpool og Birkenhead. Den er 3,4 km lang, 13,5 meter i diameter og har fire kjørebane, to i hver retning — en for langsom og en for rask trafikk.

BRUER. En bygger bruer når en vil føre, en vei eller en jernbanelinje over en elv, et sund, en kanal, en dal eller over en annen vei eller jernbanelinje.

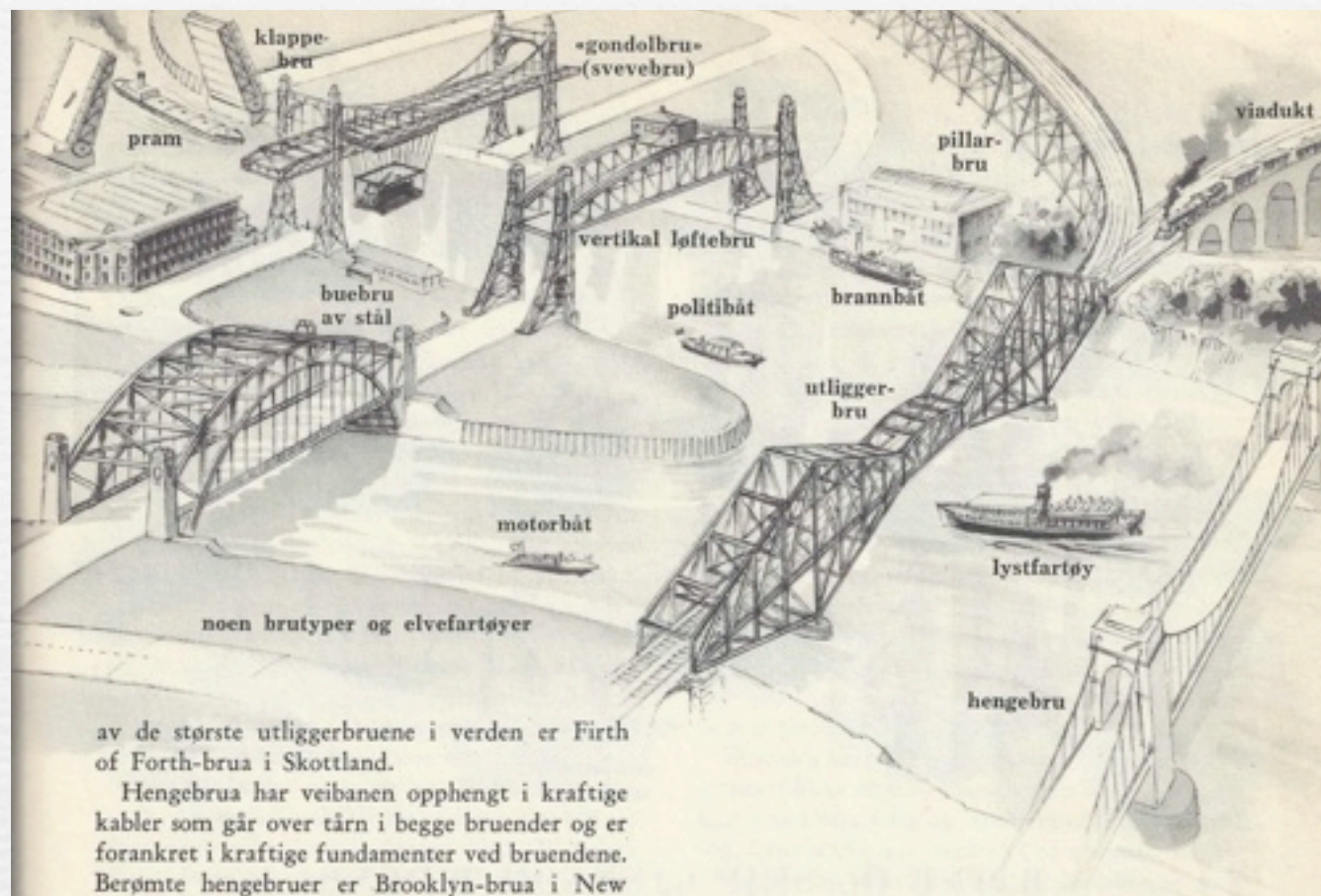
Hvilken brutype en velger, avhenger av hvor lang brua må være. Bjelkebrua brukes bare når det gjelder et nokså kort spenn. Bruer som er

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 byggemateriale. Mange av de store moderne buebruene er bygd av stål. Den største i verden er brua over havnebassenget i Sydney i Australia. En annen vakker buebru av stål går over Sambesifloden i Afrika, og så tett inntil Victoria-fallene at togene som kjører over, ofte blir oversprøytet av dusjen fra fossen.

Når det gjelder svære spenn, bruker en ofte den såkalte utliggerbru. Den hviler på en rekke pilarer, og de enkelte bruspenn rager dels ut over pilarene, dels kan de være opphengt mellom de utstikkende brudelene. Stabiliteten av slike bruer er basert på en viktig tilpassing i lengde og vekt mellom utliggerne og de innhengte spennene. En

i kanalslusen slippes vannet inn så vannspeilet stiger og løfter lekteren, eller det slippes ut så lekteren senkes og kan gå nedover til lavere nivå

håndtak til å åpne og lukke sluseportene med



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 bruker gjerne hengebruer når en skal føre en vei over svære vannflater, hvor det er umulig å bygge brupilarer ute i vannet. De moderne hengebruene av stål og betong har ikke stor likhet med de enkle hengebruene av tau eller flettede grener, slik en kan se dem ennå i verdens fjerne avkroker.

Over vann der det er skipstrafikk, må vi — hvis det er umulig å bygge et høyt bruspenn — bruke en bevegelig bru. Tower Bridge i London er en bru av denne typen. Den er en klaffebu som folder veibanen opp som to klaffer, akkurat som bladene på en foldekniv, når skip skal passere under. Svingbruene er vribare om en tapp som hviler på et fundament midt under brua. Løftebrua er en sjeldnere type. Den kan heves loddrett til værs. Enda sjeldnere er typen med en hengende plattform under en høytliggende bærekonstruksjon. Her transporteres folk og kjøretøyer fram og tilbake i en slags gondol. Den er festet til en tralle som løper på skinner oppe på bærebjelkene.

Pongtongbuer består av en veibane som er lagt over flytende metalltanker. (Før brukte man båter.) Denne brutypen brukes ofte i krig, fordi den fort kan slås over en elv og like fort rives.

KANALER OG SEILBARE ELVER.

Før det ble bygd veier, foregikk reisene over land helst på elvene, og da var vassdragene de viktigste forbindelsesveiene. I våre dager spiller slike seilbare elver fremdeles stor rolle, særlig for godstrafikken, og mange havner ligger ved munningen av store elver, f. eks. Fredrikstad, Göteborg, Hamburg, London og New York.

Innlandstransport til vanns har kunnet økes atskillig mange steder i verden ved at det er blitt bygd kanaler. Noen kanaler har samme nivå hele strekningen, mens andre klatrer opp og ned. Fartøyene blir hevet og senket ved hjelp av sluser.

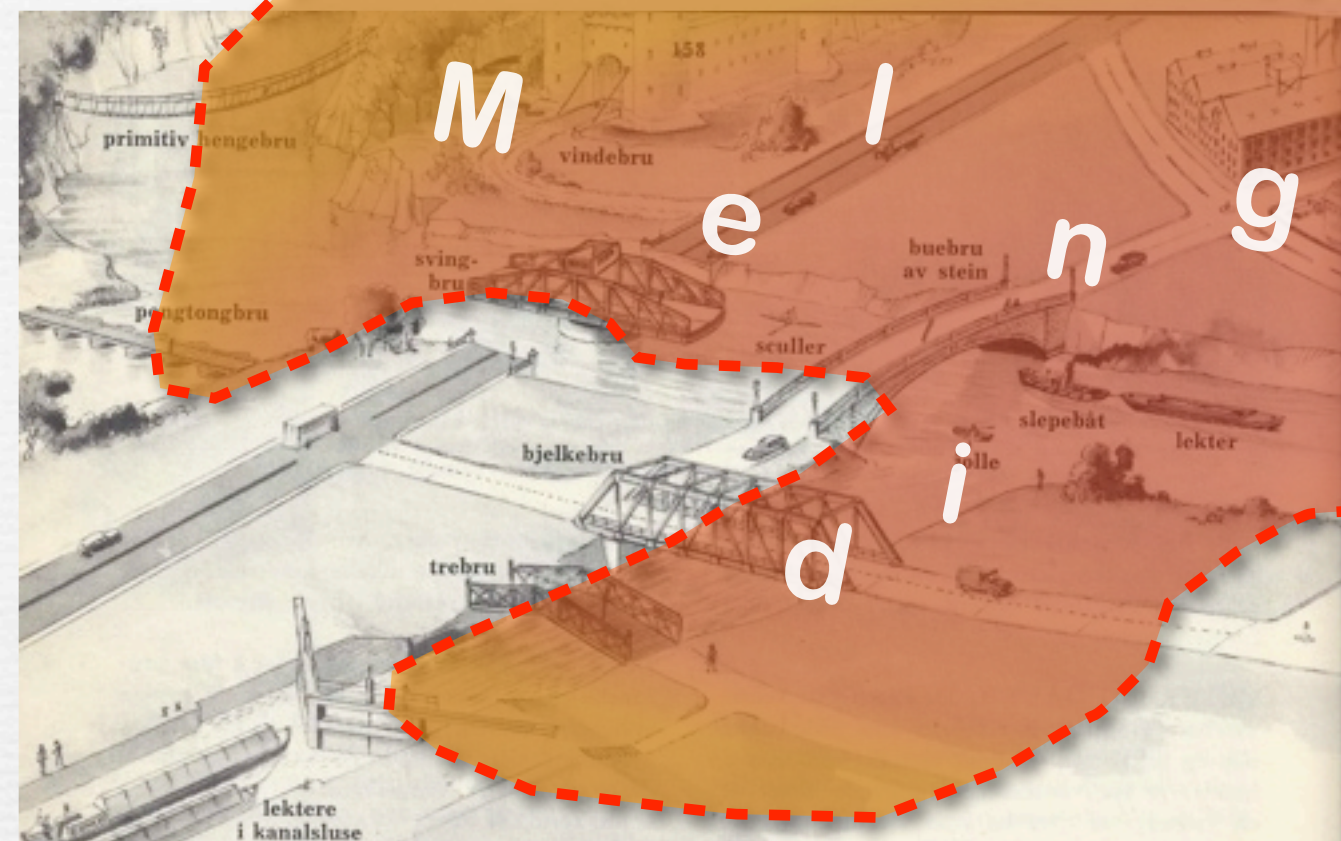
En sluse er et kort stykke av kanalen som har vanntette porter i begge ender. En båt som er på vei oppover en kanal, og som altså skal heves til et høyere nivå, går inn i slusen, og så stenges portene. Deretter åpnes sluselukene i den øvre porten, og vannet strømmer inn fra nivået som ligger høyere. Nå løftes båten av det stigende vannet til nivået er det samme som i slusen ovenfor. Så åpnes den øvre porten, og båten kan seile videre. Når en båt skal senkes, slippes vannet ut av slusen. Kanalene må ofte mudres opp så de er dype nok for fartøyene.

"Verden omkring oss", 1955 ("Odham's Encyclopedia for Children")

Miljø

Bygging

id



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 den bredeste tunnel som noen gang er bygd — er Merseytunnelen, mellom Liverpool og Birkenhead. Den er 3,4 km lang, 13,5 meter i diameter og har fire kjørebane, to i hver retning — en for langsom og en for rask trafikk.

BRUER. En bygger bruer når en vil føre, en vei eller en jernbanelinje over en elv, et sund, en kanal, en dal eller over en annen vei eller jernbanelinje.

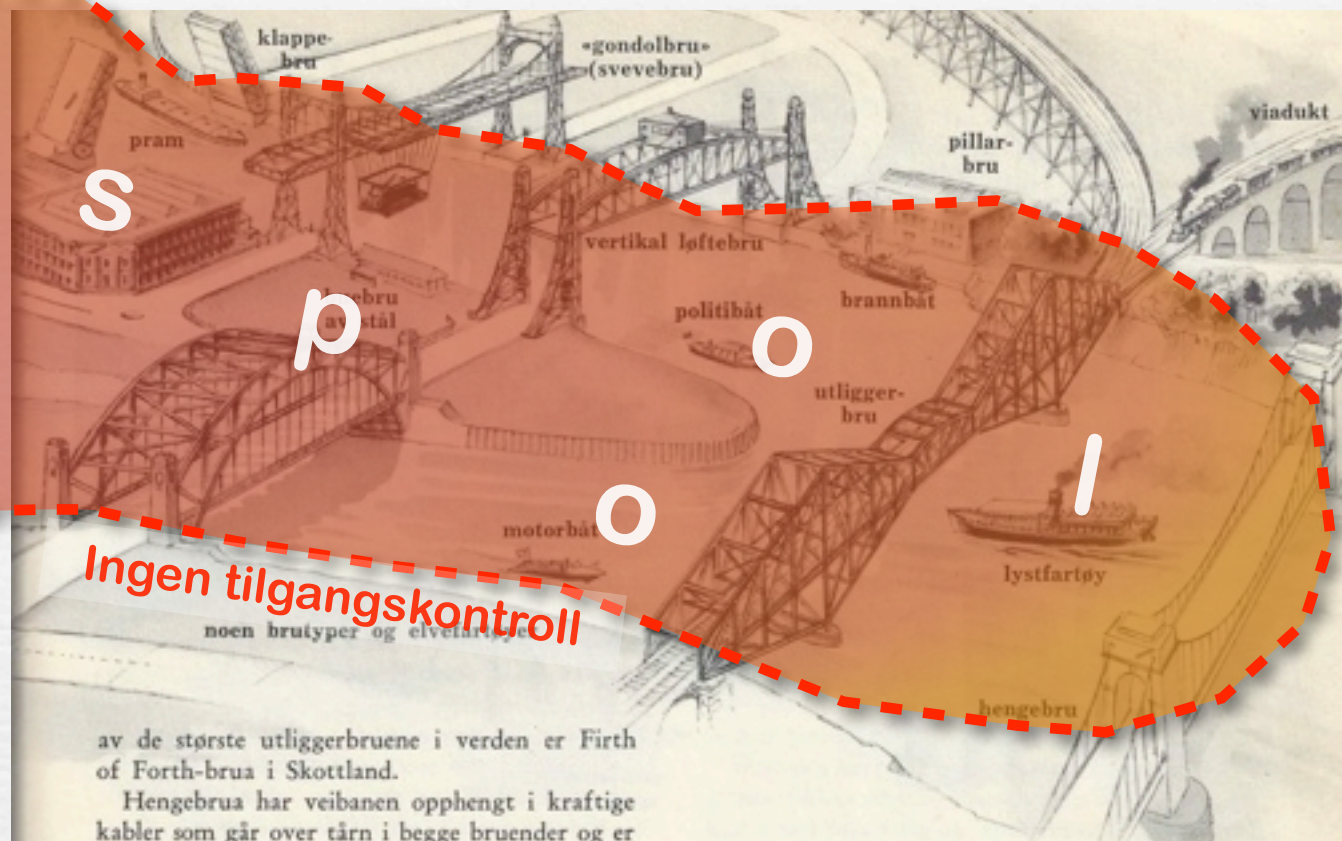
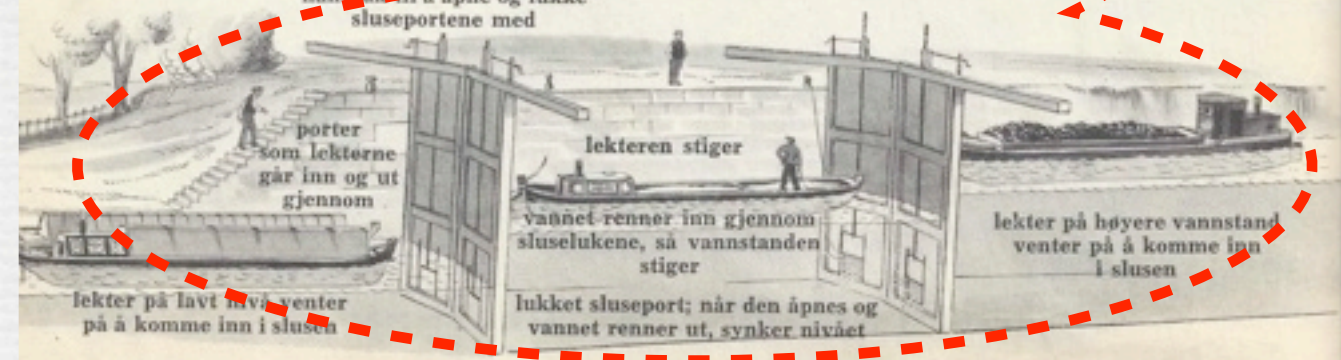
Hvilken brutype en velger, avhenger av hvor lang brua må være. Bjelkebrua brukes bare når det gjelder et nokså kort spenn. Bruer som er

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 byggemateriale. Mange av de store moderne buebruene er bygd av stål. Den største i verden er brua over havnebassenget i Sydney i Australia. En annen vakker buebru av stål går over Sambesifloden i Afrika, og så tett inntil Victoria-fallene at togene som kjører over, ofte blir oversprøytet av dusjen fra fossen.

Når det gjelder svære spenn, bruker en ofte den såkalte utliggerbru. Den hviler på en rekke pilarer, og de enkelte bruspenn rager dels ut over pilarene, dels kan de være opphengt mellom de utstikkende brudelene. Stabiliteten av slike bruer er basert på en viktig tilpassing i lengde og vekt mellom utliggerne og de innhengte spennene. En

i kanalslusen slippes vannet inn så vannspeilet stiger og løfter lekteren, eller det slippes ut så lekteren senkes og kan gå nedover til lavere nivå

håndtak til å åpne og lukke sluseportene med



Ingen tilgangskontroll

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 bruker gjerne hengebruer når en skal føre en vei over svære vannflater, hvor det er umulig å bygge brupilarer ute i vannet. De moderne hengebruene av stål og betong har ikke stor likhet med de enkle hengebruene av tau eller flettede grener, slik en kan se dem ennå i verdens fjerne avkroker.

Over vann der det er biltrafikk, må vi — hvis det er umulig å bygge et høyt bruspenn — bruke en bevegelig bru. Tower Bridge i London er en bru av denne typen. Den er en klaffebri som folder veibanen opp som to klaffer, akkurat som bladene på en foldekniv, når skip skal passere under. Svingbruene er vribare om en tapp som hviler på et fundament midt under brua. Løftebrua er en sjeldnere type. Den kan heves loddrett til værs. Enda sjeldnere er typen med en hengende plattform under en høytliggende bærekonstruksjon. Her transporteres folk og kjøretøyer fram og tilbake i en slags gondol. Den er festet til en tralle som løper på skinner oppe på bærebjelkene.

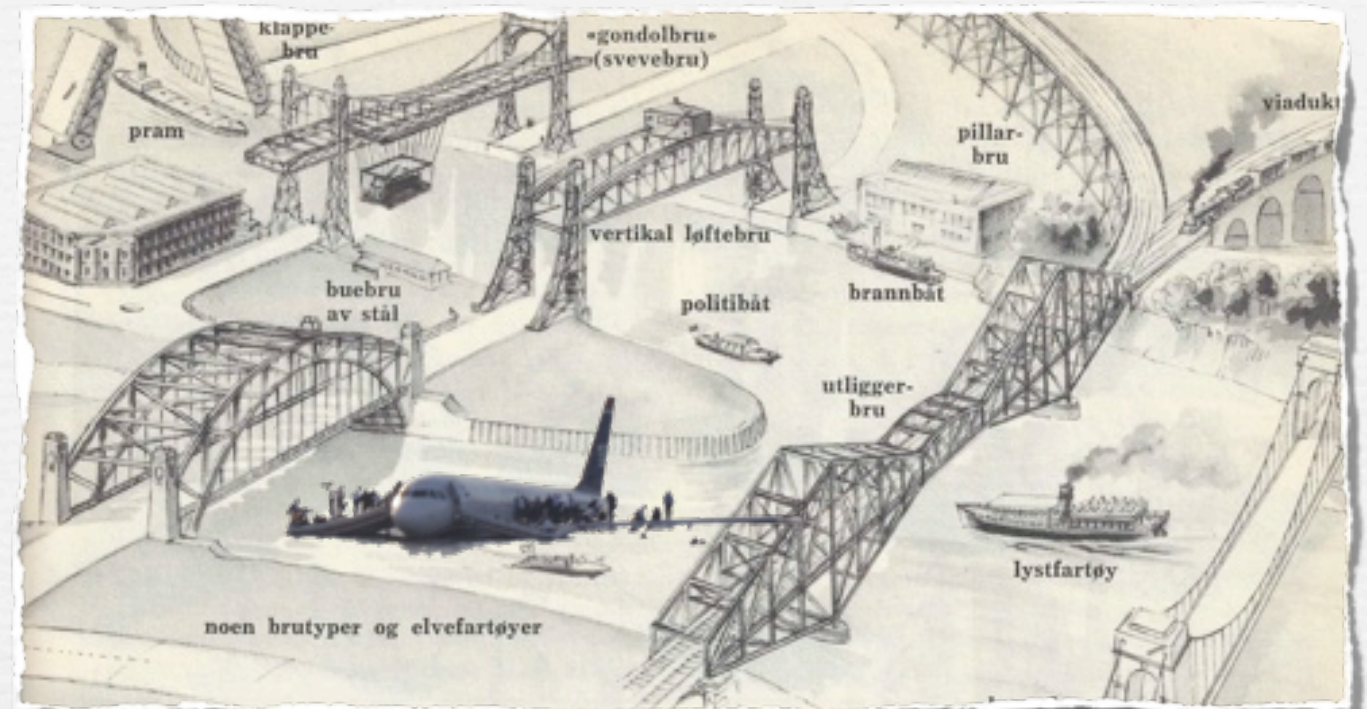
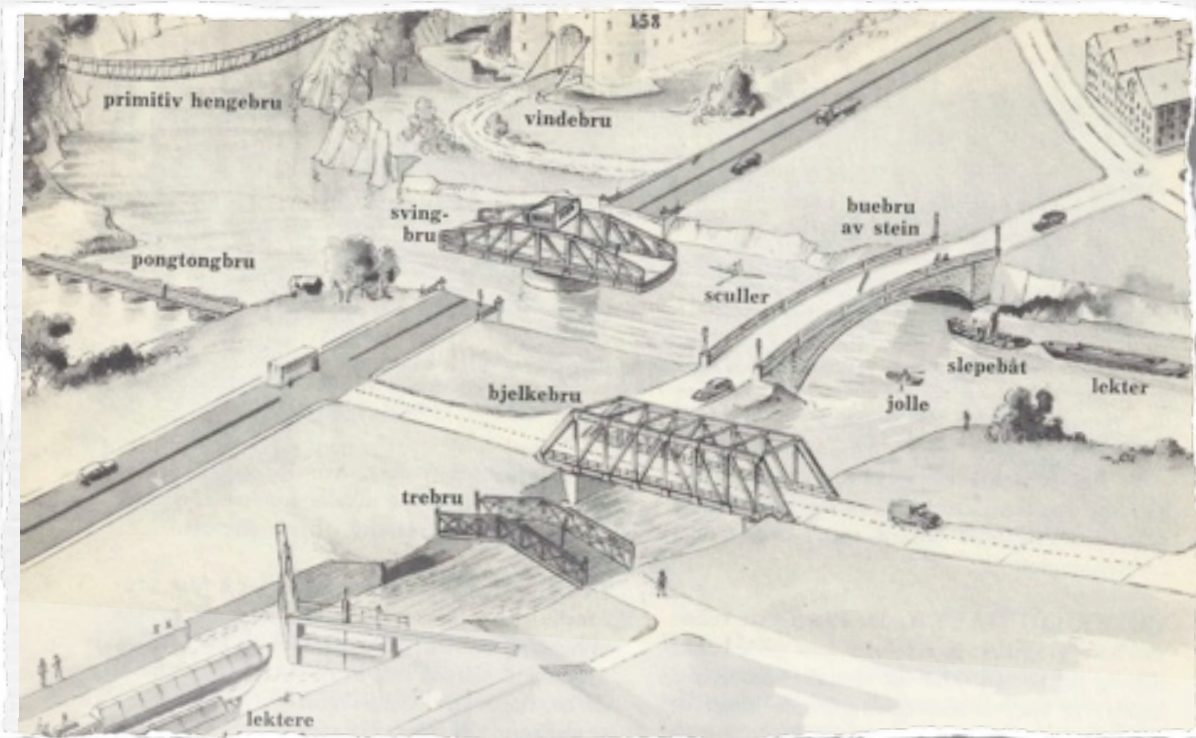
Pongtongbruer består av en veibane som er lagt over flytende metalltanker. (Før brukte man båter.) Denne brutypen brukes ofte i krig, fordi den fort kan slås over en elv og like fort rives.

KANALER OG SEILBARE ELVER.

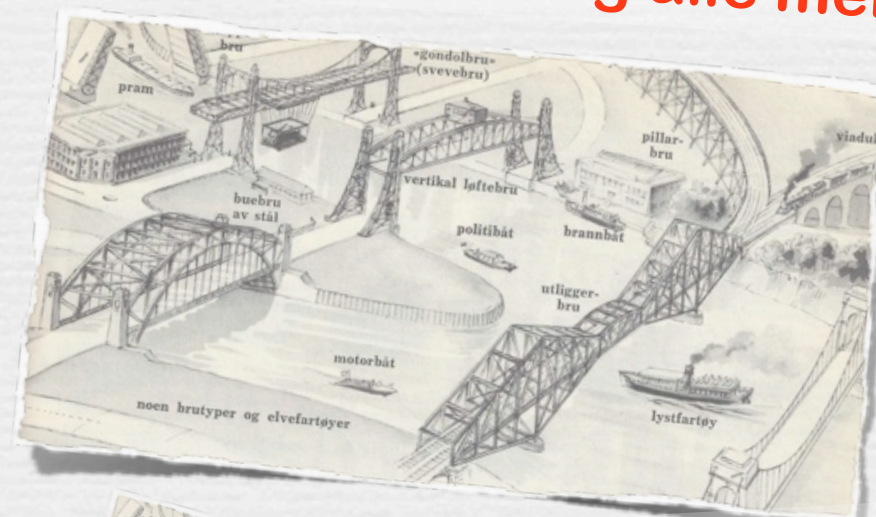
Før det ble bygd veier, foregikk reisene over land helst på elvene, og da var vassdragene de viktigste forbindelsesveiene. I våre dager spiller slike seilbare elver fremdeles stor rolle, særlig for godstrafikken, og mange havner ligger ved munningen av store elver, f. eks. Fredrikstad, Göteborg, Hamburg, London og New York.

Innlandstransport til vanns har kunnet økes atskillig mange steder i verden ved at det er blitt bygd kanaler. Noen kanaler har samme nivå hele strekningen, mens andre klatrer opp og ned. Fartøyene blir hevet og senket ved hjelp av sluser.

En sluse er et kort stykke av kanalen som har vanntette porter i begge ender. En båt som er på vei oppover en kanal, og som altså skal heves til et høyere nivå, går inn i slusen, og så stenges portene. Deretter åpnes sluselukene i den øvre porten, og vannet strømmer inn fra nivået som ligger høyere. Nå løftes båten av det stigende vannet til nivået er det samme som i slusen ovenfor. Så åpnes den øvre porten, og båten kan seile videre. Når en båt skal senkes, slippes vannet ut av slusen. Kanalene må ofte mudres opp så de er dype nok for fartøyene.



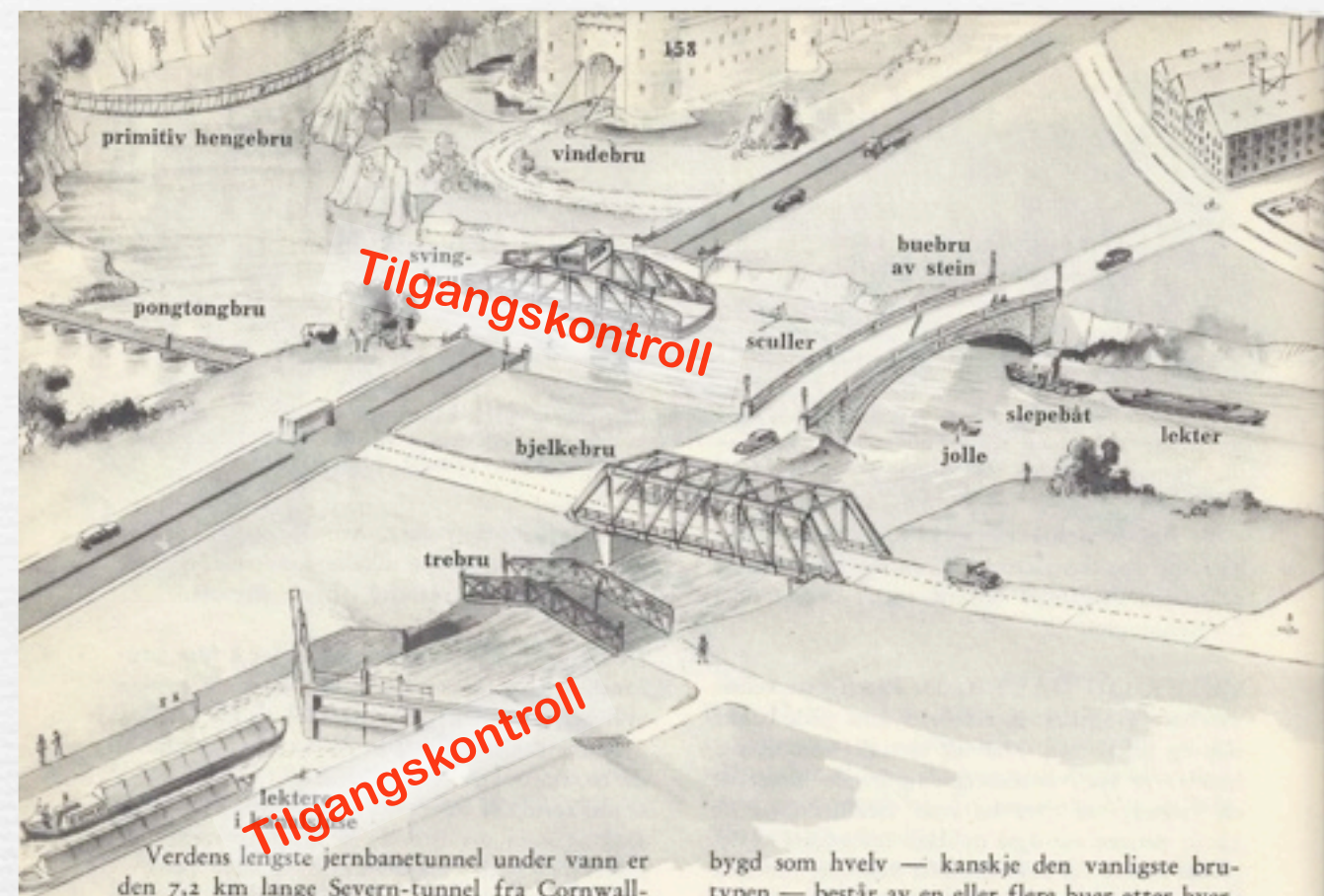
Om det sendes en eneste melding for mye så kræsjer systemet og alle meldingene mistes...



..men da øker man meldingspool-størrelse ved neste versjon til «litt mer»

..for å analysere en slik situasjon er nesten umulig

(polemikken viser et viktig punkt!)



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 den bredeste tunnel som noen gang er bygd — er Merseytunnelen, mellom Liverpool og Birkenhead. Den er 3,4 km lang, 13,5 meter i diameter og har fire kjørebane, to i hver retning — en for langsom og en for rask trafikk.

BRUER. En bygger bruer når en vil føre, en vei eller en jernbanelinje over en elv, et sund, en kanal, en dal eller over en annen vei eller jernbanelinje.

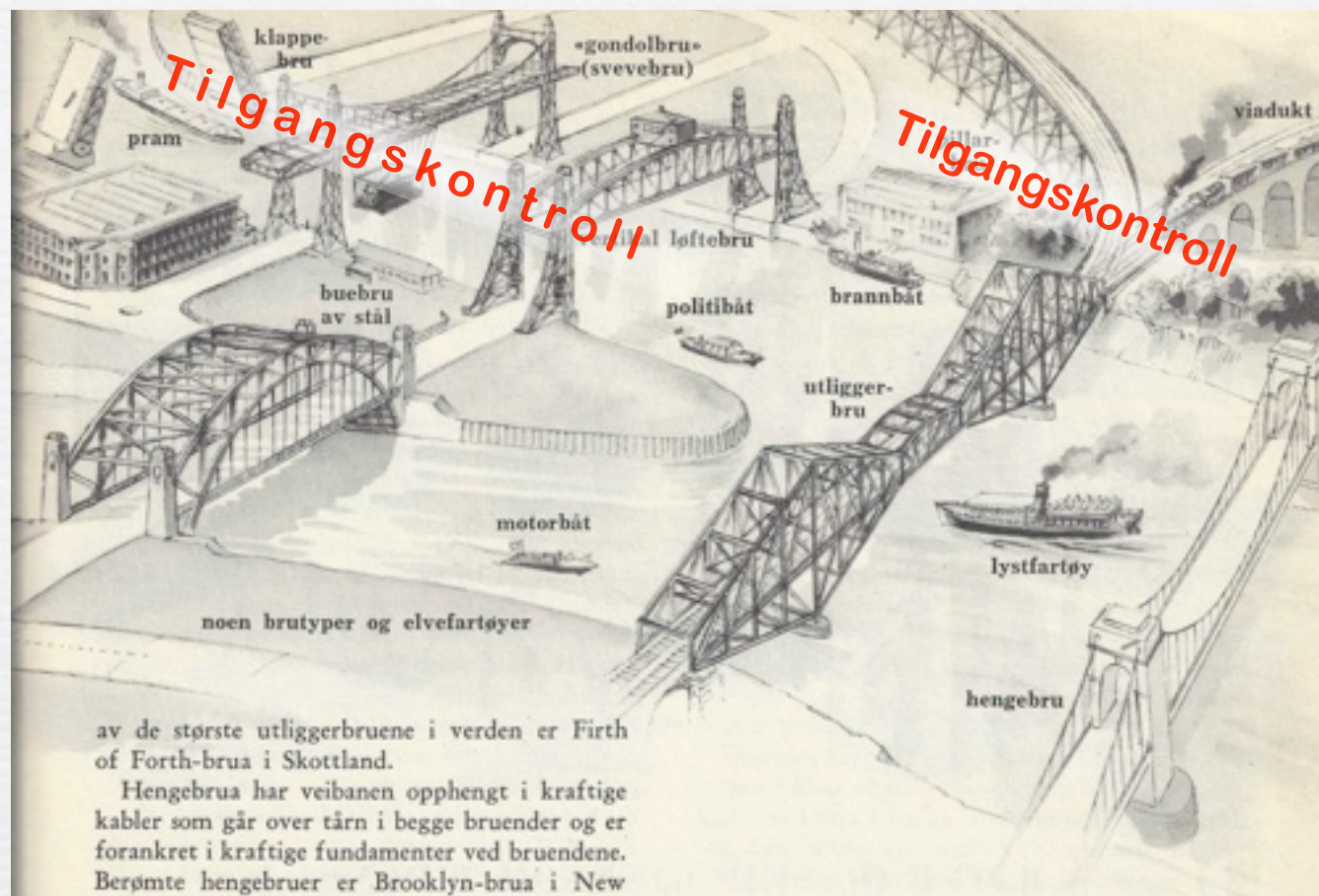
Hvilken brutype en velger, avhenger av hvor lang brua må være. Bjelkebrua brukes bare når det gjelder et nokså kort spenn. Bruer som er

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 byggemateriale. Mange av de store moderne buebruene er bygd av stål. Den største i verden er brua over havnebassenget i Sydney i Australia. En annen vakker buebru av stål går over Sambesifloden i Afrika, og så tett inntil Victoria-fallene at togene som kjører over, ofte blir oversprøytet av dusjen fra fossen.

Når det gjelder svære spenn, bruker en ofte den såkalte utliggerbru. Den hviler på en rekke pilarer, og de enkelte bruspenn rager dels ut over pilarene, dels kan de være opphengt mellom de utstikkende brudelene. Stabiliteten av slike bruer er basert på en viktig tilpassing i lengde og vekt mellom utliggerne og de innhengte spennene. En

i kanalslusen slippes vannet inn så vannspeilet stiger og løfter lekteren, eller det slippes ut så lekteren senkes og kan gå nedover til lavere nivå

håndtak til å åpne og lukke sluseportene



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 bruker gjerne hengebruer når en skal føre en vei over svære vannflater, hvor det er umulig å bygge brupilarer ute i vannet. De moderne hengebruene av stål og betong har ikke stor likhet med de enkle hengebruene av tau eller flettede grener, slik en kan se dem ennå i verdens fjerne avkroker.

Over vann der det er skipstrafikk, må vi — hvis det er umulig å bygge et høyt bruspenn — bruke en bevegelig bru. Tower Bridge i London er en bru av denne typen. Den er en klaffebu som folder veibanen opp som to klaffer, akkurat som bladene på en foldekniv, når skip skal passere under. Svingbruene er vribare om en tapp som hviler på et fundament midt under brua. Løftebrua er en sjeldnere type. Den kan heves loddrett til vørs. Enda sjeldnere er typen med en hengende plattform under en høytliggende bærekonstruksjon. Her transporteres folk og kjøretøyer fram og tilbake i en slags gondol. Den er festet til en tralle som løper på skinner oppe på bærebjelkene.

Pongtongbruer består av en veibane som er lagt over flytende metalltanker. (Før brukte man båter.) Denne brutypen brukes ofte i krig, fordi den fort kan slås over en elv og like fort rives.

KANALER OG SEILBARE ELVER.

Før det ble bygd veier, foregikk reisene over land helst på elvene, og da var vassdragene de viktigste forbindelsesveiene. I våre dager spiller slike seilbare elver fremdeles stor rolle, særlig for godstrafikken, og mange havner ligger ved munningen av store elver, f. eks. Fredrikstad, Göteborg, Hamburg, London og New York.

Innlandstransport til vanns har kunnet økes atskillig mange steder i verden ved at det er blitt bygd kanaler. Noen kanaler har samme nivå hele strekningen, mens andre klatrer opp og ned. Fartøyene blir hevet og senket ved hjelp av sluser.

En sluse er et kort stykke av kanalen som har vanntette porter i begge ender. En båt som er på vei oppover en kanal, og som altså skal heves til et høyere nivå, går inn i slusen, og så stenges portene. Deretter åpnes sluselukene i den øvre porten, og vannet strømmer inn fra nivået som ligger høyere. Nå løftes båten av det stigende vannet til nivået er det samme som i slusen ovenfor. Så åpnes den øvre porten, og båten kan seile videre. Når en båt skal senkes, slippes vannet ut av slusen. Kanalene må ofte mudres opp så de er dype nok for fartøyene.

IEC 61508

standard for funksjonell sikkerhet

- Null risiko kan aldri oppnås
- Sikkerhet må legges inn fra begynnelsen
- Ikke-tolererbare risikoer må begrenses (SIL nivå)
- Safety integrity level = SIL
SIL 1, SIL 2, SIL 3, SIL 4 (best)
- Svært opptatt av (at) concurrency (er "vanskelig")
- ... men standarden er nok tilsvarende vanskelig!
- IEC 26262 er en substandard for bilindustrien

Jeg prøver å skrive et lite bloggnotat:

<http://oyvteig.blogspot.com/2012/01/038-concurrent-programming-paradigms.html>

Fra en artikkel om dataetikk og formelle metoder

For example, if a program changes a single "1" to a "0", the entire program could result in completely opposite behavior. De Millo, Lipton, and Perlis call this property the **discontinuity property of software**...

.. imagine if one missing bolt in an American aircraft carrier turned it into a Soviet submarine. That's software.

"When Formal Systems Kill: Computer Ethics and Formal Methods"

Darren Abramson Lee Pikey

<https://www.cs.indiana.edu/~lepike/pubs/fm-ethics.pdf>

61508 & *etikkvalg*?

- Språk:
 - C, MISRA-C
 - C++, MISRA-C++
 - Java
 - Go
 - ...
- Operativsystem, rammeverk. Hyllevare:
 - Linux (subset, L4 mikrokjerne)?
 - ThreadX, VxWorks, Integrity, QNX?
 - AUTOSAR?
 - ...
- Hjemmesnekret?
 - SDL kjøresystem
 - ChanSched
 - BufChanSched
 - ...

<http://oyvteig.blogspot.com/2012/01/038-concurrent-programming-paradigms.html>

Kontaktinfo

Dette foredraget: http://www.teigfam.net/oyvind/pub/NTNU_2012/foredrag.pdf

Dette faget på NTNU: <http://www.itk.ntnu.no/fag/TTK4145/information/>

`oyvind.teig@teigfam.net`

Les dette og mer på <http://www.teigfam.net/oyvind/pub/>

♣ *Plutselig* er dere besteforeldre!

n for i 2048



1976

♣ *Plutselig* er dere besteforeldre!

n for i 2048 er det 36 år siden 2012

2012

Da har du jobbet i 36 år!





Fra harde μ -sekunder til forte år:
sann tid i industrien

takk
for meg!

