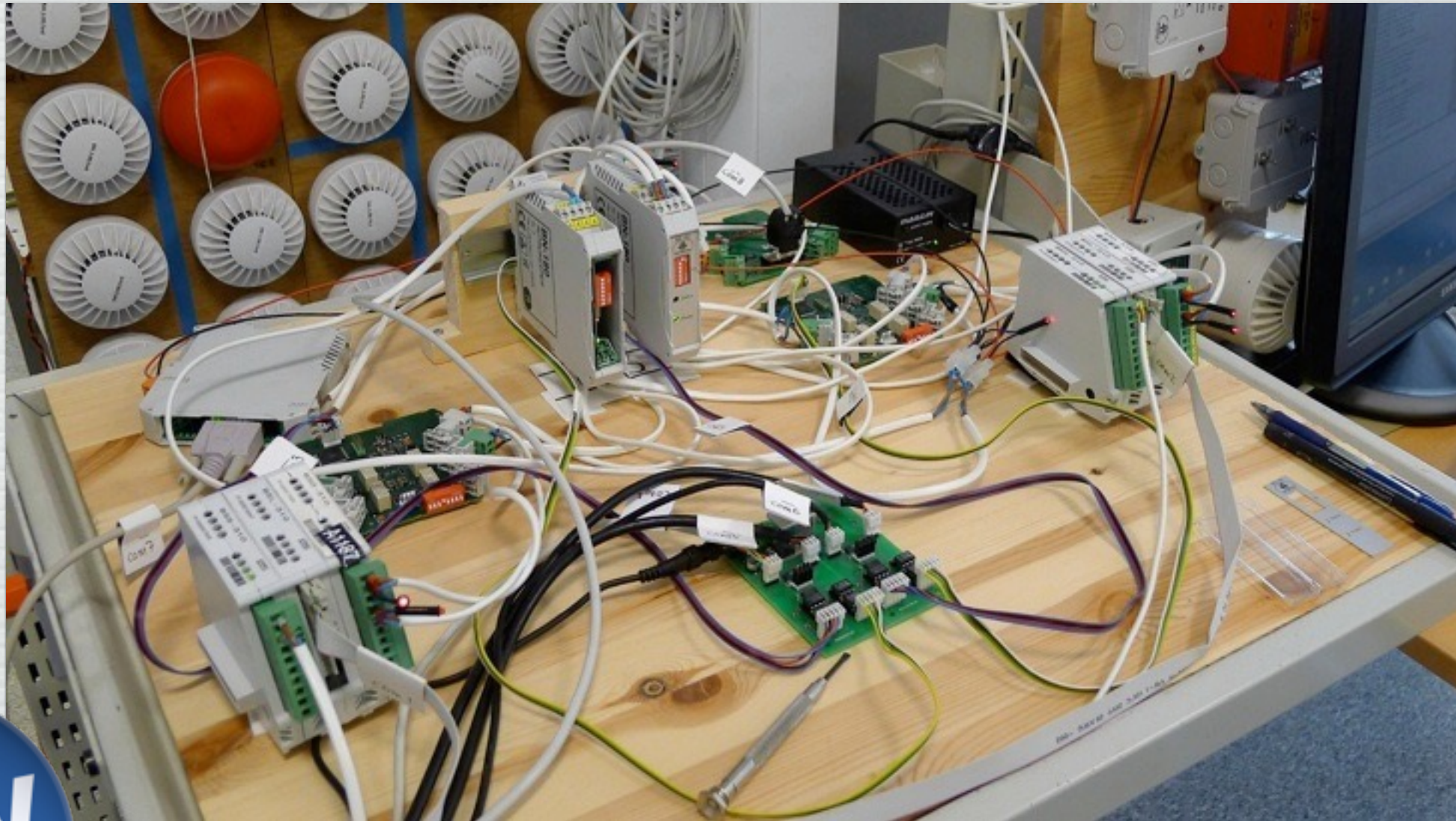


Fra harde μ -sekunder til forte år: sann tid i industrien



Øyvind Teig
Senior utviklingsingeniør, Autronica

Gjesteforelesning, 20. april 2015
NTNU, fag TTK 4145 Sanntidsprogrammering (Real-Time Programming)

http://www.teigfam.net/oyvind/pub/NTNU_2015/foredrag.pdf

Autronica Fire and Security (AFS)

- Over 1000 mill NOK omsetning (2014)
- 480 ansatte med ca. 35 på utvikling:
- På Lade, her i Trondheim
 - Utvikling i hovedsak Trondheim
 - Mindre enn ti personer i Gdansk
- Samme sted, og minst seks logoer ->
- Begynte med "gründerne" i 1957
- Børs og forskjellige eiere

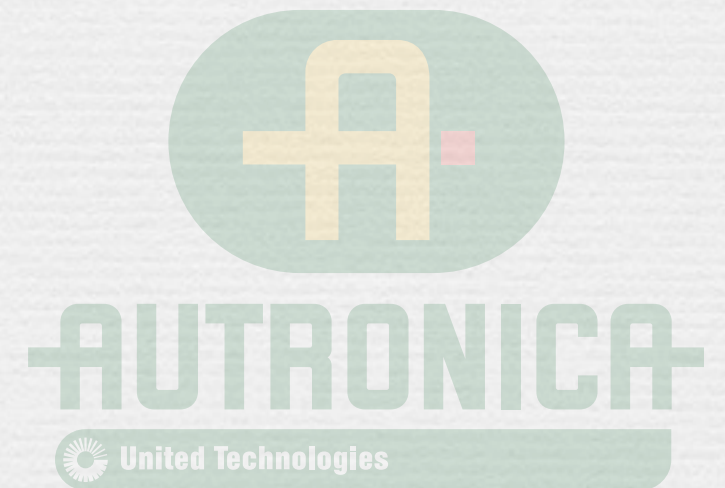


Autronica Fire and Security (AFS)

Siden 2005



- UTC = United Technologies Corporation
- 212K
 - UTC Building & Industrial Systems (119K)
 - ☒ Autronica Fire and Security
 - ☐ ...
 - Pratt & Whitney (34K)
 - UTC Aerospace Systems (43K)
 - Sikorsky (intensjon om salg) (16K)



"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
- Jeg viser min fagbakgrunn, og Autronica gjennom et sugerør. Det er



CV

AUTRONICA
United Technologies



- NTH, 1975
- 1976-2015 og jobbing som student 72-74
- HW og SW (mest SW)
- Har jobbet med embeddedsystemer hele tida
- Publisert en del - relativt uvanlig i industrien

<http://www.teigfam.net/oyvind/home>

«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 diplomten 1975)

Den papirtapen inneholder
fremdeles programmet!



Sann tid
går alt for fort!

«Pappesken» (SW)

Øverst til høyre
(fra diplomten 1975)

Den papirtapen inneholder
fremdeles programmet!



- Sanntidsprogrammering er gøy!
- Det kommer til å bli mer av det!
- Inkludert «concurrency», parallellitet og multi-core



«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 med det nye internasjonale kravet "Safe Return to Port"
Brannvarsling

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 komplekst, og har flere aksjonsspråk under paraplyen «Executable UML»
- Java -"synchronized" - det første allment brukte språket som har concurrency-tankegang innebygd..
 - ..og som er "livsfarlig" å bruke!
 - ..men som man har bygd CSP oppå (f.eks. JCSP)

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 kompleks og har flere aksjonsspråk under paraplyen «Executable UML»
- Java - "synchronized" - det første allment brukte språket som har concurrency-tankegang innebygd..
 - ..og som er "livsfarlig" å bruke!
 - ..men som man har bygd CSP oppå (f.eks. JCSP)

Dette går da ikke fort!

Lange år

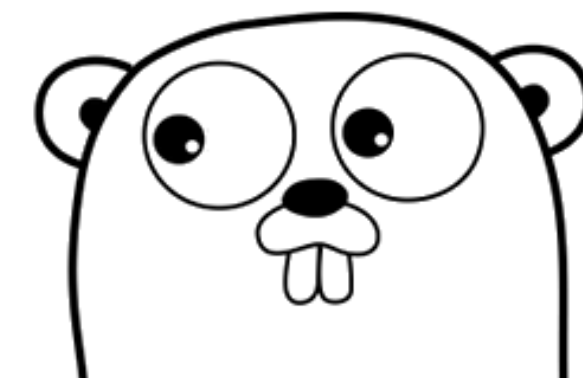
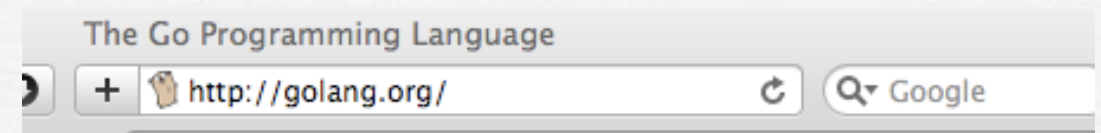
- OO er fra sekstitallet
 - I bruk nå, men lite i embedded (blir mer og mer!)
- Sanntidsmekanismene er fra 60-70+ tallet
 - Semafor etc. i bruk siden da
- CSP etc. fra 70-tallet. Lite i bruk (ennå..(?))
- UML 2.0 er komplekst, og har flere aksjonsspråk under paraplyen «Executable UML»
- Java -"synchronized" - det første allment brukte språket som har concurrency-tankegang innebygd..
 - ..og som er "livsfarlig" å bruke!
 - ..men som man har bygd CSP oppå (f.eks. JCSP)

21 år etter occam (i 2009)

- **Go**
- chan, go, select, eksplisitt typekonvertering, array- oppdeling, ikke pekeraritmetikk
- Ikke "sikkert" (no «parallel usage rules»)

27 år etter occam (i 2015)

Høsten 2009



Download Go

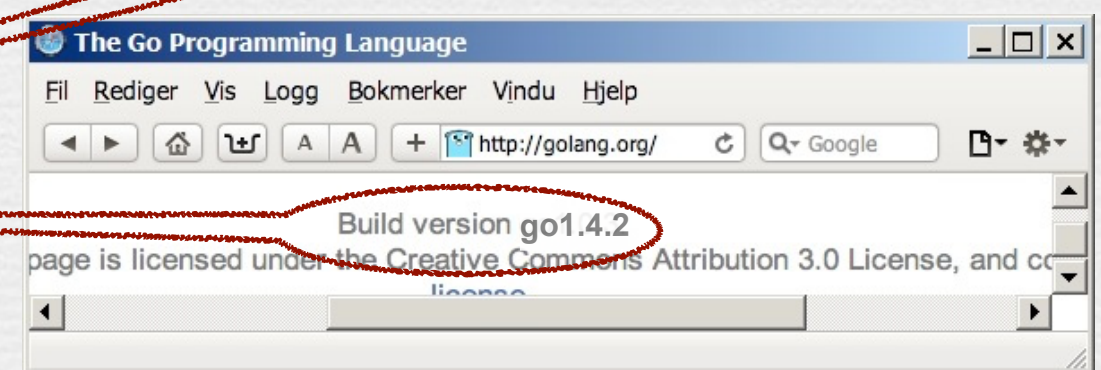
Binary distributions available for
Linux, Mac OS X, Windows, and more.

Featured articles

Go version 1 is released

28. mars 2012

Nå



«Go blir mer og mer populært innad i Google, og begynner så smått å anses som modent nok til å brukes til viktige oppgaver

Mail i 2014 fra kamerat hos Google

Etter forelesningen ønsker jeg

- At dere skal huske at å jobbe med embedded-systemer, er gøy
- At dere skal huske å fortelle om det dere har lært i dette faget,
- ..og ikke bare ta tradisjonen slik den er!

Små embeddedsystemer i industrien

- Vil nok forholde seg til C en god stund til!
- Vil nok ha prosjektledere og deltakere uten «Go erfaring» (en stund til?)!
- Ikke overta deres verktøykasse uten å legge noe annet i den (som synkrone kanaler og tette prosesser

Kjøresystem I

For små (AVR / XMEga / ARM) system i ANSI C

- I flesteparten av kontrollerne våre benytter vi et asynkront SDL kjøresystem
- I tre kontroller-enheter også synkrone kanal-baserte CSP kjøresystem (bygd oppå SDL-systemet eller selvstendig)

Kjøresystem II

For større 32-bits system (AVR32 / ARM)

Linux

- Buildroot-basert (løser mye avhengigheter, funksjonalitetsvalg, bygd for embedded)
- Kjerne 2.6 (snart 3.0)
- Egenmodifisert uClibc (pthread, timer..)
- Egne drivere (konsulentutviklet), nå lagt inn i kjerne 3.0 for AVR32 og ARM: HSR Ethernet (redundant ring), etc., med GPL lisens
- Egne RAM-tester etc.



Kjøresystem II

For større 32-bits system (AVR32 / ARM)

Linux

- Buildroot-basert (løser mye avhengigheter, funksjonalitetsvalg, bygd for embedded)
- Kjerne 2.6 (snart 3.0)
- Egenmodifisert uClibc (pthread, timer..)
- Egen drivere (konsulentutviklet), nå lagt inn i Kjerne 3.0 for AVR32 og ARM: HSR Ethernet (redundant ring), etc., med GPL lisens
- Egne RAM-tester etc.

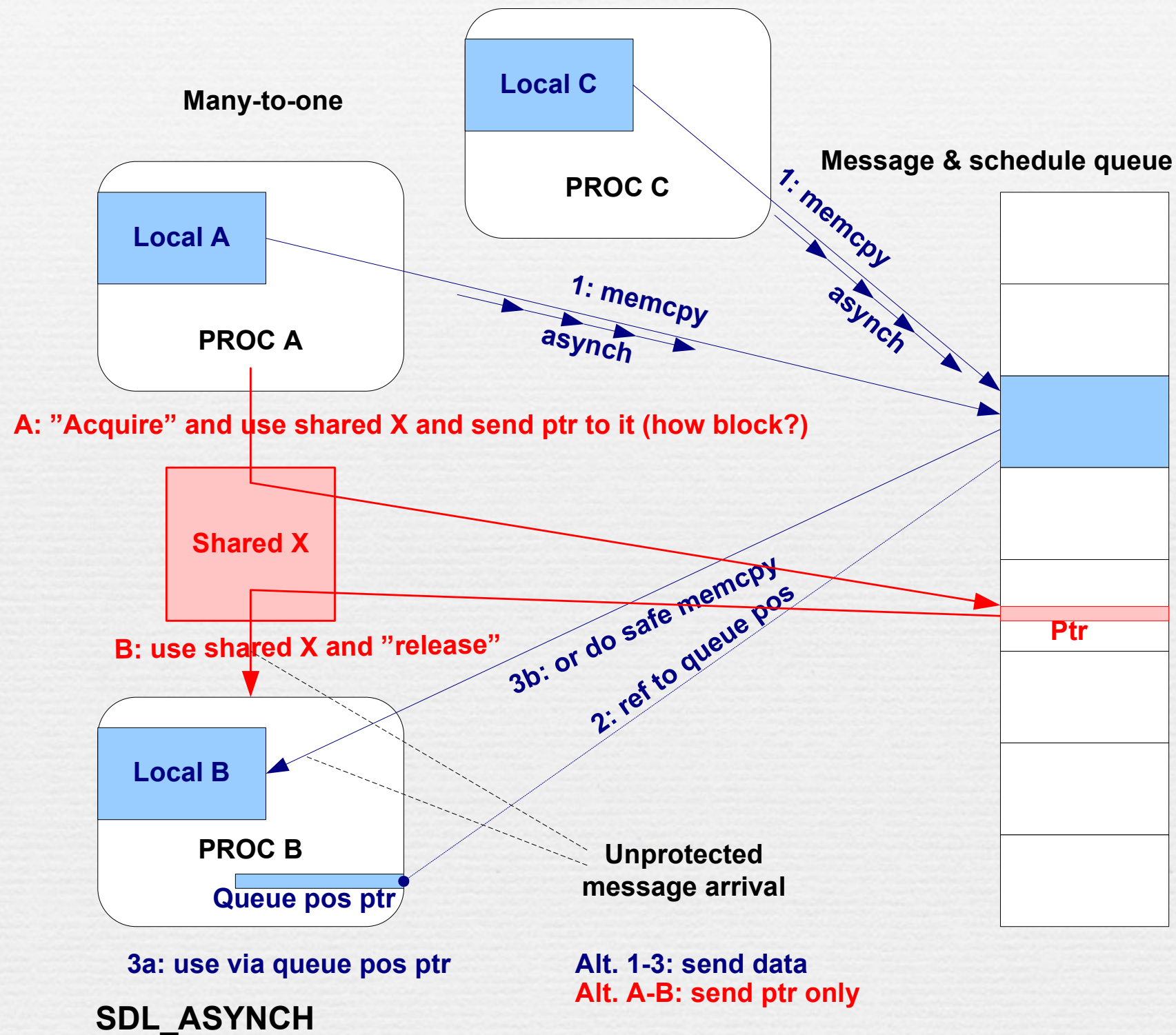


Ikke vår kjernekompetanse (som er brannvarsling!)

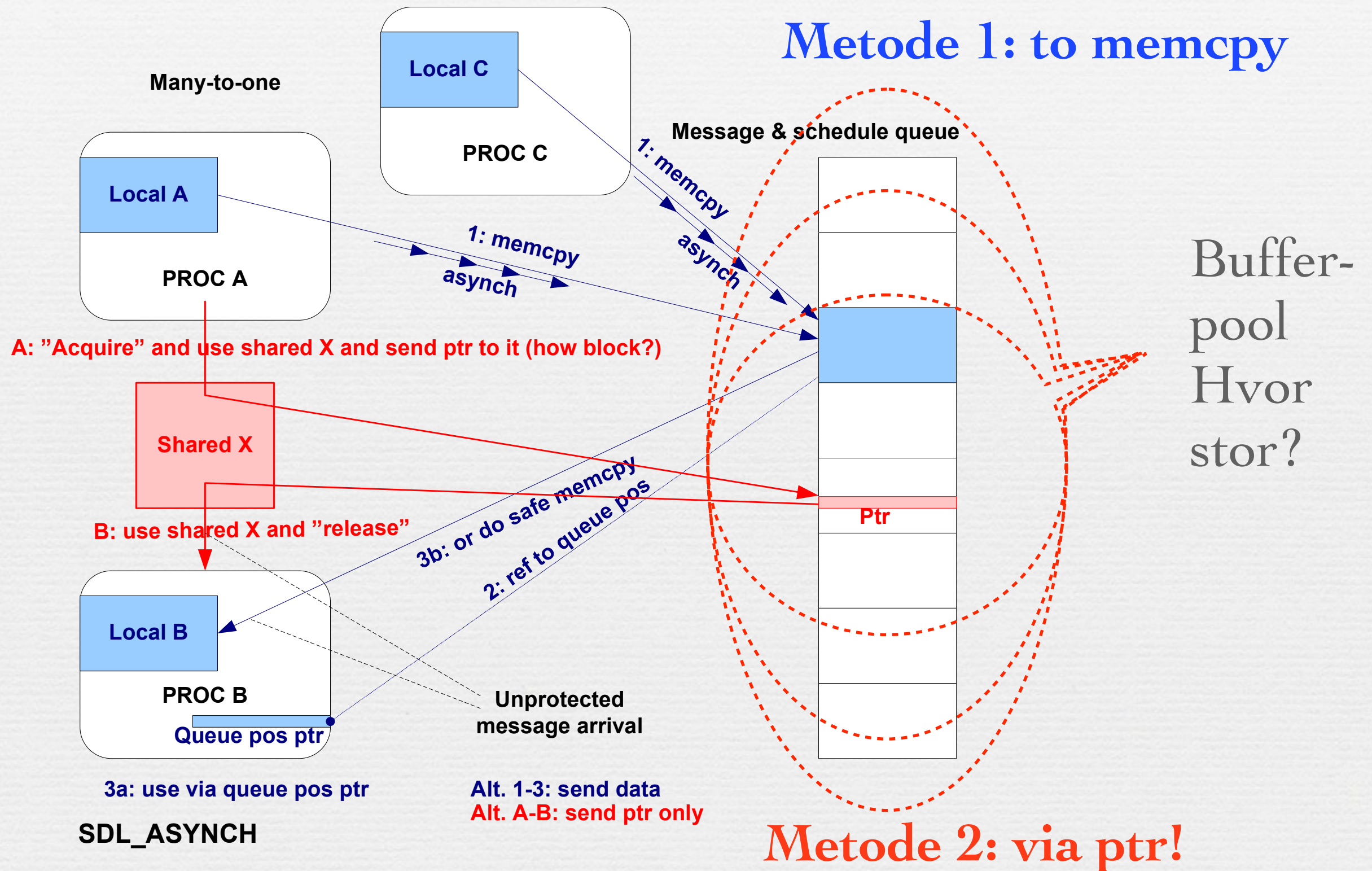
Meldingsbaserte systemer

- Via poolbasert meldingssystem
- Via kanaler (pipes?)
- Er **interface** som i xC (senere) også «meldingsbasert»?

Poolbasert meldingssystem (SDL)

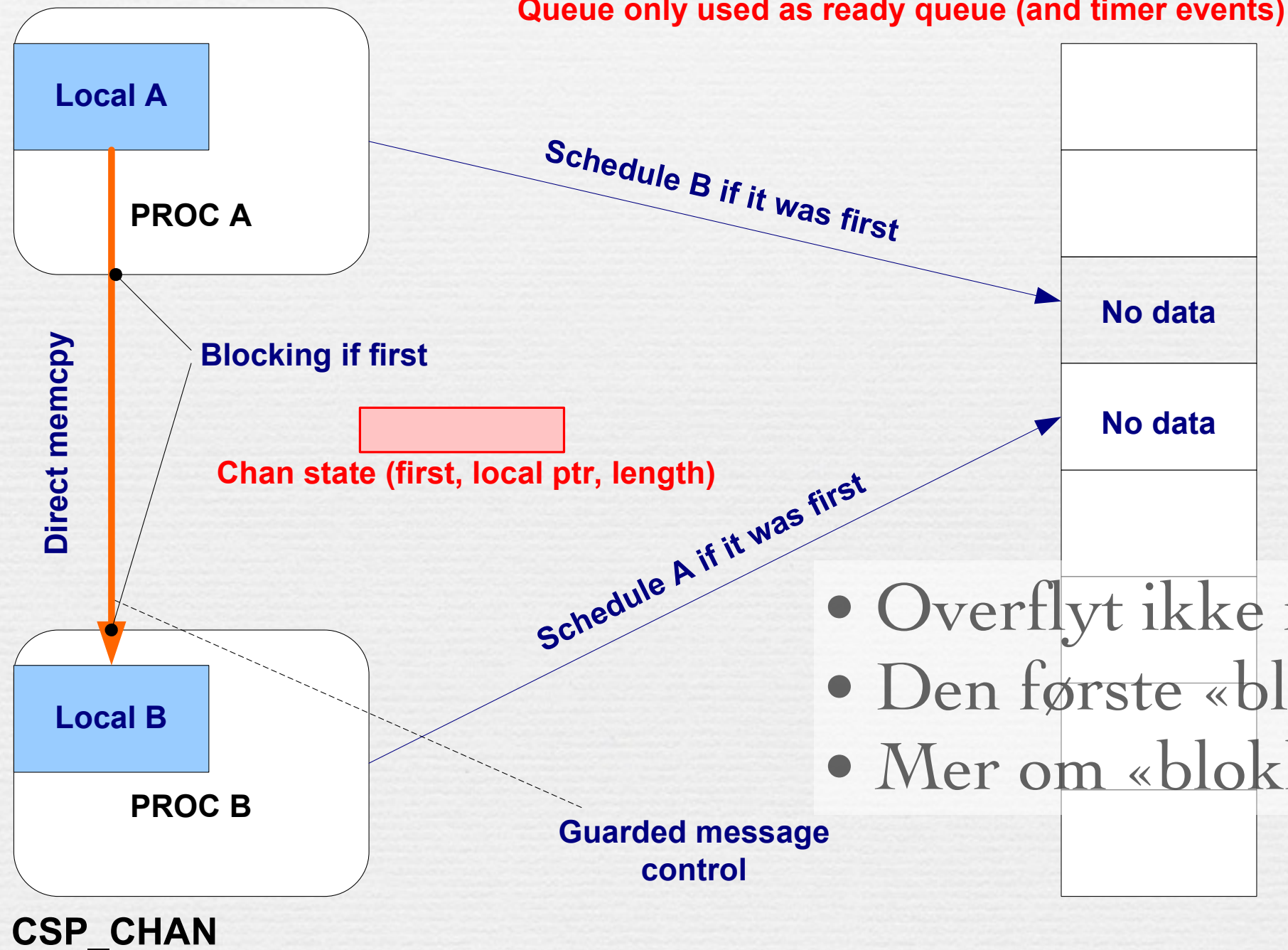


Poolbasert meldingssystem (SDL)



Kanaler (CSP)

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



- Overflyt ikke mulig
- Den første «blokkerer»
- Mer om «blokkering» senere

S a n n t i d

"Verden omkring oss", 1955 ("Odham's 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 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

porter som lekterne går inn og ut gjennom

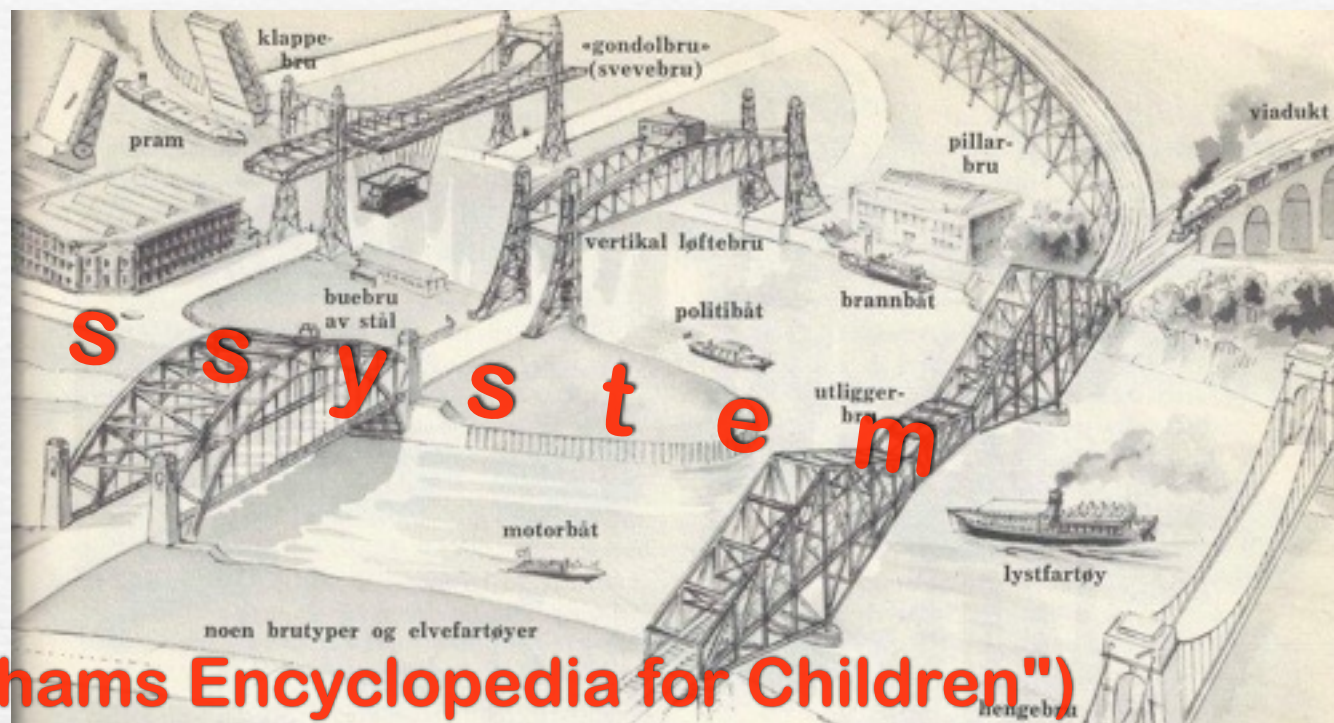
lekteren stiger

vannet renner inn gjennom sluselukene, så vannstanden stiger

lekter på høyere vannstand venter på å komme inn i slusen

lekter på lavt nivå venter på å komme inn i slusen

lukket sluseport; når den åpnes og vannet renner ut, synker nivået



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 hengebruerne av stål og betong har ikke stor likhet med de enkle hengebruerne 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.

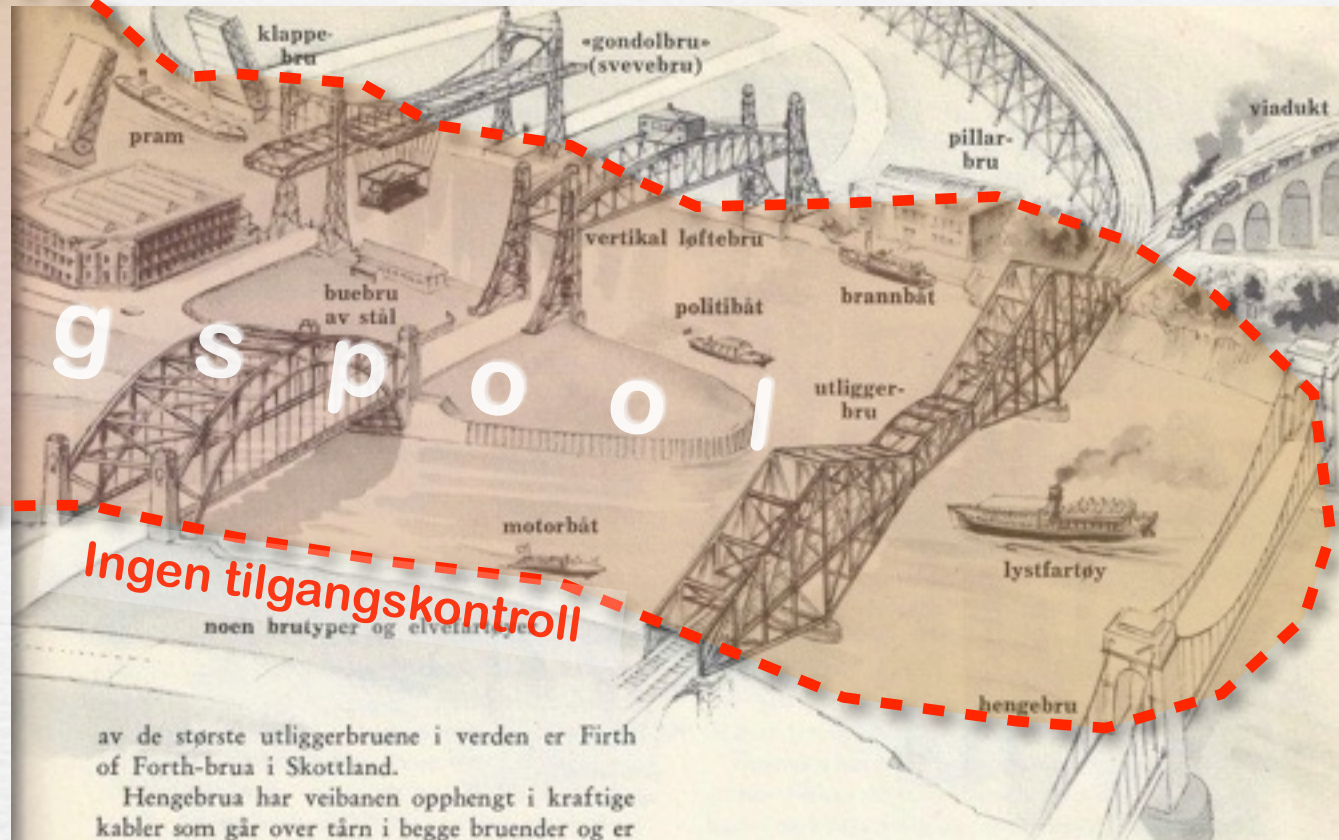
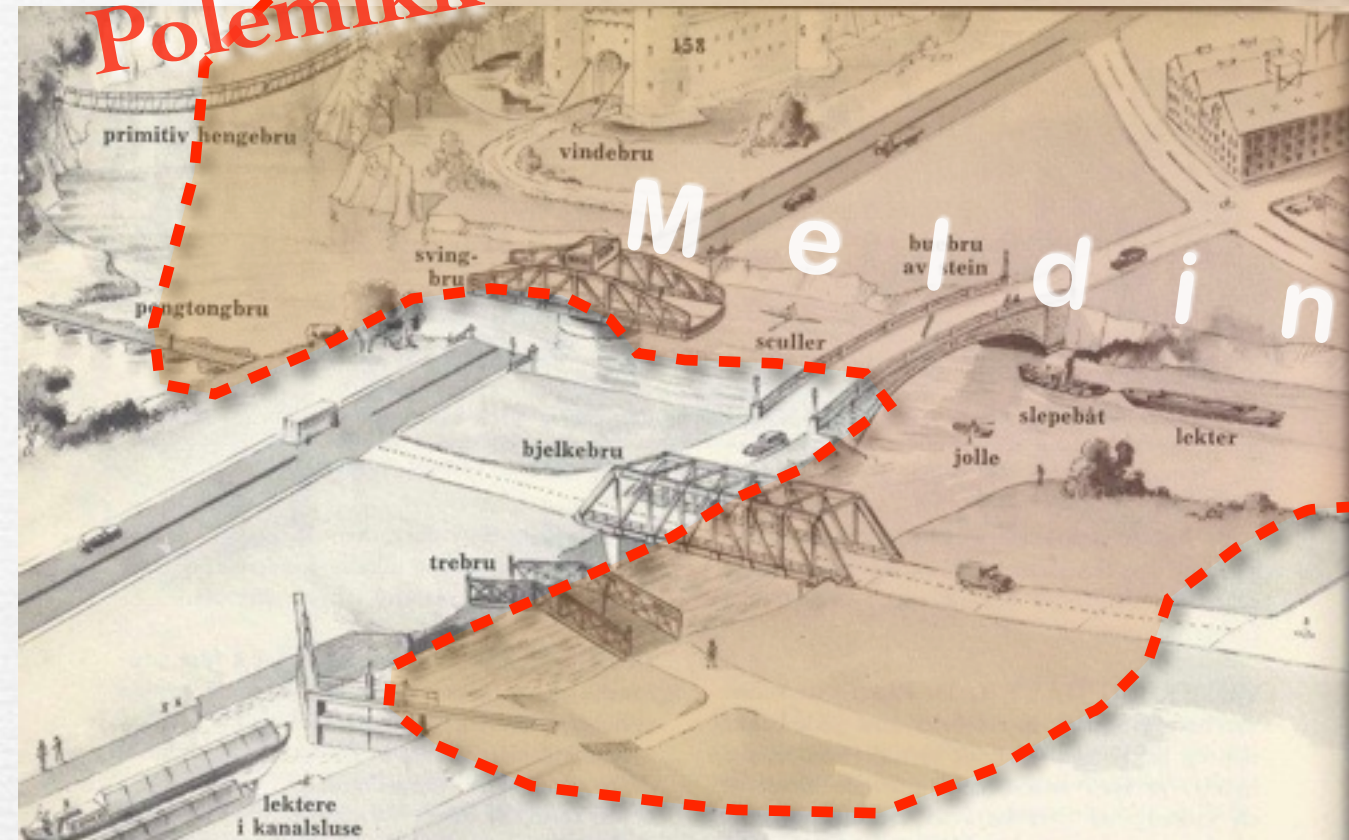
Innlandstransport til vanns har kunnet økes atskillig mange steder i verden ved at det er blitt bygd kanaler. Noen kanaler har samme nivå hele strekningen, mens andre klatrer opp og ned. Fartøyene blir hevet og senket ved hjelp av sluser.

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.

M e l d i n g

g s p o o l

Ingen tilgangskontroll



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

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 vær. 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.

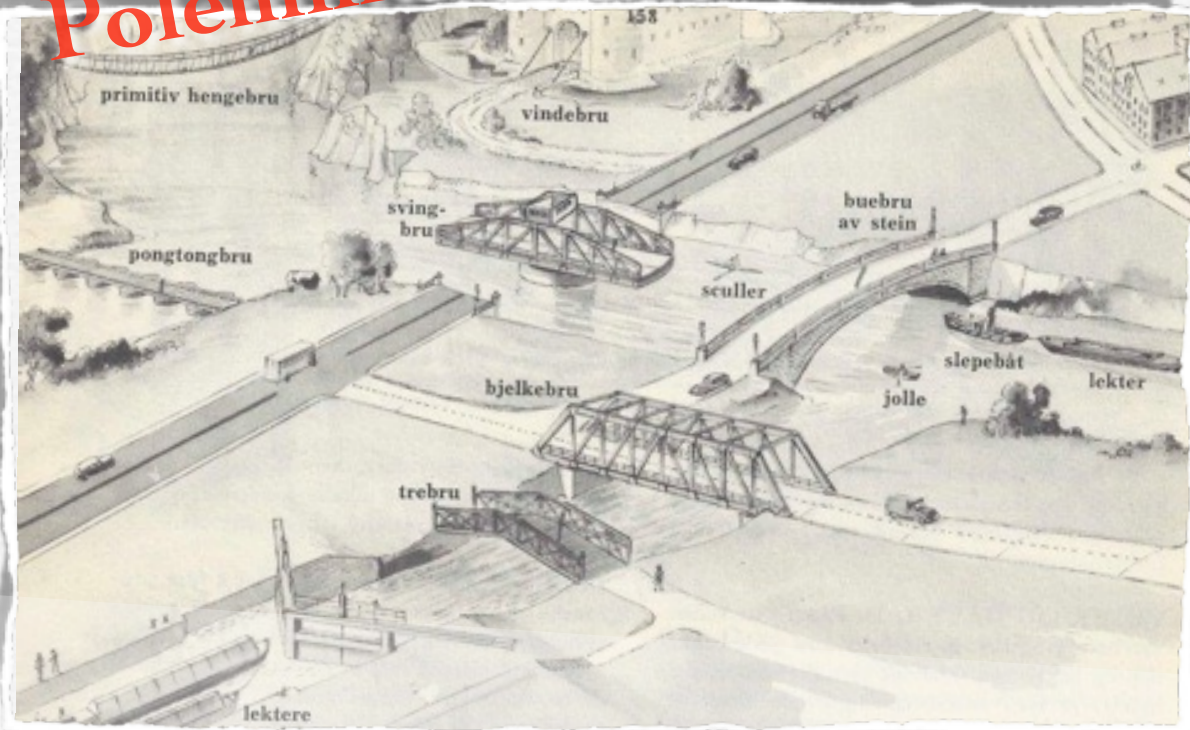
En sluse er et kort stykke av kanalen som har vanntette porter i begge ender. En båt som er på vei oppover en kanal, og som altså skal heves til et høyere nivå, går inn i slusen, og så stenges portene. Deretter åpnes sluselukene i den øvre porten, og vannet strømmer inn fra nivået som ligger høyere. 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.

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

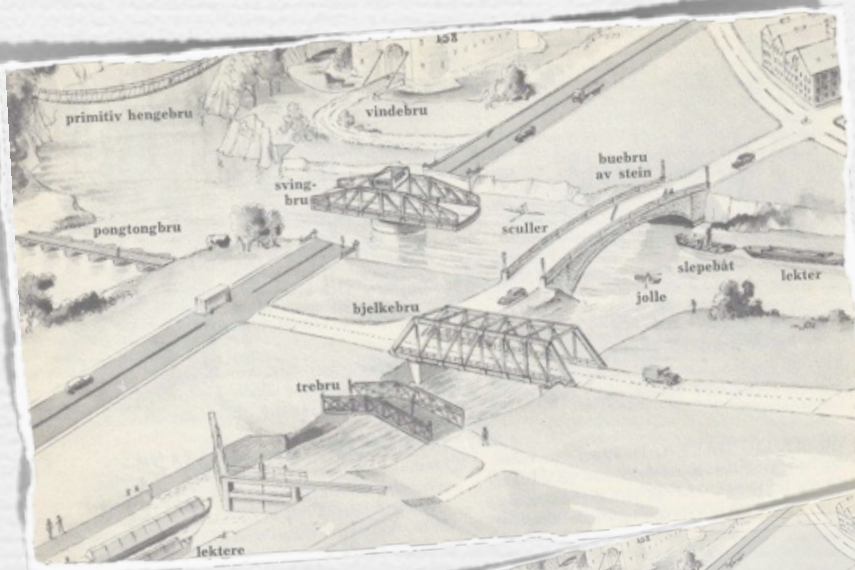
håndtak til å åpne og lukke sluseportene med



Polemikk



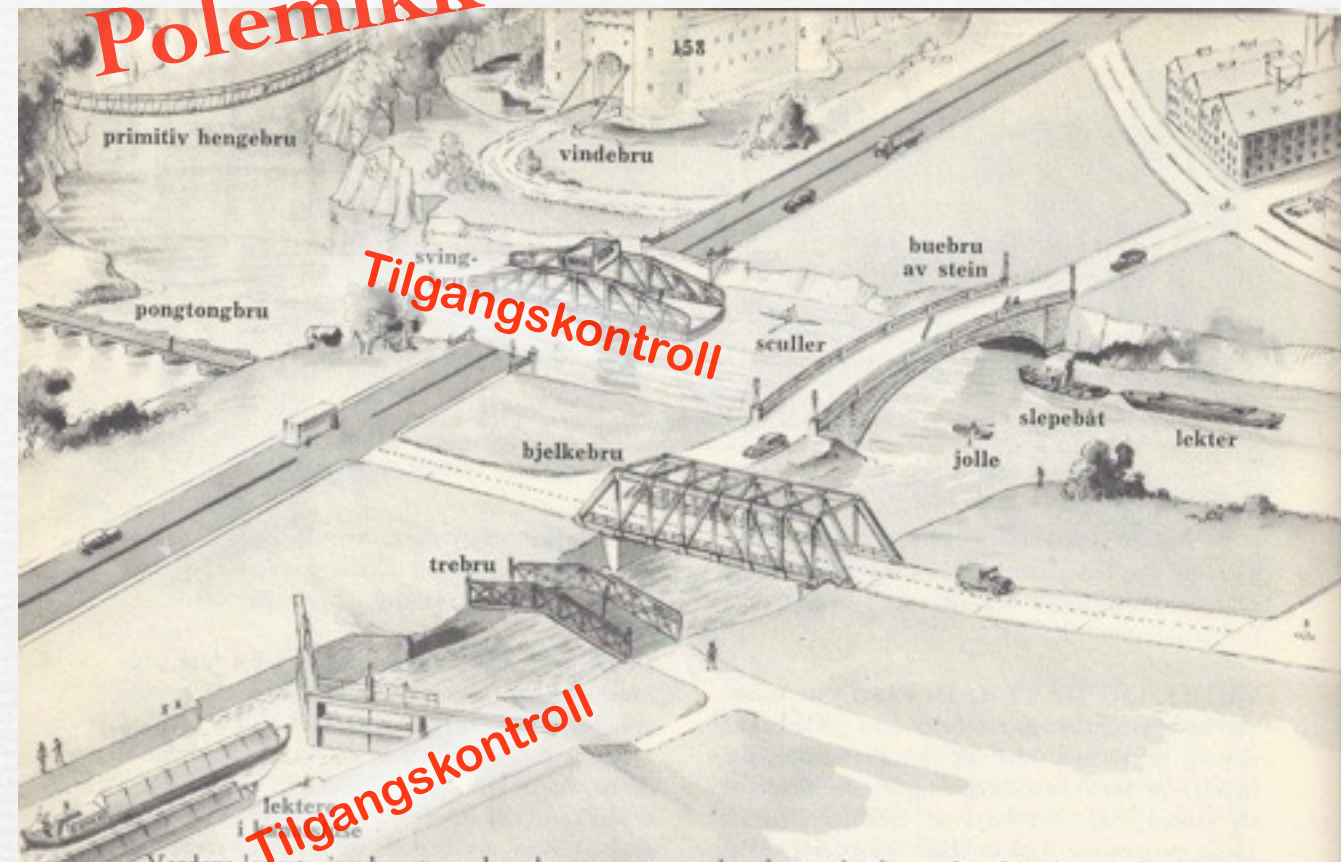
Om det sendes en eneste melding for mye så kræsjer systemet og alle meldingene mistes...



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



..for å analysere en slik situasjon er nesten umulig



Tilgangskontroll

Tilgangskontroll

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

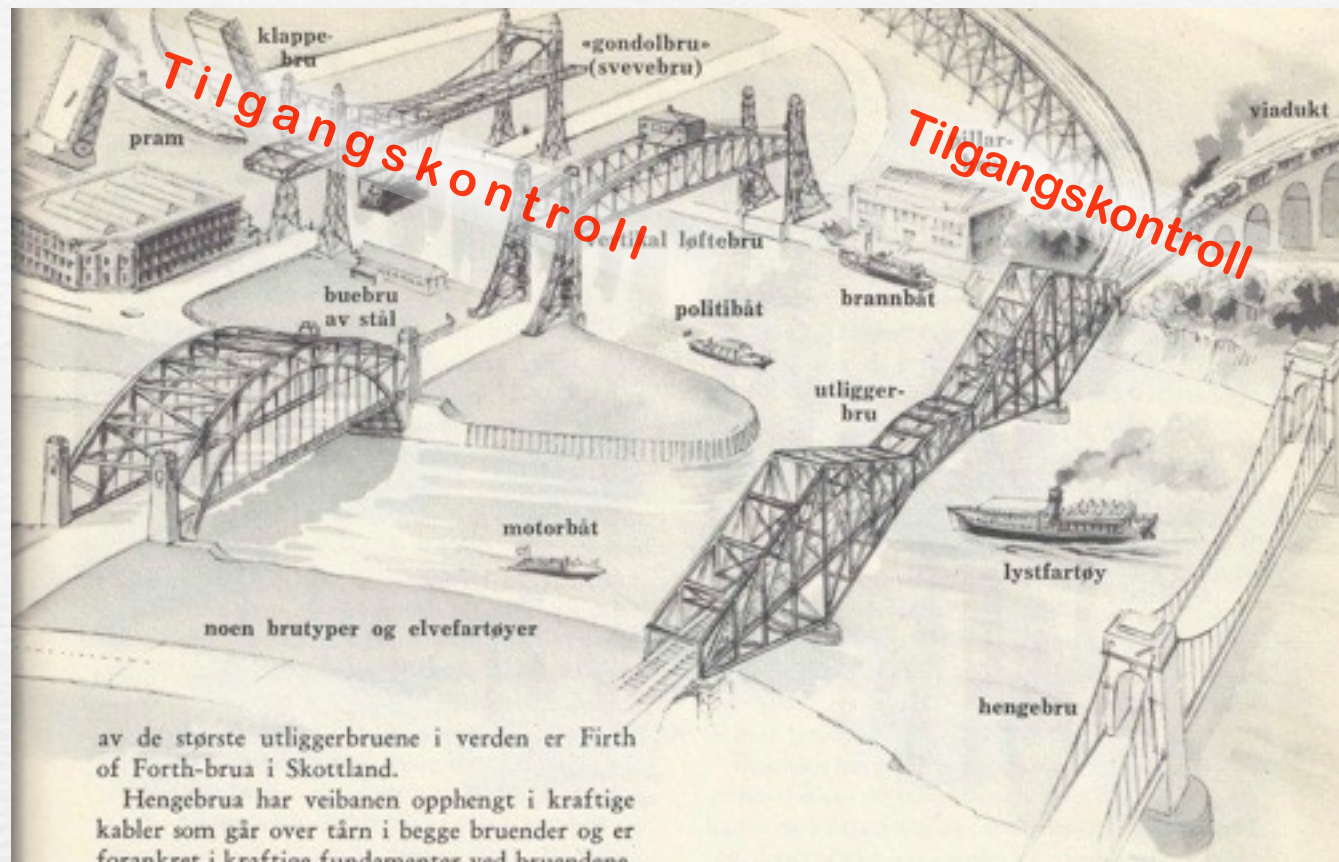
bygd som hvelv — kanskje den vanligste brutypen — består av en eller flere buer etter hverandre som bærer veibanen. En bruker stein, murstein eller betong som byggemateriale. Mange av de store moderne buebruene er bygd av stål. Den største i verden er brua over havnebassenget i Sydney i Australia. En 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

Tilgangskontroll



Tilgangskontroll

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 fløede 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 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.

Alvor

Tilgangskontroll

Tilgangskontroll

Overflyt

✓ Overflyt er naturlig

Men ikke mellom programmer

✓ Det skal håndteres på applikasjonsnivå

Og ikke av operativsystemet

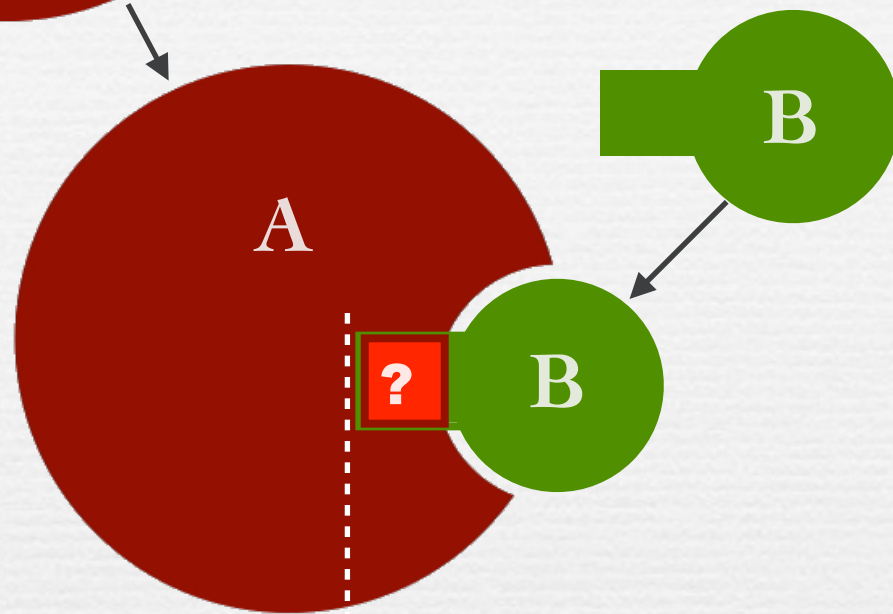
✓ Meldinger skal kastes bevisst

Men ikke alle (som er underveis)!

✓ Meldingsflyt skal håndteres bevisst

Så å holde igjen «ved halvgjort» er flott!

Concurrency: independence of execution

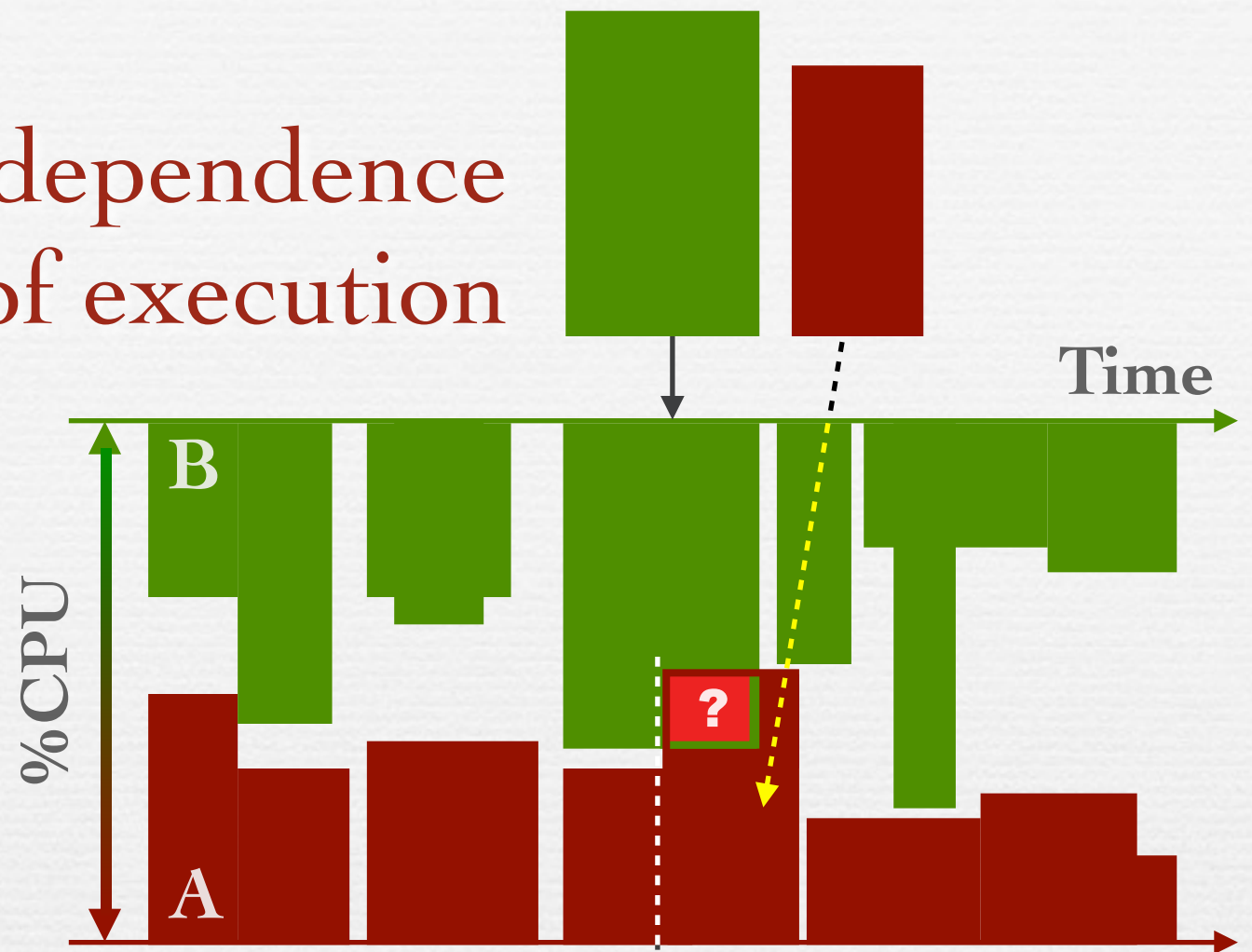


61508 terms (later):

Spatial

Ouch?

Or ok communication?



Temporal

Ouch?

Or bad «blocking»?

Worries:

deadline, deadlock, race, safety, livelock, «blocking»

Seriell OO modellering

- Her benytter vi UML
 - Vi benytter Rhapsody
 - Klassediagrammer
 - Tilstandsmaskiner
 - Sekvensdiagrammer
- Håndskrevet
 - Aksjonskode
 - Lim mot resten av systemet
- Prosesser via operativsystem

Modellering

- Både assembler, C, Java, CSP, occam, Go og UML er abstraksjoner
- Altså «modeller»
- Men i varierende grad er de «formelle modeller» - dvs. kan være input til et verktøy som gjør verifisering for korrekthet
- Som PAT

Modellering

- Både assembler, C, Java, CSP, occam, Go og UML er abstraksjoner
- Altså «modeller»
- Men i varierende grad er de «formelle modeller» - dvs. kan være input til et verktøy som gjør verifisering for korrekthet
- Som CSPm/FDR3, Promela/Spin, LTSA, PAT

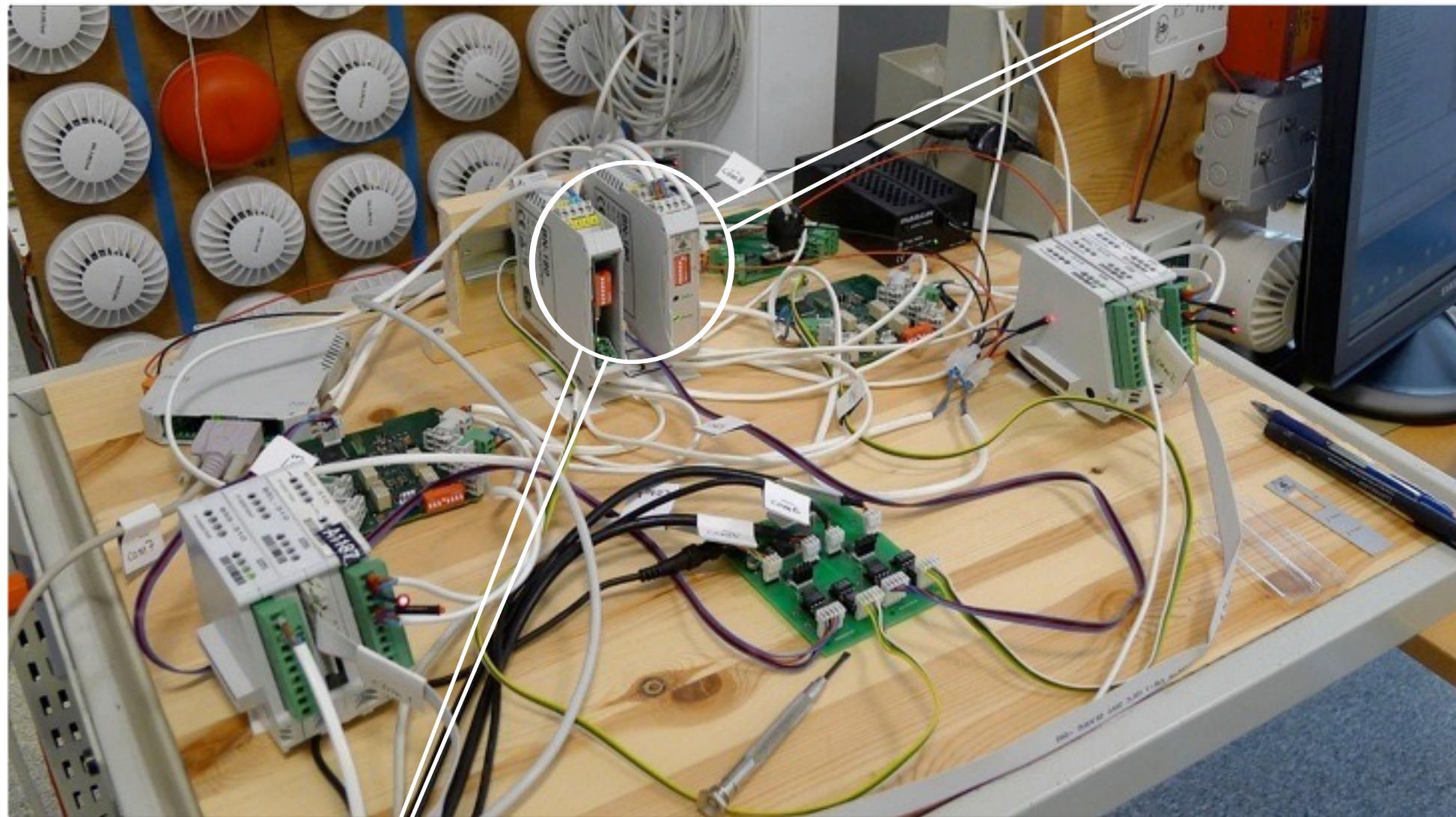
A red rectangular logo with the white text 'xC' inside.

New ALT for Application Timers and Synchronisation Point Scheduling

(Two excerpts from a small channel based scheduler)

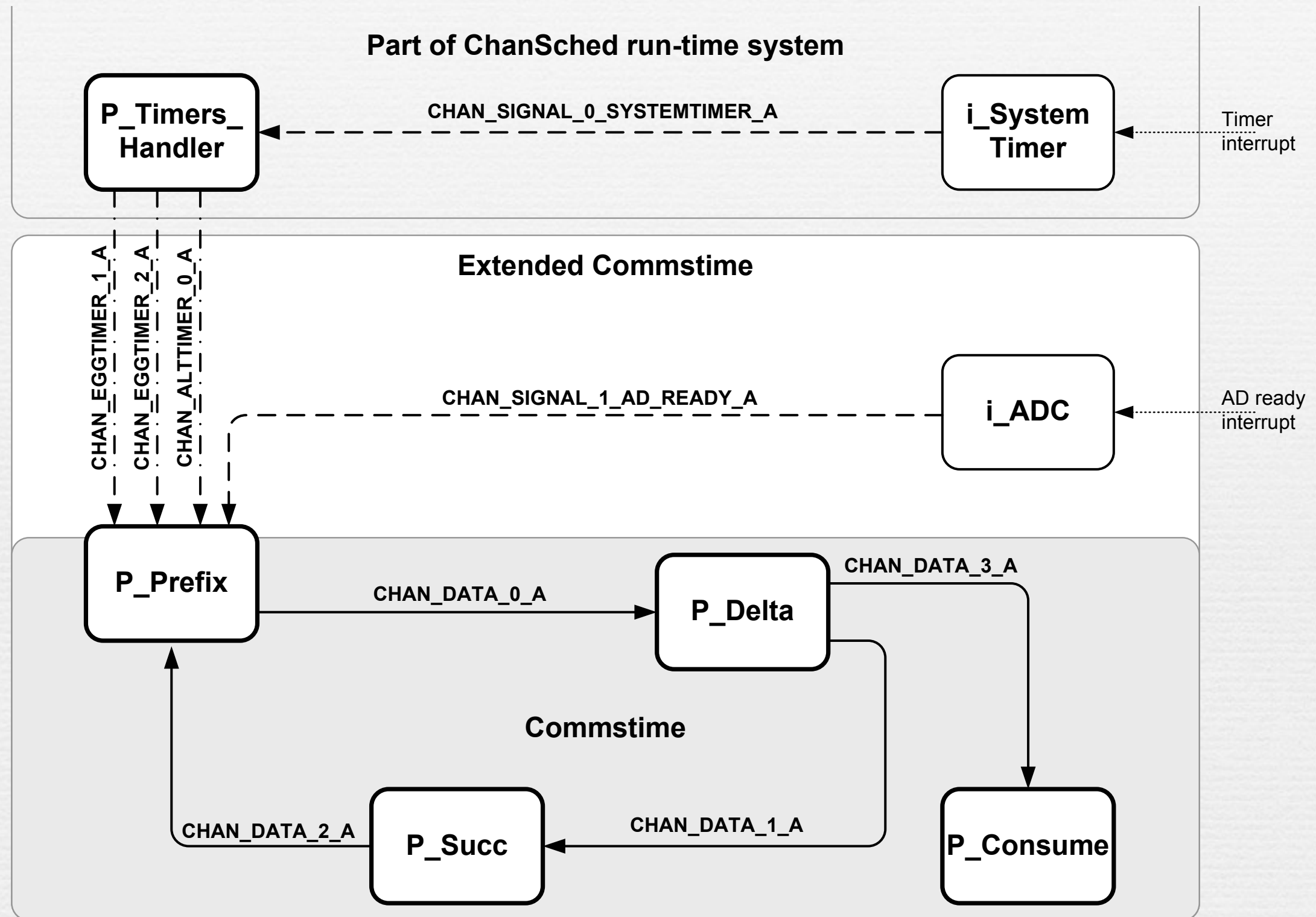
«ChanSched»

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/>

Testsystem for ChanSched



Prosess/dataflyt for utvidet «commstime»

En scheduler er ikke så usynlig som jeg trodde

```
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
```

Samme funksjonalitet!

Kjøresystem oppå annet kjøresystem

Kjøresystem alene

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. }
```

Obs: en inn-kanal og tre timere, hver sin type:


```
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.     }
```

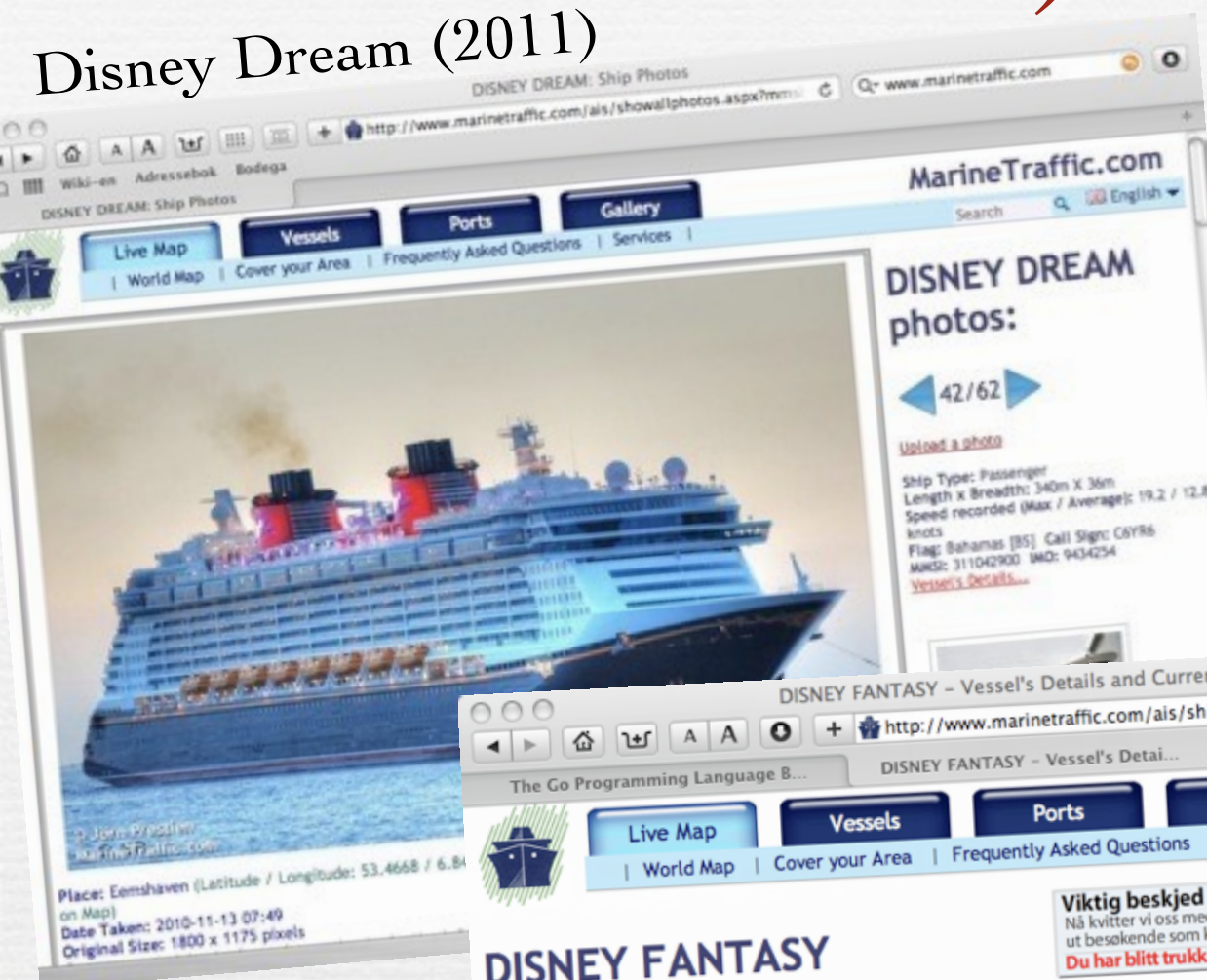
Nesten usynlig hopp
tilbake til der man
sist «blokkerte»

Bare denne er
synlig

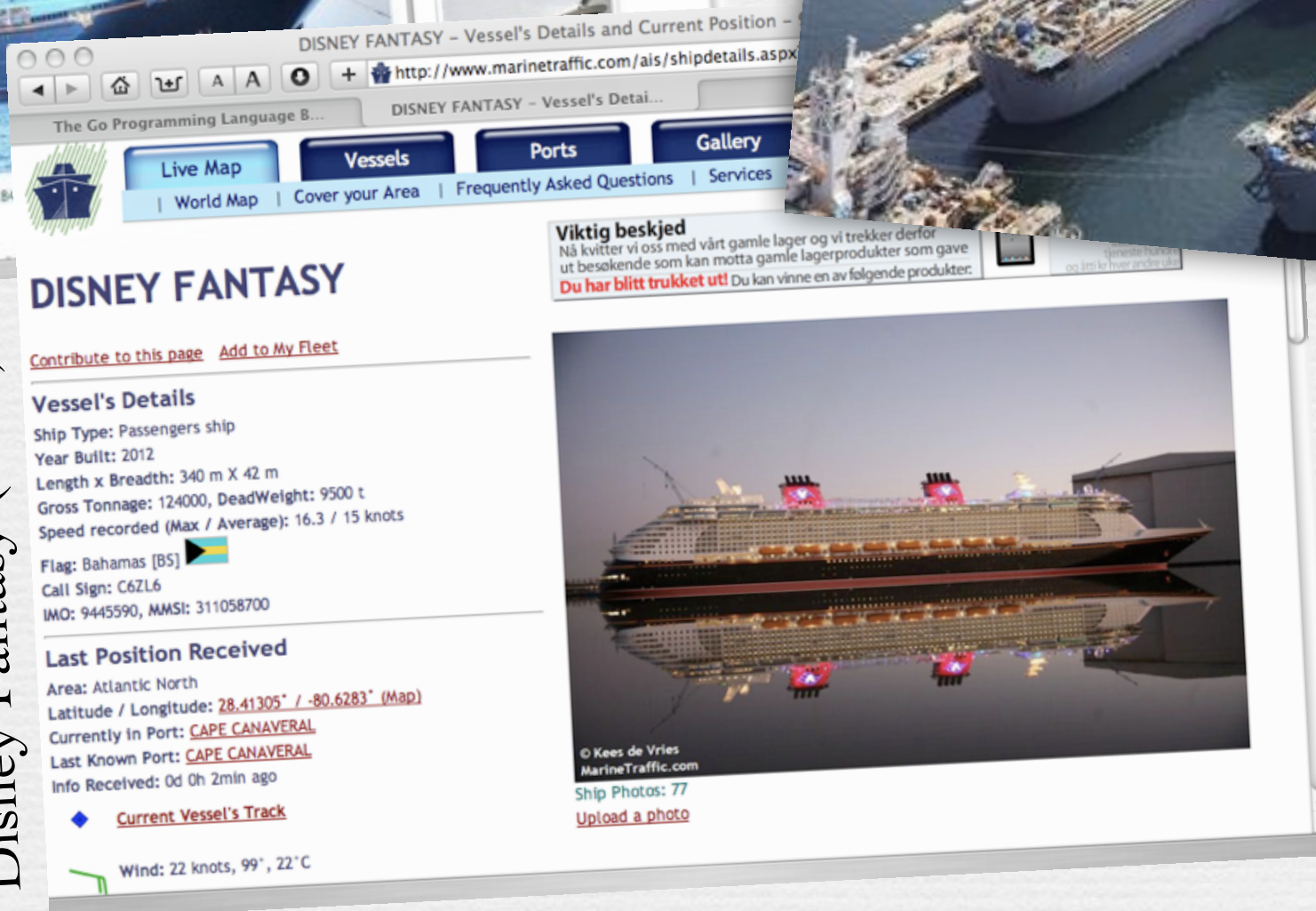
Genereres av
verktøy og ligger i
egen .h fil

Dette kjører ombord på

Disney Dream (2011)



Disney Fantasy (2012)



Pieter Schelte (2013)
Pioneering Spirit

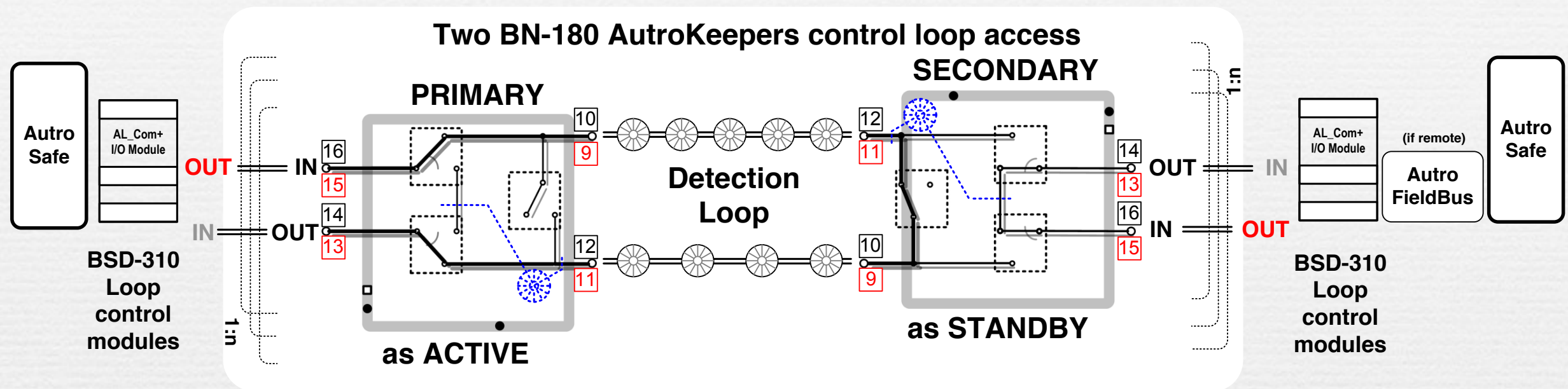
++

AutroKeeper
(BNA-180)
Safe Return to Port
Dual Safety



Brannvarsling

AutroKeeper: roller modellert i Promela



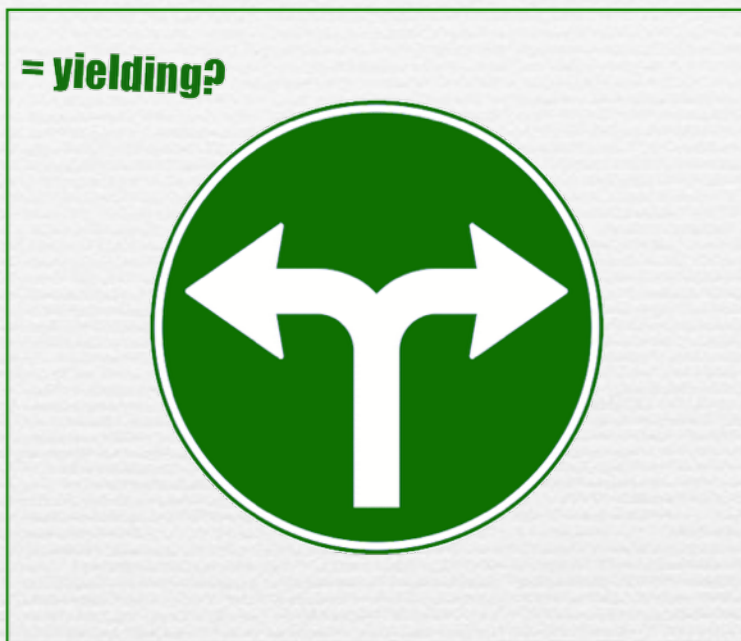
For å

- forhindre oscillasjoner (stabilitet)
- forhindre at begge blir frakoblet (alltid brannvarsling)
- forhindre samtidig styring fra to AutroSafe (WYSIWYG)
- Ascription
- Modus

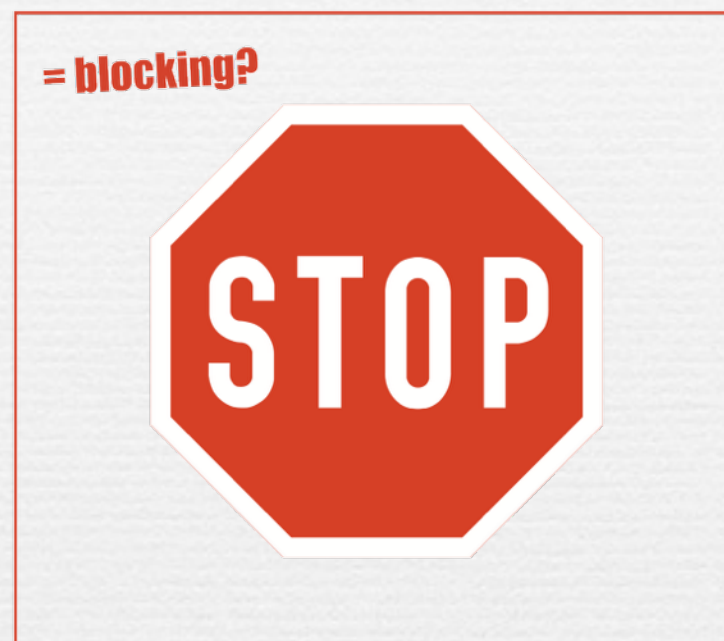
http://produkt.autronica.no/filesare/filArkivRoot/produkt/pdf/dokumentasjon/bn180_cn.pdf

Såkalt blokkerende

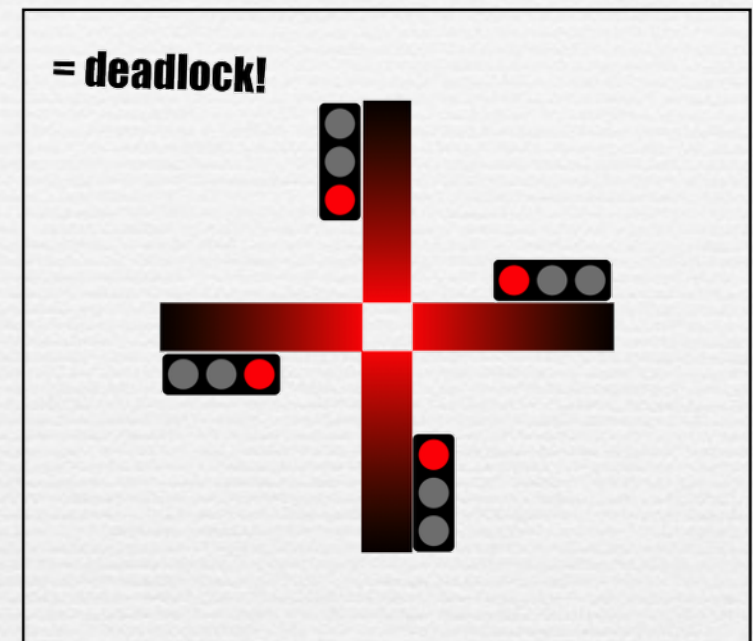
Which **block**ing do you mean?



The show goes on with this **blocking**



This **blocking** stops the show



This **blocking** stops the world

- Belastet term: vær presis!
- Fra normal venting (lokal) via dytting til side (forhindre andre) til gradvis tilfrysing (vranglås)
- «Ikke-blokkerende» algoritmer «blokkerer» like mye som den grønne: dvs. lokal venting tillates

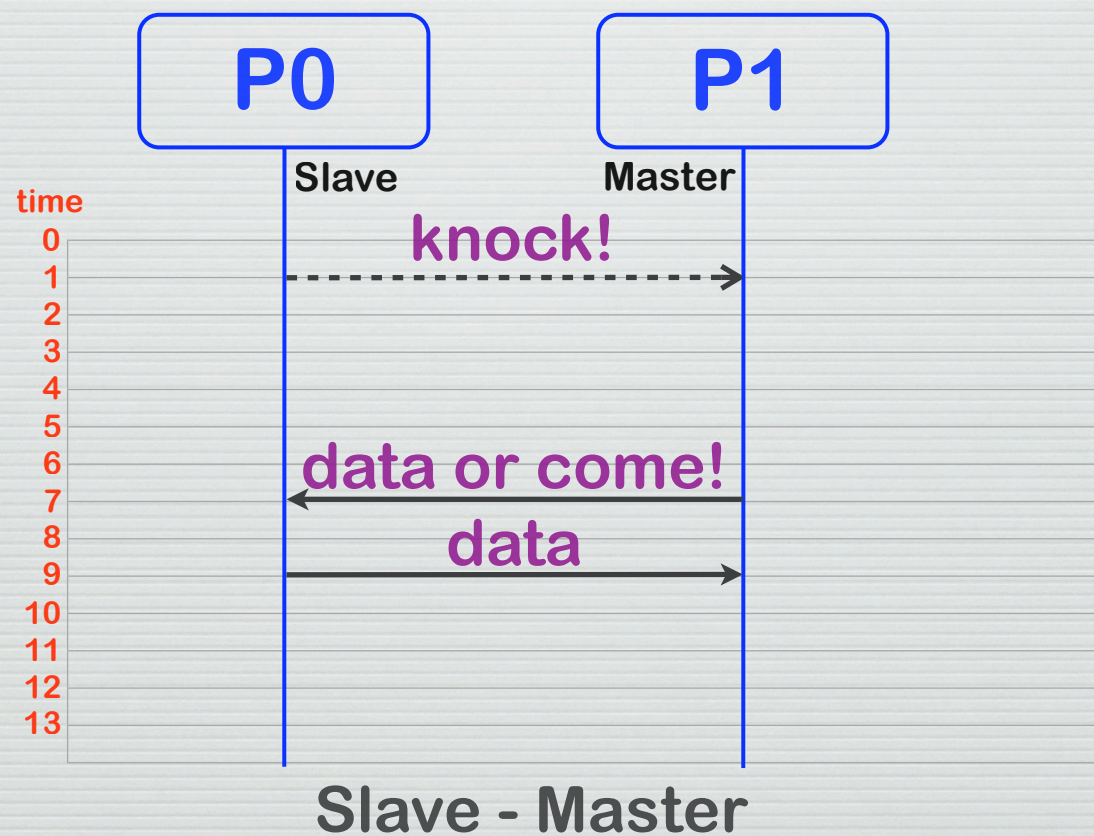
<http://www.teigfam.net/oyvind/home/technology/092-not-so-blocking-after-all/>

IEC 61508 standard for funksjonell sikkerhet for sikkerhetskritiske systemer

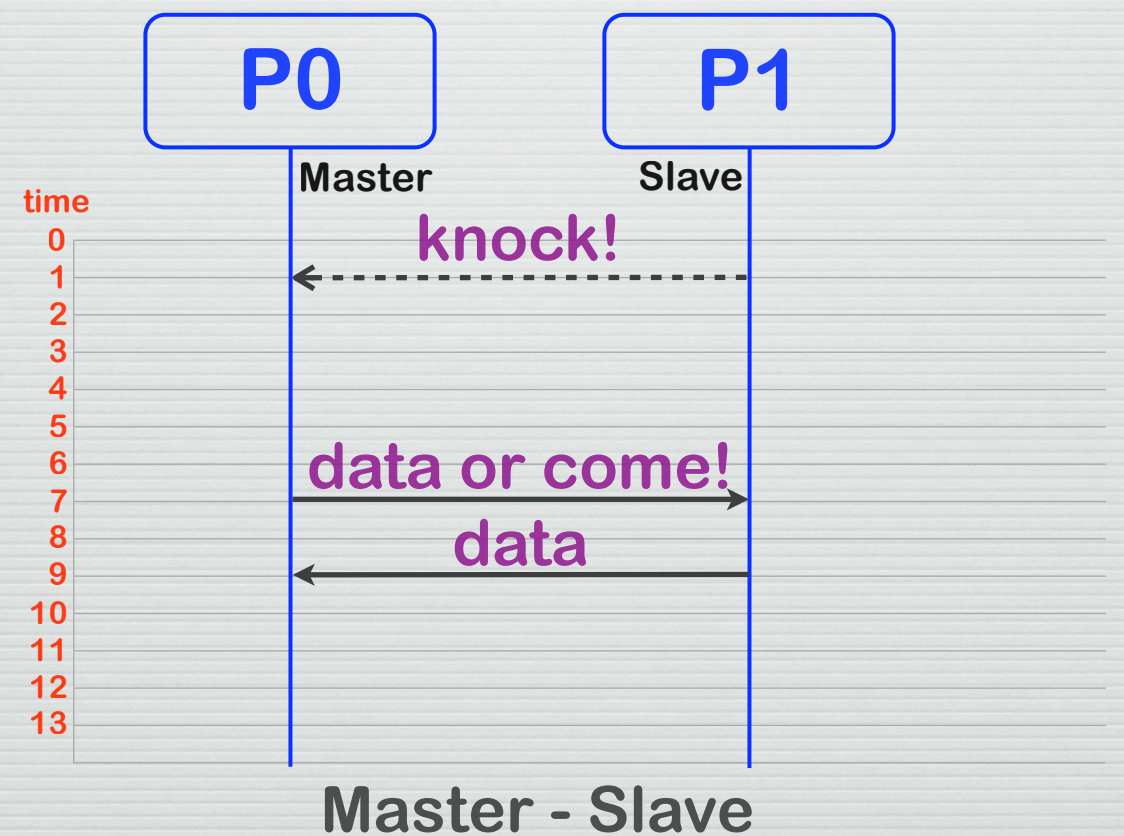
- Null risiko kan aldri oppnås
- Sikkerhet må legges inn fra begynnelsen
- Ikke-tolererbare risikoer må begrenses (SIL nivå)
- Safety integrity level = SIL
SIL 1, SIL 2, SIL 3, SIL 4 (best)
- Svært opptatt av (at) concurrency (er "vanskelig")
- ... men standarden er nok tilsvarende vanskelig!
- IEC 26262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

Roller mot vranglås

knock-come



9



8

Begge kan initiere data-transfer

XCHANS: Notes on a New Channel Type

Øyvind TEIG¹

Autronica Fire and Security AS², Trondheim, Norway

(3 typos fixed, 31.Aug.2012)

Ny kanaltype og oppførsels-mønster

Abstract. This paper proposes a new channel type, **XCHAN**, for communicating messages between a sender and receiver. Sending on an **XCHAN** is asynchronous, with the sending process informed as to its *success*. **XCHANS** may be *buffered*, in which case a successful send means the message has got into the buffer. A successful send to an unbuffered **XCHAN** means the receiving process has the message. In either case, a failed send means the message has been discarded. If sending on an **XCHAN** fails, a built-in feedback channel (the **x-channel**, which has conventional channel semantics) will signal to the sender when the channel is ready for input (i.e., the next send will succeed). This **x-channel** may be used in a **select** or **ALT** by the sender side (only input guards are needed), so that the sender may passively wait for this notification whilst servicing other events. When the **x-channel** signal is taken, the sender should send as soon as possible – but it is free to send something other than the message originally attempted (e.g. some freshly arrived data). The paper compares the use of **XCHAN** with the use of output guards in **select/ALT** statements. **XCHAN** usage should follow a design pattern, which is also described. Since the **XCHAN** never blocks, its use contributes towards deadlock-avoidance. The **XCHAN** offers one solution to the problem of overflow handling associated with a fast producer and slow consumer in message passing systems. The claim is that availability of **XCHANS** for channel based systems gives the designer and programmer another means to simplify and increase quality.

Keywords. Channels, synchronous, asynchronous, buffers, overflow, flow control, CSP.

Introduction

With the advent of the Go programming language [1], channel communication based on the CSP paradigm [2] again seems to have a potential to becoming mainstream. A previous attempt was with the occam programming language [3-5], which gained significant industrial traction during the 1980s and early 1990s (and, of course, is still being developed and applied in academic research [6-8]). Whether new languages with concurrency based on CSP repeat this success remains to be seen.

Nevertheless, channels come in more flavours than the simple channels of occam. This paper suggests a new type of channel that could be added to CSP libraries or become a new primitive in future versions of CSP-based languages. We call this channel an **XCHAN**, which contains a communication channel and a built-in *channel-ready-channel* (the **x-channel**) for flow control. An **XCHAN** may be sent to (asynchronously) and received from, but the sender must listen on the **x-channel** (usually in an **ALT/select**) when sending fails. An **XCHAN** may be buffered.

¹ The author works with concurrent software for fire detection systems, but this "industrial paper" does not necessarily reflect views taken by the company.

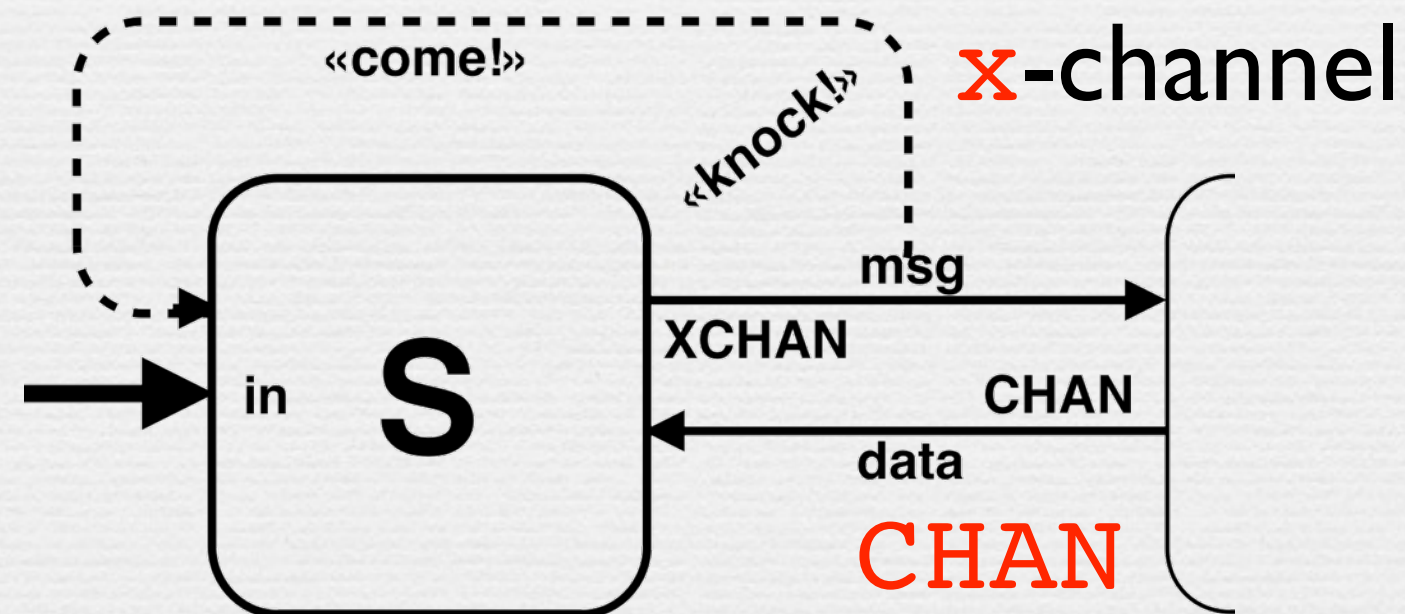
² A UTC Fire & Security company, see <http://www.autronicafire.com>.

Ville ha løst faktisk behov!

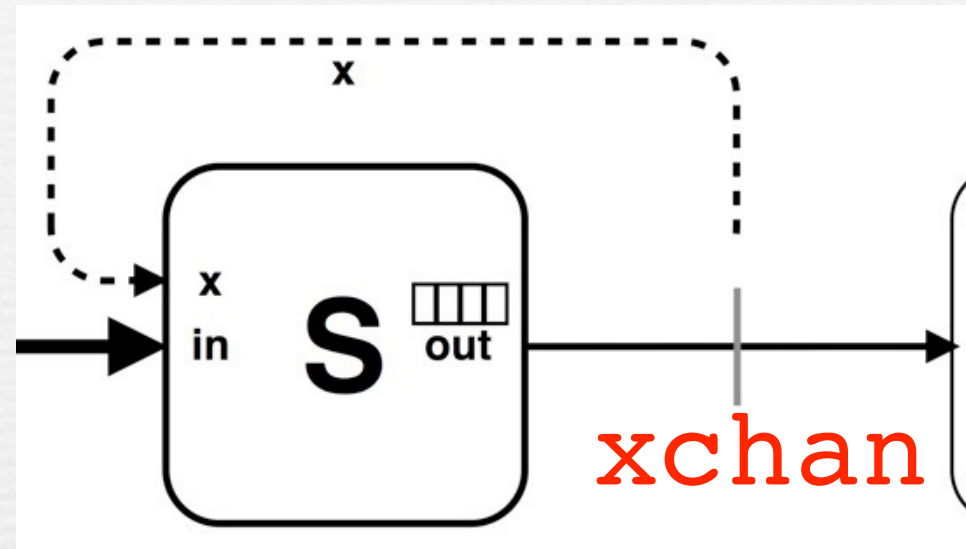
Bakgrunn for XCHAN

- Fra diskusjoner på Autronica
- Ikke implementert..
- Målet for meg var å forene **a**synkron og synkron tankemønster og tradisjon
- ..og få en felles metodikk
- Slik at det kanskje kunne bli lettere å komme i overensstemmelse med SIL og bli godkjent i henhold til IEC 61508 standard for sikkerhetskritiske system

XCHAN =
x-channel + CHAN



Rask produsent, treg konsument og XCHAN



Når Server S ikke kan bli kvitt disse dataene,
kan den fremdeles akseptere nye data (ikke «blokkere»),
og til slutt sende nyere data

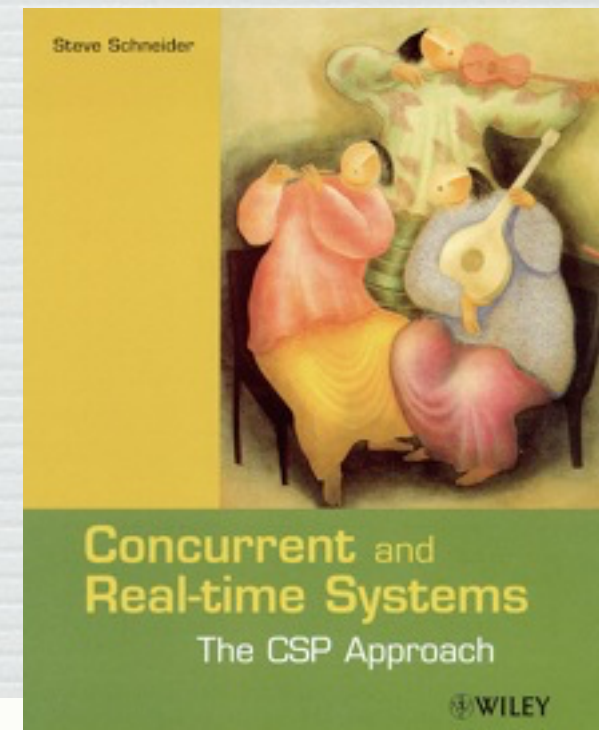
Example 7.4.1 (Buffers) The property of being a buffer of type T is expressible in a process-oriented way, by means of a mutual recursion. Internal choice is used to describe the various possibilities:

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:

XMOS

<http://www.xmos.com/products/why/determinism>

- Completely timing
- Zero jitter
- Language C, C++ and
- Single cycle instruction execution
- No interrupts
- No cache
- Multiple cores allow concurrent independent task execution
- **Hardware scheduler**
- Timing-aware development and simulation tools

XMOS

<http://www.xmos.com/products/why/determinism>

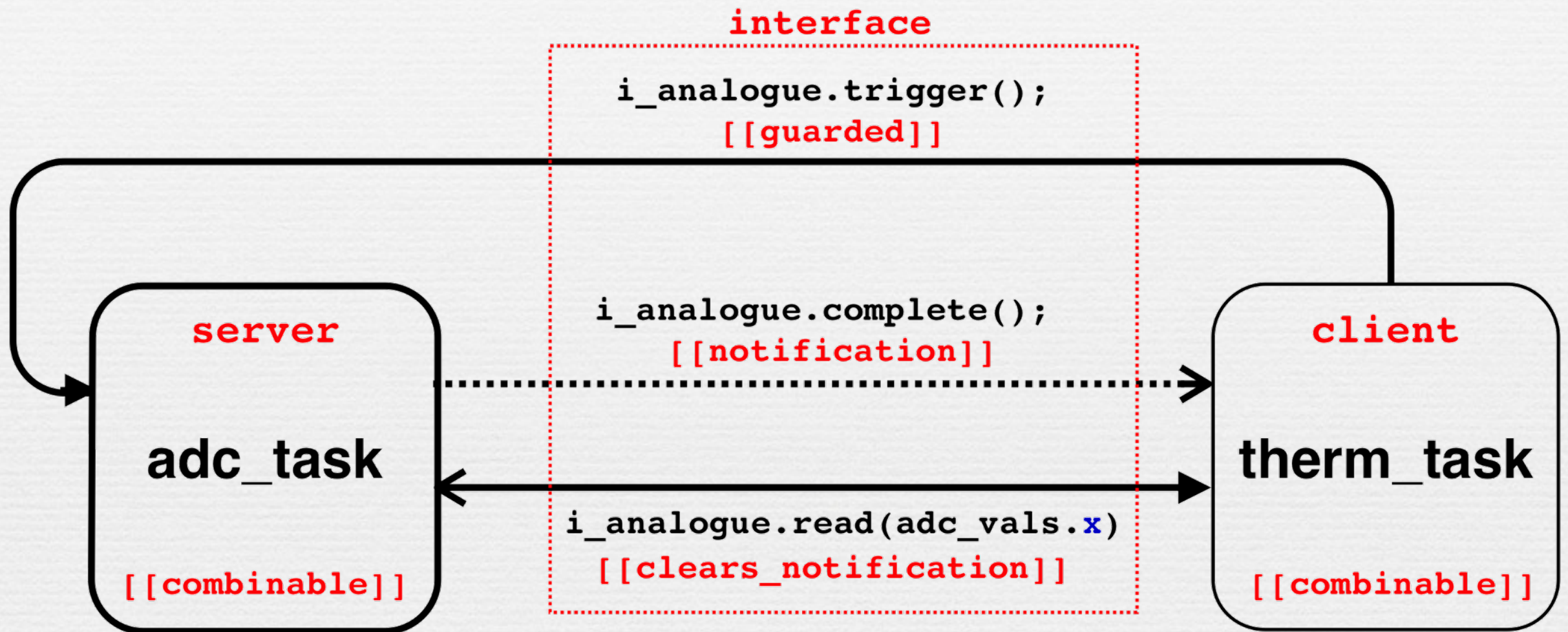
- Completely timing
- Zero jitter
- Language C, C++ and
- Single cycle instruction execution
- No interrupts
- No cache
- Multiple cores allow concurrent independent task execution
- **Hardware scheduler**
- Timing-aware development and simulation tools

par, interface, channel, port

«Kjøresystem III»

Gjør dette hjemme!

xC example*



Drawing by Øyvind Teig

Vranglås ikke mulig!
Garantert tidsrespons!

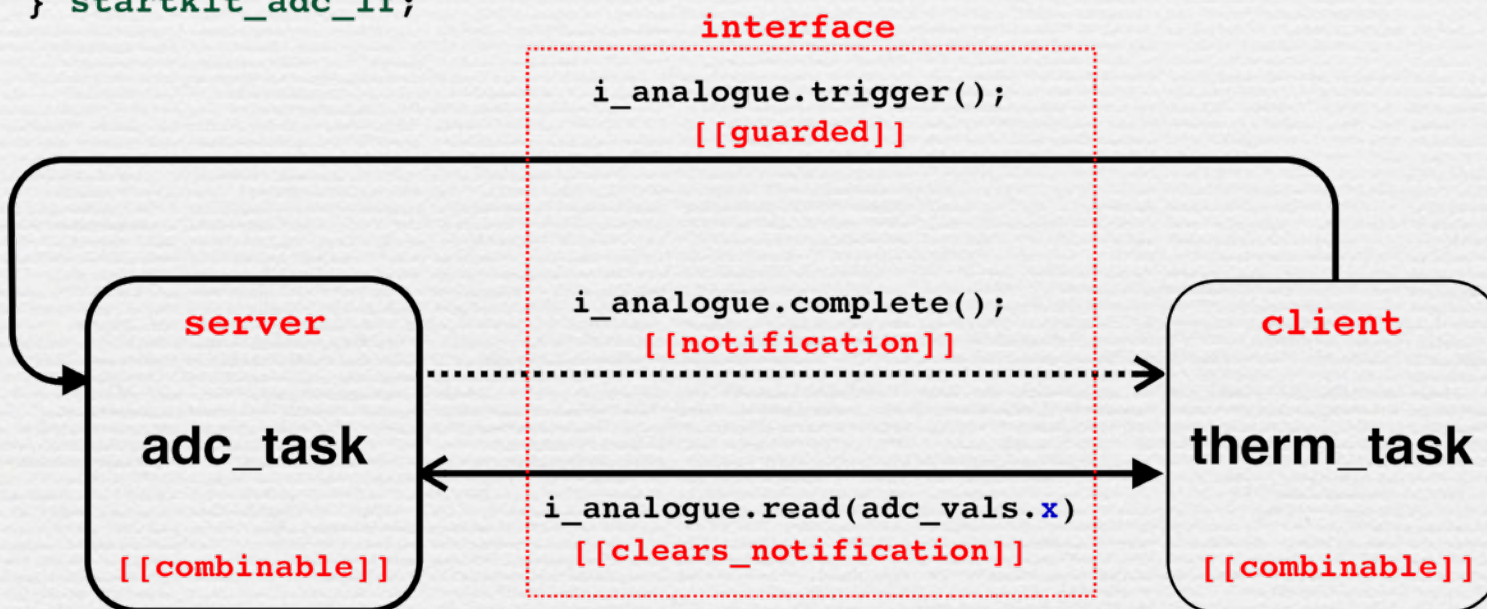
*XMOS

Gjør dette hjemme!

xC example

```
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;
```

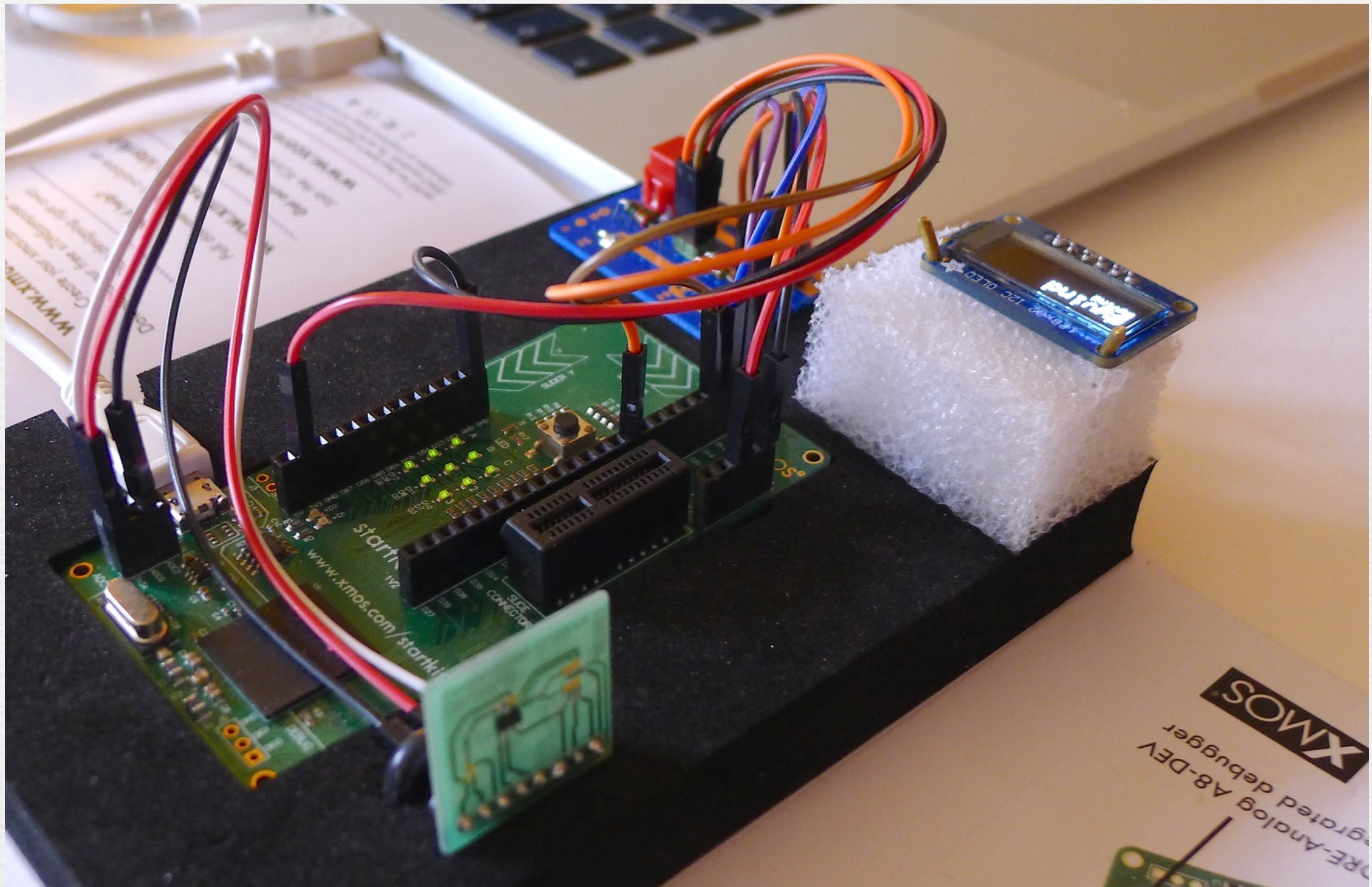


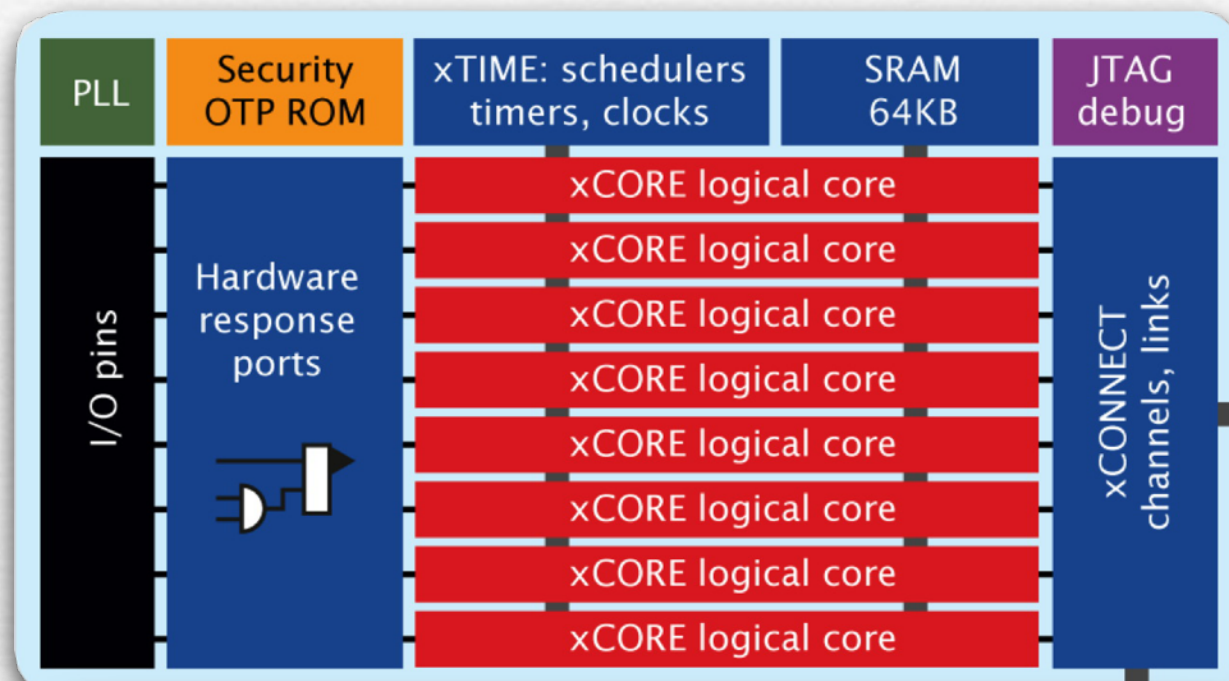
```
[[combinable]]
void therm_task
// ...
while(1) {
    select {
        case wait_for_button => c_button_2 :> int x: {
            // ...
            i_analogue.trigger();
            break; }
        case wait_for_adc => i_analogue.complete(): {
            // ...
            if (i_analogue.read(adc_vals.x))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

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

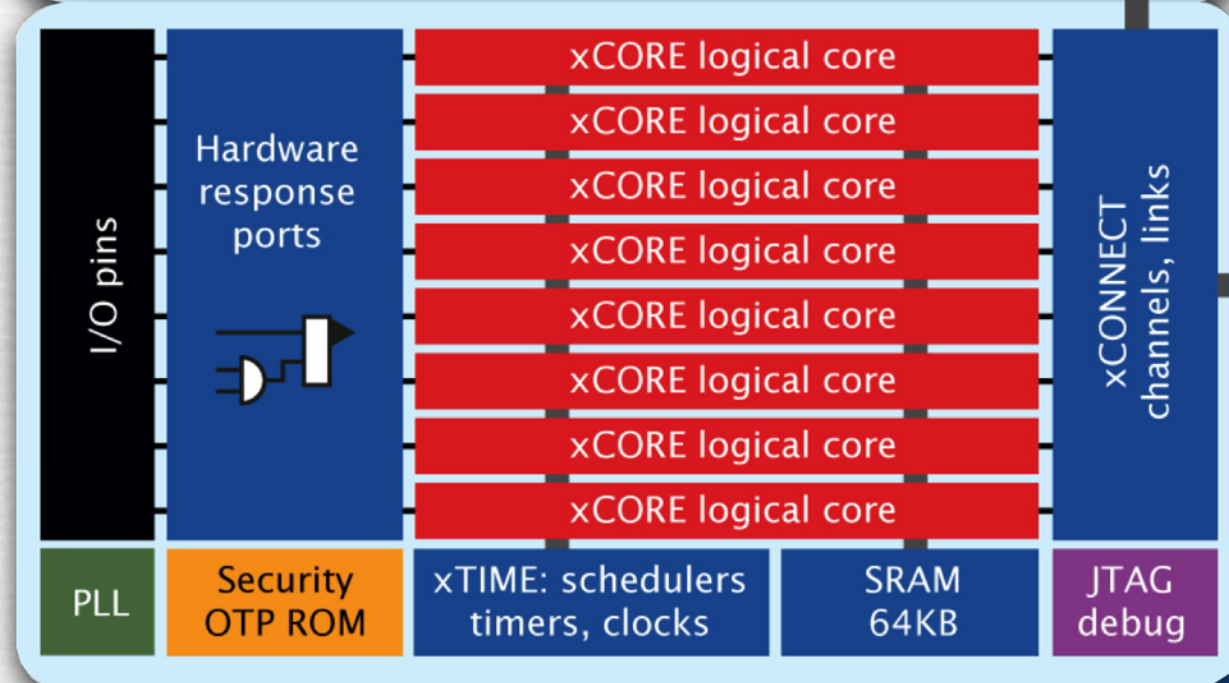
Drawing by Øyvind Teig

Hjemmeprosjekt: akvariumstyring

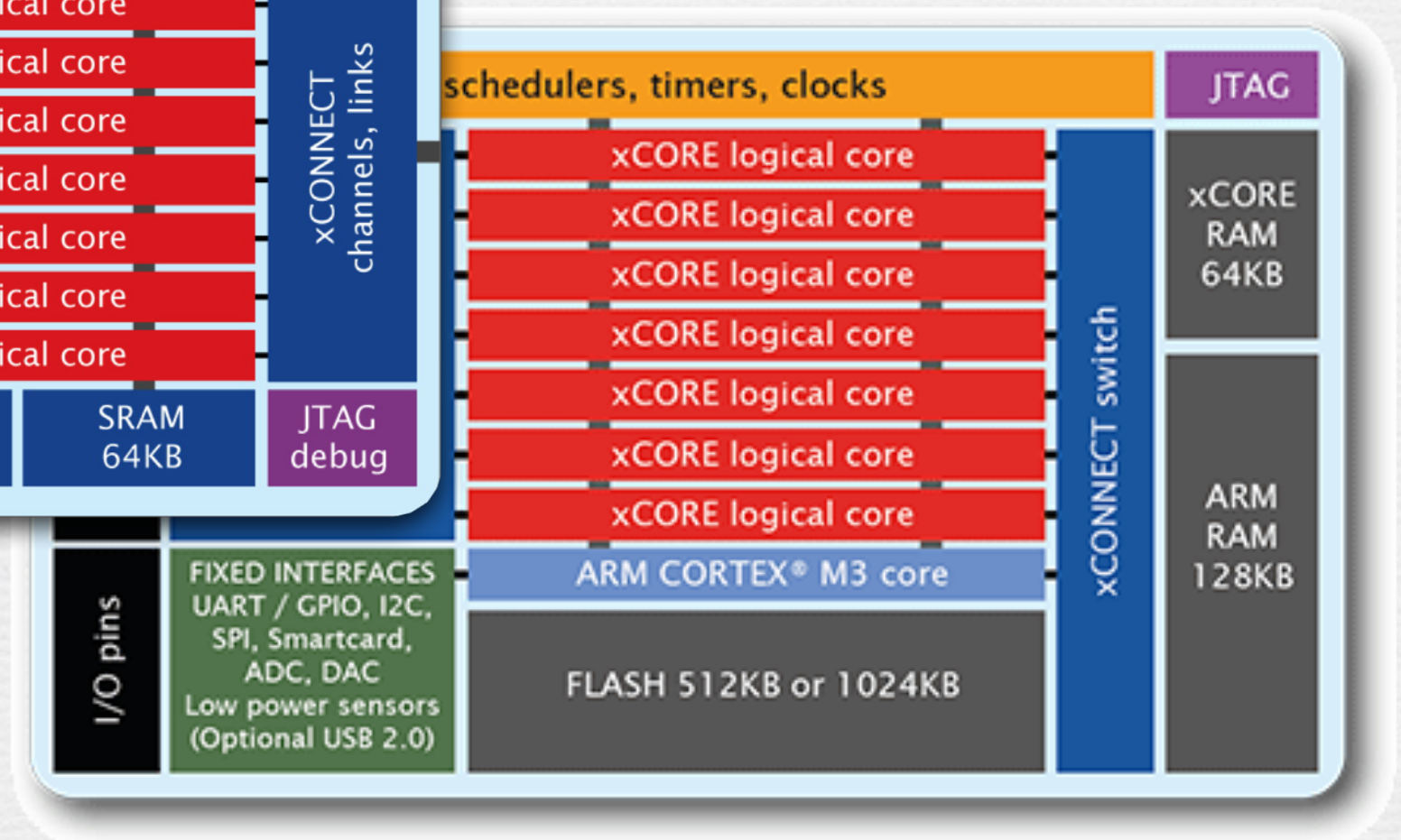




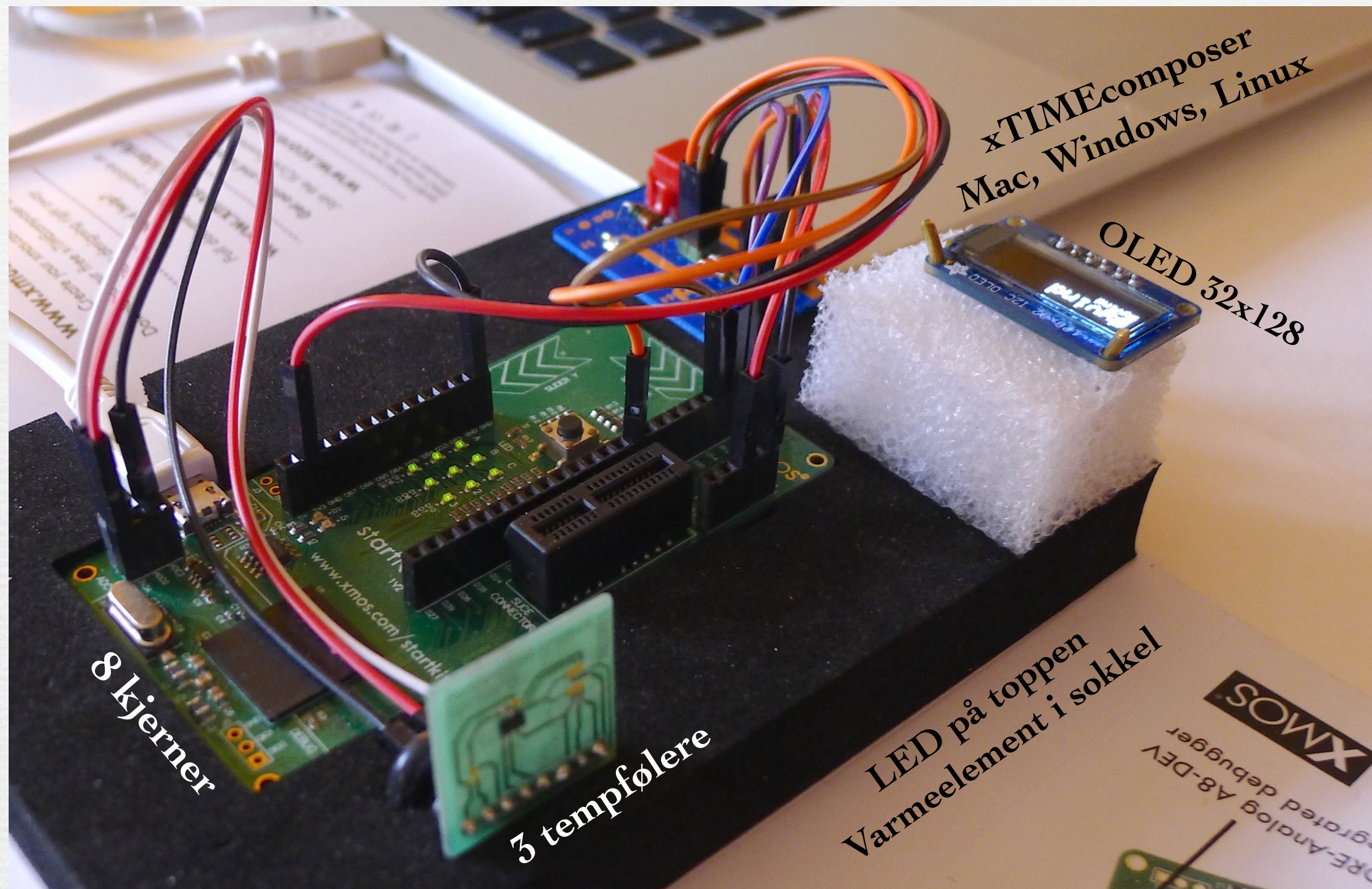
8 xCORE
på startKIT-kort



7 xCORE + ARM M3
på xCORE-XA-kort



Hjemmeprojekt: akvariumstyring



15\$ XMOS startKIT! (~~Arduino, Raspberry Pi~~)

CPA-2013

En vill ide!?

Nytt
kanalmønster,
eksempel #2 på
«høyere ordens»
kanalbruk

Selective Choice ‘Feathering’ with XCHANs

Øyvind TEIG¹

Autronica Fire and Security AS², Trondheim, Norway

Abstract. This paper suggests an additional semantics to **XCHANs**, where a sender to a synchronous channel that ends up as a component in a receiver’s selective choice (like **ALT**) may (if wanted) become signaled whenever the **ALT** has been (or is being) set up with the actual channel *not* in the active channel set. Information about this is either received as the standard return on **XCHAN**’s attempted sending or on the built-in feedback channel (called **x-channel**) if initial sending failed. This semantics may be used to avoid having to send (and receive) messages that have been seen as *uninteresting*. We call this scheme *feathering*, a kind of low level *implicit* subscriber mechanism. The mechanism may be useful for systems where channels that were not listened to while listening on some *other* set of channels, will not cause a later including of those channels to carry already declared uninteresting messages. It is like not having to treat earlier bus-stop arrival messages for the wrong direction after you sit on the first arrived bus for the correct direction. The paper discusses the idea as far as possible, since modeling or implementation has not been possible. This paper’s main purpose is to present the idea.

Keywords. channels, synchronous, asynchronous, buffers, overflow, flow control, CSP, modeling, semantics, feathering

Introduction

The idea of the suggested semantics appeared after reading Tony Hoare’s lecture “Concurrent programs wait faster” from 2003 [1]. After some pondering of a situation described there, we saw that it might be viable to use **XCHAN** [2] as a vehicle for a secondary problem not mentioned in Hoare’s lecture. The problem is how to avoid having to relate to information of busses that would potentially stop at a bus stop but heading in the wrong destination. It was believed that this could map to a new software pattern: *feathering*.

The word *feathering* is used like “turning an oar parallel to the water between pulls” [3]. Metaphorically a pull is like an **ALT** selective choice. The oar is pushed into the water at one place: only one channel is taken. But we can hear the oar whip the top of the small waves on its way saying “was there, but not interested”. So we take the step to name barely touching the small waves as *feathering*.

The author is not aware of **XCHAN** having been implemented in any language or run-time system. The time when a “channel was a channel” seems to be over; the plethora of channel types and channel usage seems to increase. Worth mentioning here is Go’s channel usage and semantics [4], which are quite different from what we are used to in the **occam** tradition. The **XCHAN** is here perceived as being an expansion of the traditional channel (**CHAN**) type.

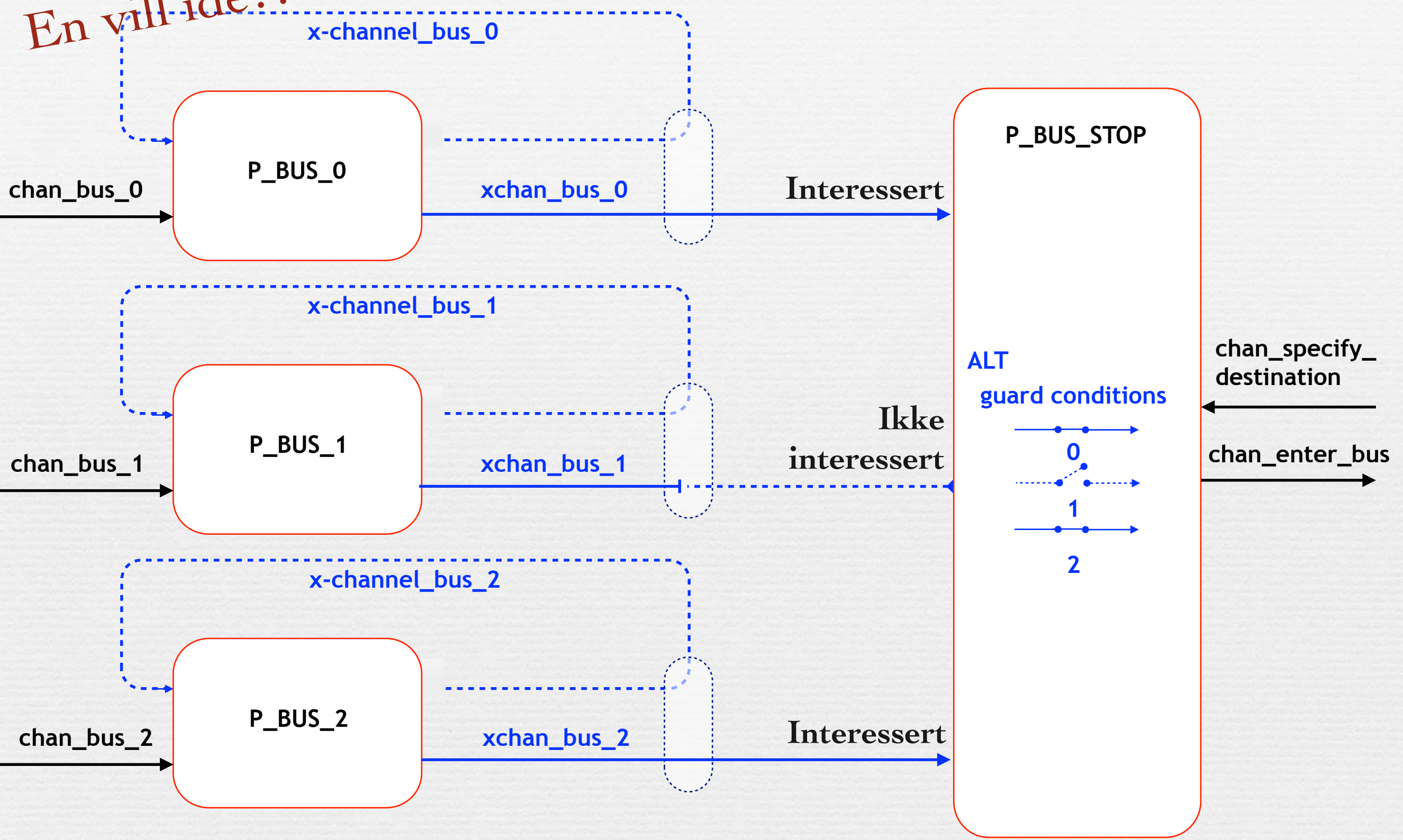
This *feathering* pattern has not been formally modeled or proven. The paper informally introduces the ideas. No literature search has been done to find similar or equal ideas.

¹ The author works with concurrent software for fire alarm systems, but this “industrial paper” does not necessarily reflect views taken by the company. See 57 www.teigfam.net/oyvind/work/work.html

² A UTC Fire & Security company. <http://www.autire.com>

FEATHERING: Implisitt abonnementsmønster

En vill ide!?



Noen objektmodeller

- C funksjoner (?)
- C++ objekt
- Go package
- -> task, thread, process (mer egnet for neste side?)

Noen prosessmodeller

- CSP-type prosess (kanaler etc.)
 - Go goroutine (med kanaler)
 - JCSP, PyCSP, node-csp
 - xC par
- Meldingsdrevet (postverk)
- Event-drevet
- Single-threaded (with callbacks og derfor «ikke-blokkerende», som de sier)
 - node.js, DOM etc.

Noen komponentmodeller

- Statisk API
- API med parameter-datakunnskap («dataflyt»)
- OO objekter (arv..)
- Meldings-sekvens-beskrivelse (MSC)
- Registrering av «svar til» (callback)
- Full MSC med timing og semantisk modell(feilfri pluggbarhet)

Rob Pike: “A Concurrent Window System”

<http://swtch.com/~rsc/thread/cws.pdf> (1989)

Sammendrag fra bloggen

Pike & Sutter: Concurrency vs. Concurrency

- Han viser et vindussystem skrevet i et CSP-basert språk som er mye enklere..
- ..enn tradisjonelle mus-hendelsesdrevne system
- «State machines are powerful but inconvenient»
- Forlanger «concurrency»-tenking fra start til slutt!
- Anbefalt lesestoff!

Russ Cox: “Bell Labs and CSP Threads”

<http://swtch.com/~rsc/thread/>

Sammendrag fra bloggen

Pike & Sutter: Concurrency vs. Concurrency

- Pipes oppfunnet av Doug McIlroy i 1964
- Doug McIlroy ønsket å lage kanaler og selective choice
- Det ble «filter-and-pipeline» og Unix «select»
- Dette var *før* Hoares artikkel (1978, 1985)
- I 1980 bygde Holzmann og Rob Pike pan (=Promela) som tar en CSP dialekt som input
- -> Google's Go
- Anbefalt lesestoff!

<

>

📖

🏠

teigfam.net

A

A

🕒

☁

⋮

📄

📄

+

Øyvind Teig

Some of my blog notes

HOME

BØKER

PUBLICATIONS

TECHNOLOGY

MODELS

HOBBY

PUBLISHING

ETC.



MY XMOS NOTES

WHAT, A SQUARE HOLE IN THE HULL! NONE ROUND?

HOW MUCH CONCURRENCY?

NOT SO BLOCKING AFTER ALL

PIKE & SUTTER: CONCURRENCY VS. CONCURRENCY

ATOMIC FOR ALL?

CSP ON NODE.JS AND CLOJURESCRIPT BY JAVASCRIPT



I am "aclassifier"

My user name "aclassifier" (or "Aclassifier") originally describing (classifying) my first-generation short- The result of this attempt may be seen at [www.teigfam.net](#) became *aclassifier*. And I have used that name where has this information been *classified*!

My "classic" home page (since 2000) is at [www.teigfam.net](#) here the action is

collecting and scale models. So *A-Class'ifiers* one. Nowhere

Background

Bøker

etc.

Leiebil

Search

men..

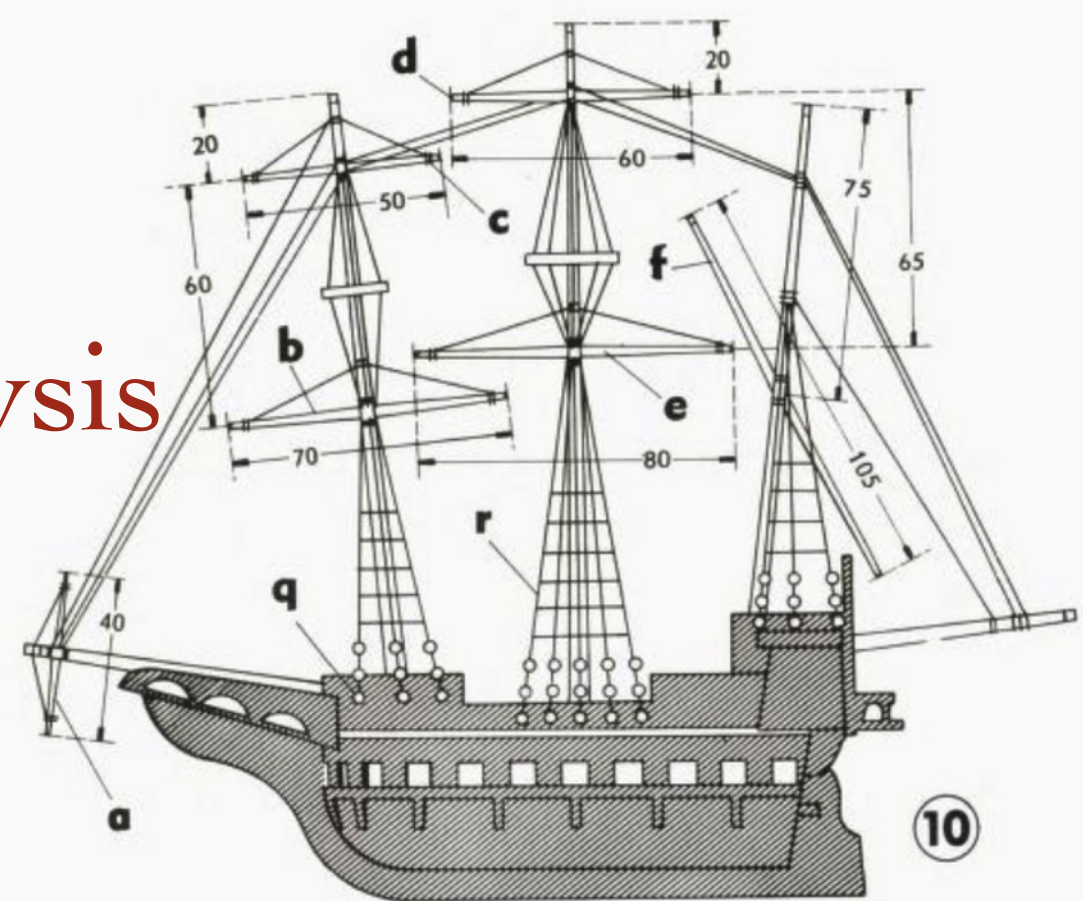
så mye kan vi vel forandre verden?

- Ved å sørge for at vi lager ting så bra at vi kan ta det fram igjen etter fem år og syns at det var bra
- Ved å passe på at vi har beveget oss så mye på de fem årene at vi tør si at vi ikke hadde gjort det akkurat slik nå

RRCA..

Relentless Root Cause Analysis

- Ganske ny prosedyre for
- Alvorlig
- Men,.. er det fler?
- (Akseptabel?) relativ risiko = sannsynlighet * konsekvens * ikke-deteksjonsgrad
- Verktøy
 - FMEA/FMECA (Failure mode and effects (and criticality) analysis (FMEA/FMECA))
 - Feiltreanalyse



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.

RRCA..

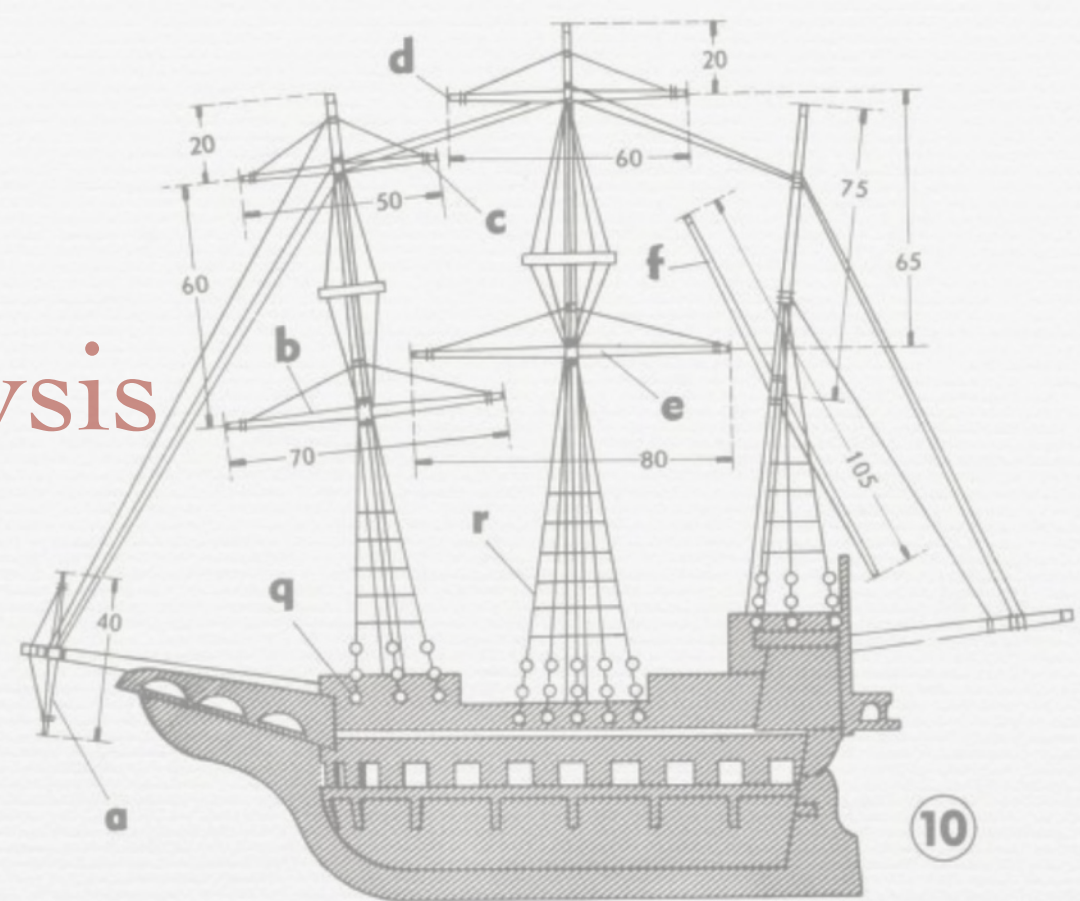
Relentless Root Cause Analysis

- Ganske ny prosedyre for
- Alvorlig
- Men,.. er det fler?
- (Akseptabel?) relativ risiko = sannsynlighet * konsekvens * ikke-deteksjonsgrad
- Verktøy

- FMEA/FMECA (Failure mode and effects (and criticality) analysis (FMEA/FMECA))
- Feiltreanalyse

**Huff, dette kan være smertefullt!
(Krever klokskap fra de andre!)**

<http://www.teigfam.net/oyvind/home/technology/096-what-a-square-hole-in-the-hull-none-round/>



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,

10

Ett eneste råd:

Svart

eller

hvitt?

- *Noe*
- — og det kan vi hevde..
- men om vi har beveget oss på de fem årene
- har også

Litt dyrekjøpt kunnskap:

Svart

eller

hvitt?

- *Noe*
- —og det kan vi hevde..
- men om vi har beveget oss på de fem årene
- har også

..»det andre» trenger ikke være noe «dritt»!

Til slutt lurер jeg på

- ☐ om par blir like selvfølgelig som i f framtida?
- ☐ og om språkene i så fall sikrer rett bruk?
- ☐ Hvilket språk (etc.) vil det vise seg er nærmest?
- ☐ Og hva vil det ta med seg av det vi alt har?

Ses vi der?



The Golang UK Conference

Friday 21st August 2015

Kontaktinfo

Dette foredraget:

http://www.teigfam.net/oyvind/pub/NTNU_2015/foredrag.pdf

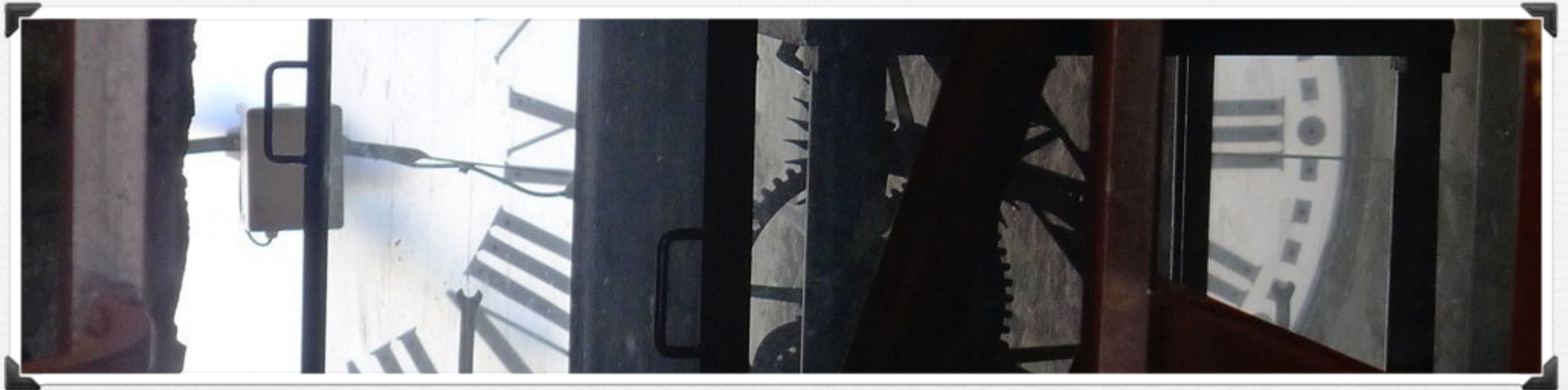
Dette faget på NTNU:

<http://www.itk.ntnu.no/fag/TTK4145/information/>

`oyvind.teig@autronicafire.no`
`oyvind.teig@teigfam.net`

Les dette og mer på:

<http://www.teigfam.net/oyvind/pub/>



- ❖ Plutselig er dere besteforeldre!
- ❖ for i 2054 er det 39 år siden 201

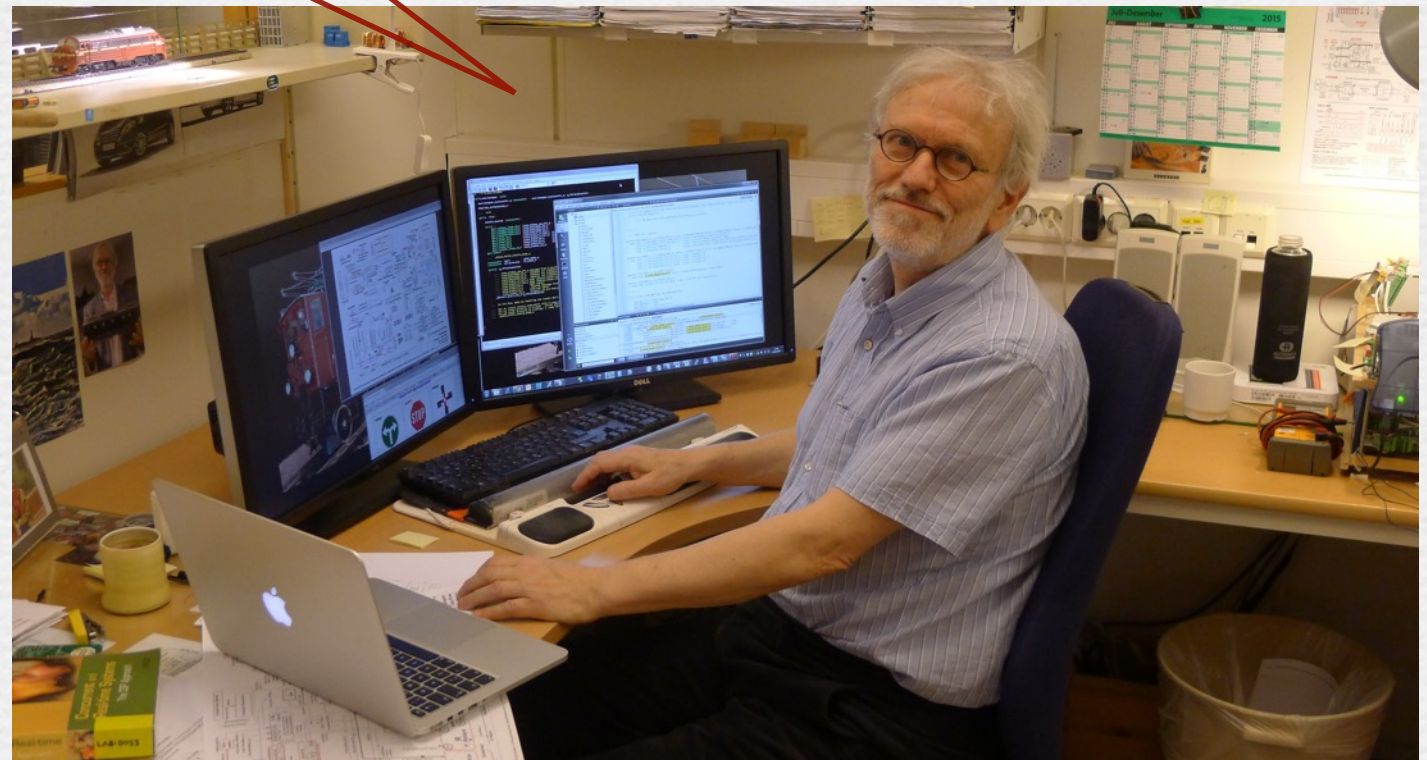


1976

- ❖ Plutselig er dere besteforeldre!
- ❖ for i 2054 er det 39 år siden 201

2015

Da har





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

takk
for meg!

