# Disposition03 Thread Communication

Det kan være svært at få tråde til at opføre sig ordenligt når de arbejder på same ressourcer, dette kan gøre via mutex og conditionals men det vil hurtigt blive uoverskueligt.

## The challenges performing intra-process communication

Hvad skal der til for at lave kommunikation mellem tråde:

* Individuelle tråde venter på at en betingelse er sand.
* De skal så gå ind i kritiske sektioner (Kan ske flere gange i en tråds loop).
* Trådene deler flere ressourcer som skal synkroniseres mellem dem. Rækkefølgen af dette kan være kritisk.

Hvad giver det så af konsekvenser?

* Det bliver en udfordring at undgå deadlocks.
* Læslighed af koden kan blive et problem er det nemt at følge hvad der sker?
* Koden bliver meget kompleks.

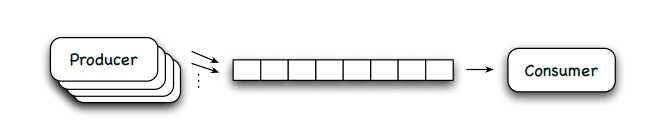
Hvad har vi så behov for?

* Alt processering i en tråd må ikke kræve at der skal låses.
* Men andre tråde skal være i stand til at vider give kontrol og/eller data til en specifik tråd.
* Flere tråde skal kunne videregive kontrol samtidig.

How do we do?

## Message queue

Denne kunne løse alle de nævnte problemer ved at gør flere tråde i stand til at sætte beskeder i kø



### Hvad er præmissen for at designe denne?

* Hvis køen er fuld skal trådene der ønske af videre give en besked blokere indtil der er plads i køen (Max antal pladser i køen).
* Hvis køen er tom skal den tråd der hente beskeder fra køen blokere
* Vi skal ikke blokere med polling og sleep, vi skal i stedet bruge conditionals det læner sig godt opad de mutex vi skal bruge på køen alligevel da vi ønsker at beskyttet denne.

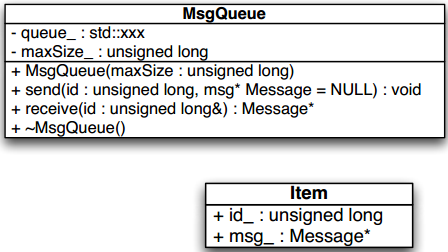
### Forskellige løsninger hvilken skal vi vælge?

Hvordan kan det så laves?

* Void pointers?
  + De kan indeholde alt
  + Men vi har ingen type information
* Templates?
  + Kunne være en god men kompleks implementation
  + Her får vi type safety
* Arv
  + Simpelt og nemt at udvide via
  + Type sikkerhed gennem basis pointer da denne kan bruges til delete
  + Kan have et overhead

Vi har brugt arv til vores system.

### Designet og implementation af denne.



Vi har en message class som alle beskeder arver fra, disse beskeder ligger ikke i køen der ligger derimod et item. Et item er en stuct som indeholder en message og et id.

Der er en send og en receive funktion det er disse der bruges til at sende og modtage data osv.

Køen bliver så kaldt fra den tråd der ejer dens dispatcher funktion der finder ud af hvilken handler der skal kaldes for at få udført den opgave beskeden påkræver.

## Impact on design/implementation between before and after the Message Queue

Før:

* Stepvis programmering vi ventede på en “besked” via conditionals
* Høj kompleksitet ved store programmer
* Stort behov for at tænke i mutex og conditionals

Efter:

* Højere abstraktion da vi ikke skal tænke på mutex og conditionals osv.
* Samme program = der sker det samme
* Vi skal nu holde styr på beskederne og deres id’er, styre hukommelse allokering og deallokering af beskeder. Handlers tager rollen for den stepvise del af programmet
* Nemmere at skalere programmet ved at udvide med nye beskeder.

## Event Driven Programming

### Basis ide

Vi ønsker at gå fra en program struktur der er sekventiel til en struktur hvor vi har et program der reagere på beskeder fra sine undersystemer som for eksempel hvis der er trykket på en knap på et keyboard.

### Reaktionsevne

Kunne kaldes reaktiv programmering vi reagere på en hændelse der er sket og vores program kan skelne mellem beskeder og der ved besluttet hvad der skal udføres alt efter hvilken besked vi modtager.

### Design fra diagram til kode

