UNIVERSITETET I OSLO

Institutt for informatikk

IN1010 våren 2023
Parallellitet og tråder – del 2
27. mars 2023

og litt om oblig 5

Stein Gjessing

Horstmann, hele kap 20

UNIVERSITETET I OSLO

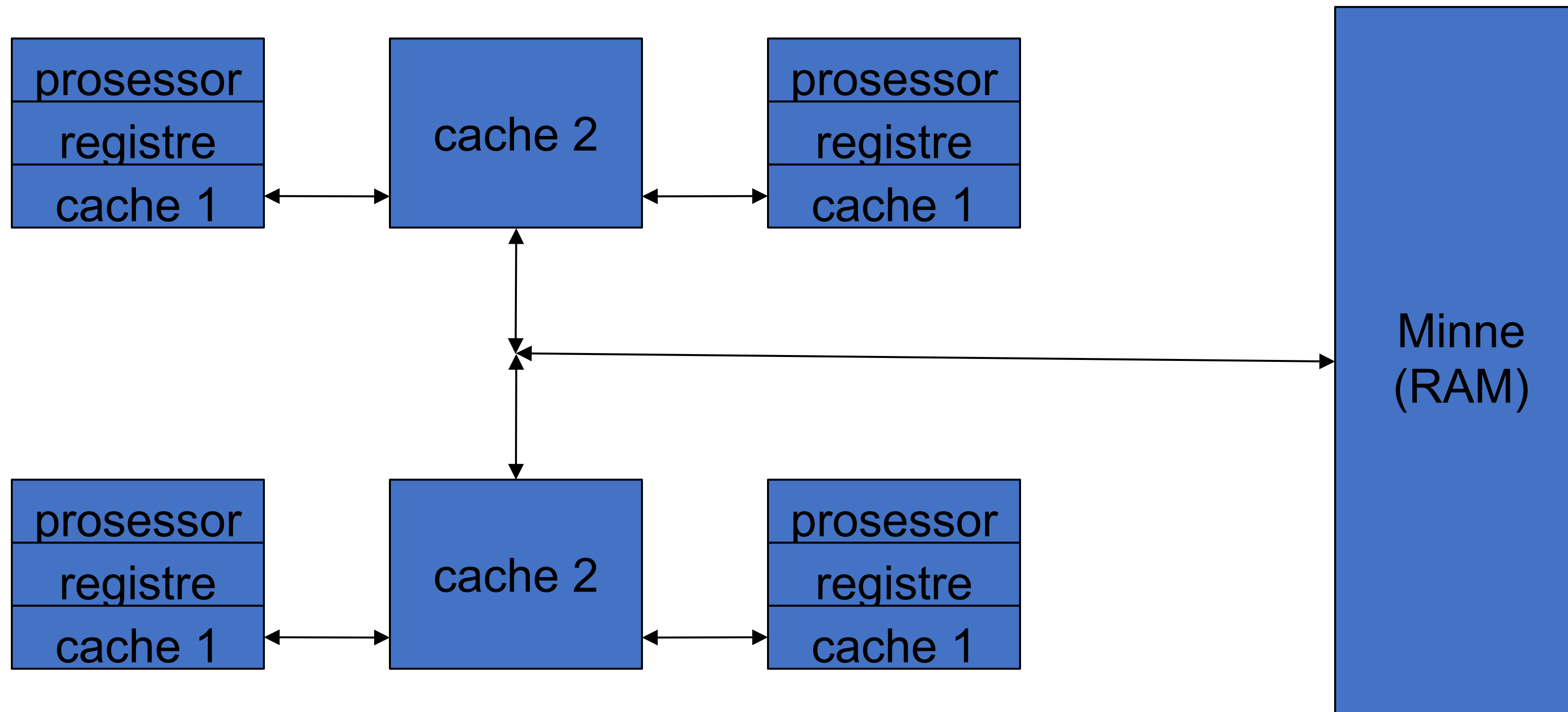
Institutt for informatikk

Agenda

- Litt mer om parallellitet, tråder og monitorer
- Noen eksempler
- Hvordan vente på at andre tråder er ferdige
- Introduksjon til oblig 5
- Vranglås (Engelsk: deadlock)

Repetisjon:

Maskinarkitektur, f.eks. 4 kjerner



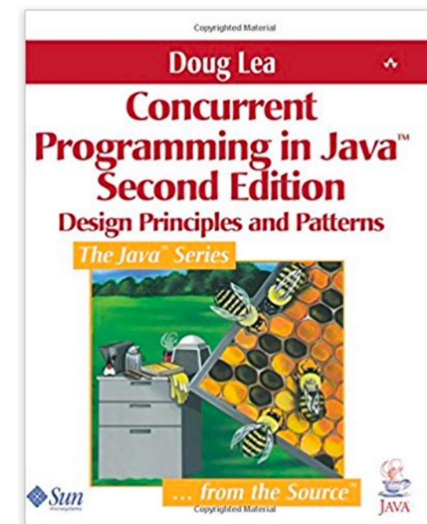
Fordelen med konstanter (immutable objects)

- Konstanter kan det aldri skrives til
- Derfor kan alle tråder aksessere de samme dataene samtidig
- Prøv å ha mest mulig konstanter når data skal deles mellom tråder.
- Eksempel: Geografiske data, observasjoner
- Hvis du ønsker å gjøre en forandring:
  - Kast det gamle objektet og lag et nytt.

Monitorer i Java

- Opprinnelig ble Java laget med et innebygget monitor-begrep (ikke pensum i IN1010):
    - Alle objekter har en lås
    - synchronized foran en metode tar låsen og gjør metoden til en kritisk region (med hensyn på data i dette objektet)
    - Se class Object (wait(), notify(), notifyAll())
    - Horstmann: Special Topic 20.2

- Doug Lea forbedret dette med java.util.concurrent-biblioteket



Java-biblioteket java.concurrent

Interface Lock:

import java.concurrent.locks.*

- void lock() Acquires the lock
- void unlock() Releases the lock.
- Condition newCondition() Returns a new **Condition** instance that is bound to this Lock instance.

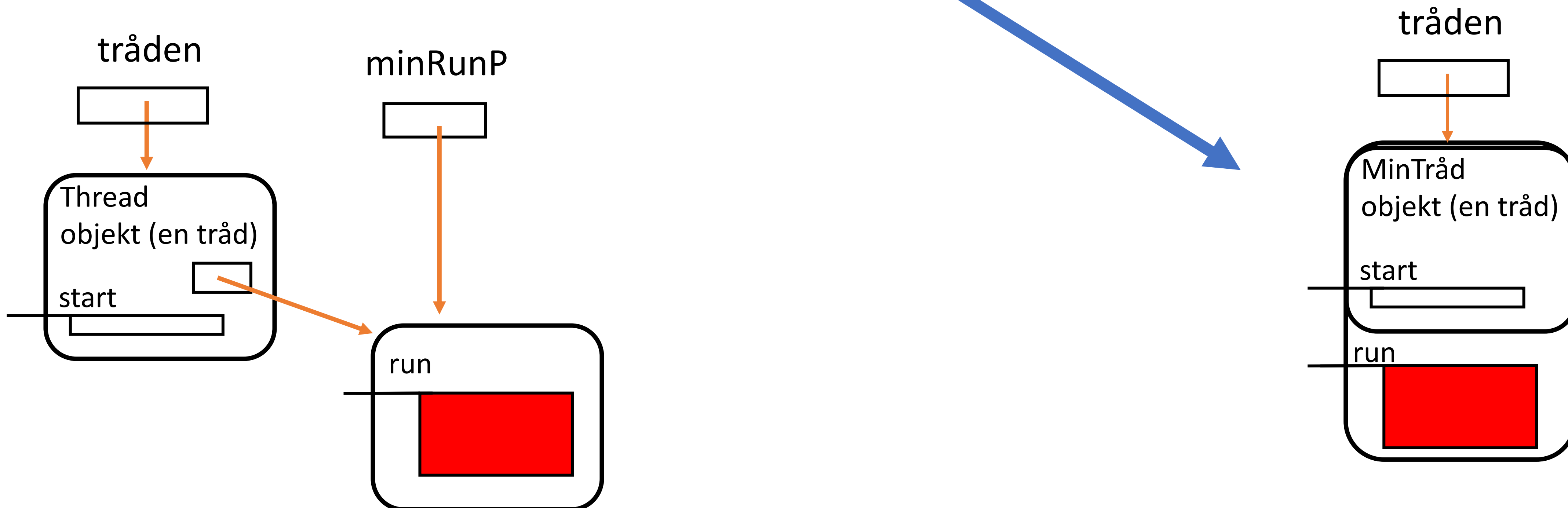
Interface Condition:

- void await() Causes the current thread to wait on this Condition until it is signalled (or interrupted)
- void signal() Wakes up one waiting thread (on this Condition).
- void signalAll() Wakes up all the waiting threads (on this Condition)

- class ReentrantLock implements Lock

Alternativ for tråder
Ikke pensum i IN1010

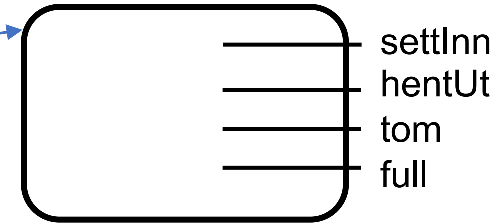
- Lage tråder ved hjelp av **subklasser** istedenfor delegering *
  - Horstmann: Programming Tip 20.1



* Vi burde brukt mer tid på subklasser vs. delegering I IN1010

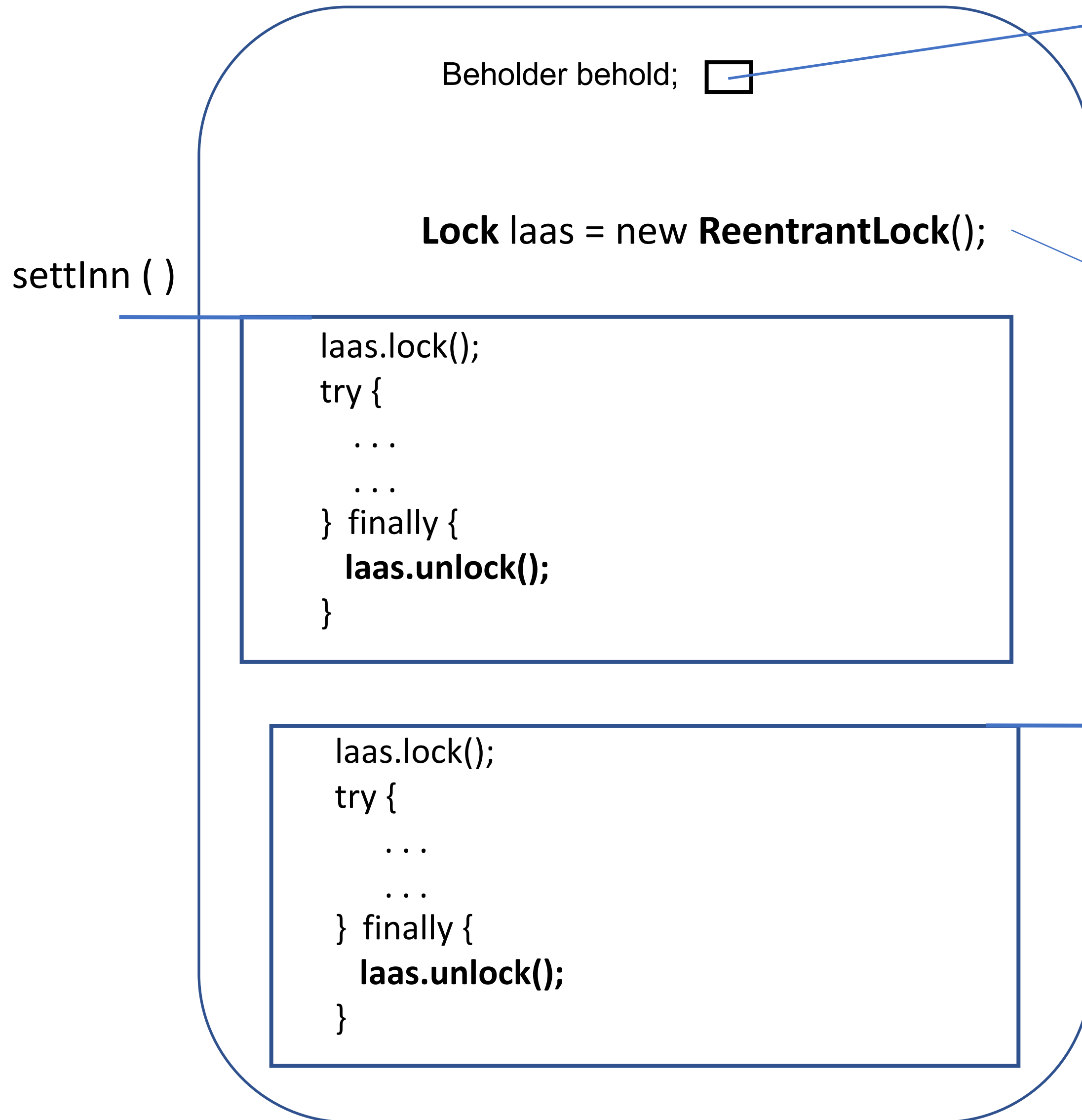
Konsistente data i kritiske regioner

Felles data:
Objekt av klassen Beholder

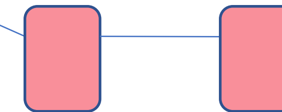


Institutt for informatikk

Komposisjon



Dette er **komposisjon**
Beholderen kan også skrives direkte inne i monitoren



Tråder som venter på låsen for å få komme inn første gang

En monitor (et objekt)

Data i kritiske regioner

- Java sørger for at alle tråder ser de same dataene inne i kritiske regioner
  - Ved utgangen av en kritisk region skrives alle data tilbake til primærlageret (RAM)
  - Alle trader som går inn i en kritisk region laster alle felles data opp fra primærlageret

- Så bruk av kritiske regioner er ikke gratis
- Men det er bedre at programmet er riktig enn at det er feil og raskt 😀

Volatile variable

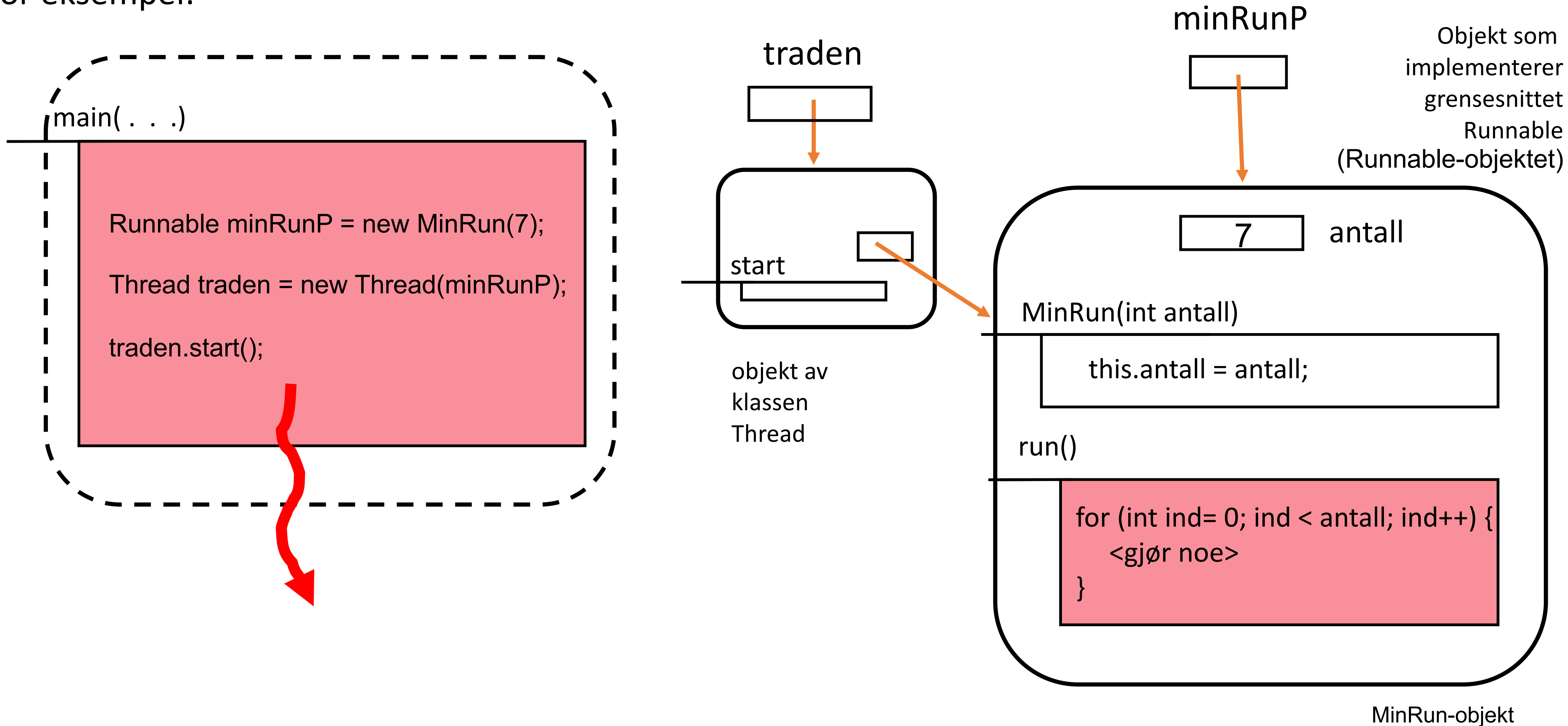
En variabel som er deklarert
volatile caches ikke
(og oppbevares ikke i registre
(lenger enn helt nødvending)).

En volatil variabel skrives helt
tilbake til primærlageret og hentes
opp derfra når den skal brukes.

```
boolean stopp = false;

//      En tråd:
stopp = true;

//      En annen tråd:

while(! stopp) {

 }

//       Må da deklarere:

volatile boolean stopp=false;
```

Parametre til tråder – Bruk konstruktøren til «Runnable-objektet»

For eksempel:



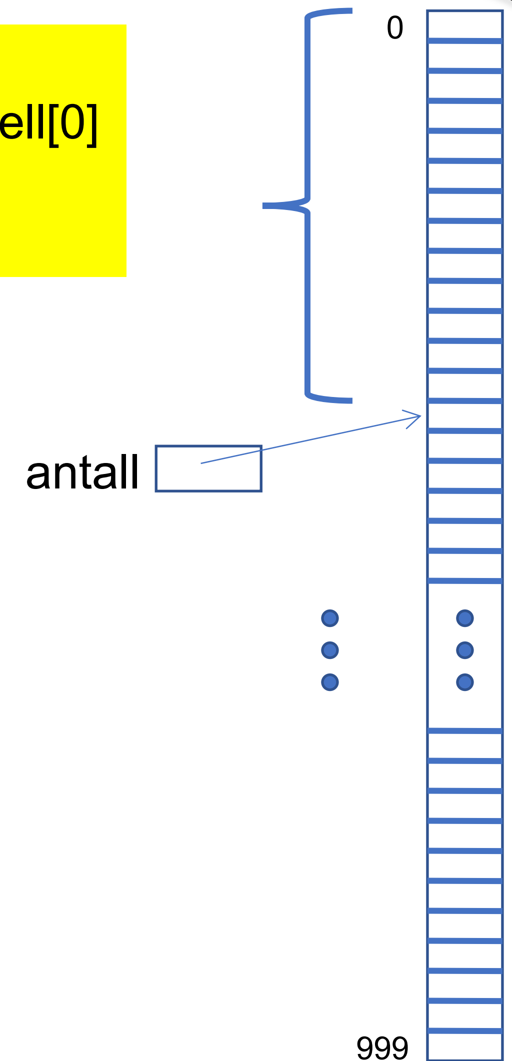
Husker dere: Invarianter på data i objekter

Uten
tråder

Invariant:
Alle dataene vi lagrer ligger i tabell[0]
til og med tabell [antall – 1] og
0 <= antall <= 1000

```
settInn(x) {
     if (antall == 1000) return ;
     antall ++;
     tabell[antall-1] = x;
}
```

```
taUt ( ) {
     if (antall == 0) return null;
     antall --;
     return (tabell[antall]);
}
```

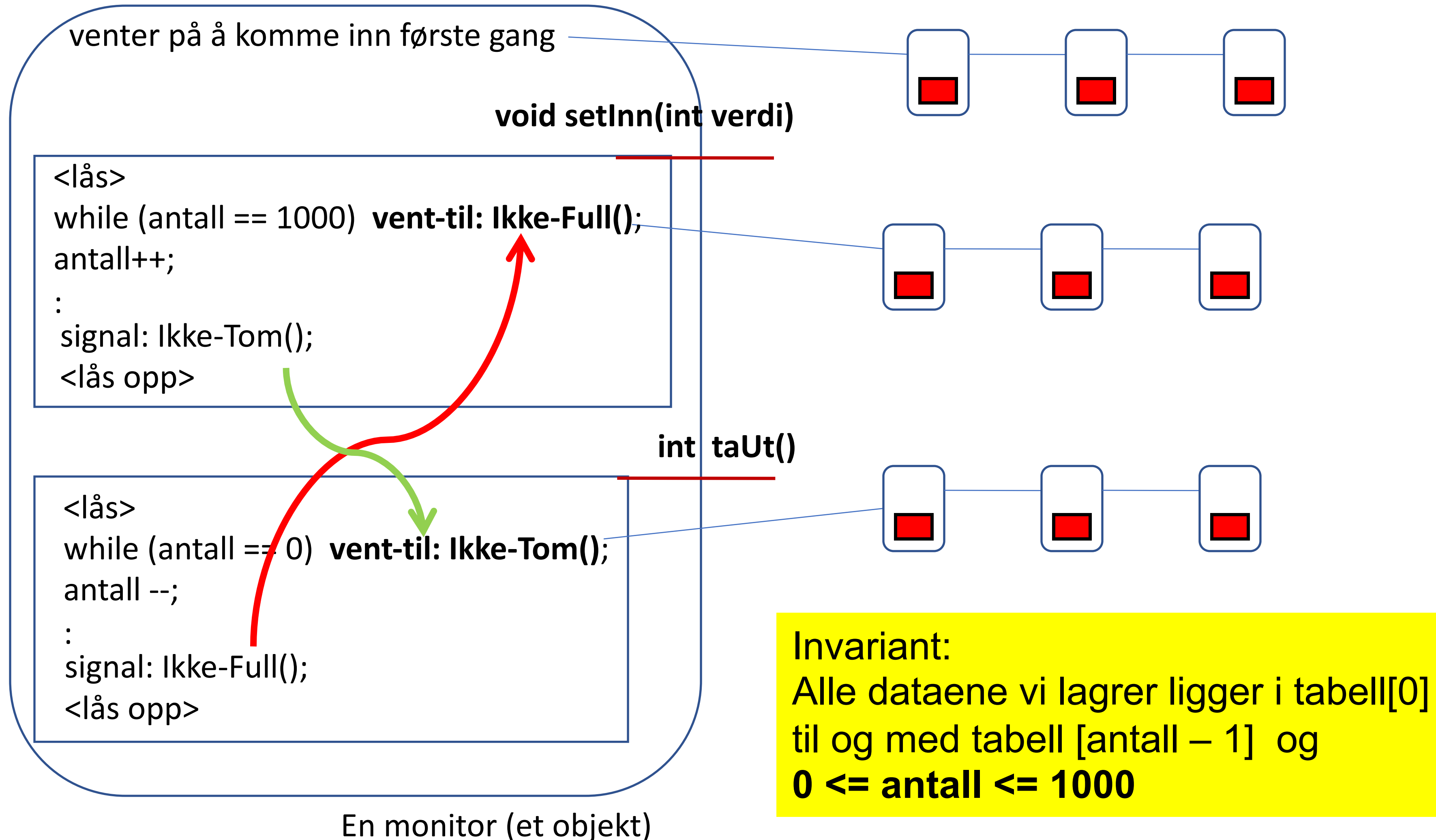


UNIVERSITETET I OSLO

Vi tar opp tråden (☺) fra forelesningen om invarianter på data I objekter:

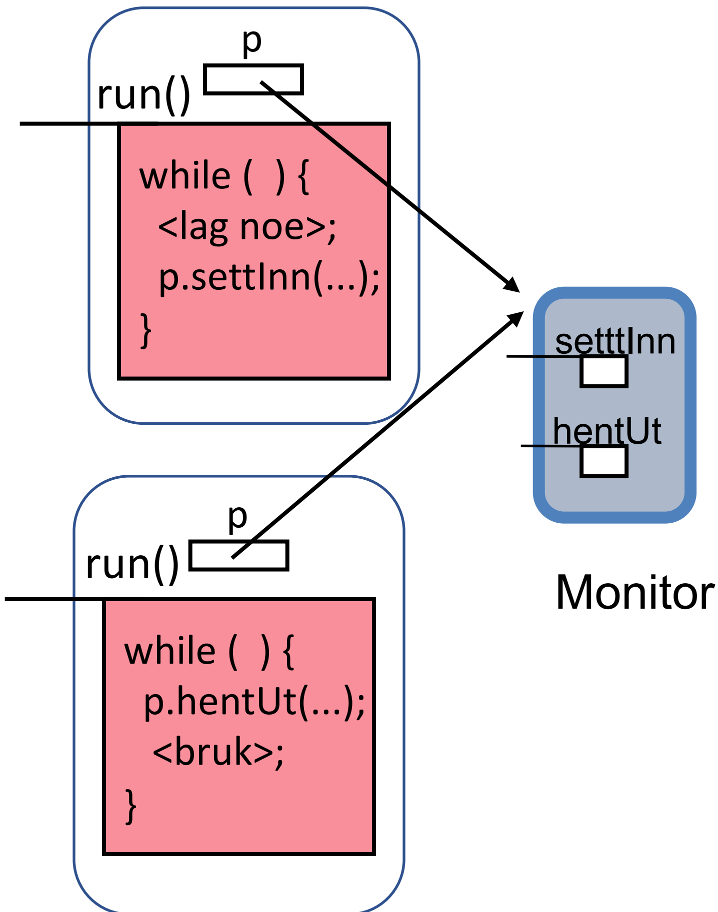
Institutt for informatikk

Hvordan bevare invarianter på data i objekter når vi ikke har ansvaret alene. Svar: *Vi venter ofte på at andre skal gjøre objektets (monitorens) tilsand hyggeligere.*



Invariant:
Alle dataene vi lagrer ligger i tabell[0] til og med tabell [antall – 1] og
0 <= antall <= 1000

Huskeregler for tråder



NB! Trådene (run()-metodene) kaller metodene i monitorene som om monitoren var et vanlig objekt.

Ingen låsing, venting eller signalering i trådene utenom monitorene!

Alle låsing, venting og signalering skjer i inne monitorene

Men en tråd kan si Thread.sleep().
Og en tråd kan kalle interrupt() og join()* i andre tråder

*join() kommer snart

Hva skjer når run() kaller andre metoder

f.eks. to tråder:

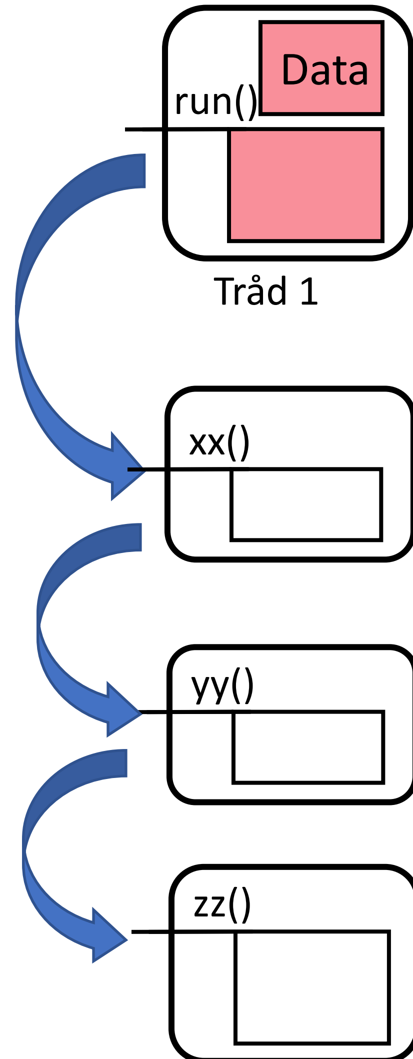
En stabel/stakk av metodekall:

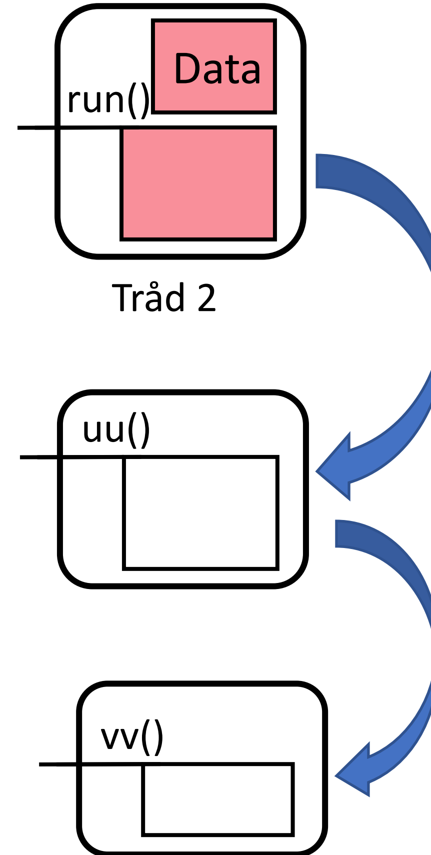
run() i tråd 1 kaller

metoden xx() som kaller

metoden yy() som kaller

metoden zz() som . . .





En stabel/stakk av metodekall

run() i tråd 2 kaller

metoden uu() som kaller

metoden vv() som . . .

Hver eneste tråd har sin egen stabel av lokale variabler i metodekall ("the call stack")

Lokale variable i metoder (uavhengig av tråder)

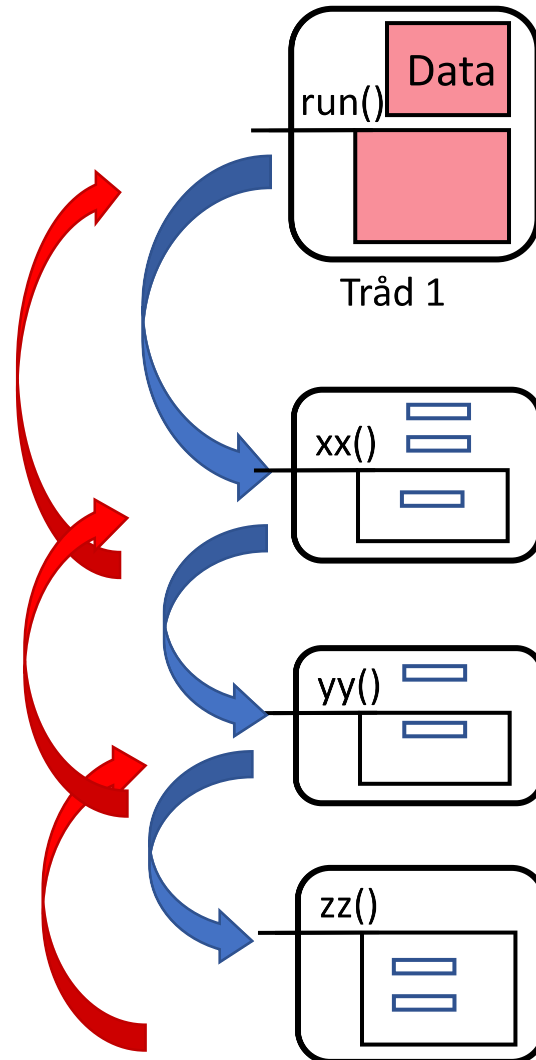
En stabel/stakk av metodekall:

run() i tråd 1, som terminerer.

metoden xx(), som returnerer til . .

metoden yy(), som returnerer til . .

metoden zz() returnerer til . .



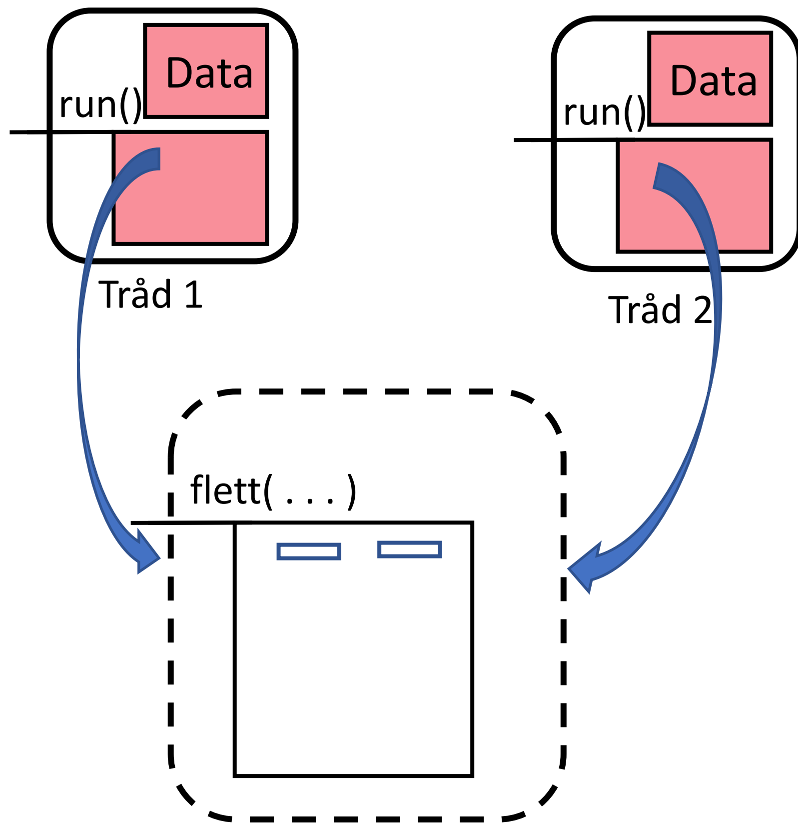
Hver eneste tråd har sin egen stabel av lokale variabler i metodekall ("the call stack")

Variabler lokale i metoder legges på kall-stakken, tilhører metodeinstansen og kan ikke akseseres av andre trådere

Instansvariable (oftest private) er felles for alle metoder i objektet og kan følgelig aksesseres av alle metoder og alle tråder som kaller disse metodene

Dette er ikke noe nytt med tråder, slik har det vært fra først forelesning den 23. januar (og det samme i Python og alle andre programmeringsspråk).

Oblig 5 og bruk av samme metode



Om du programmerer å slå sammen HashMap-er * i en metode som bare bruker lokale variable, så kan alle tråder som vil kalle og bruke denne metoden samtidig gå i ekte parallell.

Derfor kan en static-metode brukes av alle trådene dine uten å gå i bena på hverandre.

Fordi hver eneste tråd har sin egen stabel av lokale variable i metodekall ("the call stack")

* Dette vil du skjønne når du har lest obligteksten

UNIVERSITETET I OSLO

Institutt for informatikk

Generelt om funksjonell programmering og parallellitet

- Funksjonell programmering betyr at all kode består av kall på funksjoner / metoder
- To paradigmer
  - Imperativ programmering (Python, Java, . . .)
    - Imperativ: Gjør noe, skritt for skritt
  - Funksjonell programmering (Lisp, . . .)
- Parallell programmering er naturlig når flere funksjoner kan evalueres samtidig
  - Ingen delte data
- Kunklusjon: Funksjonell programmering egener seg godt for parallellitet

Parellellitet ved meldingsutvekling

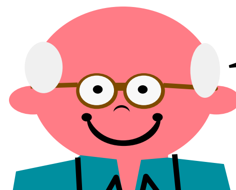
- Et meget enkelt paradigme:
- Trådene / prosessene har ingen felles variable
- All kommunikasjon skjer ved å sende meldinger mellom prosessene/trådene

- Enkelt å programmere (?)
- Lettere å finne feil (?)
- Men neppe mer effektivt

- Eksempel: MPI - Message Passing Interface

Nytt eksempel

- Finne minst tall i en array



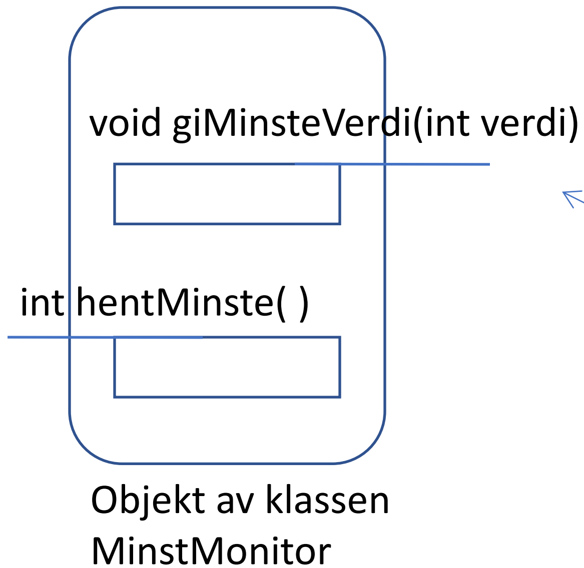
UNIVERSITETET I OSLO

Eksempel på parallellisering: Finn minste tall i en tabell

(samme teknikk for å finne sum / gjennomsnitt)

Institutt for informatikk

Hovedprogrammet starter N tråder og venter på at de alle er ferdige **før det henter minste verdi fra monitoren MinstMonitor**



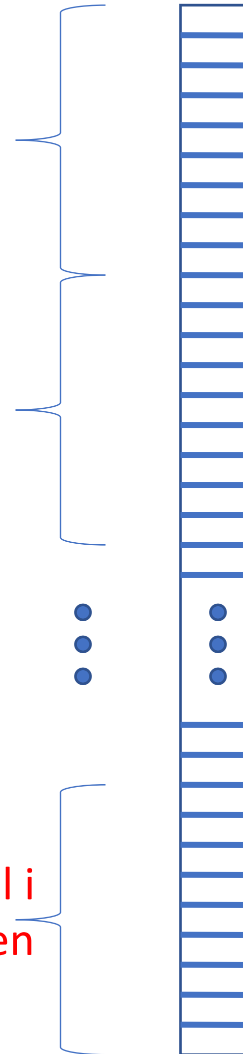
Objekt av klassen MinstMonitor

Trådenes minste verdi gis til monitoren

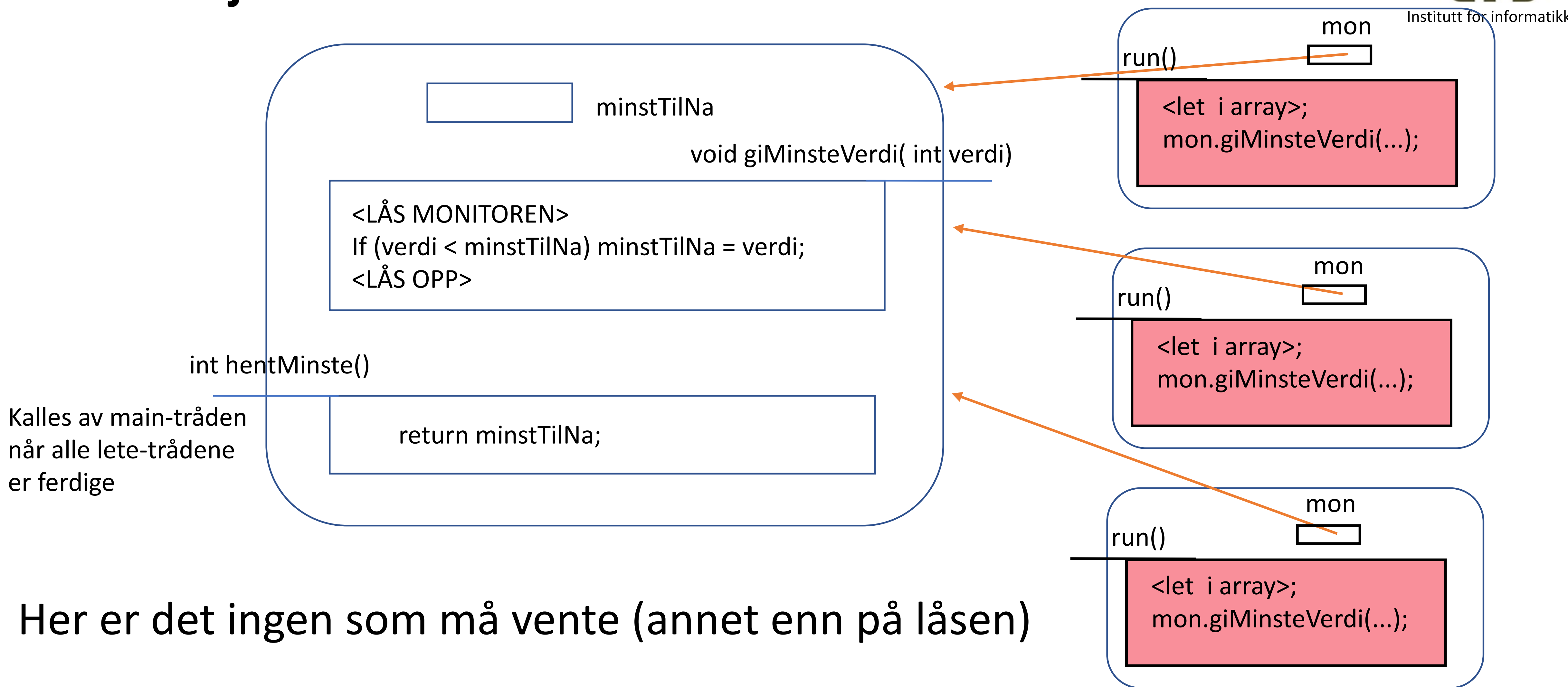
Tråd 1 finner minste tall i denne delen av tabellen

Tråd 2 finner minste tall i denne delen av tabellen

Tråd n (64 ?) finner minste tall i denne delen av tabellen



Objekt av klassen MinstMonitor

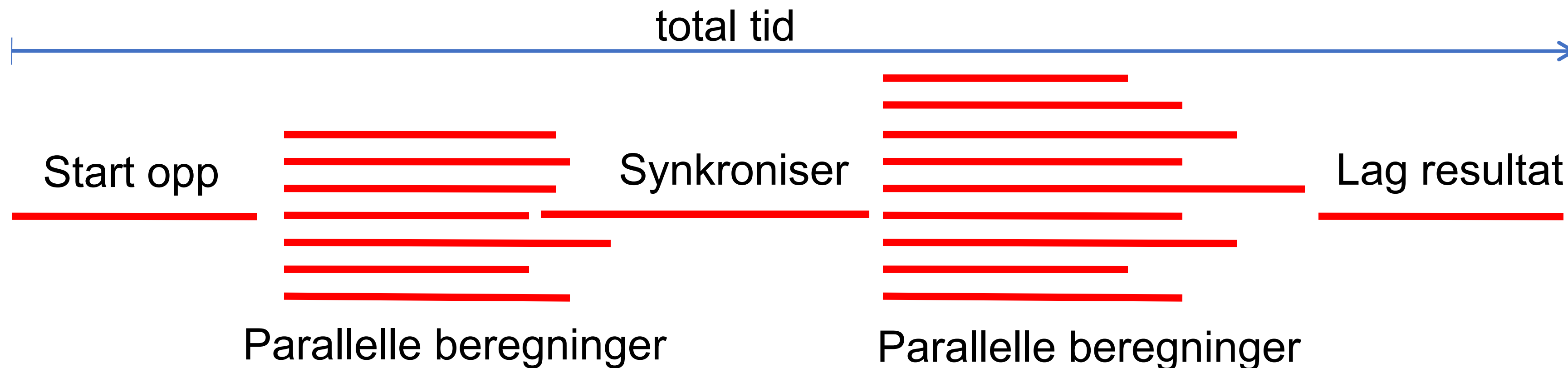


Her er det ingen som må vente (annet enn på låsen)

Men hvordan skal vi vite når alle lete-trådene er ferdige ?

Amdahls lov

- En beregning delt opp i parallell går fortere jo mer uavhengig delene er
- **Amdahls lov**:
  - Totaltiden er
    - tiden i parallell +
    - tiden det tar å kommunisere / synkronisere/ gjøre felles oppgaver
  - Tiden det tar å synkronisere er ikke parallelliserbar (hjelper ikke med flere prosessorer)
  - Men du kan være smart og lage synkroniseringen så kort eller mellom så få tråder som mulig



Amdahls lov og "finn minste tall"

- Totaltid:
  - Tiden det tar å lage og sette i gang 64 tråder
    - (kan det gjøres i parallell ?) ($\log_2 64 = 6$) *
  - Tiden det tar å finne et minste tall i min del av tabellen
  - Til slutt i monitoren: En og en tråd må teste sitt resultat
    - (Kan det gjøres i parallell ?)

* Start med én tråd, vha. doblinger 6 ganger har vi 64 tråder, dvs, $2^6 = 64$, og $\log_2 64 = 6$, se de to neste sidene

2, 4, 8, 16, 32, 64

UNIVERSITETET I OSLO

Institutt for informatikk

Trær er populære i informatikk:

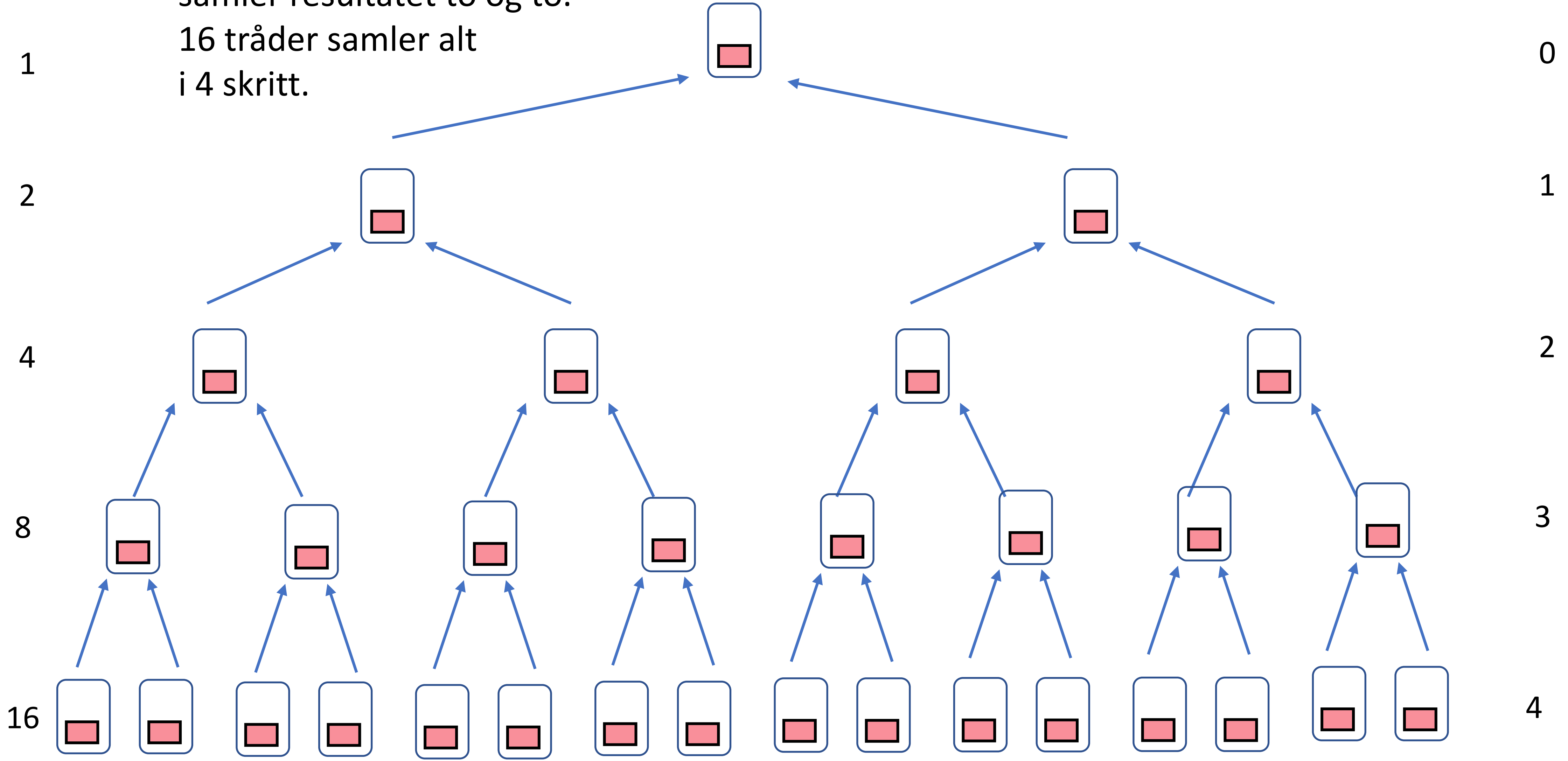
Alle trådene lager og starter to nye
Her ser vi hva som da skjer

Antall	Nivå
1	0
2	1
4	2
8	3
16	4

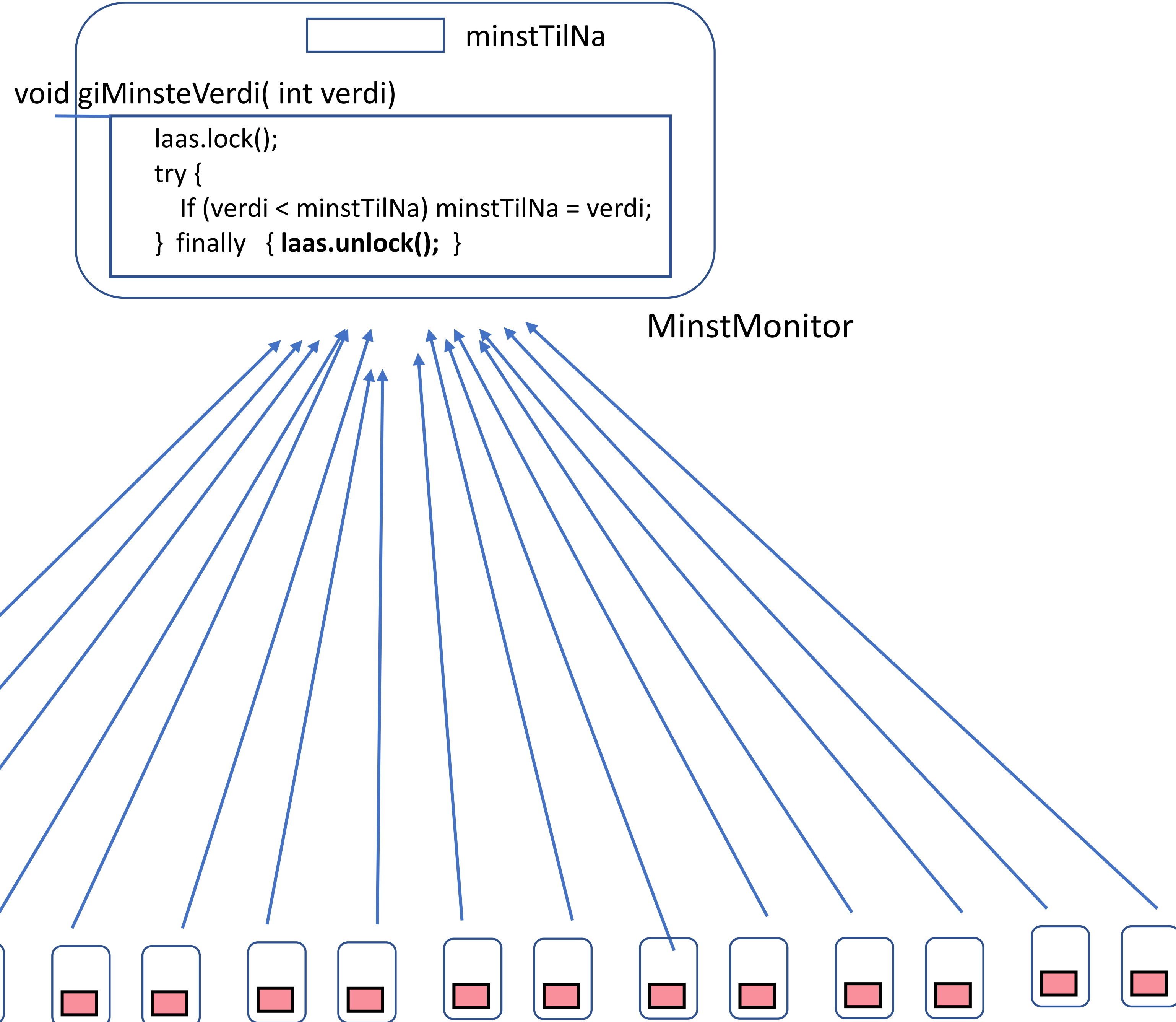
UNIVERSITETET I OSLO

Institutt for informatikk

Vi kunne tenkt oss at trådene samler resultatet to og to.
16 tråder samler alt i 4 skritt.



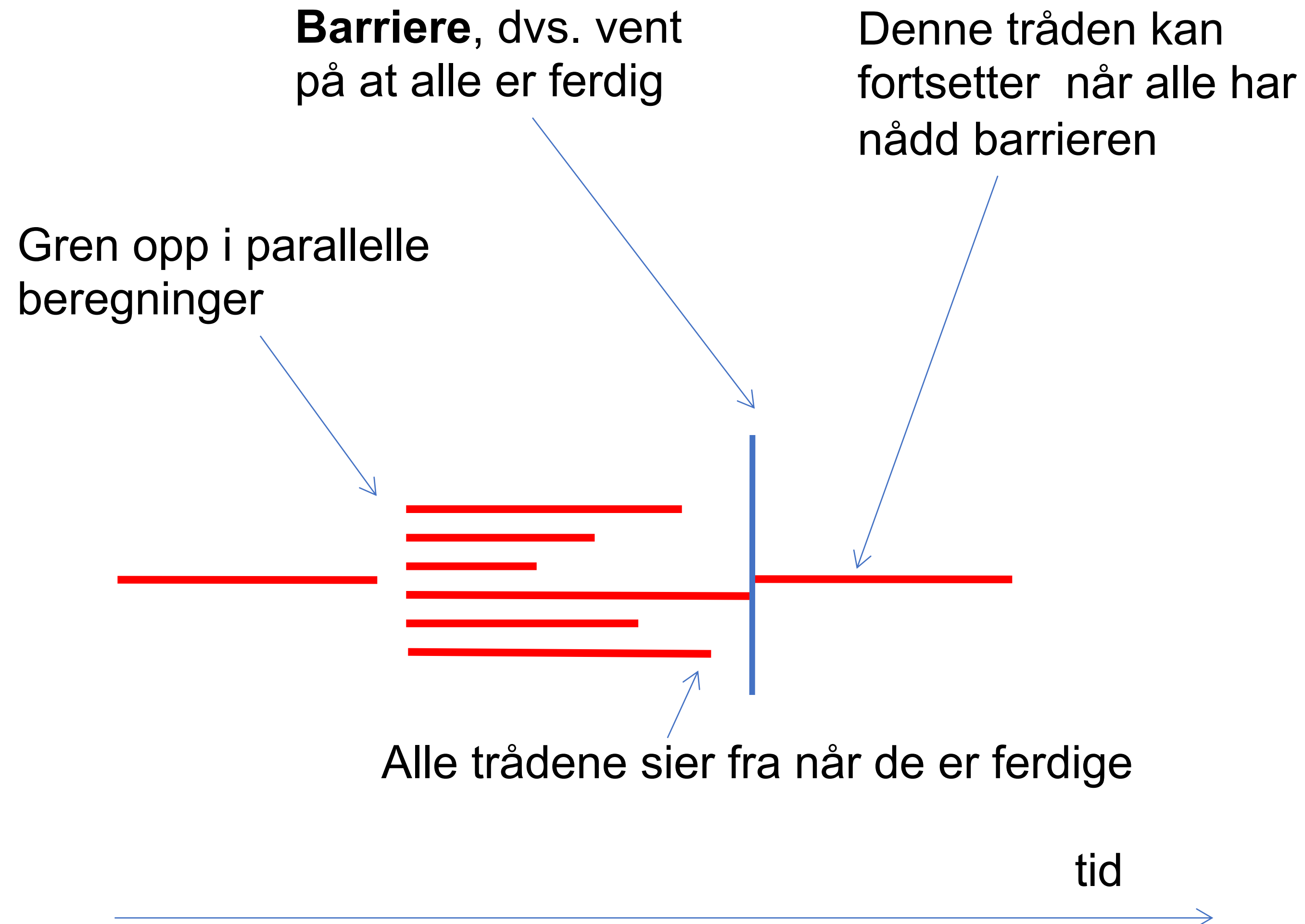
Men vi skal gjøre det enklere for oss selv:



«Finn minste tall i en stor array» ved hjelp av mange (64?) tråder

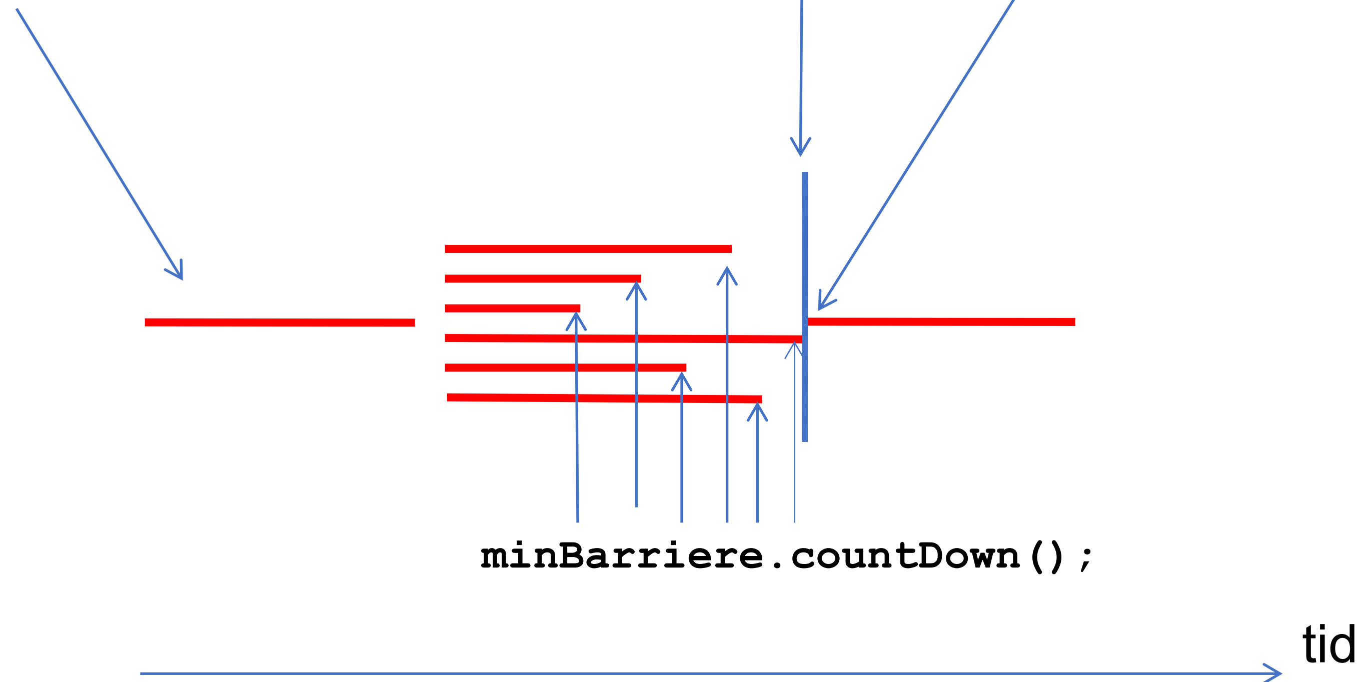
- Hvordan kan vi vite når alle lete-trådene er ferdige og resultatet ligger ferdig i monitoren?

Barrierer i parallellprogrammering



Barrierer i Java

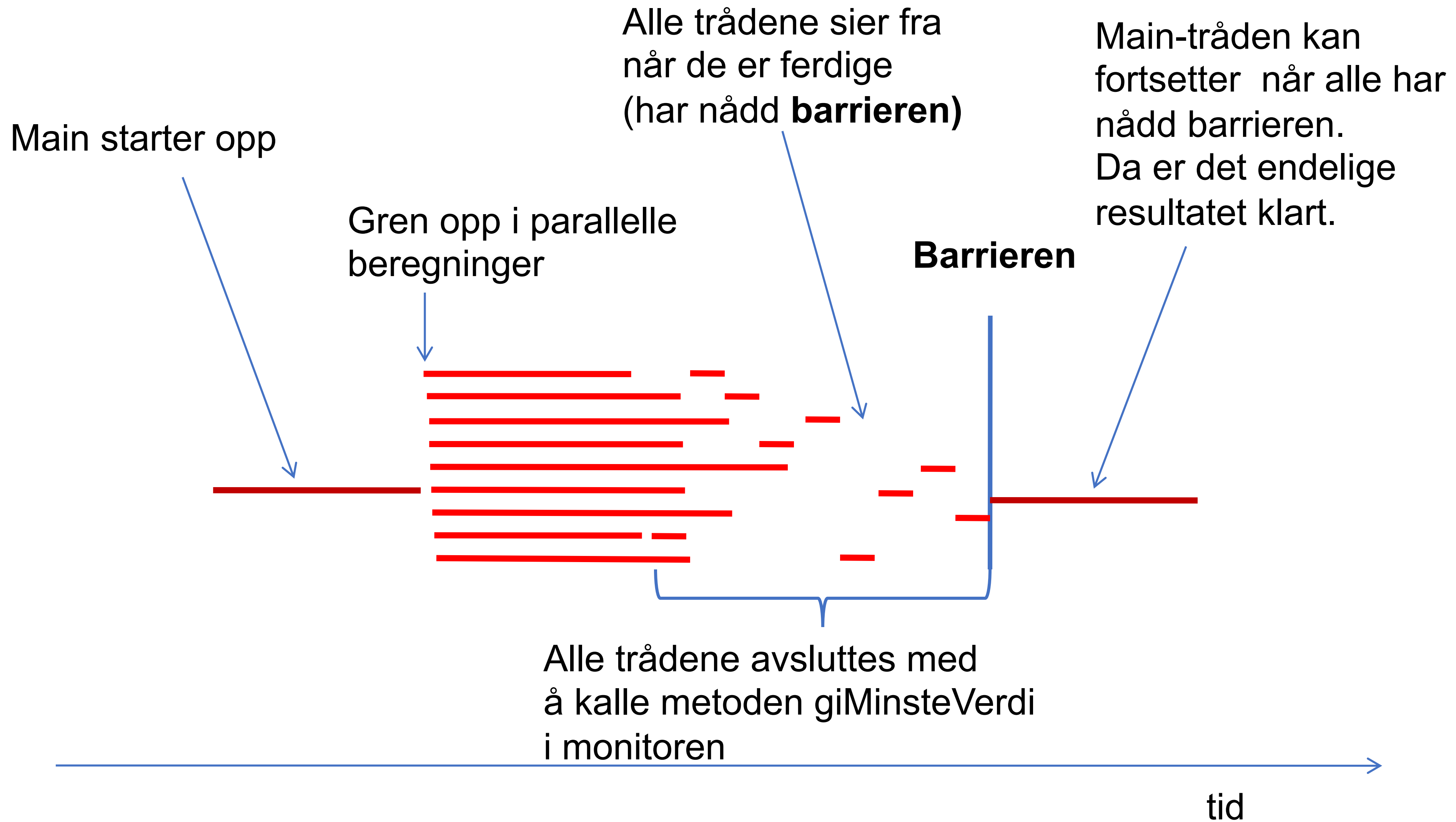

```
import java.util.concurrent.*;
```



UNIVERSITETET I OSLO

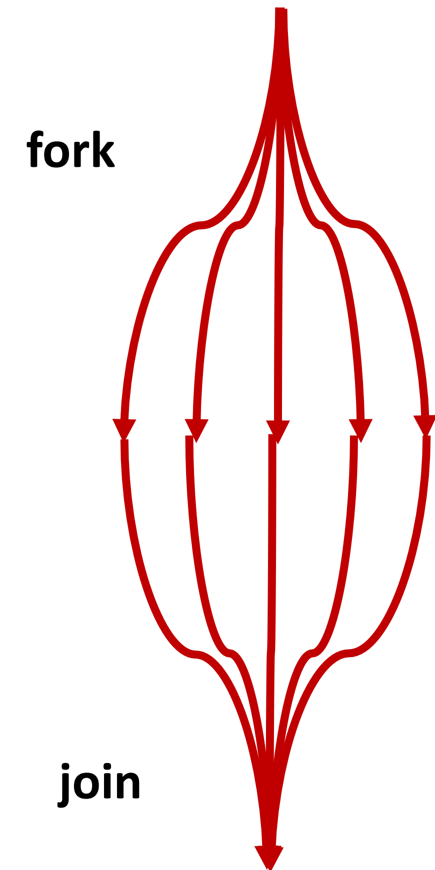
Barrieren i dette eksemplet

Institutt for informatikk



Gafling og møting (fork and join)

- I parallellprogrammering og på engelsk brukes "fork" om å lage en ny tråd eller en ny prosess
- Jeg tror ikke det brukes noen norsk oversettelse (å gafle? forgrene?)
- Det finnes ikke noen fork-mekanisme i Java
  - Du må selv lage en ny tråd, og eventuelt gå i løkke for å lage flere
  - Når tråder samler seg / terminerer kalles det gjerne join (neste side)

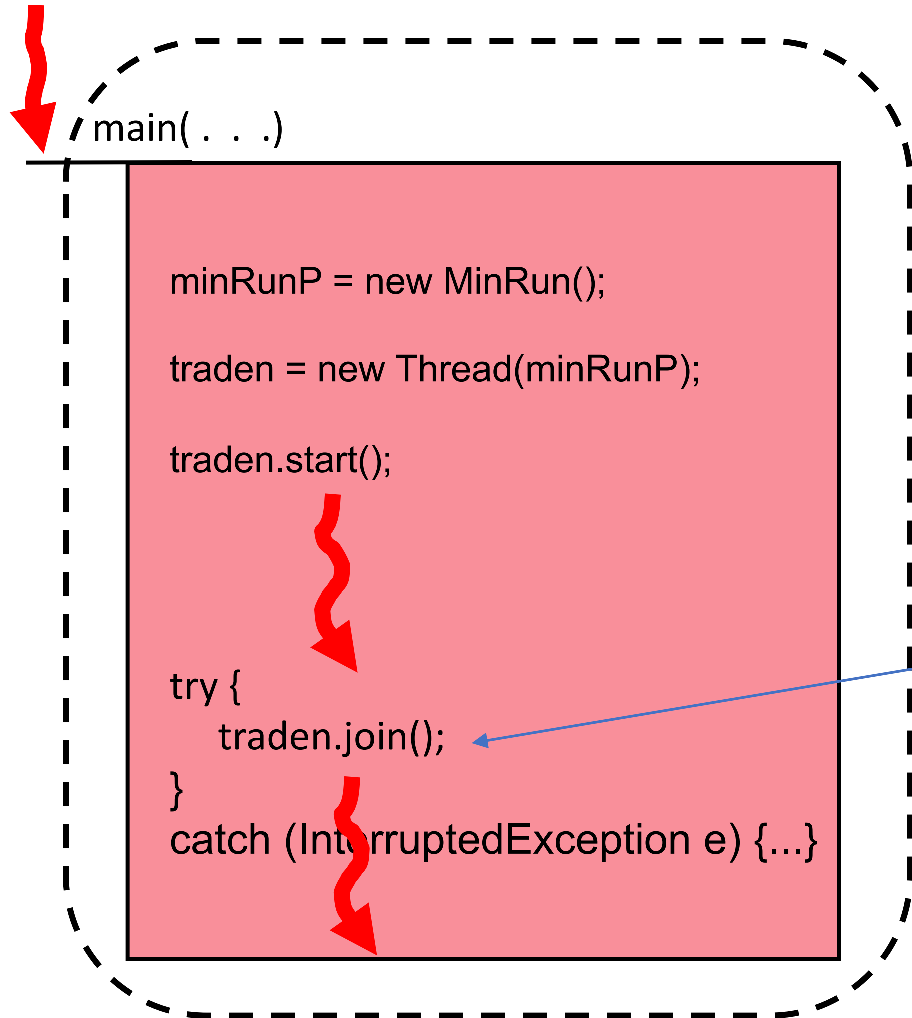


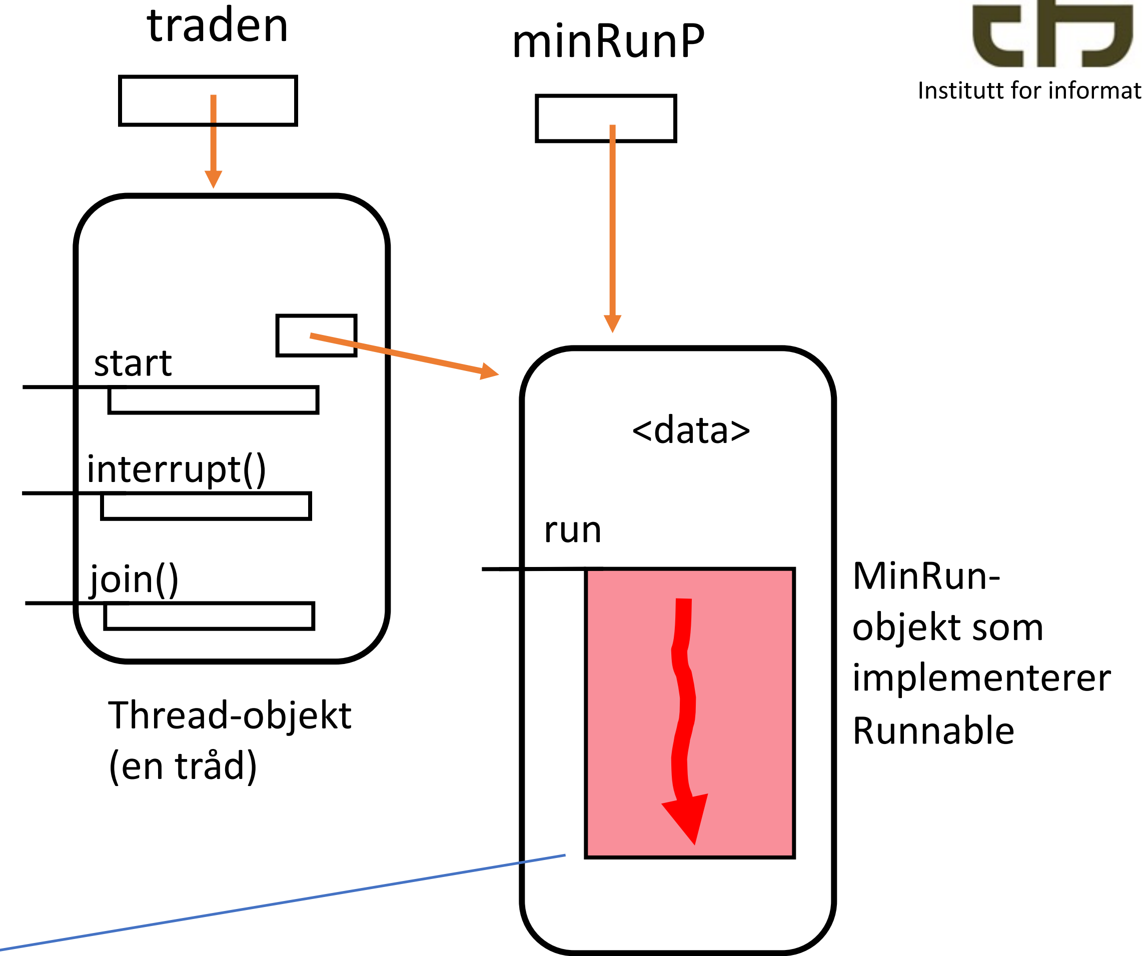
UNIVERSITETET
I OSLO

join()

Institutt for informatikk

F.eks. kjøretidsystemet kaller main()



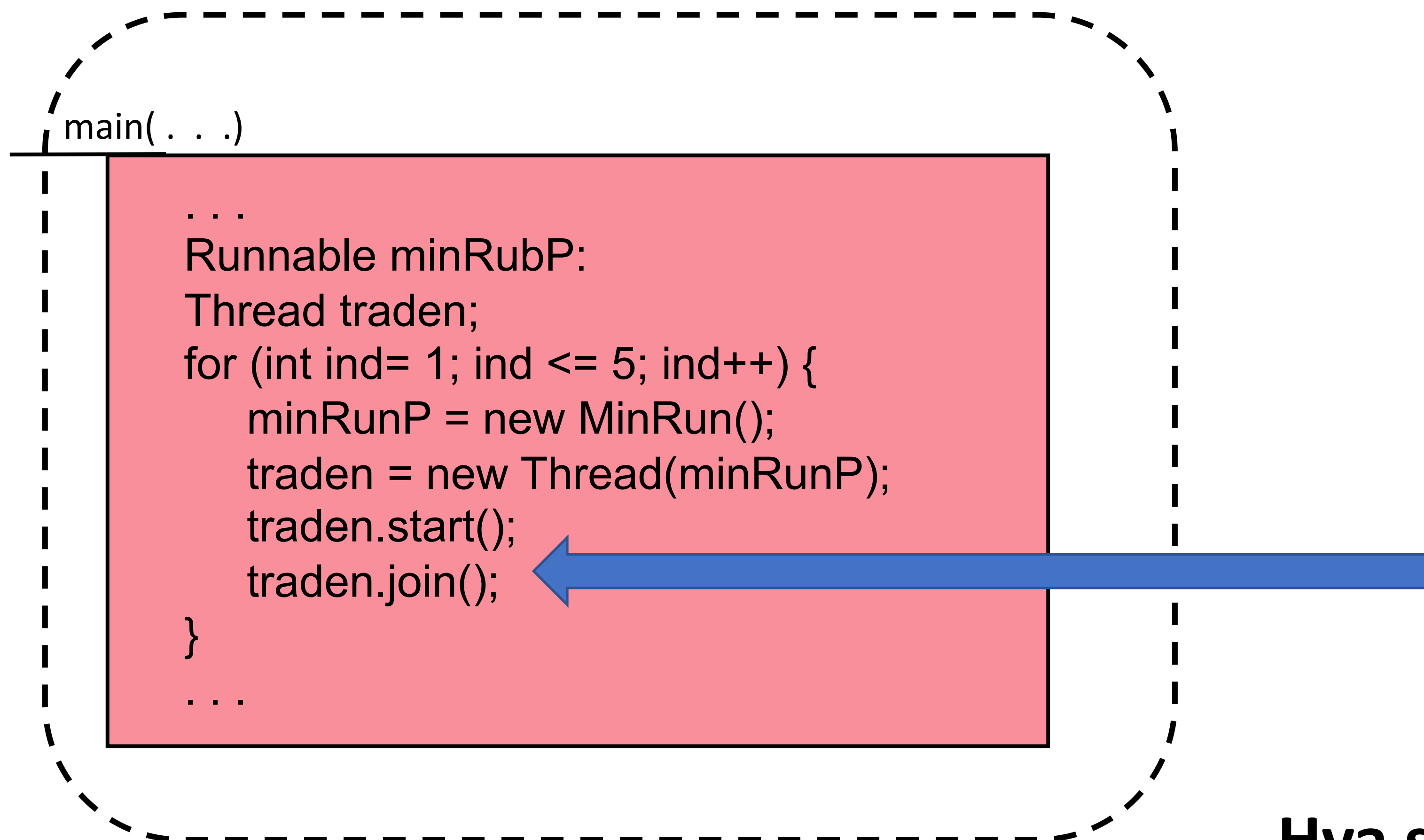


MinRun-objekt som implementerer Runnable

Tråden som kaller join() venter på at den andre tråden skal avslutte run()

Pass på, hvis du starter og venter på mange tråder:

Ikke gjør slik:

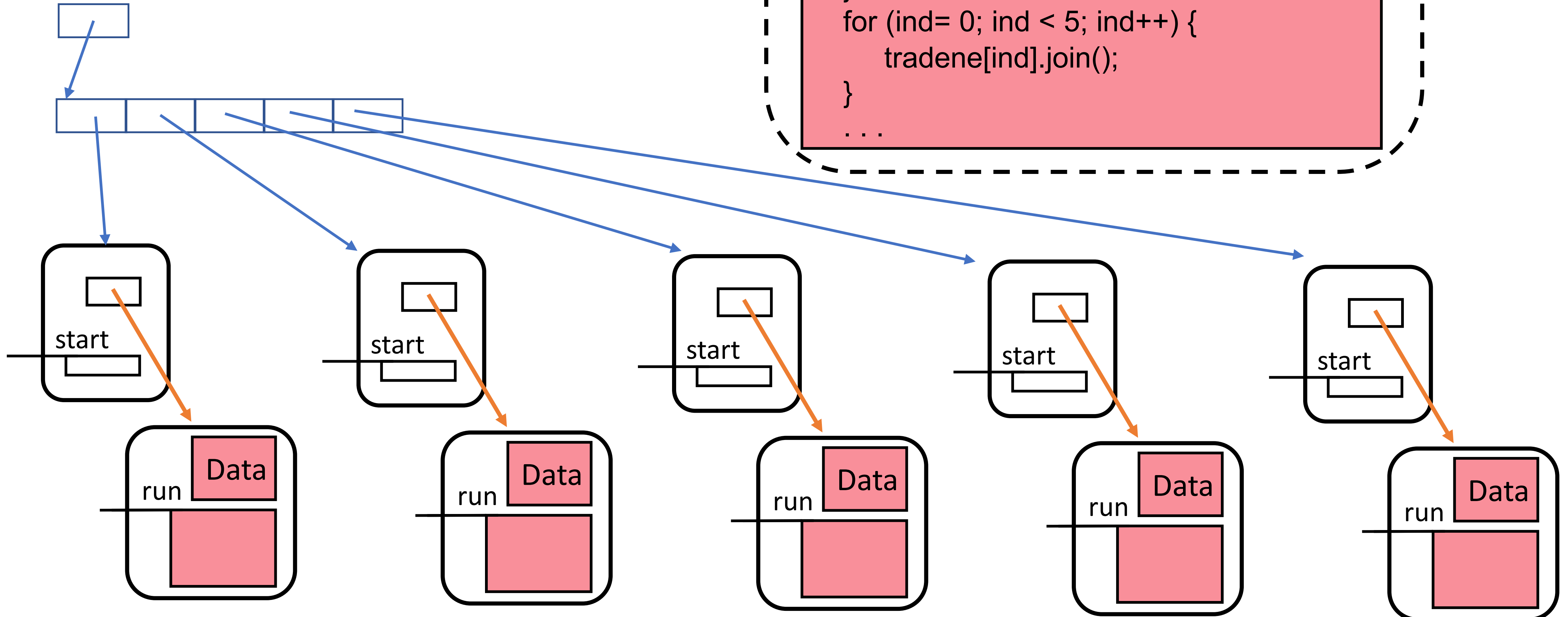


Hva skjer?

Men slik:

main( )

```
. . .
Runnable minRunP;
Thread[ ]  tradene = new Thread[5];
int ind;
for (ind= 0; ind < 5; ind++) {
     minRunP = new MinRun();
     traden[ind] = new Thread(minRunP);
     tradene[ind].start();
}
for (ind= 0; ind < 5; ind++) {
     tradene[ind].join();
}
. . .
```

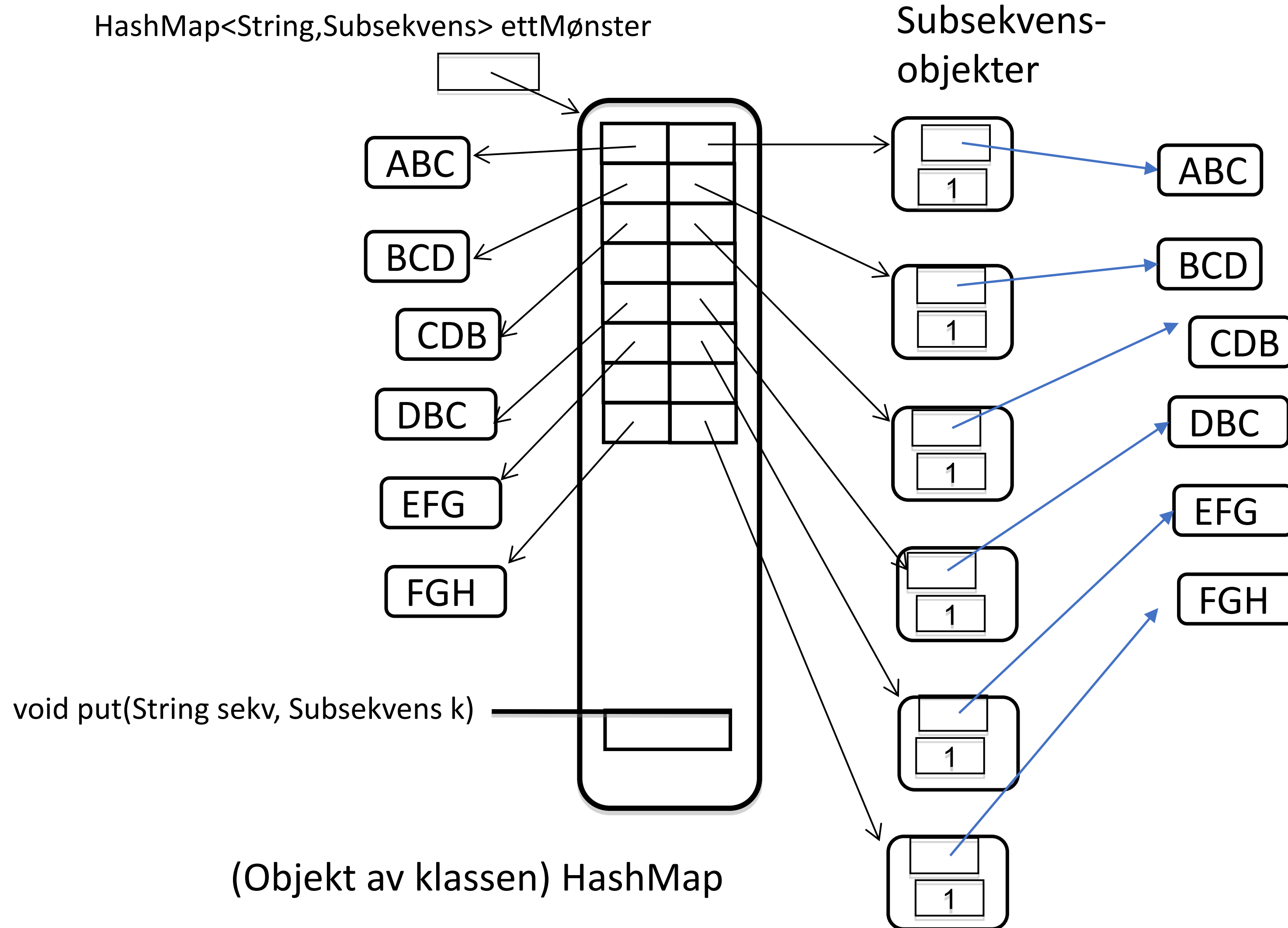


Oblig 5: Om å finne potensielle sykdommer

- I denne obligen skal vi analysere blodprøver for å finne mulige sykdommer
- Vi skal finne mønstre i immunrepertoarene i blodprøvene (i DNA)
  - Data fra én blodprøve fra én person finner vi i én fil
- Først skal vi se på en og en person - og finne et mønster for denne personen
- Deretter skal vi slå sammen (”flette”) to og to mønstrene for å finne mønstre som forekommer i flere (**mange**) personer

HashMap med mønster fra en person

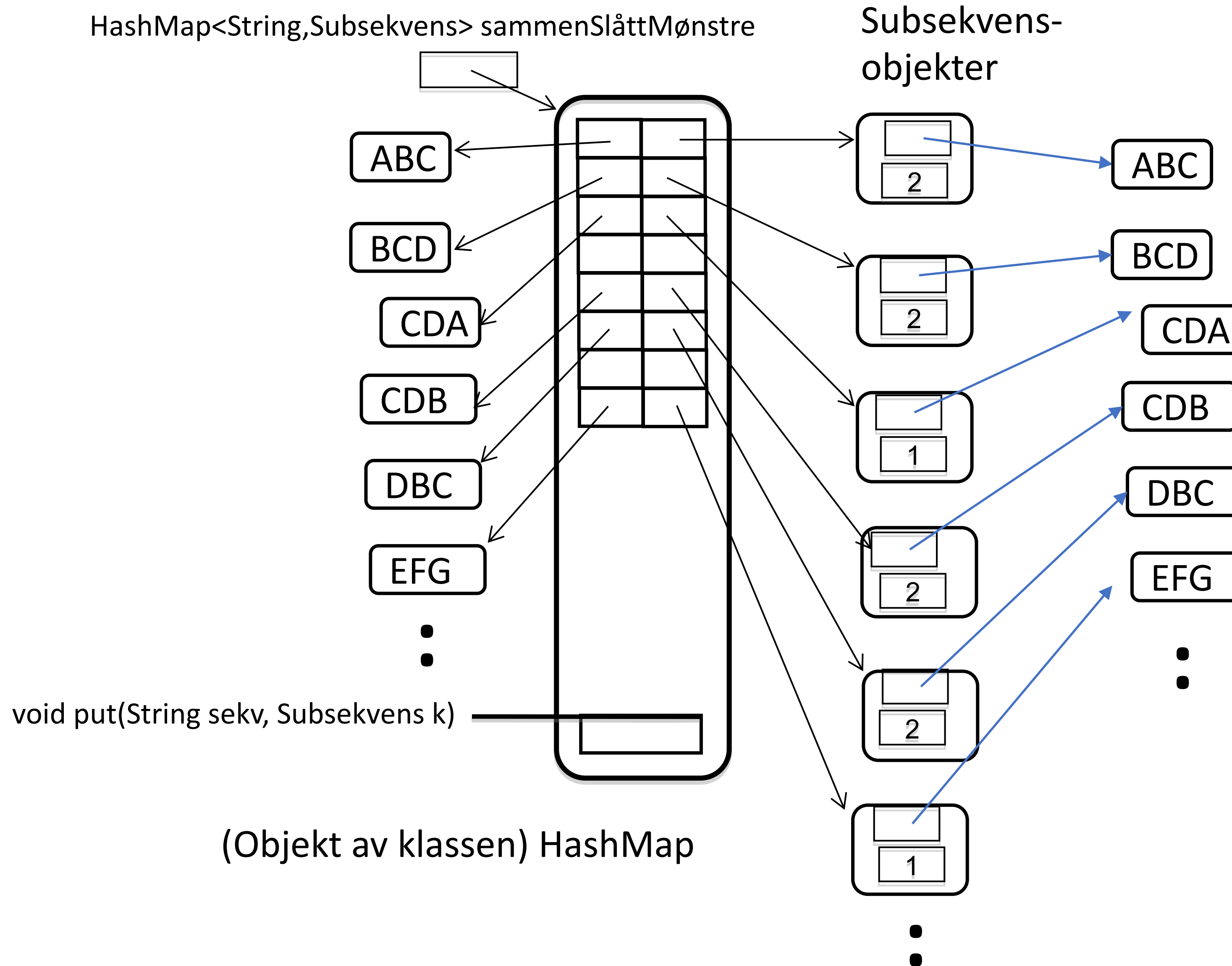
Deklarasjon: HashMap<String,Subsekvens> alle = new HashMap<>( );



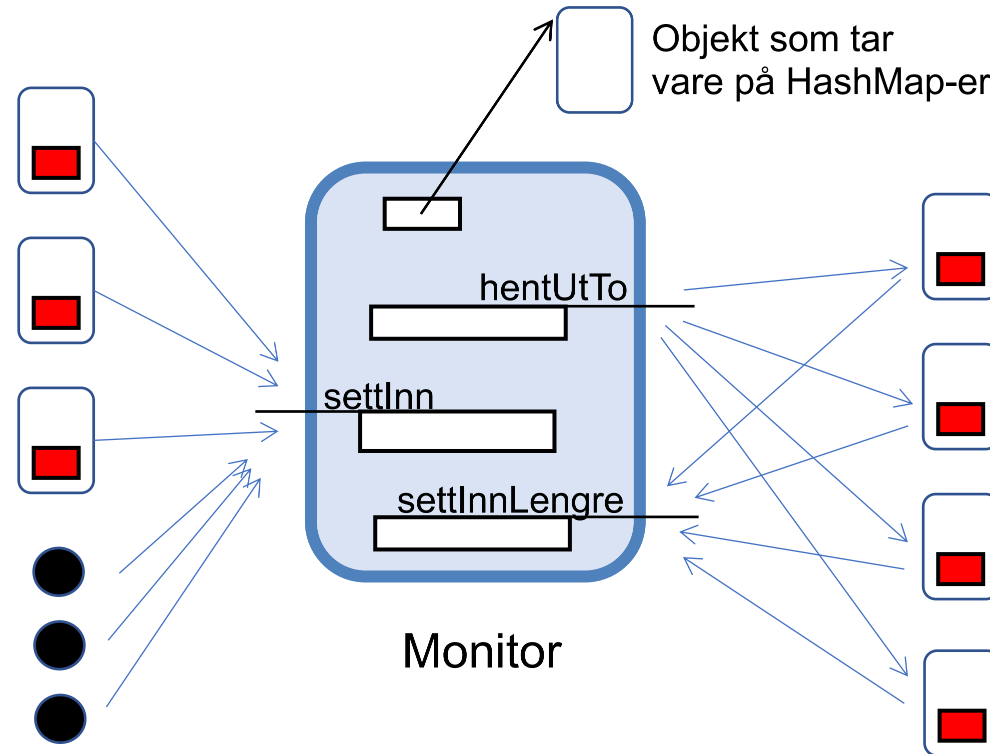
UNIVERSITETET I OSLO

HashMap med mønstre fra to person slått sammen (to «flettete» HahMap-er)

Institutt for informatikk



Figur fra obligteksten (lett modifisert)



Ferdig når alle HashMap-ene er slått sammen til **en stor HashMap**

Når vet trådne at de er ferdige?

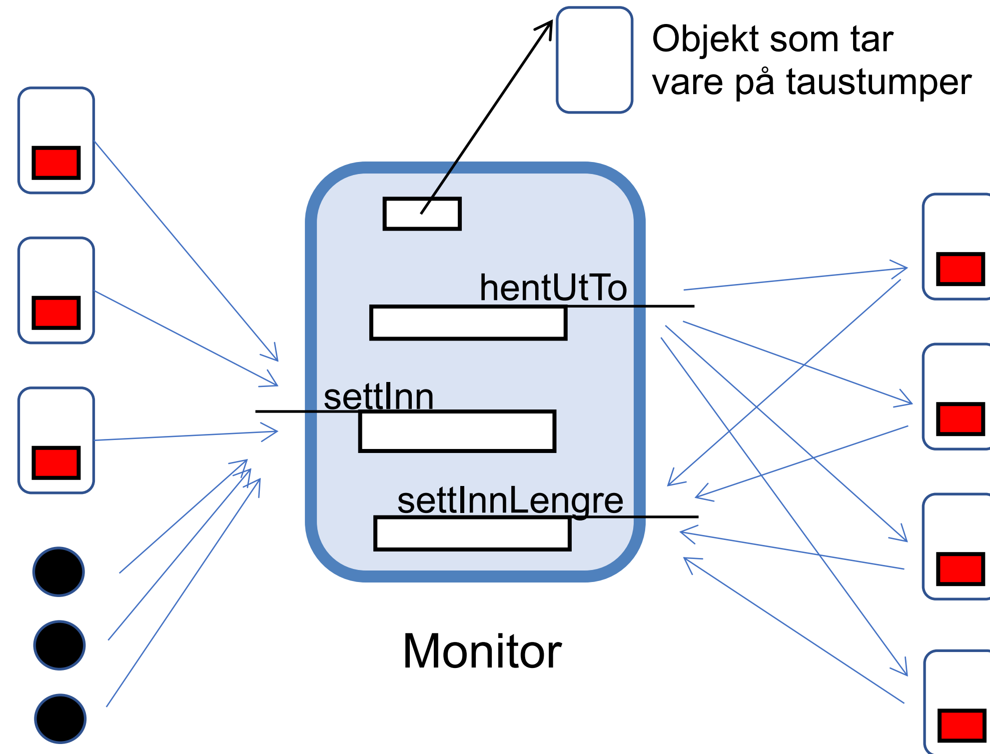
i denne illustrasjonen skal 20 tråder lese en fil hver og lager 20 mønstre

4 tråder som slår sammen to og to HasMap-er

UNIVERSITETET I OSLO

Figur fra obligteksten (lett modifisert)

Institutt for informatikk



i denne illustrasjonen skal 20 tråder lese en fil hver og lager 20 mønstre **= 20 taustumper**

4 tråder som binder sammen to og to taustumper

Ferdig når alle taustumpene er bundet sammen til **ett langt tau**

Når vet trådne at de er ferdige?

Oblig 5: Om å finne potensielle sykdommer

- I denne obligen skal vi analysere blodprøver for å finne mulige sykdommer
- Vi skal finne mønstre i immunrepertoarene i blodprøvene (i DNA)
  - Data fra én blodprøve fra én person finner vi i én fil
- Først skal vi se på en og en person - og finne et mønster for denne personen

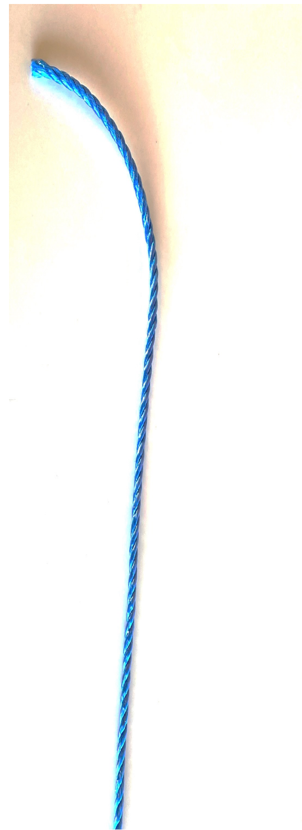
  Vi skal illustrere dette ved å late som et mønster er **et tau**

- Deretter skal vi slå sammen (”flette”) to og to mønstrene for å finne mønstre som forekommer i flere (**mange**) personer

  Vi skal illustrere det å slå sammen to og to mønstre ved å **binde sammen to og to tau**

HashMap med mønster fra en person

Deklarasjon: HashMap<String,Subsekvens> alle = new HashMap<>( );



=

HashMap<String,Subsekvens> ettMønster

Subsekvens-objekter

ABC
BCD
CDB
DBC
EFG
FGH

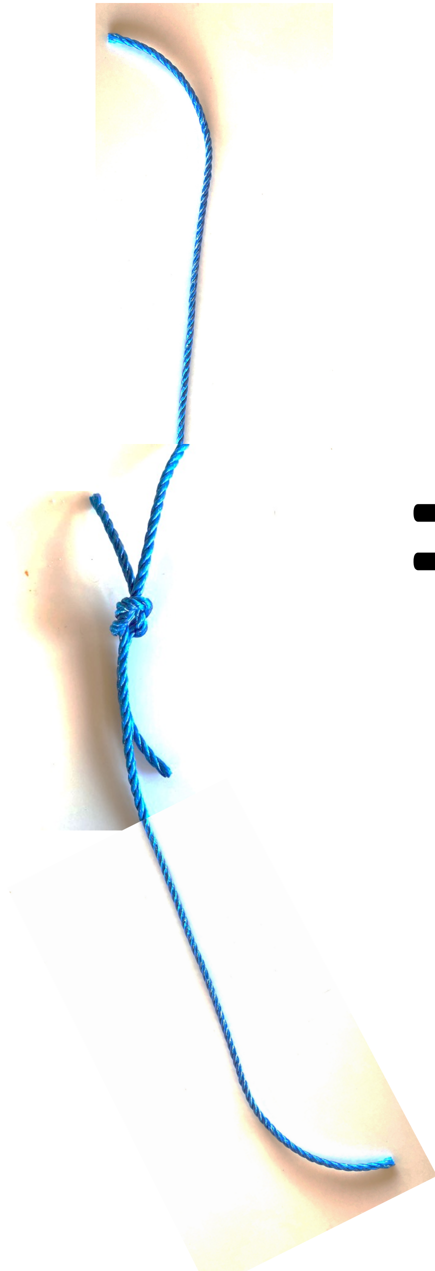
1
1
1
1
1
1

ABC
BCD
CDB
DBC
EFG
FGH

void put(String sekv, Subsekvens k)

(Objekt av klassen) HashMap

HashMap med mønstre fra to person slått sammen (to «flettete» HahMap-er)



=

HashMap<String,Subsekvens> sammenSlåttMønstre

Subsekvens-objekter

ABC
BCD
CDA
CDB
DBC
EFG
:

2
2
1
2
2
1
:

ABC
BCD
CDA
CDB
DBC
EFG
:

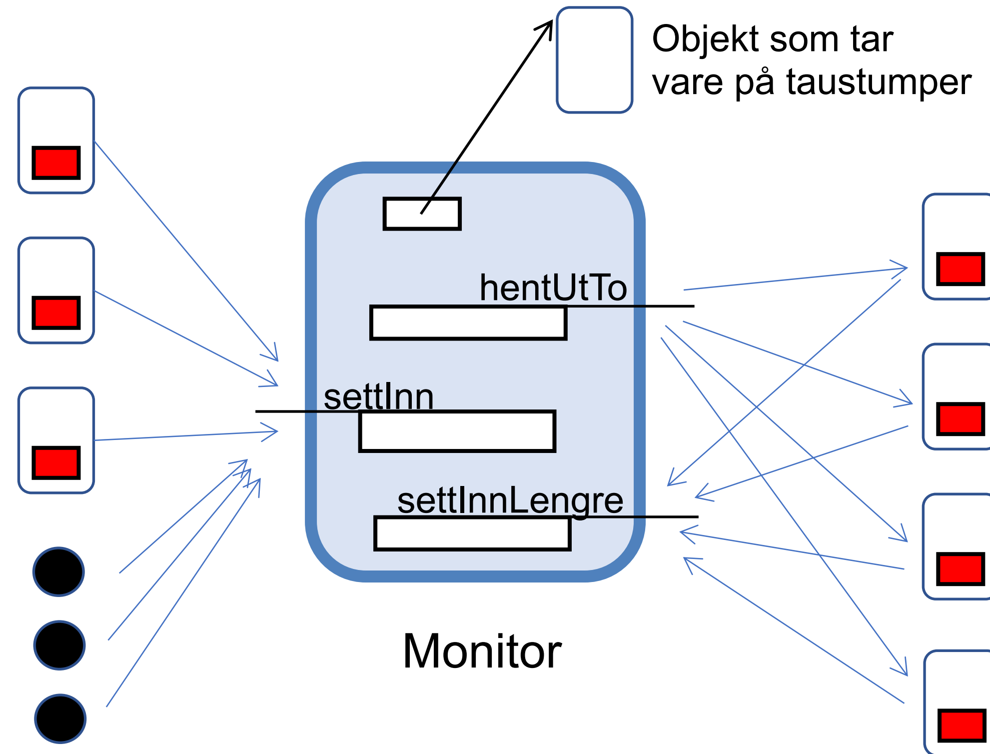
void put(String sekv, Subsekvens k)

(Objekt av klassen) HashMap

UNIVERSITETET I OSLO

Institutt for informatikk

Figur fra obligteksten (modifisert)



Ferdig når alle taustumpene er bundet sammen til **ett langt tau**

Når vet trådne at de er ferdige?

i denne illustrasjonen skal 20 tråder lese en fil hver og lager 20 mønstre
= 20 taustumper

4 tråder som binder sammen to og to taustumper

Nytt tema
(men fortsatt tråder)

Vranglås
Engelsk: Deadlock

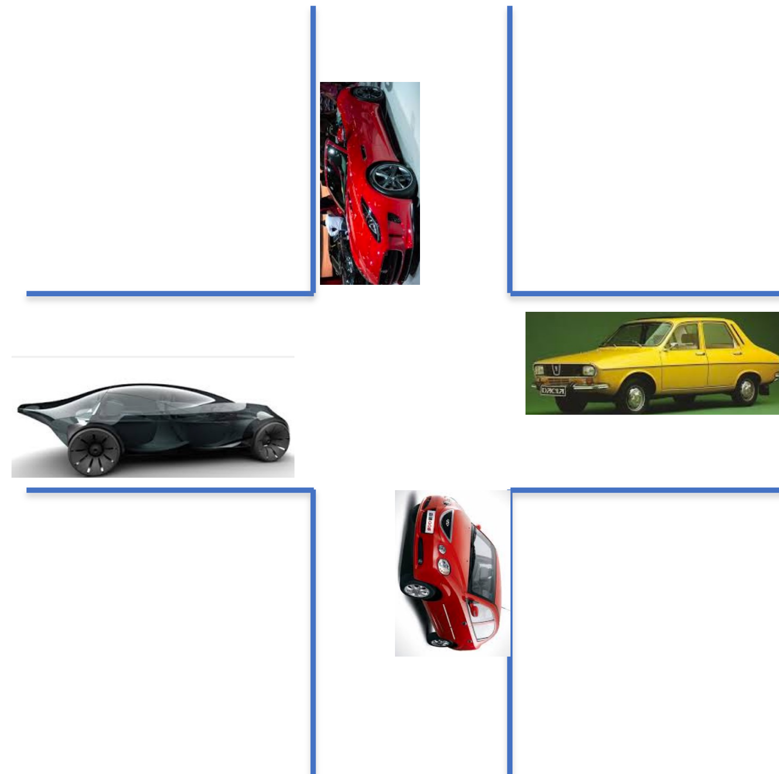
Horstmann kap. 20.5

45

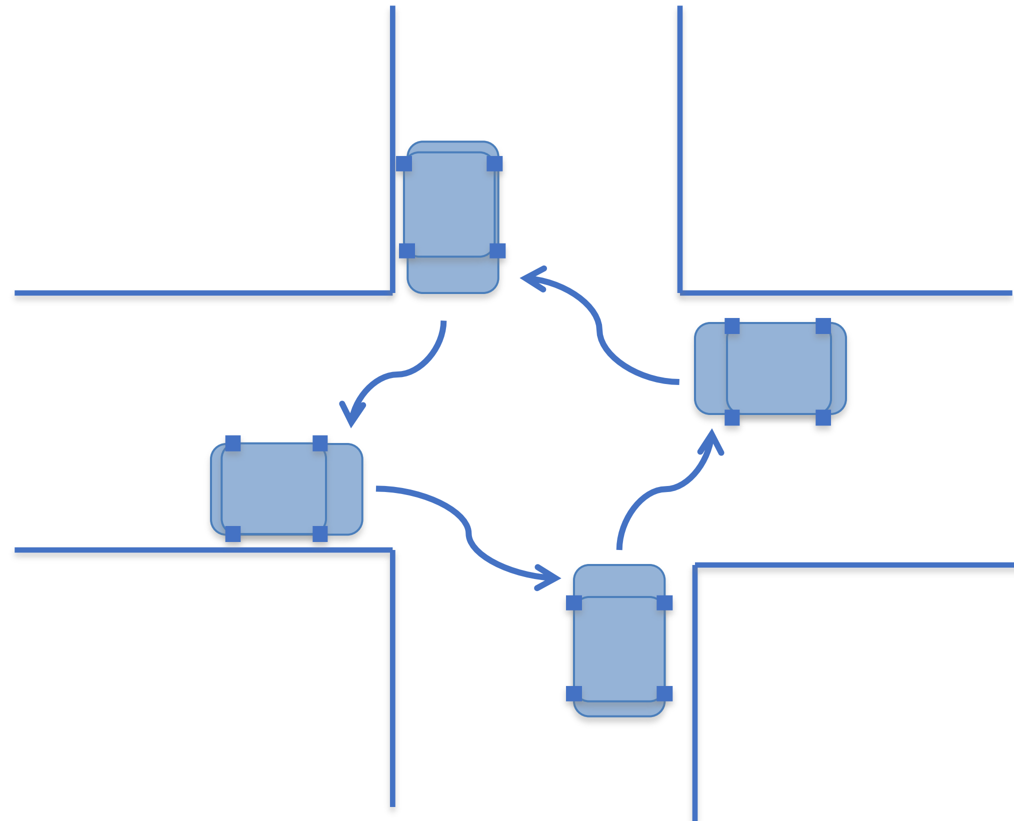
Vranglås (deadlock)

- Vranglås skjer når flere tråder venter på hverandre i en sykel:
- Eksempel
  - Veikryss:
    alle bilene skal stoppe for
    biler fra høyre

    ->
    Alle stopper = VRANGLÅS

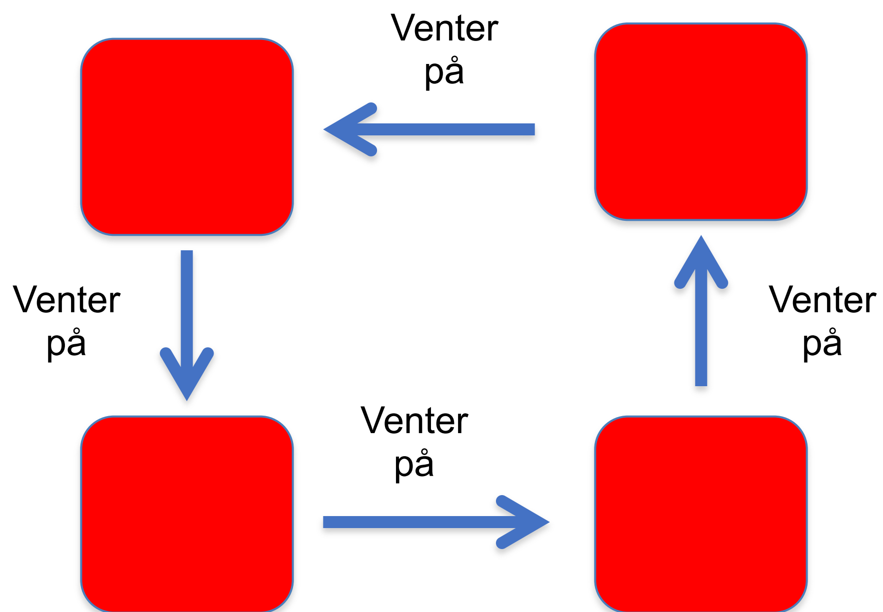


Syklisk venting



Vranglås betyr

- Flere tråder venter på hverandre
- Syklisk venting



Vranglås kan oppstå når flere tråder kjemper om felles ressurser

- En eller flere felles ressurser ønskes av mer enn en tråd
- Hvis en tråd først tar en ressurs og deretter en annen . . .

49
49

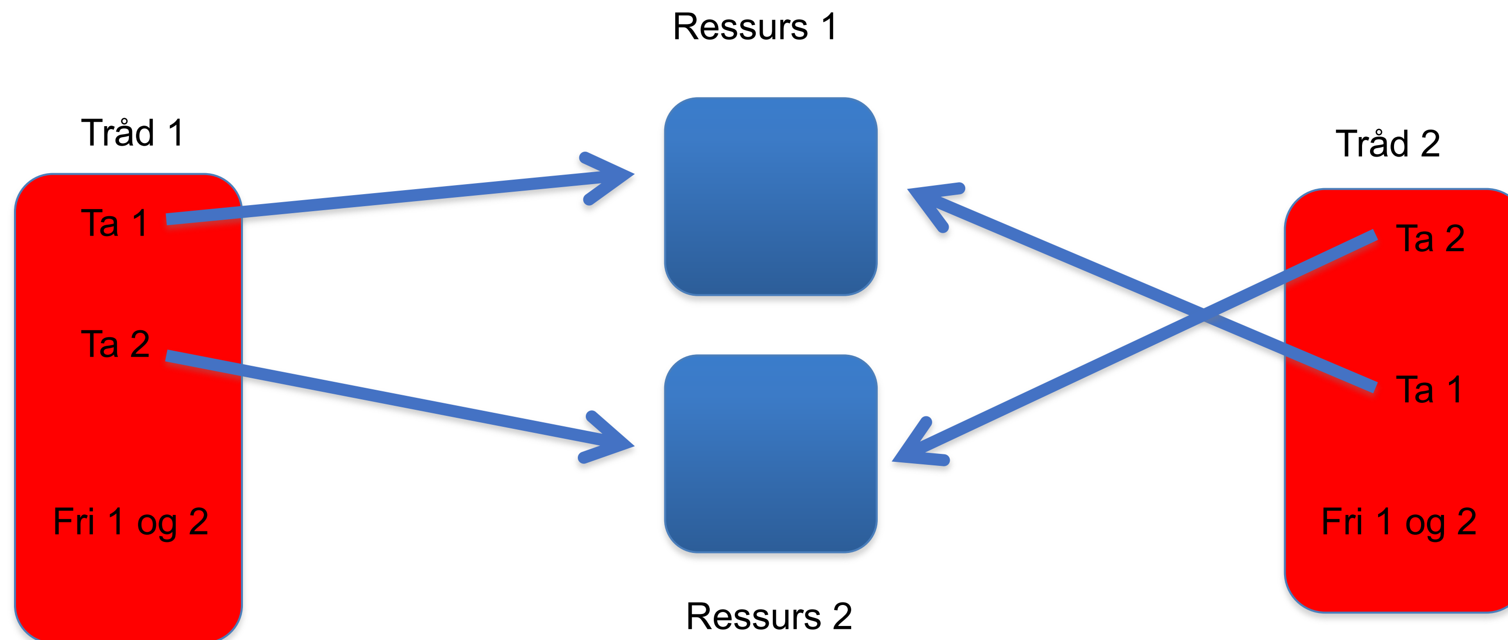
Unngå vranglås

1. Ta bare én ressurs
2. Ta alle på en gang, eller ingen
3. Alle tråder tar alle ressurser i samme rekkefølge

- Hvis vranglås har oppstått:
  - Fri en og en ressurs til det ikke lenger er vranglås

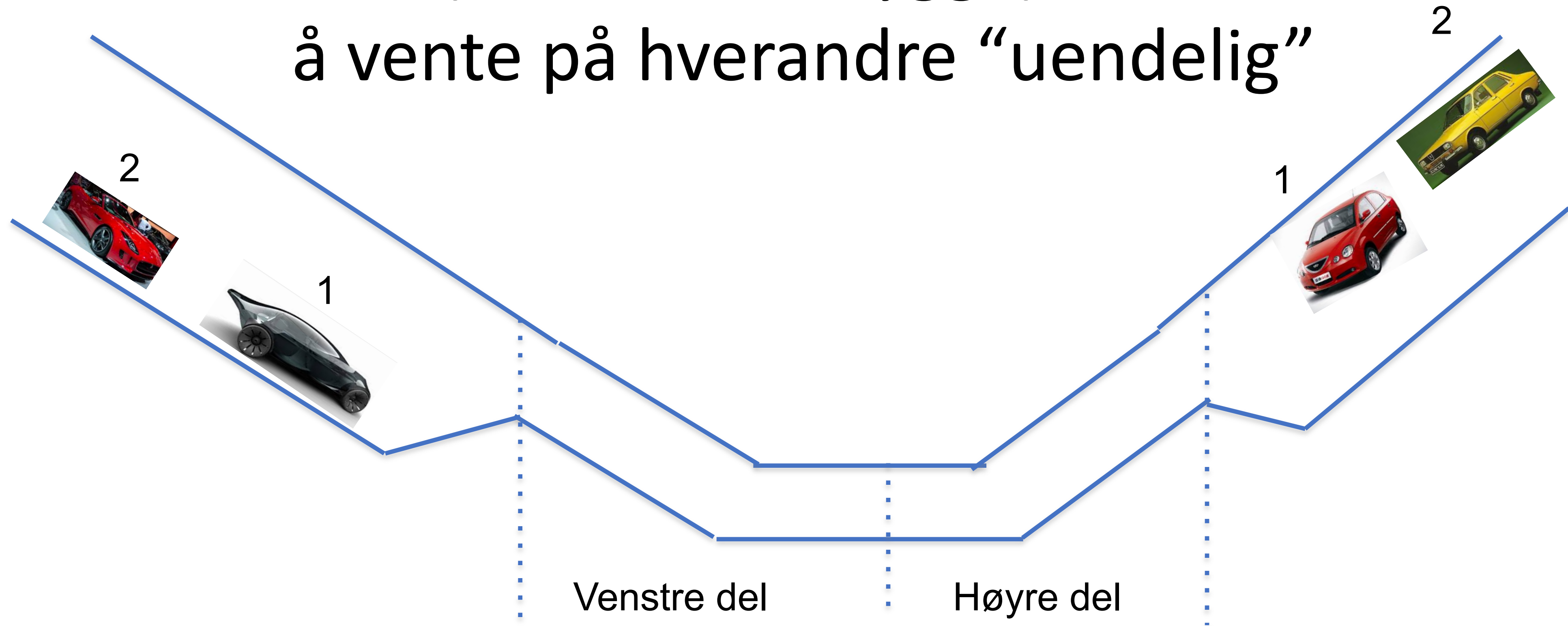
Enkleste eksempel på vranglås

- To tråder
- To ressurser som tas i forskjellig rekkefølge



Enkleste eksempel på vranglås:

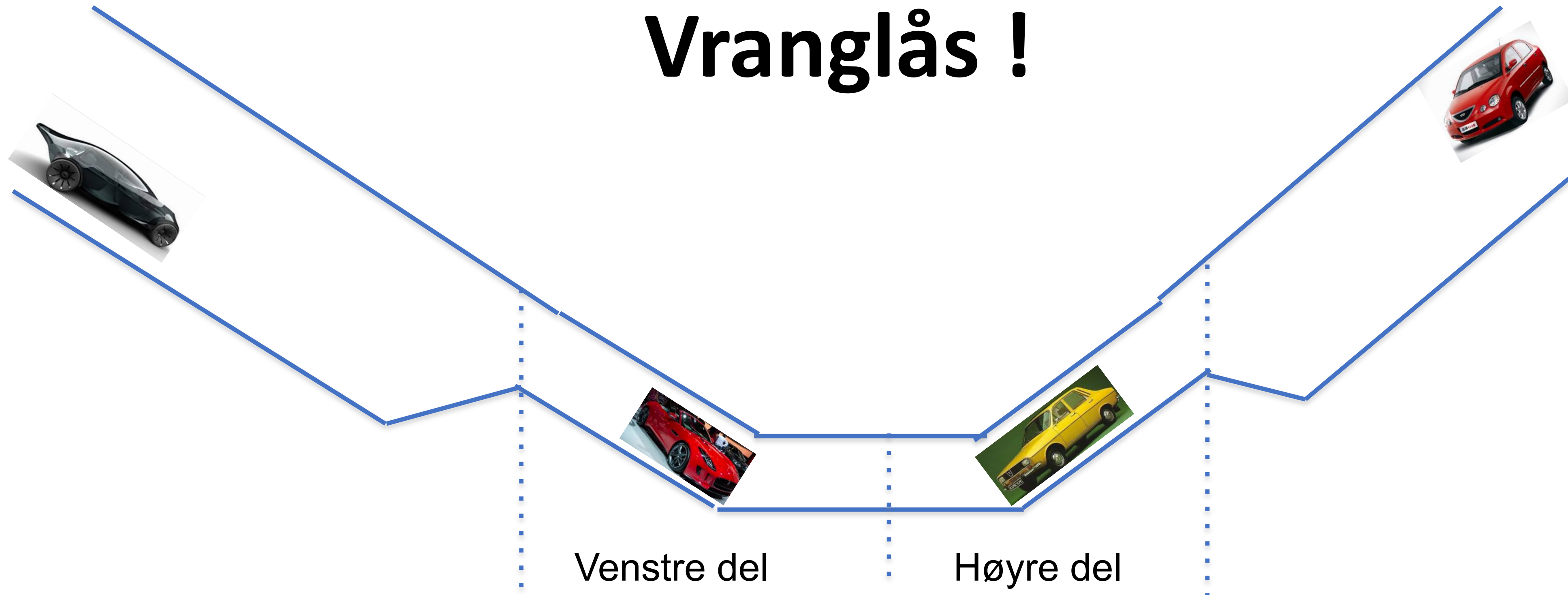
To biler (som ikke vil rygge) kan risikere å vente på hverandre “uendelig”



Smalt veistykke, to biler
kan ikke passere hverandre.
Bilene kan bare se den første delen.

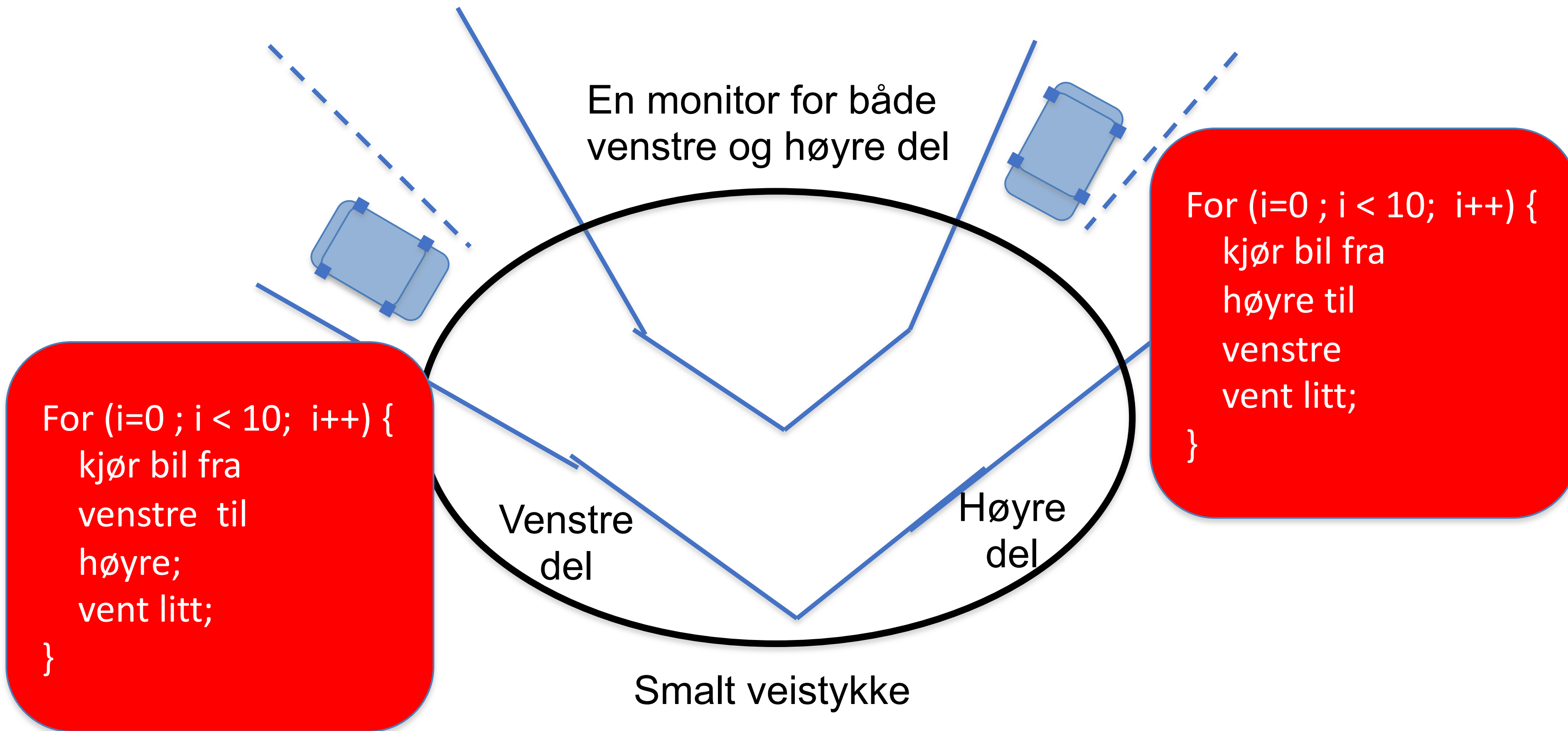
52

Vranglås !



Det smale veistykket består av to ressurser, og begge bilene venter på at den andre bilen skal bli ferdig med å bruke sin ressurs (de har tatt ressursene i forskjellig rekkefølge)

Eksempel
Program 1



Eksempel
Program 2 (refaktorert)

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

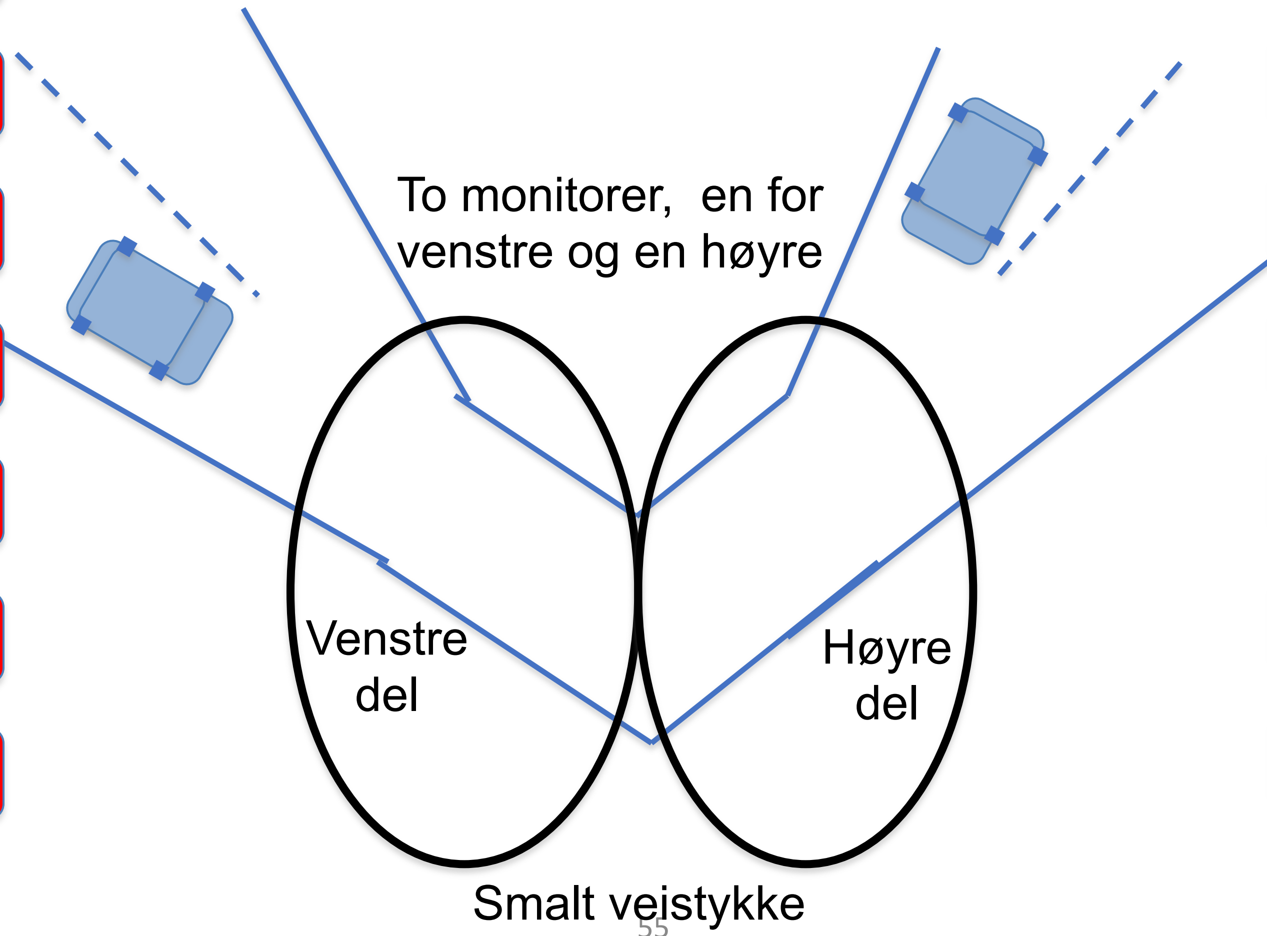
Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen



Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

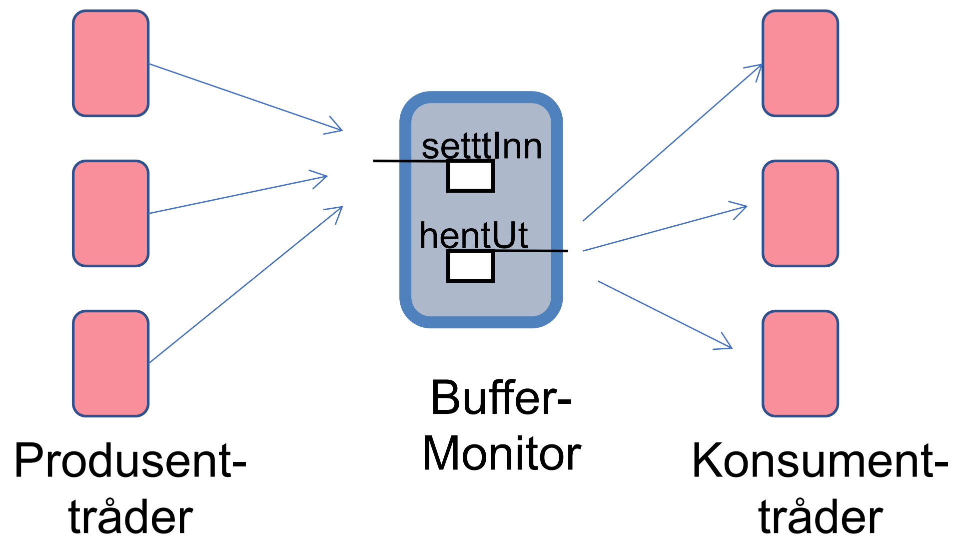
Kjør bilen

Kjør bilen

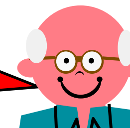
Hoved "take-away" i dag

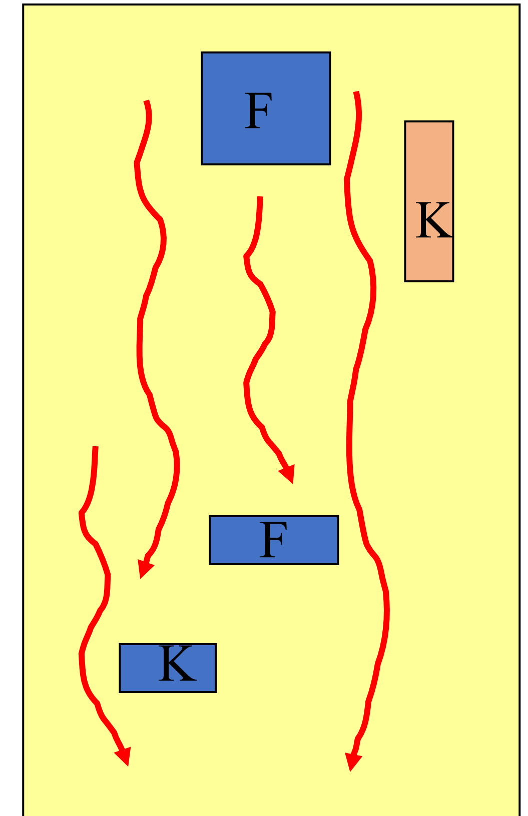
Du kan programmere parallelle aktiviteter i Java ved hjep av tråder.

Tråder deler adresserom.
Et felles objekt kalles en **monitor**. Metodene i en monitor er *kritiske regioner.*



Monitor er ikke noe ord i Java





K: konstante data (immutable)

Oppsummering. Vi har lært:

- Hvordan vente på at andre tråder er ferdig
  - join() - barrierer
- At det er utfordrende å programmere parallelle aktiviteter
  - Viktig å resonere om programmets tilstand.
    - Bruk invarianter
  - Umulig å teste nok tilfeller
    - Tidsavhengige feil kan vise seg først etter mange år.
- Amdahls lov sier at mer parallellitet er bedre, og at synkronisering koster
- Vranglås kan forekomme i parallellprogrammering.
  - Vranglås kan unngås på forskjellige måter