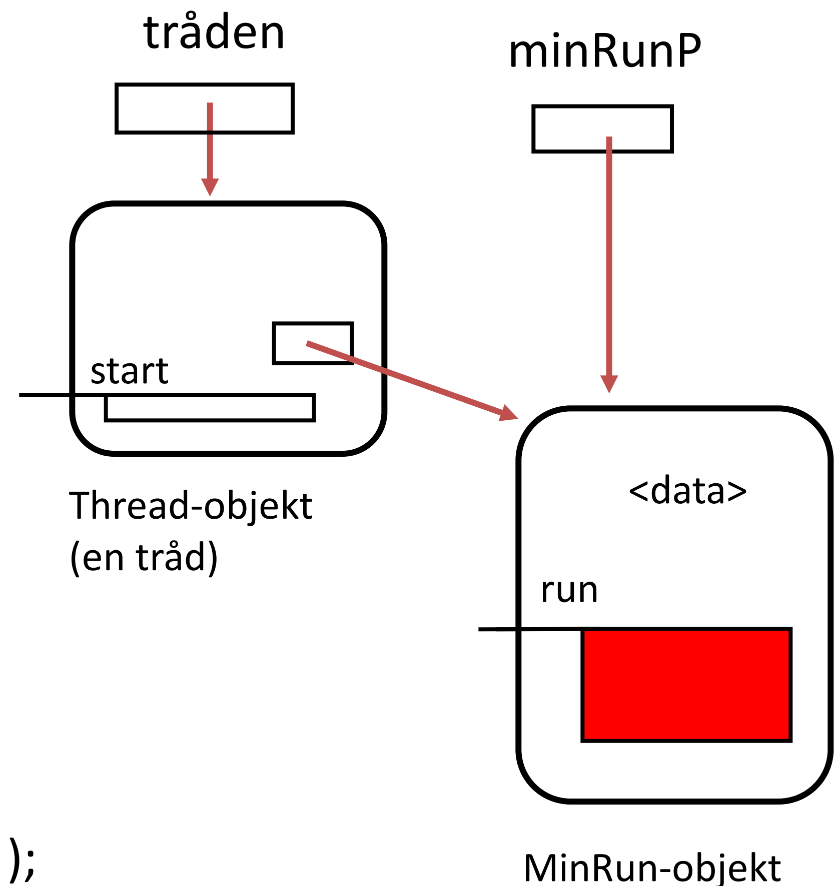
INF1010
Tråder II
6. april 2016

Stein Gjessing
Institutt for informatikk
Universitetet i Oslo

Institutt for informatikk

Tråder i Java




```
class MinRun implements Runable {
    <datastruktur>
    MinRun( . . . ) { . . . }
    public void run( ) {
        . . .
        . . .
    }
} //end class MinRun
```

En tråd lages og startes opp slik:

```
Runnable minRunP = new MinRun( . . . );
Thread tråden = new Thread(minRunP);
tråden.start( );
. . .
. . .
. . .
tråden.join();  venter på at "tråden" terminerer
```

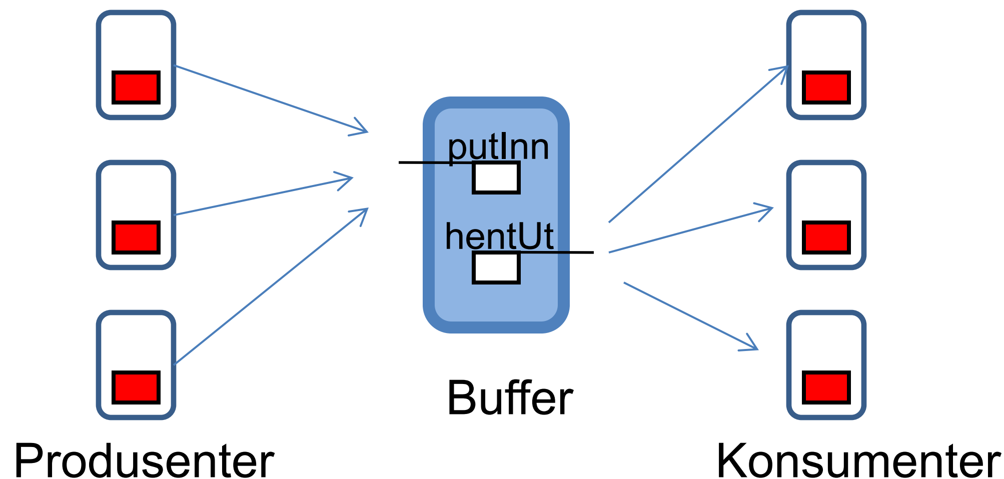
(må beskyttes av try-catch)

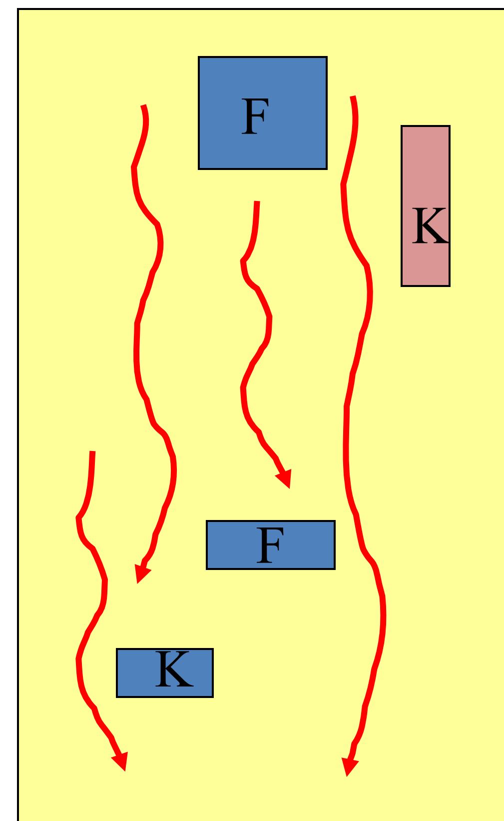
Kommunikasjon mellom tråder: Felles data

Felles data (blå felt, F) må vanligvis bare aksesseres (lese eller skrives i) av en tråd om gangen. Hvis ikke blir det kluss i dataene.
Et felles objekt kalles en **monitor**.

Metodene i en monitor må være *kritiske regioner*

f.eks.





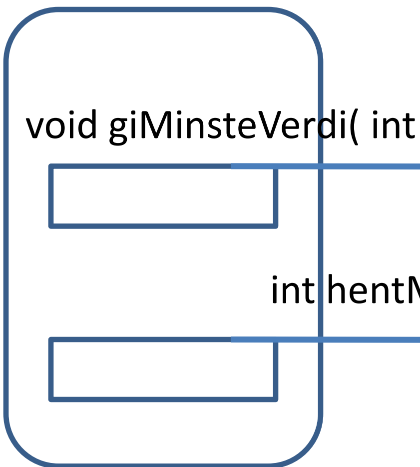
K: konstante data (immutable)

Monitor er ikke noe ord i Java

Eksempel på parallellisering: Finn minste tall i tabell

(samme teknikk for å finne sum / gjennomsnitt)

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



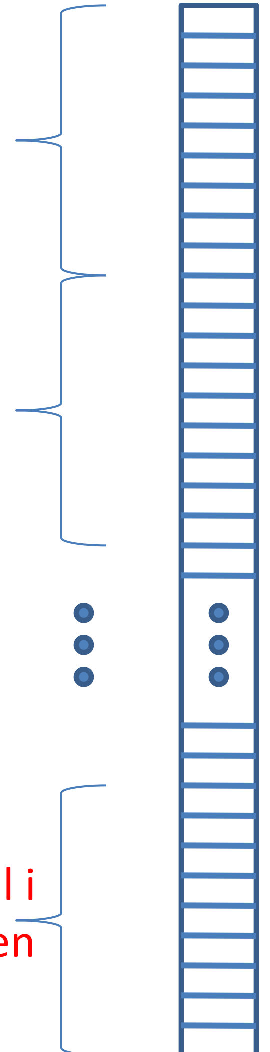
Objekt av klassen MinstMonitor

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



UNIVERSITETET I OSLO

BARE FANTASIEN SETTER GRENSER

Institutt for informatikk

Objekt av klassen MinstMonitorC

minstTilNa

void giMinsteVerdi(int verdi)

<LÅS MONITOREN>
If (verdi < minstTilNa) minstTilNa = verdi;
If (<alle tråden har gitt sine svar>) <SIGNAL>;
<LÅS OPP>

Kalles av lete-trådene

void vent()

<LÅS MONITOREN>
while(<ikke alle trådene er ferdig>) <AWAIT>;
<LÅS OPP>

Kalles av main

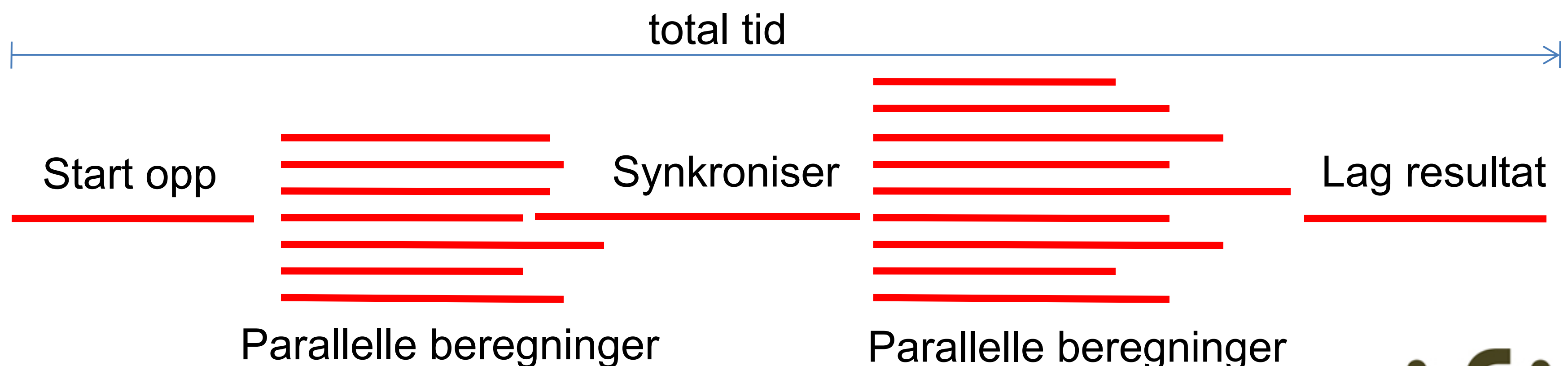
int hentMinste()

return minstTilNa

vent og hentMinste kunne vært slått sammen

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



UNIVERSITETET I OSLO

Institutt for informatikk

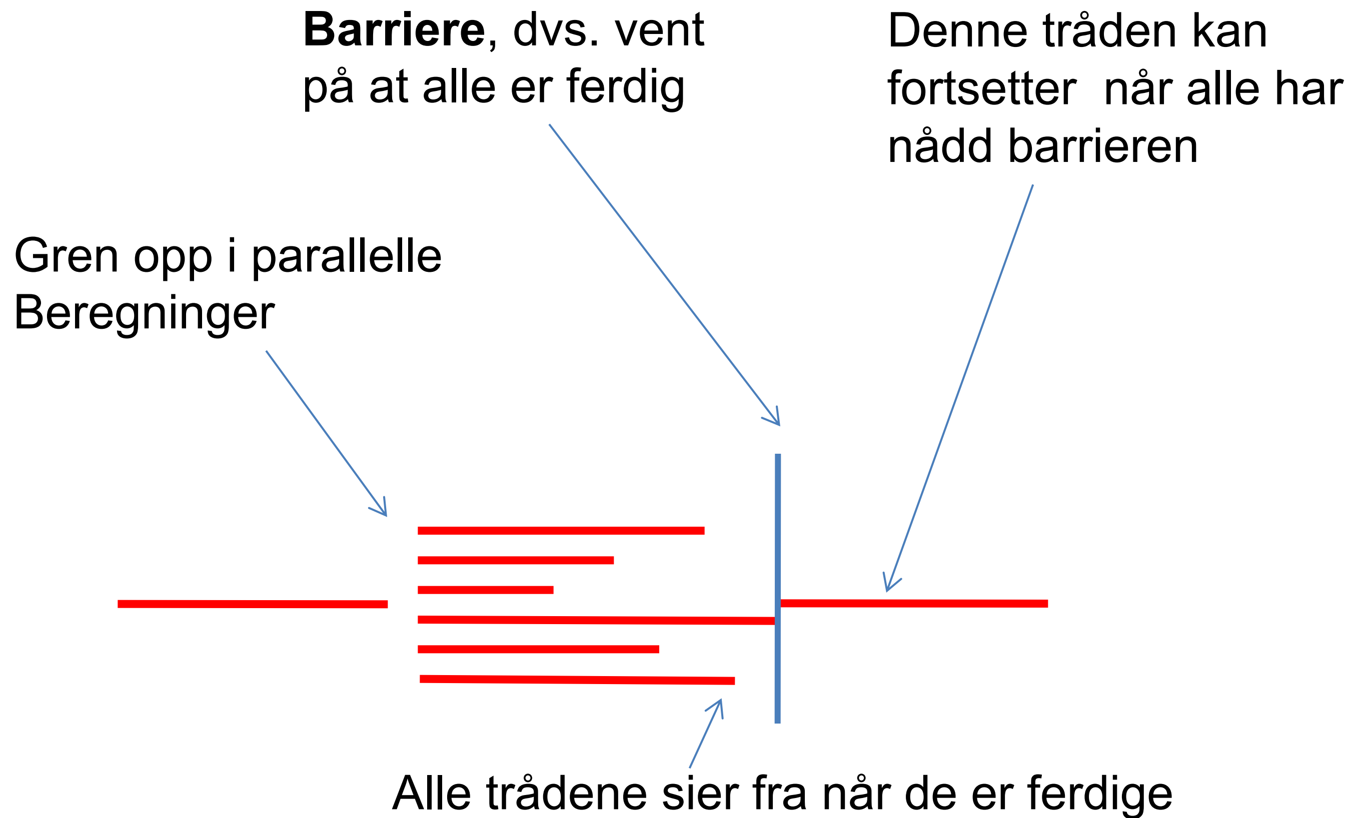
Amdahls lov og ”finn minste tall”

- Totaltid:
  - Tiden det tar å lage og sette i gang 64 tråder
    - (kan det gjøres i parallell ?) ($\log_2 64 = 6$) * Tegn opp.
  - 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 $\log_2 64 = 6$

2, 4, 8, 16, 32, 64

Barrierer i parallellprogrammering



Barrierer i Java

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

Barrieren

```
CountDownLatch minBariere =
    new CountDownLatch(6)
```

```
minBarriere.await();
```

```
minBarriere.countDown();
```

tid

Barrieren i dette eksemplet

Start opp

Gren opp i parallelle
beregninger

Alle trådene sier fra
når de er ferdige
(har nådd **barrieren)**

Denne tråden kan
fortsetter når alle har
nådd barrieren.
Da er det endelige
resultatet klart.

Barieren

Alle trådene avsluttes med
å kalle metoden giMinsteVerdi
i monitoren

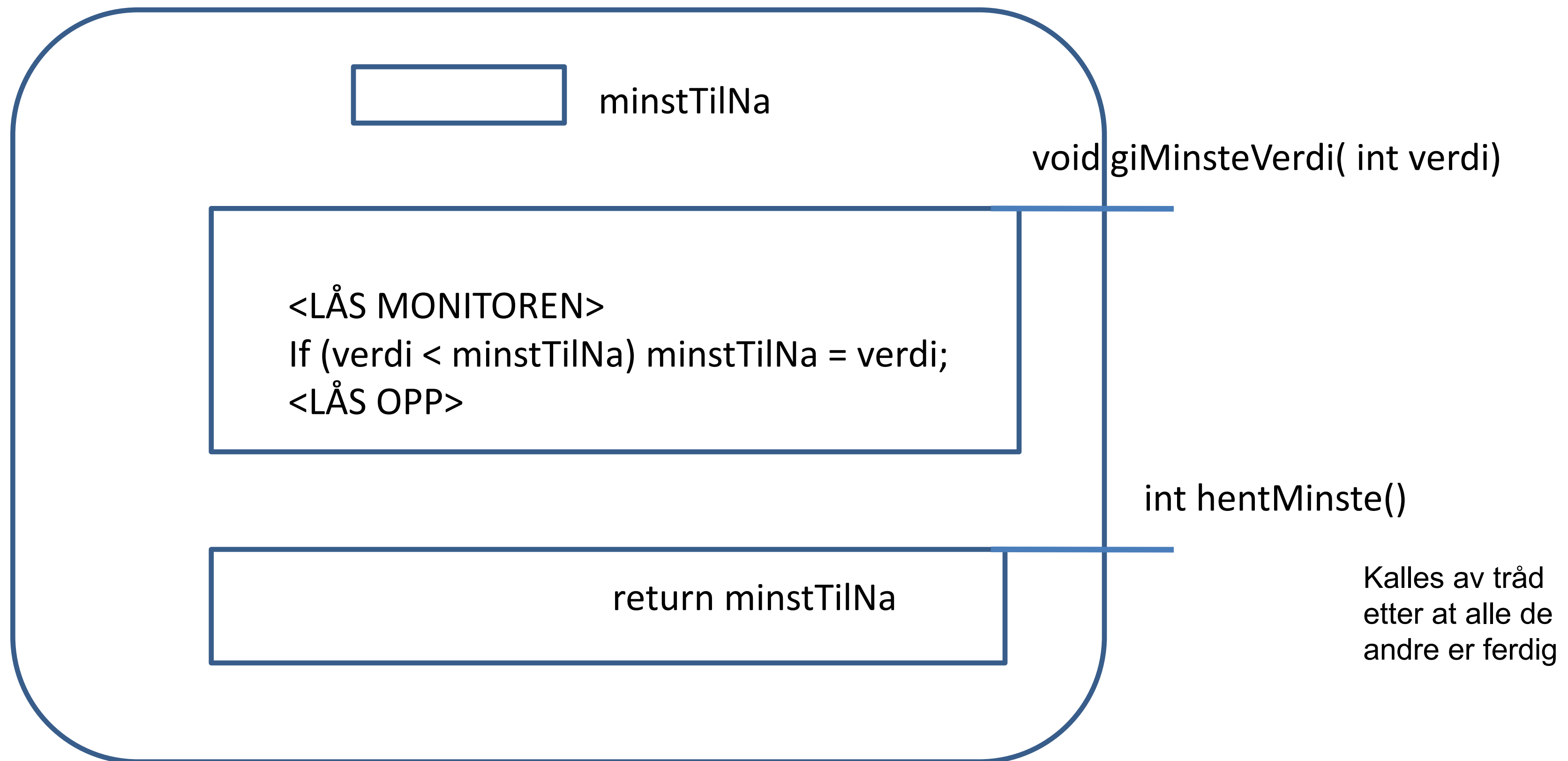
tid

UNIVERSITETET
I OSLO

Kallet **`monitor.vent();`**
i MinstC-programmet er en
hjemmelaget barriere

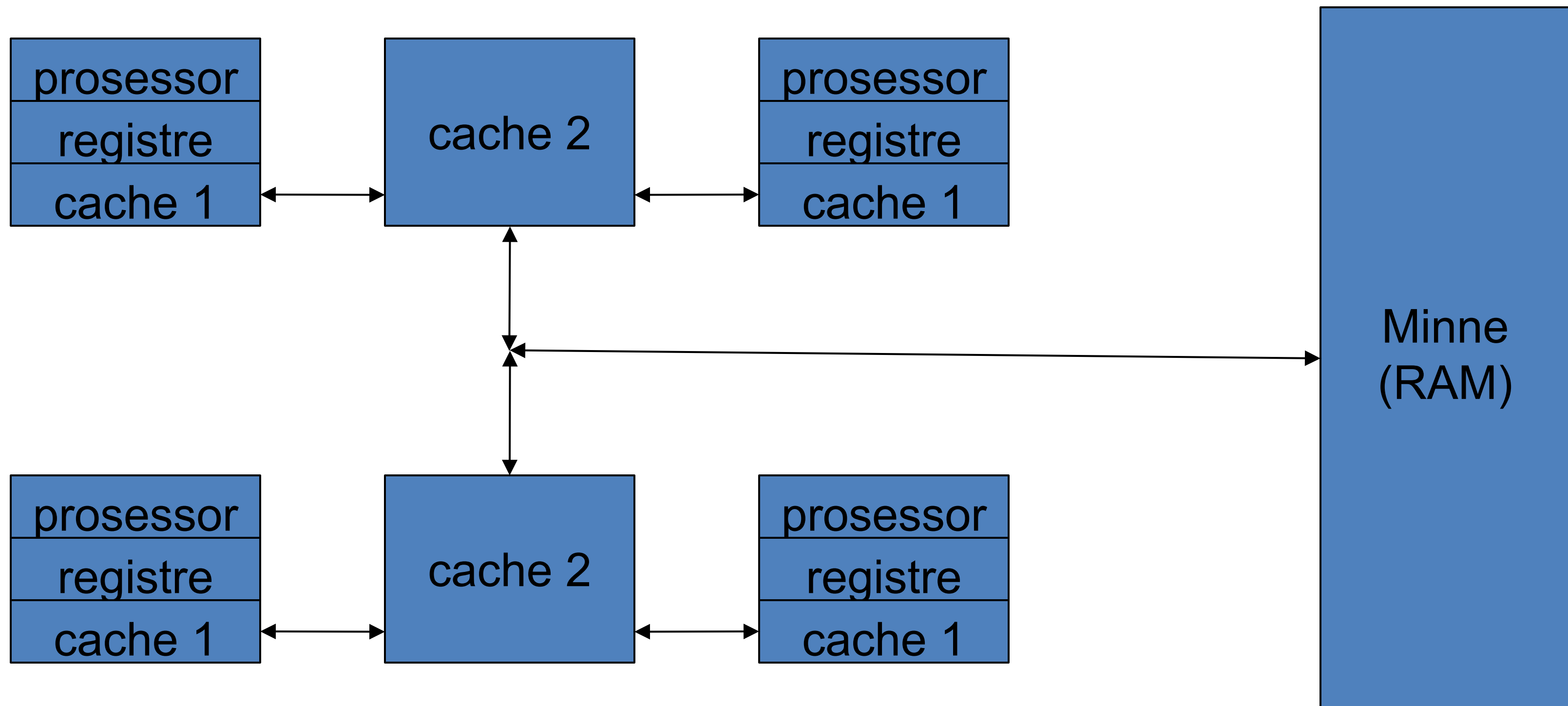
Institutt for informatikk

Objekt av klassen FinnMinstMonitorCB



Refaktorering gir en mye enklere monitor fordi nå skjer signaleringen og ventingen ved hjelp av en barriere i trådene.

Maskinarkitektur, f.eks. 4 kjerner (repetisjon)



Volatile variable

- En variabel som er deklarert volatile caches ikke (og oppbevares ikke i registre (lenger enn helt nødvending)).
  - En volatil variable skrives helt tilbake i primærlageret.

- boolean stopp = false;

  En tråd: stopp = true;

  En annen tråd:

  ```
  while(! stopp) {
  }
  ```

  Må da deklarere:

  volatile boolean stopp=false;

Synkronisering og felles variable i en monitor

- This means that any memory operations which were visible to a thread before exiting a synchronized block are visible to any thread after it enters a synchronized block protected by the same monitor, since all the memory operations happen before the release, and the release happens before the acquire.

From: JSR 133 (Java Memory Model) FAQ
Jeremy Manson and Brian Goetz, February 2004

Litt på siden av pensum i INF1010.
Mest ment som en forsmak for de som er mer interessert.

Fordelen med konstanter (immutable objects)

- Konstanter kan det aldri skrives til
- Minnemodellen for konstanter blir derfor veldig enkel.
- 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.

To typer invarianter (invariante tilstandspåstander):

- Når du lager en løkke er det alltid en invariant som sier hvor langt arbeidet i løkka er kommet
- Data i et objekt er alltid styrt av en (eller flere) invarianter eller konsistensregler

- Nå skal vi se på den siste typen invarianter
  (mer om den første typen invarianter senere i INF1010)

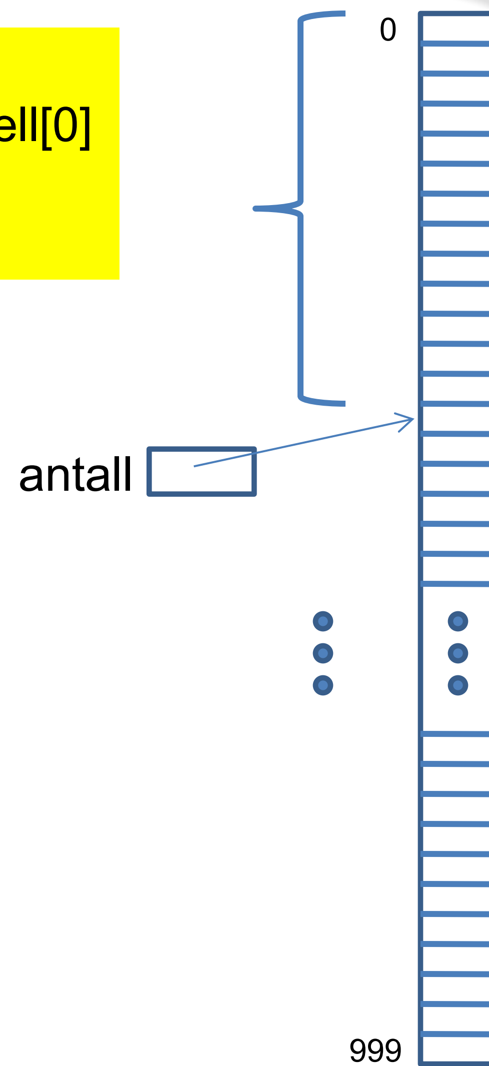
Invarianter på data i objekter

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]);
}
```



Overbevis deg (og andre) om at metodene bevarer Invarianten !!!!!!!!! (og at den er sann ved oppstart) !!!!!!

Vi tar opp tråden (☺) fra sidene foran og fra forrige gang:

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.*

venter på å komme inn første gang

void setInn(int verdi)

```
<lås>
if  (antall == 1000)  vent-til: Ikke-Full();
antall++;
:
signal: Ikke-Tom();
<lås opp>
```

int taUt()

```
<lås>
If (antall == 0)  vent-til: Ikke-Tom();
antall --;
:
signal: Ikke-Full();
<lås opp>
```

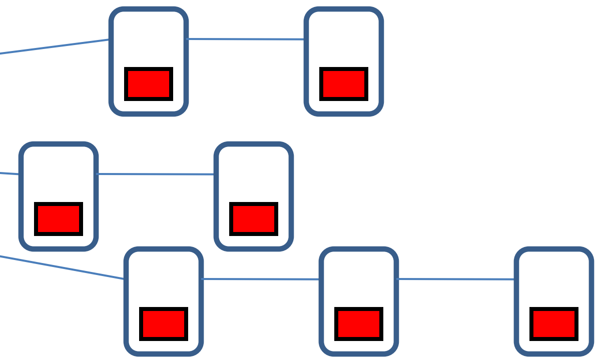
Invariant:
Alle dataene vi lagrer ligger i tabell[0] til og med tabell [antall – 1] og
0 <= antall <= 1000

En monitor (et objekt)

```
Lock laas = new ReentrantLock();
Condition ikkeFull =  laas.newCondition();
Condition ikkeTom = laas.newCondition();

void settInn ( int verdi) throws InterrupedException
    laas.lock();
    try {
      while  (full)  ikkeFull.await();          // OK med if ?
      // nå er det helst sikkert ikke fult
      :
      // det er lagt inn noe, så det er helt sikkert ikke tomt:
     ikkeTom.signal();
    }  finally {
     laas.unlock();
    }

int taUt ( )    throws InterrupedException
    laas.lock();
    try {
      while  (tom)  ikkeTom.await();          // OK med if ?
      // nå er det helst sikkert ikke tomt;
      :
      // det er det tatt ut noe, så det er helt sikkert ikke fult:
     ikkeFull.signal();
    }  finally {
     laas.unlock();
    }
```



En kø for selve låsen og en kø for hver condition-variabel

Nytt tema
(men fortsatt tråder)

Vranglås
Engelsk: Deadlock

Horstmann kap. 20.5

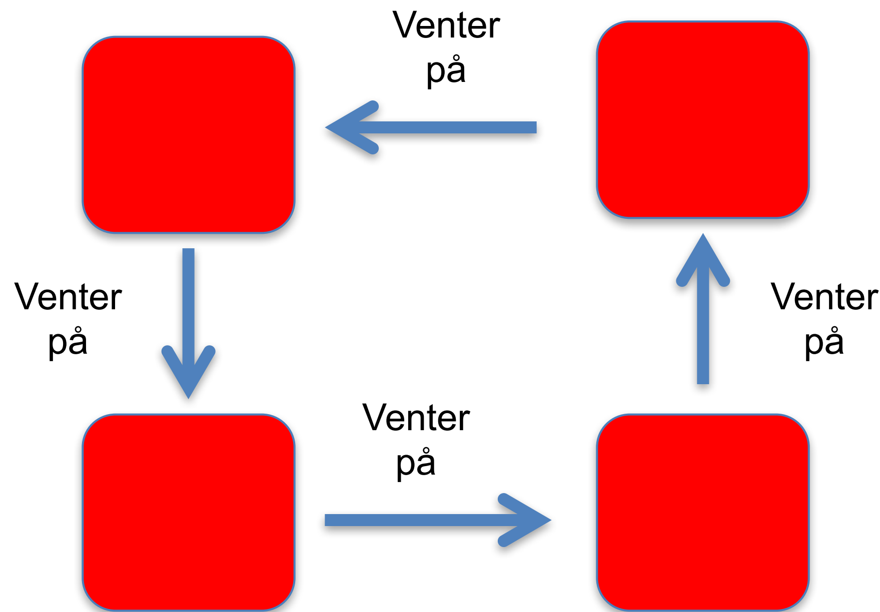
20

Mer om tråder: Vranglås (deadlock)

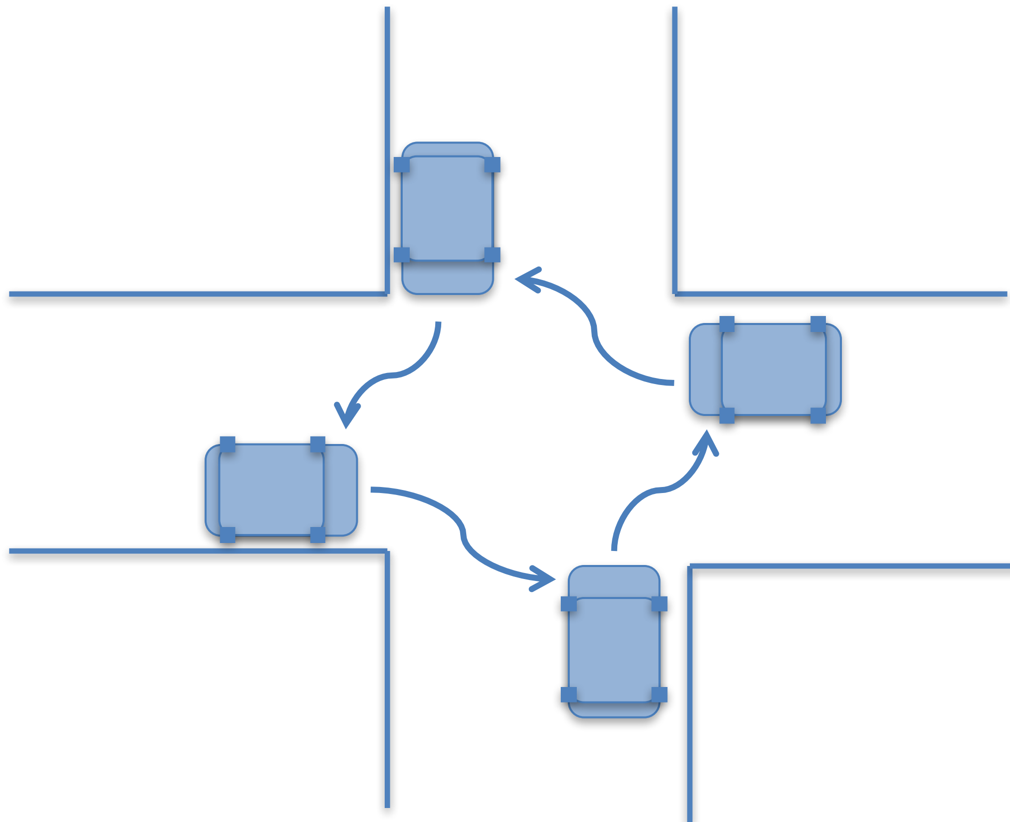
- Vranglås skjer når flere tråder venter på hverandre i en sykel:
- Eksempel
  - Veikryss: alle bilene skal stoppe for biler fra høyre -> Alle stopper = VRANGLÅS

Vranglås betyr

- Flere tråder venter på hverandre
- Syklisk venting



Hvordan kan dette skje?



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 . . .

24

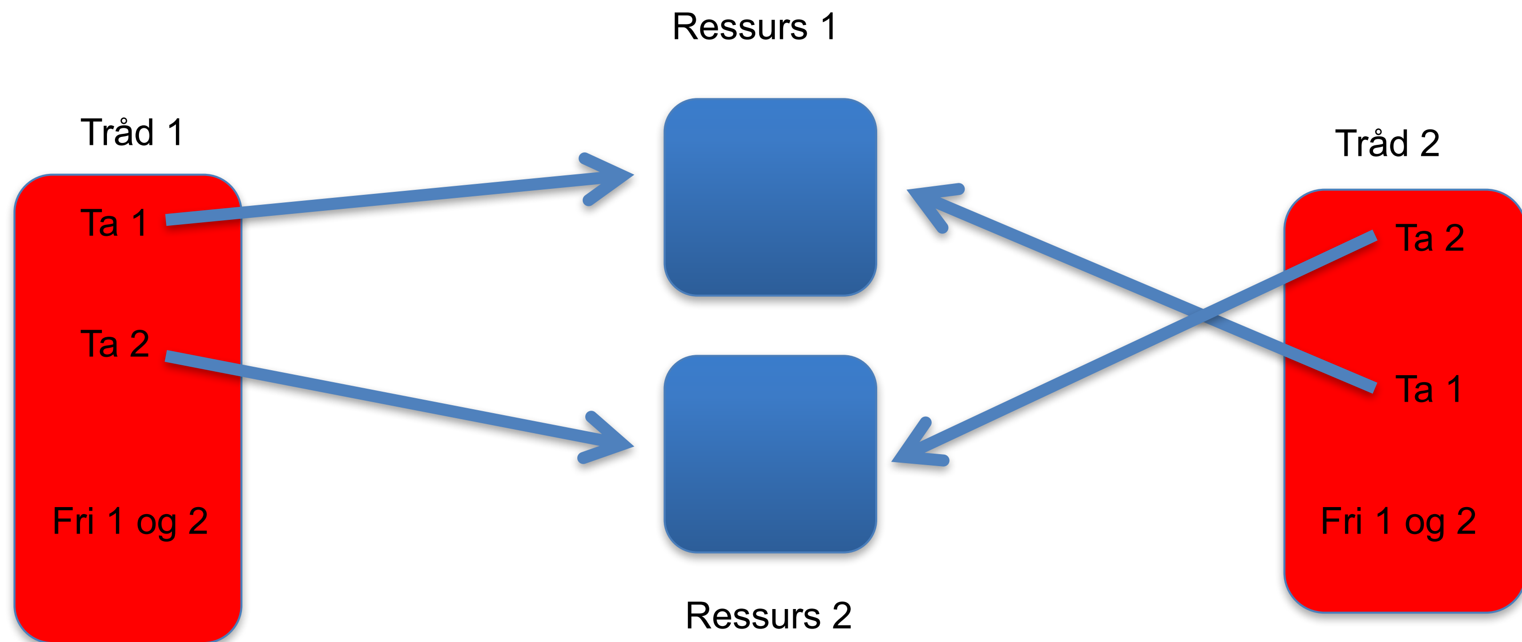
Unngå vranglås

1. Ta bare en ressurs
2. Ta alle 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

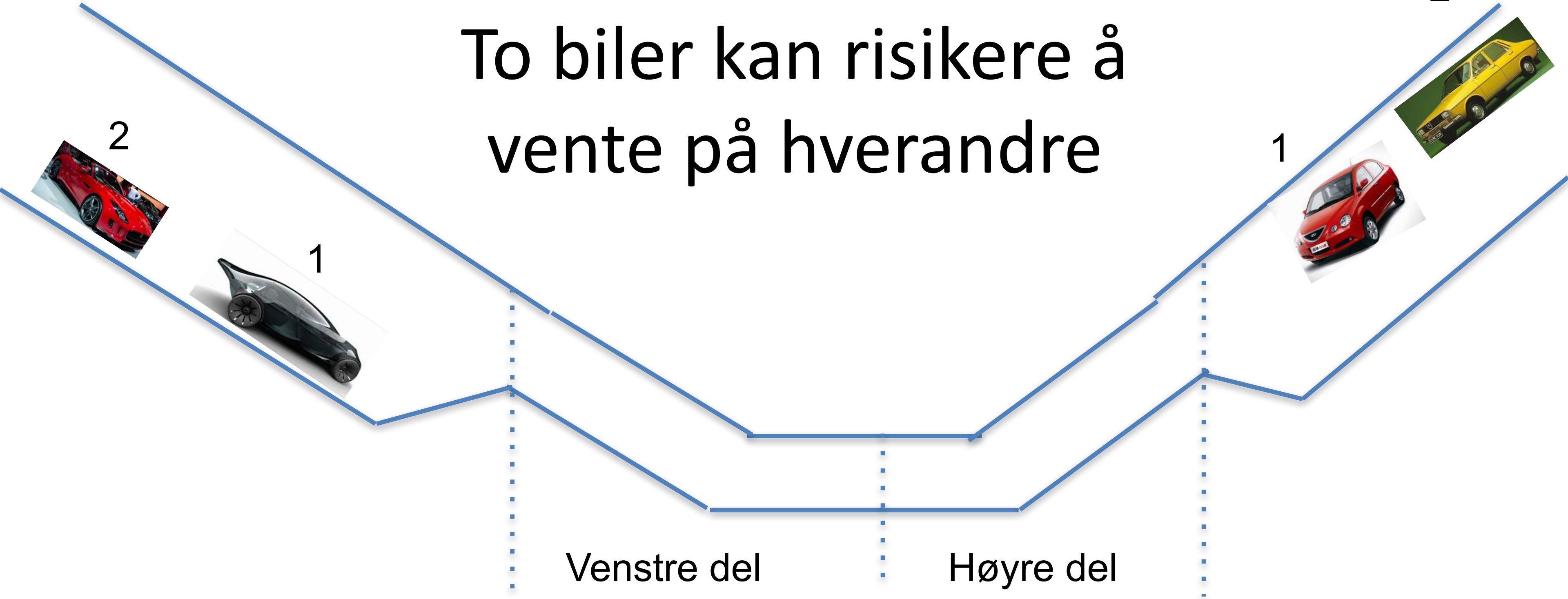
Enkleste eksempel på vranglås

- To tråder
- To ressurser som tas i forskjellig rekkefølge



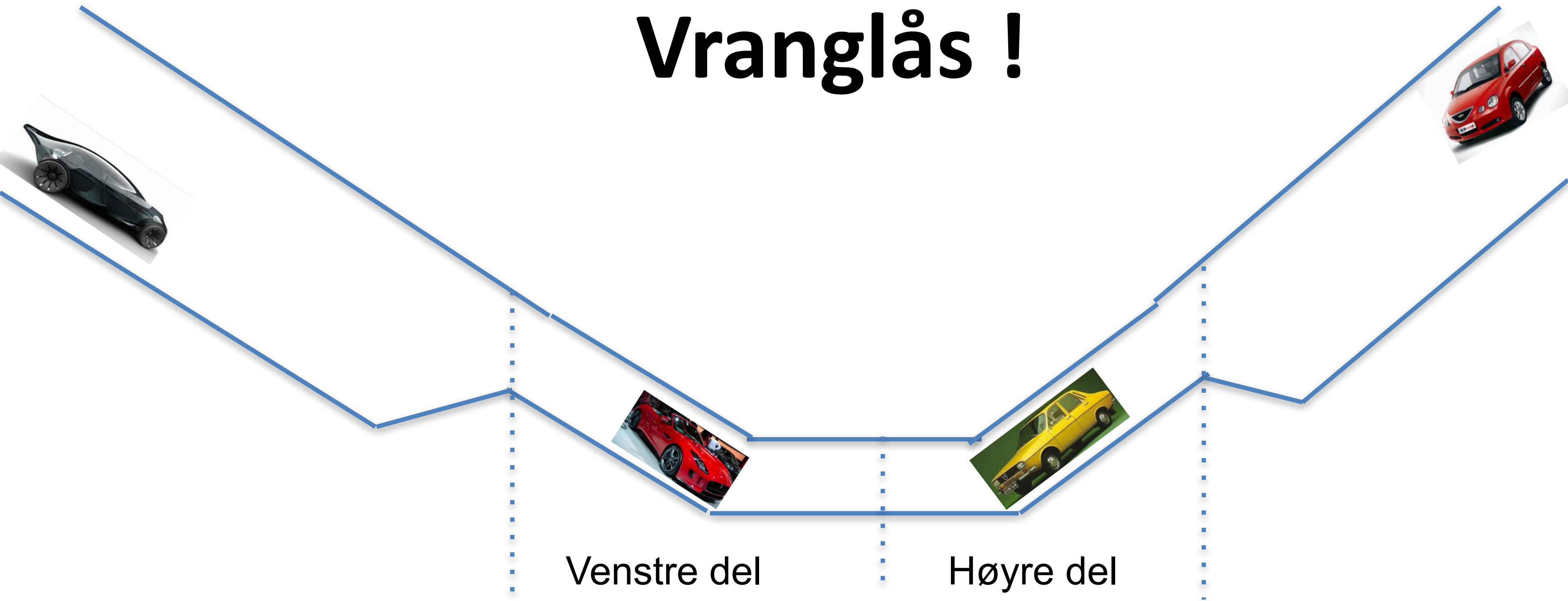
Enkleste eksempel på vranglås:

To biler kan risikere å vente på hverandre



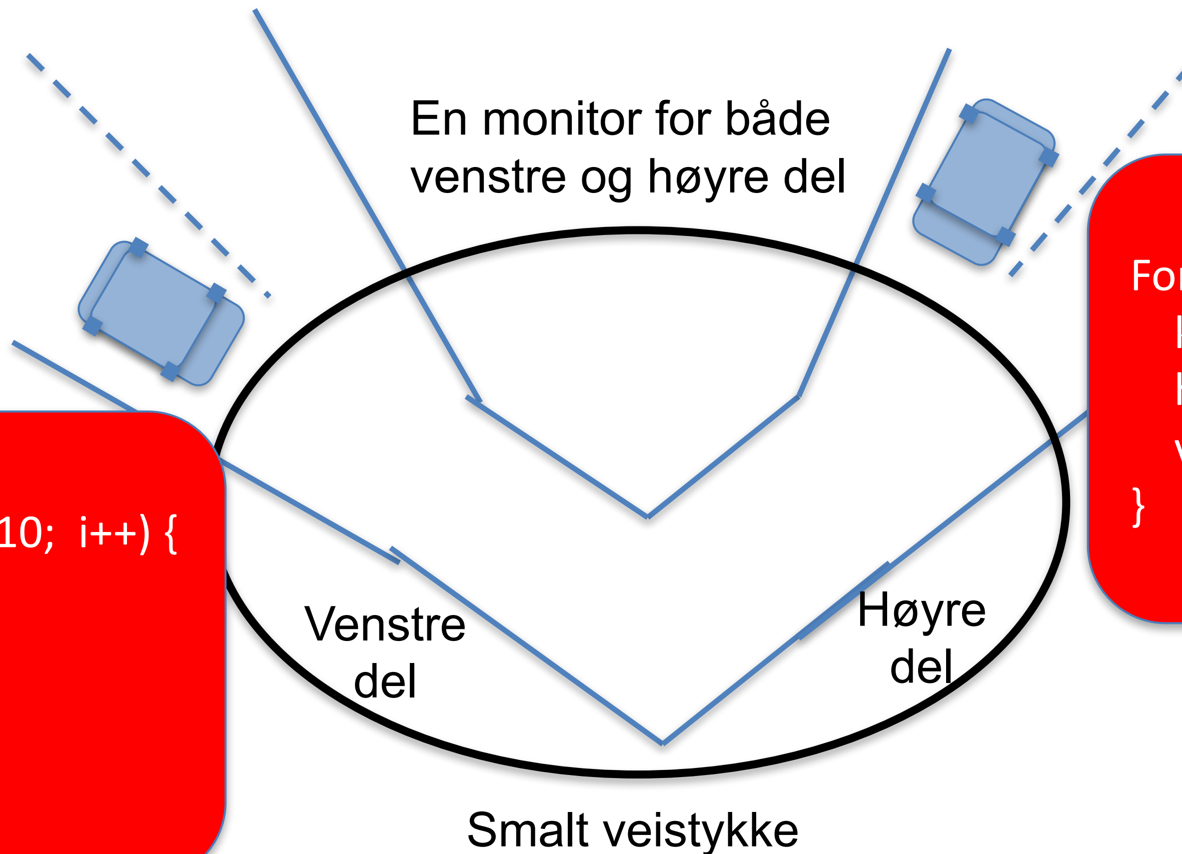
Smalt veistykke, to biler
kan ikke passere hverandre.
Bilene kan bare se den første delen.

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.

Eksempel
Program 1




```
For (i=0 ; i < 10;  i++) {
    kjør bil fra
    venstre  til
    høyre
}
```

```
For (i=0 ; i < 10;  i++) {
    kjør bil fra
    høyre til
    venstre
}
```


29

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

To monitorer, en for venstre og en høyre

Venstre del

Høyre del

Smalt veistykke