IN1010 – våren 2018

Repetisjon av tråder

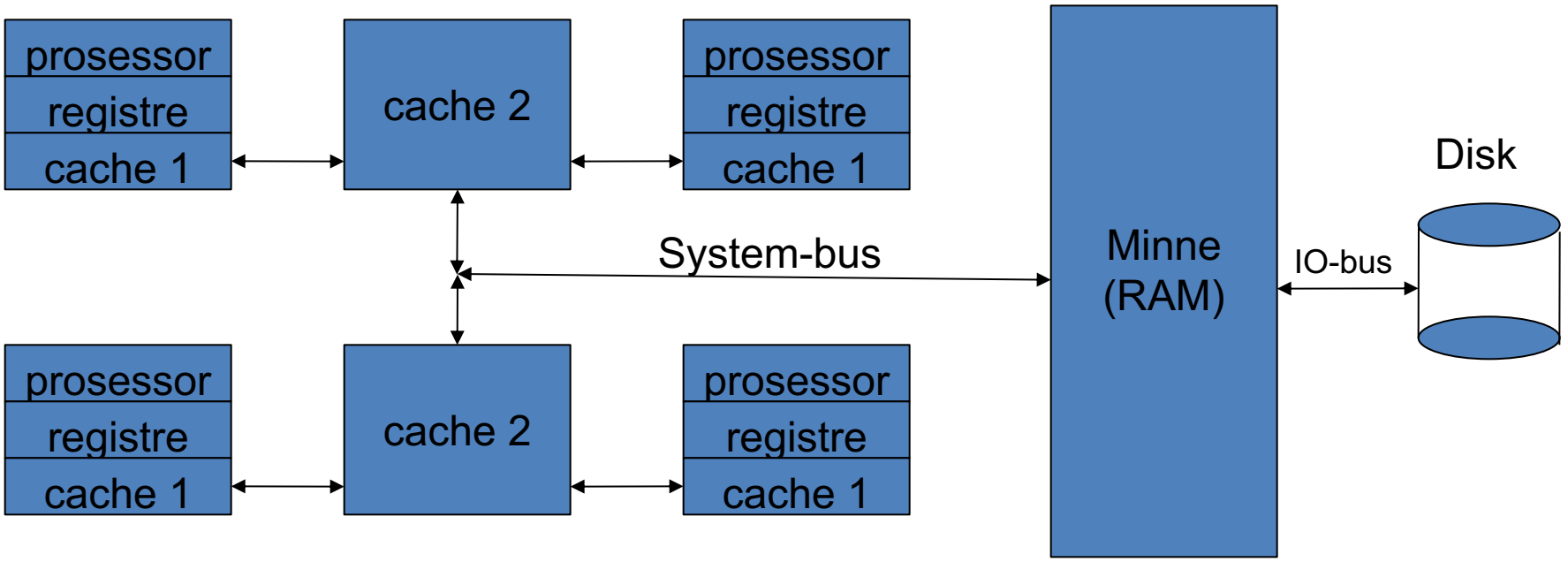
15. mai 2018

Stein Gjessing,

Institutt for informatikk,
Universitetet i Oslo

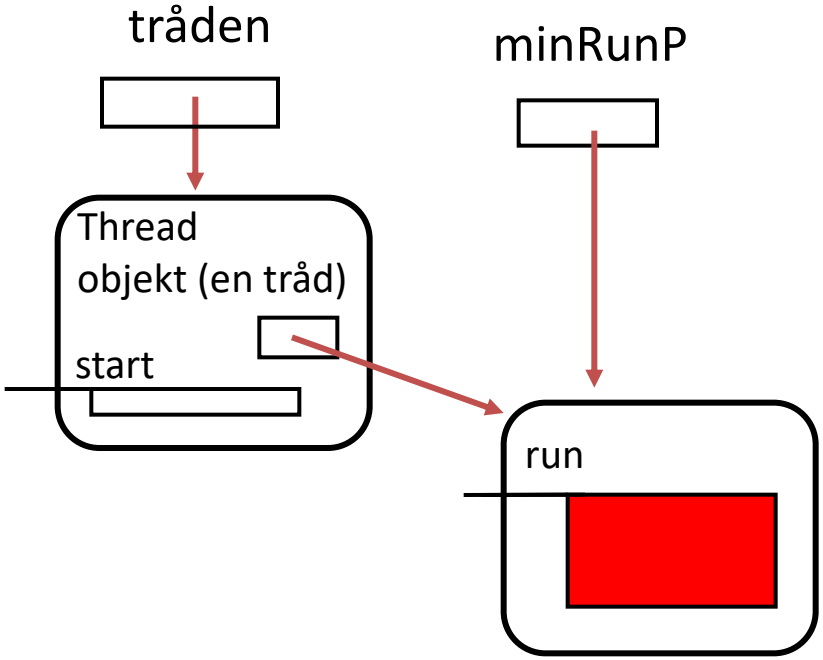
Tråder

Datamaskinarkitektur



Tråder i Java I

```
class MinRun implements Runnable {
   <datastruktur>
   public void run( ) {
     while (<mer å gjøre>) {
           <gjør noe>;
           . . .
     }
  }
}
```



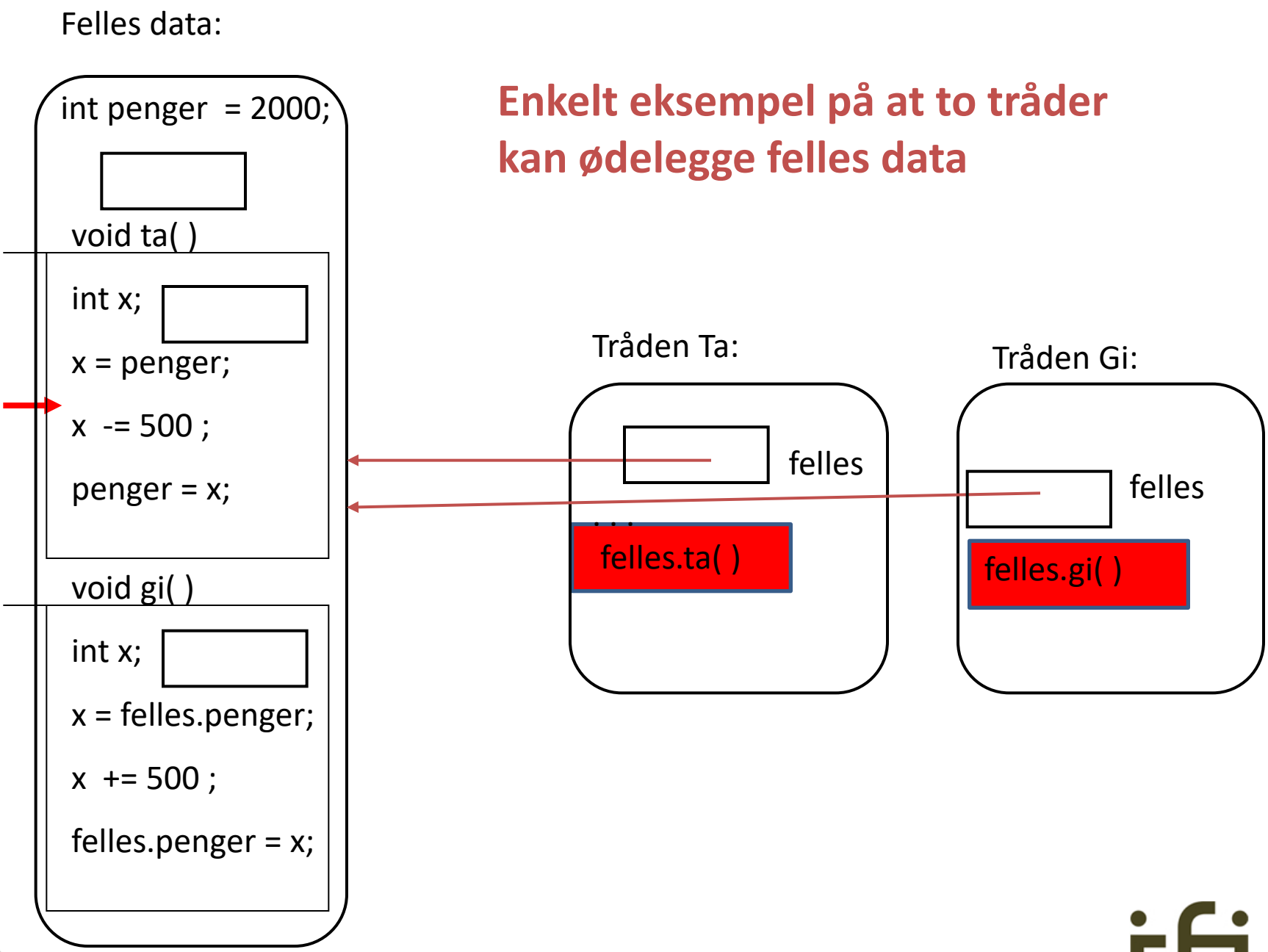
En tråd lages og startes opp slik:

```
Runnable minRunP = new MinRun();
Thread tråden = new Thread(minRunP);
tråden.start( );
```

Her går den nye og den gamle tråden (dette programmet), videre hver for seg

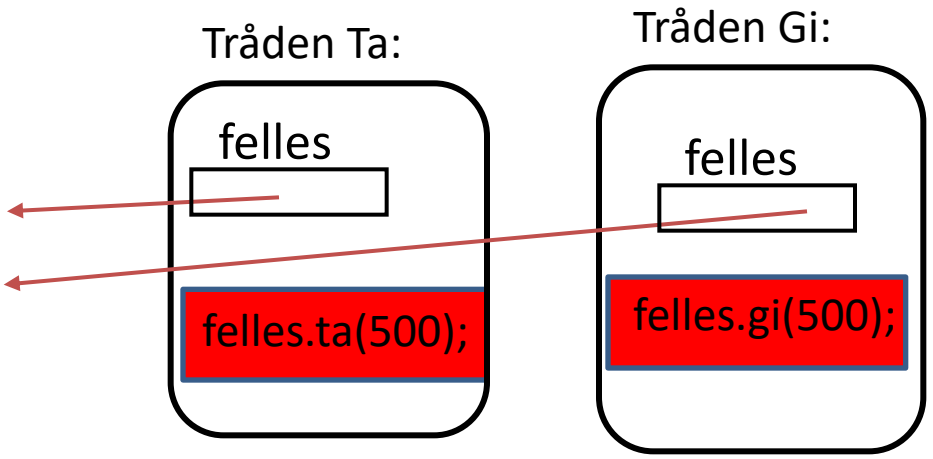
start( ) er en metode i Thread som må kalles opp for å få startet tråden.
start-metoden vil igjen kalle metoden run (som vi selv programmerer).

Enkelt eksempel på at to tråder kan ødelegge felles data



Kritiske regioner / monitor

```
Lock laas = new ReentrantLock();

Int penger;  2000

public void gi ( int verdi ) throws InterrupedException
    laas.lock();
    try {
        penger = penger + verdi;
    }  finally {
      laas.unlock()
    }

public void  ta ( int verdi ) throws InterrupedException
    laas.lock();
    try {
        penger = penger - verdi;
    }  finally {
      laas.unlock()
    }
```



En lås (laas) slik at bare en tråd kommer inn i monitoren om gangen

(import java.concurrent.locks.*)

Legg merke til bruken av finally

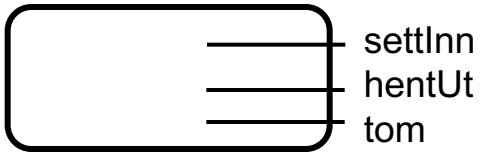
```
void  putInn (int verdi)  throws InterrupedException {
   laas.lock();
   try {
      :
      :
    }  finally {
     laas.unlock();
    }
}
```

Da blir laas.unlock() **alltid** utført !!

Monitor:
Vente på at en beholder ikke er tom lenger

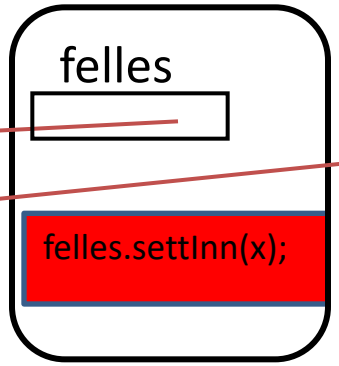
```
Lock laas = new ReentrantLock();
Condition ikkeTom =  laas.newCondition();

Beholder behold;
```

Objekt av klassen Beholder

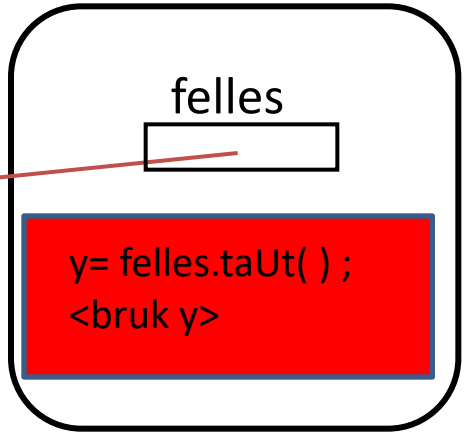



```
public void settInn ( Ting tingen ) throws InterrupedException
    laas.lock();
    try {
        behold.settInn(tingen);
        ikkeFull.signalAll();
    }  finally {
      laas.unlock()
    }

public Ting taUt ( ) throws InterrupedException
    laas.lock();
    try { if (behold.tom())
            { ikkeTom.await(); }
     return (behold.hentUt());
    }  finally {
      laas.unlock()
    }
```

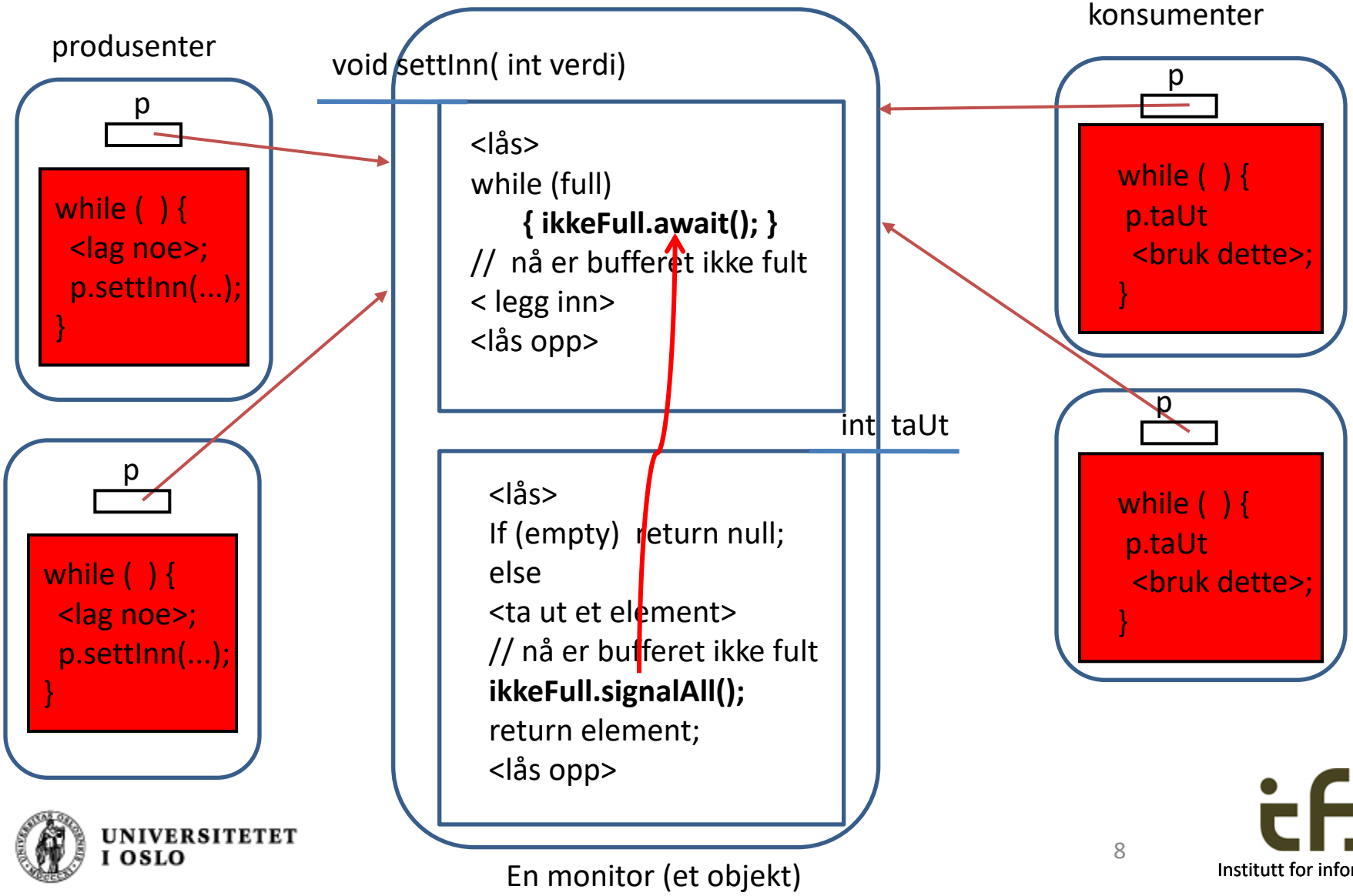
Tråden Ta:



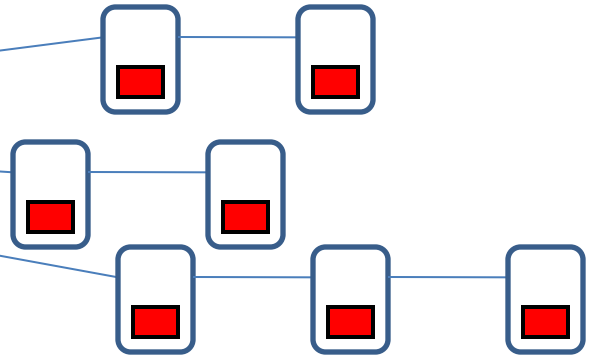
Tråden Gi:



Produsenter og konsumenter



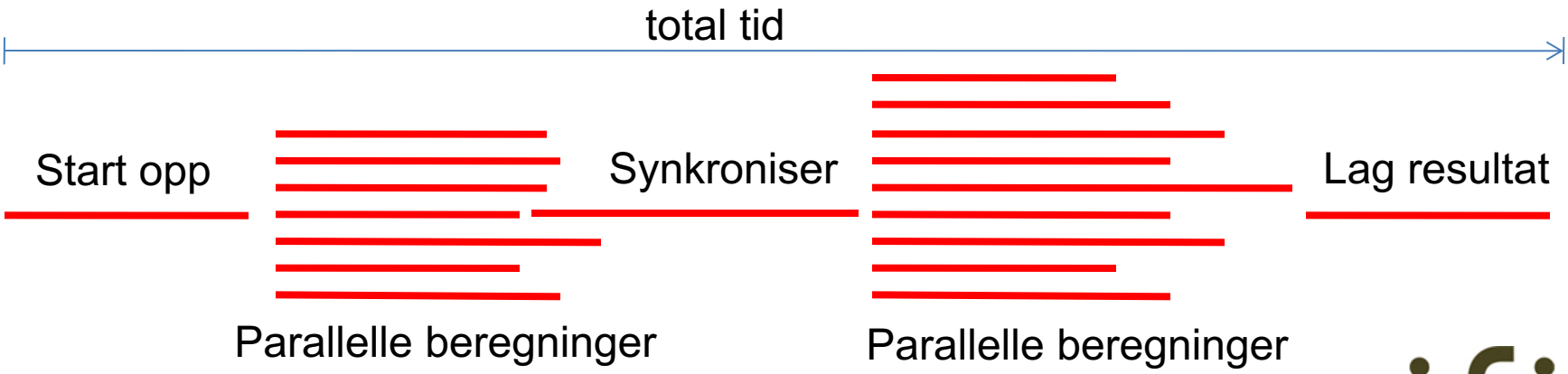
```
Lock laas = new ReentrantLock();
Condition ikkeFull =  laas.newCondition();
Condition ikkeTom = laas.newCondition();

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

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



Ofte har vi flere grunner til å vente i en monitor

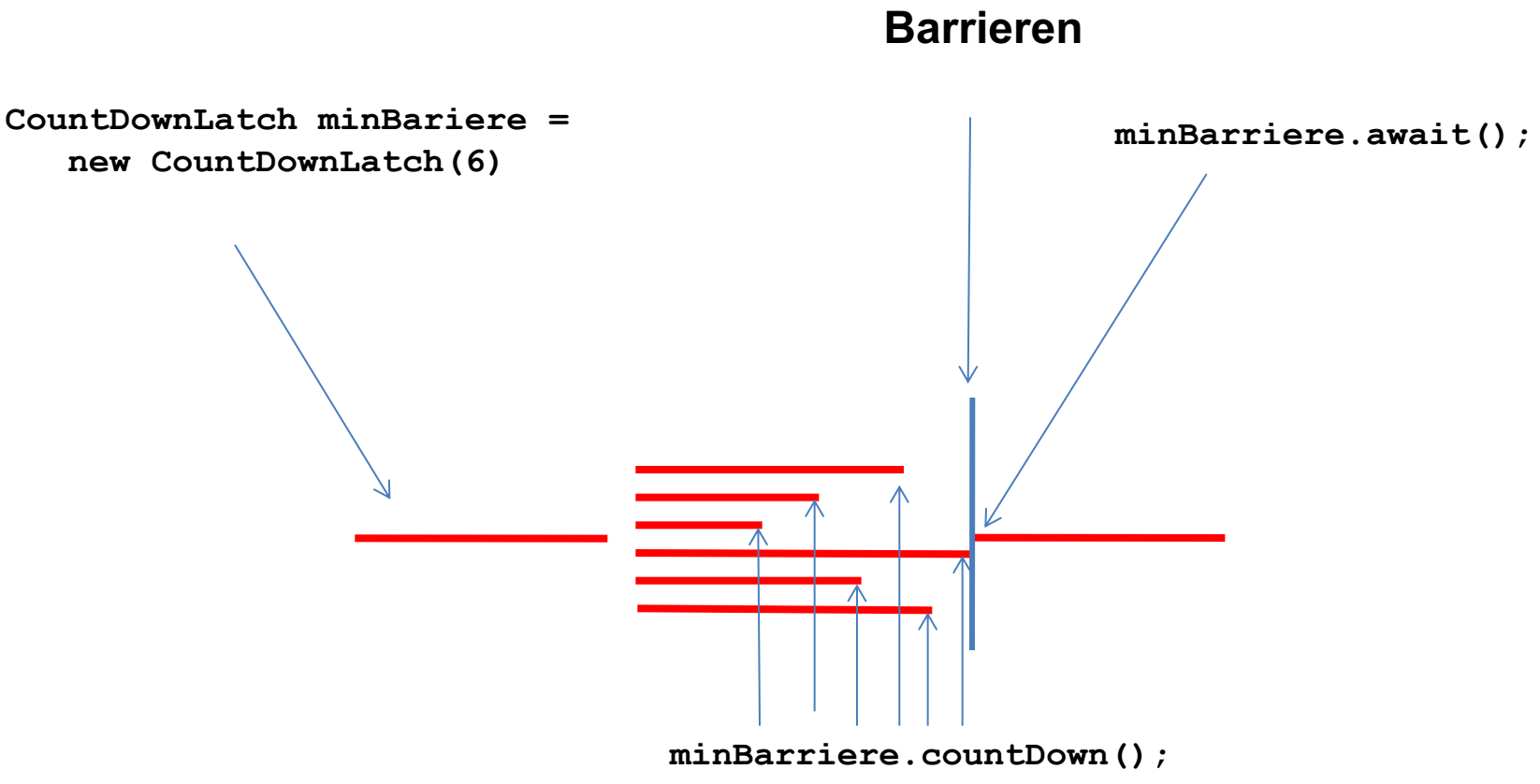
Amdahls lov

- 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



Barrierer i Java

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



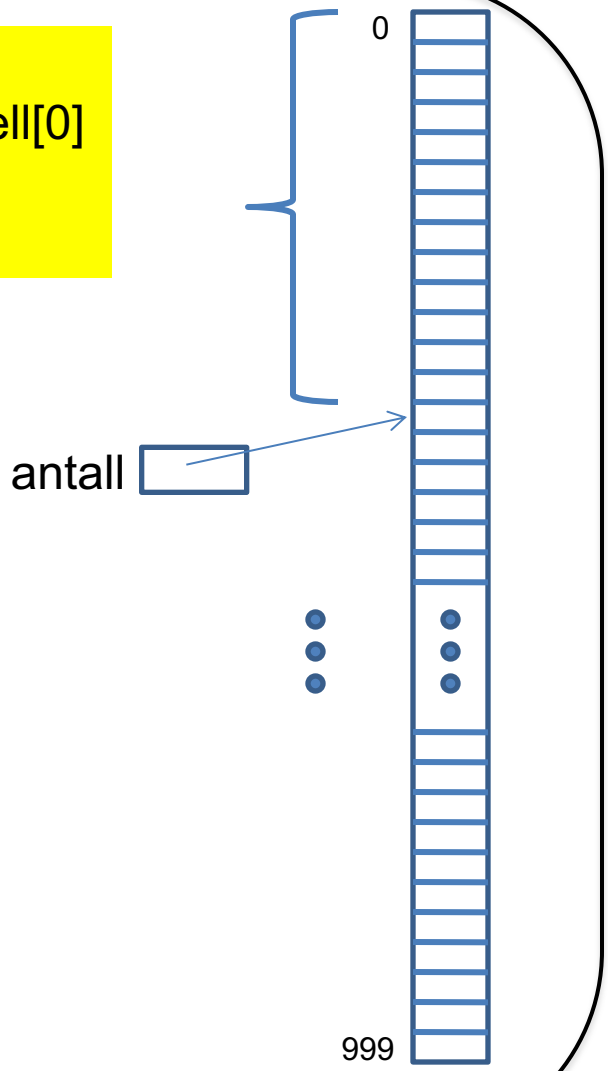
Invarianter på data i objekter

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;
}
```

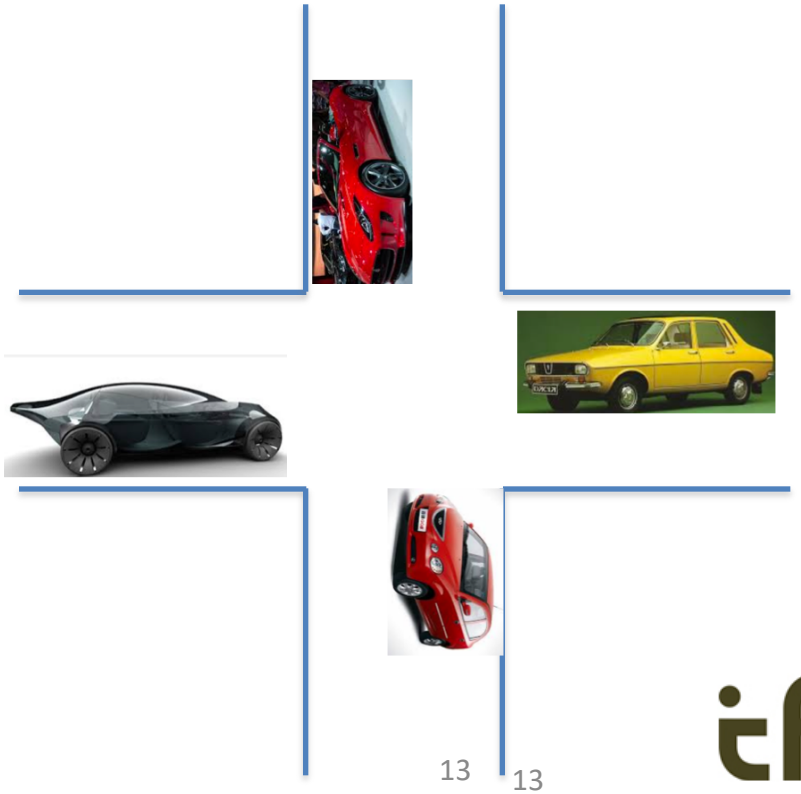
```
taUt ( ) {
    if (antall == 0) return null;
    antall --;
    return (tabell[antall]);
}
```



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

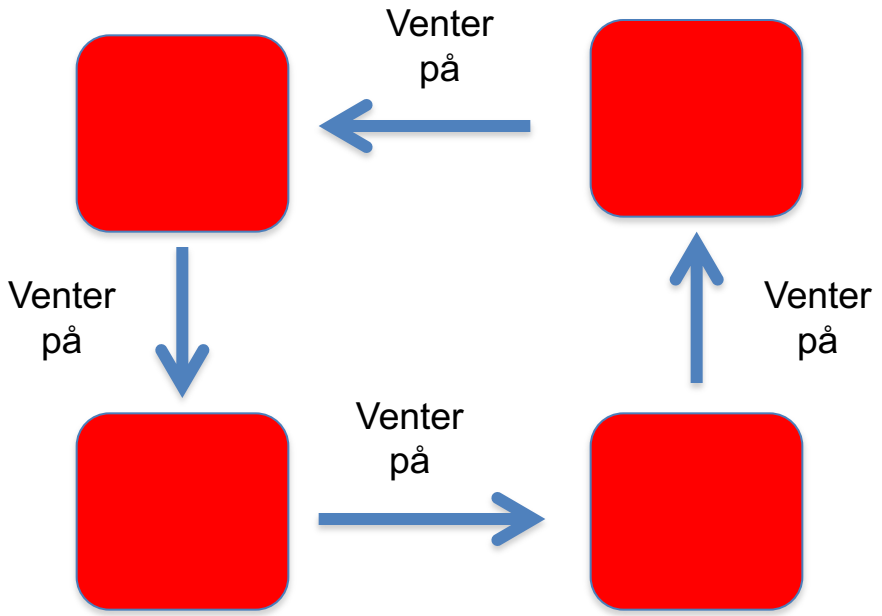
Vranglås (deadlock)

- 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



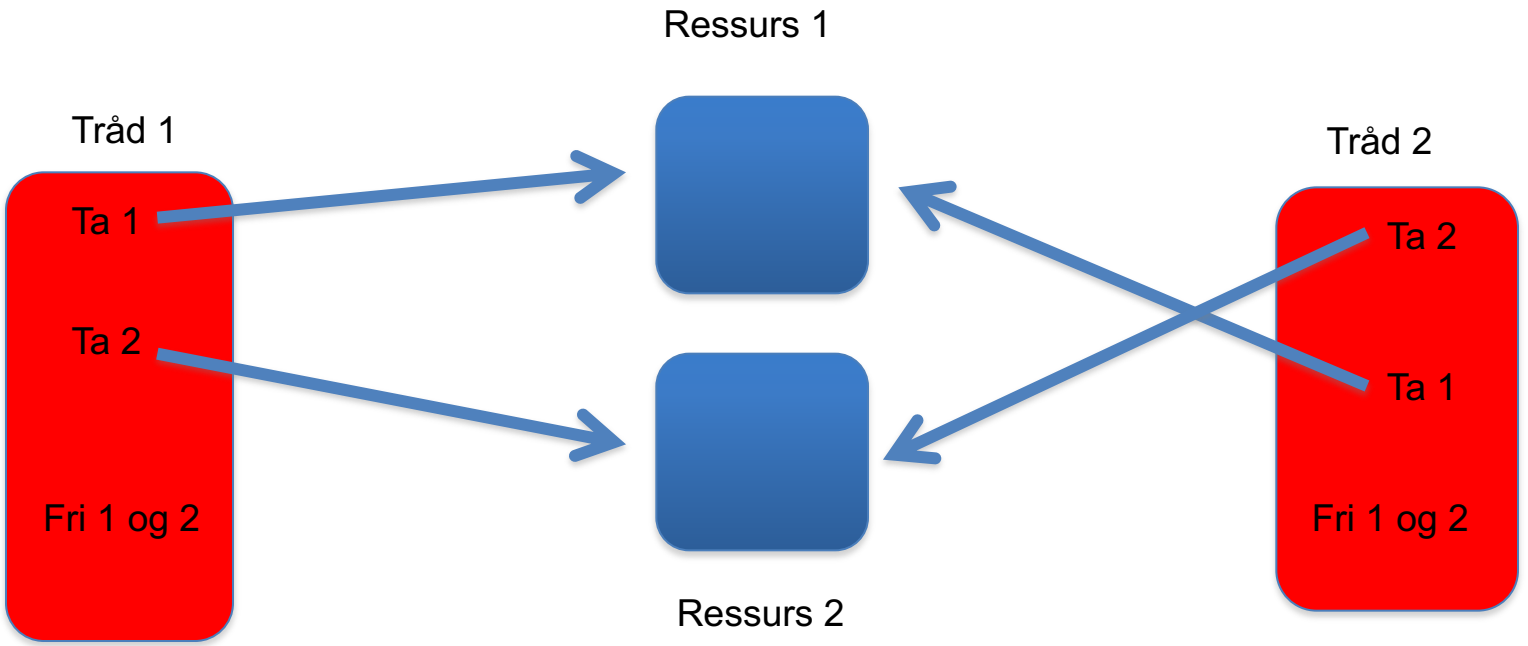
Unngå vranglås

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

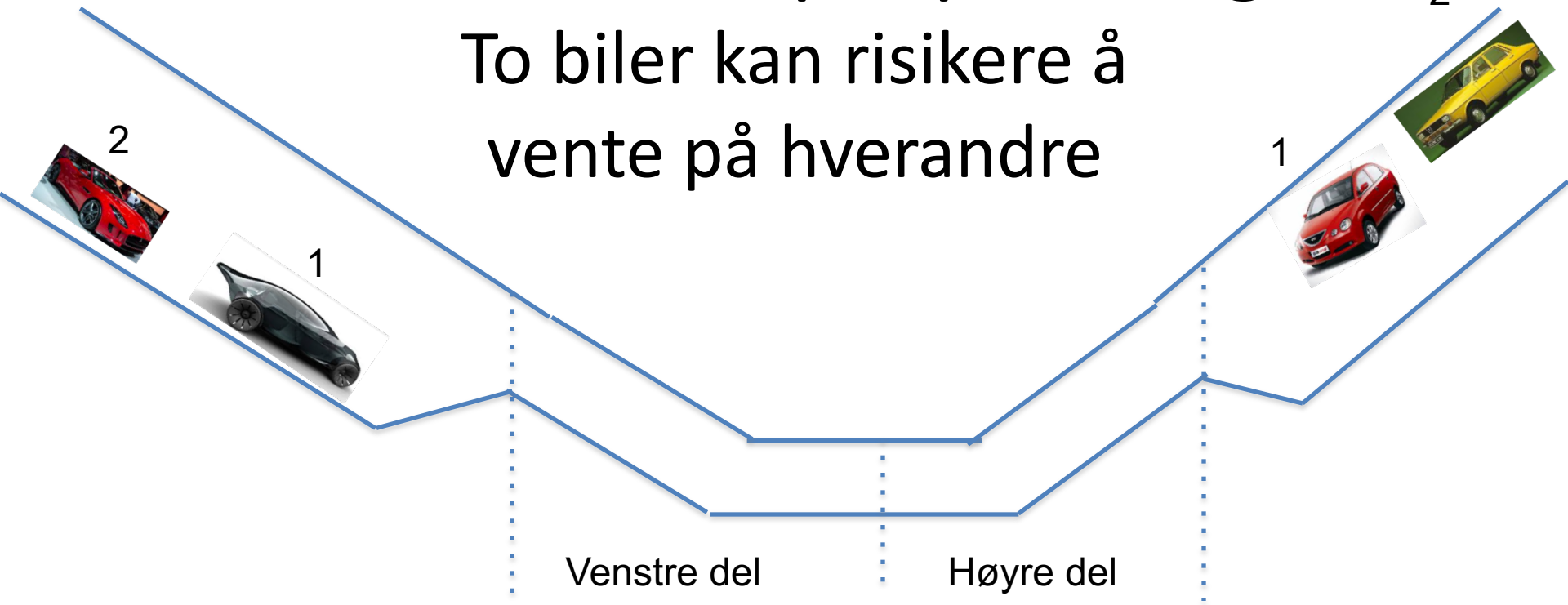
Enkleste eksempel på vranglås

- To tråder
- To ressurser som tas i forskjellig rekkefølge



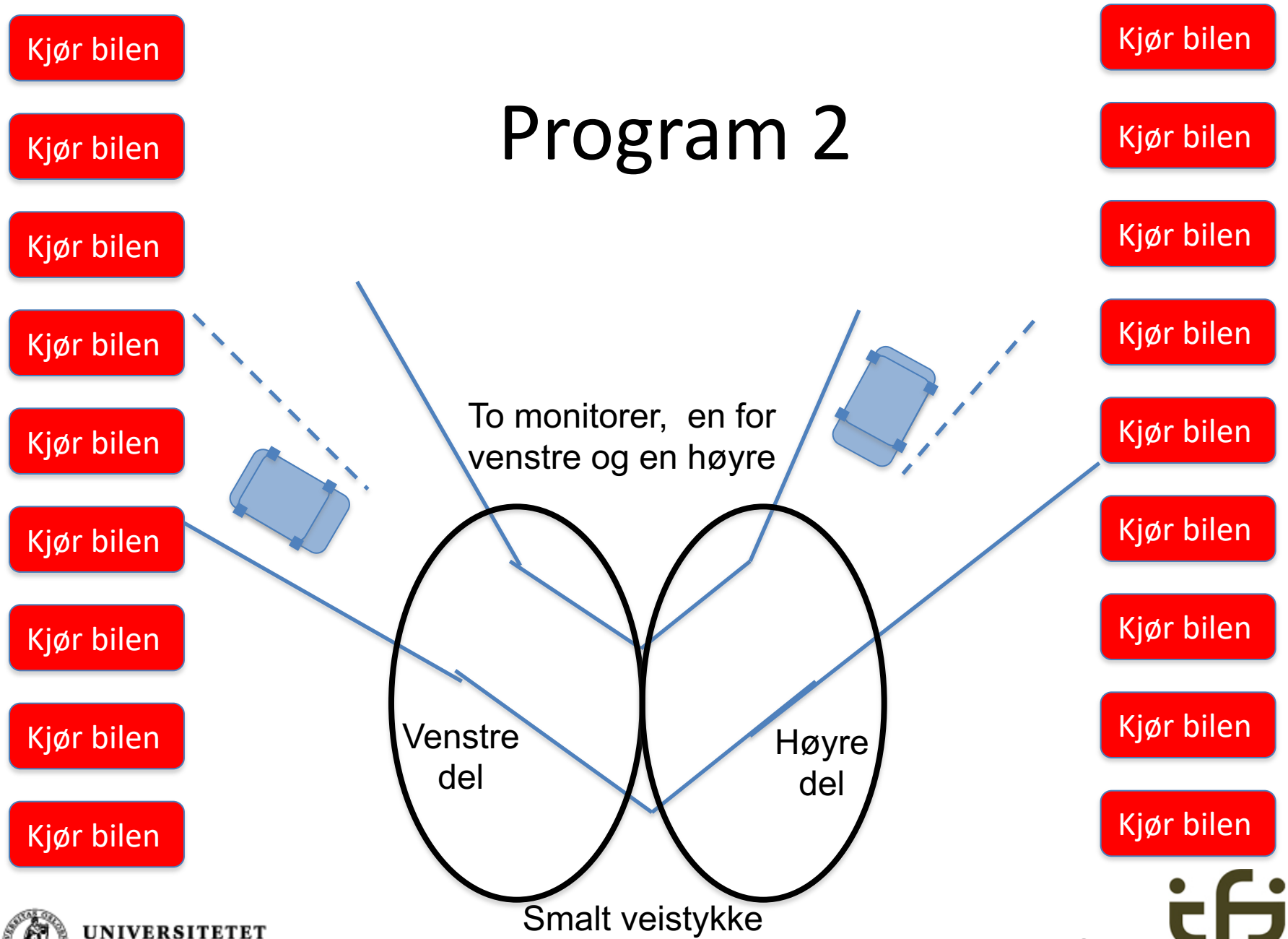
Enkleste eksempel på vranglås:

To biler kan risikere å vente på hverandre



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

Program 2


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