Parallellitet og tråder (del 2)

IN1010, 7. april 2026

Magnus Skjegstad

(basert på materiale fra Stein Gjessing)

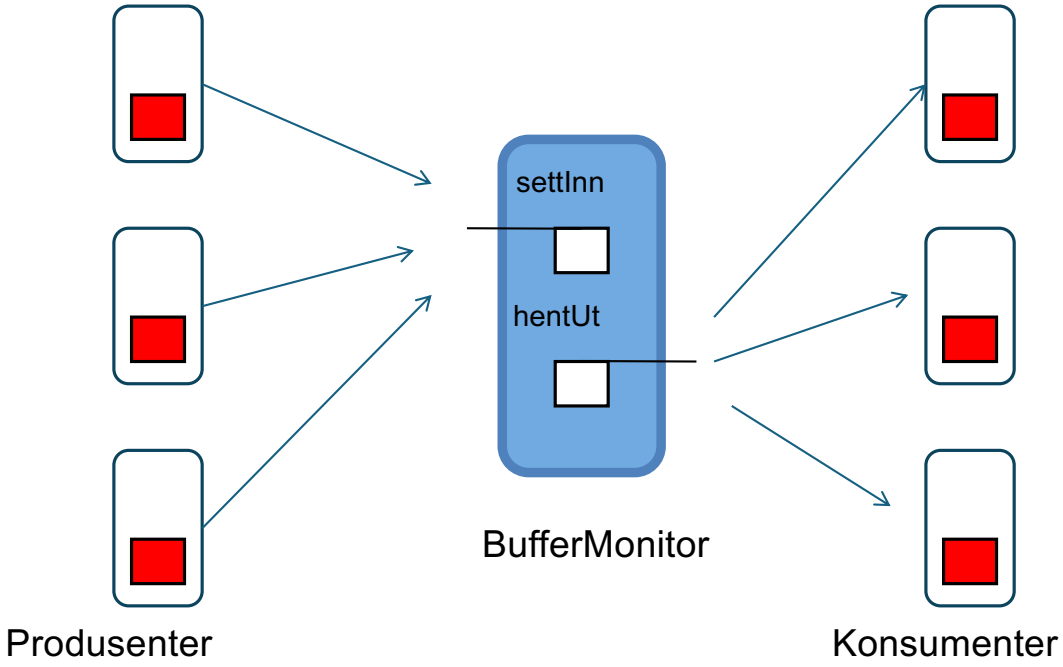
Horstmann, hele kap 20 («Tråder i Java»)

Agenda

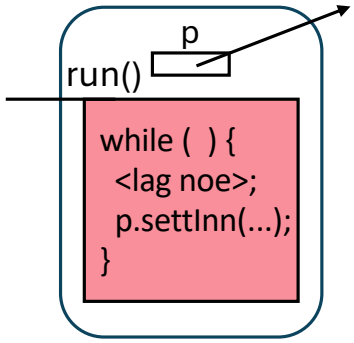
- Litt repetisjon om produsenter og konsumenter
  - Hva venter vi på (og hvorfor) i en monitor.
- Klassisk eksempel: Kokk og servitør
- Amdahls lov
- Join og barrierer (i Java)
  - Eksempel: Finn minste tall
- Vranglås (Deadlock)
- Noen tips til innlevering 5

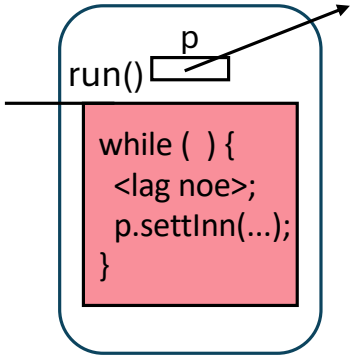
Repetisjon. Det viktigste i IN1010 om tråder.

Kommunikasjon mellom tråder: En felles beholder = en monitor



Huskeregler for tråder





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

Ingen låsing, venting eller signalering i trådene utenom monitorene!

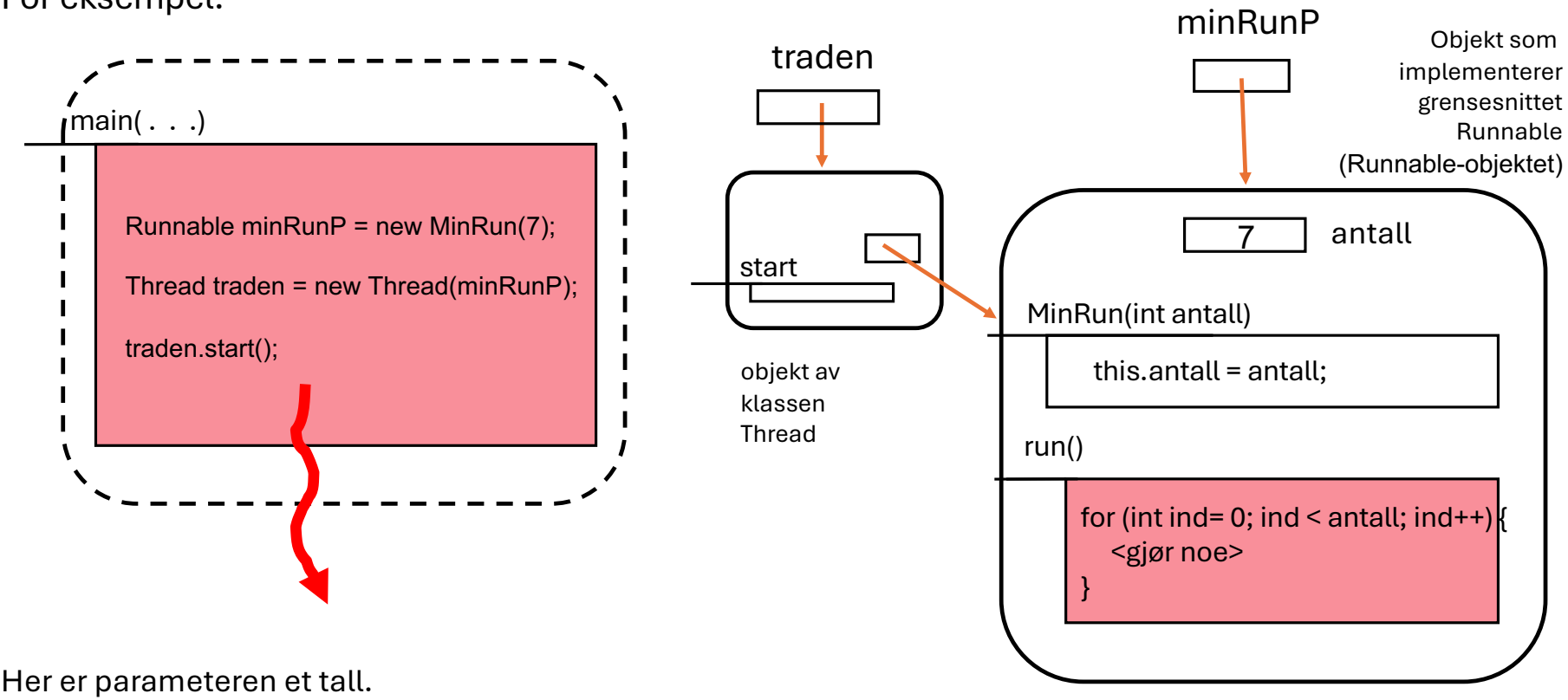
Alle låsing, venting og signalering skjer i inne monitorene

Men en tråd kan si Thread.sleep().
Og en tråd kan kalle interrupt() og join() i andre tråder
(og senere i dag skal vi se at de kan kommunisere via barrierer)

En tråd kan stoppe opp en stund når som helst.

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

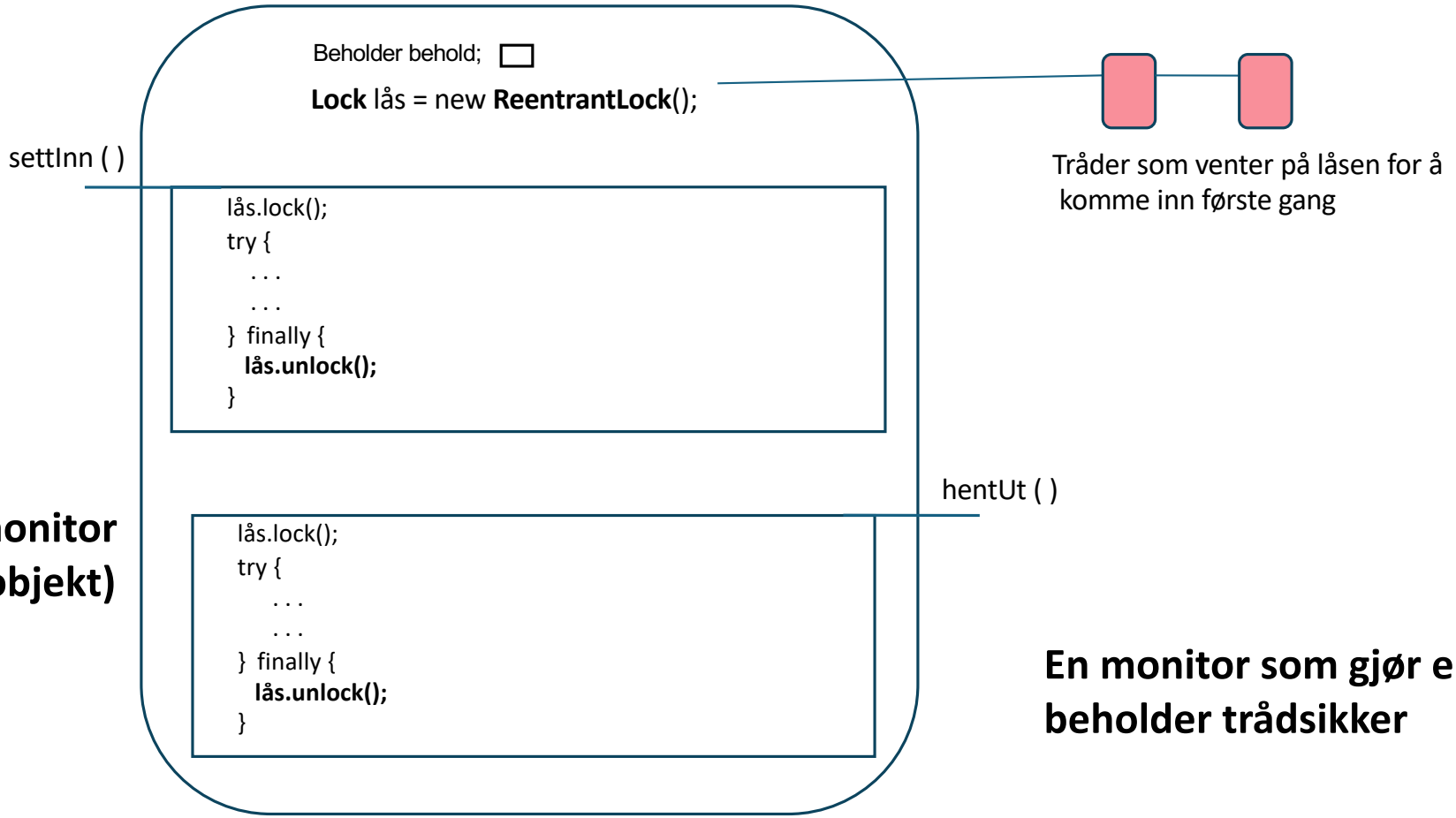
For eksempel:



Her er parameteren et tall.
Også vanlig: En referanse til en monitor

Repetisjon:

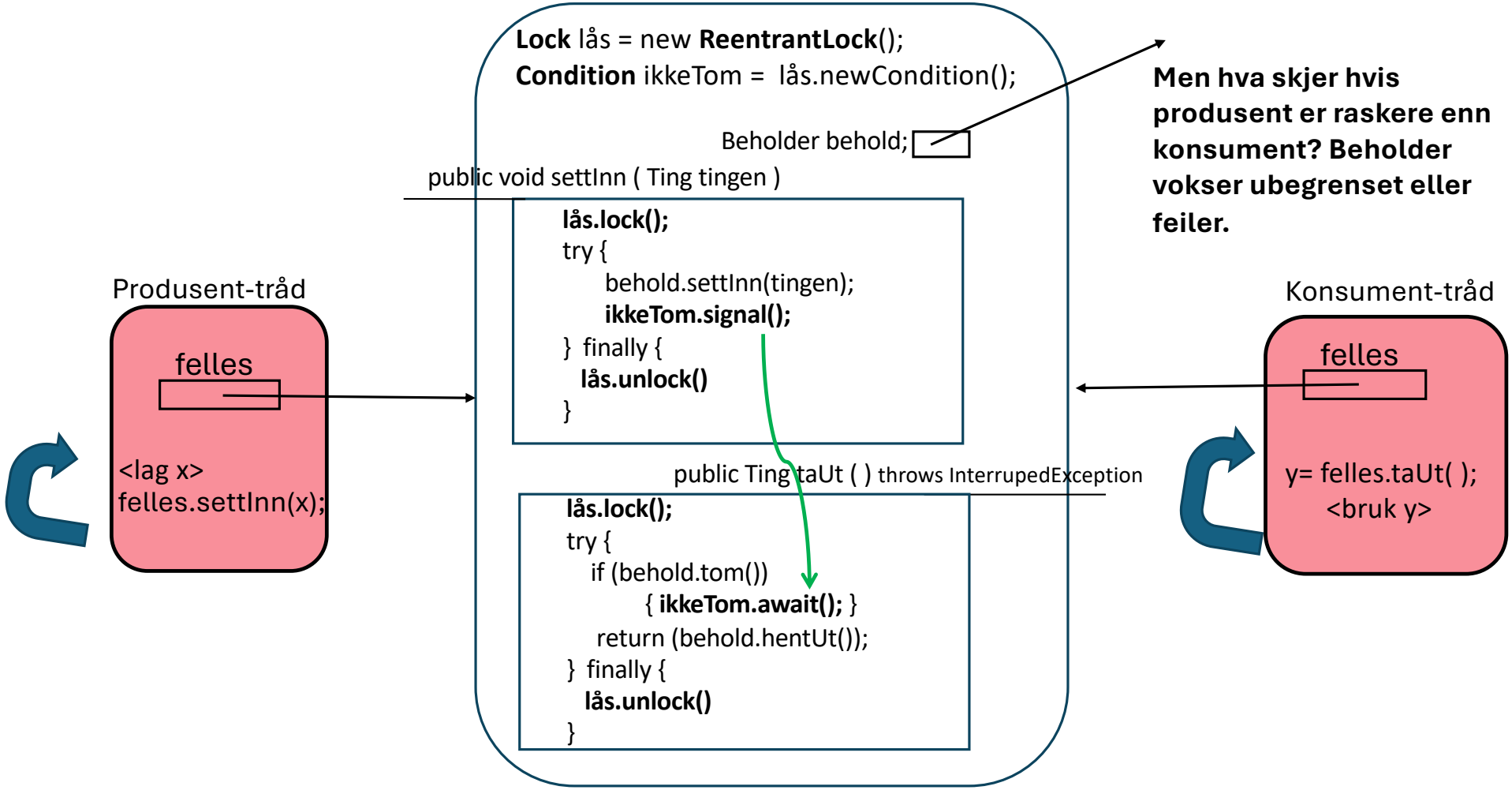
Vente på å komme inn i en kritisk region



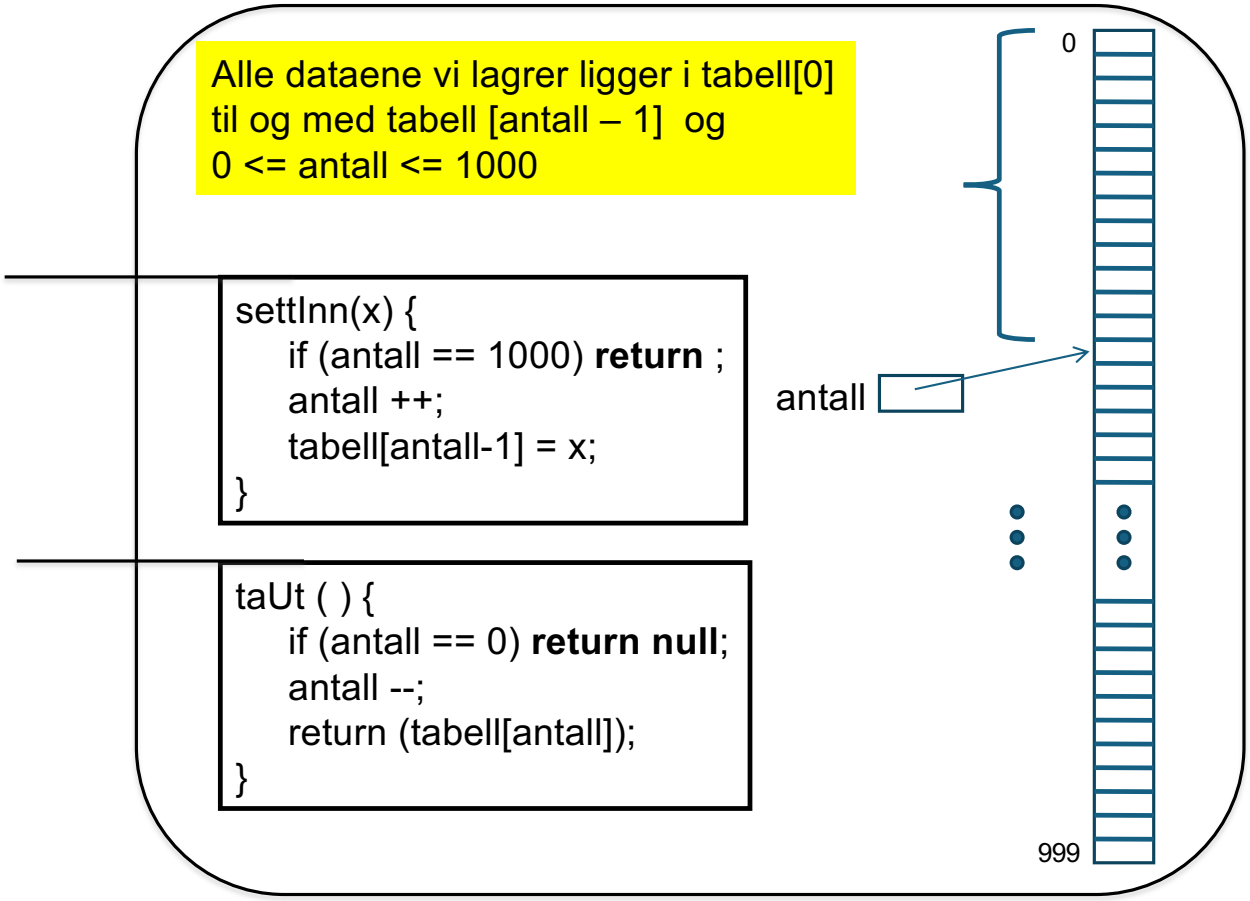
En monitor (et objekt)

En monitor som gjør en beholder trådsikker

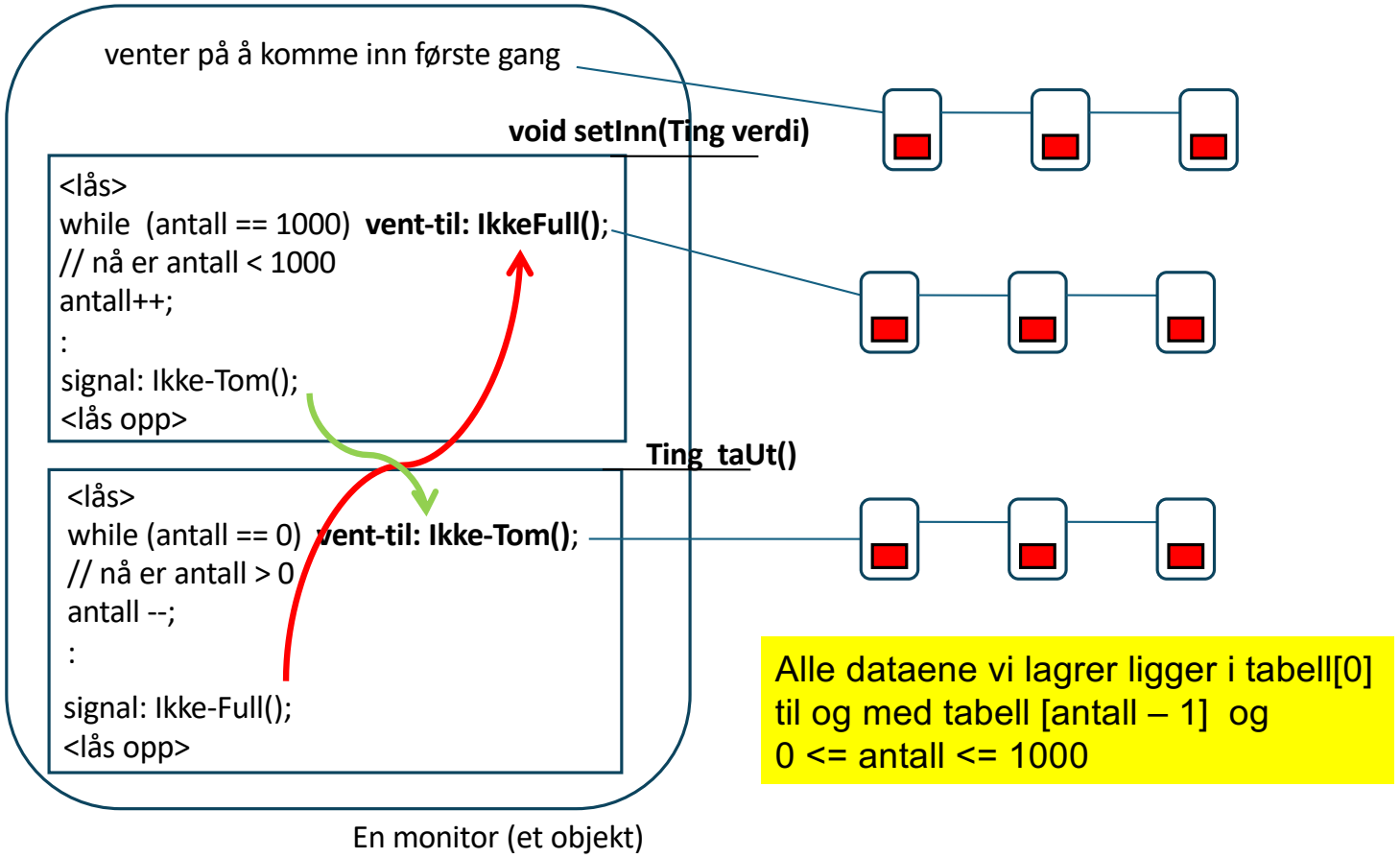
Produsent og konsument



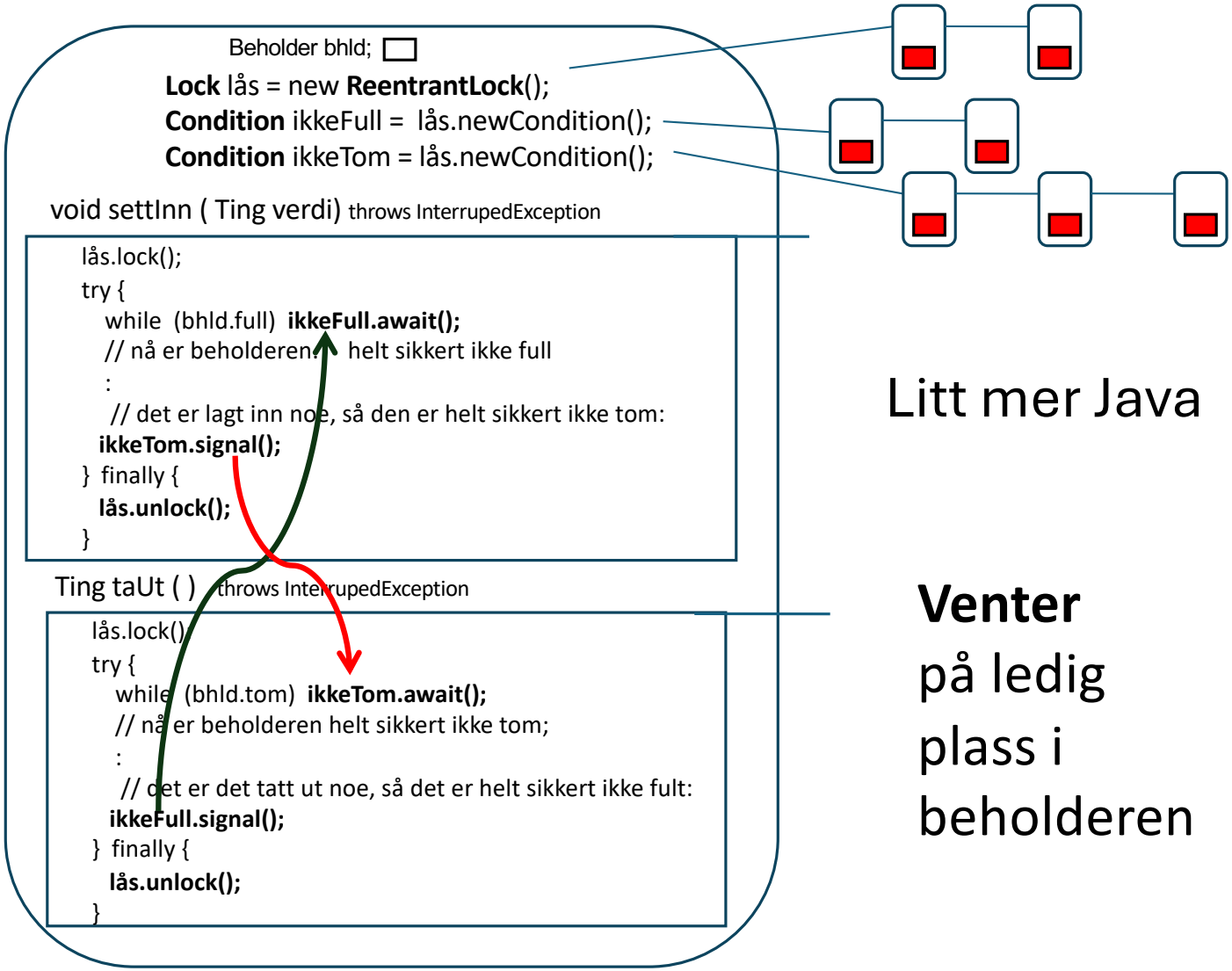
Vi kan begrense hvor mye vi vil lagre



Vi må sende signal til både produsent og konsument



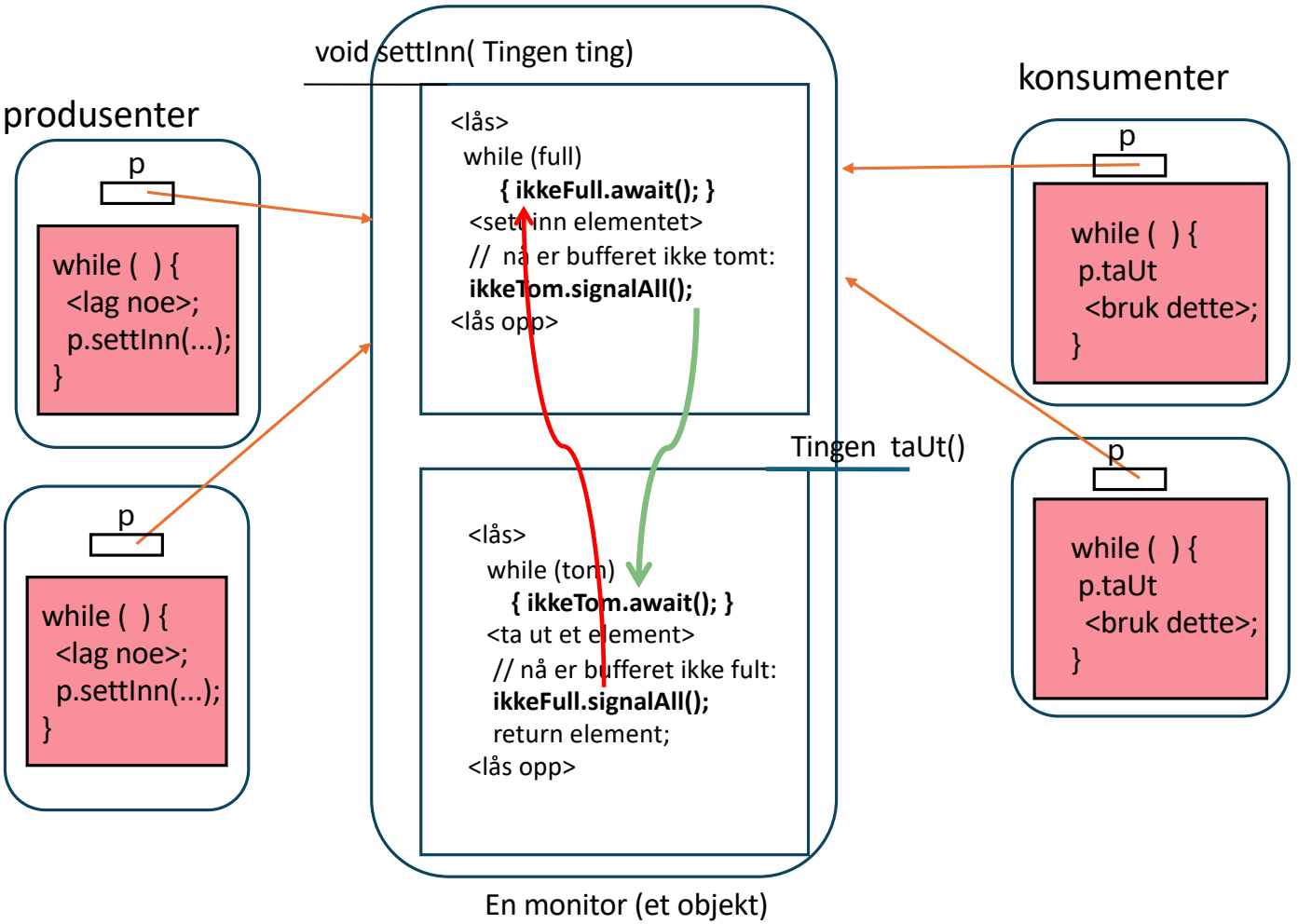
Alle dataene vi lagrer ligger i tabell[0] til og med tabell [antall – 1] og
0 <= antall <= 1000



Litt mer Java

Venter
på ledig
plass i
beholderen

Produsenter og konsumenter



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 i primærlageret.




```
boolean stopp = false;

// En tråd:
stopp = true;

// En annen tråd:
  while(! stopp) {
        . . .
  }
```

Må da deklarere:

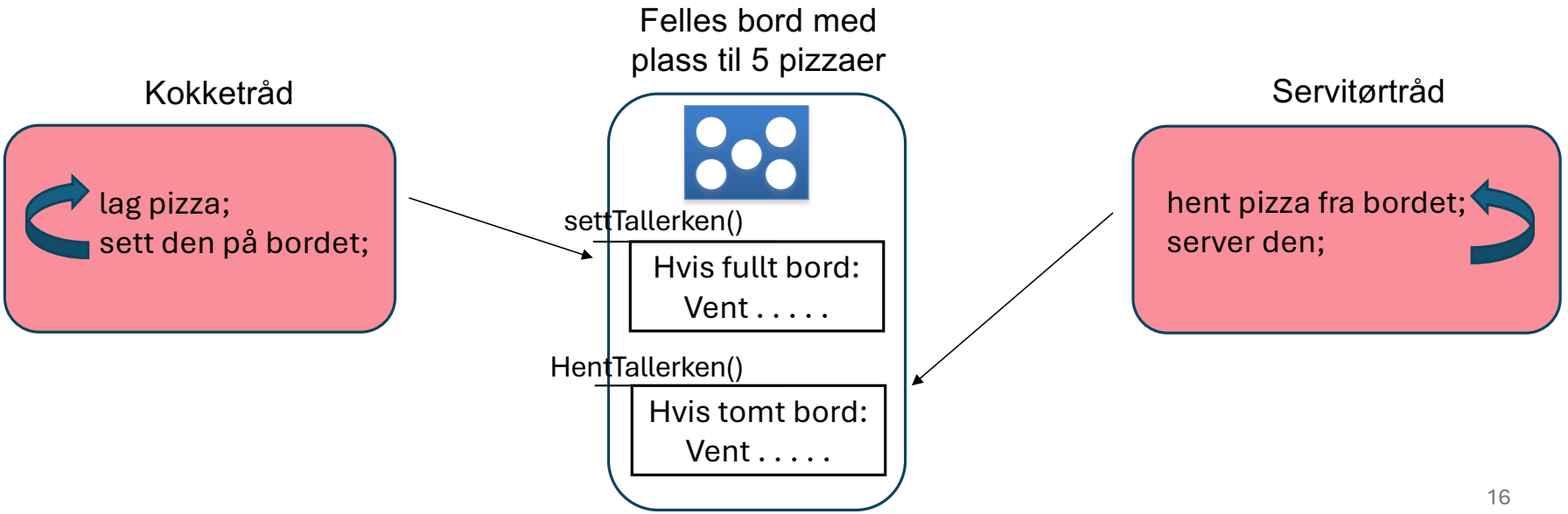
```
volatile boolean stopp=false;
```

Klassisk eksempel: Kokk og servitør



Kokk og servitør (se program på IN1010-siden)

- Kokken må vente når det allerede er fem pizzaer på bordet
- Servitøren må vente når det ikke er laget noe mat (ingen pizzaer på bordet)
- Kokken må starte opp servitøren igjen når han setter en pizza på bordet
- Servitøren må starte opp kokken igjen når hun tar en pizza fra bordet



Nytt eksempel

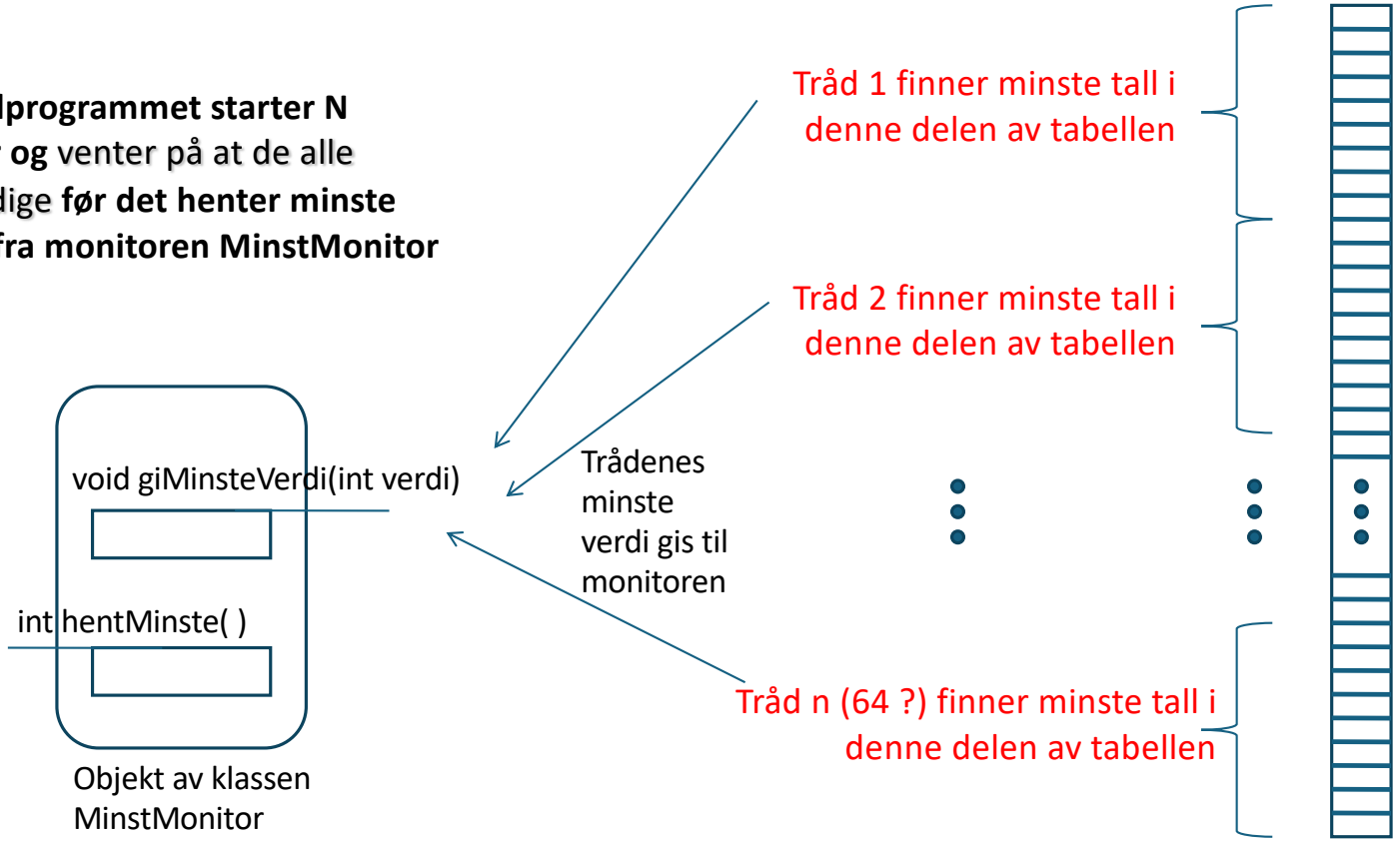
- Finne minst tall i en array



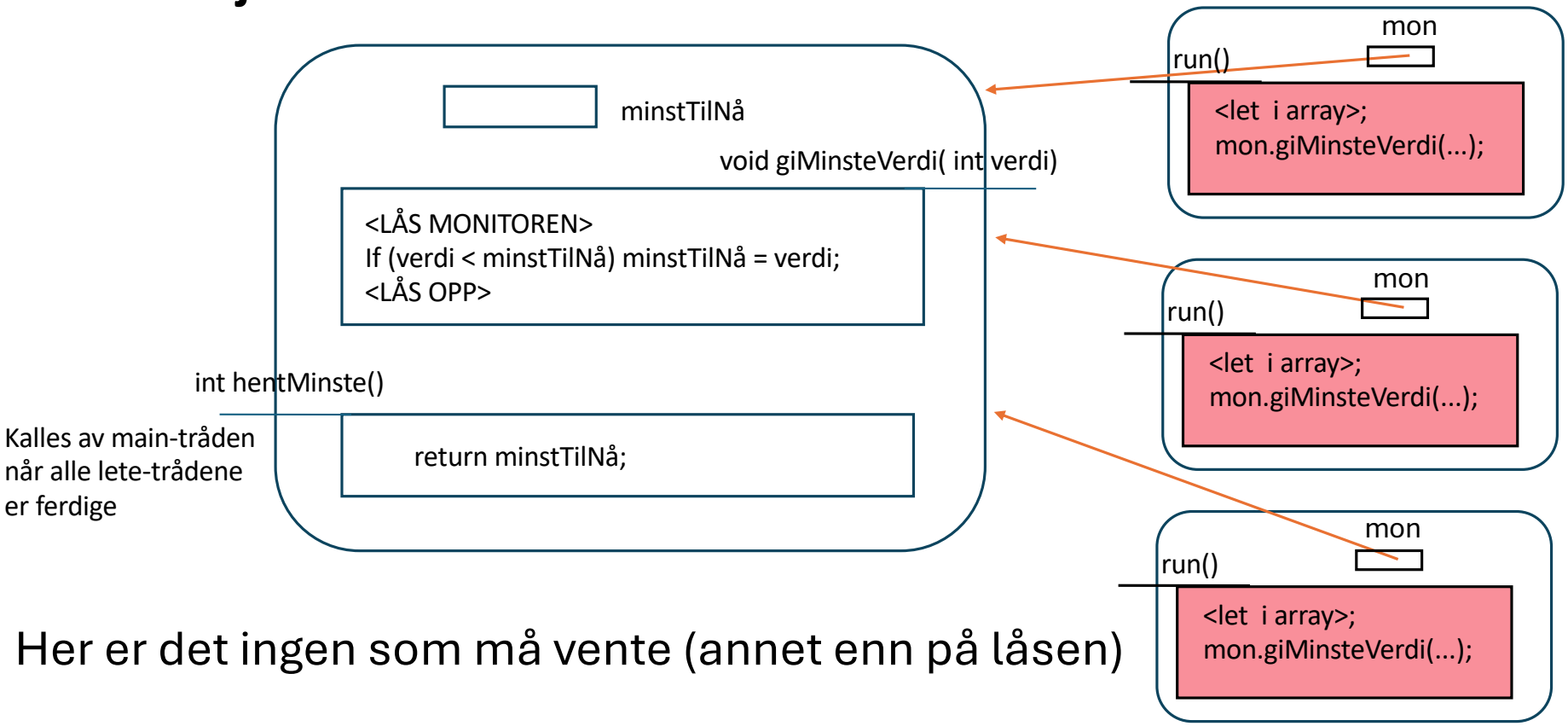
Eksempel på parallellisering: Finn minste tall i en tabell

(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



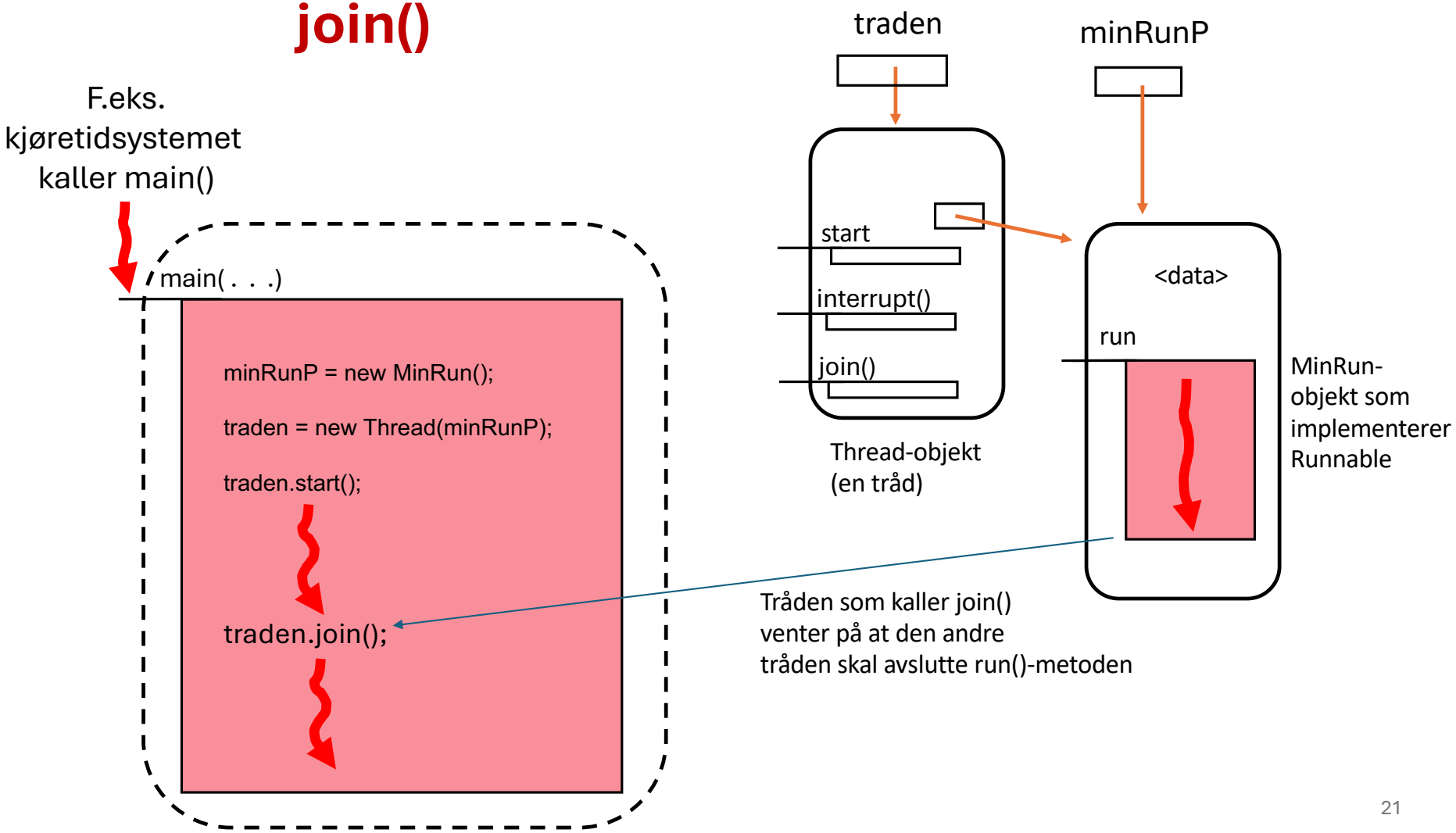
Her er det ingen som må vente (annet enn på låsen)

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

Gafling og møting (fork and join)

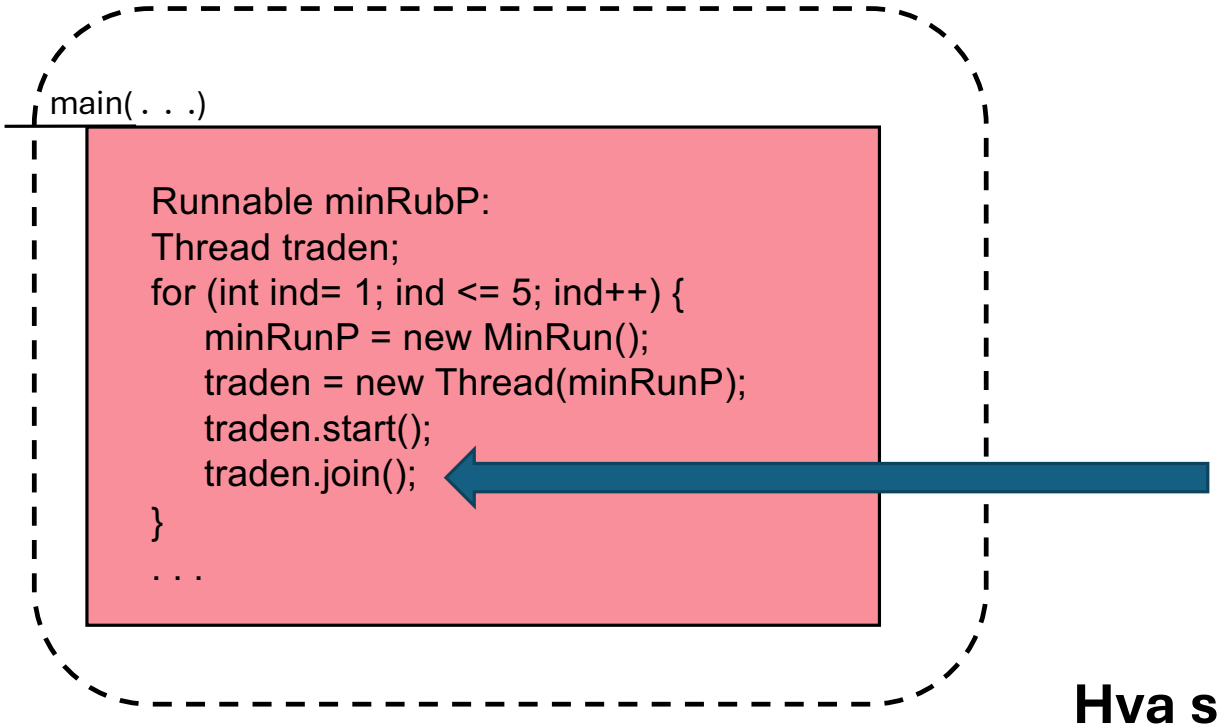
- I parallellprogrammering og på engelsk brukes «**fork**» om å lage en ny tråd eller en ny prosess
- Tror ikke det brukes norsk oversettelse (forgrening og samling?)
- Det finnes ikke noen fork-mekanisme i Java
  - Du må selv lage en ny tråd, og eventuelt gå i løkke for å lage flere
- Når tråder terminerer kalles det gjerne «**join**» (neste side)

join()


F.eks.
kjøretidsystemet
kaller main()
main(. . .)
minRunP = new MinRun();
traden = new Thread(minRunP);
traden.start();
traden.join();
traden
minRunP
start
interrupt()
join()
Thread-objekt
(en tråd)
<data>
run
MinRun-
objekt som
implementerer
Runnable
Tråden som kaller join()
venter på at den andre
tråden skal avslutte run()-metoden

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

Ikke gjør slik:

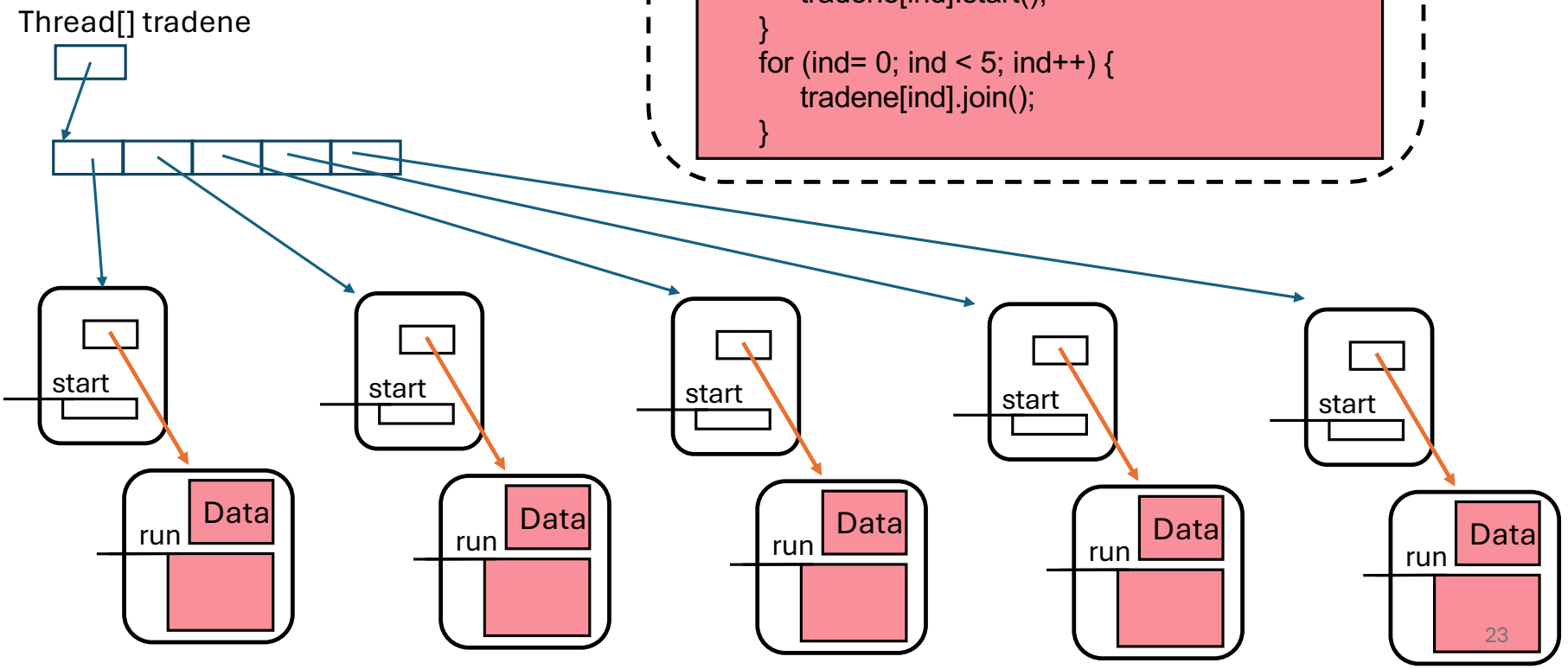


Hva skjer?

Kan vente på flere tråder:

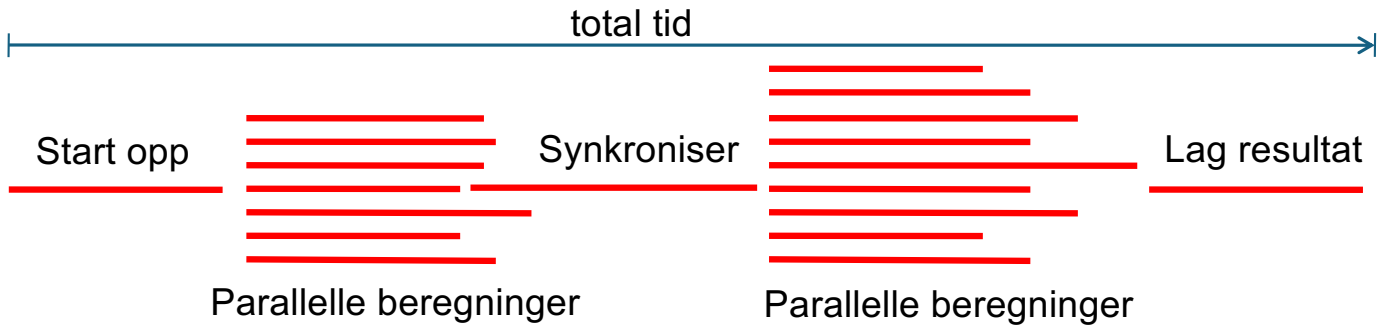
main( )

```
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();
}
```



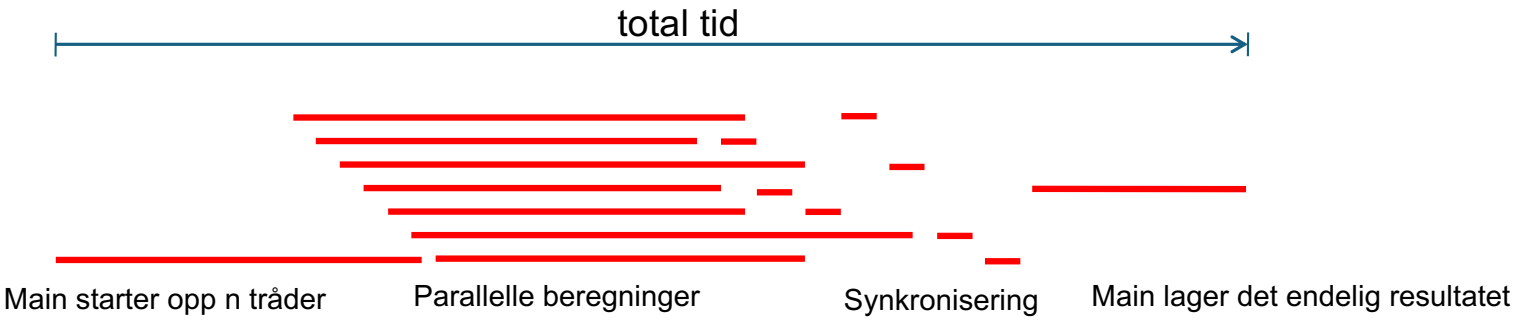
Amdahls lov

- 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”

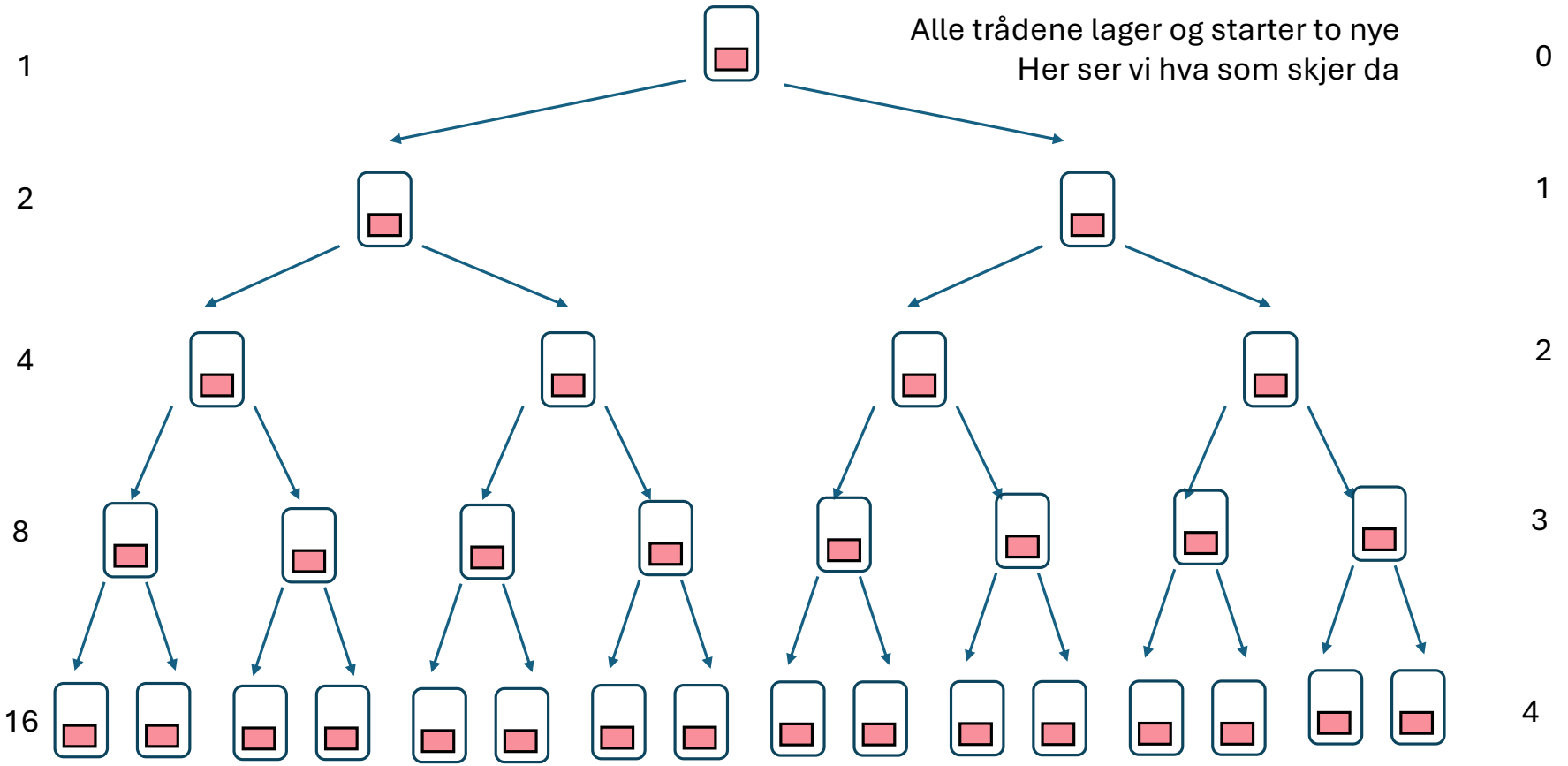
- Totaltid:
  - Tiden det tar å lage og sette i gang 64 tråder
    - (kan det gjøres i parallell ?) ($\log_2 64 = 6$*)
  - Tiden det tar å finne et minste tall i min del av tabellen
  - Til slutt i monitoren: En og en tråd må teste sitt resultat
    - (Kan det gjøres i parallell ?)



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

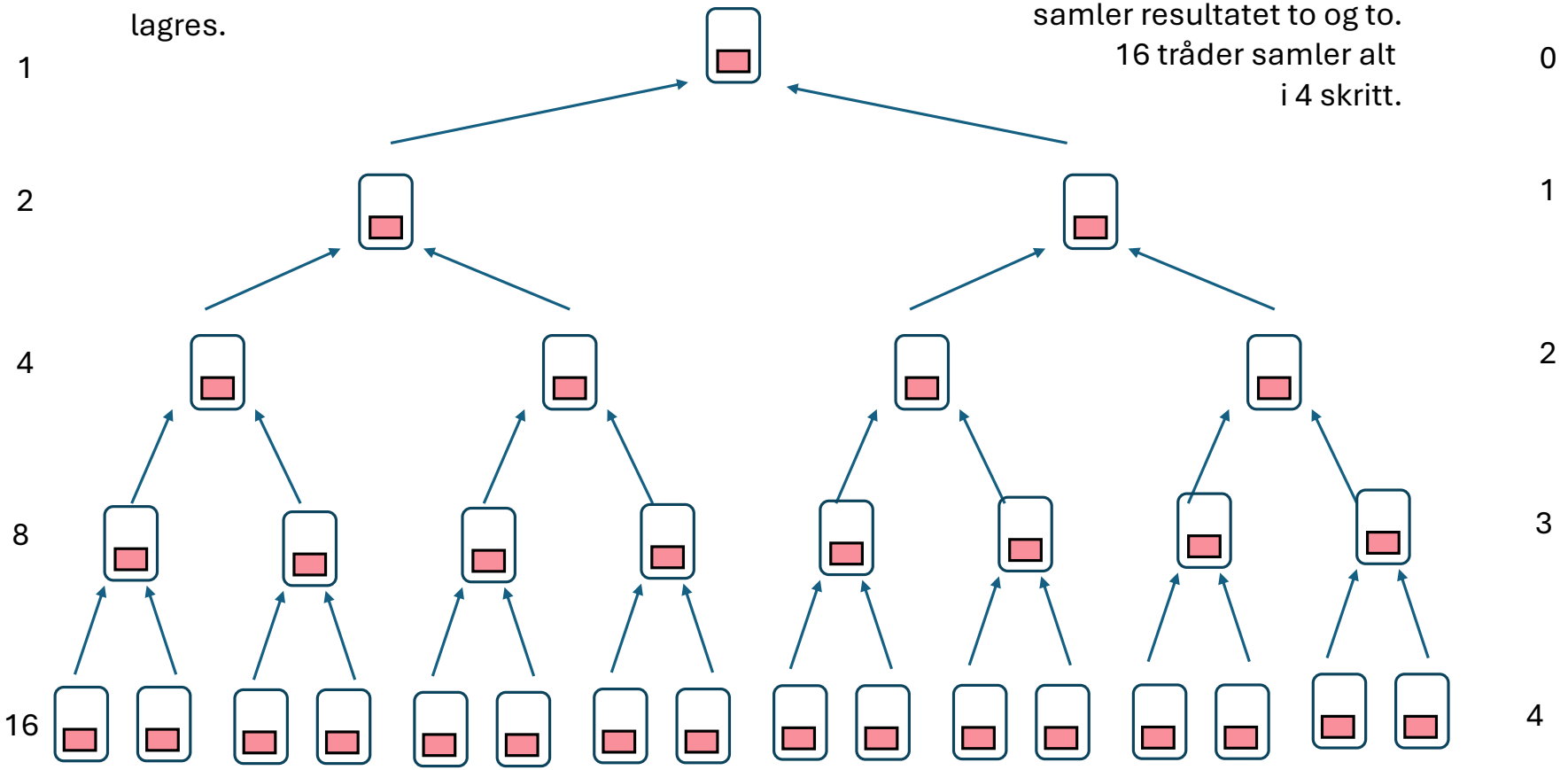
2, 4, 8, 16, 32, 64

Trær er populære i informatikk (men dette eksempelet er litt på siden i IN1010):

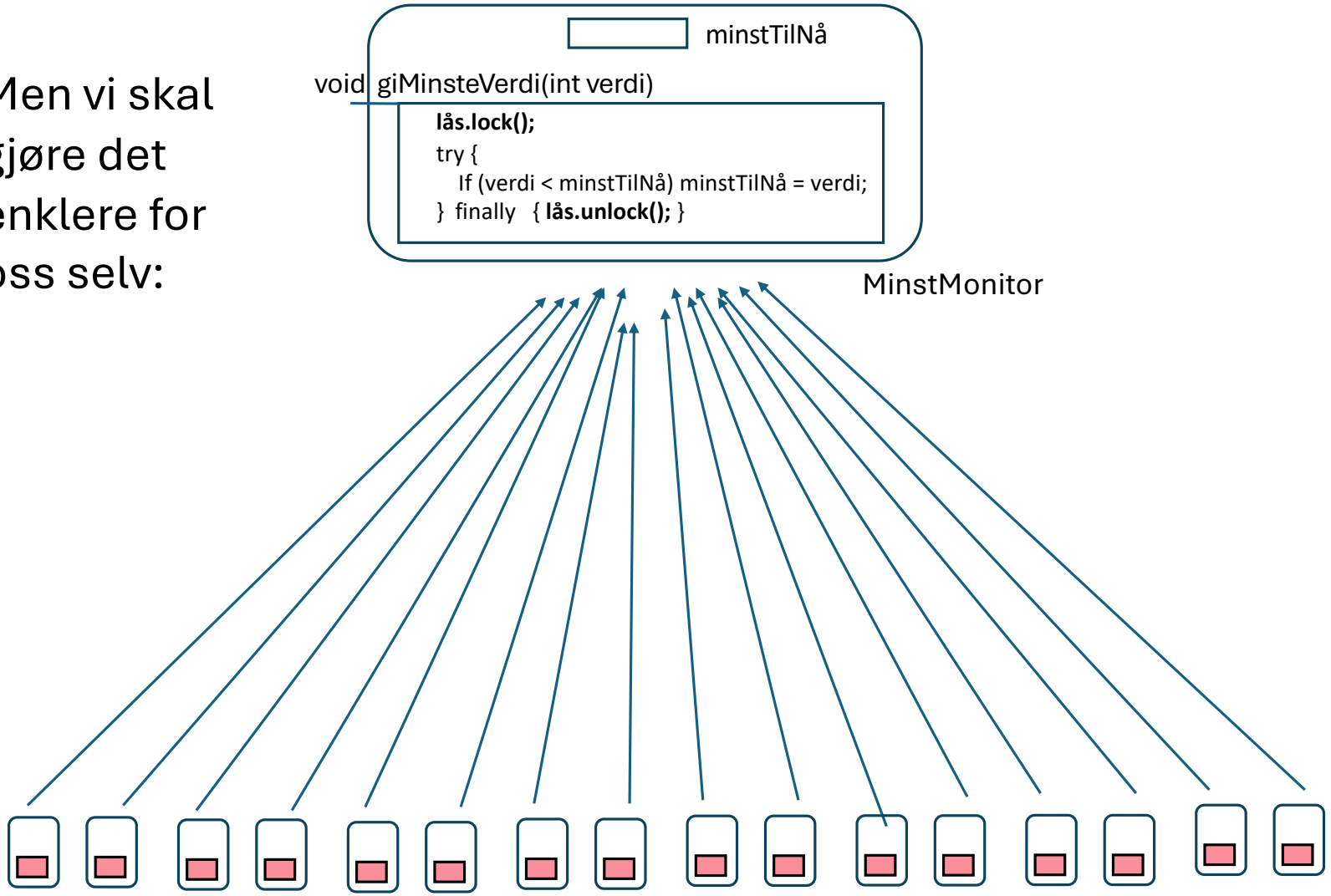


Men her mangler det mye bl.a. om hvor resultatene skal lagres.

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



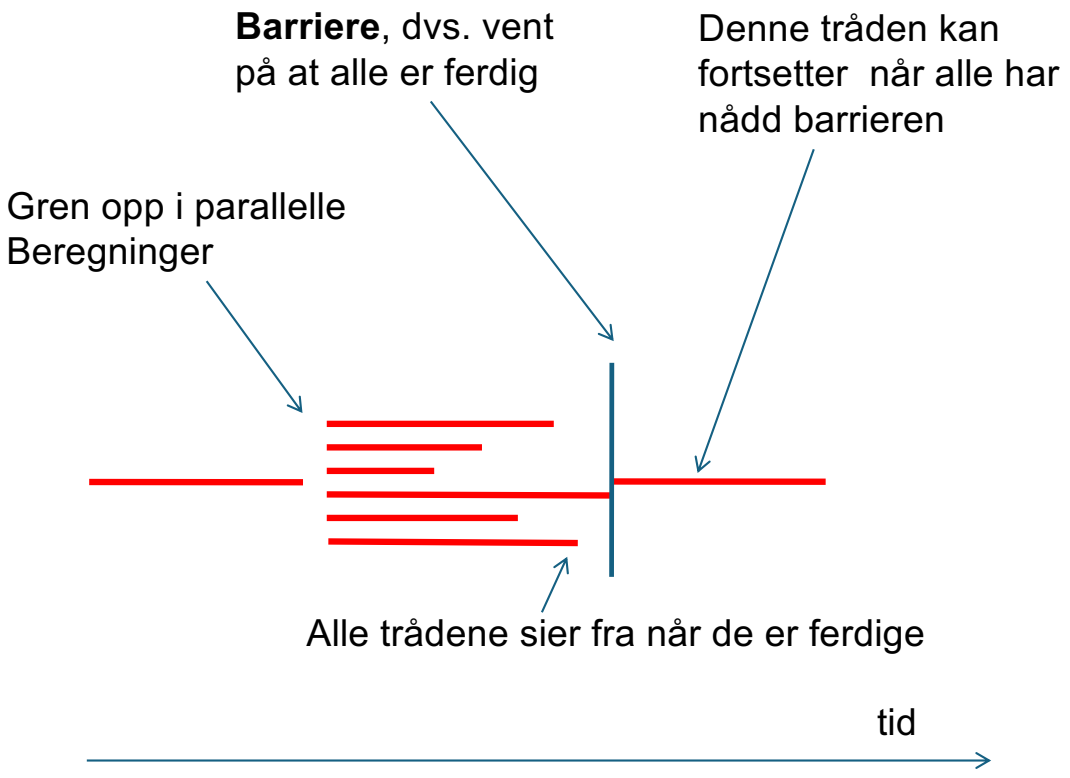
Men vi skal gjøre det enklere for oss selv:



Minst.java

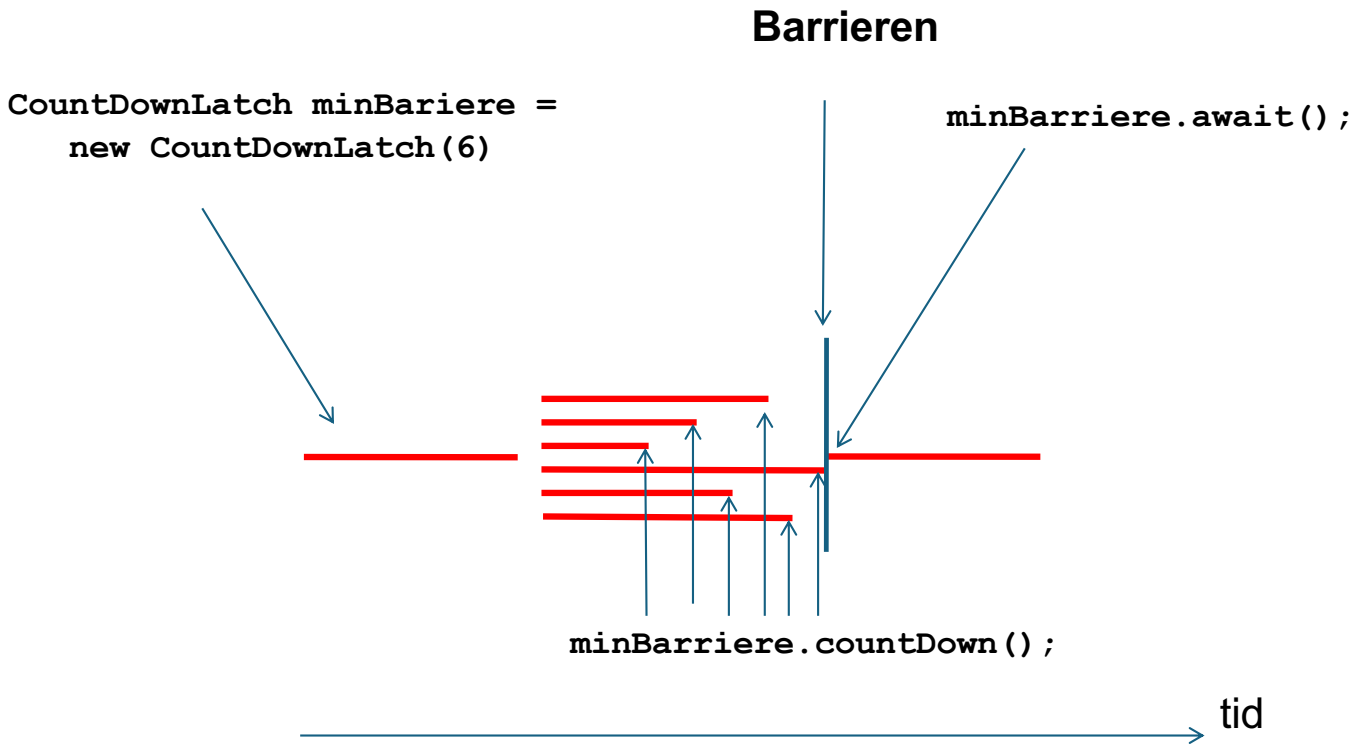
- Lag et program som finner den minste verdien i et array ved hjelp av 64 tråder
- Bruk join() for å vente på at trådene skal avslutte.

Barrierer i parallellprogrammering

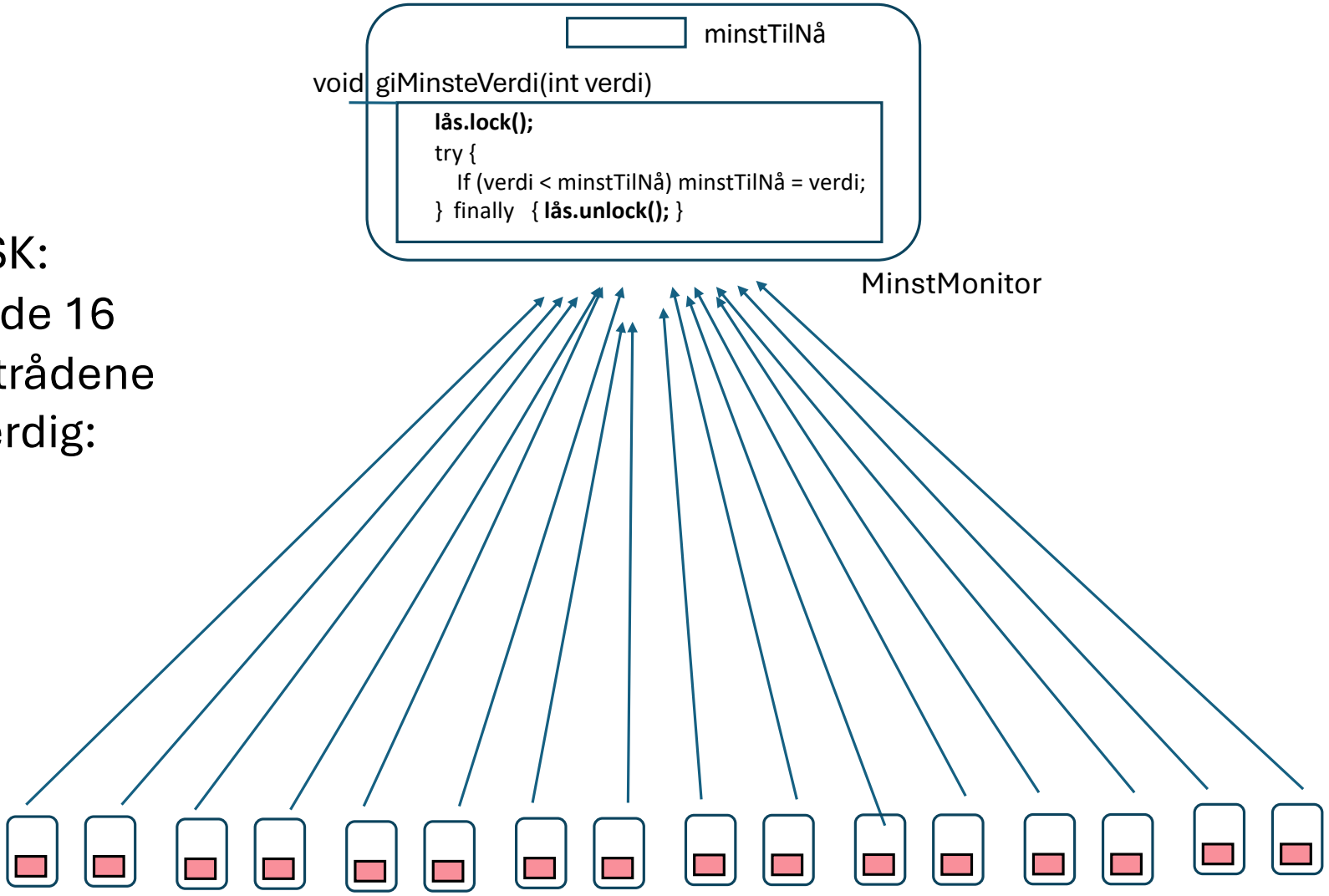


Barrierer i Java

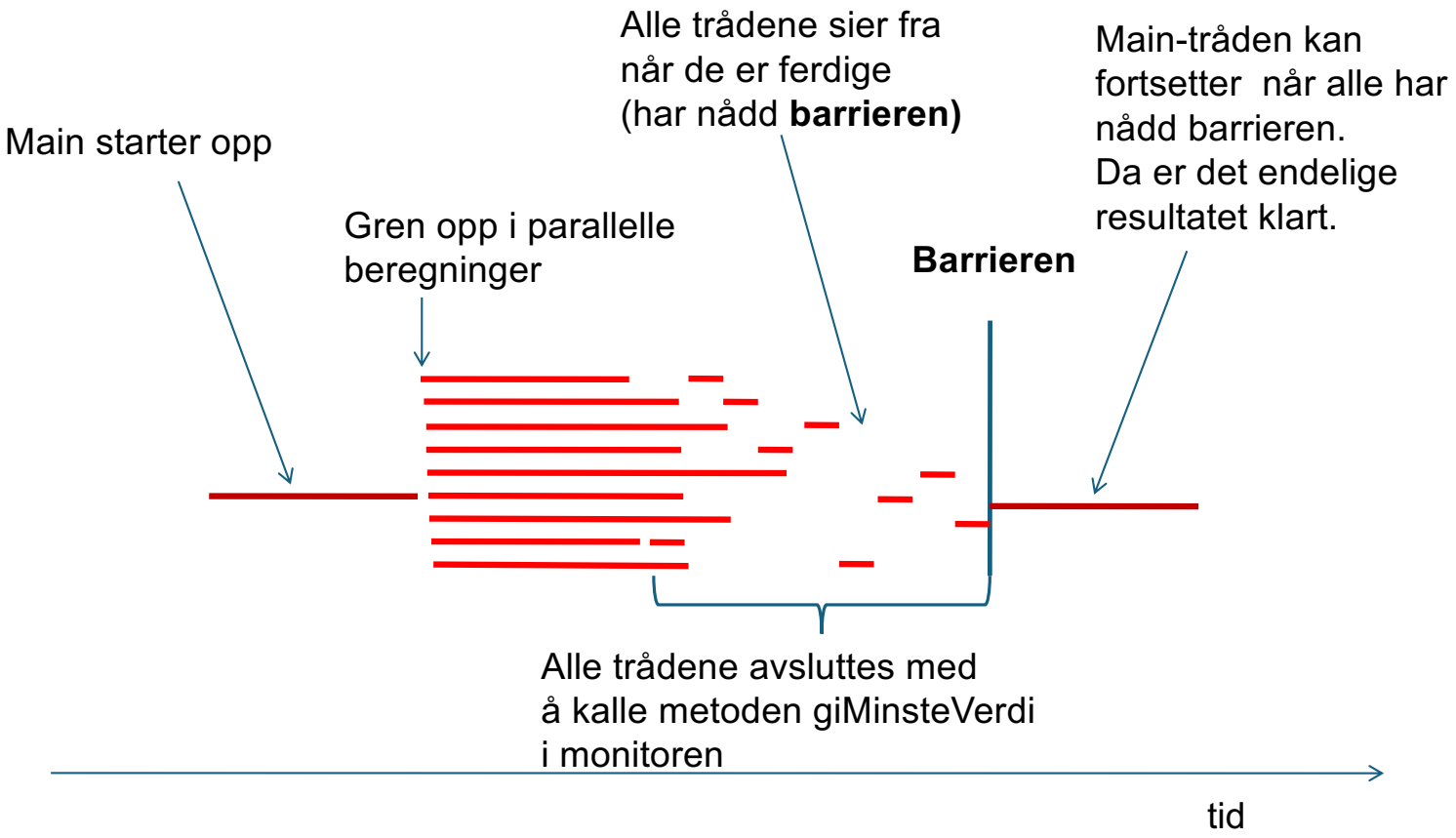
```
import java.util.concurrent.*;
```



HUSK:
Når de 16 letetrådene er ferdig:



Barrieren i dette eksemplet (Minst.java)



Minst2.java

- Vi utvider minst.java til å bruke en barriere

Nytt tema
(men fortsatt tråder)

Vranglås
Engelsk: Deadlock

Horstmann kap. 20.5

37

Vranglås (deadlock)

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

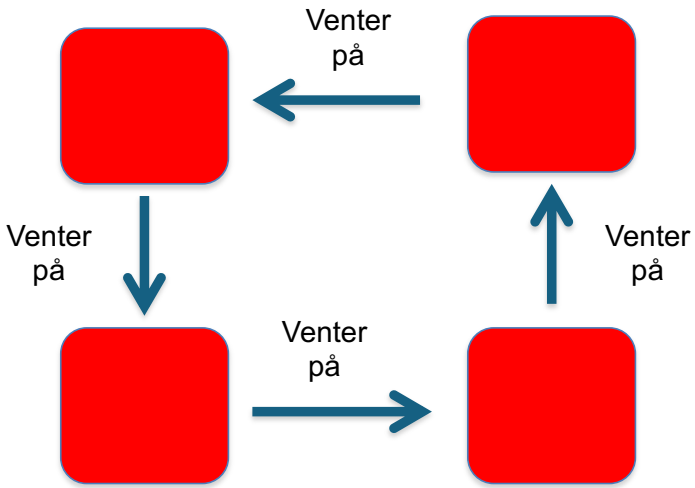
    ->

    Alle stopper = VRANGLÅS

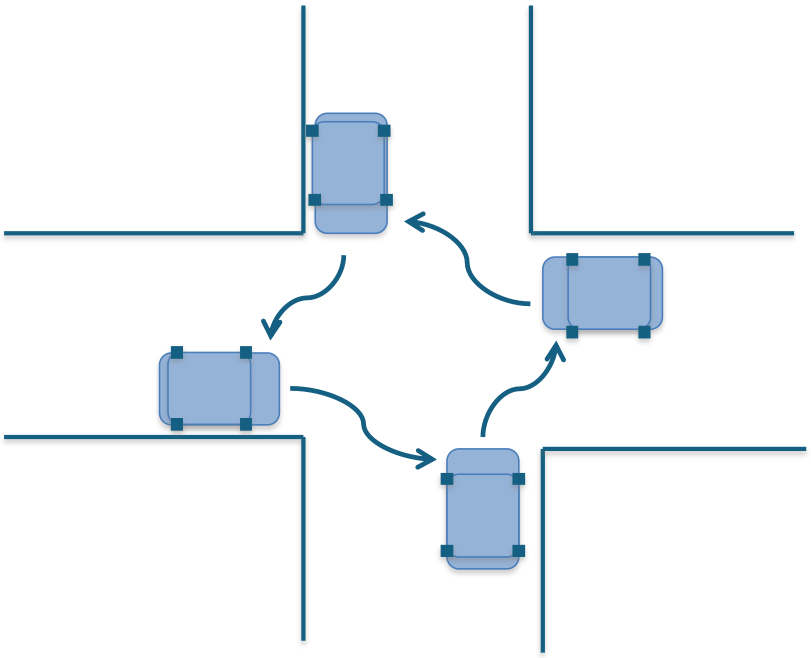


Vranglås betyr

- Flere tråder venter på hverandre
- Syklisk venting

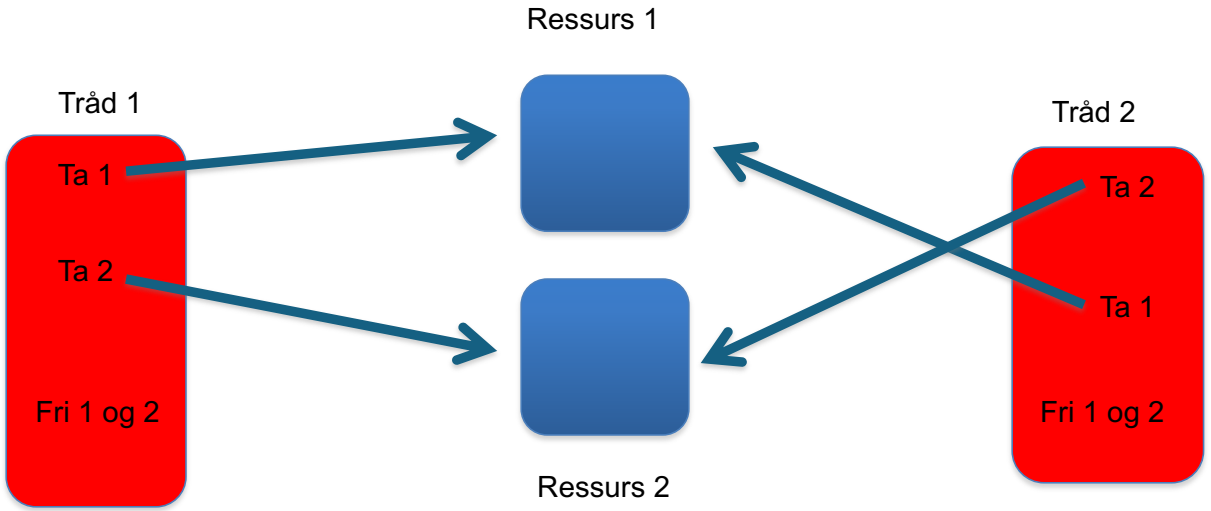


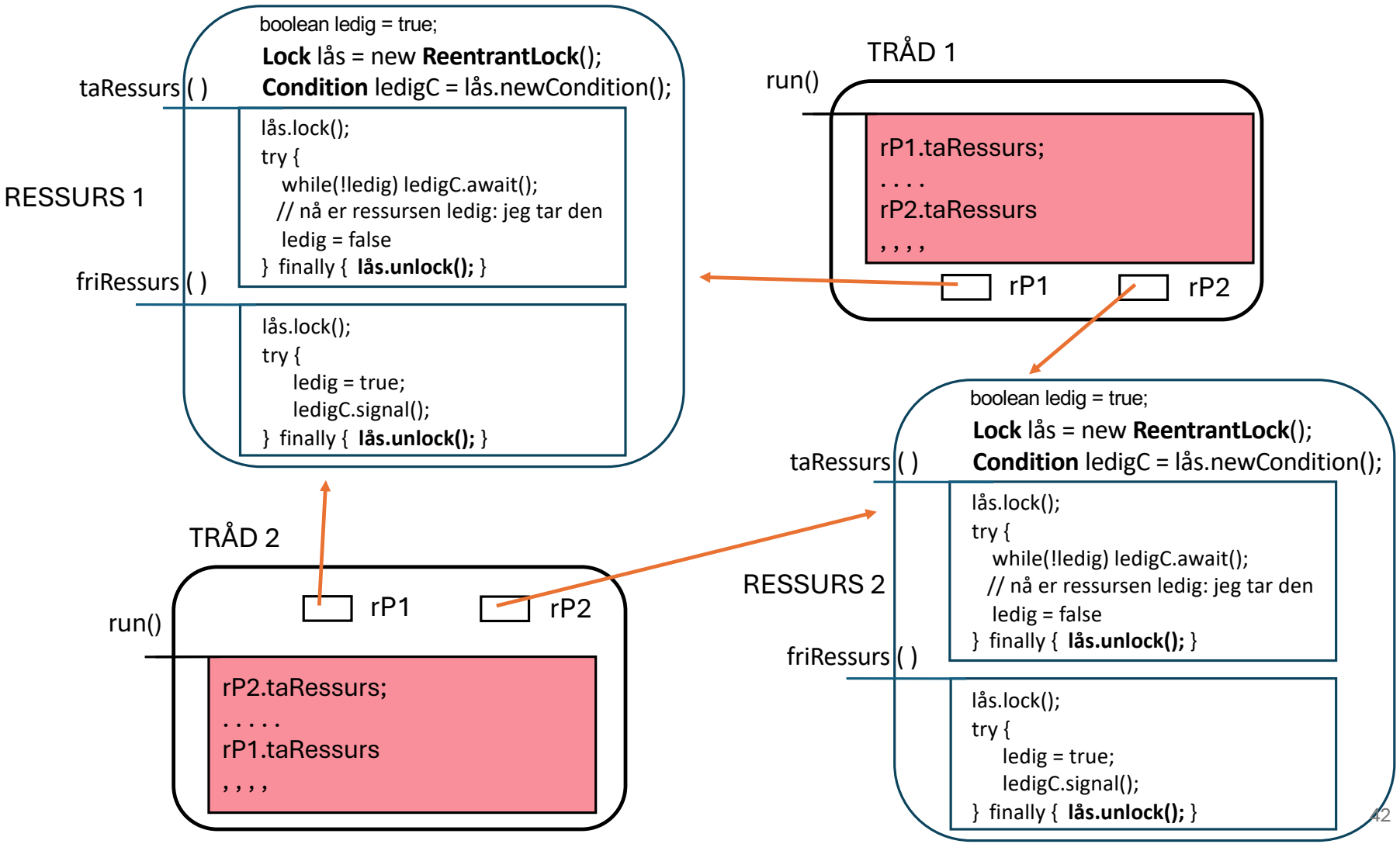
Hvordan kan dette skje?



Enkleste eksempel på vranglås

- To tråder
- To ressurser som tas i forskjellig rekkefølge

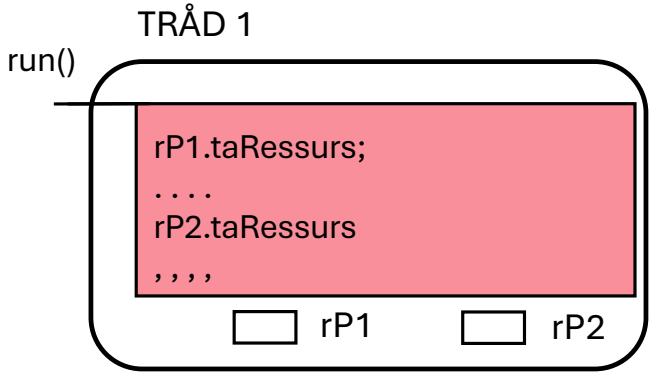


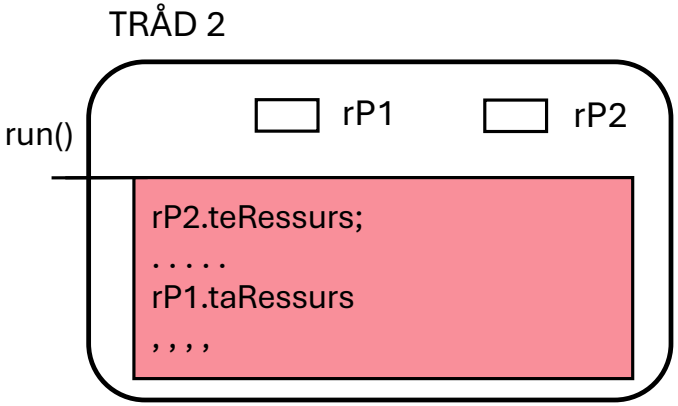

boolean ledig = true;
Lock lås = new ReentrantLock();
taRessurs ( )
Condition ledigC = lås.newCondition();
lås.lock();
try {
while(!ledig) ledigC.await();
// nå er ressursen ledig: jeg tar den
ledig = false
} finally { lås.unlock(); }
RESSURS 1
friRessurs ( )
lås.lock();
try {
ledig = true;
ledigC.signal();
} finally { lås.unlock(); }
TRÅD 1
run()
rP1.taRessurs;
. . . .
rP2.taRessurs
, , , ,
rP1
rP2
boolean ledig = true;
Lock lås = new ReentrantLock();
taRessurs ( )
Condition ledigC = lås.newCondition();
lås.lock();
try {
while(!ledig) ledigC.await();
// nå er ressursen ledig: jeg tar den
ledig = false
} finally { lås.unlock(); }
RESSURS 2
friRessurs ( )
lås.lock();
try {
ledig = true;
ledigC.signal();
} finally { lås.unlock(); }
TRÅD 2
run()
rP1
rP2
rP2.taRessurs;
. . . . .
rP1.taRessurs
, , , ,

VRANGLÅS:

Tråd 1 tar rP1
Tråd 2 tar rP2

Tråd 1 ønsker å ta rP2 men må vente (i ledig.await())
Tråd 2 ønsker å ta rP1 men må vente (i ledig.await())





Vranglås kan oppstå når flere tråder kjemper om felles ressurser

- En eller flere felles ressurser ønskes av mer enn en tråd (f.eks. i et OS)
- Hvis en tråd først tar en ressurs og deretter en annen . . .

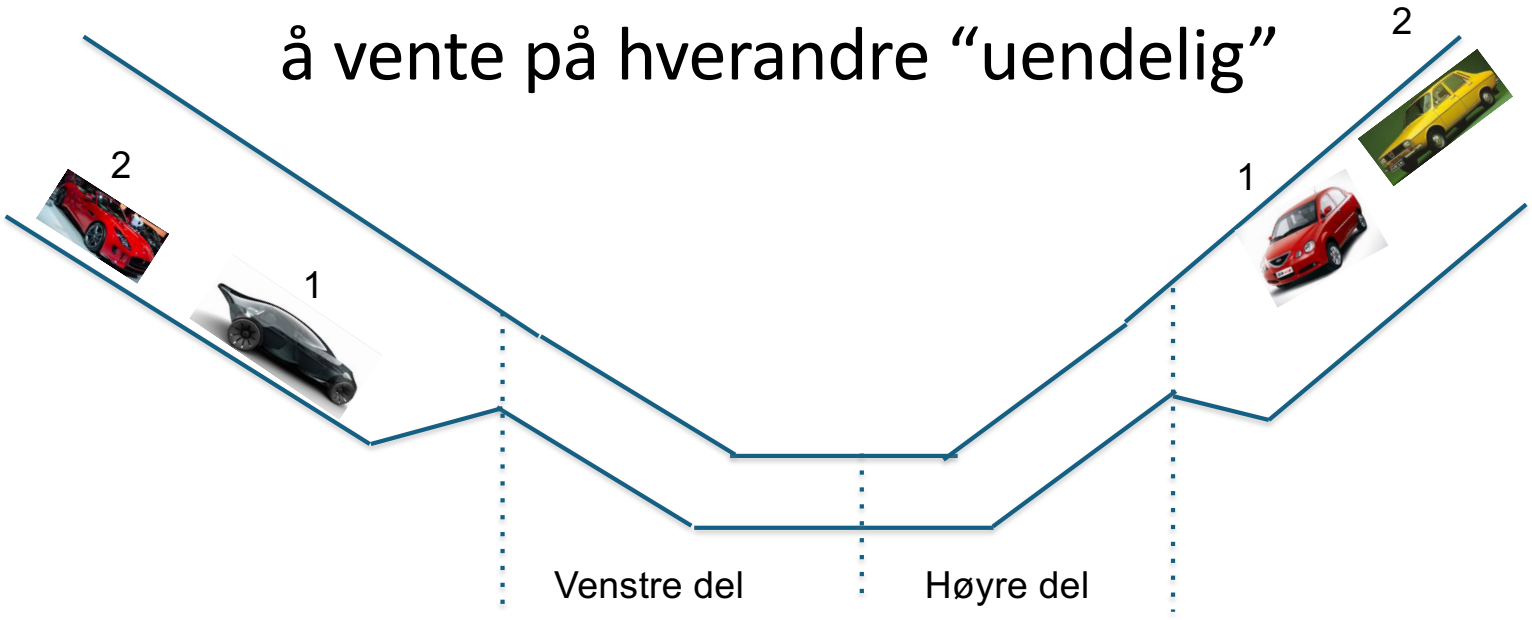
Unngå vranglås

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

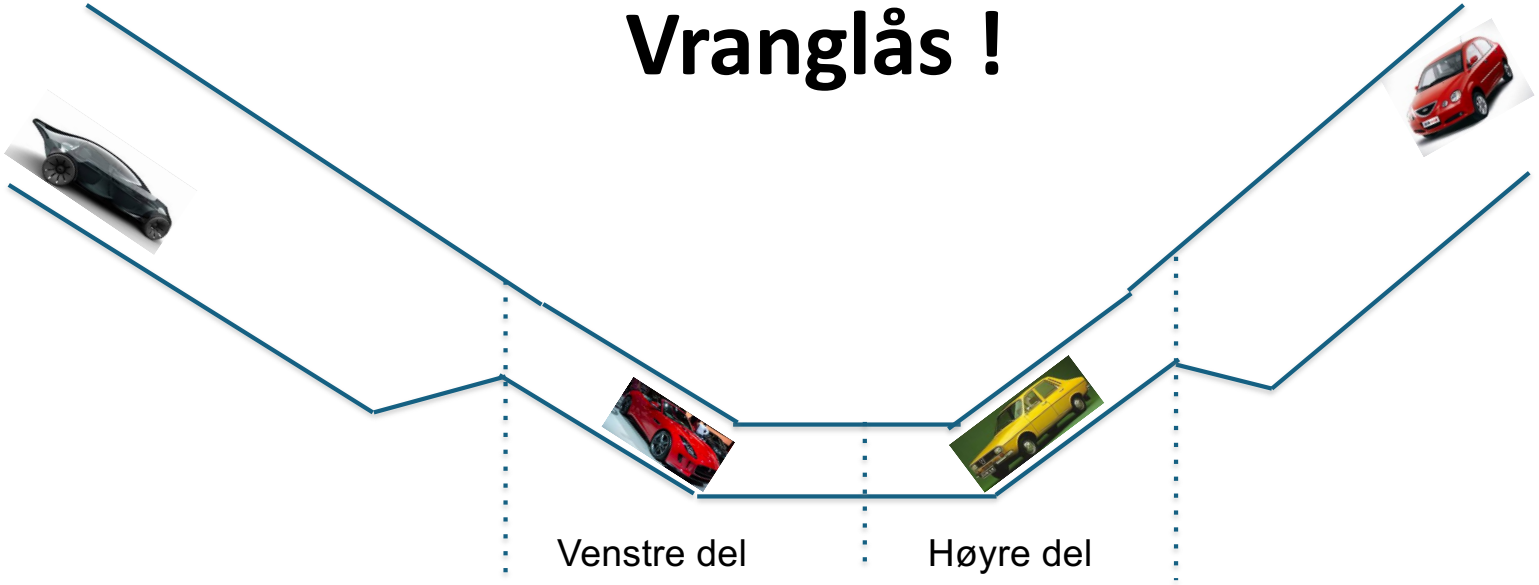
Samme eksempel på vranglås:

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.

Vranglås !

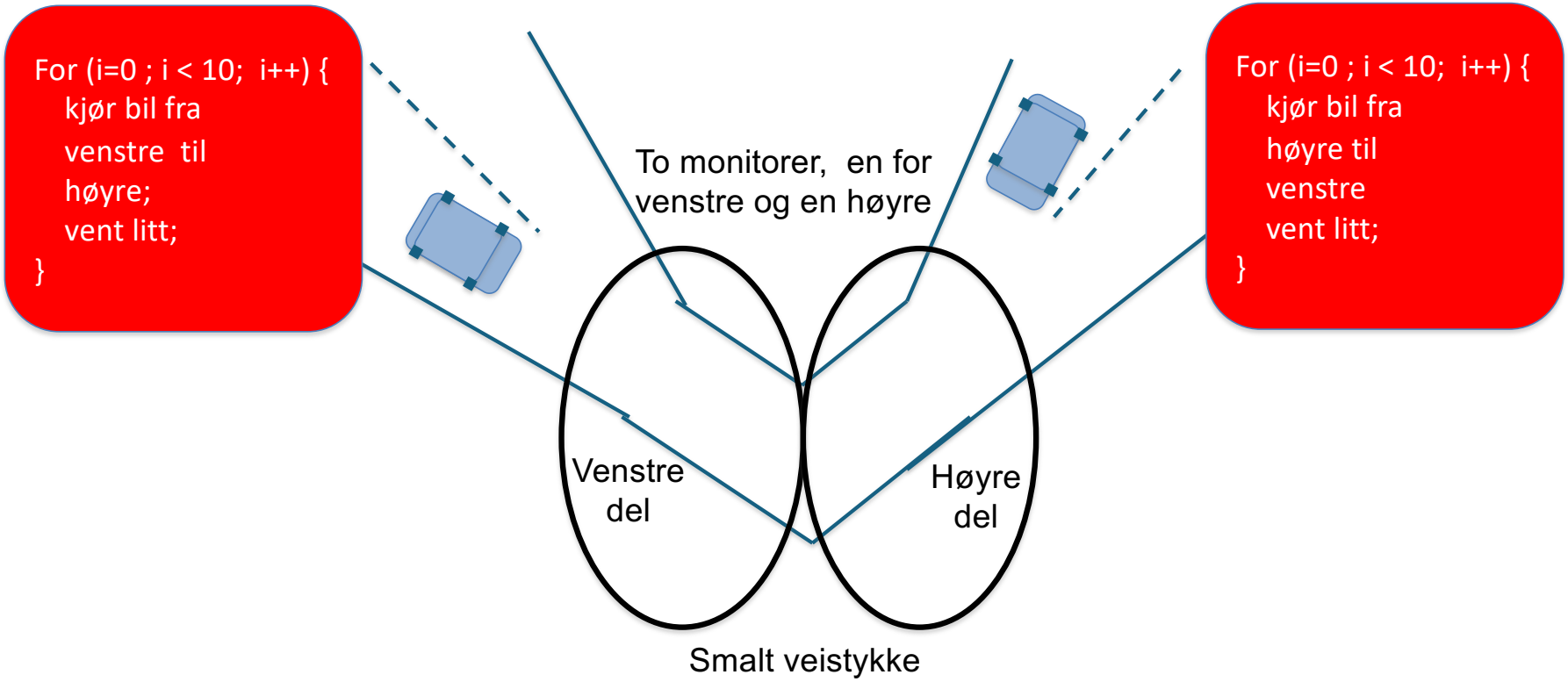


Det smale veistykket består av to ressurser, og begge bilene venter på at den andre bilen skal bli ferdig med å bruke sin ressurs

De har tatt ressursene i forskjellig rekkefølge!

Eksempel Smaltveistykkev1.java

Se IN1010-siden



Eksempel
Smaltveistykkev1.java

```
Bil nr 14 venter på å passere  venstre  veistykke
Bil nr 14 venter på å passere  høyre  veistykke
Bil nr 14 har passert hele den smale veien

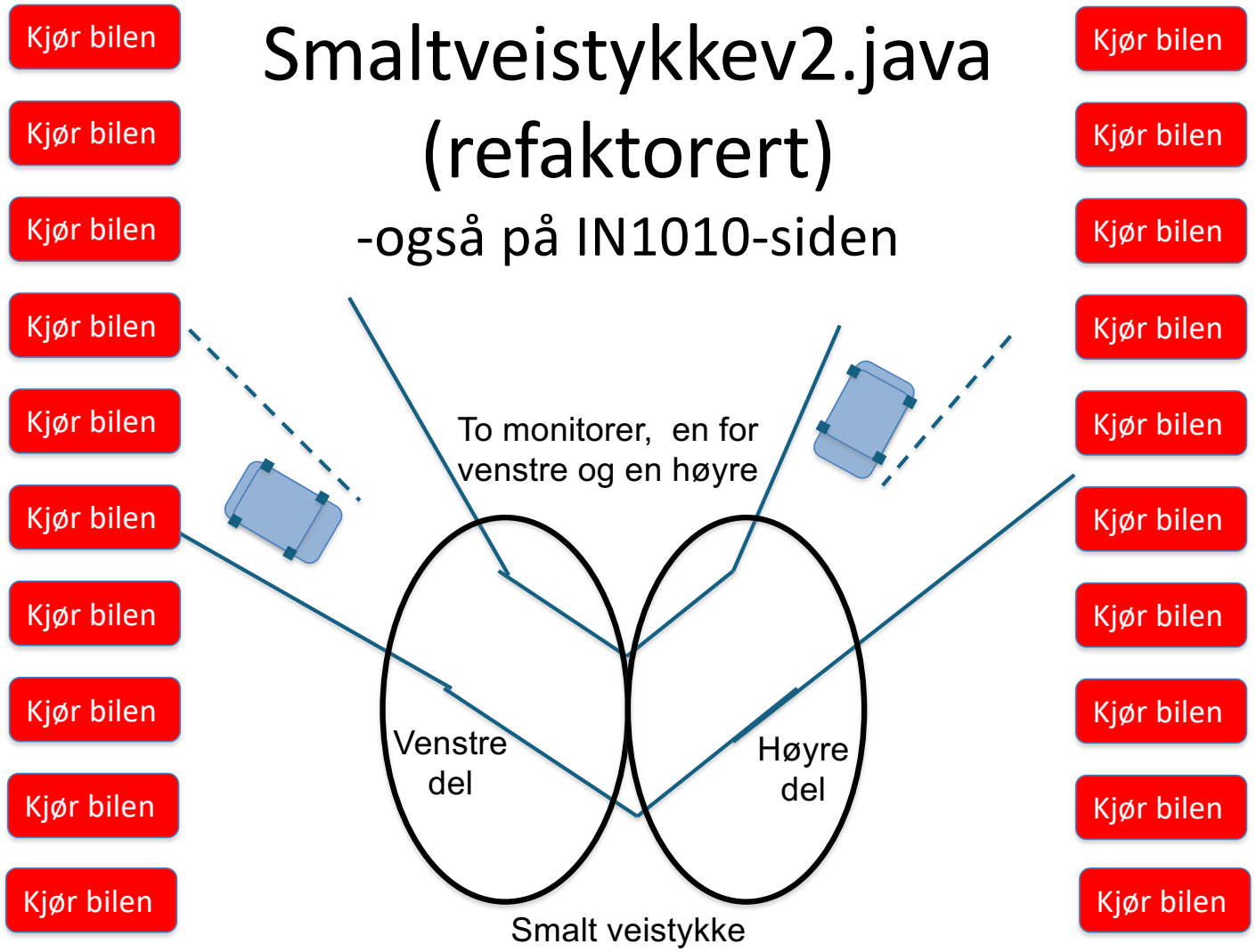
Bil nr 21 venter på å passere  høyre  veistykke
Bil nr 21 venter på å passere  venstre  veistykke
Bil nr 21 har passert hele den smale veien

Bil nr 23 venter på å passere  høyre  veistykke
Bil nr 16 venter på å passere  venstre  veistykke
Bil nr 23 venter på å passere  venstre  veistykke
Bil nr 16 venter på å passere  høyre  veistykke
```

Løkkene kommer ikke videre pga. vranglås

Smaltveistykkev2.java (refaktorert)

-også på IN1010-siden



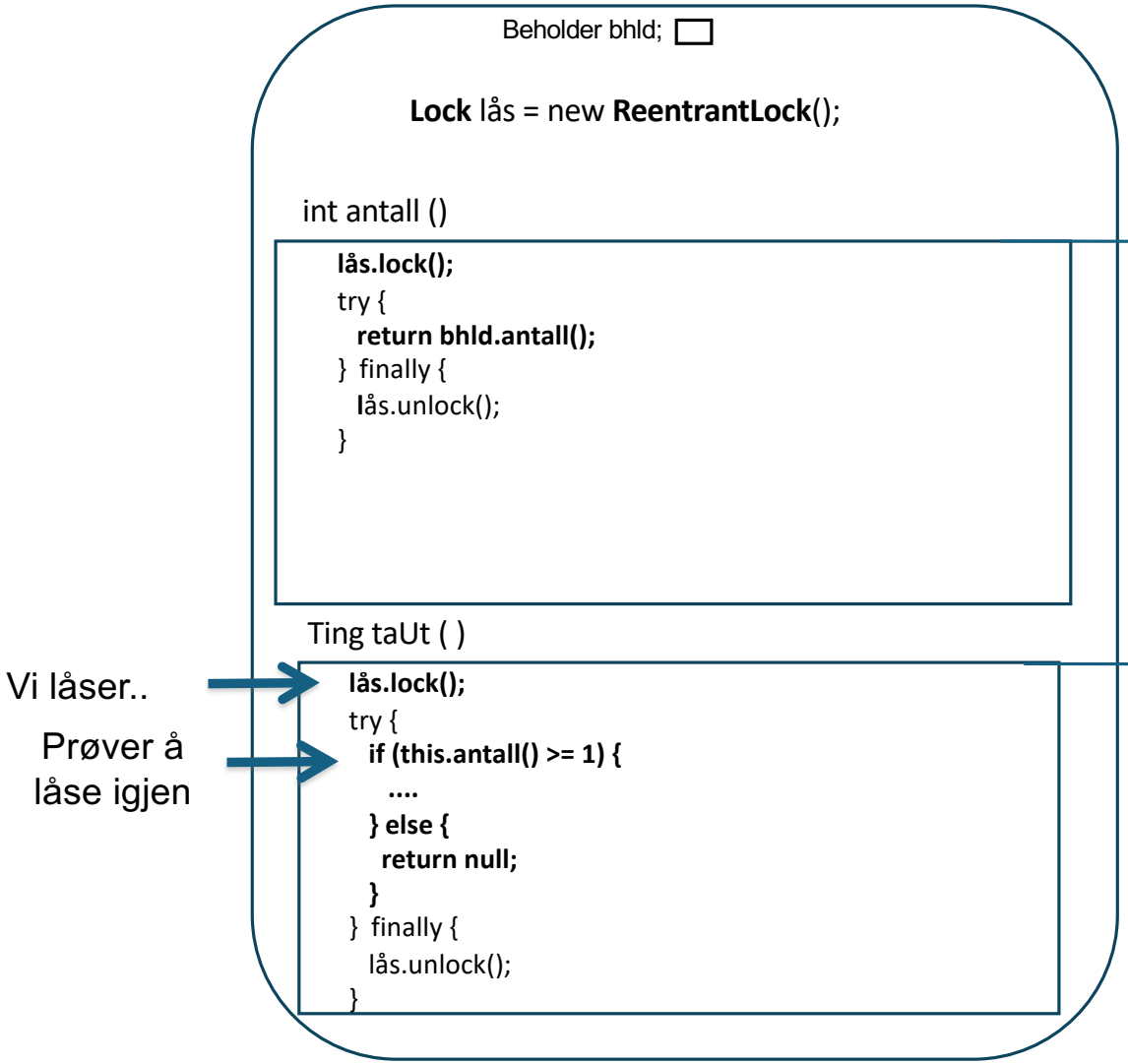
Eksempel
Smaltveistykkev2.java

```
Bil nr 11 venter på å passere  venstre  veistykke
Bil nr 11 venter på å passere  høyre  veistykke
Bil nr 11 har passert hele den smale veien

Bil nr 12 venter på å passere  høyre  veistykke
Bil nr 12 venter på å passere  venstre  veistykke
Bil nr 12 har passert hele den smale veien

Bil nr 13 venter på å passere  venstre  veistykke
Bil nr 14 venter på å passere  høyre  veistykke
Bil nr 13 venter på å passere  høyre  veistykke
Bil nr 14 venter på å passere  venstre  veistykke
Bil nr 15 venter på å passere  venstre  veistykke
Bil nr 16 venter på å passere  høyre  veistykke
Bil nr 17 venter på å passere  venstre  veistykke
Bil nr 18 venter på å passere  høyre  veistykke
```

Trådene (og main) blir aldri avsluttet pga vranglås



Fører dette til vranglås?

Nei – ReentrantLock skjønner at samme tråd har låsen og tillater dette

Disse to ukene har vi lært:

- Hvordan tråder kommuniserer seg imellom ved hjelp av monitorer
- Hvordan programmer (tråder) venter i en monitor
  - og starter opp igjen de som venter
  - Låsing og venting og signalering (på betingelser) skjer bare inne i monitorer.
- Hvordan tråder oppstår og kan vente på hverandre
- At det er utfordrende å programmere parallelle aktiviteter
  - Viktig å resonnere om programmets tilstand.
    - Ikke la beholdere vokse ubegrenset
  - Nesten 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 forekommer særlig når ressurser tas i forskjellig rekkefølge

Innlevering 5

- Oppgave 1: Frekvenstabell
- Oppgave 2: Subsekvensregister
- Oppgave 3: Monitor
- Oppgave 4: Lesetråd
- Oppgave 5: Utvidet Monitor
- Oppgave 6: Flettetråd
- Oppgave 7: Klargjør data: Hovedprogram

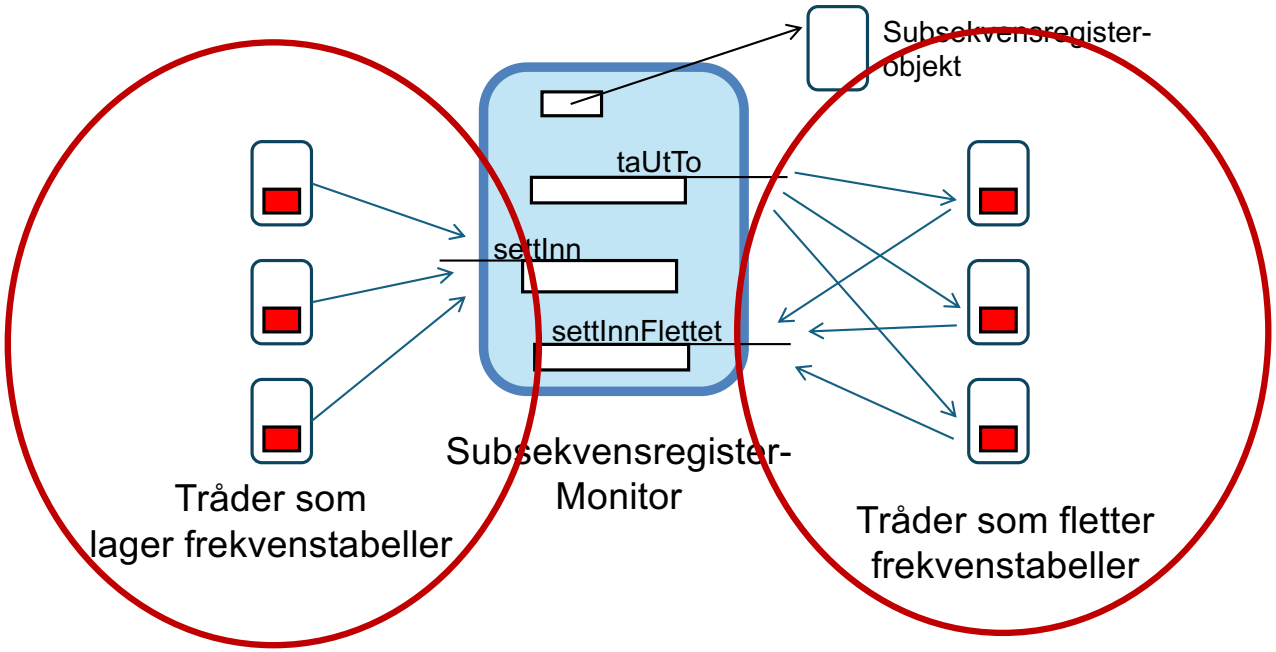
Innlevering 5

- Du skal lage et Subsekvensregister-objekt (oppgave 2)
- Du skal lage en Monitor (et Monitor-objekt) som beskytter dette (oppgave 3 og 5)

- Du skal lage N lesetråder som leser N filer, lager N frekvenstabeller og legger dem inn i Monitoren (oppgave 4)

- Du skal lage 8 flettetråder der hver tråd går i løkke og henter ut to frekvenstabeller fra Monitoren, fletter dem sammen og legger resultatet tilbake i Monitoren.
  Løkken (og alle disse trådene) avsluttes når alle de N frekvenstabellene er flettet sammen til èn. (oppgave 6)

Innlevering 5 kan løses som tre faser:



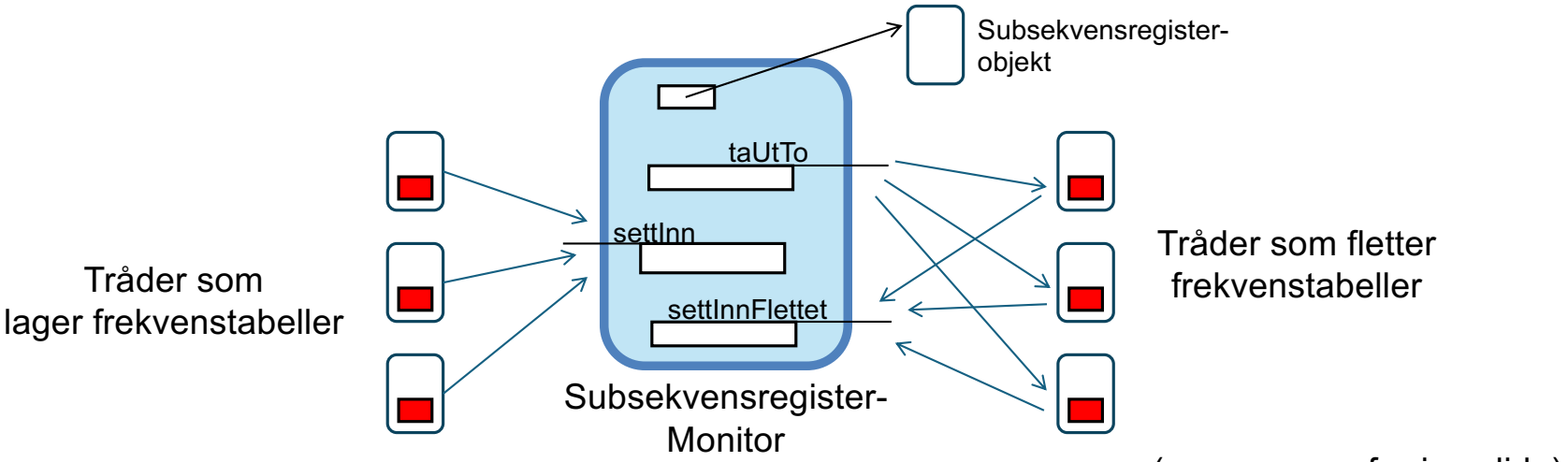
Fase 1: Lesetråder

Fase 2: Flettetråder

Fase 3: Skriv resultat til smittet / ikke_smittet

Hvordan vet man at fletting er ferdig?

- Man må ta hensyn til både antall elementer i registeret og hvor mange som er tatt ut for fletting
  - F.eks kan man ha en teller **uteTilFletting** som teller opp når man kaller **tarUtTo()** og ned når man kaller **settInnFlettet()**
  - Da vet man om det er bare en tabell igjen (uteTilFletting + antall == 1)

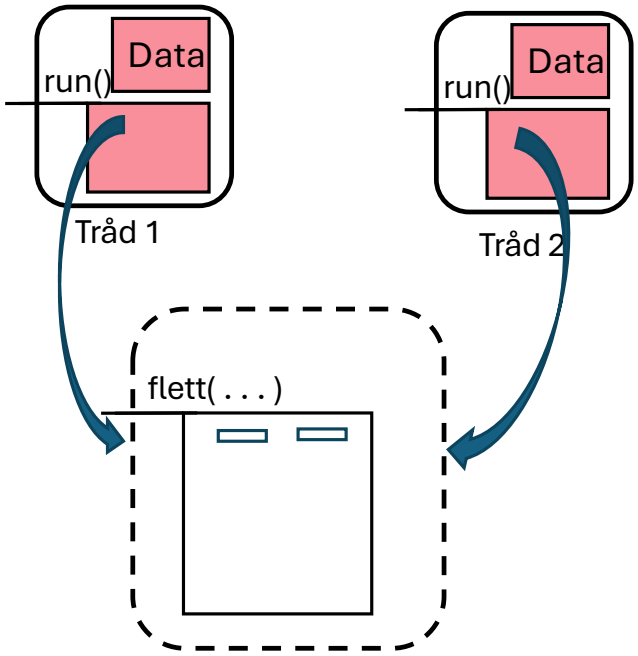


(samme som forrige slide)

Hvordan avslutte trådene?

- Flere alternativer – dette er eksempler
- Fase 1 - innlesing
  - Du kan bruke join() for å vente på lesetrådene, run() må da avslutte når hele filen er lest. Lesetrådene kan ligge i f.eks ArrayList.
- Fase 2 – fletting
  - Hvis man vet at de er ferdig og trådene venter i .await(), kall interrupt() på hver flettetråd fra main(). Da må trådene skrives så de avslutter while-løkken sin hvis de får InterruptedException (se stoppeklokke-eksempel)
  - La taUtTo() returnere null når man vet man er ferdig, da kan flettetrådene selv avslutte når dette skjer. Man kan bruke join eller barriere for å vente på dem i main().
  - System.exit() kan i verste fall kalles fra main for å tvinge avslutting av tråder og programmet (men man må vite at man er ferdig).

Innlevering 5 og bruk av samme metode



Om du programmerer flettingen av frekvenstabeller i en statisk metode som bare bruker lokale variabler, kan alle tråder som vil kalle og bruke denne metoden samtidig gå i ekte parallell.

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

Hvordan vet man at svaret er riktig?

- TestData og Data inneholder filene `ikke_smittet` og `smittet` med riktig resultat
- Kan være lurt å ta vare på en kopi av disse og bruke dem til å sammenlikne sitt eget svar
  - F.eks kan de omdøpes til `smittet.fasit` og `ikke_smittet.fasit`
- Det finnes programmer for å sammenlikne to tekstfiler
  - MacOS/Linux: `diff smittet smittet.fasit`
  - Windows: `fc.exe smittet smittet.fasit`

TreeMap

- Oppgaven angir at Frekvensregister arver fra TreeMap
- TreeMap og HashMap implementerer begge Map
- Nøklene i TreeMap er sortert alfabetisk når du itererer over dem
  - … dermed kan f.eks Frekvenstabells toString() automatisk gi sorterte data

- Lagrer dataene ulikt internt (ikke pensum i IN1010)
  - TreeMap er et rød-svart tre, kreves flere operasjoner for å sette inn elementer, nøkler automatisk sortert
  - HashMap er en hashtabell, få operasjoner for å sette inn/ta ut elementer, men tilfeldig rekkefølge når man itererer