IN1010 - Våren 2020
Tråder
Eksemplet Kokk og servitør

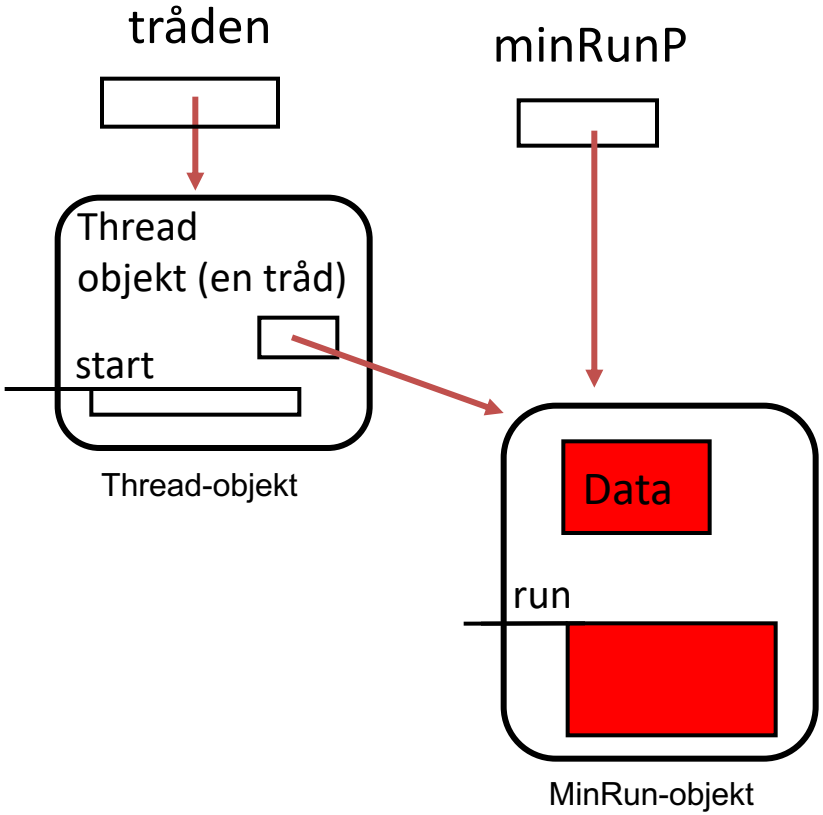
25. mars 2020

Stein Gjessing,

Institutt for informatikk,
Universitetet i Oslo

Tråder i Java I

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



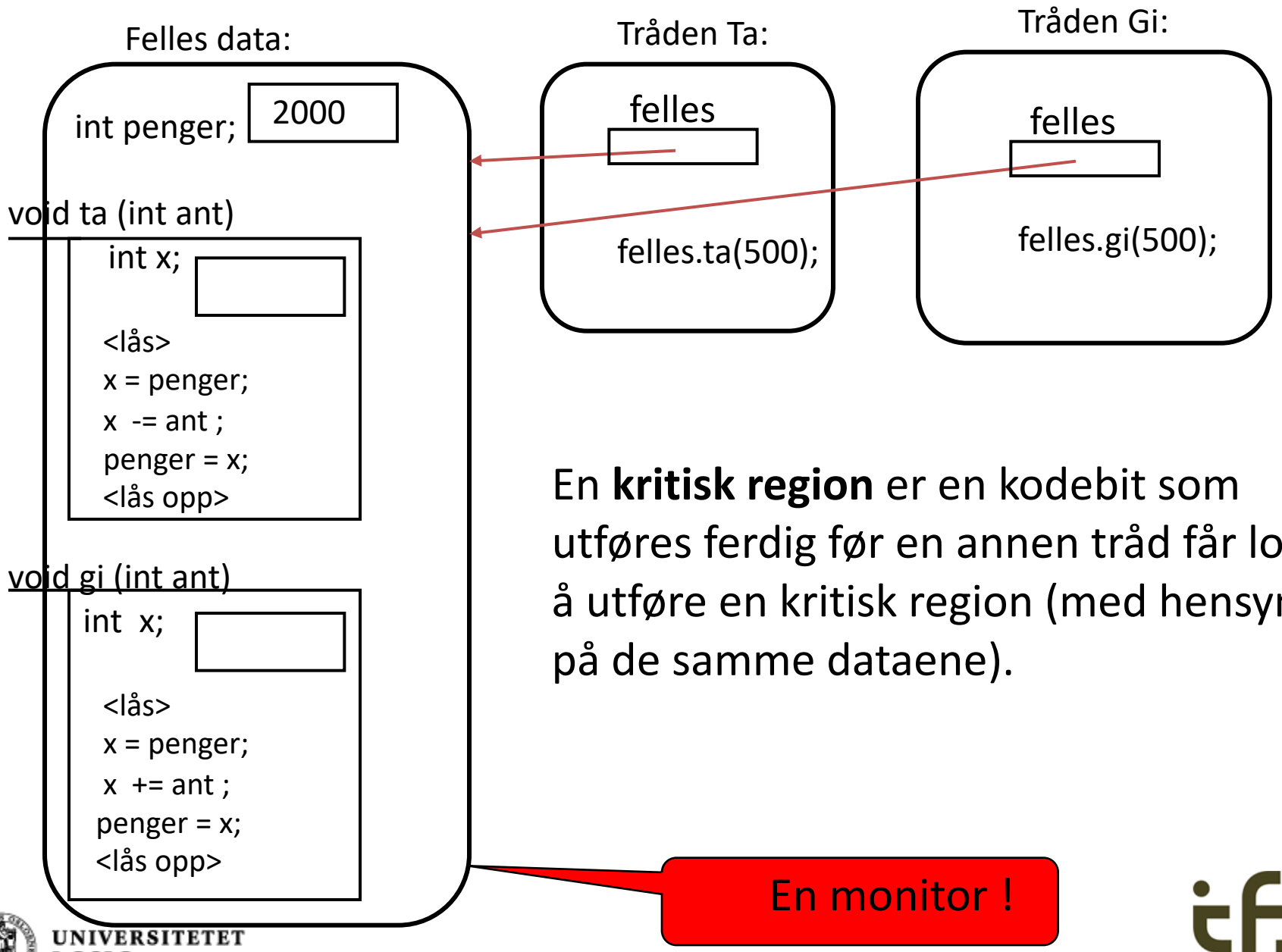
En tråd lages og startes opp slik:

```
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).

Vi ordner dette med **kritiske regioner**.

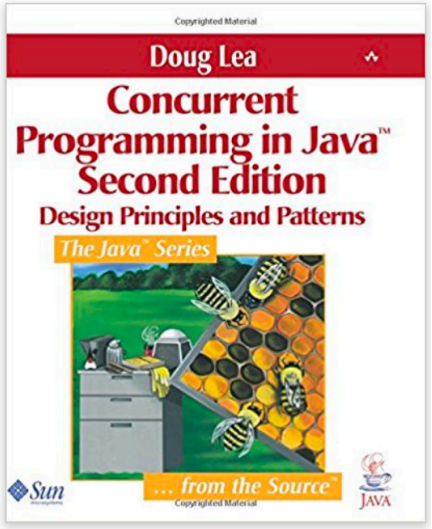


En **kritisk region** er en kodebit som utføres ferdig før en annen tråd får lov å utføre en kritisk region (med hensyn på de samme dataene).

Monitorer i Java

- Opprinnelig ble Java laget med et innebygget monitor-begrep:
  - Alle objekter har en lås
  - synchronized foran en metode gjør metoden til en kritisk region (med hensyn på data i dette objektet)
  - wait() og signal() er metoder i class Object

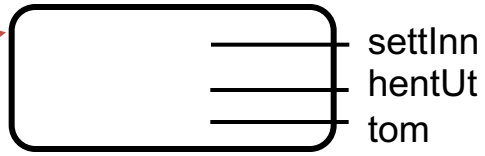
- Doug Lea forbedret dette med Java.util.concurrent-biblioteket



Passiv venting

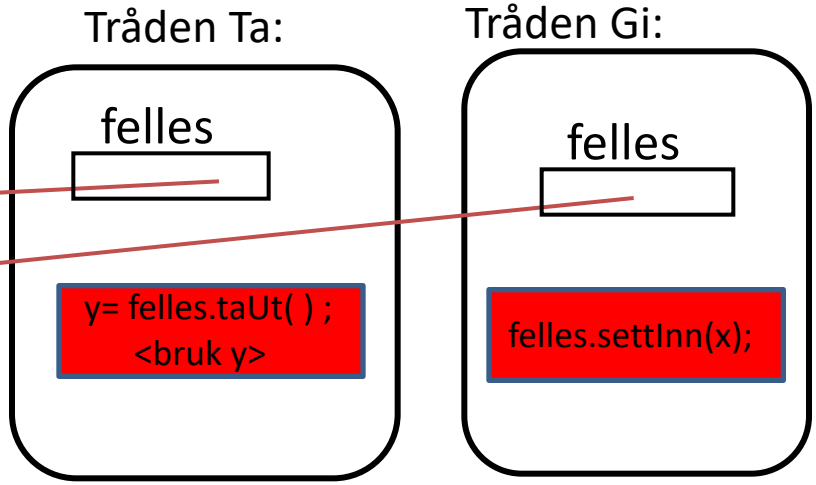
```
Lock laas = new ReentrantLock();
Condition ikkeTom =  laas.newCondition();

Beholder behold;
```

Objekt av klassen Beholder




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

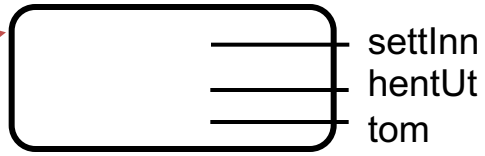
```
public Ting taUt ( ) throws InterrupedException
    laas.lock();
    try {
       if (behold.tom())
              { ikkeTom.await(); }
        return (behold.hentUt());
    }  finally {
      laas.unlock()
    }
```



Passiv venting

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

Beholder behold;
```

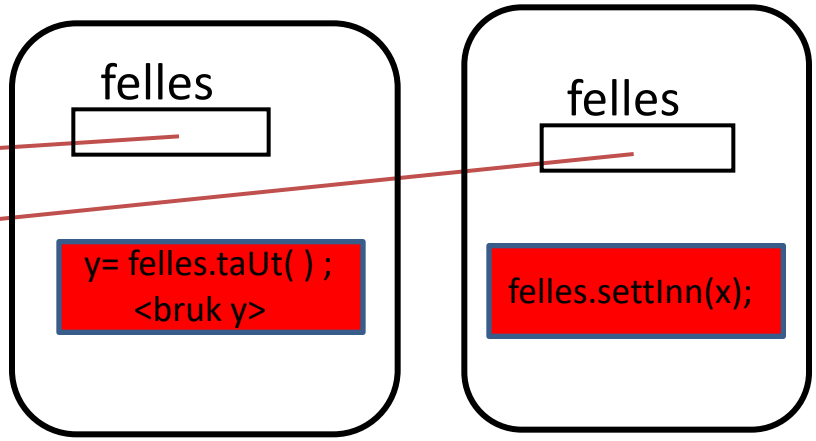
Objekt av klassen Beholder




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

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

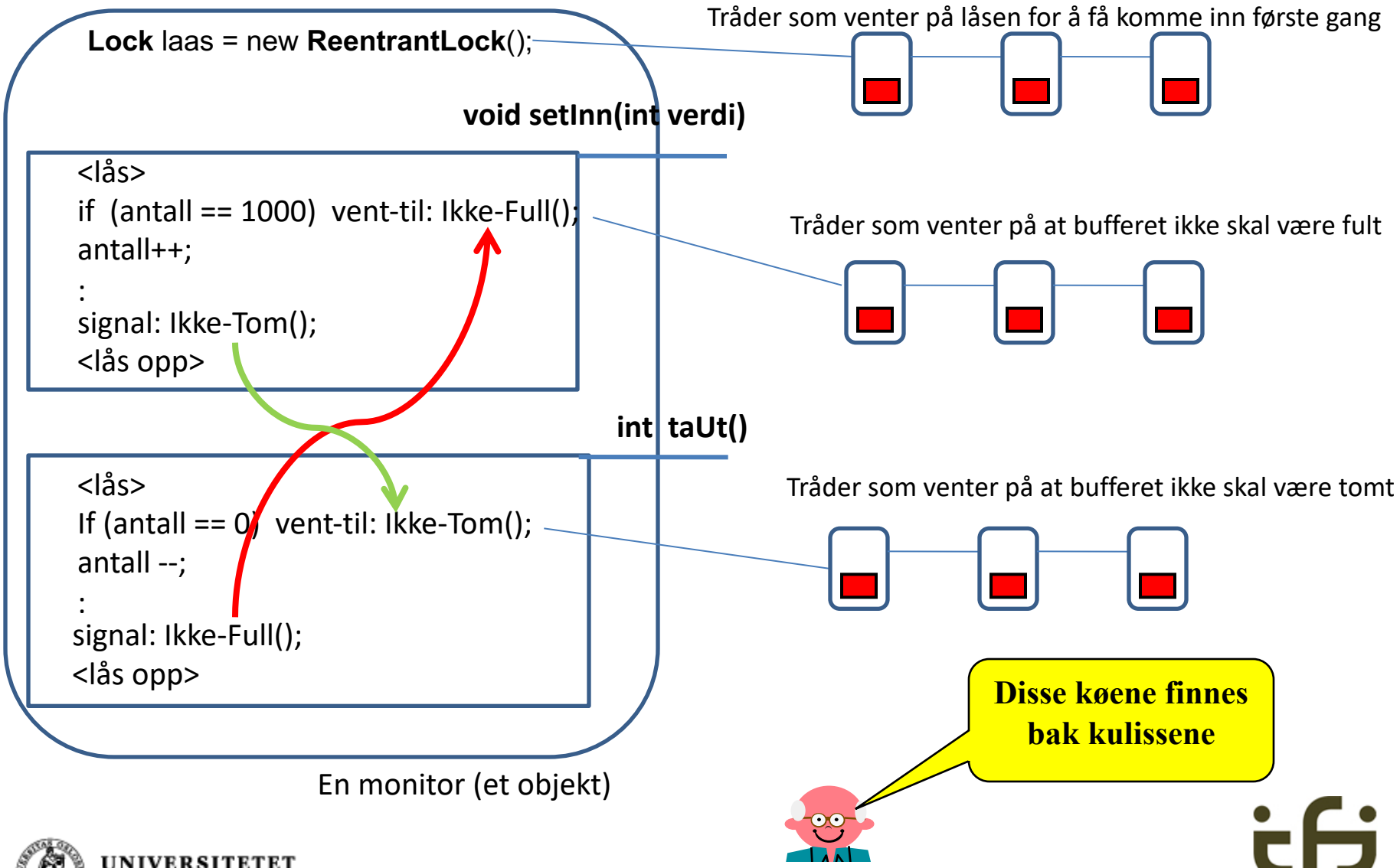
Tråden Ta:

Tråden Gi:

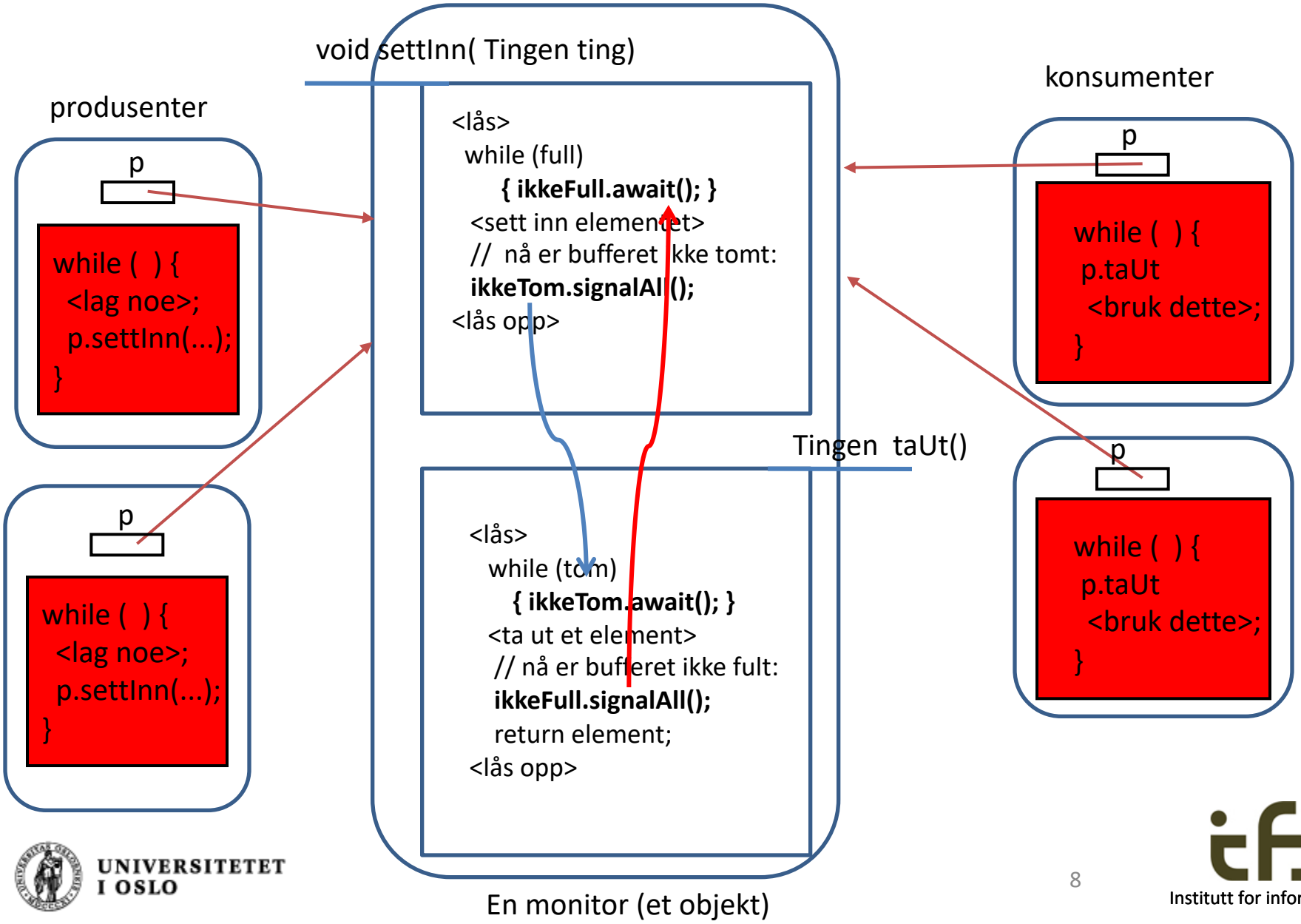


Når noe settes inn vil bufferen ikke lenger være tom

Ofte har vi flere grunner til å vente i en monitor:



Produsenter og konsumenter



Et større litt større eksempel

Kokk og servitør

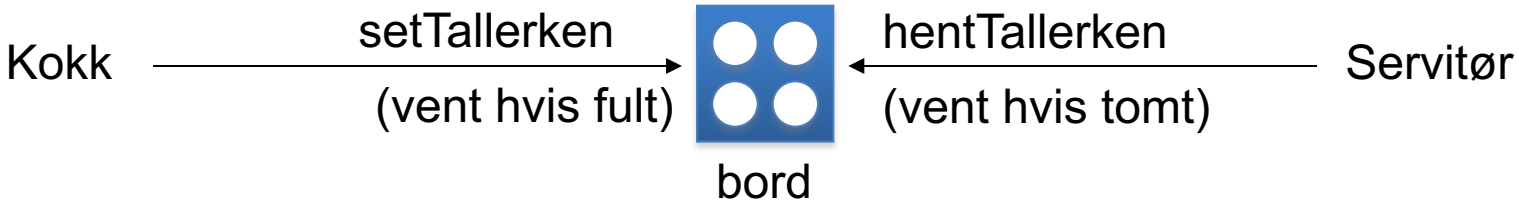
Kokk og servitør

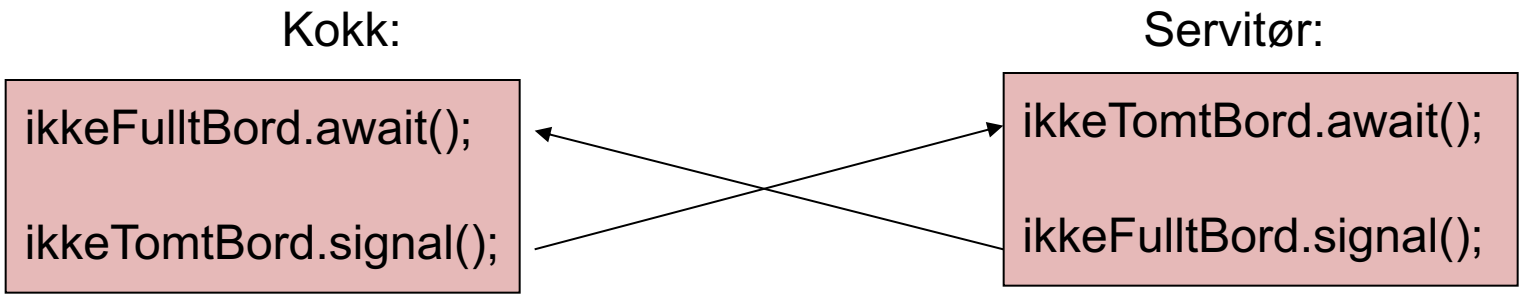
- Kokken lager mat og setter en og en tallerken på et bord
- Servitøren tar en og en tallerken fra bordet og serverer
- Kokken må ikke sette mer enn BORD_KAPASITET tallerkener på bordet (maten blir kald)
- Servitøren kan selvsagt ikke servere mat som ikke er laget (bordet er tomt)

- Her: en kokk og en servitør – Oppgave: lag flere av hver.

await(); signal();

- Kokken må vente når det allerede er fire tallerkener på bordet
- Servitøren må vente når det ikke er laget noe mat (ingen tallerkener på bordet)
- Kokken må starte opp kelneren igjen når han har satt tallerken nr. 1 på bordet (eller alltid når han har satt en tallerken på bordet ?)
- Servitøren må starte opp kokken igjen når han tar tallerken nr. 4 fra bordet (eller alltid når han tar en tallerken fra bordet ?)





Kokk og Serviør

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class RestaurantCB {

    RestaurantCB(String[] args) {
        int antall = Integer.parseInt(args[0]);
        FellesBord bord = new FellesBord();
        Kokk  kokk = new Kokk(bord,antall);
                new Thread(kokk).start();
        Servitor  servitor = new Servitor(bord,antall);
                new Thread(servitor).start();
    }

    public static void main(String[] args) {
                        new RestaurantCB(args);
    }
```

```
class FellesBord {  // En monitor
    private int antallPaBordet = 0;
      /* Invariant: 0 <= antallPaBordet <= BORD_KAPASITET */
    private final int BORD_KAPASITET = 5;

    private Lock bordlas = new ReentrantLock();
    private Condition ikkeTomtBord = bordlas.newCondition();
    private Condition ikkeFulltBord = bordlas.newCondition();

   // class FellesBord fortsetter neste side
```

```
    void settTallerken() throws InterruptedException {
        bordlas.lock();
        try {
            while (antallPaBordet >= BORD_KAPASITET) {  ikkeFulltBord.await();  }
                    // Na er antallPaBordet < BORD_KAPASITET
                    antallPaBordet++;
                    ikkeTomtBord.signal();    /* Si fra til den som tar tallerkener. */
        }
       finally {  bordlas.unlock();  }
    }

    void hentTallerken() throws InterruptedException  {
         bordlas.lock();
         try {
              while (antallPaBordet == 0) { ikkeTomtBord.await();  }
                   // Na er antallPaBordet > 0
                  antallPaBordet --;
                 ikkeFulltBord.signal();
         }
         finally {  bordlas.unlock();   }
    }
}
```

```
class Kokk implements Runnable {
      private FellesBord bord;
      private final int ANTALL;      private int laget = 0;
      Kokk(FellesBord bord, int ant) {
            this.bord = bord;mmmANTALL = ant;
     }
     public void run() {
         try {
           while(ANTALL != laget) {
                laget ++;
                bord.settTallerken();  // lag og lever tallerken
                Thread.sleep((long) (500 * Math.random()));
            }
          }
          catch (InterruptedException e) {
               System.out.println("Uventet stopp 1");
           }
         // Kokken er ferdig
      }
   }
```

```
class Servitor implements Runnable {
    private FellesBord bord;
    private final int ANTALL;    private int servert = 0;
    Servitor(FellesBord bord, int ant) {
        this.bord = bord;    ANTALL = ant;
    }
    public void run() {
        try {
            while (ANTALL != servert)  {
                bord.hentTallerken(); /* hent tallerken og server */
                servert++;
                Thread.sleep((long) (400 * Math.random()));
            }
        }
        catch (InterruptedException e) {
                System.out.println("Uventet stopp 2");
        }
       // Servitoren er ferdig
    }
  }
}
```

Enda mer om monitorer, kritiske regioner og Big Java

- Kritiske regioner (ekskusiv tilgang til felles data) kan løses på mange måter
- Det er bl. a. noe som heter *semaforer* (P og V)
  - og aktiv venting (spin locks)
- I IN1010 bruker vi *monitorer* fordi det er mest objektorientert.
- Venting og signalering i monitorer kan gjøres på forskjellige måter.
  - I dag: En god måte, og slik det gjøres i Horstmann
  - Kanskje senere i vår: Slik det gjøres innebygget i Java (notify og wait, ikke pensum i IN1010)

Oppsummering hittil: Vi har lært så langt om tråder:

- Hva tråder brukes til
- Hvordan vi lager tråder i Java
- Hvordan tråder kommuniserer seg imellom ved hjelp av monitorer
- Hvordan programmet venter i en monitor
  - og starter opp igjen de som venter