INF1010 11. mai 2017

Monitorer med kritiske regioner og passive venting innbygget i Java - Kommunikasjon mellom prosesser i Java

(Ikke pensum i INF1010)

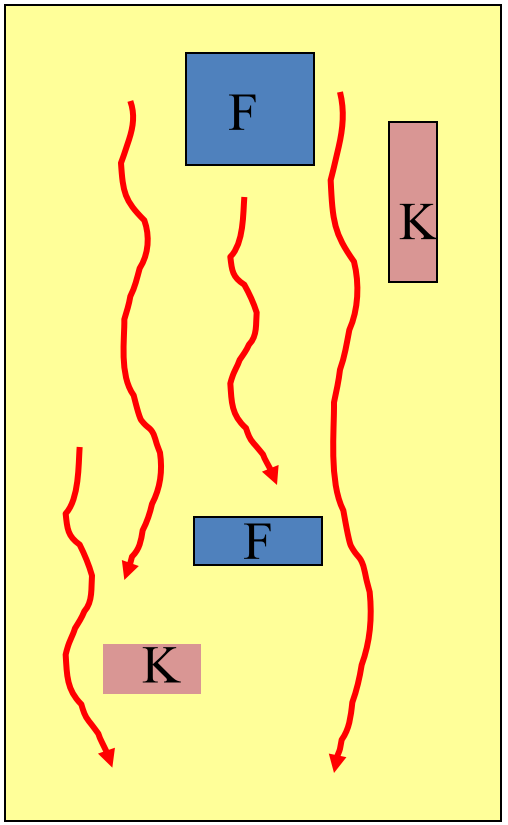
Stein Gjessing

Institutt for informatikk

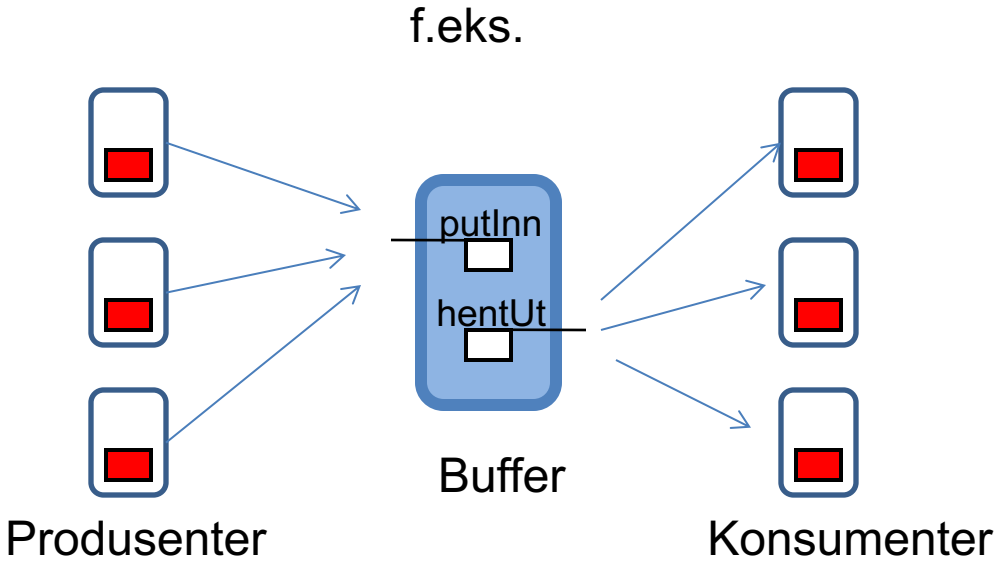
Universitetet i Oslo

Repetisjon: 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 er kritiske regioner



K: konstante data (immutable)

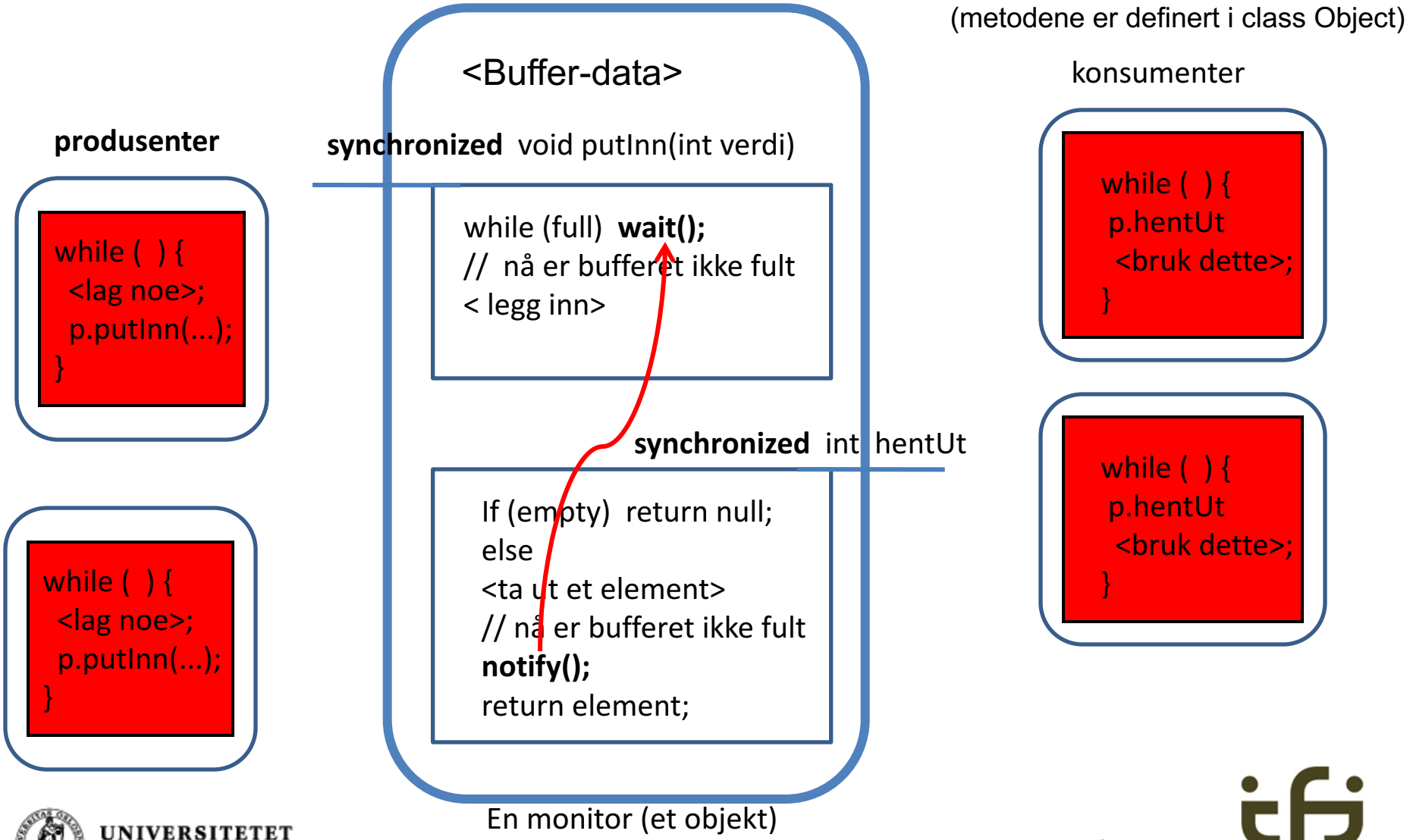


UNIVERSITETET I OSLO

Institutt for informatikk

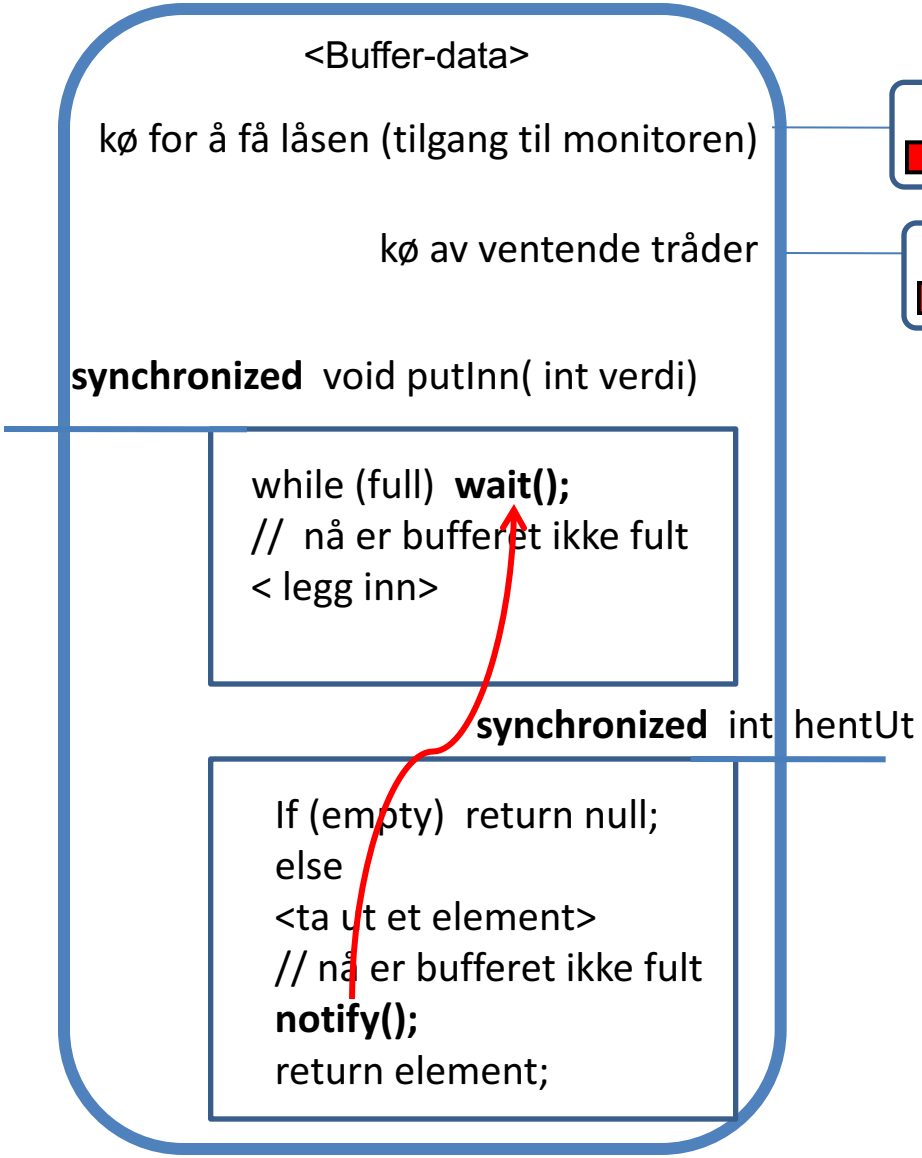
Basale Java-verktøy: synchronized, wait(), notify() og notifyAll()

(metodene er definert i class Object)



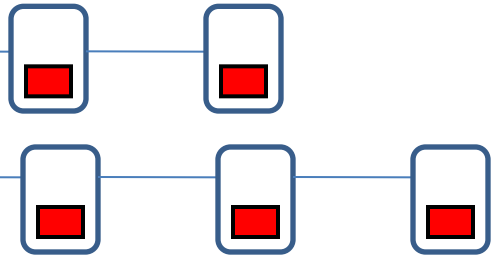
En monitor (et objekt)

To køer i en basal Java monitor:



En monitor (et objekt)

En kø av ventende tråder på hele monitoren



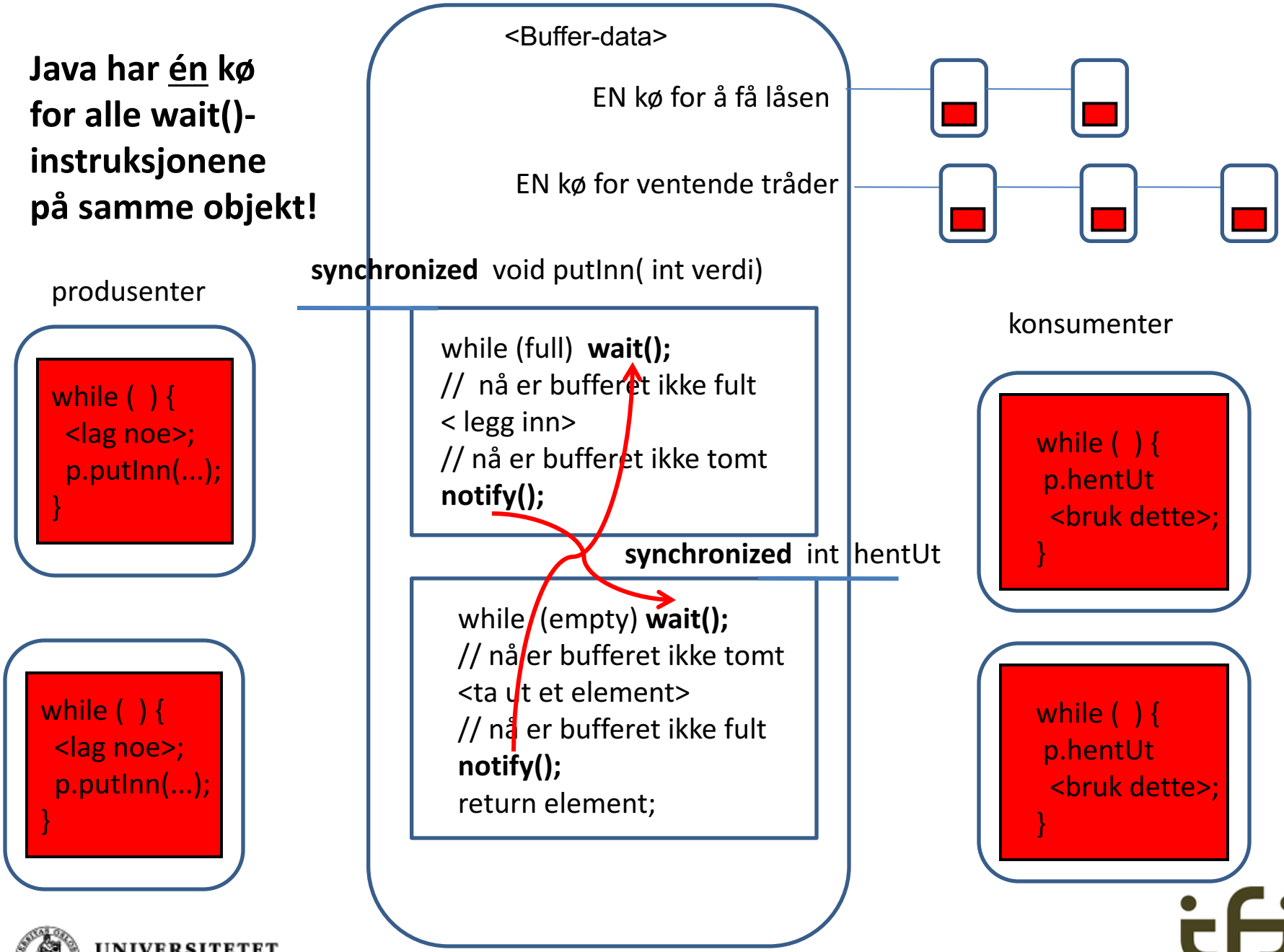
En kø av ventende tråder på ”wait”-instruksjoner (wait-set).

Startes av notify () og/eller notifyAll()

Legges da i den andre køen (først ? (Nei, ingen garanti))

Derfor er det nødvendig med ”while …”

Java har <u>én</u> kø for alle wait()-instruksjonene på samme objekt!

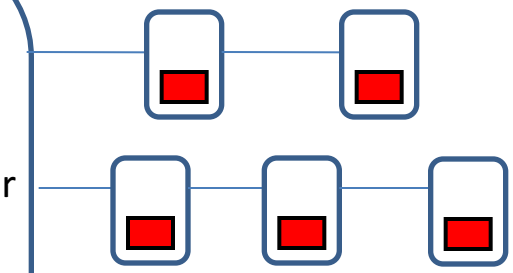


En monitor (et objekt)

<Buffer-data>

kø for å få låsen

ventende tråder



Flytt en: notify ();

Flytt alle: notifyAll ();

```
int ledig = MAX;
//invariant: 0 <= ledig <= MAX

synchronized  void reserver(int antall)
    while (ledig < antall)  wait();
    //  nå er det nok ledig
    ledig = ledig – antall;

synchronized  void  frigi(int antall)
    ledig = ledig + antall;
    // nå er mange flere ledig
    notifyAll();
```

En monitor (et objekt) for å reservere og frigi et antall like resurser

Hvis du ikke er HELT sikker på at enhver ventende tråd kan ordne skikkelig opp må du si:

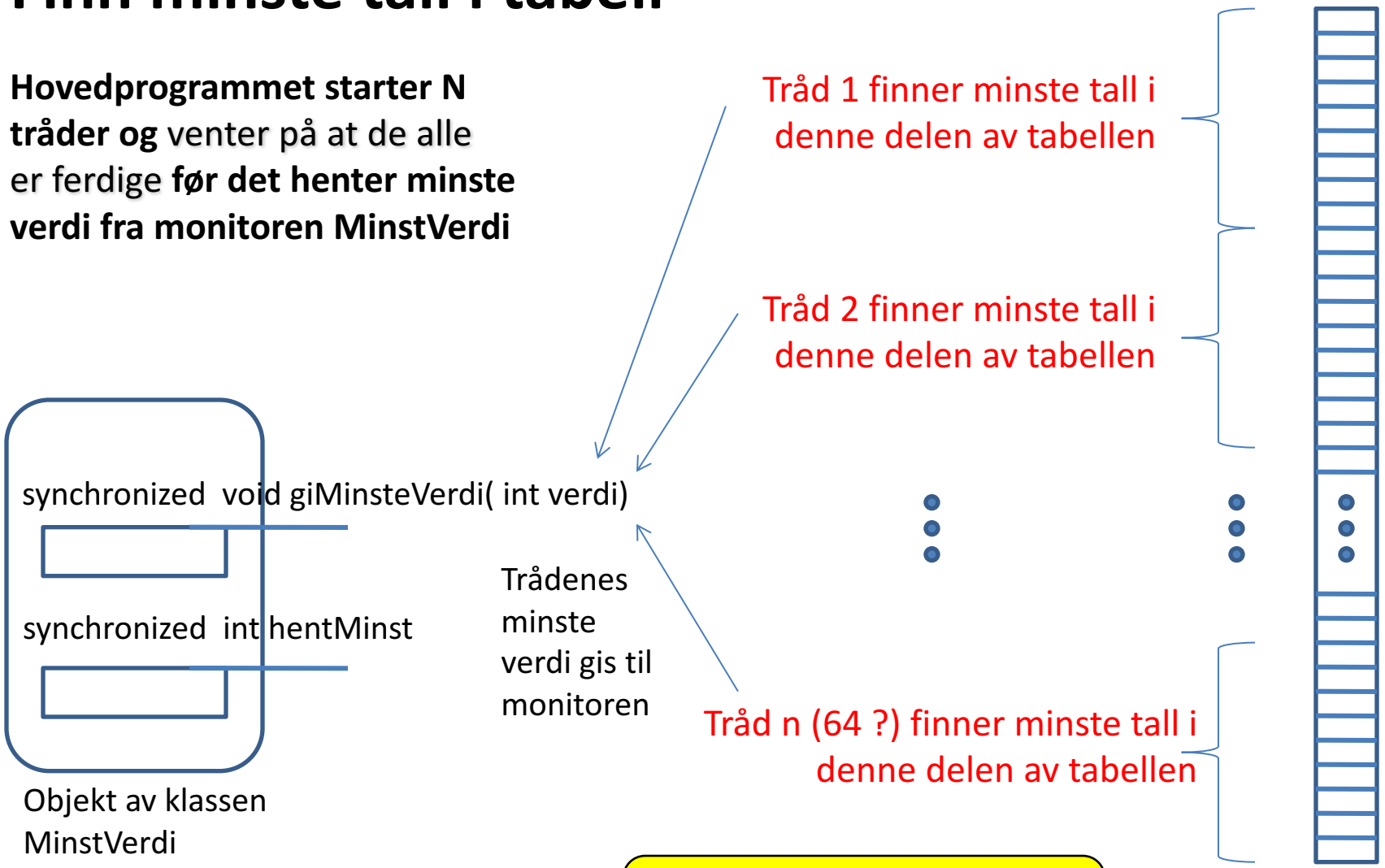
notifyAll();

Da blir ALLE ventende tråder flyttet opp til køen for å få **låsen**

Innebygget i Java: Én lås per objekt

Eksempel på parallellisering: Finn minste tall i tabell

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



BARE FANTASIEN SETTER GRENSER

Hovedprogrammet versjon 1

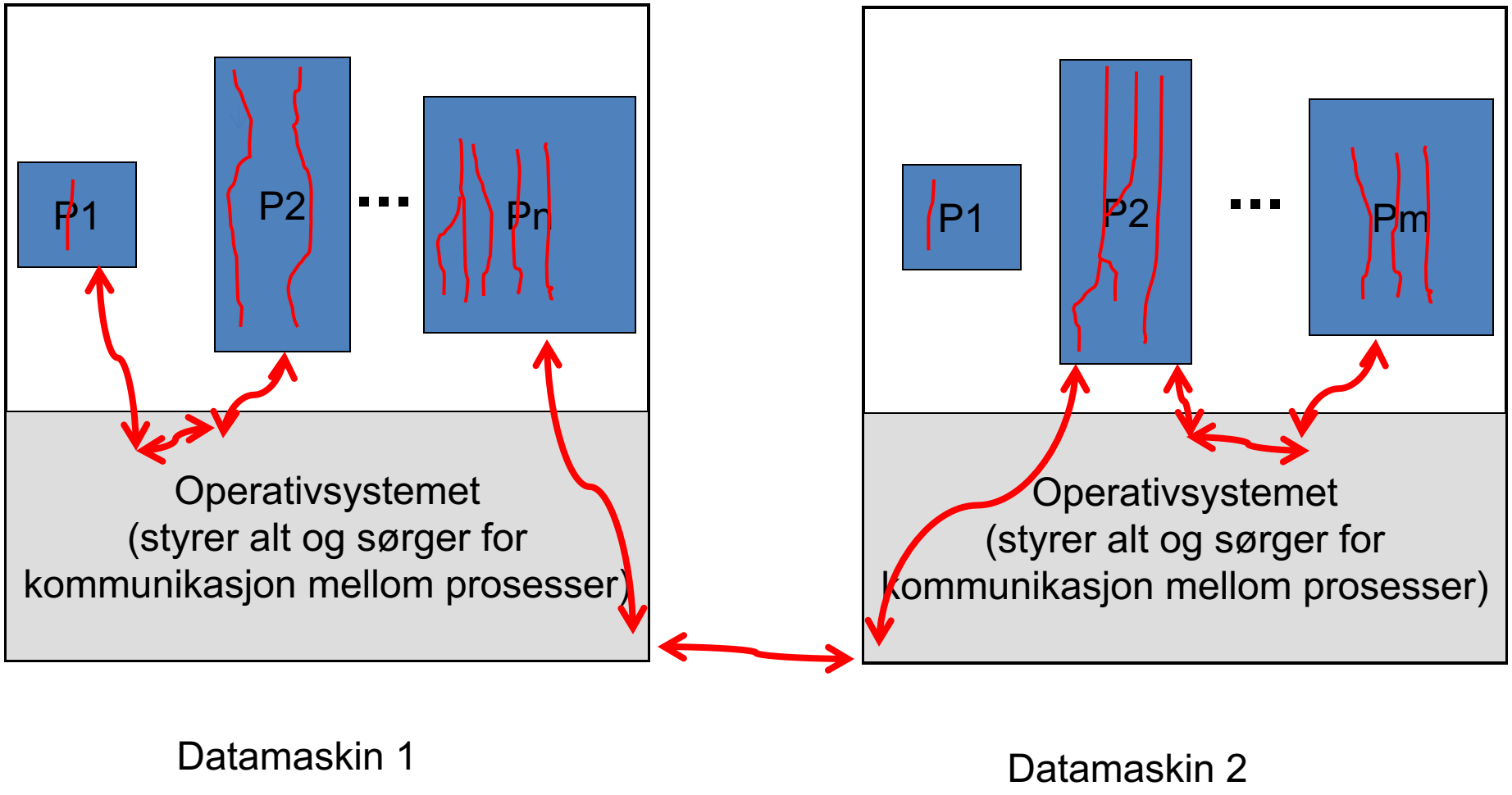
```
public class MinstR {
    final int maxVerdiInt = Integer.MAX_VALUE;
    int [ ] tabell;
    MinstMonitorR monitor;

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

    public MinstR ( ) {
         tabell = new int[640000];
         for (int in = 0; in< 640000; in++)
                  tabell[in] = (int)Math.round(Math.random()* maxVerdiInt);
         monitor = new MinstMonitorR();
         for (int i = 0; i< 64; i++)
             // Lag og start 64 tråder
             new MinstTradR(tabell,i*10000,((i+1)*10000)-1,monitor).start();
         monitor.vent();
         System.out.println("Minste verdi var: " + monitor.hentMinste());
    }
}
```

```
class MinstMonitorR {
   private int minstTilNa = Integer.MAX_VALUE;
   private int antallFerdigeSubtrader = 0;
   synchronized public void vent() {
       while (antallFerdigeSubtrader != 64) {
              try {wait();
                  System.out.println(antallFerdigeSubtrader + "
                                                  ferdige subtråder ");
              }
              catch (InterruptedException e) {
                      System.out.println(" Uventet avbrudd ");
                      System.exit(1);
              }
       // antall ferdige subtråder er nå 64
       }
   }
   synchronized public void giMinsteVerdi (int minVerdi) {
      antallFerdigeSubtrader ++;
      if(minstTilNa > minVerdi) minstTilNa = minVerdi;
      if(antallFerdigeSubtrader == 64) notify();
      // eller hver gang, (men stort sett unødvendig): notify();
   }
   synchronized public int hentMinste () {
       return minstTilNa;
   }
}
```

Tråder som finner minste tall i del av tabellen

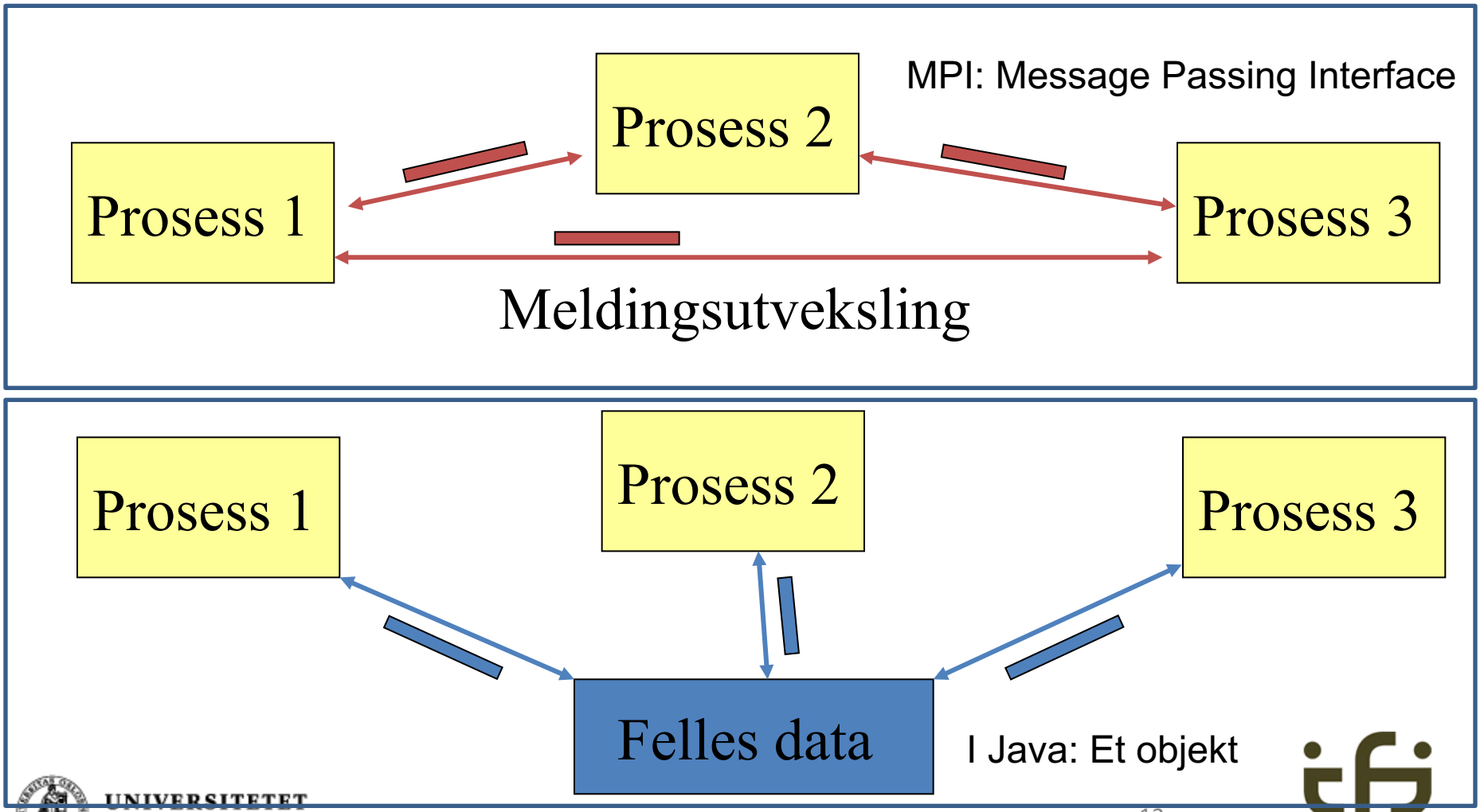
```
class MinstTradR extends Thread {

    int [ ] tab; int startInd, endInd;
    MinstMonitorR mon;

    MinstTradR(int [ ] tb, int st, int en, MinstMonitorR m) {
      tab = tb; startInd = st; endInd = en;
      mon = m;
    }

    public void run(){
      int minVerdi = Integer.MAX_VALUE;
      for ( int ind = startInd; ind <= endInd; ind++)
                  if(tab[ind] < minVerdi) minVerdi = tab[ind];
       // denne tråden er ferdig med jobben:
       mon.giMinsteVerdi(minVerdi);
     } // slutt run

} // slutt MinstTradR
```

Prosesser vs. tråder



Kommunikasjon mellom prosesser

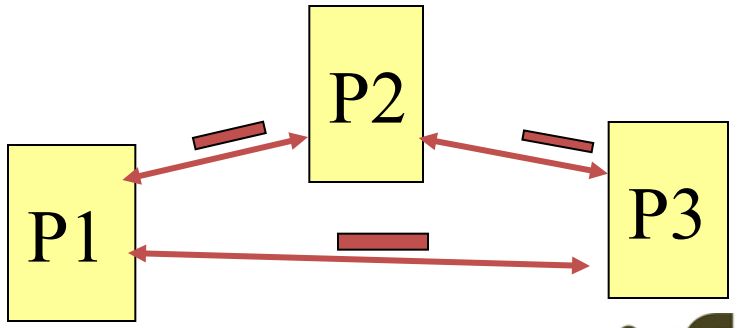
Samarbeidende prosesser sender meldinger til hverandre (brune piler, mest vanlig) **eller** leser og skriver i felles lager (blå piler).



Parallellitet mellom prosesser: Eks: CORBA, MPI

- CORBA (Common Object Request Broker Architecture) er (var) en språkuavhengig måte å definere kommunikasjon mellom prosesser vha. fjern-objekter
  Et IDL (Interface Definition Language) definerer grensesnittet til objektene (på samme måte som Interface i Java)
- En IDL-kompilator oversetter til ditt valgte språk (Java, C++, C#, Smalltalk, …)
- Dette gjør at prosesser skrevet i forskjellige objektorienterte språk og som kjører på forskjellige (eller samme) maskin kan kommunisere.

- Språk som ikke er objektorienterte:
  Sende og motta meldinger
  (MPI – Message-Passing Interface)



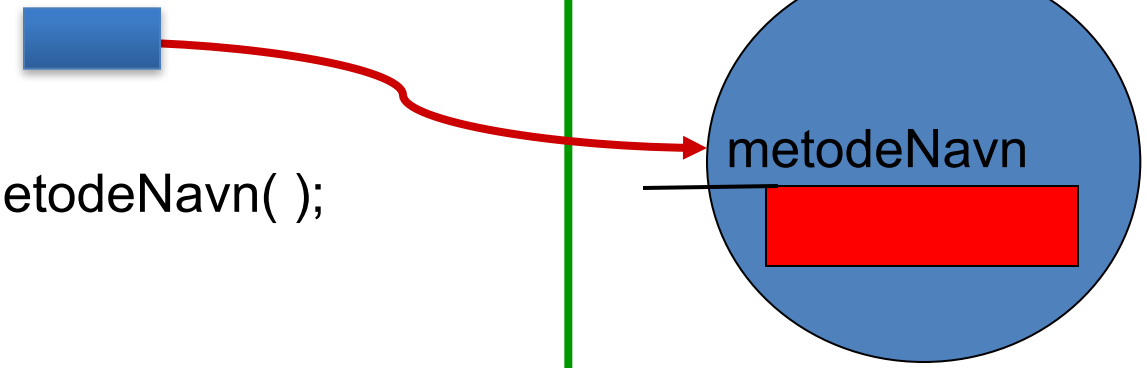
Kommunikasjon mellom Java-**prosesser**: RMI: Remote Method Invocation

(Norsk: Fjern-metode-kall)

Dette ønsker vi å oppnå:

Et Java program
(på en maskin)

```
InterfaceNavn fjernPeker = …….;

. . .
fjernPeker.metodeNavn( );
. . .
```



Et annet Java program
(på en annen maskin)

objekt av en klasse som implementerer *InterfaceNavn*

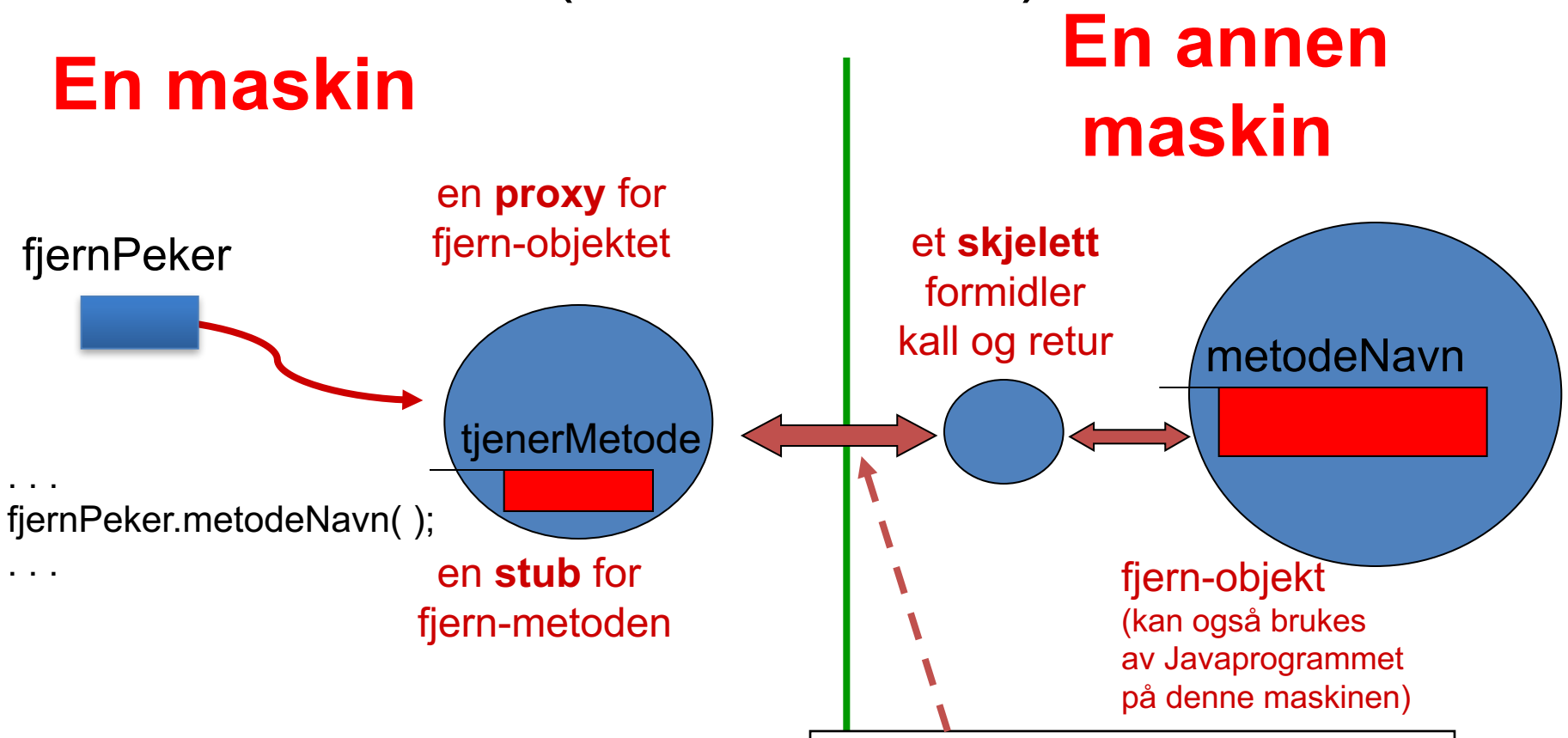
UNIVERSITETET I OSLO

Alternativt navn: RPC: Remote Procedure Call

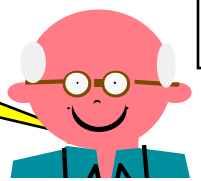
Institutt for informatikk

RMI: Implementasjon
(bak kulissene)

Ikke pensum i INF1010

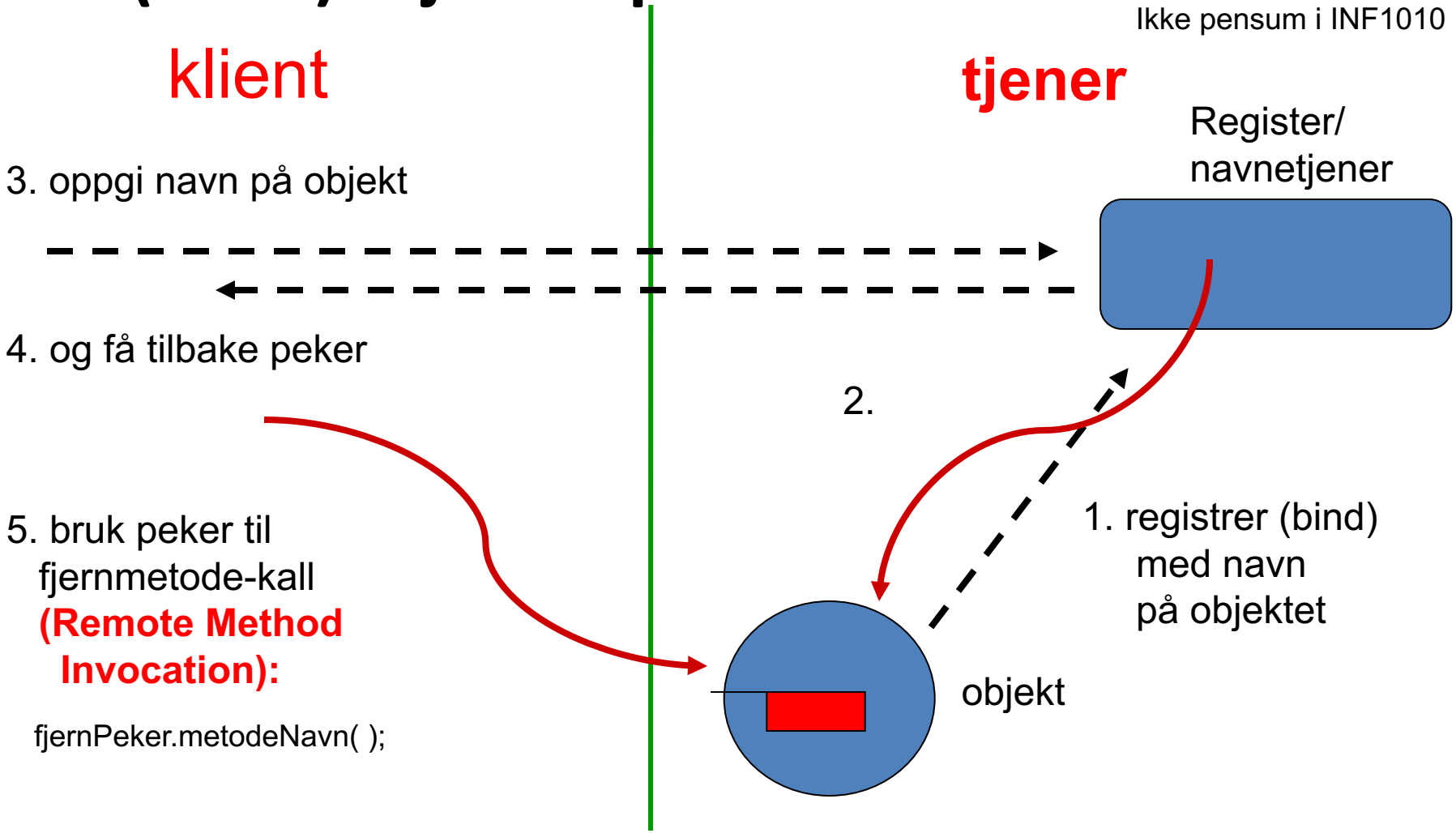


Odene proxy og stub brukes ikke alltid like konsistent



RMI: Hvordan kalle metoder i (Java-)objekter på andre maskiner

Ikke pensum i INF1010



Og det eneste som er kjent på begge maskinene er Interfacet til objektet og navnet