UNIVERSITETET I OSLO

Institutt for informatikk

Invarianter, tilstander, litt mer semantikk, litt mer monitorer (conditions), RMI (Remote Method Invocation)

INF1010 – 29. april 2020

Stein Gjessing

Institutt for informatikk

Universitetet i Oslo

Invarianter

- Som informatiker må vi beskrive **tilstander** i programmet vårt
  - En beskrivelse av en tilstand er en beskrivelse av verdier på – og sammenhenger mellom - variabler
- Tilstander og tilstandsoverganger styrer programmet vårt (se neste side)

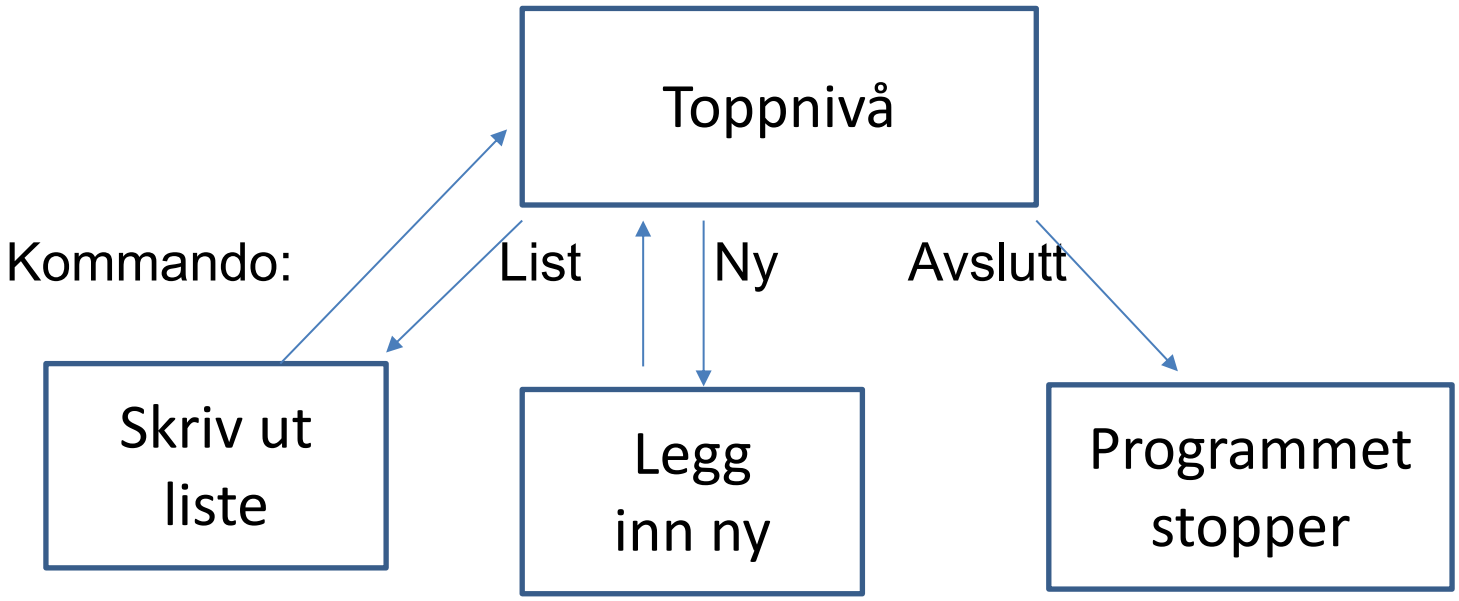
- Når du lager en **løkke** er det alltid en beskrivelse av en tilstand (en **invariant**) som sier hvor langt arbeidet i løkka er kommet
- Data i et **objekt** er alltid styrt av en (eller flere) beskrivelser av sammenhengene mellom dataene I objektet (**invarianter** (konsistensregler))

- En beskrivelse av en tilstand = engelsk: an **assertion /** a **condition**
- En **invariant** er en assertion/condition som gjelder (nesten) hele tiden

- På norsk: antagelse / betingelse

Tilstander og tilstandsoverganger

- For eksempel et kommandosystem:



Tilstandsdiagram. Engelsk: state diagram

WIKIPEDIA:

Invariant (computer science)

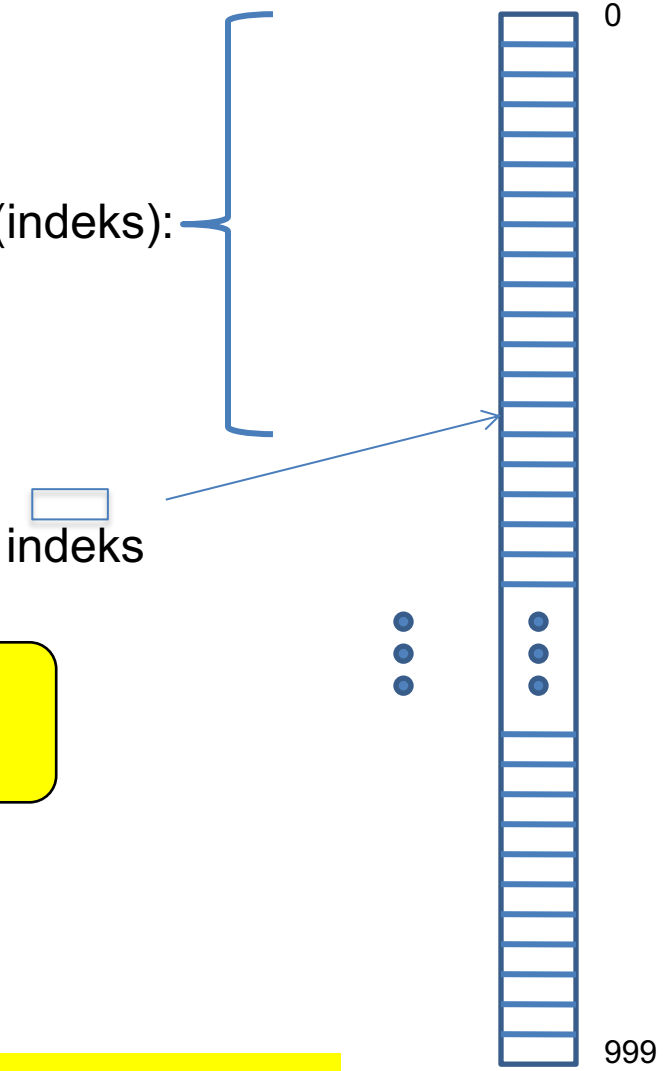
- In computer science, an **invariant** is a condition that can be relied upon to be true during execution of a program, or during some portion of it. It is a logical assertion that is held to always be true during a certain phase of execution.
  For example, a **loop invariant** is a condition that is true at the beginning and end of every execution of a loop.
  . . . . .
- Programmers often use assertions in their code to make invariants explicit. Some object oriented programming languages have a special syntax for specifying **class invariants.**

Invarianter på data i løkker

Eksempel: **Finne minste verdi i tabell**

minstTilNaa inneholder minste verdi i området fra og med tabell [0] til og med tabell[indeks]

= Inv(indeks):



Induksjons-basis: indeks = 0

Induksjons-skritt: indeks = indeks + 1

Resultat: indeks = 999:

minstTilNaa inneholder minste verdi i området fra og med tabell [0] til og med tabell[999]

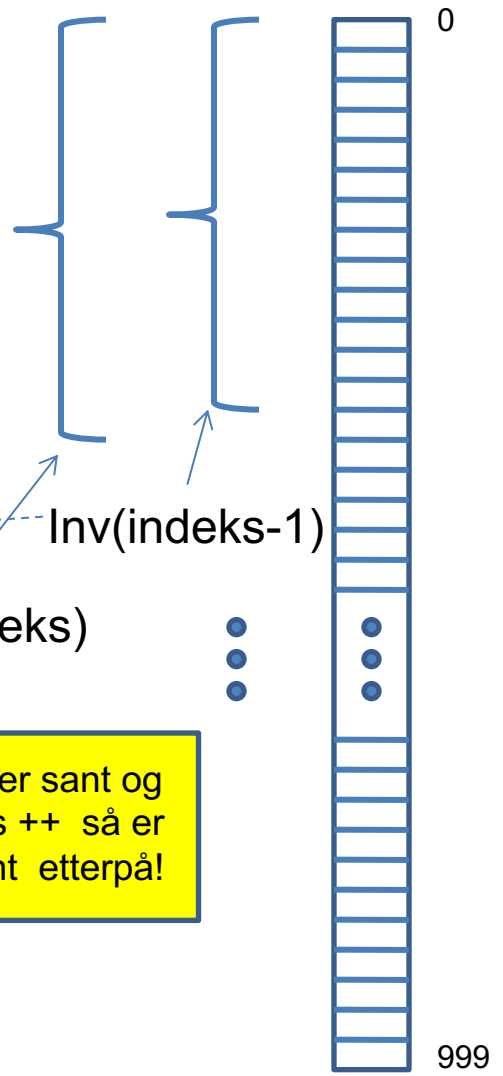
Invarianter på data i løkker

Eksempel: **Finne minste verdi i tabell**


```
// Vi vet ingenting annet enn at tabell [0] til og med
// tabell[999] inneholder tall. Vi skal finne det minste

int minstTilNaa = tabell[0];

  // minstTilNaa inneholder minste verdi i området
  //  fra og med tabell [0] til og med tabell[0]

for (indeks = 1; indeks < 1000; indeks ++) {

    // minstTilNaa inneholder minste verdi i området
    //  fra og med tabell [0] til og med tabell[indeks-1]

    if (minstTilNaa > tabell [indeks] ) minstTilNaa = tabell[indeks]

    // minst TilNaa inneholder minste verdi i området
    //  fra og med tabell [0] til og med tabell[indeks]
}

// Nå er indeks == 1000
 // minstTilNaa inneholder minste verdi i området
 //  fra og med tabell [0] til og med tabell[indeks-1]
// Da følger:
 // minstTilNaa inneholder minste verdi i området
 //  fra og med tabell[0] til og med tabell[999] !!!!!!
```

Inv(0)

Induksjons-basis



Hvis Inv(indeks) er sant og vi utfører: indeks ++ så er Inv(indeks-1) sant etterpå!

Induksjons-skritt

resultat

Invarianter på data i løkker

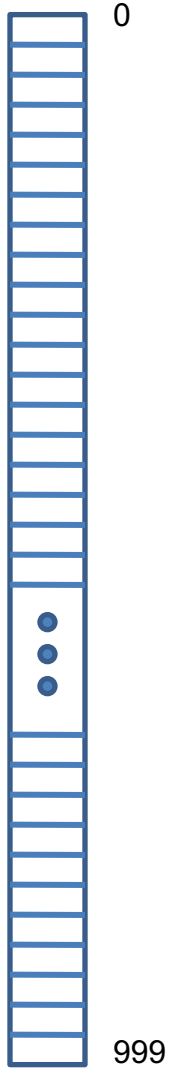
Eksempel: **Finne minste verdi i tabell**


```
// Vi vet ingenting annet enn at tabell [0] til og med
// tabell[999] inneholder tall. Vi skal finne det minste
```

For-betingelse. Engelsk: Pre-condition

Bak-betingelse.
Engelsk: Post-condition (Post-assertion)

```
// minstTilNaa inneholder minste verdi i området
 //  fra og med tabell[0] til og med tabell[999] !!!!!!
```



Pre-condition
Post-condition (post assertion)

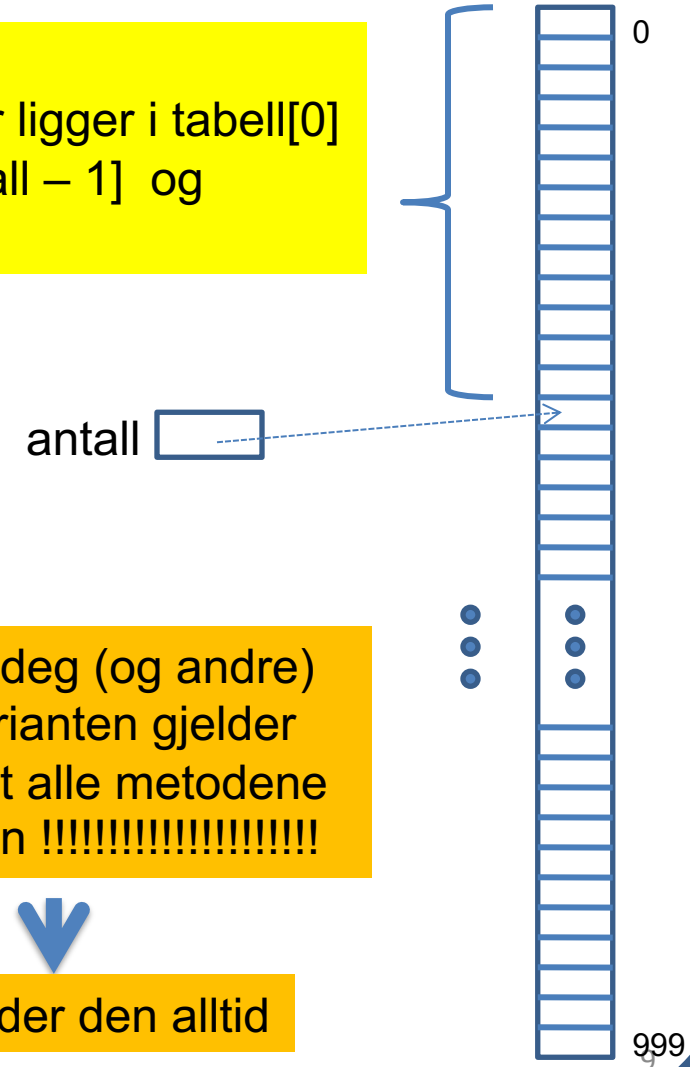
- Det er helt vanlig at parameterene til en metode må ha visse verdier
  - Hvis ikke vil ikke metoden gjøre jobben sin riktig
  - Dette kalles metodens «pre-condition»
    - «Forbetingelse» på norsk
- Når en metode er kalt med riktig forbetingelse vil tilstanden når metoden terminerer kunne beskrives av metodens «bak-betingelse» ☺
  - Post assertion (post-condition)

Invarianter på data i objekter

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 invarianten gjelder initielt og at alle metodene bevarer den !!!!!!!!!!!!!!!!!!!

Da gjelder den alltid

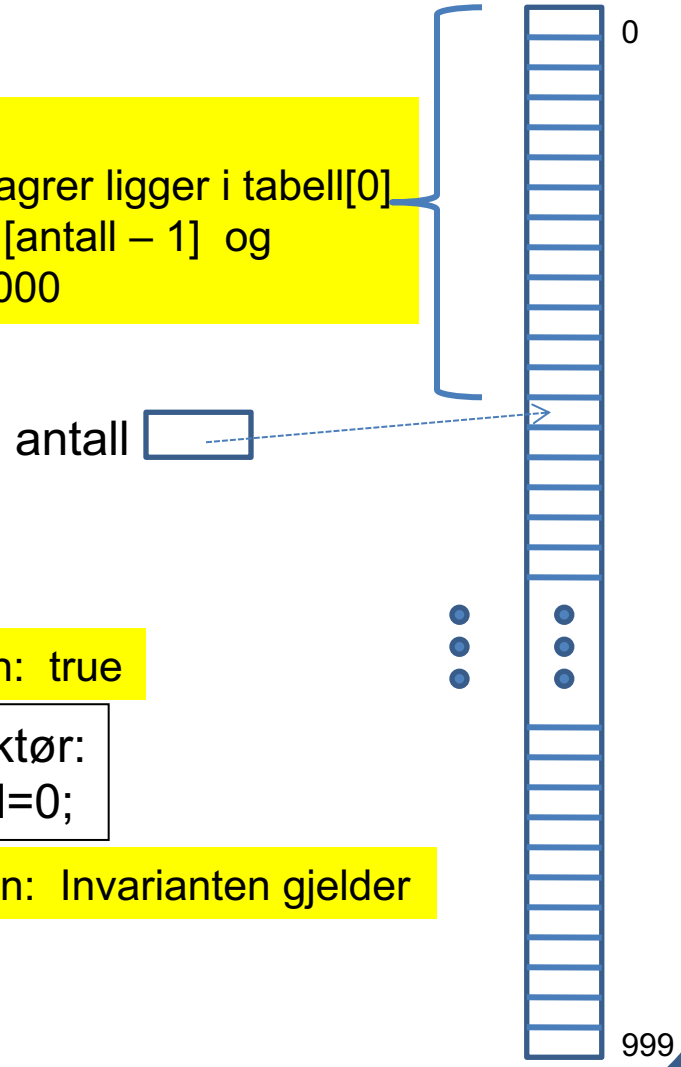
Invarianter på data i objekter

Pre-condition: Invarianten gjelder

```
settInn(x) {
    if (antall == 1000) return ;
    antall ++;
    tabell[antall-1] = x;
}
```

Post-condition: Invarianten gjelder

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



Pre-condition: Invarianten gjelder

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

Post-condition: Invarianten gjelder

Pre-condition: true

```
Konstruktør:
    antall=0;
```

Post-condition: Invarianten gjelder

Spesifikasjon av objekters oppførsel (semantikk) ved hjelp av sekvens av metodekall:

Hvis objektet (på forrige side (en stack)) har hatt følgende sekvens av kall:

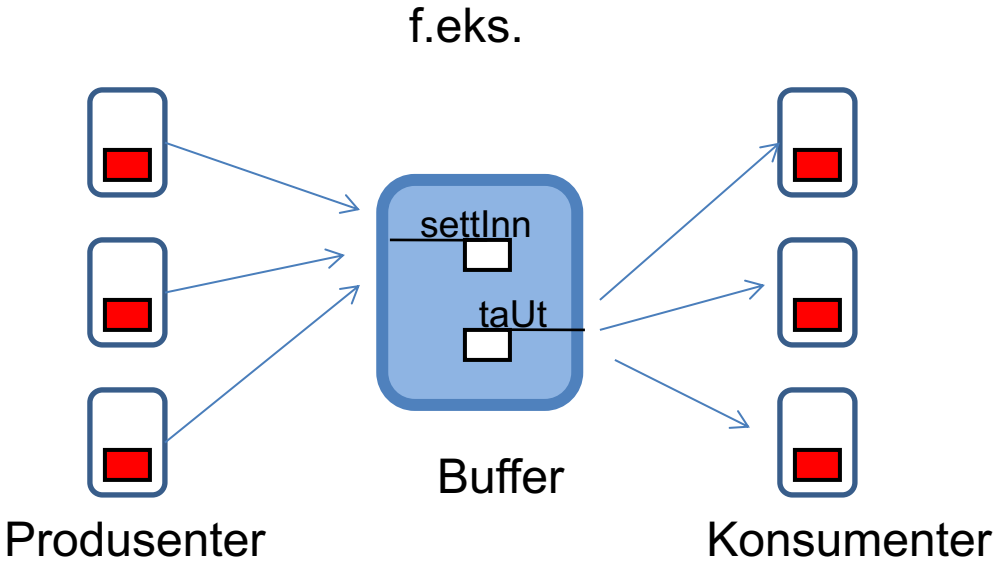
settInn(7), settInn(4), taUt(), settInn(8), settInn(2), settInn(9), taUt()

Hva returnerer nå neste kall på taUt() ?

Hva returnerer neste kall på taUt() om dette var en FIFO-kø ?

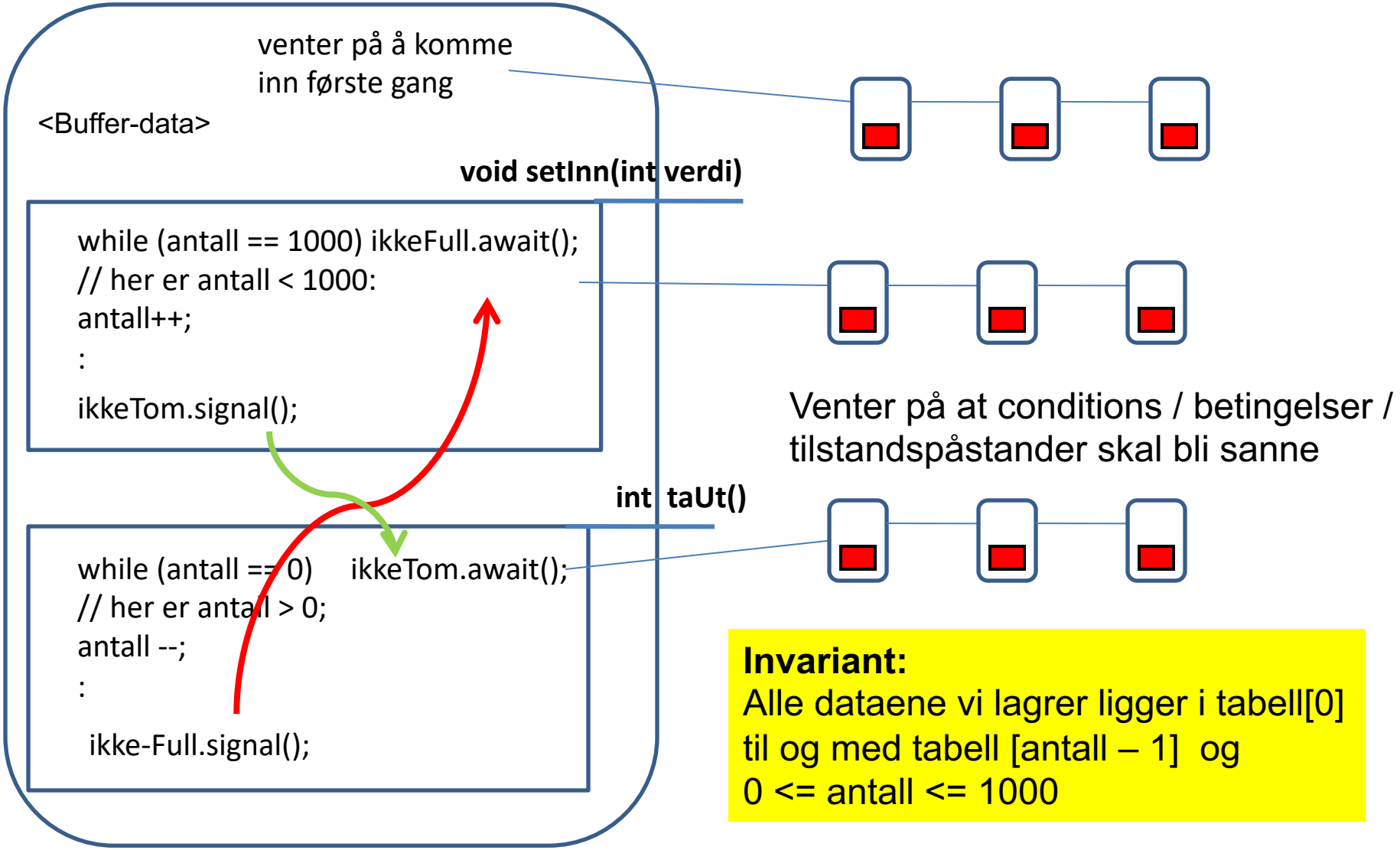
Data i objekter som brukes av flere tråder samtidig

Et felles objekt kalles en **monitor**.
Metodene i en monitor må være *kritiske regioner*



Invarianter og monitorer

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.*



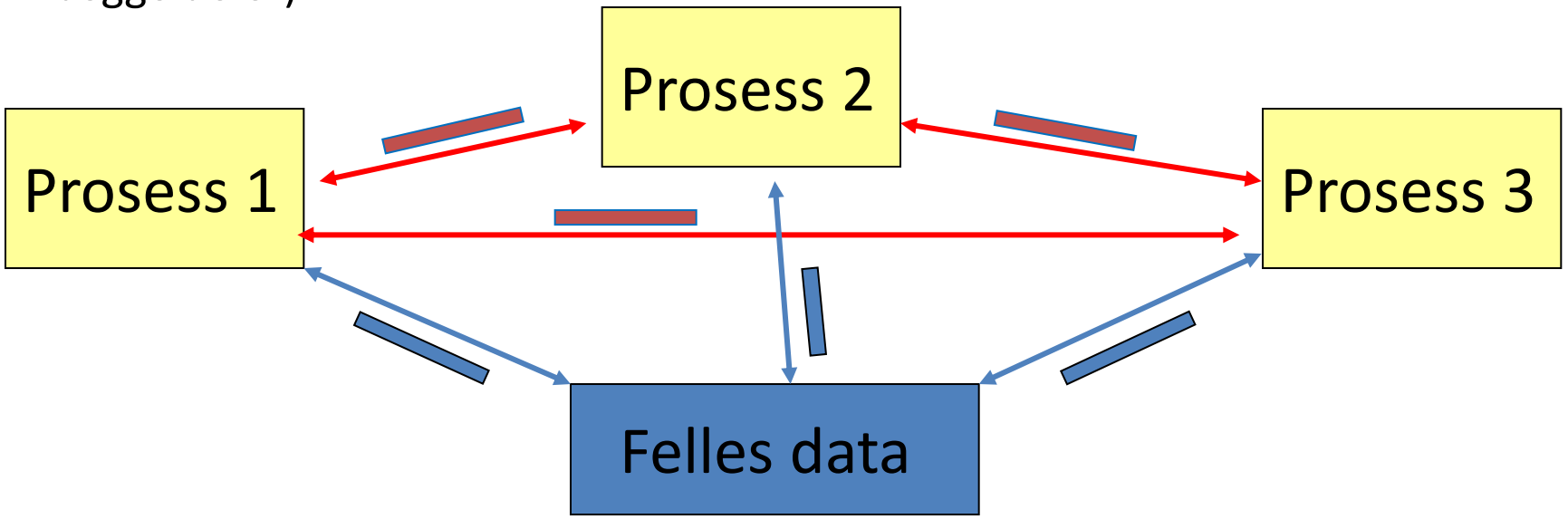
Invariant:
Alle dataene vi lagrer ligger i tabell[0] til og med tabell [antall – 1] og
0 <= antall <= 1000

Prosesskommunikasjon i høynivåspråk

Parallellprogrammering

Parallellprogrammering vil si å løse en oppgave ved hjelp av programmer (eller programbiter) som utføres samtidig.

Samarbeidende prosesser sender meldinger til hverandre (røde piler)* **eller** leser og skriver i felles primærlager (blå piler) (men vanligvis ikke begge deler).



* Mest vanlig, for eksempel MPI (Message Passing Interface)

Kommunikasjon mellom Java-**prosesser:**

RMI: Remote Method Invocation

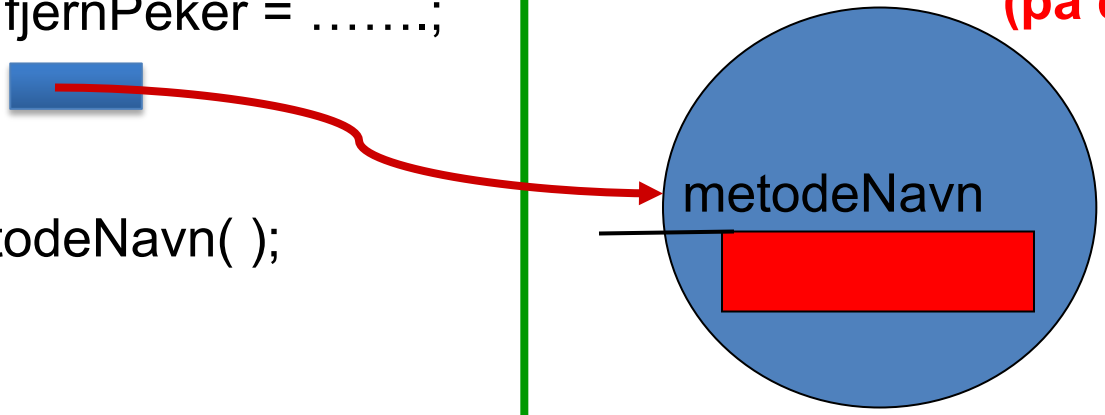
(Norsk: Fjern-metode-kall)

Dette ønsker vi å oppnå:

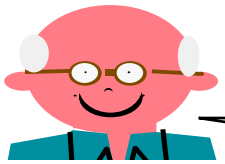
Et Java program
(på en maskin)

```
InterfaceNavn fjernPeker = …….;

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

Et annet Java program
(på en annen maskin)



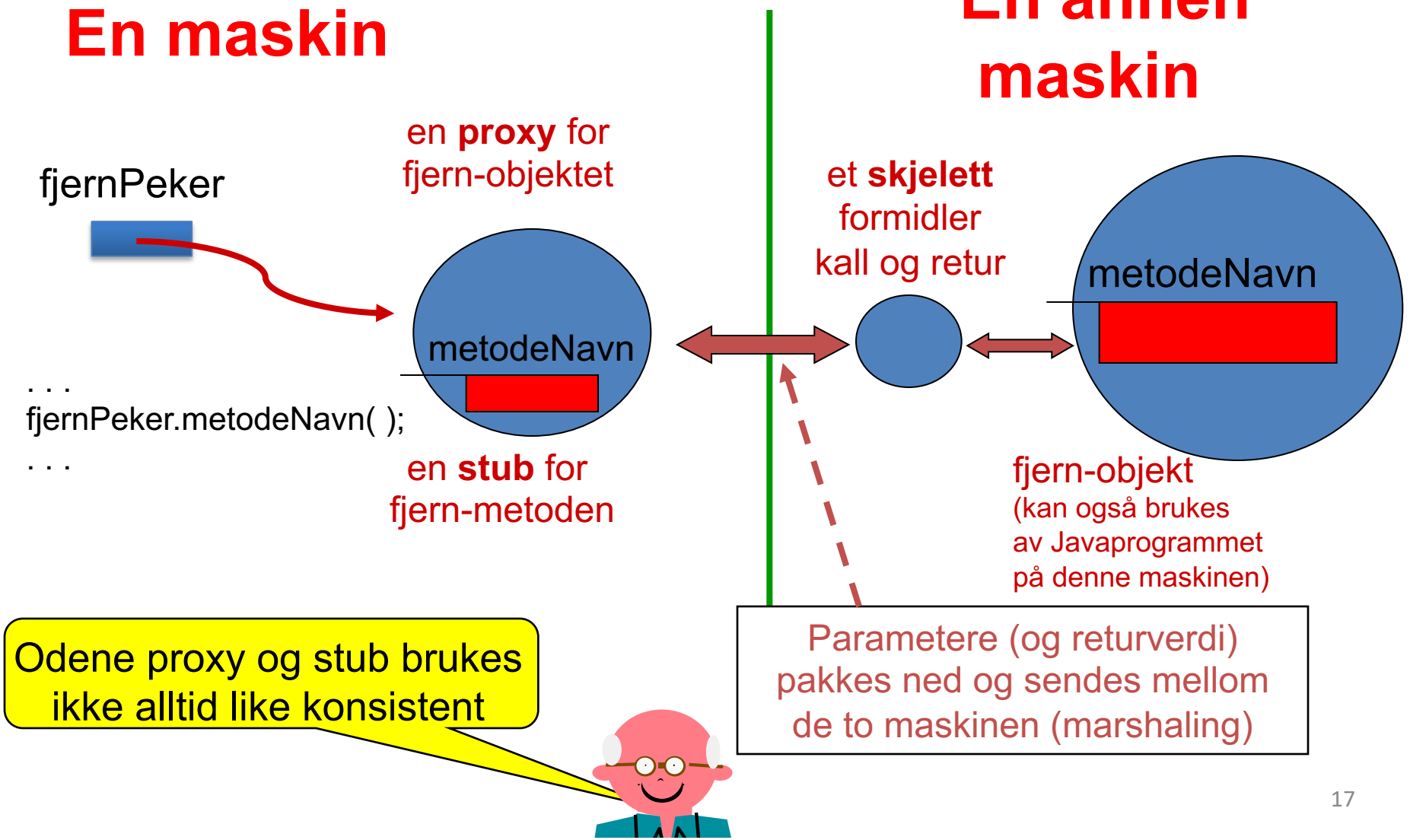
objekt av en klasse som implementerer *InterfaceNavn*



Alternativt navn: RPC: Remote Procedure Call

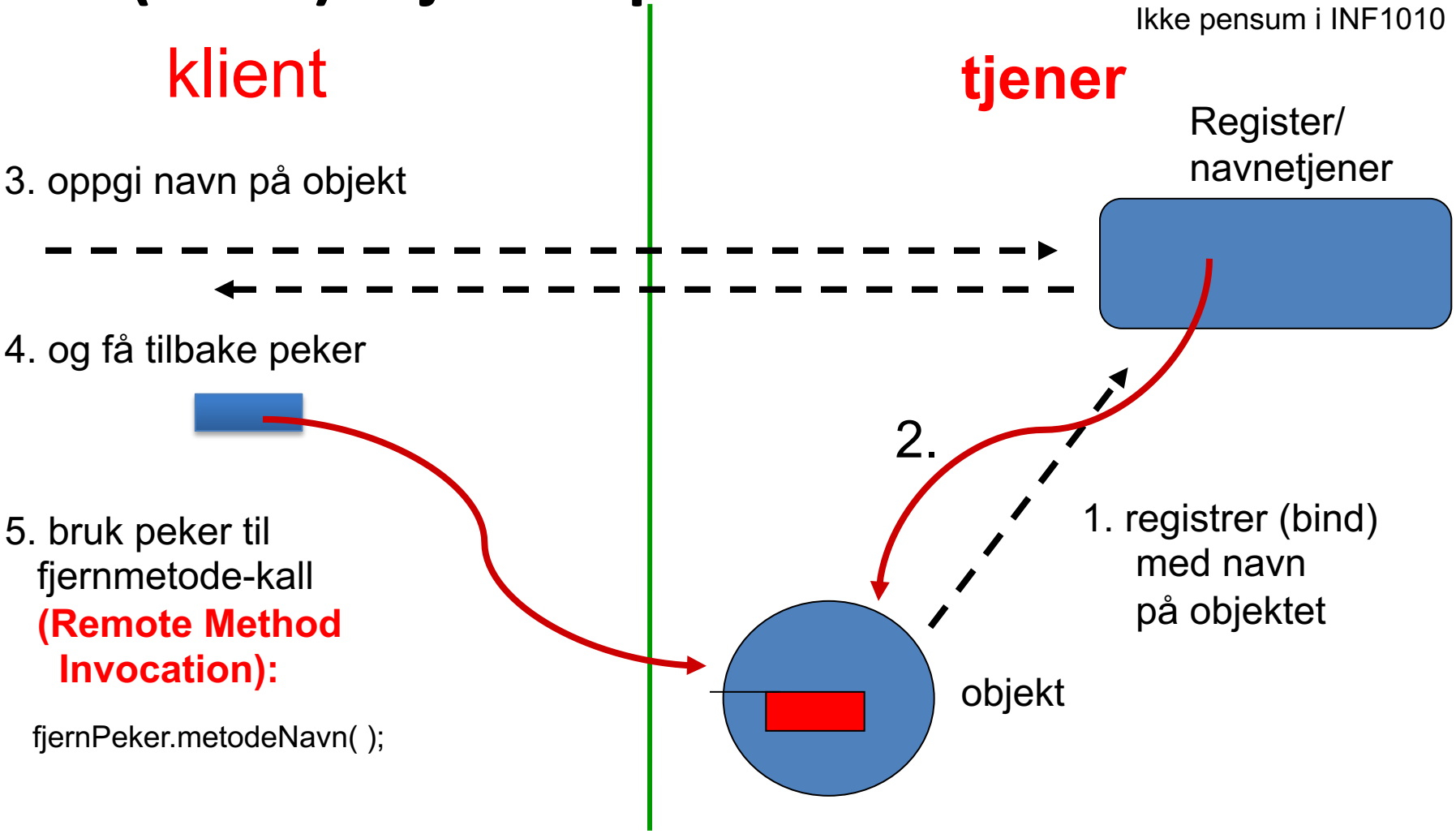
RMI: Implementasjon (bak kulissene)

Ikke pensum i INF1010



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

Ikke pensum i INF1010

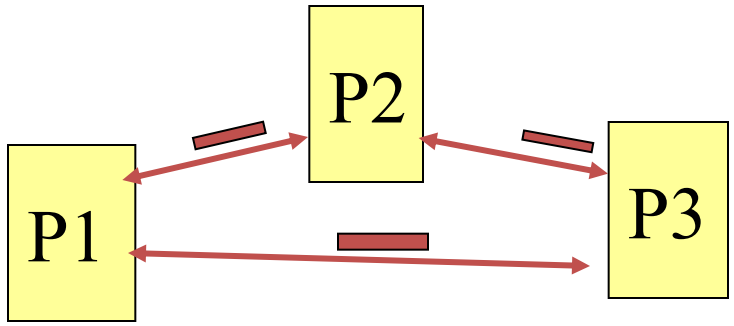


Og det eneste som er kjent på begge maskinene er Interfacet til objektet og navnet

CORBA – TCP/IP

- CORBA (Common Object Request Broker Architecture) er (var) en språkuavhengig måte å definere kommunikasjon mellom 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.

- TCP/IP
- Språk som ikke er objektorienterte:
  Send meldinger / data over forbindelser
  vha. sockets (TCP/IP, mm)
- Mye saktere



I dag har vi lært

- En god programmerer må tenke i invarianter
  - Løkker
  - Data i objekter
- En god programmerer tenker i for-betingelser og bak-betingelser (pre-conditions og post-conditions)
- Venting i monitorer kan sees på som bevaring av invarianter
  - Signalering når invarianten kan bli oppfylt
- Java-prosesser kan kommunisere ved hjelp av RMI
- Mest vanlig i dag MPI og TCP/IP (mye saktere)