Tråder I

agneshb@uio.no

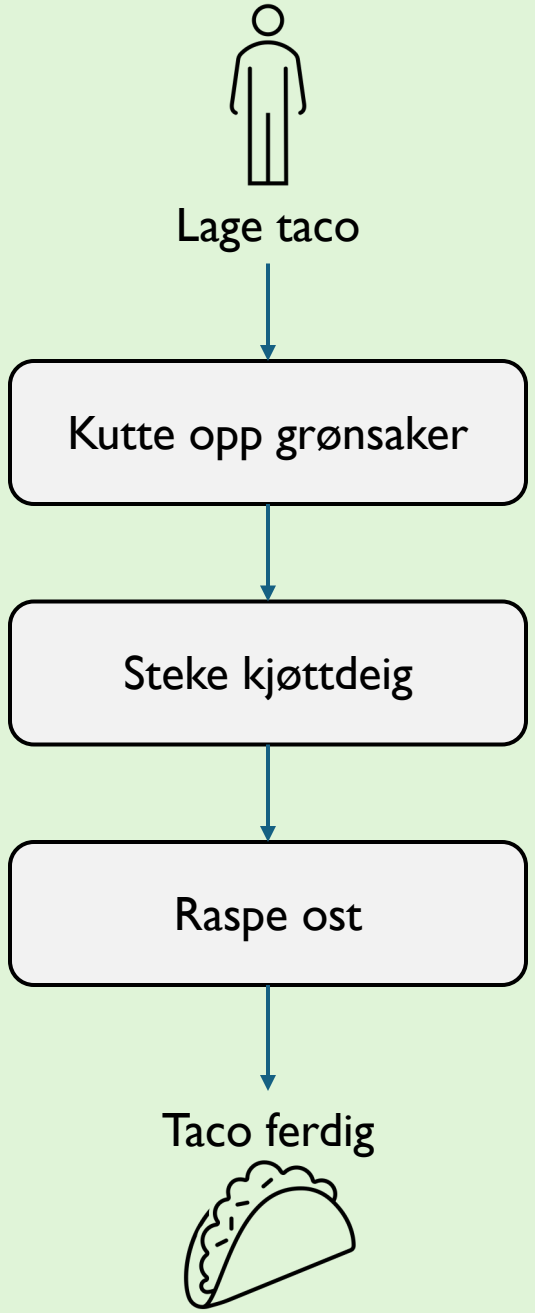
I dag

- Hva er parallellitet og hva er fordelene med det?
- Hva er tråder?
- Hvordan lage tråder
- Tråder som deler data
  - Lock
  - Condition

- Viser eksempler og livekoding

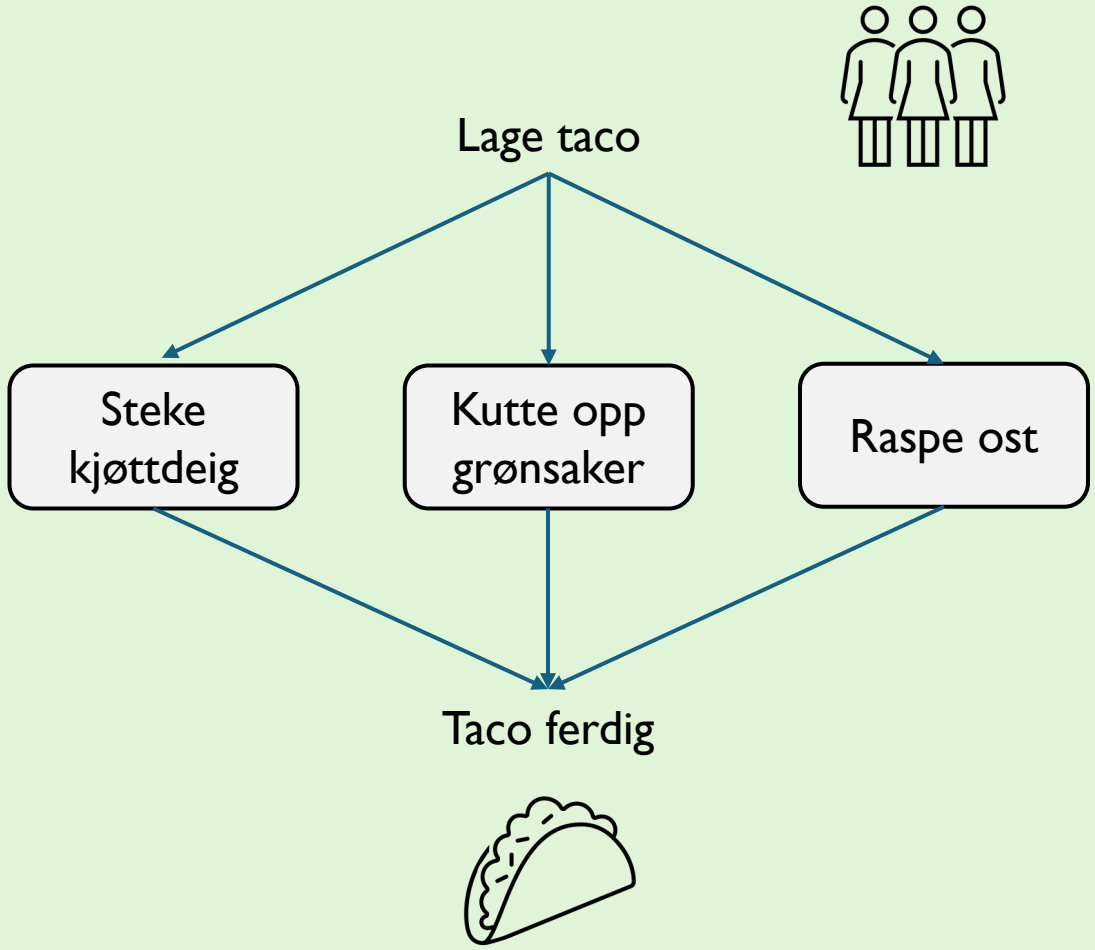
Hittil:

- Hittil har vi skrevet sekvensielle, ikke-parallelle programmer.
- Én sekvens av instruksjoner, utført i én tråd:
  - main-tråden - `public static void main(String[] args).`
- Sekvensielle programmer utfører koden i den rekkefølgen den er spesifisert i main-metoden.



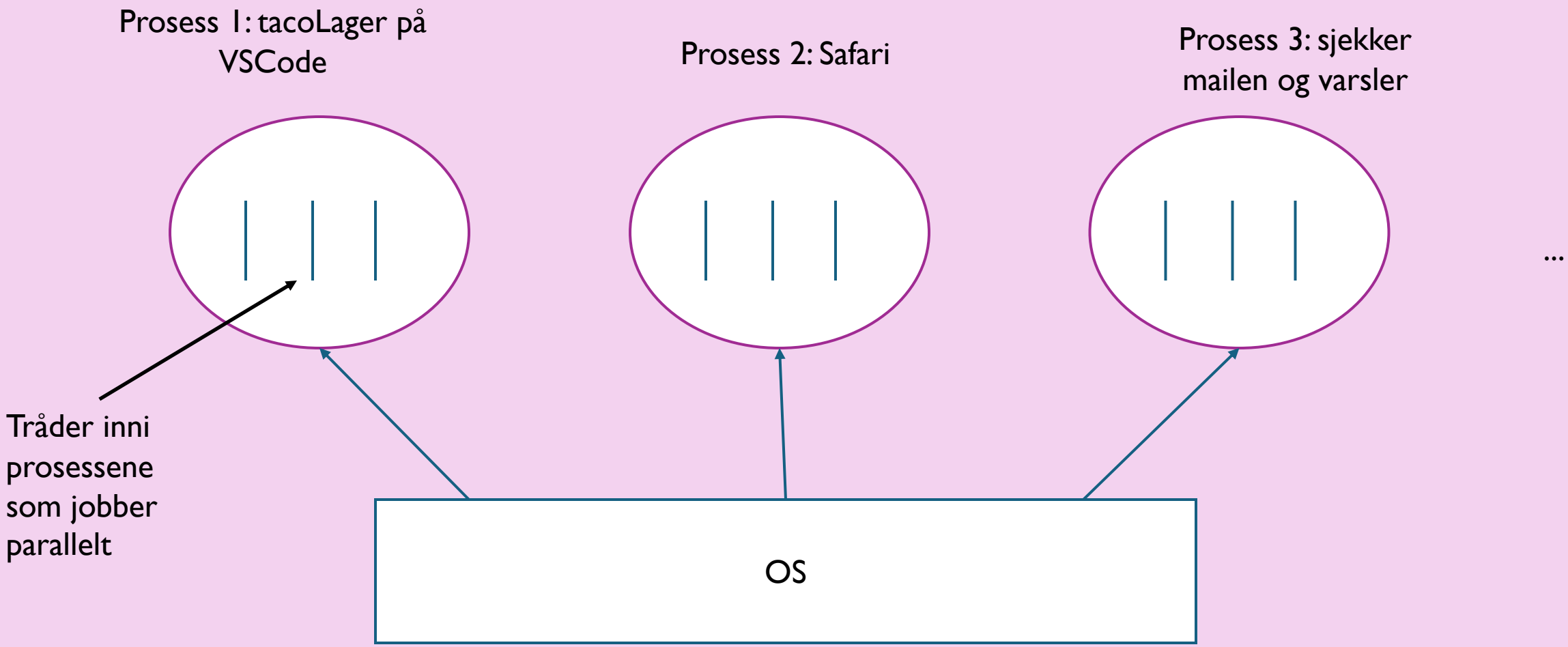
Parallellitet

- Kan lage flere tråder for at to eller flere oppgaver skal kjøre *samtidig*.
- Oppgavene kan dermed utføres parallelt, fordelt på flere tråder.
- *naturlig parallellitet*: når det er naturlig for oss å ta en oppgave, dele den opp i flere deloppgaver og løse deloppgavene samtidig for å komme frem til ønsket resultat.
- Lage taco sammen: naturlig at en kutter grønnsaker, en rasper ost og en steker kjøttdeigen.



Hva er tråder?

- En tråd er en sekvens av instruksjoner i et program.
- Når flere tråder utfører ulike oppgaver (instruksjoner) samtidig, får vi gjort mer på samme tid.
- Tråder som arbeidsdeling på jobben:
  - flere ansatte som utfører ulike oppgaver parallelt på veien mot et felles mål.
  - Tacoeksempelet: grønnsakskutteren, kjøttdeigstekeren og osterasperen kan ses på som tre separate tråder.


Prosess 1: tacoLager på VSCode
Prosess 2: Safari
Prosess 3: sjekker mailen og varsler
...
Tråder inni prosessene som jobber parallelt
OS

Tråder

- Tråder kan:
  - vente, sove og avbrytes (InterruptedException)
  - dele felles data
  - kommunisere
- Trådens oppgave defineres i et objekt av grensesnittet *Runnable* som må implementere `run()`
- I run-metoden implementerer vi oppgaven tråden skal utføre.
- Tråder er objekter av klassen *Thread*.
- For å starte tråden kaller vi på `start()` på hvert trådobjekt.
- Kan kalle `start()` på en tråd fra main-tråden eller en annen tråd

Altså bruker vi to klasser for å lage en tråd:

```
1. <class> implements Runnable
2. Thread
```

Implementerer denne klassen selv

Thread er allerede laget

Fremgangsmåte: lage en tråd

Før vi kan starte tråden(e):

- Oppretter en klasse som implementerer Runnable.
- Implementerer `run()`-metoden inni denne klassen.
  - Det som er i `run()` er trådens oppgave.

```
public class GreatingsThread implements Runnable {

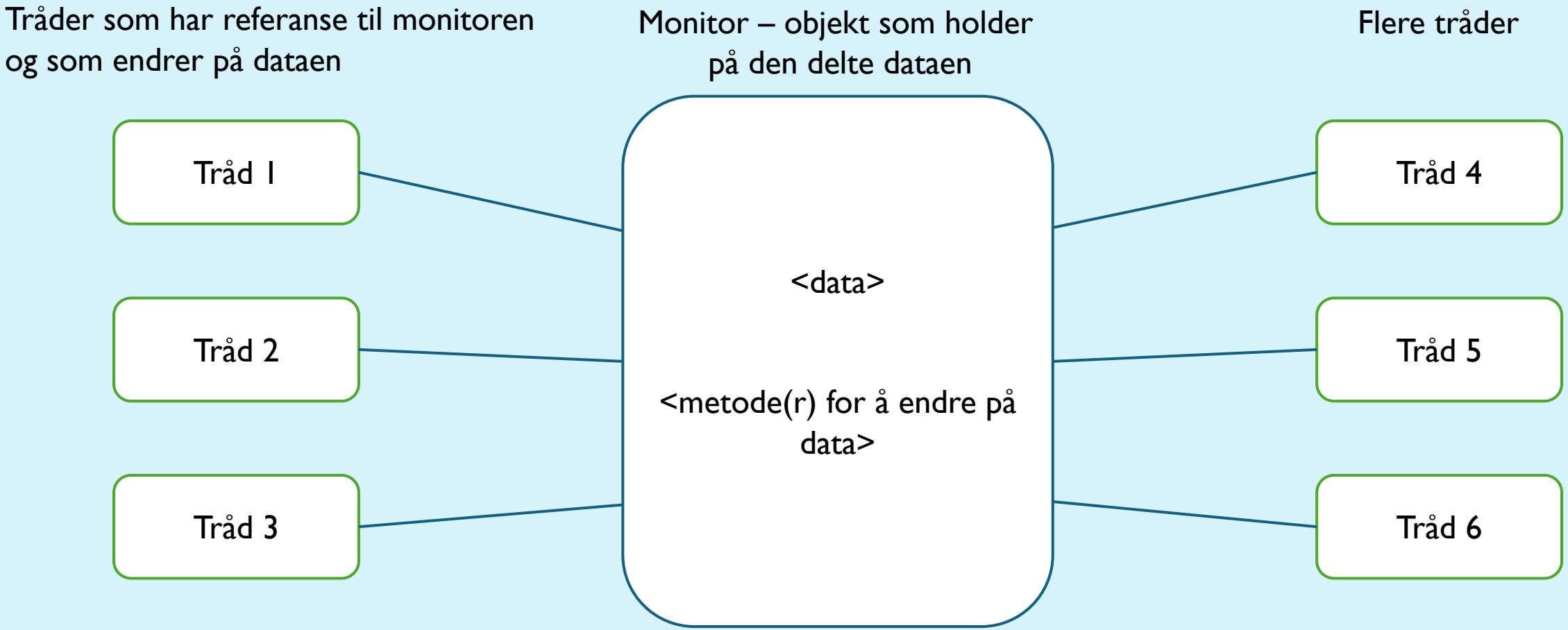
    @Override
    public void run() {
        System.out.println(x:"Hello!");
    }
```

Starter tråden(e):

- Lager et objekt av typen Thread som tar inn et objekt av klassen vi lagde over
  - Altså tar den inn et objekt av en klasse som implementerer Runnable
- Kaller `start()` på tråden vi nettopp lagde

```
public static void main(String[] args) {
    Thread thread = new Thread(new GreatingsThread());
    thread.start();
}
```

Tråder som deler data

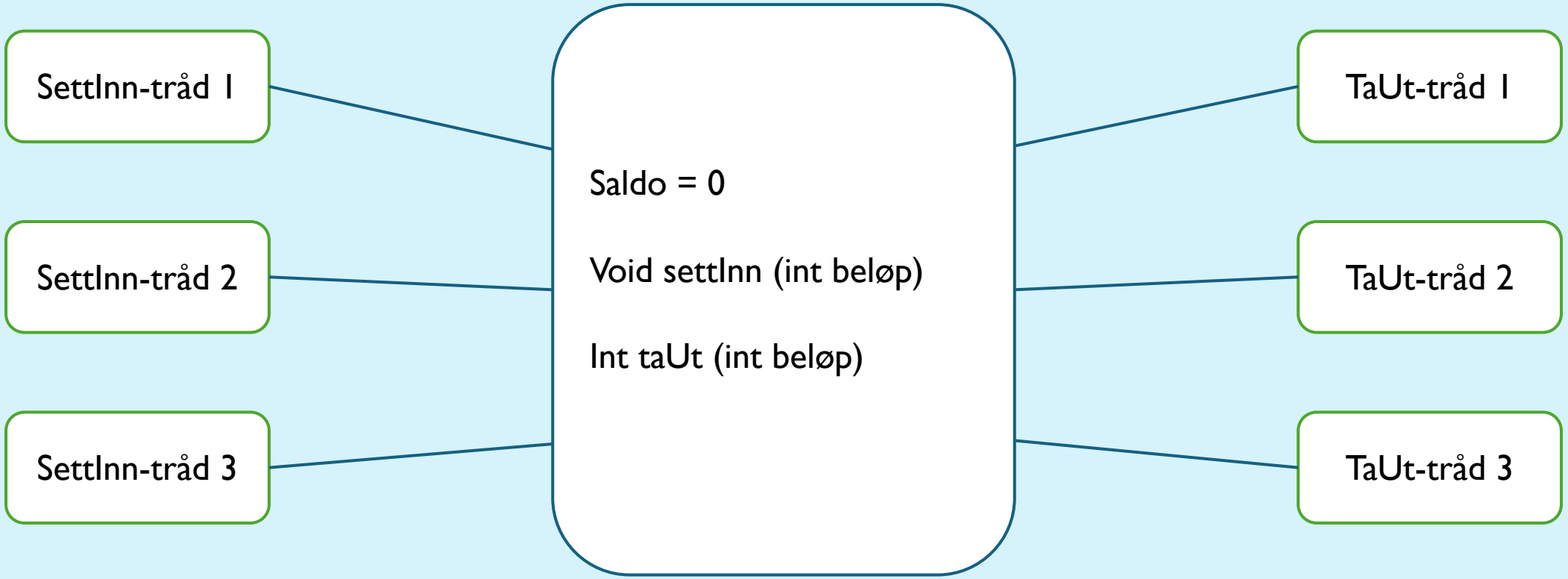


Typisk: produsent og konsument

Produsent-tråder som setter inn penger på kontoen (monitoren)

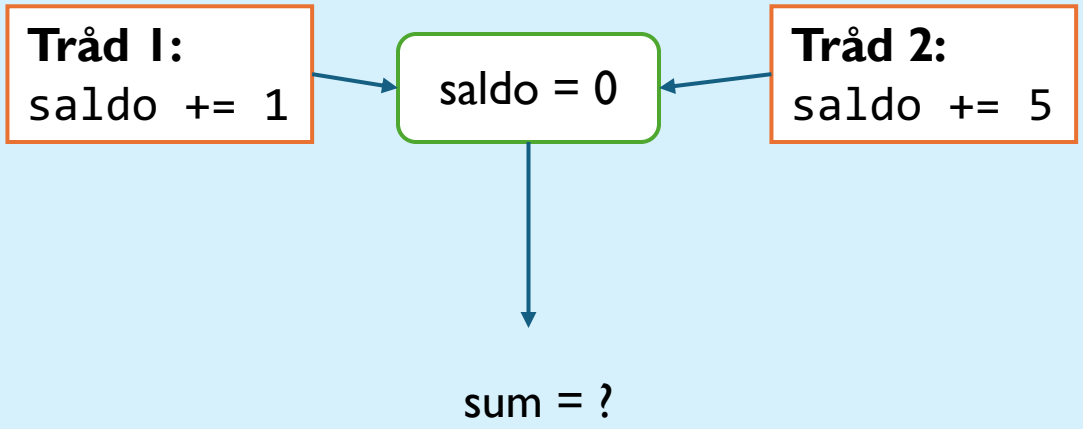
Monitor – bankkonto som inneholder en saldo og metode for å sette inn og ta ut penger

Konsument-tråder som tar ut penger fra kontoen



Tråder som deler data

- To eller flere tråder kan dele data
- Hvis to eller flere tråder kan gjøre endringer på samme data samtidig kan vi få uheldige (og uforutsigbare) resultater.



Løsning: låser 

- Kun én tråd kan aksessere delt data om gangen.
- Kritisk region der kun en tråd kan gjøre endringer om gangen.
- Andre tråder må vente i kø.

Fremgangsmåte låser i monitoren

- Lager en lås
- Kaller låsen sin `lock()`-metode
- Gjør endringen
- Kaller låsen sin `unlock()`-metode

- NB! Bruker try-finaly for å sikre at låsen blir ulåst uansett om noe går galt

```
Lock laas = new ReentrantLock();
```

```
public void leggTil(int tall){
    laas.lock(); //låser
    try{
        //gjør endringer
        verdi += tall;
    }finally{
        laas.unlock(); //låser opp
    }
```

Passiv venting

- Ofte venter tråder på at en eller flere oppgaver skal bli utført.
- Med passiv venting venter tråden til den får beskjed/signal.
  - I motsetning til aktiv venting der den sjekker om det er klart – fører blokkering av andre tråder og unødvendig bruk av CPUen
- Lager en Condition knyttet til låsen
  - `lock.await()` – tråden må vente
  - `lock.signalAll()` – vekker ventende tråder

Fremgangsmåte

- Lager en condition knyttet til låsen.
- Kaller conditionen sin `await()`-metode der vi ønsker at tråden skal vente.
- Kaller conditionen sin `signalAll()`-metode der vi ønsker at tråden skal forsøke å gjøre oppgaven sin igjen.

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

```
while (saldo < belop){
    ikkeTomt.await();
}
```

i taUt-metoden

```
saldo += belop;
//det er lagt til penger på konto, så sier i
//fra til tråder som venter på å kunne ta ut
ikkeTomt.signalAll();
```

I settInn-metoden

Jobbe med oppgaver<3

Ukesoppgaver ligger på semestersiden → grupper → gruppe 3 → uke 11

Notat om tråder