TNM094 – Medietekniskt kandidatprojekt

Multi-trådning

Idag

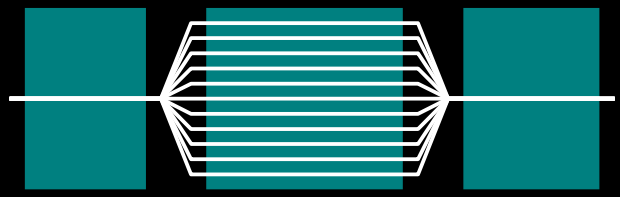
- Processorer blir inte snabbare
  - samma klockfrekvens
  - marginellt förbättrad prestanda
- Processorer får fler kärnor
  - PC: 2-8 kärnor
  - Laptop: 2-8 kärnor
  - Telefon: 2-8 kärnor
  - Snart: 128 kärnor

Vad betyder det här?

- Mer responsivt användargränssnitt
  - en tråd för användargränssnittet
  - en tråd för underliggande processer
- Bakgrundsprocesser
  - hämtar nya data från Internet
  - kontrollerar filer om de ändrats
- Parallelliserad processning
  - dela upp bildbehandling per 16x16 pixlar

Vad betyder det här?

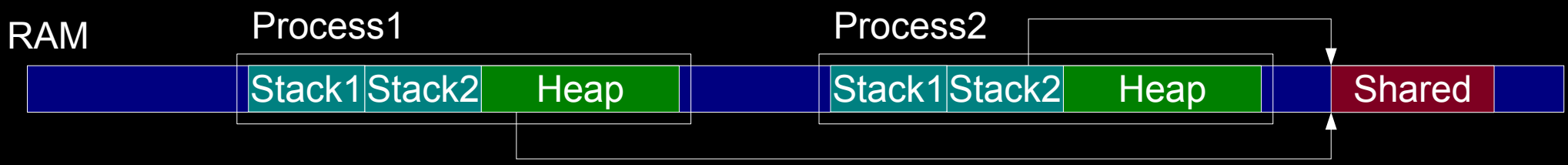
- HPC – High Performance Computing
  - 10-1000 samtida trådar
  - Single Instructions on Multiple Data (SIMD)
  - OpenMP, MPI, OpenCL, CUDA, GPGPU, clusters



- Multi-trådade system
  - 2-5 samtida trådar
  - Olika uppgift per tråd
  - Manuell synkronisering

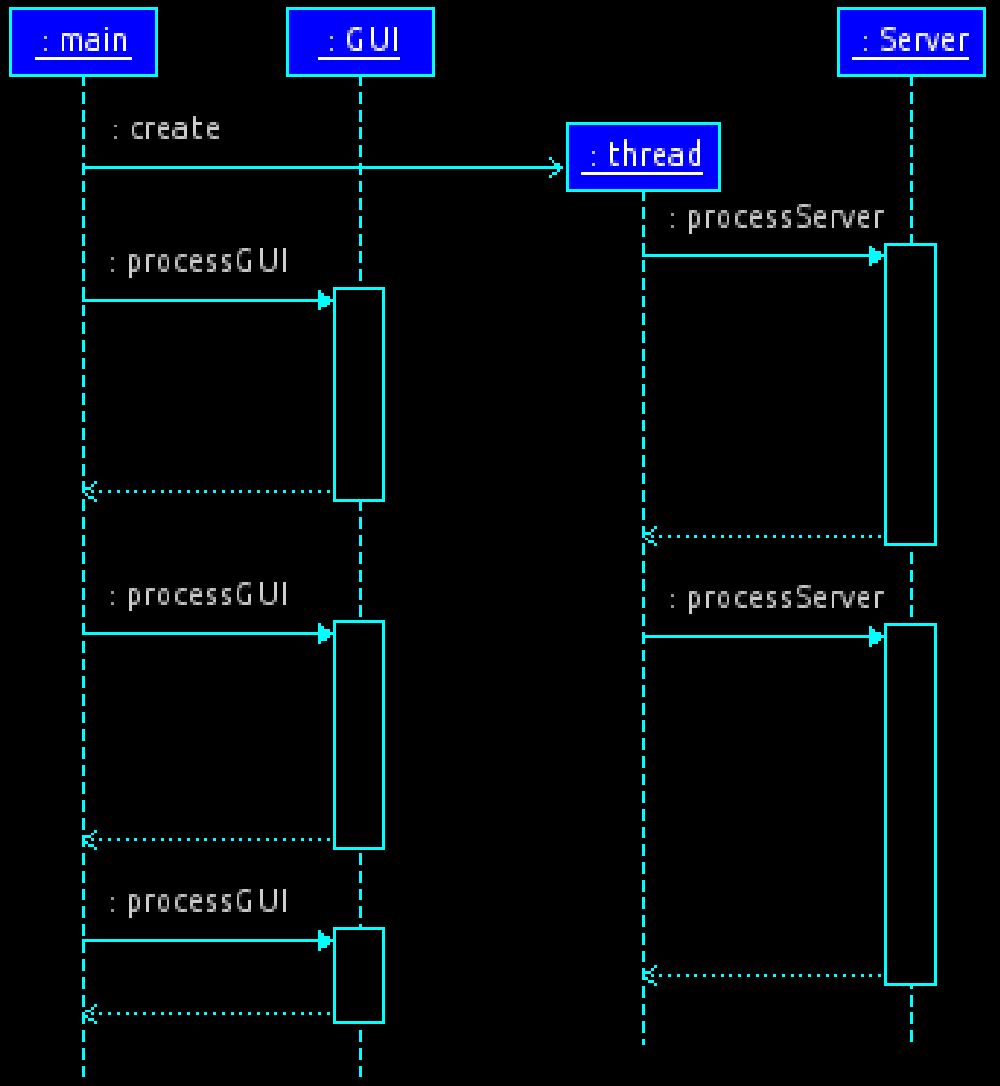
Parallell körning

- Process
  - har eget program-minne
  - kommunicerar över nätverk eller delat minne
- Tråd (thread)
  - flera trådar kör parallellt inom samma process
  - individuell anropsstack men delar heap
- Minnesaccess
  - Stack
  - Heap
  - Delat minne



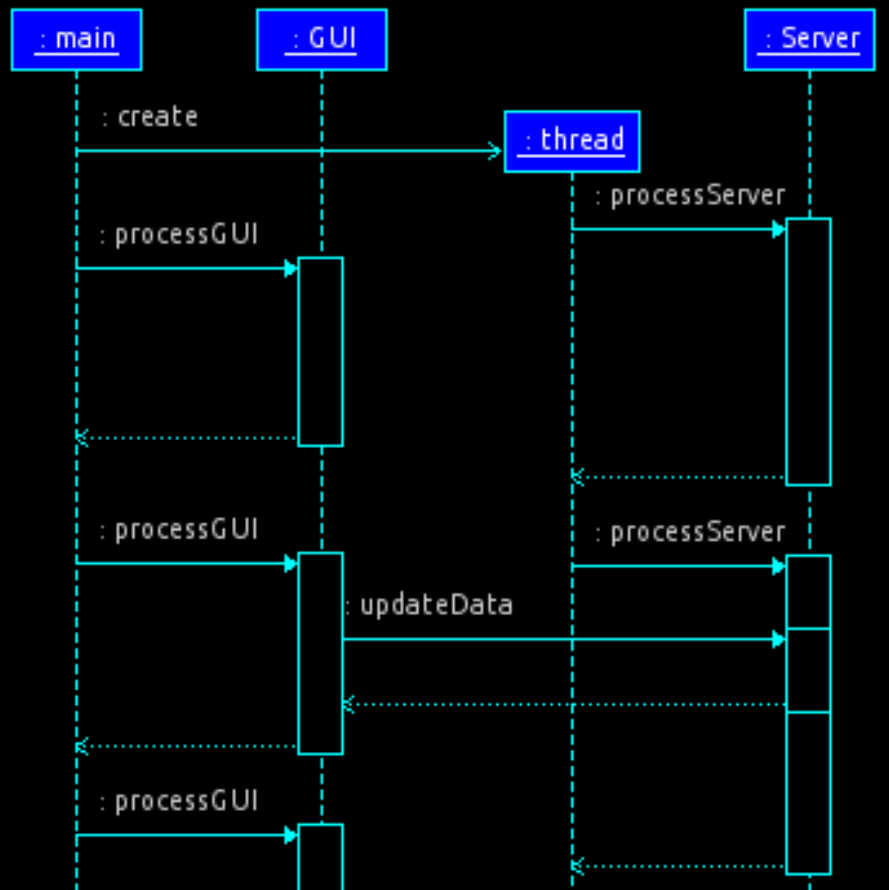
Multi-trådning

```
bool alive;

void server_function(void *ptr){
  while(alive){
    processServer();
  }
}

int main(){
  alive = true;
  Thread *thread
   = new Thread(server_function,NULL);
  while(alive){
    processGUI();
  }
}
```



Problem

- Access av minne
  - två trådar skriver olika värde till samma variabel
  - en tråd läser samtidigt som en annan skriver
- Kompletta tillstånd
  - beroende variabler
    ändras av olika trådar
- Lösning
  - atomiska operationer
  - atomiska transaktioner



Exempel

```
bool alive;
int call_count = 0;

void call_counter(){
  call_count += 1;
}

void main2(void *ptr){
  while(alive){
    call_counter();
  }
}

int main(){
  alive = true;
  Thread *thread = new Thread(main2,NULL);
  while(alive){
    call_counter();
  }
}
```

Exempel

Tråd 1

```
call_count → R0 (0)
R0 + 1 → R1     (1)


R1 → call_count (1)
```

Tråd 2

```
call_count → R2 (0)
R2 + 1 → R3     (1)
R3 → call_count (1)
```

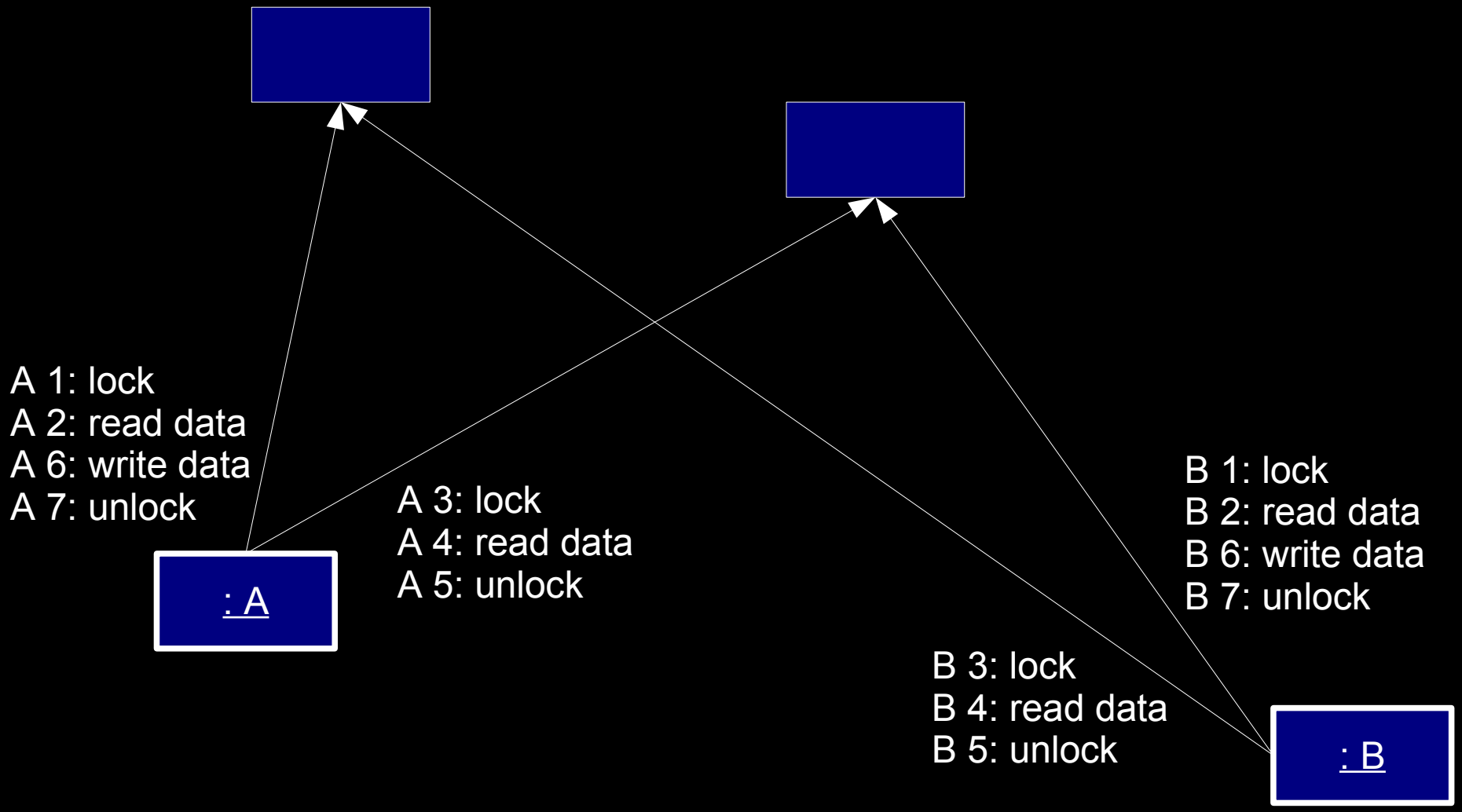
Programmeringsprinciper

- Atomiska operationer
  - för att undvika samtida hantering av samma data
    - mutex lock (**mut**ually **ex**clusive)
    - read-write lock (många kan läsa men bara en kan skriva)
- Konstrukter på högre nivå
    - barrier – alla trådar väntar på varandra
    - parallel region – flera trådar inom en begränsad bit kod
    - serial region – en enda tråd inom en begränsad bit kod
- Designmönster – bra sätt att hantera trådning
    - scoped lock
    - thread-safe interface
    - active objects

Mutex-exempel

```
bool alive;
int call_count = 0;
mutex call_counter_lock;

void call_counter(){
  call_counter_lock.lock();
  call_count += 1;
  call_counter_lock.unlock();
}

void main2(void *ptr){
  while(alive){
    call_counter();
  }
}

int main(){
  alive = true;
  mutex_init(call_counter_lock);
  Thread *thread = new Thread(main2,NULL);
  while(alive){
    call_counter();
  }
}
```

Dead-lock-exempel



Execution:
A1, A2, B1, A3(wait), B2, B3(wait)

Trådsäkerhet

- Klassificera funktioner, klasser eller andra enheter
  - Unsafe
    - trådade, samtida anrop leder till oförutsägbara resultat
    - gäller de flesta metoder som ändrar i data
  - Guarded
    - trådar kommer att vänta på sin tur
    - kan leda till deadlock
  - Thread safe
    - trådade, samtida anrop körs parallellt
    - använder medlemsvariabler som är `const`
    - processar data på anropsstacken eller duplicerar data
    - använder mer avancerade design-mönster