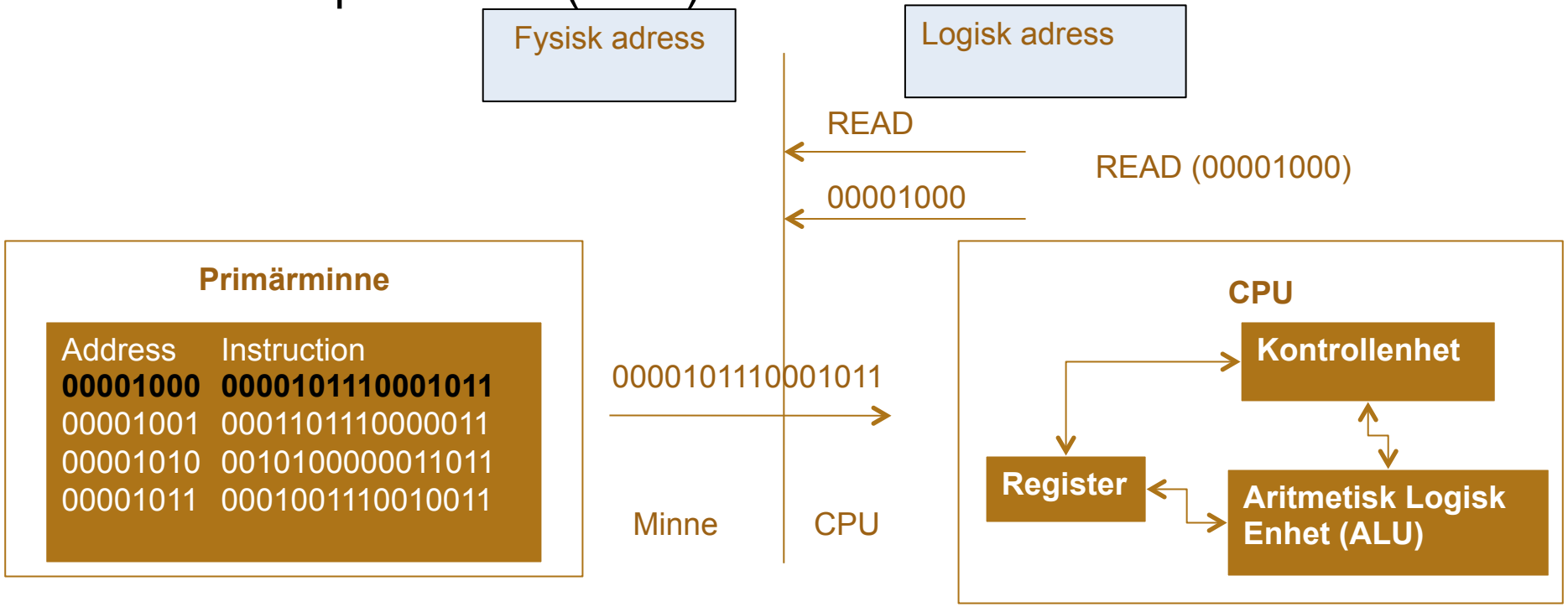
LUNDS UNIVERSITET

# Datorteknik

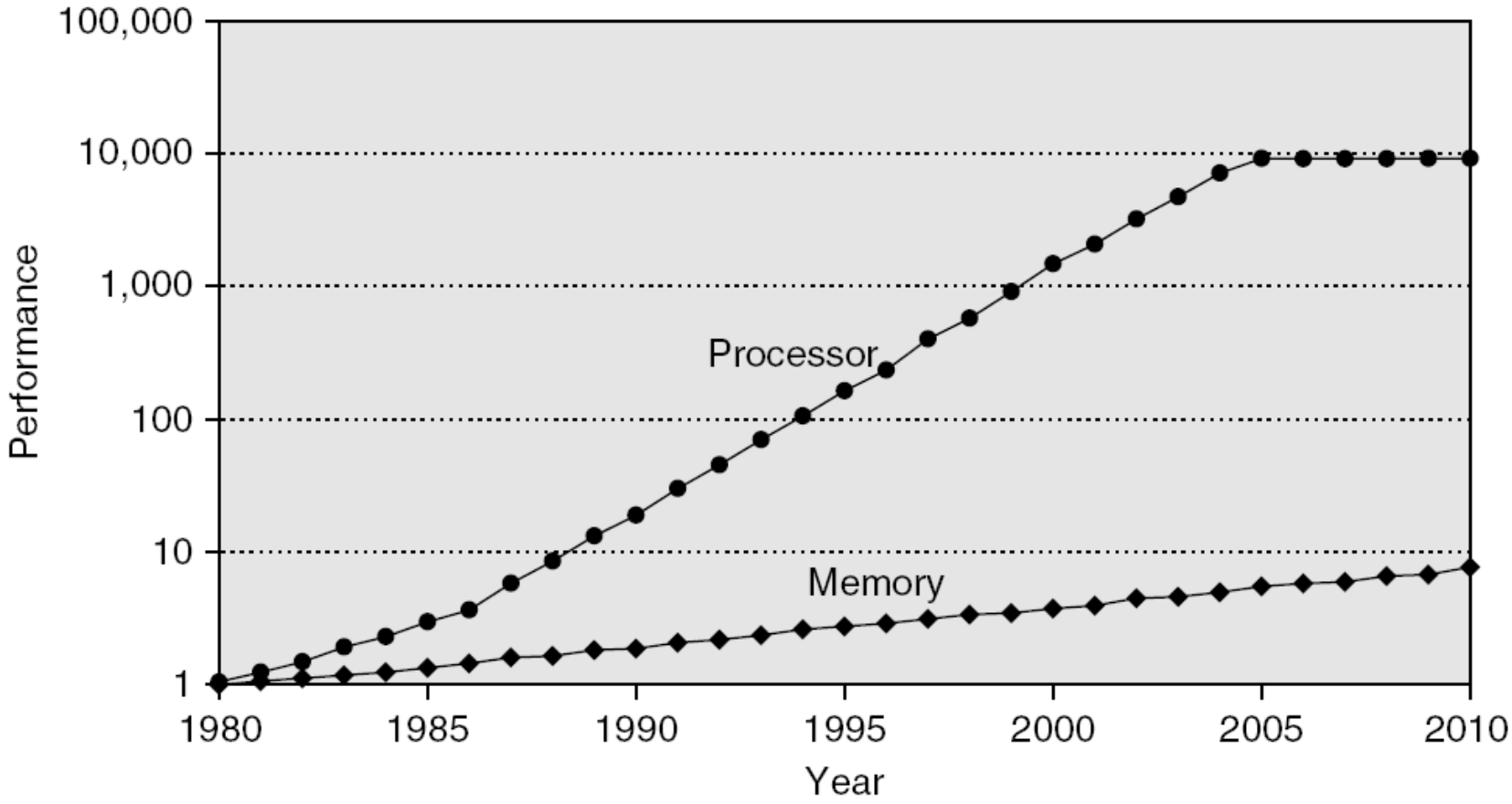
ERIK LARSSON

# Minnet från processorns sida

- Processorn ger kommandon/instruktioner med en adress och förväntar sig data.
  - Exempel: READ(ADR) -> DATA



# Minne-processor hastighet



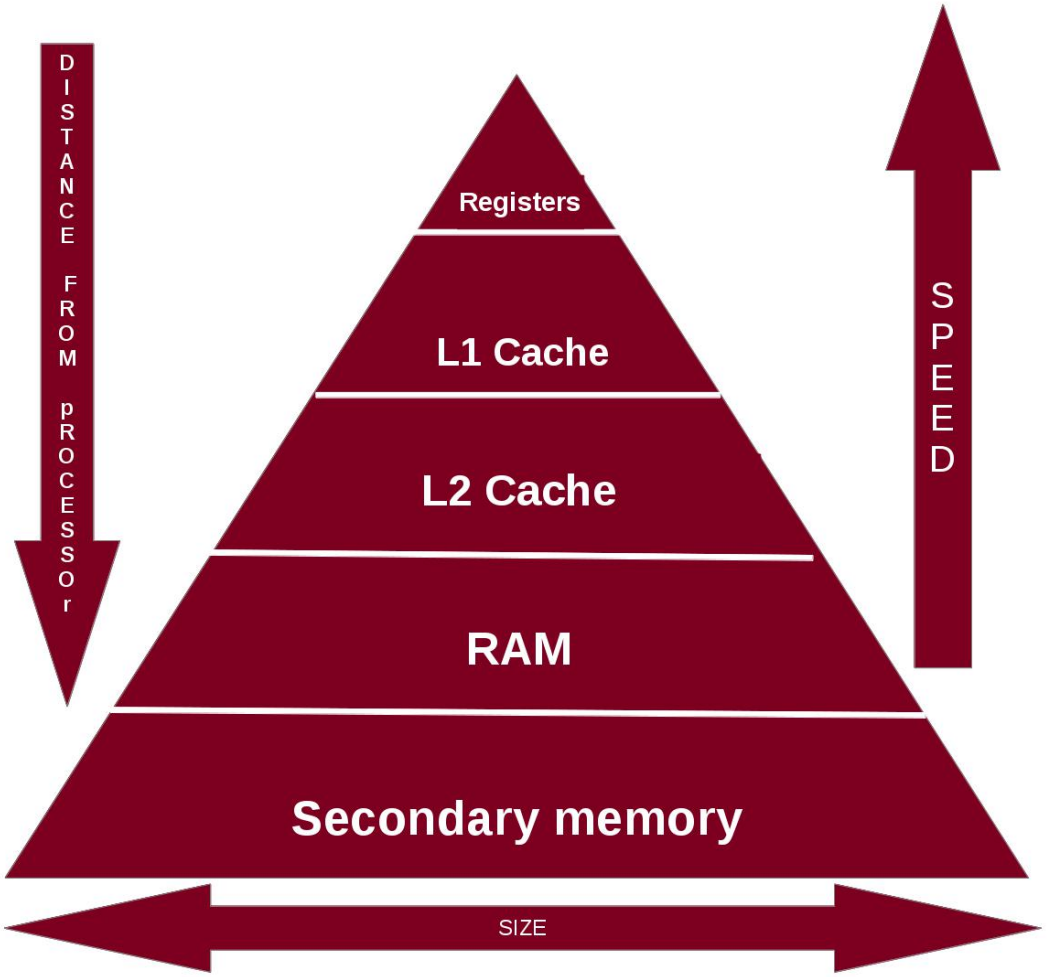
# Design av minnesystem

- Vad vill vi ha?
  - Ett minne som får plats med stora program och som fungerar i hastighet som processorn
    - » Fetch – execute (MHz/GHz/Multi-core race)


- Grundproblem:
  - Processorer arbetar i hög hastighet och behöver stora minnen
  - Minnen är mycket långsammare än processorer
- Fakta:
  - Större minnen är långsammare än mindre minnen
  - Snabbare minnen kostar med per bit

LUNDS UNIVERSITET

# Minneshierarki



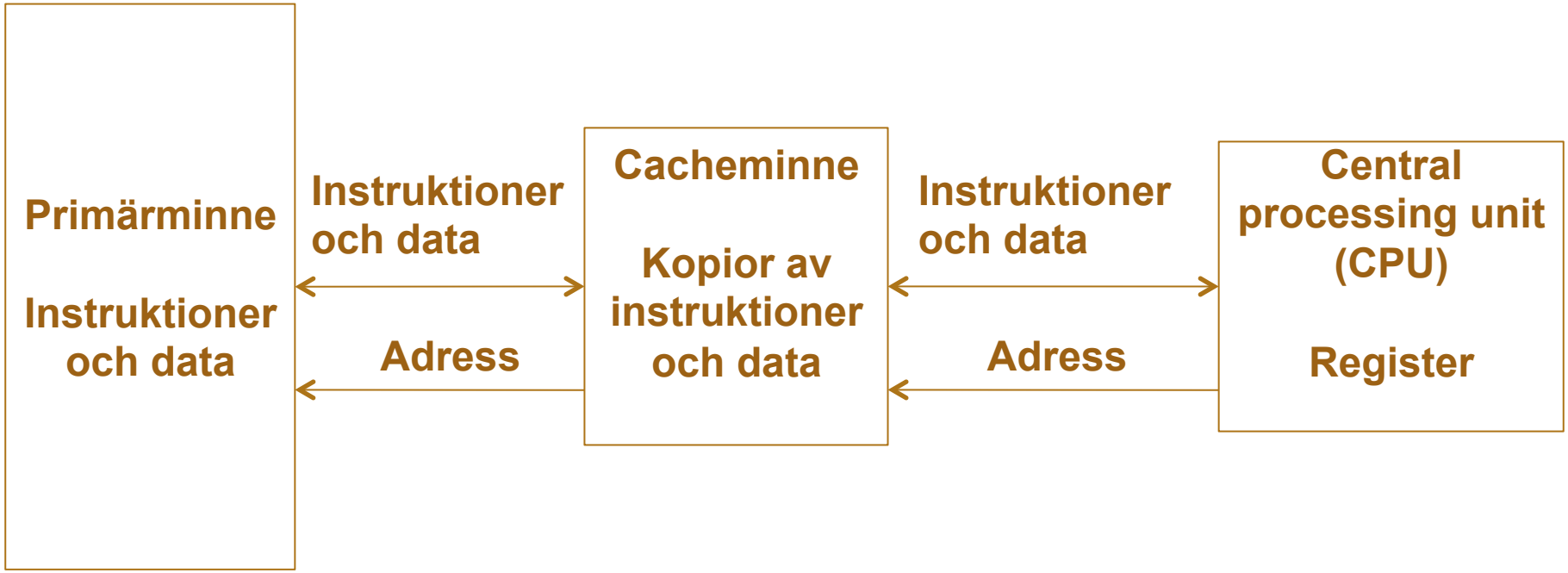

LUNDS UNIVERSITET

# Minneshierarki

- Processor registers:
  - 8-32 registers (32 bitar -> 32-128 bytes)
  - accesstid: få ns, 0-1 klockcykler
- On-chip cache memory (L1):
  - 32 till 128 Kbytes
  - accesstid = ~10 ns, 3 klockcykler
- Off-chip cache memory (L2):
  - 128 Kbytes till  12 Mbytes
  - accesstid = 10-tal ns, 10 klockcykler
- Main memory:
  - 256 Mbytes till 4Gbytes
  - accesstid = ~100 ns, 100 klockcykler
- Hard disk:
  - 1Gbyte tid 1Tbyte
  - accesstid = 10-tal milliseconds, 10 000 000 klockcykler

LUNDS UNIVERSITET

# Cacheminne




Accesstid: 100ns

Accesstid: 10ns

LUNDS UNIVERSITET

# Cache – exempel 1

- Program: Assemblyinstruktioner

```
x=x+1;     Instruktion1: x=x+1;
y=x+5;     Instruktion2: y=x+5;
z=y+x;     Instruktion3: z=y+x;
```

- Om man inte har cacheminne:
  - Accesstid för att hämta en instruktion=100ns
    - » Tid för att hämta instruktioner: 3*100=300ns
- Om man har cacheminne:
  - Accesstid för att hämta en instruktion=100+10=110ns
    - » Tid för hämta instruktioner: 3*110=330ns

LUNDS UNIVERSITET

# Cache – exempel 2

- Antag:
  - 1 maskininstruktion per rad
  - 100 ns för minnesaccess till primärminnet
  - 10 ns för minnesaccess till cacheminnet
- Programmet och dess maskininstruktioner.

Exempel program:

```
while (x<1000){
    x=x+1;
    printf("x=%i",x);
while (y<500){
    y=y+1;
    printf("y=%i",y);
}
```

Assembly

```
Instruktion1: while1000
Instruktion2: x=x+1
Instruktion3: print x}
Instruktion4: while500
Instruktion5: y=y+1
Instruktion6: print y
```

LUNDS UNIVERSITET

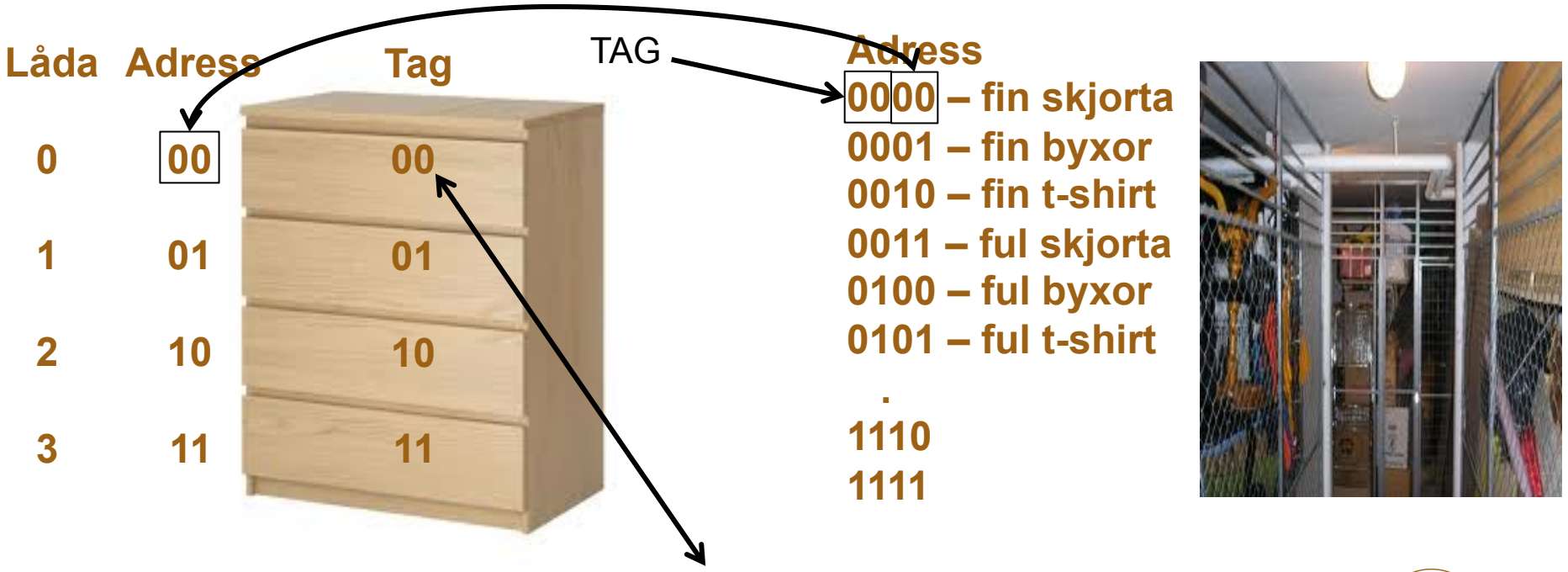
# Cacheminne

- Minnesreferenser tenderar att gruppera sig under exekvering
  - både instruktioner (t ex loopar) och data (datastrukturer)
- Lokalitet av referenser (locality of references):
  - Temporal lokalitet – lokalitet i tid –
    - » om en instruktion/data blivit refererat nu, så är sannolikheten stor att samma referens görs inom kort
  - Rumslokalitet –
    - » om instruktion/data blivit refererat nu, så är sannolikheten stor att instruktioner/data vid adresser i närheten kommer användas inom kort

LUNDS UNIVERSITET

# Cacheminne (Direktmappning)

Hamnar i denna låda

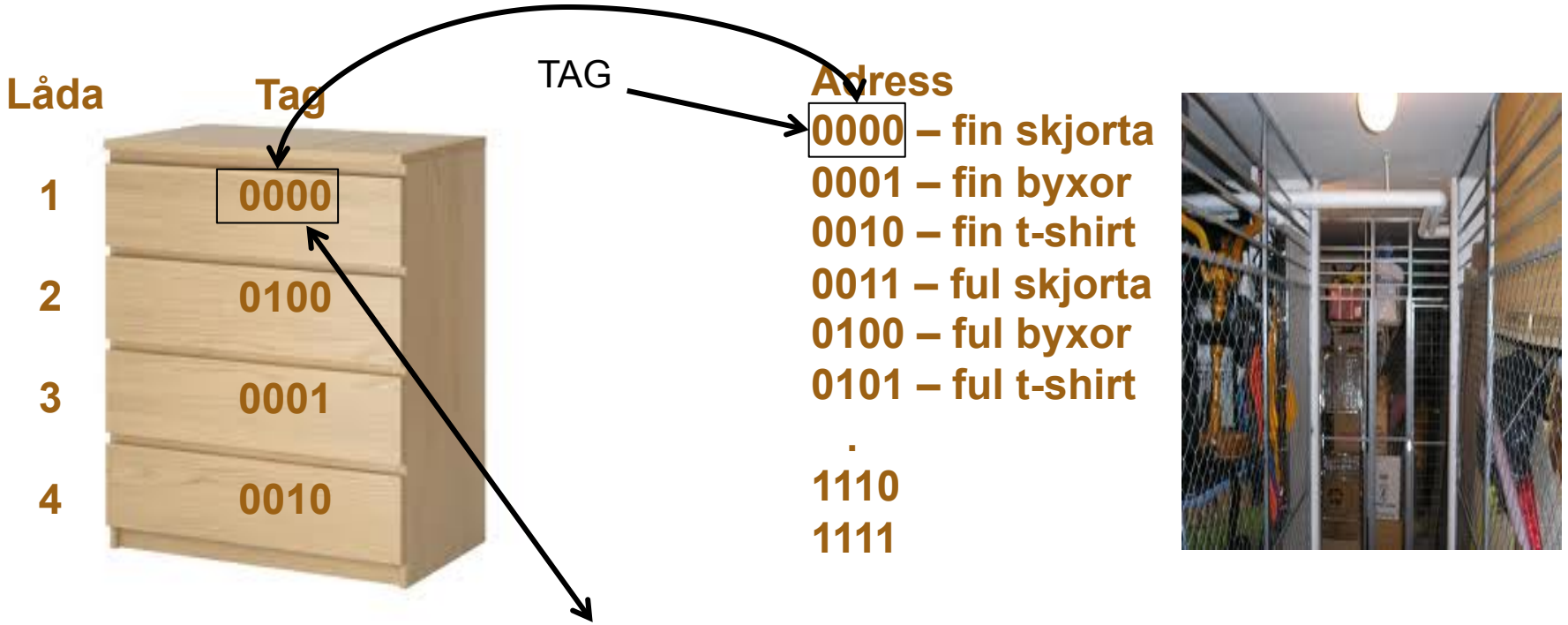



I låda 00 med TAG=00 ligger där en fin skjorta eller  ful byxor?

LUNDS UNIVERSITET

# Cacheminne (Associativmappning)

Kan hamna i vilken låda som helst

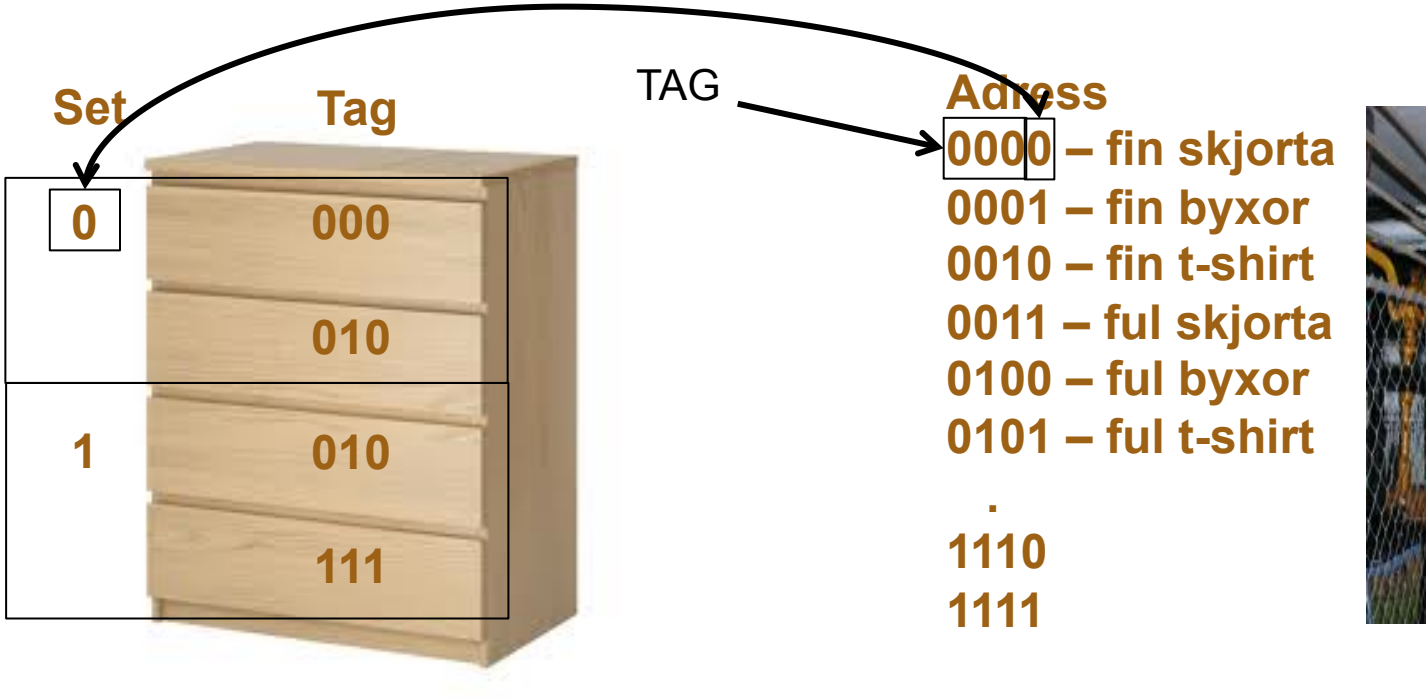



I låda 1 med TAG=0000 ligger där en fin skjorta eller  ful byxor?

LUNDS UNIVERSITET

# Cacheminne (2-vägs set associative)

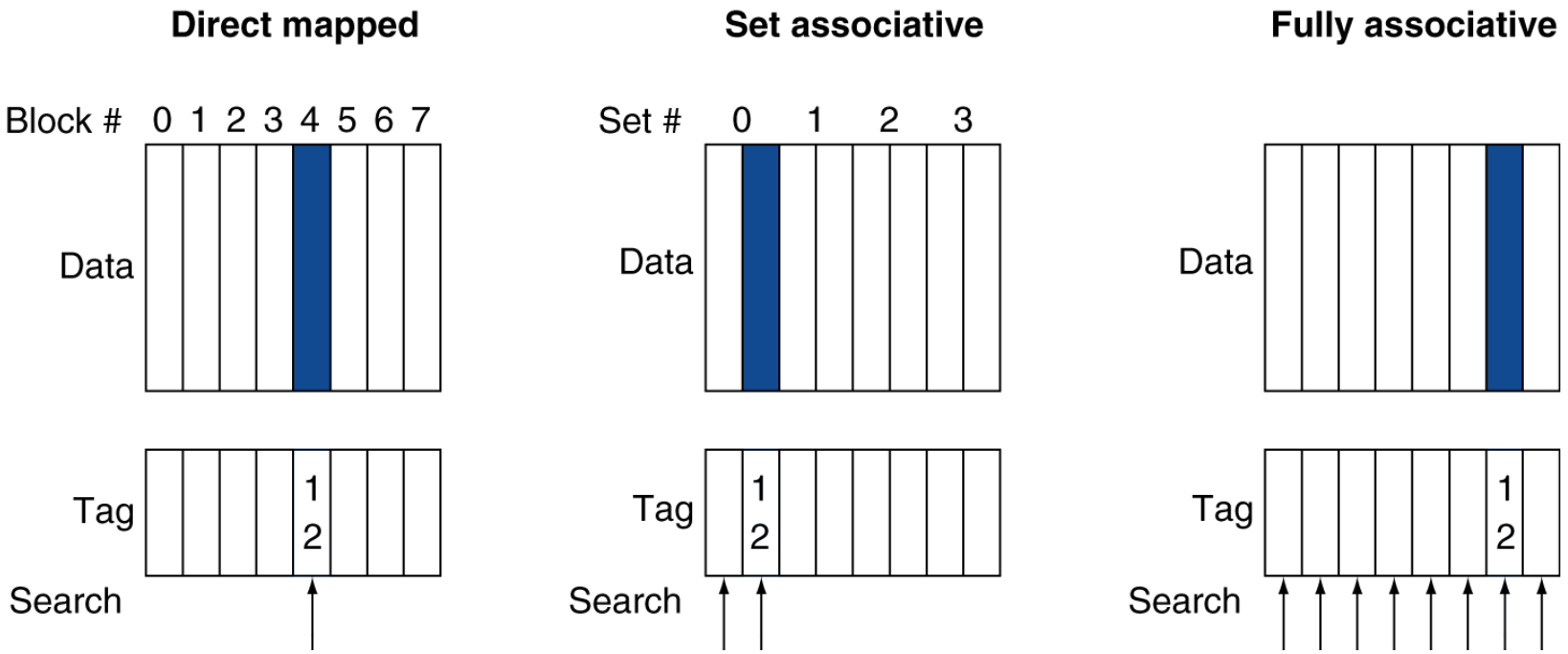
Hamnar i denna låda




I set 0 finns där en fin skjorta eller  ful byxor?

LUNDS UNIVERSITET

# Cacheminne

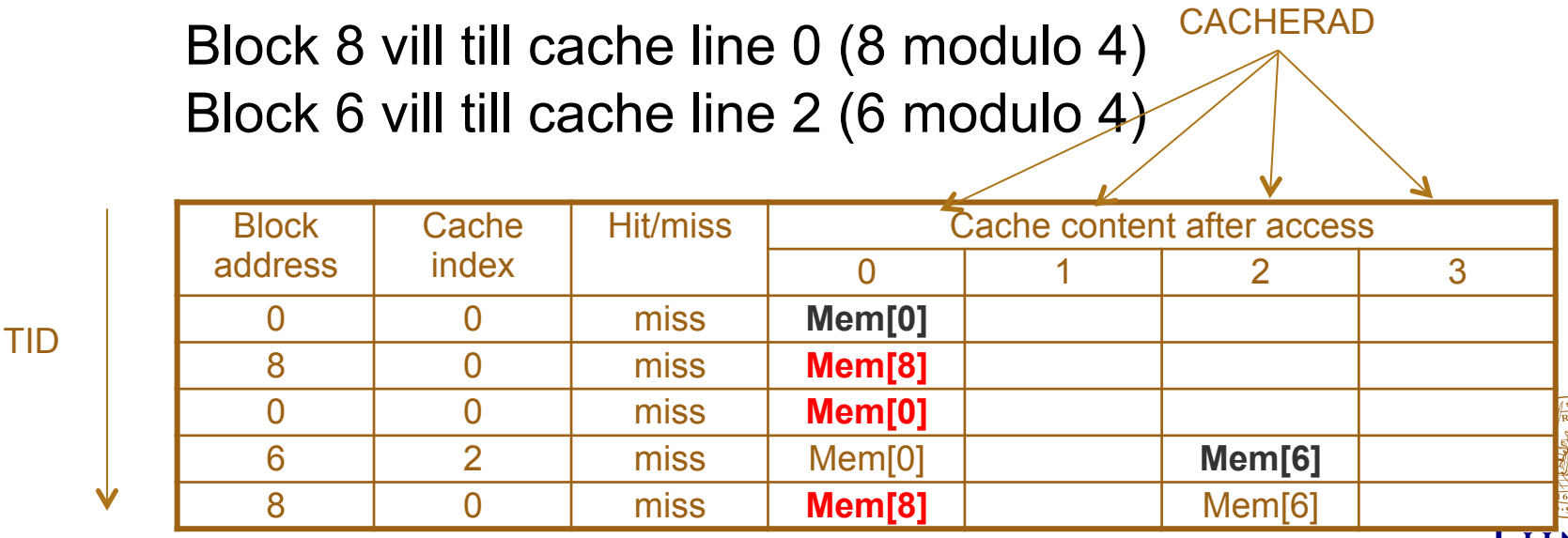



LUNDS UNIVERSITET

# Jämför cacheminnen

- Direct mapped, 2-way set associative, fully associative
- Block access sequence: 0, 8, 0, 6, 8
- Direct mapped:

  Block 0 vill till cache line 0
  Block 8 vill till cache line 0 (8 modulo 4)
  Block 6 vill till cache line 2 (6 modulo 4)




| Block address | Cache index | Hit/miss | Cache content after access | | | |
|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 |
| 0 | 0 | miss | **Mem[0]** | | | |
| 8 | 0 | miss | **Mem[8]** | | | |
| 0 | 0 | miss | **Mem[0]** | | | |
| 6 | 2 | miss | Mem[0] | | **Mem[6]** | |
| 8 | 0 | miss | **Mem[8]** | | Mem[6] | |

# Jämför cacheminnen

- Direct mapped, 2-way set associative, fully associative
- Block access sequence: 0, 8, 0, 6, 8
- 2-way set associative:

  Block 0 vill till set 0 (0 modulo 2)
  Block 8 vill till set 0 (0 modulo 2)
  Block 6 vill till set 0 (0 modulo 2)

TID ↓

| Block address | Cache index | Hit/miss | Cache content after access | | | |
|---|---|---|---|---|---|---|
| | | | Set 0 | | Set 1 | |
| 0 | 0 | miss | **Mem[0]** | | | |
| 8 | 0 | miss | Mem[0] | **Mem[8]** | | |
| 0 | 0 | hit | **Mem[0]** | Mem[8] | | |
| 6 | 0 | miss | Mem[0] | **Mem[6]** | | |
| 8 | 0 | miss | **Mem[8]** | Mem[6] | | |

LUNDS UNIVERSITET

# Jämför cacheminnen

- Direct mapped, 2-way set associative, fully associative
- Block access sequence: 0, 8, 0, 6, 8
- Fully associative: Block kan placeras var som helst

TID ↓

| Block address | | Hit/miss | Cache content after access | | | |
|---|---|---|---|---|---|---|
| 0 | | miss | **Mem[0]** | | | |
| 8 | | miss | Mem[0] | **Mem[8]** | | |
| 0 | | hit | **Mem[0]** | Mem[8] | | |
| 6 | | miss | Mem[0] | Mem[8] | **Mem[6]** | |
| 8 | | hit | Mem[0] | **Mem[8]** | Mem[6] | |

LUNDS UNIVERSITET

# Ersättningsalgoritmer

- Slumpmässigt val – en av kandidaterna väljs slumpmässigt
- Least recently used (LRU) – kandidat är den cacherad vilken varit i cachen men som inte blivit refererad (läst/ skriven) på länge
- First-In First Out (FIFO) – kandidat är den som varit längst i cacheminnet
- Least frequently used (LFU) – kandidat är den cacherad som refererats mest sällan
- Ersättningsalgoritmer implementeras i hårdvara – prestanda viktigt.

LUNDS UNIVERSITET

# Skrivstrategier

- Problem: håll minnet konsistent
- Exempel:

```
x=0;
while (x<1000)
   x=x+1;
```

- Variabeln x finns i primärminnet och en kopia finns i cacheminnet
- I primärminnet: x=0
  i cacheminnet är x=0,1,2...1000
- Uppdatera: När och hur?

LUNDS UNIVERSITET

# Skrivstrategier

- Write-through
  - skrivningar i cache görs också direkt i primärminnet
- Write-through with buffers
  - skrivningar buffras och görs periodiskt
- Write (Copy)-back
  - primärminnet uppdateras först när en cacherad byts ut
- ofta används en bit som markerar om en cacherad blivit modifierad (dirty))
- (Omodifierade cacherader behöver inte skrivas i primärminnet)

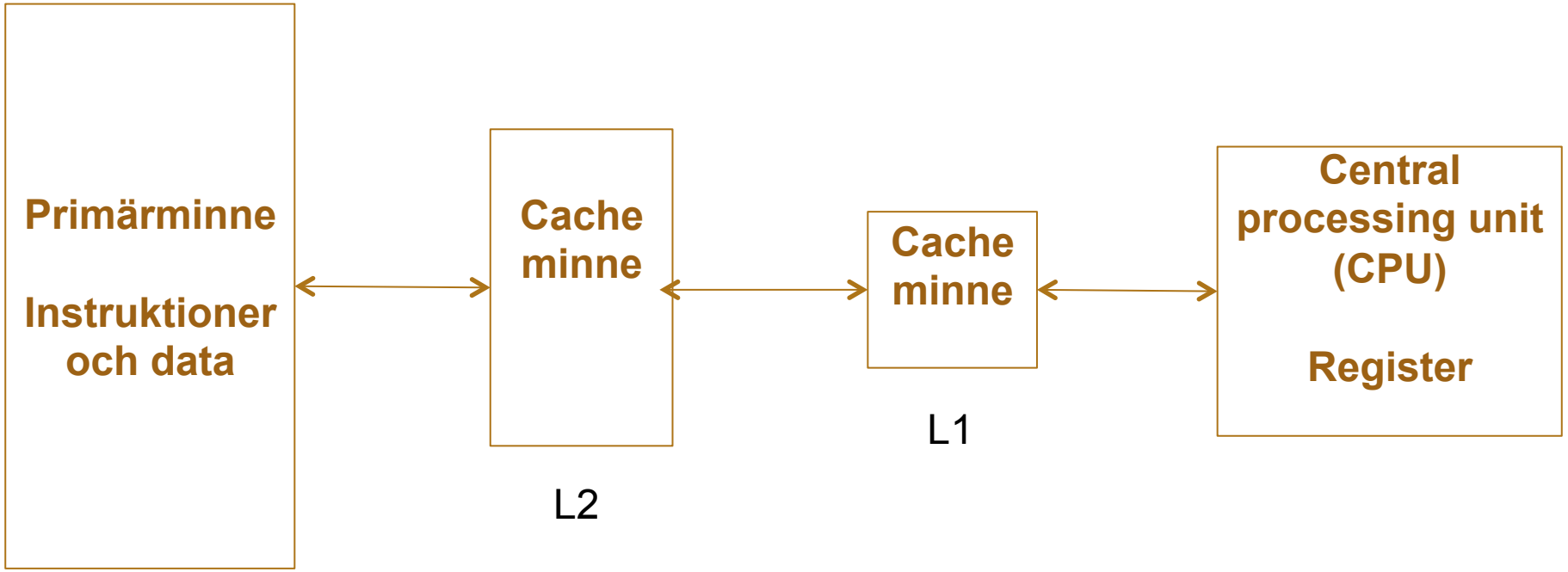
LUNDS UNIVERSITET

# Skrivstrategier

- Skilj på write-hit och write-miss
  - Write-hit: se ovan (uppdaterar data som finns i cache)
  - Write-miss: Vill skriva på plats som inte finns i cacheminne
    - » Alternativ:
      - Allokera vid miss: hämta block från primärminne
      - Write around: hämta inte in block från primärminne, skriv direkt i primärminne
    - » (För write-back: vanligen fetch block)

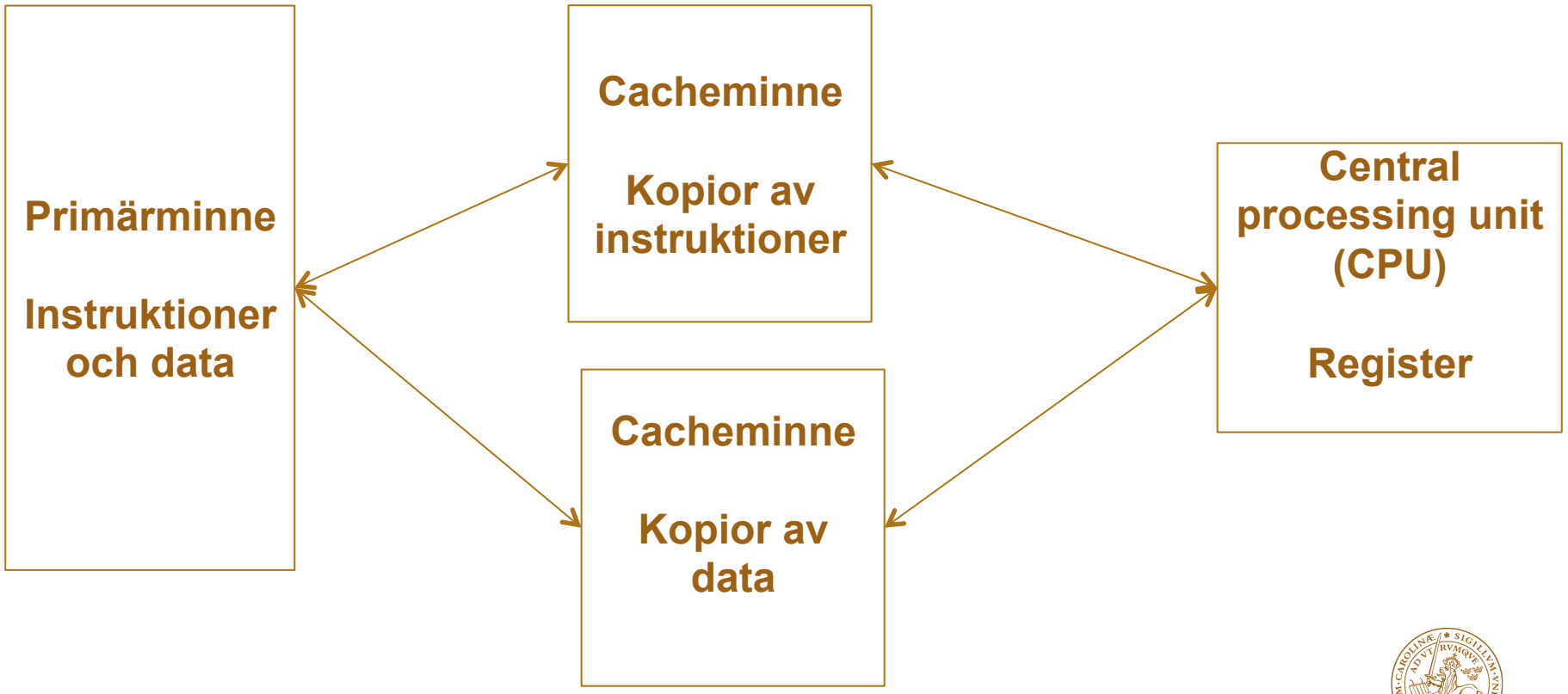
LUNDS UNIVERSITET

# Antal cachenivåer (levels)




LUNDS UNIVERSITET

# Separat instruktion/data cache



# Prestanda

- CPU tid påverkas av:
  - Cykler för programexekvering
    - » Inklusive cache hit tid
  - Tid för access i primärminne (Memory stall cycles)
    - » I huvudsak från cachemissar

**Hur mycket läses i minnet?**

**Hur ofta saknas data i cache?**

**Vad kostar en miss (tid)?**

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

LUNDS UNIVERSITET

# Prestanda

- Givet:
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2 (Clocks per instruction)
  - Load & stores är 36% av instruktionerna
- Misscykler per instruktion
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
  - Actual CPI = 2 + 2 + 1.44 = 5.44

Optimal CPU är 5.44/2 =2.72 gånger snabbare

**Om man bortser från minnesaccesser, så här snabbt går processorn**

**Antag att bara load och store används för access till minnet**

**Tid för verklig processor**

LUNDS UNIVERSITET

# Prestanda

- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Exempel:
  - CPU med 1ns klocktid, hit tid = 1 cykel,
    miss penalty = 20 cykler, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
  - Vilket är 2 klockcykler per instruktion

LUNDS UNIVERSITET

# Prestanda – multilevel cache

- Givet:
  - CPU med CPI=1, klockfrekvens = 4GHz (0.25 ns)
  - Miss rate/instruktion = 2%
  - Accesstid till primärminnet=100ns
- Med 1 cache nivå (L1)
  - Miss penalty = 100ns/0.25ns=400 cykler
- Effektiv CPI=1+0.02*400=9

**Om man bortser från minnesaccesser, så här snabbt går processorn**

**Så här mycket kostar en miss**

LUNDS UNIVERSITET

# Prestanda – multilevel cache

- Lägg till L2 cache:
  - Accesstid = 5 ns
  - Global miss rate till primärminnet = 0.5%
- Med 1 cache nivå (L1)
  - Miss penalty = 5ns/0.25ns=20 cykler
- Effektiv CPI=1+0.02*20+0.005*400=3.4
- Jämför 1-nivå cache och 2-nivå cache: 9/3.4=2.6

Förra slide

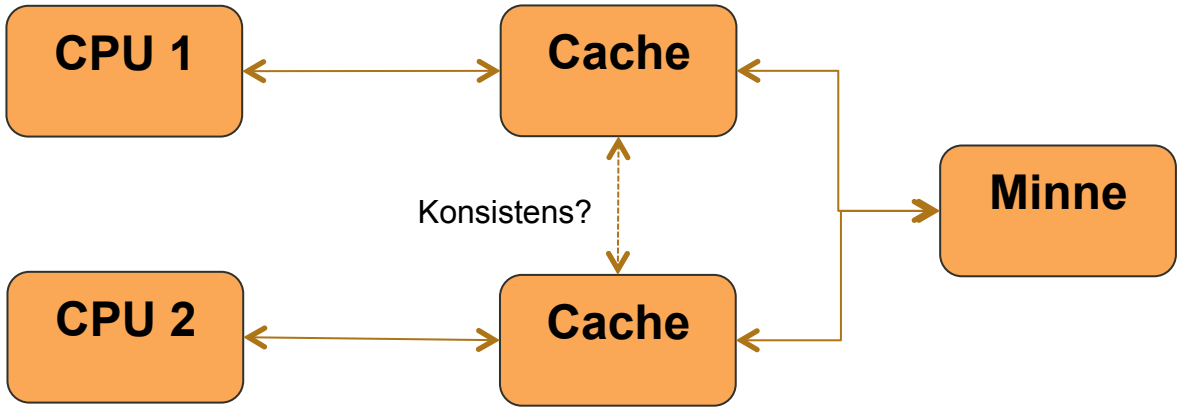
Förra slide

# Prestanda

- När CPU prestanda ökar, så blir miss penalty viktig att minimera
- För att undersöka prestanda måste man ta hänsyn till cacheminne
- Cachemissar beror på algoritm(implementation) och kompilatorns optimering

LUNDS UNIVERSITET

# Cache coherency



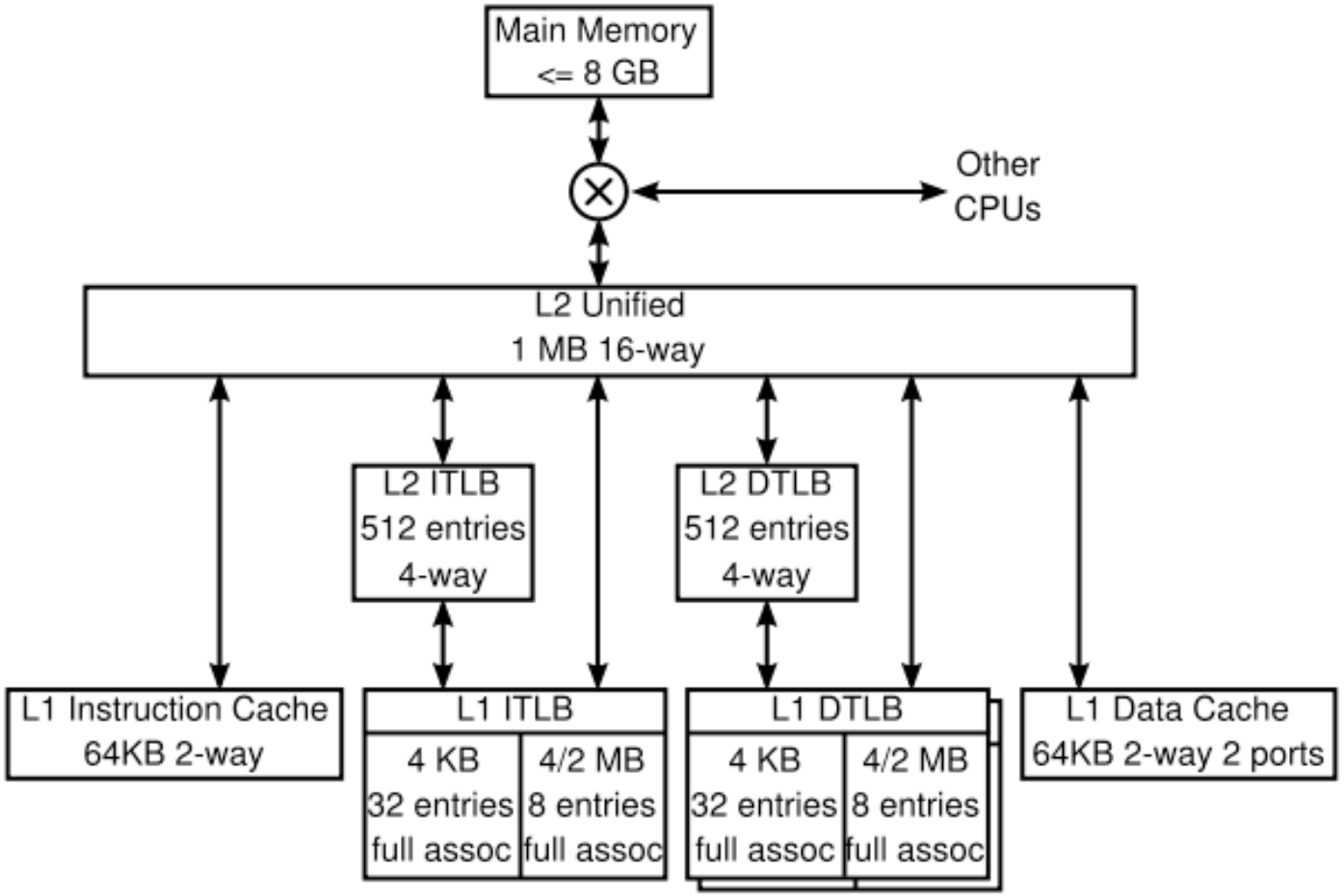
# Cache coherency - problem

- Antag att två CPU cores delar adressrymd
  - Write-through (skrivningar görs direkt)

| Time step | Event | CPU A's cache | CPU B's cache | Memory |
|---|---|---|---|---|
| 0 | | | | 0 |
| 1 | CPU A reads X | 0 | | 0 |
| 2 | CPU B reads X | 0 | 0 | 0 |
| 3 | CPU A writes 1 to X | 1 | 0 | 1 |

LUNDS UNIVERSITET

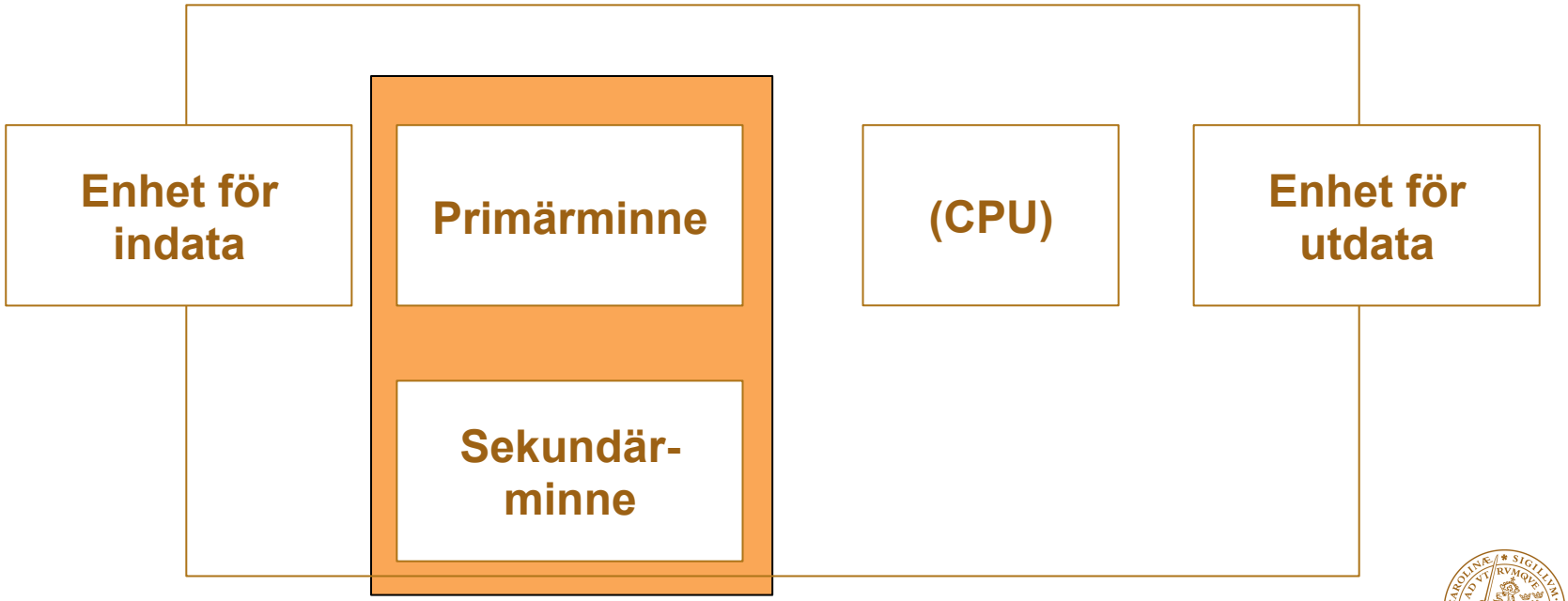
# AMD Athlon 64 CPU




CPU

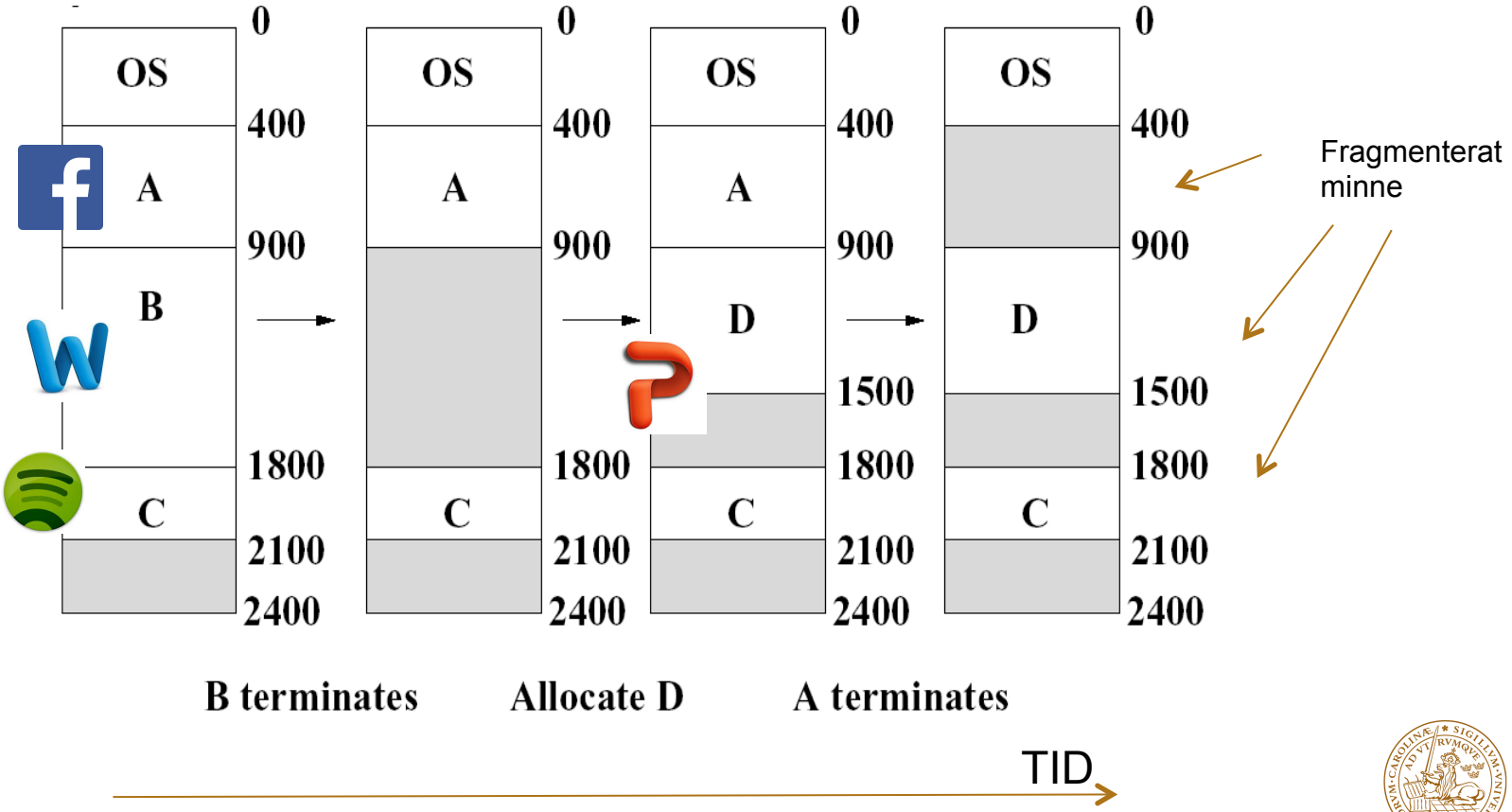
LUNDS UNIVERSITET

# Minnets komponenter



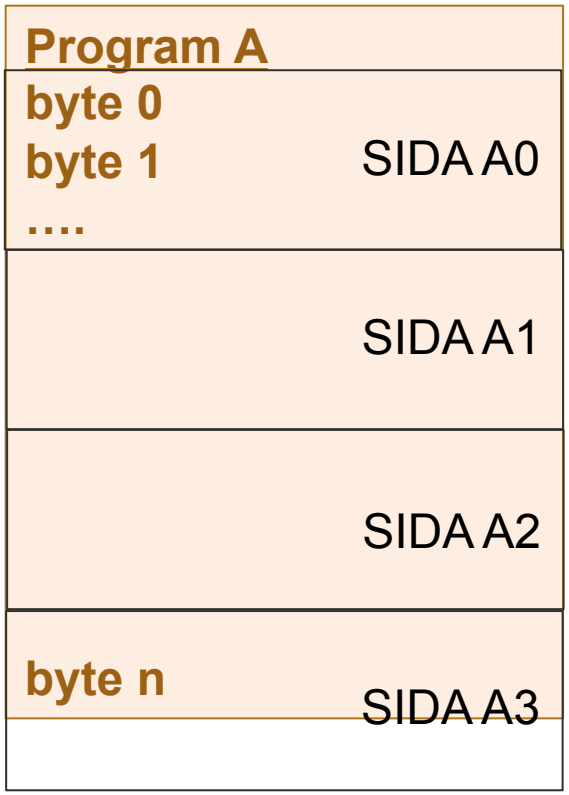

LUNDS UNIVERSITET

# Minnets innehåll över tiden

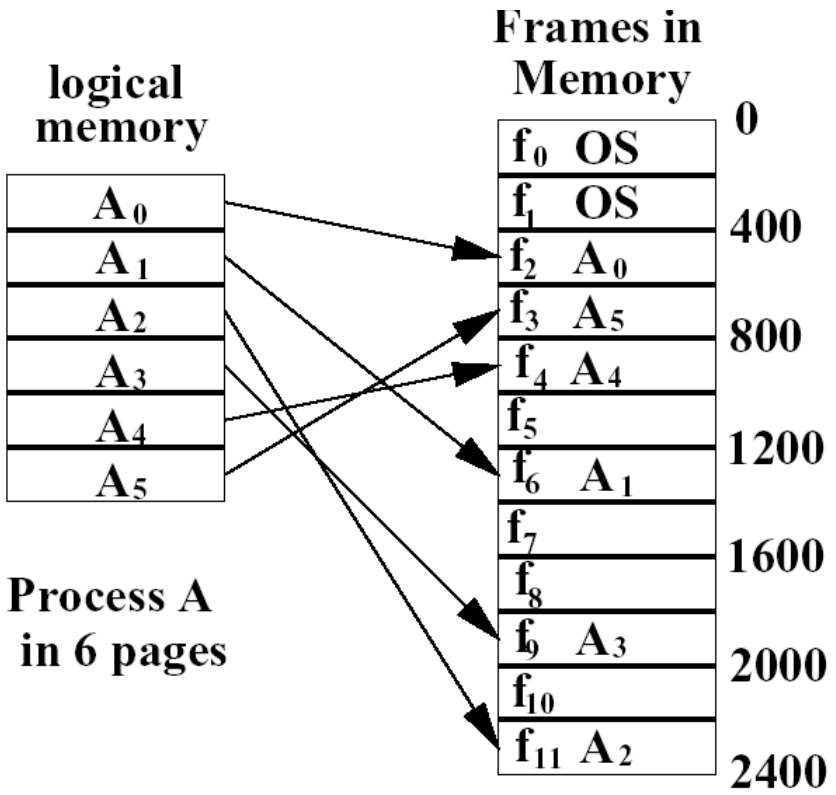



LUNDS UNIVERSITET

# Paging

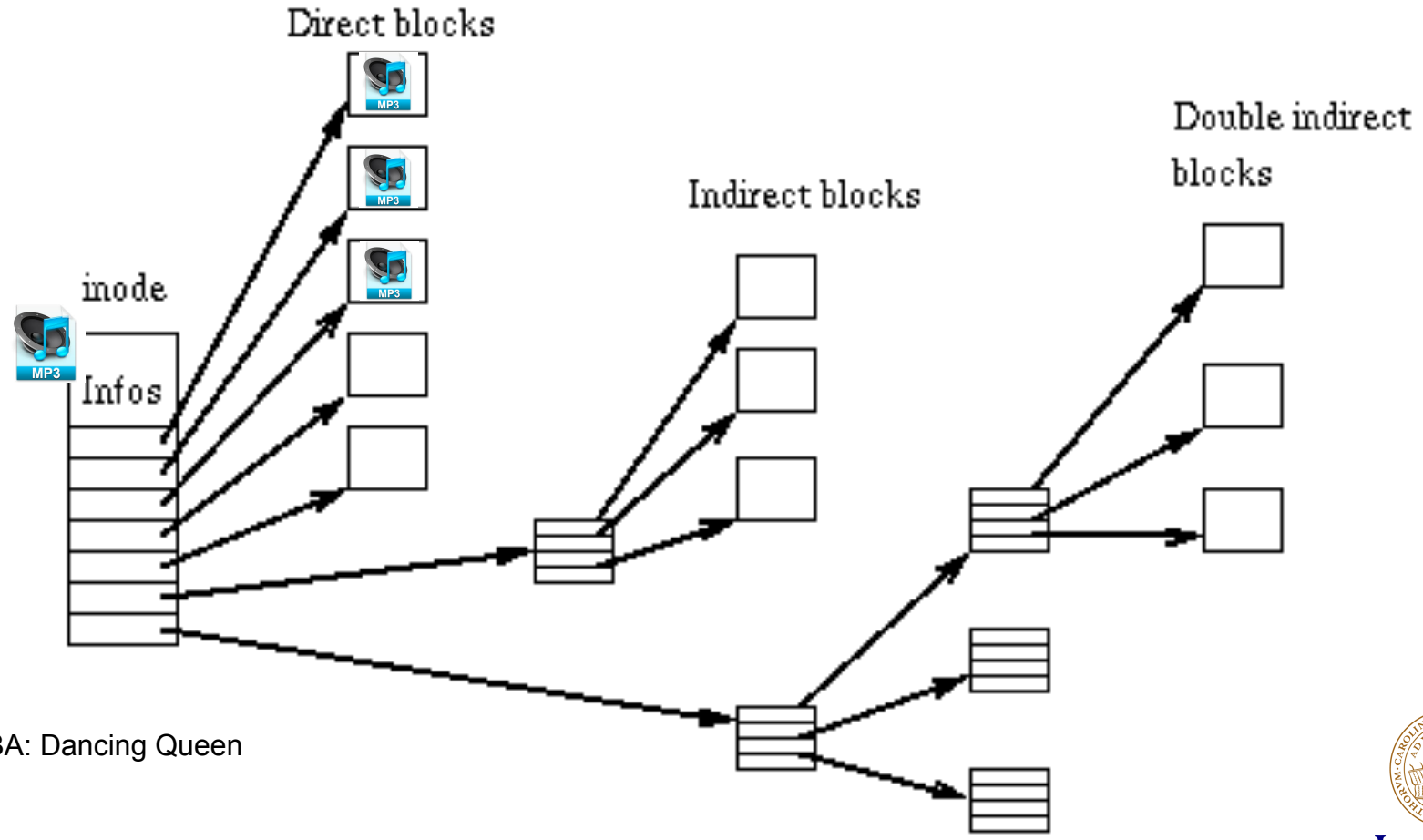



(lagring på hårddisk
ej sammanhängande –
se tidigare)




LUNDS UNIVERSITET

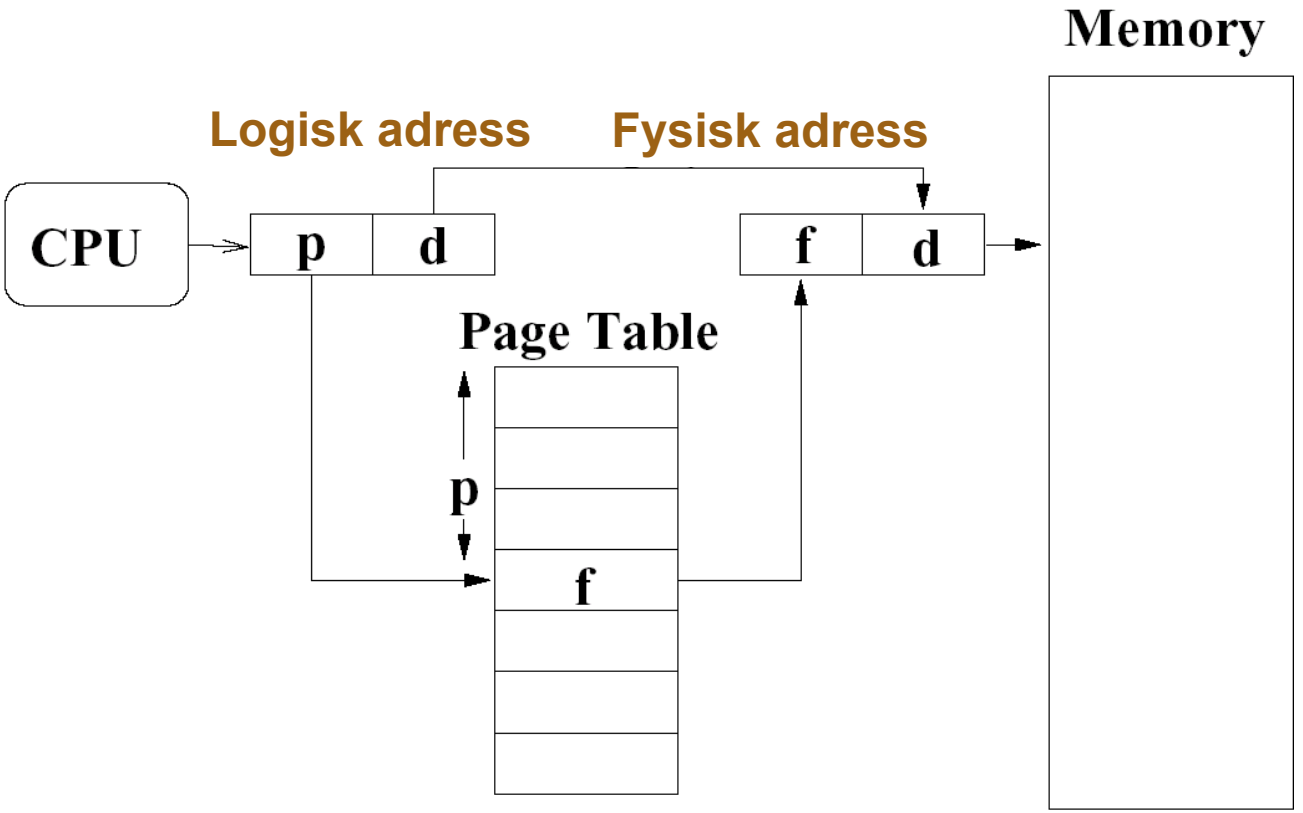
# Filsystem - Inode




ABBA: Dancing Queen

LUNDS UNIVERSITET

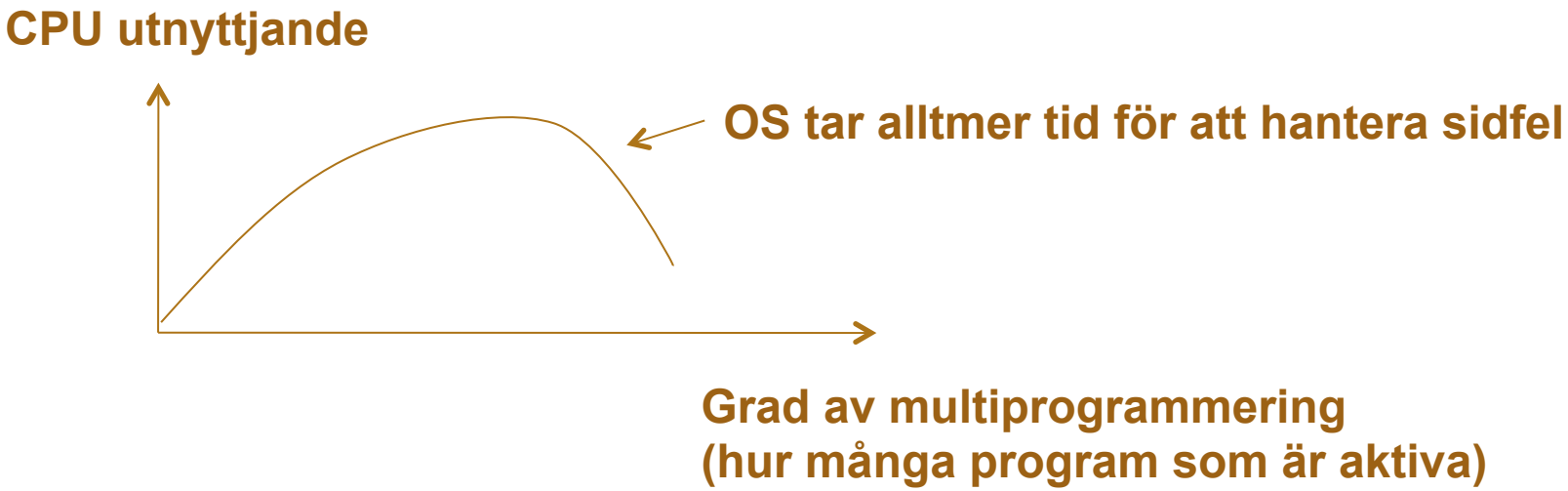
# Paging




LUNDS UNIVERSITET

# Demand paging

- Ladda endast de pages som behövs till primärminnet

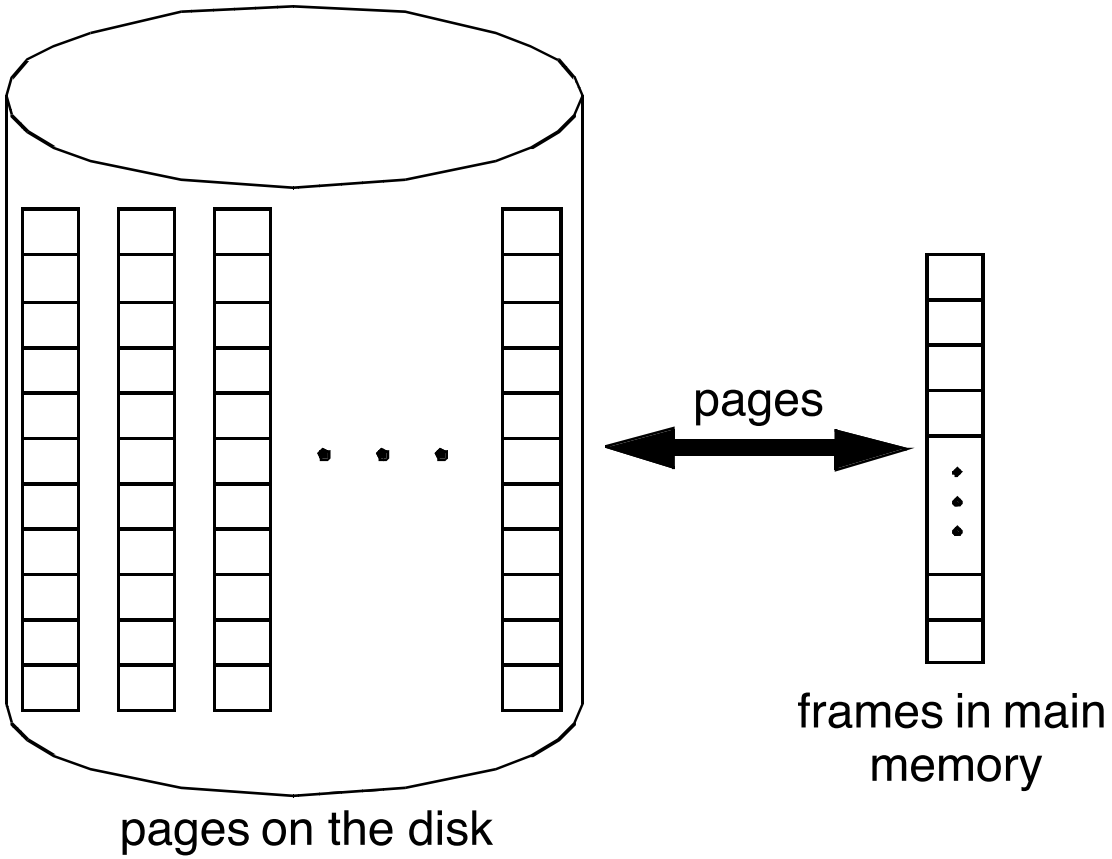


# Virtuellt minne

- Använd primärminne som “cache” för sekundärminne (hårddisk)
  - Hanteras med hårdvara och operativsystem
- Program delar primärminnet
  - Varje program får sin virtuella adressrymd
  - Skyddas från andra program
- CPU och OS översätter virtuella adresser till fysiska adresser
  - Ett “block” kallas för sida (page)
  - “Miss” kallas för sidfel (page fault)

LUNDS UNIVERSITET

# Virtuellt minne




LUNDS UNIVERSITET

# Virtuellt minne

- Exempel
  - Storlek på virtuellt minne: 2G ($2^{31}$) bytes
  - Primärminne: 16M ($2^{24}$) bytes
  - Sidstorlek (page): 2K ($2^{11}$) bytes

  - Antal sidor (pages): 2G/2K = 1M ($2^{31}/2^{11}=2^{20}$)
  - Antal ramar (frames): 16M/2K = 8K ($2^{24}/2^{11}=2^{13}$)

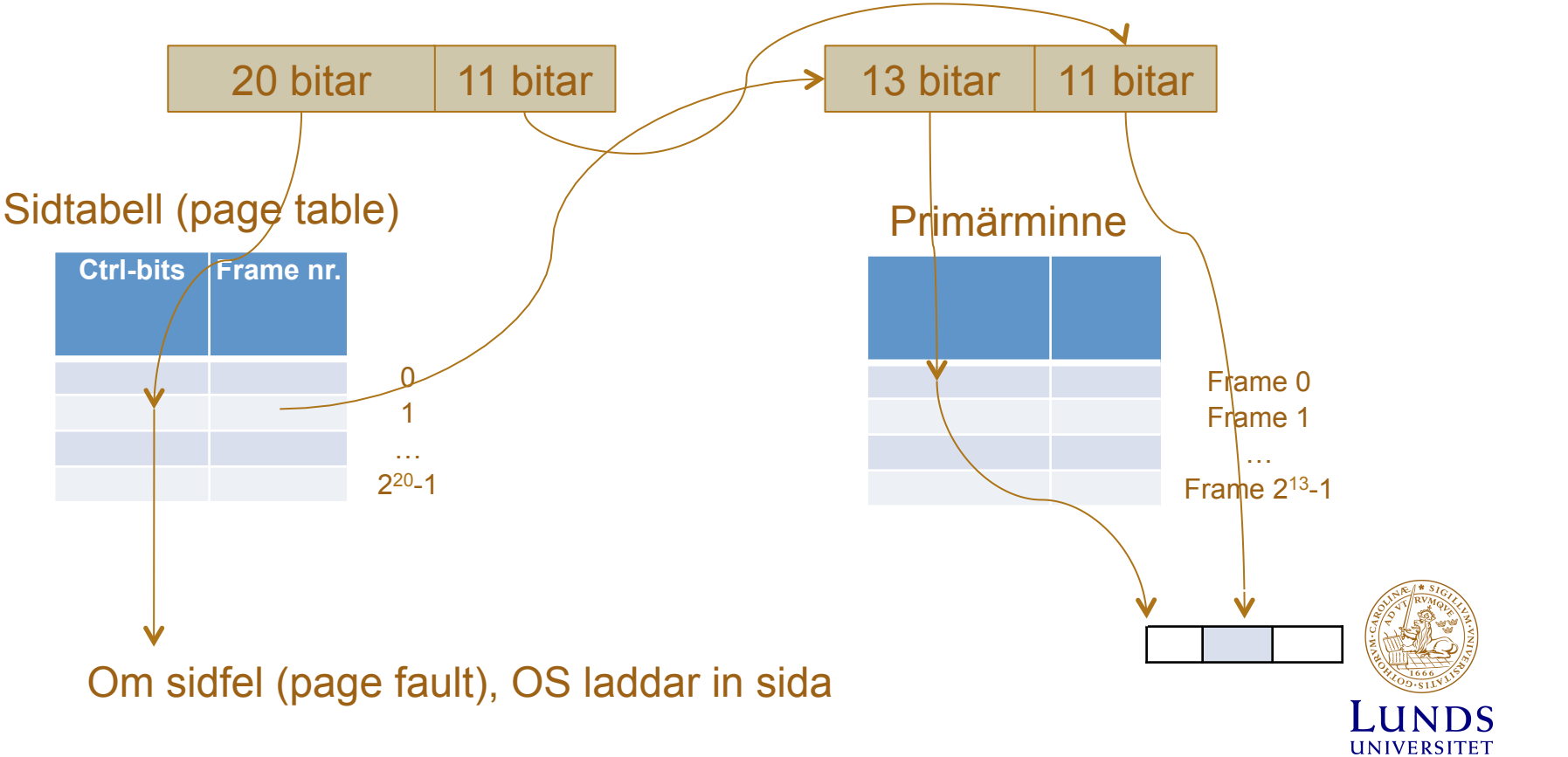
LUNDS UNIVERSITET

# Virtuellt minne

Memory Management Unit (MMU)

- Virtuell adress: 31 bitar

Fysisk adress: 24 bitar



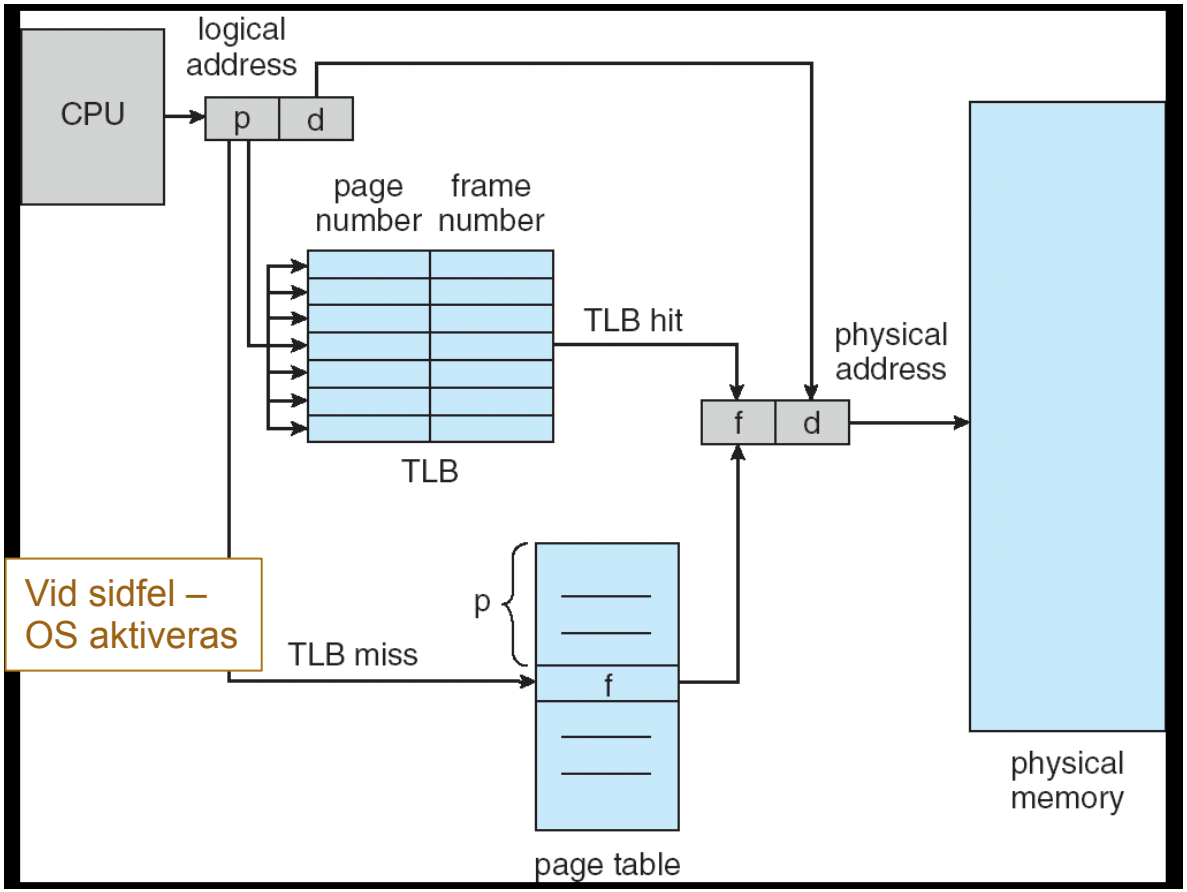

LUNDS UNIVERSITET

# Virtuellt minne

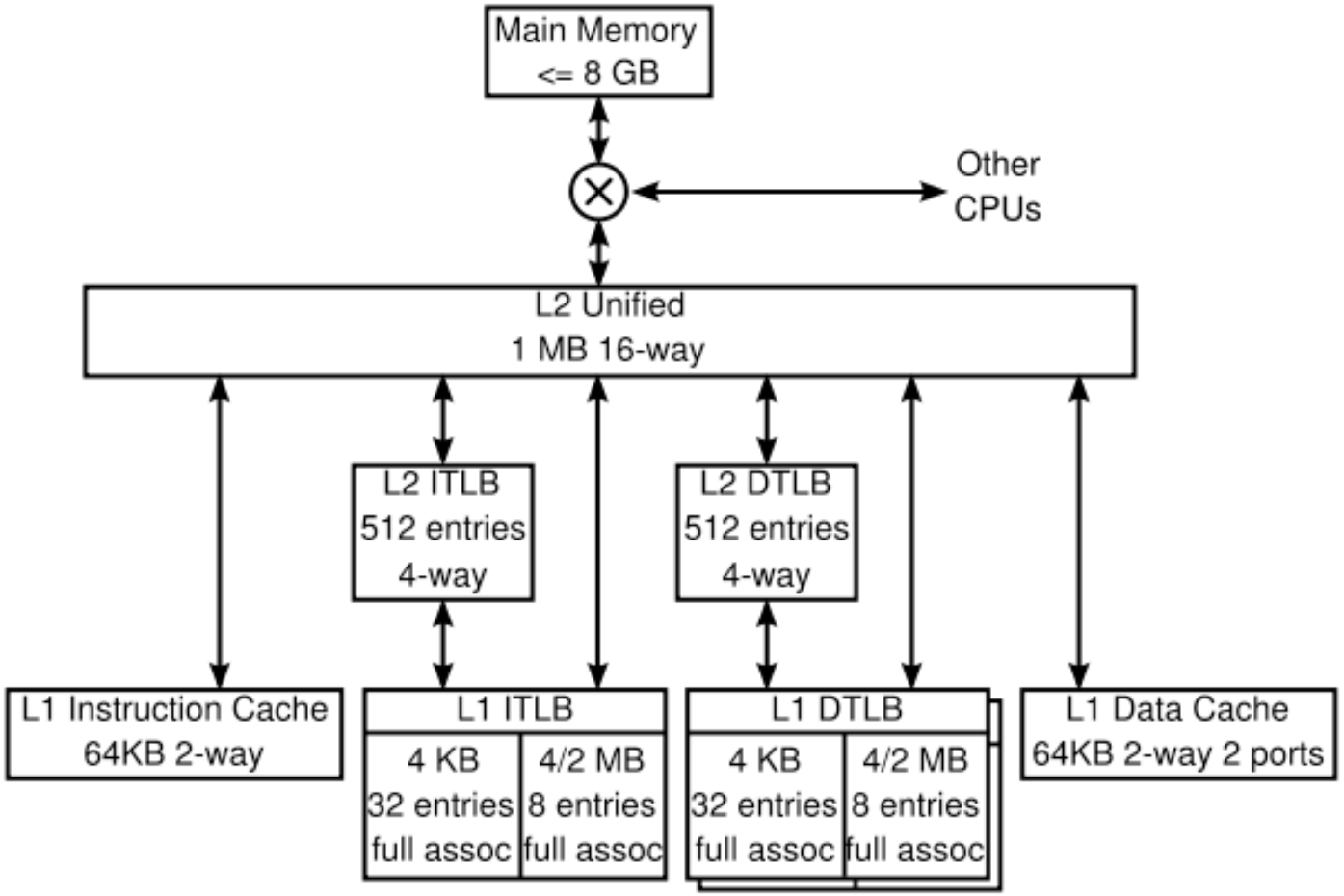
- Problem med sidtabell
  - Tid vid läsning av adress:
    - » 1 läs sidtabell
    - » 2 läs data
  - Stora sidtabeller
- Lösning: använd cache - Translation Look-Aside Buffer (TLB) – för sidtabeller

LUNDS UNIVERSITET

# Translation Look-Aside Buffer (TLB)



# AMD Athlon 64 CPU




CPU

LUNDS UNIVERSITET

# Sammanfattning

- Snabba minnen är små, stora minnen är långsamma
  - Vi vill ha snabba och stora minnen
  - Cacheminnen och virtuellt minne ger oss den illusionen
- Lokalitet viktigt för att cacheminnen och virtuellt minne ska fungera
  - Program använder vid varje tidpunkt en liten del av sitt minne ofta
- Minneshierarki
  - L1 cache <->L2 cache …. Primärminne - Sekundärminne

LUNDS UNIVERSITET


LUNDS
UNIVERSITET