LUNDS UNIVERSITET

Datorteknik

ERIK LARSSON

FÖ1

FÖ2

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

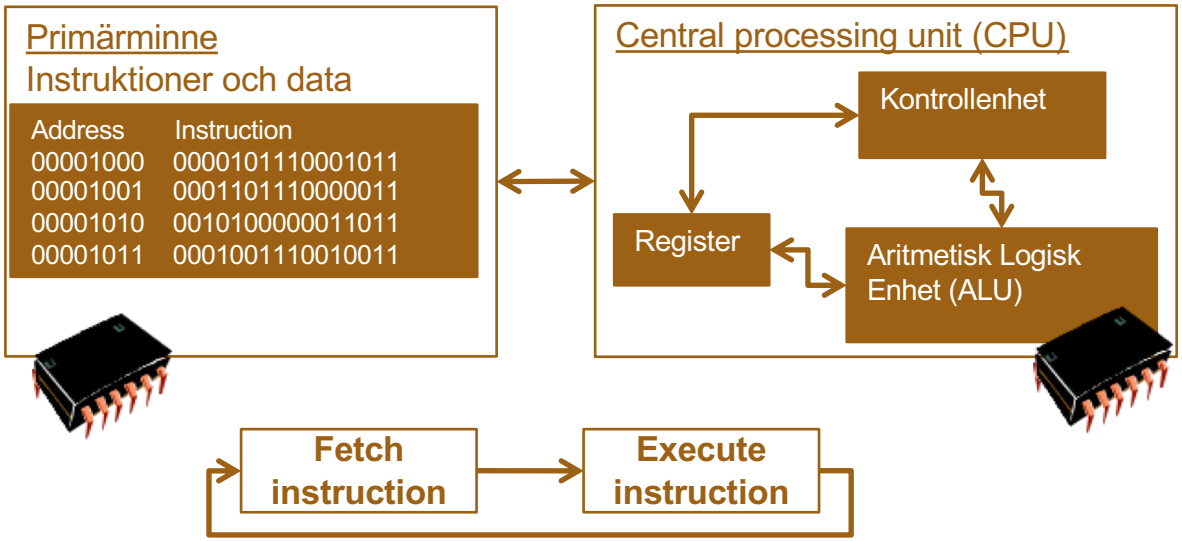
Assembly language program (for MIPS)

```
swap:
        muli $2, $5,4
        add  $2, $4,$2
        lw   $15, 0($2)
        lw   $16, 4($2)
        sw   $16, 0($2)
        sw   $15, 4($2)
        jr   $31
```

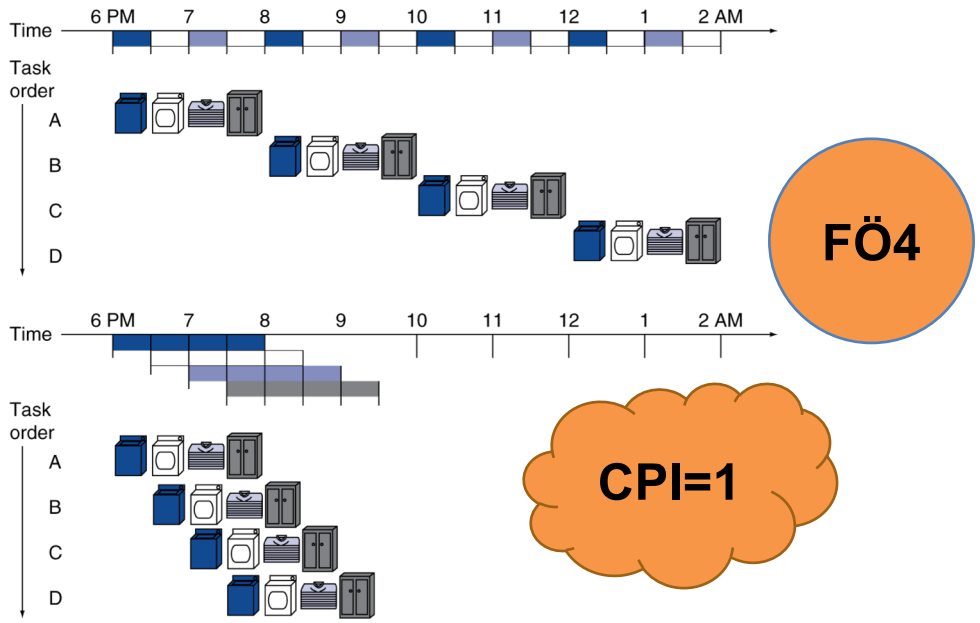
Assembler

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```



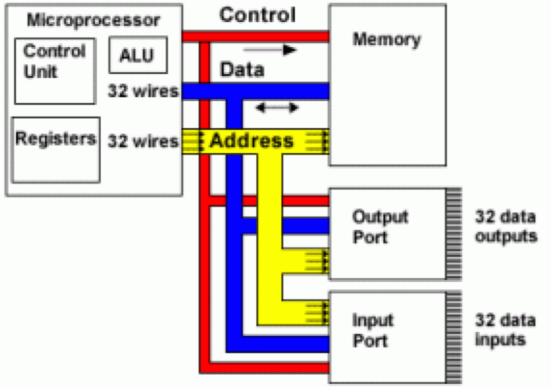
FÖ3



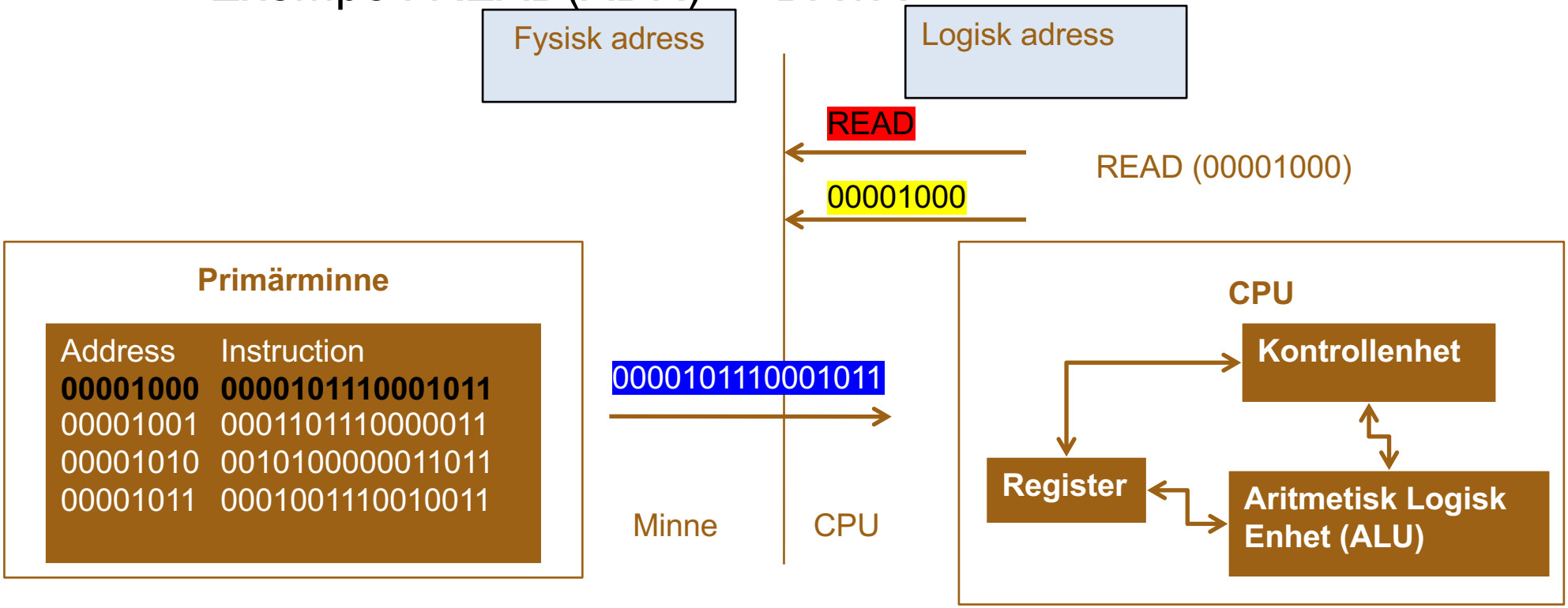
FÖ4

CPI=1

Minnet från processorns sida

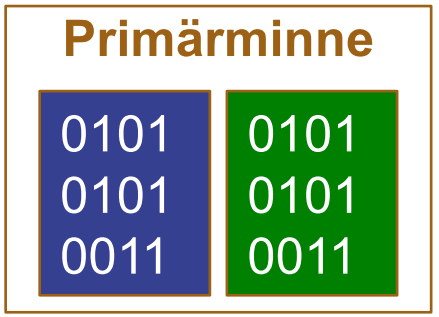


- Processorn ger kommandon/instruktioner med en adress och förväntar sig data.
  - Exempel: READ(ADR) -> DATA



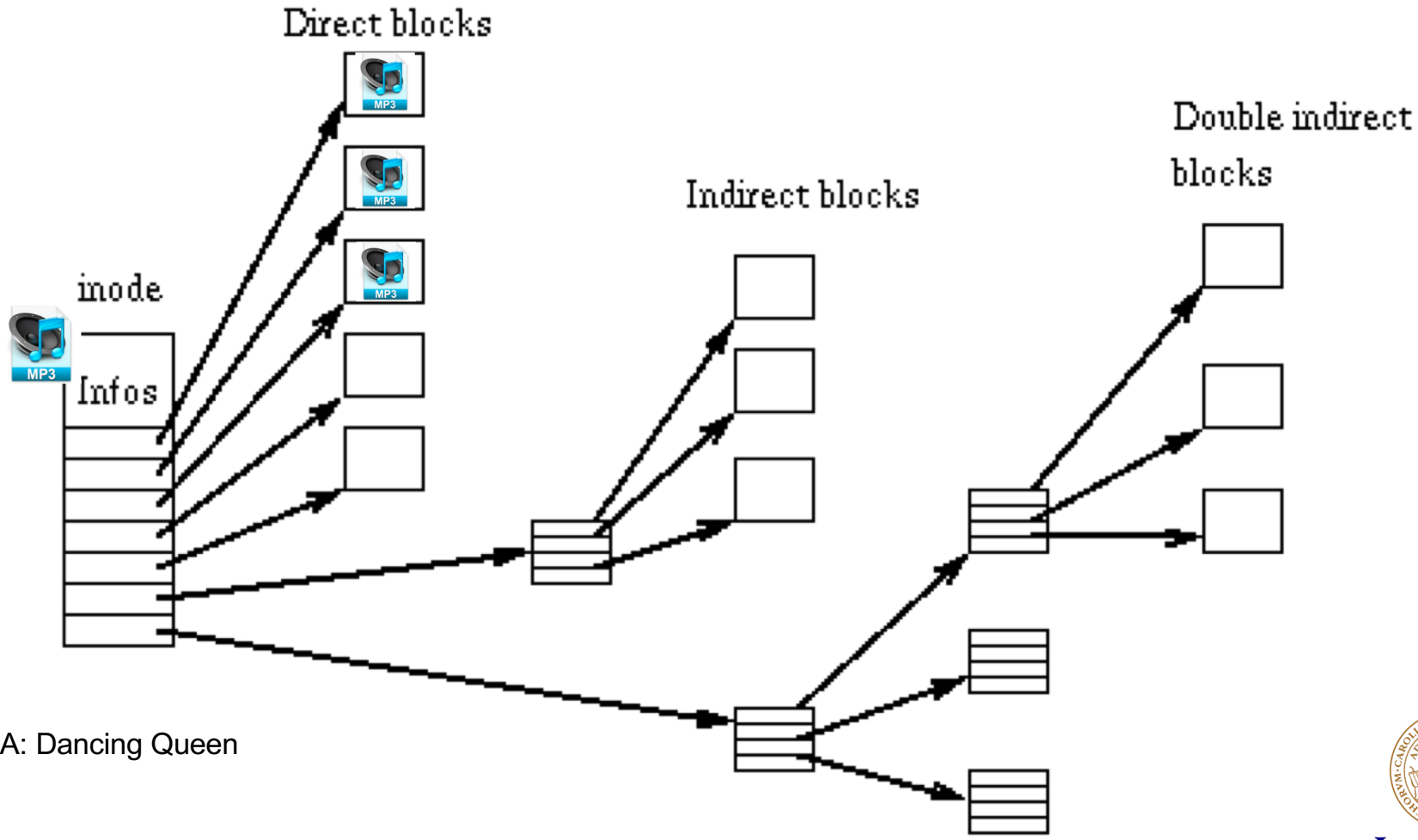
Minneshantering

- Vid multiprogrammering kommer flera olika program finnas i primärminnet. Kostar för mycket tid att flytta program till hårddisk vid kontext byte.
- T ex, två program ska exekveras “samtidigt”:



LUNDS UNIVERSITET

Filsystem - Inode



ABBA: Dancing Queen

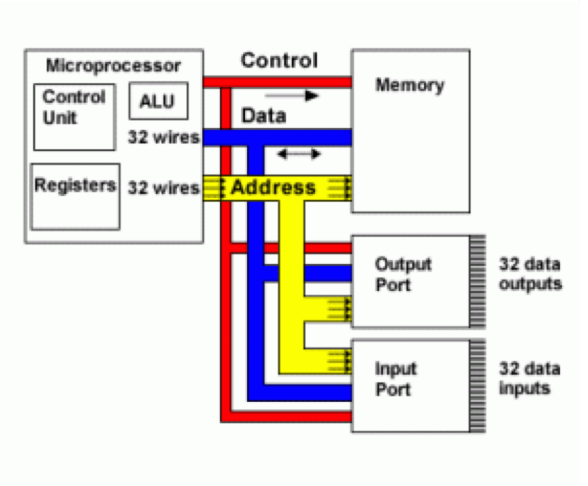
LUNDS UNIVERSITET

Minnet

- Minnet kan delas upp i primärminne och sekundärminne
- Primärminnet förlorar sitt innehåll när strömmen stängs av. Minnet är flyktigt (volatile)
  - Random-Access Memory (RAM)
    - » Dynamiska RAM (DRAM) och statiska RAM (SRAM)
- Sekundärminnet behåller sitt innehåll när strömmen slås av. Minnet är icke-flyktigt (non-volitile)
  - Hårddisk, flashminne, magnetband
- Andra: CD, DVD

LUNDS UNIVERSITET

Primärminne



Ett ord (word)

Minnesarea

1

READ(ADR)

Buffert för minnesadress

Adressavkodare

2

Adress 2
Adress 1
Adress 0

Databuffert

Minneskontroll .

En bit

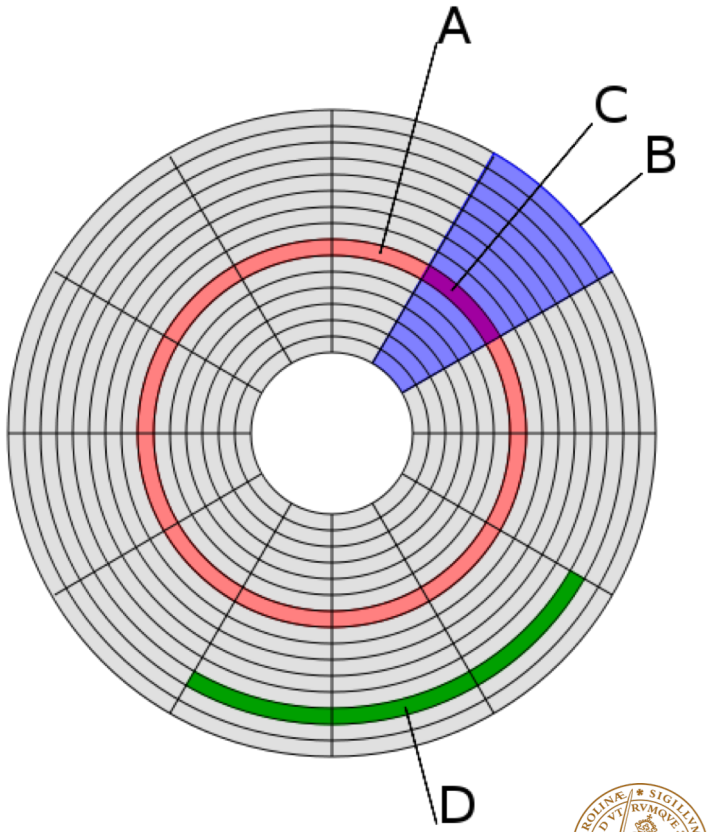
3

DATA .

LUNDS UNIVERSITET

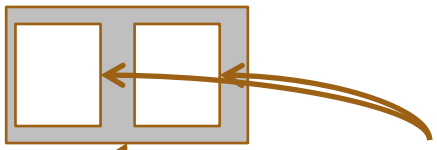
Sekundärminne

- A – track, (B – geometrisk sektor)
  C – sektor, D – cluster
- En sektor kan vara 512-4096 bytes och består av sektor header, data area, error korrektion kod (ECC)



Sektor

Exempel:



Cluster

Sektor

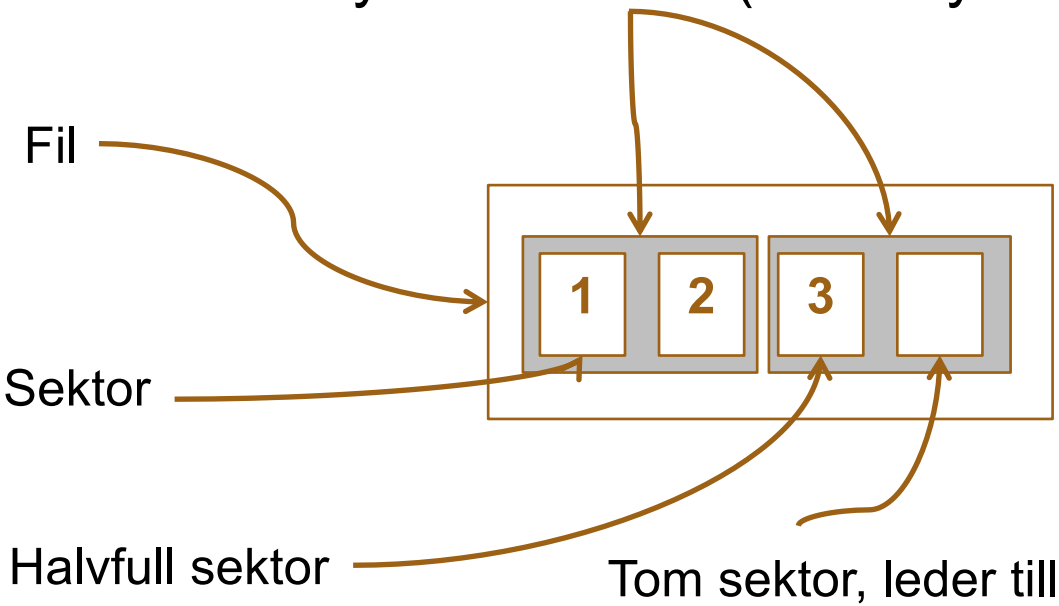
LUNDS UNIVERSITET

Sekundärminne

- Vill lagra en fil som är 1500 bytes
- Hårddisk
  - Antag sektor = 512, cluster = 2*512
- Lösning:
  - Ta 2 stycken cluster (2048 bytes)

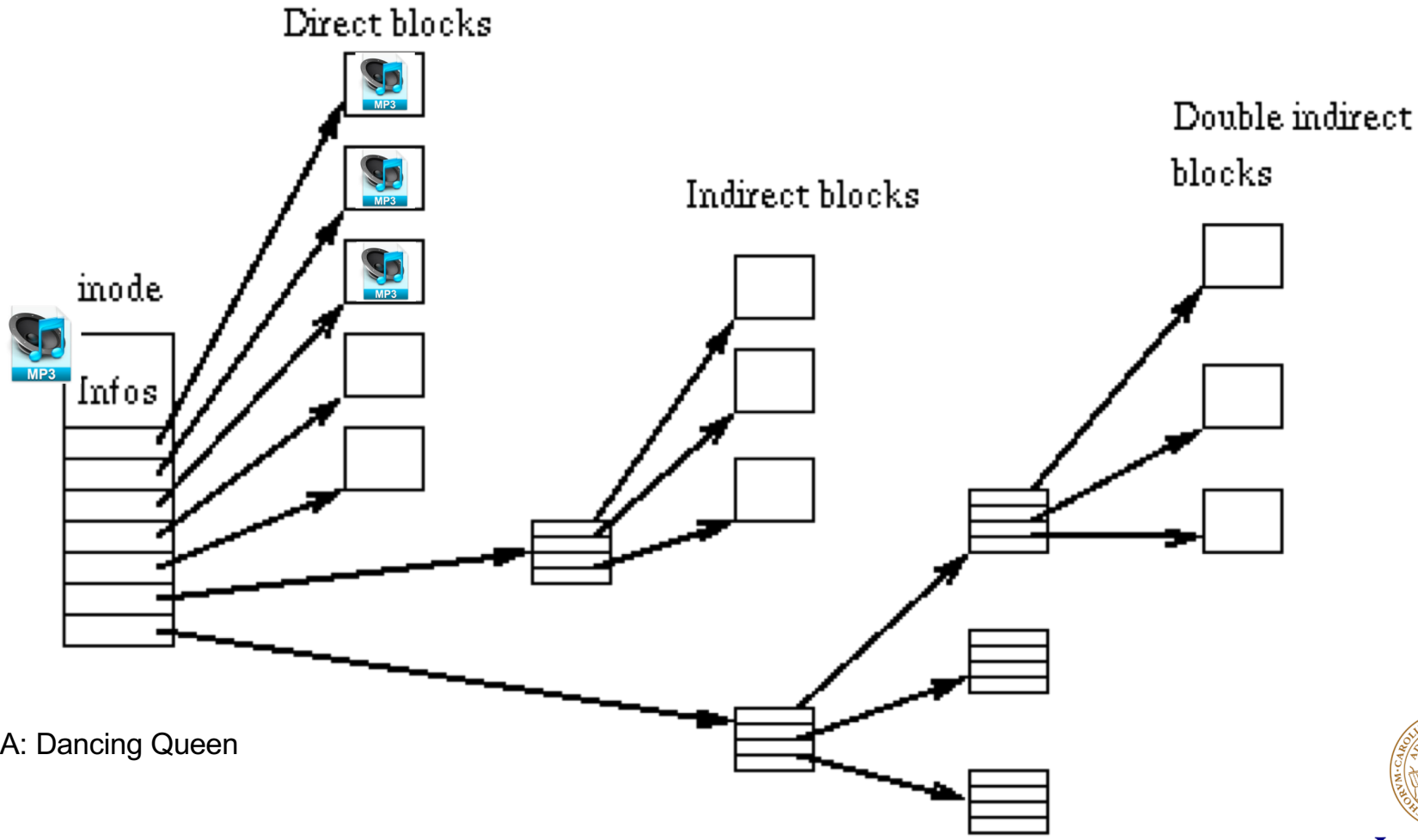
byte 0
byte 2
……
byte 1499
1500 bytes

byte 0
byte 2
……
byte 1023
2*512 bytes



LUNDS UNIVERSITET

Filsystem - Inode

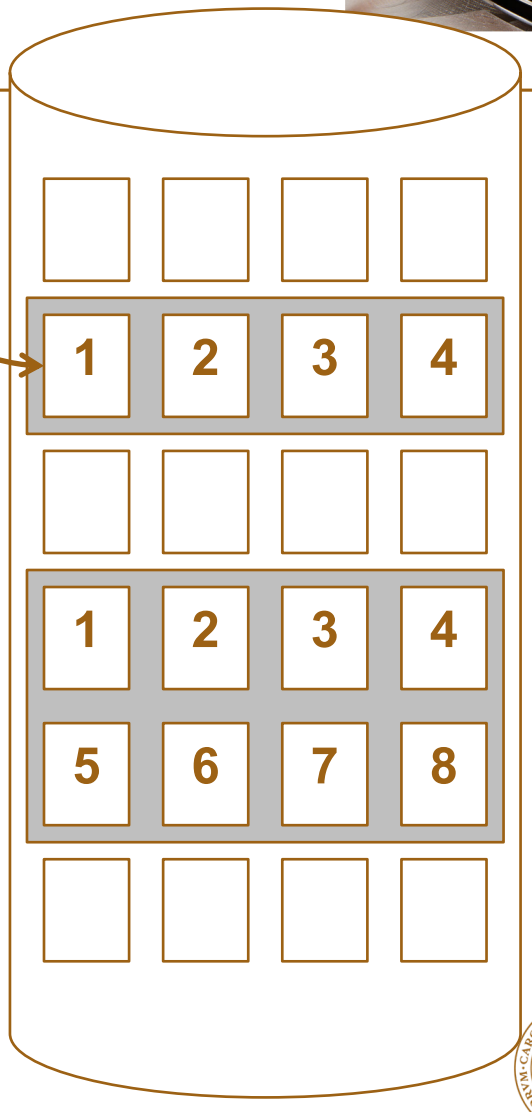


ABBA: Dancing Queen

LUNDS UNIVERSITET

Sekundärminne

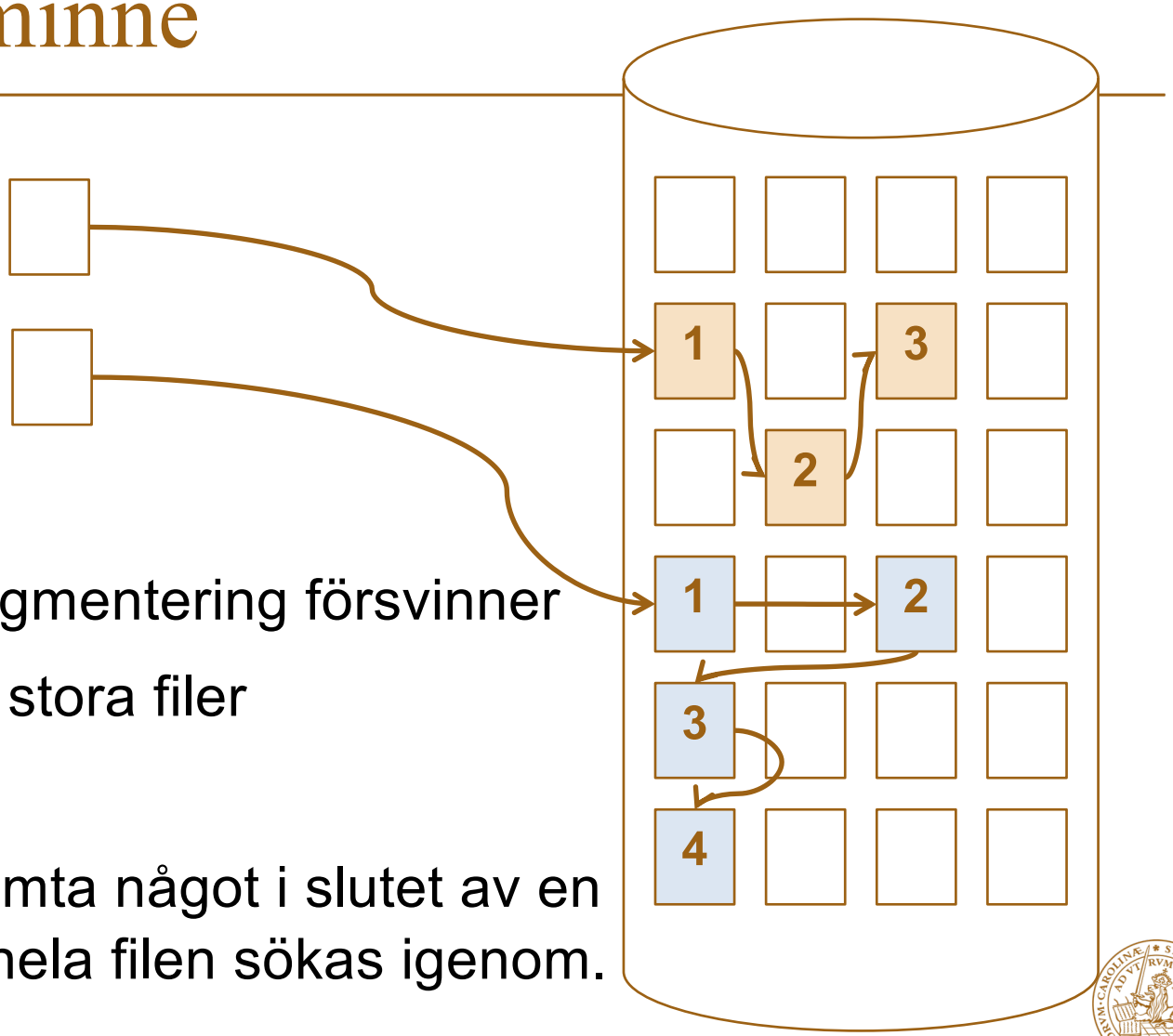
- Närliggande:
- Välj cluster som ligger bredvid varandra
- Problem – när många filer av olika storlek lagras kan det vara svårt att få plats med en stor fil trots att det finns plats.
  - Extern fragmentering: Exempel: Finns 12 lediga cluster (block). Vill lagra en fil som behöver 5 block – men hur?



LUNDS UNIVERSITET

Sekundärminne

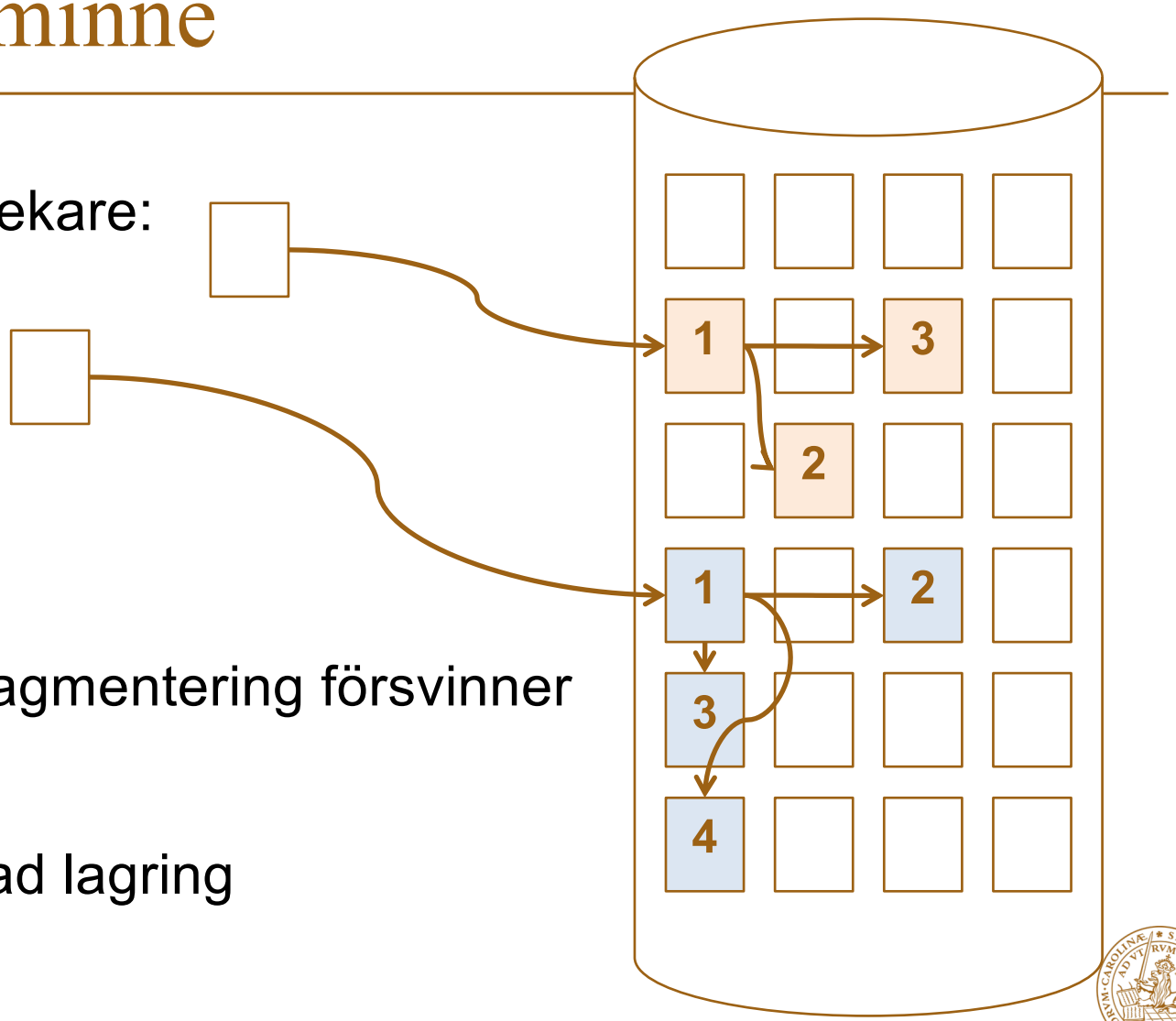
- Länkad lista:



- Fördel:
  - extern fragmentering försvinner
  - kan lagra stora filer
- Nackdel:
  - För att hämta något i slutet av en fil måste hela filen sökas igenom.

Sekundärminne

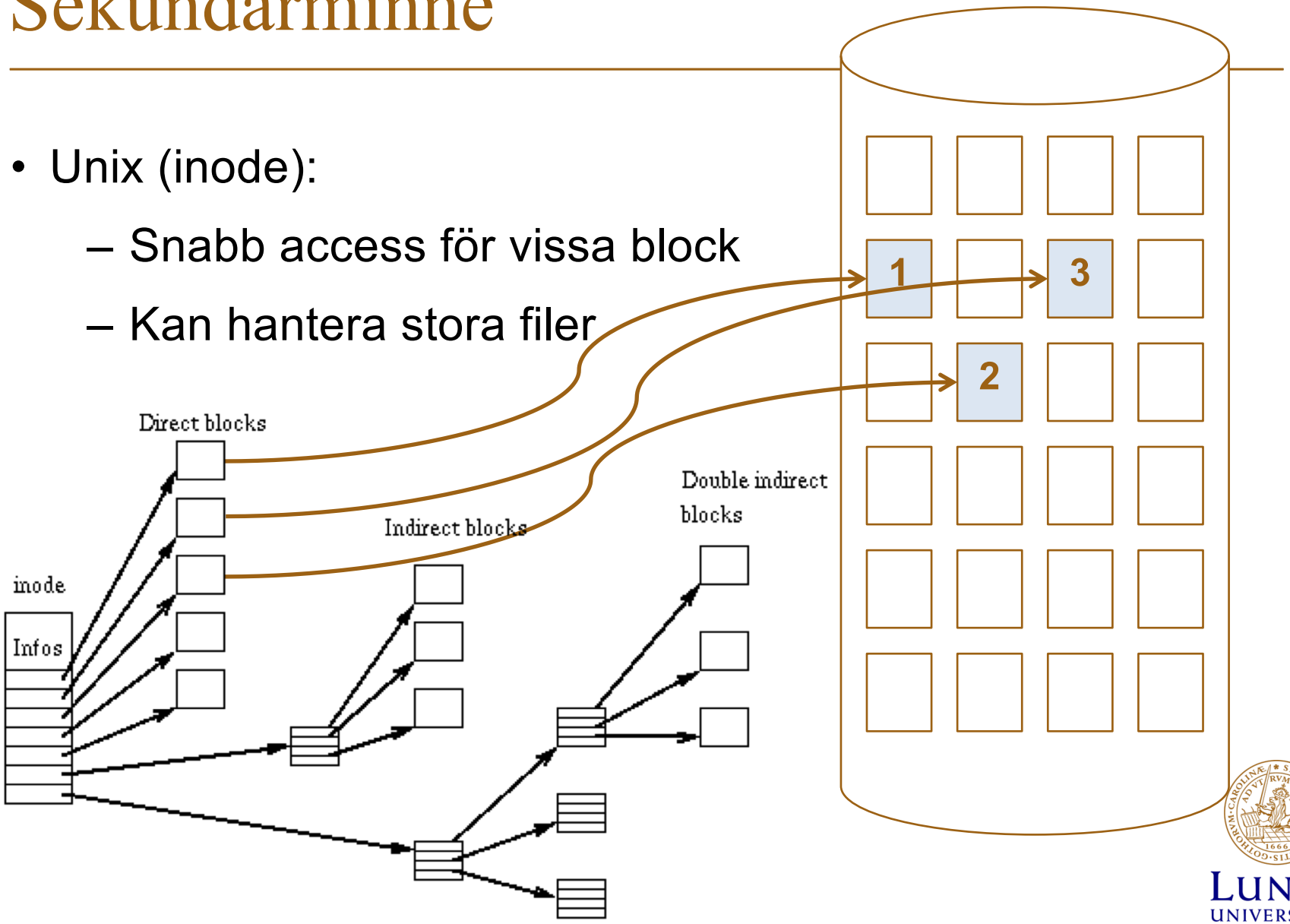
- Block med pekare:



- Fördel:
  - extern fragmentering försvinner
- Nackdel:
  - begränsad lagring

Sekundärminne

- Unix (inode):
  - Snabb access för vissa block
  - Kan hantera stora filer



Sekundärminne

- Schemaläggning (hårddisk)
  - Läs och skrivtid på hårddisk kritiskt
  - Var/hur filer lagras
  - Olika schemaläggare:
    - » shortest-seek time - from head
    - » elevator algorithm - move back and forth
    - » one-way elevator - move in one direction

LUNDS UNIVERSITET

Sekundärminne

- Flashminne
  - Utvecklat av Dr. Fujio Masuoka (Toshiba) kring 1980
- Mobiltelefoner, kameror, MP3-spelare och i datorer
  - Non-volatile och random access
- Kapacitet: mindre än en “hårddisk”
- Begränsat antal skrivningar
  - Block 0: bad blocks
  - Block 1: bootable block

LUNDS UNIVERSITET

Sekundärminne

- Lågnivåformatering
  - Dela in hårddisk i tracks och sectors
    - » En sector är 512-4098 bytes
- Partitioning
  - Dela in en fysisk hårddisk i en eller flera logiska hårddiskar, t ex C:, D:, E:
- Högnivåformatering
  - Bestäm för vilket operativ system hårddisken ska användas

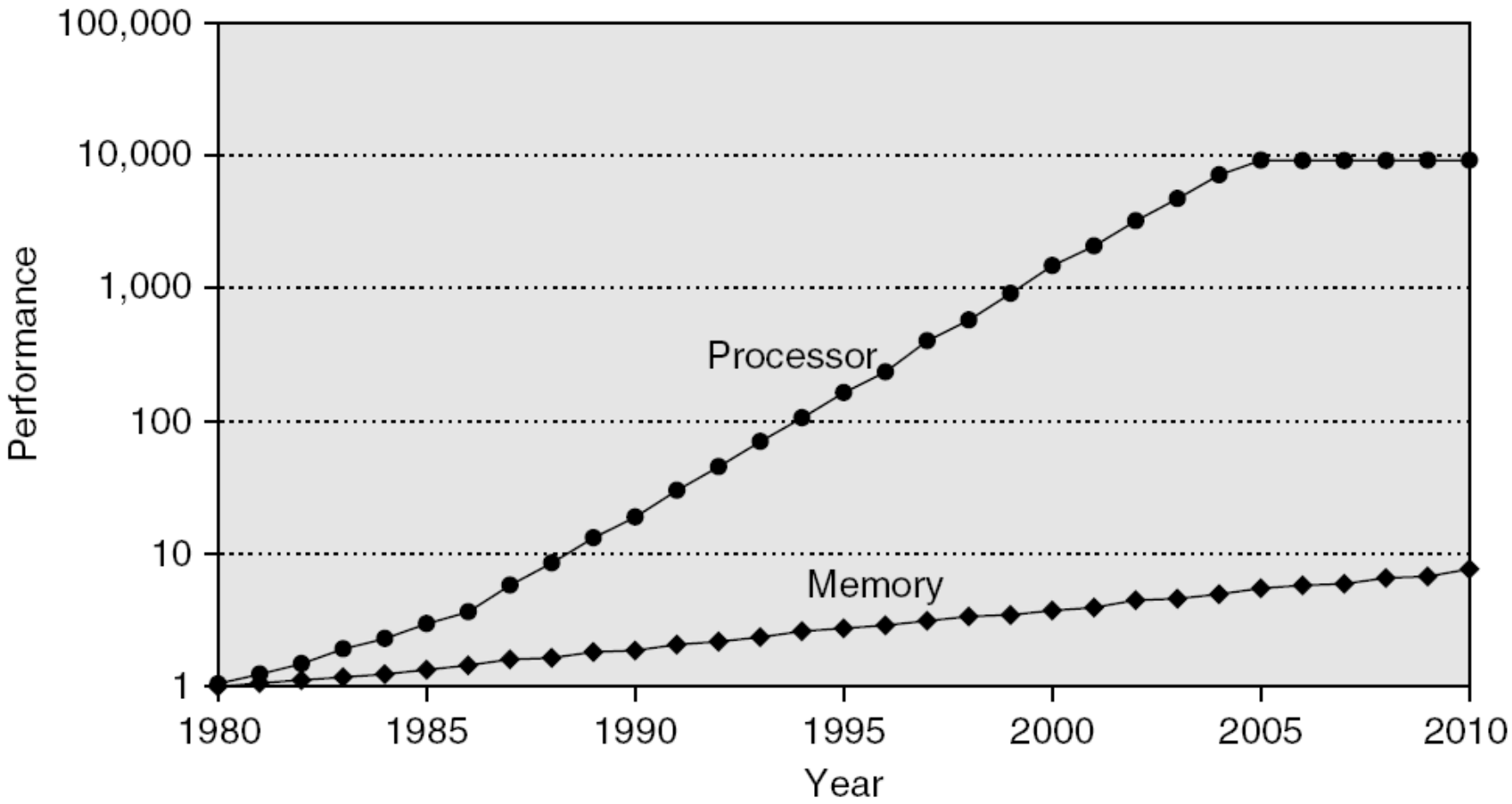
LUNDS UNIVERSITET

Design av minnesystem

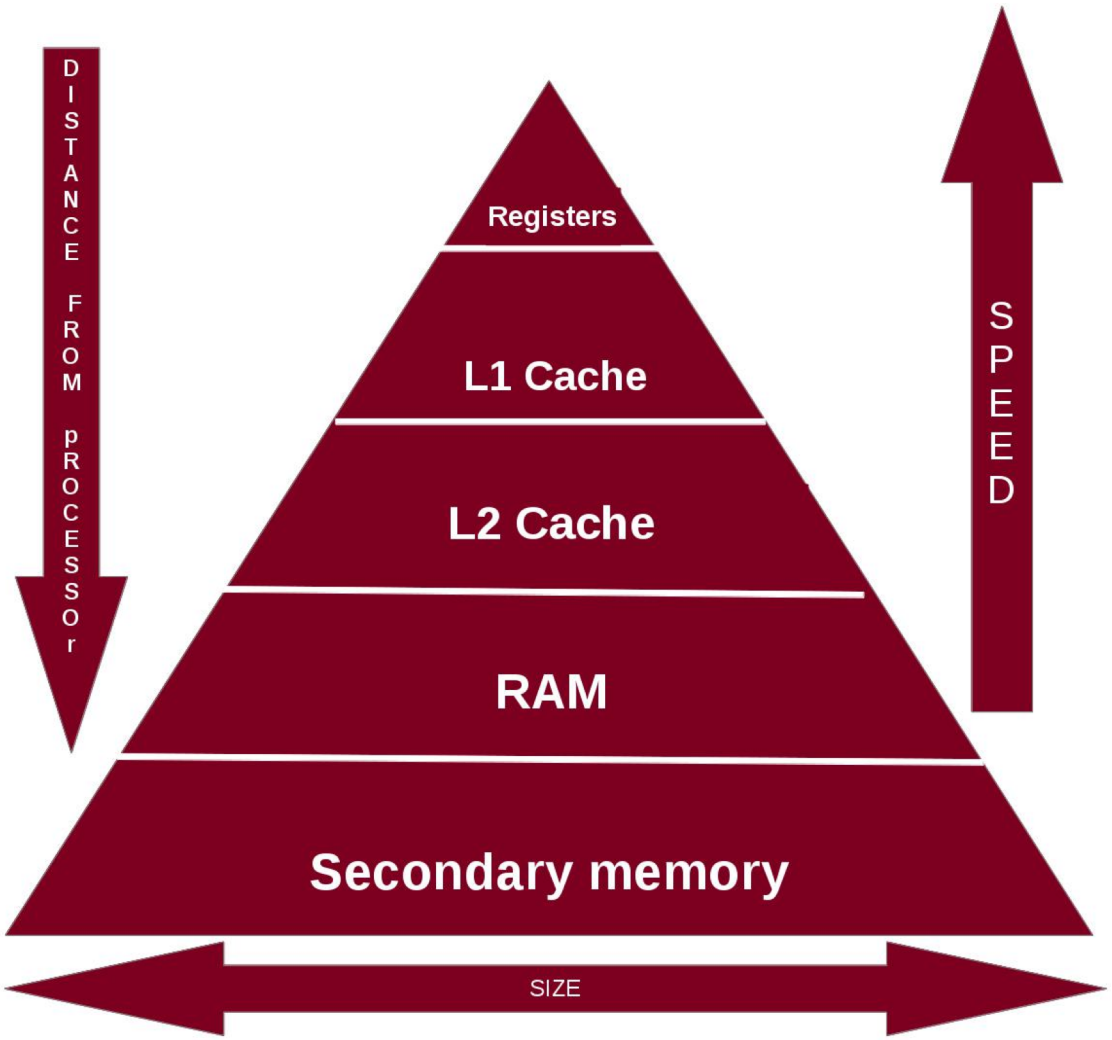
- Vad vill vi ha?
  - Ett minne som får plats med stora program och som fungerar i hastighet som processorn
    - » Fetch – execute (MHz/GHz/Multi-core race)

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

Minne-processor hastighet



Minneshierarki



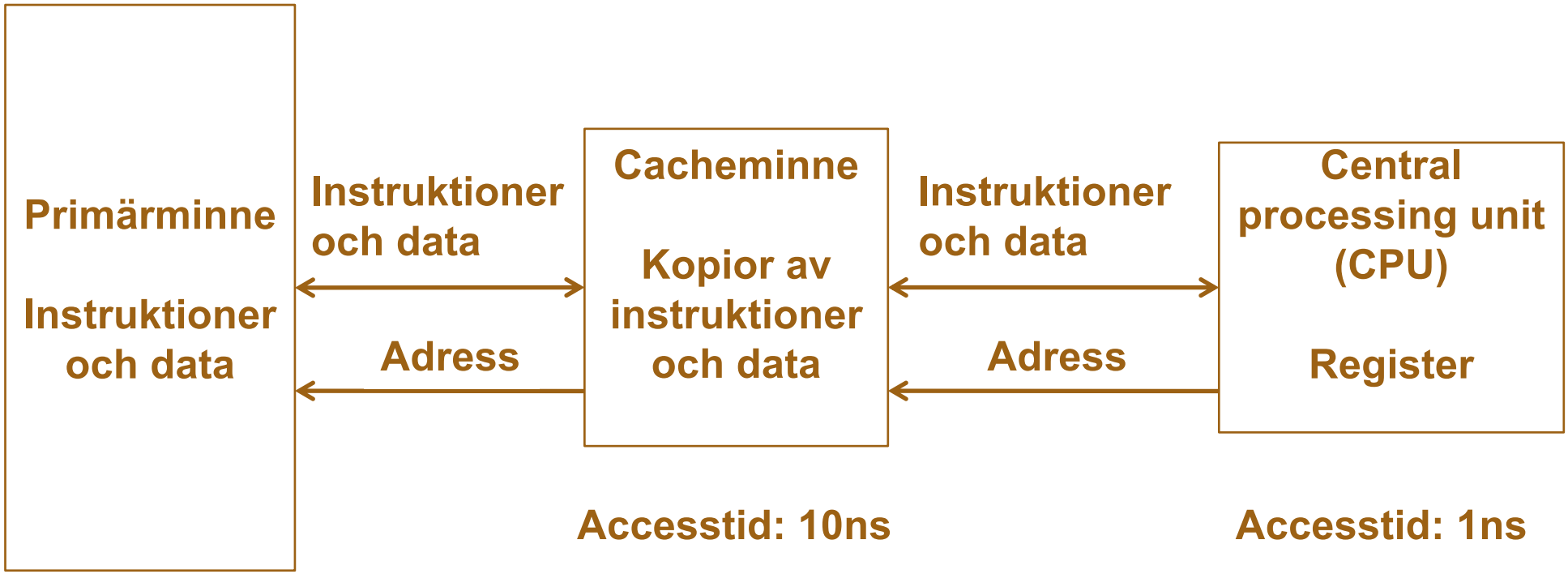
LUNDS UNIVERSITET

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



LUNDS UNIVERSITET

Cacheminne

- Ett cacheminne är mindre och snabbare än primärminnet
  - Hela program får inte plats
  - Men, data och instruktioner ska vara tillgängliga när de behövs
- Om man inte har cacheminne:
  - Accesstid för att hämta en instruktion=100ns
- Om man har cacheminne:
  - Accesstid för att hämta en instruktion från primärminnet=100ns och från cacheminnet=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
```

Utan cache – exempel 2

- Antal instruktioner Instruktioner som exekveras:

1	Instruktion1:while1000
2	Instruktion2:x=x+1
3	Instruktion3:printx
⋮	⋮
2998	Instruktion1:while1000
2999	Instruktion2:x=x+1
3000	Instruktion3:printx
3001	Instruktion4:while500
3002	Instruktion5:y=y+1
3003	Instruktion6:printy
⋮	⋮
4498	Instruktion4:while500
4499	Instruktion5:y=y+1
4500	Instruktion6:printy

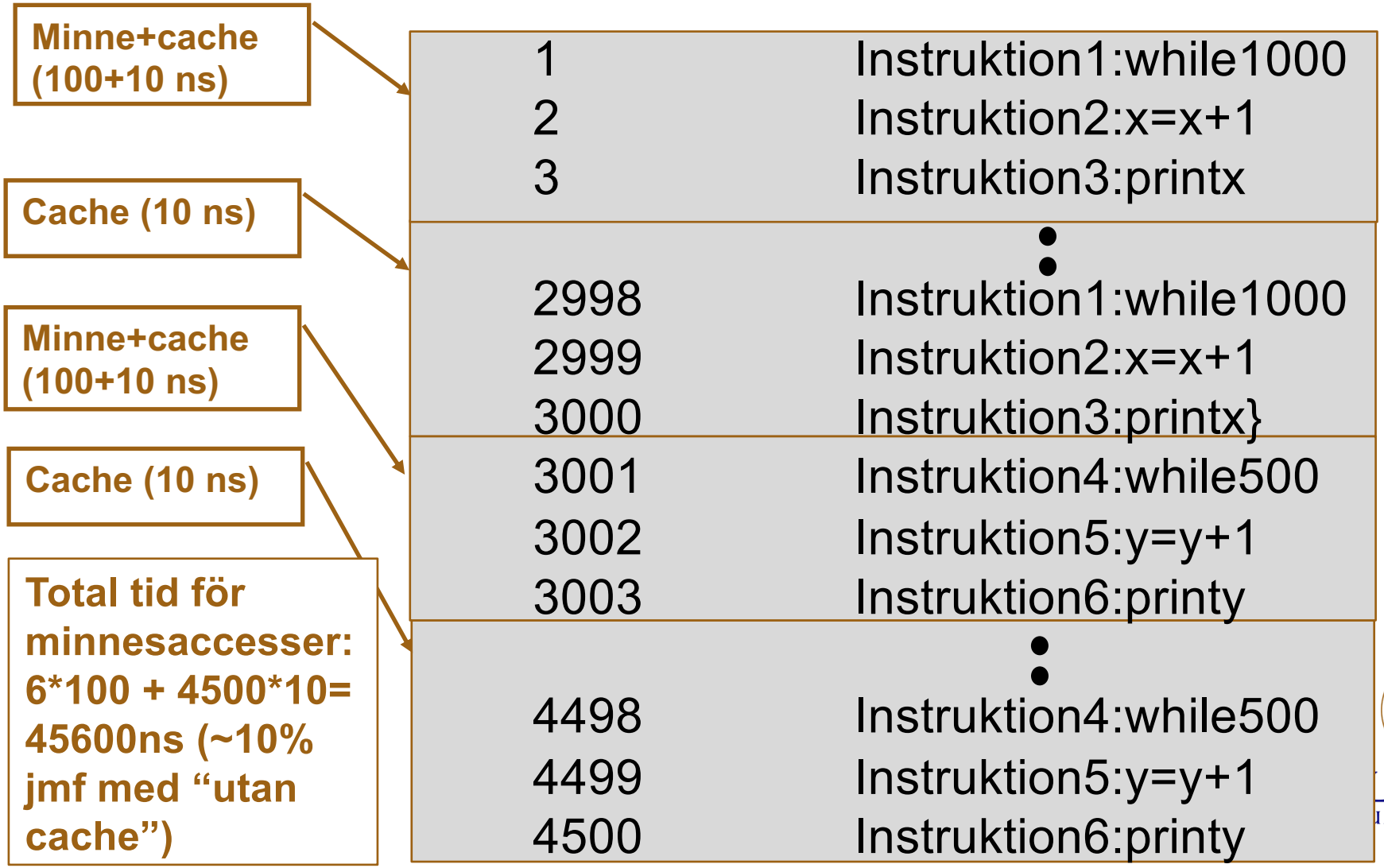
Minnesaccess för 1 instruktion: 100 ns

Totalt 4500 instruktioner.

Tid för minnesaccesser: 4500*100= 450000ns

Med cache – exempel 2

- Antal instruktioner Instruktioner som exekveras:



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 – lokalitet i rum -
    - » om instruktion/data blivit refererat nu, så är sannolikheten stor att instruktioner/data vid adresser i närheten kommer användas inom kort

```
while (x<1000){
     x=x+1;
     …
while (y<500){
     y=y+1;
     …
}
```

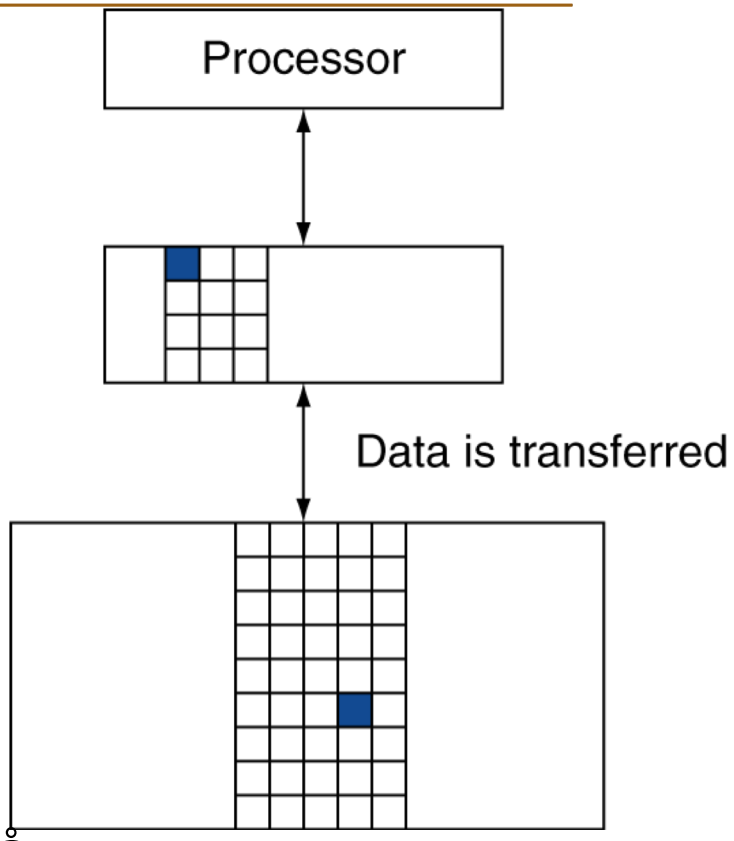
Utnyttja lokalitet

- Minneshierarki
  - Lagra allt på hårddisk
  - Kopiera ”recently accessed (and nearby) items” från disk till mindre primärminne
  - Kopiera mer ”recently accessed (and nearby) items” från primärminne till cacheminne
    - » Cacheminne kopplat till CPU

LUNDS UNIVERSITET

Minneshierarki - nivåer

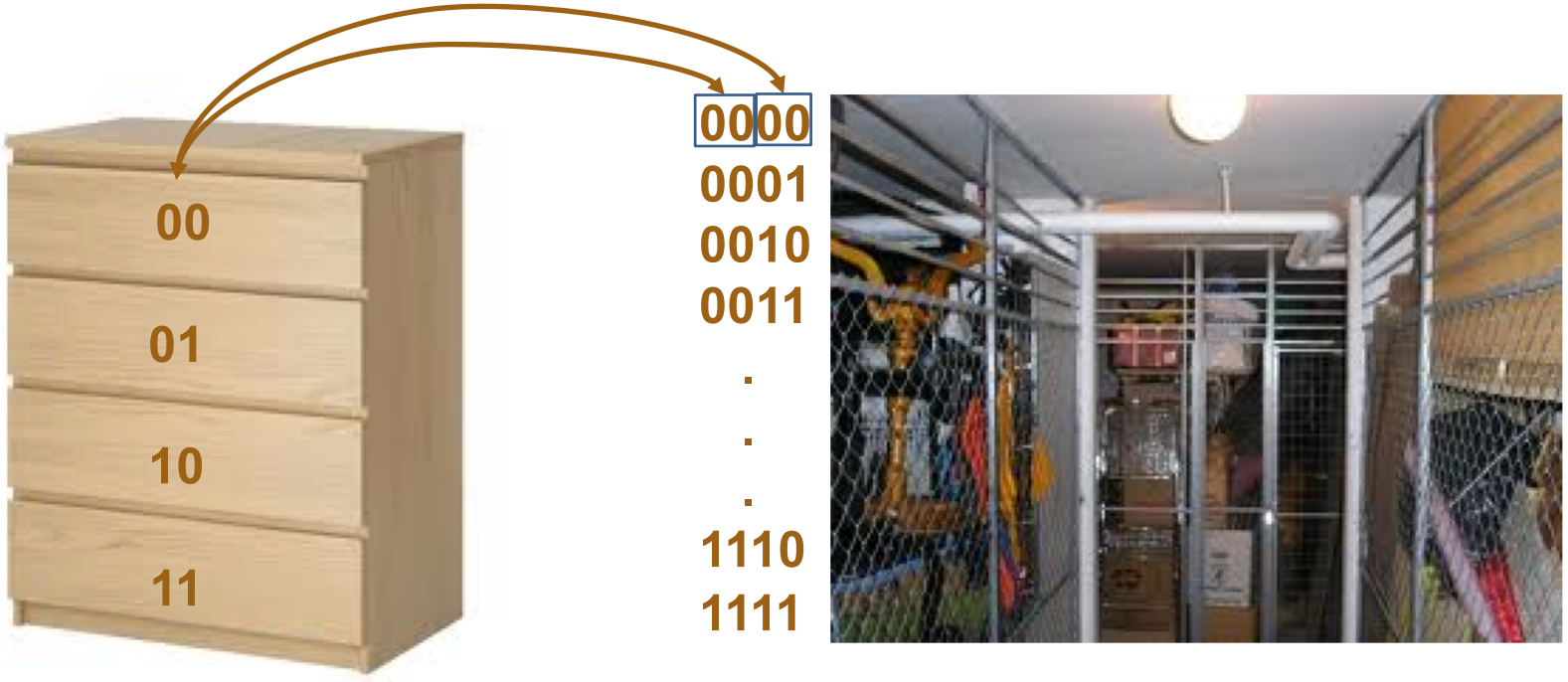
- Block (line): enhet som kopieras
  - Kan vara flera ”words”
- Om ”accessed data” finns i högsta nivån (upper level)
  - Hit: access ges av högsta nivå
    - » Hit ratio: hits/accesses
- Om ”accessed data” inte finns på aktuell nivå
  - Miss: block kopieras från lägre nivå
  - Tid: miss penalty, Miss ratio: antal missar/accesses = 1 – hit ratio
  - Sedan kan data nås från högre nivå



Princip för cacheminne

- **I vilken byrålåda hamnar en låda?**
- **Hur veta vilken låda som finns i en byrålåda?**

- Antag en liten byrå (4 lådor) och ett vindsförråd (16 lådor)
  - Alla lådor är lika stora



LUNDS UNIVERSITET

Exempel: Cacheminne

- Cacheminne med 8 block. 1 ord (word) per block

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Valid data

Rätt data?

Cache line

Exempel: Cacheminne

(1) Processorn läser på adress 22

(2) Data på adress 22 finns ej i cache

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Minnesdata på plats 22

Valid data

(1) Processorn läser på adress 26

Exempel: Cacheminne

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

LUNDS UNIVERSITET

Exempel: Cacheminne

(1) Processorn läser på adress 22 - hit

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

(2) Processorn läser på adress 26 - hit

1

2

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

LUNDS UNIVERSITET

Exempel: Cacheminne

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

LUNDS UNIVERSITET

Exempel: Cacheminne

(1) Processorn läser på adress 18 – miss och annan data fanns där

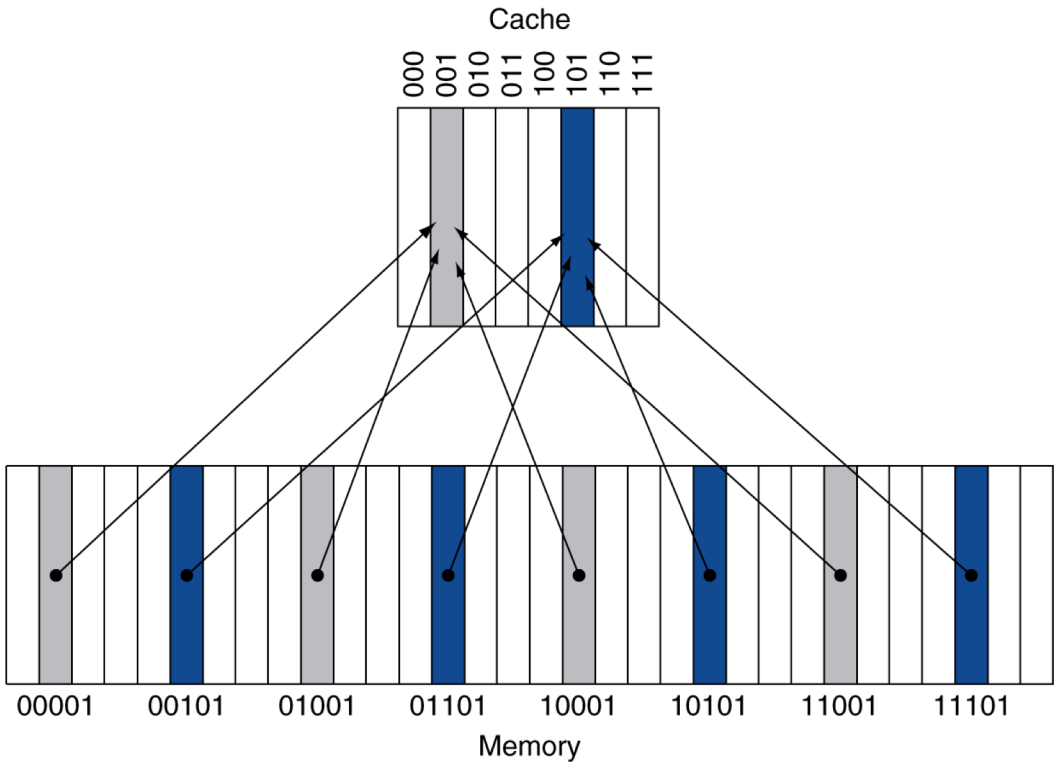
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10 (11)**	**Mem[10010] (Mem[11010])**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

LUNDS UNIVERSITET

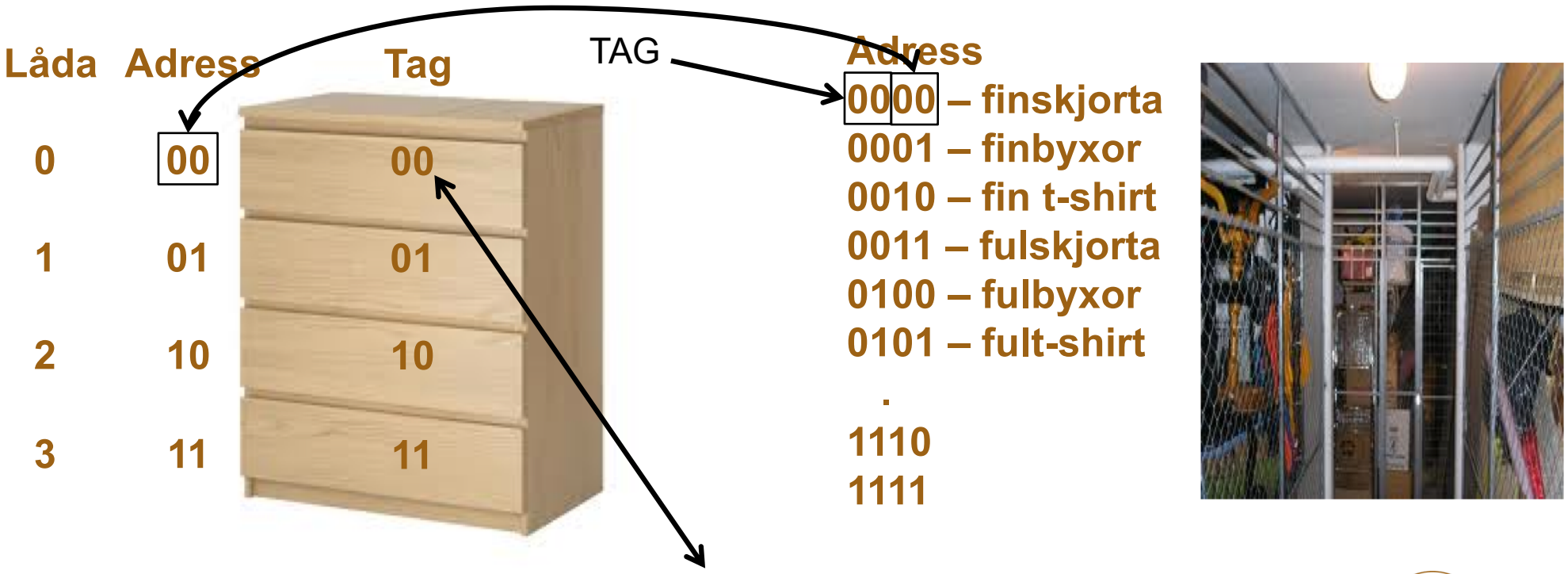
Direktmappad cache

- Primärminnet är mycket större än cacheminnet
- Direktmappning – data i cache på ett ställe



LUNDS UNIVERSITET

Cacheminne (Direktmappning)

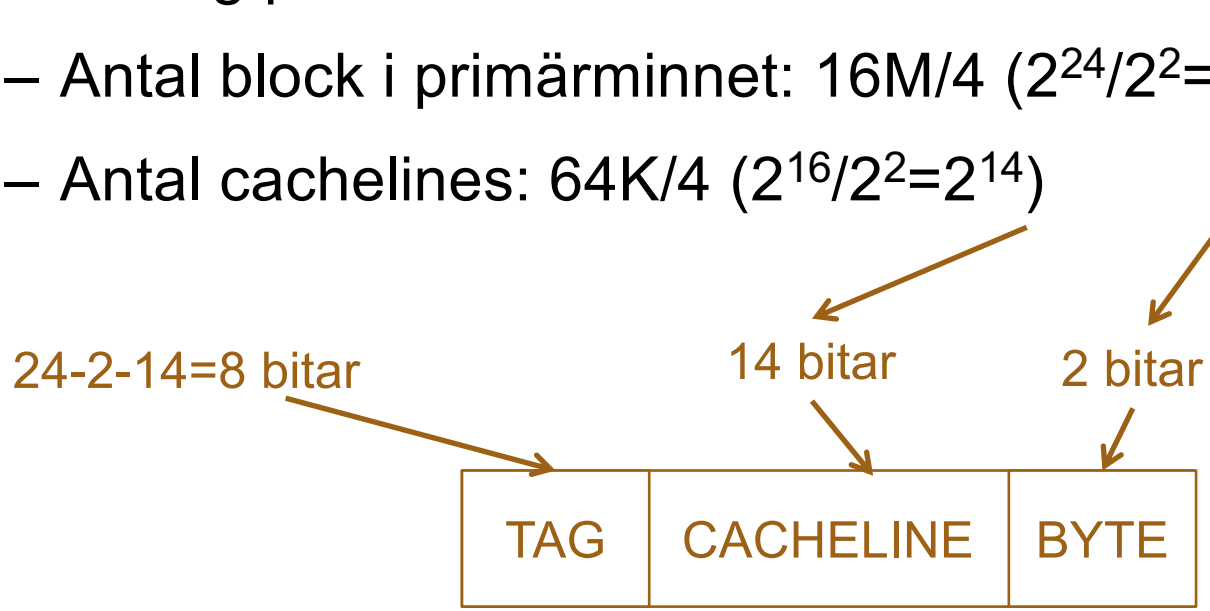


I låda 00 med TAG=00 ligger där finskjorta eller fulbyxor?

LUNDS UNIVERSITET

Cacheminne - direktmappning

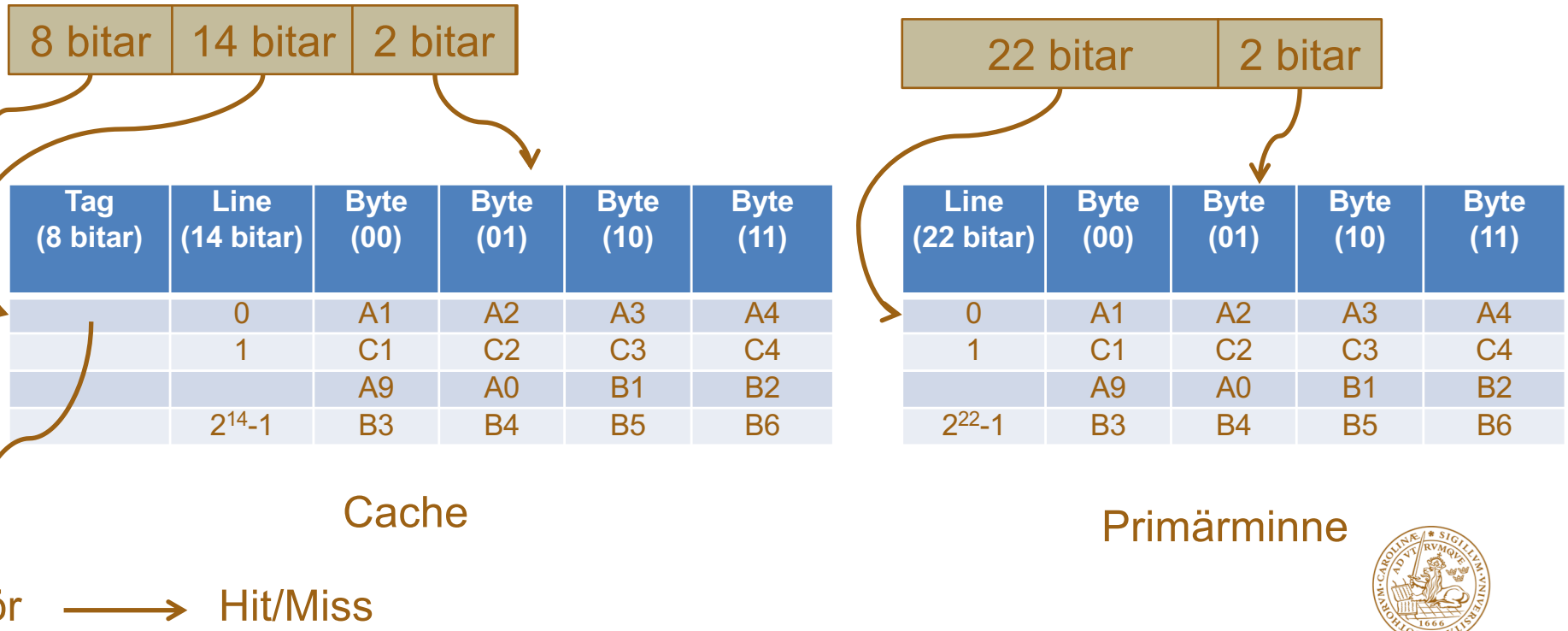
- Cache: 64K (2^{16}) bytes
- Primärminne: 16M (2^{24}) bytes
  - Adressrymd: 24 bitar
- Överföring primärminne och cache i block om 4 (2^2) bytes
  - Antal block i primärminnet: 16M/4 ($2^{24}/2^2=2^{22}$)
  - Antal cachelines: 64K/4 ($2^{16}/2^2=2^{14}$)



LUNDS UNIVERSITET

Cacheminne – direktmappning

- Adress: 24 bitar



Tag (8 bitar)	Line (14 bitar)	Byte (00)	Byte (01)	Byte (10)	Byte (11)
	0	A1	A2	A3	A4
	1	C1	C2	C3	C4
		A9	A0	B1	B2
	2^{14}-1	B3	B4	B5	B6

Cache

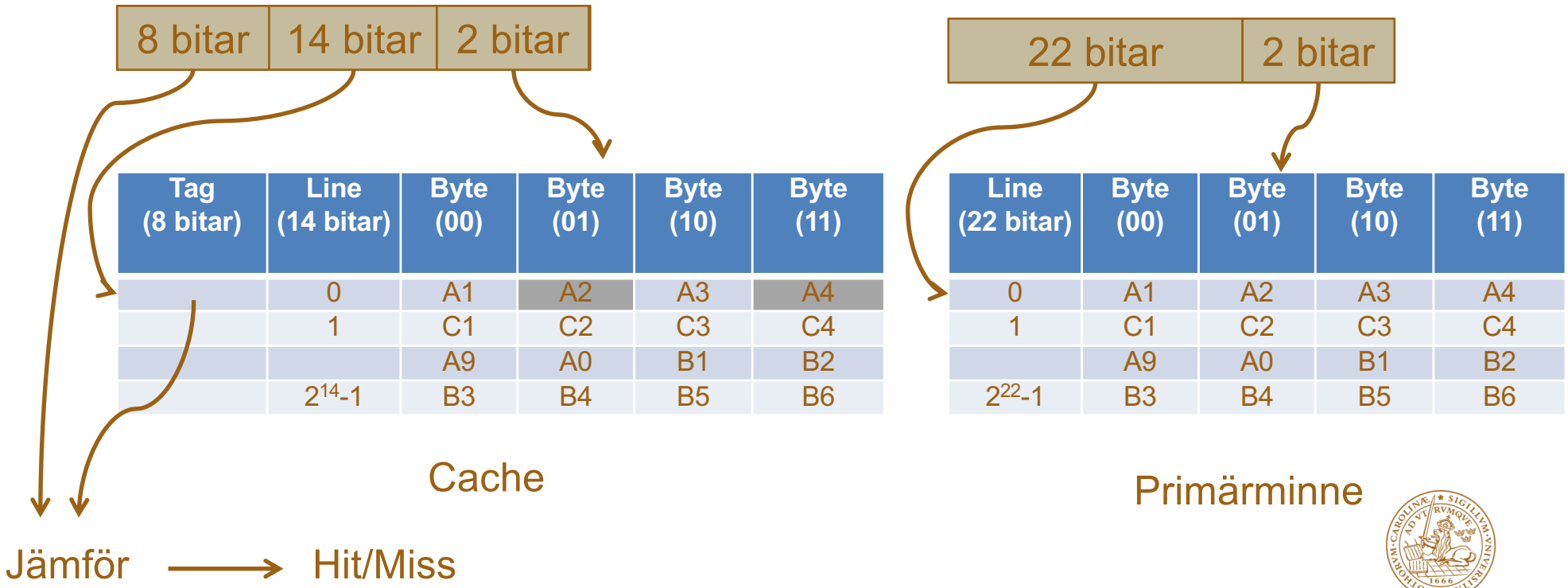
Line (22 bitar)	Byte (00)	Byte (01)	Byte (10)	Byte (11)
0	A1	A2	A3	A4
1	C1	C2	C3	C4
	A9	A0	B1	B2
2^{22}-1	B3	B4	B5	B6

Primärminne

LUNDS UNIVERSITET

Cacheminne – direktmappning

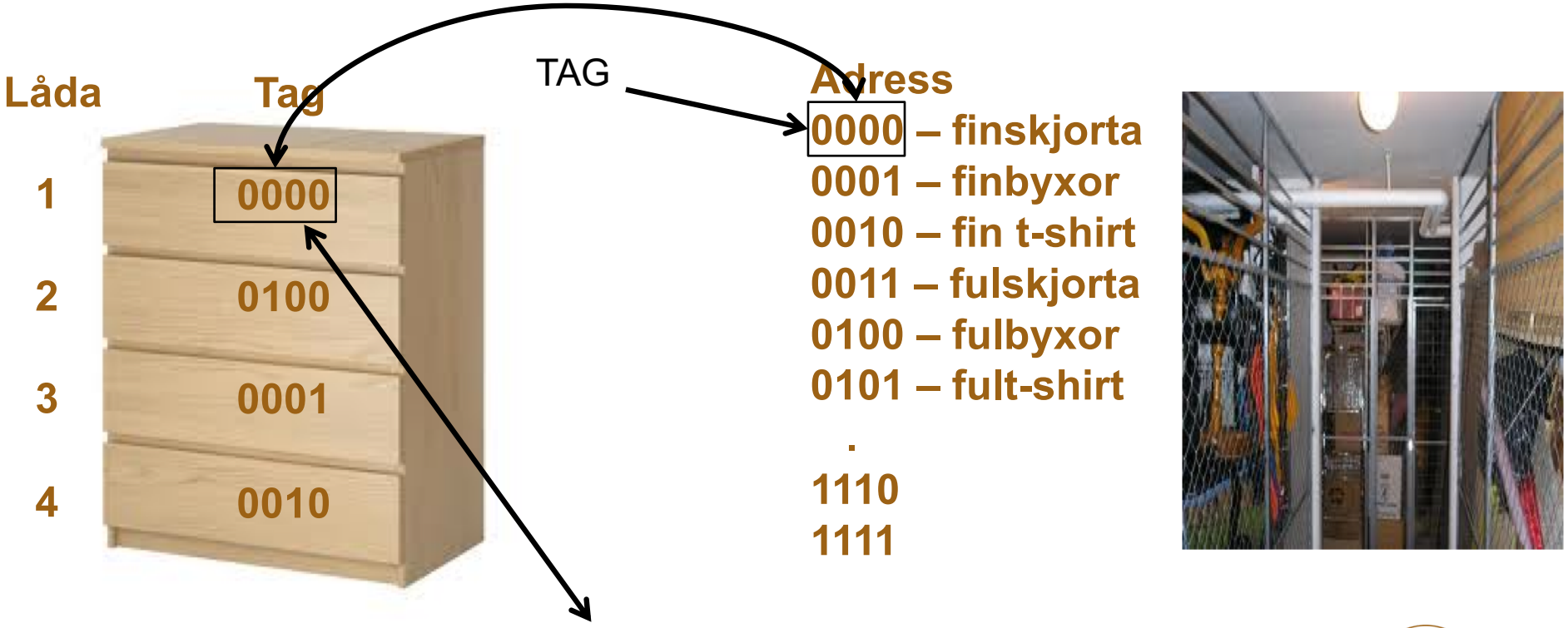
- Exempel: Har läst “A2”. Vill läsa A4 – hit/miss?



LUNDS UNIVERSITET

Cacheminne (Associativmappning)

Kan hamna i vilken låda som helst

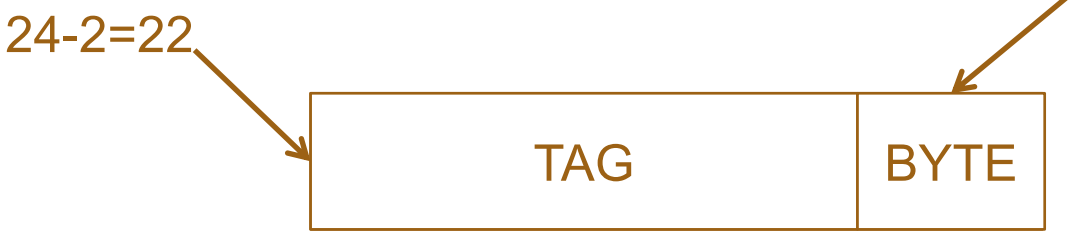


I låda 1 med TAG=0000 ligger där en fin skjorta eller ful byxor?

LUNDS UNIVERSITET

Cacheminne - associative mapping

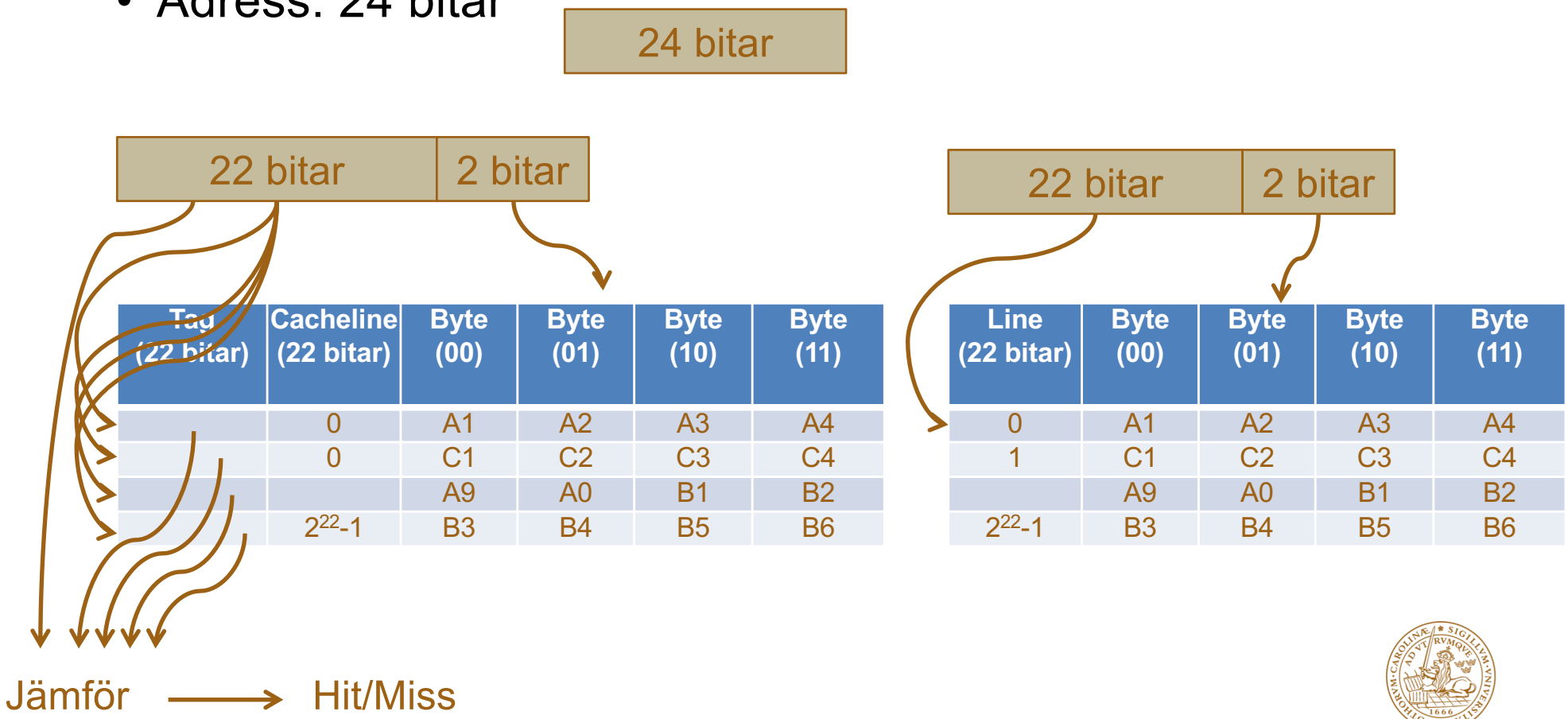
- Cache: 64K (2^{16}) bytes
- Primärminne: 16M (2^{24}) bytes
  - Adressrymd: 24 bitar
- Överföring primärminne och cache i block om 4 (2^2) bytes
- Antal cachelines: 64K/4 ($2^{16}/2^2=2^{14}$)
- Antal block i minnet: 16M/4 ($2^{24}/2^2=2^{22}$)



LUNDS UNIVERSITET

Cacheminne – associative mappning

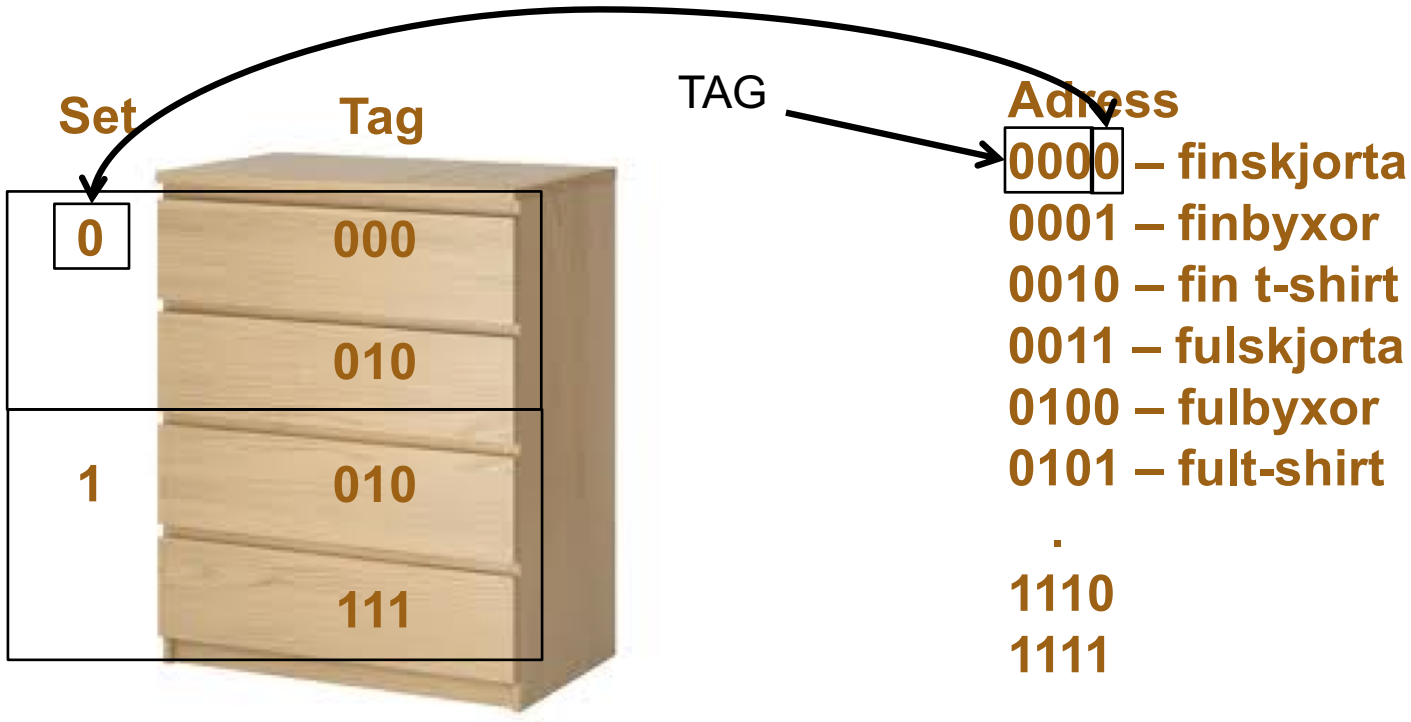
- Adress: 24 bitar



LUNDS UNIVERSITET

Cacheminne (2-vägs set associative)

Hamnar i denna låda



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

LUNDS UNIVERSITET

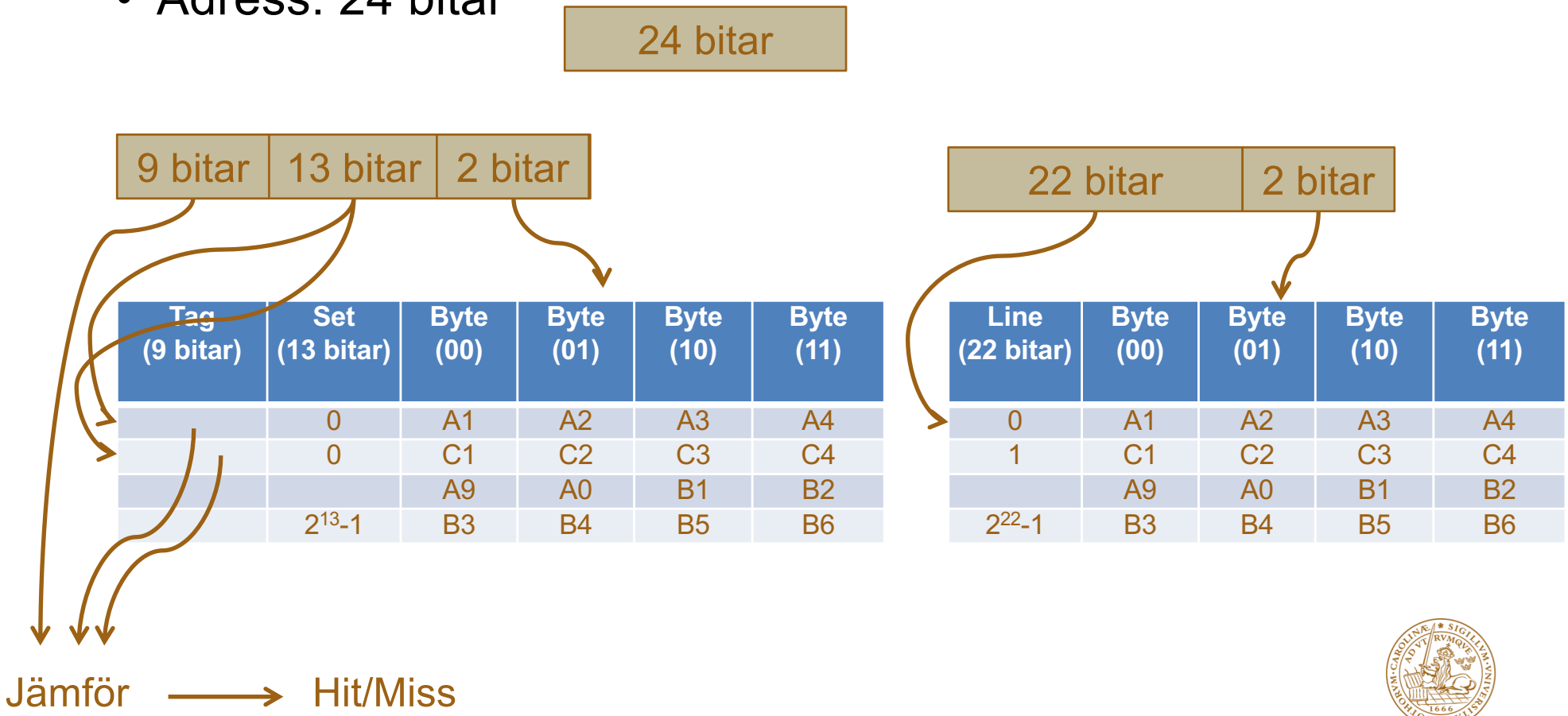
Cacheminne (2-way set associative)

- Primärminne: 16M (2^{24}) bytes
  - Adressrymd: 24 bitar
- Cache: 64K (2^{16}) bytes
- Överföring primärminne och cache och i block om 4 (2^2) bytes
- 2-way associative mapping
  - Antal block i minnet: 16M/4 ($2^{24}/2^2=2^{22}$)
  - Antal set: 64K/4 ($2^{16}/(2^{2*}2)=2^{13}$)

LUNDS UNIVERSITET

Cacheminne – set associative mappning

- Adress: 24 bitar

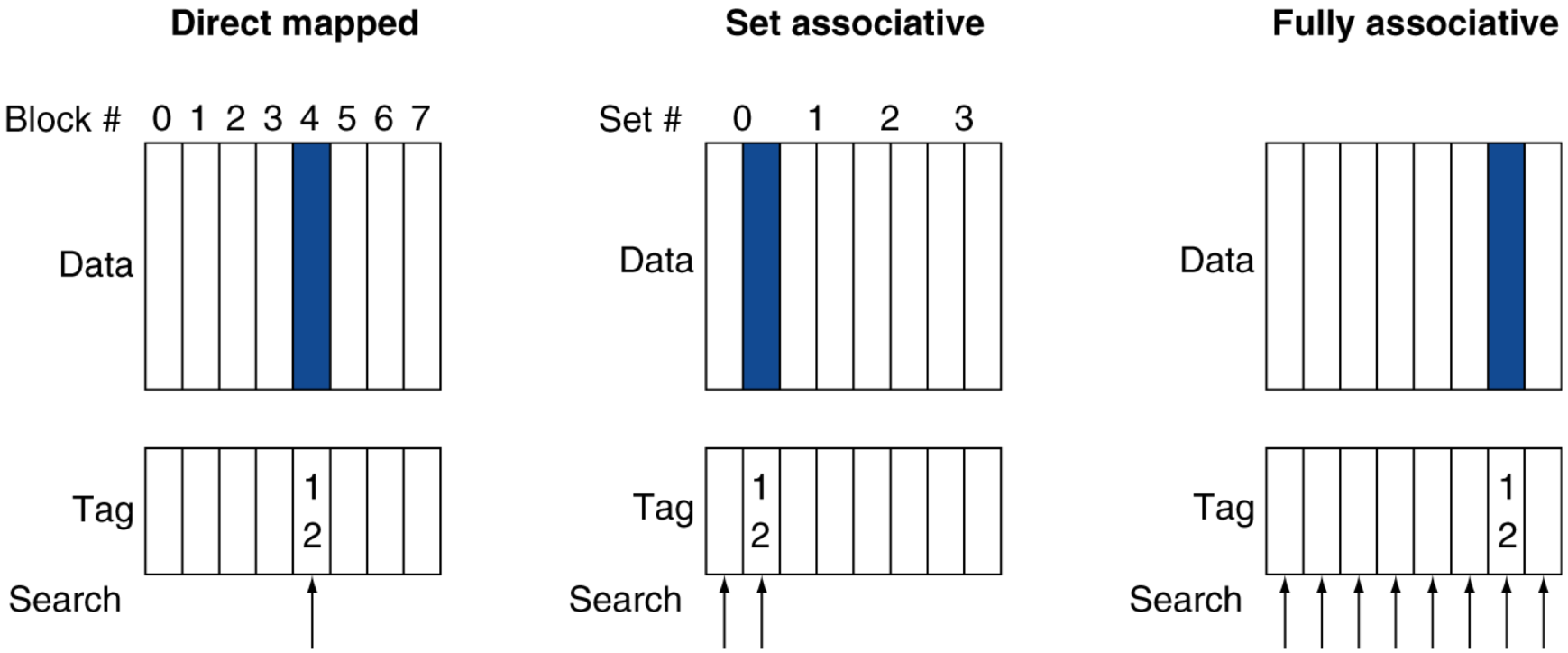


Tag (9 bitar)	Set (13 bitar)	Byte (00)	Byte (01)	Byte (10)	Byte (11)
	0	A1	A2	A3	A4
	0	C1	C2	C3	C4
		A9	A0	B1	B2
	2^{13}-1	B3	B4	B5	B6

Line (22 bitar)	Byte (00)	Byte (01)	Byte (10)	Byte (11)
0	A1	A2	A3	A4
1	C1	C2	C3	C4
	A9	A0	B1	B2
2^{22}-1	B3	B4	B5	B6

LUNDS UNIVERSITET

Cacheminne



LUNDS UNIVERSITET

Cacheminne

- Antag 32 bytes primärminne och 16 bytes cacheminne
- En adress*: tag:cache line:byte*
- En cache line: de antal byte som läses samtidigt
- Tag: avgör om "rätt" data finns i cacheminnet
- CPU vill läsa på adressen: 00000

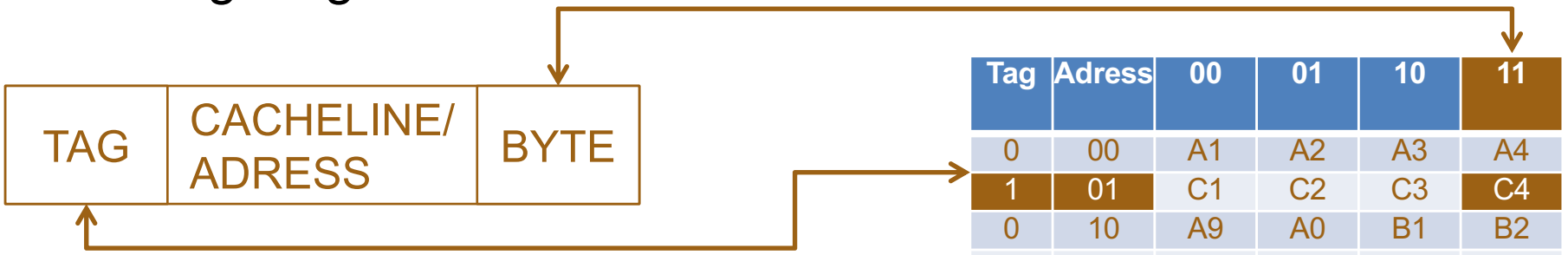
- Plocka ut TAG: 0
- Plocka ut adress: 0000
- Titta i cache på adress: 0000 och jämför tag. Om samma – hit. Annars: miss

Tag	Adress	Data
0	0000	A1
0	0001	A2
0	0010	A3
0	0011	A4
0	0100	A5
0	0101	A6
0	0110	A7
0	0111	A8
0	1000	B1
0	1001	B2
0	1010	B3
0	1011	B4
0	1100	B5
0	1101	B6
0	1110	B7
0	1111	B8

Cacheminne - direktmappning

- Antag cache lines (blocks) om 4 bytes

1
  - "BYTE" = 2 bitar
  - Tag: avgör om "rätt" data finns i cacheminnet



Tag	Adress	00	01	10	11
0	00	A1	A2	A3	A4
1	01	C1	C2	C3	C4
0	10	A9	A0	B1	B2
0	11	B3	B4	B5	B6

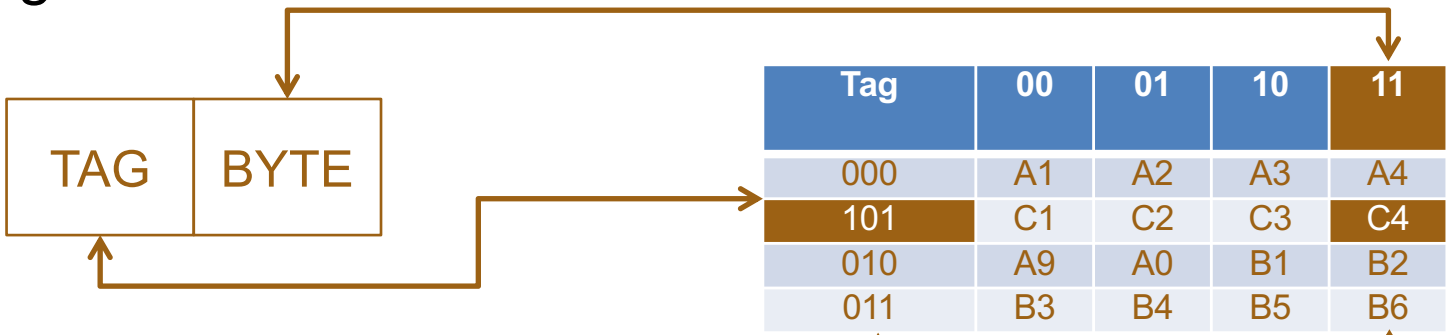
2
- CPU vill läsa på adressen: 10111
- Titta på adress: 01
- Jämför TAG: 1
- Titta i cache på adress: 01, byte: 11 - datan är: C4

LUNDS UNIVERSITET

Cacheminne – fully associative

- Antag cache lines (blocks) om 4 bytes

1
  - "BYTE" = 2 bitar
  - Tag: avgör om "rätt" data finns i cacheminnet



2
- CPU vill läsa på adressen: 10111
- Plocka ut TAG och se om den finns: 101
- TAG jämförelse ger hit: Titta i byte: 11 och datan är: C4

LUNDS UNIVERSITET

Cacheminne – 2-way set associative

- Antag cache lines (blocks) om 4 bytes

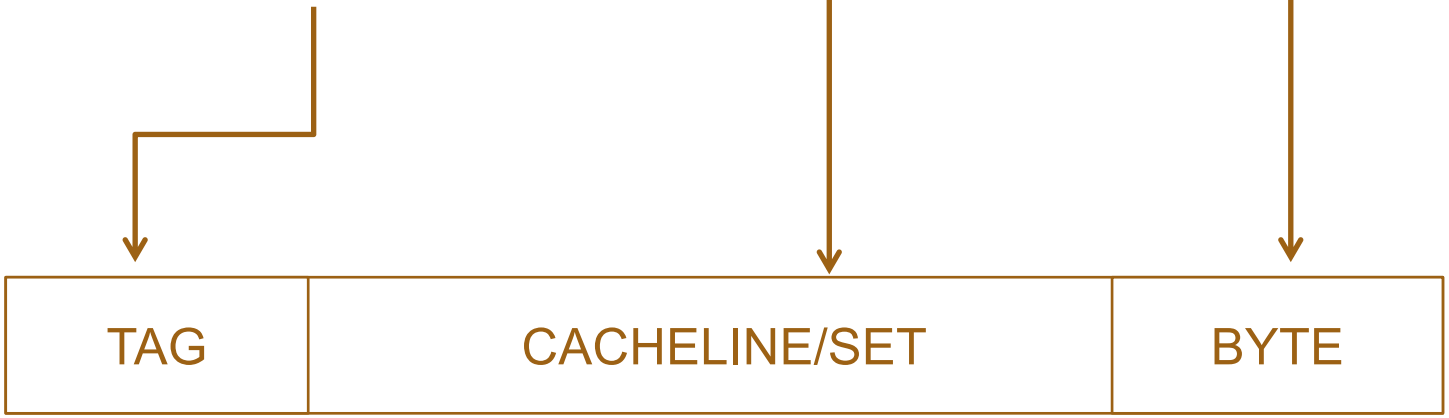
1
  - "BYTE" = 2 bitar
  - Tag: avgör om "rätt" data finns i cacheminnet

Tag	Set	00	01	10	11
00	0	A1	A2	A3	A4
00	0	C1	C2	C3	C4
10	1	A9	A0	B1	B2
00	1	B3	B4	B5	B6

2
- CPU vill läsa på adressen: 10111
- Plocka ut TAG: 10
- Plocka ut set: 1
- Titta i set 1 och jämför tag. Hit! Data finns i byte 11: B2

Cacheminne

- Adressrymd (antal bitar) givet
- Storlek på cacheline (cacherad)
- Antal cacherader/set
- Resten, används som “TAG”

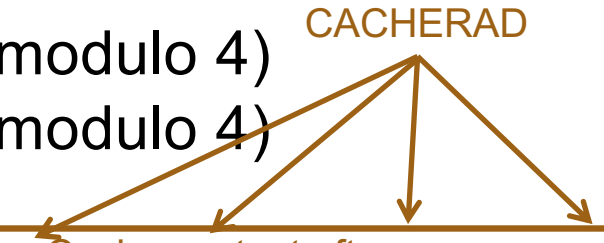


LUNDS UNIVERSITET

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)



TID

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]	

LUNDS UNIVERSITET

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]	

Design av cache

- Om cachemiss, hur välja cacherad som ska ersättas?
- Hur hålla minnet konsistent(skrivstrategi)?
- Hur många cacheminnen?
  - Nivåer - Levels (L1, L2, L3)
    - » större cache ger högre hit-rate men är långsammare
  - Unifierad eller separata cacheminnen för instruktioner och data

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

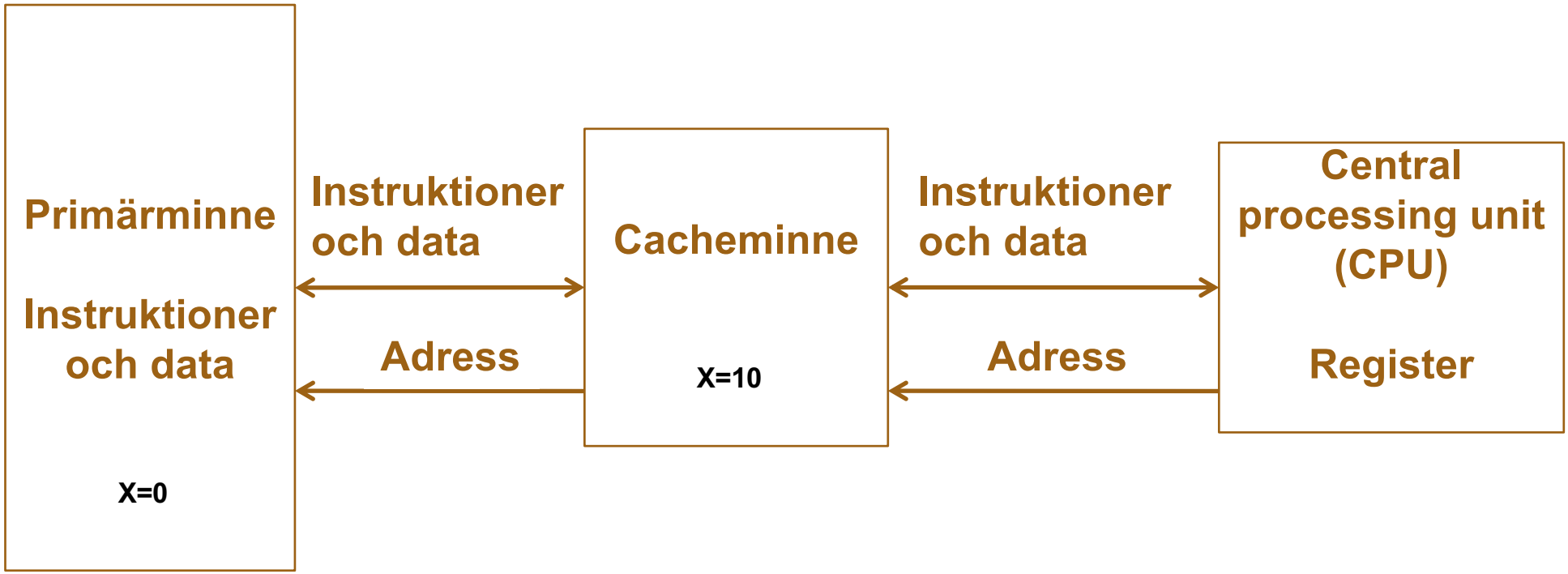
Skrivstrategier

- Problem: håll minnet konsistent
- Exempel:

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

- Variablen x kommer "finnas" i primärminnet och i cacheminnet
- I primärminnet är x=0 medan i cacheminnet är x=0,1,2…och till sist 1000

Cacheminne



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)

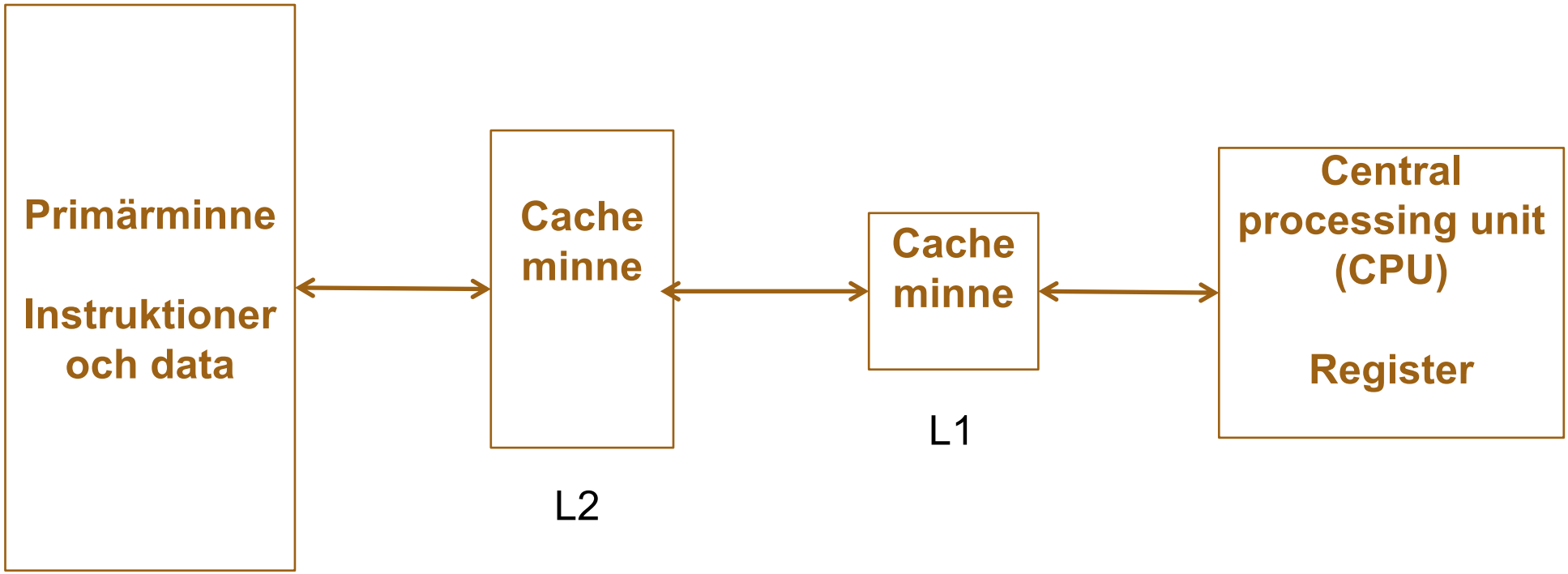
LUNDS UNIVERSITET

Skrivstrategier

- Skilj på write-hit och write-miss
  - Write-hit: se ovan
  - 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)

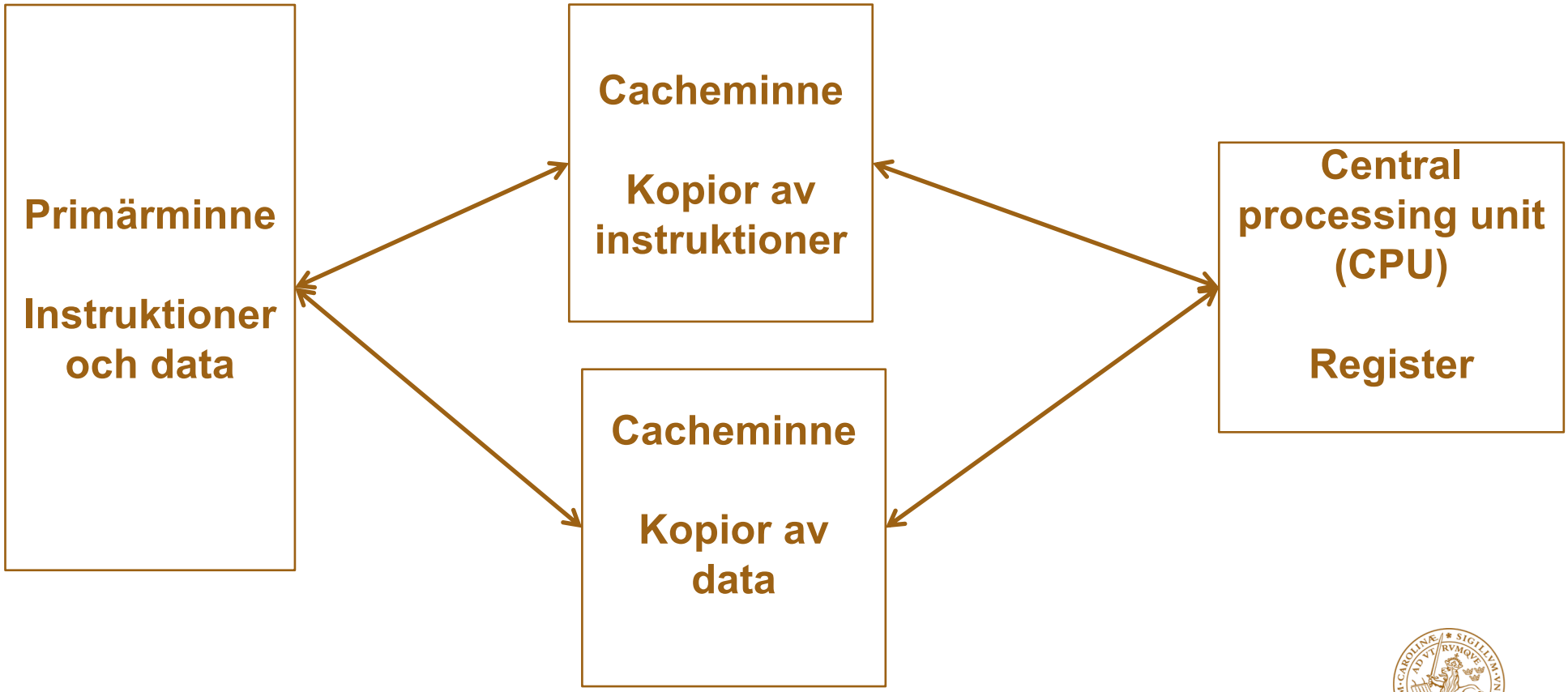


Antal cachenivåer (levels)

- Level 1: närmst CPU. Litet och snabbt
- Level 2: större men långsammare (jämfört med Level 1 cache). Level 2 cache “stödjer” missar i Level 1 cache.
- Level 3: större men långsammare (jämfört med Level 2 cache). Level 3 cache “stödjer” missar i Level 2 cache.
- Primärminne: Störst men långsammast.

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

- 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
  - Det är 2 klockcykler per instruktion

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

Med perfekt cache (alltid hit), så här snabbt går processorn

Antag att bara load och store används för access till minnet

Tid för verklig processor

LUNDS UNIVERSITET

Prestanda – multilevel cache

Med perfekt cache (alltid hit), så här snabbt går processorn

- Givet:
  - CPU med CPI=1, klockfrekvens = 4GHz (0.25 ns)
  - Miss rate/instruktion = 2%
  - Accesstid till primärminnet=100ns

Så här mycket kostar en miss

- Med 1 cache nivå (L1)
  - Miss penalty = 100ns/0.25ns=400 cykler
- Effektiv CPI=1+0.02*400=9

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

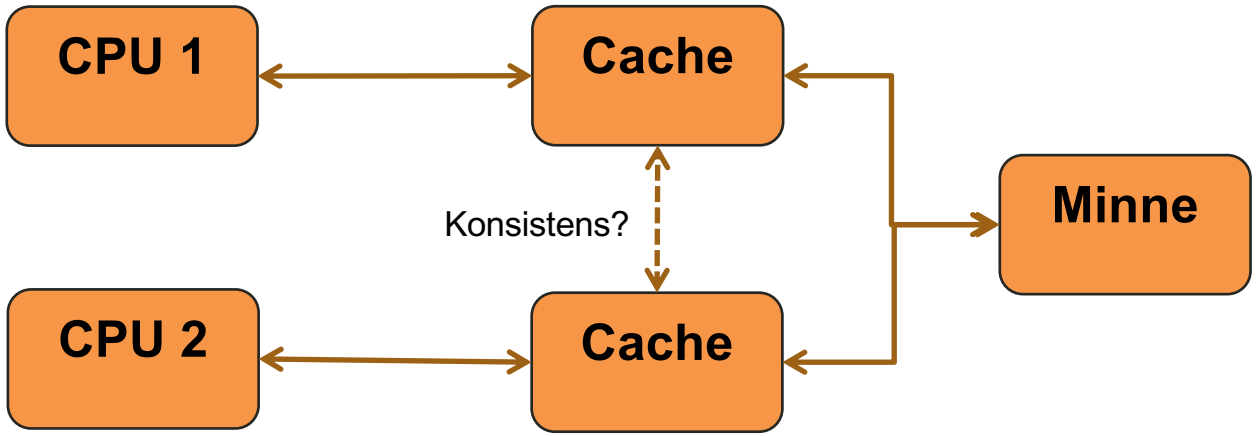
Förra slide

LUNDS UNIVERSITET

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

Cache coherency



LUNDS UNIVERSITET

Cache coherency - problem

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

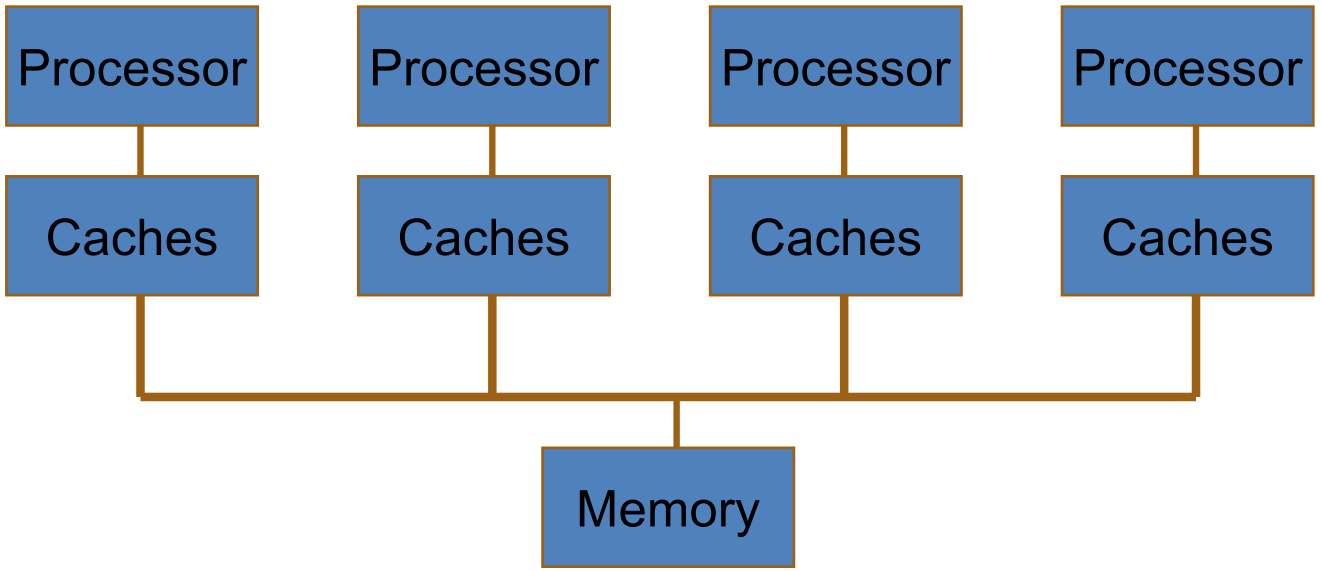
Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

LUNDS UNIVERSITET

Snooping-Based Cache Coherence

Gemensam buss ger serialisering

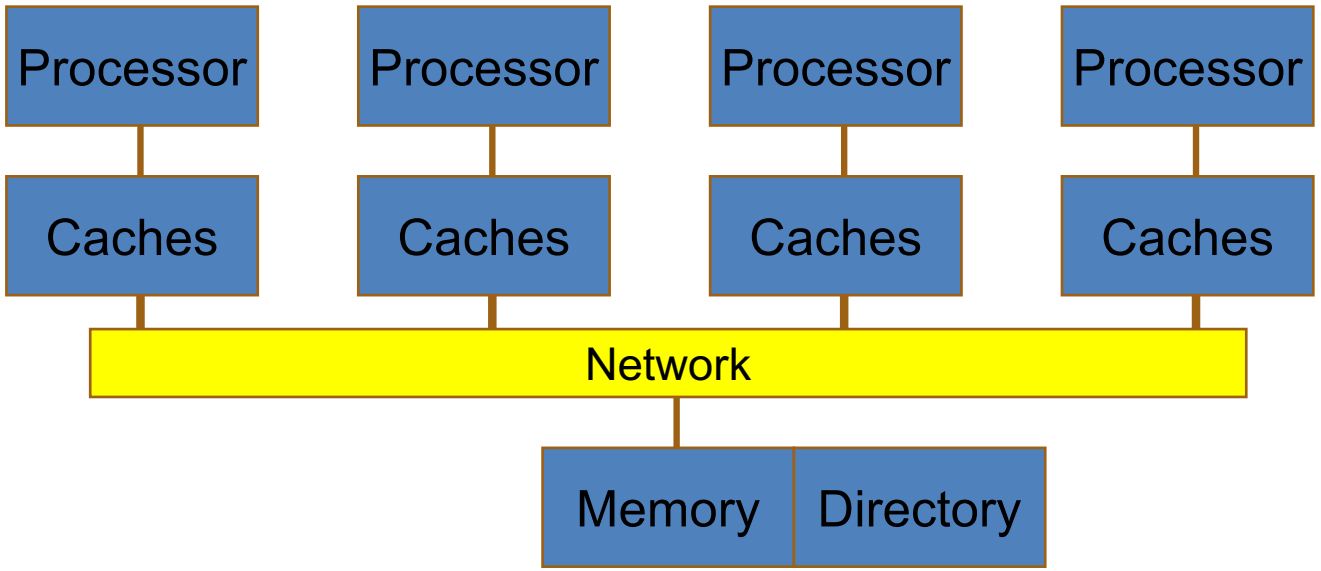
- Cacheminnen delar buss; varje cache ser alla transaktioner samtidigt; varje cache “hanterar” sig själv
- När ett cacheminne skriver (writes), markerar alla andra cacheminnet att blocket är “invalid”
- När ett cacheminne läser, fås blocket från minnet eller den som skrev senast
- Protokoll: MSI, MESI, MOESI



LUNDS UNIVERSITET

Directory-Based Cache Coherence

- Ett bibliotek håller ordning på delningsstatus för block
- Varje läsning/skrivning går till biblioteket som skickar direktiv till varje cache – biblioteket är den som serialiserar (som bussen är serialisering i snooping)
- Till exempel, vid skrivning, när önskemålet når biblioteket skickas “invalidates” till “sharers” och tillstånd (permission) till som som ska skriva.



LUNDS UNIVERSITET

Exempel

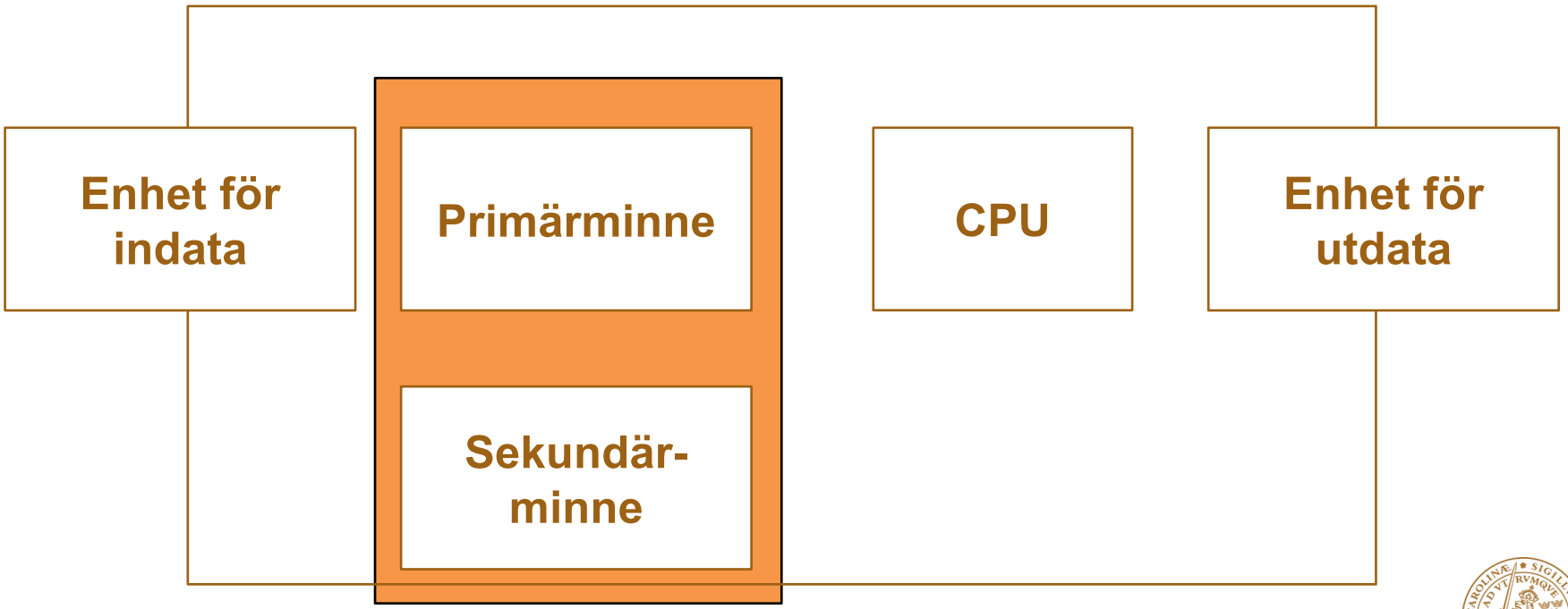
- Intel 80486 – (1989)
  - a single on-chip cache of 8 Kbytes, line size: 16 bytes, 4-way set associative organization
- Pentium – (1993)
  - two on-chip caches, for data and instructions, each cache: 8 Kbytes, line size: 32 bytes (64 bytes in Pentium 4), 2-way set associative organization, (4-way in Pentium 4)
- PowerPC 601-Introduced 1993
  - a single on-chip cache of 32 Kbytes, line size: 32 bytes, 8-way set associative organization

LUNDS UNIVERSITET

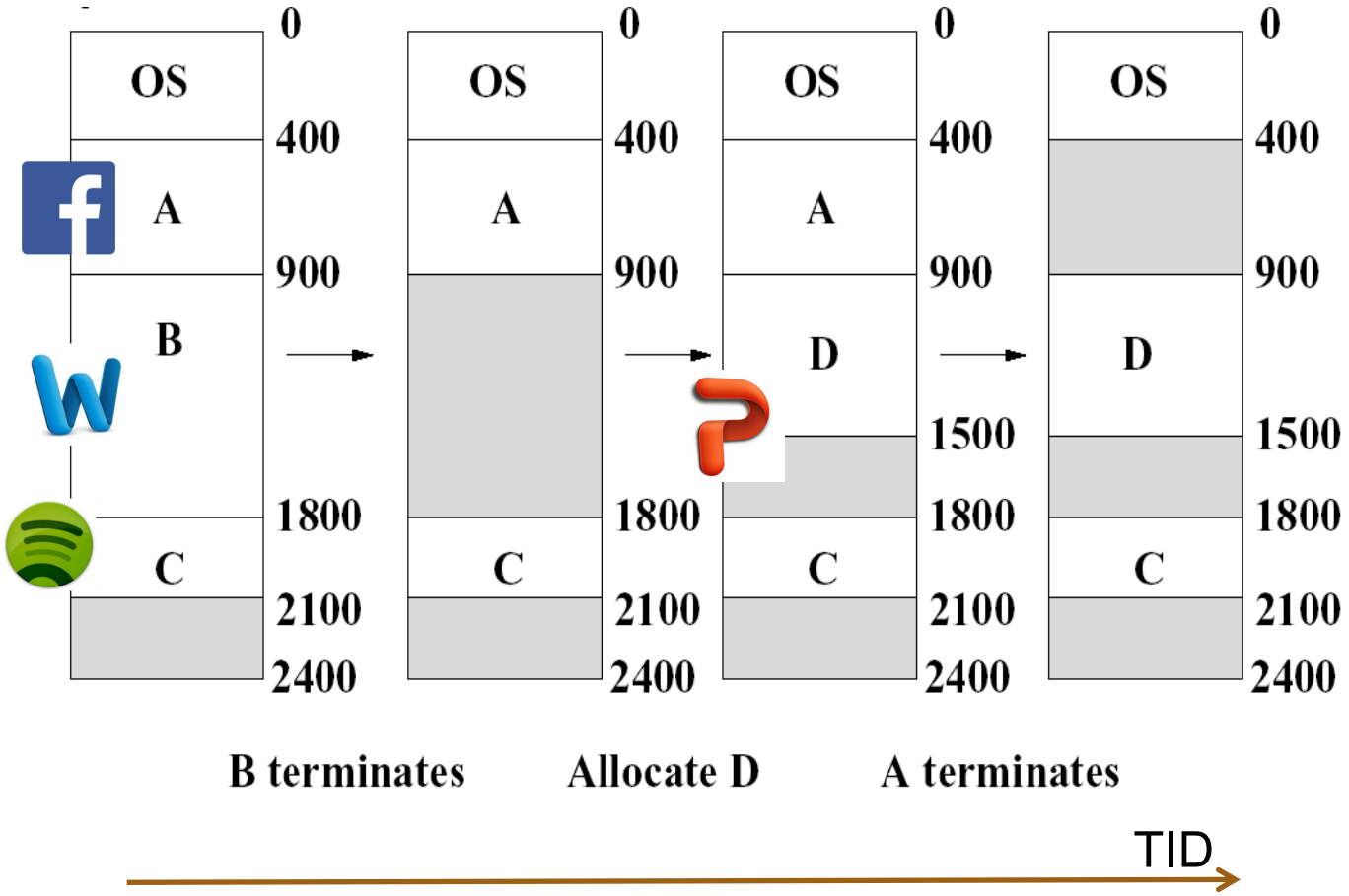
Exempel

- PowerPC 603
  - two on-chip caches, for data and instructions, each cache: 8 Kbytes, line size: 32 bytes, 2-way set associative organization, (simpler cache organization than the 601 but stronger processor)
- PowerPC 604
  - two on-chip caches, for data and instructions, each cache: 16 Kbytes, line size: 32 bytes, 4-way set associative organization
- PowerPC 620
  - two on-chip caches, for data and instructions, each cache: 32 Kbytes, line size: 64 bytes, 8-way set associative organization

Minnets komponenter



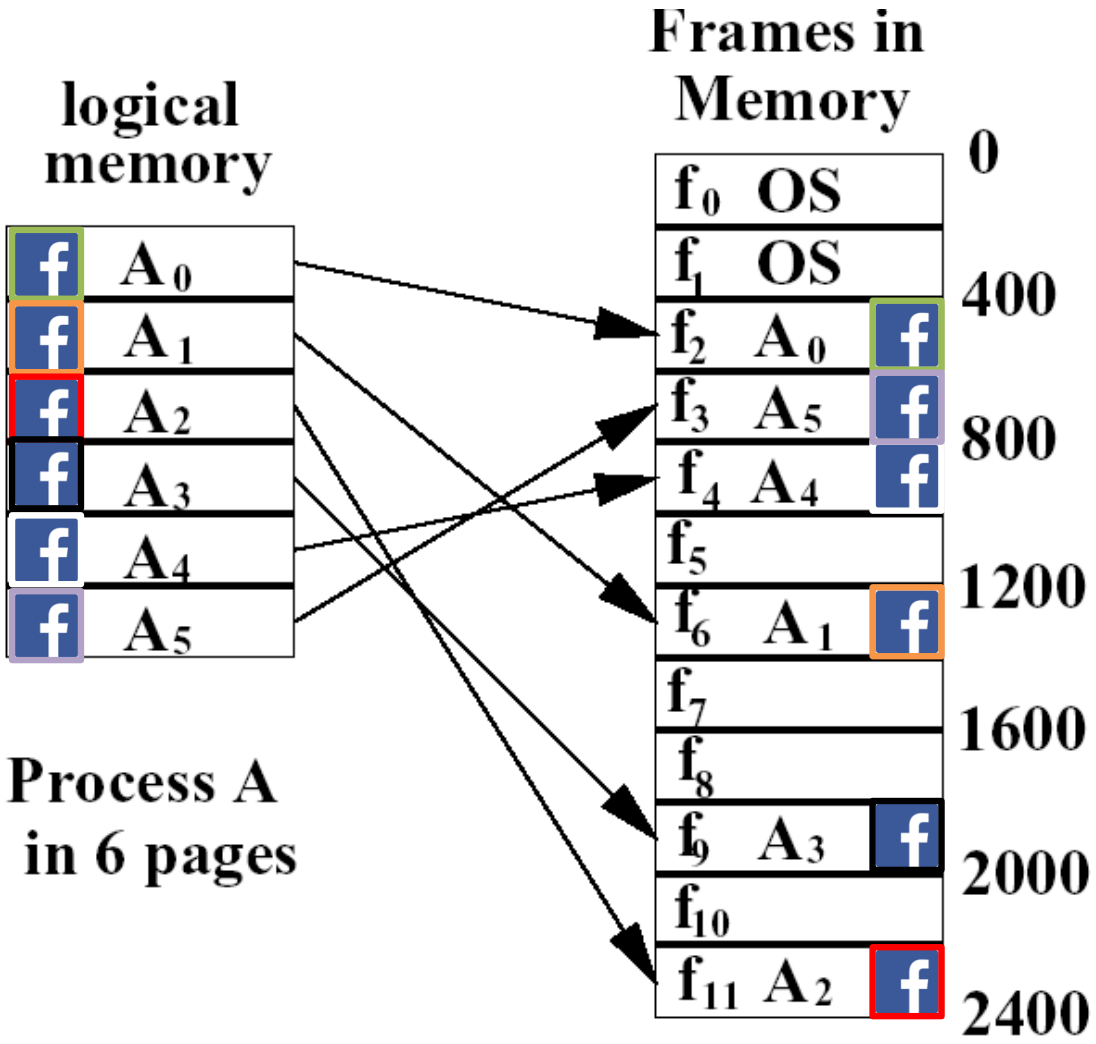
Primärminnets innehåll över tiden



LUNDS UNIVERSITET

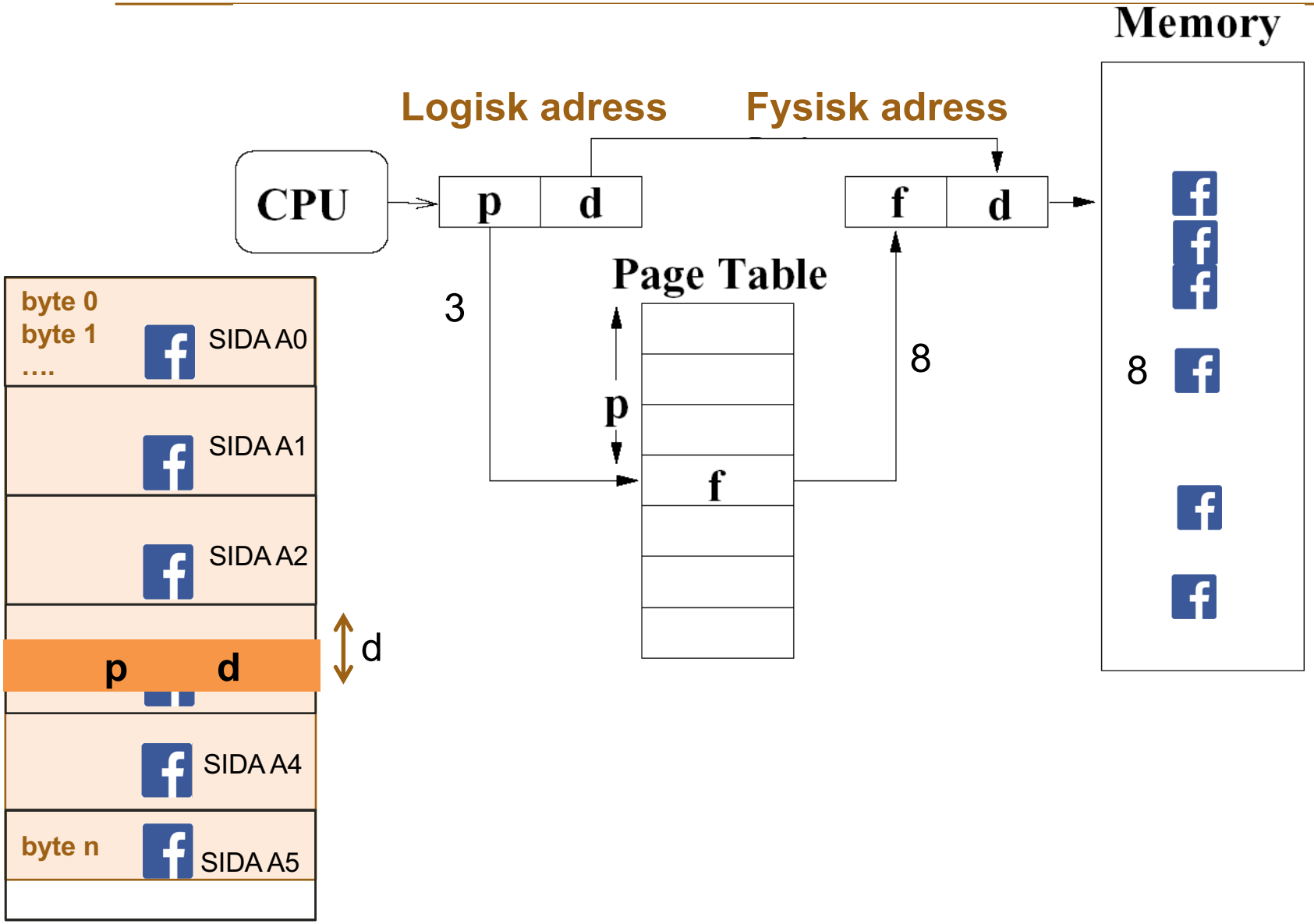
Paging

Dela upp primärminnet i ramar (frames) och program i sidor (pages)



LUNDS UNIVERSITET

Paging



LUNDS UNIVERSITET

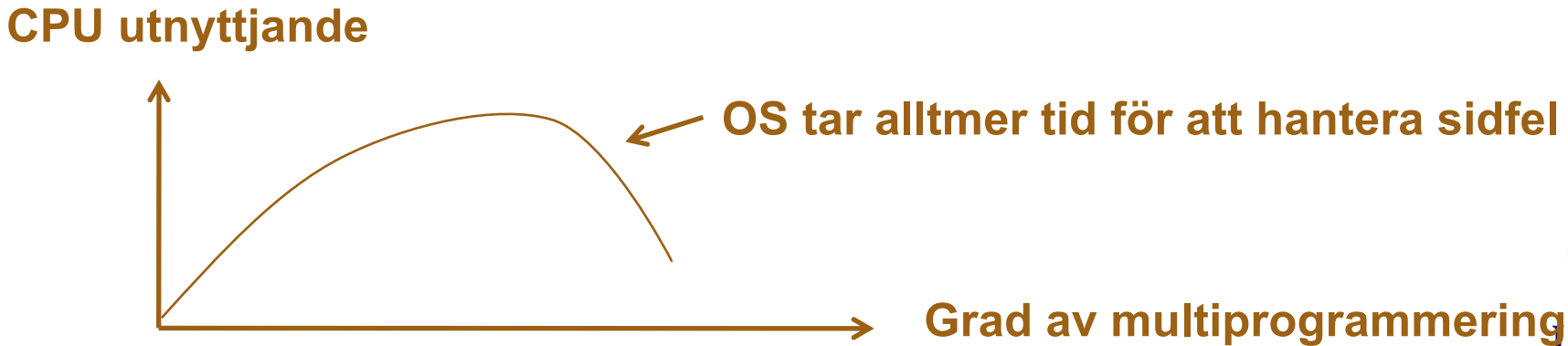
Demand paging

- Ett program består av ett stort antal “pages” som är lagrade på sekundärminnet
- Vid demand paging laddas endast de sidor som behövs.
- Om en sida (page) inte finns i primärminnet, blir det sidfel, och en ny sida laddas in.
- Sidfel – jämför med cachemiss – en algoritm används för att välja ut vilken sida som ska bytas ut
- Varje sida har en bit som indikerar om skrivningar gjorts
  - Om skrivningar gjorts, måste sidan i primärminnet sparas på sekundärminnet (jämför med cacheminnen)

LUNDS UNIVERSITET

Demand paging

- Ladda endast de pages som behövs till primärminnet
  - OS sköter detta
- OS måste balansera hur många program som är aktiva. Om OS väljer många program, ökar graden av multi-programmering. Men, varje program får mindre plats. Alltså, större risk för sidfel, som kostar i tid. OS måste se till att all kraft inte går åt för sidbyten (kallas trashing)



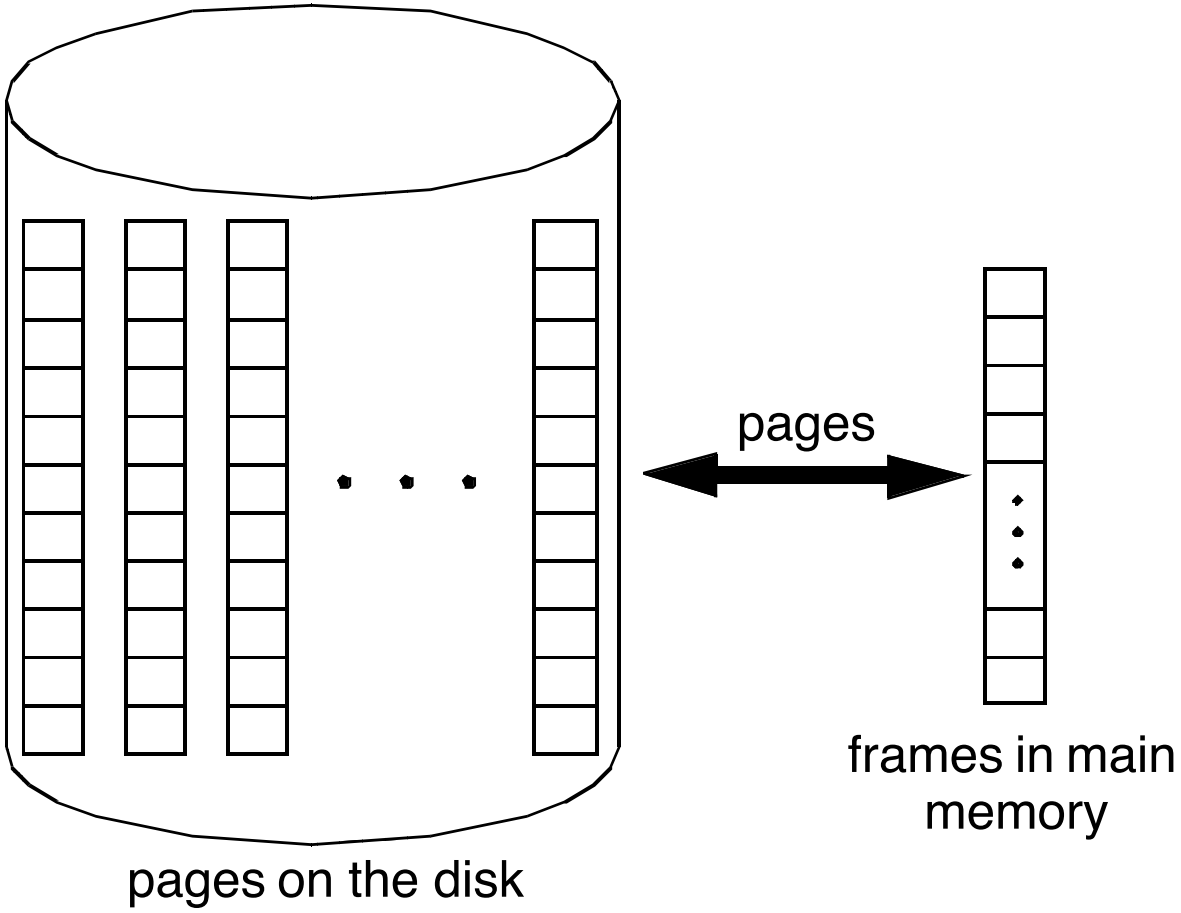
LUNDS UNIVERSITET

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



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 ($2^{24}/2^{11}=2^{13}$)

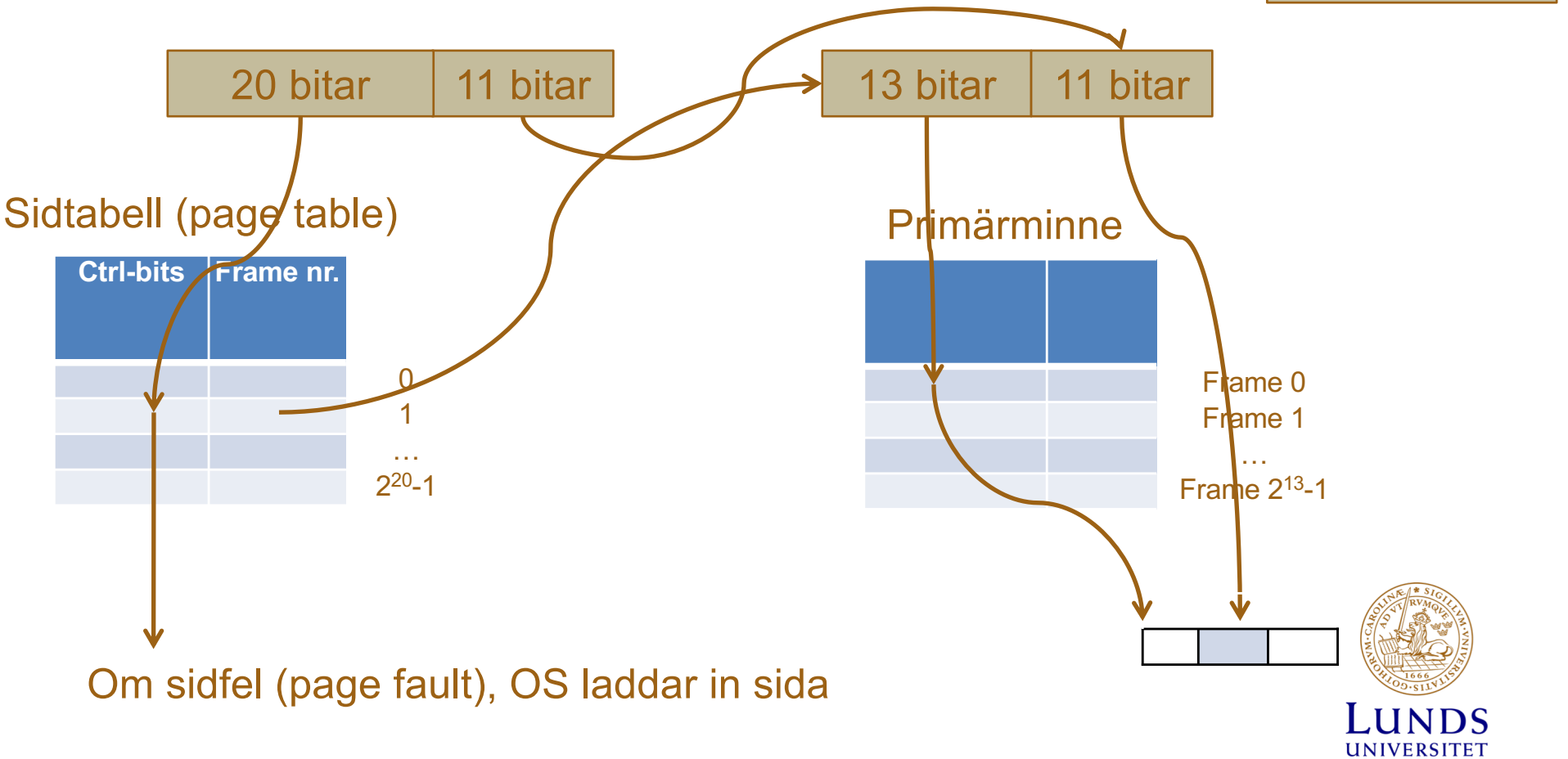
LUNDS UNIVERSITET

Virtuellt minne

Memory Management Unit (MMU)

- Virtuell adress: 31 bitar

Fysisk adress: 24 bitar



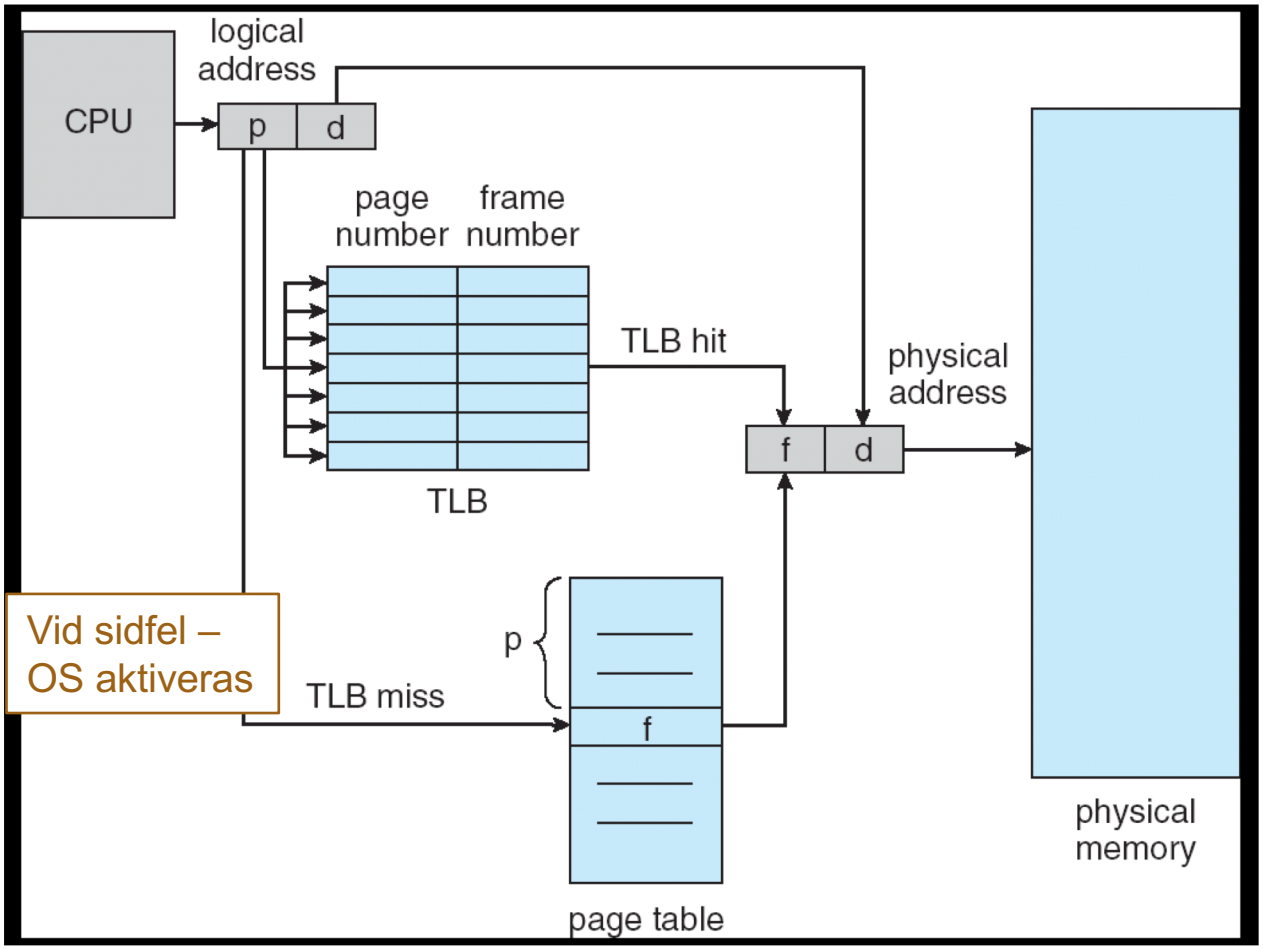
LUNDS UNIVERSITET

Virtuellt minne

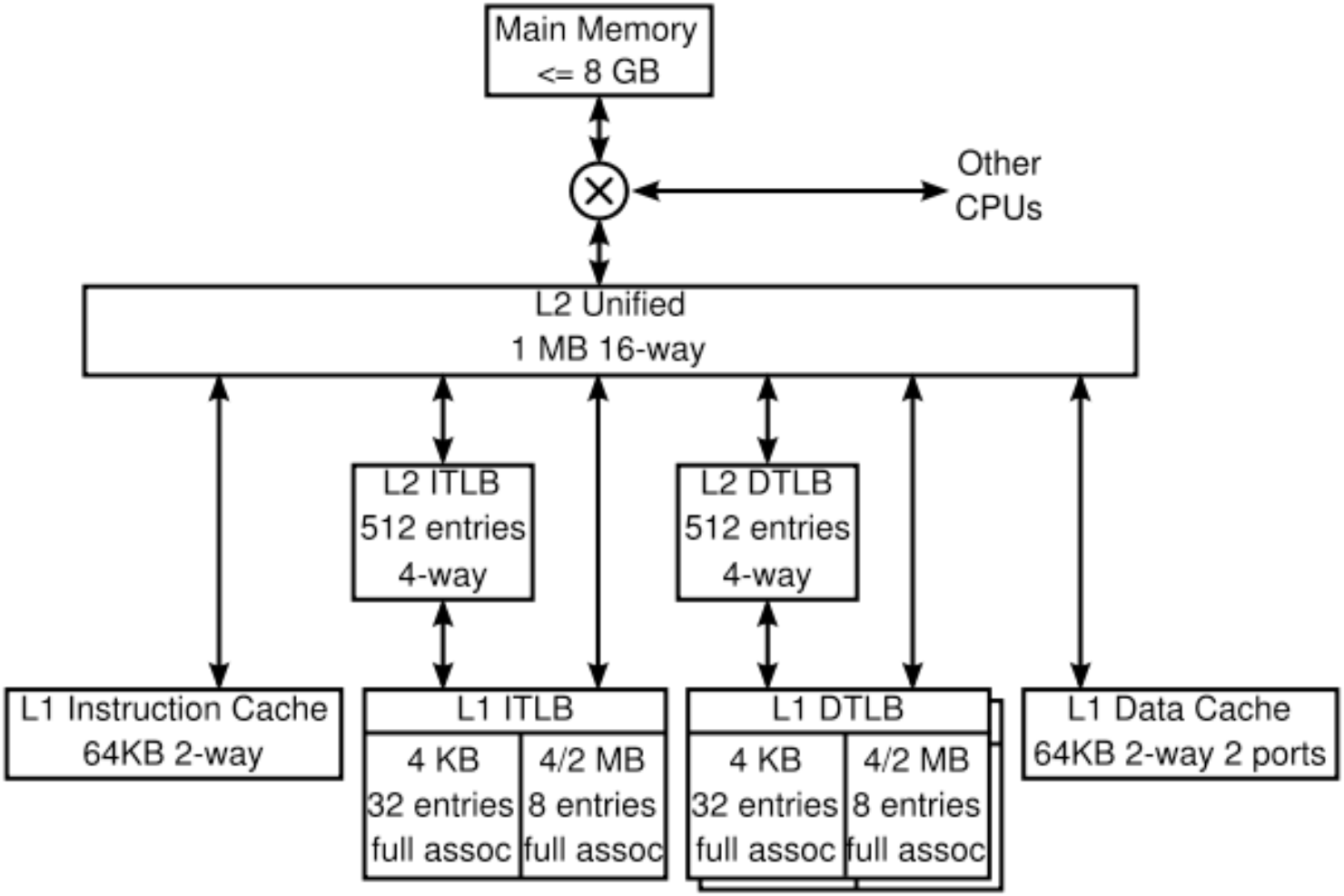
- Problem med sidtabell
  - Tid vid läsning av adress:
    - » 1 läs sidtabell
    - » 2 läs data
  - Stora sidtabeller
- Lösning: använd cache - Translation Look-Aside Buffer (TLB) – för sidtabeller

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