



LUNDS
UNIVERSITET

Datorteknik

ERIK LARSSON



Program

- Abstraktionsnivå:
 - Högnivåspråk
 - » t ex C, C++
 - Assemblyspråk
 - » t ex ADD R1, R2
 - Maskinspråk
 - » t ex 001101....101

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
```

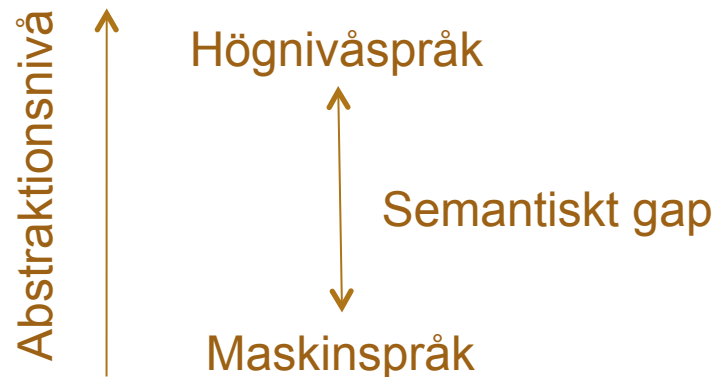
Assembler

Binary machine
language
program
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

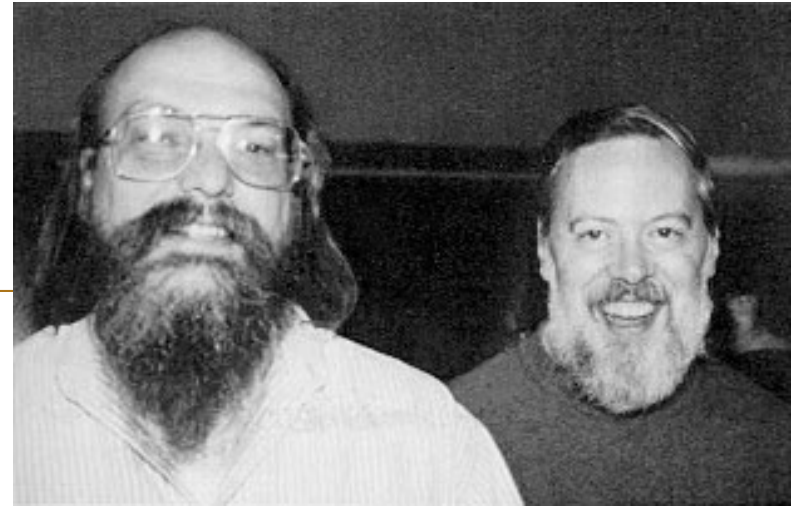
Semantiskt gap

- Alltmer avancerade programmeringsspråk tas fram för att göra programvaruutveckling mer kraftfull
- Dessa programmeringsspråk (Ada, C++, Java) ger högre abstraktionsnivå, konsistens och kraft
- Det semantiska gapet ökar (högnivåspråk-maskinspråk)



C - Inledning

- Ken Thompson och Dennis M. Ritchie utvecklade C
- Turingpriset(“Nobelpris i datavetenskap”), 1983
 - Alan Turing (1912-1954)
- För deras utveckling av generellt OS teori och speciellt för deras implementation av operativsystemet UNIX



Datatyper

<u>Datatyp</u>	<u>Antal bytes</u>	<u>Talområde</u>
unsigned char	1	0 — 255
signed char	1	-128 — 127
unsigned int	2	0 — 65535
signed int	2	-32768 — 32767
unsigned long int	4	0 — 4294967295
signed long int	4	-2147483648 — 2147483647
float	4	$\pm 1,18 \text{ E-}38$ — $3,39 \text{ E+}38$



Datatyper

- `char Tal, Max, Min;`
 - Exempel: `Tal=5;`
- `unsigned int Adress;`
 - Exempel: `Adress=512;`
- `const char Tabell [][][3] = {{ 23, 30, 64 } ,`
`{ 12, 31, 16 } ,`
`{ 42, 54, 86 } ,`
`{ 29, 32, 64 } };`
 - Exempel: `Tabell[1][1]=23;`
- `const char String [] = "ABC";`



Datatyper

- Samma sak lagras:

`char a = 65;`

decimalt

`char a = 0x41;`

hexadecimalt

`char a = 0b01000001;`

binärt

`char a = 'A';`

ASCII-kod

Primärminne

Address	Instruction/Data
00001000	01000001
00001001	00011011
00001010	00101000
00001011	00010011



LUNDS
UNIVERSITET

Datatyper

- Exempel:

```
c1=5;      /* c1 har bitmönstret 00000101 */  
c2=6;      /* c2 har bitmönstret 00000110 */
```

Binärt värde av 5



Tilldelningssatser

- Deklarera variabel:

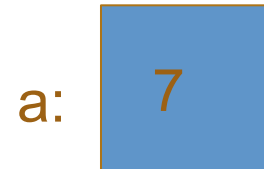
```
unsigned char a;
```

- Tilldela variabeln ett värde:

```
a = 5;
```

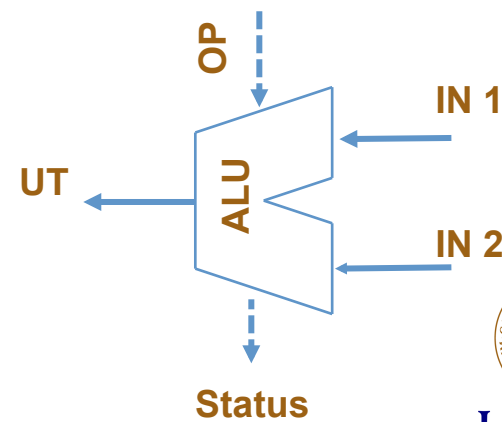
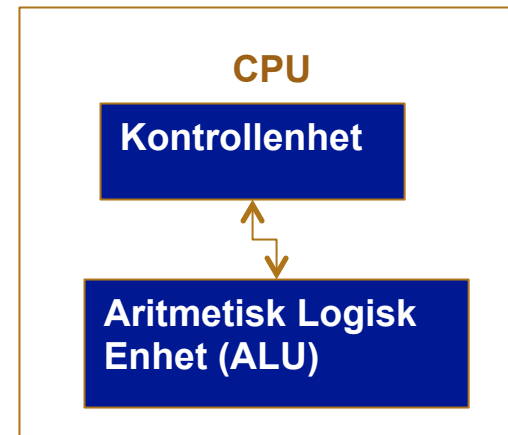
- Addera 2 till värdet i a:

```
a = a + 2;
```



Aritmetiska operationer

- Operationer:
 - + addition
 - subtraktion
 - * multiplikation
 - / division
 - % Modulodivision



Beräkningar

- `unsigned char a, b, c;`

- **Exempel:**

```
a=5;
```

```
b=10+a-3;
```

```
c=a*b;
```

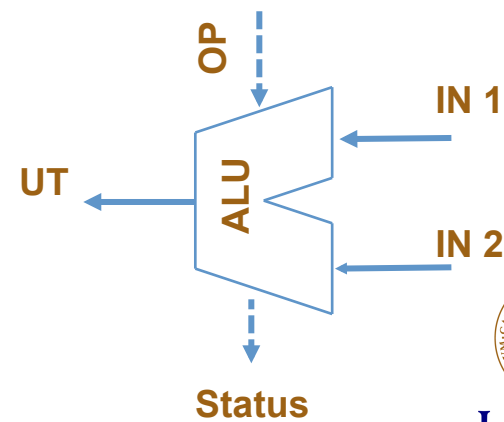
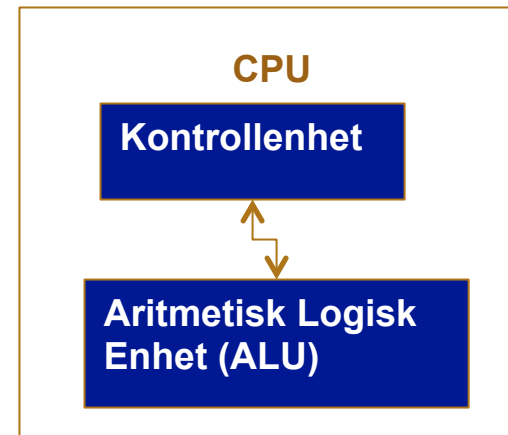
```
a=b|20;           // bitvis eller (or)
```

```
b++;             //samma som b=b+1
```



Bithantering

- Och (AND): &
- Eller (OR): |
- Exklusivt eller (XOR): ^
- Invertering (NOT): ~
- Vänstershift: <<
- Högershift: >>



Bithantering

- Exempel:

```
c1=5;          /* c1 har bitmönstret 00000101 */
c2=6;          /* c2 har bitmönstret 00000110 */
```

```
c9=~c1;        /* c9 får bitmönstret 11111010 */
c10=c1<<3;     /* c10 får bitmönstret 00101000 */
c11=c1<<6;     /* c11 får bitmönstret 01000000 */
c12=c1>>2      /* c12 får bitmönstret 00000001 */
c13=c1&c2      /* c13 får bitmönstret 00000100 */
c14=c1|c2      /* c14 får bitmönstret 00000111 */
```



Villkor

- `if (villkor) sats;`

- Exempel 1:

```
int n
if ( n == 27 ) { din kod här }
```

- Exempel 2:

```
int n;
if ( n == 27 ) { din kod om talet var 27 }
else { din kod om talet inte var 27 }
```



Villkorsuttryck

- Om $a=5$ så öka a med 1:

```
if (a == 5) a = a + 1;    // if (villkor) sats;
```

- Villkor:

$==$ lika med, $>$ större än, $<$ mindre än, $!=$ inte lika med

- Om $a=10$ så öka a med 2 i annat fall minska a med 3:

```
if (a == 10)
    a = a + 2;
else
    a = a - 3;
```



Villkorsuttryck

- Så länge $a < 5$ öka b med 3:

```
while (a < 5) {  
    a = a + 2;  
    b = b + 3;  
}
```



Loopar

- **Alternativ:**

```
while (uttryck) sats;  
do sats; while (uttryck);  
for (initiering; styruttryck; stegning)  
sats;
```

- **Exempel 1:**

```
int n=1;  
while ( n++ <= 10 ) { din kod }
```

- **Exempel 2:**

```
int n;  
for ( n=1; n <= 10; n++ ) { din kod här
```



Funktioner

- All kod paketeras i funktioner.
- Huvudprogrammet:

```
void main(void) {  
    b = 5 + my_funktion(3);  
}
```

Funktionsanrop

Inparameter

- En funktion deklarerar:

```
int my_funktion(x)  
    int x  
{  
    return (x+2);  
}
```

Datatyp som returneras

Gör/skapar retur värdet



LUNDS
UNIVERSITET

Funktioner


- Exempel: Funktionen kvadrat beräknar: $y=x*x$ kan se ut:

```
int kvadrat(x)
int x
{
    return x*x;
}
```

- Anrop: `y=kvadrat(3);` //y blir 9
- Värdet av x skickas in och funktionen returnerar kvadraten.



Pekare (Intro)

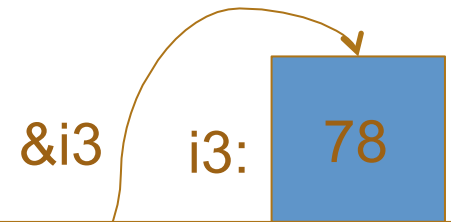
i3: 

i4: 

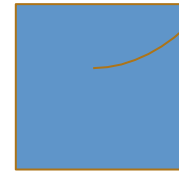
- Deklaration:
`int i3, i4`
- Ger att:
i3 och i4 är heltalsvaribler
- Exempel:
`i3=78; //sätter i3 till 78`
`i4=5 //sätter i4 till 5`
- Låt i3 vara lagrat på adress 0
- Låt i4 vara lagrat på adress 1

Adress	Data
0 (i3)	78
1 (i4)	5
2	
3	
4	
5	
6	

Pekare



`ip1:`



- Deklarationen:

```
int i3, i4, *ip1, *ip2;
```

- Ger att:

`*ip1` och `*ip2` är heltalspekarsvariabler
`i3` och `i4` är heltalsvariabel

- Exempel:

```
i3=78; //sätter i3 till 78
```

```
ip1=&i3 //sätter ip1 att peka på  
adress där i3 finns
```

& ger adressen till något

- Notera: `ip1` har plats för en pil (adress) och `i3` har plats för ett heltal

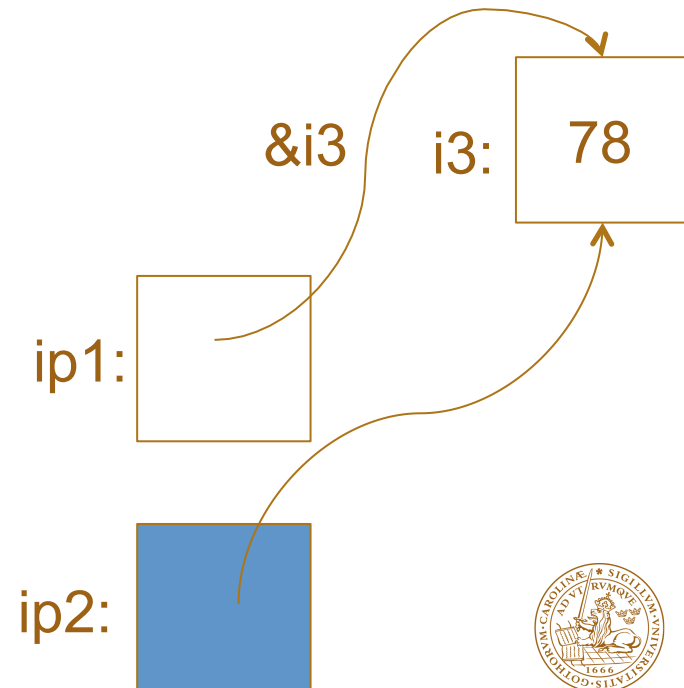
Adress	Data
0 (i3)	78
1 (i4)	5
2 (ip1)	
3 (ip2)	
4	
5	
6	

Pekare

- Exempel: Deklarationen (samma som innan):

```
int *ip1, *ip2, i3, i4  
i3=78  
ip1=&i3
```

```
ip2=ip1 //ip2 sätts att peka  
på samma som ip1
```



Pekare

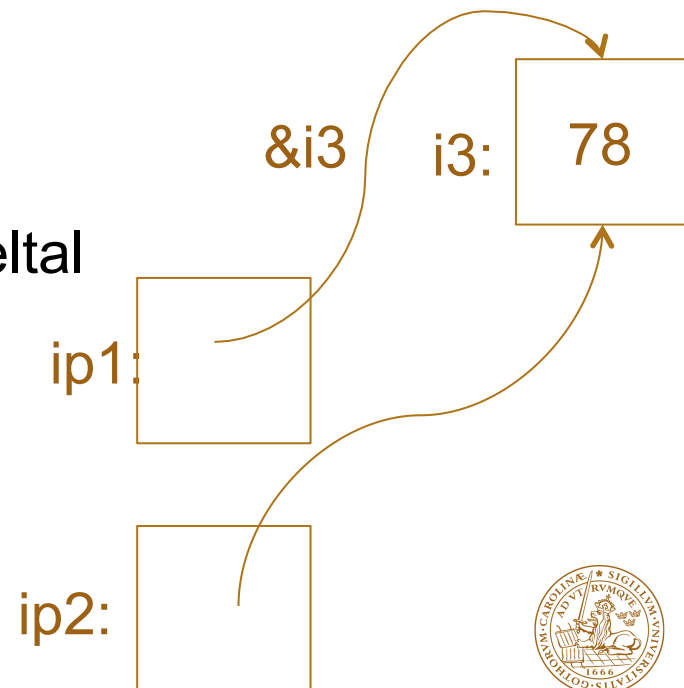
- Exempel: Deklarationen (samma som innan):

```
int *ip1, *ip2, i3, i4  
i3=78;  
ip1=&i3;  
ip2=ip1;
```

i4=*ip1; //i4 sätts till det heltal
som ip1 pekar på

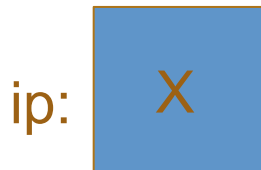
- *ip1 består av två steg. Först, tas pekaren fram. Sedan, via *, tas värdet till pekaren fram

i4: 78



Pekare

- Precis som andra variabler, blir pekare inte automatiskt tilldelade ett värde vid deklaration.
- För att sätta en pekare att peka på ingenting:
`ip=NULL;`
- Sätts inte en pekare att peka på ingenting kan den peka på vad som helst – det som råkar ligga på den minnesplatsen.



Pekare till funktioner

- Om, parametern är en pekare.....

- Exempel:

```
void kvadrat(xref)
int *xref
{
    *xref= *xref * *xref;
}
```

- Anrop: `kvadrat (&x) ;`
- x är ett pekarvärde. Funktionen tar värdet av pekaren, gör kvadrat och updaterar pekaren

Adress	Data
0 (a)	10
1 (b)	5
2 xref	
3	
4	
5	
6	



Exempel 1:Fråga

- Komplettera koden nedan så att värdet i variabel a och b byter värde.

```
void main (void){  
    int a, b;    // deklaration av värde  
    a = 10;      // a tilldelas värde  
    b = 5;       // b tilldelas värde  
    ?           // kod för att a och b  
                // byter värde  
}
```

Adress	Data
0 (a)	10
1 (b)	5
2	
3	
4	
5	
6	

Exempel 1:Lösning

- Kod där värdet i variabel a och b byter värde.

```
void main (void) {  
    int a, b, tmp;    // deklaration av värde  
    a=10;              // a tilldelas värde  
    b=5;              // b tilldelas värde  
    tmp=a;            //kod för att a och b  
    a=b;              //byter värde  
    b=tmp;  
}
```



Illustration av lösning

a=10;
b=5;

Adress	Data
0 (a)	10
1 (b)	5
2 (tmp)	
3	
4	
5	
6	

tmp=a;

Adress	Data
0 (a)	10
1 (b)	5
2 (tmp)	10
3	
4	
5	
6	

a=b;

Adress	Data
0 (a)	5
1 (b)	5
2 (tmp)	10
3	
4	
5	
6	

b=tmp;

Adress	Data
0 (a)	5
1 (b)	10
2 (tmp)	10
3	
4	
5	
6	

Exempel 2:Fråga

- Skriv en funktion swap som byter värden på två variabler

```
void main (void) {  
    int a, b;           // deklaration av värde  
    a = 10;             // a tilldelas värde  
    b = 5;              // b tilldelas värde  
    swap(a,b) ;  
}
```



Exempel 2:Fråga+problem

- Skriv en funktion swap som byter värden på två variabler

```
void main (void){  
    int a, b;  // deklaration av värde  
    a = 10;    // a tilldelas värde  
    b = 5;     // b tilldelas värde  
    a=swap(a,b) ;  
}  
  
int swap (int c, d){  
    int temp;  
    temp=c;  
    c=d;  
    d=temp;  
    return ?????  
}
```



Exempel 2: Lösning

- Skriv en funktion swap som byter värden på två variabler

```
void swap (int *a, *b) {  
    int temp; //vanlig variabel  
    temp=*a;   // * ger värdet som a pekar på  
    *a=*b;  
    *b=temp;  
}
```



Illustration av lösning

a=10;
b=5;

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	
3	
4	10
5	5
6	

tmp=*a;
(tilldelar
tmp det som
*a pekar
på)

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	10
3	
4	10
5	5
6	

*a=*b;
(tilldelar
det a pekar
på värdet
som finns i
b)

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	10
3	
4	5
5	5
6	

*b=tmp;
(tilldelar
den plats b
pekar på
värdet i
tmp)

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	10
3	
4	5
5	10
6	

Variablers synlighet

```
#include <stdio.h>  
unsigned char n;
```

Global variabel

Global variabel

```
void display (unsigned char number){  
    static int a;  
    int c;  
    c=4+a; }
```

Lokal variabel

```
int main(void){  
    int b;  
    n=5;  
    b=10;  
    display(b); }
```

Lokal variabel



LUNDS
UNIVERSITET

Include

- Includeringsbara bibliotek:

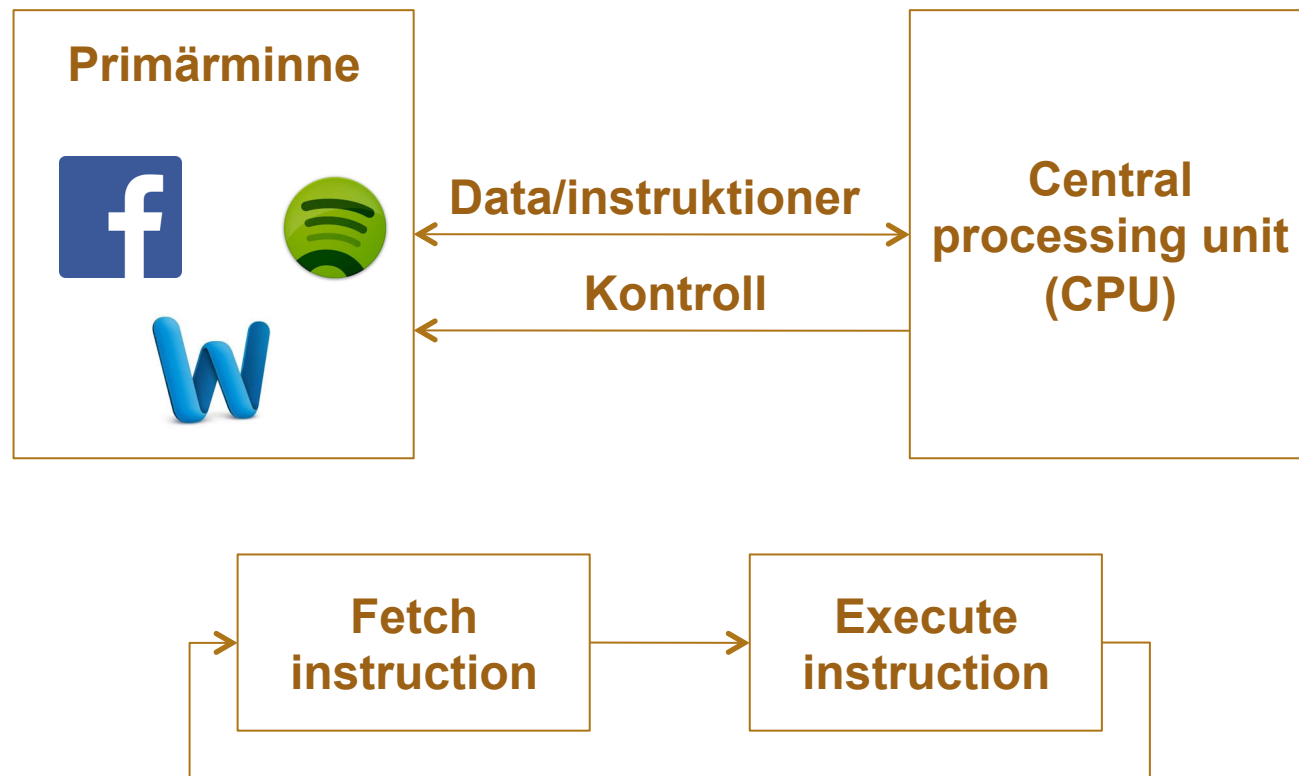
```
#include <stdio.h>
```

standardfunktioner för I/O



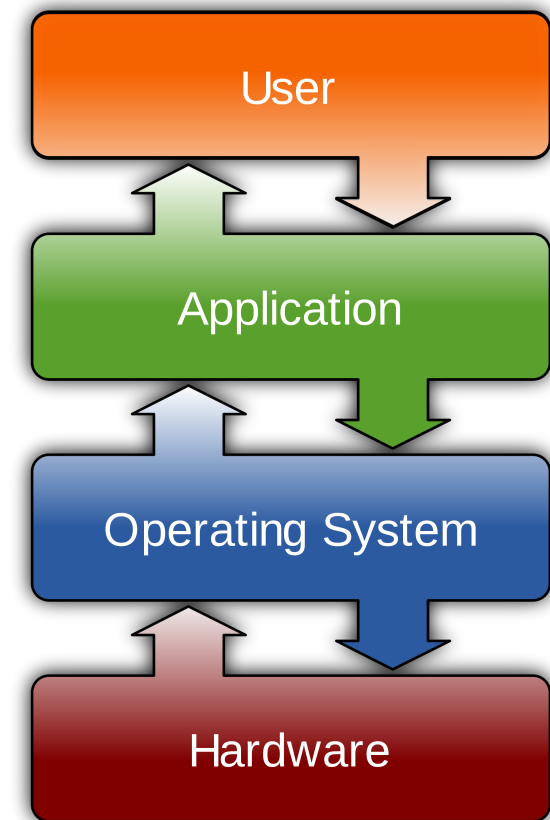
LUNDS
UNIVERSITET

Dator

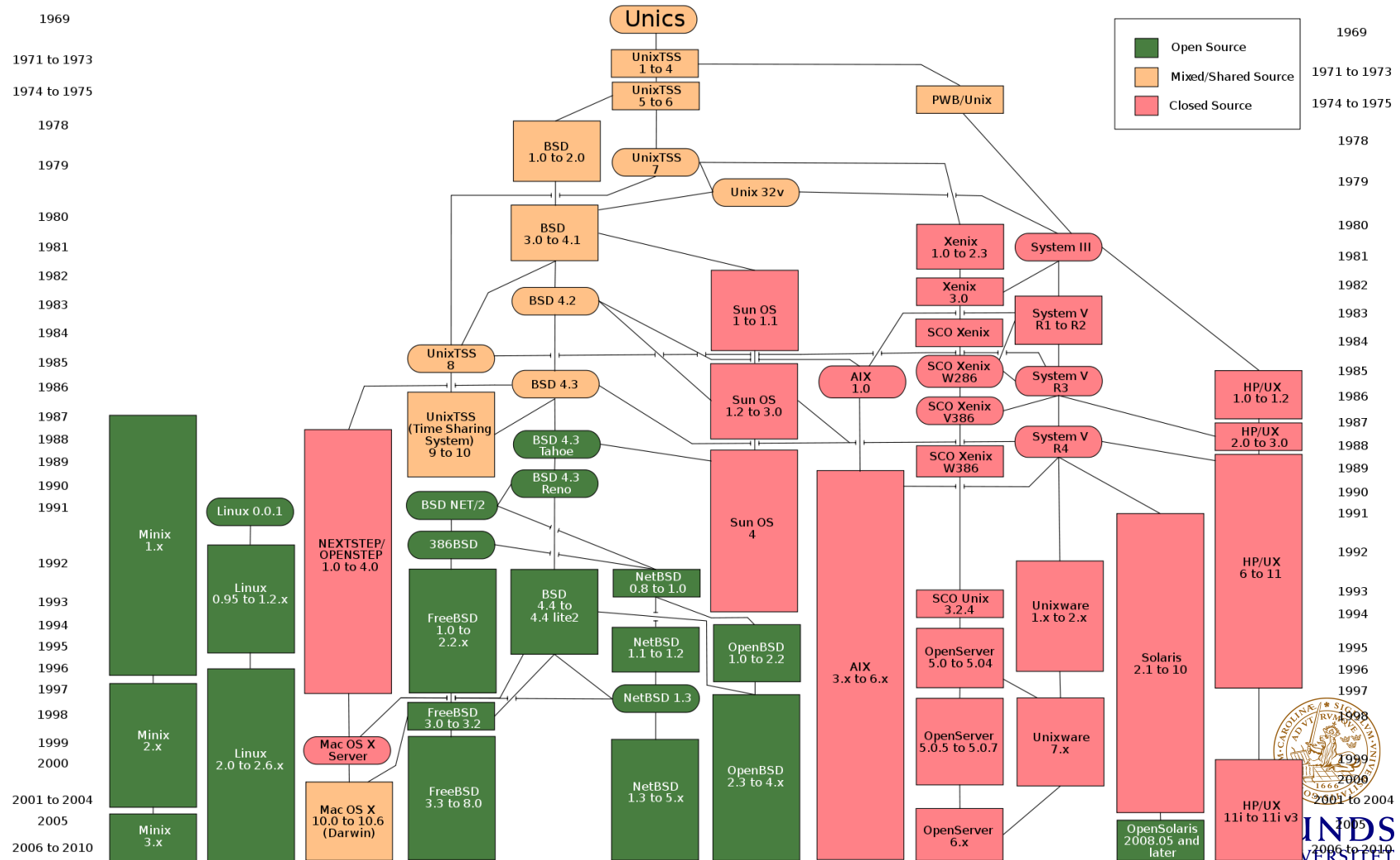


Inledning

- Ett operativsystem (Operating System - OS) är ett program som exekveras på datorn
- Mål för OS är:
 - Hantera hårdvaruresurser i datorsystemet
 - Erhålla tjänster för exekvering av applikationsprogram (t ex Facebook)
- I stort sett alla system har någon form av OS – från mobiltelefoner, datorspel, till superdatorer.

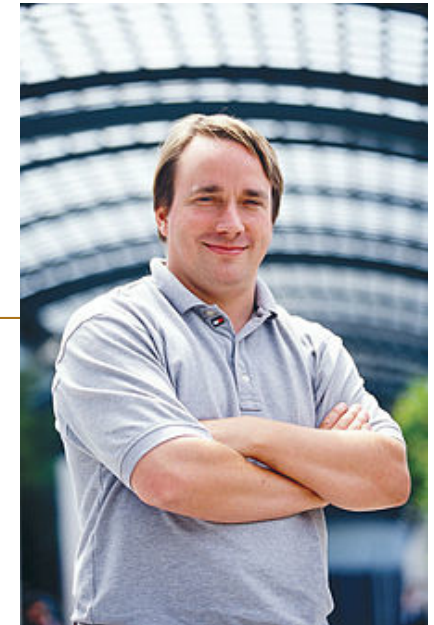


Unix



Linux

- Linux
 - Linus Benedict Torvalds, född 1969, Finland
 - Ville lära sig om OS, skrev ett OS
- Windows
 - MS-DOS (Microsoft Disk Operating System) (~1980)
 - Windows 1.0, Windows 95, 98, 2000, XP, Vista, 7, 8



Bill Gates **Paul Allen**



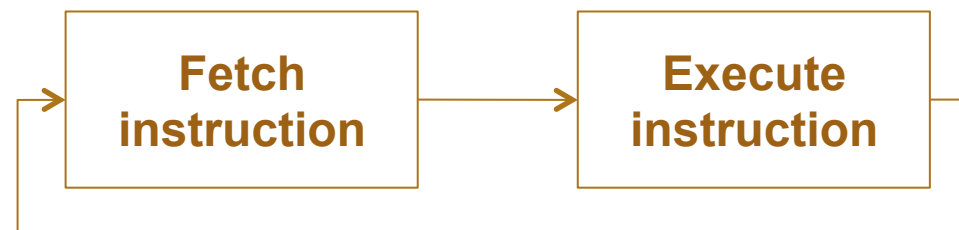
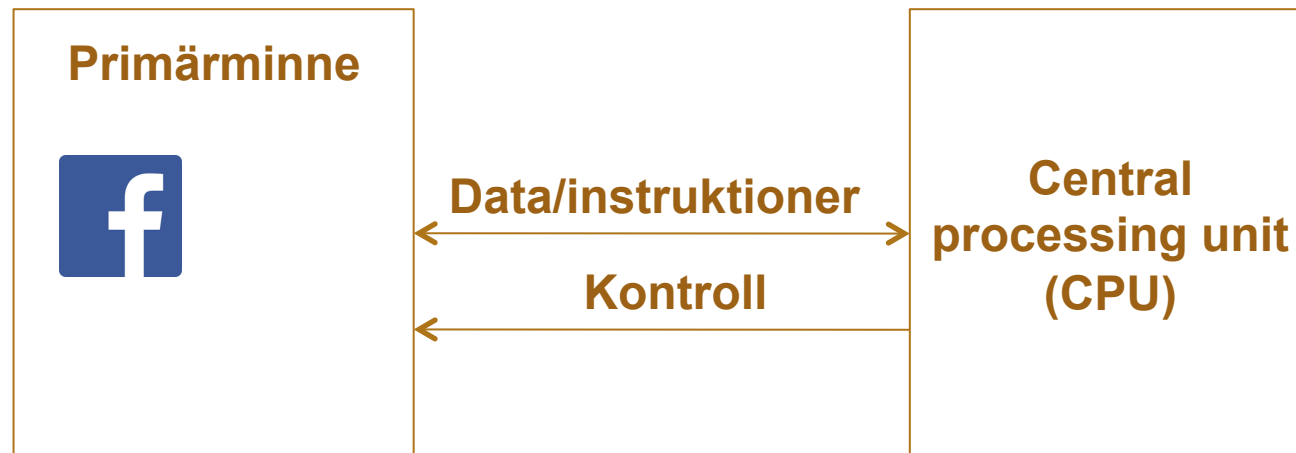
LUNDS
UNIVERSITET

Vad gör ett OS?

- Processhantering (Process management)
- Avbrott (Interrupts)
- Minneshantering (Memory management)
- Filsystem (File system)
- Drivrutiner (Device drivers)
- Nätverk (Networking)
- Säkerhet (Security)
- In och utmatning (I/O)

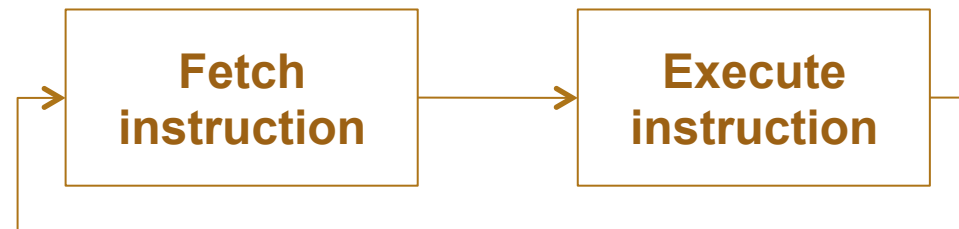
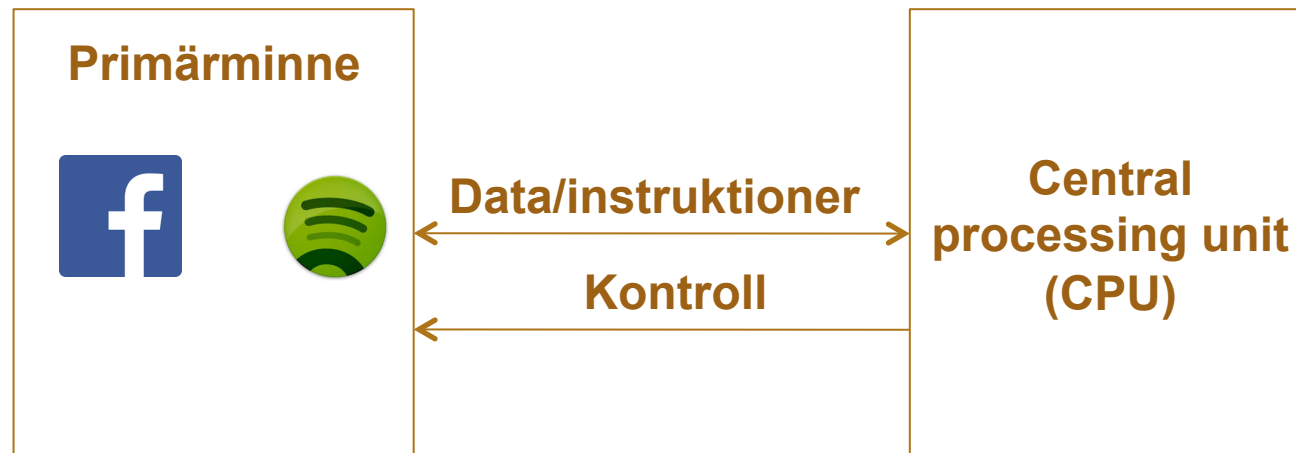


Program



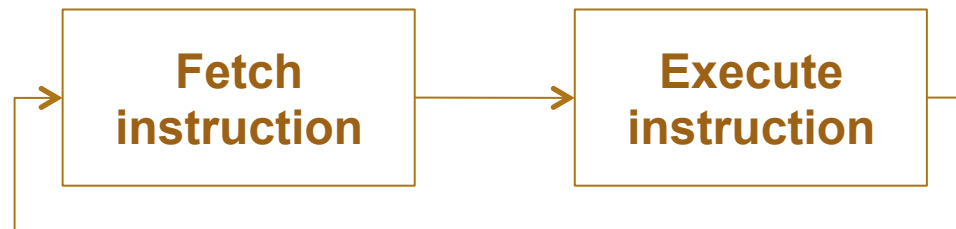
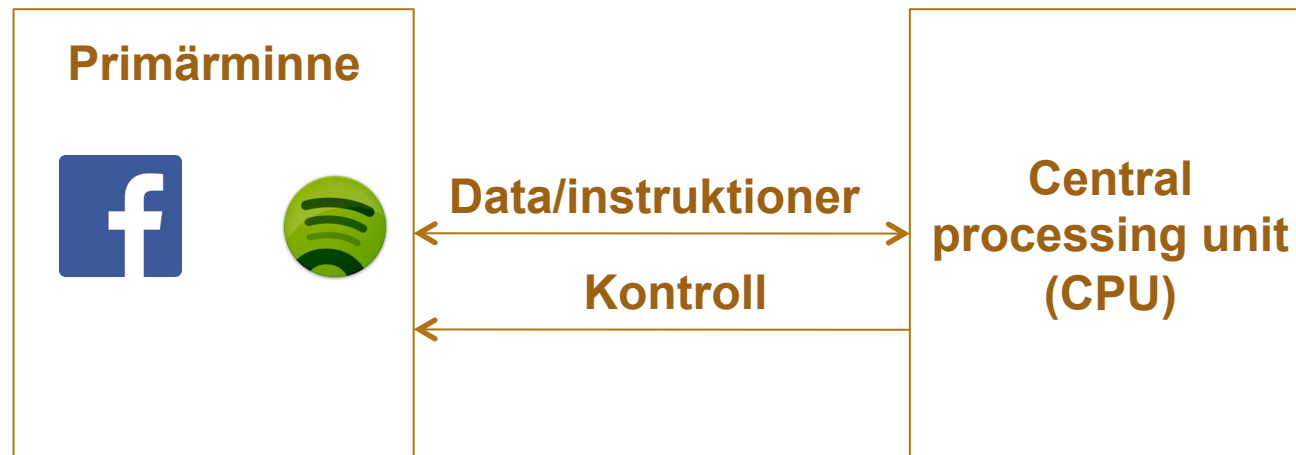
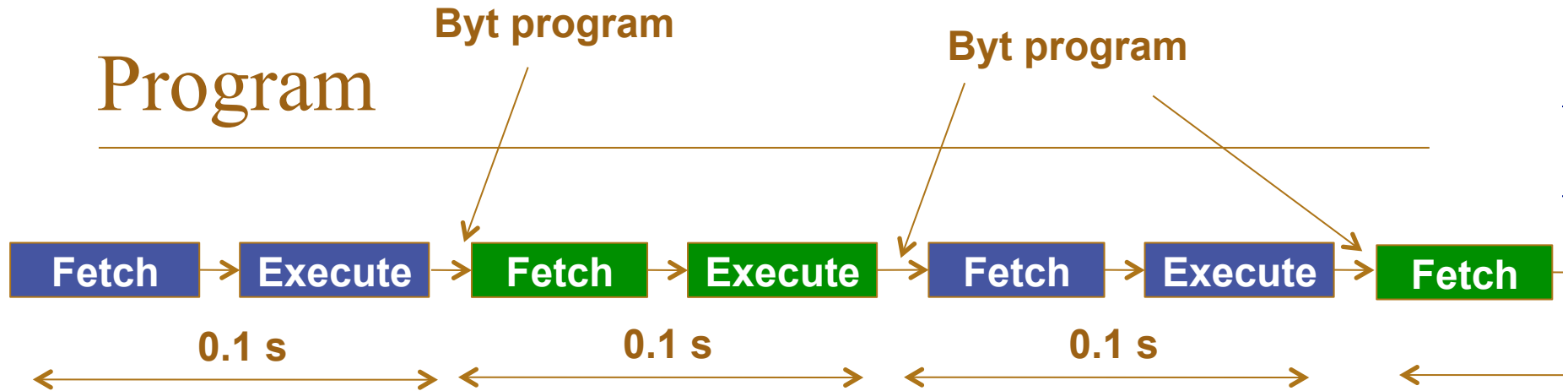
Program

Byt program



LUNDS
UNIVERSITET

Program



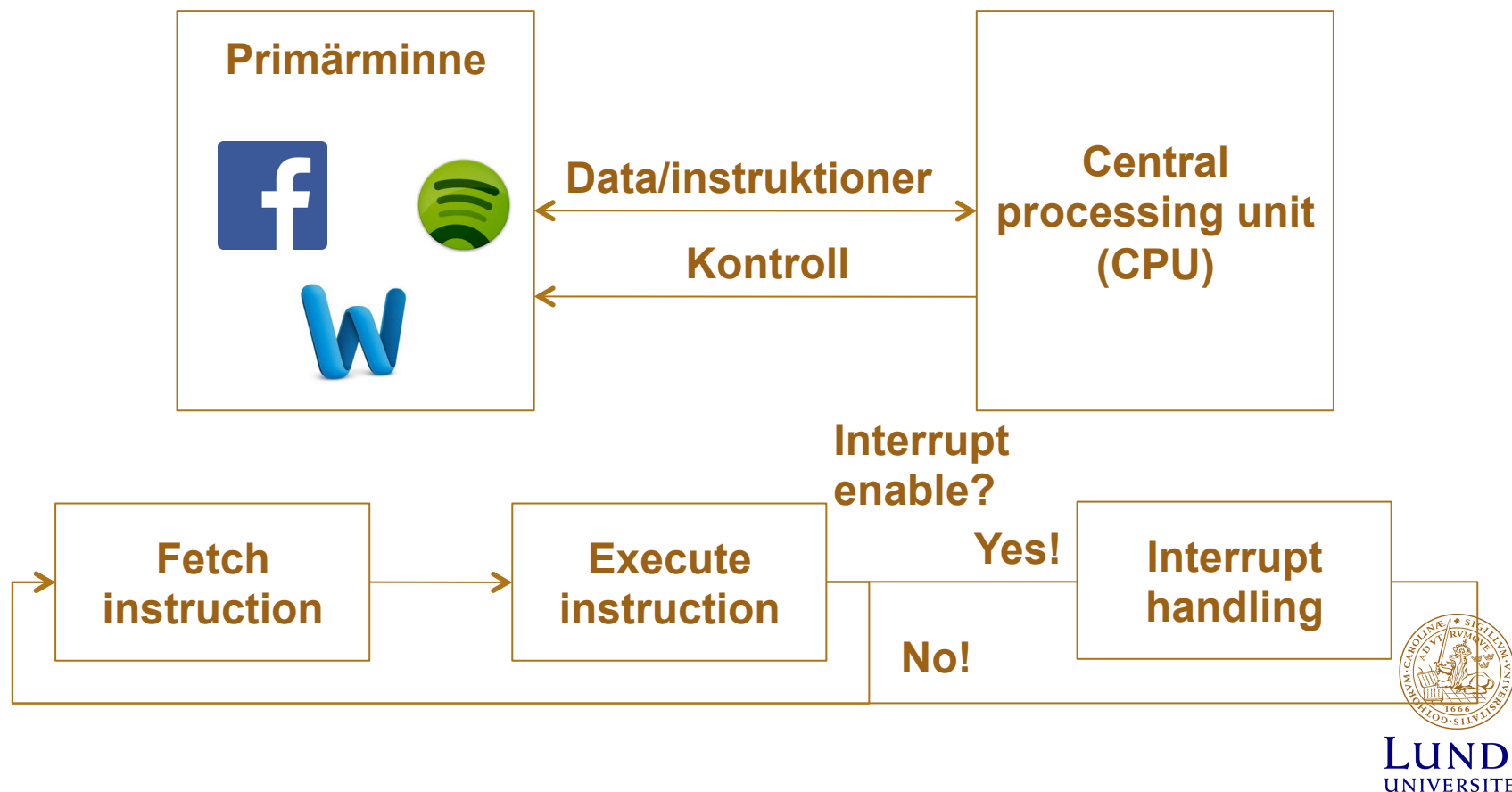
LUNDS
UNIVERSITET

Vad hinner man på 0.1 sekund?

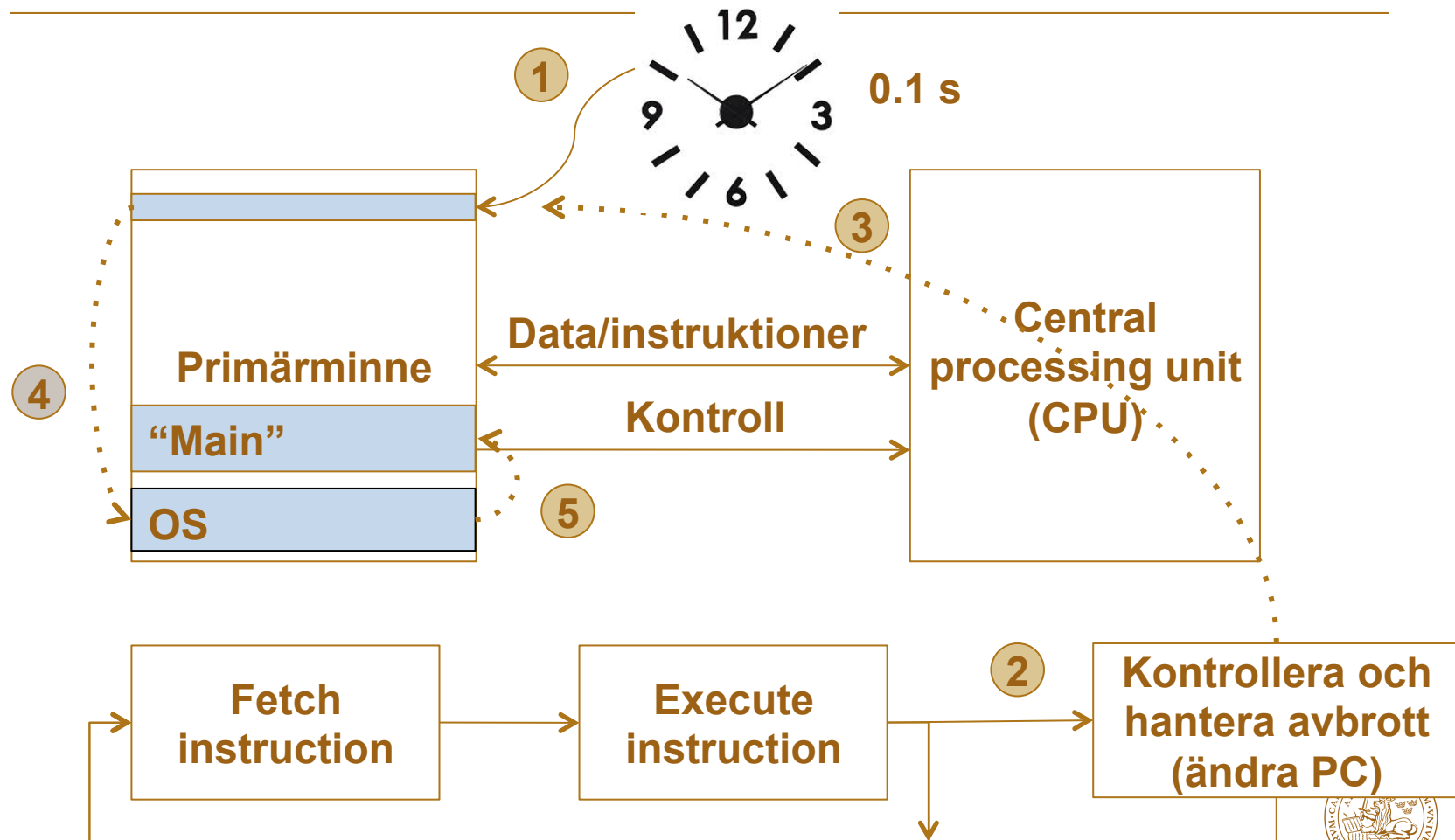
- Antag processor med 1 GHz klockfrekvens
 - 1 GHz = 1 000 000 000 Hz (svängningar per sekund)
- På 1 sekund hinner man 1 000 000 000 klockcykler
- På 0.1 sekund hinner man 100 000 000 klockcykler
 - Om varje instruktion tar 10 klockcykler, hinner man:
10 000 000 instruktioner



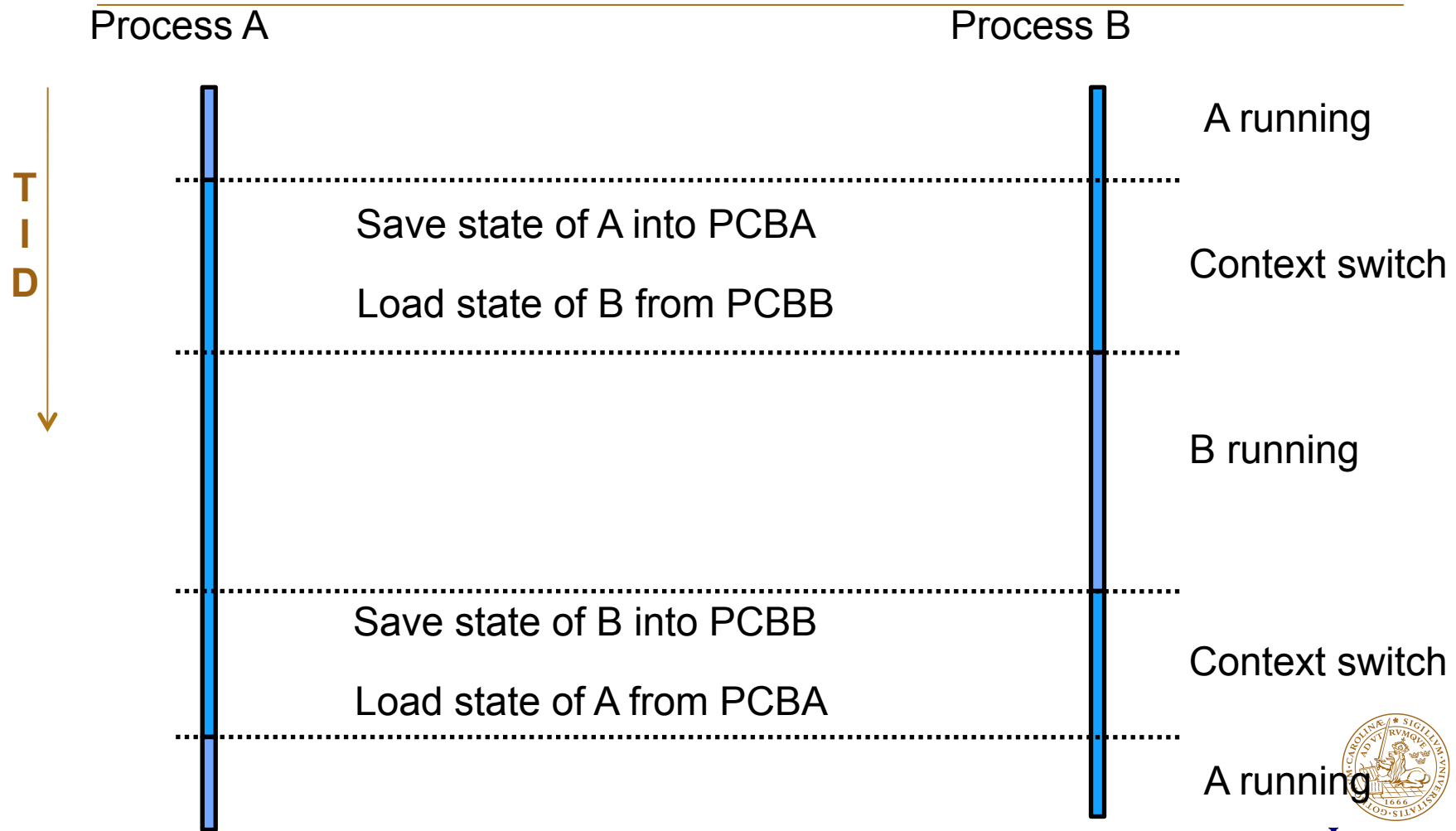
Hur går det till att byta program?



Hur går det till att byta program?



Kontextbyte (context switch)

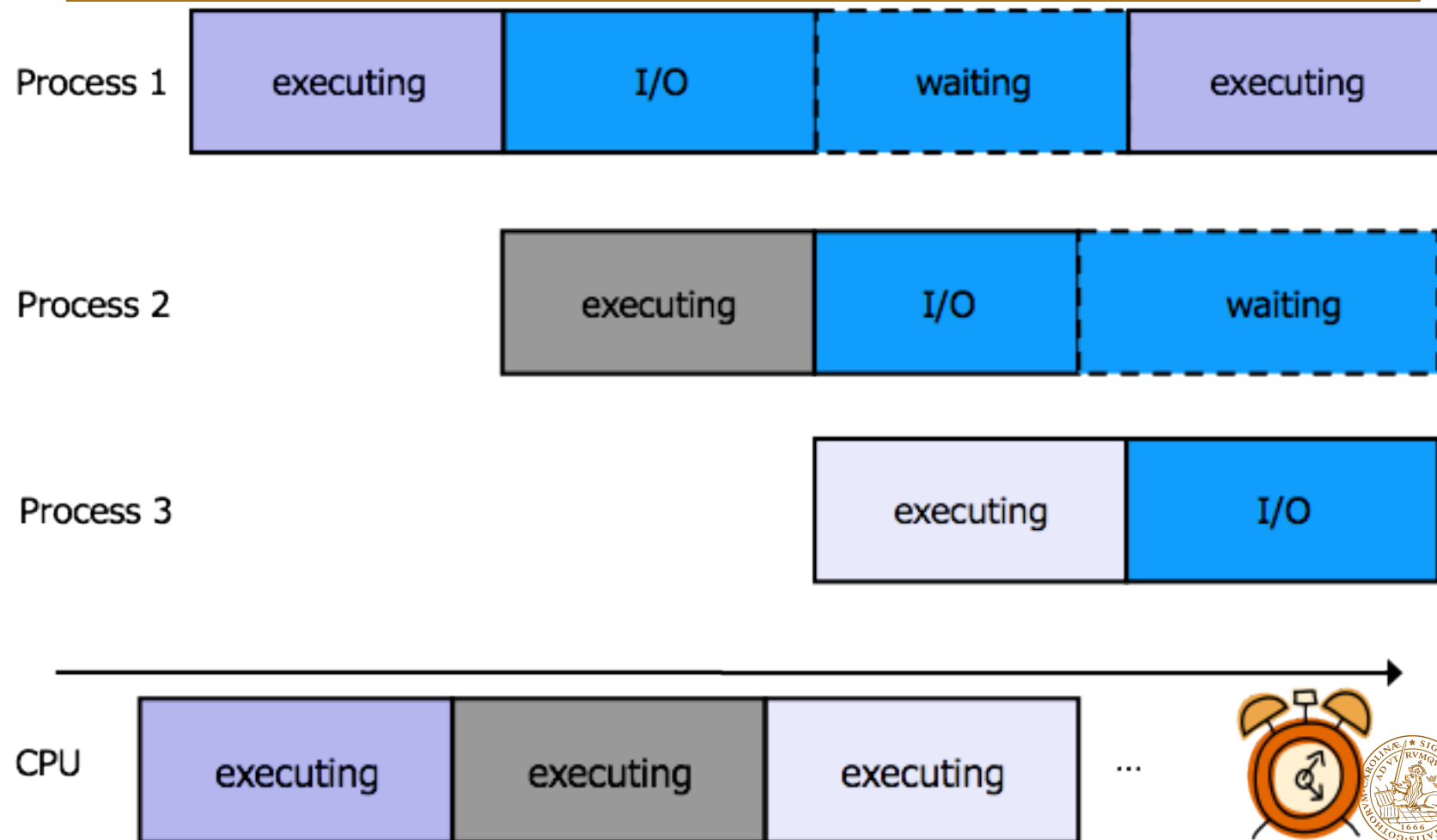


Processkontrollblock

- Process Control Block (PCB, eller Task Controlling Block eller Task Struct)
 - är en datastruktur som innehåller den information som behövs för att kunna hantera en given process.
- Ett aktivt program har ett processkontrollblock
- Typiskt innehåll:
 - Identifikation av process (a process identifier, or PID)
 - Register värden, programräknare, stackpekare
- Adressrymd, prioritet, process information, t ex när användes processen senast, I/O som används, öppna filer

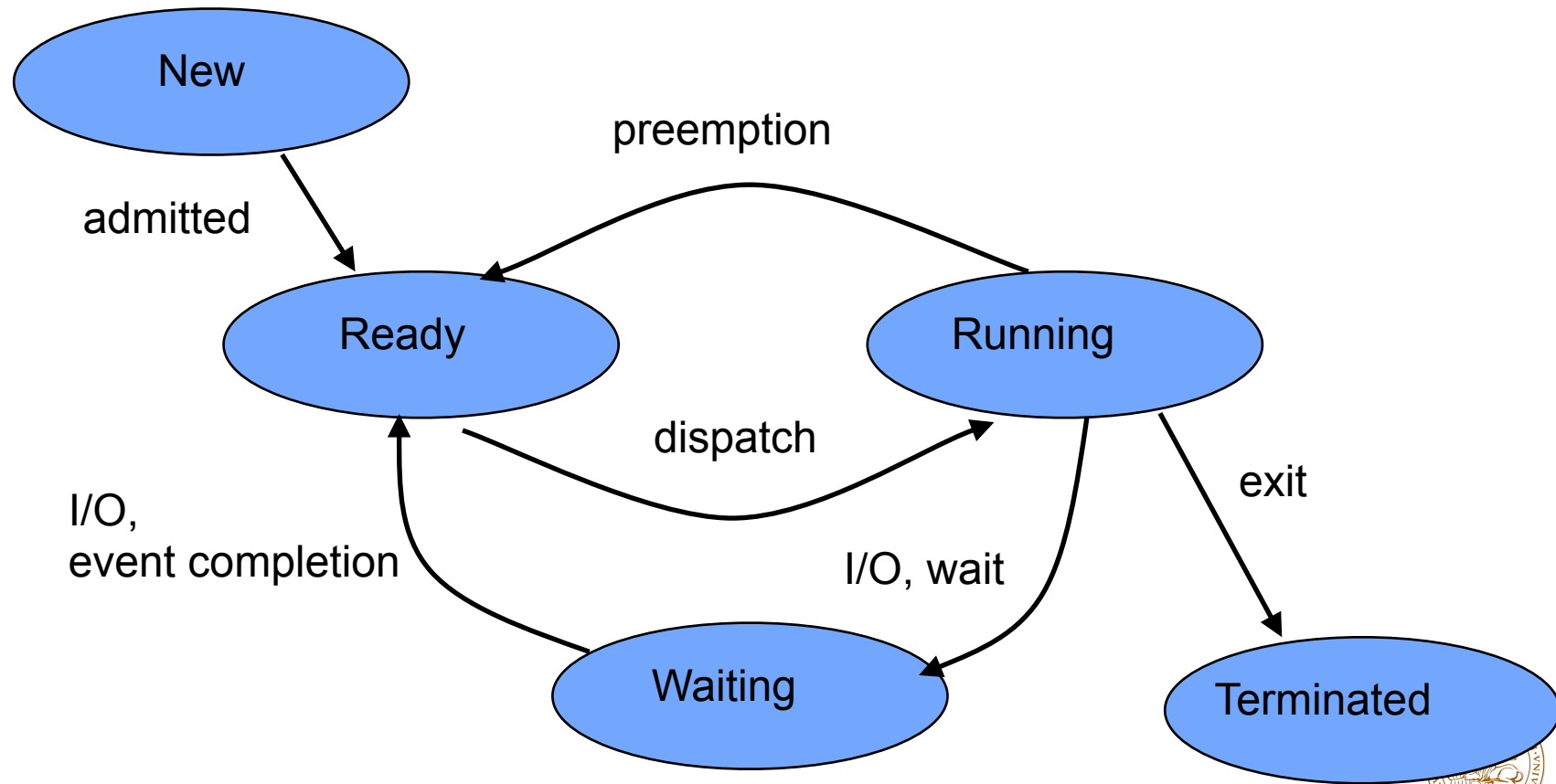


Processhantering



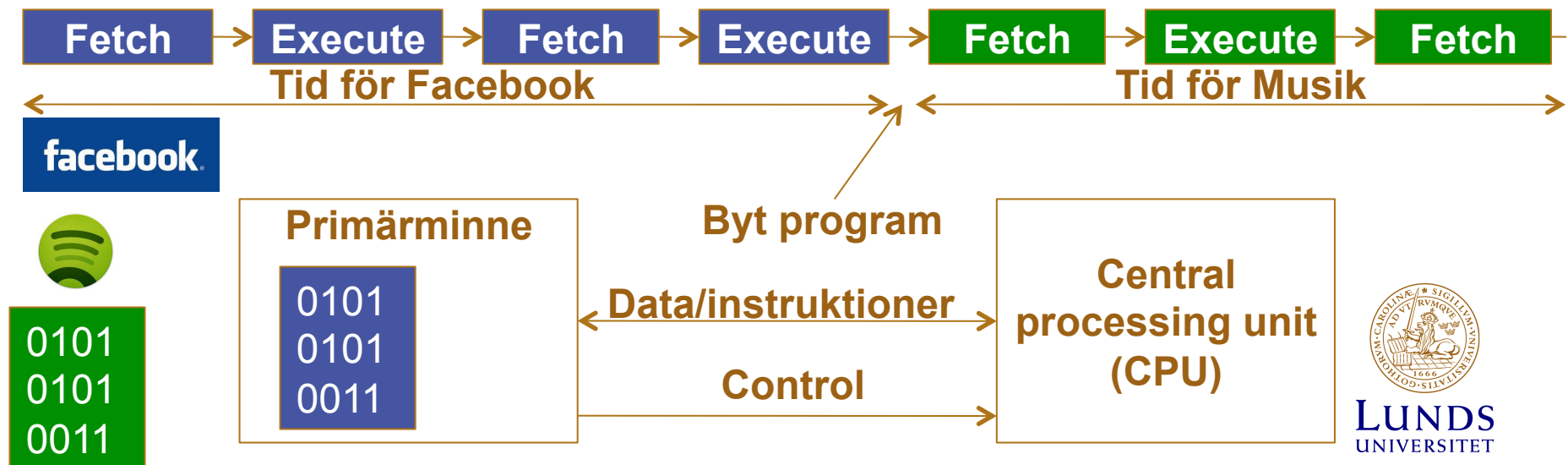
LUNDS
UNIVERSITET

Process modell





Processhantering

- Ett program behöver resurser för att kunna exekvera
- Ett alternerar mellan CPU och I/O cykler
- För att maximera utnyttjandet av CPU, används multiprogramming (time sharing, multi-tasking) – mer än ett program är aktivt.



Processhantering

- En processor – flera program som exekverar.....eller gör de?
- Nej!
- Men, en processor är väldigt snabb, så vi kan ge sken av parallellism.
- Antag att två program, t ex  och  körs
- ~0.1 sekund på varje program, dvs 100 000 000 klockcykler på varje program



LUNDS
UNIVERSITET

- ~10 olika program kan exekveras per sekund

Processmodell

- Två-tillståndsmodell (Running och Not Running)
 - Schemaläggaren väljer en ny process från Not Running kön och låter den exekvera. Ett avbrott (till exempel, vid slut av time slice), leder till context switch och ett nytt program laddas
- Tre-tillståndsmodell (Running, Ready, Blocked)
 - En nackdel med två tillstånd är att CPU:n står “idle” vid I/O. Ett nytt tillstånd (Blocked) införs).
- Fem-tillståndsmodell (Running, Ready, Blocked, Ready suspended, Blocked suspended)
 - För hantering av virtuellt minne (flytta en process från primärminnet till sekundärminnet)



Schemaläggare

- Långtidsschemaläggaren (Long-term scheduler)
 - Bestämmer vilka jobb som ska läggas i readykön (queue)
- Mellantidsschemaläggaren (Mid-term scheduler)
 - Bestämmer vilka jobb som ska vara i primärminnet (main) och vilka som ska vara i sekundärminnet
- Korttidsschemaläggaren (Short-term scheduler)
 - Bestämmer vilket jobb som ska exekvera
- Algoritmer:
 - First In First Out, Shortest Job First, Priority based scheduling, Round-robin scheduling, Multilevel Queue scheduling

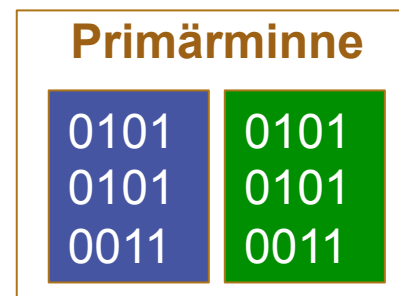
Schemaläggare

- MS-DOS– non multi-task system; hence, no scheduler
- Windows 3.1 - non-preemptive scheduler (did not interrupt programs)
- Windows NT, Linux, MacOS - multilevel feedback queue



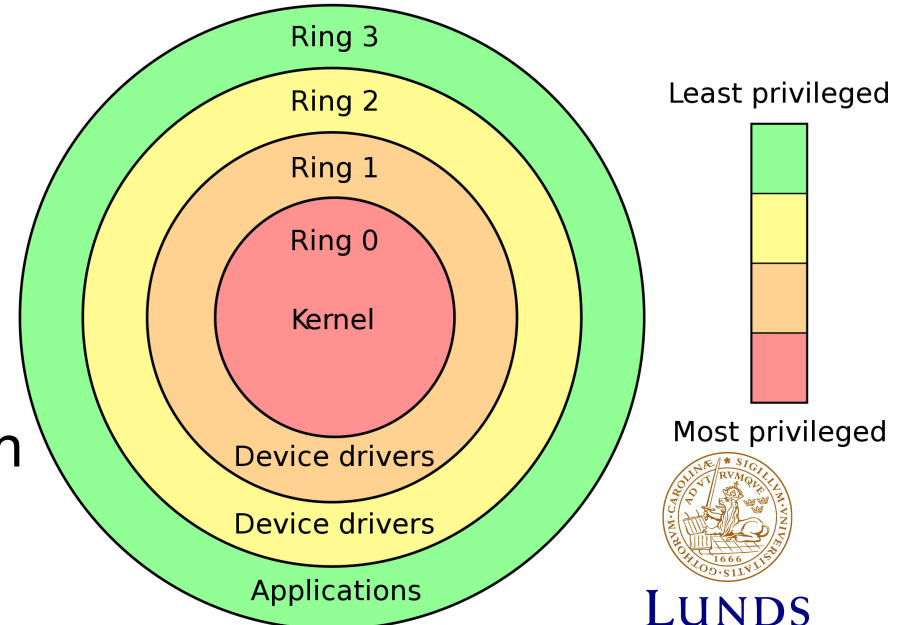
Minneshantering

- 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”:



Hantera hårdvaruresurser?

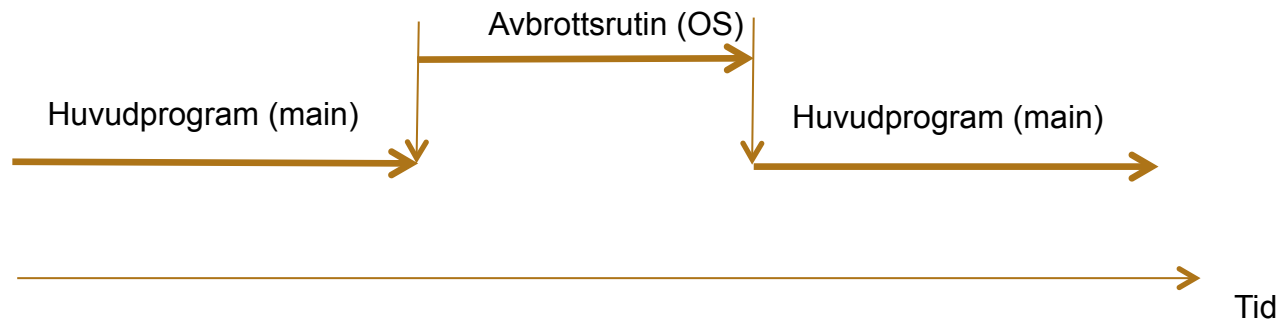
- Ringar (Rings) är hårdvarustöd för att ge skydd
- Typiskt med två moder
 - user-mode and supervisor-mode
- Applikationsprogram gör systemanrop för att läsa på hårddisk (ger OS kontroll)
- MS-DOS – endast supervisor-mode
- Windows, Linux – supervisor och user mode



Systemanrop

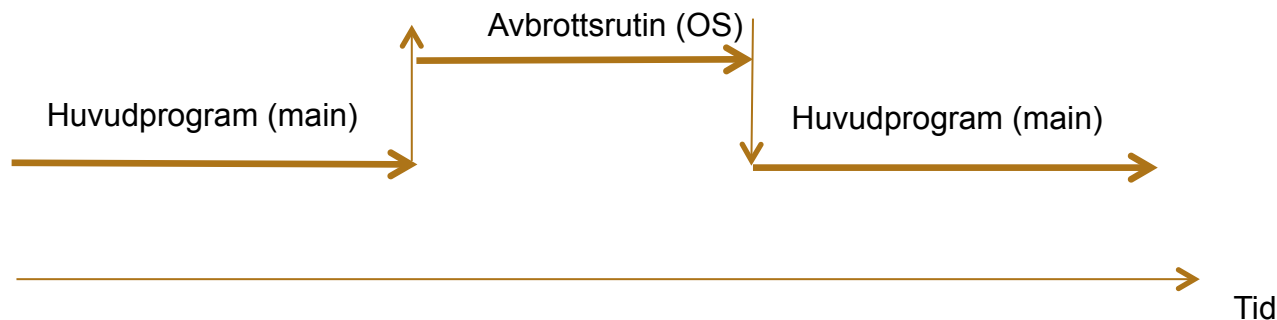
- Exempel: Byte av program (process)

Avbrott genererat av klocka (time slice)



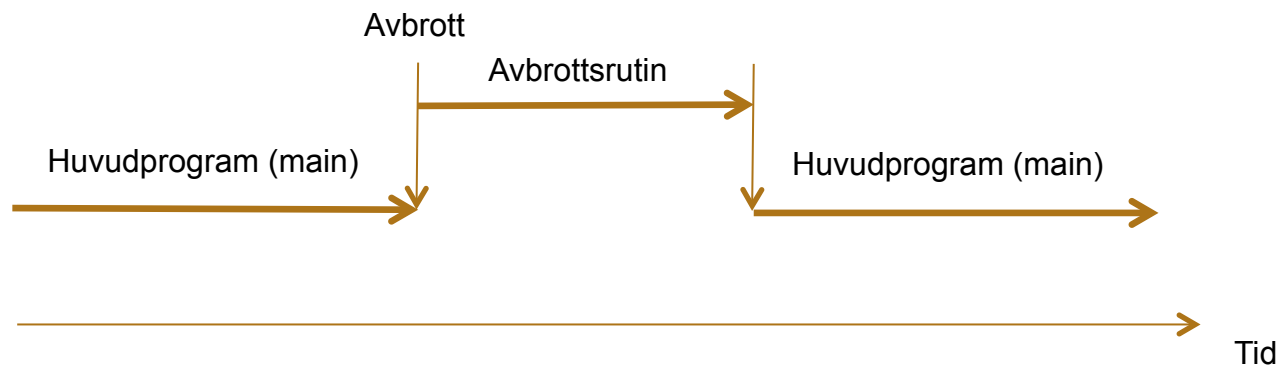
- Exempel C instruktionen: printf

Mjukvaruavbrott genererat av printf (systemanrop)



Polling/avbrott

- Polling – kontinuerlig avläsning av ingång, t ex tangent
 - Processorn slösar bort kraft på att kolla ingång
- Avbrott – ingång genererar avbrott
 - Processorn gör annat fram tills avbrott inträffar



Minneshantering

- Relocation
 - Flytta program och placera dem på andra ställen i minnet. Kunna hantera minnesreferenser och adresser vid omflyttningar.
- Minneskydd (Memory protection)
 - Processer ska inte kunna komma åt minnesarea som tilldelats andra processer utan lov
- Delning (Sharing)
 - Ibland ska processer kunna dela information och därför komma åt samma delar av minnet



Filsystem

- Hur hålla ordning på alla bitar?
- Vanligtvis kan man inte adressera individuella bitar (för stor overhead).
- Filer och bibliotek (Files and directories)
- Exempel: File Allocation Table (FAT), New Technology File System (NTFS)
- Mål: Kunna lagra stora filer, nå data snabbt (utnyttja hårddisk maximalt)

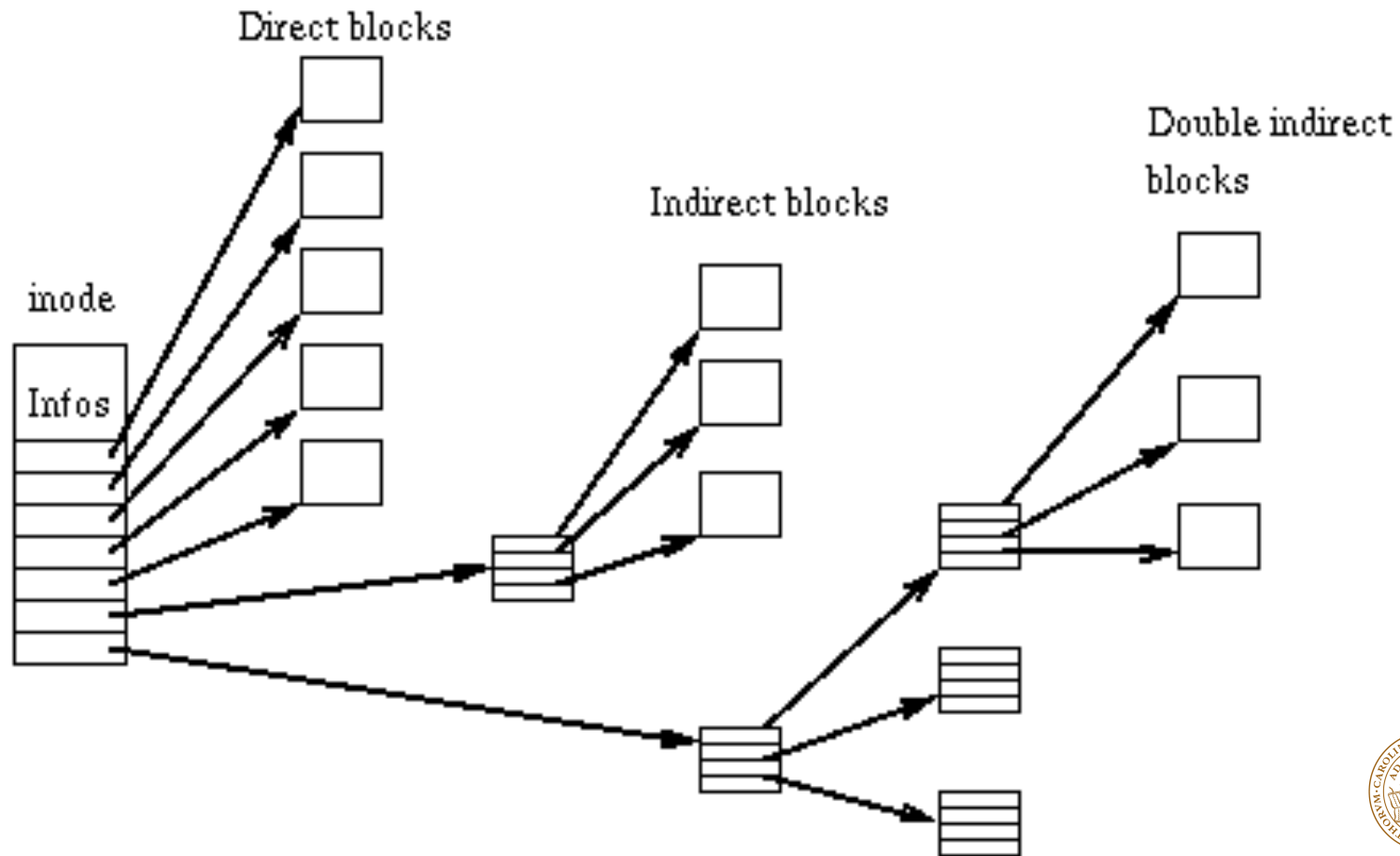


Filsystem - Inode

- Varje fil i Unix har en Inode
- Example:
 - 12 pekare som pekar direkt på block med filens data (direct pointers)
 - 1 indirekt pekare (en pekare till ett block av pekare)
 - 1 dubbel indirekt pekare (en pekare som pekar på ett block av pekare som i sin tur pekar på ett block av pekare som pekar på filens data)
 - 1 trippel indirekt pekare (som dubbel indirekt pekare men med ytterligare en nivå)



Filsystem - Inode



Vad gör ett OS?

- Processhantering (Process management)
- Minneshantering (Memory management)
- Filsystem (File system)
- Drivrutiner (Device drivers)
- Nätverk (Networking)
- Säkerhet (Security)
- In och utmatning (I/O)





LUNDS
UNIVERSITET