LUNDS UNIVERSITET

Datorteknik

ERIK LARSSON

Program

- Abstraktionsnivå:
  - Högnivåspråk
  - Assemblyspråk
  - Maskinspråk

Abstraktionsnivå

Högnivåspråk: datorns arkitektur osynlig

Assemblyspråk: datorns arkitektur synlig
Maskinspråk: datorns arkitektur synlig

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)

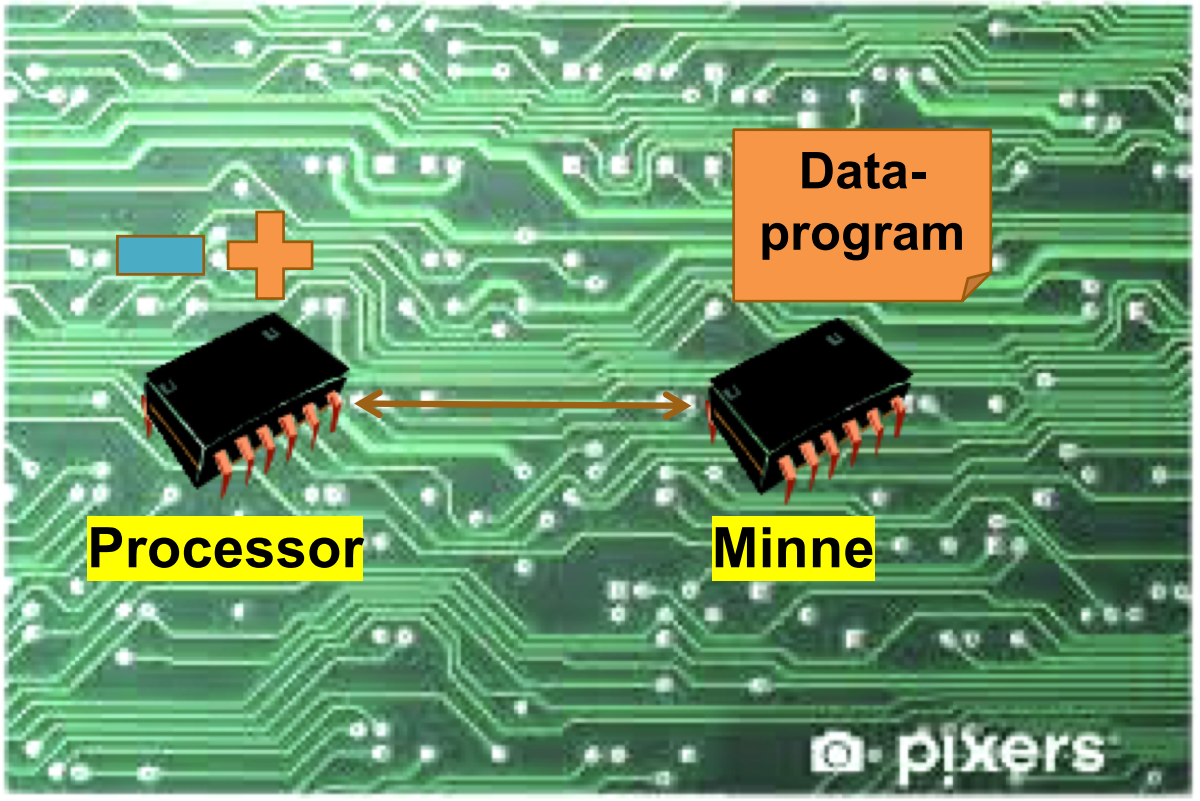
```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

Assembler

Binary machine language program (for MIPS)

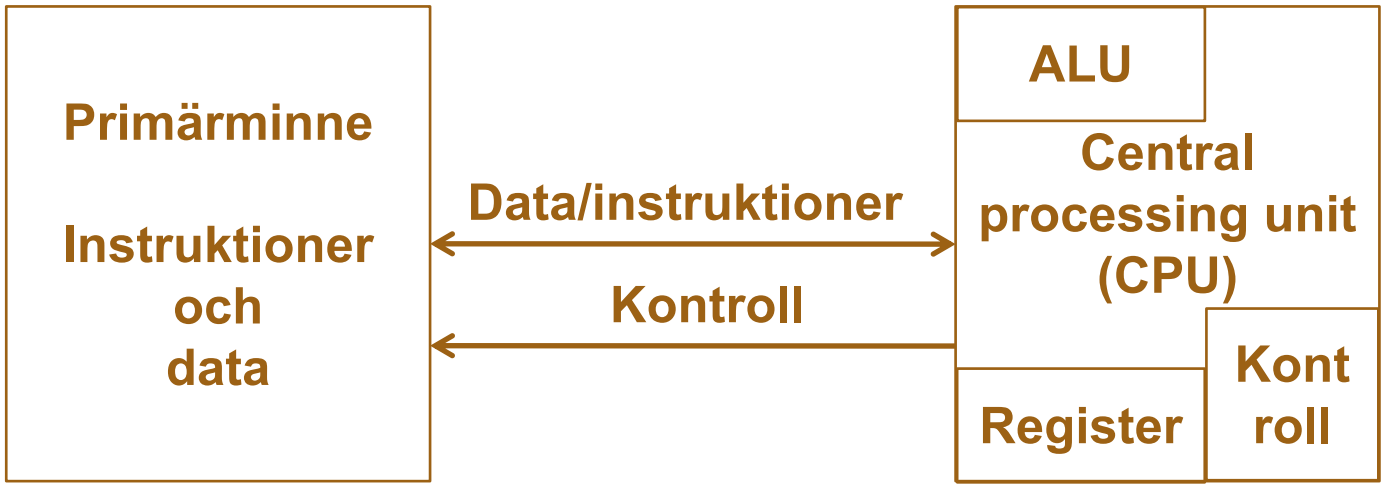
```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

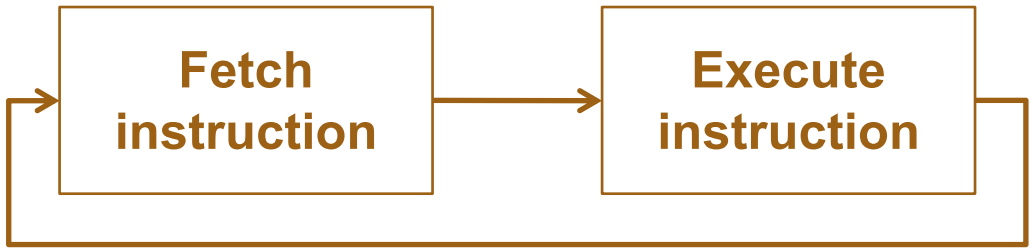
Dator



LUNDS UNIVERSITET

Dator



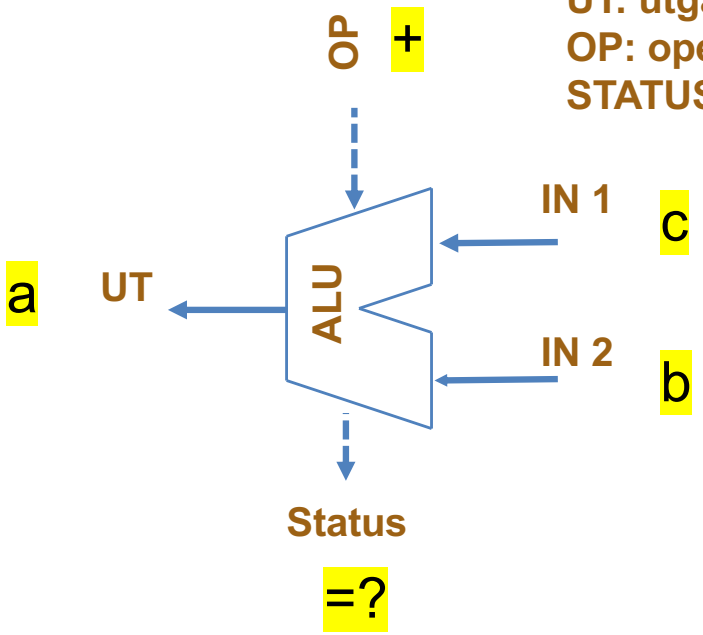


LUNDS
UNIVERSITET

ALU – arithmetic logic unit

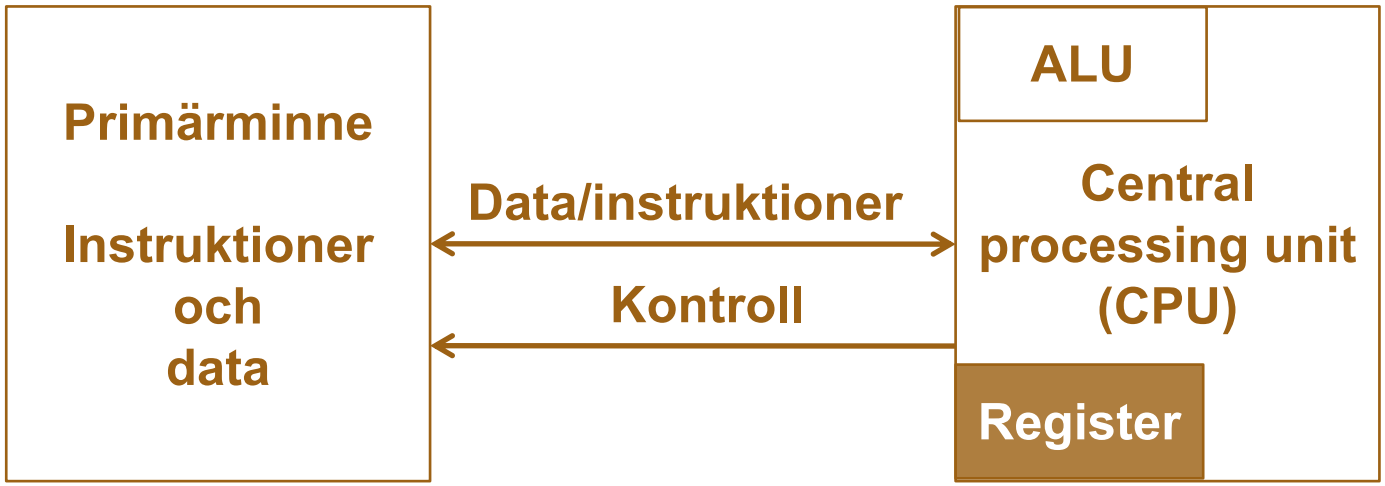
- Högnivåspråk: `a=b+c;`
- Assembly: `ADD a,b,c;`

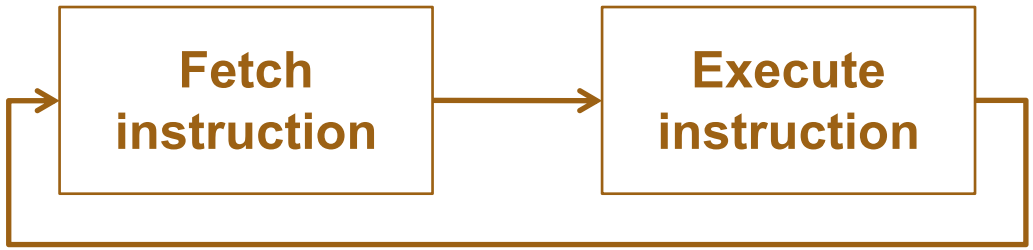
IN 1, IN 2: Ingångar
UT: utgång (resultat)
OP: operation (ADD, SUB, AND, ….
STATUS: Overflow, Zero, ….



LUNDS UNIVERSITET

Register





LUNDS UNIVERSITET

Register

- Högnivåspråk: a=b+c+d;
- Alternativ 1:

```
ADD a, b, c        //adderar b och c, resultat i a
ADD a, a, d        //adderar a (som är b+c) med d,
                     resultat i a
```

- Antal minnesaccesser:

  6=3+3

- Tid för en minnesaccess: 100 ns gör att minnesläsning/skrivning tar 600 ns

LUNDS UNIVERSITET

Register

- Högnivåspråk: a=b+c+d;
- Alternativ 2:

```
ADD r1, b, c        //adderar b och c, resultat i register 1
ADD a, r1, d        //adderar a (som är i r1) med d,
                      resultat i a
```

- Antal minnesaccesser:

  4=2+2

- Tid för en:
  - minnesaccess: 100 ns
  - registeraccess: 1 ns:
- Minnesläsning/skrivning tar 400 + 2 ns = 402 ns

30% sparad tid
(600-402)/600

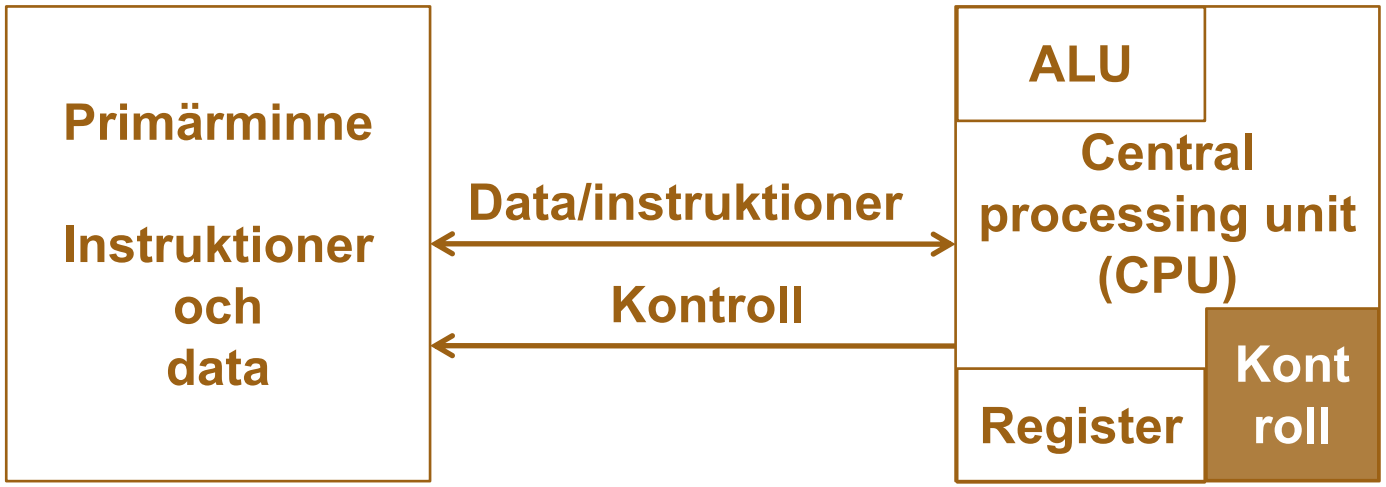
LUNDS UNIVERSITET

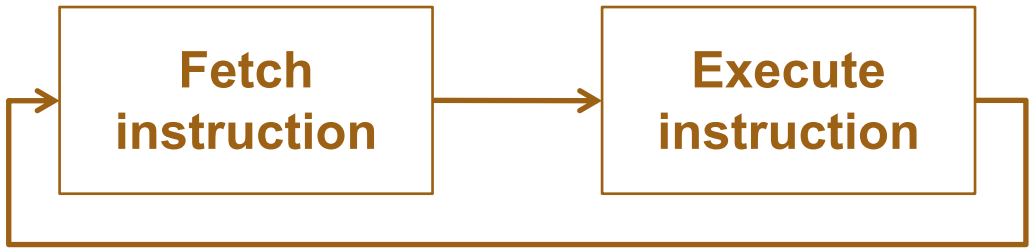
Register

- Register synliga för användare, som kan vara:
  - Helt generella, eller
  - Specifika för data och instruktioner och/eller
  - Specifika för heltal och flyttal
  - Speciella för t ex multiplikation/division för att kunna hantera resultat eller för adressberäkningar: basregister och stackpekare
- Inte direkt kontrollerbara av programmeraren, t ex:
  - Programräknare (program counter (PC))
  - Instruktions register (IR)
  - Program status word (PSW)

LUNDS UNIVERSITET

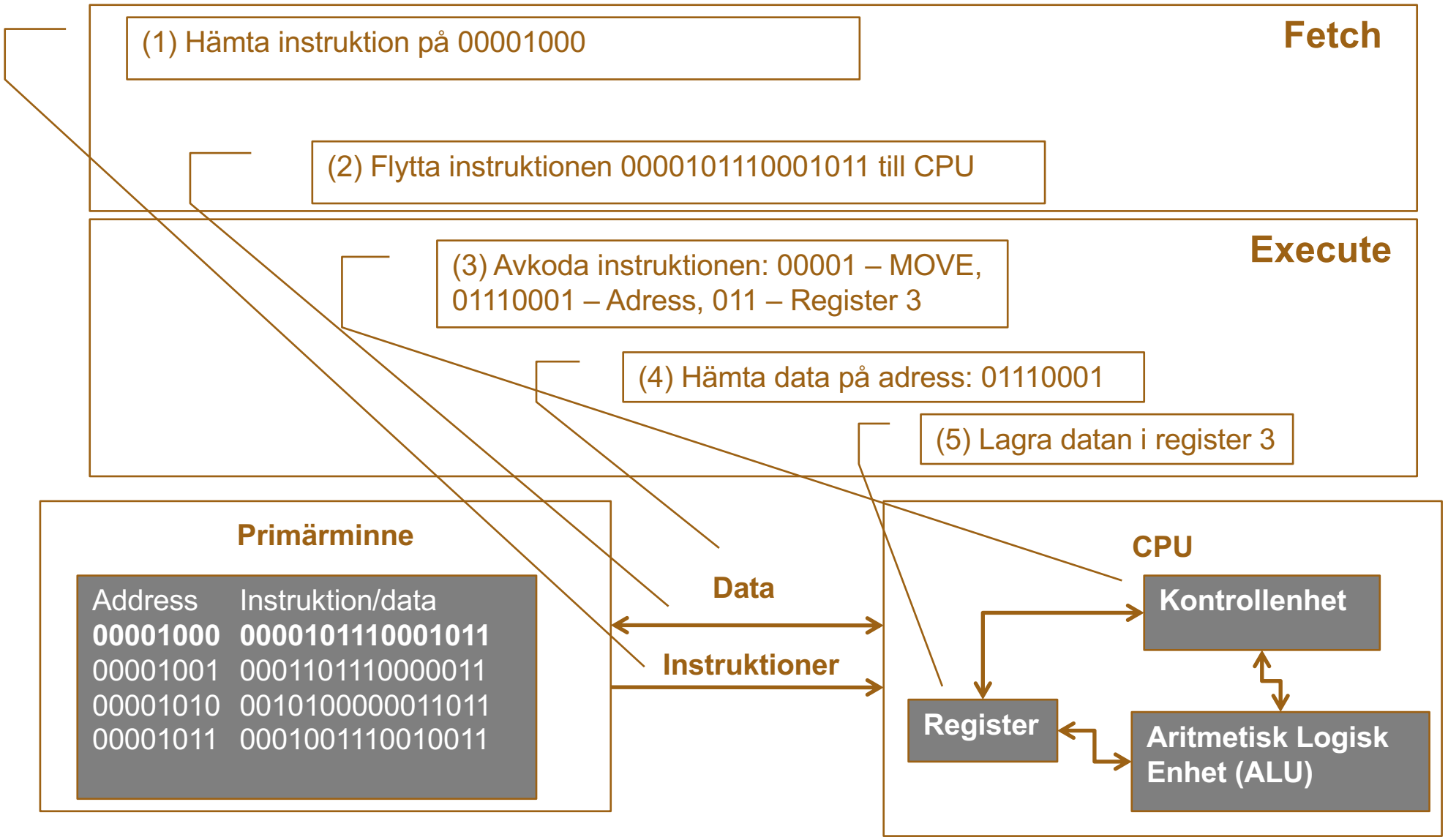
Kontrollenhet



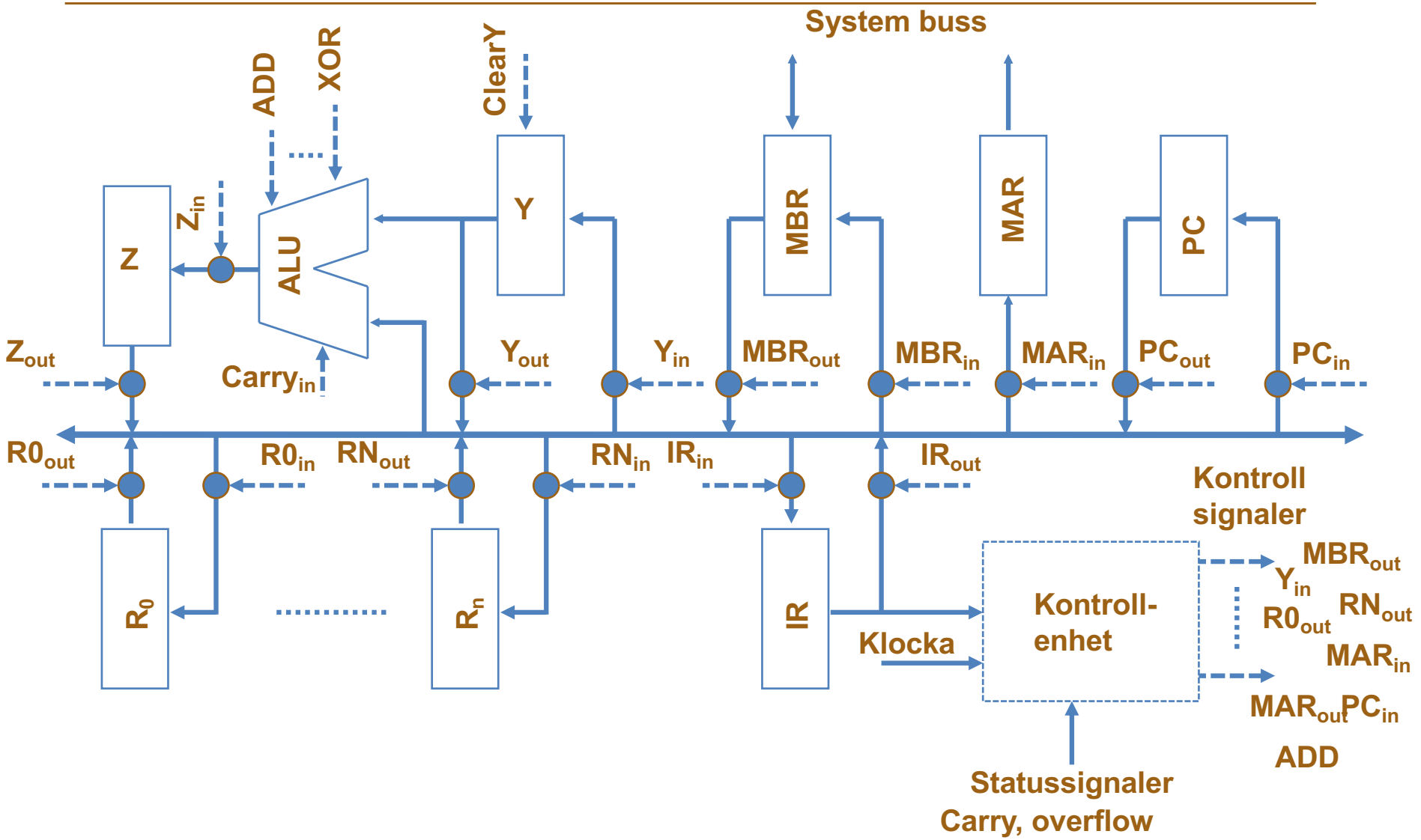


LUNDS
UNIVERSITET

Dator


Fetch
(1) Hämta instruktion på 00001000
(2) Flytta instruktionen 0000101110001011 till CPU
Execute
(3) Avkoda instruktionen: 00001 – MOVE, 01110001 – Adress, 011 – Register 3
(4) Hämta data på adress: 01110001
(5) Lagra datan i register 3
Primärminne
Address Instruktion/data
00001000 0000101110001011
00001001 0001101110000011
00001010 0010100000011011
00001011 0001001110010011
Data
Instruktioner
CPU
Kontrollenhet
Register
Aritmetisk Logisk Enhet (ALU)

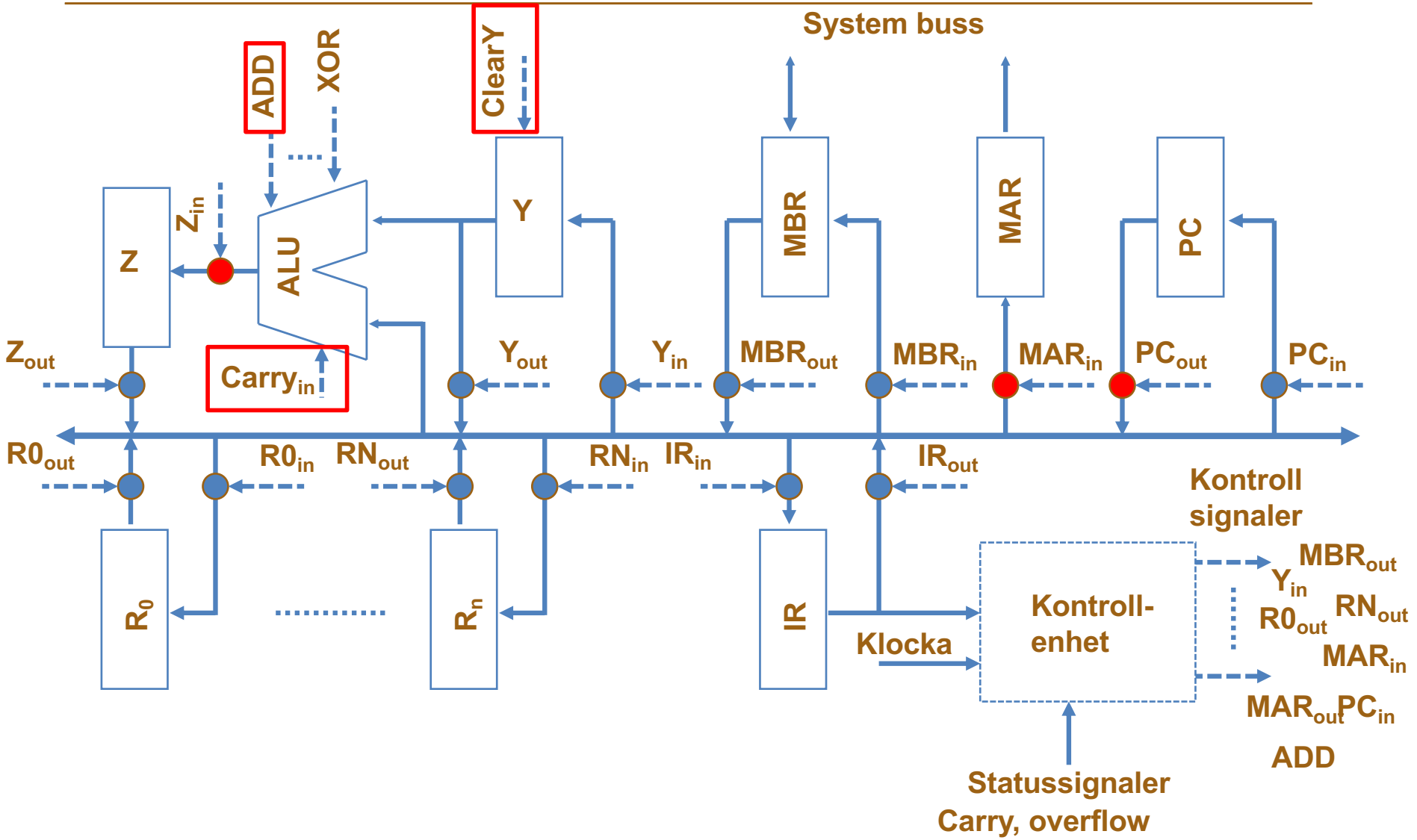
Exekvering av en instruktion



Exekvering av en instruktion

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

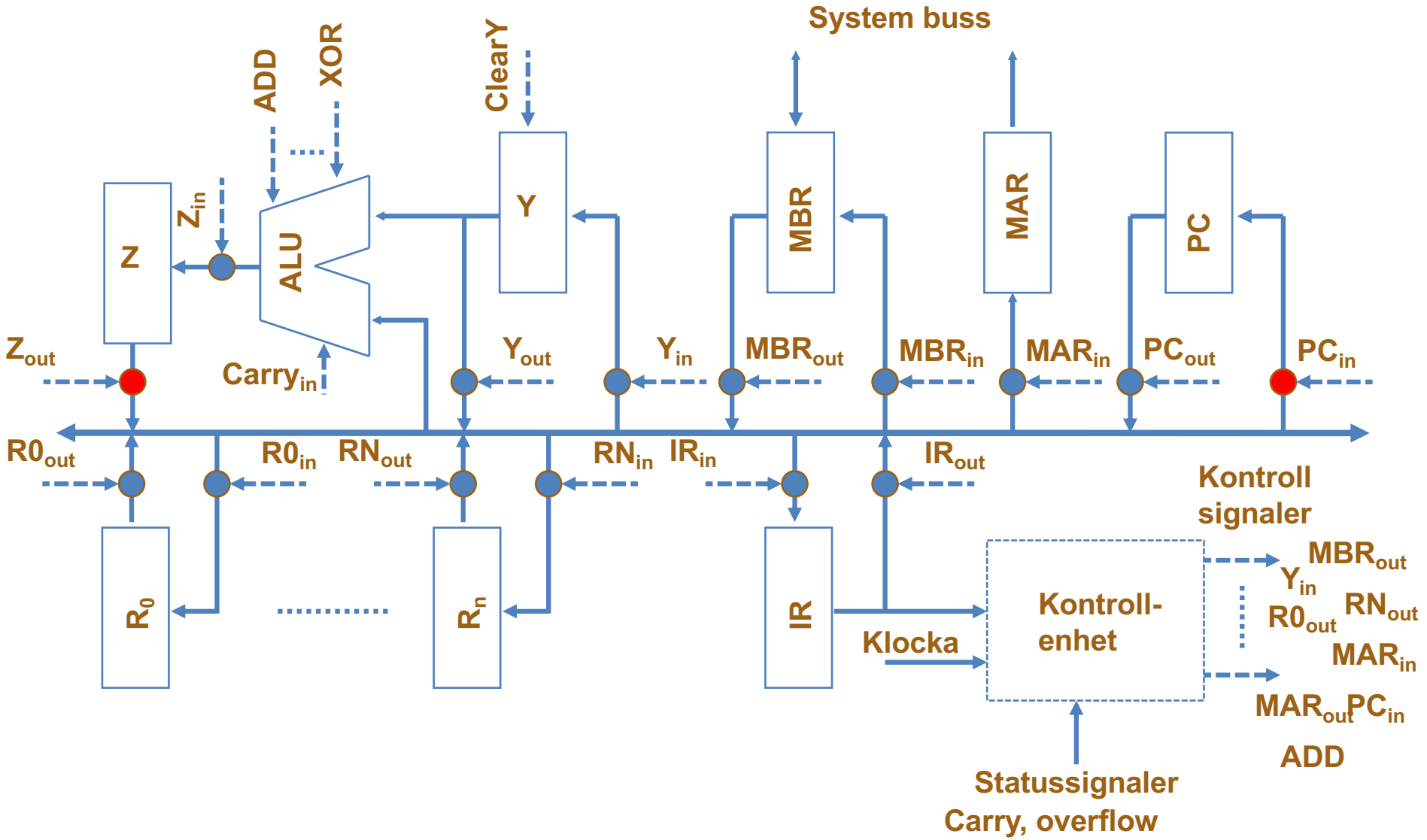
Exekvering av en instruktion



Exekvering av en instruktion

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

Exekvering av en instruktion

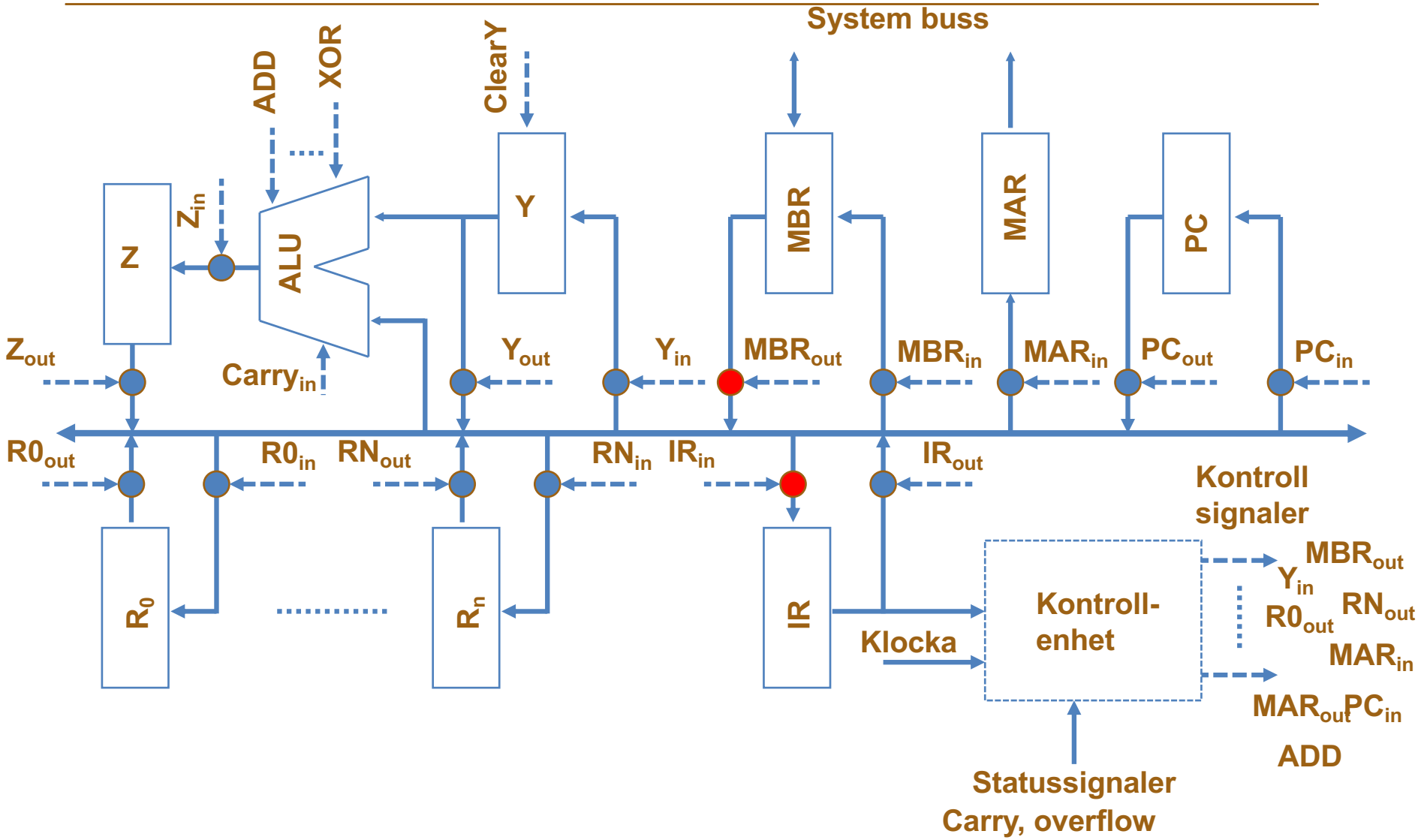


Exekvering av en instruktion

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. $\underline{MBR_{out}, IR_{in}}$
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

LUNDS UNIVERSITET

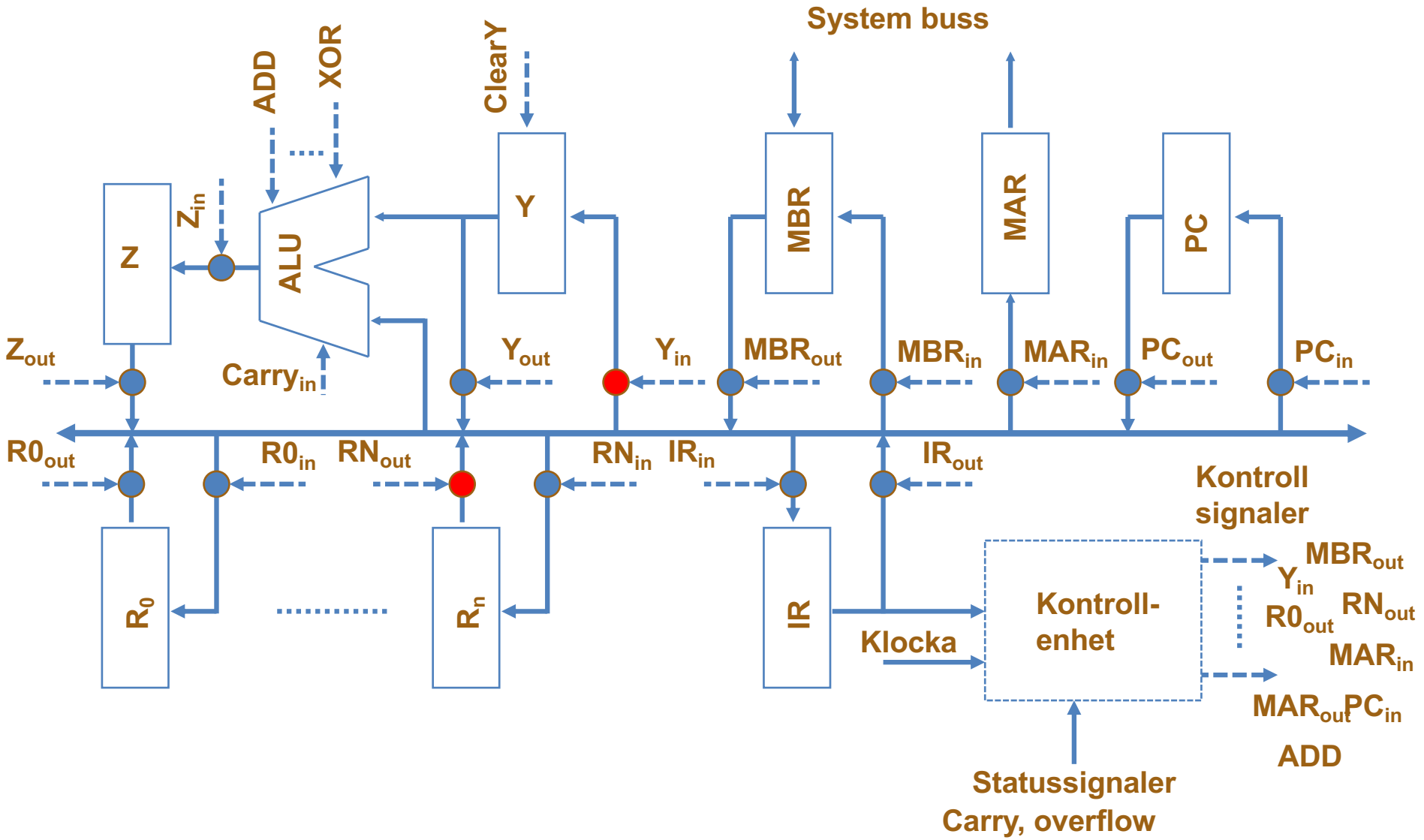
Exekvering av en instruktion



Exekvering av en instruktion

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

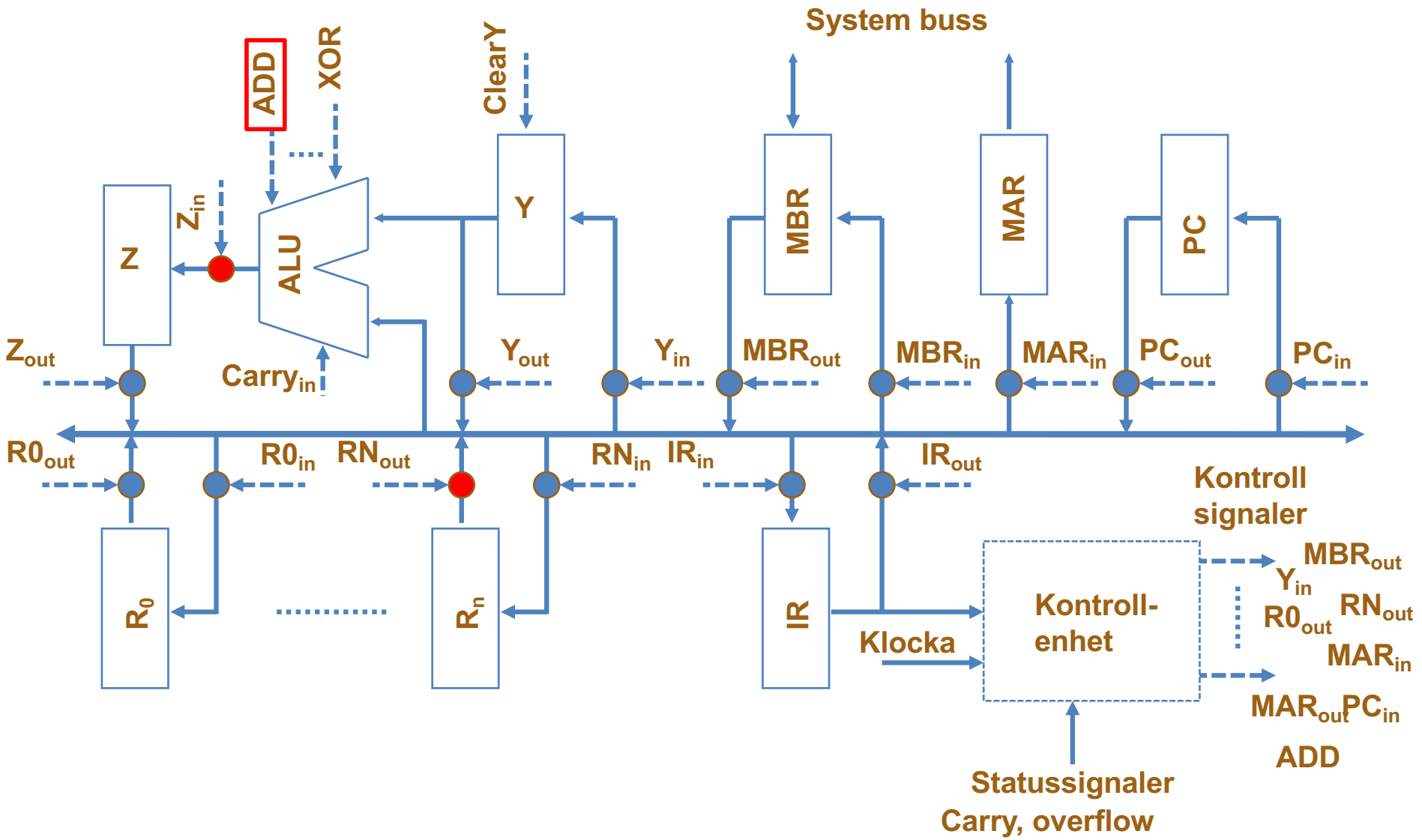
Exekvering av en instruktion



Exekvering av en instruktion

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $\underline{R3_{out}, \text{ Add}, Z_{in}}$
  6. Z_{out}, $R1_{in}$, End

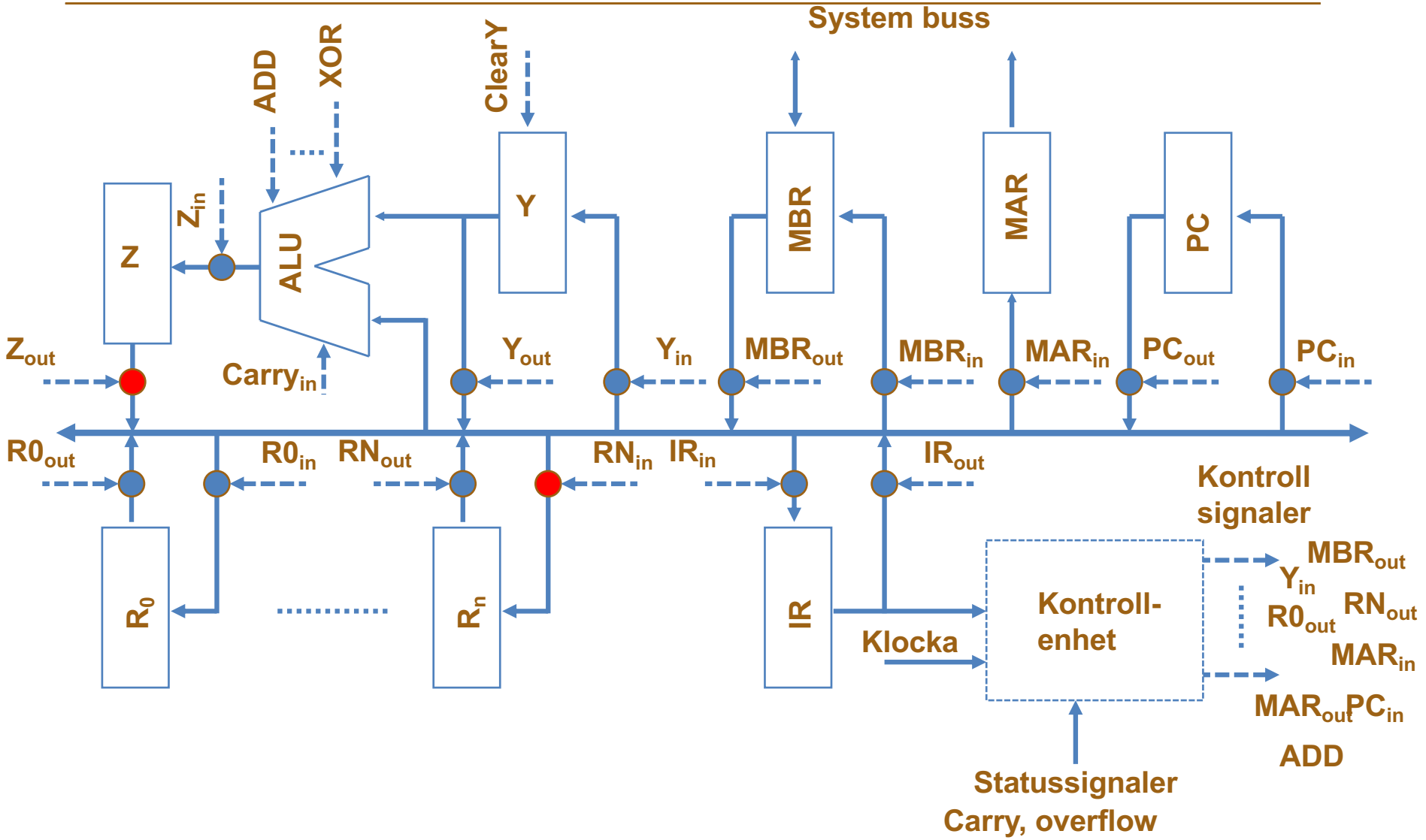
Exekvering av en instruktion



Exekvering av en instruktion

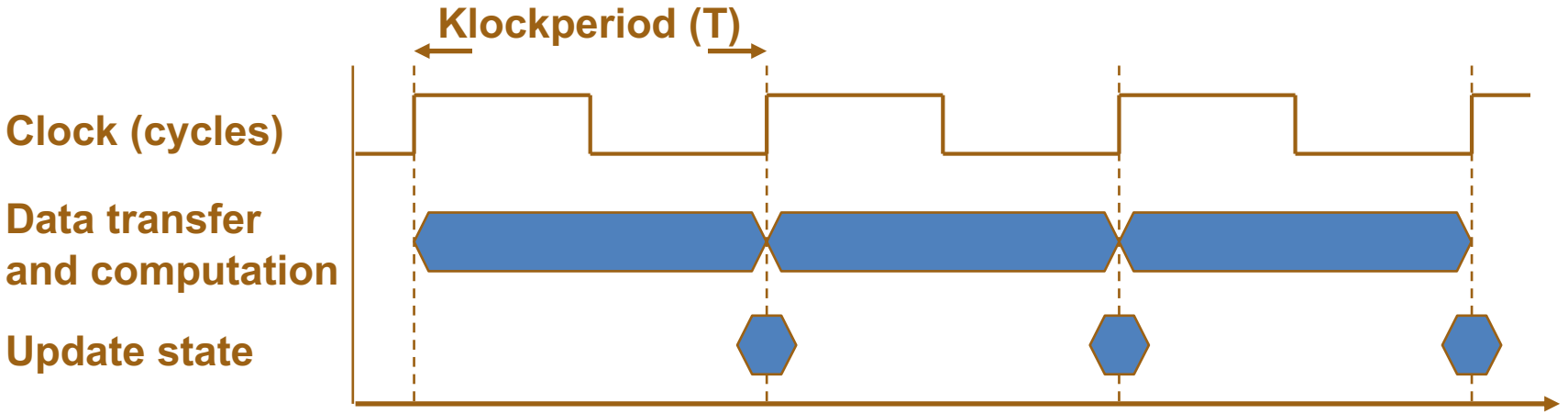
- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

Exekvering av en instruktion



Klockning

- En klocka styr hastigheten på digital hårdvara

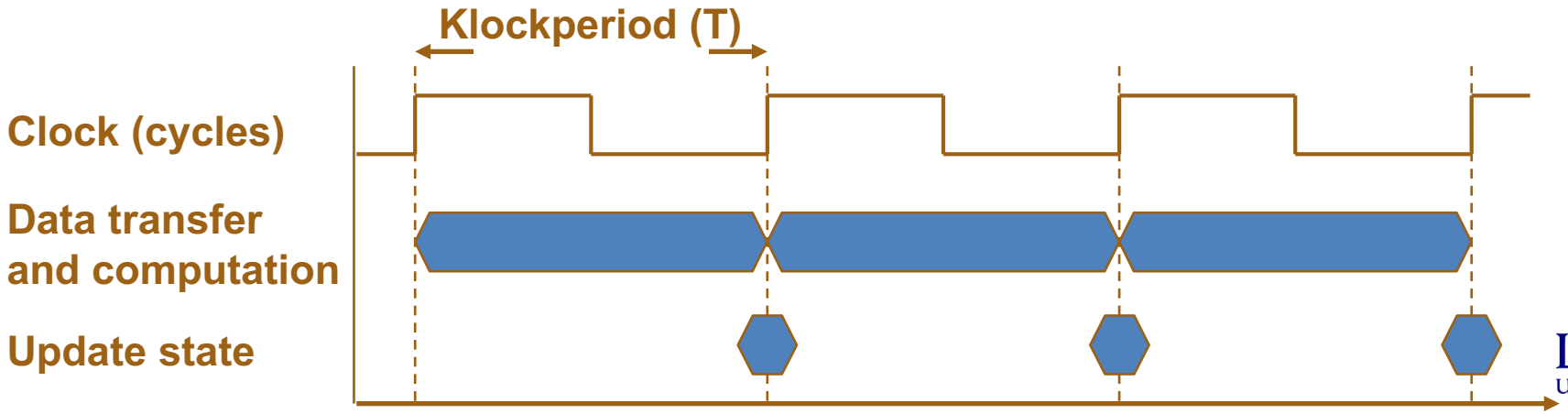


- Klockperiod: tid för en klockcykel (T)
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Klockfrekvens (rate): cycles per second (f)
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz
- Samband: f=1/T

LUNDS UNIVERSITET

Klockning

År	Intel 4004	Apple A12	Skillnad
Lansering	1971	2018	47 år
Transistorer	2300	6.9 miljarder	3 miljoner
Klockfrekvens	740KHz	2.49GHz	3364
Klockperiod	1350000ps	400ps	3364
Teknologi	10µm	7 nm	700



LUNDS UNIVERSITET

Exekveringstid

- Antal klockcykler för att exekvera en maskininstruktion
  - Clocks per instruction (CPI)
- Tid för en klockcykel
  - Tid för en klockperiod (T)
  - Frekvens (f) är: f=1/T
- Antal maskininstruktioner
  - Instruction count (IC)
- Exekveringstid i klockcykler = CPI x T x IC

LUNDS UNIVERSITET

Exekveringstid

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg = Klockcykler per instruktion (CPI)
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

Exekveringstid

- Prestanda kan ökas genom:
  - Öka klockfrekvensen (f) (minska T)
  - Minska antal instruktioner (IC)
  - Minska antal klockcykler per instruktion (CPI)

LUNDS UNIVERSITET

Exekveringstid

- MIPS (akronym av ”miljoner instruktioner per sekund” eller engelska ”million instructions per second”)

Processor	MIPS	År
Intel 8080 vid 2 MHz	0,640	1974
Motorola 68000 vid 8 MHz	1	1979
ARM 7500FE vid 40 MHz	35,9	1996
Intel 486DX vid 66 MHz	54	1992
Zilog eZ80 vid 50 MHz	80	1999
ARM10 vid 300 MHz	400	1998
IBM PowerPC 440GP vid 500 MHz	1000	2002
Athlon 64 vid 2,8 GHz	8400	2005

LUNDS UNIVERSITET

Prestanda

- Algoritm
  - Bestämmer vilka och hur många *operationer* som ska utföras
- Programmeringsspråk, kompilator, arkitektur
  - Bestämmer hur många *maskininstruktioner* som ska utföras per *operation*
- Processor och minnessystem
  - Bestämmer hur snabbt instruktioner exekveras
- I/O (Input/Output) och operativsystem
  - Bestämmer hur snabbt I/O operationer ska exekveras

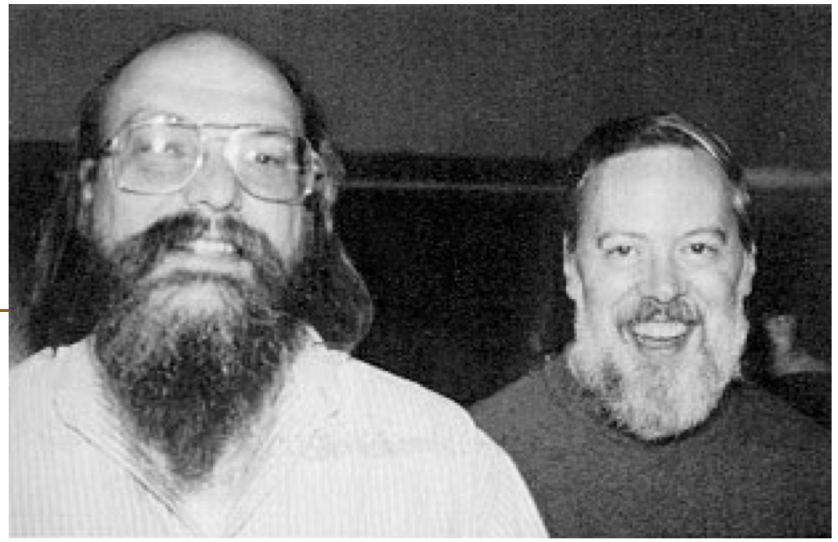
LUNDS UNIVERSITET

LUNDS UNIVERSITET

Datorteknik

ERIK LARSSON

C - Inledning



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

LUNDS UNIVERSITET

Datatyper

Datatyp	Antal bytes	Talområde
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	±1,18 E-38 — 3,39 E+38

LUNDS UNIVERSITET

Datatyper

MSB LSB

Givet: **1111 1111** **0000 1111** **1111 0000** **1010 1010**

Big-endian	
Adress	Data
….	….
n	1111 1111
n+1	0000 1111
n+2	1111 0000
n+3	1010 1010
….	….

Little-endian	
Adress	Data
….	….
n	1010 1010
n+1	1111 0000
n+2	0000 1111
n+3	1111 1111
….	….

LUNDS UNIVERSITET

Datatyper

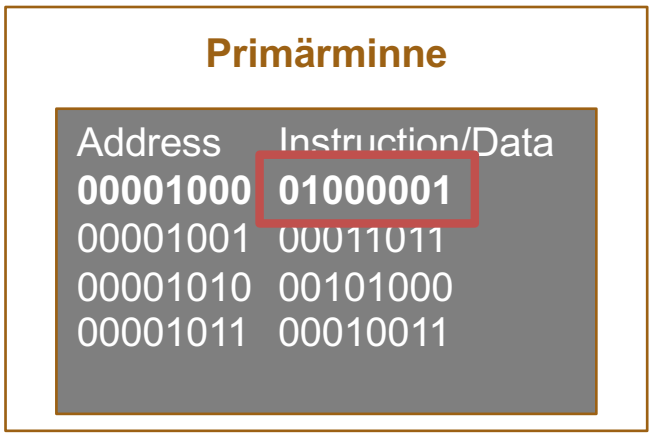
- Samma sak lagras:

```
char a = 65;
```
decimalt

```
char a = 0x41;
```
hexadecimalt

```
char a = 0b01000001;
```
binärt

```
char a = 'A';
```
ASCII-kod



LUNDS UNIVERSITET

Tilldelningssatser

- Deklarera variabel:

```
unsigned char a;
```

- Tilldela variablen ett värde:

```
a = 5;
```

- Addera 2 till värdet i a:

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

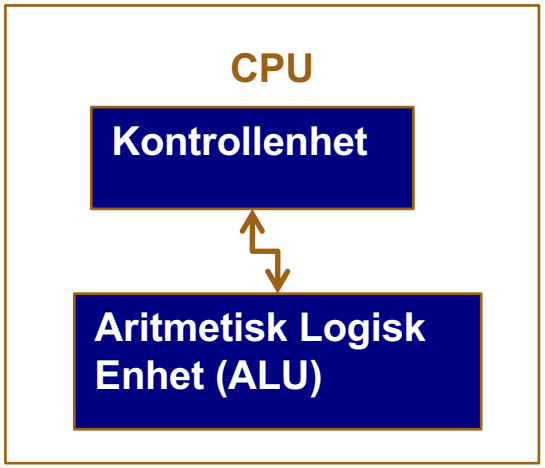
Primärminne

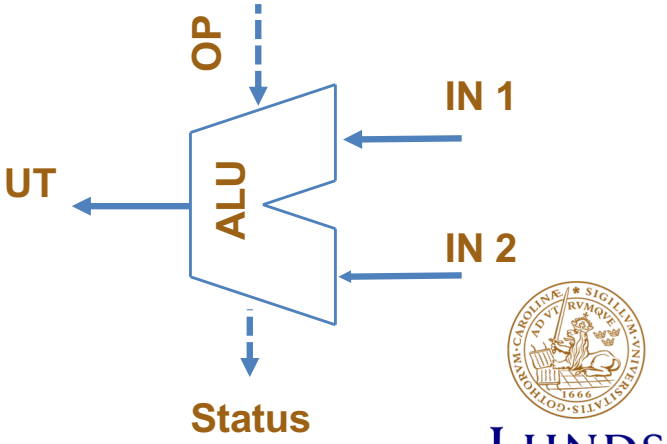
Address	Instruction/Data
00001000	**00001111**
00001001	00011011
00001010	00101000
00001011	00010011

LUNDS UNIVERSITET

Aritmetiska operationer

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

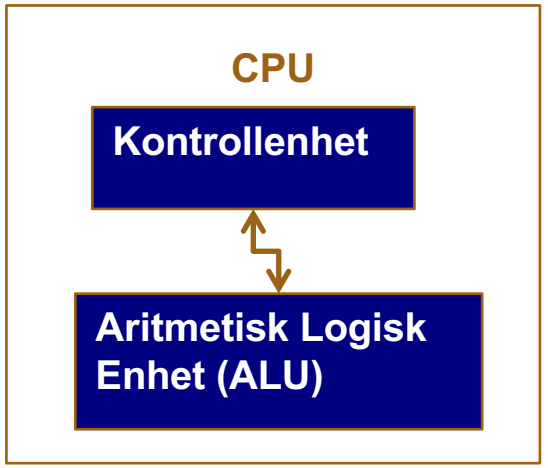


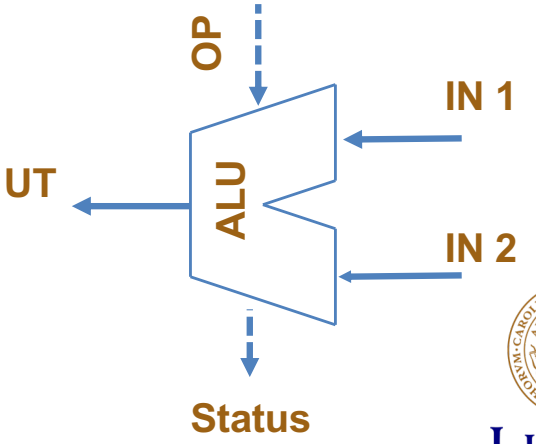


LUNDS UNIVERSITET

Bithantering

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





LUNDS UNIVERSITET

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 */
```

LUNDS UNIVERSITET

Villkor

- `if ( villkor ) sats;`
- Exempel 1:

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

- Exempel 2:

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

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

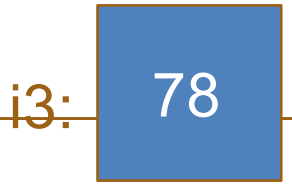
- En funktion deklareras:

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

Datatyp som returneras

Gör/skapar retur värdet

Pekare (Intro)



- Deklarationen:

```
int i3, i4
```

- Ger att:

  i3 och i4 är heltalsvaribler

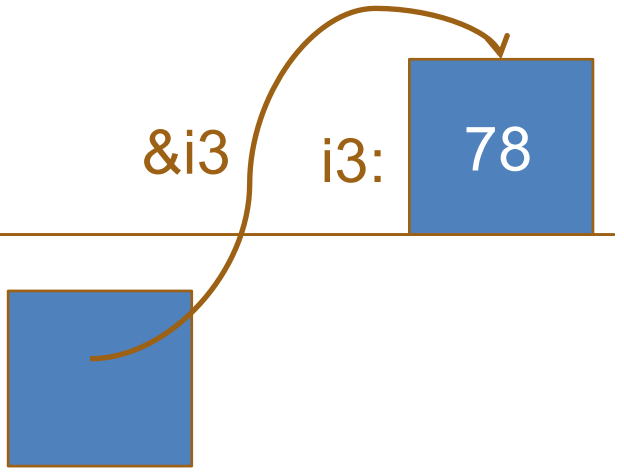
- Exempel:

```
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



- Deklarationen:

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

- Ger att:

  *ip1 och *ip2 är heltalspekarvariabler
  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

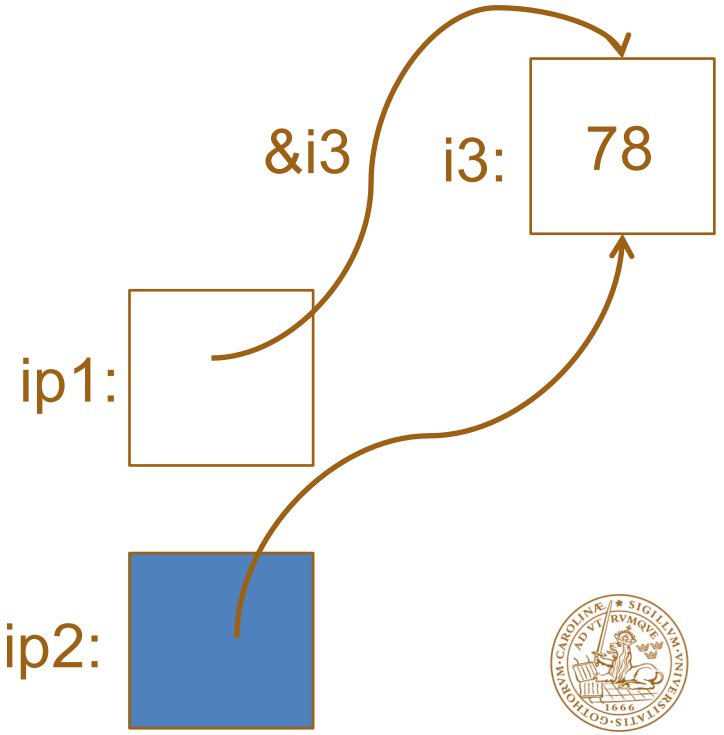
Adress	Data
0 (i3)	78
1 (i4)	5
2 (ip1)	0
3 (ip2)	
4	
5	
6	

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
```



LUNDS UNIVERSITET

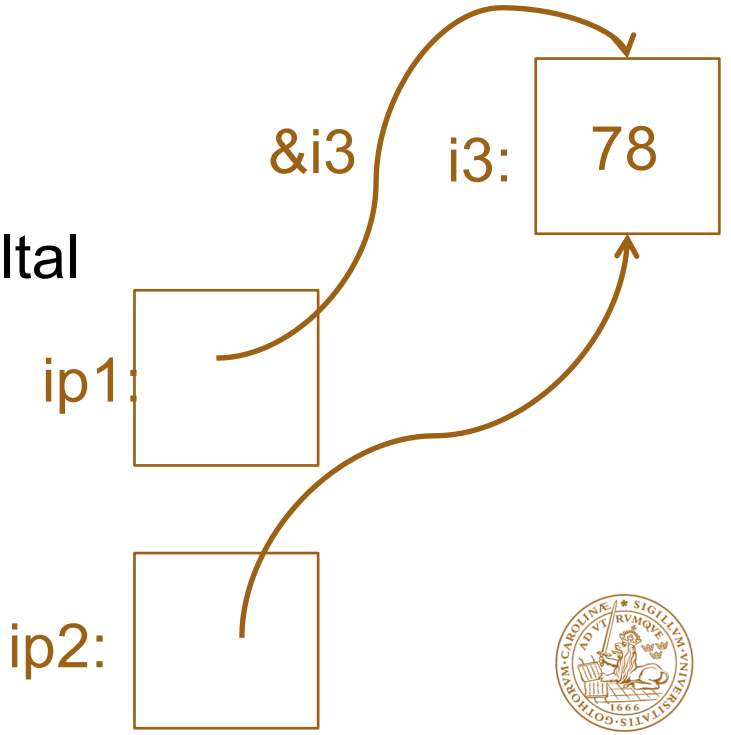
Pekare

- 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



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.

LUNDS UNIVERSITET

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)	4
1 (b)	5
2 (tmp)	
3	
4	10
5	5
6	

tmp=*a;

(tilldelar tmp det som *a pekar på)

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

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

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

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

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

Variablers synlighet

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

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


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

Global variabel

Global variabel

Lokal variabel

Lokal variabel

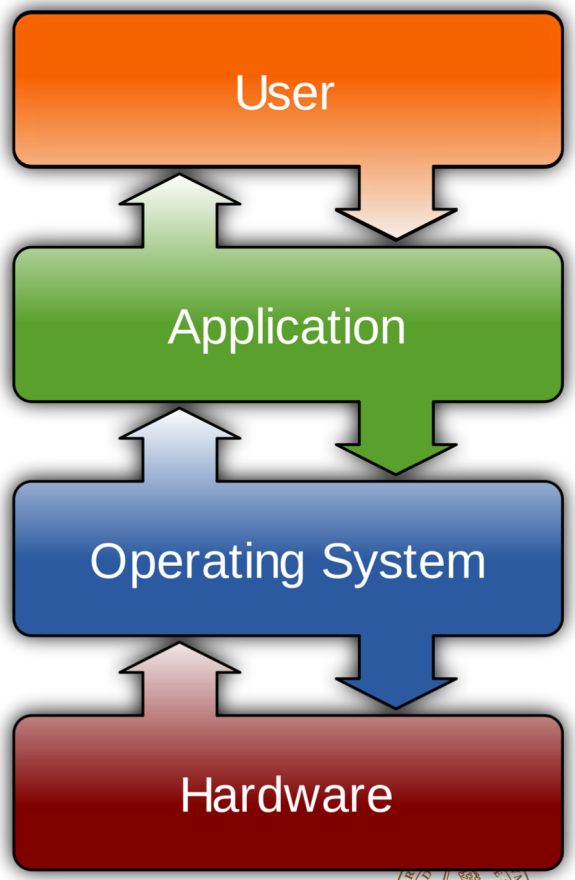
Include

- Includeringsbara bibliotek:

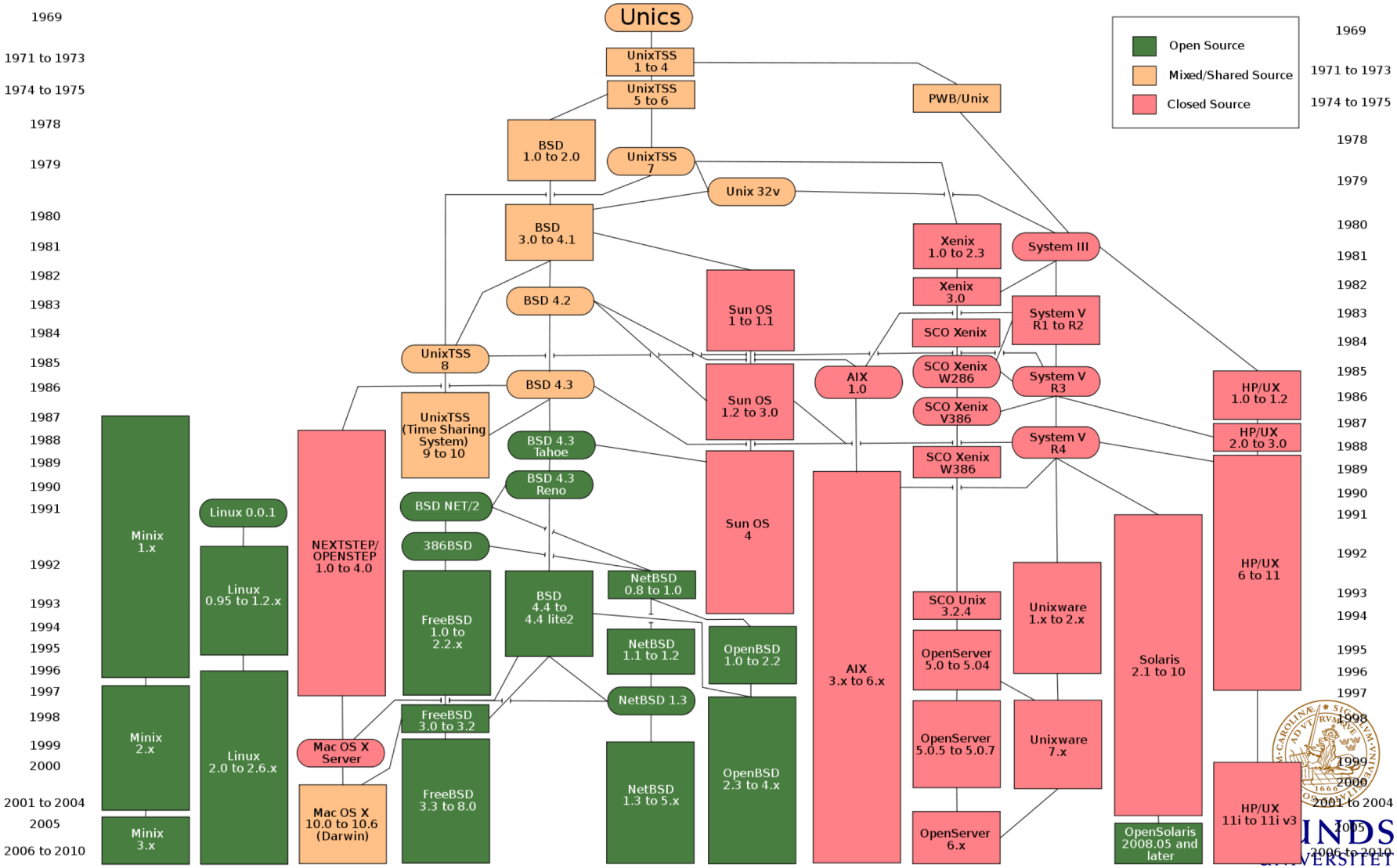
  `#include <stdio.h>` standardfunktioner för I/O

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 from av OS – från mobiltelefoner, datorspel, till superdatorer.



Unix


Unics
UnixTSS 1 to 4
UnixTSS 5 to 6
PWB/Unix
BSD 1.0 to 2.0
UnixTSS 7
Unix 32v
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
Sun OS 1 to 1.1
System V R1 to R2
SCO Xenix
UnixTSS 8
AIX 1.0
SCO Xenix W286
System V R3
BSD 4.3
Sun OS 1.2 to 3.0
SCO Xenix V386
HP/UX 1.0 to 1.2
UnixTSS (Time Sharing System) 9 to 10
BSD 4.3 Tahoe
SCO Xenix W386
System V R4
HP/UX 2.0 to 3.0
BSD 4.3 Reno
BSD NET/2
Sun OS 4
386BSD
HP/UX 6 to 11
Minix 1.x
Linux 0.0.1
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Linux 0.95 to 1.2.x
FreeBSD 1.0 to 2.2.x
BSD 4.4 to 4.4 lite2
NetBSD 0.8 to 1.0
AIX 3.x to 6.x
SCO Unix 3.2.4
Unixware 1.x to 2.x
Solaris 2.1 to 10
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
OpenServer 5.0 to 5.04
Minix 2.x
Linux 2.0 to 2.6.x
FreeBSD 3.0 to 3.2
NetBSD 1.3
OpenServer 5.0.5 to 5.0.7
Unixware 7.x
Mac OS X Server
OpenBSD 2.3 to 4.x
HP/UX 11i to 11i v3
Mac OS X 10.0 to 10.6 (Darwin)
FreeBSD 3.3 to 8.0
NetBSD 1.3 to 5.x
OpenServer 6.x
OpenSolaris 2008.05 and later
Minix 3.x
Open Source
Mixed/Shared Source
Closed Source
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2010

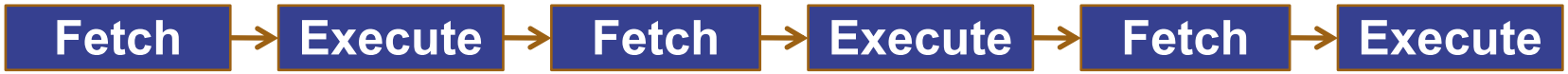
Linux

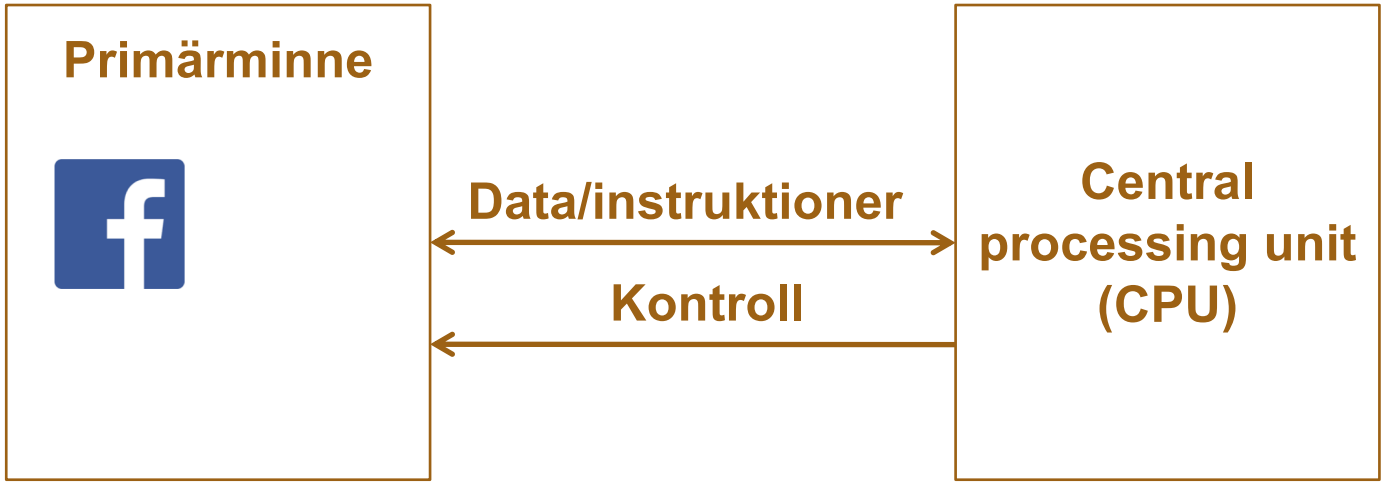
- Linux
  - Linus Benedict Torvalds, född i Finland
  - Ville lära sig om OS, skrev ett OS
- Windows
  - MS-DOS (Microsoft Disk Operating System) (~1980)
  - Windows 1.0, Windows 95, 98, 2000, XP, Vista, 7, 8

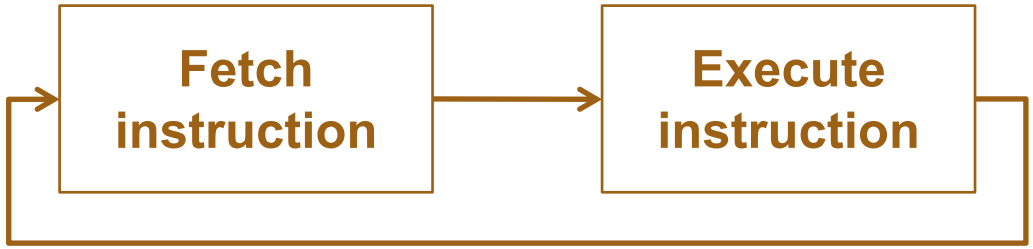
Bill Gates **Paul Allen**

LUNDS UNIVERSITET

Program

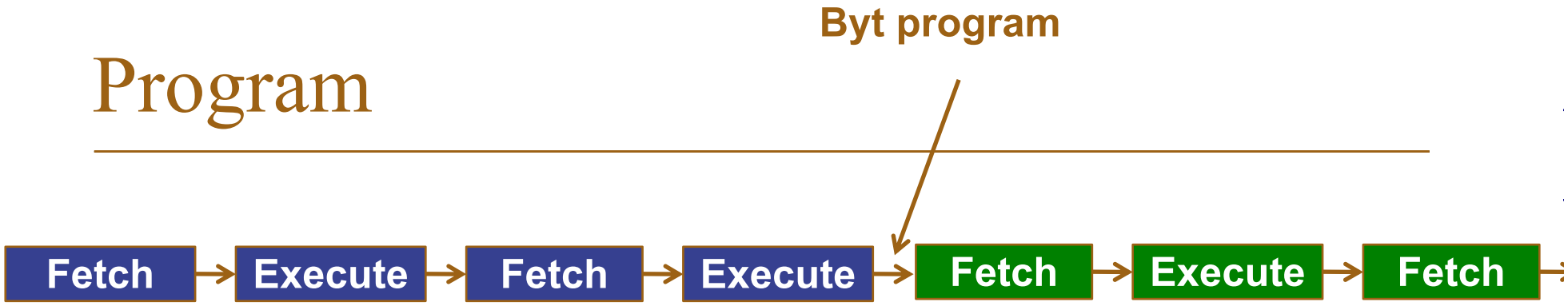


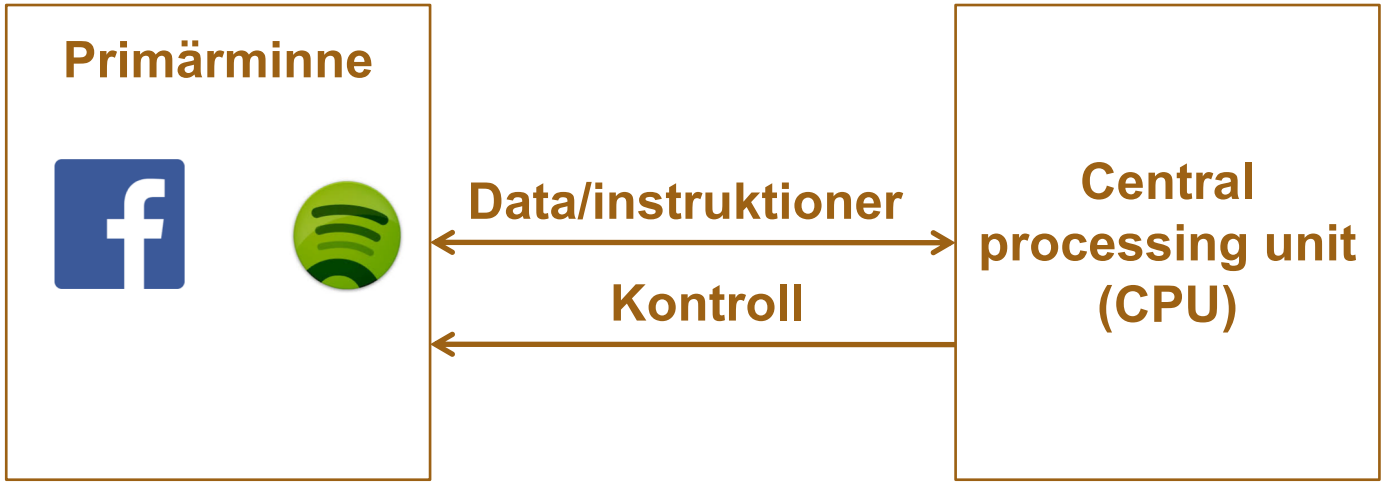


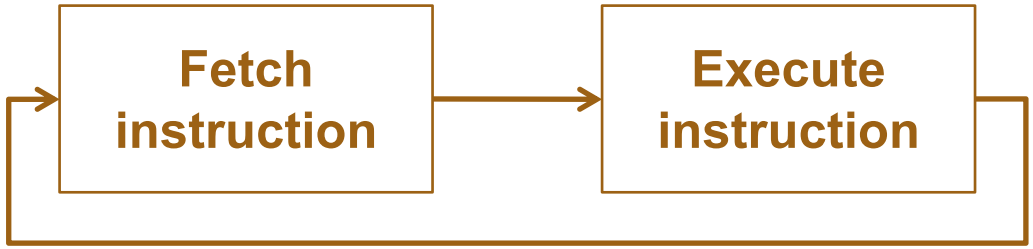


LUNDS UNIVERSITET

Program

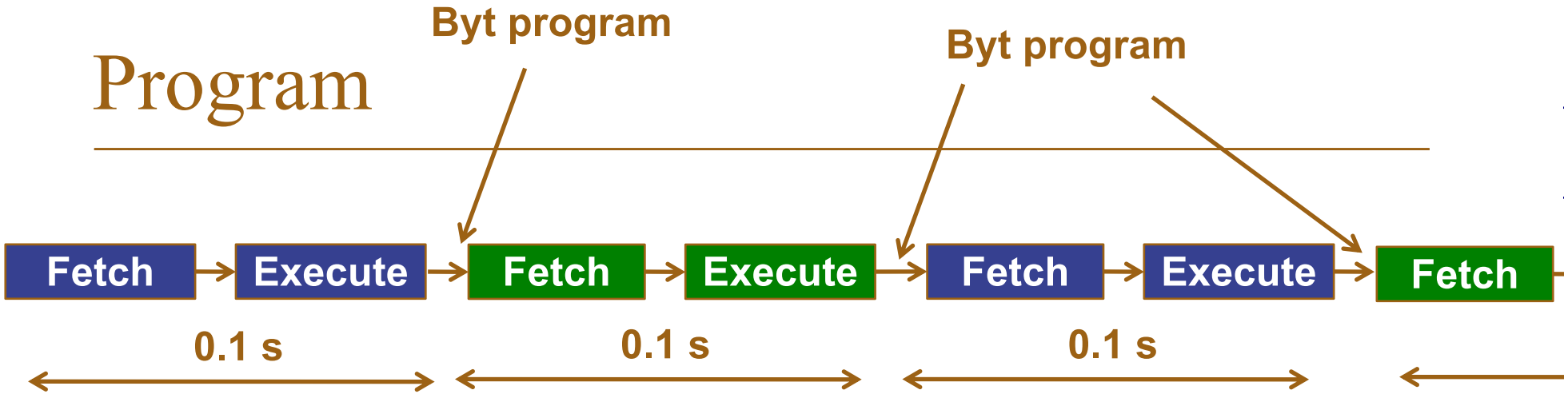


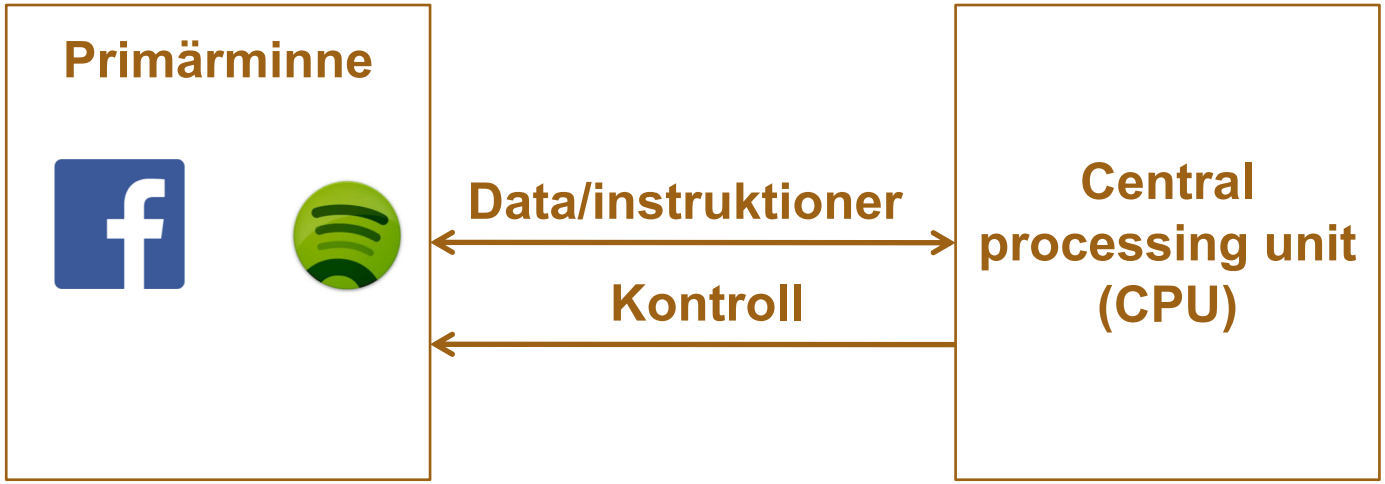


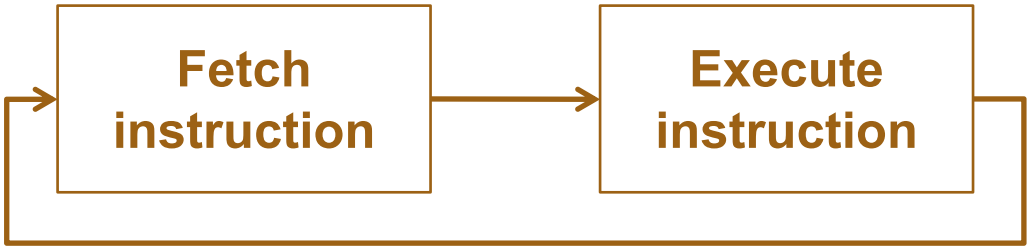


LUNDS UNIVERSITET

Program







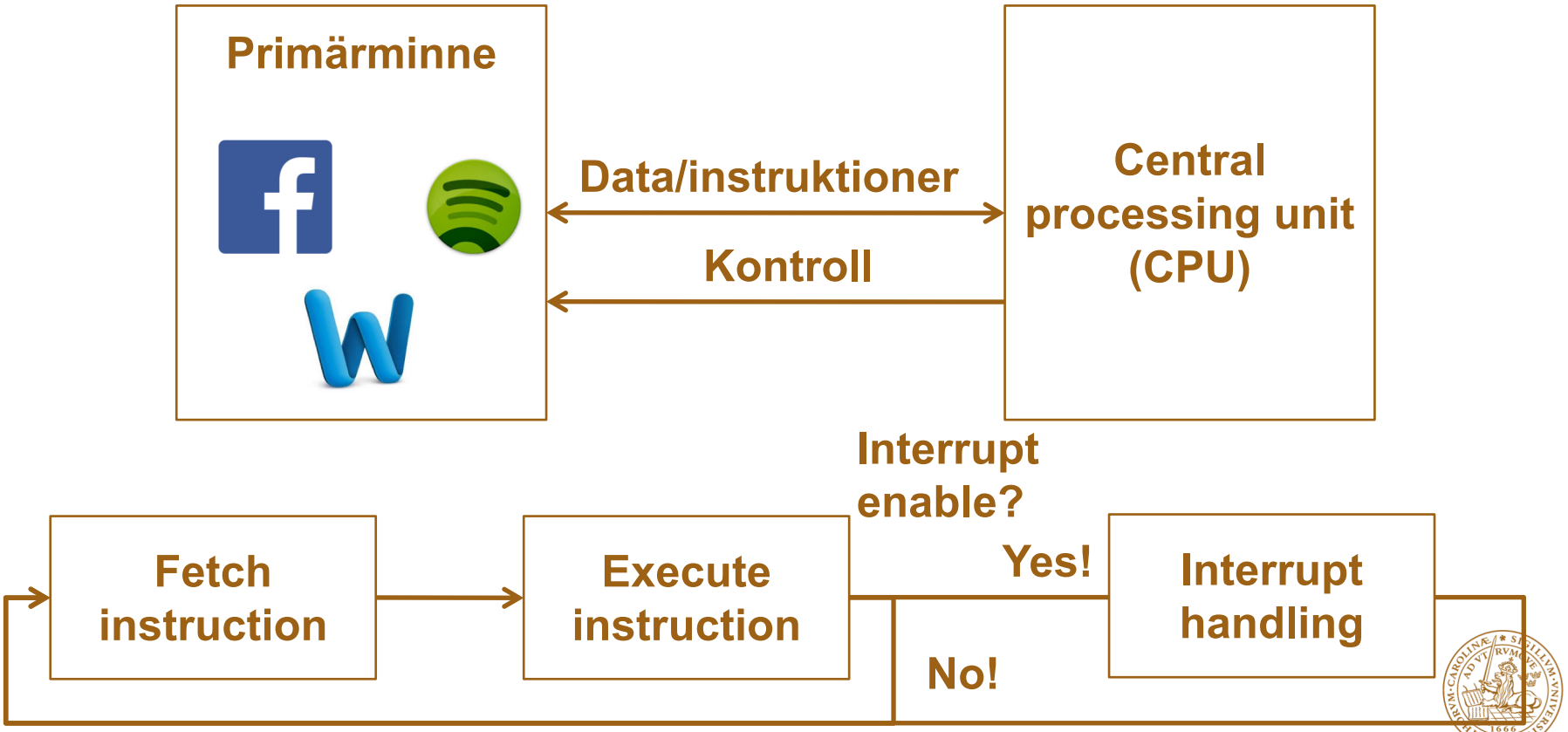
LUNDS UNIVERSITET

Vad hinner man på 0.1 sekund?

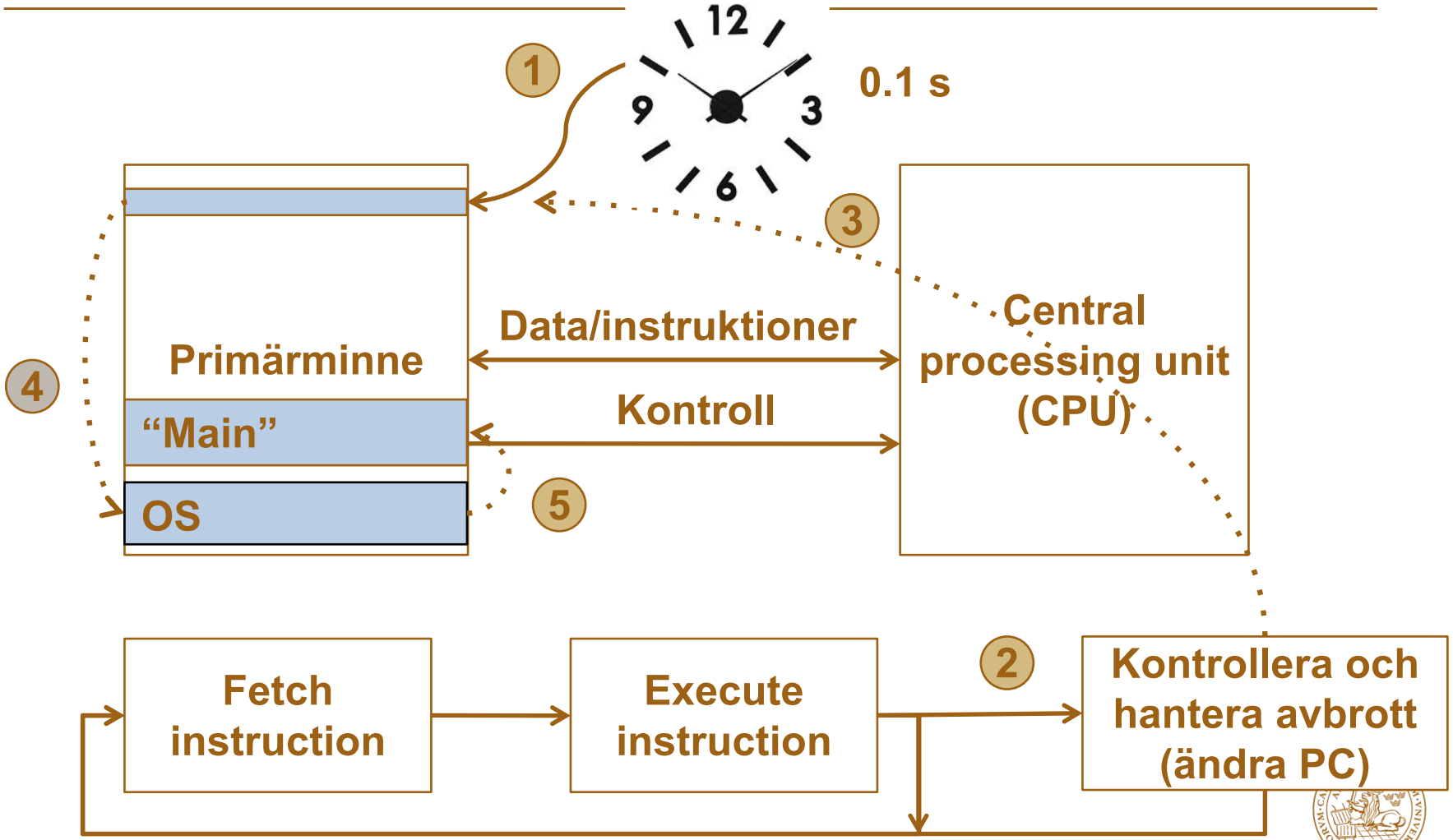
- Antag processor med 1 GHz klockfrekvens
  - 1 GHz = 1 000 000 000 Hz (svängningar per sekund)
- På 1 sekund hinner man 1 000 000 000 klockcykler
- På 0.1 sekund hinner man 100 000 000 klockcykler
  - Om varje instruktion tar 10 klockcykler, hinner man:

    10 000 000 instruktioner

Hur går det till att byta program?

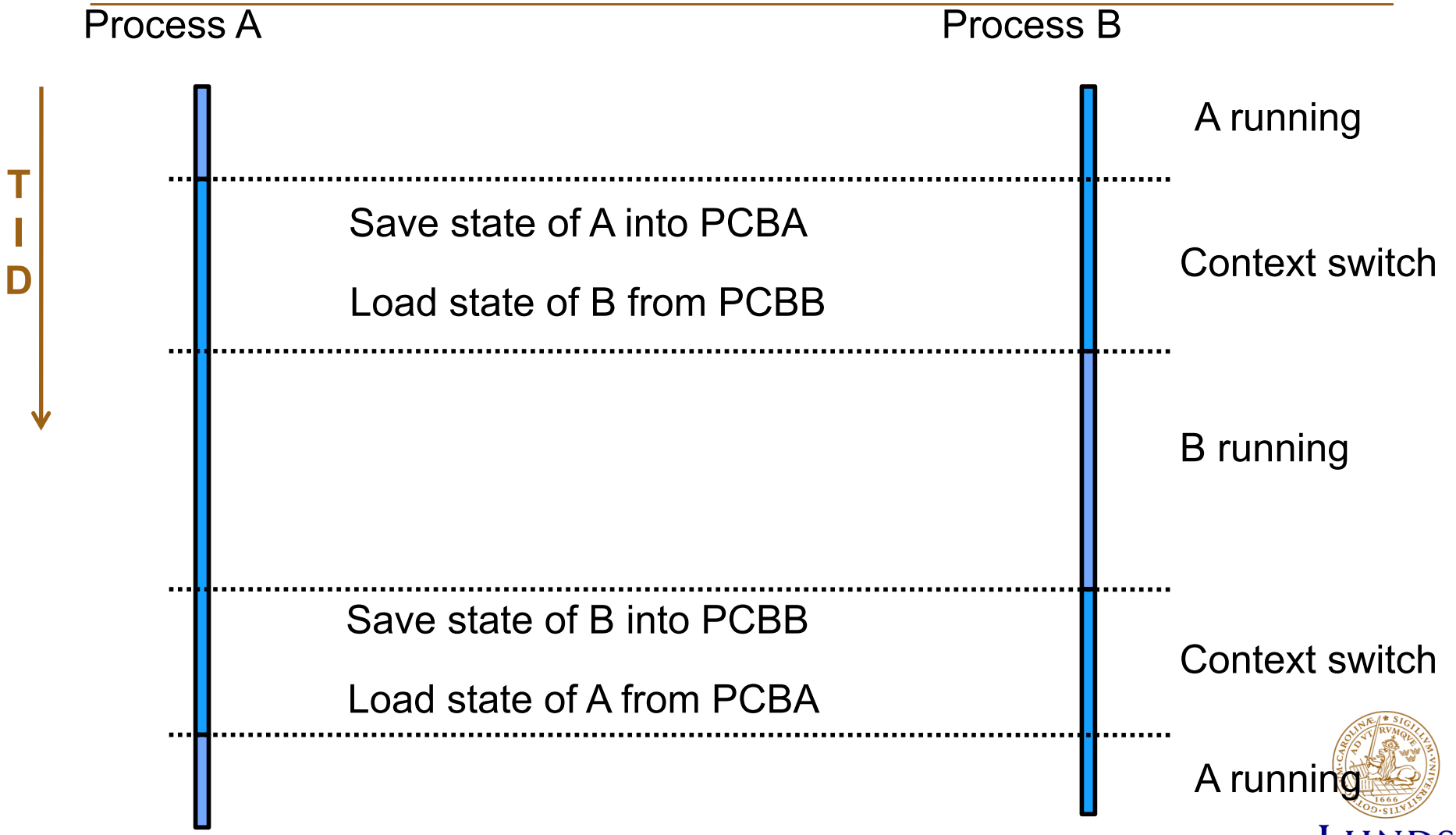


Hur går det till att byta program?



LUNDS UNIVERSITET

Kontextbyte (context switch)

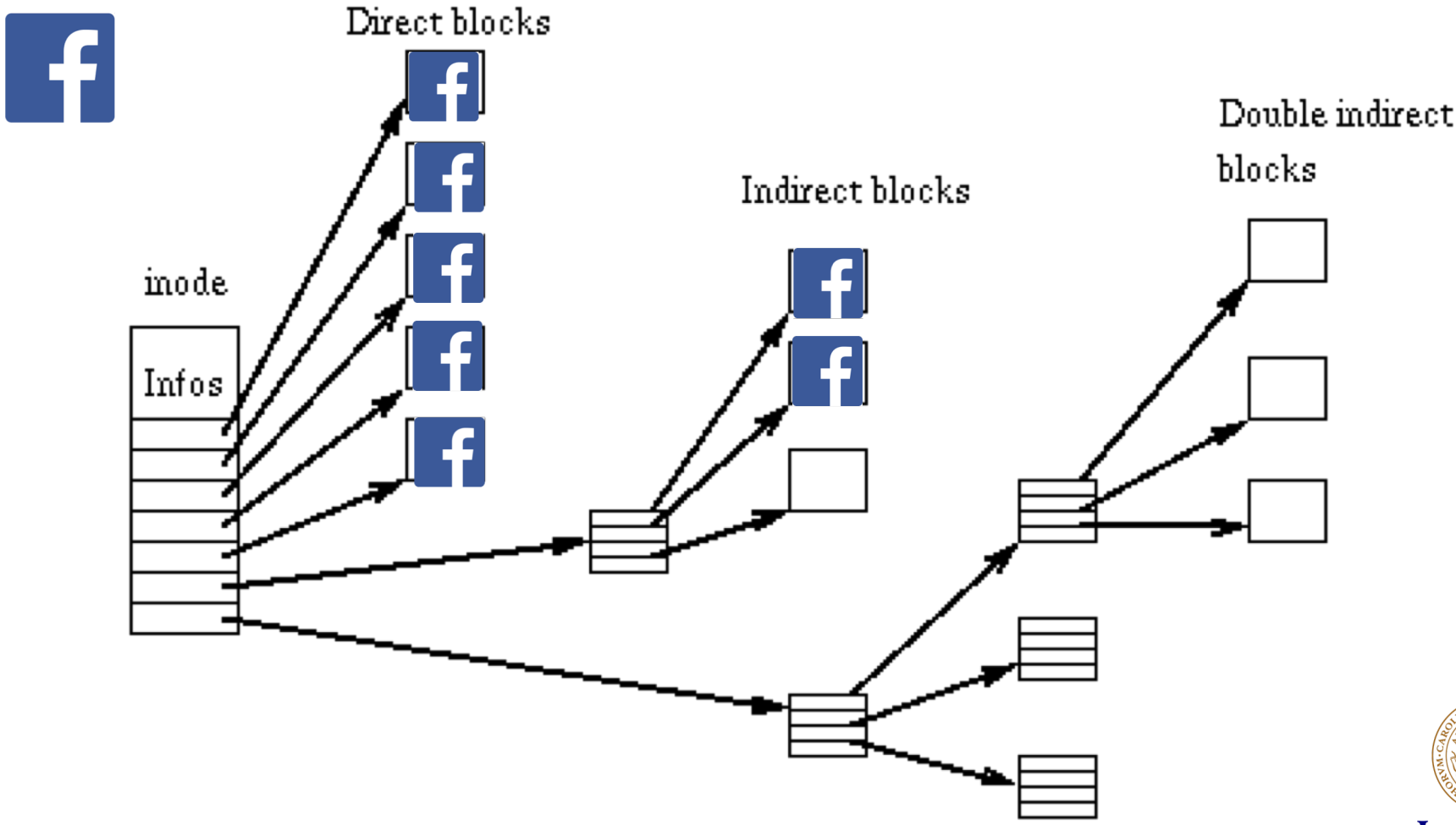


Processkontrollblock

- 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

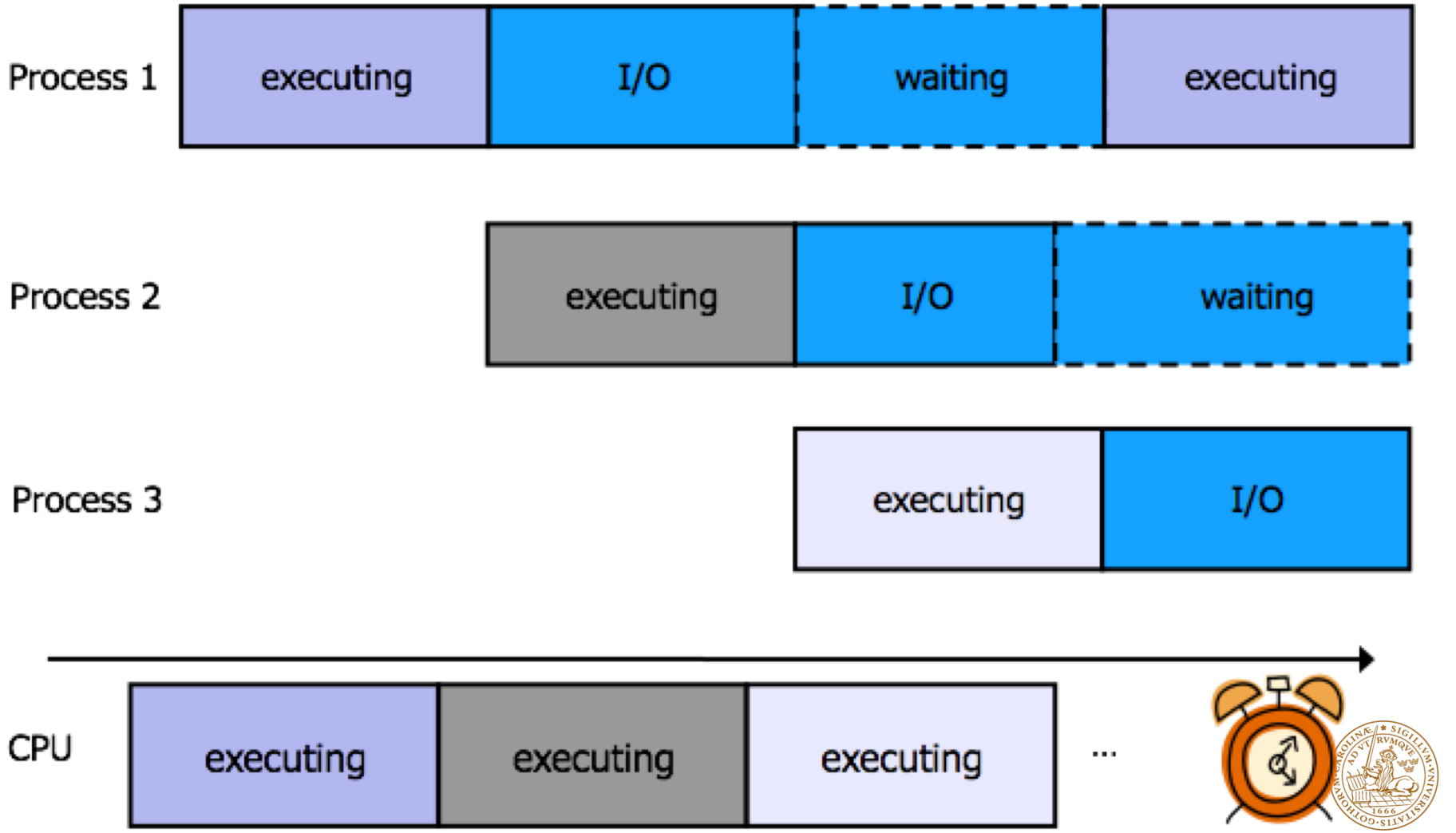
LUNDS UNIVERSITET

Processkontrollblock - Inode

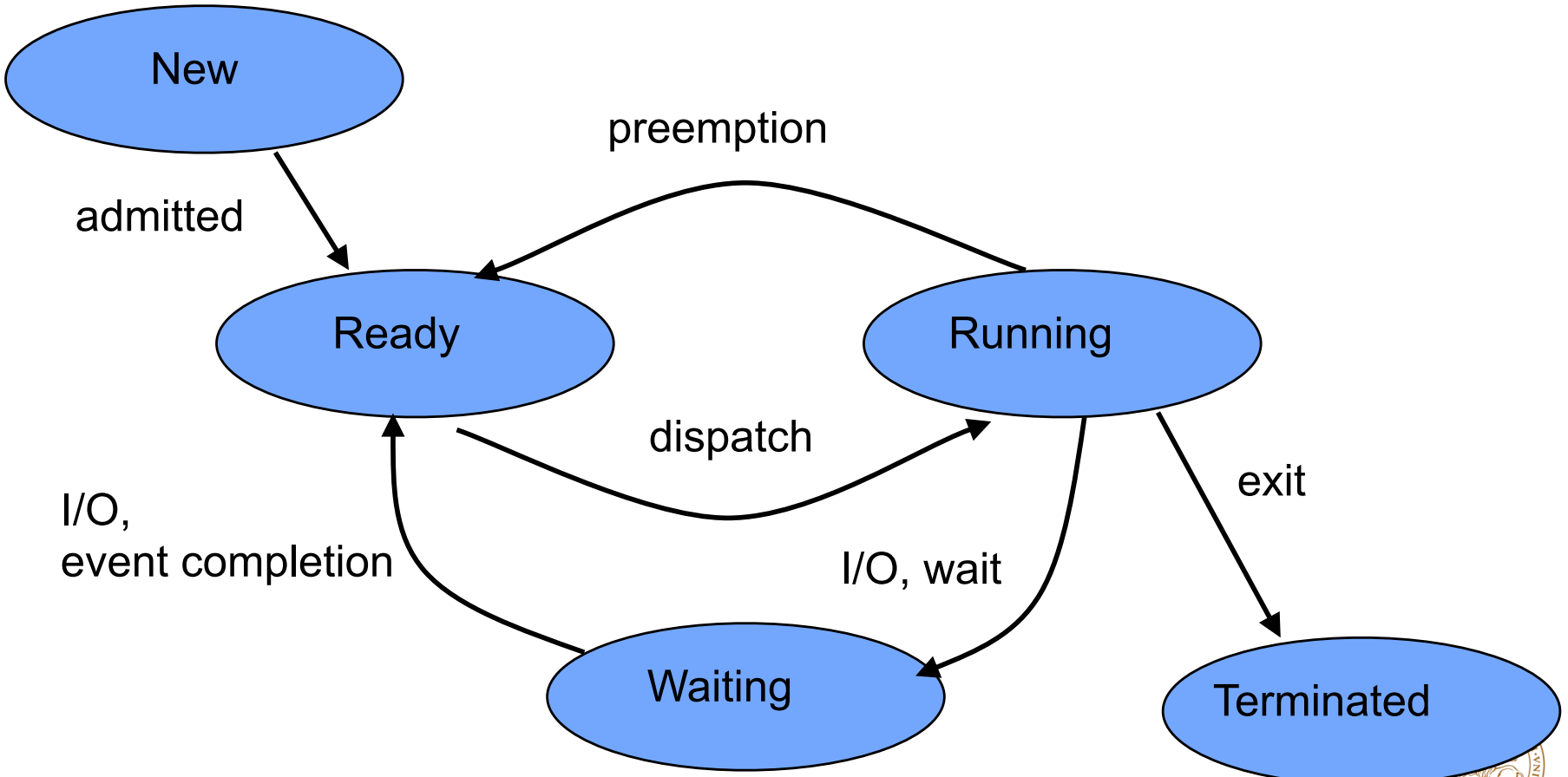


LUNDS UNIVERSITET

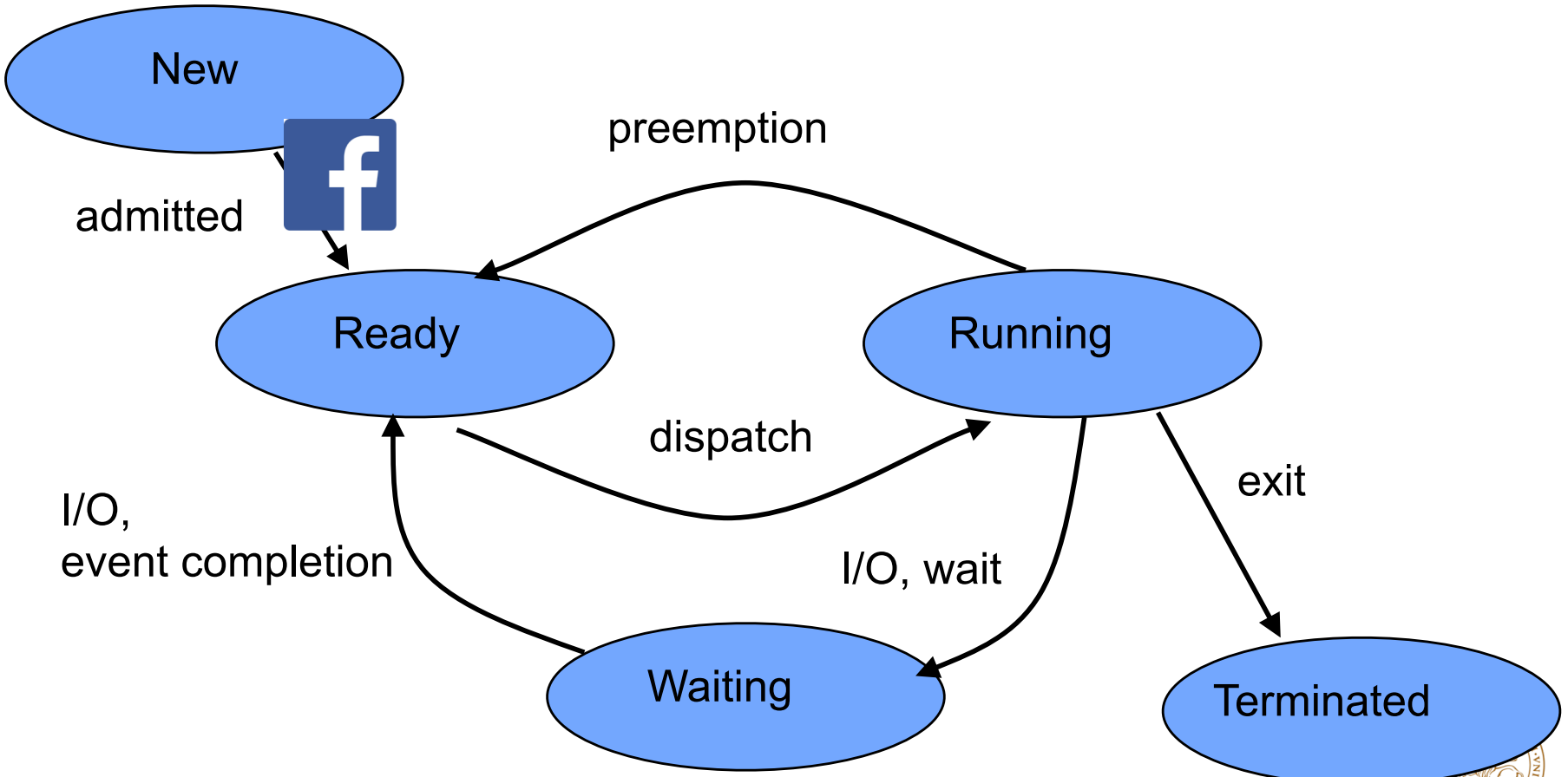
Processhantering



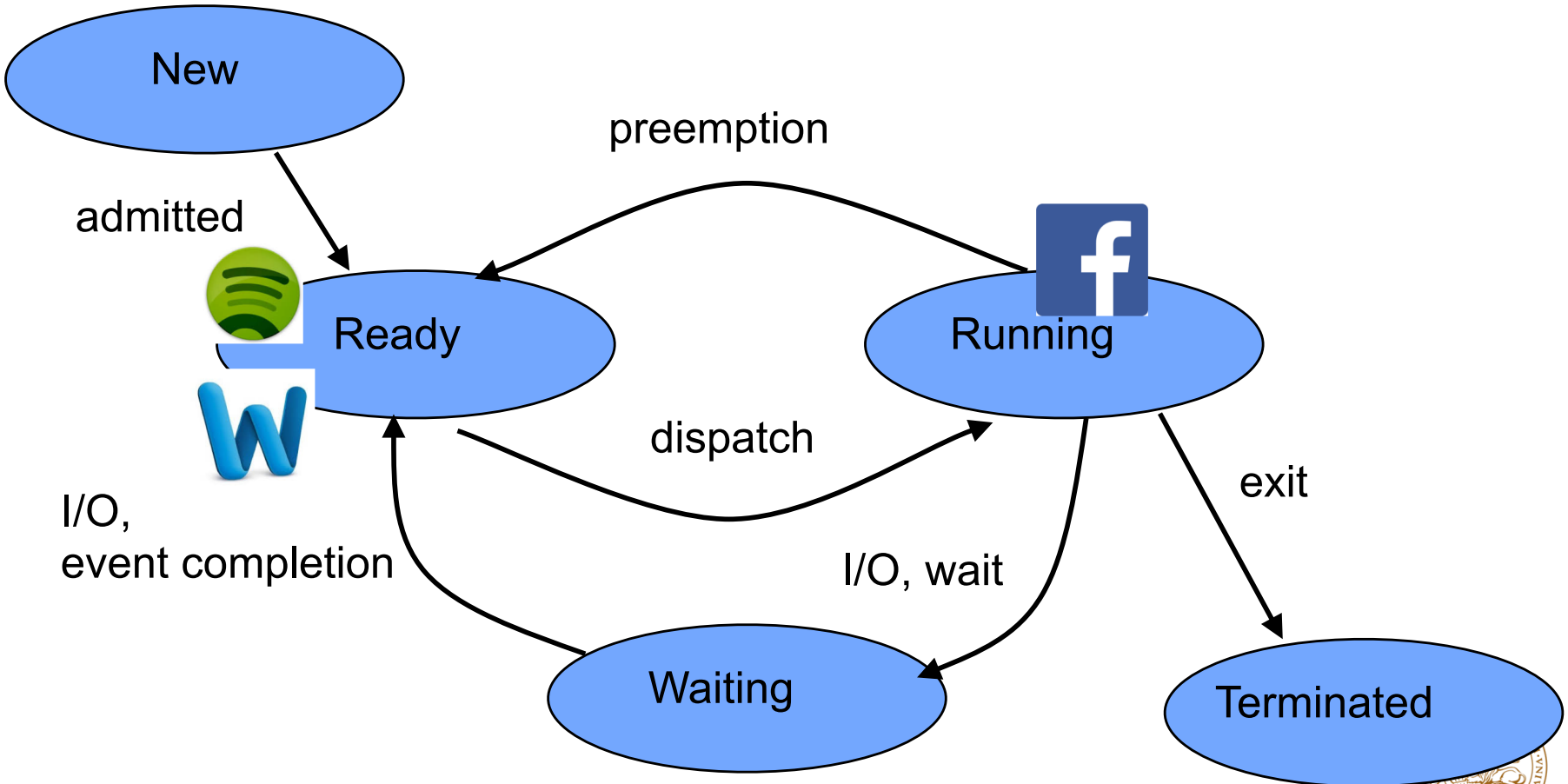
Processmodell



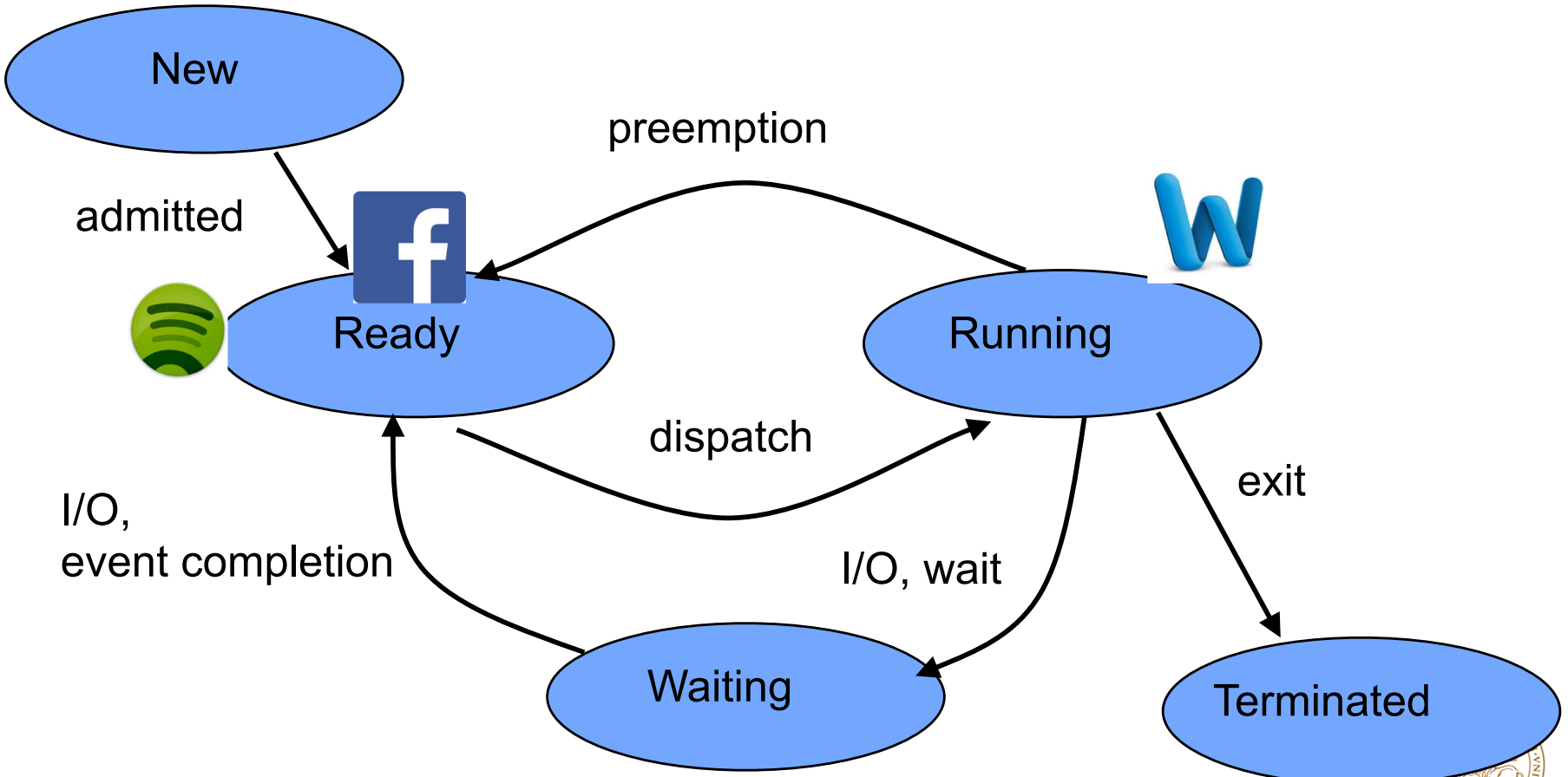
Processmodell



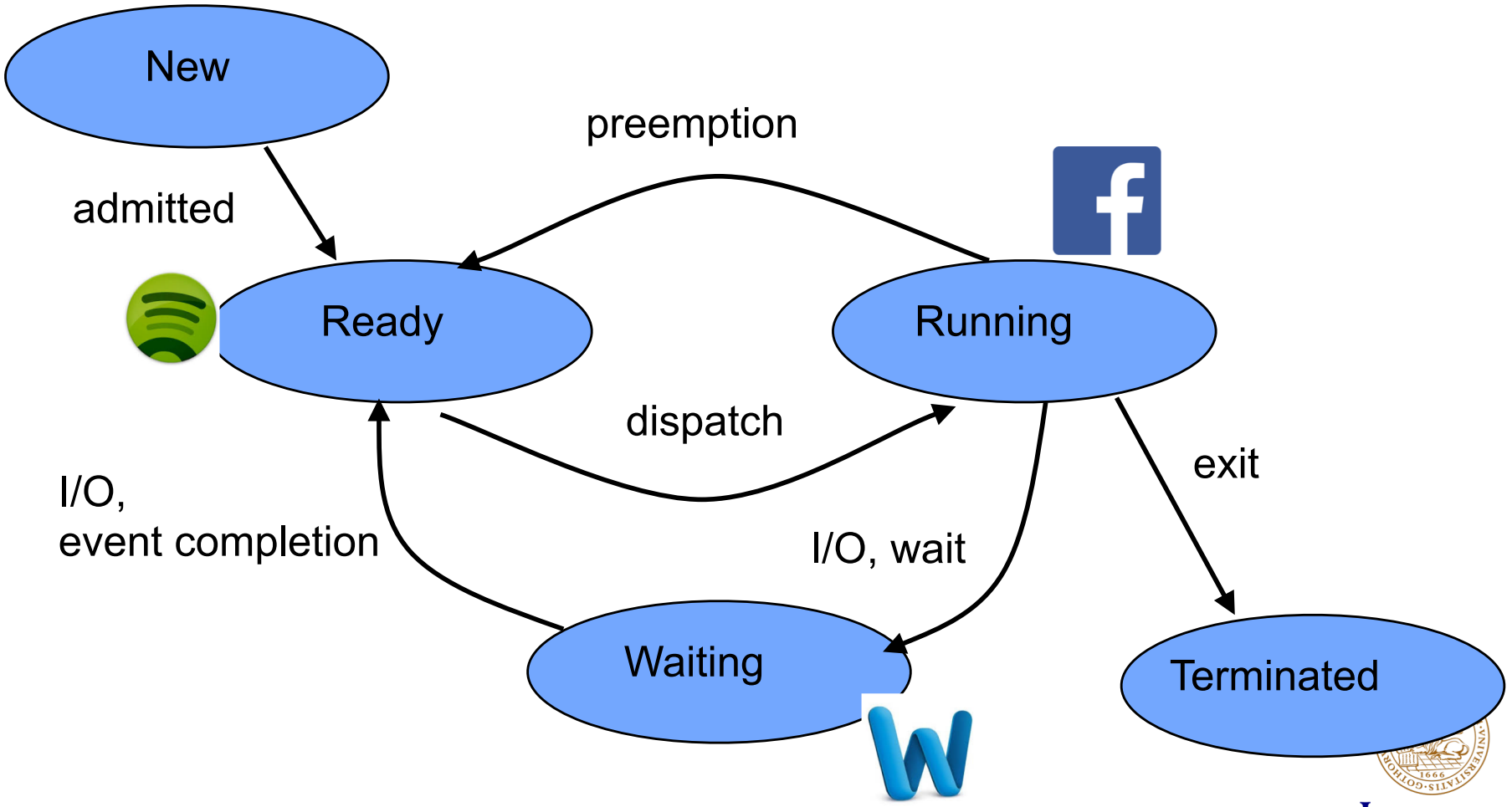
Processmodell



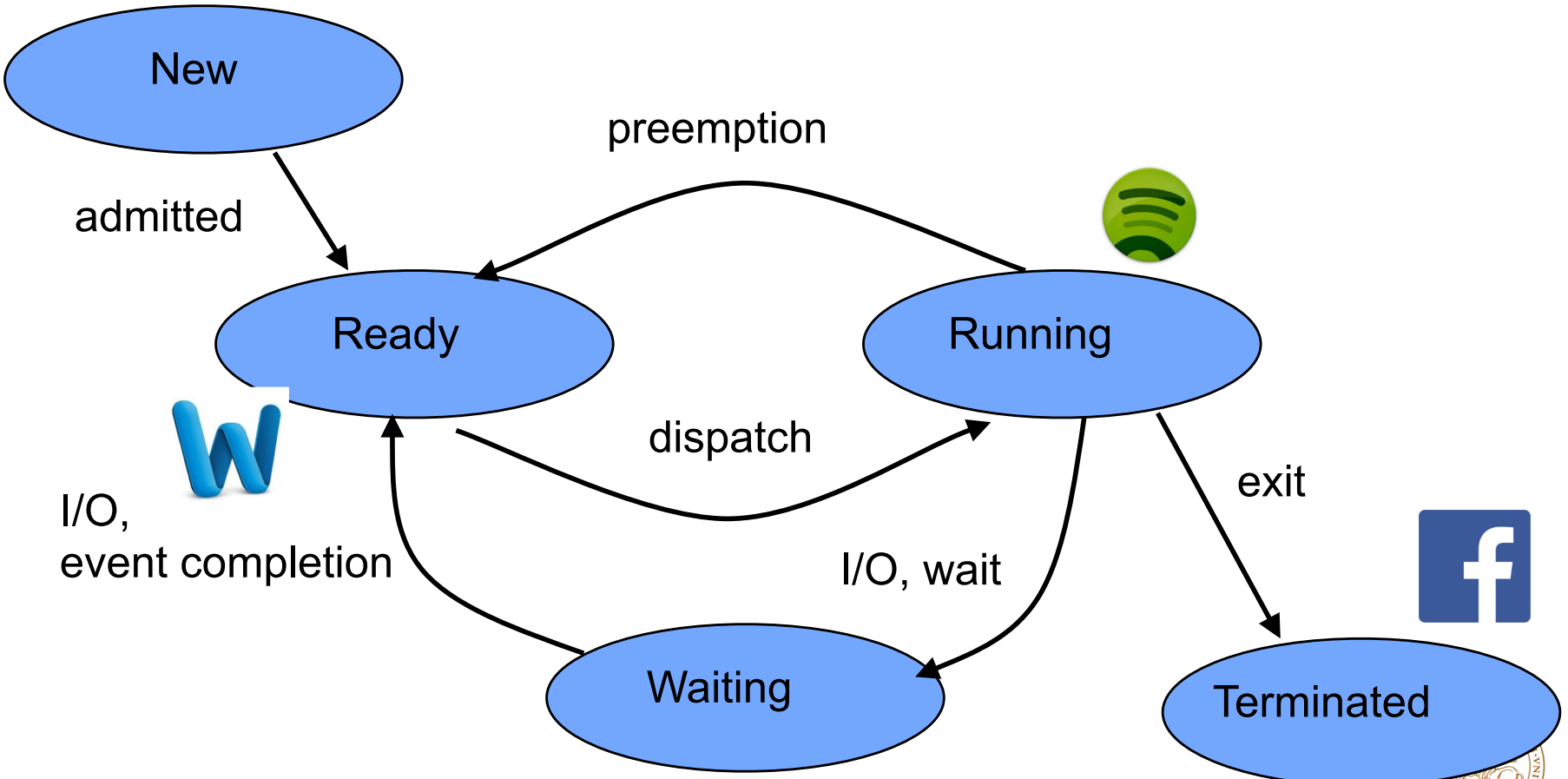
Processmodell



Processmodell



Processmodell



LUNDS UNIVERSITET

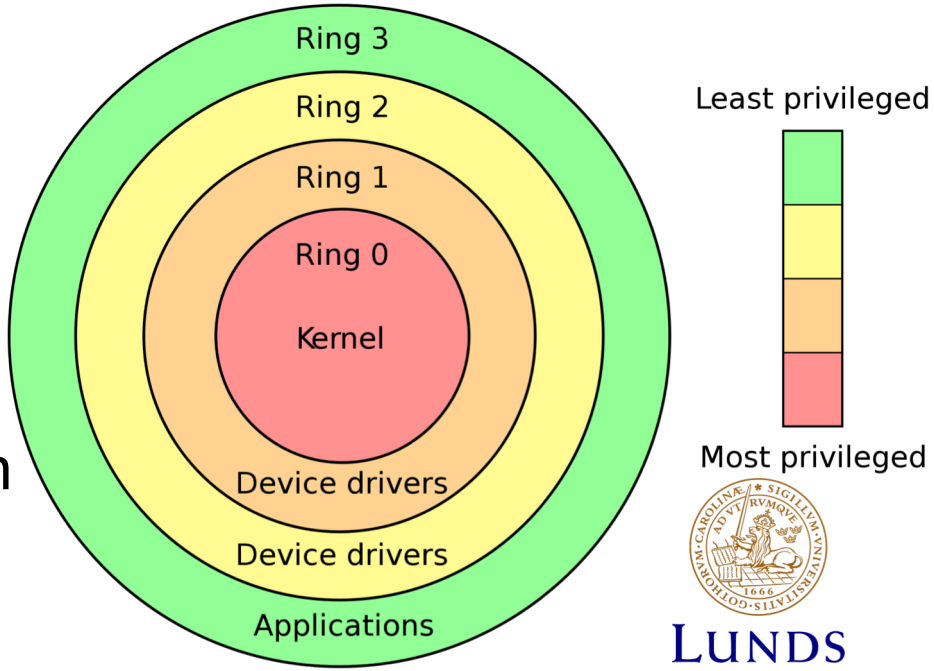
Schemaläggare

- 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

LUNDS UNIVERSITET

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

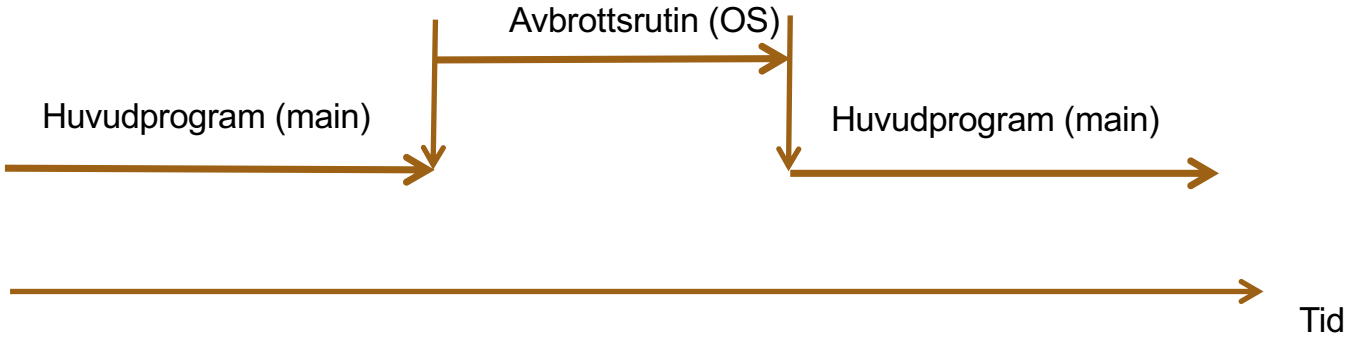


LUNDS UNIVERSITET

Systemanrop

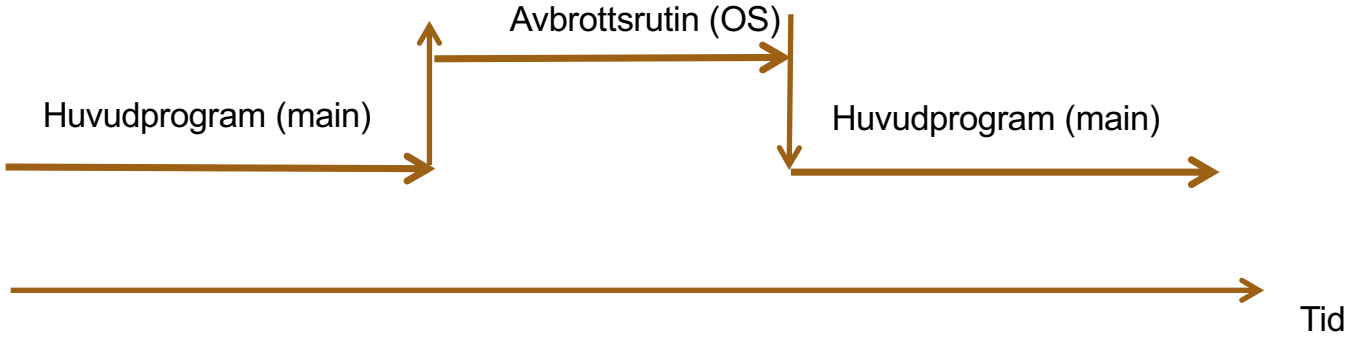
- Exempel: Byte av program (process)

Avbrott genererat av klocka (time slice)



- Exempel C instruktionen: printf

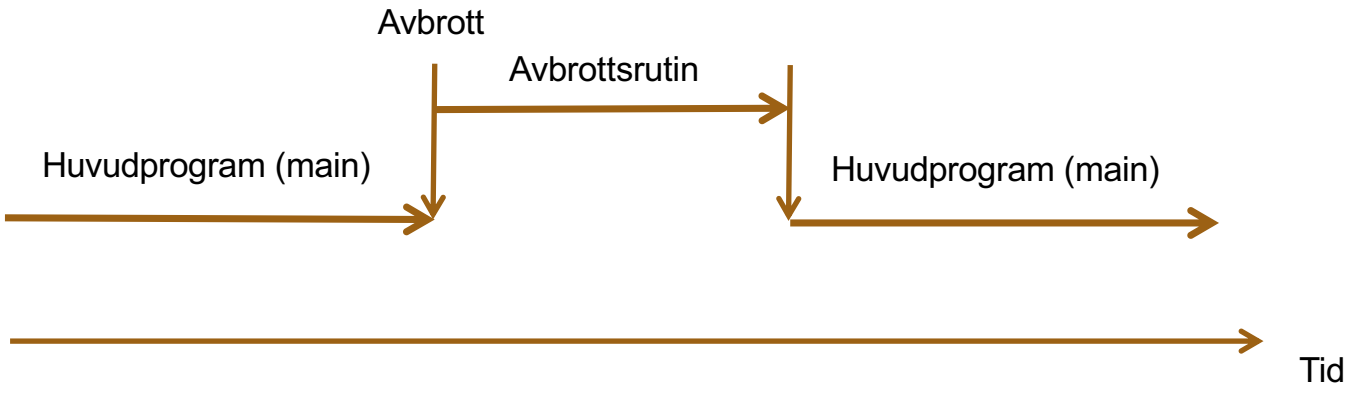
Mjukvaruavbrott genererat av printf (systemanrop)



LUNDS UNIVERSITET

Polling/avbrott

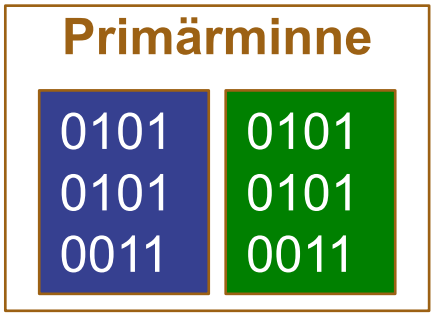
- Polling – kontinuerlig avläsning av ingång, t ex tangent
  - Processorn slösar bort kraft på att kolla ingång
- Avbrott – ingång genererar avbrott
  - Processorn gör annat fram tills avbrott inträffar



LUNDS UNIVERSITET

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

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

LUNDS UNIVERSITET

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