Datorer och datoranvändning
Föreläsningar 2016/17

Per Holm
Roger Henriksson

roger.henriksson@cs.lth.se

Föreläsning 1 — Unix

Förberedelse inför laboration 1.

- Operativsystem, Unix historik
- Filer och kataloger
- Kommandon
- Filskydd
- Kommandotolk
- Processer

Operativsystem

- Dator — kör program
- Operativsystem — en samling program som gör det möjligt att köra ”vanliga” program
- Operativsystemet hanterar:
  - de program som körs (program körs ofta parallellt, operativsystemet ser till att programmen får minne och tid att exekvera)
  - yttre enheter (tangentbord, mus, skärm, nätverk, . . .)
  - lagring av data (filer, . . .)
  - skydd, felhantering kommunikation med användaren
- Linux, Windows, Mac OS X, Unix, DOS, . . .

Unix

Historik:

- 1960-talet: Multics — Multiplexed Information and Computing System — stort operativsystem för stora datorer. Skulle lösa alla problem.
- 1969: Ken Thompson (AT&T Bell Labs) började skriva Unix, litet operativsystem för små datorer, avsett för programmerare. Idéer från Multics.
- 1973: Unix skrevs om i C av Dennis Ritchie, började distribueras.
- Två huvudvarianter: System V (AT&T), BSD (Berkeley University).
- Senare: mycket utveckling, många varianter (Linux, Mac OS X, NetBSD, A/IX, HP/UX, Irix, . . .).

Kännetecken för Unix

Grundläggande:

- Kärna — liten, grundläggande funktioner
- Processer, tidsdelning
- Filskydd
- Fleranvändarsystem
- Kommandotolk

Unix är:

- Litet, standardiserat, flyttbart
- Inte för nybörjare, fast mycket enklare att lära sig nuförtiden när det finns grafiska användargränssnitt och skrivbordsmiljöer (MATE, Gnome, KDE, Xfce . . .)

Filer i Unix

I en dator måste man kunna lagra program (Word, Excel, Javakompilator, spelprogram, ...) och data (brev, hemsidor, programmeringsuppgifter, ...).

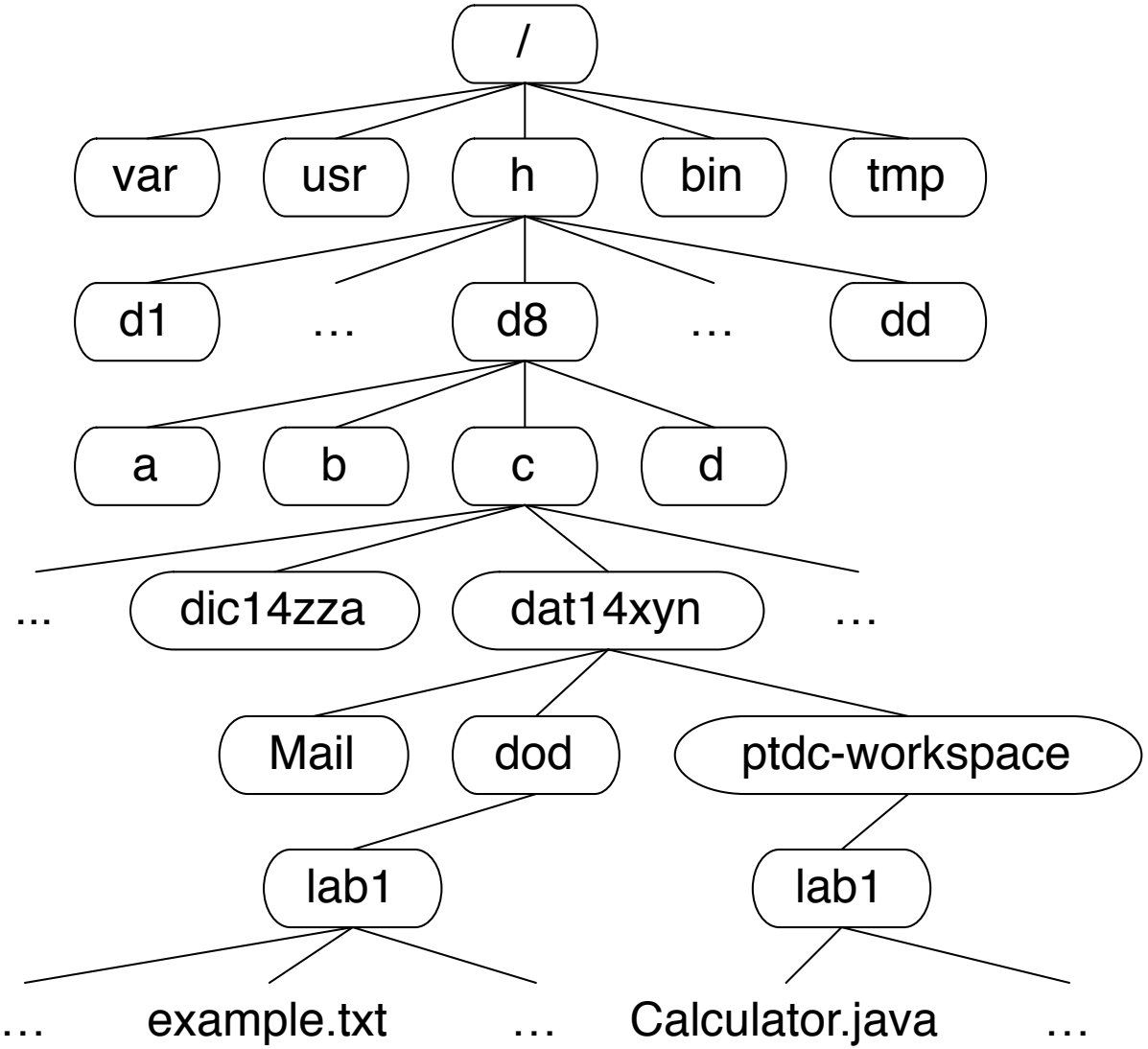
Man lagrar program och data i *filer*. En fil finns normalt på skivminne och har ett namn.

Filnamn i Unix:

- `namn.tillägg` (`Calculator.java`, `Calculator.class`, `brev.txt`, `test1`, ...)
- tillägget är bara en extra upplysning, bestämmer inte filtypen

Kataloger

Varje fil finns i en katalog. Kataloger kan innehålla underkataloger, så man får en hierarkisk struktur, ett *filträd*:



Filnamn

I praktiken är det inte så rörigt som det ser ut på den förra bilden! Ett fullständigt (absolut) filnamn börjar från rotkatalogen och man räknar upp alla katalogerna med / emellan:

```
/h/d8/c/dat14xyn/dod/lab1/example.txt
```

Men Unix håller reda på en *aktuell katalog* som man själv kan ändra. Om `lab1` är aktuell katalog så kan man komma åt ovanstående fil med följande namn (*relativt* filnamn):

```
example.txt
```

Om `dat14xyn` är aktuell katalog så kommer man åt filen med namnet:

```
dod/lab1/example.txt
```

Förkortade filnamn

En del filer har förkortade namn:

`~`	(tilde) hemkatalogen för aktuell användare
`~user`	hemkatalogen för användaren med användarnamnet user
`.`	(punkt) aktuell katalog
`..`	(punktpunkt) katalogen ovanför aktuell katalog i filträdet

Antag att `dod/lab1` är aktuell katalog. Man kan navigera upp och ned i filträdet enligt följande exempel:

```
~/ptdc-workspace/lab1/Calculator.java
(till hemkatalogen, ner i ptdc-workspace, ner i lab1)

../../ptdc-workspace/lab1/Calculator.java
(upp ett steg, upp ett steg, ner i ptdc-workspace, ner i lab1)
```

Kommandotolk

Man kommunicerar med Unix genom att ge kommandon. Ett särskilt program i operativsystemet, kommandotolken, ”shell”, läser in kommandona och utför det som ska göras.

Det finns flera olika kommandotolkar. På Linuxdatorerna i E-huset används kommandotolken `bash` (” Bourne-again shell”). Andra vanliga kommandotolkar är `sh`, `zsh` och `csh`.

Kommandotolken läser kommandon som ges i ett terminalfönster på skärmen. Så här arbetar tolken:

```
Upprepa i all oändlighet:
    Läs ett kommando
    Om kommandot är ett riktigt kommando:
        utför det som ska göras,
    annars:
        skriv ut "XXX: command not found"
```

Kommandoformat

Varje kommando skrivs på en rad:

```
kommandonamn -option1 -option2 ... argument1 ...
```

- Kommandot talar om vad som ska göras.
- Argumenten är (oftast) filer eller kataloger som påverkas av kommandot.
- Optionerna modifierar kommandot på något sätt.

Exempel:

`javac Calculator.java`	Kompilerar `Calculator.java` med Java-kompilatorn, skapar `Calculator.class`
`cp -i report.tex old.tex`	Kopierar `report.tex` till `old.tex`. `-i` betyder att systemet frågar om `old.tex` ska skrivas över om den redan finns

Kommandon för filhantering

Exempel på kommandon för att hantera filer:

`cp orig kopia`	Kopiera filen `orig` till `kopia`.
`less fil`	Skriv ut `fil` på skärmen, en sida i taget.
`ls [-la] kat`	Skriv ut en innehållsförteckning över katalogen `kat` (aktuell katalog om ingen katalog ges). `-l` skriv i ett längre format, `-a` skriv också ut punktfiler.
`mv fil1 fil2`	Döp om `fil1` till `fil2`. Om `fil2` är en katalog så flyttas filen till den katalogen.
`rm fil1 ...`	Tag bort de angivna filerna.

Kataloghantering

Exempel på kommandon för att hantera kataloger:

`cd kat`	Ändra aktuell katalog till `kat`. Utan argument blir det hemkatalogen (samma som `cd ~`).
`mkdir kat`	Skapa underkatalogen `kat` i den aktuella katalogen.
`pwd`	Skriv ut namnet på aktuell katalog.
`rmdir kat`	Tag bort `kat`. Man måste först ta bort alla filerna i katalogen.

Filskydd 1

Varje fil har en ägare. Ägaren identifieras med användarnamn och grupp, till exempel `dat14xyn` i gruppen `students`.

Ägaren kan skydda filer så att andra inte kan komma åt dem eller öppna filer så att andra kan komma åt dem.

Filskydd och ägare (och storlek och datum) skrivs ut om man gör `ls -l`:

```
hacke-3{dat14xyn}: ls -l
-rw-r-----  1 dat14xyn  students  4940 aug 21 11:14 example.txt
   skydd       ägare     grupp
```

Filskydd 2

```
-rw-r-----  1 dat14xyn  students  4940 aug 21 11:14 example.txt
  u   g   o
```

Tre kategorier av användare:

- `u` (user) filens ägare
- `g` (group) medlemmar i samma grupp som ägaren
- `o` (others) alla andra

Tre olika rättigheter:

- `r` (read) tillstånd att läsa
- `w` (write) tillstånd att skriva
- `x` (execute) tillstånd att exekvera program (för kataloger betyder `x` tillstånd att titta på innehållet i katalogen)

Filskydd 3

Kommando för att ändra filskydd:

```
chmod skydd fil1 fil2 ...
```

Filskydd kan anges symboliskt, till exempel `u=rw,o=r`, men också numeriskt, där man anger skydden som är satta med ettor, de som inte är satta med nollor. Till exempel:

```
x = 001, w = 010, r = 100, rx = 101, rw = 110, rwx = 111
```

Siffrorna tolkar man sedan som binära tal, vilket ger:

```
x = 1, w = 2, r = 4, rx = 5, rw = 6, rwx = 7
```

Fullständiga numeriska filskydd blir till exempel:

```
604 <=> u=rw,o=r     705 <=> u=rwx,o=rx
```

bash-finesser 1

Man kan editera den aktuella kommandoraden:

Delete	Tag bort tecknet till vänster om markören.
Control-U	Radera hela raden.
$\leftarrow$ $\rightarrow$	Flytta sig på raden.
Tab	Filnamnskomplettering (man behöver bara skriva början av ett långt namn).

`bash` håller reda på de senaste kommandona som utförts, och man kan få tillbaka gamla kommandon:

$\uparrow$	Återkalla senaste kommando (kan upprepas).
`!!`	Gör om senaste kommando.
`!abc`	Gör om senaste kommando som inleds med `abc`, till exempel `!javac`

bash-finesser 2

När man refererar till filer kan man utnyttja jokertecken (wildcards):

- `?` motsvarar *ett* godtyckligt tecken
- `*` motsvarar 0–flera godtyckliga tecken

Antag att vi har följande filer i vår katalog: `Test1.java`, `Test2.java`, `Test23.java`, `Final.java`, `Test1.class`, `Test2.class`, `Final.class`, `FinalReport.tex`, `Outline.tex`.

```
javac Test?.java (javac Test1.java Test2.java)
rm *.class       (rm Test1.class Test2.class Final.class)
gedit F*.tex     (gedit FinalReport.tex)
```

Initieringsfiler

Många program läser en initieringsfil när de startas. Initieringsfilen innehåller inställningar för programmet. Namnen på initieringsfilerna inleds med punkt, så man ser normalt inte dessa filer när man gör `ls`.

`bash` läser initieringsfilen `.bash_profile` när ett nytt terminalfönster skapas. Bra rader att ha i denna fil (gör att man alltid får frågor om man verkligen vill ta bort/skriva över filer):

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

Varning: kopiera inte andras initieringsfiler om du inte begriper vad som står i dem!

Vad är egentligen ett kommando?

Kommandon är av två slag:

"inbyggda" små kommandon som exekveras direkt i kommandotolken, till exempel `cd`, `pwd`, `exit`

"vanliga program" kommandotolken letar efter ett program med samma namn som kommandot

Kommandotolken letar efter program i de kataloger som anges i sökvägen (operativsystemvariabeln `PATH`, som man själv kan sätta med kommandot `export`). När man till exempel skriver `javac X.java` exekveras programmet `/usr/bin/javac`, eftersom katalogen `/usr/bin` finns i sökvägen.

När man ska exekvera ett eget program som finns i den aktuella katalogen skriver man:

```
./programnamn
```

`./` är nödvändigt eftersom `.` (den aktuella katalogen) inte finns i sökvägen.

Processer

Varje program som körs i Unix körs som en egen process som exekverar självständigt, ”parallellt” med andra processer. Varje process körs några millisekunder, sedan får nästa process exekvera, osv. Detta hanteras av operativsystemet.

När man exekverar ett program från ett terminalfönster ”låser” programmet fönstret tills det avslutas — så vill man ofta att det ska fungera.

Men man kan också köra program ”i bakgrunden” dvs frikopplade från terminalfönstret. Det gör man genom att skriva & sist på kommandoraden, till exempel:

```
gedit example.txt &
```

Fönstersystem, skrivbordsmiljö

I Unix är fönstersystemet inte inbyggt (Unix utvecklades långt innan det fanns grafiska skärmar).

De flesta Unixsystem använder fönstersystemet X Window System, som oftast kallas bara X. X gör det möjligt att ha fönster på skärmen, att flytta dem, ikonifiera dem, osv.

Exakt hur fönstren ska se ut och hur man arbetar med dem bestäms av *fönsterhanteraren*, som också är ett separat program. Ovanpå fönsterhanteraren finns skrivbordsmiljön. På Linuxdatorerna i E-huset används skrivbordsmiljön Unity.

Det bästa (enda) sättet att lära sig att använda en skrivbordsmiljö är att träna!

Föreläsning 2 — Lågnivåprogrammering

Förberedelse inför laboration 2.

- Maskinspråk, assemblerspråk
- Talrepresentation
- En enkel dator, komponenter
- Instruktionsformat, instruktionscykel
- ME-datorn, ME-assembler
- Javasatser i ME-assembler
- Vektorer (överkurs)

Lågnivåprogrammering

Normalt programmerar man i högnivåspråk, till exempel Java eller C++.
I undantagsfall kan man behöva skriva program i datorns maskinspråk, till exempel i de allra innersta delarna i ett operativsystem.

Då utnyttjar man datorns *assemblerspråk*. En instruktion i assemblerspråket motsvarar direkt en maskininstruktion men skrivs i symbolisk form, till exempel `ADD` i stället för operationskoden `001100` (eller vad nu additionsoperationen har för kod).

Men även om man inte programmerar på maskinspråksnivå så bör man förstå vad som händer på den nivån! Det underlättar förståelsen av programmering i högnivåspråk.

Binära tal

Inuti datorn lagras talvärden i form av binära tal, ”nollor och ettor”. Vilket tal som helst kan skrivas i binär form, alltså med basen 2. Exempel:

$$1001 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9$$

$$111000 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 56$$

De binära siffrorna kallas *bitar* (BInary digiTs). Negativa tal kan man representera genom att man reserverar en av bitarna i en minnescell för att ange om talet är positivt eller negativt. (I praktiken är det något mera komplicerat.)

Det blir enkelt för datorn att räkna:

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

int och double

Ett int-tal i Java omfattar 32 bitar. En av bitarna är teckenbit, så man har alltså 31 bitar till talvärdet. Det största talet man kan representera med 31 bitar är:

$$1 \cdot 2^{30} + 1 \cdot 2^{29} + \ldots + 1 \cdot 2^{1} + 1 \cdot 2^{0} = 2^{31} - 1 = 2,147,483,647$$

double-tal är mera komplicerade. Man delar upp talet i två delar, taldel och exponent:

$$\text{tal} = \text{taldel} \cdot 2^{\text{exponent}}$$

och delar de 64 tillgängliga bitarna mellan taldelen och exponenten på ett finurligt sätt som vi återkommer till senare (i föreläsningen om Matlab).

Hexadecimala tal

Stora tal blir väldigt långa när man skriver dem i binär form. Exempel:

$$1101101101111 = 7023$$

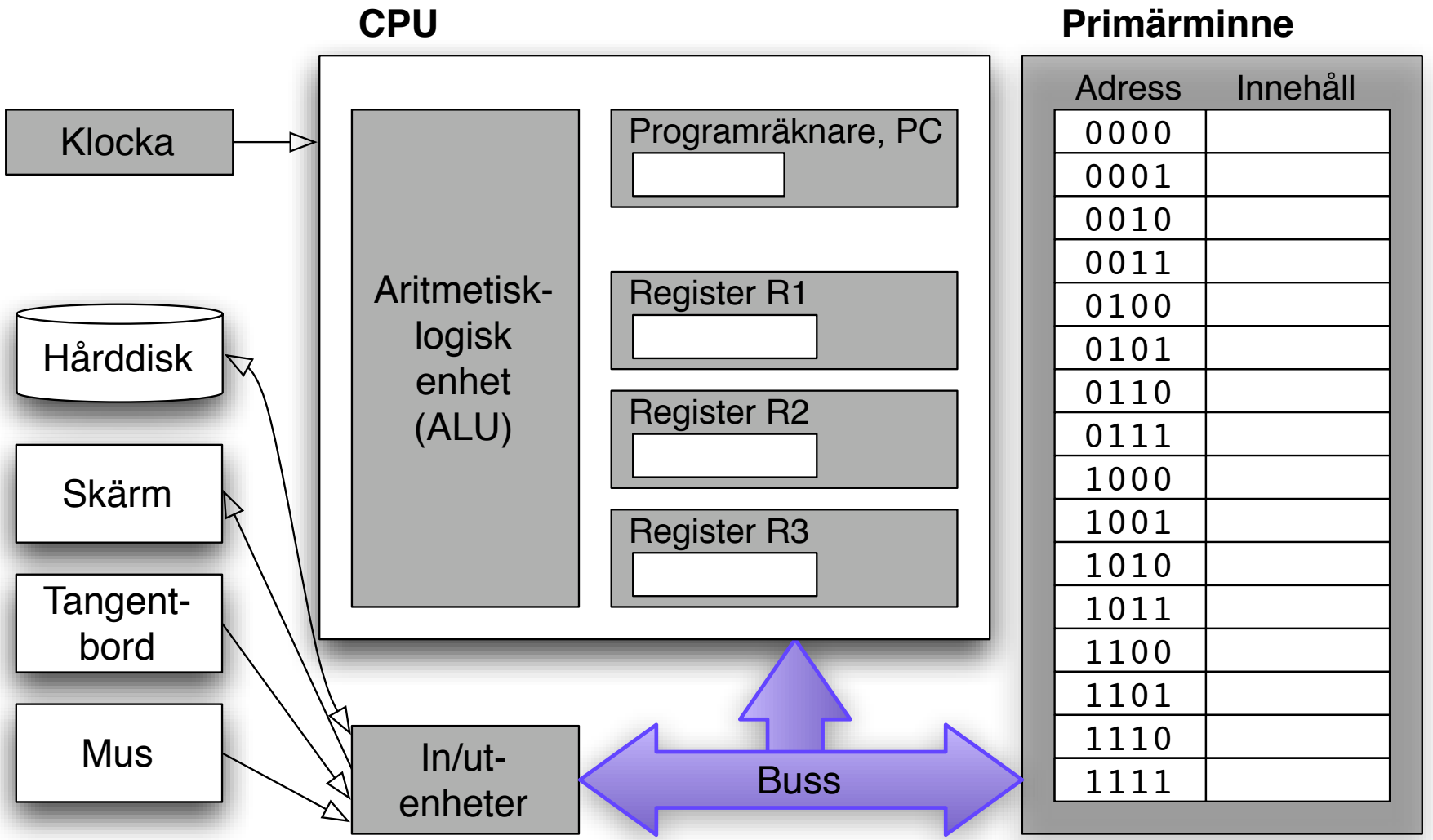
För att få kortare tal men ändå få en direkt koppling till den binära representationen av talet brukar man använda *hexadecimala* tal dvs tal skrivna i basen 16. Då behöver man 16 olika siffror: man använder 0–9 som vanligt och `A`–`F` för 10–15. Exempel:

$$\texttt{1B6F} = 1 \cdot 16^3 + 11 \cdot 16^2 + 6 \cdot 16^1 + 15 \cdot 16^0 = 7023$$

Man kan direkt översätta binära tal till hexadecimala genom att dela upp bitarna i grupper om 4 från höger:

```
1 1011 0110 1111
1    B    6    F
```

En enkel modell av datorns inre



Datorbeskrivning

Datorn på föregående bild har alltså:

- 16 minnesceller om 8 bitar, adresserna 0–15 (0000–1111). I en minnescell lagras en instruktion eller ett positivt heltal (vi har alltså ingen teckenbit).
- 3 register om 8 bitar: `R1`, `R2`, `R3`.
- En programräknare (Program Counter, PC) som håller reda på adressen till den instruktion som står i tur att exekveras.
- En räkne- och styrenhet som ser till att instruktionerna hämtas och utförs.

Instruktioner

`ADD`	`00xxyyzz`	Addera innehållet i registret med nummer `xx` och registret med nummer `yy`, lagra resultatet i registret med nummer `zz`.
`SUB`	`01xxyyzz`	Subtrahera `xx` - `yy`, lagra resultatet i `zz`.
`LADDA`	`10xxyyyy`	Ladda register `xx` med innehållet i minnescellen med adressen `yyyy`.
`LAGRA`	`11xxyyyy`	Lagra register `xx` i minnescell `yyyy`.
`VHOPP`	`0000xxxx`	Hoppa till (dvs hämta nästa instruktion från) minnescell `xxxx` om `R1` $\neq$ `R3`, annars fortsätt med nästa instruktion.
`SKRIV`	`1100xxxx`	Skriv ut innehållet i minnescell `xxxx`.

Utför instruktioner

Instruktionscykel:

1. Hämta instruktionen från minnescellen vars adress finns i `PC`.
2. Utför instruktionen.
3. Uppdatera programräknaren. Normalt:

   ```
   PC = PC + 1
   ```

   men om instruktionen var `VHOPP` och `R1` $\neq$ `R3` så blir `PC = xxxx`.

Ett program ...

Följande program adderar två tal:

```
Adress   Innehåll Läge      Symbolkod
0000     10010100           LADDA R1,DATA1
0001     10100101           LADDA R2,DATA2
0010     00011011           ADD   R1,R2,R3
0011     11110110           LAGRA R3,SVAR
0100     00000001 DATA1
0101     00000010 DATA2
0110     00000000 SVAR
```

ME — en annan dator

ME-datorn har:

- 1000 minnesceller om 32 bitar, positiva och negativa heltal.
- 5 register om 32 bitar: `R1`, `R2`, `R3`, `R4` och `R5`.
- Många fler operationer.
- Fler sätt att adressera (komma åt) värden. Exempel:

```
add r1,100,m(356) ! addera innehållet i register r1
                  ! och talet 100, lagra summan i
                  ! minnescellen med adressen 356
```

- Ett symboliskt assemblerspråk.
- En emulator där man kan köra program från början till slut eller stegvis.

Parametrar till instruktioner

Det finns fyra olika typer av parametrar:

1. Ett konstant värde: det angivna värdet används direkt som parameter. Man kan naturligtvis inte använda denna parametertyp som resultatparameter.
2. Register: ett av registren `R1`, `R2`, `R3`, `R4` eller `R5`.
3. Minne: en av minnescellerna. Skrivs `M(adress)`, till exempel `M(356)`.
4. Minne indirekt: den minnescell vars adress finns i ett av registren. Skrivs `M(register)`, till exempel `M(R1)`. Detta är överkurs!

En femte parametertyp, läge, används bara i samband med hopp::

```
back:   sub  r1,1,r1
        ...
        jump back
```

Instruktioner — aritmetik

move	p1,p2	Kopiera p1 till p2.
add	p1,p2,p3	Addera p1 och p2, lägg resultatet i p3.
sub	p1,p2,p3	Subtrahera p2 från p1, lägg resultatet i p3.
mul	p1,p2,p3	Multiplicera p1 och p2, lägg resultatet i p3.
div	p1,p2,p3	Dividera p1 med p2, lägg resultatet i p3 (heltals-division, resten kastas bort).

Exempel:

```
add  100,m(25),r1
mul  r1,3,r1
move r1,m(30)
```

Instruktioner — hopp och annat

jump	lab	Hoppa till `lab`, som ska vara ett läge.
jpos	p1,lab	Hoppa till lab om `p1` $>= 0$, annars fortsätt exekveringen med nästa sats.
jneg	p1,lab	Hoppa till lab om `p1` < 0, annars fortsätt.
jz	p1,lab	Hoppa till `lab` om `p1` $= 0$, annars fortsätt.
jnz	p1,lab	Hoppa till `lab` om `p1` $\neq 0$, annars fortsätt.
read	p1	Läs in ett talvärde, lagra det i `p1` (detta är egentligen ett anrop till operativsystemet).
print	p1	Skriv ut `p1` (också detta är ett anrop till operativsystemet).
stop		Avsluta exekveringen.

Tilldelningssatser och beräkningar

Satser i Java:

```
x = 10;
y = 20;
z = 2 * (x + 1) - 3 * y;
```

Vi förutsätter att variablerna `x`, `y` och `z` finns i `M(0)`, `M(1)` och `M(2)`, och vi använder register för att lagra mellanresultat. ME-assembler:

```
move 10,m(0)     ! x = 10
move 20,m(1)     ! y = 20
add  m(0),1,r1   ! r1 = x + 1
mul  2,r1,r1     ! r1 = 2 * r1
mul  3,m(1),r2   ! r2 = 3 * y
sub  r1,r2,m(2)  ! z = r1 - r2
```

if-satser

```
x = scan.nextInt();
y = scan.nextInt();
if (x > y) {
    z = x;
} else {
    z = y;
}
System.out.println(z);
```

```
      read  m(0)          ! läs in x
      read  m(1)          ! läs in y
      sub   m(1),m(0),r1  ! r1 = y - x
      jpos  r1,else       ! hoppa om x <= y
      move  m(0),m(2)     ! z = x;
      jump  comm          ! hoppa till comm
else: move  m(1),m(2)     ! z = y
comm: print m(2)          ! skriv ut z
```

while-satser

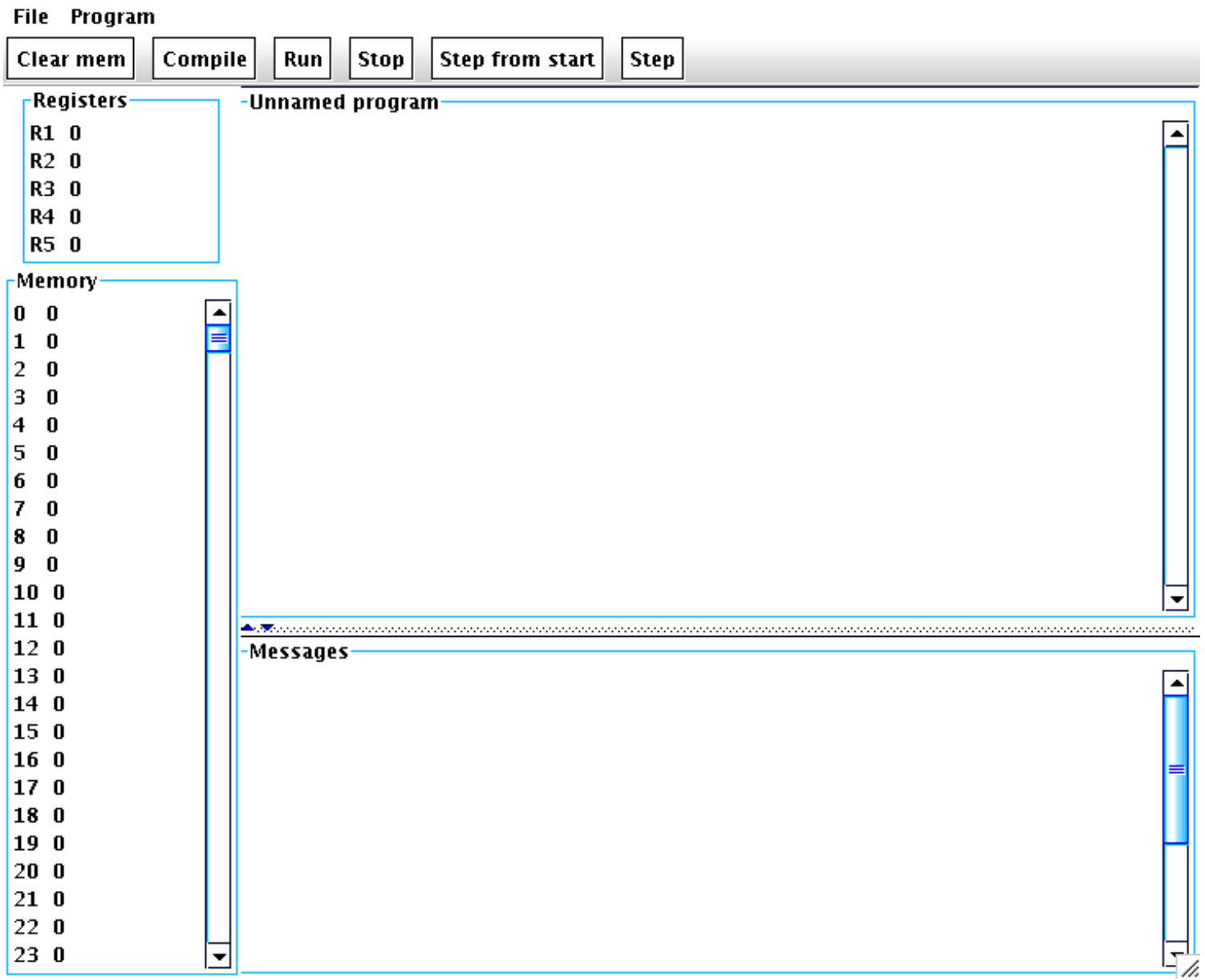
```
x = 1;
y = 0;
while (x < 100) {
    y = y + x;
    x = x + 1;
}
System.out.println(y);
```

```
        move  1,m(0)          ! x = 1
        move  0,m(1)          ! y = 0
loop:   sub   m(0),100,r3     ! r3 = x - 100
        jpos  r3,lpend        ! hoppa om x >= 100
        add   m(1),m(0),m(1)  ! y = y + x
        add   m(0),1,m(0)     ! x = x + 1
        jump  loop            ! hoppa till loop
lpend:  print m(1)            ! skriv ut y
```

Om assemblerspråket

- Man skiljer inte på små och stora bokstäver i program. En add-instruktion kan till exempel skrivas `add`, `ADD`, `Add`, ...
- En kommentar inleds med `!` och sträcker sig till slutet av raden (motsvarar Javas `//`).
- Man kan stoppa in blanka rader var som helst i ett program.

METool


File Program
Clear mem
Compile
Run
Stop
Step from start
Step
Registers
R1 0
R2 0
R3 0
R4 0
R5 0
Unnamed program
Memory
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
16 0
17 0
18 0
19 0
20 0
21 0
22 0
23 0
Messages

Indirekt adressering

Det finns fyra olika typer av parametrar:

...

4. Minne indirekt: den minnescell vars adress finns i ett av registren. Skrivs `M(register)`, till exempel `M(R1)`. Detta är överkurs!

Vad ska man nu ha detta till? Jo, till exempel till att adressera vektorer!

Vektorer i Java

En vektor är en datastruktur som tillåter att man sätter *ett* namn på flera storheter. Man kommer åt de enskilda element i vektorn genom att ge vektorns namn och elementets index (nummer). Index börjar räknas från 0.

Exempel, där vi lagrar talen 0, 1, 4, 9, . . . , 64, 81 i en vektor `v` och därefter summerar talen:

```
int[] v = new int[10];
for (int i = 0; i < 10; i++) {
    v[i] = i * i;
}
int sum = 0;
for (int i = 0; i < 10; i++) {
    sum = sum + v[i];
}
System.out.println(sum);
```

Vektorer i ME

```
        move  0,r1          ! i = 0
sqr:    sub   r1,10,r2      ! r2 = i - 10
        jpos  r2,sqend      ! hoppa till sqend om i - 10 >= 0
        add   25,r1,r2      ! r2 = adressen till v[i]
        mul   r1,r1,m(r2)   ! v[i] = i * i
        add   r1,1,r1       ! i++
        jump  sqr
sqend:  move  0,r3          ! sum = 0
        move  0,r1          ! i = 0
add:    sub   r1,10,r2
        jpos  r2,aend
        add   25,r1,r2
        add   r3,m(r2),r3   ! sum = sum + v[i]
        add   r1,1,r1       ! i++
        jump  add
aend:   print r3            ! skriv ut sum
```

Föreläsning 3 — Datorns hårdvara

- Datorhistorik
- Datorns uppbyggnad, komponenter
- Processor, primärminne, sekundärminne
- Minneshierarkier
- Inbyggda system, stora datorer

Datorhistorik

"I think there is a world market for maybe five computers."

Thomas Watson, IBM, 1943

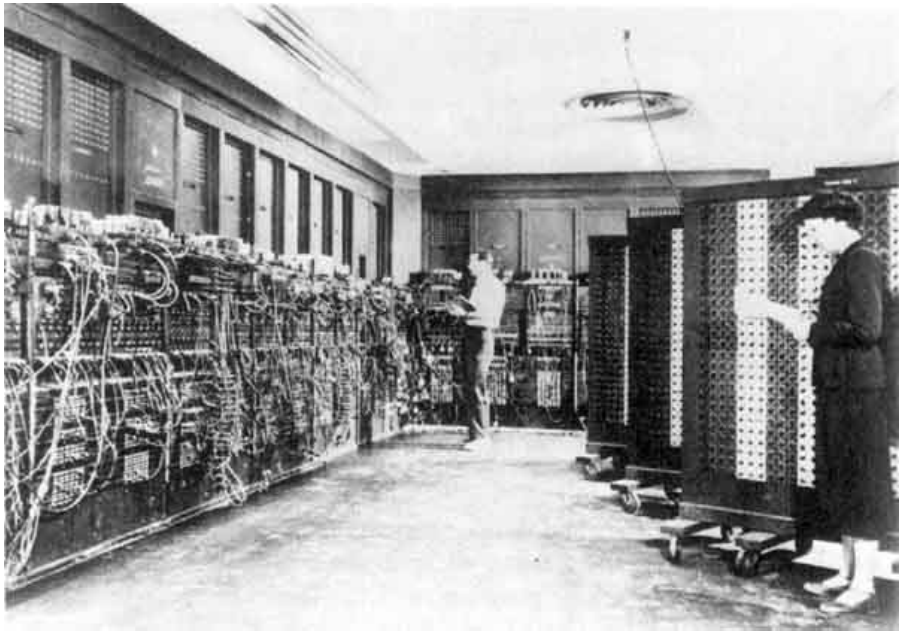
"640K of memory should be enough for anybody."

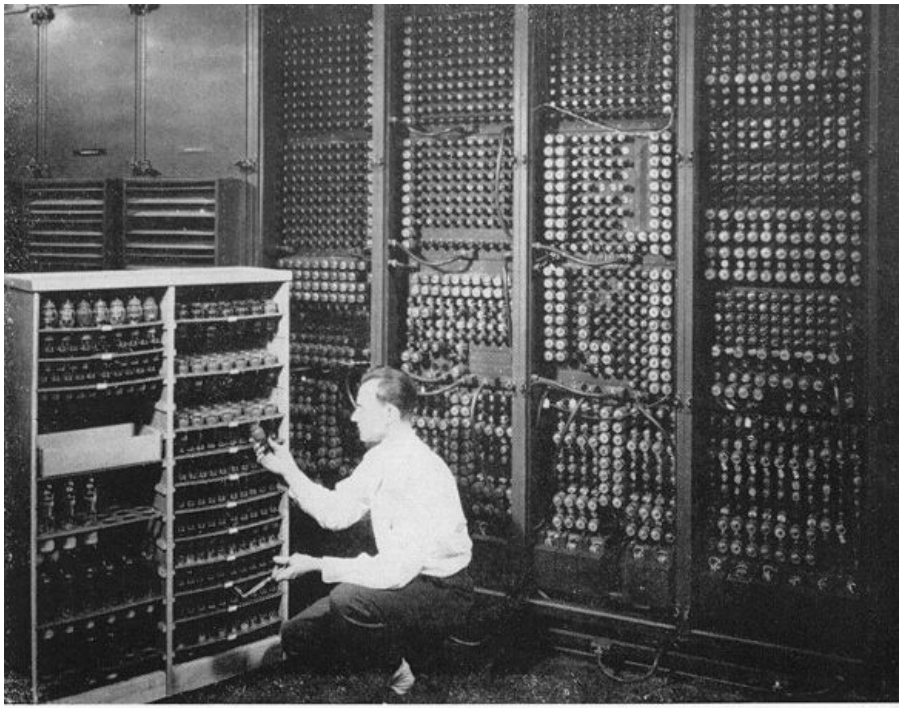
Bill Gates, Microsoft, 1981

`http://computerhistory.org`

Eniac

- 1946: Electronic Numerical Integrator And Calculator
- U.S. War Department, Pennsylvania University
- 60 m, 30 ton, 175 kW, 19 000 elektronrör
- 5000 operationer/s, ”5 kHz”

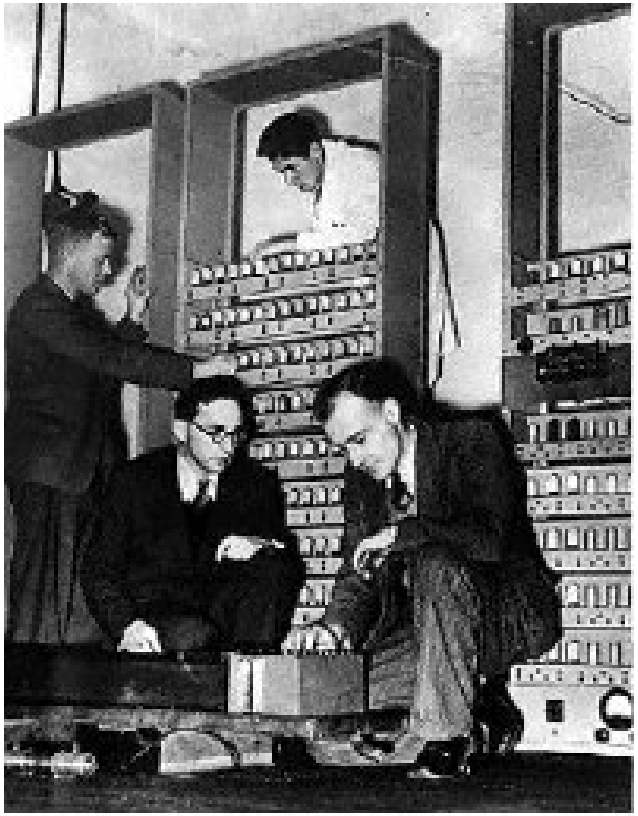


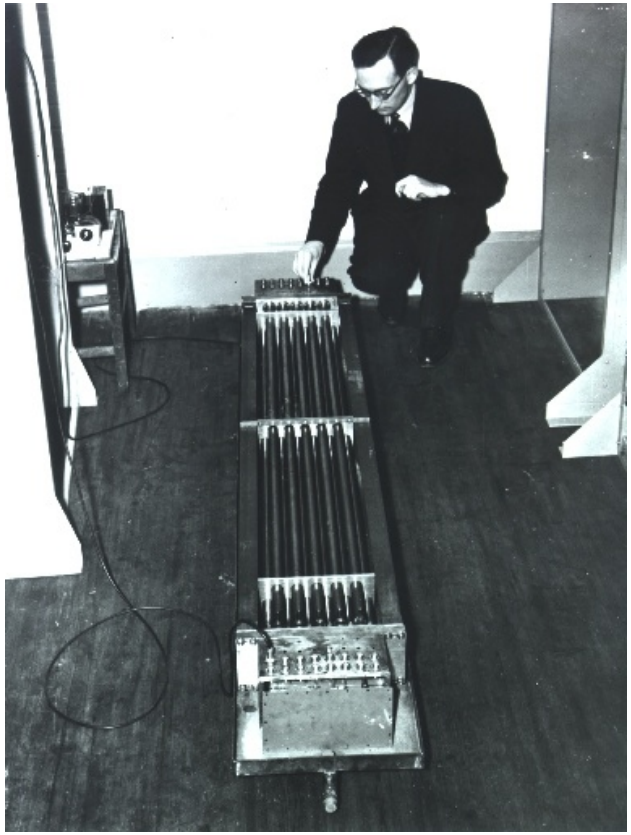


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

EDSAC

- 1949: Electronic Delay Storage Automatic Computer
- University of Cambridge
- Första praktiskt användbara datorn med lagrat program
- Primärminne: rör fyllda med kvicksilver, ultraljud



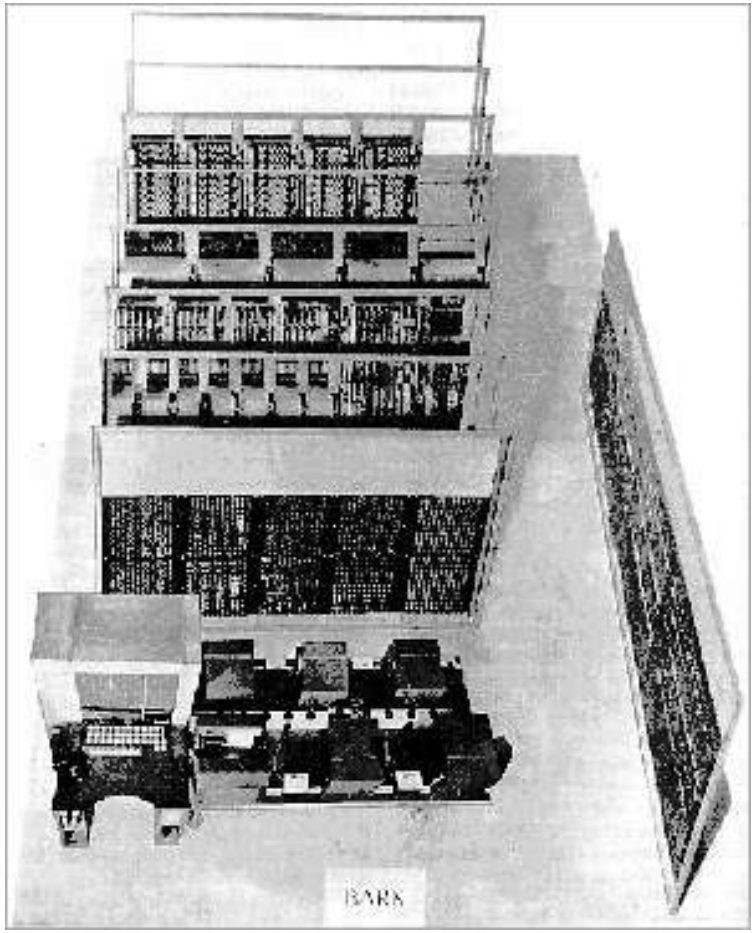


IBM 608

- 1955: IBM 608 var den första kommersiella datorn med transistorer
- 4 500 additioner/s, inköpspris 83 210 $

Sverige, BARK och BESK

- 1950: BARK (Binär Aritmetisk ReläKalkylator)
- 1953: BESK (Binär Elektronisk SekvensKalkylator) 2400 elektronrör, 400 germaniumdioder, $40 \times 512 \times 40$ bitar primärminne (katodstrålerör)



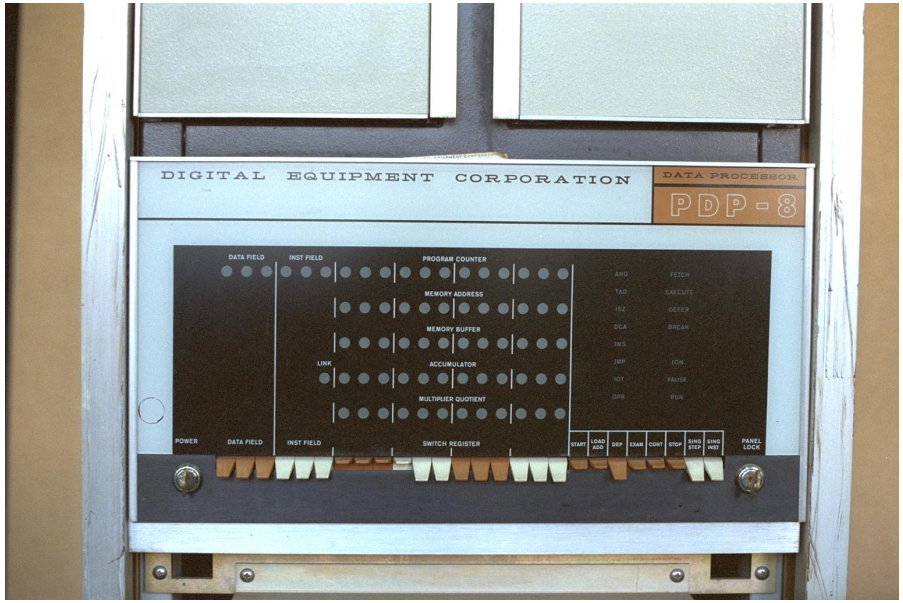
Lund, SMIL

- 1956: SMIL (SifferMaskinen I Lund) Mindre BESK-kopia, användes till 1969

Inuti SMIL

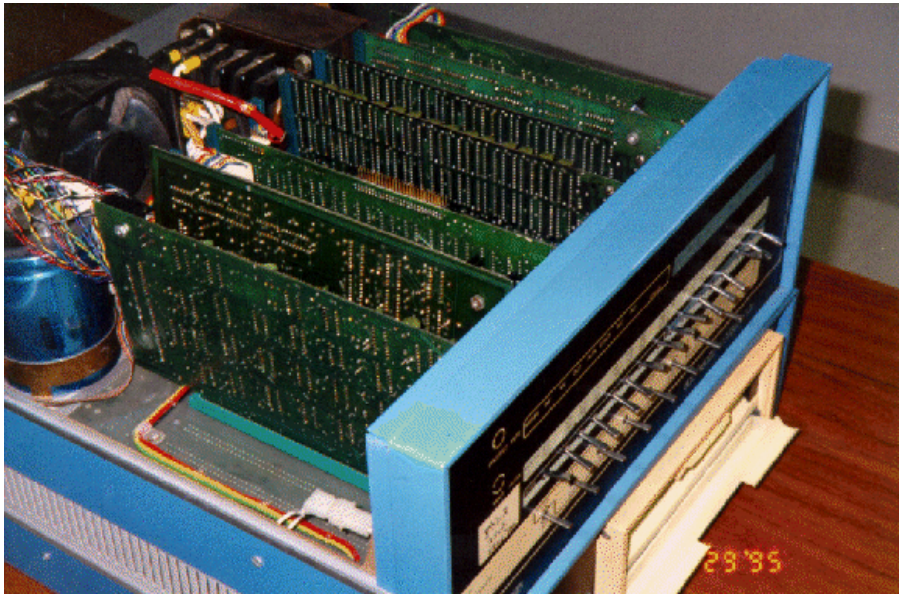
1960-talet, minidatorer

- 1963: PDP-8 (Digital Equipment Corporation)
- 1970: PDP-11, Data General Nova, DEC Vax, ...



1970-talet, mikrodatorer

- 1975: MITS Altair 8800
  Byggsats, 1 200 $, Intel 8080, 256 bytes RAM



IBM PC

- 1981: IBM 5150, 1 200 $
- Intel 8088, 4.75 MHz, 16–256 kB RAM
- MS-DOS
- Microsoft Basic

Datorns uppbyggnad

En dator kan:

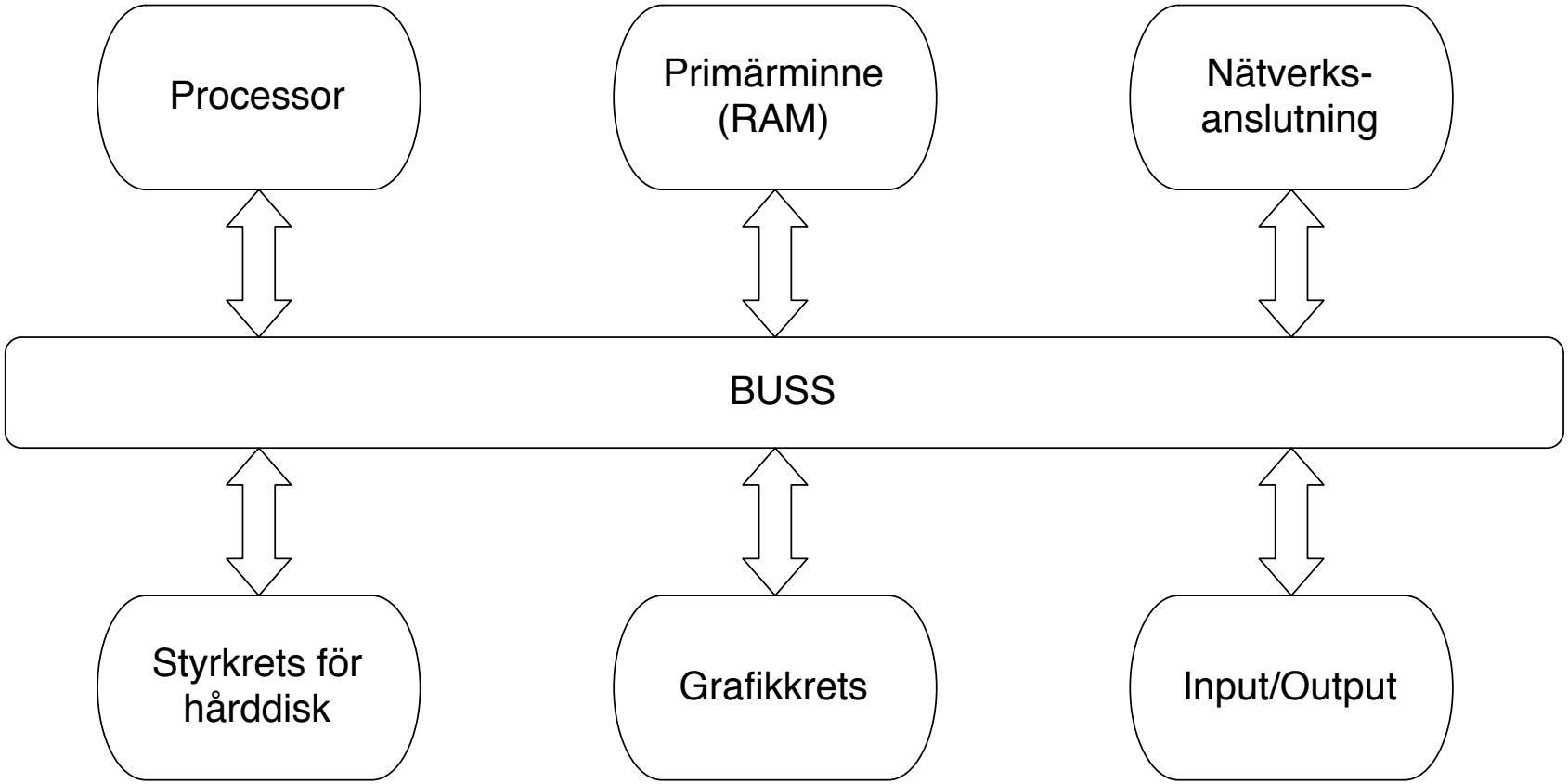
- utföra instruktioner (exekvera program),
- lagra resultat i primärminnet,
- lagra data på sekundärminne, till exempel hårddisk,
- kommunicera med användaren.

Komponenter:

- Processor, CPU (utför instruktioner)
- Primärminne (RAM)
- Läsminne (ROM, PROM, EPROM)
- Sekundärminnen (hårddisk, CD-ROM, . . .)
- Skärm, tangentbord, mus, . . .

Moderkort

Principskiss



Det finns flera bussar: databuss för att skicka data, adressbuss för att skicka adresser, styrbuss för att skicka kontrollsignaler. Bussarna är parallella dvs består av (till exempel) 32 ledningar.

Representation av data

Binära tal, ”nollor och ettor”:

- Till exempel 0 = 0 V, 1 = 5 V
- Större grupper: 1 byte = 8 bitar, 1 ord = 16/32/64 bitar

Vad bitarna betyder beror på sammanhanget:

- Heltal
- Flyttal
- Instruktion
- Text (1 eller 2 byte = 1 tecken)
- Pixel i bild
- Ljud

Processorn

- CPU, Central Processing Unit (Core 2 Duo, Pentium, PowerPC, MIPS, . . .)
- Utför instruktioner:
  - Läs in och avkoda en instruktion från minnet
  - Hämta eventuellt data från register eller minne
  - Utför instruktionen
  - Spara eventuellt data i minnet
- Instruktioner utförs i tur och ordning
  - Särskilda instruktioner för att ”hoppa”
- Register för tillfälliga resultat
  - Varje register innehåller ett ord (till exempel 32 bitar)
  - Snabbare än minnet

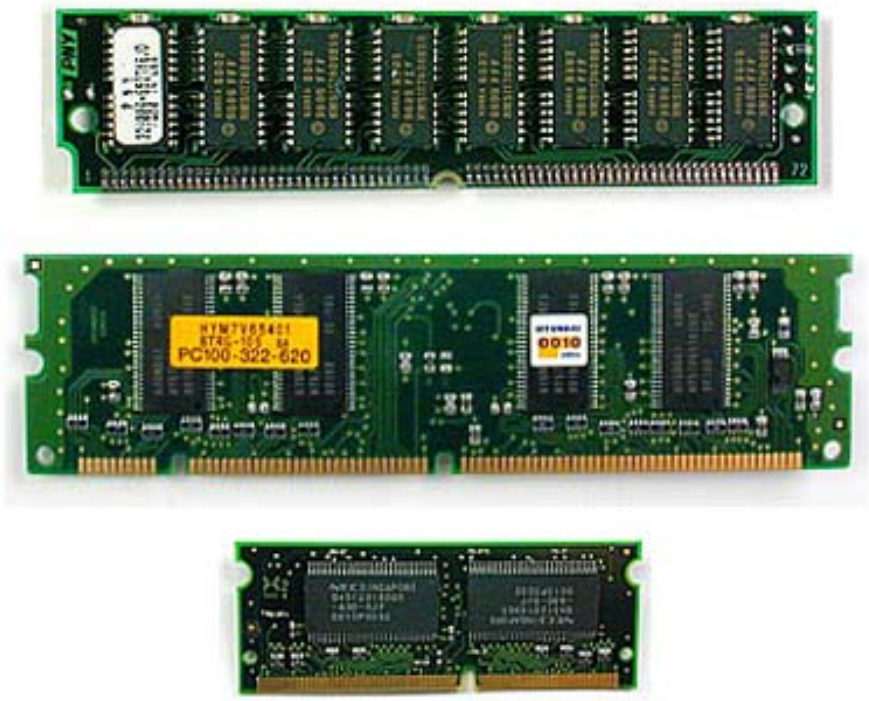
Processorn, forts

- Olika processormodeller har olika instruktionsuppsättning
- Intel x86
  - `MOV`, `ADD`, `SUB`, `MUL`, `DIV`, ...
  - `JMP`, `JL`, `JLE`, `JE`, ... (hopp)
  - Plus mer än 100 andra instruktioner
- Processorns prestanda mäts i Hz (MHz, GHz)
  - Avser klockfrekvensen, ca 1 GHz i en persondator
  - En instruktion kan kräva flera "klocktick"
  - Datorns prestanda beror inte bara på klockfrekvensen!

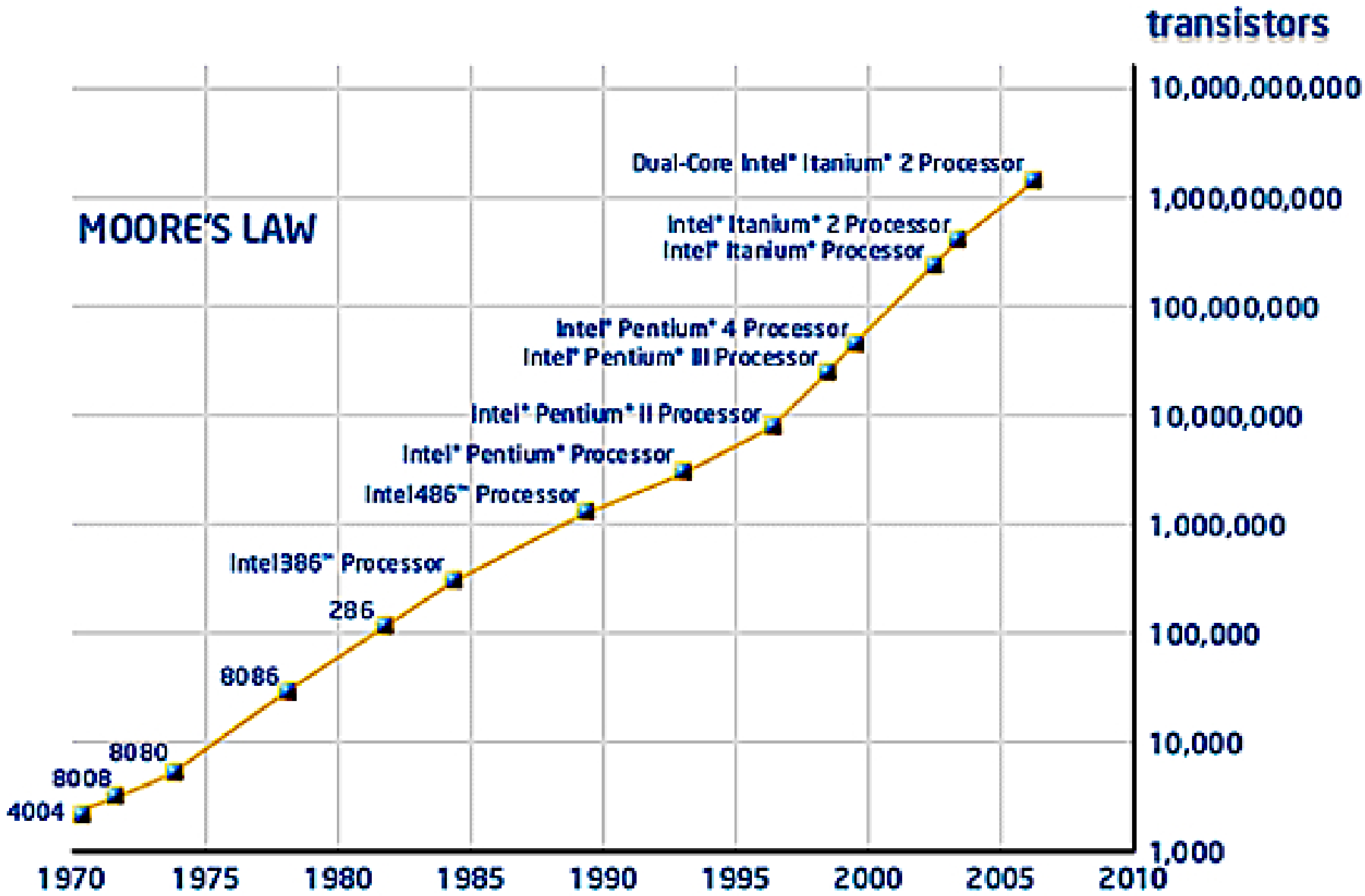
Primärminne

Cirka 1 GB i en persondator.

- SRAM: snabbt, dyrt (4–6 transistorer per bit), stabilt
- DRAM: långsammare, billigare (1 transistor per bit), måste skrivas om flera hundra gånger i sekunden

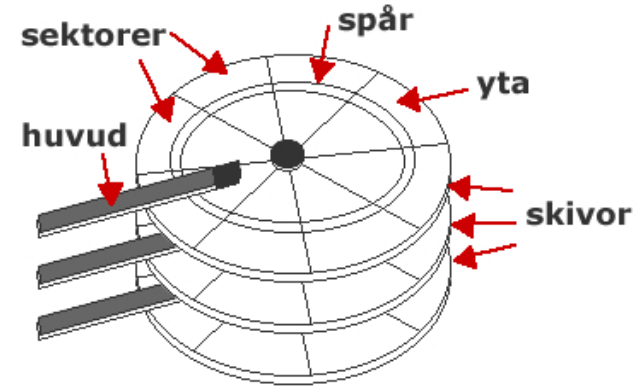


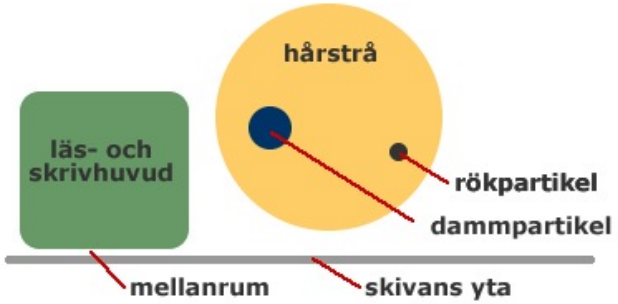
Moores lag


transistors
MOORE'S LAW
Dual-Core Intel® Itanium® 2 Processor
Intel® Itanium® 2 Processor
Intel® Itanium® Processor
Intel® Pentium® 4 Processor
Intel® Pentium® III Processor
Intel® Pentium® II Processor
Intel® Pentium® Processor
Intel486™ Processor
Intel386™ Processor
286
8086
8080
8008
4004
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010

Sekundärminne — hårddisk

- Roterande skivor, magnetisk beläggning
- Exempel: Seagate, 7200 rpm, 250GB, 8 MB cache. Cirka 400–500 kr.





Minneshierarki, 1

- Sekundärminnet är stort (många GB), billigt och långsamt.
- Primärminnet är mindre (någon GB), dyrare och snabbare.
- Men processorn är mycket snabbare än primärminnet.
- Cacheminne:
  - Data som används ofta lagras i litet (några MB), dyrt och snabbt minne
  - Data som använts nyss används sannolikt snart igen
  - När man läser från minnet sparas en kopia i cacheminnet
  - Cacheminnen kan byggas upp i flera nivåer

Minneshierarki, 2

Primärminnet räcker oftast inte till för alla program.

Virtuellt minne:

- Dela in programmen i sidor, dela in primärminnet i sidramar.
- Varje sida finns antingen i minnet eller på hårddisk.
- Vid behov hämtas en sida in från hårddisken, eventuellt offras en gammal sida.
- Blir långsammare än om hela programmet ligger i primärminnet.

Inbyggda system

- Dator som ingår som en komponent i en produkt (bil, mobiltelefon, kamera, industrirobot, flygplan, . . .)
- De allra flesta datorerna finns i inbyggda system
- Exempel: Axis 210, webbkamera:
  - 30 bilder/s, 640x480, MPEG-4
  - Inbyggd rörelsedetektering
  - Webbserver, larm via e-post
  - ETRAX 100 LX-processor, ARTPEC-2 bildkomprimeringskrets, 4 MB flashminne, 16 MB RAM
  - $38 \times 88 \times 146$ mm, 305 g
  - ca 4 000 kr

Det finns fortfarande stora datorer

Sun Fire E25K Server

- Upp till 72 processorer (Ultrasparc IV)
- Upp till 1000 GB primärminne
- Upp till 120 TB hårddisk
- $85 \times 166 \times 191$ cm, 1122 kg
- Pris från 750 000 $

Föreläsning 4 — LaTeX

Förberedelse inför laboration 3.

- Ordbehandling
- LaTeX
- Mall för rapport
- Dokumentstruktur: dokumentklasser, omgivningar, text, stycken, listor, tabeller, ...
- Programlistor
- Matematiska formler
- Bilder

Ordbehandling

De flesta moderna ordbehandlare, till exempel Microsoft Word, fungerar enligt WYSIWYG-principen:

What You See Is What You Get

Det innebär att det man ser på skärmen ser likadant ut som det som kommer att skrivas på papperet: teckensnitt, storlekar, avstånd, . . . Det innebär också att det inte blir bättre än vad det ser ut på skärmen (What You See Is *All* You Get).

Layout av text

I de flesta ordbehandlare finns det formatmallar där man till exempel kan bestämma att alla rubriker på en viss nivå ska ha ett visst utseende. Om man vill ändra utseendet på alla rubriker så räcker det att ändra i mallen.

Det brukar också finnas möjlighet till automatisk numrering av rubriker, automatisk generering av innehållsförteckning och sakregister och liknande.

När man skriver matematisk text använder man ofta en ekvationseditor för att skriva de matematiska symbolerna. Ekvationseditorer är inte enkla att använda, och slutresultatet brukar inte bli bra.

Med LaTeX arbetar man på ett helt annat sätt: man skriver texten i en vanlig textfil och lägger in *kommandon* (”taggar”) i texten som visar hur texten ska formateras. Textfilen kan bli något svårläst, åtminstone innan man är van, men resultatet blir garanterat snyggt.

Enkelt exempel:

```
Pythagoras sats ser ut så
här: $a^2 + b^2 = c^2$.
```

Pythagoras sats ser ut så här: $a^2 + b^2 = c^2$.

$-tecknen anger att en matematisk formel börjar och slutar. LaTeX vet då att variablerna a, b och c ska skrivas kursiva, hur stora exponenterna ska vara och var de ska placeras, och hur mycket mellanrum det ska vara mellan termerna.

Ett större exempel

```
If $f$ is continuous on the closed interval
$a \leq x \leq b$ and differentiable on the
open interval $a < x < b$, then there exists
a point $\xi$, $a < \xi < b$ such that
\begin{displaymath}
   f(b) - f(a) = f'(\xi)(b -a).
\end{displaymath}
```

If f is continuous on the closed interval $a \leq x \leq b$ and differentiable on the open interval $a < x < b$, then there exists a point ξ, $a < \xi < b$ such that

$$f(b) - f(a) = f'(\xi)(b - a).$$

LaTeX-historik

Donald E. Knuth skrev 1977–1982 typsättningsprogrammet TeX[1] eftersom han inte var nöjd med de möjligheter till typsättning som fanns då. TeX är ett ”lågnivåspråk”. Leslie Lamport byggde på TeX med ett makropaket som gör det möjligt för författaren av ett dokument att koncentrera sig på den logiska strukturen hos dokumentet och på själva texten i stället för på lågnivåtypsättningen. Resultatet blev LaTeX[2].
En föregångare till LaTeX, *troff*, används fortfarande ibland, till exempel till Unix man-sidor.

[1] TeX skrivs TeX i skrivmaskinsskrift och uttalas ”tech”.
[2] LaTeX skrivs LaTeX i skrivmaskinsskrift och uttalas ”lah-tech”.

Arbeta med LaTeX

När man använder LaTeX utgår man från en fil med text och kommandon. Filen ska ha tillägget `.tex`, till exempel `rapport.tex`. Sedan "översätter" man filen till `pdf`-format med programmet `pdflatex` och tittar på resultatet med en pdf-läsare, till exempel `evince`. Detta kan man naturligtvis göra genom att skriva kommandona för hand (`gedit rapport.tex`, `pdflatex rapport.tex`, `evince rapport.pdf`), men det är enklare att använda ett specialprogram. På studentdatorerna finns programmen `texmaker` (enklast) och `kile`. På Mac-datorer använder man TeXShop.

I stället för att generera pdf-filer med `pdflatex` kan man generera dvi-filer ("device independent") med programmet `latex` som man kan titta på med en "dvi-läsare" och sedan översätta till Postscript eller pdf. Numera använder de flesta `pdflatex`.

Mall för rapport

```
\documentclass[a4paper]{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[swedish]{babel}
\usepackage{fancyvrb}
\fvset{tabsize=4}
\fvset{fontsize=\small}
\title{Programmeringsteknik\\
   Inlämningsuppgift 1}
\author{Xerxes Yngvesson\\
   dat14xyn@student.lu.se}
\date{2014--10--17}

\begin{document}
\maketitle

Här skriver man texten i
rapporten.
\end{document}
```

Programmeringsteknik
Inlämningsuppgift 1

Xerxes Yngvesson
dat14xyn@student.lu.se
2014–10–17

Här skriver man texten i rapporten.

Dokumentklasser och omgivningar

`{article}` är en *dokumentklass* (den man oftast använder). Andra dokumentklasser är `{report}`, `{book}`, `{letter}` och `{beamer}` (beamer används för overheadbilder). En dokumentklass påverkar utseendet på hela dokumentet.

`\begin{document}` definierar starten på en *omgivning*, `\end{document}` slutet på omgivningen. En omgivning påverkar utseendet på den del av dokumentet som ingår i omgivningen. Vi kommer att se exempel på andra omgivningar senare.

Löpande text

Radslut och antal mellanslag mellan ord har ingen betydelse, LaTeX formaterar så att det blir snyggt. En eller flera blanka rader ger ett nytt stycke. Exempel:

```
Det här
är en text som jag
har    skrivit. Det är
en lång text med flera
rader.

Här börjar det ett
nytt stycke i texten.
```

Det här är en text som jag har skrivit. Det är en lång text med flera rader.

Här börjar det ett nytt stycke i texten.

Rubriker

LaTeX numrerar rubriker automatiskt. Man anger en rubrik med `\section` eller `\subsection`.

```
\section{Inledning}
\section{Utförande}
\subsection{Del 1}
\subsection{Del 2}
\section{Slutsatser}
```

1 Inledning

2 Utförande

2.1 Del 1

2.2 Del 2

3 Slutsatser

Ändra textens utseende

Det finns många kommandon för att ändra utseende på texten. Två sådana kommandon är `\emph` för att betona text och `\texttt` för att skriva med skrivmaskinstypsnitt. Exempel:

```
Här skriver jag något
\emph{viktigt}. Och
i Java har vi använt
klassen \texttt{Square}.
```

Här skriver jag något *viktigt*. Och i Java har vi använt klassen `Square`.

Det finns också kommandon för fetstil, lutande text, osv, och för att ändra storlek på texten. Använd sparsamt!

Specialtecken

Med tecknet % inleder man en kommentar som sträcker sig till slutet av raden.

En del tecken används för kommandon och måste skrivas på speciellt sätt:

```
\$ \% \_ \# \& \{ \} \textbackslash
```

Det finns streck, mellanrum och punkter av olika slag:

```
DoD-kursen pågår under vecka
1--3 av läsperiod ht1. Sedan
börjar Programmeringsteknik \ldots

\quad Telefon: 046--222~80~38.
Dagens datum: \today.
```

DoD-kursen pågår under vecka 1–3 av läsperiod ht1. Sedan börjar Programmeringsteknik . . .

Telefon: 046–222 80 38. Dagens datum: 29 augusti 2016.

Fotnoter

Fotnoter är lätta att skriva:

```
Om man använder \LaTeX
\footnote{uttalas
''lah-tech''} så
blir det bra. Alla rapporter
blir automatiskt snyggt
utformade.
```

Om man använder LaTeX[a] så blir det bra. Alla rapporter blir automatiskt snyggt utformade.

[a] uttalas "lah-tech"

Fotnoter numreras automatiskt 1,2,... Fast här blev "numret" på fotnoten "a" av olika anledningar. Observera att man skriver två apostrofer (’’) i stället för citationstecken (").

Listor

Punktlistor är enkla:

```
\begin{itemize}
  \item första punkten
  \item här kommer den andra
  punkten i listan
\end{itemize}
```

- första punkten
- här kommer den andra punkten i listan

Numrerade listor är lika enkla:

```
\begin{enumerate}
  \item första punkten
  \item här kommer den andra
  punkten i listan
\end{enumerate}
```

1. första punkten
2. här kommer den andra punkten i listan

I detta dokument används dokumentklassen `beamer`, och där blir numren siffror i cirklar. I den vanliga dokumentklassen `{article}` blir numren 1., 2., ...

Definitioner

```
Några klasser som vi använder:

\begin{description}
\item[SimpleWindow] Beskriver ett
enkelt ritfönster
\item[Scanner] Inläsning från
tangentbordet
\item[Random] Slumptal
\end{description}
```

Några klasser som vi använder:

SimpleWindow Beskriver ett enkelt ritfönster

Scanner Inläsning från tangentbordet

Random Slumptal

I dokumentklassen `article` blir det något annorlunda layout på definitioner. Använd en `tabular`-omgivning med kolumnspecifikationen `p{bredd}` för att få layout som liknar den ovan.

Tabeller

En tabell där den första kolumnen är vänsterinpassad (`l`), den andra centrerad (`c`) och den tredje högerinpassad (`r`). `&` avgränsar kolumnerna, `\\` betyder ny rad, `~` är ett "hårt" blanktecken. `\hline` är ett streck.

```
\begin{tabular}{lcr}
  Produkt  & Typ    & Pris    \\
  \hline
  Skruvar  & stora  & 0.18~kr \\
  Muttrar  & M16    & 0.38~kr \\
  Spikar   & 12~tum & 0.12~kr
\end{tabular}
```

Produkt	Typ	Pris
Skruvar	stora	0.18 kr
Muttrar	M16	0.38 kr
Spikar	12 tum	0.12 kr

Flytande tabeller

Med en \table-omgivning skapar man en tabell med en förklarande text och ett nummer. LaTeX placerar tabellen där det är lämpligt.

```
\begin{table}
\begin{tabular}{lcr}
  Produkt  & Typ    & Pris    \\
  \hline
  Skruvar  & stora  & 0.18~kr \\
  Muttrar  & M16    & 0.38~kr \\
  Spikar   & 12~tum & 0.12~kr
\end{tabular}
\caption{Våra produkter}
\end{table}
```

Produkt	Typ	Pris
Skruvar	stora	0.18 kr
Muttrar	M16	0.38 kr
Spikar	12 tum	0.12 kr

Tabell 7. Våra produkter

Att referera till etiketter

Om man sätter en etikett på en tabell kan man referera till den från texten. Exempel:

```
\begin{table}
\begin{tabular}{lcr}
  Produkt  & Typ    & Pris     \\
  \hline
  Skruvar  & stora  & 0.18~kr  \\
\end{tabular}
\caption{Våra produkter}
\label{produkter}
\end{table}
Senare i texten: våra produkter
finns i tabell~\ref{produkter}.
```

Produkt	Typ	Pris
Skruvar	stora	0.18 kr

Tabell 7. Våra produkter

Senare i texten: våra produkter finns i tabell 7.

Figurer hanteras likadant som tabeller, i en `\figure`-omgivning.

Programlistor

För att infoga en programlista i en rapport använder man kommandot `\VerbatimInput{filnamn}` från paketet `fancyvrb`. Man bör *inte* använda ”standard”- kommandot `\verbatiminput` eftersom det kommandot ignorerar alla tabulatortecken i programmet, och det medför att indragningarna försvinner.

```
\usepackage{fancyvrb}
\fvset{tabsize=4}
\fvset{fontsize=\small}

\VerbatimInput{Point.java}
```

```
class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Öka eller minska avstånd

Ibland behöver man öka avståndet i vertikalled mellan två avsnitt i texten, till exempel före eller efter en tabell. Det kan man göra med kommandot `\vspace{längd}`, där längden kan anges i millimeter eller punkter eller något annat som LaTeX känner igen. Längden kan vara negativ om man vill minska avståndet. Det finns också specialkommandon för att lägga in ett litet, mellanstort och stort avstånd:

```
\smallskip  \medskip  \bigskip
```

Man kan öka eller minska horisontellt avstånd med `\hspace{längd}`.

Matematiska formler

LaTeX är *mycket* bra på att formatera matematisk text. Alla (tror jag) artiklar och böcker som innehåller matematiska formler är skrivna med LaTeX. Man kan skriva formler antingen inuti löpande text eller på en egen rad:

- I texten: formeln inleds med `$` och avslutas med `$`.
- På egen rad: formeln inleds med `\begin{displaymath}` och avslutas med `\end{displaymath}`. `\begin{equation}` och `\end{equation}` ger samma resultat men formeln numreras. Med `\label` och `\ref` kan man etikettera och referera till ekvationer.

Enkla formler

```
Formeln $x=3y-2$ står
inne i texten. Däremot
står
\begin{displaymath}
  x=3y-2
\end{displaymath}
för sig själv precis som
\begin{equation}
  x=3y-2
  \label{xochy}
\end{equation}
I ekvation~\ref{xochy} fann
vi att \ldots
```

Formeln $x = 3y - 2$ står inne i texten. Däremot står

$$x = 3y - 2$$

för sig själv precis som

$$x = 3y - 2 \tag{1}$$

I ekvation 1 fann vi att ...

Symboler, index

```
\begin{displaymath}
\alpha \leq \pi \approx 3.141592654
\end{displaymath}
```

$$\alpha \leq \pi \approx 3.141592654$$

```
\begin{displaymath}
x_{k+1}=x_{k}-f(x_{k})/f'(x_{k})
\end{displaymath}
```

$$x_{k+1} = x_k - f(x_k)/f'(x_k)$$

Exponenter, rötter

```
\begin{displaymath}
e^x = 1+x+x^2/2!+x^3/3!+\cdots
\end{displaymath}
```

$$e^x = 1 + x + x^2/2! + x^3/3! + \cdots$$

```
\begin{displaymath}
x_{1,2}=\frac{p}{2}\pm
\sqrt{\frac{p^2}{4}-q}
\end{displaymath}
```

$$x_{1,2} = \frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

Integraler, summor

```
\begin{displaymath}
  \int_{-\infty}^{\infty}
  e^{-x^2} dx
\end{displaymath}
```

$$\int_{-\infty}^{\infty} e^{-x^2} dx$$

```
\begin{displaymath}
  \sum_{k=1}^n\frac{1}{a_k}
\end{displaymath}
```

$$\sum_{k=1}^{n} \frac{1}{a_k}$$

Funktioner

```
\begin{displaymath}
  \sin^2 x + \cos^2 x = 1
\end{displaymath}
```

$$\sin^2 x + \cos^2 x = 1$$

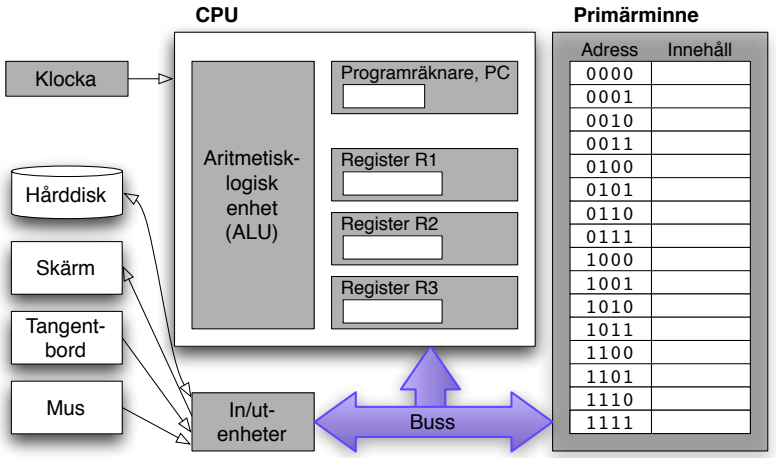
Matriser, parenteser

```
\begin{displaymath}
A=\left(
\begin{array}{cccc}
a_{11} & a_{12} & \cdots & a_{1n}\\
a_{21} & a_{22} & \cdots & a_{2n}\\
\vdots & \vdots & \ddots & \vdots\\
a_{n1} & a_{n2} & \cdots & a_{nn}\\
\end{array}
\right)
\end{displaymath}
```

$$A = \left(\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array} \right)$$

Bilder

Bilder kan inkluderas i LaTeX-dokument om de är i formatet `pdf`, `jpeg` eller `png` (`eps` om man använder `latex`). Man måste använda paketet `graphicx` (eller `graphics`).




```
\usepackage{graphicx}

\includegraphics[height=40mm]{bild.pdf}
```

ImageMagick-programmet `convert` kan konvertera från och till de flesta bildformat:

```
convert bild.fig bild.pdf
```

Egna kommandon

Man kan lätt definiera egna kommandon, till exempel ett kortare namn för en text som man använder ofta. Kommandon kan ha parametrar.

```
\newcommand{\java}[1]
{\texttt{#1}}

Klasser: \java{Random},
\java{Scanner} och
\java{PrintStream}.
```

Klasser: `Random`, `Scanner` och `PrintStream`.
Man kan definiera om existerande kommandon med `\renewcommand`. Det kan ställa till förvirring, så gör inte det.

LaTeX på egen dator

En sammanfattning av LaTeX-installationer finns på `www.latex-project.org`, sidan Getting LaTeX.

Linux LaTeX kanske redan finns på datorn; hämtas annars med den vanliga pakethanteraren.

Mac Använd MacTeX (bygger på TeXLive, som uppdateras varje år).

Windows proTeXt verkar vara enklast.

Som IDE rekommenderas Texmaker (`www.xm1math.net/texmaker`) eller TeXShop (bara för Mac, `www.texshop.org`).

Föreläsning 5 — Internet

Förberedelse inför laboration 4.

- Historik
- Protokoll
- Ethernet, TCP/IP
- Adressering, namnservrar
- WWW, HTML

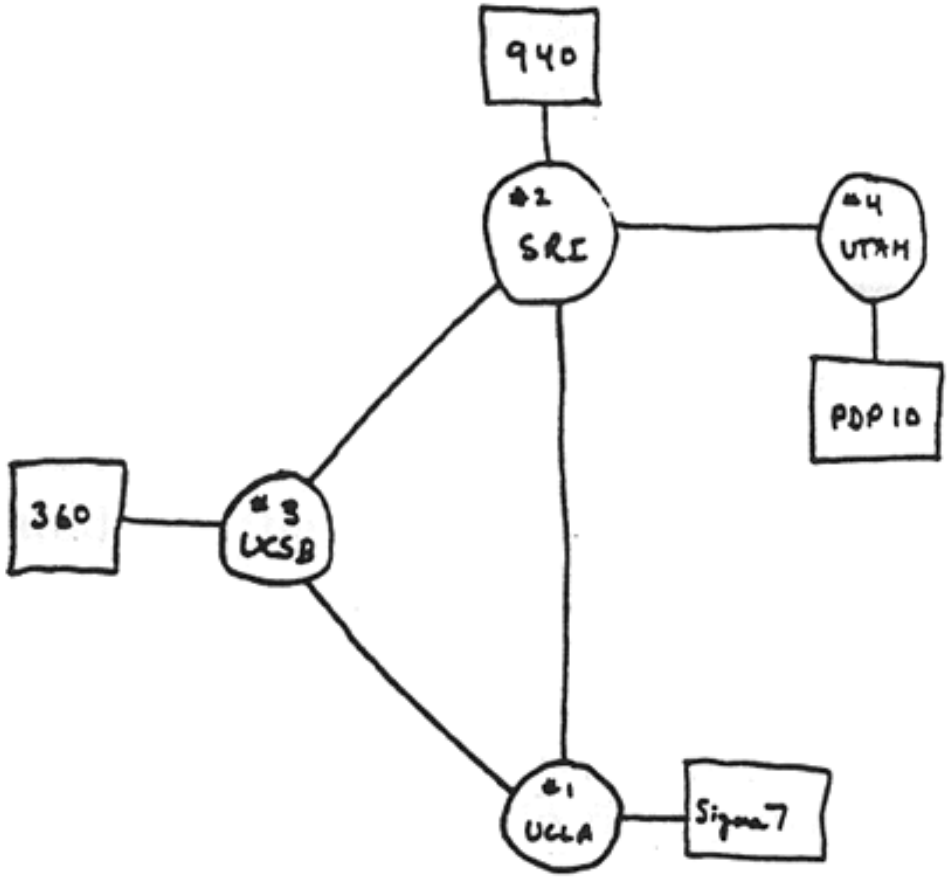
Olika slags datornätverk

- LAN — Local Area Network
  - `student.lth.se` (ganska stort LAN)
- MAN — Metropolitan Area Network
  - LUNET (Lund University Network)
- WAN — Wide Area Network
  - SUNET (Swedish University Network)
- Uppkoppling hemifrån till Internet
  - Telenätet, modem (56 kbps)
  - Telenätet ADSL, kabel-tvnätet (0,5–24 Mbps)
  - Direkt (Ethernet, 10–1000 Mbps)

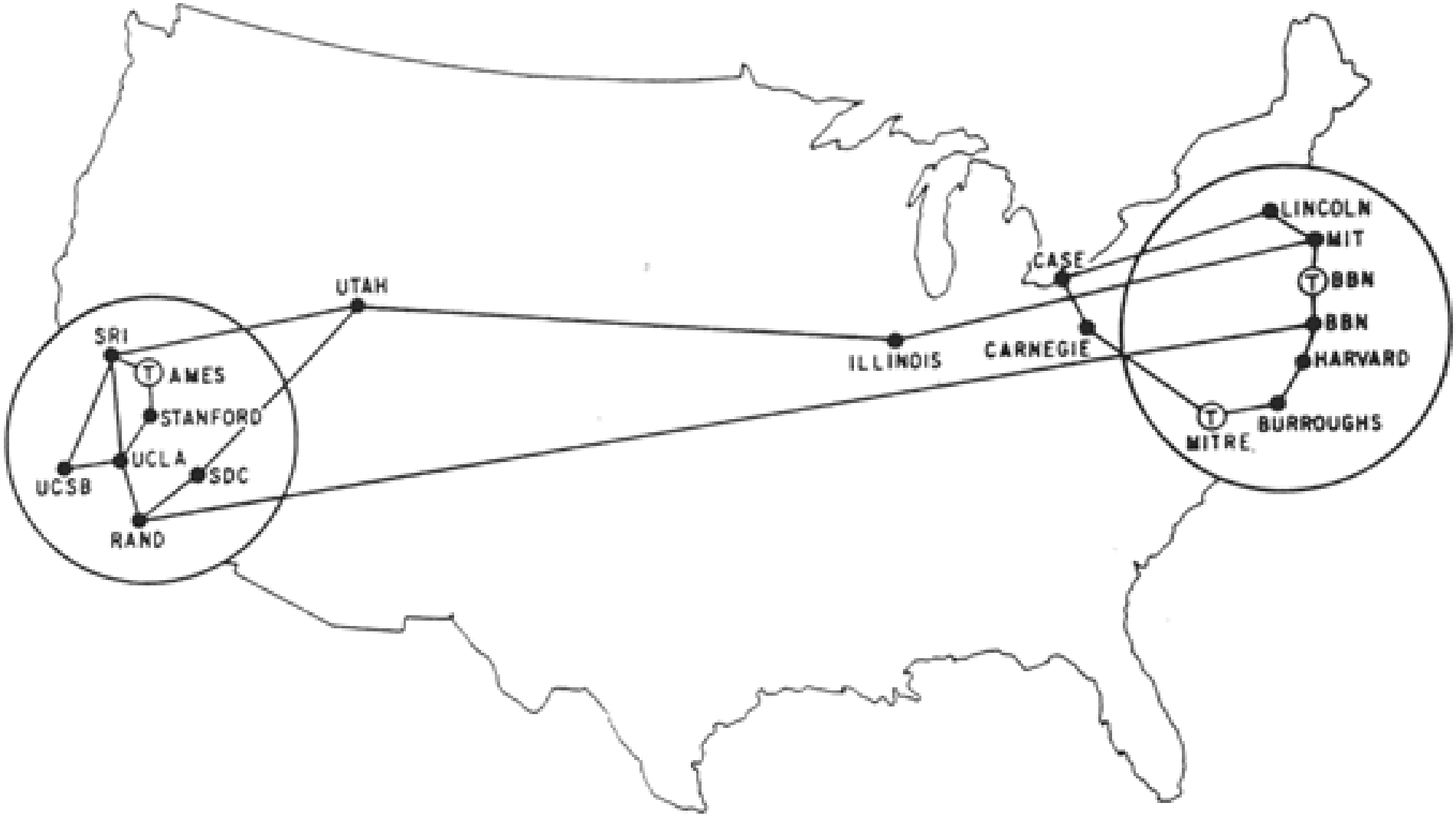
Internet började med ARPANET

ARPANET, 1969–1990:

- U.S. Defense Advanced Research Projects Agency
- Hopkoppling av datorer av olika slag
- Ingen central knutpunkt
- Robust mot fel

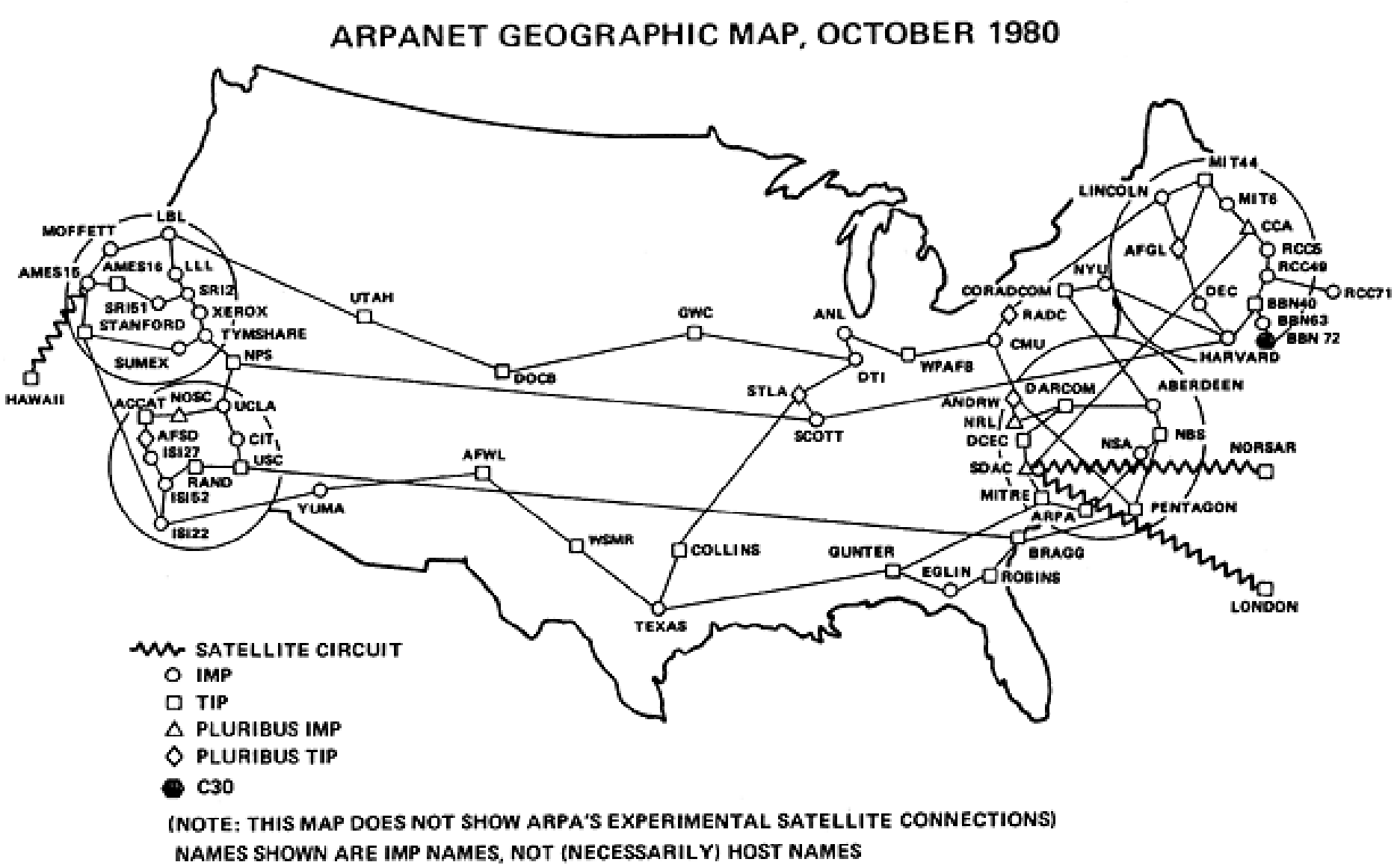


ARPANET, 1971

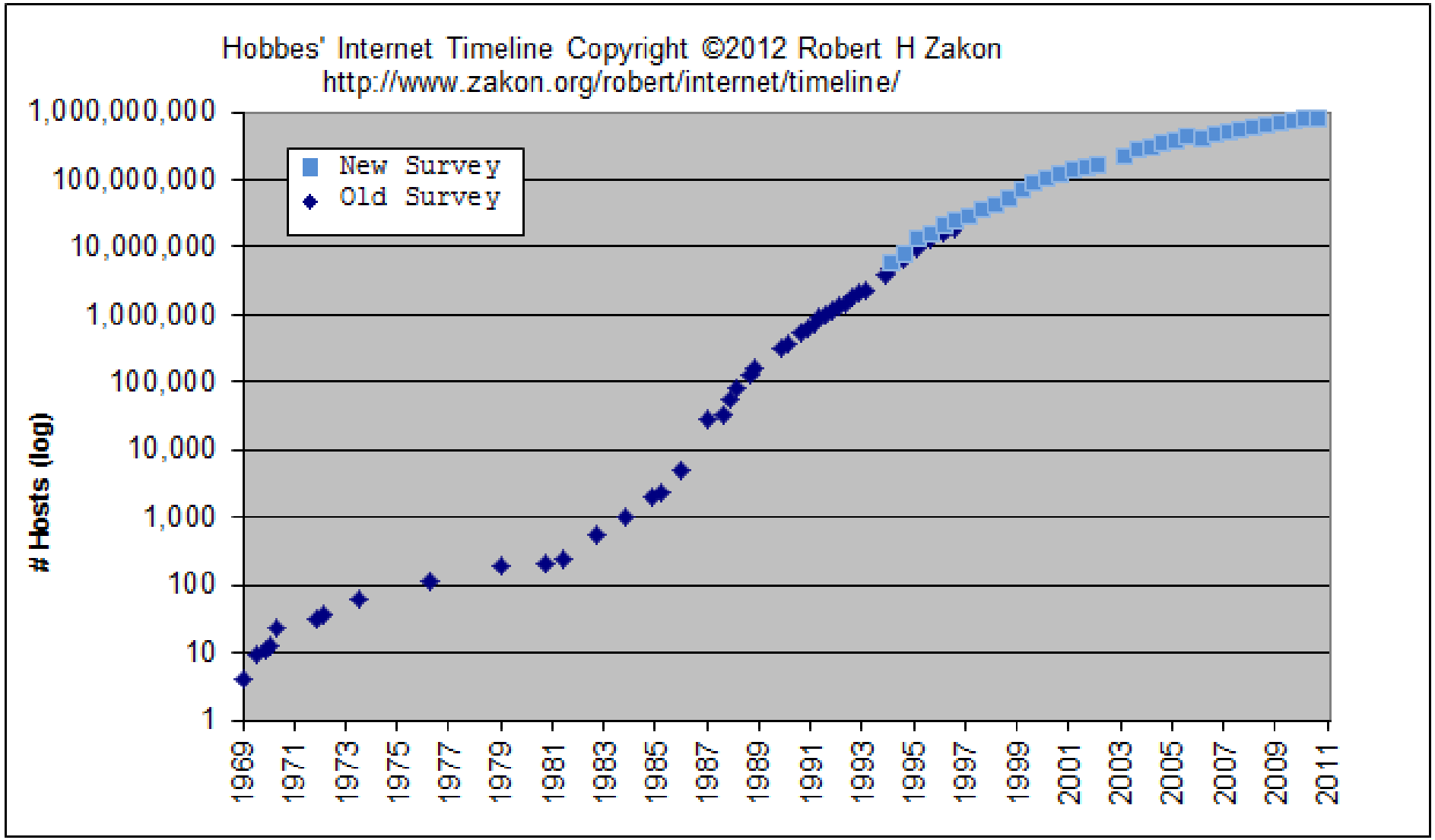


MAP 4 September 1971

ARPANET, 1980



Internet, tillväxt



Protokoll

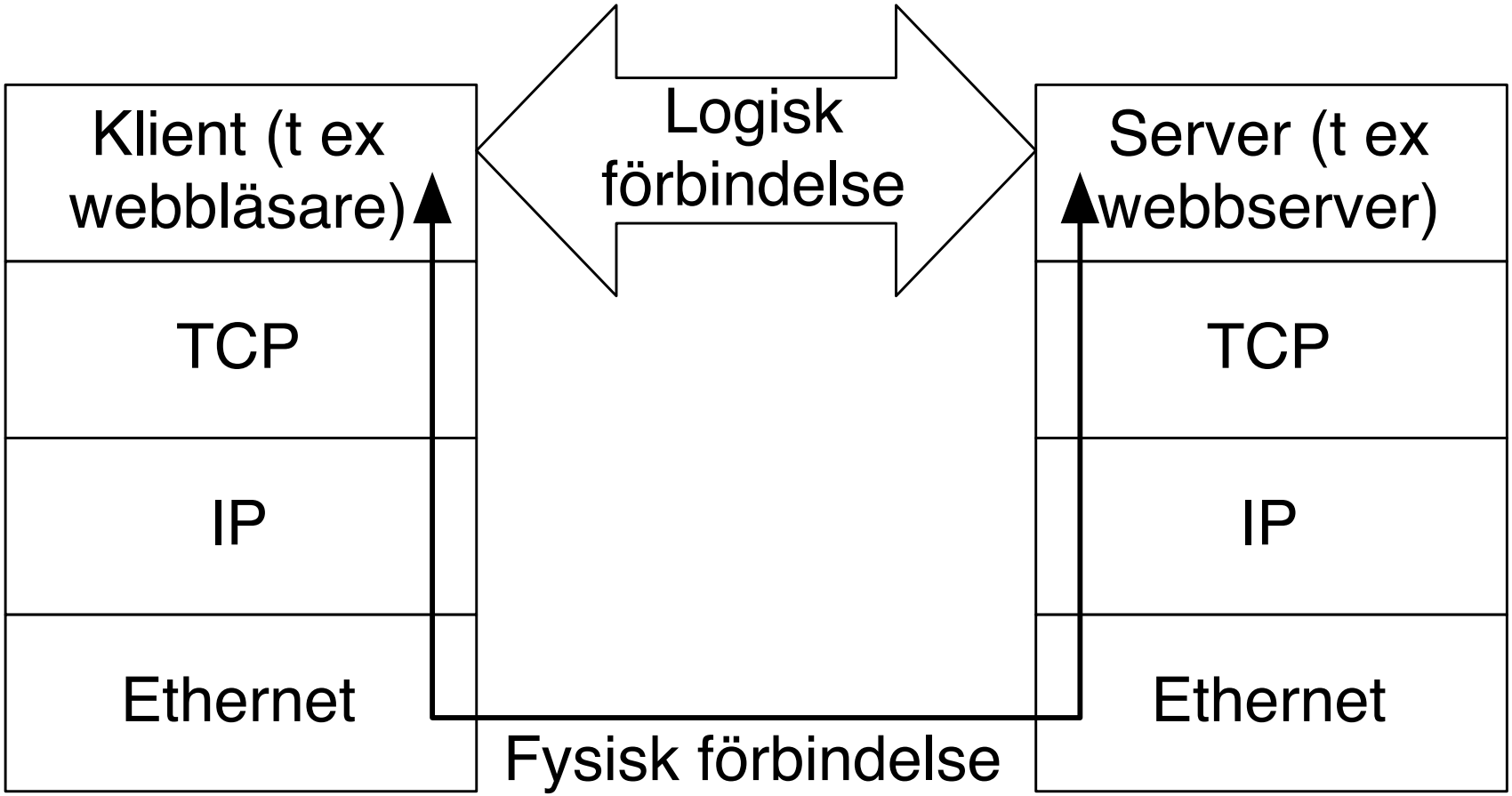
Regler för hur kommunikation ska gå till:

- TCP/IP (1974). Transmission Control Protocol / Internet Protocol. Två grundläggande protokoll för datautbyte på Internet (i vilken ordning ska data skickas, i hur stora paket, till vem, från vem, felhantering, . . .).

Protokoll för tjänster:

- SMTP (1982), Simple Mail Transfer Protocol. För att skicka e-post.
- POP (1984), Post Office Protocol. För att hämta e-post.
- FTP (1985), File Transfer Protocol. För att flytta filer.
- HTTP (1989), HyperText Transfer Protocol. För att hämta HTML-dokument, ”webbsidor”.

Lager i kommunikationen



Ethernet och TCP/IP

Ethernet

- Lågnivåprotokoll, används i lokalt nätverk. Data skickas till alla datorer som är uppkopplade.

IP

- Bygger på Ethernet.
- Kan sända och ta emot paket av data; ingen kontroll av att paketen kommer fram.

TCP

- Bygger på IP.
- Vid sändning delas data upp i paket, vid mottagning sätts paketen ihop.
- Tillhandahåller pålitlig förbindelse — felkontroll, kvittens av paket, omsändning vid behov.

Servrar och klienter

Servrar tillhandahåller tjänster:

- filer, webbsidor (HTML-filer), e-post, ...

Klienter använder tjänster. Exempel, e-post:

- Skriv ett brev i e-postprogrammet (klienten)
- Tryck på ”Skicka” — e-postprogrammet skickar brevet till e-postservern
- E-postservern skickar brevet till rätt adress
- Brev som kommer till dig lagras i en e-postserver
- Ditt e-postprogram kan hämta brev från servern, till exempel med POP

Peer-to-peer, P2P

Alla datorer som deltar i ett P2P-nätverk är likvärdiga, Varje dator fungerar både som server och klient.

- Ingen central server som är ”flaskhals” i nätet; sparar bandbredd
- ”Hybridsystem” kan ha en server med till exempel en lista över filer och var filerna finns
- Exempel: Napster, Kazaa, BitTorrent, LimeWire, . . .

Adressering

- Varje dator i ett lokalt nätverk har en egen adress som är inbränd på nätverkskortet, en MAC-adress (48 bitar).
- Varje dator som är ansluten till Internet har en IP-adress (32 bitar). Exempel: `130.235.35.177`. IPv6: ny variant av IP, 128-bitsadresser.
- En router skickar utgående datapaket till rätt destination.
- IP-adresser är svåra att läsa och komma ihåg (för människor). Datorer har också symboliska adresser, till exempel `hacke-1.student.lth.se`
- En namnserver (DNS, Domain Name Server) översätter från symbolisk adress till IP-adress.

WWW på Internet

- WWW: World Wide Web, ”nätet”.
- Startade 1989, forskare på CERN (Schweiz) behövde utbyta information (dokument).
- Utnyttjar *hypertext* — dokument innehåller länkar till andra dokument. Dokument skrivs i HTML (HyperText Markup Language).
- Webbservrar tillhandahåller dokument.
- Webbläsare hämtar dokument från webbservrar och visar dem på datorn.
- Dokument identifieras med webbadresser, URL-er (Uniform Resource Locator).
- `http://www.w3.org/History/1989/proposal.html`

HTML

HTML är ”vanlig text”.

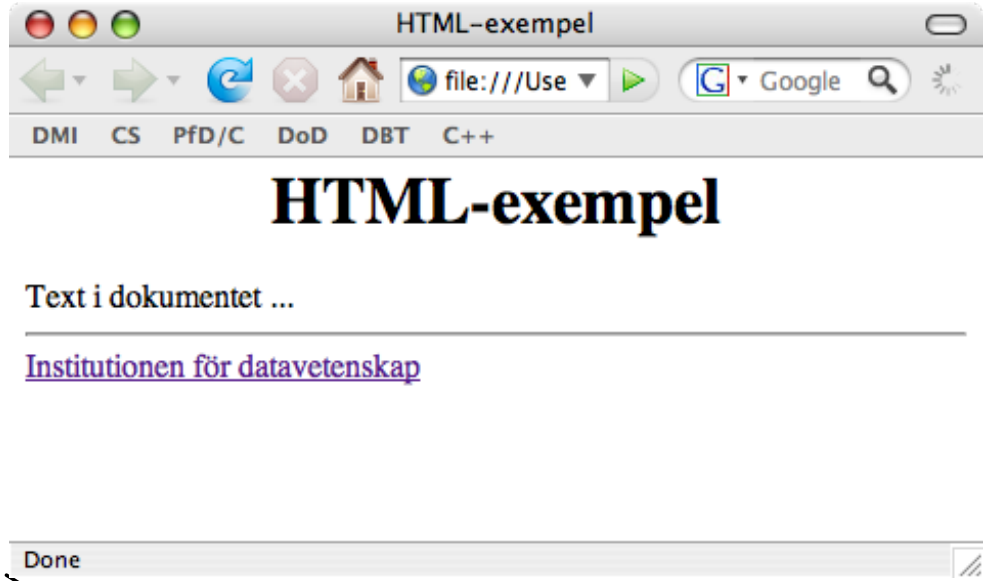
- Med ”taggar” anger man att texten ska formateras på olika sätt och lägger in länkar till andra dokument
- Starttaggar och sluttaggar: `<h1>Rubrik på nivå 1</h1>`
- Det finns också taggar för att länka till andra dokument, till bilder, osv:

```
<a href=http://www.cs.lth.se/>
Institutionen för datavetenskap</a>
```

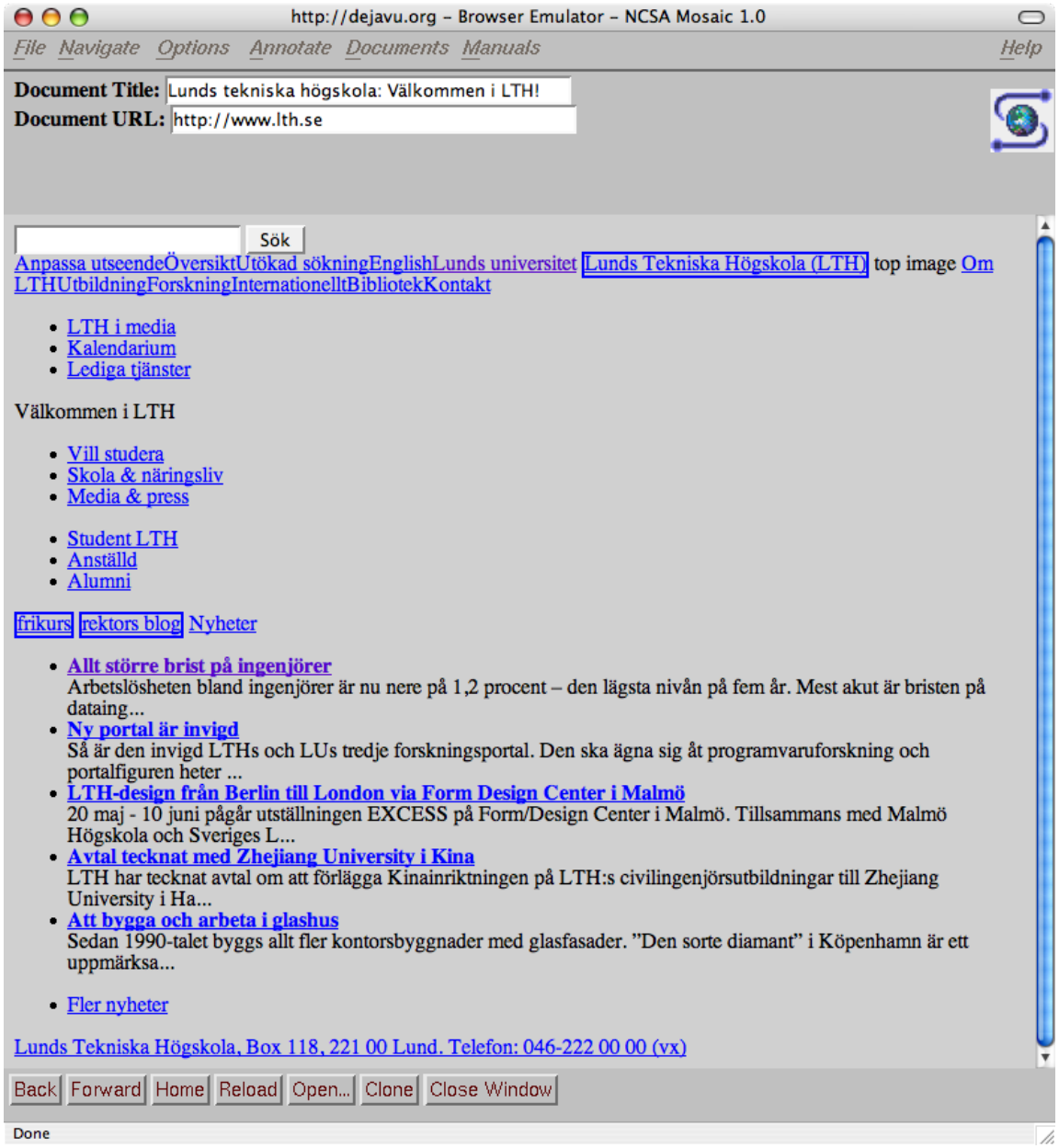
- Det finns massor av HTML-kurser på nätet

HTML-exempel

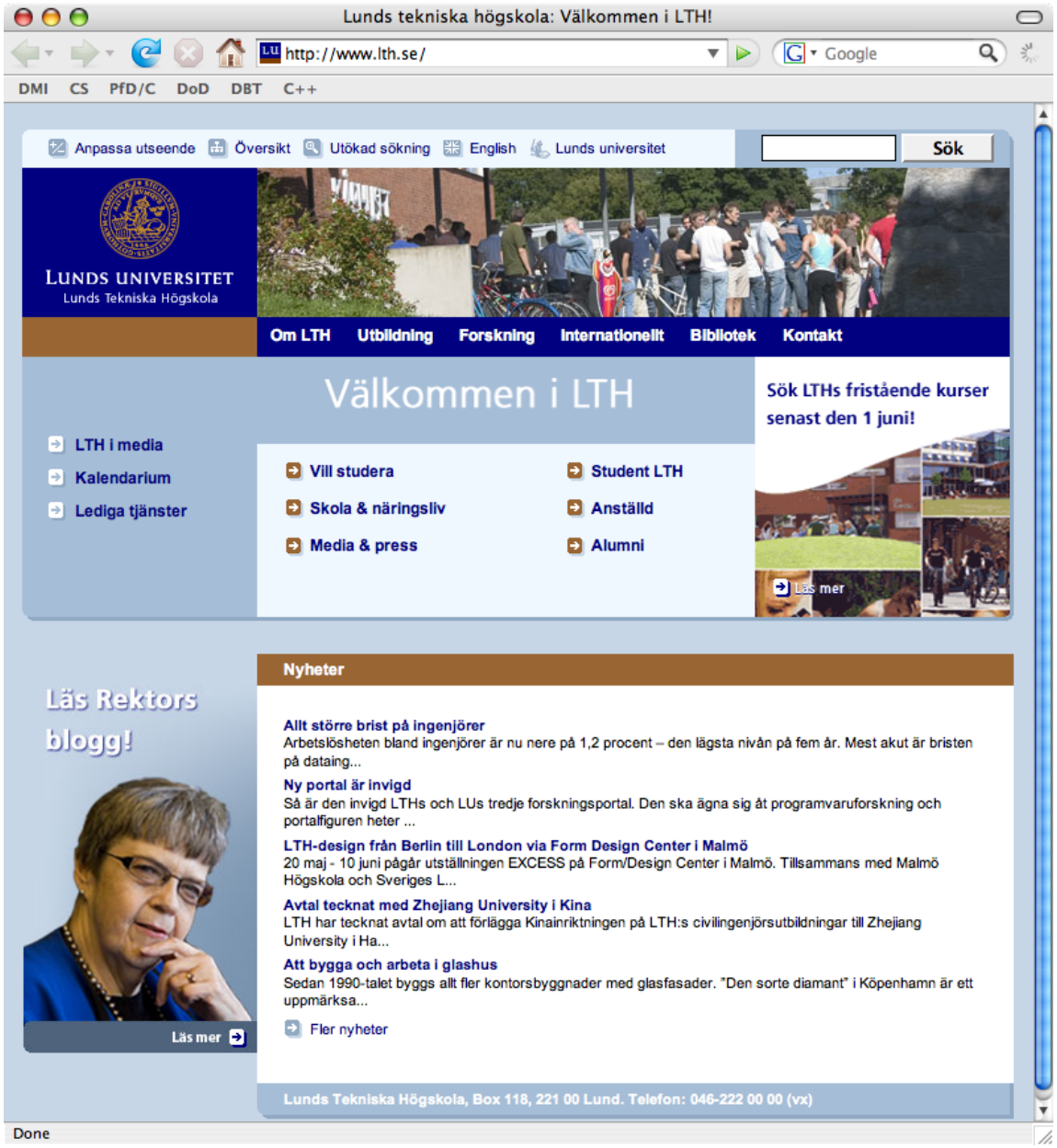
```
<html>
<head>
<title>HTML-exempel</title>
</head>

<body>
<h1><center>
   HTML-exempel
</center></h1>

Text i dokumentet ...
<hr>
<a href="http://www.cs.lth.se/">
Institutionen för datavetenskap</a>
</body>
</html>
```



Webbläsare: Mosaic, 1993



Webbläsare: Firefox, 2007



Vad händer när man surfar?

- Starta Firefox (eller Opera, eller Internet Explorer, eller . . .).
- Skriv en URL i adressfältet:
  `http://www.w3.org/History/1989/proposal.html`
- Webbläsaren kopplar upp sig mot webbservern på `www.w3.org` och skickar följande kommando:

```
GET /History/1989/proposal.html
```

- Webbservern hittar filen och skickar tillbaka den till webbläsaren.
- Webbläsaren läser HTML-koden, tolkar kommandona (taggarna) och visar filen i fönstret.
- Samma saker händer om man inte själv skriver en URL utan klickar på en länk.

Dynamiska webbsidor

- En ”vanlig” webbsida är statisk, dvs den har alltid samma utseende. Webbservern behöver bara då hitta HTML-filen och skicka tillbaka den till klienten.
- Men det behövs också dynamiska webbsidor som ser olika ut för varje uppkoppling. Till exempel Google: ett sökprogram körs på servern när klienten begär det; en HTML-sida med resultatet genereras och skickas till klienten.
- Parametrar skickas till serverdatorn, efter URL-en:

```
http://www.google.se/search?q=dynamiska+webbsidor
```

- Tekniker för att köra program på servern: CGI (program i valfritt språk), PHP (PHP: Hypertext Preprocessor), ASP (Active Server Pages, Microsoft), JSP (Java Server Pages).

Föreläsning 6 — Operativsystem

- Operativsystemets uppgifter
- Historik
- Skydd: in- och utmatning, minne, CPU
- Processer, tidsdelning
- Sidindelat minne, virtuellt minne
- Filsystem

Operativsystemets uppgifter

Operativsystemet hanterar:

- exekvering av program
- minne
- filsystem
- in- och utmatning
- skydd och felhantering
- kommunikation med användaren

Operativsystem, historik

Inget OS	• program i maskinspråk • datorn körs av användaren • ladda remsläsare med programmet, läs in, tryck på start, programmet kör, utskrift
Batch	• program läggs i kö, körs ett i taget
Multikörning	• flera program samtidigt i minnet, turas om att exekvera
Interaktivt	• användaren har kontrollen

Skydd, allmänt

Bland de viktigaste uppgifterna för ett operativsystem är skydd av olika slag. Ett program får inte påverka operativsystemet eller andra program:

- In- och utmatning — ett program får inte läsa/skriva var och hur som helst
- Minne — ett program får bara använda sitt eget minnesutrymme
- CPU — ett program får inte lägga beslag på CPU-n

Skydd — in- och utmatning

En del uppgifter i datorn får bara utföras av operativsystemet — in/utmatning, hantera avbrott, sätt klockan, stäng av datorn, ...

Dessa operationer är *privilegierade* och får bara utföras när datorn befinner sig i systemtillstånd. När ett ”vanligt” program körs befinner sig datorn i användartillstånd. Om datorn är i system- eller användartillstånd bestäms av en bit i hårdvaran.

Avbrott (till exempel från yttre enhet eller klocka) medför byte till systemtillstånd.

Användaren kommer åt de privilegierade operationerna genom programmerade avbrott, ”systemanrop”.

In- och utmatning, detaljer

När ett program ska läsa/skriva från ett skivminne måste andra program få möjlighet att köra under tiden läsningen/skrivningen görs (den tar lång tid):

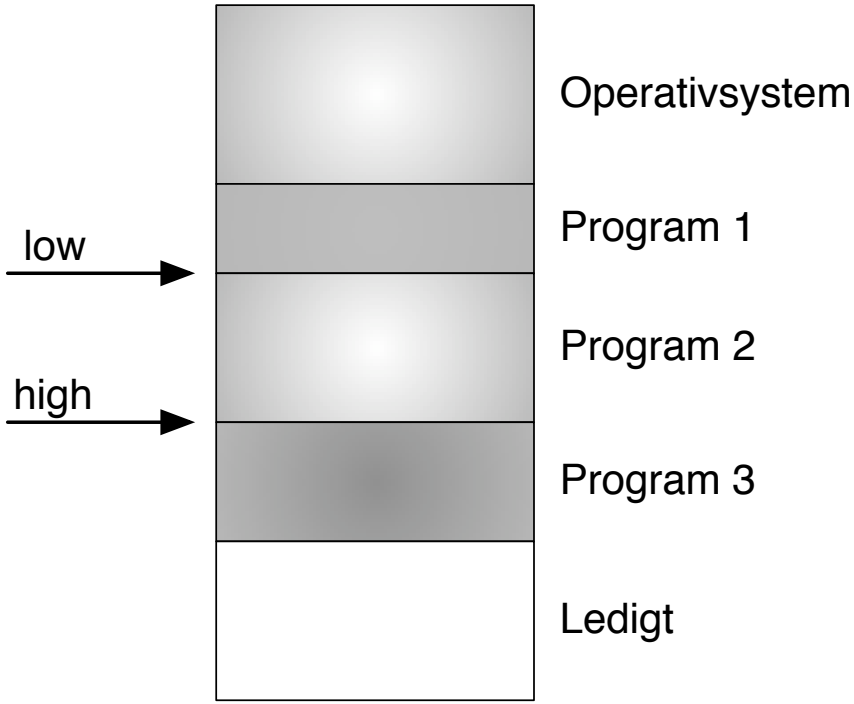
- begär läsning/skrivning
- vänta på att läsningen/skrivningen blir klar (här får andra program exekvera)
- fortsätt

Det kan inträffa att flera program vill läsa/skriva innan föregående in/utmatning är klar. Då läggs önskningarna i en kö och operativsystemet tar hand om dem i tur och ordning.

Skydd — minne

Ett program får bara referera till sitt eget minnesutrymme (inte operativsystemets eller andra programs minne).

Varje minnesreferens måste kontrolleras (kan göras med två register i hårdvara, undre och övre gräns; mera komplicerat vid virtuellt minne):



Översättning: `fysisk-adress = logisk-adress + low`

Skydd — CPU

Ett program får inte lägga beslag på CPU-n för egen del.

En *klocka* avbryter programexekveringen med jämna mellanrum, så att operativsystemet kan kontrollera CPU-tidsförbrukningen.

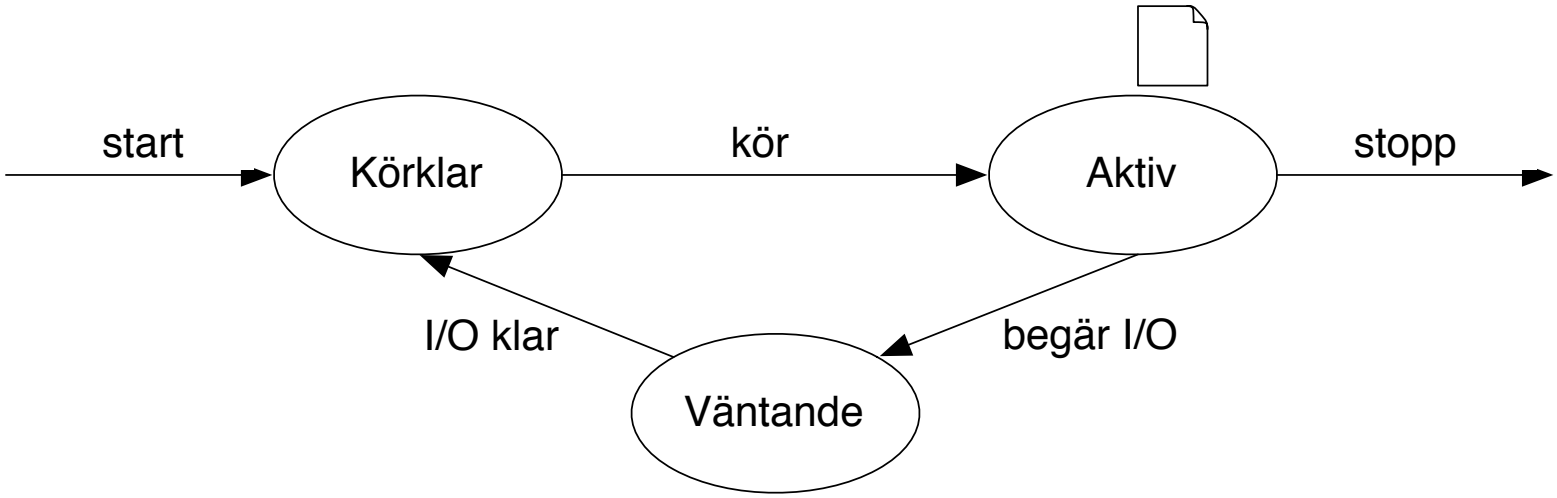
CPU-tidsfördelning

Flera program (processer) kan finnas i minnet samtidigt. En process kan befinna sig i olika tillstånd:

- aktiv (har tillgång till CPU-n)
- körklar (vill ha tillgång till CPU-n)
- väntande (väntar på in/utmatning eller något annat)

Operativsystemet flyttar processen mellan tillstånden.

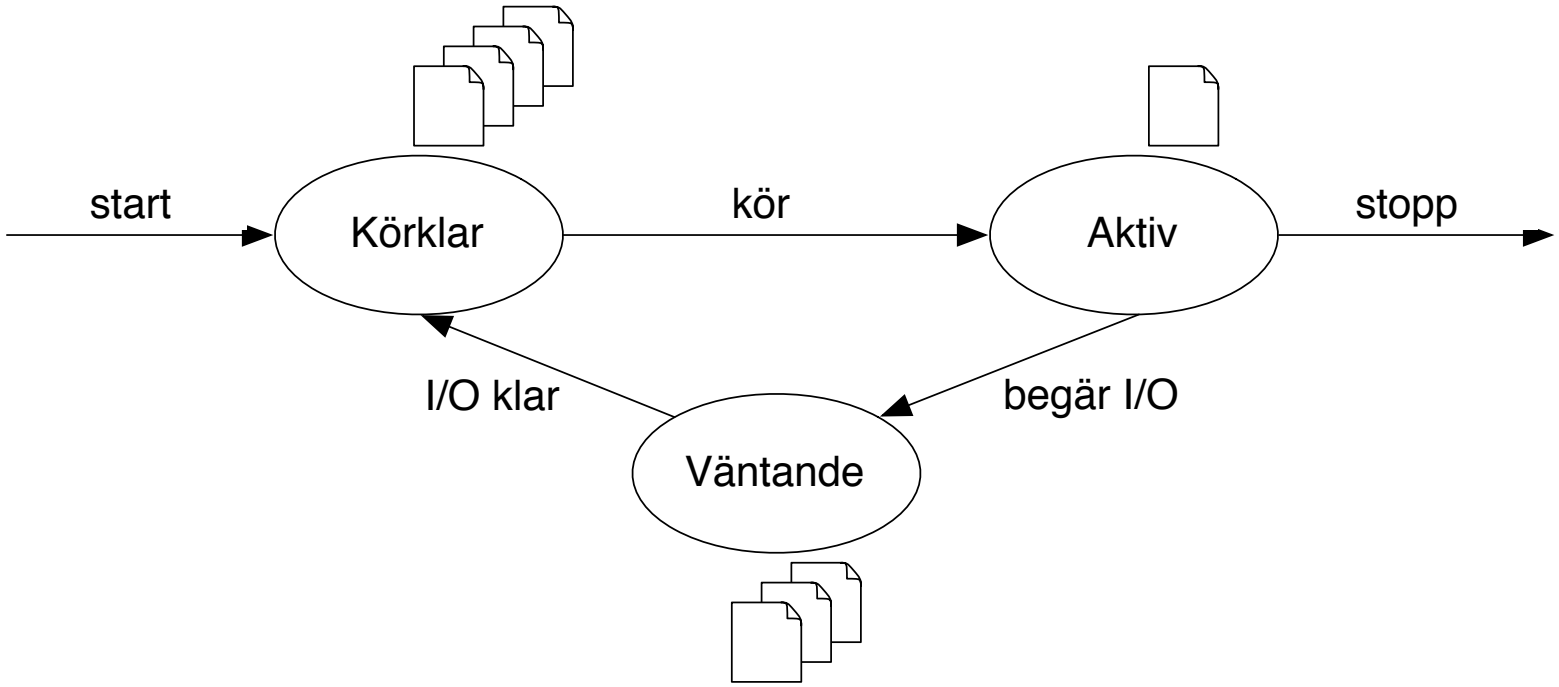
En ensam process



Om det bara finns en process i datorn:

- Körklar, blir omedelbart aktiv
- Är aktiv tills den själv begär I/O (Input/Output)
- Väntar tills I/O är klar
- Körklar, omedelbart aktiv
- Osv, tills processen avslutas

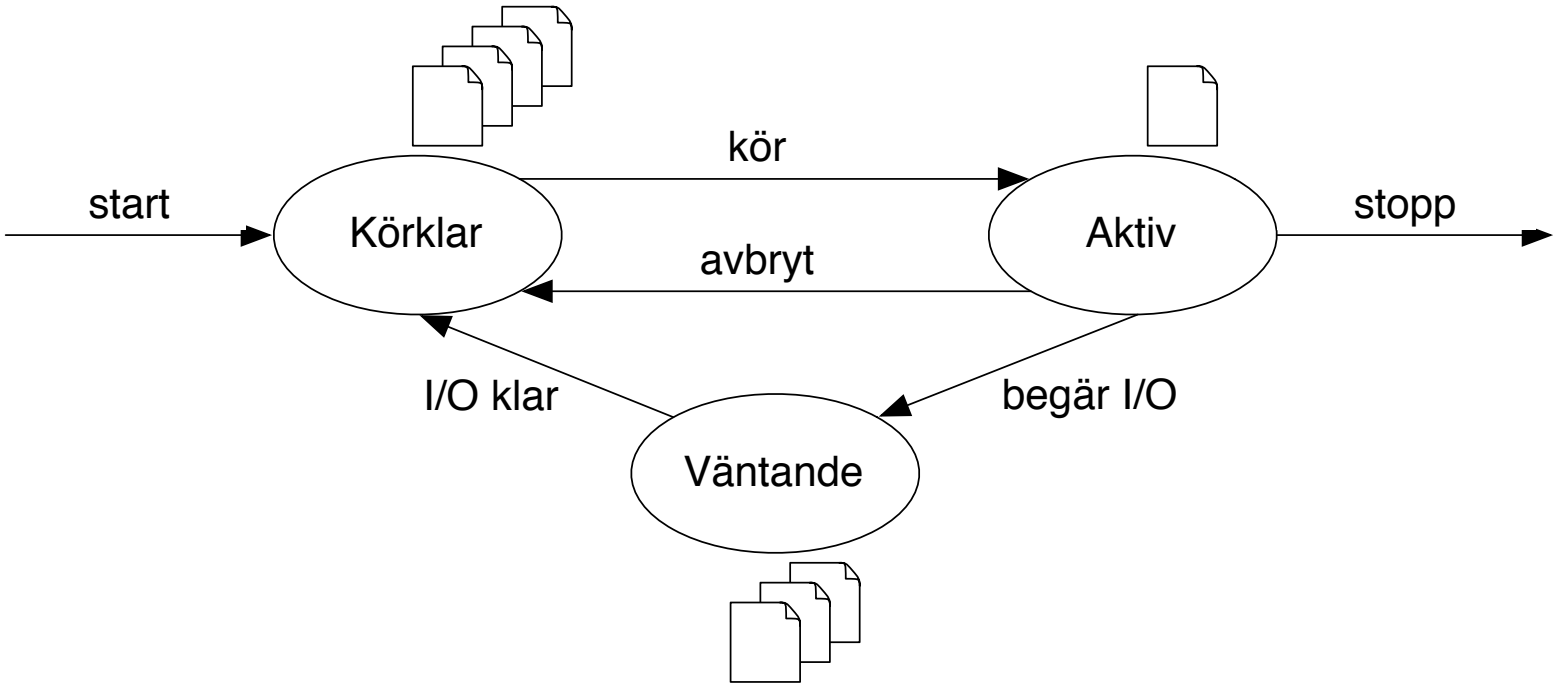
Flera processer



Det fungerar på liknande sätt om det finns flera processer. Men nu får en annan körklar process exekvera när den aktiva processen blir väntande.

Vi har alltså möjlighet till multikörning. Men observera att en aktiv process kan lägga beslag på CPU-n.

Tidsdelning



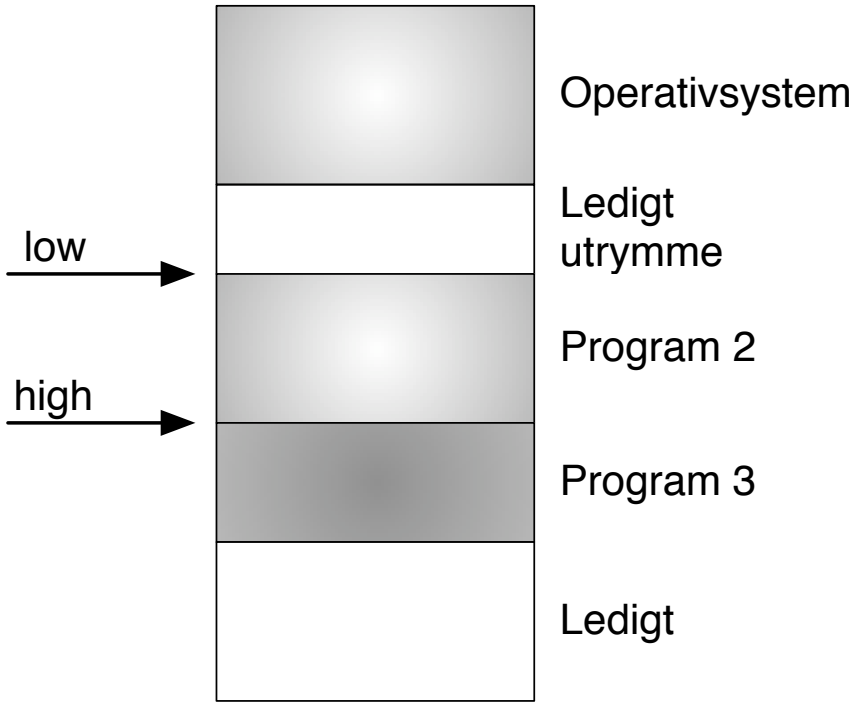
Om det finns flera processer och man vill se till att en process inte lägger beslag på CPU-n så måste CPU-tiden delas mellan processerna.

Tiden delas ut i *tidskvanta* (10–100 ms). Efter varje kvantum avbryter operativsystemet den aktiva processen och ser till att nästa körklara process blir aktiv. Processer har normalt prioritet som bestämmer turordningen.

Minneshantering

Enklaste modellen: programmet får minne i en följd ”där det finns plats”. Minnet skyddas med low-high-register.

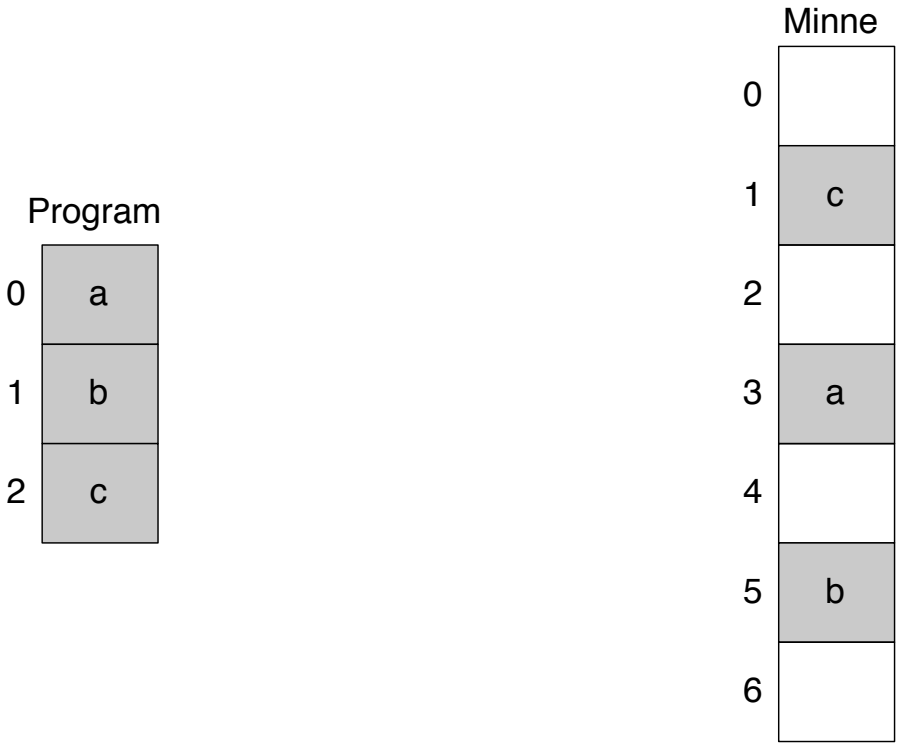
Problem: det blir ”hål” i minnet som kanske inte kan utnyttjas (fragmentering).



Sidindelat minne

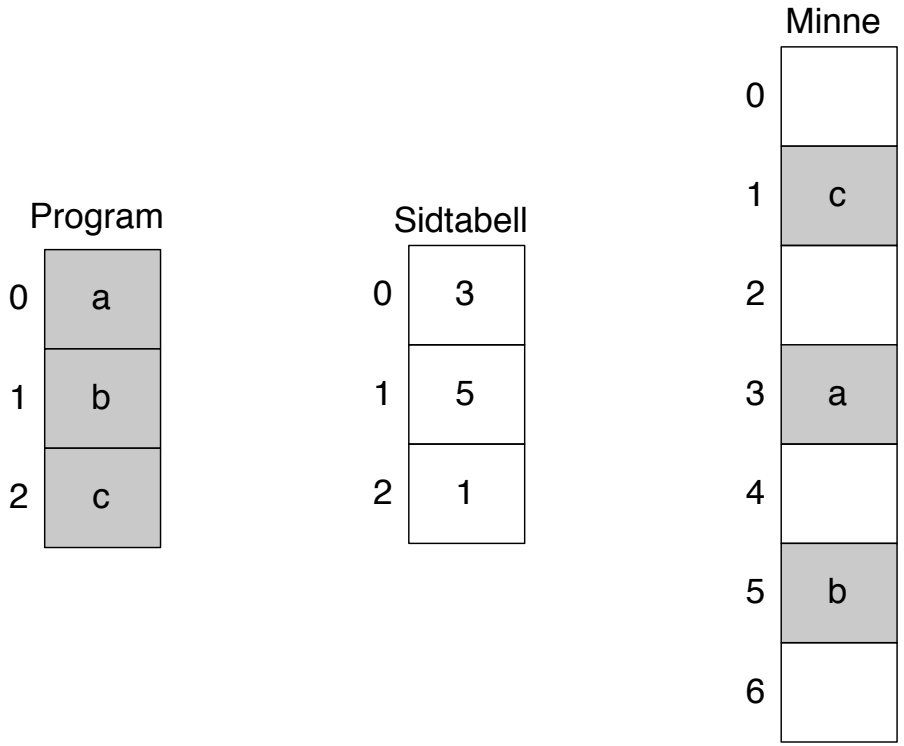
Undvik fragmentering genom att dela upp minnet i *sidramar* (några kB stora). Eftersom alla sidramar är lika stora är det lätt att hålla reda på vilka sidramar som är lediga.

Dela upp programmet i *sidor*, lika stora som sidramarna. Lägg programmets sidor i lediga sidramar, som inte behöver komma i följd. Nu blir det enkelt att stoppa in fler program i minnet.



Sidtabeller

Men tänk nu att programmet vill komma åt innehållet i minnescellen med adressen 733 (i programmet står det till exempel `move r1,733`). Antag att sidornas storlek är 1 kB. Då finns sidan som innehåller denna minnescell i det fysiska minnet på adressen 3773 (sida 0 ligger i sidram nr 3). Varje minnesadress måste översättas från logisk till fysisk adress. Detta görs med hjälp av en *sidtabell*.



Virtuellt minne

Om det fysiska minnet är mindre än det logiska minnet använder man *virtuellt minne*. Det är ett slags sidindelat minne där sidorna normalt finns på skivminne. Inte förrän programmet refererar till en sida laddas den in i primärminnet.

Om minnet blir fullt måste sidor i minnet skrivas ut på skivminne, så att sidramar blir lediga. Och då ska man helst skriva ut sidor som inte kommer att behövas under den närmaste tiden.

Allt detta hanteras av operativsystemet.

Filsystem

Användaren vill se ett skivminne som ett antal filer samlade i kataloger. Man ska kunna komma åt en fil om man känner filens namn.

Operativsystemet ser skivminnet som en följd av block, numrerade 0 och uppåt. (På den allra lägsta nivån måste man också hålla reda på spår och sektorer på skivan.)

Skivminnet måste organiseras så att översättningen filnamn $\rightarrow$ block blir möjlig. Detta är filsystemets uppgift.

Föreläsning 7 — Matlab

Förberedelse inför laboration 5.

- Matlab, Maple, Mathematica
- Flyttal
- Matlab som miniräknare
- Vektorer
- Grafik
- Funktioner
- Matriser, ekvationssystem

Matematisk programvara

Datorer kan räkna, och det finns program som hjälper till:

MATLAB ger numeriska lösningar (”lösningar med siffror”)

Maple ger symboliska lösningar (”lösningar med bokstäver och matematiska symboler”) men kan också räkna med siffror

Mathematica liknar Maple

Matlab används mycket i undervisningen (högre kurser i matematik, fysik, statistik, numerisk analys, reglerteknik, . . .).

I numerisk analys (D3) lär man sig teorin bakom de numeriska metoderna som används i Matlab.

Påpekande: för att man ska veta vilken metod man ska använda och för att man ska kunna tolka resultaten från programmen krävs kunskaper i matematik och numerisk analys!

Matlab

Matlab = MATrix LABoratory

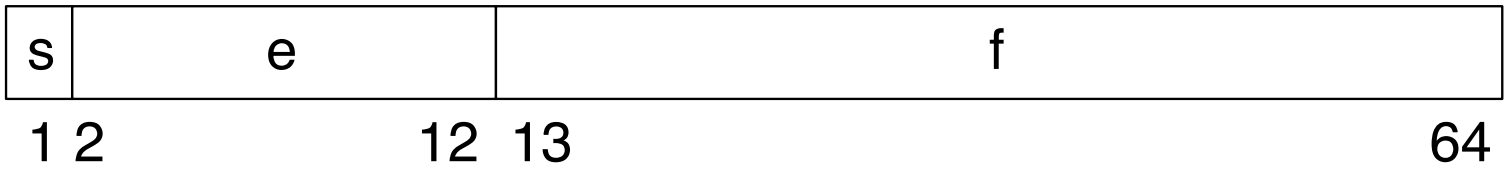
- Ursprungligen ett användargränssnitt till ett paket med standardrutiner för linjär algebra.
- Sedan 1984 kommersiell produkt, Mathworks, `http://www.mathworks.com`
- Finns för Windows, Linux, Unix, Macintosh.
- Dyrt att köpa, men Lunds Universitet har avtal med Mathworks så att studenter har gratis tillgång till Matlab.
- Du kan via ddg-webbsidorna ladda hem Matlab till din egen dator, `http://program.ddg.lth.se`
- Octave, `http://www.octave.org`, är ett "open source"-alternativ till Matlab (nästan identiska kommandon)

Användning av Matlab

- Interaktiv, interpreterande miljö: man skriver kommandon som omedelbart utförs.
- Men det finns också ett inbyggt programspråk och man kan spara sina program i "`m`-filer".
- Matlab består av en kärna med färdigskrivna rutiner skrivna i C och Fortran och en massa färdigskrivna `m`-filer. Det finns också tilläggspaket ("verktygslådor") för speciella tillämpningsområden.

Flyttal (double)

Matlab utnyttjar dubbel precisionstal för beräkningar (samma som `double` i Java, IEEE-standard). 64 bitar fördelas på följande sätt:



- talvärdet $= \pm(f + 1) \cdot 2^e$. Max $\approx 10^{308}$, min $\approx 10^{-308}$
- s är teckenbiten
- $f = d_1 \cdot 2^{-1} + d_2 \cdot 2^{-2} + \ldots + d_{52} \cdot 2^{-52}$
- $-1022 < e < 1023$

När man räknar med flyttal har man en begränsad noggrannhet (ungefär 16 decimala siffror) och avrundningsfel kan inträffa. Det är inte säkert att "normala" matematiska lagar gäller — till exempel kan det gälla att $a + b - c \neq a - c + b$ (beroende på storleken hos a, b, c). Detta studeras i numerisk analys.

Matlab som miniräknare

>> är Matlabs prompt, ans är en variabel som innehåller det senaste beräknade värdet (”answer”):

```
>> 3 + 2 - 1
ans =   4
>> 2^6 - 1  % ^ står för ’’upphöjt till’’
ans =   63

>> cos(pi / 4)  % matematiska funktioner finns inbyggda
ans =   0.7071

>> format long  % skriv ut i långt format, återgå med
>> cos(pi / 4)  % format short
ans =   0.70710678118655
```

Variabler

Allmänt:

- variabler behöver inte deklareras, typen bestäms av den första tilldelningen som görs
- små/stora bokstäver är olika

```
>> x = 2
x = 2
>> y = x / 25; % semikolon sist: resultatet skrivs inte ut
>> result = sin(y * pi)
result =    0.2487

>> who          % ger lista med alla variabler
Your variables are:
ans result  x   y
```

Spara och ladda

```
>> save temp % spara arbetsarean i filen temp.mat
>> clear     % radera arbetsarean (ta bort alla variabler)
>> who
Your variables are:
>> quit
```

I en senare körning:

```
>> load temp % återställ arbetsarean från temp.mat
>> who
Your variables are:
ans result  x   y
```

Vektorer

En vektor är en samling av element som alla har samma typ. Varje element har ett nummer, ett *index*.

En vektor skapas genom att man skriver elementen inom `[]`:

```
>> x = [ 5 4 12 ]
x = 5    4    12
>> x(1)              % index från 1, runda parenteser
ans =   5

>> y = 1 : 5         % vektor från 1 till 5
y = 1    2    3    4    5

>> z = 1 : 0.5 : 5 % 1 till 5 i steg om 0.5
z = 1    1.5    2    2.5    3    3.5    4    4.5    5
```

Räkna med vektorer

Man kan räkna med vektorer (addera, subtrahera, . . .). Vi nöjer oss med att utföra operationer element för element i vektorerna.

För att göra detta skriver vi + och - på vanligt sätt. *, / och ^ måste skrivas med en inledande punkt: .* ./ .^ (Om man glömmer punkten betyder operatorerna något annat.)

```
>> x = [ 1 2 3 4 5 ];
>> y = [ 1 1 2 2 3 ];

>> x + y
ans =
    2 3 5 6 8

>> x .* y
ans =
    1 2 6 8 15
```

Mera vektorräkning

Mera om vektorer (och matriser) kommer senare. I kursen Linjär algebra kommer ni att lära er mycket om radvektorer, kolonnvektorer och matriser. Då kan ni studera vektor- och matrisräkning i Matlab i detalj.

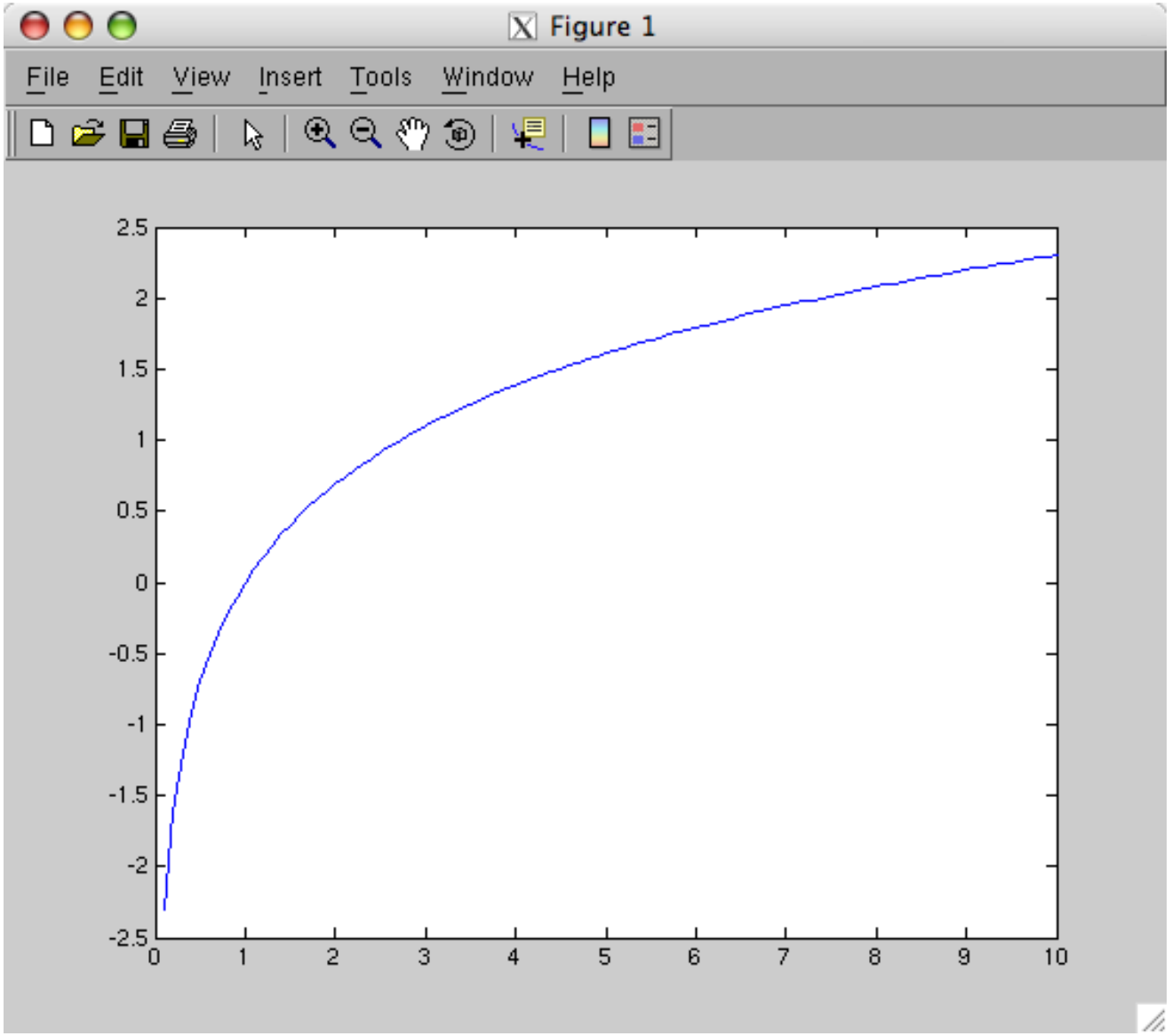
Om argumentet till en funktion är en vektor så appliceras funktionen på alla vektorelementen och resultatet är en ny vektor:

```
>> x = 1 : 5;
>> roots = sqrt(x)
roots =
    1.0000    1.4142    1.7321    2.0000    2.2361
```

Grafik

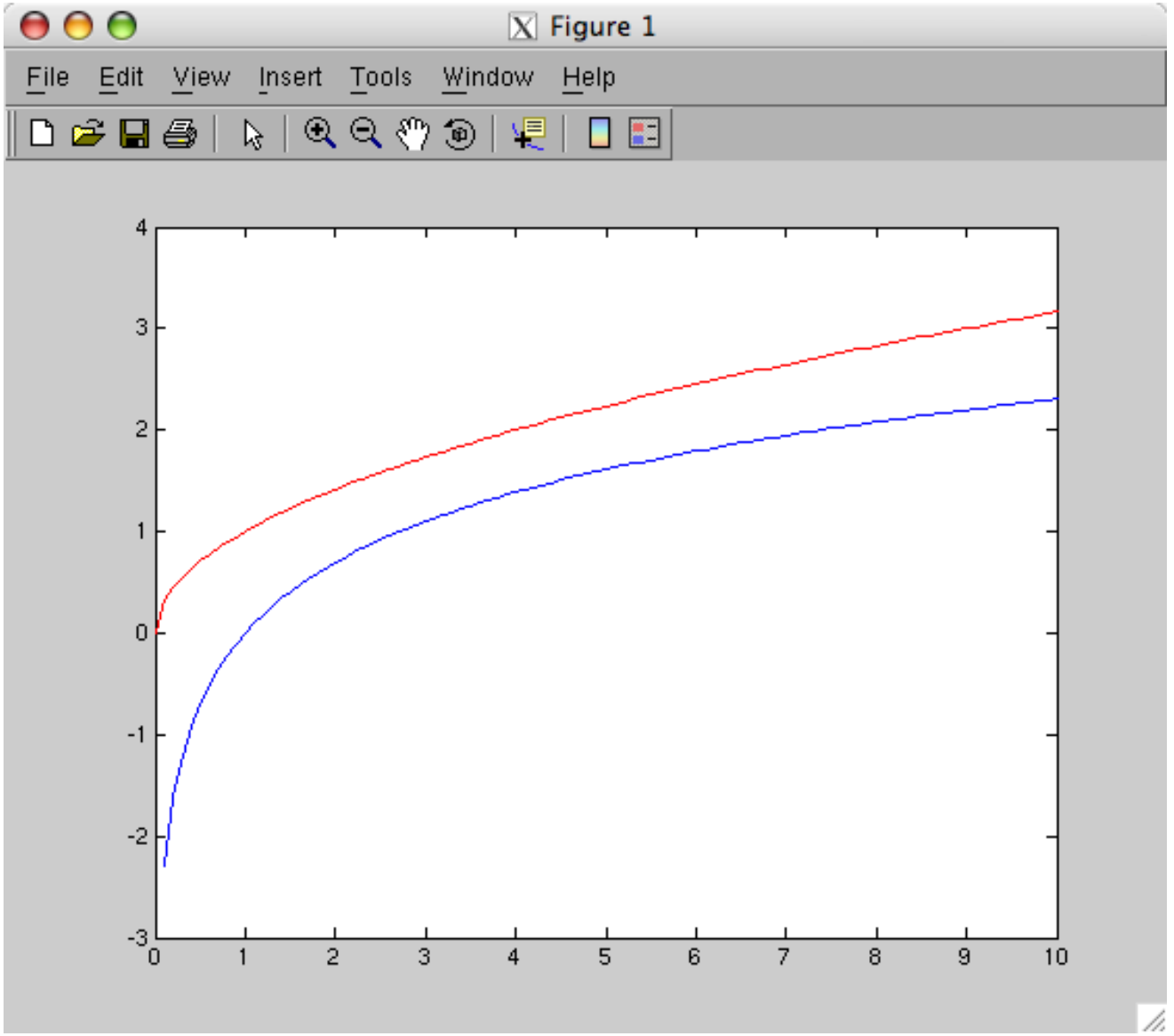
Plotta en kurva med ett antal x- och y-värden:

```
>> x = 0 : 0.1 : 10;
>> y = log(x);
>> plot(x, y)
```



Mera grafik

```
>> clf % radera
>> x = 0 : 0.1 : 10;
>> y = log(x);
>> plot(x, y)
>> hold on % rita nästa
% kurva i samma fönster
>> plot(x, sqrt(x), 'r')
% 'r' är färgen (red)
% flera alternativ finns,
% se help plot
```



Funktioner

En funktionsfil är en Matlab-fil (`m`-fil) som innehåller en funktion.
En funktion i Matlab har följande uppbyggnad:

```
function resultat = funktionsnamn(inparametrar)
% en eller flera kommentarrader som skrivs ut när
% man gör "help funktionsnamn"
... ett antal satser där resultatet tilldelas värde
```

Exempel:

```
function res = sqr(x)
% RES = SQR(X), compute the square of X
res = x * x % funktionsresultatet
```

Exempel på funktion

```
function avg = medel(x)
% AVG = MEDEL(X), compute average of elements in X
sum = 0;
for k = 1 : length(x)
    sum = sum + x(k);
end
avg = sum / length(x); % funktionsresultatet
```

Funktionen sparas i filen `medel.m` och anropas:

```
>> x = [ 8 2 7 7 34 ];
>> medel(x)
ans =   11.6000
```

(Funktionen `mean`, som är inbyggd i Matlab, ger samma resultat. `medel` är alltså en onödig funktion.)

Kontrollstrukturer

I Matlab finns `if`-, `for`- och `while`-satser. Man kan använda dessa som vanliga kommandon som skrivs vid tangentbordet, men normalt förekommer de i funktioner. `if`-sats:

```
if Villkor
    satser
else
    satser
end
```

Jämfört med Java:

- ingen parentes runt villkoret
- relationsoperatorer: som i Java, men `~=` i stället för `!=`
- `&` betyder och, `|` betyder eller (inte `&&` `||`)
- inga `{}` för att hålla ihop satser, `end` sist i stället
- `elseif` för `else if`

for och while

```
for variabel = vektor
    satser
end
```

Exempel:

```
for i = 1 : 10 ...
for i = 1 : 2 : max ...
for i = [1 4 9] ...
```

```
while Villkor
    satser
end
```

Funktioner som parametrar

Funktioner kan ha andra funktioner som parametrar. Vi visar bara exempel på anrop av sådana funktioner.

```
>> fplot('log', [0 10]) % plotta logaritmfunktionen i intervallet
                        % 0..10. Observera apostroferna runt
                        % funktionsnamnet
```

Det går bra med egna funktioner också, till exempel `myfunc`:

```
function f = myfunc(x)
% F = MYFUNC(X), compute a test function
f = (1 - exp(-x .^ 2)) ./ (1 + x .^ 2) - 0.1;
>> fplot('myfunc', [-5 5])
```

Man kan lösa ekvationer, men man ska inte lita blint på resultatet:

```
>> fzero('myfunc', 0) % finn nollställe nära 0
ans =   -0.3444
```

Numeriska metoder

De flesta matematiska problem går inte att lösa ...

Åtminstone kan man inte lösa dem exakt. I stället är man hänvisad till någon numerisk metod som beräknar lösningen med önskad noggrannhet. Det finns numeriska metoder för:

- Ekvationslösning
- Lösning av ekvationssystem
- Approximation (”kurvanpassning”)
- Integration
- Lösning av differentialekvationer
- Linjär algebra
- och mycket mera

Matriser

En matris är ett rektangulärt ”rutnät”. Nedanstående matris är kvadratisk och har 3 rader och 3 kolonner.

$$A = \begin{pmatrix} 1 & 5 & 6 \\ 2 & 4 & -1 \\ 9 & 8 & 7 \end{pmatrix}$$

Matlab är bra på att hantera matriser. Exempel:

```
>> a = [1 5 6; 2 4 -1; 9 8 7]
     1   5   6
a =  2   4  -1
     9   8   7
```

En kolonnvektor är en vektor där talen står ”ovanpå” varandra:

```
>> b = [1; 3; 5]
     1
b =  3
     5
```

Ekvationssystem med matriser

Lös följande ekvationssystem:

$$\begin{aligned} x + 5y + 6z &= 1 \\ 2x + 4y - z &= 3 \\ 9x + 8y + 7z &= 5 \end{aligned}$$

Det här kan man uttrycka med matriser. Man kan multiplicera matriser med varandra och multiplicera en matris med en kolonnvektor:

$$\begin{pmatrix} 1 & 5 & 6 \\ 2 & 4 & -1 \\ 9 & 8 & 7 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x + 5y + 6z \\ 2x + 4y - z \\ 9x + 8y + 7z \end{pmatrix}$$

Ekvationssystemet blir (b är kolonnvektorn `[1; 3; 5]`):

$$A * x = b$$

Ekvationssystem i Matlab

Lösningen till $A * x = b$ skulle bli $x = b/A$. I Matlab går det bra att "dividera" en matris med en vektor. Exempel:

```
>> a = [1 5 6; 2 4 -1; 9 8 7];
>> b = [1; 3; 5];
>> a \ b
        0.3518
ans =   0.4975
       -0.3065
```

Anmärkning: ekvationssystem är viktiga ... Det finns tillämpningar där det dyker upp ekvationssystem med tusentals obekanta.