Operativsystem

I/O I

(kap 5 i boken)

Input /Output

- En av de viktigaste delarna av ett OS är att sköta Input / Output
  - Hårdskiva / olika skivstationer / Flash
  - Nätverk (Ethernet / WLAN / Bluetooth)
  - USB / Serieportar
  - Infraröda portar
  - Tangentbord / mus
  - Skärm

Principer för I/O hårdvara

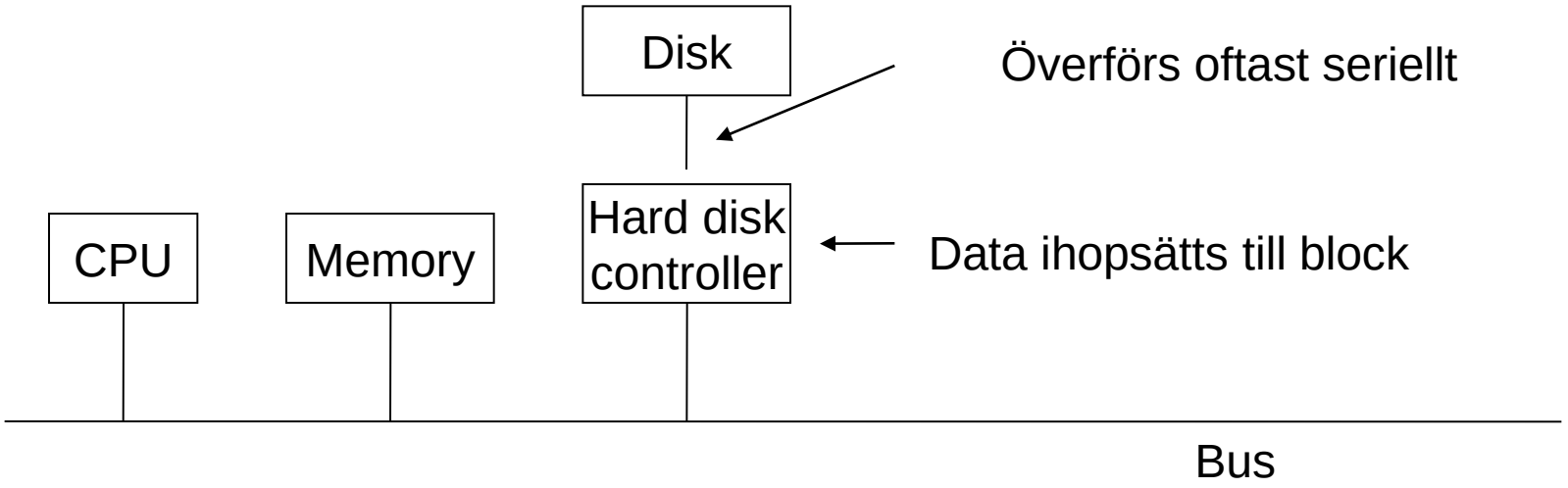
- Elektronikperspektiv
  - chips, ledare, strömkällor, motorer
  - De fysiska komponenter som bygger upp hårdvaran
- Mjukvaruperspektiv
  - Kommandon som hårdvaran förstår
  - Fuktioner som hårdvaran utför
  - Fel som rapporteras tillbaka

I/O-enheter - klassificering

- Block-baserade enheter
  - Använder fixerade storlekar på block, typiskt 512-32768 bytes
  - Det är möjligt att läsa eller skriva vilket som helst av blocken oberoende av de andra
  - T.ex. hårdskivor, CR-rom
- Teckenbaserade enheter
  - Producerar eller mottar strömmar av tecken
  - Kan inte adresseras, ingen sök-operation
  - Printrar, nätverksgränssnit, möss, etc..

Styrning av enheter

- I/O enheter
  - Mekanisk komponent
  - Elektronisk komponent = styrning av enhet / adapter



I/O arkitektur

- Buss – en samling av datastigar som sammanbinder CPU, enheter, minne
- Typer t.ex.:
  - ISA, EISA, PCI, MCA
  - Tre specialiserade bussar
    - Databus (Pentium 64-bit-wide)
    - Adressbus (Pentium 32-bit-wide)
    - Kontrollbus
      - Bus för kontrolllinjer (t.ex. Read-linje / write-linje / val processor-RAM el. processor-I/O)
  - En buss som sammanbinder CPU med I/O-enheter kallas I/O-buss

I/O portar

- Varje enheter som kopplas till I/O-bussen har sin egen mängd av I/O-adresser = I/O-portar
- I386: 65.536 8-bits I/O-portar
  - Kan dock användas som 8-bit, 16-bit el. 32-bit
- I386: assembler instruktioner
  in, ins, out, outs
- Portarna indelas i “register”
  - Kontrollregister
  - Statusregister
  - Inputregister
  - Outputregister

I/O portar - exempel

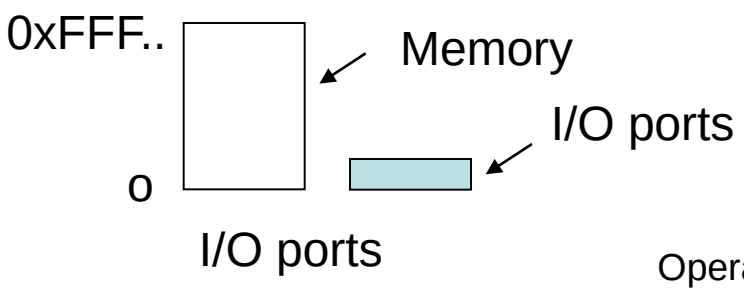
```
[jbjorkqv@borken ~]$ more /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
0376-0376 : ide1
```

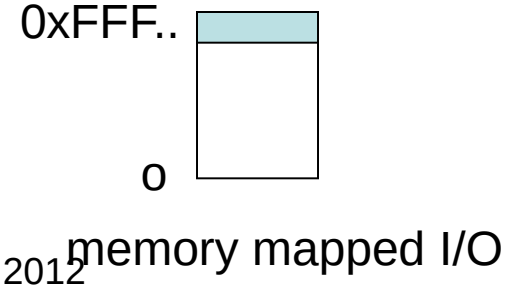
I/O-port access från user mode (Linux)

- Rättigheter
  - `ioperm(unsigned long from, unsigned long num, int turn_on)`
    - access of ports *`from`* to *`from+num`*
    - ports up to 0x3ff
  - `iopl(int level)`
    - `level=3 access all I/O ports`
- Accessering via macron (assembler):
  - `inb(), inw(), inl(), outb(), outw(), outl()`

Minnesmappad I/O

- Generellt två sätt att accessera I/O-register
  - Direkta I/O-portar (IN reg, port)
  - minnesmappad I/O
- Minnesmappad I/O
  - Portarna mappas in i den fysiska adressrymden i datorn
  - I/O-portarna kan accesseras som en normal variabel i adressrymden





Minnesmappad eller direkt?

- Accesserbarhet
  - Direkt: Kräver assemblerinstruktioner
  - MM: kan accesseras som en variabel
- Skydd
  - Direkt: Kräver extra skyddsmekanismer
  - MM: Samma som för minne i allmänhet
- Cache
  - MM: problem om cachar I/O-minne

I/O delat minne

- Hårdvaruenheter har ofta eget minne (t.ex. grafikkort)
- I/O-enheters minne mappas in i den normala (fysiska) adressrymden
  - ISA: 0xa0000 to 0xfffff (640 kB to 1 MB)
  - VESA Local Bus: 0xe00000 0xffffff (14-16 MB)
  - PCI-bus: → Mapps in i ”mycket hög fysisk adress”

I/O delat minne: Exempel

```
[jbjorkqv@borken ~]$ more /proc/iomem
00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-0fffcfff : System RAM
  00100000-0027d173 : Kernel code
  0027d174-0031fbdf : Kernel data
0fffd000-0fffefff : ACPI Tables
0ffff000-0fffffff : ACPI Non-volatile Storage
df800000-df80007f : 3Com Corporation 3c905B 100BaseTX [Cyclone]
e0000000-e1dfffff : PCI Bus #01
  e0000000-e0ffffff : nVidia Corporation RIVA TNT2 Model 64
e1f00000-e3ffffff : PCI Bus #01
  e2000000-e3ffffff : nVidia Corporation RIVA TNT2 Model 64
e4000000-e7ffffff : Intel Corp. 440BX/ZX/DX - 82443BX/ZX/DX Host
   bridge
ffff0000-ffffffff : reserved
```

Direct Memory Access (DMA)

Disk read, no DMA (Programmed Input / Output PIO)

1. CPU initiates a read by writing to device registers
2. Controller reads the block into controllers internal buffer, verifies the block
3. Controller triggers a interrupt
4. OS takes over
5. In a loop, the block is read byte by byte / word by word from controller device register and stored in main memory

Disk read, DMA

1. CPU programs the DMA controller (what to transfer and where)
2. DMA controller initiates the read by writing to device registers
3. Controller reads and verifies the block
4. The DMA controller transfers the block from controller buffer to main memory
5. DMA controller triggers an interrupt, telling the CPU the transfer is complete

I/O-enhetsfiler (Unix)

- I Unix behandlas I/O-enheter logiskt såsom filer
- Samma systemanrop används för att accessera enheter som för att accessera normala filer
- Ex. I Linux
  - Traditionella I/O-enhetsfiler
    - major number 1-254 device type, minor numer
    - mknod() creates the files
  - devfs device files
    - drivers register by name rather than by major/minor numbers (e.g. */dev/discs/disc1*)
    - Virtual file system, device files exist only after registration (*devfs_register()*)

Device files

- Traditional

```
brw-rw----    1 root     disk       3,   0 Apr 11  2002 /dev/hda
```

driver code

- devfs

```
brw-------    1 root     root       3,   0 Jan  1 01:00
/dev/ide/host0/bus0/target0/lun0/disc
```

- However, note the same device major and minor numbers

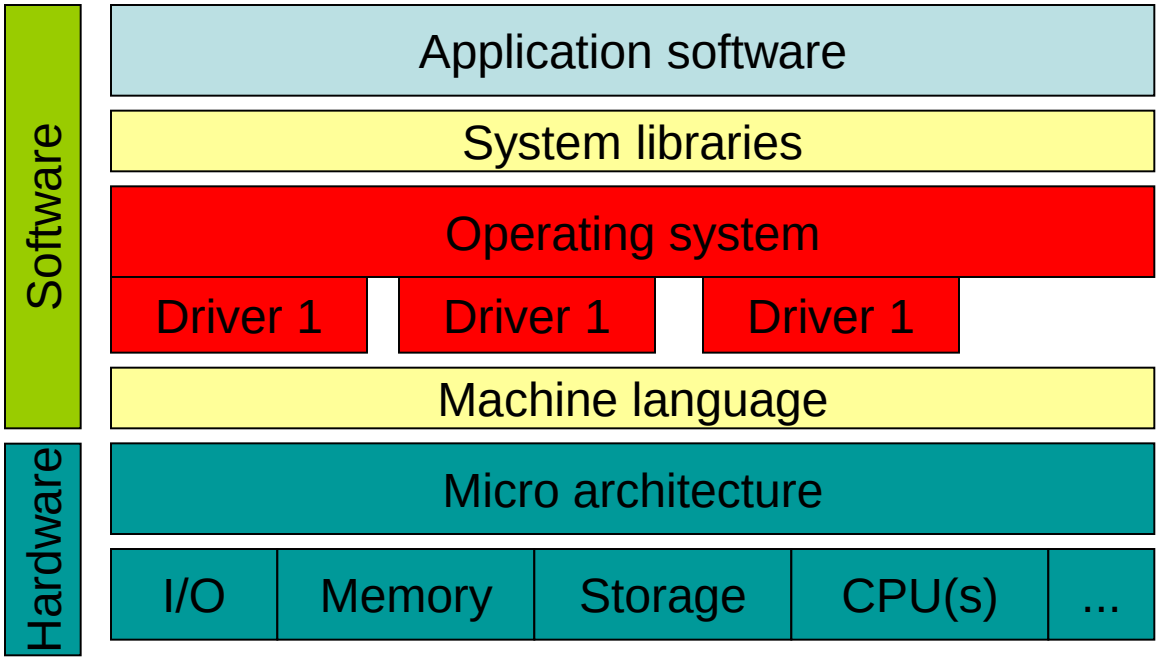
Drivrutiner för I/O-enheter

- OS uppgifter med avseende på I/O
  - Erbjuda full access till egenskaperna hos hårdvaruenheter, men med olika nivå av stöd i själva OS
    - Inget stöd alls – applikationsprogrammet använder direkt enheters I/O-portar
    - Minimalt stöd – kernelen känner igen enheten, men endast dess I/O-gränssnitt. Användarprogram måste själva känna till t.ex. vilka kommandon enheter förstår
    - Utvecklat stöd – kernelen känner igen enheter och utför alla nödvändig I/O till enheten
  - Måste koordinera hårdvaruresurser
  - Måste skydda enheter från användarprogram

Drivrutiner

- Antingen del av kernelen
- Eller laddas som moduler vid behov
- Erbjuder ett standardgränssnitt som enheter kan accesseras genom
- Vet vilka magiska bitar som skall placeras i vilka magiska positioner för att få enheten att fungera på önskat sätt

Ett datorsystem



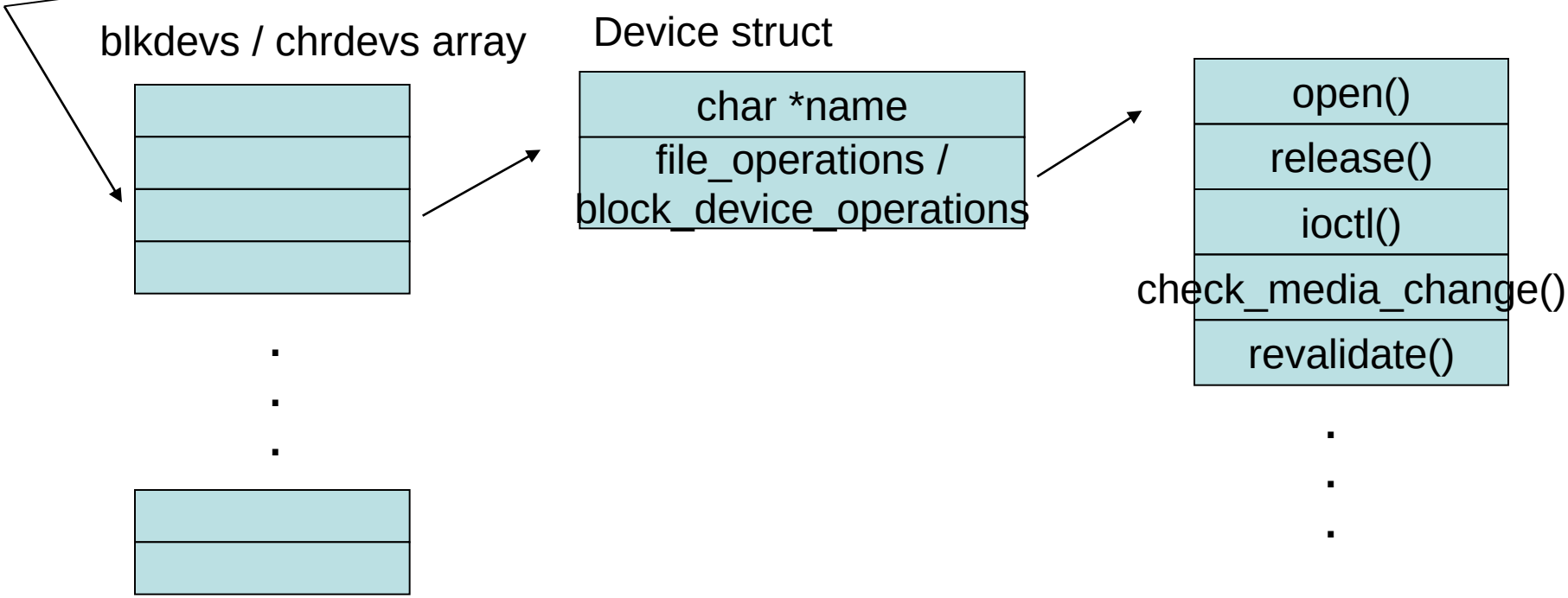
```
[jbjorkqv@borken ~]$ more
/proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 14 sound
 29 fb
116 alsa
128 ptm
136 pts
180 usb
226 drm
254 pcmcia

Block devices:
  2 fd
  3 ide0
  8 sd
  9 md
 22 ide1
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
```

(Char)Drivrutiner i linux

- Device is registered by giving the kernel the address to a `file_operations` structure
- Structure `file_operations` – some functions:
  - `poll()` – checks if activity
  - `mmap()` – maps the file into address space
  - `open()` – creates a file object
  - `close()` – close the file object
  - `flush()` – flushes the file
  - `ioctl()` – send command to hardware
  - `fsync()` – writes cached data
  - `read()` – read bytes from file
  - `write()` – writes bytes to file

Accessering av enhet

```
brw-rw----    1 root     disk       3,   0 Apr 11  2002 /dev/hda
```



Initialisering av enhet

- T.ex. `register_chrdev(6, ”lp”, &lp_fops)`
- Registrera så snabbt som möjligt
  - Tillgänglig för användarprocesser
- Initialisera så sent som möjligt
  - Allokerar värdefulla resurser
  - Normal en användnings- `counter`
    - Inkrementeras vid open()
    - Dekrementeras vid release()
    - Resurser allokeras vid första open()
    - Resurser deallokeras vid sista release()

Uppföljning av I/O operatiner

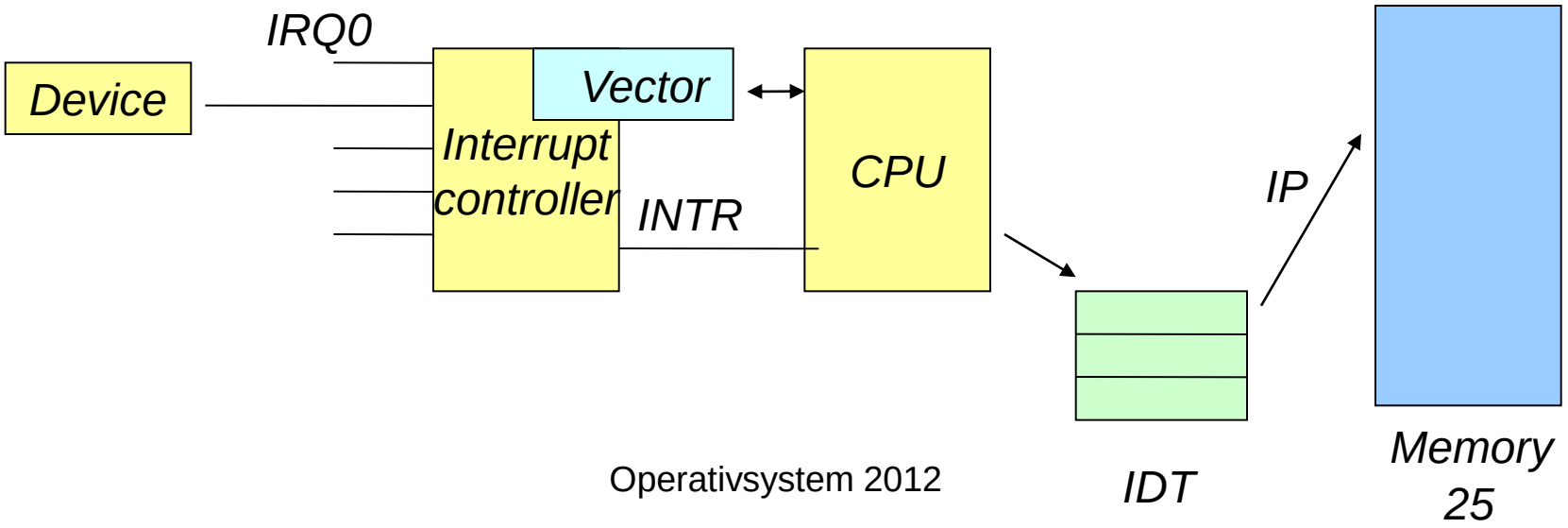
- Polling mode
  - CPU:n väntar i en loop till enheten utfört jobbet

```
for (;;) {
        if (read_status(device) & DEVICE_END_OPERATION) break;
        if (--count == 0) break;
}
```

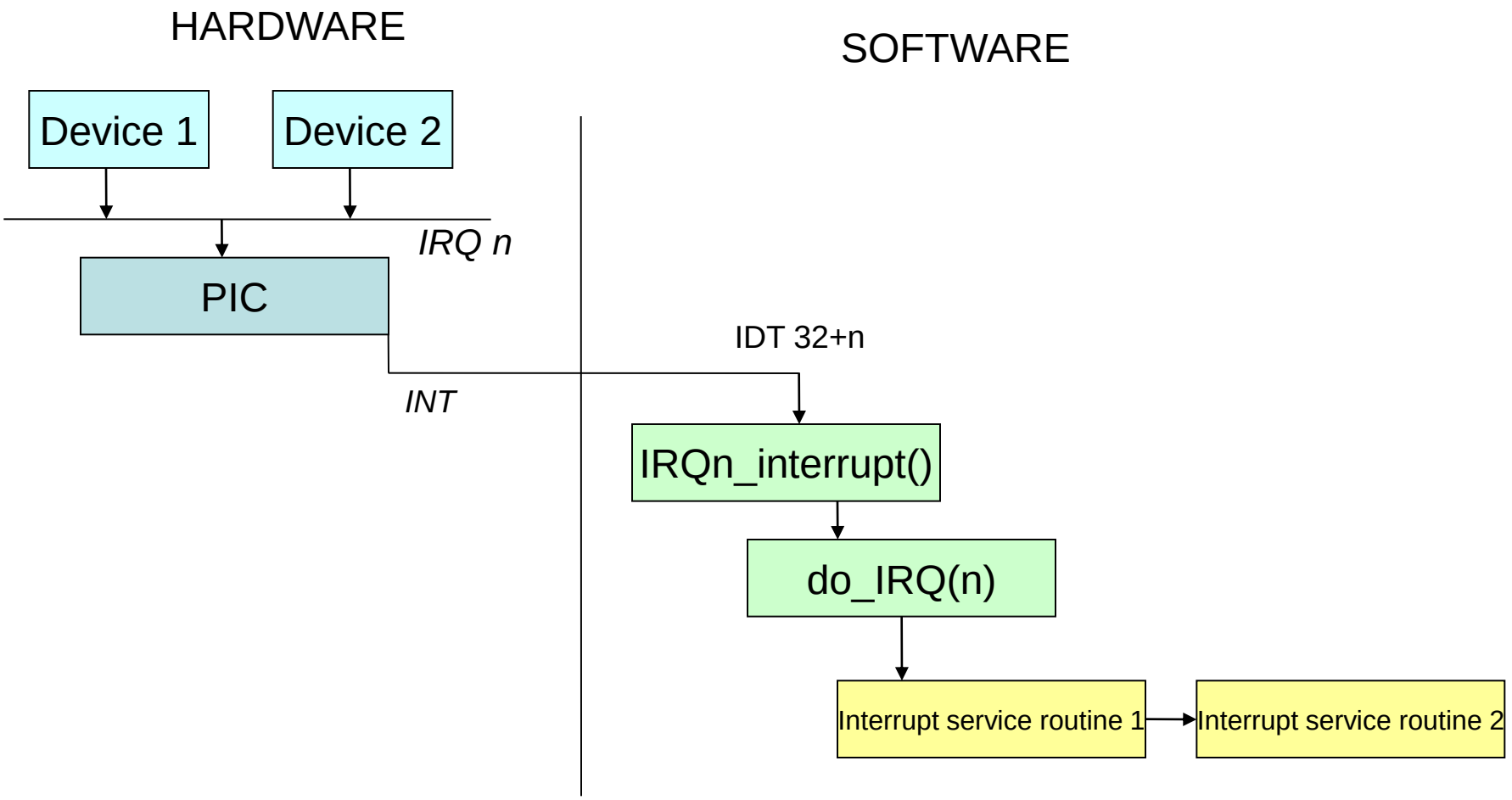
- Avbrotts-baserad funktionalitet
  - Jobbet börjar vänta
    `interruptible_sleep_on()`
  - Enheter skapar ett avbrott då den har utfört sitt jobb
  - En avbrottsservicerutin utför sina operationer och meddelar den som begärde servicen, som nu kan fortsätta

Repetition: avbrott

- Avbrottsdeskriptortabeller associerar varje avbrott eller exception med adressen till motsvarande avbrottshanterare



I/O avbrottshantering



Exampel – läsning från dev “foo”

```
ssize_t foo_read(struct file *filp, char *buf, size_t
   count, logg_t *ppos) {
    foo_dev_t * foo_dev = filp->private_data;
    if (down_interruptible(&foo_dev->sem)
        return -ERESTARTSYS;
    foo_dev->intr=0;
    outb(DEV_FOO_READ, DEV_FOO_CONTROL_PORT);
    wait_event_interruptible(foo_dev->wait, (foo_dev-
      >intr==1));
    if (put_user(doo_dev->data, buf))
        return -EFAULT;
    up(&foo_dev->sem);
    return 1;
}
```

Dev foo interrupt service routine

```
void foo_interrupt(int irq, void *dev_id, struct
   pt_regs *regs) {
    foo->data = inb(DEV_FOO_DATA_PORT);
    foo->intr = 1;
    wake_up_interruptible(&foo->wait);
}
```

Hårdvara som ej reagerar

- Om hårdvaran aldrig generar det avbrott som OS väntar på, måste OS kunna reagera på situationen
- Vanligen sätts en timer innan hårdvaran får börja jobba
- Om timern utlöper innan ett avbrott registreras, så kommer nu istället timerfunktionen att “reda upp” saken

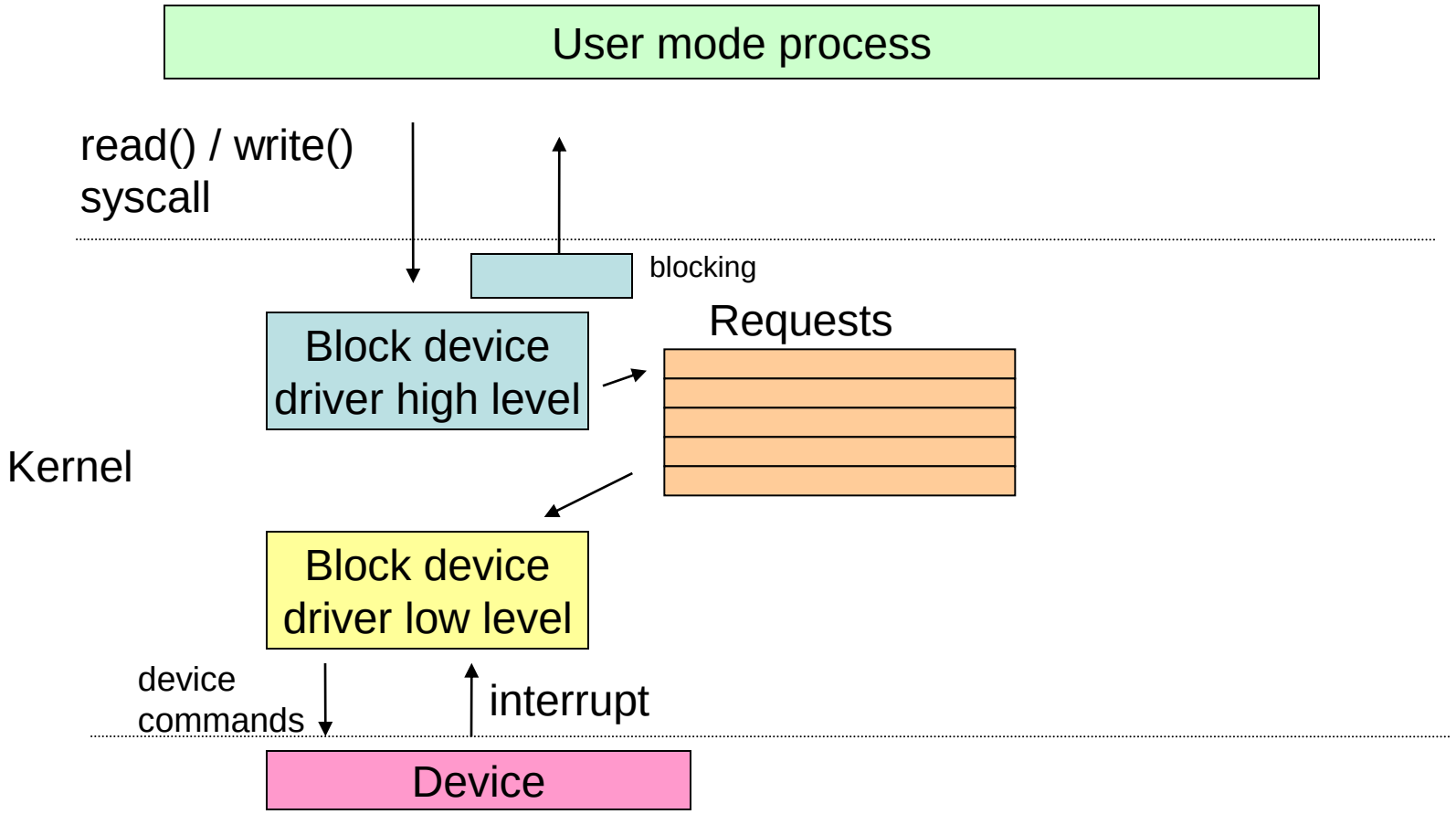
Problem med avbrott

- I normala CPU:n med pipeline och superscalarity, blir avbrottshantering något problematisk
- När kan man avbryta det som man håller på med? Vilken är nästa instruktion man skall utföra? Hur fortsätta efter ett avbrott?
- Införs en ”precise interrupt”
  - 1. PC är sparad på en känd plats
  - 2. Alla instruktionen för PC är fullt exekverade
  - 3. Ingen instruktion efter PC har exekverats
  - 4. Exekveringsstatus för instruktionen som PC pekar på är känd

Drivrutiner för blockenheter

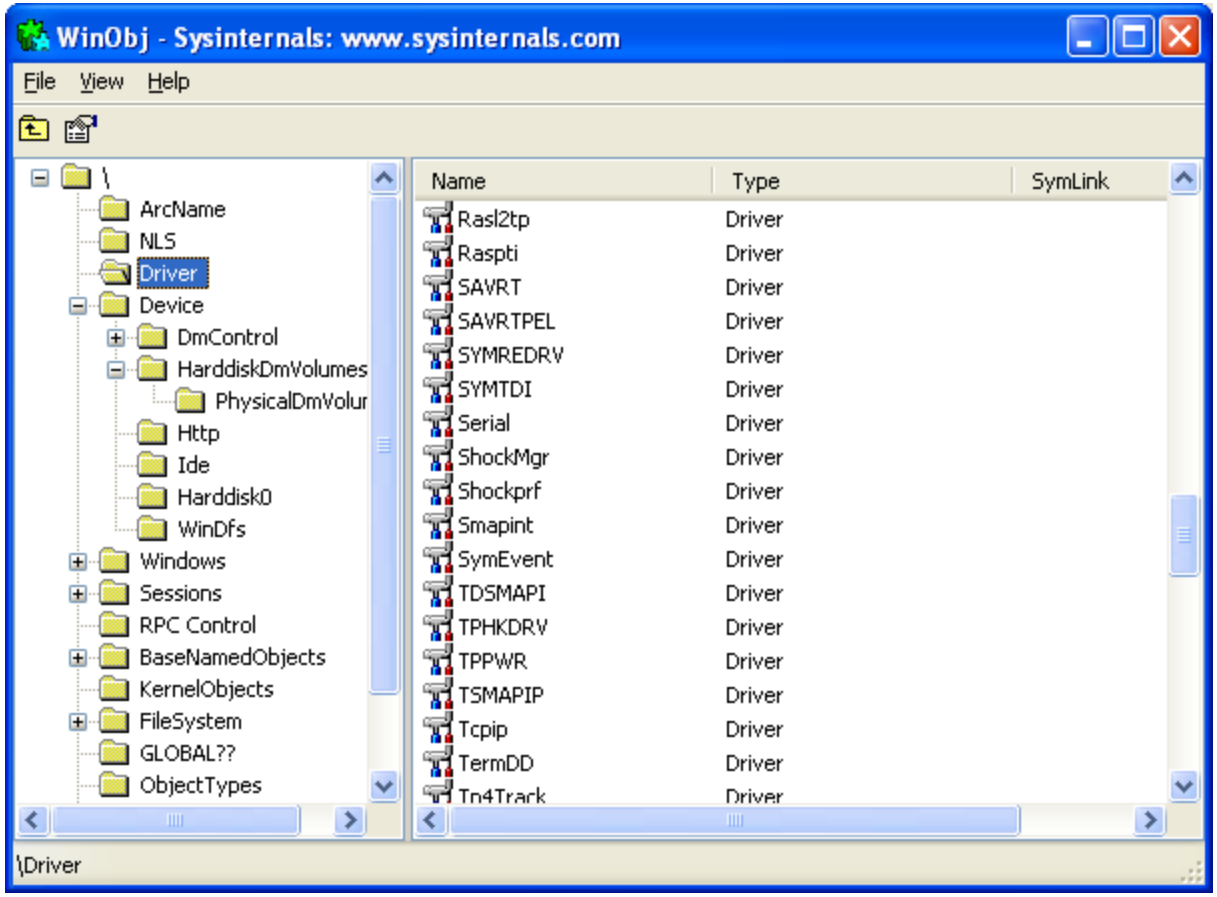
- Varje läs/skriv-begäran översätts till en *block device request*
- Denna förfrågan sätts på kö och kommer att utföras vid ett senare tillfälle
  - Den process som väntar på att begäran utförs blockeras
  - Men, själva drivrutinen skall inte blockeras i väntan på att data finns tillgänglig
    - En kernel exekveringsstig skall aldrig blockeras i väntan på data
- Arkitektur för avbrottsdrivrutin
  - Ett avbrott från enheten kommer att informera om begäran är utförd, en ny begäran skeduleras nu på enheten

Drivrutinarkitektur för blockenheter

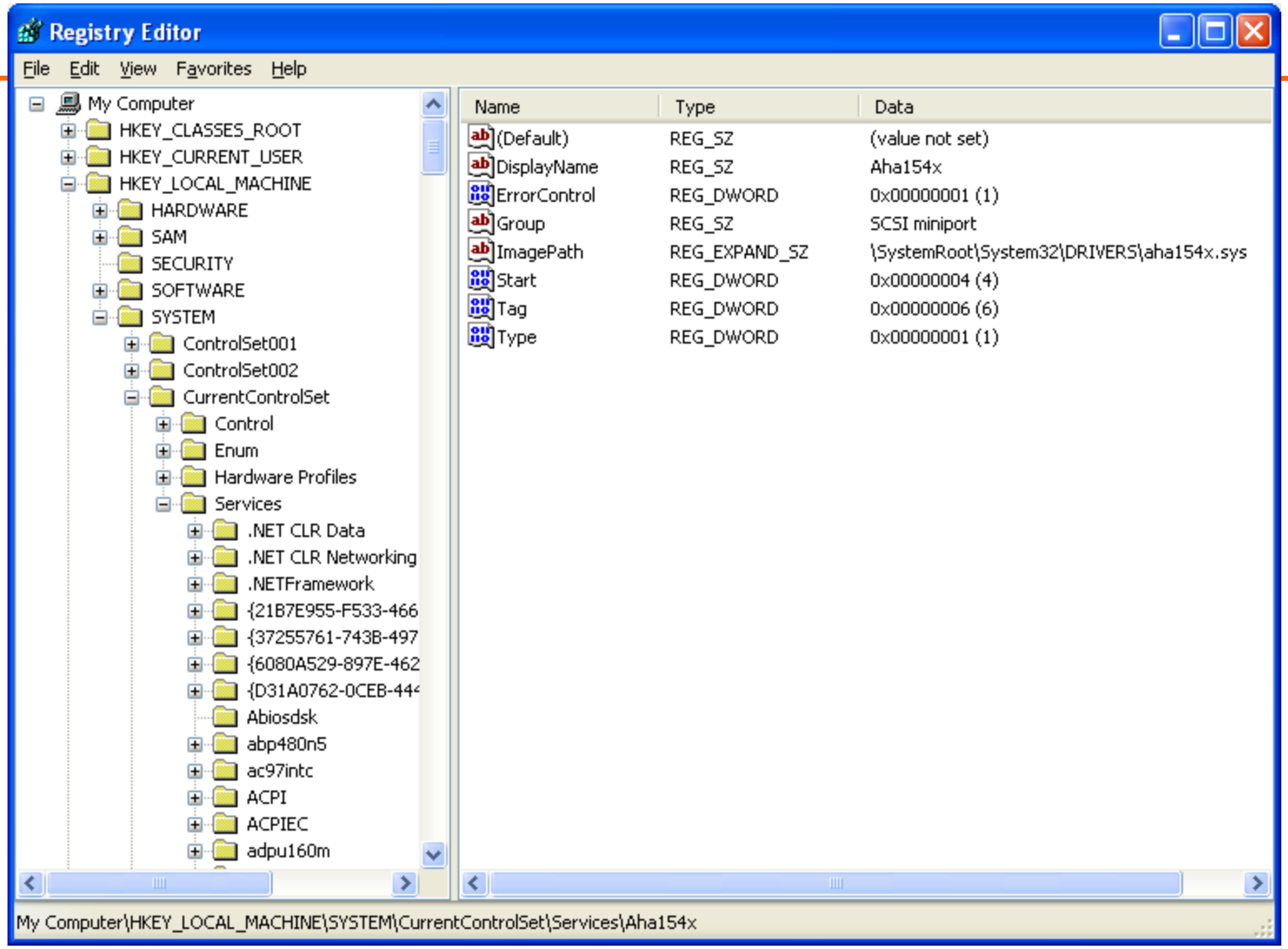


Drivrutiner i NT

Postfixet: .SYS, finns i winnt\system32\drivers\
Registreras i HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services



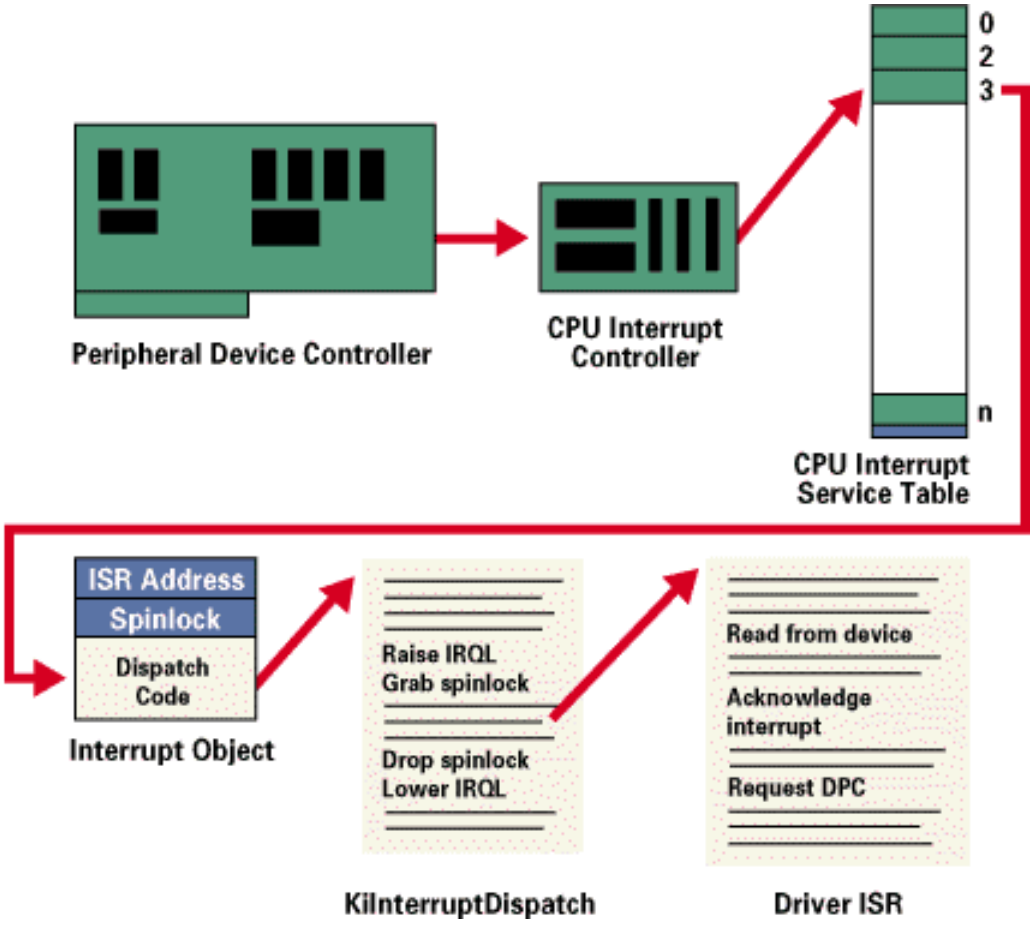
Drivrutiner i NT



hello.cpp för NT

```
extern "C" {

#include <NTDDK.h>

}

extern "C" NTSTATUS DriverEntry (IN PDRIVER_OBJECT pDriverObject,
                                 IN PUNICODE_STRING pRegistryPath)

{

    return STATUS_SUCCESS;

}
```

Avbrott i WinNT

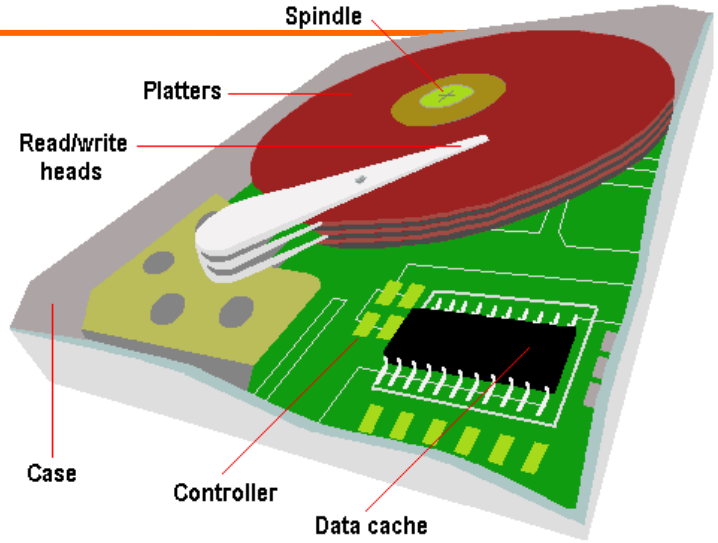


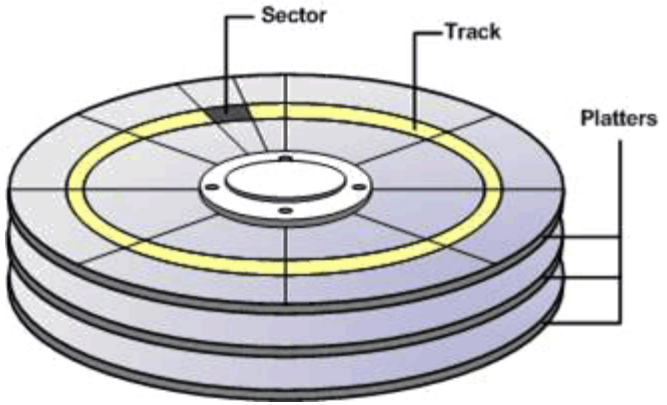
Skivminnen

- Magnetiska skivminnen
  - Hårdskivor
  - Floppy disks
- Optiska skivminnen
  - CD-ROM, CD-R,CD-RW
  - DVD, DVD-R,DVD+R

Magnetiska skivminnen

- Floppy disks
- Hårdskivor
  - IDE (Integrated Drive Electronics)
  - SATA (Serial (PC/)AT Attachment)
  - SCSI (Small Computer Systems Interface)

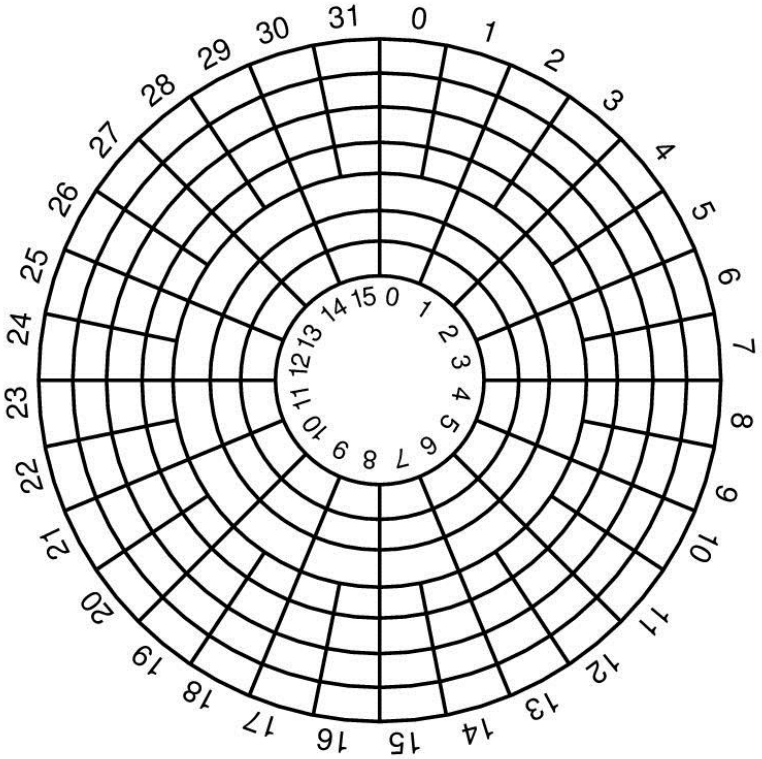


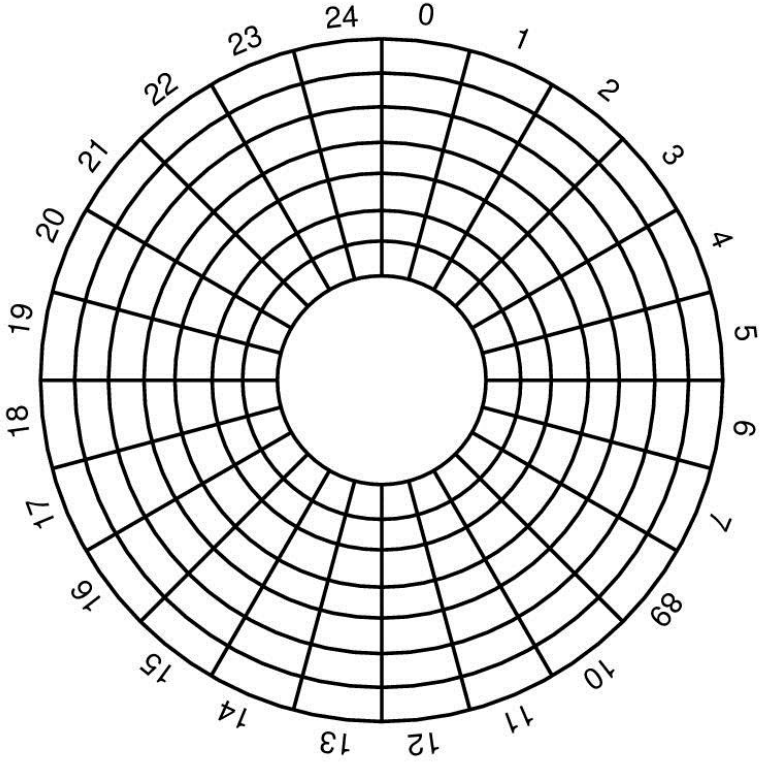


Magnetiska skivminnen

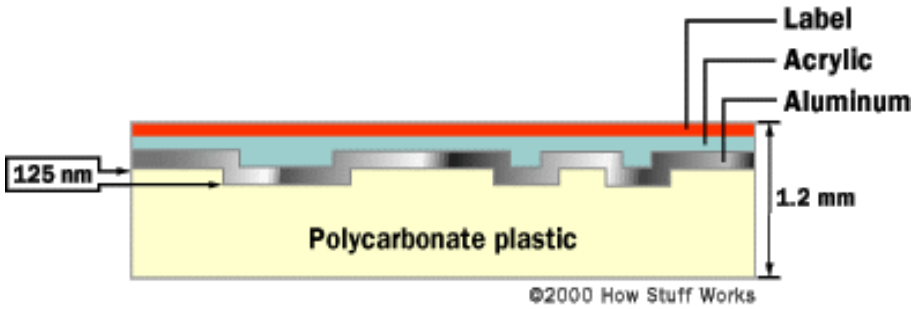
	IBM 360-kB floppy	WD 18300 HD	WD 500 GB
Cylindrar	40	10601	
Spår/cylinder	2	12	
Sektorer/spår	9	281	
Sektorer/skiva	720	35742000	976773168
Bytes/sektor	512	512	512
Diskutrymme	360 kB	18,3 GB	500 GB
Söktid (cylinder-cylinder)	6 ms	0,8 ms	2,0 ms
Söktid (medeltal)	77 ms	6,9 ms	8,9 ms
Rotationstid	200 ms	8,33 ms	8,33 ms
Motor/start/stop	250 ms	20 s	
Tid att läsa en sektor	22 ms	17 µs	5,5 µs
Skiva till buffert	186 kbit/s	240 Mbit/s	748 Mbit/s
Buffert till host			300 MB/s

Disk layout

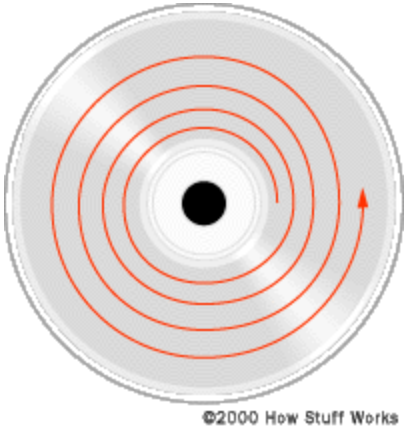

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

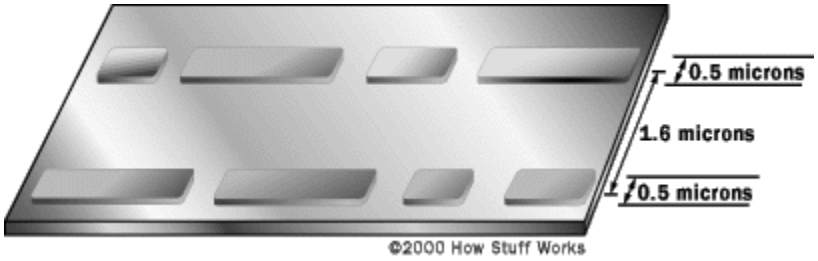

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Optiska skivminnen



44,100 samples/channel/second x 2 bytes/sample
x 2 channels x 74 minutes x 60 seconds/minute = 783,216,000 bytes



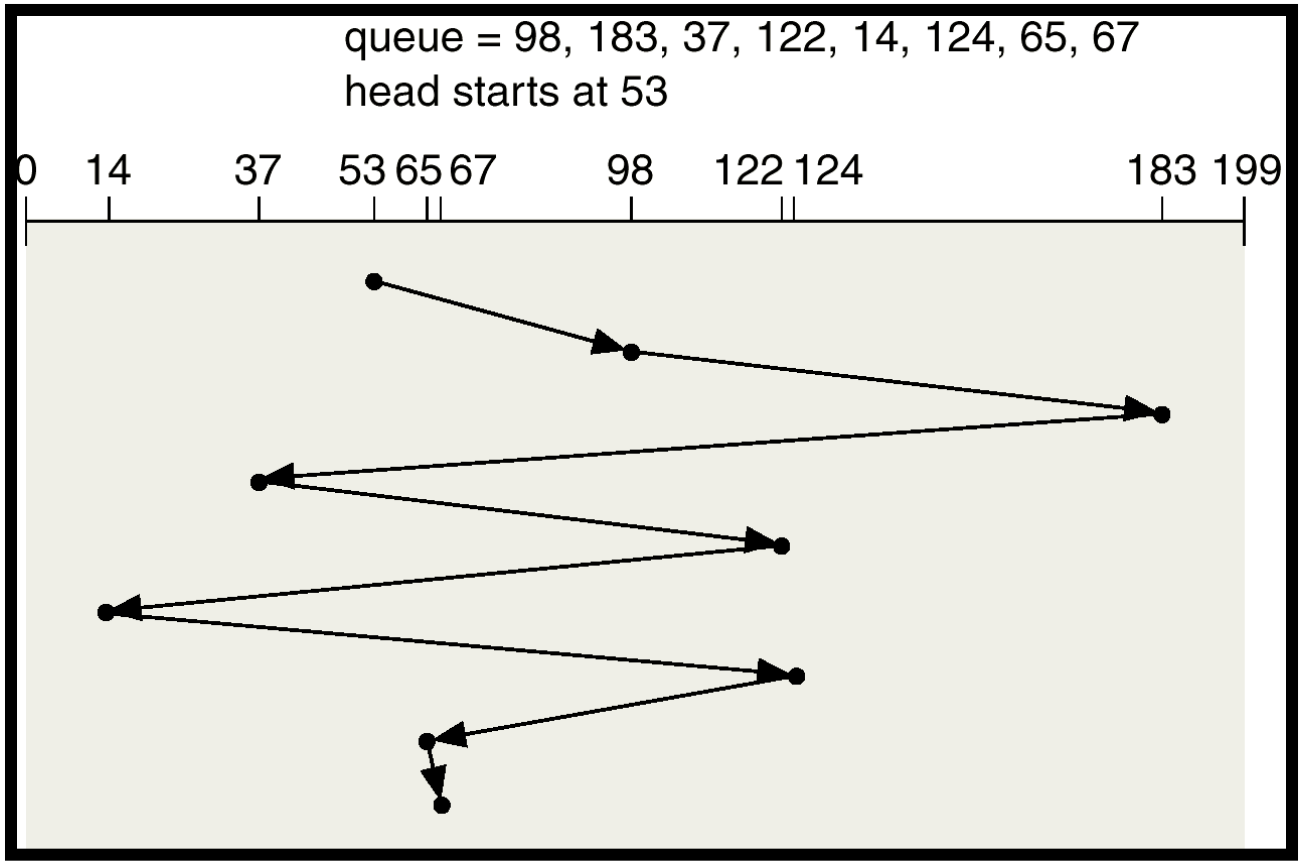


Skedulering av diskarm

- Tid som behövs för att läsa från/skriva till skiva är beroende av 3 saker
  - Söktid (tid för att förflytta till rätt cylinder)
  - Rotationsfördröjning
  - Tid för dataavläsning
- Söktid är dominant
  - Vi vill minimera söktid

FCFS

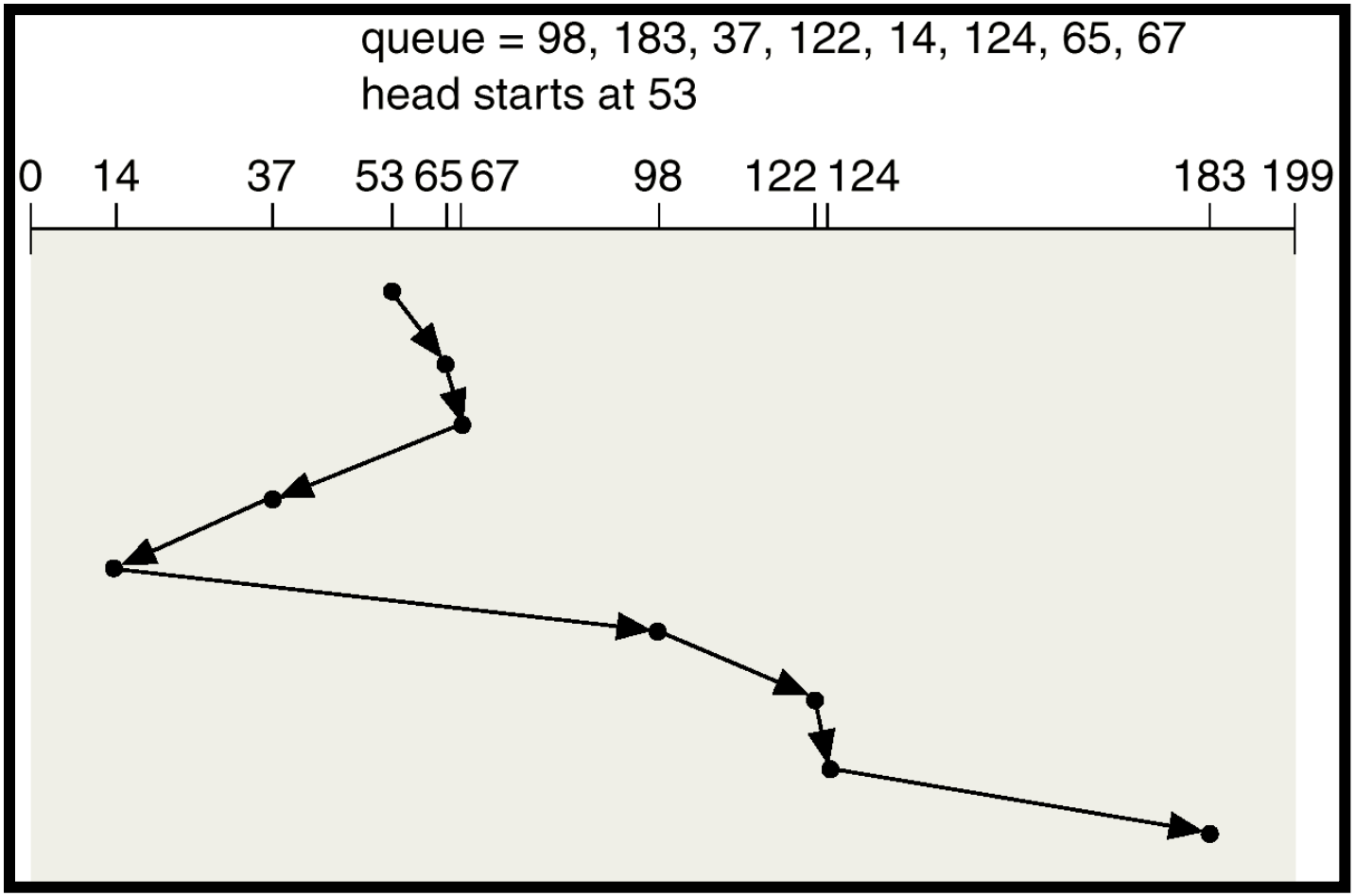
Illustration shows total head movement of 640 cylinders.



SSTF

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

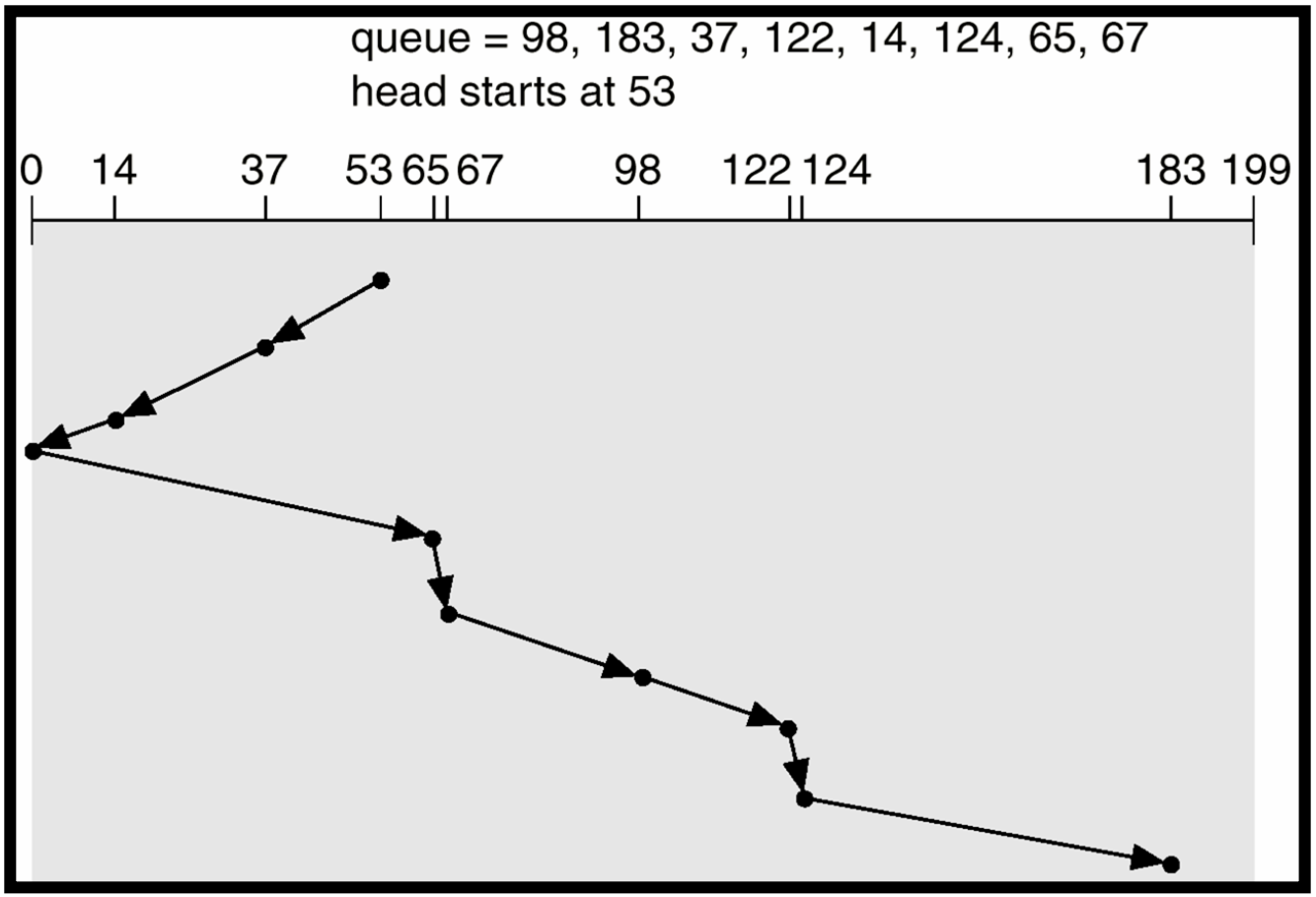
SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SCAN/Elevator

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
- Illustration shows total head movement of 208 cylinders.

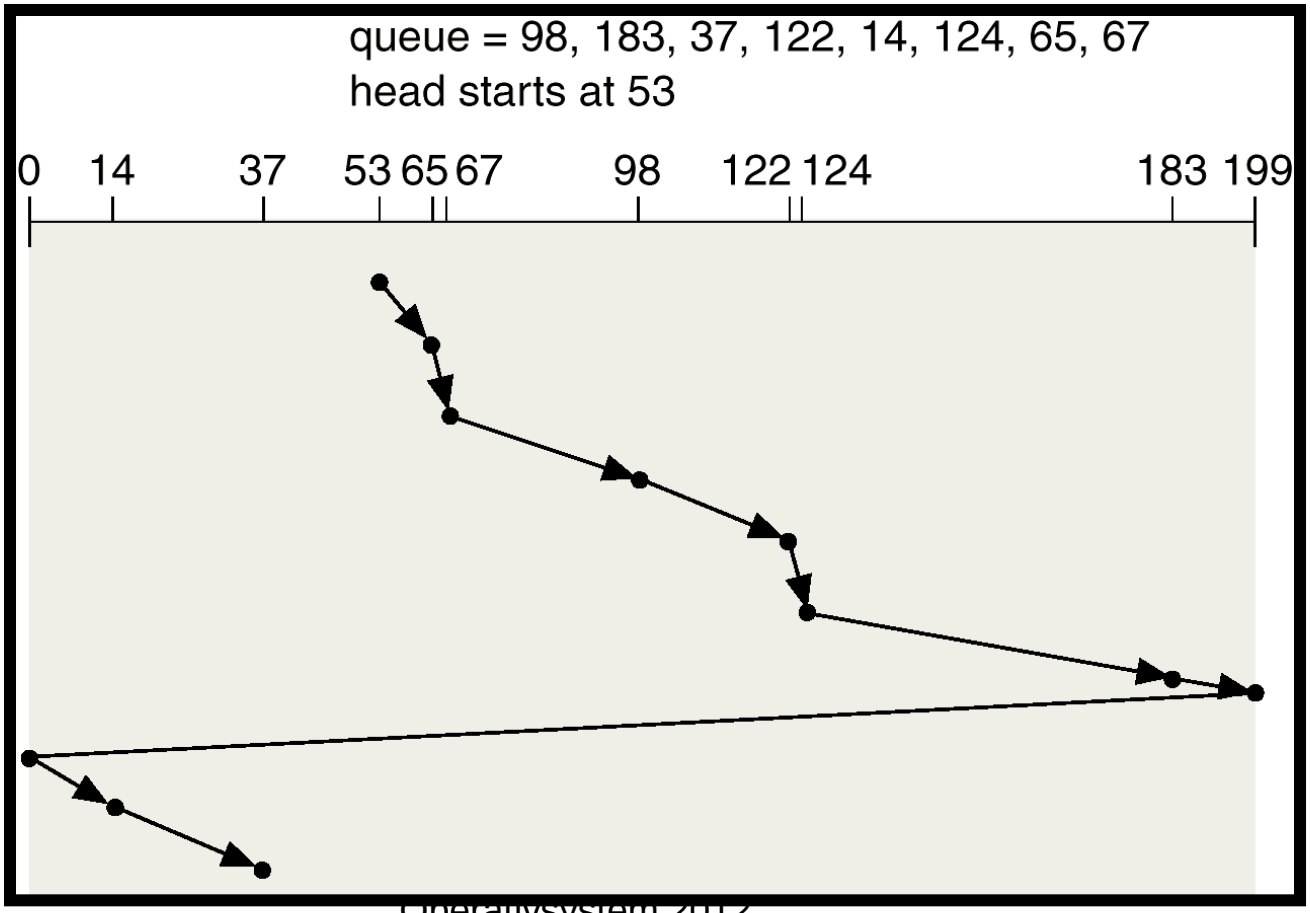
SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

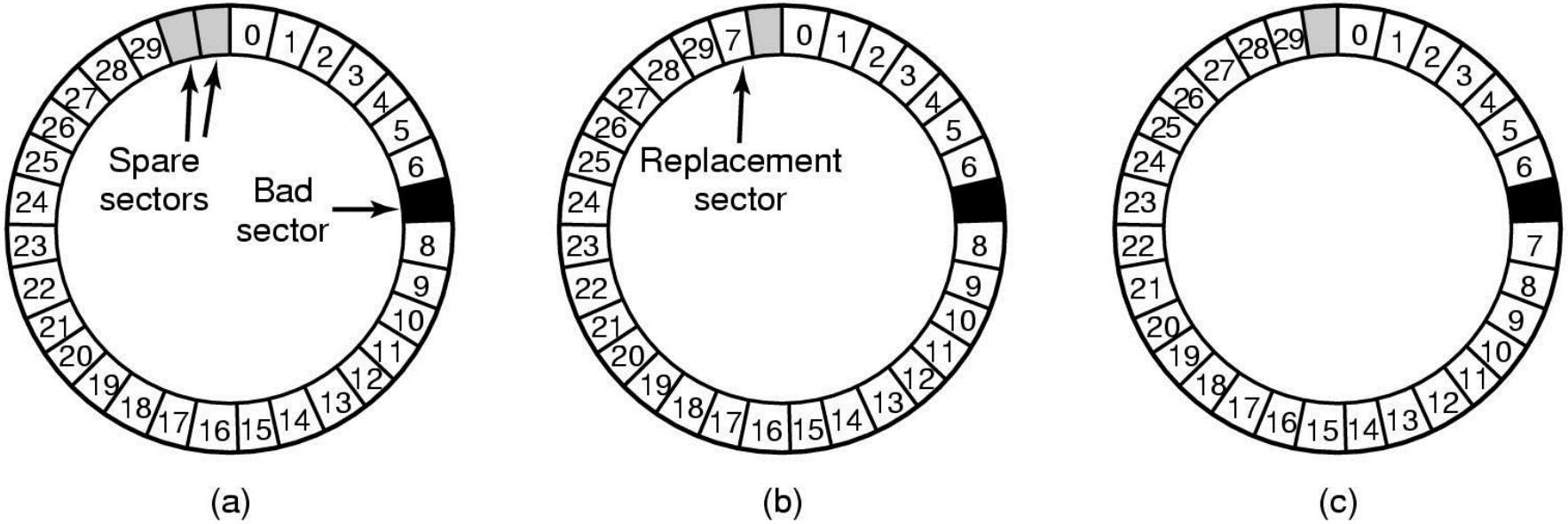
C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

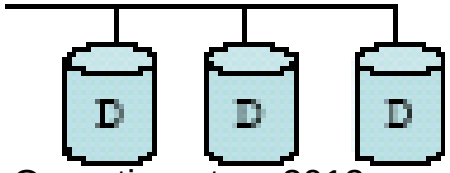
Error Handling



- A disk track with a bad sector
- Substituting a spare for the bad sector
- Shifting all the sectors to bypass the bad one

RAID

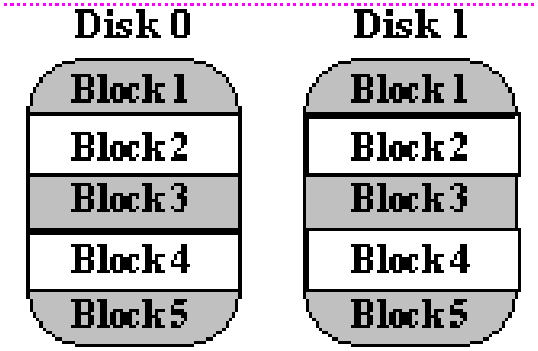
- Redundant Array of Inexpensive Disks
- I ”nivåer” 0-5
- RAID 0: Kallas även striping
  - Flera hårddiskar arbetar som en enhet – data sprids ut över hårddiskarna
- Ingen redundans – går en hårddisk sönder förlorar man all data



Disk 1	Disk 2	Disk 3
A	B	C
D	E	F
G	H	I

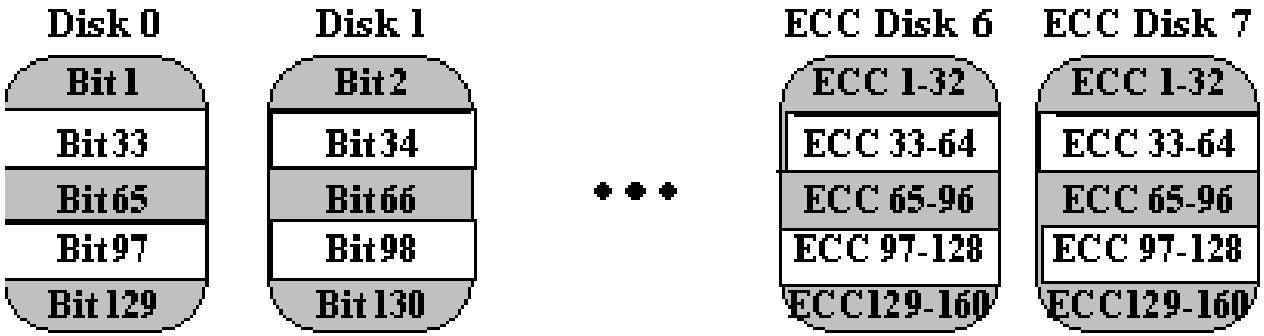
RAID 1

- Kallas även spegling (mirroring)
- Minst 2 hårddiskar – en exakt kopia av data lagaras på minst en annan hårddisk
- Höjd läsprestanda



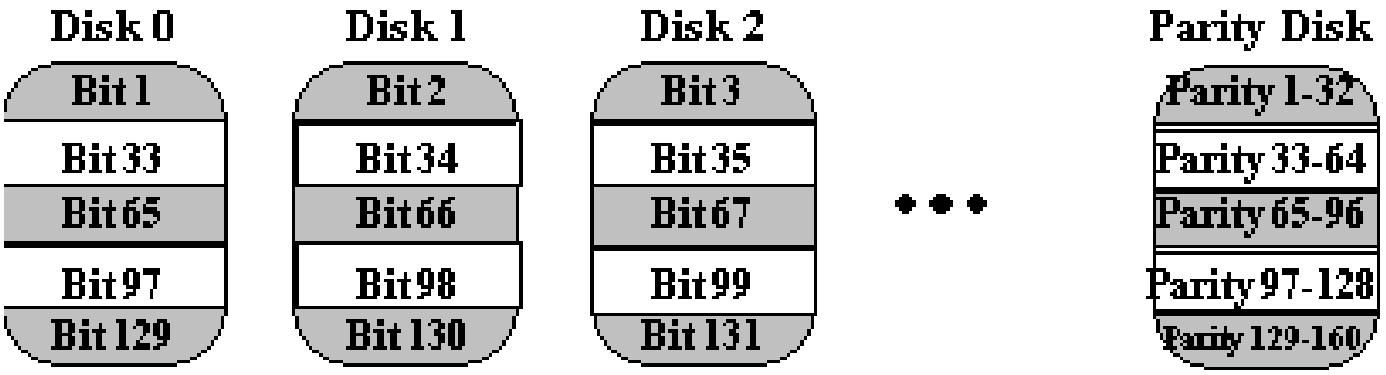
Raid 2

- Data delas i bits och sprids ut på flere skivor
- Dessutom adderas redundans, som dock kräver flere skivor



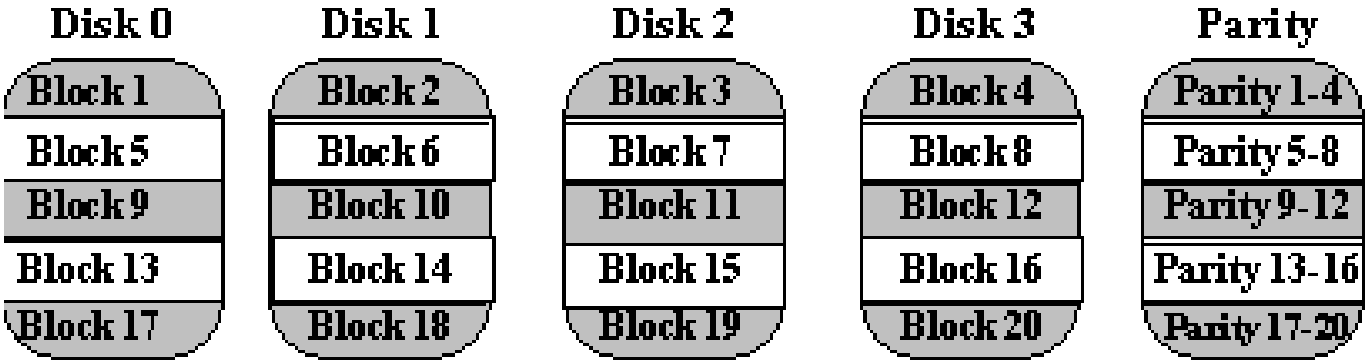
Raid 3

- Data delas i bytes och sprids ut på flere skivor
- Dessutom adderas redundans, kräver dock endast en skiva



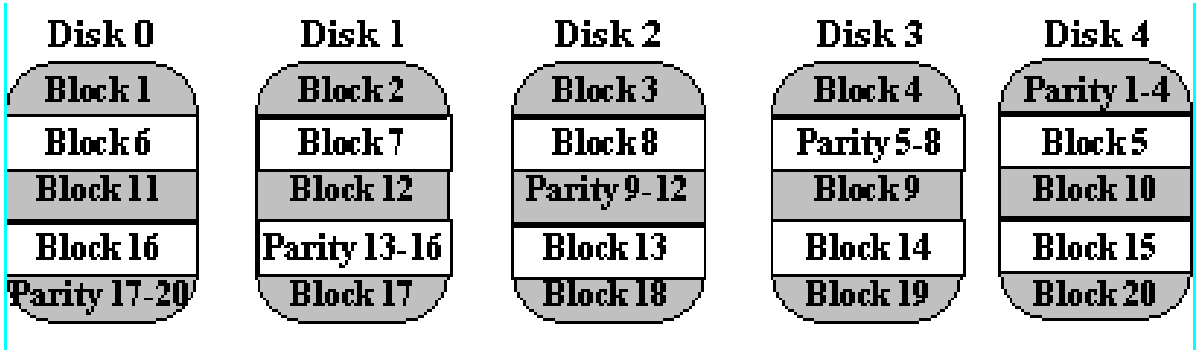
Raid 4

- Som Raid 3, men data lagras i större block
- All paritet skrivs på en skilld skiva



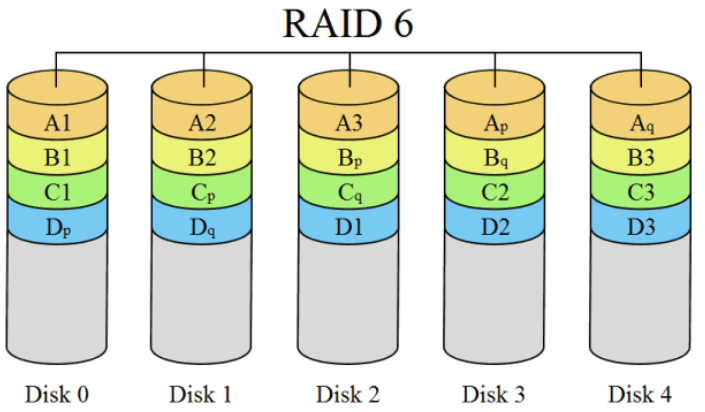
Raid 5

- Striping + parity
- Data skrivs i block på många skivor, men pariteten sprids ut över många skivor
- Data och paritet finns alltid på olika skivor



Raid 6

- Striping + dubbel paritet
- Data skrivs i block på många skivor, men pariteten sprids ut över många skivor
- Data och paritet1 / paritet2 finns alltid på olika skivor



Bildkälla: wikipedia

RAID Prestanda

RAID Type	Small Read	Small Write	Large Read	Large Write	Storage Efficiency
Level 0	1	1	1	1	1
Level 1	1	1/2	1	1/2	1/2
Level 3	1/G	1/G	(G-1)/G	(G-1)/G	(G-1)/G
Level 5	1	max (1/G,1/4)	1	(G-1)/G	(G-1)/G