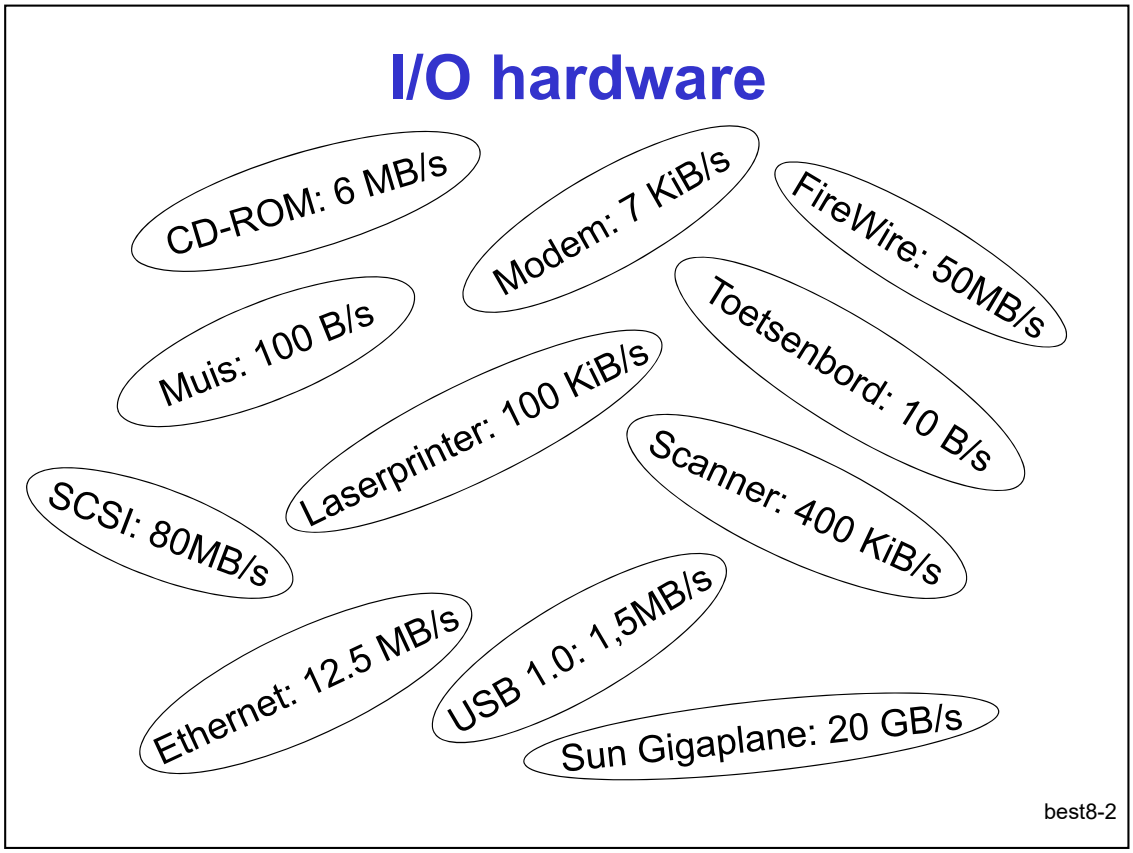
Les 8: Input/Output

The value of a system grows as the square of its number of users
Metcalfe's law – Robert Metcalfe.

best8-1



Randapparaten zijn er in vele soorten. Enkel de transfersnelheid kan al over 9 orden van grootte verschillen.

Kenmerken Input/Output

Aspect	Variatie	Voorbeeld
Transfergrootte	Teken Blok	Terminal Disk
Toegangsmethode	Sequentieel Direct	Modem CD-ROM
Transfer	Synchroon Asynchroon	Tape Toetsenbord
Delen	Gedeeld Exclusief	Toetsenbord Tape
Snelheid	10 B/s 80 MB/s	Toetsenbord Disk
IO-richting	Read-only Write-only Read-write	CD-ROM Graphics controller Disk

Doel IO: uniforme interface voor gebruiker, applicatie, kern

best8-3

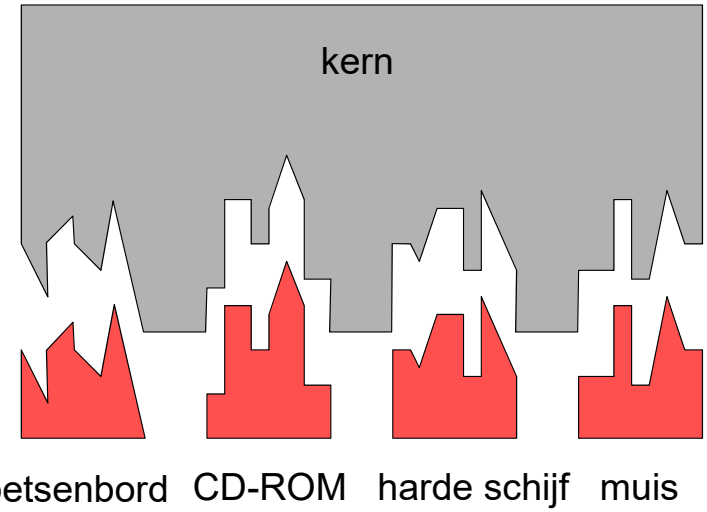
Input/output is een vrij moeilijk abstraheerbare materie omwille van de grote verschillen die er zijn in de kenmerken van de randapparaten, en omwille van de snelle evolutie in de randapparaten. Geregeld zijn er nieuwe protocollen, standaarden, en toepassingen. De voornaamste doelstelling van het IO-subsysteem is om ervoor te zorgen dat de complexiteit van dit alles zoveel mogelijk verborgen blijft voor de eindgebruiker, voor applicatieprogramma's en voor de kern, uiteraard met behoud van de vereiste prestatie.

Overzicht

- Situering
- IO-systeem
  - IO-bibliotheek
  - IO-subsysteem
  - Device drivers
  - Onderbrekingsroutines
- Schijfbeheer
  - Opbouw
  - Schijfplanning
  - RAID
  - Opslagsystemen

best8-4

Problematiek



Communicatie met randapparaat gebeurt in systeemmode

best8-5

De randapparaten kunnen enkel vanuit systeemmode aangestuurd worden. Het besturingssysteem kan immers niet toelaten dat gebruikersprogramma's op het ogenblik dat het hen uitkomt zomaar interageren met de randapparaten. Dit creëert een dilemma: het beperken van IO tot de kern maakt het besturingssysteem enerzijds stabieler, maar anderzijds moet het dan wel toelaten dat er vreemde code in die kern uitgevoerd wordt, de zgn. device drivers.

Het besturingssysteem kan immers onmogelijk zorgen voor alle device drivers van randapparaten die ooit aan de computer aangesloten zouden kunnen worden zodat de device driver in de praktijk vaak door de fabrikant van het randapparaat zal geleverd worden. Hierdoor kan de leverancier van het besturingssysteem geen garanties bieden over de goede werking van de device driver.

Op de afbeelding is te zien dat elk randapparaat een verschillende device driver vereist, en dat de kern zich in principe moet aanpassen.

Interactie met randapparaten

- Geprogrammeerde overdracht
- Onderbrekingen
- Directe geheugentoegang (DMA)

best8-6

De communicatie met randapparaten gebeurt door de device driver in systeemmode. De kern heeft drie manieren om met een randapparaat te communiceren: geprogrammeerde overdracht, onderbrekingen, en DMA.

Geprogrammeerde I/O

```
copy_from_user(buffer, p, count);
for (i=0; i<count; i++) {
   while (*printer_status != READY) /* wacht */;
   *printer_data = p[i];
}
```

best8-7

Bij geprogrammeerde overdracht worden de gegevens één na één naar het randapparaat gestuurd. Na het versturen van een gegeven wordt er gewacht totdat het randapparaat opnieuw vrij is om het volgende gegeven te versturen. Dit wachten gebeurt in een wachtlus. Dit is een snelle manier om gegevens over te dragen voor snelle randapparaten – omdat de wachtlus dan in de praktijk niet of maar zeer kort zal uitgevoerd worden. Indien men op deze manier gegevens wenst te versturen naar een traag randapparaat (zoals b.v. een printer), dan zal er te veel tijd verloren gaan bij het wachten in de wachtlus.

De oproep copy_from_user is nodig om het blok gegevens (aangewezen door buffer) te kopiëren uit de adresruimte van het proces naar de adresruimte van de kern. We zitten hier dus met twee adressen (buffer en p) die beide uit een verschillende adresruimte komen. De omgekeerde bewerking bestaat ook: copy_to_user.

Onderbrekingen

```
copy_from_user(buffer, p, count);
while (*printer_status != READY) /* wacht */;
enable_interrupt(PRINTER);
*printer_data = p[0]; i = 1;
wait(signaal);
```

onderbrekingsroutine

```
if (i > count) {
    disable_interrupt(PRINTER);
    signal(signaal);
} else {
   *printer_data = p[i]; i++;
}
return_from_interrupt();
```

best8-8

Bij onderbrekingen wordt het eerste gegeven naar het randapparaat gestuurd, en worden de bijkomende gegevens door de onderbrekingsroutine opgestuurd. Onderbrekingen veroorzaken redelijk wat overhead bij de uitwisseling van gegevens met een randapparaat. Daarom zal men onderbrekingen voornamelijk gebruiken bij trage randapparaten.

DMA

```
copy_from_user(buffer, p, count);
set_up_DMA (p, count, port);
wait(signaal);
```

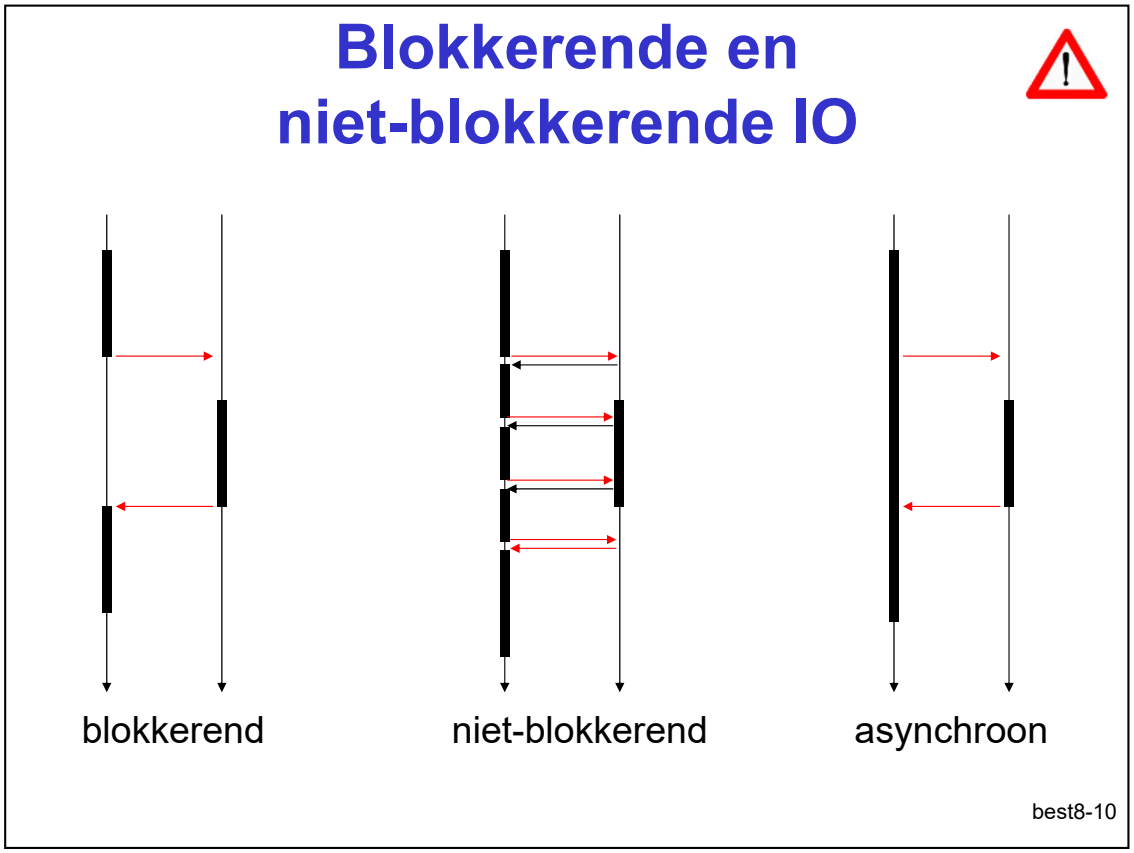
onderbrekingsroutine

```
signal(signaal);
return_from_interrupt();
```

Asynchrone communicatie

best8-9

Hier wordt al het werk uitbesteed aan de DMA regelaar. De kern wacht totdat de DMA-regelaar te kennen geeft dat de transfer afgelopen is, en dat het proces weer verder mag lopen. In dit geval is er dus maar 1 onderbreking nodig, namelijk op het einde van de volledige DMA-transfer. De aanvrager wacht in dit geval totdat het volledige blok getransfereerd werd. Dit is echter geen vereiste. De communicatie zou ook volledig asynchroon kunnen verlopen.



De meeste systeemoproepen zijn **blokkerende systeemoproepen**: ze geven de controle pas terug aan het proces waardoor ze opgeroepen werden nadat hun taak volledig afgelopen is. Indien een taak om de één of andere reden blijft hangen, dan zal het proces in de geblokkeerde toestand blijven. Dit is de eenvoudigste oproepsemantiek. Men roept de kern op, en deze oproep eindigt pas als de taak effectief vervuld is. In sommige gevallen is dit echter geen wenselijke situatie.

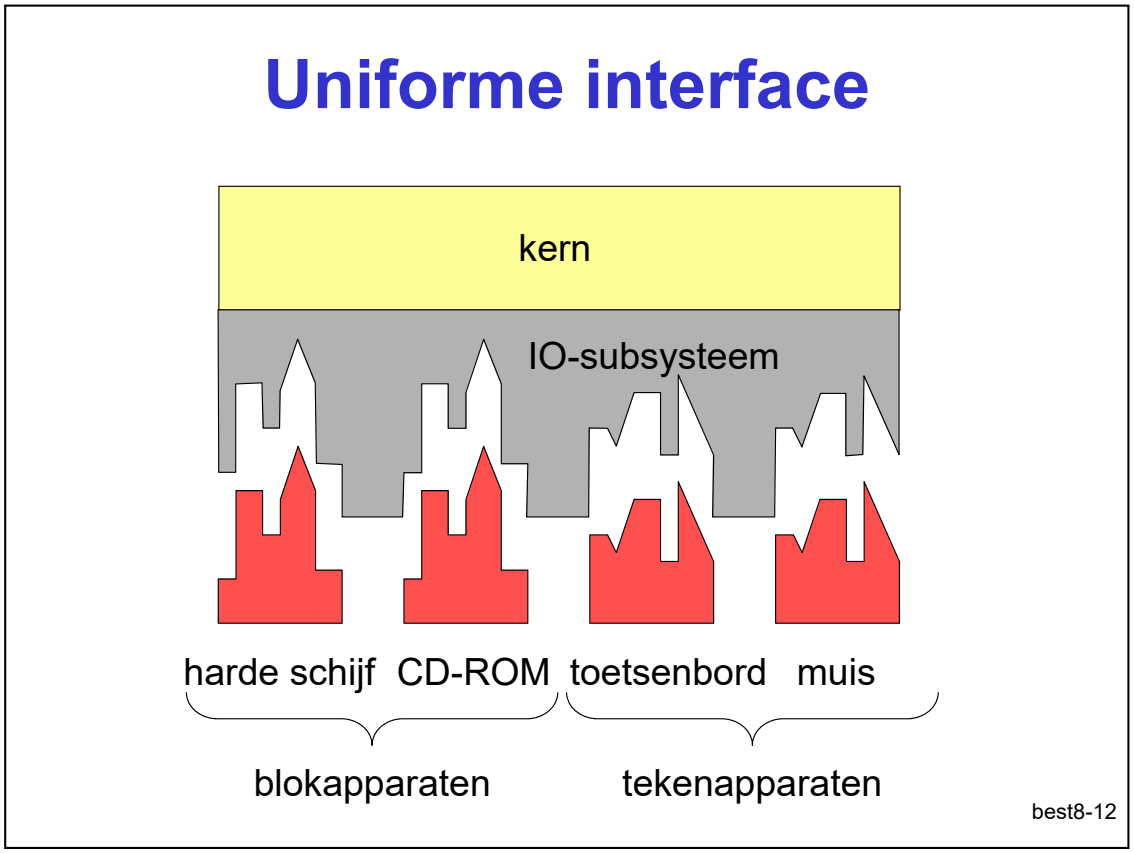
Naast de blokkerende systeemoproepen bestaan er ook **niet-blokkerende systeemoproepen**. Deze geven het resultaat van de oproep terug indien beschikbaar. Indien het resultaat nog niet beschikbaar is, geven ze ofwel een partieel resultaat terug, ofwel de boodschap dat er geen resultaat is. In de plaats van te blokkeren wordt m.a.w. de beschikbare informatie teruggegeven aan het proces die met de ontvangen informatie dan iets kan aanvangen (b.v. een wachtlus aansturen). Bij niet-blokkerende operaties vermijdt men dat een proces blokkeert zonder dat men daar veel kan aan veranderen. Door gebruik te maken van draden kan men niet-blokkerende systeemoproepen simuleren met blokkerende primitieven. Het volstaat om een afzonderlijke draad te laten blokkeren. De andere draden zullen gewoon verder werken.

Tenslotte zijn er ook de **asynchrone** oproepen. In die gevallen wacht het proces niet expliciet op een antwoord, maar zal dat antwoord door de kern teruggestuurd worden (b.v. door het genereren van een signaal, of door het veranderen van de waarde van een veranderlijke). Het volledig asynchroon werken is complex en kan gemakkelijk aanleiding geven tot fouten. Anderzijds moet men zich wel realiseren dat de randapparaten doorgaans asynchroon werken. (b.v. een printer: een byte wordt opgestuurd, en de printerinterface antwoordt met een onderbreking van zodra de byte verwerkt werd).

Overzicht

- Situering
- IO-systeem
  - IO-bibliotheek
  - IO-subsysteem
  - Device drivers
  - Onderbrekingsroutines
- Schijfbeheer
  - Opbouw
  - Schijfplanning
  - RAID
  - Opslagsystemen

best8-11

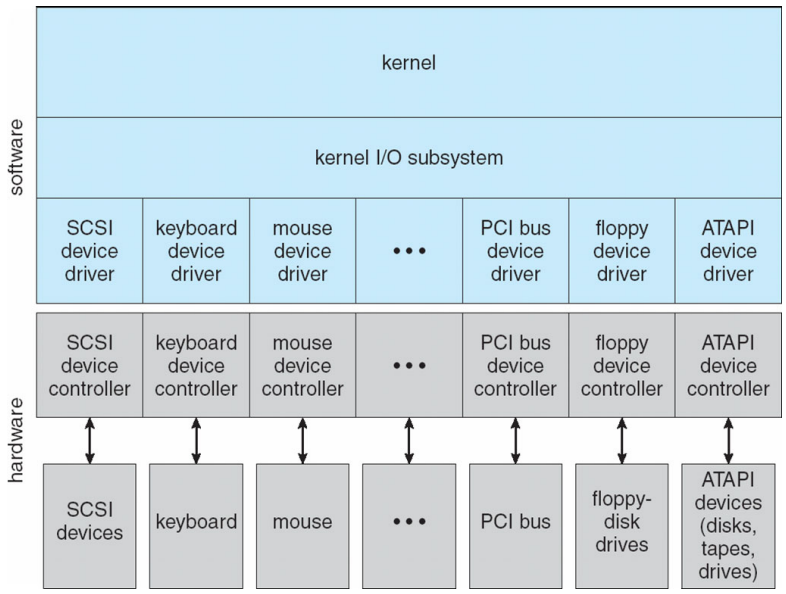


In de praktijk zal de kern een interface opleggen aan de device drivers. Omdat het gedrag en de prestatie van tekengebaseerde en blokgebaseerde randapparaten vrij sterk verschillend is, biedt men doorgaans een aparte interface aan voor de twee types. Doordat de interfaces goed gedefinieerd zijn, is het gemakkelijker om de invloed van een device driver op de rest van de kern in te schatten, en de kern in de mate van het mogelijke te beschermen tegen ongewenst gedrag door de driver. Veel van de stabiliteitsproblemen in besturingssystemen worden veroorzaakt door device drivers die fouten bevatten en op die manier de stabiliteit van heel de kern in het gedrang brengen. Het probleem met device drivers is dat ze in systeemmode (moeten) uitgevoerd worden en ze daardoor eigenlijk almachtig zijn. Probeer je even voor te stellen welke ravage een bengelende wijzer van een device driver kan aanrichten in de kern.

Naast de twee belangrijke categorieën van teken- en blokapparaten zijn er ook nog een aantal randapparaten die daar moeilijk mee te associëren zijn zoals de timer en het netwerk. Om toe te laten dat een server op verschillende sockets (netwerkverbindingen) tegelijk luistert, maakt men gebruik van een select oproep. Deze oproep retourneert informatie over de sockets die klaar zijn voor gebruik (lezen of schrijven).

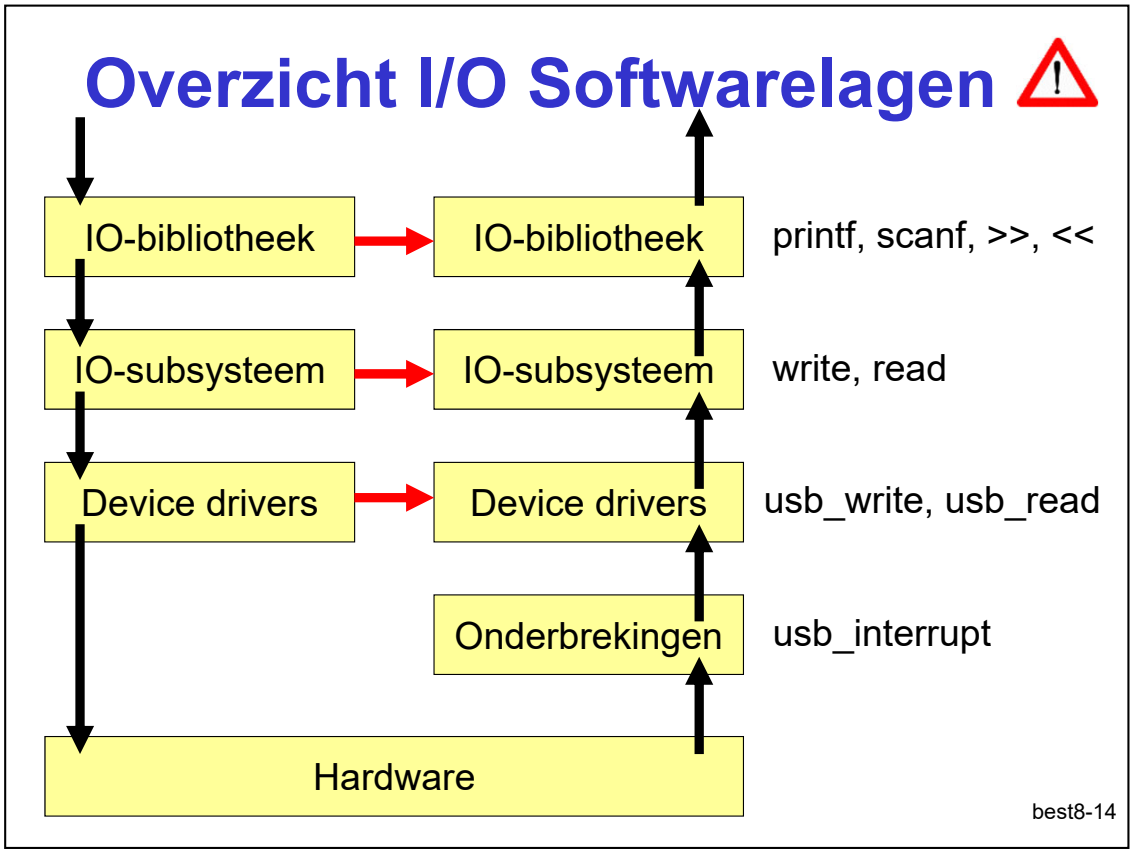
Om de deur open te houden voor randapparaten die niet 100% passen in het voorgesteld schema, is er doorgaans ook een achterpoortje om zeer apparaatspecifieke opdrachten uit te voeren. In Unix noemt deze oproep ioctl (io controle).

kern IO-structuur



best8-13

In deze afbeelding wordt geïllustreerd dat de device driver interageert met de regelaar van het betrokken randapparaat. Deze regelaar is dan op zijn beurt verbonden met het eigenlijke randapparaat. Zo zal de printer driver b.v. de parallelle poort of de USB-poort aansturen. Deze poorten zijn dan verder verbonden met de printer.



In de praktijk is er een stapel van softwaremodules betrokken bij de interactie met randapparaten. Op het topniveau is er de IO-bibliotheek voor een bepaalde taal (C, C++, Java, …). Deze bibliotheek maakt gebruik van de primitieven die door het besturingssysteem ter beschikking gesteld worden (read, write, …) en die door het IO-subsysteem vertaald worden in oproepen naar de betrokken device drivers. In het voorwaartse pad zal de device driver de hardware rechtstreeks aansturen. In het achterwaartse pad kan er al dan niet gebruik gemaakt worden van onderbrekingen om informatie van het randapparaat naar de device driver te sturen.

In de drie bovenste blokken kan men beslissen om gegevens te bufferen (zowel bij output als bij input). Dit wil zeggen dat men vanaf dat niveau meteen het opwaartse pad neemt i.p.v. de aanvraag naar de dieperliggende lagen van de stapel door te geven.

IO-bibliotheek vs. IO-subsysteem

```
char buffer[N];
int sum = 0;

FILE *h = fopen(“t.t","r");
while (sum < 8192*1024) {
   fread(buffer, 1, N, h);
   sum += N;
}
fclose(h);
```

IO-bibliotheek

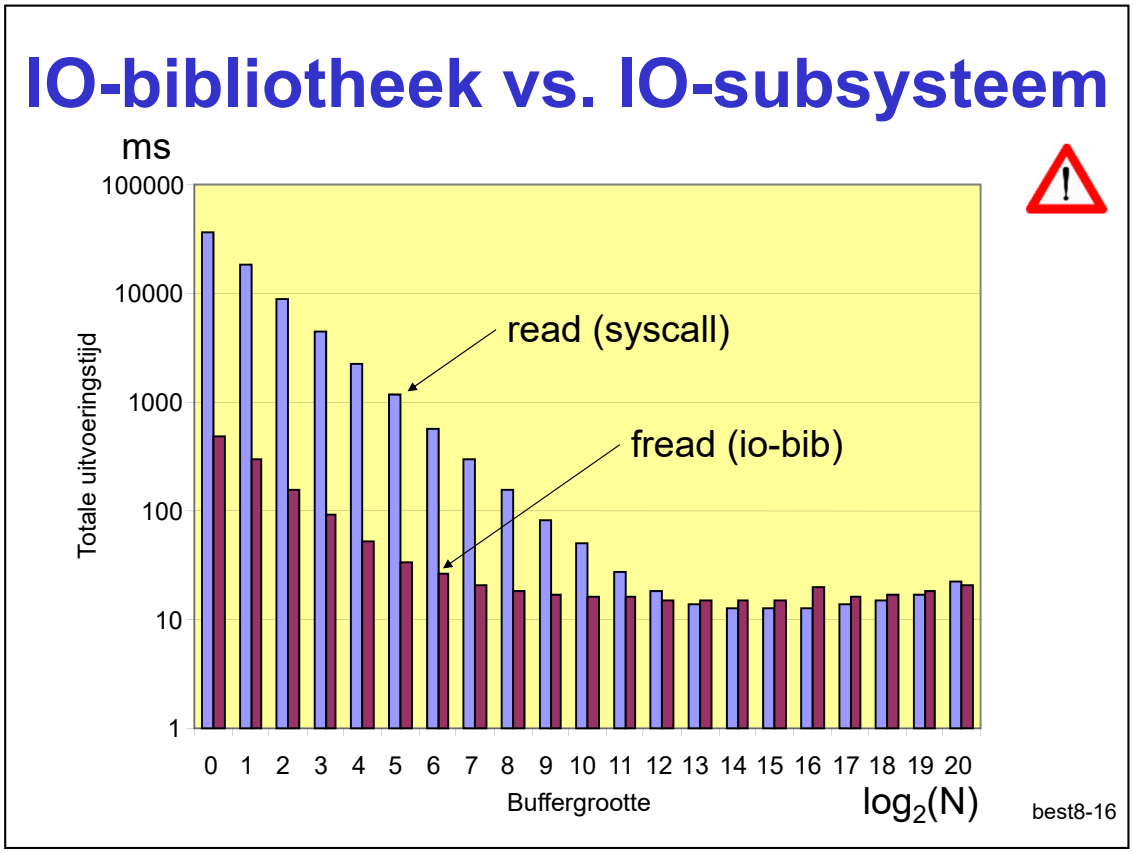
```
char buffer[N];
int sum = 0;

int h = open(“t.t",O_RDONLY);
while (sum < 8192*1024) {
   read(h, buffer, N);
   sum += N;
}
close(h);
```

IO-subsysteem

best8-15

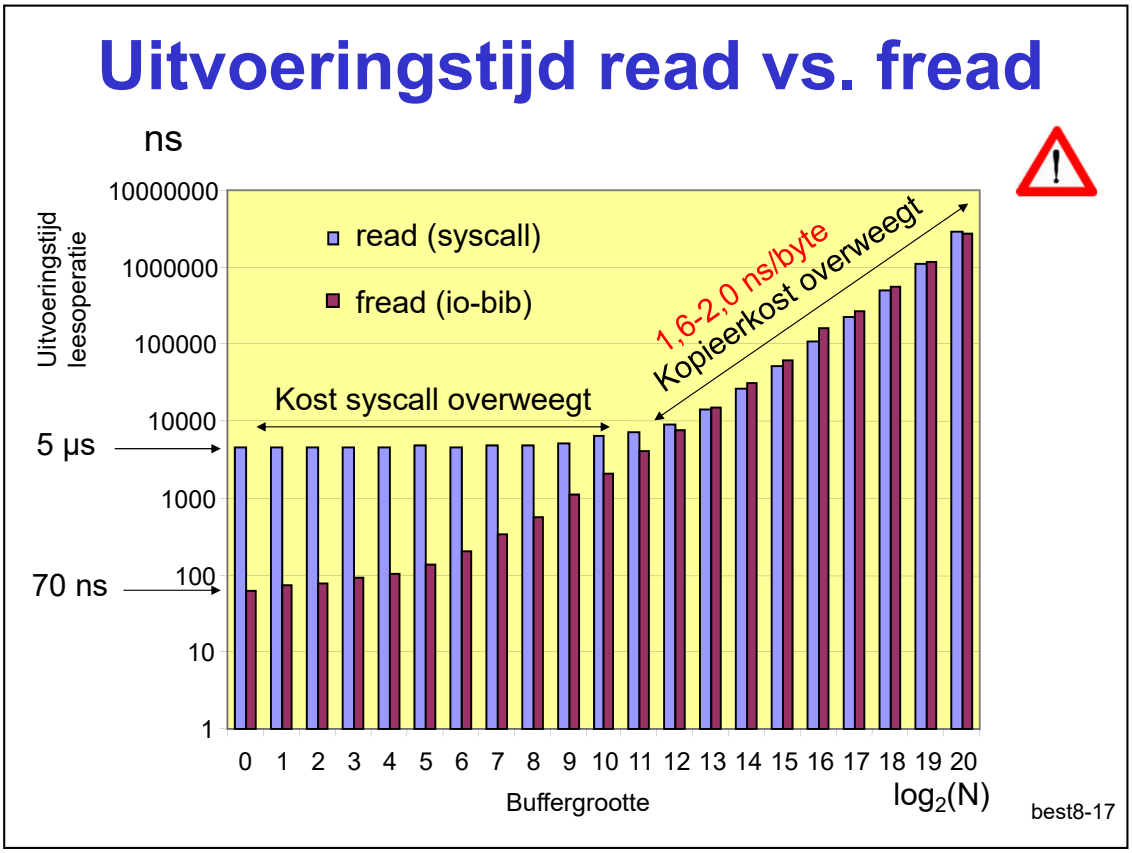
De IO-bibliotheek voorziet in een reeks oproepen die aangepast zijn aan de programmeertaal waarvoor hij ontwikkeld werd. In de linkerhelft van de afbeelding wordt het bestand ‘t.t’ geopend om te lezen en worden de eerste 8 MiB gelezen, in blokken van N groot, en met behulp van de C-bibliotheek. In de rechterhelft gebeurt precies dezelfde operatie, maar ditmaal met de systeemoproepen van het IO-subsysteem. Men zou kunnen verwachten dat indien men rechtstreeks gebruik maakt van het IO-subsysteem, men een efficiëntere implementatie krijgt. De realiteit is echter complexer.



Het verschil in prestatie tussen de twee programma's is frappant.

Vooreerst zien we dat de uitvoeringstijd daalt indien we het bestand in grotere stukken inlezen. Bij het gebruik van read is dit werkelijk spectaculair. De daling is daar evenredig met het aantal oproepen van read dat uitgevoerd wordt. Vanaf 8 KiB blijkt de uitvoeringstijd niet gevoelig meer te dalen. Blijkbaar vormt de uitvoering van read (met de bijhorende contextwisseling naar de kern) bij kleine buffergroottes de grootste kost. Vanaf 8 KiB is het aantal contextwisselingen zo klein geworden dat de invloed ervan op de uitvoeringstijd onbelangrijk wordt. Vanaf dan is het de tijd van de transfer zelf die de uitvoeringstijd bepaalt.

Bij het gebruik van fread is er ook een daling, maar minder spectaculair. Ook hier is de daling evenredig met het aantal oproepen van fread. De kost van een oproep blijkt echter minder groot te zijn. De verklaring is dat fread intern een buffer bijhoudt. Bij het lezen van de eerste byte wordt een compleet blok van de schijf ingelezen. Alle bijkomende bytes worden uit dit blok gehaald waardoor het dus niet langer nodig is om read op te roepen (en dus een contextwisseling kan vermeden worden). Het veelvuldig oproepen van fread heeft echter nog steeds zijn kost (zij het kleiner dan de kost van read). De snelheid van read en fread zijn ongeveer dezelfde voor 8 KiB wat doet vermoeden dat fread blokken van 8 KiB inleest. Voor blokken van groter dan 8 KiB is read efficiënter dan fread. De bijkomende bufferfunctionaliteit van fread heeft in deze gevallen geen nut meer, en werkt enkel maar vertragend.



Als we de uitvoeringstijd van de individuele leesoperaties bekijken, dan merken we dat voor kleine buffers de kost van het kopiëren van de gegevens verdwijnt in de algemene kost voor het oproepen van de functie read resp. fread die 5 µs resp. 70 ns blijkt te zijn. Merk op dat de systeemoproep read ongeveer 100 keer meer tijd in beslag neemt dan een bibliotheekroutine fread voor kleine aantallen bytes.Voor grote blokken zijn de beide routines ongeveer even snel. Merk op dat 70 ns overeenkomt met een 100-tal instructies terwijl 5 µs over enkele duizenden instructies gaat.

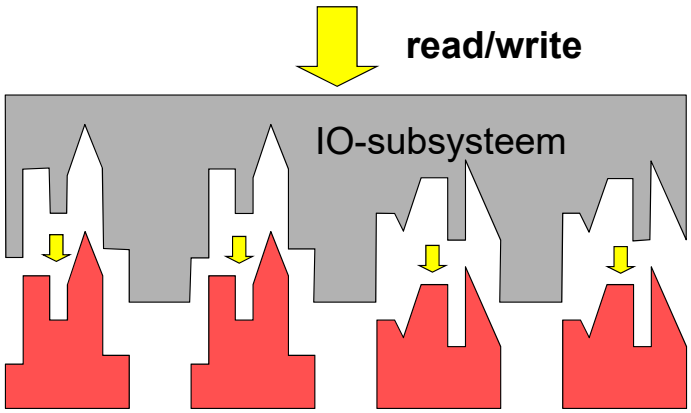
Van zodra de buffers groter worden, stijgt de kost van de systeemoproep lineair met het aantal bytes dat moet gekopieerd worden. In die gevallen overweegt de kopieerkost.

Merk op dat uit deze grafiek blijkt dat ook het IO-subsysteem buffert. In 5 µs is het volstrekt onmogelijk om een blok van de schijf te lezen. Zoals we verder zullen zien werden al deze oproepen gevoed vanuit de buffers en is er weinig of geen interactie met de fysieke schijf geweest.

De conclusie is dat systeemoproepen een heel stuk duurder zijn dan oproepen van reguliere functies en dat systeemoproepen om die reden best zoveel mogelijk vermeden worden. Bij bestands-IO kan men dit doen door grote blokken ineens in te lezen.

IO-subsysteem

- IO Planning
- Buffering
- Caches
- Spooling
- Foutafhandeling
- Apparaatbeheer



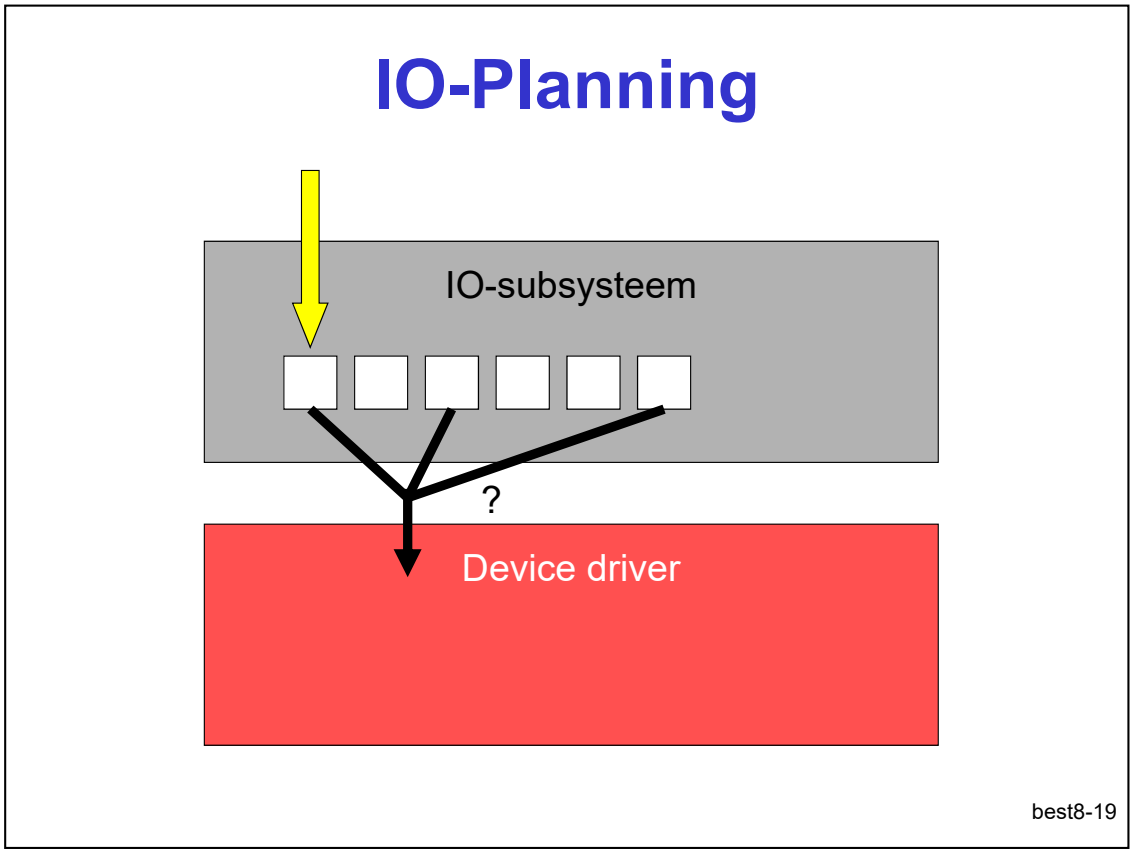
best8-18

Het IO-systeem vervult twee taken.

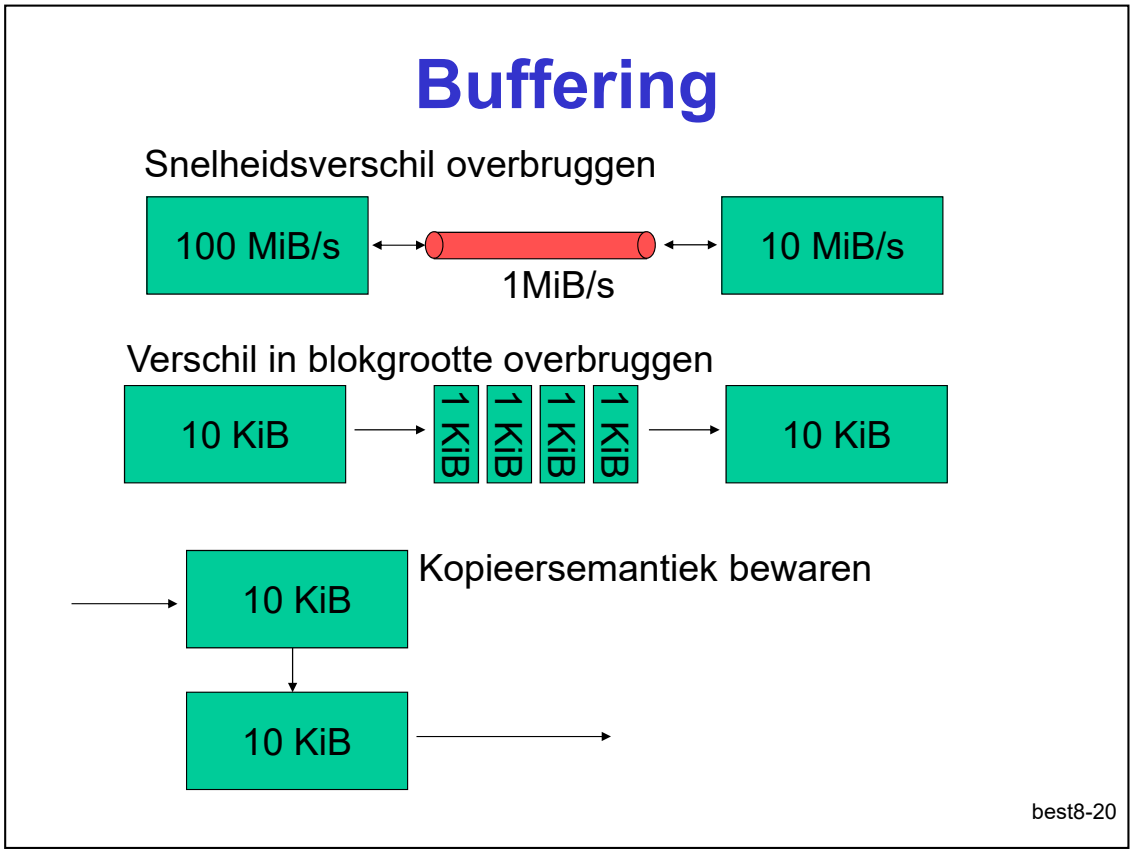
Aan de ene kant biedt het aan de processen een generieke IO-interface aan (read, write, …). Programma's kunnen via deze generieke interface gebruik maken van een ontelbaar aantal randapparaten. De interface zorgt ervoor dat het gebruik van deze randapparaten gestandaardiseerd is (tekenapparaten zullen b.v. vaak via een bestandsinterface gebruikt worden).

Aan de onderzijde biedt het IO-subsysteem ook een interface aan voor de device drivers. Deze interface zorgt ervoor dat de fabrikanten de complexiteit van hun randapparaten kunnen verbergen voor het IO-subsysteem en dus voor de kern. Indien hun device driver voldoet aan de specificaties zoals vooropgesteld door het besturingssysteem, zal het IO-subsysteem met deze driver en dus ook met het achterliggende randapparaat kunnen communiceren.

Het IO-subsysteem neemt alle taken voor zijn rekening die generiek door meerdere device drivers kunnen gebruikt worden.



Bij de IO-planning worden de aanvragen voor een bepaald randapparaat bijgehouden in een wachtlijn van aanvragen. Deze aanvragen kunnen sequentieel afgewerkt worden, maar het IO-subsysteem kan er ook voor opteren om dit niet te doen, en ze af te werken in een andere volgorde als deze volgorde een beter gebruik van het randapparaat garandeert. Op deze manier zal b.v. schijfplanning (zie verder) geïmplementeerd worden.



Buffering zal gebruikt worden in drie gevallen.

1. Daar waar er een groot verschil in snelheid is tussen het randapparaat en het computersysteem. Zo kan een programma sneller gegevens aanmaken dan ze kunnen opgeslagen worden op de schijf. Men zal de gegevens tijdelijk bufferen om ze naderhand op de schijf te bewaren. Omgekeerd zullen de sectoren van de schijf soms sneller gelezen kunnen worden, dan ze kunnen doorgestuurd worden naar het geheugen. Ook daar zal buffering dus noodzakelijk zijn.
2. Soms is er een verschil in grootte van blokken waarmee gegevens doorgestuurd worden. Ook dan zal er gebufferd worden om ofwel kleine blokken te sparen tot er een groot blok verstuurd kan worden, ofwel zal een groot blok gebufferd worden om het in kleine blokjes verder te versturen.
3. Om de kopieersemantiek te bewaren. Als men een buffer wegschrijft met write(buffer, …) dan gaat men ervan uit dat men in de volgende instructie die buffer opnieuw mag gebruiken. Dit kan enkel indien de write-operatie blokkeert totdat de gegevens effectief geschreven werden. Indien de schrijfoperatie om efficiëntieredenen asynchroon werkt, zal de buffer eerst gekopieerd moeten worden alvorens terug te keren. Anders riskeert men de verkeerde gegevens weg te schrijven.

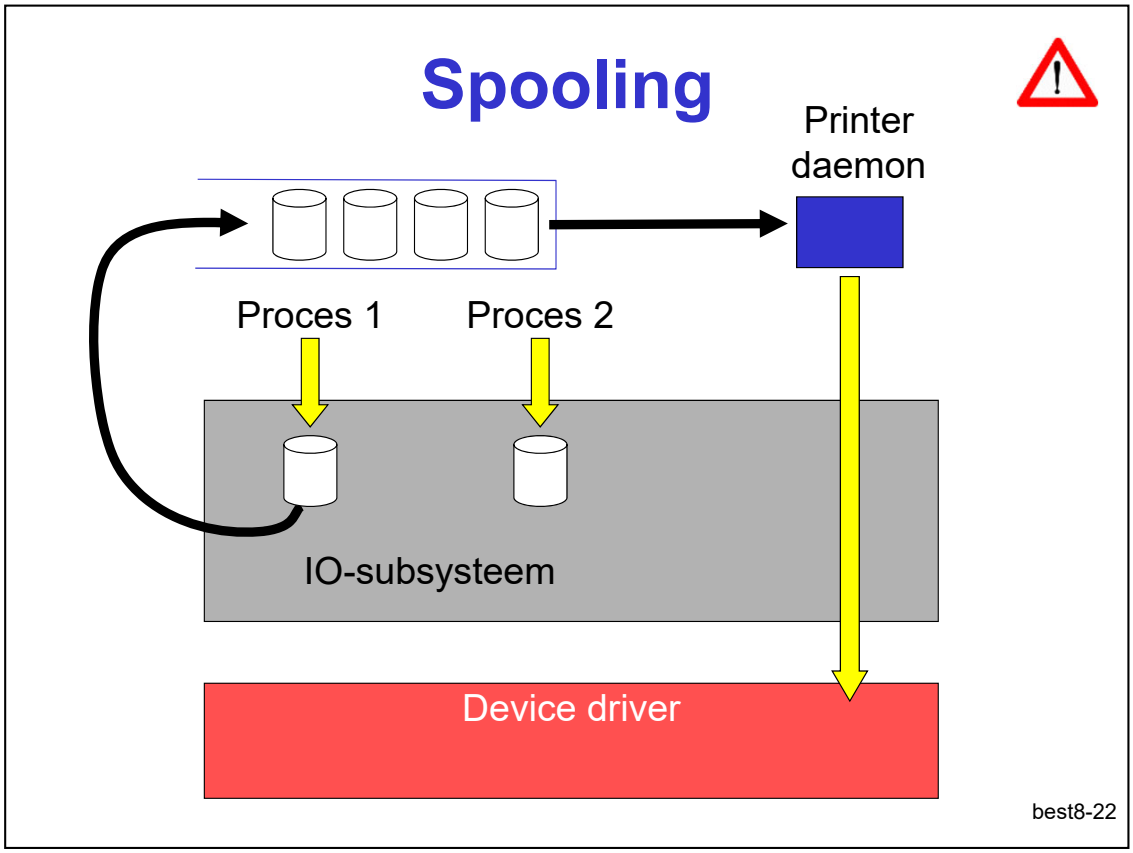
Caches

Omstandigheid	Uitvoeringstijd
Eerste keer	249 ms
Tweede keer	25 ms
Na enige tijd	140 ms

Lezen van 8 MiB in 8 blokken van 1 MiB

best8-21

Een cache is een verzameling van recent gebruikte gegevens. Door de gegevens tijdelijk in het fysiek geheugen bij te houden kunnen een aantal dure toegangen tot het randapparaat vermeden worden (zie voorbeeld van de IO-bibliotheek). Net zoals in de IO-bibliotheek zal het IO-subsysteem de gegevens ook intern bijhouden. Dit blijkt duidelijk uit het volgende experiment. De eerste keer dat een bestand van 8 MiB ingelezen wordt in blokken van 1 MiB neemt dit 249 ms in beslag of omgerekend ongeveer 30 ms per leesoperatie van 1 MiB. Dit doet vermoeden dat de schijf verschillende blokken ineens transfereert van de schijf naar het IO-subsysteem. Indien men hetzelfde programma een tweede keer uitvoert, is de uitvoeringtijd 10 x kleiner. De oorzaak is dat het bestand van 8 MiB zich nog steeds in de bestandscache van het IO-subsysteem bevindt. Na enige tijd zijn sommige blokken uit de cache verdwenen en neemt de uitvoeringstijd opnieuw toe.



De spooler zorgt ervoor dat apparaten die niet deelbaar zijn (zoals b.v. de printer) toch simultaan door verschillende processen kunnen gebruikt worden. Dit wordt gerealiseerd door de processen naar een virtuele printer te laten printen. Deze virtuele printers vangen de output op in een bestand dat – eenmaal de printopdracht afgerond is – in een wachtlijn voor de printer gezet wordt. Verder is er nog een speciaal systeemproces, de zgn. printer daemon, die de wachtlijn in het oog houdt en de printjobs één na één naar de fysieke printer stuurt, daarbij de spooling in het IO-subsysteem overslaat omdat het uiteraard niet de bedoeling is dat ook deze printopdracht in een bestand terecht komt. De printerdaemon is eigenlijk het enige proces dat rechtstreekse toegang tot de printer heeft. Alle andere processen moeten via bestanden printen.

Foutafhandeling

```
int h = open("…);
if (h == -1) {
  printf("%d\n", errno);
  printf("%s\n", strerror(errno));
  perror("testprogramma");
}
```

```
./a.out
2
No such file or directory
testprogramma: No such file or directory
```

best8-23

Foutafhandeling is ook een taak van het IO-subsysteem. Het aantal fouten dat kan voorkomen varieert sterk. Zo zal een printer willen te kennen geven dat de inktpatroon of de papierlade leeg is. Andere randapparaten zoals een schijf moet dan weer kunnen uitdrukken dat bepaalde blokken niet bestaan, of dat bepaalde sectoren fouten geven.

Het IO-subsysteem kan fouten op verschillende manieren teruggeven: b.v. door de systeemoproepen een speciale waarde te laten teruggeven b.v. een 'open' systeemoproep die -1 als handle teruggeeft wijst op een fout. De oorzaak van de fout kan dan teruggevonden worden in de veranderlijke errno (een systeemveranderlijke). Bij HP-UX kan errno in dit geval 23 verschillende waarden aannemen. In Windows dient men de functie GetLastError op te roepen die per draad de laatst opgetreden fout zal teruggeven.

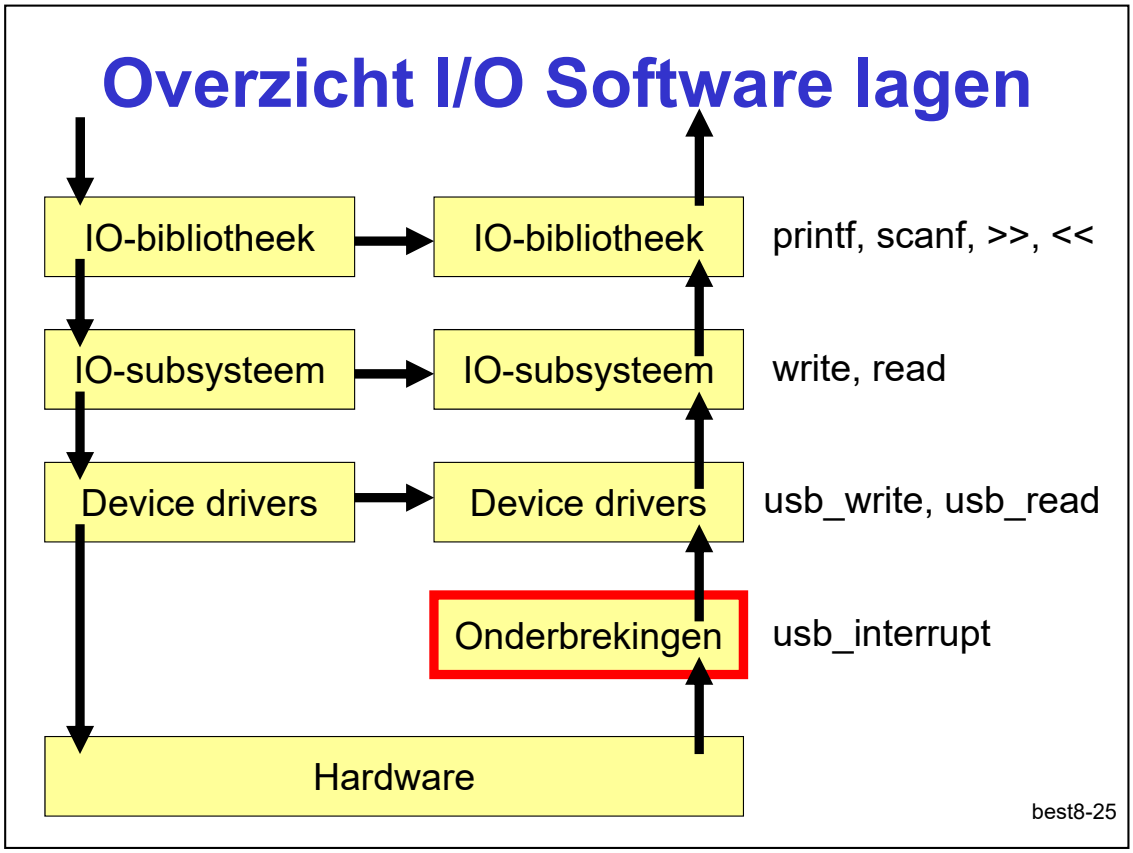
Apparaatbeheer

- Beheer ondeelbare apparaten
  - Via `open()` en `close()` op speciale bestanden. Indien apparaat in gebruik, faalt de `open()`.
  - Soms zijn er speciale primitieven die blokkeren indien het apparaat in gebruik is (allocate/deallocate).

best8-24

Het IO-subsysteem houdt de boekhouding bij van de randapparaten: welk proces heeft welk randapparaat in gebruik, kan een randapparaat door meer dan één proces gebruikt worden, welke buffering hoort bij welk randapparaat, soms ook impasse-informatie, enz.

Randapparaten kunnen exclusief gereserveerd worden door een proces door ze op de gepaste wijze te alloceren. Sommige besturingssystemen (zoals VMS) beschikken over de commando's 'allocate' en 'deallocate' die toelaten om een randapparaat exclusief te alloceren. Unix stelt randapparaten voor als speciale bestanden die met de gewone 'open' en 'close' primtieven kunnen geopend en gesloten worden. Net zoals bestanden al dan niet kunnen gedeeld worden, kunnen ook randapparaten al dan niet gedeeld worden door verschillende processen.



De onderbrekingen worden in de context van device drivers meestal gebruikt om geblokkeerde device drivers opnieuw vrij te geven. Nadat een device driver het randapparaat aangestuurd heeft, wacht het op een antwoord door te blokkeren op een signaal. De onderbrekingsroutine hoeft in dit geval niets anders te doen dan het signaal te genereren. In een besturingssysteem probeert men de onderbrekingsroutines zo kort mogelijk te houden omdat lange onderbrekingsroutines verstorend kunnen zijn. Ze treden asynchroon met de rest van het systeem op, en ze zijn niet onderhevig aan de controle van de planner. Daarom is het beter om de taak van de onderbrekingsroutine te beperken tot het geven van signalen, en de vrijgekomen device driver dan over te dragen aan de werking van de procesplanner.

Device drivers

Er zijn essentieel vier types

- Blokapparaten
- Tekenapparaten
- Netwerkapparaten
- Andere (klokken, timers,…)

best8-26

Device drivers kunnen ingedeeld worden in vier types, die hierna besproken worden.

Blokapparaten

- Doorgaans gebruikt bij opslagapparaten: bv. schijf
- Leest/schrijft per blok (bv.512 bytes)
- Adresseerbaar, kan elk blok apart manipuleren (random access)
- Operatie: ioctl voor extra commando buiten read, write, seek.
- Vaak gelinkt met een bestandssysteem
- Blokken kunnen in het geheugen afgebeeld worden (memory mapped files).

best8-27

Blokapparaten worden vooral gebruikt bij opslagapparaten en zijn daarom vaak gelinkt met een bestandssysteem.

Tekenapparaten

- Leest/schrijft stroom van bytes
- Niet adresseerbaar (sequentiële toegang)
- Operaties: read, write
- Soms interactieve editeermogelijkheden
- Vb: toetsenbord, muis, seriële poort

best8-28

Tekenapparaten worden vaak gebruikt voor de tragere randapparaten die teken per teken communiceren zoals toetsenbord, printer, muis, …

Netwerkapparaten

- Verschillen voldoende van blok- en tekenapparaten om een eigen interface te hebben.
- Unix en Windows voorzien in een socketinterface
  - Scheiding tussen protocol en werking van het netwerk
  - Biedt een select functie aan om tegelijk verschillende sockets in het oog te kunnen houden.
- Vb: sockets, maar ook pipes, FIFOs, streams, queues, mailboxes

best8-29

Andere: klokken en timers

- Basisfuncties: huidige tijd, verstreken tijd, timer
- Programmeerbare intervaltimer wordt gebruikt b.v. om een tijdskwantum af te bakenen, om processen te alarmeren op een afgesproken tijdstip, als waakhond, of om te profileren.
- De aansturing van de timers gebeurt doorgaans via ioctl (bij gebrek aan beter)

best8-30

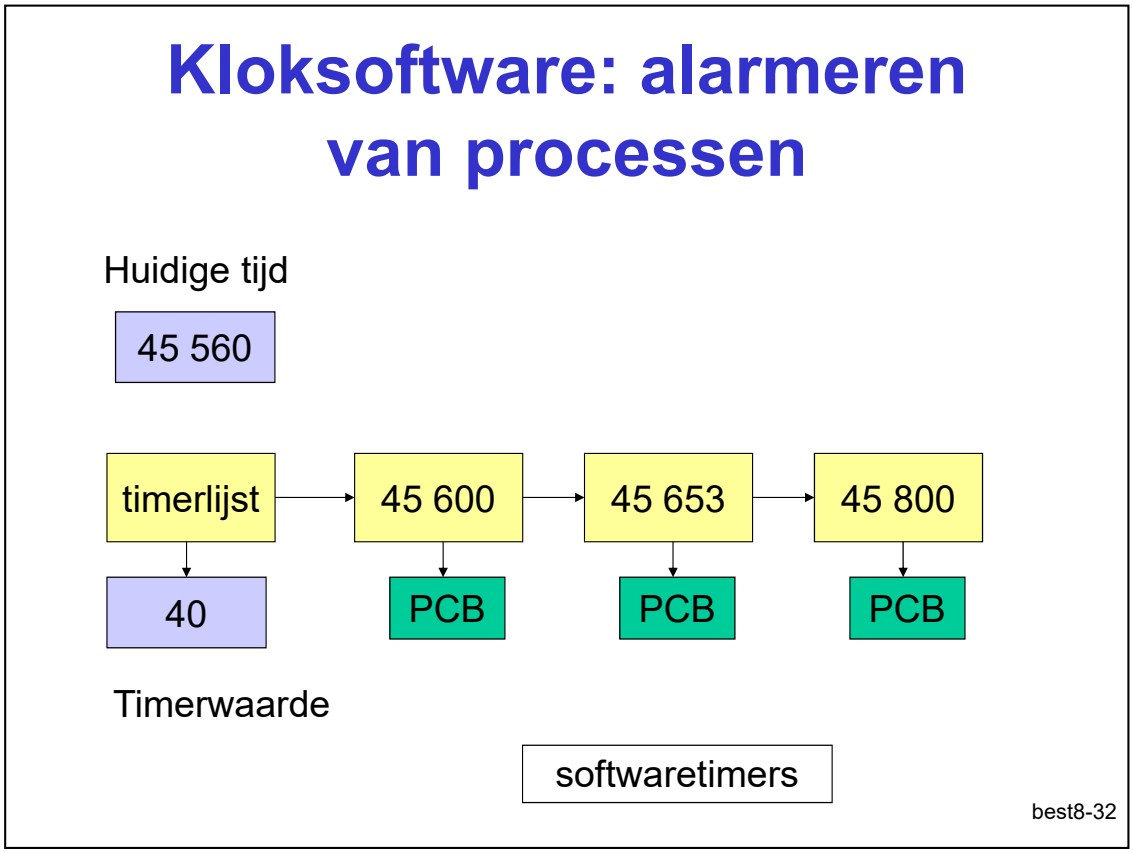
Het bijhouden van de tijd is cruciaal in een besturingssysteem. De tijd wordt gebruikt om b.v. een tijdstempel te geven aan de gebeurtenissen die zich voordoen in een besturingssysteem (b.v. creatietijdstip van een bestand). Verder hebben we de tijd ook nodig om aan preëmptieve planning te kunnen doen. De timer moet een proces in uitvoering op het einde van een tijdskwantum kunnen onderbreken. Verder moet omwille van de facturering ook het CVE-gebruik van de verschillende processen bijgehouden worden.

Tenslotte zijn er nog de tijdsgebonden processen. Sommige processen moeten op gestelde tijdstippen een taak uitvoeren (b.v. om de 50 ms een frame tonen), of wensen na een bepaalde tijd onderbroken te worden (b.v. indien er na 5 seconden nog steeds geen netwerkverbinding kon gemaakt worden; een zgn. waakhond of watchdog timer). Verder worden timers ook gebruikt om te profileren, programma's te observeren of om statistieken over een uitvoering te verzamelen.

Kloksoftware: tijd bijhouden

- Tijd sinds 1 januari 1970 (Unix) 1980 (Windows)
- Synchronisatie ≠ computers over netwerk: NTP (network time protocol)
- 60 kloktikken/seconde: 32 bit gaat maar voor 2 jaar → meer bits nodig.

best8-31



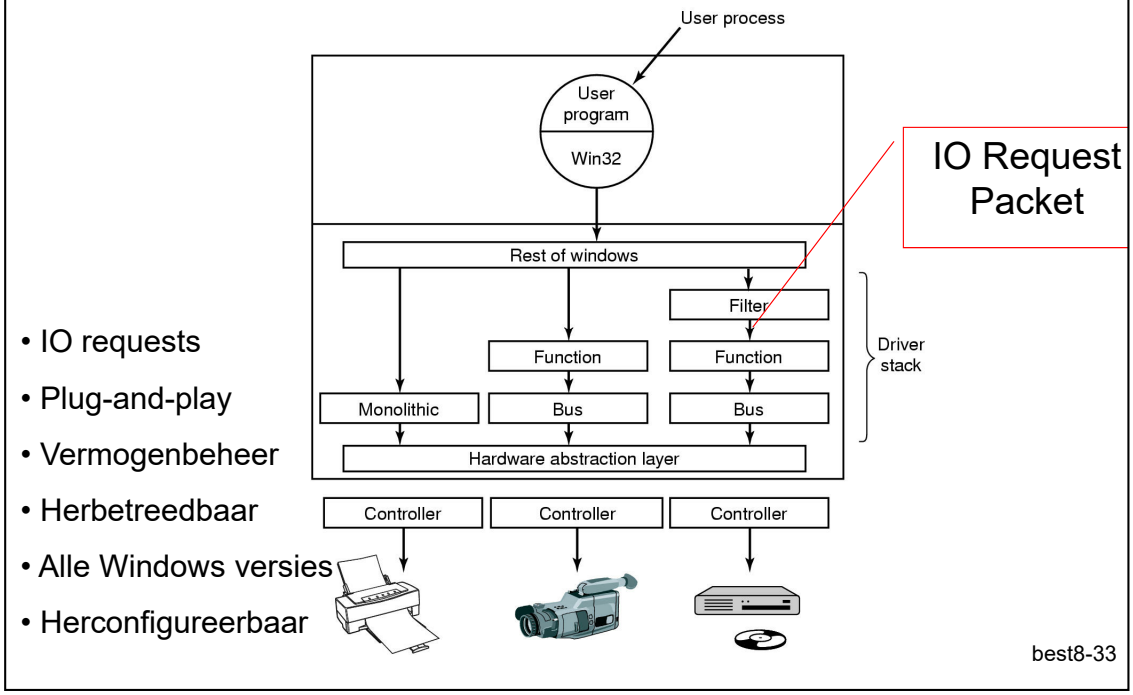
Ideaal zou er 1 hardwaretimer zijn per alarm. Doorgaans zijn er maar een heel klein aantal hardwaretimers.

Processen die wensen gealarmeerd te worden op een zeker tijdstip, kunnen zich echter laten opnemen in een timerlijst. Deze timerlijst houdt de procescontroleblokken van de te alarmeren processen bij in volgorde van alarmeringstijdstip. Eén timer wordt ingesteld om af te lopen op het ogenblik dat het volgende proces moet gealarmeerd worden. Vervolgens wordt de timer ingesteld om af te lopen op het proces dat daarop volgt (hier zal dit 53 zijn, en naderhand 147). Op die manier wordt er slechts 1 onderbreking gegenereerd per proces dat moet gealarmeerd worden. Indien het gealarmeerde proces een hoge prioriteit heeft kan het na het alarmeren meteen aan de slag.

Om het aantal onderbrekingen nog verder terug te dringen, kan men er ook voor kiezen om gebruik te maken van zgn. softwaretimers. Deze timers vereisen geen extra timeronderbrekingen, maar garanderen ook geen signaal op het precieze afgesproken tijdstip. Ze controleren bij het aflopen van een systeemoproep, of ter gelegenheid van de tussenkomst van de procesplanner, of er een timer is afgelopen. In die gevallen zit men met de overhead van de extra controle bij elke systeemoproep of tussenkomst van de procesplanner, maar anderzijds heeft men geen extra hardwaretimer nodig. Indien het wachten op het aflopen van het tijdsquantum te lang is, kan men er ook voor opteren om een hardwaretimer in te stellen op b.v. 1 ms zodat een alarm nooit meer dan 1 ms vertraagd kan worden.

Het bijhouden van de tijd via een timer is niet zo’n goed idee omdat er bij vertraging van de onderbrekingen na verloop van tijd drift zal ontstaan. Het is beter om hiervoor een gededihardwareteller te gebruiken die gewoon kan uitgelezen worden als de tijd opgevraagd wordt.

Windows Device Driver Model



Het windows driver model is een complexe aangelegenheid. Om conform te zijn moet een Windows driver aan de volgende voorwaarden voldoen: (i) hij moet in staat zijn om (welomschreven) IO request pakketten te verwerken, (ii) hij moet plug-and-play ondersteunen, wat wil zeggen dat hij automatisch moet kunnen geactiveerd en gedeactiveerd worden naargelang de aanwezigheid van een randapparaat, (iii) hij moet vermogenbeheer ondersteunen – indien de kern vraagt om vermogen te besparen moet hij het randapparaat in de vermogenspaarstand kunnen brengen, (iv) hij moet herbetreedbaar zijn (voor multiprocessorsystemen), (v) hij moet met alle versies van Windows werken, en tenslotte (vi) moet hij herconfigureerbaar zijn wat wil zeggen dat er geen hardgecodeerde IO-adressen of onderbrekingsnummers mogen in voorkomen.

Per actieve device driver wordt er een driverobject in de kern gecreëerd met pointers naar de verschillende driverroutines. Dit driverobject creëert dan zoveel deviceobjecten als er nodig zijn (1 driver kan immers verschillende randapparaten aansturen). Deze deviceobjecten worden gegroepeerd in de device directory '\??'. Deze directory is een interne systeemdirectory die niet zichtbaar is in het bestandensysteem.

Drivers kunnen individueel werken (zoals voor de printer in de afbeelding), maar ook gestapeld. De drivers zullen dan onderling IO Request Packets uitwisselen. Het is gebruikelijk om een bus management driver te hebben, met daarboven de driver voor het randapparaat, eventeel voorafgegaan door een filter driver (die b.v. compressie uitvoert).

Unix Special Files

In **/dev** directory

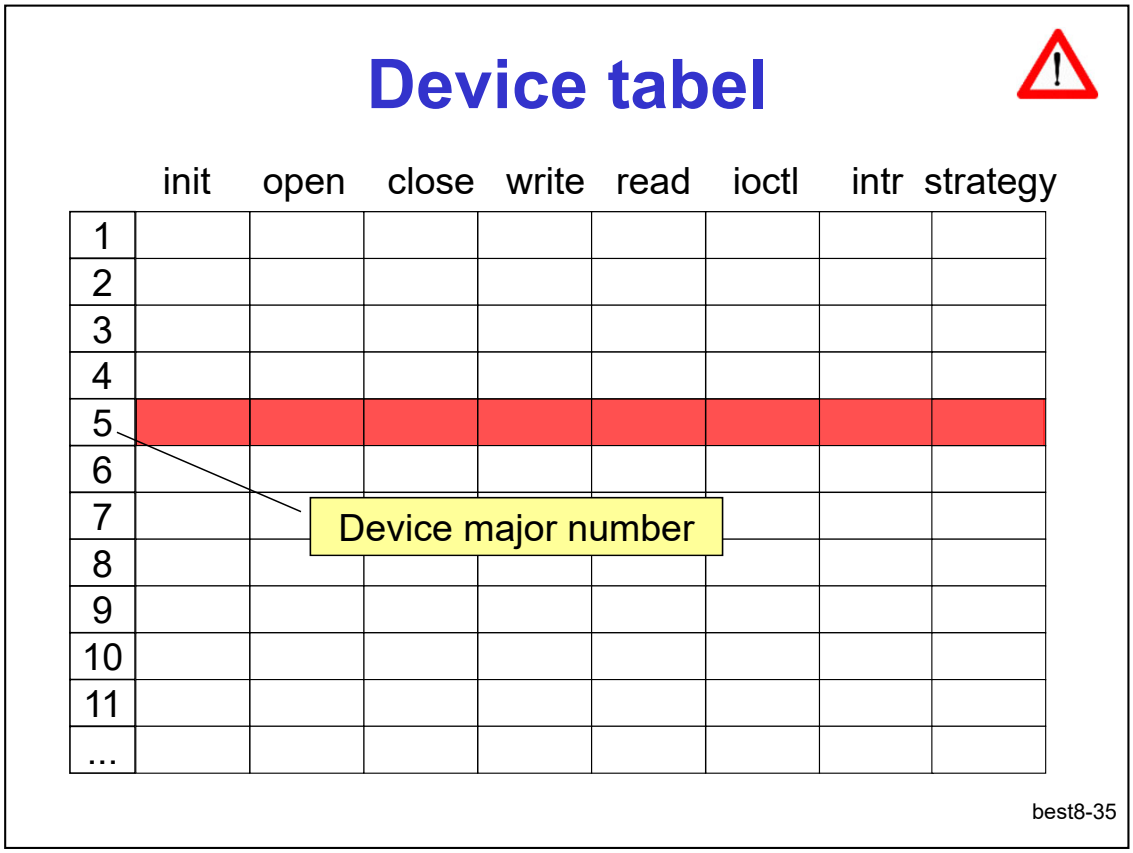
```
mknod naam c major minor
mknod naam b major minor
mknod naam p
```

driver randapparaat

best8-34

Unix heeft een verschillend drivermodel. De drivers krijgen allemaal een plaatsje in het bestandssysteem als special files. Deze bestanden hebben geen geassocieerde gegevensblokken, maar dienen enkel maar om toe te laten dat de bestandsoperaties read en write kunnen toegepast worden op randapparaten. Deze bestanden worden aangemaakt met het commando mknod en dienen naast het type (b=blokapparaat of c=tekenapparaat) ook nog een major en minor device number te krijgen. Het major number identificeert de driver, en het minor number identificeert het randapparaat (opnieuw kan 1 driver verschillende randapparaten besturen; het minor number dient om de verschillende randapparaten te kunnen identificeren).

De code van de device driver kan ofwel hard in de kern van het besturingssysteem gelinkt worden, of dynamisch ingeladen worden met het commando insmod.



Een Unix device driver moet een vastomschreven aantal routines aanbieden. De adressen van deze routines worden opgenomen in de devicetabel. Als een driver in de kern gelinkt wordt, gebeurt dit statisch. Indien de driver dynamisch ingeladen wordt bevat de initialisatie van de driver de nodige instructies om de driver op de juiste plaats in de tabel te laden.

Routines die zeker moeten aangeboden worden zijn: init: het initialiseren van de driver, open, close, write, read: worden opgeroepen bij de gelijknamige operatie op het bestand of op het open bestand. De ioctl is de vuilbak van de driverroutines. Het krijgt een datagebied waarvan de layout en de semantiek volledig door de driver mag gedefinieerd worden. Via deze weg kan men onbeperkt met het randapparaat communiceren. Intr is de onderbrekingsroutine die in sommige gevallen kan opgeroepen worden. Strategy is de planningsroutine (zie io-planning) die bepaalt welke aanvraag (van alle gebufferde aanvragen; write en read zullen in dit geval de aanvragen gewoon bufferen) er als eerste mag uitgevoerd worden.

Voorbeeld

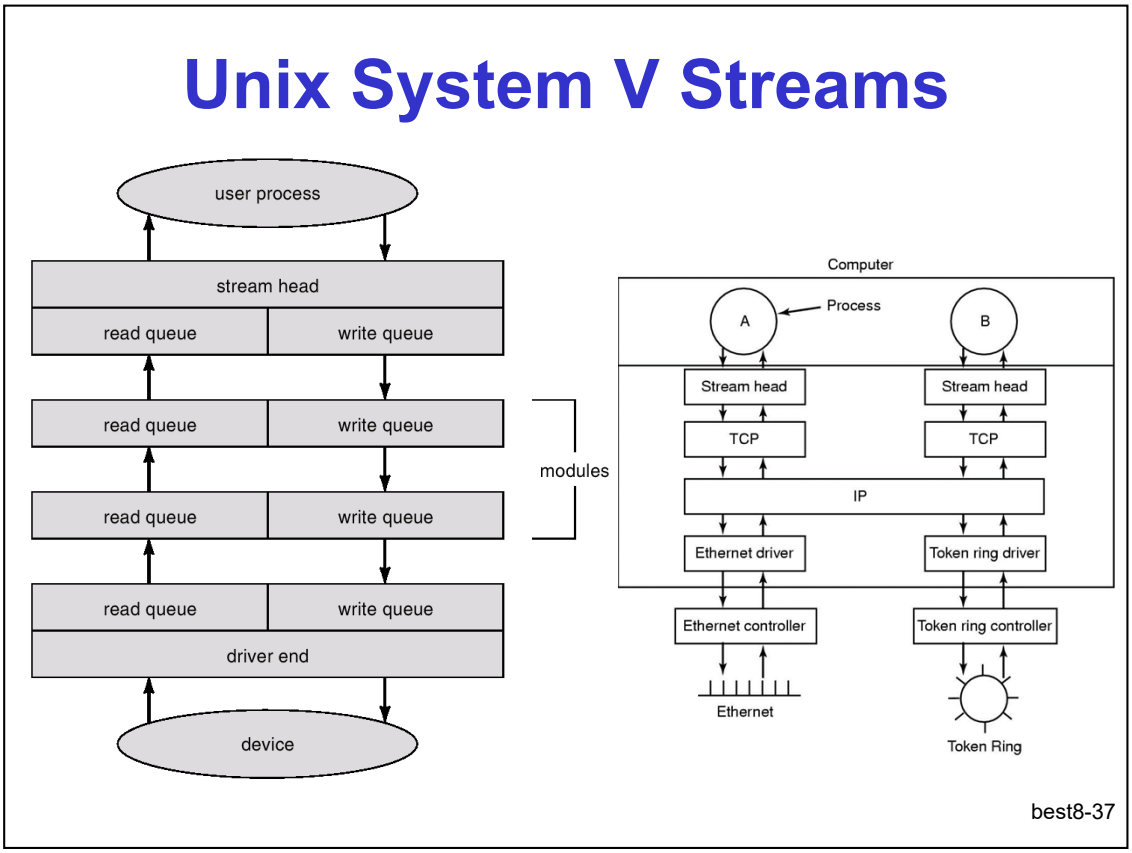
```
#include <fcntl.h>
#include <stdio.h>
#define DEVICE ”/dev/ttyp5”

main()
{
  int handle = open(DEVICE,O_WRONLY);
  if (handle != -1) {
    write(handle, ”Hallo\n”, 6);
    close(handle);
  } else
    fprintf(stderr, ”Cannot open %s\n”,DEVICE);
}
```

best8-36

Een Unix tekenapparaat kan als een traditioneel bestand gebruikt worden. Ter gelegenheid van het openen van het bestand zal het bestandssysteem nagaan of het een speciaal bestand betreft. De globale bestandentabel zal nu wijzen naar de device driver i.p.v. naar een inode. Vervolgens worden de speciale lees- en schrijfroutines gebruikt in de plaats van de traditionele lees- en schrijfroutines van het bestandssysteem. De applicatie heeft totaal geen weet van het feit dat het schrijft naar een randapparaat. De semantiek van de schrijfoperatie is precies dezelfde als deze voor een gegevensbestand.

Doordat een speciaal bestand een directoryelement heeft, kan het ook gebruik maken van de bestaande afschermingsmethode om ervoor te zorgen dat het gebruik van bepaalde randapparaten beperkt blijft tot bepaalde gebruikers. Het exclusief gebruik van een device driver wordt geregeld in de open-routine. De device driver houdt intern bij hoeveel keer hij geopend werd. Indien men niet wenst dat een device driver door meer dan één proces simultaan gebruikt wordt, kan men in de open routine een foutcode teruggeven.



Ook in Unix kunnen device drivers gestapeld worden. In System V spreekt men dan van streams. Een stream is een full duplex communicatiekanaal tussen een gebruikersproces en een randapparaat. Een stream bestaat uit een **stream head interface** en een **driver end interface**. Daartussen kunnen er zich een aantal stream modules bevinden. Elke module heeft een lees- en schrijfwachtlijn waarover boodschappen uitgewisseld worden tussen de verschillende modules. Streams worden o.a. gebruikt om een netwerkstapel te implementeren.

Prestaties

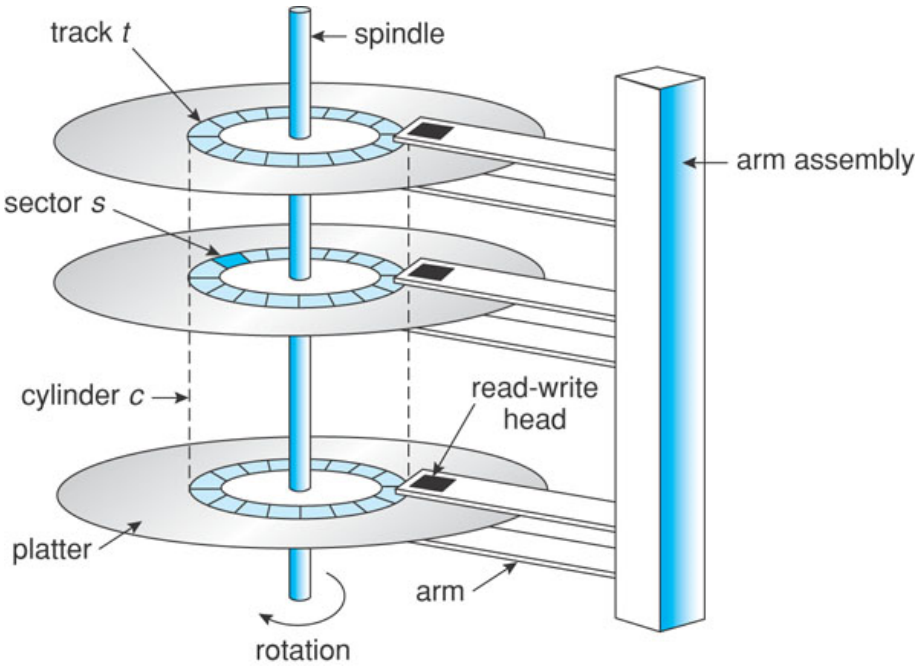
- IO is een belangrijke factor in de prestatie van een systeem (zeker voor b.v. databanken en webservers).
  - De driver moet zo efficiënt mogelijk gecodeerd worden, de aanwezige hardware moet maximaal geëxploiteerd worden (b.v. DMA)
  - Het aantal contextwisselingen (incl. onderbrekingen) moet tot het minimum beperkt worden
  - Het kopiëren van data moet zoveel mogelijk beperkt worden

Overzicht

- Situering
- IO-systeem
  - IO-bibliotheek
  - IO-subsysteem
  - Device drivers
  - Onderbrekingsroutines
- Schijfbeheer
  - Opbouw
  - Schijfplanning
  - RAID
  - Opslagsystemen

best8-39

Structuur harde schijf



best8-40

Formattering

1. **Fysieke formattering**: verkaveling van de schijfoppervlakte in sporen en sectoren.

preamble: synchronisatiepatroon, ID

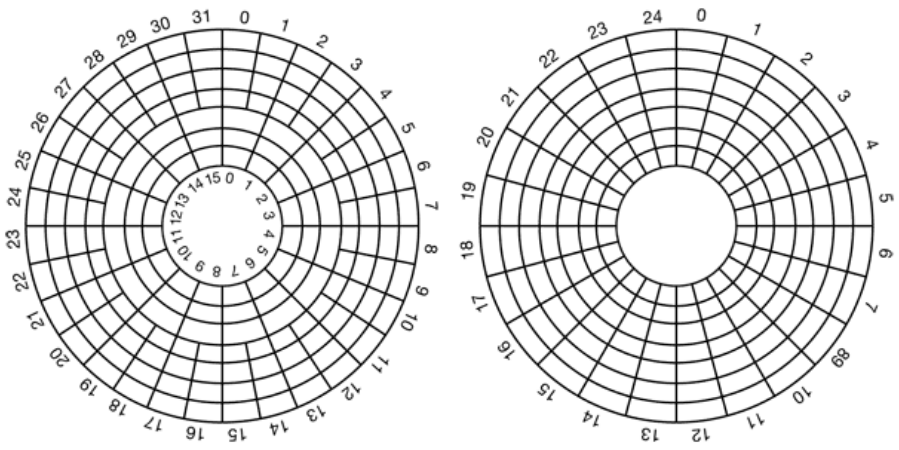
ECC (paar % overhead)

Vrij: veiligheidsmarge

2. **Partitioneren**: aanbrengen van partities + master boot record.
3. **Logische formattering**: aanbrengen in een partitie van de boot sector en leeg bestandssysteem

best8-41

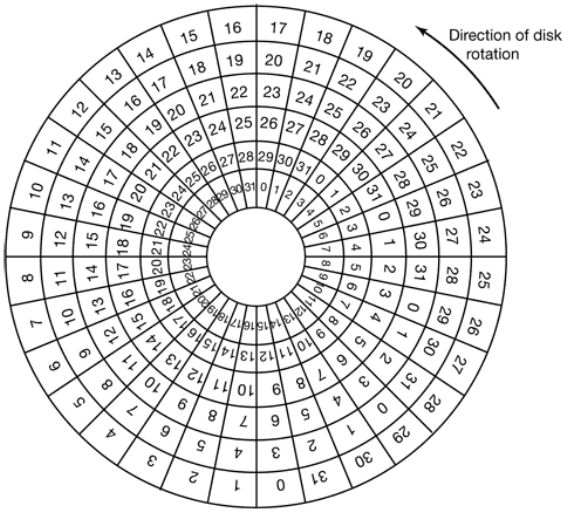
Schijfgeometrie



best8-42

Bij grote schijven zal het aantal sectoren per spoor niet steeds gelijk zijn. De omtrek van de binnenste sporen laat niet toe om evenveel sectoren te plaatsen als de omtrek van de buitenste sporen. Omdat sommige (oudere) besturingssystemen ervan uitgaan dat een schijf een vast aantal sectoren per spoor heeft zal de schijf soms de illusie wekken dat dit zo is door een virtuele geometrie aan te bieden. Tegenwoordig wordt een schijf gemodelleerd als een grote eendimensionale array van sectoren. Deze sectoren worden sequentieel vanaf 0 genummerd. Daardoor is de fysieke geometrie van de schijf eigenlijk minder belangrijk. Het is tegenwoordig aan de schijfregelaar om optimaal gebruik te maken van de fysieke geometrie van de schijf. De gebruiker van de schijf kan in veel gevallen zelfs niet meer te weten komen wat de fysieke geometrie is.

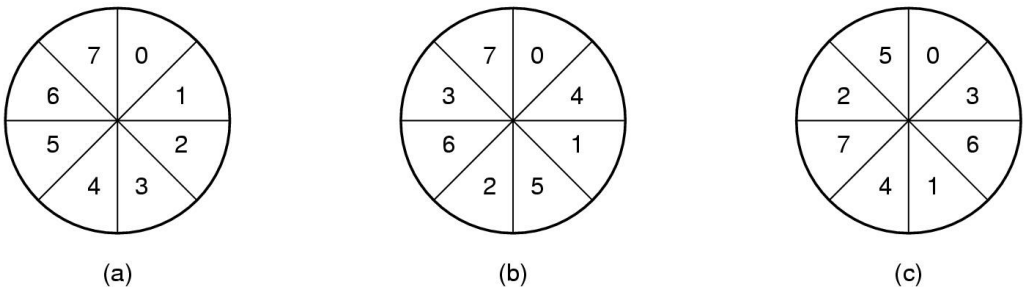
Cilinderverschuiving (cylinder skew)



best8-43

Cilinderverschuiving wordt toegepast om bij het verspringen van een spoor ervoor te zorgen dat men geen hele omwenteling moet wachten om de eerste sector te kunnen lezen. Indien het binnenste spoor het 0-de spoor is, dan hebben we hier te maken met een cilinderverschuiving van 3. Op die manier kan men ervoor zorgen dat het verspringen van spoor gepaard gaat men een minimale verstoring van de gegevensstroom.

Indeling: interleaving

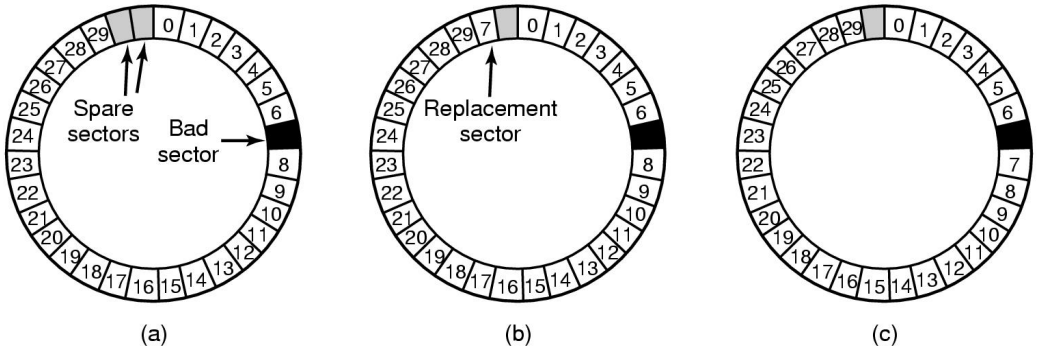


best8-44

Interleaving is een techniek die gebruikt wordt indien de gegevens sneller van de schijf kunnen gelezen worden, dan ze b.v. kunnen verstuurd worden. Door de sectoren niet sequentieel na elkaar op het spoor te zetten, maar er steeds een paar andere sectoren tussen te zetten kan de communicatiesnelheid verlaagd worden. Bij gewone interleaving reduceert men de bandbreedte met een factor twee. Bij dubbele interleaving met een factor 3.

Bij goed gebufferde schijven is interleaving overbodig omdat deze schijven meteen een heel spoor bufferen in de schijfregelaar van waaruit het aan een lagere snelheid naar het geheugen kan gestuurd worden.

Foutafhandeling: Magnetische fouten

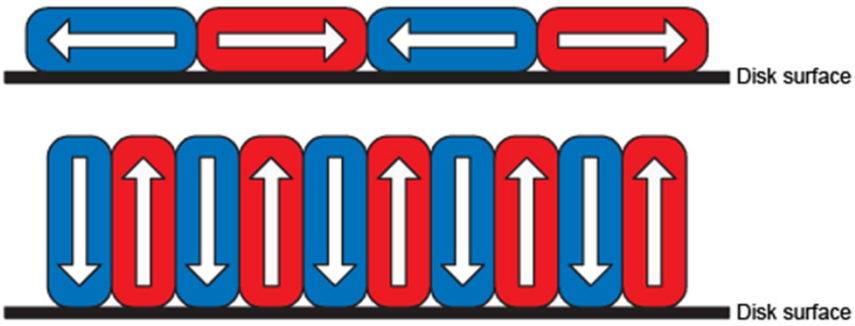


Sector forwarding Sector slipping

best8-45

Bij het fysiek formatteren valt het voor dat bepaalde sectoren onbruikbaar zijn door onzuiverheden in het oppervlak. Deze sectoren moeten dan vervangen worden. Daarvoor kan men per spoor een paar extra sectoren ongebruikt laten. Een defecte sector kan op twee manieren vervangen worden, ofwel door de defecte sector af te beelden op een vrije sector, ofwel door alle sectoren 1 plaats op te schuiven en op die manier plaats te maken naast de defecte sector. Bij het fysiek formatteren is het gemakkelijk om oplossing (c) te implementeren. Om defecte sectoren te vervangen die na het formatteren ontstaan, is het vaak gemakkelijker om voor oplossing (b) te kiezen.

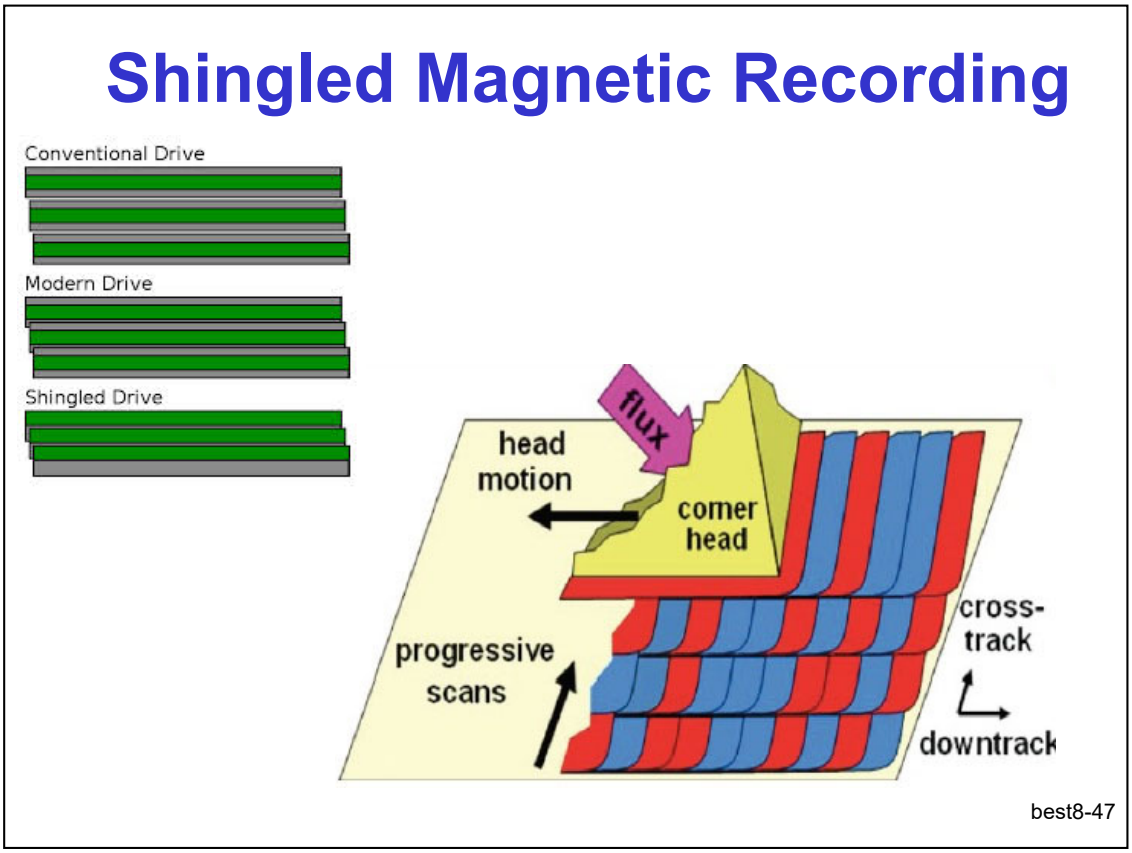
LMR vs. PMR



75nm x 14 nm

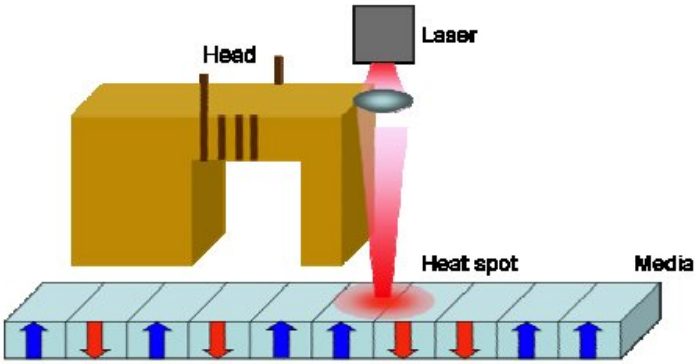
best8-46

Sinds 2005 heeft men de densiteit van de magnetische opslag kunnen verhogen door het oppervlak niet langer te magnetiseren in de lengterichting van het spoor (LMR = longitudinal magnetic recording), maar loodrecht op het oppervlak (PMR = perpendicular magnetic recording). Hierdoor kon de densiteit met ongeveer een factor drie verhoogd worden. De grootte van de bits bedraagt ongeveer 75 nm (spoorbreedte) bij 14 nm in de lengterichting (ongeveer even groot als een transistor). Een bit bestaat uit ongeveer 20-30 magnetische korreltjes van 8nm diameter. Deze korreltjes kunnen niet veel kleiner meer gemaakt worden omdat ze anders spontaan van richting zullen omklappen bij te hoge temperatuur (als gevolg van superparamagnetisme). Het aantal korreltjes kan ook niet verkleind worden omdat anders de signaal/ruisverhouding te klein wordt. De theoretische limiet van PMR is ongeveer 1 Tb/in2. In 2014 waren er reeds schijven te koop met die densiteit.



Als de bits niet kleiner meer gemaakt kunnen worden, kan de ruimte tussen de sporen wel nog verkleind worden. Het idee is dat de verschillende sporen gedeeltelijk overlappen net zoals dakpannen of schalies. De bits zijn nog groot genoeg om te lezen, maar eenmaal geschreven kunnen ze niet meer overschreven worden zonder de omliggende bits ook te overschrijven. Dergelijke schrijven bestaan uit stroken van sporen die sequentieel kunnen geschreven worden, en random access kunnen gelezen worden. Als er iets moet veranderd worden moet de volledige strook telkens opnieuw herschreven worden. Dit levert 25% meer capaciteit op. De prijs is een verlies aan flexibiliteit.

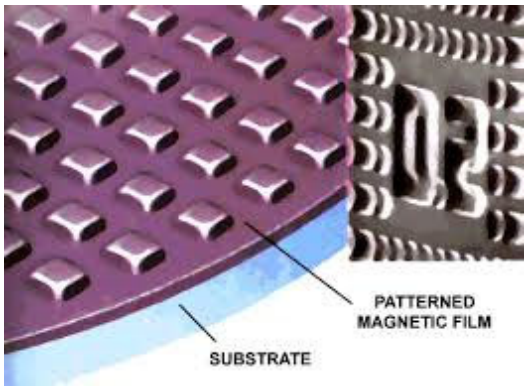
Heat assisted magnetic recording

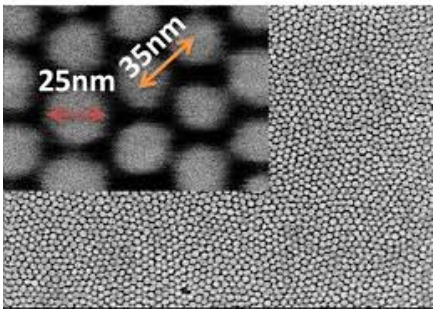


best8-48

Als we de densiteit nog hoger willen maken, dan moet er gebruik gemaakt worden van magnetische materialen die minder gemakkelijk spontaan van toestand veranderen. Dat betekent echter dat de velden die nodig zijn om een toestandsverandering tot stand te brengen ook veel groter moeten zijn, waardoor ook naburige bits van toestand kunnen veranderen. Een alternatief is om de bits tijdelijk op te warmen waardoor ze gemakkelijker van toestand kunnen veranderen. Dit vereist echter wel zeer geavanceerde technologische oplossingen waarbij een laserstraal de bits heel kortstondig moet opwarmen om een toestandsverandering mogelijk te maken (tot 400°C). Het duurt ongeveer om 1ns om de bits terug af te koelen.

Bit Patterned Media

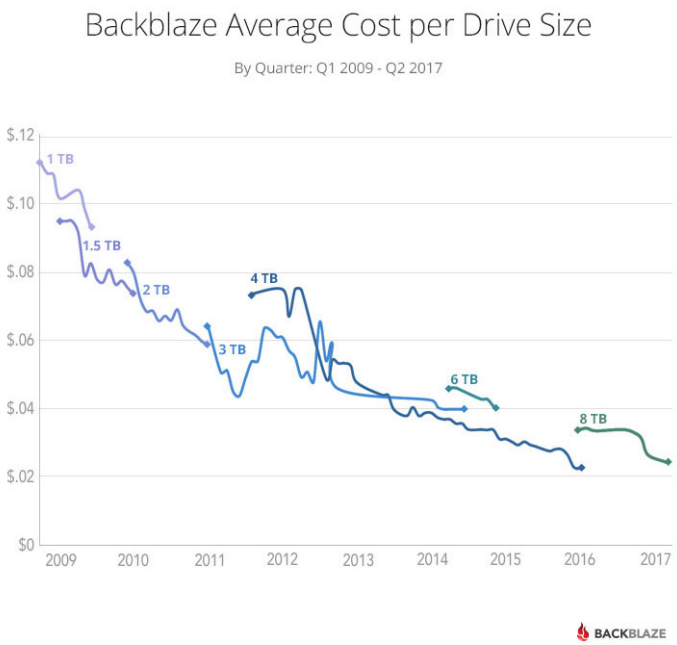




best8-49

Verder kan er ook nog gebruik gemaakt worden van zogenaamde bit patterned media, dit zijn eilanden van magnetisch materiaal ingebed in een niet-magnetisch substraat. In deze opstelling kunnen de magnetische eilanden bestaan uit één magnetische korrel. Om een densiteit van 1 TB/in2 te bereiken mogen de eilanden niet groter zijn dan 12nm. Dit is kleiner dan wat de hedendaagse lithografische systemen mogelijk maken. Men hoopt oplossingen te vinden waarbij de media zelf-organiserend zijn (en er dus geen geavanceerde lithografische systemen aan te pas komen). In combinatie met een warmtebron kunnen op die manier een theoretische densiteit van 50 Tb/in2 bereikt worden. In 2014 is het echter nog niet klaar voor de markt.

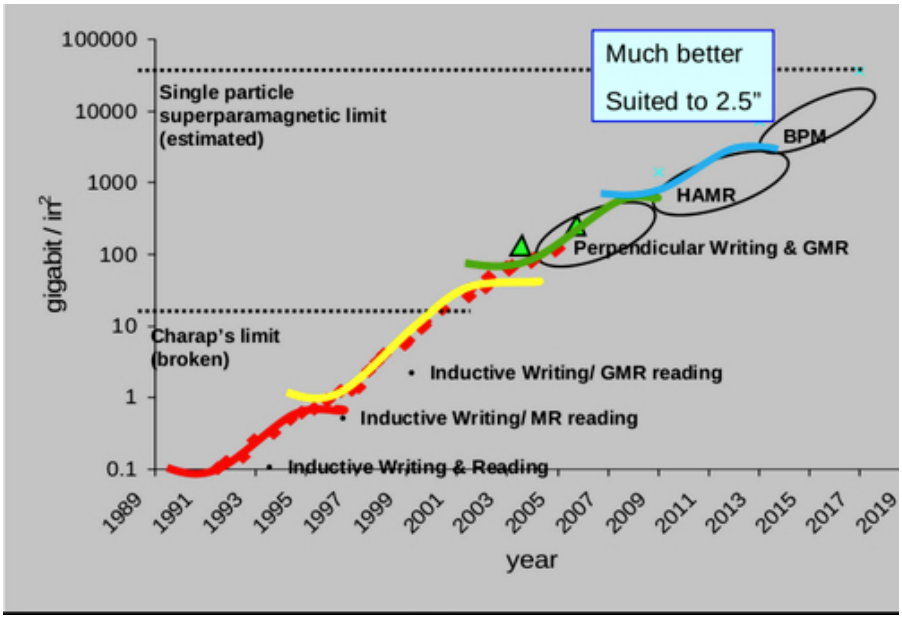
Disk cost-per-GB



best8-50

Tot 2010 daalde de prijs per GB ongeveer samen met de toename van de densiteit. Vanaf 2010 is er een vertraging opgetreden en daalt de prijs minder snel. De verdubbeling van de schijfcapaciteit vertraagt ook. Het heeft duidelijk langer geduurd om van 4 TB naar 8TB te gaan dan bv van 2 TB naar 4TB. De plotse stijging van de prijs in 2012 heeft te maken met grote overstromingen in Thailand waar de meeste schijven geproduceerd worden.

Evolutie opslagdensiteit (Wet van Kryder)



best8-51

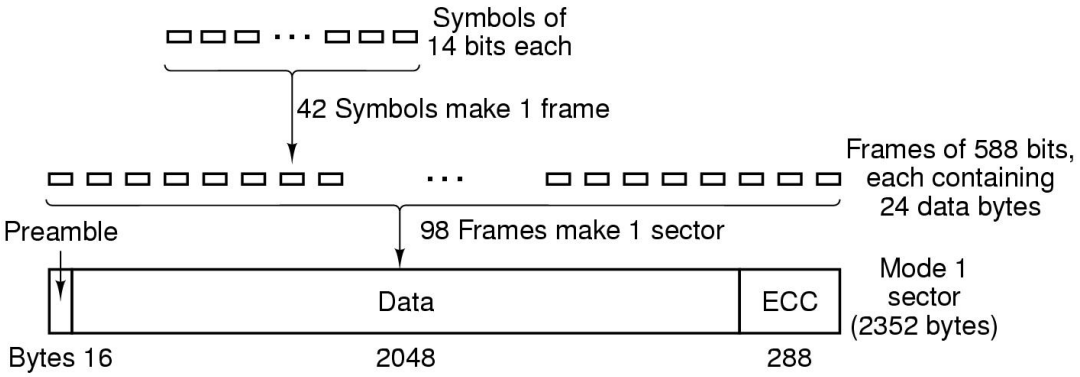
Deze grafiek toont de opvolging van de diverse technologieën.

Flash

- Veel duurder dan harde schijven ($0,50/GiB ipv $0,05/GiB)
- Onvoldoende productiecapaciteit (fabs)
- Betrouwbaarheid: 100 000 cycli voor SLC (cellen met 1 bit) en 10 000 cycli voor MLC (50 nm technologie; voor <20nm: 1000-3000 cycli)
- Retentie: 10-20 jaar voor nieuwe cellen. Na 10 000 cycli: 6-12 maand bij 50°C. Ook lezen beschadigt de cel.

best8-52

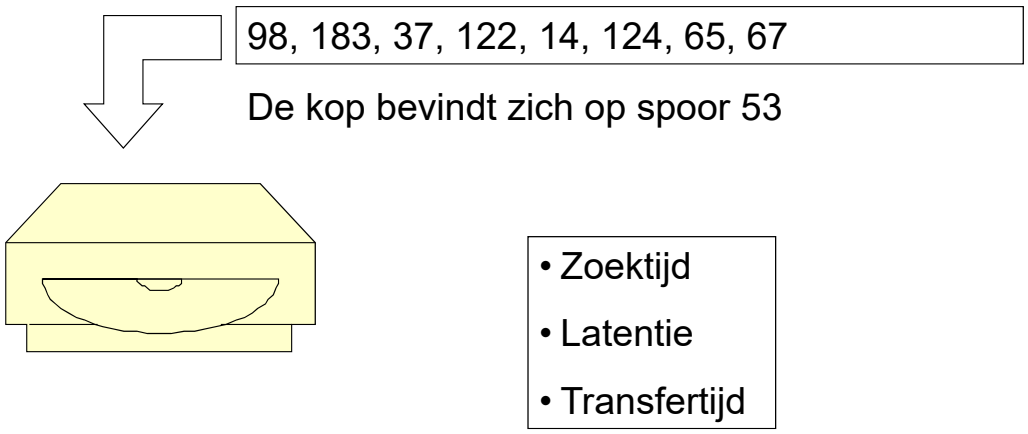
CD-ROM



- Muziek: symbolen en frames; 2x foutcorrectie
- Frame past niet echt bij data (588 bit = 73,5 B)
- Sector: 98 frames
  - bevat 2048 data bytes + 3e foutcontrole (ECC)
- Sector: totale omvang 7203 bytes, zeer veel overhead (351,7%)

best8-53

Schijfplanning



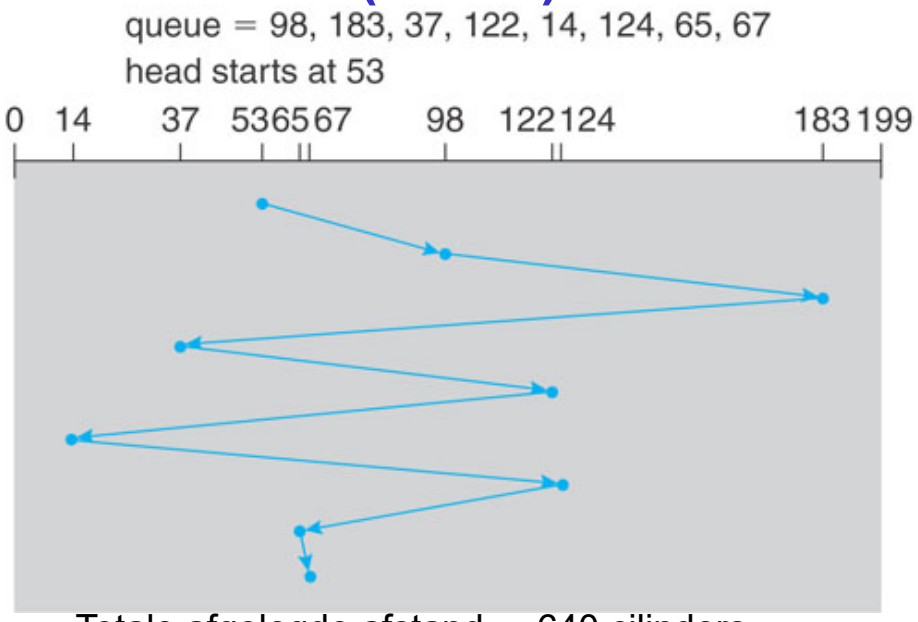
best8-54

Op de fysieke schijf heeft een sector een driedimensionaal adres (cilinder, plaat, sector).

Gezien de werking van een schijf zal de tijd nodig om een bepaalde sector te lezen bestaan uit drie componenten (i) de tijd nodig om de kop boven een bepaalde cilinder of spoor te positioneren (zoektijd), (ii) de tijd totdat de gewenste sector zich onder de kop bevindt (latentie), en (iii) de tijd nodig om de gegevens fysiek te transfereren (transfertijd).

Indien er verschillende schijfoperaties staan te wachten op de schijf kunnen we de prestaties van de schijf soms verbeteren door de volgorde ervan te veranderen zodat het toegangspatroon naar de schijf geoptimaliseerd wordt (dit probleem is zeer verwant met dat van de handelsreiziger). Deze strategieën zullen enkel een verschil opleveren als de schijf effectief zwaar belast wordt (en er dus keuze is). Op systemen met één gebruiker zoals PCs komen ze vaak neer op een sequentiële afwerking van de gegenereerde schijfoperaties. Schijfstrategieën kunnen door het besturingssysteem geïmplementeerd worden (rekening houdend met de kenmerken van het besturingssysteem), of ook door de schijfregelaar (rekening houdend met de kenmerken van de schijf). In de nu volgende bespreking wordt verondersteld dat de volgorde van sporen waarnaar gelezen of geschreven moet worden is: 98, 183, 37, 122, 14, 124, 65, 67 is en dat de kop initieel op positie 53 staat.

First-Come First-Served (FCFS)

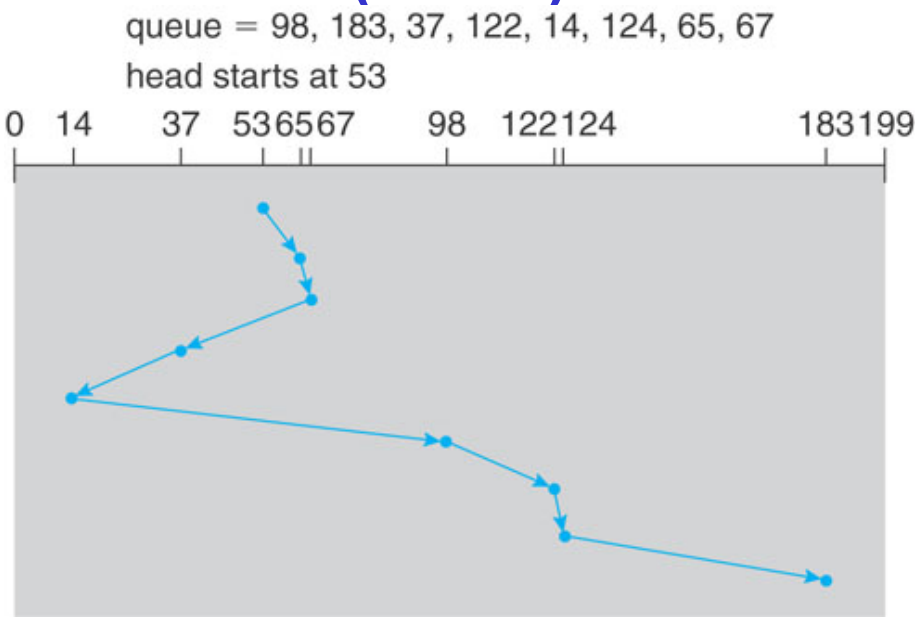


Totale afgelegde afstand = 640 cilinders.

best8-55

De First-Come-First-Served strategie is de eenvoudigste, en bovendien is ze eerlijk hetgeen wil zeggen dat schijfoperaties niet oneindig lang kunnen uitgesteld worden. Ze kan echter niet de beste prestatie van de schijf garanderen omdat opeenvolgende schijfoperaties kunnen plaatsvinden op totaal verschillende delen van de schijf waardoor er dus veel tijd verloren gaat aan het zoeken van de sporen.

Shortest Seek Time First (SSTF)

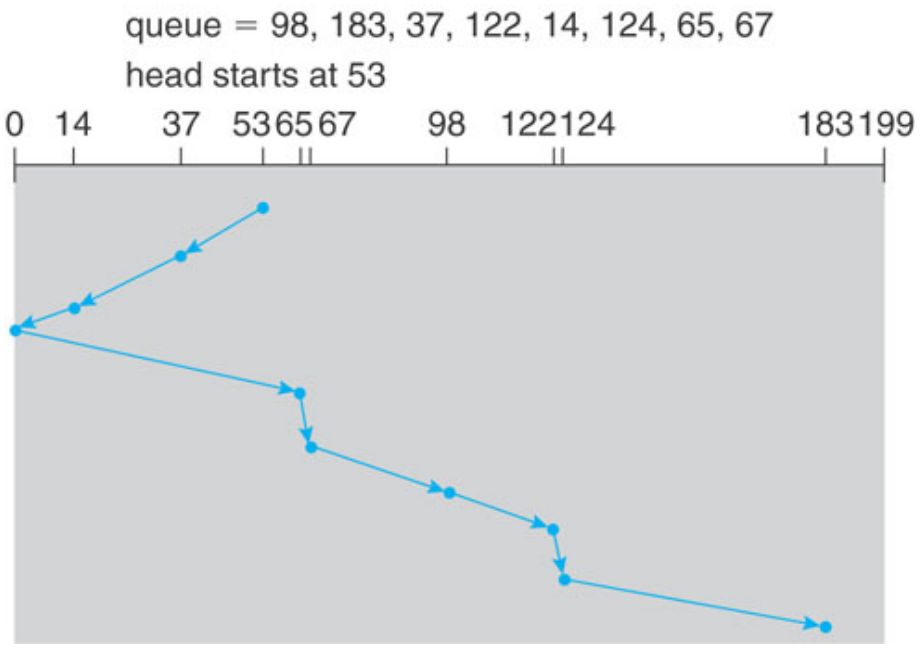


Totale afgelegde afstand = 236 cilinders

best8-56

Om te vermijden dat de lees/schrijfkop zich wild heen en weer beweegt over de schijf indien FCFS gebruikt wordt kan men de voorkeur geven aan die schijfoperaties die in de onmiddellijke nabijheid van de huidige koppositie moeten doorgevoerd worden. Het voordeel van deze strategie is dat de zoektijden hierdoor zullen afnemen, maar anderzijds is deze strategie niet langer gegarandeerd eerlijk, omdat bij een zware belasting er blijvend aanvragen kunnen binnenkomen voor een bepaald deel van de schijf waardoor verder afgelegen delen van de schijf niet bediend worden. In tegenstelling met SJF is SSTF geen optimaal algoritme. In het voorbeeld zou het bijvoorbeeld efficiënter zijn om eerst naar cilinder 37 te gaan en dan pas SSTF toe te passen.

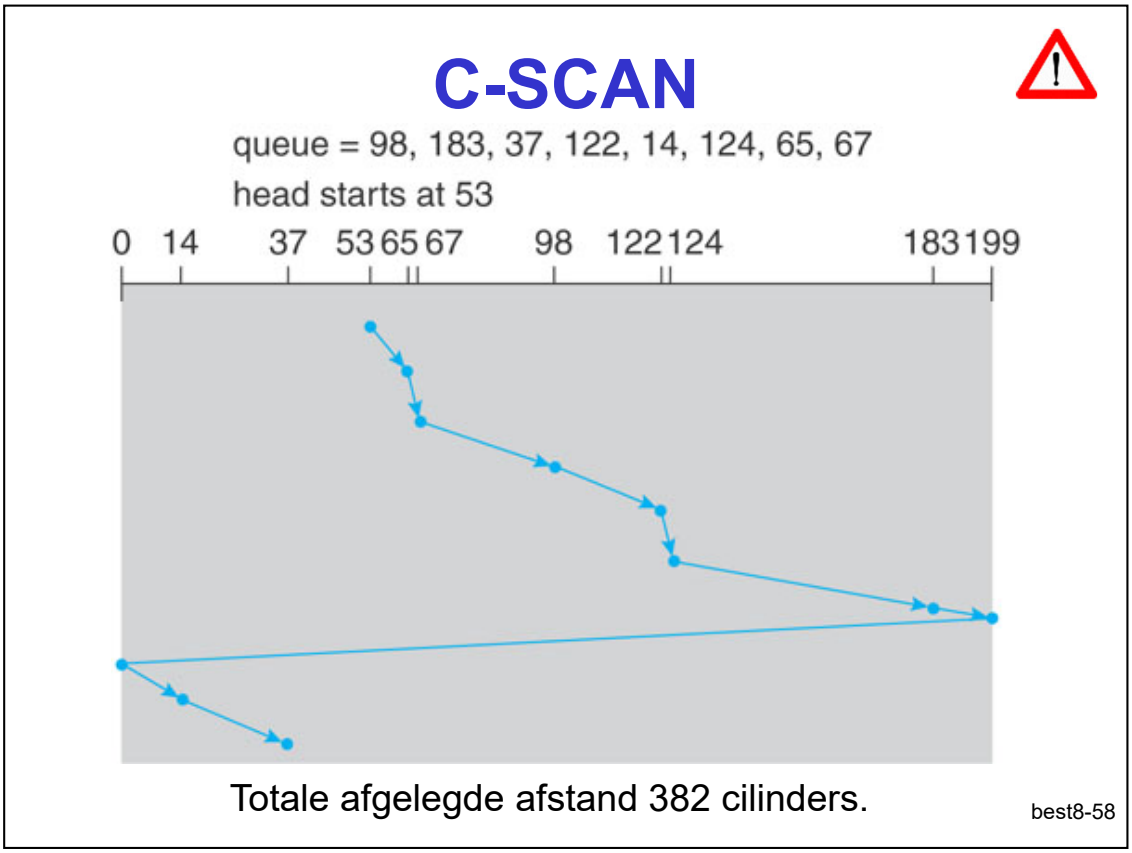
SCAN (busalgoritme)



Totale afgelegde afstand 236 cilinders.

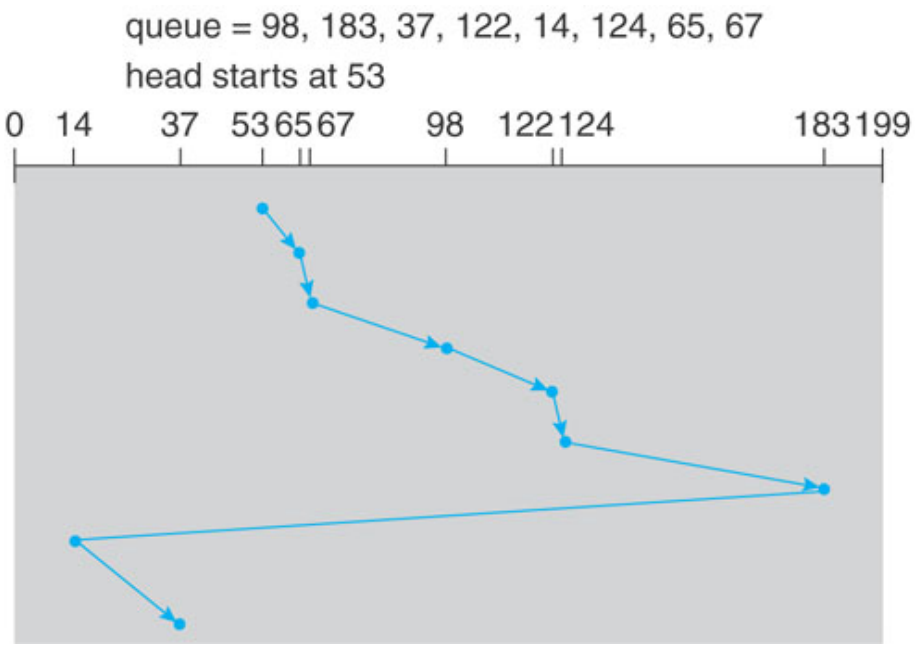
best8-57

Dit algoritme tracht de voordelen van SSTF te combineren met een eerlijk gedrag. Het overloopt alle sporen van de schijf in een bepaalde richting en omgekeerd hierbij alle operaties uitvoerend ‘en passant’. Dit garandeert dat alle schijfoperaties finaal zullen uitgevoerd geraken en is te vergelijken met een bus die op zijn heen- en terugweg bij alle haltes stopt waar personen staan te wachten.



Dit is een variant op de SCAN strategie. Op zijn terugweg zal de kop in het begin nauwelijks of geen operaties ontmoeten (deze sporen werden immers pas net bediend). Indien er dan toch al operaties zijn, zullen ze alle zeer recent zijn. De operaties aan de overzijde van de schijf staan echter reeds geruime tijd te wachten. Daarom zal C-SCAN op zijn terugweg geen sporen bedienen maar meteen bij spoor 0 herbeginnen. Hier zullen de gemiddeld oudere aanvragen eerst bediend worden. Dit algoritme is eerlijker dan SCAN omdat er minder variatie zal ontstaan op de wachttijd, maar het geeft wel een slechtere doorvoercapaciteit omdat de terugweg onderbenut wordt. Deze strategie zal zich bijzonder slecht gedragen indien de sporen sequentieel van rechts naar links moeten gelezen worden.

C-LOOK (liftalgoritme)



Totale afgelegde afstand 322 cilinders.

best8-59

In de praktijk zal men de kop van de schijf niet tussen de twee uiterste uiteinden van de schijf laten bewegen, maar wel tussen die twee uiterste sporen waarvoor er aanvragen zijn (vergelijkbaar met een lift). Deze strategie wordt LOOK genoemd. Ook van LOOK bestaat de circulaire variant, nl. C-LOOK.

Samenvattend kan gesteld worden dat SSTF een natuurlijke manier van plannen is, maar dat bij zwaar belaste systemen LOOK en C-LOOK de voorkeur genieten. Verder zal veel afhangen van de manier waarop de schijf precies gebruikt wordt, en dit wordt in belangrijke mate mee bepaald door het bestandssysteem. Een bestandsysteem waar de directoryinformatie ver verwijderd is van de gegevensblokken zal moeilijker te plannen zijn dan een bestandssysteem waar de directoryinformatie en de datablokken dicht bij elkaar op de schijf opgeslagen liggen.

RAID

- Redundant Array of Independent/Inexpensive Disks
- Idee: meerdere schijven om de betrouwbaarheid en/of de prestatie te verhogen
- Transparant, lijkt op gewone enkele schijf
- Typisch SCSI schijven, nu ook IDE RAID controllers

best8-60

Om secundaire opslag betrouwbaarder te maken kan men een aantal (onbetrouwbare) schijven zodanig combineren dat hun combinatie als geheel betrouwbaarder wordt. RAID (Redundant Array of Inexpensive Disks) is een technologie die hiervan gebruik maakt. Er bestaan 7 verschillende elementaire RAID niveaus.

Betrouwbaarheid verhogen door redundantie

- Verlies van 1 schijf is geen ramp indien de informatie van de schijf kan gereconstrueerd worden.
- Systeem is kwetsbaar indien er tijdens de hersteltijd een tweede fout optreedt
- Probleem: wet van Kryder (capaciteit x 2 per jaar) → hersteltijd neemt toe (tot 1 dag)

best8-61

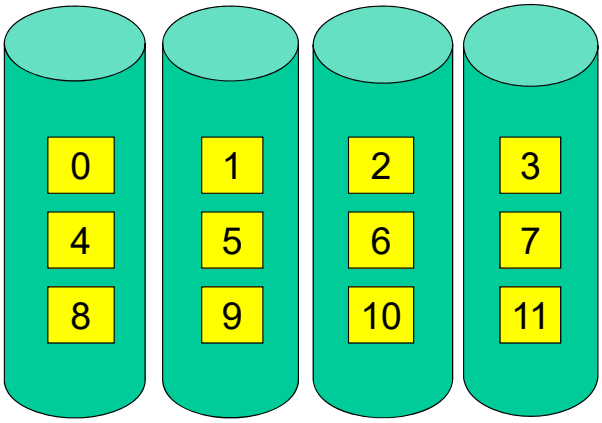
Prestatie verbeteren door parallellisme

- Indien informatie over verschillende schijven verspreid ligt, kunnen de verschillende schijven parallel gebruikt worden.

best8-62

RAID-0

Striping op sector/blokniveau



Grote transfers: lees/schrijfsnelheid x N (N=4)

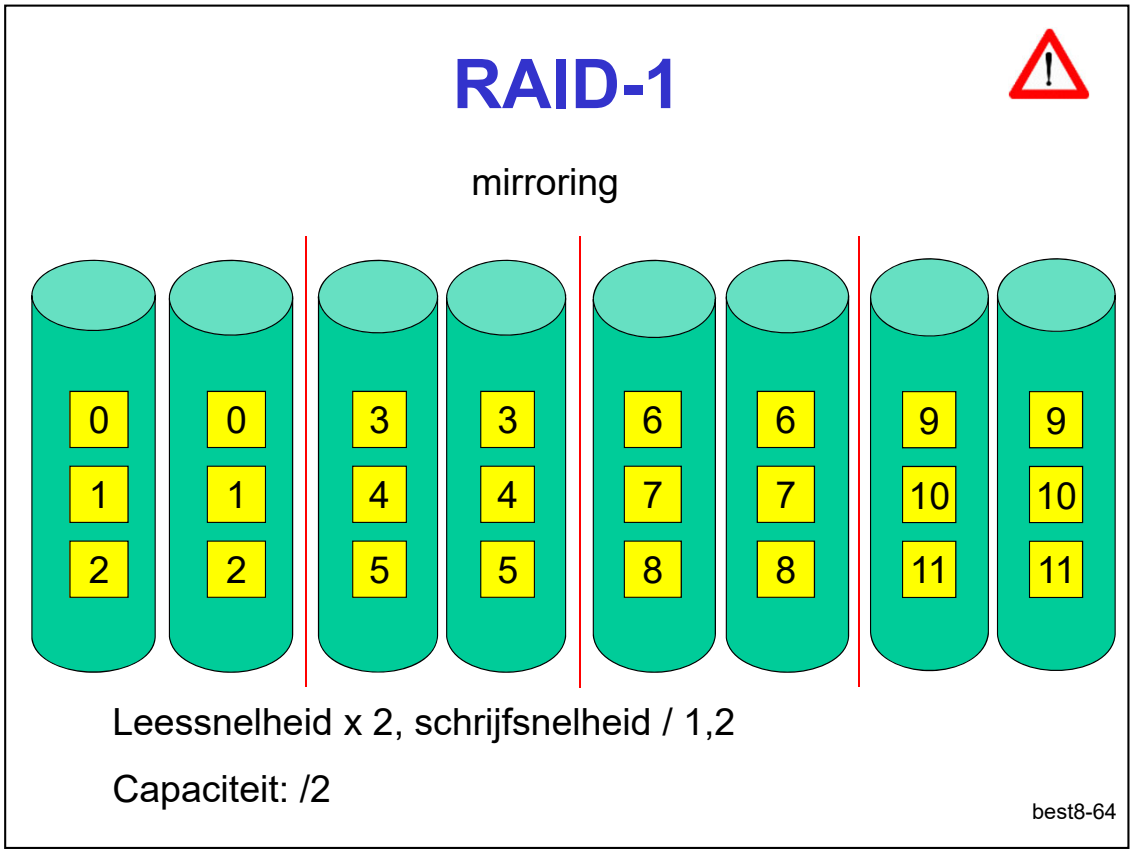
Kleine transfers ≤ 1 blok: geen verbetering

Grote bestanden: 1 schijf kapot: alle bestanden verloren

best8-63

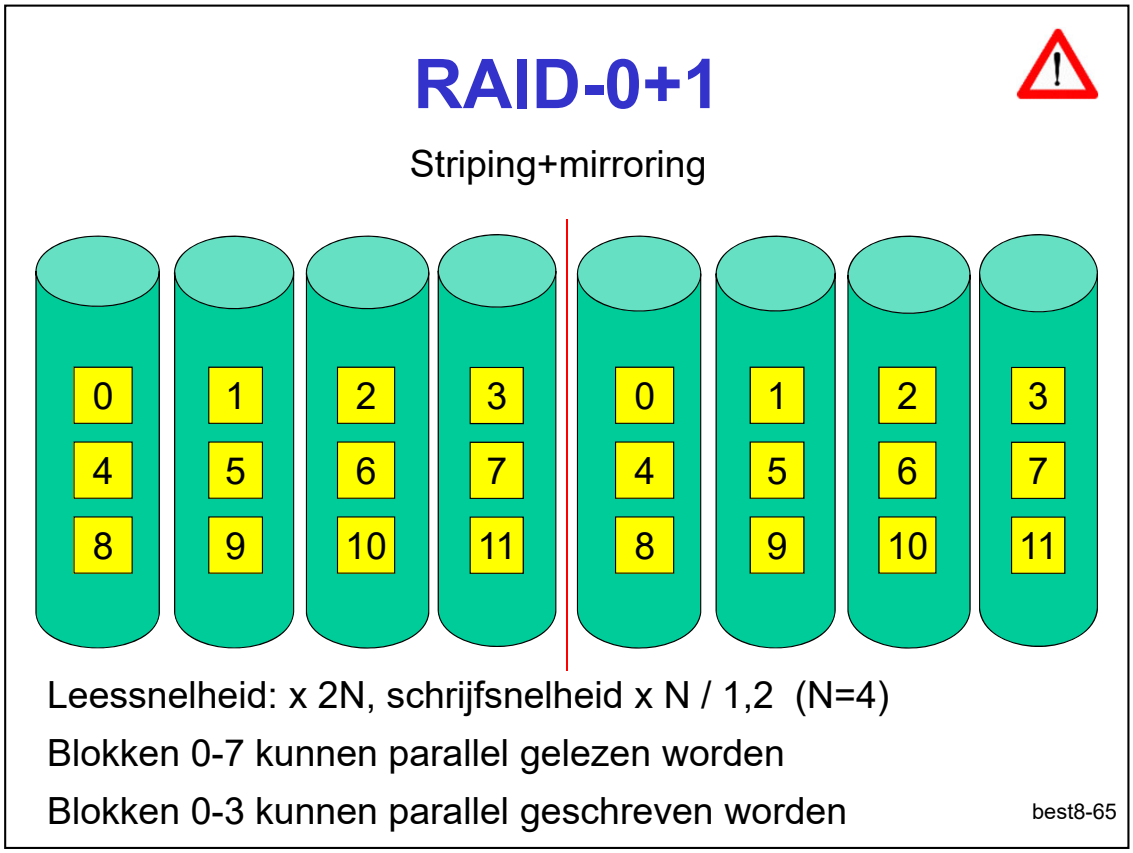
RAID-0 wordt ook striping genoemd. Hierbij zal men de blokken van een bestand uniform trachten te verdelen over het beschikbaar aantal schijven.

Een logische schijfoperatie zal dan vertaald worden naar verschillende fysieke schijfoperaties. RAID-0 is prima voor sequentiële toegang naar grote bestanden, als gevolg van de parallelle werking van de verschillende schijven. RAID-0 biedt minder voordelen voor directe toegang en is minder betrouwbaar dan één enkele schijf (er is geen opslagredundantie en doordat het bestandssysteem verspreid is over 4 schijven is de kans op defect 4 x groter dan bij 1 schijf). De doorvoercapaciteit is echter wel groter dan die van één enkele schijf.

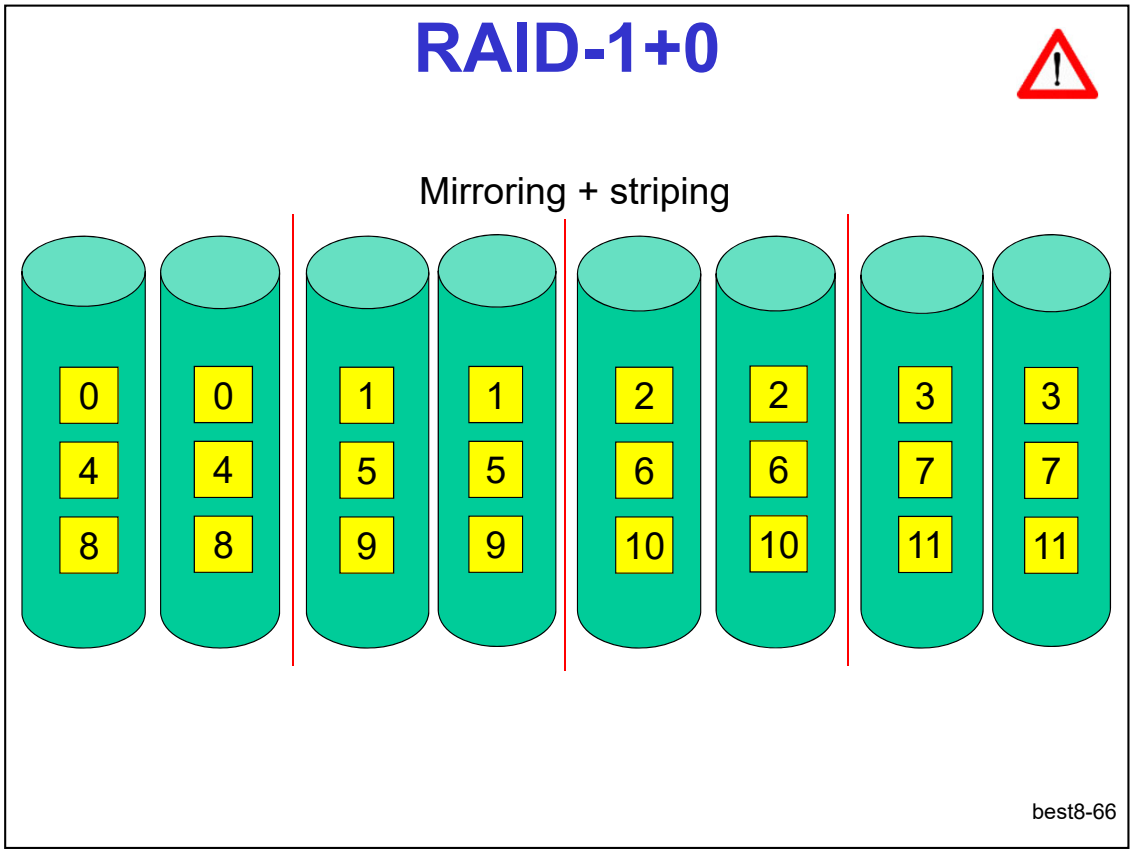


RAID-1 wordt ook mirroring of schadowing genoemd. Hierbij worden de gegevens telkens naar twee schijven geschreven.

Deze oplossing is duidelijk betrouwbaarder dan de vorige. De oplossing is wel duur gezien in het beste geval slechts 50% van de aanwezige opslagruimte gebruikt wordt. Het schrijven van gegevens in RAID-1 is 15 tot 20% trager dan in schijfsystemen zonder mirroring (veel van de manipulaties kunnen gemeenschappelijk gebeuren maar er moet tweemaal geschreven worden). Het lezen kan dan weer sneller gebeuren omdat de identieke gegevens op twee schijven aanwezig zijn, en dat men hierdoor de doorvoercapaciteit van het totale systeem verbetert. Hierbij kan men verschillende planningsstrategieën gebruiken zoals round robin (opeenvolgende leesoperaties worden telkens naar een andere schijf gestuurd, hetgeen nadelen heeft i.v.m. read-ahead) of geometrische planning waarbij men de schijf opdeelt in regio's. Een schijfoperatie voor een bepaalde regio wordt dan steeds naar dezelfde schijf gestuurd. Geometrische planning presteert dan weer slecht bij schijven waarvan de (actieve) gegevens ongelijk verdeeld zijn over de schijf.

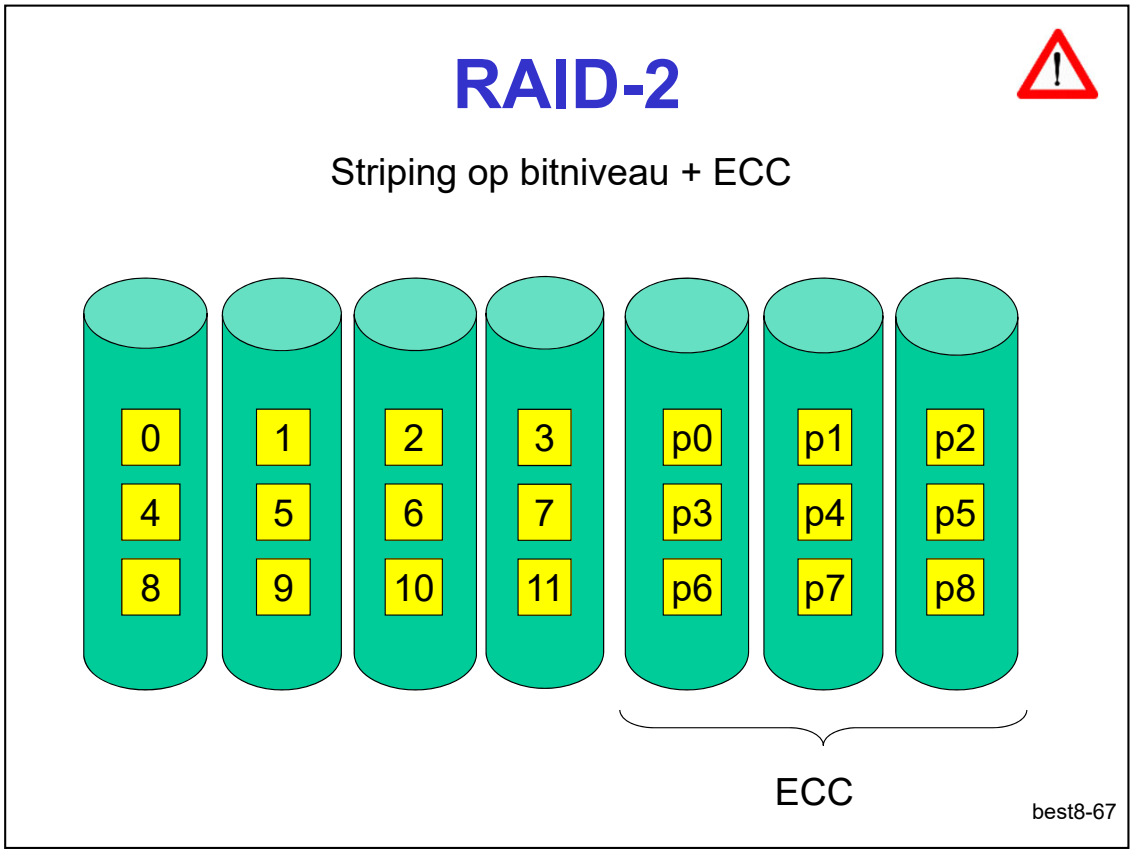


RAID 0+1 is de combinatie van striping + mirroring.



RAID 1+0 is de combinatie van mirroring en striping. Deze configuratie is betrouwbaarder dan RAID-0+1 dat niet meer werkt indien er twee schijven defect gaan die in twee verschillende mirror sets liggen (geen van beide sets kunnen dan nog individueel werken).

RAID-1+0 faalt pas nadat er twee schijven van de zelfde mirror set falen. Zolang het in verschillende mirror sets is, blijven deze gewoon verder werken.

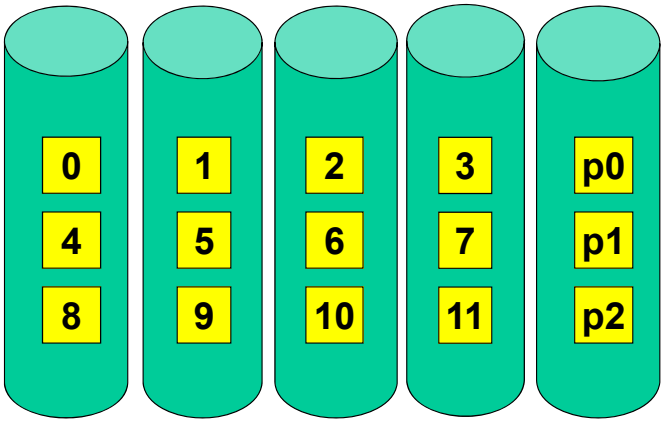


RAID-2 is striping op bitniveau, met opslag van een foutcorrigerende code op afzonderlijke schijven (ECC: error correcting code). De drie bits p0, p1, p2 volstaan om een enkelvoudige bitfout in de bits 0,1,2,3 te corrigeren. M.a.w. als er 1 van de 7 schijven uitvalt, blijft het RAID-systeem toch correcte data retourneren. In deze configuratie heeft RAID-2 dus 1 schijf minder nodig dan RAID-1 om dezelfde redundantie te garanderen.

RAID-2 vereist wel dat de assen van de verschillende schijven gesynchroniseerd zijn. De maximale bandbreedte van een dergelijk RAID systeem kan zeer hoog zijn, maar de prijs is navenant. RAID-2 werd gebruikt in de CM-2 computer waar 39 synchrone schijven gebruikt werden om 32 bits op te slaan. De bandbreedte was wel immens. Op de tijd nodig om 1 sector te lezen, kon deze RAID er 32 lezen!.

RAID-3

Striping op bitniveau + pariteit

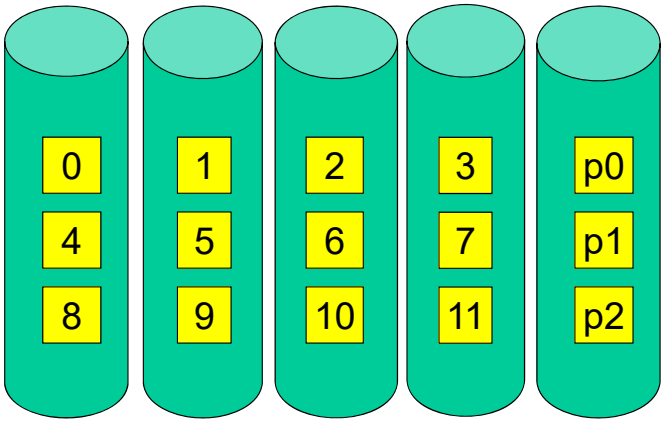


best8-68

RAID-3 is striping op bitniveau, met opslag op een afzonderlijke schijf van pariteitsinformatie. Pariteitsinformatie alleen is niet in staat om een bitfout te corrigeren, tenzij men weet over welke bit het gaat. Indien schijf 3 faalt, dan weet men welke bit moet berekend worden, en dit kan gemakkelijk uit de inhoud van de andere schijven en de pariteitsinformatie. De overhead aan redundantie is in dit geval beperkt tot 1 schijf, en dit ongeacht het aantal dataschijven. De schijven moeten wel gesynchroniseerd worden omdat we bitstriping toepassen.

RAID-4

Striping op sector/blok-niveau + pariteit

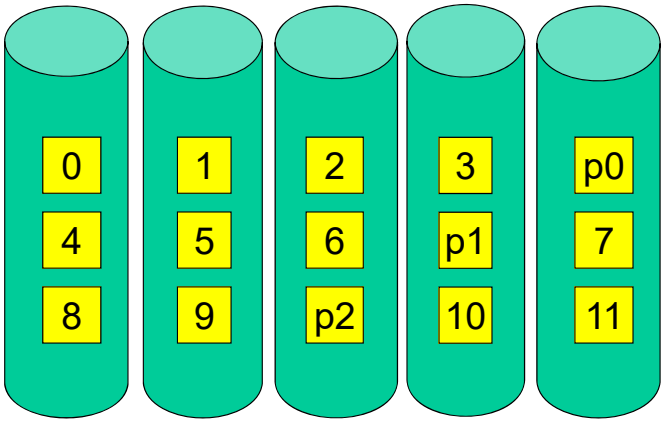


best8-69

RAID-4 is vergelijkbaar met RAID-3 maar striping gebeurt nu op het niveau van de blokken. Kleine leesoperaties kunnen nu typisch door één schijf afgehandeld worden. Een nadeel van RAID-4 is dat er bij elke schrijfoperaties minstens twee schijven betrokken zijn, waaronder steeds de pariteitsschijf. Deze schijf vormt de flessenhals van het totale systeem. Bij het lezen is dit niet nodig omdat de extra informatie enkel gebruikt wordt nadat een leesfout is opgetreden om de gelezen informatie te corrigeren. Voor wat betreft het lezen gedraagt RAID-4 zich dus zoals RAID-0. Bij het falen van één schijf kan de informatie gereconstrueerd worden, maar dit gaat wel gepaard met een aanzienlijk prestatieverlies.

RAID-5

Striping op sector/blok-niveau + gedistribueerde pariteit



best8-70

RAID-5 vergelijkbaar met RAID-4, maar nu wordt ook de pariteitsinformatie over alle schijven verdeeld (roterende of gedistribueerde pariteit genoemd), waardoor de pariteitsflessenhals weggewerkt wordt. Dit is het meest gebruikte RAID-systeem.

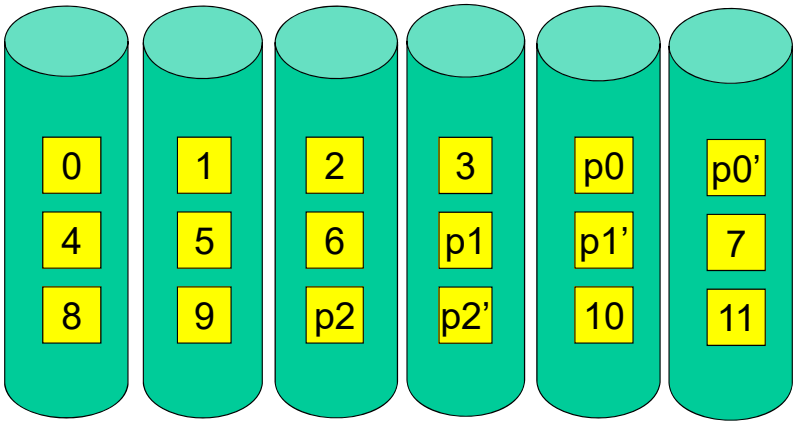
Bij sequentiële toegang zal de totale belasting gelijkmatig over alle schijven verdeeld worden. Gezien bij het lezen de pariteit niet moet gelezen worden (tenzij er fouten zouden optreden), zal de prestatie van RAID-5 voor het lezen dezelfde zijn als RAID-0 (zelfs nog iets beter omdat de extra schijf ten behoeve van de pariteit ook een gedeelte van de belasting op zich neemt).

Het veranderen (schrijven) van een blok in RAID-5 (maar ook in RAID-4) is echter nogal complex. In plaats van gewoon te schrijven moeten eerst het blok dat zal overschreven worden, en het bijbehorende pariteitsblok gelezen worden om toe te laten de pariteitsinformatie aan te passen, en pas nadien kan het nieuwe gegevensblok en de aangepaste pariteitsinformatie naar de schijf geschreven worden. Een schrijfoperatie zal dus 4 schijfoperaties (twee lees- en twee schrijfoperaties) veroorzaken.

Door het gebruik van aangepaste schijftussengeheugens wordt gepoogd om het verschil in toegangstijd tussen lezen en schrijven zoveel mogelijk weg te werken.

RAID-6

Striping op sector/blok-niveau + 2 pariteitsbits



best8-71

RAID-6 is vergelijkbaar met RAID-5, maar ditmaal met een code die ervoor zorgt dat er tot 2 schijven mogen uitvallen. Ook deze extra bits worden geroteerd.

Samenvattend kunnen we stellen dat bij het uitvallen van een schijf er, tenzij bij RAID-0, geen gegevens verloren zullen gaan. Wel zal de toegangstijd vanaf RAID-4 toenemen omdat de gegevens van de uitgevallen schijf enkel kunnen gereconstrueerd worden aan de hand van alle parallelle blokken op de andere schijven. Indien de RAID-configuratie 6 schijven breed is, en er één schijf uitvalt, zal men dus blokken moeten lezen van alle overblijvende schijven om het zesde te reconstrueren. Er zullen geen gegevens verloren gaan, maar de doorvoercapaciteit zal wel dalen. In het geval van mirroring (RAID-1) zal men van het uitvallen van een schijf nauwelijks last ondervinden.

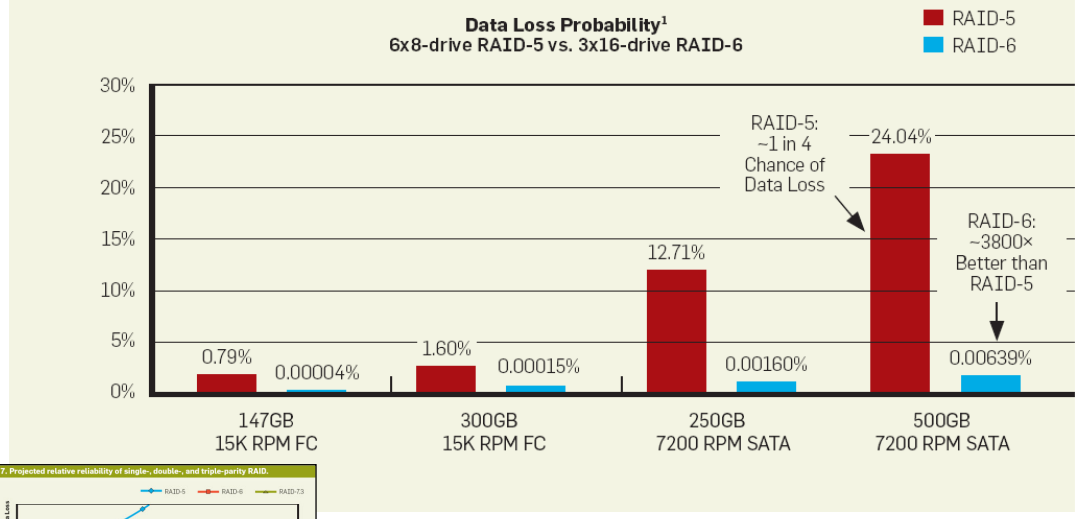
RAID-systemen met **hot spare** hebben een vrije schijf die stand-by is en meteen kan ingeschakeld worden om een schijf die fouten begint te vertonen automatisch te vervangen, nog voor er gegevens verloren gegaan zijn. Het reduceert de hersteltijd en dus de kans op het simultaan voorkomen van 2 defecte schijven.

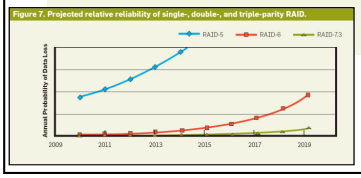
RAID-systemen met **hot swap** laten toe dat een defecte schijf fysiek kan vervangen worden zonder de werking van de andere schijven te verstoren.

De combinatie van hot spare en hot swap laat toe om secundaire-geheugensystemen te bouwen die 24 uur per dag, alle dagen van een jaar operationeel zijn en niet voor onderhoud moeten afgeschakeld worden, in het bijzonder wanneer ook het voedingsgedeelte dubbel uitgevoerd werd.

Raid-6 effect

Figure 1. Comparison of RAID-5 and RAID-6 reliability.[1]





[dam Leventhal, Triple-Parity RAID and Beyond, Communications of the ACM, Vol. 53 No. 1, Pages 58-63, Jan 2010]

best8-72

Door het raid-systeem te beschermen tijden de herstelfase, kan de kans op fatale defecten aanzienlijk gereduceerd worden. De kans op defect bij raid-5 lijkt misschien artificieel groot, maar dit komt o.a. omdat tijdens de reconstructie alle sectoren van alle schijven moeten gelezen worden, en er hierbij problemen kunnen vastgesteld worden in bestanden die zeer lange tijd niet gebruikt geweest zijn (eigenlijk verborgen defecten). Bij een reconstructie geeft dit dan aanleiding tot informatieverlies. Raid-6 zorgt voor bijkomende bescherming tijdens de reconstructie. Op termijn zal ook raid-6 niet meer volstaan, en zal men moeten overstappen naar een bijkomende pariteitsbit (zie figuur linksonder).

Om dergelijke verborgen problemen sneller op het spoor te komen past men preventief ‘scrubbing’ toe, dit is het regelmatig lezen, schrijven en verifiëren van alle sectoren. Op die manier kunnen fouten vastgesteld worden nog voor er effectief dataverlies optreedt. Scrubbing is echter niet gratis, en kost energie en bandbreedte.

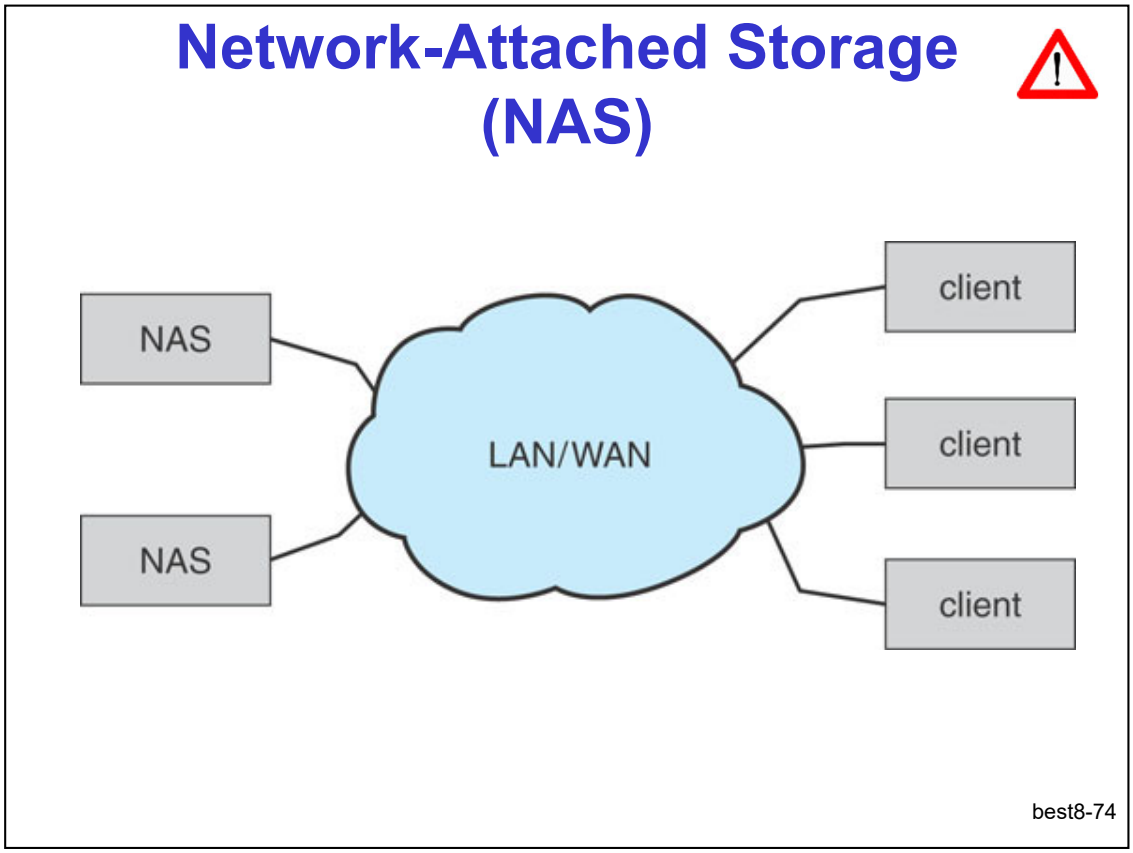
Opslagsystemen

Schijven kunnen verbonden worden met een computer

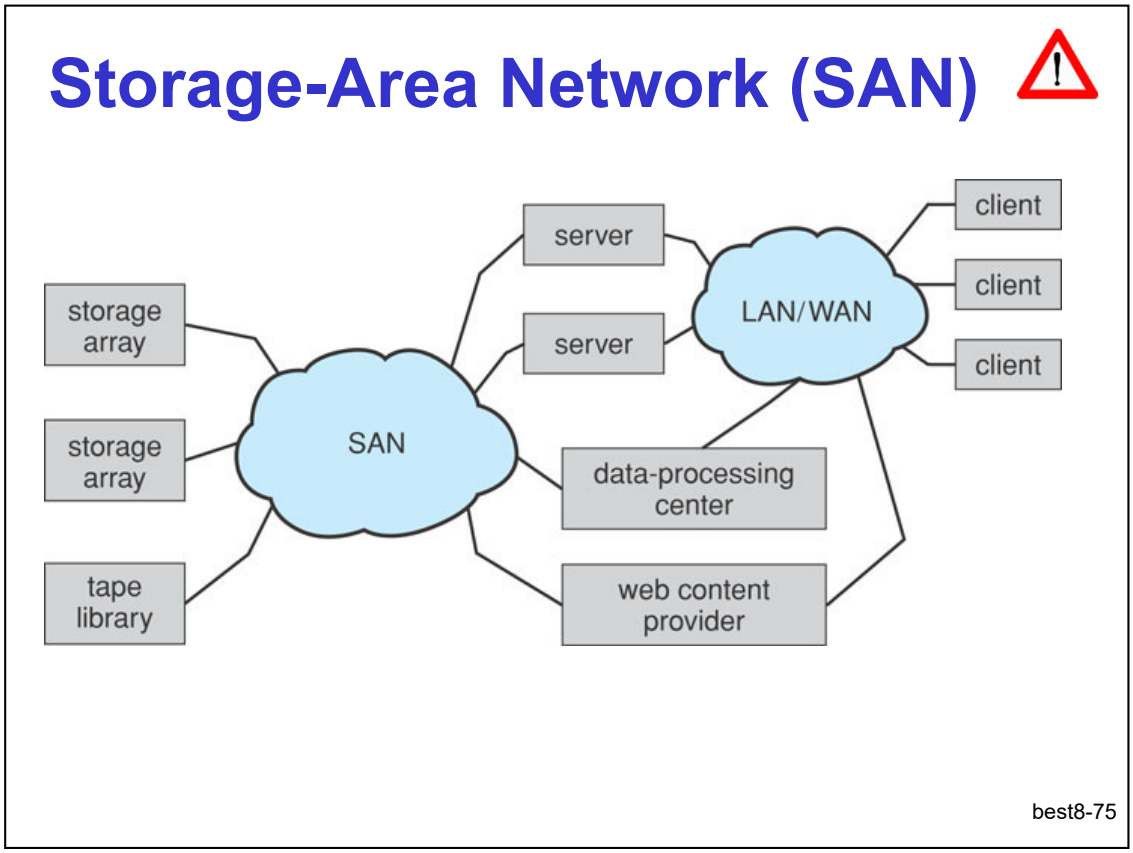
1. Via een IO-poort (IDE, SCSI, SATA,…)

2. Via een netwerk
    1. NAS: Network attached storage
    2. SAN: Storage area network

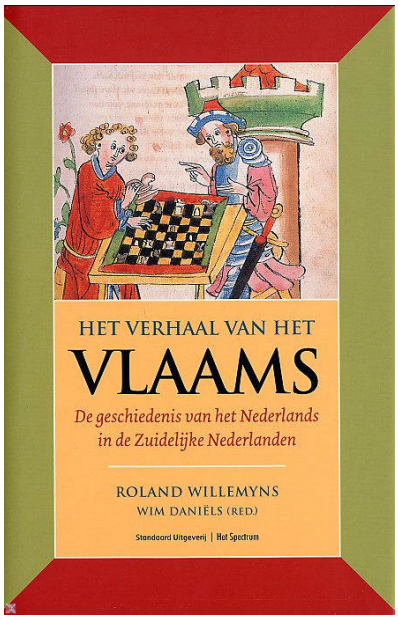
best8-73



Het aanbieden van schijfcapaciteit via het gewone LAN is een goedkope en aanvaardbare oplossing waarbij gebruik gemaakt wordt van het klassieke TCP/IP protocol, en bijvoorbeeld NFS. De NAS knopen in het netwerken kunnen b.v. bestandsservers zijn. Het nadeel van deze oplossing is dat de schijftrafiek nu ook over het LAN moet, en in competitie komt met het andere LAN-verkeer.



In deze oplossing wordt er een tweede netwerk gecreëerd, specifiek voor de uitwisseling van schijfgegevens. Dit netwerk kan een gewone LAN-verbinding zijn, maar het kan ook bestaan uit een specifiek voor dit doel ontworpen krachtig netwerk. Het grote voordeel is dat er hier geen competitie is tussen het LAN-verkeer en het schijfverkeer, en dat het SAN-netwerk kan geoptimaliseerd worden voor schijftrafiek. SAN-netwerken zijn uiteraard wel duur.



best8-76

best8-76