

# Computerarchitectuur

Roel Van Steenberghe

---

# Table of Contents

Over deze cursus .....	x
1. Licentie .....	1
Voorwoord .....	ii
2. Computervoeding .....	3
2.1. Principe en werking .....	3
2.2. Eigenschappen .....	4
2.2.1. Vormfactor en connectoren .....	4
2.2.2. Vermogen .....	6
2.2.3. Rendement .....	7
2.2.4. Geluid .....	7
2.2.5. Problemen met voedingen .....	9
2.3. Accu's .....	10
2.3.1. Eigenschappen .....	10
2.4. Uninterruptible Power Supply (UPS) .....	12
2.4.1. Online UPS .....	14
2.4.2. Offline UPS .....	14
2.4.3. Line-interactive UPS .....	15
2.5. Bibliografie bij dit hoofdstuk .....	15
3. CPU .....	16
3.1. Overzicht .....	16
3.2. Technologie en functionaliteit .....	17
3.3. Kloksnelheid .....	19
3.4. Processorarchitectuur .....	20
3.4.1. Pipelining .....	20
3.4.2. Superscalaire processoren .....	21
3.4.3. Cache .....	24
3.5. APU, SoC .....	25
3.6. Montage .....	26
3.7. Processoren van de toekomst .....	27
3.8. Bibliografie bij dit hoofdstuk .....	27
4. Het Geheugen .....	28
4.1. Overzicht .....	28
4.2. Lokaliteitsprincipes .....	29
4.3. Soorten geheugens .....	32
4.3.1. Behuizing .....	32
4.4. Technologie .....	34

4.4.1. Gemultiplexte adresklemmen .....	36
4.4.2. Destructieve leescyclus .....	36
4.4.3. Refresh .....	36
4.4.4. Bandbreedte bij DRAM .....	37
4.5. Fast Page DRAM (FP-DRAM) .....	37
4.6. EDO RAM .....	38
4.7. Synchronous DRAM .....	39
4.7.1. DDR SDRAM - DDR2 - DDR3 .....	39
4.8. Optimalisatietechnieken .....	43
4.8.1. Interleaving .....	43
4.8.2. Dual channel .....	43
4.8.3. Buffered/registered RAM en foutdetectie .....	45
4.9. Cache geheugen .....	45
4.9.1. Werking .....	47
4.9.2. Soorten caches .....	49
4.9.3. Overschrijfstrategieën .....	49
4.10. Associativiteit .....	50
4.10.1. Fully Associative cache .....	50
4.10.2. Direct mapped cache .....	52
4.10.3. Set-associative cache .....	53
4.10.4. Snelheid van de cache .....	55
4.11. Virtueel geheugen .....	57
4.11.1. Werking .....	57
4.11.2. Paging - segmenting .....	58
4.11.3. Snelheid en virtueel geheugen .....	60
4.12. Bronvermelding bij dit hoofdstuk .....	61
5. Opslagmedia .....	62
5.1. Harde schijf .....	62
5.1.1. Fysieke opbouw .....	62
5.1.2. Data .....	64
5.1.3. Toegangstijd .....	65
5.1.4. Invloed van fysieke geometrie op de performantie .....	66
5.1.5. Adressering .....	68
5.1.6. Scheduling .....	69
5.1.7. FCFS: First-Come First-Serve .....	70
5.1.8. SSTF: Shortest Seek Time First .....	71
5.1.9. SCAN: Scannen .....	72
5.1.10. C-SCAN: Cirkulair Scannen .....	73

5.1.11. LOOK + C-LOOK: Aangepaste SCAN + C-SCAN .....	73
5.1.12. Schijfcontroller .....	74
5.2. Solid state drive .....	75
5.2.1. Flash technologie .....	77
5.2.2. Schrijfcyclus van Flash Memory .....	78
5.2.3. Optimalisatie van Flash .....	80
5.3. Snelheid bij disks: IOPS .....	81
5.3.1. Types .....	81
5.3.2. Gecombineerde snelheid .....	82
5.4. Logische indeling opslagmedia .....	82
5.4.1. Boot blok .....	82
5.4.2. Master Boot Record (MBR) layout .....	83
5.4.3. GPT layout .....	87
5.5. Bestandssystemen .....	99
5.5.1. inleiding .....	99
5.5.2. een bestandssysteem 'mounten' .....	100
5.5.3. types: journaling file systems .....	100
5.5.4. Case study 1: FAT file system .....	101
5.5.5. Case study2: NTFS .....	107
5.5.6. case study 3: Ext4 .....	108
5.5.7. Andere bestandssystemen .....	110
5.5.8. Bestanden wissen .....	111
5.6. Bronvermelding bij dit hoofdstuk .....	112
6. Opstartroutine .....	113
6.1. Geheugens voor opstartroutine .....	113
6.2. Opstartroutine met BIOS .....	114
6.2.1. Systeemstart .....	114
6.2.2. Power On Self Test (POST) .....	114
6.2.3. Zoeken naar BIOS-uitbreidingen .....	114
6.3. Plug and play configuratie .....	115
6.4. Extensible Firmware Interface .....	116
6.4.1. BIOS op pensioen .....	116
6.4.2. (U)EFI verspreiding .....	117
6.5. Coreboot .....	118
7. Internal I/O .....	119
7.1. I/O transfers .....	119
7.1.1. Pollen .....	119
7.1.2. Interrupts .....	120

7.1.3. Direct Memory Access (DMA) .....	123
7.2. Bussystemen .....	125
7.2.1. In den beginne... .....	125
7.2.2. PCI .....	126
7.2.3. AGP (Accelerated Graphics Port) .....	130
7.2.4. PCI-express .....	131
7.3. Bussystemen voor harde schijven .....	135
7.3.1. In den beginne... .....	135
7.3.2. SATA .....	136
7.3.3. eSATA .....	139
7.3.4. Serial Attached SCSI (SAS) .....	139
8. Externe I/O .....	141
8.1. In den beginne .....	141
8.2. PS/2 .....	142
8.3. USB .....	143
8.3.1. Overzicht .....	143
8.3.2. Mechanische en elektrische opbouw .....	144
8.3.3. Communicatie over USB .....	146
8.3.4. Endpoints .....	148
8.3.5. Types van transfers .....	148
8.3.6. Enumeratie en P&P .....	151
8.3.7. Device Classes, Drivers .....	152
8.3.8. USB 3.0 .....	153
8.4. Thunderbolt .....	155
8.4.1. Architectuur .....	156
9. Literatuurlijst .....	158

---

## List of Figures

2.1. Principe schakelende voeding .....	3
2.2. ATX 2.0 moederbord connector (CC Wikimedia commons) .....	5
2.3. accu .....	10
2.4. Rack-mountable UPS (bron: www.apc.com) .....	13
2.5. Online UPS ( © GFDL Joslee 2007 ) .....	14
2.6. Offline UPS ( © Joslee 2007 GFDL) .....	15
2.7. Line interactive UPS ( © Joslee 2007 GFDL) .....	15
3.1. Wet van Moore (CC, Wikimedia Commons) .....	18
3.2. Intel roadmap (copyright 2008-2012 WhiteTimberwolf GFDL) .....	19
3.3. processor pipeline (CC mediawiki) .....	21
3.4. Sandy bridge microarchitectuur .....	22
3.5. AMD bulldozer architectuur (copyright AnandTech) .....	23
3.6. single threaded applicatie op multicore processor .....	24
3.7. LGA2011 socket zonder processor .....	26
4.1. Geheugenpiramide .....	29
4.2. Courante DDR-dimm modules (Wikimedia public domain) .....	33
4.3. SO-DIMM modules (Wikimedia public domain) .....	34
4.4. voorstelling statisch geheugen .....	34
4.5. Voorstelling leescyclus dynamisch geheugen .....	35
4.6. Dynamisch geheugen met statische buffer .....	37
4.7. Fast page DRAM .....	38
4.8. EDO-RAM .....	38
4.9. afbeelding 20 Leesoperatie bij SD-RAM (bron: Micron) .....	39
4.10. Write cyclus bij DDR-RAM (bron: Micron) .....	40
4.11. DDR-timing $T_{CL}=2$ (bron: Micron) .....	40
4.12. normale geheugentoegang .....	43
4.13. Memory interleaving .....	43
4.14. geheugentoegang met Dual Channel .....	44
4.15. dual channel sockets .....	44
4.16. Cache als buffer tussen CPU en geheugen .....	47
4.17. principe fully associative cache .....	50
4.18. voorbeeld fully associative geheugen .....	51
4.19. principe direct mapped cache .....	52
4.20. voorbeeld direct mapped .....	52
4.21. principe set-associative cache .....	53
4.22. voorbeeld set associative .....	53

4.23. cache misses in functie van grootte en associativiteit (bron: wikipedia) .....	55
4.24. Page-faults bij opstarten software .....	58
4.25. Paging table (Wikimedia Commons, BSD) .....	59
5.1. Fysieke opbouw harde schijf .....	62
5.2. logische indeling van een schijf (bron: technet.microsoft.com) .....	63
5.3. Onderverdeling van tracks in sectors en bits .....	64
5.4. Perpendicular recording (bron: Wikipedia) .....	65
5.5. HDD opdeling in zones .....	67
5.6. First-Come First-Served Disk Scheduling .....	71
5.7. Shortest Seek Time First Disk Scheduling .....	72
5.8. Scan Disk Scheduling .....	72
5.9. Circular Scan Disk Scheduling .....	73
5.10. Look Disk Scheduling .....	74
5.11. Cirkular Look Disk Scheduling .....	74
5.12. SSD drive zonder behuizing .....	75
5.13. SLC geheugen .....	77
5.14. MLC geheugen .....	77
5.15. Schrijfcyclus Flash .....	79
5.16. Garbage collection bij Flash Memory .....	81
5.17. partitiestructuur (bron:Microsoft Technet) .....	85
5.18. Voorbeeld partitietabel met uitgebreide partitie .....	86
5.19. Partitietabel met meerdere logische partities .....	87
5.20. GPT versus MBR layout (bron: Microsoft Technet) .....	89
5.21. Contigue allocatie .....	90
5.22. Gelinkte allocatie .....	92
5.23. Allocatietabel .....	93
5.24. Geïndexeerde allocatie .....	94
5.25. FAT organisatie .....	102
5.26. voorbeeld FAT-tabel met fragmentatie .....	107
5.27. Inode structuur (bron onbekend) .....	110
7.1. DMA request (bron: <a href="http://www.talktoanit.com/">http://www.talktoanit.com/</a> ) .....	124
7.2. oude en nieuwe microarchitectuur .....	128
7.3. PCI slots .....	130
7.4. PCIe slots .....	132
7.5. PCIe protocolstack .....	134
7.6. sata connector .....	137
7.7. sata multiplier .....	138

---

## List of Tables

2.1. 80 plus certificatie (bron: Wikipedia ) .....	8
3.1. processoroverzicht .....	16
3.2. Cache in desktop en serverprocessoren (actuele topmodellen, feb 2014) ....	25
4.1. DDR3 snelheden (bron: wikipedia) .....	41
5.1. Inhoud Master Boot Record .....	83
5.2. Inhoud partitietabel-record .....	84
5.3. PBS van een FAT32 partitie .....	103

---

## List of Examples

2.1. rekenvoorbeeld stroomverbruik .....	8
2.2. Oefening .....	11
3.1. oefening .....	26
4.1. oefening .....	30
4.2. oefening .....	31
4.3. voorbeeld .....	53
4.4. voorbeeld .....	54
4.5. Voorbeeld .....	56
4.6. Voorbeeld .....	59
5.1. voorbeeld .....	67
5.2. Voorbeeld: zoeken van een bestand in de op een FAT-partitie. ....	106
5.3. interpretatie van een runlist .....	108
7.1. oefening .....	138

---

# Over deze cursus

Dit lesmateriaal werd opgesteld doorheen de jaren, met tal van auteurs. Dhr Sven Sanders en Dhr Johan Donne verdienen daarbij een speciale vermelding, omdat zij de fundamenten van deze informatiebron reeds jaren geleden gelegd hebben.

Er werd in deze cursus gepoogd om steeds correct om te gaan met materiaal van externe bronnen, inzake copyrights en bronvermelding. Mocht je toch een fragment of afbeelding vinden zonder correcte bronvermelding, geef dat dan zeker door.

Stuur hiervoor zeker een [mailtje<sup>1</sup>](#). Er zal meteen het nodige gedaan worden om de situatie recht te zetten.

Door deze cursus aan te bieden op Github is er ook de hoop dat er ook door anderen (ja, ook jij :-) ) toevoegingen kunnen gebeuren.

---

<sup>1</sup> <mailto:roel.vansteenbergh@hubkaho.be>

---

# Chapter 1. Licentie

Hergebruik van het materiaal is dan ook toegelaten, maar wel onder voorwaarden:

- het materiaal, of fragmenten ervan, mogen niet commercieel beschikbaar gesteld worden zonder uitdrukkelijke en schriftelijke toestemming van de auteur(s)
- de inhoud aanpassen mag, maar dan op voorwaarde dat de aanpassingen ook publiek beschikbaar worden gesteld. Bij voorkeur gebeurt dit online (met een pull request op Github), zodat iedereen mee kan genieten van de verbeteringen.

Concreet betekent dit dat al het materiaal onder de [Creative Commons Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Internationaal-licentie<sup>1</sup>](#) valt, tenzij anders gespecificeerd.

---

<sup>1</sup> <http://creativecommons.org/licenses/by-nc-sa/4.0/deed.nl>

---

# Voorwoord

Als iemand van de buitenwereld je de komende jaren vraagt wat je precies gestudeerd hebt, of wat je doet als werk, dan zal je antwoord vaak ‘iets met computers’ zijn. Soms is dat nu eenmaal de makkelijkste manier om je er vanaf te maken.

Tegenwoordig zijn computers, tablets, smartphones en andere devices immers zo gewoon, dat ze bijna thuis horen in het rijtje van basisbehoeften als stromend water, elektriciteit en TV.

Toch blijft het belangrijk voor jullie als ICT’ers om te weten wat er onder de motorkap van je devices schuilt. Die achtergrondkennis is onontbeerlijk om later efficiënt problemen op te lossen of producten (in de breedste zin van het woord) van goede kwaliteit af te leveren. Zelfs wie zich later zal toespitsen op het ontwikkelen van software, zal efficiënter kunnen werken als hij ook snapt wat achter de schermen gebeurt.

Deze cursus probeert je een overzicht te geven van de interne keuken van een moderne computer terwijl de cursus processorarchitectuur dan weer iets dieper ingaat op de werking van het kloppend hart ervan.

Uiteraard zijn twaalf lessen veel te weinig om alle onderdelen tot op het bot uit te benen. Daarom kan ik enkele standaardwerken aanbevelen, die zeker een bron van inspiratie vormden voor deze cursus. De boeken van William Stallings [**STALLINGS**] en Umakishore Ramachandran [**RAMA**] verdienen zeker je aandacht.

Een overzicht van de werkvormen die bij dit vak gebruikt worden, vind je terug in de ECTS fiche. Deze kan je vinden op [onderwijsaanbod.kahosl.be](http://onderwijsaanbod.kahosl.be)<sup>1</sup>

Veel succes met deze cursus!

Roel Van Steenberghe

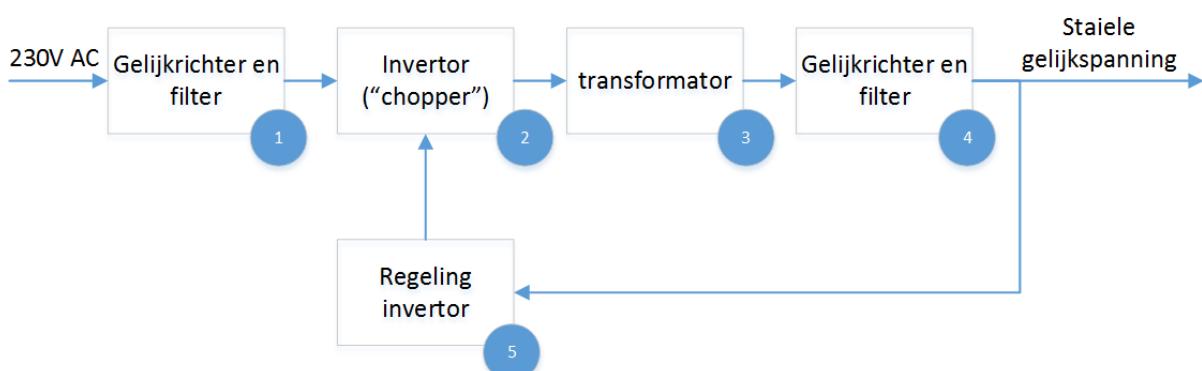
---

<sup>1</sup> <http://onderwijsaanbod.hubkaho.be>

# Chapter 2. Computervoeding

## 2.1. Principe en werking

Een computer, of het nu een PC, laptop, server of smartphone is, kan enkel functioneren als de juiste elektrische spanningen aangebracht worden. Elektronische componenten vragen een relatief lage, maar erg stabiele gelijkspanning om te functioneren. Deze wordt geleverd door een voeding, die de wisselspanning van het elektriciteitsnet omzet naar de nodige gelijkspanningsniveaus. De techniek die hierbij gebruikt wordt noemt men switched mode power supply of schakelende voeding. Het schema van de omzetting wordt weergegeven in onderstaande afbeelding



**Figure 2.1. Principe schakelende voeding**

Een eerste stap ① is de **gelijkrichter** aan de ingang, waarmee de wisselspanning wordt omgezet naar een gelijkspanning. Deze gelijkspanning wordt vervolgens gefilterd, zodat de variatie in het spanningsniveau beperkt wordt. De tweede stap ② is een stuk minder voor de hand liggend. Op basis van de gelijkgerichte en gefilterde ingangsspanning zal een invertor een blokgolf genereren. Deze wisselspanning wordt bekomen door het aan- en afschakelen van de ingangsspanning. Eigenlijk is de combinatie van de eerste twee stappen samen te vatten als een frequentieomvormer. Het ingangssignaal wordt omgezet naar een hoger-frequent signaal (van 50Hz naar frequenties boven 20kHz).

Het voordeel van deze omzetting is dat de **transformator** in de volgende stap een stuk kleiner en efficiënter kan zijn. Deze gelijkrichter en transformator ③ brengt de spanning naar het gewenste niveau, waarna het met de uitgangs-gelijkrichter en filter ④ wordt omgezet naar een stabiele gelijkspanning.

In de figuur is ook te zien dat de uitgangsspanning teruggekoppeld wordt ⑤ naar de invertor. Op die manier kan de uitgangsspanning nog geregeld worden. De invertor-

stap heeft immers ook invloed op de amplitude van zijn uitgang. Deze **terugkoppeling** gebeurt meestal met optocouplers om een galvanische scheiding te bekomen.



Om het verbruik van een CPU uit te drukken, wordt vaak gesproken over **TDP** (Thermal Design Power). Dat is een waarde die aanduidt hoeveel energie de CPU maximaal dissipert onder een zware, maar realistische belasting gedurende een bepaalde tijd. Echter dient opgemerkt te worden dat 'AMD' en 'Intel' hiervoor verschillende berekeningsmethodes gebruiken. De TDP geeft dus een indicatie over het maximale verbruik, maar gedetailleerde benchmarks blijven nodig om exacte waarden te kennen.<sup>1</sup>

## 2.2. Eigenschappen

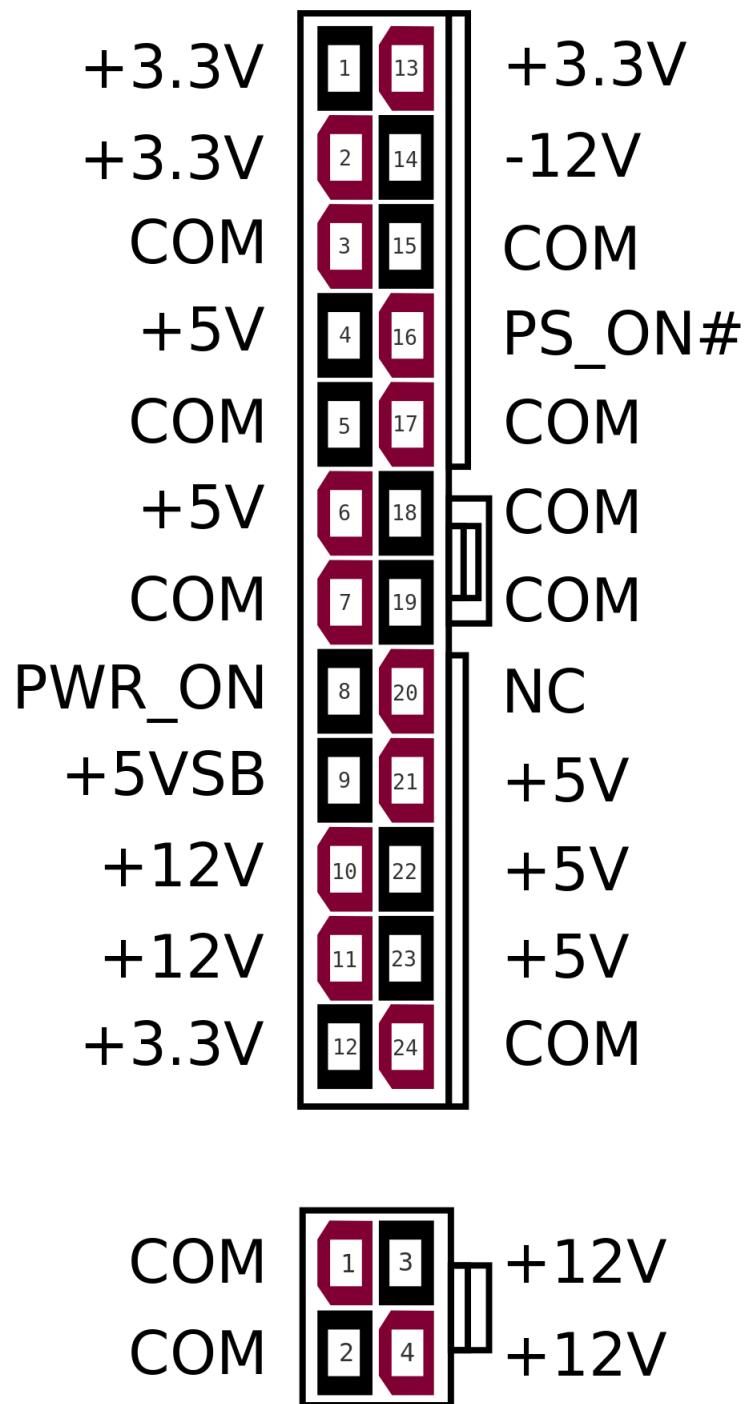
### 2.2.1. Vormfactor en connectoren

De eerste (IBM) PC beschikte over een moederbord met AT vormfactor, de bijhorende voeding was dan ook een AT voeding. In 1995 kwam er een opvolger, met name de ATX vormfactor. Ondertussen is deze specificatie ook geëvolueerd (ondertussen ATX 2.3).

De belangrijkste verschillen situeren zich op het vlak van de spanningen die de voeding kan afgeven en de connectoren die voorzien zijn op de voeding. In het bijzonder is er natuurlijk een verschil tussen de verschillende connectoren die op het moederbord worden aangesloten. Een AT voeding bood een connector van tweemaal zes aansluitingen, een ATX voeding biedt daarentegen een connector met 24 aansluitingen. (in de oorspronkelijke versie was dit 20, tegenwoordig is er ook steeds een vierpolige connector die twee keer twaalf Volt levert aan de processor.)

---

<sup>1</sup>meer uitleg over de berekening van TDP bij Intel vind je in [deze whitepaper](http://www.intel.com/content/www/us/en/benchmarks/resources-xeon-measuring-processor-power-paper.html) [http://www.intel.com/content/www/us/en/benchmarks/resources-xeon-measuring-processor-power-paper.html]



**Figure 2.2. ATX 2.0 moederbord connector (CC Wikimedia commons)**

Een belangrijk verschil tussen beiden is de aanwezigheid bij ATX van een 3.3V spanning, een +5V standby en power on signaal. Deze laatste twee maken het mogelijk

dat de mechanische schakelaar van de AT voeding (die rechtstreeks de voeding aanstuurde), vervangen kon worden door een elektronisch signaal van het moederbord naar de voeding. In eerste instantie kan het moederbord dit signaal sturen als het zelf een input krijgt van een drukknop. Er zijn echter ook alternatieven mogelijk, zoals wake-on-lan, speciale toetsen op een toetsenbord,...

Hieruit kan je afleiden dat bij een computer die uitgeschakeld is, een deel van het moederbord nog steeds onder spanning staat. Deze spanning kan je alleen wegnemen door de voeding uit te schakelen (schakelaar op voeding, stekker uittrekken).

Naast de verschillen in connectoren die op het moederbord worden aangesloten, is er ook onderscheid op het vlak van de andere connectoren. Afhankelijk van de andere apparaten (en hun voedingsaansluiting), moet je erop letten om een voeding te kiezen die de nodige connectoren aanbiedt.

Enkele belangrijke connectoren zijn:

- ATX power connector (ook wel moederbord connector genoemd)
- 4-pin connector (molex): o.a. voor (ATA) schijven, optische drives
- SATA voedingsconnector
- Auxiliary connectors: verschillende varianten van extra voedingsconnector en om extra vermogen te leveren

## 2.2.2. Vermogen

Een erg belangrijke eigenschap voor een voeding is het vermogen dat ze kan leveren. Uiteraard moet dit te leveren vermogen voldoende zijn om alle componenten in het systeem te voorzien van stroom. Het vergelijken van voedingen op dit vlak is iets complexer dan kijken naar de waarden die de fabrikant op zijn verpakking adverteert.

Belangrijker dan het getal is de betekenis ervan. Aangezien er geen voorschriften zijn voor de bepaling van die vermogenswaarde, kan 500W bij de ene fabrikant betekenen dat de voeding 500W piekvermogen kan leveren bij 10°C en bij een andere een continu vermogen van 500W bij 40°C. Als het systeem continu 450W nodig heeft, zou de eerste voeding kunnen falen.

Een tweede belangrijke opmerking is dat niet alleen het totale vermogen belangrijk is, maar ook het vermogen dat op elke voedingsspanning apart geleverd kan worden. Het is duidelijk dat een computervoeding meerdere eindtrappen moet bevatten voor

de verschillende spanningen. Op elk van deze rails is er een maximale stroom die geleverd kan worden.

Als de maximale stroom op de 12V rail 5A is, kan je met een 500W voeding niet voorzien in de behoeften van een computer die een vermogen van 200W nodig heeft, maar wel 6A op de 12V rail. Dit kan een belangrijke reden voor prijsverschillen in voedingen zijn. Goedkopere voedingen kunnen typisch meer stroom leveren bij de lagere spanningen en minder bij 12V. Er moet nog worden opgemerkt dat sommige voedingen verschillende rails hebben voor eenzelfde voedingsspanning. Op elk van deze rails is dan een maximale stroom vastgelegd. Het zal wel duidelijk zijn dat je dan best de verbruikers op een zo evenwichtig mogelijke manier over deze rails moet verdelen.

Een laatste opmerking is dat het vermogen van de voeding zo goed mogelijk op het systeem moet worden afgestemd. Uiteraard betekent dit dat je voldoende piekvermogen nodig hebt, maar zomaar een voeding van 1kW aanschaffen voor een systeem dat 200W nodig heeft is niet meteen een goede keuze.

### 2.2.3. Rendement

Het ‘groene’ aspect bij computers komt steeds meer naar voor. Het rendement van de voeding is daarbij een belangrijke factor. Je wil natuurlijk voor elke 100 Watt die je uit het stroomnet haalt, ook 100W prestaties zien. Helaas is dit niet mogelijk: voedingen hebben een rendement dat een stuk lager ligt dan de ideale 100%. Dat verlies uit zich voornamelijk in warmte, die dan weer moet afgevoerd worden. Het spreekt voor zich dan een hoger rendement meestal ook een iets hoger prijskaartje met zich zal meebrengen. Toch is dit het overwegen waard als je een kleine rekenoefening maakt.

### 2.2.4. Geluid

De geluidsproductie van een computer is in verschillende gebruiksomgevingen liefst zo klein mogelijk. Een belangrijke bron van lawaai wordt gevormd door de verschillende koelingen en in het bijzonder de ventilatoren die hierbij worden gebruikt. Hier blijkt alvast het belang van het rendement van een voeding. Hoe hoger het rendement, des te minder verlies er is. Dit verlies manifesteert zich steeds onder de vorm van warmte.

Naast het rendement is ook de grootte van de ventilator belangrijk. Een grotere ventilator zal bij lagere toerentallen voldoende kunnen koelen en daarbij minder lawaai produceren. Er bestaan ook voedingen die volledig passief (zonder ventilatoren) gekoeld worden. Deze produceren uiteraard geen lawaai, maar zijn typisch iets duurder.

## Example 2.1. rekenvoorbeeld stroomverbruik

Een computer (inclusief scherm) die niet erg zwaar belast wordt, verbruikt ongeveer 200 Watt. Als je deze pc elke werkdag 10 uur gebruikt, dan komt het verbruik op

$$0,150 \text{ kW} \times 10 \text{ uur per dag} \times 250 \text{ werkdagen} = 375 \text{ kWh per jaar}$$

Als je daar de prijs tegenover zet die een gemiddeld gezin (bron: VREG, oktober 2012) betaalt per kWh, dan kost deze pc je  $375 * 0,2\text{€} = \text{€} 75$ . Een voeding met een rendement dat 20% beter is zal je dus op jaarrichting makkelijk 15 Euro opleveren.

Het loont dus de moeite om bij de aankoop de voeding zorgvuldig te kiezen. De meerprijs voor een duurdere PSU (Power Supply Unit) kan dus zeker renderen. In een bedrijf met honderden desktops begrijp je dat dit een verkoopsargument kan zijn.

**Het 80-plus certificatieprogramma** probeert voor de consument duidelijkheid te scheppen door voedingen een label te geven naargelang de efficiëntie. De certificatie is echter geen verplichting voor fabrikanten.

**Table 2.1. 80 plus certificatie (bron: Wikipedia<sup>2</sup> )**

	standaard	brons	zilver	goud	platinum	titanium a
20% belast	$\geq 80\%$	$\geq 82\%$	$\geq 85\%$	$\geq 87\%$	$\geq 90\%$	$\geq 94\%$
50% belast	$\geq 80\%$	$\geq 85\%$	$\geq 88\%$	$\geq 90\%$	$\geq 92\%$	$\geq 96\%$
100% belast	$\geq 80\%$	$\geq 82\%$	$\geq 85\%$	$\geq 87\%$	$\geq 89\%$	$\geq 94\%$

a bij titanium worden ook nog extra eisen gesteld

Laptops hebben een verbruik dat typisch een flink stuk lager zit. Hoewel ze een voeding hebben die meestal een behoorlijk hoog wattage aankan om de accu snel op te laden, is het gemiddeld verbruik meestal slechts rond de 30Watt. Nieuwere toestellen die een hoge autonomie tot hun belangrijkste verkoopargumenten rekenen,

<sup>2</sup> [http://en.wikipedia.org/wiki/80\\_Plus](http://en.wikipedia.org/wiki/80_Plus)

zoals chromebooks, kunnen zelfs onder maximale belasting onder de grens van 15 Watt blijven. [\[ANAND\]](#)

Het matige rendement van PSU's is voor een deel eigen aan de opbouw ervan. Omdat veel verschillende eindtrappen nodig zijn voor de verschillende spanningen, is het totale rendementsverlies een accumulatie van de kleinere verliezen bij de deeltrappen.

Ondertussen verlaten sommige grote spelers om die reden de ATX standaard om met eigen oplossingen hogere rendementen te behalen. Google ontwikkelt bijvoorbeeld z'n eigen servervoedingen die door hun eenvoud een veel hoger rendement halen. De eenvoud bestaat erin dat ze slechts 1 spanning aanbieden aan het moederbord: 12V. Als componenten een andere voedingsspanning vereisen, worden die waar nodig getransformeerd op het moederbord, wat veel efficiënter kan. Google research publiceerde een paper [\[GOOGLE\]](#) die schat dat de energiebesparing die je hiermee kan behalen op een populatie van 100 miljoen computers 13 miljard kWh betreft op jaarbasis. Dat komt, om je een idee te geven, ongeveer overeen met de opbrengst van de helft van een kerncentrale zoals die in Doel (jaarproductie 22 miljard kWh).

## 2.2.5. Problemen met voedingen

Problemen met voedingen hebben altijd gevolgen voor het volledige systeem, aangezien ze dit volledige systeem van stroom moeten voorzien. Een belangrijke oorzaak van problemen is een te klein vermogen voor het systeem of onvoldoende koeling. Dit probleem uit zich meestal niet in het niet opstarten van het systeem, maar eerder in het onverwacht afsluiten (of eventueel herstarten) ervan. Dit is dan nog het meest aangename gevolg van het probleem. Het is belangrijk om bij dergelijke problemen de voeding en de koeling ervan te controleren.

Minder aangename gevolgen kunnen zijn dat de voeding beschadigd raakt en in het meer dramatische geval dat er rook uit de computerkast komt. Deze kan dan afkomstig zijn van de voeding zelf, maar ook van andere componenten(moederbord, RAM, CPU). Een situatie die de meesten liever vermijden.

Een voeding kan ook slijtage vertonen. In het bijzonder op het vlak van de elektrolytische condensatoren kan er veel verschil zijn tussen voedingen. Minder kwalitatieve condensatoren kunnen uitdrogen (elektrolyt dat verdampst), waardoor ze hun functie minder tot niet meer vervullen en de voeding uiteindelijk rook in plaats van gelijkspanning produceert. Dit gebeurt uiteraard pas na verloop van tijd (afhankelijk van de belasting van de computer).

Sommige voedingen hebben een controlesysteem dat je door middel van geluidssignalen preventief waarschuwt als er problemen dreigen, zoals overbelasting of een gebrekkige koeling.

## 2.3. Accu's

Tegenwoordig kunnen we het niet meer hebben over computervoedingen zonder even uit te wijden over accu's. In de trend naar mobiliteit (laptops, tablets, smartphones), vormen die een onmisbare schakel.

### 2.3.1. Eigenschappen

#### Capaciteit

De capaciteit van batterijen wordt meestal uitgedrukt in Ah (ampère/uur) of mAh (milliampère/uur). Met die eenheid kan je makkelijk accu-packs vergelijken. Een batterij van 6Ah zal theoretisch bijvoorbeeld in staat zijn om gedurende 6 uur een stroom af te leveren van 1 Ampère, of gedurende bijvoorbeeld 2 uur een stroom van 3 Ampère. Uiteraard zijn er in realiteit door de fabrikant maxima gedefinieerd zodat de levensduur van de accu niet bedreigd wordt.

Sommige fabrikanten verkiezen echter om hun capaciteiten uit te drukken in Wh, wat vergelijken lastiger kan maken. Toch kan je eenvoudig omrekenen bij gelijkstroom: Je weet immers dat

$$P=U \cdot I \quad (\text{Vermogen} = \text{Spanning} \times \text{Stroom})$$

Willen we dus de stroom I(A) kennen, dan moeten we het vermogen delen door de spanning. Nemen we onderstaand voorbeeld:



**Figure 2.3. accu**

We kunnen hier dus de capaciteit in Ah bepalen door

$$I_h = P_h/U = 2,4Wh/3,6V = 0,666Ah \text{ (of } 666mAh)$$

## Aantal cellen

Een accu wordt opgebouwd uit verschillende cellen. Bijvoorbeeld bij Li-ion accu's kunnen die elk ongeveer 3V leveren. Het spreekt voor zich dat een toename van het aantal cellen zal betekenen dat de totale capaciteit ook toeneemt.

### Example 2.2. Oefening

Wat is de capaciteit van je eigen laptopaccu?

Stel dat je deze accu gebruikt om een lamp (bijvoorbeeld een powerLED) te doen branden die 5 Watt verbruikt. Hoe lang zal de lamp branden?

Uit hoeveel cellen bestaat je accu-pack?

## Laadcurve

Om de optimale kwaliteit van de accu te garanderen over langere termijn is het nodig om de juiste laadcurve te respecteren. Een batterij zal uiteraard stroom nodig hebben om zich op te laden, maar het is niet noodzakelijk zo dat een hogere stroom zal betekenen dat de batterij sneller oplaat. Het gebruik van de juiste en kwalitatieve adapter is hierbij erg belangrijk.

## Memory-effect

Het zogenaamd memory-effect is een term die vaak gebruikt wordt om aan te geven dat bepaalde types batterijen, met NiCd (Nikkel-Cadmium) op kop, vaak een effect vertonen waarbij het lijkt dat de batterijen snel hun capaciteit verliezen als je ze halverwege de ontladcyclus terug oplaat. Dat fenomeen is eigenlijk de verzamelnaam van effecten die worden veroorzaakt door een combinatie van elektrische en chemische processen.

## LI-ION accu's

Tegenwoordig is dit zowat het meest voorkomende type in hoogwaardige mobiele apparatuur. Dit type onderscheidt zich door een erg hoge energiedichtheid, en het 'memory-effect' is niet bestaande.

Toch zijn er enkele belangrijke eigenschappen aan dit type, die je beter kent.. Het zwakke punt van Li-Ion: degradatie

Wie een laptop of gsm gebruikt, kent het fenomeen: na enkele jaren is de capaciteit van de batterij slechts nog een fractie van wat ze was bij aankoop. Dit fenomeen kan je niet omkeren, maar het kan wel vertraagd worden als je weet wat de factoren zijn die dit proces versnellen...

Een Li-ion-accu verliest zijn capaciteit het snelst als hij zich in een warme ruimte bevindt, en opgeladen is. Een volledig opgeladen Li-ion accu zal bijvoorbeeld na een jaar rusten in een ruimte waar het gemiddeld 20°C is, 20 procent van zijn capaciteit verliezen.

Is diezelfde accu slechts half opgeladen, dan zal de capaciteit met slechts enkele procenten dalen. Het is dus niet verstandig een Li-ion-accu voor lange tijd weg te bergen in opgeladen toestand. Ook door stockage in koele ruimtes kan de capaciteit langer bewaard blijven. Een laptop die snel erg warm wordt bij gebruik zal dus meteen ook nefast zijn voor de capaciteit van de batterij op langere termijn.

Bij een temperatuur van iets boven het vriespunt en een lading van ongeveer 40% zal dit type batterij de langste levensduur ‘on the shelf’ hebben.

## Toekomstige ontwikkelingen

Gezien de enorme markt die ontstaan is voor accu’s, is er enorm veel druk om betere modellen te ontwikkelen. Daarbij worden bestaande types geperfectioneerd, maar ook nieuwe types ontwikkeld. Zo zijn er de **LiPo** (Lithium polymeer) batterijen die ongeveer 50% efficiënter zijn dan klassieke Li-Ion equivalenten, en ook de brandstofcellen (fuel cells) die mogelijks een oplossing kunnen vormen voor de steeds grotere autonomiebehoefte van toestellen. Omdat veel van deze technieken gebruik maken van erg zeldzame delfstoffen, komen ook geavanceerde technieken met courante materialen in het vizier ter optimalisatie of vervanging, zoals nanostructuren met koolstof. Deze blijven echter toekomstmuziek voor consumentenelektronica...

## 2.4. Uninterruptible Power Supply (UPS)

Een UPS is een toestel dat het wegvallen van de netspanning kan opvangen. Hiervoor bestaat een UPS uit een accu en een elektronische schakeling die de accuspanning kan omzetten naar een netspanning.



**Figure 2.4. Rack-mountable UPS (bron: [www.apc.com](http://www.apc.com)<sup>3</sup>)**

Bij het wegvallen van de netspanning zal de UPS ogenblikkelijk de stroomvoorziening overnemen. Voor de aangesloten toestellen treedt er dus geen onderbreking op. Een UPS kan de stroom natuurlijk niet onbeperkt in de tijd overnemen. Hoe lang de UPS dit kan volhouden, hangt af van de accu's en het gevraagde vermogen. Om te vermijden dat apparatuur plotseling en ongecontroleerd stilvalt, heeft een UPS dikwijls ook een interface naar de computer. Deze laat toe dat de UPS de computer 'proper' afsluit op het ogenblik dat de accu-stroom een bepaalde ondergrens bereikt. Een alternatief kan erin bestaan dat de UPS gecombineerd wordt met een dieselgenerator.

De UPS zorgt dan voor de ogenblikkelijke overname van de stroomvoorziening en geeft de generator de nodige tijd om op te starten. Zodra de generator actief is (en de uitgangsspanning gestabiliseerd is), neemt deze de stroomvoorziening op zich.

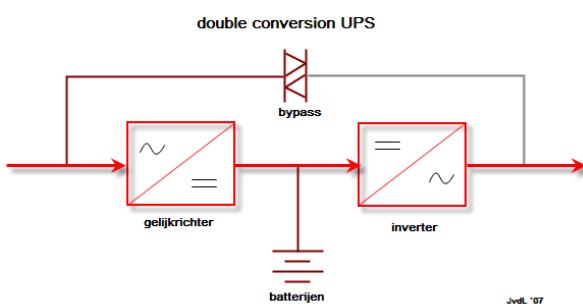
Een UPS heeft meestal ook een spanningsbeveiliging aan boord die je apparatuur kan beschermen tegen storingen op het elektriciteitsnet. UPS'en vind je in alle prijsklassen, wat vaak te maken heeft met de inwendige opbouw ervan. Er onderscheiden zich enkele grote types.

---

<sup>3</sup> <http://www.apc.com>

### 2.4.1. Online UPS

De online UPS wordt ook wel “double conversion” UPS genoemd. Alle stroom die naar de IT-apparatuur gaat, loopt door de UPS. Hierdoor is het niet nodig om over te schakelen bij het uitvallen van de stroom. Met de bypass kan je evenwel de ups overbruggen. Dat kan bijvoorbeeld interessant zijn als er onderhoud nodig is. Omdat hierdoor veel gevraagd wordt van alle elektronica (die constant volledig belast wordt), is dit een relatief duur concept.

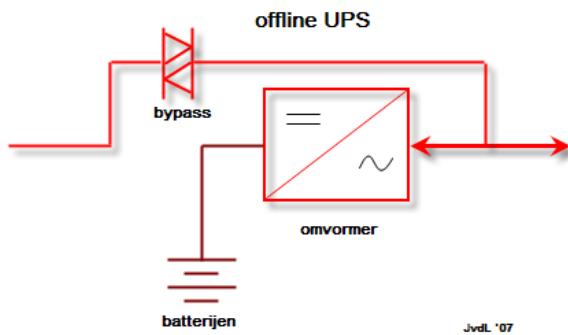


**Figure 2.5. Online UPS ( © GFDL Joslee 2007 )**

### 2.4.2. Offline UPS

Dit type UPS vind je voornamelijk terug bij particuliere ups'en waar kostprijs een belangrijk criterium is. Bij het wegvalLEN van de spanning, wordt een bypass ingeschakeld. Die procedure duurt enkele milliseconden waarbij je geen uitgangsspanning hebt, en dat moet opgevangen worden door de voeding van je computer of server. Een nadeel van dit type UPS is dat je hem ook niet zonder risico kan testen.

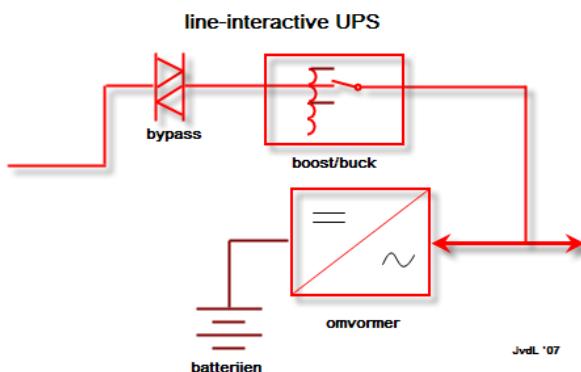
Een ander nadeel is dat in gewone omstandigheden de netspanning rechtstreeks gekoppeld is aan je IT-apparatuur. Als er storingen op het net zitten, zal je IT apparatuur daar hinder van ondervinden. De apparatuur is dus niet beveiligd.



**Figure 2.6. Offline UPS ( © Joslee 2007 GFDL)**

### 2.4.3. Line-interactive UPS

Deze vorm van UPS vormt een hybride oplossing. In feite gaat het om een off-line UPS waar de line-feed voorzien is van aanvullende filters. Zo ben je zeker dat de spanning die aan je servers aangelegd is, gezuiverd werd van pieken en storingen. In omgevingen waar veel storing optreedt is dat geen overbodige luxe. (bijvoorbeeld fabriekshallen, gebieden met gebrekkige stroomvoorziening)



**Figure 2.7. Line interactive UPS ( © Joslee 2007 GFDL)**

## 2.5. Bibliografie bij dit hoofdstuk

[GOOGLE] Google. High efficient power supplies for home computers and servers. 2006. [http://static.googleusercontent.com/media/services.google.com/nl/blog\\_resources/PSU\\_white\\_paper.pdf](http://static.googleusercontent.com/media/services.google.com/nl/blog_resources/PSU_white_paper.pdf)

[ANAND] AnandTech. Samsung Chromebook Review. <http://www.anandtech.com/show/6422/samsung-chromebook-xe303-review-testing-arms-cortex-a15/>

# Chapter 3. CPU

## 3.1. Overzicht

In onderstaande tabel worden een aantal processoren van de x86 familie weergegeven met hun belangrijkste eigenschappen. De processor die in de eerste (IBM-)PC werd gebruikt was een 8088. Eigenlijk was dit een 8086 waarvan de databus beperkt werd tot 8 bits in plaats van 16 bits. De enige reden hiervoor was dat op dat ogenblik er geen andere 16bit componenten beschikbaar waren. Deze processorfamilie komt uitgebreid aan bod binnen het vak processorarchitectuur. Het is niet de bedoeling om hier terug te komen op programmeermodel, segmentering, ...

Wel zullen we de evolutie van een aantal eigenschappen bekijken.

**Table 3.1. processoroverzicht**

type	jaar	data/ adres- bus	L1 cache (kB)	FSB (Mhz)	Clock(Mhz)	transistoren (miljoen)	technologie (nm)
8088	1979	8/20	-	4,77..8	4,77..8	0.029	3000
8086	1978	16/20	-	4,77..8	4,77..8	0.029	3000
80286	1980	16/24	-	6..20	6..20	0.134	1500
80386DX	1985	32/32	-	16..33	16..33	0.275	1500
80486DX/ SX	1989	32/32	8	25..50	25..50	1.2	1000
80486DX2	1992	32/32	8	25..40	25..80	1.2	800
80486DX4	1994	32/32	8+8	25..40	75..120	1.2	600
Pentium	1993	64/32	8+8	60..66	60..200	3	600
Pentium MMX	1997	64/32	16+16	66	166..233	4.5	350
Pentium Pro	1995	64/36	8+8	66	150..200	5.5	350
Pentium II	1997	64/36	16+16	66/100	300..450	7.5	250
Pentium III	1999	64/36	16+16	100/133	450..1300	28	130

## CPU

---

type	jaar	data/ adres- bus	L1 cache (kB)	FSB (Mhz)	Clock(Mhz) (miljoen)	transistoren (miljoen)	technologie (nm)
AMD Athlon	1999	64/36	64+64	200/266	500..2200	37	130
Pentium IV	2001	64/36	8+96	400/533	1400..2800	42	130
AMD 64	2005	64/36	2*512k L2	2000	2,4GHz	233	102
Core duo	2006	64/36	2*2M L2	800	3,6GHz	376	65
Intel Nehalem	2008	64/36	32+32/ core	-	3,2 Ghz	731 (QC)	45/32
Intel Sandy Bridge	2011	64/36	32+32/ core	-	3,8 Ghz. <sup>a</sup>	995 (QC)	32
Intel Ivy bridge	2012	64/36	32+32/ core	-	3,9 Ghz. <sup>a</sup>	1400 (QC)	22
Intel Haswell	2013	64/36	32+32/ core	-	3,9 Ghz. <sup>a</sup>	1400	22

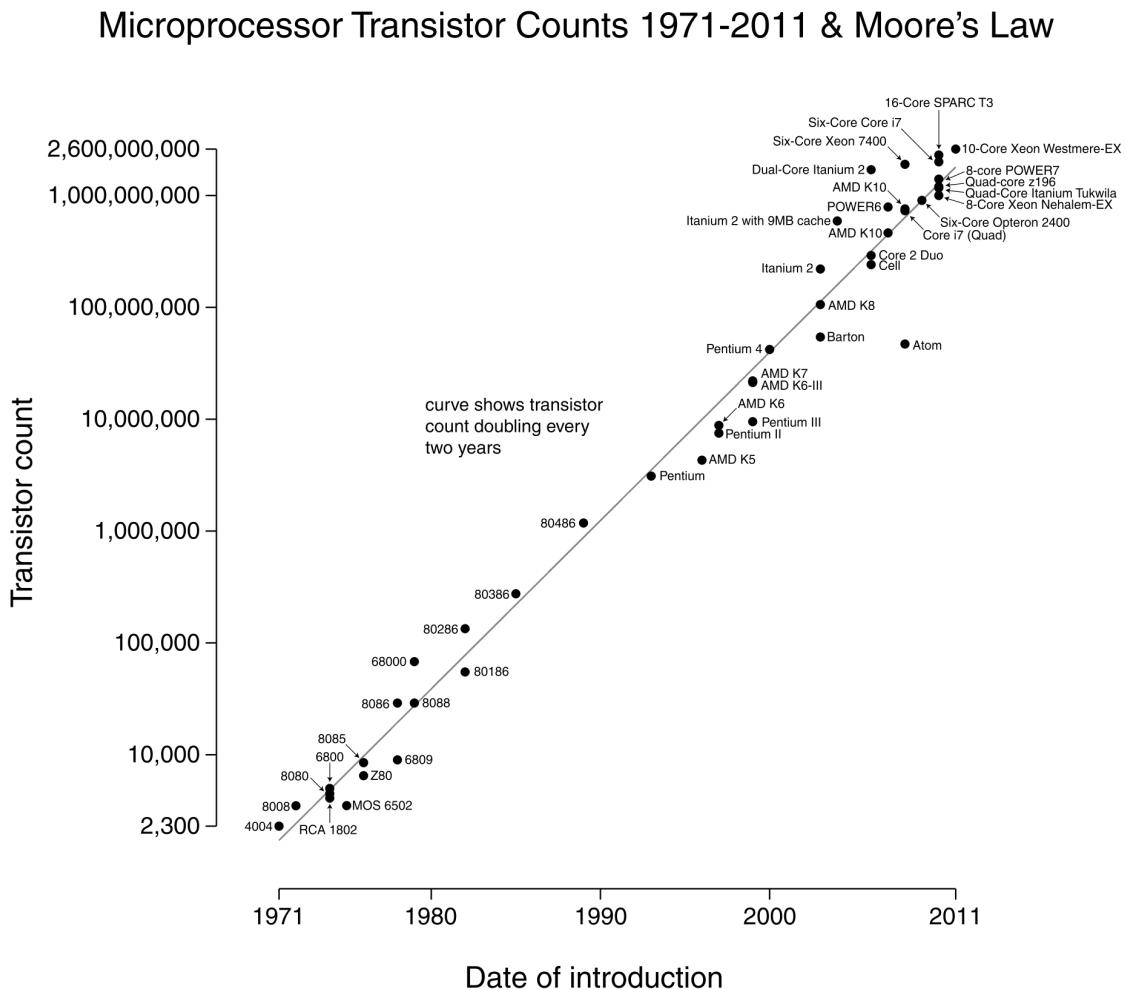
<sup>a</sup> deze waarden zijn niet continue en kunnen pas tijdelijk gehaald worden met technologieën als Turboboost.

### 3.2. Technologie en functionaliteit

Een eerste duidelijke evolutie is de toename van het aantal transistors. Volgens de wet van Moore verloopt deze stijging zelfs exponentieel. Elke vierentwintig maanden zou het aantal transistors in een processor verdubbelen. Die toename is uiteraard enkel mogelijk als de transistordichtheid kan toenemen. In het verleden werd hierbij vaak gedacht dat er technische beperkingen zouden opduiken, maar tot dusver blijven fabrikanten slagen om vast te houden aan de ontwikkelingsnelheid die geponeerd werd door Gordon Moore, één van de oprichters van Intel.

The number of transistors incorporated in a chip will approximately double every 24 months

— Gordon Moore *Electronics Magazine* 1965



**Figure 3.1. Wet van Moore (CC, Wikimedia Commons)**

Met deze stijging van het aantal transistoren gaat uiteraard ook een toename in functionaliteit gepaard. Zo kent een x86 processor vanaf de 80286/80386 (in principe vanaf de 286, praktisch vanaf de 386) twee werkingsmodi: real mode en protected mode.

In real mode heeft de cpu dezelfde functionaliteit als een 8086. In deze compatibiliteitsmodus gedraagt hij zich met andere woorden als een snellere versie van de 8086. In protected mode krijgt de processor extra functionaliteit. De naam protected mode komt van de extra toevoegingen op het vlak van geheugenbescherming. Daarnaast ondersteunde de processor vanaf deze modus een aantal, vandaag onmisbare, extra mogelijkheden.

Onder andere multitasking en virtueel geheugen zijn enkel mogelijk met een protected mode processor. Hier moet nog opgemerkt worden dat processoren nog steeds

opstarten in real mode. Het is de taak van het besturingssysteem (of beter de loader ervan) om de processor om te schakelen naar protected mode. Andere voorbeelden van extra functionaliteit zijn de integratie van functies die eerst door externe componenten werden vervuld. Bijvoorbeeld werd vanaf de 486 een floating point unit in de processor geïntegreerd. Een ander voorbeeld zijn cache geheugens. De extra functionaliteit uit zich ook op het vlak van de instructieset. Zo zijn in de loop der tijden een aantal extra instructies toegevoegd om aan bepaalde behoeften te voldoen. Een belangrijk voorbeeld zijn de instructies die het gebruik van multimedia moeten ondersteunen (bijvoorbeeld MMX, SSE, 3DNow) en de ondersteuning voor virtualisatie (bijvoorbeeld Intel VT-d, Amd-V).

Software die gebruik maakt van dit soort instructies, kan uiteraard niet uitgevoerd worden op processoren die deze instructies niet ondersteunen.

Intel, de grootste producent van x86 processoren, hanteert voor de ontwikkeling een model dat het tick/tock-model genoemd wordt. Afwisselend worden nieuwe modellen uitgebracht met nieuwe functionaliteit (tock) en verbeterde technologie (tick). Dit wordt duidelijk in volgende intel roadmap.

### **Figure 3.2. Intel roadmap (copyright 2008-2012 WhiteTimberwolf GFDL )**

### **3.3. Kloksnelheid**

Een tweede evolutie is waar te nemen op het vlak van de kloksnelheid, die duidelijk toegenomen is. In de beginjaren van de pc was te zien dat CPU klok en FSB klok samen toenamen. Na verloop van tijd ontstaat er een verschil tussen de processorklok en de FSB klok, die nog wel stijgt, maar een stuk minder snel. De processor wordt met andere woorden duidelijk het snelste onderdeel in het computersysteem. Het zal erop aan komen de werkkracht van de CPU zo weinig mogelijk onbenut te laten. In het bijzonder zullen maatregelen genomen moeten worden om zo weinig mogelijk tijd te verliezen bij het wachten op tragere componenten. De trend naar steeds stijgende kloksnelheden is tijdens het laatste decennium afgangen. Bij de Pentium 4 werd nog volop gemikt op de 4GHz grens, waar enige jaren tegenaan gebotst werd.

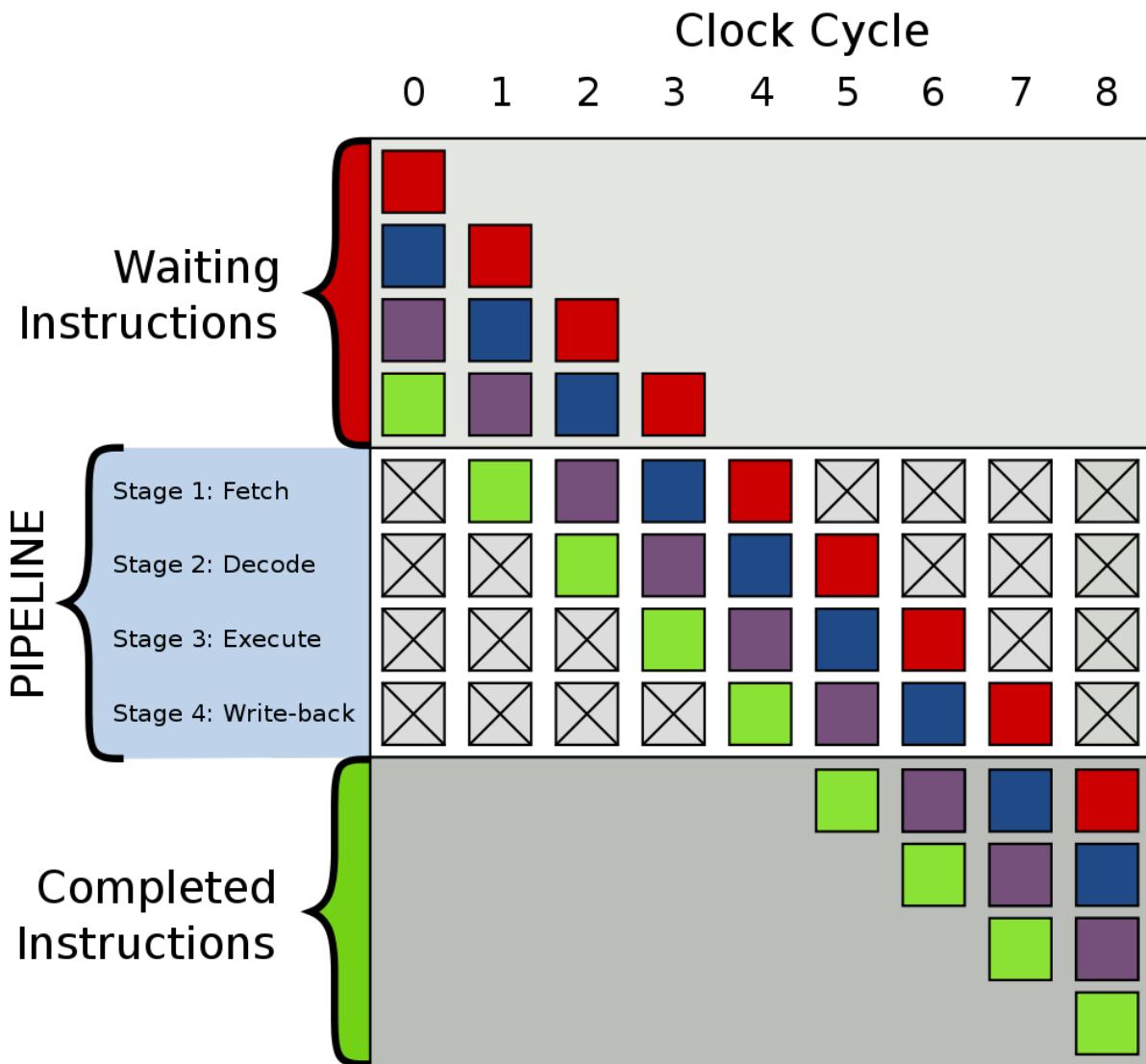
Het belangrijkste probleem bij steeds hogere kloksnelheden is de gegenereerde warmte. Die moet in de eerste plaats uiteraard afgevoerd worden, maar geeft daarnaast ook nog een hoger verbruik.

In het bijzonder bij laptops zijn dit twee vervelende problemen: de warmteafvoer vraagt grotere (en dus zwaardere) koellichamen en ventilatoren. Extra verbruik verkleint uiteraard de autonomie van een draagbaar toestel (tijd dat op accu gewerkt kan worden).

## 3.4. Processorarchitectuur

### 3.4.1. Pipelining

Naast de kloksnelheid werd ook aan de interne opbouw van de processor gesleuteld om hem sneller bepaalde taken te laten uitvoeren. Zo werken processoren instructies niet na elkaar af, maar gedeeltelijk tegelijkertijd. Dit gebeurt in een zogenaamde pipeline. Het is eenvoudig om in te beelden dat terwijl de ene instructie uit het geheugen wordt opgehaald, een andere gedecodeerd kan worden en van nog een andere het resultaat berekend kan worden. Hieronder wordt dit principe grafisch voorgesteld. Helaas is dit principe niet zalmakend: soms zijn instructies afhankelijk van andere, waardoor er een ‘bubble’ optreedt: een tijdspanne waarin de processor verplicht moet wachten.

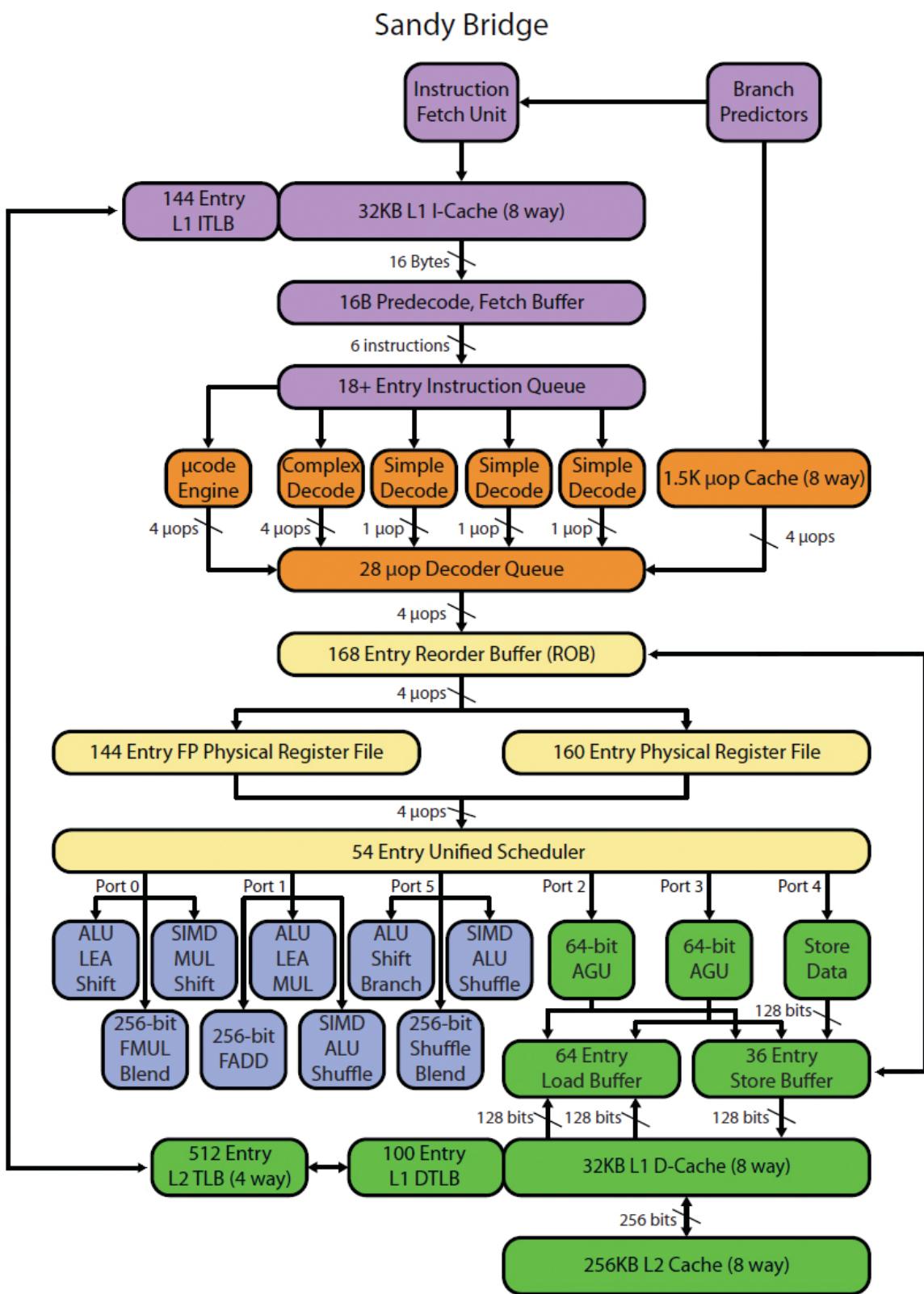


**Figure 3.3. processor pipeline (CC mediawiki)**

### 3.4.2. Superscalaire processoren

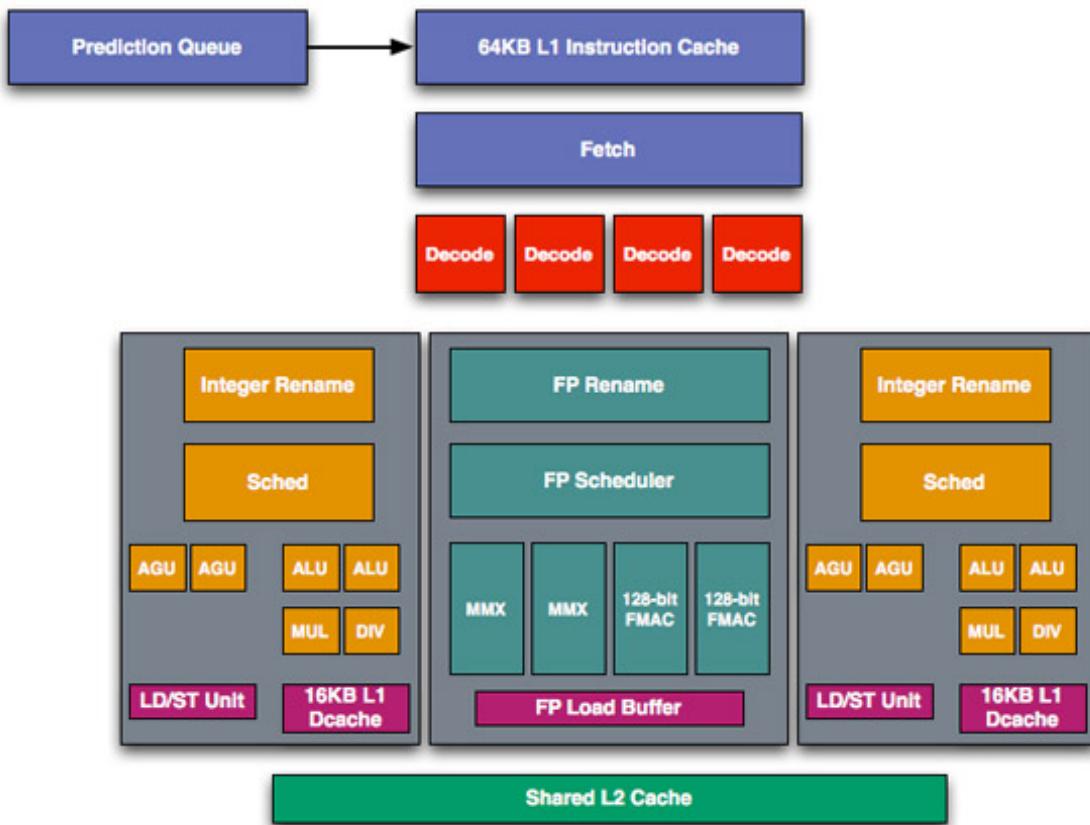
Als dit principe verder gedreven wordt, kunnen stappen die veel tijd in beslag nemen dubbel uitgevoerd worden. Men spreekt dan over een superscalaire processor. In onderstaande afbeeldingen worden de blokschema's getoond van de Sandy bridge en de Athlon Bulldozer microarchitectuur. In deze blokschema's is duidelijk te zien hoe er verschillende eenheden zijn die berekeningen kunnen maken, waardoor verschillende instructies tegelijkertijd uitgevoerd kunnen worden. Een belangrijke uitdaging hierbij vormen voorwaardelijke spronginstructies. Aangezien pas bij de uitvoering van de instructie geweten is of de sprong uitgevoerd wordt of dat gewoon de volgende instructie wordt uitgevoerd. In het schema zijn hiervoor branch prediction eenheden voorzien.

Meer details over hun werking en de principes van pipelining en superscalaire architecturen krijg je in het vak "microprocessoren".



**Figure 3.4. Sandy bridge microarchitectuur**

## Bulldozer Microarchitecture



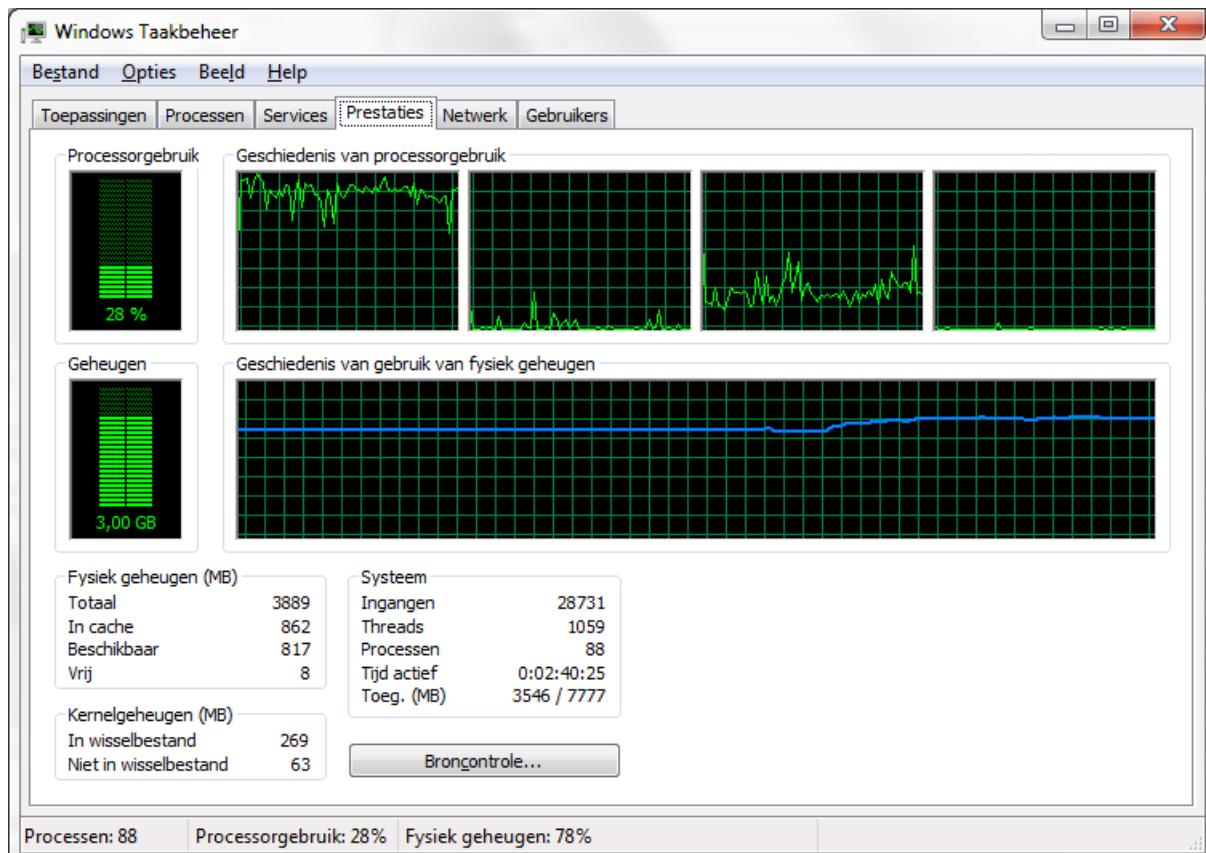
**Figure 3.5. AMD bulldozer architectuur (copyright AnandTech)**

Multicore processoren zijn al geruime tijd niet meer weg te denken. Hexacores en octocores zullen de komende jaren eerder regel dan uitzondering worden. Je zou dit een verder doorgedreven vorm van een superscalaire architectuur kunnen noemen. In plaats van delen van de processor te ontdubbelen, wordt een volledige processor ontdubbeld. Een grote moeilijkheid bij deze werkwijzen is om de caches op elkaar af te stemmen. Een probleem dat duidelijker zal worden in het volgende hoofdstuk.

Net zoals een superscalaire architectuur pas voordeel geeft als de verschillende enheden tegelijk gebruikt worden, zal een dual core pas voordeel geven als meerdere cores tegelijk werk verrichten. Dit kan als er bijvoorbeeld verschillende programma's tegelijk actief zijn of als de software zodanig geschreven is, dat ze bestaat uit verschillende threads die naast elkaar (en dus tegelijk door verschillende processorkernen) kunnen uitgevoerd worden.

Een simpel voorbeeldje om de beperkingen van een multicore processor aan te tonen: als je een eenvoudige toepassing een rekenintensieve opdracht laat uitvoeren, dan zal

een multicore processor slechts een deel belast worden. Eén processorkern verricht namelijk al het werk.



**Figure 3.6. single threaded applicatie op multicore processor**

Het voordeel van de multi-core merk je pas als je tegelijk nog een ander programma probeert te gebruiken. Dat zal met een multi-core vlot lukken, in tegenstelling tot een single core. Het OS zal op zoek gaan naar de minst belaste core, en het nieuwe proces daarop uitvoeren.

Uiteraard is dit principe nog verder schaalbaar. In servers worden vaak meerdere processoren op één moederbord geplaatst, en als ook dat niet langer volstaat, wordt de rekenkracht van meerdere servers gecombineerd. Deze principes overstijgen deze cursus, en komen later in de opleiding aan bod.

### 3.4.3. Cache

Een andere eigenschap die plots opduikt en doorheen de processorgeschiedenis steeds toeneemt is het cache geheugen. De toename van het cache volgt de trend van alle soorten geheugens die in een pc te vinden zijn. Dit is een gevolg van de eerder opgemerkte trend dat de processor veruit het snelste onderdeel is in het systeem, dat zo optimaal mogelijk benut moet worden. Naarmate data en programma's steeds

groter werden, werd ook het belang van geheugen groter. Tot de intrede van de grafische interface was de belangrijkste parameter in het systeem de kloksnelheid van de processor. Met de intrede van de grafische interface was een groter geheugen soms te verkiezen boven een hogere kloksnelheid. Het belang van cache geheugen is ook duidelijk als je de budget- en performance-processoren van fabrikanten met elkaar gaat vergelijken.

In onderstaand lijstje staan enkele desktop en serverprocessoren opgeliist. Je merkt dat ze qua kloksnelheid niet voor elkaar moeten onderdoen, maar dat de hoeveelheden cache wel verschillen.

De werking van de cache wordt verder in detail besproken in het derde hoofdstuk.

**Table 3.2. Cache in desktop en serverprocessoren (actuele topmodellen, feb 2014)**

CPU	doel	cache	maxCPU	#cores/threads
Atom 7560	mobile	512KB	2.13 Ghz	1/2
i7-4771	desktop	8MB	3.50 Ghz	4/8
E7-8893v2	server	37.5MB	3.40 Ghz	15/30

### 3.5. APU, SoC

De wet van Moore impliceert dat steeds meer mogelijk is op eenzelfde oppervlakte substraat. Die ruimte wordt ingenomen door bijvoorbeeld meerdere cores te huisvesten op eenzelfde processor, maar dat is maar een deel van het verhaal.

Het is namelijk ook zo dat men probeert om steeds meer functionaliteit die voorheen op andere plaatsen op het moederbord te vinden was, te verzamelen op eenzelfde chip.

Daar zijn een aantal goede redenen voor te bedenken:

- het aantal verschillende chips (en dus kostprijs) op een moederbord kan zo teruggedrongen worden
- als alle componenten dicht bij elkaar zitten, zijn geen 'trage' bussen nodig tussen deze onderdelen
- de oppervlakte die nodig is om het systeem te bouwen verkleint zo, een belangrijk argument bij de ontwikkeling van mobile devices.

Bij recente processoren zit bijvoorbeeld steeds vaker een grafische chip ingebouwd. Dan spreekt men niet meer over CPU, maar over APU (=advanced processing unit) om dit verschil in de verf te zetten.

Het integratieproces gaat soms zo ver dat je kan spreken van een *System On A Chip*: alle belangrijke onderdelen (cpu, gpu, IO) zitten dan verzameld op één enkele chip. De rol van secundaire chips ("de chipset") wordt dus steeds kleiner.

### Example 3.1. oefening

Op welke SoC is jouw telefoon gebaseerd?

## 3.6. Montage

Bij de montage van een processor moet je enkele zaken in acht nemen.

- De processor moet compatibel zijn met het moederbord. Meestal kom je dit te weten door de socket van de processor te vergelijken met die van het moederbord.
- De processor plaatsen moet gebeuren zonder het uitoefenen van kracht: de processor valt normaalgezien in z'n socket (ZIF: zero insertion force), waarna je hem kan inklemmen.
- Mobiele processoren zijn vaak vast op het moederbord gemonteerd, vervangen is dan onmogelijk.



**Figure 3.7. LGA2011 socket zonder processor**

Tweede belangrijk aandachtspunt bij de installatie van een processor is dat gezorgd moet worden voor voldoende koeling. Dit betekent dat gezorgd moet worden voor een

voldoende grote koelvin en ventilator en dat er goed contact is tussen de chip en de koelvin. Hiervoor moet eventueel koelpasta aangebracht worden. Een slecht gekoelde processor kan aanleiding geven tot een instabiel werkende computer en in het meest dramatische geval tot een beschadigde processor.

### 3.7. Processoren van de toekomst

Voorspellingen maken is geen sinecure. De trends die ingezet zijn, zullen vermoedelijk nog een hele poos verder gaan, met een verdere miniaturisatie en toename van efficiëntie tot gevolg. Een kaper op de kust voor de x86 technologie die momenteel monopolist is op de PC-markt, is de ARM architectuur. Hoewel deze absoluut niet nieuw is (eerste ontwerpen midden jaren 80), biedt deze processorfamilie grote voordelen:

- Deze architectuur is steeds ontworpen voor toestellen met een laag verbruik. Het succes op de mobiele markt (iPAD2,3, nagenoeg alle android smartphones, consumer elektronica, ...)
- Deze architectuur is in licentie bij de meeste chipbakkers
- Door een RISC (Reduced instruction set computing) architectuur van nature efficient

De kans dat de RISC architectuur op korte termijn succesvol wordt op de desktopmarkt is gering, en ook het omgekeerde kan gezegd worden over CISC (x86) op mobiele devices. Voor specifieke servertoepassingen zijn er wel [aankondelingen gebeurd](#)<sup>1</sup> door bijvoorbeeld AMD, dat zich hier sterk wil in specialiseren en profileren.

Toch lijken deze twee werelden van de gespecialiseerde ARM-architectuur en de meer universele x86 naar elkaar toe te groeien, en zullen de grenzen ongetwijfeld snel vervagen. Microsoft heeft bijvoorbeeld eind 2012 z'n RT tablet vrijgegeven, met ARM SOC. Uiteraard zal software die gecompileerd werd voor x86 op dit soort toestellen niet werken. Ook Google Chromebooks worden zowel met ARM als met x86 gebouwd.

### 3.8. Bibliografie bij dit hoofdstuk

[INTEL] Intel. <http://www.intel.com/content/www/us/en/history/museum-gordon-moore-law.html>.

---

<sup>1</sup> <http://www.anandtech.com/show/7724/it-begins-amd-announces-its-first-arm-based-server-soc-64bit8core-opteron-a1100>

---

# Chapter 4. Het Geheugen

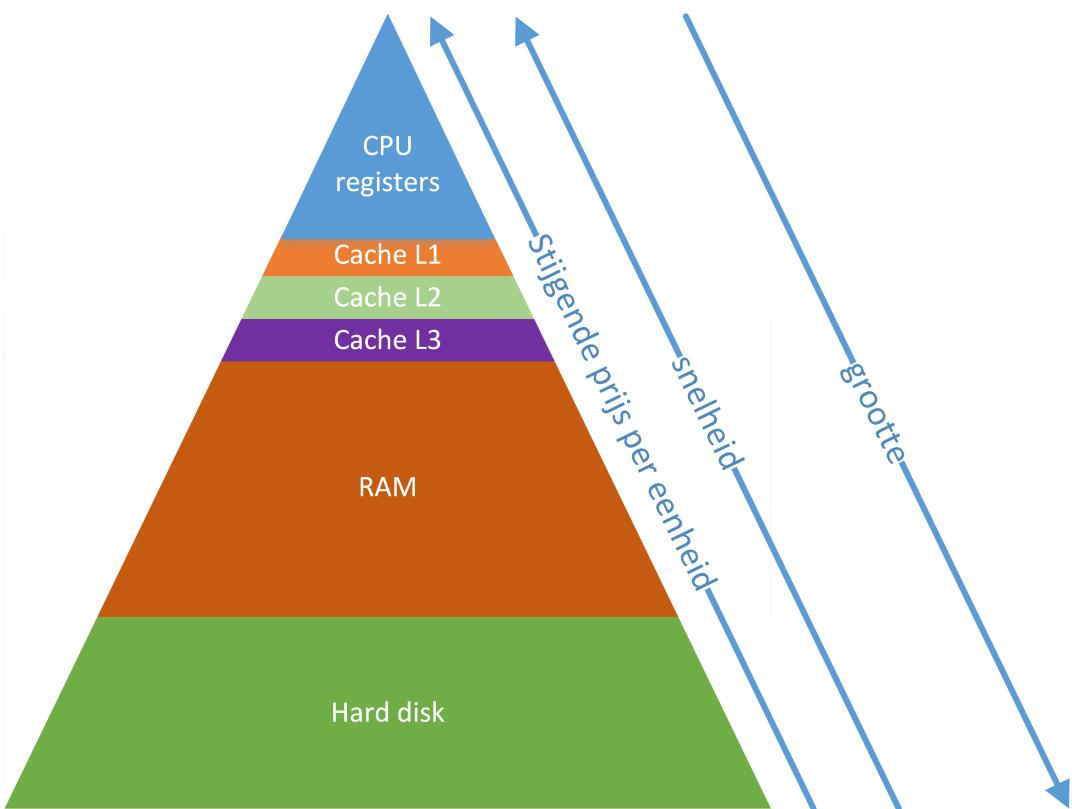
## 4.1. Overzicht

Het belang van geheugen is tijdens de cursus computertechniek al voldoende gebleken. Zoals bekend worden zowel uit te voeren instructies als data opgeslagen in dit geheugen. (cfr de 'von Neumann architectuur') Dit legt twee belangrijke behoeften bloot: snelheid en grootte.

**Snelheid** is erg belangrijk omdat het geheugen de processor moet voorzien van uit te voeren instructies. Een snelle processor met traag geheugen geeft een traag systeem.

**Grootte** is dan weer belangrijk omdat zowel programma's als de te bewerken data aanzienlijk zijn toegenomen. Bovendien moeten in een multitasking omgeving meerdere programma's en dus ook grotere hoeveelheden data opgeslagen worden in het geheugen.

Als de verschillende soorten geheugens bekeken worden, wordt ook wel gesproken van de geheugenpiramide. Deze term wordt gebruikt omdat de hoeveelheid geheugen afneemt naarmate je stijgt in de piramide. De reden hiervoor is dat ook de prijs per byte toeneemt naarmate je hoger gaat in de piramide.



**Figure 4.1. Geheugenpiramide**

Daarnaast neemt de snelheid van het geheugen toe in de richting van de top van de piramide. In deze voorstelling is het begrip geheugen vrij ruim geïnterpreteerd. We zullen het in dit hoofdstuk niet hebben over registers of opslagmedia, wel over de technieken die toegepast worden om het geheugen voldoende snel en groot te maken.

## 4.2. Lokaliteitsprincipes

Bij het bestuderen van de technieken die gebruikt worden om het geheugen groot en snel te maken, zullen we regelmatig beroep doen op de zogenaamde lokaliteitsprincipes. Daarbij maken we een onderscheid tussen plaatsgebonden lokaliteit en tijdsgebonden lokaliteit. [PATT1]

### Tijdsgebonden lokaliteit

Als je een bepaald item gebruikt, dan is de kans groot dat je dat binnenkort terug nodig hebt.

### Plaatsgebonden lokaliteit

Als je een bepaald item gebruikt, dan is de kans groot dat je binnenkort iets nodig hebt dat in de buurt voorkomt...

Een paar voorbeeldjes bij deze principes:

- Sequentiële uitvoering van instructies in een programma. (plaatsgebonden)
- Lussen in code, er is een regel die zegt dat programma's 90% van hun tijd spenderen met het uitvoeren van 10% van de code, dus worden regelmatig dezelfde stukken geheugen aangesproken (tijdsgebonden & plaatsgebonden)
- Bestanden op een schijf worden, indien mogelijk, in opeenvolgende clusters opgeslagen. (plaatsgebonden)
- bij een computerprogramma zullen bepaalde variabelen erg vaak gebruikt worden (tijdsgebonden)

### Example 4.1. oefening

op welke manieren komen deze principes tot uiting in onderstaande java-code?

```
public class Vermenigvuldigingstafels {  
    int score;  
  
    public void test() {  
        for (int i = 1; i <= 10; i++) {  
            vraagEnAntwoord();  
        }  
  
        drukResultaat();  
    }  
  
    public void vraagEnAntwoord() {  
        int getal1 = (int)(Math.random() * 11);  
        int getal2 = (int)(Math.random() * 11);  
  
        System.out.print(getal1 + " * " + getal2 + " = ");  
  
        if (Input.readInt() == getal1 * getal2) {  
            score++;  
        } else {  
            System.out.println("fout");  
        }  
    }  
  
    public void drukResultaat() {  
        System.out.println("Je score: " + score + "/10");  
    }  
}
```

```
public static void main (String args[]) {  
    Vermenigvuldigingstafels tafels = new Vermenigvuldigingstafels();  
    tafels.test();  
}  
}
```

- 
- ❶ variabelen worden typisch meerdere keren opgevraagd tijdens de duur van een programma. (=tijdsgebonden lokaliteit)
  - ❷ een lus zorgt ervoor dat binnen bepaalde tijd code meerdere keren uitgevoerd wordt. (=tijdsgebonden lokaliteit)
  - ❸ deze regels code worden na elkaar uitgevoerd. Er is dus plaatsgebonden lokaliteit.

### Example 4.2. oefening

Uiteraard speelt de programmeertaal geen rol bij dit principe. Kan je de dit aantonen met onderstaand assembler-fragment?

---

```
; x86 ASM code  
  
.model small  
.stack  
.data  
.code  
  
start:  
    mov ah,01h ; karakter inlezen  
    int 21h     ; resultaat in AL  
    mov dl,31h ; cijfer 1  
    mov bl,al   ; AL wordt beïnvloed door int 21h verderop  
    inc bl ; eentje bijtellen  
lus:  
    mov ah,02h ; karakter schrijven  
    int 21h  
    inc dl     ; teller verhogen  
    cmp dl,bl  ; vergelijken met eindwaarde  
    jnz lus    ; conditioneel springen  
  
    mov ah,08h ; karakter inlezen  
    int 21h  
    mov ah,4Ch ; afsluiten  
    int 21h  
end start
```

## 4.3. Soorten geheugens

Tussen de verschillende soorten geheugens kan een onderscheid gemaakt worden op een aantal vlakken.

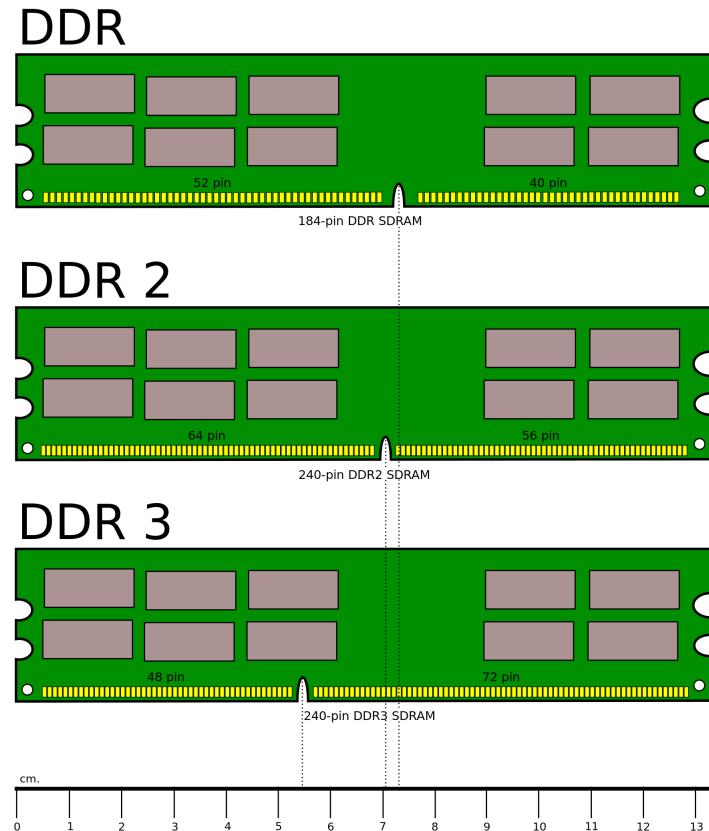
### 4.3.1. Behuizing

Het meest tastbare onderscheid kan gemaakt worden op het vlak van de behuizing. Origineel gebruikte men op de PC geheugen onder de vorm van discrete chips. Naarmate de capaciteit van het geheugen steeg, werd dit te duur en ging men over op geheugenmodules.

Daarnaast spreekt men soms van het gebruik van geheugenbanken. Een geheugenbank op een moederbord bestaat uit een of meerdere sockets of geheugenvoeten (insteekplaatsen voor geheugenchips). Het aantal sockets per geheugenbank hangt af van de uitvoeringsvorm van het gebruikte geheugen en van de breedte van de databus. Een geheugenbank heeft dezelfde breedte als de databus die voor de aansluiting op de datalijnen zorgt.

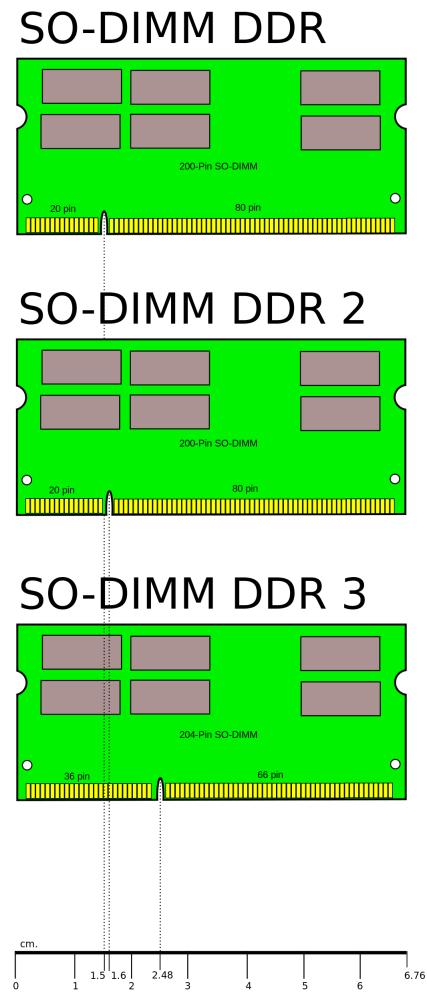
Niet alle geheugenbanken moeten gevuld zijn maar iedere geheugenbank waar geheugen in geplaatst werd, moet volledig gevuld zijn. In moderne systemen vult een module een volledige geheugenbank en is deze dus automatisch vervuld. Een geheugenmodule wordt gekenmerkt door het aantal contactpunten (pins), de werkspanning en het soort geheugenchip. Het is de geheugencapaciteit van alle chips samen die de capaciteit van de module bepalen.

Langs beide zijden van een geheugenmodule bevinden zich contactpunten. Indien deze contactpunten inwendig verbonden zijn spreekt men van een SIMM (Single Inline Memory Module). Indien deze contactpunten afzonderlijk werken en niet verbonden zijn spreekt men over een DIMM (Dual Inline Memory Module). Een DIMM biedt op dezelfde afstand veel meer contactpunten en wordt dan ook toegepast in de moderne modules, die onder andere voor de steeds breder wordende databussen extra contactpunten nodig hebben.



**Figure 4.2. Courante DDR-dimm modules (Wikimedia public domain)**

Een variante van DIMM is so-DIMM (small outline), een miniatuurversie van DIMM, specifiek geschikt voor mobiele apparatuur als laptops.



This dimensions are for reference to give a general idea.  
This is not an exact technical diagram. Standards may vary between manufacturers

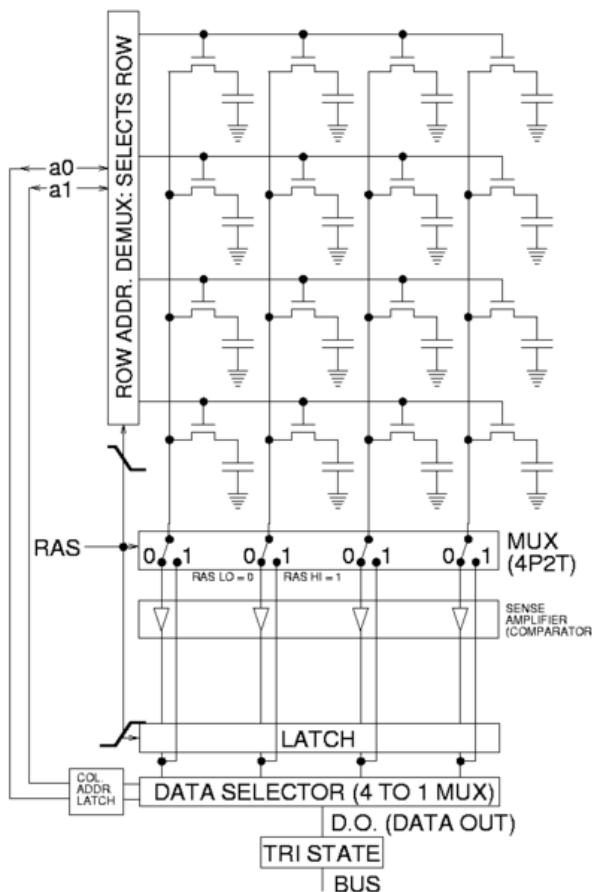
**Figure 4.3. SO-DIMM modules (Wikimedia public domain)**

## 4.4. Technologie

Een zeer belangrijk onderscheid tussen geheugens kan gemaakt worden op het vlak van de technologie die gebruikt werd om de geheugencellen te bouwen. Er zijn twee soorten: statisch en dynamisch geheugen. Statisch geheugen is opgebouwd uit actieve geheugencellen (flipflop schakelingen). Deze vragen een grotere complexiteit bij het IC ontwerp en zijn dus duur. Anderzijds zijn ze zeer snel. Uit deze eigenschappen kan je afleiden dat ze hoog in de piramide worden toegepast. Meer bepaald gebeurt dit bij de snelle cache geheugens.

**Figure 4.4. voorstelling statisch geheugen**

Dynamisch geheugen bestaat in essentie eigenlijk gewoon uit een condensator. Als je over een condensator een gelijkspanning aanbrengt en die vervolgens wegneemt, kan je achteraf nog meten welk spanning erop stond. Dit is een vorm van geheugen. Deze geheugens hebben twee belangrijke nadelen. Ten eerste zal een leesoperatie de condensator ontladen.



**Figure 4.5. Voorstelling leescyclus dynamisch geheugen**

Een leesoperatie is met andere woorden destructief. Ten tweede bestaan er geen perfecte condensatoren en vertoont dit soort geheugen dus ook een lek. Dit betekent dat mettertijd de inhoud van het geheugen verloren gaat, tenzij die ververst wordt. Bij dit soort van geheugen is dan ook een regelmatige refresh noodzakelijk. Vergelijken met statisch geheugen is dynamisch geheugen trager. Het statisch geheugen is een actieve schakeling en kan dus stroom sturen of opnemen als het gelezen wordt. Dynamisch geheugen kan niet echt stroom sturen, het is de condensator die ontladen of opgeladen wordt. Anderzijds is dynamisch geheugen dan weer goedkoper, waardoor het toegepast wordt op een lager niveau in de piramide. Meer bepaald is dit de technologie die in geheugenmodules wordt gebruikt. Deze zijn ook gangbaar bekend onder de term RAM of DRAM. === DRAM technologie

#### 4.4.1. Gemultiplexte adresklemmen

Dynamische RAMs hebben vanwege de grote densiteit meestal ook een grote capaciteit op de chip (tegenwoordig tot 16 Gbit per chip). Een dergelijke capaciteit betekent ook dat er heel wat adressignalen noodzakelijk zijn om een welbepaalde geheugencel te selecteren. Wanneer elk signaal op een aparte pin zou aangesloten worden, zou het noodzakelijk zijn om zeer grote behuizingen te gebruiken. Om dit probleem te omzeilen, wordt gebruik gemaakt van gemultiplexte adreslijnen: het volledige adres wordt opgesplitst in een Column Address en een Row Address.

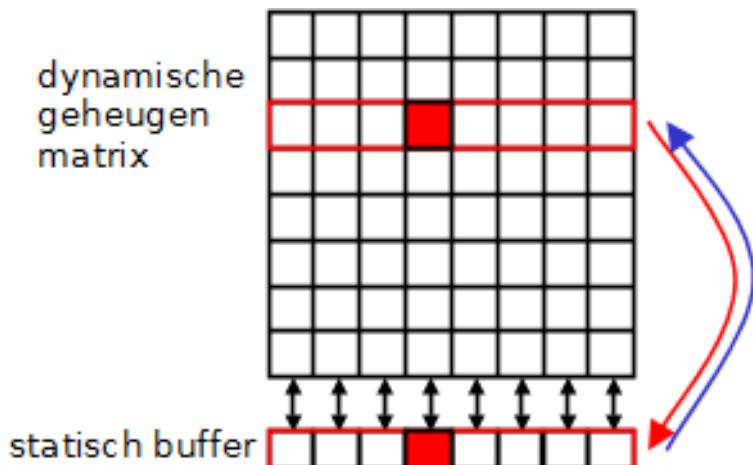
Deze twee adresgedeeltes worden de een na de ander aangeboden aan de adresklemmen van het IC. Hierbij wordt gebruik gemaakt van de RAS- en CAS-klem om de twee adresgedeelten te latchen. De snelheid van het geheugen wordt in grote mate bepaald door de toegangstijd tacc.

#### 4.4.2. Destructieve leescyclus

Eerder werd al aangegeven hoe een leescyclus de data op de condensatoren zal vernietigen. Dit is uiteraard een onaanvaardbare situatie. De oplossing ligt voor de hand. Als data gelezen is, wordt dezelfde data nadien terug weggeschreven, zodat de originele toestand hersteld wordt. Uiteraard is het niet verstandig deze taak aan de processor toe te wijzen, het is iets wat in de geheugenchips zelf geregeld moet worden. Het geheugen is, zoals in vorige paragraaf werd aangegeven, opgebouwd als een matrix van rijen met een welbepaald aantal kolommen. Naast deze matrix van dynamische geheugencellen is er ook een rij van statische geheugencellen. Op het ogenblik dat een rij-adres aangelegd wordt, zal de dynamische rij gekopieerd worden naar de statische rij. Hierbij verliest de dynamische rij dus haar inhoud. Vervolgens kan de gewenste cel gelezen worden en daarna wordt de inhoud van de statische rij weer naar de matrix gekopieerd. Hierdoor wordt de inhoud van het geheugen hersteld.

#### 4.4.3. Refresh

Met deze kennis wordt ook duidelijk hoe een refresh georganiseerd kan worden. Op regelmatige tijdstippen zal een zogenaamde RAS-only cyclus uitgevoerd worden. Hierbij wordt eigenlijk elke rij geselecteerd, gekopieerd naar de statische rij en weer weggeschreven. Hierdoor is de originele inhoud weer op peil gebracht, op voorwaarde dat deze cyclus voldoende regelmatig herhaald wordt. Met deze manier moet niet elke cel afzonderlijk gerefreshed worden, maar wordt een volledige rij ineens hersteld.



**Figure 4.6. Dynamisch geheugen met statische buffer**

#### 4.4.4. Bandbreedte bij DRAM

DRAM heeft, zoals je verder zal lezen, de eigenschap te werken met cyclussen. Om te berekenen wat de effectieve bandbreedte is (=geheugendebiet) hoor je steeds dezelfde benadering te maken:

"Aantal bytes die getransfereerd worden bij een cyclus"/"tijdsduur van een cyclus" = "bandbreedte"

Deze erg eenvoudige benadering wordt bij de verschillende types geheugen die volgen telkens toegepast.

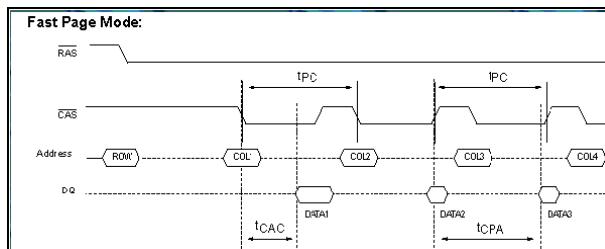
#### 4.5. Fast Page DRAM (FP-DRAM)

Hierboven werd reeds beschreven hoe met gemultiplexte adresklemmen eerst een rij-adres en vervolgens een kolomadres worden doorgegeven (langs dezelfde aansluitpinnen). Het voordeel hiervan is duidelijk: minder adresklemmen.

Het nadeel is dat een lees- of schrijfcyclus langer wordt. Het kost immers extra tijd om de adressen na elkaar door te geven. FP-DRAM verbetert de snelheid door cycli te combineren.

Zoals aangehaald bij het lokaliteitsprincipe gaan opeenvolgende cycli meestal door op naburige cyclusadressen. De kans dat meer dan een byte gelezen wordt in dezelfde rij, is dus vrij groot. FP-DRAM maakt hiervan gebruik door eenmaal een rij-adres op te geven en vervolgens een kolom te selecteren en deze te lezen of te schrijven. Onmiddellijk hierna wordt een tweede kolom geselecteerd en wordt deze gelezen of beschreven, vervolgens kan een derde kolom geselecteerd worden...Op die manier

worden een aantal cycli vermeden. Het zal relatief lang duren vooraleer het eerste geheugenwoord gelezen kan worden, terwijl de volgende minder tijd vragen.



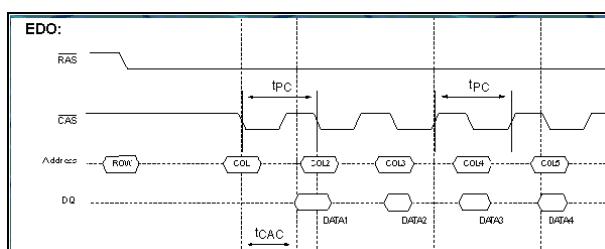
**Figure 4.7. Fast page DRAM**

In praktijk wordt er bijna steeds gewerkt met een burst van vier leescycli waarbij aangeduid wordt hoeveel klokcycli er nodig zijn per transfert, bijvoorbeeld 5-3-3-3. FP-DRAM werd gebruikt tot busfrequenties van 66 MHz.

Op een computersysteem met een 486 processor (32-bit databus) met 5-3-3-3 FP-DRAM geheugen aan 66Mhz betekent dit dat het maximale geheugendebiet gelijk is aan:

$$(4 \text{ "bytes"/"transfer"} \times 4 \text{ "transfers"/"burst"} \times 66 \text{ "Mcycli"/"sec"}) / (14 \text{ "cycli"/"burst"}) = 75 \text{ "MB"/"sec"}$$

## 4.6. EDO RAM

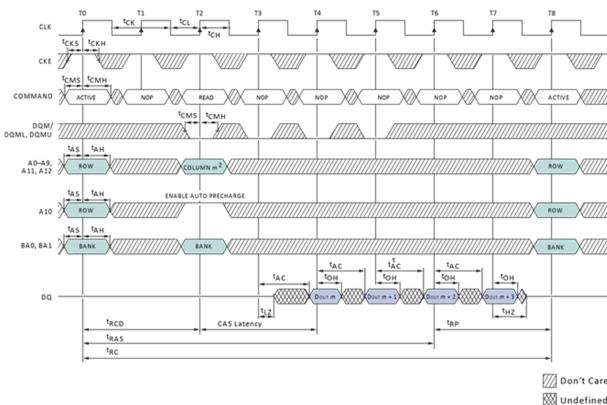


**Figure 4.8. EDO-RAM**

Extended Data Out-RAM is een aanpassing van het Fast Page-concept. Daarbij moet de memorycontroller wachten met het aanbieden van een nieuw kolomadres tot de vorige data gelezen waren. Bij EDO-RAM blijven de data op de uitgangen van het geheugen nog een tijd langer beschikbaar (zelfs tot na het aanbieden van het volgende kolomadres). Hierdoor wint men tijd: terwijl de data gelezen worden, kan men al het volgende kolomadres aanleggen. EDO-RAM kon gebruikt worden tot een busklok van 75 MHz met een timing van 5-2-2-2 klokcycli. Als we EDO-RAM dan nog combineren met een 64-bit bus (Pentium) geeft dit een maximaal debiet van 218 MB/s

(8 "bytes"/"transfer" xx 4 "transfers"/"burst" xx 75 "Mcycli"/"sec")//(11 "cycli"/"burst")=218 "MB"/"sec"

## 4.7. Synchronous DRAM



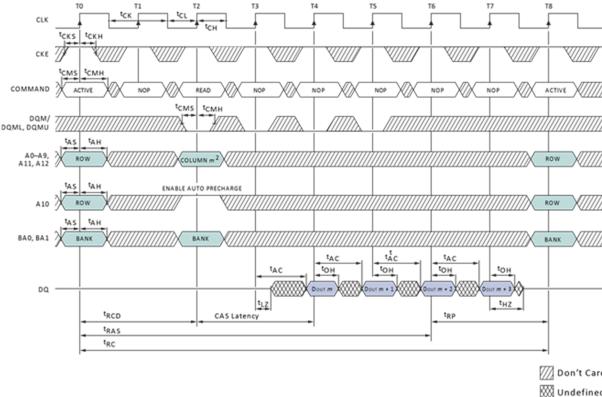
**Figure 4.9. afbeelding 20 Leesoperatie bij SD-RAM (bron: Micron)**

Bij SDRAM gaat men nog een stap verder met het lokaliteitsprincipe. In plaats van uit te gaan van het lezen van naburige kolommen, wordt nu vertrokken van het idee dat opeenvolgende kolommen uitgelezen zullen worden. In het deel over cache geheugens zal duidelijk worden dat het RAM geheugen effectief op deze manier wordt aangesproken. De cyclus kan nu aangepast worden tot het aanleggen van een rij-adres, het selecteren van een kolom en vervolgens het inlezen of naar buiten brengen van een aantal opeenvolgende kolommen. Die kunnen naar buiten gebracht worden op het tempo van de klok. Vandaar spreekt men over synchroon DRAM.+ In de afbeelding krijgen we een timing van 2-1-1-1. Daarnaast is SD-RAM geschikt voor busfrequenties tot 133 MHz (PC133), wat neerkomt op een maximaal debiet van 851 MBps.

(8 "bytes"/"transfer" xx 4 "transfers"/"burst" xx 133 "Mcycli"/"sec" )//(5 "cycli"/"burst") =  
851 "MB"/"sec"

#### 4.7.1. DDR SDRAM - DDR2 - DDR3

Principieel werkt DDR op dezelfde manier als SDRAM. Er wordt nog steeds een rijadres en een kolomadres aangelegd, waarna meerdere opeenvolgende cellen worden uitgelezen. Het verschil zit in het tempo waarop dit gebeurt. Bij Double Data Rate wordt data naar buiten gebracht op stijgende en dalende flank van de klok. Om dit te kunnen bereiken wordt gebruik gemaakt van een prefetch buffer. In elke cyclus worden nu 2 bits getransfereerd naar het prefetch buffer, dat de data dan aan een dubbele snelheid naar buiten kan brengen.



**Figure 4.10. Write cyclus bij DDR-RAM (bron: Micron)**

Het voordeel van deze werkwijze is een hogere maximale bandbreedte (datasnelheid bij het effectief overbrengen van data). Het nadeel zit in een hogere latentietijd. Tussen het aanleggen van de adressen en het naar buiten brengen van de data verloopt iets meer tijd. De snelheid van de modules wordt uitgedrukt op een aantal verschillende manieren. Een eerste manier is in de naam, waar twee verschillende mogelijkheden bestaan. DDR400 en PC3200 duiden op hetzelfde soort geheugenchips. De 400 duidt de kloksnelheid aan (2x200MHz), de 3200 duidt de maximale transfersnelheid aan.

Op een 64-bit databus is die:

$$8 \text{ "bytes"/"transfer"} \times (2 \times 200 \text{ "MHz"}) = 3200 \text{ "MB"/"sec"}$$

Er kan echter nog veel verschil zijn tussen twee PC3200 modules. De werkelijke snelheid hangt namelijk ook af van de totale latentietijd. Die kan op verschillende manieren worden aangegeven, maar een gangbare manier is het opgeven van vier getallen: TCL-Trcd-Trp-Tras.

- TCL = CAS Latency Time: tijd tussen CAS en beschikbaar worden van data
- T\_rcd = DRAM RAS to CAS Delay: tijd tussen RAS en CAS (ook tijd tussen active en read/write-commando )
- T\_rp = DRAM RAS Precharge: tijd tussen selecteren van twee rijen
- T\_ras = Precharge delay: minimale tijd tussen actief worden en precharge van volgende rij.

In elke leescyclus is zeker Trcd en TCL nodig. Indien bursts uit verschillende rijen nodig zijn, dan is ook Trp belangrijk.

**Figure 4.11. DDR-timing Tcl=2 (bron: Micron)**

### Voorbeeld

---

PC3200 geheugen met parameters 2-2-2-6 heeft voor een burst met vier transfers van 8 bytes  $2+2+4 \times 0.5=6$  klokcycli van 200 MHz nodig. Dit geeft een snelheid van 1067 MB/s. Voor twee dergelijke opeenvolgende transfers zijn

stem:  $[2(2+2+4 \times 0.5) + 2 = 14$  klokcycli] van 200 MHz nodig. Dit geeft 914 MBps.

---

Bij DDR kan ook het aantal cellen dat in een burst gelezen wordt variëren. Hetzelfde geheugen dat in een burst 8 transfers van 8 bytes uitvoert, haalt een snelheid van 1600MBps. Tras is in dit verhaal niet naar voor gekomen. Tras bepaalt de tijd waarin de volgende rij nog niet geladen mag worden. Deze moet groot genoeg zijn om de buffer niet te overschrijven voordat het volledig getransfereerd is over de databus. Deze parameter moet minimaal Trcd + TCL + 1 bedragen. Indien de parameter te klein is gaat uiteraard data verloren.

Opvolgers van DDR zijn DDR2 en DDR3. Behalve verbeteringen op het vlak van klokfrequenties en spanningen (DDR2 en DDR3 gebruiken telkens lagere spanningen) is het grootste verschil dat het prefetch buffer vergroot. Bij DDR2 worden vier bits gebufferd [3], bij DDR3 acht. DDR4, dat er binnenkort zit aan te komen, zal dit nogmaals verdubbelen.

Het gevolg is dat deze geheugens nog sneller data naar buiten kunnen brengen (hogere maximale transfer), maar dat dit weer een hogere latentietijd met zich meebrengt. Hou wel in gedachten dat slechts een klein stukje van het geheugen aan deze hoge snelheid werkt. Intern wordt nog steeds een relatief lage snelheid gebruikt om de cellen te beschrijven, maar door de buffers kan data toch aan een dubbele (DDR), viervoudige (DDR2) of achtvoudige (DDR3) snelheid naar buiten gebacht worden.

**Table 4.1. DDR3 snelheden (bron: wikipedia)**

Type geheugen	Alternatieve naam	Kloksnelheid	Theoretische bandbreedte
PC3-10600 DDR3 SDRAM	DDR3-1333	167 MHz	10.667 GB/s
PC3-11000 DDR3 SDRAM	DDR3-1375	172 MHz	11 GB/s

Type geheugen	Alternatieve naam	kloksnelheid	Theoretische bandbreedte
PC3-12800 DDR3 SDRAM	DDR3-1600	200 MHz	12.8 GB/s
PC3-13000 DDR3 SDRAM	DDR3-1625	203 MHz	13 GB/s
PC3-14400 DDR3 SDRAM	DDR3-1800	225 MHz	14.4 GB/s
PC3-14900 DDR3 SDRAM	DDR3-1866	233 MHz	14.933 GB/s
PC3-15000 DDR3 SDRAM	DDR3-1866	233 MHz	14.933 GB/s
PC3-16000 DDR3 SDRAM	DDR3-2000	250 MHz	16 GB/s
PC3-17000 DDR3 SDRAM	DDR3-2133	266 MHz	17.066 GB/s
PC3-17600 DDR3 SDRAM	DDR3-2200	275 MHz	17.6 GB/s
PC3-19200 DDR3 SDRAM	DDR3-2400	300 MHz	19.2 GB/s
PC3-21300 DDR3 SDRAM	DDR3-2666	333 MHz	21.3 GB/s
PC3-24000 DDR3 SDRAM	DDR3-3000	375 MHz	24 GB/s

Een belangrijke opmerking, die reeds gedeeltelijk aangehaald werd, is dat DDR, DDR2 en DDR3 gebruik maken van een andere werkspanning. Deze wordt alsmaar lager om het gedissipeerd vermogen en de bijhorende warmteontwikkeling te verkleinen, wat nodig is om hogere kloksnelheden toe te laten. Bovendien hebben de modules ook een verschillend aantal aansluitpinnen, waardoor het duidelijk zal zijn dat ze niet compatibel zijn.

Om ongelukken te vermijden wordt daarom een andere behuizing gebruikt (inkeeping in de module zit op een andere plaats).

## 4.8. Optimalisatietechnieken

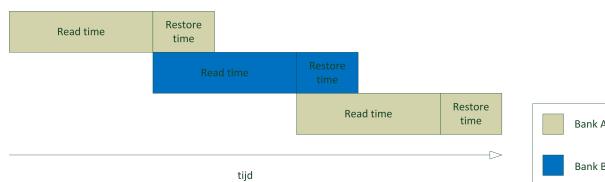
De evolutie in DRAM technologie is er steeds op gericht om de maximale bandbreedte te verbeteren, terwijl bijzonder weinig aan de latentietijd werd gedaan. Deze verbeterde hooguit in absolute waarde, doordat de kloksnelheid verhoogde. Relatief gezien (dus uitgedrukt in klokcycli) is de latentietijd eerder gestegen. Uit het voorgaande zou al gebleken moeten zijn dat na elke rijtoegang een hersteltijd nodig is om de bufferij(en) vrij te maken.



**Figure 4.12. normale geheugentoegang**

### 4.8.1. Interleaving

Een techniek die gebruikt kan worden om een deel van de dode tijd te vermijden, is interleaving. Meer bepaald gaat het dan om het vermijden van de tijd tussen twee rijen. Deze hersteltijd kan vermeden worden indien de volgende operatie doorgaat op een andere geheugenmodule. Om dit te bereiken worden bij interleaving naburige geheugenblokken verdeeld over verschillende geheugenbanken, die onafhankelijk van elkaar (en dus zonder hersteltijd) aangesproken kunnen worden. Belangrijke opmerking hierbij is dat er enkel snelheidswinst kan zijn als er gewisseld kan worden tussen banken. Het is dan ook belangrijk dat de memory controller weet of er al dan niet gewisseld kan worden tussen banken, zodat hij al dan niet rekening kan houden met de hersteltijd.



**Figure 4.13. Memory interleaving**

### 4.8.2. Dual channel

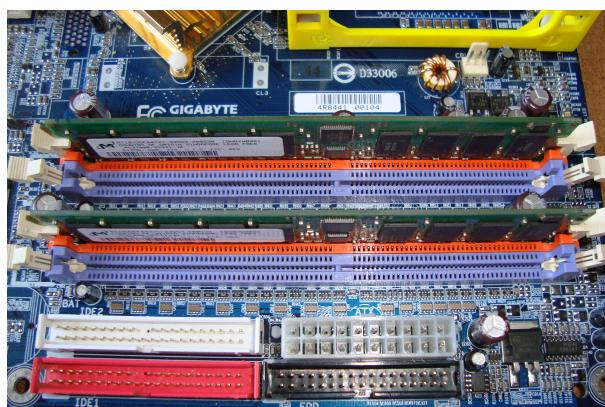
Een andere heel eenvoudige techniek om de snelheid te verhogen is het vergroten van de databus. Dit brengt wel een paar problemen met zich mee. Eerst en vooral moeten er extra aansluitingen voorzien worden op zowel de geheugenmodule als het moederbord en bovendien moeten de signaallijnen ook voorzien worden op het moederbord. Daarnaast zal er, zeker bij steeds toenemende kloksnelheden, een probleem ontstaan van tijdverschillen tussen de verschillende datalijnen.

Dual channel is een techniek die probeert de datasnelheid te verhogen door de databus naar de memory controller te verdubbelen, zonder de databus van de geheugenmodules te vergroten. Dit gebeurt weer door gebruik te maken van verschillende geheugenbanken. Elke geheugenbank heeft een databus van 64 bit, maar aangezien deze niet samenvallen is er een 128 bit databus naar de memory controller. De DIMM sockets op een moederbord zijn dus fysiek verbonden met één van de twee 64 bit kanalen. Uiteraard kan je enkel voordeel halen als geheugenmodules aangesloten zijn op de twee kanalen.



**Figure 4.14. geheugentoegang met Dual Channel**

Behalve het aanschakelen van meerdere modules is het dan ook belangrijk om ze in de juiste sockets te steken, zodat je beide kanalen gebruikt (en niet twee modules op hetzelfde kanaal). Moederborden met meerdere sockets op de kanalen, hebben overeenkomstige plaatsen op die kanalen die eenzelfde kleur hebben. Op deze manier zijn er dus matched sockets. Op deze matched sockets moet dus geheugen geïnstalleerd worden.



**Figure 4.15. dual channel sockets**

Dit geheugen moet identiek zijn in capaciteit, anders zou een deel van het geheugen niet bereikbaar zijn in dual channel mode. Verder moeten de modules in principe niet gelijk zijn. Zelfs modules met verschillende snelheden zijn mogelijk, al zal dan tegen de traagste snelheid gewerkt worden. In principe betekent echter dat er in de praktijk wel problemen kunnen ontstaan, zodat moederbord fabrikanten het gebruik van identieke modules aanraden. Deze worden door verschillende geheugenfabrikanten ook aangeboden.

Hoewel dual channel een veelbelovende techniek is, blijkt uit benchmarks dat de winst (ondertussen) toch marginaal is. Een deel van de verklaring hiervoor zou liggen in

het cache geheugen, dat steeds groter en efficiënter wordt. Later zullen we zien dat de cache geheugens het snelheidsverschil tussen geheugen en processor moeten opvangen. Je kan trouwens zelf aan de berekeningen bij DDR al zien dat dual channel weinig invloed heeft, tenzij echt grote opeenvolgende stukken gelezen worden. Vergelijk de tijd die nodig is voor een burst van 4 transfers met die voor een burst van 8 transfers. Op een dual channel systeem zullen de bursts immers maar half zo groot zijn als bij single channel.

#### 4.8.3. Buffered/registered RAM en foutdetectie

Een term die regelmatig terug te vinden is bij RAM geheugen is (un)buffered of registered. De termen buffered en registered hebben dezelfde betekenis. Op basis van de eerder uitleg zou je kunnen vermoeden dat elk DRAM geheugen gebufferd is met de statische bufferrij. De term buffered slaat in dit geval niet op die statische rij, maar wel op de aanwezigheid van een extra bufferchip. Deze bufferchip doet dienst als elektrisch buffer/versterker tussen de geheugenIC's en de rest van het systeem. Op die manier kan het bijvoorbeeld problemen met onvoldoende stroom helpen oplossen. De bufferchip is dikwijls te herkennen aan zijn dwarse plaatsing op de module.

Twee andere termen die je kan terugvinden zijn (non)ECC en parity. Beiden houden verband met het vermogen van het geheugen om fouten te laten detecteren. In het geval van pariteit wordt per byte een pariteitsbit berekend. Dit bit wordt mee verzonden. De ontvanger herberekent de pariteitsbit en vergelijkt het resultaat van zijn berekening met het ontvangen pariteitsbit. Indien ze verschillen is er een fout geweest en moet de transfer opnieuw gebeuren.

ECC werkt op gelijkaardige manier, maar maakt gebruik van een hash functie. Voordeel van deze manier van werken is dat meervoudige bitfouten gedetecteerd kunnen worden of dat enkelvoudige bitfouten gecorrigeerd kunnen worden. Pariteit kan enkel enkelvoudige bitfouten detecteren. Uiteraard kost deze foutcontrole ook rekenkracht en dus tijd. Door de betrouwbaarheid van moderne geheugens hebben deze technieken enkel zin in kritische toepassingen (bijvoorbeeld servers, procescontrole, ...)

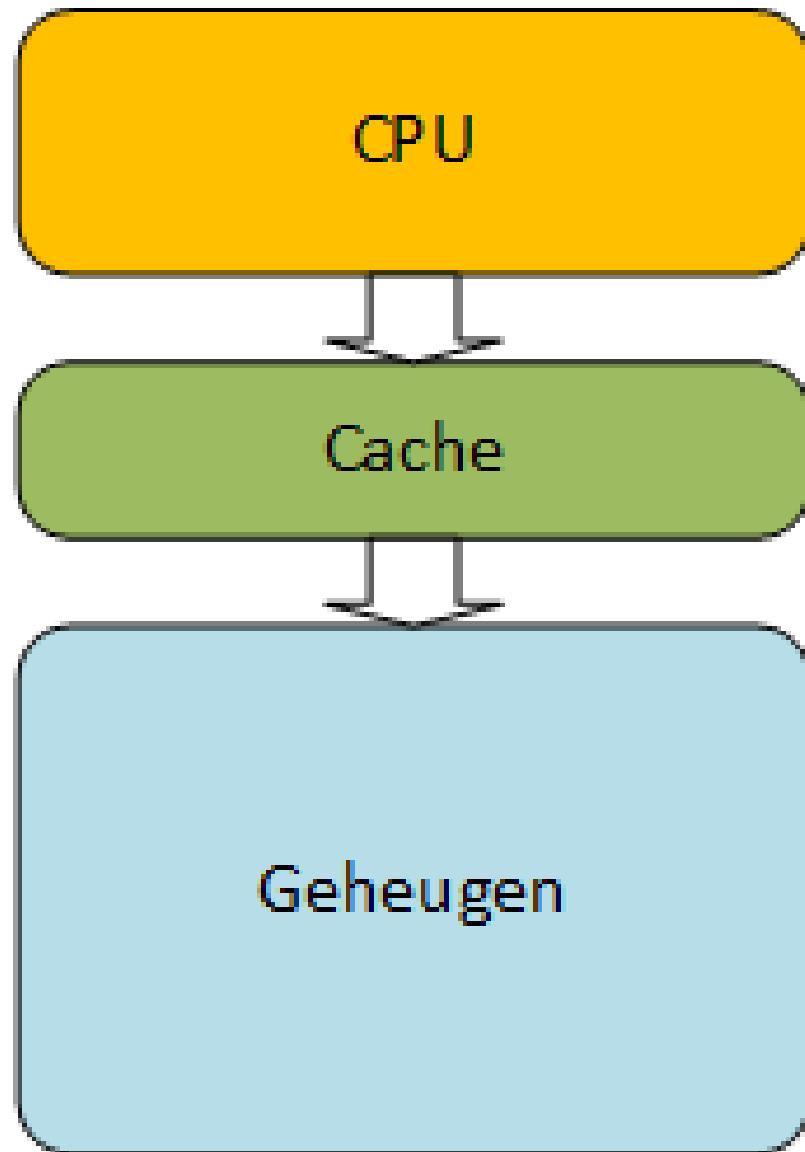
### 4.9. Cache geheugen

Zelfs met alle optimalisatietechnieken blijft er een snelheidsprobleem. Instructies die de processor moet uitvoeren, zitten in het geheugen en als de snelheid waarmee het geheugen instructie kan leveren, vergeleken wordt met de snelheid waarmee de processor ze kan uitvoeren, blijkt die laatste een stuk sneller. Een belangrijk verschil, want een processor kan nu eenmaal niet sneller instructies afwerken dan dat ze

door het geheugen aangeboden kunnen worden. Sneller geheugen maken, ligt voor de hand. Zoals eerder al aangehaald is statisch geheugen sneller dan dynamisch geheugen.

Belangrijk nadeel van statisch geheugen is dat het per byte veel duurder is dan dynamisch geheugen. De hoeveelheid statisch geheugen in een computersysteem zal dus eerder klein zijn en het komt erop aan deze kleine hoeveelheid zo efficiënt mogelijk te gebruiken. Een tweede probleem is dat niet enkel de snelheid van het geheugen problemen geeft, maar ook de snelheid van de interface naar het geheugen. Om dit probleem aan te pakken was er ooit een snelle back-side bus die de verbinding met het cache geheugen verzorgde. Ondertussen zijn er al verschillende niveaus van cachegeheugens in de processor geïntegreerd, zodat de verbinding met dit geheugen ook in de processor zelf zit en daardoor veel sneller kan zijn.

#### 4.9.1. Werking



**Figure 4.16. Cache als buffer tussen CPU en geheugen**

Het cache geheugen vormt een buffer dat het snelheidverschil tussen processor en CPU moet opvangen. Aangezien het cache geheugen een stuk kleiner zal zijn dan het dynamisch geheugen, komt het erop aan om de nodige gegevens klaar te hebben zitten in het cache geheugen.

Om dit te realiseren wordt weer uitgegaan van het lokaliteitsprincipe. Zowel het cache geheugen als het hoofdgeheugen worden onderverdeeld in blokken van dezelfde grootte. De blokken in het cache geheugen (cache lines) kunnen een kopie bevatten van een blok uit het hoofdgeheugen.

Een leescyclus verloopt als volgt:

#de processor vraagt een adres #Indien dat adres in een blok ligt dat in de cache te vinden is, wordt er gesproken van een cache-hit en heeft er een snelle leescyclus plaats vanuit de cache. #In het andere geval (een cache-miss) wordt via een trage leescyclus het gevraagde woord opgehaald uit het centrale geheugen terwijl ook onmiddellijk het blok waarin dit woord zich bevindt naar de cache gekopieerd wordt.

Aangezien opeenvolgende geheugentoegangen meestal op naburige adressen doorgaan, is de kans groot dat een volgende toegang een cache-hit geeft en dus snel verwerkt kan worden.

Een belangrijke parameter voor de snelheid van het cache geheugen is dus de hitrate. Dit is het percentage geheugentoegangen dat rechtstreeks via het snelle cache geheugen kan verlopen. De hitrate ligt typisch tussen 80 en 99%.

Een schrijfcyclus kan gelijkaardig verlopen, maar er stelt zich wel een nieuw probleem.+ In het geval van een schrijfcyclus worden er immers gegevens aangepast. Indien er een cache-hit is, lijkt het logisch om de inhoud van het cache geheugen aan te passen en pas later (bij het verwijderen van de cacheline) wordt de inhoud van het hoofdgeheugen aangepast. Deze manier van werken heet write-back cache. Belangrijk nadeel hiervan is dat op een bepaald ogenblik de inhoud van het hoofdgeheugen niet consistent is met het cache geheugen. Dit kan bijvoorbeeld bij DMA-toegangen problemen opleveren. Tweede nadeel is dat het wegschrijven naar het geheugen gebeurt op het ogenblik dat de pagina uit het cache geheugen verwijderd wordt. Dit is het ogenblik waarop in het cache geheugen plaats gemaakt wordt voor een nieuwe pagina. Dit vertraagt dus het inladen van de nieuwe pagina.

Een alternatief is gebruik maken van write through cache. In dit geval worden steeds zowel het hoofdgeheugen als het cache geheugen aangepast. Nadeel is duidelijk dat steeds gebruik gemaakt wordt van het tragere hoofdgeheugen.

Dit kan gedeeltelijk opgevangen worden doordat de processor niet moet wachten tot de volledige cyclus is afgewerkt, maar bij opeenvolgende schrijfopdrachten zal het toch leiden tot vertragingen. Bij schrijfcycli zijn er nog verschillen mogelijk: write allocate en write no-allocate. Bij write allocate zal bij een cache miss het geheugenblok in het cache geheugen geladen worden, bij write no-allocate niet.

Het voordeel is dat een schrijfoperatie naar het geheugen typisch sneller is dan het ophalen van een volledig blok. Bijvoorbeeld bij write-through geheugen is makkelijk in te zien dat het interessant kan zijn om het blok niet volledig in te laden.

### 4.9.2. Soorten caches

Er zijn een aantal onderscheiden te maken tussen cache geheugens. Een eerste belangrijk verschil is dat tussen level 1 en level 2 caches. In principe zijn zelfs nog meer niveau's van cache geheugens mogelijk. L1 cache is het cache geheugen dat het dichtst bij de processor staat. Het moet dan ook het snelste geheugen zijn, zodat het de processor kan volgen. Naarmate de snelheid van de processor toeneemt, werd het verschil in snelheid zo groot dat het interessant werd om een tweede niveau buffer in te zetten. Dit geheugen is minder kritisch op het vlak van snelheid (het moet de processor niet rechtstreeks voorzien van gegevens) en kan dus op andere vlakken geoptimaliseerd worden. Zo zal L2 cache typisch minder snel, maar wel een stuk groter zijn.

Een ander verschil tussen cache geheugens is dat L1 cache dikwijls opgedeeld worden in data cache en instructie cache. Hiervoor zijn verschillende redenen te bedenken. Een belangrijke reden is dat met pipelining, een deel van de processor (instruction fetch unit) instructies ophaalt en tegelijkertijd een ander deel (operand fetch) gegevens ophaalt om de bewerkingen uit te voeren.

Als beide geheugens gescheiden zijn, kunnen de twee units onafhankelijk van elkaar hun operaties uitvoeren. Anderzijds kunnen de cache geheugens ook geoptimaliseerd worden. Zo zal een processor nooit wijzigingen aanbrengen in de instructies die hij uitvoert. Voor de instructiecache moeten geen voorzieningen getroffen worden voor schrijfoperaties.

### 4.9.3. Overschrijfstrategieën

Bij een cache miss zal in de meeste gevallen een volledig geheugenblok ingeladen worden. Dit betekent dat in het cache geheugen plaats zal moeten gemaakt worden. Er zal met andere woorden een lijn geselecteerd moet worden, die uit het cache geheugen verwijderd zal worden. Het selecteren van die lijn moet uiteraard doordacht gebeuren om de hit rate zo hoog mogelijk te houden. In principe is het best om de cacheline te verwijderen die het langst niet zal gebruikt worden. Het probleem met deze keuze is dat er kennis over de toekomst voor nodig is. In plaats daarvan zijn er een aantal strategieën die op een andere manier proberen de meest geschikte lijn te selecteren:

#### **First In First Out (FIFO)**

selecteert de lijn die al het langst in het cachegeheugen zit. Het is een erg eenvoudig te implementeren techniek, aangezien de lijnen eigenlijk gewoon cyclisch overschreven worden. Deze techniek is daarentegen weinig efficiënt op het

vlak van het optimaliseren van de hitrate. Een lang geleden, maar veel gebruikte lijn zal bijvoorbeeld eerder verwijderd worden dan een lijn, waar maar een keer data uit gelezen wordt, maar die wel later is ingeladen.

#### **Least Recently Used (LRU)**

selecteert de lijn die al het langst niet meer gebruikt wordt. De kans dat deze lijn dan plots weer gebruikt zal worden, is een stuk kleiner dan bij FIFO. Hierdoor zal deze techniek leiden tot een hogere hitrate. Nadeel is dan weer dat er een pak meer bij komt kijken om bij te houden welke lijn geselecteerd wordt. Deze berekening kost zowel geld (om te implementeren) als tijd (om de lijn te selecteren). In het bijzonder als uit een groot aantal lijnen gekozen moet worden, is het gebruik van deze techniek niet aangewezen.

#### **Least Frequently Used (LFU)**

deze techniek zal de lijn selecteren die het minst frequent gebruikt wordt. De techniek haalt gelijkaardige resultaten als LRU, maar heeft ook dezelfde nadelen.

#### **Adaptive Replacement Cache (ARC)**

combineert zowel LRU als LFU. Hierdoor worden nog betere resultaten gehaald op het vlak van hitrate, maar de bewerkingen en de implementatie ervan worden anderzijds ook complexer.

#### **Random**

selecteert willekeurig een lijn. Dit is een eenvoudige te implementeren techniek, die toch aanvaarbare resultaten haalt, in het bijzonder als er een groot aantal lijnen zijn waaruit geselecteerd moet worden. Deze techniek wordt soms gecombineerd met LRU, door een bit te koppelen aan een lijn. Dit bit geeft aan of de lijn al gebruikt is. Als alle lijnen gebruikt zijn, worden alle bits weer gereset. Als dan een lijn gekozen moet worden, zal random geselecteerd worden uit alle lijnen die gemarkeerd zijn als "niet gebruikt".

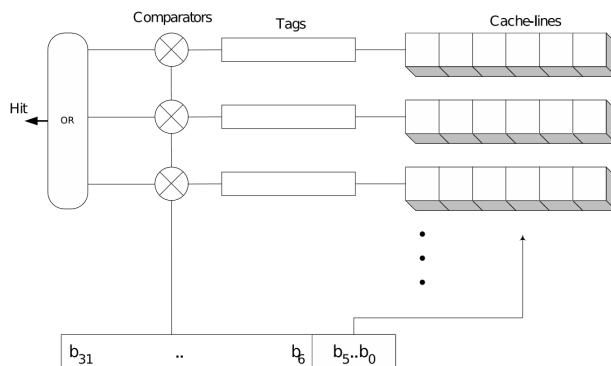
## **4.10. Associativiteit**

De associativiteit van het cache geheugen bepaalt op welke cachelines een welbepaald geheugenblok terecht kan komen. De associativiteit bepaalt dus ook het aantal lijnen waaruit geselecteerd kan worden en bepaald dus zowel rechtstreeks (beperking van lijnen) als onrechtstreeks (welke overschrijfstrategie) mee de hitrate.

### **4.10.1. Fully Associative cache**

#### **Figure 4.17. principe fully associative cache**

Bij fully associative cache kan een blok uit het geheugen terecht komen in gelijk welke cacheline. Om te kunnen bepalen welk geheugenblok in een bepaalde cacheline zit, wordt een elke cacheline een tag gekoppeld. Deze is nodig om te kunnen bepalen of het blok aanwezig is in het cache en eventueel de juiste cacheline te kunnen selecteren.



**Figure 4.18. voorbeeld fully associative geheugen**

Een voorbeeld zie je in bovenstaande afbeelding.

De processor beschikt over een 32-bit adresbus en een cache geheugen van 16kB met cachelines die 64-byte breed zijn. Om binnen elke cacheline het juiste byte te selecteren zijn er 6 bits nodig.

Hiervoor worden uiteraard de minst significante gebruikt. De overige 26 bits worden gebruikt om de inhoud van de cacheline te identificeren. Je zou dit kunnen zien als het nummer van het geheugenblok. Merk immers op dat voor elk byte van een geheugenblok, deze 26 bits van het adres steeds overeenkomen. Het zijn dan ook deze bits die opgeslagen worden in de tag. Naast de cachelines en de tags die eraan gekoppeld zijn, zijn er een aantal comperatoren voorzien. Bij het begin van een cyclus worden deze comperatoren gebruikt om de bovenste 26 bits van het adres te vergelijken met de inhoud van de tags. Indien de uitkomst van een van de comperatoren een gelijkheid aangeeft, is er een cache-hit en is meteen ook de juiste cacheline geselecteerd.

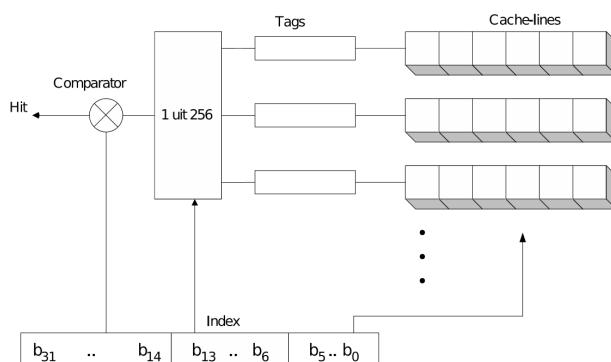
In het voorbeeld zijn er 256 tags van 26 bits elk (6656 bits) die vereist zijn naast de data-cache. Dit komt neer op bijna 5% van de cache-capaciteit. Vermits elke pagina uit het geheugen in om het even welke cache-line geplaatst kan worden, kan de cache optimaal benut worden. Bovendien kan gebruikt gemaakt worden van de optimale overschrijfstrategie. Het nadeel van associative cache is dat het een vrij dure implementatie is. Er is immers behoefte aan veel supersnel geheugen om de tags te implementeren. Daarnaast moeten ook de snelle comparatoren gerealiseerd worden. Binnen de tijd van een cyclus moet immers geweten zijn of er een cache hit is.

Fully associative cache geeft goede resultaten. Indien deze techniek dan ook nog gecombineerd wordt met de meest complexe overschrijfstrategieën, wordt het geheel extreem complex en duur. Zoals al aangehaald bij de overschrijfstrategieën is hier het aantal lijnen meestal te groot om gebruik te maken van LRU, LFU of ARC, zonder daarvoor een andere prijs te betalen.

#### 4.10.2. Direct mapped cache

**Figure 4.19. principe direct mapped cache**

Bij direct mapped cache kan elk geheugenblok slechts in één cacheline terecht. Dit maakt dat overschrijfstrategieën overbodig zijn, er is namelijk maar een plaats waar het geheugenblok terecht kan en de inhoud van die cacheline zal verwijderd worden. Evident nadeel is dat als er geen keuze mogelijk is, de optimale keuze niet gemaakt kan worden. Deze manier van werken heeft dus een negatief effect op de hitrate. Om dit te staven met een extreem voorbeeld: bij direct mapped cache is het mogelijk dat een cacheline plaats moet maken voor een andere, terwijl een deel van het cache nog niet in gebruik is. Doordat een geheugenblok maar op een lijn terecht kan, worden de tags kleiner en is er ook slechts een comparator nodig.



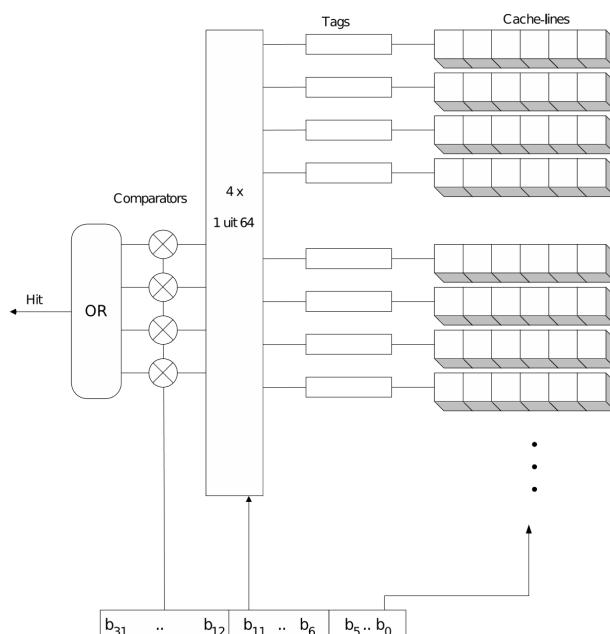
**Figure 4.20. voorbeeld direct mapped**

Een voorbeeld zal dit verduidelijken... De processor beschikt overeen 32-bit adres bus en een cache geheugen van 16kB met cachelines die 64-byte breed zijn. Naast de zes bits die nodig zijn om een byte binnen de cacheline te selecteren, zijn er nu ook acht bits nodig om een van de 256 (=16kB/64B) lijnen te selecteren. Op die manier blijven er maar 18 bits over voor de tag. Dit soort cache-organisatie is dus veel goedkoper en eenvoudiger dan de vorige: er is slechts 1 comparator (van 18 bits) nodig, en het taggedeelte is ook kleiner: 256 x 18 bits (4608 bits, 3,4%). Dit soort cache wordt toegepast op plaatsen waar de snelheid minder kritisch is, met andere woorden in de caches die verder van de processor staan.

### 4.10.3. Set-associative cache

**Figure 4.21. principe set-associative cache**

Set-associative cache is de tussenvorm. Hierbij worden de cachelines gegroepeerd in sets. Elk geheugenblok kan terecht in slechts een set, maar binnen die set kan het in elke cacheline terecht.



**Figure 4.22. voorbeeld set associative**

### Example 4.3. voorbeeld

Een voorbeeld: een processor met een 32-bit brede adresbus en 16kB 4-way set associative cache geheugen met cachelines van 64 byte. 4-way betekent dat een set bestaat uit vier cachelines. De 256 lijnen worden nu verdeeld in  $256/4=64$  sets.

Zes bits zijn nog steeds nodig om een byte in de cache line te selecteren. Daarnaast zijn er nu een aantal bits nodig om een set te kiezen. Aangezien er 64 sets zijn, zijn er hiervoor zes bits nodig. De resterende bits (20) worden gebruikt voor de tag. Als nu een adres aangeboden wordt, wordt een set van vier lijnen geselecteerd en worden de vier tags vergeleken met de meest signicante bits van het adres. Als één van de tags gelijk is aan deze bits is er een cache hit.

## Example 4.4. voorbeeld

Rekenvoorbeeld (meer voorbeelden vind je in [4] )

**Opgave:** Een 4-way set associative cache heeft een grootte van 64KB. De CPU werkt met 32-bit addressering, elk geheugenwoord bevat 4 bytes. Elke cacheline bevat 16 bytes. De cache gebruikt een write-through policy. Bereken de totale benodigde hoeveelheid geheugen (geheugen+tags) die nodig is om dit te implementeren...

### Antwoord:

het aantal bits is een adres is 32 (a)

Een cacheline omvat 16 bytes, wat betekent dat de vier (b) LSB's niet moeten geadresseerd worden.

Het aantal sets is dan  $64\text{KB}/(4*16) = 1024$  (c)

Het aantal bits dat nodig is om deze te kunnen adresseren:  $\log_2 1024 = 10$  bits (d)

Het aantal tag bits wordt dan:  $32 - 4$  (b) –  $10$  (d) = 18 bits

Om de totale grootte dan te berekenen kan je redeneren:

\*Elke cachelinje heeft 16 bits data, en een tag van 18 bits. Dit komt op een totaal van 146 bits. \*Dit moet je met vier vermenigvuldigen ('4'-way) en met het aantal sets (d), wat resulteert in 598016 bits.

Het percentage overhead is dus (totaal benodigde bits)/('nuttige<sup>A</sup> bits) of 14%



bereken aan de hand van vorig voorbeeld hoe de overhead zich gedraagt bij 8-way en 16-way cache geheugen.

Zoals al gezegd is set associative de tussenvorm. Eigenlijk kan je zeggen dat fully associative cache ook set associative cache is, maar met slechts één set. Direct mapped cache is anderzijds set associative met slechts één lijn per set.

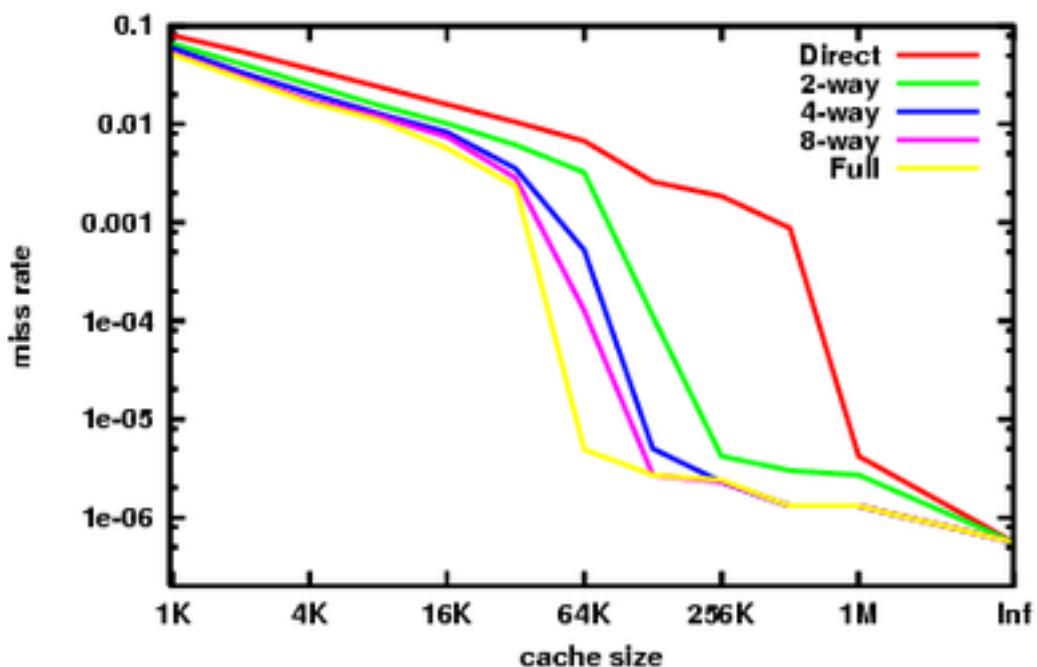
De eigenschappen van set associative liggen dan ook tussen de twee vormen in. Het laat minder mogelijkheden voor het kiezen waar een cacheline terecht kan dan fully associative, maar meer dan direct mapped. Anderzijds heeft het minder bits nodig voor de tags en ook minder comparatoren dan bij fully associative cache. vergeleken met direct mapped is het dan weer complexer. Meer algemeen geldt dat naarmate de associativiteit toeneemt er meer keuze is in de cache lijnen, maar dat de kost die hieraan verbonden is ook hoger wordt. Omwille van de beperking van het aantal

cachelines in een set is set-associative cache gemakkelijker te combineren met betere overschrijfstrategieën. Mede daardoor is fully associative cache een eerder zeldzame vorm van cache.

Opmerking: in literatuur worden de termen ‘cache line’, ‘cache block’, ‘cache entry’ en ‘cache set’ vaak door elkaar gebruikt. [4]

#### 4.10.4. Snelheid van de cache

In het voorgaande hebben we reeds een aantal eigenschappen aangehaald die mee de efficiëntie van het cache geheugen bepalen. Dit is uiteraard een heel belangrijke eigenschap, aangezien bij cache hits, de processor aan zijn volle snelheid kan werken.



**Figure 4.23. cache misses in functie van grootte en associativiteit  
(bron: wikipedia)**

In bovenstaande afbeelding toont een grafiek het aandeel missers in functie van de grootte van het cache geheugen en dit voor verschillende associativiteit. Een deel van de missers zijn onvermijdelijk. Het zijn de zogenaamde coldstart misses, die afkomstig zijn van de eerste keer dat toegang tot een blok gezocht wordt. Een tweede deel hangt samen met de capaciteit. Dit neemt duidelijk af naarmate het cache geheugen groter wordt. Een derde deel hangt samen met de associativiteit. In deze grafiek komt duidelijk naar voor dat met toenemende associativiteit het aantal missers afneemt. Naast de hitrate zijn er nog een aantal andere eigenschappen, die samen de snelheid van het volledige geheugen bepalen.

Deze eigenschappen zijn:  
Hit time:: de tijd nodig om bij een hit de gegevens op te halen.  
Miss time:: de tijd nodig om bij een miss het nieuwe geheugenblok te laden en de gevraagde gegevens beschikbaar te maken  
Miss penalty:: de extra tijd die nodig is bij een cache miss (eigenlijk misstime-hit time)  
Hit rate:: percentage toegangen die rechtstreeks langs de cache gaan

De formule voor de totale toegangstijd van de gecombineerde caches is dan:

$$t_{\text{totaal}} = HR_{\text{L1}} \times t_{\text{L1}} + (1 - HR_{\text{L1}}) \times HR_{\text{L2}} \times t_{\text{L2}} + (1 - HR_{\text{L1}}) \times (1 - HR_{\text{L2}}) \times t_{\text{RAM}}$$

Waarbij:

HR = hitrate

t = aantal cycli

### Example 4.5. Voorbeeld

Een computer heeft een L1-cache, toegangstijd 1 ns en een hitrate van 96%. De L2 cache heeft een hitrate van 88 % bij een toegangstijd van 2 ns. Het externe geheugen heeft een toegangstijd van 8 ns. De gemiddelde toegangstijd wordt dan:

$$0.96 \times 1 \text{ "ns"} + 0.04 \times 0.88 \times 2 \text{ "ns"} + 0.04 \times 0.12 \times 8 \text{ "ns"} = 1.0688 \text{ "ns"}$$

Probeer zelf eens uit wat de invloed is van de toegangstijd van het RAM-geheugen in dit systeem. Probeer eens wat er gebeurt zonder L2 cache. De verschillende factoren worden niet op dezelfde manier beïnvloed door de eigenschappen van het cache. Bijvoorbeeld de associativiteit van de cache biedt meer mogelijkheden op het vlak van cachelines selecteren, waardoor de hitrate hoger kan zijn. Daar staat tegenover dat een complexere berekening nodig is om een lijn te selecteren, waardoor bij een cache miss er extra tijd verloren gaat (hogere miss time).

Een ander voorbeeld zijn de grootte van de cachelines. Grottere cachelines geven in eerste instantie een hogere hit rate, omdat meer naburige gegevens gekopieerd worden. Anderzijds zijn grotere cachelines nefast voor de miss time. Bij heel grote cachelines zal bovendien zelfs de hitrate dalen, omdat lijnen te snel uit het cachegeheugen verdwijnen (omdat bij dezelfde grootte er nu eenmaal minder lijnen zijn).

Een groter cachegeheugen geeft dan weer meer cachelines. Mits een goede overschrijfstrategie toegepast wordt en de associativiteit hoog genoeg is, kan de hit rate

toenemen. Nadeel is dat de selectie van een lijn dan weer complexer wordt waardoor de miss time weer toeneemt.

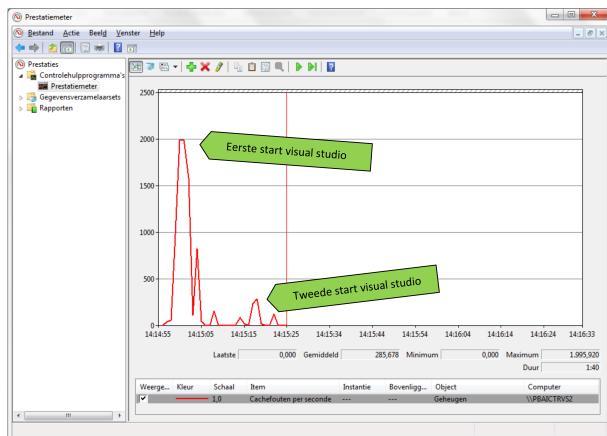
## 4.11. Virtueel geheugen

Virtueel geheugen is een oplossing om het tekort aan fysiek geheugen op te lossen. Het tekort aan geheugen komt van steeds groter wordende toepassingen en bestanden die bewerkt worden. Bovendien worden ook verschillende programma's naast elkaar uitgevoerd. Toch zijn er vandaag ook een aantal situaties waarin een computer ruim voldoende heeft met het fysieke geheugen. In dat geval biedt virtueel geheugen weinig meerwaarde.

### 4.11.1. Werking

Virtueel geheugen zal gebruik maken van de beschikbare ruimte op een of ander medium voor massaopslag om het geheugen groter te laten lijken.

Meestal zal het gebruikte medium een harde schijf zijn. We zullen hier in het volgende van uitgaan. Op de harde schijf zal ruimte voorzien worden die gebruikt wordt als swapruimte. Het kan een swap-file zijn of een swap-partitie. Indien de processor toegang wil tot een bepaald adres, zal aan de hand van het adres nagegaan worden of de gevraagde gegevens aanwezig zijn in het fysieke geheugen. Indien de gegevens in het fysieke geheugen zitten, worden ze opgehaald en wordt de instructie gewoon verder afgewerkt. Als de gegevens niet in het fysieke geheugen zitten, treedt er een page fault op. Dit is een speciale exceptie. De processor wordt met andere woorden onderbroken en zal de exceptieroutine uitvoeren. In dit geval gaat het om een roll-back exception. De processor zal eerst terugkeren naar de toestand vlak voor de uitvoering van de instructie, die het probleem veroorzaakte en vervolgens de exception handler uitvoeren. De exception handler zal indien nodig plaats maken en vervolgens de nodige gegevens verplaatsen naar het fysieke geheugen. Uiteraard zal het hier niet gaan over een byte, maar wel over een groter stuk geheugen. We zullen later terugkomen op de grootte van dit geheugenblok.



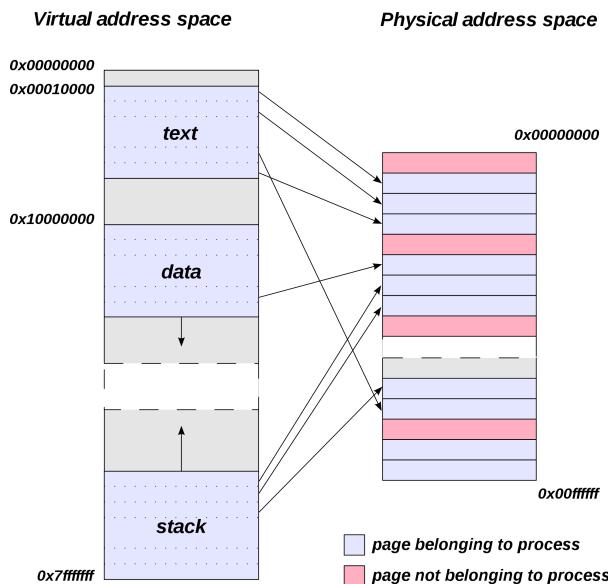
**Figure 4.24. Page-faults bij opstarten software**

Na de uitvoering van de handler keert de processor terug naar de toestand bij de oproep van de handler. Dit is uiteraard de instructie die in eerste instantie het probleem veroorzaakte. Deze keer zal bij de uitvoering van die instructie blijken dat de gegevens beschikbaar zijn in RAM.

Om virtueel geheugen te ondersteunen is, naast het opslagmedium, ook een processor nodig die de roll-back exception ondersteund. Daarnaast moet ook het besturingssysteem de nodige routines omvatten en tenslotte is er ook behoefte om bij te kunnen houden waar de gegevens zich bevinden (fysiek geheugen of swap). Belangrijk is ook dat bepaalde delen nooit mogen verdwijnen uit het fysieke geheugen. Een voorbeeld is de handler: als die op de swap staat, is er geen routine meer beschikbaar om hem naar het fysiek geheugen te verplaatsen.

#### 4.11.2. Paging - segmenting

De meest gebruikte manier om met virtueel geheugen te werken, is zowel de swap als het fysieke geheugen te verdelen in stukken van dezelfde grootte. Zo'n blok wordt dan een pagina genoemd. Voor elke pagina zou dan in een tabel opgeslagen kunnen worden waar de pagina zich bevindt. In deze tabel zou dan een bit aangeven of het fysiek dan wel virtueel aanwezig is en de andere bits zouden de locatie kunnen aanduiden. Uiteraard moet deze tabel ook ten allen tijde in het fysiek geheugen aanwezig blijven. Soms is het noodzakelijk om de grootte van deze tabel te beperken.



**Figure 4.25. Paging table (Wikimedia Commons, BSD)**

### Example 4.6. Voorbeeld

Op een 32-bit processor met pagina's van 4kB is een tabel nodig van 4MB (op voorwaarde dat de elementen van de tabel 32 bit groot zijn).

Om de tabellen klein te houden kan gewerkt worden met meerdere niveaus van tabellen. In het reeds aangehaalde voorbeeld zouden er 1024 tabellen van 4kB nodig zijn. Deze tabellen bevinden zich op het tweede niveau. Op het eerste niveau is er een tabel van 4kB, die toelaat een van de 1024 secundaire tabellen te selecteren.

Een groot voordeel is dat het enkel de eerste niveau tabel aanwezig moet zijn in het fysieke geheugen. De secundaire tabellen kunnen zich in de swapruimte bevinden.

Het alternatief voor paging is werken met segmenten. In grote lijnen is het verhaal hetzelfde, er is bijvoorbeeld ook een tabel die bijhoudt waar de segmenten zich bevinden. Het grote verschil is dat de gegevens nu niet opgedeeld worden in pagina's van gelijke grootte. Het grootste gevolg hiervan situeert zich op het vlak van geheugenfragmentatie. Bij het vervangen van geheugenpagina's of segmenten is er nu extra tijd (traagheid van harde schijf laat complexe berekeningen toe). Er kan dus een optimale keuze gemaakt worden. Bij segmentering kan het gebeuren dat een groot segment vervangen wordt door een veel kleiner segment, waardoor een stuk van het geheugen niet in gebruik zou zijn. Dit soort fragmentatie heet externe fragmentatie.

Bij paging gebeurt dit uiteraard niet. Daar worden immers pagina's van gelijke grootte vervangen. Bij paging kan wel interne fragmentatie optreden. In feite worden de gegevens van een segment verdeeld in pagina's van 4kB. Voor een bestand van 13kB geeft dit drie volledig gevulde pagina's en een pagina met 3kB ongebruikte ruimte.

#### 4.11.3. Snelheid en virtueel geheugen

Virtueel geheugen heeft de reputatie om de computer te vertragen. Hoewel dit strikt genomen wel waar kan zijn, klopt dit niet helemaal. De reputatie komt namelijk van het swappen van gegevens naar de harde schijf. Aangezien de harde schijf een stuk trager is dan het RAM, gaat dit uiteraard een stuk trager dan gewoon lezen van gegevens uit het RAM.

Dit is echter geen eerlijke vergelijking. Indien de swap ruimte niet gebruikt zou kunnen worden, zou er gewoon onvoldoende geheugen beschikbaar zijn om in deze situatie te komen. De gebruiker zou zelf bestanden of programma's moeten afsluiten om te kunnen doen wat het swappen veroorzaakte. Het gebruik van virtueel geheugen biedt de gebruiker dus in eerste instantie extra mogelijkheden. Als die extra mogelijkheden gebruikt worden, gaat dat inderdaad trager.

Los daarvan kan het gebruik van virtueel geheugen wel vertraging geven. Een eerste evidente reden is dat in plaats van een adres gewoon op te zoeken in het geheugen, nu het adres eerst opgezocht moet worden in twee tabellen, alvorens de eigenlijke operatie kan doorgaan. Aangezien de tabel in het geheugen zit, zijn er dus drie geheugentoegangen nodig om een gegeven te manipuleren. Dit probleem wordt grotendeels opgelost door een Translation Lookaside Buffer. Dit is een snel soort cachegeheugen in de processor waar de meest recent gebruikte fysieke adressen opgeslagen worden. De hitrate van dit buffer ligt weer bijzonder dicht bij 100%, zodat deze vertraging vrijwel geen probleem meer is.

Een tweede vertraging kan ontstaan als het besturingssysteem onnodig voorbereidingen treft om een toekomstige swap-operatie te versnellen. Zoals daarnet vermeld is swappen een trage operatie. Bovendien moet eerst een pagina geswapt worden naar de harde schijf, waarna de gevraagde pagina in het fysieke geheugen geladen kan worden. Dit betekent tweemaal 4kB verplaatsen naar en van de harde schijf. Om op het ogenblik dat er gegevens uit de swap-ruimte nodig zijn, de transactie te versnellen, kan het besturingssysteem proberen om op de achtergrondpagina's te zoeken die waarschijnlijk niet onmiddellijk gebruikt zullen worden en deze dan al naar de swap te verplaatsen.

Als dan gegevens uit de swap nodig zijn, kan dat met een 4kB verplaatsing van de harde schijf. Het kan natuurlijk gebeuren dat de verplaatste pagina's toch eerder nodig zijn dan dat er een swapping nodig is. In dat geval zullen de verplaatste pagina's, wanneer ze terug nodig zijn, uit de swap moeten gehaald worden. Hierdoor reageert de computer trager dan in het geval er geen virtueel geheugen zou zijn.

## 4.12. Bronvermelding bij dit hoofdstuk

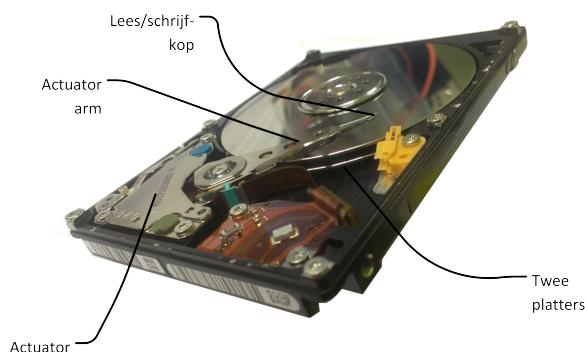
- [PATT1] 'Computer Organization And Design'. David A. Patterson, John L. Hennessey. fifth edition. p374

# Chapter 5. Opslagmedia

In dit hoofdstuk duiken we onder de motorkap van het secundair geheugen. Heel vaak wordt dit geïmplementeerd met HDD's of SSD's. Naast de fysieke eigenschappen bespreken we ook de logica die softwarematig een managementlaag (=bestandssysteem) voorziet voor gebruik door het besturingssysteem.

## 5.1. Harde schijf

### 5.1.1. Fysieke opbouw



**Figure 5.1. Fysieke opbouw harde schijf**

Bovenstaande afbeelding toont de fysieke opbouw van een harde schijf. Een harde schijf bestaat uit een aantal aluminium schijven, die men platters noemt. De keuze voor aluminium is ingegeven door de mechanische eigenschappen van dit materiaal. Aluminium is licht en toch voldoende stijf. Belangrijk nadeel van aluminium is dat het een niet magnetisch materiaal is, zodat het niet bruikbaar is voor magnetische opslag.

Dit probleem wordt verholpen door de platter te voorzien van een dun laagje magnetisch materiaal. Uiteraard kan een dergelijk laagje langs beide kanten van de platter voorzien worden.

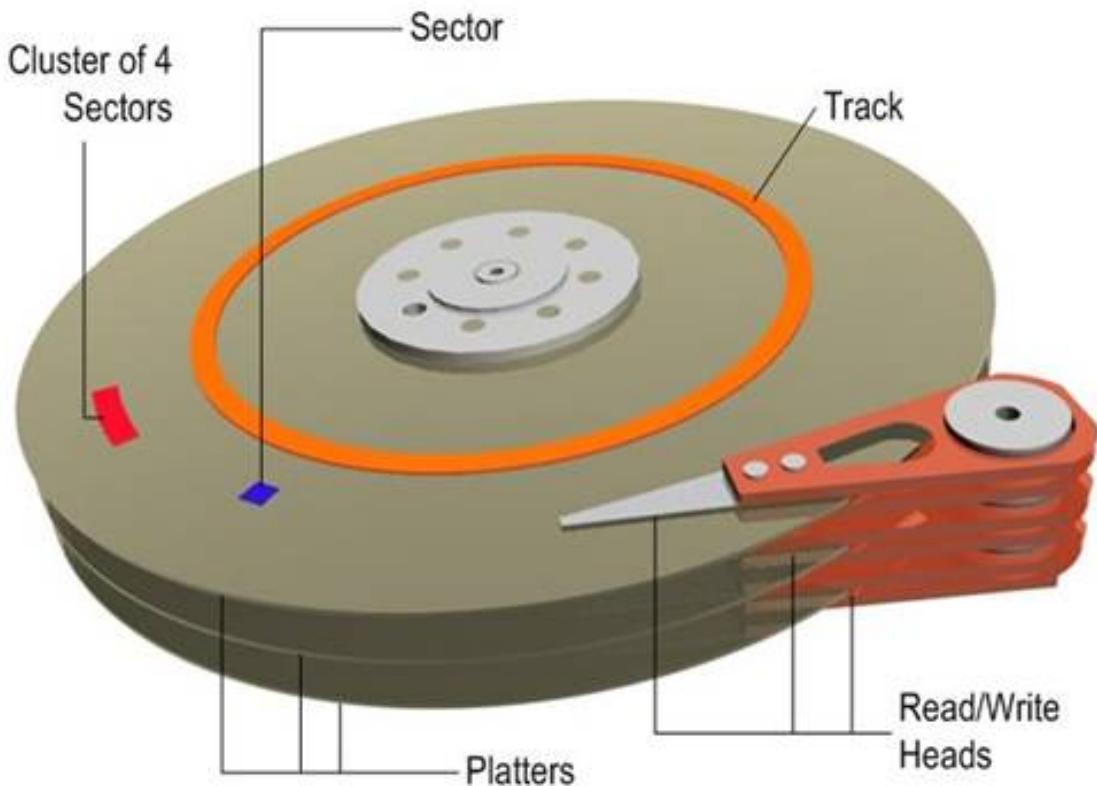
Vlak boven (en meestal ook onder) het schijfoppervlak zweeft een schijfkop (head).

Voor een goed functionerende schijf is het dus belangrijk dat de lucht rond de platters zuiver is. Daarom worden schijven bij de productie luchtdicht verpakt. De platters zitten allemaal op eenzelfde as, die door een motor aan hoge snelheid wordt rondgedraaid. Op die manier beweegt de schijfkop over het oppervlak.

Wanneer door de kop een positieve stroom loopt, worden de magnetische deeltjes op de schijf in een bepaalde zin georiënteerd. Wanneer een negatieve stroom door de kop gestuurd wordt, zullen de deeltjes in tegengestelde zin gericht worden.

Omgekeerd, als een kop, waarin geen stroom gestuurd wordt, over het schijfoppervlak beweegt, dan zal in de kop een positieve of negatieve stroom geïnduceerd worden naargelang de oriëntatie van de deeltjes op het schijfoppervlak. Wanneer de kop stil wordt gehouden, passeren een opeenvolgende reeks bits op de schijf die men een track of spoor noemt.

De verschillende sporen vormen concentrische cirkels op de schijf. Alle sporen met dezelfde straal, die op de verschillende platters liggen, vormen samen een cilinder (afbeelding 38).

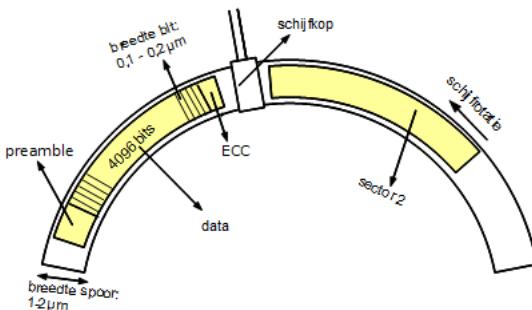


**Figure 5.2. logische indeling van een schijf (bron: [technet.microsoft.com](http://technet.microsoft.com))**

Door het ronddraaien van de schijf bewegen de koppen over een cilinder. De verplaatsing naar een andere cilinder gebeurt door een axiale verplaatsing van de arm, waarop alle koppen bevestigd zijn.

### 5.1.2. Data

De kleinste hoeveelheid data, die een kop in een keer kan verwerken is een sector. Een sector vormt een onderverdeling van een spoor en kan dus gelezen worden zonder de kop te verplaatsen.



**Figure 5.3. Onderverdeling van tracks in sectors en bits**

Bovenstaande afbeelding toont de onderverdeling van tracks in sectors en sectors in bits. Een track is opgebouwd uit een aantal sectoren met tussen de sectoren een tussenruimte.

De sectoren zelf beginnen met een preamble. Dit is een vastgelegd patroon waaraan de schijfcontroller de start van een sector kan herkennen.

Vervolgens volgen een aantal databits. Dit aantal kan variëren, maar zal bijna altijd gelijk zijn aan 4096 of 512 bytes. Bij recente schijven wordt steeds een sector size verkozen van 4096 bytes. (dit wordt Advanced Format genoemd) Door een verminderde overhead kan zo ongeveer 13% meer nuttige ruimte op een harde schijf ontstaan. [\[MANGAN\]](#)

Na de databits volgt een foutcorrigerende code (ECC - Error Correcting Code). Deze laat toe om fouten in de databits te herstellen. Meestal is dit een Reed-Solomon code. Ook deze eigenschappen werden verbeterd door het 4k advanced format van sectoren.

De verdeling van de schijf in sectoren en tussenruimte en van de sectoren in preamble, data en ECC gebeurt tijdens het formatteren. Bij het formatteren gaat er dus een stuk schijfruimte verloren aan tussenruimte, preambles en ECC-bits. Hierdoor kan de capaciteit afnemen met 15% ten opzichte van de ongeformatteerde capaciteit. Hier moet je op letten als je verschillende schijven met elkaar vergelijkt. Sommige fabrikanten geven de ongeformatteerde grootte op.

In de afbeelding zijn ook maten opgegeven voor de breedte van een spoor en de breedte van een bit in een spoor. Hoewel deze maten ondertussen al achterhaald zullen

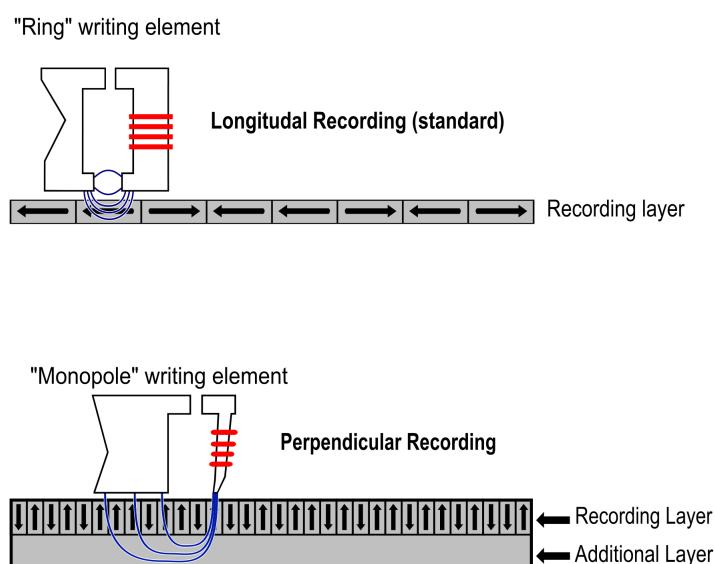
zijn, geven ze wel aan dat de bitdichtheid op een spoor groter is dan de bitdichtheid in radiale richting.

De bitdichtheid is uiteraard een belangrijke parameter, waar hard aan gewerkt wordt om deze zo groot mogelijk te maken. Een grotere bitdichtheid maakt immers schijven met grotere capaciteit mogelijk of maakt het mogelijk om kleinere platters te maken met behoud van de capaciteit.

Ondertussen botst men in het streven naar steeds grotere dichthesen op fysische limieten. Wanneer een magnetisch materiaal gemagnetiseerd wordt, worden dipool moleculen in een bepaalde richting georiënteerd. Het is niet mogelijk om de moleculen individueel te gaan draaien, het gaat steeds om een groep van moleculen die samen gericht worden. De kleinste dergelijke groep heet een magnetisch domein.

Omdat men met de dichthesen die op een spoor zitten stilaan in de buurt komt van het oriënteren van een domein en dus niet kleiner meer kan gaan in die richting, is er ondertussen een nieuwe techniek ontstaan. Deze nieuwe techniek gaat geen gebieden in de tangentiële richting gaan magnetiseren, maar gaat werken in de diepte.

Dit heet perpendicular recording (loodrechte opname).



**Figure 5.4. Perpendicular recording (bron: Wikipedia)**

### 5.1.3. Toegangstijd

De snelheid van schijven hangt van diverse factoren af. Een eerste stap is dat de schijfkop boven het juiste spoor gebracht moet worden, de kop moet in radiale richting gepositioneerd worden.

Deze actie noemt men een seek. De seek time is dus een belangrijke parameter bij een harde schijf. Dikwijls worden twee verschillende seek times opgegeven: een track-

to-track seek time en een average seek time. De eerste is de tijd nodig om de kop te bewegen tussen twee aan elkaar grenzende sporen, de tweede is de gemiddelde tijd die nodig is om de kop te verplaatsen naar een willekeurig ander spoor. Uiteraard is de eerste tijd een stuk korter dan de tweede tijd (mogelijk reeds achterhaalde richtwaarden: 1ms voor track-to-track, 10ms voor average seek time).

Eens de kop boven het juiste spoor zweeft, is er nog een rotatiewachttijd nodig alvorens de kop boven de gewenste locatie komt. Deze rotatie wachttijd is vooral afhankelijk van de rotatiesnelheid. Gangbare snelheden zijn 5400, 7200, 10000 of 15000 toeren per minuut. De gemiddelde wachttijd (average latency) is de tijd die nodig is om de schijf een halve rotatie te laten uitvoeren. Bij de vermelde snelheden is dit 5,56ms, 4,16ms en 3ms. Tenslotte is er nog de tijd nodig om de gegevens effectief te verwerken.

Een moderne schijf haalt gemiddelde overdrachtssnelheden van ongeveer 60MB/s (opgelet: niet vergelijken met de maximale overdrachtssnelheid die fabrikanten opgeven).

Bij deze snelheid duurt het 8,5ms om een sector van 512 bytes te verplaatsen. Het zal dus duidelijk zijn dat de seek time en de average latency duidelijk domineren ten opzichte van de eigenlijke overdrachtstijd. Het komt er dus ook op aan om te vermijden dat willekeurige sectoren over de schijf gelezen moeten worden.

### 5.1.4. Invloed van fysieke geometrie op de performantie

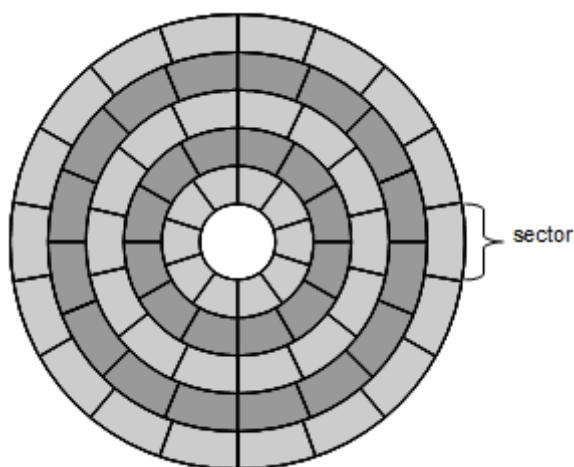
Eerder werd al de fysieke opbouw van de harde schijf besproken. Deze fysieke geometrie bepaalt mee de snelheid die gehaald kan worden.

Helaas zijn sommige eigenschappen tegenstrijdig met andere kwaliteiten die nagestreeft worden, waaronder ook het goedkoop kunnen aanbieden van de schijven. Een eerste duidelijk voorbeeld is de rotatiesnelheid van de harde schijf. Hoe sneller de schijf rond draait, hoe kleiner de average latency kan zijn. Daar staat tegenover dat een sneller draaiende schijf meer energie vraagt, wat een ongewenste eigenschap is in het geval van batterijgevoede apparaten zoals laptops. Een tweede voorbeeld is het aantal platters. Meer platters betekent dat er meer gegevens beschikbaar zijn, zonder dat de leeskoppen radiaal verplaatst moeten worden. Daar staat dan weer tegenover dat de motoren krachtiger moeten zijn (grote massa roteren of verplaatsen), met een groter verbruik tot gevolg. Door de grotere massa zullen de schijven ook trager op gang komen, waardoor het minder interessant wordt om ze uit te schakelen (wat het verbruik uiteraard nog meer negatief beïnvloedt).

Bovendien geven extra platters aanleiding tot grotere trillingen en dus meer lawaai. Tenslotte moeten de platters natuurlijk fysiek in de behuizing van de schijf passen. Een laatste voorbeeld is de capaciteit van de schijf. Als de capaciteit van de

schijf vergroot, terwijl alle andere parameters (aantal platters, grootte van platters) dezelfde blijven, dan vergroot de bitdichtheid. Een grotere bitdichtheid betekent dat bij dezelfde rotatiesnelheid bits sneller onder de kop passeren en dus sneller gelezen of geschreven kunnen worden.

Omwille van de eenvoud van de omrekening werden eerst over de gehele schijf hetzelfde aantal sectoren per cilinder gebruikt. Cilinders die aan de buitenkant van de schijf lagen, hebben een grotere omtrek. Als op een grotere omtrek eenzelfde aantal sectoren ligt met eenzelfde aantal bits per sector, dan betekent dit dat de bitdichtheid aan de buitenkant kleiner is dan aan de binnenkant en dus ook kleiner dan wat technologisch haalbaar is. Op deze manier worden dus zowel snelheid als capaciteit beperkt, voldoende redenen om de schijf ook nog eens te verdelen in zones.



**Figure 5.5. HDD opdeling in zones**

Cilinders die binnen eenzelfde zone liggen hebben hetzelfde aantal sectoren per cilinder. In de figuur is te zien dat het aantal sectoren toeneemt naar de buitenkant van de schijf toe.

### Example 5.1. voorbeeld

Je koopt jezelf een SATA harde schijf, waarvan enkele specificaties bekend zijn. 256 bytes/sector 100 sectors per track 1000 tracks per oppervlak 3 platters gemiddelde zoekijd van 8ms rotatiesnelheid is 15000 RPM

Hoe lang duurt het om tien opeenvolgende sectoren te lezen van dezelfde track?  
Hoe lang duurt het om 10 random sectoren te lezen?

Op [http://en.wikipedia.org/wiki/Hard\\_disk\\_drive](http://en.wikipedia.org/wiki/Hard_disk_drive) vind je een aanvullend educatief filmpje over de werking van de harde schijf.

### 5.1.5. Adressering

Origineel gebruikte men voor harde schijven een bijzonder logische Cylinder Head Sector (CHS)-adressering. Elke willekeurige sector wordt uniek gekenmerkt door een (kant van een) platter te kiezen (head), op die platter een cilinder (radiale positie kop boven schijf) en een sector op het geselecteerde spoor (tangentiële positie kop boven schijf).

De oorspronkelijke CHS adressering gebruikte tien bits voor de cilinder, vier bits voor de kop en 6 bits voor de sector. Merk op hoe opeenvolgende adressen zich eerst op hetzelfde spoor bevinden, vervolgens op verschillende platters en tenslotte pas de cilinder wijzigt. Zolang bestanden dus op opeenvolgende locaties staan, kunnen ze met minimale verplaatsingen (en dus minimale seek en latency times) verwerkt worden. De originele adressering had twee belangrijke nadelen.

Een eerste belangrijk nadeel is dat er slechts twintig bits beschikbaar zijn voor het adres van een sector. Bovendien startte de nummering van de eerste sector (vermoedelijk door een fout in de (in assembler geschreven) BIOS routines) bij 1 in plaats van 0. Daardoor waren er 1024 cilinders, 16 koppen en 63 sectoren mogelijk.

Met een sectorgrootte van 512 bytes betekent dit dat maximaal  $512 \times 1024 \times 16 \times 63 = 504\text{MB}$  op de schijf kunnen worden aangesproken. Een dergelijke schijf leek op dat ogenblik waarschijnlijk gigantisch, maar is op dit ogenblik belachelijk klein.

Tweede nadeel is dat de adressering direct gelinkt is aan de geometrie van de harde schijf. Een schijf met bijvoorbeeld 2048 cilinders, 8 koppen en 63 sectoren, had de maximale grootte van 504MB, maar kon niet volledig geadresseerd worden. De schijf zou slechts 252MB groot lijken.

Om het tweede probleem te omzeilen begonnen schijfcontrollers te liegen. Ze maakten de computer wijs dat ze een andere geometrie hadden dan de werkelijke fysieke opbouw en vertaalden het CHS-adres naar een fysiek adres op de schijf. De oplossing voor de capaciteit van de schijf kan enkel zijn dat er meer bits werden voorzien om de sectoren te adresseren. Een eerste uitbreiding gebruikte 10 bits voor de cilinder, 8 voor de kop en 6 voor de sector. Wat een capaciteit gaf van 8GB. Bij verdere uitbreiding van dit aantal bits koos men meteen ook voor een ander soort adressering, die niet rechtstreeks aan de geometrie van de harde schijf gelinkt was. Deze vorm van

adressering werd dan Logical Block Addressing (LBA). Hierbij krijgt elke sector gewoon een volgnummer, dat dan op de schijf zelf omgezet wordt naar een fysiek CHS-adres. Dit laatste CHS adres wordt enkel op de schijf zelf gebruikt en kan dus volledig aan de fysieke geometrie van de schijf worden aangepast.

Bij de eerste versie van LBA werden 28 bits voorzien voor het adresseren van een sector. Dit geeft een maximale grootte van  $512 * 2^{28} = 128\text{GB}$ .

Ondertussen blijkt ook dit ontoereikend en is er al een nieuwe adressering die 48 bits gebruikt en dus 128PB (petabyte) toelaat. Hiervan wordt vermoed dat dit voldoende zal zijn tot 2035. De gebruikte adressering is belangrijk omdat ze ervoor kan zorgen dat niet de volledige capaciteit van de harde schijf beschikbaar is. In het bijzonder kan dit een probleem zijn indien de BIOS nog een oudere vorm van adressering gebruikt. De bios moet immers bij het opstarten van de computer de schijf aanspreken om het besturingssysteem te laden. Dan moeten de relevante gegevens zich wel bevinden binnen de beperkingen van de adressering van de BIOS. Eens het besturingssysteem is opgestart, neemt dit de taken van de bios over en bepalen de mogelijkheden van het besturingssysteem in hoeverre de schijf volledig aangesproken kan worden.

### 5.1.6. Scheduling

Gezien de werking van een schijf zal de tijd nodig om een bepaalde sector te lezen bestaan uit drie componenten:

1. de tijd nodig om de kop boven een bepaald spoor te positioneren (seek time)
2. de tijd totdat de gewenste sector zich onder de kop bevindt (latency time)
3. de tijd nodig om de gegevens fysiek te transfereren (transfer time)

Om de totale verloren tijd zoveel mogelijk te beperken zal men de sectoren meestal niet één per één van de schijf lezen, maar in blokken of clusters. Een dergelijk blok zal minstens één sector moeten bevatten. Vaak zal een blok 4 tot 8 kB zijn terwijl een sector typisch 512 bytes is. Hoe groter de blokken, des te efficiënter de gegevensoverdracht, maar ook des te groter de interne fragmentatie gezien een blok de kleinste hoeveelheid schijfruimte is die gealloceerd kan worden. De blokgrootte kan bij de generatie van het bestandensysteem gekozen worden en zal afhangen van de kenmerken van de bestanden op de schijf. Indien men louter met zeer grote bestanden werkt, kan men de voorkeur geven aan vrij grote blokken.

Voortaan zullen we ervan uitgaan dat alle interactie met de harde schijf in blokken gebeurt.

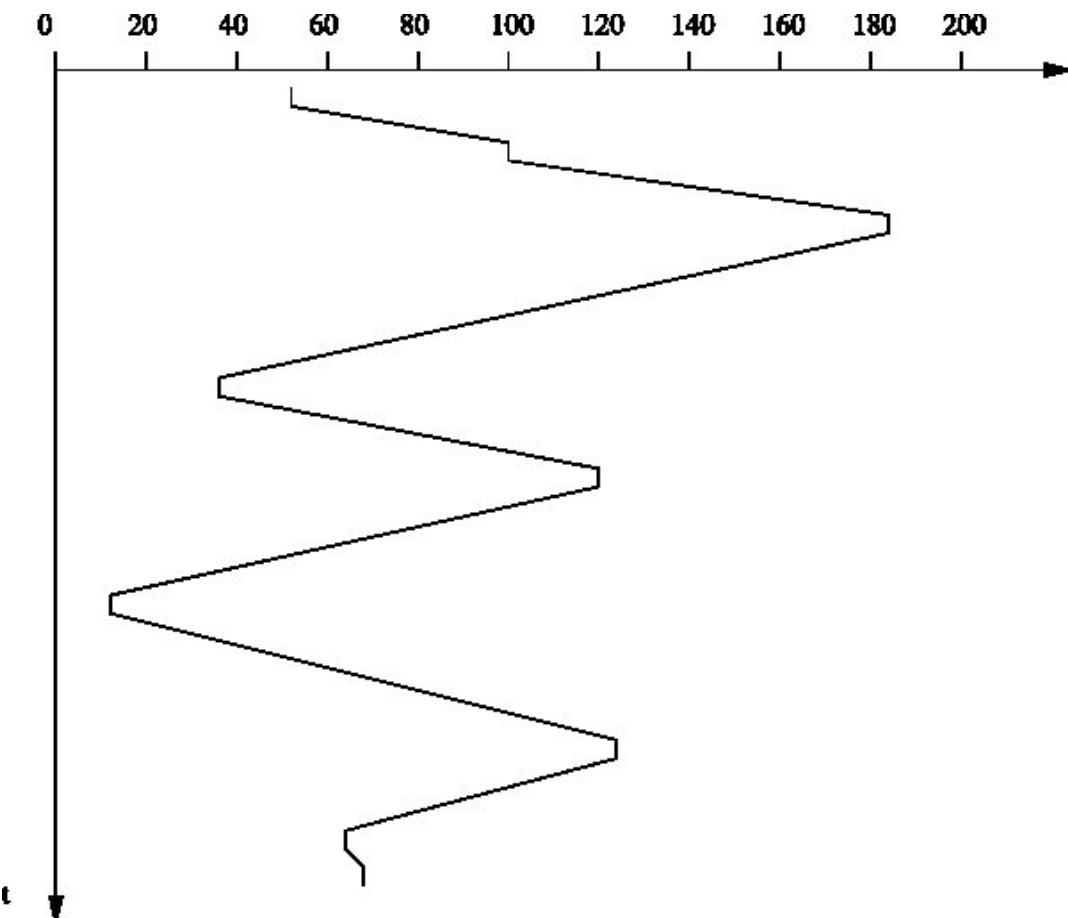
Indien er verschillende schijfoperaties staan te wachten op de schijf kunnen we, door de volgorde ervan te veranderen de prestaties van de schijf verbeteren. Deze strategie zal enkel een verschil opleveren op het ogenblik dat de schijf effectief zwaar belast wordt (en er dus keuze is).

Schijfstrategieën kunnen door het besturingssysteem geïmplementeerd worden (rekening houdend met de kenmerken van het besturingssysteem), of (meer en meer) ook door de schijfregelaar (rekening houdend met de kenmerken van de schijf). Wanneer het in de harde schijf zelf gebeurt wordt het ook wel NCQ genoemd (Native Command Queuing). Indien NCQ niet ondersteund wordt door de schijf dan zorgt enkel het besturingssysteem voor de scheduling, indien wel is het een combinatie van beide wat voor minder processorbelasting zorgt in de computer.

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** en dat de kop initieel op positie **53** staat.

### 5.1.7. FCFS: First-Come First-Serve

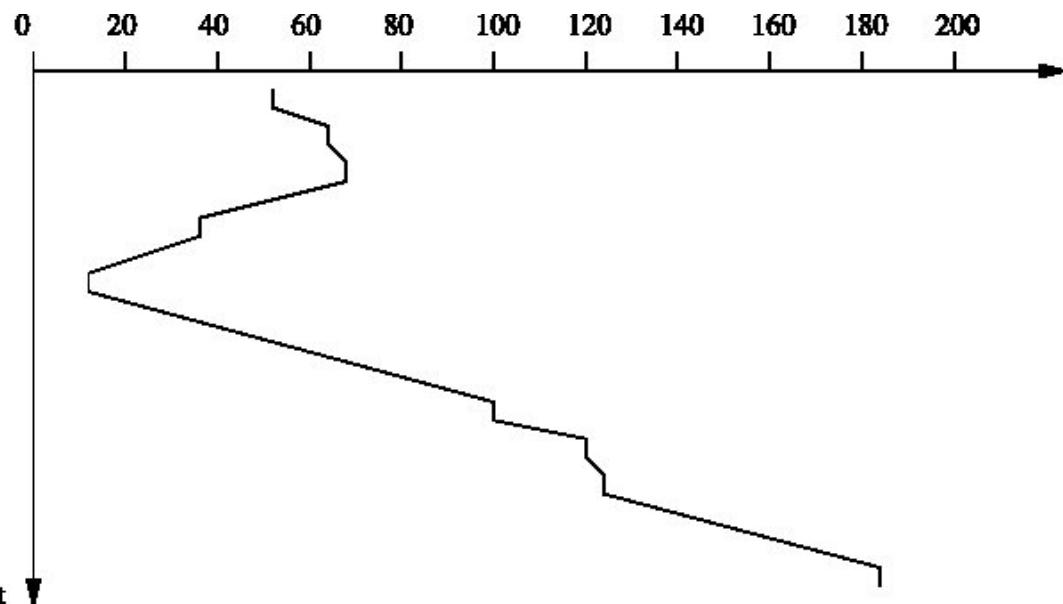
Deze strategie is de eenvoudigste, en bovendien is ze eerlijk omdat schijfoperaties niet oneindig lang kunnen uitgesteld worden. Ze kan echter niet de beste prestatie van de schijf garanderen omdat opeenvolgende schijfoperaties kunnen plaatsvinden op totaal verschillende delen van de schijf waardoor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



**Figure 5.6. First-Come First-Served Disk Scheduling**

### 5.1.8. SSTF: Shortest Seek Time First

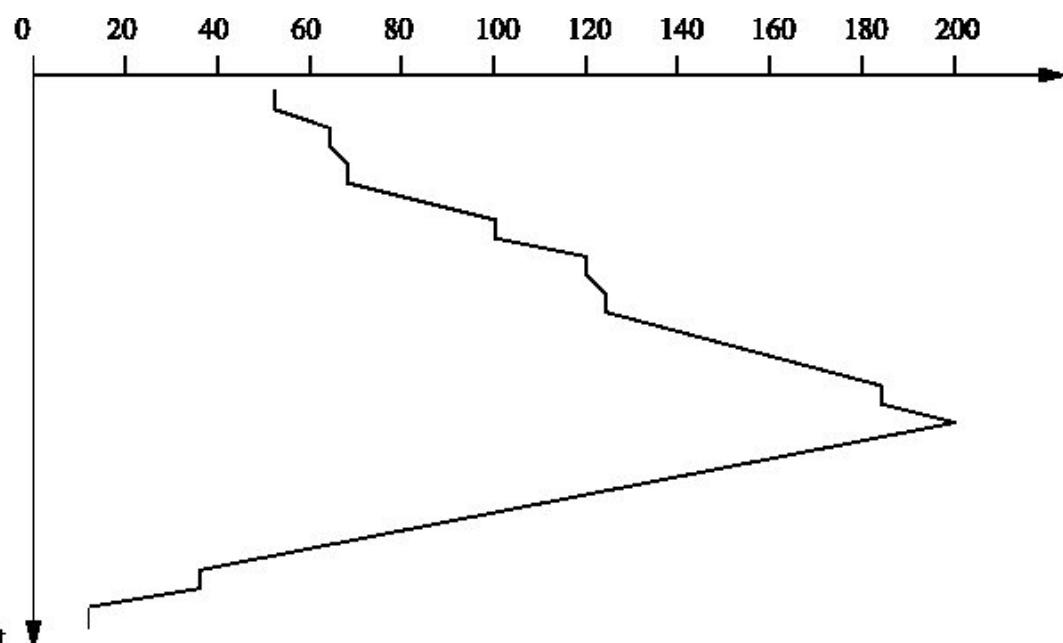
Om te vermijden dat de lees/schrijfkop zich wild heen en weer gaat bewegen over de harde schijf indien FCFS gebruikt wordt kan men de voorkeur geven aan die schijfoperaties die in de onmiddellijke nabijheid van de huidige koppositie moeten doorgevoerd worden. Het voordeel van deze strategie is dat de zoektijden hierdoor zullen afnemen, maar anderzijds is deze strategie niet langer gegarandeerd eerlijk, omdat bij een zware belasting er blijvend aanvragen kunnen binnengaan 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 de afbeelding zou het bijvoorbeeld efficiënter zijn om eerst naar cilinder 37 te gaan en dan pas SSTF toe te passen.



**Figure 5.7. Shortest Seek Time First Disk Scheduling**

### 5.1.9. SCAN: Scannen

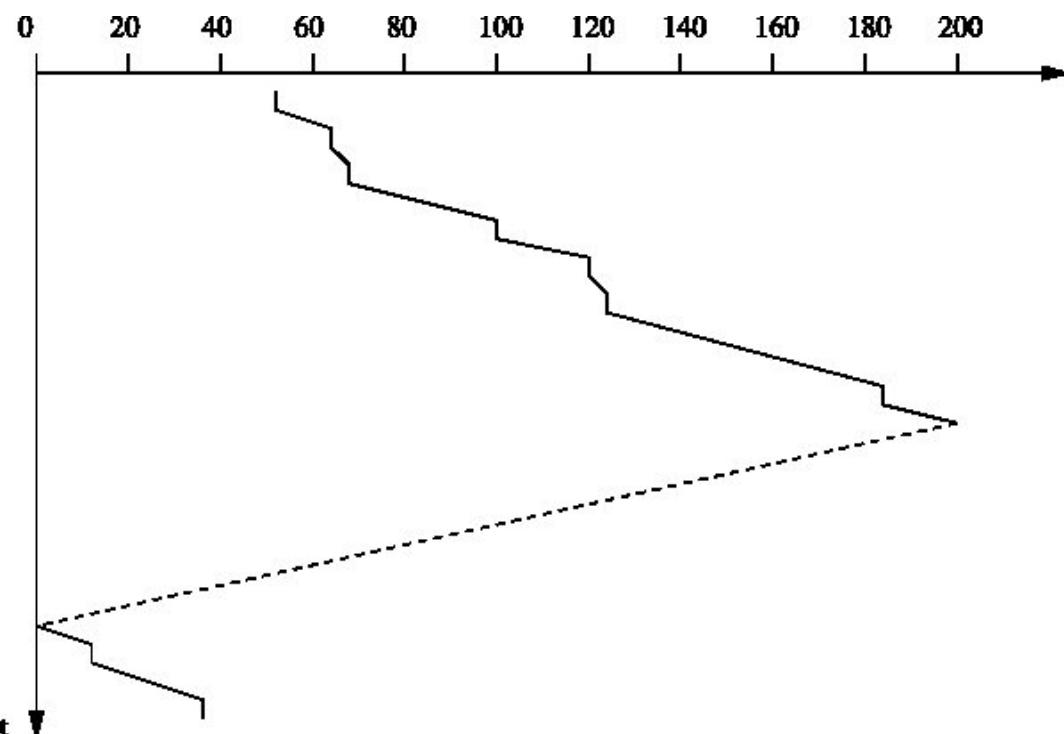
Dit algoritme tracht de voordelen van SSTF te combineren met een eerlijk gedrag. Het overloopt de schijf van het eerste tot het laatste spoor en omgekeerd hierbij alle operaties uitvoerend 'en passant'. Dit garandeert dat alle schijfoperaties finaal zullen uitgevoerd geraken en is te vergelijken met een bus die op zijn heen- en terugweg bij alle haltes stopt waar personen staan te wachten.



**Figure 5.8. Scan Disk Scheduling**

### 5.1.10. C-SCAN: Cirkulair Scannen

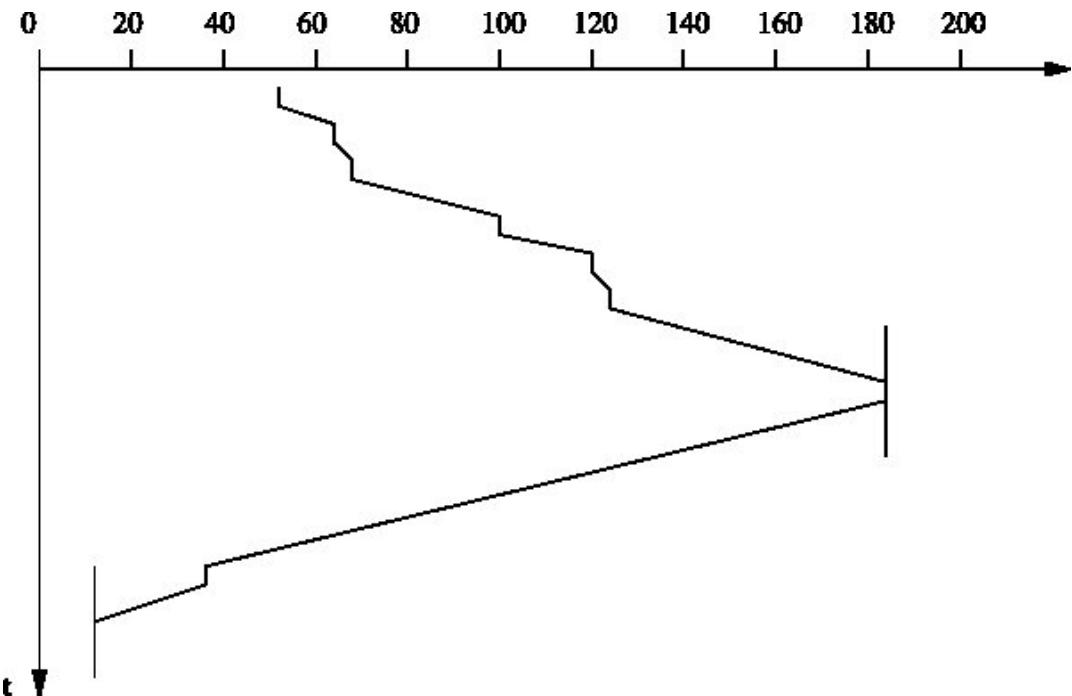
Dit is een verbetering van de SCAN strategie. Op zijn terugweg zal de kop in het begin nauwelijks of geen operaties ontmoeten (deze sporen werden immers pas net bediend). Indien er dan toch al operaties zijn, zullen ze alle zeer recent zijn. De operaties aan de overzijde van de schijf staan echter reeds geruime tijd te wachten. Daarom zal C-SCAN op zijn terugweg geen sporen bedienen maar meteen bij spoor 0 herbeginnen (zie figuur). Hier zullen de gemiddeld oudere aanvragen eerst bediend worden. Dit algoritme is in zekere zin eerlijker dan SCAN, maar 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.



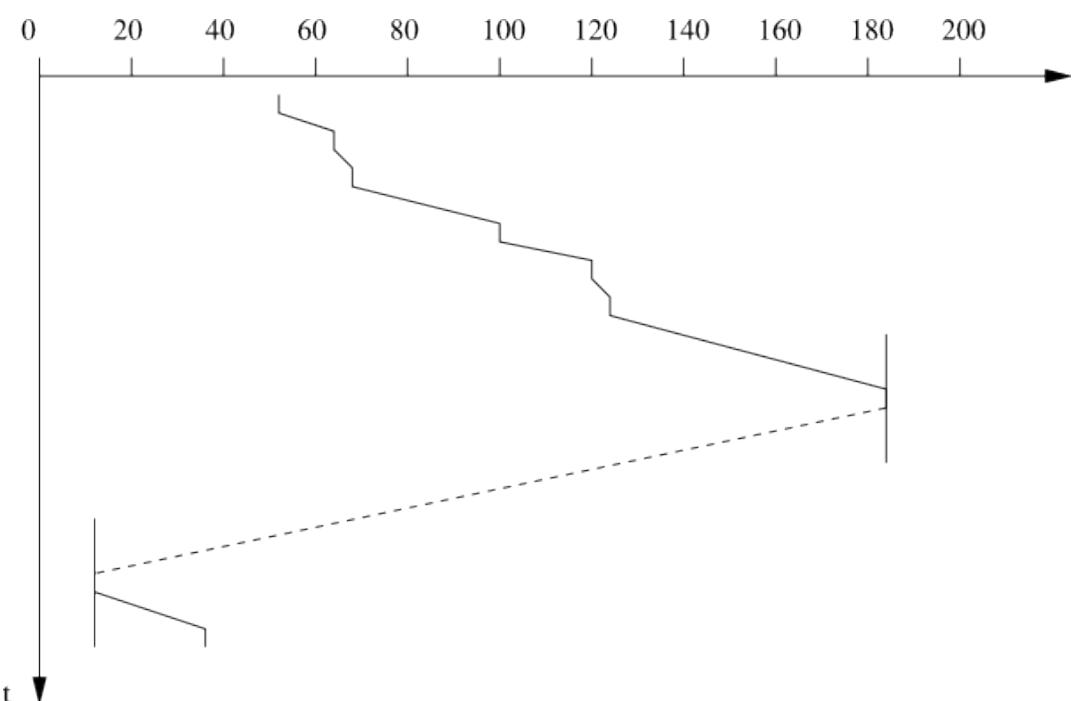
**Figure 5.9. Circular Scan Disk Scheduling**

### 5.1.11. LOOK + C-LOOK: Aangepaste SCAN + C-SCAN

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)



**Figure 5.10. Look Disk Scheduling**



**Figure 5.11. Cirkular Look Disk Scheduling**

### 5.1.12. Schijfcontroller

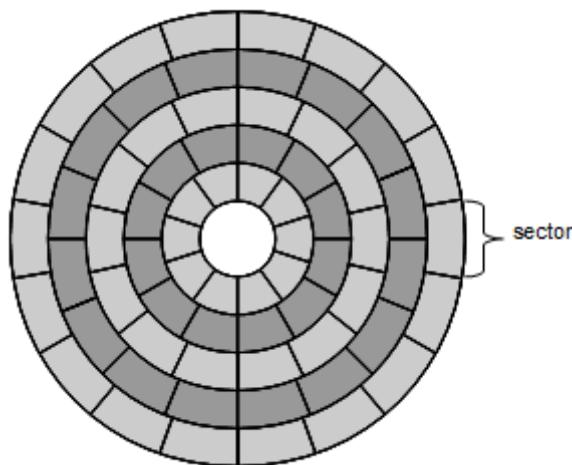
Een laatste belangrijk onderdeel van de harde schijf is de schijfcontroller. Deze stuurt enerzijds de motoren en leeskoppen aan, zodat de juiste gegevens bereikt worden, anderzijds verzorgt hij de communicatie met de buitenwereld.

Zoals al eerder vermeld zal deze controller onder andere een omzetting doen van de LBA adressen die hij binnenkrijgt, naar een fysieke kop, cilinder en sector.

Samen met deze vertaling houdt de controller ook een mapping bij van slechte sectoren. Sommige sectoren kunnen een permanent gemagnetiseerde plek vertonen, waardoor ze onbruikbaar worden voor het opslaan van variabele informatie. Op elk spoor worden een aantal reserve sectoren voorzien, die de plaats kunnen innemen van een van de defecte sectoren.

Uiteraard moet de controller dan zijn mapping van logische naar fysieke adressering aanpassen. Controllers kunnen ook sectoren bufferen in een cache, zodat operaties naar de schijf sneller kunnen verlopen.

## 5.2. Solid state drive



**Figure 5.12. SSD drive zonder behuizing**

In plaats van een magnetische schijf wordt in een SSD gebruik gemaakt van DRAM of flash geheugen om de gegevens op te slaan. DRAM heeft uiteraard als nadeel dat met het wegvalLEN van de stroom ook de gegevens verdwijnen.

Daarom worden dergelijke ‘schijven’ uitgerust met een batterij, die toelaat om de gegevens nog naar een backup schijf te schrijven. Dergelijke schijven zijn geschikt om dienst te doen als opslagmedium voor de swap. Hoewel er nog een beperking is van de interface waarmee de SSD verbonden is met het systeem, zal dit sneller zijn dan een magnetische schijf. Daar staat tegenover dat DRAM duur is en dat het logischer is om het gewoon rechtstreeks in de geheugenbanken te voorzien.

Deze oplossing is dus vooral interessant als het fysieke geheugen niet uitgebreid kan worden (bijvoorbeeld maximum 4GB met 32-bit processor).

Veel fabrikanten werken aan drives met flash geheugen. Deze technologie is een stuk goedkoper en verliest zijn inhoud niet bij het wegvalLEN van de stroom. Hoewel flash nog steeds een stuk duurder is dan magnetische opslag, lijkt deze technologie toch een alternatief te kunnen vormen voor magnetische harde schijven.

Recente productlanceringen lijken erop te wijzen dat deze technologie heel snel matuur wordt.

In 2014 lijkt de SSD-prijs te stabiliseren rond 0.8USD/GB [[PCPART](#)], en de trend is nog steeds dalend.

De meeste voordelen volgen uit het ontbreken van mechanische onderdelen:

- De schijf moet niet draaien en moet dus ook niet opstarten, geen kop die bewogen moet worden, dus willekeurige toegang kan sneller.
- Fragmentatie van bestanden wordt minder belangrijk
- Geen bewegende onderdelen, dus geen lawaai
- Stroomverbruik ligt lager dan bij conventionele schijven
- Beter bestand tegen schokken, temperatuur, hoge hoogte tenzij voor high-end schijven, ligt het verbruik lager

Uiteraard zijn er ook nadelen. Zoals al vermeld is er voorlopig nog de kostprijs. Daarnaast is er ook de specifieke eigenschap van flash dat het slechts een beperkt aantal keer beschrijfbaar is.

Door recente ontwikkelingen valt dit nadeel echter bijna volledig weg. Tegenwoordig zijn cellen in SSD-schijven meer dan 1 000 000 keer herschrijfbaar, en dat zou voldoende moeten zijn om bij normaal gebruik vele jaren correct te functioneren. Behalve bij erg specifieke schrijfintensieve taken (vb logging server) zal deze limiet nooit een probleem vormen.

Een belangrijke rol hierbij is weggelegd voor de controller, het intelligente hart van de SSD-schijf. Die zal ervoor zorgen dat elk deel van het geheugen ongeveer evenveel beschreven wordt, zodat de slijtage over het hele geheugenbereik ongeveer gelijk is. Aangezien verspreidde bestanden toch geen invloed hebben op de snelheid, veroorzaakt dit geen performantieverlies.

Het gebruik van de erase-blocks maakt ook dat meer gegevens moeten worden aangepast, waardoor echt random schrijven trager wordt (veel meer extra gegevens die moeten worden aangepast). SSD's zijn eigenlijk al goed ingeburgerd onder de vorm van geheugenkaarten en USB sticks. Stilaan beginnen ook laptops op te duiken die voorzien zijn van een solid state schijf.

Ook voor specifieke servertoepassingen wordt de SSD stilaan een interessant alternatief voor 'gewone schijven'. Denk daarbij maar aan webservers, die heel veel

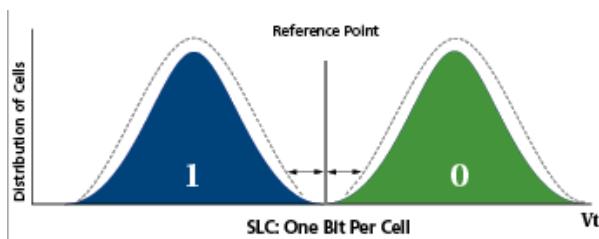
kleine bestanden (webpagina's) moeten lezen vanop de schijf: die hebben duidelijk baat bij snelle access-tijden.

### 5.2.1. Flash technologie

Flash geheugen bestaat uit cellen die in staat zijn om een spanningsniveau te onthouden. er onderscheiden zich twee belangrijke types:

#### SLC

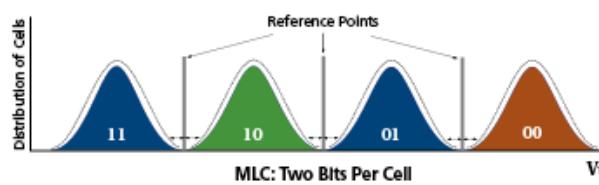
Flash-geheugen bestaat uit cellen die data bevatten. Bij SLC zal elke cel één bit bevatten. Dat is de meest eenvoudige en snelle manier om geheugen te produceren, maar helaas ook de duurste: je hebt immers veel cellen nodig om een grote capaciteit te behalen.



**Figure 5.13. SLC geheugen**

#### MLC

MLC (multi level cell) houdt in dat iedere cel in het geheugen meerdere bits kan onthouden. Beschouw daarvoor onderstaande figuur:



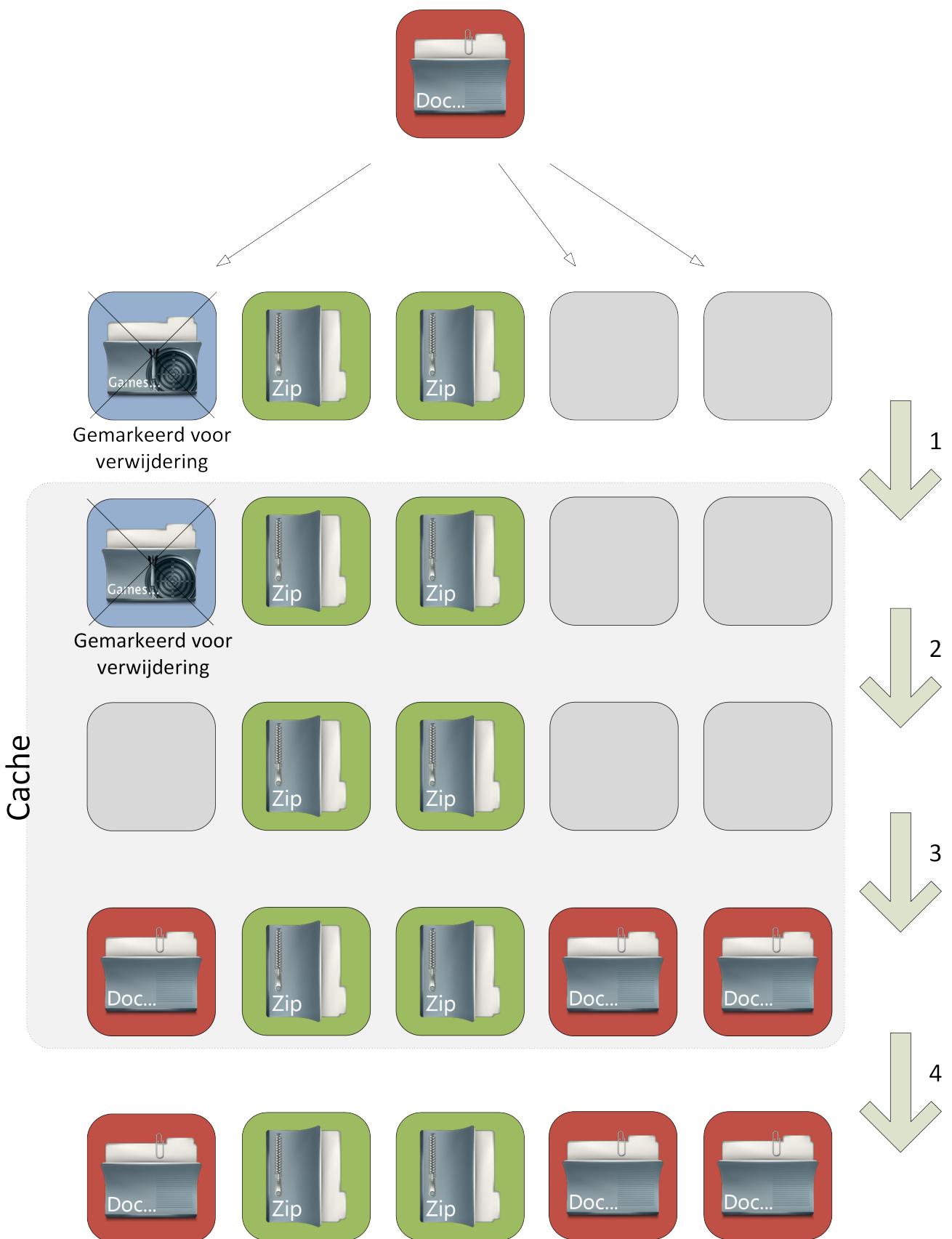
**Figure 5.14. MLC geheugen**

Iedere cel zal dus in staat zijn om bijvoorbeeld vier verschillende spanningsniveaus te bewaren. Je ziet dat MLC gevoeliger zal zijn voor fouten: als er een klein beetje van het spanningsniveau is weggelekt zal je de data niet meer juist uitlezen.

Vanwege die nood aan precisie zal MLC dus trager gelezen en beschreven kunnen worden. MLC wordt dus voornamelijk gebruikt waar een grote capaciteit belangrijker is dan een erg grote betrouwbaarheid of snelheid.

### 5.2.2. Schrijfcyclus van Flash Memory

Het beschrijven van Flash is in tegenstelling tot lezen (gebeurt in één beweging) een heel karwei. Beschouw onderstaande figuur, waarbij een document moet geschreven worden, gespreid over drie plaatsen in een datablok.



**Figure 5.15. Schrijfcyclus Flash**

De verschillende stappen die nodig zijn gaan als volgt: . De huidige data wordt in de snelle cache ingelezen . De cellen die aangeduid staan als waardeloos worden effectief leeggemaakt . De huidige data en de te schrijven data worden samengevoegd in de cache . De cache wordt terug weggeschreven naar een (leeg!) block in flash.

### 5.2.3. Optimalisatie van Flash

De SSD-technologie staat nog in zijn kinderschoenen. De recentste types kampen dan ook vaak nog met problemen. Zeker als ze zwaar belast worden vallen ze vaak door de mand. Ook het besturingssysteem vervult hierin een belangrijke rol. Enkele van de problematieken...

#### Buffering

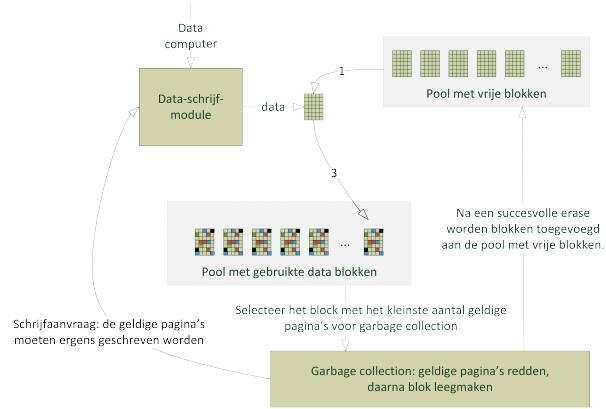
Het concept van cache buffering is jullie uiteraard niet onbekend. Een bijzonderheid van flashgeheugen is dat je voor het schrijven eerst de benodigde blokken moet wissen. Bij reeds gewiste cellen is dit uiteraard niet nodig. Je begrijpt dat deze werkwijze flink wat tijd vraagt. Een cache zal dan ook de schrijfacties bufferen. Als de cache te klein is, zal na een periode van continu schrijven de prestatie sterk degraderen.

#### Schijfverlamming

Enkele maanden na het lanceren van SSD's in 2009 zaten verschillende reviewers met de handen in het haar. Toen ze de schijven reviewden vlak na verschijnen waren die erg snel en enkele maanden later bleek daar amper nog iets van over te blijven. [4]

De verklaring zit in de manier waarop data geschreven wordt naar SSD disks. Uit de uitleg bij 'cache' had je al begrepen dat schrijven enkel snel lukt als dat gebeurt naar volledig vrije blokken, anders moeten ze eerst ingelezen/leeggemaakt worden.

Naarmate de tijd vordert zullen die vrije blokken uiteraard zeldzamer worden. De fabrikanten die getroffen werden door dit gênant verschijnsel bakten daarom een nieuwe firmware. Ze bouwden een feature in die soms wel 'garbage collection' genoemd wordt. Het concept is dat je schijf probeert op rustige momenten om op schijfniveau te defragmenteren. informatie in blokken wordt samengevoegd zodat er terug lege blokken bij komen. Die zijn immers veel sneller te beschrijven dan 'half gevulde' blokken.



**Figure 5.16. Garbage collection bij Flash Memory**

Een bijkomende manier om hetzelfde probleem te verhelpen is het besturingssysteem laten communiceren met de SSD door middel van het ATA-Trim commando. Het besturingssysteem zal met dit commando aan de schijf vertellen welke data niet meer nodig is. (bijvoorbeeld bestanden die door NTFS als verwijderd staan aangeduid) Dit commando werd geïntroduceerd bij Windows 7 en is ingebakken in Linux kernels vanaf 2.6.28. Zo wordt vermeden dat sectoren verplaatst worden op de SSD, die door NTFS reeds lang als verwijderd werden aangeduid.

## Alignment

Bij SSD's is het belangrijk dat de grenzen van partities exact overeenkomen met de grenzen van SSD-blocks. Zo komt elke cluster (4k) precies overeen met één block (4k) op de harde schijf. Als er dan data moet geschreven worden in een cluster, dan hoeft maar één block geschreven te worden, en geen twee. Vista en Windows 7 schijnen hier in de setup rekening mee te houden. Van 3rd party tools is dat natuurlijk minder zeker. Je kopieert een gewone schijf dan ook beter niet via sector-based copy tools als Ghost of dd (linux commando)

## 5.3. Snelheid bij disks: IOPS

Zie bijkomend lesmateriaal op Toledo === Opslagmedia combineren: RAID

### 5.3.1. Types

#### RAID0

Zie slides op Toledo

## RAID1

Zie slides op Toledo

## RAID10

Zie slides op Toledo

## RAID5

Zie slides op Toledo

### 5.3.2. Gecombineerde snelheid

Zie slides op Toledo

## 5.4. Logische indeling opslagmedia

In het voorgaande werd de harde schijf voornamelijk bekeken vanuit zijn fysieke kenmerken. In wat volgt zullen we de logische indeling van de schijf bekijken. We zullen met andere woorden bekijken hoe de schijf toegankelijk is voor bijvoorbeeld het besturingssysteem. Harde schijven worden in de eerste plaats verdeeld in één of meerdere partities (= fysieke formatering), terwijl elke partitie dan georganiseerd wordt volgens de regels van een welbepaald bestandsysteem. (=logische formatering)

Sommige toepassingen die zeer intensief gebruik maken van secundair geheugen (zoals gegevensbanken) vereisen dat een schijf(-partitie) niet logisch geformateerd wordt. Zij gebruiken de 'raw disk' en gebruiken hun eigen logische formatering. Hetzelfde geldt voor de swapruimte op een aantal systemen. De bedoeling is steeds om de overhead van het bestandensysteem te omzeilen.

### 5.4.1. Boot blok

Bij het opstarten van een computer is het niet duidelijk wat die computer precies zal moeten uitvoeren (Linux, MS-DOS, Windows). De keuze van het besturingssysteem zal onder andere gemaakt worden door de inhoud van de schijf die als 'systeemschijf' bekend staat. Het opstarten van een computersysteem staat bekend onder de naam 'bootstrappen'. Na het aanleggen van de spanning begint de processor code uit te voeren op een bepaalde vaste plaats in het geheugen. Door op die plaats een ROM-geheugen aan te brengen (geheugen met een voorgedefinieerde vaste inhoud),

zal een programma beginnen uitvoeren. Dit (kleine) generische programma gaat na of de hardware van de computer zich gedraagt zoals het hoort, en gaat dan op zoek naar de systeemschijf en tracht daar één of meerdere sectoren te lezen (de zogenaamde boot sectoren). Deze sectoren bevatten de code die nodig is om andere delen van het besturingssysteem in te lezen. Deze code zal op zijn beurt het volledige besturingssysteem inladen, de configuratiebestanden lezen en de controle overleveren aan het besturingssysteem. Het proces van bootstrappen is dus een proces waarbij stap voor stap complexere software van de schijf ingeladen wordt en er gaandeweg meer en meer functies van het besturingssysteem ter beschikking komen.

### 5.4.2. Master Boot Record (MBR) layout

**Table 5.1. Inhoud Master Boot Record**

offset	lengte (bytes)	inhoud
0	446	MBR programmacode
446 (1BEh)	16	eerste partitie-record
462 (1CEh)	16	tweede partitie-record
478 (1DEh)	16	derde partitie-record
494 (1EEh)	16	vierde partitie-record
510 (1FEh)	2	55 AA (einde markering)

Het master boot record is de eerste fysieke sector van de harde schijf. Deze bevat twee gedeeltes: de primaire partitietabel en de MBR-programmacode. De MBR programmacode bevat de nodige instructies om te beslissen van welke primaire partitie opgestart moet worden, om de bootsector van de betreffende partitie in het geheugen te laden en om te starten met het laden van het betreffende besturingssysteem. In de meest eenvoudige vorm zoekt de MBR-code in de partitietabel naar de actieve partitie (zie verder), laadt de bootsector van die partitie in het geheugen en voert de code uit die daarin opgeslagen is. Soms kan de MBR-code ook meer gesofisticeerd zijn en interactie met de gebruiker mogelijk maken.

Dit is het geval bij Boot-managers als LILO (Linux Loader), grub en BootMagic. Partitionering is het onderverdelen van de harde schijf in verschillende blokken, met elk een bestandsysteem. Partitioneren kan interessant zijn om bijvoorbeeld verschillende soorten gegevens te groeperen (bijvoorbeeld aparte data partitie, die niet aangepast wordt als het besturingssysteem opnieuw geïnstalleerd wordt). Het kan ook gebruikt worden om bijvoorbeeld een aparte swap-partitie te voorzien of om meerdere besturingssystemen (multi-boot systeem) mogelijk te maken. Bij het partitioneren is het

wel belangrijk om op voorhand goed na te denken wat je wil, want achteraf aanpassen van groottes van partities is (afhankelijk van het gebruikte bestandsysteem) niet altijd mogelijk en dikwijls gevaarlijk voor dataverlies.

Hoe een schijf is onderverdeeld in partities wordt opgeslagen in een partitietabel.

De primaire partitietabel bevat voor elke primaire partitie een record van zestien bytes.

De inhoud van zo'n record wordt weergegeven in onderstaande tabel.

**Table 5.2. Inhoud partitietabel-record**

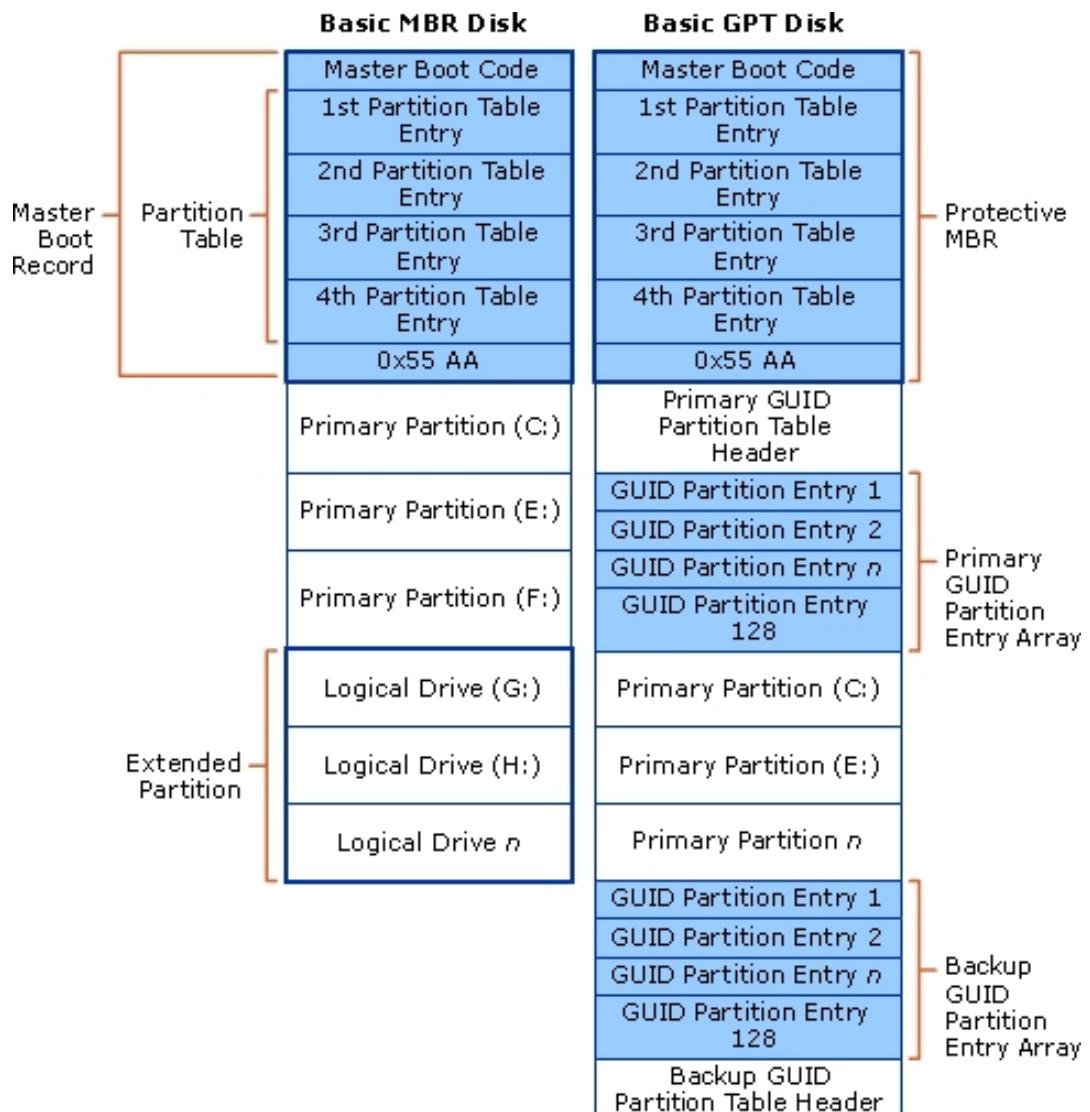
offset	lengte (bytes)	inhoud
0	1	80h (actieve partitie) of 00h (niet actief)
1	3	CHS adres eerste sector van de partitie
4	1	type partitie
5	3	CHS adres laatste sector van partitie
8	4	LBA adres eerste sector
12	4	aantal sectoren in partitietabel

Het eerste byte duidt aan of de partitie actief is of niet. Dit kan belangrijk zijn, in het bijzonder in het geval waarbij de MBR op zoek gaat naar de actieve partitie om te booten. Daarnaast zijn er een aantal parameters die de locatie en grootte van de partitie vastleggen en er is een byte dat het type vastlegt. Met dit byte kan aangegeven worden welk bestandsysteem op de partitie staat (b.v. FAT16, FAT32, EXFAT, NTFS, EXT4, ZFS, ...).

Aangezien er slechts vier records zijn in de primaire partitietabel, kunnen slechts vier primaire partities gedefinieerd worden. Indien meer partities nodig zijn, moet gebruik gemaakt worden van extended of uitgebreide partities. Belangrijke opmerking hierbij is dat niet alle informatie op een extended partitie terecht mag komen. Een belangrijk voorbeeld is een partitie waar Windows op geïnstalleerd wordt. In de primaire partitietabel is er slechts een extended partitie mogelijk. Deze partitie krijgt een type aanduiding die aangeeft dat het gaat om een extended partitie.

Op de eerste sector van de extended partitie bevindt zich een extended master boot record (EMBR). In het EMBR is er plaats voor twee partitabel-records. Een van de partities kan weer een uitgebreide partitie zijn.

Op die manier kunnen in principe oneindig veel partities aangemaakt worden (al zijn er natuurlijk wel beperkingen, zoals de eindige capaciteit van de schijf). De partities die aangemaakt worden binnen een uitgebreide partitie, noemt men logische partities.

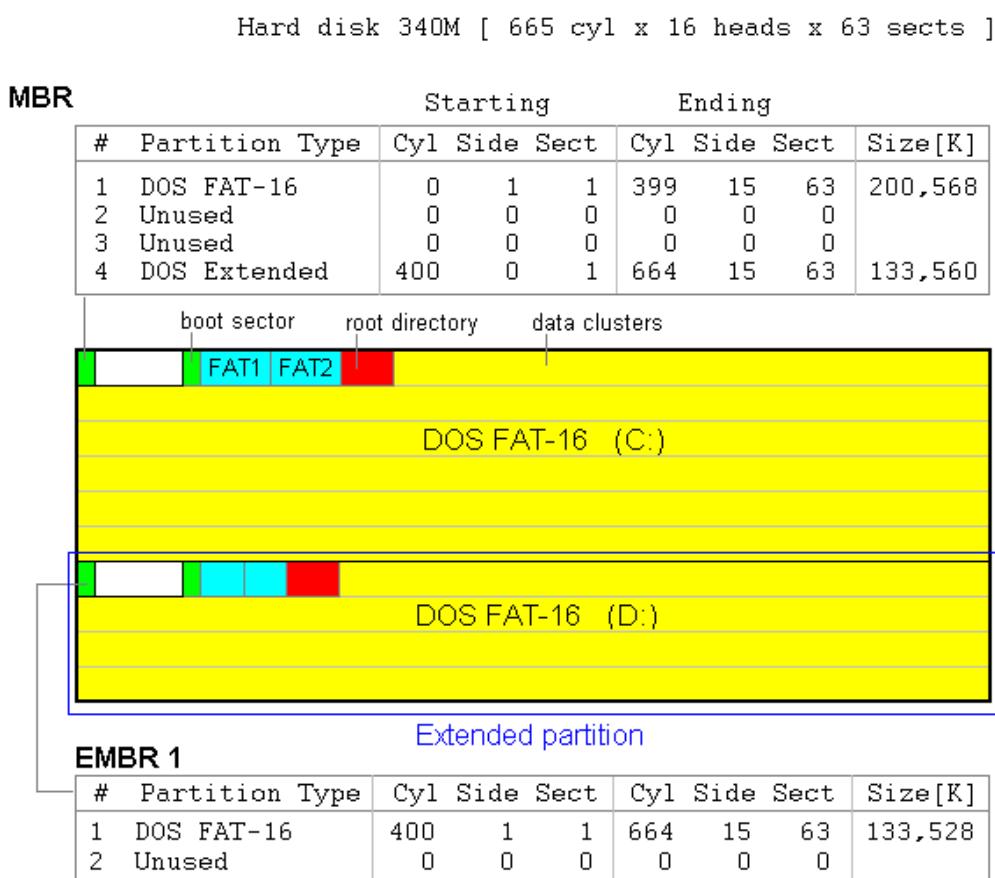


**Figure 5.17. partiestructuur (bron:Microsoft Technet)**

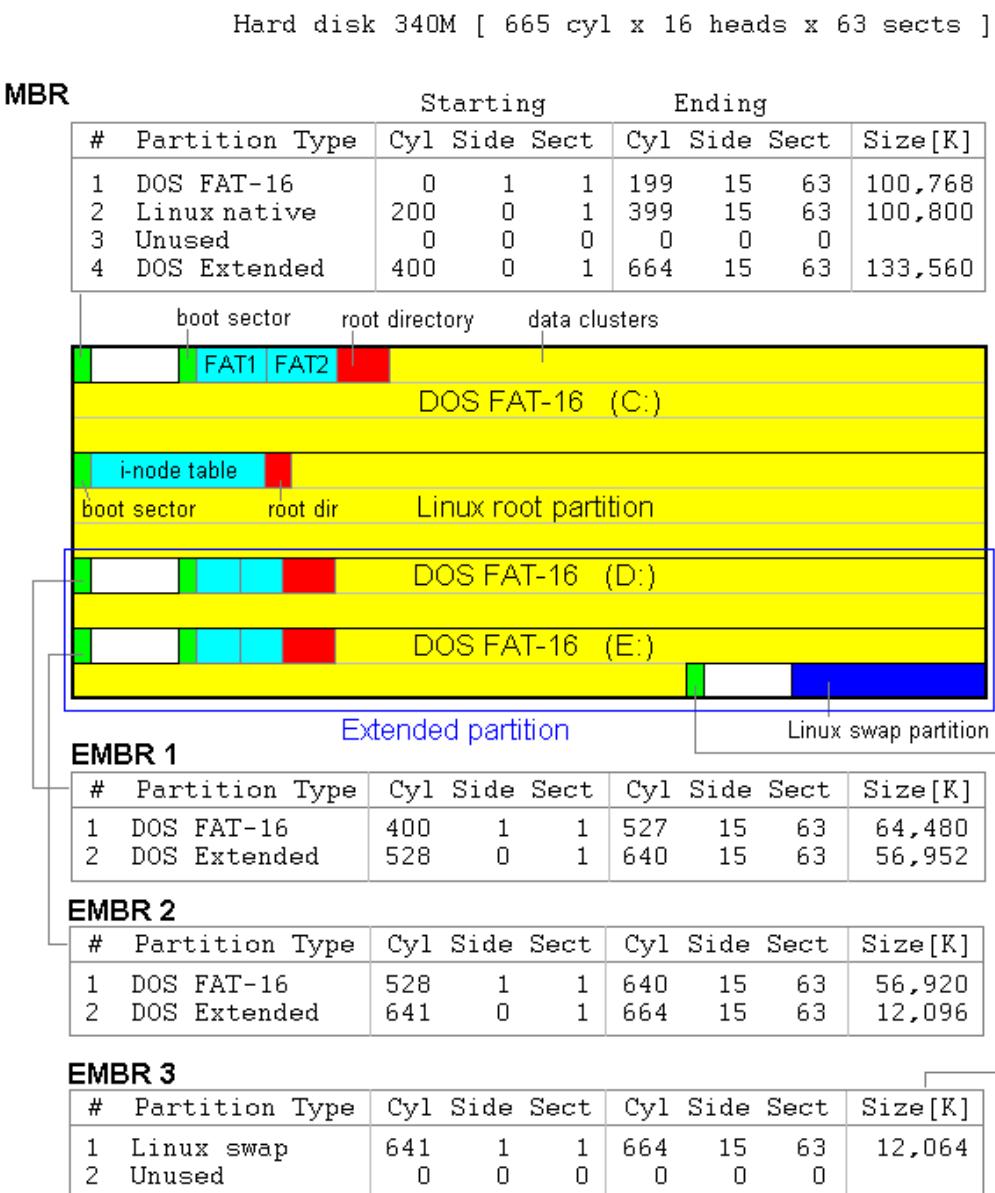
Meestal moet men slechts eenmaal een extended partitie aanmaken en kan men vervolgens in deze partitie logische partities definiëren. Het partitioneringsprogramma maakt automatisch de nodige EMBR's aan. Bij het aanmaken van een extended partitie moet je goed opletten dat je voldoende ruimte voorziet voor het definiëren van alle logische partities.

In onderstaande twee afbeeldingen worden twee voorbeelden gegeven van de indeling van een harde schijf. Het eerste is een eenvoudig voorbeeld met een primaire en een logische DOS-partitie (Microsoft laat slechts een FAT per partitietabel toe). Het tweede voorbeeld toont een complexer voorbeeld met meerdere logische partities. Nu is te

zien dat elke extended partitie een EMBR bevat waarin informatie zit voor een logische partitie en een verwijzing naar de volgende extended partitie.



**Figure 5.18. Voorbeeld partitietabel met uitgebreide partitie**



**Figure 5.19. Partitietabel met meerdere logische partities**

### 5.4.3. GPT layout

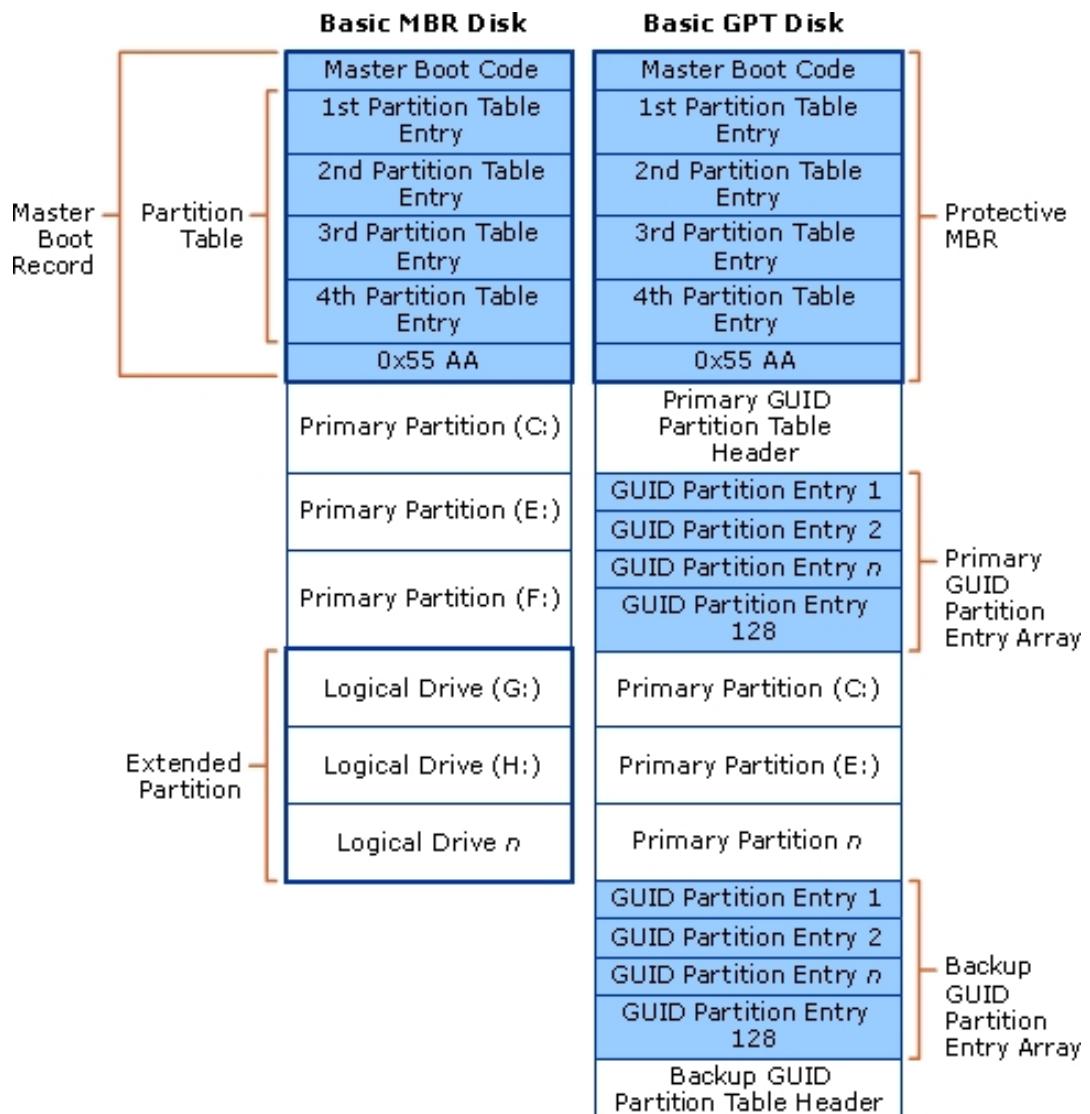
De disk-layout met het MBR wordt tegenwoordig nog vaak gebruikt. Toch zijn er enkele belangrijke nadelen:

- Partitiegrootte is beperkt tot 2TB, wat met de huidige nieuwe harde schijven problematisch wordt.
- De partitietabel is een erg belangrijk stukje data op de schijf, maar het wordt op geen enkele manier beschermd. Als deze cluster defect is, dan is het moeilijk om de logische layout van de schijf te achterhalen.

De GPT layout probeert hier oplossingen voor te verzinnen. Zo staat de partitietabel ook op het einde van de schijf, zodat een defect in het begin van de schijf niet hoeft te betekenen dat je de data op de partities kwijt bent. De eerste sector van een GPT-schijf bevat een valse MBR-record (protective MBR) om oude partitioningstools te misleiden.

Het maximale aantal partities is bij GPT 128 stuks. De grootte voor elke partitie is geen limiet meer. (om precies te zijn: bij sectoren van 512 bytes kan je  $2^{64} * 512$  bits opslaan in elke partitie.)+ Elke partitie-entry bij GPT bevat volgende gegevens:

#bytes	Naam	Beschrijving
16 bytes	Partition Type GUID	Bevat een GUID die zegt over welk soort partitie het gaat. Vb Linux Swap Partition, Windows Basic Data Partition, Apple HFS + partitie, ...
16 bytes	Unique Partition GUID	Unieke GUID voor deze partitie
8 bytes	Starting LBA	Begin van de partitie
8 bytes	Ending LBA	Einde van de partitie
8 bytes	Attribute bits	Extra info over de partitie
72 bytes	Partition name	Leesbare naam voor de partitie



**Figure 5.20. GPT versus MBR layout (bron: Microsoft Technet)**

## Bestanden

Het probleem van de allocatie van bestanden betreft de vraag op welke manier de blokken of clusters van de schijf samengevoegd kunnen worden tot een bestand. Er zijn verschillende mogelijkheden:

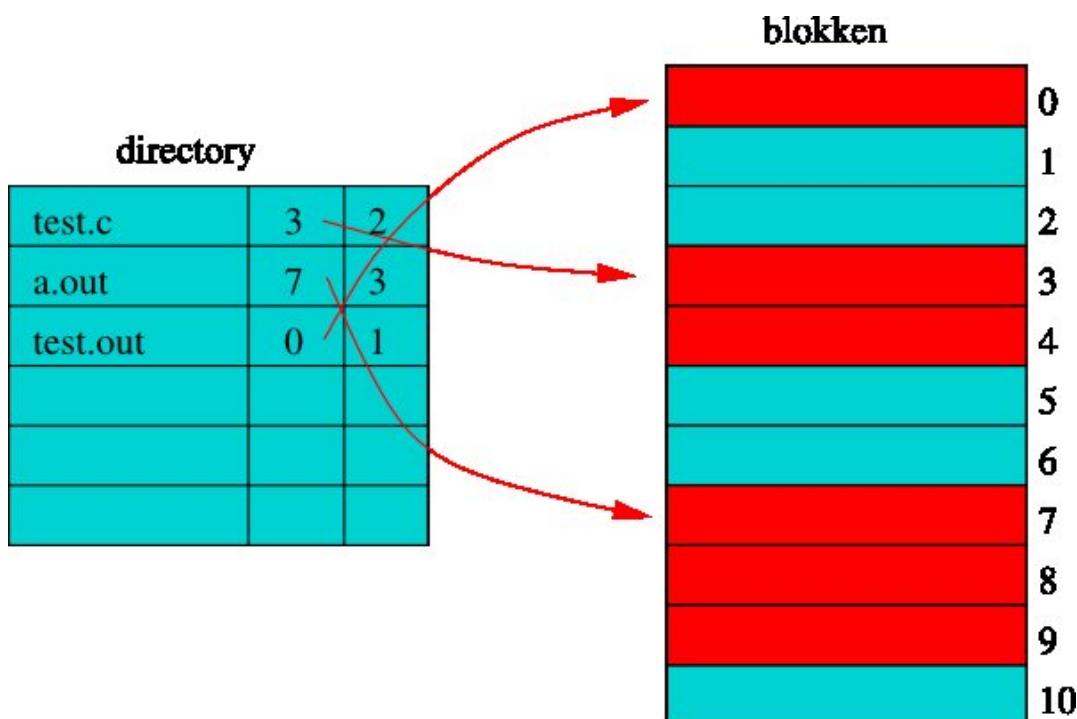
- Contigu
- Gelinkt
- Gebaseerd op een allocatietafel
- Gebaseerd op een indextafel.

De keuze van een bepaalde methode zal onder meer afhangen van de manier waarop de gegevens van een schijf gebruikt zullen worden: sequentieel, direct of geïndexeerd,

en van de prestaties die men van de schijf verwacht. Doordat de schijf zeer traag is in vergelijking met de processor, zal men doorgaans trachten om het aantal toegangen naar de schijf zoveel mogelijk te beperken door bepaalde tabellen in het geheugen te houden, door gegevens intelligent over de schijf/schijven te verdelen zodat de koppbewegingen beperkt worden, enz. Vaak kan het de moeite lopen om de processor een complexer algoritme te laten uitvoeren indien dit het gebruik van de schijf kan verminderen.

### Contigue allocatie

Contigue allocatie is de eenvoudigste allocatiemethode. Een bestand bestaande uit n blokken zal n opeenvolgende fysieke blokken op de schijf innemen. De verplaatsingen van de lees/schrijfkop zullen hierdoor minimaal zijn: bij het sequentieel lezen zal de kop enkel op het einde van het laatste spoor van een cilinder moeten veranderen naar een volgende cilinder en ook bij directe toegang zullen de verplaatsingen van de kop minimaal zijn. Contigue allocatie is hierdoor de snelste allocatiemethode.



**Figure 5.21. Contigue allocatie**

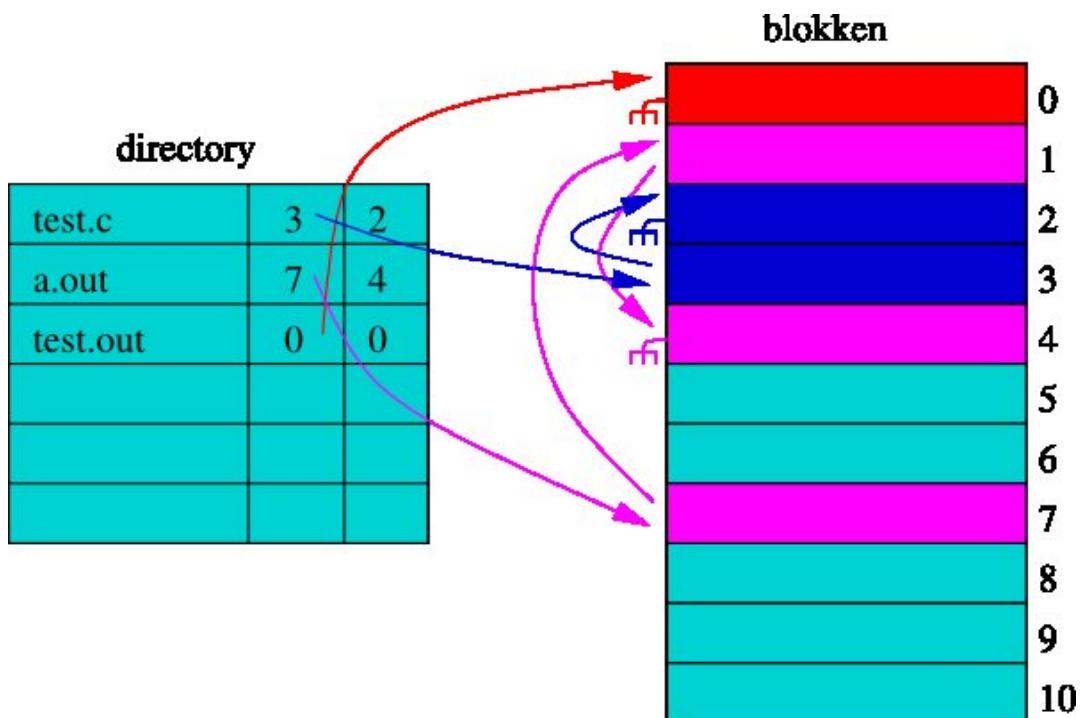
Ze heeft echter ook heel wat nadelen. Vooreerst is er het probleem van de externe fragmentatie. Doordat het schijfgeheugen dat gealloceerd moet worden contigu moet zijn heeft men in werkelijkheid stukken geheugen van ongelijke lengte waarvoor het geheugenbeheer relatief moeilijk is. Compactering kan hier een oplossing brengen, maar meestal zal het bestandssysteem tijdens het compacteren onbruikbaar zijn. Dit is niet steeds aanvaardbaar.

Ten tweede is er het probleem dat men nog vóór de creatie van een bestand moet weten hoe groot het zal worden. Schat men de werkelijke grootte te hoog, dan krijgt men een aanzienlijke interne fragmentatie, schat men het te klein, dan zal het programma dat het bestand aanmaakt afgebroken worden. Dit laatste kan vermeden worden door in dat geval het bestand te kopiëren naar een grotere vrije ruimte en daar de uitvoering verder te zetten. Het hoeft geen betoog dat dit de turn around time van een proces negatief zal beïnvloeden. Langzaam groeiende bestanden zoals log-bestanden zijn vrij moeilijk efficiënt te implementeren.

Sommige systemen laten daarom toe om de contigue allocatie stuksgewijs te ondersteunen. Men begint dan met een bestand met een gegeven grootte, en indien het te klein zou blijken te zijn, kan men een bijkomende uitbreiding toevoegen. De grootte van deze uitbreiding moet ook op voorhand vastgelegd worden. Ofschoon deze oplossing efficiënter is dan het volledig kopiëren van het bestand, heeft ze ook te lijden onder het probleem van de externe fragmentatie.

## Gelinkte allocatie

Deze allocatiemethode lost alle externe-fragmentatieproblemen van de contigue allocatie op. In de plaats van alle blokken contigu op de schijf op te slaan worden de blokken op de schijf met elkaar gelinkt (zie figuur \ref{gelinkte}). Zolang er vrije blokken zijn, kan een bestand blijven groeien. Ofschoon deze methode het probleem van de externe fragmentatie effectief oplost, heeft ze ook haar problemen. Vooreerst wordt directe toegang nagenoeg onmogelijk omdat steeds de lijst van blokken moet afgelopen worden. Ten tweede kan het sequentieel lezen van een bestand zeer traag worden omdat er per nieuw in te lezen blok in principe een verplaatsing van de kop kan nodig zijn. Ten derde zullen de blokken nu een beetje kleiner zijn omdat de link naar het volgende blok ook moet opgenomen worden. Tenslotte is gelinkte allocatie ook niet zeer betrouwbaar omdat van zodra er één blok corrupt wordt de hele schijf onbetrouwbaar wordt.



**Figure 5.22. Gelinkte allocatie**

Om het sequentieel lezen te versnellen kan men bij de allocatie van de blokken er trachten voor te zorgen dat ze toch zoveel mogelijk sequentieel op de schijf staan zodat de bewegingen van de lees/schrijfkop beperkt worden. Bijgevolg kan de contiguïteit en dus de snelheid door regelmatig te compacteren verbeterd worden.

Het derde probleem kan minder erg gemaakt worden door de blokgrootte te vergroten. Gezien er slechts één link per blok opgeslagen wordt, verkleint hierdoor het percentage aan links.

### Allocatietafel

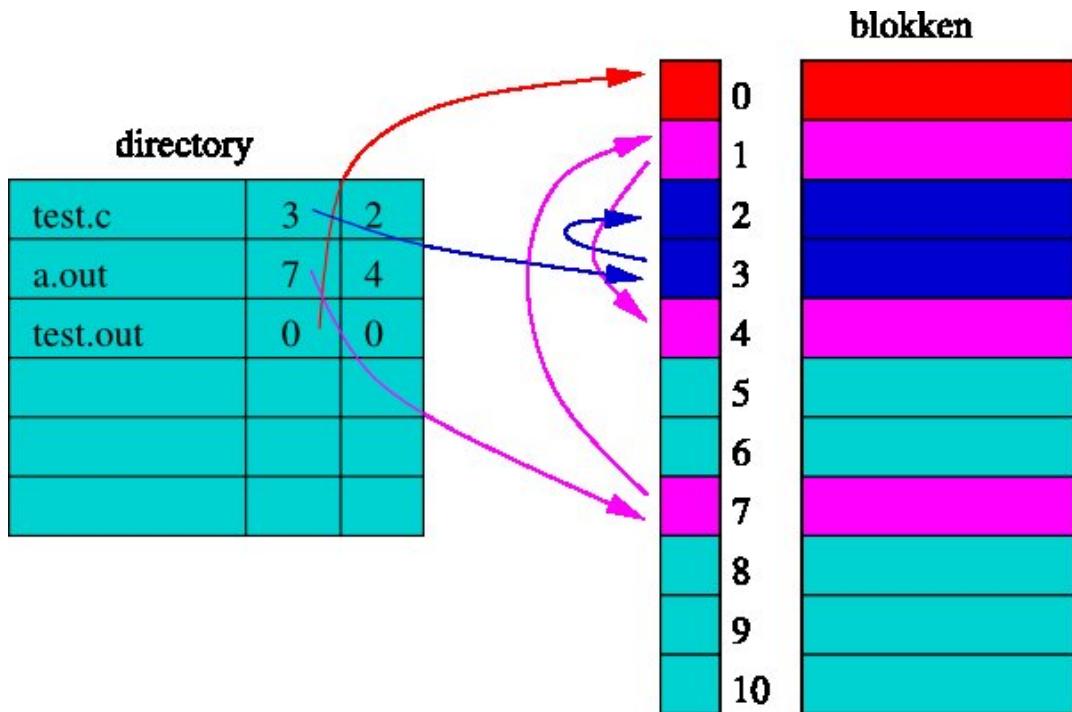
De overige problemen kunnen opgelost worden door alle wijzers naar de blokken bij te houden in afzonderlijke blokken, de zogenaamde FAT of file allocation table.

Deze tabel bevat een wijzer per blok. In de plaats van de wijzers fysiek in de blokken te schrijven worden ze nu in de FAT geschreven.

Dit heeft als voordelen: \* om een bestand te zoeken moet men slechts een paar blokken van de schijf lezen(de FAT) \* er moeten geen wijzers meer in de gegevensblokken opgeslagen worden \* men kan de FAT meer dan eens opslaan op de schijf om beperkte schijfdefecten op te vangen.

Vrije blokken kunnen teruggevonden worden door een speciaal teken op de plaats van de wijzer in de FAT. MS-DOS en OS/2 maken gebruik van een FAT.

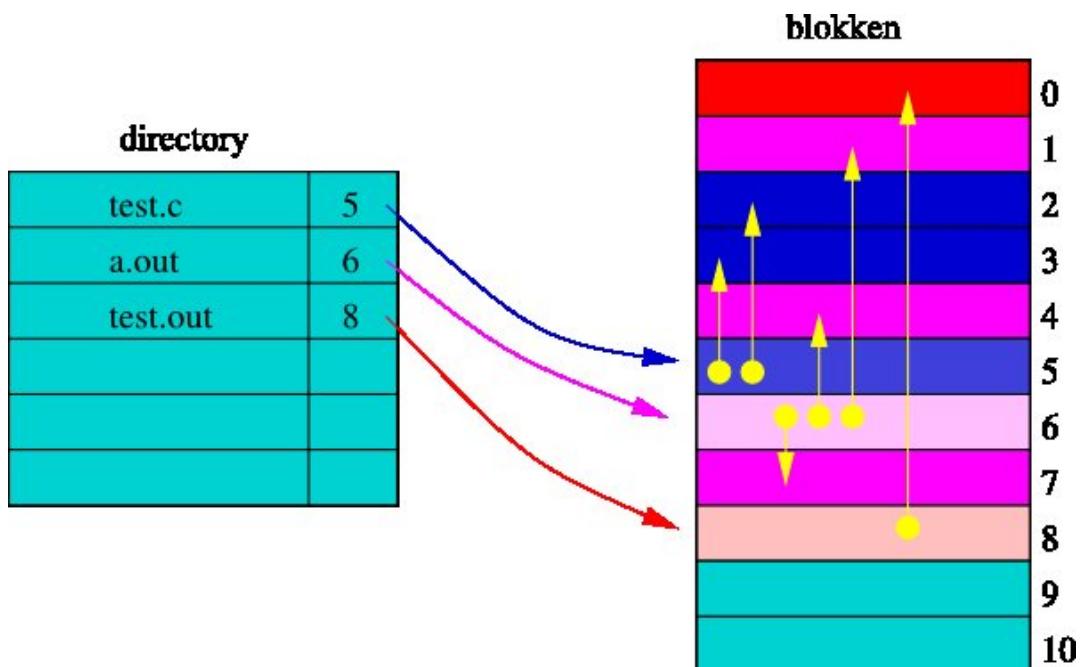
Meestal bewaart men een kopie van de FAT (of delen ervan) in het geheugen om snel een blok op de schijf terug te kunnen vinden. Indien de FAT niet in het geheugen bijgehouden wordt, zal het telkens heen en terug swingen tussen de FAT-blokken en de werkelijke gegevens een aanzienlijke extra belasting van de schijf vormen.



**Figure 5.23. Allocatietabel**

### Indextabel

Gelinkte allocatie of allocatietabellen zijn een afdoende oplossing voor het probleem van de externe fragmentatie, maar laten geen efficiënte directe toegang tot een bestand toe omdat de links sequentieel moeten afgelopen worden. Geïndexeerde allocatie laat dit wel toe. Het idee is dat alle wijzers naar de gealloceerde blokken sequentieel opgenomen worden in een zogenaamd indexblok. Dit indexblok kan dan gebruikt worden om een gegevensblok rechtstreeks terug te vinden.



**Figure 5.24. Geïndexeerde allocatie**

Om niet beperkt te zijn tot bestanden met een te kleine maximale lengte kan men ofwel de indexblokken linken, ofwel verschillende niveaus van indexblokken creëren. Het topniveau verwijst dan naar indexblokken die op hun beurt verwijzen naar de gegevensblokken. Gecombineerde systemen komen ook voor.

Geïndexeerde allocatie maakt doorgaans minder efficiënt gebruik van de schijfruimte dan gelinkte allocatie. De meeste indexblokken zullen immers maar gedeeltelijk gevuld zijn. Bovendien zal men voor kleine bestanden i.p.v. een paar wijzers in de gegevensblokken zelf, een volledige indexblok moeten alloceren.



### Opmerking: optimalisatie

De bovenstaande methodes zijn enkel de basisprincipes voor het beheer van bestanden op een schijf. In de praktijk zijn er veel variaties mogelijk om een snellere werking van het bestandssysteem te verkrijgen. In het bijzonder zal het nuttig blijken om:

- De blokken zo groot mogelijk te nemen, rekening houdend met de interne fragmentatie dat dit met zich mee zal brengen. BSD Unix verandert de grootte van de blokken zelfs naarmate het bestand groter wordt. Dit laat toe om het percentage interne fragmentatie toch nog laag te houden.
- De dynamische informatie in de directories beperkt te houden. Het bijhouden van het tijdstip waarop een bestand voor het laatst

gebruikt werd zal een schrijfoperatie veroorzaken ook indien het bestand enkel gelezen werd.

- Zoveel mogelijk blokken van de schijf in het geheugen te bewaren om meer dan eens lezen te vermijden. Sommige schijfregelaars zullen i.p.v. een sector steeds een volledig spoor inlezen omdat zij ervan uitgaan dat door de lokaliteit meer dan één sector zal gelezen worden. De meeste besturingssystemen hebben ook voorzieningen voor \textit{disk caches}. Solaris en Linux zal al het ongebruikte interne geheugen ter beschikking stellen als disk cache (zowel voor bestands-IO als voor paginering).
- Free-behind - Read-ahead zijn twee methodes die naast LRU gebruikt kunnen worden om de disk cache beter te beheren bij sequentiële toegang. Free-behind zal een blok vrijgeven van zodra een volgend blok ingelezen werd, en read-ahead zal ervoor zorgen dat er steeds 1 blok verder ingelezen wordt in een blokbuffer om de wachttijden bij het effectief opvragen van gegevens uit dat blok te beperken.
- Om de schijfcapaciteit beter te benutten kan men de gegevens gecomprimeerd op de schijf plaatsen. Afhankelijk van het soort van gegevens dat men wil opslaan kan de hoeveelheid die men op deze manier kan opslaan een veelvoud zijn van de originele schijfcapaciteit. Bovendien kan de bandbreedte naar de schijf hierdoor in sommige gevallen vergroot worden omdat er minder gegevensuitwisseling zal zijn met de schijf terwijl het comprimeren en decomprimeren vrij snel kan gaan. Deze compressie is totaal transparant voor de gebruiker.

## Mappen (directories)

Tegenwoordige hebben de meeste bestandssystemen een boomvormige directorystructuur waarbij alle gebruikers voorkomen als deelbomen van de boomstructuur die geassocieerd wordt met het volume. In deze vorm heeft de gebruiker beschikking over subdirectories die op hun beurt bestanden en subdirectories kunnen bevatten. Bestanden hebben nu niet enkel een naam, maar ook een pad, dit is een opeenvolging van de namen van subdirectories. Het pad en de bestandsnaam vormen dan een eenduidige specificatie van een bestand. Doordat alle gebruikers opgenomen zijn in dezelfde boomstructuur is het ook mogelijk om naar bestanden van andere gebruikers te refereren door zowel hun padnaam als hun bestandsnaam op te geven.

Padnamen kunnen zowel absoluut (ten opzichte van de root-directory), als relatief (ten opzichte van de huidige directory) zijn. Absolute padnamen hebben als voordeel dat ze in alle omstandigheden verwijzen naar dezelfde directory, maar ze kunnen hierbij wel aardig lang worden (100 tekens zijn geen uitzondering). Relatieve padnamen hebben als voordeel dat ze korter kunnen zijn, en dat ze in zekere zin 'positie-onafhankelijk' zijn, dit wil zeggen dat ze binnen een bepaalde boomstructuur geldig blijven, ook wanneer de totale boom verplaatst wordt.

Boomvormige directorystructuren hebben echter ook hun nadelen: . men zal het concept van 'huidige directory' moeten invoeren om te weten in welke directory men aan het werken is, . om bestanden terug te vinden kan men in verscheidene directories moeten gaan kijken, en soms wil men hierbij een bepaalde volgorde kiezen (PATH), . er moeten voorzieningen getroffen worden om bepaalde bestanden in meer dan één directory op te nemen zonder deze te moeten kopiëren of telkens opnieuw de volledige padnaam te moeten ingeven.

Om aan dit euvel te verhelpen bestaan er directorystructuren die kunnen voorgesteld worden door een acyclische graaf die zal toelaten dat een bepaald bestand in twee subdirectories voorkomt.

Om dit te realiseren kan men gebruik maken van zogenaamde links of shortcuts. Unix onderscheidt twee soorten links: symbolische links en harde links. Een symbolische link is gewoon een verwijzing naar een ander bestand. Uitwendig ziet een symbolische link eruit als een gewoon bestand, maar het is wel een bestand van een speciaal type, en het bevat geen gegevens, enkel de naam van het bestand waarnaar het wijst. Als het bestand waarnaar gewezen wordt verdwijnt, dan zal de symbolische link blijven bestaan, maar verwijzen naar een niet langer bestaand bestand.

De tweede soort link is de harde link. Deze is te vergelijken met het delen van geheugen door dezelfde frame-adressen op te nemen in twee paginatabellen. In dit geval zullen twee entries in de directorytabel wijzen naar hetzelfde fysieke bestand. Hierbij wordt een teller bijgehouden met het aantal links die verwijzen naar het bestand. Het wissen van een harde link zal enkel het wissen van het bestand tot gevolg hebben indien deze link de laatste link met een bestand was. Dit verklaart meteen waarom het wissen van een bestand in Unix soms ook unlink genoemd wordt. Voorbeelden van harde links zijn de bestanden `.' en `..' in de directories. Zij komen zowel voor in de huidige directory als in de ouderdirectory.

Het gebruik van links zorgt ervoor dat een directorystructuur niet langer een boomstructuur is maar een graaf. Indien er geen lussen voorkomen in de graaf spreekt

men van een acyclische graaf. Een probleem met grafen is dat het doorzoeken van een graaf op zoek naar bepaalde informatie (bv. een bestand) complexer is dan het doorzoeken van een boom omdat men moet vermijden tweemaal hetzelfde deel van een graaf te doorzoeken. Bij acyclische grafen heeft het meermaals doorzoeken van een graaf enkel een effect op de snelheid waarmee een algoritme uitgevoerd wordt. Bij cyclische grafen kan dit aanleiding geven tot oneindige lussen, en dus een niet-terminerend algoritme.

## Swapruimte

Zoals aangewezen wordt in het hoofdstuk over geheugenbeheer is het effect van een paginafout op de uitvoeringssnelheid van een programma dramatisch en moet men dan ook al het mogelijke doen om de interactie met de swapruimte zo efficiënt mogelijk te maken. We hebben gezien dat een virtueel-geheugensysteem gebruik maakt van pagina's en dat deze pagina's in paginering op aanvraag moeten kunnen uitgewisseld worden met de swapruimte.

Deze pagina's zullen direct moeten kunnen geadresseerd worden. Met de technieken die gebruikt worden bij het klassieke bestandsbeheer zal deze directe toegang soms het lezen van meer dan één blok tot gevolg hebben. Bovendien zijn de blokken van het bestandssysteem niet steeds even groot als de pagina's en de frames.

Men zal dan ook meestal verkiezen om de swapruimte onder te brengen in een afzonderlijke partitie met een eigen beheerssysteem waardoor het aantal schijftoegangen absoluut minimaal gehouden wordt bij het pagineren. Het voornaamste nadeel van deze aanpak is dat de swapruimte bij de generatie van het besturingssysteem moet vastgelegd worden (in de vorm van een partitie). Naderhand uitbreiden (als gevolg van het vergroten van het intern geheugen) is moeilijk omdat het het uitbreiden van een schijfpartitie betreft. Anderzijds zal bij een overdimensionering van de swapruimte de ongebruikte ruimte niet gebruikt kunnen worden voor bestandsopslag. Deze oplossing, ofschoon efficiënt, is heel wat minder flexibel dan het gebruik van een gegevensbestand als swapruimte.

Sommige systemen zoals Windows alloceren de swapruimte als een contigu bestand. Het voornaamste voordeel van deze aanpak is de eenvoud en het feit dat de swapruimte gemakkelijk kan uitgebreid worden. Het nadeel van deze aanpak is de traagheid omdat elke toegang tot de swapruimte via het bestandssysteem moet gaan.

De keuze tussen beide is een compromis tussen gewenste snelheid en gebruikersgemak.

## Vrije ruimte

De vrije ruimte op een schijf moet op een efficiënte manier kunnen bijgehouden worden om snel een vrij blok te kunnen terugvinden. Een aantal gebruikelijke methoden wordt hier besproken.

### Bitmap

Er wordt een bitrij bijgehouden waarbij elk bitje een blok voorstelt. Een vrij blok kan dan voorgesteld worden door 1, en een blok dat in gebruik is door een 0. Voor grote schijven kunnen de bitmaps ook een aanzienlijke ruimte in het geheugen gaan innemen. Ze moeten nu en dan op de schijf bewaard worden als beveiliging. Na een systeemcrash zal deze lijst opnieuw moeten opgebouwd worden uitgaande van de bestanden die op de schijf teruggevonden worden.

### Gelinkte lijst

Gezien de vrije blokken toch niet gebruikt worden, kan een deel van elk blok gebruikt worden om ze met elkaar te linken. De kop van de lijst wijst steeds naar een vrij blok indien er een aanwezig is. Nadeel van deze methode is dat indien er bijvoorbeeld 100 blokken moeten gealloceerd worden, er ook 100 schijftoegangen nodig zijn om de gelinkte lijst af te lopen.

### Wijzerblok

Net zoals bij de gelinkte allocatie kunnen ook de vrije blokken in indexblokken opgenomen worden. In dit geval zal men meestal kiezen voor gelinkte indexblokken i.p.v. voor verschillende niveaus. Het voordeel van deze methode is dat men verschillende blokken ineens kan alloceren.

## Fragmentatie

Welke allocatiemethode men ook gebruikt na verloop van tijd zal de vrije ruimte geen aaneengesloten blok meer vormen, maar verspreid geraken over de volledige schijf. Het gevolg van deze versnippering zal zijn dat bestanden opgebouwd zullen worden uit blokken die zich op verschillende plaatsen op de schijf bevinden (behalve bij contigue allocatie). Doordat de kop van de schijf zich verschillende keren zal moeten verplaatsen om een dergelijk bestand te lezen, zal de prestatie van het bestandssysteem sterk dalen. Dit probleem is bekend, en wordt fragmentatie van de schijf genoemd.

Het probleem kan opgelost worden door de bestanden op de schijf te herschikken (defragmentatie en compactering). Hierbij zullen alle bestanden contigu gemaakt worden en wordt ook de vrije ruimte samengebracht. Deze techniek heeft zijn aanhangers en zijn tegenstanders. Aanhangers claimen dat de fragmentatie van de

bestanden weggewerkt wordt hetgeen nuttig is voor de prestatie. Tegenstanders argumenteren dat door de compactie de volgorde van bestanden gewijzigd wordt, en dat dit een negatieve impact kan hebben op de prestatie. Indien men een pakket installeert op een nagenoeg lege schijf, is de kans groot dat alle bestanden van het pakket na elkaar op de schijf zullen komen te staan. Bij het opstarten van het pakket kunnen de bestanden dan één na één ingelezen worden, zonder dat de kop zich veel moet verplaatsen. Na compactering kunnen deze bestanden verspreid geraken over heel de schijf waardoor er tijd verloren wordt bij het springen van het ene bestand naar het andere.

## 5.5. Bestandssystemen

### 5.5.1. inleiding

Een bestandssysteem bestaat uit verschillende lagen die elk van de diensten van de onderliggende lagen gebruik maken.

1. Op het bovenste niveau is er het \textbf{logische bestandssysteem}. Dit is het niveau waarmee de gebruiker in contact komt. Hier wordt er met bestandsnamen, protecties, enz. gewerkt. Dit niveau beheert ook de directories die als speciale gegevensbestanden beschouwd worden.
2. Het logische bestandssysteem maakt gebruik van de **bestandsorganisatie** die de verbinding vormt tussen het logische en het fysieke niveau. Hier wordt aan het beheer van de vrije schijfruimte gedaan, hier worden de bestandsnamen omgezet naar de logische schijfadressen.
3. Het fysieke bestandssysteem krijgt van de bestandsorganisatie de logische schijfadressen binnen en vertaalt deze naar fysieke schijfadressen voor de betreffende schijven.
4. IO-controle: dit zijn de drivers voor de schijven. Deze drivers verbergen alle details van de schijfhardware voor de bovenliggende lagen.

Een bestandssysteem zal de bestanden op een partitie organiseren en zorgen dat het besturingssysteem over voldoende informatie beschikt om elk bestand terug te vinden. Er zijn veel verschillende soorten bestandssystemen, we zullen ons hier beperken tot FAT16 en NTFS. Eigenschappen van andere bestandssystemen zullen in de cursus besturingssystemen nog besproken worden. Elke partitie begint met een Partition Boot Sector (PBS). Dit is dus weer de eerste fysieke sector die bij een bepaalde partitie behoort. De PBS bestaat weer uit twee delen. Helemaal vooraan staat een

spronginstructie naar de eventueel aanwezige boot routine. Deze boot routine zal het besturingssysteem laden (dus op elke partitie waarop een besturingssysteem geïnstalleerd is, zullen een dergelijk programma en spronginstructie terug te vinden zijn). Daarnaast is er het BIOS Parameter Block. Dit gedeelte bevat een aantal parameters die belangrijk zijn om toegang te krijgen tot het bestandsysteem. Welke parameters hier terug te vinden zijn, hangt natuurlijk af van het type bestandssysteem.

### 5.5.2. een bestandssysteem 'mounten'

Het secundair geheugen van een computersysteem kan uit verscheidene schijven bestaan. Deze schijven kunnen op verschillende manieren aan de gebruiker aangeboden worden. Ofwel krijgen ze een naam zoals in Windows: A: B: C: enz. waardoor ze expliciet zichtbaar blijven, ofwel worden ze opgenomen als subdirectory in een bestandssysteem waardoor ze transparant worden voor de gebruiker. (zoals bij unix/linux het geval is)

Men kan de directories van een bestandssysteem verdelen over de schijven, of met andere woorden, een schijf met haar eigen bestandssysteem (of systemen) wordt als subdirectory van het bestandssysteem van een andere schijf beschouwd. Dit laat toe om bestandssystemen te creëren die groter zijn dan één fysiek volume. Indien een schijf niet fysiek aanwezig is (bij DVD-stations bijvoorbeeld), zal deze subdirectory gewoon leeg zijn; indien de schijf aanwezig is, zal men in deze directory gegevens kunnen opvragen en bewaren. Eenmaal geconfigureerd is dit totaal transparant voor de gebruiker.

### 5.5.3. types: journaling file systems

Bestandssystemen zijn meestal grote datastructuren, veranderingen eraan veroorzaken meestal meerdere schrijfoperaties aan bestanden en directories. Dit introduceert een race-conditie, waarbij een onderbreking (zoals een stroomstoring of systeemcrash) kan leiden tot een inconsistente toestand.

Bijvoorbeeld, het wissen van een bestand op een UNIX systeem veroorzaakt twee stappen:

1. het verwijderen van de directory-ingang
2. item het aanduiden van de bestandsinode als vrij ruimte

Als stap 1 uitgevoerd wordt net voor een crash dan zal er een inode als wees achterblijven.

Omgekeerd als stap 2 als eerste voor de crach uitgevoerd wordt dan wordt de inode als vrije ruimte aangeduid en kan overgeschreven worden.

Een manier tot herstel is om de complete datastructuren van het bestandssysteem te doorlopen bij een volgende aankoppeling zodat onvolkomenheden kunnen gedetecteerd worden. Dit kan zeer traag verlopen voor grote bestandssystemen.

Een andere manier tot herstel is om een logbestand (journaal) bij te houden met alle veranderingen die moeten gemaakt worden.

Herstellen kan dan eenvoudig door het logbestand te doornemen en alle onvolkomenheden te herstellen.

Enkele voorbeelden van journalizing bestandssystemen zijn JFS, EXT3/4, ReiserFS en NTFS

#### 5.5.4. Case study 1: FAT file system

Het FAT bestandssysteem komt voor in een aantal varianten, waarvan FAT16 (1986) en FAT32 (1996) ontegensprekelijk de bekendste zijn. Als primair bestandssysteem hebben ze allebei reeds lang afgedaan om redenen die later aan bod komen, maar ze vormen wel een ideale educatieve instap in de wereld van bestandssystemen. Recentere bestandssystemen zijn een stuk complexer, zoals we later zullen leren. FAT wordt tegenwoordig vanwege zijn eenvoud wel nog gebruikt op USB dongles, flash kaartjes etc.

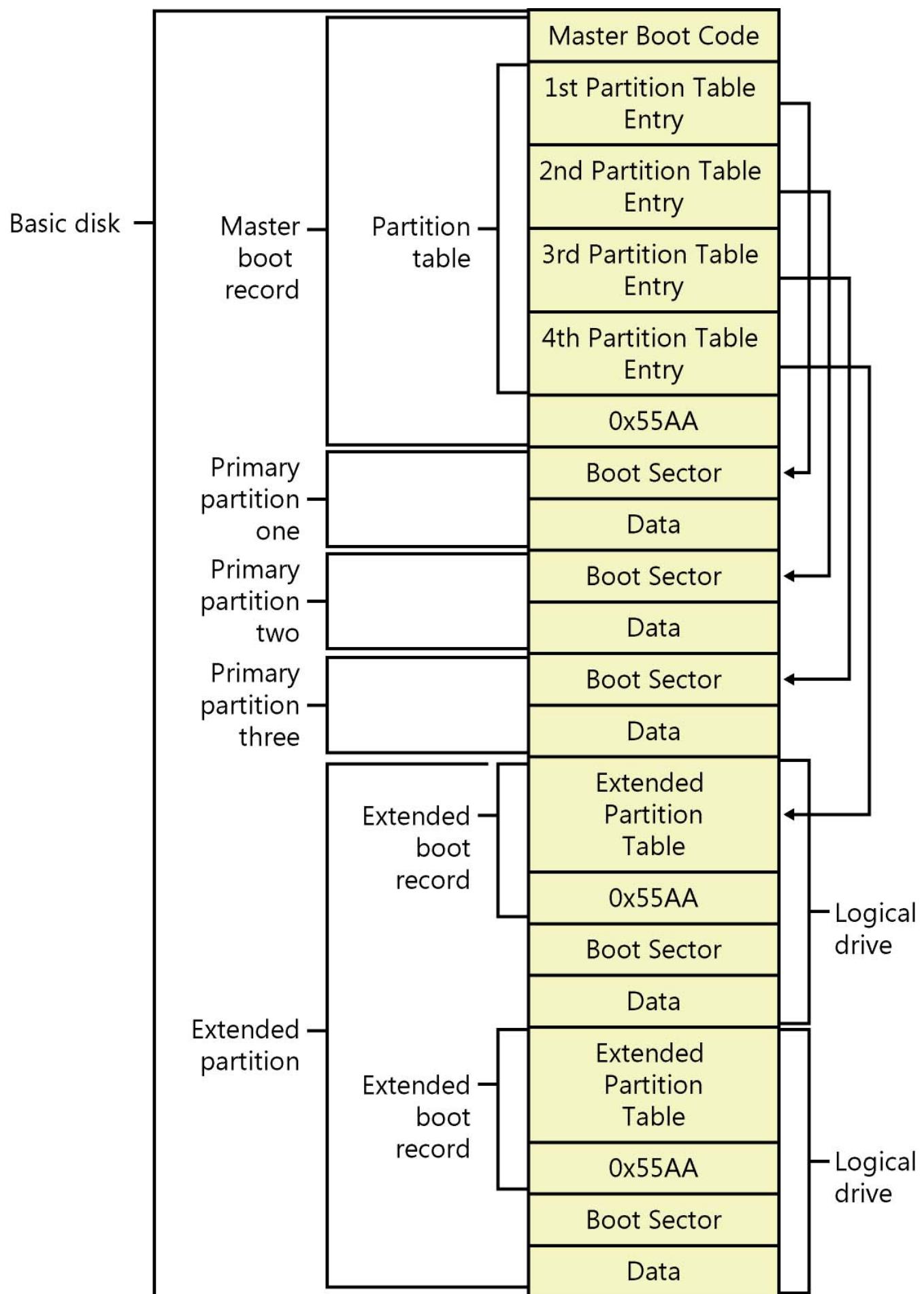


Figure 5.25. FAT organisatie

In bovenstaande afbeelding zie je de opbouw van een FAT16 partitie. Vooraan bevindt zich, zoals altijd, het PBS met de nodige parameters. De PBS is een van de gereserveerde sectoren vooraan de partitie. Onmiddellijk na de gereserveerde sectoren volgen een aantal File Allocation Tables (FATs). In de figuur is dit aantal gelijk aan twee, het meest voorkomende geval.

Na de FATs komt de root folder, gevuld door de data clusters. In deze data clusters worden uiteraard de bits opgeslagen die de bestanden vormen. Belangrijke opmerking is dat deze bits georganiseerd worden in clusters. Een cluster is een reeks sectoren die steeds bij een welbepaald bestand horen.

Bijvoorbeeld: indien een cluster bestaat uit 16kB (32 sectoren), dan zal een bestand van 1kB 16kB schijfruimte bezetten (een sector).

Een bestand van 20kB zal verspreid worden over twee clusters, deze clusters moeten geen opeenvolgende clusters zijn (fragmentatie).

Soms kan je de grootte van de clusters kiezen (bij het formatteren, afhankelijk van het bestandsysteem). Grote clusters betekent dat bestanden minder snel gefragmenteerd zullen worden, maar betekent anderzijds dat meer ruimte verloren zal gaan (door onvolledig gevulde clusters).

**Table 5.3. PBS van een FAT32 partitie**

Offset	beschrijving	grootte
00h	Jump Code + NOP	3 Bytes
03h	OEM Name (Probably MSWIN4.1)	8 Bytes
0Bh	Bytes Per Sector	1 Word
0Dh	Sectors Per Cluster	1 Byte
0Eh	Reserved Sectors	1 Word
10h	Number of Copies of FAT	1 Byte
11h	Maximum Root DirectoryEntries (N/A for FAT32)	1 Word
13h	Number of Sectors inPartition Smaller than 32MB (N/A for FAT32)	1 Word
15h	Media Descriptor (F8h forHard Disks)	1 Byte

<b>Offset</b>	<b>beschrijving</b>	<b>grootte</b>
16h	Sectors Per FAT in Older FATSystems (N/A for FAT32)	1 Word
18h	Sectors Per Track	1 Word
1Ah	Number of Heads	1 Word
1Ch	Number of Hidden Sectors inPartition	1 Double Word
20h	Number of Sectors inPartition	1 Double Word
24h	Number of Sectors Per FAT	1 Double Word
28h	Flags	1 Word
2Ah	Version of FAT32 Drive (HighByte = Major Version, Low Byte = Minor Version)	1 Word
2Ch	Cluster Number of the Startof the Root Directory	1 Double Word
30h	Sector Number of the FileSystem Information Sector (See Structure Below)(Referenced from the Start of the Partition)	1 Word
32h	Sector Number of the BackupBoot Sector (Referenced from the Start of the Partition)	1 Word
34h	Reserved	12 Bytes
40h	Logical Drive Number ofPartition	1 Byte
41h	Unused (Could be High Byteof Previous Entry)	1 Byte
42h	Extended Signature (29h)	1 Byte

Offset	beschrijving	grootte
43h	Serial Number of Partition	1 Double Word
47h	Volume Name of Partition	11 Bytes
52h	FAT Name (FAT32)	8 Bytes
5Ah	Executable Code	420 Bytes
1FEh	Boot Record Signature (55hAAh)	2 Bytes

Hierboven zie je de inhoud van de partition boot sector van een FAT32 partitie. We zullen niet alle parameters aanhalen, maar wel aantonen hoe het besturingssysteem een welbepaald bestand kan terugvinden en hoe het daarbij gebruik maakt van de opgeslagen informatie.

FAT32 heeft de volgende voordelen tegenover FAT16:

1. FAT32 kan met grotere harde schijven (max. 2 Terabyte) werken dan FAT16 (max. 2 GB).
2. In FAT16 zijn er maar 512 entries mogelijk in de hoofddirectory. Met lange bestandsnamen die meerdere entries innemen kan dit al snel problemen geven. Daarom wordt de hoofddirectory in FAT32 opgeslagen zoals een gewone directory, zodat er een onbeperkt aantal entries in kunnen.
3. FAT32 gebruikt kleinere clusters, zodat er minder slack is. Dit is vooral van belang voor grotere schijven. Op een schijf van 2 GB kan je bijvoorbeeld gemakkelijk 200 tot 300 MB winnen met FAT32 (dat is 10 tot 15%).
4. De beide FAT-tabellen staan niet meer in het begin van de schijf, zodat ze minder snel beschadigd worden.

In onderstaande afbeelding zie je de PBS van een SD-kaartje dat uit een Android smartphone komt. De sector werd hexadecimaal voorgesteld met een hex-editor. Probeer enkele van de eerder aangehaalde parameters terug te vinden. (de entries zijn gelijkaardig aan de entries voor een FAT16 partitie)

**hexadecimale voorstelling van de PBS van een FAT16 SD-kaart** De root folder bevat al de informatie over bestanden en mappen die opgeslagen zijn op de root folder. Als we ons beperken tot namen die voldoen aan het 8.3 formaat, dan is elk element in de root folder 32 bytes groot. Een belangrijk deel hiervan zijn de eerste elf bytes, die de karakters van de naam bevatten.

Indien een bestand niet in de root folder zit, maar is ondergebracht in een andere map, kan vanuit de root folder het volledige pad gevuld worden tot we uiteindelijk in de juiste

map terecht komen. Daar staat de informatie dan op gelijkaardige manier opgeslagen als in de root folder.

### **Example 5.2. Voorbeeld: zoeken van een bestand in de op een FAT-partitie.**

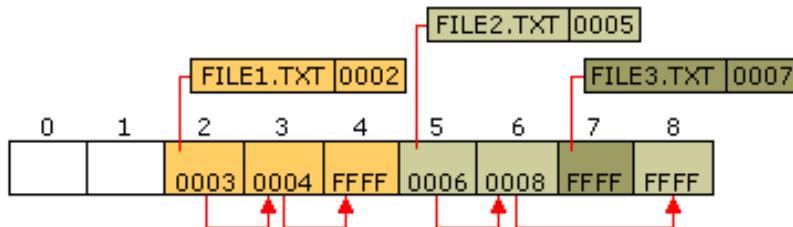
1. De eerste stap is het opzoeken van de root folder. Uit het partition boot sector kan je de structuur van de partitie afleiden:
  - het aantal gereserveerde sectoren
  - het aantal FAT's
  - het aantal sectoren per fat
  - aantal bytes per sector
2. Op basis hiervan kan het besturingssysteem bepalen op welk sectornummer de root folder start ( $\text{start root} = \text{aantal gereserveerde sectoren} + \text{aantal FAT's} * \text{aantal sectoren per FAT}$ ). Merk op dat in het PBS ook het aantal bytes per sector is opgegeven, zodat je ook het byte-adres kan berekenen ( $\text{sectornummer} * \text{aantal bytes per sector}$ ).
3. Zodra het adres van de root folder gevonden is kan het besturingssysteem op zoek naar de naam van het gevraagde bestand.
4. Als dan het juiste element in de root folder gevonden is (op basis van bestandsnaam), kan de overige informatie gebruikt worden om het bestand terug te vinden op de partitie. Vooral de laatste 4 bytes zijn belangrijk. Die bevatten namelijk de lengte van het bestand (in bytes) en de 16 bits daarvoor bepalen het eerste clusternummer waar er data terug te vinden is.
5. Op basis van het clusternummer moet het besturingssysteem gaan kijken in de FAT tabel om zo te weten te komen als het bestand bestaat uit één dan wel uit meerdere clusters.
  - De verschillende FAT-tabellen zijn in principe (tenzij er fouten optreden) kopieën van elkaar, dus maakt het in principe niet uit in welke tabel gekeken wordt.
  - Bij FAT16 bestaat elk element in de FAT tabel uit 16 bits. Elk element is gelinkt met een welbepaald clusternummer (element 2 hangt samen met cluster 2, element 3 hangt samen met cluster 3, ...).
  - De inhoud van de twee bytes bepalen hoe een cluster samenhangt met de andere:

```

# 0000h: Available Cluster
# 0002h-FFEFh used, Next Cluster in File
# FFF0h-FFF6h reserved Cluster
# FFF7h BAD Cluster
# FFF8h-FFFF Used, Last Cluster in File

```

- Op basis van het eerste clusternummer kan het besturingssysteem alle clusters en de volgorde waarin ze een bestand vormen terug vinden. Het komt er dan op aan om alle clusters volledig uit te lezen, behalve het laatste cluster, waarvan enkel voldoende bytes gelezen moeten worden om in totaal aan de in de root folder terug te vinden grootte te voldoen.



**Figure 5.26. voorbeeld FAT-tabel met fragmentatie**

### 5.5.5. Case study2: NTFS

NTFS is een bestandssysteem dat standaard gebruikt wordt op alle recente Windows versies. Het is een gesloten systeem, ontstaan door een doorontwikkeling van HPFS, een samenwerking tussen IBM en Microsoft.

NTFS werkt op een andere manier dan FAT16 en zal dus ook andere informatie opslaan in de PBS. NTFS maakt bijvoorbeeld geen gebruik van FATs, dus zal er ook geen aantal FATs of grootte van de FAT opgegeven worden in de PBS. NTFS maakt gebruik van een Master File Table (MFT), waarin alle relevante informatie over een bestand wordt opgeslagen. [3]

Deze informatie is een stuk uitgebreider dan in de root folder van een FAT systeem, het is zelfs mogelijk dat voor een klein bestand de data volledig in de MFT wordt opgeslagen. Is dit niet het geval, dan is voor het opzoeken van het bestand de meest relevante informatie die in het MFT aanwezig is, de naam van het bestand en de runlist. Bij het opzoeken van een bestand is de eerste stap dus het terugvinden van de MFT. Hiervoor staat in het PBS het cluster nummer opgegeven waar het MFT start. Aangezien in het PBS ook het aantal sectoren per cluster terug te vinden zijn, kan

weer het sector adres van de MFT teruggevonden worden. Binnen de MFT zal gezocht moeten worden op de naam van het gevraagde bestand. Als het betreffende record dan wordt teruggevonden, kan binnen dit record de runlist worden opgezocht. De runlist is onderdeel van het \$DATA attribuut (start van dit attribuut wordt gekenmerkt door 80H), waarbinnen ook de ingenomen en werkelijke grootte van het bestand terug te vinden is. De runlist bestaat uit een aantal opeenvolgende bytes:

### opbouw van een runlist

1. Het eerste byte wordt opgesplitst in twee nibbles.
  - Het meest significante nibble duidt aan hoeveel bytes gebruikt worden voor de offset (stel K)
  - Het minst significante nibble geeft aan hoeveel bytes gebruikt worden voor de lengte (stel N)
2. De volgende N bytes geven in little endian notatie aan hoeveel clusters na elkaar in gebruik zijn voor deze stream.
3. De volgende K bytes geven aan op welke clusteroffset de stream begint (weer in little endian notatie).

Indien het volgende byte 0x00 is, eindigt de runlist hier, anders moet je vanaf hier de bytes weer op dezelfde manier interpreteren.

#### Example 5.3. interpretatie van een runlist

runlist = 31 0A 21 23 05 00 34 ...

Het eerste byte geeft aan dat het eerstvolgende het aantal clusters bepaalt en de drie daarop volgende een offset. Daarna volgt er 00, dus verder geen data. Dus de data staat in 10 clusters te beginnen vanaf cluster 0x052321. Op deze manier is het mogelijk om verschillende reeksen clusters in de juiste volgorde te gaan lezen.

Van de laatste cluster moeten weer net genoeg bytes gelezen worden om evenveel bytes te lezen als opgegeven bij de werkelijke grootte van het bestand.

### 5.5.6. case study 3: Ext4

EXT staat voor Extended file system. Dit bestandssysteem bouwt verder op standaard linux file systems. Het wordt gezien als een brug tussen Ext3 en de meer geavanceerde

bestandssystemen, en is tegenwoordig zowat op alle linux distributies standaard. (al lijkt het rijk van EXT bedreigd: bijvoorbeeld op RHEL is tegenwoordig XFS de standaard bestandsindeling. Ook op Android is EXT sinds versie 2.3 het default bestandssysteem.

## Inodes

In Unix maakt men gebruik van een gecombineerd systeem met inodes.

Inodes komen voor in twee types:

### Directory inodes

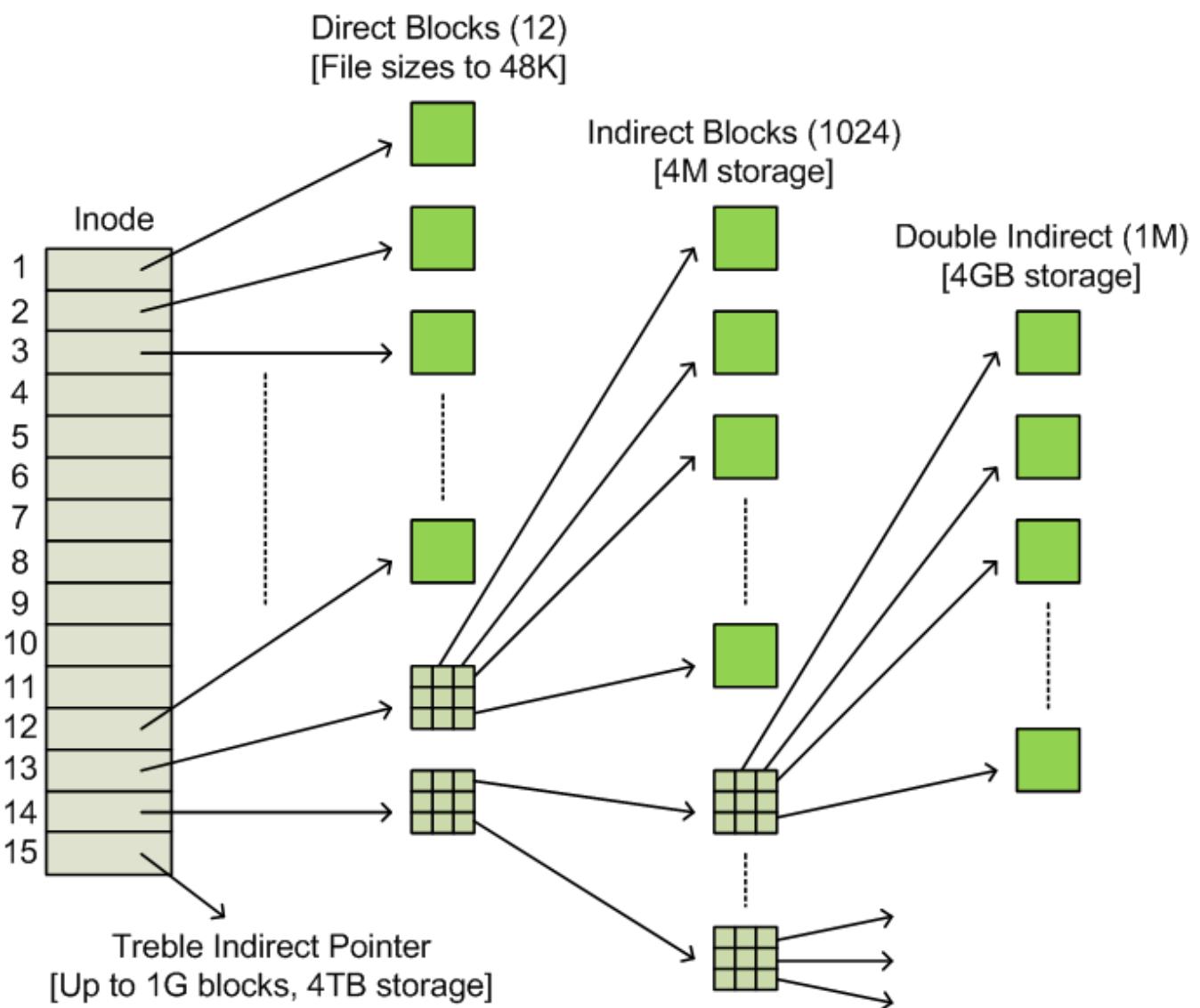
Deze bevatten de info van een bepaalde map, met onder meer de subdirectories en bestanden. Merk op dat deze inodes ook de bestandsnaam bevatten.

### Data file inodes

De data file inodes bevatten de informatie over waar een bepaald bestand kan gevonden worden op de schijf. Merk op dat deze inodes de bestandsnaam NIET bevatten. Door deze werkwijze kan je in principe twee verschillende bestandsnamen gebruiken voor één fysiek bestand. In Linux heet zo iets ‘hard-linking’. Zo’n inode voor een bestand bevat Directe, single indirecte, double indirecte en zelfs triple indirecte verwijzingen naar bestandslocaties. Deze constructie wordt snel duidelijker met een afbeelding:

Hierbij wordt een vast aantal indices opgenomen als attributen van een bestand in de zogenaamde inode (dit is een gegevensstructuur die zich tussen de directory-entry en het fysiek bestand bevindt. Per fysiek bestand is er slechts 1 inode, maar kunnen er verschillende directory-entries zijn. De inode houdt o.m. bij hoeveel harde links er naar een bestand bestaan). Naast dit beperkt aantal directe indices (bv. 12) die zullen volstaan voor de bestanden kleiner dan bv. 48 Kb, is er nog een wijzer naar een indexblok van niveau 1, en ook nog een wijzer naar een indexblok van niveau 2 (en zelfs naar niveau 3).

Deze oplossing laat toe om snel directe toegang te hebben tot kleine bestanden, redelijk snel tot de middelgrote bestanden, en aanvaardbaar voor de echt grote bestanden waarbij de grootte van de bestanden niet beperkt wordt door de implementatie van het bestandssysteem, maar door de grootte van de schijf. Dit is prima omdat in de praktijk toch blijkt dat minder dan 5% van alle bestanden groter zijn dan 48KB en dus een indexblok nodig hebben. Alle andere kunnen het stellen zonder afzonderlijk indexblok.



**Figure 5.27. Inode structuur (bron onbekend)**

### 5.5.7. Andere bestandssystemen

Met de opgesomde bestandssystemen is de lijst uiteraard niet compleet. Een niet limitatief overzichtje met de meest opvallende features. Deze zijn vaak niet toegankelijk met Windows, omdat Microsoft hiervoor geen drivers ontwikkelt.

#### BTFRS

BTFRS wordt algemeen beschouwd als de opvolger van EXT4 in Linux. Het bestandssysteem kan reeds gebruikt worden, maar wordt niet nog stabiel genoeg geacht voor productieomgevingen. Bij elke nieuwe versie van de Linuxkernel worden nieuwe features toegevoegd en bugs geplet.

Features:

- Ondersteuning voor snapshots
- Ondersteuning voor quotas
- ...

## ZFS

ZFS werd oorspronkelijk ontwikkeld door SUN, en heeft een erg rijke featureset, die Btrfs en EXT4 ruim achter zich laat. Door conflicterende licenties is het niet standaard te vinden in linux-distributies, maar het kan meestal wel achteraf geconfigureerd worden.

## Re-Fs

Dit nieuwe bestandssysteem is momenteel enkel ondersteund door Windows server 2012, en ervan booten is nog niet ondersteund. Het bevat enkele features die vooral bij kritieke data belangrijk zijn.

## F2FS

Dit nieuwe bestandssysteem is speciaal opgebouwd om een hoog rendement te halen op flash disks.

### 5.5.8. Bestanden wissen

Bestanden verwijderen van een harde schijf is soms minder eenvoudig dan het lijkt. Indien de bits herschreven worden, zijn er technologisch nog mogelijkheden die toelaten om na te gaan wat vorige waarden waren. Hier gaan we niet op die technieken in. Het is ook interessant om te weten wat er op een bestandsysteem gebeurt wanneer een bestand gewist wordt. Aan de snelheid waarmee een grote hoeveelheid of grote bestanden gewist worden kan je al merken dat niet elk byte op de schijf gewist wordt. In plaats daarvan zal de verwijzing in het bestandsysteem worden aangepast.

In het geval van FAT16 wordt een speciaal karakter gebruikt om de naam in de root folder mee te beginnen en worden alle clusters die gebruikt werden, in de FAT tabel als ongebruikt gemarkerd.

Het is in dit geval nog mogelijk om het bestand terug te vinden, op voorwaarde dat het niet gefragmenteerd was. In geval van fragmentatie is de (gewiste) FAT tabel nodig om de juiste clusters en hun volgorde terug te vinden.

Bij NTFS wordt enkel het MFT record aangepast, waarin een parameter zal aangeven dat het bestand gewist is. Aangezien dit anders gemarkerde, maar niet gewiste MFT

record nog steeds alle informatie bevat (via de runlist) is het nu mogelijk om ook gefragmenteerde gewiste bestanden terug te vinden. Belangrijke voorwaarde is wel dat de vrijgekomen ruimte nog niet beschreven mag zijn voor een ander bestand (en uiteraard mag het MFT record ook nog niet overschreven zijn).

Bedenk zelf of data nog terug te vinden is als je een schijf formatteert. Bij het formatteren worden de nodige controlestructuren (PBS, FATs/MFT, ...) aangebracht en worden alle preambules van de sectoren vastgelegd.

## 5.6. Bronvermelding bij dit hoofdstuk

- [MANGAN] 'Advanced format 4K disk drives and performance'. Tim Mangan, <http://www.brianmadden.com/blogs/timmangan/archive/2010/08/16/advanced-format-4k-disk-drives-and-performance.aspx>
- [PCPART] 'storage trends'. <https://pcpartpicker.com/trends/internal-hard-drive/> . accessed 27/08/2014

---

# Chapter 6. Opstartroutine

## 6.1. Geheugens voor opstartROUTINGe

Een processor kan bij het opstarten enkel spreken met geheugen. Om een besturingssysteem te laden op een harde schijf is dus een tussenstap nodig. Dit geheugen mag uiteraard niet volatiel zijn.

Er zijn verschillende soorten ROM geheugens die momenteel nog relevant zijn:

### MROM

Masked-programmed ROM is echt read-only geheugen. Het krijgt zijn inhoud tijdens de fabricage en deze inhoud kan nadien niet meer veranderd worden door een gebruiker.

### PROM

Programmable ROM is eigenlijk ook read-only. De inhoud kan met behulp van een speciaal programmeertoestel eenmaal vastgelegd worden. Nadien kan de inhoud niet meer gewijzigd worden.

### EEPROM

(*Electrically Erasable PROM*). Ook met dit type geheugen kan de inhoud gewist en vervolgens aangepast worden. Het verschil met EPROM is dat nu geen UV licht nodig is, maar dat elektrische signalen volstaan voor het wissen. Dit kan dus met hetzelfde toestel als waarmee geprogrammeerd wordt. Het nadeel van EEPROM is de kostprijs. Voor dezelfde densiteit is het een pak duurder dan EPROM.

### Flash

is net zoals EEPROM te wissen en opnieuw te beschrijven met elektrische signalen. Bovendien is voor flash geen speciaal programmeertoestel nodig. Aan de andere kant haalt het met EPROM vergelijkbare densiteiten. Bovendien is geen speciale verpakking met glas nodig.

Het nadeel van flash is dat het minder dikwijs herschreven kan worden dan EEPROM. Voor toepassingen waar regelmatig de data moet worden aangepast, bestaat ondertussen high-endurance flash. Gewoon flash geheugen kan tussen 300.000 en 500.000 keer herschreven worden. De high-endurance variant een paar miljoen keer. Extra probleem hierbij is dat geen individuele cellen aangepast kunnen worden, maar dat steeds ganse blokken in één keer geschreven moeten worden. Hierdoor kan deze beperking nog zwaarder doorwegen (cel wordt meer beschreven dan dat ze wordt veranderd).

De BIOS routines worden nu meestal opgeslagen op flash geheugens, wat toelaat om de routines te upgraden. Bij dergelijke upgrades is het natuurlijk cruciaal dat er niets mis gaat, want dan zal de computer niet meer in staat zijn om op te starten. Sommige moederbord fabrikanten voorzien in een procedure om terug te keren naar een vorige versie wanneer een flash operatie mislukt. Als een dergelijke procedure niet vorhanden is, kan je enkel nog hopen dat de flash-IC niet vast gesoldeerd is, zodat je hem met een andere programmer kan programmeren.

## 6.2. Opstartroutine met BIOS

Bij het aanschakelen van de spanning wordt een reeks stappen doorlopen. De volgorde is afhankelijk van de fabrikant van de hardware (de hoofdkaart, BIOS, CMOS en andere componenten). Onderstaande stappen worden steeds uitgevoerd.

### 6.2.1. Systeemstart

Eerst start de interne voeding op. Het duurt even voor alle lijnen een stabiele spanning aangeven. Eens dit het geval is geeft de voeding een power good-signal. De chipset, die gevoed werd door de 5V standby lijn, zal het reset-signaal pas wegnemen op het ogenblik dat het power good signaal komt. Zodra het reset signaal wegvalt, zal de computer beginnen opstarten. De processor begint op adres FFFF0h, waar de uitvoering van de BIOS opstartroutine begint.

### 6.2.2. Power On Self Test (POST)

De opstartroutine uit de BIOS voert de Power-on-self-test (POST) uit. Indien er fatale fouten ontdekt worden, geeft het een foutmelding, indien om de een of andere reden de grafische kaart niet bruikbaar is enkele BEEP-signalen, en stopt het startproces. Dit kan bijvoorbeeld het geval zijn indien er een probleem met de grafische kaart ontdekt wordt (de kaart is defect of is niet goed ingeplugged) of wanneer de kabel voor de harde schijf verkeerd aangesloten werd.

### 6.2.3. Zoeken naar BIOS-uitbreidingen

De systeem-BIOS is niet het enige BIOS in een computer. Grafische kaarten, harde schijven, storage-adapters en andere hebben ook hun eigen BIOS. Het systeem-BIOS gaat daarom onmiddellijk op zoek naar de grafische kaart en laat het ingebouwde video-BIOS programma uitvoeren. Hierdoor verschijnt er informatie over de kaart op het scherm. Deze informatie is doorgaans slechts heel even op het scherm.

Daarna voert de computer enkele tests uit op het systeem (vervolg POST): onder andere het geheugen wordt onderzocht en de geheugenplaatsen geteld. Het BIOS zal bij een defect een foutmelding geven.

Vervolgens begint het de startroutine aan een inventarisatie waarbij de meeste klassieke randapparatuur (printerpoorten, seriële poorten, floppy- en harddisk controllers) gedetecteerd zal worden.

Een recente BIOS zal zelfs automatisch het soort geheugen herkennen en de juiste parameters voor de harde schijf instellen.

Meestal wordt op het scherm getoond welke apparaten, harde schijf, DVD-ROM, ... gevonden en herkend worden.

### Oefening

---

Zoek voor het moederbord van jouw pc op welke auditieve foutcodes van toepassing zijn.

---

## 6.3. Plug and play configuratie

Indien een plug and play ondersteunend bussysteem (zoals PCI) aanwezig is, zal een plug and play configuratie fase doorlopen worden: Er wordt gezocht naar P&P-devices op alle aanwezige I/O bussen (PCI, PCI Xpress, AGP). Dit wordt de enumeration genoemd. Voor elk gevonden device wordt gelezen welke resources (I/O-adressen, Interrupts, DMA-kanalen, geheugenruimte) er nodig zijn. De lijst met de vorige configuratie (inclusief de vroegere toekenning van de resources) wordt uit het ESCD geladen en vergeleken met de nieuwe informatie. Het ESCD (Extended System Configuration Data) is een deel van het CMOS-RAM geheugen. Hierin wordt de P&P-configuratie opgeslagen.

Bij het opstarten controleert het BIOS deze gegevens op wijzigingen sinds de laatste start om te weten of bepaalde instellingen moeten aangepast worden. Indien er geen verandering is, gaat het opstartproces gewoon verder.

In het andere geval begint er een herconfiguratie.

De BIOS-setup informatie wordt geraadpleegd voor eventuele gereserveerde systeembronnen door niet Plug and Play-kaarten. Overblijvende systeembronnen worden toegekend aan de PnP-kaarten. Het ESCD wordt bijgewerkt met de mededeling "Updating ESCD... successfull".

Het bootproces gaat verder. Wanneer het besturingssysteem opgestart is, heeft dat tenslotte nog de mogelijkheid om de Plug and Play-configuratie te controleren en eventueel te wijzigen.

## 6.4. Extensible Firmware Interface

### 6.4.1. BIOS op pensioen

Het systeem van de BIOS is één van de laatste onderdelen die nog niet substantieel veranderd zijn sinds de introductie van de PC lang geleden. Hoewel er geprobeerd wordt om de technische details wat weg te stoppen voor de eindgebruiker, is het toch nog steeds een erg gebruiksvriendelijke omgeving. Het gebrek aan ondersteuning van bijvoorbeeld muizen is daar een goed voorbeeld van.

Aan de binnenkant van het BIOS-raderwerk was de situatie zo mogelijk nog een pak erger. Het programmeren ervan gebeurde voor een stuk in machinecode, en het toevoegen van functionaliteit was grotendeels patchwork. Slechts erg weinig mensen hebben de nodige ervaring om hieraan te programmeren. Sommige zaken zijn zelfs niet op te lossen. Een voorbeeld hiervan is de maximum grootte van de opstartpartitie, die is 2TB. Over enkele jaren zal dat ongetwijfeld te weinig zijn.

Daarom zijn er in de loop der tijd enkele pogingen geweest om dit systeem te vervangen. Weinigen daarvan hadden veel succes. Veel had te maken met de licentie waaronder moest gecodeerd worden. Bedrijven waren immers niet bereid om de code van hun drivers vrij te geven.

Ook Intel deed een poging om de BIOS overbodig te maken door een ander systeem. Om het project voldoende momentum te bezorgen, werd een organisatie opgebouwd rond deze technologie, met als doel ze te standaardiseren. Deze technologieën zijn verzameld rond de naar EFI (Extensible Firmware Interface).

Deze technologie verschilt fundamenteel van de verouderde BIOS.

- Gecodeerd in een breed gekende taal (C++)
- Volledig modulair
- Compatibel met de oude BIOS om de overgang mogelijk te maken.
- Drivers kunnen opgenomen worden in deze EFI
- De 2TB-grens voor de opstartpartitie is hier niet bestaande

Door dit vernieuwde opzet zullen ongetwijfeld een pak nieuwe mogelijkheden toegevoegd worden aan de PC zoals we hem nu kennen. EFI laat zich nog het

best vergelijken met een micro-besturingssysteem. Enkele voorbeelden van nieuwe toepassingen:

- Omdat drivers kunnen geïntegreerd worden, kan het veel makkelijker worden om een nieuw besturingssysteem te installeren. Rondzeulen met driverdisks voor eenvoudige devices zou dus moeten verleden tijd worden.
- EFI kan makkelijk uitgebreid worden met kleine toepassingen. Denk bijvoorbeeld aan een mediaplayer. Zo zou je kunnen je laptop gebruiken als mediaspeler, zelfs zonder dat die ‘echt’ opgestart is.
- EFI zou kunnen voorzien worden van een virtualisatielaag zodat je meerdere besturingssystemen tegelijkertijd kan gebruiken.
- Hoewel de mogelijkheden nagenoeg eindeloos zijn, is te verwachten dat de eerste gebruikte EFI's niet veel meer functionaliteit hebben dan de vroegere BIOS'en.



### 6.4.2. (U)EFI verspreiding

Aan EFI wordt sedert 2002 gewerkt door enkele grote firma's. Nagenoeg elke grote hardware bouwer heeft zich bij de organisatie aangesloten. Toch zal het nog een tijd duren vooraleer we het BIOS volledig kunnen naar de geschiedenis verwijzen. Een overzichtje van de ondersteuning tegenwoordig:

- Apple: standaard op alle recente computers (met BIOS-support)
- Microsoft: standaard vanaf vista X64 sp1
- Linux: reeds lange tijd standaard geïmplementeerd

Ook moederborden bieden tegenwoordig meestal ondersteuning, al zullen ze vaak nog manueel moeten ingesteld worden om UEFI te selecteren, BIOS blijft voorlopig nog even default.

## 6.5. Coreboot

# Chapter 7. Internal I/O

In dit hoofdstuk wordt een overzicht gegeven van de verschillende manieren waarop data uitgewisseld wordt op het moederbord, en met de systeemapparaten.

## 7.1. I/O transfers

Naast processor en geheugen vormt I/O het derde fundamenteel onderdeel van een computersysteem. Er zijn een aantal verschillende soorten manieren om data te versturen of te ontvangen van een I/O apparaat. Afhankelijk van het type I/O apparaat is een bepaald soort transfer beter geschikt dan de andere. Het is aan het besturingssysteem of aan de in het besturingssysteem geïntegreerde drivers om de I/O transfers af te handelen.

Drie types onderscheiden zich:

- pollen
- interrupts
- direct memory access

### 7.1.1. Pollen

'Pollen' staat voor bevragen. Dit is ook exact wat de processor zal doen: de processor zal voortdurend de I/O gaan bevragen om te kijken of er een actie van de CPU gewenst is.

Het voordeel van pollen is dat een verandering bij het I/O apparaat onmiddellijk wordt opgemerkt en de processor dus ook ogenblikkelijk kan reageren. Het belangrijkste nadeel is dat de processor voortdurend de toestand van het I/O apparaat moet controleren.

Wanneer de processor de toestand controleert, kan hij uiteraard geen andere (nuttige) taken uitvoeren.

Pollen vraagt dus behoorlijk wat rekenkracht, ook al is die niet echt nodig om het trage I/O apparaat aan te sturen.

Voor dit probleem kan eventueel gedeeltelijk een oplossing voor gevonden worden bij multitasking, waardoor het wachten kan afgewisseld worden met andere taken.

Voorgaande maakt waarschijnlijk wel duidelijk dat pollen niet echt bruikbaar is voor invoerapparaten, die moeten immers voortdurend gecontroleerd worden.

Polling wordt niet veel meer gebruikt voor gangbare PC I/O, maar is een zeer eenvoudige techniek en is dikwijls de eerste oplossing bij het schrijven van eigen interfaces voor bijvoorbeeld microcontroller-toepassingen.

### 7.1.2. Interrupts

In het geval van interrupt driven I/O zal het I/O apparaat de processor onderbreken. Dit betekent dat het I/O apparaat zelf om aandacht zal vragen en niet voortdurend gecontroleerd moet worden door de processor. De werking van interrupts zal hier uitgelegd worden voor een 8086 processor. Op een modern systeem verandert de werking van het geheel een beetje omdat de afscherming van I/O door een buscontroller (zie verder), maar de principiële afhandeling van een interrupt door een processor blijft gelijk.

#### Interrupthardware

De processor beschikt over een aparte INT-ingang, langs waar I/O een interruptsignaal kan doorgeven. Om meerdere interrupts van verschillende I/O toe te laten zou de processor moeten beschikken over meerdere ingangen, waar liefst nog een prioriteit aan verbonden kan worden.

Om dit mogelijk te maken wordt gebruik gemaakt van twee interrupt controllers (PIC - Programmable Interrupt Controller ), die elk beschikken over acht interrupt-ingangen. De uitgang van de eerste interrupt controller wordt verbonden met de INT-ingang van de processor, de uitgang van de tweede controller zit op ingang nummer 2 van de eerste controller. Aangezien de controllers intern prioriteit hebben van lage naar hoge cijfers, zijn er dus vijftien mogelijke interruptniveaus.

#### Interrupt transfer

Een interrupt transfer verloopt als volgt:

1. Als een I/O apparaat data (of een toestand) wil doorgeven aan de CPU, geeft het zelf een interruptsignaal.
2. Dit interruptsignaal komt toe op de interrupt controller. Als er geen andere interrupts zijn met hogere prioriteit, dan geeft de controller het signaal onmiddellijk door naar de processor. In het andere geval wordt pas na de afhandeling van de hogere prioriteitsinterrupt het interruptsignaal doorgegeven.
3. Als de controller het INT signaal geeft aan de processor, zal deze eerst de huidige instructie afwerken en zijn context opslaan (onder andere inhoud instruction pointer en statusregister).

4. Nu de processor klaar is voor de verwerking van de interrupt geeft hij een INTA (interrupt acknowledge) signaal door aan de interruptcontroller.
5. De controller reageert hierop door over de databus een vectornummer door te geven naar de processor.
6. Op basis van het vectornummer kan de processor de juiste interrupthandler starten en de interrupt afwerken
7. Zodra de interrupt volledig is afgewerkt, kan de processor zijn werk hervatten op de plaats waar hij onderbroken werd. Hiervoor laadt hij de instruction pointer met het eerder opgeslagen adres van de eerstvolgende af te werken instructie

Deze uiteenzetting zou duidelijk moeten maken dat de reactie van de processor op de gebeurtenis bij de I/O trager is dan bij polling. Eerst moet nog een instructie afgehandeld worden, vervolgens moet de juiste handler geladen worden en vervolgens kunnen pas de nodige acties ondernomen worden. Daar staat tegenover dat de processor alleen maar tijd moet spenderen aan de I/O op het ogenblik dat dit echt nodig is. Hierdoor zal de processor minder tijd spenderen aan de I/O operatie. Het maakt interrupts dan ook uitermate geschikt voor invoerapparaten.

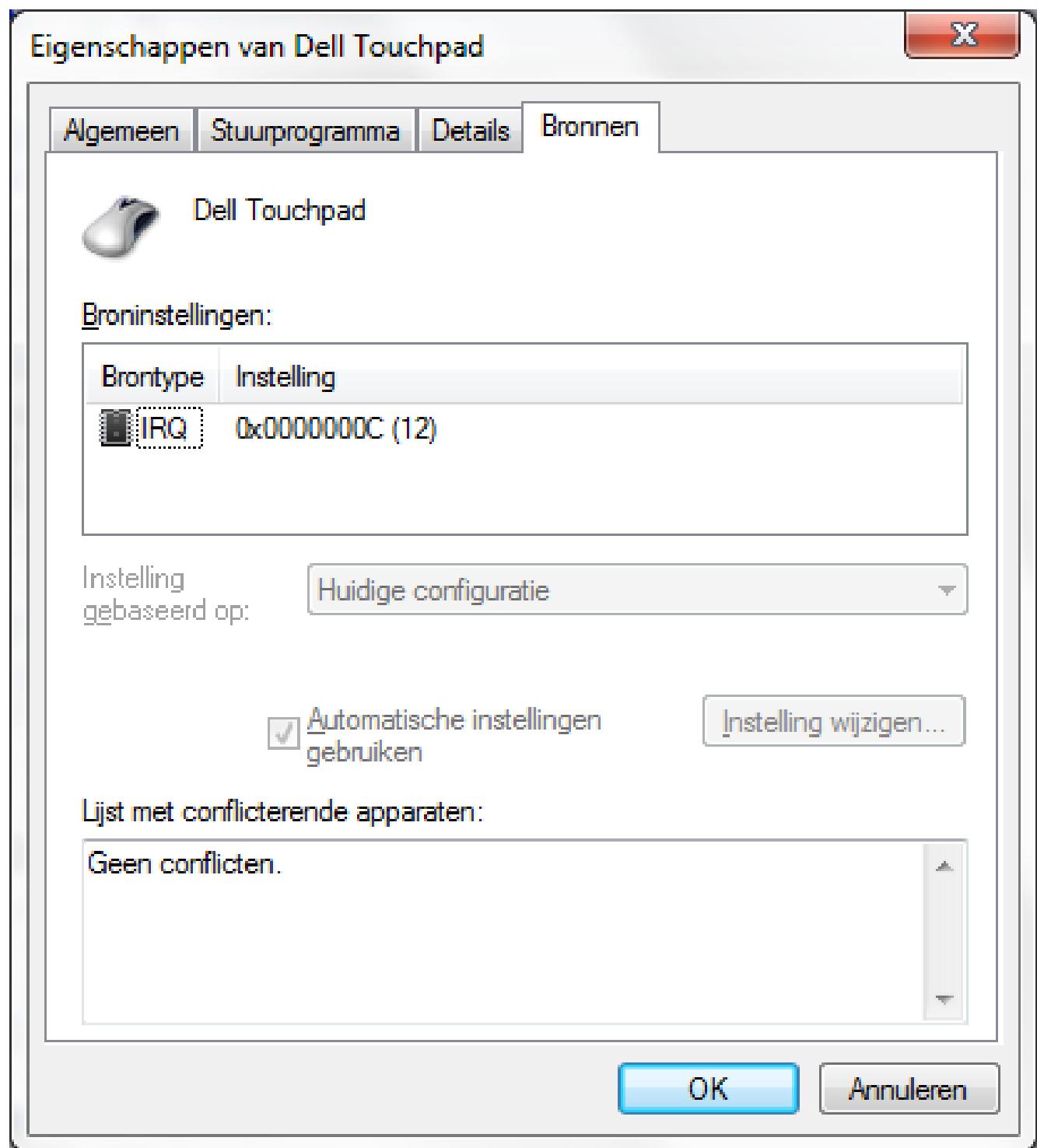
## Interrupts op bussystemen

Op de ISA bus (voorganger van PCI) waren interrupts vast gelinkt op een bepaalde IRQ. Dit moest ingesteld worden met jumpers en vroeg enige zorg, want een IRQ kon slechts door één apparaat gebruikt worden.

Om het gebruiksgemak te verhogen werd bij de PCI bus gebruikt gemaakt van een buscontroller die de PCI bus afschermt van de processor.

Een manier om op PCI een interrupt te genereren is het versturen van een speciaal interrupt bericht naar de bus controller. Dit bericht bevat de nodige informatie over het type en de bron van de interrupt. De bus controller vertaalt dit dan naar een IRQ voor de processor.

Op de PCI bus kunnen ook een aantal interruptlijnen beschikbaar zijn, die gedeeld kunnen worden tussen verschillende apparaten. Een interrupt op deze lijnen wordt dan door de controller vertaald naar een signaal dat uiteindelijk doorgegeven wordt aan de processor. Op die manier kunnen apparaten ook hetzelfde IRQ nummer delen.

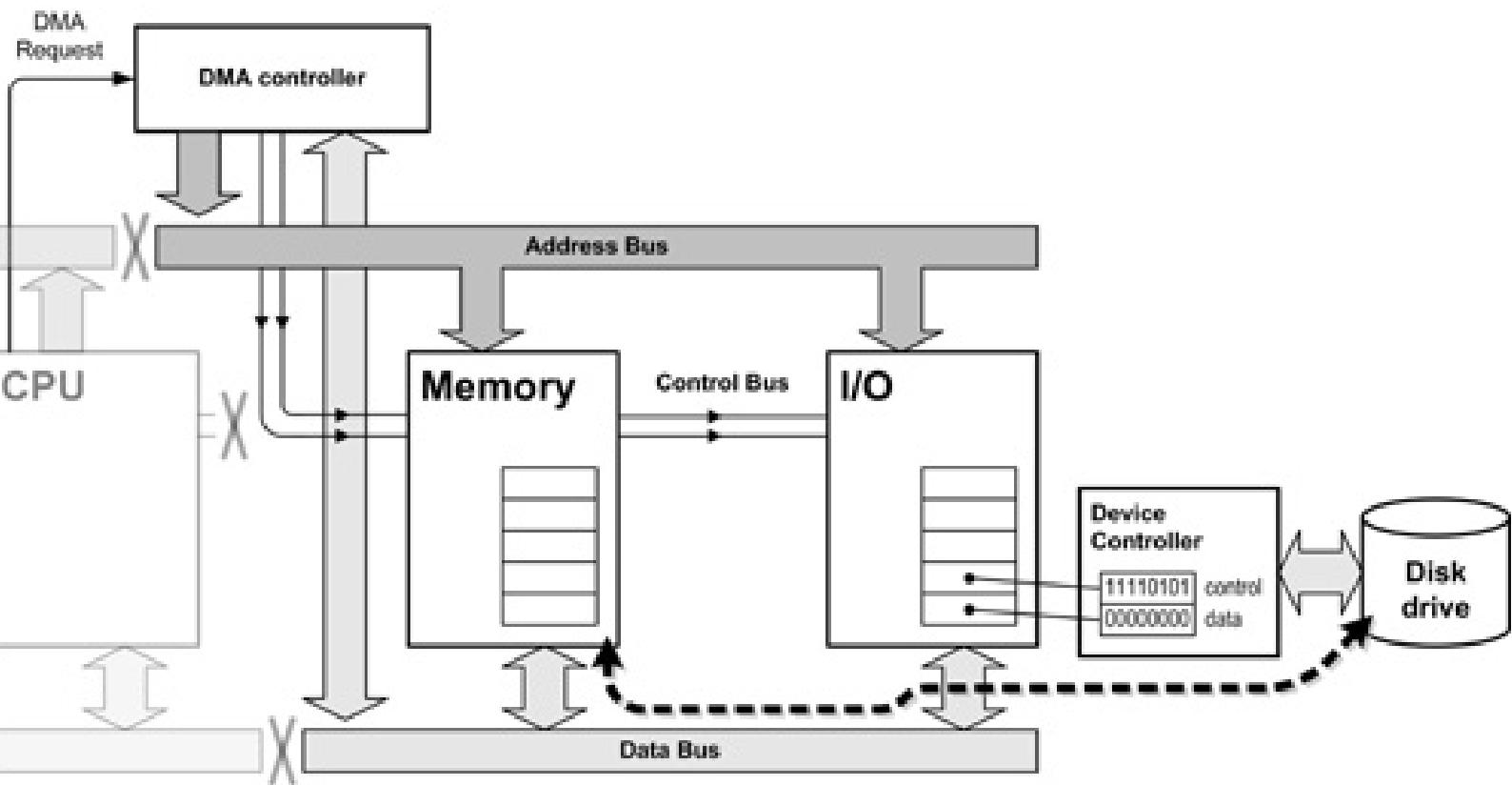


Aangezien de banen van de PCI slots wel vast op een bepaald PCI interruptkanaal zaten, waren (zeker in de beginlagen van PCI) nog wel conflicten mogelijk tussen apparaten die toch niet met elkaar overweg konden. In zo'n geval kon het volstaan om een van de uitbreidingskaarten naar een ander slot te verplaatsen om het conflict op te lossen.

### 7.1.3. Direct Memory Access (DMA)

Bij een Direct Memory Acces (DMA) toegang, zal het I/O apparaat rechtstreeks data uitwisselen zonder dat de processor hierbij moet tussenkomen. Hiervoor is er weer behoefte aan een DMA controller. Dit kan een aparte controller zijn, maar de functies kunnen ook geïntegreerd worden in de chipset of in de I/O controller zelf. Om het verloop van een DMA transfer duidelijk te maken, zullen we het lezen van een sector van een harde schijf bekijken als voorbeeld.

1. De DMA transfer begint met de processor die de opdracht doorgeeft aan de DMA controller. Hierbij wordt onder meer het aantal bytes en het geheugenadres doorgegeven
2. De I/O controller krijgt de nodige instructies, in dit geval dat er gelezen moet worden en op welk adres dit moet gebeuren
3. Het I/O apparaat onderneemt de nodige acties (koppen verplaatsen, lezen) om de data beschikbaar te maken in het buffer
4. Zodra de data beschikbaar is, geeft de I/O controller een DRQ (DMA-request) signaal door aan de DMA controller.
5. De DMA controller probeert nu de controle te krijgen over de bus. Dit gebeurt door een HOLD signaal door te geven aan de processor.
6. De processor werkt eventuele lopende transfers op de bus af en geeft de bus dan vrij met een HOLDA signaal.
7. De DMA controller heeft nu controle over de databus en kan nu de transfer van schijf naar geheugen sturen. Dit gebeurt door het juiste geheugenadres op de bus te plaatsen en met een DACK-signaal aan de harde schijf aan te geven dat de data op de bus geplaatst mag worden.
8. Als de data verplaatst is, worden alle stuursignalen inactief gemaakt en wordt de DMA cyclus beëindigd.



**Figure 7.1. DMA request (bron:<http://www.talktoanit.com/>)**

Het belangrijkste voordeel van dit soort transfer is dat de processor niet betrokken is bij het eigenlijke verplaatsen van de gegevens. In dit voorbeeld moet de CPU enkel de nodige commando's doorgeven aan de DMA controller. Het voordeel hiervan is uiteraard dat de processor ondertussen kan verder werken en dus niet moet wachten op de trage harde schijf. Bovendien worden de gegevens nu meteen in het geheugen geplaatst, anders zou de processor ze eerst moeten lezen van de schijf en dan naar het geheugen schrijven. Tijdens de eigenlijke overdracht van gegevens is de bus natuurlijk bezet en kan de processor daar geen gebruik van maken. Dit heet cycle stealing. De DMA controller steelt tijd van de processor op de bus. Hoewel de DMA controller toegang tot de bus aanvraagt, zal hij bijna altijd voorrang krijgen op de processor.

Eens een I/O apparaat zoals een schijf op gang komt, is het belangrijk om de gegevens zo snel mogelijk te kunnen verplaatsen. Meestal zal het ook zo zijn dat de DMA controller meerdere keren de bus moet overnemen om bytes te verplaatsen.

Dit hangt natuurlijk ook af van de grootte van buffers en interface-snelheden. Het opstarten van een DMA transfer kost wel wat extra processorcycli. Alle nodige gegevens doorgeven aan de DMA controller kost nu eenmaal tijd. DMA is dan ook een transfermethode die vooral voordeel biedt als er grote hoeveelheden data verplaatst moeten worden. Dan is slechts een opdracht naar de DMA controller nodig, die vervolgens autonoom de transfer kan regelen.

## 7.2. Bussystemen

### 7.2.1. In den beginne...

In de geschiedenis van de personal computer zijn reeds een heel aantal verschillende bussystemen de revue gepasseerd.

De eerste IBM PC gebruikte als bus de zogenaamde PC-bus. In feite was dit in grote lijnen een gebufferde versie van de processor in- en uitgangen.

Er waren 62 signaallijnen: 20 adreslijnen, 8 datalijnen, een paar controlelijnen (I/O read/write, memory read/write) en een aantal lijnen ten behoeve van interrupts en DMA transfers. Bij de lancering van de PC/AT met 80286 stond IBM voor een dilemma. Er moest een keuze gemaakt worden voor een bussysteem, dat ofwel toeliet om de kaarten voor de PC-bus te blijven gebruiken ofwel gebruik kon maken van de extra mogelijkheden van de 80286 (24 adreslijnen, 16 datalijnen).

De oplossing was het uitbreiden van de slots op de PC bus met een extra connector. Nieuwe kaarten konden gebruik maken van dit extra deel en dus ook van de extra mogelijkheden in het nieuwe systeem.

Oudere kaarten konden gewoon verder gebruikt worden langs het PC-bus gedeelte, uiteraard zonder de voordelen van de nieuwe processor. Toen IBM zijn PS/2-serie uitbracht, vonden ze de tijd gekomen om een nieuwe bus te ontwikkelen. Dit was de microchannel bus. Een deel van de reden was dat de PC-bus ondertussen echt wel verouderd was, een ander deel van de reden was dat IBM een obstakel wilde opwerpen voor de kloombouwers. Microchannel werd dan ook afgeschermd met een muur van patenten met daarachter een legertje advocaten.

De reactie van de overige PC-bouwers was de ontwikkeling van een eigen bus, namelijk de ISA-bus (Industry Standard Architecture). Dit was in feite gewoon een AT-bus die op een hogere klokfrequentie werkte. Belangrijk voordeel was dat deze bus compatibel was met de oudere kaarten. Bovendien waren er een groot aantal fabrikanten van uitbreidingskaarten, die vrij van licentiezorgen, kaarten ontwikkeld hadden voor de AT-bus en dit konden doen voor de ISA-bus.

Gevolg was dat dit de facto standaard werd en dat IBM in de vrij absurde situatie terecht kwam, waarin het als enige geen IBM-compatibele PC's produceerde en het bijna volledig uit de PC-markt verdreven werd. Later werd de ISA-bus nog uitgebreid naar de 32 bit EISA (Extended ISA) versie.

Op moderne computersystemen zal je deze bussen niet meer aantreffen.

### 7.2.2. PCI

Uit het voorgaande blijkt al dat de ISA-bus technologisch zeker niet superieur is. De belangrijkste kwaliteit was de backward compatibiliteit met oudere uitbreidingskaarten. Het gebrek aan bandbreedte werd echt een belangrijk tekort als grafische interfaces en toepassingen hun intrede deden. Een rekenvoorbeeld om dit duidelijk te maken: om full screen bewegende kleurbeelden weer te geven op een computerscherm van 1024\*768 pixels zijn dertig beelden per seconde nodig.

Als voor de kleurweergave per pixel drie bytes nodig zijn (RGB), dan is een transfersnelheid nodig van

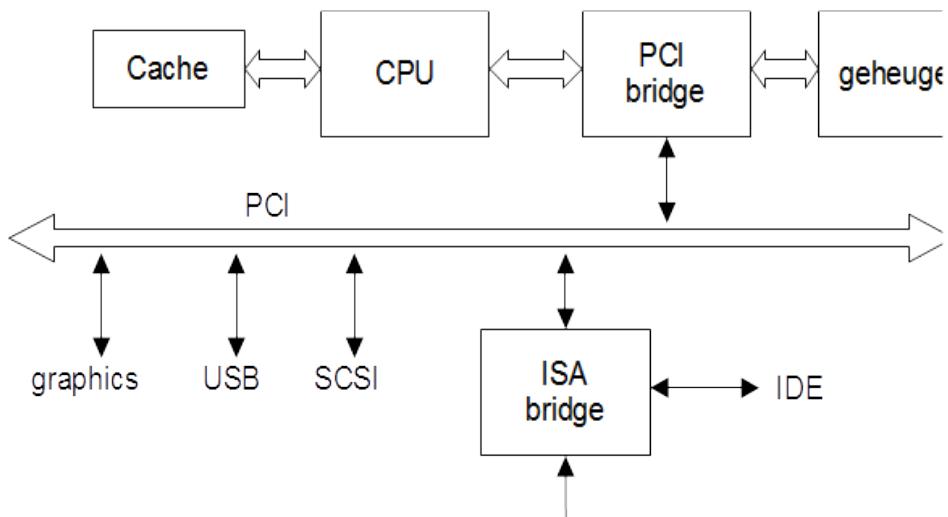
$$3 \text{ Byte} * 1024 * 768 * 30 = 70 \text{ MB // sec}$$

De EISA bus met 32 bit databus en 8,33MHz kloksnelheid, kon maximaal 33,3MB/s geven. Bovendien moet de bus nog gedeeld worden door andere apparaten, waaronder het geheugen, waarvoor deze snelheid ook onvoldoende zal blijken. De oplossing bestaat erin een nieuw bussysteem te ontwikkelen. Dit zal de PCI-bus (Peripheral Component Interconnect) worden.

PCI werd ontwikkeld door Intel, dat verstandig genoeg was om de patenten in het publiek domein te plaatsen, zodat alle fabrikanten randapparatuur konden bouwen, zonder hiervoor rechten te moeten betalen. Intel richtte ook de PCI Special Interest Group op, een industrieconsortium dat de toekomst van de PCI-bus moest regelen. Belangrijke eisen bij de ontwikkeling van PCI zijn onder andere:

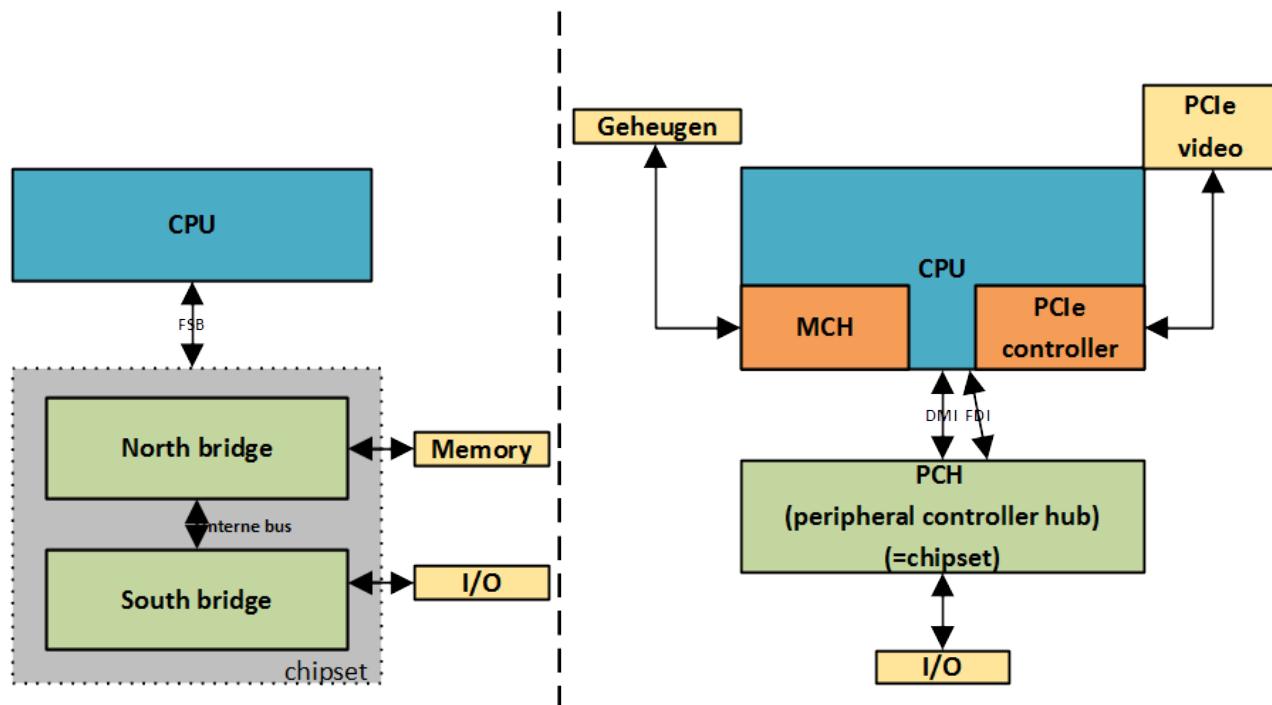
- zorgen voor voldoende bandbreedte, ook naar het geheugen
- backward compatibility verzekeren
- kleinere connector dan ISA slots
- ondersteuning voor plug-and-play, power management (laag stroomverbruik)
- hiërarchie van bussen

De eerste twee eisen zijn uiteraard de belangrijkste. De eerste ligt voor de hand, de tweede heeft in de PC-geschiedenis zijn belang bewezen. De oplossing voor beide ligt in de ontwikkeling van een I/O systeem, waarin verschillende bussen aanwezig zijn. Er ontstaat op die manier een bushiërarchie van snellere naar tragere bussen, waarbij over de snelle bussen data uitgewisseld kan worden zonder rekening te moeten houden met de tragere onderdelen van het computersysteem (die op een tragere bus aangesloten zijn).



Bovenstaande afbeelding geeft de opbouw van het bussysteem bij de ontwikkeling van de PCI-bus. De processor was via de snelle back-side bus verbonden met het cachegeheugen, dat toen nog niet (volledig) in de processor geïntegreerd was. Via de front-side bus was de processor verbonden met de PCI-bridge.

De PCI bridge kon dan enerzijds verbinding maken met het hoofdgeheugen en anderzijds met de PCI bus. Op de PCI bus kon dan de ISA bridge aangesloten worden, die verbinding kon maken met de IDE kanalen en uiteraard ook met de ISA-bus. De compatibiliteit werd verzekerd door naast PCI slots ook een aantal ISA slots te blijven voorzien. Op de PCI bus kon dan allerhande hardware aangesloten worden, waaronder ook adapters voor andere bussystemen zoals SCSI en USB. De IC's die de verschillende bussystemen van elkaar scheiden, vormen de chipset. Het is dit onderdeel van het computersysteem dat de ontwikkeling voor Intel zo interessant maakte. Intel wou (en zou) immers de chipsets ontwikkelen en verkopen. De namen voor de verschillende bridge chips veranderd in de loop der tijden wel een aantal keer (net zoals de functies die erin geïntegreerd zijn). Gangbare, maar verouderde terminologie hiervoor zijn North en South bridge, waarbij PCI gekoppeld was aan de south-bridge. Nieuwere versies verlaten de 'north' en 'south' terminologie. De micro-architectuur die momenteel gebruikt wordt, vind je hieronder:



**Figure 7.2. oude en nieuwe microarchitectuur**

Het gebruik van de bridges heeft als bijkomend voordeel dat de PCI-bus eigenlijk processoronafhankelijk wordt. Hoe transfers op PCI verlopen is volledig van de processor afgeschermd door de bridge (die uiteraard wel processor-afhankelijk is). Dit maakt PCI eigenlijk platformonafhankelijk, wat betekent dat het ook met andere processoren gebruikt werd (b.v. ultrasparc).

## Transfers

PCI is een gedeeld bussysteem. Dit betekent dat de datalijnen gebruikt worden door alle aangesloten apparaten. Dit betekent dus ook dat er enerzijds nood is aan adressering (welk apparaat moet data accepteren) en anderzijds nood aan toegangscontrole (twee apparaten mogen niet tegelijkertijd toegang krijgen tot de bus). Adreslijnen en datalijnen worden op de PCI-bus gemultiplexed. Dit zorgt ervoor dat er eerst een adrescyclus nodig is en dat pas daarna de data getransfereerd kan worden. Dit is iets trager, maar bespaart ruimte op de connector. Er zijn verschillende manieren om toegang tot een bus te controleren. Op PCI is het zo dat elk apparaat op elk ogenblik een transfer kan starten. Elk apparaat kan initiator zijn.

Om conflicten te vermijden is er op de PCI bus ook een arbitrator, deze functie is meestal geïntegreerd in de chipset. Een apparaat dat toegang wil tot de bus zal dit aanvragen bij de arbitrator. Indien meerdere apparaten tegelijk toegang vragen, zal de

arbitrator een van de apparaten toegang verlenen via een grant-singaal. De andere moeten wachten. Het apparaat met het grant signaal wordt op dat ogenblik master op de PCI-bus. Het algoritme dat hierbij gebruikt wordt, ligt niet vast in de standaard, maar is bij voorkeur wel rechtvaardig, zodat elk apparaat aan bod komt.

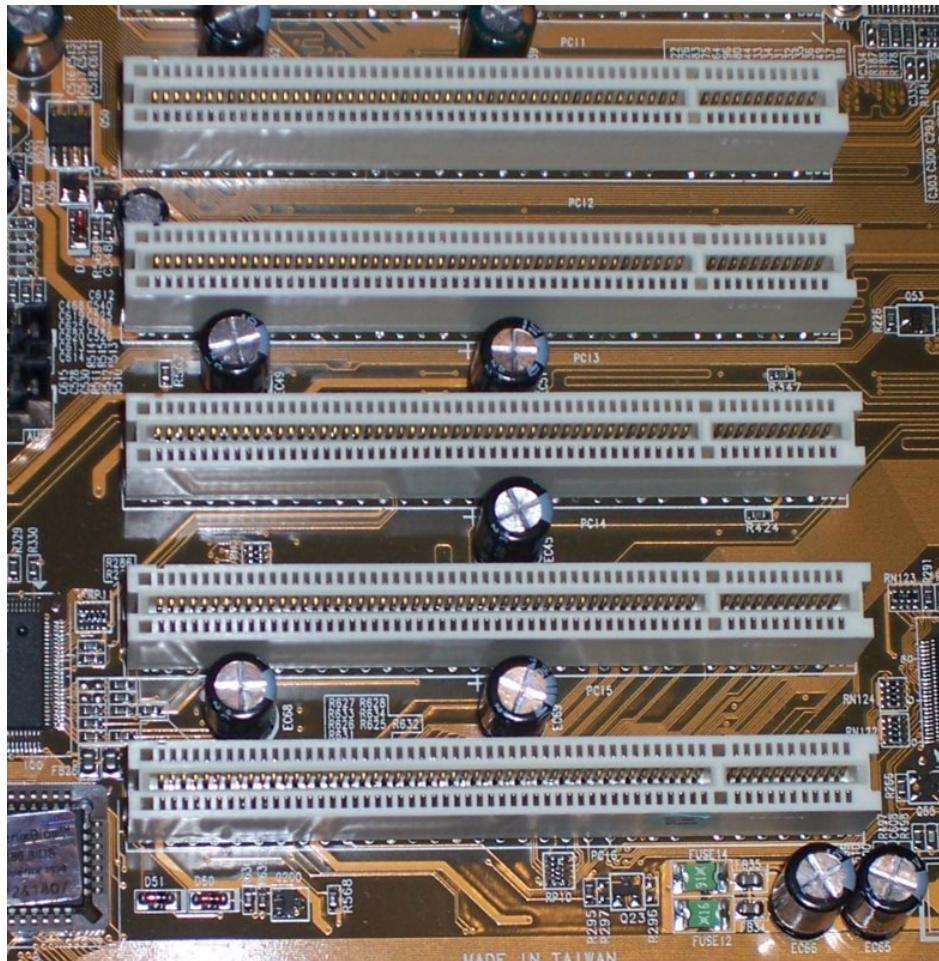
Eens een apparaat toegang tot de bus krijgt kan het dan een data transfer starten. Deze kan in principe variëren in het aantal bits dat overgebracht wordt, zodat grote blokken getransfereerd kunnen worden en de bus dus lang aan haar maximale datasnelheid kan werken. Om te vermijden dat een apparaat met een lange transfer de bus gaat monopoliseren, moet het voortdurend zijn grant-singaal in de gaten houden. Zodra een ander apparaat toegang tot de bus wil, zal de arbitrator het grant-singaal wegnemen. De data die al op de bus staat, wordt afgeleverd, waarna de bus wordt vrijgegeven. De onderbroken transfer moet uiteraard later afgewerkt worden, daarvoor zal het apparaat weer toegang vragen.

## Snelheid

De originele PCI bus maakte gebruik van een 32 bits databus en 33MHz klokfrequentie. Dit geeft een maximale bandbreedte van 133MB/s. Belangrijke opmerking is dat deze snelheid niet voortdurend gehaald wordt. Op sommige ogenblikken moeten andere gegevens verstuurd worden over de bus(bijvoorbeeld adres). Bovendien moet deze bandbreedte verdeeld worden over alle aangesloten apparaten, zodat de gemiddelde beschikbare snelheid voor een bepaald apparaat lager zal liggen dan de vermelde 133MB/s.

Er bestaan varianten van PCI die hogere snelheden toelaten. Dit kan ofwel door de breedte van de databus te vergroten tot 64 bit ofwel door de kloksnelheid te verhogen tot 66MHz (in beide gevallen geeft dit een verdubbeling van de snelheid tot 266MB/s) ofwel door een combinatie van beiden (533MB/s). Deze laatste variant heet PCI-X (PCI eXtended) en is nog terug te vinden in oudere servers en workstations. In een gewone desktop PC vind je deze variant niet terug.

Ter illustratie staan in onderstaande figuur de bekende 32-bit connectoren.



**Figure 7.3. PCI slots**

### 7.2.3. AGP (Accelerated Graphics Port)

Uit de eerdere berekening voor de bandbreedte voor het bekijken van videobeelden en de bandbreedte van PCI kan je afleiden dat er al snel problemen ontstonden op het vlak van beschikbare bandbreedte. Een oppervlakkige vergelijking van de cijfers leidt tot de conclusie dat de 133MB/s voldoende is voor de benodigde 70MB/s. Als je de cijfers iets nauwkeuriger gaat bekijken, dan moet je vaststellen dat de 70MB/s echt nodig zijn en de 133MB/s enkel gehaald worden op het ogenblik dat een apparaat echt data stuurt. Als we rekening houden met andere apparaten die data moeten uitwisselen (bijvoorbeeld de harde schijf of dvd waar de film op staat), dan blijkt al snel dat PCI onvoldoende bandbreedte biedt voor de huidige eisen aan video. Dezelfde conclusie kan je ook trekken als je rekening houdt met hogere schermresoluties. Om dit probleem aan te pakken, werd het bussysteem uitgebreid met een extra interface (AGP). De functionaliteit voor het besturen van deze interface werd geïntegreerd in de chipset. Deze chipsets werden initieel de north bridge en south bridge genoemd.

Merk op dat de north en south bridge ondertussen omgedoopt zijn in memory controller hub (MCH) en I/O controller hub (ICH). In de nieuwste processoren is de North Bridge zelfs helemaal opgegaan in de processor.

De besturing van AGP bevindt zich in de MCH en de besturing van PCI in de ICH. AGP bood in zijn eerste versie (AGP 1x) een databus van 32 bit bij 66MHz, hetgeen een bandbreedte geeft van 266MB/s. Behalve de vergroting van de kloksnelheid heeft AGP nog een bijkomend voordeel dat tot een grotere datasnelheid leidt: de AGP verbinding is een punt-tot-punt verbinding en de beschikbare bandbreedte moet dus niet gedeeld worden en is ogenblikkelijk beschikbaar. Een ander voordeel van AGP is dat het voorziet in mogelijkheden om rechtstreeks uit het geheugen te lezen, door gebruik te maken van een Graphics Addressing Remapping Table.

Een grafische kaart op PCI moest eerst de data kopiëren naar het framebuffer op de kaart. AGP komt in een aantal varianten. Het voornaamste onderscheid is de snelheid. Naast AGP 1x, bestaan ook 2x, 4x en 8x. Deze boden respectievelijk 533,1066 en 2133MB/s aan over een databus van 32 bit. Het verschil tussen elk van deze varianten is dat de kloksnelheid volgens het DDR principe wordt vergroot. De basisfrequentie was steeds 66MHz, maar deze werd op 2x verdubbeld naar 133MHz (net zoals bij DDR). Op 4x en 8x wordt ze respectievelijk verviervoudigd (zoals bij DDR2) en verachtvoudigd (zoals bij DDR3).

Daarnaast is er ook nog variatie in voedingsspanningen (eerst 3.3V, daarna 1.5V en tenslotte 0.8V). Hierbij moet wel opgelet worden met compatibiliteit.

De pro-versies vormen een eerder zeldzame variant, waarbij er extra stroom geleverd kan worden.

#### 7.2.4. PCI-express

Er komt steeds meer hardware die een te grote bandbreedte vraagt voor PCI. Een voorbeeld zijn gigabit netwerkkaarten die een bandbreedte van 125MB/s vragen, zeer dicht bij de (gedeelde bovenlimiet van) 133MB/s die PCI kan bieden. Een oplossing zou kunnen zijn om voor die apparaten de functie van de chipset aan te passen en een aparte poort te voorzien (zoals voor AGP). Op deze manier wordt de flexibiliteit wel beperkt en daalt ook de overzichtelijkheid van het systeem. Het zou interessanter zijn om een bus te ontwikkelen die gewoon een hogere bandbreedte haalt en bijvoorbeeld ook voldoende bandbreedte kan bieden om grafische kaarten te ondersteunen. Er zijn een aantal oplossingen ontwikkeld, maar diegene die het uiteindelijk gehaald heeft is PCI-express (PCIe) (ook hier was Intel weer een van de drijvende krachten achter de ontwikkeling).

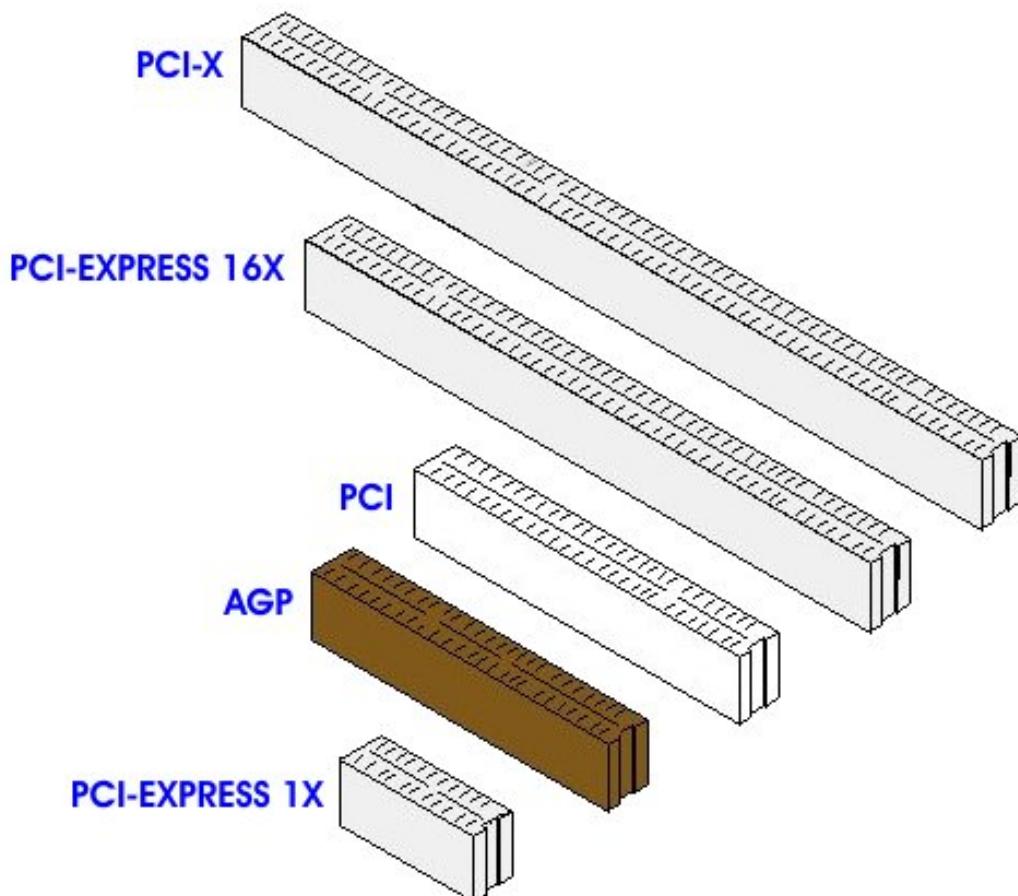
## Architectuur

Het ontwerp van PCIe breekt radicaal met dat van de traditionele bussystemen. In plaats van het traditionele concept van een bussysteem met een brede databus, die gedeeld wordt door meerdere apparaten, gebruikt PCIe een snelle seriële punt-tot-punt verbinding.

Seriële verbindingen hebben als belangrijk voordeel dat problemen met looptijdverschillen en overspraak vermeden kunnen worden, waardoor een veel hogere kloksnelheid gebruikt kan worden. De topologie lijkt dan ook geweldig op die van switched ethernet.

Centraal in het concept staat een PCIe switch, die met een aantal gepaarde seriële links verbonden is met de aanwezige hardware.

## Snelheid



**Figure 7.4. PCIe slots**

De verbinding tussen de PCIe-switch en de I/O bestaat uit een of meer paren van eenrichtingslinks. Een link bestaat dus uit een upstream en downstream-verbinding.

Een dergelijke link noemt men een lane. In het eenvoudigste geval bestaat de verbinding uit een lane, maar het kunnen ook 2, 4, 8, 16 of 32 paren zijn. Uiteraard nemen uitbreidingsslots met meerdere lanes meer plaats in op het moederbord (zie bovenstaande afbeelding).

Op elke lane wordt er snelheid gehaald van 2,5Gbps. Deze snelheid zal in de toekomst nog kunnen vergroten (PCIe2.0: 5Gbps, 3.0: 8Gbps, 4.0: 16Gbps). De vermelde snelheid is wel een brutosnelheid, dus zonder rekening te houden met extra informatie die verzonden moet worden. vergeleken met de maximale transfer rate van PCI is de snelheid op per lane ongeveer 2,3 keer groter.

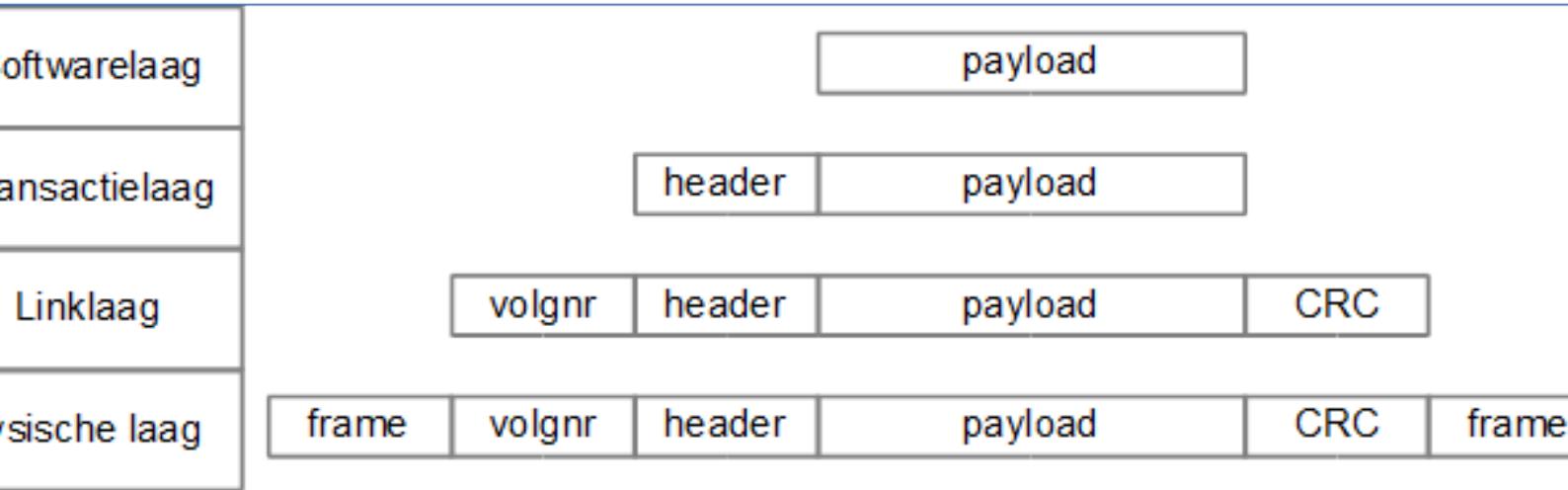
## Protocol stack

Gebruik maken van een snelle synchrone seriële verbinding vraagt wel dat er extra informatie verzonden wordt. Eerst en vooral moet ervoor gezorgd worden dat zender en ontvanger met elkaar gesynchroniseerd geraken. Aangezien er geen extra lijnen voorzien zijn voor een gemeenschappelijk kloksignaal of om te signaleren dat de transmissie start, moet de zender de start van een pakket aangeven en zorgen dat de bitstroom voldoende synchronisatie informatie bevat. Het eerste probleem wordt aangepakt door de te verzenden data in te pakken tussen twee gekende vlaggen, die start en einde van de data aangeven. Om twee partijen met elkaar te synchroniseren, zonder extra kloklijn, moeten er in het verzonden datapatroon regelmatig overgangen voorkomen.

De oplossing die op PCIe gebruikt wordt (althans in versie1), is een 8b/10b codering. Dit betekent dat om acht bits te versturen er werkelijk tien verstuurd worden. De vertaling van acht bits naar tien bits is natuurlijk zodanig dat in elk mogelijk verzonden patroon voldoende overgangen zitten. Merk op hoe deze vertaling de brutosnelheid vermindert naar 2Gbps. Een nieuwigheid ten opzichte van PCI is de aanwezigheid van flow control en error control.

Flow control moet zorgen dat de zender niet sneller data verstuurt dan de ontvanger ze kan verwerken. Het principe dat wordt toegepast is gelijkaardig aan het sliding window principe dat ook door TCP wordt toegepast: de zender krijgt een bepaald venster en kan binnen dit vensterdata versturen. Bij bevestiging van pakketten verschuift het venster en kan de zender weer pakketten versturen. Error control wordt verwezenlijkt door een foutdetecterende code (CRC -Cyclic Redundancy Check) toe te voegen aan het pakket. De ontvanger controleert de data op fouten en vraagt in het geval van een fout een hertransmissie aan. Het voordeel van deze foutcontrole is dat PCIe een stuk betrouwbaarder wordt, maar vooral dat hogere kloksnelheden mogelijk worden. Bij hogere kloksnelheden wordt de kans op bitfouten groter, maar door de aanwezigheid

van foutcontrole is dit niet noodzakelijk een drama (te veel bitfouten gaan natuurlijk de netto snelheid nog meer laten afnemen).



**Figure 7.5. PCIe protocolstack**

Omdat er op de seriële verbinding geen plaats is voor controlelijnen, moet alle controle-informatie verzonden worden in een header. Deze laat onder andere toe om het soort transfer aan te geven (bijvoorbeeld gewone I/O, configuratie bericht voor PnP, interrupt, ...).

### Vergelijking met PCI

Ten opzichte van PCI zijn er een aantal fundamentele veranderingen, waarbij de overgang naar seriële transmissie waarschijnlijk de grootste is.

Deze verandering, gecombineerd met hogere kloksnelheden en de foutcorrectie laat veel grotere datasnelheden toe. Bovendien moet de beschikbare bandbreedte op de punt-tot-punt verbinding niet gedeeld worden. Het is wel mogelijk om een PCIe apparaat zelf te gebruiken als switch en op die manier grotere netwerken te bouwen. In dat geval wordt een deel van het pad richting centrale switch en dus ook de bandbreedte gedeeld. De seriële verbinding met zijn zeer beperkt aantal geleiders laat ook veel kleinere connectoren toe, wat het geschikter maakt voor bijvoorbeeld laptops. Ook andere ontwerpen van computerbehuizing (bijvoorbeeld Apple-gewijs alles in het scherm steken) worden eenvoudiger. Een minder voor de hand liggend voordeel van de minder storingsgevoelige seriële verbinding is dat de afstand tussen communicerende apparaten groter kan worden.

Uit al deze verschillen zou duidelijk moeten blijken dat PCI Express bijzonder weinig te maken heeft met PCI. Behalve de naam (die om marketingredenen begint met PCI) hebben ze enkel gemeen dat dezelfde commando's ondersteund worden. In tegenstelling tot wat de naam probeert te suggereren, zijn PCI en PCI express absoluut niet compatibel. Om te zorgen voor backwards compatibility worden op de moederborden (zeker de eerste jaren) nog een aantal PCI-slots voorzien. De PCI controller functionaliteit is natuurlijk terug te vinden in de ICH.

## 7.3. Bussystemen voor harde schijven

### 7.3.1. In den beginne...

De IDE, ATA en ATAPI benamingen worden zeer vlot door elkaar gebruikt om bussystemen voor harde schijven (origineel) en optische stations aan te duiden.+ Op de eerste IBM PC was er optioneel een harde schijf van 10MB. De schijfcontroller zat op het moederbord en er liepen analoge signalen naar de schijf om de motoren te sturen. Naarmate de technologie zich verder ontwikkelde, werd de controller geïntegreerd op de schijf en ontstond een IDE-schijf (Integrated Drive Electronics). Een IDE interface had een 16-bit databus en voorzag origineel in de 20 bit CHS adressering. Transfers gebeurden volgens het polling principe (programmed I/O) en haalden snelheden van 3.3, 5.2 of 8.3 MB/s naargelang de gebruikte kloksnelheid. Na IDE kwam er EIDE (Extended IDE) dat voorzag in LBA met 28 bit en dus grotere schijven kon aanspreken. Een andere belangrijke verbetering was een verhoging van de datasnelheid waardoor maximaal 16.6MB/s gehaald kon worden. Bovendien deed een nieuwe transfertechniek zijn intrede: multiword DMA.

Andere verbetering was dat een EIDE controller meerdere apparaten kon aansturen. Ze konden twee kanalen aan en op elk kanaal twee apparaten. Het ene apparaat werd master genoemd, het ander slave. Master en slave-instellingen moesten via jumpers op de schijf gedaan worden. Nadien kon de keuze ook gebeuren met de kabel (cable select). Het is een gangbaar misverstand om te denken dat de master ook effectief de transacties controleert en dus de slave aanstuurt. Het is steeds de EIDE-controller die de bus bestuurt.

Master en slave zijn eerder een soort van adressen (in moderne versies van de standaard spreekt men ook van device 0 en device 1). Naarmate EIDE verder ontwikkelt werd en de controller aangesloten werd op de AT bus, duikt plotseling de naam ATA-3 op (AT attachment). De vorige twee standaarden krijgen als extra naam ATA-1 en ATA-2.

ATA-3 voegt weinig extra toe, belangrijkste bijdrage is S.M.A.R.T. De opvolger van ATA-3 zou logischerwijs ATA-4 zijn, maar plotseling spreekt men nu van ATA Packet Interface en dus ATAPI-4.

Belangrijke aanpassingen zijn de ondersteuning voor optische stations en de hogere transfersnelheden(tot 33MB/s).

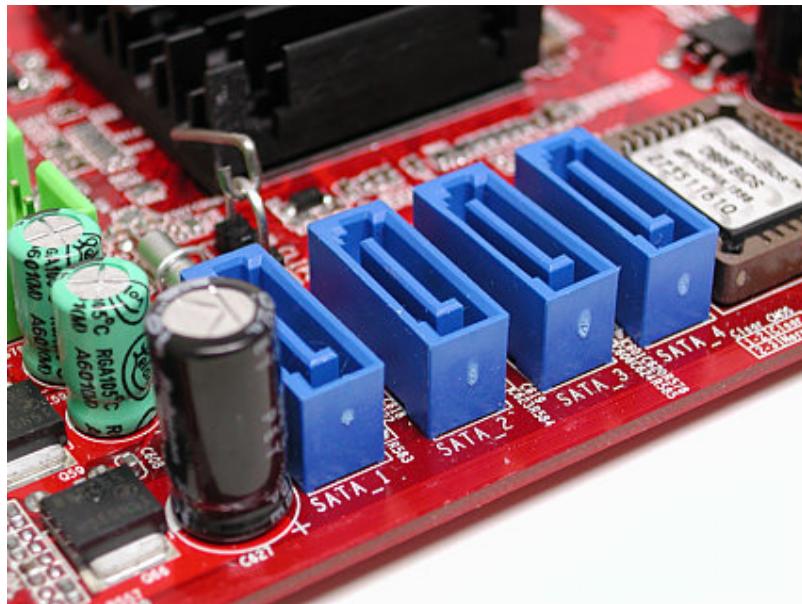
Er wordt ook een nieuwe transfertechniek geïntroduceerd: ultra-DMA. Hierbij werden transfers op beide klokflanken mogelijk en werd CRC gebruikt om fouten op te sporen.ATAPI-5 ging tot 66MB/s en introduceerde hiervoor een kabel met 80 geleiders. ATA/ATAPI-6 verhoogde de snelheid tot 100MB/s en introduceerde 48 bit LBA adressering. ATA/ATAPI-7 introduceerde enerzijds UDMA/133 (133MB/s) en anderzijds SATA.

Er zijn (waren?) twee soorten ATA-kabels. Het aantal aansluitingen op de connectors is echter niet gewijzigd, de extra geleiders worden allemaal verbonden met de aarding. Ze komen tussen de signaalgeleiders te liggen, zodanig dat er steeds afwisselend een signaal geleider en een geaarde draad ligt. Op die manier wordt de capacitieve koppeling en de daarmee gepaard gaande overspraak tussen twee signaaldraden verminderd. Dit laat hogere datasnelheden toe. Voor transfers vanaf 66MB/s moet verplicht een 80-polige kabel gebruikt worden.

PATA kan ondertussen gecatalogeerd worden als geschiedenis.

### 7.3.2. SATA

Het verhaal van de overgang van ATA (dat ondertussen ook PATA genoemd wordt) naar SATA is heel gelijkaardig aan het verhaal van PCI naar PCI-express. Ook in dit geval wordt er overgegaan van een parallelle interface naar een veel sneller geklokte seriële interface, die hogere datasnelheden toelaat.



**Figure 7.6. sata connector**

### Compatibiliteit sata vs pata

Zoals steeds in de PC wereld moet er opgelet worden voor backward compatibility. Uit het voorgaande zal wel duidelijk gebleken zijn dat PATA en SATA niet compatibel zijn met elkaar. Om die reden worden (voorlopig) nog extra PATA connectoren voorzien. PATA en SATA zijn wel software compatibel, in die zin dat de commando's die op PATA gebruikt worden ook begrepen worden op SATA. Dit maakte de stap naar SATA hardware eenvoudiger.

### Signalering en bandbreedte

De dataverbinding tussen schijf en moederbord bestaat uit zeven geleiders. Drie daarvan zijn ground, de andere vier vormen twee paren voor dataverkeer. Met deze kleine connector gaat uiteraard ook een veel kleinere kabel gepaard, wat de luchtstroming in de computerkast ten goede komt.

Door de overgang van parallel naar serieel zijn hogere kloksnelheden mogelijk om de eerder vermelde redenen. Bovendien werkt SATA differentieel, dit wil zeggen dat de data verzonden wordt over paren. De geleiders van deze paren liggen dicht bij elkaar, zodat ze ongeveer op dezelfde manier gestoord worden. Aangezien het verschil tussen de geleiders de data bepaalt, is het mogelijk om een groot deel van deze storingen weg te werken. De hogere kloksnelheid bedroeg in de eerste versie van SATA 1500MHz. Hierdoor was een bitrate mogelijk van 1500Mbps. Ook hier wordt omwille van de synchronisatie een 8b/10b codering gebruikt, zodat de werkelijke bandbreedte 120MB/

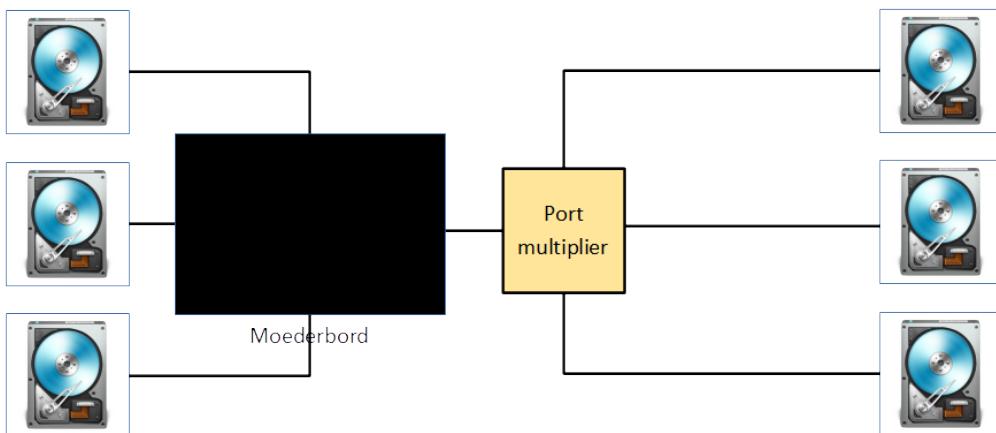
s wordt. Dit is eigenlijk lager dan de maximale 133MB/s, maar hoger dan de maximale transfer rates van schijven.

### Example 7.1. oefening

Serial ATA 3 haalt een bitrate van 6Gbps, en toch maar een nuttige bandbreedte van 600MB/s. hoe verklaar je dat?



In tegenstelling tot PATA heeft SATA een verbinding in elke richting, zodat in principe full duplex mogelijk wordt (het is echter niet zo voor de hand liggend om tegelijk de schijf te schrijven en te lezen). Anderzijds beschikt SATA niet over controlelijnen om de commando's door te sturen. Deze moeten ook over de dataverbinding verzonden worden (wat de netto bandbreedte nog iets meer zal doen dalen), maar hier kan eventueel wel gebruik gemaakt worden van de full duplex eigenschap.



**Figure 7.7. sata multiplier**

In principe is SATA een punt-tot-punt verbinding tussen de SATA controller en de schijf. Het is echter ook mogelijk om met behulp van een expander of multiplier meerdere SATA apparaten aan te sluiten op een connector van de controller (afbeelding 63). Dit betekent dus ook dat (tenzij met expander) de bandbreedte niet gedeeld moet worden over verschillende apparaten. Voor de signalering op de dataparen gebruikt SATA LVDS (Low Voltage Differential Signaling). In plaats van de op PATA gangbare 5V, wordt een spanningszwaai van 0.5V gebruikt. Belangrijk voordeel hiervan is een veel lager stroomverbruik. Dit wordt zoals altijd op de mobiele markt gewaardeerd, maar

ook aan het andere uiterste, in bijvoorbeeld data centers, waar deze besparing bij een grote hoeveelheid schijven een groot verschil maakt.

Een ander voordeel van SATA is dat het hot-pluggable is. Dit betekent dat de schijf vervangen kan worden terwijl het computersysteem in werking is. Dit kan bijzonder bruikbaar zijn in RAID configuraties.

## NCQ

SATA ondersteunt ook Native Command Queueing (NCQ). Als een schijf verschillend lees- of schrijfcommando's toegestuurd krijgt, zullen die meestal doorgaan op verschillende locaties op de schijf. NCQ kan rekening houden met de locatie van de gegevens op de schijf om de gegevens zo snel mogelijk beschikbaar te maken. NCQ vraagt natuurlijk wel enige tijd om te bekijken in welke volgorde de commando's best verwerkt kunnen worden en er moeten natuurlijk ook meerdere commando's beschikbaar zijn. NCQ is dan ook vooral handig bij zwaarder belaste schijven (bijvoorbeeld file server).

### 7.3.3. eSATA

External SATA is een SATA-variant die de aansluiting van externe schijven mogelijk moet maken. Er zijn kleine verschillen met SATA, onder andere op het vlak van de signaleering, zodat langere kabels gebruikt kunnen worden.

eSATA treedt op het vlak van de externe opslag in concurrentie met USB, en zou dus eigenlijk even goed in het volgende hoofdstuk thuis horen, met de interfacebussen. Ten opzichte van deze laatste twee heeft eSATA als voordeel dat het hogere bitsnelheden kan halen. Het belangrijkste nadeel is dat de andere twee veel beter ingeburgerd zijn. Een van de voordelen (of beter doelstellingen bij het ontwerp) van USB was het uniformiseren van de aansluitingen op een computer: een connector voor allerlei soorten apparaten. Ondertussen is de specificatie van USB 3 volledig af en wijd verspreid, wat het voor eSATA waarschijnlijk nog moeilijker zal maken om aan populariteit te winnen...

### 7.3.4. Serial Attached SCSI (SAS)

Ondertussen zal wel duidelijk zijn dat naarmate de datasnelheid echt groot moet zijn, seriële transfers de voorkeur krijgen op parallelle (toch als een zekere afstand overbrugd moet worden). Dit komt ook naar voor als je de geschiedenis van SAS bekijkt: deze komt voort uit de SCSI standaard, die parallelle communicatie gebruikte.

Vanzelfsprekend is er dus ook een seriële SCSI variant opgedoken, met name SAS (Serial Attached SCSI). SAS lijkt heel hard op SATA. Het is eveneens een seriële punt-tot-punt verbinding die gelijkaardige snelheden haalt. Het belangrijkste verschil is dat het zich net als het verouderde SCSI meer richt op de servermarkt, waarvoor het een iets grotere betrouwbaarheid biedt. Deze betrouwbaarheid uit zich vooral in een aantal foutcorrectie- en rapporteringmechanismen die in de loop der tijden binnen SCSI ontwikkeld zijn. Deze gaan een stuk verder dan wat met S.M.A.R.T. mogelijk is op ATA (en dus ook SATA).

Daar staat dan weer tegenover dat SAS iets duurder is dan SATA.

---

# Chapter 8. Externe I/O

In dit hoofdstuk komen een aantal veel voorkomende interfaces voor randapparatuur voor. Een aantal hiervan worden uitgebreider besproken in de cursus “interfacing & microcontrollers”.

## 8.1. In den beginne

Dit hoofdstuk heeft niet de pretentie om alle mogelijke interfacebussen op te lijsten. Een aantal van die bussen worden niet meer gebruikt of raken stilaan in onbruik. De klassieke printerpoort of Centronics poort is er zo eenntje. Deze parallelle poort zoals ze ook genoemd werd, is primair ontworpen voor het aansturen van printers. Getuige daarvan zijn de specifieke controlelijnen om aan te geven als bijvoorbeeld het papier uitgeput is. Deze poort werd later ook nog veelvuldig gebruikt, omdat ze erg eenvoudig is van opbouw, en daardoor makkelijk te gebruiken voor kleinere elektronica projectjes. Tegenwoordig is deze poort slechts zelden nog te vinden op een pc.

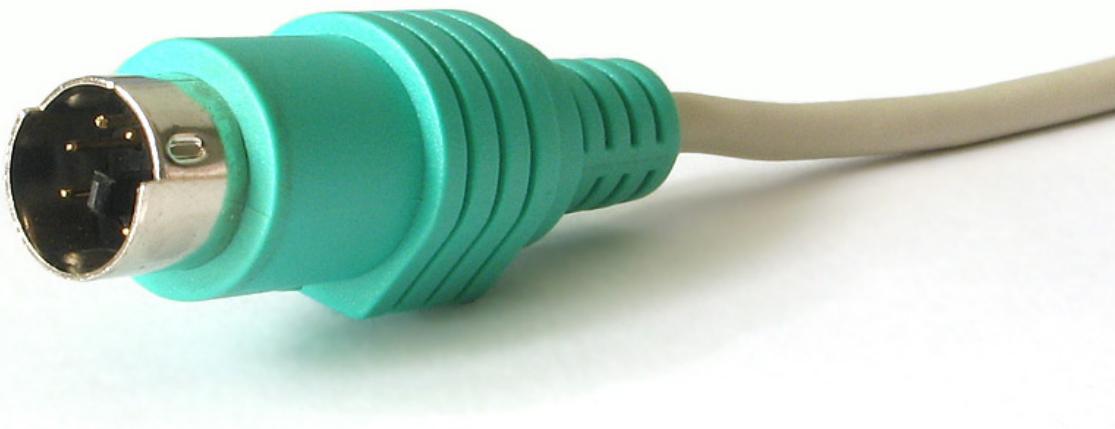
Ook de seriële poort is stilaan in z'n bestaan bedreigd. Enkel voor heel specifieke toepassingen (vb configuratie van apparatuur als routers) wordt ze nog gebruikt, maar ook daar moet ze stilaan plaats ruimen voor USB.

De COM poort (zoals de seriële poort ook genoemd werd) laat asynchrone seriële communicatie toe. Dit betekent relatief lage bitsnelheden (minimum 50bps, standaard 9600bps, maximum 115kbps). Deze vorm van communicatie wordt gebruikt voor tragere I/O, zoals barcodescanners, modems, ...

Omwille van de eenvoud en de ondersteuning op tal van microcontrollers wordt ze dikwijls gebruikt bij embedded systemen.

Firewire is een andere IO mogelijkheid die vooral door Apple gepromoot werd, maar die stilaan in onbruik geraakt is.

## 8.2. PS/2



De PS/2 interface duikt in de PC wereld op met de introductie van de IBM PS/2 (Personal System 2), ondertussen enkele decennia geleden. Het is een interface die dient voor het aansluiten van toetsenborden en muizen. Er bestaan twee connectoren die identiek zijn, op hun kleur na. De paarse connector dient voor een toetsenbord, de groene voor een muis. Aangezien beiden gebruik maken van een ander protocol is het aangewezen om de kleurcode te respecteren. Met name het BIOS kan problemen maken van een toetsenbord dat op de verkeerde poort is aangesloten, zodat bij het opstarten geen opdrachten doorgegeven kunnen worden via het toetsenbord. Een tweede mogelijk verrassend feit, is dat PS/2 niet hot pluggable is.

Dit komt door het elektronisch ontwerp. De pinnen van de connector zijn rechtstreeks verbonden met de pinnen van de microcontroller die beschadigd kan worden door hot swapping. Anderzijds is er ook geen ondersteuning voor het herkennen van een nieuw aangesloten apparaat. In moderne interfaces worden de contacten naar de microcontroller robuuster uitgevoerd, zodat er minder kans op beschadiging is. Bij het gebruik van standaardapparaten is de herkenning van het nieuw aangesloten apparaat ook geen probleem, zodat hot swapping soms wel succesvol aflopen, maar absoluut niet aan te raden is.

PS/2 is ondertussen technologisch al lang achterhaald, maar blijft desondanks (backward compatibility) toch aanwezig aan de achterkant van vele PC's en servers. De opvolger van PS/2 is USB, dat tevens een aantal andere interfaces moet vervangen, met als belangrijke pluspunt een uniforme aansluiting voor alle soorten apparaten.

## 8.3. USB

### 8.3.1. Overzicht

Om een oplossing te vinden voor de problemen bij het gebruik en vooral de configuratie van de klassieke (legacy) I/O-interfaces zoals de Centronics Interface en RS232, werd onder andere onder impuls van Intel een werkgroep opgericht met als doel het ontwikkelen van een nieuwe interface standaard. Dit werd de Universal Serial Bus (USB). USB is een snelle en flexibele interface om randapparatuur te verbinden met een computer. Ze is ontworpen met als doel het gebruiksgemak en betrouwbaarheid te verhogen.

Doelstellingen:

- eenvoudige en betrouwbare aansluiting geschikt voor alle soorten randapparatuur
- immuun voor problemen met systeemresources als interrupt- of DMA-conflict
- automatische detectie en configuratie van aangesloten apparaten
- goedkoop te realiseren
- laag stroomverbruik
- hot-pluggable

In de originele specificatie (USB 1.1) is er sprake van twee mogelijke transmissiesnelheden:

- **Low Speed:** 1,5 Mbps, bedoeld voor I/O met een beperkt datadebiet: muis, toetsenbord, joystick, ...
- **Full Speed:** 12 Mbps, bedoeld voor hogere debieten: audio, printers, scanners, processorcommunicatie... In een latere revisie (USB 2.0) is er een derde mogelijkheid bijgekomen:
- **High Speed:** 480 Mbps, bedoeld voor video, externe harde schijven, DVD of CD-romspelers, ... De nieuwste revisie (USB 3.0) krikt de snelheid verder op:
- **Superspeed:** 4,8 Gbps, bedoeld om extra hoge snelheden te halen, zoals voor SSD. De nieuwe mogelijkheden van deze versie van de standaard komen op het eind van dit hoofdstuk aan bod. De USB-bus is gebaseerd op het master/slave principe: de host (de computer) is verantwoordelijk voor het beheer en de

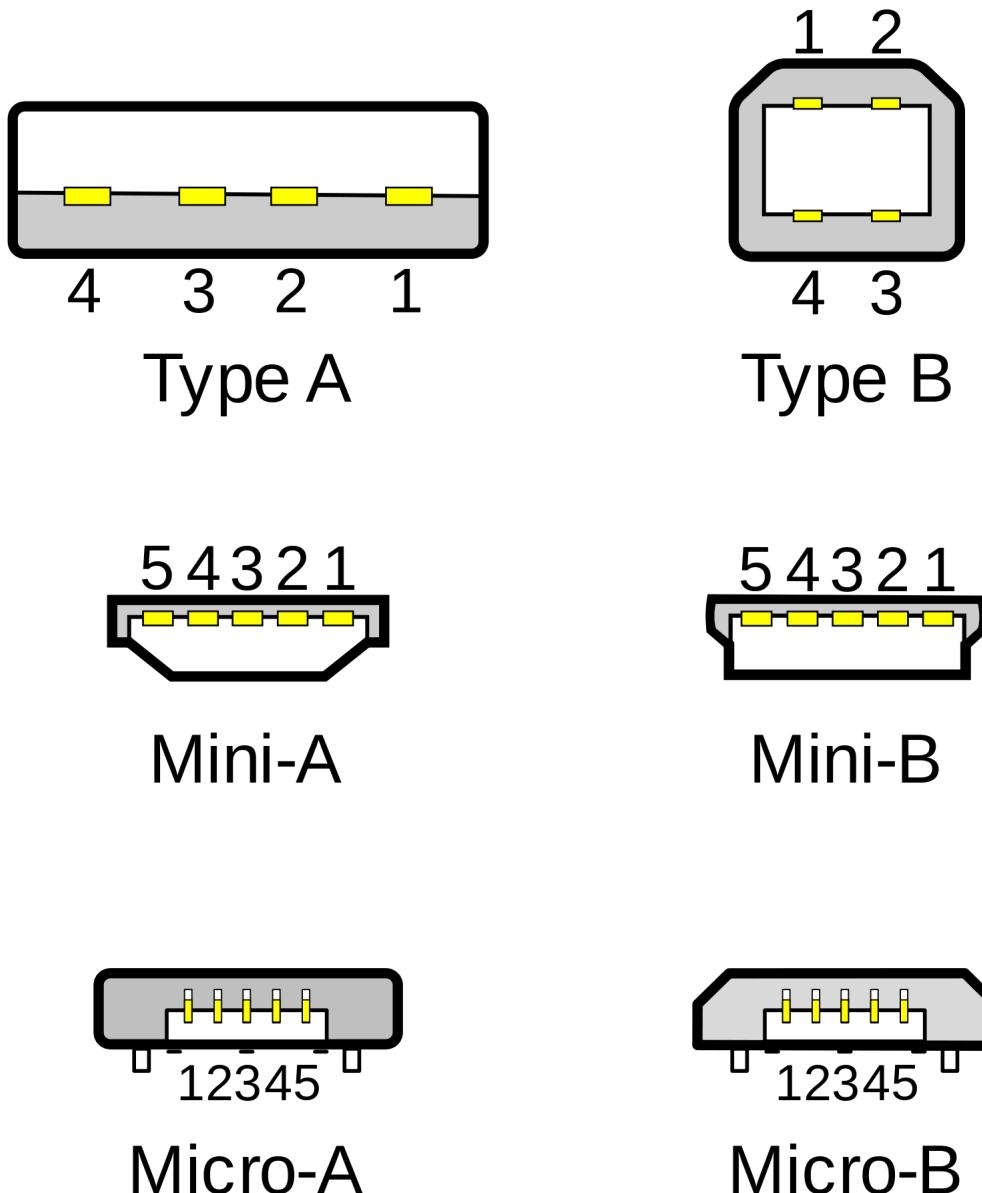
coördinatie van de activiteit op de bus. Er kan dus enkel communicatie gebeuren op initiatief van de host. Deze bevat hiervoor een zogenaamde USB Host-controller.

### 8.3.2. Mechanische en elektrische opbouw

De USB-bus werkt met relatief korte kabels (maximum 5 m.) voor punt tot punt verbindingen: er wordt telkens een apparaat verbonden met een poort van een USB-hub. Ook de aansluitingen op een computer maken deel uit van één of (soms) meerdere hubs. Een USB2-kabel bevat vier geleiders:

- Rood: +5V
- Zwart: Gnd
- Wit: Data
- Groen: Data+

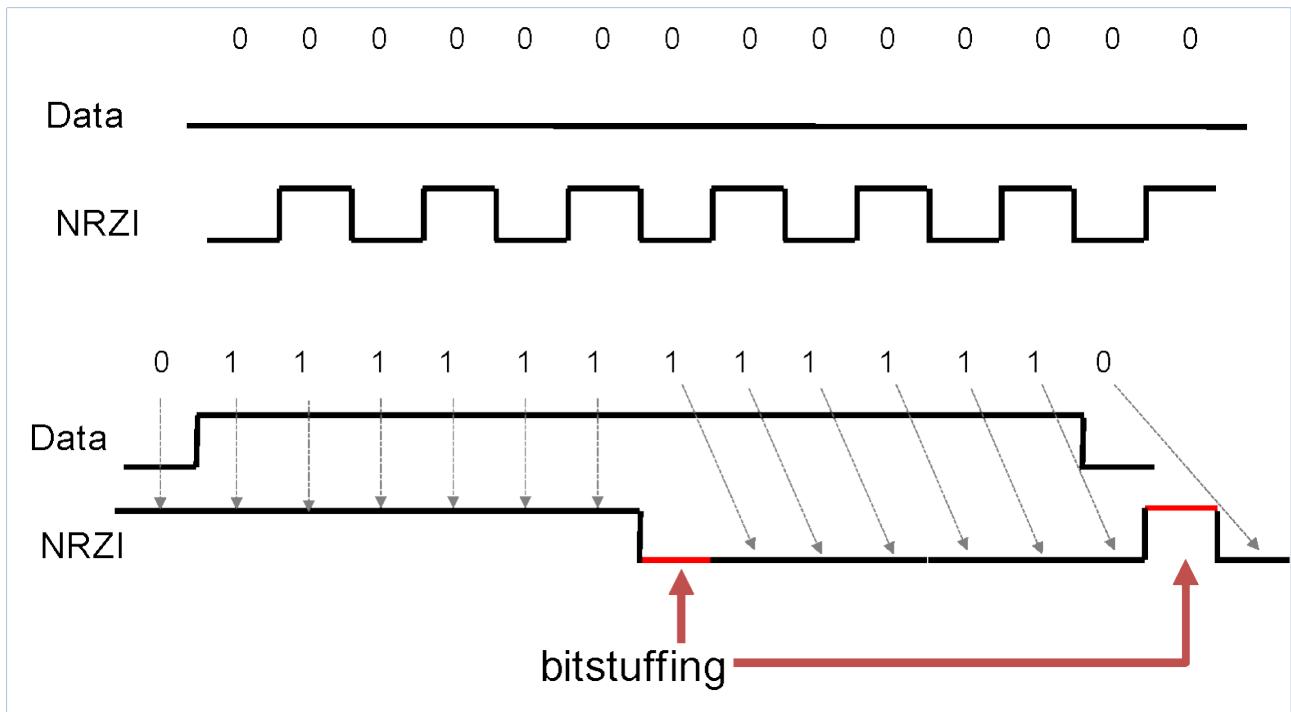
In het geval van Full- of High Speed USB heeft het datapaar een afzonderlijke afscherming. Om grotere afstanden te overbruggen of meer apparaten aan te sluiten kan de installatie uitgebreid worden door gebruik te maken van externe USB-Hubs. In totaal mogen er maximaal 4 externe hubs achter elkaar geschakeld worden (geeft een maximum afstand van 30 m) en mogen er maximaal 127 apparaten aangesloten worden op een USB-bus. Er zijn twee types connectoren. Type A wordt gebruikt voor de aansluiting aan de hub (of op de computer). Type B wordt gebruikt voor de aansluiting. Er bestaan ook mini en micro varianten van de connectoren. Volgende figuur toont een aantal voorbeelden.



Als een randapparaat geen al te groot stroomverbruik heeft, kan het gevoed worden vanuit de USB-kabel: stromen tot 500 mA zijn toegestaan bij een spanning van 5V (vanwege spanningsverlies over de kabel kan deze spanning zakken tot minimaal 4 V bij het randapparaat). Voor grotere vermogens moet het randapparaat een eigen voeding voorzien.

De USB-bus verstuurde data serieel met NRZI-codering (Non Return on Zero Inverted) via een differentieel signaal (D+, D-). Een overgang in het datasignaal stelt een 0 voor, terwijl een ongewijzigd signaal een 1 voorstelt.

Opdat de ontvanger synchroon zou blijven lopen met de zender (er is geen afzonderlijk kloksignaal), is het nodig dat er regelmatig een overgang in het signaal optreedt. In het geval dat er meer dan 6 opeenvolgende bits op 1 staan, wordt na het 6e bit automatisch een 0 ingevoegd (bit-stuffing) door de zender. Dat extra 0-bit moet door de ontvanger uiteraard weer verwijderd worden. Op deze manier wordt een betrouwbare gegevensoverdracht verzorgd, die nog aangevuld wordt met een CRC-foutcontrole per pakket.



De communicatie in de twee richtingen (host->device) gebeurt over hetzelfde draadpaar. Dit is dus vergelijkbaar met half-duplex.

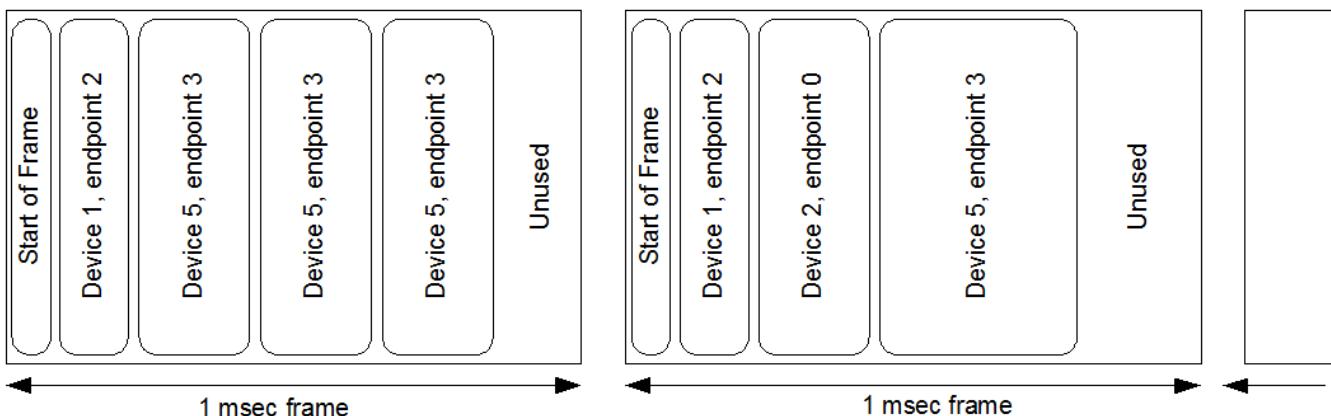
### 8.3.3. Communicatie over USB

Zoals de naam al aangeeft, wordt het medium bij de Universal Serial Bus gedeeld over alle aangesloten apparaten (shared bandwidth). Daarnaast wordt er gebruik gemaakt van het master/slave principe zodat de host-controller(de master) de volledige controle heeft over de communicatie via de USB-bus.

#### Frames, pakketten

De host verdeelt de tijd in frames met een vaste lengte (1 msec voor Low- en Full-speed, 125 µsec voor High-speed). Elk frame wordt op zijn beurt opgedeeld in een aantal pakketten (lengte kan variëren) waarvan het eerste een SOF (Start-Of-Frame)

een pakket is, bedoeld voor frame synchronisatie tussen de verschillende apparaten. Elk pakket is gekoppeld aan een apparaat en daarbinnen aan een endpoint (vergelijkbaar met een TCP-poortnummer).



Binnen een pakket vind je enkele belangrijke velden:

#### **De Sync-vlag (00000001)**

is bedoeld voor bit- en byte-synchronisatie.

#### **De Packet-ID (of PID)**

geeft het type van het pakket weer (Token, Data, Handshake, Special, ...).

#### **Het Data-veld**

bevat naast eventuele gegevens ook het adres van het apparaat waaraan het pakket gekoppeld is en het endpoint.

Gedurende het EOP-gedeelte worden de twee datalijnen beide laag gehouden gedurende een bittijd. Hierdoor detecteren alle aangesloten apparaten het einde van het pakket. Een volledige transfer tussen de host en een apparaat zal opgebouwd worden door verschillende pakketten die eventueel in verschillende frames kunnen zitten.

Er onderscheiden zich vier types van pakketten.

- Start-of-frame: geeft het begin aan van een nieuw frame inclusief een 11 bit frame nummer.
- Token: het token geeft de richting aan van de pakketten.

```
# IN:: informeert het toestel dat de host iets wil ontvangen
# OUT:: informeert het toestel dat de host iets wil versturen
# SETUP:: wordt gebruikt om een transfer te starten.
```

- Handshake

- # ACK:: geeft een bevestiging dat de info correct werd ontvangen
  - # NAK:: geeft aan dat het toestel eventjes geen data kan zenden of ontvangen, of dat er geen data beschikbaar is op dit moment.
  - # STALL:: het toestel bevindt zich in een staat waarin de host actie moet ondernemen.
- Data: bevat maximum 1024 bytes aan data. Er zijn ook verschillende subtypes:
    - # Data0
    - # Data1
    - # Data2 (enkel bij high-speed)
    - # MDATA (enkel bij high-speed)

Aanvullende uitleg en verdere verduidelijking vind je op <http://www.beyondlogic.org/usbnutshell/usb3.shtml#USBPacketTypes>

### 8.3.4. Endpoints

Voor de communicatie met de host, beschikt elk USB-device over een aantal endpoints (maximaal 16): dit zijn meestal kleine databuffers in het geheugen van de microcontroller van het randapparaat. Die buffers worden gebruikt om de informatie uit een pakket op te slaan.

Om gegevens uit te wisselen tussen de host en een USB-apparaat wordt een verbinding opgezet (pipe genaamd) tussen een endpoint en de host. De richting wordt gespecificeerd vanuit het standpunt van de host: Een IN-endpoint stuurt data van het apparaat naar de computer. Bij een OUT-endpoint ontvangt het apparaat data van de computer.

Een uitzondering op die regel: endpoint 0 is bedoeld voor controle van de verbinding en laat communicatie in de twee richtingen toe (is eigenlijk een combinatie van een IN- en een OUT-endpoint, beide met hetzelfde nummer).

images:endpoint.gif[align="center", scaledwidth="40", alt="USB endpoints (bron: beyondlogic.org", width="500"]

### 8.3.5. Types van transfers

USB is ontworpen om verschillende soorten apparaten te bedienen, elk met zijn eigen specificaties (gegevensdebit, reactietijd, foutcorrectie,...). Om aan alle mogelijke situaties tegemoet te komen zijn er vier soorten transfers voorzien:

### **CONTROL transfers**

worden gebruikt voor het beheer van de USB. Via control transfers over de control-pipe (naar endpoint 0) leest de host informatie over een apparaat en ontvangt het apparaat configuratie-informatie (zoals een device-adres). Elk USB-apparaat moet controltransfers ondersteunen.

### **BULK transfers**

zijn bedoeld voor situaties waar veel gegevens moeten doorgestuurd worden, maar waarbij de snelheid niet kritisch is: disk, printers, scanners,... Als de USB bezet is met verkeer dat een gegarandeerde bandbreedte vereist (isochrone transfers), dan zal een bulktransfer moeten wachten. Wanneer er daarbuiten weinig verkeer is, zal een bulk-transfer zeer snel verlopen.

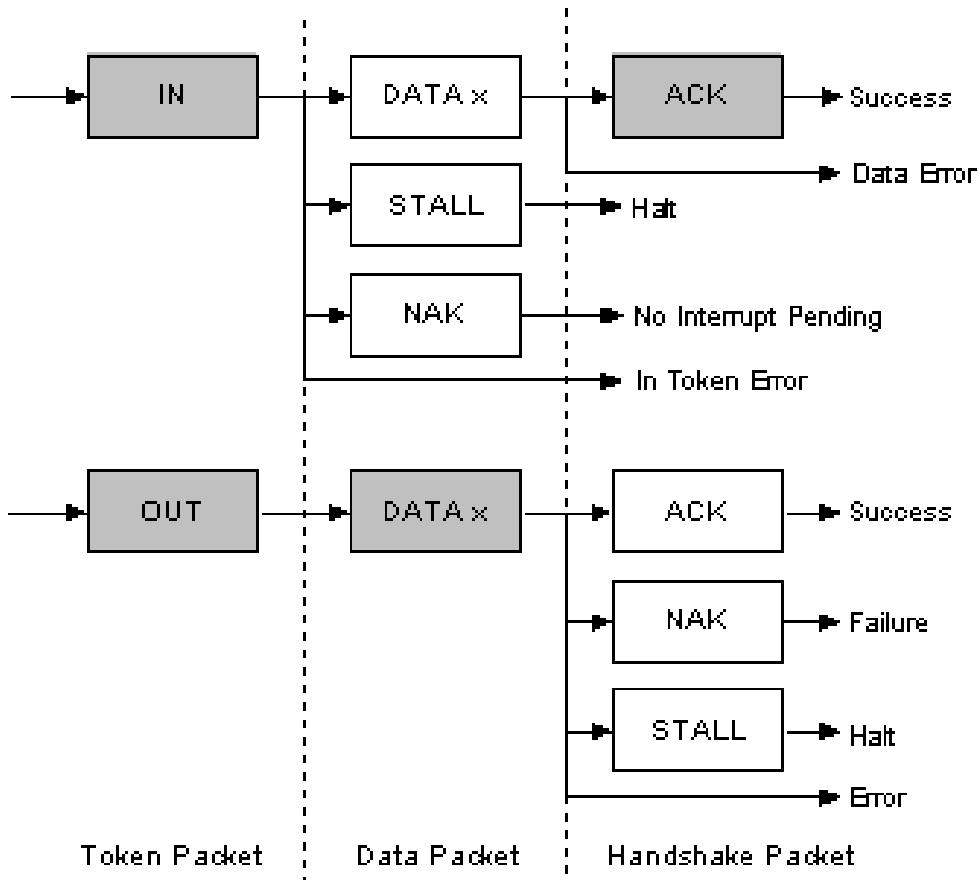
### **INTERRUPT transfers**

worden gebruikt wanneer een apparaat op regelmatige tijdstippen de gelegenheid moet krijgen om gegevens te verzenden of ontvangen. Voorbeelden hiervan zijn een toetsenbord, een joystick, een muis: hierbij wordt een limiet vereist op de reactietijd van de computer. In tegenstelling tot wat je uit de naam zou verwachten, is een interrupttransfer gebaseerd op regelmatige polling door de host-controller. Vermits USB een puur master/slave systeem is, zijn er geen interrupt-mogelijkheden voor de USB-apparaten.

### **ISOCHRONIC transfers**

garanderen een gevraagde bandbreedte, maar voorzien geen foutcorrectie. Wanneer geluid of video over USB wordt doorgestuurd, zal men gebruik maken van isochrone transfers. Hierbij zorgt de host-controller ervoor dat alle toegestane transfers goed verlopen. Wanneer software een aanvraag doet om bijvoorbeeld een isochrone transfer op te zetten met een bepaald apparaat, wordt eerst nagegaan of er nog voldoende bandbreedte over is op de bus (of er in elk frame nog voldoende plaats vrij is). Voldoende plaats betekent dat steeds voldoende ruimte moet overblijven voor noodzakelijke controle transfers. In een frame komen eerst de isochrone transfers aan bod. Vervolgens worden de interrupt apparaten gepold (en wordt eventueel de nodige interrupt data getransfereerd). Op deze manier komen alle apparaten die een bepaalde bandbreedte of een bepaalde korte reactietijd nodig hebben steeds aan bod. Wat overblijft van de tijd in een frame, kan ingenomen worden door bulktransfers. Dit is minimaal de nog niet toegekende tijd, maar kan ook niet gebruikte tijd voor controle of interrupt transfers zijn.

## Verloop van een transfer.

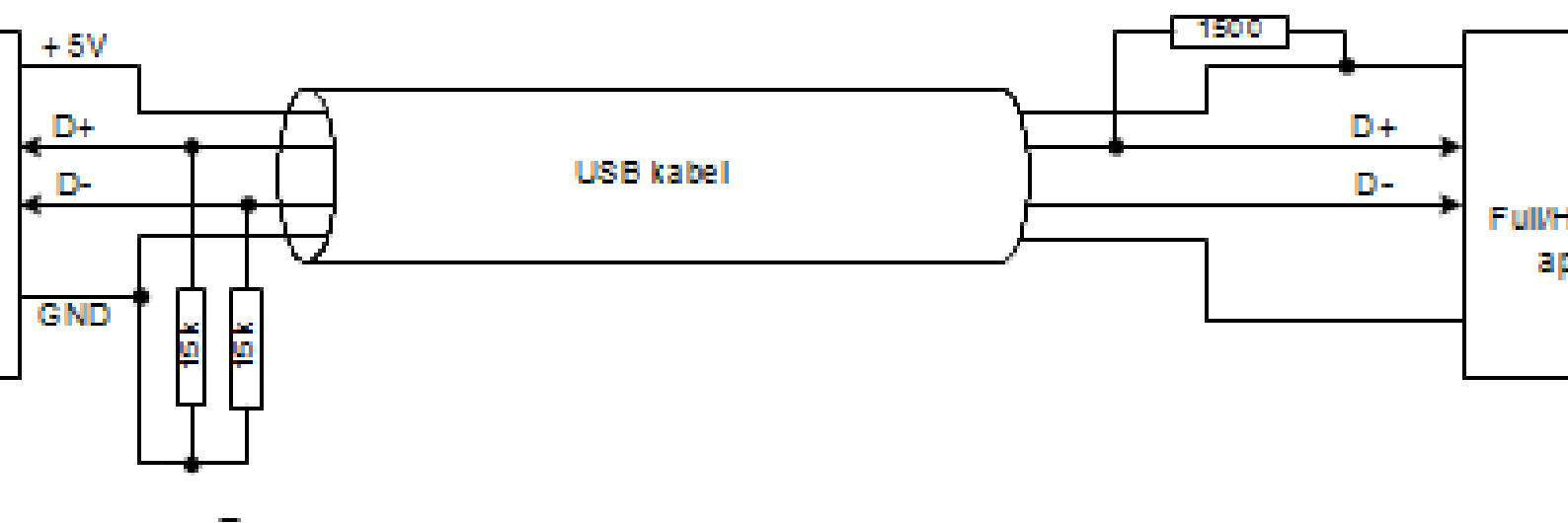


De transfer begint met de host controller die aangeeft in welke richting de data zal gaan: in-token voor data van apparaat naar host controller, out-token voor de omgekeerde inrichting. Vervolgens wordt de data verzonden (afhankelijk van de richting gebeurt dit door het apparaat of de host controller). In het geval van een transfer richting host controller, kan het apparaat ook reageren met een NAK. Dit betekent dat er geen data te versturen is. Geen reactie op het token bericht betekent dat dit token een fout bevatte. Na de data volgt nog een bevestiging, die aangeeft dat de data correct ontvangen is. In het geval van een transfer richting apparaat kan dit ook bevestigen met een NAK. Dit betekent dat het apparaat de inkomende data op dat ogenblik niet kan verwerken. Andere transfers verlopen op gelijkaardige manier.

Bij isochrone transfers zullen er geen bevestigingen zijn, terwijl na een controle transfer nog een extra statusfase zal volgen waarbij de verwerking van een commando wordt bevestigd.

### 8.3.6. Enumeratie en P&P

Zoals bij de inleiding al vermeld, is USB ontworpen met het oog op een volledige integratie met het Plug&Play gebeuren. De bedoeling is dan ook dat een gebruiker een USB-apparaat aansluit, dat het automatisch herkend wordt door het besturingssysteem, dat de juiste drivers geladen worden waarna het apparaat gebruikt kan worden, zonder dat die gebruiker iets moet configureren. Om dit mogelijk te maken zijn er wel een aantal stappen nodig: detectie van nieuwe apparatuur en snelheid daarvan.



Langs de kant van de host (of hub) zijn de twee datalijnen via 2 pull-down weerstanden (15 k) verbonden met de massa. Daardoor ziet de host een laag niveau op beide datalijnen (D+ en D-).

Een Low-speed device heeft van zijn kant een pull-up weerstand van 1,5k tussen de voedingsspanning en D- (bij de full- en high-speed apparaten is dat naar de D+ lijn). Wanneer een apparaat aangesloten wordt, zal langs de kant van de host (hub) een van beide D-lijnen naar een hoog niveau getrokken worden. Hierdoor detecteert deze dat er een nieuw apparaat op de bus aanwezig is en ook of dit al dan niet een low-speed device is. Elke hub heeft een interrupt IN-pipe waardoor de nieuwe aansluiting doorgegeven wordt naar de host.

### Enumeratie

Bij het opstarten van de computer en telkens er een nieuw apparaat gedetecteerd wordt, zal er een procedure doorlopen worden: de zogenaamde enumeratie. Hierbij is het de bedoeling dat het apparaat een uniek adres toegewezen krijgt, dat het besturingssysteem informatie opvraagt over het apparaat en dat het weet welke drivers er eventueel geladen moeten worden.

Wanneer een nieuw device gedetecteerd wordt (Windows XP/Vista geeft hiervan een melding in de system-tray), zal de host een aanvraag voor informatie sturen naar endpoint 0 (de controle-endpoint) van device-adres 0 (geen enkel actief apparaat heeft adres 0). Die informatie bestaat uit een aantal descriptors: records met vooraf bepaalde velden die alle nodige informatie bevatten die de host-controller nodig heeft. In een volgende stap kent de host een USB-adres toe aan het nieuwe apparaat (verschillend van 0), waarna nog wat informatie uitgewisseld wordt tussen host en apparaat.

In de ontvangen device-descriptor staat onder andere een veld dat de fabrikant van het apparaat identificeert (zoals bij een netwerkkaart, aan te vragen bij het USB-consortium) en een veld dat door die fabrikant toegekend wordt om het model van het apparaat te identificeren. Bij die configuratie worden ook andere configuratieparameters (configuration-descriptor) uitgewisseld waarin onder meer bepaald wordt hoeveel stroom het apparaat nodig zal hebben.

Daarna zullen ook nog interface descriptors uitgewisseld worden. Deze bevatten informatie over het aantal endpoints dat het device nodig heeft, en welke device classes ondersteund worden.

Tot slot worden ook nog endpoint descriptors doorgestuurd. Daarin staat per endpoint beschreven hoe het verkeer zal georganiseerd worden (bulk, interrupt, ...), hoe vaak er moet gepold worden in het geval van interrupt transfer, de richting van de endpoint (IN/OUT), ...

Op basis van die Vendor-en Product- ID's en de informatie die in de descriptors gevonden wordt, zal het besturingssysteem in de beschikbare INF-bestanden zoeken naar een geschikte driver. Als die niet gevonden wordt zal bijvoorbeeld Windows 8 online op zoek gaan. De pas geladen driver zal bij zijn initialisatie zelf het USB-apparaat (of beter de controller daarin) verder configureren, waarna het toestel gebruikt kan worden.

### 8.3.7. Device Classes, Drivers

Bij het overlopen van alle mogelijke randapparaten voor een computer kunnen die in een aantal grote groepen ingedeeld worden. Bij het ontwerp van de USB-standaard zijn een aantal device-classes en subclasses gedefinieerd. Elk toestel moet bij de enumeratie in de device-descriptor ook meegeven tot welke groep het behoort. Voor elke vastgelegde class en subclass is ook de structuur van de driver vastgelegd samen met de vereisten voor de firmware van het apparaat zelf (een soort API).

Enkele voorbeelden van classes:

- Audio devices
- Mass Storage devices

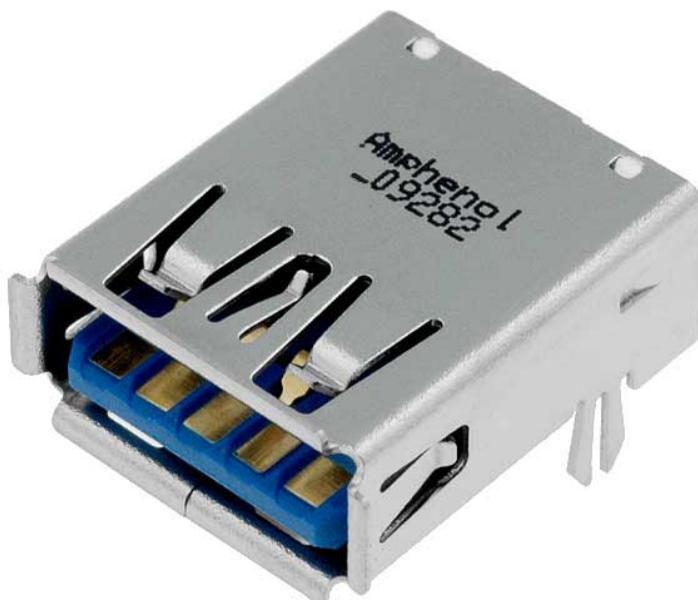
- Monitors
- Communication devices (modems)
- Printers Still image capture devices (scanner, camera,...)
- HID (Human interface devices)

Een modern besturingssysteem bevat drivers voor een aantal veel voorkomende classes. Voor apparaten uit die groep hoeft de fabrikant geen eigen drivers meer te schrijven (tenzij er extra functionaliteit gevraagd wordt).

Een belangrijke groep is de HID-class: deze is origineel bedoeld voor I/O apparatuur als toetsenbord, muis en dergelijke. De vereisten voor deze toestellen zijn zeer beperkt, zodat het vrij gemakkelijk is om een toestel (en de bijhorende firmware) te ontwerpen dat binnen deze groep past, met als voordeel dat daarvoor geen drivers geschreven hoeven te worden (ze zijn immers al binnen het besturingssysteem aanwezig).

### 8.3.8. USB 3.0

De USB standaard versie 3 verzekert de toekomst van deze technologie. Om deze hoge snelheden te kunnen halen, waren wel enkele compromissen nodig. Zo was dezelfde connector niet langer bruikbaar. Alles is gelukkig backwards compatibel, maar een USB3 connector zal je makkelijk kunnen herkennen aan de blauwe kleur van de binnenkant van de connector.



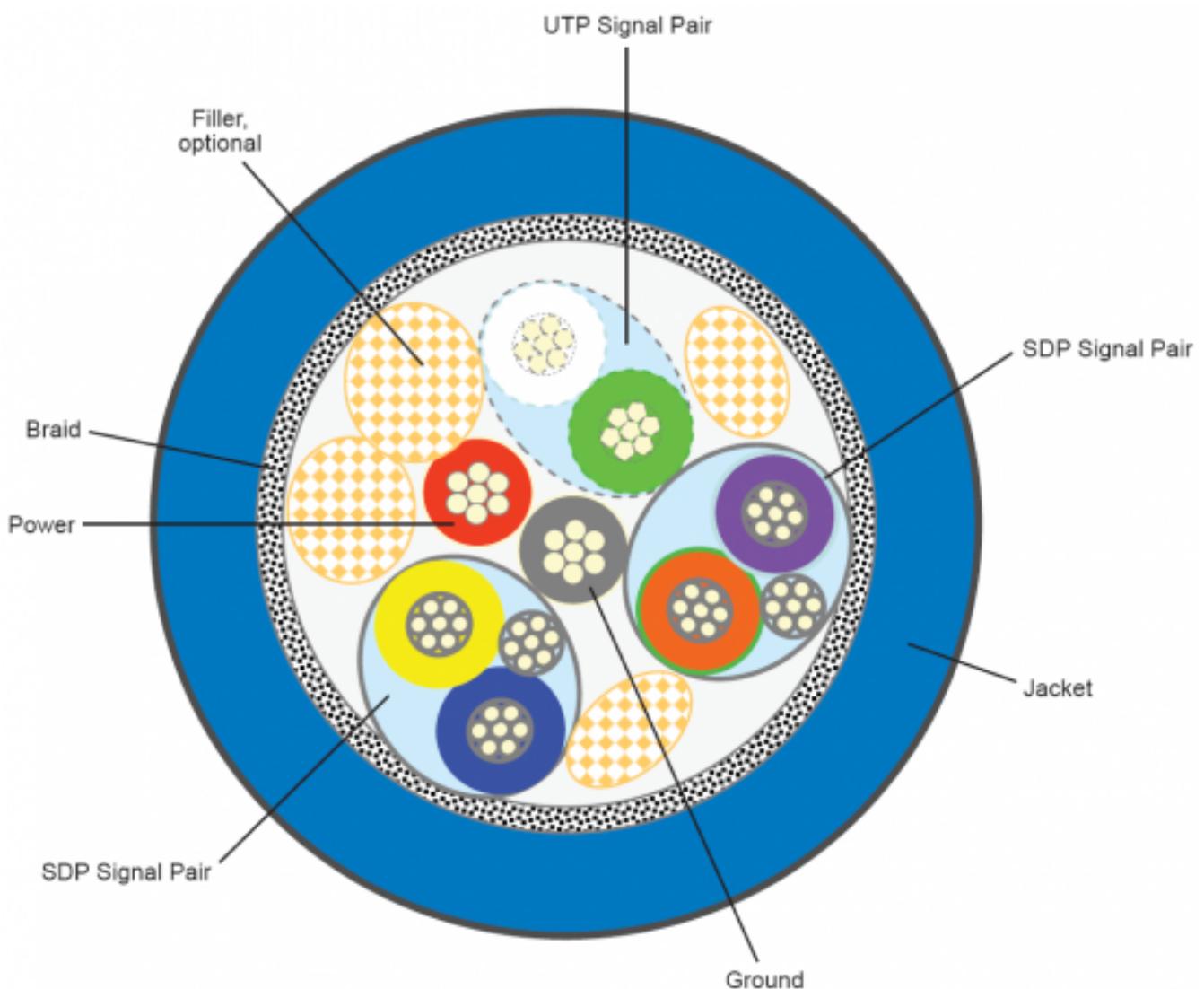
Ondanks de fysieke compatibiliteit zijn er een flink aantal verschillen te bespeuren in de nieuwe versie. Kijk je bijvoorbeeld naar een dwarsdoorsnede van een USB3.0-kabel, dan merk je een groot verschil. Het data-paar van USB2.0 is nog steeds aanwezig,

maar is enkel daar om compatibiliteit te garanderen. De snelle data zal verplaatst worden over twee Shielded Differential Pairs.

Deze geïsoleerde geleiders zorgen er uiteraard voor dat deze kabels een stuk minder soepel zullen zijn dan hun voorgangers.

Een ander interessant detail is dat deze standaard voorziet in een grotere stroomlevering aan apparaten: 900mA ten opzichte van 500mA bij de huidige (2.0) standaard.

De encoding gebeurt ook niet langer met NRZI, maar met 8b/10b, de methode die ook gebruikt wordt bij SATA, SAS, PCI Express.

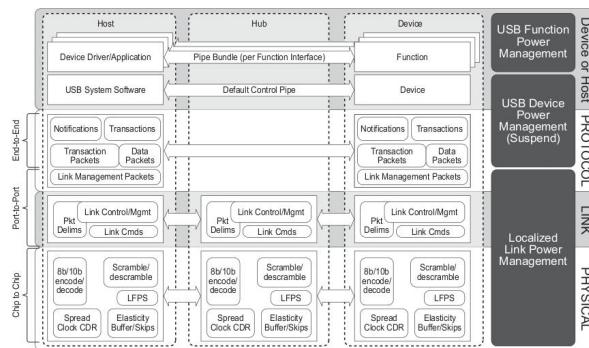


## Signaleren

Ook de signalering is bij USB 3.0 op de schop genomen. Het typerende polling-concept is niet meer zo sterk aanwezig. Dat geeft voordelen naar zuinigheid van de apparaten. Ze moeten immers niet constant actief zijn.

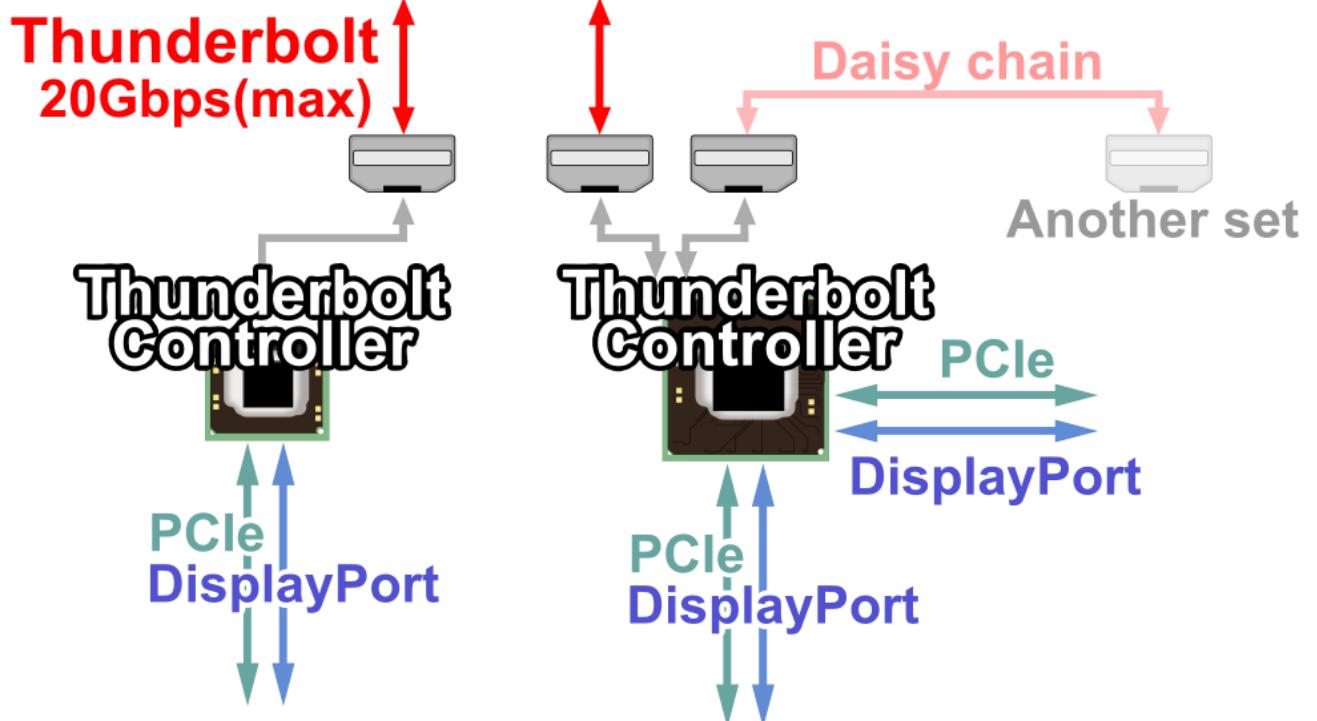
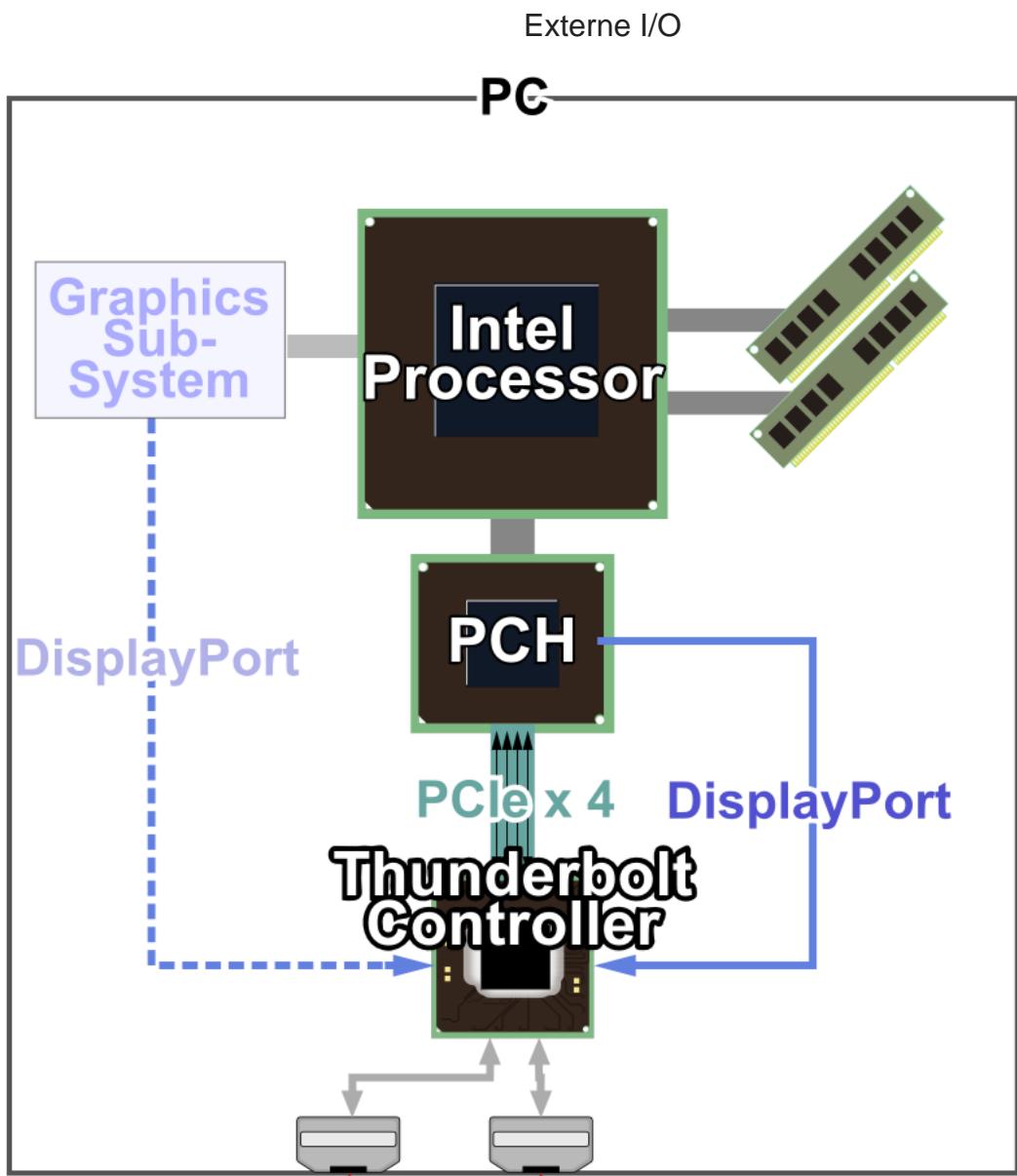
Andere significante wijzigingen:

- verkeer is nu full duplex ten opzichte van half-duplex in USB 1 en 2.
- de protocolstack is een stuk complexer
- bulk streams is een nieuwe, vijfde transfermethode
- toevoegen van een link layer
- fel verbeterd power management
- door complexiteit is de maximale kabellengte beperkt tot ongeveer 3 meter.



## 8.4. Thunderbolt

De Thunderbolt interface is te vinden op alle nieuwe MAC-hardware, en heeft z'n reden. De standaard werd ontwikkeld door Intel in samenwerking met Apple. De eerste apparaten die hiermee werden uitgerust, werden uitgebracht in februari 2011 (Macbook Pro). Mondjesmaat wordt extra randapparatuur uitgebracht die dit ondersteunt. De standaard probeert nog universeler te zijn dan USB door de kracht van displayport (zie verder) en PCIx te combineren in één enkele poort.



Thunderbolt is in staat om 10Gpbs te transfereren over een koperverbinding. Initieel had Intel de bedoeling om de bekabeling met glasvezels te produceren, maar voorlopig is dat niet het geval. De bouw ervan was te duur, en een bijkomend probleem was dat op die manier geen stroomvoorziening kan gebeuren voor de randapparaten. Er wordt echter nog steeds ontwikkeld, dus misschien duikt dit alsnog op in een volgende versie. De architectuur van Thunderbolt zet alles in op performantie. Daarvoor wordt de PCH via een PCIe 4x poort verbonden met de Thunderbolt controller, en is die laatste ook verbonden met het interne grafisch systeem van de computer. Het resultaat is dat over deze verbinding een PCIx verbinding en een Displayport gebundeld zijn. Daarnaast ondersteunt Thunderbolt ook daisy chaining tot maximaal zeven apparaten. Dat betekent dat je Thunderbolt-compatibele apparaten kan doorlussen.

Een goed voorbeeld van hoe dit gebruikt wordt, vind je bijvoorbeeld in het Apple Thunderbolt Display. Dat is een scherm dat bedoeld is voor de apple macbook pro's als tweede scherm. Met een enkele Thunderbolt kabel naar dat scherm wordt echter veel mogelijk:

- een 27" scherm met 2560-by-1440 resolutie krijgt z'n beelden door deze kabel
- er is een camera ingebouwd
- er is HD audio voorzien
- er is een connector voor Gigabit Ethernet voorzien
- er is een Firewire 800 connector voorzien.

Dat alles gebeurt met deze enkele Thunderbolt kabel.



Criticasters hekelen de veiligheid van Thunderbolt, net omdat je rechtstreeks verbonden bent als een PCIx poort, en je dus ook toegang hebt tot het geheugen van het host-apparaat. Bedenk zelf de risico's die daarmee gepaard gaan...

---

## Chapter 9. Literatuurlijst

Onderstaande werken bevatten een stuk diepgaandere informatie over deze materie. Ze kunnen dan ook een ideaal vertrekpunt vormen voor een verdere studie van de computerarchitectuur.

[STALLINGS] Andy Hunt & Dave Thomas. 'Computer Organization and architecture, Ninth edition'. Pearson education. 2012.

[RAMA] Umakishura Ramachandran. 'Computer Systems: An Integrated Approach to Architecture and Operating Systems'. Manning Publications. 2010.