

Computersystemen 2

Theorie

0. Intro

De Docent

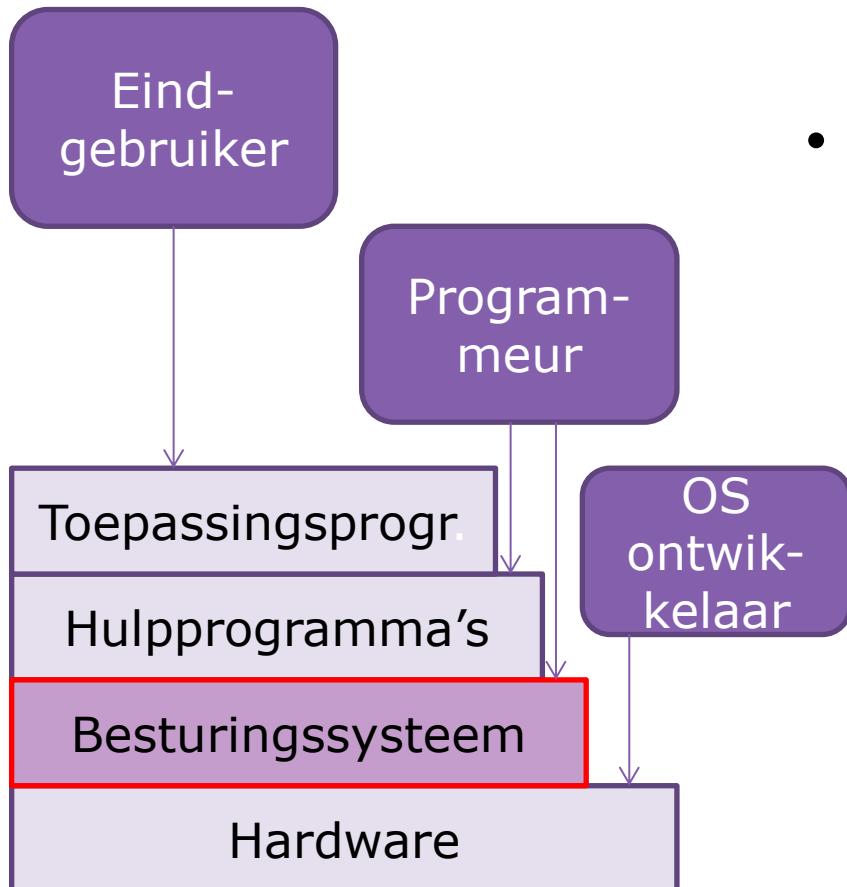
- Geert De Paepe
(geert.depaepe@kdg.be)
 - docent
 - Computersystemen 2: Theorie deel
 - Computersystemen 2 – ISB
 - Data Science 2
 - Integratieproject 2 – ISB
 - The Lab
 - Stage
 - TI Flex Computersystems 2



Overzicht vak

- deel van Computersystemen 2
 - (30%) Theorie: nadruk op kennis van interne werking van besturingssystemen
 - (70%) Praktijk
 - Scripting / Google Cloud (40%)
 - Docker (30%)
 - Opgelet: dit zijn GEEN aparte deelopleidingsonderdelen!
 - Praktisch: bij 2de zit moet je alle delen opnieuw afleggen

Overzicht vak



- Computersystemen 1
 - Principes werking computer
- Computersystemen 2
 - Principes werking besturingssysteem (operating systeem)

Overzicht Theorie

- periode 1 en 2
- studielast:
 - 30% van 5 SP = $0,3 \cdot 5 \cdot 25 = 37,5$ uren
- examen na periode 2
 - laptopexamen
 - gesloten boek (KdG rekenmachine toegelaten)
- inhoud theorie deel
 - theoretische principes van een besturingssysteem (bv. hoe zit een filesysteem in elkaar)
 - toepassingen hierop
 - theoretische oefeningen (bv. wat is de maximale grootte van een filesysteem)
 - praktische oefeningen/labs (bv. opzetten van een RAID filesysteem)

Een besturingssysteem?

- wat is het?
 - software? hardware? firmware?
- wat doet het?
- welke besturingssystemen ken je?

Taken van een besturingssysteem

- boot-process
- hardware abstraction
- i/o management
- file management
- process management
- memory management
- window management

Week 1
IV
...

Computersystemen 2

Theorie

1. Herhaling

Inhoud

- Grootheden
- Harvard / Von Neumann architectuur
- Registers / RAM geheugen

Grootheden

- **Binair**

Macht	van	2	
1 Ki	2^{10}	1024	1,024 K
1 Mi	2^{20}	1024^2	$1,024^2$ M
1 Gi	2^{30}	1024^3	$1,024^3$ G
1 Ti	2^{40}	1024^4	$1,024^4$ T
1 Pi	2^{50}	1024^5	$1,024^5$ P

- **Decimaal**

Macht	van	10	
1 K	10^3	1000	$1,024^{-1}$ Ki
1 M	10^6	1000^2	$1,024^{-2}$ Mi
1 G	10^9	1000^3	$1,024^{-3}$ Gi
1 T	10^{12}	1000^4	$1,024^{-4}$ Ti
1 P	10^{15}	1000^5	$1,024^{-5}$ Pi

Zet om

- 68 608 = Ki
- 46 080 = Ki
- 5 242 880 = Mi
- 150 KiB = KB
- 300 MiB = MB
- 2 GB = GiB
- 5 MB = MiB
- Past een bestand van 95 GiB op een HD van 100 GB?
- 0x100000 = i
- 0x4000 0000 = i

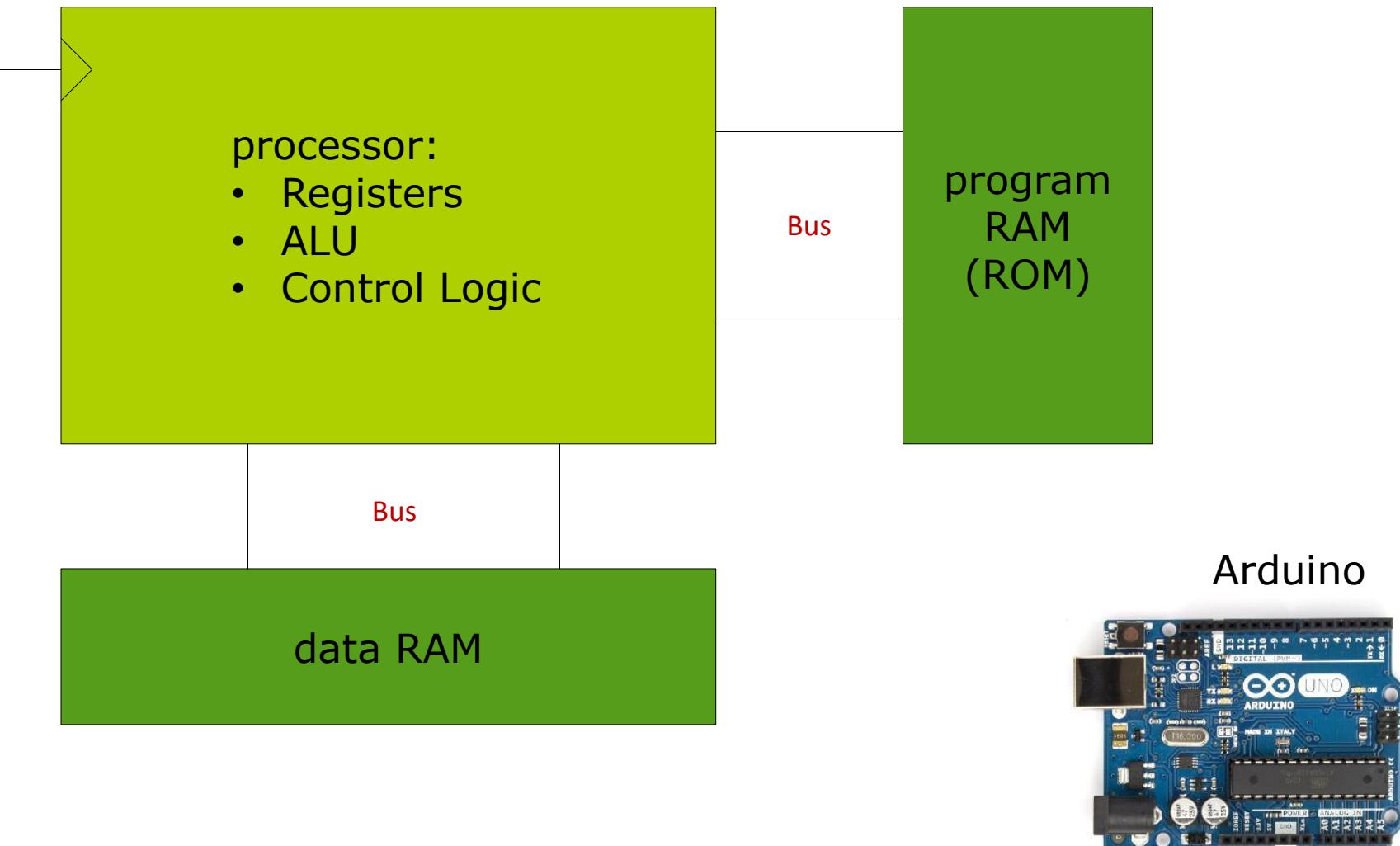
Zet om

1. $68\ 608 = 68\ 608 / 1024 \text{ Ki} = 67 \text{ Ki}$
2. $46\ 080 = \dots \text{ Ki}$
3. $5\ 242\ 880 = \dots \text{ Mi}$
4. $150 \text{ KiB} = \dots \text{ KB}$
5. $300 \text{ MiB} = 300 * 1,024^2 \text{ MB} = 314,6 \text{ MB}$
6. $2 \text{ GB} = \dots \text{ GiB}$
7. $5 \text{ MB} = \dots \text{ MiB}$
8. Past een bestand van 95 GiB op een HD van 100 GB?
9. $0x100000 = 1 * 16^5 = (2^4)^5 = 2^{20} = 1 \text{ Mi}$
10. $0x4000\ 0000 = \dots \square$

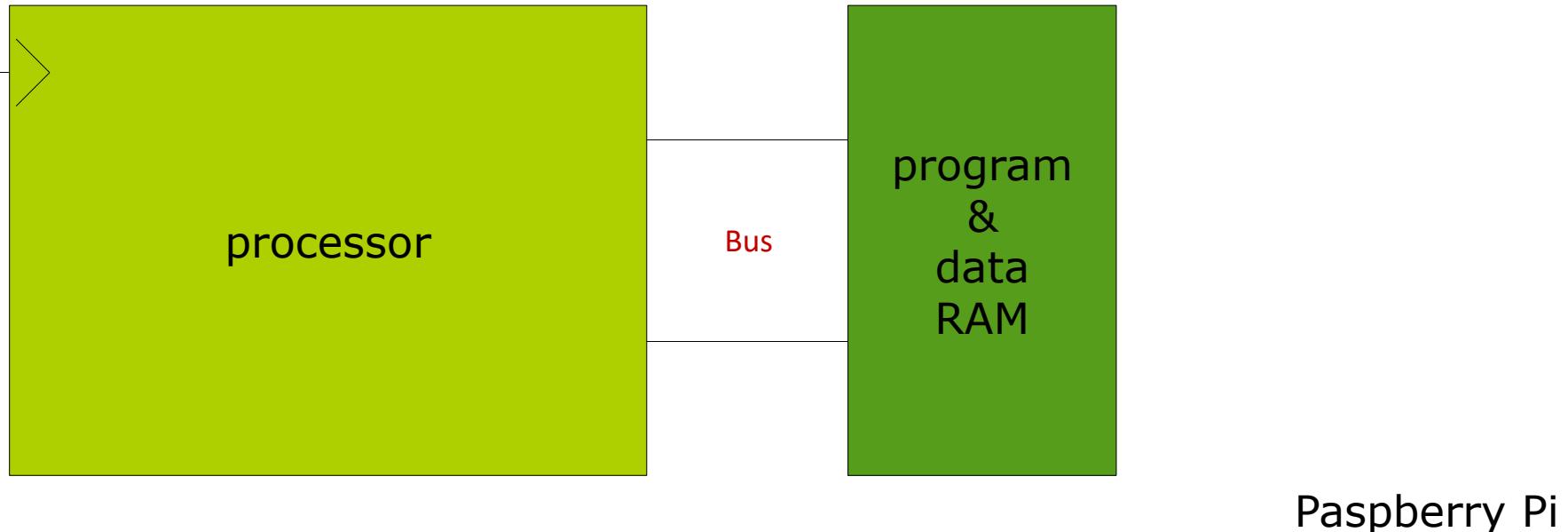
Inhoud

- Grootheden
- Harvard / Von Neumann architectuur
- Registers / RAM geheugen

Harvard architectuur



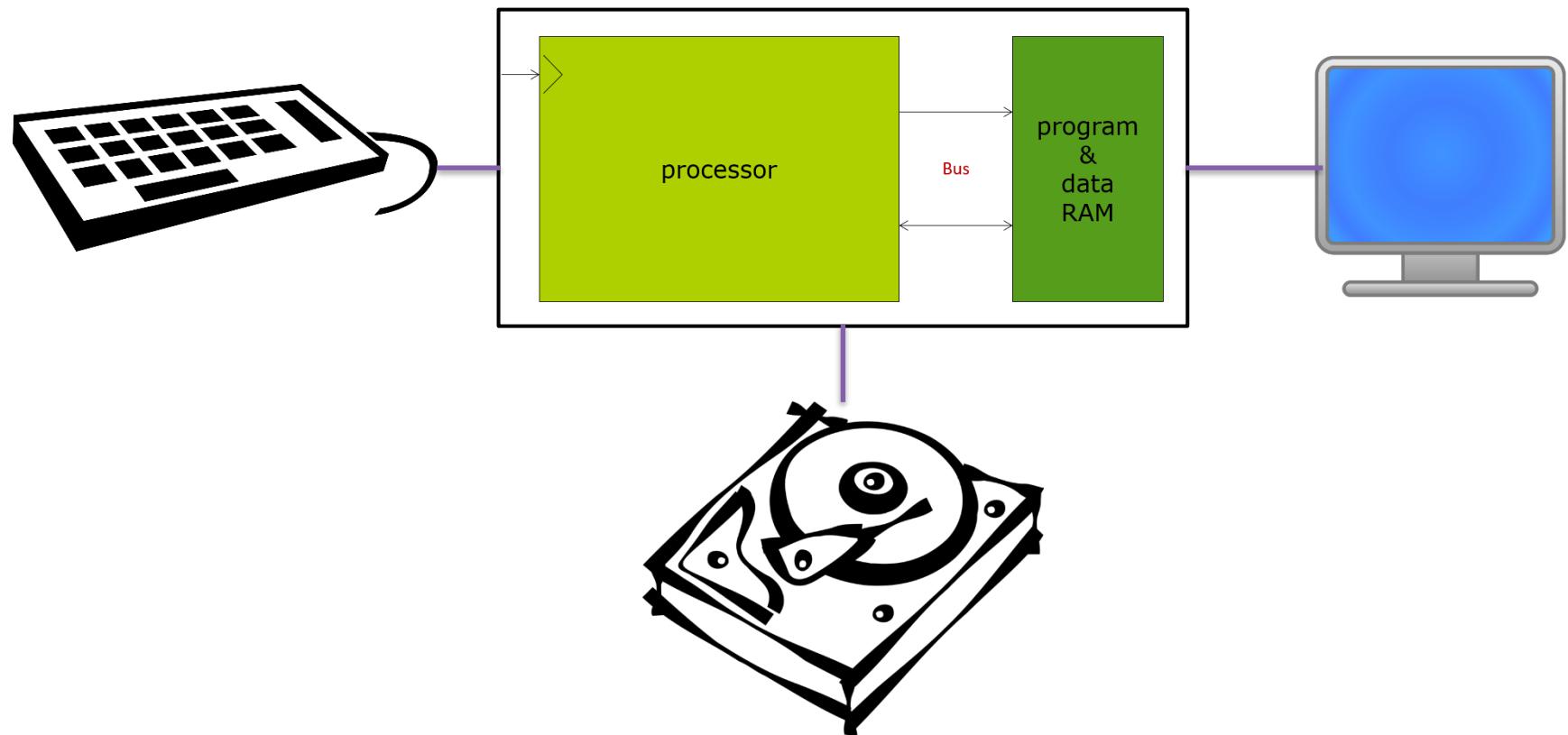
Von Neumann



Instructiecyclus: fetch, decode, execute



Von Neumann – with I/O devices

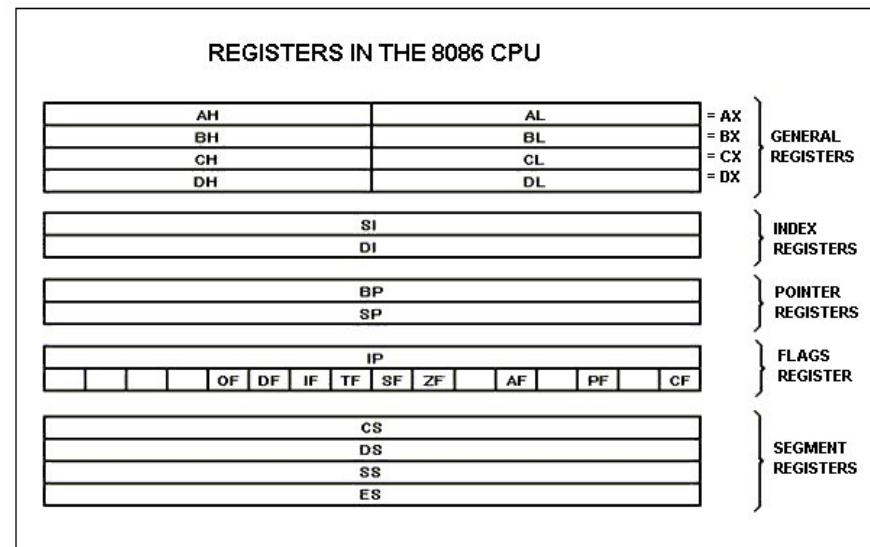


Inhoud

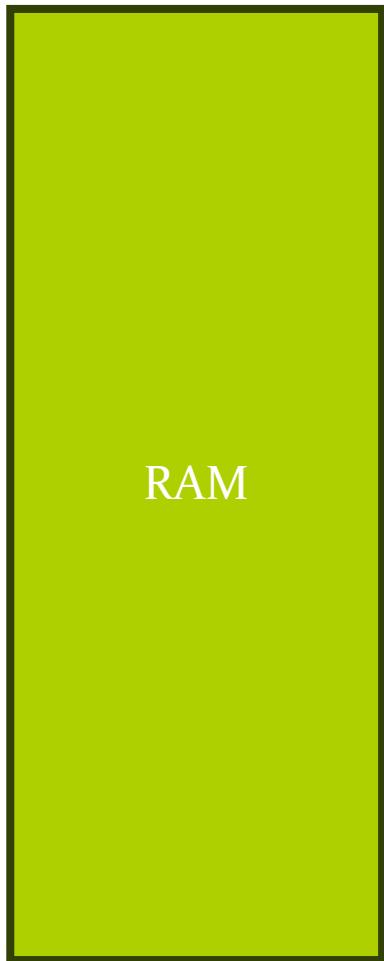
- Grootheden
- Harvard / Von Neumann architectuur
- Registers / RAM geheugen

Registers

- Gegevensregisters (= tijdelijke opslagplaatsen)
- Adresregisters
 - indexregisters
 - segmentregisters
 - stackpointer
- Stuur- en statusregisters
 - program counter
 - flags



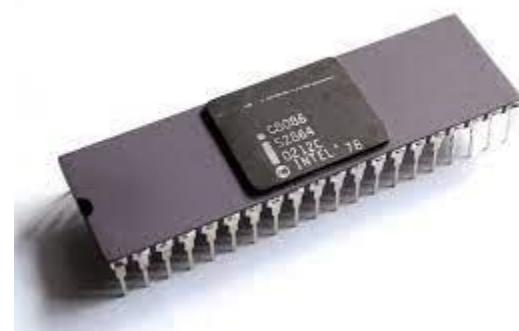
RAM Geheugen



- code
 - machine-code: bytes
 - instructie: instructie code + argumenten
 - vb:
 - laad waarde uit geheugen nr register
 - stockeer waarde uit register in geheugen
 - laad register met waarde
 - tel 2 registers op
 - spring naar ander adres als...
- data
 - getallen (bytes, words, floating point, ...)
 - tekst (ASCII, unicode, EBCDIC, ...)
- zowel code als data zijn bytes
 - data kan als code uitgevoerd worden!
 - beveiliging...

Voorbeeld Intel 8086

- 16 bit processor
 - registers zijn 16 bit
 - grootste getal in register: $2^{16} = 2^6 \cdot 2^{10} = 64 \text{ Ki}$
- Adresbus 20 bit
 - maximale grootte RAM geheugen: $2^{20} = 1 \text{ Mi}$
- Hoe kan je 20 bit adressen maken met 16 bit registers?
 - door combinatie van een register met een segment register
 - adres = CS * **16** + IP
 - maximale grootte: $2^{16} * 16 + 2^{16}$
- PS:
 - Intel Pentium: 32 bit registers en 32 bit adresbus
 - i7: 64 bit en adresbus 40->52 bit



Computersystemen 2

Theorie

2. Booten

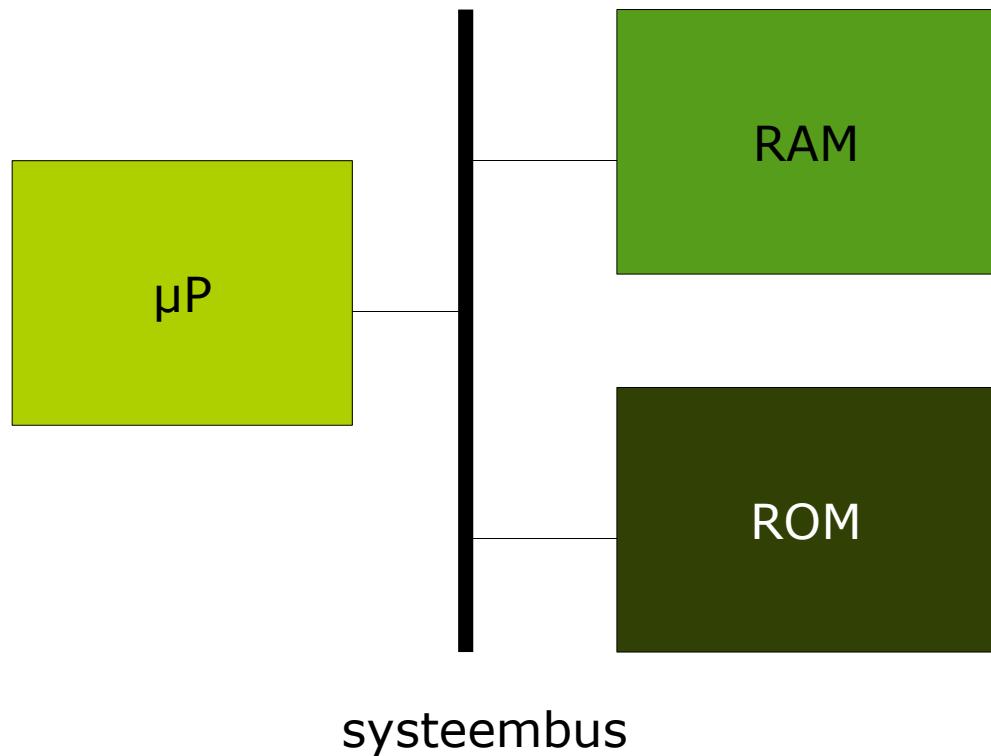
Inhoud

- ROM firmware
- Booten
- MBR en boot-loader
- UEFI
- Mogelijke vragen

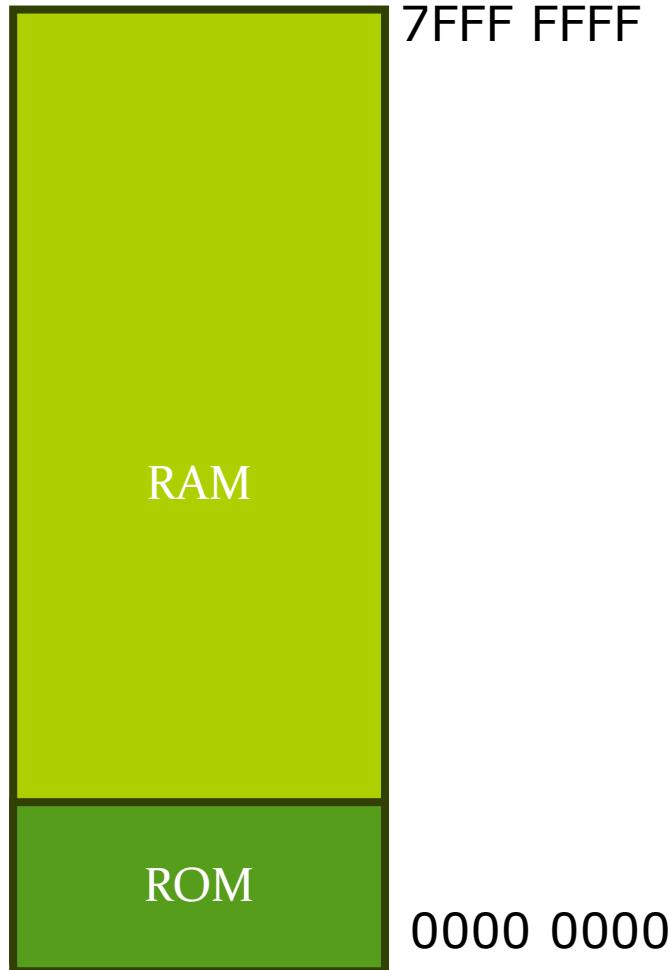
Booting

- Ik zet deze computer aan
- Wat gebeurt er?
- Wat is het probleem?
- Oplossing?

Booting



Booting



- Wat zit er in de ROM?
 - PowerOn Self Test (POST)
 - Hardware Abstraction Layer (HAL)
 - Shell (interface voor gebruiker)
 - Code om andere code te laden van extern medium (bv. harde schijf)

Inhoud

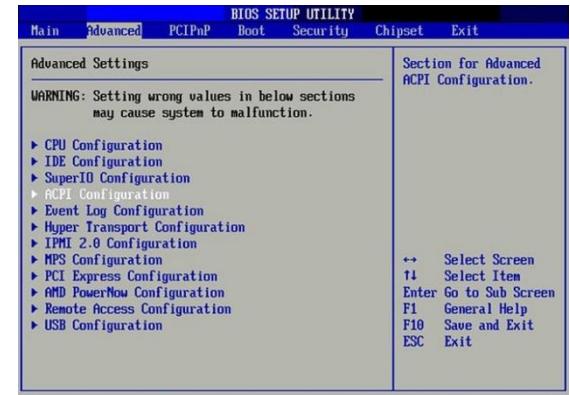
- ROM firmware
- Booten
- MBR en boot-loader
- UEFI
- Mogelijke vragen

Booting

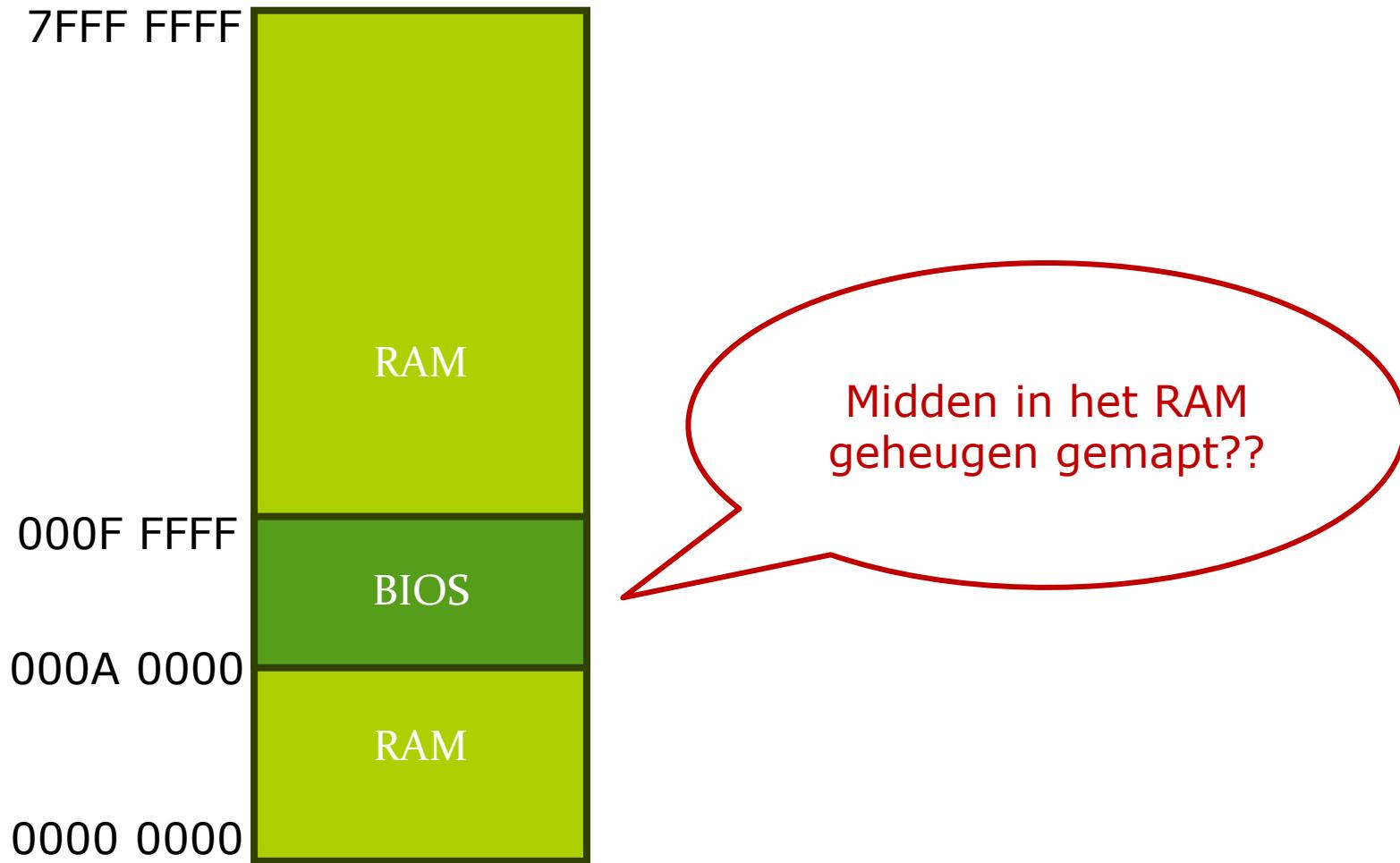
1. ROM firmware
2. Boot loader: als laatste stap laadt de firmware de boot loader van de startup disk naar RAM en voert deze uit
3. Kernel: boot loader start kernel na het laden van disk naar RAM
4. Volledig OS: processen opstarten, filesystemen mounten, netwerk configureren,...

Booting - voorbeeld x86

- PC architectuur:
 - Firmware = **BIOS**
 - BIOS: Basic input/output system
 - gemapt tussen A0000 en FFFFF
 - bevat drivers voor toetsenbord, scherm, HDD
- BIOS voert volgende stappen uit
 - POST
 - HAL
 - laad boot-loader van HDD in RAM op 7C00 en voert uit



Booting - voorbeeld x86



BIOS HAL

- BIOS levert functies via "API"
- geïmplementeerd met "software interrupts"
 - instructie INT
 - http://en.wikipedia.org/wiki/BIOS_interrupt_call
 - parameters worden via registers doorgegeven! -> wat is het gevolg hiervan???
- voorbeelden
 - zet karakter op scherm (INT 0x10)
 - lees karakter van toetsenbord (INT 0x16)
 - lees **sector¹** van HDD (INT 0x13) – BIOS kan enkel sectoren lezen, geen files!

¹ een HDD is opgedeeld in sectoren van 512 byte, zie later

Inhoud

- ROM firmware
- Booten
- MBR en boot-loader
- UEFI
- Mogelijke vragen

Booting – voorbeeld x86

- laatste stap van BIOS:
 - lees eerste sector van HDD of memory stick
 - zet deze op plaats 7C00
 - spring naar 7C00
- eerste sector = **Master Boot Record (MBR)**
 - Eerste 512 bytes van HDD
 - boot-loader (bv. grub): code die het OS laadt, met eventueel menu om verschillende OS-en te kunnen starten
 - gegevens over HDD (grootte, partities, ...)
 - bootsignatuur (magic number)

Booting - laden van OS

- Computer start op via BIOS
- Eerste sector van HDD wordt geladen, maar die is heel klein (512 bytes)
- Laad rest van boot-loader uit de volgende sectoren
- Boot-loader bevat code om een filesysteem te lezen en kan dan de kernel van het OS van de HDD laden

MBR Lab

- Bekijk 1^{ste} sector van je harde schijf of memory stick (PC)
- Linux:

```
sudo dd if=/dev/sda bs=512 count=1 | hexdump -v
```

- Windows HexEditor:

<http://mh-nexus.de/en/hxd/>

- Op welke adressen bevinden zich:
 - Code boot-loader
 - Partitietabel
 - bootsignatuur



Inhoud

- ROM firmware
- Booten
- MBR en boot-loader
- UEFI
- Mogelijke vragen

Booting - UEFI

- BIOS is verouderd, **UEFI** is nieuwe standaard
 - (U)EFI: (Unified) Extensible Firmware Interface
 - Heeft BIOS compatible mode

	BIOS	UEFI
Laad code van	MBR = 1 ^{ste} sector HDD	
Partitietabel	in MBR	
Grootte partitietabel	4 partities die elk 16 byte innemen in MBR (14 part. met extended/logische)	
Max grootte filesysteem	2TiB	
Kan secure boot	Nee	
Boot-loader nodig	Ja altijd	

Booting - UEFI

- BIOS is verouderd, **UEFI** is nieuwe standaard
 - (U)EFI: (Unified) Extensible Firmware Interface
 - Heeft BIOS compatible mode

	BIOS	UEFI
Laad code van	MBR = 1 ^{ste} sector HDD	EFI-file op EFI systeem partitie (bv. \EFI\BOOT\BOOTx64.EFI)
Partitietafel	in MBR	in GPT (GUID ¹ partitie tabel)
Grootte partitietafel	4 partities die elk 16 byte innemen in MBR (14 part. met extended/logische)	128 partities die elk 128 byte innemen in GPT
Max grootte filesysteem	2TiB	8ZiB
Kan secure boot	Nee	Ja
Boot-loader nodig	Ja altijd	Kan rechtstreeks kernel opstarten

¹ GUID: Globally unique identifier

Booting lab

1. Zoek uit of jou laptop BIOS of UEFI gebruikt
2. Installeer Ubuntu in VirtualBox met UEFI
3. Ga naar de UEFI Interactive shell (ev. via de UEFI Firmware settings) door het booten te onderbreken met de Esc toets

CS2 [Draaiend] - Oracle VM VirtualBox

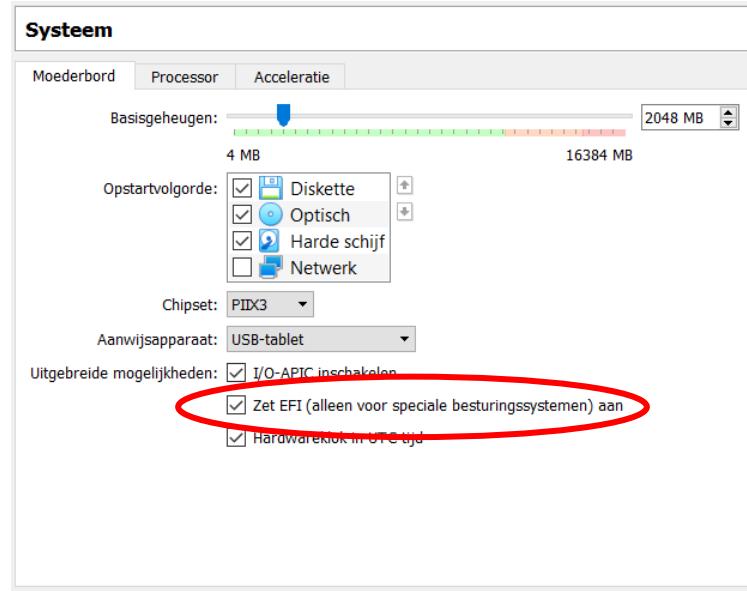
Bestand Machine Weergeven Invoer Apparaten Hulp

UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)

Mapping table

```
FS0: Alias(s) :HD1a65535a1::BLK2:  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x0,0xFFFF,0x0)/HD(1,GPT,05D69016-3AB4-4775-80  
BLK0: Alias(s) :  
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)  
BLK1: Alias(s) :  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x0,0xFFFF,0x0)  
BLK4: Alias(s) :  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x1,0xFFFF,0x0)  
BLK6: Alias(s) :  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x2,0xFFFF,0x0)  
BLK3: Alias(s) :  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x0,0xFFFF,0x0)/HD(2,GPT,E0CD2840-C87E-4FB9-B4  
BLK5: Alias(s) :  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x1,0xFFFF,0x0)/HD(1,GPT,B000024F-A22B-054A-95  
BLK7: Alias(s) :  
    PciRoot(0x0)/Pci(0x0,0x0)/Sata(0x2,0xFFFF,0x0)/HD(1,GPT,E00F6CBB-ACCA-5749-B4  
Press ESC in 2 seconds to skip startup.nsh or any other key to continue.
```

Shell> fs0:
fs0:> ls
Directory of: FS0:\
09/14/2020 11:18 <DIR> 4,096 EFI
0 File(s) 0 bytes
1 Dir(s)
FS0:> _



Inhoud

- ROM firmware
- Booten
- MBR en boot-loader
- UEFI
- Mogelijke vragen

Voorbeelden van examenvragen

- Wat zijn de taken van een besturingssysteem?
- Wat is een Von Neumann architectuur?
- Wat is het verschil met een Harvard architectuur?
- Wat is een bus?
- Wat zijn I/O devices?
- Wat is POST?
- Wat is een HAL?
- Welke stappen worden uitgevoerd bij het opstarten?
- Hoe ziet de MBR eruit?
- Wat zijn de verschillen tussen BIOS en UEFI?
- Is een boot-loader nodig?
- Wat is het nut van de boot signature?
- Hoe ziet de partitietabel eruit?
- Hoe kan je zien of BIOS of UEFI hebt?

Computersystemen 2

Theorie

3. I/O Beheer

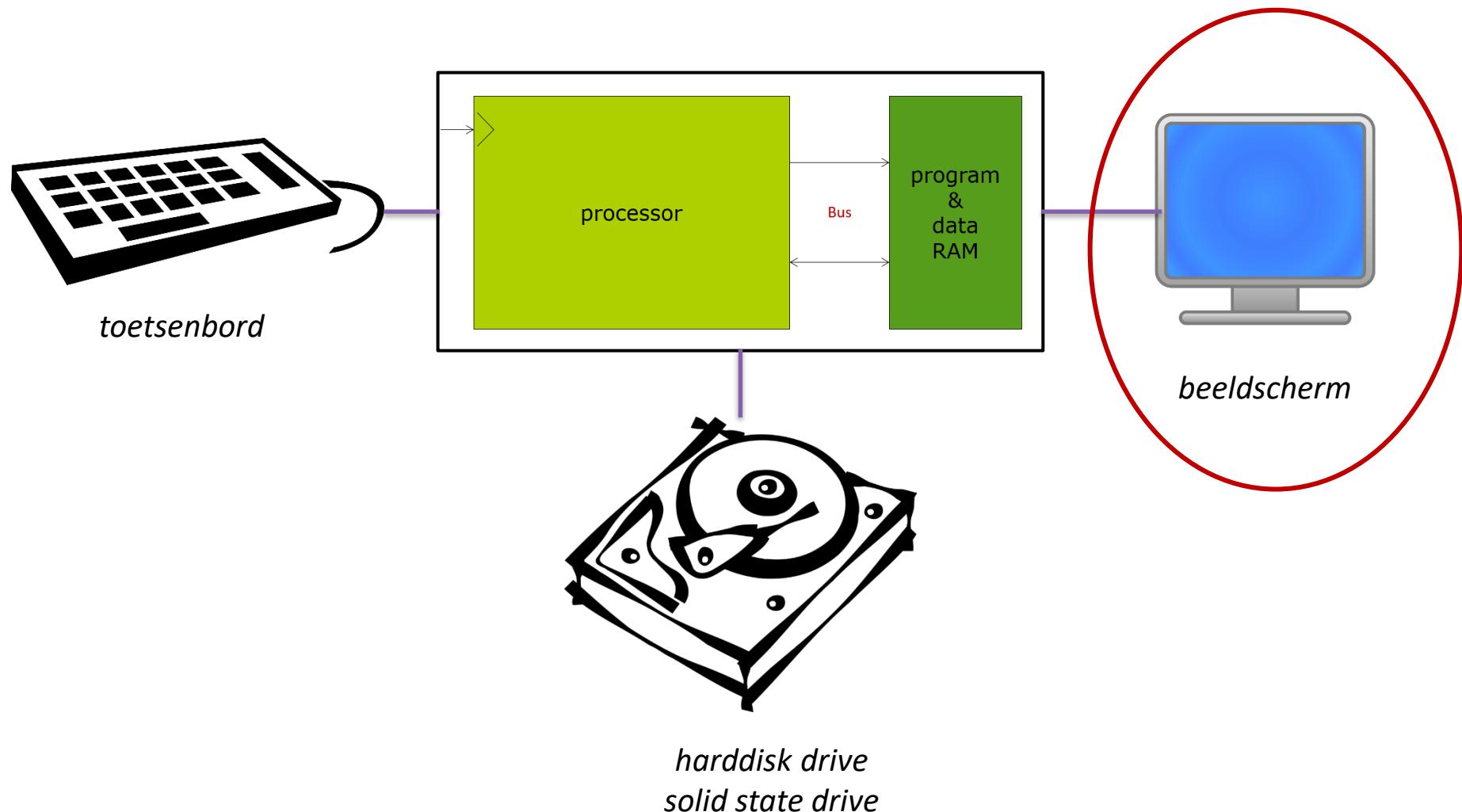
Input/Output

- OK, computer start op
- maar hoe kan je iets op een scherm zetten?
- hoe weet je welke toets een gebruiker heeft aangeslagen?
- hoe lees je iets van de harddisk drive (HDD)/solid state drive (SSD)?
- De BIOS biedt wel iets aan, maar veel te beperkt

Inhoud

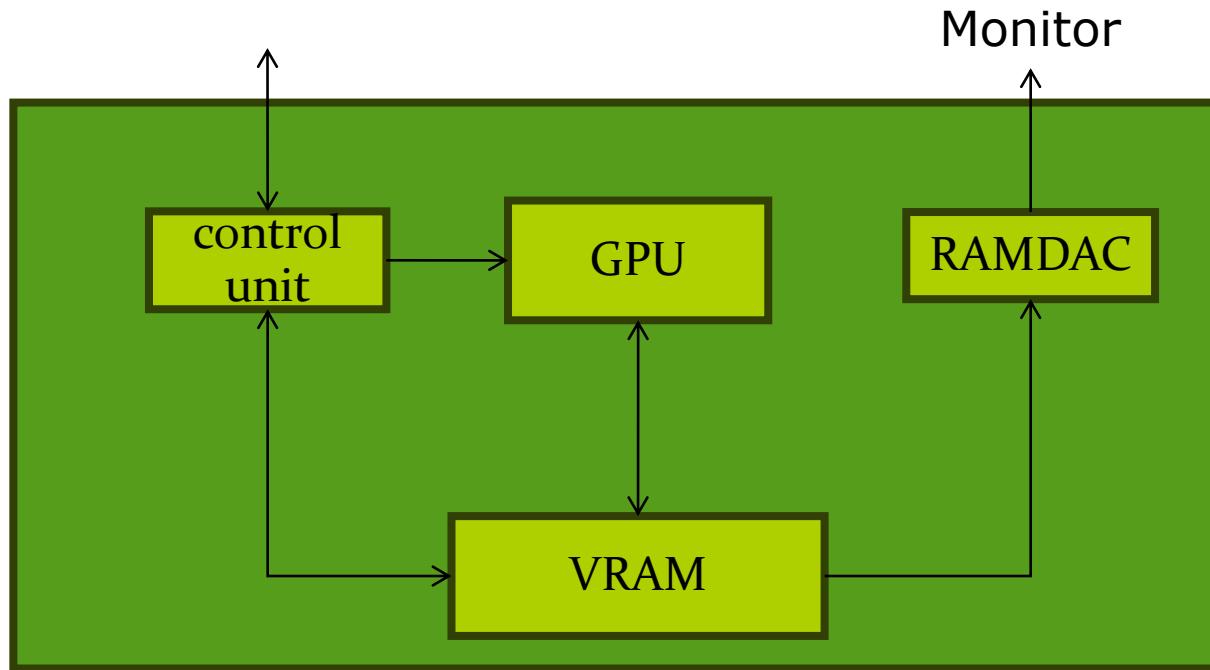
- Beeldscherm
- Toetsenbord
- Disk drive
- Herhalingsvragen

Input/Output - Beeldscherm



Grafische kaart

- Het beeldscherm wordt bestuurd door grafische kaart

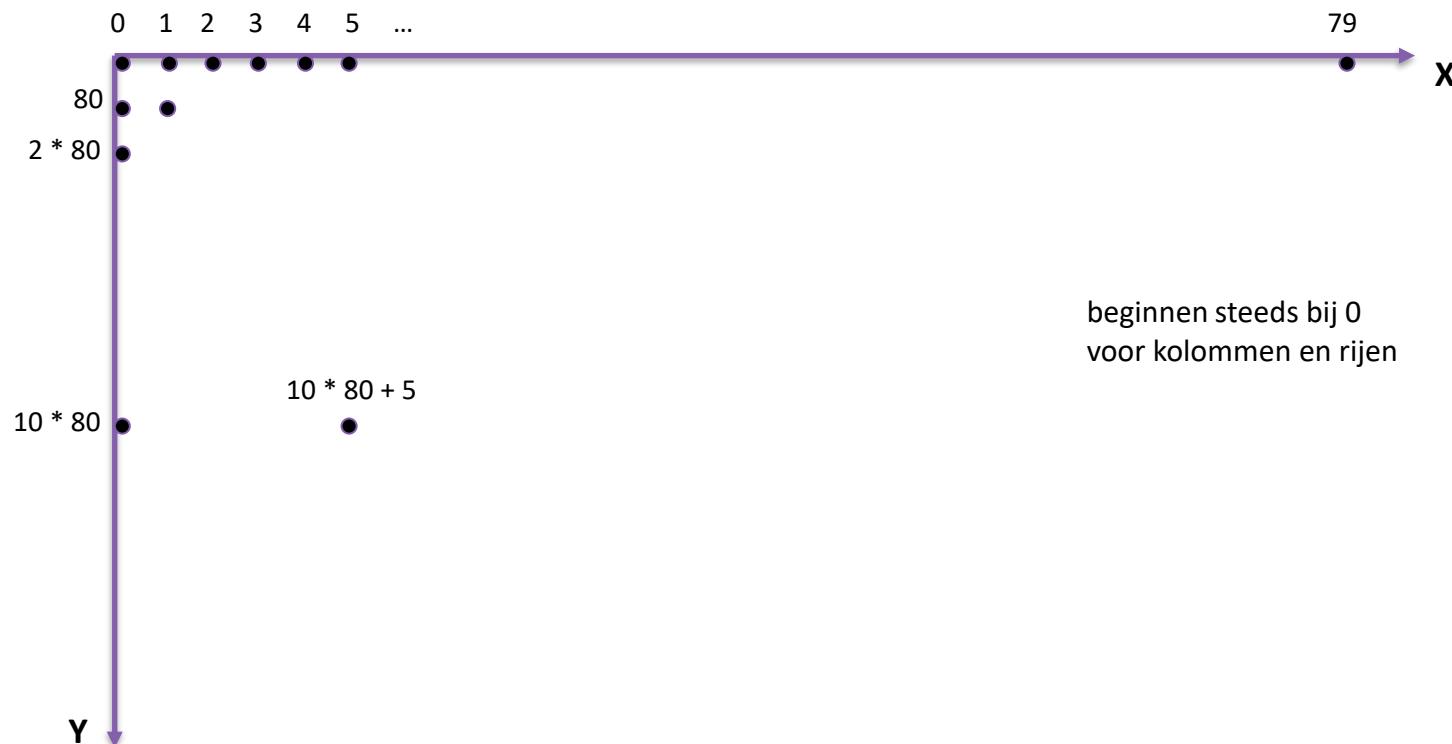


Video modes

- video geheugen bevat data voor beeld
- **text-mode**
 - 80x25 karakters
 - 1 byte per karakter op het scherm
- **grafische mode**
 - pixels van links naar rechts en van boven naar onder
 - **indexed**
 - 1 byte per pixel
 - byte is een kleurnummer uit tabel
 - tabel bevat eigenlijke kleuren
 - **true-color**
 - 3 bytes per pixel (RGB)

Video RAM

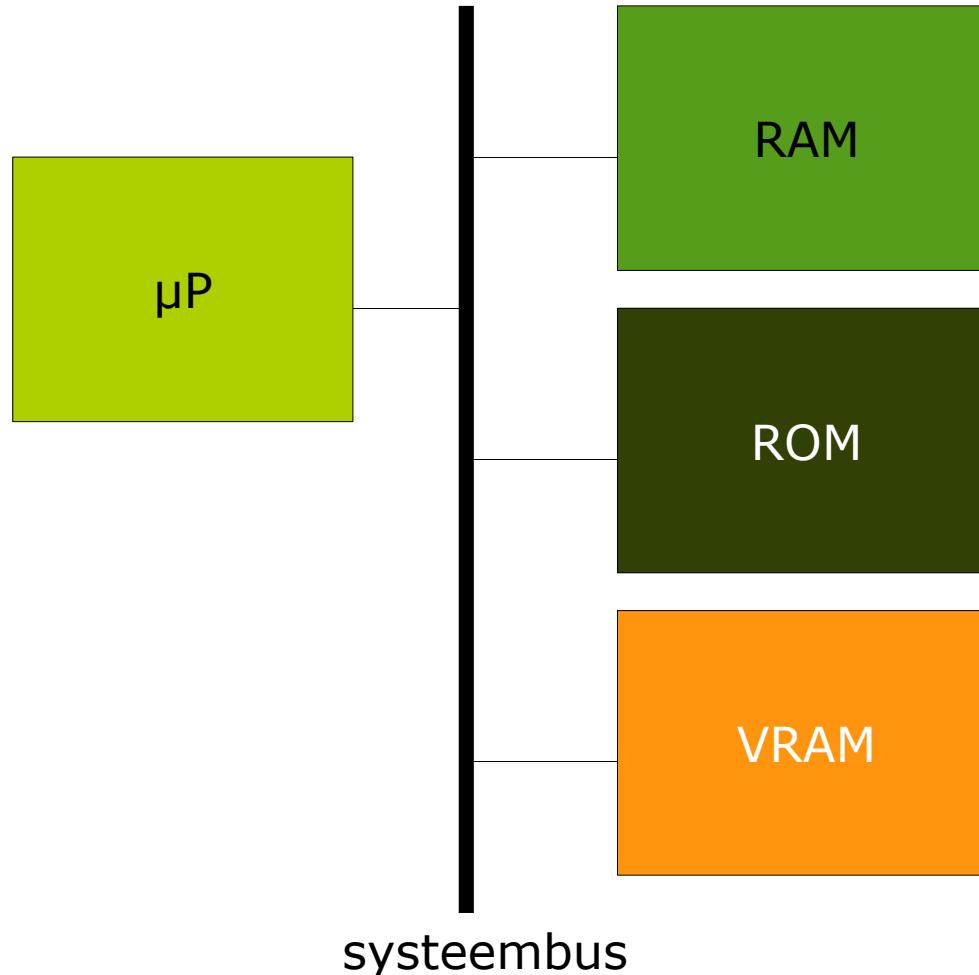
- voorbeeld 1: text-mode
 - zet karakter 'A' op kolom 5 van regel 10
 - $\text{VRAM}[10*80+5] = 65$



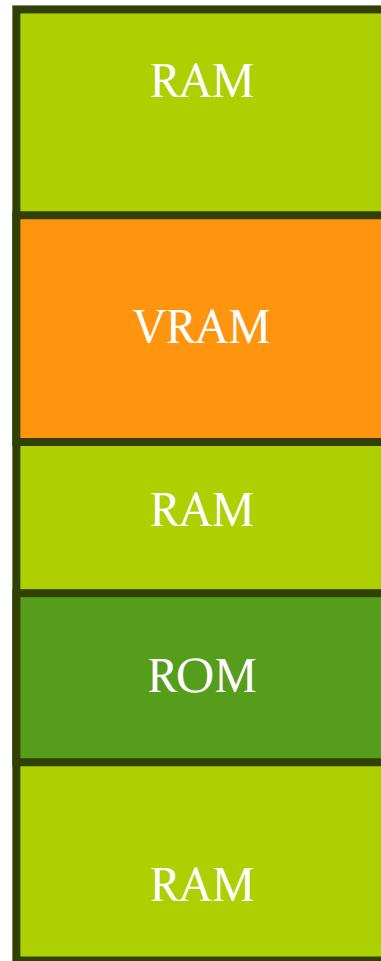
Memory-mapped I/O

- Hoe kan de processor nu een karakter/pixel tekenen op het scherm?
- hoe krijgt de processor toegang tot de VRAM?
- --> **memory-mapped I/O**

VRAM – Memory-mapped



VRAM – Memory-mapped

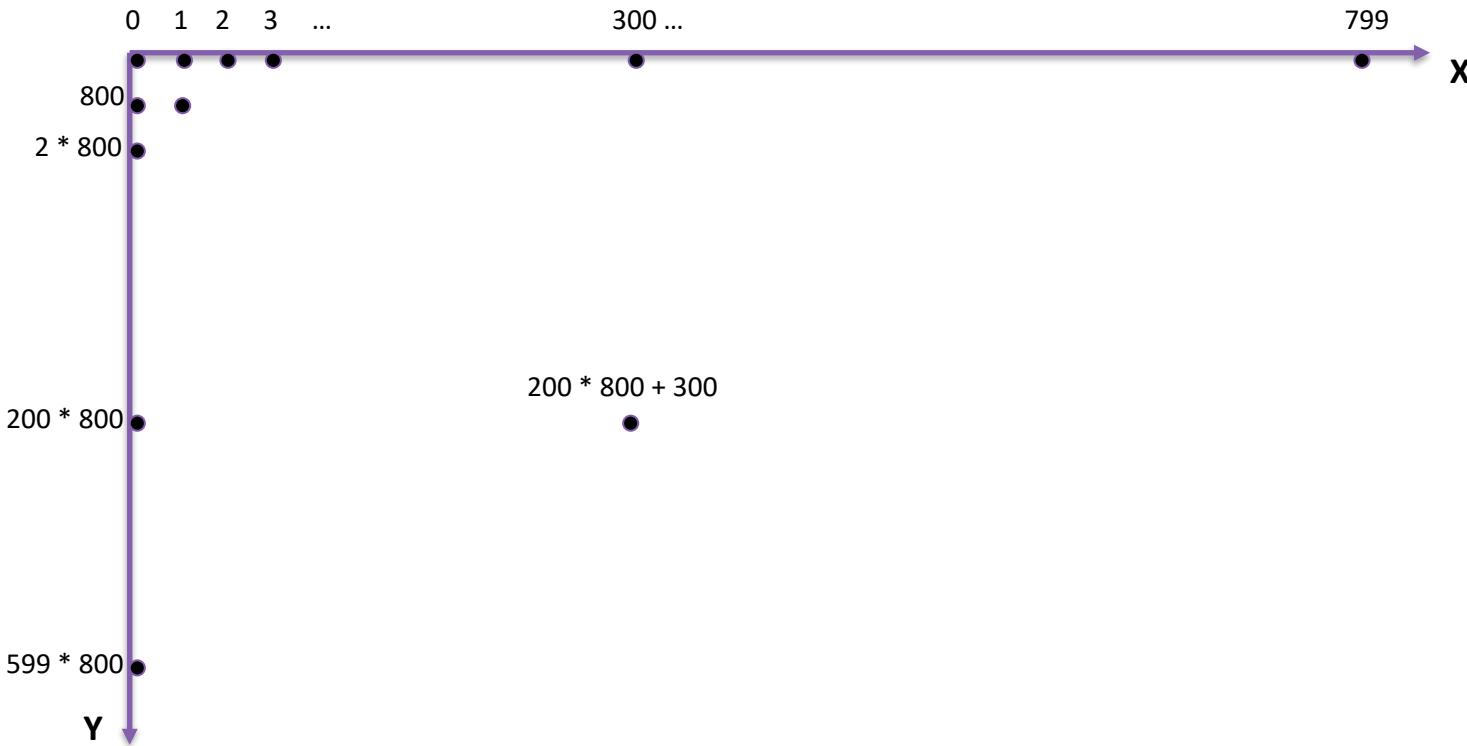


VRAM – Memory-mapped

- voorbeeld 1:
 - stel dat VRAM gemapt is op 0xA0000
 - video kaart staat in text-mode
 - zet karakter 'A' op kolom 5 van regel 10
 - **VRAM[805] = 65 => RAM[0xA0325]= 65**
- voorbeeld 3 (oefening):
 - VRAM gemapt op 0x1E2345
 - video kaart staat in indexed graphical mode (800x600)
 - zet pixel (300, 200) op kleur nummer 5
 - geef adres hexadecimaal



Beeldscherm



- VRAM[160300]=VRAM[0x2722C]=5
- RAM[0x1E2345 + 0x2722C]=RAM[0x209571]=5

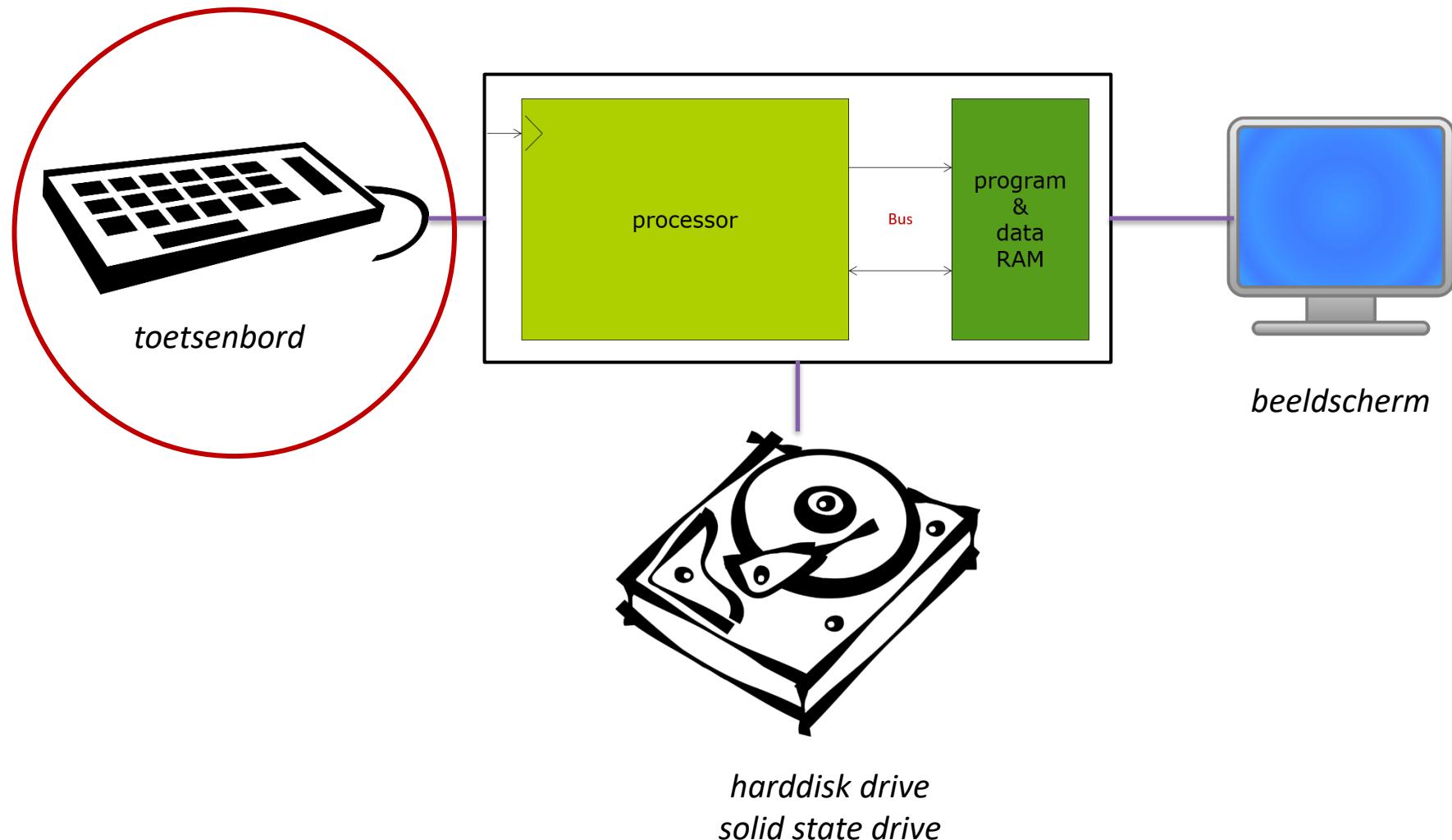
Beeldscherm

- sommige operaties kosten veel tijd
 - bv: drawCircle()
 - ==> processor moet alle pixels berekenen
- oplossing: GPU
 - CPU zet opdracht in VRAM
 - GPU voert dit uit
 - GPU kan: lijnen, cirkels, polygonen, 3D, ...

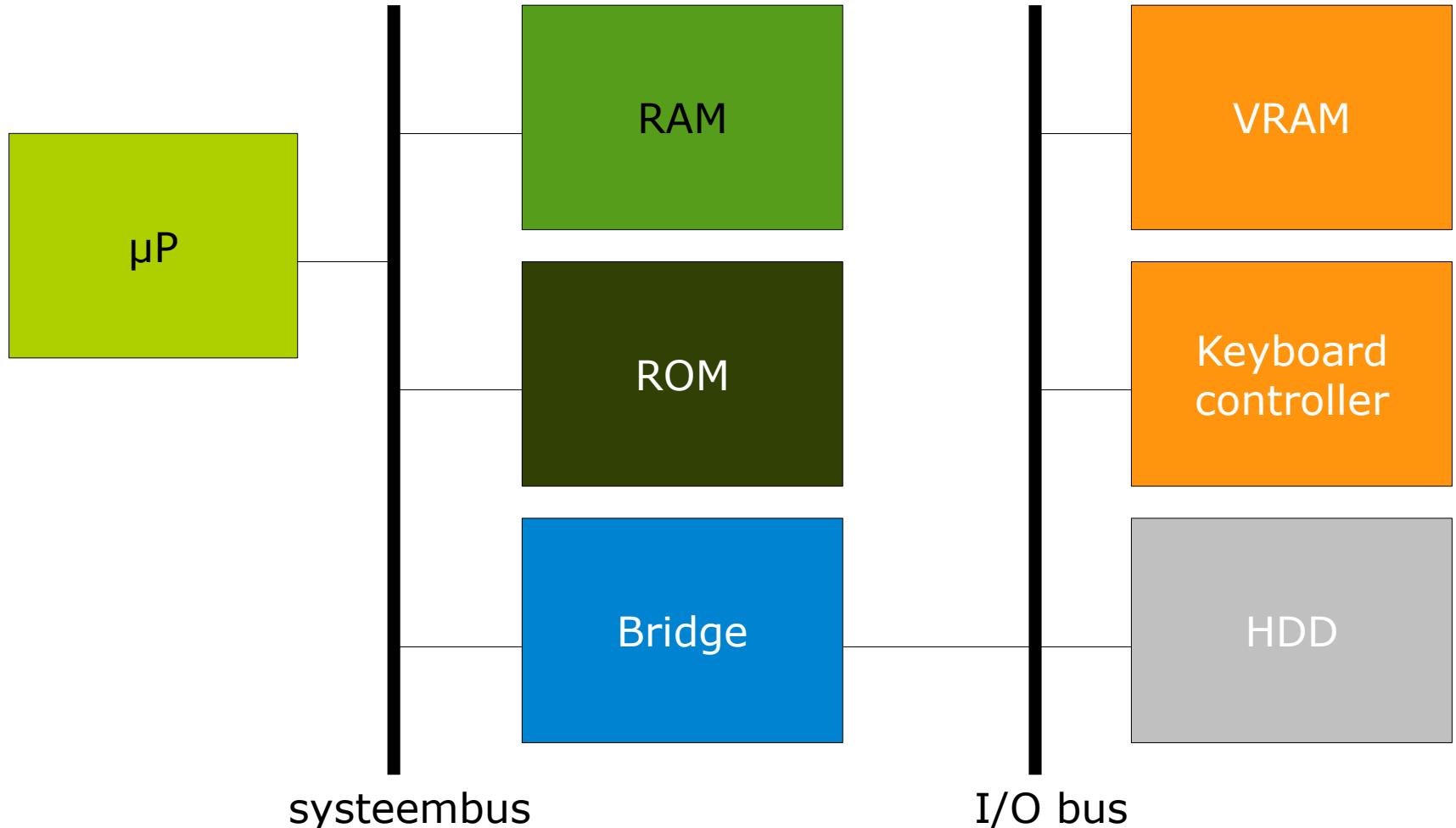
Inhoud

- Beeldscherm
- Toetsenbord
- Disk drive
- Herhalingsvragen

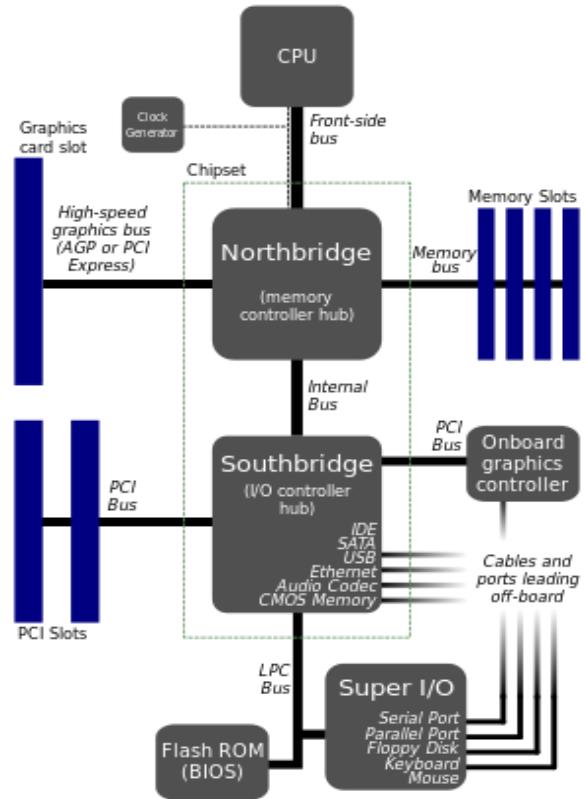
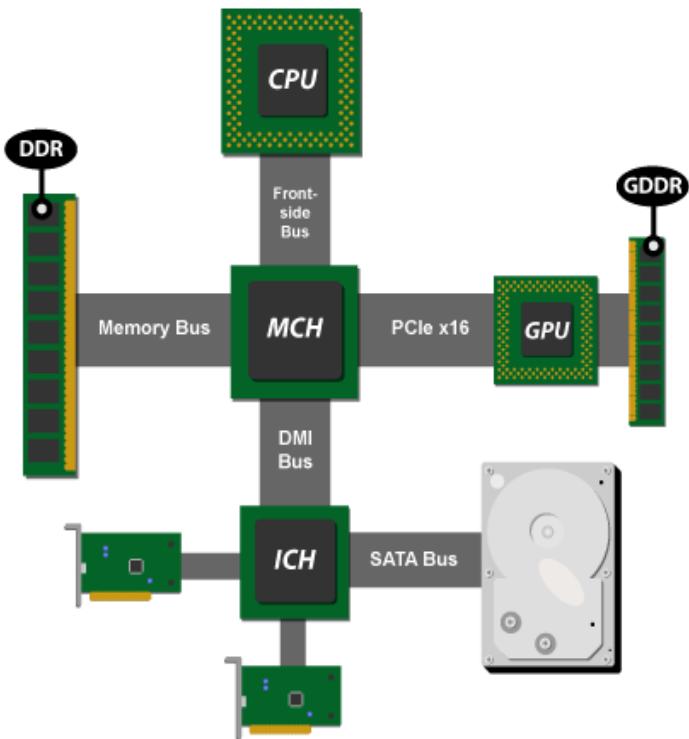
Input/Output - Toetsenbord



Toetsenbord – Memory-mapped



Bridges en bussen

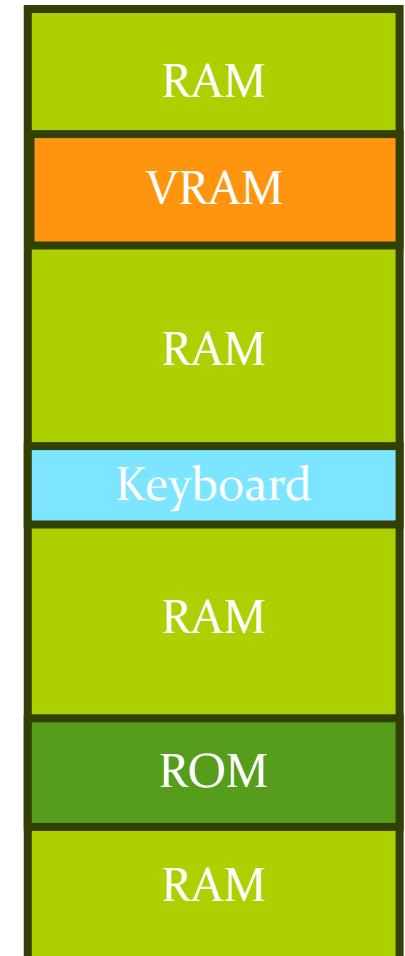


Toetsenbord – Memory-mapped



LET OP
register keyboard != register CPU

- keyboard controller bevat 3 "registers"
 - data (laatste aanslag)
 - status (is er data klaar?)
 - control (settings)
- keyboard "registers" worden in geheugen gemapt:
memory-mapped I/O*



* Het grote voordeel van memory mapped I/O is dat de gewone ld en st machinetaal instructies gebruikt kunnen worden om I/O te doen

Programmed I/O

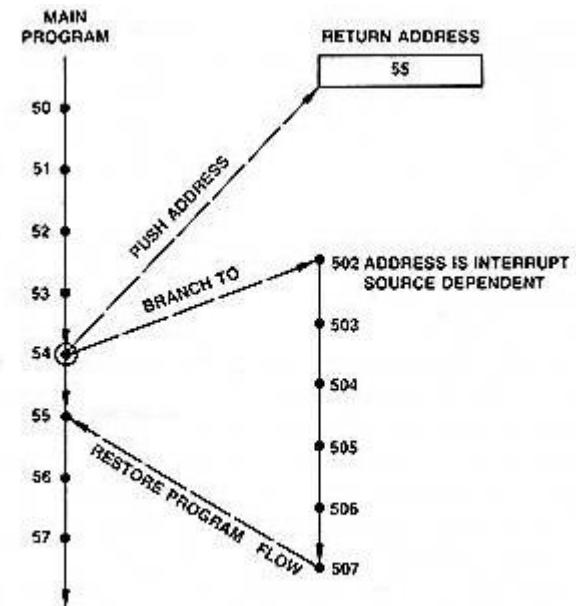
- stel: programma wacht op toets

```
int status;  
do {  
    status = memory[KEYB_STATUS];  
} while (status != GEREED)  
char c = memory[KEYB_DATA];
```

- polling
- dit heet **programmed I/O**
- nadeel?

Interrupt driven I/O

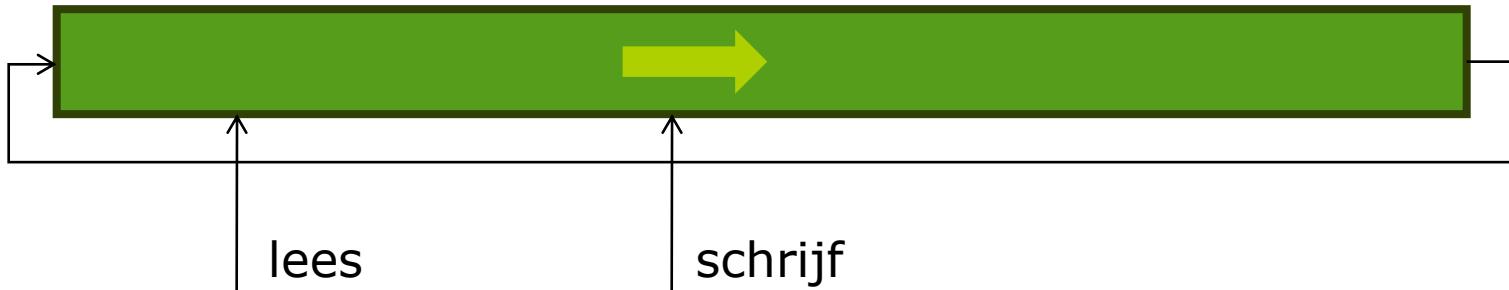
- het zou beter zijn als de processor een sein krijgt als er data klaar staat -> interrupt
 - Pin op de CPU
 - CPU krijgt interrupt van keyboard controller wanneer op toets wordt gedrukt
 - Interrupt tabel: interrupt nr. <-> adres ISR
 - Interrupt Service Routine (ISR) wordt uitgevoerd
 - Nadien teruggesprongen naar oorspronkelijke taak



Interrupt
driven I/O

Circulaire buffer

- ISR zal toetsaanslagen in "circulaire buffer" in RAM zetten
- Circulaire buffer
 - Vaste grootte
 - lees=schrijf => empty
 - schrijf=lees-1 => full + risk for overflow

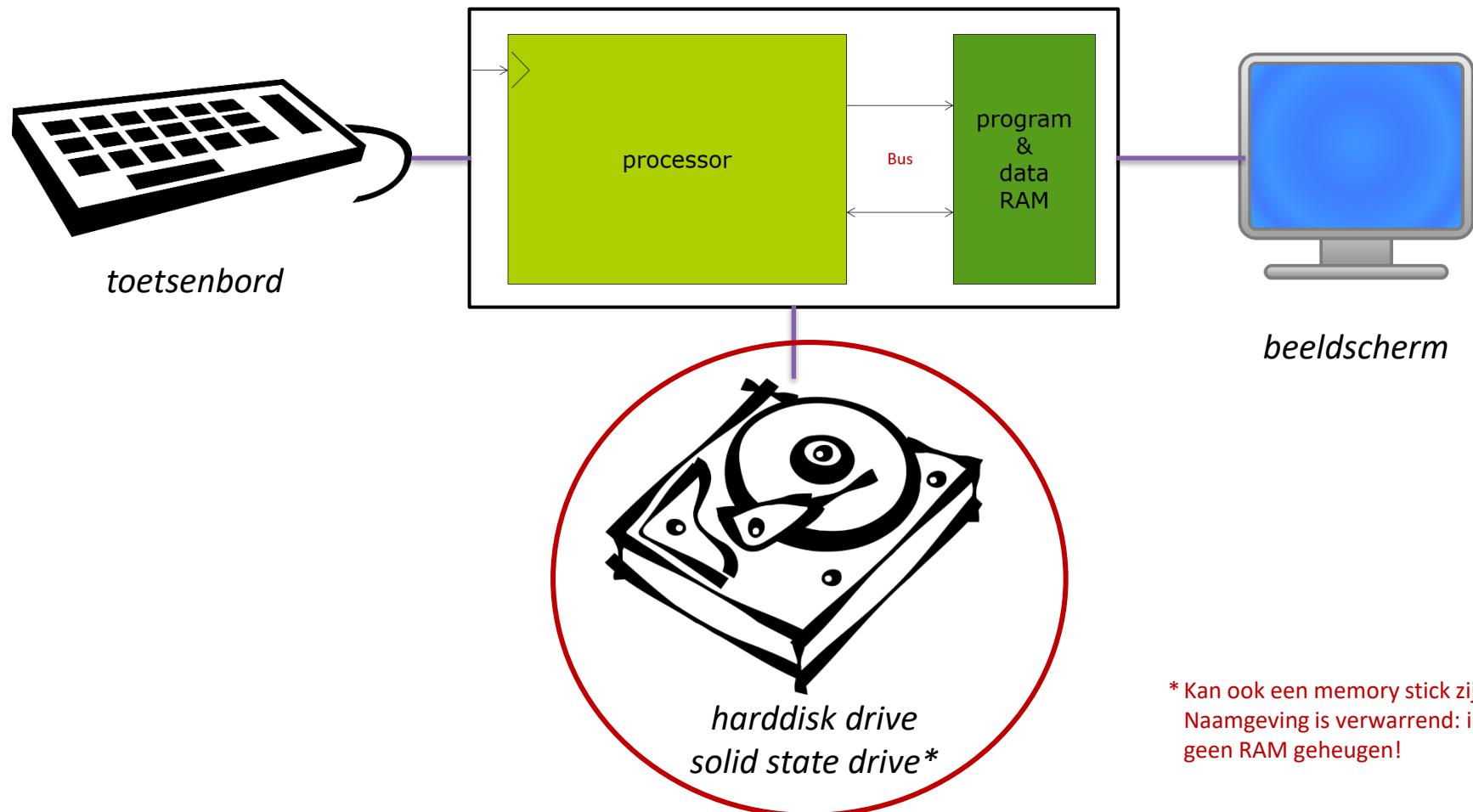


- software zal buffer af en toe lezen

Inhoud

- Beeldscherm
- Toetsenbord
- Disk drive
- Herhalingsvragen

Input/Output - Disk drive

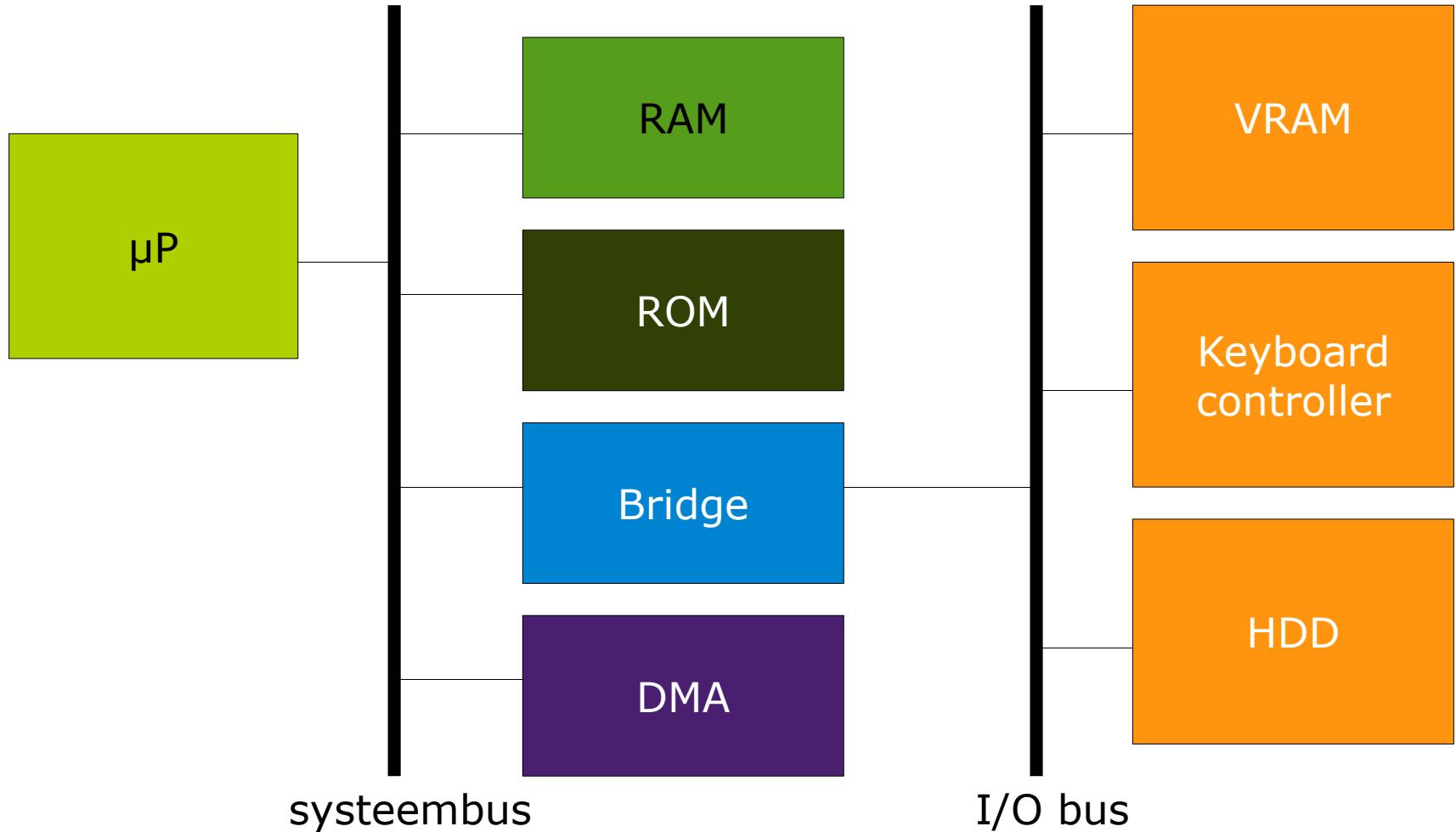


* Kan ook een memory stick zijn
Naamgeving is verwarrend: is
geen RAM geheugen!

Disk drive

- Interrupt driver I/O veel efficiënter dan programmed I/O
- Bij lezen van disk worden grote hoeveelheden data verplaatst
- Wat is probleem?
- --> **Direct Memory Access (DMA)**
- Processor geeft aan DMA opdracht om data te lezen (of schrijven) en kan ondertussen iets anders doen
- DMA krijgt op dit moment controle over de bus!

DMA



Inhoud

- Beeldscherm
- Toetsenbord
- Disk drive
- Herhalingsvragen

herhalingsvragen

- Welke componenten zitten er op een video-kaart?
- Wat is de rol van de GPU?
- In welke modes kan een video kaart (grosso modo) werken?
- Stel dat een grafische kaart in ... mode staat met een resolutie van ... De vram is gemapt op ... Op welke plaats in het RAM geheugen moet wat geschreven worden om een pixel/karakter te tekenen?
- Wat is memory-mapped I/O? Wat zijn de voordelen ervan? Gebruiken alle computers memory-mapped I/O
- wat is het verschil tussen programmed I/O en interrupt-driven I/O? Wat is het voordeel van interrupt-driven I/O?
- Leg de werking van een circulaire buffer uit. Wanneer vol? Wanneer leeg?
- Wat is een ISR? wat doet het?
- Wat is een DMA controller? Wat doet deze? Waarom is die nodig?

Computersystemen 2

Theorie

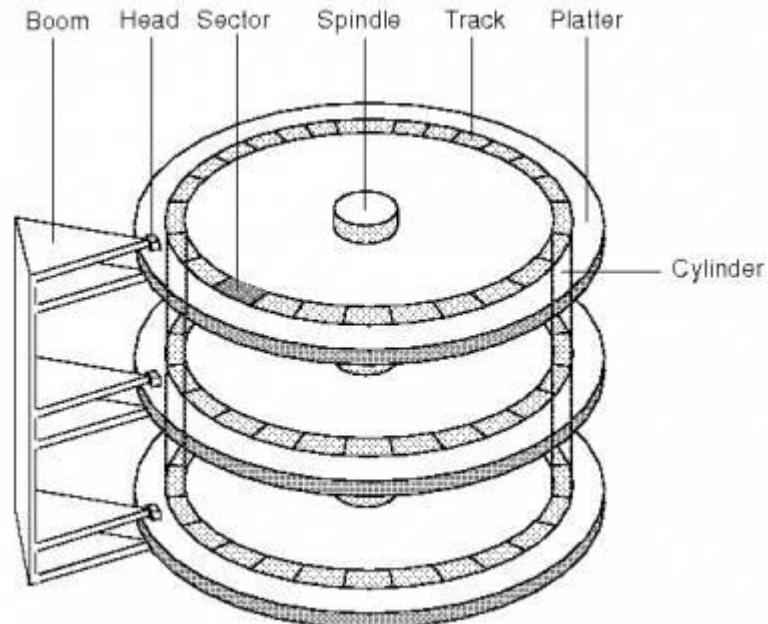
4. Bestandsbeheer

Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

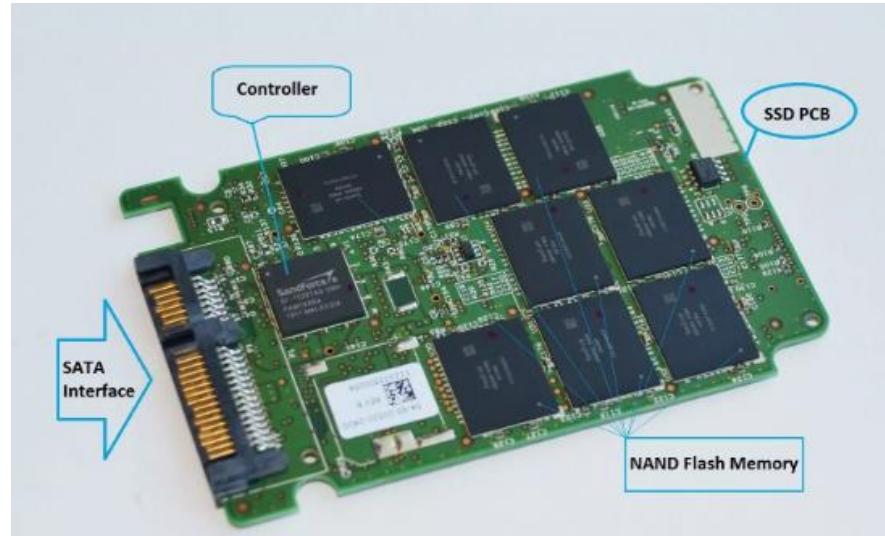
HDD - Hard Disk Drive

- harde schijf
 - disks
 - heads
 - cilinders
 - tracks
 - sectors
- magnetisch
 - weinig schokbestendig
 - latency (kop heen en weer bewegen)
- bij delete file
 - file wordt verwijderd uit directory tabel
 - sectoren van file blijven op disk staan (en kunnen nadien overschreven worden)



SSD - Solid-state Drive

- SSD:
 - Interface: SATA, ...
 - Controller (firmware)
 - NAND Flash Memory:
 - vgl. met static RAM (D-flipflop)
 - maar met floating gate transitoren
 - (bewaren toestand bij power off)
- Voltage levels:
 - SLC (single level cell): 1 bit/cell, 0 of 1 (bv. 0 en 3V)
 - MLC (multi level cell): 2 bits/cell, 00, 01, 10 of 11 (bv. 0, 1, 2 en 3V)
 - hogere capaciteit, trager, minder betrouwbaar
 - zelf TLC (tripe level cell): 3bits/cell



SSD - Solid-state Drive

- geschreven page kan niet overschreven worden
 - eerst leeg maken
 - bij delete file:
 - niet enkel file verwijderen uit directory
 - **TRIM instructie** aan OS toegevoegd: doorgeven aan SSD om pages te markeren om leeg te maken (gebeurt door de controller van de SSD)
- cellen degraderen bij schrijven
 - aantal write cycles is beperkt
 - **geen defragmentatie!**
 - **wear-leveling**
 - erases & writes are evenly distributed over the SSD IC's
 - move frequently accessed data to lower used blocks

SSD - HDD

	SDD	HDD
Access Time	35 to 100 µsec	5 to 10 ms
Price/Capacity	Most Expensive	Cheaper (>1TB)
Reliability	No moving parts (ICs)	Platters
Power	Less	More
Noise	No noise	Spinning Disks Moving heads
Size	Many size + small	Typical only 3,5" and 2,5"
Heat	Almost no heat	Moving parts cause heat
Magnetism	No effect	Erase possible
FileSystem	No Defragmentation	Defragmentation

Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

Schijftoegang - RAID

- **RAID**: Redundant Array of Independent Disks
- virtuele schijf bestaande uit verschillende fysische harde schijven
- filesystemen kunnen verspreid staan over verschillende harde schijven
- voordelen
 - grotere filesystemen
 - meerdere schijfoperaties tegelijk uitvoeren
 - redundantie is mogelijk
- implementatie
 - software: in het OS
 - hardware: OS weet van niets

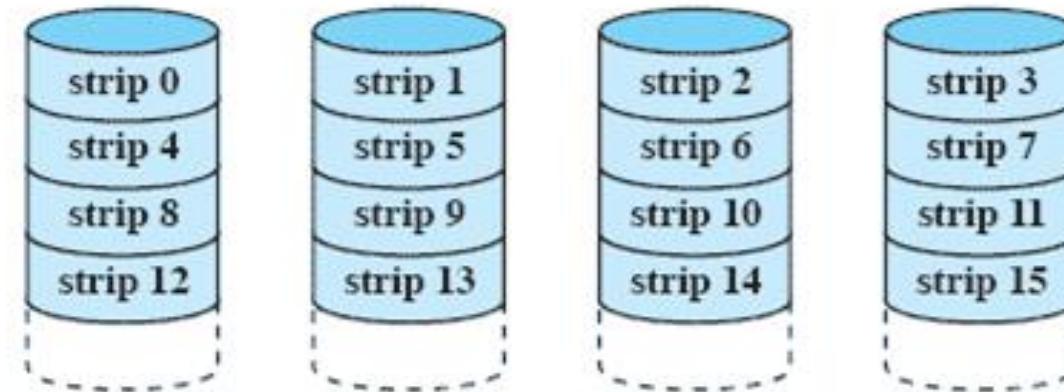


Schijftoegang - RAID

- RAID heeft verschillende niveau's
 - 0 tot 6
 - enkel 0, 1 en 5 komen veel voor
 - je kan ook combinaties maken
(vb: RAID-10, RAID-51)

RAID-0 (striping)

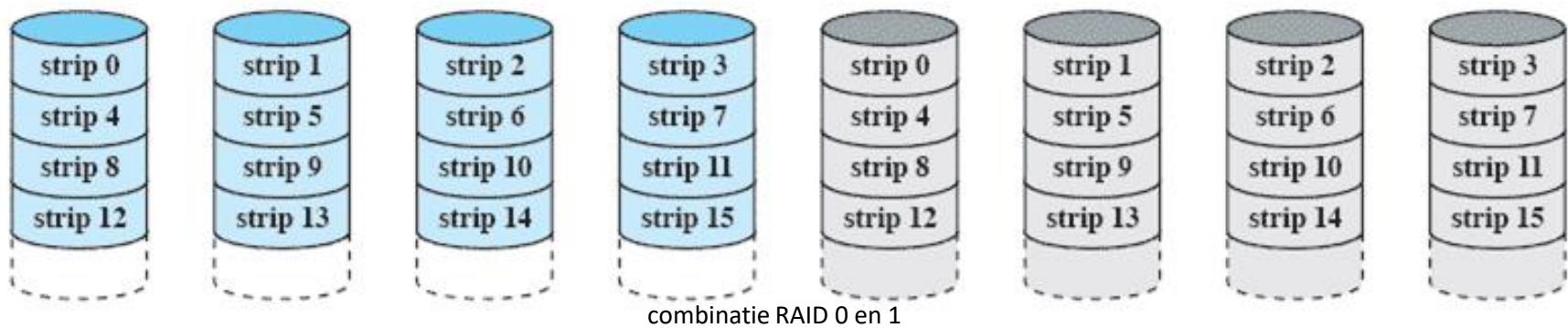
- verdeel de schijven in kleine delen (strips)
- verspreid de data over de schijven



- voor-en nadeel?
 - voordeel: snellere toegang (parallel)
 - nadeel: fout in 1 schijf ==> alle data verloren

RAID-1 (mirroring)

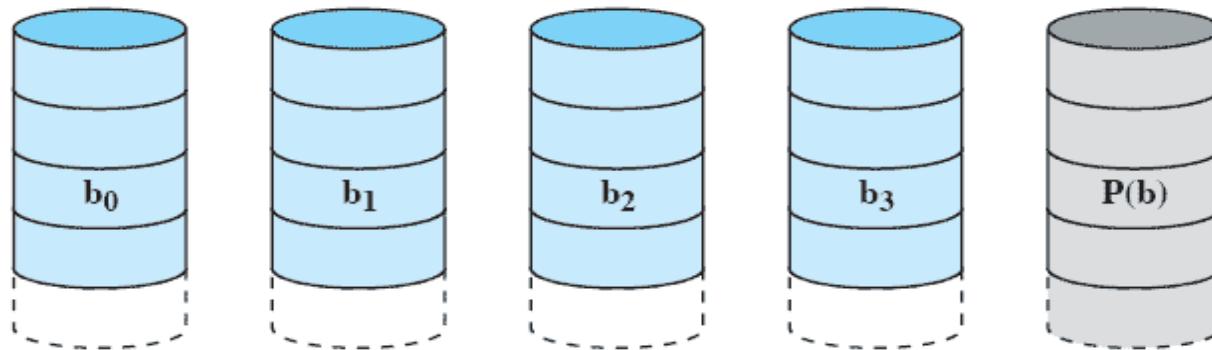
- verdubbel het aantal schijven
- bewaar alle data 2 keer (mirroring)



- voor- en nadeel?
 - voordeel: foutcorrectie mogelijk, snelle leestoegang
 - nadeel: duur (2 x aantal schijven nodig)

RAID-3

- gebruik 1 schijf voor "pariteitsbits" (redundancy)



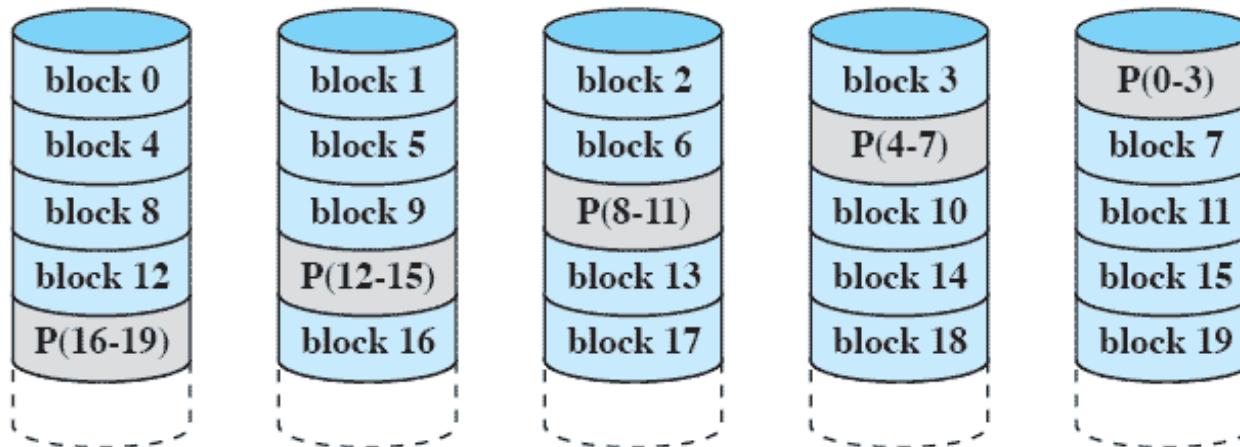
- voorbeeld: $P(b) = b_0 \oplus b_1 \oplus b_2 \oplus b_3$ (**even pariteit**)
- stel: schijf 2 is stuk en er wordt het volgende gelezen:
 - 1010 1101, 1010 1100, xxxx xxxx, 0010 0000, 1001 0101
 - Wat was de waarde van b_2 ?

RAID-3

- voor- en nadelen?
 - voordeel:
 - goedkoper dan RAID-1
 - mogelijkheid tot correctie
 - snel lezen
 - nadeel:
 - traag bij schrijven: redundancy staat steeds op dezelfde schijf

RAID-5

- de pariteitsblokken worden gespreid over de schijven



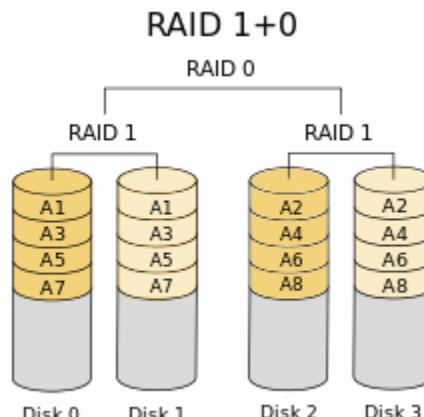
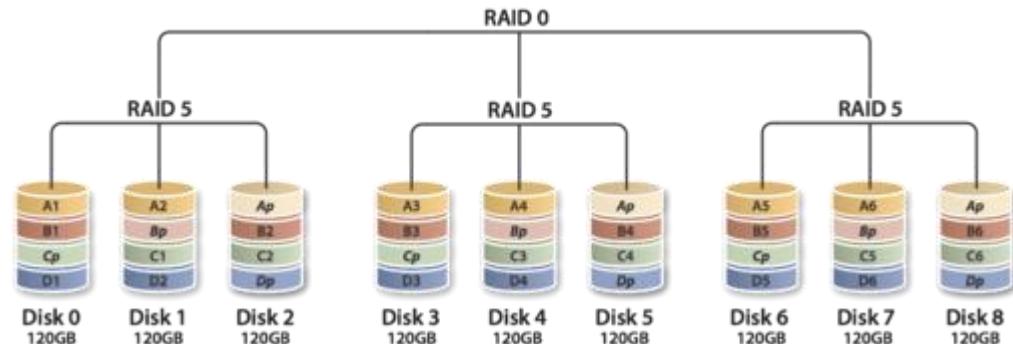
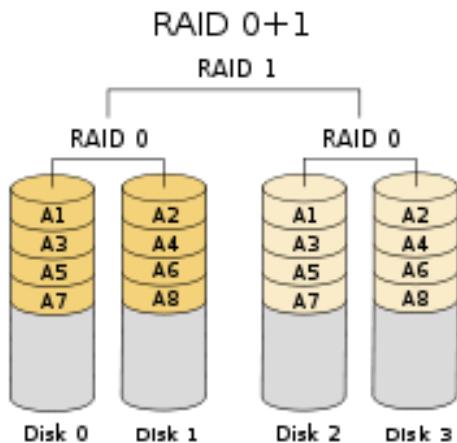
- voordeel: sneller schrijven

RAID-6

- gebruik meerdere schijven voor redundancy
- nadeel:
 - meer schijven nodig
 - berekeningen worden ingewikkelder (Reed-Solomon codes)
- voordeel: meerdere schijven mogen tegelijk stuk gaan

Nested/Hybrid RAID

- RAID 01, RAID 10, RAID 50



RAID is no substitute for back-up!

- All RAID levels except RAID 0 offer protection from a single drive failure. A RAID 6 system even survives 2 disks dying simultaneously. For complete security you do still need to back-up the data from a RAID system.
- That back-up will come in handy if all drives fail simultaneously because of a power spike.
- It is a safeguard when the storage system gets stolen.
- Back-ups can be kept off-site at a different location. This can come in handy if a natural disaster or fire destroys your workplace.
- The most important reason to back-up multiple generations of data is user error. If someone accidentally deletes some important data and this goes unnoticed for several hours, days or weeks, a good set of back-ups ensure you can still retrieve those files.

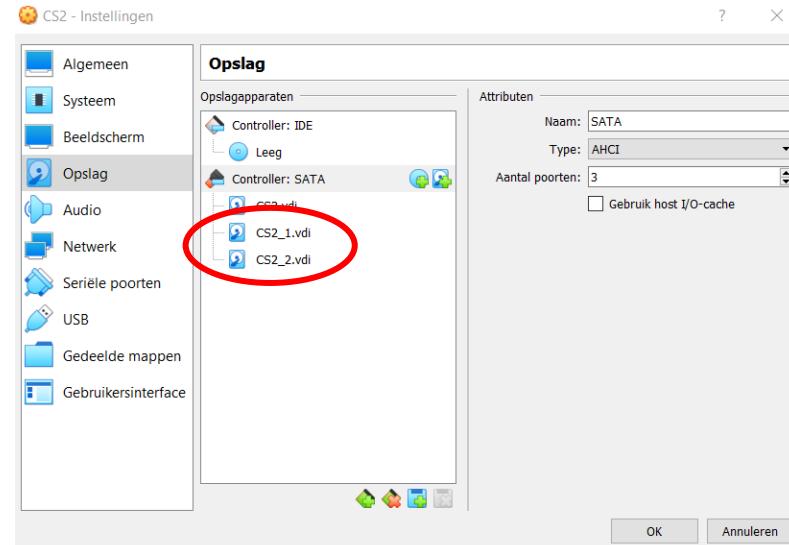
RAID lab

1. Voeg 2 (virtuele) disken toe aan je Linux virtuele machine

2. Configureer RAID1:

<https://www.linuxbabe.com/linux-server/linux-software-raid-1-setup#>

- gebruik GPT partitie tabellen i.p.v. MBR
- welke raid levels worden ondersteund?
- simuleer uitvallen van 1 disk



1. Partitioneren

- fdisk

2. RAID opzetten

- mdadm --create /dev/md0 --level=mirror --raid-devices=2 /dev/sdb1 /dev/sdc1
- mdadm --detail /dev/md0
- mdadm --examine /dev/sdb1 /dev/sdc1

3. Formatteren

- mkfs.ext4 /dev/md0

4. Mounten

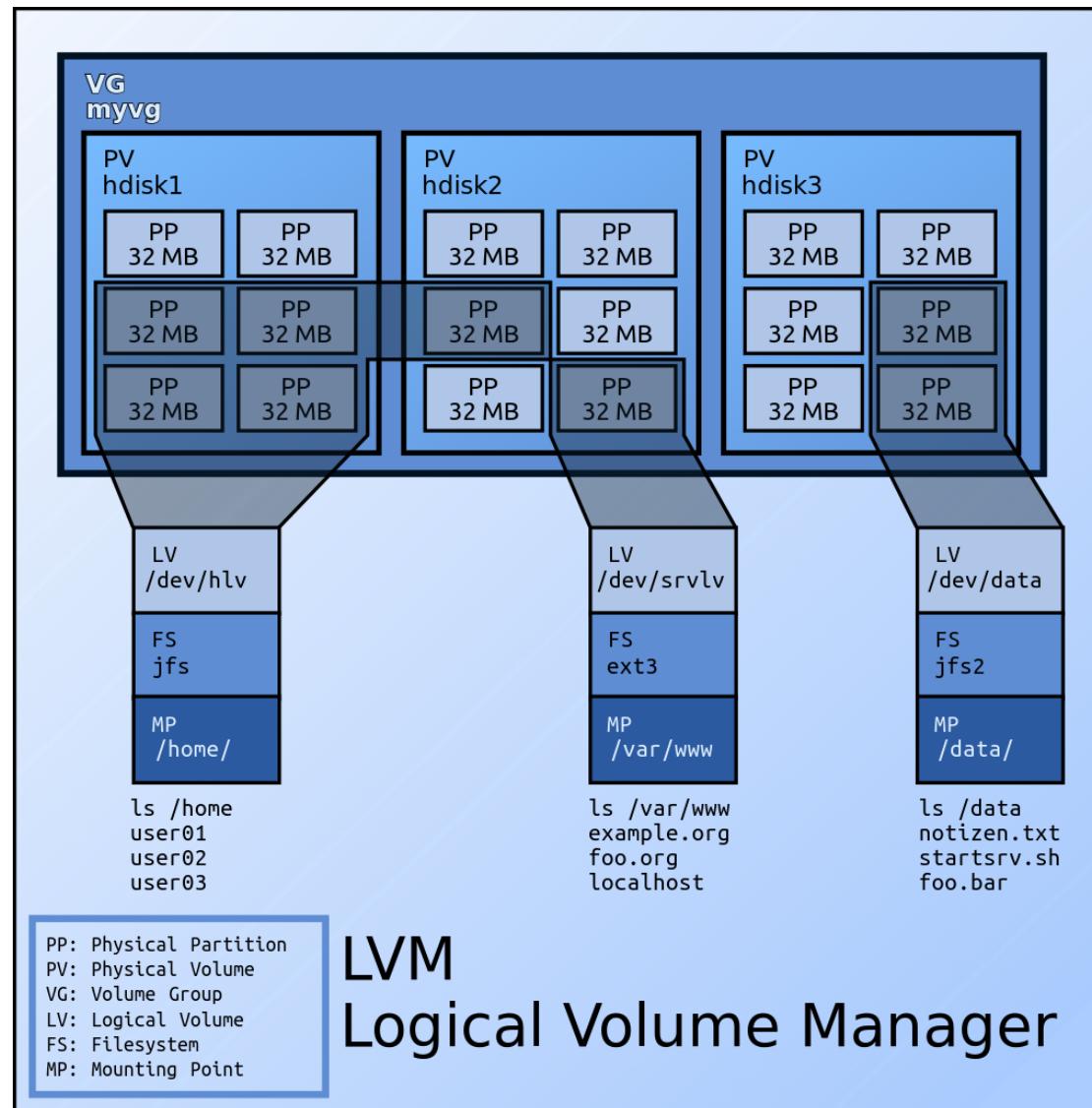
- mount /dev/md0 /mnt/raid1



Logical volume management

- LVM
 - logical volumes (=logische partitie) kunnen meerdere fysische partities bevatten
 - dynamisch resizen van volume
 - snapshots van volumes
 - encryptie van volumes
 - RAID functionaliteit

Logical volume management



Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

File management

- = deel van OS dat boven disk i/o draait
- definieert
 - bestanden
 - directories (folders, mappen)
 - links (shortcuts)
 - vuilbak
 - meta-informatie (file attributes)

Bestandsattributen

- eigenaar van het bestand
- lees/schrijfrechten
 - voorbeeld Unix: rwx rwx rwx
- is bestand gecomprimeerd?
- moet bestand gebackupt worden?
- tijdstip creatie, laatste wijziging, laatste toegang
- met welke applicatie moet dit bestand geopend worden?
- zijn er verschillende versies van dit bestand?
- ...

Interne fragmentatie

- bestanden worden bewaard in blokken op de schijf
- voorbeeld
 - blokken van 4096 bytes *
 - bestandsgrootte van 10 000 bytes
 - 3 blokken nodig
 - laatste blok is niet volledig gevuld
 - = interne fragmentatie
- hoeveel bytes gaan er gemiddeld verloren per bestand?
 - voorbeeld: blokken van 4096 bytes, 1 000 000 bestanden op schijf ==> hoeveel plaats gaat er verloren?
- besluit: maak blokgrootte zo klein mogelijk



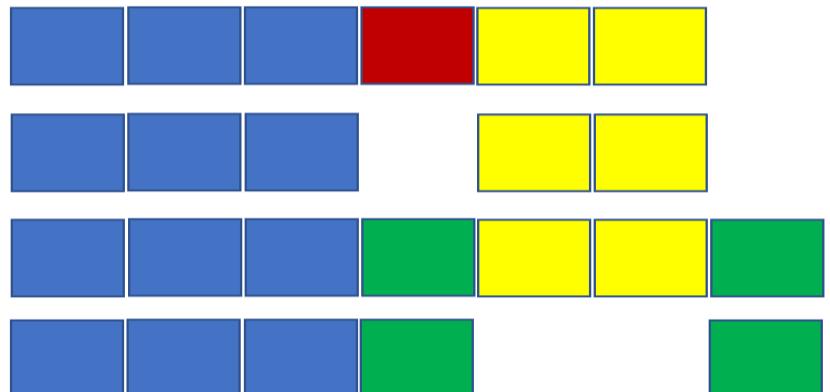
* Blokgrootte is altijd een veelvoud van 512 byte. Waarom? De blokgrootte kan meegegeven worden bij het formatteren, zie man mkfs.ext4

Voorbeeld: FAT filesystem

- FAT16
 - FAT16 kan schijven indelen in hoogstens 65536 blokken
 - blokken kunnen maximaal 65536 bytes groot zijn
 - ==> maximale diskcapaciteit = 4 GiB
 - stel: 10 000 bestanden
 - ==> door interne fragmentatie gaat 327,68 MB verloren
- FAT32
 - heeft zelfde max blokgrootte
 - maar nu 4 miljard blokken mogelijk

(Externe) Fragmentatie

- bestanden worden geschreven en terug verwijderd
 - er ontstaan dus "gaten"
 - bestanden kunnen verspreid zijn over niet-aangrenzende blokken
- sequentieel lezen van bestand: kop HDD heen en weer ==> traag
- beter: grote bestanden in aansluitende blokken
- oplossing: defragmentatie*
- hoe groter de blokgrootte, hoe minder externe fragmentatie
 - vs interne fragmentatie...



* Niet bij SSD om degradatie cellen te voorkomen. Fragmentatie kan bij SSD veel minder/geen kwaad. Waarom?

Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

De allocation table

- Er is een soort inhoudstafel nodig voor de bestanden
 - op welke blokken staat het bestand?
 - **file allocation table (FAT)**
- Twee mogelijkheden:
 - Eén per filesysteem: 1 FAT voor alle bestanden (Windows)
 - Eén per file: inode die allocation table bevat (Linux)

FAT 12, 16, 32

- Schijf/partitie bevat
 - blok met bootsector
 - bootcode
 - gegevens over de schijf
 - FAT: 2 keer (voor foutcorrectie)
 - root folder
 - rest van de disk

FAT 12, 16, 32

- schijf van 131 072 bytes
 - 32 blokken van 4096 bytes
 - 1 partitie
- blok 0 = bevat boot sector
- blok 1 = FAT1
- blok 2 = FAT2
- blok 3 = root folder

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

FAT 12, 16, 32

filename	start	length	attributes
readme.txt	4	15 867	...
myApp.exe	6	27 814	...
verslag.doc	16	22 010	...

root directory tabel: bevindt zich in blok 3

FAT

0	0	17
1	0	18
2	0	19
3	0	20
4	5	21
5	8	FFFF
6	7	0
7	10	0
8	9	0
9	FFFF	0
10	13	0
11	0	0
12	0	0
13	14	0
14	15	0
15	31	FFFF

array of int (12, 16, 32)
met evenveel elementen
als blokken in het filesysteem;
bevindt zich in blok 1 en 2

Blokken met nullen worden niet door files
gebruikt en zijn dus beschikbaar voor nieuwe
files (behalve 1ste vier, waarom?)

FAT voor- en nadelen

- voordeel: heel eenvoudig
- nadeel: heel traag
 - voor iedere blok die geschreven wordt, moet de FAT aangepast worden
 - de FAT moet 2 keer geschreven worden (mirroring)
 - kop moet heel de tijd heen en weer
 - oplossing: caching
 - hou FAT in memory en schrijf af en toe weg naar schijf

FAT oefening

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

- stel dat het OS een bestand wil bijmaken
 - testje.txt
 - 10.340 bytes
- wat moet het OS doen?

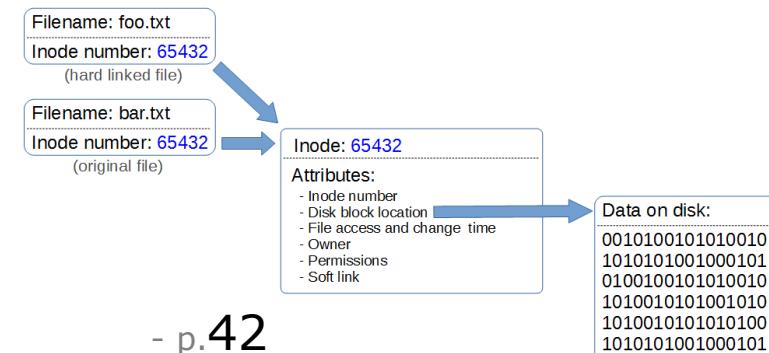


Inhoud

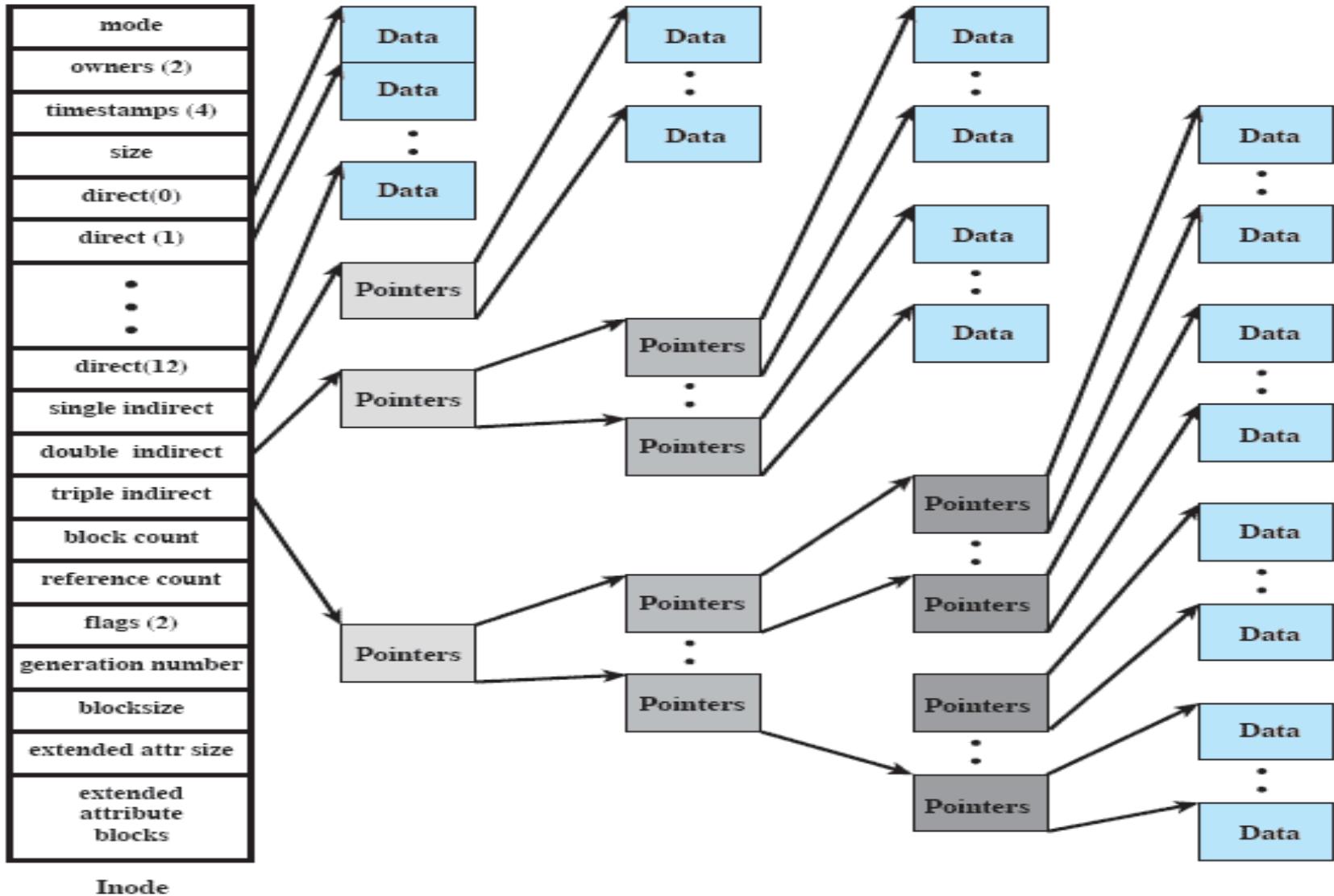
- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

Linux inodes

- Unix heeft een inode per bestand (=datastructuur)
- directory is blok gevuld met pointers naar inodes
- inode bevat filename, lengte, ...
- inode bevat 16 'pointers'
 - eerste 13 wijzen naar de eerste 13 blok van het bestand
 - de 14e verwijst naar een 'single indirect block'. Die bevat pointers naar de volgende sectoren van het bestand
 - de 15e verwijst naar een 'double indirect block'. Die bevat pointers naar single indirect blocks
 - de 16e verwijst naar een 'triple indirect block'. Die bevat pointers naar double indirect blocks



Linux inodes



Inode

Inodes

- stel: blokgrootte van 512 bytes, pointers worden opgeslagen als 32 bit unsigned integers
 - blok kan $512/4 = 128$ pointers bevatten
- hoe groot kan een bestand worden zonder dat er indirects nodig zijn?
 - $13 * 512$
- hoe groot kan een bestand worden zonder dat er double indirects nodig zijn?
 - $13 * 512 + 128 * 512$
- hoe groot kan een bestand maximaal worden?
 - $13 * 512 + 128 * 512 + 128^2 * 512 + 128^3 * 512 \approx 1 \text{ GiB}$
- Oefening: maak de berekening opnieuw voor een blokgrootte van 4096 bytes



Blokgrootte filesysteem

- De blokgrootte van een filesysteem bepaalt:
 - Hoeveelheid interne fragmentatie
 - Impact van externe fragmentatie
 - Maximale grootte van het filesysteem
 - Maximale grootte van een file
 - Performantie van het filesysteem
- Keuze van de blokgrootte:
 - Afwegen van wat belangrijk is
 - Afhankelijk van soort files op het filesysteem
 - Snelheid <-> verlies
 - Maximale grootte file/filesysteem

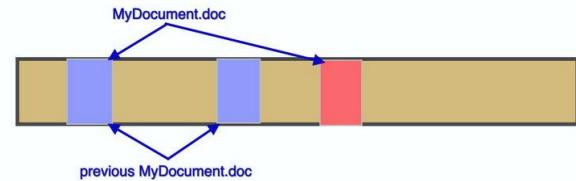
```
mkfs.ext4 -b block_size
mkfs.vfat -s sectors_per_block
```

Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

Next gen filesystems

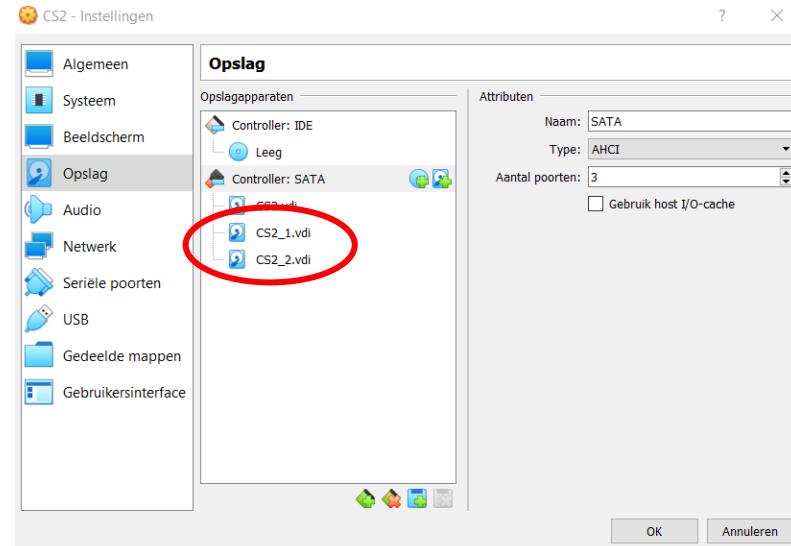
- ZFS: Zettabyte File System
- Btrfs: B-tree FS, Better FS
- combination of filesystem and logical volume manager
 - protection against data corruption
 - high storage capacity
 - copy-on-write
 - snapshots and cloning
 - RAID



ZFS lab

1. Hergebruik de 2 (virtuele) disk en je Linux virtuele machine en verwijder de "md raid"
2. Configureer ZFS met dezelfde 2 partities

<https://ubuntu.com/tutorials/setup-zfs-storage-pool>



```
zpool create new-pool /dev/sdb1 /dev/sdc1          # striping
zpool create new-pool mirror /dev/sdb1 /dev/sdc1    # mirroring
zpool create -m /usr/share/pool new-pool mirror /dev/sdb1 /dev/sdc1  # includes mounting
zpool status
zpool destroy new-pool
```

Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

Disklen delen over het netwerk

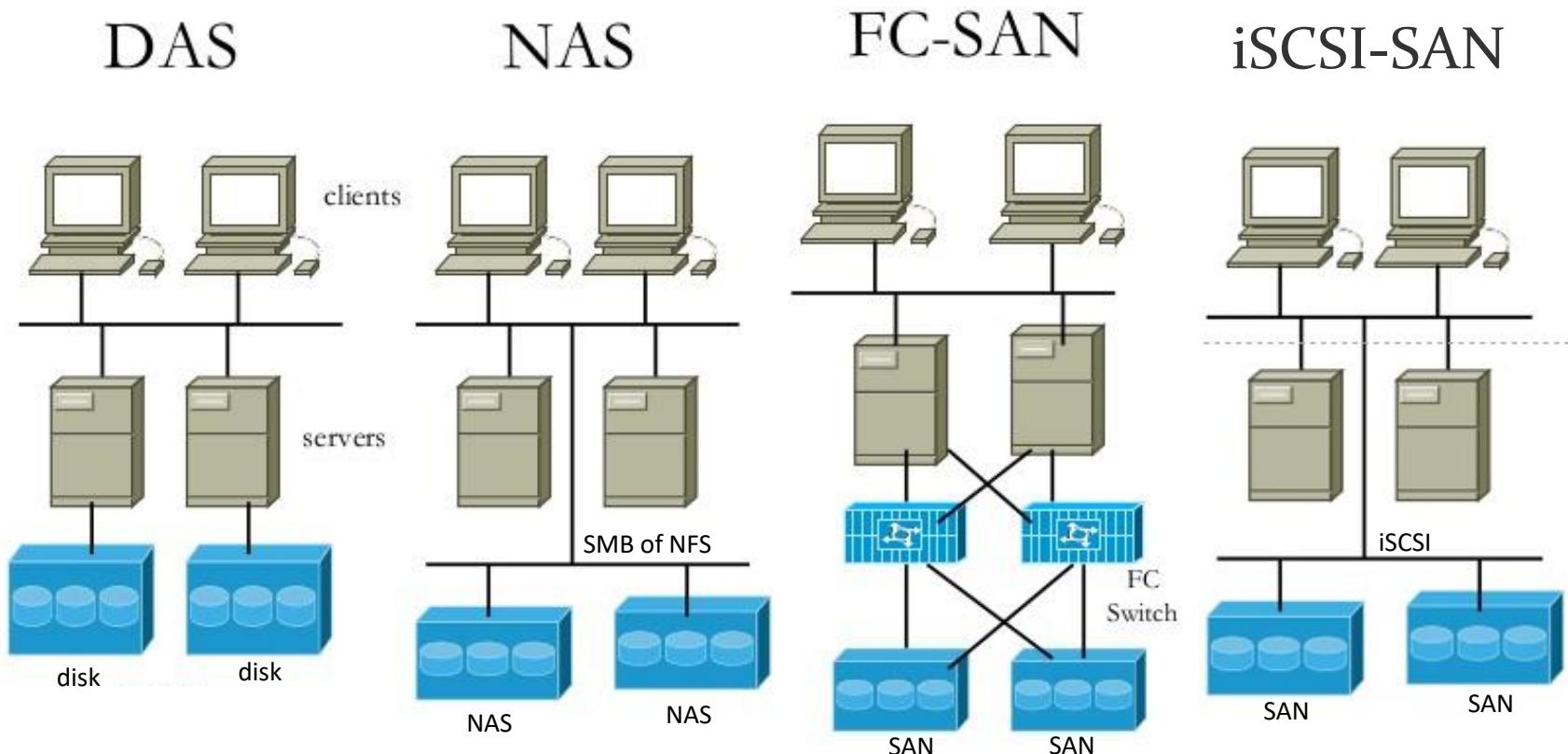
- DAS
 - direct attached storage
 - HDD/SSD zit in computer (of is direct aangesloten aan computer)
- Disklen delen over het netwerk
 - NAS: deelt filesystemen over netwerk (“netwerk drives” of “shares”)
 - SAN: deelt blokken over het netwerk

SAN - NAS

- Wat is het verschil tussen SAN en NAS?

	NAS	SAN
afkorting staat voor	Network attached storage	Storage area network
werkt op welk niveau	file-level	block-level
filesystem staat op	NAS box / File server	Computer
gebruikt welk netwerk	gewone TCP/IP netwerk <ul style="list-style-type: none">• SMB (server message block)/CIFS (common Internet filesystem)• NFS (network filesystem)	<ul style="list-style-type: none">• Fibre Channel met SAN switches (storage area network)• iSCSI: SCSI in TCP/IP packetten

SAN - NAS



Inhoud

- HDD en SSD
- RAID
- Bestandsystemen
- FAT Tabel
- Linux Inodes
- Next Gen File Systems
- NAS en SAN
- Herhalingsvragen

Herhalingsvragen

- hoe werkt een HDD? Hoe werkt een SSD?
- wat doet RAID-x? vergelijk RAID-x met RAID-y... Ken de voor- en nadelen.
- Stel: RAID-5 systeem met ... harde schijven (waarvan 1 voor redundantie). Schijf ... gaat kapot. De data op de andere schijven is Reconstrueer de data op de kapotte harde schijf

Herhalingsvragen

- wat is caching (bij harde schijven). Wat is het voordeel? Wat is het nadeel?
- wat is interne fragmentatie in een bestandsysteem? geef een voorbeeld.
- Hoeveel ruimte gaat er aan interne fragmentatie verloren op een harde schijf van ... als je weet dat er ... bestanden op staan en de blokgrootte gelijk is aan ... bytes?
- wat is (externe) fragmentatie van een harde schijf? Hoe ontstaat dit?
- wat is een "file allocation table"?

Herhalingsvragen

- gegeven volgende FAT: ... in welke sectoren staat een bestand dat op sector ... begint?
- gegeven volgende FAT: ... Hoeveel files staan hier op?
- gegeven volgende FAT: ... met een blokgrootte van ... hoeveel plaats is er nog vrij voor data op deze schijf?
- gegeven volgende FAT: ... met een blokgrootte van ... Stel dat je een nieuw bestand van ... bytes wil bijmaken. Teken dan de nieuwe FAT.
- gegeven een unix file system met een blokgrootte van ... Wat is de maximale grootte van een bestand?
- gegeven een unix file system met een blokgrootte van ... Wat is de maximale grootte van een bestand als je geen (dubbel/triple) indirects mag gebruiken?

Herhalingsvragen

- Hoe bepaal je de blokgrootte van een filesystem?
- Wat is LVM?
- Wat doen volgende commando's: fdisk, mdadm, mkfs, mount
- Wat zijn eigenschappen van next gen filesystems?
- Wat doet het zpool commando?
- Wat is het verschil tussen SAN en NAS?

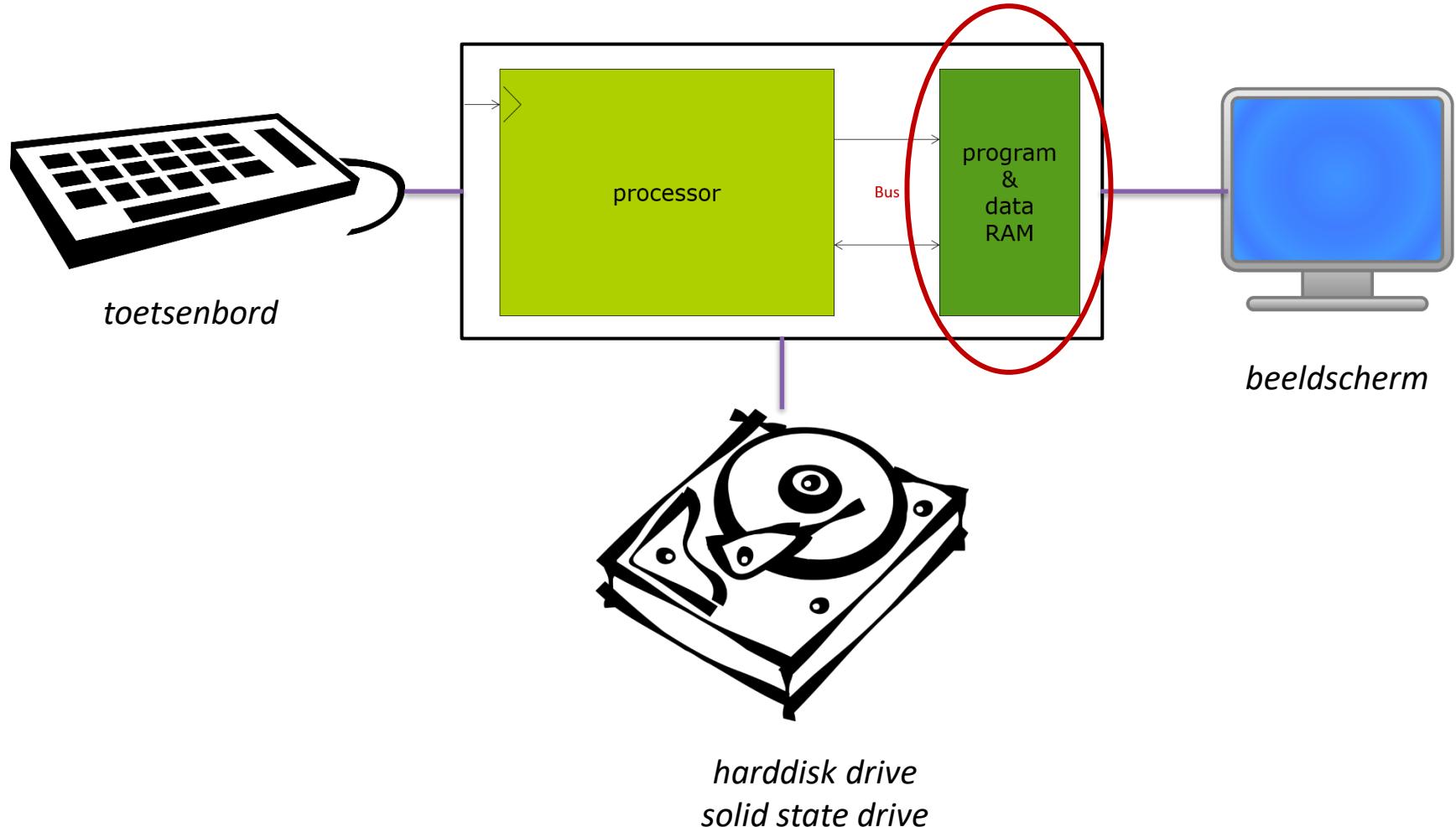
Computersystems 2

Theorie

5. Geheugenbeheer

Geheugenbeheer

Hoe wordt bijgehouden waar een programma in het RAM geheugen zit?
Hoe gebeurt vertaling van logisch naar fysisch adres?



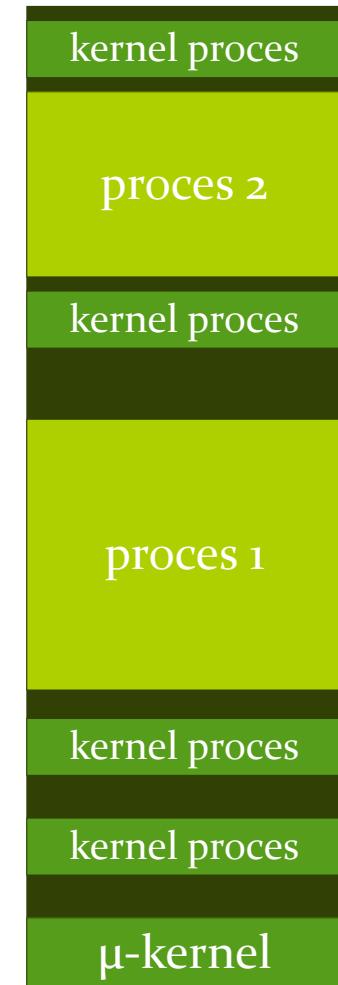
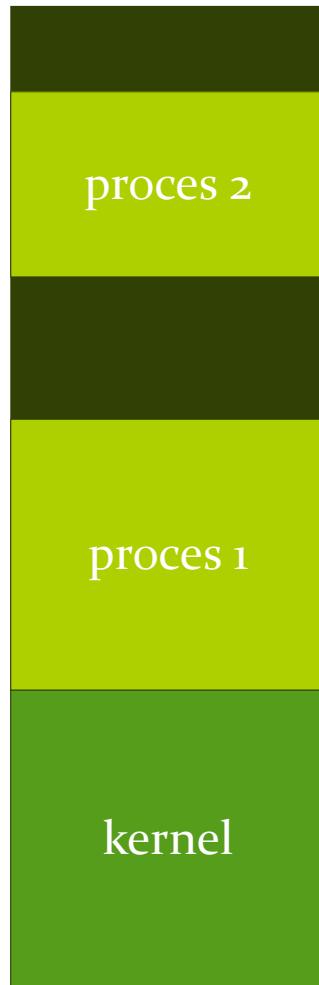
Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Laden van het OS

- boot-loader zal OS laden in het geheugen
- dit kan dankzij Von Neumann architectuur
- verschillende strategieën
 - monolytic kernel
 - modules
 - micro-kernel

laden van het OS



kernel

- monolitische kernel
 - alle OS functionaliteiten in **1 block** of code (1 proces)
 - bij **verandering** (bv. nieuwe device driver): kernel moet opnieuw gelinked worden en het system moet rebooten
- modulaire kernel
 - kernel is **1 proces** dat **dynamische gelinked** (zie verder) is
 - kernel modules (bv. device driver) kunnen **at runtime** gelinked (of unlinked) worden
- micro-kernel
 - micro-kernel bevat **basis OS** functionaliteit
 - rest: "drivers" als **aparte processen** (bv. voor HDD, scherm, netwerk, file management)
 - processen praten met elkaar via IPC (zie later)

Ubuntu kernel modules

1. Hoe noemen de loadable kernel modules bij Ubuntu Linux (welke extensie)?
2. Waar staan deze?
3. Met welk commando kan men zien welke loadable kernel modules geladen zijn?

Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Programma's laden

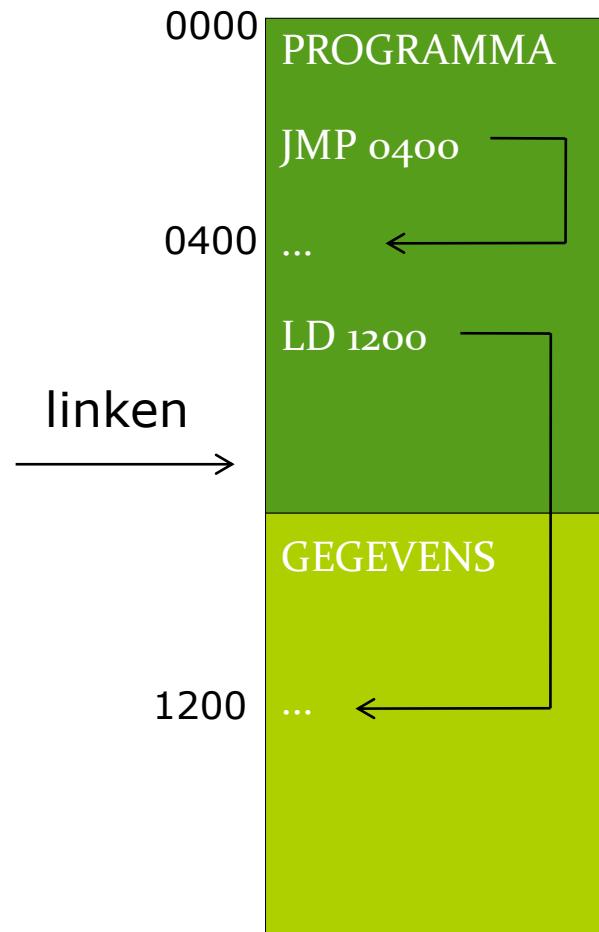
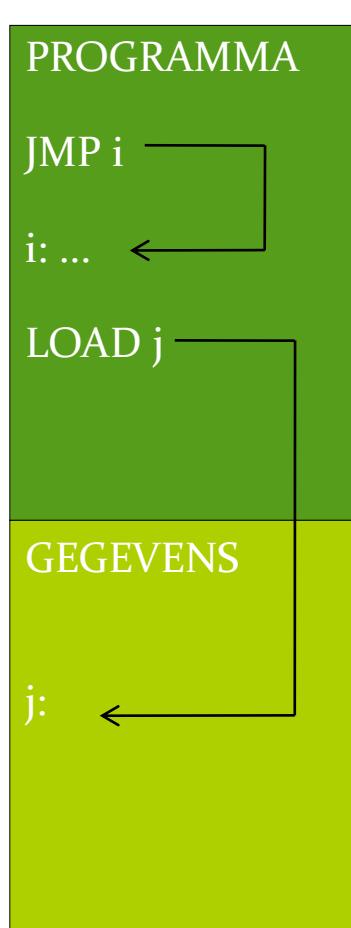
- OS start op, zit klaar in geheugen
- I/O drivers zijn geladen
- nu: programma opstarten
 - laad code/data in geheugen
 - spring naar start
- probleem: programma kan op verschillende plaatsen in het RAM geheugen terecht komen
 - wat als er een jmp instructie staat?
 - waar zal de data in het geheugen terecht komen?

Assembler	#	Hex	Interpretatie
li r0, 4	; 00	1040	Immediate: Laad getal 4 in reg r0
jp r0	; 01	c000	PC: Spring naar code op plaats r0
li r1, 42	; 02	12a1	Immediate: Laad getal 42 in reg r1
li r2, 32	; 03	1202	Immediate: Laad getal 32 in reg r2

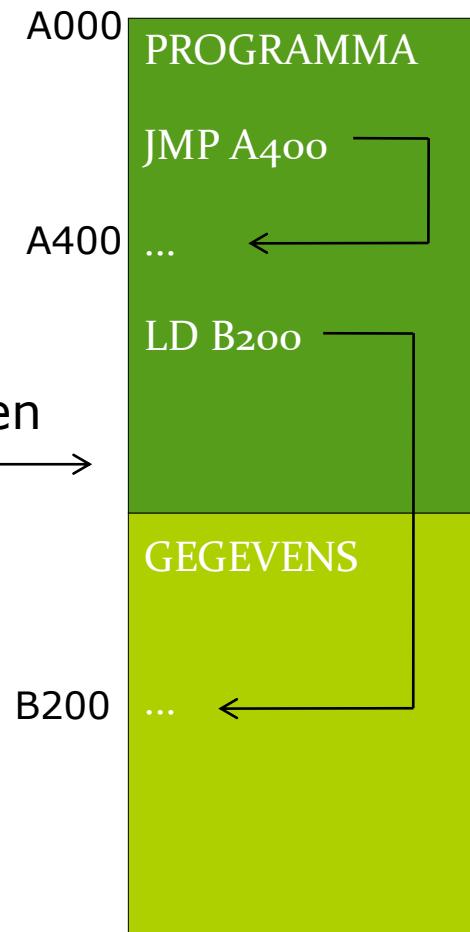
Laden - Relocatie

- processen moeten op verschillende plaatsen kunnen staan
- soms worden processen verhuisd
- ==> geheugenreferenties moeten veranderen
- **logische adressen en fysieke adressen**
 - logische adressen zijn relatief t.o.v. begin proces
 - fysieke adressen zijn absoluut (t.o.v. begin geheugen)
 - software maakt altijd gebruik van logische adressen
 - vertaling gebeurt door hardware

Een proces laden



linken



laden

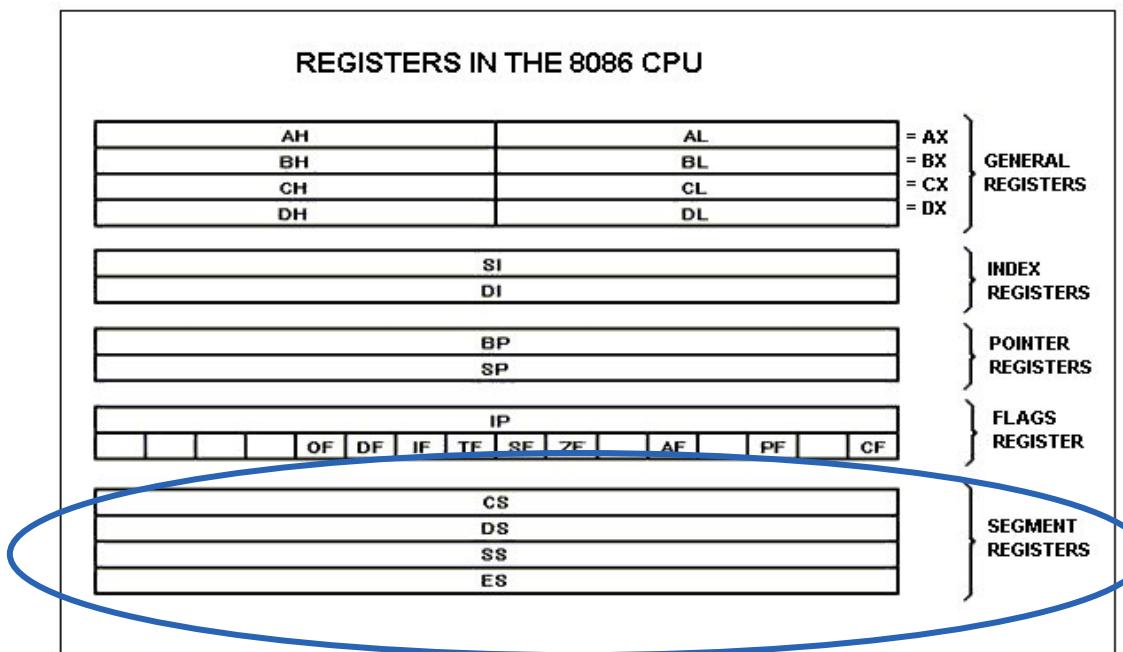
Symbolisch (met labels)

Logische adressen

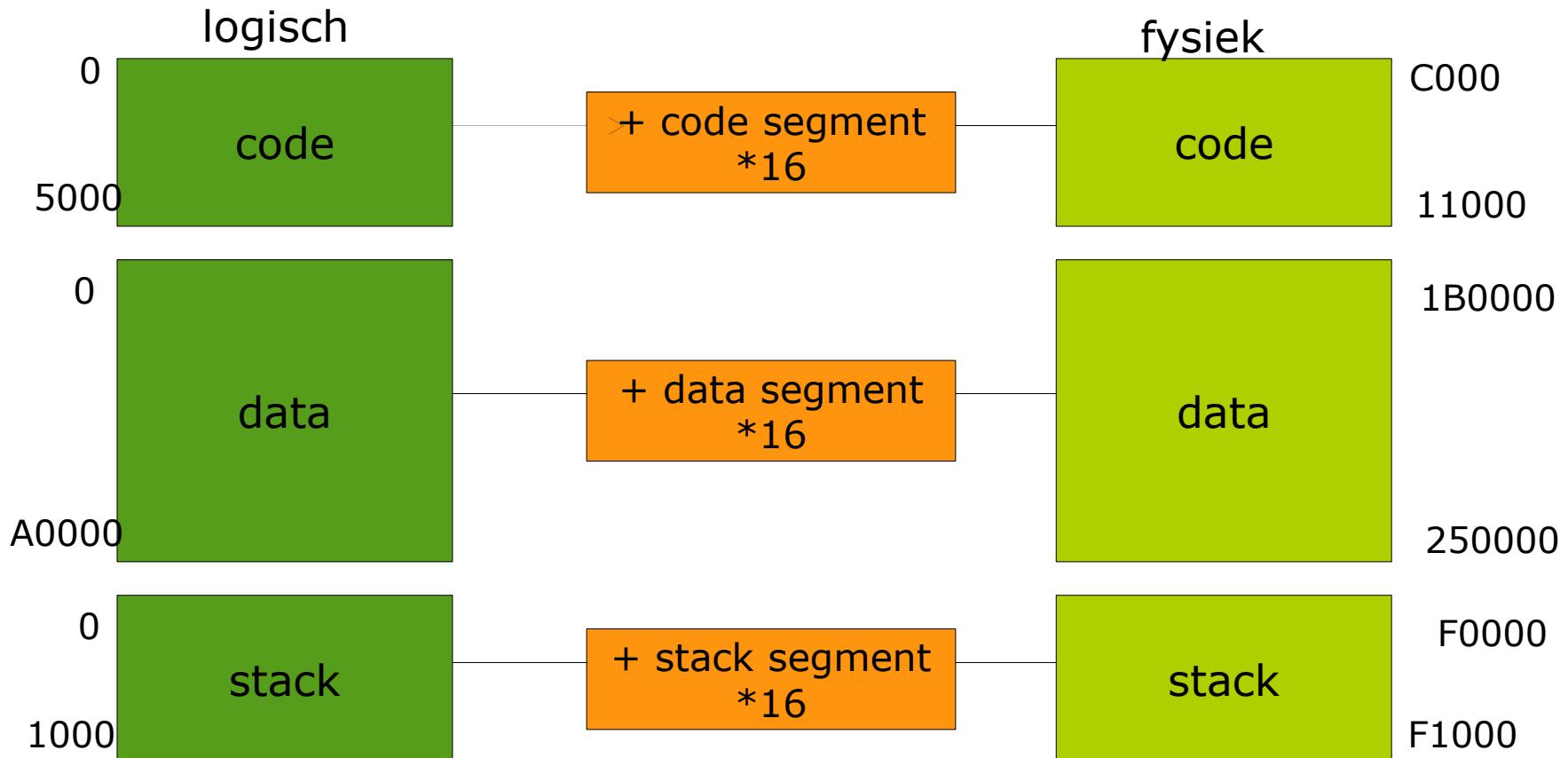
Fysische adressen
= logisch adressen + start adres

Vb. Relocatie 8086

- voor de start adressen worden de **Segment Registers** gebruikt
- segment registers worden * 16 vermenigvuldigd omdat registers 16 bit zijn en de adres bus 20 bit is
- gebeurt per segment (zie volgende slide): code, data (globale variabelen en heap), stack (lokale variabelen en parameters)



Vb. Relocatie 8086



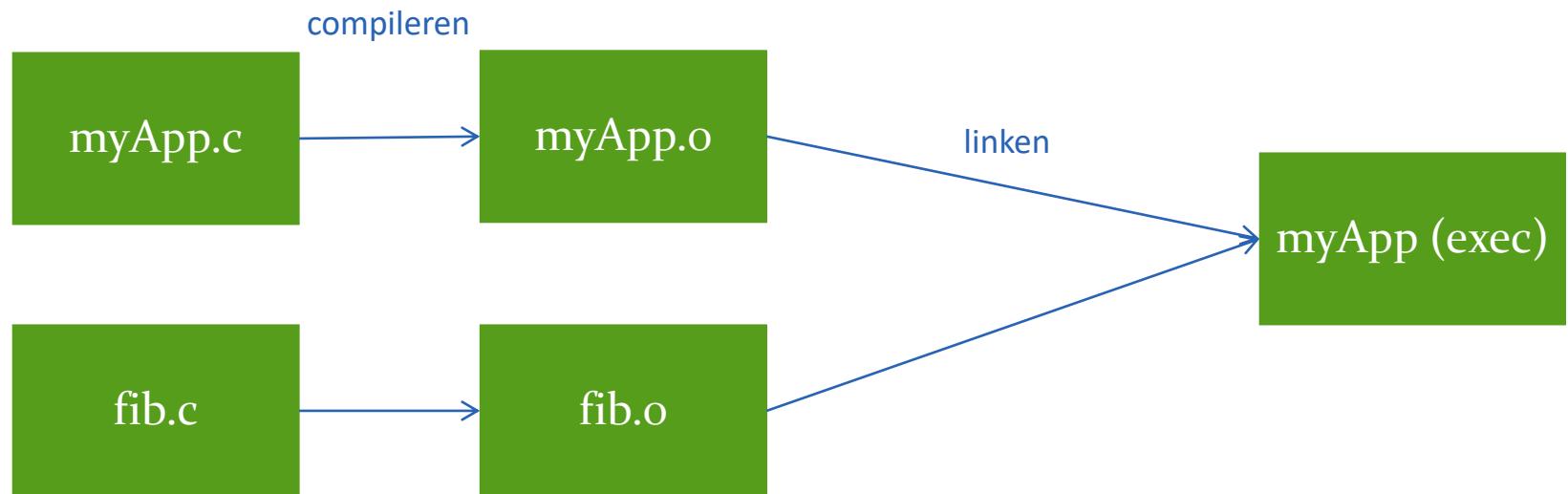
Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Een proces linken

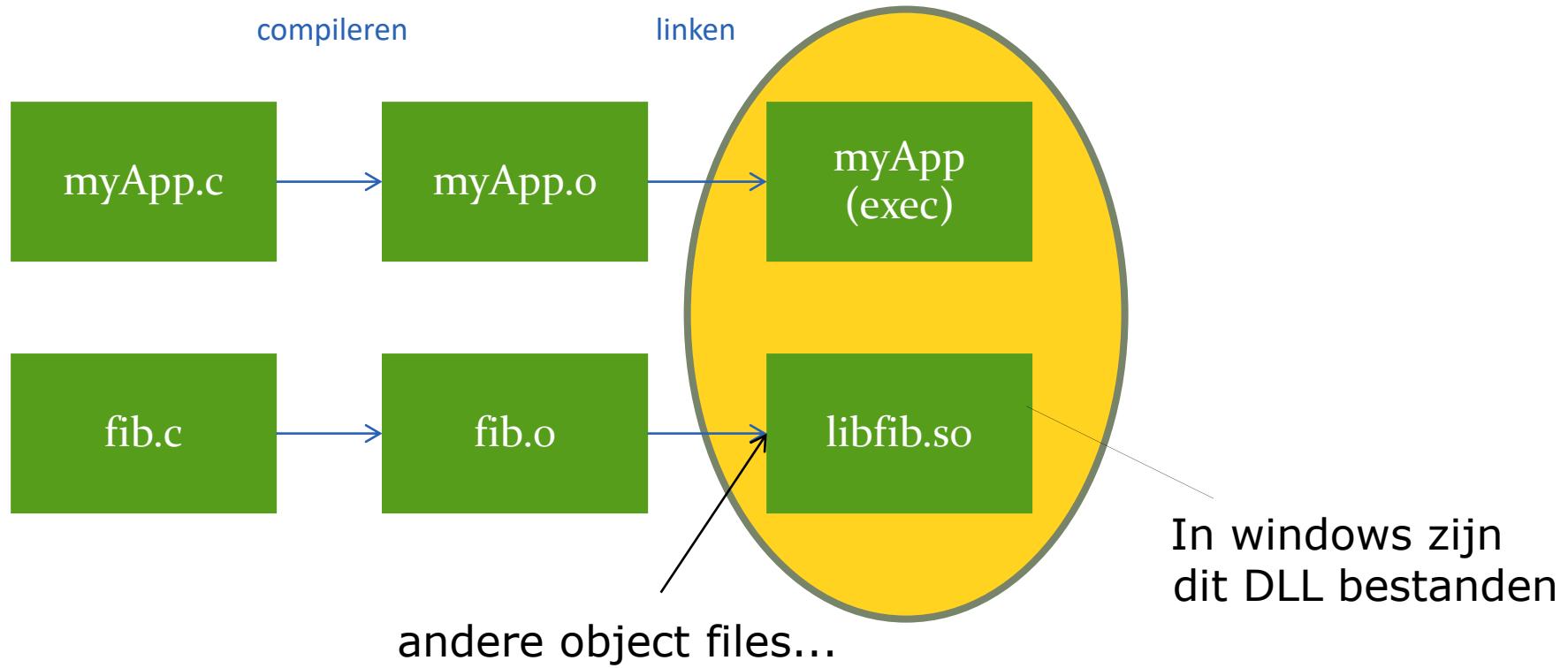
- in de praktijk
 - programma bestaat uit # modules
 - iedere module wordt gecompileerd naar object-file
 - object-file bevat code+labels
 - linker voegt object-files samen tot executable
 - linker vervangt labels door adressen
- 2 mogelijkheden
 - Statisch gelinkt
 - Dynamisch gelinkt

Statisch linken



myApp.c gebruikt een functie uit module fib.c

Dynamisch linken (1)



.so: shared object

Dynamisch linken (2)

Voordelen:

- Bij nieuwe versie module dient exe niet opnieuw gecompileerd te worden
- Koppelen tijdens uitvoering ("dynamic" linking): als programma module **niet gebruikt** wordt deze ook **niet in geheugen** geladen
- Dynamische modules kunnen **gedeeld** worden, slechts 1 maal in geheugen
- Functionaliteit uitbreidbaar: tekenprogramma gebruikt nieuwe plotter die nog niet bestond toen programma werd geschreven

Compile lab

- Canvas: myApp.c en fib.c
- bekijk de inhoud van beide files
- installeer de GNU C-compiler (gcc) indien nog niet gebeurd (zoek op hoe)
- compileer naar .o bestand
 - gcc -c myApp.c fib.c
 - wat doet -c optie?
- bekijk inhoud
 - objdump -d fib.o
 - wat zie je hier?
- link 2 files naar executable
 - gcc -o myApp_stat myApp.o fib.o
 - wat doet -o optie?
 - run het programma
- compileer naar .so bestand
 - gcc -c -fPIC fib.c
 - gcc -shared -Wl,-soname,libfib.so.1 -o libfib.so.1.0 fib.o
 - ln -s libfib.so.1.0 libfib.so.1
 - ln -s libfib.so.1.0 libfib.so
- Compile myApp met so
 - gcc -L. myApp.c -lfib -o myApp_dyn
 - LD_LIBRARY_PATH=.
 - export LD_LIBRARY_PATH
 - run de dynamische versie
- Bekijk so dependencies
 - ldd myApp_dyn



Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Geheugenbeheer

- een programma bestaat minstens uit:
 - **code**: instructies (machine-code)
 - **data**: globale variabelen, statische data, dynamische variabelen (heap)
 - **stack**: lokale variabelen, parameters, return-values, return addresses

Geheugenbeheer

werking van de **call stack**

```
main() {  
    int a = fac(2);  
}
```

a=2

```
int fac(int i) {  
    int result = 0;  
  
    if (i < 2) result=1;  
    else result=fac(i-1)*i;  
  
    return result;  
}
```

stack groeit naar onder

Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Geheugenbeheer

- geheugenbeheer houdt (o.a.) bij welke delen van het geheugen in gebruik zijn en welke niet
- verschillende strategieën
 - partitionering
 - segmentering
 - paging

Partitionering



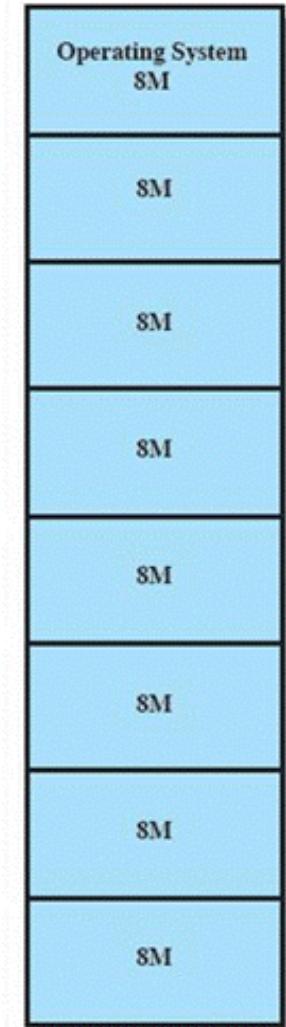
LET OP

partitie HDD != RAM geheugen partitie

- Partities
 - deel RAM geheugen op in 'partities' (blokken)
 - hou lijst bij van partities die nog vrij zijn
 - zoek partitie waar proces in past en laad proces
- 2 soorten
 - **vaste partities** (fixed partitions)
 - **dynamische partities**

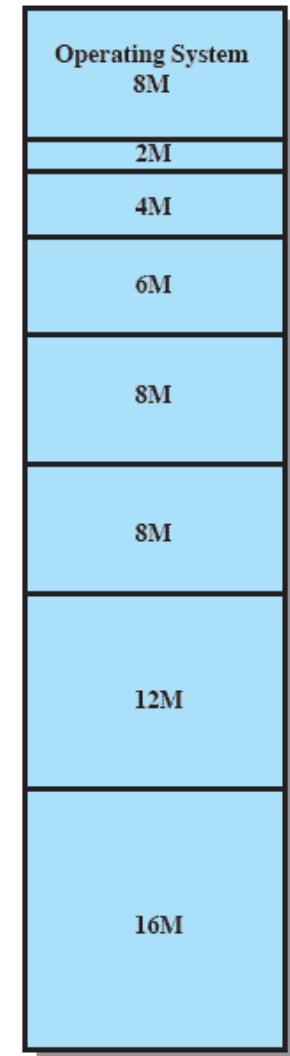
Vaste partities

- gelijke grootte
- wordt vastgelegd bij het opstarten
- voordeel
 - eenvoudig
 - processen kunnen gemakkelijk naar schijf worden geschreven om plaats te maken voor andere
- nadeel
 - proces mag niet groter zijn dan partitie
 - max aantal partities is vast
 - er gaat ruimte verloren (interne fragmentatie)



Vaste partities

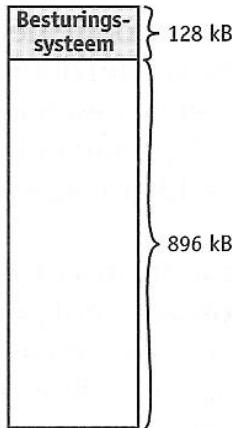
- verschillende groottes
- worden vastgelegd bij het opstarten
- proces krijgt kleinste partitie die groot genoeg is
- voordeel
 - zoals vorige
 - meer mogelijkheden voor grotere processen
- nadeel
 - nog steeds interne fragmentatie
 - max aantal partities nog steeds vast



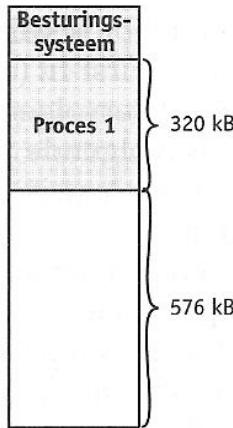
Dynamische partitionering

- OS heeft gelinkte lijst van partities
- Partities dynamisch gecreëerd bij laden van een proces
- Partitie is even groot als proces
- Geen interne fragmentatie

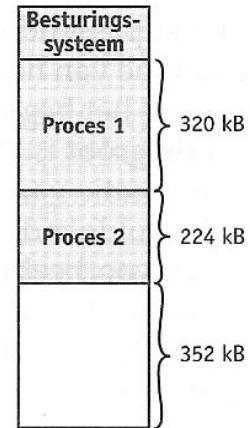
Dynamische partitionering



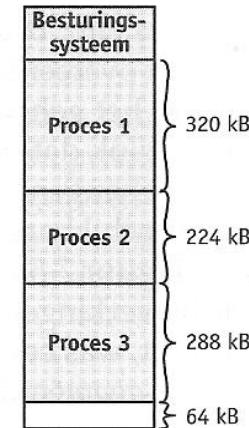
(a)



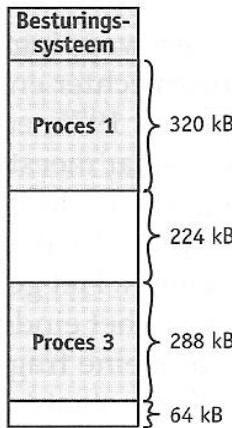
(b)



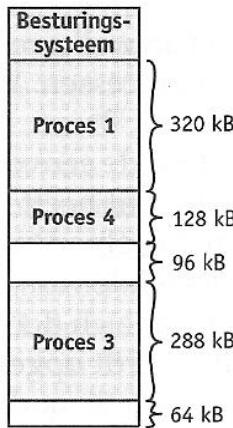
(c)



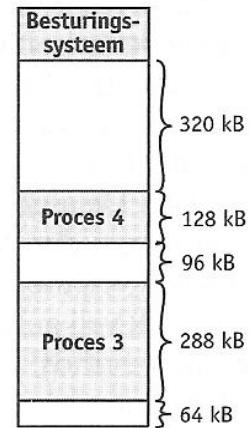
(d)



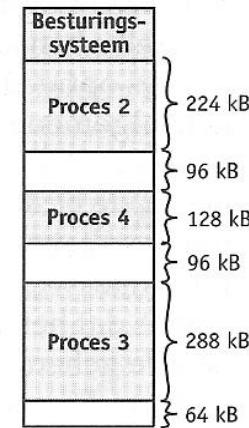
(e)



(f)



(g)

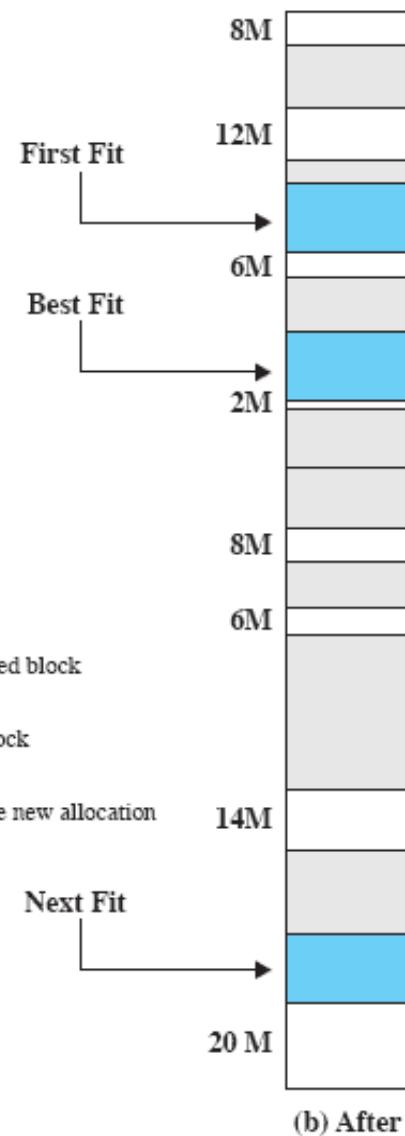
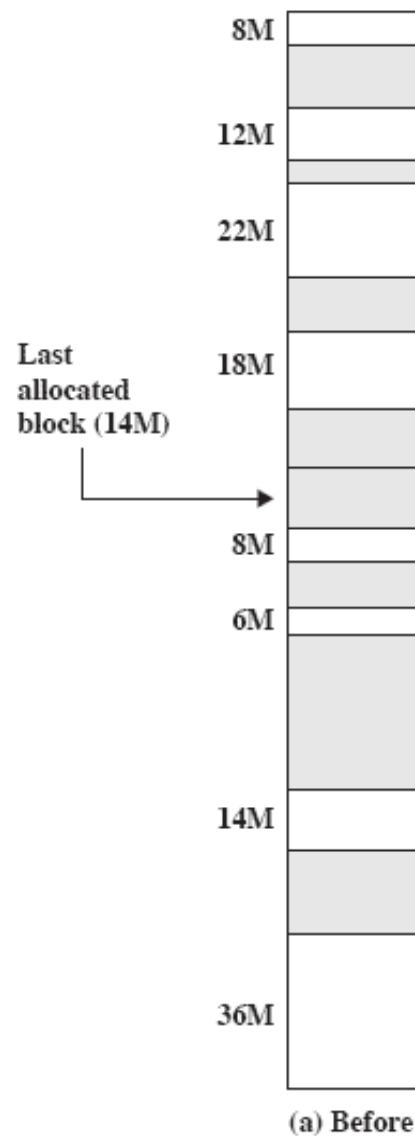


(h)

Dynamische partitionering

- plaatsing van een proces
 - **best-fit**: zoek kleinste partitie die groot genoeg is
 - **first-fit**: zoek eerste partitie die groot genoeg is
 - **next-fit**: zoek vanaf vorige partitie tot partitie die groot genoeg is
- nadeel
 - soms blijven kleine, onbruikbare stukken geheugen over
 - men noemt dit externe fragmentatie
 - oplossing
 - **compaction**: schuif alle processen terug bij elkaar

Dynamische partitionering

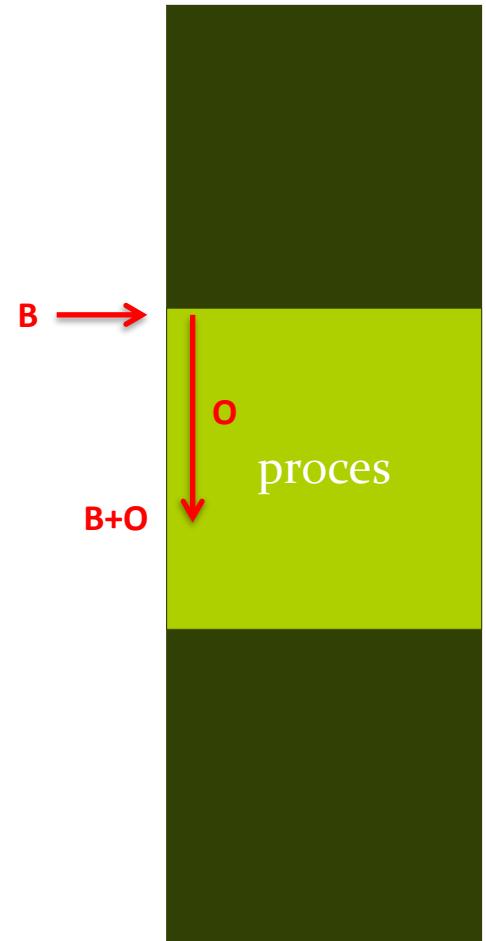


Nieuw proces:
16 MB

Welke van de 3
is het beste
algoritme?

Partitionering

- Proces zit in 1 partitie in het geheugen
- Eén aaneensluitend blok
- Relocatie door:
 - Beginadres **B** van partitie
 - Logisch adres **O** (offset binnen partitie)
 - **fysisch adr = logisch adr + beginadres**
- Interne (vaste) of externe (dynamische) fragmentatie
 - Niet meer gebruikt door moderne OS



Inhoud

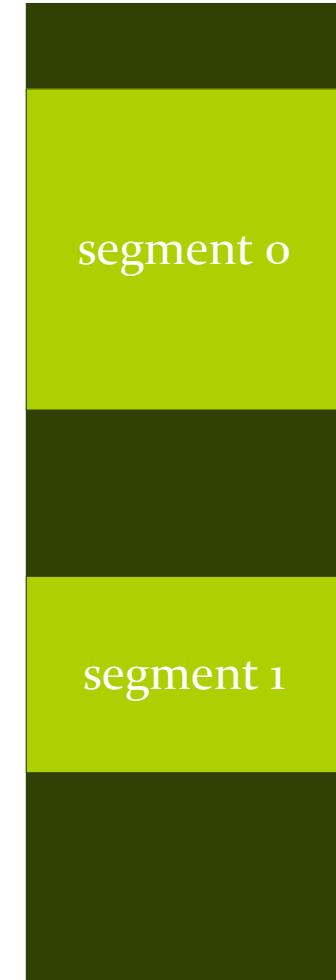
- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Overzicht

	Proces is één aaneengesloten blok	Verschillende niet noodzakelijk aaneengesloten blokken
Blokken vast bij startup	Vaste partitionering	Paging
Blokken dynamisch	Dynamische partitionering	Segmentation

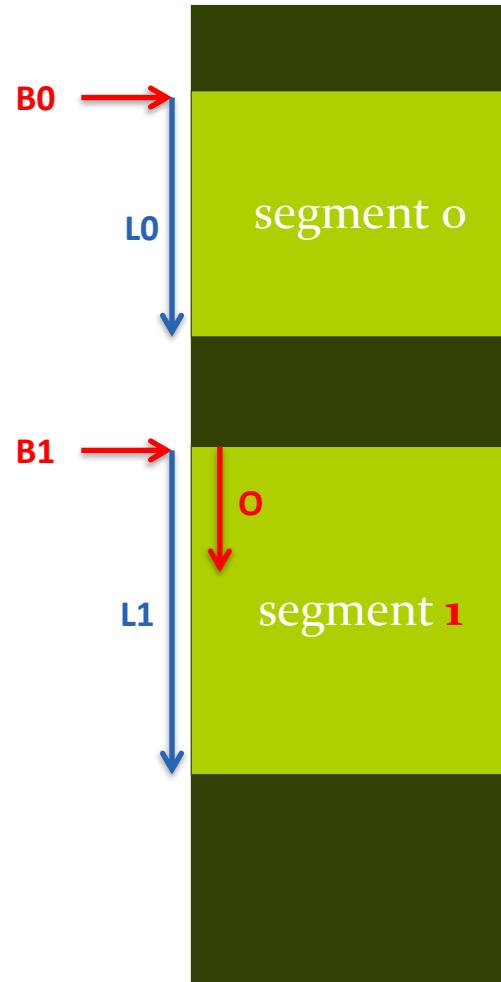
Segmentation

- Dynamische partitionering
 - proces = 1 aaneengesloten partitie
- Segmentation
 - proces kan verspreid zijn over verschillende partities (hier **segmenten** genoemd)
 - Segmenten hoeven niet aaneengesloten te zijn

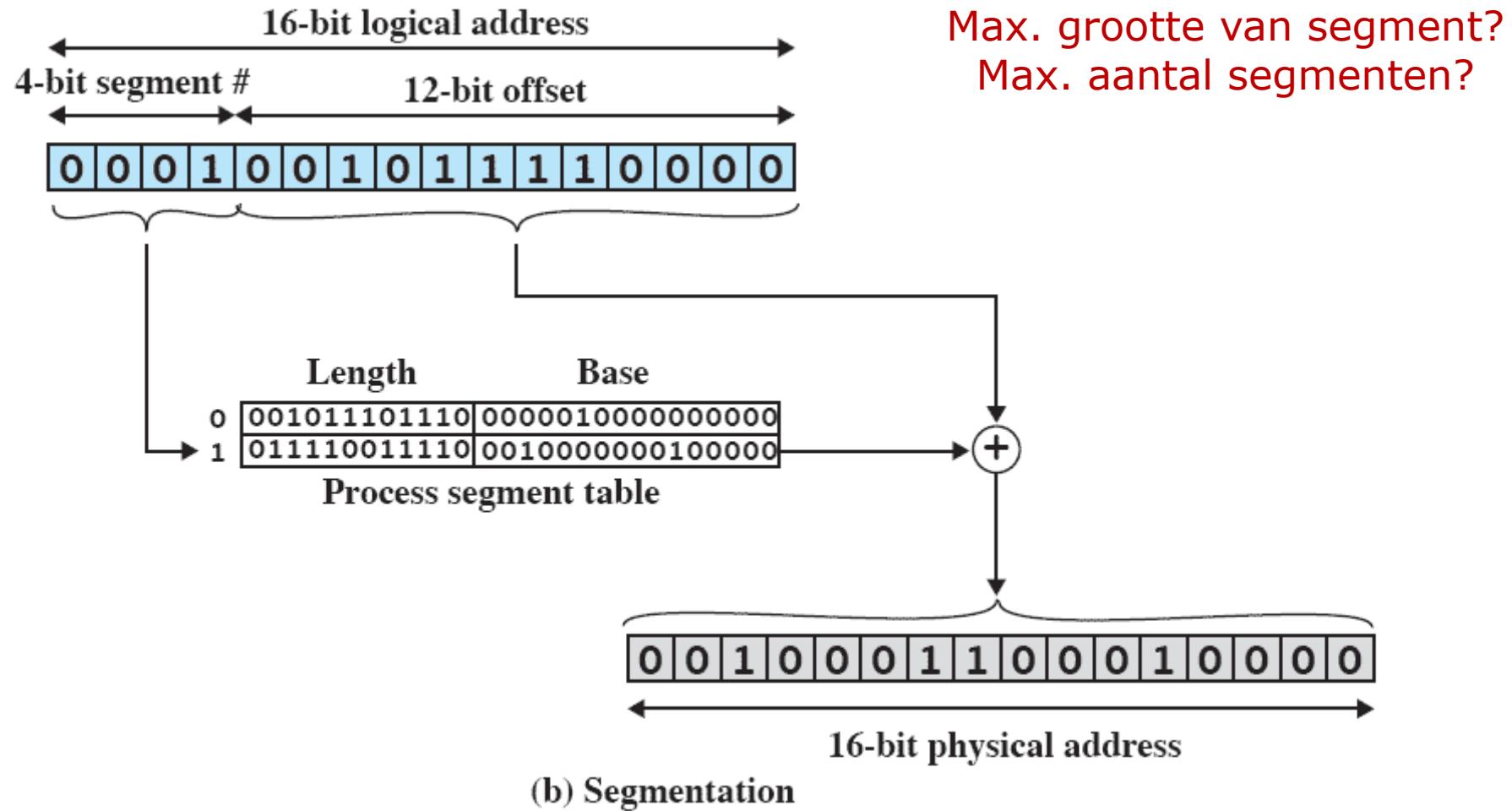


Segmentation

- ieder segment heeft een nummer
- ieder segment heeft een beginadres en een lengte (in **segment table**)
- een logisch adres bestaat nu uit
 - segment nummer
 - offset binnen dat segment
- hardware doet vertaling naar fysiek adres
- bij overschrijden van grenswaarde: interrupt ("**segmentation fault**")



Segmentation



Segmentation

- Oefening:
 - 2 bits voor segmenten
 - segment table zit in het geheugen
(nummer: start, lengte)
 - 00: 0x00010000, 0x00000100
 - 01: 0x0011F000, 0x00100000
 - 02: 0x11000000, 0x00010000
 - 03: 0x20000000, 0x00001000
 - processor wil instructie lezen op adres 0x400FAE23
- Wat is het fysisch adres (hexadecimaal)?
- Wat gebeurt er bij jmp naar 0x4010A003 ?



Oefening thuis

- Voorbeeld: 2 bits voor segmenten
 - logisch adres: 1010 1100 0110 1101
 - segment-table:

segment	start	lengte
00	0000 0010 0110 1100	0000 0010 0001 0000
01	0001 0000 0100 1000	0000 1011 0000 0110
10	1011 0000 0110 1011	0011 0000 0000 0000
11	1001 1001 0110 1010	0000 0000 0010 0000

- offset binnen segment groter dan lengte?



Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Paging

- Net zoals bij vaste partitionering, wordt het RAM geheugen opgedeeld in vaste blokken (die allemaal even groot zijn), **frames** genoemd
- Frame grootte is macht van 2, typische: 1KiB, 4KiB
- Proces kan nu wel verspreid zijn over verschillende frames
- Deze hoeven niet aaneengesloten te zijn



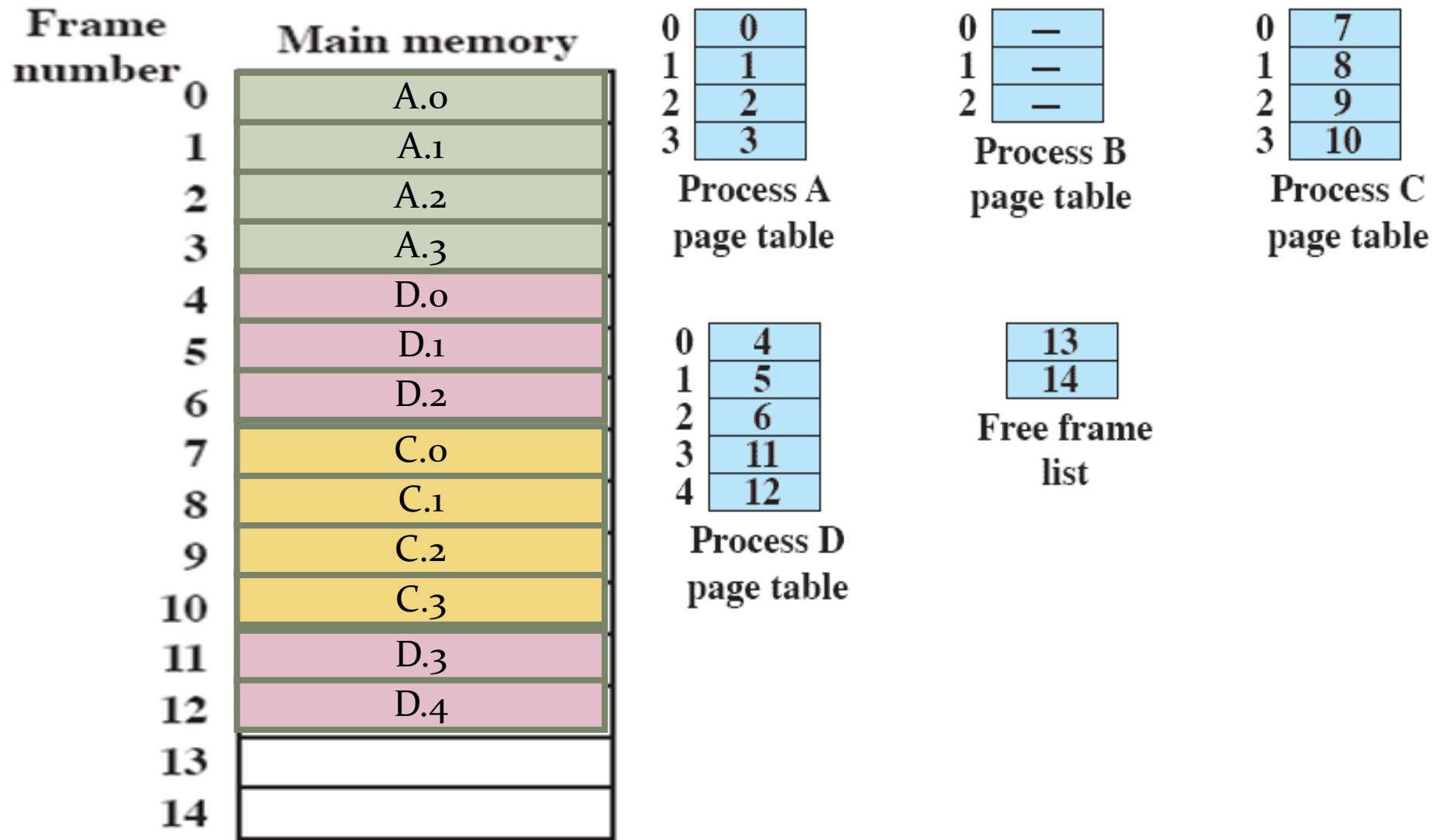
PS: principe is vergelijkbaar met dit van een filesysteem, files zitten ook in vaste blokken, maar nu gaat het over processen in het RAM geheugen

Paging

- Wanneer een programma opgestart wordt (het wordt van de HDD in het RAM geheugen geladen)
 - Proces wordt opgedeeld in blokken, **pages** genoemd, even groot als de frame grootte
 - Deze **pages** worden toegewezen aan vrije frames in het RAM geheugen
 - De **page table** houdt bij in welke frames het proces zit
 - hier: page 0 in frame 2, page 1 in frame 4

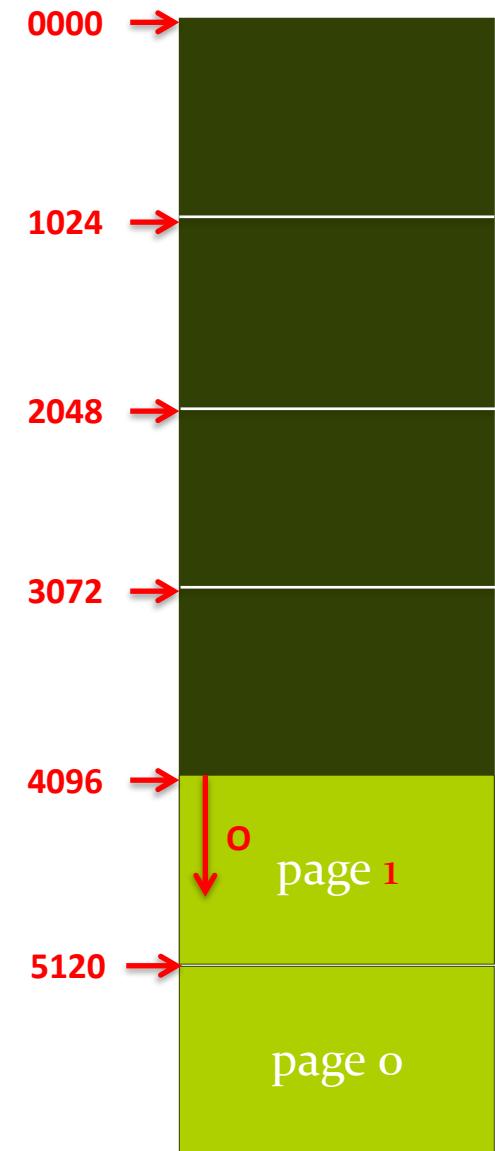


Paging

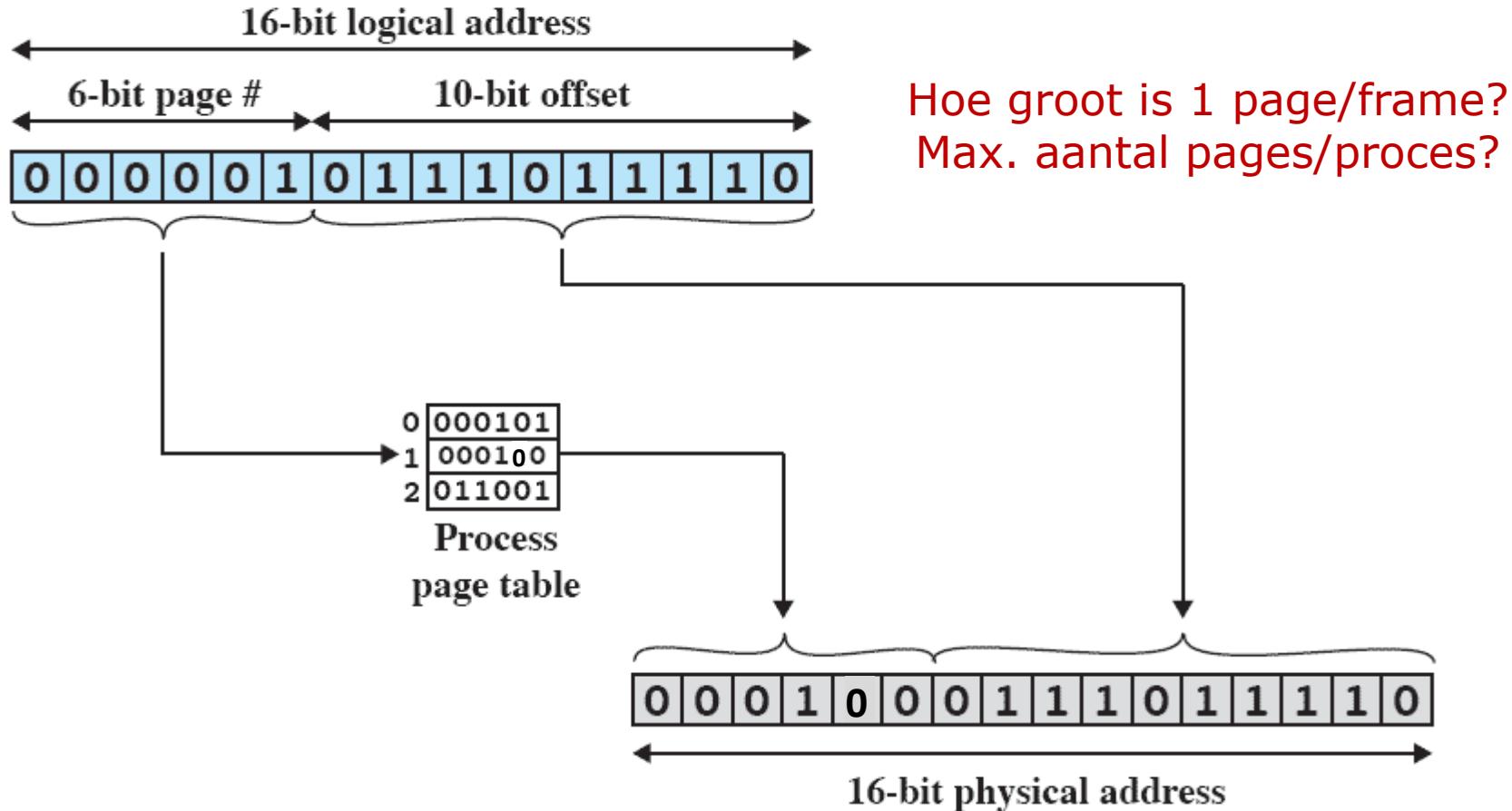


Paging

- de pages hoeven niet opeenvolgend te zijn!
- vraagt wel speciale hardware
 - logisch adres bestaat uit *page* en *offset*
 - hardware zoekt frame (startadres voor page) in geheugen (**page-table**)
 - hardware voegt startadres aan offset toe = fysiek adres
 - hardware zendt fysiek adres naar geheugen
- er zijn dus weer 2 geheugentoegangen nodig
 - soms opgelost via caching of page-table in processor



Paging



Vertaling is eenvoudiger dan bij Segmentation. Er moet geen optelling gebeuren (paginagrootte macht van 2!)

Paging

- Oefening: 8 bit pages
 - logisch adres: 1101 1100 1101 1000
 - page-table:

page	frame
...	...
1101 1011	1110 1010
1101 1100	1110 1101
1101 1101	1110 1110
...	...

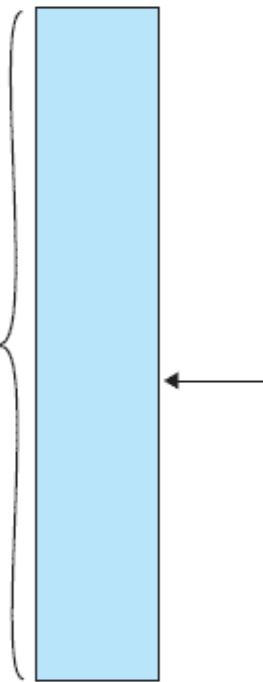
- Fysisch adres?

Logische en fysieke adressen

Relative address = 1502

0000010111011110

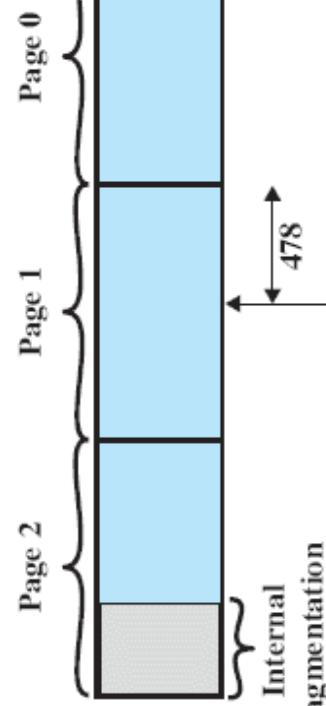
User process
(2700 bytes)



(a) Partitioning

Logical address =
Page# = 1, Offset = 478

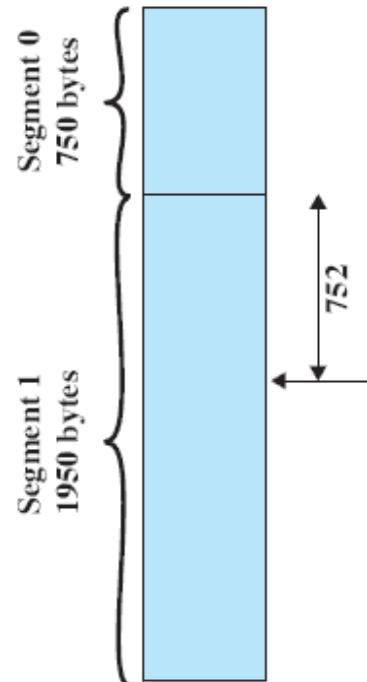
0000010111011110



(b) Paging
(page size = 1K)

Logical address =
Segment# = 1, Offset = 752

0001001011110000



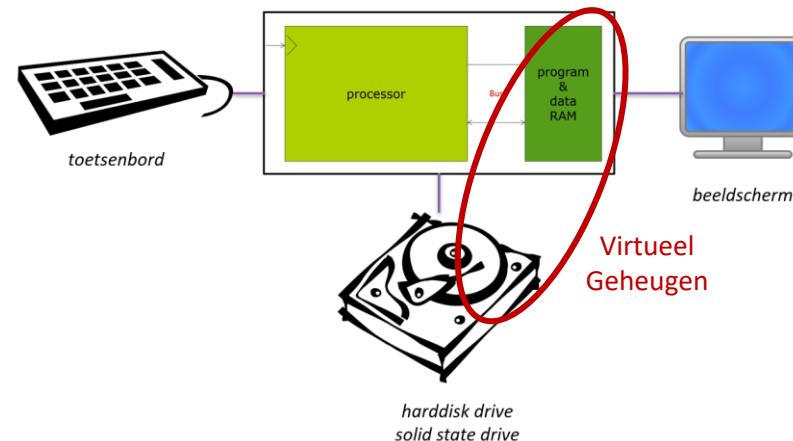
(c) Segmentation

Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Virtueel geheugen

- niet gebruikte pages kunnen naar **swap space** op de HDD geschreven worden om plaats te maken voor een nieuw proces (uitswappen)
- wanneer het proces deze page terug nodig heeft moet deze terug ingeswapt worden
- swap space kan zich in een file of in een aparte partitie van de HDD bevinden
- laat toe om meer programma's te laden dan het RAM geheugen groot is



RAM
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

page table A
1
2
on disk
on disk
on disk
8
7
on disk

page table B
on disk
on disk
on disk
on disk
14
15
3
on disk
on disk
on disk
on disk
5
10
on disk
on disk
on disk
on disk
4
on disk

Virtueel geheugen

- processor gebruikt logisch adres
- hardware zet adres om in fysiek adres
 - als pagina niet in het geheugen is: "**page fault**"
 - is een interrupt
 - swapping:
 - schrijf eventueel frame naar disk om plaats te maken
 - laad page in het vrijgemaakte frame
 - resume proces

Virtueel geheugen

- Design issues
 - swapping vraagt veel tijd: moet geminimaliseerd worden
 - OS meer tijd nodig om te swappen dan processen uit te voeren = thrashing
 - grootte van de pages bepaalt performantie en geheugengebruik
 - sommige frames moeten misschien gelockt worden (frames die het OS bevatten)
 - welke pagina's moeten worden bewaard om plaats te maken?
 - minst gebruikte?
 - langst niet gebruikt?
 - welke pagina's moeten worden geladen?
 - enkel de pagina die nodig is?
 - naburige pagina's ook?

Ubuntu pagesize

1. Wat is de grootte van een page in jou Linux systeem?

Inhoud

- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Herhalingsvragen

- wat is het verschil tussen een monolytisch, modulair en micro-kernel?
- wat is de taak van de linker? Bespreek het verschil tussen static en dynamic linken. Wat zijn de voordelen van dynamic linken?
- wat is partitionering met vaste partities? welke 2 vormen bestaan er?
- wat is interne fragmentatie bij vaste partitionering van het geheugen? Geef een voorbeeld
- wat is dynamische partitionering?
- wat is externe fragmentatie bij dynamische partitionering van het geheugen? Geef een voorbeeld. Wat is de oplossing voor dit probleem?
- stel dat je volgende dynamische partitionering hebt. Er wordt een aanvraag gedaan voor ... MB. Waar zal deze terecht komen volgens first-fit, best-fit en next-fit?
- leg de werking van segmenting uit.

Herhalingsvragen

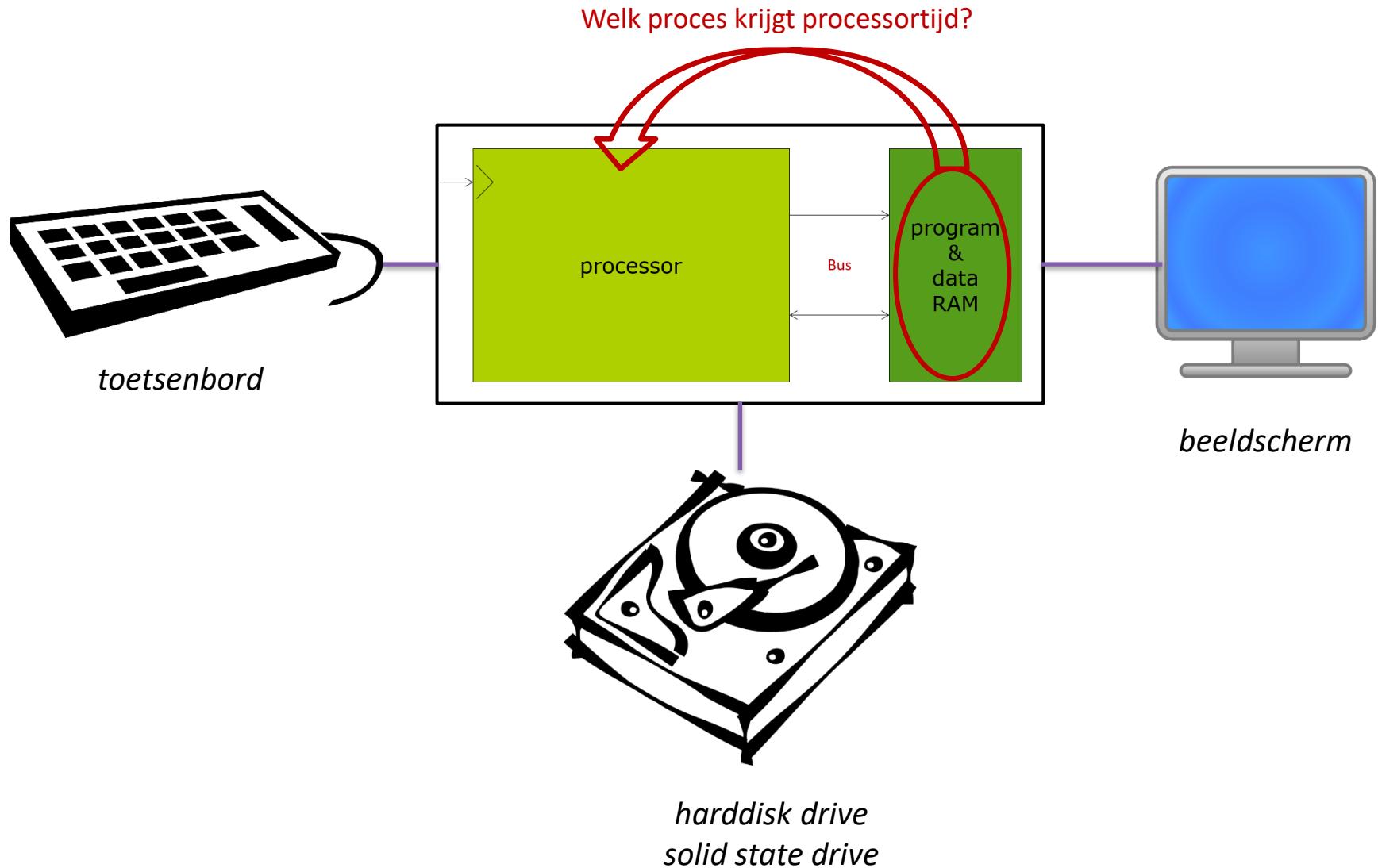
- gegeven volgende segment-table. Waar zullen de volgende logische adressen op gemapt worden?
- gegeven volgende page-table. Waar zullen de volgende logische adressen op gemapt worden?
- gegeven volgende segment- en page-table. Waar zullen de volgende logische adressen op gemapt worden?
- wat is een segmentation fault?
- wat is virtueel geheugen? Hoe werkt virtueel geheugen?
- wat is thrashing?
- wat is een page fault? hoe reageert het OS hierop?

Computersystems 2

Theorie

6. Procesbeheer

Procesbeheer

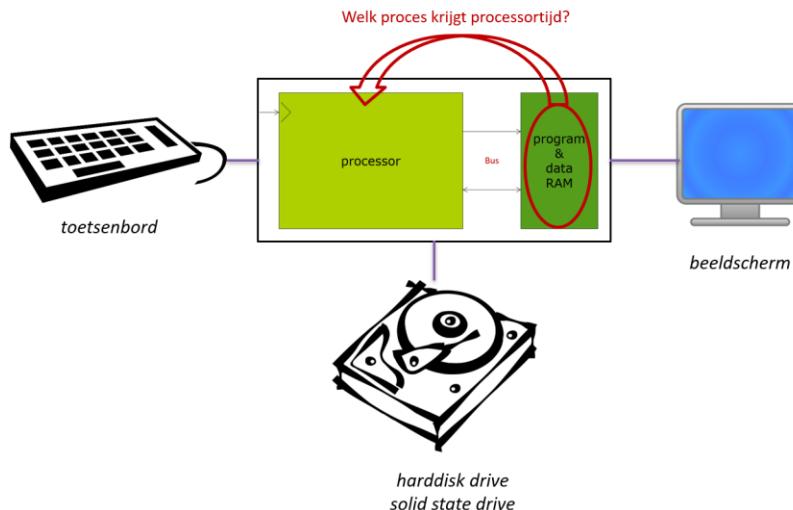


Inhoud

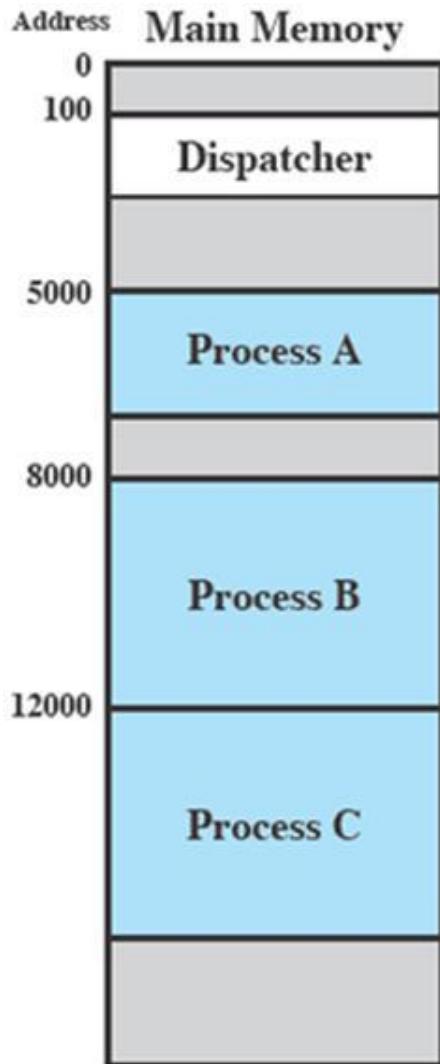
- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Procesbeheer

- **Multiprocessing**
 - Computer runt verschillende processen “tergelijkertijd”
- **Multi-user**
 - Verschillende gebruikers werken “tergelijkertijd” op de computer



Sporen (traces)



- 3 processen in het geheugen
- **dispatcher** is deel van het OS
- dispatcher heet soms '**scheduler**'

Traces gezien vanuit de processen

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of Process A

(b) Trace of Process B

(c) Trace of Process C

5000 = Starting address of program of Process A

8000 = Starting address of program of Process B

12000 = Starting address of program of Process C

Traces gezien vanuit de processor

1	5000	27	12004
2	5001	28	12005
3	5002	----- Timeout	
4	5003	29	100
5	5004	30	101
6	5005	31	102
----- Timeout		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----- Timeout	
16	8003	41	100
----- I/O Request		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
----- Timeout			

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Scheduling in de praktijk

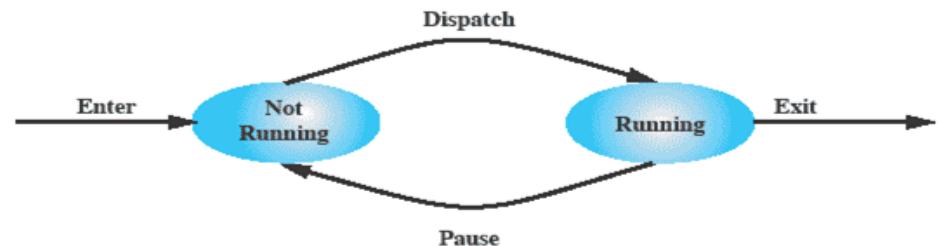
- in Linux en Mac OS-X
 - xload
 - top
 - ps
 - pstree
- in Windows
 - Taakbeheer
 - PowerShell: Get-Process

Inhoud

- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Procestoestanden

- Een proces bestaat uit:
 - code
 - data, stack
 - toestand (context)
- Een proces moet minstens 2 toestanden hebben:
 - running
 - not-running
- OS moet de toestand van een proces (context) bewaren en omwisselen voor volgend proces (=context switch)

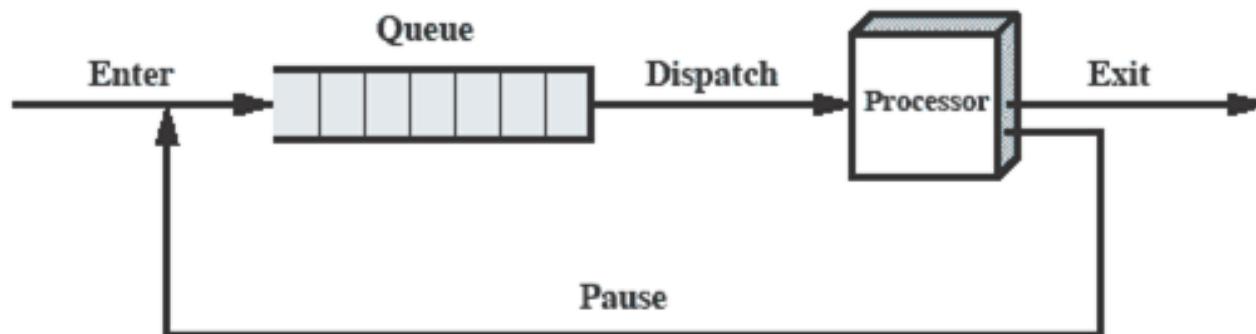


Processtoestanden

- context wordt opgeslagen in **PCB**
- **Process Control Block** = datastructuur
 - welk was de laatst uitgevoerde instructie?
 - wat waren de waarden van de registers?
 - hoe was het geheugen geconfigureerd (paging, segmenting, virtual memory)
 - wat is de id van dit proces?
 - welke resources gebruikt dit proces (open files, netwerkverbindingen, ...)?
 - heeft dit process kind-processen?
 - starttijd, processortijd, ...
 - ...

Procestoestanden

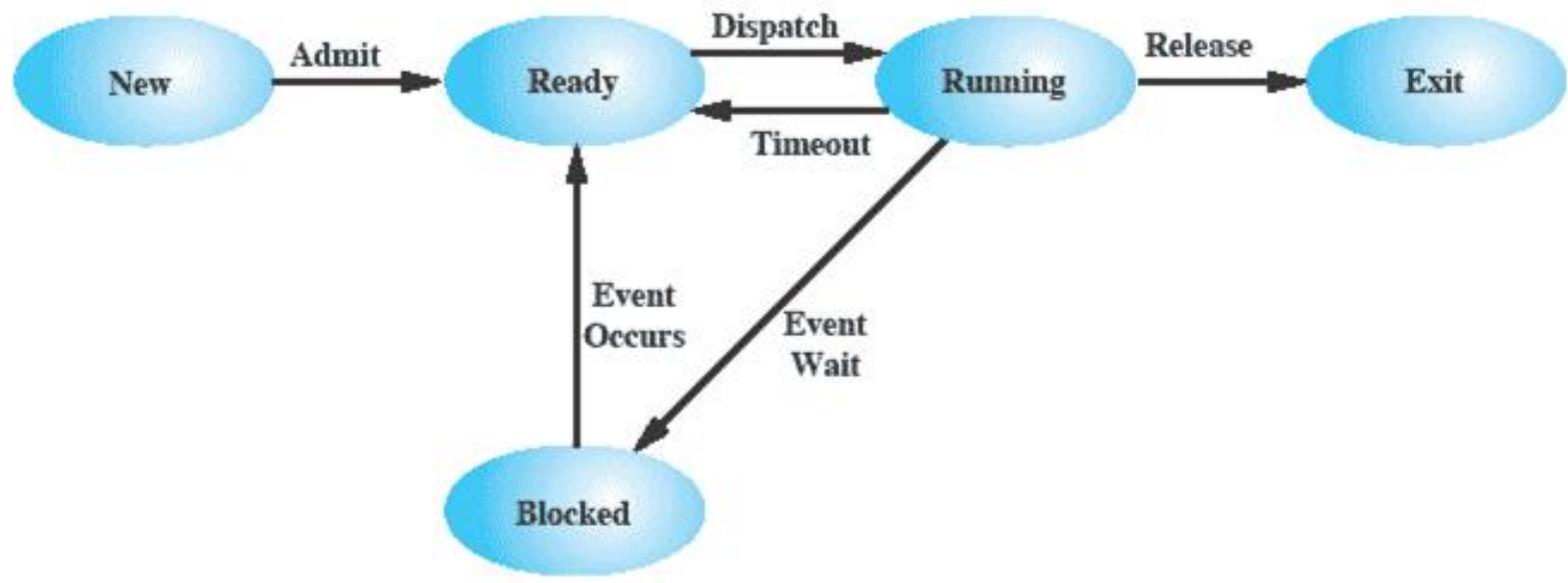
- OS heeft een queue van PCB's



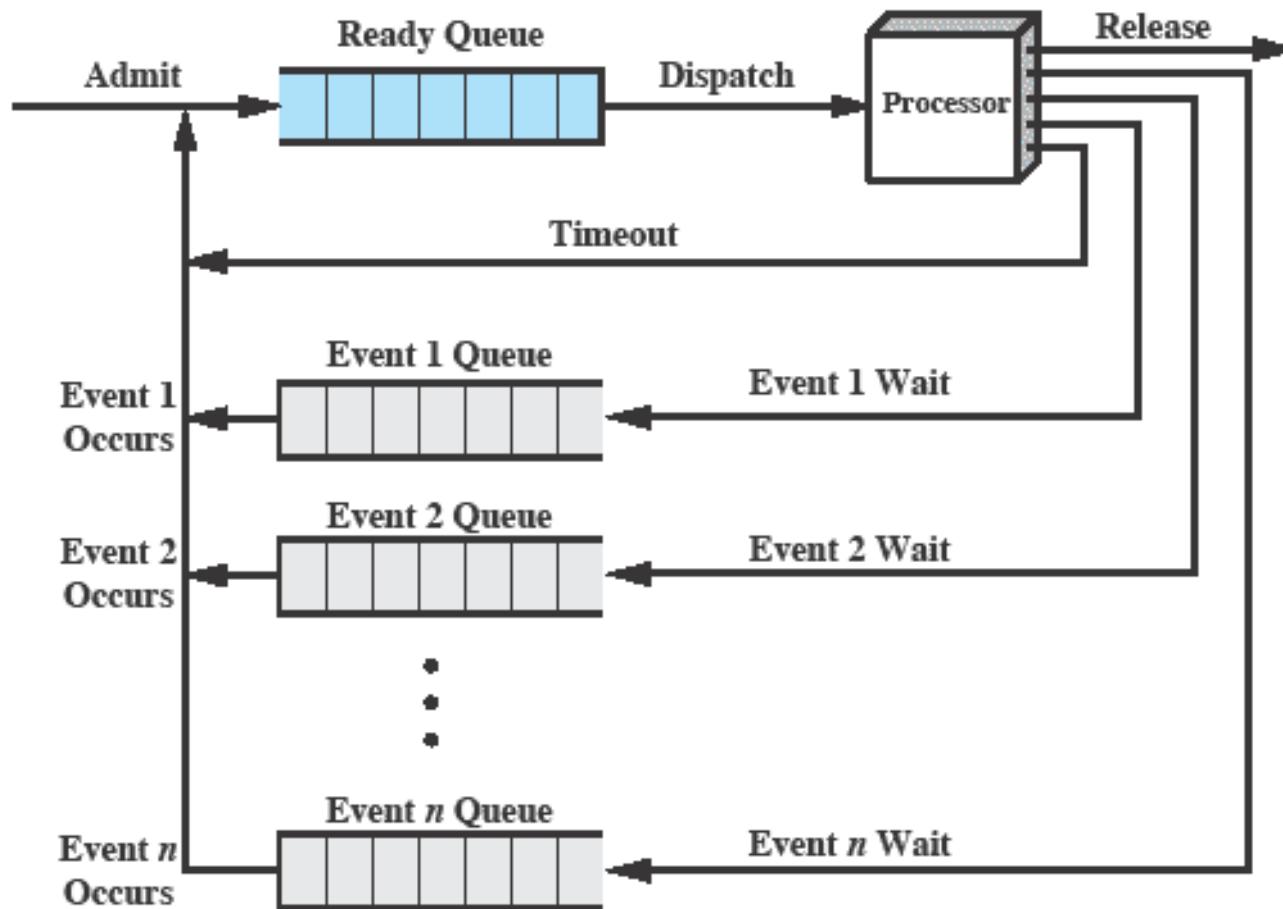
Processtoestanden

- 2 toestanden is niet genoeg: geen verschil tussen wachten op i/o of gepauzeerd
- dus: 3 toestanden
 - ready
 - running
 - blocked
- veel OS-en voegen er nog 2 toestanden bij:
 - new
 - exit

Processtoestanden



Queues in het OS



Linux procestoestanden

1. Met welk commando kan je de processen en hun procestoestand zien?
2. Welke toestanden bestaan?
3. Wat is idle time? Met welk commando kan je deze zien?

Inhoud

- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Nieuw proces starten in Linux

Opstarten nieuw programma: 2 system calls:

1. int fork()

- huidige proces wordt gekopieerd
 - code segment
 - data en stack segment
 - PCB
- huidige proces kan verder lopen
- nieuw 'kind' proces wordt opgestart in gekopieerde versie
- kindproces krijgt 0 als return-value
- ouderproces krijgt process-id als return value

2. exec("executable")

- vervangt code segment met nieuwe code nieuw programma
- reset stack en data segment

Nieuw proces in bash

bash_ \$ xload

1. bash doet *fork* om child proces aan te maken
2. daarna *exec* om in het child proces de xload code te laden

Nieuw proces starten in Linux

```
int main() {  
    int i;  
  
    for(i=0; i<10; i++) {  
        int f = fork();  
        if (f==0) {  
            doe_child(i);  
            return 0;  
        }  
    }  
    sleep(3);  
    execl("/bin/ps", "ps", "-f",  
          NULL);  
    printf("Niet uitgeprint!");  
}
```



Inhoud

- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Context switch

- **non pre-emptive**
 - OS onderbreekt de applicatie niet
 - Applicatie runt tot return of tot deze moet wachten (bv. op I/O)
 - Voordeel: heel eenvoudig
 - Nadeel: ?
 - Voorbeelden: Windows 3.11, Mac OS9, Oberon, ...
- **pre-emptive**
 - OS onderbreekt proces om een ander proces Running te maken, bv. na time slice
 - Voorbeelden: huidige Windows en Linux

Pre-emptive context switch

- veroorzaakt door interrupt
- ga naar 'kernel mode'
- save registers en config (seg/page table) in PCB
- zet PCB in juiste queue
- zoek nieuw proces om te starten
- bijwerken PCB nieuw proces (toestand Running)
- laad registers uit nieuwe PCB
- zet timer voor volgende context switch
- ga naar 'user mode'
- jmp juiste adres

Online and batch job scheduling

- **Online scheduling**
 - Jobs runnen in foreground
 - Interactie met user
- **Batch scheduling**
 - Jobs runnen in background
 - Geen interactie met user
 - Gebruikt voor jobs die lange run time hebben
 - Material Requirement Planning voor een fabriek
 - Trainen van Deep Learning modellen

Inhoud

- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Scheduling strategieën

- bij context-switch: "kies nieuw proces"...
- verschillende strategieën mogelijk
- doel:
 - eerlijk verdelen van processortijd over processen
 - "uthongering" beletten
 - weinig overhead veroorzaken
 - prioriteiten van processen respecteren

Scheduling strategieën

- ieder proces heeft volgende waarden (in PCB):
 - **w** = tijd doorgebracht in het systeem voor wachten (tot nu toe)
 - **e** = tijd besteed aan uitvoering (tot nu toe)
 - **s** = totale uitvoeringstijd die nodig is om het proces af te ronden (inclusief e) (=schatting!)

First come first served (FCFS)

- kies proces met **maximale w**
- dit is het 'oudste proces'
- **non pre-emptive**: wacht gewoon tot i/o request of einde
- voordeel: lange processen worden sneller doorlopen (in vgl. met 2 volgende algoritmen)
- nadeel: korte processen moeten soms lang wachten
- niet voor online scheduling

Shortest Process Next (SPN)

- kies het proces met de kleinste s
- non pre-emptive
- voordeel: schnellere response-tijd
- nadelen:
 - je moet weten hoe lang een proces nodig heeft
 - mogelijkheid tot starvation (uithongering)
- niet voor online scheduling

Shortest remaining Time (SRT)

- kies proces met kleinste (s-e)
- pre-emptive: als er een nieuw proces bij komt met kleinere (s-e), dan wordt huidige proces onderbroken
- nadeel: zoals vorige
- niet voor online scheduling

Highest Response Ratio Next (HRRN)

- zoek proces met **hoogste $(w+s)/s$**
- **non pre-emptive**
- Response ratio = $(w+s)/s$
 - initieel gelijk aan 1
- voordeel: korte processen hebben grote RR en worden snel afgewerkt maar lange processen die al lang wachten krijgen steeds hogere RR
- nadeel: s moet nog steeds geschat worden
- niet voor online scheduling

Round-Robin

- meest voorkomende scheduling in moderne OS'en
- queue van processen: neem steeds de volgende
- **pre-emptive**
 - ieder proces krijgt evenveel tijd (**time-slice**)
 - proces wordt onderbroken door interrupt
- voordeel: goede response-tijd, rechtvaardige behandeling
- nadeel: i/o gebonden processen krijgen minder tijd

Ubuntu timeslice

1. Wat is de grootteorde van de timeslice bij Linux?

Inhoud

- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Scheduling in Linux

- pre-emptive, round-robin
- verschillende prioriteiten
 - ieder proces prioriteit/nice-value
 - nieuw proces start default met PR 20 (NI 0)
 - PR = NI + 20
 - kan aangepast worden met nice (of renice)
 - NI: -20 tot +19
 - PR: 0 (hoog) tot 39 (laag)
 - Hoe lager PR of NI, hoe hoger de prioriteit

Slurm (optional, niet te kennen)

- Simple Linux Utility for Resource Management
- Open source batch job scheduler
- Used by many supercomputers
- Allocating exclusive/non-exclusive access to computer nodes

- cat submit.sh

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=test.out
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100
srun testprogram
```

- sbatch submit.sh
 - squeue

- <https://drtailor.medium.com/how-to-setup-slurm-on-ubuntu-20-04-for-single-node-work-scheduling-6cc909574365>
- https://support.cecil-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html



Inhoud

- Dispatcher/scheduler
- Toestanden van een proces
- Opstarten van processen in Linux
- (Non) pre-emptive context switch
- Scheduling strategieën
- Linux scheduling
- Herhalingsvragen

Voorbeelden examenvragen

- Bespreek de delen van een proces: stack, data, code, PCB
- Leg de 5 toestanden van een proces uit en geef uitleg bij de overgangen.
- Wat is een PCB?
- Wat is idle time?
- Wat gebeurt er tijdens een pre-emptive context switch?
- Wat is het verschil tussen pre-emptive en non pre-emptive scheduling?
- Wat is een context-switch?
- Wat is scheduling?
- Hoe start een proces in Linux? Verklaar de fork en exec system calls.
- Wat doet de volgende C code?
- Bespreek de scheduling strategie X (wat/hoe, voordeel, nadeel)
- Is scheduling strategie X pre-emptive of niet, waarom?
- Wat is uithongering (starvation)?

Computersystems 2

Theorie

7. IPC & threads

Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Interprocess communication (IPC)

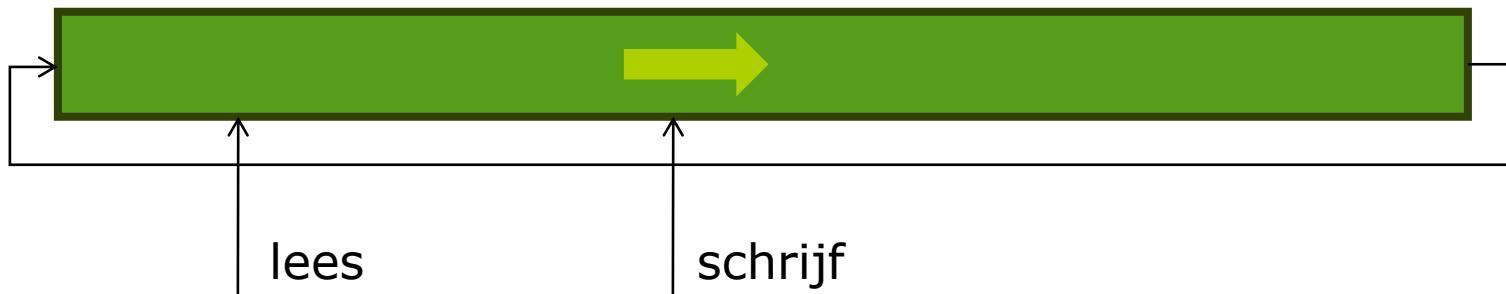
- processen hebben eigen stuk afgeschermd geheugen
- processen moeten kunnen communiceren
 - copy/paste
 - parallel processing
 - bij μ -kernel

Interprocess communication (IPC)

- Unix
 - pipes
 - Berichten/message queue
 - shared memory
- Windows
 - pipes, clipboard, files

Unix en Windows pipes

- ieder proces heeft stdin, stdout en stderr
- koppel stdout van 1 proces aan stdin van ander proces
- cat <filename> | grep 'class' | sort
- **FIFO**
- OS heeft per pipe een circulaire buffer waarin geschreven en gelezen kan worden:
- C: via pipe() system call



Unix berichten

- Unix voorziet message queues, toegankelijk via een id
- Unix system calls
 - int msgget(key, flags)
 - creëert een message queue
 - geeft id terug
 - void msgsnd(qid, message, size, flags)
 - zendt een bericht naar een gegeven queue
 - void msgrcv(qid, message, maxSize, type, flags)
 - haalt een bericht van een queue
 - type=0: FIFO
 - type>0: leest bericht van dit type
 - type<0: leest bericht met laagste type (=prioriteit)
 - void msgctl(qid, cmd, data)
 - o.a. gebruikt om queue te verwijderen

Unix berichten

```
#define MSGKEY 0x7B  
  
struct msgform {  
    long type;  
    char message[255];  
};
```

client

```
int main() {  
  
    int msgid=msgget(MSGKEY,0777);  
    printf("Ga bericht sturen op de queue\n");  
    struct msgform message;  
    message.type = 1;  
    strcpy(message.message, "Hello, world!");  
    msgsnd(msgid,&message,255,0);  
  
}
```

server

```
int main() {  
  
    int msgid=msgget(MSGKEY,0777|IPC_CREAT);  
    printf("message queue %d created.", msgid);  
    printf("Nu wachten...\n");  
    struct msgform message;  
    msgrcv(msgid, &message, 255, 1, 0);  
    printf("bericht ontvangen: ");  
    printf("%s\n", message.message);  
    msgctl(msgid, IPC_RMID, 0);  
}
```

Compileer server.c en client .c en start de server in een terminalvenster.

Met welk commando kan je in Linux de message queues zien?

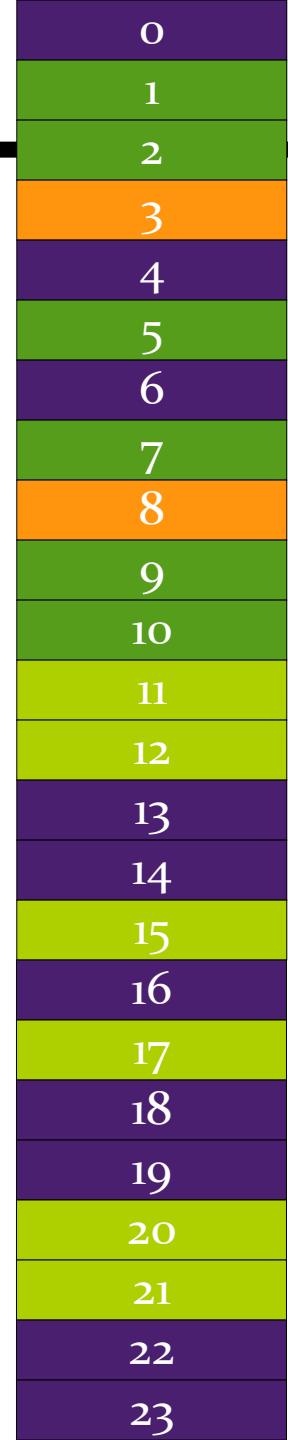
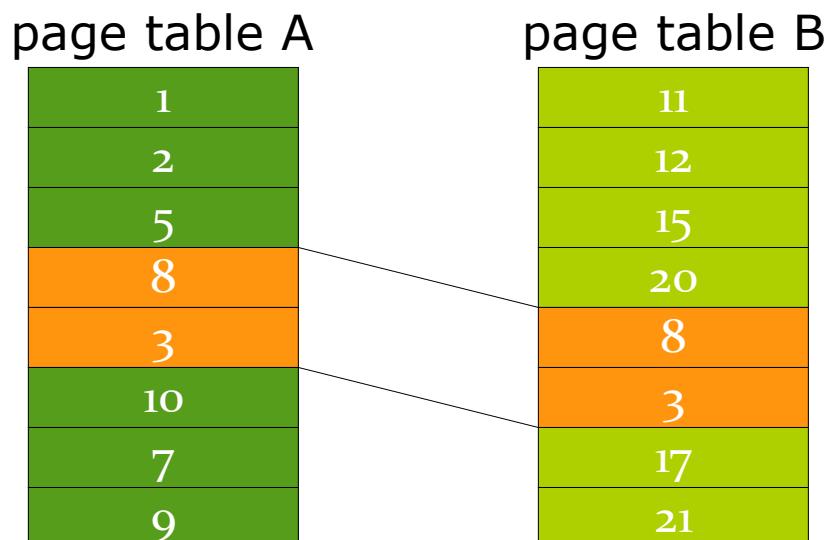
Start daarna de client in een ander terminal venster.

Unix shared memory

- Unix kan een stuk geheugen aan verschillende processen toewijzen
- dit geheugen wordt niet vrijgegeven als een proces stopt
- system calls
 - int shmget(key, size, flags)
 - void *shmat(mid, address, flags)
 - mapt shared memory op een adres
 - void shmdt(address)
 - unmapt shared memory
 - void shmctl(mid, cmd, data)

Unix shared memory

- shared memory wordt geïmplementeerd adhv paging



Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

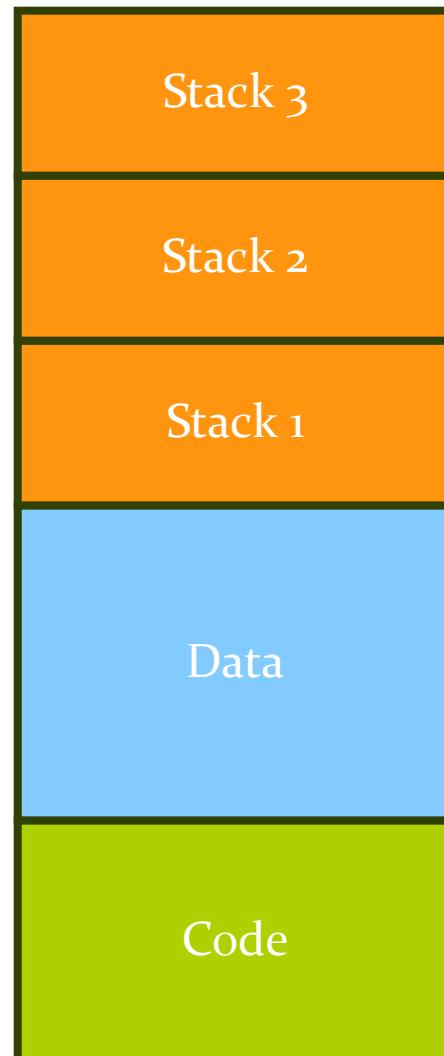
Threads

- problemen met processen
 - processen opstarten duurt "lang"
 - context-switch duurt "lang"
 - sommige applicaties hebben nood aan verschillende processen die met dezelfde data (variabelen) werken
 - IPC mechanismen zijn dan te complex

Threads

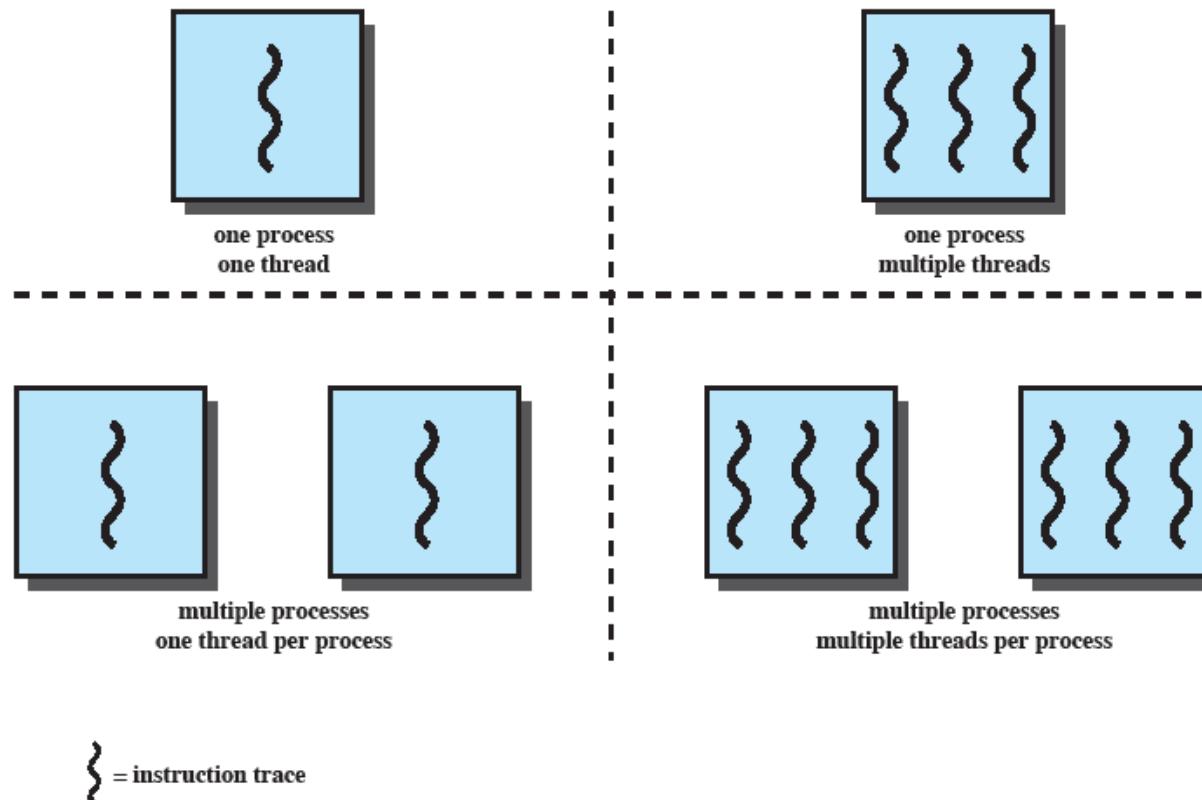
- light-weight process = thread
- 1 proces kan verschillende threads hebben
- threads delen
 - code-segment
 - data-segment
- threads hebben wel **eigen stack**
- context-switch is eenvoudiger

Threads



1 proces
3 threads

Threads



Threads: implementatie

- user-level threading
 - N:1
 - N threads : 1 OS scheduling entiteit
 - ieder proces heeft zelf een scheduler voor threads
 - kernel weet zelfs niet dat er threads zijn
 - voordeel: geen switch naar kernel-mode (snel)
 - nadeel:
 - meer logica in proces
 - als proces in 'ready' staat, dan staan alle threads ook in wacht
 - threads van 1 proces draaien op 1 processor/core (zelf bij multi-core / multi-processor)

Threads: implementatie

- kernel-level threading
 - 1:1
 - 1 thread : 1 OS scheduling entiteit
 - OS regelt scheduling van threads
 - voordeel:
 - transparant voor de processen
 - OS kan threads schedulen, onafhankelijk van processen
 - threads van 1 proces kunnen op verschillende processoren / cores draaien
 - nadeel: switch naar kernel-mode nodig (trager)

Threads: implementatie

- hybrid threading
 - M:N
 - M thread : N OS scheduling entiteiten
 - Combinatie van 1:1 en M:1
 - Combineert voordelen, meer complexe implementatie

Threads

- communicatie tussen threads is veel eenvoudiger
 - data- en code-memory is altijd shared
 - opgelet: lokale variabelen en parameters staan op de stack en zijn dus niet shared!

Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Threads in Linux

```
#include<pthread.h>
pthread_t tid1, tid2;
char bericht[30];

void* doThread1(void *arg)
{
    printf("1 stuurt naar 2\n");
    strcpy(bericht,"Bericht van 1");
    sleep(10);
}

void* doThread2(void *arg)
{
    sleep(10);
    printf("2 ontvangt : %s\n", bericht);    }
int main(void)
{
    pthread_create(&tid1, NULL, &doThread1, NULL);
    printf("\n Thread 1 created\n");
    pthread_create(&tid2, NULL, &doThread2, NULL);
    printf("\n Thread 2 created\n");
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf("\n Threads done!\n");
    return 0;
}
```

gcc threadvb.c -o threadvb **-pthread**

Compileer vorig vb. en test dit uit.

Met welk commando kan je in Linux de threads zien?

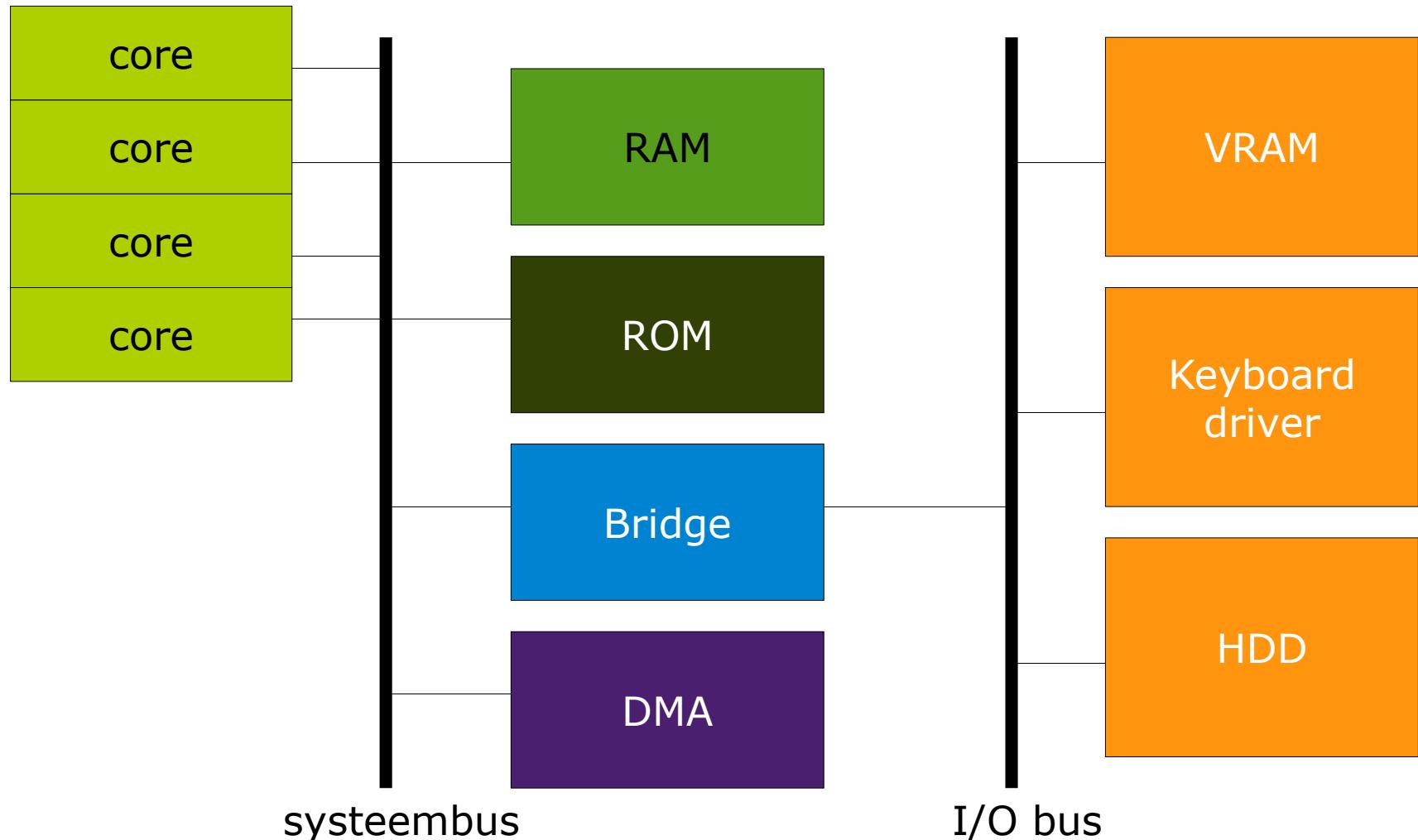
Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Multi-processoren

- multi-processor/multi-core
 - meerdere 'cores' of meerdere processoren
 - meestal shared memory
- gevolgen voor OS
 - 1 ready-queue, meerdere processoren
 - load-balancing
 - conflicten verwerken als processoren dezelfde resource willen gebruiken

Architectuur met multi-processor/core

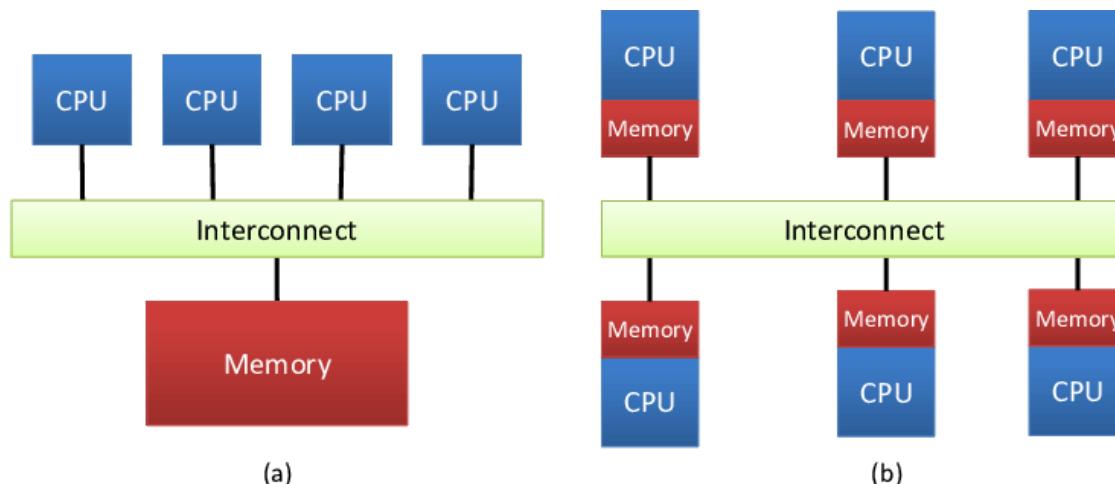


Shared Memory Multi-processor: strategieën

- **master-slave**
 - 1 processor is de master, de anderen zijn slave
 - de master draait de kernel en doet scheduling
 - redelijk eenvoudig te implementeren (1 processor master over geheugen en I/O)
 - master kan wel bottleneck worden
- **symmetric (SMP)**
 - kernel kan worden uitgevoerd op elke processor
 - iedere processor doet eigen scheduling
 - ingrijpende wijzigingen in de kernel-code voor synchronisatie
 - verschillende processoren kunnen dezelfde code willen uitvoeren
 - verschillende processoren kunnen in hetzelfde deel van het geheugen lezen en schrijven
 - kernel gebouwd met meerdere processen/threads

Shared memory / Distributed memory

- Supercomputers met 1000's cpu's/cores (bv. Cray van ECMWF 260.000 cores!): geen shared memory architectuur, maar distributed memory architectuur (want connectie naar shared memory is bottleneck).



Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Gelyktijdigheid vb.

```
#include<pthread.h>
pthread_t tid1, tid2;
char naam[10];

void* doThread1(void *arg)
{
    scanf("%s",naam);
    sleep(3);
    printf("%s\n",naam);
}

void* doThread2(void *arg)
{
    sleep(2);
    scanf("%s",naam);
    printf("%s\n",naam);
}

int main(void)
{
    pthread_create(&tid1, NULL, &doThread1, NULL);
    pthread_create(&tid2, NULL, &doThread2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    return 0;
}
```

Gelyktijdigheid

- meerdere processoren/threads = probleem met gelyktijdig aanspreken van bepaalde resources (bv. geheugen)
- sommige stukken code mogen maar door 1 proces/thread tegelijk worden uitgevoerd
 - = "**critical section**"
 - er moet een soort locking mogelijk zijn
 - deze locking moet light-weight zijn
- deze problematiek bestaat ook met 1 processor!!!

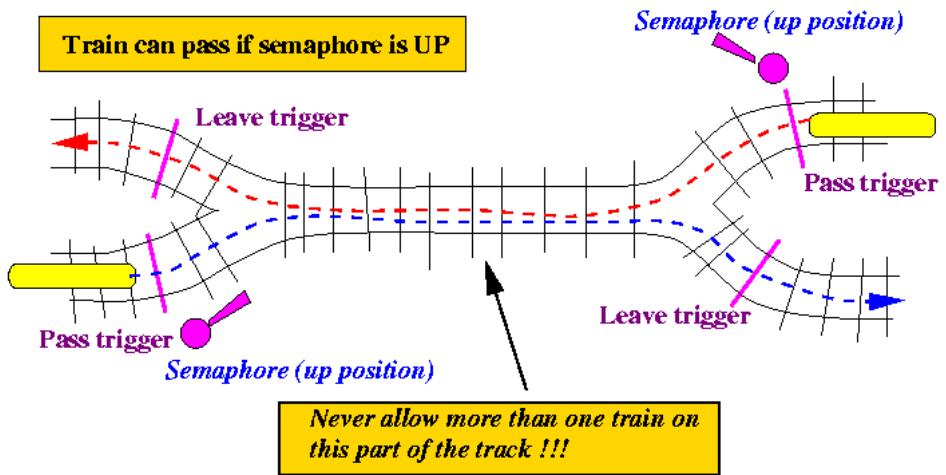
Gelyktijdigheid

- `wait()` en `signal()` zijn 'atomic operations'
 - geïmplementeerd en gegarandeerd door OS
 - “**Semaforen**”



- 0: sein neer (stop)
- 1: sein op (mag doorrijden)

Semaforen in Linux



- `sem_wait()`
 - Als 0: wacht
 - Als > 0 (sein op): doe semafoor -1 (sein neer) en ga door
- `sem_post()`
 - Doe semafoor +1 (sein op)
- `sem_init()`
 - Initialiseer semafoor
- `sem_wait()` en `sem_post()` zijn '**atomic operations**'

Semaforen

```
#include<pthread.h>
#include <semaphore.h>
pthread_t tid1, tid2;
char naam[10];
sem_t sema;

void* doThread1(void *arg)
{
    sem_wait(&sema); /*semafoor NEER*/
    scanf("%s",naam);
    printf("%s\n",naam);
    sem_post(&sema); /*semafoor OP*/
}

void* doThread2(void *arg)
{
    sem_wait(&sema); /* semafoor NEER */
    scanf("%s",naam);
    printf("%s\n",naam);
    sem_post(&sema); /* semafoor OP */
}

int main(void)
{
    sem_init(&sema, 0, 1); /* semafoor OP */
    pthread_create(&tid1, NULL, &doThread1, NULL);
    pthread_create(&tid2, NULL, &doThread2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    sem_destroy(&sema);
}
```

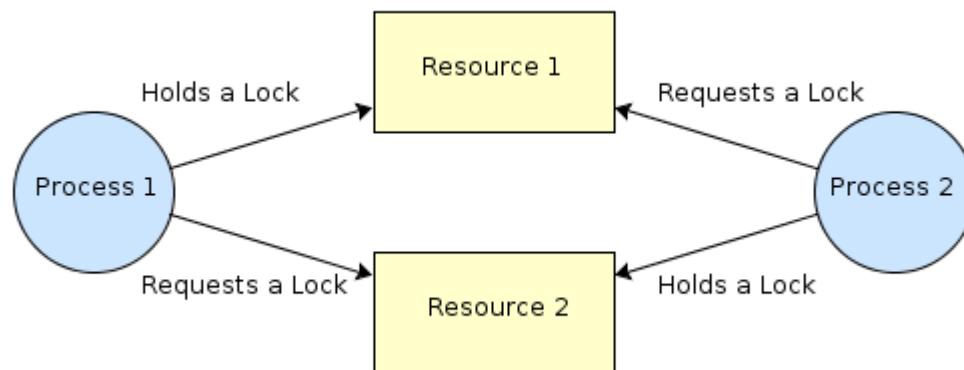
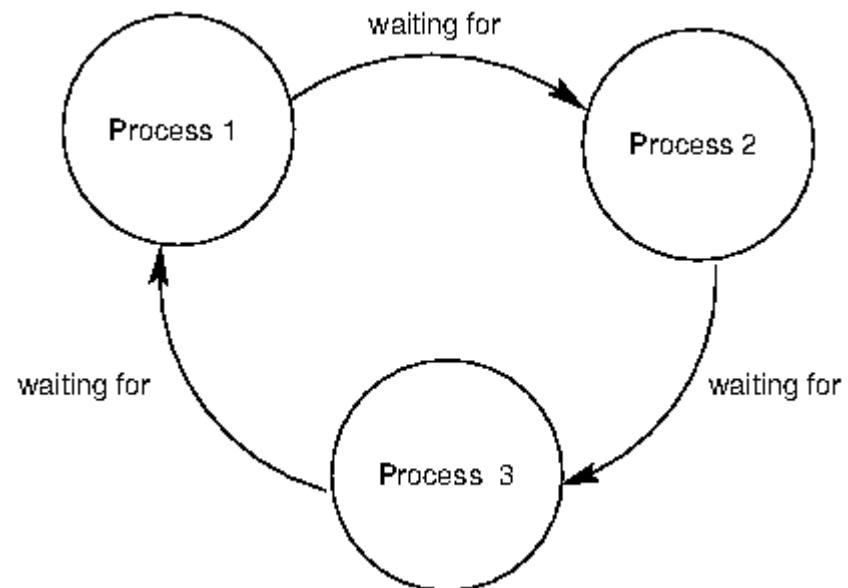
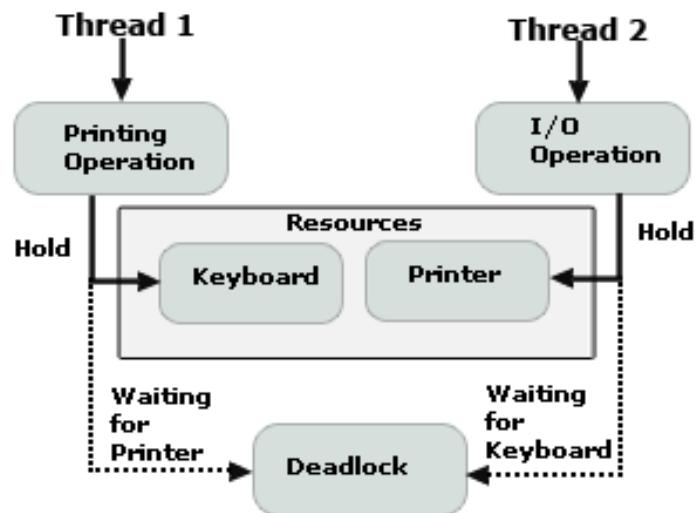
Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

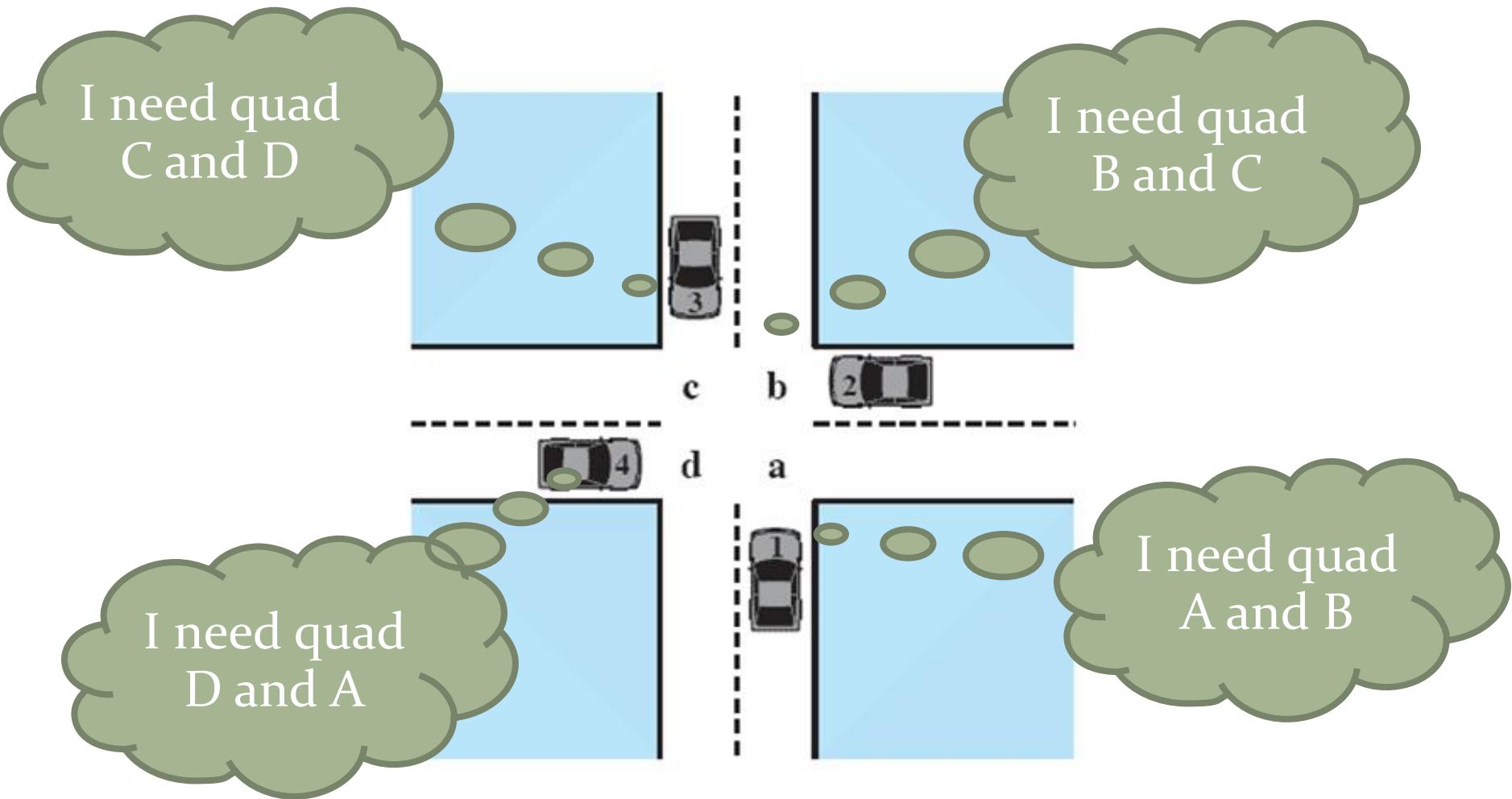
Deadlock

Wat is een deadlock?

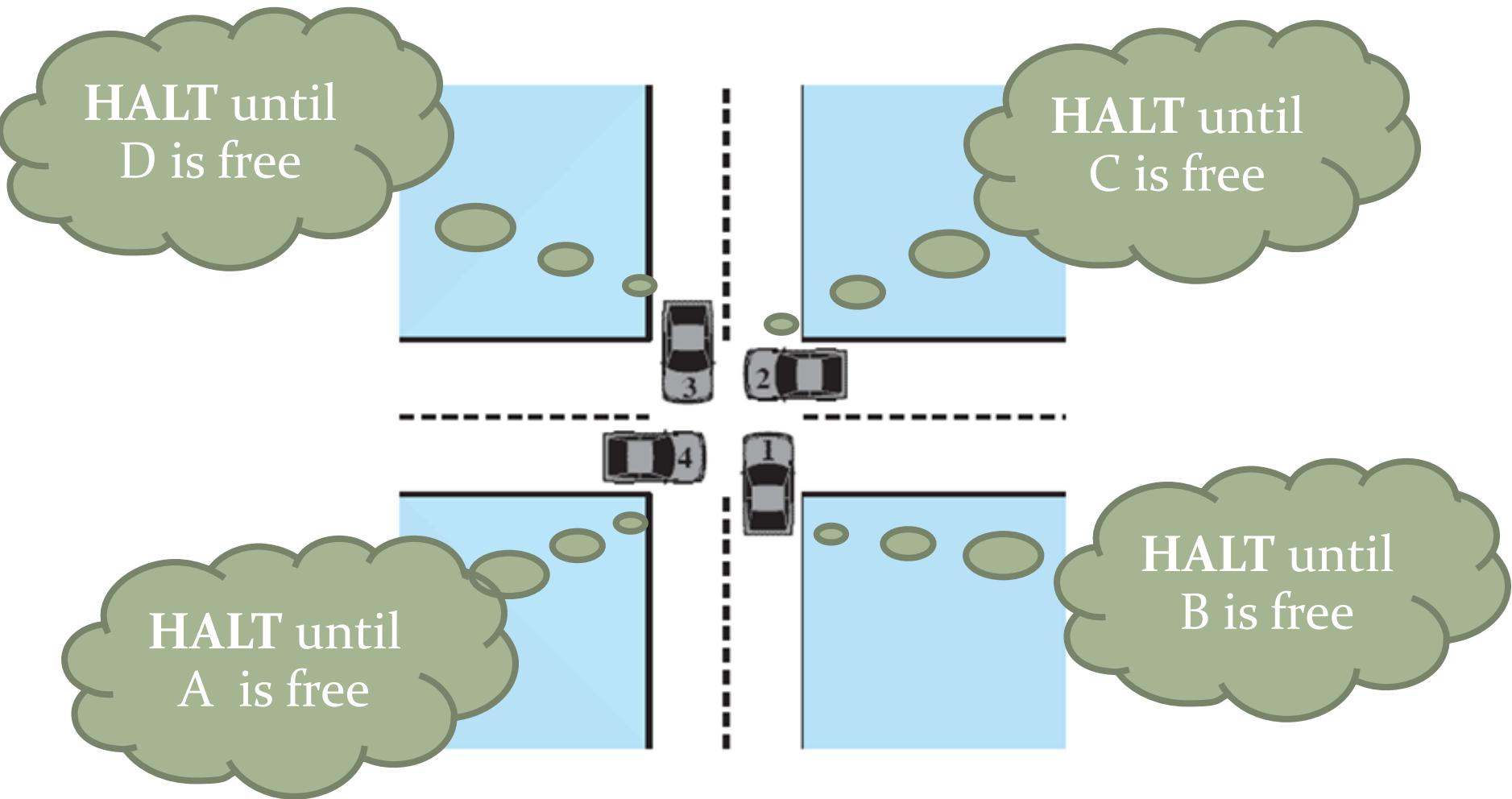
Deadlock



Deadlock



Deadlock



Deadlock

```
void* doThread1(void *arg)
{
    sem_wait(&sema1);
    printf("1 gelocked, nu nog 2...\n");
    sleep(1);
    sem_wait(&sema2);
    printf("1 en 2 gelocked!\n");
    sem_post(&sema2);
    sem_post(&sema1);
}
```

```
void* doThread2(void *arg)
{
    sem_wait(&sema2);
    printf("2 gelocked, nu nog 1...\n");
    sleep(1);
    sem_wait(&sema1);
    printf("2 en 1 gelocked!\n");
    sem_post(&sema1);
    sem_post(&sema2);
}
```

Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Herhalingsvragen

- Wat zijn pipes?
- Welke stappen moet je doorlopen om een bericht van een proces naar een ander te sturen in Unix?
- Welke stappen moet je doorlopen om een shared memory segment op te zetten in Unix?
- Hoe kan het OS een deel geheugen delen tussen verschillende processen?
- Welke geheugensegmenten worden gedeeld door threads?
- Wat is het verschil tussen een proces en een thread?
- Wat zijn de voor- en nadelen van threads?
- Wat is het verschil tussen een user-level thread en een kernel-level thread? Wat zijn de voor- en nadelen?
- Waarom worden lokale variabelen niet gedeeld tussen threads en globale wel?
- Wat is load-balancing in een multiprocessor?
- Teken de architectuur van een multiprocessor machine
- Wat is het verschil tussen een master-slave en een SMP multiprocessor OS?
- Wat is een 'critical section'?
- Leg uit wat een semafoor is. Welke operaties kan je erop uitvoeren en wat doen deze?
- Leg uit wat deadlocks zijn. Wanneer treden ze op?

Computersystems 2

Theorie

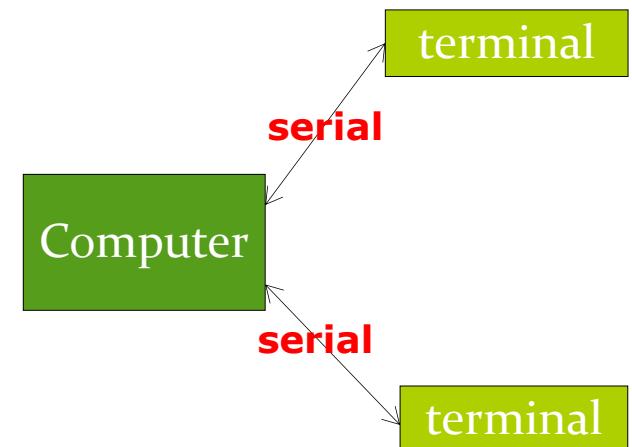
8. UI & virtualisatie

Inhoud

- User interfaces
- X Windows
- Virtualisatie
- Cloud computing
- Blade servers
- Herhalingsvragen

Text terminals

- text terminal = scherm + toetsenbord
- enkel karakter-output (bv 80x25)
- verbonden via **serial** connection (of via modem) (telnet)
- veel gebruikt voor communicatie met embedded systemen
 - router
 - auto-elektronica
 - ...
- in Unix heeft iedere gebruiker een "tty"
- ieder proces heeft stdin, stdout, stderr
 - stdin is input-stream vanuit terminal
 - stdout is output-stream naar terminal
 - stderr is standaard zoals stdout



Windowing system

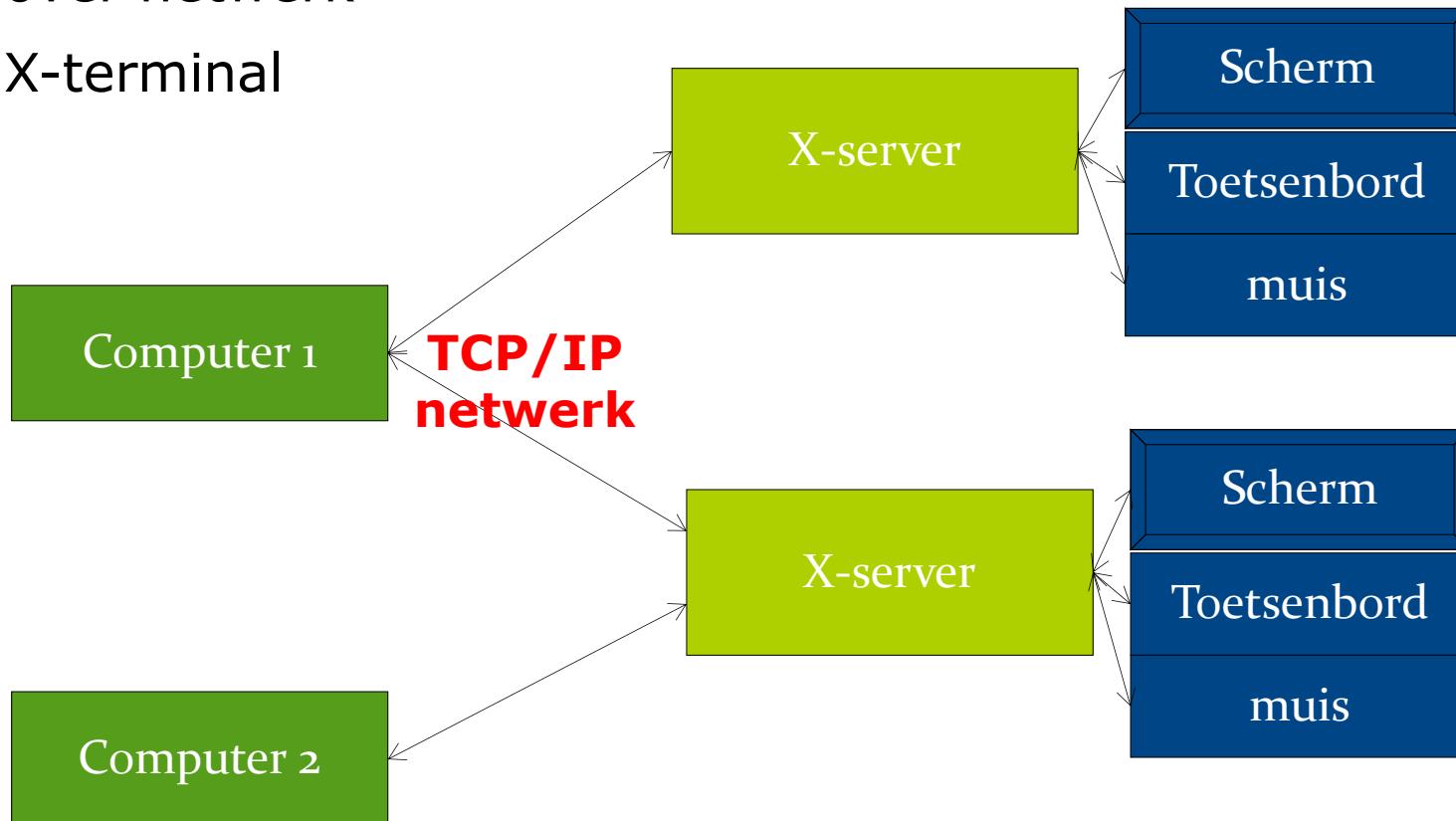
- meerdere processen, 1 scherm: windowing system nodig
- window = virtueel scherm
- windowing system = softwarelaag tussen applicatie en OS (die de hardware bestuurt)
- twee mogelijkheden:
 - in besturingssysteem ingebouwd (MS Windows)
 - als apart proces (X-Windowing System)

Inhoud

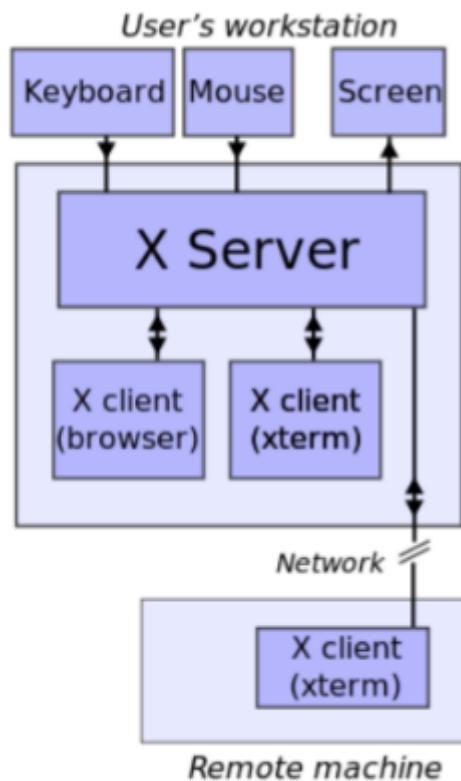
- User interfaces
- X Windows
- Virtualisatie
- Cloud computing
- Blade servers
- Herhalingsvragen

X Window System

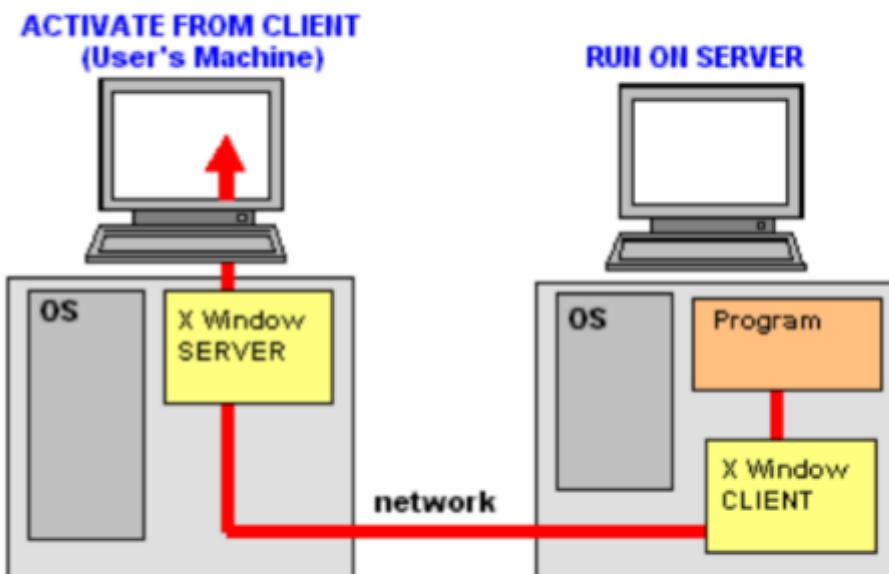
- bedoeld voor meerdere gebruikers op 1 systeem
- X applicaties van verschillende computers op 1 X-server
- over netwerk
- X-terminal



X Window System



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co., Inc.

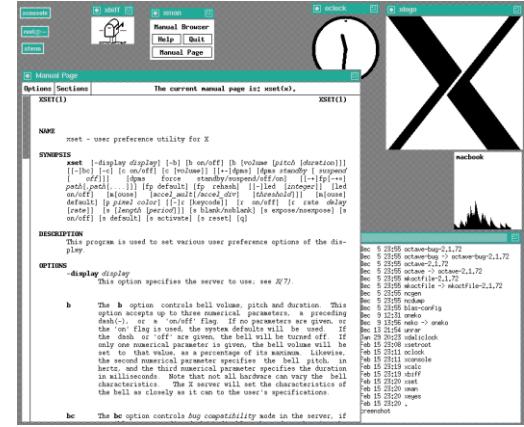


X Window System

- X, X11
- ontwikkeld door MIT
- iedere X-server is aparte machine of proces
- applicatie kan berichten sturen naar de X-server
 - via TCP/IP connectie
- X-server zorgt voor
 - hiërarchie van windows (zonder rand)
 - tekenen van punten, lijnen, bitmaps, ...
 - opvangen events en doorsturen naar applicatie
- Op Windows: MobaXterm, ...
- Wayland: vervanger van X

X Window System

- X server
 - applicaties
 - display manager
 - Greeter: inloggen
 - Run startup scripts
 - XDM protocol
 - window manager
 - is gewoon proces!
 - tekent randen van windows
 - regelt minimaliseren, maximaliseren, verplaatsen, vergroten, verkleinen van windows
 - desktop environment
 - Menu, Tool bar, widgets, file browser,...



Desktop environment

- **GNOME**

- Ontwikkeld in C
- Ubuntu: vanaf Ubuntu 17.10 wordt de standaard GNOME gebruikt

- **KDE Plasma**

- K Desktop environment
- Ontwikkeld in C++

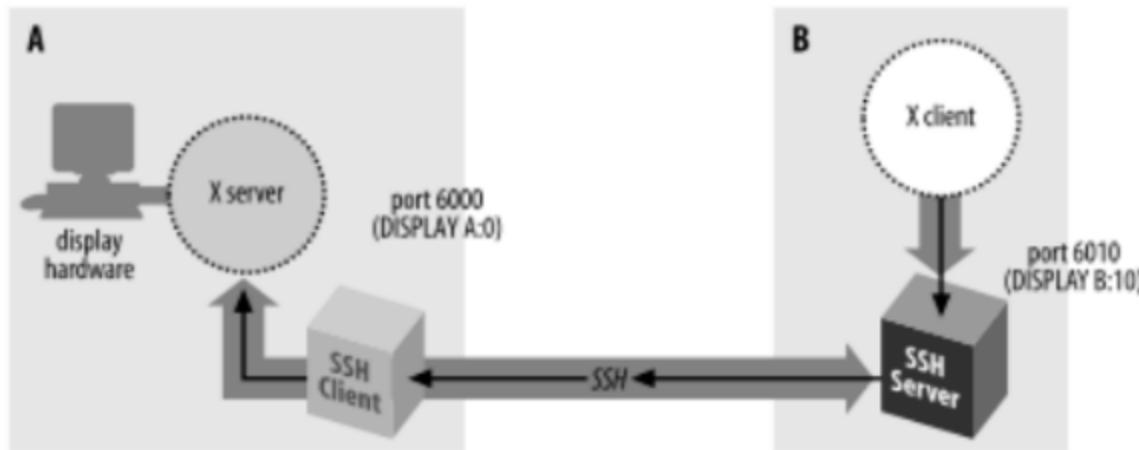
Desktop environment	Window Manager	Display Manager	Vb. Linux
GNOME	Mutter	gdm	Redhat, Ubuntu, Debian, Fedora
KDE Plasma	kwin	kdm/sddm	Kubuntu
Xfce	Xfwm	xdm	Xubuntu
Unity	Compiz	lightdm	Ubuntu 16.04

X Window lab



- Cntl-Alt-F3: inloggen
- sudo service gdm stop
- Cntl-Alt-F3
- sudo apt-get install xterm
- sudo apt-get install blackbox
- sudo xinit
- xeyes -geometry +400+400 &
- xcalc -geometry +600+100 &
 - kan je windows verplaatsen?
- blackbox &
 - kan je nu windows verplaatsen?
- fg
- Cntl-C
- Cntl-Alt-F3
- Cntl-C
- sudo service gdm start
- Cntl-Alt-F1

ssh X forwarding



- X applicatie die op Host B draait gebruiken vanaf Host A (m.a.w. afbeelden op de X server van Host A)
- Host B:
 - sudo apt-get install openssh-server
- Host A:
 - ssh -X hostB
 - Start X applicatie

Terminal services

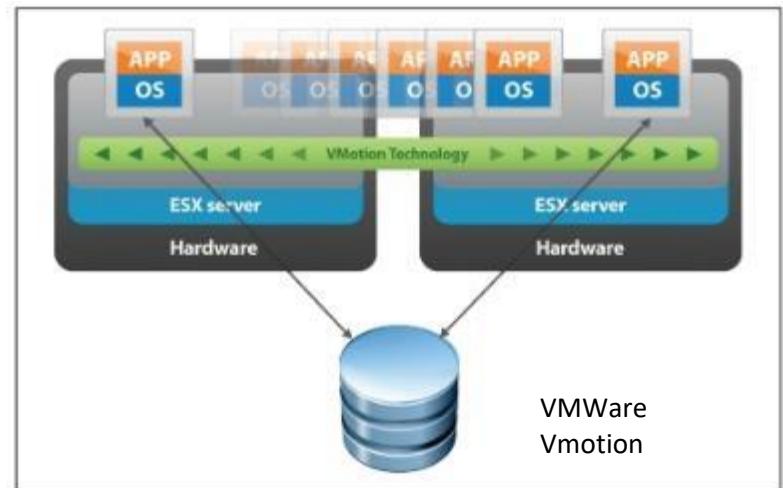
- is Microsoft antwoord op X-Windowing System
- terminal
 - = scherm/toetsenbord/muis
 - = "thin client"
- verbonden met computer via netwerk connectie
- eigen protocol: RDP
- oorspronkelijk bedoeld om sessie over te nemen (remote assistance)
- drivers voor scherm/toetsenbord/muis worden vervangen door drivers die communiceren met de terminal
- nu ook meerdere sessies (gebruikers) mogelijk

Inhoud

- User interfaces
- X Windows
- Virtualisatie
- Cloud computing
- Blade servers
- Herhalingsvragen

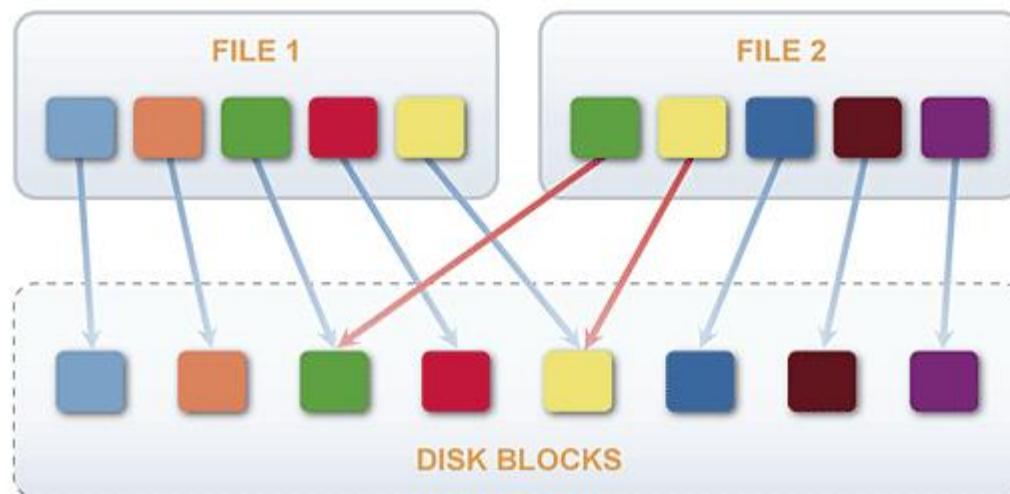
Waarom virtualisatie?

- Benutting HW
 - Gemiddeld 5 – 15 % belast !!!
 - Cost saving
- Deployment
 - Veel schnellere installatie van nieuwe server
 - Self-deployment
- High-availability en load-balancing
 - Pool van virtuele servers op verschillende fysische servers
- Management
 - Klonen
 - Snapshots
 - Anti virus in hypervisor
- VM op NAS of SAN
 - Deduplicatie



Deduplicatie

- Wat is data deduplicatie?

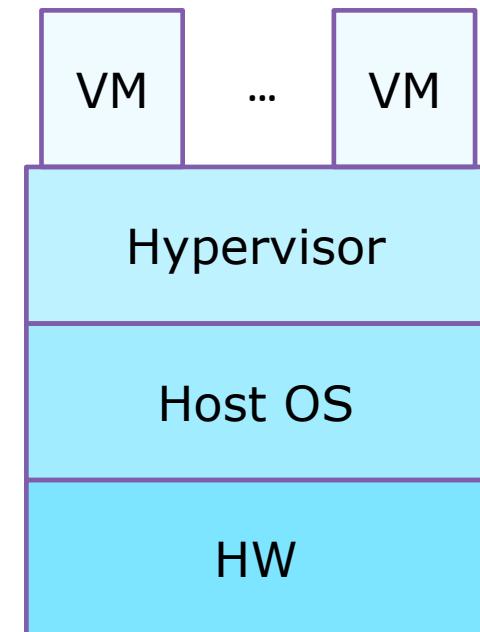


Soorten virtualisatie

- Full virtualisation
 - HW wordt gevirtualiseerd
- Paravirtualisation
 - Aanpassingen aan guest OS
- OS virtualisation
 - Kernel wordt gedeeld

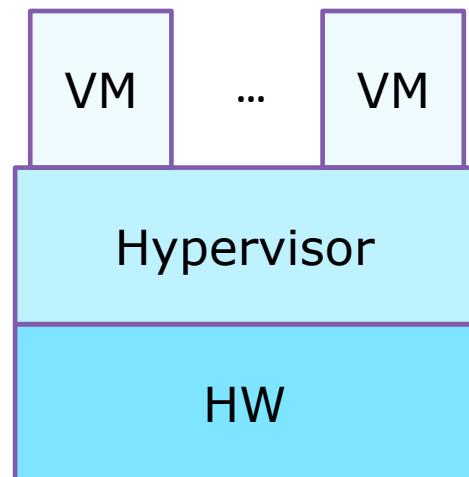
Full virtualisation - Hosted hypervisor

- Type 2 hypervisor
- Verschillende OS's op zelfde HW
- Hypervisor emuleert alle HW devices
- Hypervisor draait op **host OS**
- Guest OS:
 - Geen aanpassing nodig
 - Weet niet dat hij op VM runt
- Vb: VirtualBox



Full virtualisation - Bare Metal hypervisor

- Type 1 hypervisor
- Verschillende OS's op zelfde HW
- Hypervisor emuleert alle HW devices
- Hypervisor draait rechtstreeks op **HW**
- Guest OS:
 - Geen aanpassing nodig
 - Weet niet dat hij op VM runt
- Vb: VMWare ESX



Paravirtualisation

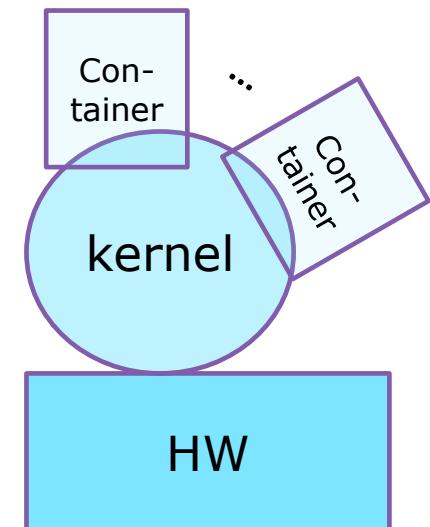
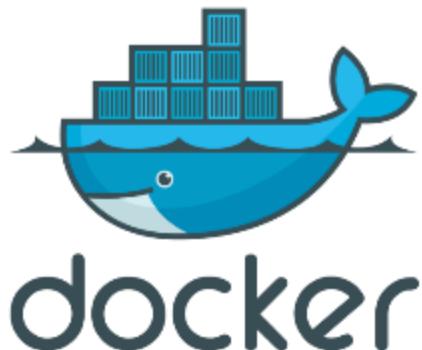
- Verschillende OS's op zelfde HW
- Guest OS:
 - Wel aanpassing nodig aan OS kernel *
 - Geen aanpassing aan User Applications
- Vb: Xen



* Hypervisor minder overhead

OS virtualisation

- OS host kernel gedeeld door alle guests
- Guest OS = host OS
- Snel, geen vertaling of HW abstractie
- “guest **containers**”
- Vb: Solaris containers, Docker, LXD Containers



Inhoud

- User interfaces
- X Windows
- Virtualisatie
- Cloud computing
- Blade servers
- Herhalingsvragen

Cloud computing

- Public, Private, Hybrid
- IAAS: Amazon Elastic Compute Cloud (EC2)
- PAAS: AWS, Microsoft Azure Web Sites,...
- SAAS: Salesforce.com

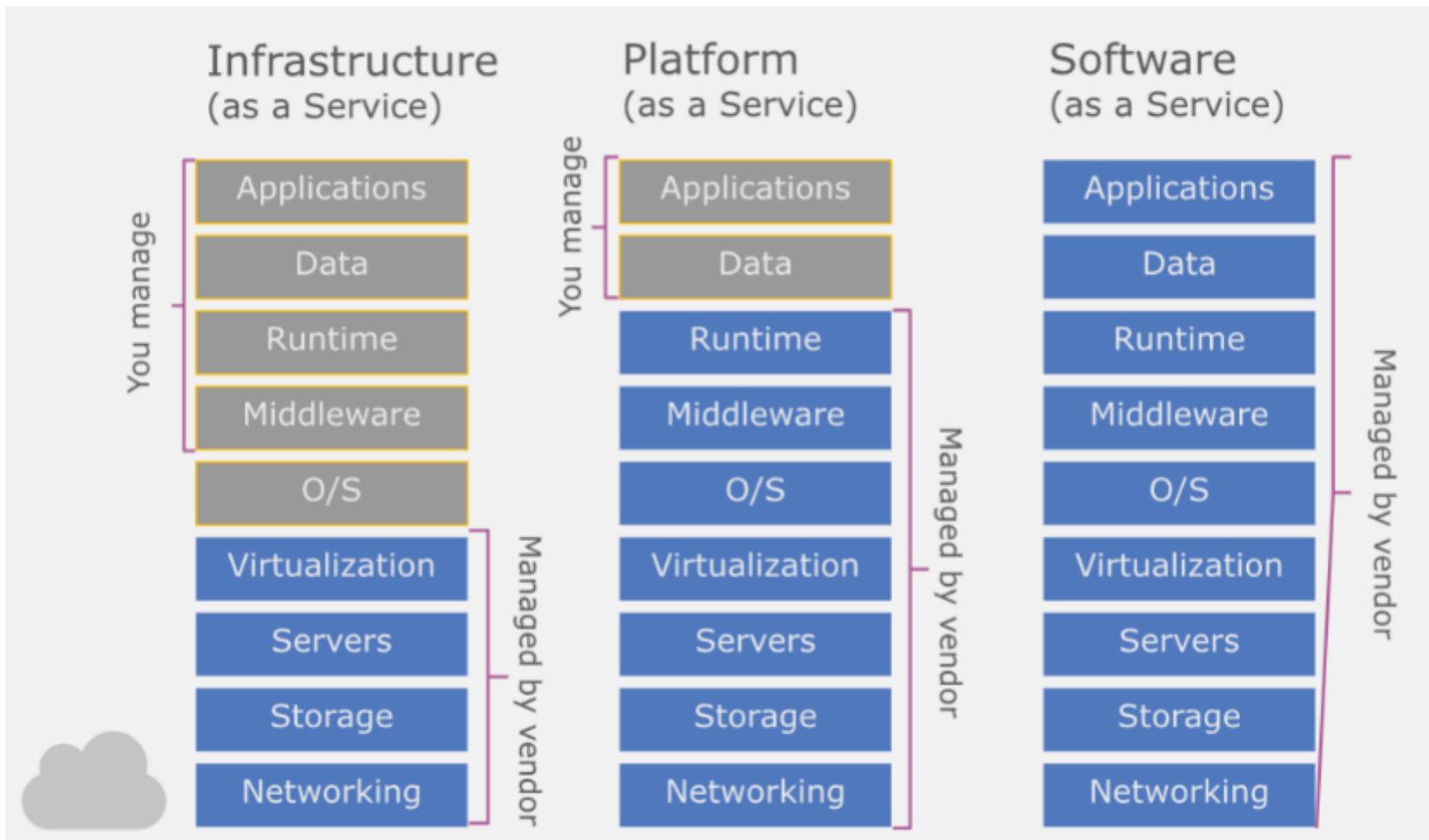
Cloud computing

- Private Cloud
 - Dedicated VMs (your organisation only)
 - Complete infrastructure control
 - Intelligent resource monitoring
 - Automated self-provisioning
 - Local or remote virtual infrastructure, for your organisation only.
- Public Cloud
 - Resource consumption without management
 - Resource metering
 - “Pay as you go”
 - Remote applications/infrastructure, shared with others.
- Hybrid Cloud
 - Organisation has private and public cloud resources
 - Best of both worlds
 - “Cloud connectors”
 - Bridge between private and public.

Cloud computing

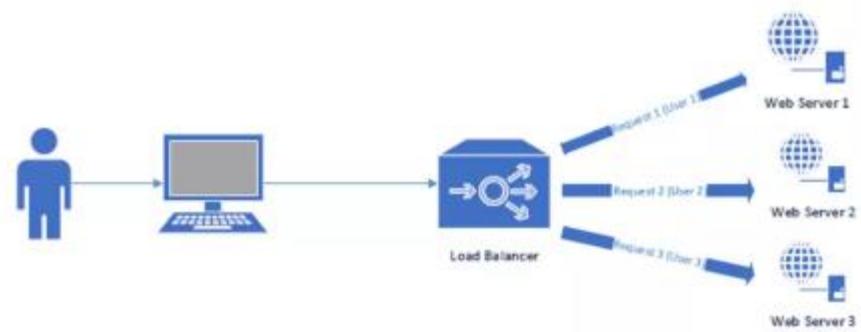
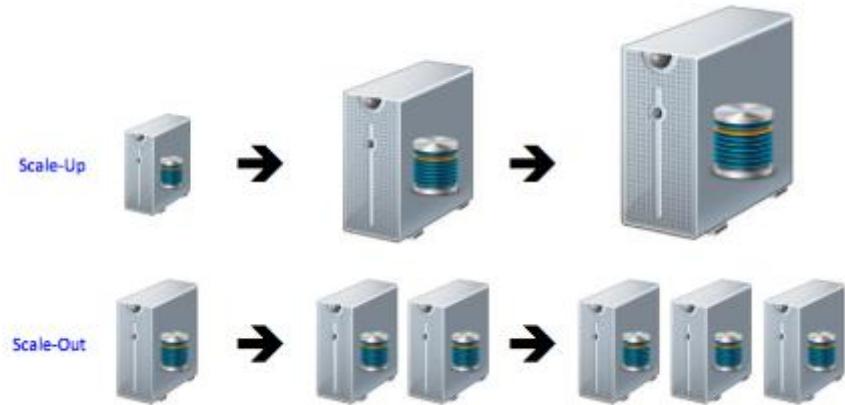
- **Infrastructure as a Services (IaaS)**
 - Developers/Engineers get virtual machines (VM's), to which they have full access
 - Provides total control, but also total responsibility
- **Platform as a Services (PaaS)**
 - Developers provide an application, which the platform runs
 - Provides an easy-to-use execution environment, with limited features
- **Software as a Services (SaaS)**
 - End-users consume a hosted application

Cloud computing



Cloud computing

- Vertical scaling (**scale up**): adding more power (CPU, RAM) to an existing server
- Horizontal scaling (**scale out**): adding more servers
 - Sessie informatie (keys,...) niet bewaren op (file systeem van) de applicatie server, maar in een gemeenschappelijke DB (of in een in-memory datastore zoals Redis)
 - Indien toch sessie informatie bewaard wordt op de server: Session Affinity (requests van specifieke client steeds naar dezelfde backend sturen) aanzetten op loadbalancer



Inhoud

- User interfaces
- X Windows
- Virtualisatie
- Cloud computing
- Blade servers
- Herhalingsvragen

Blade servers

- Wat zijn Blade servers?
Wat hebben ze
gemeenschappelijk?



Inhoud

- User interfaces
- X Windows
- Virtualisatie
- Cloud computing
- Blade servers
- Herhalingsvragen

Voorbeeld Examenvragen

- Uit welke componenten bestaat X-Windows?
- Wat is een Window manager?
- Wat is RDP?
- Wat is een thin client?
- Wat is data deduplicatie?
- Waarom wordt virtualisatie toegespast?
- Wat is full virtualisatie, paravirtualisatie, hosted hypervisor, OS virtualisatie (met tekening erbij)?
- Wat is Public, private, hybrid cloud?
- Wat is IAAS, PAAS, SAAS?
- Wat zijn Blade servers, wat delen ze?