# Computersystemen 2

## Booting & I/O output

### Inleiding

**Besturingssysteem**

| Software | Communicatie tussen hardware en user | Interface naar gebruiker van computer |
|---|---|---|

**Taken van besturingssysteem**

- Boot-proces
- Hardware abstraction
- i/o management
- File management
- Proces management
- Memory management
- Window management

### Herhaling computersystemen 1

#### 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 |

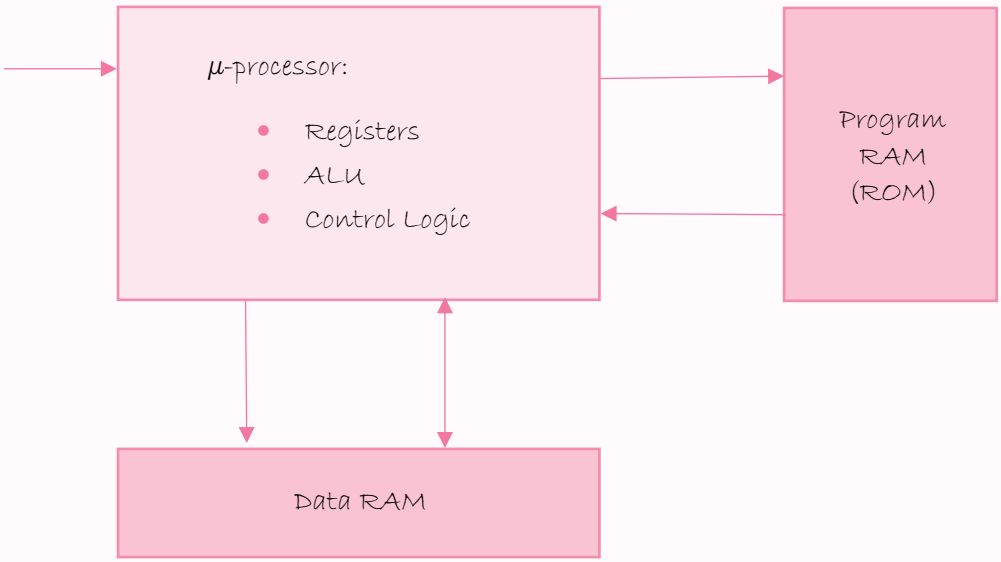
- Decimaal

| 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 |

## Harvard architectuur

Ontwerp van CPU's dat gescheiden bussen (en adresruimtes) heeft voor data en instructies

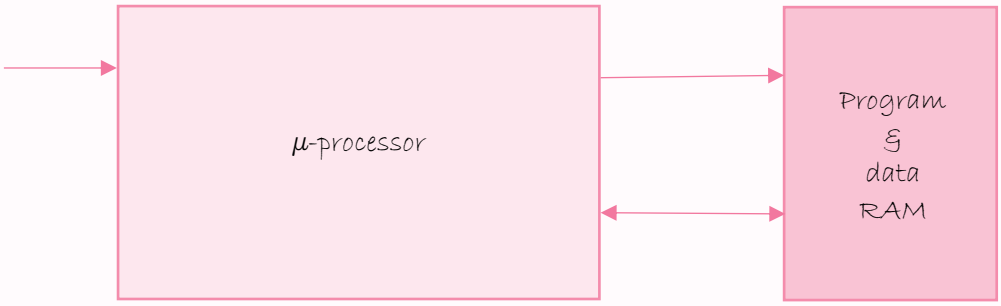
- Snellere processor
  - Ophalen van volgende instructie en wegschrijven van het resultaat kan gelijktijdig gebeuren
- Uitsparing opslagruimte
- Wordt voornamelijk gebruikt in microcontrollers en DSP's




## Von Neumann-architectuur

Ontwerparchitectuur voor een elektronische digitale computer, waar de delen bestaan uit

- Central Processing Unit (CPU)
  - Bestaat uit
    - Arithmetic/Logic Unit (ALU)
    - Control Unit
- Memory Unit
- Input en Output Devices



## RAM-geheugen

| Code | Data |
| --- | --- |
| Machine-code: bytes | Getallen (bytes, words, floating point ...) |
| Instructie: instructie code + argumenten | Tekst (ASCII, Unicode, EBCDIC ...) |

Zowel code als data zijn bytes

## Registers

Gegevensregisters (=tijdelijke opslagplaatsen)

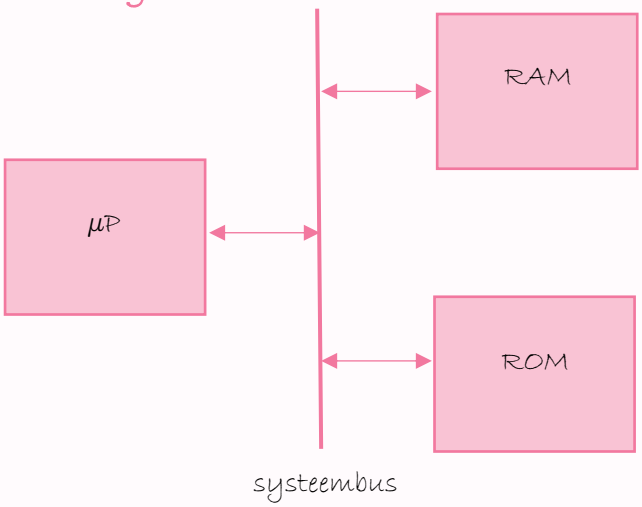
Adresregisters

- Indexregisters
- Segmentregisters
- Stackpointer

Stuur- en statusregisters

- Program counter / instruction pointer
- Flags

## Booting




Booting is de procedure waarbij eerst de systeemcomponenten worden getest door het Basic Input/Output System (BIOS of UEFI) en vervolgens de bootloader wordt geladen en opgestart.

### Read-Only Memory (ROM)

→ Is een geheugenopslagmedium in computers en andere elektronische apparaten

In ROM zit firmware, gekend als BIOS

Bevat:

- POST
  - Power-On Self-Test
- HAL
  - Hardware Abstraction Layer
- Shell (interface voor gebruiker)
- Code om andere code te laden van extern medium (bv.harde schijf)

## Booting

- ROM firmware
- Boot loader
  - Als laatste stap laadt de firmware de boot loader van de startup disk naar RAM en voert deze uit
- Kernel
  - Boot loader start kernel na het laden van disk naar RAM
- Volledig OS
  - Processen opstarten, filesystemen mounten, netwerk configureren …

| | BIOS | UEFI |
|---|---|---|
| Laadt code van | MBR = 1e sector HDD | EFI-file op EFI systeem partitie (bv. \EFI\BOOT\BOOTx64.EFI) |
| Partitietabel | In MBR | In GPT (GUID part. tabel) |
| Grootte partitietabel | 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 filesystem | 2 TiB | 8 ZiB |
| Kan secure boot | Nee | Ja |
| Boot-loader nodig | Ja altijd | Kan rechtstreeks kernel opstarten |

**BIOS**

**B**asic **I**nput/**O**utput **S**ystem

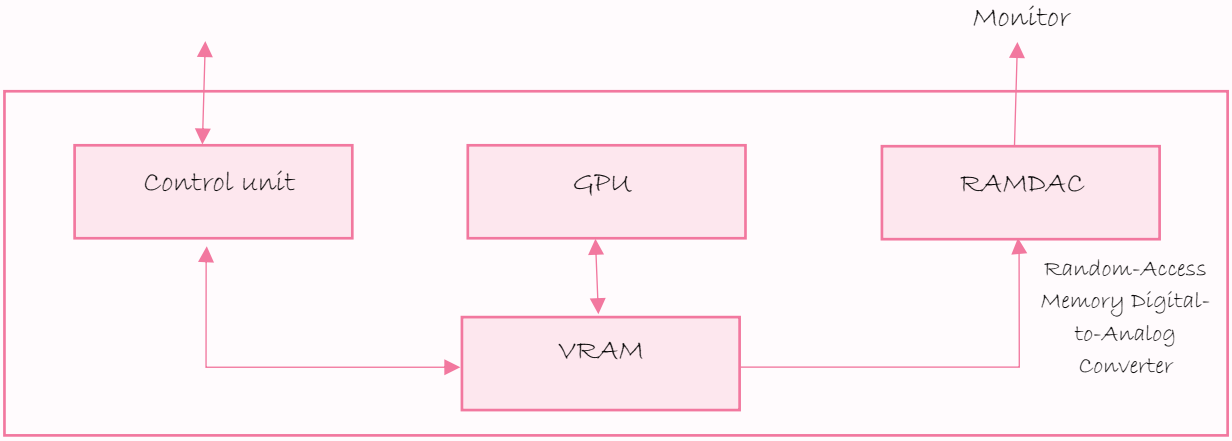
**UEFI**

(**U**nified) **E**xtensible **F**irmware **I**nterface

# I/O output

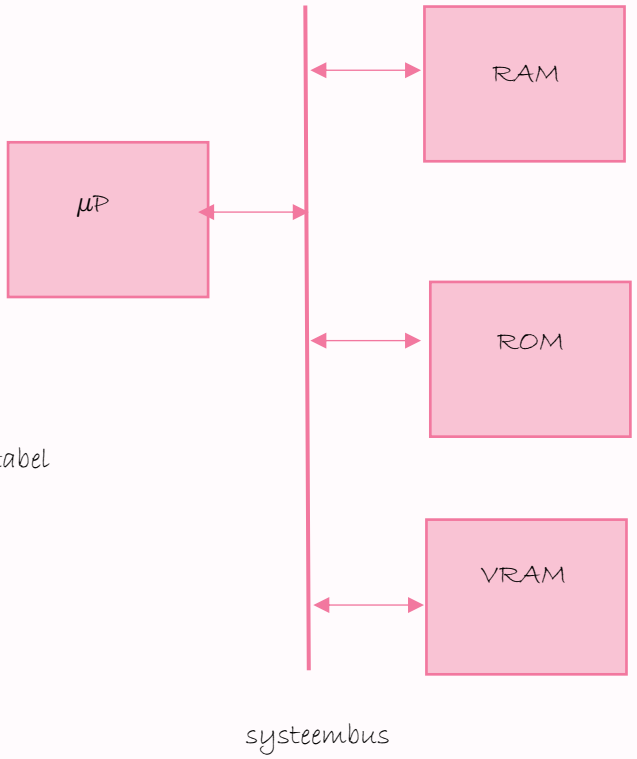
## Beeldscherm

Beeldscherm wordt bestuurd door grafische kaart



## Input/Output – het scherm

- 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)

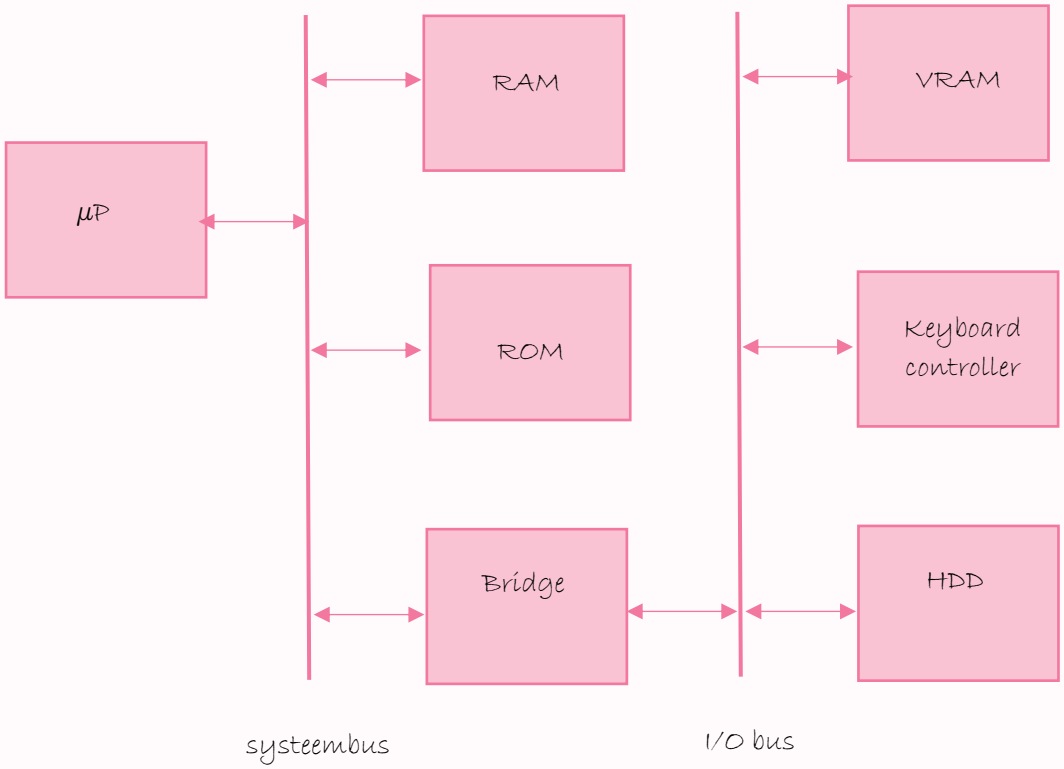



Memory-mapped I/O uses the same address space to address both memory and I/O devices

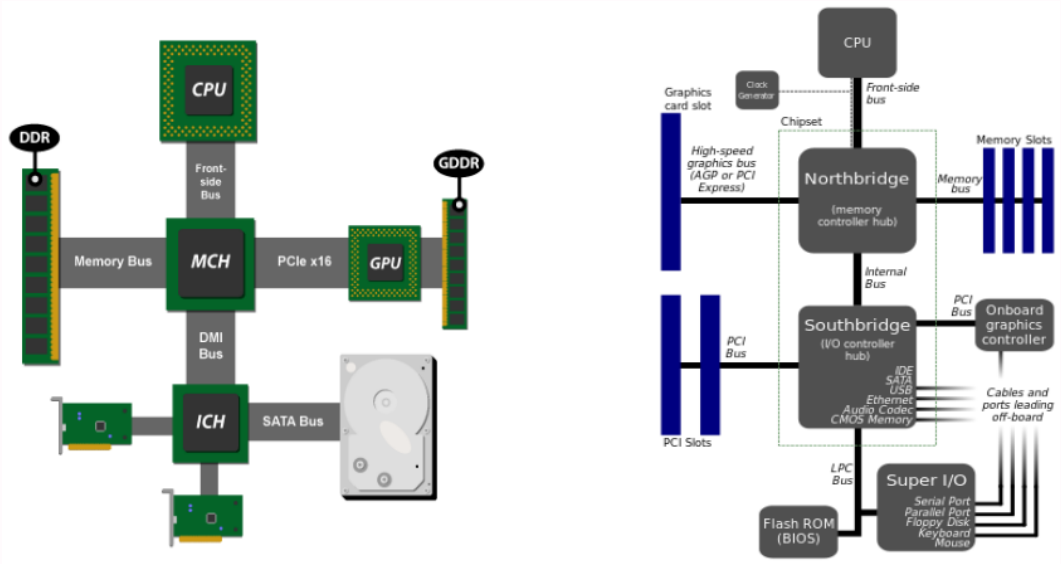
# I/O beheer en bestandsbeheer

## I/O beheer

### Input en output - toetsenbord

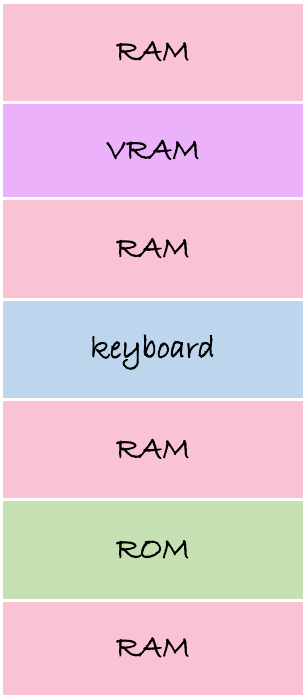



### PC architectuur




MCH = Memory Controller Hub (Northbridge)
ICH = Input/Output Controller Hub (Southbridge)

## Input en output - toetsenbord




Keyboard controller bevat 3 "registers" (register keyboard != register CPU)

- Data
- Status
- Control

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
- Nadeel:
  - CPU zou continu de loop moeten doen om te checken
- Het gebeurt dus niet op deze manier

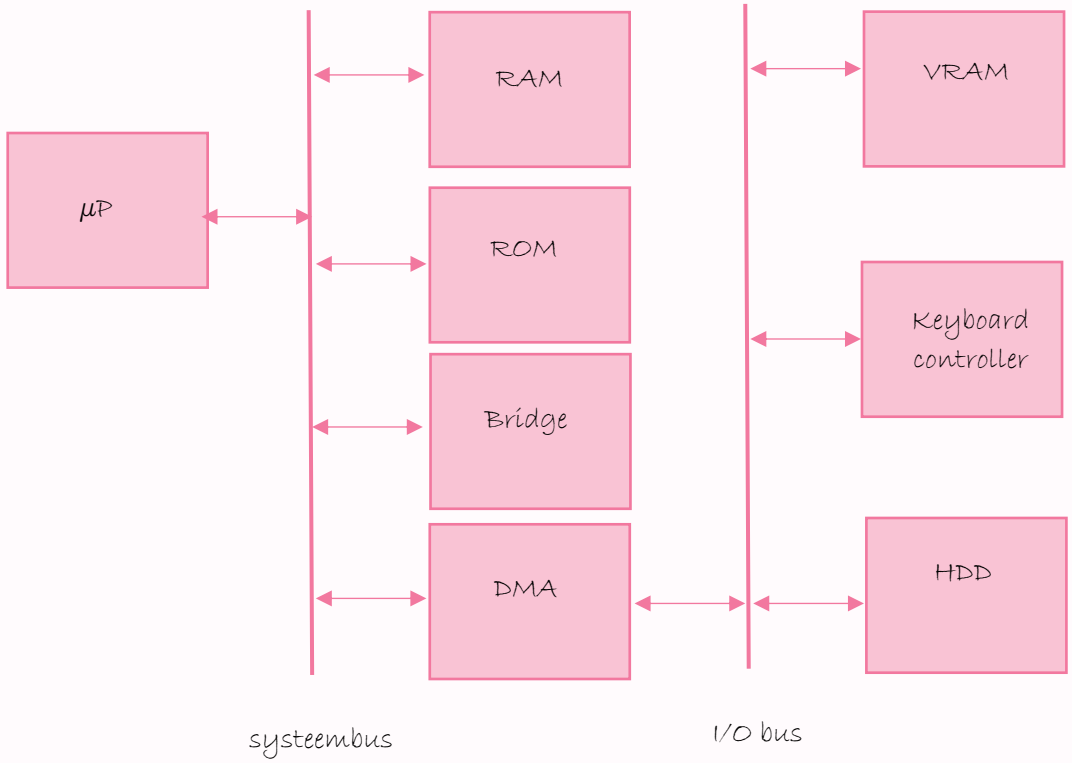
### Interrupt driven I/O

Processor krijgt een sein 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

ISR zal toetsaanslagen in "circulaire buffer" in RAM zetten

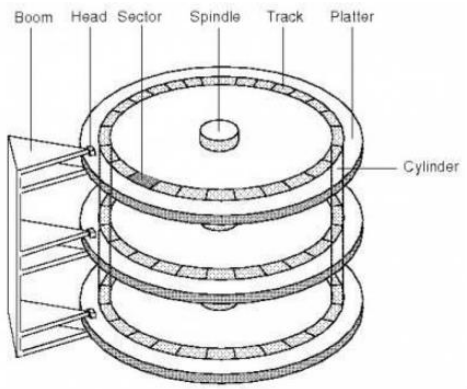
- Circulaire buffer
  - Vaste grootte
  - Lees = schrijf => empty
  - Schrijf = lees - 1 => full + risk for overflow
  - Applicatie zal buffer af en toe lezen




- Interrupt driver I/O veel efficiënter dan programmed I/O
- Bij lezen HDD, 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!

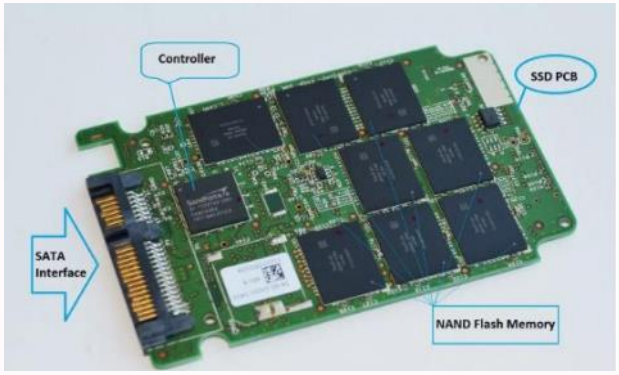
## HDD

- **Hard Disk Drive**
- Harde schijf
  - Disks
  - Heads
  - Cilinders
  - Tracks
  - Sectors
- Magnetisch
  - Weinig schokbestendig
  - Latency
- Bij delete file
  - File wordt verwijderd uit directory tabel
  - Sectoren van file blijven op disk staan (en kunnen nadien overschreven worden)




## SDD

- **Solid-State Drive**
- PCB:
  - Interface: SATA, mSATA ...
  - Controller (firmware)
  - NAND Flash Memory
- Voltage levels:
  - SLC (Single Level Cell)
  - MLC (Multi Level Cell)
  - Zelf TLC (Triple Level Cell)
- Geschreven page mag niet overschreven worden
  - Eerst leeg maken
  - Bij delete file:
    - Niet enkel file verwijderen uit directory
    - TRIM instructie aan OS toegevoegd
- Cellen degraderen bij schrijven
  - Aantal write cycles is beperkt
  - Geen defragmentatie!
  - Wear-leveling



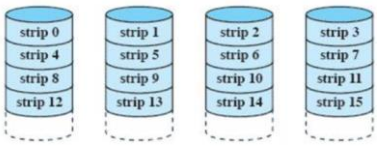
# RAID

- **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
- RAID heeft verschillende niveau's
  - 0 - 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)

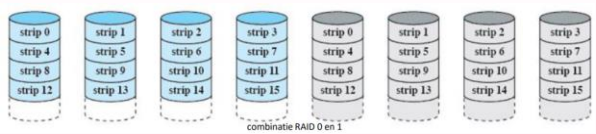
- Verspreid de data over de schijven
- Voor-en nadeel?
  - voordeel: snellere toegang (parallel)
  - nadeel: fout in 1 schijf ==> alle data verloren




## RAID-1 (mirroring)

Verdubbel het aantal schijven

- Bewaar alle data 2 keer (mirroring)
- Voor-en nadeel?
  - Voordeel: foutcorrectie mogelijk, snelle leestoegang
  - Nadeel: duur (2x aantal schijven nodig)

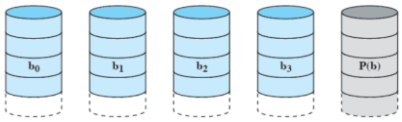


combinatie RAID 0 en 1

## RAID-3

Gebruik 1 schijf voor "pariteitsbits" (redundancy)

- 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

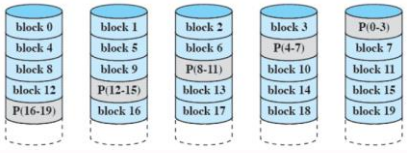
De pariteitsblokken worden gespreid over de schijven

- 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



## Caching

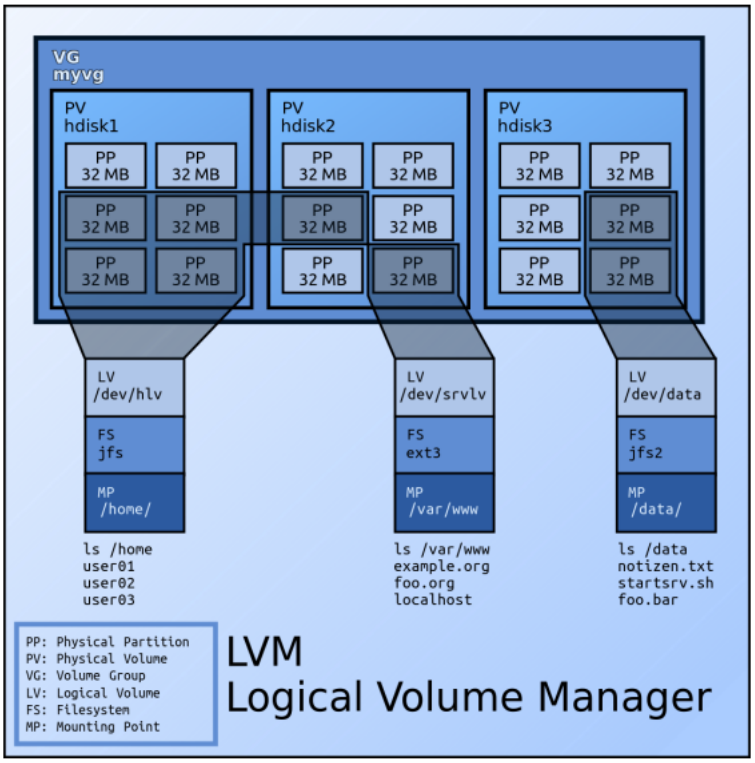
schijftoegang is traag

- oplossing
  - schrijf gegevens niet direct naar schijf, maar houdt ze in het geheugen (cache)
  - hou gelezen blokken ook in het geheugen
  - schrijf gegevens pas naar schijf als processor idle is
- nadeel
  - als de stroom uitvalt...
  - schijf moet worden afgesloten alvorens te verwijderen (vb: USB)

## LVM

### Logical Volume Management

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



# Bestandsbeheer

## File management

**Bestandsbeheer**

is een essentieel onderdeel van een besturingssysteem (OS) dat zich bezighoudt met de organisatie en manipulatie van gegevens op opslagapparaten.

- Bestanden: Gegevens die logisch zijn gegroepeerd en een naam hebben.
- Directories (Folders, mappen): Hiërarchische structuren die bestanden organiseren.
- Links (Shortcuts): Verwijzingen naar bestanden of mappen.
- Vuilbak: Een locatie waar verwijderde bestanden tijdelijk worden opgeslagen.
- Meta-informatie (file attributes): Eigenschappen zoals eigenaar, lees/schrijfrechten, compressiestatus, back-upbehoefte, creatietijd, laatste wijziging en laatste toegang.

### Bestandsattributen

#### Belangrijke Attributen:

- Eigenaar van het bestand:
  - De gebruiker die het bestand bezit.
- Lees/Schrijfrechten:
  - Bijvoorbeeld in Unix-stijl: rwx rwx rwx.
- Compressie:
  - Geeft aan of het bestand gecomprimeerd is.
- Back-upbehoefte:
  - Geeft aan of het bestand moet worden geback-upt.
- Tijdstippen:
  - Creatie, laatste wijziging, laatste toegang.
- Applicatieassociatie:
  - De toepassing waarmee het bestand moet worden geopend.
- Versiebeheer:
  - Geeft aan of er verschillende versies van het bestand zijn.

### Taken van Bestandsbeheer

#### Basistaken:

- Creëren, verwijderen, en wijzigen van bestanden.
- Autoriseren van toegang tot bestanden.
- Creëren en verwijderen van directories.
- Kopiëren, verplaatsen en andere bestandsmanipulaties.
- Omzetten van bestandsbewerkingen naar blokbewerkingen op de schijf.

## Interne fragmentatie

**Interne fragmentatie ontstaat wanneer de toegewezen blokken voor een bestand niet volledig worden gebruikt, wat resulteert in verspilde ruimte.**

Voorbeeld:

- Blokken van 4096 bytes, bestandsgrootte van 10.000 bytes, vereisen 3 blokken, waarbij het laatste blok niet volledig wordt benut.

Besluit:

- Het advies is om de blokgrootte zo klein mogelijk te maken om interne fragmentatie te verminderen.

## Externe fragmentatie

**Externe fragmentatie treedt op wanneer bestanden verspreid zijn over niet-aangrenzende blokken op de schijf.**

Probleem:

- Sequentiële lezing van een bestand kan traag zijn omdat de kop van de harde schijf heen en weer moet bewegen.

Oplossing:

- Defragmentatie, vooral effectief bij grote bestanden in aaneengesloten blokken.

## File Allocation Table (FAT)

**File Allocation Table (FAT)**
een bestandssysteem dat bestandslocaties opslaat op een opslagapparaat zoals een harde schijf

Het File Allocation Table (FAT)-bestandssysteem is oorspronkelijk ontwikkeld in 1977 voor floppy disks en later aangepast voor harde schijven en andere apparaten. Er zijn drie belangrijke varianten: FAT12, FAT16 en FAT32. Het werd veel gebruikt in het DOS- en Windows 9x-tijdperk, maar werd vervangen door NTFS op Windows XP. Toch wordt FAT nog steeds gebruikt op flashstations en in draagbare apparaten vanwege de compatibiliteit.

FAT gebruikt een tabel, de **File Allocation Table (FAT)**, om clusters op de schijf te koppelen aan bestanden. Elke vermelding in de FAT bevat het nummer van het volgende cluster in een bestand. Het besturingssysteem volgt deze keten om het volledige bestand te lezen. FAT wordt nog steeds gebruikt op schijven die door verschillende besturingssystemen worden gedeeld en in draagbare apparaten zoals camera's en USB-flashstations.

### FAT Voor- en Nadelen

Voordelen:

- Eenvoudige structuur.

Nadelen:

- Traagheid door frequente updates van de FAT.
- Noodzaak van mirroring (FAT moet twee keer worden geschreven).
- → Oplossing: caching (FAT in het geheugen houden en periodiek naar de schijf schrijven

## Unix file system

**Unix inodes zijn datastructuren die informatie bevatten over bestanden in een Unix-bestandssysteem.**

- Elke inode vertegenwoordigt één bestand en bevat details zoals bestandsnaam en lengte.
- Een inode heeft 16 pointers, waarvan de eerste 13 direct verwijzen naar blokken van het bestand.
- De 14e pointer verwijst naar een 'single indirect block' met extra blok-pointers.
- De 15e en 16e verwijzen naar 'double' en 'triple indirect blocks', die pointers bevatten naar meer blokken.

Als de blokgrootte bijvoorbeeld 512 bytes is en pointers worden opgeslagen als 32-bits unsigned integers, kan een blok 128 pointers bevatten. Zonder indirecte blokken kan een bestand maximaal 13 * 512 bytes groot zijn. Met één indirect blok wordt dit 13 * 512 + 128 * 512, en met double indirects wordt de maximale bestandsgrootte ongeveer 1 GiB.

De blokgrootte van het bestandssysteem heeft invloed op zaken als fragmentatie, bestandsgrootte en prestaties. Bij het maken van een bestandssysteem kun je de blokgrootte instellen met bijvoorbeeld 'mkfs.ext4 -b block_size' voor ext4 of 'mkfs.vfat -s sectors_per_block' voor het vfat-bestandssysteem.

## Zettabyte File System (ZFS) & B-tree FS (Btrfs)

**ZFS staat voor Zettabyte File System.**

**Btrfs staat voor B-tree FS, of Better FS.**

- Ze combineren bestandssystemen en logische volumebeheerders.
- Bieden bescherming tegen gegevenscorruptie, hoge opslagcapaciteit, copy-on-write, snapshots, cloning en RAID-functionaliteit.
- Voor ZFS-configuratie op Linux met twee partities: *zpool create new-pool /dev/sdb1 /dev/sdc1*.
- Verschillende configuraties mogelijk, zoals striping (*zpool create new-pool /dev/sdb1 /dev/sdc1*), mirroring (*zpool create new-pool mirror /dev/sdb1 /dev/sdc1*), enz.
- Commando's zoals *zpool status* voor het bekijken van de status en *zpool destroy new-pool* voor het verwijderen van een ZFS-pool.

# Geheugenbeheer

## Laden van OS

- Een boot-loader laadt het besturingssysteem in het geheugen.
- Dit proces wordt mogelijk gemaakt door de Von Neumann-architectuur.
- Verschillende strategieën voor het ontwerp van de kernel zijn onder andere monolithische kernel, modulaire kernel en micro-kernel.

### Kernel-ontwerpen:

#### Monolithische kernel:

Alle OS-functionaliteiten zijn samengevoegd in één blok code (één proces).

Bij wijzigingen, zoals het toevoegen van een nieuwe device driver, moet de kernel opnieuw worden gelinkt en het systeem opnieuw opstarten.

#### Modulaire kernel:

De kernel is één proces dat dynamisch gelinkt is.

Kernelmodules (bijv. device drivers) kunnen dynamisch worden gelinkt (of losgekoppeld) tijdens runtime.

#### Micro-kernel:

De micro-kernel bevat basis-OS-functionaliteiten.

Overige functies, zoals "drivers," zijn geïmplementeerd als aparte processen (bijv. voor HDD, scherm, netwerk, file management).

Processen communiceren met elkaar via Inter-Process Communication (IPC).

### Ubuntu Kernel Modules:

1. Loadable kernel modules bij Ubuntu Linux hebben de extensie *.ko*.
2. Ze bevinden zich op een specifieke locatie op het bestandssysteem.
3. Het commando *lsmod* kan worden gebruikt om te zien welke loadable kernel modules momenteel zijn geladen.

## Laden van processen

### Programma's laden:

- Het besturingssysteem (OS) start op en zit klaar in het geheugen, samen met geladen I/O-stuurprogramma's.
- Bij het starten van een programma wordt de code/data in het geheugen geladen, en de uitvoering begint bij de startinstructie.
- Een uitdaging is dat het programma op verschillende locaties in het RAM-geheugen kan belanden, wat problemen oplevert, zoals onvoorziene jmp (jump) instructies en onzekerheid over de locatie van gegevens in het geheugen.

**Laden - Relocatie:**

Processen moeten op verschillende plaatsen in het geheugen kunnen staan.

Soms worden processen verplaatst, wat vereist dat geheugenreferenties worden aangepast.

Er zijn logische adressen (relatief ten opzichte van het begin van het proces) en fysieke adressen (absoluut ten opzichte van het begin van het geheugen).

Software gebruikt altijd logische adressen, en de vertaling naar fysieke adressen wordt uitgevoerd door de hardware.

**Voorbeeld van Relocatie (8086):**

Voor de startadressen worden segmentregisters gebruikt.

Segmentregisters worden met 16 vermenigvuldigd omdat de registers 16-bit zijn en de adresbus 20-bit is.

Deze vermenigvuldiging gebeurt per segment, zoals code, data (globale variabelen en heap), en stack (lokale variabelen en parameters).

## Linken

Een process linken:

- In de praktijk bestaat een programma uit verschillende modules.
- Elke module wordt gecompileerd tot een object-bestand met code en labels.
- De linker voegt object-bestanden samen tot een uitvoerbaar bestand (executable) en vervangt labels door adressen.

### Statisch linken

**Statisch linken is een compilatieproces waarbij alle modules van een programma worden samengevoegd tot een op zichzelf staand uitvoerbaar bestand voordat het wordt uitgevoerd.**

Dit resulteert in een autonoom bestand dat alle benodigde code en gegevens bevat. Voordelen zijn onder andere eenvoudige distributie en onafhankelijkheid van de externe omgeving. Echter, de nadelen omvatten een potentieel grotere bestandsgrootte, complexiteit bij updates en geheugeninefficiëntie door het dupliceren van bibliotheken.

### Dynamisch linken

**Dynamisch linken is een proces waarbij de definitieve binding van externe verwijzingen plaatsvindt tijdens de uitvoering van het programma, niet tijdens de compilatie.**

Dit biedt voordelen zoals flexibiliteit bij het toevoegen van nieuwe modules en gedeeld geheugengebruik. Dynamische modules kunnen worden gedeeld en bijgewerkt zonder het hele programma opnieuw te compileren. Dit resulteert in een efficiënter gebruik van resources en maakt het programma aanpasbaar aan veranderende vereisten.

# Call stack

Een programma is opgebouwd uit verschillende elementen:

- Code: Bevat de instructies in de machinecode die het programma uitvoert.
- Data: Omvat globale variabelen, statische data en dynamische variabelen (heap).
- Stack: Hier worden lokale variabelen, parameters, return-values en return-adressen beheerd.

### Werking van de Call Stack:

De call stack is een mechanisme dat de uitvoering van functieoproepen beheert. In het voorbeeldprogramma hieronder wordt de functie *fac* opgeroepen vanuit de *main*-functie. Tijdens deze oproep wordt een lokale variabele *a* gedefinieerd. De *fac*-functie berekent de faculteit van een getal en gebruikt de stack om tussenresultaten en return-adressen op te slaan.

```
1. main() {
2.  int a = fac(2);
3. }
4.
5. int fac(int i) {
6.  int result = 0;
7.  if (i < 2) result = 1;
8.  else result = fac(i - 1) * i;
9.  return result;
10. }
11.
```

Dit voorbeeld illustreert hoe de call stack wordt gebruikt om de controle over de uitvoering van het programma te behouden, waarbij elke functieoproep zijn eigen set lokale variabelen en return-adressen heeft.

# Partitioning

Geheugenbeheer omvat het bijhouden van het gebruik van geheugendelen.

Er zijn verschillende strategieën:

1. Partitionering:
    a. Vaste Partities:
        i. Gelijke grootte, bij opstarten vastgelegd.
        ii. Eenvoudig, maar beperkt in flexibiliteit.
        iii. Kan leiden tot interne fragmentatie.
    b. Dynamische Partities:
        i. Verschillende groottes, bij opstarten vastgelegd.
        ii. Meer flexibiliteit voor grotere processen.
        iii. Kan nog steeds interne fragmentatie veroorzaken.
        iv. Partities dynamisch gecreëerd bij het laden van een proces.
        v. Keuzes bij plaatsing: best-fit, first-fit, next-fit.
        vi. Kan externe fragmentatie veroorzaken, oplossing: compactation.

2. Segmentering:
    a. Geheugen wordt opgedeeld in logische segmenten.
    b. Elk segment heeft een naam en lengte.
    c. Meer flexibiliteit dan partitionering, maar kan fragmentatie veroorzaken.
3. Paging:
    a. Geheugen wordt opgedeeld in pagina's van vaste grootte.
    b. Processen worden in pagina's geladen.
    c. Voorkomt externe fragmentatie, maar kan leiden tot interne fragmentatie.
    d. Meer complex, maar efficiënt gebruik van geheugen.

Opmerking: Moderne besturingssystemen maken vaak gebruik van paging en hebben methoden ontwikkeld om fragmentatie te minimaliseren.

## Segmenting

### Segmenting

een geheugenbeheertechniek waarmee een proces kan worden opgedeeld in segmenten die niet noodzakelijk aaneengesloten zijn in het geheugen.

Elk segment krijgt een nummer, en een logisch adres bestaat uit een segmentnummer en een offset binnen dat segment. Deze techniek biedt flexibiliteit, omdat segmenten van verschillende grootte kunnen zijn en niet aaneengesloten hoeven te zijn.

### Maximale Grootte van Segment:

Er is geen vaste regel voor de maximale grootte van een segment.

De grootte wordt bepaald door het bijbehorende segmentnummer en offset binnen dat segment.

### Maximaal Aantal Segmenten:

Het aantal segmenten wordt beperkt door het aantal bits dat wordt gebruikt om het segmentnummer weer te geven.

### Dynamische Partitionering:

Elk proces neemt één aaneengesloten partitie in.

### Segmentatie:

Een proces kan worden verspreid over verschillende segmenten, die niet per se aaneengesloten zijn.

### Segmentinformatie:

Elk segment heeft een nummer, een beginadres en een lengte.

### Logisch Adres:

Bestaat uit segmentnummer en offset binnen dat segment.

### Vertaling:

Hardware vertaalt logische adressen naar fysieke adressen.

### Foutafhandeling:

Bij overschrijding van de grens veroorzaakt dit een interrupt (bijv. "segmentatiefout").

## Paging

een geheugenbeheertechniek waarbij het RAM in vaste blokken, frames wordt verdeeld.

Bij het opstarten krijgt een programma's pagina's toegewezen aan vrije RAM-frames. De page table houdt bij welke pagina's waar zitten, waardoor logische naar fysieke adressen snel vertaald kunnen worden. Dit optimaliseert het laden van processen in het geheugen, vergelijkbaar met hoe een filesysteem werkt, maar dan voor processen in RAM.

### Opdeling in Pages en Frames:

Processen worden in "pages" verdeeld, gelijk aan "frames" in het RAM.

Frames hebben een uniforme grootte, vaak een macht van 2 (bijv. 1KiB, 4KiB).

### Toewijzing bij Opstarten:

Bij opstarten krijgt een programma's pages toegewezen aan vrije RAM-frames.
Een page table houdt bij welke frames aan welke pagina's zijn toegewezen.

### Logische naar Fysieke Adresvertaling:

Logische adressen hebben page- en offsetwaarden.

Hardware gebruikt de page table voor het bijbehorende frame.Fysieke adres = Startadres van de page + Offset.

### Speciale Hardware en Geheugentoegangen:

Logisch adres heeft page en offset.
Hardware zoekt frame in de page table.
Twee geheugentoegangen: één voor page table, één voor het geheugen.

### Flexibiliteit en Grootte bepaling:

Pages hoeven niet opeenvolgend te zijn voor flexibiliteit.

Grootte van page/frame wordt ingesteld bij systeemopzet.

### Eenvoudige Adresvertaling:

Vertaling logisch naar fysiek adres is eenvoudiger dan segmentation omdat paginagrootte een macht van 2 is.

## Virtueel geheugen

een geheugenbeheertechniek waarbij niet gebruikte delen van programma's tijdelijk naar de harde schijf worden verplaatst, waardoor efficiënter gebruik van RAM mogelijk is voor actieve processen.

### Swappen:

Niet-gebruikte pages gaan naar swap space op de HDD en worden bij behoefte ingeswapt.

### Swap Locatie:

In een HDD-file of aparte partitie.

### Doel Virtueel Geheugen:

Laat meer programma's toe dan RAM groot is.

### Adresvertaling en Page Fault:

Logische naar fysieke adressen; "page fault" bij ontbrekende page.

### Swapping Procedure:

Frames naar disk schrijven, benodigde page inladen, en proces hervatten.

### Design Overwegingen:

- Minimaliseer swapping, voorkom thrashing.
- Grootte van pages, vergrendeling van frames.
- Keuzes over welke pagina's behouden en geladen moeten worden, gebaseerd op gebruikspatronen.Voorkom thrashing: OS mag niet meer tijd besteden aan swappen dan aan processen uitvoeren.

# Procesbeheer

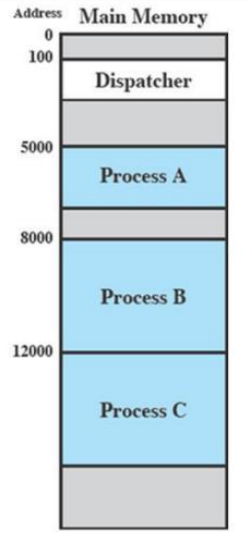
## Wat is een proces

**Een proces is een uitvoerbare eenheid bestaande uit:**

Code | Data en stack | Toestand (context)

Waarom multi-processing?

- Verminderen van wachttijd op resources.
- Tegelijkertijd werken met verschillende applicaties.
- Ondersteuning voor multi-useromgevingen.




**Sporen (traces)**

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

Een dispatcher, ook wel 'scheduler', beheert processen in het geheugen. Het is een integraal onderdeel van het besturingssysteem dat verantwoordelijk is voor het toewijzen van de processor aan verschillende processen.

**Traces gezien vanuit de processen**

| (a) Trace of Process A | (b) Trace of Process B | (c) Trace of Process C |
|---|---|---|
| 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 |

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

# Toestanden van een process

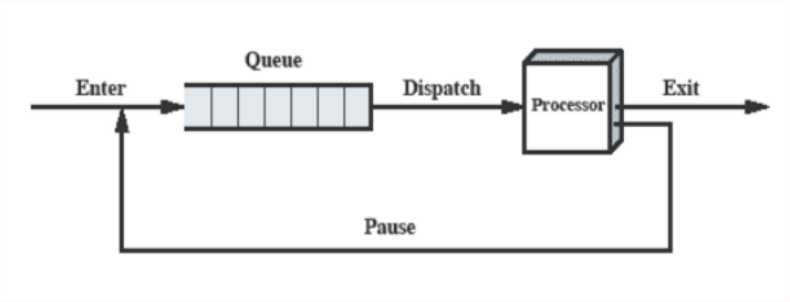
Een proces kan **minstens 2** toestanden hebben:

Running: Actief uitgevoerd.

Not-running: Inactief, wachtend op hervatting.

Context wordt bewaard in een **Process Control Block** (PCB):

- Bevat informatie zoals instructiegeschiedenis, registerwaarden, geheugenconfiguratie, proces-ID, gebruikte resources, starttijd, en meer.
- Het besturingssysteem (OS) onderhoudt een queue van PCB's.



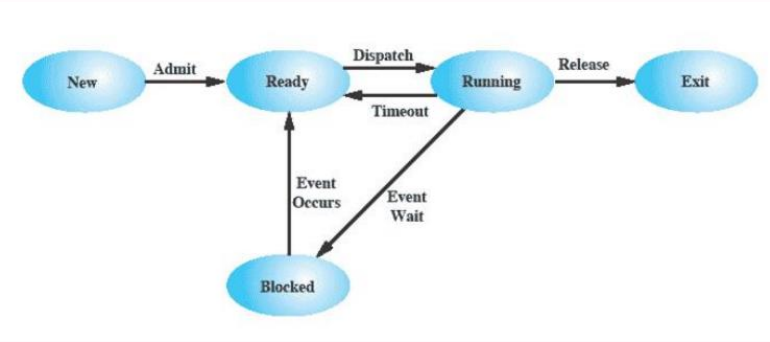

Uitbreiding naar 3 toestanden:

- Ready: Klaar om uitgevoerd te worden.
- Running
- Blocked: Geblokkeerd, bijvoorbeeld tijdens het wachten op I/O.

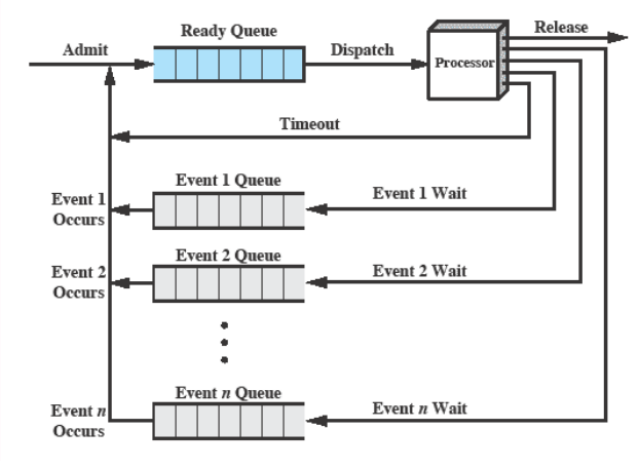
Optioneel, sommige OS-en voegen nog 2 toestanden toe:

New: Net gecreëerd.

Exit: Beëindigd.




Dit systeem biedt eenvoudige, geordende informatie over de diverse toestanden van processen in een besturingssysteem.



# Scheduling in de praktijk

**Tools voor scheduling** in verschillende besturingssystemen:

Linux en Mac OS-X:

- xload
- top
- ps
- pstree

Windows:

- Taakbeheer
- PowerShell: Get-Process

**Idle Time:**

- Alle processen wachten op I/O:
  - Batchsysteem laadt nieuw proces in geheugen.
  - Idle-process (oneindige lus, 'halt' instructie, berekening).

**Context Switch:**

**Non Pre-emptive:**

OS roept procedure aan (call)

Applicatie doet return of blokkeert zichzelf

Eenvoudig

Voorbeelden: Windows 3.11, Mac OS9, Oberon.

**Pre-emptive:**

OS onderbreekt proces om ander proces te starten

- OS programmeert timer in interrupt-controller
- OS start proces met jmp
- Interrupt controller stuurt interrupt
- OS doet context-switch

Nieuw Proces Starten in Unix:

- fork(): Huidig proces wordt gekopieerd, kindproces start in gekopieerde versie.
- exec("executable"): Vervangt code segment, reset stack en data segment.

Nieuw Proces in Shell-Scripts:

Voorbeeld met xload & in bash: fork om child proces te maken, exec om xload te laden in het child proces.

# Scheduling strategieën

### Doel:

- Eerlijk verdelen van processortijd over processen
- 'uithongering' beletten
- Weinig overhead veroorzaken
- Prioriteiten van processen respecteren
- Eventueel: real-time constraints naleven

### Procesattributen in PCB:

w: Tijd in het systeem gewacht

e: Tijd besteed aan uitvoering

s: Geschatte totale uitvoeringstijd

### FCFS (First Come First Served)

Kies het oudste proces (maximale w).

Non pre-emptive.

Voordeel: Lange processen worden snel doorlopen.

Nadeel: Korte processen kunnen lang wachten.

### Shortest Process Next (SPN)

Kies het proces met de kleinste s

Non pre-emptive.

Voordeel: snellere response-tijd
Nadeel:

- Je moet weten hoe lang een proces nodig heeft
- Mogelijkheid tot starvation (uithongering)

### Shortest remaining Time (SRT)

Kies process met kleinste (s-e)

Pre-emptive: nieuw proces met kleinere (s-e) → huidige proces onderbroken

Nadeel:

- Je moet weten hoe lang een proces nodig heeft
- Mogelijkheid tot starvation (uithongering)

### Highest Response Ratio Next (HRRN)

Zoek proces hoogste RR

Non pre-emptive

Response Ratio = (w+s)/s
(initieel gelijk aan 1)

Voordeel: korte processen worden snel afgewerkt

Nadeel: s moet nog steeds geschat worden

### Round-Robin

Meest voorkomend in moderne OS
Pre-emptive: ieder proces krijgt een gelijke time-slice

Voordeel: goede response-tijd, eerlijke behandeling
Nadeel: minder tijd voor i/o-gebonden processen

# IPC & threads

## Interprocess communication (IPC)

### Interprocess Communicatie (IPC)

is de set methoden waarmee processen op een computer gegevens delen of coördineren, waardoor samenwerking tussen applicaties mogelijk is.

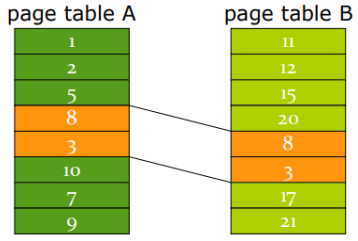
Processen hebben apart geheugen.

Noodzaak tot communicatie, bijvoorbeeld copy/paste en parallelle verwerking.

In Unix communicatie via pipes, berichten, gedeeld geheugen. Windows gebruikt pipes, clipboard en bestanden.

### Unix shared memory

Shared memory wordt geïmplementeerd adhv paging




## Threads

Lichtgewicht processen.

1 proces kan verschillende threads hebben

Threads delen code-segment en data-segment

Threads hebben individuele stacks, wat contextschakeling vereenvoudigt.

### Threads

zijn lichte processen binnen een enkel proces, delen code- en data-segmenten, hebben een eigen stack en maken context-switches efficiënter.

Problemen met Processen:

- Het opstarten van processen duurt lang.
- Context-switches hebben een vertraging.
- Sommige applicaties vereisen meerdere processen met gedeelde data, wat IPC-complicaties oplevert.

Oplossing:

→ Threads bieden een lichtgewicht alternatief voor processen, delen efficiënt geheugen en maken snellere context-switches mogelijk. Ze vereenvoudigen communicatie tussen taken in een programma.

### User-level threading (N:1):

N threads delen 1 OS scheduling entiteit.

Sneller, geen kernel-mode switch.

### Kernel-level threading (1:1):

1 thread per 1 OS scheduling entiteit.

Transparantie, trager vanwege kernel-mode switch.

### Hybrid threading (M:N):

M threads delen N OS scheduling entiteiten.

Combineert voordelen, complexere implementatie.

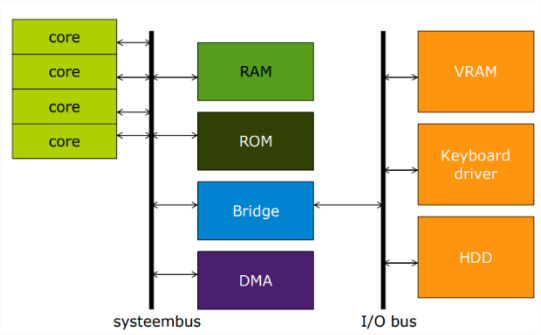
# Multi-processor

### Multi-Processor/Multi-Core:

Meerdere cores/processoren, vaak gedeelde geheugenstructuur.

- Gevolgen voor OS:
- 1 ready-queue voor meerdere processoren.
- Load balancing.
- Conflictoplossing bij gedeelde resources.

### Architectuur met Multi-Processors:




### Shared Memory Multi-processor: strategieën

#### Master-Slave:

Eén processor als master, anderen als slave.

Master draagt 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 draaien op elke processor.

Elke processor heeft eigen scheduling.

Vereist ingrijpende kernelwijzigingen voor synchronisatie.
Processoren kunnen dezelfde code uitvoeren en in hetzelfde geheugendeel lezen/schrijven.

Kernel gebouwd met meerdere processen/threads.

# Gelijktijdigheid

### Gelijktijdigheid:

- Probleem bij meerdere processoren/threads: gelijktijdig aanspreken van resources zoals geheugen.
- Sommige code moet seriëel worden uitgevoerd in een "critical section."
- Noodzaak van lichte vergrendeling (locking) voor deze kritieke secties.
- Zelfs relevant bij één processor.

### Wait() en Signal():

- "Atomic operations" geïmplementeerd en gegarandeerd door het OS.
- Introductie van "Semaforen" voor synchronisatie.

### Semaforen in Linux:

**Sem_wait()**: Wacht als de waarde 0 is, anders verlaagt het de semafoor en gaat door.

**Sem_post()**: Verhoogt de semafoor met 1.

**Sem_init()**: Initialiseert de semafoor.

**Sem_wait()** en **sem_post()**: "atomic operations."

# Dead-lock

## Dead-lock

Een situatie waarin twee of meer processen vastzitten omdat elk wacht op de andere om een bron vrij te geven
Het leidt tot een impasse waarin geen van de betrokken processen verder kan gaan

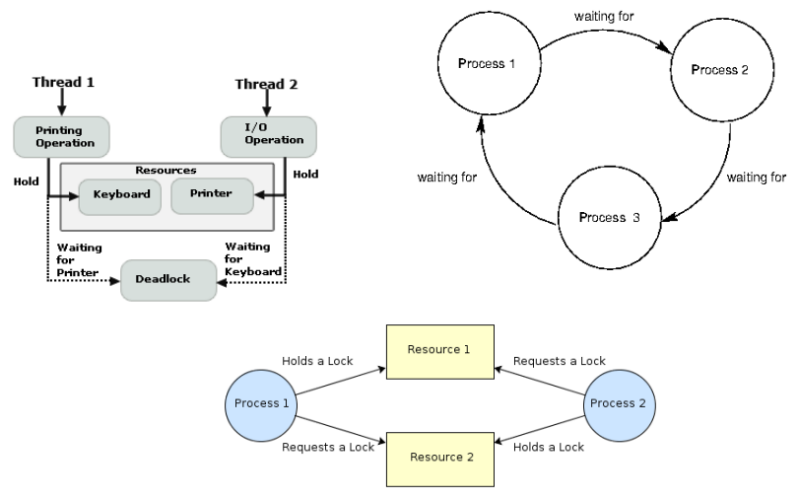
Treedt op wanneer processen of threads wachten op resources die door elkaar worden vastgehouden.

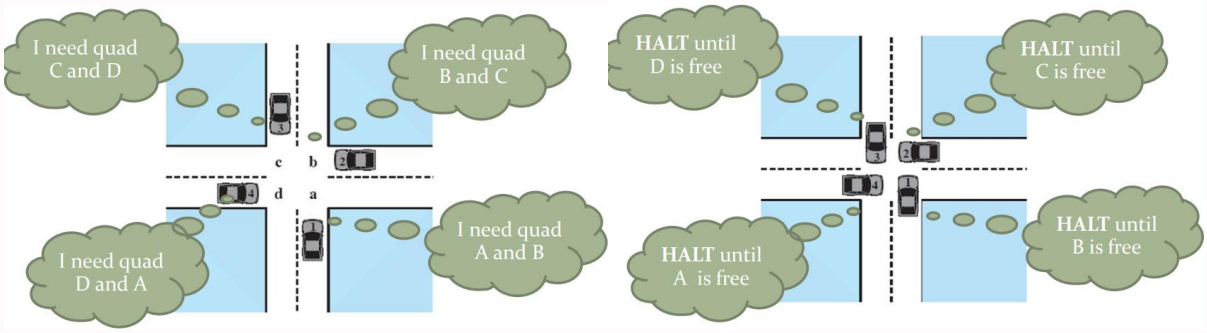
Voorbeeld:

- Processen A en B wachten op bronnen die de ander heeft.
- Processen C en D hebben vergelijkbare afhankelijkheden.
- Geen van de processen kan verdergaan, waardoor een impasse ontstaat.

Voorbeeld met semaforen:

- Twee threads wachten op elkaars semafoor, waardoor geen van beide threads kan doorgaan.



# UI & virtualisatie

## User interfaces

### Terminal services

Text terminal:

- Scherm + toetsenbord.
- Karakter-output, bijvoorbeeld 80x25.
- Verbinding via serial connection (of modem) zoals telnet.
- Veel gebruikt voor communicatie met embedded systemen (router, auto-elektronica, enz.).
- In Unix heeft elke gebruiker een "tty".
- Ieder proces heeft stdin, stdout, stderr.
  - stdin: Input-stream vanuit terminal.
  - stdout: Output-stream naar terminal.
  - stderr: Standaard zoals stdout.

### Windowing systems

Multiple processes, 1 screen

➔ Vereist een windowing system.

Window = virtueel scherm

Windowing system: Softwarelaag tussen applicatie en OS, die de hardware bestuurt.

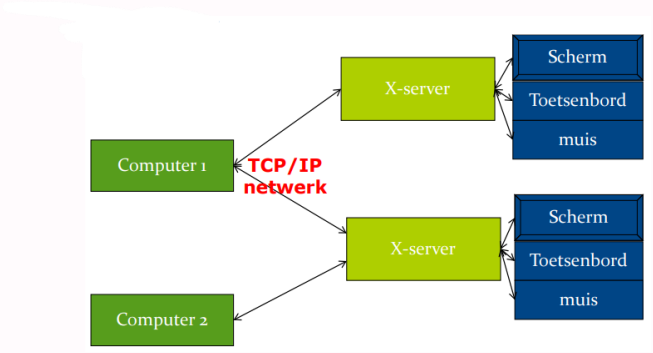
Twee mogelijkheden:

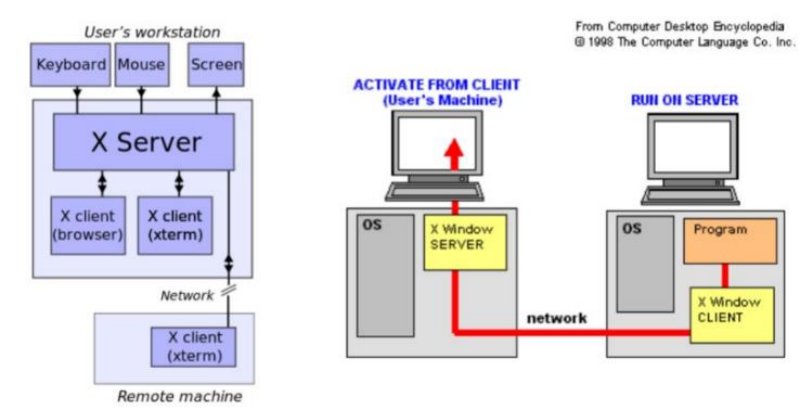
- Ingebouwd in het besturingssysteem (bijvoorbeeld MS Windows).
- Als een apart proces (bijvoorbeeld X-Windowing System).

### X window system

- Bedoeld voor meerdere gebruikers op één systeem.
- X applicaties van verschillende computers op één X-server over het netwerk.

X-terminal:





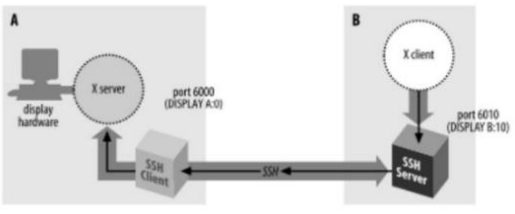

- X, X11 - Ontwikkeld door MIT.
- Elke X-server is een aparte machine of proces.
- Applicaties sturen berichten naar de X-server via TCP/IP.

X-server zorgt voor hiërarchie van windows, tekenen van elementen, en het afhandelen van gebeurtenissen.

Window Manager, Desktop Environment, en Display Manager zijn essentiële componenten.

- Window Manager: Beheert vensters.
- Desktop Environment: Inclusief menu, toolbar, widgets, file browser, etc.
- Display Manager: Beheert inloggen en start scripts.

Voorbeelden van desktop environments: GNOME (C), KDE Plasma (C++).




X Forwarding: Mogelijkheid om een X-applicatie op een andere host te draaien en weer te geven op de lokale X-server.

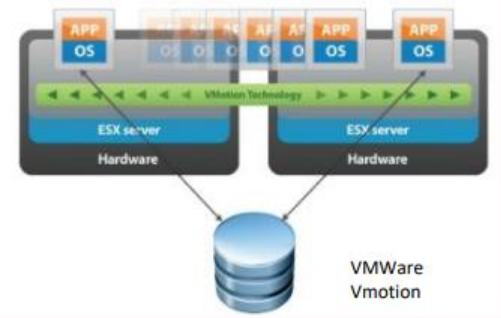
Microsoft Remote Desktop Protocol (RDP):

- Terminal = scherm/toetsenbord/muis, "thin client" via netwerk.
- Oorspronkelijk voor remote assistance, nu ook voor meerdere sessies (gebruikers).

# Virtualisatie

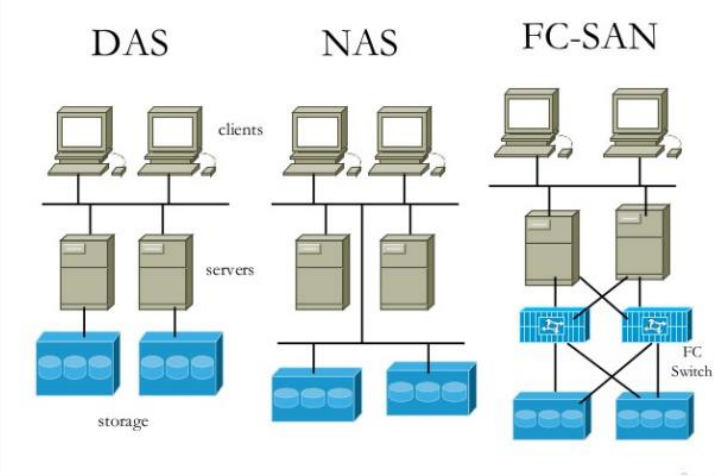
## Waarom virtualisatie

1. Benutting van Hardware:
    - Gemiddeld 5-15% belasting zonder virtualisatie.
    - Kostenbesparing.
2. Deployment:
    - Snellere installatie van nieuwe servers.
    - Self-deployment mogelijk.
3. High Availability en Load Balancing:
    - Pool van virtuele servers op verschillende fysieke servers.
4. Management:
    - Klonen van virtuele machines.
    - Het maken van snapshots voor herstelpunten.
    - Anti-virus op hypervisorniveau.
5. Opslag op NAS of SAN:
    - Deduplicatie voor efficiënter gebruik van opslagruimte.




### NAS - SAN

| | NAS | SAN |
|---|---|---|
| Afkorting staat voor | Network Attached Storage | Storage Area Network |
| Werkt op welk niveau | File-level | Block-level |
| Filesystem staat op | NAS box (server) | Computer (client) |
| Gebruikt welk netwerk | gewone TCP/IP netwerk<br>• SMB (server message block)/CIFS (common Internet filesystem)<br>• NFS (network filesystem) | • Fibre Channel met SAN switches (storage area network)<br>• iSCSI: SCSI in TCP/IP packetten |




DAS: Direct Attached Storage

## Full virtualisatie

Beschrijving:

**Hardware wordt volledig gevirtualiseerd.**

Implementatie:

Type 1 Hypervisor (Bare Metal) of Type 2 Hypervisor (Hosted).

Voorbeeld:

VMware ESX (Bare Metal), VirtualBox (Hosted).

Kenmerken:

- Gast-OS vereist geen aanpassingen.
- Hypervisor emuleert alle hardwareapparaten.
- Type 1 draait rechtstreeks op hardware, terwijl Type 2 op het hostbesturingssysteem draait.

## Paravirtualisation

Beschrijving:

**Aanpassen aan het guest OS zijn nodig**

Implementatie:

Hypervisor deelt de hostkernel met gast-OS.

Voorbeeld:

Xen

Kenmerken:

- Gast-OS vereist aanpassingen aan de OS-kernel
- Minder overhead, sneller zonder vertaling van hardwareabstractie
- Geen aanpassingen aan gebruikersapplicaties

## OS virtualisation

Beschrijving:

**Hostkernel wordt gedeeld door alle gast-OS**

Implementatie:

"Guest containers" zoals Solaris containers, Docker, LXD

Kenmerken:

- Hostkernel gedeeld door alle gasten
- Snelle prestaties zonder vertaling of hardwareabstractie
- Geen aanpassingen aan gebruikersapplicaties

## Cloud computing

### Modellen

**Public Cloud:**

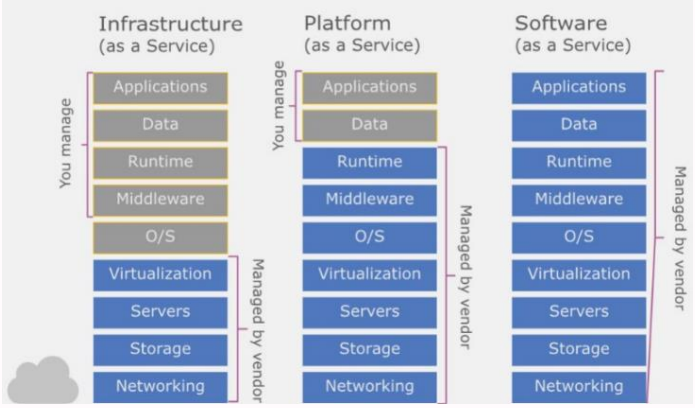
Toegankelijk voor algemeen publiek

**Private Cloud:**

Exclusief voor één organisatie

**Hybrid Cloud:**

Combinatie van publieke en private clouds




### Service modellen

IAAS (Infrastructure as a Service)

PAAS (Platform as a Service)

SAAS (Software as a Service)

### Scaling

Verticale Schaling (Scale Up):

Meer kracht (CPU, RAM) toevoegen aan een bestaande server.

Horizontale Schaling (Scale Out):

Meer servers toevoegen.

Sessiebeheer:

- Vermijd bewaren van sessie-informatie op lokale servers.
- Gebruik een gemeenschappelijke database of in-memory datastore zoals Redis.
- Bij gebruik van lokale servers: activeer Session Affinity op de load balancer om verzoeken van een specifieke client naar dezelfde backend te sturen.

## Blade servers

**Blade servers**

Compacte, modulaire computerservers die in een gemeenschappelijke behuizing, bekend als een blade enclosure of chassis

Blade servers delen een chassis in een datacenter, inclusief voeding, koeling en netwerkconnectiviteit. Ze delen gemeenschappelijke fysieke middelen zoals voeding, koeling, en netwerkcomponenten, wat zorgt voor een compacte en efficiënte serveroplossing.