Een Besturingssysteem is een verzameling samenhangende programma's die een 'beperkt' aantal middelen ter beschikking stelt aan een 'onbeperkt' aantal gebruikers.

<u>Middelen</u>:

-- Hardware:

CVE
werkgeheugen
I/O-apparaten

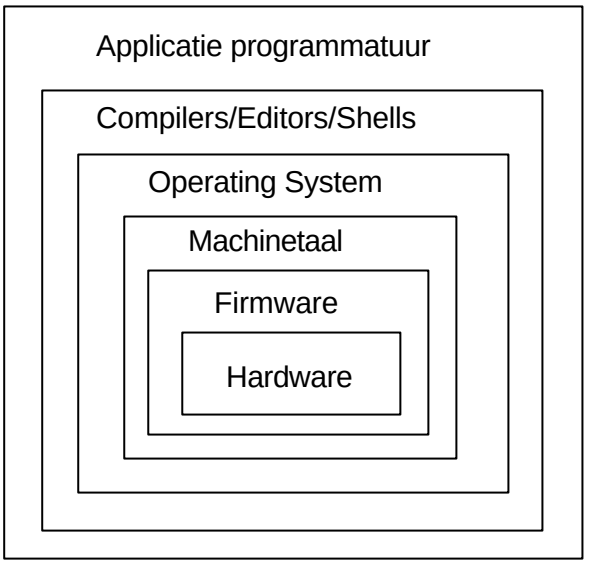
-- Software:

programma's
gegevens

---> BESTURINGSSYSTEEM als

-- MIDDELEN-BEHEERDER

VIRTUELE MACHINES



---> BESTURINGSSYSTEEM als

-- VIRTUELE MACHINE

Soorten besturingssystemen:

Naar gebruik

- **'Single-user, Single-tasking',**
- **'Single-user, Multi-tasking',**
- **'Multi-user, Multi-programmering',**

Naar functie

- **'Batch',**
- **Interactief,**
- **Transactie,**
- **'Real time',**
- **‘Embedded’,**
- **combinaties.**

Systeemstructuur

- **Monolithisch**
- **Gelaagd**
- **Cliënt- Server**

Functies

- **Verwerking van programma's**
- **Allocatie van werkgeheugen**
- **Invoer/ Uitvoer operaties**
- **Allocatie van apparatuur**
- **Manipulatie van bestanden**
- **Fout- detectie/ afhandeling**
- **Bescherming, Beveiliging**
- **“Accounting”**

Kenmerken

- **Parallellisme**
- **Gemeenschappelijk gebruik**
- **Opslag op lange termijn**
- **Non-determinisme**

GESCHIEDENIS

1e GENERATIE (1945 - 1955):

- elektronenbuizen
- 'single user'

2e GENERATIE (1955 - 1965):

- transistoren
- 'batch'- systemen
- 'time- sharing'

3e GENERATIE (1965 - 1980):

- geïntegreerde circuits
- multiprogrammering
  - 'batch'
  - 'time- sharing'
- "alleskunners", concentratie
- IBM S/360 / 370
- virtueel geheugen
- “Unbundling”

4e GENERATIE (1980 - 1990):

- chips
- persoonlijke computers
- gebruikersvriendelijk
- gedistribueerde systemen

COMPONENTEN en hun functie

Proces- beheer:

-- **afhandeling van gebeurtenissen.**
-- **realiseert interprocescommunicatie.**
-- **realiseert tijdsverdeling van de CVE ('short term scheduling').**
-- **realiseert het creëren van processen ('long term scheduling').**

Geheugen- beheer:

-- **realiseert de verdeling van het werkgeheugen over de processen.**

I/O- beheer:

-- **realiseert de fysieke I/O dmv 'device drivers'.**

Bestands- beheer:

-- **vertaling van logische opdrachten naar fysieke opdrachten.**
-- **alloceert een hulpmiddel aan een proces na toetsing van rechten.**
-- **realiseert verschillende typen bestandsorganisatie.**
-- **realiseert de administratie van bestanden en andere I/O- middelen.**

Diversen:

-- **klok- beheer.**

Enkele begrippen:

-Programma:

een formele beschrijving van een algoritme of een combinatie van algoritmes die door een computer geïnterpreteerd kan worden.

-Proces:

de (sequentiële) verwerking van programma's.

-'Thread':

“lichtgewicht” proces.

-'Job':

een werkeenheid voor een computer.

-Taak:

een proces(onderdeel) dat een bepaalde taak uitvoert, onder verantwoordelijkheid van 1 gebruiker.

-'single-user':

gebruiker beschikt over het gehele computersysteem.

-'multi-user':

meerdere gebruikers tegelijkertijd per computersysteem.

-multiprogrammering:

meerdere processen worden (schijnbaar) simultaan verwerkt.

-'multitasking':

multiprogrammering voor 1 gebruiker.

-'multiprocessing':

meerdere processoren (CVE's).

Een (OPERATING) SYSTEM in actie is een verzameling samenwerkende sequentiele processen, concurrent en asynchroon executerend, en gemeenschappelijke datastructuren manipulerend.

Asynchroon betekent dat de onderlinge verwerkingssnelheden non-deterministisch zijn.

Basisprobleem:

Wederzijdse uitsluiting

Voorbeeld:

```
program regelverwerker;
integer regelsin;

procedure proceseen;
begin
     herhaal eindeloos
          lees_een_regel;

          regelsin := regelsin +
1;

          verwerk_regel;
end;

procedure procestwee;
begin
     herhaal eindeloos
          lees_een_regel;

          regelsin := regelsin +
1;

          verwerk_regel;
end;

begin
     regelsin := 0;
     parbegin
          proceseen;
          procestwee;
     parend;
end.
```

Vraag: Welk probleem doet zich hier voor?

Een oplossing:

```
program regelverwerker;
integer regelsin;

procedure proceseen;
begin
    herhaal eindeloos
        lees_een_regel;

        StartWedUit;
        regelsin := regelsin + 1;
        StopWedUit;

        verwerk_regel;
end;

procedure procestwee;
begin
    herhaal eindeloos
        lees_een_regel;

        StartWedUit;
        regelsin := regelsin + 1;
        StopWedUit;

        verwerk_regel;
end;

begin
    regelsin := 0;
    parbegin
        proceseen;
        procestwee;
    parend;
end.
```

Dus: Processen moeten elkaar wederzijds uitsluiten als ze bijvoorbeeld gemeenschappelijke datastructuren manipuleren, in hetzelfde bestand schrijven, enz.

Noodzakelijk:

executie in Kritieke Sectie

**Een Kritieke Sectie is
een verzameling instructies
vooraf gegaan door een StartWedUit
en afgesloten door een StopWedUit.**

StartWedUit en StopWedUit moeten ondeelbare acties zijn !

De opdracht wordt dus:

Zoek oplossingen voor StartWedUit en StopWedUit die:

- **in software te realiseren zijn,**
- **onafhankelijk zijn van de onderlinge snelheden van processen,**

en daarbij:

- **wederzijdse uitsluiting garanderen,**
- **dodelijke omarming uitsluiten,**
- **uithongering uitsluiten,**
- **slechts zwakke samenhang toestaan.**

Opmerkingen

Nederlands	Engels
Wederzijdse uitsluiting:	Mutual exclusion
WedUit:	Mutex
Dodelijke omarming:	Deadlock
Uithongering:	Starvation
	Lockout
Zwakke samenhang:	Loose connection
Zwak samenhangend:	Loosely coupled

We beperken ons tot twee processen.
We formuleren de oplossingen mbv de concurrente statements.

Algemene structuur:

```
program probeer;

globale datastructuren;
declaratie proceseen;
declaratie procestwee;

begin
    parbegin
        proceseen;
        procestwee;
    parend;
end.
```

Beschrijving van de Semafoor

Een Semafoor is een object, dwz een datastructuur en enige operaties op die datastructuur.

Datastructuur
Struct sema { Int waarde; List of process L;}; Typedef struct sema Semafoor; Semafoor S;

Operaties	met hun functionaliteit
Wait(S)	S.waarde := S.waarde - 1; Als S.waarde < 0 Dan zet dit proces in S.L en blokkeer;
Signal(S)	S.waarde := S.waarde + 1; Als S.waarde <= 0 Dan haal proces uit S.L en maak ready;
sinit(S,i)	Als i < 0 Dan fout; S.waarde := i; initialiseer L;

NB: De organisatie van de lijst is niet vastgelegd.

NB: Aliassen voor Wait zijn Down (Tanenbaum) en de oorspronkelijke naam P ("passeren").
Aliassen voor Signal zijn Up (Tanenbaum) en de oorspronkelijke naam V ("vrijgeven v h traject").

Gebruiksmogelijkheden

- Mutex voor kritieke secties

- Synchronisatie van processen

Vb van een toepassing van een semafoor tbv mutex: Een concurrent programma waarin 5 processen 3 afdrukkers moeten delen. De uitvoer van de verschillende processen moet gescheiden blijven.

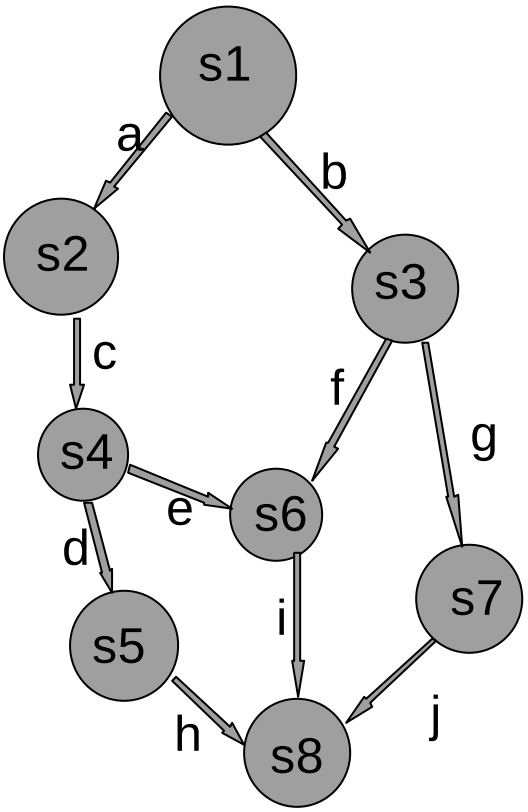
```
Program Afdruk_processen;
Semafoor Print;

Procedure P_1;
    .
    .
Procedure P_i;
Begin
  Herhaal
    doe_iets i;
    Wait(Print);
    bepaal welke afdrukker;
    doe het afdrukwerk;
    markeer afdrukker vrij;
    Signal(Print);
    rest i;
  Tot klaar;
End;
    .
    .
Procedure P_5;

Begin
  sinit(Print, 3);
  Parbegin
    P_1; P_2; P_3; P_4; P_5;
  Parend;
End.
```

Vb van het gebruik van een Semafoor ter synchronisatie van processen:
Proces_2 mag actie S2 pas uitvoeren als Proces_1 actie S1 voltooid heeft.
De Semafoor Sync wordt geinitialiseerd op 0.

```
Proces_1          Proces_2
.                      .
.                      .
S1                     P(Sync);
V(Sync);          S2
.                      .
.                      .
```

Nog een voorbeeld:

Een precedentiegraaf

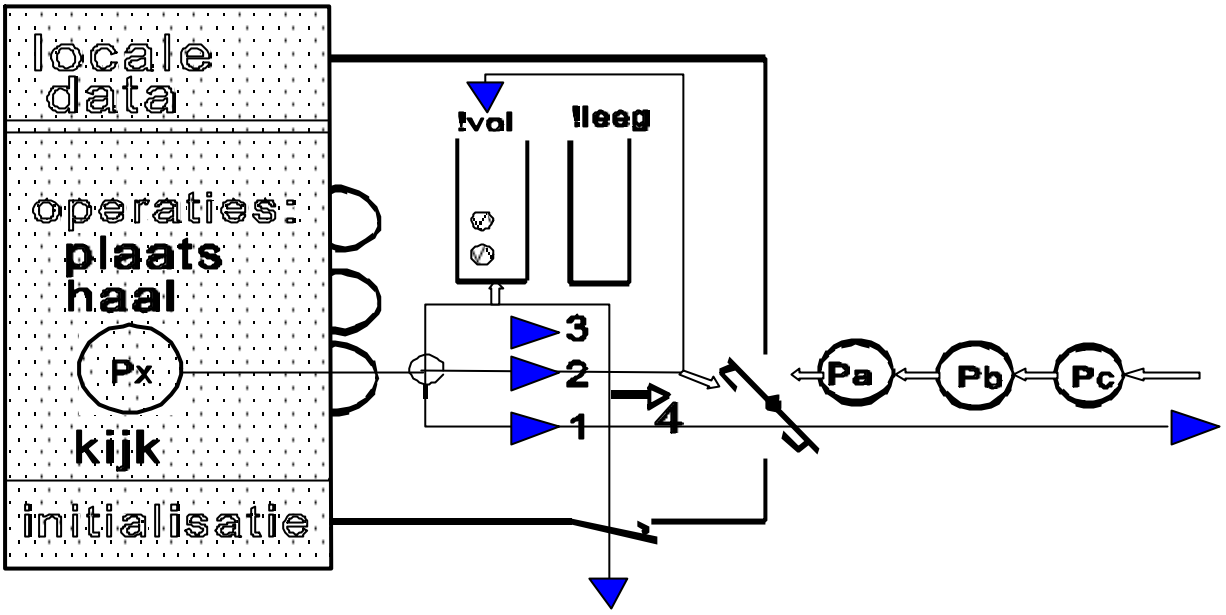


Het concurrent programma

```
Semafoor a,b,c,d,e,f,g,h,i,j;
      alle geinitialiseerd op 0

Parbegin
  Begin
      s1;V(a);V(b);
  end;
  Begin
      P(a);s2;V(c);
  end;
  Begin
      V(c);s4;V(d);V(e);
  end;
  Begin
      P(b);s3;V(f);V(g);
  end;
  Begin
      P(d);s5;V(h);
  end;
  Begin
      P(e);P(f);s6;V(i);
  end;
  Begin
      P(g);s7;V(j);
  end;
  Begin
      P(h);P(i);P(j);s8;
  end;
Parend;
```

```
Optimaal?:  Parbegin
             Begin s1;vb;s2;s4;ve;s5;pi;pj;s8 end;
             Begin pb;s3;vf;s7;vj end;
             Begin pe;pf;s6;vi end;
            Parend;
```

MONITOR met 2 condities



1 Px heeft een Kijk uitgevoerd en vertrekt door de voordeur;

2 Px probeert een Plaats uit te voeren maar de buffer is vol, dus Px gaat in de wachtrij voor conditie !vol na de voordeur geopend te hebben;

3 Px heeft een Haal uitgevoerd, laat een proces toe dat wacht op !vol en vertrekt door de zijdeur;

(4 Als bij 3 de wachtrij leeg blijkt te zijn, dan vertrekt Px door de voordeur.)

. **Mutex**

. **Synchronisatie**

- Communicatie

via:

. **gemeenschappelijk geheugen**

. **boodschappen**

system calls:

- **Send**
- **Receive**

Multiprogrammering en architectuur

Een consequentie van multiprogrammering:

-- Bescherming mbt

**** programma-fouten,**

**** gebruik I/U-apparatuur,**

**** gebruik werkgeheugen,**

**** monopolisering van de CVE.**

Remedies:

- 'trap'(interrupt)-mechanisme.

- CVE-toestand ('systeem', 'applicatie').

- 'SPOOL'en van I/O naar niet-simultane apparatuur.

- geheugentechnieken zoals:
grensregisters,
paginering,
segmentering.

- CPU-wekker('timer') en klokken.

Multiprogrammering en processen

Multiprogrammering:

meerdere processen worden (schijnbaar) parallel verwerkt.

--> (schijnbaar) parallel = concurrent

CVE-wisseling: nav gebeurtenissen
(interrupts en traps)

dus: processen worden op willekeurige tijstippen onderbroken.

--> proces-context
context-'switch'

Soorten interrupts en traps:

- **system call**
- **I/O**
- **'timer'**
- **programma-fout**

Schets van de kern

- **Afhandeling van interrupts en traps**
- **Proceswisseling op de cpu (dispatcher en context switch)**
- **Gereedschap tbv Interprocescommunicatie**

Representatie van een proces:

PROCESTABEL :

{ proces-administratie }

PROCES :

Grootheid waarvoor een PCB bestaat in de Proces-Tabel

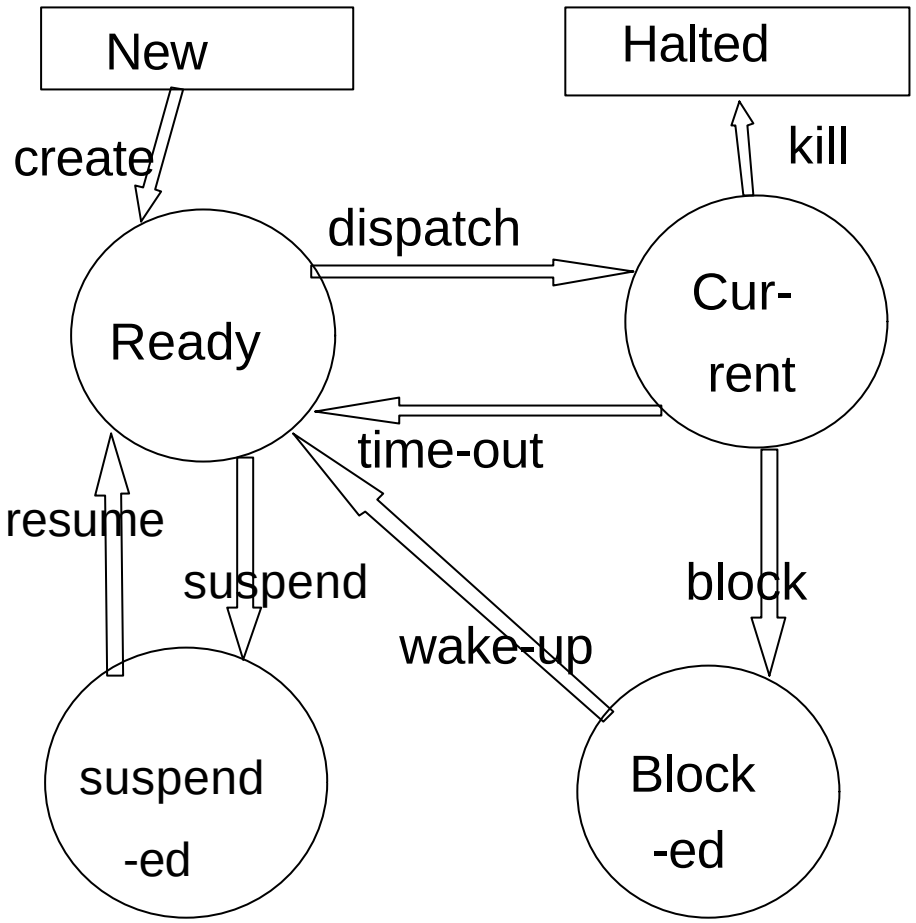
Context:

- **PCB**
  - **administratie- gegevens**
- **geheugenbeslag ('core-image')**
  - **instructies**
  - **data**
  - **(stapel)**
- **registers**
  - **instructie- teller**
  - **(stapel- wijzer)**
  - **andere registers**

<u>Proces-toestanden</u>:

'Current':	**wordt verwerkt**
'Ready':	**gereed voor verwerking**
'Blocked':	**wachtend op gebeurtenis**
'Suspended':	**tijdelijk opgeschort**

Proces-toestand-overgang-diagram



Toestandsovergangen:

'Create': nieuw proces --> ready
'Dispatch': ready proces --> current
'Time-out': current proces --> ready
'Block': current proces --> blocked
'Wake-up': blocked proces --> ready
'Kill': current proces --> halted
'Suspend': ready proces --> suspended
'Resume': suspended proces --> ready

Geheugenbeheer

Primair (werk-, voorgrond-) geheugen

- **Direct refereerbaar door de CPU**
- **Per cel adresseerbaar**
- **Vergankelijk**
- **Snel**
- **Duur**

Secundair (achtergrond-) geheugen

- **Niet direct refereerbaar door de CPU**
- **Blok adresseerbaar**
- **Niet vergankelijk**
- **Langzaam**
- **Goedkoop**

Reëel (fysiek) geheugen:

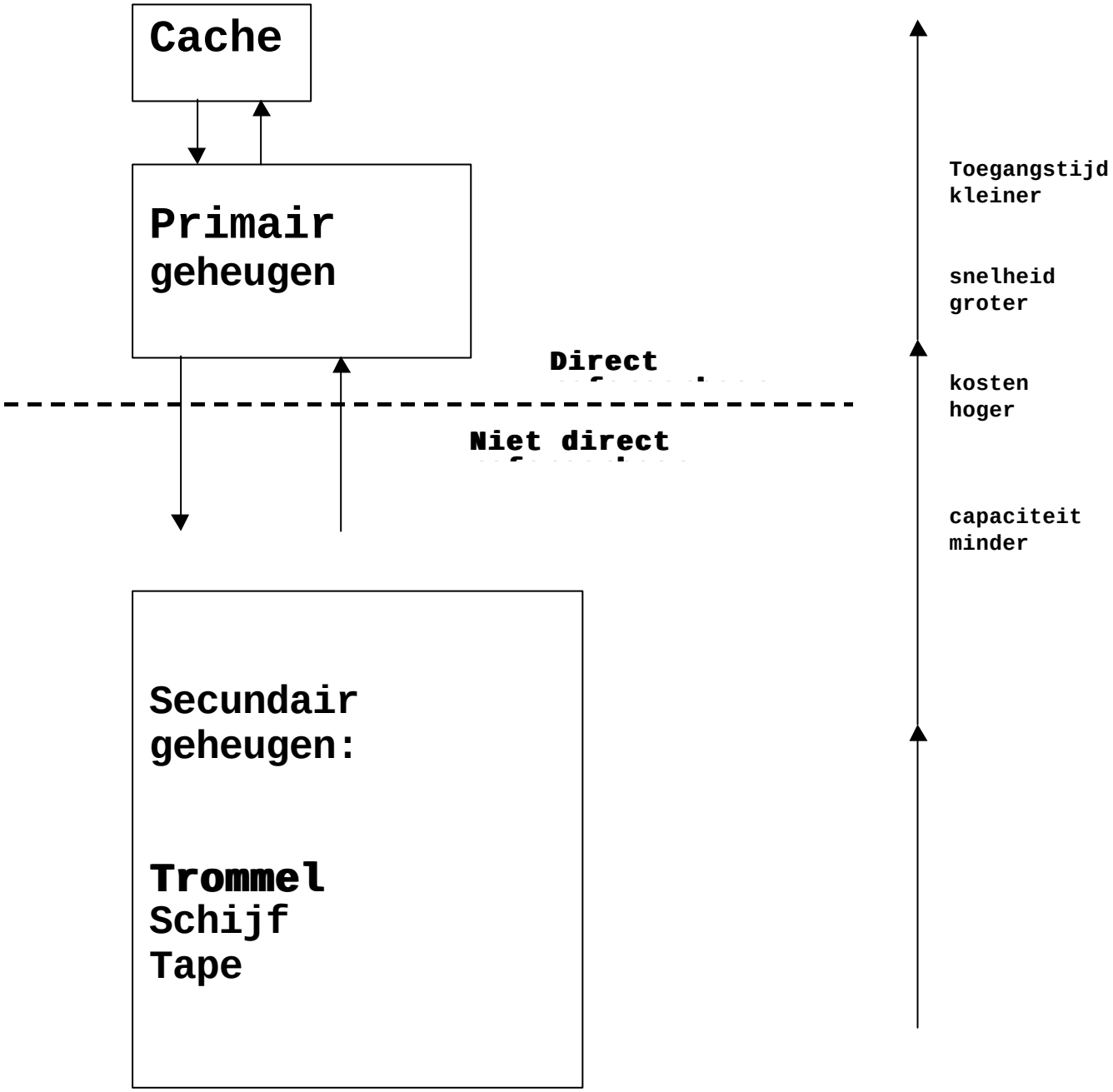
Werkelijk aanwezig (werk)geheugen

Virtueel geheugen:

Schijnbaar aanwezig (werk)geheugen

NB: geheugens zijn van grote invloed geweest op de ontwikkeling van besturingssystemen.

Geheugen- hiërarchie

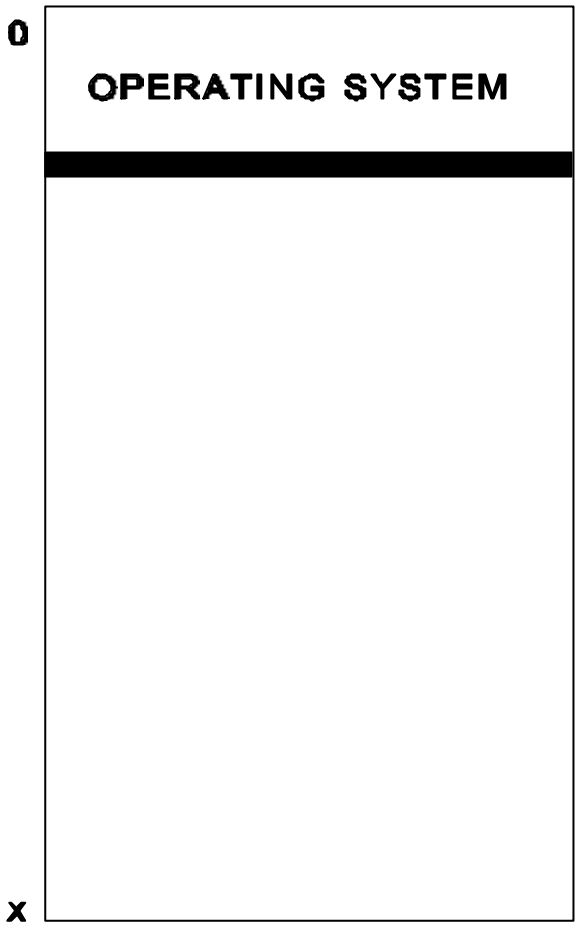


Geheugen- strategieen:

1. Fetch : wanneer naar primair?
2. Placement : waar in primair?
3. Replacement : wat moet plaats maken?

Geen geheugenbeheer

Kenmerk: reëel
gebruiker regelt zelf

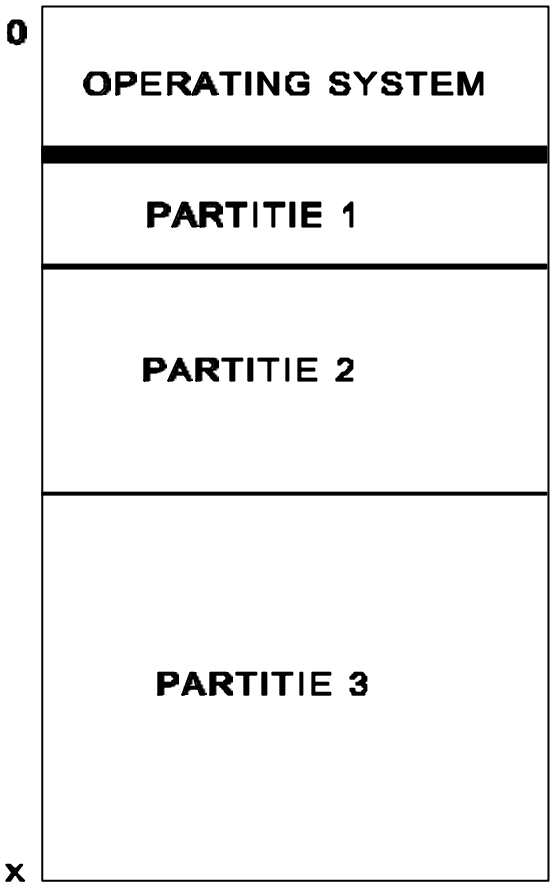


Mogelijkheden:
- **overlay (één gebruiker)**
- **swapping (meer gebruikers)**

Fragmentatie niet relevant

M F T : Multiprogramming with a Fixed number of Tasks (vaste partities)

Kenmerk: reëel
vaste geheugenindeling
(wel instelbaar)

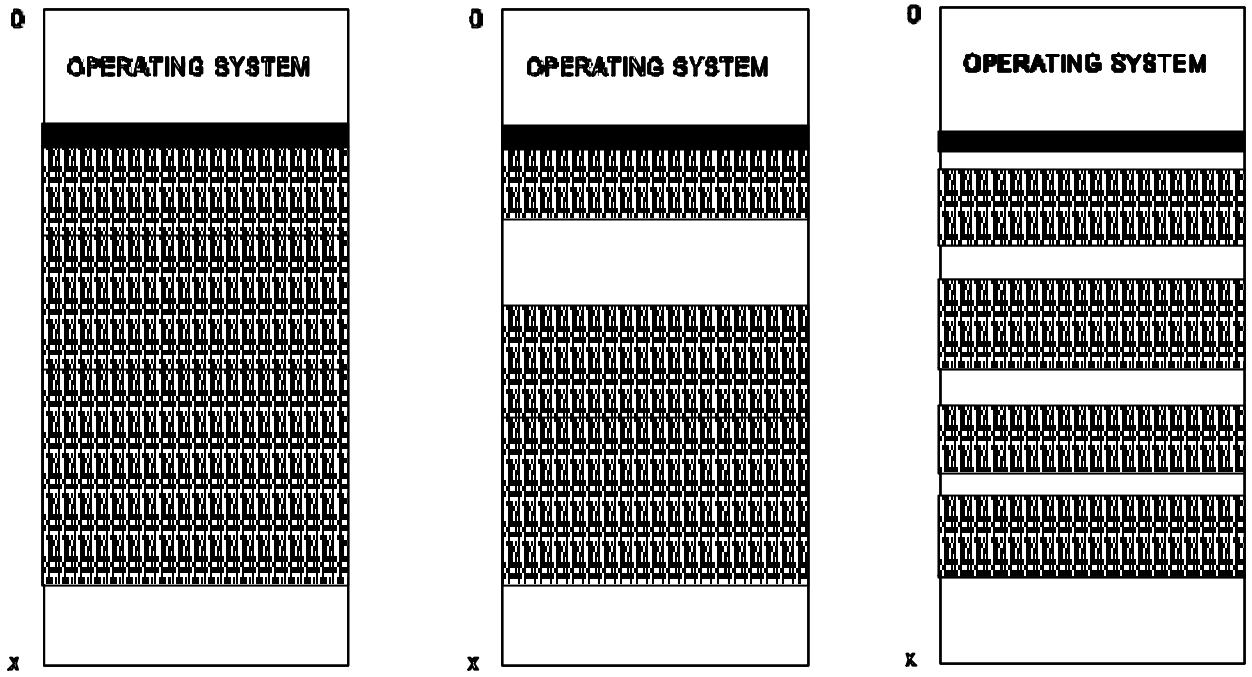


nadelen: niet flexibel
interne fragmentatie
programmagrootte gelimiteerd

overlay: mogelijk
swapping: mogelijk

M V T : Multiprogramming with a Variable number of Tasks (variabele partities)

Kenmerk: reëel



veranderende geheugenindeling

nadelen: gaten
externe fragmentatie
programmagrootte gelimiteerd

overlay: mogelijk
swapping: mogelijk

remedies tegen onbruikbare gaten:
- **samensmelten**
- **optimaal vullen:**
  - **"first fit": eerst passend**
  - **"best fit": best passend**
  - **"worst fit": slechtst passend**
  - **"quick fit": meerdere lijsten**
- **verzamelen (compaction)**

- **Voorafgaande indelingen hebben betrekkelijk eenvoudige gevolgen voor de hardware: een paar registers per programma, elk adres moet gevalideerd worden.**

- **Uitgangspunt is dat programma's als ze verwerkt worden, *volledig* in het primair geheugen aanwezig zijn.**

De nu volgende beheersmethoden

- **Paginering**
- **Segmentering**

maken het mogelijk dat programma's slechts gedeeltelijk aanwezig zijn.

➢ **Geschikt voor "virtuele" technieken**

Virtueel geheugen

Overwegingen:

- **Verwerking is “plaatselijk”**
  - **m.b.t. plaats**
  - **m.b.t. tijd**

- **Elke instructie en elk gegeven is slechts tijdelijk nodig**

- **Laad niet alles tegelijk maar haal slechts dat wat nodig is (“working set”)**

Gevolg: ruimte voor meer processen

Opmerking:
Het moet transparant zijn voor de gebruiker.

Verantwoording

“Plaatselijkheid” (Eng: Locality)

- **Plaats: de kans dat plaatsen in de omgeving van een gerefereerde plaats ook aangesproken zullen worden, is groot.**
  Bijv. Arraybewerkingen.

- **Tijd: de kans dat een aangesproken plaats binnen afzienbare tijd weer gerefereerd zal worden, is groot.**
  Bijv. Lussen.

Weinig gebruikte programmadelen (80-20regel)

Gebruiksgemak:

- **Meer multiprogrammering**
- **Grote adresruimte voor de gebruiker**

Implementatie-mogelijkheden

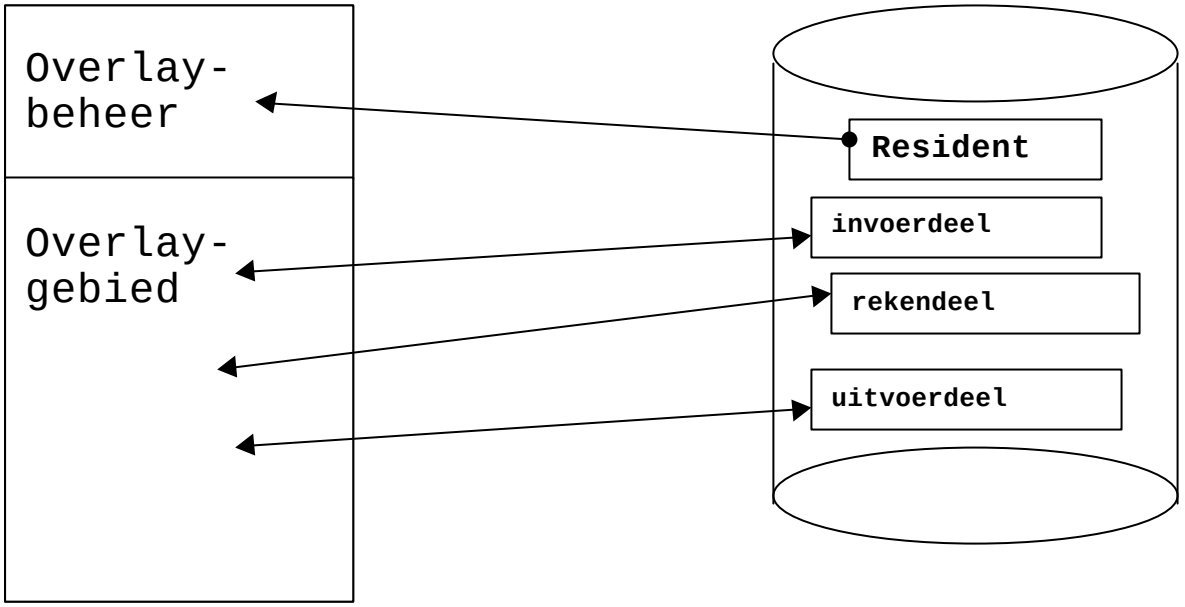
- **Gebruiker doet het zelf: overlays**
- **Systeem regelt het: paginering**
- **Logische indeling door gebruiker, implementatie door systeem: segmentering**

Optimaal combinatie Paginering / segmentering

Beslissingen:

- **Fetch : wanneer?**
- **Placement : waar?**
- **Replacement : wat weg?**

Overlay



Primair van de gebruiker

Secundair

De gebruiker bepaalt zelf de “swapping” binnen het eigen gebied.

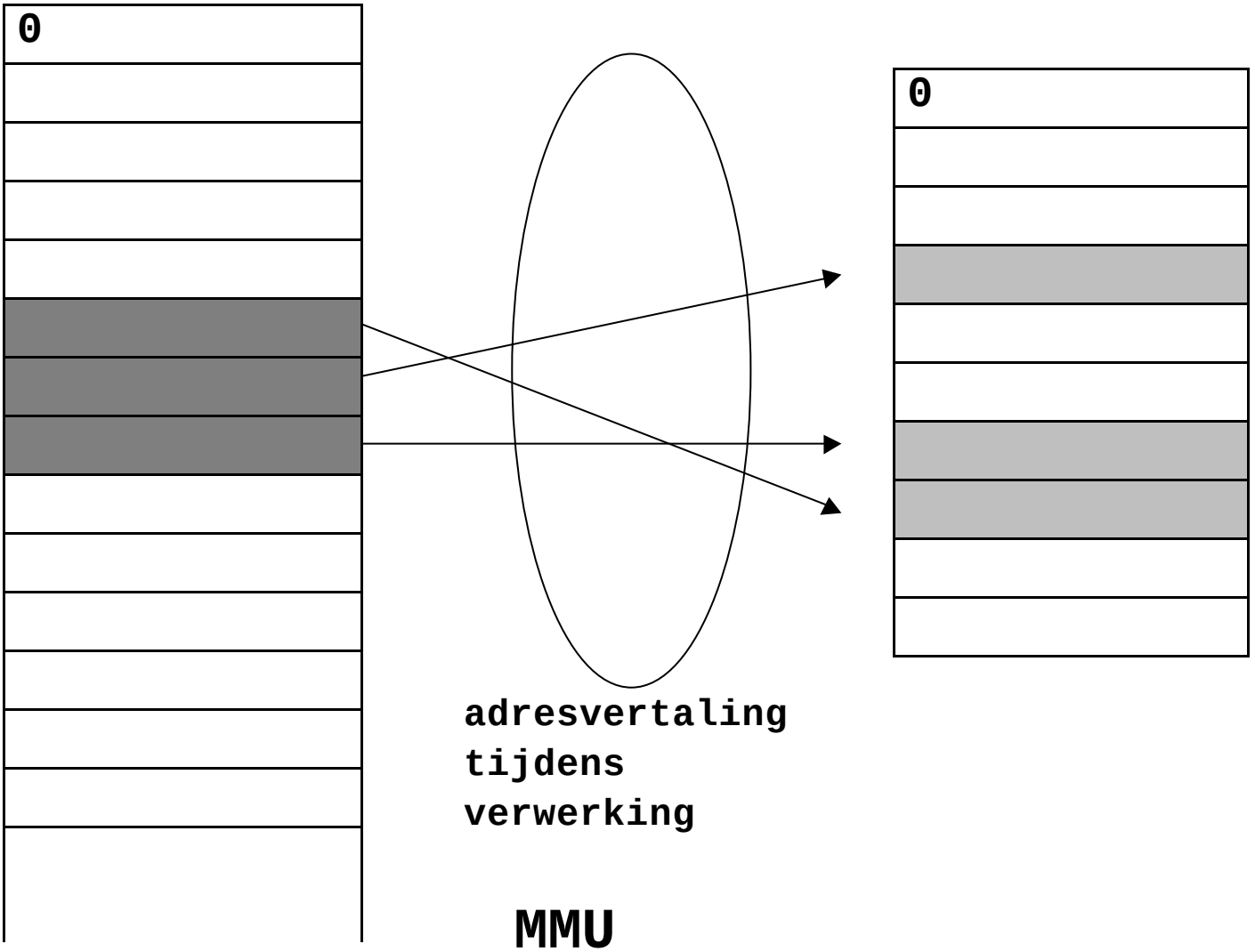
Toepasbaar voor alle reele geheugensystemen.

- **Fetch : gebruiker bepaalt**
- **Placement : gebruiker bepaalt**
- **Replacement : gebruiker bepaalt**

Virtueel

Reeel geheugen: fysiek aanwezig geheugen
Virtueel geheugen: schijnbaar aanwezig geheugen

Koppeling tussen reëel en virtueel



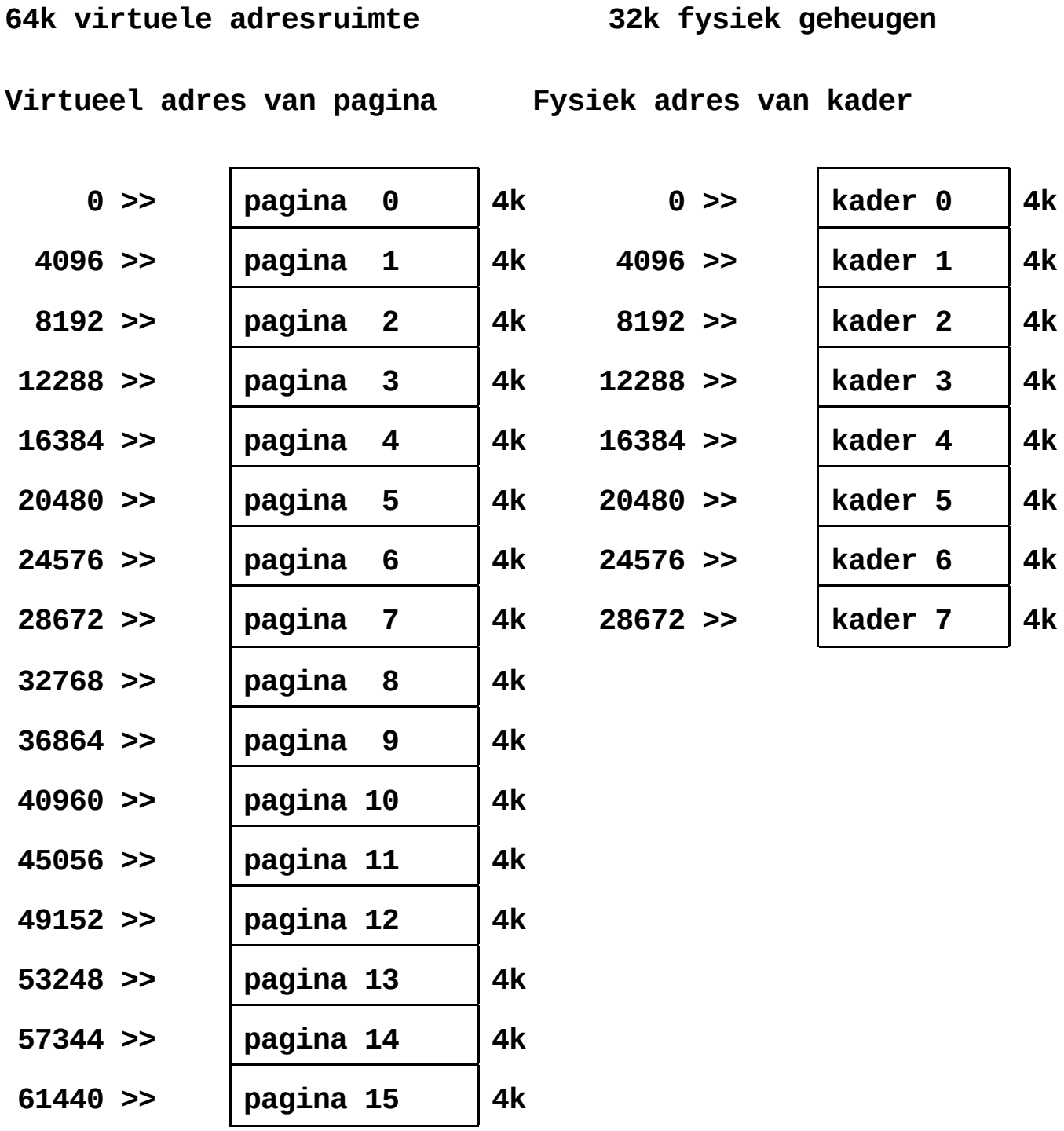
Virtuele adresruimte = Adresruimte van de gebruiker

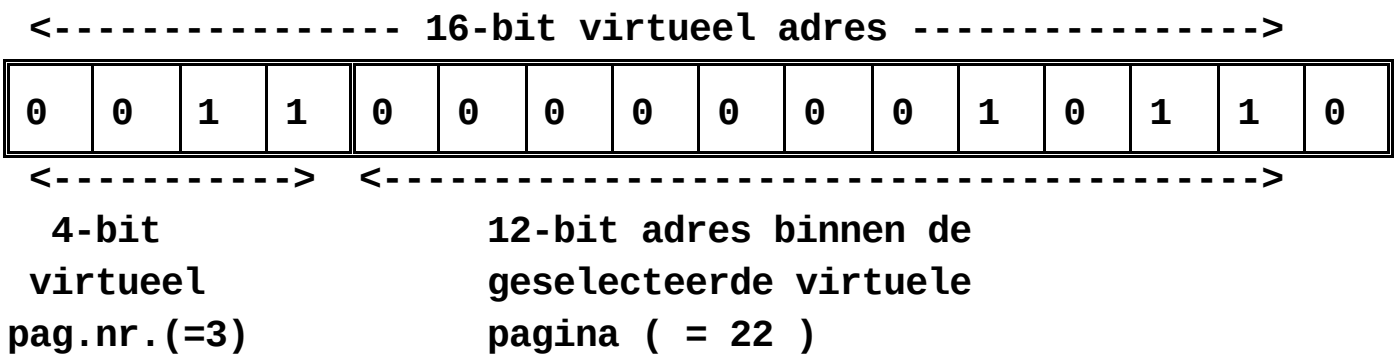
Fysieke adresruimte

Paginering

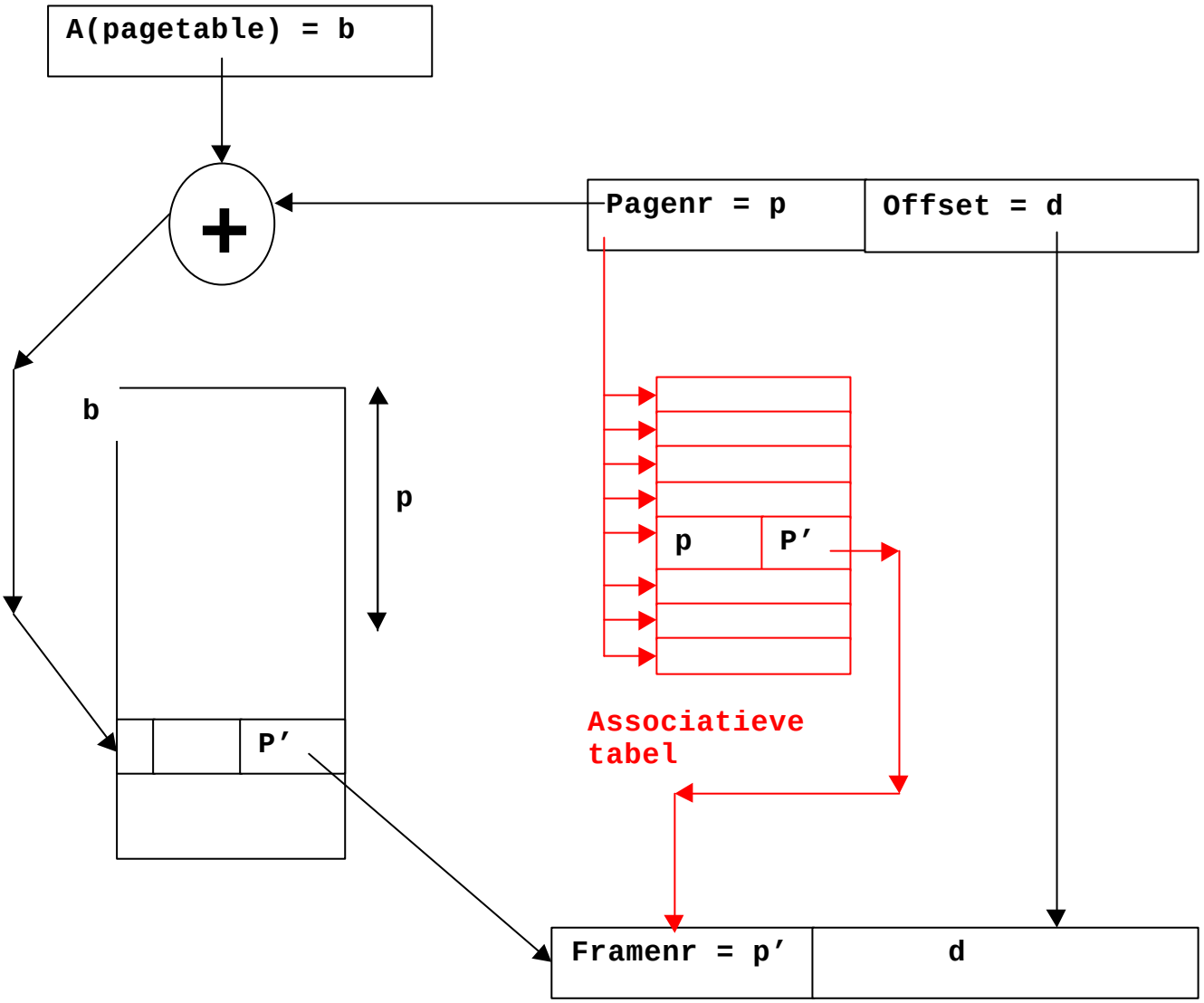
- **De virtuele adresruimte is verdeeld in blokken van vaste lengte (pagina's, pages)**
- **Het fysieke geheugen is verdeeld in blokken van dezelfde lengte (kaders, frames)**
- **Per gebruikersproces is een paginatabel in primair geheugen beschikbaar waarin de toestand van elke pagina vastgelegd is.**
- **Een logisch adres bestaat uit een paginanummer gevolgd door de relatieve positie binnen de pagina (offset)**

Paginering


64k virtuele adresruimte
32k fysiek geheugen
Virtueel adres van pagina
Fysiek adres van kader
0 >> pagina 0 4k
4096 >> pagina 1 4k
8192 >> pagina 2 4k
12288 >> pagina 3 4k
16384 >> pagina 4 4k
20480 >> pagina 5 4k
24576 >> pagina 6 4k
28672 >> pagina 7 4k
32768 >> pagina 8 4k
36864 >> pagina 9 4k
40960 >> pagina 10 4k
45056 >> pagina 11 4k
49152 >> pagina 12 4k
53248 >> pagina 13 4k
57344 >> pagina 14 4k
61440 >> pagina 15 4k
0 >> kader 0 4k
4096 >> kader 1 4k
8192 >> kader 2 4k
12288 >> kader 3 4k
16384 >> kader 4 4k
20480 >> kader 5 4k
24576 >> kader 6 4k
28672 >> kader 7 4k


<---------- 16-bit virtueel adres ---------->
0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0
<---------> <---------->
4-bit virtueel pag.nr.(=3)
12-bit adres binnen de geselecteerde virtuele pagina (= 22)

Adresvertaling bij paginering



Een associatieve tabel van 8 of 16 plaatsen zorgt voor een "hitrate" van meer dan 90%.

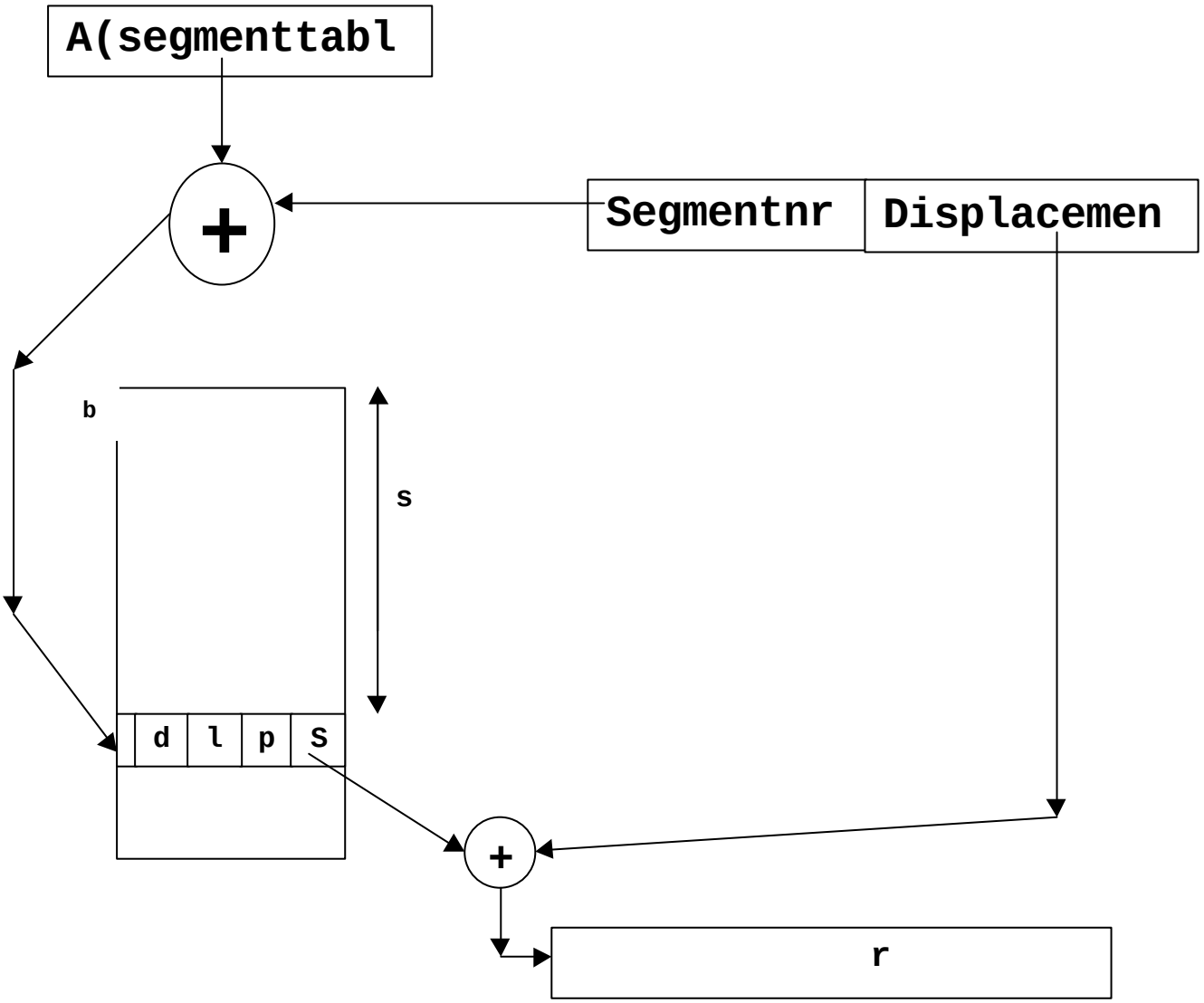
Segmentering

- **De virtuele adresruimte wordt door de gebruiker verdeeld in delen van verschillende grootte . Elk deel wordt een apart objectmodule met als beginadres 0. Elk deel krijgt een nummer. Zo’ n deel heet een segment.**

- **Per gebruikersproces is een segementtabel in primair geheugen beschikbaar waarin de toestand van elk segment vastgelegd is.**

- **Een logisch adres bestaat uit een segmentnummer gevolgd door de relatieve positie binnen het segment (displacement)**

Adresvertaling bij segmentering



Opmerkingen:
d = schijfadres
l = lengte
p = protectie
S = segmentadres
r = reëel adres

- **wordt vergeleken met l**
- **toegangsrecht (p) wordt getoetst (r w e a)**
- **eventueel associatieve tabel**

Aspecten van PAGINERING:

- Verdeling van “fysieke” eenheden
- Externe fragmentatie bestaat niet
- Interne fragmentatie in laatste pagina van het gebruikersprogramma
- Extra geheugen referentie door de pagina tabel
- **Dus:** hardware oplossing nodig
- Gemeenschappelijk gebruik van pagina ’s
- Herbetreedbare (reentrant) code
- Aparte data pagina’s
- Protectie op pagina-niveau
- Keuze: pagina grootte (belangrijk bij virtueel geheugen)

Aspecten van SEGMENTERING

- Verdeling in “logische” eenheden
- Externe fragmentatie als bij MVT
- Interne fragmentatie bestaat niet
- Extra geheugenreferentie voor de segment-tabel
- **Dus:** hardware oplossing nodig
  - Gemeenschappelijk gebruik van segmenten
  - Herbetreedbare (reentrant) code
- Aparte data segmenten
- Protectie op segment niveau

Dus: logisch daarom meer mogelijkheden dan bij paginering

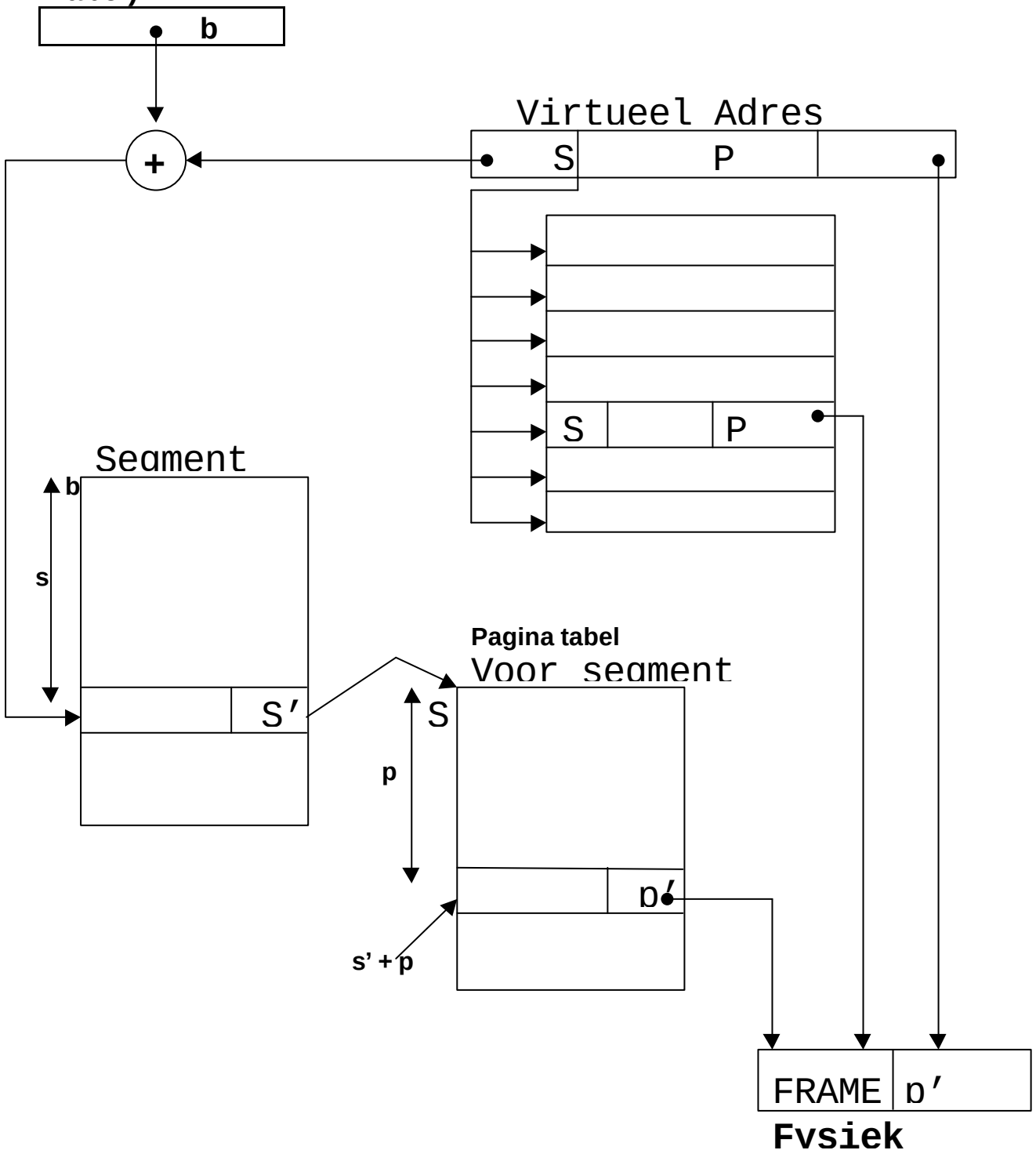
Paginering/Segmentering

Adres:

s	p	d

Adresvertaling:

Regular:
A (Segment Tabel)



protectie op segment-niveau

Vergelijking paginering – segmentering

	paginering	segmentering
Opbouw van de geheugenstructuur	Systeem – “fysiek”	Gebruiker – “logisch”
Bescherming	RWEA onnatuurlijk meestal niet aanwezig	RWEA goed mogelijk op segmentniveau
Efficiënt geheugengebruik	Weinig verlies op interne fragmentatie	Externe fragmentatie zie MVT
“sharing”	Problematisch (partiële pagina’s, uitbreiding)	Simpel
Soort systemen	interactief	Productie

Gepagineerde segmentering combineert de voordelen van paginering en segmentering.

Protectie op segmentniveau.

Echter drie geheugenreferenties zijn nodig dus hardware voorzieningen zijn nodig

PAGINERING:

Uitgangspunt: Een pagina wordt pas geladen als ernaar gerefereerd wordt

DEMAND PAGING

Wat gebeurt er?
Het programma staat in zijn geheel op secundair geheugen
Als het programma verwerkt wordt, is de desbetreffende pagina in primair geheugen aanwezig.
Stel een adres wordt aangesproken in een andere pagina.

1. Het adres wordt gevalideerd
2. Het adres mapping mechanisme bepaalt met behulp van de pagina tabel of de betrokken pagina reeds aanwezig is in het primair geheugen
3. Wel aanwezig: ga voort bij 1

Niet aanwezig: een "pagefault" interupt wordt gegenereerd

4. Het gebruikersproces stopt en de "pagefault" wordt doorgegeven aan de component: "geheugenbeheer".

Acties van het O.S.:

- Zoek een vrij frame (placement)
- Geef opdracht aan de "page driver" om de betrokken pagina van secundaire geheugen te halen
- Als binnen, dan wordt de administratie bijgewerkt
- Geef aan het betrokken proces door dat het weer verder kan

Wat is hiervoor nodig:

- Aangepaste architectuur (hardware)
- Paginatabellen
- Snel achtergrondgeheugen
- Geheugenbeheerroutines
- Speciale I/O-routines

Wat zijn de kosten van een geheugen referentie in een virtueel (demand paging) systeem in vergelijking met een “gewone” geheugen referentie?

dwz. Wat is de effectieve (gemiddelde) toegangstijd?

Stel geheugentoegangstijd = ma
Stel kans op pagefault = p
Stel gemiddelde effectieve toegangstijd = ema

➔ bepaal: ema = (1-p) * ma + p * ”pagefaulttijd”

Acties bij een pagefault:

1. Trap O.S.
2. Bewaar registers en PS
3. Bepaal soort trap (pagefault)
4. Valideer het adres en bepaal de plaats van de pagina op secundair geheugen
5. Bepaal vrij frame *
6. Geef leesopdracht
7. Wacht voor de schijf
8. Seek/latency op schijf
9. Overdracht
10. Interrupt page binnen
11. Bewaar registers en PS van onderbroken proces
12. Bepaal de interrupt
13. Pas pagina administratie aan
14. Wacht op dispatch van het proces
15. Herstel de registers en PS en ga door met de betrokken instructie

*: we nemen aan dat er een vrij frame is.

In hoofdzaken:

1. Behandeling pagefault interrupt
2. Binnen halen van de pagina
3. Herstart acties

Karakterestieke getallen:
1 + 3 : 100 – 1000 μsec

drum swap: 9 msec

schijfswap (bewegende kop): 30 msec

➔ gemiddelde pagefault tijd: 10msec
stel “gewone” geheugen referentie: 1μsec

➔ $\overline{ema}$ = (1 – p) + 1 μsec + P * 10 msec
= 1 + 9999 * P μsec

Als we minder dan 10% degradatie wensen dan moet:
1.10 > 1 + 9999 * P
0.10 > 9999 * P
P < 0.00001

Resumé:

Fetch:
- demand
- anticiperend

Placement: geen probleem bij paging
Replacement: ?

Als een frame geleegd moet worden t.b.v. een nieuwe pagina, moet de oude inhoud bewaard worden. → overhead wegschrijven naar schijf
Daarom: DIRTY bit per frame tegen overbodig wegschrijven

Blijft de vraag:
Welke frame moet geleegd worden?
→ Replacement strategieën

Willekeurige replacement:

- Simplele en snelle strategie
- Weinig efficient (alle pagina's even grote kans)

FIFO:

- Tijdregistratie bij fetch van een pagina
- **`Intuïtief redelijk`**
- Niet goed bij veel "sharing"

De "FIFO tegenstrijdigheid"

Referen**ties**		**Met 3 frames**				**met 4 frames**			
	pf				**pf**				
A	*****	**A**	**-**	**-**	*****	**A**	**-**	**-**	**-**
B	*****	**B**	**A**	**-**	*****	**B**	**A**	**-**	**-**
C	*****	**C**	**B**	**A**	*****	**C**	**B**	**A**	**-**
D	*****	**D**	**C**	**B**	*****	**D**	**C**	**B**	**A**
A	*****	**A**	**D**	**C**	**-**	**D**	**C**	**B**	**A**
B	*****	**B**	**A**	**D**	**-**	**D**	**C**	**B**	**A**
E	*****	**E**	**B**	**A**	*****	**E**	**D**	**C**	**B**
A	**-**	**E**	**B**	**A**	*****	**A**	**E**	**D**	**C**
B	**-**	**E**	**B**	**A**	*****	**B**	**A**	**E**	**D**
C	*****	**C**	**E**	**B**	*****	**C**	**B**	**A**	**E**
D	*****	**D**	**C**	**E**	*****	**D**	**C**	**B**	**A**
E	**-**	**D**	**C**	**E**	*****	**E**	**D**	**C**	**B**

LRU: (Least Recently Used)

- Bij referentie krijgt een pagina een “tijdstempel”
- Veel extra administratie
  - Tijd in pagetabel entry
  - Lijst manupilatie

→hardware voorzieningen nodig

LFU: (Least Frequently Used)

- Teller per pagina, opgehoogd bij referentie
- Gevaar: nieuwe pagina’s nog weinig gebruikt
  in het verleden gebruikt blijft aanwezig

NUR: (Not Used Recently)

- Twee bits per pagina
  1. Reference bit
  2. Modified bit

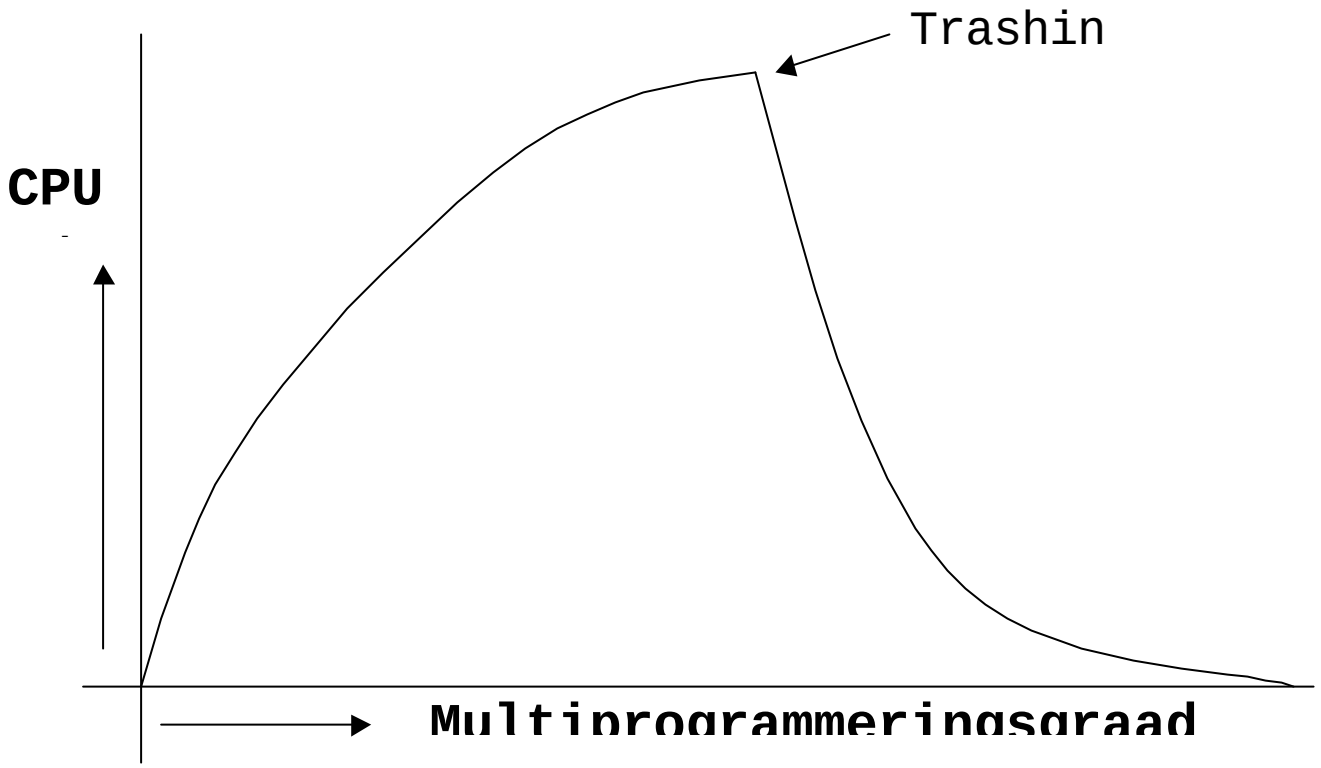
  Wordt een pagina gerefereerd: r ← 1
  Wordt in een pagina geschreven: m ← 1

- Periodiek worden alle r-bits op 0 gezet

- Replacement schema

1^e	r = 0, m = 0
2^e	r = 0, m = 1
3^e	r = 1, m = 0
4^e	r = 1, m = 1

Het verschijnsel TRASHING



Er moet in het werkgeheugen ruimte zijn voor de “actieve” pagina’s van actieve processen

Remedies tegen Trashing:

- **Pagefault frequentie**
  - Ondergrens: als PFF “te laag”, - verminder aantal frames
  - Bovengresen: als PFF “te hoog”, - vermeerder aantal frames

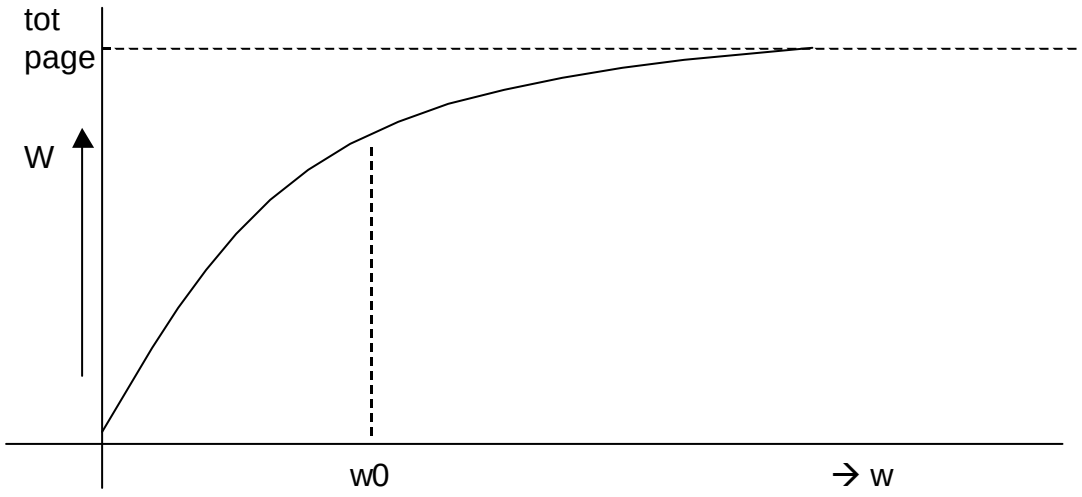
- **Working set**
  Het toegewezen aantal frames wordt bepaald door het aantal pagina’s in het recente verleden gerefereerd is.

Bepalen van de working set:

Bekijk het verleden

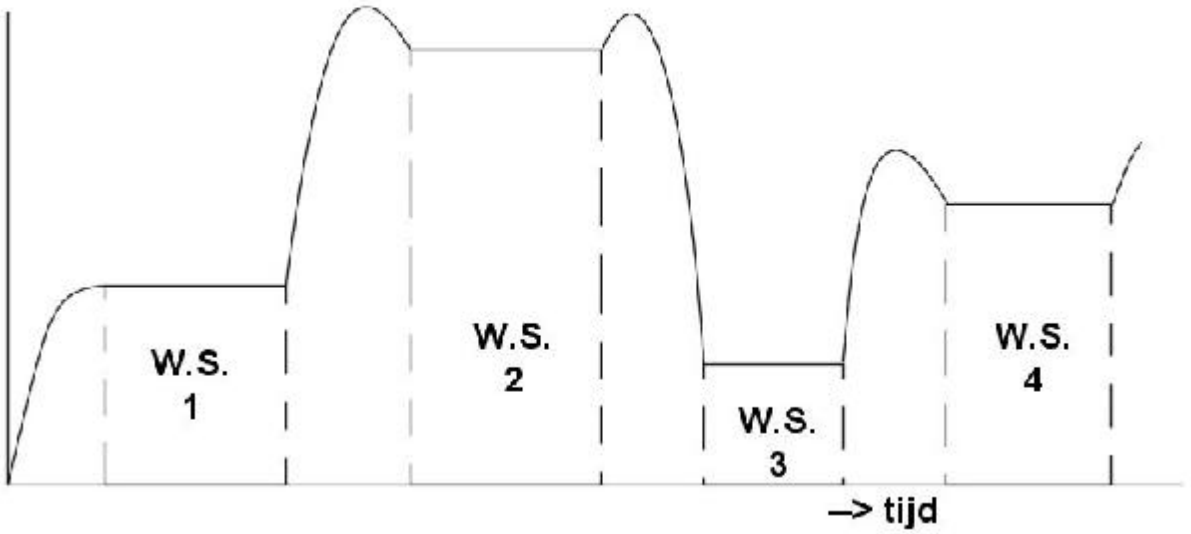
Def: W(t,w) = verzameling pagina’s gerefereerd in het tijdsinterval <t-w,t>

Verwachting van W:



Dus: hoe groter w0, des te minder verandert W. w0 dient in de praktijk bepaald te worden

Extra probleem:
Per proces varieert de grootte van de w.s. tijdens de executie



De working set verandert in de tijd.

Bestandsbeheer File Systems

Fysiek:
Achtergrondgeheugen, Secundair geheugen: verzameling blokken

Volume DASD: schijf, trommel, partitie
Sequentieel: magneetband

Logisch:
Bestand File

bestand = een verzameling samenhangende gegevens, veelal voorzien van een naam.

Het Besturingssysteem overbrugt het gat tussen de fysieke en de logische structuur.

Bestandsbeheer beheert Bestanden.

- naamgeving
- structuur
- toegang
- gebruik
- bescherming
- implementatie

Naamgeving:

- bouwstenen:
  - hoofd- en/of kleine letters
  - cijfers
  - speciale tekens

- structuur:
  - twee of meer delen
  - naam.extensie[.extensie...]
  - conventies voor extensies

Structuur:

- sequentie van bytes (ongestructureerd)
  voordeel: maximaal flexibel
  nadeel: alles zelf organiseren

- sequentie van records (vaste lengte)
  voordeel: os doet de recordorganisatie

- boom van records (b-tree)
  zowel sequentieel als direct

Soorten bestanden:

- gegevensbestanden
  - ascii
  - binaire
    - executeerbare
    - object
    - archief
- administratiebestanden (directorie)
- karakter-speciaal
- blok-speciaal

Toegang:

- sequentiëel
- direct

Attributen: zie lijst

- plaats (bijv dir,i-node)

Operaties:

- creëer	create
- verwijder	delete
- open	open
- sluit	close
- lees	read
- schrijf	write
- voegtoe	append
- positioneer	seek
- haal attributen	get attributes
- zet attributen	set attributes
- hernoem	rename

file descriptor:

```
int fd, in, buffer[BUFFR_AFMETING];
fd = open("Bestand", read_only);
in = read(fd, buffer, BUFFER_AFMETING);
close(fd);
```

Administratie van bestanden

directorie:

- boomgestructureerd (zie figuur)
  - enkelvoudig, 1-niveau
  - per gebruiker, 2-niveau
  - meer-niveau (willekeurig)
  - acyclische graaf (dwz met 'links' tbv gemeenschappelijk gebruik)
  - één boom (Unix)
  - boom per volume (Dos)

- padnaam
  - absoluut
  - relatief

- werk-directorie, actieve directorie
  (working-, current- directory)

- bijzondere namen:
  "." (punt, actieve directorie)
  ".." (puntpunt, ouder, parent)

- operaties op directories:

. creëer	create
. verwijder	delete, remove
. open dir	open dir
. sluit dir	close dir
. lees dir	read dir
. hernoem	rename
. koppel	link
. ontkoppel	unlink

Implementatie

Toewijzing (allocatie) van DASD-ruimte

Problematiek:

- efficiëntie
  - . gebruik
  - . toegang
- fragmentatie
- reorganisatie

* Aaneensluitende blokken:

voordeel: eenvoudige implementatie
uitmuntende prestatie

problemen: lengte bekend?
fragmentatie (als MVT)

* Ketting van blokken:

voordeel: geen fragmentatie
ok voor sequentiële toegang

problemen: willekeurige toegang slecht
ruimteverlies (wijzers)

* Ketting mbv index-blokken:

voordeel: index-sequentiëel: sequentiële en willekeurige toegang
geen ruimteverlies tbv wijzers

problemen: meer I/O nodig
ruimteverlies door indexblokken

* Ketting mbv blokkaart (Dos, Fat):

voordeel: nadelen van gewone ketting geëlimineerd

problemen: gehele blokkaart nodig in ram

* Ketting mbv i-nodes (Unix, Minix):

lijkt op ketting met index-blokken:
wijzerblokken op meer niveaus:
direct, enkel-, dubbel-, tripel-indirect

opm: snel voor kleine bestanden, trager voor grotere bestanden door de meervoudige indirecties.

Ruimtebeheer op DASD-volumes

* Allocatie-eenheid
  - sector en blok
* Vrije ruimte
  - bitmap
  - als bijzonder bestand mbv ketting
  - in de blokkaart
* Quotering

Betrouwbaarheid

* Slechte blokken

  - hardware: vervang door reserve, dit wordt door de controller gedaan mbv een lijst van slechte blokken

  - software: bestandsbeheer alloceert slechte blokken aan een bestand

* Bewaren & herstellen Backup & recovery

  - Periodieke dump
    . simpel, systeem niet bruikbaar,
    . eenvoudig herstel,
    . laatste transacties verloren

  - Incrementele dump
    . veranderd bestand wordt bewaard
    . tijdens gebruik
    . ingewikkeld herstel

  - Extra: Bewaren van alle transacties, als niets verloren mag gaan

* Consistentie van het bestandssysteem

  Inconsistentie kan ontstaan als,terwijl er bestanden open zijn, het systeem crasht. Herstel is niet altijd mogelijk, wel kan de consistentie van blokken en bestanden getoetst worden.
  Bijv CHKDSK (Dos), FSCHK (Unix)

Prestatie Performance

* Blok-cache (of buffer-cache)

  - vergelijk met paginering, techniek: LRU met categorie-indeling

  - gevoelig voor inconsistentie, daarom:
    'write-through cache' (Dos);
    SYNC system call, process UPDATE (Unix).

* minimalisatie van de beweging van de arm ('seek') door aaneensluitende allocatie

* allocatie op dezelfde cilinder met rotatie-optimalisatie

* veelgebruikte blokken (directories, i-nodes) in het midden van de schijf

* verdeling van een volume in cilinder-groepen

Directorie-ingang en
BestandsAttributen van CP/M

offset	inhoud
0	identificatie van de eigenaar
1	naam van het bestand
9	extensie van de naam van het bestand
12	extent: vlag als meer dan 16 blokken
13	gereserveerd
15	aantal blokken gebruikt
16	16 bytes voor maximaal 16 bloknummers

Voor een bestand dat bestaat uit meer dan 16 blokken worden één of meer extra ingangen van de directorie gealloceerd (opeenvolgend)

==========

Directorie-ingang en
BestandsAttributen van UNIX

offset	inhoud
0	nummer van de i-node
2	naam van het bestand (14 bytes)

attributen in de betrokken i-node

Directorie-ingang en
BestandsAttributen van MS-DOS

offset	inhoud
0	Naam van het bestand waarbij de eerste byte een speciale betekenis heeft: 00h: nooit gebruikt E5h: bestand is gewist 05h: 1e teken van de naam is E5h
8	bestands-extensie
11	bestands-attribuut bit 0: read only bit 1: hidden bit 2: system bit 3: volume label bit 4: subdirectory file bit 5: archive
12	gereserveerd
22	tijdstip van creatie of laatste verandering
24	datum van creatie of laatste verandering
26	clusternr (nummer van eerste gealloceerde cluster)
28	lengte van het bestand (lage 2 bytes)
30	lengte van het bestand (hoge 2 bytes)

Voorbeeld: MS-DOS: ketting mbv index (FAT)

Cluster= de eenheid waarin schijfruimte gealloceerd wordt.
Cluster van 5.25 DSDD diskette is twee opeenvolgende sectoren.
Clusterindeling van de 5.25 DSDD diskette:

	spoor 0						spoor 1					
sector	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
cyl 0	btf1		f1f2		f2 d		d d		d d		d d		c-02		c-03		c-04	
cyl 1	c-05		c-06		c-07		c-08		c-09		c-10		c-11		c-12		c-13	
cyl 2	c-14		c-15		c-16		c-17		c-18		c-19		c-20		c-21		c-22	
.																		
cyl 39	c347		c348		c349		c350		c351		c352		c353		c354		c355	

waarin bt = boot record, f1 = FAT, f2 = FAT(kopie),
c_XXX = clusternummer.

De File Allocation Table registreert het clustergebruik in een "block (cluster) map".
Van het eerste aan een bestand gealloceerd cluster is het nummer vastgelegd in de betrokken directorie-ingang. Dit nummer verwijst naar de betrokken ingang in de FAT. In deze ingang staat het nummer van het volgende gealloceerde cluster of een code die aangeeft dat dit het laatste cluster van het bestand is, enz.
Dit betekent dat het aantal ingangen in de FAT tenminste 355 moet zijn en dat een FAT-ingang het getal 355 moet kunnen bevatten. Daarom heeft een FAT-ingang een lengte van 1.5 bytes. Gezien de clusternummering van 2 t/m 355 moeten er dus tenminste 354 FAT-ingangen aanwezig zijn. Dus zijn er voor de FAT twee sectoren gereserveerd. Om veiligheidsredenen houdt het systeem een extra kopie van de FAT bij.

Een FAT-ingang bevat dus een clusternummer van het volgende aan het betrokken bestand gealloceerd cluster

- **of een van de waarden FF8h t/m FFFh als dit het laatste cluster van een bestand is,**
- **of 000h als dit een vrij cluster is,**
- **of een van de waarden FF0h t/m FF6h als dit een speciaal MS-DOS cluster is,**
- **of FF7h als dit een "bad" (niet te gebruiken) cluster is.**

Clusternummering begint bij 2 omdat de eerste twee FAT-ingangen speciale gegevens bevatten: de eerste byte bevat de omschrijving van het betrokken medium, bijv FDh voor een 5.25 DSDD-diskette, de rest bevat altijd FFh.

NB: FAT-ingangen kunnen twee bytes groot zijn afhankelijk van het betrokken medium.

MinixBestanden

Elk volume heeft een standaard indeling:

- boot-blok
- super-blok
- bitmap van de i-nodes
- bitmap van de (data)blokken
- blokken met i-nodes
- blokken met gegevens

Het SUPERBLOK bevat de opmaak van het bestandssysteem op het betrokken volume.

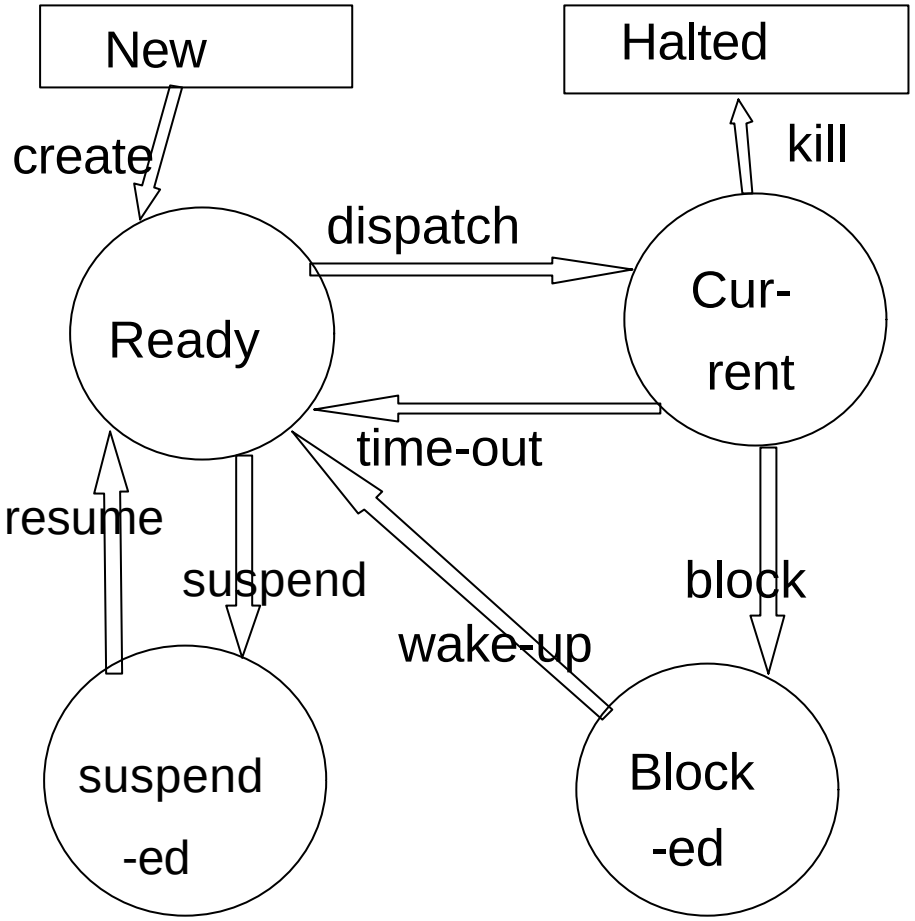
i-nodes
data-blokken
blokken van de bitmap van de i-nodes
blokken van de bitmap van de datablokken
wijzer naar 1e datablok
log2 (bloklengte / sectorlengte)
maximum afmeting van een bestand
magisch getal
#wijzers -> bitmapblokken van i-nodes
#wijzers -> bitmapblokken van datablokken
nummer van dit volume
i-node van de locale hoofddirectorie
i-node van het 'aanhaak-bestand'
tijd van laatste wijziging
'read-only', 'dirty'-vlaggen
. .

I-node bevat de attributen van het bestand

<---------- 16 bits ---------->

bestandssoort en RWX-bits	
identificatie van de eigenaar	
afmeting van het bestand	
tijd van laatste wijziging	
# links	indentificatie van de groep
nummer van datablok 0	
nummer van datablok 1	
nummer van datablok 2	
nummer van datablok 3	
nummer van datablok 4	
nummer van datablok 5	
nummer van datablok 6	
nummer van blok van enkele indirectie	
nummer van blok van dubbele indirectie	

Proces-toestand-overgang-diagram



Toestandsovergangen:

```
'Create':      nieuw proces --> ready
'Dispatch':    ready proces --> current
'Time-out':    current proces --> ready
'Block':       current proces --> blocked
'Wake-up':     blocked proces --> ready
'Kill':        current proces --> halted
'Suspend':     ready proces --> suspended
'Resume':      suspended proces --> ready
```

SCHEDULING (Werkindeling)

Short Term Scheduler: Ready -> Running

Long Term Scheduler: New -> Ready

Medium Term Scheduler: suspend, resume

(Disk Scheduler: later)

Criteria:

- **eerlijkheid**
  ieder proces eerlijk deel CVE

- **efficientie**
  maximaliseer zinvol gebruik CVE

- **responstijd**
  optimaliseer tbv interactieven

- **doorlooptijd (turn-a-round)**
  minimaliseer tijd t/m uitvoer

- **bezettingsgraad (throughput)**
  maximaliseer # processen / uur

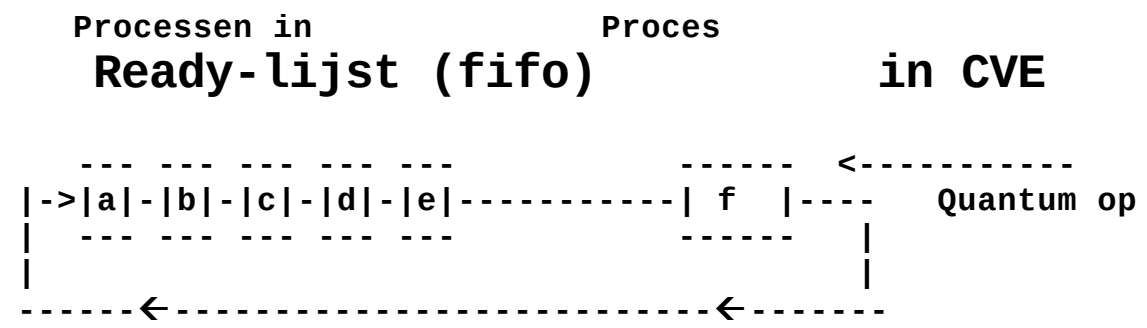
Scheduling Algoritmen (algemeen):

- **preemptive ("CVE afneembaar")**

- **non-preemptive ("CVE niet afneembaar")**

Round Robin (preemptive)

Ieder proces krijgt gedurende een tijds-interval (= QUANTUM) de CVE toegewezen.



- **eenvoudig**
- **eerlijk**
- **geschikt voor interactief**

Probleem: Quantum-grootte

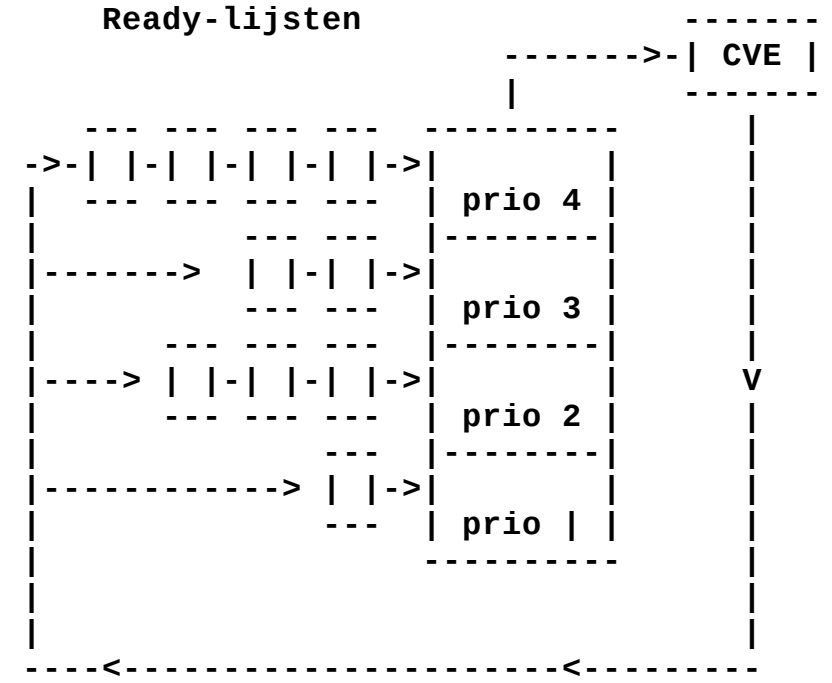
klein: niet efficient
groot: slechte respons

vuistregel: 100 msec

Scheduling op grond van prioriteiten

Per prioriteit een Ready-Lijst

Per Ready-Lijst bv: RR, FIFO



- preemptive vs (per prio)non-preemptive
- prioriteit statisch of dynamisch
- I/O-gebonden vs CVE-gebonden

VB: Prio := 1 / f
waarin f = verbruikte deel van laatste quantum

- zonder dynamische prioriteitsaanpassing kans op uithongering van lage prioriteiten

Meer wachtrijen

Processen worden ingedeeld in een van meer wachtrijen op grond van bepaalde criteria.

Voorbeeld:

Elke wachtrij heeft een eigen quantum, naarmate het quantum van een wachtrij kleiner is wordt hij eerder geselecteerd.

Afhankelijk van het gedrag van een proces, wordt het in een bepaalde wachtrij geplaatst.

Is een proces CVE-gebonden, dan wordt het in een wachtrij met een groter quantum geplaatst.

Maakt een proces zijn quantum niet vol, dan wordt het in een 'hogere' wachtrij geplaatst; overschrijdt een proces zijn quantum, dan wordt het in een lagere wachtrij geplaatst.

Shortest Job First (SJF)

- non-preemptive ('run-to-completion')
- voor batch-systemen
- afhankelijk van door de gebruiker opgegeven rekentijd (pasop: misbruik)

nadeel:
uithongering mogelijk, remedie: aging

voordeel:
gemiddelde doorlooptijd minimaal

Ook voor interactief??

Beschouw elke 'CPU-burst' als een job en schat de duur van de volgende burst naar aanleiding van de duren van voorgaande bursts.

vb: exponentieel gemiddelde

$$t_{n+1} = a\,t_n + (1-a)\,T_n$$

t_{n+1}	t_n	T_n
schatting volgende burst	vorige burst	geschiedenis

Scheduling met garantie

- gebruiker krijgt ongeveer 1/n-de van de beschikbare CPU-tijd als er n gebruikers zijn

dwz:
recht = verbindingstijd / n (stel = a)
bekend is hoeveel reeds gebruikt (b)
Dan: naarmate a/b groter is, wordt het proces eerder geselecteerd.

Deadline

- het proces dat de grootste kans heeft om zijn deadline te missen, wordt geselecteerd.

Scheduling op twee niveaus

- samenwerking tussen 'short-term'- en 'medium-term'-schedulers.

Deadlock (Dodelijke omarming)

Introductie met voorbeelden:

1. **Verkeersprobleem met 4 rijen**
2. **T-kruising**
3. **Processen en hulpmiddelen in ring**

(onvolledige) Omschrijving:

Wachten op een gebeurtenis die nooit zal optreden.

Aandachtsgebieden

- **Voorkomen**
- **Vermijden**
- **Ontdekken**
- **Herstellen na**

Gerelateerd probleem:

- Eindeloos uitstel

Noodzakelijke en voldoende voorwaarden voor het optreden van deadlock

- **“Mutex”: Processen eisen exclusieve toegang tot een hulpmiddel.**
- **“Wacht op”: Processen bezitten (mutex)-hulpmiddelen en wachten op andere.**
- **“Niet preemptief”: Een (mutex)-hulpmiddel blijft toegewezen tot het proces het vrijwillig afstaat.**
- **“Ring-wacht”: Er bestaat een gesloten keten (ring) van processen waarin elk proces een (mutex-)hulpmiddel bezit dat gevraagd wordt door het volgende proces in de ring.**

Deadlock-gevoeligheid van Hulpmiddelen

- **CPU**
- **Geheugen**
- **Schijf**
- **Sequentieel randapparaat**
- **Bestand**
- **Code**
- **Data**

Welke problematisch?

Voorkomen van deadlock

Voorkomen: een omgeving scheppen waarin deadlock onbekend is.

Hoe: probeer een omgeving te scheppen waarin aan tenminste 1 van de voorwaarden niet wordt voldaan.

- Mutex:
  hef de exclusieve toegang tot een hulpmiddel op.
  Bv: spooling, herbetreedbare code

- Wacht op:
  vraag alle hulpmiddelen tegelijk aan.

- Niet preemptief:
  dwing een proces een hulpmiddel terug te geven.
  Bijv: cpu-scheduling

- Ring-wacht:
  zorg dat er geen ring kan ontstaan.
  Bijv: data, aanvragen in volgorde van oplopend rangnummer.

Vermijden van deadlock

Voor sequentiële randapparaten.

Bankiers-algoritme (Dijkstra, 1965)

Een bankier heeft een bepaald aantal guldens beschikbaar om uit te lenen aan een vast aantal klanten. Dat gebeurt volgens een vastgesteld protocol.
Het protocol luidt:

- De klant geeft van te voren op hoeveel hij maximaal zal lenen (maximale claim).
- De klant belooft het geleende bedrag binnen afzienbare tijd terug te betalen.
- De klant heeft geen haast.
- De bankier leent dan en slechts dan een bedrag uit als hij zeker is dat hij elke klant uiteindelijk het gevraagde bedrag zal kunnen lenen.

Dit laatste betekent concreet het volgende:
1. Hij staat een lening toe als er daarna nog tenminste 1 klant die zijn totale claim kan krijgen.
2. Hij laat deze klant schijnbaar het totale geleende bedrag teruggeven en bekijkt of er nu weer tenminste 1 klant is die zijn totale claim kan krijgen. Daarna weer naar 2 tot er geen klanten meer zijn.

Voorbeeld:

In een computersysteem kunnen 2 processen P en Q beschikken over 5 tape-eenheden.
De begintoestand:

	Toegewezen	open claim	Max
P	0	4	4
Q	0	3	3

Beschikbaar: 5

P krijgt 2 eenheden, Q 1 eenheid:

	toegewezen	open claim	Max
P	2	2	4
Q	2	3	3

Beschikbaar: 2

Bekijk de volgende voortzetting:

- P bezet er nog 2
- Q bezet er nog 2
- P bezet er 1, Q bezet er 1

Wat zijn de consequenties?

Dus: de gevolgen van een toewijzing moeten geëvalueerd worden.

Onderscheid:

- Een veilige toestand: alle klanten kunnen zeker binnen afzienbare tijd tevreden gesteld worden.
- In tegenstelling tot een onveilige toestand waarin dat niet zeker is.

Voorbeeld:

	lening	claim	Max
P	1	3	4
Q	4	2	6
R	5	3	8

Beschikbaar: 2

Alleen Q wordt bediend, daarna bv P en R.
Dus: veilige toestand

Voorbeeld:

	lening	claim	Max
P	8	2	10
Q	2	3	5
R	1	2	3

Beschikbaar: 1

Geen proces kan nog het maximum krijgen.
Dus: onveilige toestand

Samenvatting Bankiers-algoritme

```
Doe
    Als aanvraag > claim dan fout
    Als aanvraag > beschikbaar dan wacht
    Als veilig dan wijs toe
                  anders wacht
```

Veilig-algoritme

1. Wijs schijnbaar toe.
   Merk alle processen ?

2. Zoek een proces dat kan “aflopen”.
   Als niet gevonden dan ga naar 4.

3. Laat gevonden proces alle gealloceerde middelen
   schijnbaar terug geven.
   Merk proces !
   Ga naar 2.

4. Als alle processen ! dan veilig
                         anders onveilig.

Zwakke punten van het Bankiers-algoritme

- Vereist een vast aantal middelen (problemen bij uitval)
- Gaat (in principe) uit van een vast aantal gebruikers (uithongering)
- Toewijzing “binnen afzienbare tijd” onmogelijk in een real-time-omgeving
- Gebruiker moet maximale eisen van te voren opgeven (onvriendelijk)
- Algoritme kost veel tijd:
  als n = aantal processen en m = aantal middelen,
  dan aantal operaties = m*n*n.

Ontdekken van deadlock

Manier:

Houd allocatie-grafen bij en controleer deze periodiek op de aanwezigheid van ringen.

Herstel na deadlock

Mogelijkheden:

- Beëindig alle processen betrokken bij de deadlock.

- Beëindig steeds 1 proces tot de deadlock opgeheven is.

  - Op grond van prioriteiten
  - Op grond van reeds gebruikte tijd
  - Op grond van: Hoeveel middelen nog nodig tot gereed?

- Ontneem middelen aan bepaalde processen

  - Welke processen?
  - Wat te doen met de geselecteerde processen?