Computersystems 2 Theorie

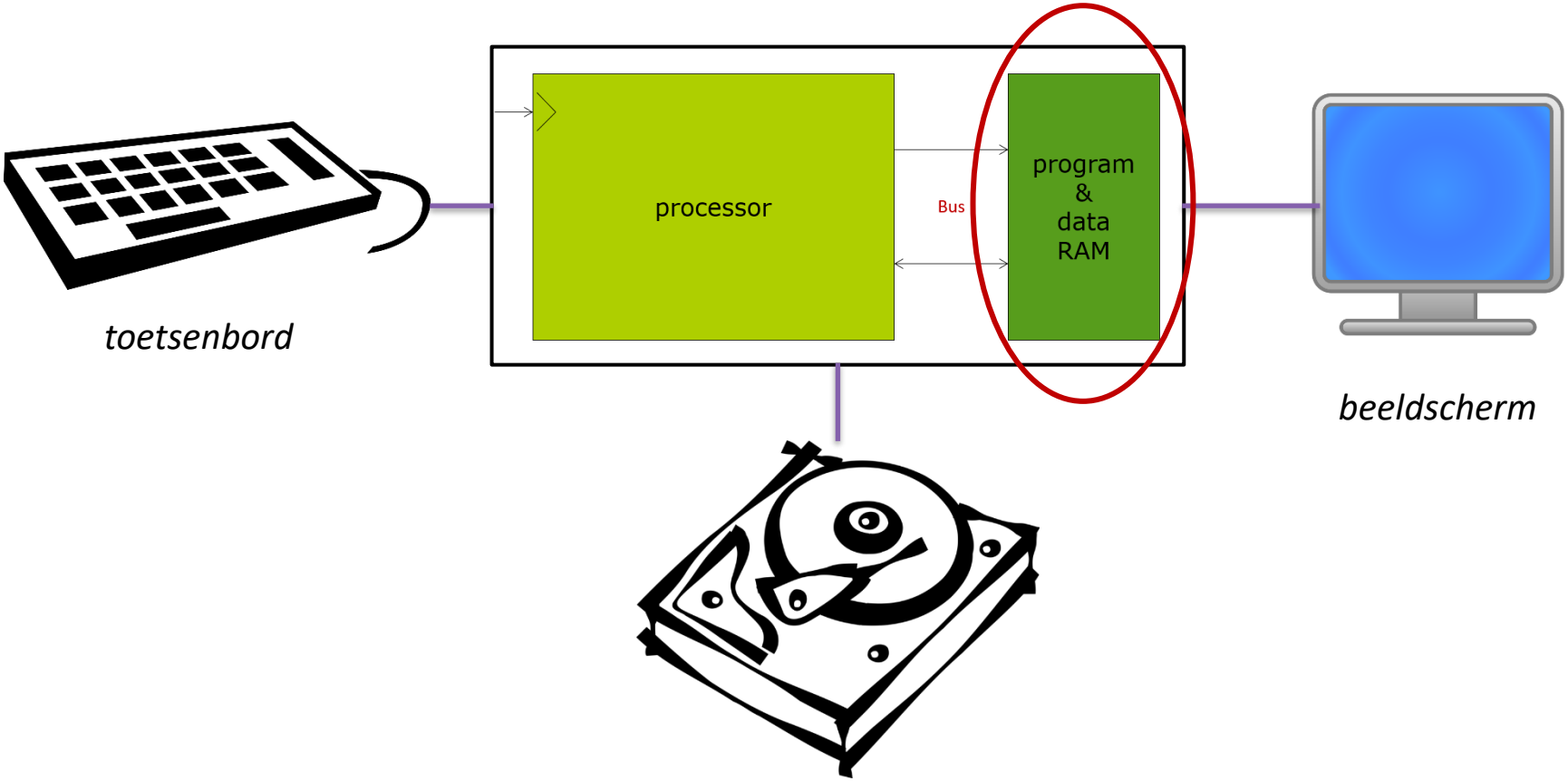
5. Geheugenbeheer

KdG Karel de Grote Hogeschool

19/11/2024

Geheugenbeheer

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



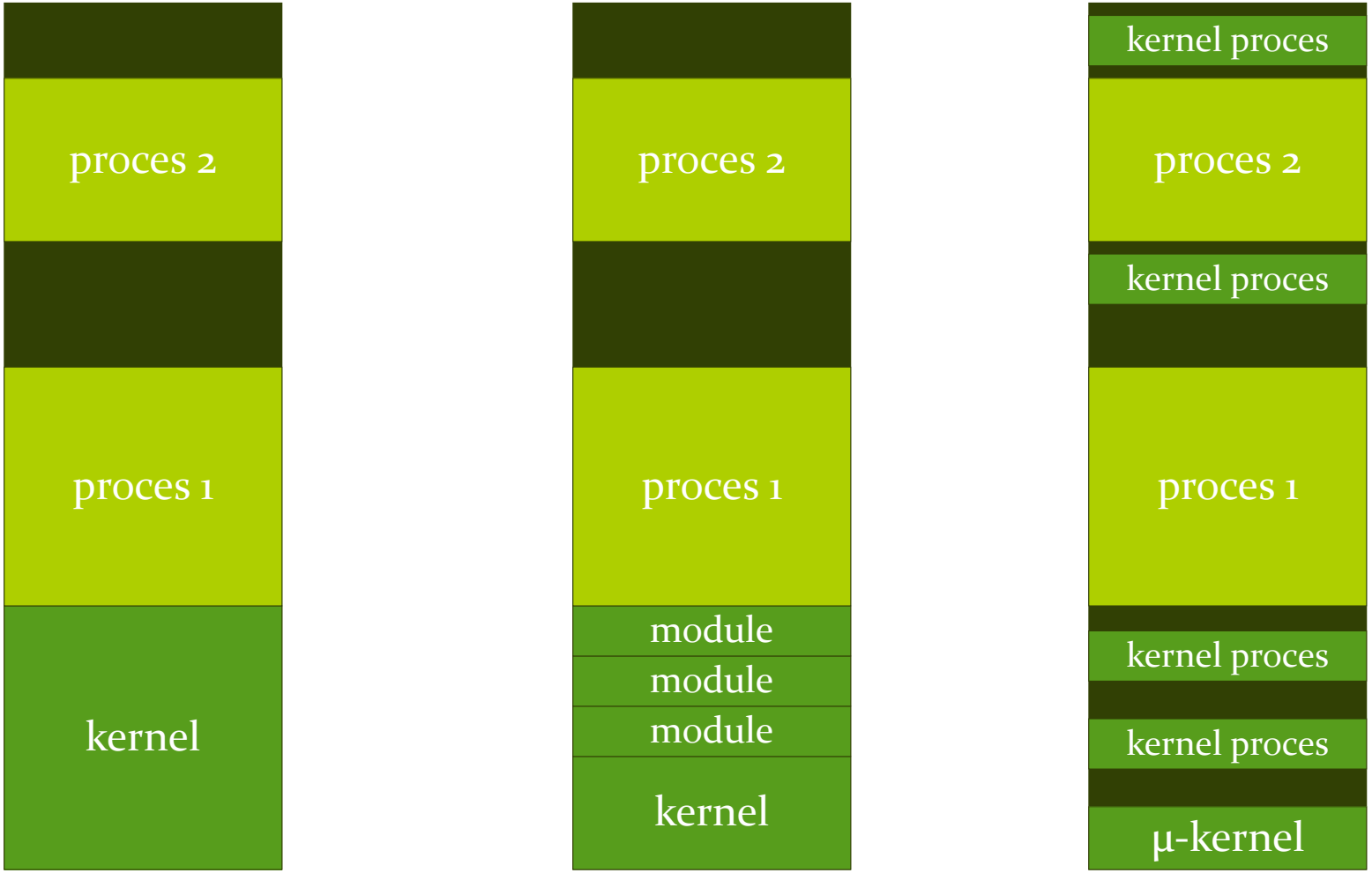
Inhoud

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

Laden van het OS

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

laden van het OS



kernel

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

Ubuntu kernel modules

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

Inhoud

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

Programma's laden

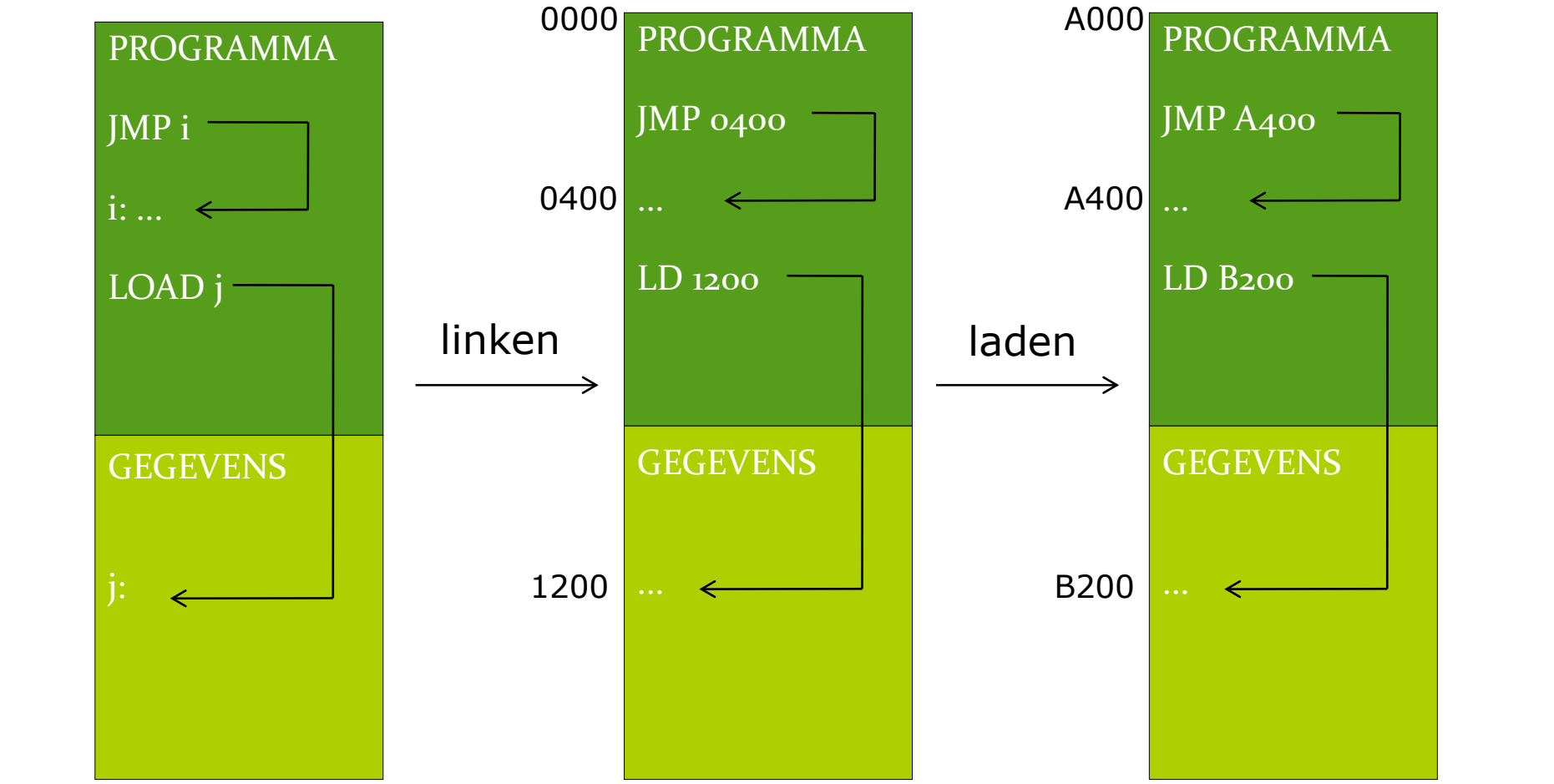
- OS start op, zit klaar in geheugen
- I/O drivers zijn geladen
- nu: programma opstarten
  - laad code/data in geheugen
  - spring naar start
- probleem: programma kan op verschillende plaatsen in het RAM geheugen terecht komen
  - wat als er een jmp instructie staat?
  - waar zal de data in het geheugen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

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

Laden - Relocatie

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

Een proces laden

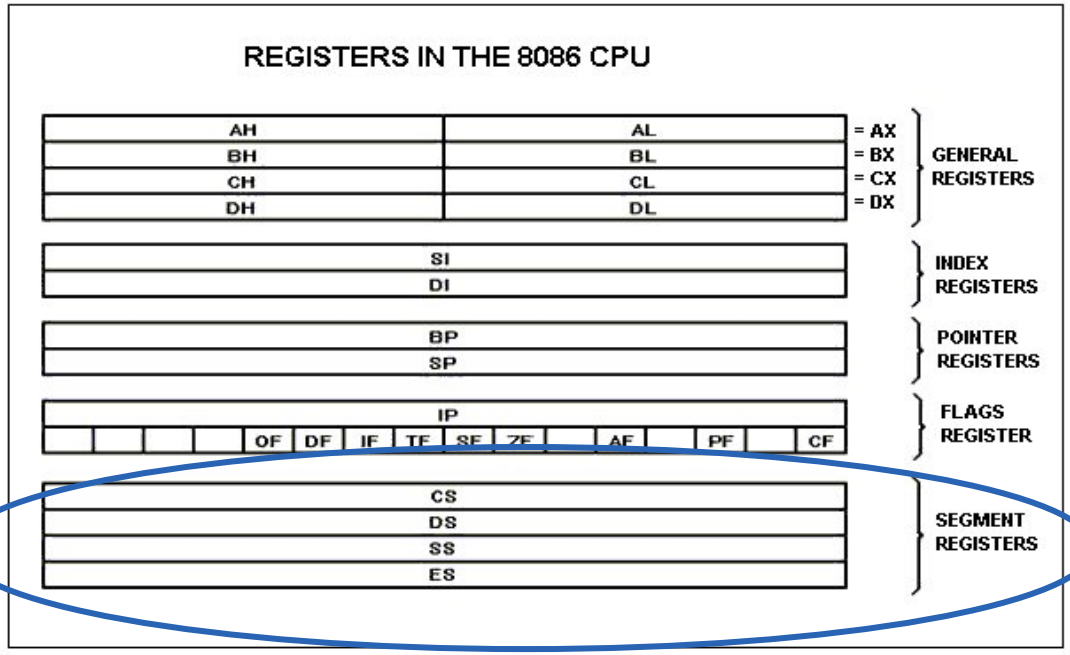


Symbolisch (met labels) Logische adressen Fysische adressen

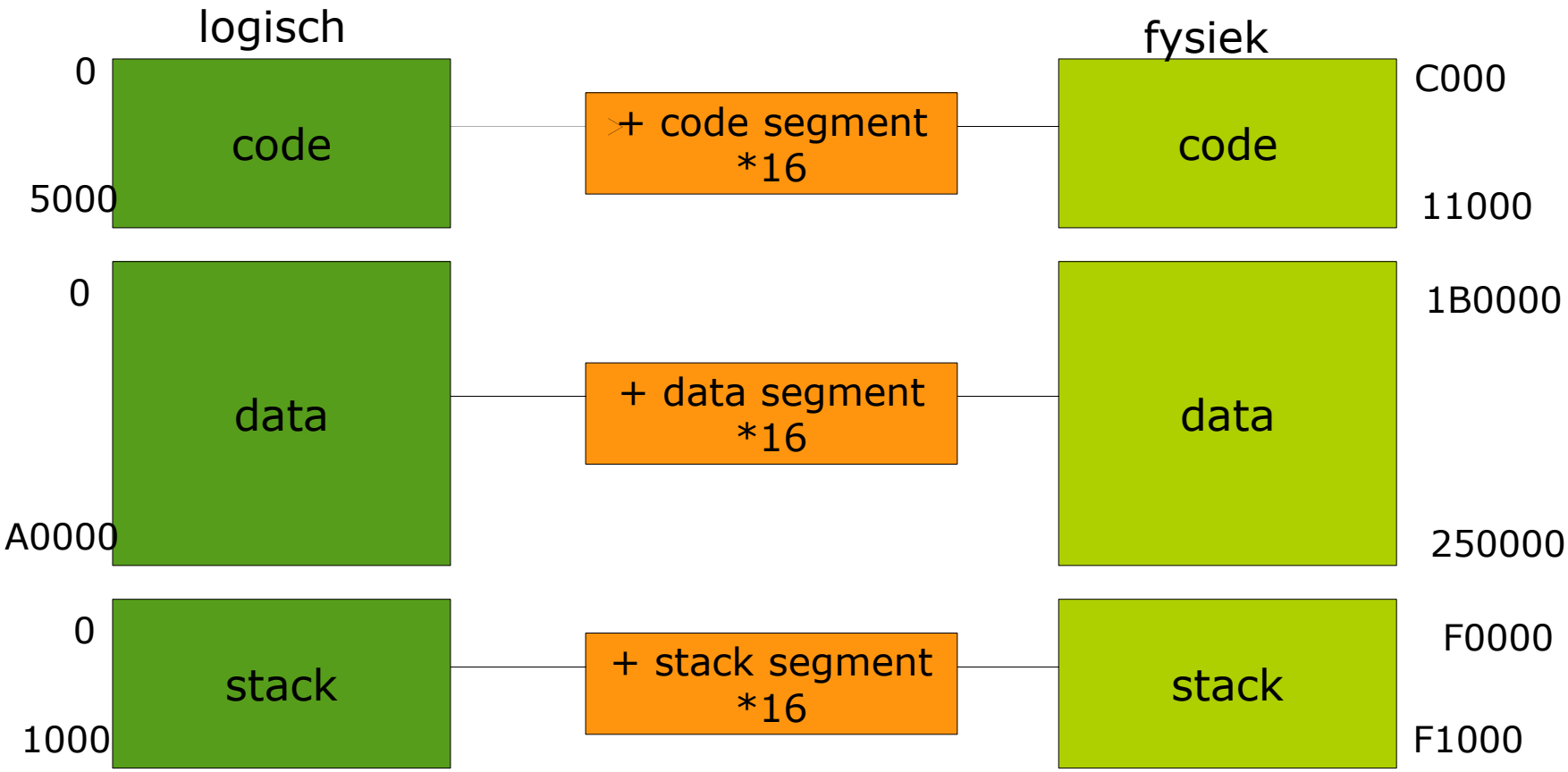
= logisch adressen + start adres

Vb. Relocatie 8086

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



Vb. Relocatie 8086



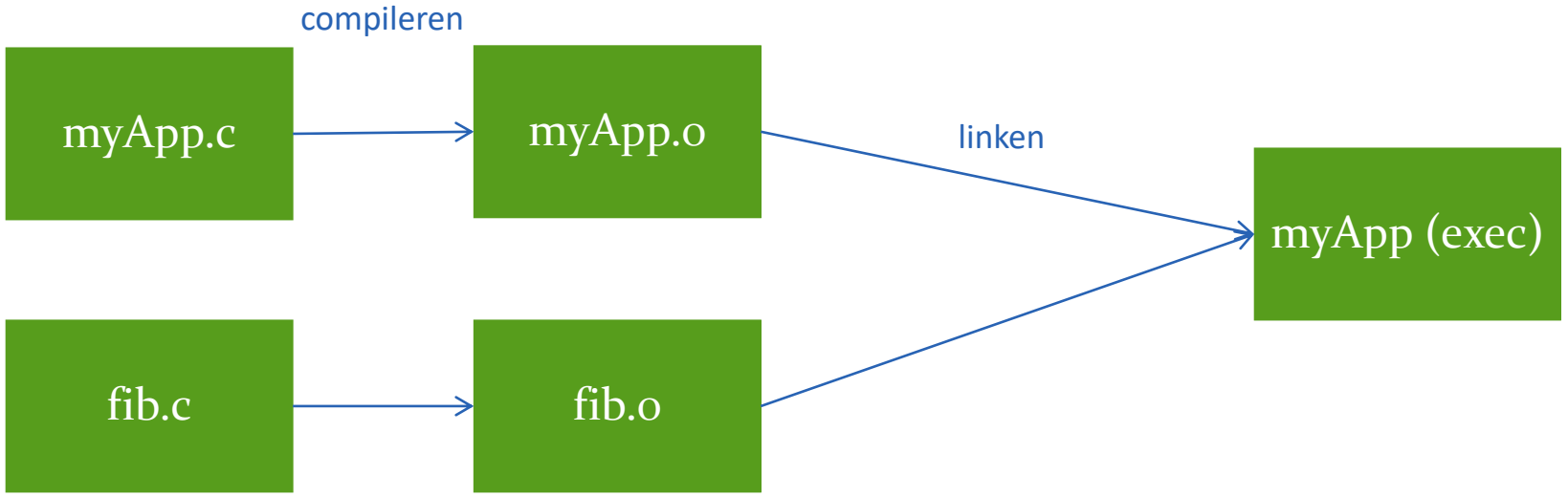
Inhoud

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

Een proces linken

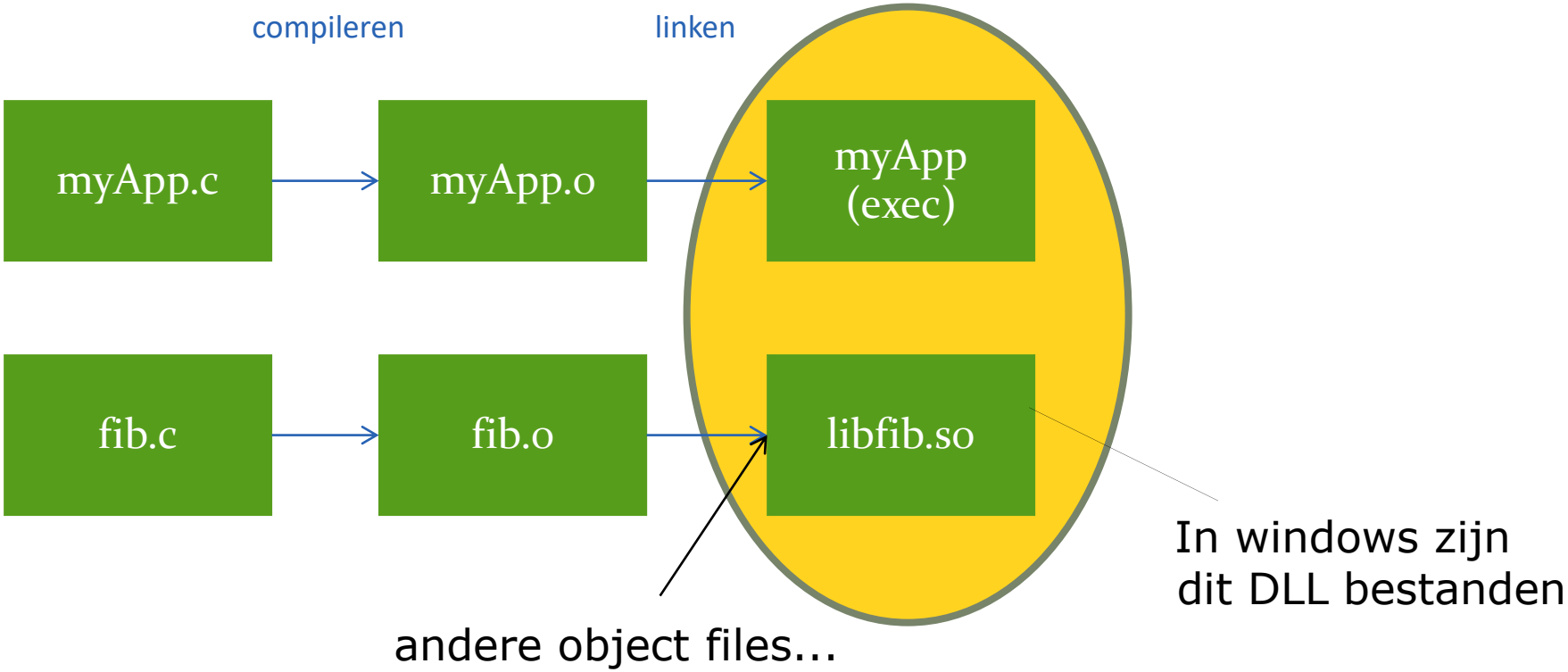
- in de praktijk
  - programma bestaat uit # modules
  - iedere module wordt gecompileerd naar object-file
  - object-file bevat code+labels
  - linker voegt object-files samen tot executable
  - linker vervangt labels door adressen
- 2 mogelijkheden
  - Statisch gelinkt
  - Dynamisch gelinkt

Statisch linken



myApp.c gebruikt een functie uit module fib.c

Dynamisch linken (1)



.so: shared object

Dynamisch linken (2)

Voordelen:

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

Compile lab

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

Inhoud

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

Geheugenbeheer

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

Geheugenbeheer
werking van de call stack

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

   int fac(int i) {
     int result = 0;

     if (i < 2) result=1;
     else result=fac(i-1)*i;

     return result;
   }
```

a=2

stack groeit naar onder

Inhoud

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

Geheugenbeheer

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

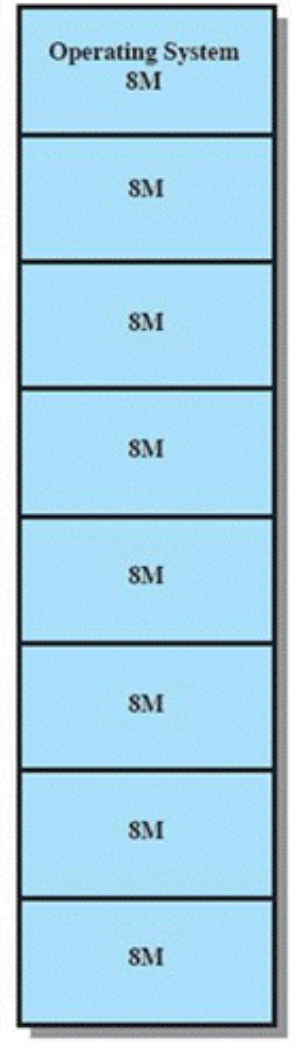
Partitionering

partitie HDD != RAM geheugen partitie

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

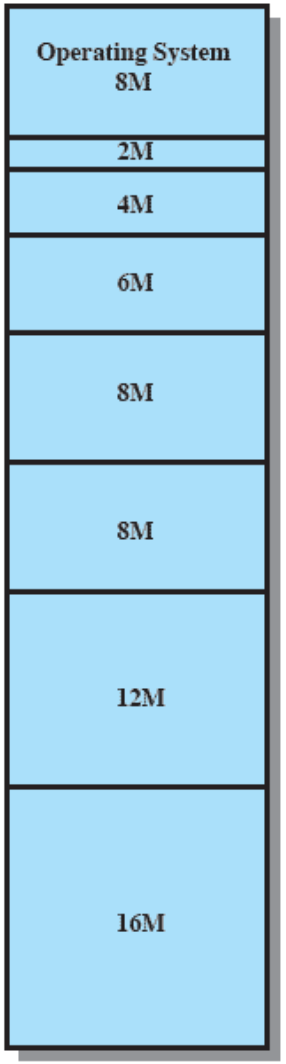
Vaste partities

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



Vaste partities

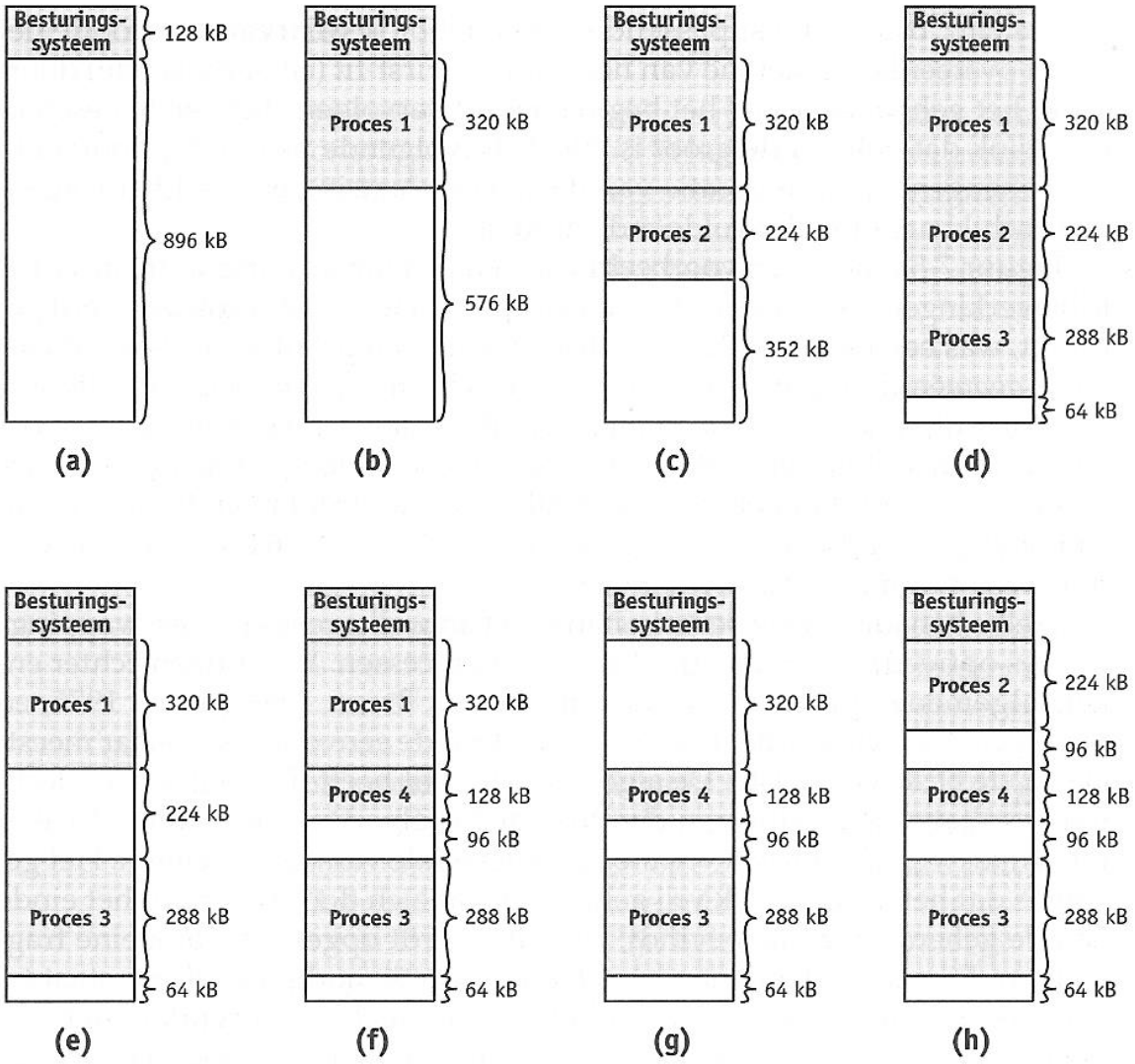
- verschillende groottes
- worden vastgelegd bij het opstarten
- proces krijgt kleinste partitie die groot genoeg is
- voordeel
  - zoals vorige
  - meer mogelijkheden voor grotere procesen
- nadeel
  - nog steeds interne fragmentatie
  - max aantal partities nog steeds vast



Dynamische partitionering

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

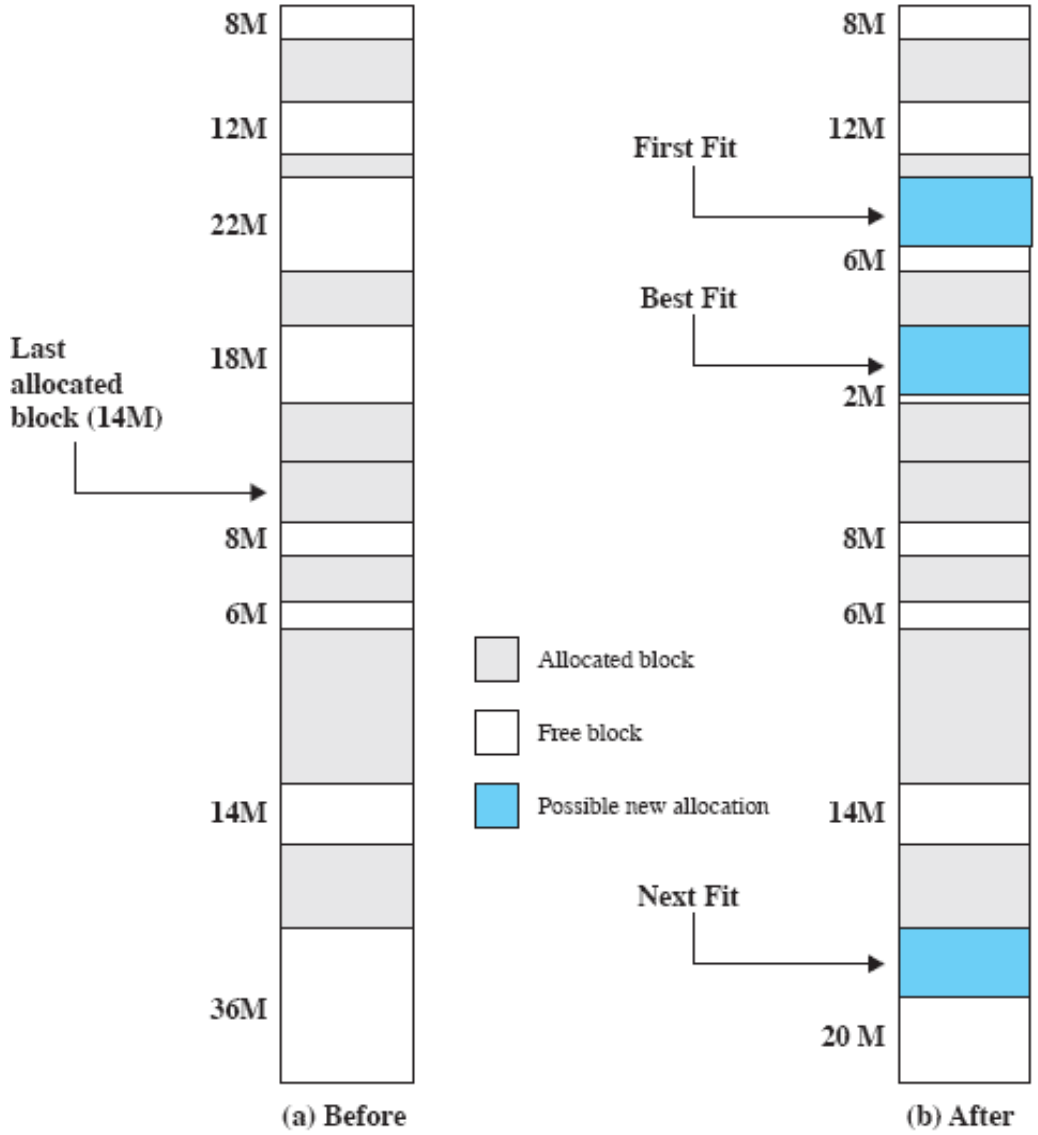
Dynamische partitionering



Dynamische partitionering

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

Dynamische partitionering

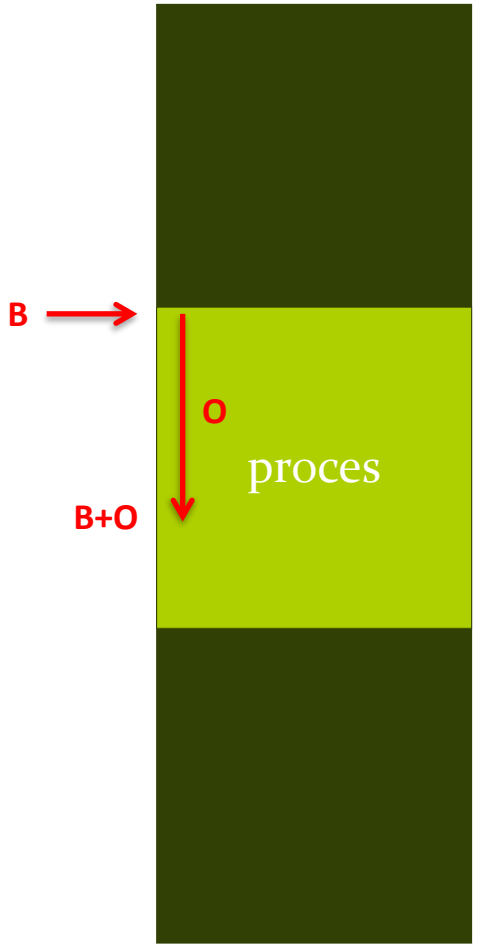


Nieuw proces: 16 MB

Welke van de 3 is het beste algoritme?

Partitionering

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



Inhoud

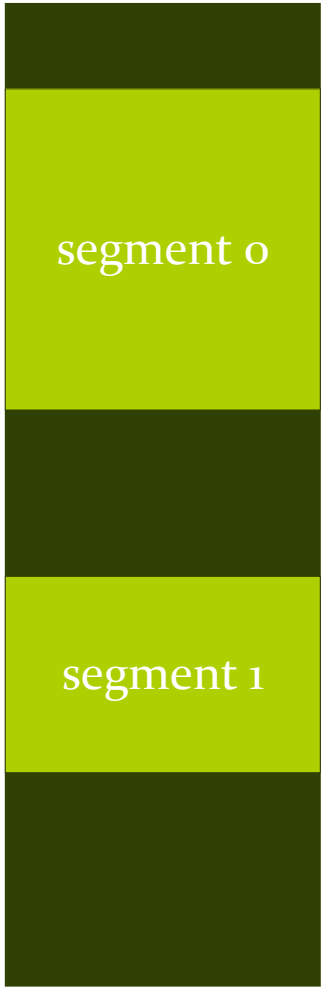
- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Overzicht

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

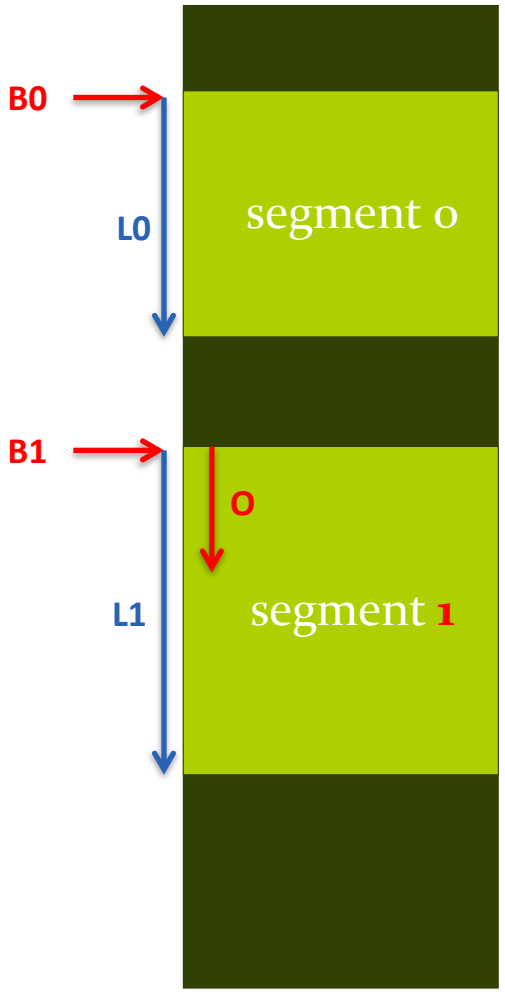
Segmentation

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

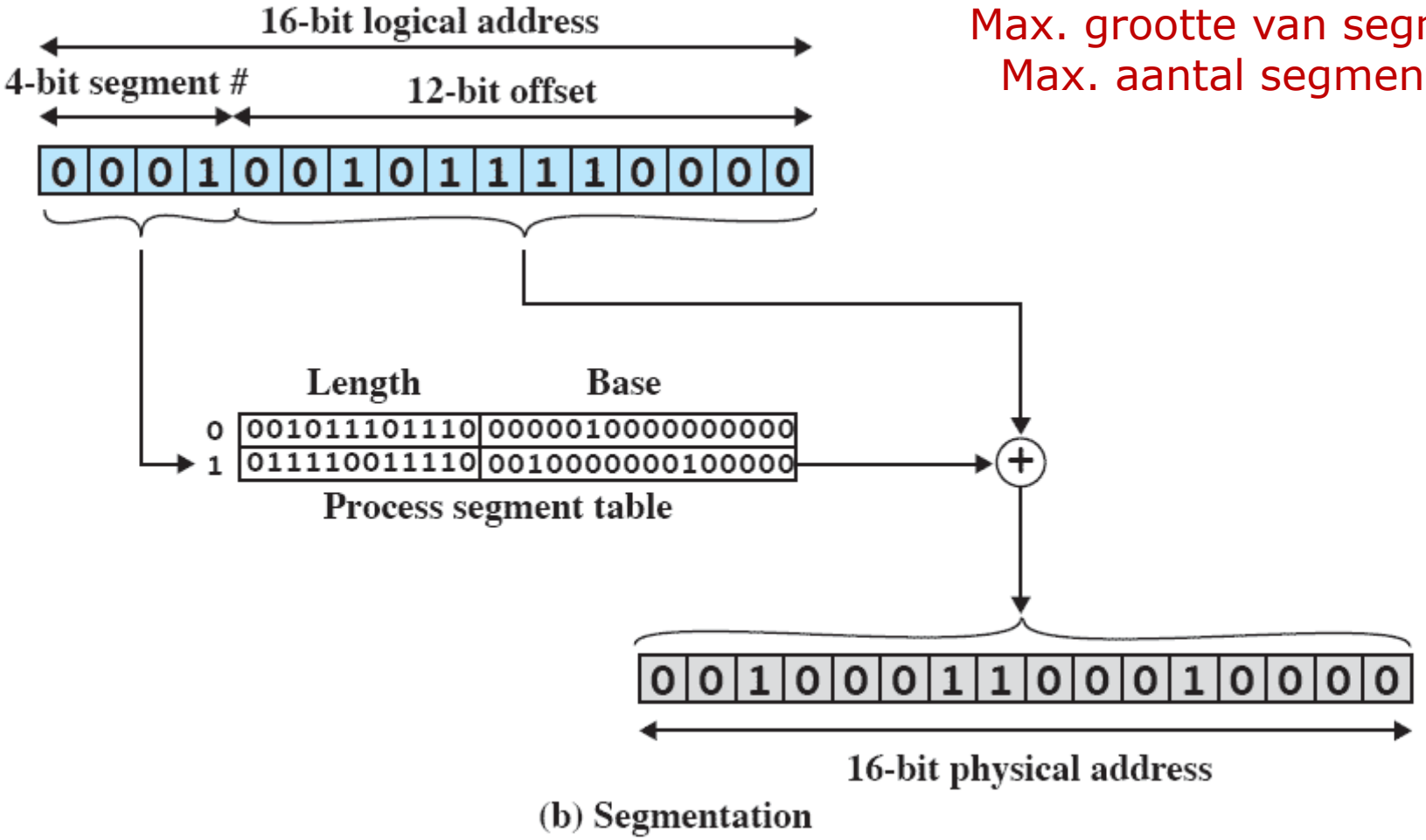


Segmentation

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



Segmentation



(b) Segmentation

Max. grootte van segment?
Max. aantal segmenten?

Segmentation

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

Oefening thuis

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

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

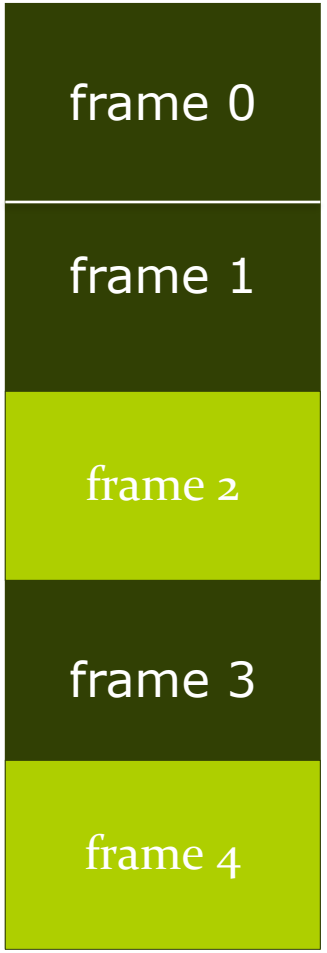
  - offset binnen segment groter dan lengte?

Inhoud

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

Paging

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

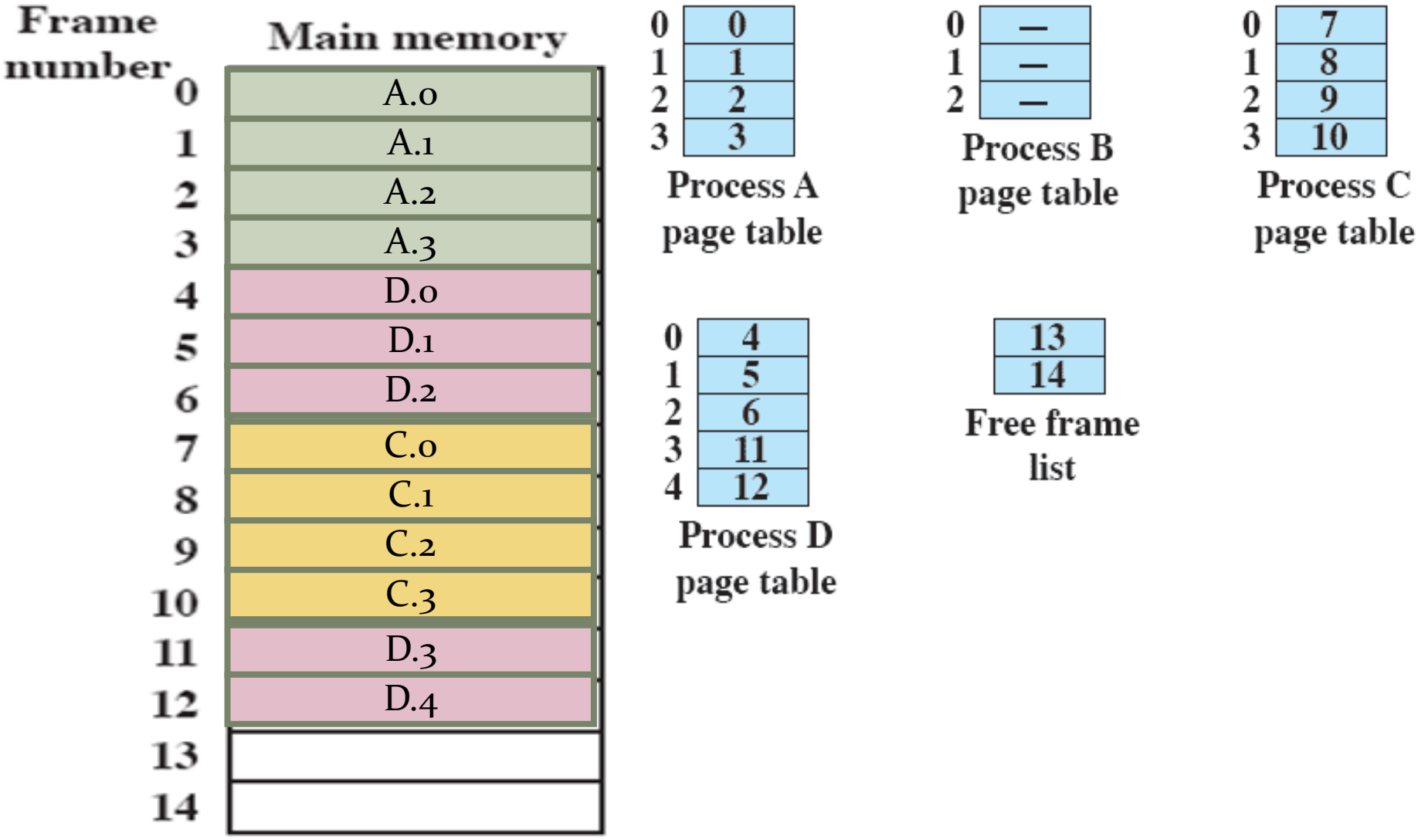


PS: principe is vergelijkbaar met dit van een filesysteem, files zitten ook in vaste blokken, maar nu gaat het over processen in het RAM geheugen

Paging

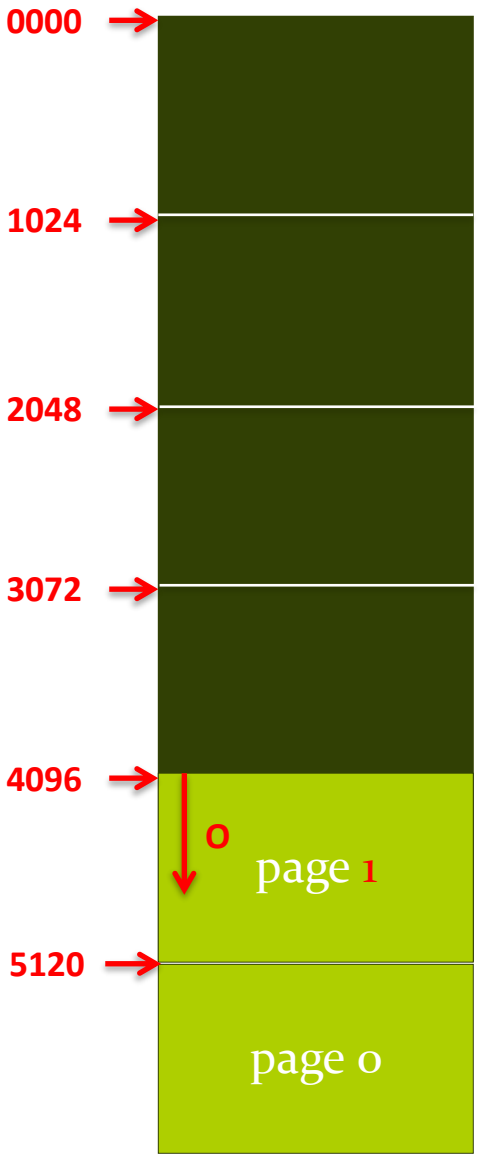
- Wanneer een programma opgestart wordt (het wordt van de HDD in het RAM geheugen geladen)
  - Proces wordt opgedeeld in blokken, pages genoemd, even groot als de frame grootte
  - Deze pages worden toegewezen aan vrije frames in het RAM geheugen
  - De page table houdt bij in welke frames het proces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

Paging

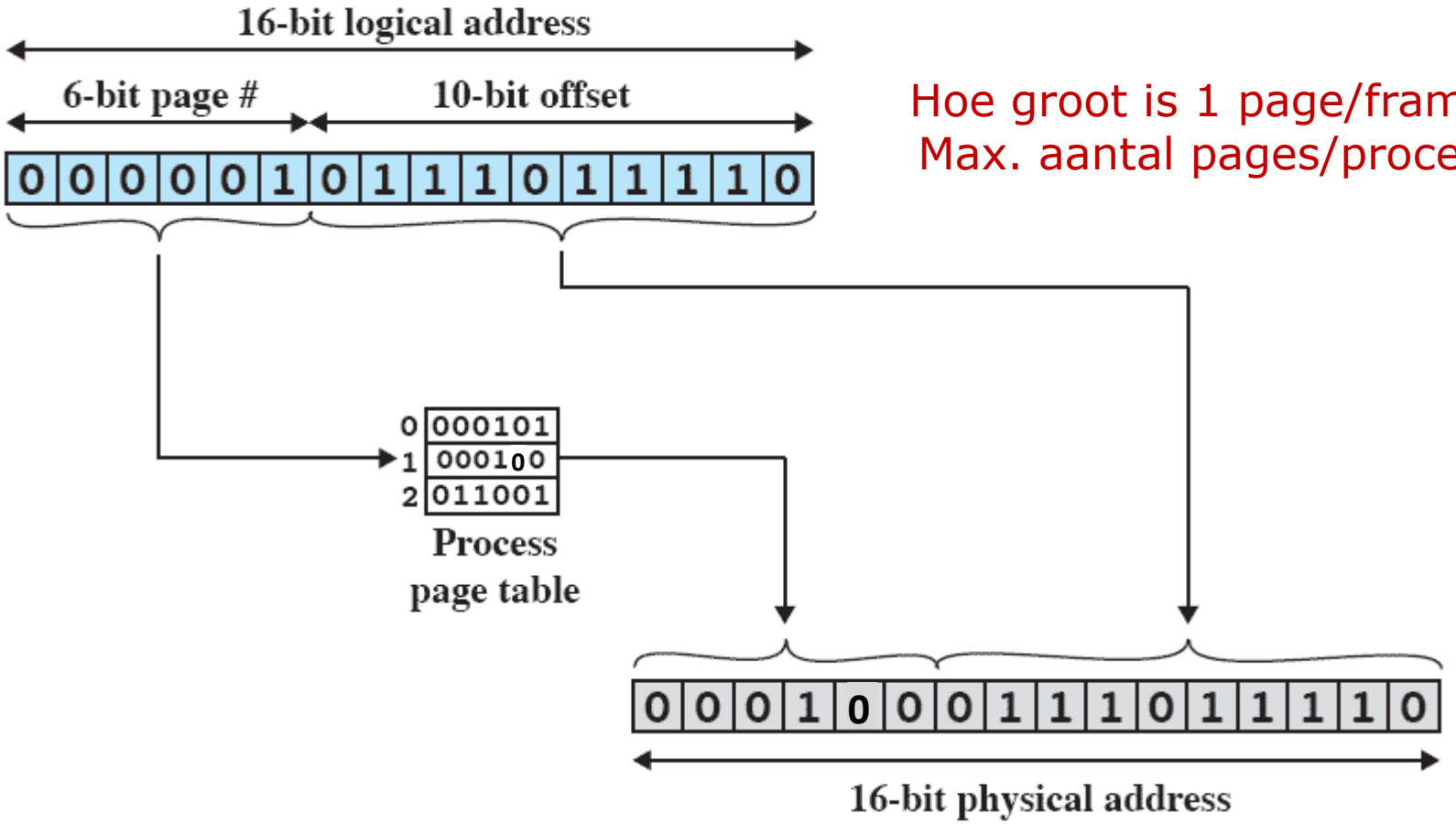

Frame number
Main memory
0 A.0
1 A.1
2 A.2
3 A.3
4 D.0
5 D.1
6 D.2
7 C.0
8 C.1
9 C.2
10 C.3
11 D.3
12 D.4
13
14
0 0
1 1
2 2
3 3
Process A page table
0 —
1 —
2 —
Process B page table
0 7
1 8
2 9
3 10
Process C page table
0 4
1 5
2 6
3 11
4 12
Process D page table
13
14
Free frame list

Paging

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



Paging



Hoe groot is 1 page/frame?
Max. aantal pages/proces?

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

Paging

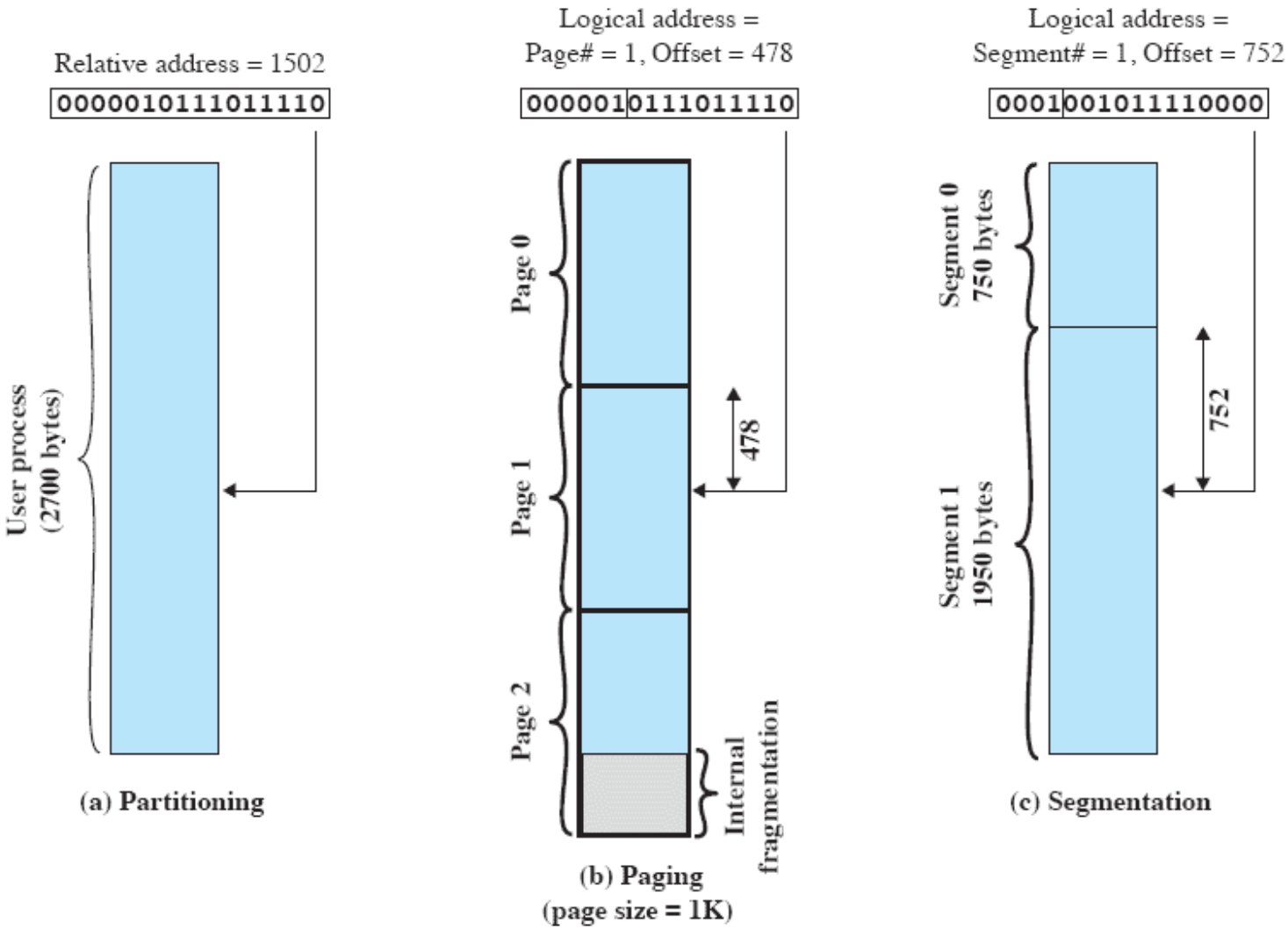
- Oefening: 8 bit pages
  - logisch adres: 1101 1100 1101 1000
  - page-table:

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

  - Fysisch adres?

OPDRACHT

Logische en fysieke adressen

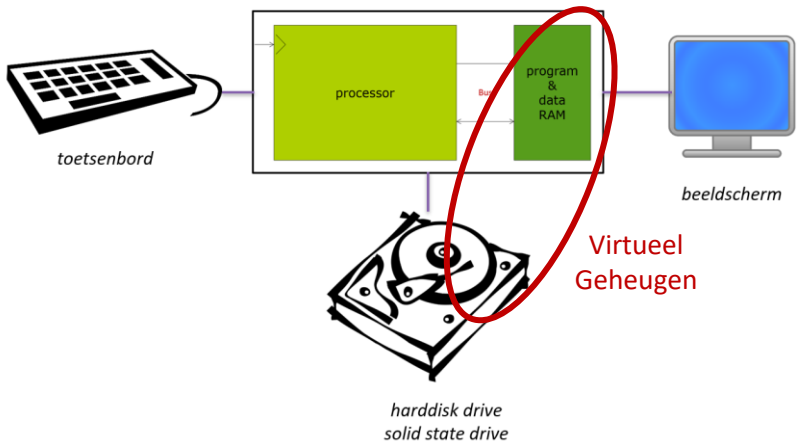


Inhoud

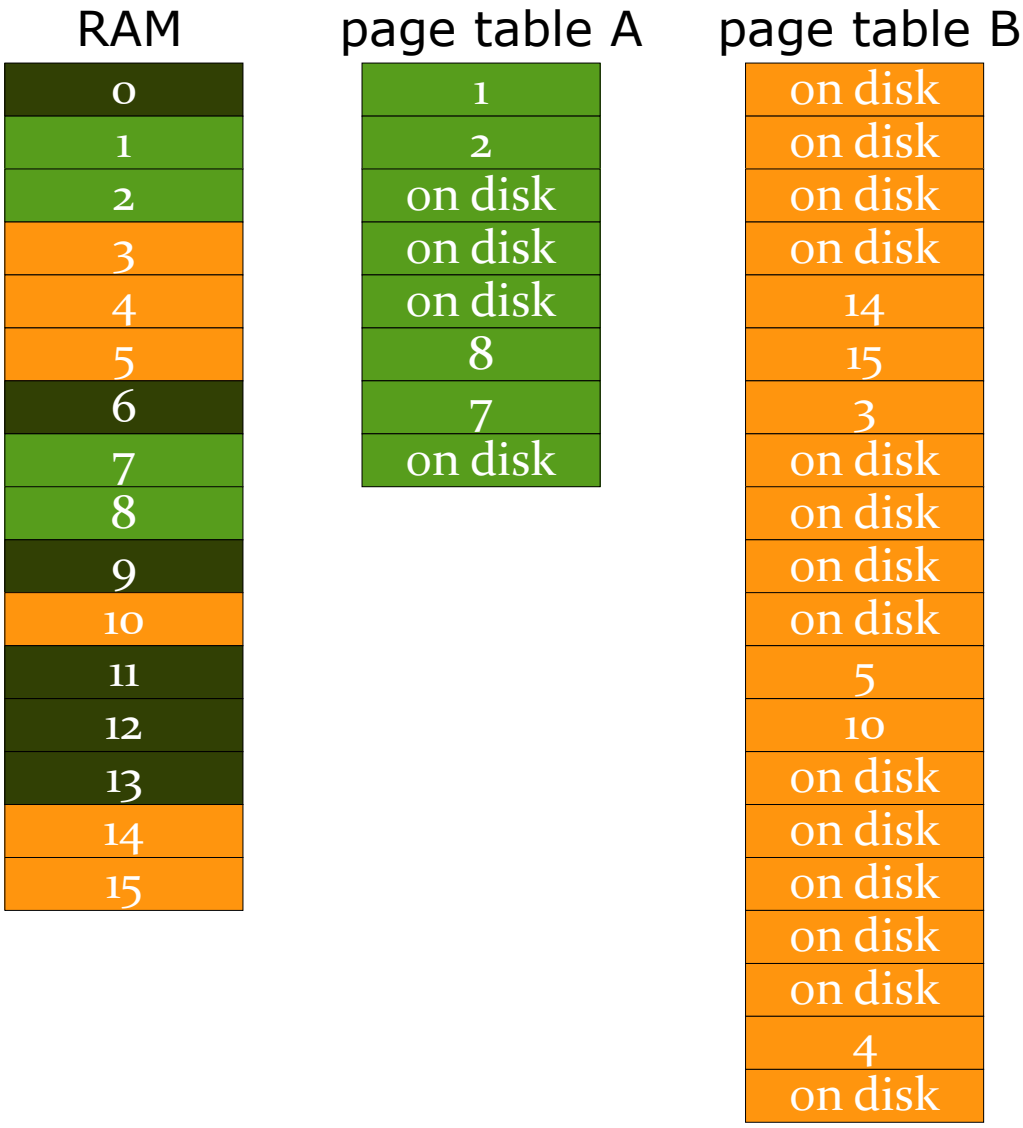
- Laden van OS
- Laden van processen
- Linken
- Call stack
- Partitioning
- Segmenting
- Paging
- Virtueel geheugen
- Herhalingsvragen

Virtueel geheugen

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



Virtueel geheugen



Virtueel geheugen

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

Virtueel geheugen

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

Ubuntu pagesize

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

Inhoud

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

Herhalingsvragen

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

Herhalingsvragen

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