Speichermanagement 2: *Virtualisierung*

Paging

Technische Grundlagen der Informatik

Automation Systems Group E183-1
Institute of Computer Aided Automation
Vienna University of Technology
email: tgi@auto.tuwien.ac.at

Recap

- Befehlssatz
  - Erweitern durch Interpreter
  - In Hardware implementieren (CISC)
- Pipelining
  - Erhöht den Durchsatz um Faktor k (optimal)
  - Erweiterung der Architektur notwendig
- Caching
  - Beschleunigt den Speicherzugriff durch mehrere Ebenen
  - Verschiedene Strategien möglich
    - Direct mapped, n-way set associative, fully associative

Speicherverwaltung

Speicherverwaltung

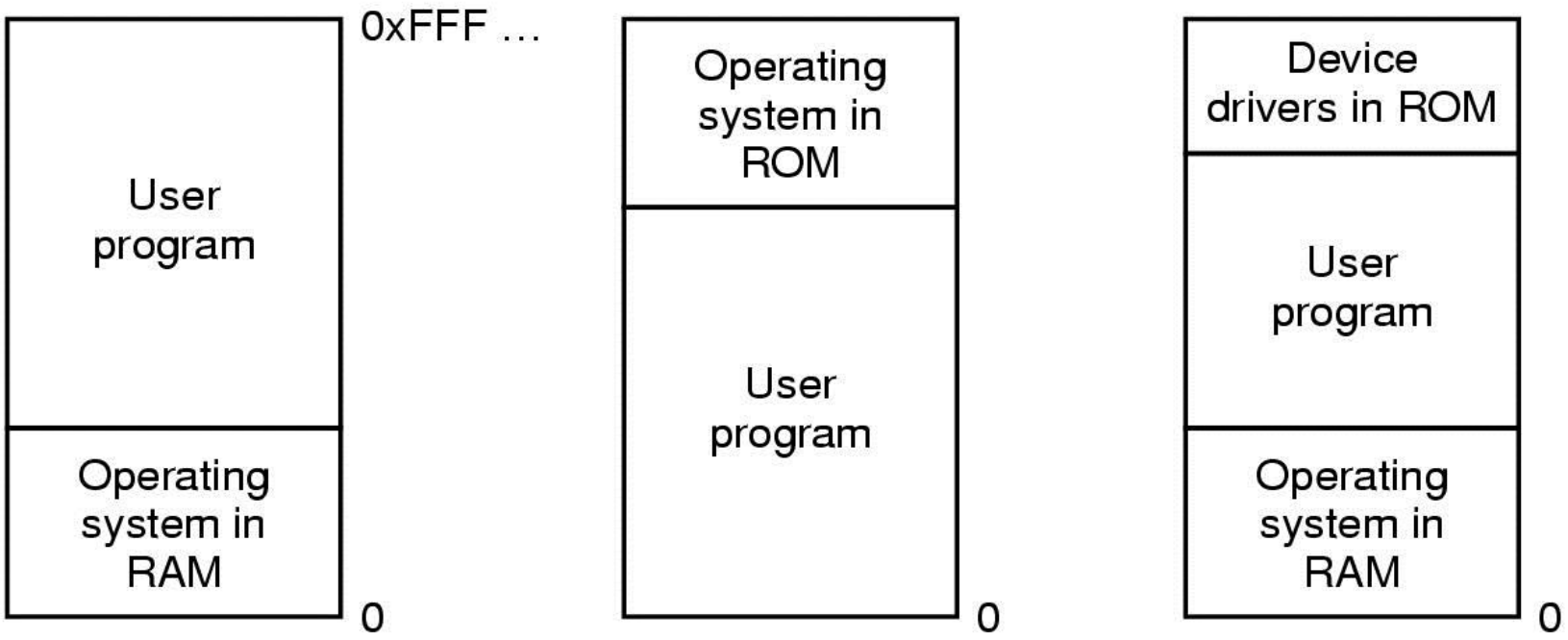
- Ideal
  - Speicher ist groß
  - Speicher ist schnell (niedrige Zugriffszeiten)
  - Speicher füllt den maximal adressierbaren Bereich

- Praxis
  - Speicherhierarchie
    - schneller Cache
    - mittlerer Hauptspeicher
    - langsamer Plattenspeicher

  - physikalischer Speicherplatz muss unter Prozessen aufgeteilt werden

Einfache Speicherverwaltung

- Nur ein Prozess läuft im Speicher
- Betriebssystem und Gerätetreiber resident oder im ROM



Methoden

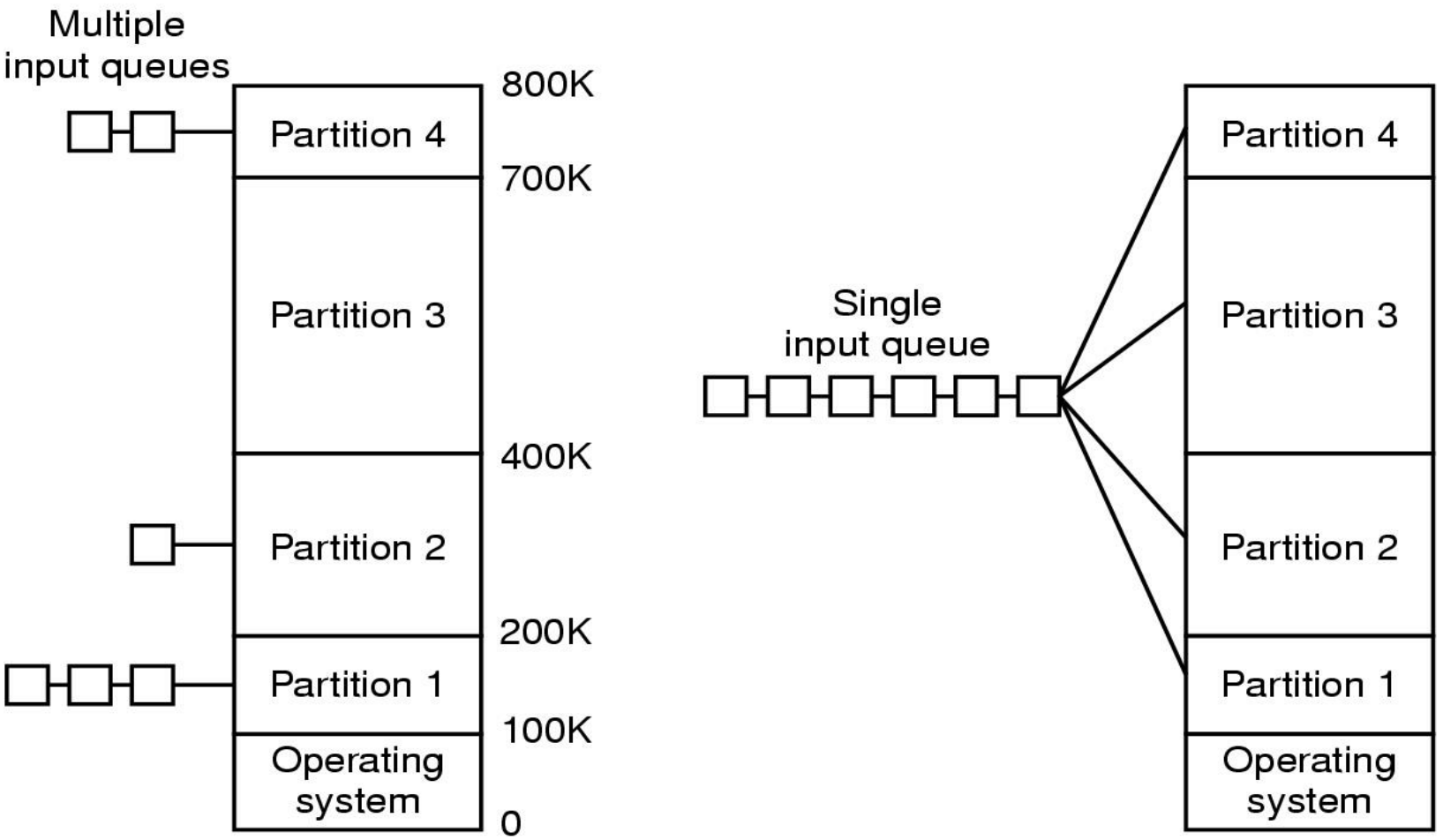
- Keine Speicherverwaltung (nur physikalische Adresszuordnung)
  - Keine parallele Verarbeitung möglich
  - Bei 32 Bit nicht automatisch 4GB vorhanden

- Swapping (Roll-in / Roll-out) Achtung Swapping!=Swap Datei
  - Phys. Speicher ist in fixe Partitionen aufgeteilt
  - Speicher ist in variable Partitionen aufgeteilt

- Probleme
  - wenn ein kleines Programm eine große Partition erhält, wird Speicher verschwendet

- Lösung
  - Warteschlangen (Queues) für unterschiedliche Partitionen
  - Verwenden einer Queue, und der größte Prozess wird gewählt (kleine Prozesse dürfen nicht „verhungern“

Fixe Partitionen



Relocation und Schutz

- Programmierer weiß nicht, wo Programm geladen wird
  - Adressen von Variablen und Funktionen müssen relativ sein
  - Wie verhindert man, dass ein Programm Daten eines anderen verändert?
- Relocation (= Anpassen von Adressen) zur Ladezeit durchführen
  - Meta-Information notwendig
  - Schutz noch nicht gegeben
- Basis und Limitregister
  - speichern Anfang und Länge eines Speichersegments
  - Zugriffe außerhalb des erlaubten Bereichs werden abgefangen

Virtueller Speicher

- Was passiert wenn ein Programm zu groß für den Speicher ist?
- Was, wenn man nur Teile laden will (besseres multi-programming)?

- Physikalischer Speicher wird in Bereiche (Segmente, Frames) zerlegt
- Programm wird in Bereiche (Segmente, Pages) zerlegt

- Immer nur ein Teil (die gerade verwendeten Bereiche) wird geladen
- Mapping zwischen den Teilen des Programms und dem physikalischen Speicher verwaltet das Betriebssystem

Virtuelle Speicherverwaltung

- Das Betriebssystem stellt jedem Prozess exklusiv einen riesigen Speicherbereich zur Verfügung
  - üblicherweise, das Maximum des adressierbaren Bereichs
  - bei 32-bit Architektur, d.h., 32-bit Adressen → 4 Gibi (-Byte)
  - dieser Speicher ist allerdings nicht wirklich vorhanden, also nur virtuell
  - ➢ virtuelle Adressen

- physikalischer Speicher ist begrenzt und muss unter allen Prozessen aufgeteilt werden
  - ➢ physikalische Adressen

Speicherverwaltung

Lösung des Problems

- Aufteilen des virtuellen Adressraums in kleinere Stücke
- diese Stücke können nicht mehr weiter unterteilt werden, und werden als ganzes in den physikalischen Teil geladen

- 2 Probleme
  1. Wie unterteile ich den virtuellen Adressraum?
  2. Wie bekomme ich die richtige physikalische Adresse, wenn ich eine virtuelle Adresse gegeben habe?

Speicherverwaltung

1. Problem: Aufteilung des virtuellen Adressraums

- 2 Möglichkeiten

1. Aufteilung in unterschiedlich große Teile, die direkt auf Teile des Programms abgebildet werden können (z.B., Codebereich, Stack)
   - Aufteilung erfolgt nicht transparent
   - Teile (segmente) sind nicht gleich groß
   - ➢ Segmentierung

2. Aufteilung in gleich große Teile, die vom Programm unabhängig sind
   - Aufteilung erfolgt transparent
   - Teile (pages, frames) sind gleich groß
   - ➢ Paging

Speicherverwaltung

2. Problem: Umsetzen einer virtuellen Adresse in die passende physikalische Adresse

- jede virtuelle Adresse liegt genau in einem Segment oder in einer Page
- virtuelle Adresse kann angegeben werden als
  - Start-Adresse des entsprechenden Segments (der entsprechenden Page) *plus* ein Offset (Abstand zur Start-Adresse)

- Beispiel mit Paging
  - Gegeben ist eine virtuelle Adresse = 0x4321 $(17185)_{10}$
  - Pages sind 0x1000 $(4096)_{10}$ groß
  - wie lautet die Start-Adresse und der Offset?

Speicherverwaltung

- Beispiel mit Paging (Lösung)

1. Page-Nummer berechnen

   $0x4321 \ / \ 0x1000 = 0x4 \ (4)_{10}$

2. Start-Adresse berechnen

   $0x1000 * 0x4 = 0x4000 \ (16384)_{10}$

3. Offset berechnen

   $0x4321 - 0x4000 = 0x321 \ (801)_{10}$

$$0x4321 = \underbrace{(0x1000 * 0x4)}_{\text{Start-Adresse}} + \underbrace{0x321}_{\text{Offset}}$$

Speicherverwaltung

- Was bringt diese Darstellung als Start-Adresse und Offset?

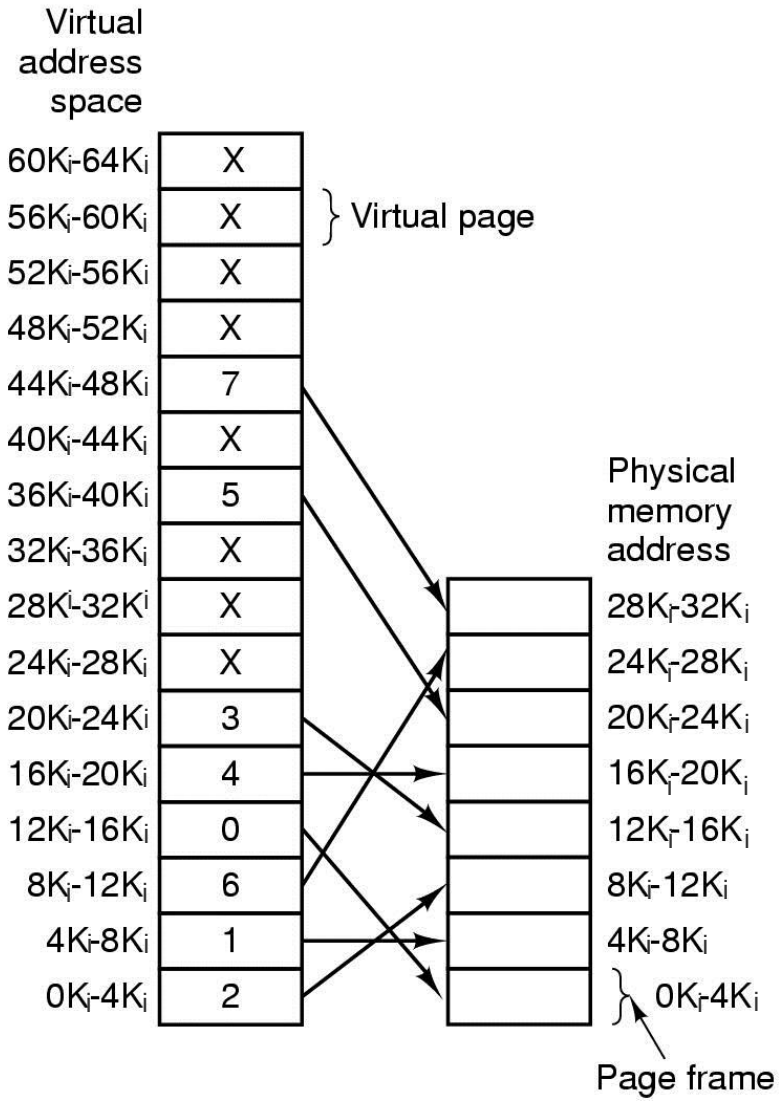
  Wenn ein Segment (oder Page) in den physikalischen Speicher geladen ist (beginnend bei Adresse A), dann muss nur die virtuelle Start-Adresse durch die physikalische Start-Adresse A ersetzte werden, um die Umsetzung zu erledigen.

- Vorteil
  - alle Adressen eines Segments (einer Page) können mit einer Operation umgewandelt werden
- Fragen
  - Wie bekomme ich die physikalische Start-Adresse, wenn ich die virtuelle Start-Adresse habe?
  - Wer genau macht diese Umsetzung?

Speicherverwaltung

- Paging
  - virtuelle Adressen sind in Pages unterteilt
  - typischerweise zwischen 512 Bytes und 4 KiB groß
  - physikalischer Speicher ist in gleich große Page Frames (oder Frames) unterteilt

Speicherverwaltung



Speicherverwaltung

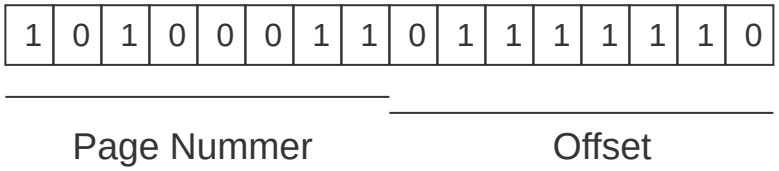
- Pages haben eine Zweierpotenz Größe

- Grund
  - einfache Berechnung der Page Nummer
  - virtuelle Adresse zerfällt in Page Nummer und Offset
  - Page Nummer einfach aus Adresse ablesen

Speicherverwaltung

- Beispiel
  - 16-bit virtuelle Adresse, Page ist 0x100 = 256 (2^8) Bytes groß
  - letzten 8 Bits der Adresse sind Offset
  - ersten 8 Bits der Adresse sind Page Nummer

  - virtuelle Adresse 0xA37E



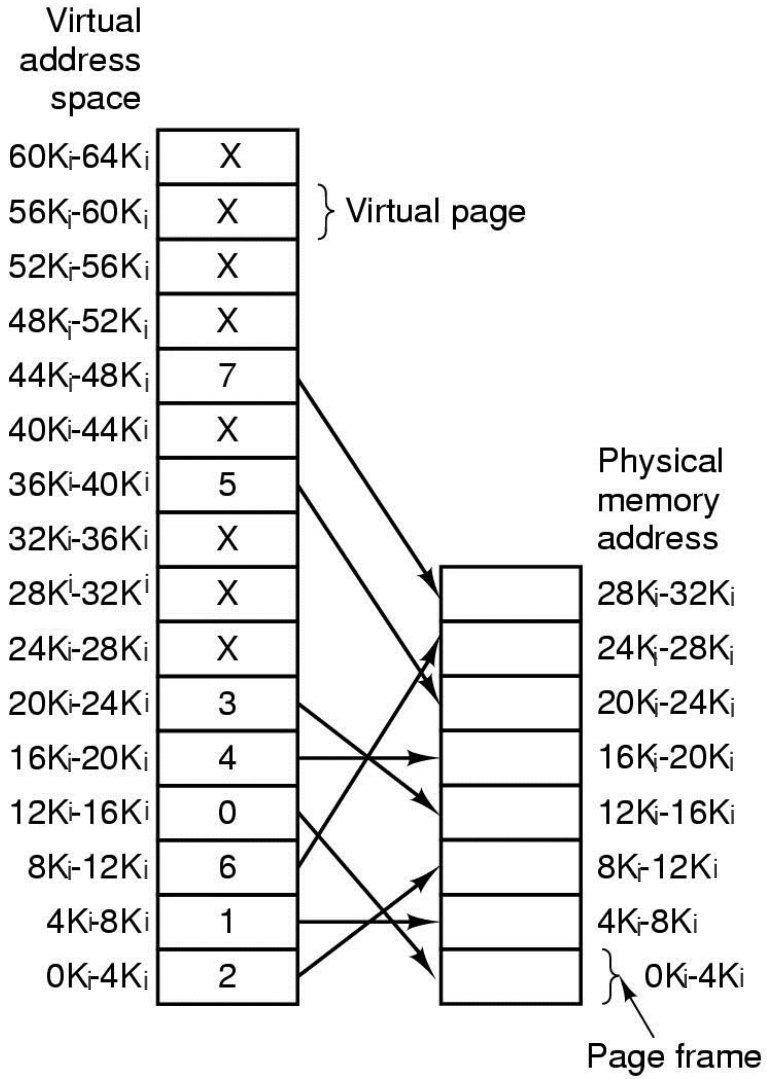
  - Page Nummer = 0xA3, Offset = 0x7E

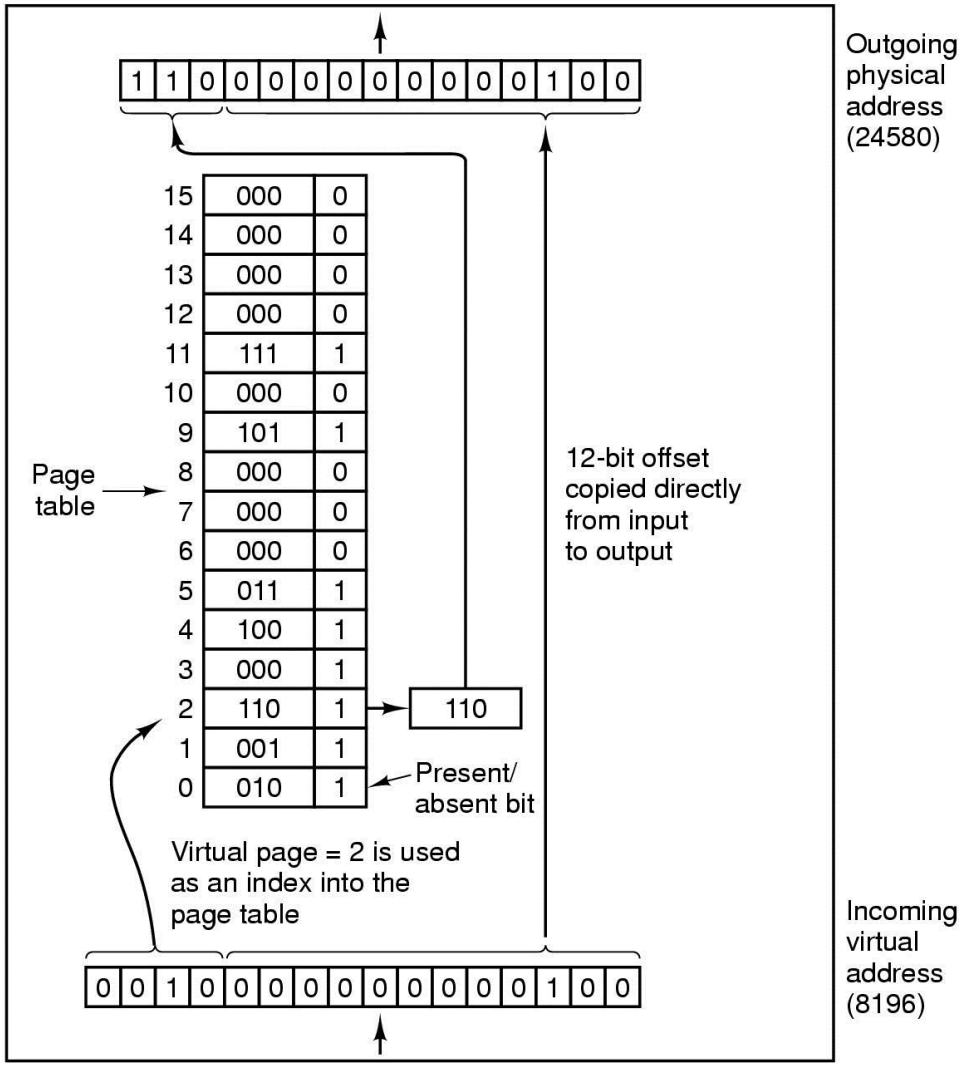
Speicherverwaltung

- Frage
  - Wie bekomme ich die physikalische Start-Adresse, wenn ich die virtuelle Start-Adresse habe?

- Page Table
  - virtuelle Start-Adresse wird nicht benötigt
  - Page Nummer reicht aus
  - Page Table speichert für jede Page, wo der entsprechende Page Frame im Speicher liegt
  - außerdem, ein Bit (present bit), welches angibt, ob die Page überhaupt geladen ist
  - falls auf eine Page zugegriffen wird, die nicht im physikalischen Speicher liegt → *page fault*

Speicherverwaltung

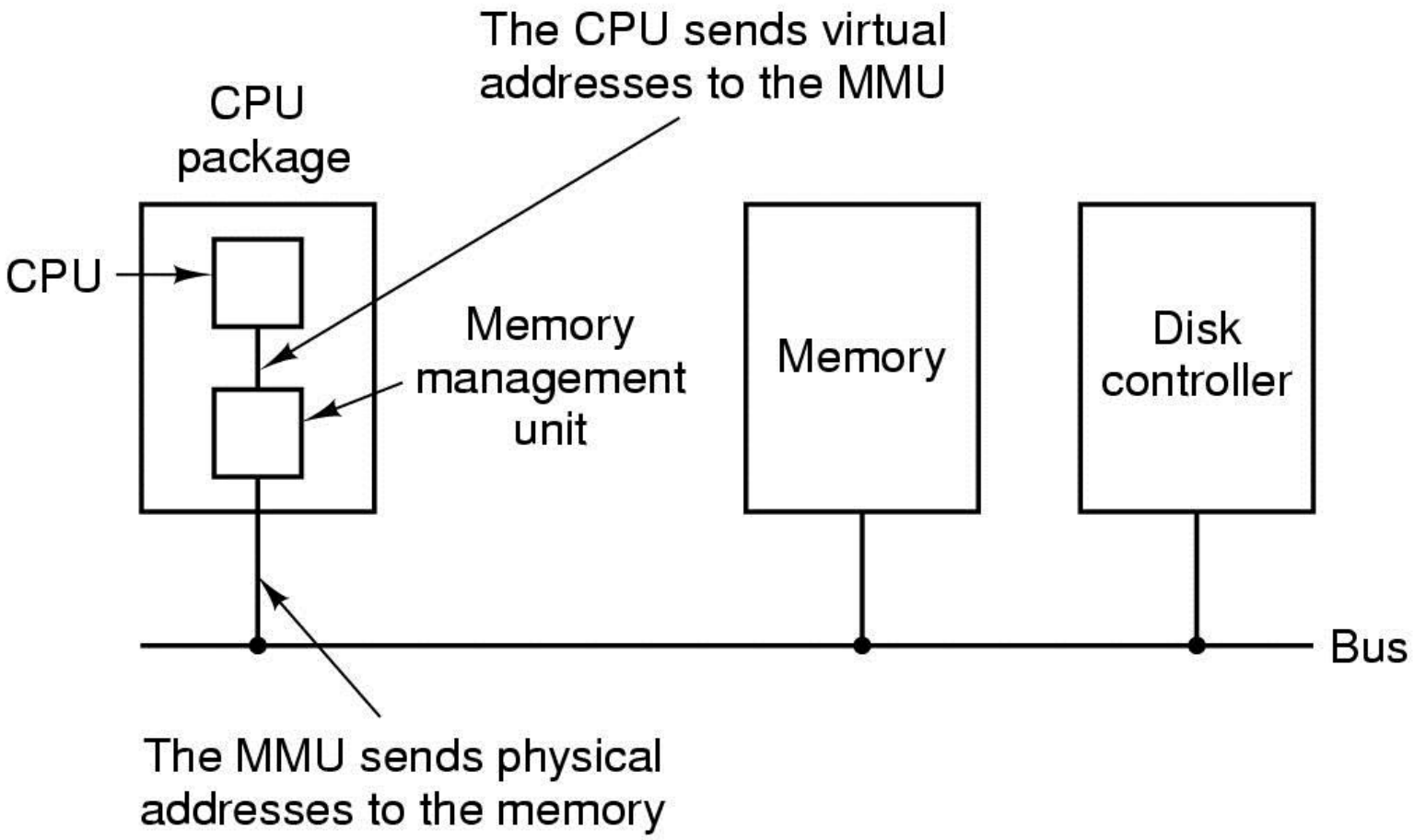




Speicherverwaltung

- Frage
  - Wer macht die Umsetzung?
- Memory Management Unit (MMU)
  - Hardware Baustein, der vom Betriebssystem entsprechend geladen wird
  - eine Page Table pro Prozess ist notwendig
  - muss bei jedem Context Switch passend geladen werden
- Page Tables brauchen einen Eintrag pro Page
  - kann sehr viel werden
  - 32-bit Adressraum mit 4 KibiByte Pages ergibt 2^{20} Page Table Einträge
  - Teile müssen in den Hauptspeicher ausgelagert werden
  - Page Table Hierarchie

Speicherverwaltung



Speicherverwaltung

- Beispiel
  - Umsetzen einer virtuellen Adresse in die entsprechende physikalische Adresse
  - gegeben ist Page Table, Page Größe, und virtuelle Adresse
  - gefragt ist die physikalische Adresse

Speicherverwaltung

- Angabe
  - Page Größe ist 4 KiB, virtuelle Adressen haben 32-bit, physikalische Adressen haben 24-bit

Page Nummer	Frame Nummer
0x00	0x7C
0x01	0x8A
0x02	(not present)
……	…..

virtuelle Adressen a) 0x00001BCD

b) 0x00002FFE

Speicherverwaltung

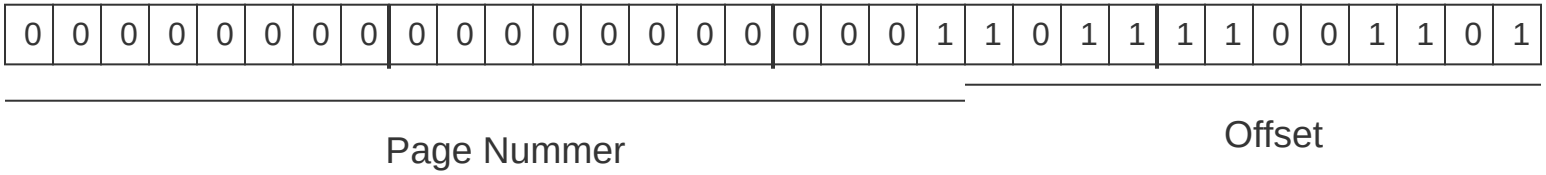
- 1. Schritt

  Zerlegen der virtuellen Adresse in Page Nummer und Offset

  4 KiB Pages bedeutet 12 Bits Offset (weil 4096 = 2^{12}),

  daher ist die Page Nummer 20 Bit groß

  a) 0x 00 00 1B CD



  Page Nummer = 0x01

  Offset = 0xBCD

Speicherverwaltung

- 2. Schritt

Page Nummer = 0x01

Offset = 0xBCD

Nachschlagen der entsprechenden Frame Nummer in der Page Table

Page Nummer	Frame Nummer
0x00	0x7C
0x01	0x8A
0x02	(not present)
……	…….

Frame Nummer = 0x8A

Speicherverwaltung

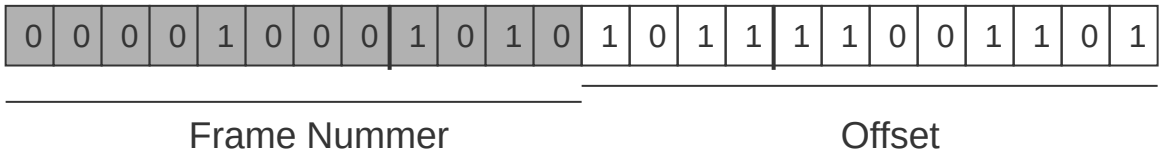
- 3. Schritt

Page Nummer = 0x01

Offset = 0xBCD

Frame Nummer = 0x8A

Zusammensetzen von Frame Nummer und Offset (24 Bit)



physikalische Adresse = 0x 08 AB CD

Speicherverwaltung

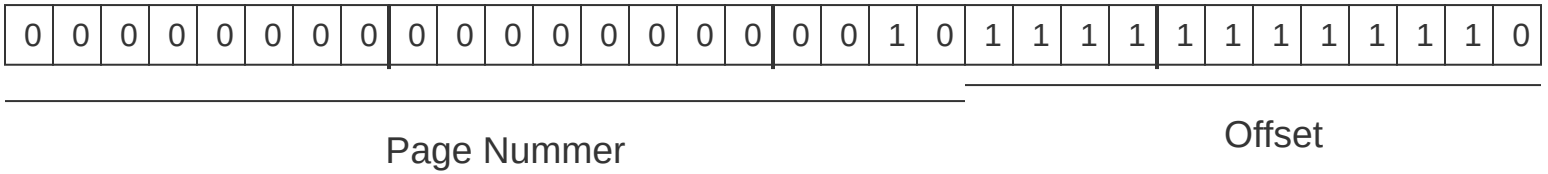
- 1. Schritt

  Zerlegen der virtuellen Adresse in Page Nummer und Offset

  4 KiB Pages bedeuten 12 Bits Offset (4096 = 2^{12}),

  daher ist die Page Nummer 20 Bit groß

  b) 0x 00 00 2F FE



  Page Nummer = 0x02

  Offset = 0xFFE

Speicherverwaltung

- 2. Schritt

  Page Nummer = 0x02

  Offset = 0xFFE

  Nachschlagen der entsprechenden Frame Nummer in der Page Table

Page Nummer	Frame Nummer
0x00	0x7C
0x01	0x8A
0x02	(not present)
…….	…….

  Frame Nummer = (not present) → *page fault*

Speicherverwaltung

- Was passiert bei einem *page fault*?
  - Betriebssystem lädt Page in ein freies Frame im Speicher
  - passt Page Table entsprechend an
  - setzt danach Anwendung fort

- Was, wenn alle Frames belegt sind?
  - ein existierendes Frame muss überschrieben werden
  - veränderte Frames müssten zurückgeschrieben werden
  - daher, besser nicht modifizierte Frames nehmen

- Entscheidung
  - ➢ Page replacement algorithm

Speicherverwaltung

- Page replacement algorithm
  - Ziel
    - Minimiere die Anzahl der page faults
  - Optimal
    - ersetze die Page, die am weitesten in der Zukunft gebraucht wird
    - unmöglich, aber gut für Vergleiche
  - FIFO
    - first-in, first-out
    - ersetze älteste Page
  - LRU
    - least-recently-used
    - ersetze die am längsten nicht gebrauchte Page

Speicherverwaltung

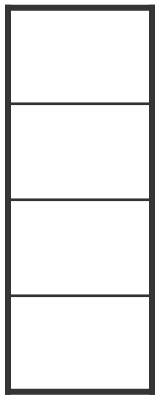
- Beispiel
  - gegeben ist die Anzahl der Frames, die benutzt werden können,
    sowie eine Reihenfolge von Zugriffen (*reference string*) auf die Pages
  - gefragt sind die Entscheidungen des page replacement algorithm, d.h.,
    welche Pages befinden sich nach jedem Zugriff im Speicher (in den Frames)

- Angabe
  - zu verwenden ist LRU
  - 4 Frames sind verfügbar
  - reference string

    0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1

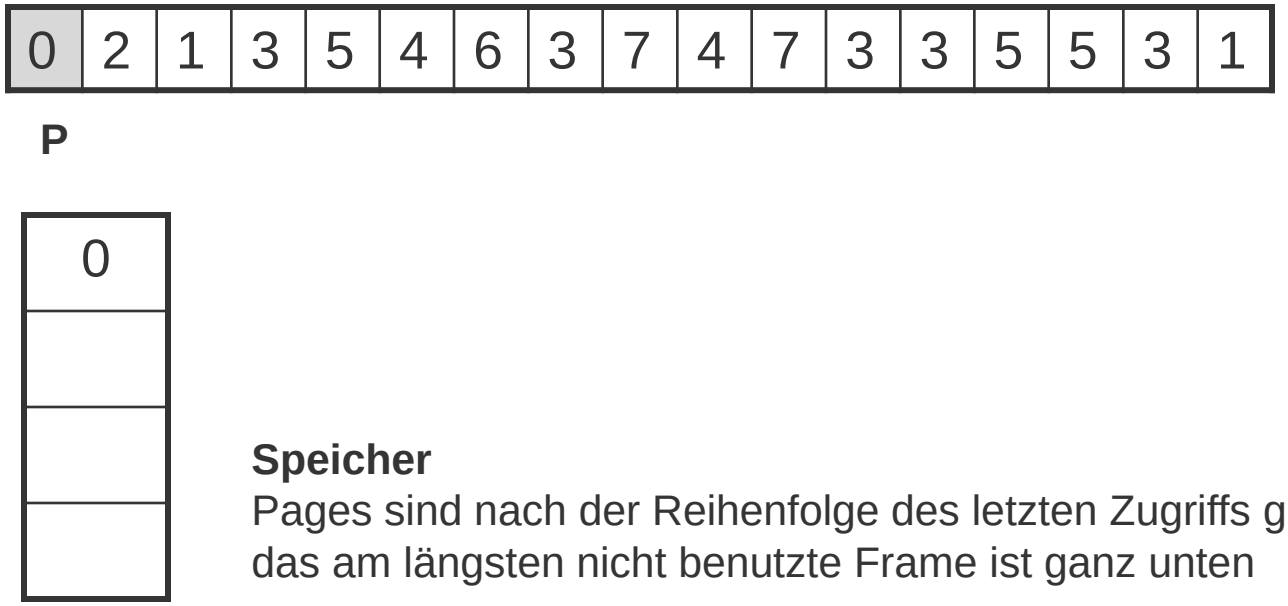
Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1



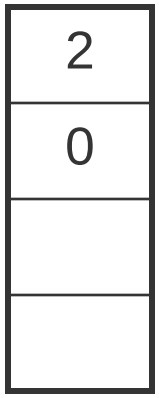
Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



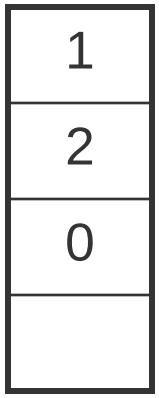
Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



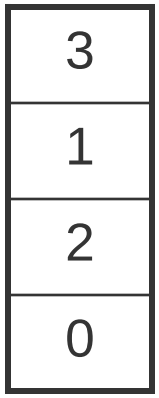
Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



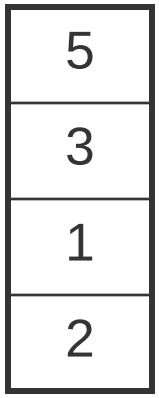
Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



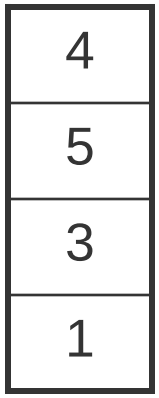
Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



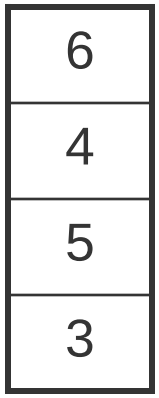
Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

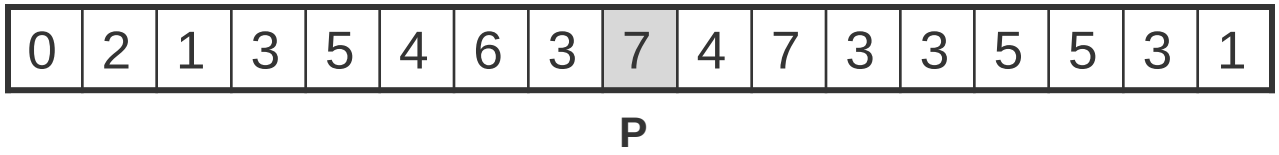
Speicherverwaltung

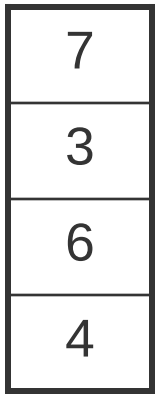
0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
6
4
5

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung





Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

4
7
3
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

7
4
3
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
7
4
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

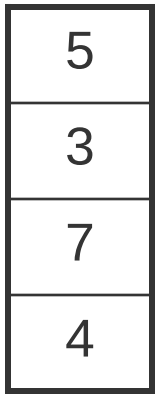
Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
7
4
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung



Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

5
3
7
4

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
5
7
4

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
																P

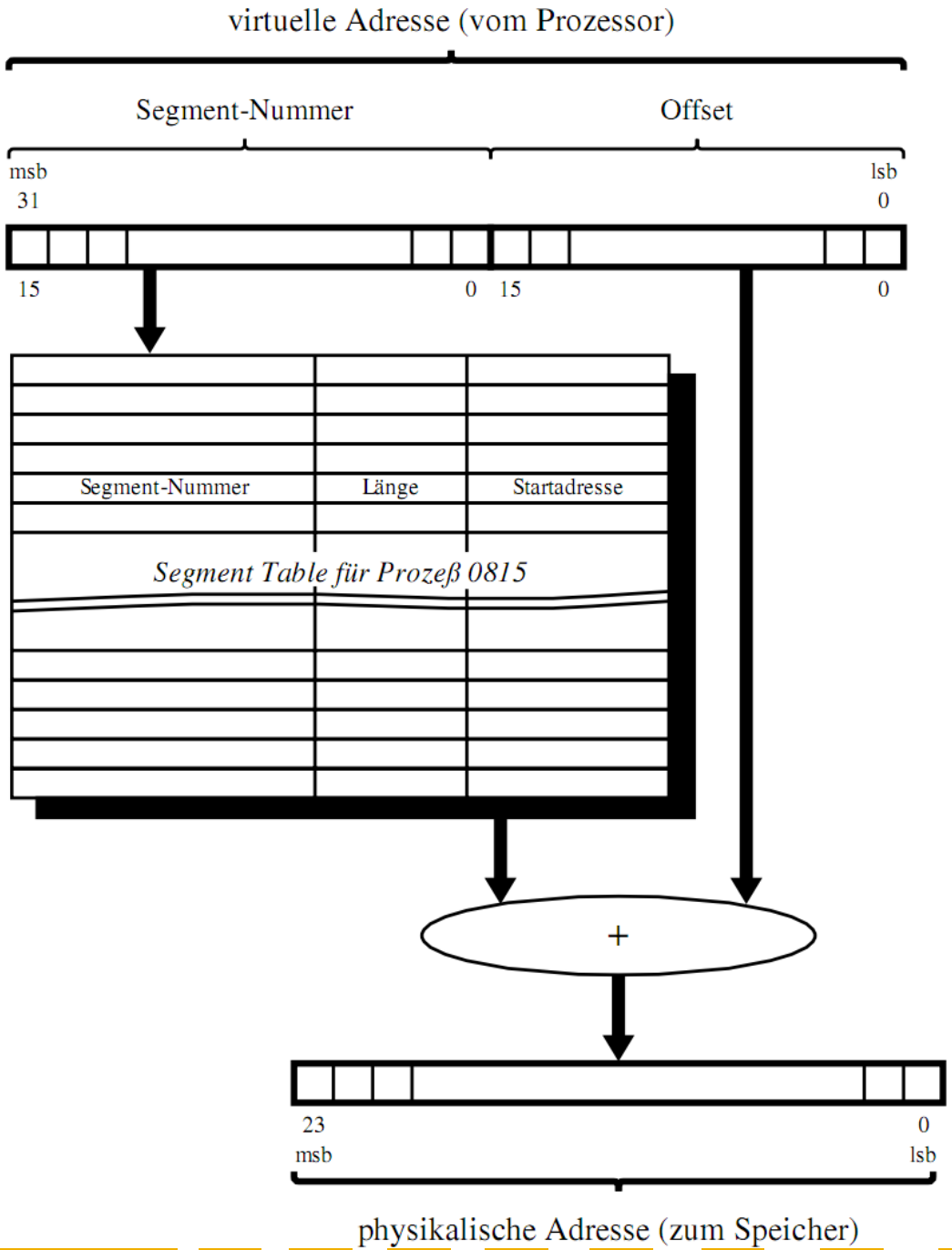
1
3
5
7

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Segmentierung

- Funktionsweise von Segmentierung
  - Ähnlich wie Paging
  - Segment-Tabelle verwaltet die Speichersegmente wie z.B.:
    - Segment-Nummer
    - Länge
    - Startadresse
    - *Zusatzinformation*

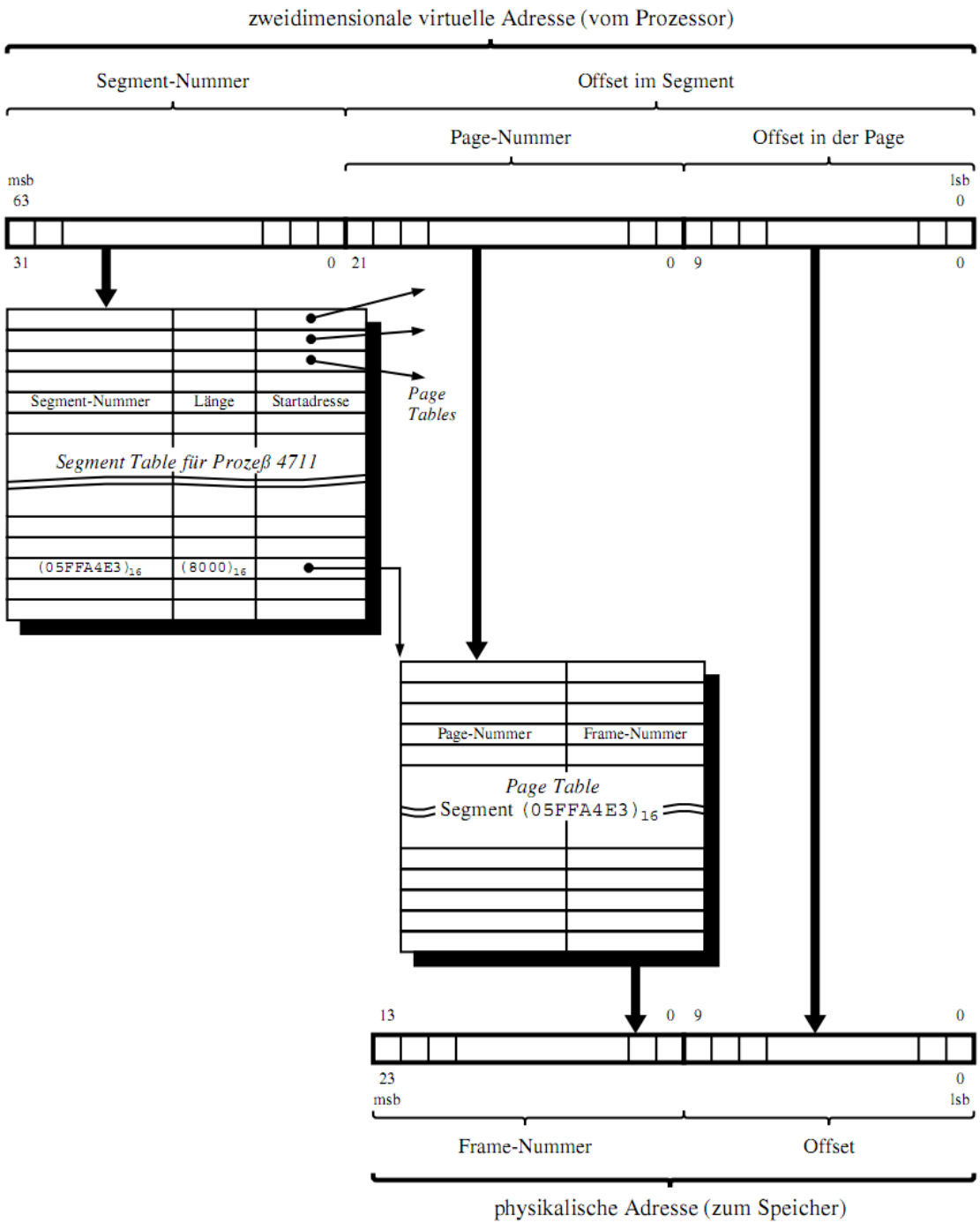
- Unterschied zum Paging
  - Segmente haben keine vorgegebene Länge
  - Startadresse und Offset werden addiert
  - Impliziter Speicherschutz


virtuelle Adresse (vom Prozessor)
Segment-Nummer
Offset
msb
31
lsb
0
15
0
15
0
Segment-Nummer
Länge
Startadresse
Segment Table für Prozeß 0815
+
23
msb
0
lsb
physikalische Adresse (zum Speicher)

Segmentierung + Paging

- Kombination von Paging und Segmentierung
  - Vorteile der Segmentierung und des Pagings
  - Großer Berechnungsaufwand
  - Kann viel Speicher verbrauchen (Page Table + Segment Table)

- Unterschied zum Paging / Segmentierung
  - Segmentnummer gibt Segment in der Segment Table an. Ausgelesen wird aber eine ganze Page Table
  - Page Table + Page Nummer aus virtueller Adresse ergibt Frame Nummer des Hautspeichers
  - Offset wird direkt aus der virt. Adresse genommen


zweidimensionale virtuelle Adresse (vom Prozessor)
Segment-Nummer
Offset im Segment
Page-Nummer
Offset in der Page
msb
63
lsb
0
31
0
21
0
9
0
Segment-Nummer
Länge
Startadresse
Page Tables
Segment Table für Prozeß 4711
(05FFA4E3)16
(8000)16
Page-Nummer
Frame-Nummer
Page Table
Segment (05FFA4E3)16
13
0
9
0
23
0
msb
lsb
Frame-Nummer
Offset
physikalische Adresse (zum Speicher)

Zusammenfassung

- Betriebssysteme verwalten
  - Prozesse
  - *Speicher*
  - Dateisystem
  - Eingabe und Ausgabe

- Speichermanagement
  - Speicherhierarchie
  - Caches
    - Typen von Caches
  - Hauptspeicher
    - virtuelle Speicherverwaltung

Zusammenfassung

- Virtuelle Speicherverwaltung
  - Segmentation
  - Paging
  - Adressumrechnungen
  - page faults
  - page replacement algorithms