Virtueller Speicher und Memory Management

Speicher-Paradigmen

- Programmierer
  - ein großer Adressraum
  - linear adressierbar
- Betriebssystem
  - eine Menge laufender Tasks / Prozesse
  - read-only Instruktionen
  - read-write Daten

Virtueller Speicher und Memory Management (cont.)

Aufgaben des Betriebssystems

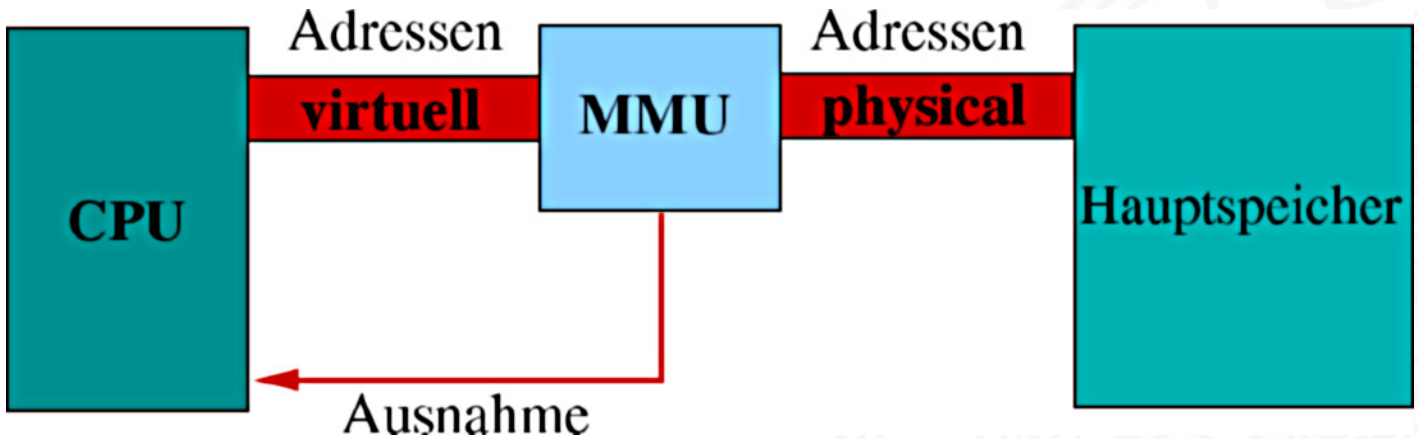
- effiziente Nutzung des Speichers gewährleisten
  - Prozess benötigt oft nicht alle Teile seines Adressraums
  - Speicherbedarf aller Prozesse kann größer als der verfügbare Hauptspeicher sein

  $\Rightarrow$ nicht benötigte Teile des Adressraums auf Hintergrundspeicher auslagern
- Speicherschutz
  - Prozesse haben nur Zugriff auf eigenen Daten

Virtueller Speicher und Memory Management (cont.)

Prinzip des virtuellen Speichers

- jeder Prozess besitzt seinen eigenen virtuellen Adressraum
- MMU – **M**emory **M**anagement **U**nit



- Umsetzung von virtuellen zu physikalischen Adressen, Programm-Relokation

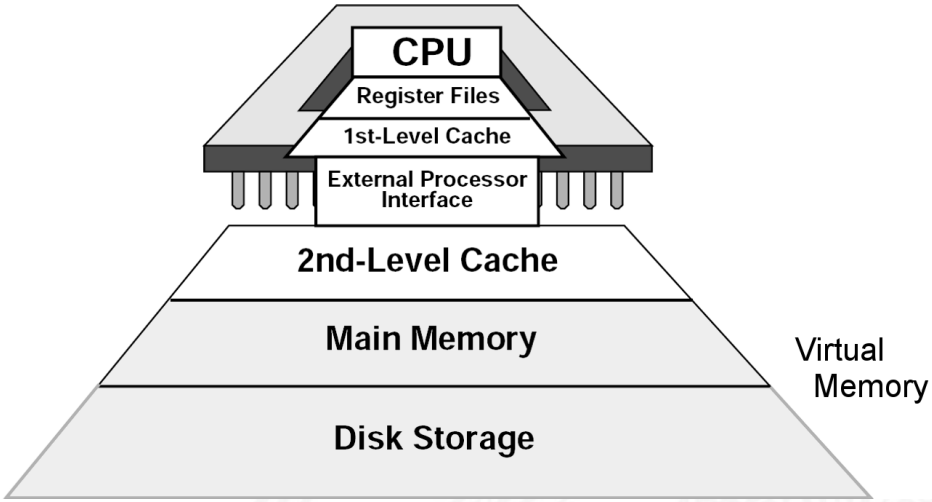
Virtueller Speicher und Memory Management (cont.)

- Umsetzungstabellen werden vom Betriebssystem erzeugt und aktualisiert
- wegen des Speicherbedarfs der Tabellen beziehen sich diese auf größere Speicherblöcke

- Umgesetzt wird nur die Anfangsadresse, der Offset innerhalb des Blocks bleibt unverändert
- Blöcke dieses virtuellen Adressraums können durch Betriebssystem ausgelagert werden
- Methoden zur Implementation virtuellen Speichers
  - *Segmentierung*
  - *Paging*, Speicherzuordnung durch *Seiten*
  - gemischte Verfahren (Standard bei: Desktops, Workstations...)

Virtueller Speicher und Memory Management (cont.)

- Hauptspeicher als Cache für den Plattenspeicher

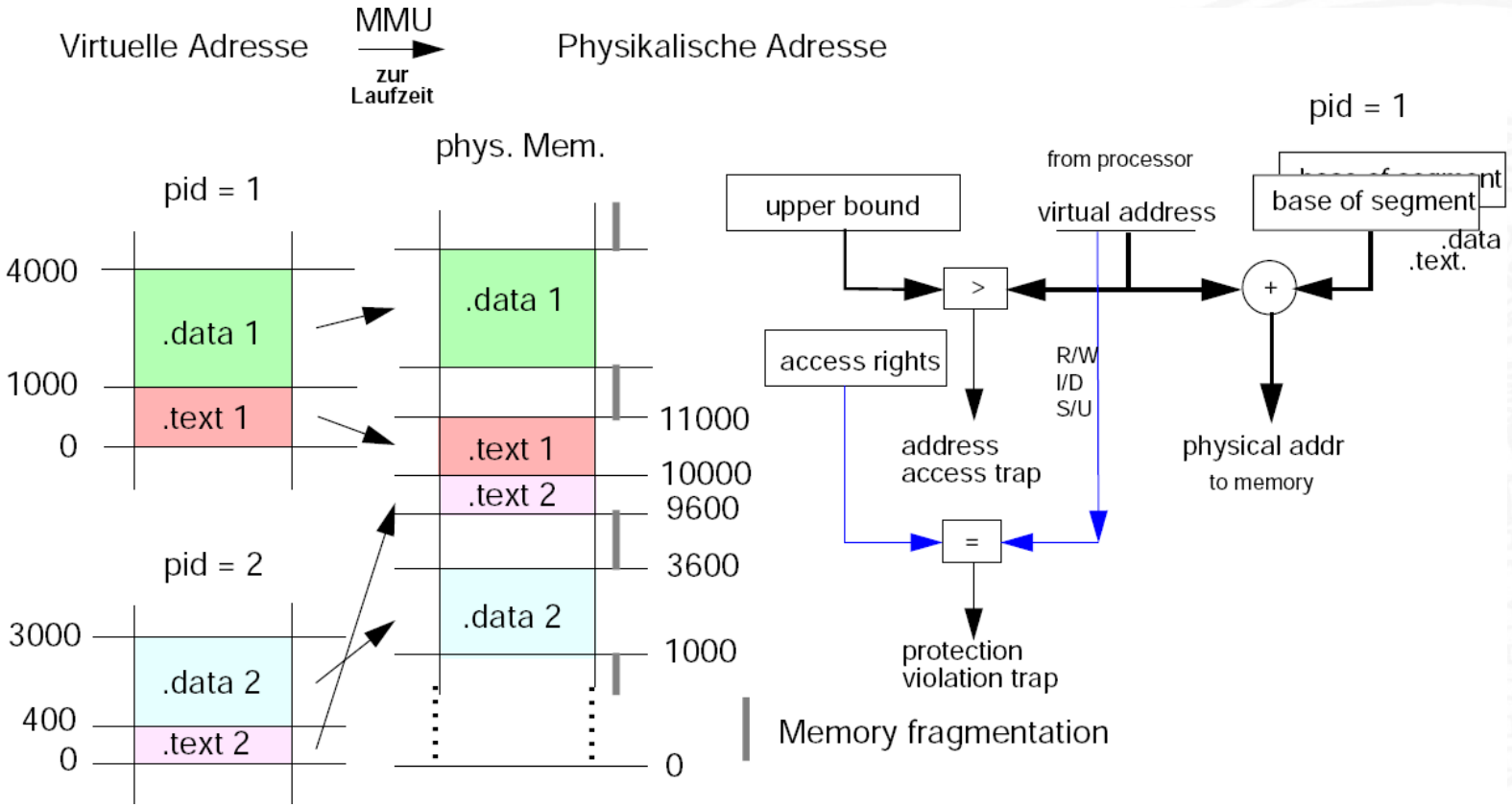


- Parameter der Speicherhierarchie

	1st-Level Cache	virtueller Speicher
Blockgröße	16-128 Byte	4-64 kByte
Hit-Dauer	1-2 Zyklen	40-100 Zyklen
Miss Penalty	8-100 Zyklen	70.000-6.000.000 Zyklen
Miss Rate	0,5-10 %	0,00001-0,001 %
Speichergröße	8-64 kByte	16-8192 MByte

Virtueller Speicher: Segmentierung

- Unterteilung des Adressraums in kontinuierliche Bereiche *variabler* Größe



Virtueller Speicher: Segmentierung (cont.)

- Idee: Trennung von Instruktionen, Daten und Stack

$\Rightarrow$ Abbildung von *Programmen* in den *Hauptspeicher*

$+$ Inhalt der Segmente: logisch zusammengehörige Daten

$+$ getrennte Zugriffsrechte, Speicherschutz

$+$ exakte Prüfung der Segmentgrenzen

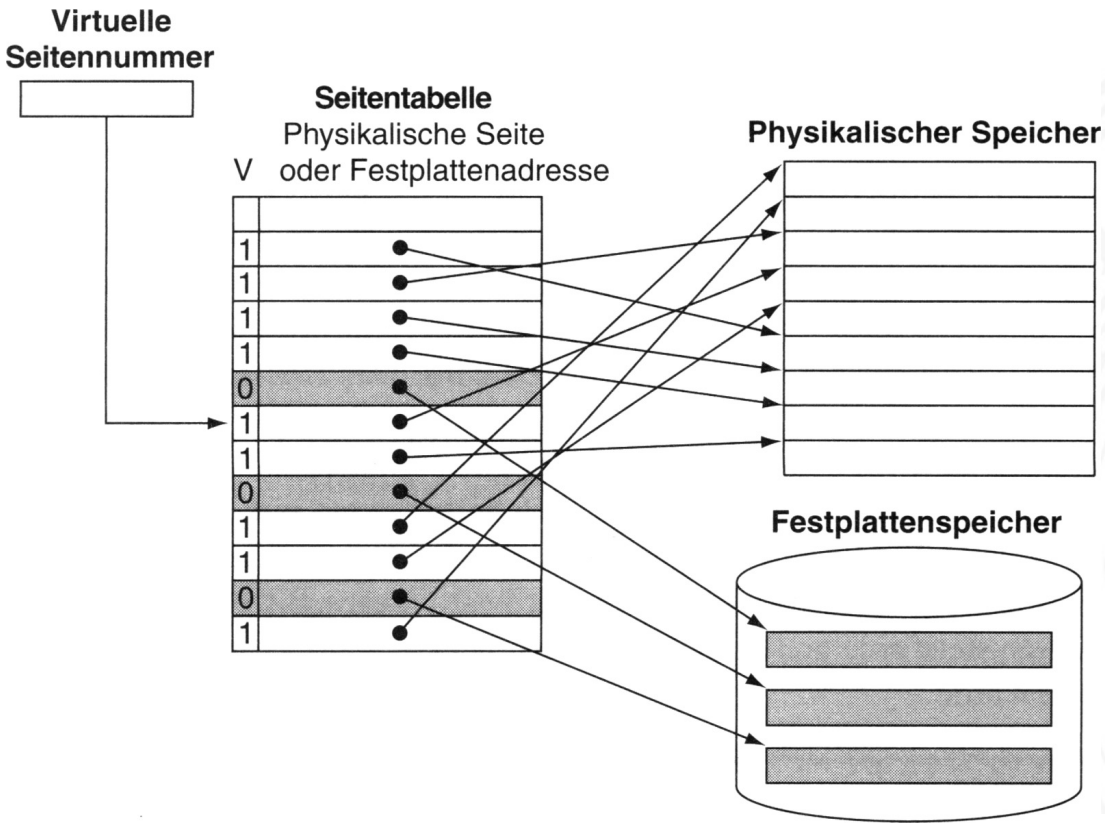
$-$ Segmente könne sehr groß werden

$-$ Ein- und Auslagern von Segmenten kann sehr lange dauern

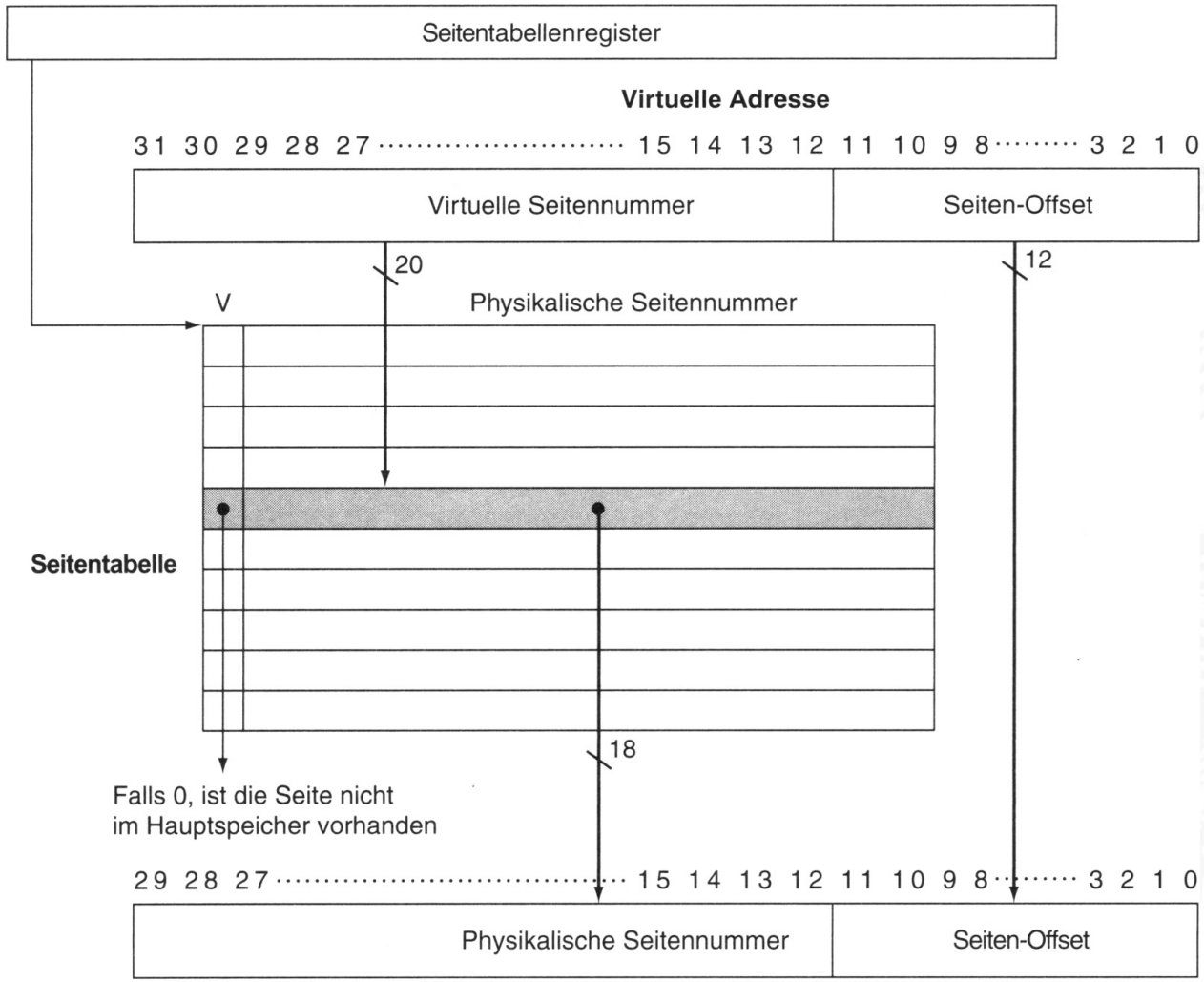
$-$ „Verschnitt“, **Memory Fragmentation**

Virtueller Speicher: Paging / Seitenadressierung

- Unterteilung des Adressraums in Blöcke *fester* Größe = Seiten
  Abbildung auf Hauptspeicherblöcke = Kacheln



Virtueller Speicher: Paging / Seitenadressierung (cont.)



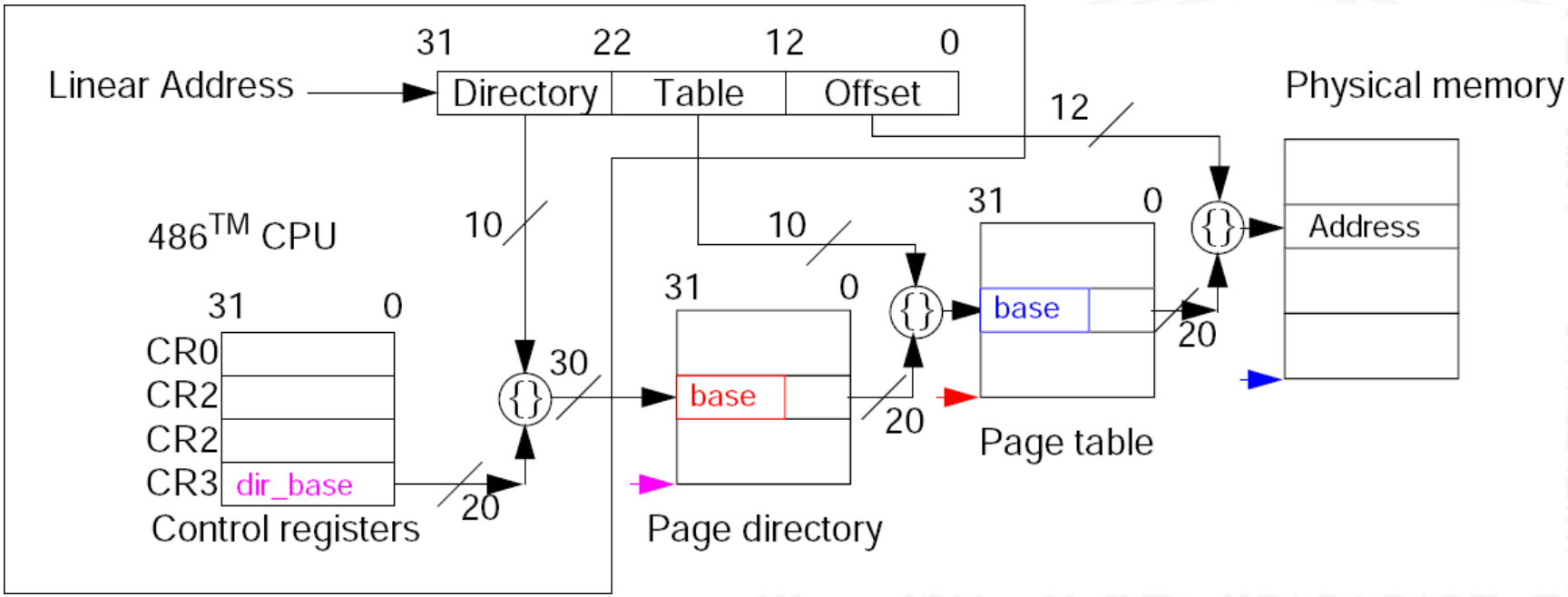
Virtueller Speicher: Paging / Seitenadressierung (cont.)

- ⇒ Abbildung von *Adressen* in den *virtuellen Speicher*
- \+ Programme können größer als der Hauptspeicher sein
- \+ Programme können an beliebige physikalischen Adressen geladen werden, unabhängig von der Aufteilung des physikalischen Speichers
- \+ feste Seitengröße: einfache Verwaltung in Hardware
- \+ Zugriffsrechte für jede Seite (read/write, User/Supervisor)
- ▶ große Miss-Penalty (Nachladen von der Platte)
  ⇒ Seiten sollten relativ groß sein: 4 oder 8 kByte
- – Speicherplatzbedarf der Seitentabelle
  viel virtueller Speicher, 4 kByte Seitengröße
  ⇒ große Pagetable

Virtueller Speicher: Paging / Seitenadressierung (cont.)

⇒ Hash-Verfahren (*inverted page tables*)

⇒ mehrstufige Verfahren

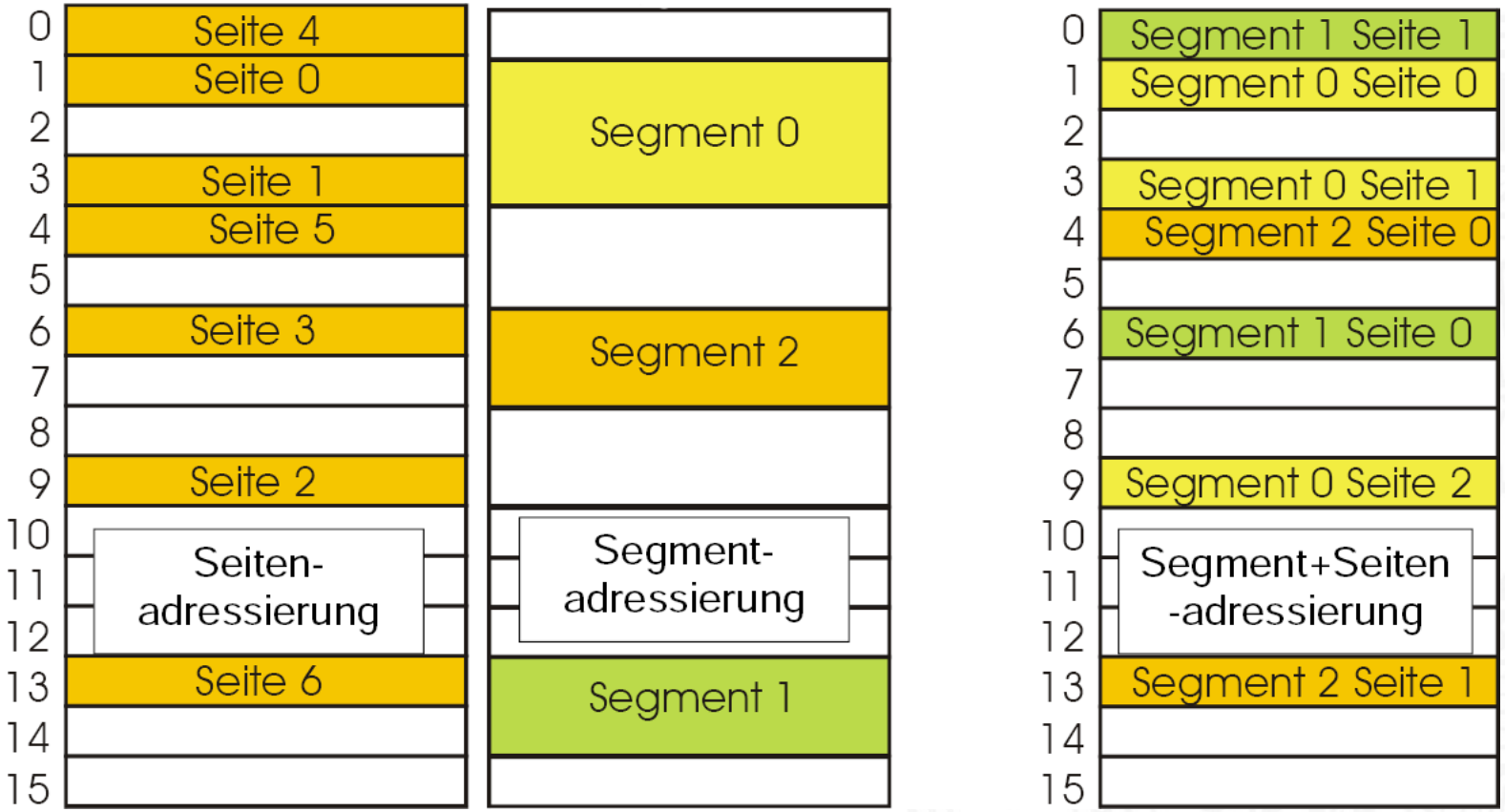


Virtueller Speicher: Paging / Seitenadressierung (cont.)

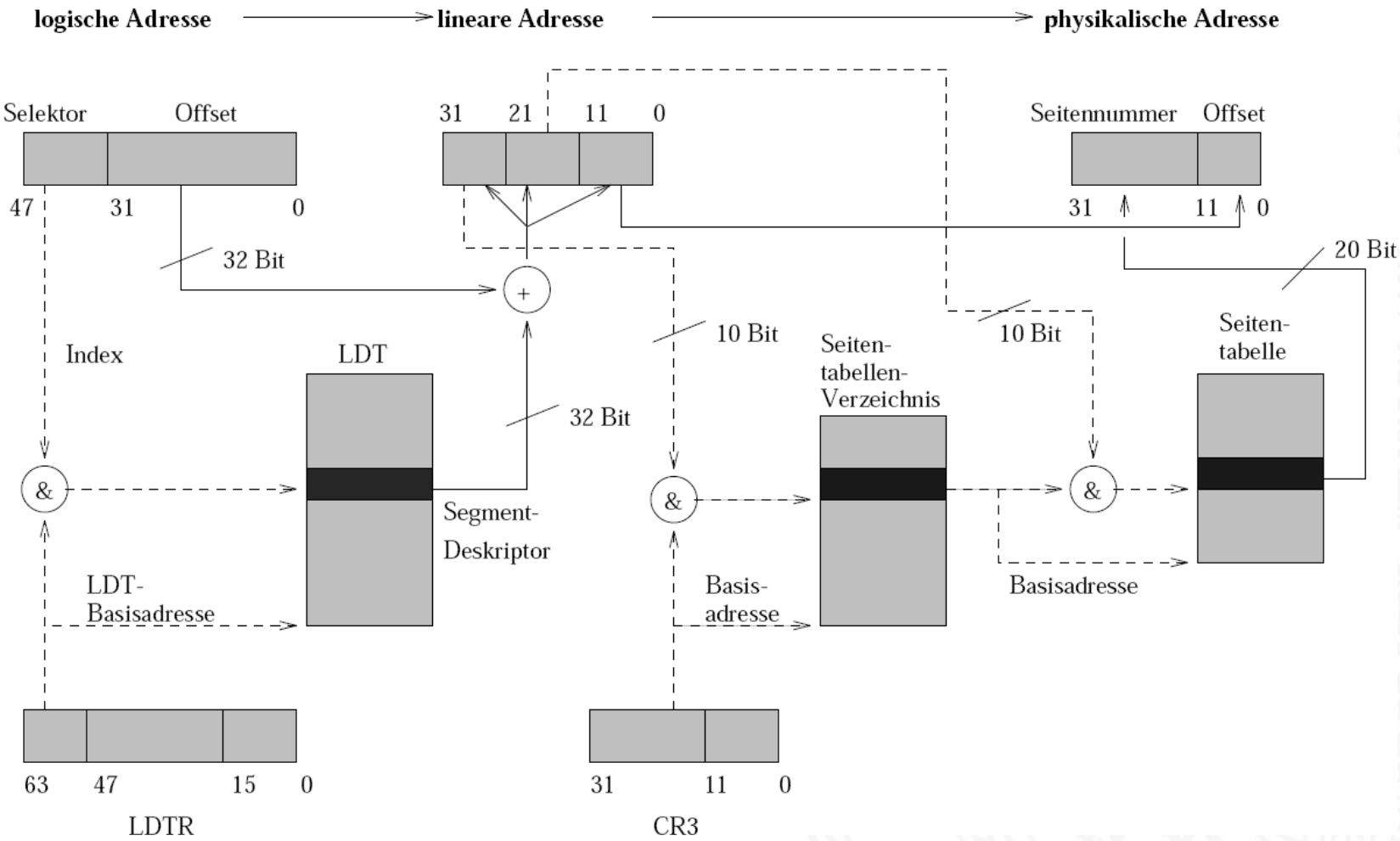
- Schreibstrategie
  - Write-Back (Write-Through wegen Plattenzugriffs zu langsam)
- Ersetzungsstrategie
  - Not recently used
  - Least recently used
  - FIFO...
- Behandlung von Seitenfehlern („*Page-Faults*“) in Software, durch Betriebssystem
  1. Umschalten auf anderen laufbereiten Prozess
  2. Daten in freie Page im Hauptspeicher laden (DMA)
  3. Interrupt wenn DMA fertig ist
  4. Interrupt-Handler: Page-Tabelle des Prozesses updaten
  - beim Zurückschalten auf den ursprünglichen Prozess sind die Daten dann vorhanden

Virtueller Speicher: Segmentierung + Paging

Gemischte Verfahren: Segmentierung und Paging (Beispiel: I386)



Virtueller Speicher: Segmentierung + Paging (cont.)



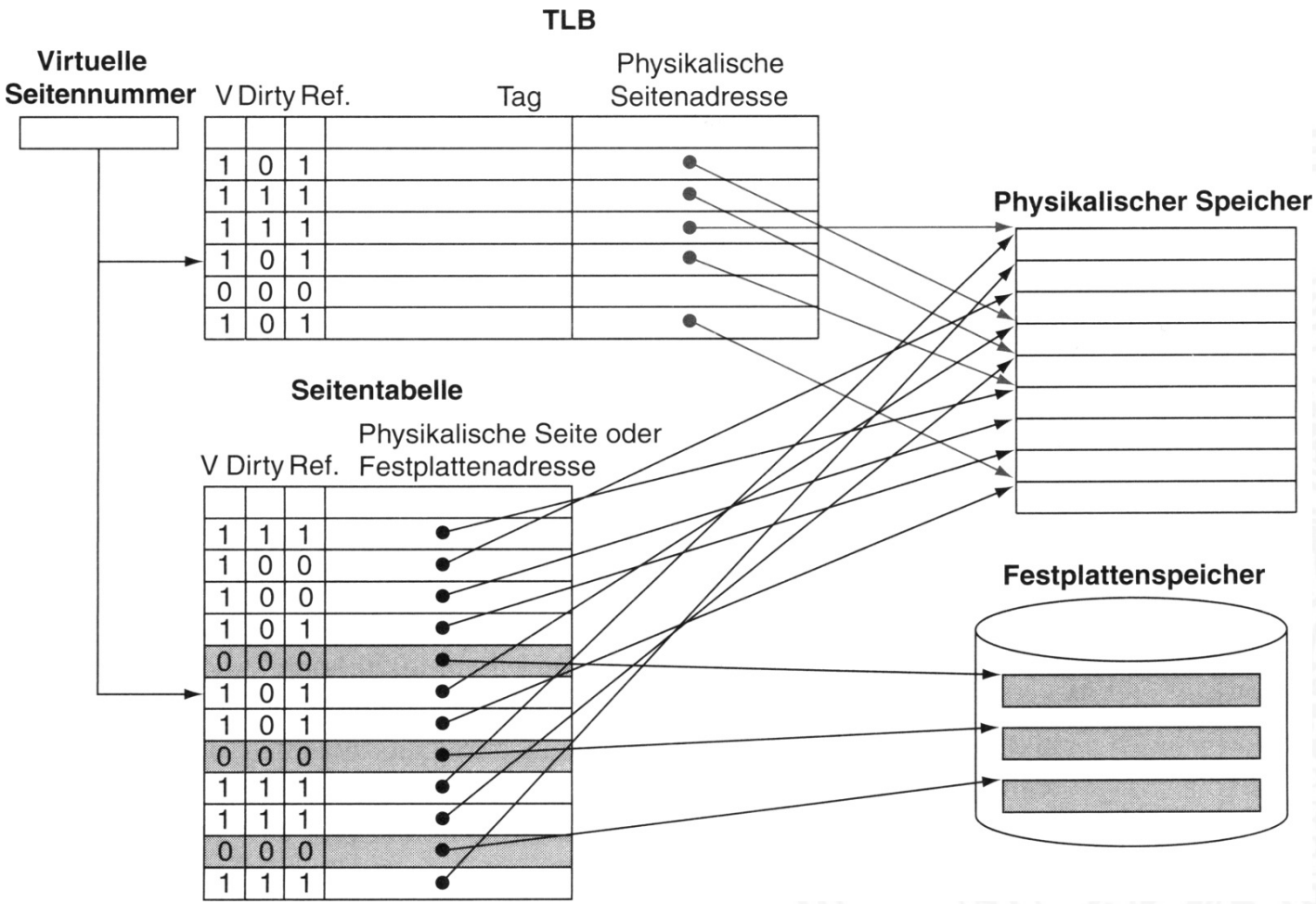
Virtueller Speicher: Segmentierung + Paging (cont.)

- Problem der Adressübersetzung
  - mehrstufige Verfahren
    Beispiel: Segmentierung + 2-stufige Seitenadressierung
  - die jeweiligen Tabellen sind selber im Speicher abgelegt
  - – mehrere Speicherzugriffe

⇒ TLB – **T**ranslation **L**ookaside **B**uffer

- Cache für Seitentabelle

Virtueller Speicher: Segmentierung + Paging (cont.)



Bussysteme

- Aufgaben
  - Kernkomponenten (CPU, Speicher...) miteinander verbinden
  - Verbindungen zu den Peripherie-Bausteinen
  - Verbindungen zu Systemmonitor-Komponenten
  - Verbindungen zwischen I/O-Controllern und -Geräten
  - ...
- viele unterschiedliche Typen, standardisiert
  mit sehr unterschiedlichen Anforderungen
  - High-Performance
  - einfaches Protokoll, billige Komponenten
  - Multi-Master-Fähigkeit, zentrale oder dezentrale Arbitrierung
  - Echtzeitfähigkeit, Daten-Streaming

Bussysteme (cont.)

- wenig Leitungen bis zu Zweidraht-Bussen:
  I^2C, System-Management-Bus...
- lange Leitungen: RS232, Ethernet...
- Funkmedium: WLAN, Bluetooth

- Bus-Bandbreite
  - Menge an Nutzdaten, die pro Zeiteinheit übertragen werden kann
  - RS232 300 Bit/sec ... 110 KBit/sec
    I^2C 100 KBit/sec (Standard), 400 KBit/sec (Fast Mode)
    USB 1,5 MBit/sec (USB 1.x), 480 MBit/sec (USB 2.x)
    ISA 16 MByte/sec
    PCI 128 MByte/sec
    AGP 256 MByte/sec (1x) ... 2 GByte/sec (8x)
    PCIe 250 MByte/sec (x1) ... 8 GByte/sec (x32)
    HyperTransport 12,8 GByte/sec

Bus-Arbitrierung

- mehrere Komponenten wollen Übertragung initiieren
- der Zugriff muss serialisiert werden

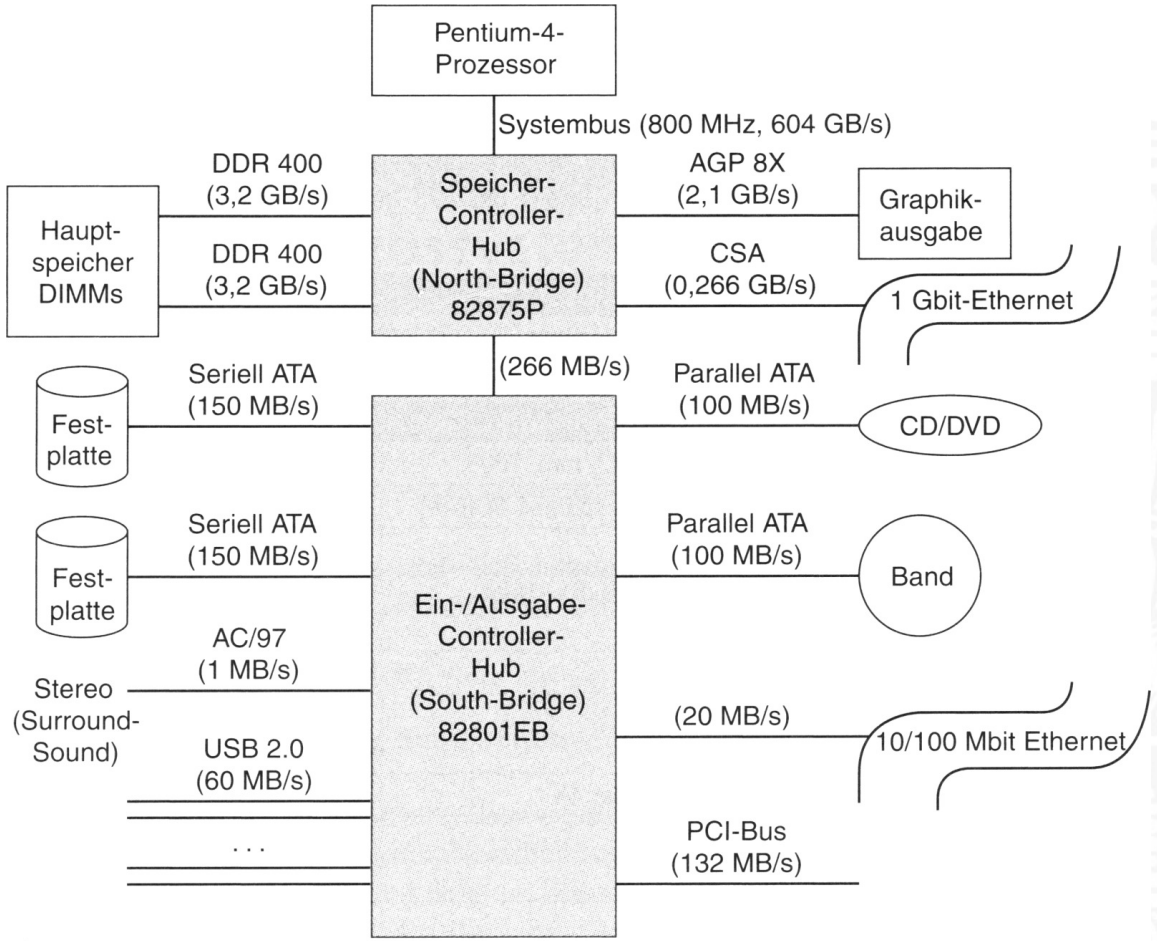
⇒ zentrale Arbitrierung

- Arbiter gewährt Bus-Requests
- Strategien: Priorisierung, Round-Robin...

⇒ dezentrale Arbitrierung

- protokollbasiert
- Beispiel
  - Komponenten sehen ob Bus frei ist
  - beginnen zu senden
  - Kollisionserkennung: gesendete Daten lesen
  - ggf. Übertragung abbrechen
  - „später“ erneut versuchen

PC Bussysteme



Beispiel:
Intel 875 Chipsatz