Ostfalia
Hochschule für angewandte Wissenschaften

Speicherverwaltung

Design Digitaler Systeme

Prof. Dr.-Ing. Rainer Bermbach

Übersicht *Speicherverwaltung*

- Virtueller Speicher
- Memory Management Unit
- Segmentierung
- Paging
- Kombination Segmentierung/ Paging
- Speicherschutz
- Beispiel IA 32 (Intel, AMD u.a.)
- Zusammenfassung

1 Virtueller Speicher

- im Computational Bereich (PC/WS) und im Highend Embedded Bereich häufig verschiedene Aufgaben (Tasks) quasi-parallel
  = Multitasking-Betrieb
- eventuell auch mehrere Benutzer
  = Multiuser-Betrieb
- daraus resultiert hoher Hauptspeicherbedarf, damit Umschaltung einigermaßen transparent vor sich geht
  => hohe Kosten

1 Virtueller Speicher (Forts.)

- Abhilfe: virtueller Speicher
  - Kapazitätserweiterung auf der Festplatte
  - Betriebssystem handhabt dies transparent
  - es verwaltet freien und benötigten Speicher
  - bei Bedarf Auslagern von momentan nicht benötigten Blöcken und Lesen der gebrauchten Bereiche
    = Swapping
  - Programme müssen lageunabhängig laufen
- => Bedarf für eine Verwaltungseinheit:
  MMU - Memory Management Unit

2 Memory Management Unit

- führt Adressumsetzung durch
  - Umwandlung von logischen (virtuellen) in physikalische Adressen
- benötigt Memory Map vom Betriebssystem
- Memory Map ist anpassbar => flexibel
- realisiert Zugriffsschutz
  - extrem wichtig bei Multiuser/Multitasking
- muss schnell sein (liegt im Zugriffspfad)

2 Memory Management Unit (Forts.)

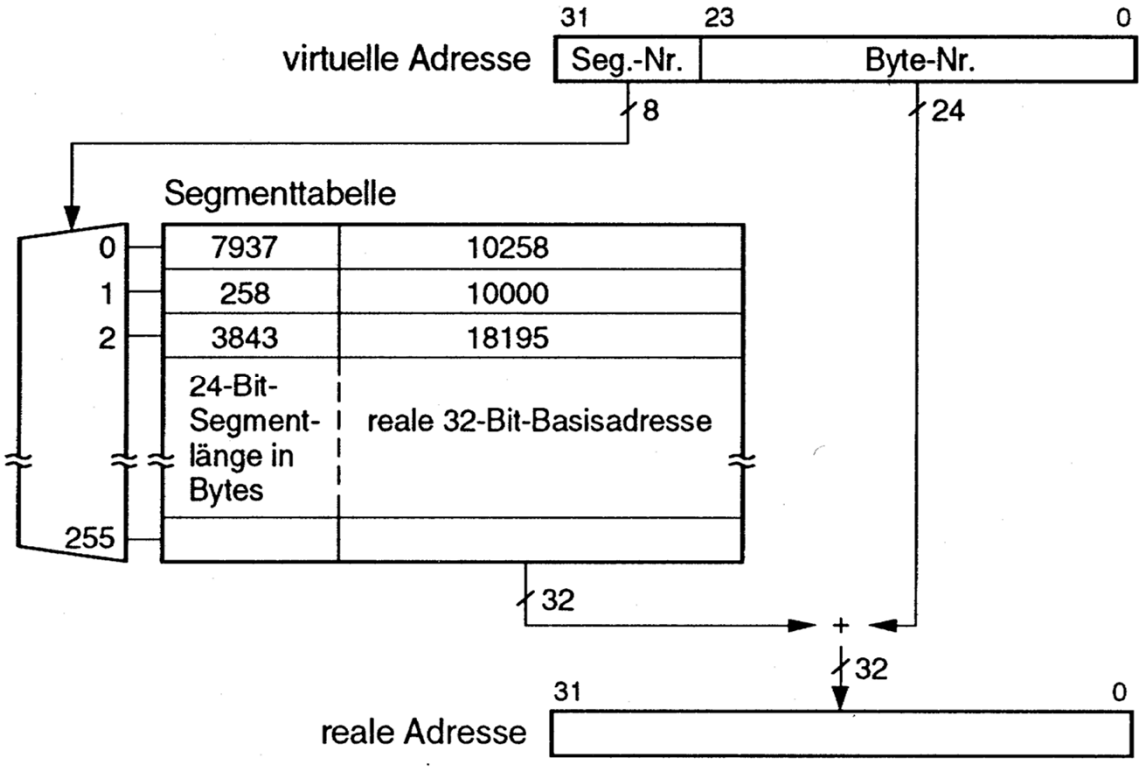
- Realisierung der Adressumsetzung nicht für jede Adresse einzeln sondern in Blöcken
- verschiedene Realisierungen:
  - *Segmentierung*
  - Seitenverwaltung (*Paging*)
  - Kombination von beiden
- Adressumsetzungsdaten
  + Schutzattribute
  + Statusangaben
  = *Deskriptor* (eines Speicherbereiches)

3 Segmentierung

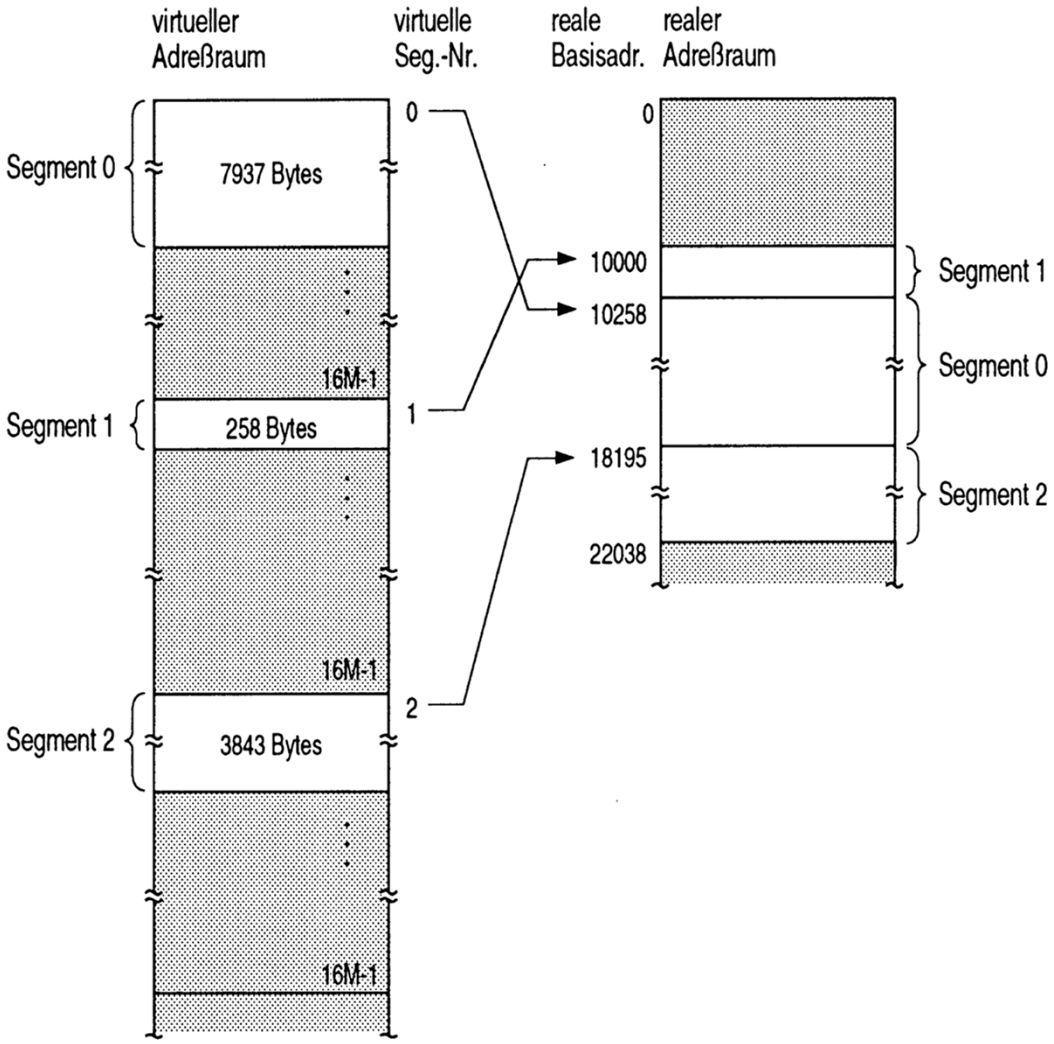
- Segmente:
  - Speicherblöcke unterschiedlicher Größe
  - bis zu einer festen, maximalen Größe
  - im virtuellen Speicher freie Bereiche, wenn Segmentgröße < maximale Größe
  - im physikalischen Speicher anschließend
- Segmentierung bei 8086 etc.:
  - logische Adresse: Segment : Offset
  - physische Adresse: Segment x 16 + Offset
  - Trivialform

3 Segmentierung (Forts.)

- Beispiel 1 :
  - Segmentbasisadressen auf *Byte*grenzen
  - Segmentinformation aus der logischen Adresse



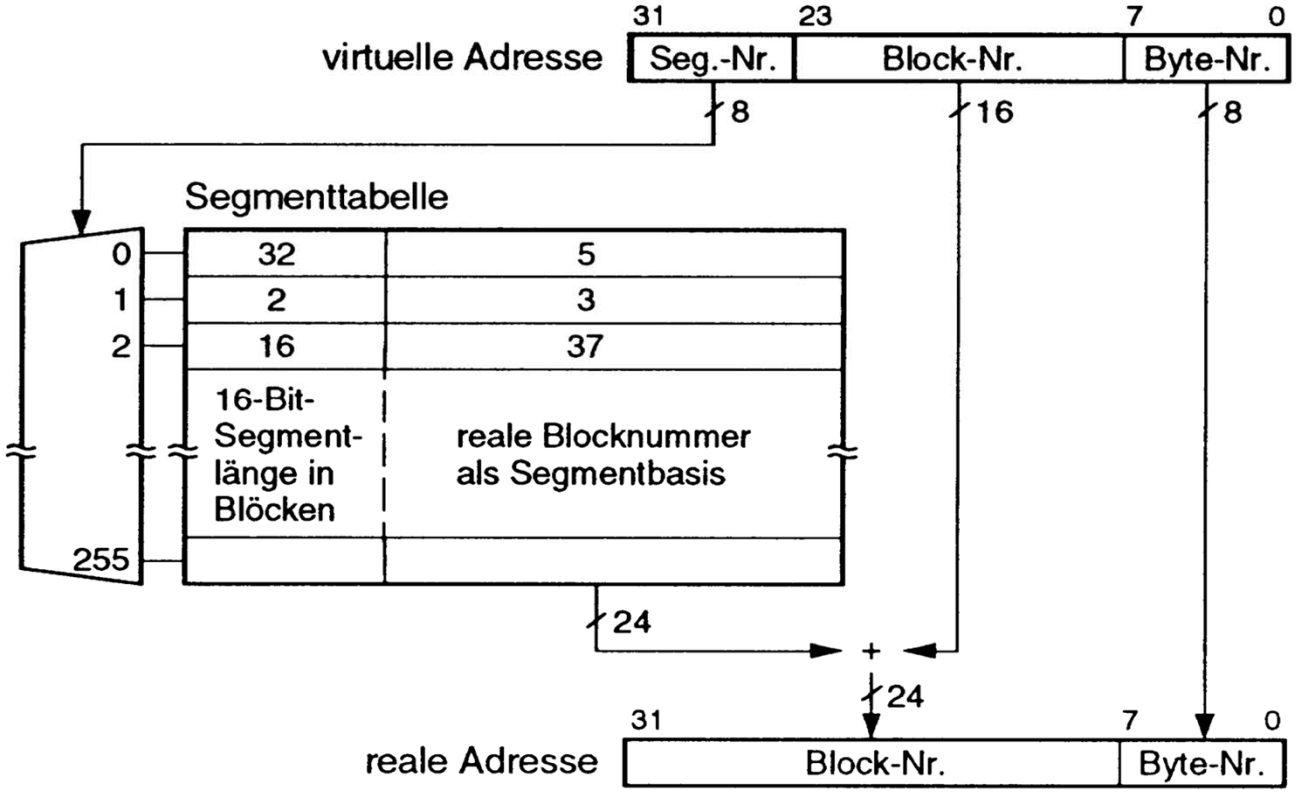
3 Segmentierung (Forts.)



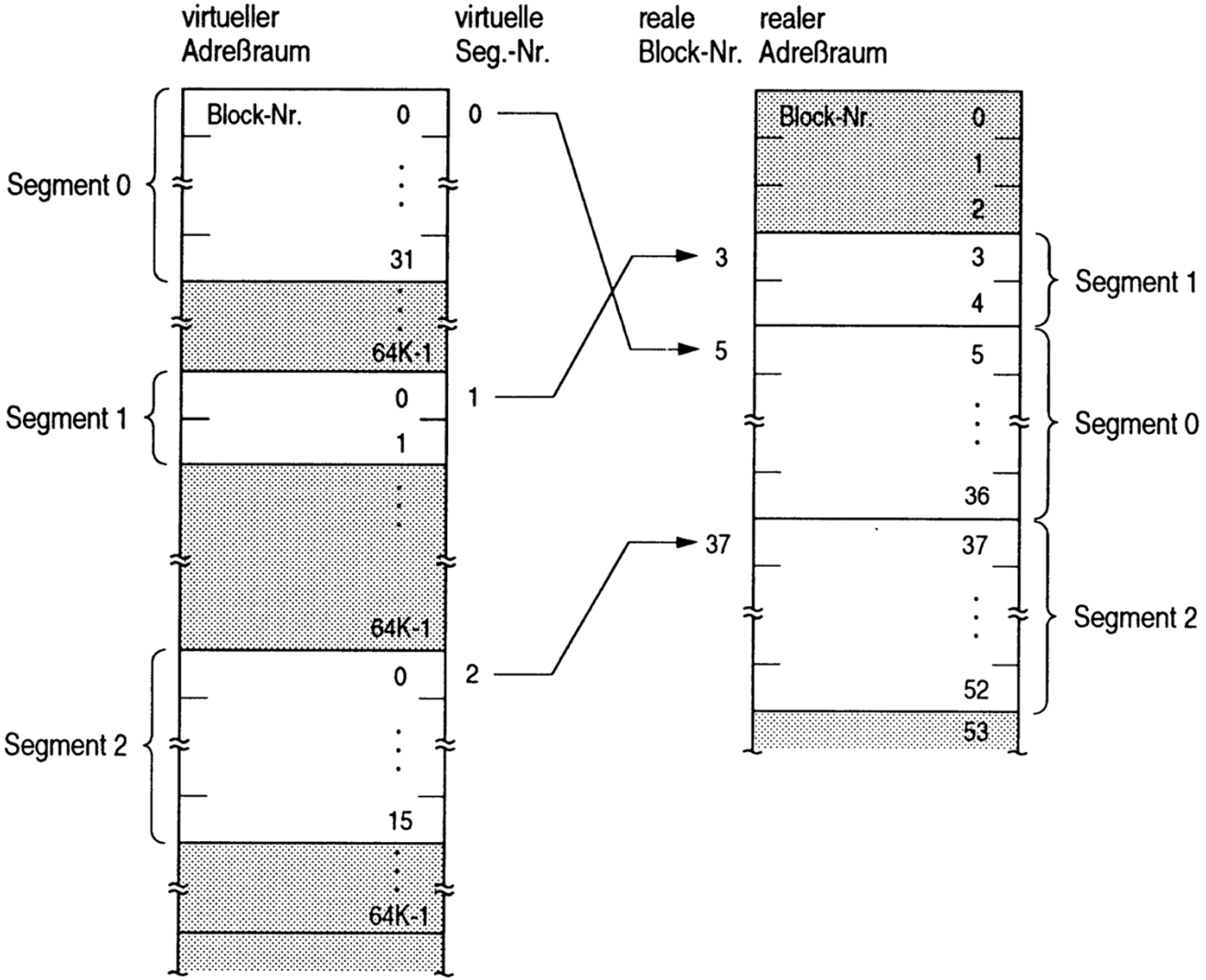
- max.16 MByte virtuelle Segmentgröße = maximale physische Segmentgröße
- 256 Segmente
- d.h. 4 GB virtueller Adressraum (=32 Adr.)
- Bytegrenzen für physischen Speicher, d.h. kein „Verschnitt“
- 32-Bit-Addierer nötig

3 Segmentierung (Forts.)

- Beispiel 2
  - Segmentbasisadressen auf *Block*grenzen
  - Segmentinformation aus der logischen Adresse



3 Segmentierung (Forts.)



3 Segmentierung (Forts.)

- im Beispiel 2
  - Blöcke von 256 Byte
  - Adressbits 8 - 23 = Offset in Blockansammlung
  - Segmenttabelle liefert Blockbasis und Länge
  - maximal 64K Blöcke
  - virtuell: max. Segmentgröße 64K Blöcke =16 MByte
  - 4 GB virtueller Adressraum (entspr. 32 Adr.)
  - Blockgrenzen für physikalischen Speicher, d.h. max. Verschnitt 256 Byte
  - 24-Bit-Addierer nötig
  - ähnlich Zilog Z8000-Familie

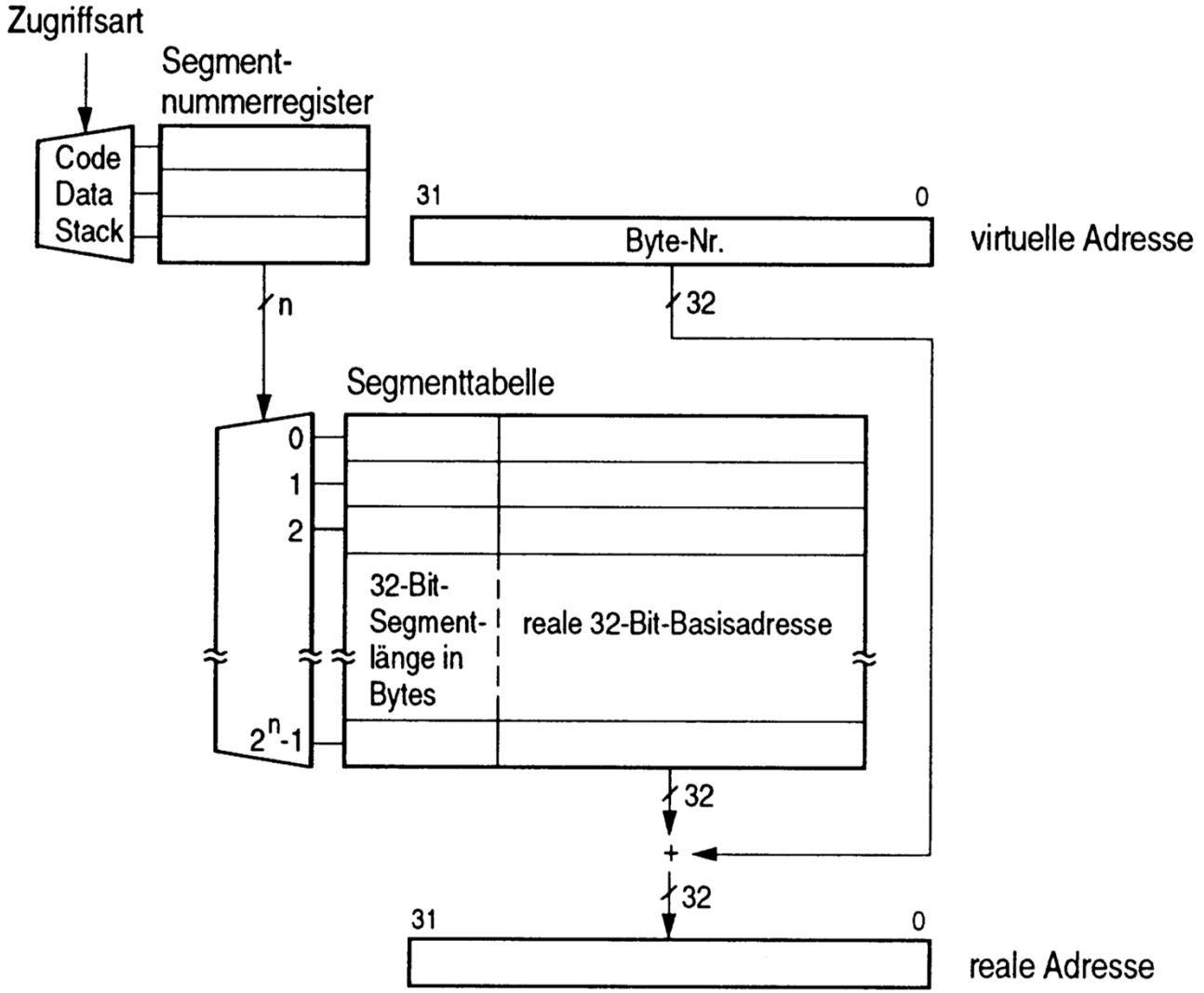
3 Segmentierung (Forts.)

- Beispiel 3
  - Verbesserung :
  - Aufteilung der Adresse variabel:
  - Grenze zwischen Block- und Segmentnummer verschiebt sich für jedes Segment
  - auch im Virtuellen nur erforderlichen Platz belegt
  - ähnlich Motorola 68000-Familie

3 Segmentierung (Forts.)

- Beispiel 4
  - Erweiterung der virtuellen Adresse
  - bisherige Beispiele:
    - Segmentkennzeichnung Teil der Adresse
  - günstiger:
    - getrennte Bereitstellung von Segmentadressen
    - Vergrößerung des virtuellen Adressraums
    - jedes Segment hat 4 GByte zur Verfügung
    - vgl. Intel (IA32): 80386 - Pentium III
    - virtueller Adressraum 64 Terabyte (2^{14} x 4 GByte)

3 Segmentierung (Forts.)



3 Segmentierung (Forts.)

- Vorteile der Segmentierung:
  - Verwaltung von logisch zusammengehörigen Einheiten (Code, Daten, Stack)
  - schnelle Änderungen, nur ein Deskriptor
- Nachteile:
  - Swapping betrifft immer ganzes Segment
  - u.U. Zerstückelung des Hauptspeichers (wg. unterschiedlicher Größen)
- Nachteile vermeiden mit *Paging*

4 Paging

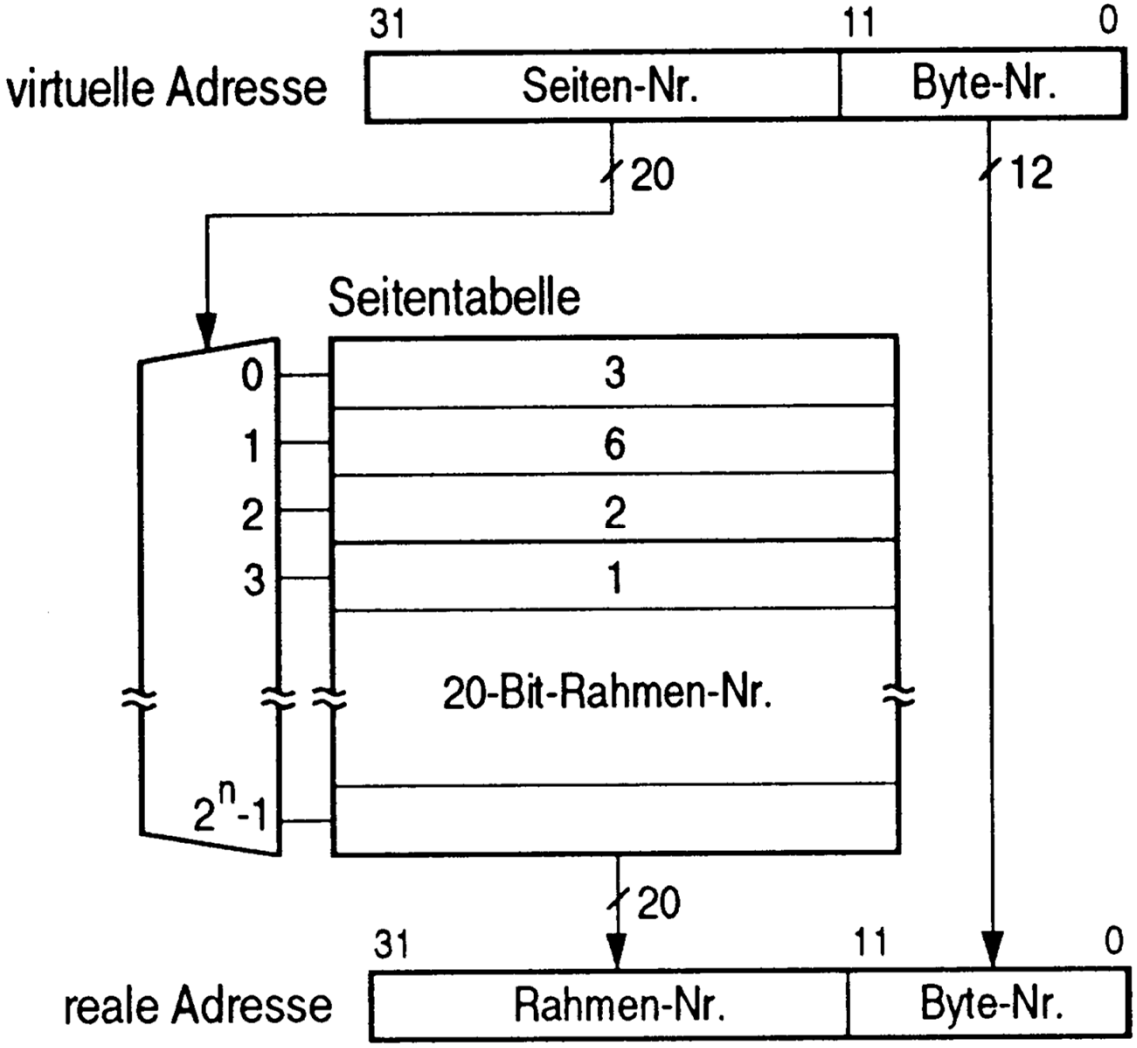
- bessere Speichernutzung durch Aufteilung in sog. Pages
  - typische Page-Größe 4 KByte
  - Abbildung auf physikalische Frames gleicher Größe
- Vorgehen:
  - höherwertige Adressbits sind virtuelle Seiten-nummer auf physische Frame-Nummer
  - niederwertige Bits adressieren in der Page
  - Frames können beliebig im Speicher verteilt sein

4 Paging (Forts.)

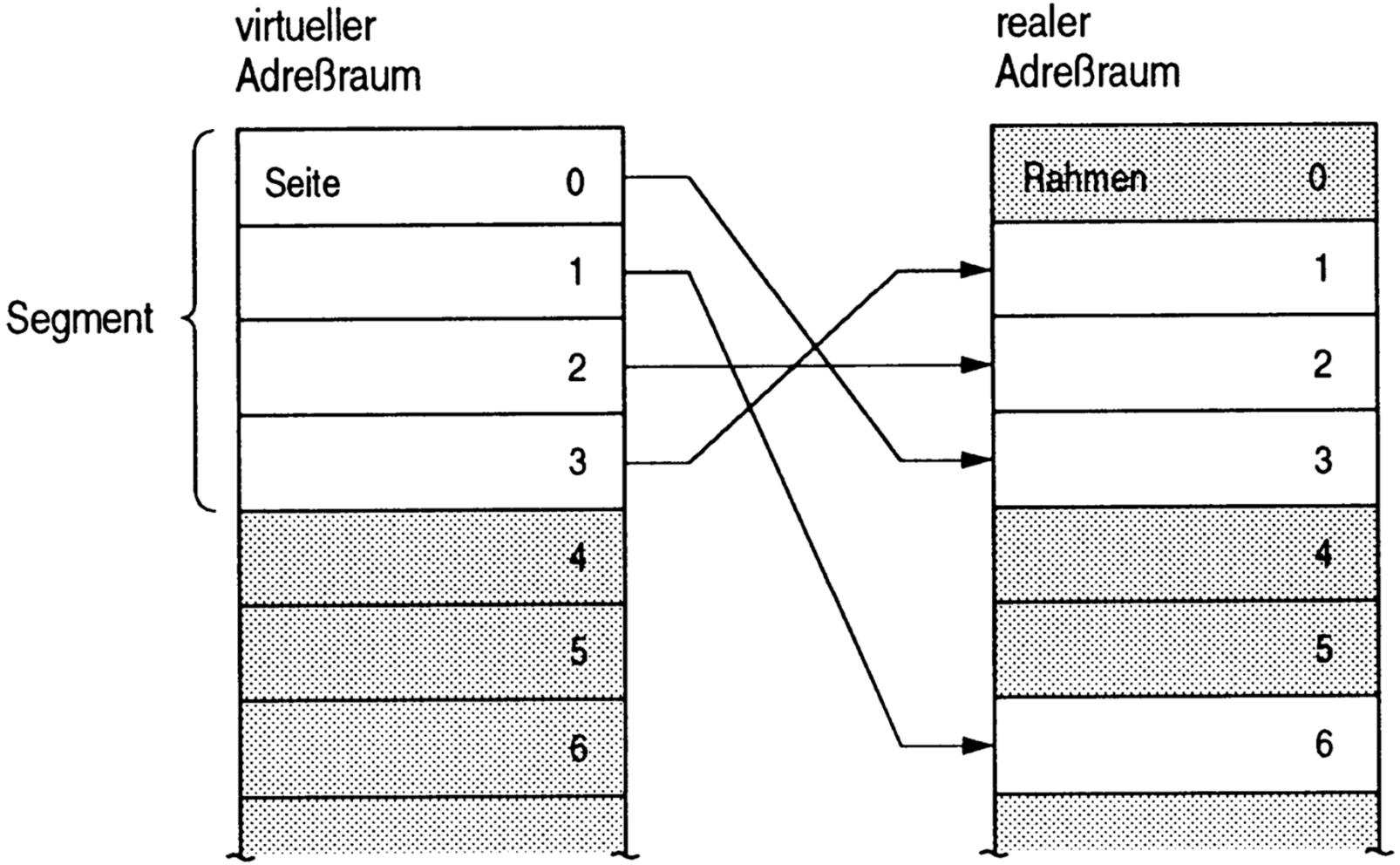
- Vorteile:
  - es ist möglich, nur wenige Seiten im Hauptspeicher zu halten (Working Set)
  - bedeutet Reduzierung des Hauptspeicherbedarfs
  - effizienteres Swapping
  - keine Addierer
  - kein „Aufräumen“ des Hauptspeichers nötig, da beliebige Zuordnung
- bei sog. *Page Fault* Nachladen der benötigten Seite:

  Demand Paging

4 Paging (Forts.)



4 Paging (Forts.)

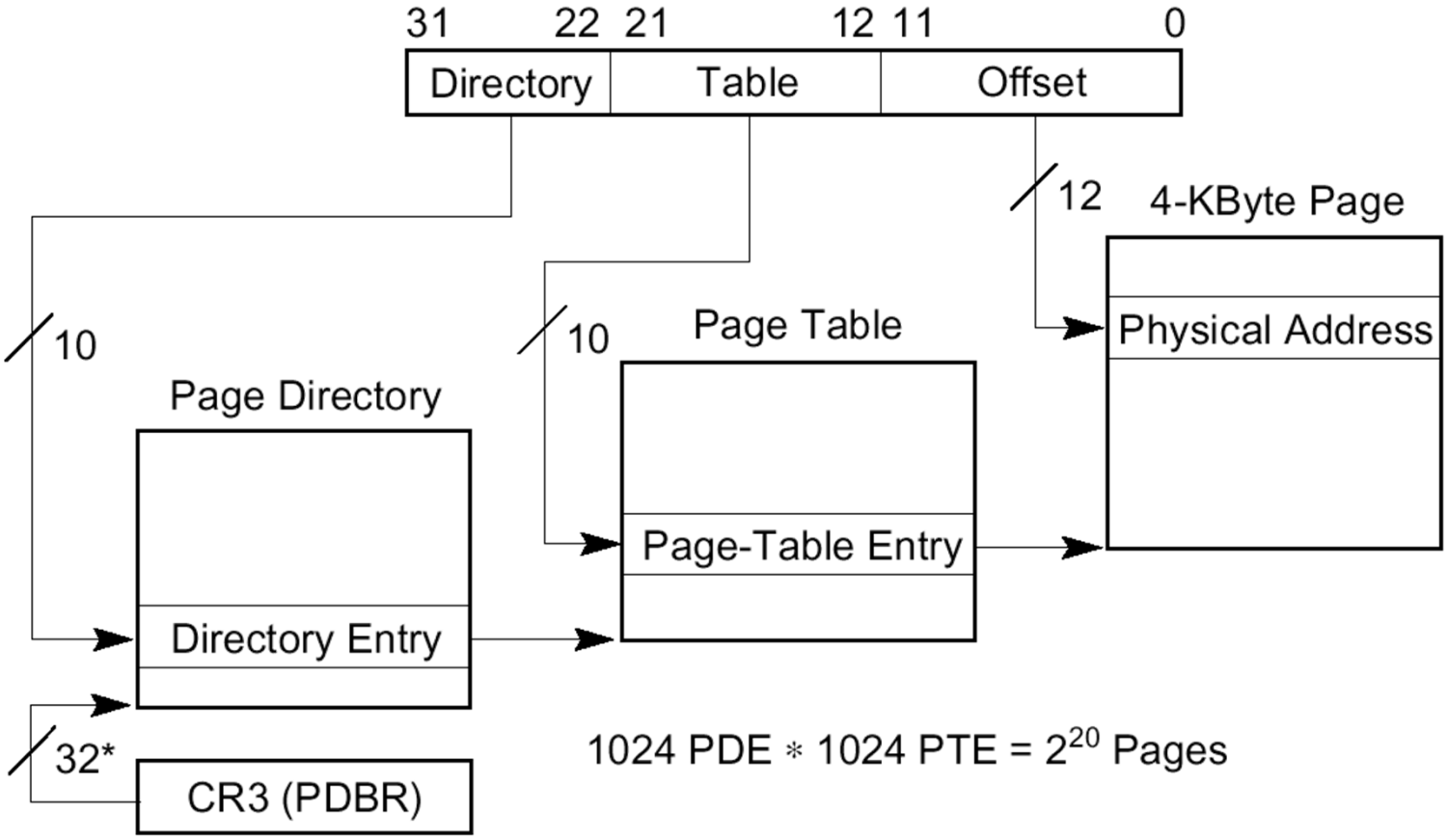


4 Paging (Forts.)

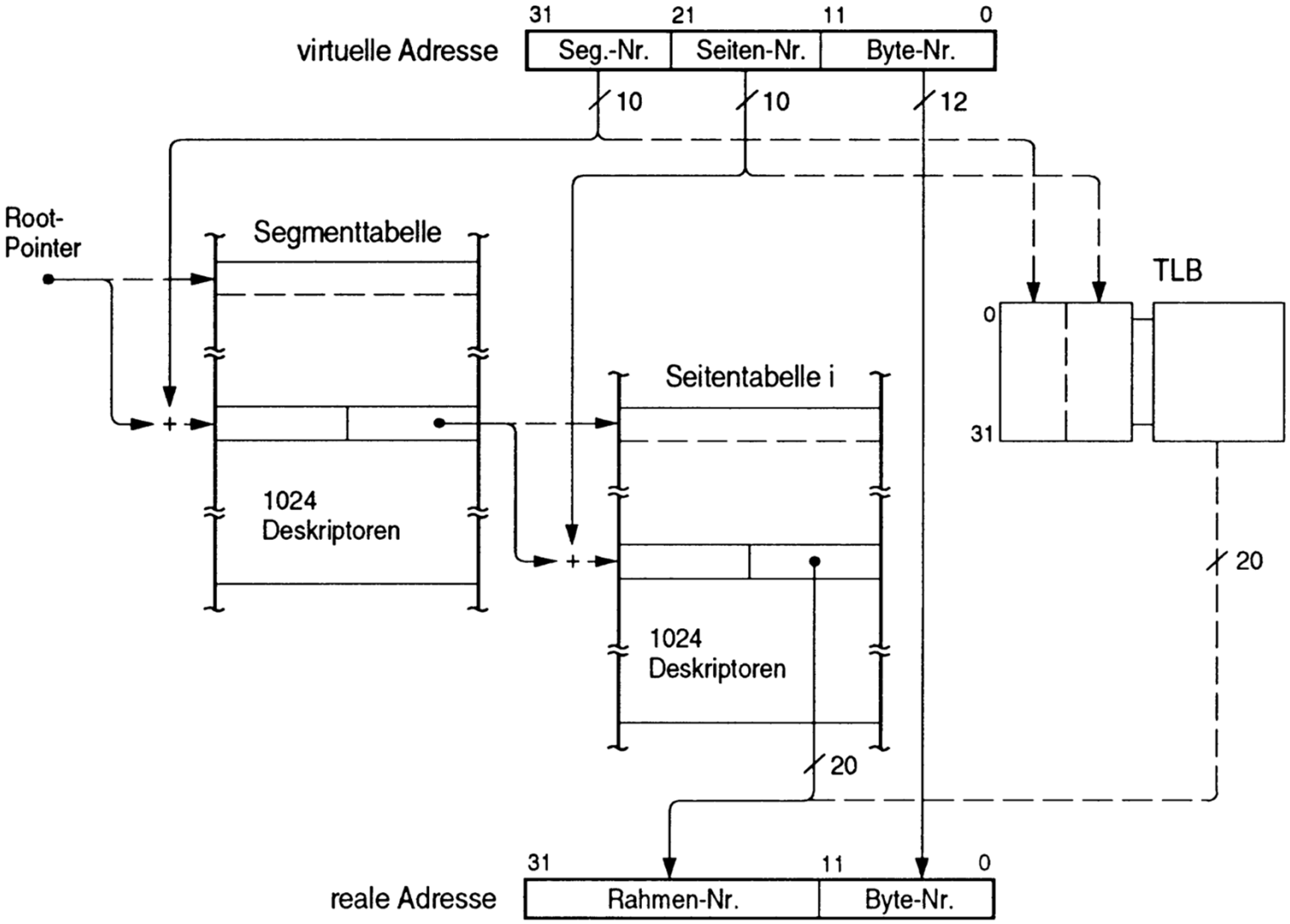
- Nachteil:
  - Adressumsetzungstabelle u.U. sehr groß
  - im Beispiel (2^{20}) 1 M x 20 Bit
  - deshalb reines Paging nur bei Prozessoren mit Adressraum 64KB
  - Änderungen betreffen u.U. sehr viele Einträge
  - keine logische Umsetzung
- Abhilfe:
  - mehrstufige Umsetzung (typ. 2-3 Levels)
  - und/oder Kombination mit Segmentierung

4 Paging (Forts.)

- zweistufiges Paging (Beispiel IA 32)



5 Kombination Segmentierung/ Paging



5 Kombination Segment./ Paging (Forts.)

- hier: Unterteilung in
  - 1.024 Segmente (10 Bit) à
  - 1.024 Pages (10 Bit) zu je
  - 4 KByte (12 Bit)
- Zugriff über Speichertabellen langsam, deshalb Cache für die MMU (auch bei reinem Paging)
- TLB - Translation Look-Aside Buffer
  - enthält die Deskriptoren der zuletzt benutzten Pages
  - typische Größe 32 - 64 Einträge (voll-assoziativ)
  - sehr schnell

6 Speicherschutz

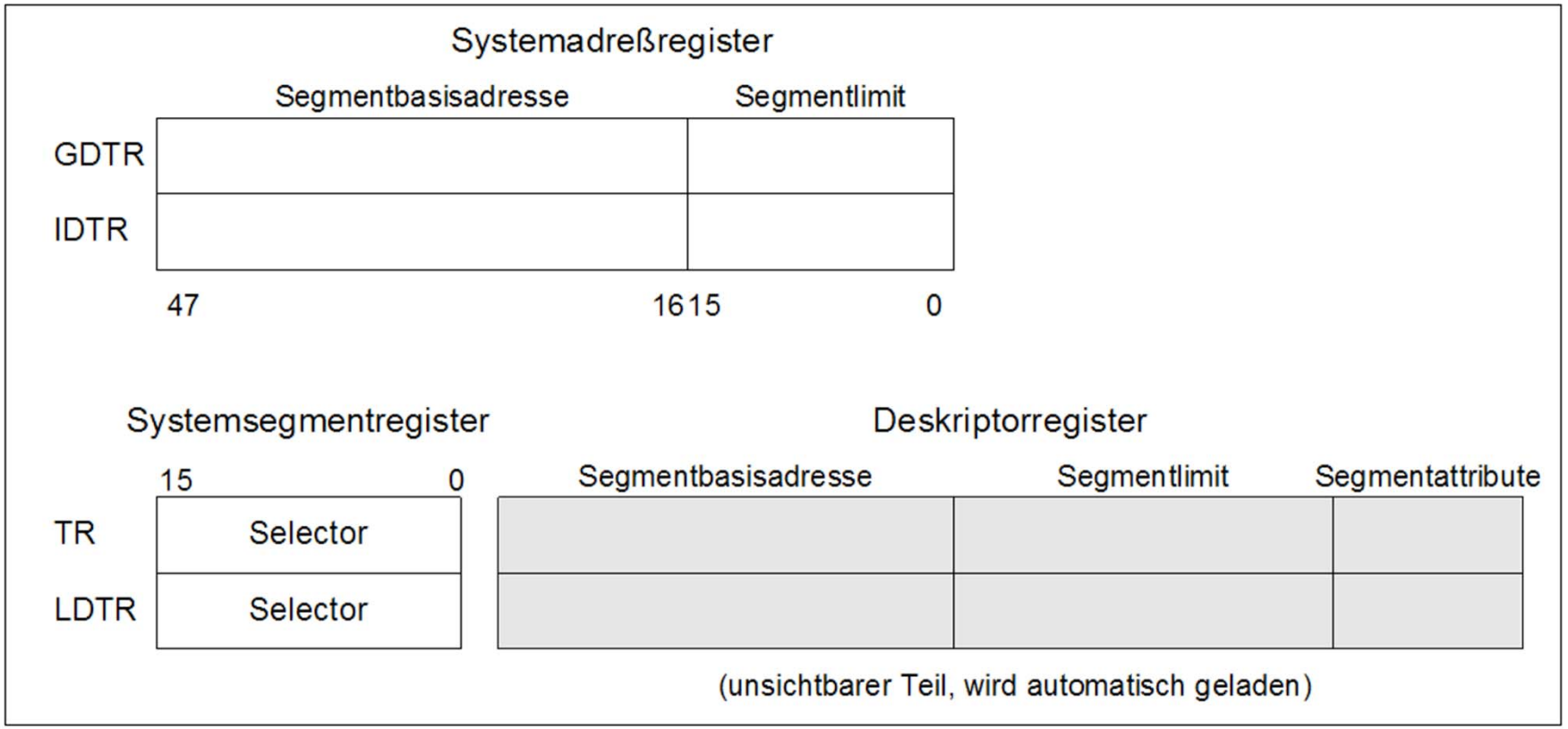
- weitere Informationen in den Deskriptoren
  - Länge des Segmentes => Test ob Überschreitung
  - Attribute:
    - Daten/Code, read-only, execute-only etc.
    - Segment „gültig“ / „muss nachgeladen werden“
  - vergleichbare Attribute bei Page-Deskriptoren plus
    - „Zugriff erfolgte“, „wurde verändert“
    - cacheable / non cacheable
  - globale und task-spezifische Deskriptoren
  - Privilegebenen
    - nur Zugriff auf Segmente gleichen Levels
    - sog. Call Gates für Zugriff auf Systemroutinen

7 Beispiel IA 32 (Intel, AMD u.a.)

- Intel verwendet in der sog. IA 32 (80386 bis Pentium ..., ansatzweise bereits im 80286) *Segmentierung und Paging*
  - Segmentierung mit Adresserweiterung über Segmentregister
    - erlaubt volle Segmentierung
    - aber auch Deaktivierung (Flat Memory Model)
  - im Anschluss wahlweise zusätzliches Paging
    - zweistufige Umsetzung, 4 KByte Pages
    - deaktivierbar

7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- vier Systemregister zur Adressierung von Segmenttabellen:

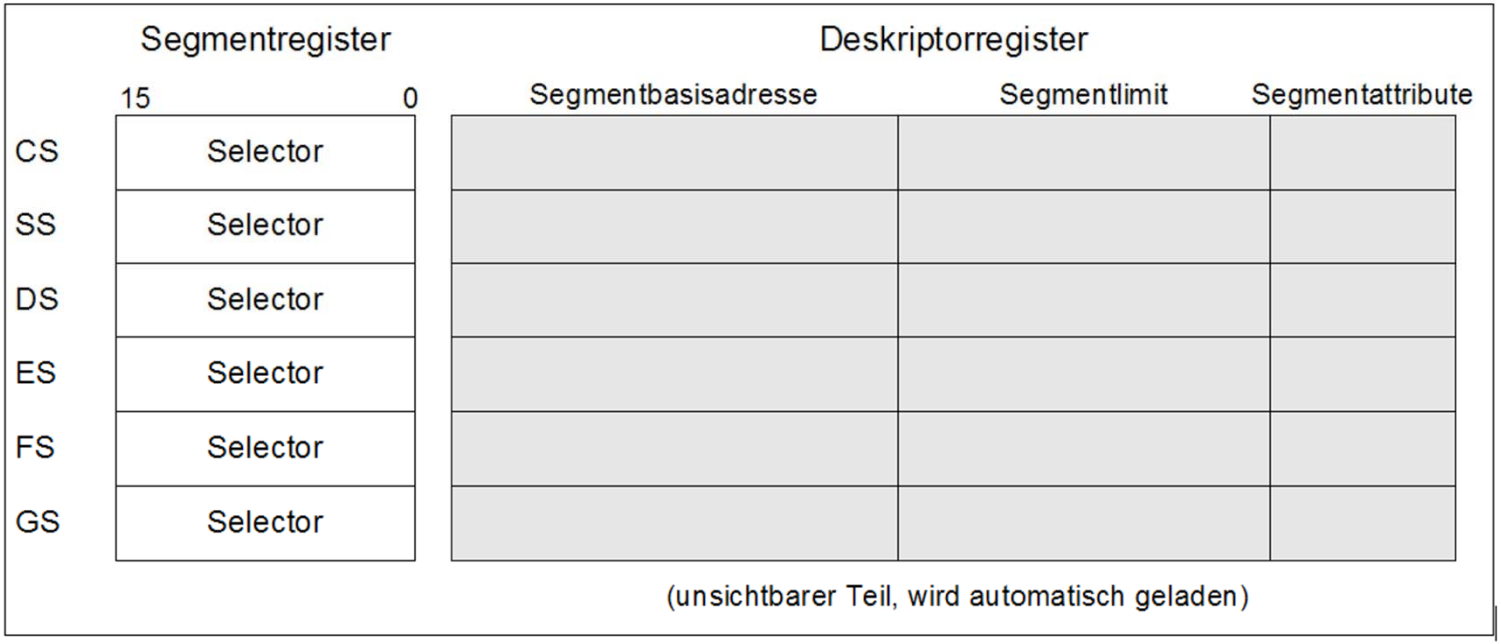


7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- *GDTR* - Global Descriptor Table Register
  - zeigt auf Tabelle mit globalen Segmenten, für alle Tasks zugänglich (wenn Priorität)
- *IDTR* - Interrupt Descriptor Table Register
  - zeigt auf Tabelle mit Interruptsegmenten
- *LDTR* - Local Descriptor Table Register
  - zeigt auf Tabelle Task-lokaler Segmente
- *TR* - Task Register
  - zeigt auf Tabelle (TSS) momentan aktiver Task

7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- Segmentregister (mit verborgenem Descriptor Cache)

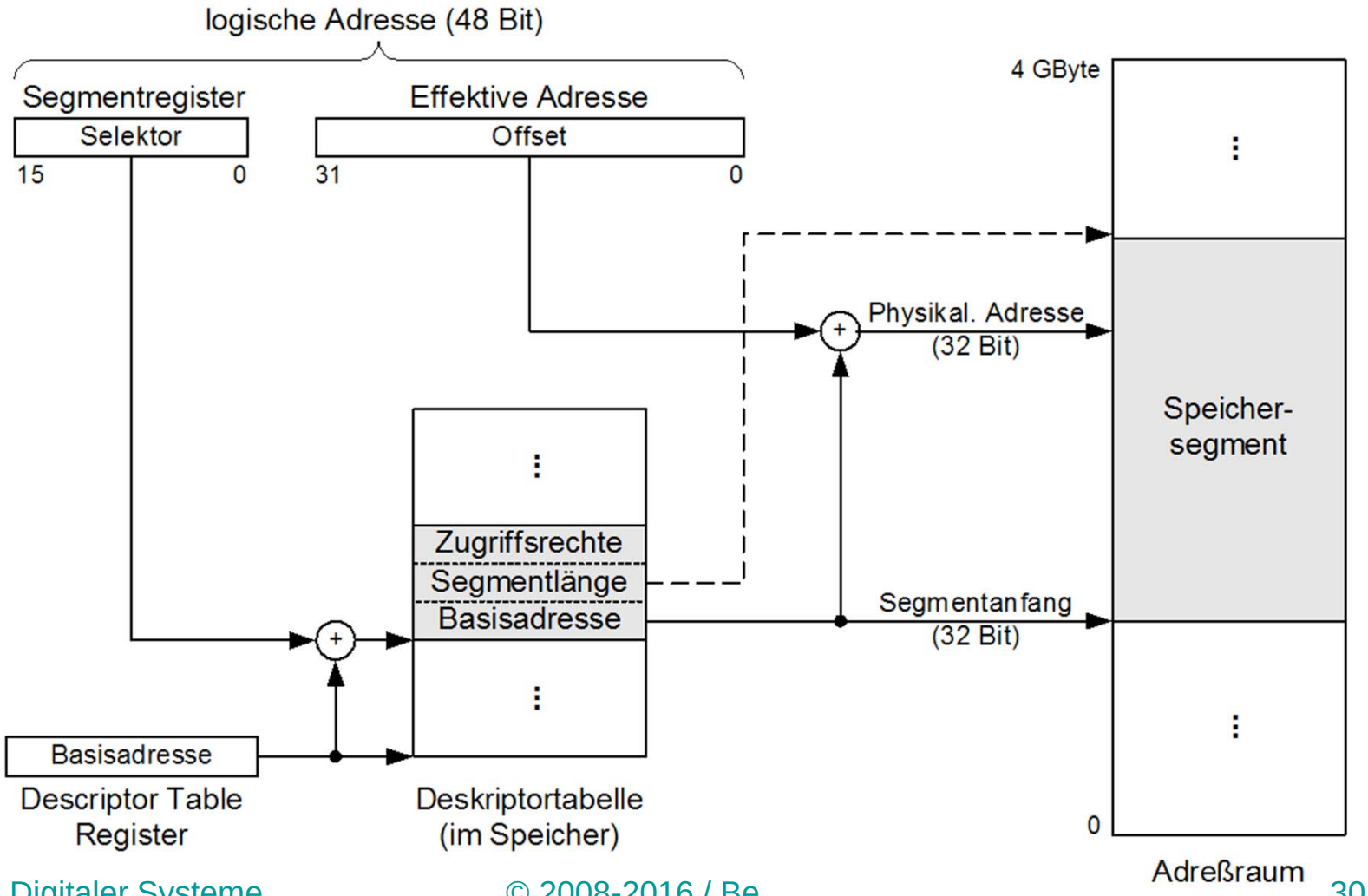


TI: Table Indicator

RPL: Requestor‘s Privilege Level

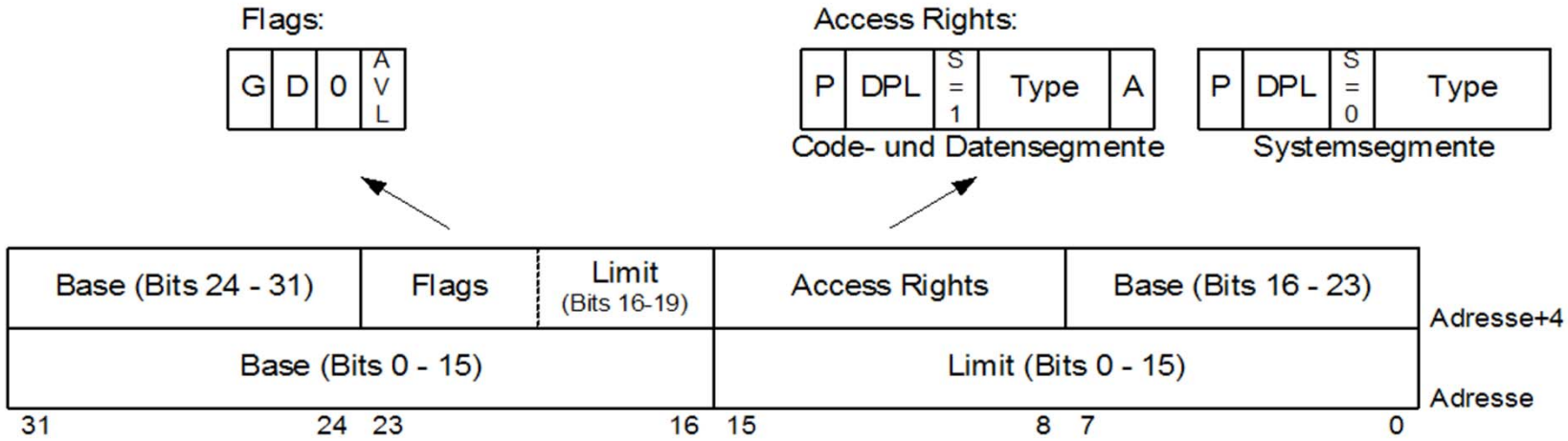
7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- Adressumsetzung mit Deskriptoren (GDT/LDT)



7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- Format eines Deskriptors



G: **Granularity (Länge in Byte / Länge in 4KB-Blöcken)**

D: **Default Operation Size (16-Bit- / 32-Bit-Segmente)**

P: **Present (ungültig / im Speicher)**

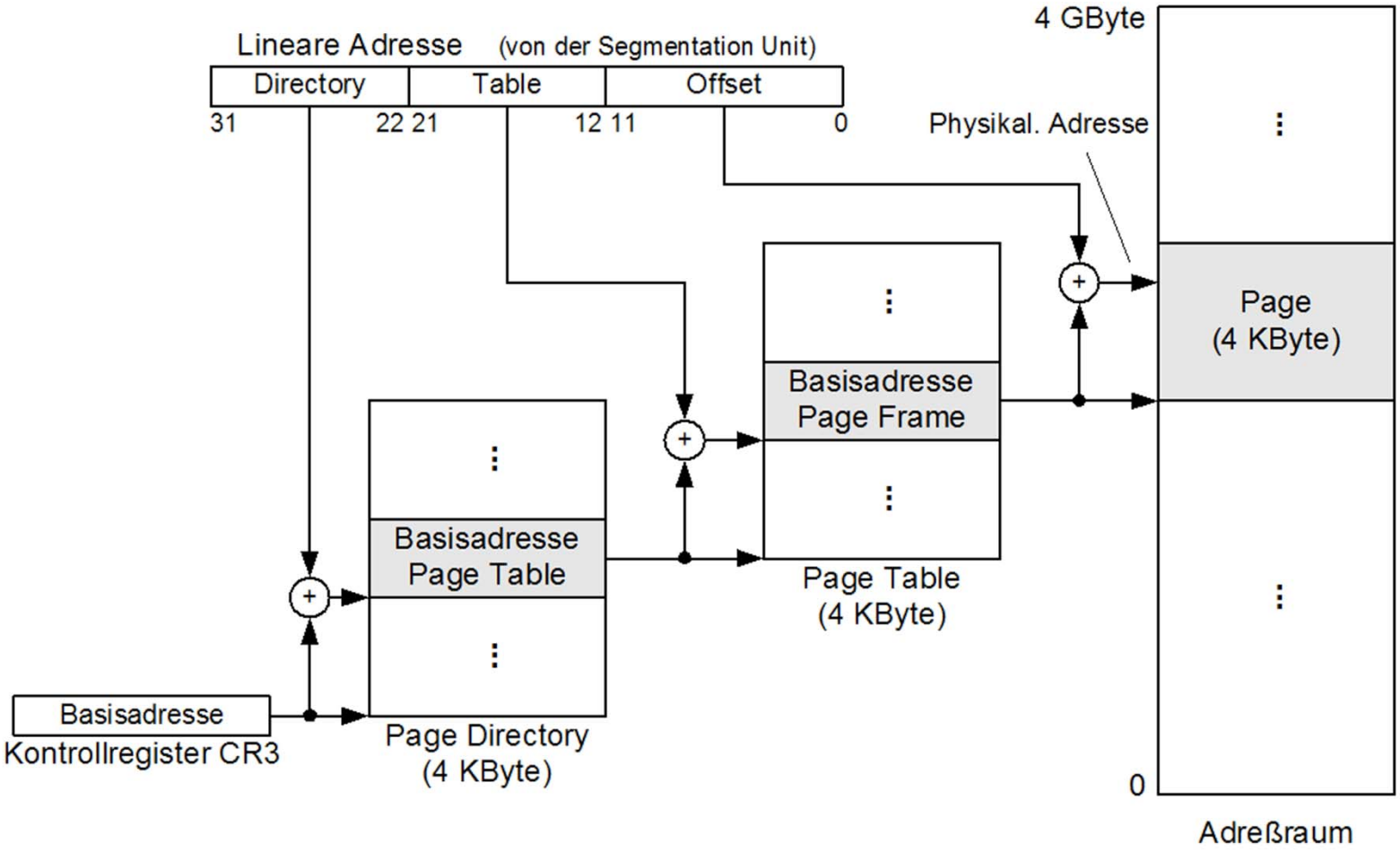
DPL: **Descriptor Privilege Level (Privilegebene d. Segmentes)**

Type: **Type des Segmentes (Code/Daten etc. bzw. LDT/TSS etc.)**

A: **Accessed (gesetzt, wenn Zugriff erfolgte)**

7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- **Paging** baut auf der Ausgabeadresse der Segmentation auf („Lineare Adresse“)



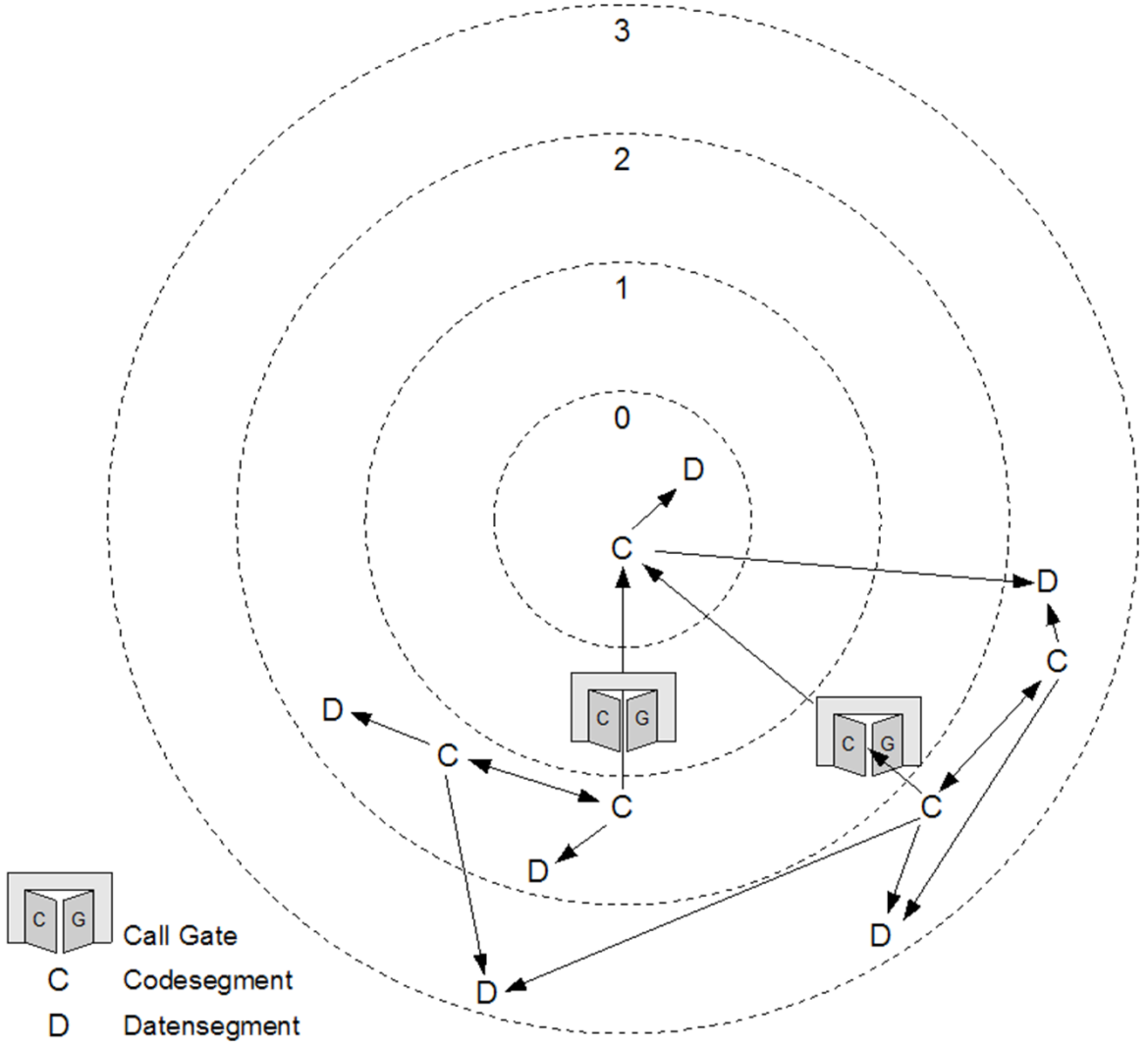
7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- Speichernutzung in der IA 32
  - Flat Memory Model
    - linearer Adressraum, unsegmentiert
  - Segmentierung
    - wenige Segmente oder voll segmentiert
    - nur GDT oder mit einer oder mehreren LDTs
  - Paging alleine
    - setzt z.B. auf Flat Memory Model auf
  - Paging mit vorheriger Segmentation

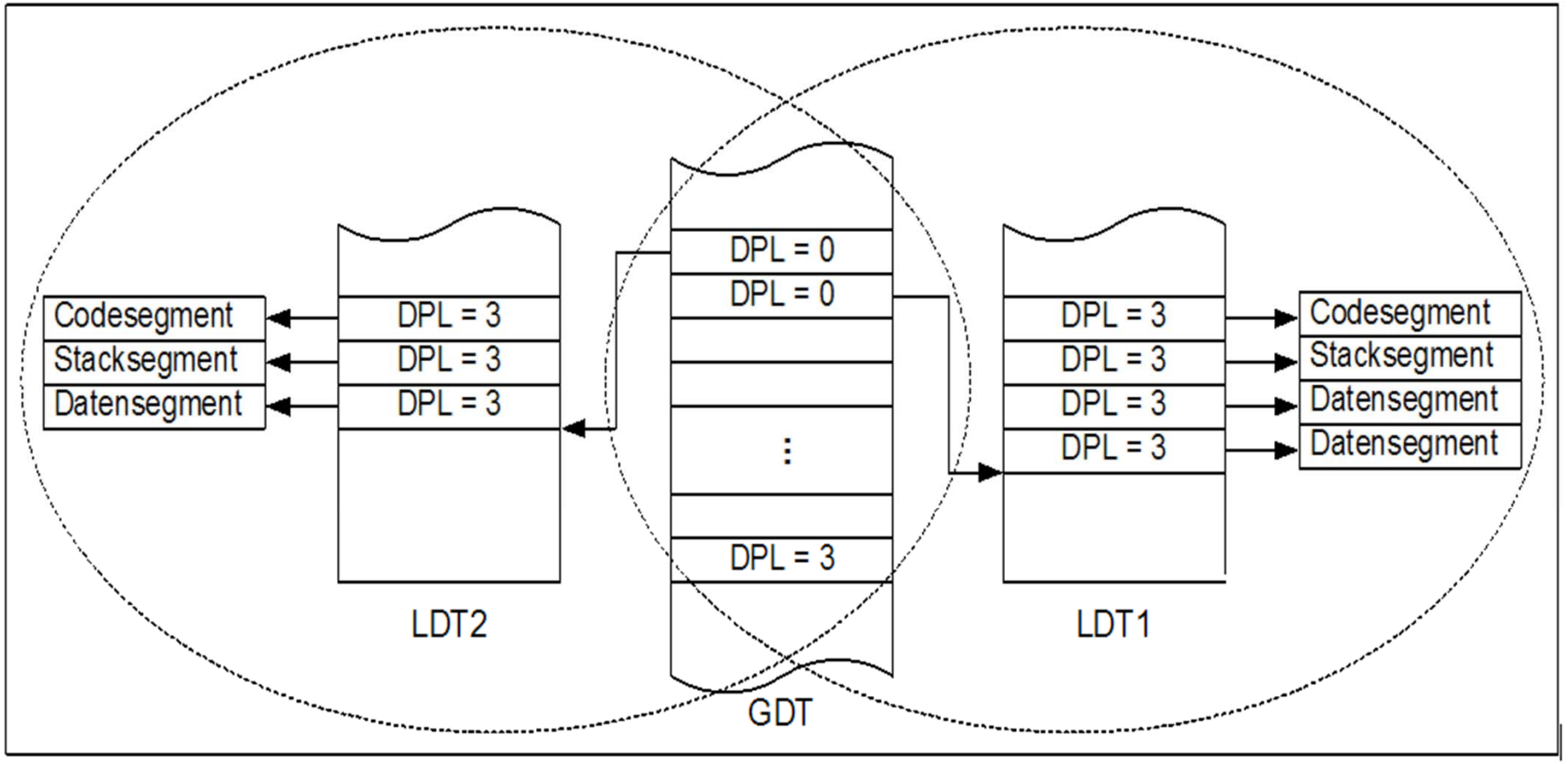
7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)

- Schutzmechanismen
  - Segmentlänge
  - Segmenttypen (Type-Feld in Access Rights)
    - Beispiel Call Gates
  - Unterscheidung in globale (GDT) und private Deskriptortabellen (LDTs)
  - vierstufige Privileg-Hierarchie (Ring 0- 3)

7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)



7 Beispiel IA 32 (Intel, AMD u.a.) (Forts.)



8 Zusammenfassung

- Notwendigkeit für virtuellen Speicher, MMU
- Segmentierung
  - Segmentinfo aus Adresse, Byte- oder Blockgrenze
  - Erweiterung mit Segmentregistern
  - Verwaltung logischer Einheiten
  - im Speicher u.U. sehr groß
- Paging
  - bessere Speicherausnutzung durch feste Größe
  - Abbildung von Blöcken gleicher Größe
  - evtl. mehrstufig und/oder in Kombination mit Segmentierung
- Speicherschutz
- Beispiel IA 32

Vielen Dank für Ihre Aufmerksamkeit !

ENDE