6.Vorlesung Grundlagen der Informatik

Christian Baun

Hochschule Darmstadt
Fachbereich Informatik
christian.baun@h-da.de

17.11.2011

Wiederholung vom letzten Mal

- Ersetzungsstrategien
- Festplatten
- Solid State Drives
- Redundant Array of Independent Disks (RAID)

Heute

- Zeichen-/Blockorientierte Geräte
- Daten von Ein- und Ausgabegeräten lesen
- Adressraum
- Speicheradressierung und Speicherverwaltung
- Speicherpartitionierung
  - Statische Partitionierung
  - Dynamische Partitionierung
  - Buddy-Verfahren
- Virtueller Speicher
  - Memory Management Unit
  - Seitenorientierter Speicher (Paging)
  - Segmentorientierter Speicher (Segmentierung)

Zeichenorientierte und Blockorientierte Geräte

- Geräte an Computersystemen werden bezüglich der kleinsten Übertragungseinheit unterschieden in:
  - **Zeichenorientierte Geräte**
    - Bei Ankunft/Anforderung jedes einzelnes Zeichens wird immer mit dem Prozessor kommuniziert
    - Beispiele: Maus, Tastatur, Drucker, Terminals und Magnetbänder
  - **Blockorientierte Geräte**
    - Datenübertragung findet erst statt, wenn ein kompletter Blocks (z.B. 1-4 KB) vorliegt
    - Beispiele: Festplatten, CD-/DVD-Laufwerke und Disketten-Laufwerke
    - Die meisten blockorientierten Geräte unterstützen Direct Memory Access (DMA), um Daten ohne Prozessorbeteiligung zu übertragen

3 Möglichkeiten um Daten zu lesen

- Soll z.B. ein Datensatz von einer Festplatte gelesen werden, sind folgende Schritte nötig:
  1. Die CPU bekommt von einem Programm die Anforderung, einen Datensatz von einer Festplatte zu lesen
  2. Die CPU schickt dem Controller mit Hilfe des installierten Treibers einen entsprechenden I/O-Befehl
  3. Der Controller lokalisiert den Datensatz auf der Festplatte
  4. Die Anwendung erhält die angeforderten Daten
- Es gibt 3 Möglichkeiten, wie die Anwendung die Daten einliest:
  - **Busy Waiting** (geschäftiges bzw. aktives Warten)
  - **Interrupt-gesteuert**
  - **Direct Memory Access** (DMA)

Busy Waiting (geschäftiges bzw. aktives Warten)

- Der Treiber sendet die Anfrage an das Gerät und wartet in einer Endlosschleife, bis der Controller anzeigt, dass die Daten bereit stehen
- Stehen die Daten bereit, werden sie in den Speicher geschrieben und die Anwendung kann weiterarbeiten
- Beispiel: Zugriffsprotokoll **Programmed Input/Output** (PIO)
  - Die CPU greift mit Lese- und Schreibbefehlen auf die Speicherbereiche der Geräte zu und kopiert so Daten zwischen den Geräten und den Prozessorregistern
- **Vorteil**:
  - Leicht zu implementieren (keine zusätzliche Hardware nötig)
- **Nachteile**:
  - Belastet den Prozessor
  - Behindert die gleichzeitige Abarbeitung mehrerer Programme, da regelmäßig überprüft werden muss, ob die Daten bereit stehen

Interrupt-gesteuert

- Interrupt-Controller und Leitungen für das Senden der Interrupts
- Der Treiber initialisiert die Aufgabe und wartet auf einen Interrupt (Unterbrechung) durch den Controller $\Longrightarrow$ Der Treiber *schläft*
- CPU ist während des Wartens auf den Interrupt nicht blockiert und das Betriebssystem kann die anderen Anwendungen weiter abarbeiten
- Kommt es zum Interrupt, wird der Treiber dadurch *geweckt* und die CPU in ihrer momentanen Arbeit unterbrochen
  - Danach holt die CPU die Daten vom Controller ab und legt sie in den Speicher
  - Anschließend kann die CPU ihre unterbrochene Arbeit fortsetzen
- **Vorteile**:
  - CPU wird nicht blockiert
  - Gleichzeitige Abarbeitung mehrerer Programme wird nicht behindert
- **Nachteile**:
  - Zusätzliche Hardware nötig

Direct Memory Access (1/2)

- Zusätzlicher DMA-Baustein
  - Kann ohne Mithilfe der CPU, Daten direkt zwischen Arbeitsspeicher und Controller übertragen
  - Wird mit den nötigen Informationen (Speicherstelle, Anzahl der Bytes, Controller, usw.) initialisiert und erzeugt nach Beendigung einen Interrupt
- DMA wird nicht nur für den Zugriff auf Datenträger verwendet, sondern auch zum beschleunigten und CPU entlastenden Zugriff auf Peripheriegeräte wie zum Beispiel:
  - Soundkarten, ISDN-Karten, Netzwerkkarten, TV-/DVB-Karten
- Beispiel: Zugriffsprotokoll **Ultra-DMA** (UDMA)
  - Regelt, wie Daten zwischen dem Controller einer Festplatte und dem Arbeitsspeicher übertragen werden
  - Nachfolgeprotokoll des PIO-Modus
  - Daten werden unter Verwendung eines DMA-Controllers direkt vom bzw. zum Arbeitsspeicher zu übertragen, ohne dabei die CPU zu verwenden

Direct Memory Access (1/2)

- **Vorteile**:
  - Vollständige Entlastung der CPU
  - Gleichzeitige Abarbeitung mehrer Programme wird nicht behindert
- **Nachteile**:
  - Hoher Hardware-Aufwand
  - Neben dem Interrupt-Controller wird noch ein DMA-Baustein benötigt

Speicheradressierung und Speicherverwaltung

- Auf 16 Bit-Architekturen sind 2^{16} Speicheradressen und damit bis zu 65.536 Byte, also **64 Kilobyte** adressierbar
- Auf 32 Bit-Architekturen sind 2^{32} Speicheradressen und damit bis zu 4.294.967.296 Byte, also **4 Gigabyte** adressierbar
- Auf 64 Bit-Architekturen sind 2^{64} Speicheradressen und damit bis zu 18.446.744.073.709.551.616 Byte, also **16 Exabyte** adressierbar

! ! ! Frage ! ! !

Wie wird der Speicher eines Computers angesprochen und verwaltet?

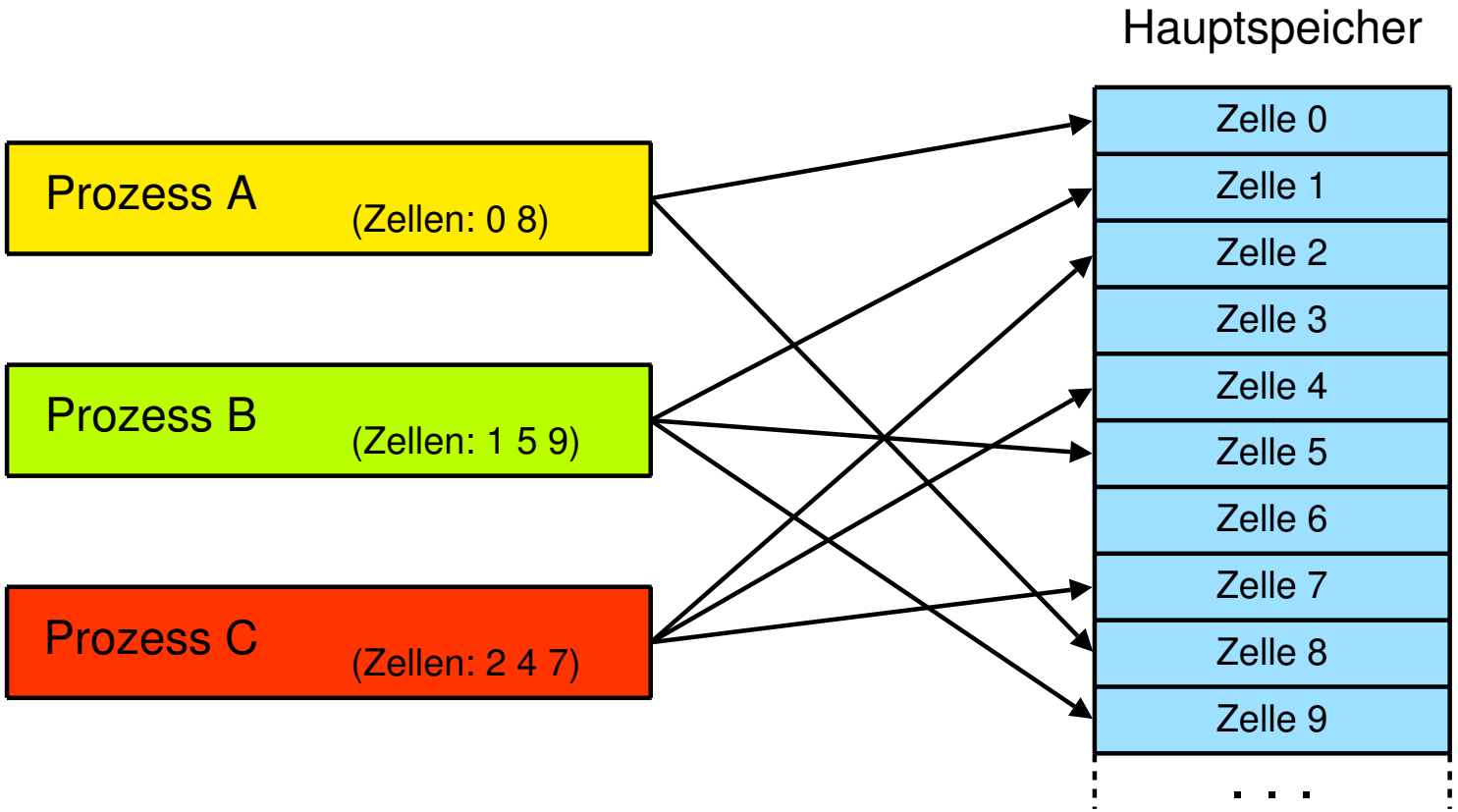
Speicheradressierung und Speicherverwaltung (2)

- Bei Einzelprogrammbetrieb (Singletasking) wird der Hauptspeicher in zwei Bereiche unterteilt
  - Ein Bereich für das Betriebssystem und einer für das aktuell ausgeführte Programm
- Beim Mehrprogrammbetrieb (Multitasking) muss der Benutzer-Bereich des Hauptspeichers weiter unterteilt werden, damit die gestarteten Prozesse im Speicher aufgenommen werden können
- Folgende Anforderungen sind zu beachten:
  - **Relokation**
  - **Schutz**
  - **Gemeinsame Nutzung**

Anforderungen and die Speicheradressierung/-verwaltung

- **Relokation**
  - Mehrere Prozesse befinden sich im Hauptspeicher
  - Werden Prozesse aus dem Hauptspeicher verdrängt, ist nicht bekannt, an welcher Stelle sie später wieder in den Hauptspeicher geladen werden
  - Bindungen an den alten Ort im Speicher sind ungünstig
  - Eine freie Ortswahl wäre vorteilhaft
  - Probleme sind u.a. bei Sprüngen und Referenzen auf Daten zu erwarten
- **Schutz**
  - Speicherbereiche müssen geschützt werden vor unbeabsichtigtem oder unzulässigem Zugriff durch anderen Prozesse
  - Darum müssen alle Zugriffe (durch die CPU) überprüft werden
- **Gemeinsame Nutzung**
  - Trotz Speicherschutz muss eine Kooperation der Prozesse mit gemeinsamem Speicher (Shared Memory) möglich sein
  - Es soll auch möglich sein, dass mehrere Prozesse das gleiche Programm ausführen und der Code nur einmal im Speicher ist

Idee: Direkter Zugriff auf die Speicherstellen



- Naheliegende Idee: Direkter Speicherzugriff durch die Prozesse
  $\Longrightarrow$ **Real Mode**
- Leider unmöglich in Multitasking-Systemen

Real Mode (Real Address Mode)

- Betriebsart x86-kompatibler Prozessoren
- Kein Zugriffsschutz
  - Jeder Prozess kann auf den gesamten Hauptspeicher und die Hardware zugreifen
  - Inakzeptabel für Multitasking-Betriebssysteme
- Die Bezeichnung wurde mit dem Intel 80286 eingeführt
- Im Real Mode greift der Prozessor wie ein Intel 8086-Prozessor auf den Hauptspeicher zu
- Jeder x86-kompatible Prozessor startet nach dem Reset im Real Mode
- Der Real Mode ist der Standardmodus für MS-DOS und dazu kompatible Betriebssysteme (u.a. PC-DOS, DR-DOS und FreeDOS)
- Neuere Betriebssysteme verwenden ihn nur noch während der Startphase und schalten dann in den Protected-Mode um

Protected Mode (Schutzmodus)

- Betriebsart x86-kompatibler Prozessoren
- Eingeführt mit dem Intel 80286
- Erlaubt die Begrenzung von Speicherzugriffsrechten
- Erhöht die Menge des direkt zugreifbaren Speichers
  - 16-Bit Protected Mode beim 80286 $\Longrightarrow$ 16 MB Hauptspeicher
  - 32-Bit Protected Mode beim 80386 $\Longrightarrow$ 4 GB Hauptspeicher
- Basiert auf dem Konzept des virtuellen Speichers
  - Mit der Memory Management Unit (MMU) kann einem Prozess jederzeit ein vollständiger Adressraum wie im Real Mode bereitgestellt werden
  - Virtuelle Speicheradressen werden von der CPU mit Hilfe der MMU in physische Speicheradressen übersetzt
- Jeder Prozess wird in seiner eigenen, von anderen Prozessen abgeschotteten Kopie des physischen Adressraums ausgeführt

Speicherpartitionierung

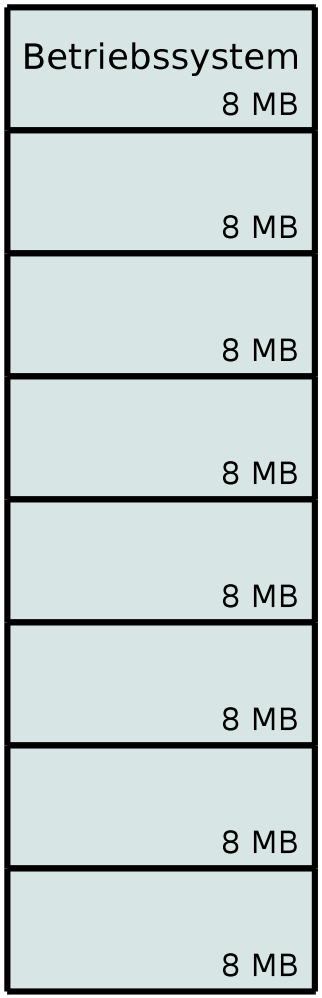
- 3 Konzepte zur Speicherpartitionierung:
  - **Statische Partitionierung**
  - **Dynamische Partitionierung**
  - **Buddy-Algorithmus**

Diese Konzepte werden von modernen Betriebssystemen nicht 1:1 angewendet und gelten als veraltet. Sie sind aber grundlegendes Wissen und das Fundament des virtuellen Speichers

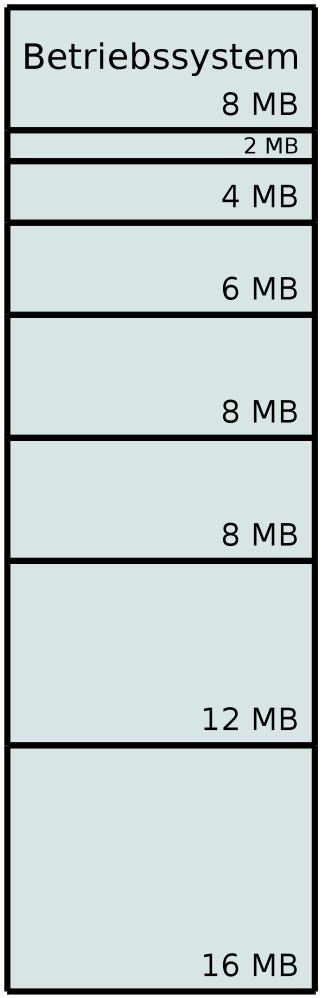
Statische Partitionierung

- Der Hauptspeicher wird in Partitionen gleicher oder unterschiedlicher Größe unterteilt
- Jeder Prozess, dessen Größe $\leq$ einer Partition ist, kann eine freie Partition erhalten
- Benötigt ein Prozess mehr Speicher, als eine Partition groß ist, muss es zum Overlay kommen
  - Dabei werden einzelne Daten eines Prozesses überschrieben
  - Entwickler müssen Anwendungen so schreiben, dass immer nur ein Teil der Daten im Speicher nötig ist und Daten überschrieben werden können
  - Dieses Vorgehen ist ineffizient und fehleranfällig
- Weitere Nachteile statischer Partitionierung:
  - Es kommt zwangsläufig zu interner Fragmentierung
    - Das Problem wird durch Partitionen unterschiedlicher Größe abgemildert, aber nicht gelöst
  - Anzahl der Partitionen limitiert die Anzahl möglicher Prozesse
- Mainframe-Betriebssystem IBM OS/MFT nutzte statische Partitionierung

Beispiel zur statischen Partitionierung



Partitionen
gleicher Größe



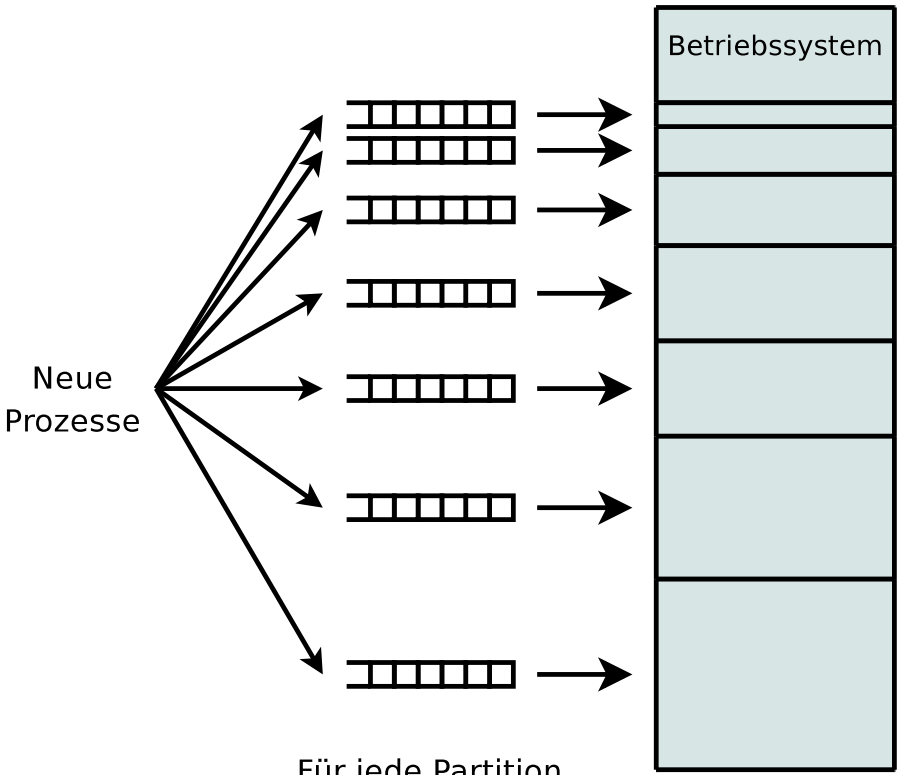
Partitionen
unterschiedlicher Größe

Quelle: Wiliam Stallings.
Operating Systems.
Prentice Hall. 2001

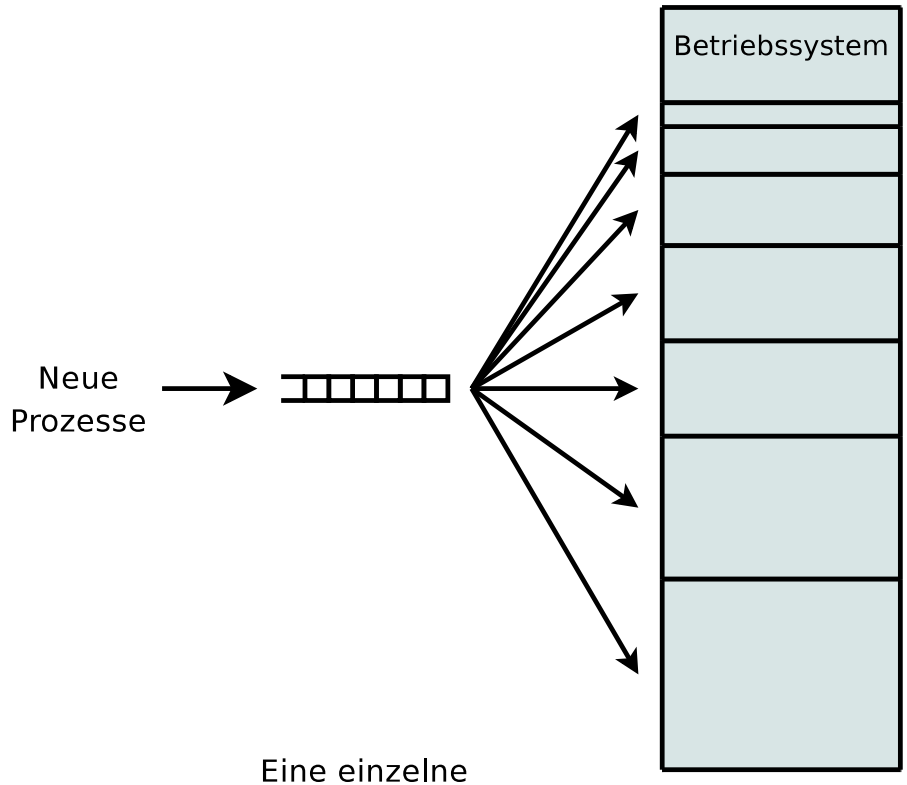
Speicherbelegungsverfahren (Statische Partitionierung)

- Die Speicherbelegung bei statischer Partitionierung ist sehr einfach
- Werden **Partitionen gleicher Größe** verwendet, ist es egal, welche freie Partition ein Prozess zugewiesen wird
  - Sind alle Partitionen belegt, muss ein Prozess aus dem Hauptspeicher verdrängt werden. Die Entscheidung welcher Prozess verdrängt wird, hängt vom verwendeten Scheduling-Verfahren ab
- Werden **Partitionen unterschiedlicher Größe** verwendet, gibt es 2 Möglichkeiten um Prozessen Partitionen im Speicher zuzuweisen
  - Prozesse sollen eine möglichst passgenaue Partition erhalten, damit wenig interne Fragmentierung entsteht. Es ist eine eigene Prozess-Warteschlange für jede Partition nötig. In jeder Warteschlange werden die ausgelagerten Prozesse abgebildet. Bei diesem Verfahren kann es vorkommen, dass bestimmte Partitionen nie genutzt werden
  - Eine einzelne Warteschlange für alle Partitionen. So kann die Zuweisung der Partitionen an die Prozesse besser kontrolliert werden

Speicherbelegungsverfahren



Für jede Partition
eine eigene
Prozess-Warteschlange



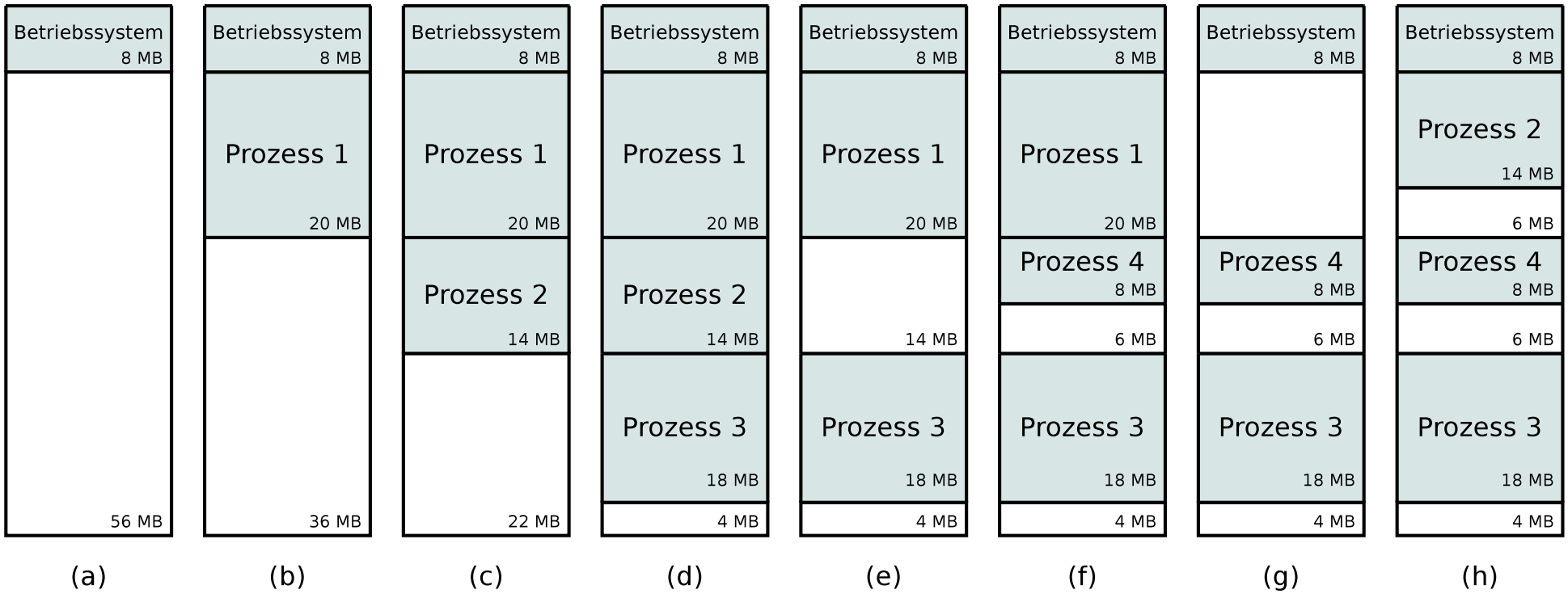
Eine einzelne
Prozess-Warteschlange
für alle Partitionen

Quelle: Wiliam Stallings. Operating Systems. Prentice Hall. 2001

Dynamische Partitionierung

- Der verfügbare Hauptspeicher wird in zusammenhängende Partitionen unterschiedlicher Größe unterteilt
- Jeder Prozess erhält eine zusammenhängende Partition mit exakt der notwendigen Größe
- Durch das unvermeidliche Einlagern und Verdrängen von Prozessen kommt es zu externer Fragmentierung
- Eine Möglichkeit um diese Fragmentierung zu beheben ist Kompaktierung (Defragmentierung)
  - Voraussetzung: Verschiebbarkeit von Speicherblöcken
  - Verweise in den Applikationen dürfen durch ein Verschieben von Datenblöcken nicht ungültig werden
- Die Mainframe-Betriebssysteme OS/MVT und OS/MFT von IBM nutzten in den 60er Jahren dynamische bzw. statische Partitionierung

Beispiel zur dynamischen Partitionierung

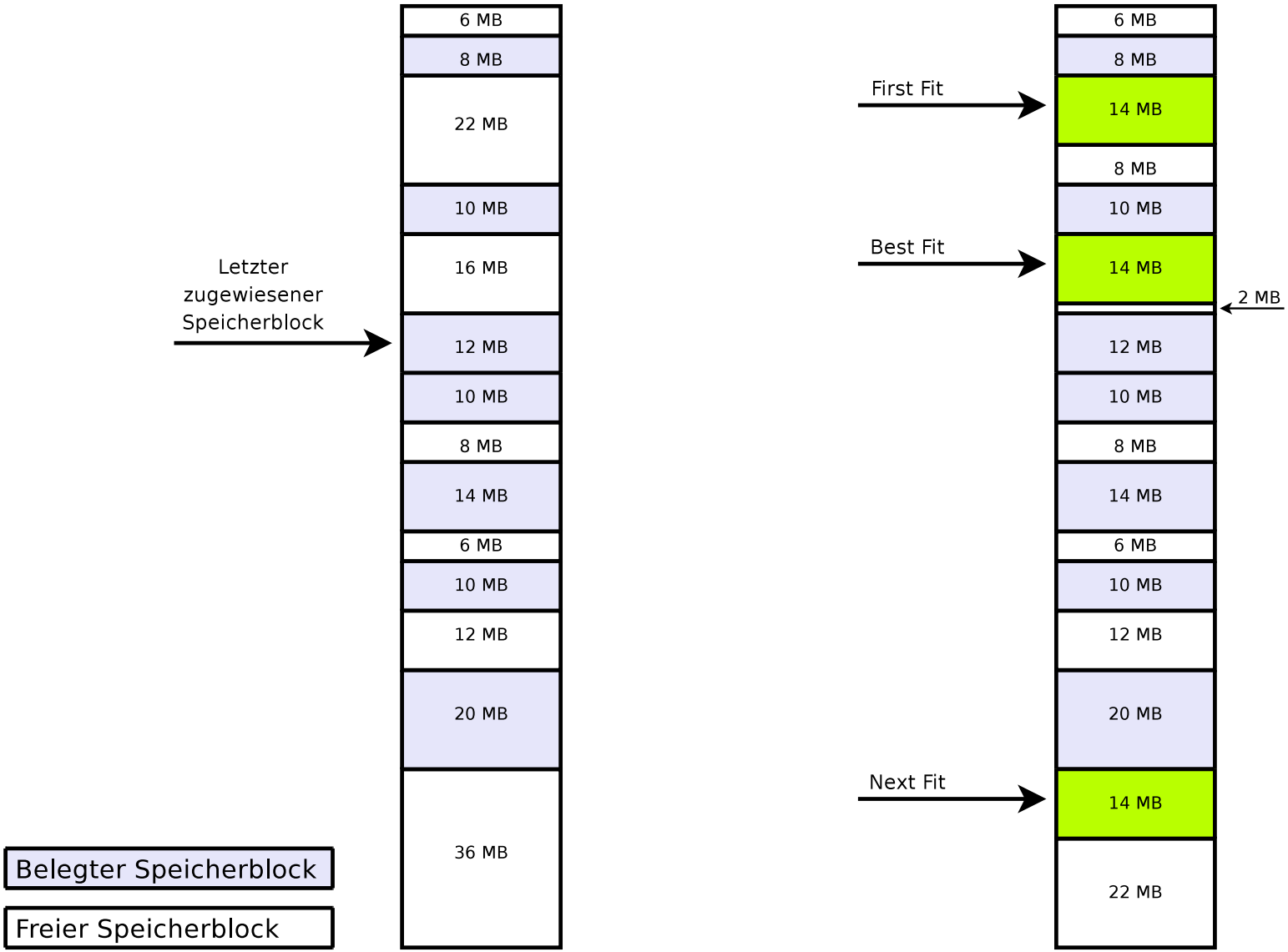


Quelle: Wiliam Stallings. Operating Systems. Prentice Hall. 2001

Speicherbelegungsverfahren (Dynamische Partitionierung)

- **First Fit**
  - Sucht vom Speicheranfang nach einem passenden freien Block
  - Einfachstes und schnellstes Verfahren
- **Next Fit**
  - Sucht ab der Stelle der letzten Blockzuweisung nach einem passenden freien Block
  - Etwas schlechter als First Fit. Zerstückelt schnell den großen Bereich freien Speichers am Ende des Adressraums
- **Best Fit**
  - Sucht einen Block mit optimaler Größe
  - Produziert viele Minifragmente und arbeitet am langsamsten
- **Worst Fit**
  - Sucht den größten freien Block
- **Random**
  - Liefert manchmal sehr gute Ergebnisse und ist schnell

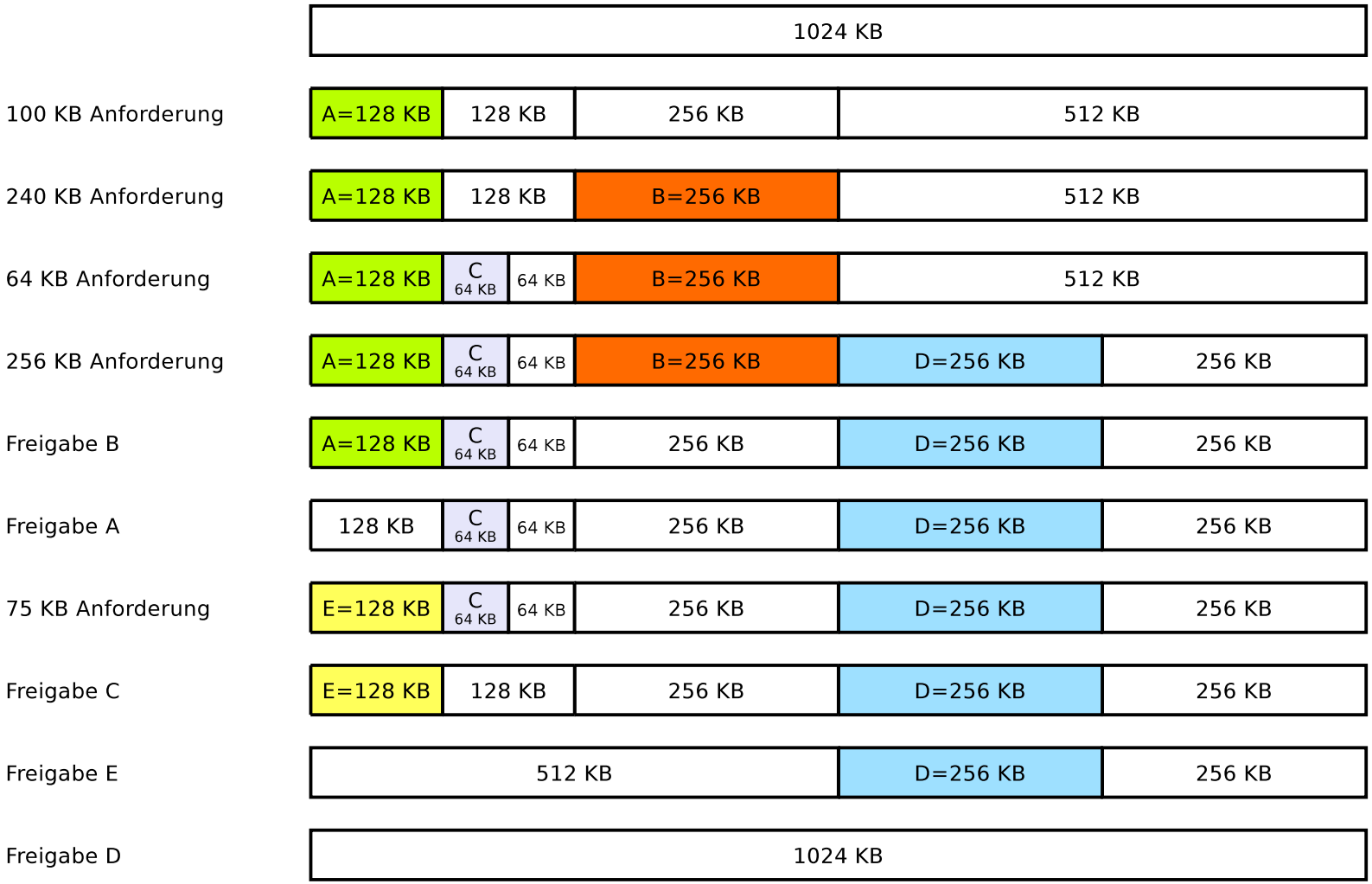
Beispiel zum Speicherbelegungsverfahren



Buddy-Verfahren von Donald E. Knuth

- Versucht die Vorteile fester und dynamischer Partitionierung zu nutzen
- Speicher wird in Bereiche der Länge 2^K mit $L \leq K \leq U$ aufgeteilt
  - 2^L ist der kleinste Block der zugewiesen wird
  - 2^U ist der größte Block der zugewiesen wird (der komplette Speicher)
- Zu Beginn gibt es nur einen Block, der den gesamten Speicher abdeckt
- Fordert ein Prozess eine Speicher an, wird zur nächsthöheren Zweierpotenz aufgerundet und ein entsprechender, freier Block gesucht
  - Existiert noch kein Block dieser Größe, wird nach einem Block doppelter Größe gesucht und dieser in zwei Hälften (sogenannte *Buddies*) unterteilt
    - Einer der beiden Blöcke wird dann dem Prozess zugewiesen
  - Existiert auch kein Block doppelter Größe, wird ein Block vierfacher Größe gesucht, usw. . .
- Wird Speicher freigegeben, wird geprüft, ob zwei Hälften gleicher Größe sich wieder zu einem größeren Block zusammenfassen lassen
- Die Implementierungsvarianten von Knuth basieren darauf, dass das Betriebssystem für jede möglich Segmentgröße eine Frei-Liste verwaltet

Beispiel zum Buddy-Verfahren



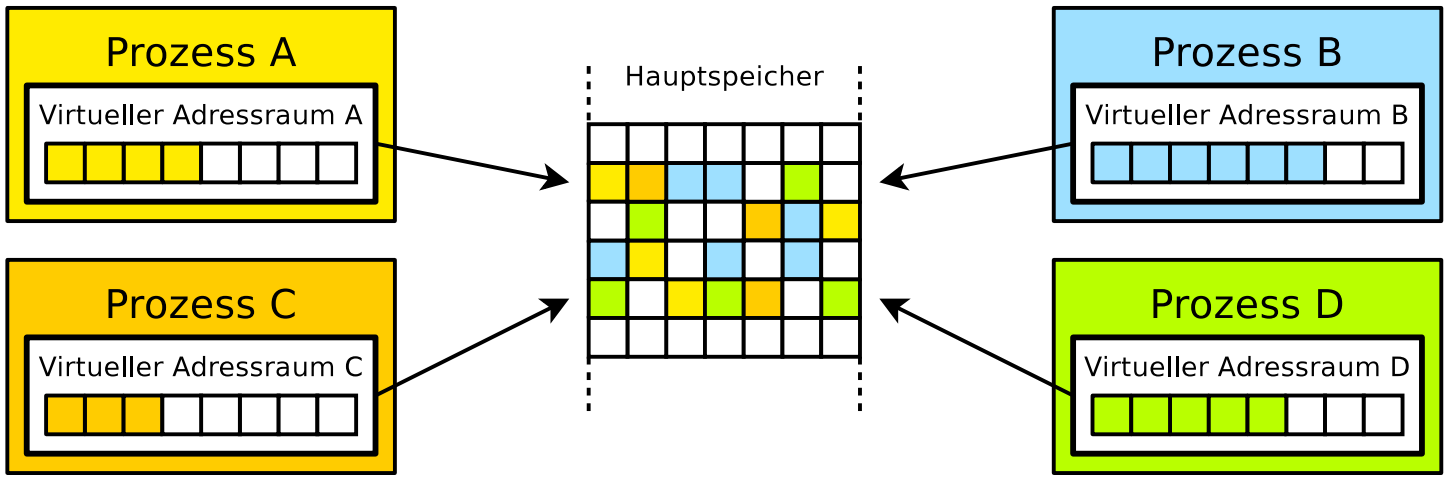
- Nachteil: Interner und externer Verschnitt

Virtueller Speicher (1)

- Moderne Betriebssysteme arbeiten im Protected Mode (Schutzmodus)
- Im Protected Mode unterstützt der Prozessor 2 Verfahren zur Speicheradressierung und Speicherverwaltung
  - **Segmentierung** existiert ab dem 80286
  - **Paging** existiert ab dem 80386
  - Die Verfahren sind Implementierungsvarianten des **virtuellen Speichers**
- Prozesse verwenden keine realen Hauptspeicheradressen
  - Das würde bei Multitasking-Systemen zu Problemen führen
- Stattdessen besitzt jeder Prozess einen **Adressraum**
  - Der Adressraum ist eine Abstraktion des physischen Speichers
  - Der Adressraum ist der von der verwendeten Speichertechnologie und den gegebenen Ausbaumöglichkeiten unabhängige **virtuelle Speicher**
  - Jeder Adressraum besteht aus Speicherstellen, die von der Adresse 0 (logische Adressen) an, aufwärts durchnummeriert sind

Virtueller Speicher (2)

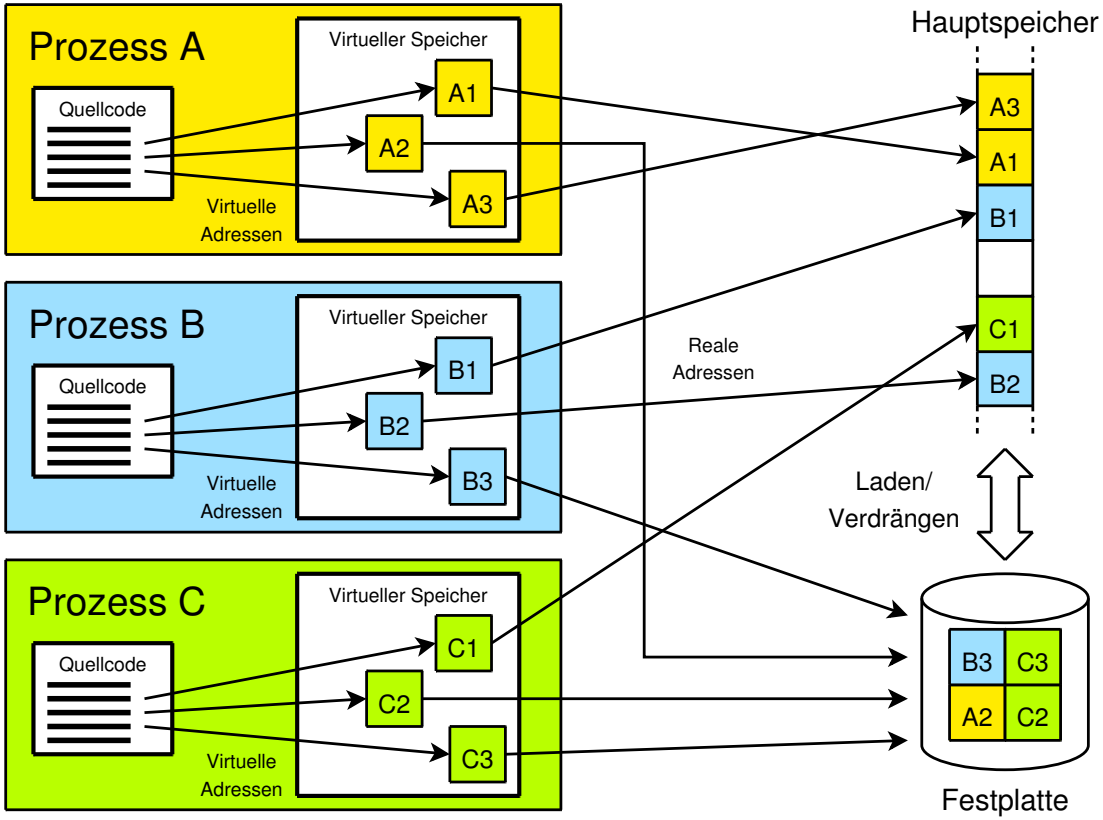
- Adressräume können nach Bedarf erzeugt oder gelöscht werden und sind voneinander abgeschottet und damit geschützt
- Kein Prozess kann nicht ohne vorherige Vereinbarung auf den Adressraum eines anderen Prozesses zugreifen
- **Mapping** = Abbilden des virtuellen Speichers auf den realen Speicher



- Dank virtuellem Speicher wird der Hauptspeicher besser ausgenutzt
  - Die Prozesse müssen nicht am Stück im Hauptspeicher liegen
  - Darum ist die Fragmentierung des Hauptspeichers kein Problem

Virtueller Speicher (3)

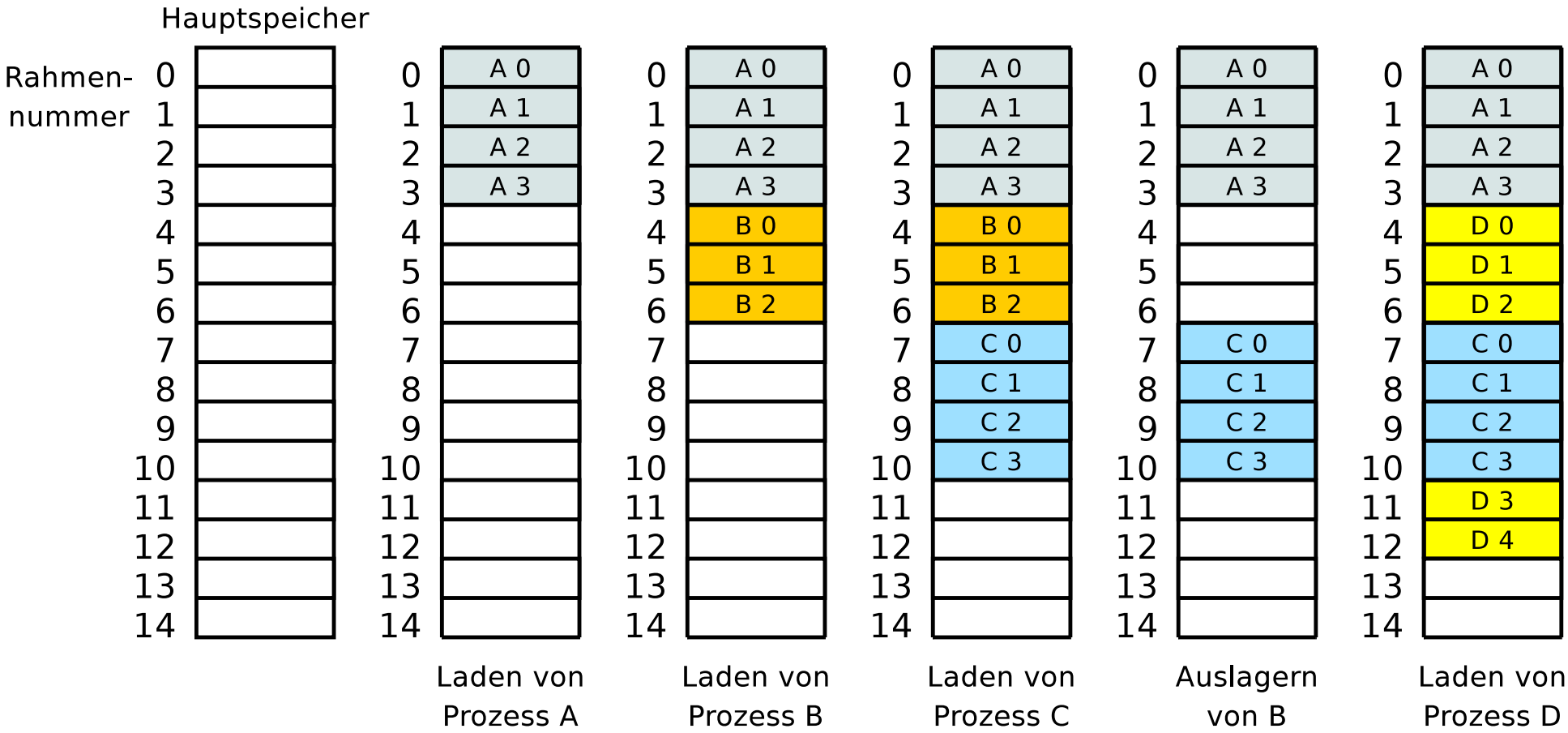
- Durch virtuellen Speicher kann mehr Speicher angesprochen und verwendet werden, als physisch im System vorhanden ist
- **Auslagern** (**Swapping**) geschieht für die Benutzer und Prozesse transparent



Paging: Seitenorientierter Speicher

- Teile eines Prozesses ($\Longrightarrow$ **Seiten**) werden freien Speicherbereichen im Hauptspeicher ($\Longrightarrow$ **Rahmen** bzw. **Kacheln**) zugewiesen
- Alle Seiten haben die gleiche Länge
  - Länge einer Seite ist üblicherweise zwischen 256 Byte und 8 KB
- Externe Fragmentierung gibt es beim Paging nicht
- Interne Fragmentierung kommt in der letzten Seite eines Prozesses vor
- Das Betriebssystemen verwaltet für jeden Prozess eine Seitentabelle
  - In dieser steht, wo sich die einzelnen Seiten des Prozesses befinden
- Prozesse arbeiten nur mit virtuellen Speicheradressen
- Virtuelle Speicheradressen bestehen aus 2 Teilen
  - Der werthöhere Teil repräsentiert die Seitennummer
  - Der wertniedrigere Teil repräsentiert den Offset
- Der Offset repräsentiert das Speicherwort innerhalb einer Seite
- Länge einer virtuellen Adresse ist architekturabhängig und ist in der Regel 16, 32 oder 64 Bit

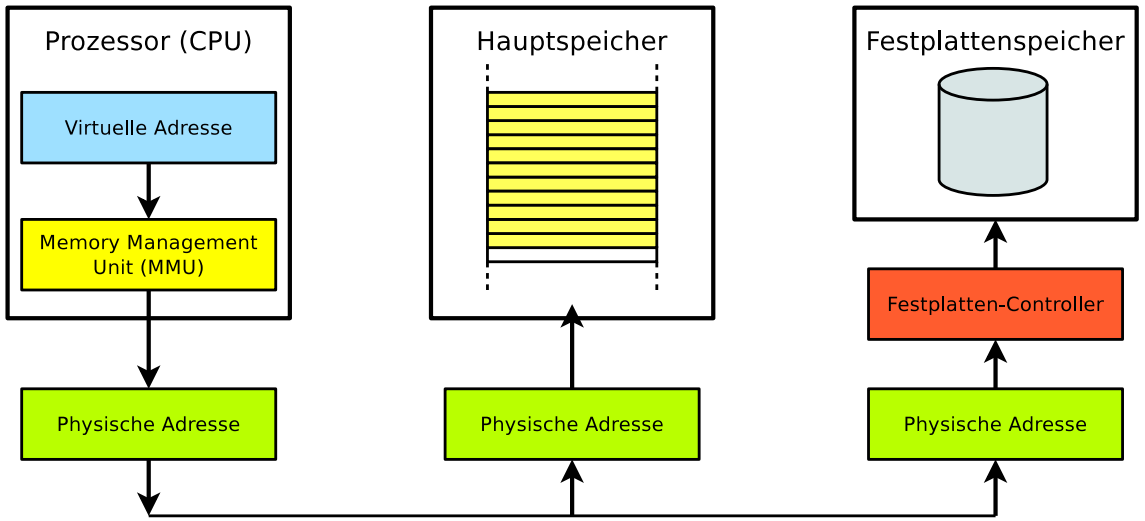
Zuweisung von Prozessseiten zu freiem Speicher (Rahmen)



Quelle: Wiliam Stallings. Operating Systems. Prentice Hall. 2001

Adressumwandlung durch die MMU

- Paging übersetzt logische (virtuelle) Speicheradressen von der CPU mit der MMU und der Seitentabelle in physische Adressen
- Das Betriebssystem prüft dann, ob sich die physische Adresse im Hauptspeicher, oder auf der Festplatte befindet
  - Befinden sich die Daten auf der Festplatte, muss das Betriebssystem die Daten in den Hauptspeicher einlesen
  - Ist der Hauptspeicher voll, muss das Betriebssystem andere Einträge aus dem Hauptspeicher verdrängen



Implementierung der Seitentabelle

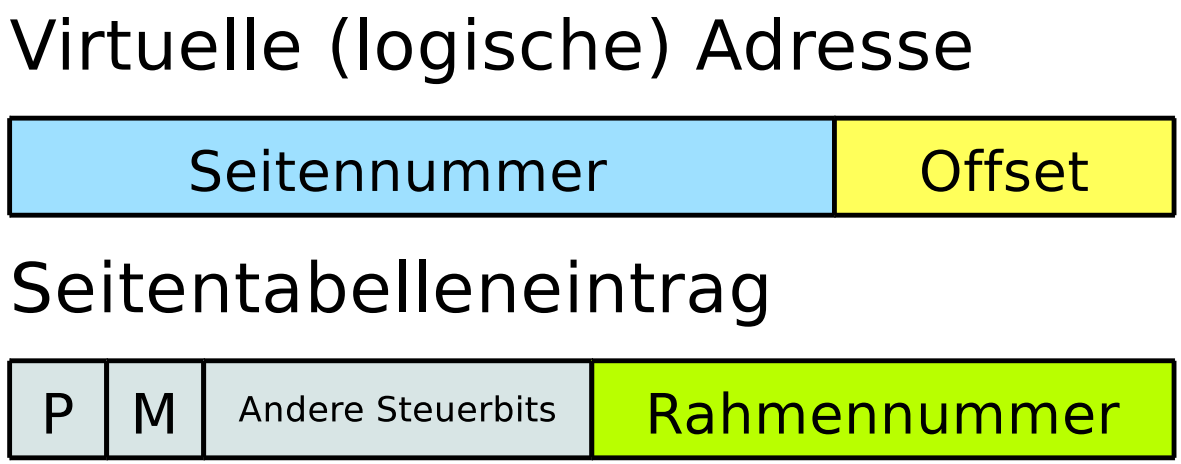
- Die Länge der Seiten hat Auswirkungen:
  - **Kurze Seiten**: Geringer interner Verschnitt, aber lange Seitentabelle
  - **Lange Seiten**: Kurze Seitentabelle, aber hoher interner Verschnitt
- Ist die Seitentabelle des aktuell laufenden Prozesses kurz, wird sie in Registern in der CPU abgelegt
  - Das führt zu einer hohen Geschwindigkeit, da zur Berechnung der physischen Speicheradressen nur Registermanipulationen notwendig sind
- Bei kleinen Seiten kann die Seitentabelle sehr groß werden
  - In diesem Fall wird die Seitentabelle im Hauptspeicher abgelegt

2 Register ermöglichen der MMU den Zugriff auf die Seitenabelle

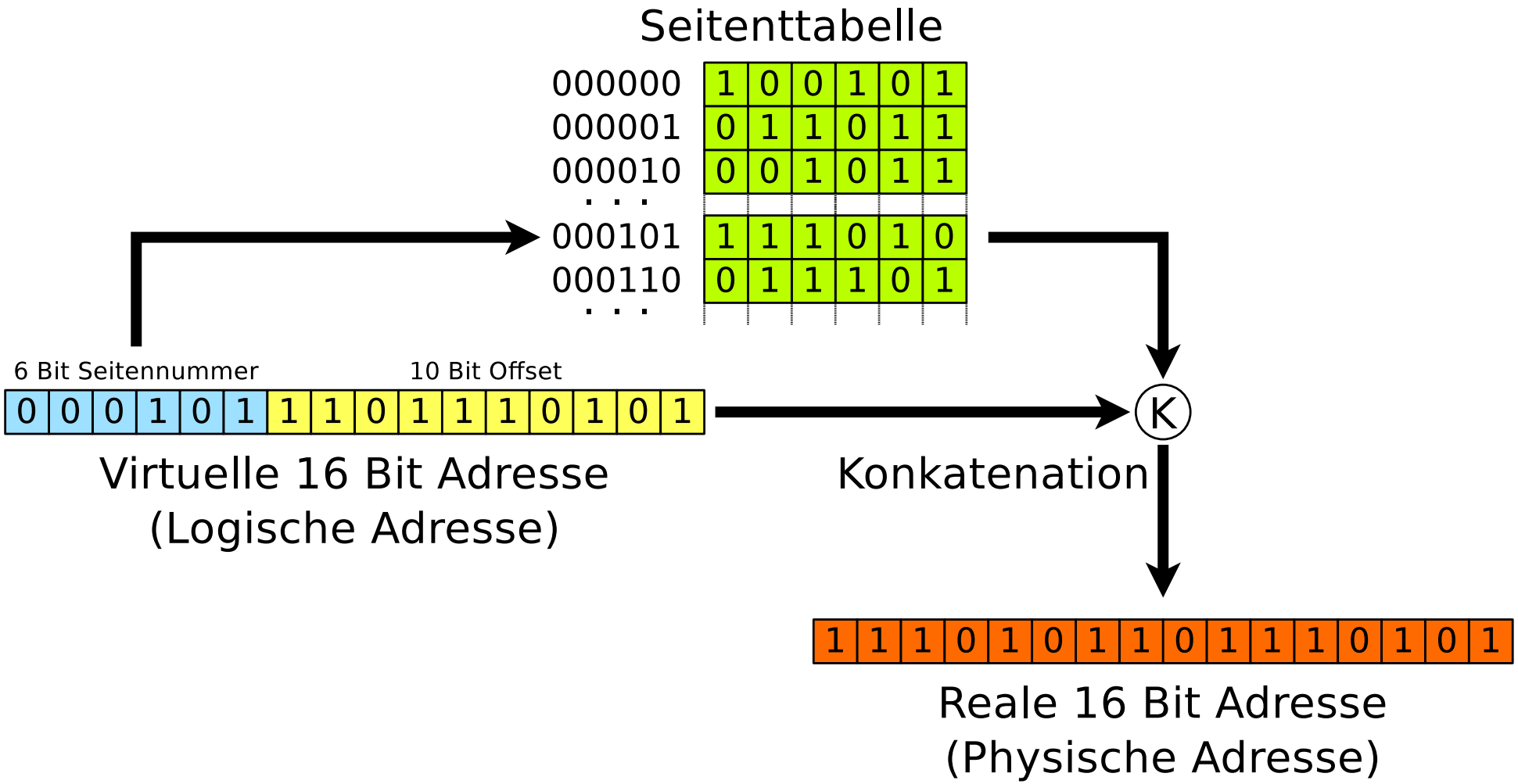
- **Page-table base register** (PTBR): Zeigt auf die Seitentabelle des aktuell laufenden Prozesses
- **Page-table length register** (PTLR): Gibt die Länge der Seitentabelle des aktuell laufenden Prozesses an

Seitentabellenstruktur

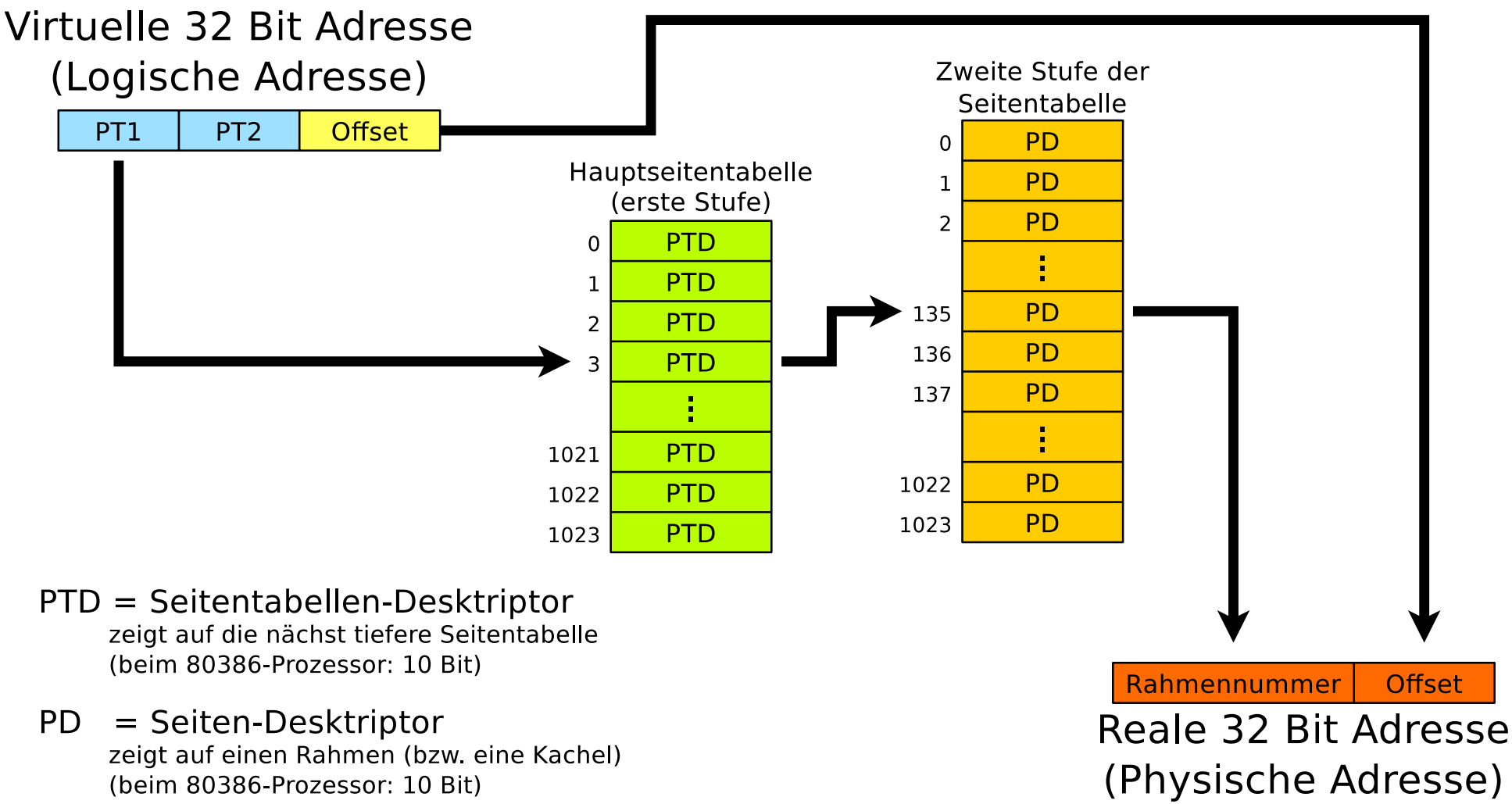
- Die Seitentabellenstruktur beim Paging ist so, dass jeder Seitentabelleneintrag folgende Komponenten enthält:
  - **Present-Bit**: Legt fest, ob die Seite im Hauptspeicher ist
  - **Modify-Bit**: Legt fest, ob die Seite verändert wurde
  - **Andere Steuerbits**: Hier werden Schutzrechte und Rechte zur gemeinsamen Nutzung festgelegt
  - **Rahmennummer**: Wird mit dem Offset der virtuellen (logischen) Adresse verknüpft



Adressumwandlung beim Paging (einstufig)



Adressumwandlung beim Paging (zweistufig)



PTD = Seitentabellen-Desktriptor
zeigt auf die nächst tiefere Seitentabelle
(beim 80386-Prozessor: 10 Bit)

PD = Seiten-Desktriptor
zeigt auf einen Rahmen (bzw. eine Kachel)
(beim 80386-Prozessor: 10 Bit)

Einige Berechnungen zum seitenorientierten Speicher

- Seitenlänge beim Paging:
  - **Kurze Seiten**: Geringer interner Verschnitt, aber lange Seitentabelle
  - **Lange Seiten**: Kurze Seitentabelle, aber hoher interner Verschnitt
- Sei
  - s die durchschnittliche Prozessgröße in Byte
  - p die Seitenlänge in Byte
  - e ein Eintrag in der Seitentabelle in Byte
- Dann gilt:
  - Ein Prozess belegt $\frac{s}{p}$ Seiten und damit $\frac{es}{p}$ Byte in der Seitentabelle
    $\frac{es}{p}$ wird kleiner mit wachsender Seitenlänge
  - Durch interne Fragmentierung gehen $\frac{p}{2}$ Byte verloren
    $\frac{p}{2}$ wird kleiner mit schrumpfender Seitenlänge
- Somit ist der absolute Gesamtverlust (in Byte): $v_{abs} = \frac{es}{p} + \frac{p}{2}$
- Somit ist der relative Gesamtverlust (in Prozent): $v_{rel} = \frac{1}{s}(\frac{es}{p} + \frac{p}{2})$
- Durch Minimierung ergibt sich als optimale Seitenlänge: $p_{opt} = \sqrt{2se}$

Ein Beispiel

- Durchschnittliche Prozessgröße $= s = 1\,\text{MB}$
- Größe eines Eintrags in der Seitentabelle $= e = 8\,\text{Byte}$
- Optimale Seitenlänge $= p_{opt} = \sqrt{2se} = 4\,\text{KB}$

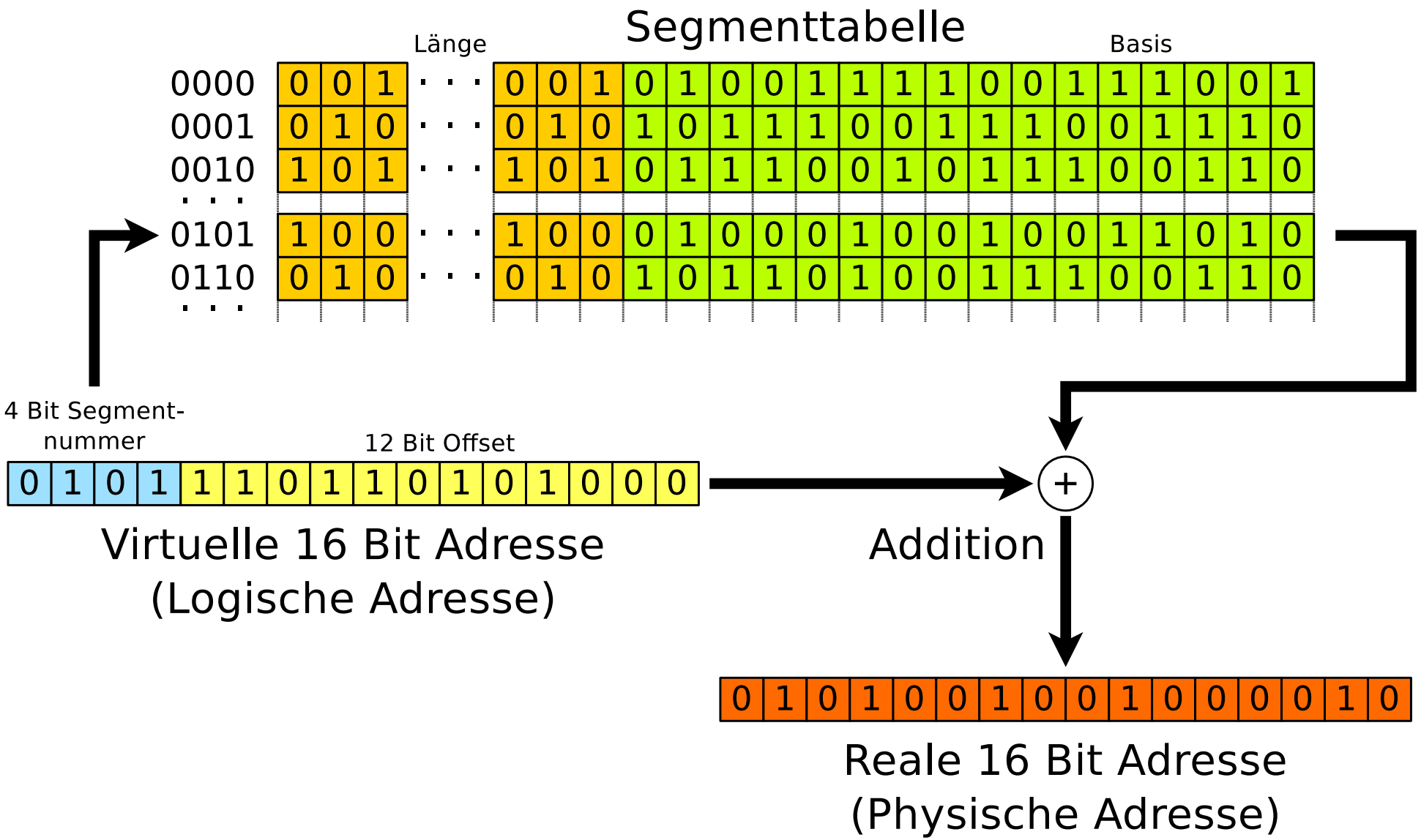
Durchschnittliche Prozessgröße	Optimale Seitenlänge	Relativer Gesamtverlust
$s = 10\,KB$	$p_{opt} = 400\,Byte$	$v_{rel} = 4\,\%$
$s = 1\,MB$	$p_{opt} = 4\,KB$	$v_{rel} = 0,4\,\%$
$s = 100\,MB$	$p_{opt} = 40\,KB$	$v_{rel} = 0,04\,\%$
$s = 10000\,MB$	$p_{opt} = 400\,KB$	$v_{rel} = 0,004\,\%$
$s = 1000000\,MB$	$p_{opt} = 4\,MB$	$v_{rel} = 0,0004\,\%$

- Wählt man die optimale Seitenlänge, nimmt der Speicherverlust mit zunehmender Prozessgröße ab

Segmentierung

- Weitere Methode um virtuellen Speicher zu verwalten
- Der virtuelle Speicher eines Prozesses besteht aus Einheiten (Segmenten) unterschiedlicher Länge
- Maximale Segmentlänge bestimmt der Offset der virtuellen Adressen
  - Im Beispiel ist der Offset 12 Bit $\Longrightarrow 2^{12} = 4096$ Bit
- Das Betriebssystemen verwaltet für jeden Prozess eine Segmenttabelle
  - Jeder Eintrag in der Segmenttabelle enthält die Startadresse des Segments im Hauptspeicher und dessen Länge
  - Virtuelle (logische) Adressen werden mit Hilfe der Segmenttabelle, die jeder Prozess besitzt, in reale (physische) Adressen umgerechnet
- Interne Fragmentierung gibt es bei Segmentierung nicht
- Es kommt zu externer Fragmentierung wie bei dynamischer Partitionierung
  - Diese ist allerdings nicht so ausgeprägt

Adressumwandlung bei Segmentierung



Segmenttabellenstruktur

- Jeder Eintrag der Segmenttabelle enthält folgende Komponenten:
  - **Present-Bit**: Legt fest, ob die Seite im Hauptspeicher ist
  - **Modify-Bit**: Legt fest, ob die Seite verändert wurde
  - **Andere Steuerbits**: Hier werden Schutzrechte und Rechte zur gemeinsamen Nutzung festgelegt
  - **Länge**: Länge des Segments
  - **Rahmennummer**: Wird mit dem Offset der virtuellen (logischen) Adresse verknüpft

Virtuelle (logische) Adresse

Segmenttabelleneintrag

Nächste Vorlesung

Nächste Vorlesung:

24.11.2011