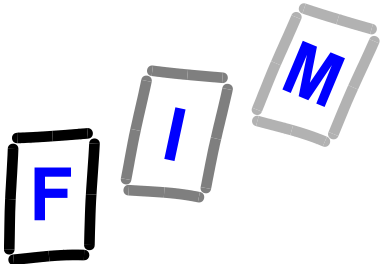


Betriebssysteme

I: Speicherverwaltung

(Teil B: Hauptspeicherverwaltung)

Logische Adressen
Physische Adressen

- **Dieser Aspekt ist in der Folge sehr wichtig**

- **Grund:**
  Verschiedene Sichten

  was Adressen sind

  und

  wie man Speicher (Hauptspeicher) verwaltet

Logische ↔ Physische Adressen

- **Logische Adresse =**
  Adresse aus Programmsicht,
  egal wo sich das Programm oder Teile davon im Speicher befinden (werden)

- **Physische Adresse =**
  tatsächliche Adresse zur Ansteuerung der Speicherbänke (RAM)

Logische ↔ Physische Adressen (2)

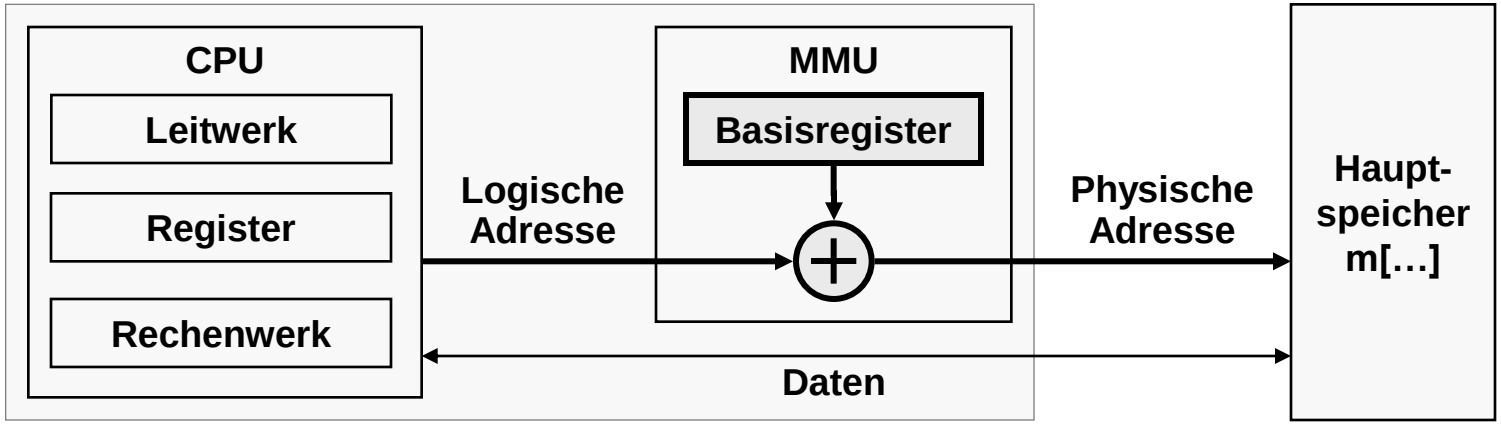
- **Bei der Adressbindung zur Übersetzungszeit sind die tatsächlichen Adressen gleich den logischen Adressen**
- **Ev. auch bei der Adressbindung zur Ladezeit, aber der Lader kann die Adressen anpassen**

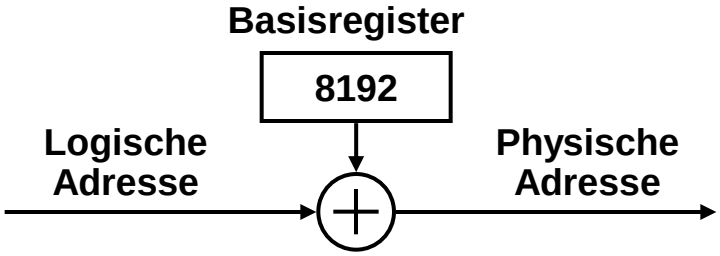
- **Bei der Zuweisung zur Laufzeit muss es möglich sein, dass logische Adressen ungleich den tatsächlichen Adressen im Speicher sein können: „Binden während der Ausführung“**

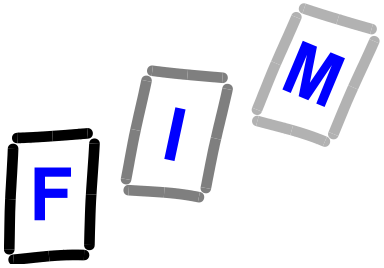
Logische ↔ Physische Adressen (3)

- **Logischer Adressraum:**
  Menge aller logischen Adressen
  - **Hängt von der „Wortlänge n“ ab: 2^n**
  - **Hängt von der BS Architektur ab: Maximaler Platz, der für Programme zur Ausführung bereitgestellt wird**
- **Physischer Adressraum:**
  Menge aller physisch verfügbaren Adressen
  - **Hängt von Speichergröße ab: 2^m mit $m \leq n$**
- **Daher:**
  Adressraum aus Sicht des Programmierers wird i.A. vom tatsächlich verfügbaren physischen Adressraum abweichen!

Adressbindung mit Hardwareunterstützung





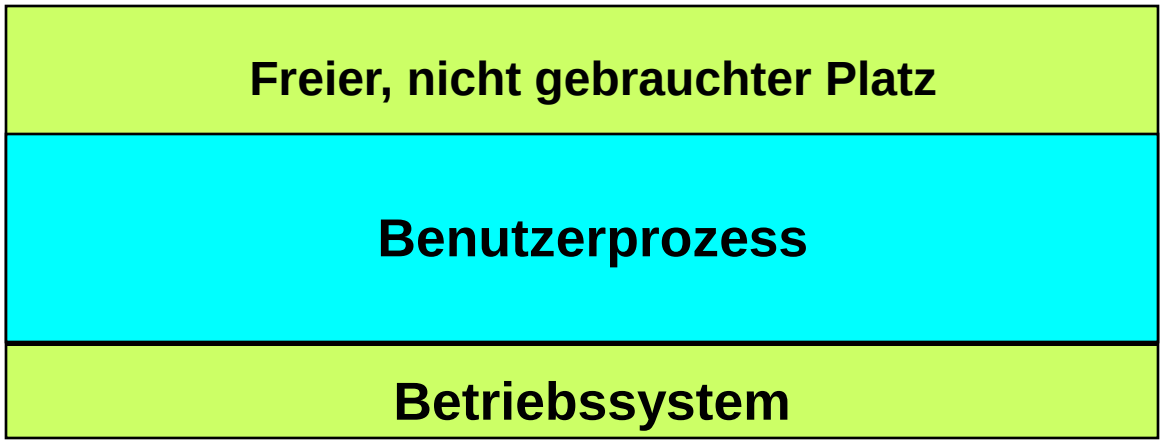


Hauptspeicher-Zuweisung

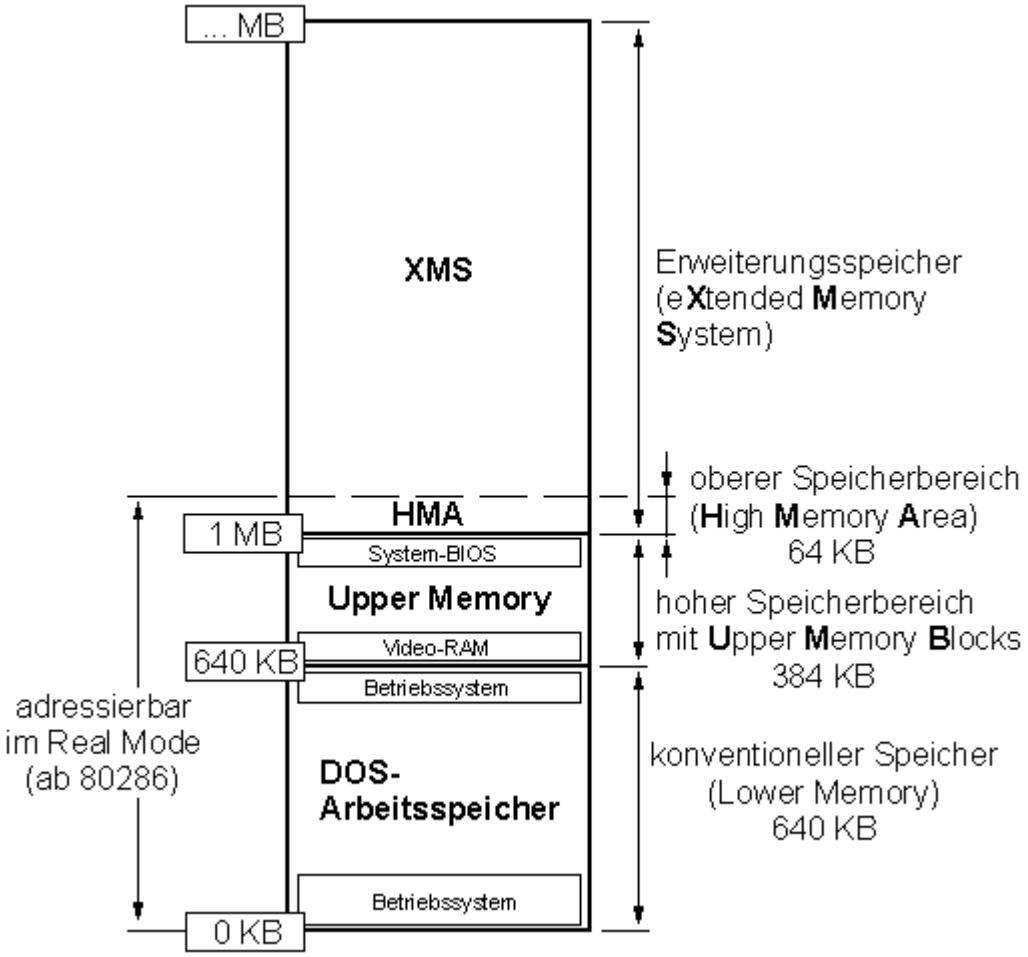
- **Das Betriebssystem selbst und jedes laufende, im Speicher liegende Programm benötigt Speicher für Code und Daten.**
- **Aufgaben des BS:**
  - **Verwaltung des verfügbaren Speichers**
  - **Zuweisung an Programme**
  - **Schutzmechanismen gegen unerlaubten Zugriff auf „fremden“ Speicher**
    - **Schutz des BS-Codes**
    - **Schutz der Programme vor Zugriff / Änderung durch andere Programme**

Zuweisung von kontinuierlichem Speicher (contiguous memory allocation)

- **Ein einfacher Ansatz ist:**
  - **Teile den Speicher in zwei Partitionen :**
    - **Eine für das BS**
    - **Eine für den Benutzerprozess**

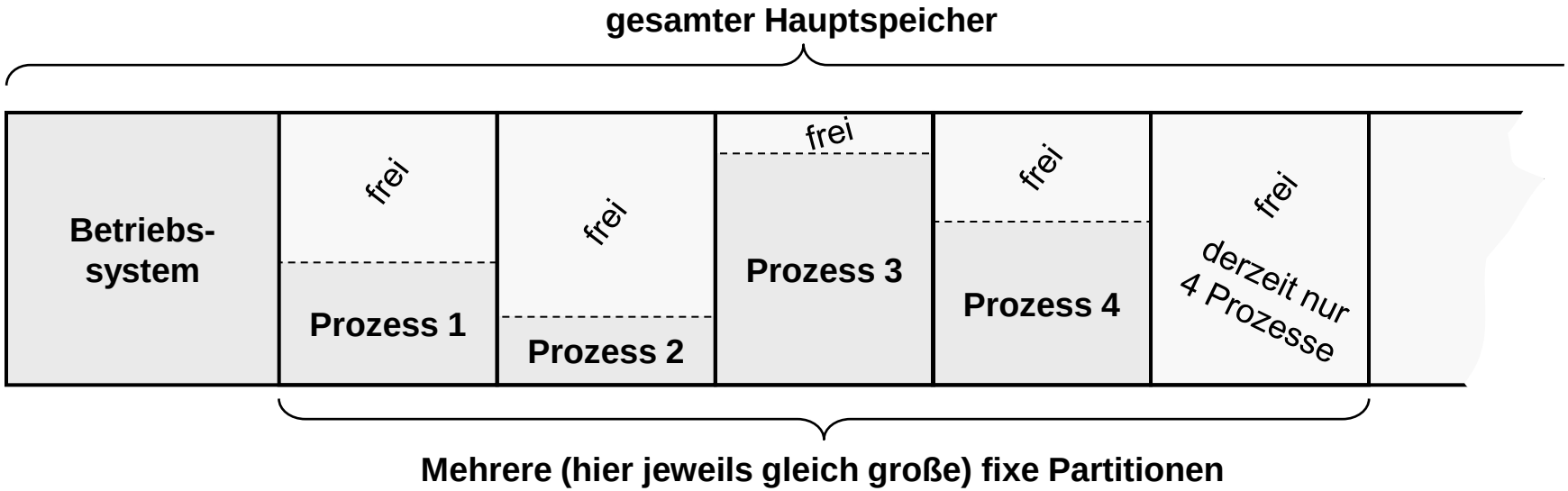


Layout: DOS


... MB
XMS
Erweiterungsspeicher
(eXtended Memory
System)
oberer Speicherbereich
(High Memory Area)
64 KB
HMA
1 MB
System-BIOS
Upper Memory
hoher Speicherbereich
mit Upper Memory Blocks
384 KB
Video-RAM
640 KB
Betriebssystem
adressierbar
im Real Mode
(ab 80286)
DOS-
Arbeitsspeicher
konventioneller Speicher
(Lower Memory)
640 KB
Betriebssystem
0 KB

Zuweisung von kontinuierlichem Speicher

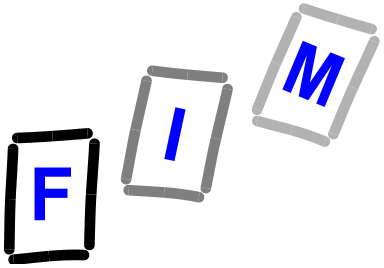
- **Ein einfacher Ansatz ist:**
  - **Teile den Speicher in n > 2 Partitionen bestimmter Größe**
    - **Eine für das BS**
    - **n-1 für die Benutzerprozesse P_i**



Zuweisung von kontinuierlichem Speicher

Problem:

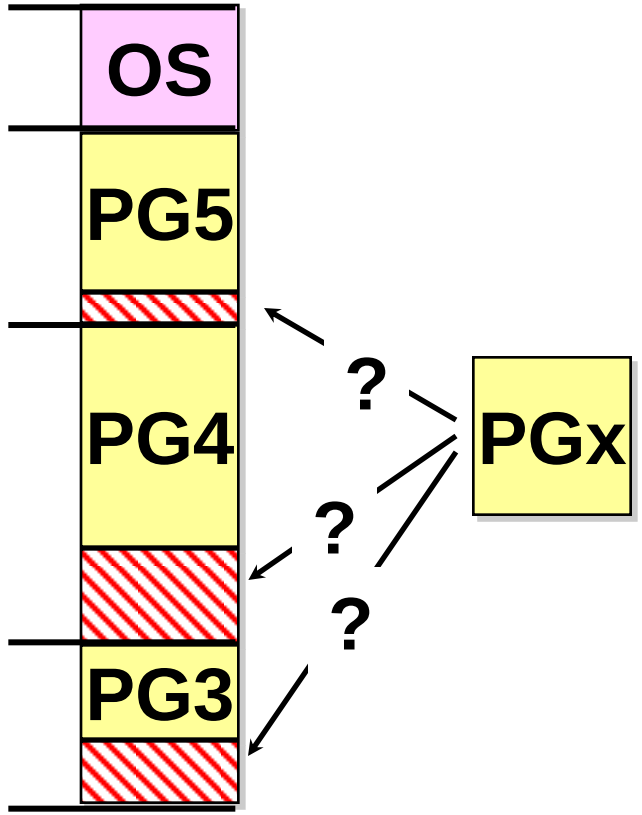
- **Prozesse P_i sind verschieden groß**
- **Daher:**
  - **Eine Partition könnte zu groß sein, der übrige Speicher bleibt ungenützt**
  - **Eine Partition kann zu klein sein,**
  - **Ein Prozess P_i kann nicht ausgeführt werden, da keine Partition mehr frei ist, obwohl z.B. zwei Partitionen zusammen genügend freien Platz für P_i hätten**



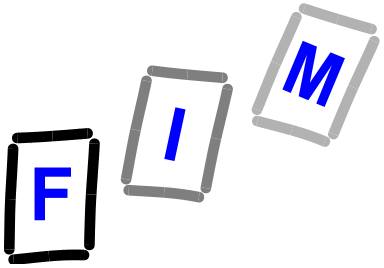
Speicherfragmentierung

- **Beim Laden und Entfernen von Prozessen aus dem Speicher wird der freie Speicher in verschieden große Stücke zerteilt**
- **Nennt man *Speicherfragmentierung***
  - ***Externe Fragmentierung***
  - ***Interne Fragmentierung***

Externe Fragmentierung



Obgleich in Summe noch genügend Platz für Programm PGx im Speicher wäre, kann durch die externe Fragmentierung das Programm nicht mehr geladen werden

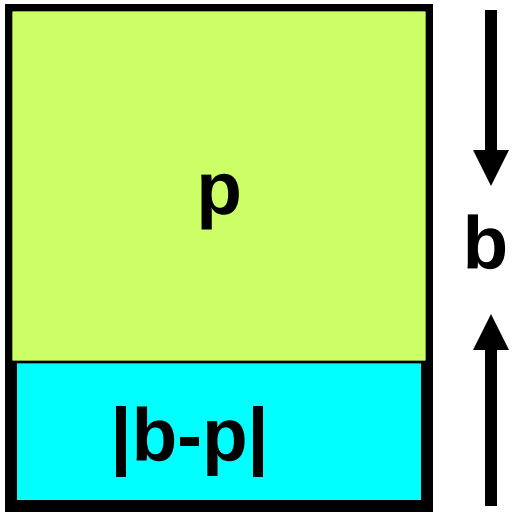


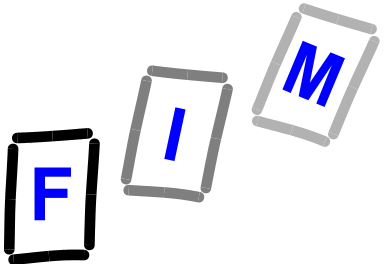
Externe Fragmentierung

- **Externe Fragmentierung:**
  Genug Speicher ist insgesamt vorhanden, aber der freie Speicher ist nicht kontinuierlich
- **Der Speicher wurde in eine große Anzahl von kleinen Blöcken zerstückelt, von denen keiner groß genug für die Aufnahme von Programm P_i ist**
- **Lösung: Verdichtungsstrategie**
  (engl.: compacting strategy)

Interne Fragmentierung

- **Interne Fragmentierung**
  Prozess P braucht einen Speicherplatz der Größe p und erhält einen Block der Größe b > p. Die Differenz |b-p| geht verloren
- **Sinnvoll:**
  - **Wenn |b-p| sehr klein ist, dann zahlt sich der Aufwand für die Einbindung in die Liste des freien Speichers nicht aus**
  - **Verdichtung ist bei festen Block-größen b einfacher**





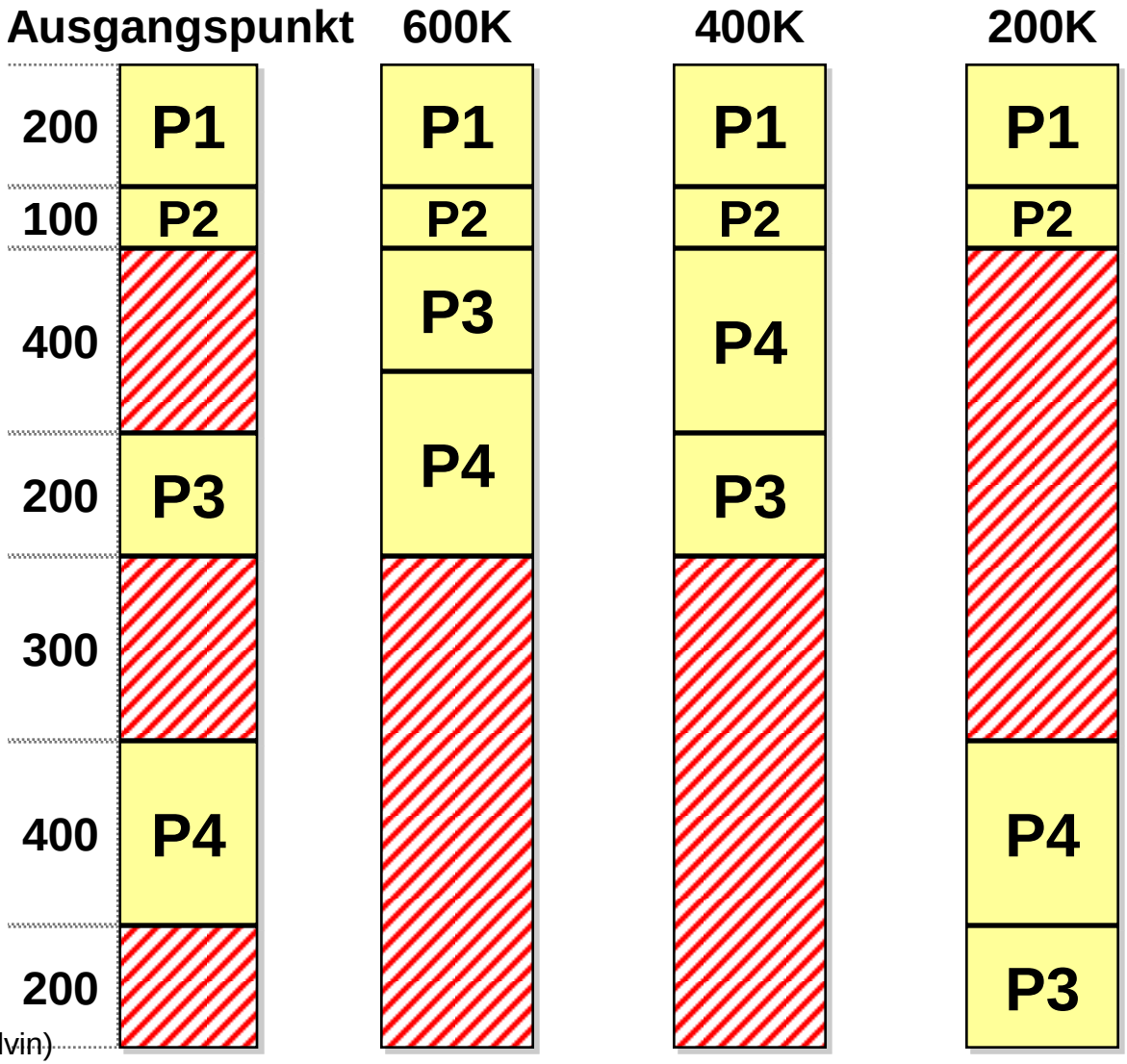
Maßnahmen gegen Externe Fragmentierung

- ***Verdichtung*:**

  Speicherblöcke so anordnen, dass freie Blöcke nebeneinander liegen und zu größeren kombiniert werden können
- **Aber: Benötigt *Adressbindung zur Laufzeit* (und Hardware-Unterstützung durch eine „memory management unit“ MMU)**
- **Auch wichtig:**
  Neuanordnung der Blöcke = Kopieraufwand
  Verlangt nach einer Strategie, um den Aufwand dazu (Zeit!) zu minimieren

Verdichtungsstrategie

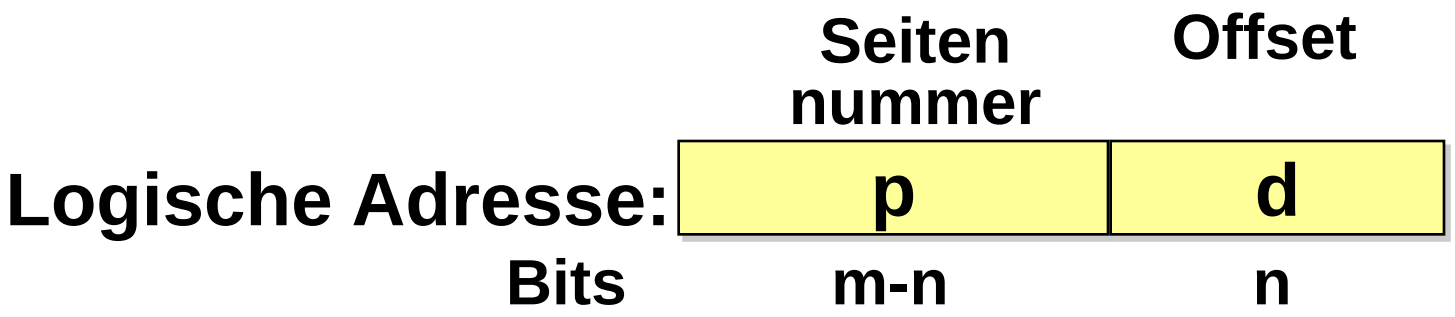
Verschiedene kompakte Segmente mit verschiedenem Aufwand (600K/ 400K / 200K) zu verschieben



(vergleiche Bild 8.11 in Silberschatz/Galvin)

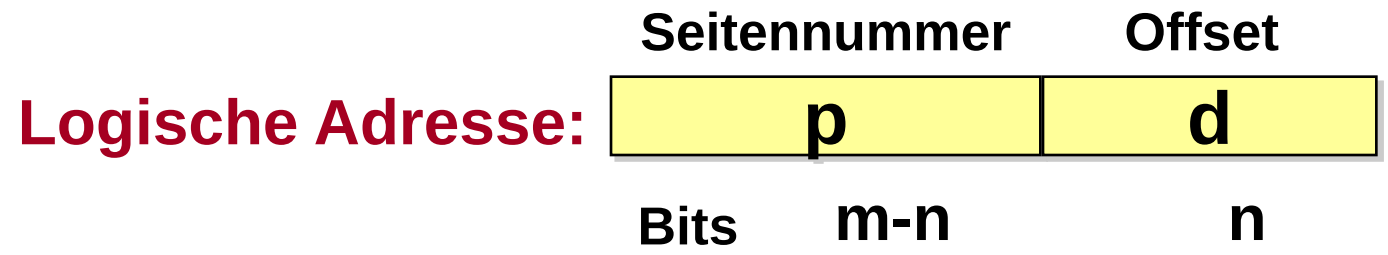
Seitenverwaltung
(engl.: paging)

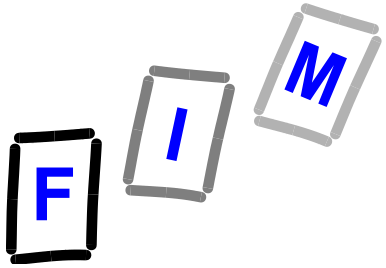
- **Teilung (getrennte Sicht)**
  - **Physischer Adressraum**
  - **Logischer Adressraum**
- **CPU erstellt logische Adresse**
  - **Gesamter Adressraum: 2^m**



Seitenverwaltung

- **Der physische Speicher wird in Blöcke mit fixer Größe (2^n) unterteilt:**
  Seitenrahmen (engl.: frame)
- **Der logische Speicher wird in Blöcke der gleichen Größe (2^n) zerlegt:**
  Seiten (engl.: page)
  - **page offset: ($0 \leq d < 2^n$) : Position innerhalb jeder Seite / Seitenrahmen in Bezug auf die Basisadresse der Seite**
  - **Jede Seite hat eine Nummer: Seitennummer p**



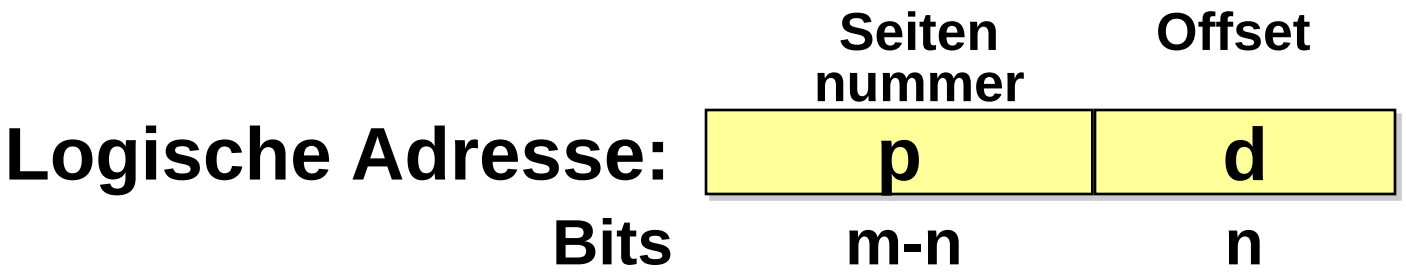


Zuordnung mittels Seitentabelle

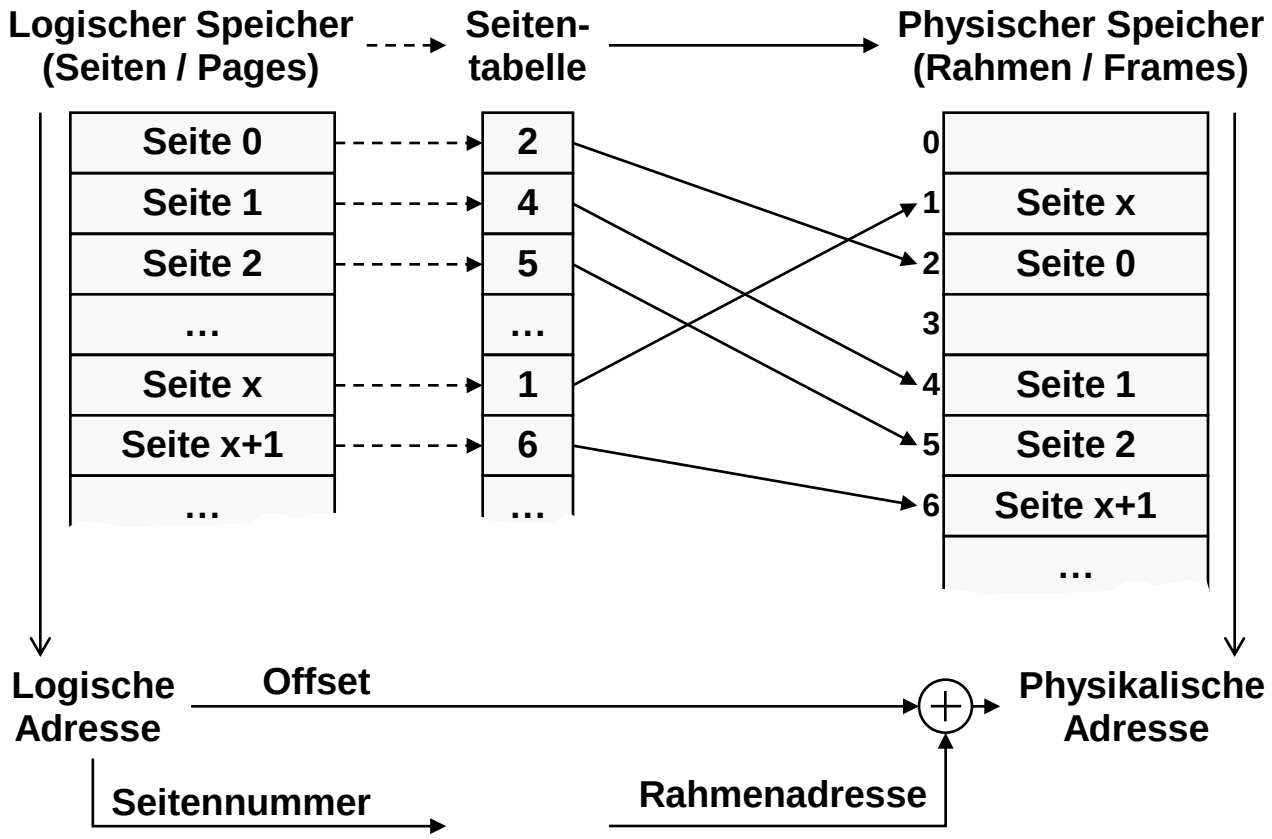
- **Zuordnung des logischen Adressraums zum physischen Speicher erfolgt durch die Seitentabelle (Page Table):**
  - **Logischer Adressraum: kontinuierliche Folge von Seiten**
  - **Physischer Adressraum: Seiten sind auf den ganzen Speicher verteilt**
- **Keine externe Fragmentierung: Seiten passen genau in den Seitenrahmen**
- **Interne Fragmentierung möglich: Seite ist nicht komplett mit Code oder Daten ausgefüllt**

Seitengröße

- **Die Größe der Seite / des Seitenrahmens wird von der Hardware vorgegeben**
  - **Potenz von 2, normalerweise 4 K = 2^{12}**
  - **dies vereinfacht die Übersetzung der logischen Adresse in die Seitennummer und das verbleibende Offset**
- **Wenn der logischer Adressraum 2^m und die Seitengrösse 2^n ist,**
  - **high-order m-n bits: Seitennummer**
  - **low-order n bits: Offset**



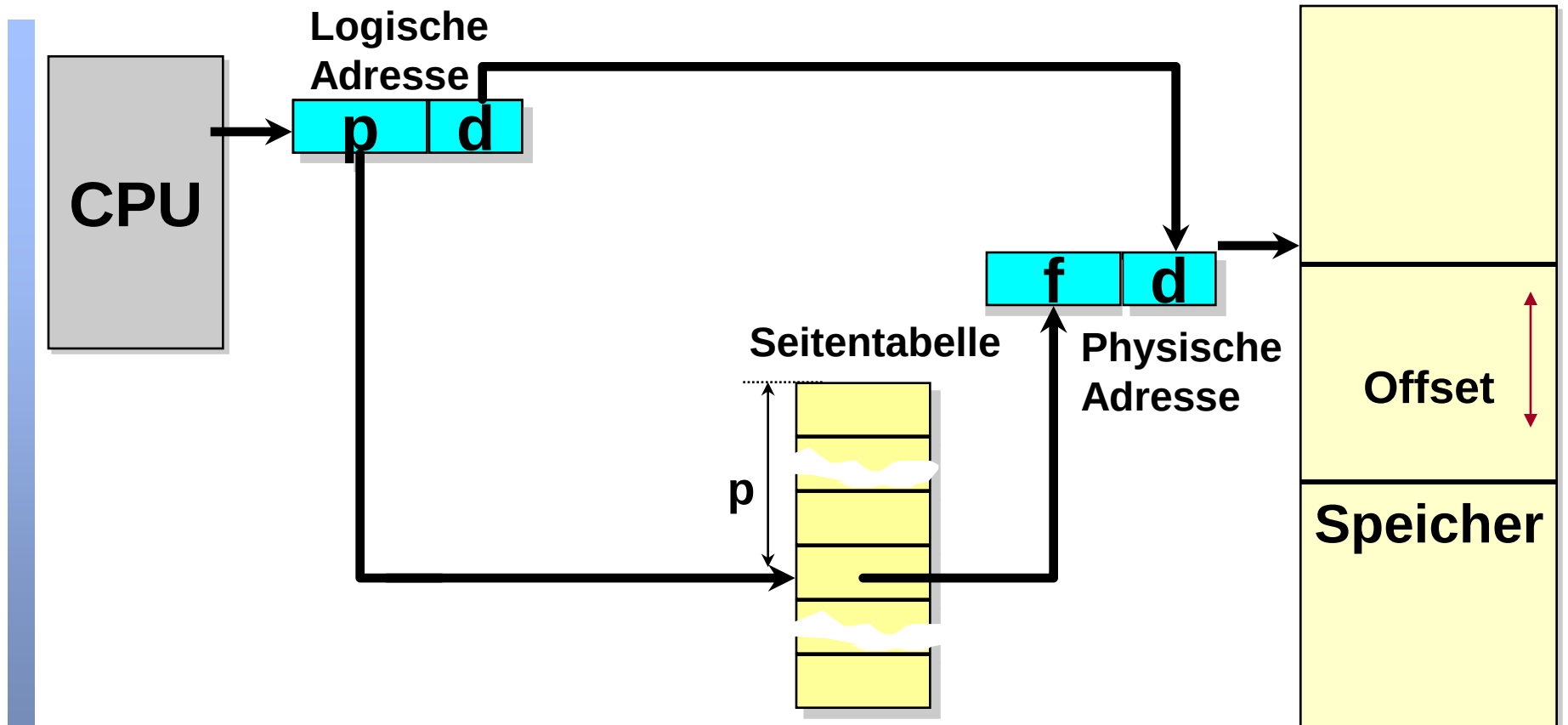
Zuordnung Pages -> Frames



Gesamtsicht

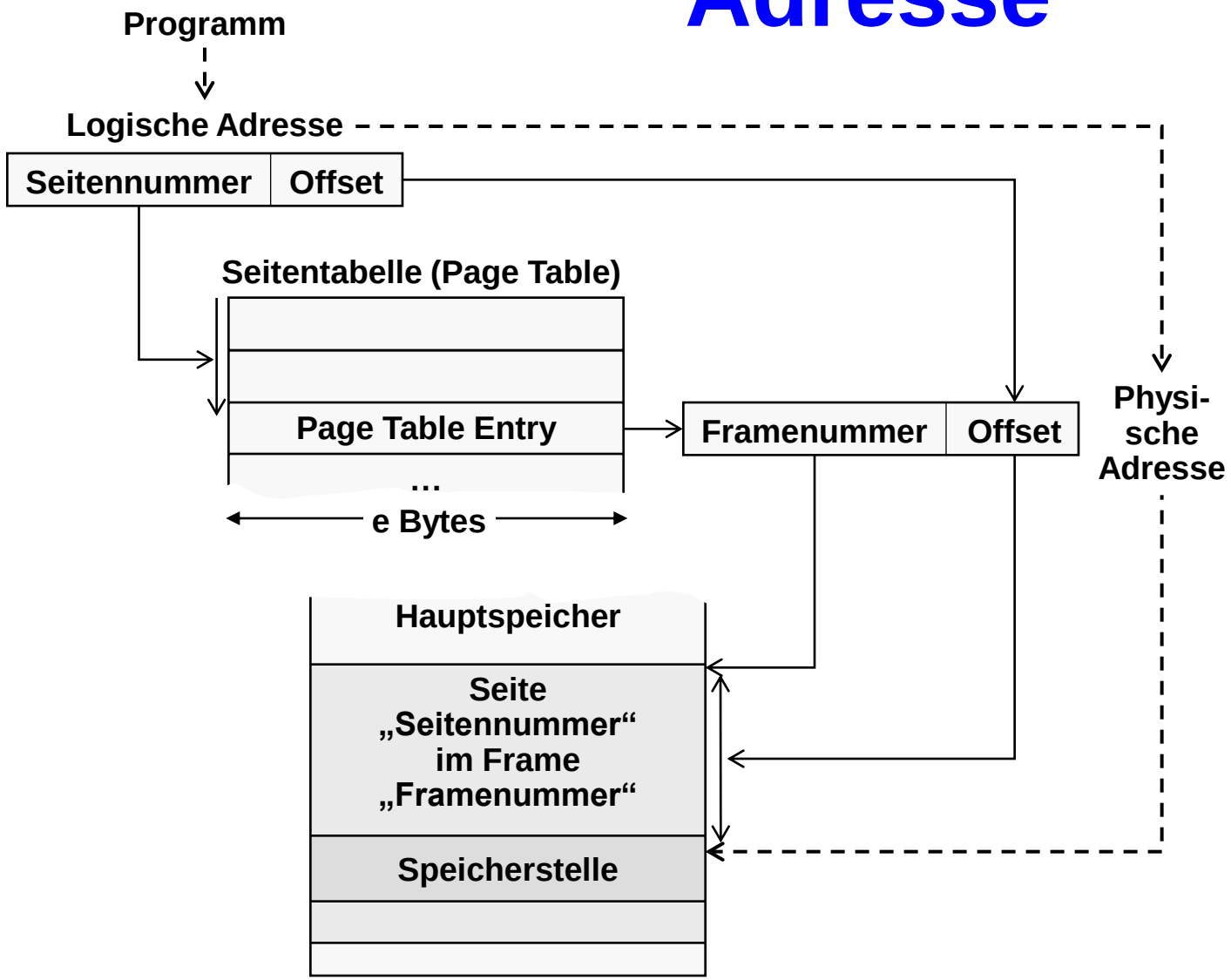
- **Das Programm / der Programmierer sieht einen kontinuierlichen logischen Adressraum.**
- **Dieser ist jedoch nicht unbedingt in einen kontinuierlichen physischen Adressraum abgebildet**
- **Der logische Adressraum ist immer „vollständig“, aber nicht überall gibt es auch zugeordneten physischen Adressraum**
- **Die Zuordnung des logischen Adressraums zu den physischen Adressen mittels Seitentabelle ist den Programmen gegenüber „verborgen“ (Fachausdruck: transparent)**
  - **Beachte: „transparent“ bedeutet im Deutschen ?**

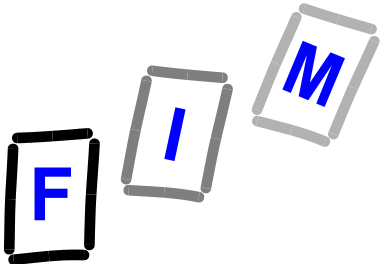
Seitenzuordnung (vereinfacht)



p Seiten, jede mit der Größe 2^n :
2^m gesamter logischer Adressraum

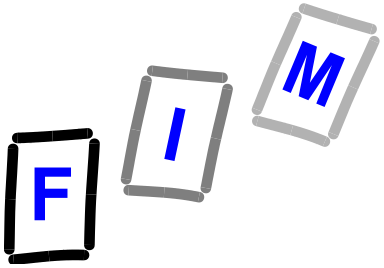
Logische → Physische Adresse





Besonderer Vorteil der Seitenverwaltung

- **Es gibt keine externe Fragmentierung, jeder freie Seitenrahmen kann einem Prozess, der ihn braucht, zugeordnet werden**
- **Interne Fragmentierung: Der letzte zugeordnete Seitenrahmen könnte nicht vollständig voll sein**
  - **Mehrere „Löcher“ → Stärkere interne Fragment.**
- **Da Seiten(-rahmen) die gleiche Größe haben geht das Aus- und Einlagern vom / in den Massenspeicher vom / in den Hauptspeicher schneller als bei Blöcken variabler Größe**



Seitenverwaltung → Eigener Adressraum für jeden Prozess

- **Wenn das BS Seitenverwaltung benutzt, dann erhält jeder Prozess seinen eigenen Adressraum durch die zugewiesene Tabelle**
  - **Für jeden Prozess gibt es eine Seitentabelle**
  - **Der Wechsel zwischen Programmen (Kontextswitch) …**
    - **inkludiert einen Wechsel der Seitentabellen**
    - **oder: Aktualisierung des Speicherbereichs, der die Seitentabelle enthält**

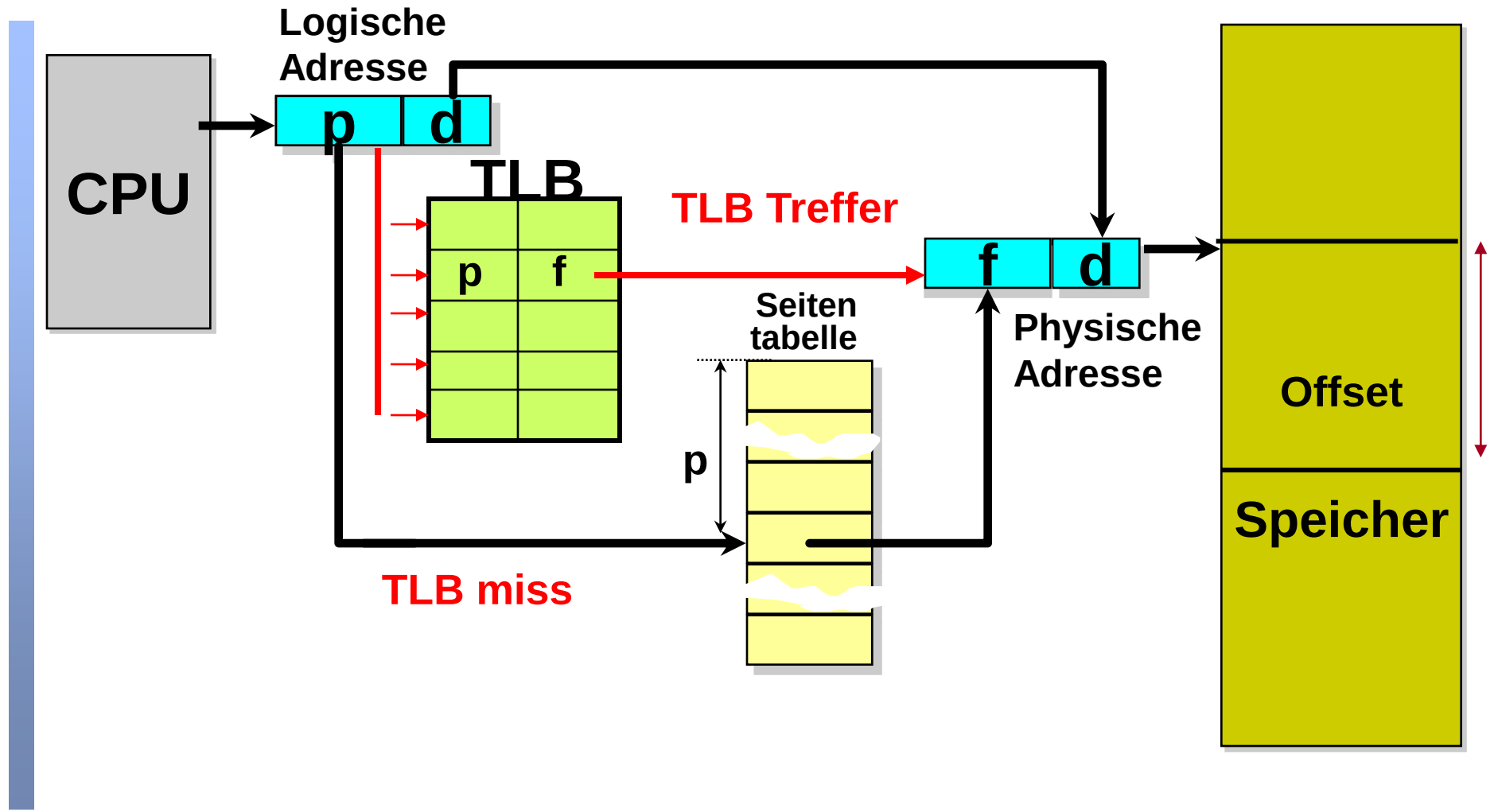
Seitenverwaltung Hardware-Unterstützung

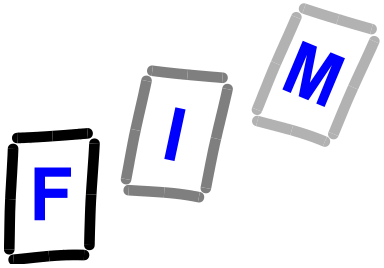
- **Jeder Speicherzugriff über die Seitentabelle!**
  → Effizienz?
- **Die Seitentabelle kann mittels besonderen zusätzlichen Registern implementiert :**
  - **Nur kleine Tabellen (<~256 Einträge) möglich**
  - ***Kontextwechsel:***
    Alle Registerinhalte müssen „gerettet“ werden
- **Seitentabelle vollkommen im Hauptspeicher:**
  - **Große Tabellen möglich (~1024^2 Einträge)**
  - **„Page-table base register (PTBR)“ (Seitentabellen Basisregister) notwendig, verweist auf die Basisadresse der Tabelle**
  - **Zugriffszeit: Zwei Speicherzugriffe notwendig**

"Translation Look-aside Buffer" (TLB)

- **Spezieller schneller Hardware-Cache für Seitentabellen**
- **Arbeitet wie ein sehr schneller Assoziativspeicher**
  - **Jeder Eintrag enthält einen Schlüssel und einen Wert**
  - **Alle Schlüssel werden gleichzeitig verglichen, wenn ein Element gesucht wird**
  - **Enthält normalerweise 64 bis zu 1024 Einträge**
    - **Warum nicht mehr? Assoziativspeicher ist sehr „teuer“ (=viele Gates)!**
- **Seitennummer wird dem TLB weitergegeben**
  - **Falls er sie findet: TLB Treffer, Seitenrahmennummer wird ausgegeben**
  - **Sonst: TLB Fehler (TLB-miss): Normaler Zugriff auf Seitentabelle notwendig**

Seitenverwaltungs-Hardware mit TLB

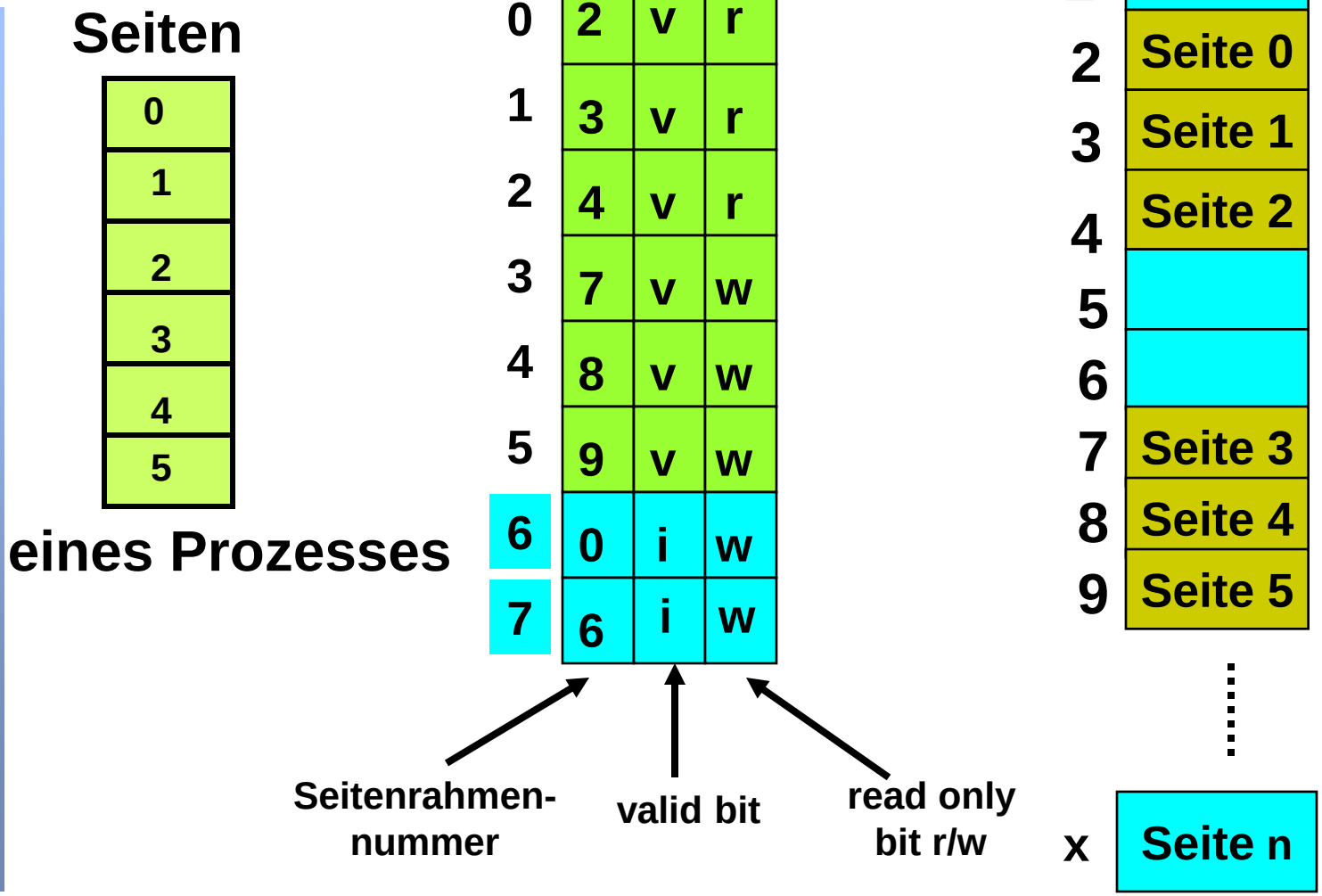


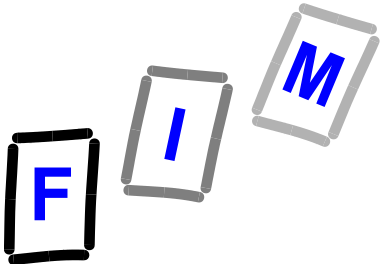


Seitenverwaltung und Schutzmaßnahmen

- Den Tabelleneinträgen werden Schutz-Bits hinzugefügt
- Read-only Bit
  - Nur Lesen / Lesen-Schreiben / nur Ausführen
- Valid Bit
  - Zeigt, ob die zugehörige Seite zum logischen Adressraum des Prozesses gehört
- No-Execute Bit
  - Ausführen von Programmcode verboten
- Damit überprüft man den Zugriff auf Zulässigkeit
  - Unzulässige Zugriffe werden abgefangen
    - BS löst einen Interrupt aus

Seitenverwaltung und Schutz-maßnahmen

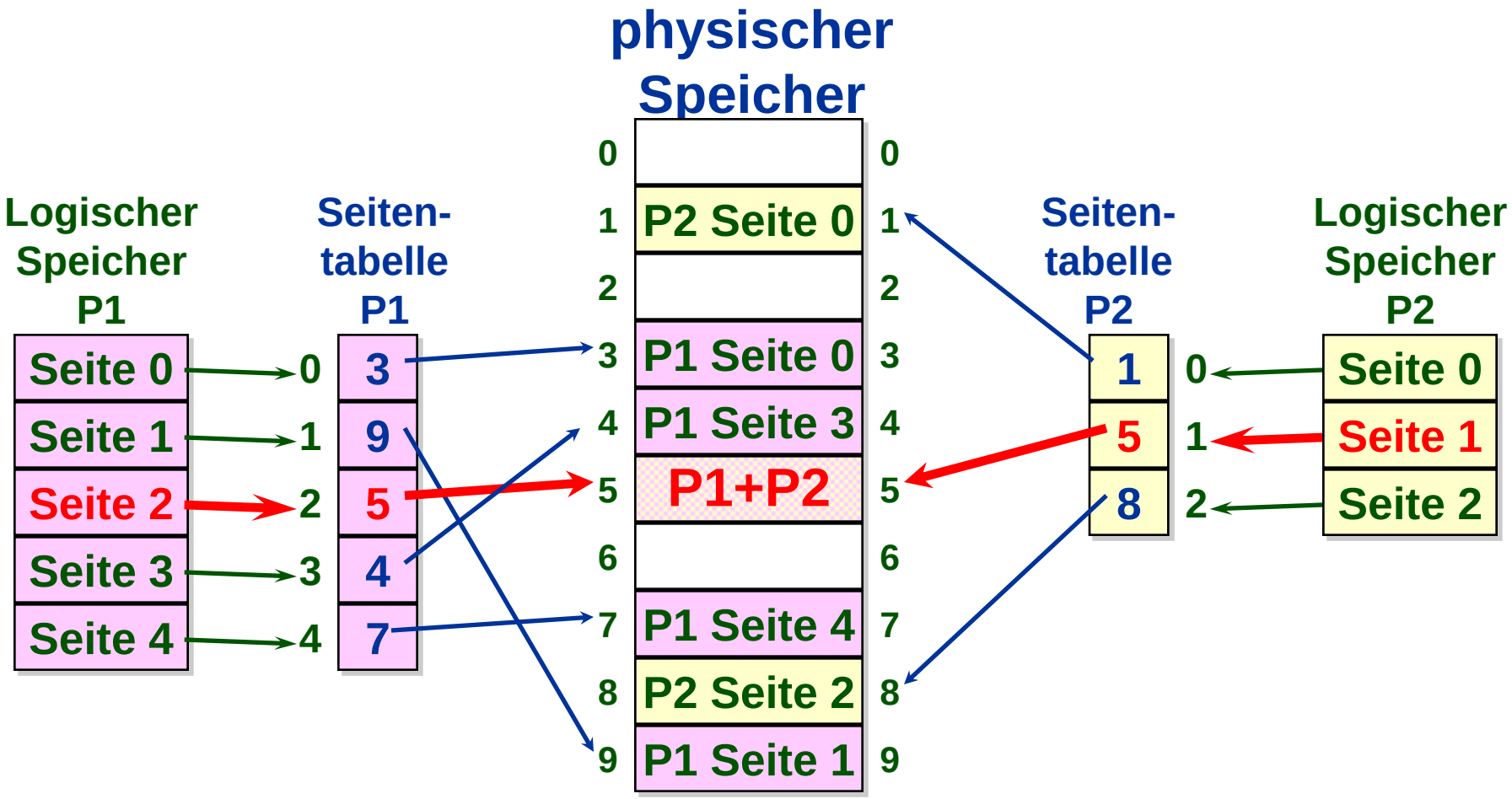




Gemeinsam benutzte Seiten

- **Eine Seite / ein Seitenrahmen kann zu verschiedenen Prozessen gehören =**
  „shared page“
- **Erlaubt Daten und Code zwischen Prozessen auszutauschen bzw. gemeinsam zu haben**
  - **Daten: Gemeinsame Datenpuffer**
    - **Problem: Wechselseitiger Ausschluss**
  - **Code: Mehrfach benutzte Programm-Module**
    - **Code muss ablaufinvariant (reentrant) sein**
    - **Seite muss „Nur Lesen“ (read only/execute only) sein**
    - **Z.B. für DLLs, die gemeinsam verwendet werden**

Beispiel: Shared pages



Beispiel: Code-Sharing

User1 und User2 verwenden gleichzeitig das selbe Editor-Programm, editieren aber verschiedene Daten