Memory Management

Peter Puschner
Institut für Technische Informatik
peter.puschner@tuwien.ac.at

Speicherverwaltung

- Effektive Aufteilung und Verwaltung des Arbeitsspeichers für BS und Programme
- Anforderungen
  - *Partitioning* (Speicheraufteilung auf Prozesse)
  - *Relocation* (flexible Positionierung von Code und Daten im Speicher), Virtual-Memory Management
  - *Protection* (Speicherschutz)
  - *Sharing* (gemeinsamer Zugriff auf Speicher)
  - *Performance* (effektive log./phys. Organisation)

Partitionierung des Speichers

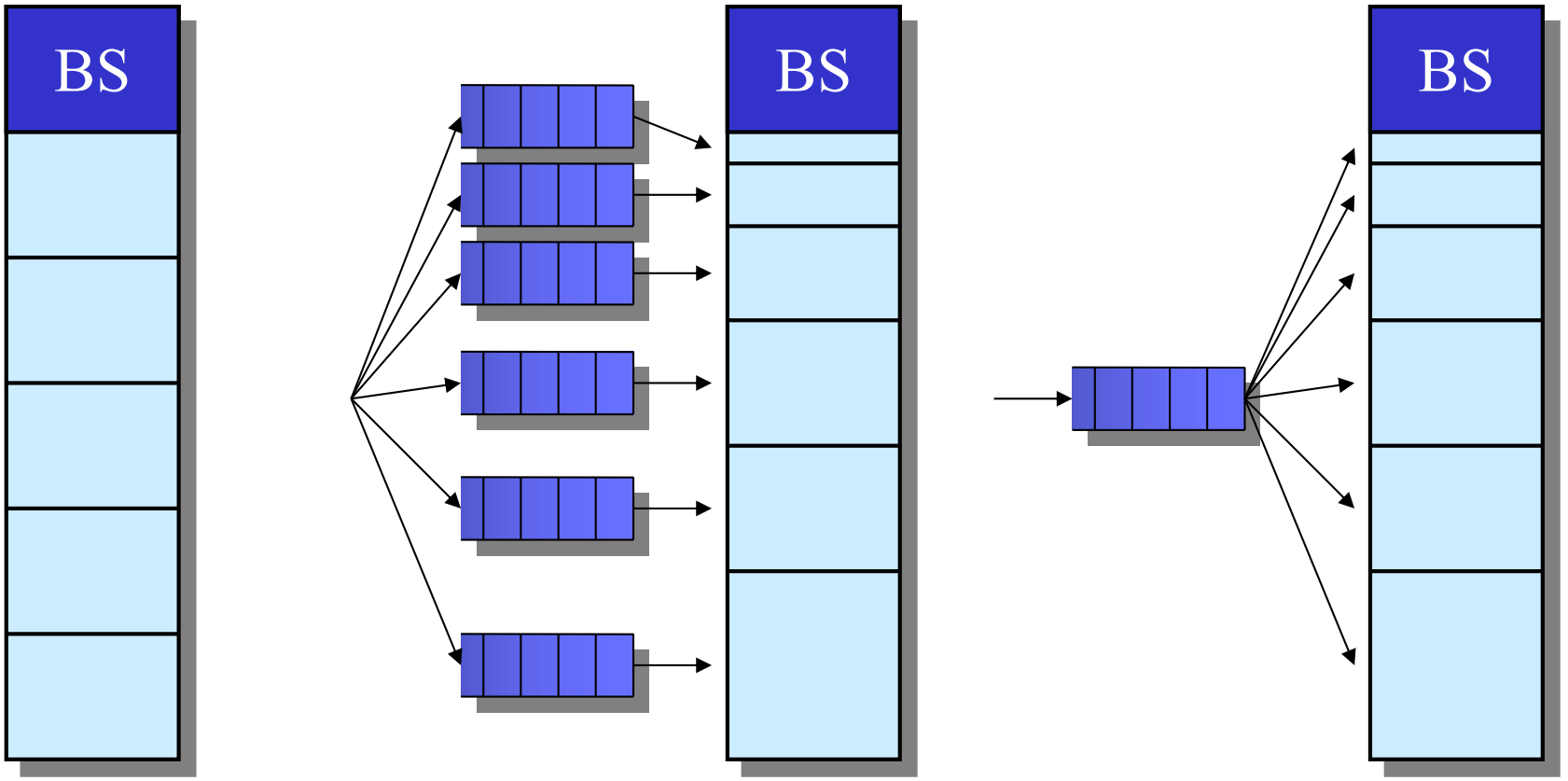
Einfache Ansätze zur Speicherverwaltung:

- Prozesse werden als Ganzes im Hauptspeicher abgelegt
- Grundlage für moderne Speicherverwaltung
  - *Fixed partitioning*
  - *Dynamic partitioning*
  - *Buddy System*

Fixed Partitioning

gleiche Partitionsgrößen

unterschiedliche Partitionsgrößen



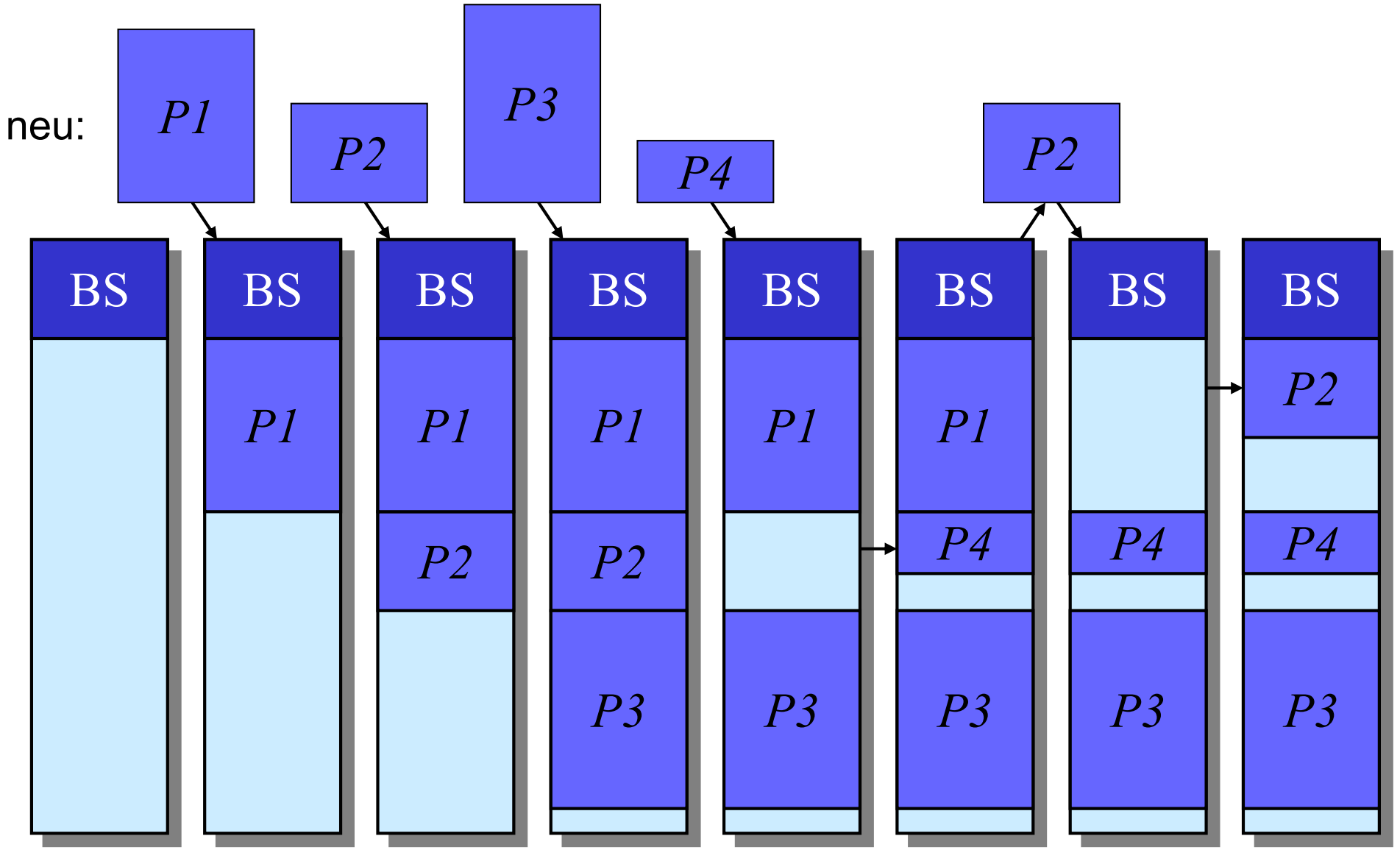
eine Process-Queue pro Partition

eine Process-Queue

Fixed Partitioning

- Unterteilung des Hauptspeichers in nicht überlappende Teile
- Prozesse, die kleiner gleich einer Partition sind, können geladen werden
  - gleiche bzw. ungleiche Partitionsgrößen
- Auslagern eines Prozesses durch das BS, wenn alle Partitionen belegt sind
- Programme, die größer als Partitionen sind
  - ⇨ Programmierung von *Overlays*

Dynamic Partitioning



Dynamic Partitioning

- Partitionen variabler Länge und Anzahl
- Zwischen Partitionen entstehen Löcher
  ⇨ *Compaction*: Verschieben der Partitionen, um statt Löchern größeren zusammenhängenden freien Speicherbereich zu erhalten
- *Placement Strategies* beim Einfügen
  - *Best fit*
  - *First fit*
  - *Next fit*

Fragmentierung des Speichers

- *Interne Fragmentierung*: Verschwendung von Speicherplatz innerhalb einer Partition; Daten und Programm füllen Partition nicht aus (z.B. bei fixer Partitionierung)
- *Externe Fragmentierung*: Zerstückelung des Speicherbereichs außerhalb der Partitionen (z.B. bei dynamischer Partitionierung)

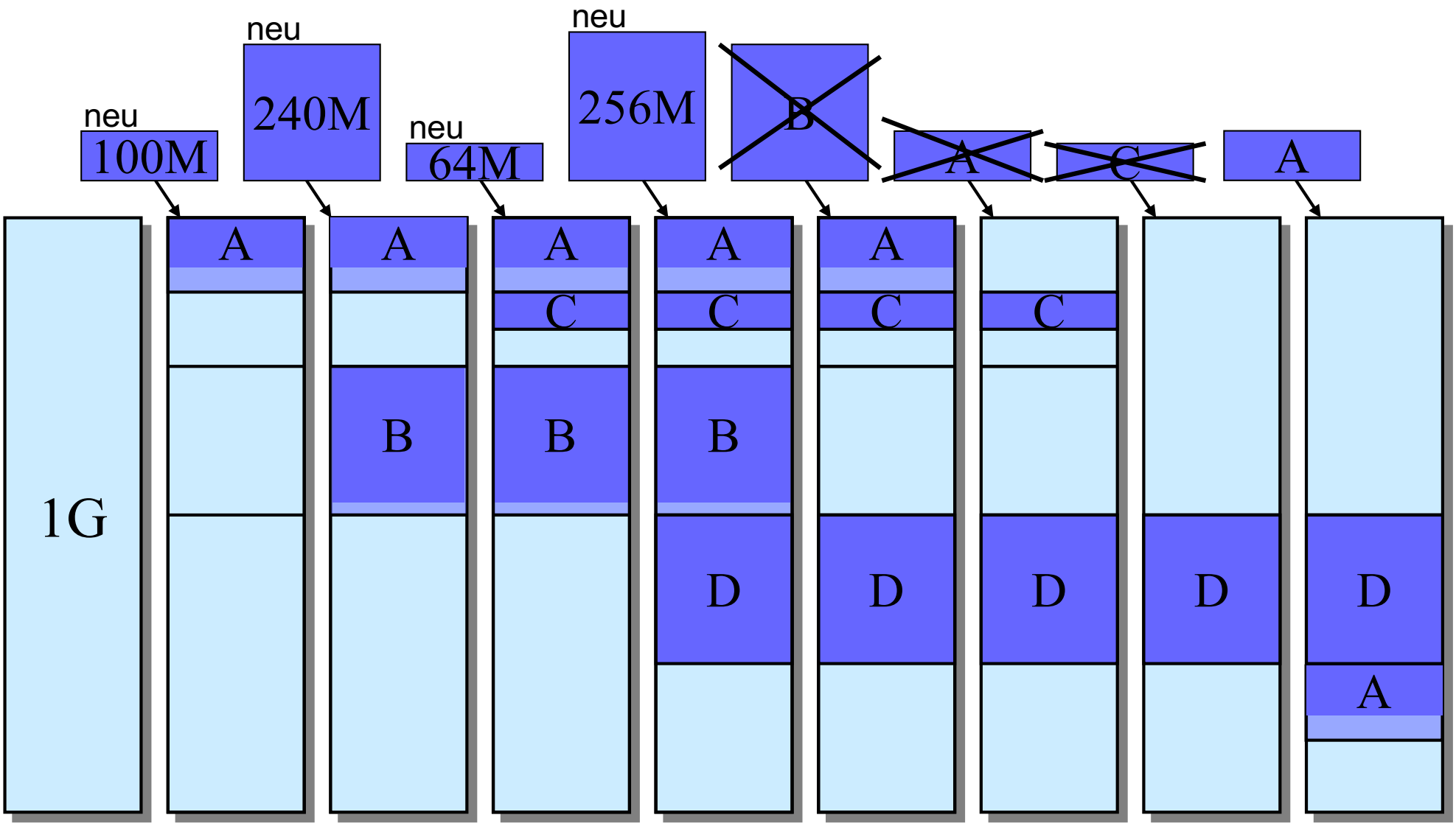
Buddy System

- Versuch, die Nachteile der vorher angeführten Strategien zu verringern
- Zerlegung des Speichers in “Buddies” der Größe 2^k, mit *MinVal* <= *k* <= *MaxVal*

Buddy System

- Start: ein Block max. Größe
- Anforderung eines Blocks der Größe *S*
  - Allokiere einen Block der Größe 2^U, wobei $2^{U-1} < S <= 2^U$
  - gibt es keinen solchen Block, zerteile den nächstgrößeren Block in zwei kleinere
  - wiederhole ggf. den vorigen Schritt
- Bei Freiwerden eines Blocks kombiniere ursprünglich zusammengehörige Blöcke

Buddy System



Relocation

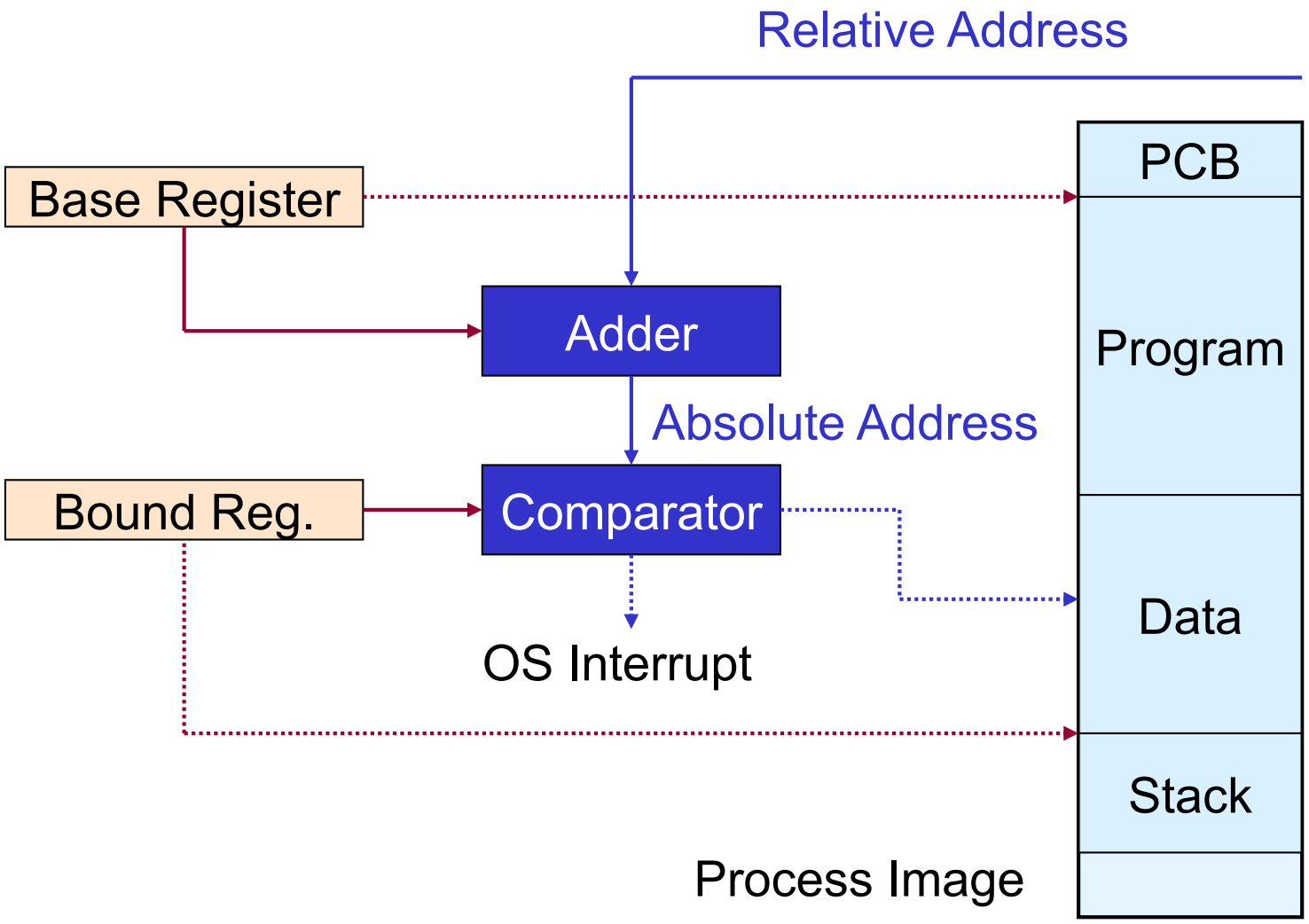
- Der von Prozessen belegte Speicherbereich ist nicht statisch fixiert (Aufruf bei unterschiedlicher Speicherbelegung, Swapping, Compaction)
- Instruktionen, Daten, etc. müssen an verschiedene Speicherstellen positioniert werden können
- Referenzen auf physischen Speicher müssen veränderbar sein

⇨ Unterscheidung zwischen logischer Adresse und physischer (absoluter) Adresse

Speicheradressierung

- *Physische (absolute) Adresse*:
  absolute Position im Hauptspeicher
- *Logische Adresse*: Referenz einer Position im Speicher, unabhängig von Organisation des Speichers
- *Relative Adresse*: Position relativ zu einem bekannten Punkt im Programm (= logische Adr.)
- Übersetzter Code enthält logische, meist relative Adressen ⇨ zur Laufzeit: Adressübersetzung auf absolute Adressen

Einfache Adressübersetzung



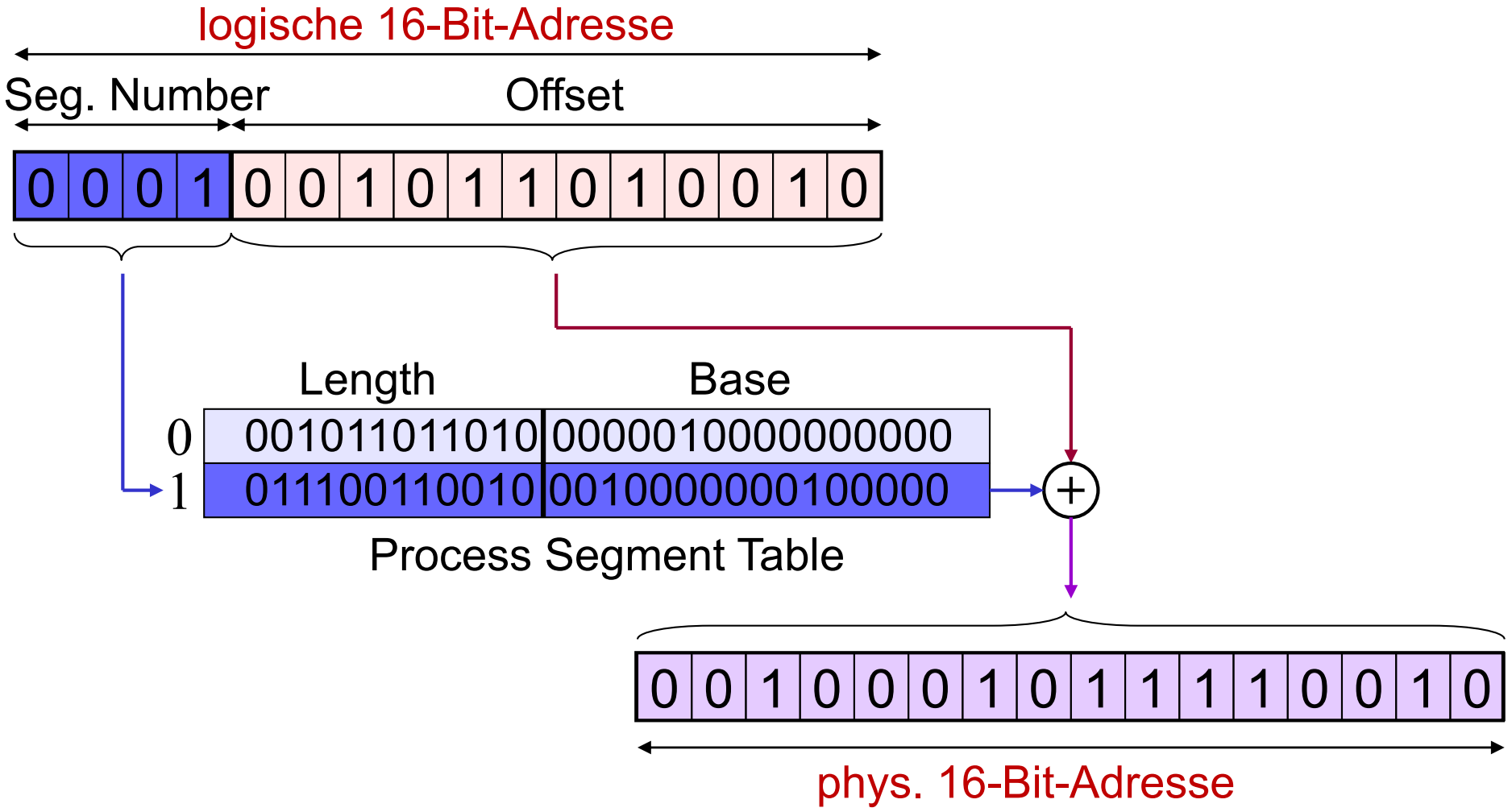
Einfache Adressübersetzung

- Adressübersetzung durch Hardware
- Physische Startadresse des laufenden Prozesses im *Base Register*
- Endadresse des physischen Speichers des Prozesses im *Bound Register*
- Zugriff auf relative Adresse: Inhalt des Basisregisters wird zu relativer Adresse addiert
- Speicherschutz (*Protection*): Überprüfung, ob die Adresse im gültigen Bereich (= innerhalb der Bound) liegt

Segmentierung

- Unterteilung von Programmen in Blöcke unterschiedlicher Länge (=Segmente)
- Segmente enthalten Code und/oder Daten
- Beim Laden des Prozesses werden seine Segmente “beliebig” im Speicher platziert
- keine interne, aber externe Fragmentierung
- sichtbar für den Programmierer
- Segmenttabelle pro Prozess, für jedes Segment existiert ein Eintrag: (*Start*, *Länge*)

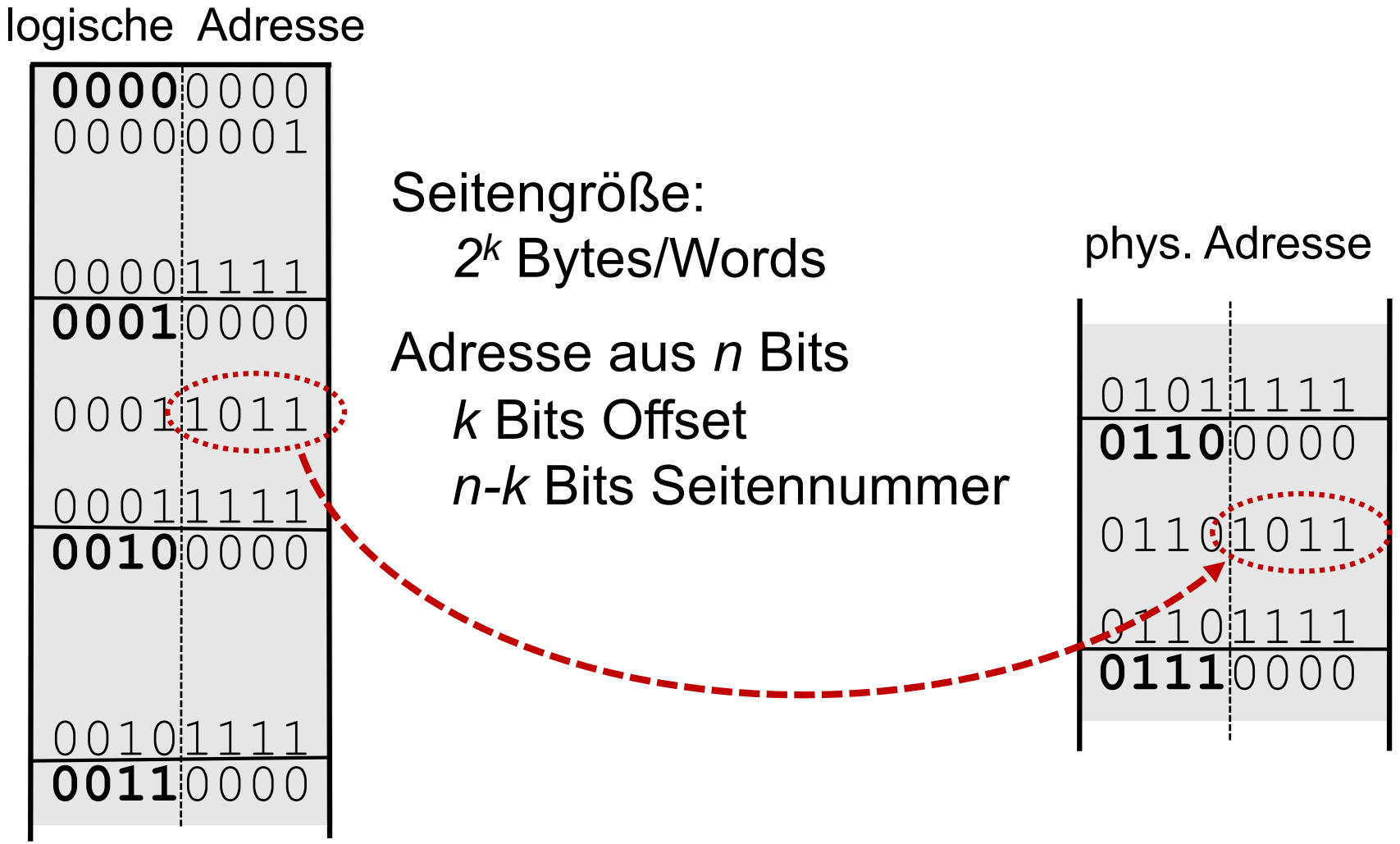
Segmentierung - Adr.übersetzung



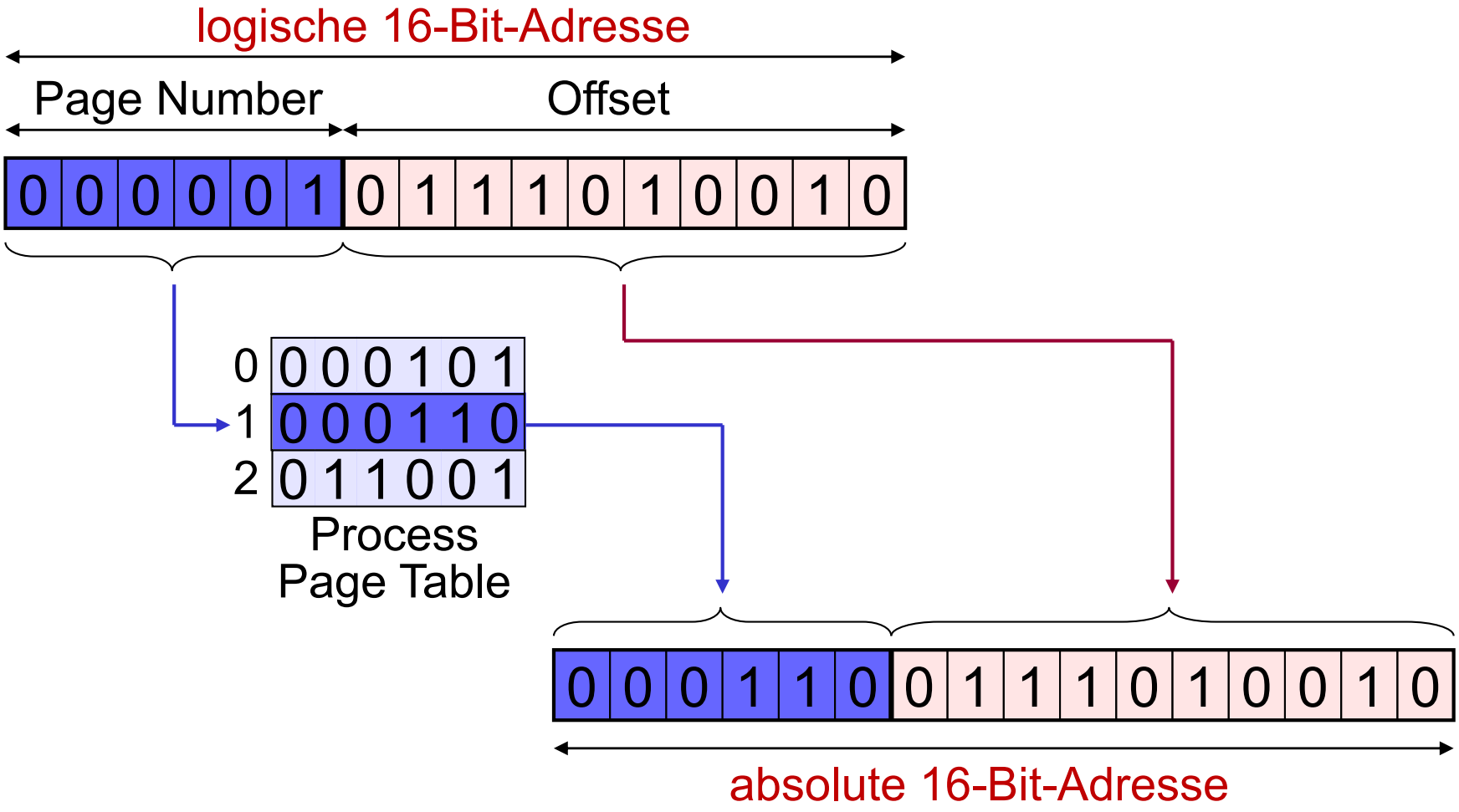
Paging

- Unterteilung des Hauptspeichers in (kleine) *Seitenrahmen* (*Frames*) gleicher Größe
- Prozesse werden in *Seiten* (*Pages*) der selben Größe unterteilt
- Seiten werden in Seitenrahmen geladen
  - Frames eines Prozesses nicht unbedingt dicht
- *Page Table* pro Prozess: enthält für jede Page die aktuelle Frame-Nummer
- *Free Frame List* verweist auf freie Frames im Speicher

Paging – Seitengröße



Paging - Adressübersetzung



Paging - Logische Adressen

- Logische Adresse: (*Page Number*, *Offset*)
- Mit Hilfe der *Page Table* ermittelt der Prozessor die absolute Adresse:
  (*Frame Number*, *Offset*)
- Seitengröße: 2^k Bytes bzw. Worte
  - Seitenunterteilung für Tools (Compiler, Linker) bzw. Programmierer “unsichtbar”
  - Adressen einfach zu übersetzen

Virtual Memory

- Dynamische Adressübersetzung
- Aufspaltung von Prozessen in Seiten (oder Segmente), die nicht hintereinander im Speicher abgelegt werden müssen

⇨ Log. Adressraum eines Prozesses muss sich nicht als ganzes im Speicher befinden

⇨ nur “aktuell benötigte” Seiten im Speicher

⇨ mehr Prozesse im Hauptspeicher

⇨ einzelne Prozesse können größer als der Hauptspeicher sein

Virtual Memory

- Logische Adressen referenzieren Virtual Memory
- Seitentabelle (Segmenttabelle) und Memory-Management HW unterstützen die Übersetzung von virtuellen Adressen in absolute Adressen
- Teile eines Prozesses, die sich im Hauptspeicher befinden: *Resident Set*

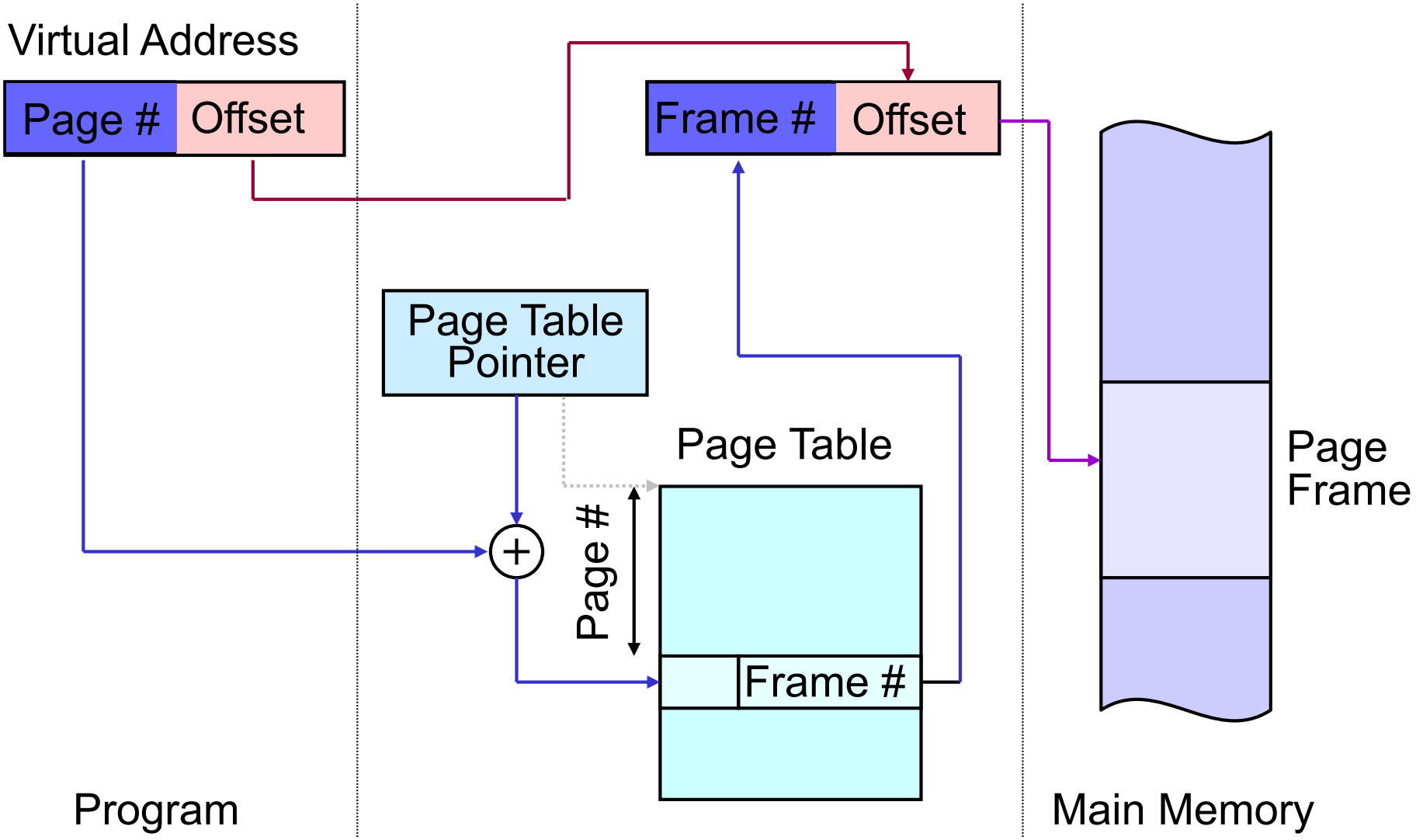
Virtual Memory (VM)

- *Page Fault*: Ausnahmebehandlung, wenn eine Adresse im VM referenziert wird, die sich nicht im Hauptspeicher befindet
  - entsprechender Speicherbereich wird von Sekundärspeicher geladen (*Demand Paging*)
- *Thrashing*: Durch häufige Page Faults verbringt der Prozessor sehr viel Zeit mit dem Laden vom Sekundärspeicher
  ⇨ drastischer Einbruch der Effektivität
  (z.B., Resident Set eines Prozesses zu klein)

Paging

- Pro Prozess eine Seitentabelle
  - Tabellengröße hängt von Seiten- u. Prozessgröße ab
- Tabelleneinträge in Page Table
  - *Present Bit* – Seite im Hauptspeicher?
  - *Frame Number* bzw. wo Seite zu finden ist
  - *Modified Bit* – Seite seit Laden verändert?
  - *Control Bits*: r/w-Bits, Locking, Kernel vs. User Page

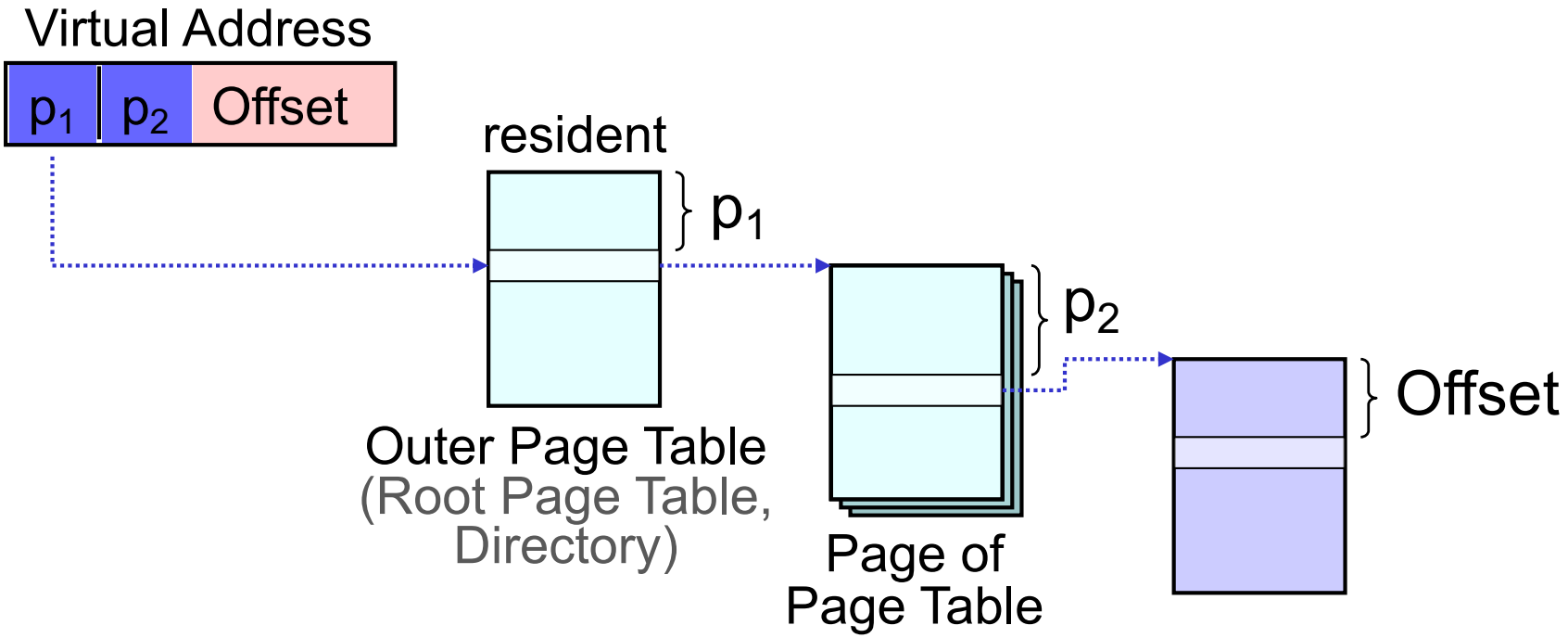
Paging - Adressübersetzung



Multilevel Page Tables

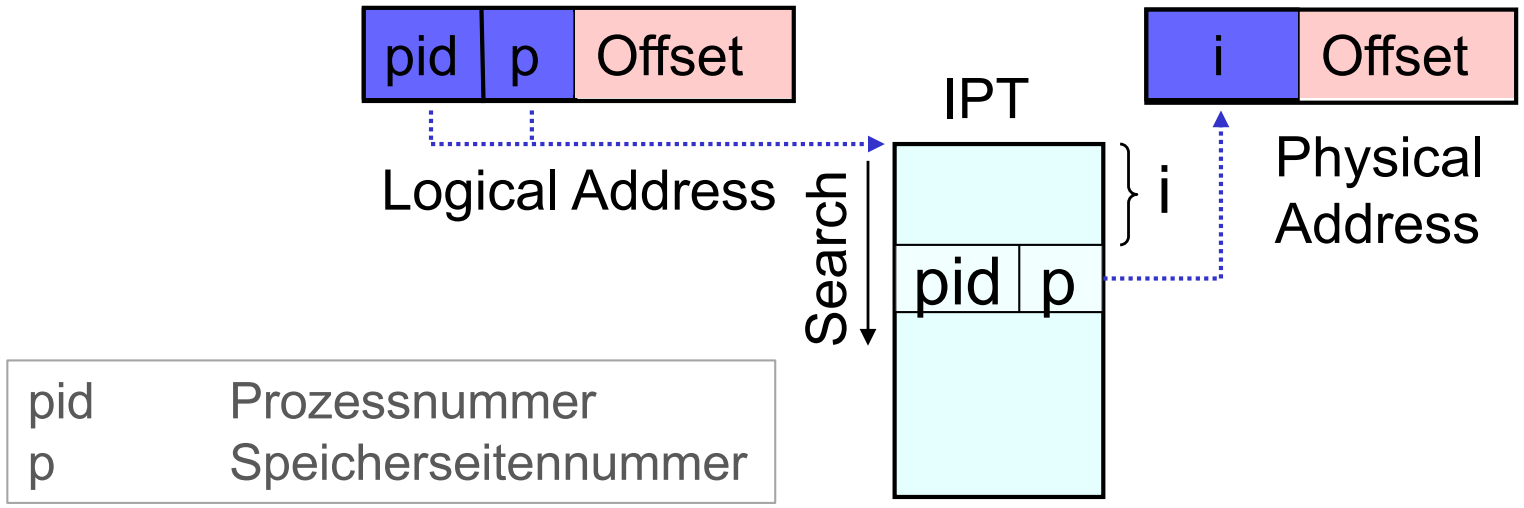
- Bei großen Prozessen benötigt die Seiten-tabelle selbst mehrere Seiten

⇨ *Multilevel Page Table*



Inverted Page Table (IPT)

- Eine IPT für das gesamte System
- Ein IPT-Eintrag pro Frame
  - Bei Zugriff auf Speicher: assoziative Suche in IPT bzw. Hashing um Suche in IPT zu beschleunigen
  - Page Fault, wenn gesuchter Eintrag nicht in IPT



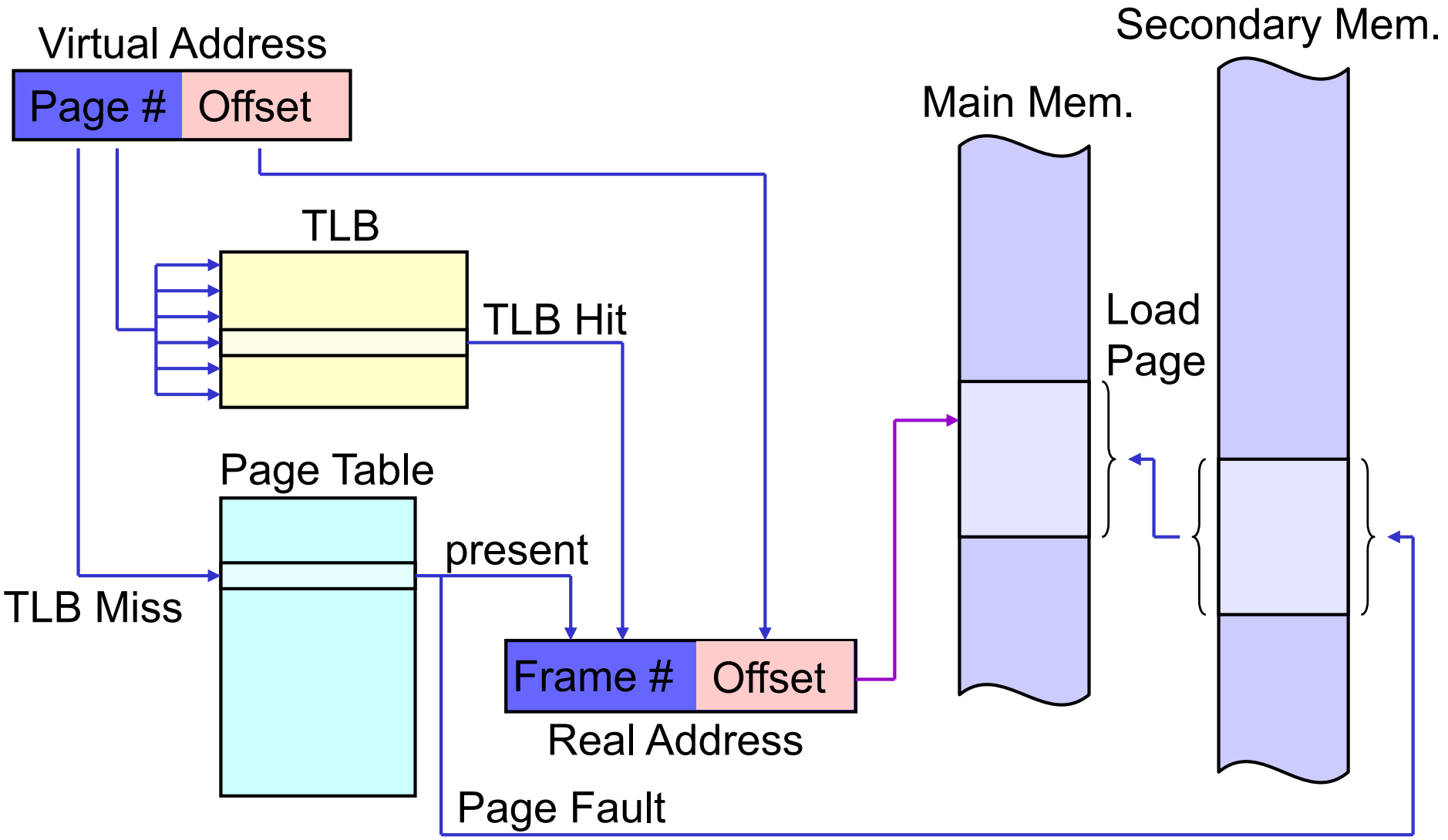
Translation Lookaside Buffer

- Page Table im Hauptspeicher ⇨ pro Zugriff auf das VM sind (mindestens) zwei Zugriffe auf den physischen Speicher notwendig

⇨ *Translation Lookaside Buffer* (*TLB*) = Cache für Einträge der Seitentabelle

  - Einträge enthalten Nummern (Page, Frame) der "zuletzt" verwendeten Seiten
  - 16-512 Einträge
  - Assoziativer Zugriff beim Suchen
  - Löschen des TLB bei jedem Context Switch

Translation Lookaside Buffer (TLB)



Fetch Policy

bestimmt, wann eine Seite geladen wird

- *Demand Paging*
  - lädt eine Seite, wenn eine Adresse der Seite referenziert wird
  - viele Page Faults beim Start eines Prozesses
- *Prepaging*
  - lädt mehr Seiten als angefragt im Voraus
  - durch Lokalitätsprinzip motiviert (Prog., Disk)
  - Laden nicht benötigter Seiten …

Ähnlich: Pre-Cleaning vs. Demand Cleaning

Replacement Policy

bestimmt, welche Seite beim Laden einer neuen Seite im Hauptspeicher ersetzt wird

- globale vs. lokale Ersetzungsstrategie
  - lokal: Austausch innerhalb der Seiten des Prozesses
  - global: Austauschstrategie wird auf alle Seiten des Hauptspeichers angewandt

OPT Policy

- OPT: optimale Strategie
  - minimale Anzahl von Page Faults
  - ersetzt die Seite, deren nächste Referenz am weitesten in der Zukunft liegt
  - keine reale Strategie
  - wird zur Bewertung anderer Strategien herangezogen

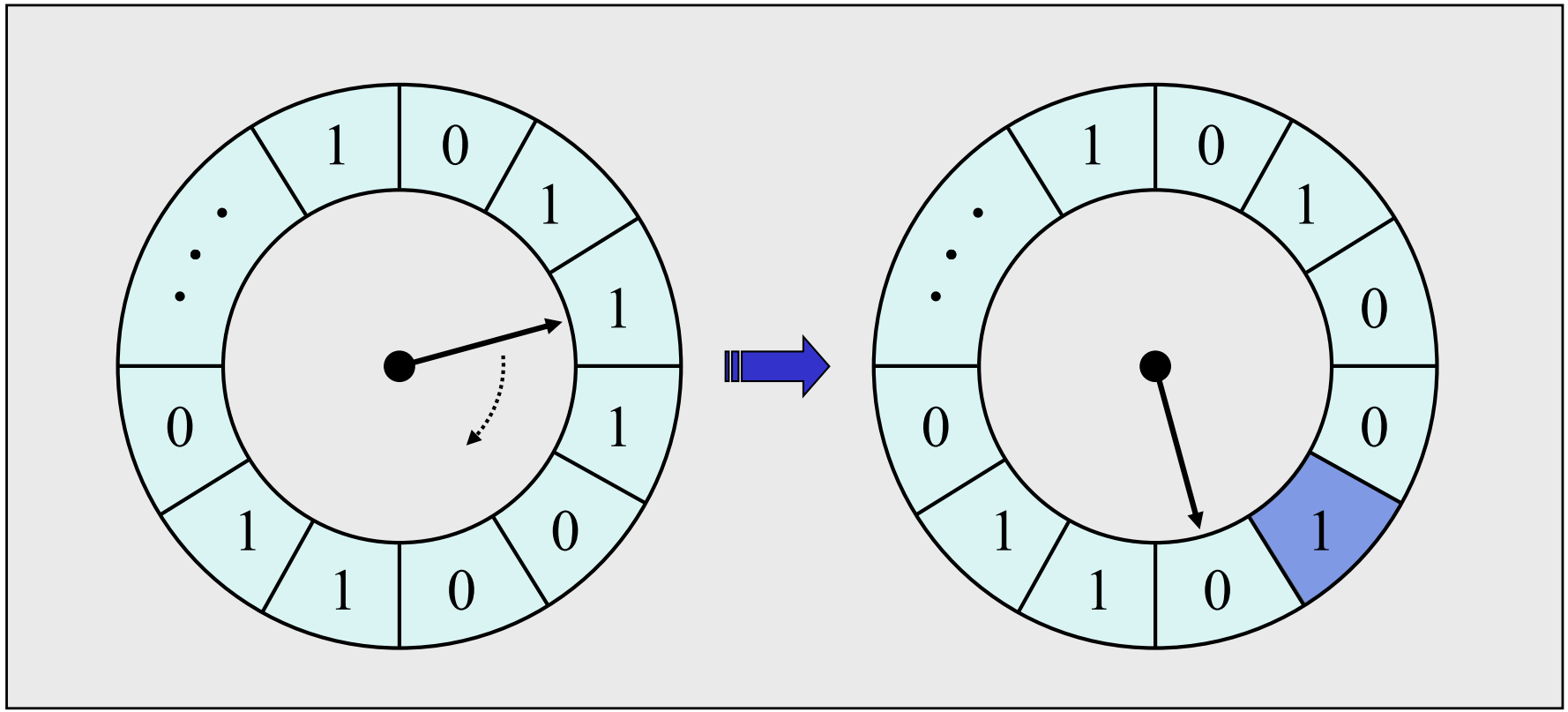
LRU Policy

- LRU (least recently used)
  - ersetzt die Seite, die am längsten nicht benützt worden ist
  - aufgrund des Lokalitätsprinzips ist zu erwarten, dass die Seite auch in naher Zukunft nicht referenziert werden wird
  - Anzahl der Page Faults kaum höher als bei OPT
  - Implementierung aufwändig:
    Speichern der Zugriffszeiten für Seiten,
    Suche nach der am längsten nicht verwendeten Seite

FIFO Policy

- FIFO (first in - first out)
  - bei vollem Seitenkontingent wird die älteste Seite ersetzt
  - Nachteil: eine Seite, die gerade oft verwendet wird, kann die älteste sein; sie wird ausgelagert
  - einfache Implementierung: Pointer, der zyklisch über die Seitenrahmen fortgeschaltet wird

Clock Policy



kaum mehr Page Faults als LRU

mehr Frames pro Prozess ⇨ bessere Annäherung

Clock Policy

- Ringpuffer mit Frames, die für Austausch in Frage kommen + Positionszeiger
- nach Page Fault: Zeiger wird auf den folgenden Frame im Ringpuffer gesetzt
- *Use Bit* pro Frame, wird auf 1 gesetzt, wenn Seite geladen oder referenziert wird
- bei Page Fault:
  - Suche erste Seite ab Zeigerposition mit *Use Bit* = 0
  - Setze dabei *Use Bits* mit Wert 1 auf 0

Größe des Resident Set

Resident Set: Seiten eines Prozesses, die sich im Hauptspeicher befinden

- Größe des Resident Set: Wieviele Seiten-rahmen sollen dem Prozess zugeteilt werden?
  - wenige Frames ⇨ viele Page Faults
  - viele Frames ⇨ Einschränkung der Parallelität
  - *Fixed Allocation*
  - *Variable Allocation*

Working Set Strategie

- Allokation einer variablen Anzahl von Frames pro Prozess basierend auf Lokalitätsannahme
- *Working Set* eines Prozesses zum Zeitpunkt *t*: *W*(*D*, *t*) = Menge der Seiten des Prozesses, die in den letzten *D* virtuellen Zeiteinheiten referenziert wurden
  - virtuelle Zeit: bezieht sich auf betrachteten Prozess
  - *D* … Zeitfenster (Window of Time)
  - Funktion *W*(*D*, *t*) beschreibt Lokalität des Prozesses

Working Set Strategie

- Working Set eines Prozesses wächst beim Start schnell an,
- stabilisiert sich während der weiteren Prozessausführung (Lokalitätsprinzip),
- wächst, wenn sich die Abarbeitung in einen anderen Adressbereich verschiebt, …

Working Set Strategie

Ermittlung der Größe des Resident Set:

- Beobachtung des Working Set der Prozesse
- Periodisches Löschen der Seiten, die sich nicht im Working Set befinden
- Wenn Resident Set $\subset$ Working Set
  - allokiere fehlende Frames
  - sind nicht genügend Frames verfügbar, suspendiere den Prozess und probiere Allokation später nochmals

Working Set Strategie

- Probleme bei der Realisierung:
  - Mitloggen der Seitenreferenzen
  - Ordnen der Seitenreferenzen
  - optimales *D* ist zur Laufzeit unbekannt und variiert

⇨ Gängige Praxis: Beobachtung der Anzahl der Page Faults pro Zeitintervall und Prozess statt Working Set

VM, Paging – Protection

- Verfügbarkeit von linearen, kontinuierlichen virtuellen Adressbereichen für jeden Prozess
- Adressbereiche werden entweder mittels Seitentabelle auf physischen Speicher abgebildet oder sind “protected”
- bei jedem Zugriff auf eine (beliebige) Adresse
  - entweder es gibt einen Verweis auf eine phys. Adresse in der Page Table
  - oder es kommt zum Page Fault: wenn es sich nicht um eine ausgelagerte Seite handelt, liegt eine Speicherbereichsverletzung vor

Protection Keys

Variante 1:

- Pro phys. Frame gibt es einen *Protection Key*
- Jeder Prozess hat einen Protection Key, der beim Speicherzugriff mit dem Key des Frames verglichen wird: Ungleichheit triggert einen Fehler

Variante 2:

- Pro TLB Eintrag gibt es einen *Key* (bzw. *Access ID*)
- Jeder Prozess hat eine Menge von *Protection Key Registers*, die bei Speicherzugriff mit *TLB Key* verglichen werden
- Fehler, wenn kein *Key* gleich dem *TLB Key* (Bsp.: *Itanium* Prozessor)

Zusammenfassung

- Partitionierung des Hauptspeichers
  - Aufteilung auf mehrere Prozesse
- Relocation ⇨ logische Adressierung
- Virtual Memory
  - logischer Speicher > phys. Speicher
  - Paging (Segmentierung)
  - Adressübersetzungstabellen und -hardware
  - Lade- und Austauschstrategien
  - Wahl der Größe des Resident Set
  - Protection