

Systeme I: Betriebssysteme

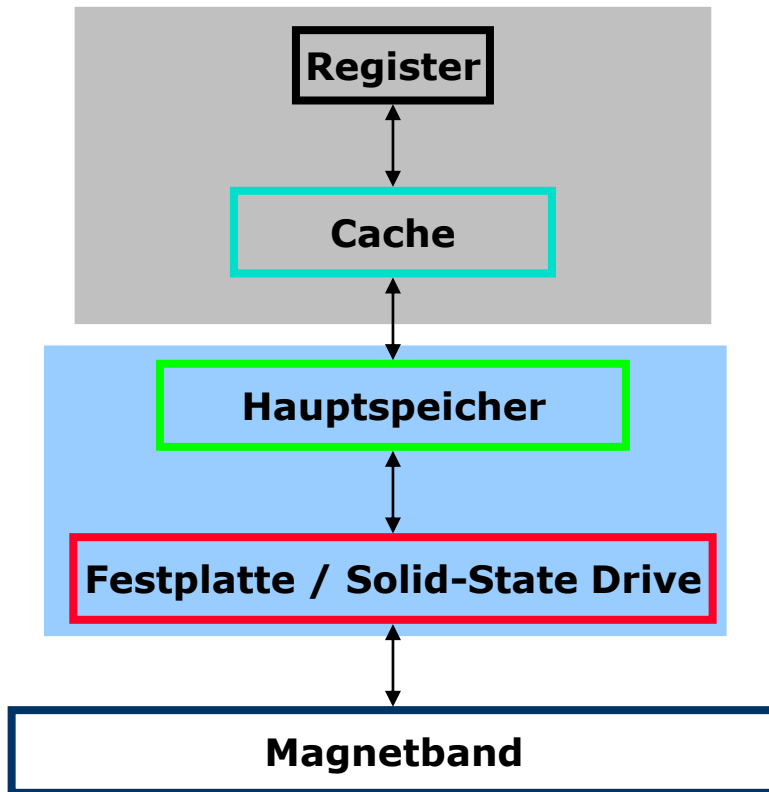
Kapitel 8 **Speicherverwaltung**



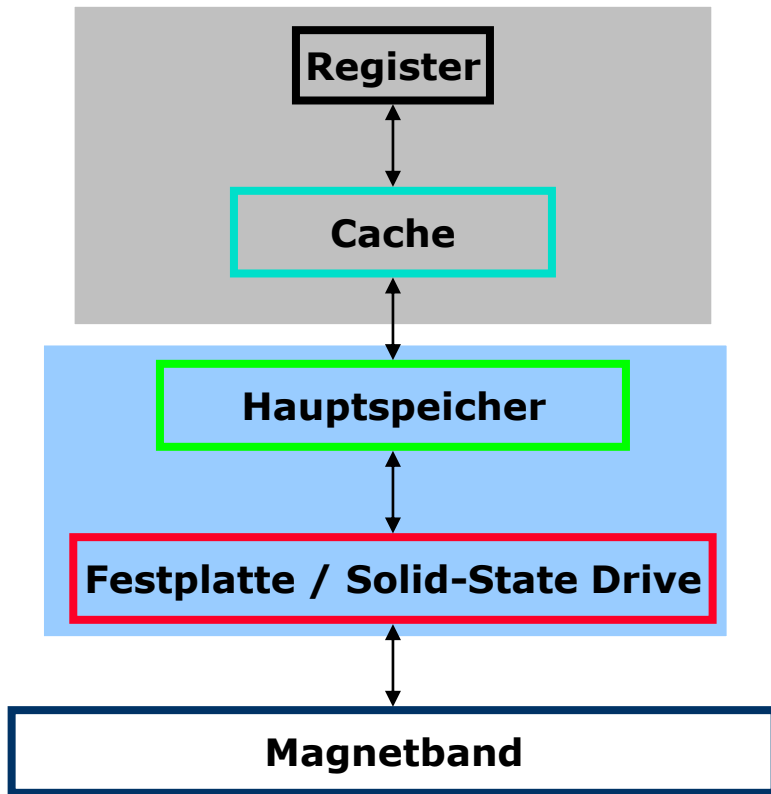
Inhalt Vorlesung

- Aufbau einfacher Rechner
- Überblick: Aufgabe, historische Entwicklung, unterschiedliche Arten von Betriebssystemen
- Betriebssysteme: Komponenten & Konzepte
 - Dateisysteme
 - Prozesse
 - Nebenläufigkeit und wechselseitiger Ausschluss
 - Deadlocks
 - Scheduling
 - **Speicherverwaltung**
 - Sicherheit

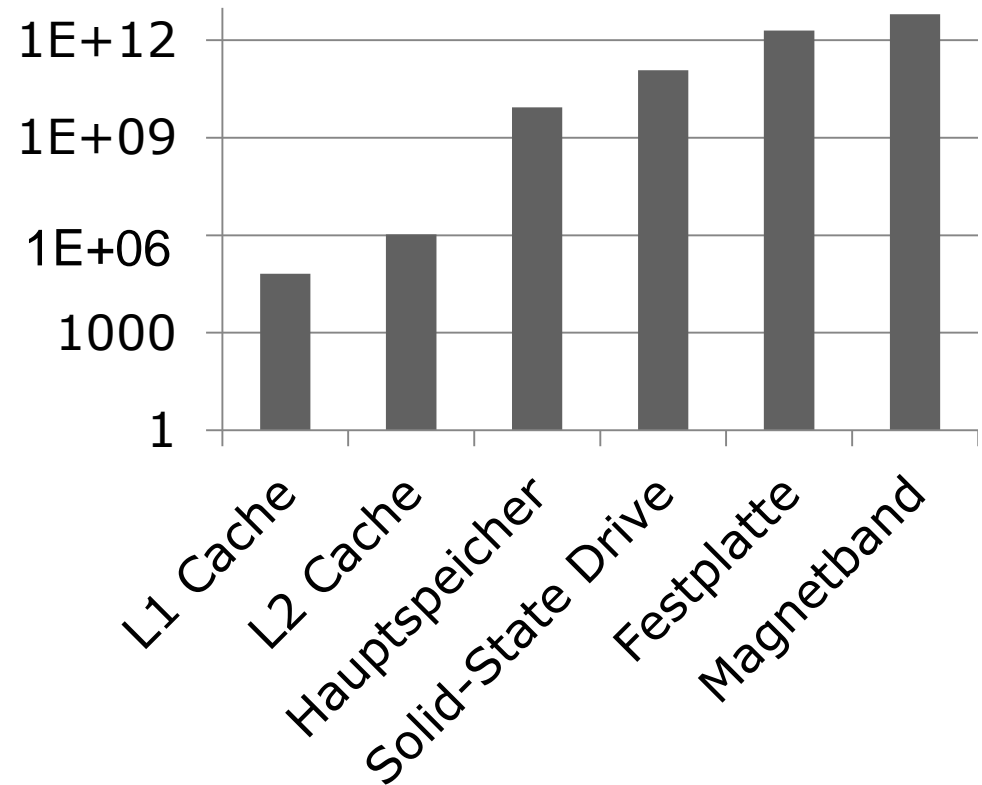
Speicherhierarchie



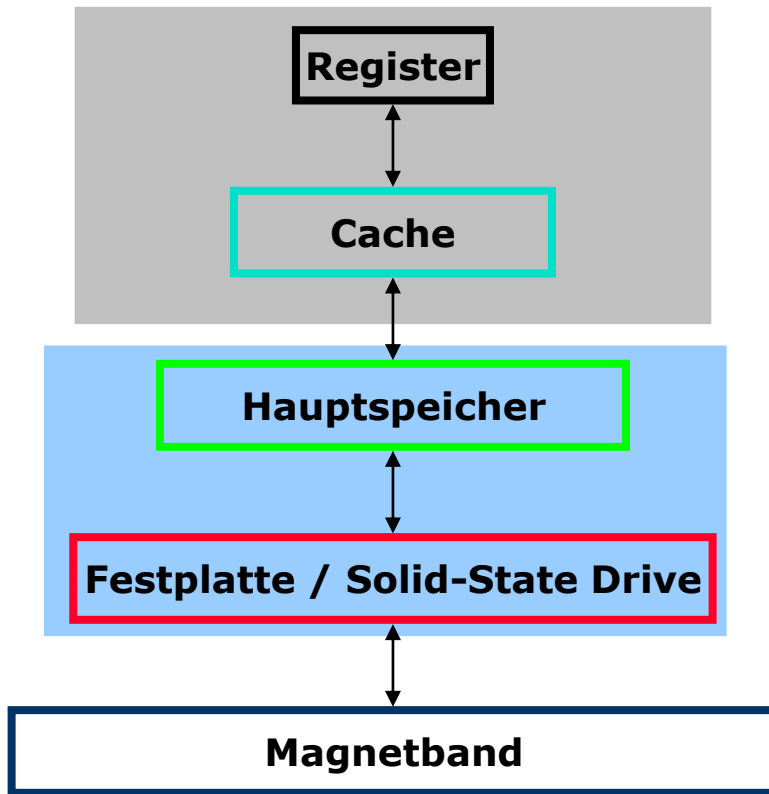
Speicherhierarchie



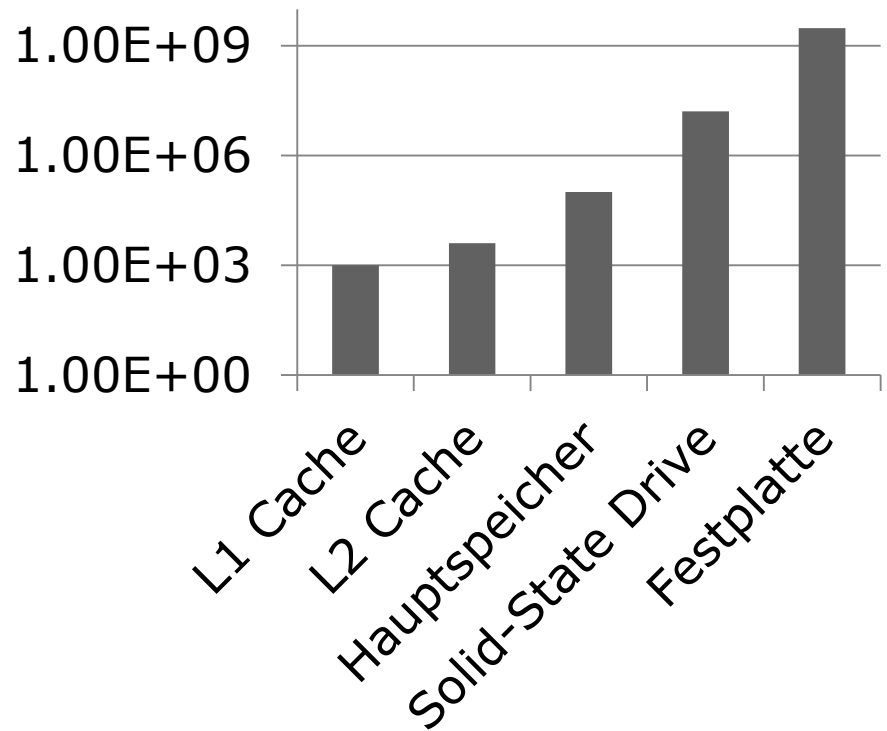
Typische Kapazität [Byte]



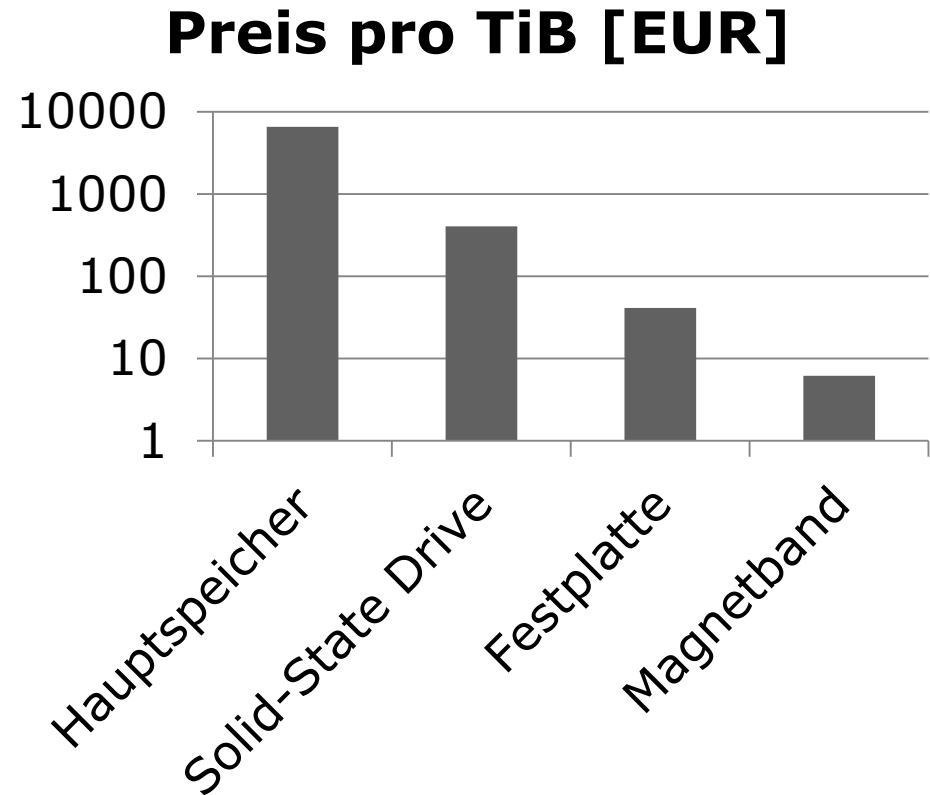
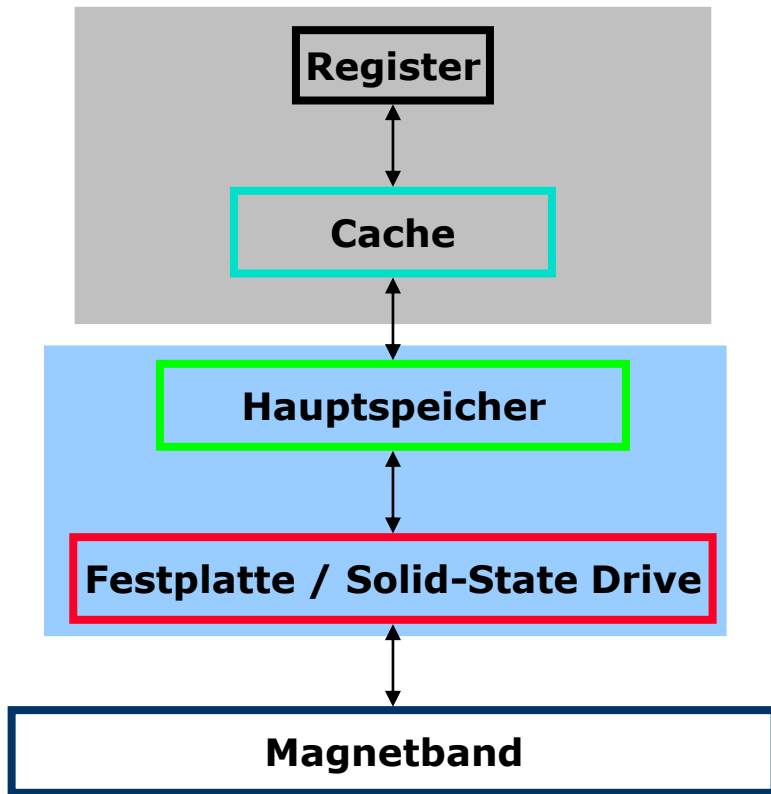
Speicherhierarchie



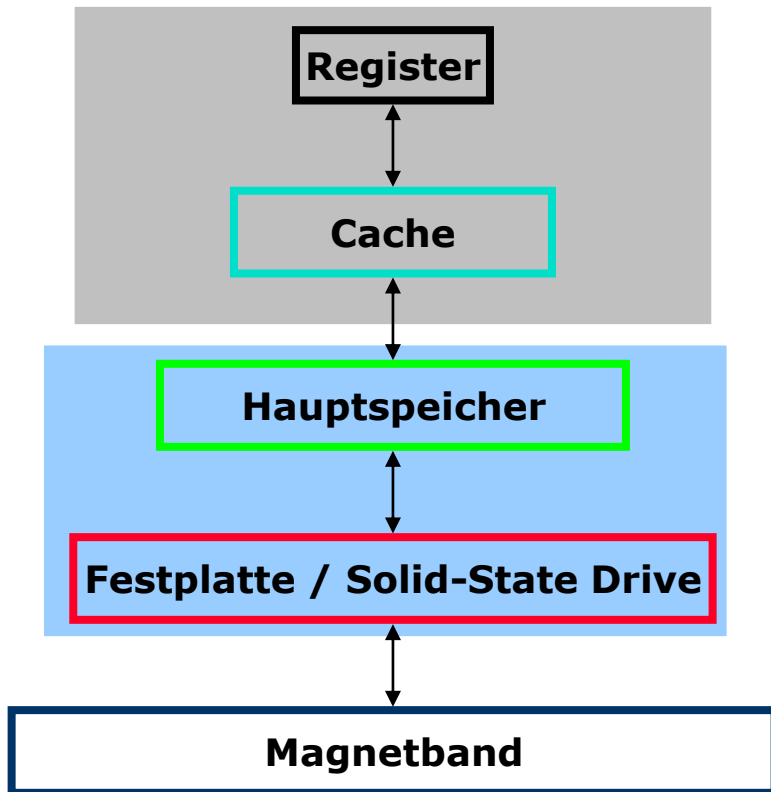
Zugriffszeit [ps]



Speicherhierarchie



Speicherhierarchie



Kleine Kapazität, kurze Zugriffszeit,
hohe Kosten pro Bit

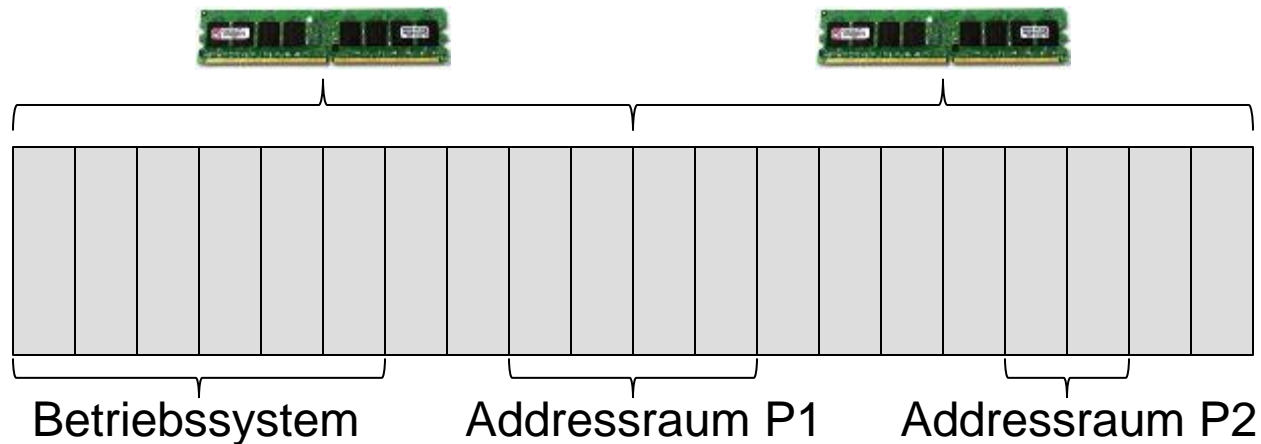
Große Kapazität, lange Zugriffszeit,
niedrige Kosten pro Bit

Einführung

- Hauptspeicher ist in mehrere Bereiche aufgeteilt
 - Bereich für das Betriebssystem
 - Bereich für Prozesse
- Speicherverwaltung: Dynamische Aufteilung entsprechend aktueller Prozesse
- Speicher muss effizient aufgeteilt werden, damit möglichst viele Prozesse Platz haben

Adressraum

- Abstraktion vom physikalischen Speicher
- Speicherzellen im Hauptspeicher haben eindeutige Adresse
- Adressraum: Menge von Adressen, die ein Prozess benutzen darf (lesen / schreiben)
- Jeder Prozess hat eigenen Adressraum



Anforderungen an Speicherverwaltung

- Bereitstellung von Platz im Hauptspeicher für Betriebssystem und Prozesse
- Ziel aus Betriebssystemersicht: Möglichst viele Prozesse im Speicher
- Fünf wichtige Anforderungen:
 - Relokation
 - Schutz
 - Gemeinsame Nutzung
 - Logische Organisation
 - Physikalische Organisation

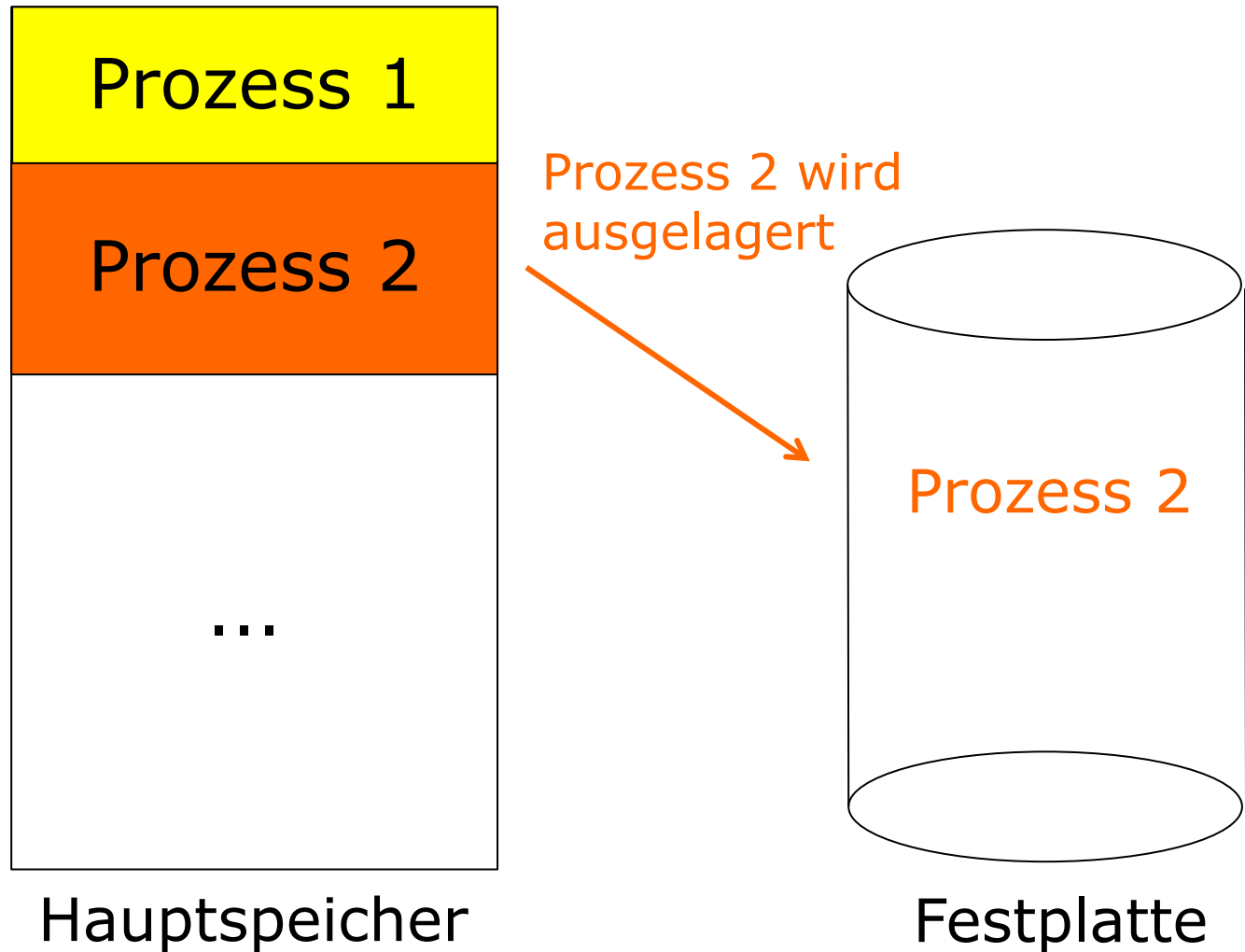
Anforderungen an Speicherverwaltung

- Bereitstellung von Platz im Hauptspeicher für Betriebssystem und Prozesse
- Ziel aus Betriebssystemersicht: Möglichst viele Prozesse im Speicher
- Fünf wichtige Anforderungen:
 - **Relokation**
 - Schutz
 - Gemeinsame Nutzung
 - Logische Organisation
 - Physikalische Organisation

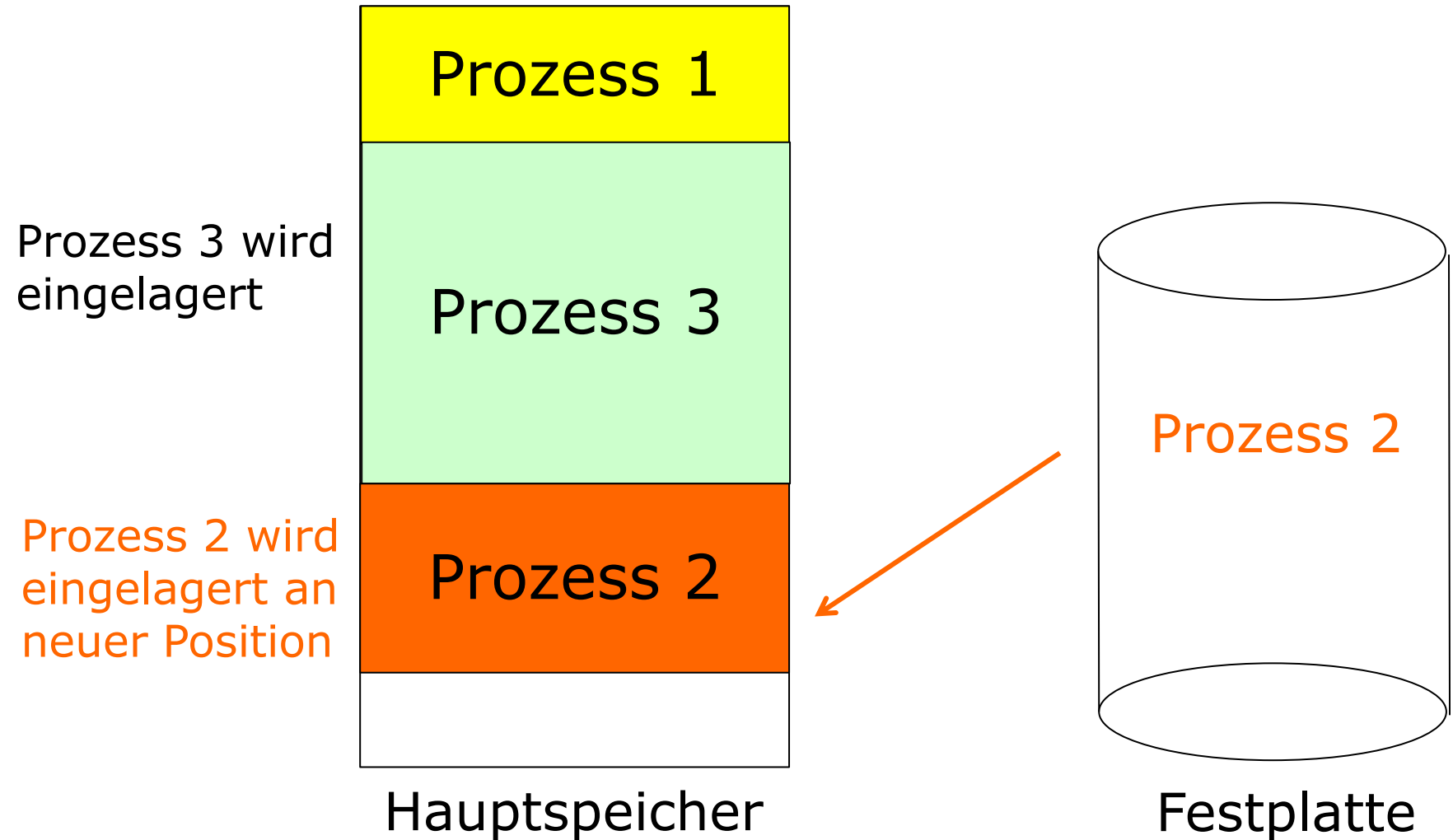
Relokation (1)

- Relokation = Verlagerung
- Mehrere Prozesse gleichzeitig im System
- Auslagern und Wiedereinlagern von Prozessen aus dem Hauptspeicher
- Ort der Einlagerung im Voraus **unbekannt**

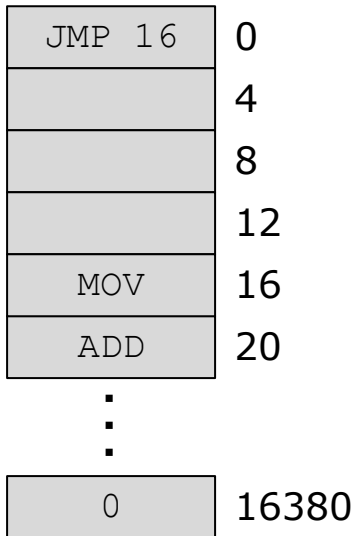
Relokation (2)



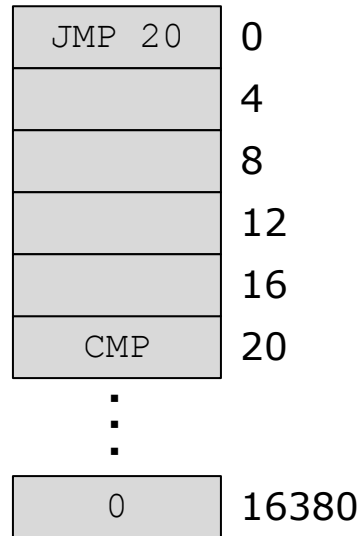
Relokation (3)



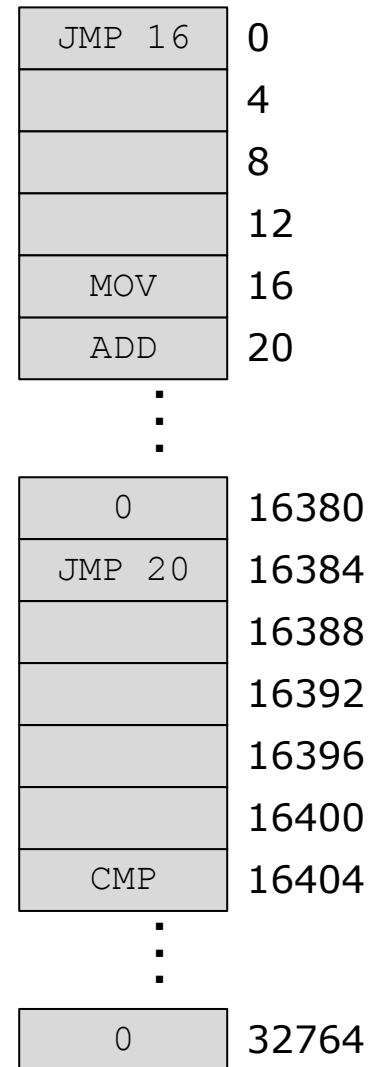
Beispiel Relokationsproblem



Programm A



Programm B

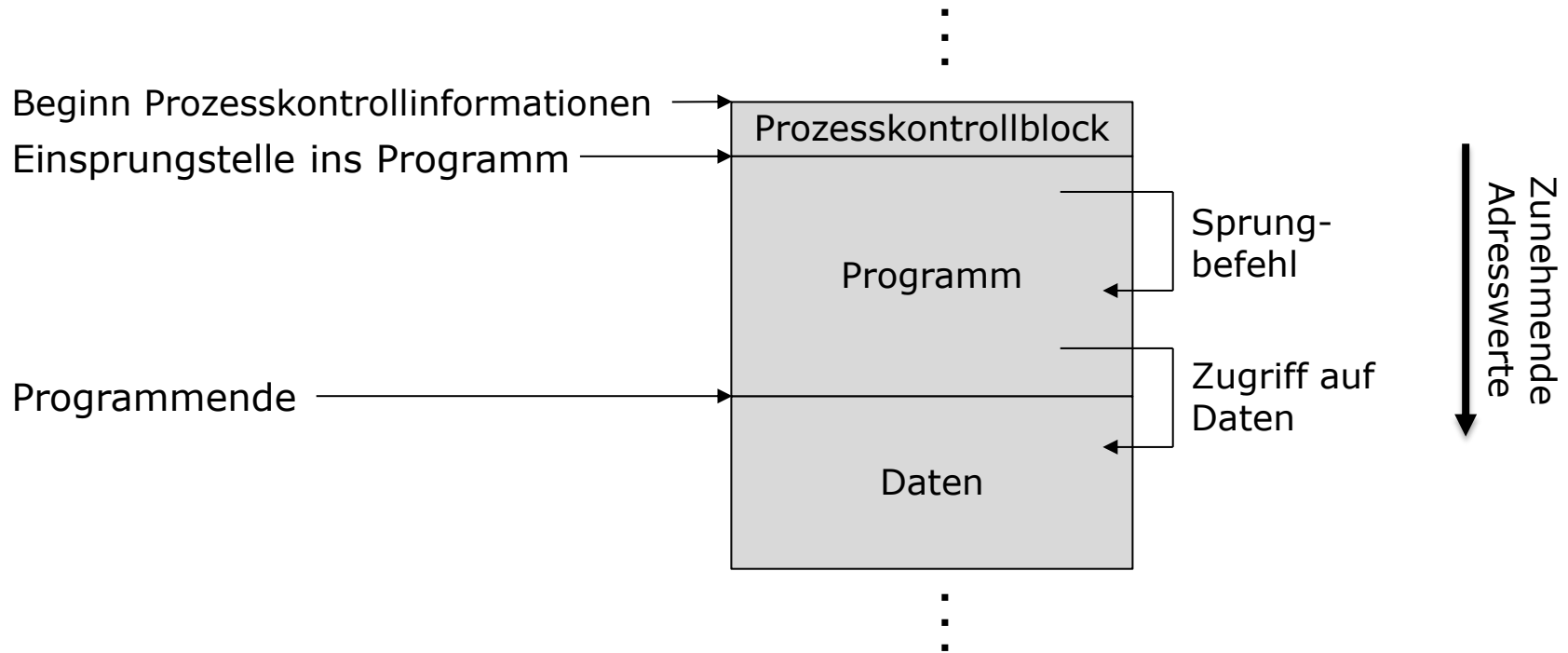


A und B im Speicher

Relokation (4)

- Problem: Speicherreferenzen innerhalb des Programms
- Absolute Sprungbefehle: Adresse auf den nächsten auszuführenden Befehl
- Datenzugriffsbefehle: Adresse des Bytes, das referenziert wird
- Prozessorhardware und Betriebssystem müssen die Speicherreferenzen in physikalische Speicheradressen übersetzen

Relokation (5)



- Beispiel Sprungbefehl: `JMP i`
- Beispiel Datenzugriffsbefehl: `MOV REG1, j`

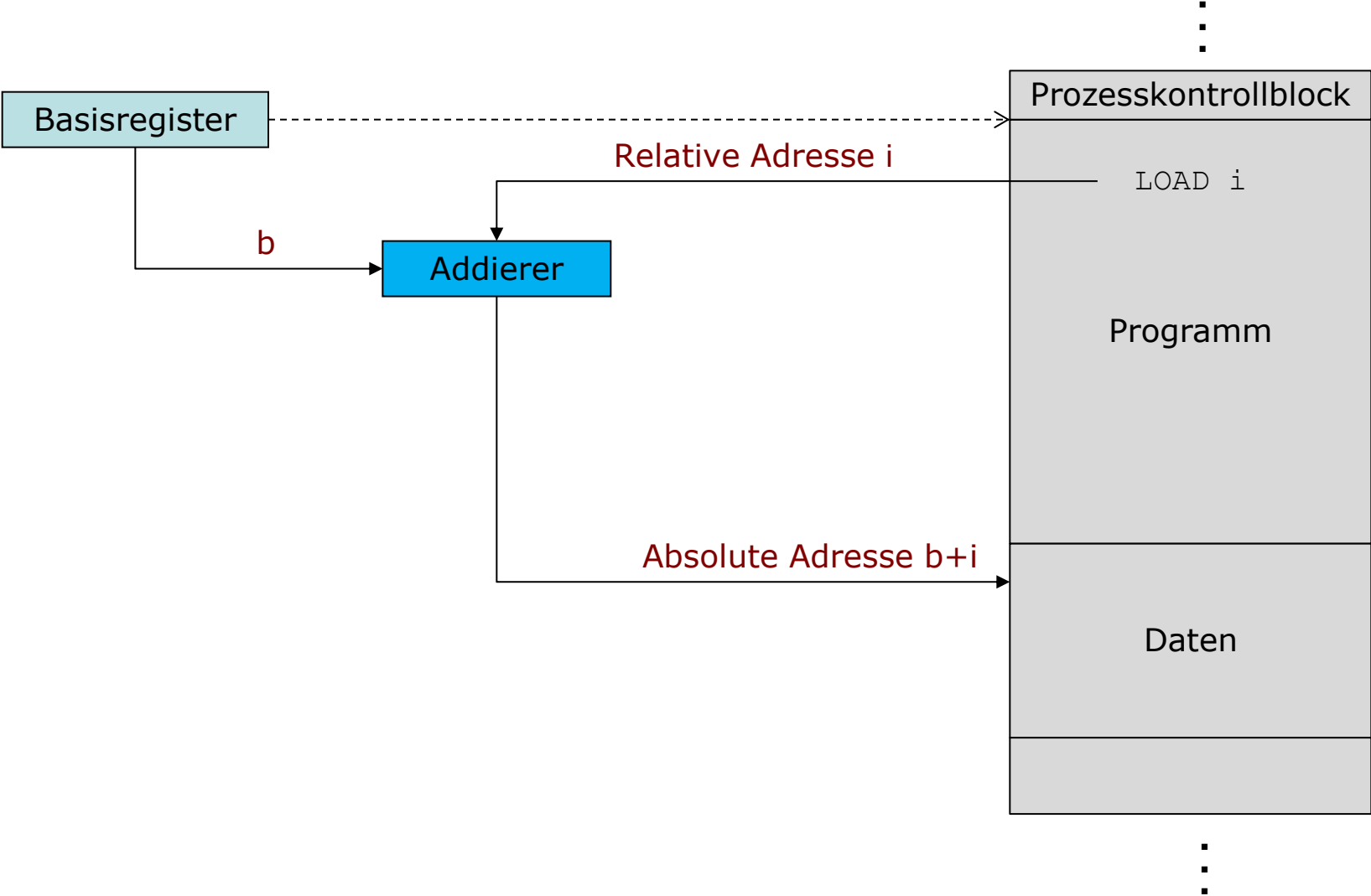
Relokation (6)

- **Physikalische bzw. absolute Adresse:** Konkrete Stelle im Hauptspeicher
- **Logische Adresse:** Bezug auf eine Speicherstelle, unabhängig von der aktuellen Zuteilung im Speicher
- **Relative Adresse:**
 - Spezialfall einer logischen Adresse
 - Adresse relativ zu einem bekannten Punkt (in der Regel Programmmanfang)

Relokation (7)

- Dynamisches Laden zur Laufzeit:
Berechnung von absoluten Adressen aus relativen Adressen durch Hardware
- Beim Einlagern: Adresse des Programm-
anfangs wird im **Basisregister** gespeichert

Relokation über Basisregister



Relokation (7)

- Dynamisches Laden zur Laufzeit:
Berechnung von absoluten Adressen aus relativen Adressen durch Hardware
- Beim Einlagern: Adresse des Programm-
anfangs wird im **Basisregister** gespeichert
- Absolute Adresse: Relative Adresse wird um
den Wert erhöht, der sich im Basisregister
befindet

Anforderungen an Speicherverwaltung

- Bereitstellung von Platz im Hauptspeicher für Betriebssystem und Prozesse
- Ziel aus Betriebssystemersicht: Möglichst viele Prozesse im Speicher
- Fünf wichtige Anforderungen:
 - Relokation
 - **Schutz**
 - Gemeinsame Nutzung
 - Logische Organisation
 - Physikalische Organisation

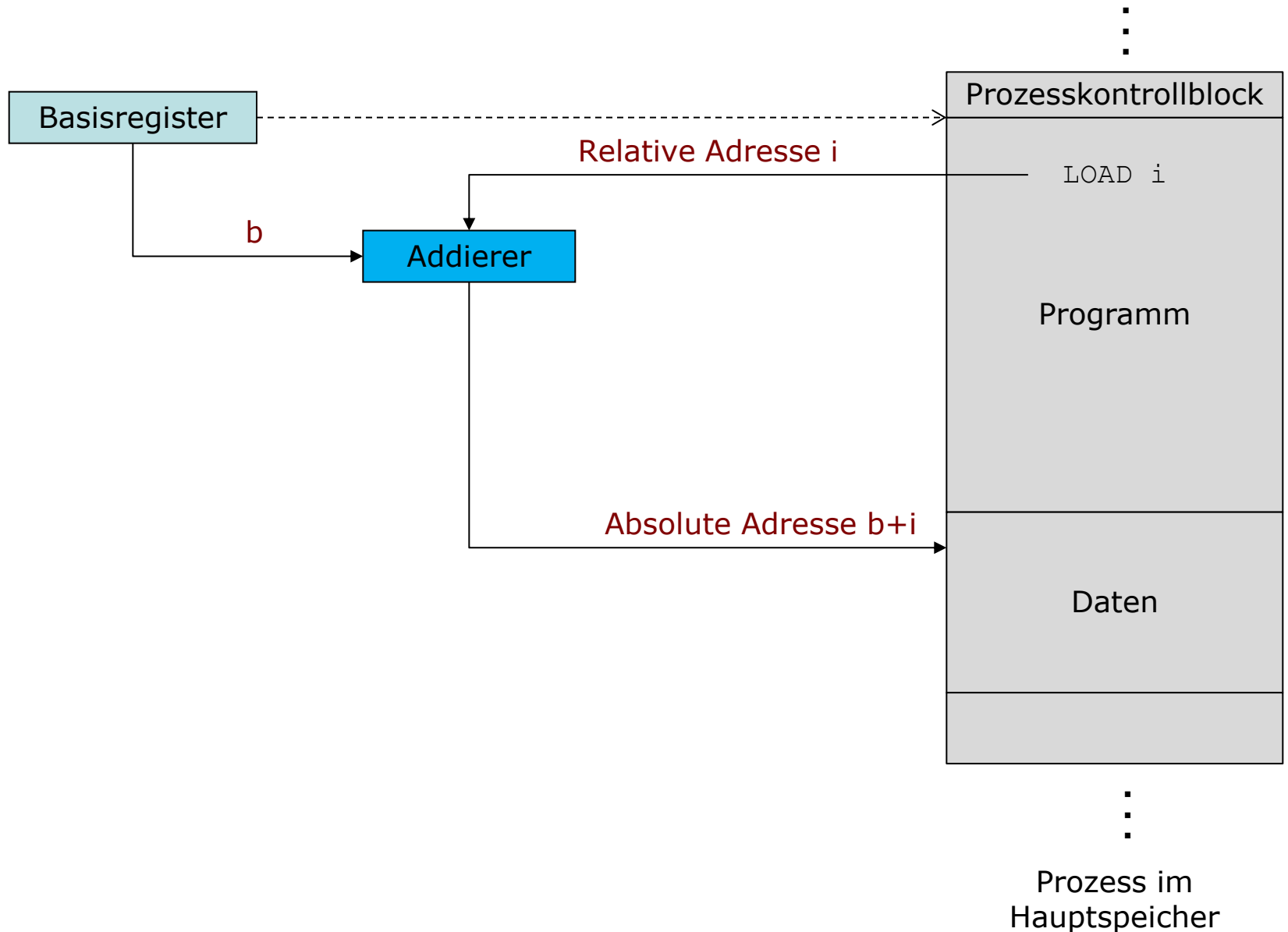
Schutz (1)

- Schutz von Prozessen gegen Störungen durch andere Prozesse
- Überprüfung aller Speicherzugriffe notwendig
- Schwierigkeit: In der Regel und auch im Allgemeinen nicht zur Übersetzungszeit eines Programms überprüfbar
- Grund: Dynamisch berechnete Adressen während der Laufzeit, absolute Adressen nicht bekannt (siehe Theoretische Informatik)

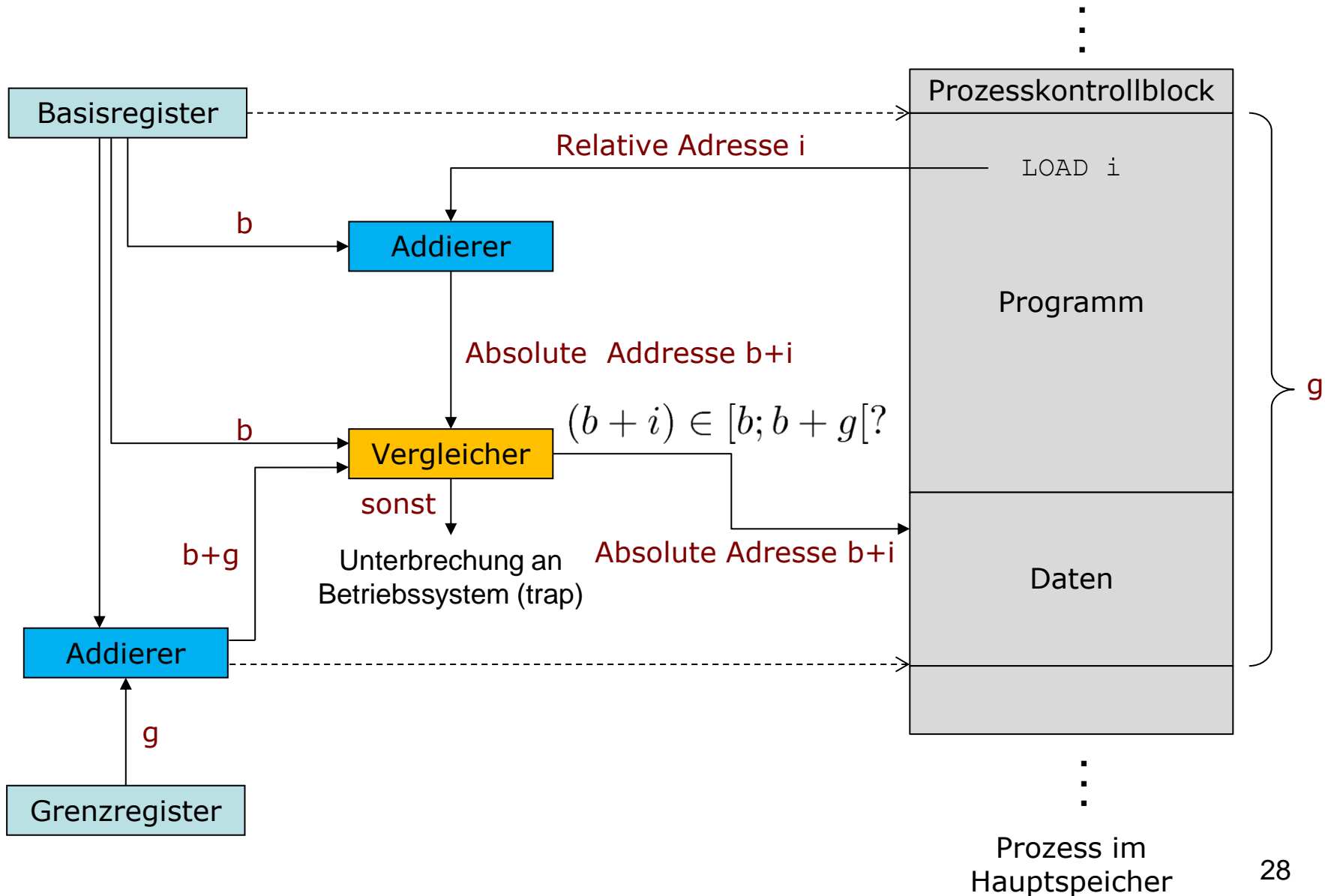
Schutz (2)

- Lösung: Dynamische Überprüfung zur Laufzeit
- Ggf. Abbruch von Befehlen bei Zugriff auf Datenbereich anderer Prozesse
- **Grenzregister/Limitregister**: Enthält die Größe des Adressraums eines Programms

Relokation und Schutz (1)



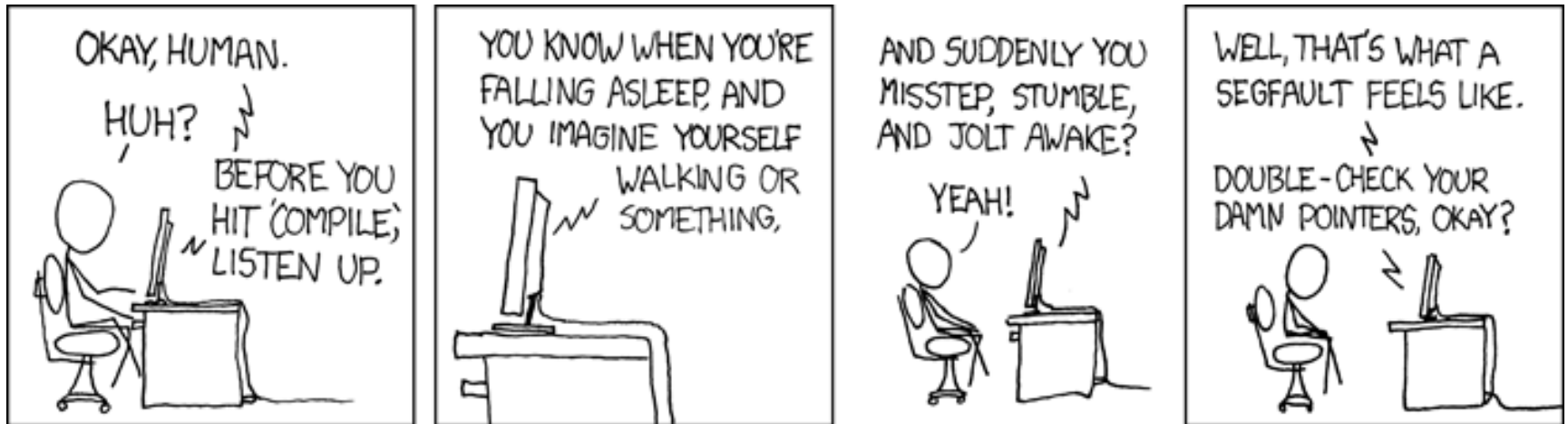
Relokation und Schutz (1)



Relokation und Schutz (2)

- Absolute Adresse: Relative Adresse wird um den Wert erhöht, der sich im Basisregister befindet
- Vergleich der resultierenden Adresse
 - Mit Basisregister
 - Mit Basisregister + Grenzregister
- Befehlsausführung nur, wenn die Adresse innerhalb der Grenzen liegt, sonst Interrupt

Relokation und Schutz (3)



<http://xkcd.com/371/>

Anforderungen an Speicherverwaltung

- Bereitstellung von Platz im Hauptspeicher für Betriebssystem und Prozesse
- Ziel aus Betriebssystemersicht: Möglichst viele Prozesse im Speicher
- Fünf wichtige Anforderungen:
 - Relokation
 - Schutz
 - **Gemeinsame Nutzung**
 - Logische Organisation
 - Physikalische Organisation

Gemeinsame Nutzung

- Kontrollierter Zugriff mehrerer Prozesse auf gemeinsam genutzte Bereiche des Speichers
- Anwendungsbeispiele:
 - Ausführung des gleichen Programms durch eine Reihe von Prozessen, Code nur einmal im Speicher
 - Zugriff auf dieselbe Datenstruktur bei Zusammenarbeit von Prozessen
 - Kooperation von Prozessen über gemeinsam genutzten Datenspeicher („Shared Memory“)

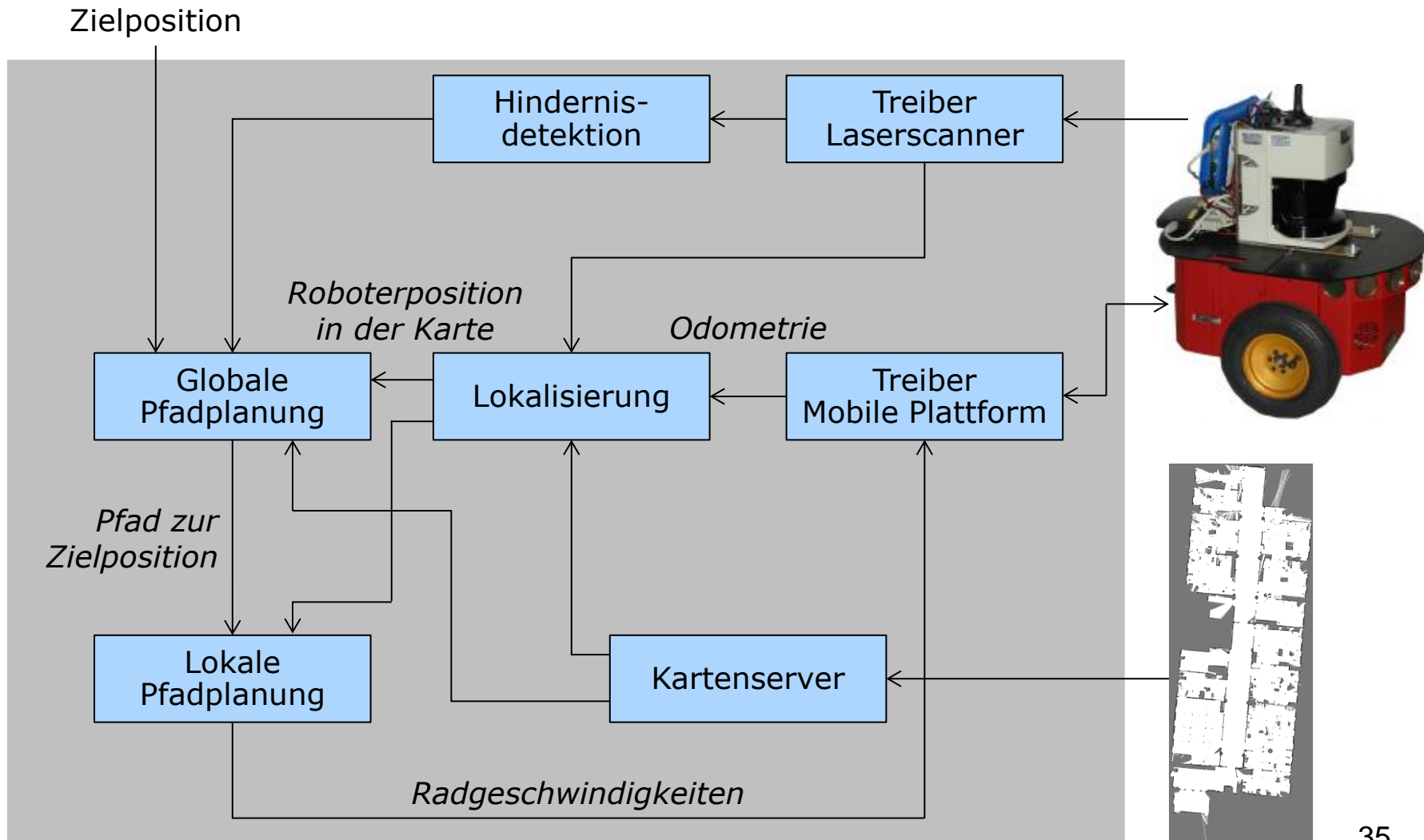
Anforderungen an Speicherverwaltung

- Bereitstellung von Platz im Hauptspeicher für Betriebssystem und Prozesse
- Ziel aus Betriebssystemersicht: Möglichst viele Prozesse im Speicher
- Fünf wichtige Anforderungen:
 - Relokation
 - Schutz
 - Gemeinsame Nutzung
 - **Logische Organisation**
 - Physikalische Organisation

Logische Organisation

- Logischer Aufbau großer Programme:
 - Verschiedene Module
 - Unabhängig übersetzt; Referenzen auf Funktionen in anderen Modulen werden erst zur Laufzeit aufgelöst
 - Verschiedene Module können unterschiedliche Grade von Schutz besitzen (z.B. nur lesen / ausführen)
 - Gemeinsame Nutzung von Modulen durch verschiedene Prozesse
- Betriebssystem muss mit Modulen umgehen können

Exkurs Logische Organisation: Mobile Roboterplattform



Anforderungen an Speicherverwaltung

- Bereitstellung von Platz im Hauptspeicher für Betriebssystem und Prozesse
- Ziel aus Betriebssystemersicht: Möglichst viele Prozesse im Speicher
- Fünf wichtige Anforderungen:
 - Relokation
 - Schutz
 - Gemeinsame Nutzung
 - Logische Organisation
 - **Physikalische Organisation**

Physikalische Organisation

- Betrachte zwei Ebenen
 - Hauptspeicher (schnell, teuer, flüchtig)
 - Festplatte (langsam, billig, nicht flüchtig)
- Grundproblem: Daten zwischen Haupt- und Sekundärspeicher verschieben
 - Aufwändig, erschwert durch Multiprogramming
 - Verwaltung durch das Betriebssystem

Grundlegende Methoden der Speicherverwaltung

Partitionierung

- Speicheraufteilung zwischen verschiedenen Prozessen (Partitionierung mit festen Grenzen)

Paging

- Einfaches Paging / kombiniert mit Konzept des virtuellen Speichers

Segmentierung

- Einfache Segmentierung / kombiniert mit Konzept des virtuellen Speichers

Grundlegende Methoden der Speicherverwaltung

Partitionierung

- Speicheraufteilung zwischen verschiedenen Prozessen (Partitionierung mit festen Grenzen)

Paging

- Einfaches Paging / kombiniert mit Konzept des virtuellen Speichers

Segmentierung

- Einfache Segmentierung / kombiniert mit Konzept des virtuellen Speichers

Partitionierung

- Aufteilung des Speichers in Bereiche mit festen Grenzen
- Fester, zusammenhängender Teil des Hauptspeichers für Betriebssystem
- Pro Prozess ein zusammenhängender Teil des Speichers
- Verschiedene Varianten:
 1. Statische Partitionierung
 2. Dynamische Partitionierung
 3. Buddy-Verfahren

Partitionierung

- Aufteilung des Speichers in Bereiche mit festen Grenzen
- Fester, zusammenhängender Teil des Hauptspeichers für Betriebssystem
- Pro Prozess ein zusammenhängender Teil des Speichers
- Verschiedene Varianten:
 - 1. Statische Partitionierung**
 2. Dynamische Partitionierung
 3. Buddy-Verfahren

Statische Partitionierung (1)

- Einteilung des Speichers in **feste Anzahl von Partitionen**
- **Zwei Varianten**

Alle Partitionen mit gleicher Länge

Betriebssystem
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB

Partitionen mit unterschiedlicher Länge

Betriebssystem
8 MB
4 MB
4 MB
8 MB
10 MB
14 MB

Statische Partitionierung (2)

Zuweisung von Partitionen an Prozesse:

- Bei Bereichen mit gleicher Länge: trivial
- Bei Bereichen mit variabler Länge: Kleinste verfügbare Partition, die gerade noch ausreicht (Verwaltung nicht trivial)
- Oft Speicherbedarf nicht im Voraus feststellbar (dafür Verfahren des virtuellen Speichers, siehe später)

Statische Partitionierung (3)

Probleme bei gleich großen Partitionen:

- Programm möglicherweise zu groß für Partition
- Interne Fragmentierung:
Platzverschwendung, wenn Programm kleiner als Größe der zugeordneten Partition
- Fest vorgegebene, maximale Anzahl von Prozessen im Speicher

Partitionen unterschiedlicher Länge

- Größere Programme können untergebracht werden
- Kleinere Programme führen zu geringerer interner Fragmentierung

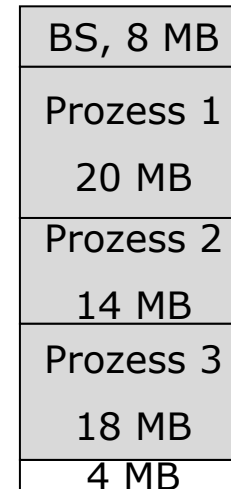
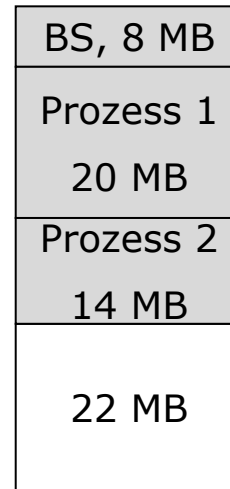
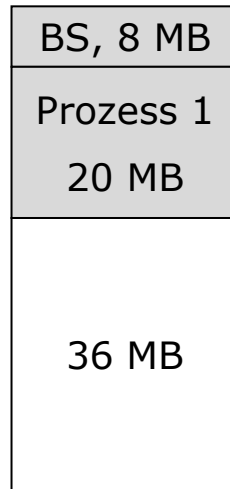
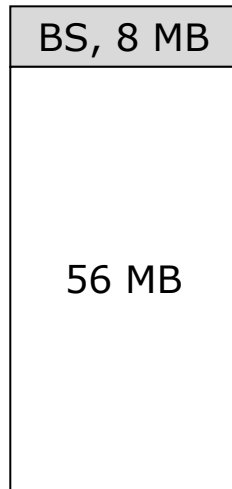
Partitionierung

- Aufteilung des Speichers in Bereiche mit festen Grenzen
- Fester, zusammenhängender Teil des Hauptspeichers für Betriebssystem
- Pro Prozess ein zusammenhängender Teil des Speichers
- Verschiedene Varianten:
 1. Statische Partitionierung
 2. **Dynamische Partitionierung**
 3. Buddy-Verfahren

Dynamische Partitionierung (1)

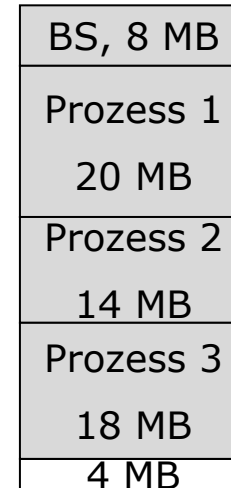
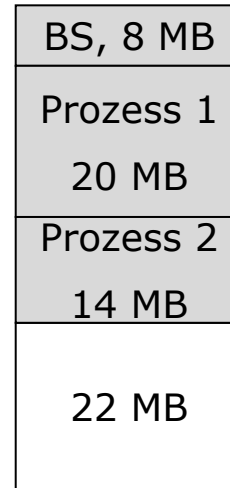
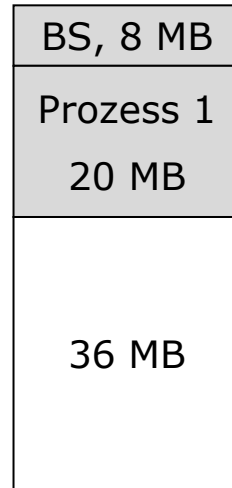
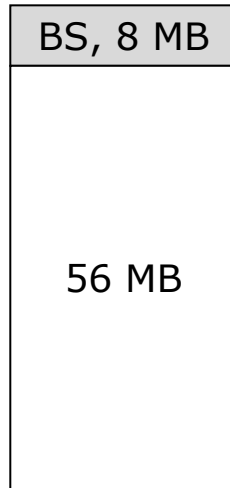
- Einteilung des Speichers in Partitionen
 - variable Länge
 - variable Anzahl
- Prozesse erhalten exakt passende Speicherbereiche (keine interne Fragmentierung)
- Aber: Ein- und Auslagern führt zu **externer Fragmentierung** mit einer Vielzahl kleiner Lücken, d.h. Speicherauslastung nimmt ab

Dynamische Partitionierung (2)

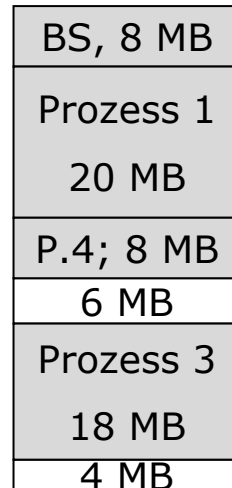
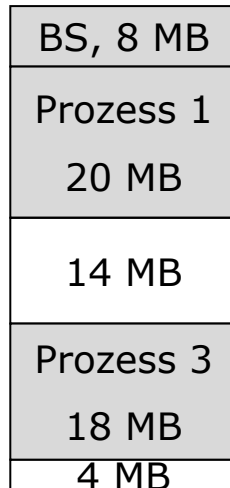


Anforderung:
Prozess 4
braucht 8 MB

Dynamische Partitionierung (2)



Anforderung:
Prozess 4
braucht 8 MB

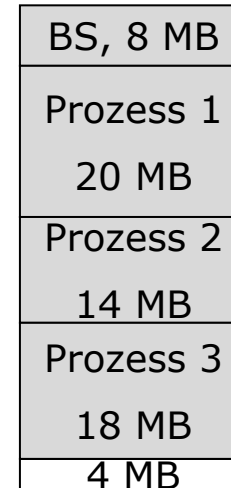
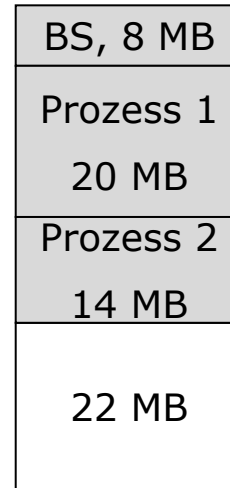
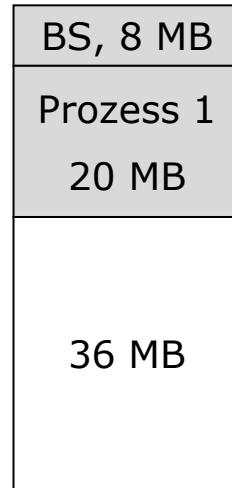
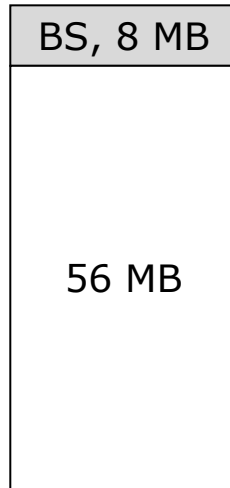


Genügend
Platz für
Prozess 4,
aber Lücke
entsteht

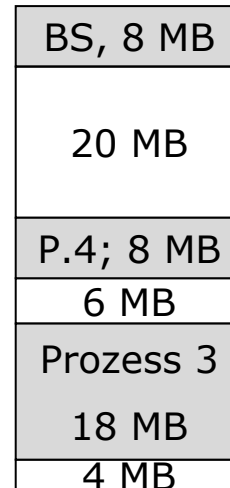
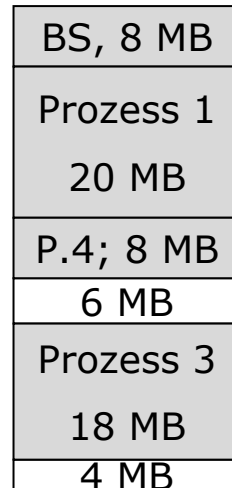
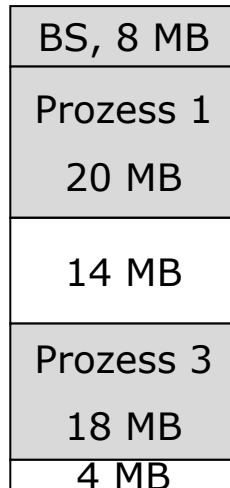
Annahme:
Kein Prozess
im
Hauptspeicher
bereit, aber
ausgelagerter
Prozess 2
(14MB) bereit

Prozess 2 wird
ausgelagert

Dynamische Partitionierung (2)



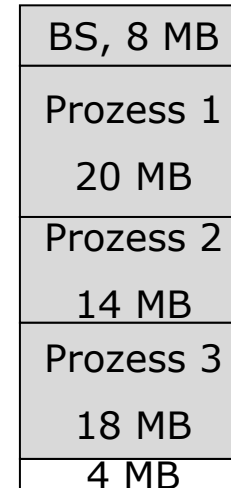
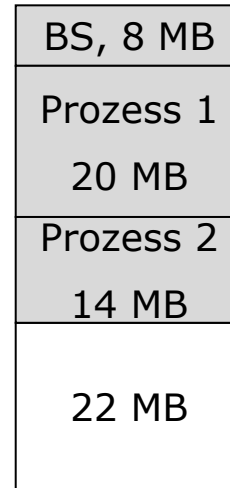
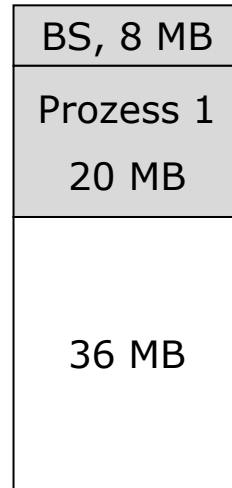
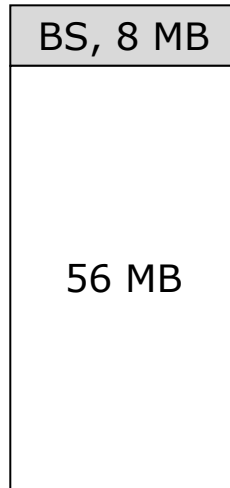
Anforderung:
Prozess 4
braucht 8 MB



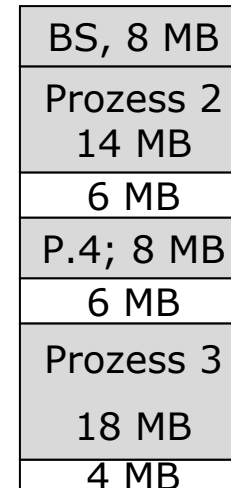
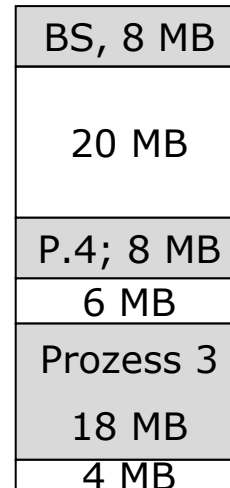
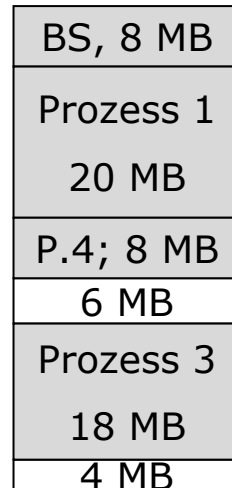
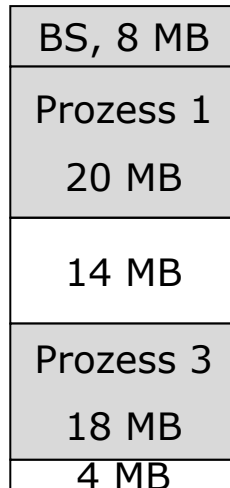
Prozess 2 wird
ausgelagert

Da nicht
genügend
Platz für
Prozess 2:
Prozess 1 wird
ausgelagert

Dynamische Partitionierung (2)



Anforderung:
Prozess 4
braucht 8 MB



Prozess 2 wird
wieder
eingelagert

Prozess 2 wird
ausgelagert

Dynamische Partitionierung (3)

Defragmentierung möglich, aber

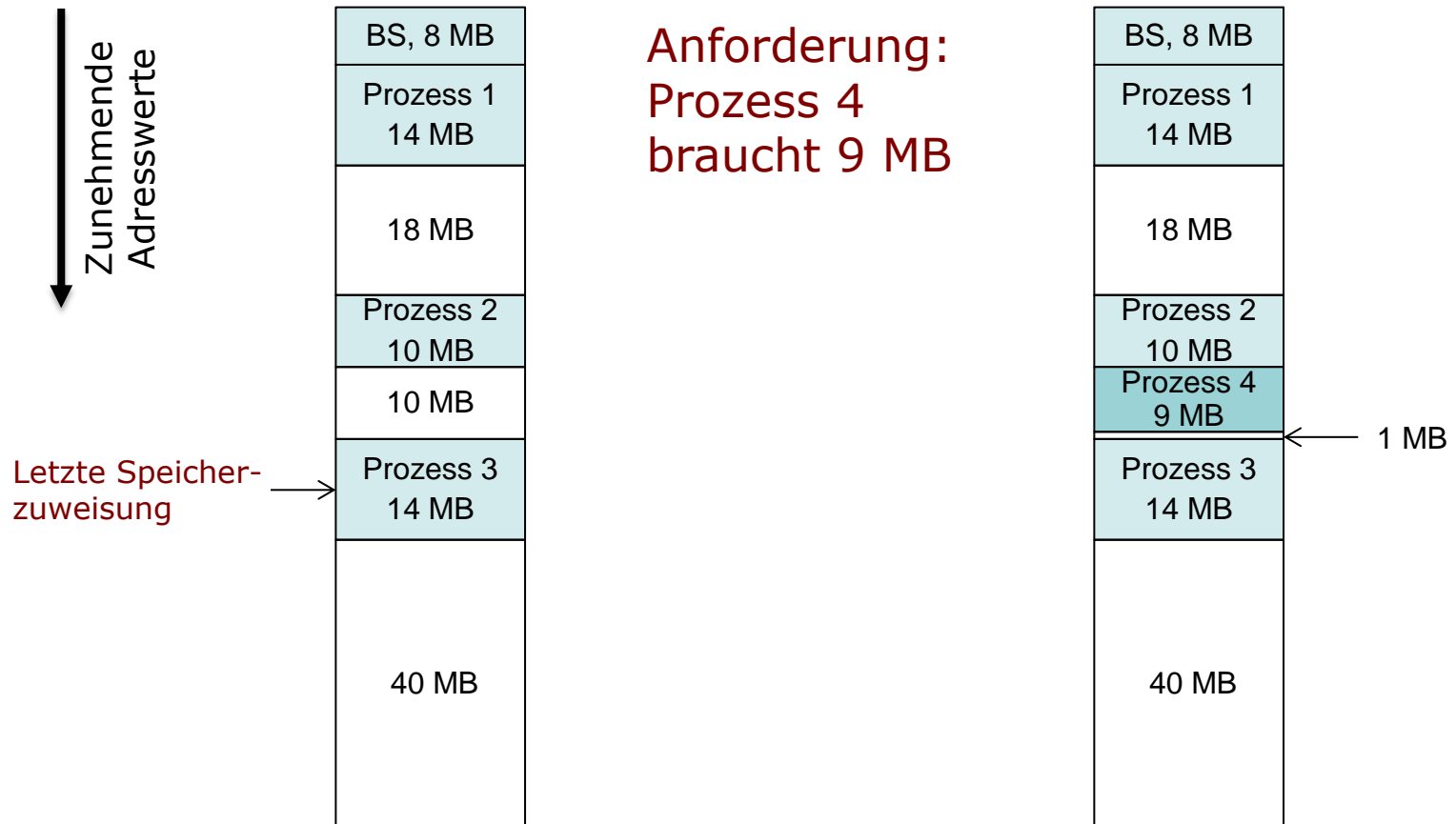
- Verschiebung aufwändig:
Speicherzuteilungsstrategie wichtig
- Speicherverdichtung nur erfolgreich, wenn
dynamische Relokation möglich

Dynamische Partitionierung (4)

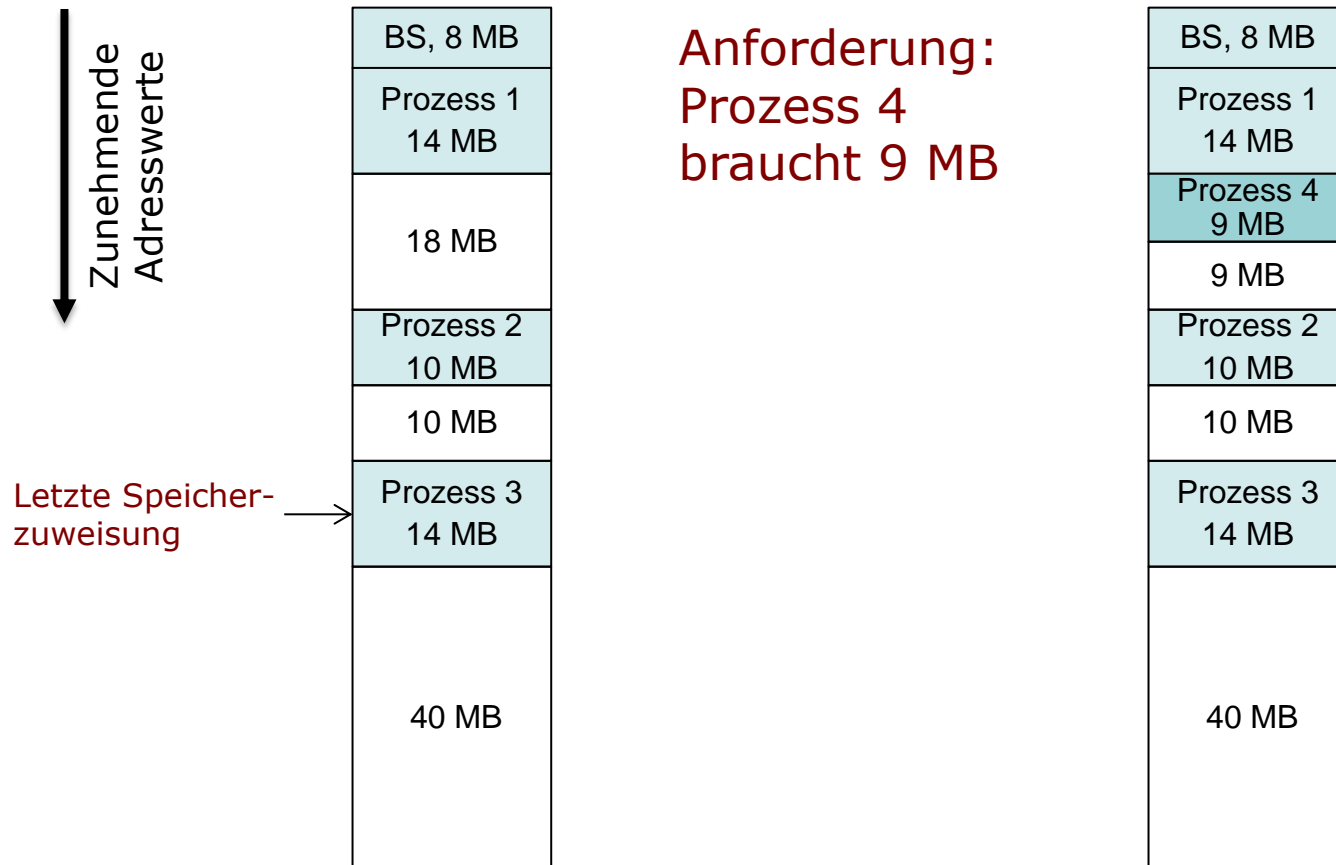
Speicherzuteilungsalgorithmen:

- **Best Fit:** Suche kleinsten Block, der ausreicht
- **First Fit:** Suche beginnend mit Speicheranfang bis ausreichend großer Block gefunden
- **Next Fit:** Suche beginnend mit der Stelle der letzten Speicherzuweisung

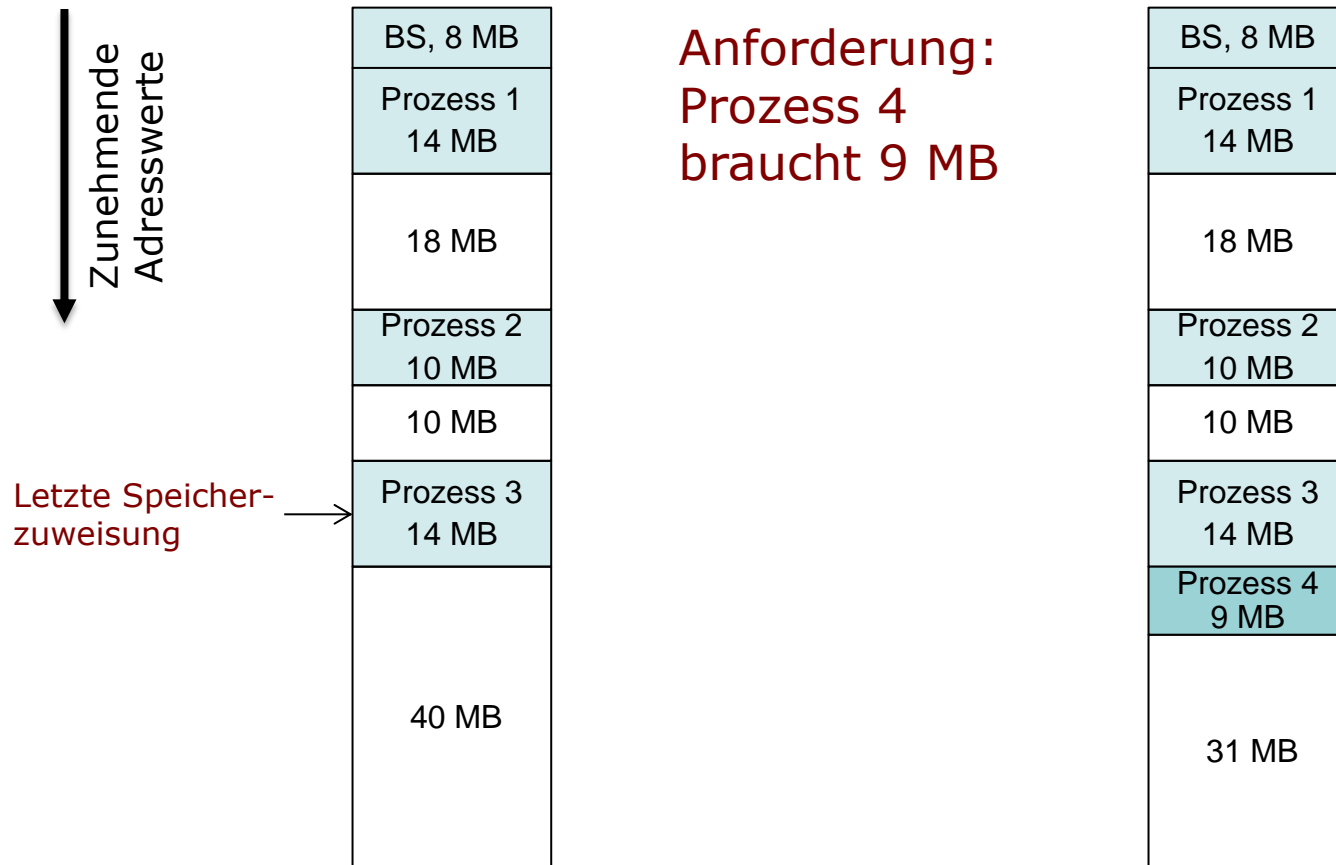
Speicherzuteilungsalgorithmen: Best Fit



Speicherzuteilungsalgorithmen: First Fit



Speicherzuteilungsalgorithmen: Next Fit



Dynamische Partitionierung (5)

Analyse der Speicherzuteilungsalgorithmen:

- Im Schnitt ist First Fit am besten!
- Next Fit: Etwas schlechter
 - Typischer Effekt: Schnelle Fragmentierung des größten freien Speicherblocks am Ende des Speichers
- Best Fit: Am schlechtesten
 - Schnell eine Reihe von sehr kleinen Fragmenten, Defragmentierung nötig
 - Außerdem: Suche braucht Zeit

Partitionierung

- Aufteilung des Speichers in Bereiche mit festen Grenzen
- Fester, zusammenhängender Teil des Hauptspeichers für Betriebssystem
- Pro Prozess ein zusammenhängender Teil des Speichers
- Verschiedene Varianten:
 1. Statische Partitionierung
 2. Dynamische Partitionierung
 3. **Buddy-Verfahren**

Nachteile Partitionierung

- **Statische** Partitionierung:
 - Anzahl von Prozessen im Speicher beschränkt
 - Interne Fragmentierung
- **Dynamische** Partitionierung:
 - Schwierigere Verwaltung
 - Externe Fragmentierung
- **Buddy-System** (Halbierungsverfahren):
Kompromiss zwischen statischer und dynamischer Partitionierung

Buddy System (1)

- Dynamische Anzahl nicht-ausgelagerter Prozesse
- Interne Fragmentierung **beschränkt**
- **Keine** explizite Defragmentierung
- **Effiziente Suche** nach „passendem Block“

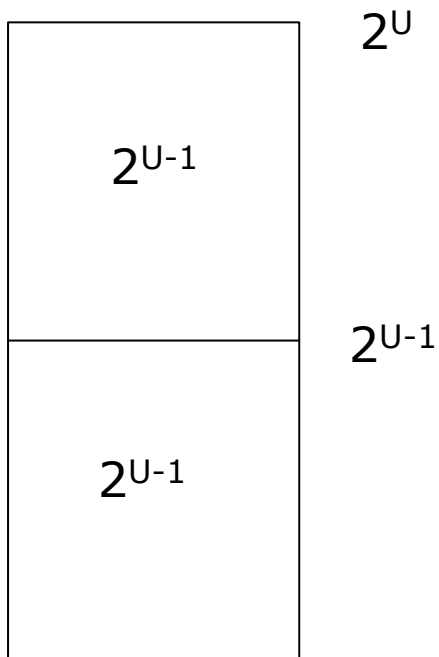
Buddy System (2)

- Verwalte Speicherblöcke der Größe 2^K
 - mit $L \leq K \leq U$, wobei
 - 2^L = Größe des kleinsten zuteilbaren Blocks
 - 2^U = Größe des größten zuteilbaren Blocks
(i.d.R. Gesamtgröße des verfügbaren Speichers)
- Zu Beginn: Es existiert genau ein Block der Größe 2^U

Buddy System (3)

Anforderung eines Blocks der Größe s :

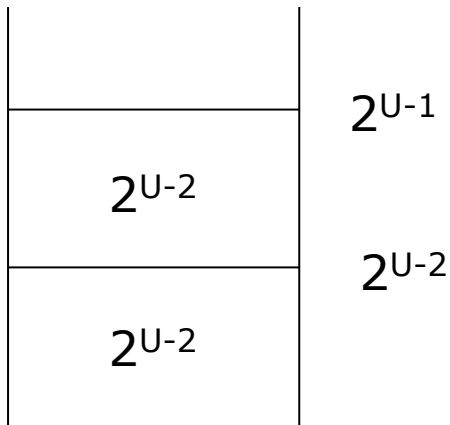
- Wenn $2^{U-1} < s \leq 2^U$: Weise gesamten Speicher zu
- Sonst: Teile auf in zwei Blöcke der Größe 2^{U-1}



Buddy System (4)

Anforderung eines Blocks der Größe s :

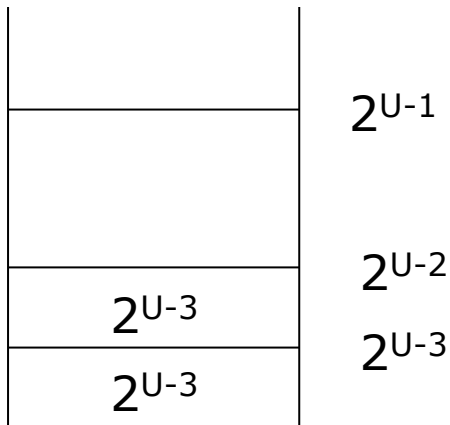
- Wenn $2^{U-1} < s \leq 2^U$: Weise gesamten Speicher zu
- Sonst: Teile auf in zwei Blöcke der Größe 2^{U-1}
- Wenn $2^{U-2} < s \leq 2^{U-1}$: Weise Block der Größe 2^{U-1} zu
- Sonst: Wähle Block der Größe 2^{U-1} und halbiere



Buddy System (4)

Anforderung eines Blocks der Größe s :

- Wenn $2^{U-1} < s \leq 2^U$: Weise gesamten Speicher zu
- Sonst: Teile auf in zwei Blöcke der Größe 2^{U-1}
- Wenn $2^{U-2} < s \leq 2^{U-1}$: Weise Block der Größe 2^{U-1} zu
- Sonst: Wähle Block der Größe 2^{U-1} und halbiere
- ...



Buddy System (4)

Anforderung eines Blocks der Größe s :

- Wenn $2^{U-1} < s \leq 2^U$: Weise gesamten Speicher zu
- Sonst: Teile auf in zwei Blöcke der Größe 2^{U-1}
- Wenn $2^{U-2} < s \leq 2^{U-1}$: Weise Block der Größe 2^{U-1} zu
- Sonst: Wähle Block der Größe 2^{U-1} und halbiere
- ...
- Fahre fort bis zu Blöcken der Größe 2^K mit $2^{K-1} < s \leq 2^K$ oder bis minimale Blockgröße erreicht
- Weise einen der beiden Blöcke zu (bzw. weise Block mit Minimalgröße zu)

Vorteil: Bei resultierendem Block ist der Verschnitt kleiner als die halbe Blockgröße

Buddy System (5)

- Verwalte für alle $L \leq K \leq U$ Listen mit freien Blöcken der Größe 2^K
- Allgemeiner Fall: Anforderung eines Blocks der Größe $2^{i-1} < s \leq 2^i$:
 - Vergib Block aus Liste i , wenn vorhanden
 - Sonst: Wähle Block aus nächstgrößerer nichtleerer Liste
 - Teile diesen rekursiv auf, bis ein Block der Größe 2^i vorhanden, weise diesen zu (bzw. weise Block mit Minimalgröße zu)

Buddy System (6)

- Wenn nach Freigabe eines Blocks der Größe 2^k der entsprechende **Partnerblock** der Größe 2^k ebenfalls frei ist:
 - Fasse die Blöcke zu einem Block der Größe 2^{k+1} zusammen
 - Mache ggf. rekursiv weiter
- Binärbaumdarstellung der Blockzuteilung

Buddy-System (7)

Beispiel: Speicher der Größe 1 GiB

Folge von Anforderungen und Freigaben:

- A fordert 100 MiB an
- B fordert 240 MiB an
- C fordert 64 MiB an
- D fordert 256 MiB an
- Freigabe B
- Freigabe A
- E fordert 75 MiB an
- Freigabe C
- Freigabe E
- Freigabe D

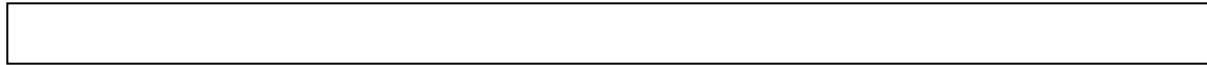
Annahme:

- Obergrenze der Blockgröße: 1 GiB
- Untergrenze: 64 MiB

Buddy-System (8)

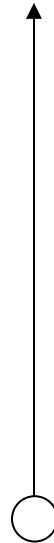
Zunächst verfügbar:

1 GiB



Freie Blöcke:

1 GiB: 1
512 MiB: 0
256 MiB: 0
128 MiB: 0
64 MiB: 0



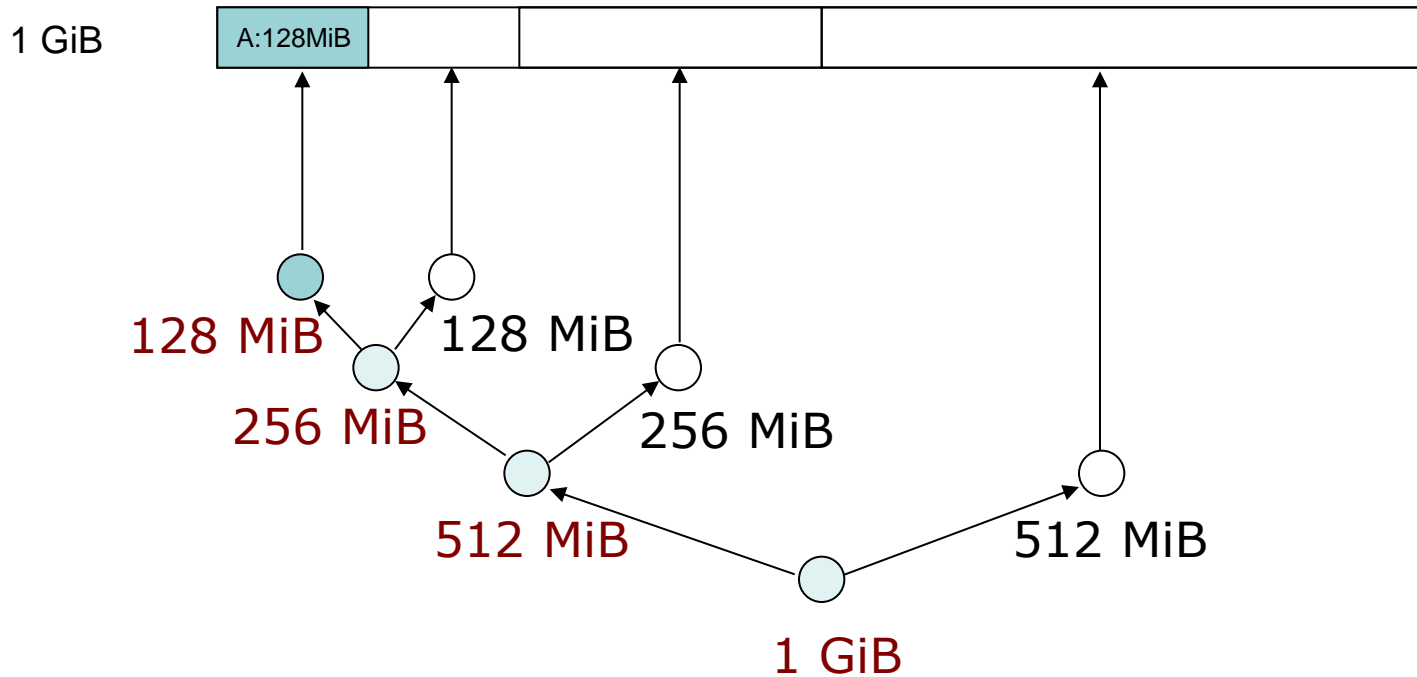
Gesamter Speicher als 1 Block verfügbar

Es folgt Anforderung A: 100 MiB, d.h. Block der Größe 128 MiB

Buddy-System (9)

Nach Anforderung A: 100 MiB, d.h. Block der Größe 128 MiB:

Freie Blöcke:



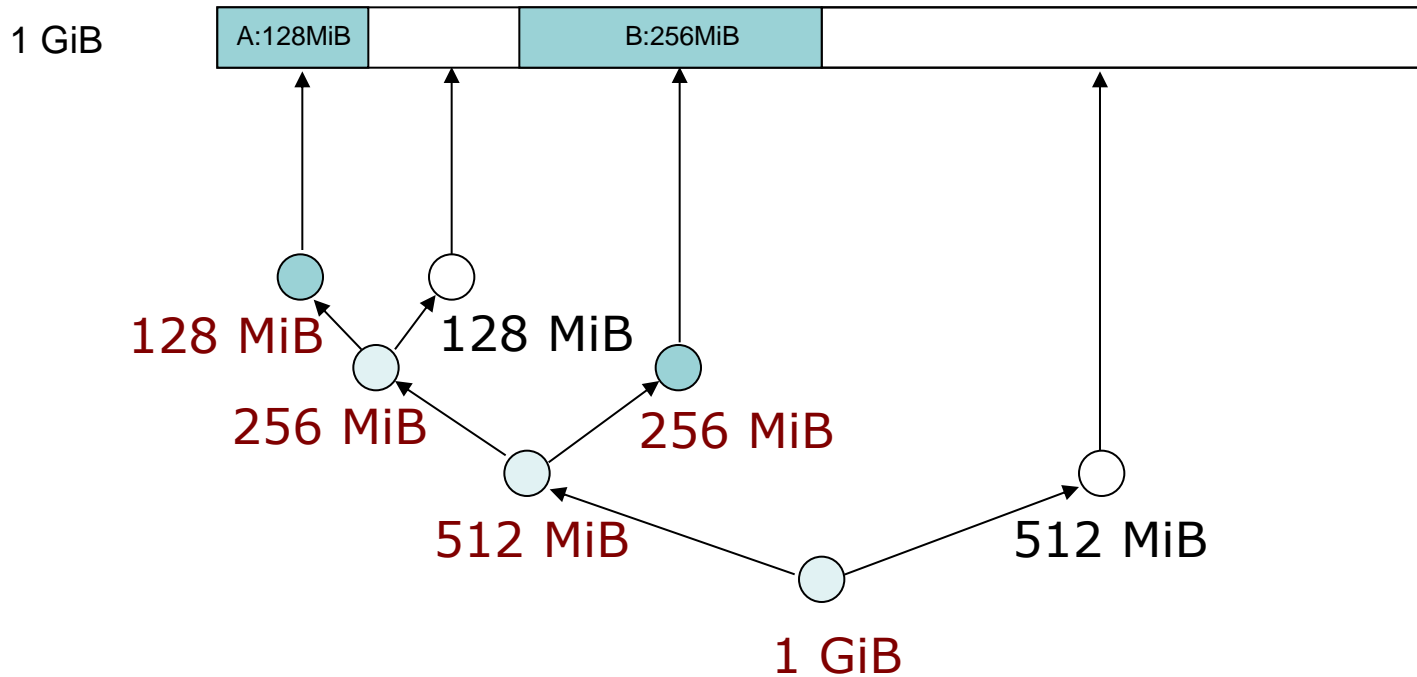
1 GiB: 0
512 MiB: 1
256 MiB: 1
128 MiB: 1
64 MiB: 0

Es folgt Anforderung B: 240 MiB, d.h. Block der Größe 256 MiB

Buddy-System (10)

Nach Anforderung B: 240 MiB, d.h. Block der Größe 256 MiB.

Freie Blöcke:



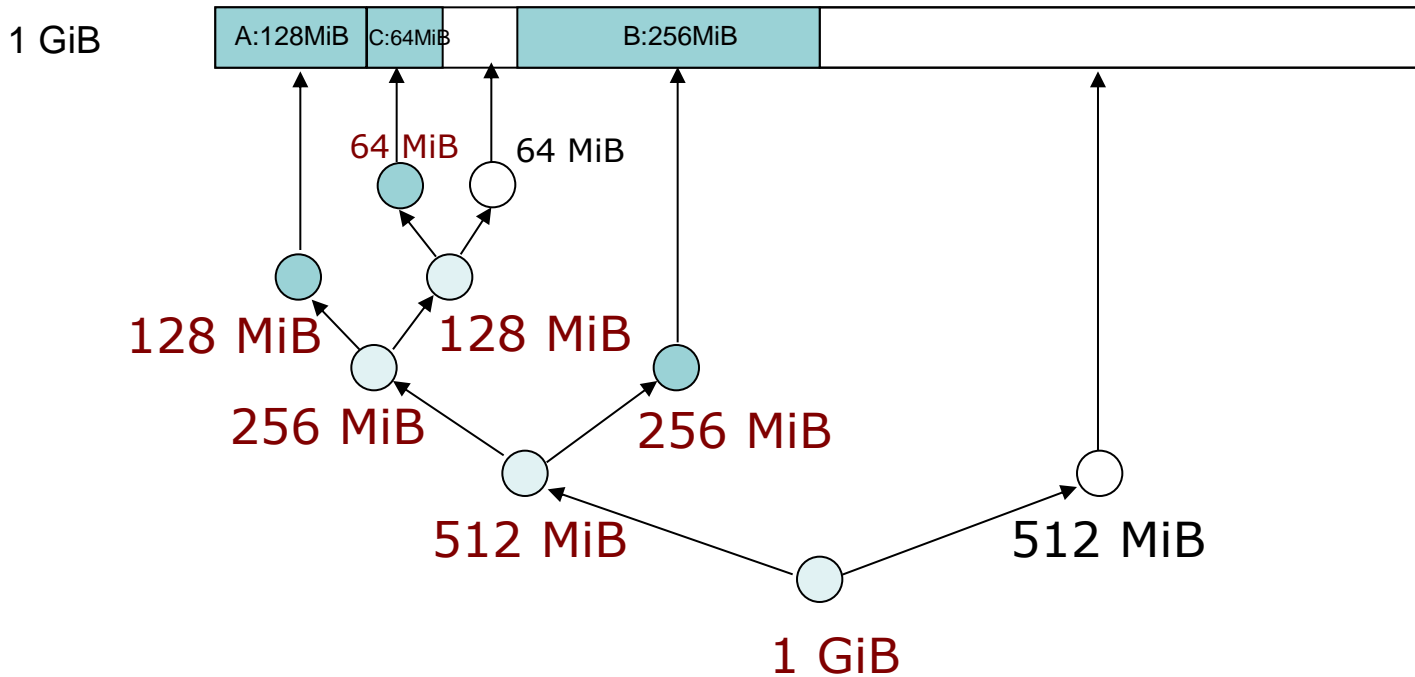
1 GiB: 0
512 MiB: 1
256 MiB: 0
128 MiB: 1
64 MiB: 0

Es folgt Anforderung C: 64 MiB, d.h. Block der Größe 64 MiB

Buddy-System (11)

Nach Anforderung C: 64 MiB, d.h. Block der Größe 64 MiB.

Freie Blöcke:



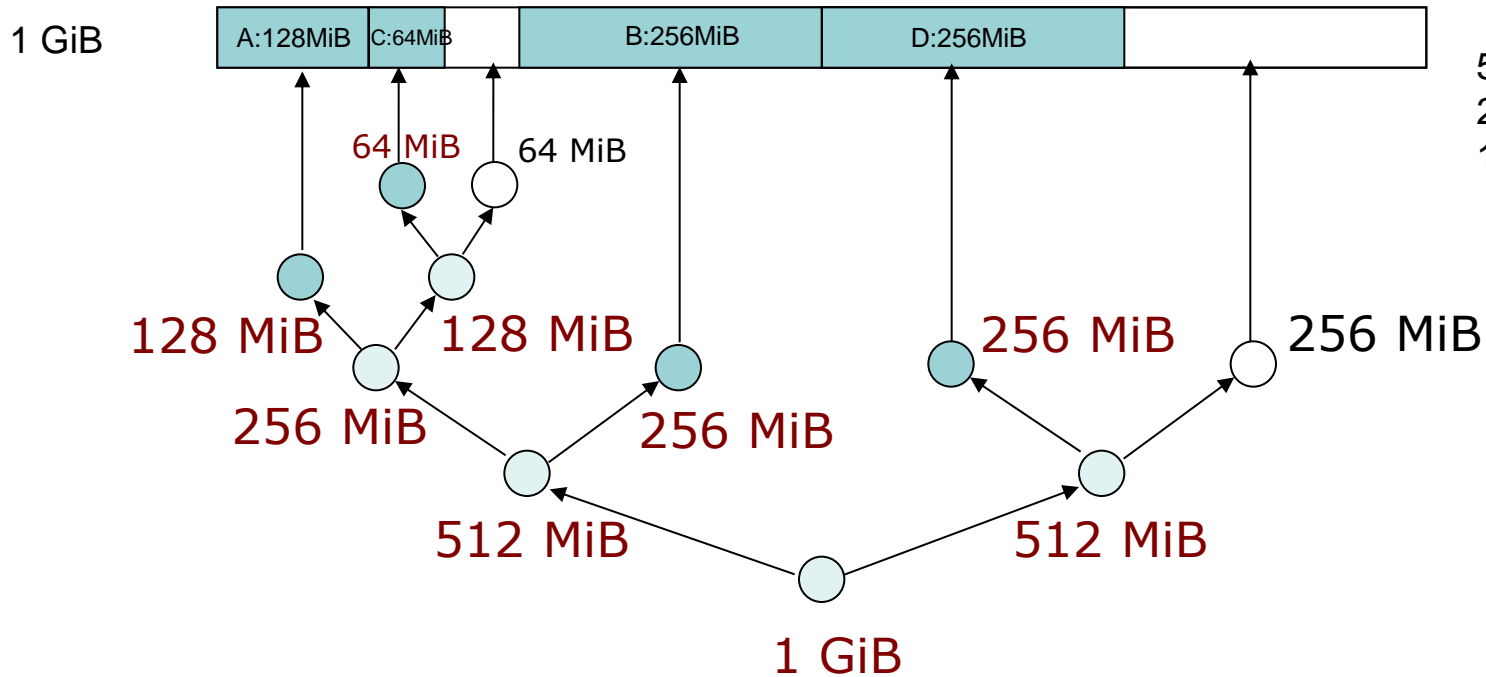
1 GiB: 0
512 MiB: 1
256 MiB: 0
128 MiB: 0
64 MiB: 1

Es folgt Anforderung D: 256 MiB, d.h. Block der Größe 256 MiB

Buddy-System (12)

Nach Anforderung D: 256 MiB, d.h. Block der Größe 256 MiB.

Freie Blöcke:

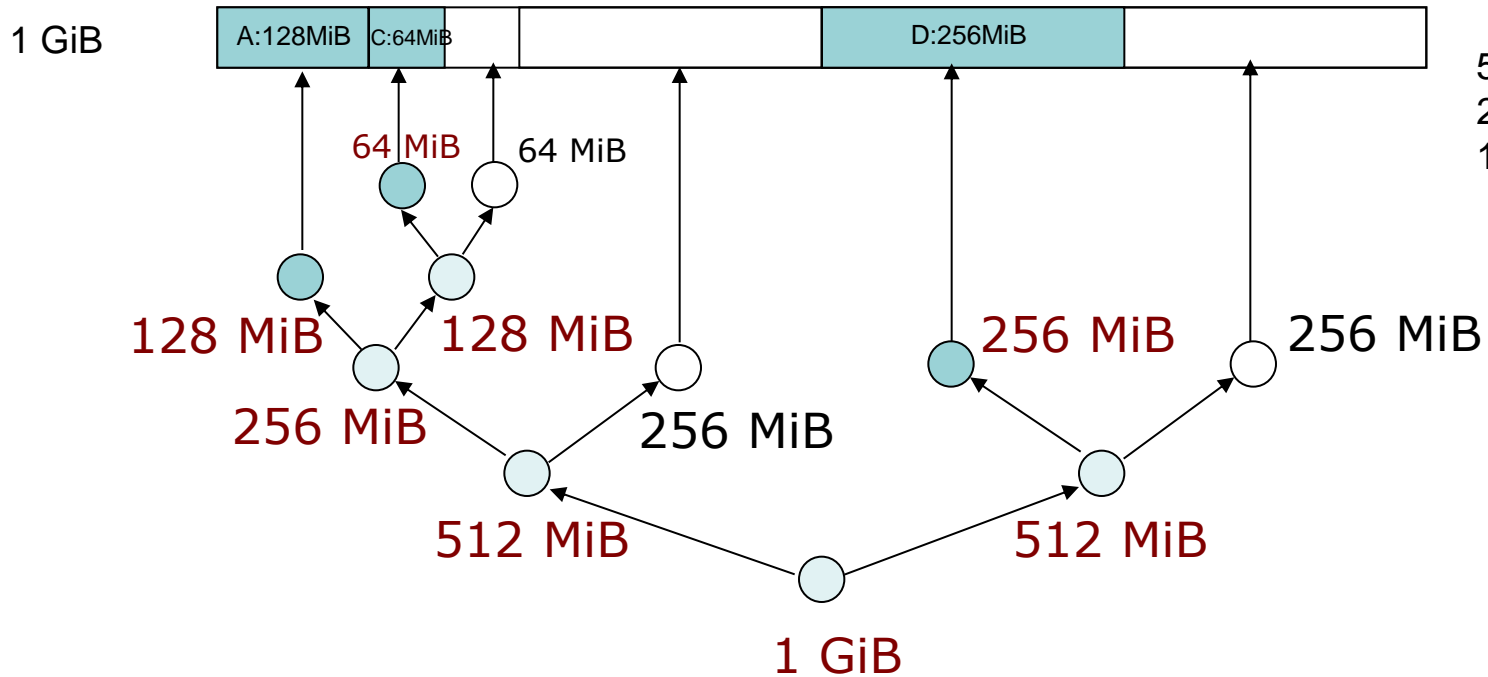


Es folgt Freigabe B

Buddy-System (13)

Nach Freigabe B:

Freie Blöcke:

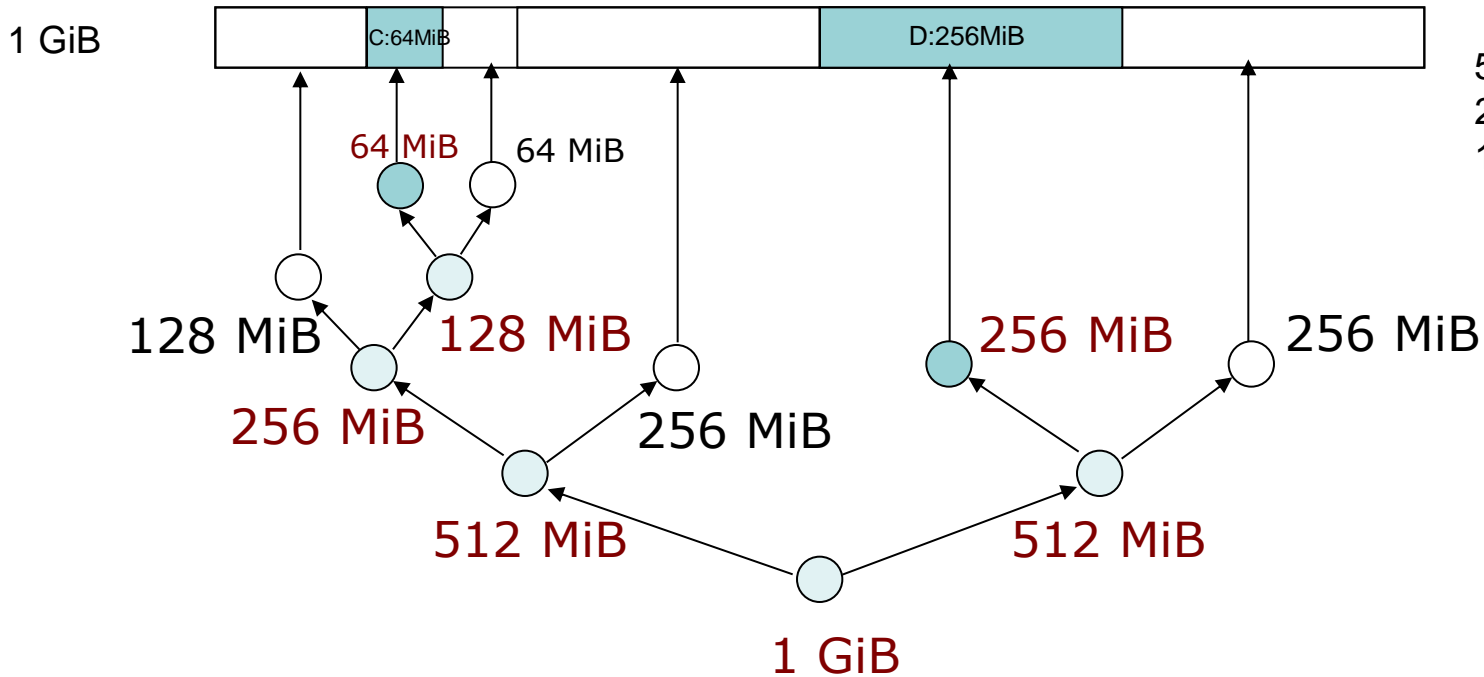


Es folgt Freigabe A

Buddy-System (14)

Nach Freigabe A:

Freie Blöcke:

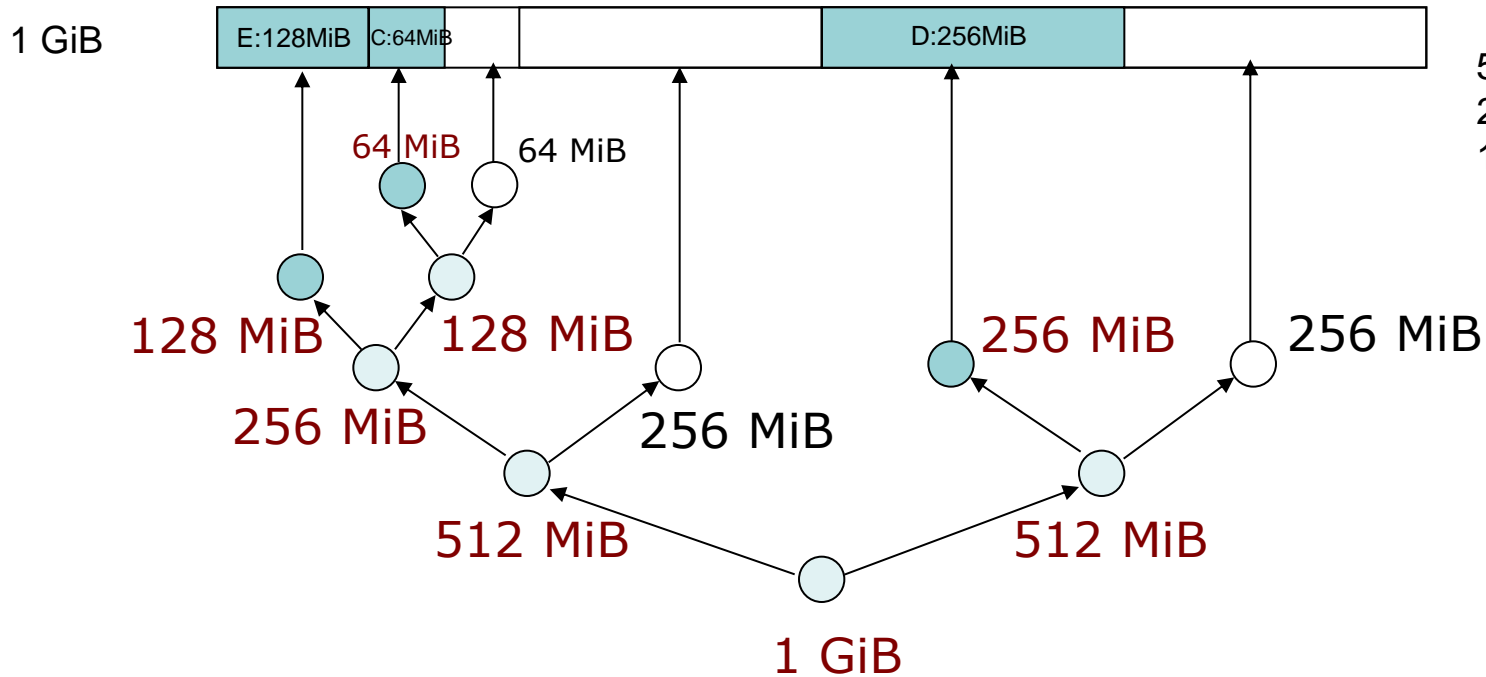


Es folgt Anforderung E: 75 MiB, d.h. Block der Größe 128 MiB

Buddy-System (15)

Nach Anforderung E: 75 MiB, d.h. Block der Größe 128 MiB:

Freie Blöcke:



1 GiB: 0
512 MiB: 0
256 MiB: 2
128 MiB: 0
64 MiB: 1

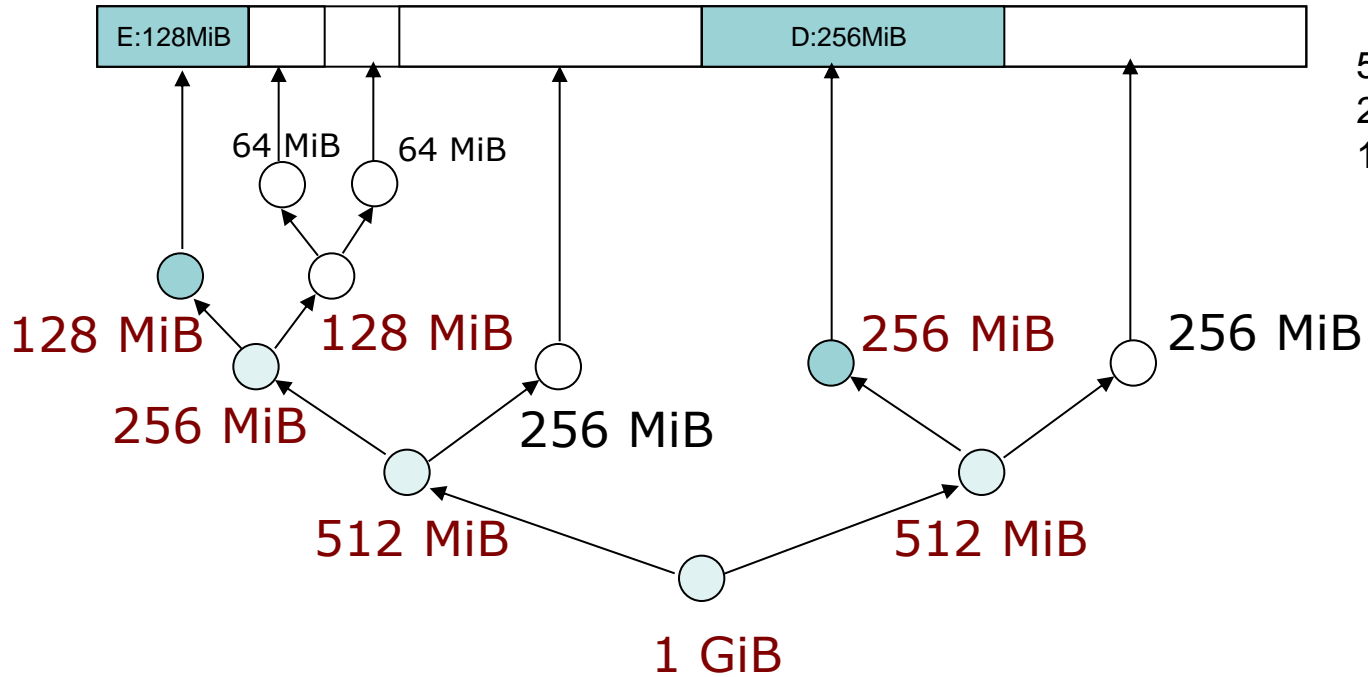
Es folgt Freigabe C

Buddy-System (16)

Bei Freigabe C:

Freie Blöcke:

1 GiB



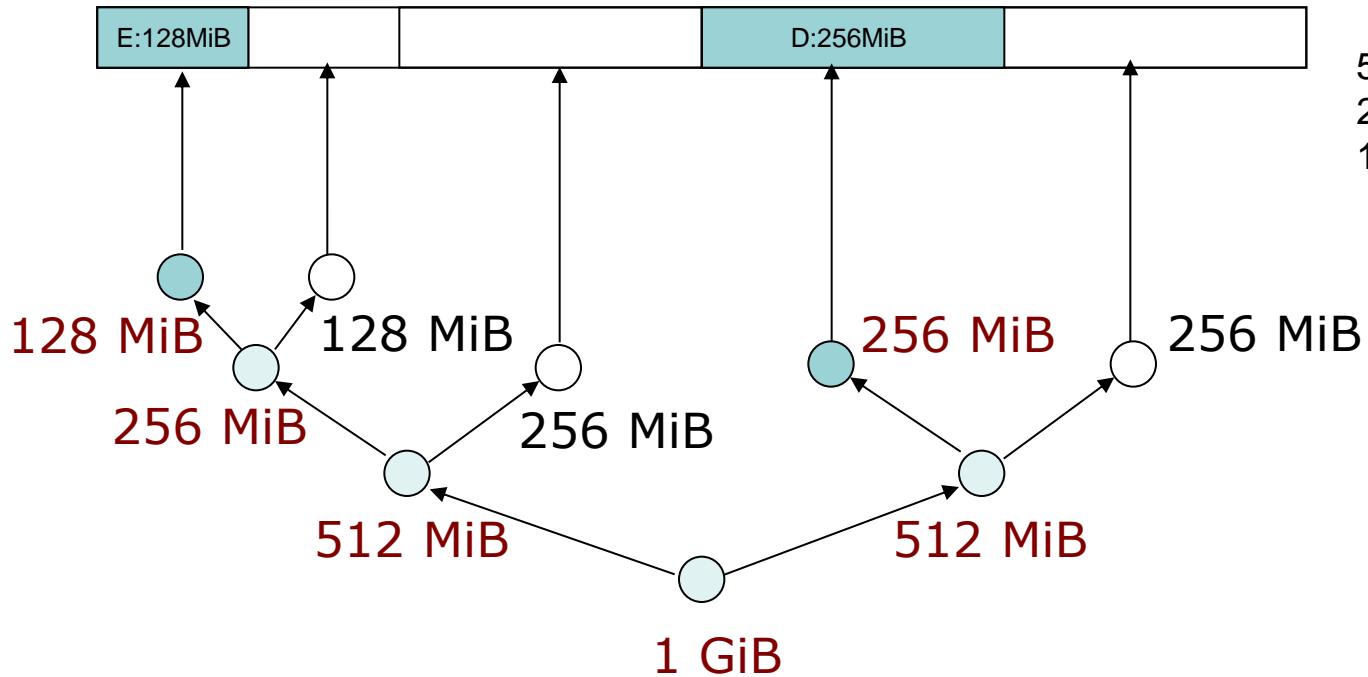
1 GiB: 0
512 MiB: 0
256 MiB: 2
128 MiB: 0
64 MiB: 2

Buddy-System (17)

Nach Freigabe C:

Freie Blöcke:

1 GiB



1 GiB: 0
512 MiB: 0
256 MiB: 2
128 MiB: 1
64 MiB: 0

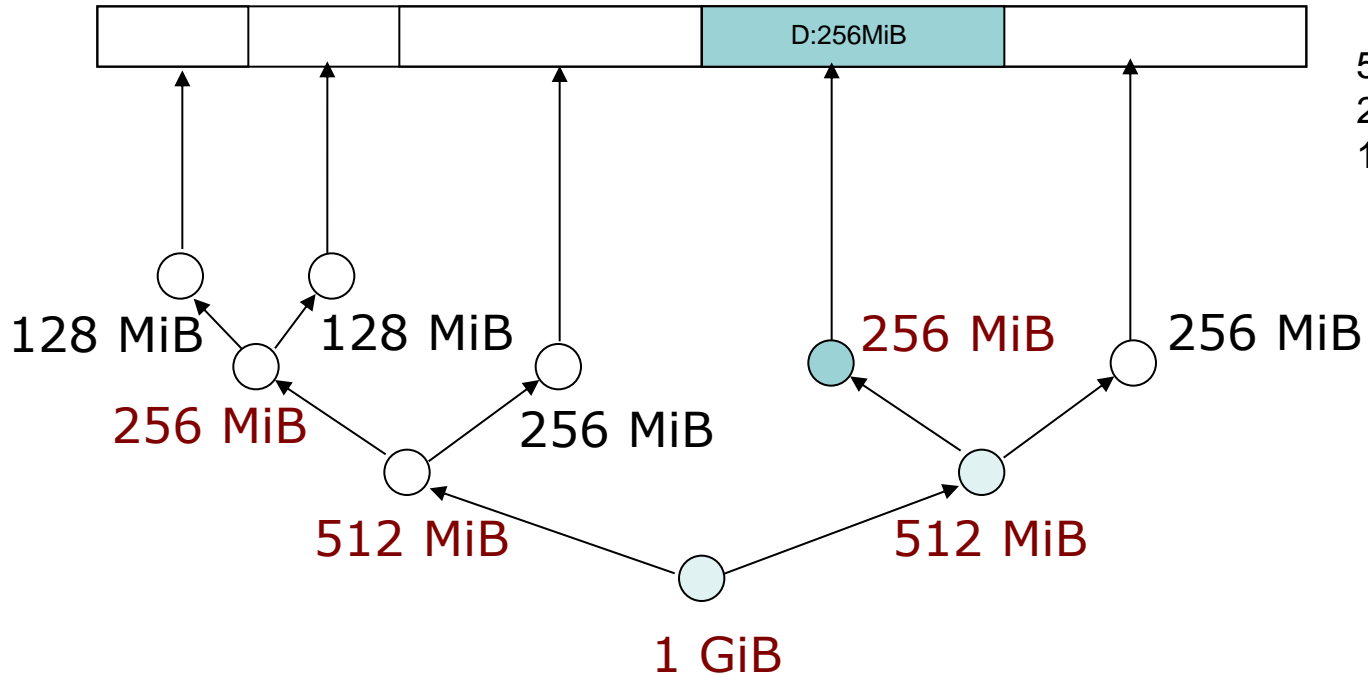
Es folgt Freigabe E

Buddy-System (18)

Bei Freigabe E:

Freie Blöcke:

1 GiB



1 GiB: 0
512 MiB: 0
256 MiB: 2
128 MiB: 2
64 MiB: 0

Buddy-System (19)

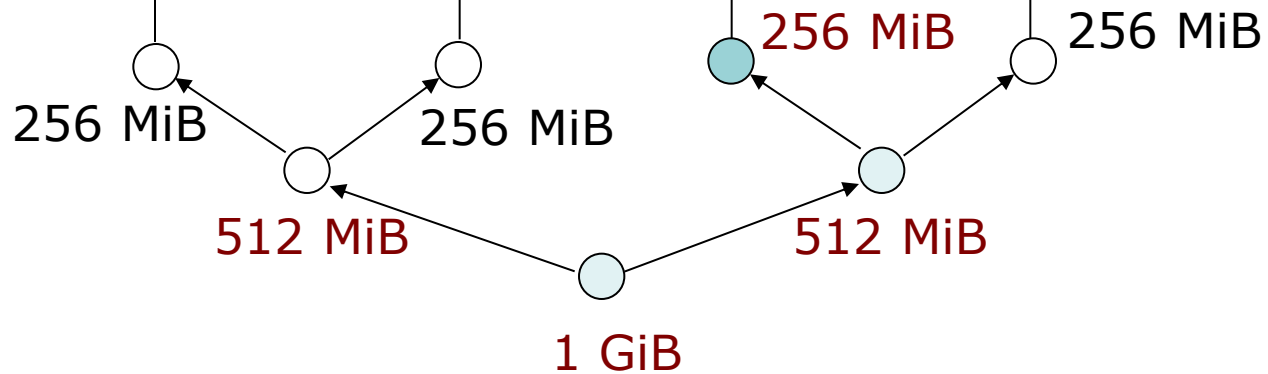
Bei Freigabe E:

Freie Blöcke:

1 GiB



1 GiB: 0
512 MiB: 0
256 MiB: 3
128 MiB: 0
64 MiB: 0



Buddy-System (20)

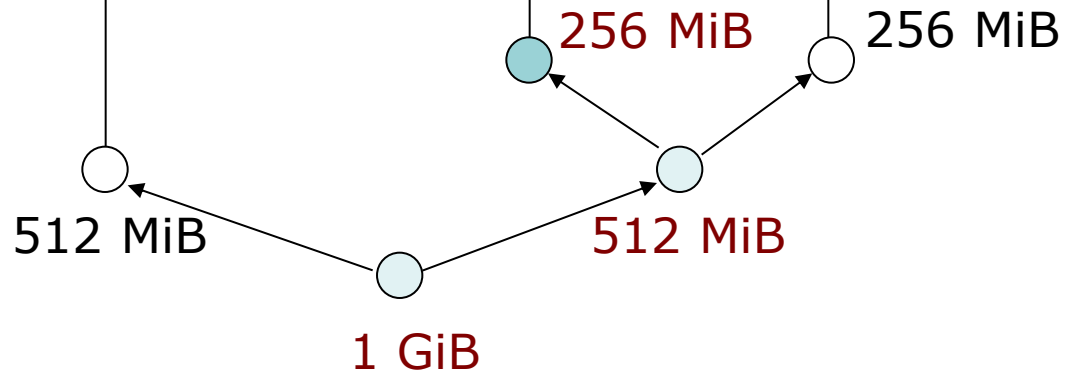
Nach Freigabe E:

1 GiB



Freie Blöcke:

1 GiB: 0
512 MiB: 1
256 MiB: 1
128 MiB: 0
64 MiB: 0



Es folgt Freigabe D

Buddy-System (21)

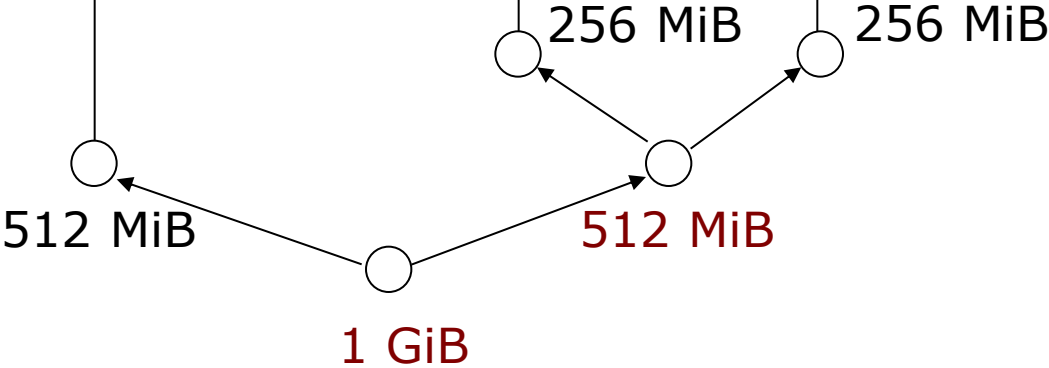
Bei Freigabe D:

Freie Blöcke:

1 GiB



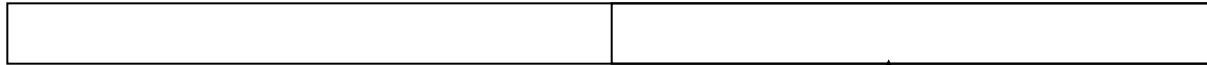
- 1 GiB: 0
- 512 MiB: 1
- 256 MiB: 2
- 128 MiB: 0
- 64 MiB: 0



Buddy-System (22)

Bei Freigabe D:

1 GiB



512 MiB

1 GiB

512 MiB

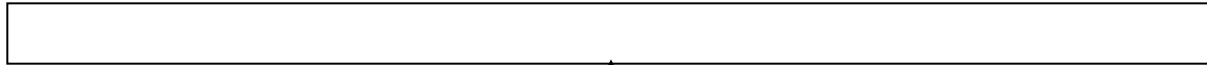
Freie Blöcke:

1 GiB: 0
512 MiB: 2
256 MiB: 0
128 MiB: 0
64 MiB: 0

Buddy-System (23)

Nach Freigabe D:

1 GiB



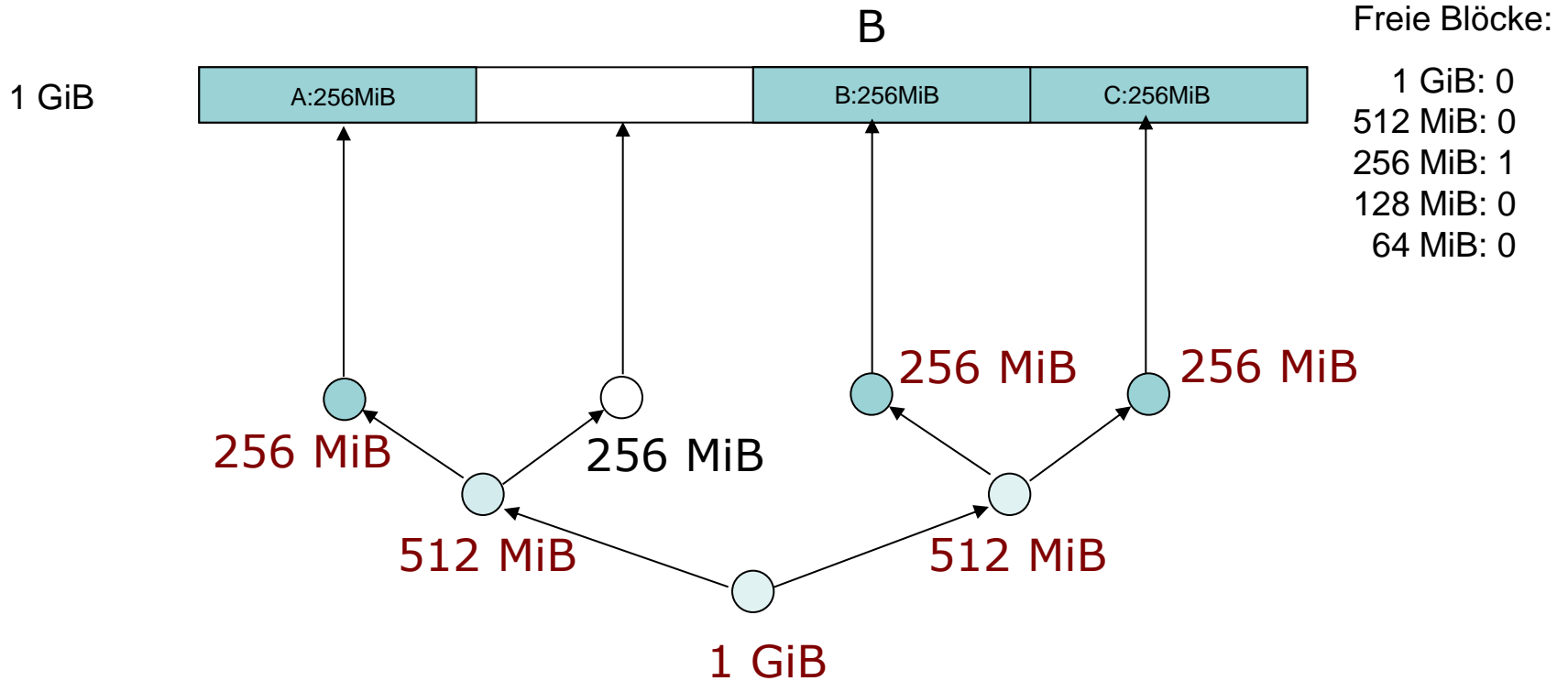
1 GiB

Freie Blöcke:

1 GiB: 1
512 MiB: 0
256 MiB: 0
128 MiB: 0
64 MiB: 0

Gesamter Speicher wieder als 1 Block verfügbar

Buddy-System (24)



Was passiert bei Freigabe von B?

Buddy-System (25)

- Effiziente Suche nach freiem Block
- Dynamische Anzahl nicht-ausgelagerter Prozesse
- Beschränkte interne Fragmentierung: I.d.R. kleiner als die halbe Größe des gewählten freien Blockes
- Wenig externe Fragmentierung, schnelle Zusammenfassung von freien Blöcken

Grundlegende Methoden der Speicherverwaltung

Partitionierung

- Speicheraufteilung zwischen verschiedenen Prozessen (Partitionierung mit festen Grenzen)

Paging

- Einfaches Paging / kombiniert mit Konzept des virtuellen Speichers

Segmentierung

- Einfache Segmentierung / kombiniert mit Konzept des virtuellen Speichers

Einfaches Paging (1)

- Zunächst wie bisher: Prozesse sind entweder ganz im Speicher oder komplett ausgelagert
- Prozessen werden Speicherbereiche zugeordnet, die **nicht** notwendigerweise zusammenhängend sind
- Hauptspeicher ist aufgeteilt in viele **gleichgroße Seitenrahmen**
- Prozesse sind aufgeteilt in **Seiten derselben Größe**

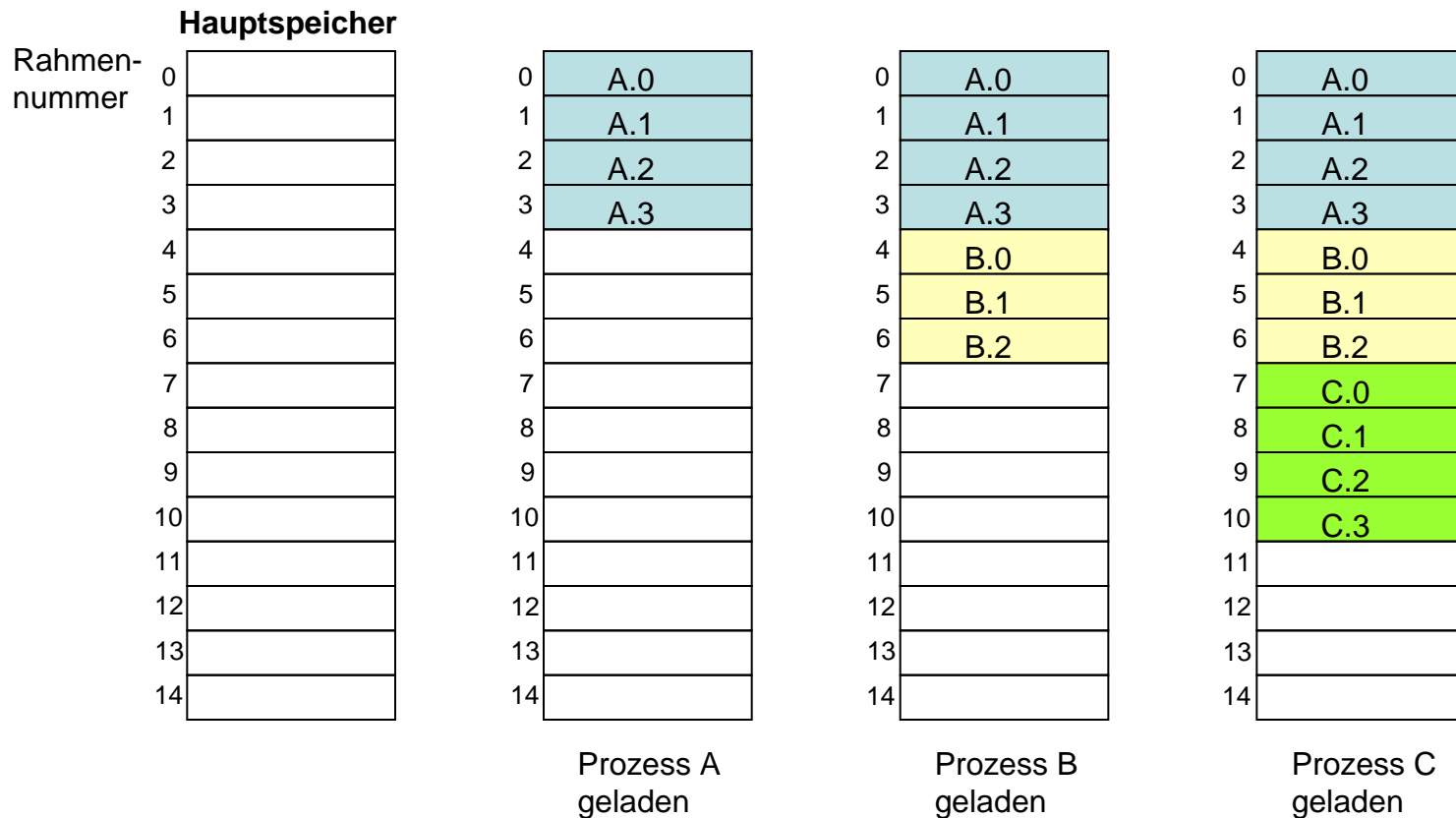
Einfaches Paging (2)

- Betriebssystem verwaltet **Seitentabelle** für jeden Prozess
- Zuordnung von Seiten zu Seitenrahmen bei der Ausführung von Prozessen
- Innerhalb Programm: Logische Adresse der Form „**Seitennummer, Offset**“
- Durch Seitentabelle: Übersetzung der logischen Adressen in physikalische Adressen „**Rahmennummer, Offset**“

Einfaches Paging (3)

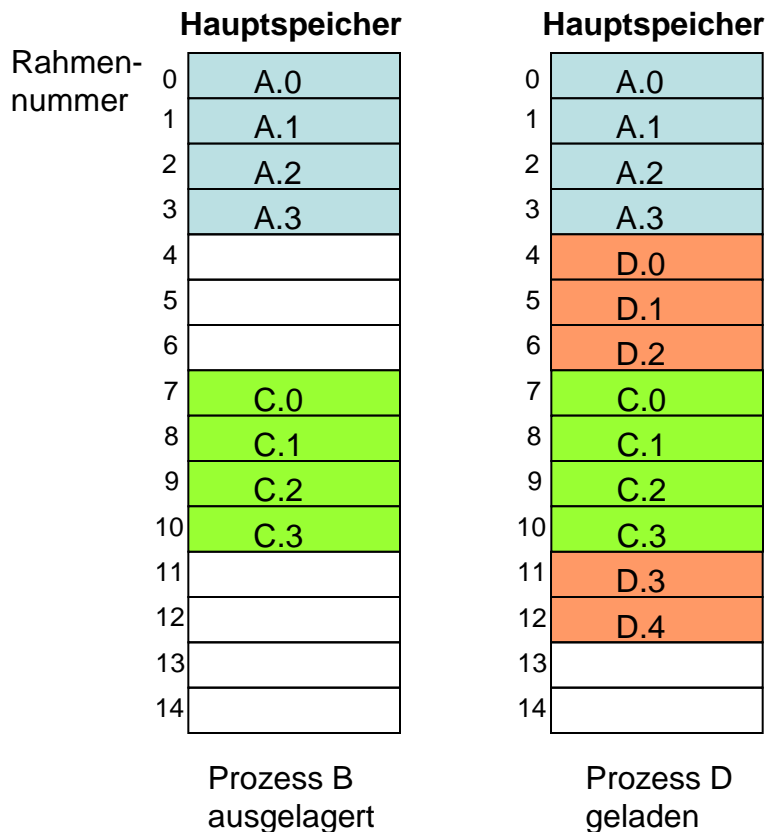
- Prozessorhardware übersetzt logische Adresse in physikalische Adresse
- Interne Fragmentierung nur bei letzter Seite eines Prozesses
- Keine externe Fragmentierung

Einfaches Paging (4)

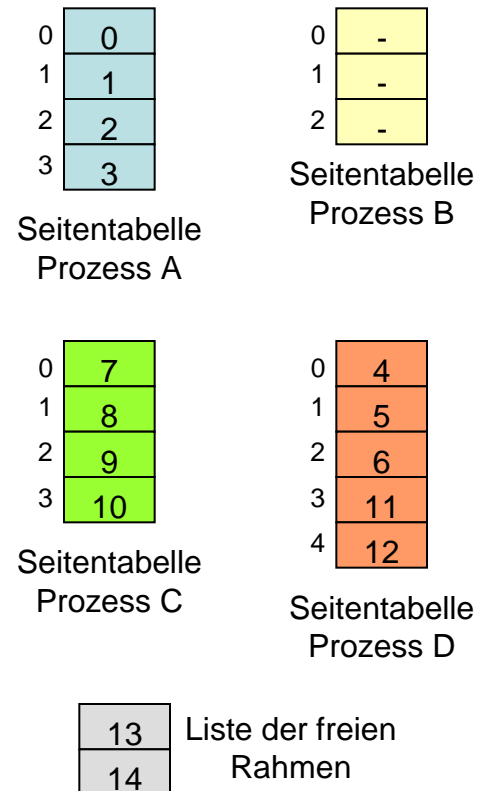


Prozess D
mit 5 Seiten
soll jetzt
geladen
werden!

Einfaches Paging (5)



Datenstrukturen zum aktuellen Zeitpunkt:



Einfaches Paging (6)

- Einfaches Paging ähnlich zum Konzept des statischen Partitionierens
- Aber: Beim Paging sind die Partitionen relativ klein
- Programm kann mehrere Partitionen /Rahmen belegen, die nicht aneinander angrenzen

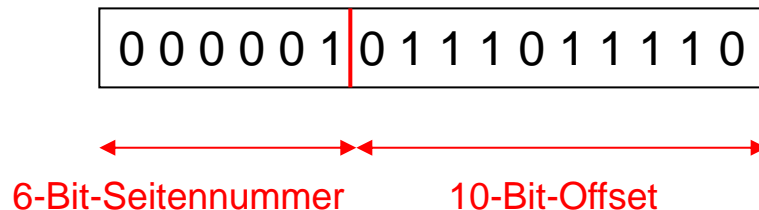
Einfaches Paging (7)

Berechnung von physikalischen Adressen aus logischen Adressen:

- Seitengröße (und Rahmengröße) ist eine Zweierpotenz
- Logische Adresse im Programm besteht aus Seitennummer und Offset
- Physikalische Adresse besteht aus Rahmennummer und Offset

Einfaches Paging (8)

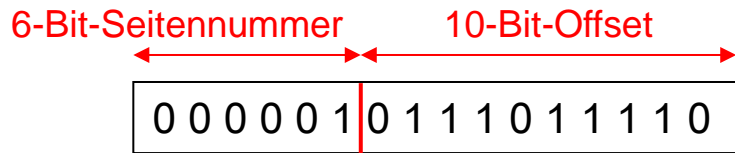
- Beispiel: Logische Adresse der Länge 16 Bit
- Seitengröße 1 KiB = $2^{10} = 1024$ Bytes
- Offset-Feld von 10 Bit wird benötigt, um alle Bytes referenzieren zu können



- Der Prozess kann bis zu $2^6=64$ verschiedene Seiten haben, die über die Seitentabelle des Prozesses auf Seitenrahmen im Hauptspeicher abgebildet werden

Einfaches Paging (9)

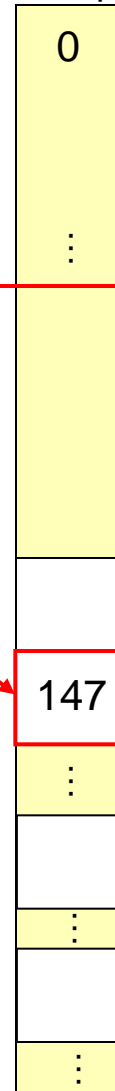
alle Seitenrahmen
des Arbeitsspeichers



Seitentabelle
des Prozesses

0	10010010
1	10010011
2	11011001
3	11111010

Seitenrahmen Nr. 147
(= $\langle 10010011 \rangle$)



Speicherzelle Nr. 478
(= $\langle 0111011110 \rangle$)
innerhalb des
Seitenrahmens

Physikalische Adresse

10010011	0111011110
----------	------------

Rahmennr. * Seitengröße + Offset

Einfaches Paging (10)

- Hauptspeicher wird in viele kleine Rahmen gleicher Größe unterteilt
- Jeder Prozess wird in Seiten geteilt, deren Größe der der Rahmen entspricht
- Seitentabelle enthält Zuordnung von Prozessseiten an Seitenrahmen des Speichers
- Interne Fragmentierung nur bei letzter Seite eines Prozesses

Einfaches Paging (11)

Entfernen eines Prozesses aus dem Speicher:

- Seitentabelle enthält Information, welche Seitenrahmen dem Prozess gehören
- Füge diese Rahmen zur Liste der freien Rahmen hinzu
- Keine zusätzlichen Datenstrukturen des Betriebssystems benötigt

Paging mit virtuellem Speicher (1)

Grundidee:

- Lagere **Teile** von Prozessen ein- bzw. aus anstelle kompletter Prozesse
- Programm kann auch weiter ausgeführt werden, auch wenn **nur die aktuell benötigten** Informationen (Code und Daten) im Speicher sind
- Bei Zugriff auf aktuell ausgelagerte Informationen: Nachladen von Seiten

Paging mit virtuellem Speicher (2)

- Hauptspeicher = realer Speicher
- Hauptspeicher + Hintergrundspeicher = virtueller Speicher
- Vorteile:
 - Platz für mehr bereite Prozesse
 - Tatsächlicher Speicherplatzbedarf eines Prozesses muss nicht im Voraus feststehen
 - Adressraum eines Prozesses kann größer sein als verfügbarer Hauptspeicher

Paging mit virtuellem Speicher (3)

Nachteile:

- Nachladen von Seiten
- Notwendiges Auslagern von anderen Seiten
- System wird langsamer

Lokalität (1)

Effizienz von virtuellem Speicher

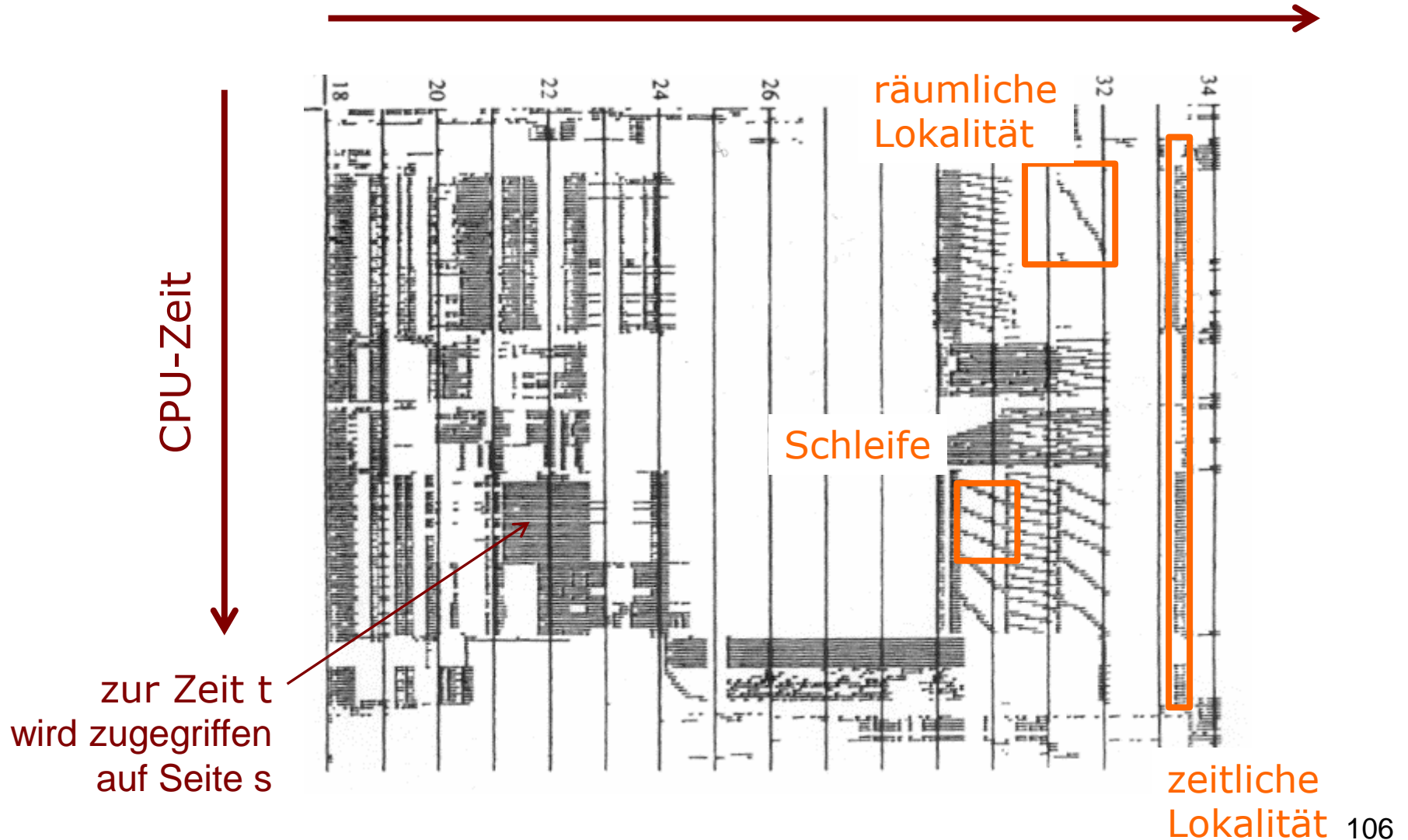
- Typischerweise räumliche und zeitliche Lokalität von Programmen
- **Zeitliche Lokalität:** Nach Zugriff auf eine Speicherzelle ist die Wahrscheinlichkeit hoch, dass in naher Zukunft noch einmal darauf zugegriffen wird
- **Räumliche Lokalität:** Nach Zugriff auf eine bestimmte Speicherzelle gehen die Zugriffe in naher Zukunft auf Speicheradressen in der Nähe

Lokalität (2)

- Die Abarbeitung während kürzerer Zeit bewegt sich häufig in engen Adressbereichen
- Zeitliche Lokalität:
 - Abarbeitung von **Schleifen**
 - In zeitlich engem Abstand Zugriff auf **gleiche Daten**
- Räumliche Lokalität:
 - Sequentielle Abarbeitung von Programmen: Zugriffe auf **benachbarte** Daten
 - Lage von zusammenhängenden Daten

Lokalität (3)

Seitennummern eines Prozesses



Lokalität (4)

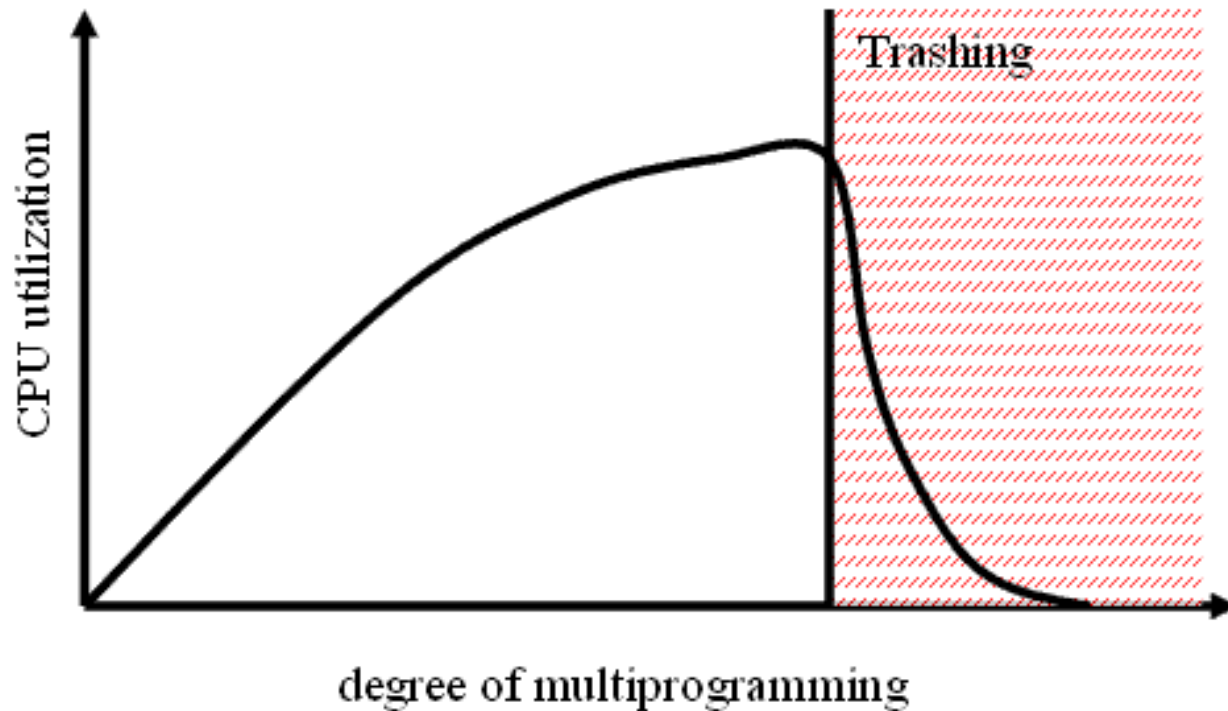
- Paging mit virtuellem Speicher ist nur dann effizient, wenn Lokalität gegeben ist
- Ansonsten Gefahr von „Thrashing“:
 - Ständiges Aus- und Einlagern von Seiten zwischen Hauptspeicher und Festplatte
 - Der Prozessor ist mehr mit Ein- und Auslagern anstatt mit Ausführen von Befehlen beschäftigt

Thrashing (1)

Mögliche Gründe für Thrashing:

1. Zu wenig Speicher
2. Zu viele Prozesse
3. Zu viele speicherintensive Prozesse
4. Schlechte Ausnutzung von Lokalität
5. ...

Thrashing (2)



Um Thrashing zu vermeiden versucht das Betriebssystem vorherzusagen, welche Seiten in naher Zukunft nicht benötigt werden

Thrashing (3)

Es hängt auch vom Programmierer ab:

```
int m[256][256];  
for (i=0; i<256; i++)  
    for (j=0; j< 256; j++)  
        m[i][j] = 0;
```

im Vergleich zu:

```
int m[256][256];  
for (j=0; j<256; j++)  
    for (i=0; i< 256; i++)  
        m[i][j] = 0;
```

Während die erste Version die Lokalität ausnutzt, hat die zweite eine deutlich schlechtere Effizienz, weil sie ständig zwischen weiter entfernten Speicherbereichen springt.