




Vorlesung Betriebssysteme

Abschnitt 4 – Speicherverwaltung

Inhalt: Speichersystem / Adressräume / Realer Speicher / Virtueller Speicher

M.Sc. Patrick Eberle

Symbol	Bedeutung
	Übung
	Beispiel
	Kommentar
	Definition



Kapitel X

Speichersystem

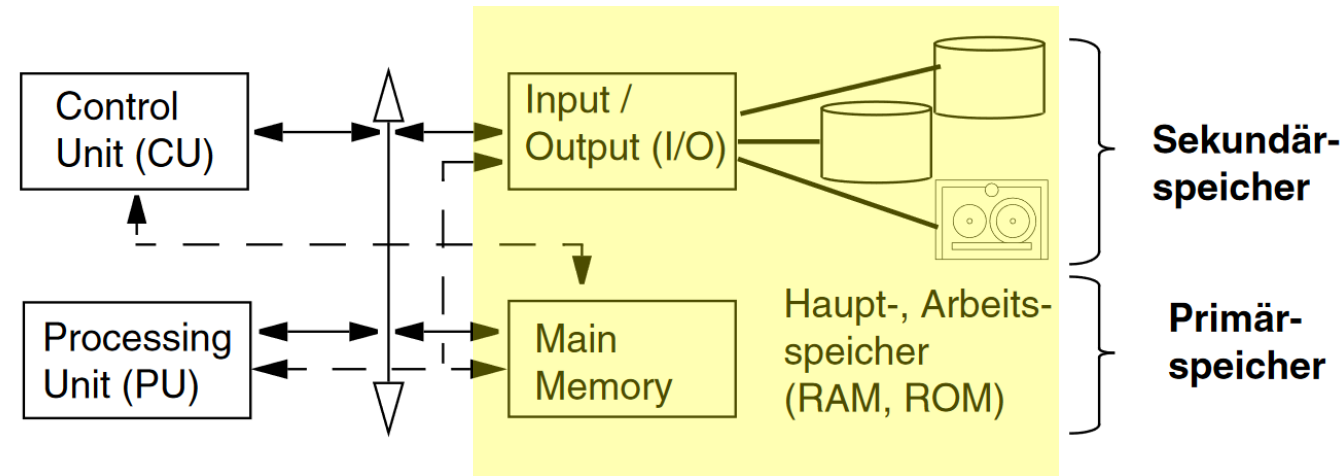
- Das Speichersystem eines Betriebssystems ist vergleichbar mit der Lagerhaltung in der Logistik:
 - Aufgabe besteht darin, Objekte so schnell wie möglich aus dem Lagerhaushalt an einen Platz zu bringen, an dem Mehrwert geschaffen wird: also vom Lager an den Verbraucher bzw. Hauptspeicher in den Cache bzw. die Register zur Verarbeitung auf der CPU
 - Objekte werden zunächst an einem zentralen Ort mit hoher Kapazität gelagert (Zentrallager / Festplatte), bevor sie in lokale Nähe zum eigentlichen Wirkungsort gebracht werden (Dezentrallager / Hauptspeicher, CPU-Cache und Register)
 - Bestimmte Parameter und Algorithmen entscheiden darüber, welche Objekte besser im Zentrallager / Festplatte gehalten werden oder dezentral / Hauptspeicher, CPU-Cache, Register gehalten werden

- In der Computertechnik: Unterscheidung zwischen Primär- und Sekundärspeicher
- Primärspeicher bzw. Hauptspeicher dient kurzzeitiger Ablage von Daten während des Betriebs eines Rechensystems; man spricht von *flüchtigem* oder *transientem Speicher*
- Sekundärspeicher dient der langfristigen Speicherung von Daten, welche neben dem aktiven Betrieb auch im inaktiven Betrieb eines Rechensystems erhalten bleiben; man spricht von *nicht-flüchtigem* oder *persistentem Speicher*
- Im Rahmen dieses Abschnittes: Konzentration auf die Organisation des Primärspeichers

- Zur Einordnung des Speichersystems in der Rechnerarchitektur:

Betrachtung der Von-Neumann-Architektur

- Hauptspeicher besitzt direkte Verbindung zur CPU und ist direkt adressierbar
- Sekundärspeicher werden als Peripheriegeräte an den Block Input / Output angeschlossen; hierbei indirekte Adressierung



Quelle: [BS15]

- Unterscheidung Primärspeicher und Sekundärspeicher:

Merkmal	Primärspeicher	Sekundärspeicher
Adressierung	Direkt	Indirekt
Datenorganisation	Physisch	Logisch
Speicherart	Flüchtig / Transient / Festwertspeicher	Dauerhaft / Persistent
Realisierung	RAM / ROM	Plattenspeicher / Halbleiterspeicher

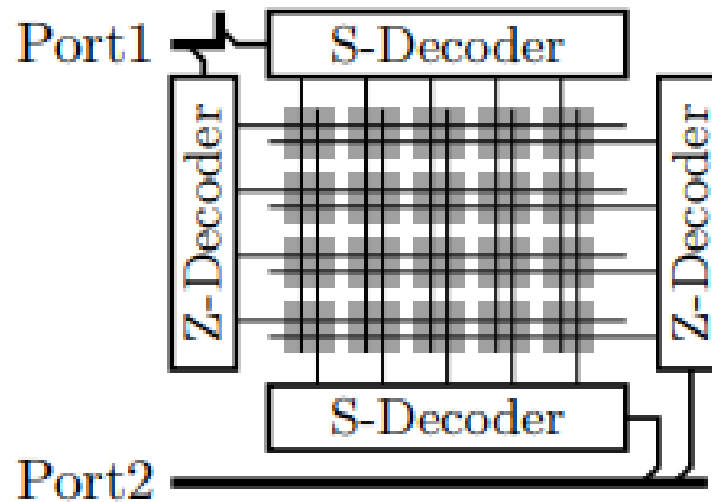
- Je nach Anwendungsfall werden zur Realisierung der Speicherverwaltung unterschiedliche Prinzipien angewendet
- Daher im Folgenden: Vorstellung sechs grundlegender Speicherprinzipien

▪ Speicherprinzip 1: Direkt adressierter Speicher / direkte Speicherverwaltung

- Über eine Adresse wird die gewünschte Speicherstelle direkt angesprochen
- Zugriff auf Speicherstelle findet unmittelbar und gleichwertig statt, daher spricht man auch von *Wahlfreiem Speicher*
- Einfach zu verwalten, keine MMU wie bei indirekter Speicherverwaltung nötig
- Typische Anwendung: Hauptspeicheradressierung

▪ Speicherprinzip 2: Mehrportspeicher

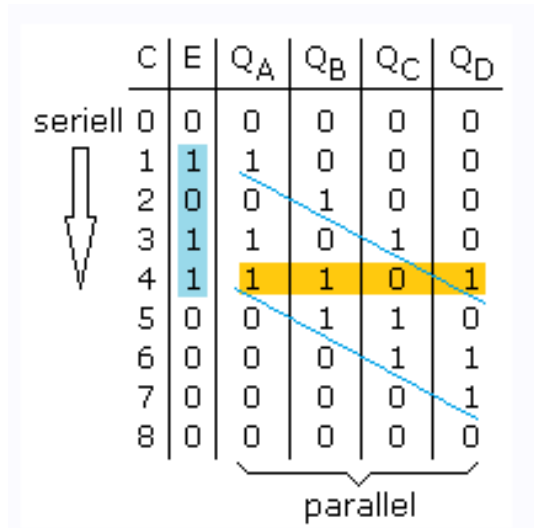
- Zugriff auf Speicher kann über mehrere Zugriffspfade abgewickelt werden
- Dadurch: Zwei aktive Hardware-Komponenten können zeitgleich Daten unterschiedlicher Adressen austauschen, ohne sich gegenseitig zu beeinflussen bzw. stören
z. B. CPU und Peripheriecontroller



Bildquelle: [T111]

- **Speicherprinzip 3: Schieberegisterspeicher**

- Bitmuster werden durch eine Kette von 1-Bit großen Speicherzellen in Fließband-Art verschoben
- Findet insbesondere nützliche Anwendung bei der Umwandlung serieller in parallele Daten, beispielsweise:
 - zwischen CPU und Netzwerk
 - bei einem seriellen Peripheriebus wie USB oder SATA



Quelle: <https://elektroniktutor.de/digitaltechnik/register.html>

▪ Speicherprinzip 4: FIFO-Speicher

- Arbeitet nach dem Senioritätsprinzip:

Es werden immer die Daten ausgelesen, die sich am längsten im Speicher befinden

- Typische Anwendung: Zwischenpufferung von Daten

▪ Speicherprinzip 5: Stapelspeicher (Stack)

- Arbeitet nach dem LIFO-Prinzip:
Zuletzt gespeicherte Daten werden als erstes wieder ausgelesen
- Typische Anwendung: Stack-Speicher eines Rechners

▪ Speicherprinzip 6: Assoziativspeicher

- Auch bezeichnet als *Content Adressable Memory (CAM)*

- Inhaltsadressierter Speicher:

Mit einer Teilinformation eines Eintrags wird gesamter Informationseintrag abgefragt

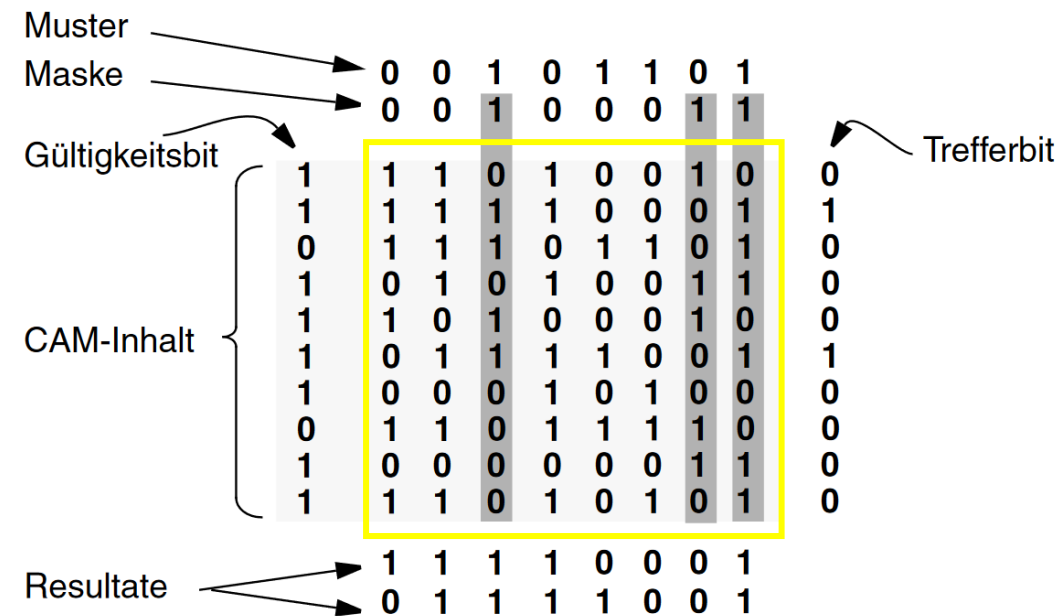
- Typische Anwendung:

Im Speichersystem an mehreren Orten von Relevanz, daher im Folgenden nähere Betrachtung

- Auch als *Content Adressable Memory (CAM)* bezeichnet
- Assoziativspeicher verwendet beim Lesezugriff zur Adressierung keine Adressen in Form von Nummern, sondern Teilinformationen des jeweiligen Informationseintrags
- Zur Adressierung verwendete Teilinformation kann in 0 ... n Informationseinträgen vorkommen, sodass Lesezugriff entsprechende Anzahl Ergebnisse liefert
- Beim Schreibzugriff: Auswahl einer unbelegten Speicherstelle kann mittels Adressen abgewickelt werden

- Funktionsweise

- Assoziativspeicher enthält 10 Worte (gelber Kasten):
Pro Wort eine Zeile
- Eingangsinformation für den Lesezugriff wird als „Muster“ bezeichnet
- „Maske“ identifiziert zu benutzende Teilinformationen:
Bit gesetzt = Teilinformation ist zu verwenden
- „Gültigkeitsbit“ gibt Gültigkeit pro Eintrag an:
beim Lesezugriff nur Berücksichtigung gültiger Einträge
- Folgende Bedingung muss gelten, damit ein Wort in das Resultat übernommen wird:
 $(\text{Muster} \ \& \ \text{Maske}) == (\text{Maske} \ \& \ \text{Wort})$



Quelle: [BS15]

▪ Prinzip des Assoziativspeichers bietet sich z. B. an für:

- Datenbankanfragen der Art:

„Ich habe einen Preis, liefere mir zugehörige Artikel“

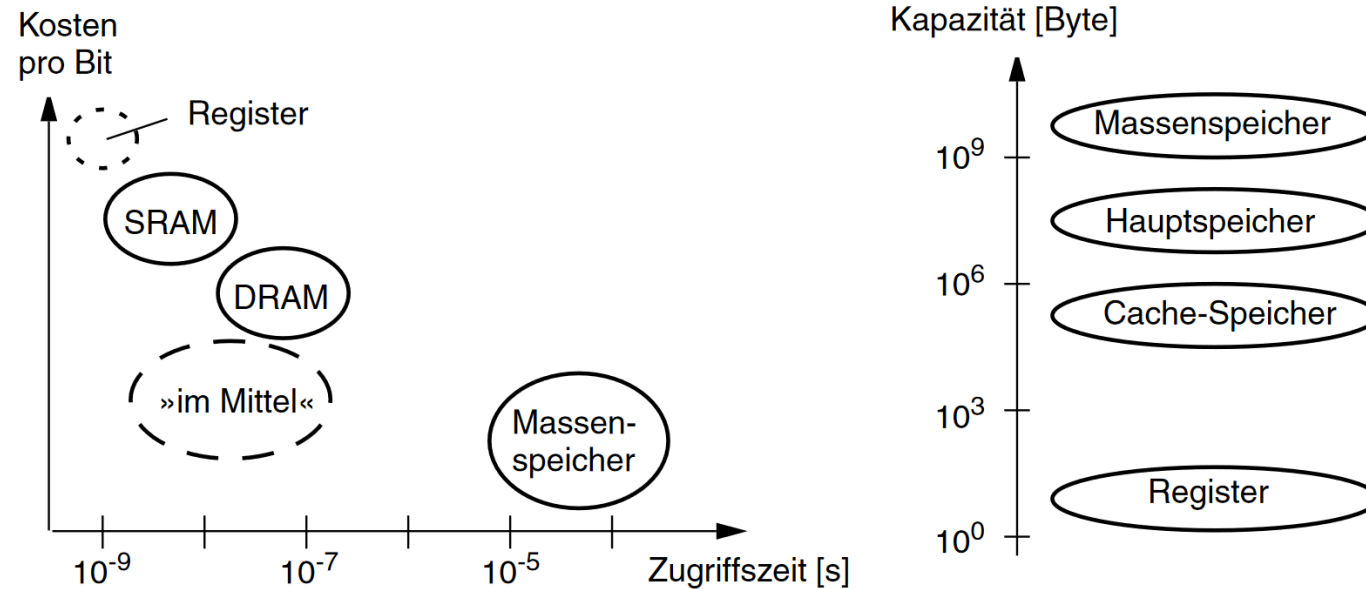
- Anwendung im Bereich der Cache-Hardware
- Adresstransformation

▪ Nachteile des Assoziativspeichers:

- Mehrfachtreffer möglich
- Problematische Verwaltung freier Speicherstellen
- Aufwendige Hardware bei Direktzugriffs-CAM

- Ausgehend von unterschiedlichen Anforderungen benutzen Rechner eine hierarchische Speicheranordnung
- Dadurch: Ermöglichung einer vertretbaren Speicheranordnung hinsichtlich des Trade-offs zwischen minimaler Zugriffszeit und den Kosten pro gespeichertes Bit
- Dabei gilt nach wie vor:
 - Teuerster und schnellster Speicher sind prozessorinterne Register
 - Billigster und langsamster Speicher ist Massenspeicher
- Zwischen Register und Massenspeicher liegt Hauptspeicher, der als RAM realisiert und über kleinen Pufferspeicher (Cache memory) an CPU gekoppelt ist
- Anzahl der Hierarchiestufen divergent und abhängig vom Anwendungsgebiet:
Eingebettetes Kleinsystem vs. voll ausgestatteter Arbeitsplatzrechner

- Herausforderung und Zielsetzung bei der Speicher-Hierarchisierung:
Mit geringstmöglichen Kosten im Mittel an die Zugriffszeit des schnellsten Speichers heranzukommen



Quelle: [BS15]

- Im Allgemeinen ist Optimierungsziel der Speicher-Hierarchisierung durch kombinierten Hardware-Einsatz bedingt erreichbar, dazu zählt z. B.:
 - Auf Hardware-Ebene:
 - Adresstransformation
 - Cache-Speicher
 - Auf Software-Ebene:
 - Paging
 - Swapping

- Speicherhierarchie ist transparent gegenüber dem Anwender / Programmierer:
 - Anwender sieht einen großen Adressraum, ist sich jedoch der dahinterstehenden Hierarchie aus verschiedenen Speichern nicht bewusst
 - Speicherverwaltung in Programmen erfolgt unabhängig von der Hierarchie
 - Datentransfer zwischen den Hierarchiestufen erfolgt automatisiert im Hintergrund mit Hilfe von Hardware und des Betriebssystems
 - Dabei: die im Zuge eines Speicherzugriffs langsamste Hierarchiestufe bestimmt die Speicherzugriffszeit der CPU (Vgl. auch: Memory Gap aus VL-Abschnitt Grundlagen der Betriebssysteme)
 - Deshalb: Ohne Ausnutzung des sogenannten *Lokalitätseffekts* ist Speicherhierarchie wenig gewinnbringend

▪ Lokalitätseffekt:

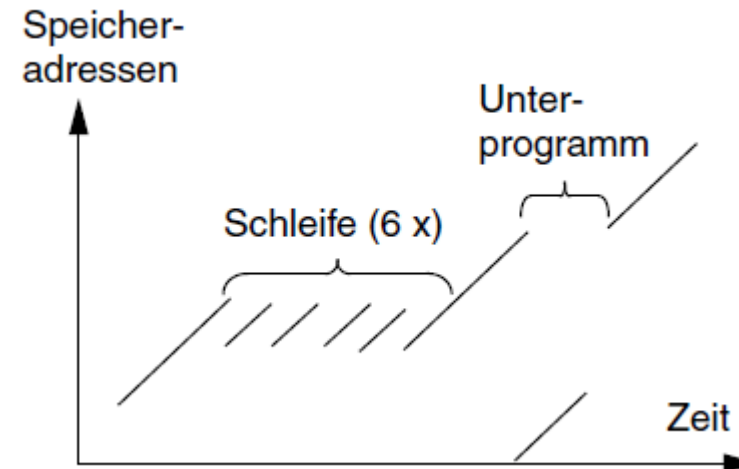
- Begründet die Idee der Speicherhierarchisierung anhand des nachfolgend beschriebenen Sachverhalts
- Dazu folgende Definitionen:
 - Es existiert ein sogenannter **Arbeitsbereich (Working Set) W** :
Menge von Speicheradressen, auf denen Zugriffe aufgezeichnet / beobachtet werden
 - Betrachtung eines Zeitraums $t-T$ und t
 - Dann gilt:
Der Arbeitsbereich $W(t-T, t)$ bleibt für größere Zeiträume unverändert
- Diesen Effekt bezeichnet man auch als *Lokalität der Referenzierung*
- Gewinnbringend kann der Effekt eingesetzt werden, wenn der Arbeitsbereich in einer möglichst schnellen Speicherstufe gehalten werden kann

- Bei der Betrachtung des Lokalitätseffekts: Unterscheidung zwischen räumlicher und zeitlicher Lokalität

- Bei der **räumlichen Lokalität** gilt:

Wird auf eine Speicheradresse zugegriffen, dann ist die Wahrscheinlichkeit hoch dass der nachfolgende Zugriff in der Nachbarschaft erfolgt.

→ Deshalb: Zusammenlegung räumlich benachbarter Speicherinhalte in Blöcke und Verschiebung in höhere Hierarchiestufe.

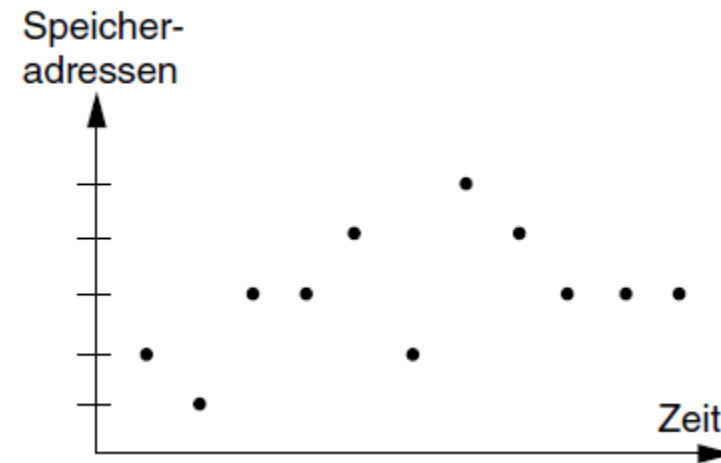


Quelle: [BS15]

- Bei der **zeitlichen Lokalität** gilt:

Wird auf eine Adresse zugegriffen, dann ist die Wahrscheinlichkeit hoch, dass zeitnah auf dieselbe Adresse nochmals zugegriffen wird.

→ Deshalb: Daten auf die zuletzt zugegriffen wurde, werden auf der schnellsten Hierarchiestufe gehalten.

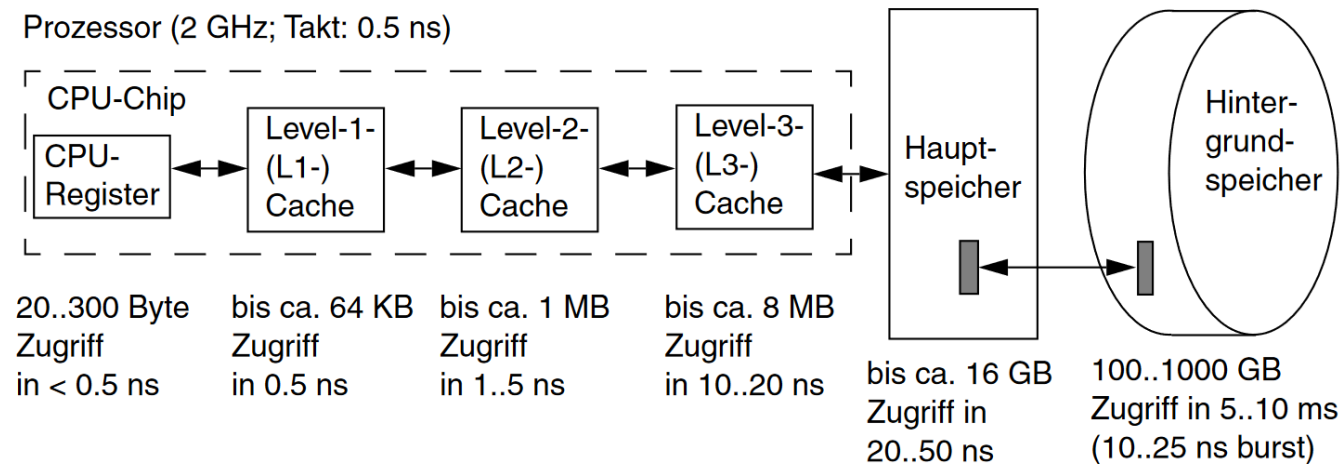


Quelle: [BS15]

- Während der Ausführung eines Programmes ändert sich der Arbeitsbereich zwar, wenn aber das Nachladen selten genug vonnöten ist, reicht dies aus um eine hohe Leistung zu erzielen
- Dementgegen sinkt die Leistung markant, sofern ein Arbeitsbereich nicht in einer begrenzt großen oberen Hierarchiestufe untergebracht werden kann
- Die Leistung sinkt ebenfalls, sofern der Arbeitsbereich stark frequentiert

- Weitere Komponente des Speichersystems besteht im Cache
- Anforderung an eine Art Zwischenspeicher ist begründet in dem Umstand, dass moderne Prozessoren wesentlich schneller arbeiten als Speicherbausteine (vgl. hierzu: Memory Gap aus VL-Abschnitt Grundlagen der Betriebssysteme)
- Aus diesem Grund: Einfügen einer weiteren Stufe in der Speicherhierarchie, welche als Cache-Speicher bezeichnet wird und gegenüber dem Programmierer transparent ist, also nicht in Erscheinung tritt
- Primärer Vorteil durch die Verwendung von Caching: Erzielung höherer Verarbeitungsleistungen durch prozessornahen Zwischenspeicher

- Cache ist zwischen CPU und Hauptspeicher platziert und kann mehrstufig realisiert werden
- In diesem Zusammenhang:
 - Level 1 Cache ist CPU-nächster Cache
 - Level 2 und Level 3 Cache sind CPU-fernere Caches



burst = Transfer adressmäßig aufeinander folgender Byte
(nach dem Zugriff auf erstes Byte)

Quelle: [BS15]

- Mehrstufige Cache-Speicher heute i. d. R. On-Chip, also prozessorintern realisiert
- Im Rahmen eines Betriebssystems: Einsatz des softwareseitigen Cachings in verschiedenen Anwendungen:
 - Disk Cache
 - Buffer Cache
 - File Cache
- Herausforderung beim Caching besteht im Aktuell-Halten des Cache-Inhaltes bei der Verwendung von verteilten Caches
 - Ziel ist es, Cache-Kohärenz sicherzustellen

- Cache-Kohärenz:
 - Wenn zwei oder mehr verschiedene Kontrolleinheiten auf gemeinsamen Cache zugreifen, kann der Effekt auftreten, dass auf unterschiedlichen, unsynchronisierten Inhalten gearbeitet wird:
 - Wenn mehrfache Kopien desselben Datensatzes gleichzeitig in verschiedenen Caches vorliegen
 - In Multiprozessorumgebungen
 - Wenn Rückschreibeverfahren (Write-Back) anstelle von Durchschreibeverfahren (Write-Through) verwendet werden
→ Write-Back aktualisiert Datenquelle verzögert
 - Deshalb: Datenkonsistenz muss bei der Verwendung von Caches gewährleistet werden
 - Abhilfe schaffen sogenannte Cache-Kohärenzprotokolle, die Cache-Zugriffe überwachen und synchronisieren, bzw. Kontrolleinheiten blockieren



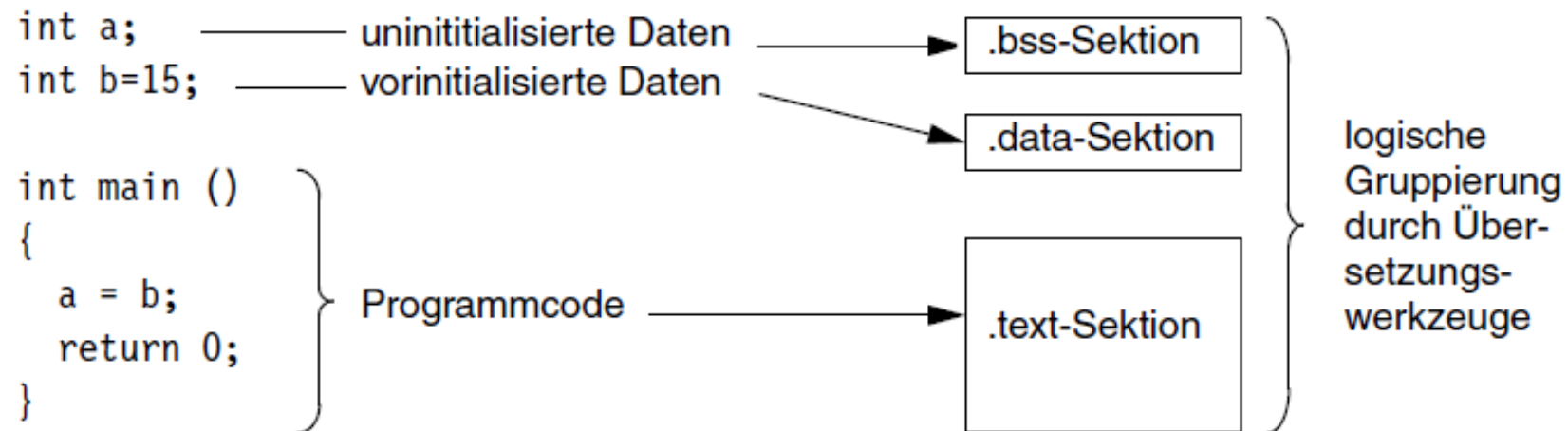
Kapitel XI

Prozessadressräume

- Ein Prozessadressraum reserviert für den jeweiligen Prozess einen Speicherbereich, in dem Adressräume reserviert sind für
 - Code
 - Daten
 - Heap
 - Stack
- Dabei: Zur Laufzeit können zusätzlich gemeinsame Speicherbereiche (Shared Memory) mit anderen Prozessen eingerichtet werden
- Aufgabe des Betriebssystems besteht in der Steuerung und Überwachung der Belegungen des Adressraumes

- Bei der Übersetzung eines Programmes von Hochsprache in ausführbaren Code:
Übersetzungswerkzeuge gruppieren einzelne Programmteile logisch entsprechend der Adressraumnutzung
- Eine Gruppierung von Programmteilen stellt dabei eine *Sektion* dar
- Beim Laden eines Programms:
Betriebssystem behandelt die verschiedenen Sektionen in unterschiedlicher Weise (siehe im Folgenden)
- Das Betriebssystem ist durch die Gruppierung der Programmteile in der Lage:
 - entsprechende Adressraum-Bestandteile bereitzustellen
 - Adressrauminhalte aus der ausführbaren Datei zu laden

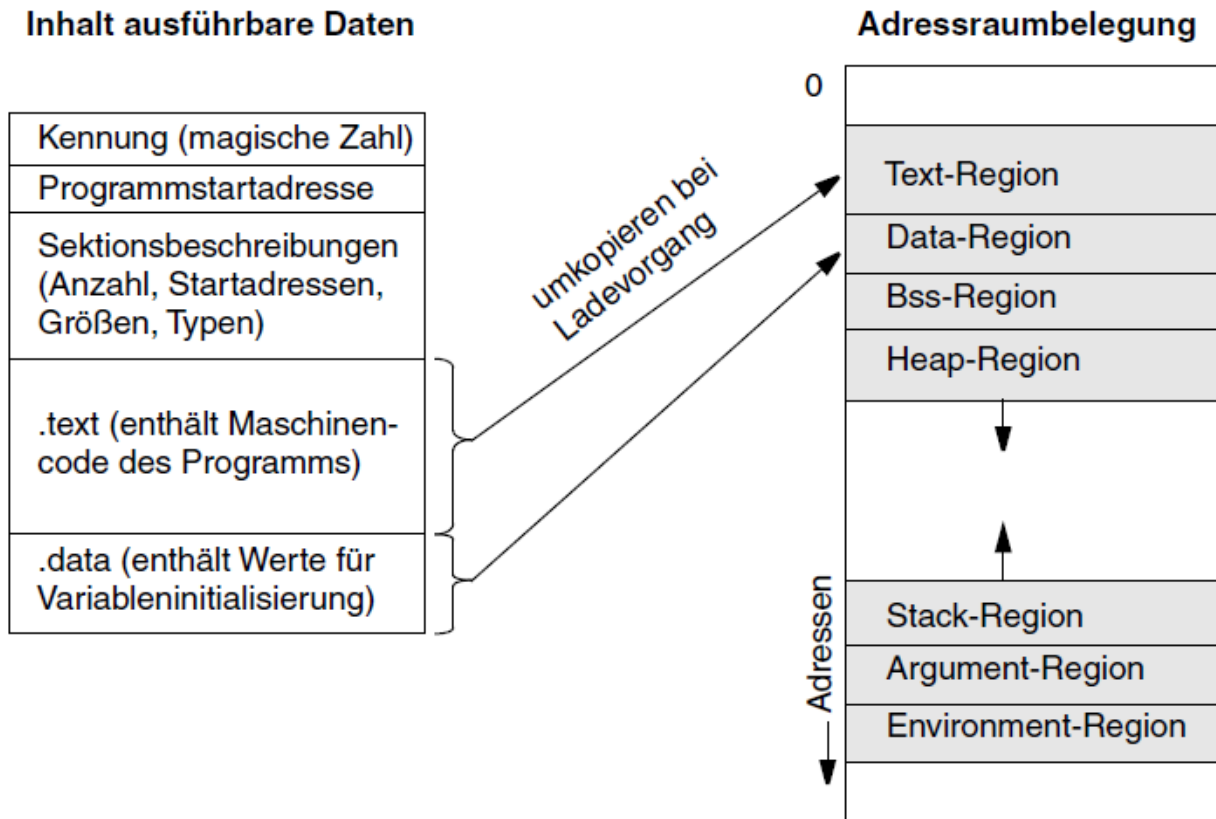
- Sektionsnamen unter GCC (GNU Compiler Collection):



Quelle: [BS15]

- Aufgaben des Betriebssystems hinsichtlich der Sektionen:
 - .bss: Speicherbereich reservieren
 - .data: Speicherbereich reservieren und Setzen der Initialwerte für die Variablen
 - .text: Den in der Sektion enthaltenen Maschinencode in Adressraum kopieren;
anschließend kann Betriebssystem im Sinne der Sicherheit den Adressraum mit Schreibschutz versehen
- Außer den Sektionen noch weitere Inhalte in einer ausführbaren Datei vorhanden, auf die im Folgenden eingegangen wird

- Genauer Aufbau von ausführbaren Dateien und Adressraumbellegung

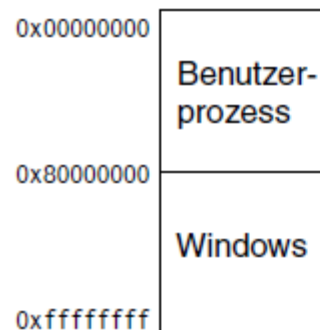
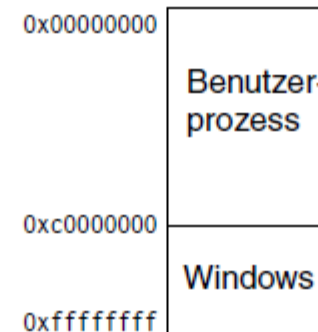
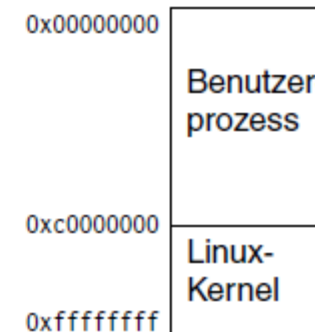


Quelle: [BS15]

Adressraumnutzung durch Programme (V)

- Magische Zahl: Spezieller Zahlenwert als Programm-Kennung
- Programmstartadresse: Gibt Auskunft darüber, an welcher Adresse Programm zu starten ist

- Platzierung des Betriebssystems im Adressraum:
 - Betriebssystem muss aufgrund etwaiger Systemaufrufe durch Benutzerprozess im Adressraum sichtbar sein
 - Daher: Zweiteilung des Adressraums mit in der Regel kleinem Bereich für Betriebssystem und dem Rest für Applikationsprogramme
 - Abhängig vom Betriebssystem unterschiedliche Ausgestaltung:

Windows (Standard)**Windows (3 GB-Option)****Linux (auf PC)***Quelle: [BS15]*

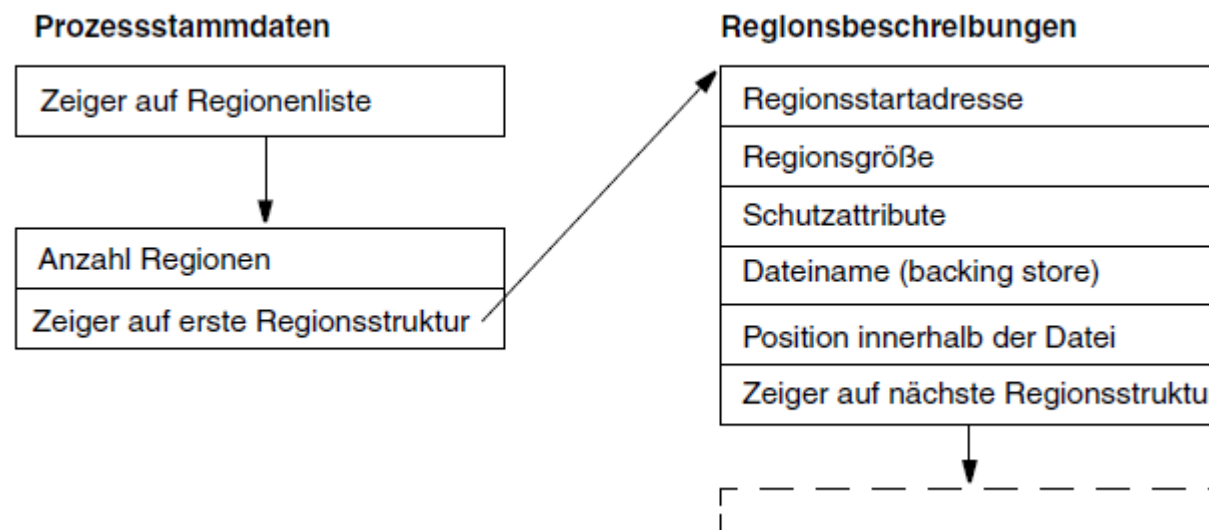
- Betriebssystem verwaltet pro Prozess die im Adressraum belegten und freien Bereiche
- Oftmals bei Prozessen: Adressraum ist nur schwach belegt, sodass Betriebssystem typischerweise nur Buch über Belegungen führt
- Dies ist Voraussetzung dafür, dass zur Prozesslaufzeit weitere Bereiche kollisionsfrei reservierbar sind
- Bei passender Hardware zusätzlich: Schutz vor Fehlzugriffen des Speichers
- Nachträgliche Speicherplatzreservierungen können aus verschiedenen Gründen erforderlich sein, auf die im Folgenden eingegangen wird

- Bei Thread-Erzeugung:
Bereitstellung des Stack-Bereichs für den Thread / ggf. zwei Stack-Bereiche für Kern- und Benutzermodus
- Einrichtung von Shared Memory:
Für Interprozesskommunikation
- Laden von Bibliotheksdateien:
Inhalt der Bibliotheksdatei wird geladen und in geeignetem Bereich des Speichers verfügbar gemacht
- Gemeinsame Bibliothek, sogenannte Shared Library:
Mehrere Prozesse nutzen gemeinsame Bibliothek, die einmal geladen und im Sinne des Shared Memory bereitgestellt wird
- Einrichten speicherbasierter Dateien:
Dateiinhalte oder Teile einer Datei werden im Adressraum sichtbar und änderbar gemacht

- Verwaltung von Regionen

- Regionen repräsentieren lückenlos zusammenhängende Adressbereiche
- Diese weisen unterschiedliche Attribute auf:
 1. Startadresse
 2. Größe
 3. Schutzattribute (Lesen, Schreiben, Ausführen oder Kombination)
 4. Zugehöriger Hauptspeicher
- Insbesondere zum Suchen von Lücken im Adressraum von Relevanz: Attribute 1 und 2
- Bei virtueller Speichertechnik: Attribut 3 erlaubt Identifizierung von Fehlzugriffen und Attribut 4 die Zuordnung zum Hauptspeicher

- Verwaltungsdatenhaltung



Quelle: [BS15]



Kapitel XII

Realer Speicher

- Realer Speicher ist insbesondere bei früheren Speicherverwaltungsformen zum Einsatz gekommen, welche uniprogrammierbar waren oder einfache Formen der Multiprogrammierung unterstützten
- In diesem Zusammenhang: Wdh. Uniprogrammierung aus Abschnitt Prozessverwaltung:
 - Betriebssystem und Benutzerprogramm teilen sich gemeinsam den vorhandenen Hauptspeicher
 - Maximale Programmgröße ist durch physisch verfügbaren Speicher begrenzt
 - Soll neues Programm gestartet werden, muss aktuell ausgeführtes terminiert werden
 - Heute vorwiegend im Bereich eingebetteter Systeme vorzufinden

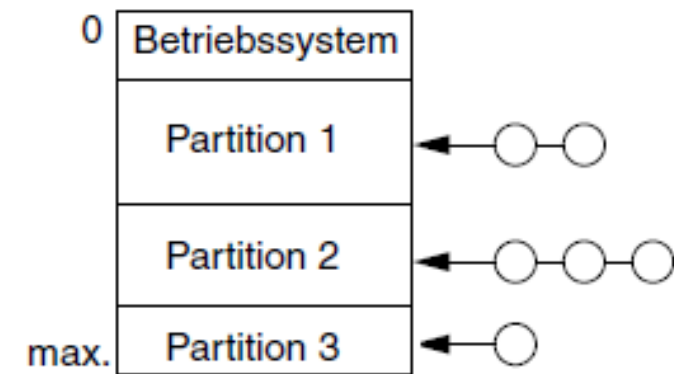
- Multiprogrammiersysteme hingegen erlauben auf Basis des CPU-Scheduling gleichzeitige Ausführung mehrerer Programme
- Einzige Einschränkung: Wird Prozess ausgeführt, muss sich zugehöriges Programm vollständig im Hauptspeicher befinden
- Zur Aufteilung des vorhandenen Speichers auf die einzelnen Benutzerprogramme existieren folgende Möglichkeiten:
 - Partitionierung mit fester Speichergröße pro Prozess
 - Partitionierung mit variabler Speichergröße pro Prozess
- Im Folgenden: Nähere Betrachtung der beiden Möglichkeiten

- Grundgedanke: Verfügbarer Speicher wird in n feste Bereiche (Partitionen) unterteilt, ablaufbereite Prozesse werden jeweils einer Partition zugewiesen
- Für nicht berücksichtigte Prozesse: Existenz von Warteschlangen
- Je nach Erfordernis: Partitionen haben alle dieselbe Größe oder sind verschieden groß
- Zur Zuteilung Prozess \Leftrightarrow Partition existieren zwei verschiedene Ansätze:
 - Verteilte Warteschlange: 1 Partition \Leftrightarrow 1 Warteschlange
 - Zentrale Warteschlange: Alle Partitionen \Leftrightarrow 1 Warteschlange

▪ Verteilte Warteschlange:

- Jede Partition besitzt eigene Warteschlange
- Ablaufwilliger Prozess trägt sich bei kleinstmöglicher Partition ein, die gerade ausreichend ist
- Dadurch: nahezu optimale Speicherausnutzung
- Allerdings: Fall kann eintreten, dass Partitionen zwar frei sind, Prozesse aber dennoch warten müssen weil sie sich aufgrund der kleinstmöglichen Partitionsgröße bei einer stärker genutzten Partition eingereiht haben

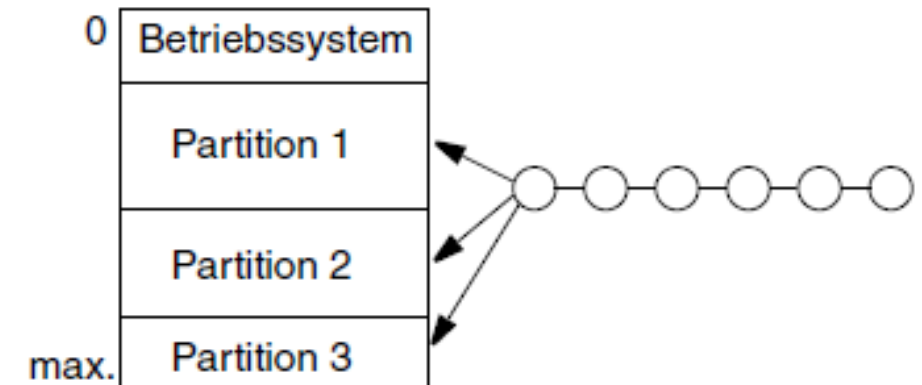
(A) Verteilte Warteschlange



Quelle: [BS15]

- Zentrale Warteschlange:
 - Alle Partitionen besitzen gemeinsame Warteschlange
 - Fall kann nicht mehr eintreten, dass Prozesse warten müssen obwohl Partitionen frei sind
 - Allerdings: Wenn Prozess mit geringem Speicherbedarf großer Partition zugeordnet wird: ineffiziente Speichernutzung
 - Abhilfe schafft Partitionszuordnung anhand des am besten in eine Partition passenden Prozesses; hierbei allerdings wiederum Gefahr des Verhungerns von Prozessen

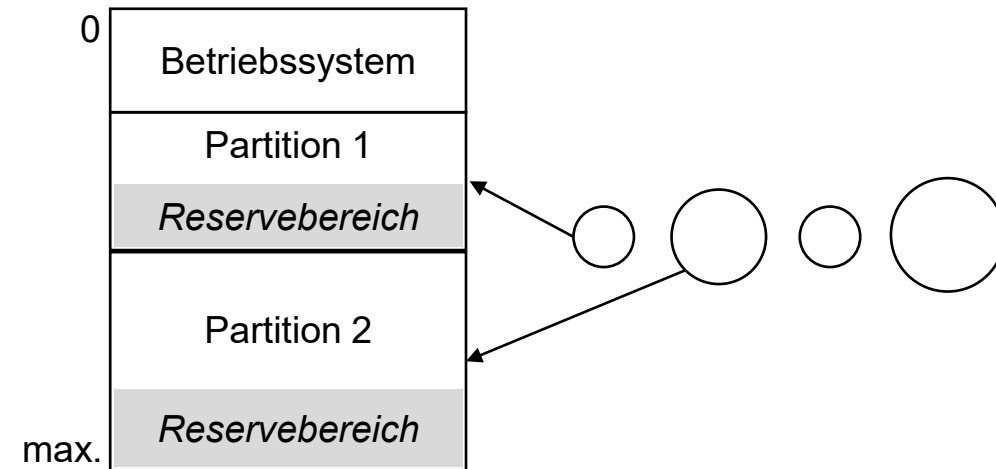
(B) Zentrale Warteschlange



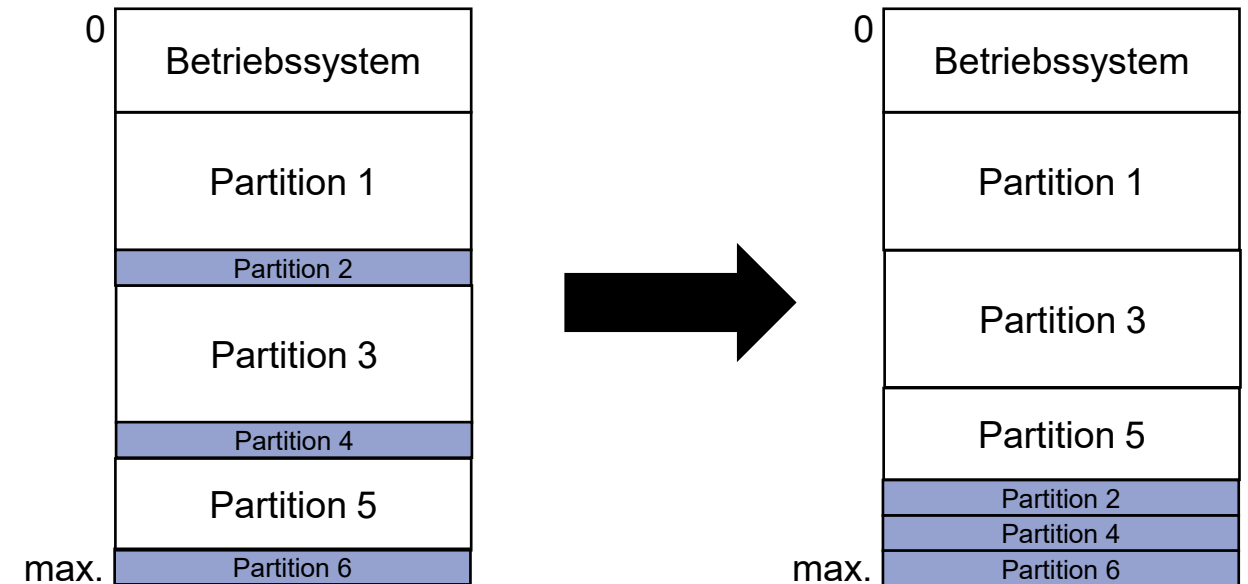
Quelle: [BS15]

- Partitionen variabler Größe ermöglichen es, Partitionsgrößen entsprechend der Bedürfnisse einzelner Programme zu wählen
- Wie bei Partitionen fester Größe muss auch hier der Platzbedarf des jeweiligen Programms bereits beim Programmstart bekannt sein
- Wächst die benötigte Speichergröße eines Programms zur Laufzeit an, kann dies mit Hilfe eines sogenannten *Reservebereichs* bewerkstelligt werden
- Es existiert in jedem Fall eine zentrale Warteschlange für alle Prozesse

- Bei Speicherzuteilung:
 - Suchen einer ausreichend großen Lücke für den entsprechenden Prozess
 - Dafür Verwendung verschiedener Suchalgorithmen
- Bei Speicherfreigabe:
 - Prüfen, ob benachbarte Lücken zusammengelegt werden können
 - Bei Fragmentierung des Speicherplatzes, also der Entstehung vieler kleiner und nicht mehr brauchbarer Lücken:
Durchführen einer Speicherverdichtung bzw. Kompaktierung



- Speicherverdichtung / Kompaktierung:
 - Verschiebung der Programme im Speicher so, dass sie lückenlos hintereinander platziert sind
 - Dadurch: Möglichkeit der Zusammenlegung der Partitionen 2, 4, 6 zu einer großen Partition
 - Nachteilig: Kompaktierung zieht Relokationen nach sich → Hoher Aufwand und Zeitbedarf

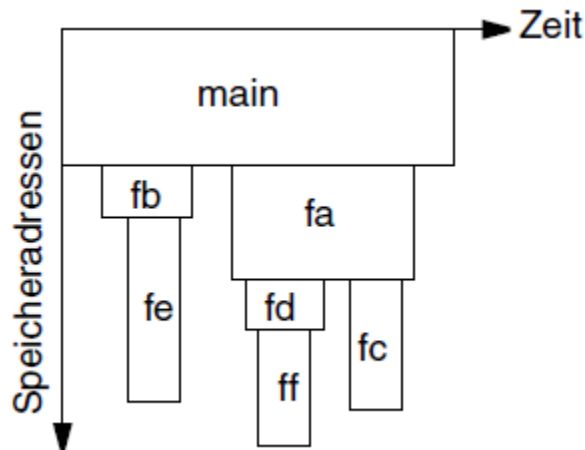


Genutzte Partition
Ungenutzte Partition

- Insbesondere bei der Anwendung der Multiprogrammierung: Problem des knappen Speichers
- Speicherknappheit liegt dann vor, wenn:
 - zur Verfügung gestellter Adressraum nicht ausreicht und / oder
 - die Hauptspeicherkapazität erschöpft ist
- Für solche Fälle existieren zwei Verfahren, die bei der Verwaltung des realen Speichers angewendet werden können:
 - Overlay-Technik: Heute kaum noch in Verwendung
 - Swapping: Heute primär im Einsatz, wenn Hardware für das Paging nicht verfügbar ist

■ Overlay-Technik:

- Älteste Technik zur Ausführung von Prozessen bei kleinem Hauptspeicher
- Auch bei der Monoprogrammierung angewendet
- Idee: Es werden immer nur aktuell benötigte Programmteile in den Hauptspeicher geladen
- Dabei zu beachten: Noch aktive Programmteile dürfen nicht überschrieben werden



Quelle: [BS15]

▪ Swapping:

- Basiert in seiner Grundform auf der Multiprogrammierung mit festen Speicherpartitionen, welche erweitert wird um Fähigkeit des Ein- und Auslagerns von Prozessen
- Idee: Auf Sekundärspeicher bzw. Festplatte wird sogenannte *Swapping Area* reserviert, die speziellen Bereich darstellt, auf den Prozesse ausgelagert werden können

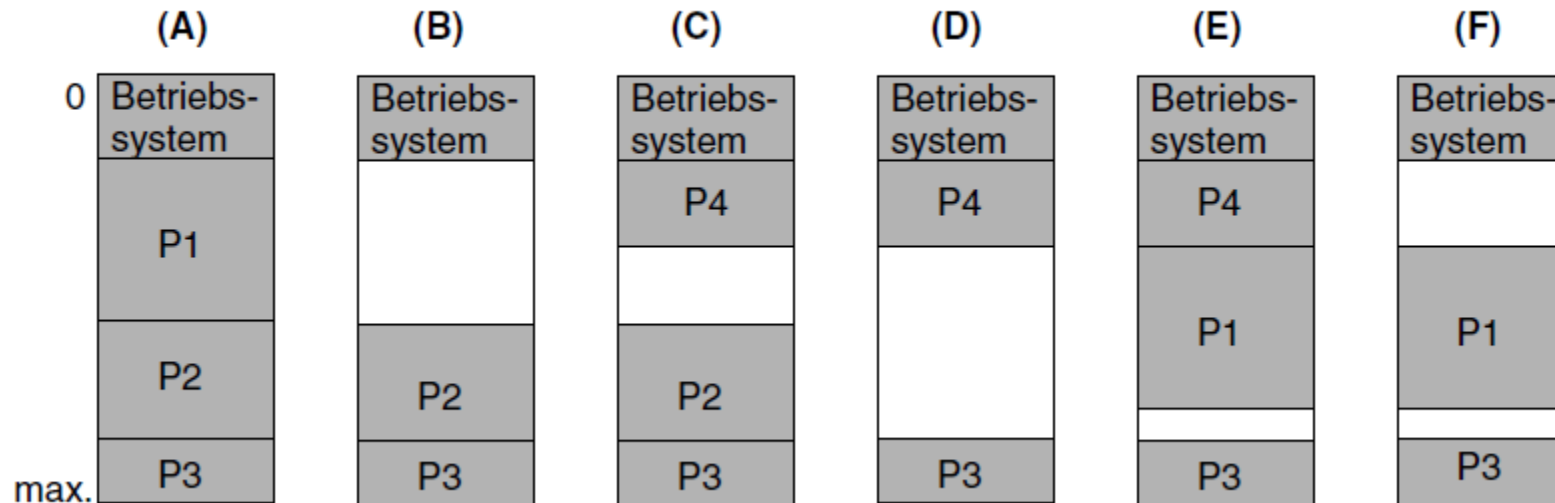


Quelle: [BS15]

- Schwierigkeit beim Swapping besteht im Festlegen der Partitionsgrößen für die Prozesse:
 - Erforderliche Informationen über benötigten Speicherplatz nur für Programmstart verfügbar
 - Bei wachsendem Speicherbedarf müssen bereits im Voraus Annahmen getroffen werden und Reservebereiche in der Partition eingeplant werden, die nicht immer ausreichen
- Abhilfe mittels Swapping mit dynamischer bzw. variabler Partitionierung:
 - Bei der Multiprogrammierung hohe Ein- und Auslagerungsfrequenz der Prozesse
 - Beim Einlagerungsprozess deshalb: Partitionsgröße kann jeweils an die neuen Speicheranforderungen angepasst werden
- Mit Hilfe des Swapping-Verfahrens: Gesamtanzahl der Prozesse \neq Anzahl Prozesse im Speicher, sondern: Gesamtanzahl der Prozesse = Anzahl Prozesse im Speicher + ausgelagerte Prozesse

- Dynamische Partitionierung beim Swapping:

Beispiel-Belegungsabfolge für eine Situation, in der mehr Prozesse gestartet wurden, als Speicherplatz zur Verfügung steht



Quelle: [BS15]