— Informatik I —
Modul 6: Betriebssysteme

Universität Zürich[UZH]

CSG
Communication Systems Group

ifi

Modul 7: Betriebssysteme

- Überblick von Betriebssystemen
- Auftrags- und Speicherverwaltung
- Einlagerung, Zuweisung, Ersetzung

Definition Betriebssystem

Definition des Begriffs „Betriebssystem“ (Operating System):
z.B. nach dem Deutschen Institut für Normung (DIN 44300):

> Ein Betriebssystem umfaßt die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

➔ Ziel eines Betriebssystems ist es, die semantische Lücke zwischen Anwendung (wünscht möglichst hohe Operationen) und Hardware (bietet elementare Operationen) zu verkleinern.

Semantische Lücke

- Applikation und Hardware sind (weit) voneinander entfernt.
  - Semantische Lücke!

- Betriebssystem erweitert die Fähigkeiten der Hardware.
- Betriebsystem verkleinert die semantische Lücke.

Geschichte von Betriebssystemen (1)

1. Generation (1940–1950): Röhren und Steckbretter

- Erste Rechner hatten überhaupt kein Betriebssystem
- Programme wurden in Maschinensprache geschrieben
- Nicht selten wurden Kabel umgesteckt
- Programmiersprachen waren unbekannt

2. Generation (1950–1960): Transistoren und Stapelsysteme (batch systems)

- Mit Einführung der Lochkarte 1950 war es möglich, Programme auf Lochkarten zu schreiben und einzulesen
- Erfindung des Transistors (leistungsfähigere Rechner)
  - Unterscheidung zwischen Designer, Computer-Architekten,Operator, Programmierer und Wartungspersonal

Geschichte von Betriebssystemen (2)

3. und 4. Generation (1960–1975): Spooling – Simultaneous Peripheral Operation On Line

- Jobs wurden von Lochkarten auf Festplatte (in Warteschlange) geschrieben, sobald diese in Computerraum gebracht wurden.
- Bei Job-Ende, konnte das Betriebssystem sofort neuen Job von der Festplatte in leeren Speicherbereich laden und starten.

Timesharing – Einteilen der CPU-Zeit in Zeitscheiben

- Jeder Anwender hat eigenes Terminal (Bildschirm und Tastatur), das direkt mit Computer verbunden ist
- Abwechselnd hat jeder Benutzer die CPU für kurze Zeitspanne
  - Kurze Jobs wurden in dieser Zeitscheibe erledigt, zeitaufwendigere Jobs benötigten mehrere Zeitscheiben
  - Nach Ablauf, wurden Jobs aus CPU entfernt und wieder in Warteschlange eingereiht. Bei nächster Zuteilung der CPU wurde Ausführung an Stelle fortgesetzt, an der Unterbrechung stattfand.

Geschichte von Betriebssystemen (3)

5. Generation (1975 - heute): Computernetze und PCs

- Wichtige Betriebssysteme
  - MS-DOS als erstes standardisiertes Betriebssystem für Personal Computer (PC)
    - „Single-user“-Systeme
  - WindowsXX-Systeme als Nachfolger von MS-DOS
    - Multitasking und später auch Multiuser
  - MacOS/iOS als Betriebssystem für „Apple“-Systeme
    - Multitasking und Multiuser
  - Unix/Linux als Betriebssystem für Großrechner und PCs
    - Multitasking und Multiuser

- Neben heute allgemein bekannten Betriebssystemen aus dem PC-Bereich, wie z.B. eben Linux und Windows, gibt es vor allem für Embedded und Mobile Systeme viele weitere Betriebssysteme.

Aufgaben eines Betriebssystems (1)

- Schnittstelle zwischen Mensch und Hardware
  - Bedienschnittstelle (UI, User Interface)
    - Dialogorientierte Konsole
      - Benutzer gibt über Tastatur Befehle ein und erhält entsprechende Antworten
    - Grafische Benutzeroberfläche
      - Intuitive Erledigung von Aufgaben über Aktivieren von Schaltflächen, Menüs und Symbolen mit der Maus
  - Programmierschnittstelle (API, Application Programming Interface)
    - Programmierer eine leicht verständliche und gut handhabbare Schnittstelle zur eigentlichen Maschine anbieten
    - Komplexität der darunterliegenden Maschine verstecken
    - Ansprechpartner für Programmierer ist nicht mehr Maschine, sondern virtuelle Maschine (Betriebssystem), die wesentlich einfacher zu verstehen bzw. zu programmieren ist

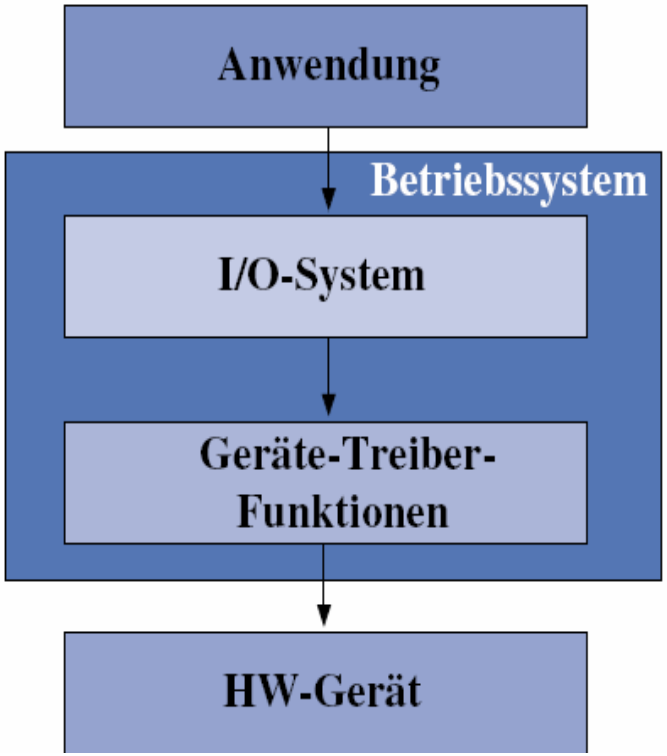
```
write(dateinummer, text_adresse, bytezahl);
```

Aufgaben eines Betriebssystems (2)

- Betriebsmittelverwaltung (*resource management*)
  - Verwaltet alle peripheren Betriebsmittel des Rechnersystems, wie z.B.
    - Festplatte
    - Floppy Disk
    - CDROM
    - Drucker, ...
- Auftragsverwaltung (Prozeßverwaltung, *tasks*)
  - Verwaltet Prozesse und Threads
- Speicherverwaltung (*memory management*)
  - Durch immer größer werdende Programme sowie mehrerer quasi gleichzeitig laufender Programme wird zur Verfügung stehender Arbeitsspeicher zu klein
- Allgemeine Verwaltung
  - Aller Einzelteile eines komplexen Systems (Prozessoren, Arbeitsspeicher, Festplatten, Bildschirme, Drucker, ...)

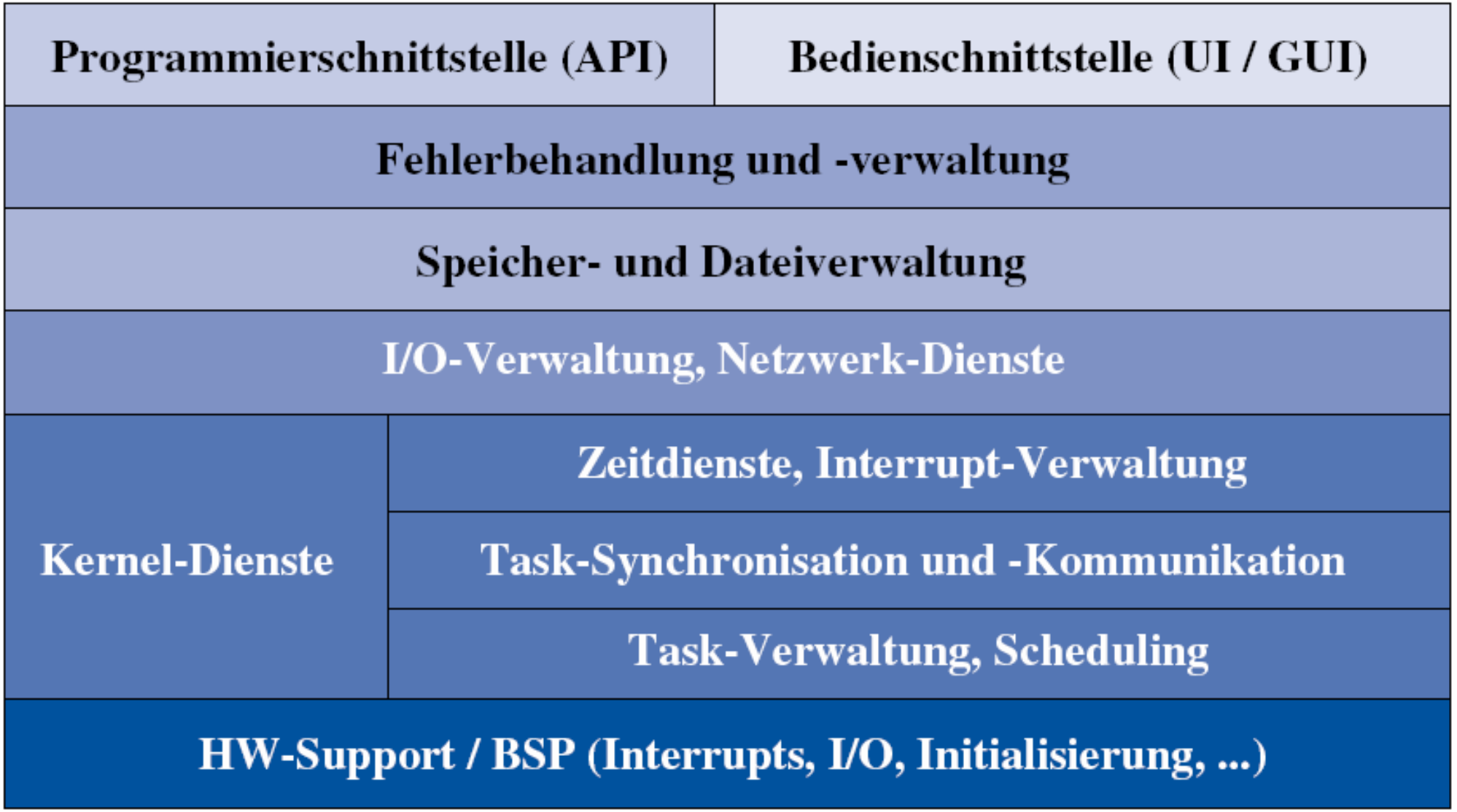
Aufgaben eines Betriebssystems (3)

- Geräteverwaltung und Treiber
- Die typische Standard-Treiber-Schnittstelle („Geräte-Treiber“) hat folgende Funktionen:
  - create(), open(), close(), remove()
    zum Anlegen, Öffnen, Schließen, Entfernen eines Geräts
  - read(), write()
    zum Lesen/Schreiben vom/zum Gerät
  - ioctl()
    zur Änderung von internen Geräteparametern.



- Kommunikation eines Anwendungsprogramms mit dem Gerät erfolgt über definierte I/O-Systemschnittstelle des Betriebssystems und Treiberprogramms

Schichtenaufbau von Betriebssystemen



Modul 7: Betriebssysteme

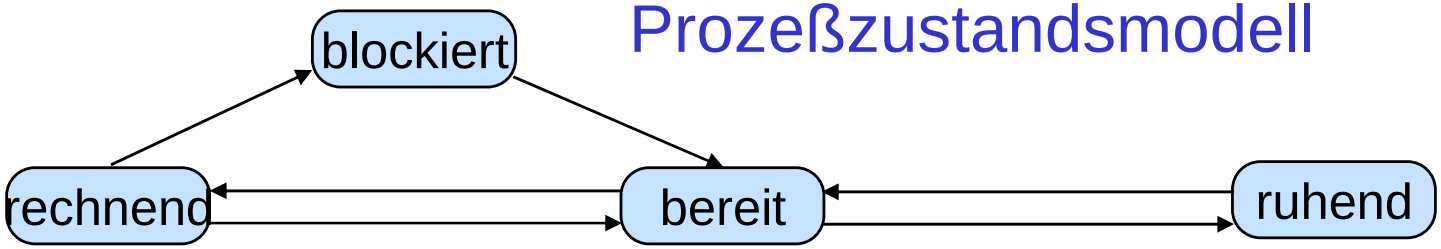
- Überblick von Betriebssystemen
- Auftrags- und Speicherverwaltung
- Einlagerung, Zuweisung, Ersetzung

Prozeß und Scheduling

- Schlüsselkonzept moderner Betriebssysteme: Prozeßkonzept
  - Anwendungsprogramme werden in Form von Prozessen verwaltet
  - Multitasking: mehrere Prozesse können abwechselnd bearbeitet werden
    - Bessere Ressourcenausnutzung: durch Multiplexing zu erreichen
  - Einfachere Programmierung: durch Aufteilung der Aufgaben zu erreichen
- Prozeß (Rechenprozeß, Auftrag) ist die Ausführung eines Algorithmus (Ablauf), der durch ein ausführbares Programm beschrieben ist.
  - Ein Prozeß ist ein aktives Programm
- Auswahl des nächsten zu bearbeitenden Prozesses und Umschaltung erfolgt durch Scheduler (selber ein Dienstprogramm) bzw. Dispatcher-Programm, das zu bestimmten Zeitpunkten oder bei bestimmten Ereignissen gestartet wird
  - Scheduler unterbricht dabei den gerade laufenden Prozeß und
  - Startet u.U. einen anderen Prozeß

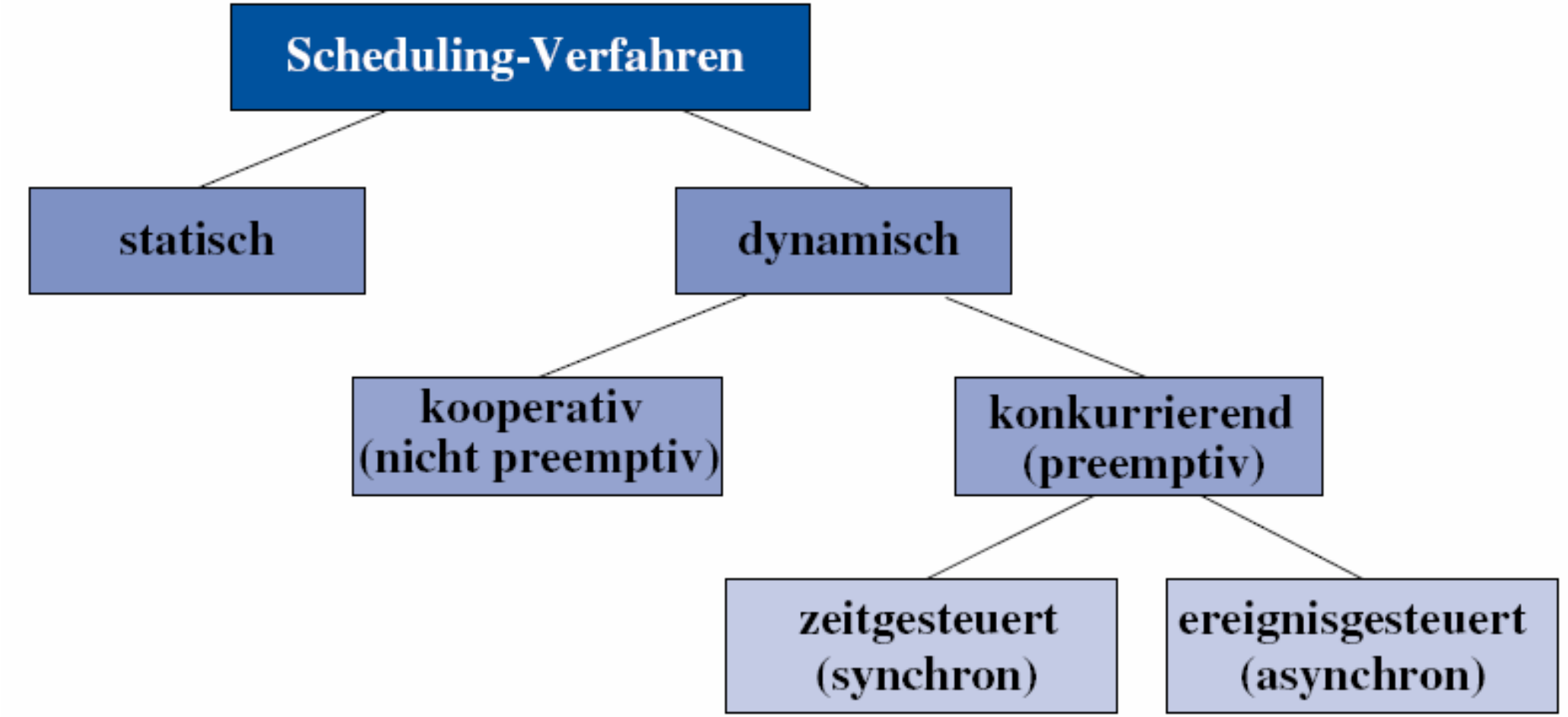
Prozesse

- Prozeß kann im Laufe seiner Bearbeitung im wesentlichen folgende unterscheidbaren Zustände (vereinfacht) annehmen:



- Operationen zu ihrer Kontrolle:
  - Create, exit, join, mutex_init, cond_wait, ...
- Prozeßkoordination:
  - Mutex, semaphore, wait conditions, ...
- Inter Process (IPC) und Inter Object Communications (IOC):
  - Pipes, sockets, RPC, RMI, COM, DCOM, Corba, .NET, ...
- Threads (leichtgewichtige Prozesse):
  - Abläufe im gleichen Adreßraum (Programm- und Datensegment)
  - Separater Registersatz, Programmzähler, Kellerspeicher

Scheduling-Verfahren für Prozesse (1)



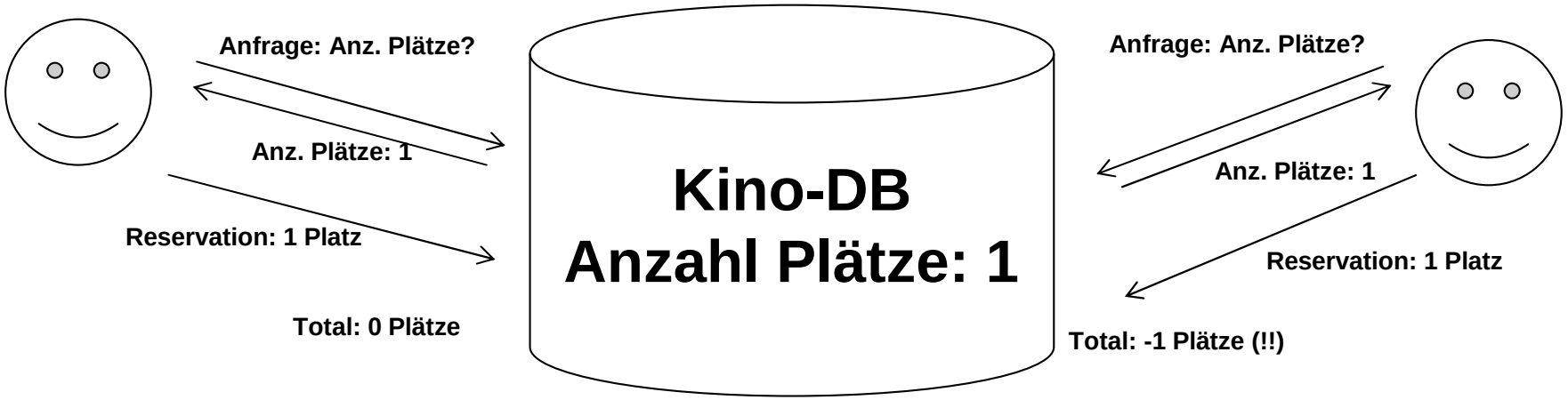
Scheduling-Verfahren für Prozesse (2)

- Für statische Systeme (z.B. kleine Embedded Systeme) mit immer gleichen Aufgaben wird statisches Scheduling („Zugfahrplan“) eingesetzt
- Für gößere Systeme mit beliebig start- und anhaltbaren Programmen werden dynamische Schedulingverfahren benötigt.

- Nächster abzulaufender Prozeß/Thread innerhalb eines Prozesses vom Scheduler nach bestimmten Algorithmus bestimmt.
  - Kooperative Verfahren: Prozeß muß den Prozessor freiwillig abgeben, d.h. direkt oder indirekt selbst Scheduler aufrufen, um anderem Prozeß Ablauf zu erlauben
  - Konkurrierende Verfahren: Scheduler kann dem gerade laufenden Prozeß den Prozessor entziehen, was eine vorzeitige Unterbrechung bewirkt (Preemption).
    - Scheduler muß für anzuhaltenden Prozeß eine Sicherung und für den ausgewählten weiterlaufenden Prozeß die Restaurierung des Prozeß-Kontextes durchführen (siehe Stapelspeicher).

Anwendungsfall eines Mutex (1)

- Situation: Zwei Internetbenutzer wollen den letzten Platz in einem Kino reservieren.



- Zeitliche Abfolge:
  - 1. Anfrage des 1. Benutzers nach Anzahl verfügbarer Plätze: 1
  - 2. Anfrage des 2. Benutzers nach Anzahl verfügbarer Plätze: 1
  - 3. 1. Benutzer reserviert den letzten Platz. Verfügbare Plätze: 0
  - 4. 2. Benutzer reserviert den (vermeintlich letzten Platz). Verfügbare Plätze: -1

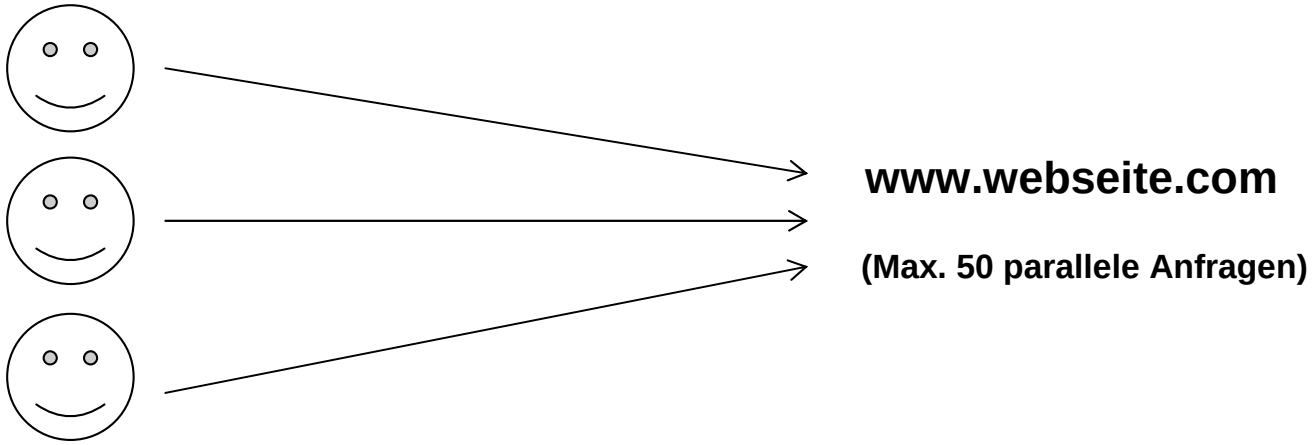
Anwendungsfall eines Mutex (2)

- Folgerungen:
  - Ohne erneutes Lesen aus der Datenbank nach Schritt 3 geht der 2. Benutzer fälschlicherweise davon aus, daß noch ein Platz frei ist
  - Bei Schreiboperationen darf immer nur ein Benutzer eine Änderung vornehmen, ansonsten kann es zu Inkonsistenzen in der Datenbank kommen.

- Lösung:
  Mutex - Nur der Benutzer, der den Mutex hat, darf eine Veränderung machen.

Anwendungsfall einer Semaphore

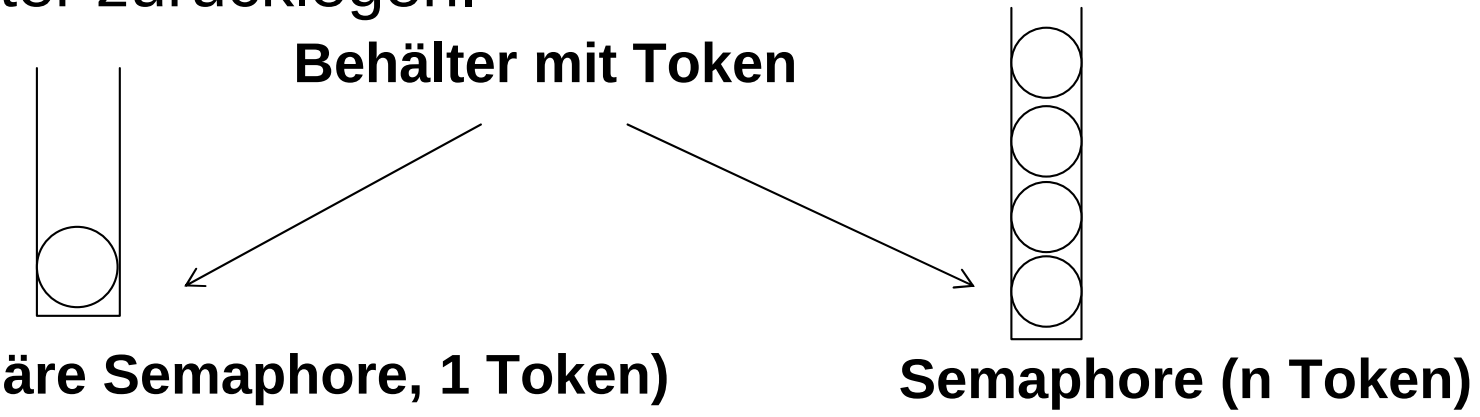
- Situation: 100 Benutzer wollen auf eine Webseite zugreifen. Es ist jedoch bekannt, dass der Webserver bei über 50 parallelen Zugriffen abstürzt.



- Lösung: Die Semaphore steuert den Zugriff auf die Webseite, indem genau 50 „Zugriffsrechte“ vergeben werden.

Implementierung von Mutex und Semaphore

- Implementierung mittels Tokens
  - Token bezeichnet das Recht, eine bestimmte Aktion ausführen zu dürfen
- Ein Prozeß, der eine Aktion ausführen will, muß ein Token aus einem Behälter nehmen, und dieses nach Gebrauch wieder in den Behälter zurücklegen.



- Ist der Behälter leer, muß der Prozeß warten, bis ein anderer Prozeß ein Token zurück in den Behälter gelegt hat.
- Fehlerquellen:
  1. Der Prozeß legt das Token nicht mehr zurück
  2. Der Prozeß stürzt ab, während er das Token noch besitzt

Kritische Regionen in Java

- In Java wird das Monitor-Pattern verwendet, um Zugriffe auf Datenstrukturen zu synchronisieren, welche von mehereren Threads verwendet werden.

- Bsp:

```
public class Container {
  private final List<Integer> list = new ArrayList<Integer();
  public boolean add(Integer i){
      boolean r = false;
      synchronized (list) { // list is used as monitor
           r = list.add(i);
      }
      return r;
  }
}
```

Illustration Deadlock (1)

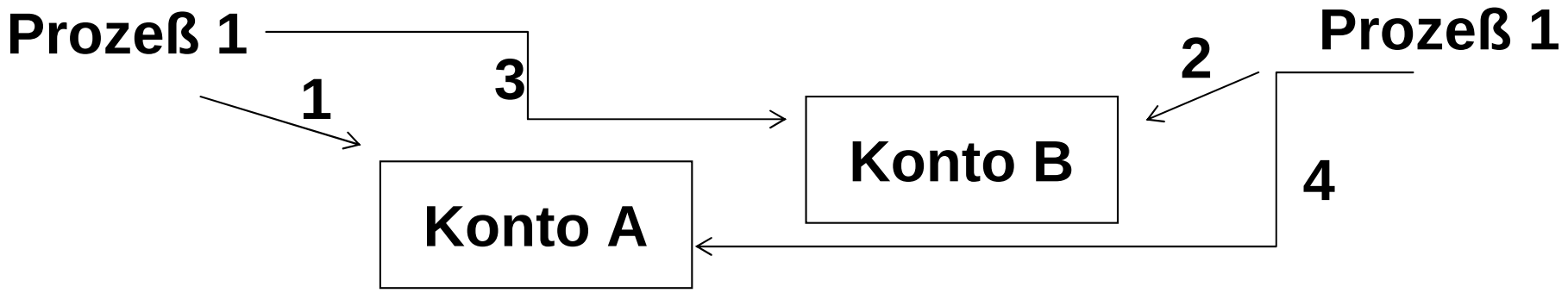
- Anhand des Kinoreservationssystems wurde gezeigt, daß man (z.B. mittels eines Mutex) sicherstellen muß und daß immer nur ein Prozeß gleichzeitig eine Änderung an einem Datum vornehmen darf.

- Dies kann jedoch zu Problemen führen, wenn für eine Veränderung mehrere Dateien involviert sind!

- Beispiel: Gegeben
  - Zwei Bankkonti A und B.
  - Ein 1. Prozeß, der einen Betrag von A abbuchen und auf B gutschreiben möchte
  - Ein 2. Prozeß, der einen Betrag von B abbuchen und auf A gutschreiben möchte

Illustration Deadlock (2)

- Zeitliche Abfolge:
  - 1. Prozeß 1 erwirbt ein Lock auf Konto A
  - 2. Prozeß 2 erwirbt ein Lock auf Konto B
  - 3. Prozeß 1 möchte ein Lock auf Konto B erwerben -> scheitert
  - 4. Prozeß 2 möchte ein Lock auf Konto A erwerben -> scheitert



- Konsequenz: Deadlock. Jeder Prozeß wartet darauf, daß der andere sein Lock freigibt
- Lösungsmöglichkeiten:
  - 1. Locks immer in der gleichen Reihenfolge erwerben
  - 2. Die Prozesse nach einer Wartezeit abbrechen und neu starten

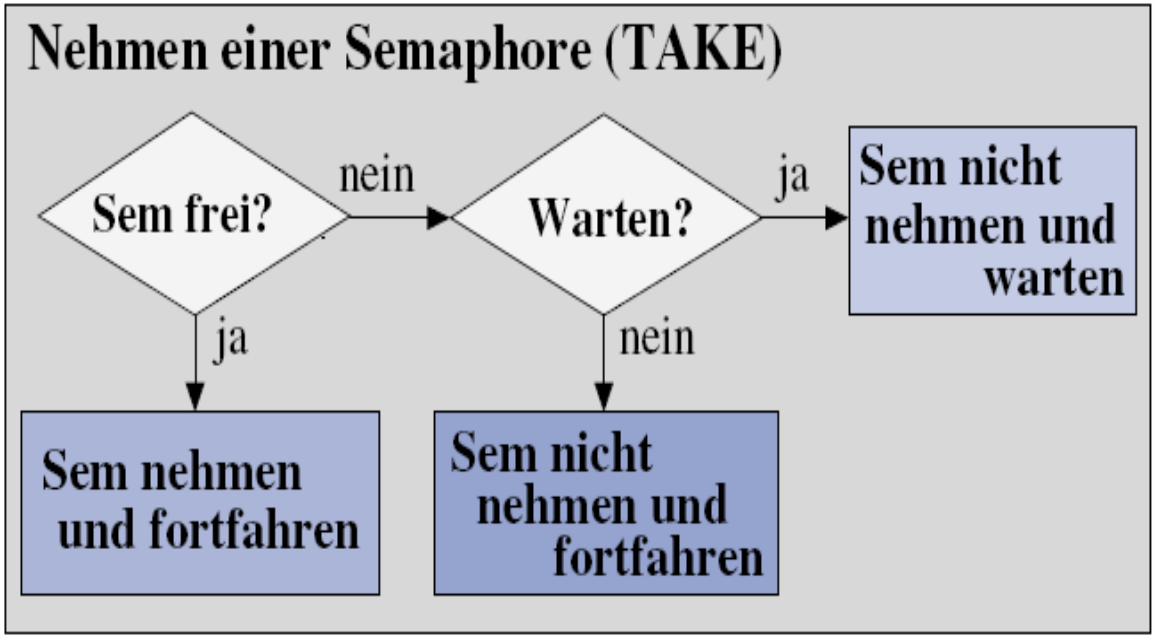
Synchronisationsmechanismen (1)

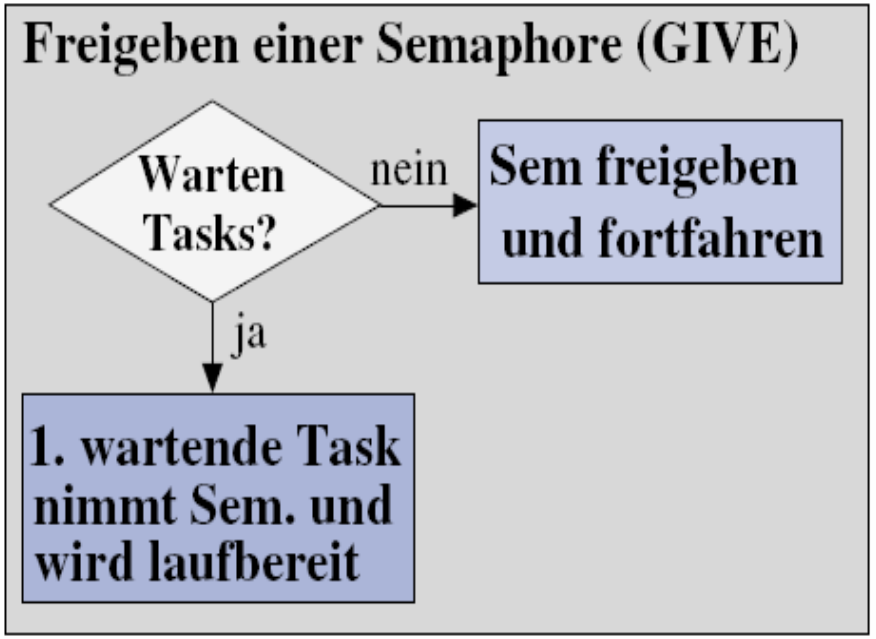
- Kritische Abschnitte (Critical Regions)

```
                                    int  std=11, min=59;
/* Task 1 */                        |
min = min + 1;                      |
if ( min == 60 ) {                  |
  std = std + 1;                    |
        /* Hier wird Task 1        |   /* Task 2 */
           von Task 2 abgebrochen */|   Ausgabe: Uhrzeit=std:min
  min = 0;                          |            (12:60)
}                                   |
```

Synchronisationsmechanismen (2)

- Semaphore (Sem) häufig zur Synchronisation von Prozessen/Threads mittels gegenseitigen Ausschlusses (mutual exclusion) eingesetzt
- Semaphor-Variablen von Dijkstra definiert und als P/V-, DOWN/UP- oder TAKE/GIVE-Operationen modelliert

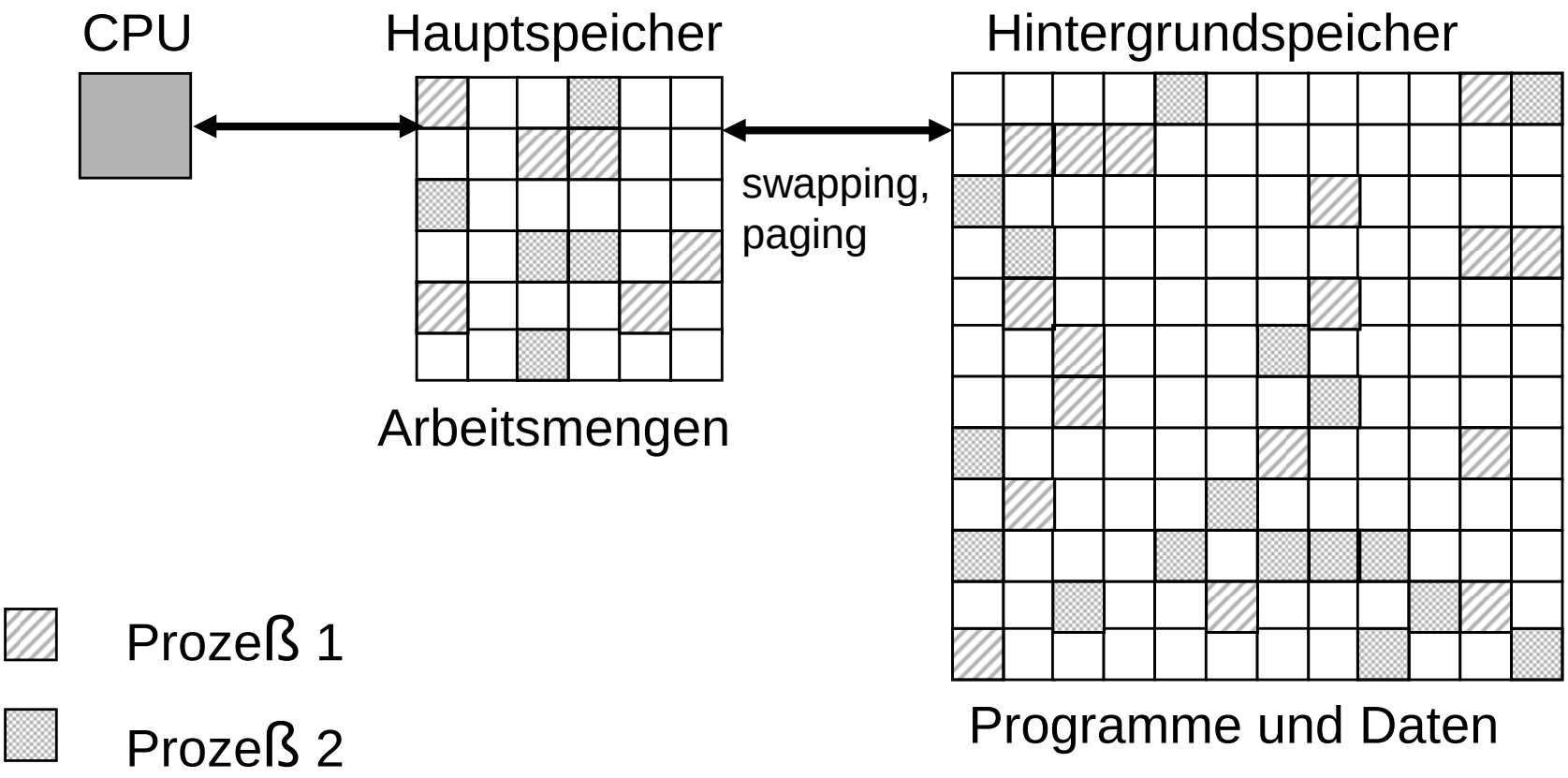




Synchronisationsmechanismen (3)

- Semaphore sind üblicherweise als boolsche oder ganzzahlige Variable mit einer dazugehörigen Warteschlange (WS) für Tasks modelliert
  - Möchte Task nicht freie Semaphore nehmen, wird sie auf „wartend“ gesetzt und in die Semaphore-WS eingetragen
  - Beim Zurückgeben der Semaphore wird eine in der WS eingetragene Task sofort wieder „laufbereit“ gesetzt
- Dieses Verfahren stellt sicher, daß Tasks nur dann warten müssen, wenn sie gleichzeitig den gleichen kritischen Abschnitt betreten möchten

Grundstruktur virtueller Speicherverwaltung



Virtuelle Speicherverwaltung (1)

- Vorgang bleibt dem Anwender völlig verborgen, d.h. Arbeits-speicher erscheint dem Anwender wesentlich größer, als er ist.
- Ein nach diesem Konzept verwalteter Speicher heißt virtueller Speicher.
- Von modernen Prozessoren wird die Verwaltung dieses virtuellen Speichers hardwaremäßig durch eine MMU (Memory Management Unit) unterstützt.
- Hauptaufgabe dieser virtuellen Speicherverwaltung: Umsetzung virtueller (logischer) Adressen in physikalische Adressen

Virtuelle Speicherverwaltung (2)

- Benutzer:
  - Kennzeichnet Objekte (Programme, Unterprogramme, Variablen) durch Namen
- Compiler:
  - Übersetzt diese Namen in virtuelle (logische) Adressen
- Virtuelle Speicherverwaltung:
  - Wandelt diese Adressen zur Laufzeit je nach gerade gegebener Speicherbelegung in physikalische Adresse (wirklicher Ort des Objekts im Hauptspeicher) um

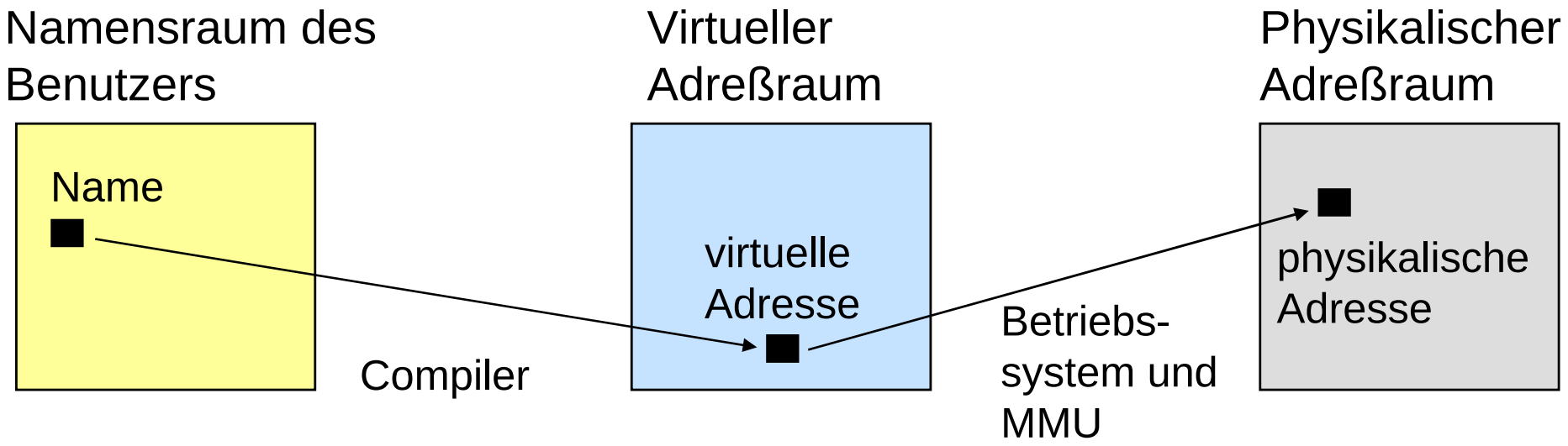
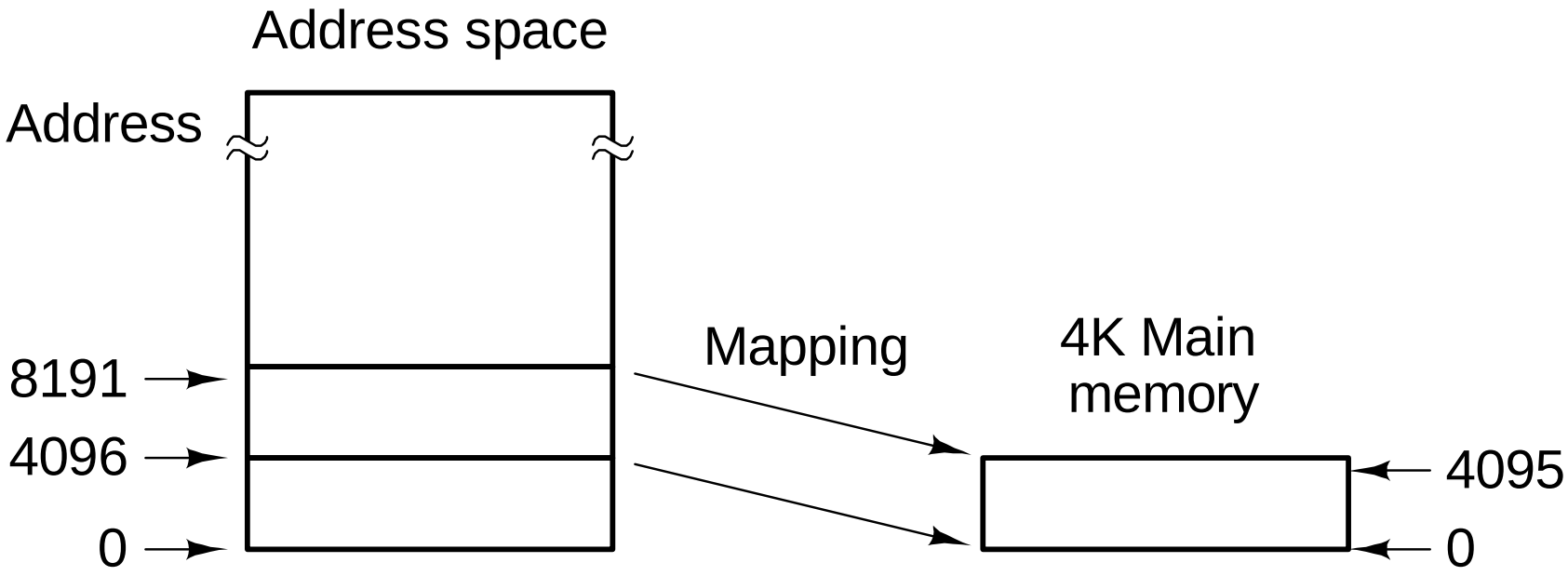


Abbildung virtueller Adressen



Name:
JMP Weiter

Code:
JMP -128

Logische Adresse:
4583

Physikalische Adresse:
23512

Compiler/ Assembler

dyn. Adressrechnung ((PC) - 128)

Virtuelle Speicher-verwaltung (MMU)

Segmentierungs- und Seitenwechselverfahren

- Es existieren zur virtuellen Speicherverwaltung zwei grundlegende Verfahren: Seitenwechsel und Segmentierung

- Aufteilung in Seiten (paging)
  - Hierbei wird der logische und der physikalische Adreßraum in "Segmente fester Länge", sogenannte Seiten (pages) unterteilt.
  - Die Seiten sind relativ klein (256 Byte - 4 kByte)
  - Ein Prozeß wird auf viele dieser Seiten verteilt (keine logischen Zusammenhänge wie bei der Segmentierung)
- Aufteilung in Segmente (swapping)
  - Hierbei wird der virtuelle Adreßraum in Segmente verschiedener Länge zerlegt.
  - Jedem Prozeß sind ein oder mehrere Segmente z.B. für den Programmcode und die Daten, zugeordnet.
  - Die einzelnen Segmente enthalten logisch zusammenhängende Informationen (Programm- und Datenmodule) und können relativ groß sein.

Seiten im virtuellen und realen Speicher

Page	Virtual addresses
≈	≈ ≈
15	61440 - 65535
14	57344 - 61439
13	53248 - 57343
12	49152 - 53247
11	45056 - 49151
10	40960 - 45055
9	36864 - 40959
8	32768 - 36863
7	28672 - 32767
6	24576 - 28671
5	20480 - 24575
4	16384 - 20479
3	12288 - 16383
2	8192 - 12287
1	4096 - 8191
0	0 - 4095

(a)

Page frame	Bottom 32K of main memory Physical addresses
7	28672 - 32767
6	24576 - 28671
5	20480 - 24575
4	16384 - 20479
3	12288 - 16383
2	8192 - 12287
1	4096 - 8191
0	0 - 4095

(b)

Diskussion

- Seitenaufteilung
  - Vorteile:
    - Durch kleine Seiten wird wirklich benötigter Teil des Programms eingelagert
    - Geringerer Verwaltungsaufwand als Segmentierung
  - Nachteil:
    - Häufiger Datentransfer
- Segmentierung
  - Vorteile:
    - Segmentierung spiegelt logische Programmstruktur wieder
    - Durch große Segmente relativ seltener Datentransfer
    - Segmentspezifische Schutzmaßnahmen möglich
  - Nachteile:
    - Wenn Datentransfer, dann jedoch umfangreich.
    - Besteht Programm nur aus einem Code- und Daten-Segment (wird vom Compiler oder Benutzer festgelegt), so muß vollständig eingelagert werden

Wesentliche Unterschiede

	Seiten	Segmentierung
Programmierertransparenz?	Nein	Ja
Anzahl linearer Adreßräume?	1	Viele
Kann die virtuelle Adreßgröße die Speichergröße übertreffen?	Ja	Ja
Können Tabellen variabler Größe behandelt werden?	Nein	Ja
Grund der Erfindung?	Emulation großer Adreßräume	Angebot mehrfacher Adreßräume

- Viele der heutigen Prozessoren unterstützen beide Techniken!

Modul 7: Betriebssysteme

- Überblick zu Betriebssystemen
- Auftrags- und Speicherverwaltung
- Einlagerung, Zuweisung, Ersetzung

Probleme der virtuellen Speicherverwaltung

- Beim Austausch von Daten zwischen Arbeits- und Hintergrundspeicher ergeben sich drei Problemkreise:

1. Der Einlagerungszeitpunkt

- Wann werden Segmente oder Seiten in den Arbeitsspeicher eingelagert ?

2. Das Zuweisungsproblem

- An welche Stelle des Arbeitsspeichers werden die Seiten oder Segmente eingelagert ?

3. Das Ersetzungsproblem

- Welche Segmente oder Seiten müssen ausgelagert werden, um Platz für neu benötigte Daten zu schaffen?

1. Einlagerungszeitpunkt

- Gängiges Verfahren:
  - Einlagerung auf Anforderung (Demand Paging bei Seitenverfahren)
- Hierbei werden Daten eingelagert, sobald auf sie zugegriffen wird, sie sich aber nicht im Arbeitsspeicher befinden.
- Der Zugriff auf ein nicht im Arbeitsspeicher vorhandenes Segment oder Seite heißt Segment- oder Seiten-Fehler *(segment fault, page fault).*

2. Zuweisungsproblem (1)

- Bei Seitenwechselverfahren
  - Dieses Problem ist nicht existent, da alle Seiten gleich groß sind und somit immer "passende Lücken" nutzbar ist

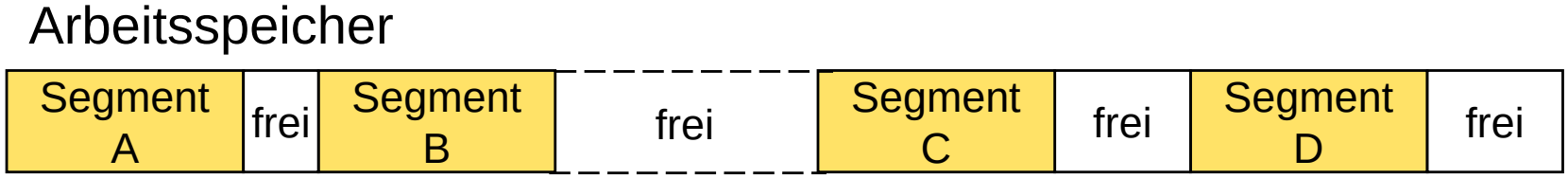
  ➔ keine externe Fragmentierung.
  - Jedoch: Problem der internen Fragmentierung
    - Diese entsteht bei der Aufteilung eines Programms auf die Seiten.
  - Einheitliche Seitengröße

  ➔ auf der letzten Seite jedes Programm-Moduls entsteht mit hoher Wahrscheinlichkeit ein ungenutzter Leerraum.
- Bei Segmentierungsverfahren:
  - Hier muß eine ausreichend große Lücke im Arbeitsspeicher gefunden werden.
  - Drei Strategien:
    - *first-fit:* erste passende Lücke wird genommen
    - *best-fit :* kleinste passende Lücke wird genommen
    - *worst-fit:* größte passende Lücke wird genommen

2. Zuweisungsproblem (2)

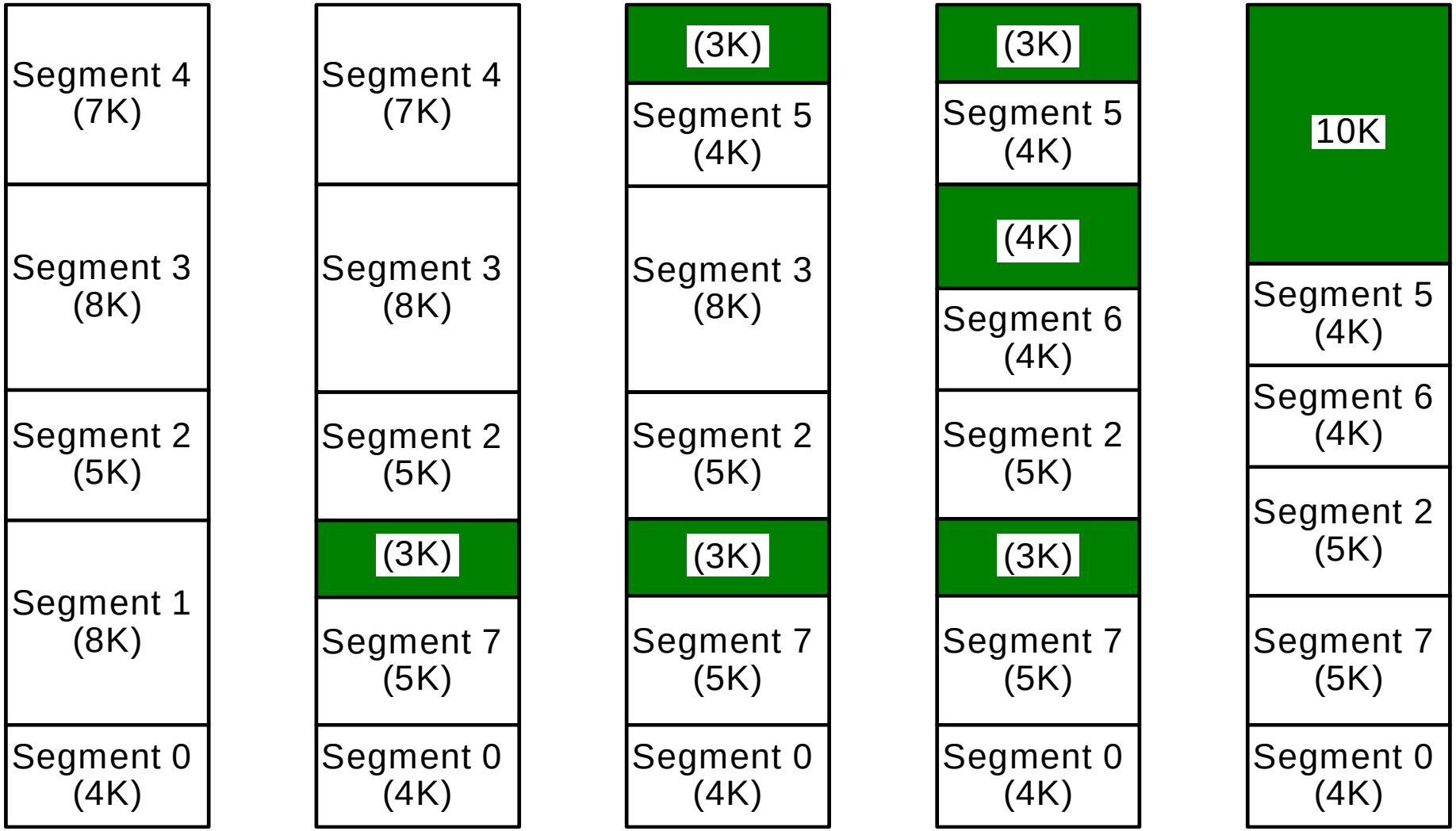
- Problem bei allen drei Verfahren:
  - Der Speicher zerfällt nach einiger Zeit in belegte und unbelegte Speicherbereiche

  ➔ externe Fragmentierung.



- Die unbelegten Speicherbereiche sind hierbei oft zu klein, um Segmente aufnehmen zu können

Externe Fragmentierung und Kompaktifizierung



3. Ersetzungsproblem

- Bei Segmentierungsverfahren:
  - Meist wird Anzahl gleichzeitig von einem Prozeß benutzbaren Segmente limitiert:

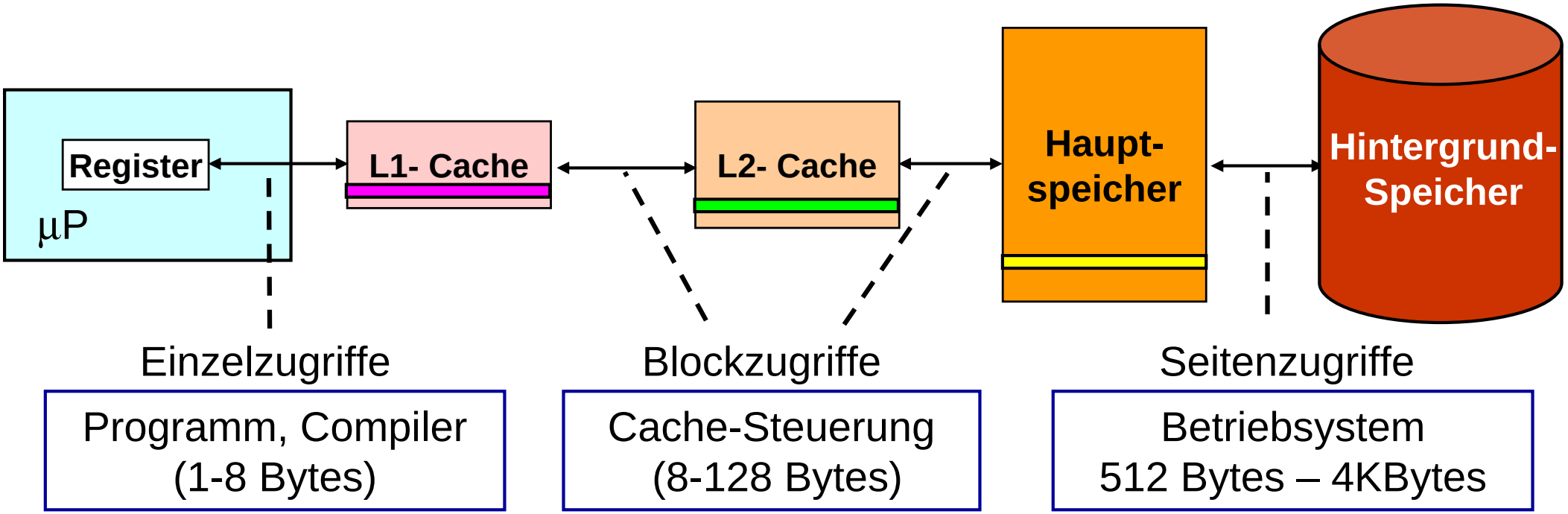
    ➔ Bei Einlagerung eines neuen Segments wird ein zuvor für diesen Prozeß benutztes Segment ausgelagert
  - Es ist jedoch auch eine der im folgenden für Seitenwechsel-Verfahren beschriebenen Methoden möglich
- Bei Seitenwechselverfahren:
  - **FIFO (*first-in-first-out*):** Sich am längsten im Arbeitsspeicher befindende Seite wird ersetzt
  - **LIFO (*last-in-first-out*):** Zuletzt eingelagerte Seite wird ersetzt
  - **LRU (*least recently used*):** Seite, auf die am längsten nicht zugegriffen wurde, wird ersetzt
  - **LFU (*least frequently used*):** seit ihrer Einlagerung am seltensten benutzte Seite wird ersetzt
  - **LRD (*least reference density*) :** Mischung aus LRU und LFU. Seite mit der geringsten Zugriffdichte (Anzahl Zugriffe / Einlagerungszeitraum) wird ersetzt

Segment- und Seitenfehler

- Sowohl bei segmentorientierter wie bei seitenorientierter Speicherverwaltung gilt:
  - Befindet sich eine Seite oder ein Segment nicht im Hauptspeicher, so löst der Prozessor eine Unterbrechung aus, um die Seite oder das Segment durch das Betriebssystem zu laden (Seiten- oder Segmentfehler)
- Erkennung eines Segmentfehlers:
  - Bit im Segment-Deskriptor zeigt an, ob Segment im Hauptspeicher ist oder nicht
- Erkennung eines Seitenfehlers:
  - Seitennummer befindet sich nicht in der Seitentabelle (Seitenfehler)
  - Spezielles Kennungsbit im Seitentabellen-Verzeichnis (Seitentabellenfehler)

Speicherhierarchie

- Daten werden nur zwischen aufeinanderfolgenden Ebenen der Speicherhierarchie kopiert.



Speicherverwaltung (1)

- Das Betriebsystem führt über die freien Speicherbereiche Buch und lagert bei nicht ausreichendem Freiplatz im Hauptspeicher diejenigen Speicherinhalte auf den Hintergrundspeicher aus, die gegenüber ihrem Originalen auf dem Hintergrundspeicher verändert worden sind.

- Die eindeutige Abbildung des großen virtuellen Speichers auf die effektive Hauptspeicherkapazität wird von der Hardware durch Speicherverwaltungseinheiten unterstützt *(memory management units, MMU)*

- Die erforderliche Abbildungsinformation stellt das Betriebs-system in Form einer oder mehrerer Übersetzungstabellen zur Verfügung.

Speicherverwaltung (2)

- Lokalitätseigenschaften von Programmen und Daten:
  - Programme greifen in einem kleinen Zeitintervall auf einen relative kleinen Teil des Adreßraumes zu.

- Die Lokalitätseigenschaften gewährleisten eine hohe Wahrscheinlichkeit, daß die Daten, welche die CPU anfordert, im physikalischen Hauptspeicher zu finden sind.

- Zwei Arten der Lokalität:
  - Zeitliche Lokalität:
    - Falls ein Datum oder ein Befehl referenziert wird, so werden sie bald wieder referenziert.
  - Örtliche Lokalität:
    - Falls ein Datum oder ein Befehl referenziert wird, werden bald Daten oder Befehle mit benachbarten Adressen referenziert.

Schutzmechanismen (1)

- Moderne Mikroprozessoren bieten Schutzmechanismen an, um während der Laufzeit von Programmen unerlaubte Speicherzugriffe zu verhindern.

- Dies geschieht im wesentlichen durch:
  - Trennung der Systemsoftware, z.B. des Betriebsystems
    Insbesondere des Ein-/Ausgabe-Subsystem (BIOS, basic I/O system), von den Anwendungsprozessen.

  - Trennung der Anwendungsprozessen voneinander
    Ist dies gewährleistet, könnte ein fehlerhaftes Anwenderprogramm andere, fehlerfreie Programme beeinflussen (Schutzebenen und Zugriffsrechte)

Schutzmechanismen (2)