LUDWIG-
MAXIMILIANS-
UNIVERSITY
MUNICH

DEPARTMENT
INSTITUTE FOR
INFORMATICS

DATABASE
SYSTEMS
GROUP

Skript zur Vorlesung:

Einführung in die Informatik: Systeme und Anwendungen

Sommersemester 2013

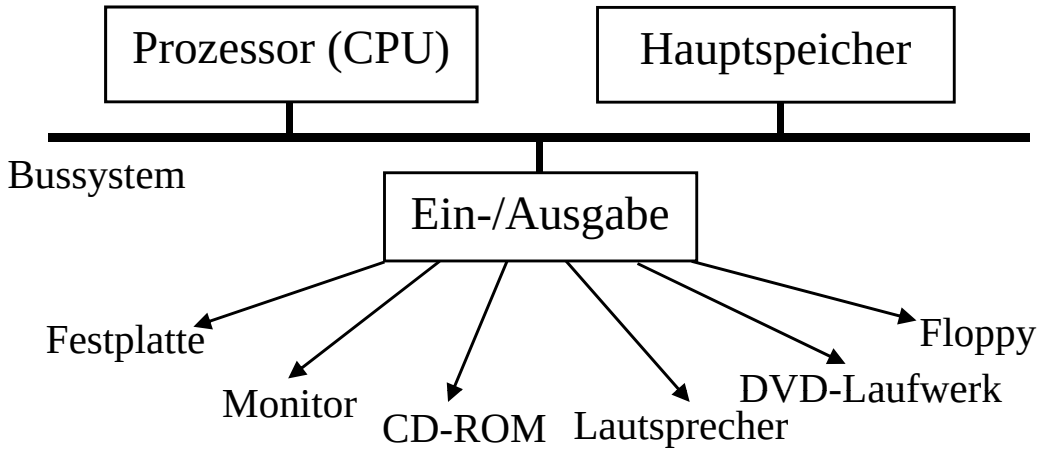
Kapitel 2: Betriebssysteme

Vorlesung: PD Dr. Peer Kröger
Übungen: Johannes Niedermayer

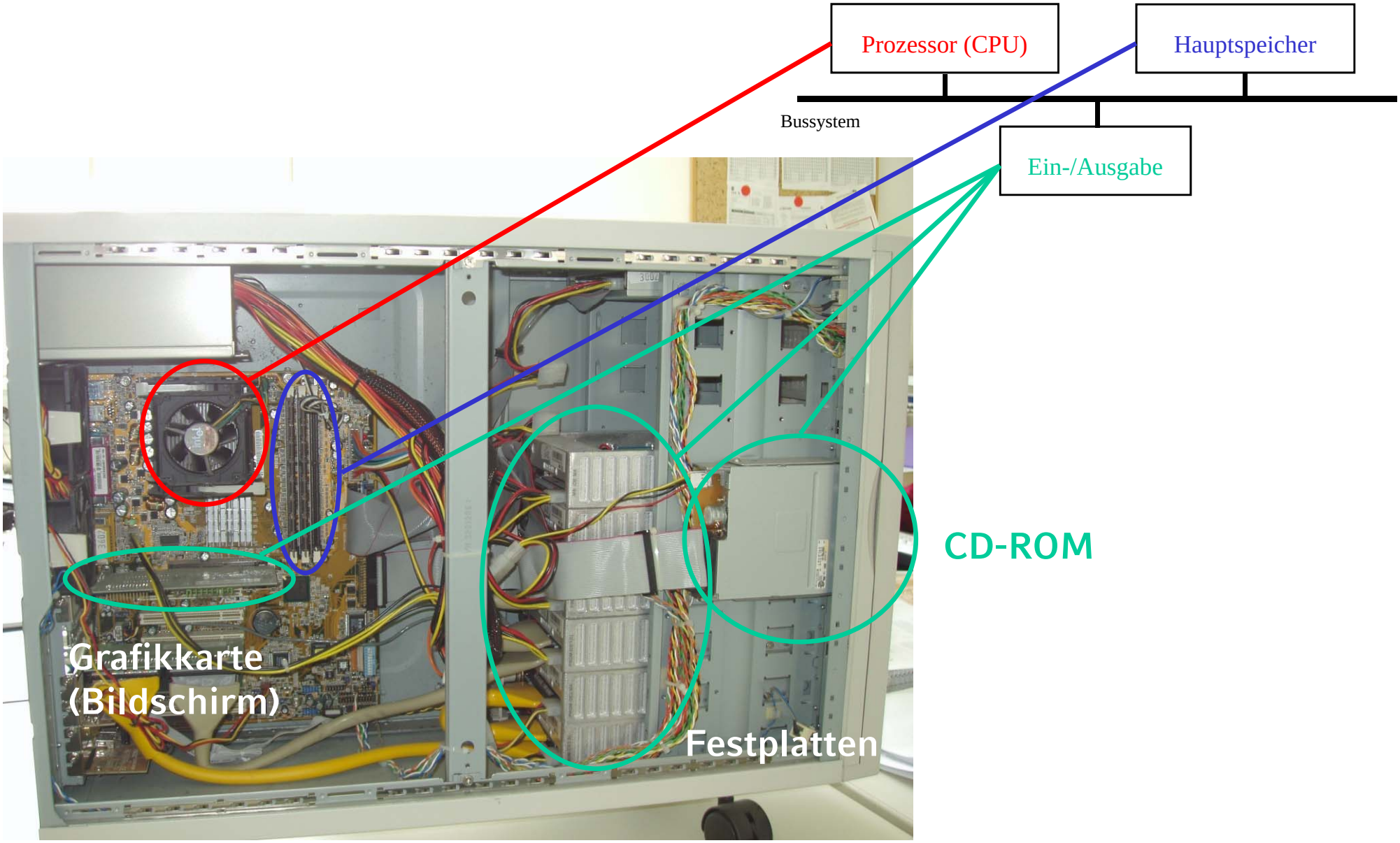
http://www.dbs.ifi.lmu.de/cms/ Einführung_in_die_Informatik_Systeme_und_Anwendungen

2.1 Rechner und Programme

- Komponenten eines Rechners (von-Neumann-Architektur)/Hardware
  - CPU (Prozessor)
    - Ausführung von Befehle und Ablaufsteuerung
  - Speicher (Hauptspeicher)
    - Ablegen von Daten und Programmen, binär codiert
  - Ein-/Ausgabe-Einheiten
    - Ein- und Ausgabe von Daten und Programmen
    - Bildschirm, Tastatur, Drucker, Festplatte, …
  - Busse
    - Informationsübertragung zwischen diesen Einheiten



2.1 Rechner und Programme



2.1 Rechner und Programme

- Warum Betriebssysteme (Teil 1)?
  - CPU arbeitet schneller als Hauptspeicher
    - Beim Laden von Daten aus dem Hauptspeicher muss die CPU warten
  - Hauptspeicher arbeitet schneller als Hintergrundspeicher
    - Bei I/O-Operation (Laden von Daten, Drucken, CD-Brennen, etc.) muss CPU warten

    ⇒ CPU als wichtigste Ressource ist nicht ausgelastet!!!

    („von-Neumann-Flaschenhals“)
  - Lösung: Prozesskonzept
    - Der Rechner führt mehrere „Arbeiten“ (Prozesse) quasi gleichzeitig aus
    - Wenn die CPU für die Bearbeitung des einen Prozesses wartet (z.B. auf eine I/O-Operation) kann die CPU mit der Bearbeitung eines anderen Prozesses fortfahren
    - Das Betriebssystem muss die Verwaltung der Prozesse für den Benutzer transparent organisieren

2.1 Rechner und Programme

- Warum Betriebssysteme (Teil 2)?
  - Ausführung von Programmen auf Rechnern
    - Zentrale Verarbeitungsschritte als „Maschinenanweisungen“ („Mikroprogramm“, ca. 50 – 300 Anweisungen je nach Hersteller)
      - Lesen/Schreiben einer Speicherzelle
      - Einfache Arithmetik
      - etc.
    - Daten sind maschinennah (meist binär) repräsentiert
  - Darstellung von Programmen:
    - Direkte Programmierung der Hardware mit „maschinennaher“ Programmiersprache (Beispiel Fahrtkosten-Algorithmus)

```
pushl   %ebp
movl    %esp, %ebp
movl    16(%ebp), %eax
subl    8(%ebp), %eax
leal    (%eax,%eax,2), %edx
...
```

2.1 Rechner und Programme

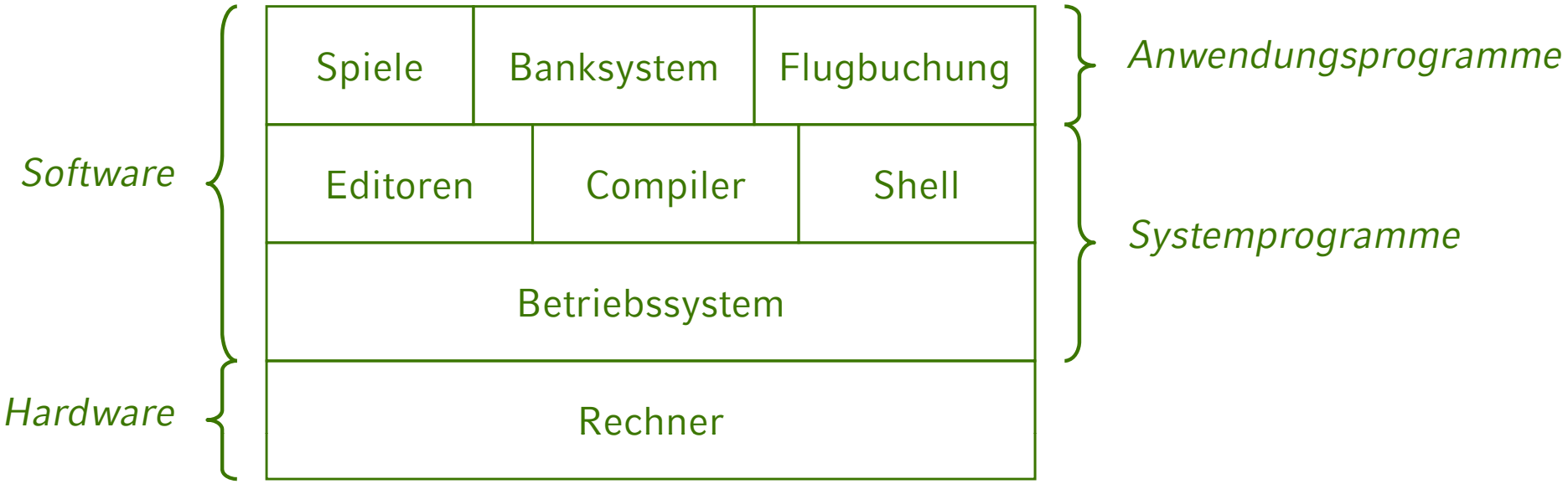
- Programmierung muss sich auch um die einzelnen Hardware-Komponenten kümmern, die benutzt werden sollen
- Beispiel: Laden einer Datei von Festplatte
  1. [optional] starten des Laufwerkmotors
  2. Positionierung des Lesekopfs
  3. Sektorenweise Einlesen des Directory
  4. Suchen der Dateiinformationen (Dateianfang) im Directory
  5. Positionierung des Lesekopfs
  6. Teil einlesen, Verknüpfung zum nächsten Teil erkennen und weiter mit Schritt 5 solange Ende der Datei noch nicht erreicht

  ⇒ *Für jeden Befehl an den Plattenkontroller werden die Adresse der Spur, die Adresse der Hauptspeicherzelle (Ziel), die Menge der zu übertragenden Daten, der Befehlscode (Lesen/Schreiben), etc. benötigt*

– Problem:
  - Programm ist sehr unübersichtlich und für Menschen schwer zu verstehen
  - Hardware ist ebenfalls sehr komplex und besteht aus vielfältigen Komponenten, deren Realisierungsdetails ebenfalls schwer für den Menschen zu verstehen sind

2.1 Rechner und Programme

- Lösung: Prinzip der „Software-Schichtung“
  - BS bildet Schnittstelle für Anwendungsprogramme und spezielle Systemprogramme zur Hardware
    - ⇒ BS bewahrt den Nutzer vor der Komplexität der HW
    - ⇒ ermöglicht indirekt das Nutzen von Programmiersprachen, die für den Menschen leichter zu verstehen/benutzen sind
  - BS bildet die SW-Schicht, die alle Teile des Systems verwaltet und auf dem Anwendungen einfacher zu programmieren sind



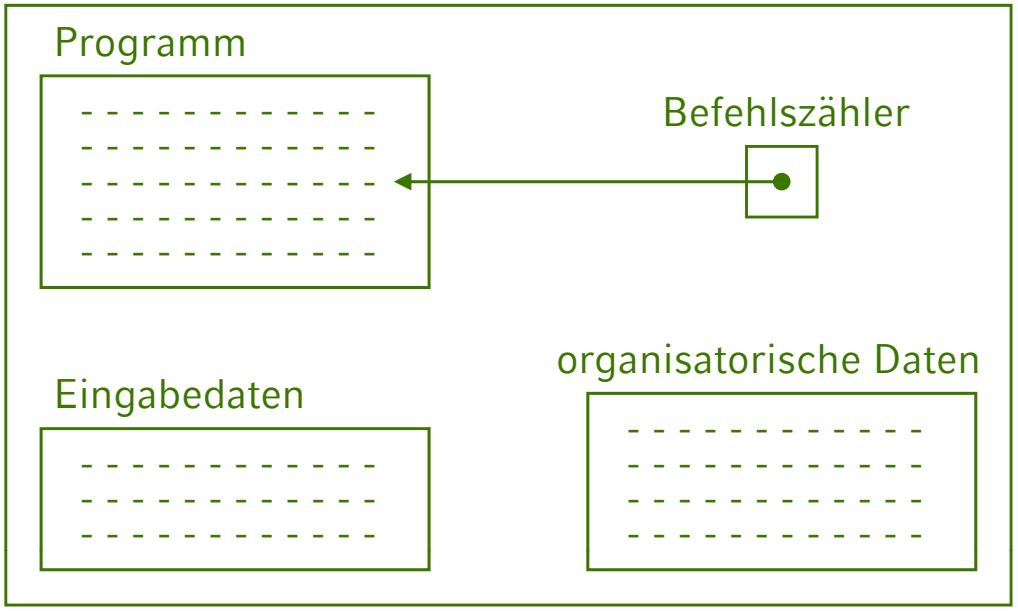
2.1 Rechner und Programme

- Aufgaben des Betriebssystems
  - Schnittstelle für Anwendungsprogramme zur HW
  - Steuerung und Verwaltung von Computersystemen
    - ***Prozessverwaltung***: Steuerung der Ausführung eines oder mehreren Prozesse, insbesondere im Mehrprogrammbetrieb
    - ***Speicherverwaltung*** für den Hauptspeicher
    - ***Dateiverwaltung*** (im Hintergrundspeicher)
    - ***Verwaltung*** der E/A-Geräte
- Einige bekannte Betriebssysteme
  - Windows NT, 2000, XP, Vista, Mobile, 7, 8, …
  - Apple OS X
  - LINUX (versch. Distributionen)
  - Weitere Unix-Varianten wie Solaris, HP/UX, Android, …
  - MVS, VM/SP, CMS, BS 2000 (alle für Großrechner)

2.2 Prozesse

- Auf einem Rechner laufen „gleichzeitig“ verschiedene (Anwender- und System-) Programme
- ***Prozess*:**

  Ein Prozess ist ein in Ausführung befindliches Programm. Dies umfasst:
  - Befehlszähler (Programmzähler) bestimmt den als nächstes auszuführenden Befehl
  - Programmtext
  - Eingabedaten
  - Organisatorische Daten



2.2 Prozesse

- Multiprogramming:
  - BS kann mehrere Prozesse gleichzeitig ausführen, d.h. auch ein Programm kann mehrmals gleichzeitig ausgeführt werden
  - Warum ist das sinnvoll?
    - Programme benötigen außer CPU meist auch E/A-Geräte
    - E/A-Geräte sind deutlich langsamer als CPU (Flaschenhals)

      $\Rightarrow$ Prozessor muss warten und ist nicht ausgelastet

      Beispiel:

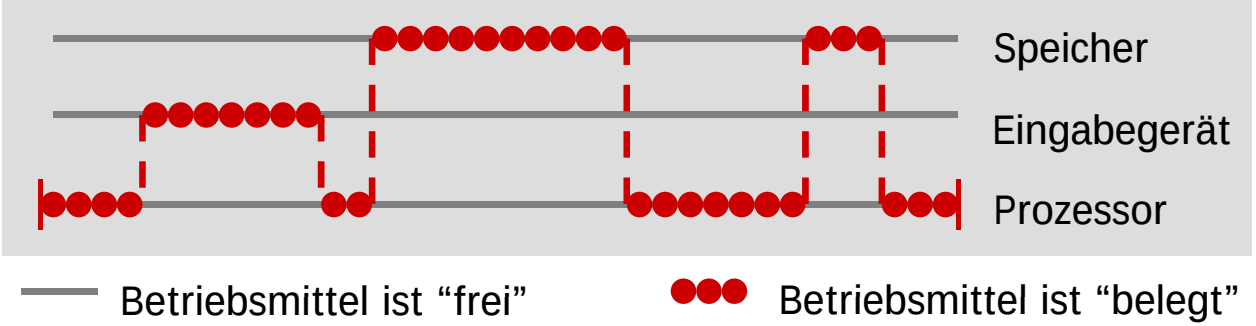
Lesen eines Datensatzes	0,0015 sec.
Ausführen von 100 Befehlen	0,0001 sec.
Schreiben des Datensatzes	0,0015 sec.
	0,0031 sec.

CPU-Auslastung: 0,0001/0,0031 $\approx$ 3,2 %

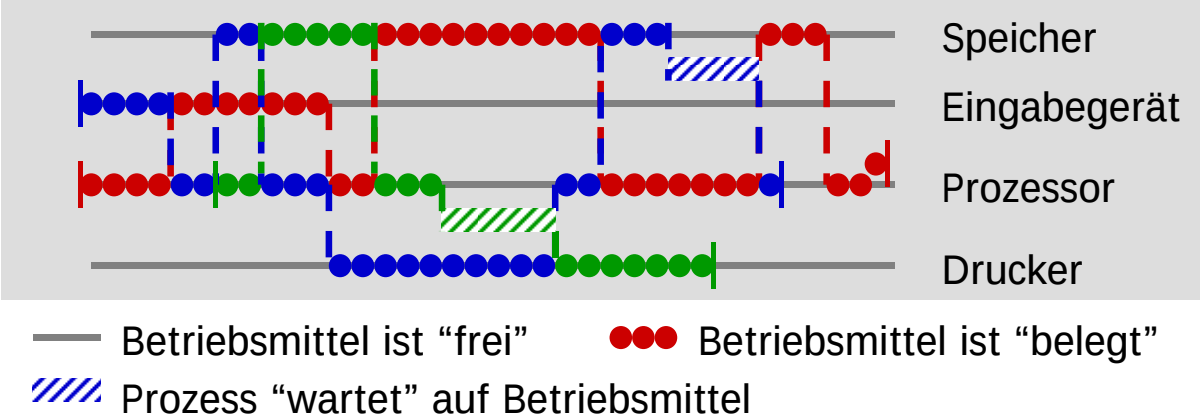
3,2 % der Zeit arbeitet CPU; 96,8 % der Zeit wartet CPU

2.2 Prozesse

- Lösung: Anwendungsprogramme sollen also dem Prozessor abwechselnd zugeteilt werden
  - Einbenutzerbetrieb (***Uniprogramming***) [1 Prozess]



  - Mehrbenutzerbetrieb (***Multiprogramming***) [hier: 3 Prozesse rot/grün/blau]



2.2 Prozesse

- Motivation (cont.)
  - Einzelne Prozesse durchlaufen damit ihre Anweisungsfolge (zumeist) nicht in einem Schritt sondern werden häufig unterbrochen
  - Es entsteht der Eindruck von ***Parallelität*** aller momentan existierenden Prozesse (dies ist allerdings nur eine ***Quasi-Parallelität*** !)
  - Folge:
    - zeitliche Dauer eines Prozesses kann bei verschiedenen Programmausführungen unterschiedlich sein
    - Insbesondere keine *a priori* Aussagen über den zeitlichen Ablauf eines Prozesses möglich
- Rolle des Betriebssystems
  - Aus Sicht des Anwendungsprogramms steht jedem Prozess ein eigener (virtueller) Rechner (CPU aber auch Hauptspeicher, etc.) exklusiv zur Verfügung
  - BS muss Abbildung auf den realen Rechner leisten

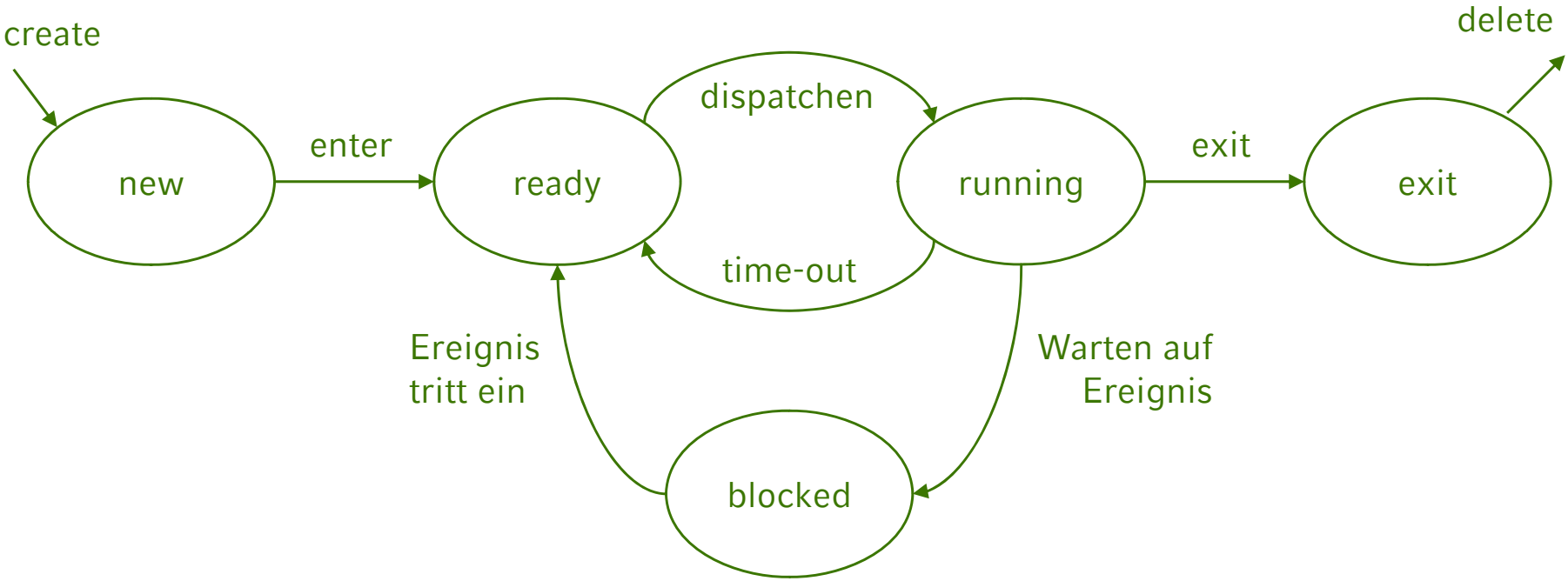
2.2 Prozesse

- Prozesszustände
  - 5-Zustands-Modell

    Prozess ist entweder
    - **new**: Prozess ist erzeugt, aber noch nicht zu der Menge der ausführbaren Prozesse hinzugefügt
    - **ready**: Prozess ist zur Ausführung bereit, aber ein anderer Prozess ist dem Prozessor zugeteilt
    - **running**: Prozess ist dem Prozessor zugeteilt und wird gerade ausgeführt
    - **blocked**: Prozess ist blockiert, d.h. er wartet auf das Eintreten eines externen Ereignisses (Beendigung einer E/A-Operation, benötigtes Betriebsmittel ist belegt, …)
    - **exit**: Prozess wurde beendet, d.h. Ausführung ist abgeschlossen

2.2 Prozesse

– Graphische Darstellung



2.2 Prozesse

- 7-Zustands-Modell
  - bisher: alle Prozesse werden im Hauptspeicher gehalten
  - wenn alle Prozesse blockiert sind (E/A-intensive Prozesse) und der Hauptspeicher voll ist, können keine weiteren Prozesse gestartet werden

    ⇒ Prozessor wäre wieder unbenutzt!!!
  - Lösung 1: Hauptspeicher erweitern

    ⇒ schlecht: kostet Geld, größere Programme?
  - Lösung 2: Prozesse (oder Teile davon) auf den Hintergrundspeicher (Festplatte, „HGS“) auslagern (***Swapping***)

    ⇒ gut: neue Prozesse haben im Hauptspeicher Platz

    ⇒ schlecht: eine zusätzliche E/A-Operation
  - Um übermäßig große Warteschlangen auf Festplatte zu vermeiden, sollte immer ein ausgelagerter Prozess wieder eingelagert werden; dieser sollte aber nicht mehr blockiert sein

    ⇒ unterscheide, ob ausgelagerte Prozesse blockiert oder bereit sind

    ⇒ 2 neue Zustände

2.2 Prozesse

- **ready, suspend**: auf dem HGS ausgelagert, bereit
- **blocked, suspend**: auf dem HGS ausgelagert, wartet auf Ereignis (blockiert)