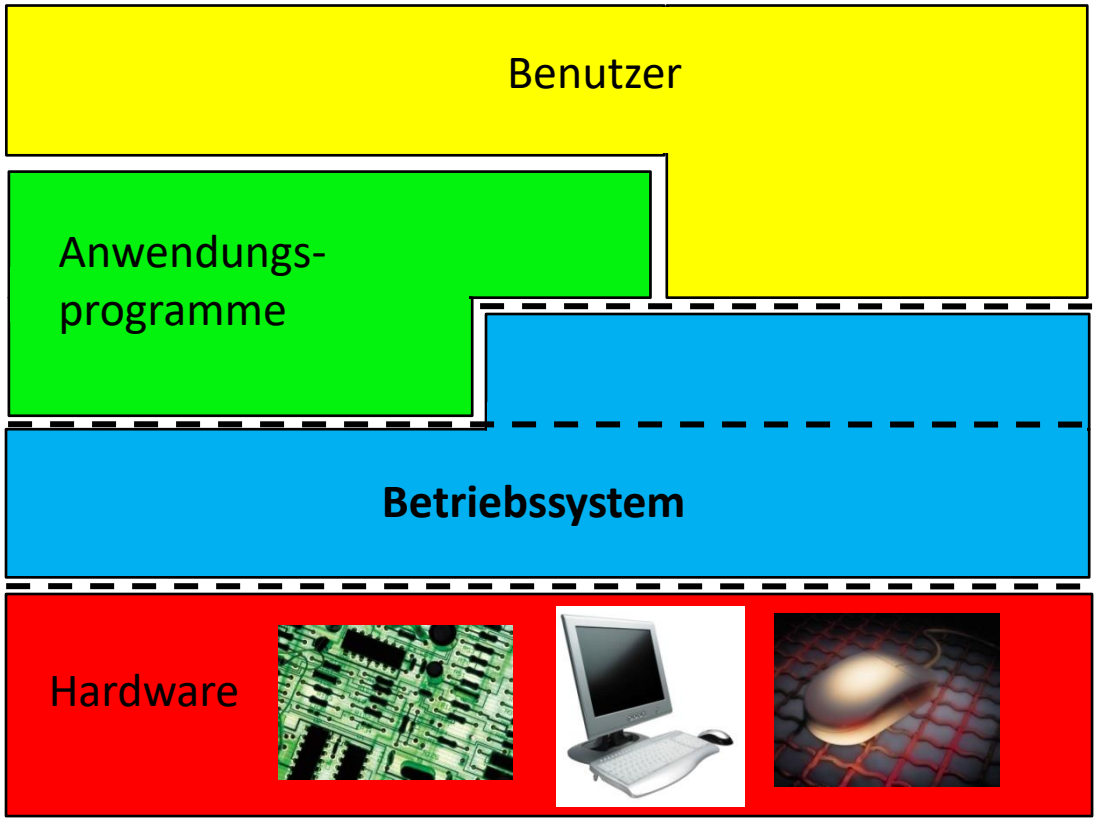
Kapitel 1
Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen

Was ist ein Betriebssystem?



Benutzer-
schnittstelle

System-
schnittstelle

Hardware-
schnittstelle

Ein Betriebssystem

- … bietet Schnittstellen zwischen dem Benutzer, den Anwendungsprogrammen und der Hardware
- … steuert die Ausführung von Programmen

Sichten auf ein Betriebssystem (1)

Benutzer / Programmierer-Sicht: **„Abstrakte Maschine“**

- **Hauptziel: Einfache und komfortable Schnittstellen!**
  - Benutzerschnittstelle
    - Graphiksystem (GUI)
    - Kommandointerpreter mit Skript-Sprache
  - Programmierschnittstelle
    - Hardware-unabhängige Prozeduraufrufe („System Calls“) für alle Betriebssystemfunktionen

Sichten auf ein Betriebssystem (2)

System-Sicht: **„Betriebsmittelverwalter“**

- **Hauptziel: Effiziente Betriebsmittelausnutzung!**
  - Prozessor
  - Speicher
  - Dateien
  - Ein-/Ausgabegeräte
- Ein Betriebssystem verwaltet die Betriebsmittel und teilt sie den Anwenderprogrammen zu

Betriebssystem – Generationen und parallele Hardware-Entwicklungen

- Erste Generation 1945 - 1955
  - Relais, Elektronenröhren // Schalttafeln
- Zweite Generation 1955 - 1965
  - Transistoren // Stapelverarbeitung (“batch systems”)
- Dritte Generation 1965 – 1980
  - VLSI (“Very Large Scale Integration”) // Multiprogramming
- Vierte Generation 1980 – heute
  - Personal Computer // Mobile Computer // Verteilte Systeme

Erste Generation: 1945 - 1955

- Kein Betriebssystem!
  - Nur ein Benutzer und ein Programm gleichzeitig
  - Programmierer = Operator = Benutzer („Open Shop“)
  - Computer werden von einer Konsole bedient
    - Mit Lampen zur Statusanzeige
    - Mit Schaltern für bitweise Eingabe (z.B. Startadresse)
    - mit Eingabegeräten wie Lochstreifenlesern etc.
    - mit Ausgabegeräten wie Drucker etc.
  - Zeitbuchung auf Papier
  - Programmausführung war umständlich
  - Einzige Unterstützung bei der Fehlersuche: Speicherauszug („Core Dump“)

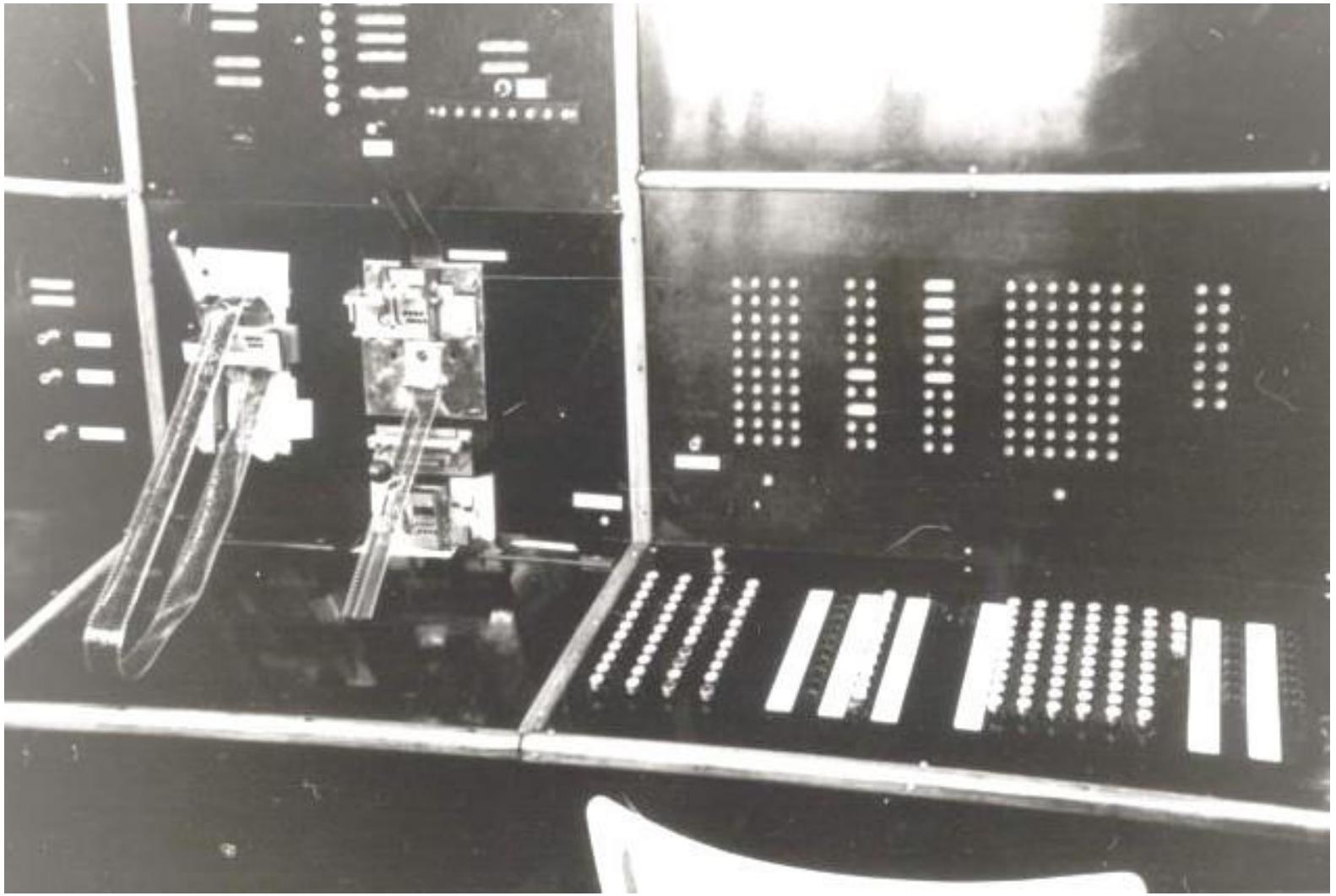
Väter:

John von Neumann (USA - Princeton)

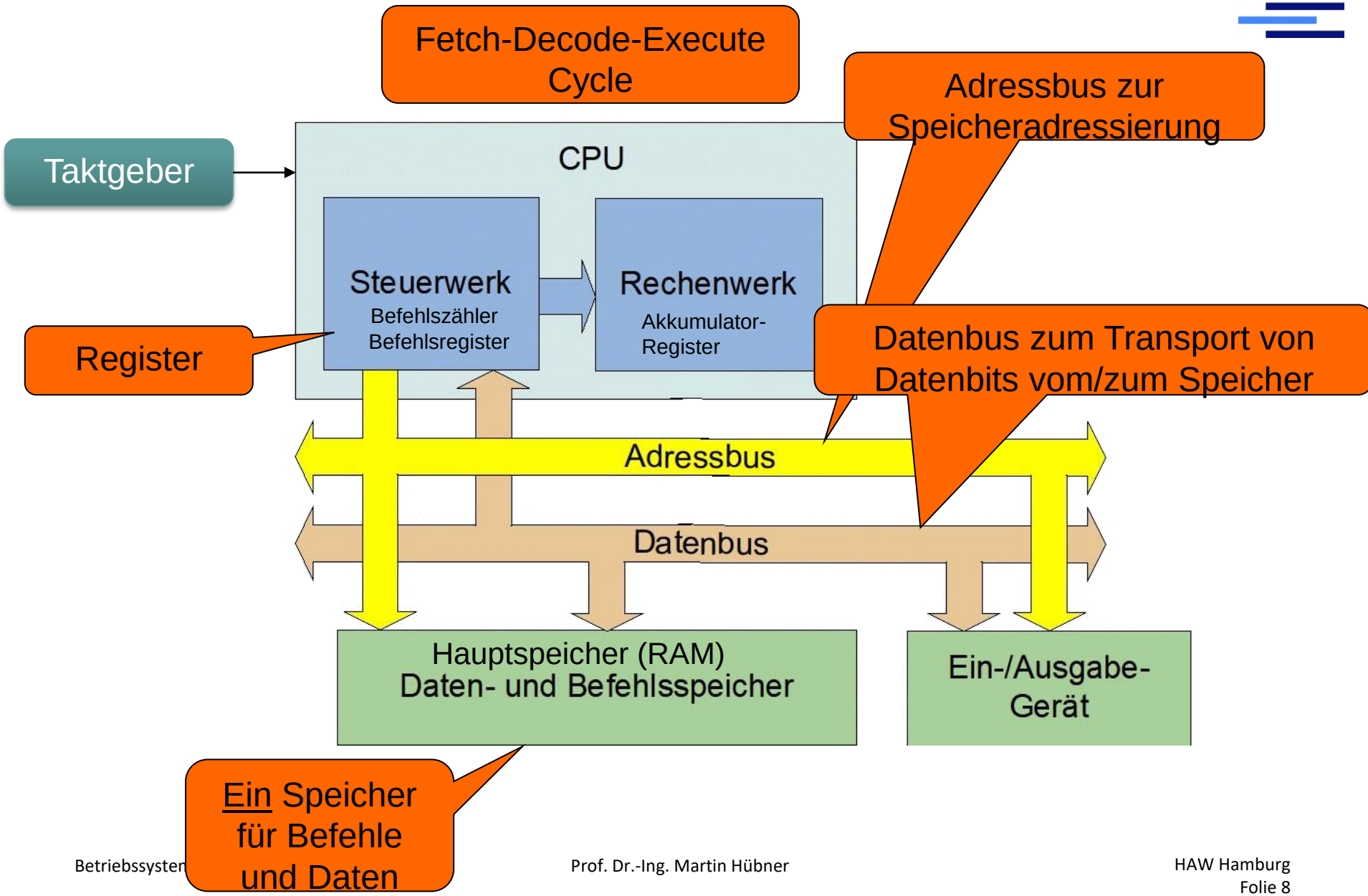
Konrad Zuse (Deutschland)

Howard Aiken (USA - Harvard)

Schalttafel einer Zuse Z4 (1945)

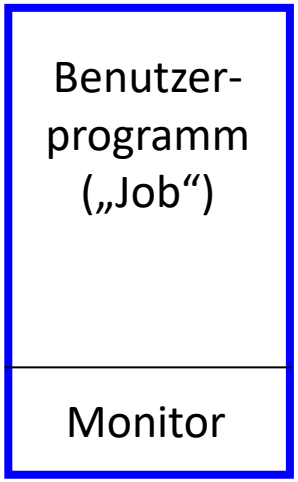


Übersicht: „Von-Neumann-Architektur“



Zweite Generation: 1955 - 1965

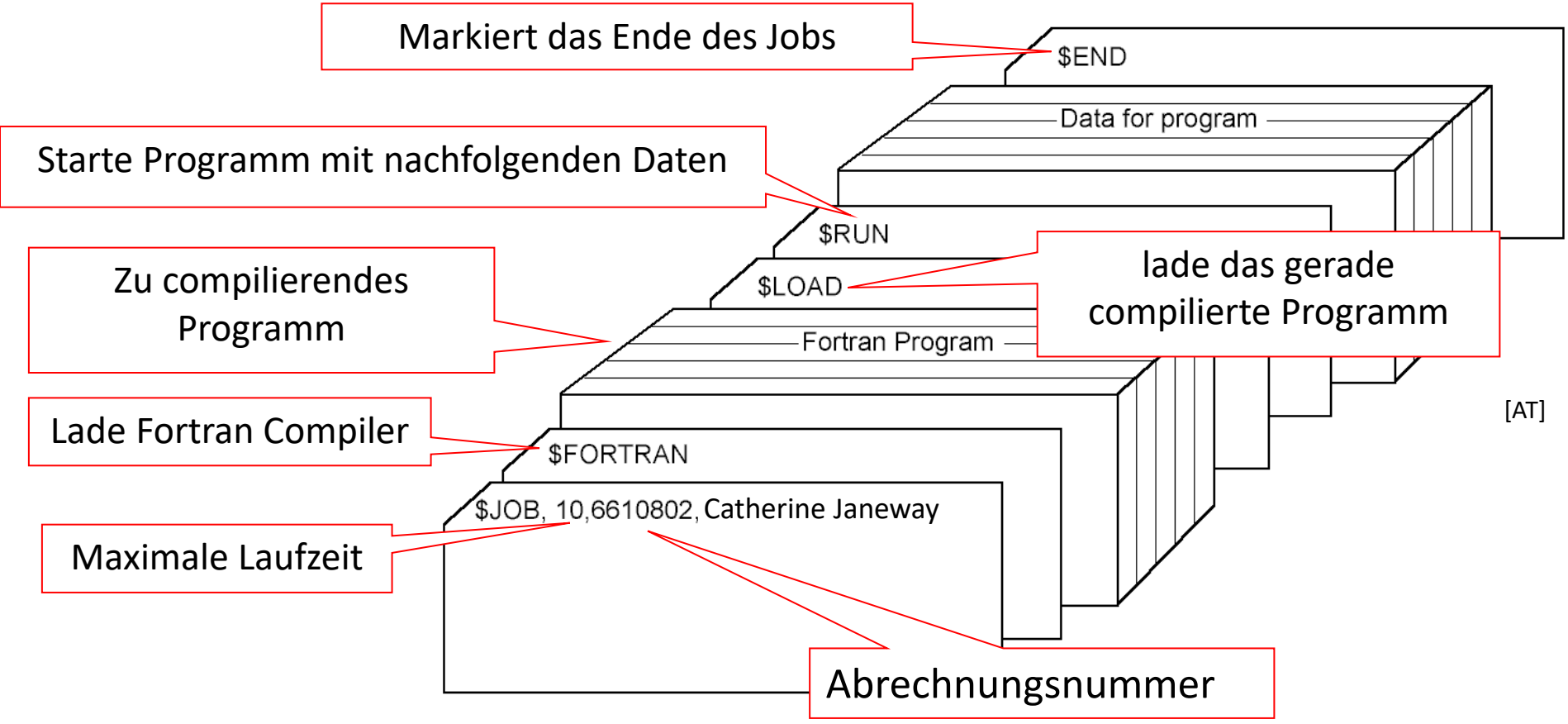
- Einfache Batch-Systeme („Stapelverarbeitung“)
  - Monitor (Betriebssystemvorläufer)
    - Software, die den Ablauf der Programme steuert
    - Der Monitor ist speicherresident, d.h. er ist immer geladen
    - Der Monitor lädt ein Programm und die Eingabedaten in den Hauptspeicher (von Lochkarte / Band) und verzweigt zu dessen Startadresse
    - Nach Programmende / bei Programmfehler wird wieder der Monitor aufgerufen, der das nächste Programm lädt
  - Computer wurden im Rechenzentrum betrieben
    - Zugang für Benutzer gesperrt
    - Benutzer gab sein Programm (auf Lochkarten) ab
    - Benutzer erhielt Ergebnis z.B. durch einen Ausdruck später



Job Control Language (JCL)

- Eine Kommmandosprache
  - entwickelt, um dem Monitor für das gerade aktuelle Programm („Job“) Informationen geben zu können
  - gibt Kommandos an den Monitor:
    - welcher Compiler genutzt werden soll
    - welche Daten genommen werden sollen
    - welche Ein- / Ausgabe gewählt werden soll
    - etc.
  - JCL wurde im Laufe der Entwicklung immer leistungsfähiger

Struktur eines Fortran-Jobs (Lochkarten)



[AT]

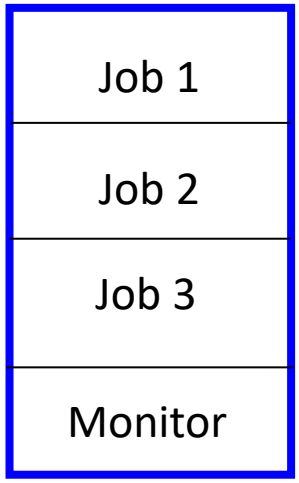
Uniprogramming

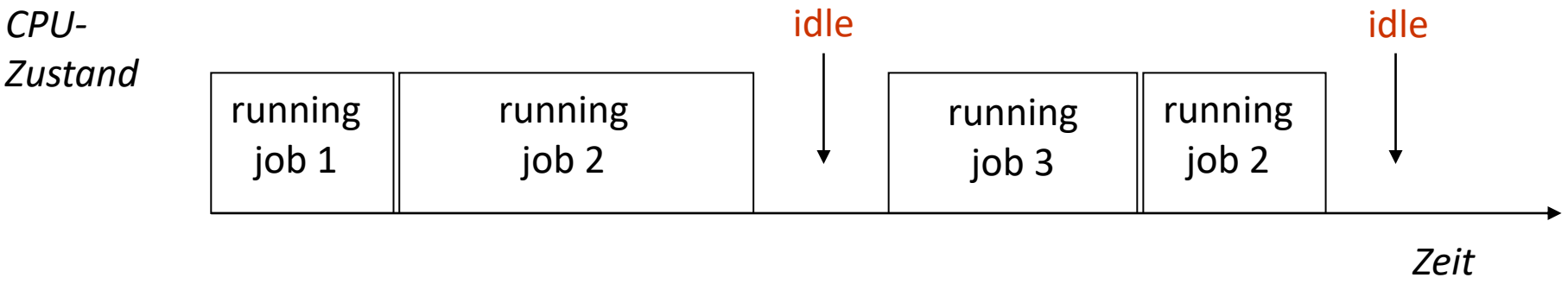
- Nur ein Programm gleichzeitig *(bei billigen Rechnern)*
- Prozessor wartet auf das Ende jeder (langsamen!) I/O-Operation, bevor das Programm fortfahren kann.

(„idle“: im Leerlauf)

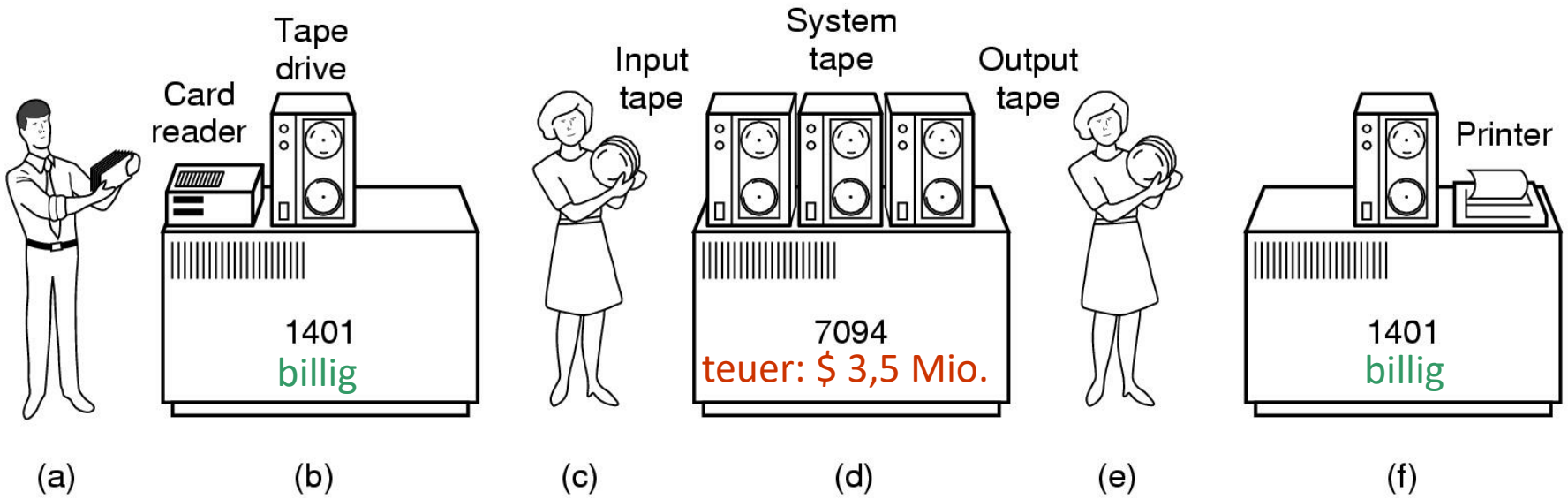
Multiprogramming

- Mehrere Programme (Jobs) sind gleichzeitig im Hauptspeicher *(bei teuren Rechnern)*
- Wenn ein Programm (Job) die CPU „freiwillig“ abgibt und auf I/O wartet, kann der Monitor die CPU auf ein anderes Programm umschalten *(Welches? → Scheduling!)*





Optimierte Stapelverarbeitung



[AT]

- Programmierer bringt Lochkarten zur IBM 1401
- Karteninhalt wird auf Band geschrieben
- Sammeln von Jobs; anschließend Zurückspulen des Bandes
- Band wird in die IBM 7094 eingelegt (Hauptrechner); Jobabarbeitung
- Druckoutput wird wieder auf Band geschrieben
- Ausdruck vom Band über die IBM 1401

Vergleich Kartenleser / Bandstation

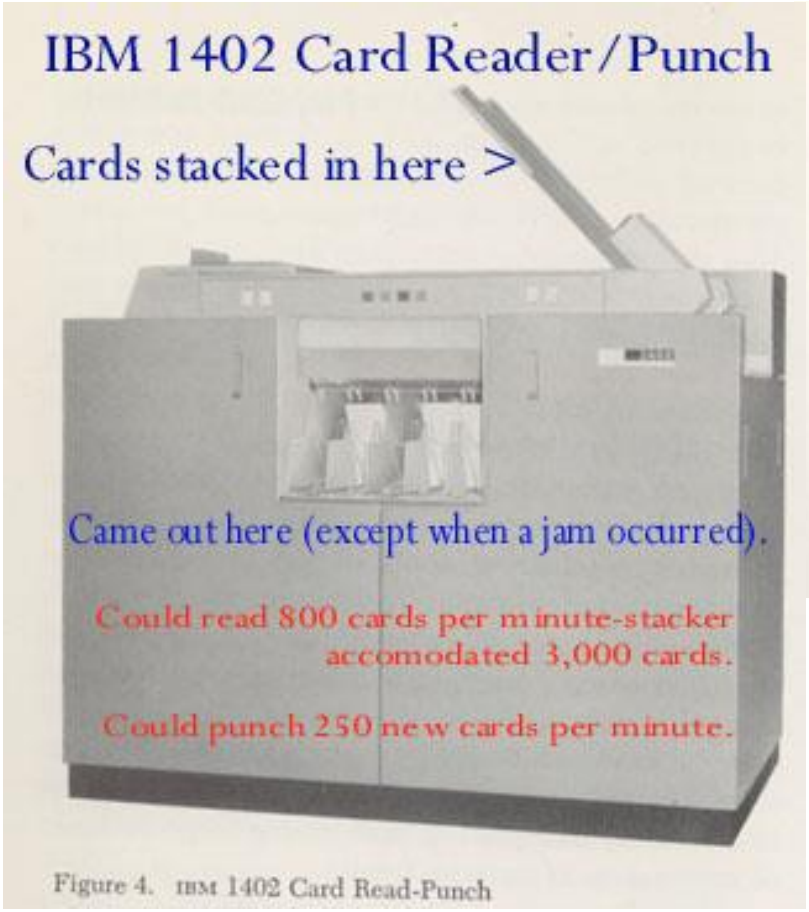


Figure 4. IBM 1402 Card Read-Punch

OPERATING CHARACTERISTICS	729-II	729-IV	7330
Density, Characters Per Inch	200 or 556	200 or 556	200 or 556
Tape Speed, Inches Per Second	75	112.5	36
Inter-Record Gap Size, Inches	3/4	3/4	3/4
Character Rate, Characters Per Second	15,000 or 41,667	22,500 or 62,500	7,200 or 20,016
High Speed Rewind, Minutes	1.2	.9	2.2
Regular Rewind, Inches Per Second	75	112.5	36

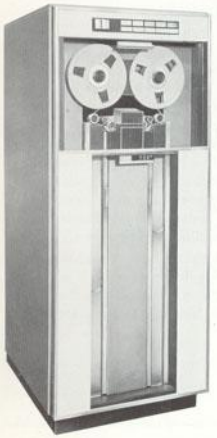


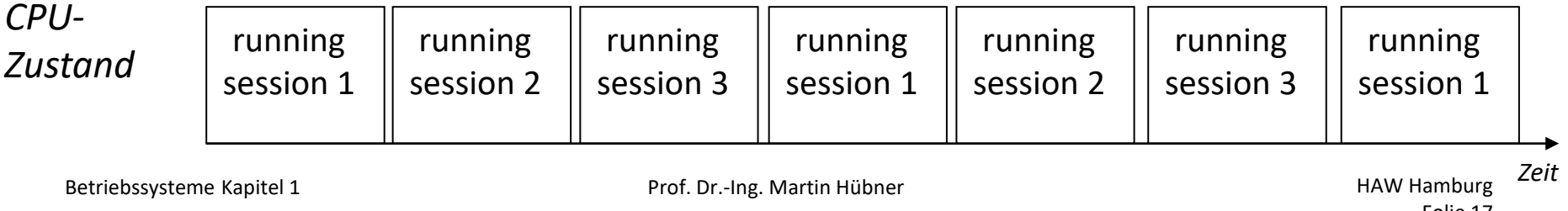
Figure 97. IBM 729 Magnetic Tape Unit

Dritte Generation: 1965 - 1980

- Die Hardware-Hersteller (IBM: OS/360) lernten schnell, dass folgende Hardware-Eigenschaften für ein stabiles, sauber strukturiertes Betriebssystem notwendig sind:
  - Speicherschutz
    - Ein Benutzerprogramm darf nur seinen eigenen Speicherbereich verändern
  - Schutz vor illegalen Operationen
    - Nur der Monitor soll z.B. I/O-Befehle ausführen können
  - Timer / Interrupts
    - Ein **Timer** (Zeitschaltuhr) ...
      - ist ein eigener Hardware-Baustein, der bei Ablauf der eingestellen Zeit einen **Interrupt** (Unterbrechung des laufenden Programms) auslösen kann
      - wird vom Monitor bei Start eines Benutzer-Programms gestartet
      - unterbricht bei Ablauf das laufende Benutzer-Programm und ruft den Monitor wieder auf
      - verhindert, das ein Benutzer-Programm die CPU nicht wieder abgibt (d.h. den Rechner monopolisiert)

Time Sharing

- Wunsch nach Interaktion mit dem Computer führt zur Entwicklung von **Time-Sharing Systemen**
- Benutzt Multiprogramming, um viele interaktive Jobs („Sessions“) zu verarbeiten
- Die CPU-Zeit wird zwischen den vielen Programmen / Benutzern aufgeteilt:
  - Automatische Unterbrechung eines Programms nach Ablauf einer (kurzen) „Zeitscheibe“ (durch Timer und Interrupt)
- Die Benutzer haben durch viele Bildschirme gleichzeitig Zugriff auf das System
- Jeder Benutzer kann wie an einem Einplatz-System arbeiten



Vergleich zwischen Batch Multiprogramming und Time Sharing

- Mit den unterschiedlichen Betriebsarten werden unterschiedliche Ziele verfolgt :

	Batch Multiprogramming	Time Sharing
Prinzipielles Ziel	**Maximierung der CPU-Auslastung (Durchsatz)**	**Minimierung der Antwortzeit**
Anweisungen kommen von:	JCL – Elementen, die jedem Programm beigefügt sind	Kommandos, die am Bildschirm eingegeben werden
Unterbrechung wegen:	I/O-Anforderung	Ablauf der Zeitscheibe

Vierte Generation 1980 – heute

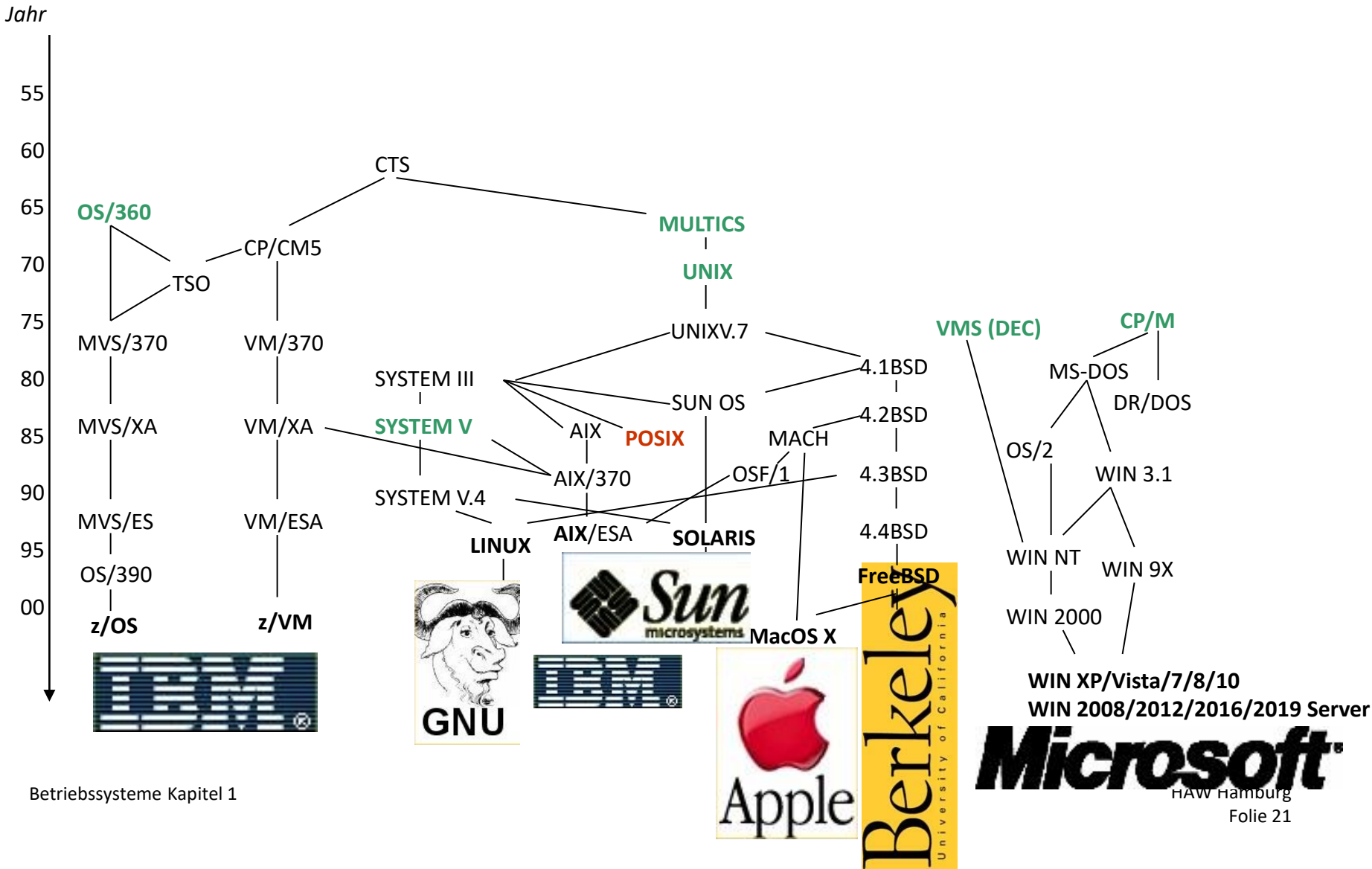
- Multiprozessorsysteme
  - Meist Symmetrisches Multiprocessing (SMP):
    Der BS-Code ist nur einmal im Speicher, doch jede CPU kann ihn ausführen (→ Intelligentes Sperren nötig!)
- Multicomputersysteme
  - Cluster
  - Storage Area Networks
- Mobile Systeme
  - Notebook / Tablet / Smartphone
- Verteilte Systeme
  - Client-/Server-Systeme
  - Netzbetriebssysteme
  - Verzeichnisdienste
  - …

➔ Bauen auf den Erfahrungen und Prinzipien der 1.-3. Generation auf!

Aktuelle Betriebssystem - Typen

- Mainframe-Betriebssystem
  - Sehr hohe I/O-Kapazität, sehr viele parallele Programme *(IBM z/OS)*
- Server-Betriebssystem
  - Hohe I/O-Kapazität, User-Interface nicht wichtig *(Windows Server, Unix)*
- Echtzeit-Betriebssystem
  - Zeitkritische Anwendungen, z.B. Anlagensteuerung *(VxWorks, QNX)*
- Desktop-Betriebssystem
  - Workstation, Notebook *(Windows, Linux, Mac OS-X)*
- Handheld-Betriebssystem
  - Smartphone, Tablet *(Android [Linux], iPhone iOS, ~~Windows Phone~~)* – z.T. „abgespeckte“ Desktop-Systeme
- Eingebettetes Betriebssystem („Embedded System“)
  - BS in technischen Geräten, z.B. Waschmaschinen, Fernsehern, Chipkarten

Betriebssysteme – Entwicklung (historisch)



Zusammenfassung Abschnitt 1: Was ist ein Betriebssystem?

- Schnittstelle zwischen Hardware und Applikationen
- Mögliche Sichten:
  - „Virtuelle Maschine“
  - Betriebsmittelverwalter
- Vier Generationen (1945 – heute)
- Permanente Orientierung an den Möglichkeiten der Hardware
- Die modernen Konzepte haben sich im Lauf der Zeit entwickelt

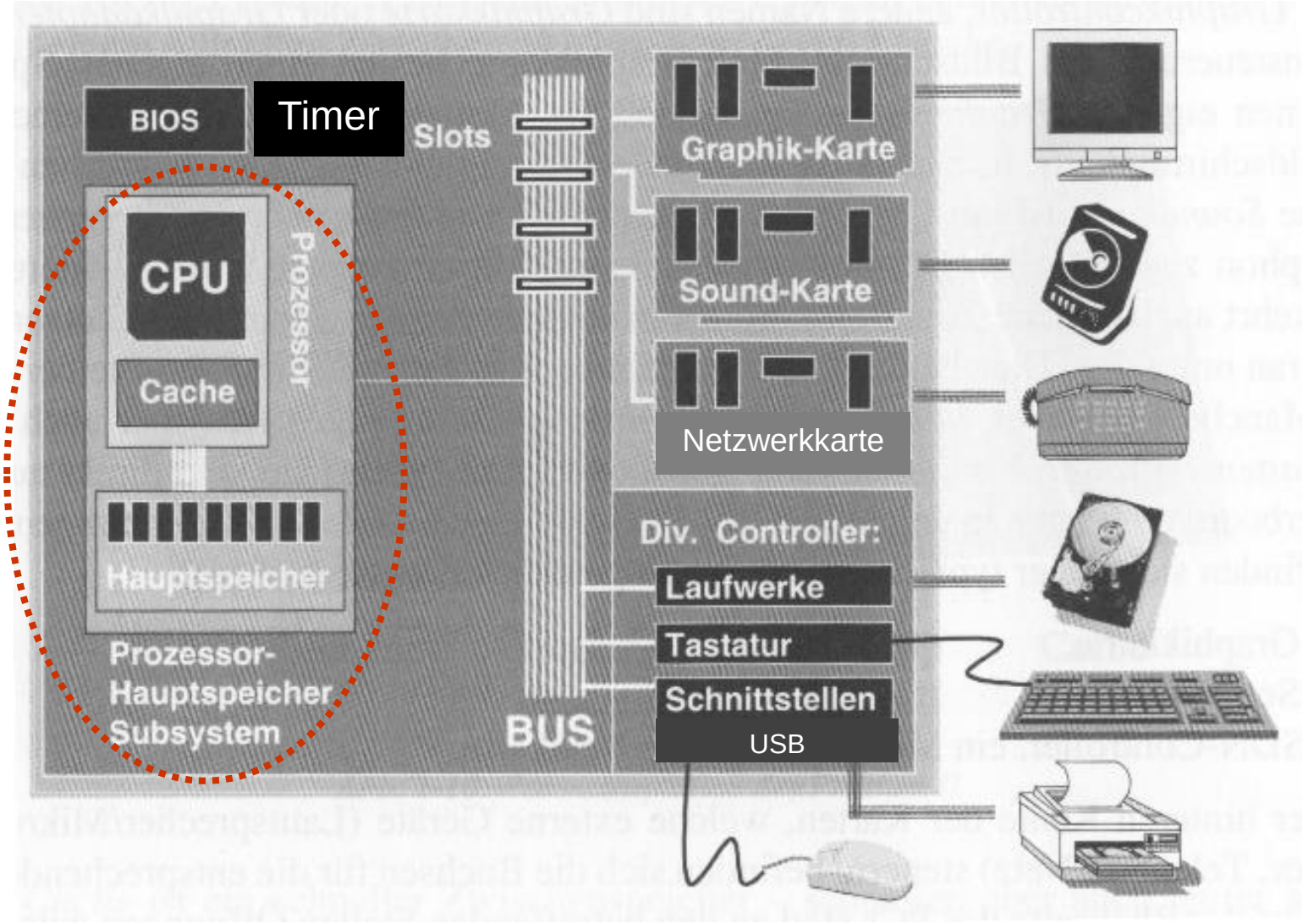
Kapitel 1
Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen

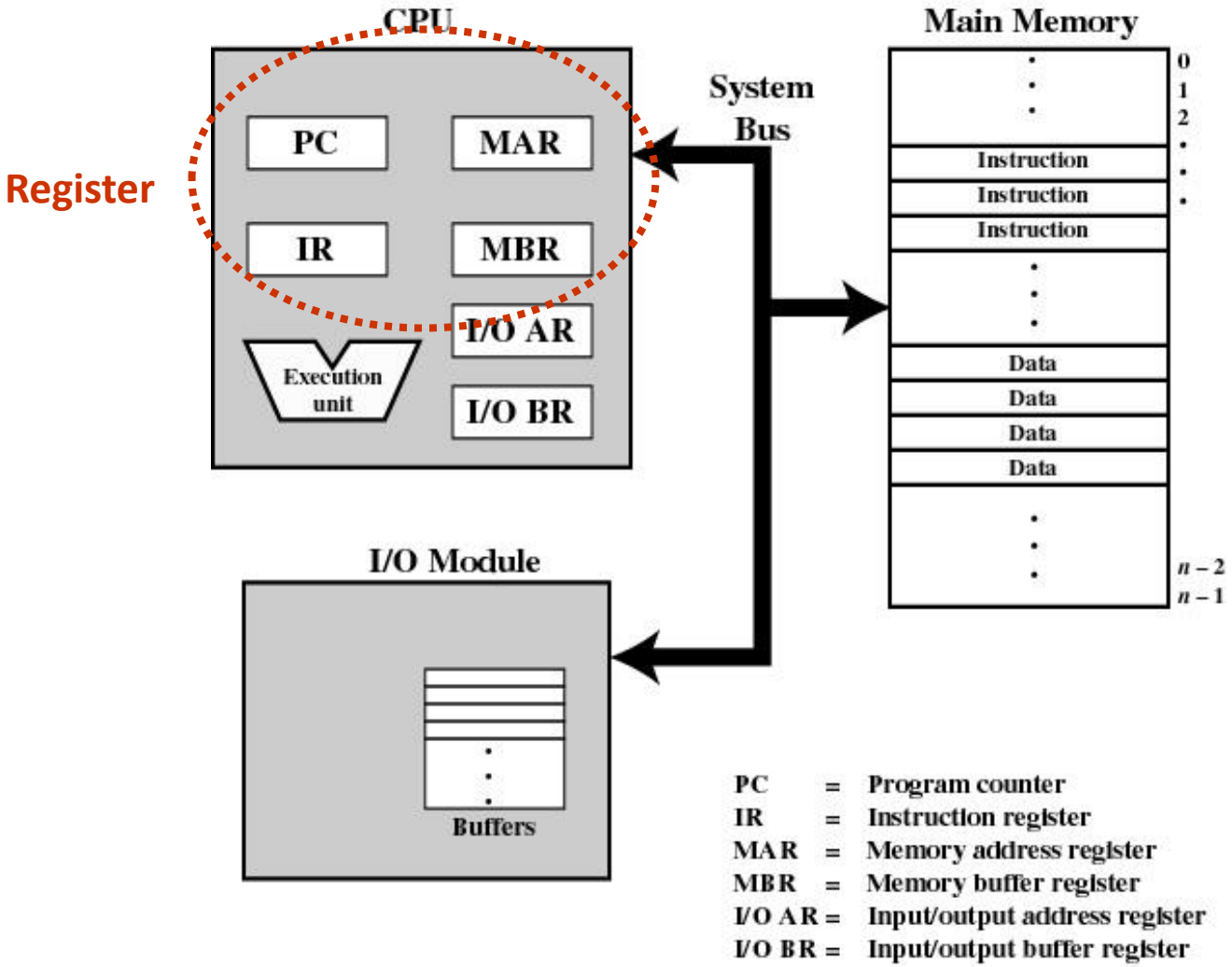
Kap. 1 Abschnitt 2
Welche Hardware-Konzepte sind für Betriebssysteme wichtig?

- Prozessor
  - Kontroll- und Statusregister
  - Befehlsablaufzyklus
  - Interrupts
- Speicher
  - Speicherhierarchie
  - Cache-Prinzip
- Schutzmechanismen
  - Privilegierte Instruktionen
  - I/O-Zugriffsschutz
  - Speicherschutz
  - CPU-Schutz

Komponenten eines Computers



Grundprinzip des Universalrechners

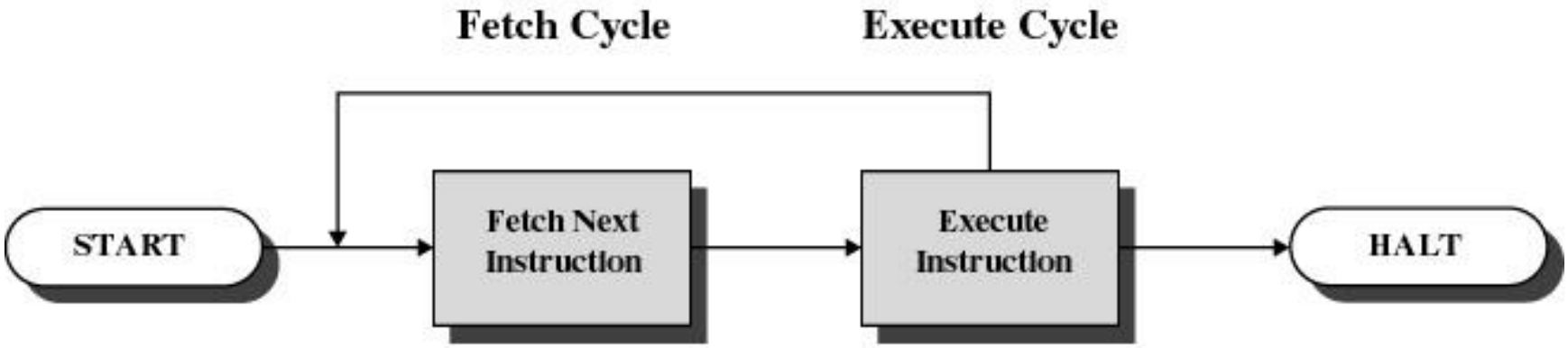


[WS]

Prozessor-Register (1)

- Befehlszähler (**PC - Program Counter**)
  - enthält die Hauptspeicher-Adresse des Befehls, der ausgeführt werden soll
- Speicheradressregister (**MAR – Memory Address Register**)
  - Hauptspeicheradresse des nächsten zu holenden Datenworts (für Adressbus)
- Speicherpufferregister (**MBR – Memory Buffer Register**)
  - Nimmt das nächste zu holende Datenwort vom Datenbus
- Befehlsregister (**IR - Instruction Register**)
  - enthält den Befehl, der ausgeführt werden soll

Befehlsablaufzyklus (vereinfacht)



[WS]

- Die CPU lädt den nächsten Programmbefehl aus dem Hauptspeicher (über PC → MAR → MBR) in das Befehlsregister IR (**Fetch**).
- Anschließend wird der Befehl ausgeführt (**Execute**).
- Der Befehlszähler PC wird nach jedem ausgeführten Befehl aktualisiert (meist hochgezählt).

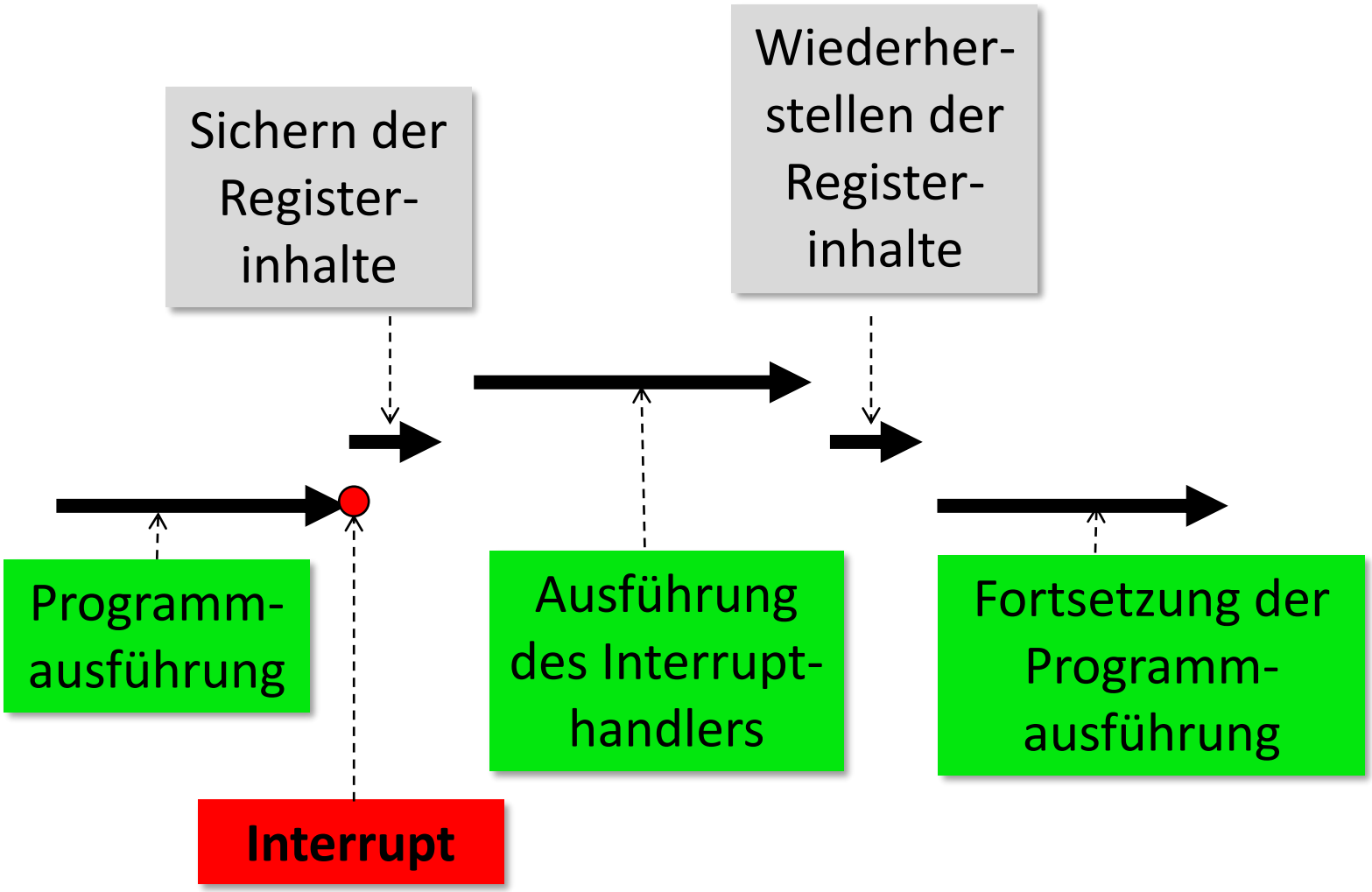
Prozessor-Register (2)

- Ein oder mehrere Statusregister (PSW – Program Status Word) mit einzelnen Bits für
  - Condition Codes: Flags - Ergebnisanzeigen
  - **Kernel / User – Modusbit**
    Usermodus = eingeschränkter Zugriff, nicht alle Instruktionen stehen zur Verfügung!
  - **Interrupt-Bit:** Interrupts freigegeben / gesperrt?
    (enabled / disabled?)
- User – Register
  - sind Arbeitsregister für Benutzerprogramme (wie temporäre Variablen)
  - reduzieren die Zugriffe auf Hauptspeicher

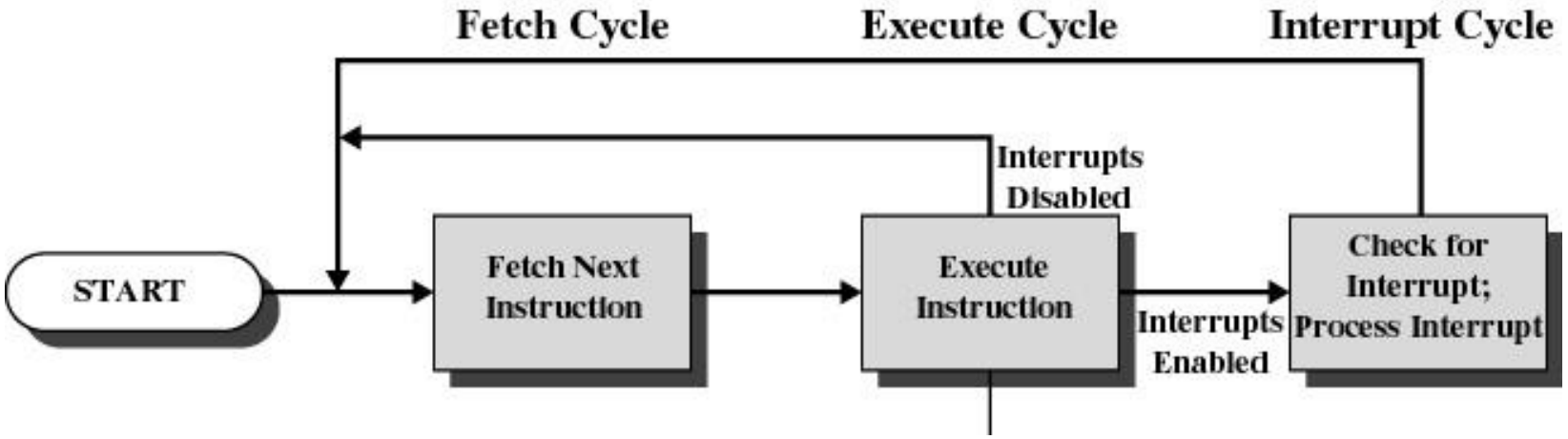
Interrupts („Unterbrechungen“)

- Interrupt = Unterbrechung des Programmablaufs
- Unterbrechungen werden durch äußere Ereignisse (asynchron) oder durch das Programm selbst verursacht (synchron)
- Die Unterbrechung des Programms erfolgt so, dass es später fortgesetzt werden kann (Sichern aller Register-Inhalte)
- Typischerweise werden in der Unterbrechung kurze Programmteile ausgeführt, die als Reaktion auf das Ereignis notwendig sind: Interrupt-Handler *(auch Interrupt Service Routine genannt)*
- Das BS verwaltet eine Interrupttabelle, in der die Startadressen aller Interrupt-Handler aufgelistet sind.

Zeitlicher Ablauf einer Interruptbehandlung



Befehlsablaufzyklus mit HW-Interrupt-Behandlung



- Interrupts disabled / enabled: → PSW Interrupt-Statusbit
- Check for Interrupt: Test, ob Signal an Interrupt-Eingangsleitung (Bus) anliegt
- Process Interrupt:
  - Wenn ein Interruptsignal anliegt, wird der entsprechende Bitvektor („Interrupt-Vektor“) vom Bus gelesen
  - Der Interrupt-Vektor wird als Codezahl interpretiert und dient als **Index** in der Interrupttabelle, aus der die Startadresse des Interrupt-Handlers gelesen wird
  - Der Wert des PC-Registers ("Befehlszähler") wird auf die Startadresse des Interrupt-Handlers gesetzt

Arten von Interrupts

- **Asynchroner Interrupt**
  - **Hardware Interrupt**
    Ein Hardwarebaustein oder ein Peripheriegerät löst einen Interrupt aus. Man unterscheidet zwei Arten: **NMI** (Non Maskable Interrupt – kann nicht *disabled* werden) und **IRQ** (Interrupt Request).
    z.B.: Timer, Reset, Serielle Schnittstelle, Festplatte
- **Synchroner Interrupt („Trap“)**
  - **Exception**
    Aufgrund eines Programmfehlers erzeugt die CPU einen Interrupt (z.B. Overflow, Division durch 0, ungültiger Operationscode, Zugriff auf einen privilegierten Befehl im User-Modus, ...)
  - **Software Interrupt**
    Über SW-Interrupt werden Dienste des BS angefordert (System Call). Der Interrupt wird gezielt durch den **Trap** - Befehl im Programm ausgelöst.

Schachtelung von Interruptbehandlungen

- Während der Behandlung eines Interrupts können weitere Interrupts auftreten
- Nur wenn ein Interrupt mit einer höheren Priorität (hier: höhere Zahl) auftritt, wird die aktuelle Interruptbehandlung unterbrochen!
- Speicherung der neu aufgetretenen Interrupts (kleinere oder gleiche Priorität) im Interrupt-Speicherregister
  - Pro Interrupt-Priorität ein Bit (Flag)
    - ➔ ein weiterer Interrupt derselben Priorität kann nicht mehr gespeichert werden und geht daher bei Ankunft verloren!

Beispiel: Schachtelung von Interrupts

Ereignis	Behandelter Interrupt	Interrupt-Speicherregister (I_1,I_2,I_3)
Benutzerprog. läuft	-	0 0 0
I_3 tritt ein	I_3	0 0 0
I_2 tritt ein	I_3	0 1 0
I_1 tritt ein	I_3	1 1 0
I_3 beendet	I_2	1 0 0
I_1 (neu) tritt ein und geht verloren	I_2	1 0 0
I_2 beendet	I_1	0 0 0
I_1 beendet (Benutzerprog. läuft weiter)	-	0 0 0

nach [CV]

Kap. 1 Abschnitt 2
Welche Hardware-Konzepte sind für Betriebssysteme wichtig?

- Prozessor
  - Kontroll- und Statusregister
  - Befehlsablaufzyklus
  - Interrupts
- Speicher
  - Speicherhierarchie
  - Cache-Prinzip
- Schutzmechanismen
  - Privilegierte Instruktionen
  - I/O-Zugriffsschutz
  - Speicherschutz
  - CPU-Schutz

Einheiten für die Speicherkapazität

SI-Präfixe sind für die Verwendung im Internationalen Einheitensystem (SI) definierte **Dezimalpräfixe:**

1 KB = 1 Kilobyte = 10^3 Byte = 1.000 Byte = Eintausend Byte

1 MB = 1 Megabyte = 10^6 Byte = 1.000.000 Byte = Eine Million Byte

1 GB = 1 Gigabyte = 10^9 Byte = 1.000.000.000 Byte = Eine Milliarde Byte

1 TB = 1 Terabyte = 10^{12} Byte = 1.000.000.000.000 Byte = Eine Billion Byte

Achtung! Für RAM, Cache, Register u.a. sind zur Adressierung **Zweierpotenzen** nötig, daher gilt für diese Speicherangaben:

1 KB = 2^{10} Byte = 1.024 Byte = 1 KiB = 1 Kibibyte

1 MB = 2^{20} Byte = 1.048.576 Byte = 1 MiB = 1 Mebibyte

1 GB = 2^{30} Byte = 1.073.741.824 Byte = 1 GiB = 1 Gibibyte

1 TB = 2^{40} Byte = 1.099.511.627.776 Byte = 1 TiB = 1 Tebibyte

Um Verwechslungen zu vermeiden, hat die IEC in der Norm IEC 80000-13:2008 neue Bezeichnungen für binäre Vielfache definiert *(KiB, MiB, ...)*

→ *leider in der Praxis noch keine konsequente Verwendung!*

Speicherhierarchie

Typische Zugriffszeit	Speichermedium	Typische Kapazität
1 Nanosekunde (10^{-9} s)	Prozessor-Register	< 1 KiB
2 Nanosekunden ($2*10^{-9}$ s)	Prozessor-Cache	< 4 MiB
20 Nanosekunden ($20*10^{-9}$ s)	Hauptspeicher (RAM)	8 GiB
20 Mikrosekunden ($20*10^{-6}$ s)	Halbleiter-Disk (SSD)	512 GB
5 Millisekunden ($5*10^{-3}$ s)	Magnet-Festplatte	4 TB
100 Sekunden (10^{2} s)	Magnetband-kassettensystem	100 TB

Cache-Prinzip

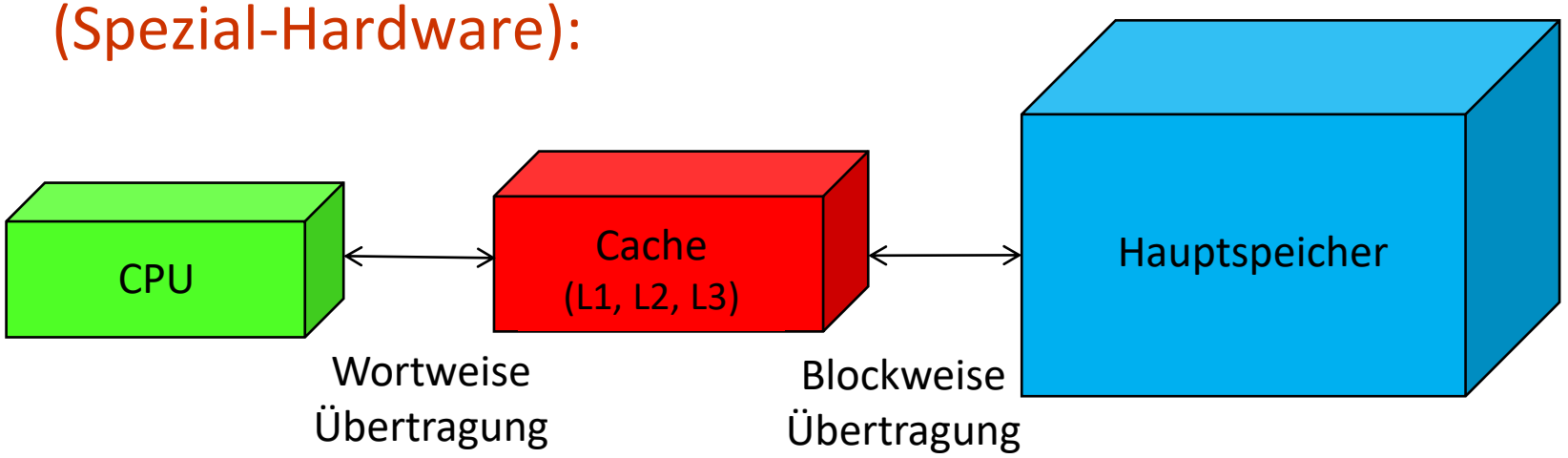
- Nach Zugriff auf ein langsameres Speichermedium werden die zuletzt gelesenen Daten im schnelleren Speicher gehalten (➔ „**Cache-Speicher**“)
- Vor erneutem Zugriff wird geprüft, ob Daten bereits im Cache vorhanden sind ➔ **schnellerer Zugriff**!
- Ist vorteilhaft aufgrund des **Lokalitätsverhaltens** der meisten Programme (z.B. Schleifen etc.)
- Cache-Speicher ist unsichtbar für den Benutzer (Realisierung durch Hardware oder Betriebssystem)

Typisches Lokalitätsverhalten von Programmen

- **Zeitliche Lokalität** (temporal locality)
  - Wenn eine Adresse referenziert wurde, dann wird sie mit hoher Wahrscheinlichkeit **bald** wieder referenziert.
- **Räumliche Lokalität** (spatial locality)
  - Wenn eine Adresse referenziert wurde, dann werden mit hoher Wahrscheinlichkeit die **benachbarten** Adressen bald referenziert.
- **Working Set** W(t,T) eines Programms
  - Die Menge der Adressen, die das Programm in Zeitintervall (t-T,t) referenzierte
  - Je kleiner |W(t,T)|, desto besser das Lokalitätsverhalten des Programms.

Cache-Beispiele

- Prozessor-Cache für Hauptspeicherzugriffe (Spezial-Hardware):



nach
[WS]

- Hauptspeicher-Cache für Festplattenzugriffe *(Bereich im Hauptspeicher)*
- Festplattencache für Zugriff auf das Netzwerk (*z.B. durch lokalen Browser gespeicherte Dateien)*

Kap. 1 Abschnitt 2
Welche Hardware-Konzepte sind für Betriebssysteme wichtig?

- Prozessor
  - Kontroll- und Statusregister
  - Befehlsablaufzyklus
  - Interrupts
- Speicher
  - Speicherhierarchie
  - Cache-Prinzip
- Schutzmechanismen
  - Privilegierte Instruktionen
  - I/O-Zugriffsschutz
  - Speicherschutz
  - CPU-Schutz

Hardwareunterstützte Schutzmechanismen (1)

- **Privilegierte Instruktionen**
  - Schutz des Systems vor fehlerhaften Programmen (alle Fehler von Anwendungsprogrammen müssen vom Betriebssystem entdeckt und bereinigt werden können)!
  - → Verwendung des Kernel- / Usermodus-Bit
  - **→ Nur das Betriebssystem darf privilegierte Instruktionen ausführen (im „Kernelmodus“)!**
- **I/O-Zugriffsschutz**
  - Schutz des Systems vor illegalen oder schädlichen I/O-Operationen
  - → Alle I/O-Operationen sind nur über privilegierte Instruktionen ausführbar (nur durch Betriebssystem-Kernel)
  - → Jeder I/O-Befehl in einem Anwendungsprogramm muss daher durch einen „System Call“ des Betriebssystems ausgeführt werden

Achtung: Gerätetreiber müssen im Kernelmodus laufen (!)

Hardwareunterstützte Schutzmechanismen (2)

- **Speicherschutz**
  - Schutz des Systems vor Zugriff auf unerlaubte Speicherbereiche
  - → Umsetzung von logischen (Programm-)Adressen in physische (Speicher-)Adressen zur Laufzeit
    - Basis-/Limitregister
    - Memory Management Unit (MMU)

    } kommt später!
- **CPU-Schutz**
  - Wie kann garantiert werden, dass das Betriebssystem garantiert wieder die CPU zurück erhält?
    - Timer + Interrupts

Zusammenfassung Abschnitt 2: Welche Hardware-Konzepte sind für Betriebssysteme wichtig?

- Prozessor
  - Kontroll- und Statusregister
  - Befehlsablaufzyklus
  - Interrupts
- Speicher
  - Speicherhierarchie
  - Cache-Prinzip
- Schutzmechanismen
  - Privilegierte Instruktionen
  - I/O-Zugriffsschutz
  - Speicherschutz
  - CPU-Schutz

Kapitel 1
Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen

Kap. 1 Abschnitt 3
Die Struktur von Betriebssystemen

- Betriebssystemdienste
- Betriebssystemaufrufe („System Calls“)
- Betriebssystem-Architekturmodelle

Basis - Betriebssystemdienste

- **Programmausführung**
  - Laden von Programmen in den Hauptspeicher sowie Veranlassen und Überwachen ihrer Ausführung
- **Synchronisations- und Kommunikationsdienste**
  - Regelung der lokalen Kommunikation von Programmen (Prozessen) sowie von Netzwerkverbindungen
- **Dateisystem-Verwaltung**
  - Optimierter und effizienter Zugriff auf Dateien
- **I/O-Operationen**
  - Sollten aus Sicherheits- und Effizienzgründen nicht durch Anwendungsprogramme, sondern nur durch das Betriebssystem erfolgen
- **Fehlerentdeckung und –korrektur**
  - Software- und Hardwarefehler

Erweiterte Betriebssystemdienste

- **Betriebsmittel - Sharing**
  - Faire Aufteilung von Betriebsmitteln (Ressourcen) auf mehrere Benutzer
- **Accounting (Systembenutzung / Dienstbenutzung)**
  - Statistische Daten über die Nutzung des Systems (→ Abrechnung)
  - Daten über die Leistungsfähigkeit des Systems (→ Optimierung)
- **Schutz- und Sicherheitsmechanismen**
  - Benutzerauthentifikation und Zugriffskontrolle

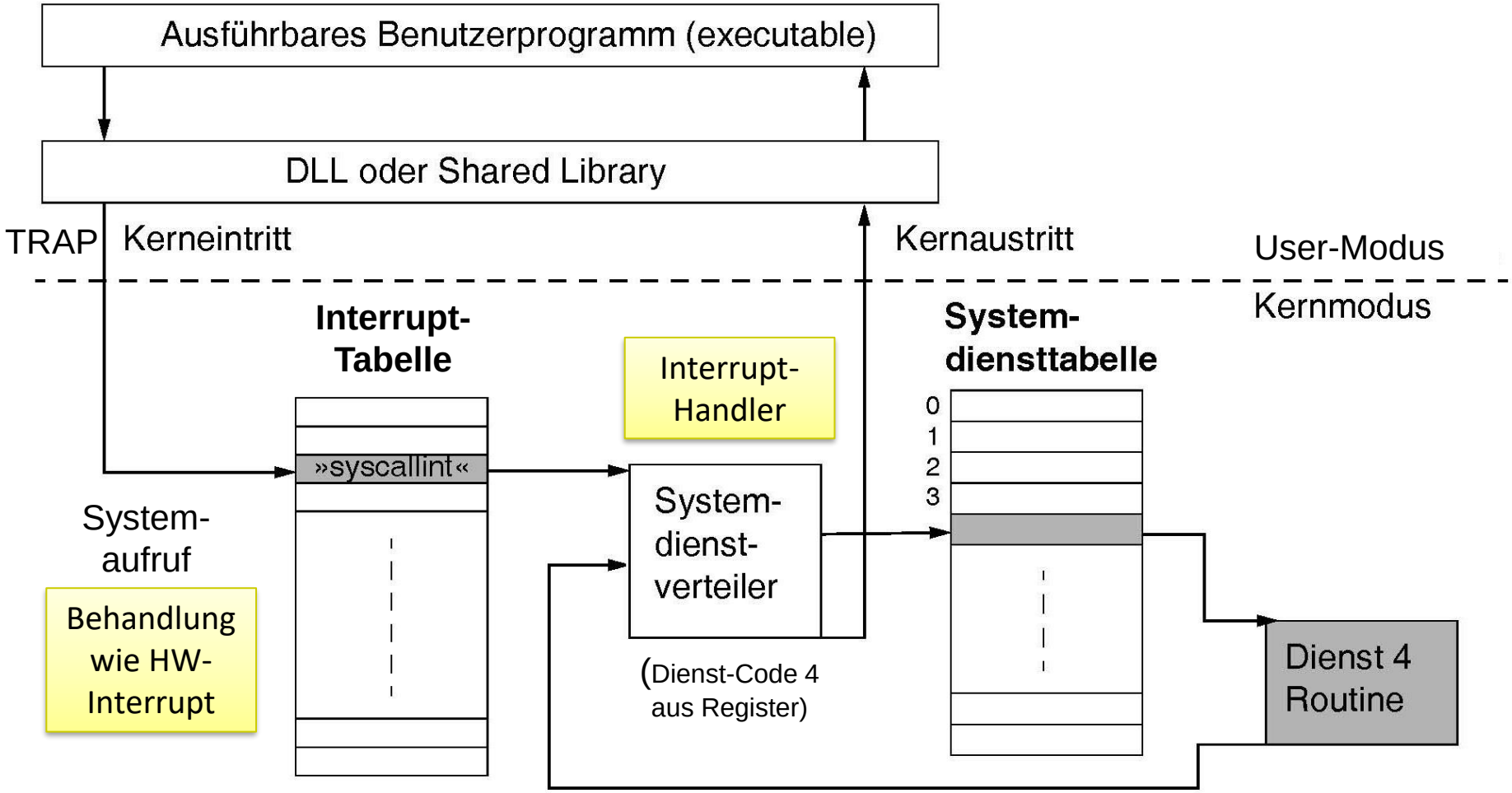
Kap. 1 Abschnitt 3
Die Struktur von Betriebssystemen

- Betriebssystemdienste
- Betriebssystemaufrufe („System Calls“)
- Betriebssystem-Architekturmodelle

Betriebssystemaufrufe („System Calls“)

- Mittels Systemaufrufen werden die Betriebssystemdienste von laufenden Programmen (Prozessen) benutzt
- Systemaufrufe sind über spezielle Bibliotheks-Funktionen möglich (i.d.R. in Programmierumgebung eingebunden, z.B. **`read`**, **`write`**, ..)
- Ein Systemaufruf erzeugt einen Software-Interrupt (mittels "Trap"-Befehl), damit das Betriebssystem im privilegierten Kernelmodus die CPU erhält und die geforderten Dienste ausführen kann.
- Der vom Trap-Befehl übergebene Interruptvektor liefert in der Interrupttabelle als Interrupt-Handler den Systemdienstverteiler ("Dispatcher"). Der Code des aufgerufenen Systemdienstes (z.B. **`read`**) wird diesem **per Register übergeben** und wird als Index in der Systemdiensttabelle verwendet. Nach Beendigung des Systemdienstes sorgt der Systemdienstverteiler für eine Rückkehr in den User-Modus.

System Call - Prinzip



Typische Systemaufrufe

- **Prozesse (Process control)**
  - load, execute, end, abort process
  - createProcess, terminateProcess
  - allocate / free memory
- **Synchronisation und Kommunikation**
  - wait for time, wait for event
  - send / receive message
  - create / delete connection
- **Dateiverwaltung (File management)**
  - create, delete, open, close, read, write file
- **Geräteverwaltung (Device management)**
  - request / release device
  - read, write, reposition
  - get devices attributes, set device attributes
- **Allgemeine Informationen**
  - get / set time or date
  - get /set system data

Kap. 1 Abschnitt 3
Die Struktur von Betriebssystemen

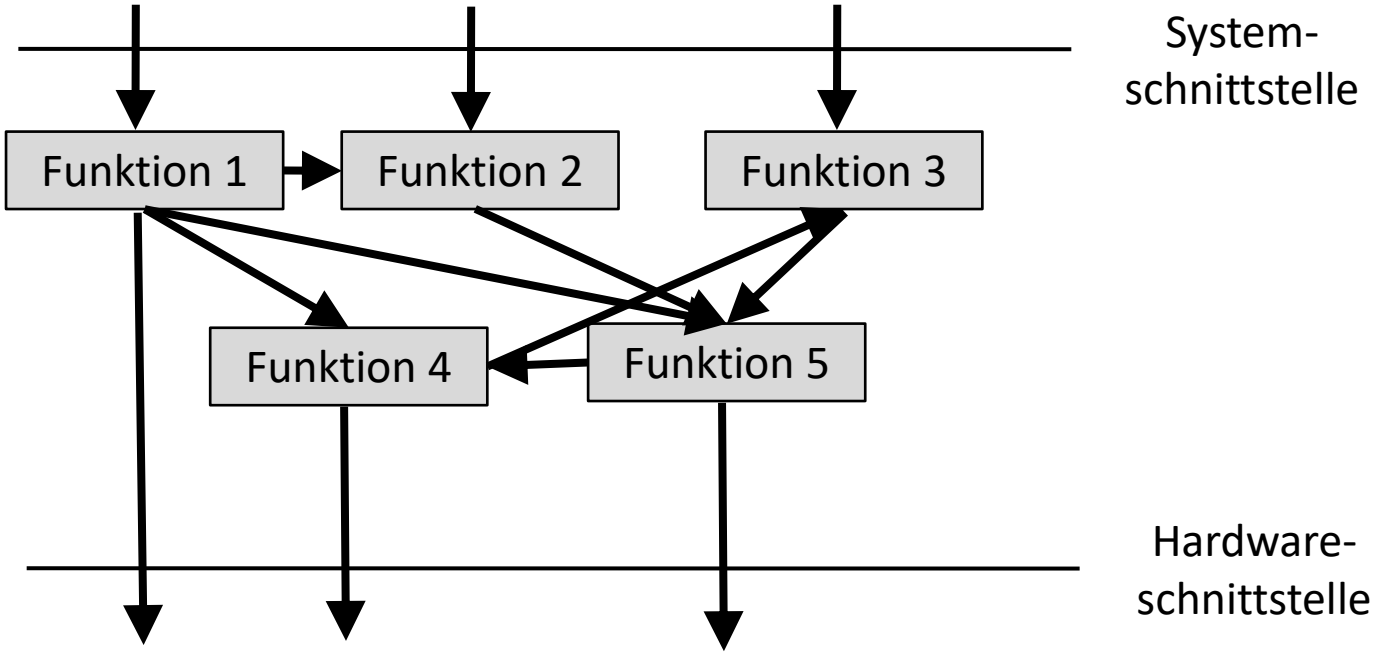
- Betriebssystemdienste
- Betriebssystemaufrufe („System Calls“)
- Betriebssystem-Architekturmodelle

Betriebssystem-Architekturen

Architekturvarianten:

- Monolithisches System (ein Block)
- Schichten-Hierarchie
- Microkernel + Client-/Server-Kommunikation

Betriebssystem-Architekturmodelle (1): Monolithisches System

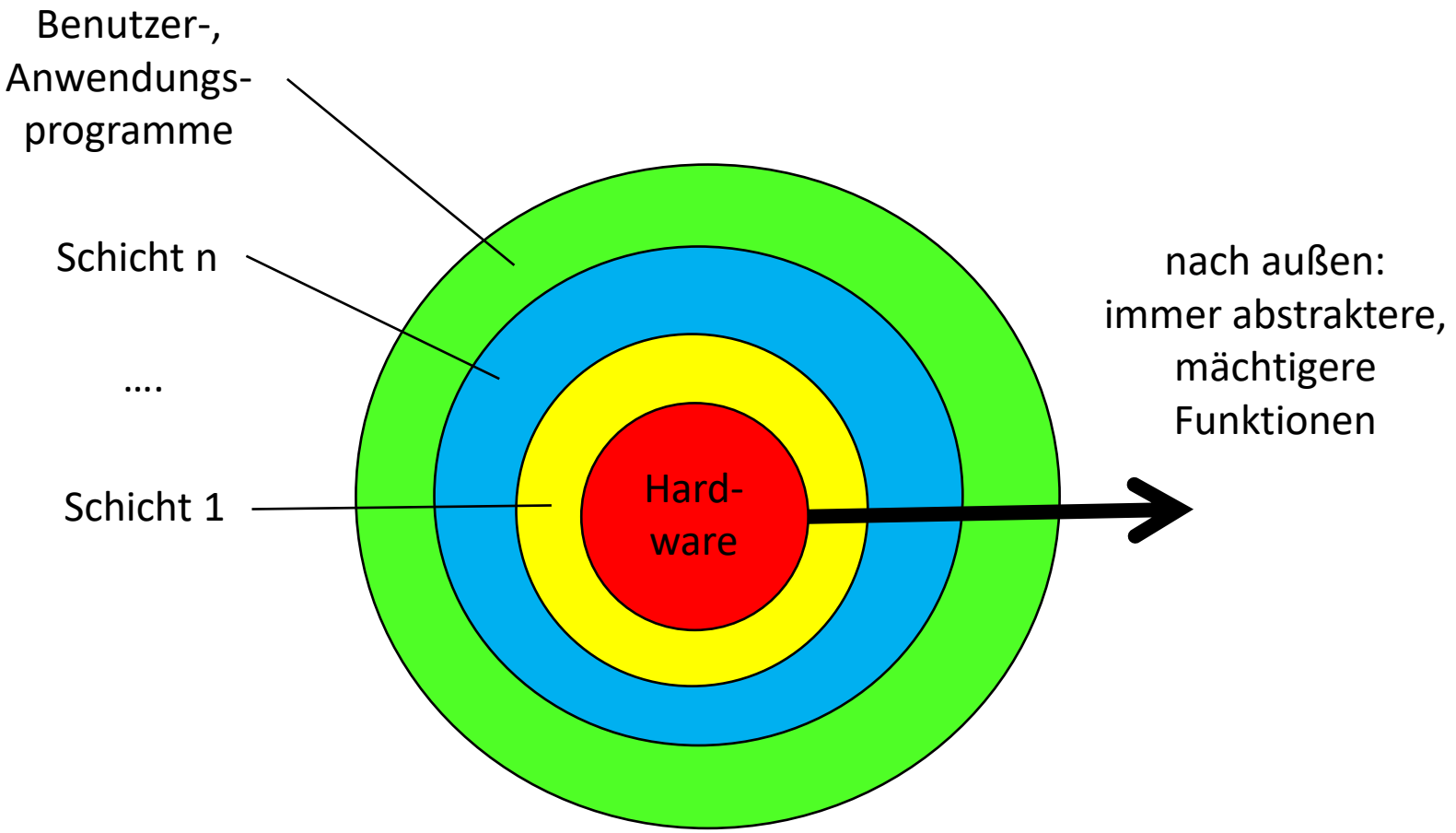


nach [CV]

- Aus einem Block aufgebaut
- Ohne geordnete Struktur oder nur minimale Struktur

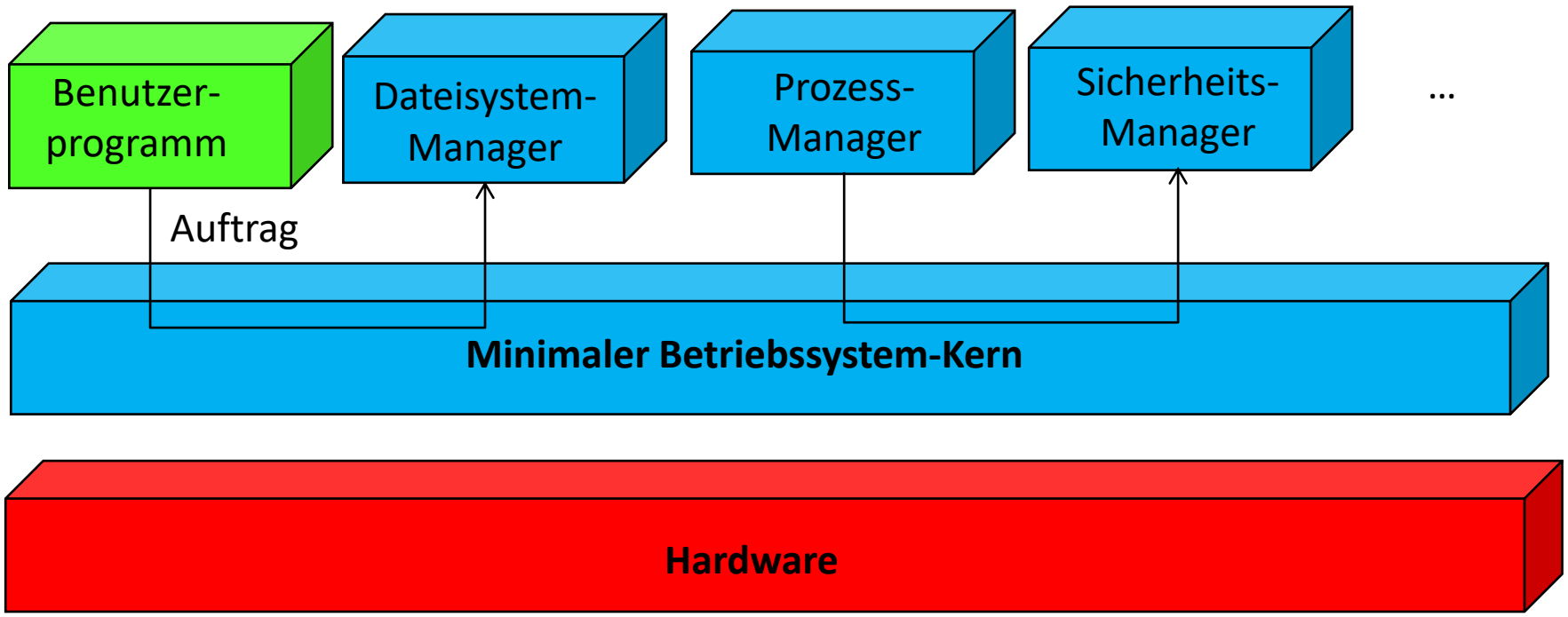
  (z.B. Haupt-, System- und Hilfsfunktionen)

Betriebssystem-Architekturmodelle (2): Schichten-Hierarchie



[CV]

Betriebssystem-Architekturmodelle (3): Minimaler Kern + Client-/Server-Architektur



Vorteil: gute Eignung für verteilte Implementierung
(➔ Client-/Servermodell)

Nachteil: aufwändige interne Kommunikation

[CV]

Zusammenfassung Abschnitt 3: Struktur von Betriebssystemen

- Betriebssystemdienste
- Betriebssystemaufrufe („System Calls“)
- Betriebssystem-Architekturmodelle

Kapitel 1
Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen

UNIX-Geschichte

- 1965: MULTICS-Projekt
  - Neues Schichtenmodell
  - Sehr hohe Ansprüche an die Hardwareleistung
  - Implementierung gescheitert
- 1970-1974: Ken Thompson, Dennis Ritchie (Bell Labs)
  - Entwicklung der Programmiersprache „C“
  - Portable Implementierung eines kleinen, kompakten Mehrbenutzer-Betriebssystems „UNIX“ in C
  - Lizenzvergabe an viele Universitäten

UNIX-Versionen (Auswahl)

- PDP-11-UNIX
  - Originalversion von Thompson/Ritchie
    (8200 Zeilen C-Code, 900 Zeilen Assembler-Code)
- Berkeley-UNIX (BSD)
  - Universität von Kalifornien, viele Weiterentwicklungen
- AT&T System V
  - Eigene Weiterentwicklung (bis 1993)
- **POSIX**
  - Standard zur Vereinheitlichung von UNIX-Befehlen (Schnittmenge von BSD und System V),
    Spezifikation mit Befehlsreferenz:
    https://pubs.opengroup.org/onlinepubs/9699919799/nframe.html
- **MAC OS X**
  - BSD-Weiterentwicklung von Apple *(auch Basis von iOS)*
- **LINUX** (Linus Torvalds)
  - Neuentwicklung, mittlerweile auf OpenSource-Basis *(GNU-Lizenz)*
  - Einheitlicher Kern, viele verschiedene „Distributionen“ mit eigenen Zusatzprogrammen *(u.a. Android)*

UNIX/LINUX – Zugangsmöglichkeiten (kostenlos)

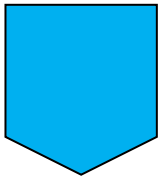
- PC-Pool der Informatik im 11. oder 7. Stock nutzen (Linux booten!)
  Nach Reservierung unter https://workplaces.informatik.haw-hamburg.de
- Rechner mit **Linux** oder **Mac OS X** verfügbar:
  - Terminal (Shell) starten
- Rechner mit **Windows** verfügbar:
  - Zugriff auf UNIX-Server der Informatik mittels PuTTY (ssh-Client)
    - Putty.exe downloaden (http://www.putty.org) und starten
    - SSH-Verbindung aufbauen mit usershell.informatik.haw-hamburg.de unter Port 22 (Benutzername und Passwort: HAW-Account)

ODER

  - Linux auf eigenem Windows-Rechner ausführen
    - Linux MINT (http://www.linuxmint.com) oder oder Ubuntu (https://www.ubuntu.com/#download) oder … downloaden, auf DVD brennen oder bootfähigen USB-Stick kopieren und auf Festplatte installieren
    - Software für virtuelle Maschinen installieren (https://www.vmware.com/go/downloadplayer oder https://www.virtualbox.org/), anschließend heruntergeladenes Linux in virtueller Maschine installieren
    - Windows Subsystem for Linux (WSL) unter Windows 10 installieren: https://docs.microsoft.com/en-us/windows/wsl/install-win10 (WSL 1 reicht - nur Schritt 1 und 6)

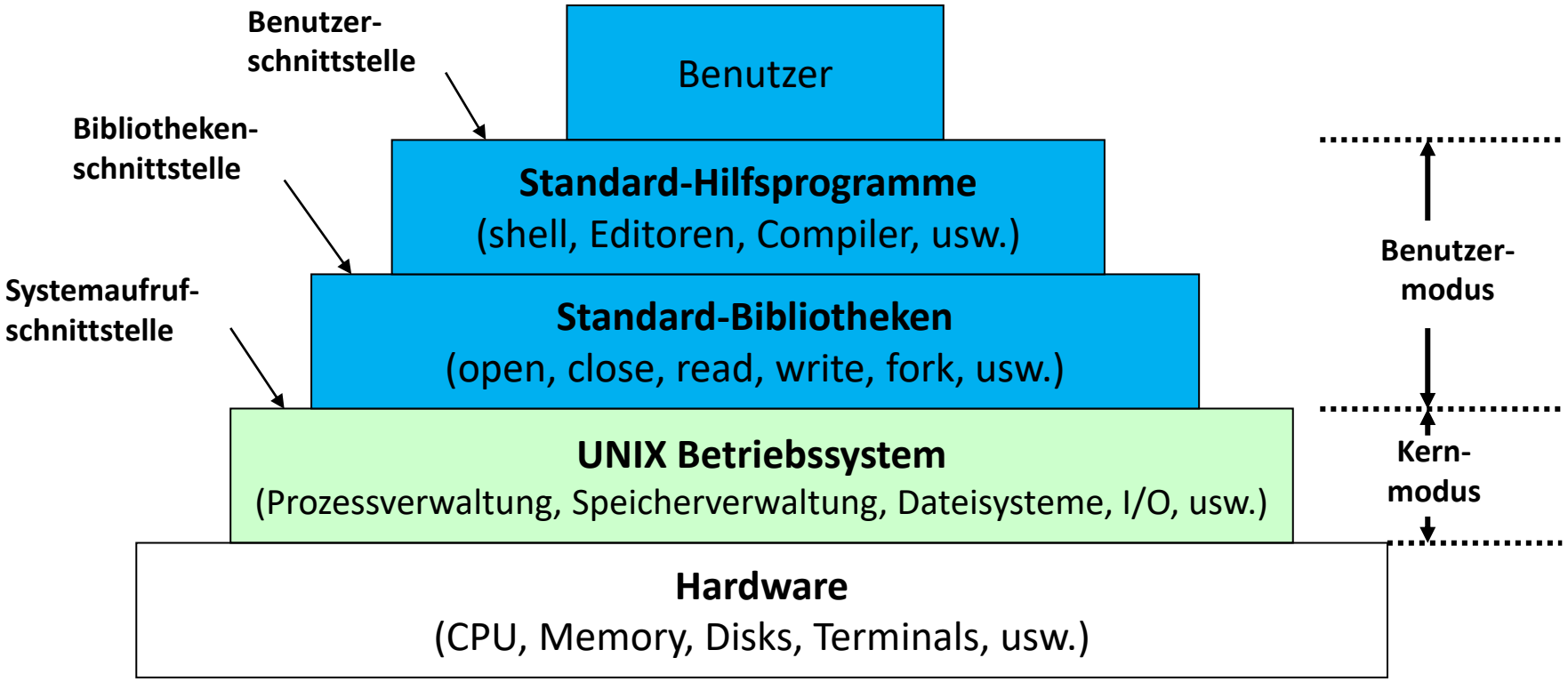
UNIX - Ziele

- Von Programmierern **für Programmierer** entworfen
- Mehrheit der **Benutzer ist relativ erfahren** und oftmals in komplexe Softwareentwicklung eingebunden
- Modell „**enge Zusammenarbeit von Programmierer-Team“** (unterscheidet sich fundamental vom Modell „persönlicher Computer mit einzigem Anfänger“)

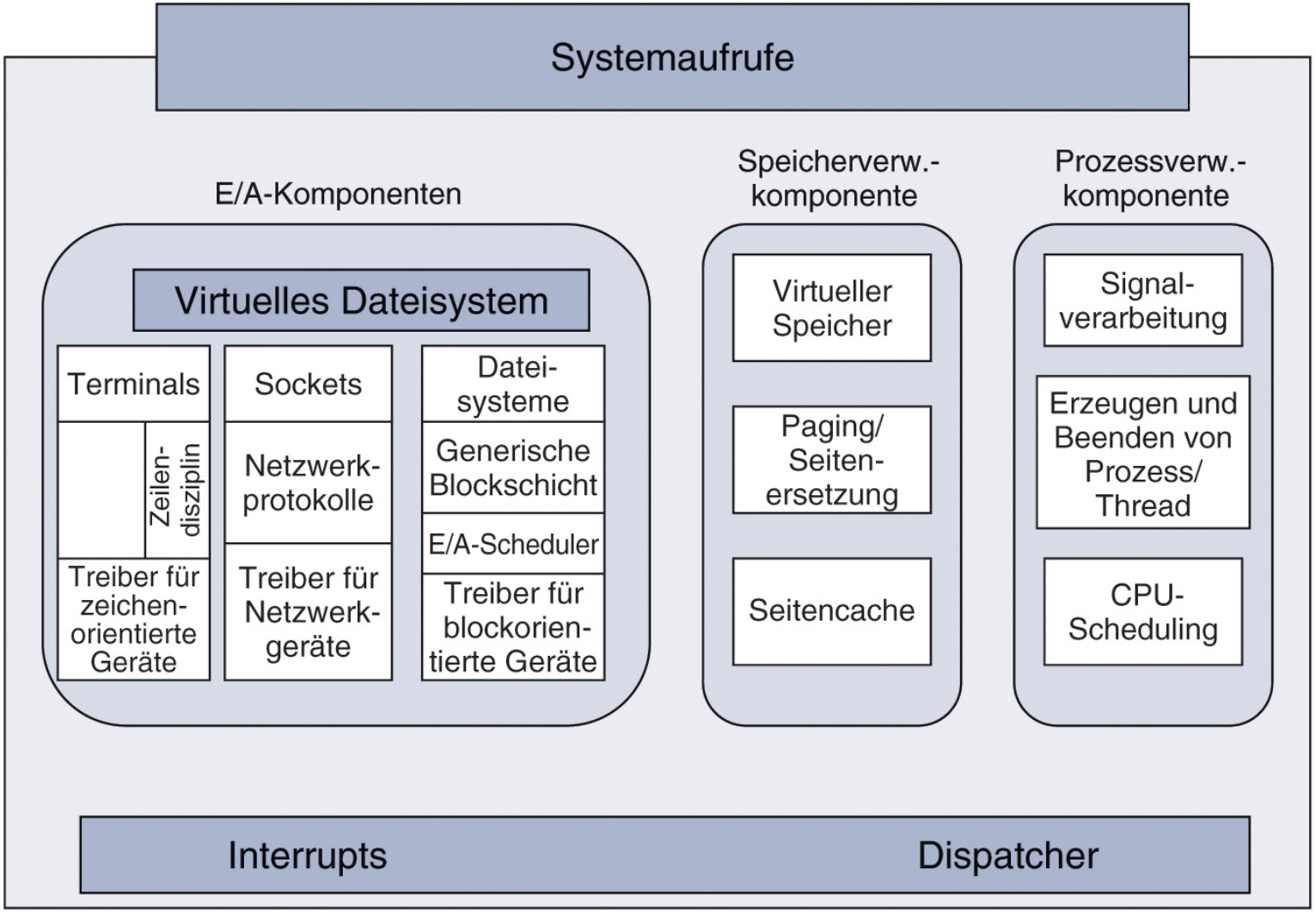


Die Frage „**Was erwartet ein guter Programmierer vom System?“** ist bei den wesentlichen Designentscheidungen von UNIX zu Grunde gelegt worden

UNIX – Schnittstellen



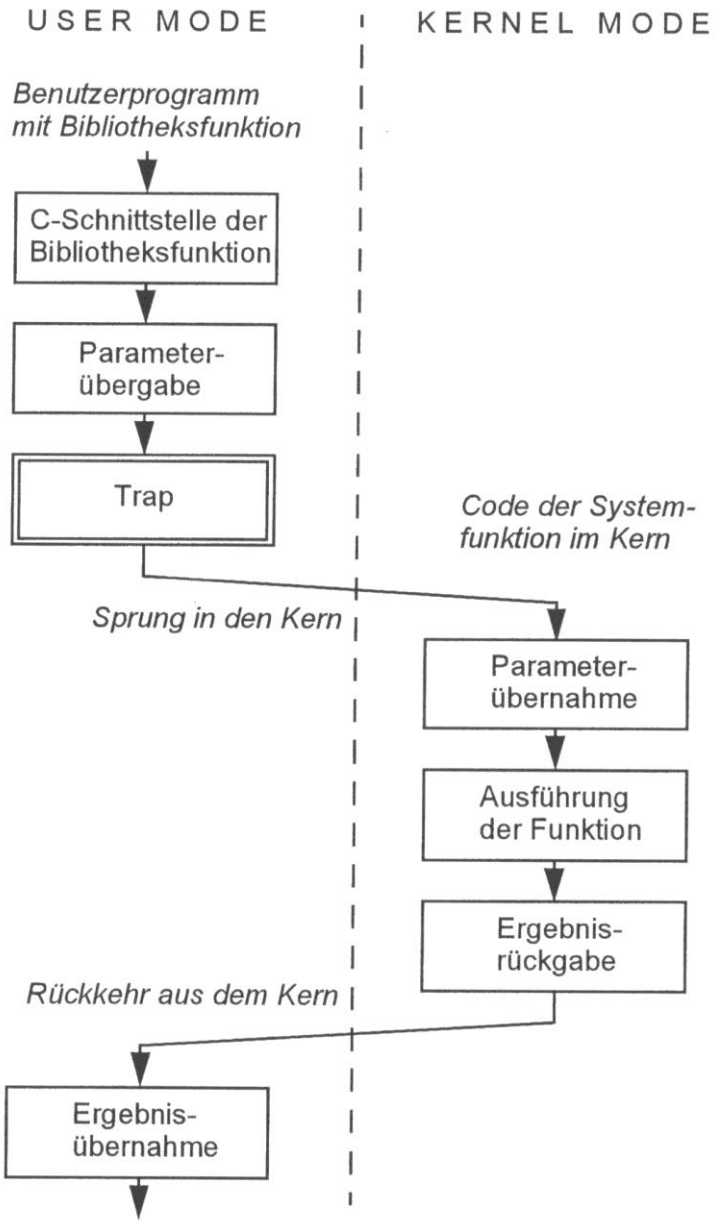
UNIX – Architektur (Linux)



(Systemdienstverteiler)

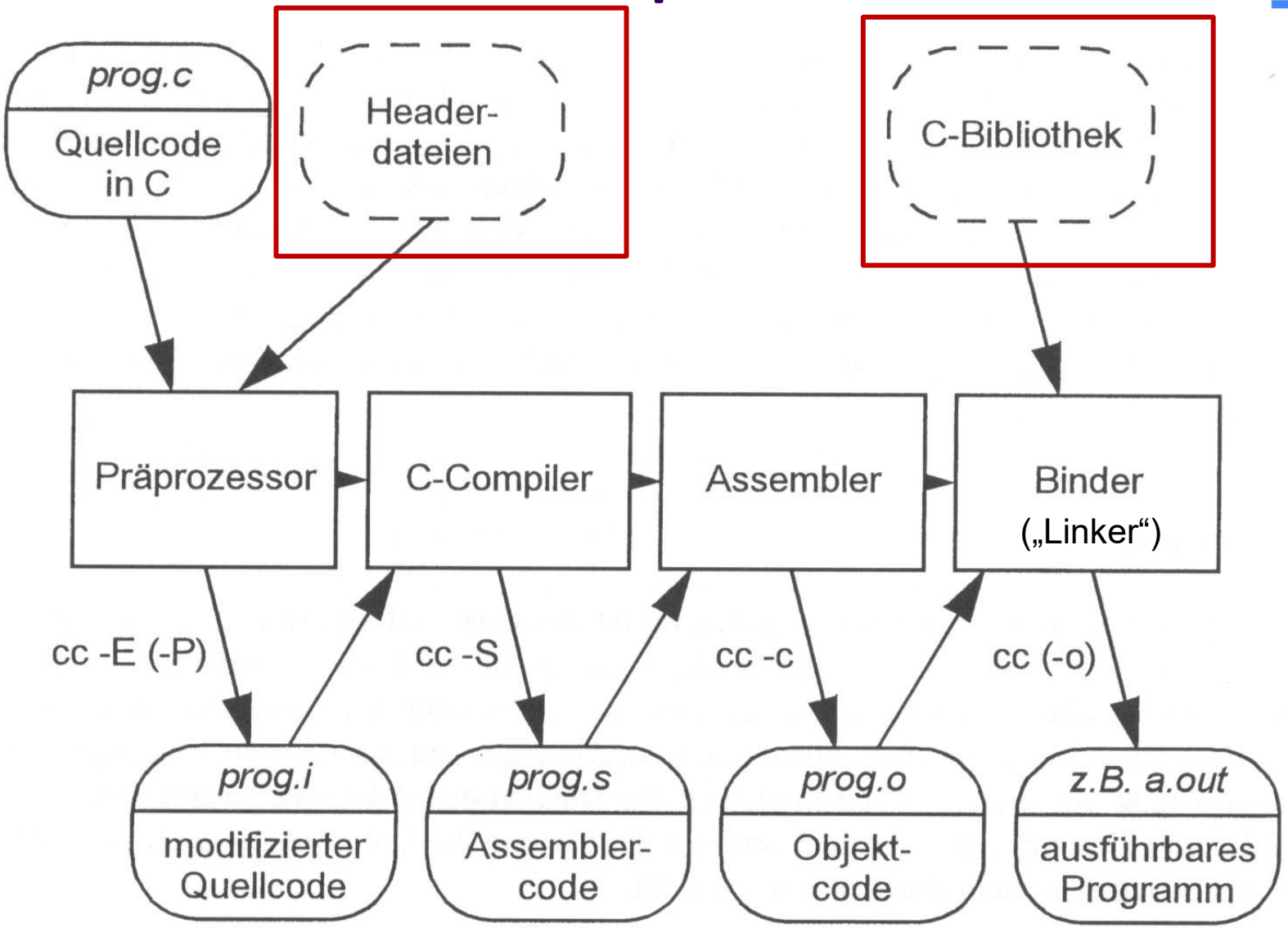
Systemaufrufe in UNIX

- Systemaufruf (Wechsel in den Kern mittels SW-Interrupt) nur über Bibliotheksfunktion möglich
- Einbindung in C-Programme über #include der Headerdateien (/usr/include bzw. /usr/include/sys)
- TRAP-Funktion verursacht (Software-)Interrupt mit Wechsel in den Kernel-Mode → „System Call“ *(Parameterübergabe mittels Stack)*



[CV]

Schritte des UNIX-C-Compilers



Shell: Ablauf einer Befehlsausführung

- Die Shell gibt ein Bereitzeichen *(„Prompt“)* aus *(„$“, „#“ oder „%“)*
- Der Benutzer gibt einen Befehl ein:

  BEFEHL [-OPTIONEN] [ARGUMENT1] [..] [&]

[ ] bedeutet: kann weggelassen werden

- Die Shell analysiert den Befehl:
  - Führt ihn selbst aus, falls es sich um einen „Built-in“-Befehl für die Shell selbst handelt
  - Ansonsten Ausführung des Befehls als neues Programm *(über durch die Shell ausgelöste System Calls):*
    - Suchen der angegebenen Programmdatei *(Dateiname = Befehl)*
    - Laden der Programmdatei in den Hauptspeicher
    - Starten des Programms (mit Übergabe von Optionen und Argumenten)
- Die Shell gibt ein neues Bereitzeichen aus *(„$“, „#“ oder „%“)*

Shell-Variablen (bash)

- Variable: Ein benannter Speicherplatz
- **Shell-Variablen können nur Zeichenketten (Strings) speichern**

- Erzeugung / Wertzuweisung einer Shell-Variablen: **VAR=VALUE**
  Wenn die Variable **VAR** noch nicht existiert, wird sie automatisch erzeugt.
  Der String **VALUE** wird der Variablen **VAR** als Wert zugewiesen
  Beispiel:
  ```
  $ USRBIN=/usr/local/bin
  ```
  (Wert-String ohne Leerzeichen, sonst müssen " " verwendet werden!)

- Zugriff auf den gespeicherten Wert durch **$** vor dem Variablennamen: **$VAR**
  Die **Zeichenkette $VAR** wird durch den aktuellen **Wert** der Variablen **VAR** ersetzt (→ „$“ bewirkt eine Ersetzung eines Strings durch einen anderen!)
  Beispiel:
  ```
  $ echo $USRBIN
  /usr/local/bin
  ```

Shell-Skripte

- Shell-Skript: „Normale“ Textdatei mit beliebig vielen Shell-Befehlen
- Alle Befehle werden der Reihe nach ausgeführt (wie bei einer manuellen Eingabe), Shell-Variablen (z.B. `$HOME`) werden dabei durch den aktuellen Wert ersetzt
- Übergabe von Argument-Werten durch spezielle Variablen: 1, 2, 3 …
  → Die Zeichenkette $1, $2, $3, .. wird durch jeweils 1., 2., 3. .. Argument der Befehlszeile ersetzt *($0: Programmname)*
- Das Shell-Skript `$HOME/.bashrc` wird bei jedem Start einer `bash` ausgeführt → Definition eigener Befehle/Variablen möglich

Ausführung eines Shells-Skripts

1. Möglichkeit:

Start einer neuen Shell und Übergabe der Skriptdatei als Argument: **`bash SKRIPT-DATEINAME`**

2. Möglichkeit:

Skriptdatei als „ausführbar“ deklarieren:

```
chmod a+x SKRIPT-DATEINAME
```

und wie eine „normale“ Programmdatei starten, z.B. durch Angabe eines absoluten Dateipfads

```
./SKRIPT-DATEINAME
```

Weitere Infos zu Linux siehe Vorlesung „Grundlagen der Informatik“ (GI) Kap. 3 oder Aufgabenstellung Praktikum 1!

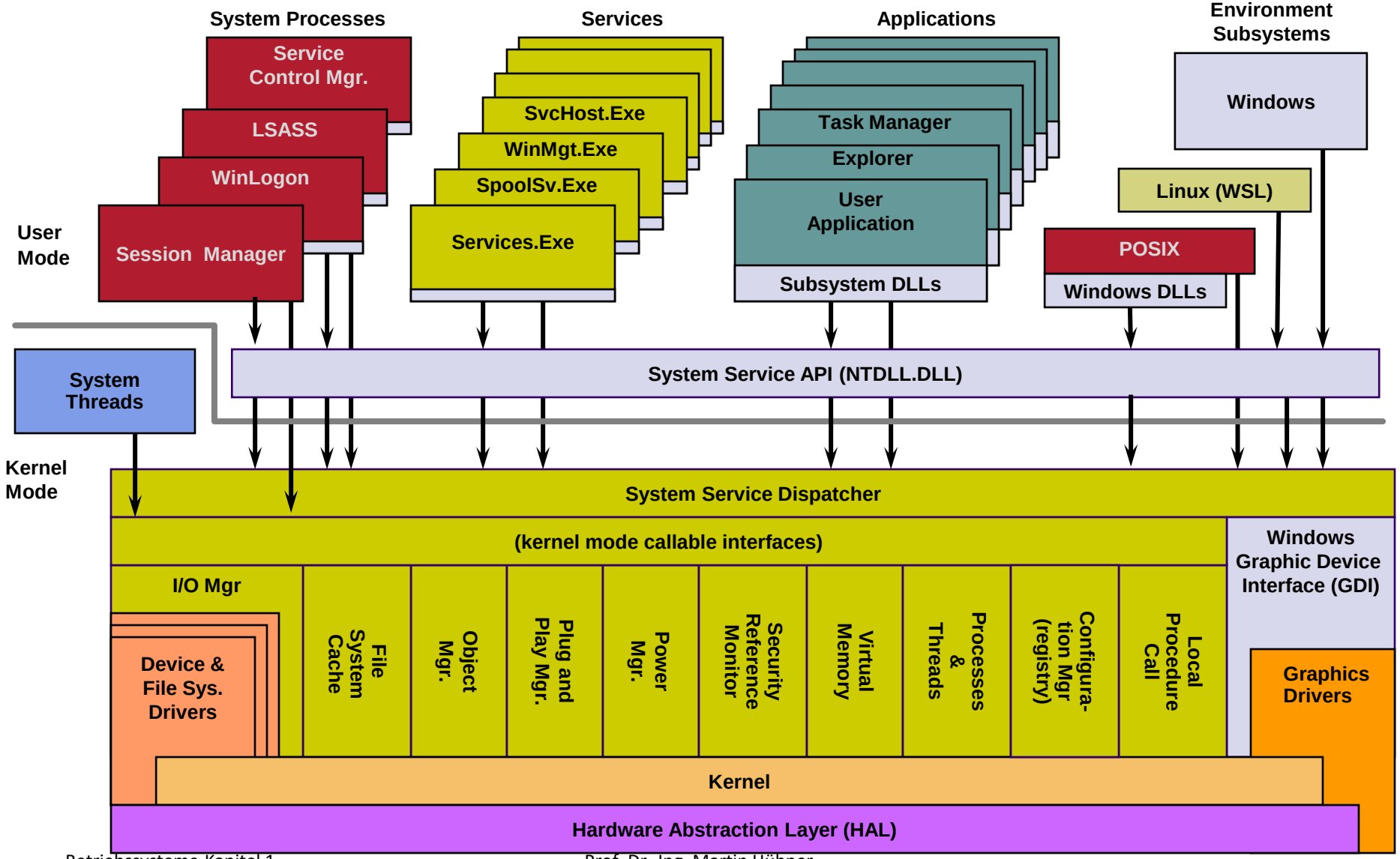
Kapitel 1
Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen

Windows 2000/XP/Vista/7/8/10 - Struktur

- Modulare Struktur ➔ Flexibilität
- Überwiegend in C geschrieben (kleine Teile Assembler bzw. C++), dennoch Umsetzung eines Objektmodells
  ➔ Portabilität gegeben
- Modifizierter Microkernel
  - Keine reine Microkernel – Architektur, da gemeinsamer Adressraum aller Module
  - Viele Funktionen außerhalb des Kernels laufen im Kernel-Modus (Vermeidung von Kontextwechseln)
- Module können entfernt / aktualisiert / ersetzt werden, ohne das gesamte System zu ändern

Windows Architektur



Strukturierung der Windows-Bibliotheken

- Subsystem-Bibliotheksfunktionen:
  - Dokumentierte, von Benutzerprogrammen für die Nutzung von Betriebssystem-Funktionen aufrufbare Bibliotheksfunktionen **(.DLL)**
  - Beispiele Windows: *CreateProcess, CreateFile, GetMessage*
- Windows System Services (Datei: NTDLL.DLL):
  - Undokumentierte Funktionen, die aber im User-Mode aufrufbar sind (z.B. von Subsystem-Bibliotheken und Services)
  - Erzeugen ggf. Trap (Wechsel in den Kernelmode)
  - Beispiel: *NtCreateProcess* wird von Windows *CreateProcess(..)* und POSIX *fork()* als ein interner Dienst genutzt
- Windows-interne Funktionen:
  - Funktionen innerhalb der Windows Module („executive"), dem Kernel oder der HAL
  - Nur im Kernelmode aufrufbar
  - Beispiel: *ExAllocatePool* reserviert Speicher im Windows system heap

Strukturierung der Windows-Bibliotheken (2)

- Windows Services (Dienste):
  - Prozesse, die vom Service Control Manager gestartet werden
  - Laufen i.d.R. ohne Benutzerschnittstelle im Hintergrund
  - Beispiel: Der *Taskplaner-Dienst* (startet bestimmte Programme zu einer festgelegten Zeit)

Dateiformat: DLL (dynamic link library)

- Bibliotheksfunktionen im Binärformat (Maschinencode)
- Werden nur bei Bedarf dynamisch in den Hauptspeicher geladen
- Liegen immer nur einmal im Hauptspeicher → können von mehreren Prozessen (lesend) verwendet werden
- Beispiele: *MSVCRT.DLL – MS Visual C++ run-time Bibliothek*
  KERNEL32.DLL – eine der Windows API Bibliotheksdateien

Windows APIs

- Die Windows-Architektur unterstützt mehrere Subsysteme mit eigenem API ("Application Programming Interface")
  - **Windows** (Standard), **POSIX** (Unix), **Linux** (ab Windows 10)
  - Benutzeranwendungen erzeugen System Calls über das API einer Subsystem-Bibliothek (.DLL-Datei)
- **Universal Windows Platform (UWP)** (ab Windows 10)
  - Bibliotheken und Klassen für Windows-Applikationsentwicklung, z.B. in .NET-Sprachen (C#, F#, VB, C++, ..)
    → Benutzen intern u.a. Win32 API
- **Win32 API**
  - Gemeinsame C++/C-Programmierschnittstelle für alle Windows-Versionen
  - Header-File für alle Funktionen: <windows.h>
  - Windows-Datentypen für C/C++: https://docs.microsoft.com/de-de/windows/win32/winprog/windows-data-types
  - Beispiel: https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-getcurrentdirectory

Dokumentation: https://docs.microsoft.com/en-us/windows/apps/

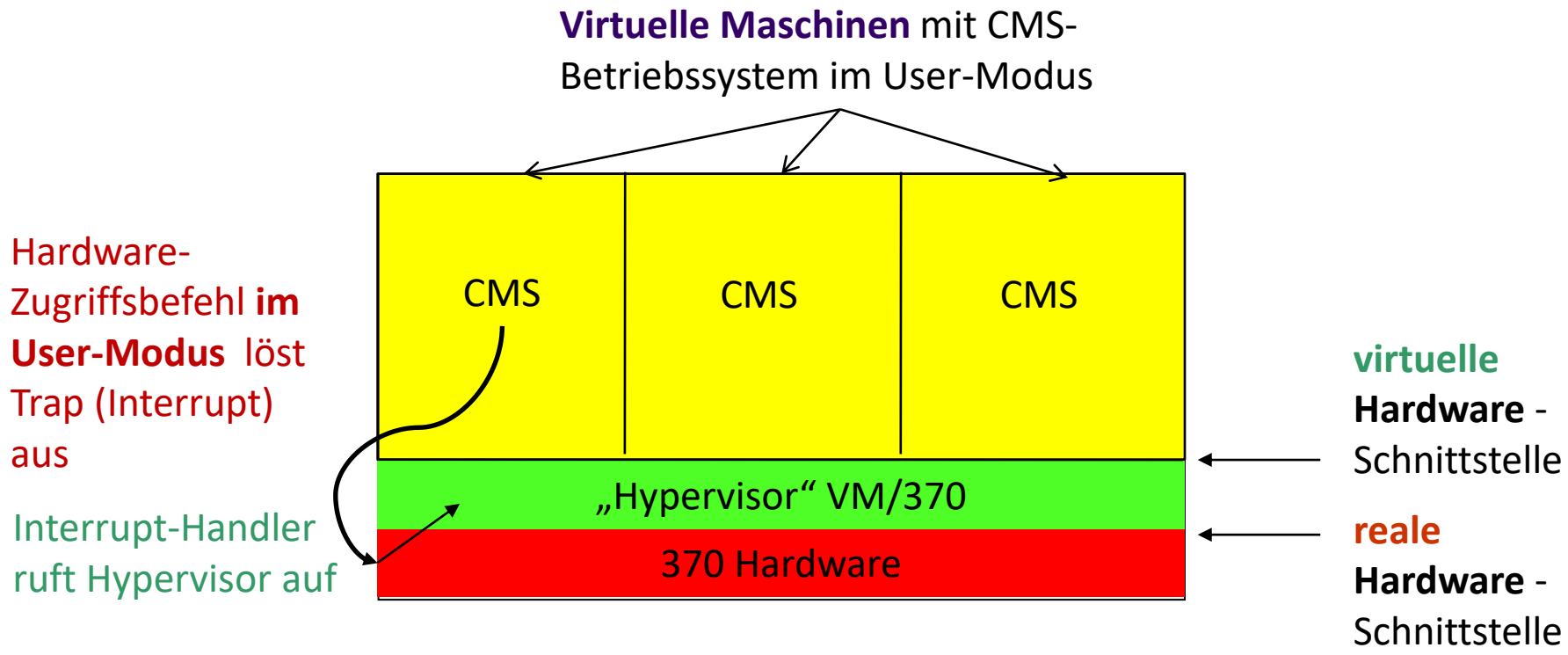
Kapitel 1
Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen

Virtuelle Maschinen

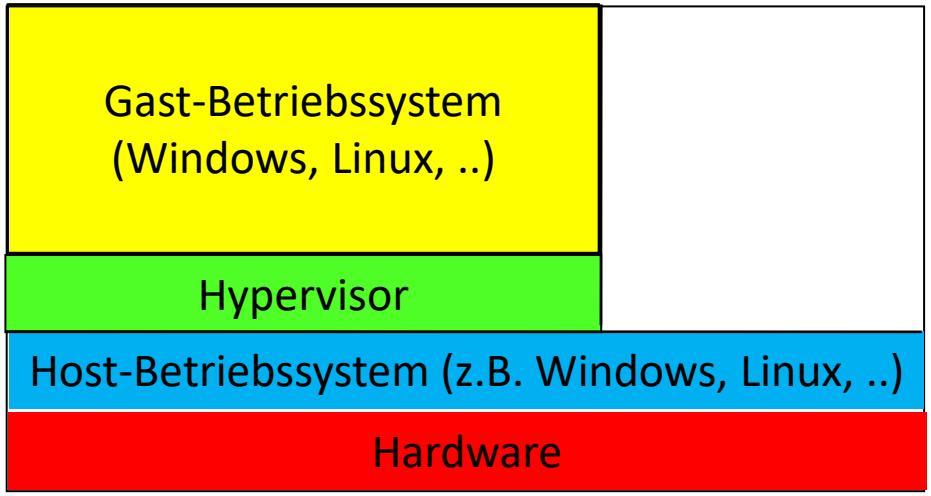
- Eine virtuelle Maschine wird durch einen Monitor-Betriebssystemkern (“Hypervisor”) realisiert, der alle Hardwareschnittstellen einer realen Maschine softwaretechnisch zur Verfügung stellt
- Auf dieser Grundlage können mehrere “Gast”-Betriebssysteme auf einer realen Maschine parallel betrieben werden
- Jedes Gast-Betriebssystem greift auf seine eigene virtuelle Maschine mit allen Hardware-Schnittstellen und –geräten zu
- Privilegierte Maschinenbefehle eines Gast-Betriebssystems werden vom Hypervisor abgefangen und interpretiert, wobei der Zugriff auf die realen Hardware – Ressourcen zwischen allen virtuellen Maschinen koordiniert wird

Typ-1-Hypervisor mit "Trap and Emulate": Beispiel VM/370 (IBM - 1972)



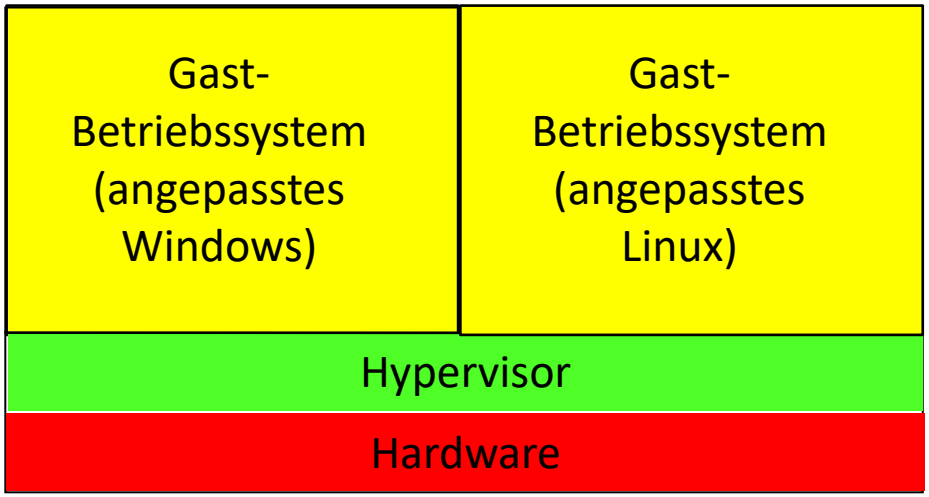
- Auf einem IBM-Mainframe unter VM/370 (heute z/VM) konnten eine Vielzahl von virtuellen Maschinen mit dem Einplatz-Betriebssystem CMS gleichzeitig laufen (→ Ermöglichung von Multiprogramming)
- *Voraussetzung: Die Hardware implementiert das Abfangen von privilegierten Befehlen im User-Modus (Auslösen eines Traps) effizient und sicher, d.h. der Prozessor unterstützt "Virtualization Technology" (VT)*

Typ-2 Hypervisor mit "Binary Translation" (z.B. VMWare Workstation)



- Der Hypervisor läuft als Benutzerprogramm eines Host-Betriebssystems
  → keine Kernelmodus-Rechte!
- Abfangen der privilegierten Befehle des Gast-Betriebssystems durch den Hypervisor (*"Binary Translation"*):
  - Lesen des auszuführenden Maschinencodes des Gastsystems (zur Laufzeit)
  - Ersetzen der privilegierten Befehle durch Hypervisor-Aufrufe (Cachen des Codes)
  - Umsetzung der privilegierten Befehle durch den Hypervisor (durch System Call des Host-Betriebssystems)

Paravirtualisierung (z.B. XEN)

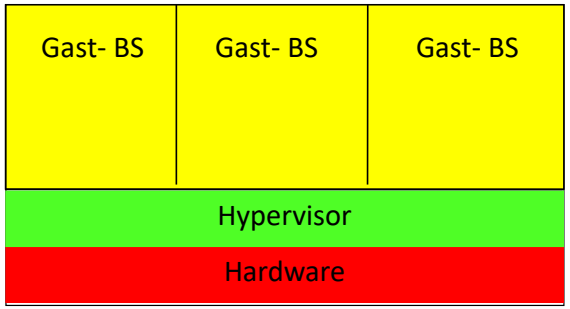


Anpassung des Gastbetriebssystem-Codes an den Hypervisor **vor** der Installation:

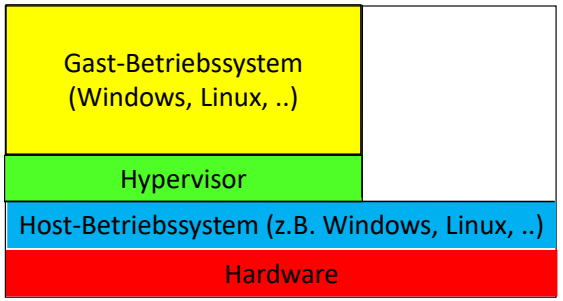
- Ersetzen der privilegierten Befehle (Hardwarezugriff) durch Hypervisor-Aufrufe

Zusammenfassung: Hypervisor-Realisierung

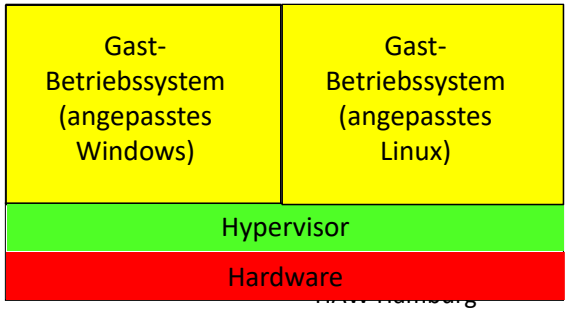
- **Typ-1-Hypervisor**
  - Ein Hypervisor schirmt die komplette Hardware ab
  - Es gibt nur Gast-Betriebssysteme
  - Technik: **Trap and Emulate** oder **Binary Translation**



- **Typ-2-Hypervisor**
  - Jedes Gast-Betriebssystem hat einen eigenen Hypervisor
  - Auf der Hardware läuft ein Host-Betriebssystem
  - Technik: **Binary Translation**



- **Paravirtualisierung**
  - Typ-1 – oder Typ-2- Hypervisor
  - Technik: **Codeanpassung der Gast-Betriebssysteme**



Vorteile von virtuellen Maschinen im Rechenzentrum

- Bessere Auslastung von Hardware-Ressourcen
  - Viele Applikationen benötigen spezielle Systemeinstellungen
    → eigene Server nötig (Webserver, Mailserver, Datenbankserver, …)
  - Mehrere Server (als virtuelle Maschinen) auf einem Hardware-Server möglich
- Höhere Flexibilität
  - Verschieben von virtuellen Maschinen zur Laufzeit möglich, wenn Server-Hardware identisch
    → 7x24-Stundenbetrieb ohne HW-Wartungsfenster realisierbar
  - Hardware-Erweiterbarkeit ohne Neuinstallation von Betriebssystem und Applikationen
    → Erhöhung der Ressourcenzuteilung für eine virtuelle Maschine zur Laufzeit
- Höhere Ausfallsicherheit
  - Virtuelle Maschine auf anderer Hardware lauffähig

Nachteile / Grenzen von virtuellen Maschinen

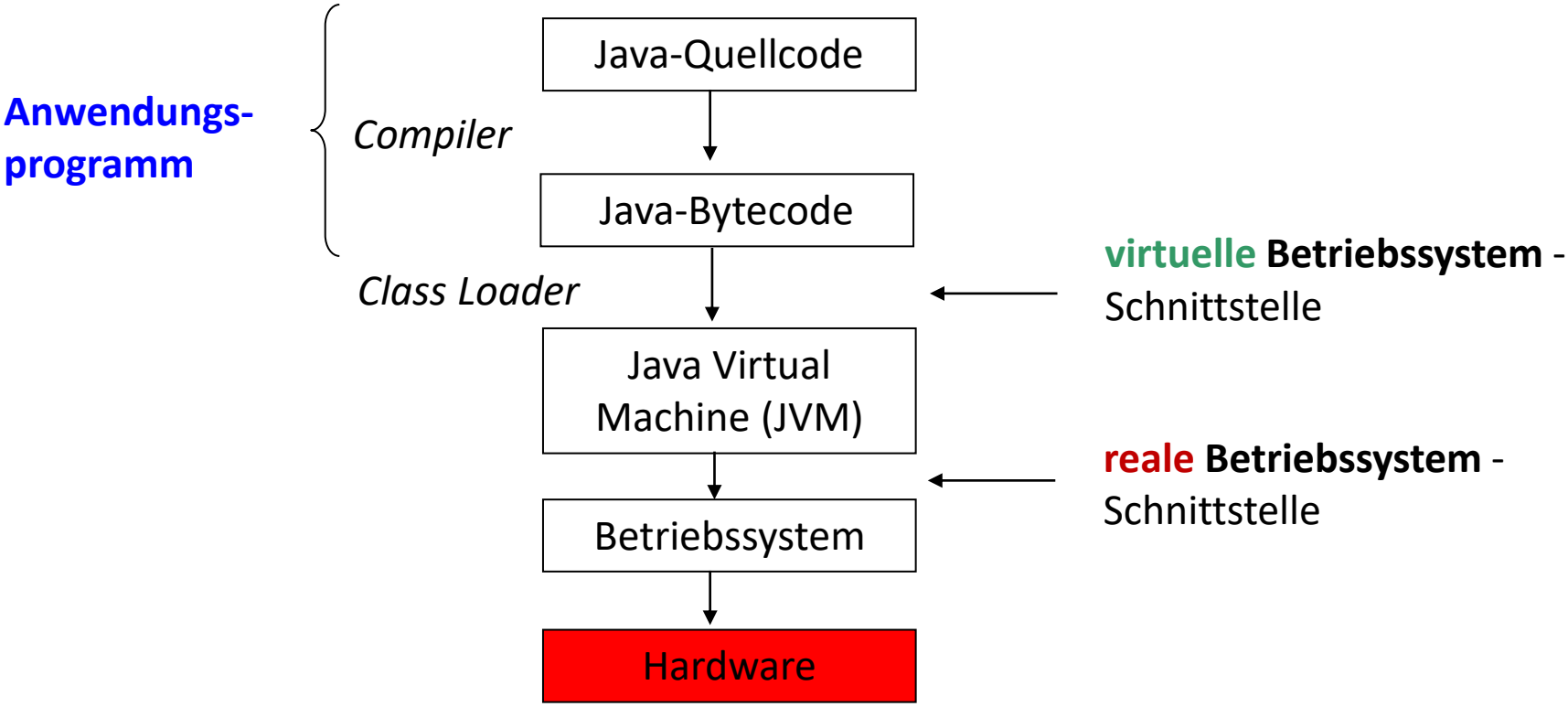
- Leichter Performanceverlust (ca. 5%)
- Erhöhung der Komplexität *(z.B. virtuelles Netzwerk)*
- Kein Einfluss auf die Auslastung der zugeteilten Ressourcen durch die virtuelle Maschine

→ Aktueller Ansatz:
„Container“ für Applikationen statt virtueller Maschinen

Containertechnologie: Beispiel „Docker“

- **Docker** benutzt Linux-Kernelfunktionen *(cgroups, namespaces, Netfilter, ...)* zur **Kapselung von Applikationen in Containern**
- Ein Container enthält die komplette Systemumgebung für eine Applikation *(Code, Laufzeitmodul, Systemwerkzeuge, Systembibliotheken)*
- Ein Container wird in einer Image-Datei gespeichert und von der „Docker-Engine“ gestartet
  → Kernel-Konfiguration zur Laufzeit über Docker-Engine
- Container-Images können andere Images enthalten
  → „Layering“ - Konzept durch „union mounts“ *(OverlayFS)*
- Verwaltung / Organisation von vielen Containern über Zusatz-Tools wie „Kubernetes“ *(Google)*

JAVA: Virtuelle Maschine für Anwendungsprogramme



- JVM: Abbildung einer virtuellen auf eine reale **Betriebssystem** – Schnittstelle!

Ende des 1. Kapitels: Was haben wir geschafft?

Einführung & Überblick

1. Was ist ein Betriebssystem?
2. Grundlegende Hardware-Konzepte
3. Die Struktur von Betriebssystemen
4. Überblick UNIX
5. Überblick Windows
6. Virtuelle Maschinen