



Betriebssysteme und Netzwerke

Vorlesung 2



Artur Andrzejak

Aufgaben der Betriebssysteme

Zwei primäre Aufgaben eines BSs

- ▶ **A.** Bereitstellung einer erweiterten Maschine
 - ▶ BS als „komfortable Hardware“ => **erweiterte Maschine**
 - ▶ **Abstraktionen** (u.a. der Hardware) erlauben eine einfachere Programmierung und Nutzung der Hardware
- ▶ **B.** Verwaltung der Ressourcen
 - ▶ BS muss Zugriff auf Prozessor(en), I/O-Geräte, Speicher, ..., verwalten
 - ▶ Die einzelnen Prozesse bekommen die HW-Ressourcen vom BS zugeteilt
 - ▶ Vermeidung von Konflikten, Fairness, Schutz voneinander
 - ▶ Ein muss bei Multiprogrammierung!

A: BS als eine erweiterte Maschine /1

- ▶ Problem: Hardware (**HW**) ist oft komplex, Verwendung ist „hässlich“ und sehr kompliziert
- ▶ Z.B.: direktes Lesen von Floppy Disk (NEC PD765)
 - ▶ „Read Track“ – 2H: ein Befehl, das in ein Gerätereister geschrieben wird
 - ▶ 13 Parameter in 9 Bytes
 - ▶ Zusätzlich: Programmierer muss selbst dafür sorgen, den Motor ein- und auszuschalten

Bit Byte	7	6	5	4	3	2	1	0
0	0	F	S	0	0	0	1	0
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length							
7	Length of GAP3							
8	Data Length							

A: BS als eine erweiterte Maschine /2

- ▶ Ein BS versteckt diese Komplexität der Hardware
 - ▶ BS bietet den Benutzern und Programmierern einfachere, mehr komfortable Wege, auf die HW + Dienste zuzugreifen
- ▶ Das geschieht mittels **Systemaufrufen**: Funktionen, die das BS zur Verfügung stellt
- ▶ Diese Funktionen bilden die **Schnittstellen** eines BS
 - ▶ Schnittstelle = **Application Programming Interface (API)**
- ▶ Die Schnittstellen arbeiten mit **Abstraktionen**
 - ▶ Statt dem „Lesen von *Spuren* / *Sektoren* auf einer Floppy“ (konkretes Konzept niedriger Stufe) ...
 - ▶ ... man „öffnet eine *Datei*“ (abstraktes Konzept – Dateien sind nichts physisches)

B: BS als Verwalter der Ressourcen

- ▶ Die Rolle des BSs hier ist eine geordnete und kontrollierte **Zuteilung (allocation) der Ressourcen**
- ▶ Zwingend bei der mehrfachen Nutzung von Ressourcen
 - ▶ Was würde passieren, wenn mehrere Programme versuchen würden, zugleich einen Drucker zu nutzen?
- ▶ Relevanten Funktionalität (Auswahl):
 - ▶ Schutz der Prozesse / Nutzer voreinander
 - ▶ Sicherstellung der Fairness („jeder kommt dran“)

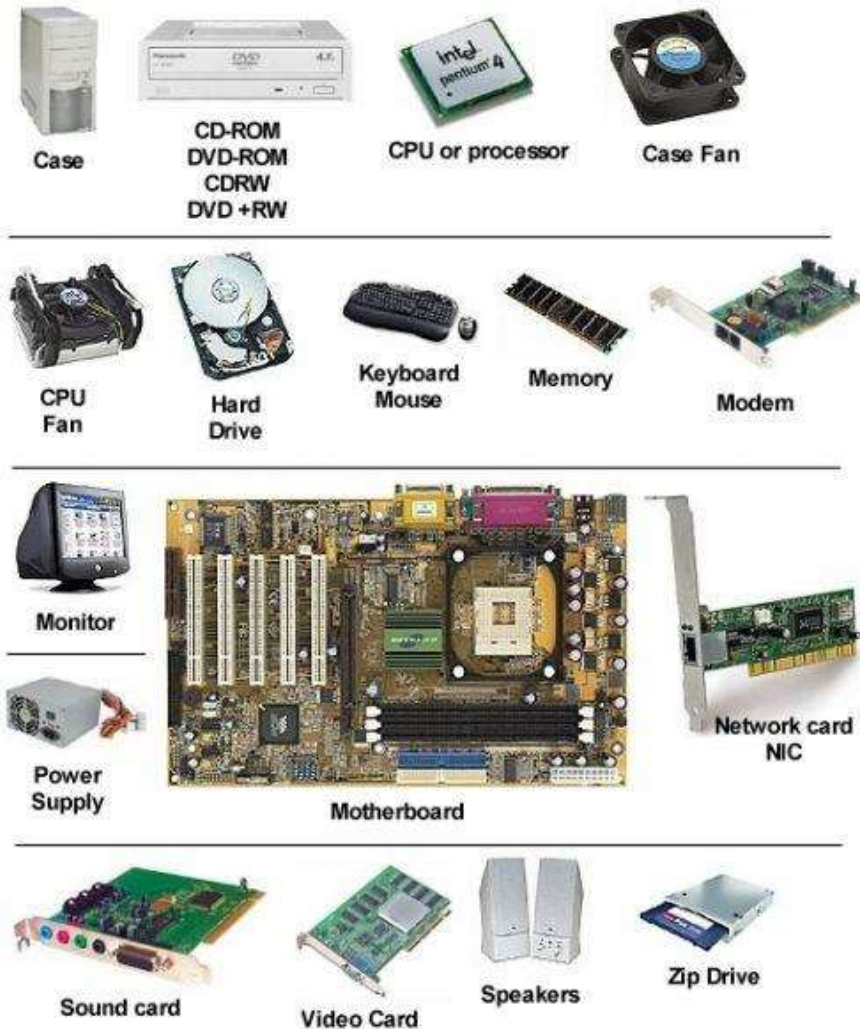
Was ist richtig? <https://pingo.coactum.de/301541>

- a. Die primäre Aufgabe eines BS ist es, die *Ausführungsgeschwindigkeit* der Anwendungsprogramme zu erhöhen
- b. Batch-Processing würde man in interaktiven Computerspielen verwenden, um die Effizienz der Rechenauslastung zu steigern
- c. Ein BS erleichtert wesentlich die *Programmierung* von Anwendungsprogrammen
- d. Die ersten Versuche mit *Timesharing* scheiterten, da ältere Hardware keinen Schutz zwischen den Anwendungsprogrammen untereinander und dem BS ermöglichte



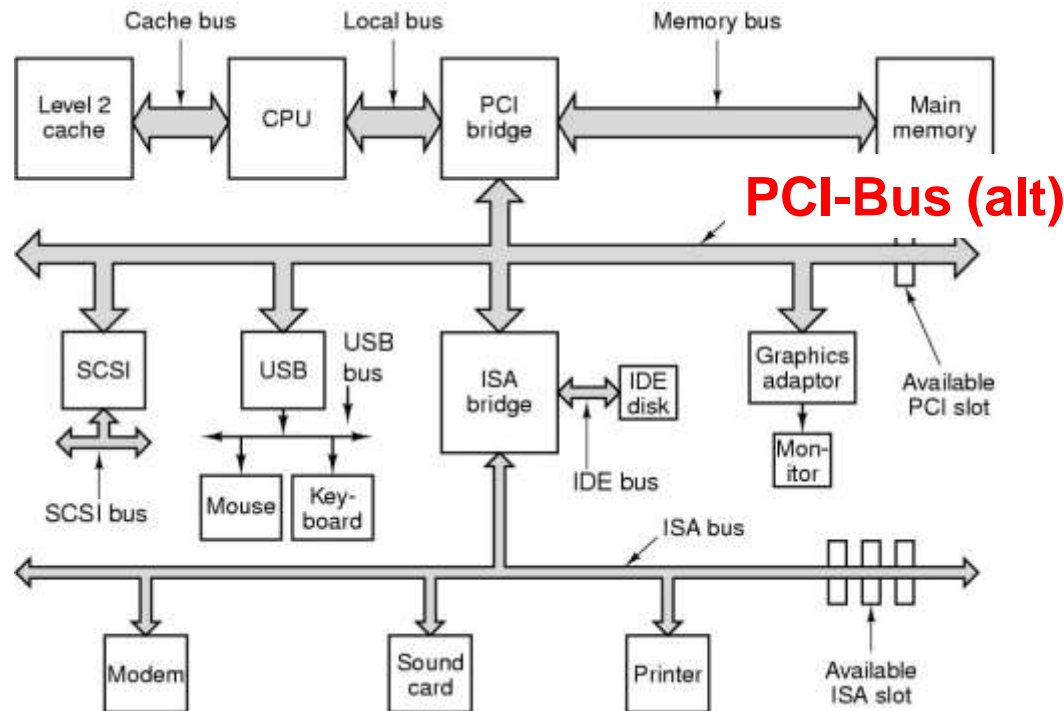
Crashkurs Rechnerarchitektur

Übersicht Rechneraufbau (PC)



- ▶ Ein PC besteht aus vielen Komponenten
- ▶ CPU, Speicher, Busse, Speicherdisks, Ein/Ausgabe-Geräte, ...
- ▶ Die CPU und andere Komponenten arbeiten **gleichzeitig** (und weitgehend **unabhängig**)

Bussysteme



Älterer
Rechner

- ▶ Die Kommunikation zwischen diesen Einheiten passiert über mehrere Arten von (Kommunikations-) **Bussen**
- ▶ Die Systemleistung wird (u. A.) durch ihren **Datendurchsatz** bestimmt

Bussysteme: Entwicklung

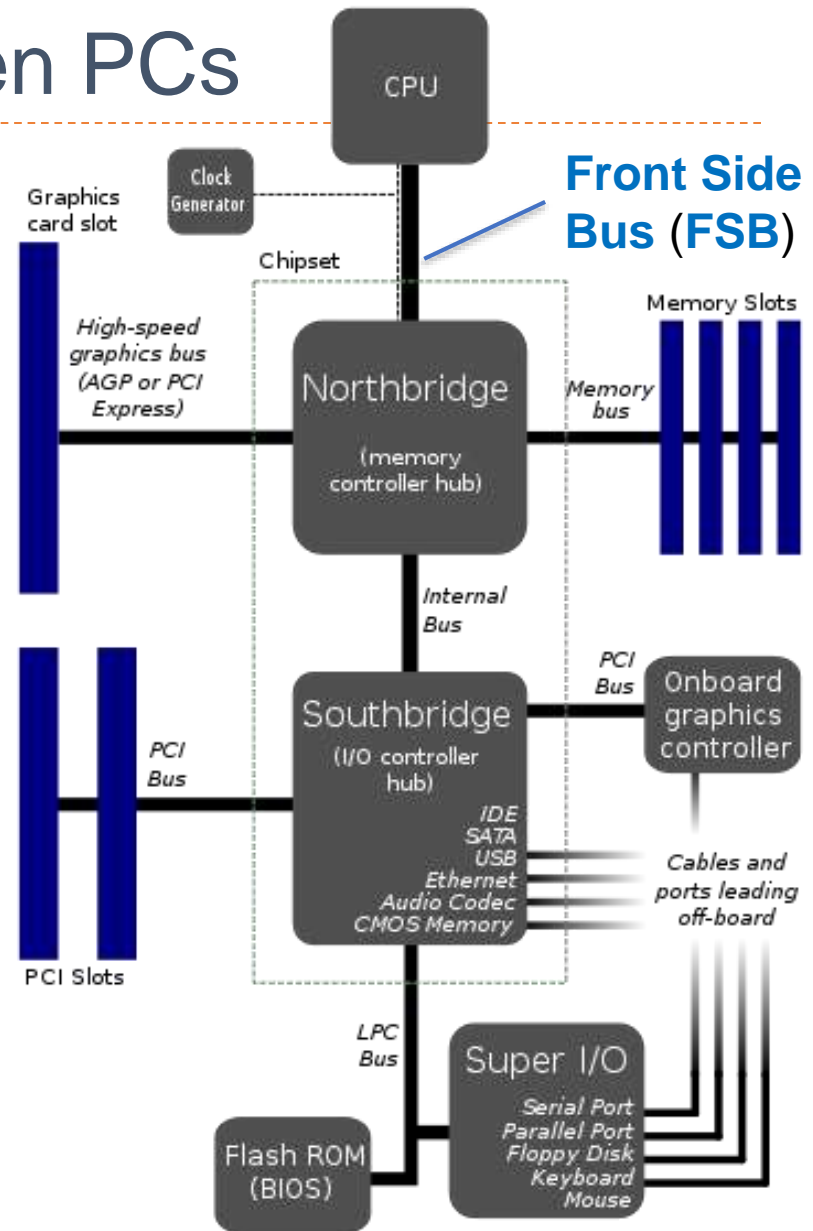
Bus-System	CPUs (ab...)	Takt (MHz)	Transfer-rate (MB/s)
PC	8088	4.7	1
ISA	286	8	4-5
VLB	386	25-50	40/64
MCA	386	10-25	40
EISA	386	8.33	33
PCI	486	25-33	132

- ▶ Der **PCI**-Bus (Peripheral Component Interconnect) und Nachfolger dominierten von 1991 bis ca. 2004
- ▶ PCI-Nachfolger:
 - ▶ AGP (1997)
 - ▶ PCI-X (1999)
 - ▶ **PCI Express** (2004)

- ▶ Moderne Rechner: **Front Side Bus (FSB)** für die Kommunikation zwischen CPU und dem „Rest“
- ▶ FSB Weiterentwicklungen: **HyperTransport**, **QPI**, **InfiniBand**

Bus-Systeme in Modernen PCs

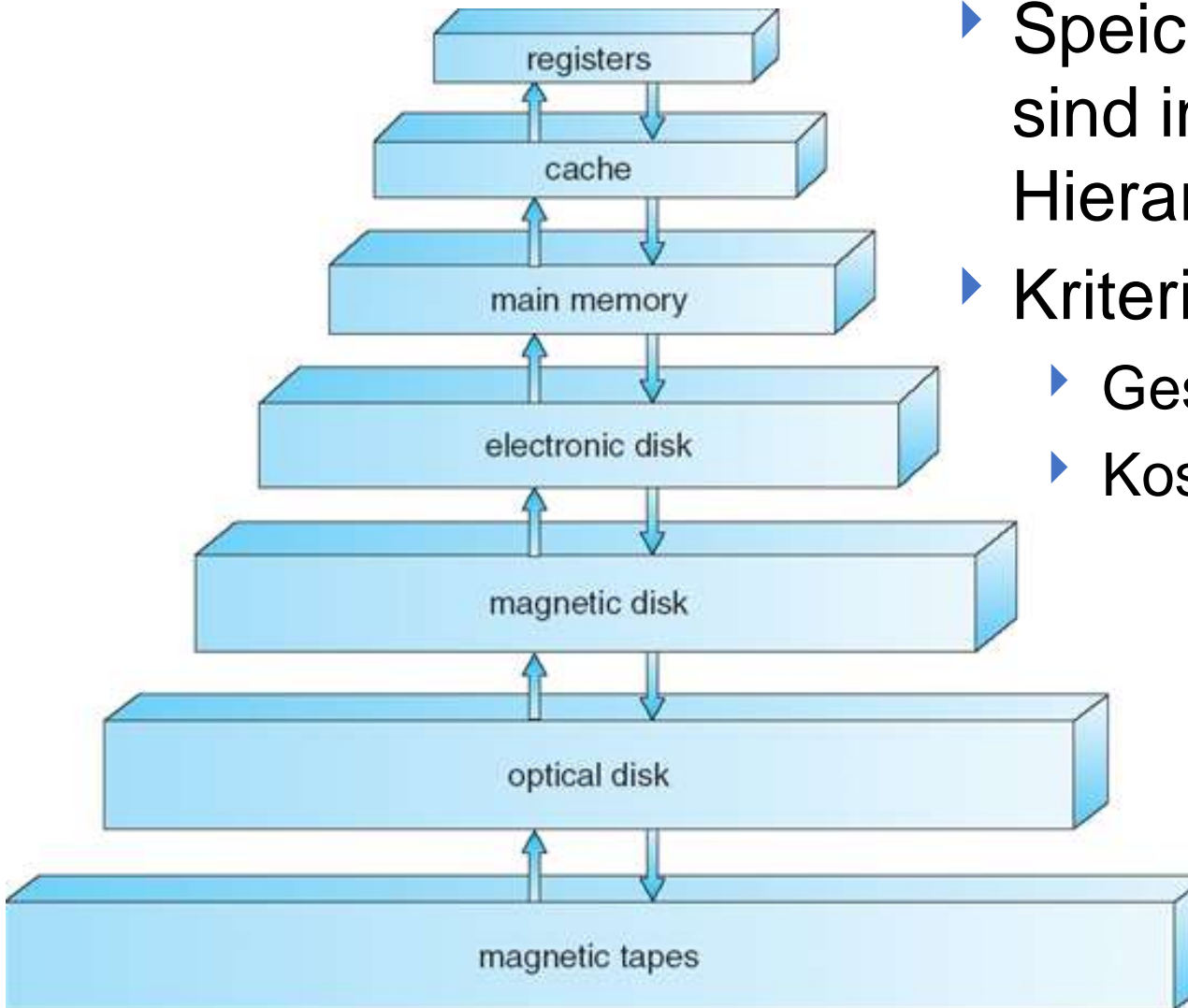
- ▶ Heutige PCs haben spezielle Chips für die Kommunikation zwischen Komponenten
- ▶ **Northbridge**
 - ▶ Schnelle Kommunikation
 - ▶ FSB zwischen CPU und Northbridge-Chip
- ▶ **Southbridge**
 - ▶ Für langsamere Geräte, z.B. USB, Keyboard



E/A-Geräte und Unterbrechungen

- ▶ Jedes Ein-/Ausgabe-Gerät (**E/A**-Gerät, **I/O**-Gerät) hat einen **Controller** (Steuerungseinheit, Elektronik)
- ▶ Ein solcher Controller besitzt einen lokalen **Puffer**
 - ▶ Die Ein- und Ausgabedaten werden zunächst in diesem Puffer hineingeschrieben und ausgelesen
 - ▶ Sobald der Puffer voll ist bzw. neue Daten anliegen, muss die CPU den Puffer auslesen
- ▶ Damit die CPU nicht ständig mit „Nachschauen“ beschäftigt ist, wurden **Unterbrechungen** (**Interrupts**) eingeführt
 - ▶ = Hardware-Mechanismus, der das laufende Programm unterbricht und eine Funktion des BS abarbeitet

Speicherhierarchie



- ▶ Speicher-Systeme sind in einer Hierarchie organisiert
- ▶ Kriterien:
 - ▶ Geschwindigkeit vs.
 - ▶ Kosten (pro Byte)

Leistungen von Speichertypen (Historisch)

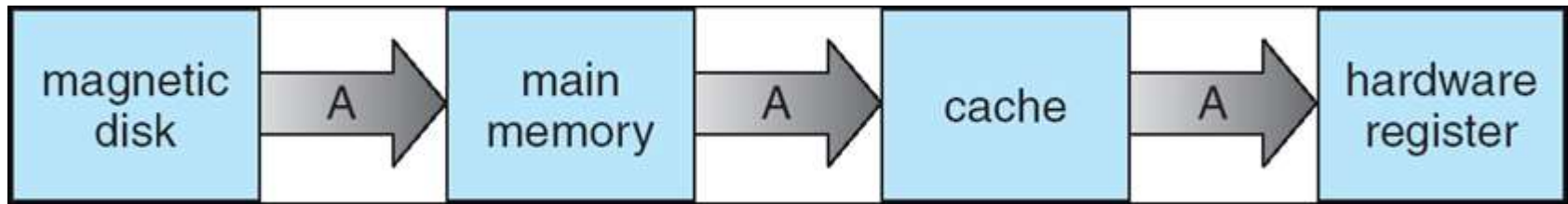
Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

- ▶ Idealer Speicher: schnell, groß und sehr billig
- ▶ Typischer **Trade-off**: Geschwindigkeit vs. Größe

Leistung Moderner Hardware

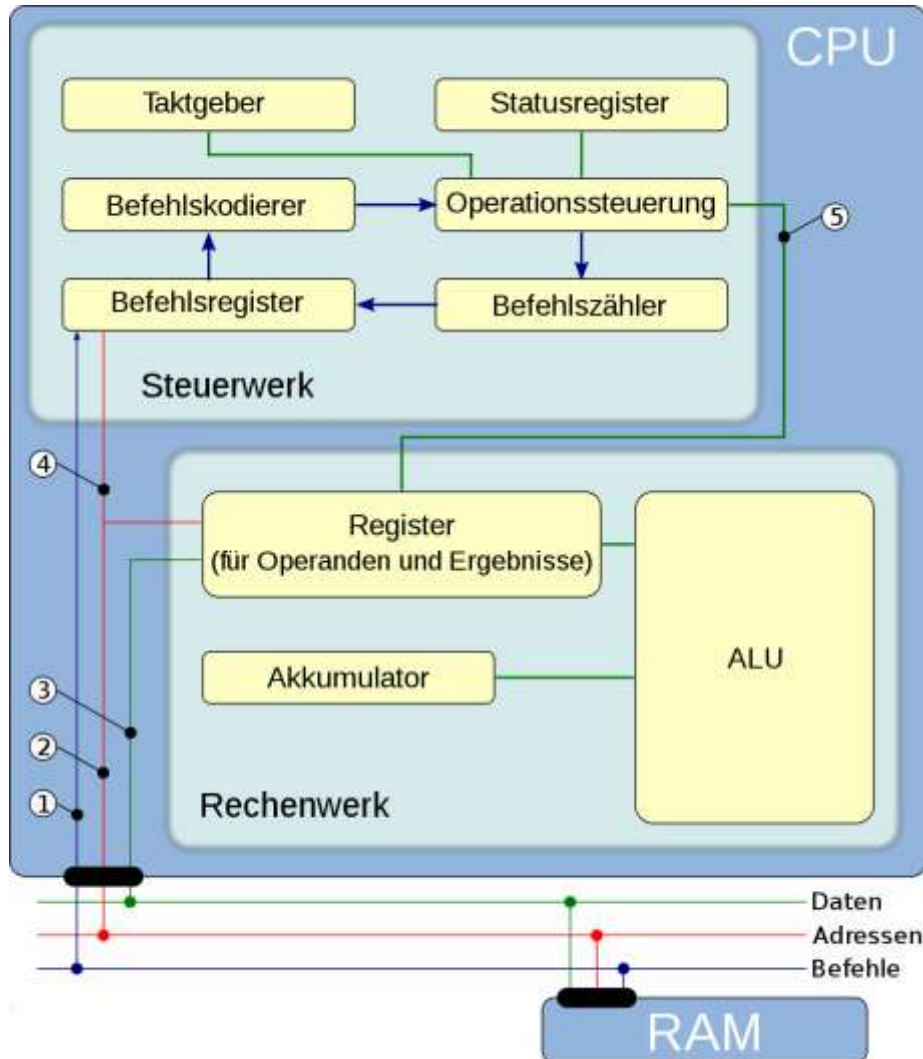
Typ	Technologie	Leistung
Berechnung	Server CPU	24+ cores per die, 10nm, 4GHz, 100M transistors per mm ²
Speicher	DDR4-3200 DRAM	25.6GB/s (288-pin DIMM)
Mainboard	PCI Express 4.0	31GB/s for x16 channels
Netzwerk	Mellanox ConnectX-6	200Gbps
SSD Speicher	Intel Optane P4800X NVMe	2.3GB/s random read/write, < 10 µsec latency
SSD Speicher	Samsung PM1725a NVMe	6.4GB/s sequential read, 1.08MIOPS, 95 µsec latency
Non-volatile Memory	3D XPoint NV-DIMM Technology	< 1 µsec latency (erwartet)

Caches – Definition und Bedeutung



- ▶ **Caches**: Puffer-Speicher – um Daten, die bereits einmal vorlagen, beim nächsten Zugriff schneller lesen zu können
- ▶ Probleme: Multitasking-Umgebungen müssen aufpassen, nur den aktuellsten Wert zu verwenden
 - ▶ Situation in Multiprozessor-Systemen und verteilten Umgebungen ist sogar noch komplexer
- ▶ **Caching** ist ein wichtiges Prinzip in vielen Bereichen der Computersysteme (Hardware, BS, Software, Netzwerke)

Central Processing Unit - CPU



1. Befehle vom Arbeitsspeicher
2. Befehlsadressen vom Arbeitsspeicher
3. Daten von/zur Arbeitsspeicher (o. I/O)
4. Prozessorinterne Adressen
5. Daten von / zur Operationssteuerung

Typen von CPUs

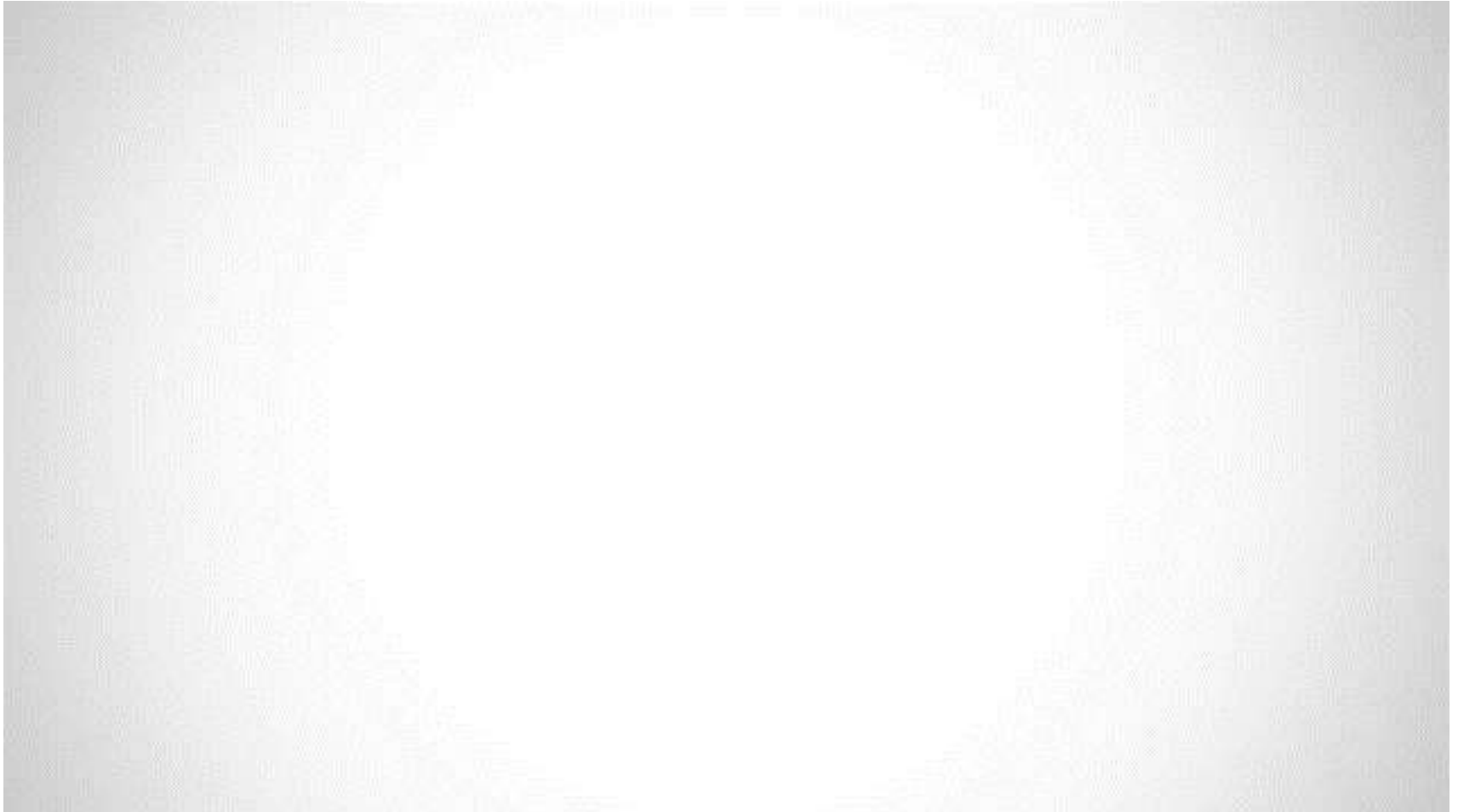
- ▶ **CISC** oder „Complex Instruction Set Computer“
 - ▶ Mächtige aber ggf. langsame Befehle
- ▶ **RISC** oder „Reduced Instruction Set Computer“
 - ▶ Einfache aber schnelle Befehle
- ▶ Bei aktuellen CPUs vermischt

Wichtigste Register der CPU

- ▶ **Allgemeine Arbeitsregister** für Daten und Adressen
- ▶ **Befehlszähler** (program counter)
 - ▶ Enthält Speicheradresse des nächsten Befehls
- ▶ **Kellerregister (stack pointer, SP)**
 - ▶ Zeigt auf das Ende des aktuellen **Stacks** (Keller, Stapel)
 - ▶ Stack enthält die sog. **Rahmen (frames)**
 - ▶ Separater Frame für jede Prozedur, die angesprungen ist, aber noch nicht verlassen wurde
 - ▶ Ein Frame enthält **Eingabeparameter, lokale Variablen** und die **Adresse des Aufrufers einer Funktion**
- ▶ **Statusregister** (Programmstatuswort, program status word)
 - ▶ Enthält Bits, die bei Vergleichsoperationen gesetzt wurden, CPU-Priorität, Ausführungsmodus (mehr dazu), ...

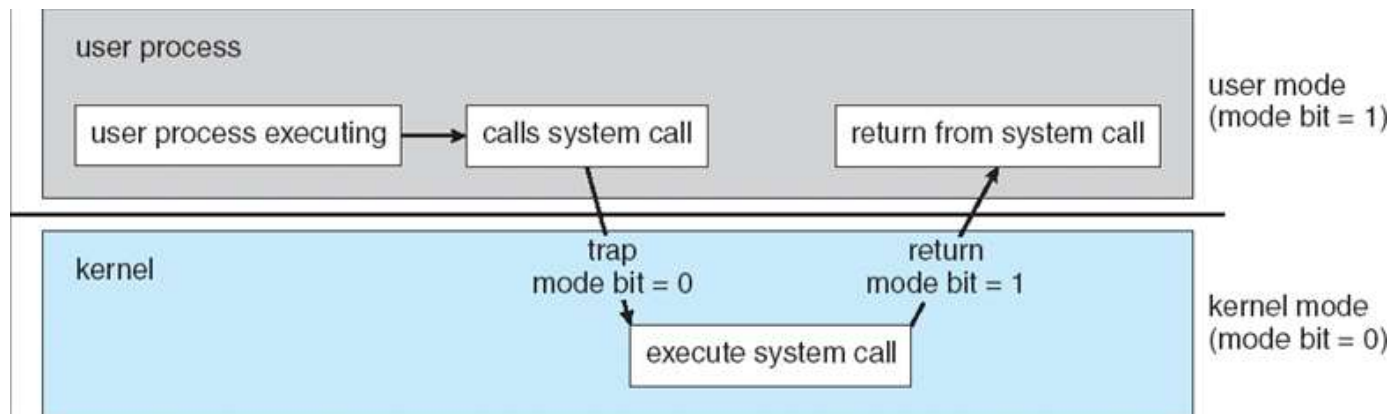
Wichtigste Register der CPU: Video

- ▶ Video [02a]: <https://www.youtube.com/watch?v=RVNXZS-HOgw>
 - ▶ 1:25 bis 3:10 und 8:05 bis 9:20 (min:sec)



Ausführungsmodi

- ▶ Moderne Prozessoren laufen in einem von mehreren **Modi**, die unterschiedliche Ausführungsrechte haben
 - ▶ **Benutzermodus (user mode)**
 - ▶ Manche Operationen gesperrt zum Schutz des BS und anderer Prozesse; Zugriff nur auf begrenzten Speicherbereich möglich
 - ▶ **Kern(el)modus (kernel mode, privileged mode)**
 - ▶ Alle Operationen sind erlaubt; gesamter Speicher zugreifbar
- ▶ Wichtig: Bei einem Interrupt oder Systemaufruf wechselt CPU in den Kernmodus (dabei werden Registerwerte gerettet)



Ausführungsmodi bei IA-32 Architektur

- ▶ Die CPUs ab Pentium (**IA-32** Architektur) kennen vier **Privileg-Ebenen**, die ein Prozess annehmen kann
- ▶ Jede Ebene hat (in HW festgelegte) Rechte für den Zugriff auf Daten und Code-Ausführung
 - ▶ 0: Höchste, dem Betriebssystem-Kern vorbehalten
 - ▶ 1: Für Systemdienste (Teile des BS)
 - ▶ 2: Kundenspezifische BS-Erweiterungen
 - ▶ 3: Anwendungen
- ▶ Kann Prozess auf Ebene k ...
 - ▶ Auf Daten der Ebene $k-1$, $k-2$, ... zugreifen / Code ausführen? Nein
 - ▶ Auf Daten der Prozesse in Ebenen $k+1$ zugreifen? Ja
 - ▶ Den Code „in“ Ebenen $k+1$ ausführen? Nein


Maschinensprache vs. Assembler

- ▶ Eine CPU kann nur **Maschinensprache** ausführen
 - ▶ Eine Folge von Opcodes = Zahlen im Speicher; sie zu verstehen ist wie eine CD durch das Anschauen der Oberfläche zu „hören“
- ▶ Die menschenfreundlichere Form ist der **Assembler**
 - ▶ Zwei Bedeutungen

Assemblersprache: eine Programmiersprache (Sammlung von Mnemonics)

Assembler (DIN 44300): ein Hilfsprogramm, das Assemblersprache in Maschinensprache übersetzt

```
ORG 100h
mov ax, cs
mov ds, ax
mov ah, 09h
mov dx, Meldung
int 21h
mov ax, 4C00h
int 21h
Meldung: db "Hello World"
db "$"
```

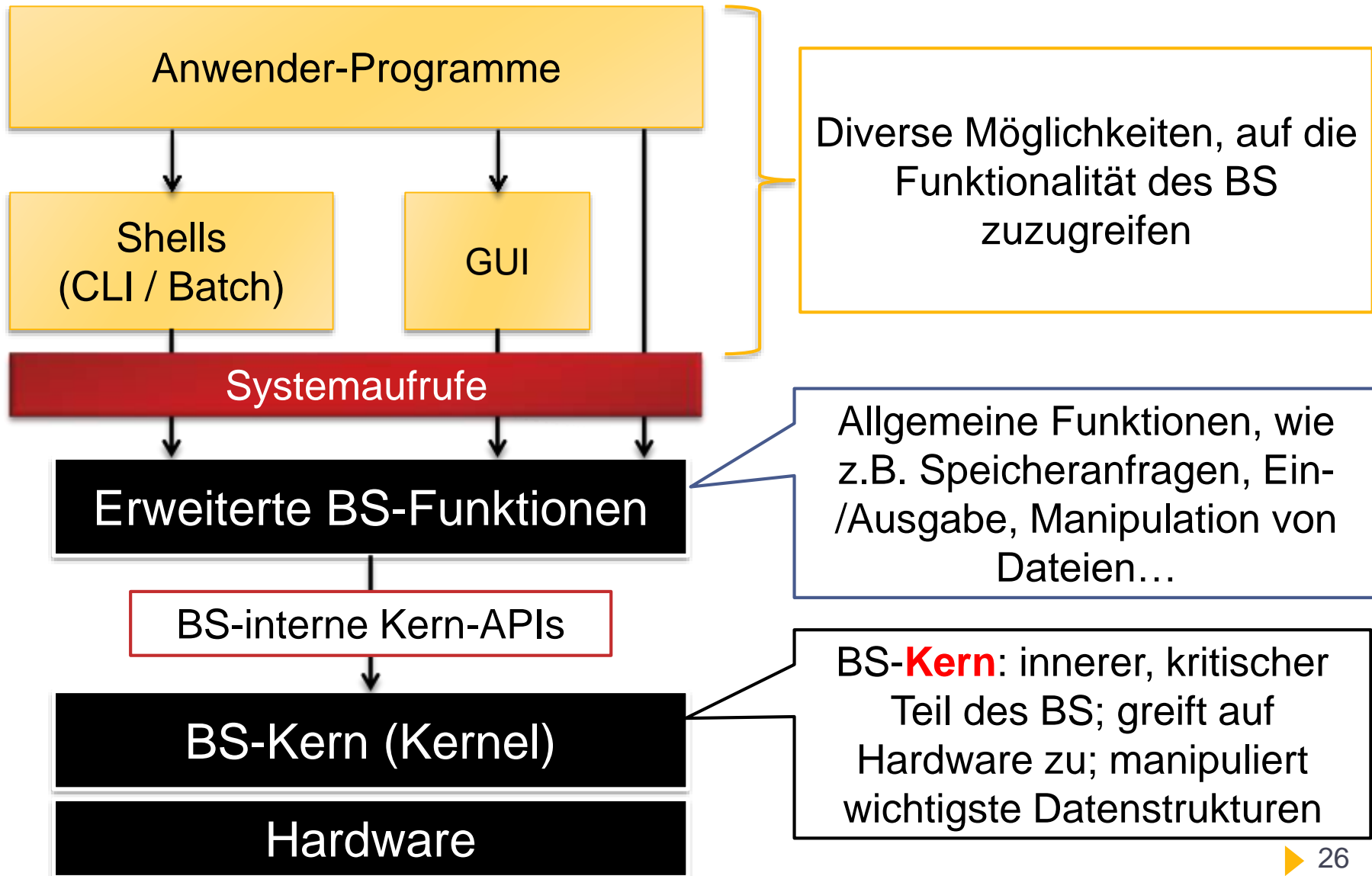


Parameterübergabe via Stack

- ▶ Video „[02b] The call stack“ (0:00 – 2:30 (min:sec))
 - ▶ <https://www.youtube.com/watch?v=Q2sFmqvpBe0>
 - ▶ Einfache Einführung, erläutert die Anrufe zwischen **normalen Prozeduren** (z.B. C-Funktionen untereinander)
- ▶ Videos mit mehr Details:
 - ▶ „Procedures, Video 2: Call stack“,
 - ▶ https://www.youtube.com/watch?v=XbZQ-EonR_I
 - ▶ „Procedures, Video 4: Linux stack frame“ [02x]
 - ▶ <https://www.youtube.com/watch?v=PrDsGldP1Q0>
 - ▶ 0:25 bis ca. 2:15 (Übersicht), Beispiel ab 2:15 (min:sec)
 - ▶ Video “Assembly Primer For Hackers (Part 11) Functions Stack“ (Von 0:30 bis ca. 5:00 (min:sec))
 - ▶ https://www.youtube.com/watch?v=KRajoeVXF_8
 - ▶ Noch eins: <https://www.youtube.com/watch?v=vcfQVwtoyHY>

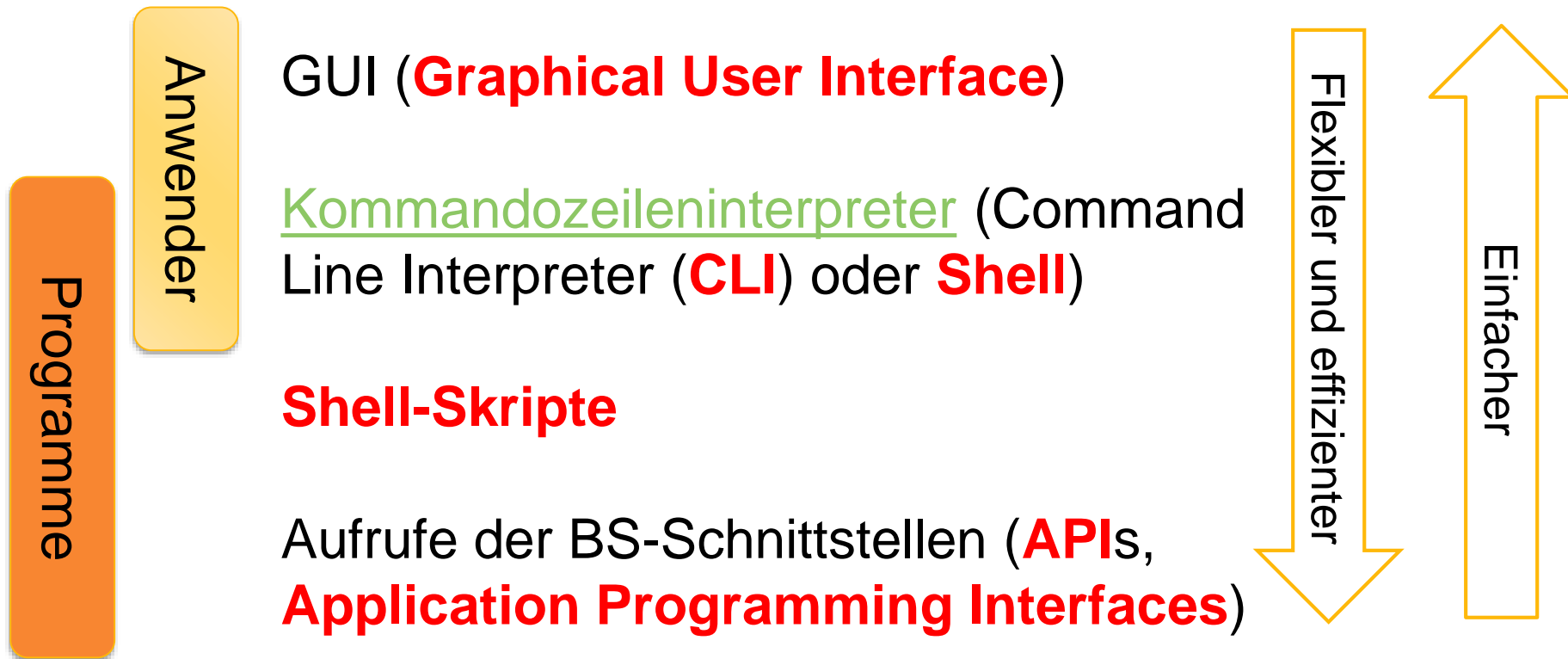
Schnittstellen eines Betriebssystems: Systemaufrufe

Schichten der Software und APIs

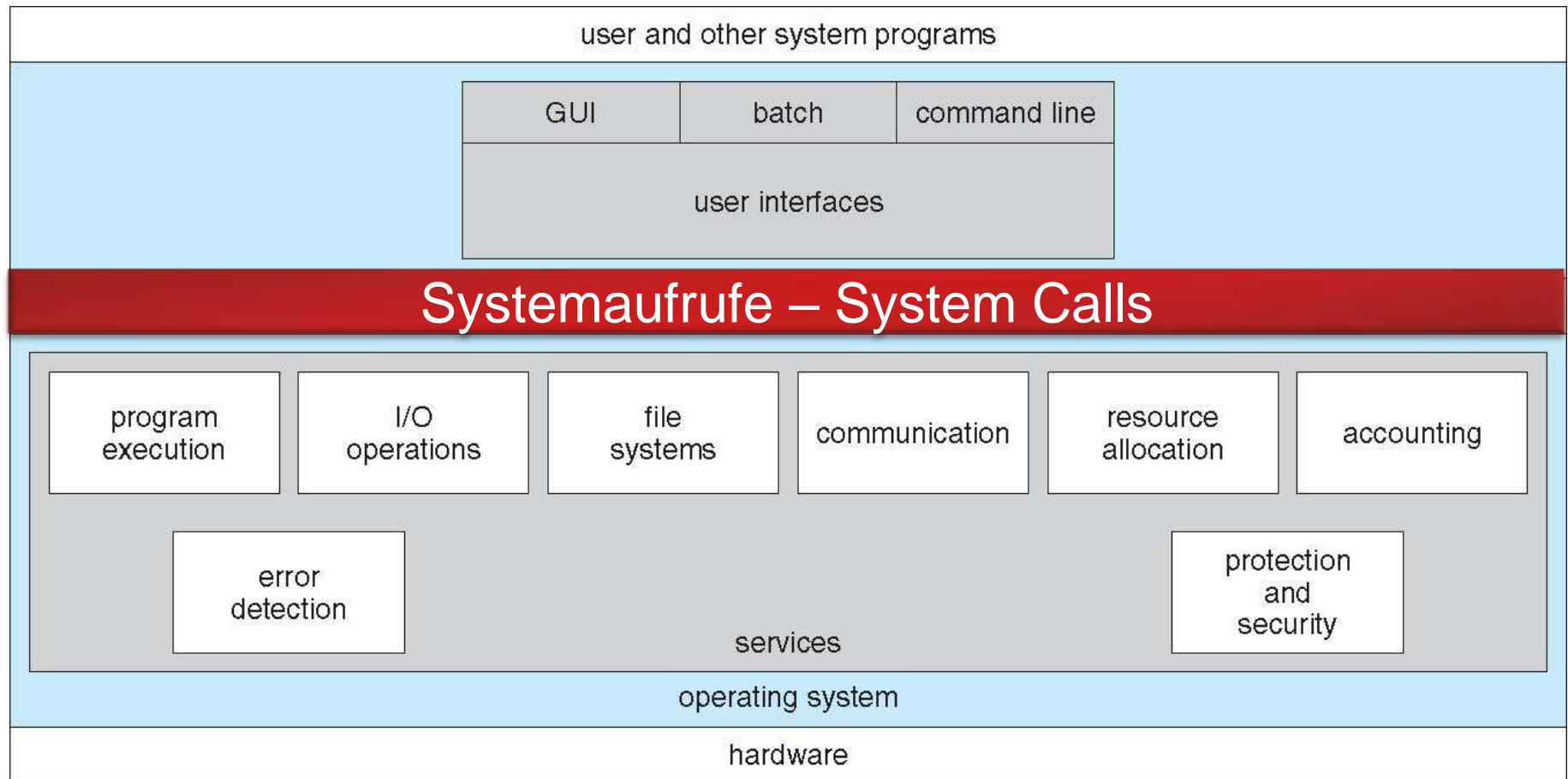


Zugriff auf Funktionen eines BS

- ▶ Wie können Anwender und Programme auf die Funktionen eines BS zugreifen?



Überblick - Zugriff auf die Dienste eines BSs



Systemaufrufe

- ▶ Systemaufrufe eines BS sind typischerweise in einer Hochsprache (C oder C + +) geschrieben
- ▶ Meist wird auf sie von Programmen über Bibliotheksfunktionen zugegriffen (warum?)
- ▶ Die Bibliotheksfunktionen bilden die **APIs** eines BS
 - ▶ Wiederholung: **API** = **Application Programming Interface**
- ▶ Einige populäre Familien von APIs sind:
 - ▶ **Win32**-API für Windows
 - ▶ **POSIX**-API für POSIX-basierte Systeme (einschließlich praktisch alle Versionen von UNIX, Linux und Mac OS X)
 - ▶ **Java-API** für die **Java Virtual Machine (JVM)**

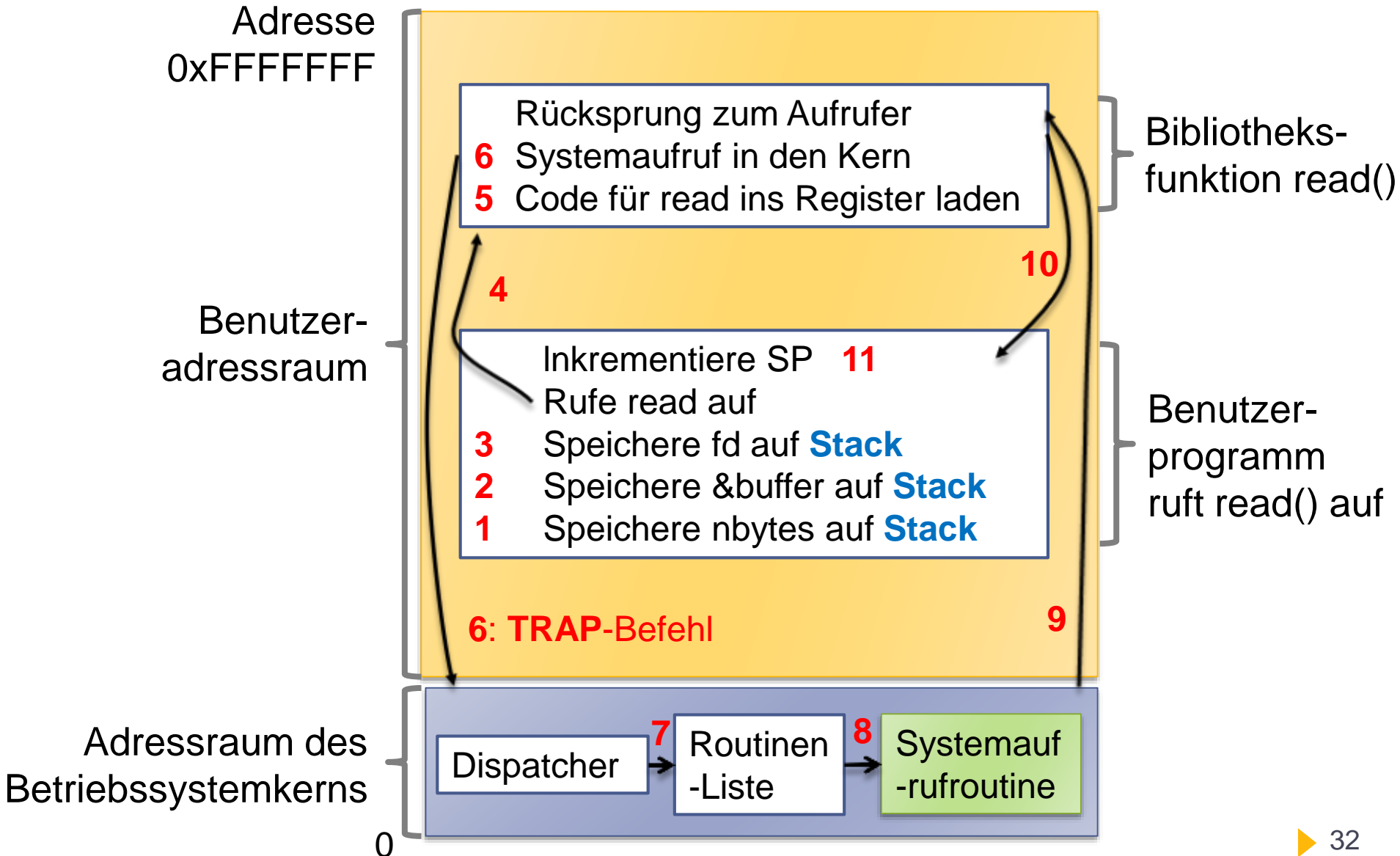
Beispiel eines Systemaufrufs in POSIX (C)

- ▶ Lesen der Daten aus einer Datei
- ▶ **count = read (fd, destination, nbytes);**
 - ▶ **count**: Anzahl der tatsächlich gelesenen Zeichen
 - ▶ **fd**: file descriptor, Zeiger auf Dateibeschreibung
 - ▶ **destination (buffer)**: Zeiger auf Zieldatenbereich
 - ▶ **nbytes**: Anzahl der zu lesenden Zeichen
- ▶ Fehlerbehandlung: bei einem Problem (falsche Parameter, Lesefehler, ...) wird **count** auf **-1** gesetzt und die Fehlernummer in einer globalen Variablen abgelegt, **errno**

POSIX Verzeichnis- und Dateisystemverwaltung

Aufruf	Beschreibung
s = mkdir (name, mode)	
s = rmdir (name)	
s = link (src, target)	
s = unlink (name)	
s = mount (special, name, flag)	
s = umount (special)	

Beispiel eines Systemaufrufs in POSIX



Systemaufrufe in UNIX

- ▶ Video [02c] „Unix system calls (1/2)“
 - ▶ <https://www.youtube.com/watch?v=xHu7qI1gDPA>
- ▶ 2:50 – 4:55 (min:sec): Sprung in den Kernel-Code, Kernel-Code im Speicherraum des Prozesses

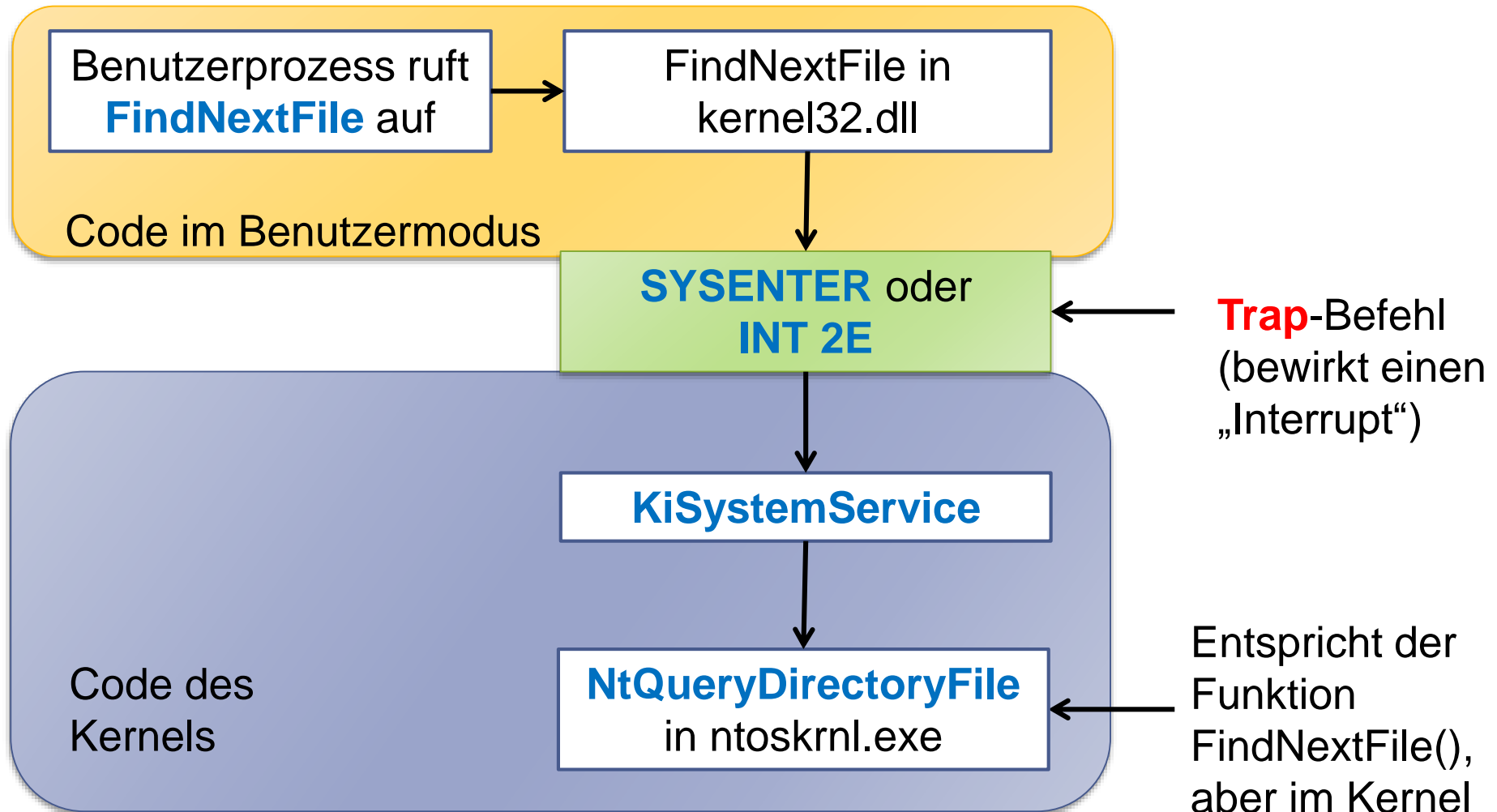
Windows-Subsysteme

- ▶ In Windows gibt es drei **Subsysteme** (= Mengen von Systemaufrufen): **Win32**, **POSIX** und **OS/2**
- ▶ .. Könnte man Linux-Prg unter Windows ausführen?
- ▶ Ja, in Windows 10 (build 16215+) ist das möglich!
 - ▶ <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
 - ▶ Ubuntu, OpenSUSE, SLES, Kali Linux, Debian GNU
- ▶ Auch die graphische Oberfläche von Linux möglich:
 - ▶ Installiere X11-Server (in Windows), z.B. [Xming](#)
 - ▶ Setze in Linux die Umgeb.-Variable: **DISPLAY=:0**
 - ▶ Starte eine beliebige X11-Anwendung (in Linux), z.B.
 - ▶ xclock, xedit, gnome-terminal, ...
- ▶ Windows-Dateisystem unter: **/mnt/c/**

Systemaufrufe in Windows: Beispiel

- ▶ **Ziel:** Auflisten aller Dateien eines Verzeichnisses
- ▶ Der Prozess (= Programm in der Ausführung) braucht mehrere **DLLs = Dynamic Load Libraries**
 - ▶ U.a. **Kernl32.dll**, **User32.dll**, **Gdi32.dll** und **Advapi.dll**
 - ▶ DLLs werden automatisch beim Prozessstart geladen
- ▶ Der Prozess muss zuerst die Funktion **FindFirstFile** aufrufen (aus Kernel32.dll)
- ▶ Bei einem erfolgreichen Aufruf gibt **FindFirstFile** ein **Handle** (= Referenz bzw. Pointer) zurück,
 - ▶ Dieser wird für folgende Aufrufe (durch **FindNextFile**) verwendet

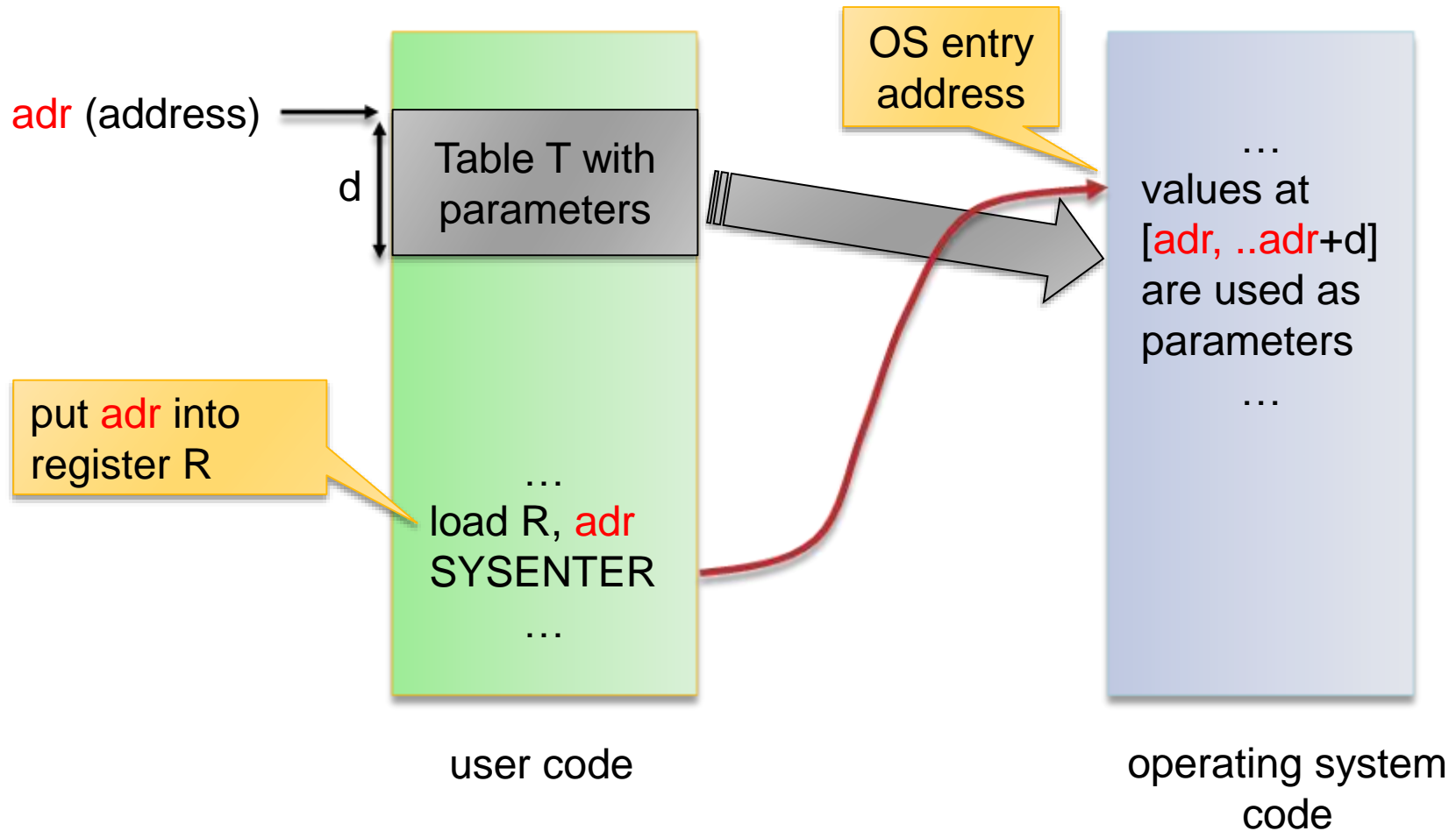
FindNextFile - Ablauf



Parameterübergabe in Systemaufrufen

- ▶ Oft müssen mehr Parameter (als nur die Wahl der Systemroutine) übergeben werden
- ▶ Es gibt drei allgemeine Methoden dazu
- ▶ **1.** Einfachste: Parameter **in den Registern** übertragen
 - ▶ Problem?
- ▶ **2.** Parameter werden in einer **Tabelle** T abgelegt, Adresse von T wird in einem Register übergeben
 - ▶ Dieser Ansatz wird in Linux und Solaris genutzt
- ▶ **3.** Parameter werden auf den **Stack** geschoben und durch das BS herausgeholt (siehe Beispiel zuvor!)

Parameterübergabe durch eine Tabelle (Nr. 2)



Note: the value in CPU-register R „survives“ TRAP and still is **adr**

Parameterübergabe via Stack (Nr. 3)

- ▶ Erinnerung: Videos am Ende des HW-Crashkurses, „Parameterübergabe via Stack“
- ▶ In früheren Betriebssystemen wurde nach einem Trap (**SYSENTER** / **INT 2E**) ein anderer Stack benutzt - der des **Kernel-Modus**
- ▶ BS musste den Wert des Stack Pointers (des **Benutzermodus**) aus den geretteten Registerwerten ermitteln, um die Parameter zu holen

Zusammenfassung Vorlesung 2

- ▶ **Wichtige Hardwarekonzepte**
 - ▶ Interrupts, (DMA, Speicherhierarchie, Caches), CPU-Register, Ausführungsmodi, Assembler
- ▶ **Systemaufrufe**
 - ▶ POSIX: Beispiel des Aufrufs von `read()`
 - ▶ Systemaufrufe in Windows
 - ▶ Parameterübergabe
- ▶ **Literatur**
 - ▶ Silberschatz et al., Kapitel 1 und 2
 - ▶ Tannenbaum, Kapitel 1
 - ▶ Wikipedia

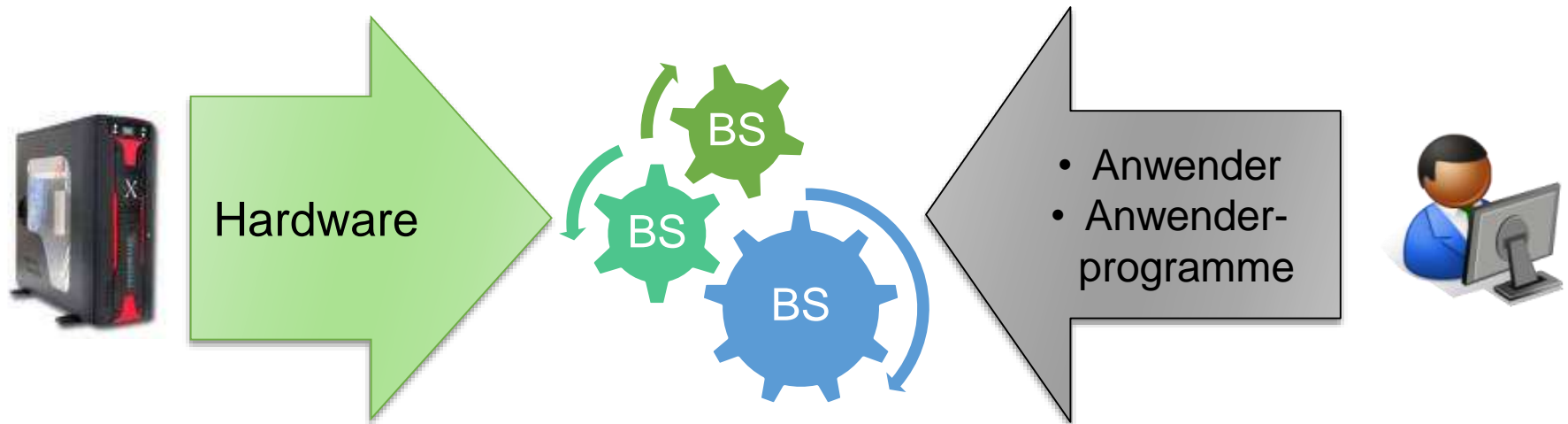
Danke schön.

Aufgaben der Betriebssysteme

Zusatzfolien

A: BS als eine Erweiterte Maschine /3

- ▶ Eine andere Sicht auf die „erweiterte Maschine“
 - ▶ Software, die zwischen Rechnerhardware und Anwendern vermittelt („acts as an intermediary ...“)

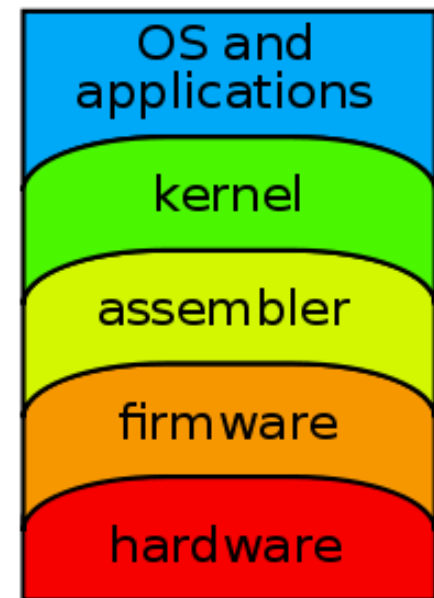


B: Mehrfachnutzung der Ressourcen

- ▶ Es gibt zwei Arten der mehrfachen Nutzung (des **Multiplexings**) – welche?
- ▶ **Zeitliche** Mehrfachnutzung
 - ▶ ein Nutzer / Prozess bekommt die ganze Ressource für eine beschränkte Zeit
 - ▶ Abwechseln der Nutzung eines Prozessors zwischen mehreren Prozessen oder eines Druckers zwischen mehreren Nutzern / Rechnern
- ▶ **Räumliche** Mehrfachnutzung
 - ▶ jeder Nutzer / Prozess bekommt einen Teil der Ressource
 - ▶ Speicher, Festplatte, Datenbank können von mehreren Prozessen / Nutzern zugleich verwendet werden

Ein Wort zu Abstraktionen

- ▶ Abstraktionen sind ein sehr mächtiges Werkzeug in Informatik
 - ▶ Verwendet in BS; Netzwerken; Software
- ▶ *“All problems in computer science can be solved by another level of indirection”*
[mis-quoted: abstraction]
 - ▶ *Butler Lampson, ACM Turing Award Winner (1992)*
- ▶ *“...except for the problem of too many layers of indirection”*
 - ▶ David Wheeler



Rechenarchitektur
als Schichten von
Abstraktionen (Wikipedia)

Schnittstellen eines Betriebssystems Zusatzfolien

Beispiele von Systemaufrufen /1

Windows

CreateProcess()

ExitProcess()

WaitForSingleObject()

CreateFile()

ReadFile()

WriteFile()

CloseHandle()

Unix

fork()

exit()

wait()

open()

read()

write()

close()

Prozess-
verwaltung

Datei-
manipulation

Beispiele von Systemaufrufen /2

Windows

CreatePipe()

CreateFileMapping()

MapViewOfFile ()

SetConsoleMode()

ReadConsole()

WriteConsole()

Unix

pipe()

shmget()

mmap()

ioctl()

read()

write()

Interprozess-
kommunikation

Ein- und Ausgabe bei
STREAM-Geräten
(z.B. Konsole)

Beispiele von Systemaufrufen /3

Windows

SetFileSecurity()

InitializeSecurityDescriptor()

SetSecurityDescriptorGroup()

GetCurrentProcessID()

SetTimer()

Sleep()

Unix

chmod()

umask()

chown()

getpid()

alarm()

sleep()

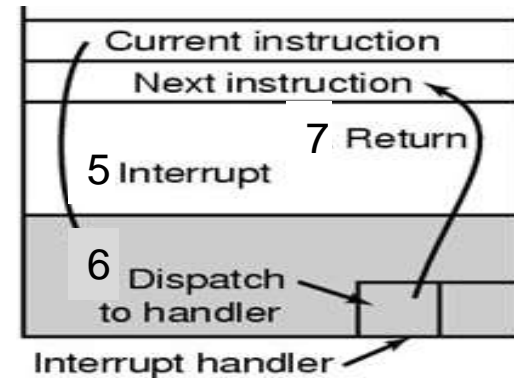
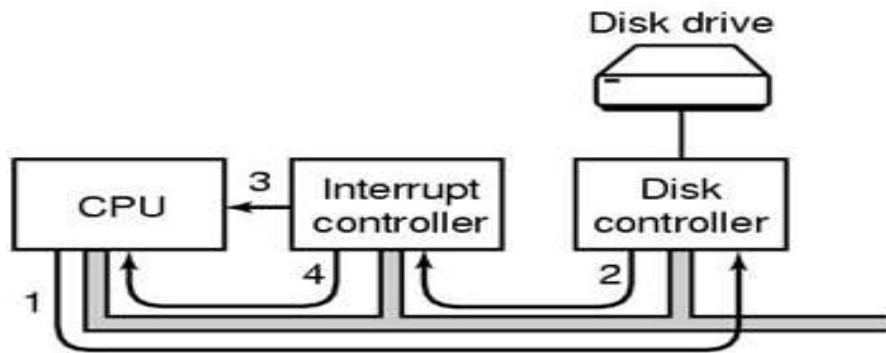
Schutz-
mechanismen
(protection)

Diverse



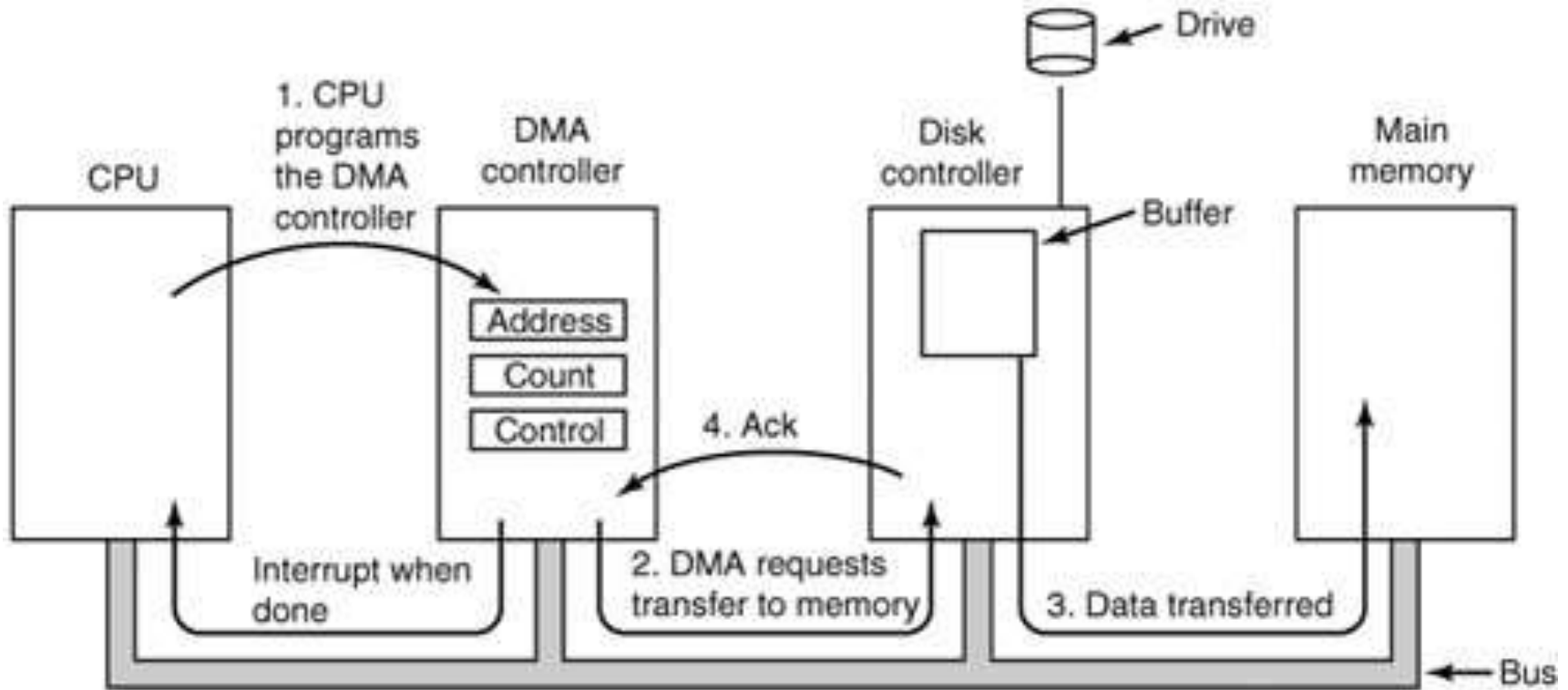
„Crashkurs“ Rechnerarchitektur Zusatzfolien

Unterbrechungen: Ablauf



1. Gerätetreiber (Teil des BS) programmiert den Controller
2. Controller signalisiert dem **Interrupt-Controller** (IC) neue Daten
3. Falls IC den Interrupt annehmen kann, signalisiert es dies der CPU
4. Der IC legt Geräteindex (Quelle) auf den Bus
5. Sobald sich CPU entschieden hat, den Interrupt anzunehmen, holt es aus einer Liste (**Interrupt Vector**) die Adresse der Routine, die zu dem aktuellen Gerät gehört
6. Dann wird diese Interruptroutine (Teil des Treibers) abgearbeitet
7. Die Ausführung kehrt zu dem unterbrochenem Programm zurück

Direct Memory Access (DMA) Controller



- ▶ Automatisiert den Datentransfer vom Puffer des Controllers in den Hauptspeicher (und umgekehrt)
 - ▶ Interrupts werden viel seltener ausgelöst, CPU ist entlastet

Schnittstellen eines Betriebssystems Zusatzfolien

Win32 API

- ▶ Es ist nicht ganz klar, woraus sich Win32 API wirklich zusammensetzt, da mit Windows 2000, XP und Vista viele neue Systemaufrufe eingeführt wurden
- ▶ Die Anzahl der Funktionen ist extrem groß – *einige Tausend*
 - ▶ Viele werden vollständig im **Benutzermodus** abgearbeitet
 - ▶ Unklar, welche davon Systemaufrufe sind
 - ▶ Riesige Anzahl der Funktionen zum Verwalten von Fenstern, Text, Schriftarten, Scrollbalken, Dialogboxen, Menüs und sonstigen GUI-Elementen

Win32 API – Lesen von einer Datei ([Link](#))

Ab welcher Position wird gelesen?	BOOL WINAPI ReadFile (__in HANDLE	<u>hFile</u> ,
		__out LPVOID	<u>lpBuffer</u> ,
		__in DWORD	<u>nNumberOfBytesToRead</u> ,
		__out_opt LPDWORD	<u>lpNumberOfBytesRead</u> ,
		__inout_opt LPOVERLAPPED	<u>lpOverlapped</u>);

- ▶ **file** – **Handle** (d.h. Referenz + Daten) der zu lesenden Datei
- ▶ **buffer** - Puffer, in den das BS die Daten schreibt
- ▶ **bytesToRead** - die Anzahl der Bytes, die in den Puffer eingelesen werden sollen
- ▶ **bytesRead** – Anzahl der Bytes, die bei der letzten Operation gelesen wurden
- ▶ **lpOverlapped** – falls null, blockiert der Aufruf, bis alles gelesen wurde; sonst sog. **asynchroner** Aufruf (“overlapped I/O”)
 - ▶ Asynchron = der Aufruf ist nicht blockierend und die Datei wird “im Hintergrund” gelesen

APIs - Win32 – vs. POSIX - Beispiele

POSIX	Win32	Erklärung
fork	CreateProcess	Erzeugen eines neuen Prozesses
waitpid	WaitForSingleObject	Warten auf das Ende eines Prozesses
execve	-	CreateProcess ersetzt fork + execve
exit	ExitProcess	Ausführung beenden
open	CreateFile	Erzeugen einer Datei oder Öffnen einer existierenden Datei
close	CloseHandle	Datei schließen
lseek	SetFilePointer	Dateizeiger bewegen
stat	GetFileAttributesEx	Dateiattribute erfragen

Portabilität = Plattformunabhängigkeit

- ▶ Das Aufwand des Umschreibens der Software auf verschiedene BS ist erheblich (siehe read / ReadFile)
 - ▶ Gewünscht ist **Portabilität** = Plattformunabhängigkeit
- ▶ Dieses Problem wird z.T. auf der Ebene der Bibliotheken und Compiler gelöst
 - ▶ Z.B. Stellt der Standard „ANSI C“ sicher, dass auf allen ANSI-kompatiblen Compilern die C-Standardbibliotheken gleiche Syntax und Semantik haben
- ▶ Auch die Verbreitung der Java Virtual Machine (JVM) beruht z.T. auf dem Wunsch nach SW-Portabilität
 - ▶ Wieder ein Beispiel für eine Lösung durch Indirektion / Abstraktion