

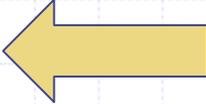
Betriebssysteme Vorlesung 8

Ein/Ausgabe Management,
Scheduling von Festplattenzugriffen

Literatur: Stallings Kapitel 11

J. Quade, E. Kunst; Linux-Treiber entwickeln,
dpunkt.Verlag 2006

Stand der Vorlesung

- ◆ Einführung: Ein Betriebssystem - Was ist das ?
- ◆ Prozeßmanagement
 - Prozesse, Threads
 - Wechselseitiger Ausschluß, Deadlocks & Starvation
- ◆ Speicher Management
 - einfache Speicherverwaltung, virtueller Speicher, Paging
- ◆ Prozessor Scheduling
 - Time-sharing, Realtime, Scheduling bei 1 CPU, mehreren CPUs
- ◆ E/A Management, Festplattenscheduling  HEUTE !
- ◆ Datei Management
- ◆ Netzwerke
- ◆ Sicherheit

Übersicht

- ◆ Struktur und Design von E/A Software im Betriebssystem
- ◆ Techniken zum Puffern von E/A Daten

Wegen der Bedeutung von Festplatten als Speichermedien fokussieren wir auf Festplatten als E/A Geräte

- ◆ Ein/Ausgabe Operationen auf Festplatten
- ◆ Disk Scheduling
- ◆ RAID Systeme (Redundant Array of Independent Disks)
 - Level 0-6 mit zunehmender Sicherheit
- ◆ Disk Cache

Nutzen

- ◆ Sie erkennen das bestimmte Grundkonzepte zur Beherrschung komplexer Aufgaben im SW Design in vielen Bereichen genutzt werden. Hier: Schichtenweise Zerlegung des Problems der E/A Behandlung
- ◆ Sie lernen Ablauf und Optimierungspotentiale beim Zugriff auf Festplatten kennen.
- ◆ Sie lernen Unterschiede der RAID Varianten kennen, deren Auswahl im praktischen Einsatz gravierende Folgen für die Leistung des Gesamtsystems haben kann.
- ◆ Sie erkennen, dass Cacheing eine generelle Technik ist, die zur Leistungssteigerung bei unterschiedlichen Zugriffszeiten auch bei Plattenzugriffen eingesetzt wird.

Verwaltung von Ein-/Ausgabegeräten (I/O Management)

Das Betriebssystem verdeckt weitgehend die Besonderheiten der jeweiligen Hardwareeinheiten eines Rechners, insbesondere die Vielzahl Ein-/Ausgabegeräte (E/A Geräte, I/O Devices) werden hinter weitgehend homogenen Programmierschnittstellen verborgen.

Für die Realisierung benötigt ein OS für jeden Gerätetyp eine passende Steuerungssoftware (Gerätetreiber, Device driver).

Aufgrund der großen Zeitunterschiede zwischen der Bearbeitung in der CPU und in E/A Geräten erfolgt E/A typischerweise asynchron (z.B. Direct memory access DMA), wodurch zusätzliche Kommunikationsmöglichkeiten erforderlich werden (Interrupts).

Wesentliche Ziele im OS Entwurf sind:

Effizienz & Allgemeingültigkeit, d.h. E/A muss möglichst schnell sein und eine einheitliche Sichtweise muß die Programmierung von E/A Funktionen und die Implementierung E/A Software kennzeichnen.

Klassifikation von E/A Geräten

nach Kommunikationspartner

◆ Geräte zur Interaktion mit menschlichen Anwendern

- z.B. Drucker, Terminal: Monitor, Tastatur, Maus

◆ interne Geräte

- z.B. Festplatten, Bandgeräte, Sensoren, Aktoren, Steuerungen

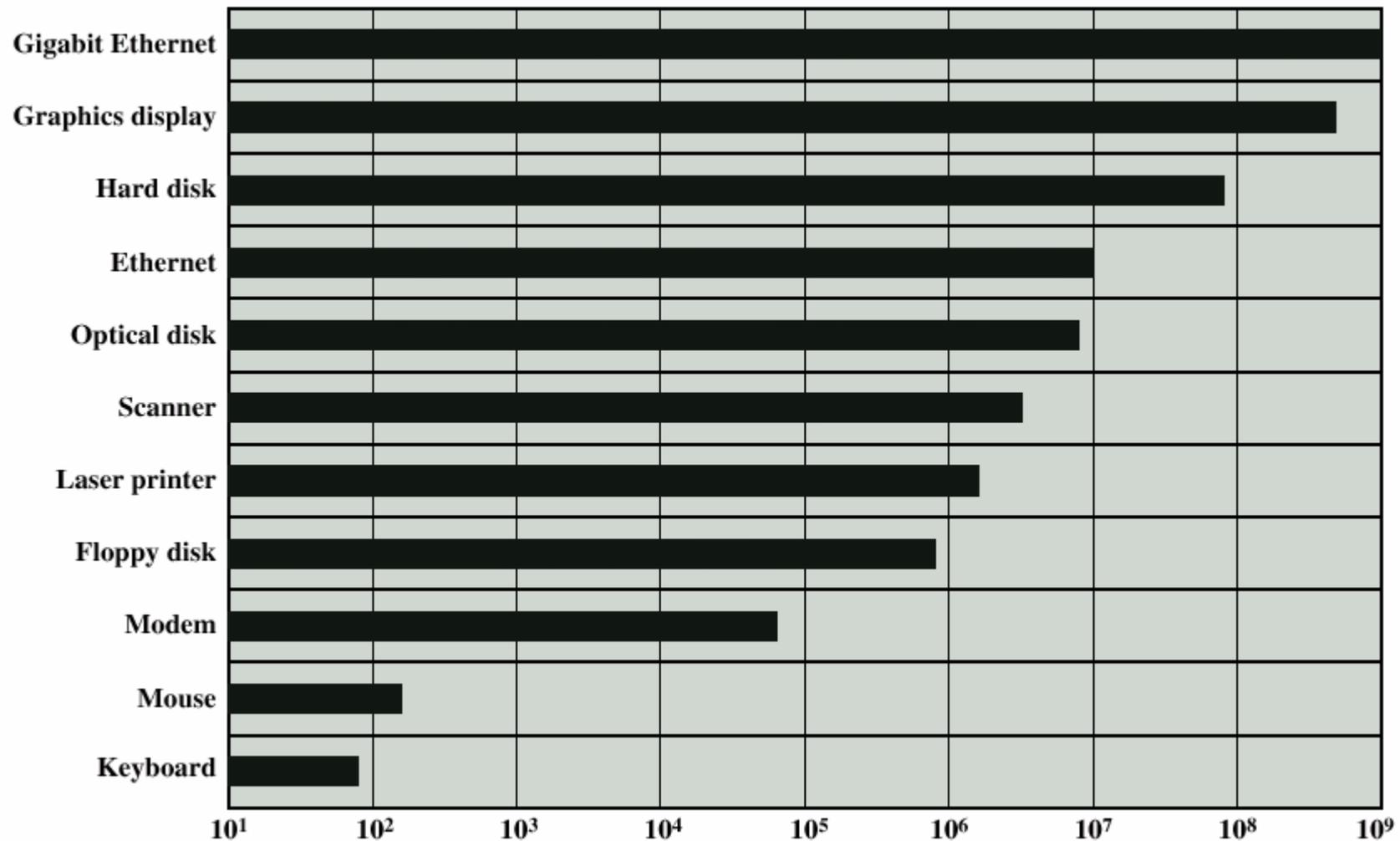
◆ Geräte zur Interaktion mit externen Rechnern

- z.B. Modem, Netzwerkkarten

weitere Unterscheidungskriterien

- Anwendung: Disk für Dateimanagement oder für Virtual Memory
- Komplexität der Steuerung, wird z.T. durch E/A Modul gekapselt
- Einheit für transferierte Daten: streams vs blocks
- Datenformate: character code, parity Konventionen
- Fehler: Art der Fehler, Ursachen, Auswirkungen, Reaktionen
- Datentransferraten: variieren um mehrere Größenordnungen

Typische Transferraten von E/A Geräten in Bit / sec (bps)



Achtung: Kommunikationszeiten unterscheiden sich um mehrere Größenordnungen, sind jedoch deutlich langsamer als die CPU, daher Effizienz von E/A Operationen kritisch!

Entwicklungsstufen für E/A Module

1. CPU kontrolliert das E/A Gerät selbst
2. Programmierte E/A: Ein Controller übernimmt die Ansteuerung des Gerätes, ohne Interrupts wartet die CPU auf Beendigung von E/A Operationen.
3. Interruptgesteuerte E/A, wie 2. nur kann die CPU während der E/A Operation andere Prozesse bearbeiten.
4. Direct Memory Access (DMA): Das E/A Modul erhält die Kontrolle für einen direkten Zugriff auf den Hauptspeicher, Datenübermittlung asynchron zur CPU.
5. E/A Kanal: eigener Prozessor mit speziellem Instruktionsset übernimmt E/A Aufgabe von der CPU, kann dadurch Sequenz von E/A Operationen abarbeiten.
6. E/A Prozessor: Spezial CPU mit eigenem Speicher zu Abwicklung von E/A Aufgaben, kann mehrere E/A Geräte steuern.

zur Begrifflichkeit DMA: 4-6, E/A Kanal: 5-6

DMA

Grundidee: CPU-unabhängige, aktive HW Einheit erhält Zugriff auf den Hauptspeicher

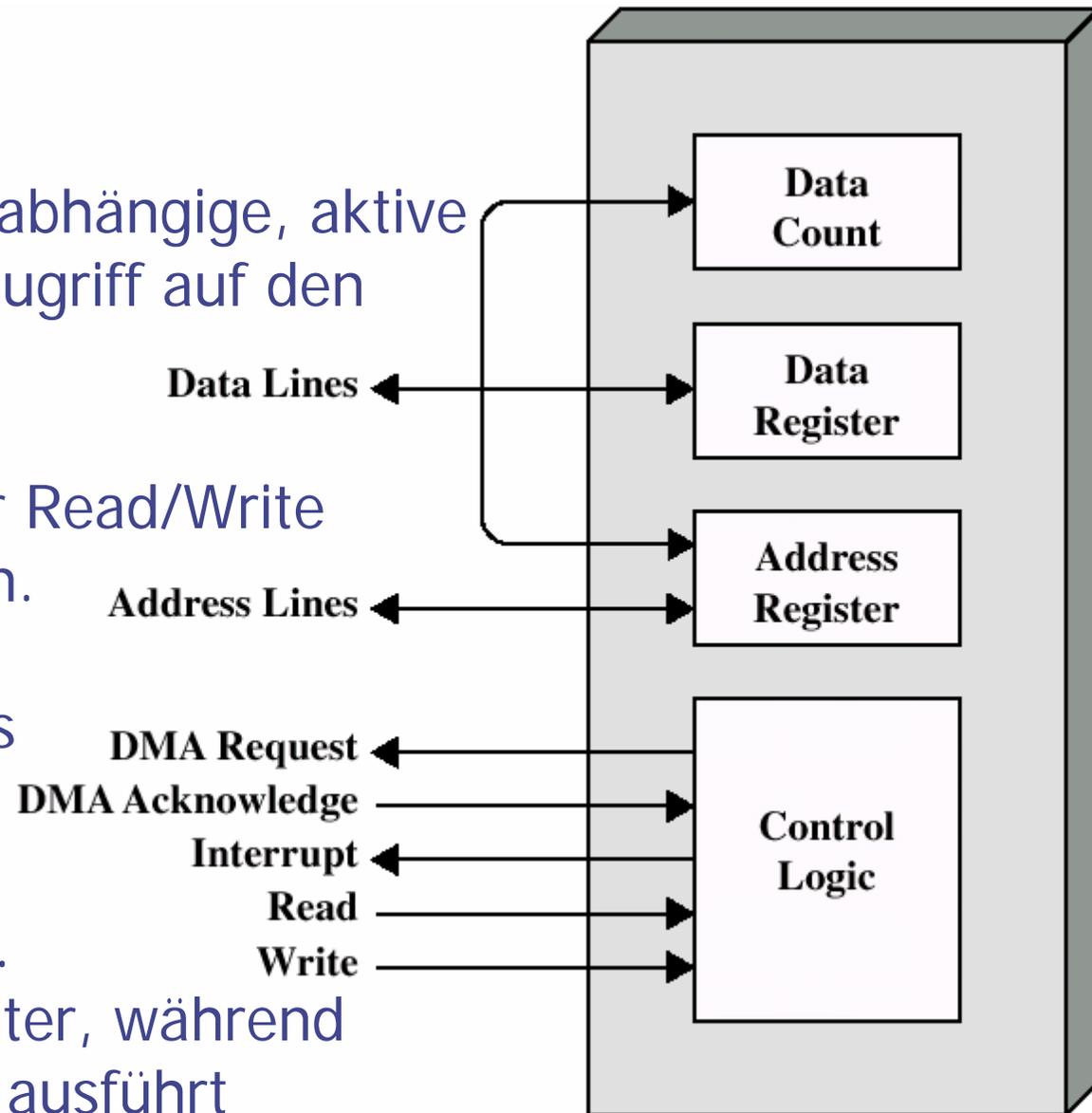
Ablauf:

1) CPU spricht über Read/Write Verbindung DMA an.

Geräteadresse, Anfangsadresse des Speicherbereichs und Anzahl Bytes werden übermittelt.

2) CPU arbeitet weiter, während DMA die Operation ausführt

3) DMA signalisiert per Interrupt die Erledigung der Aufgabe



Gemeinsame Nutzung des Busses: Cycle Stealing

Datentransfer Speicher <-> E/A

über den Systembus

in Konkurrenz zur

CPU

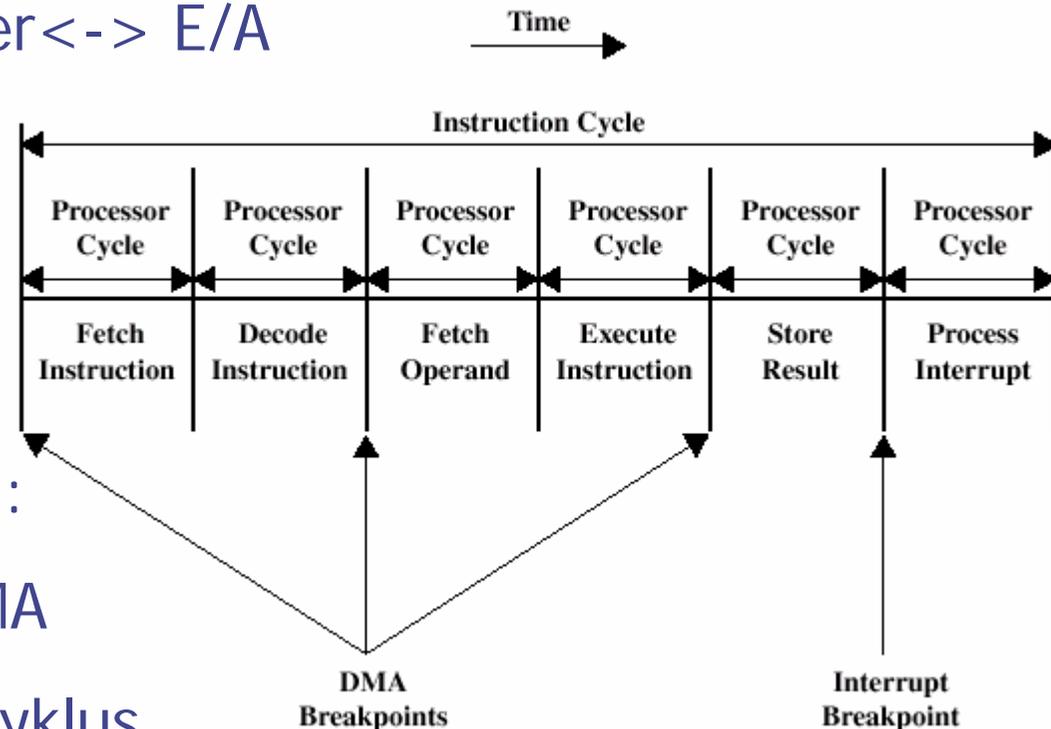
Lösung Cycle Stealing:

die CPU wartet an DMA

Breakpoints im CPU Zyklus

ggfs einen Buszyklus ab.

Die CPU arbeitet dadurch ggfs langsamer, Breakpoints sind jeweils unmittelbar vor einem Buszugriff durch die CPU. Es erfolgt kein Interrupt, kein Modeswitch etc.



Unterschiedliche DMA Hardware Konfigurationen

Offensichtlich wird durch DMA der Bus leicht zum Engpass.

Variante a)

2 Buszyklen je Word, (Anfrage + Transfer)

Variante b)

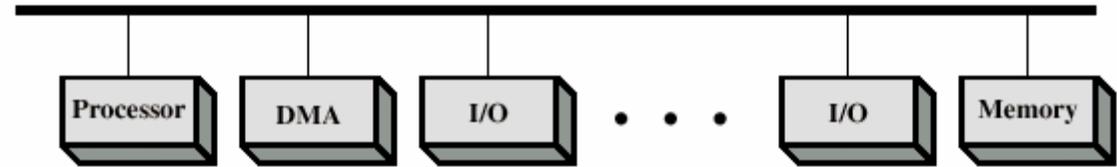
direkte Verbindung DMA <-> E/A Gerät

verringert Last auf dem Bus

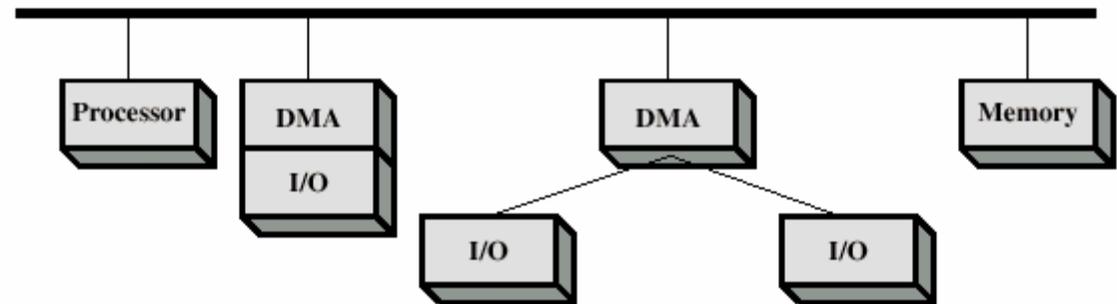
Variante c)

eigener E/A Bus,

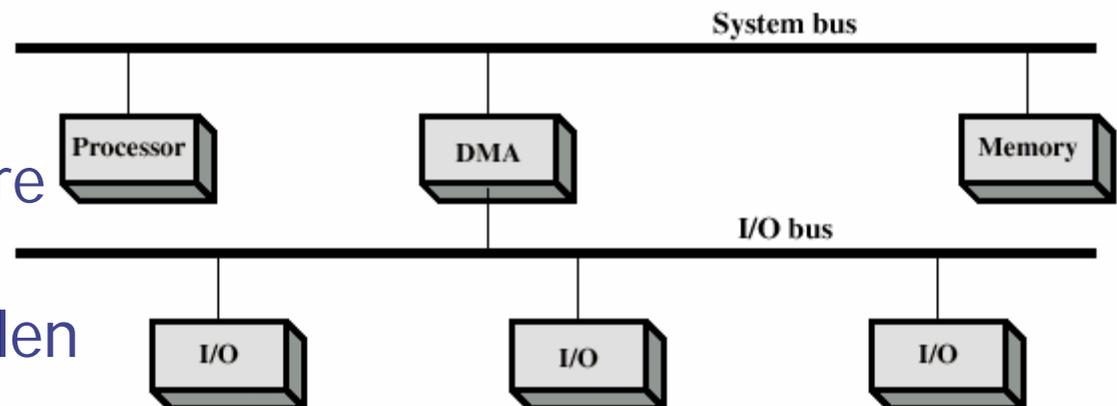
DMA verwaltet mehrere Geräte, reduziert Anzahl E/A Schnittstellen der DMA



(a) Single-bus, detached DMA



(b) Single-bus, Integrated DMA-I/O



Schichtenmodell zur Strukturierung von E/A Software

Anforderungen E/A:

- Effizienz, Zugriffszeiten sollen möglichst gering sein, werden durch asynchrone Behandlung mit DMA kaschiert
- Homogenität (Generality), d.h. alle E/A Geräte sollen vom OS gleichartig behandelt werden können.

Typischer Ansatz aus dem SW Design

Schichtenmodell und Modularisierung:

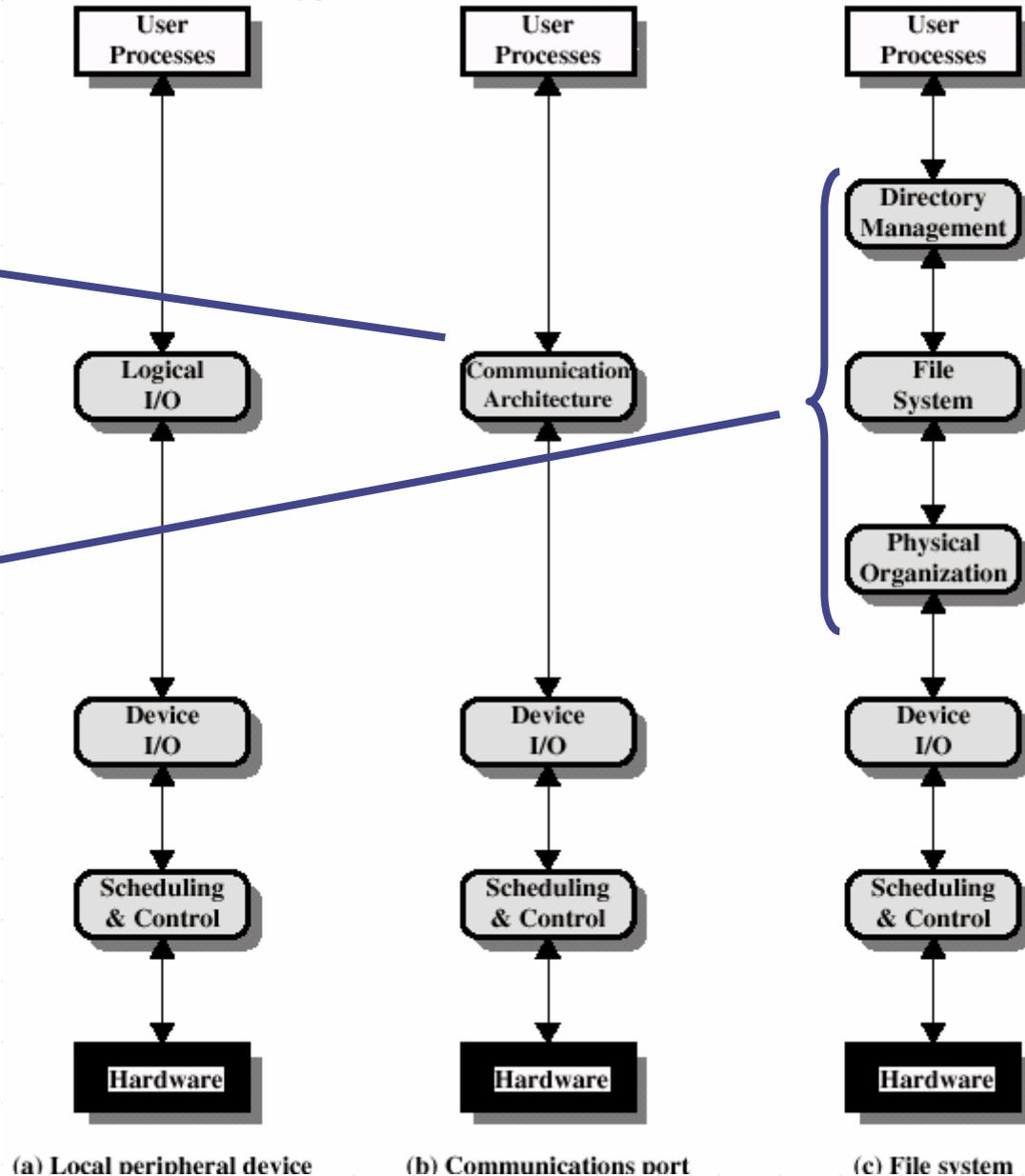
- Zerlegung nach Funktionalität nach Komplexität, Zeitskalen und Abstraktionsniveau
- tieferliegende Schichten haben geringere Zeitskalen und geringeres Abstraktionsniveau, sind näher an realer HW
- Aufteilung in (im Prinzip) 3 Schichten
 - logische E/A Schicht: organisiert E/A für Anwendungsprozeß, erbringt Schnittstelle mit open, close, read, write Operationen
 - Geräte Ein/Ausgabe, Operationen werden in E/A Instruktionscode transformiert, Daten werden ggfs gepuffert
 - Scheduling und Steuerung, interagiert mit E/A Gerät
- einzelne Schichten daraus können selbst wiederum aufgeteilt sein

Schichtenmodell zur Strukturierung von E/A Software

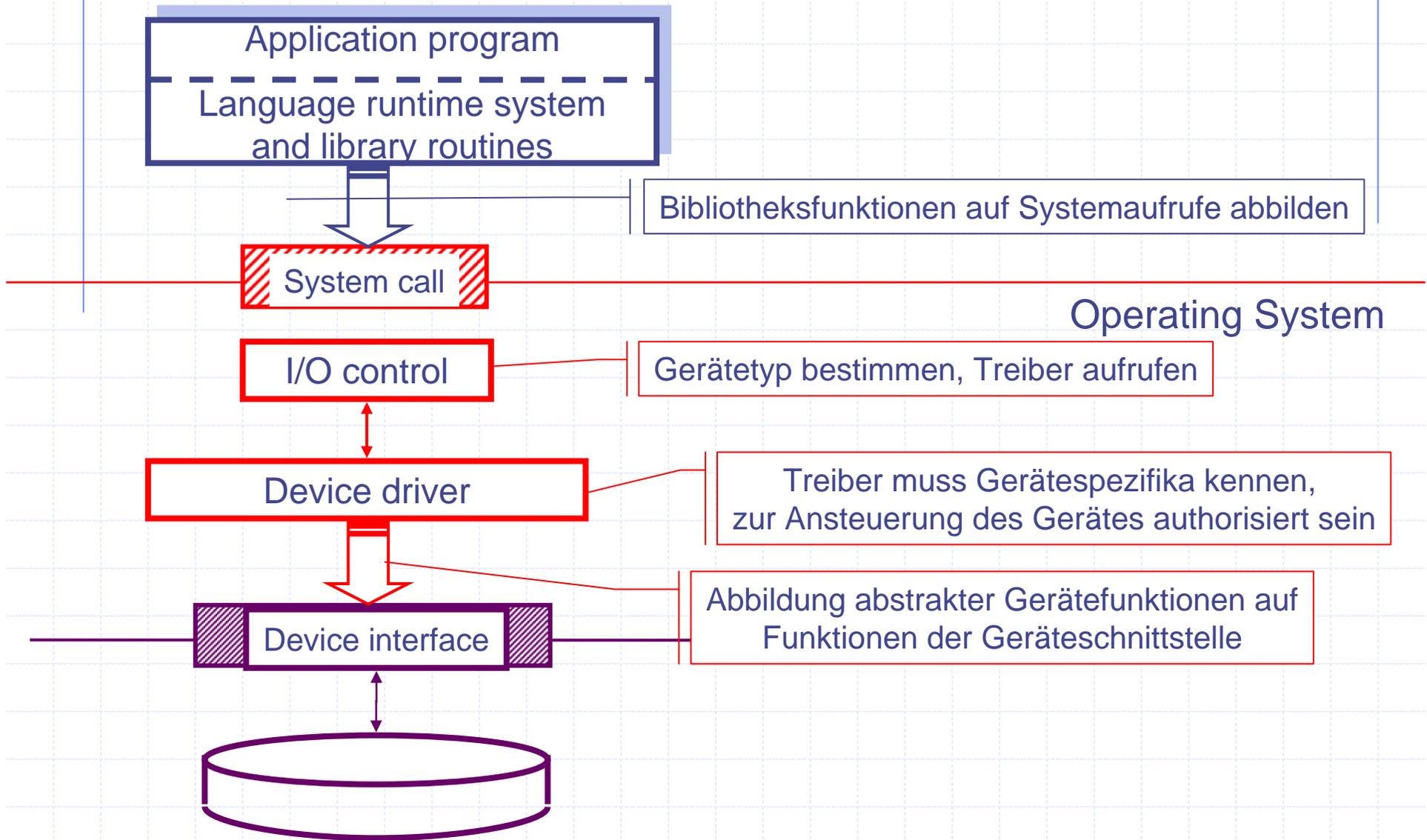
bei einer Netzwerkschnittstelle werden statt logischer E/A eine Schichtenarchitektur für die Kommunikationsprotokolle eingesetzt.

beim Dateisystem wird logische E/A zu

- Directory Management, (Verwaltung eindeutiger Dateinamen, Rechte etc)
- Dateisystem, logische Struktur von Dateien
- Physikalische Struktur Adreßumwandlung, Speicherverwaltung



Typische Schichten eines I/O-Subsystems



Entkoppeln von E/A Operationen und Speicherverwaltung

Bei E/A werden Daten in den Hauptspeicher hinein oder heraus transferiert, der entsprechende Block müsste für die Speicherverwaltung gesperrt werden

- störend, z.B. bei Swapping wg E/A Blockierung

Um Speicherverwaltung und E/A Verwaltung zu entkoppeln, werden Daten gepuffert.

Speicherplatz für Puffer liegt im Speicherbereich des OS.

Je nach E/A unterschiedliche Puffervarianten im Einsatz

- einfacher Puffer
- doppelter Puffer
- zirkulärer Puffer

Man unterscheidet noch

- blockorientierte Geräte, Daten werden en block behandelt, z.B. Bandgeräte, Festplatten, typischerweise Speichermedien
- streamorientierte Geräte, Daten werden als Folge von Bytes behandelt, z.B. Monitore, Drucker, Maus, Kommunikationsports, ...

Varianten zur Pufferung von E/A Eingabedaten

Variante a) kein Puffern

Variante b) einfaches Puffern,
Block wird nach vollständigem
Transfer kopiert,

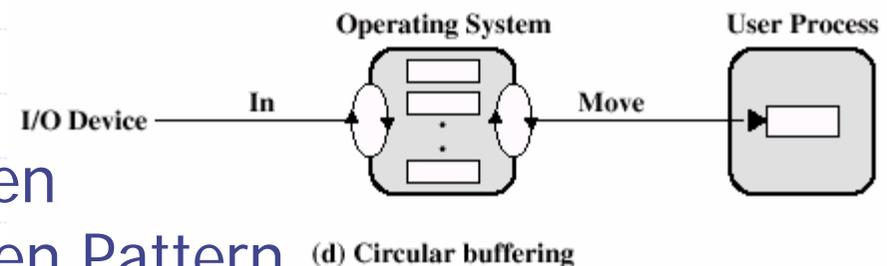
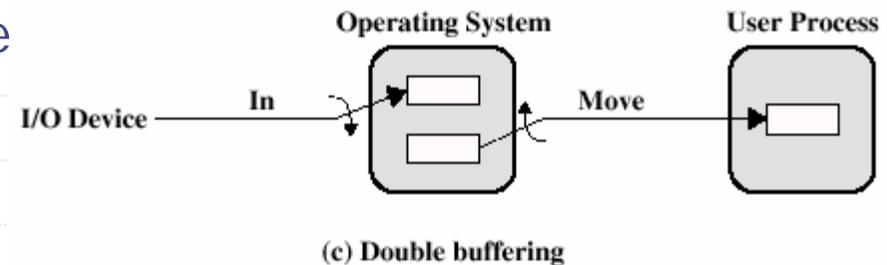
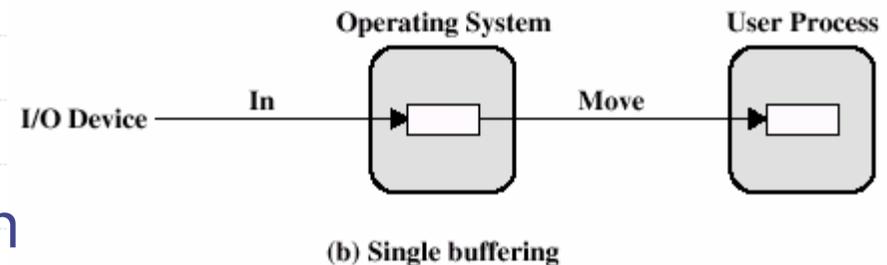
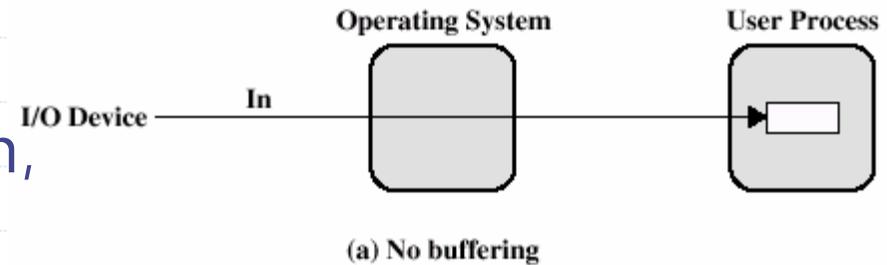
erlaubt z.B. Reading ahead,
leistet Performancegewinn, aber
hat Schwierigkeiten mit Swapping

Variante c) doppeltes Puffern

Gerät und Prozess können
simultan lesen und schreiben ohne
sich zu stören; nur der schnellere
von beiden wird gebremst

Variante d) zirkulärer Puffer

entspricht doppeltem Puffer nur
mit mehr Speicherzellen



Alle Puffervarianten realisieren
ein Produzenten/Konsumenten Pattern

Festplatten Scheduling

- ◆ Festplattenzugriffe sind etwa 4 Größenordnungen langsamer als Hauptspeicherzugriffe und der Abstand wird mit der technischen Entwicklung größer!
- ◆ Wegen des virtuellen Speichers und der dauerhaften Speicherung von Datenmengen auf der Festplatte ist die Performance von Festplatten als E/A Gerät besonders wichtig.
- ◆ Parameter für die Performance eines Plattenzugriffs
 - Suchzeit, 5-10ms: Zeit zur Positionierung des Schreib/Lesekopfes auf dem richtigen Track, Zeiten für Beschleunigung+ Bewegung+Justieren,
 - Rotationsbedingte Latenzzeit, 3ms bei 10000 rpm: Zeit bis passender Sektor den Schreib/Lesekopf passiert; etwa 1/2 Umdrehung
 - Zugriffszeit T: abhängig von der Rotationsgeschwindigkeit

$$T = b / rN \text{ für } b \text{ Bytes}$$

bei N Bytes je Track, r Umdrehungen/s

$$\text{Insgesamt: } T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

Aufbau einer Festplatte



Spur/Track: Ring mit Daten
aufgeteilt in Sektoren gleicher Größe

Virtuelle Geometrie (für Betriebssystem):

x Zylinder, y Köpfe, z Sektoren zB Pentium-PC, max $x=65535$, $y=16$, $z=63$

Alternativ: logical block addressing (LBA): 0,1,..., Sektoren numerieren

Physische Geometrie

(intern für Controller):

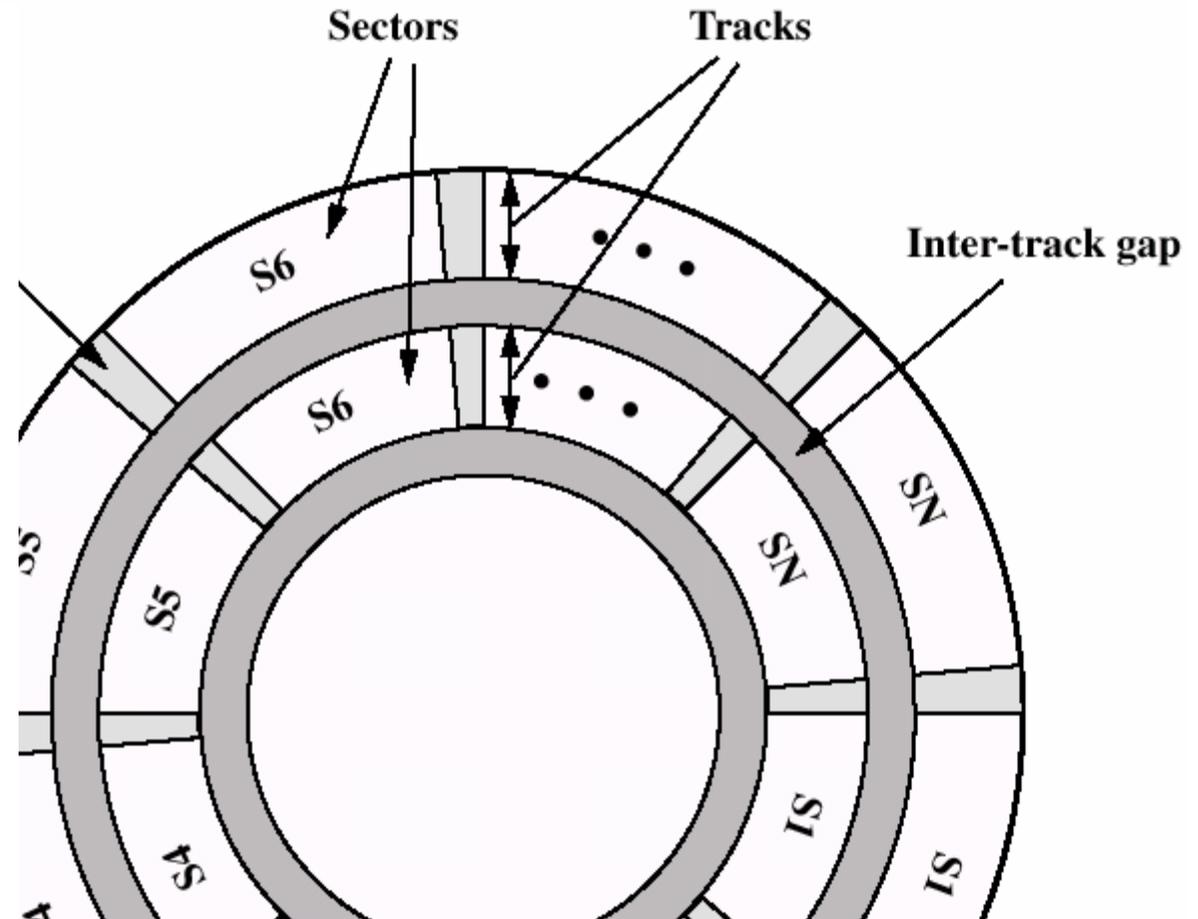
Früher: Anzahl Sektoren
je Spur konstant

Heute: n Zonen
(zB $n=16$),

In einer Zone ist die
Anzahl Sektoren je Spur
gleich. Äußere Zonen
haben mehr Sektoren als
innere.

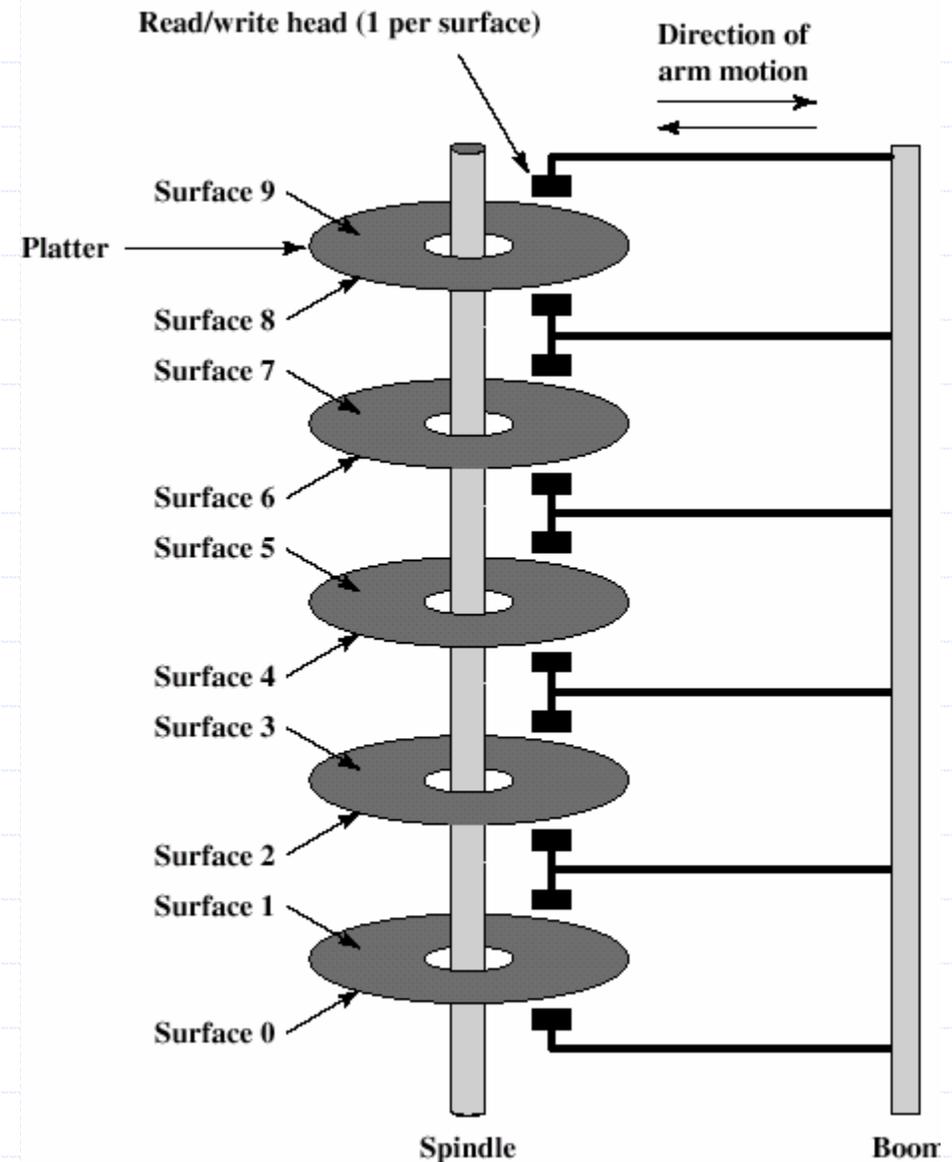
Abbildung:

virtuell->physisch
durch den Controller



Aufbau einer Festplatte

- Platten sind vertikal übereinander angeordnet, werden gemeinsam angetrieben,
- Drehgeschwindigkeit in rpm ist konstant (zB IDE 7200 rpm, SCSI 10000, 15000 rpm),
- Schreib/Leseköpfe werden nur alle zusammen bewegt, lesen jeweils den gleichen Track -> Begriff Zylinder, d.h. alle Tracks mit gleicher Distanz zum Mittelpunkt
- Datenmengen bis zu 500 GB



Zeiten zwischen sequentiell und zufälligem Lesen variieren wegen der Suchzeit beträchtlich !!!

Scheduling Algorithmen für Festplatten

- ◆ Random Scheduling als Benchmark liefert schlechteste Performance wegen häufiger Suchzeiten
- ◆ FCFS ist fair, nutzt jedoch nur sequentielles Auftreten, falls dies zufällig auftritt, bei vielen Prozessen wie Random
- ◆ Prioritätsscheduling, liegt außerhalb der Gerätesteuerung, könnte z.B. kurze interaktive Jobs bevorzugen
- ◆ Last-in-first-out, gut bzgl. Lokalität, jedoch Risiko für Starvation bei hoher Last

Weitere Ansätze wählen gezielt aus der Menge von wartenden Anfragen anhand der Position und Datenmenge aus

- ◆ Shortest-service-time-first wählt Anfrage mit geringster Suchzeit (Entfernung) aus, typische Greedy-Strategie, die häufig gut funktioniert, jedoch globales Optimum nicht garantieren kann. Achtung: Starvation möglich!

Disk Scheduling Algorithmen II

- ◆ SCAN, maximiert Weg in einer Richtung
 - verfolgt die aktuelle Bewegungsrichtung bis zum Ende, nimmt alle Anfragen auf dem Weg soweit möglich mit, dann rückwärts.
 - verhält sich ähnlich zu SSTF jedoch kein Verhungern,
 - nutzt jedoch Lokalität weniger als SSTF und LIFO,
 - bevorzugt Tracks am Rand und Jobs, die zuletzt eintreffen, falls sie auf den aktuellen Track zugreifen.
 - ◆ C-SCAN (circular SCAN)
 - verfolgt 1 Richtung bis zum Rand, dann „Rücksprung“ zum Anfang
- Problem bei STTF, SCAN und C-SCAN: bei vielen Zugriffen auf 1 Track müssen entfernte Zugriffe lange warten („arm stickiness“)
- ◆ N-step-SCAN
 - Warteschlange in Segmente der Länge N unterteilt
 - je Segment von N Jobs in der Schlange wird SCAN angewendet
 - bei $N=1$ wie FIFO, bei großen Werten für N wie SCAN
 - ◆ FSCAN
 - 2 Warteschlangen, von denen eine abgearbeitet wird und in dieser Zeit werden neue Anfragen in die andere Schlange verwiesen

Scheduling Algorithmen für Festplatten

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with $N =$ queue size at beginning of SCAN cycle	Load sensitive

Beispiel: Linux 2.6 Festplattenscheduler

- ◆ 4 zur Auswahl, Festlegung beim Booten oder zur Laufzeit
- ◆ Deadline
 - 2 sorted lists: nach Blocknummern sortierte Listen (read,write)
 - 2 fifo lists: Warteschlangen (read,write), sortiert nach Deadlines
 - Dispatch list: sortiert nach Blocknummern
 - Scheduler verschiebt Anfrage (Batch) aus sorted,fifo -> dispatch
 - 4 Auswahlregeln mit Bevorzugung von reads mit default deadline 500 ms und unter Vermeidung von Starvation für writes
 - Work conserving / Arbeitserhaltend
 - Parameter: read_expire, fifo_batch, write_starved
- ◆ Default: Anticipatory
 - Non-work-conserving, nicht arbeitserhaltend
 - Ziel: Minimiere Anzahl Suchoperationen
 - Grundidee: nach einem Read Request warte kurze Zeit, ob nicht vom gleichen Prozess ein Read Request auf den Folgeblock erzeugt wird.
 - Deadline Scheduler + Anticipation Heuristik + Anticipation Kern
 - Kern liefert Werte für Heuristik: Exit probability, Mittlere Wartezeit
- ◆ Und zwei weitere Noop und CFQ: completely fair queueing

RAID = redundant array of independent disks

Festplatten sind kritische Komponenten

- Zugriffszeiten sind kritisch für die Performance eines Rechners
- Betriebssicherheit ist essentiell

Versuch der Leistungssteigerung durch Parallelisierung:

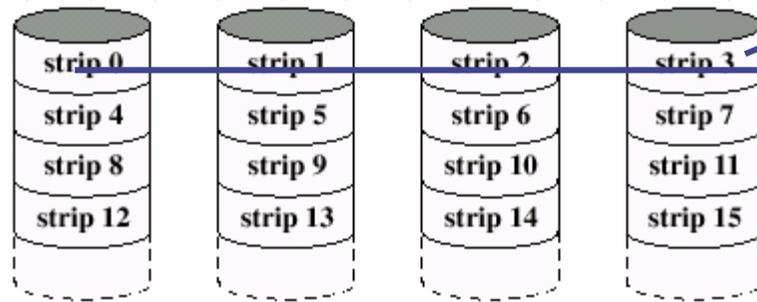
RAID

- RAID System besteht aus N physikalisch eigenständigen Festplatten, die vom OS als 1 Festplatte betrachtet werden
- Zugriffe erfolgen möglichst parallel
- Daten werden als Streifen (Stripes) über die Festplatten verteilt, um parallele Zugriffe auch auf Teile einer Datei zu unterstützen, einzelner Abschnitt eines Stripes auf 1 Platte heißt Block (Strip)
- Parity Information wird intern erzeugt, um bei Ausfall einzelner Platten Daten im laufenden Betrieb rekonstruieren zu können

7 RAID Konfigurationen, Unterschiede bzgl

- Datentransferraten bzgl 1 Anfrage
- E/A Anfrage Raten
- Kosten für Lese/Schreiboperationen bei großen / kleinen Datenmengen
- Ausfallsicherheit
- Kosten

RAID Systeme, Level 0 bis 2

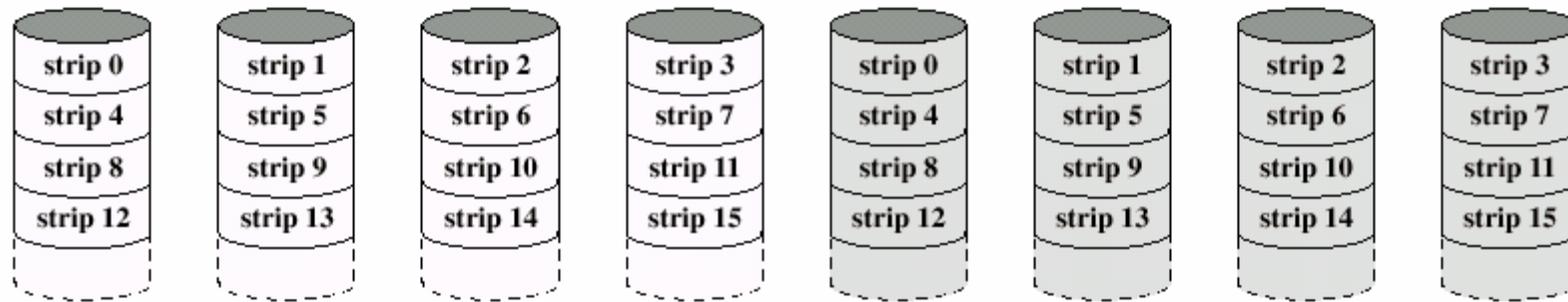


Strip: einzelne Dateneinheit aus k Sektoren

Stripe/Streifen aus $N=4$ konsekutiven Strips, z.B. für 1 große Datei

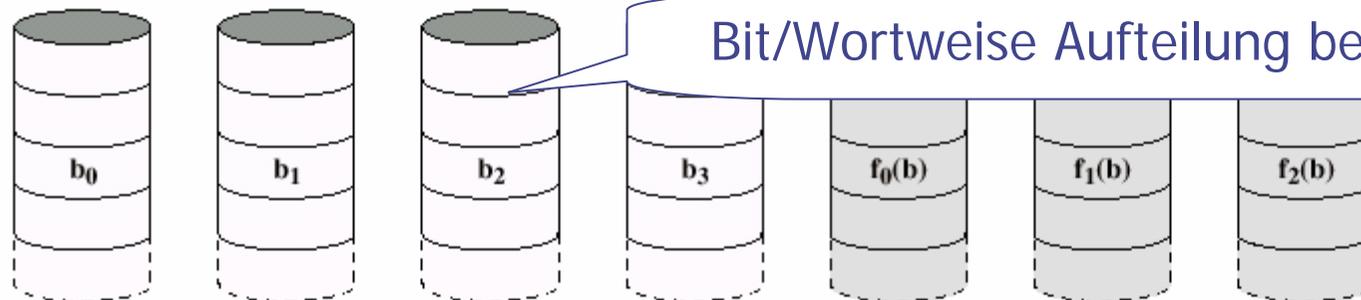
(a) RAID 0 (non-redundant)

paralleles Lesen, Schreiben, keine Redundanz



(b) RAID 1 (mirrored)

paralleles Lesen, einfaches Schreiben, Kosten 8-



Bit/Wortweise Aufteilung bei level 2,3

Kosten 8-(
Overkill

(c) RAID 2 (redundancy through Hamming code)

synchrone Zugriffe, Rechenaufwand, Platzbedarf, Fehlererkennung & -korrektur

RAID-Systeme, Level 3-6

nur 1 redundante Platte
Paritybit überbrückt
den Ausfall einer Platte

$$p_4(i) = x_0(i) \oplus x_1(i) \oplus x_2(i) \oplus x_3(i)$$

bei Ausfall von $x_1(i)$, durch $\oplus x_1(i) \oplus p_4(i)$

$$x_1(i) = x_0(i) \oplus p_4(i) \oplus x_2(i) \oplus x_3(i)$$

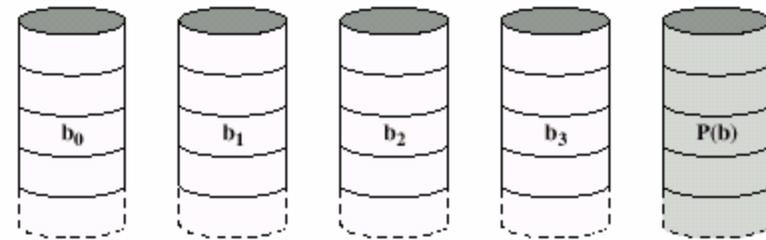
RAID 3: kleine Strips,
synchronisierter Zugriff
hohe Transferraten
wie bei Level 2 Platten synchron

RAID 4-6 unabhängiger Zugriff:
hohe E/A Anfrageraten
große Strips, Schreiben kleiner
Datenmengen teuer

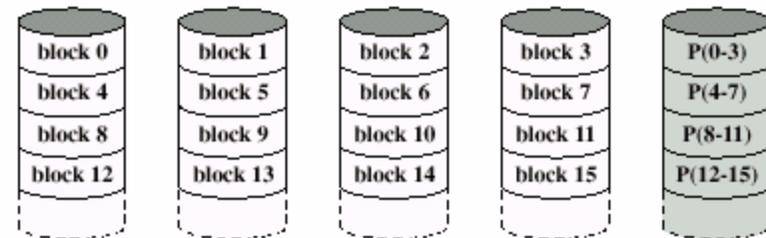
RAID 4: blockweise

RAID 5: Blöcke sind verteilt

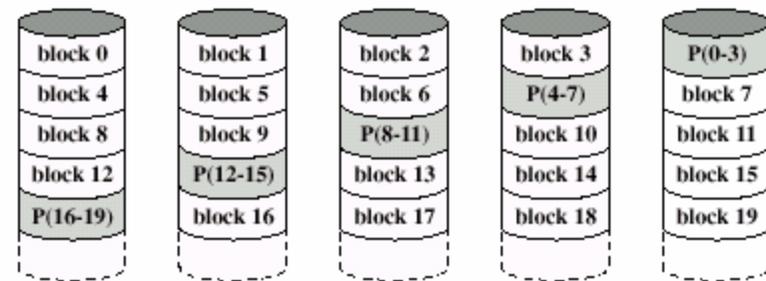
RAID 6: doppelte, unterschiedliche
Parity Berechnung



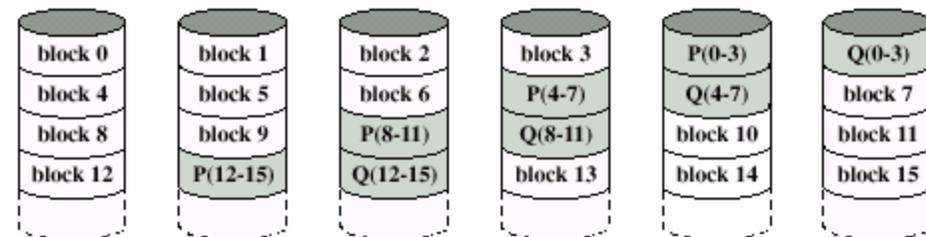
(d) RAID 3 (bit-interleaved parity)



(e) RAID 4 (block-level parity)



(f) RAID 5 (block-level distributed parity)



(g) RAID 6 (dual redundancy)

RAID

Category	Level	Description	I/O Request Rate (Read/Write)	Data Transfer Rate (Read/Write)	Typical Application
Striping	0	Nonredundant	Large strips: Excellent	Small strips: Excellent	Applications requiring high performance for noncritical data
Mirroring	1	Mirrored	Good/Fair	Fair/Fair	System drives; critical files
Parallel access	X	Redundant via Hamming code	Poor	Excellent	
	3	Bit-interleaved parity	Poor	Excellent	Large I/O request size applications, such as imaging, CAD
Independent access	X	Block-interleaved parity	Excellent/Fair	Fair/Poor	
	5	Block-interleaved distributed parity	Excellent/Fair	Fair/Poor	High request rate, read-intensive, data lookup
	6	Block-interleaved dual distributed parity	Excellent/Poor	Fair/Poor	Applications requiring extremely high availability

X: nicht eingesetzt, unterlegen oder zu teuer

zur Erinnerung: Cacheing

Cacheing ist ein bewährtes Konzept der Informatik um Zugriffs-/Berechnungszeiten zu verringern

Grundidee: häufig genutzte Informationen werden schnell zugreifbar gehalten

Grundlage: räumliche & zeitliche Lokalität der Zugriffe

◆ Programmierung:

- Variable speichern berechnete Werte, die noch gebraucht werden

◆ Hardware, CPU:

- Register, Level 1, Level 2 Cache für Instruktionen und Daten
- TLB für häufige Anfragen: logische -> physikalische Adressen

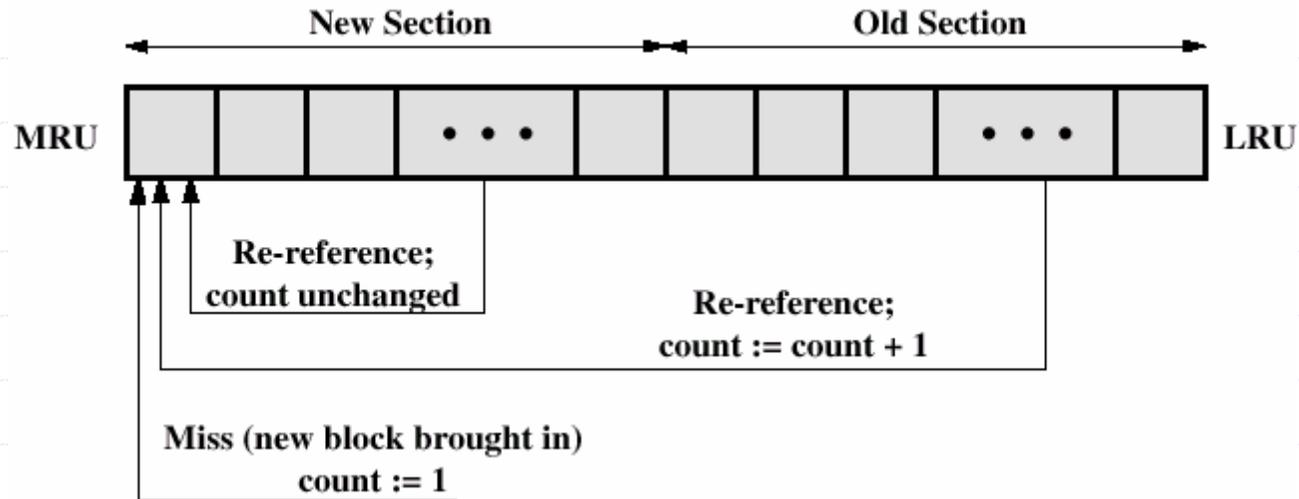
◆ Virtueller Speicher:

- LRU: häufig genutzte Seiten bleiben im Hauptspeicher

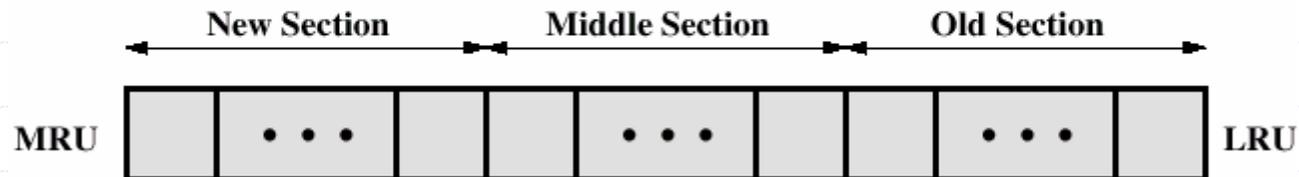
Zugriffszeiten auf Festplatten und Hauptspeicher sehr unterschiedlich -> Einsatzmöglichkeit für Cacheing?

Festplatten Cache, Ersetzung mit LFU

LFU hat Problem mit kurzen Phasen mit vielen Zugriffen (Bursts), weil Zählerwerte dann hoch werden und später irreführend sind.
Ausweg: Zähler im Bereich „neu“ bei Zugriff nicht erhöhen



(a) FIFO



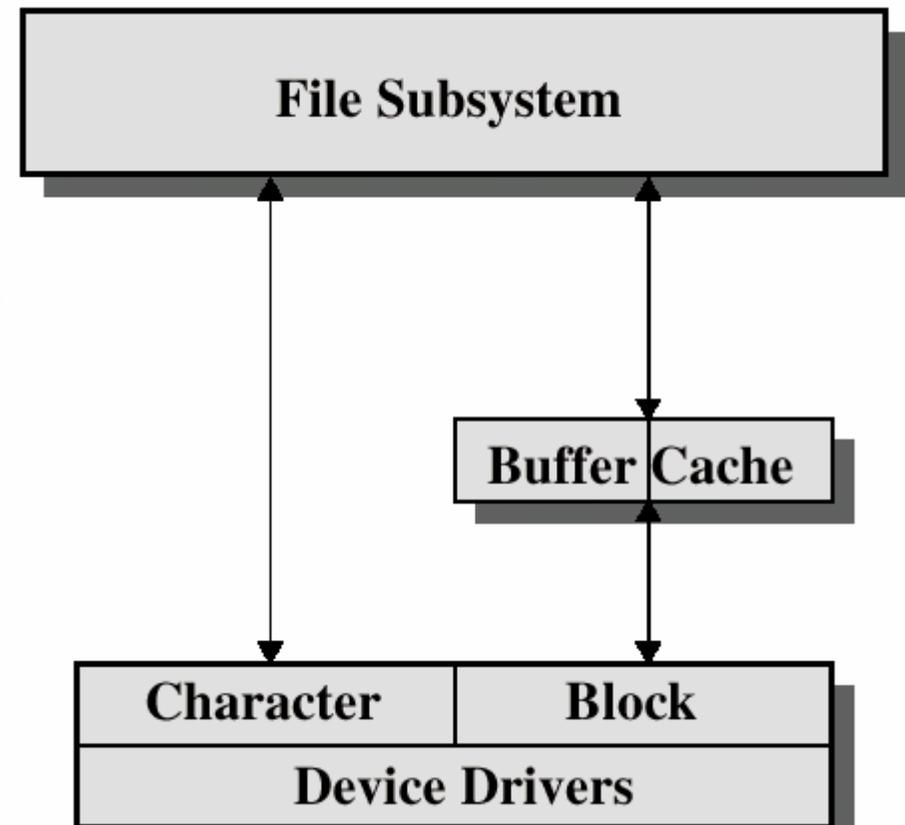
(b) Use of three sections

Beispiel: UNIX SV R4

- ◆ alle E/A Geräte werden als spezielle Dateien verwaltet,
 - einheitliches Interface mit Lese/Schreiboperationen wie bei „normalen“ Dateien mit Anwenderdaten
 - Teil des Dateisystems, das Dateien auf Speichermedien verwaltet, aber durch diese Konstruktion Schnittstelle für E/A Geräte herstellt

◆ 2 Arten von E/A

- ungepuffert: DMA übernimmt Transfer zwischen E/A Gerät und dem Speicherbereich von Prozess für E/A, behindert Swapping
- gepuffert: Daten werden in Systempuffern/Caches zwischengespeichert entweder im „Buffer Cache“ oder in „Character Queues“

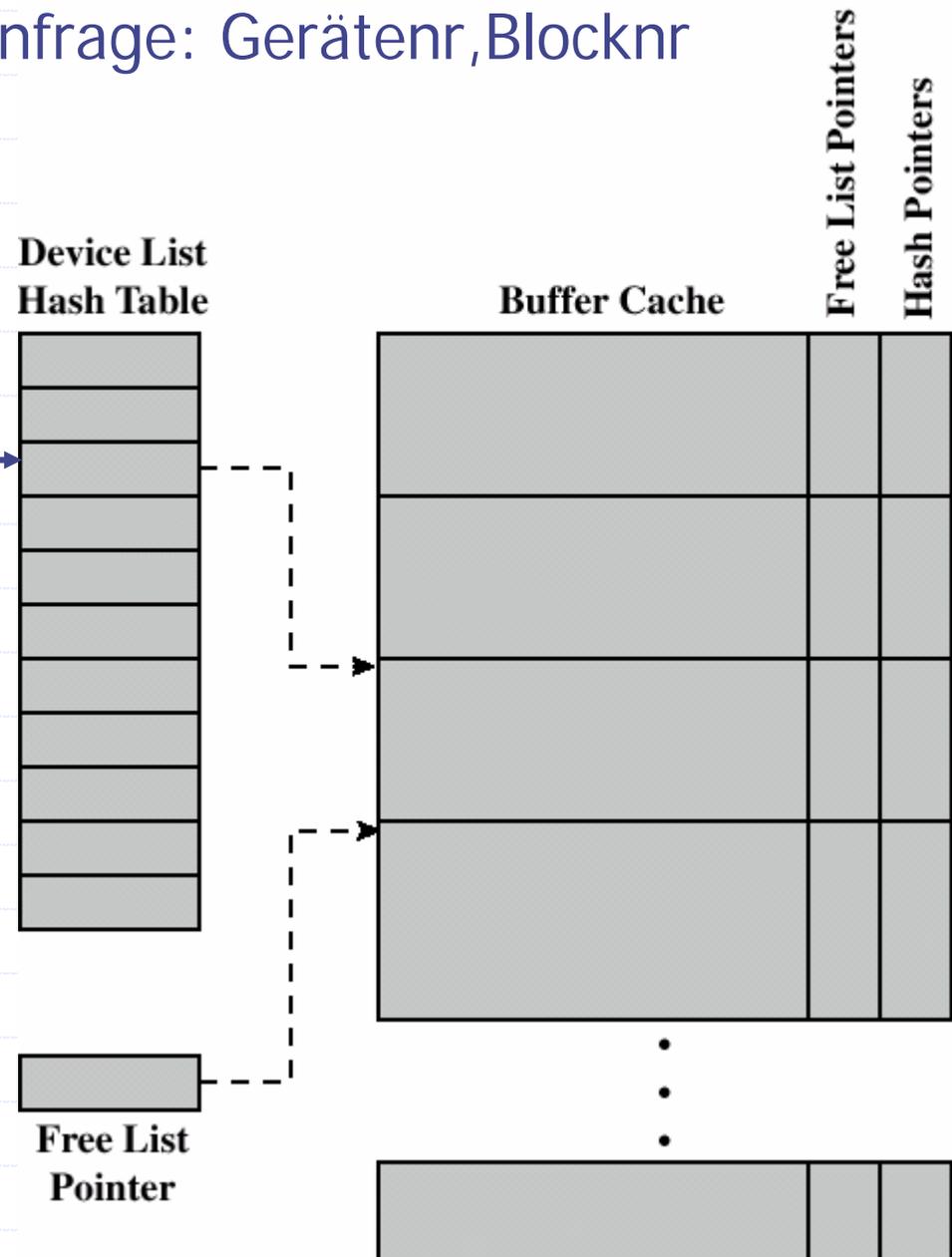


Beispiel: UNIX SV R4, Buffer Cache

Buffer Cache ist
Festplatten Cache

- DMA transferiert Daten zwischen Festplatte - Hauptspeicher und im Hauptspeicher zwischen Prozessbereich und Buffer Cache (erfordert lediglich Buszyklen, keine CPU Belastung)
- Eintrag im Cache: Festplattensektor, z.B. 512 Bytes
- Verwaltung: Liste freier Einträge, Liste aller Einträge je Gerät, Anfragewarteschlange für ein Gerät
- LRU Ersetzungsstrategie

Anfrage: Gerätenr, Blocknr



Beispiel: Unix SV R4

◆ Character Queues

- Produzent/Konsumentverhältnis, typ. bei Terminals, Druckern etc.,
- Inhalte werden durch Lesen „verbraucht“

◆ 5 Kategorien von Geräten

1) Platten 2) Bandgeräte 3) Terminals 4) Drucker 5) Kommunikation

	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers	X		X

Beispiel: Windows 2000, E/A Manager

einheitliche Schnittstelle durch E/A Man.

◆ Cache Manager

- verwaltet Cache im Hauptspeicher für Dateisysteme und Netzwerkkomponente
- Lazy write, Lazy commit

◆ Dateisystemtreiber

- wie Hardwaretreiber behandelt, mit Messages angesteuert

◆ Netzwerktreiber

- integrierte Behandlung von Netzwerkanwendungen und verteilten Applikationen

◆ Hardwaregerätetreiber

- steuern HW Register externer Geräte via DLLs an, dadurch je Gerät nur passende DLL erforderlich, Namensgebung für Dienste/Funktionen ist fest.

I/O Manager

**Cache
Manager**

**File System
Drivers**

**Network
Drivers**

**Hardware
Device Drivers**

Beispiel: Windows 2000, asynchrone und synchrone E/A

- ◆ asynchron: der aufrufende Prozess muß nicht auf das Ende der E/A Anforderung warten -> Performance !!!

Benachrichtigung über Terminierung durch Signale

- Signal eines Geräte-Kernelobjektes: bei Beendigung der E/A wird an dem zugehörigen E/A Objekt, z.B. einer Datei, ein entsprechendes Flag gesetzt, ein wartender Thread wird geweckt.
- Signal eines Ereignis-Kernelobjektes: erlaubt mehrere simultane Anfragen an ein Objekt zu unterscheiden.
- Alarmierbare E/A mittels Asynchroner Prozeduraufrufe (APC): Resultate von E/A Anfragen werden in der APC Warteschlange des aufrufenden Threads abgelegt.
- E/A Terminierungsports: wird in Servern zur effizienten Nutzung eines Thread Pools genutzt.

Beispiel: Windows 2000

RAID Systeme

- ◆ Hardware RAIDs mit mehreren Festplatten
- ◆ Software RAIDs
 - aus nicht-kontinuierlichem Plattenplatz werden eine oder mehrere logische Partitionen erstellt
 - fehlertoleranter Softwareplattentreiber FTDISK
 - erlaubt die Erstellung von RAID 1 (Spiegeln) und RAID 5 (Blockparitybits, Parityblocks verteilt) bei mehreren Platten und mit einem oder mehreren Festplattencontrollern.

Zusammenfassung

- ◆ Struktur und Design von E/A Software im Betriebssystem
- ◆ Techniken zum Puffern von E/A Daten

Wegen der Bedeutung von Festplatten als Speichermedien fokussieren wir auf Festplatten als E/A Geräte

- ◆ Ein/Ausgabe Operationen auf Festplatten
- ◆ Festplatten Scheduling
- ◆ RAID Systeme (Redundant Array of Independent Disks)
 - Schichten 0-6 mit zunehmender Sicherheit
- ◆ Festplatten Cache

- ◆ Beispiele: Unix, Windows 2000