Vorlesung: Rechnerstrukturen, Teil 2 (Modul IP7)

J. Zhang

zhang@informatik.uni-hamburg.de

Universität Hamburg

Fachbereich Informatik

AB Technische Aspekte Multimodaler Systeme

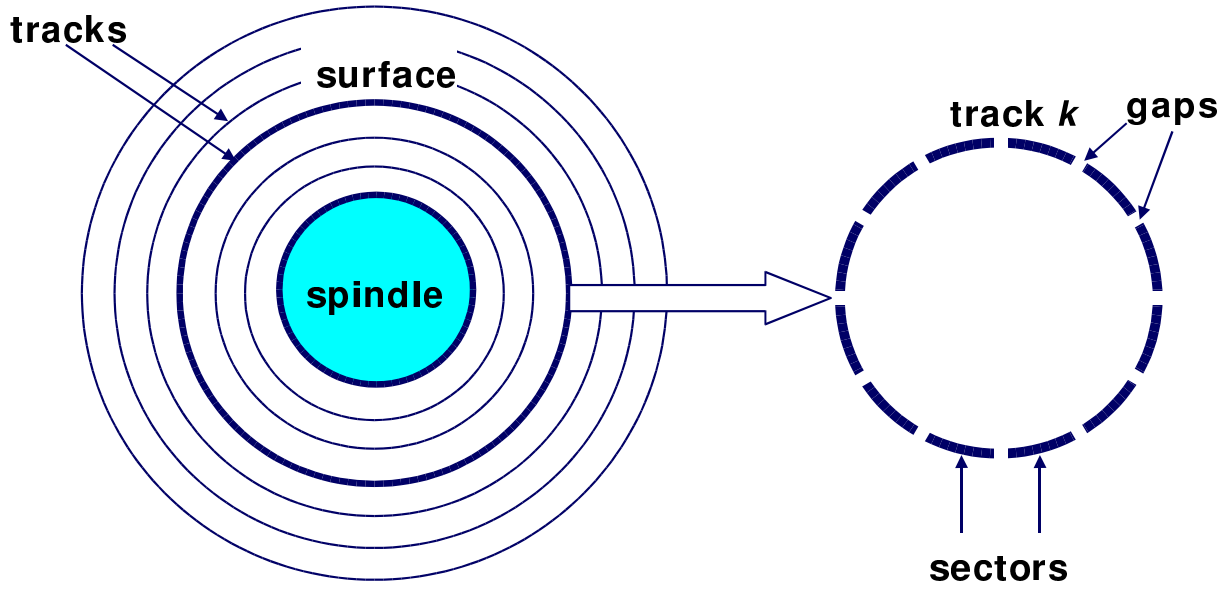
Inhaltsverzeichnis

Festplattengeometrie

Festplatten bestehen aus Platten mit jeweils zwei Oberflächen ("surfaces").

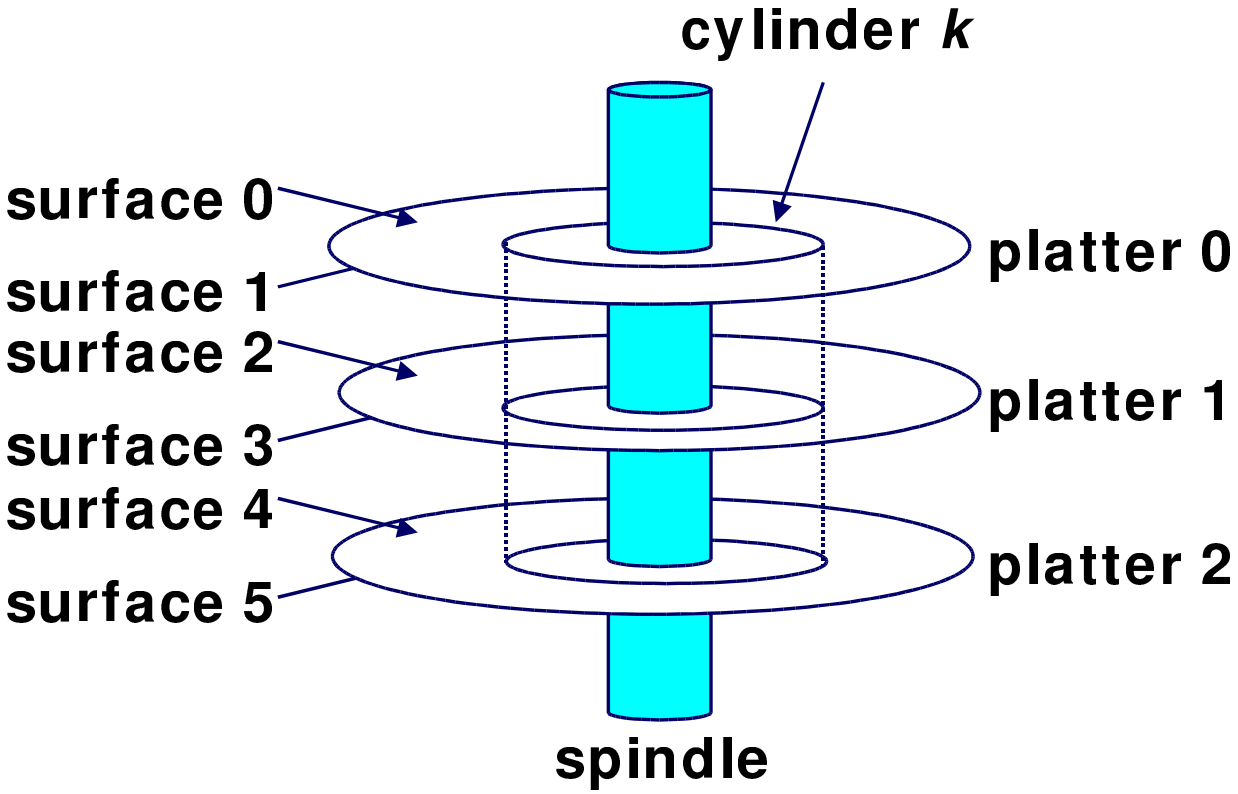
Jede Oberfläche besteht aus konzentrischen Ringen, die Spuren ("tracks") genannt werden.

Jede Spur besteht aus Sektoren ("sectors"), die durch Lücken ("gaps") getrennt sind.



Festplattengeometrie (Ansicht mehrerer Platten)

Untereinander liegende Spuren bilden einen Zylinder.



Festplattenkapazität

Kapazität: Höchstzahl speicherbarer Bits

- Verkäufer geben Kapazität in Gigabyte (GB) an, wobei $1GB = 10^9 Byte$.

Kapazität wird von diesen technologischen Faktoren bestimmt:

- Aufnahmedichte (bits/in): Anzahl der Bits, die in ein 1-Inch Segment einer Spur passen.
- Spurdichte (tracks/in): Anzahl der Spuren, die in ein radiales 1-Inch Segment passen.
- Flächendichte (bits/in2): Resultat aus Aufnahme- und Spurdichte.

Moderne Festplatten unterteilen Spuren in getrennte Unterbereiche, die Aufnahmezonen ("Recording Zones")

- Jede Spur einer Zone hat gleichviele Sektoren, bestimmt durch die Ausdehnung der innersten Spur.
- Jede Zone hat unterschiedlich viele Sektoren/Spuren.

Berechnen der Festplattenkapazität

Kapazität = (#Bytes/Sektor) x (Durchschn. # Sektoren/Spur)x (Spuren/Oberfläche) x (# Oberflächen/Platten) x (# Platten/Festplatte)

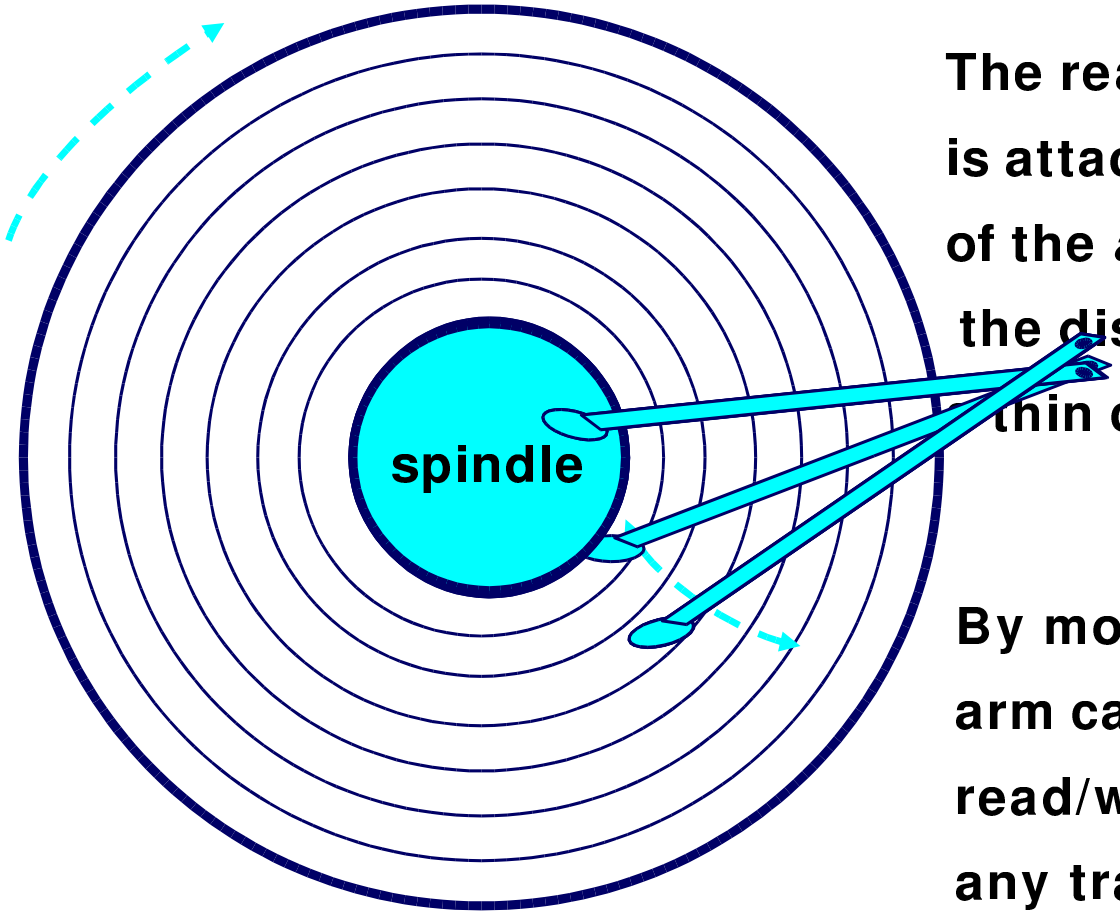
Beispiel:

- 512 Bytes/Sektor
- 300 Sektoren/Spuren (im Durchschnitt)
- 20.000 Spuren/Oberfläche
- 2 Oberflächen/Platten
- 5 Platten/Festplatte

Kapazität = 512 x 300 x 20000 x 2 x 5
= 30.720.000.000
= 30.72 GB

Festplatten-Operation (Ansicht einer Platte)

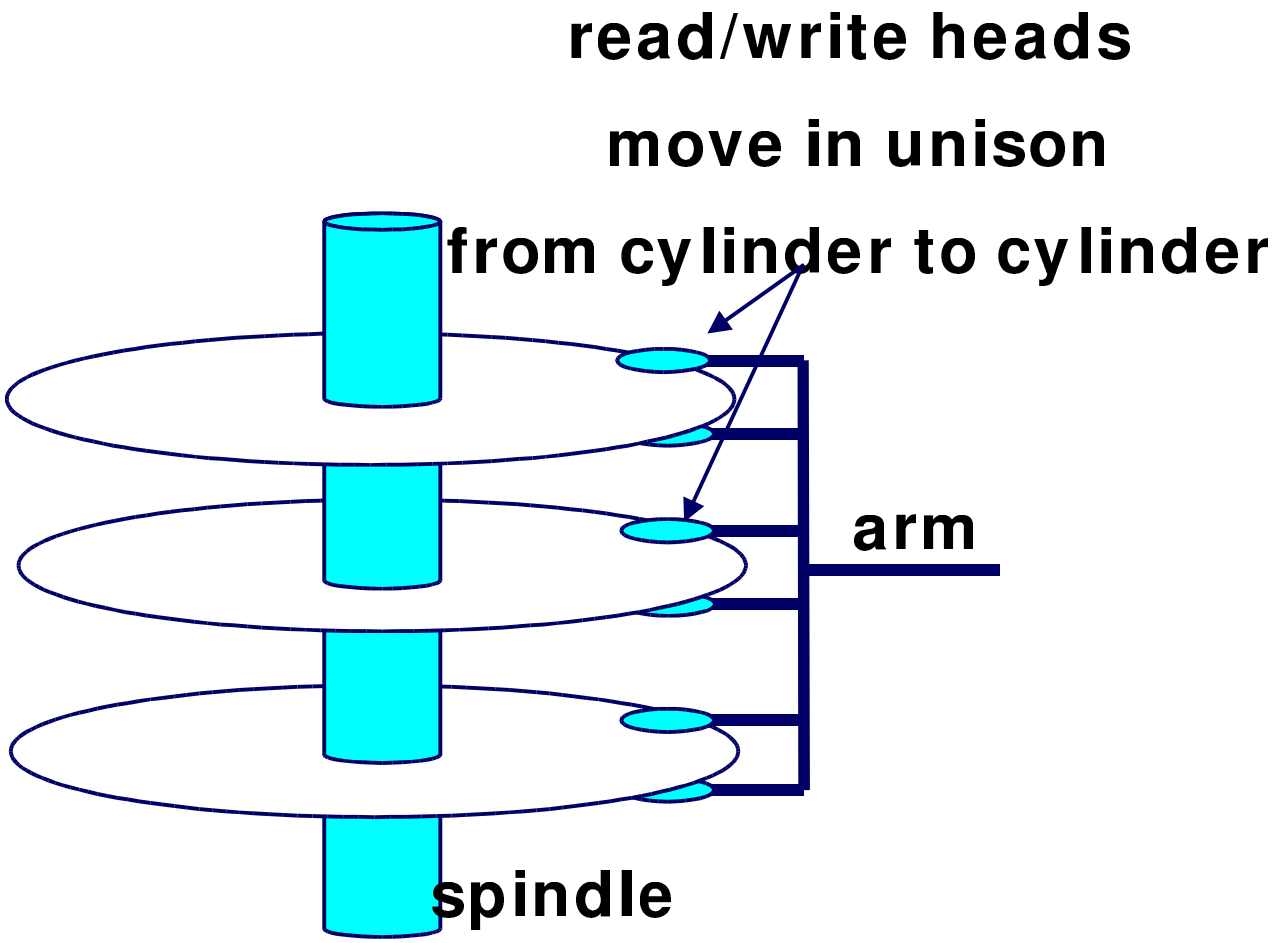
The disk surface spins at a fixed rotational rate



The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

Festplatten-Operation (Ansicht mehrerer Platten)



Festplatten-Zugriffszeit

Durchschnittliche (avg) Zugriffszeit auf einen Zielsektor wird angenähert durch:

- TZugriff = Tavg Suche + TDurchschn Rotationslatenz + Tavg Transfer

Suchzeit (Tavg Suche)

- Zeit in der Schreib-Lese-Köpfe ("heads") über den Zylinder mit dem Targetsektor positioniert werden
- Üblicherweise Tavg Suche = 9 ms

Festplatten-Zugriffszeit II

Rotationslatenzzeit (Tavg Rotationslatenz)

- Wartezeit, bis das erste Bit des Targetsektors unter dem r/w Schreib-Lese-Kopf durchrotiert.
- Tavg Rotationslatenz = 1/2 x 1/RPMs x 60 Sek/1 Min

Transferzeit (Tavg Transfer)

- Zeit, in der die Bits des Targetsektors gelesen werden
- Tavg Transfer = 1/RPM x 1/(Durchschn # Sektoren/Spur) x 60 Sek/ 1 Min

Beispiel für Festplatten-Zugriffszeit

Gegeben:

- Umdrehungszahl = 7.200 RPM ("Rotations per Minute")
- Durchschnittliche Suchzeit = 9 ms.
- Avg # Sektoren/Spur = 400

Abgeleitet:

- Tavg Rotationslatenz = 1/2 x (60 Sek/7200 RPM) x 1000 ms/Sek = 4 ms.
- Tavg Transfer = 60/7200 RPM x 1/400 Sek/Spur x 1000 ms/Sek = 0.02 ms
- Tavg Zugriff = 9 ms + 4 ms + 0.02 ms

Beispiel für Festplatten-Zugriffszeit

Wichtige Punkte:

- Zugriffszeit wird von Suchzeit und Rotationslatenzzeit dominiert
- Das erste Bit eines Sektors ist das "teuerste", der Rest ist quasi umsonst
- SRAM Zugriffszeit ist ca. 4ns/Doppelwort, DRAM ca. 60 ns
  - ✦ Festplatte ist ca. 40.000 mal langsamer als SRAM,
  - ✦ 2.500 mal langsamer als DRAM.

Logische Festplattenblöcke

Moderne Festplatten haben eine simplere abstrakte Ansicht der komplexen Sektorengeometrie:

- Der Bereich verfügbarer Sektoren wird als eine Sequenz logischer Blöcke der Größe b modelliert (0,1,2,...)

Abbildung zwischen logischen Blöcken und tatsächlichen (physikalischen) Sektoren

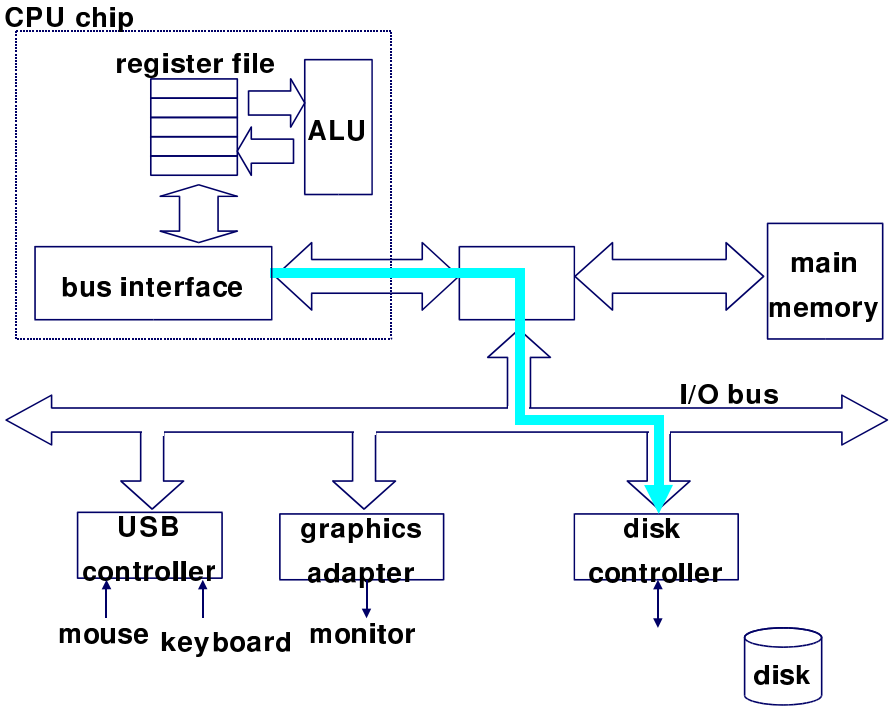
- Wird ausgeführt durch eine Hardware/Firmware Einheit names Festplattencontroller
- Konvertiert Anfragen nach logischen Blöcken zu Tripeln (Oberfläche, Spur, Sektor).

Erlauben dem Controller, für jede Zone Ersatzzylinder bereit zu stellen.

- Bedingt den Unterschied zwischen “formatierter Kapazität” und “Maximaler Kapazität”

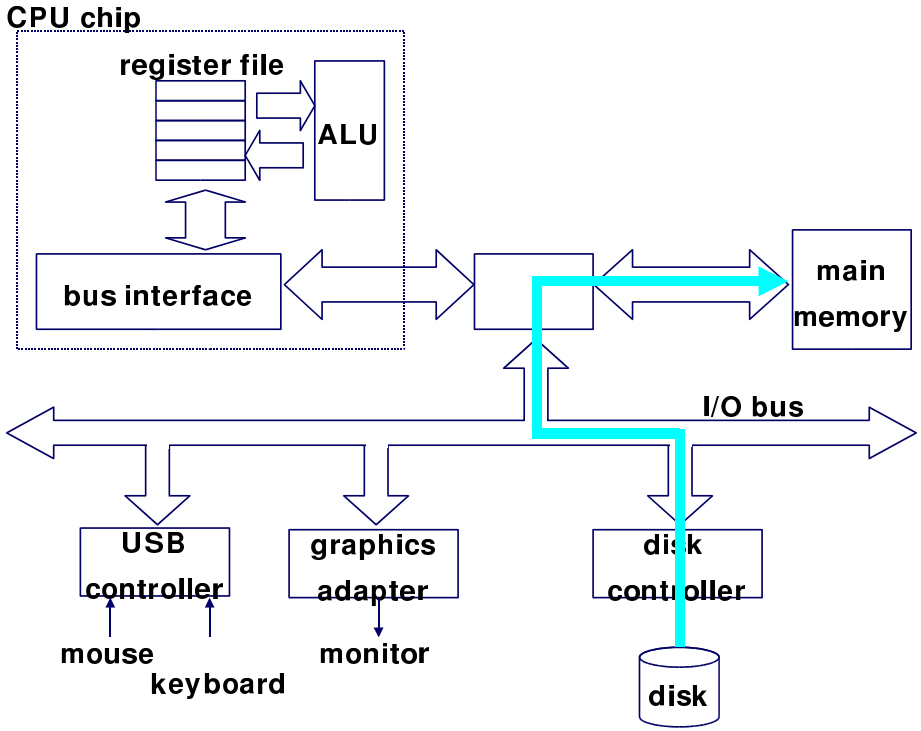
Lesen eines Festplattensektors (1)

Die CPU initiiert einen Lesevorgang von einer Festplatte, indem sie Befehl, logische Blocknummer und Zielspeicheradresse auf einen Port (Adresse) schreibt, der dem Festplattencontroller zugeordnet ist.

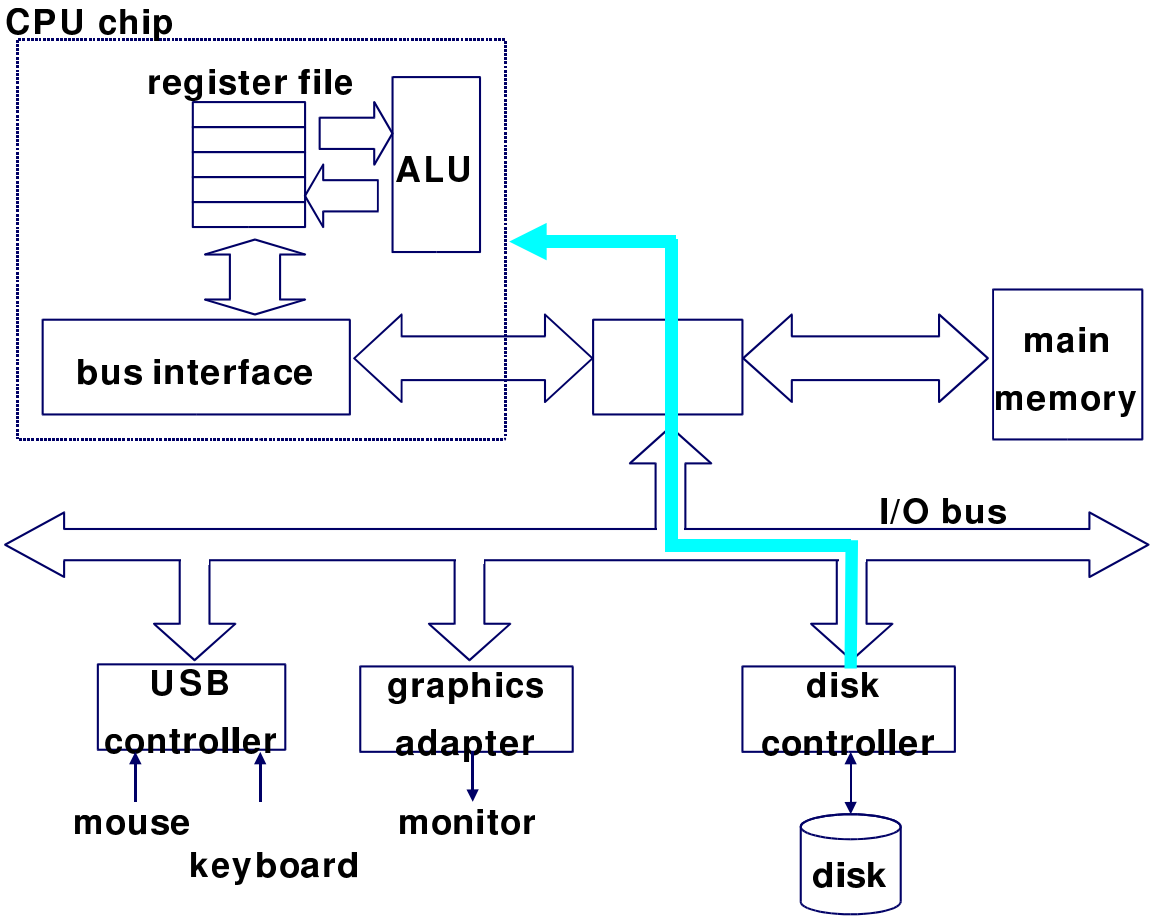


Lesen eines Festplattensektors (2)

Der Festplattencontroller liest den Sektor aus und führt einen direkten Speicherzugriff (DMA) auf den Hauptspeicher aus.



Lesen eines Festplattensektors (3)



Speicherhierarchien

Einige fundamentale und bleibende Eigenschaften von Hard- und Software:

- Schnelle Speichertechnologien kosten mehr pro Byte und haben geringere Kapazität.
- Der Abstand zwischen CPU und Hauptspeichergeschwindigkeit vergrößert sich.
- Gut geschriebene Programme weisen meist eine gute Lokalität auf

Diese fundamentalen Eigenschaften ergänzen sich sehr gut.
Sie legen einen Ansatz zur Organisation von Speichern und Speichersystemen nahe, der als Speicherhierarchie bekannt ist.

Cache Speicher

Themen

- Allgemeine Cache Speicher Organisation
- Direkt abgebildete Caches ("direct mapped")
- Bereichsassoziative Caches ("set associative")
- Einfluss der Caches auf die Leistung

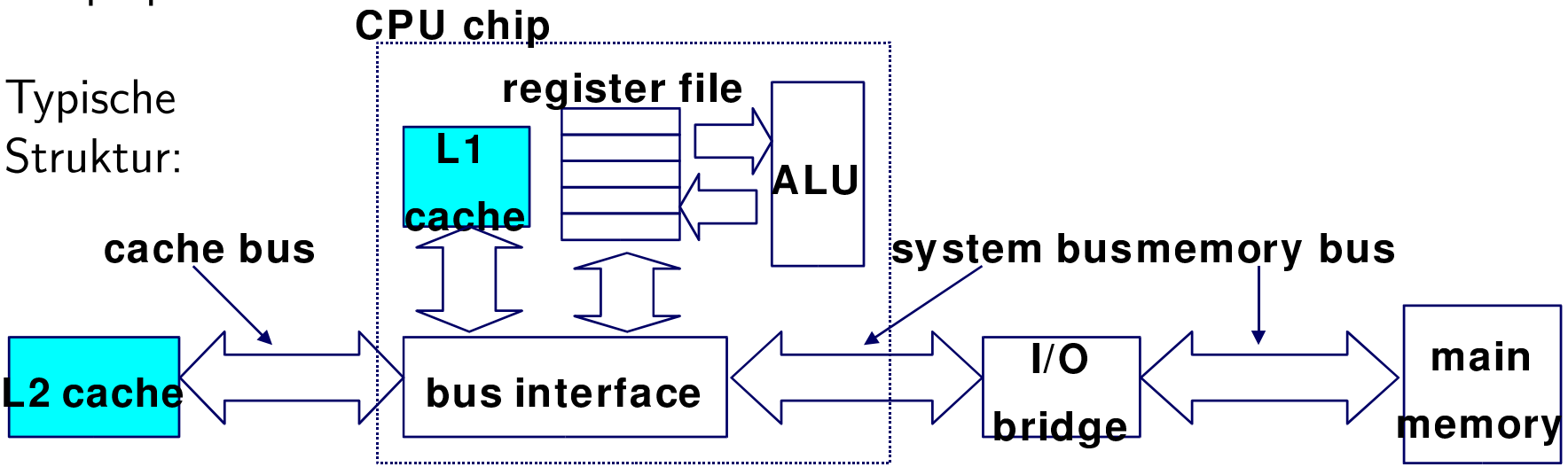
Cache Speicher

Cache Speicher sind kleine, auf schnellen SRAM basierende Speicher, die in der Hardware automatisch verwaltet werden

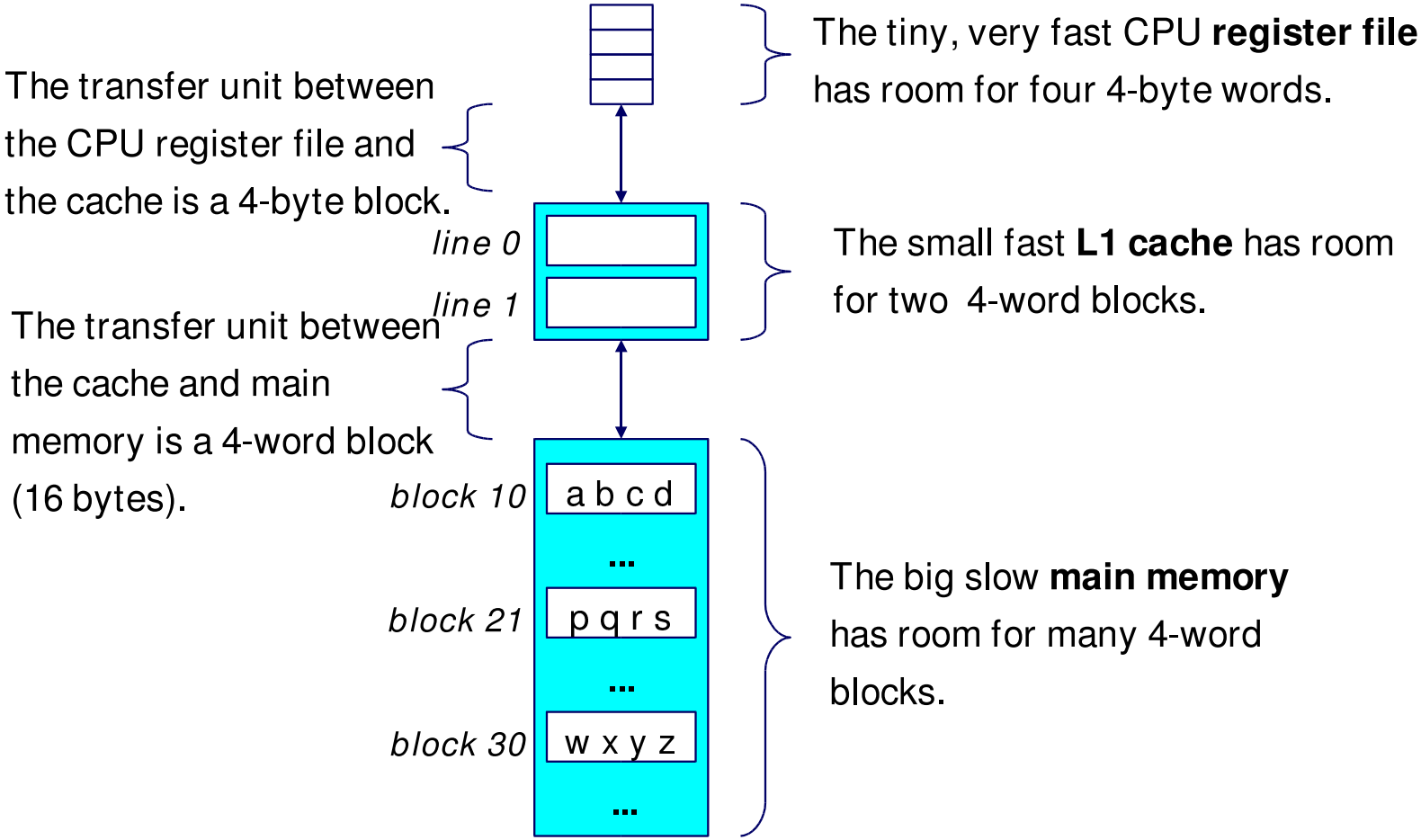
- Enthalten Blöcke des Hauptspeichers, auf die oft zugegriffen wird

CPU sucht erst nach Daten in L1 (level 1), dann in L2 (level 2), dann im Hauptspeicher.

Typische Struktur:

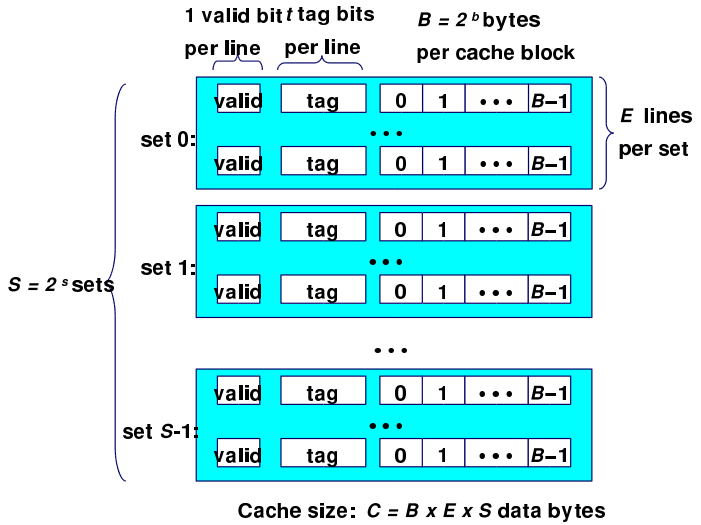


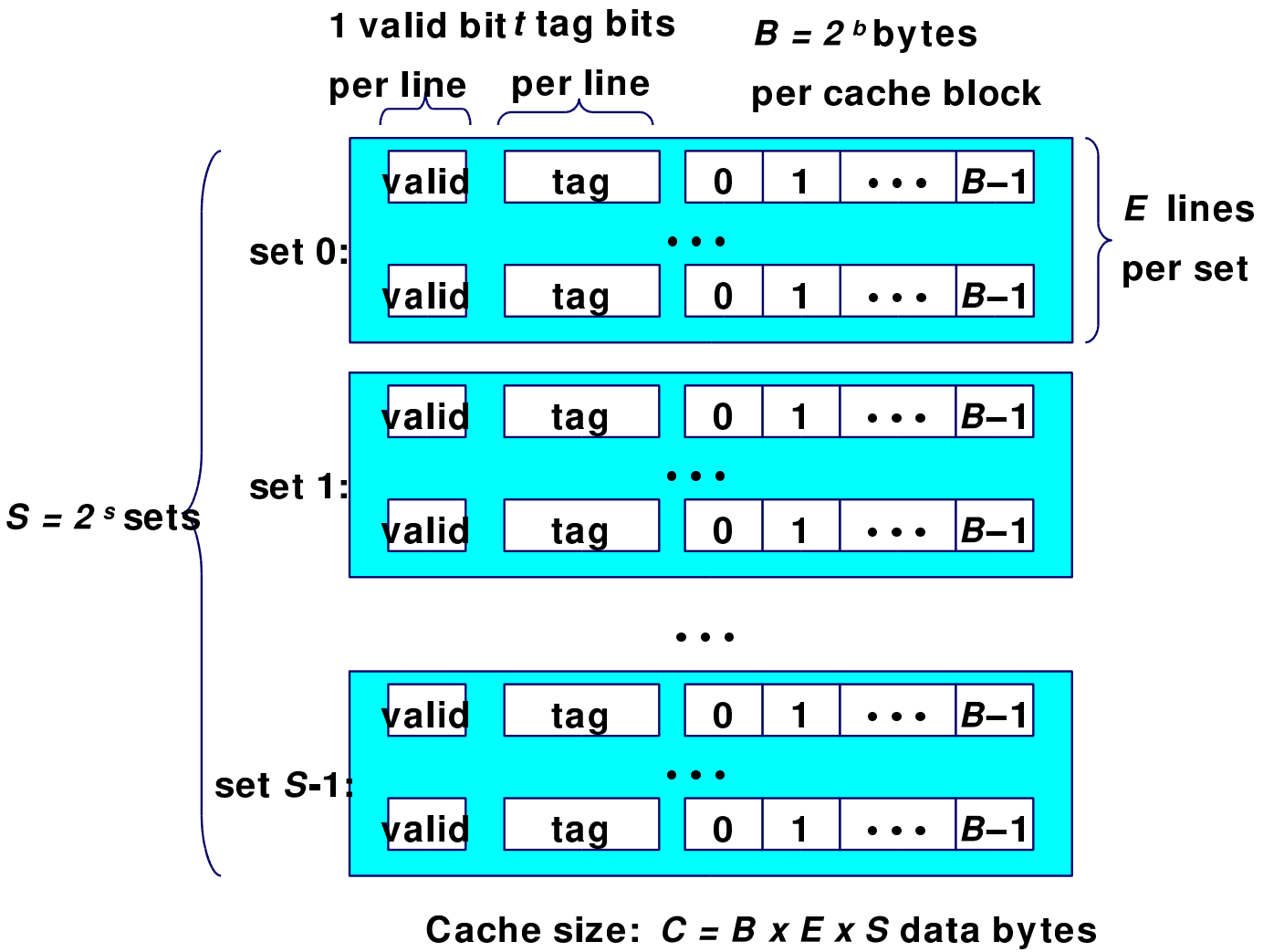
Einschieben eines L1 Cache zwischen CPU und Hauptspeicher



Allgemeine Organisation eines Cache Speichers

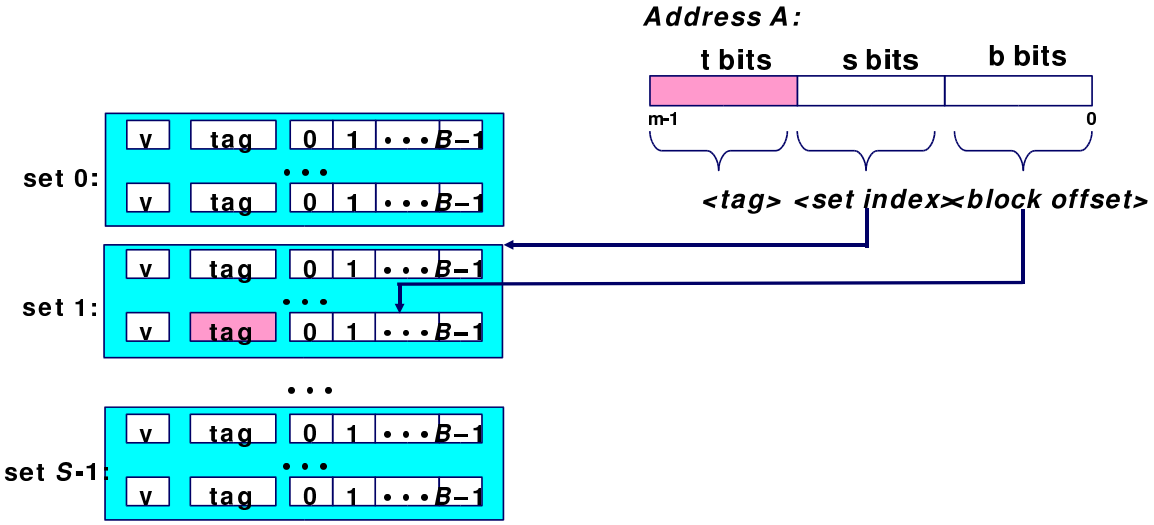
- Cache ist ein Array von Speicher-Bereichen ("sets")
- Jeder Bereich enthält eine oder mehrere Zeilen
- Jede Zeile enthält einen Datenblock




1 valid bit per line
t tag bits per line
B = 2^b bytes per cache block
S = 2^s sets
set 0:
set 1:
set S-1:
valid
tag
0
1
• • •
B–1
E lines per set
Cache size: C = B x E x S data bytes

Adressierung von Caches

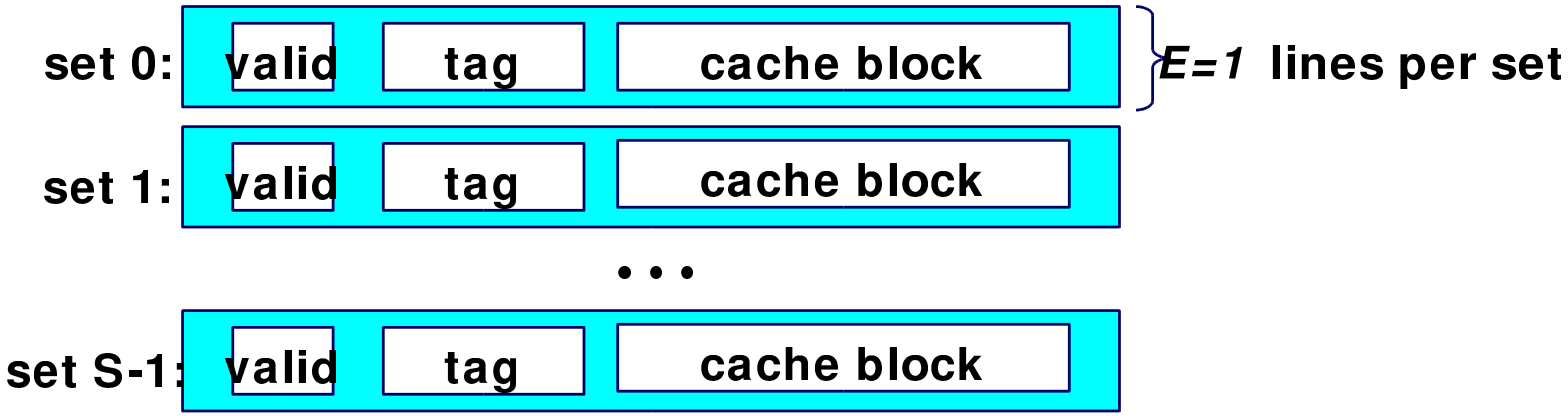
- Das Wort bei Adresse A liegt im Cache, wenn die "tag" Bits in einer der als gültig markierten Zeilen <"valid"> im aktuellen Bereich <"set index"> mit dem <"tag"> der Adresse A übereinstimmen.
- Das gesuchte Wort beginnt im Cache unter dem angegebenen Offset and der Stelle <"block offset bytes"> hinter dem Anfang des Blocks.



Direkt abgebildete Cache (direct mapped)

Einfachste Cache-Art

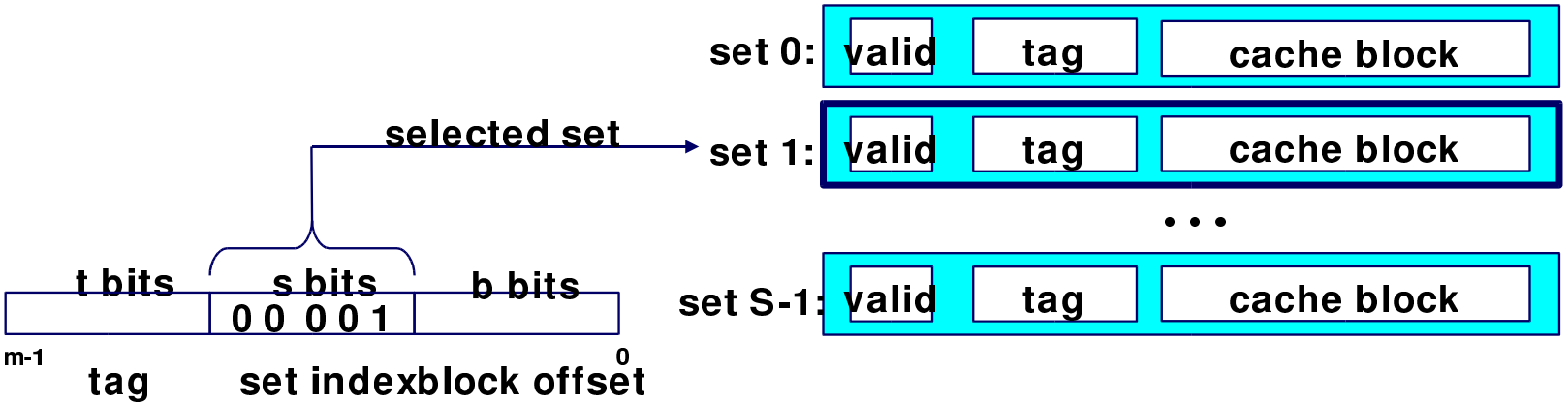
Verfügt über genau 1 Zeile pro Bereich.



Zugriff auf direkt abgebildete Caches

Bereichsauswahl

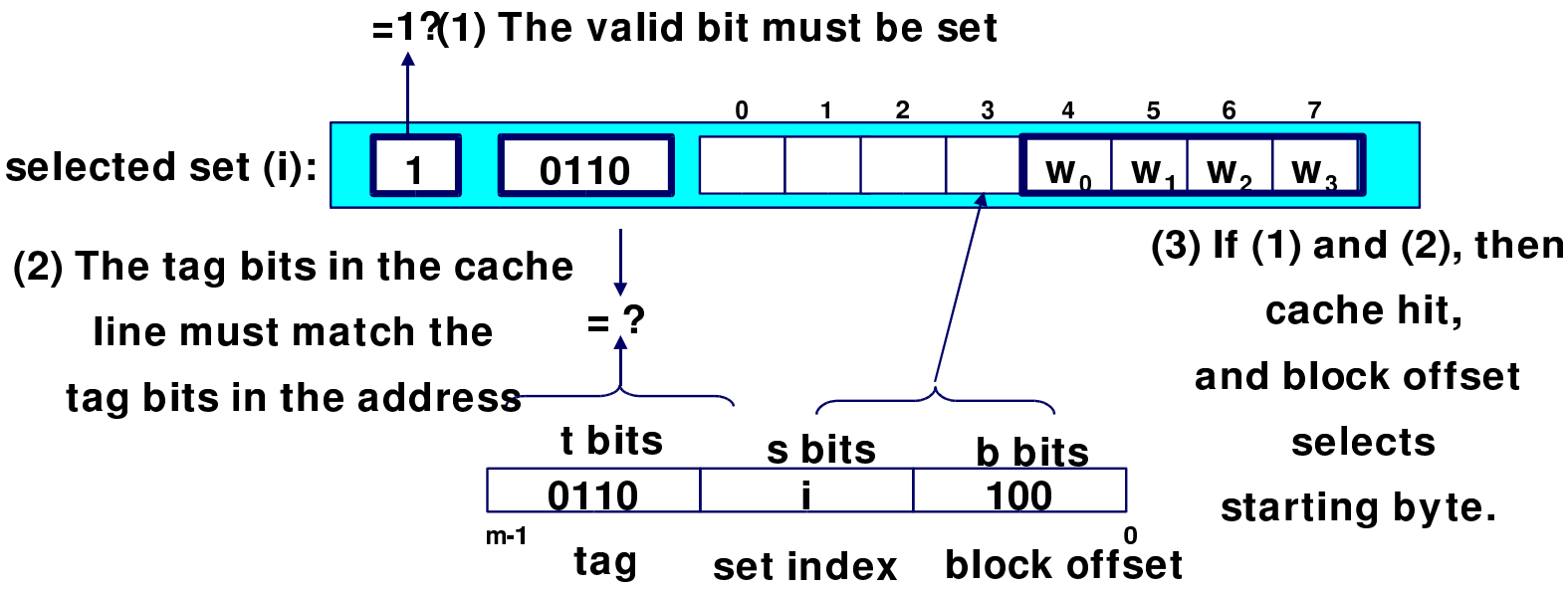
- Benutze die Bereichsindex-Bits ("set index") um den fraglichen Bereich zu bestimmen



Zugriff auf direkt abgebildete Caches

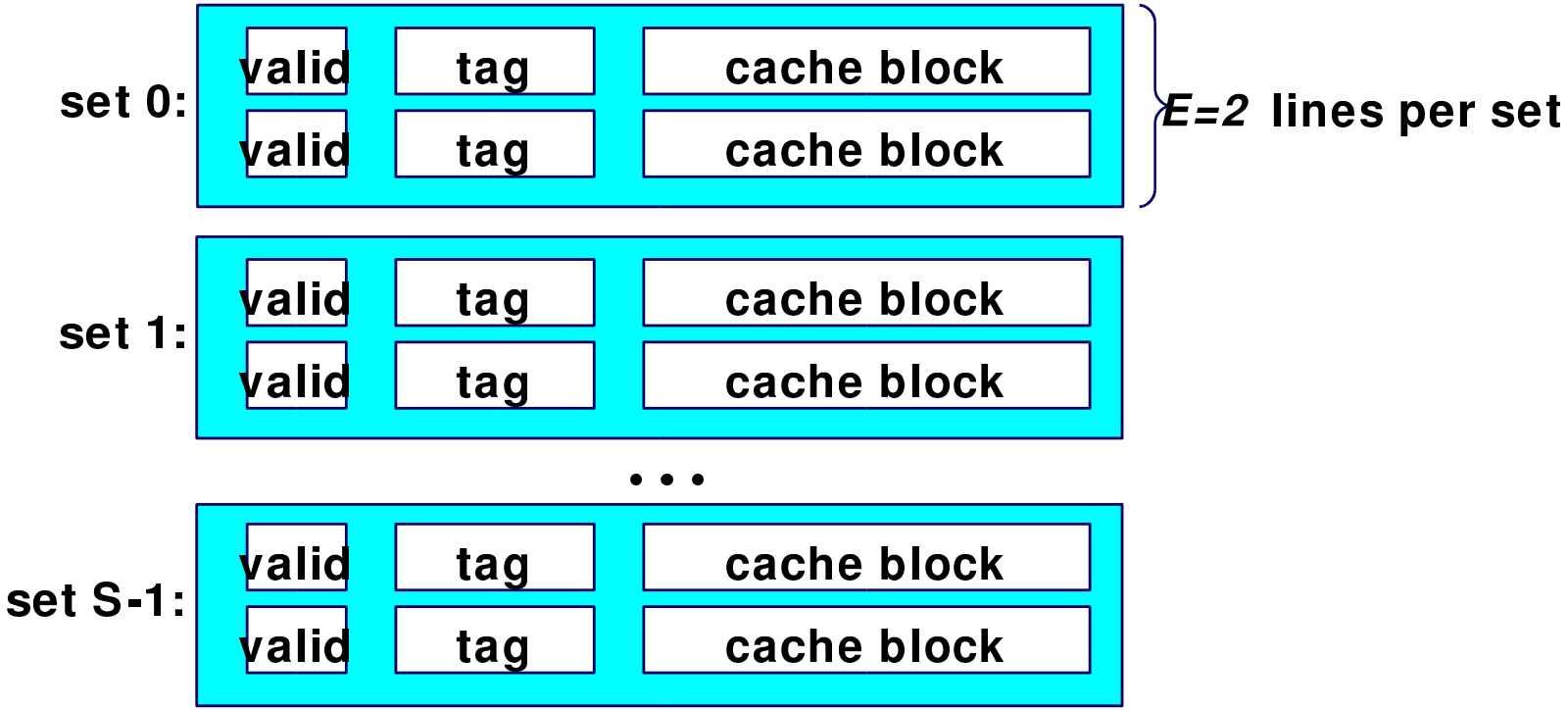
"Line matching" und Wortselektion

- "Line matching": Finde eine gültige Zeile im ausgewählten Bereich mit übereinstimmendem "tag".
- Wortselektion: Dann extrahiere das Wort



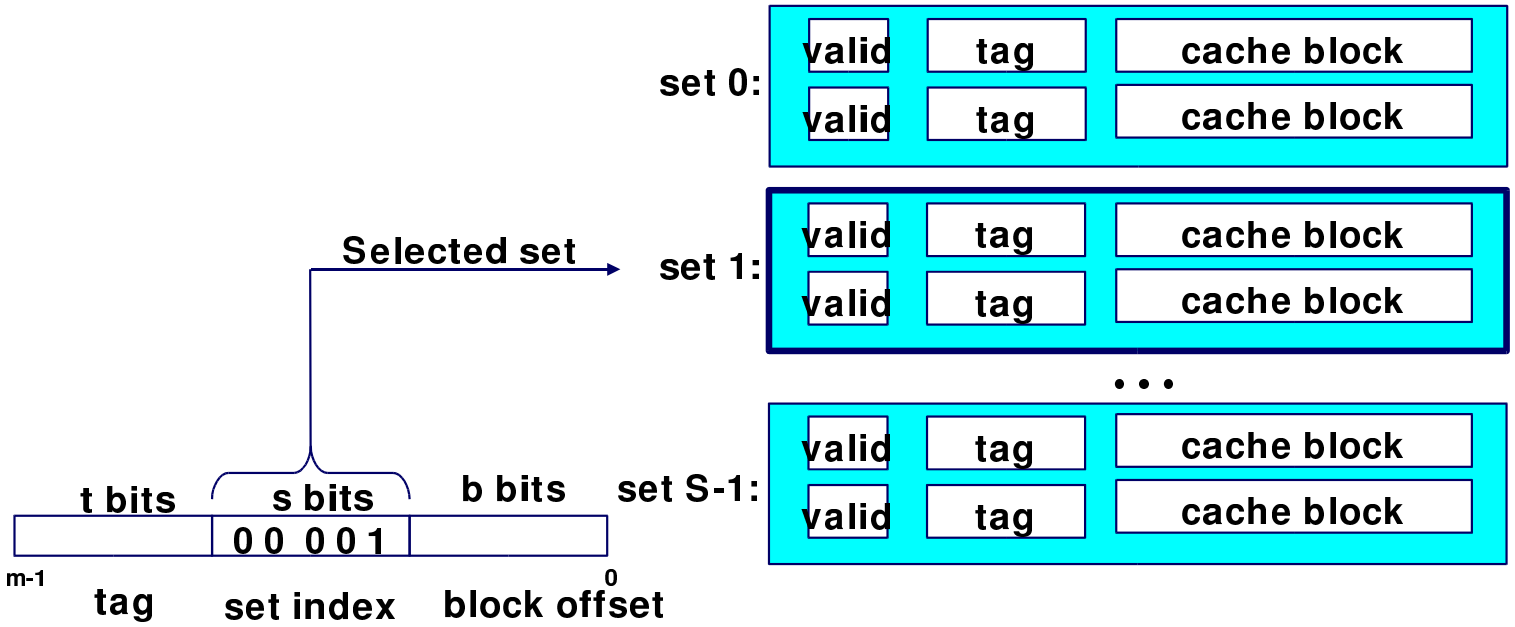
Bereichsassoziative Caches ("set associative")

Verfügen über mehr als eine Zeile pro Bereich.



Zugriff auf Bereichsassoziative Caches

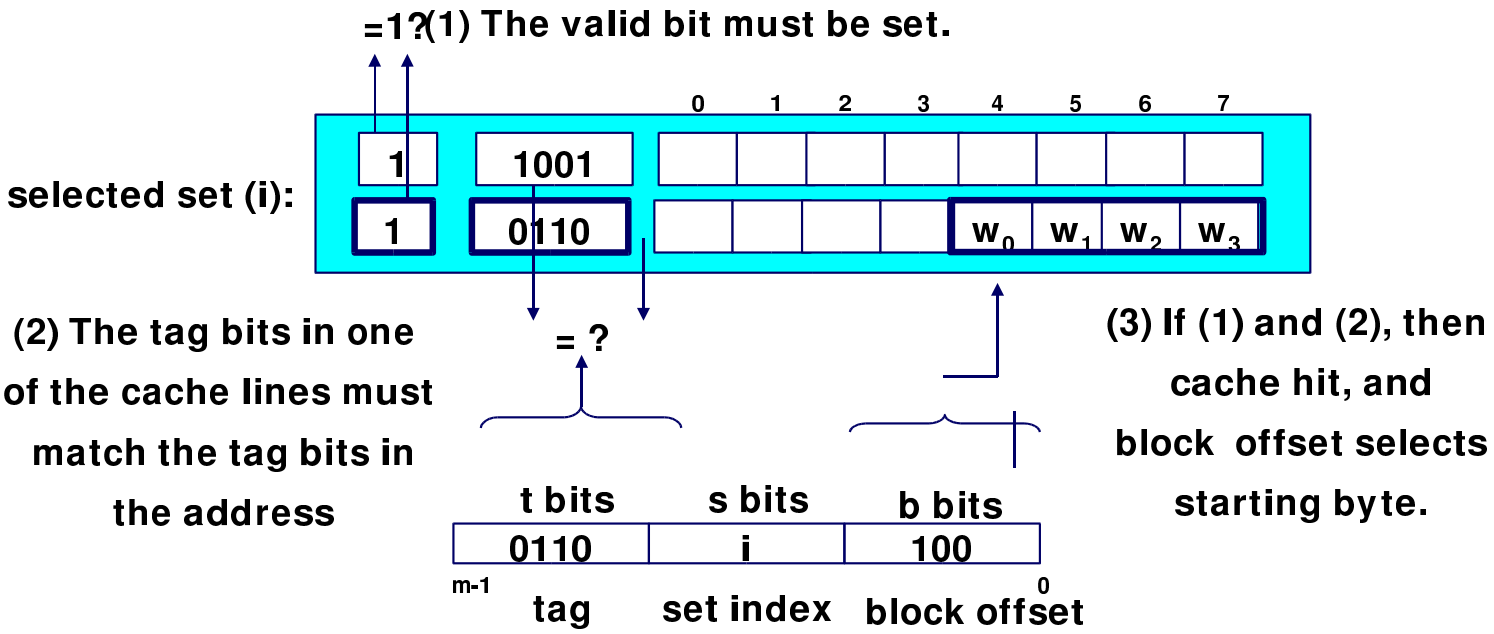
Bereichsauswahl: identisch mit direkt abgebildetem Cache



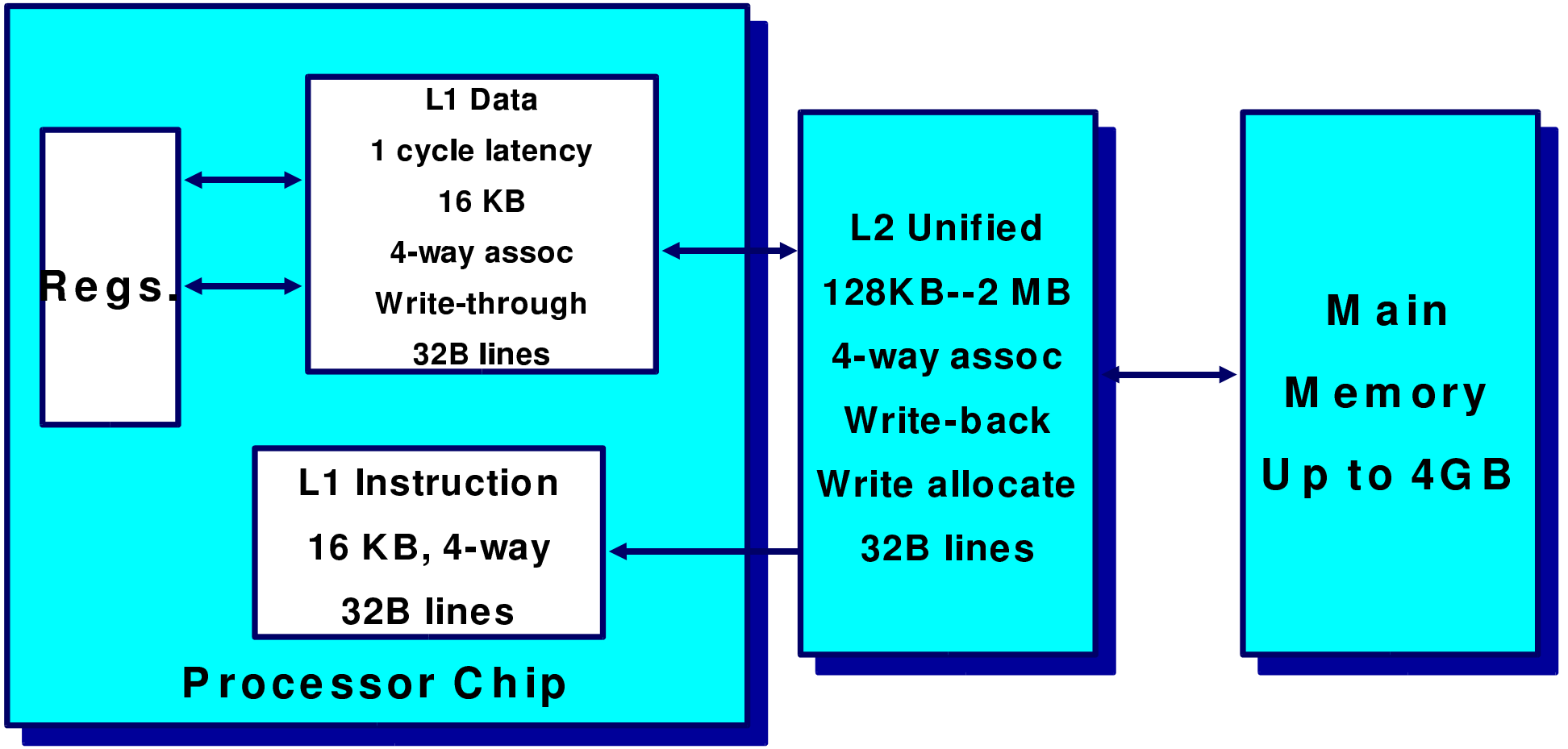
Zugriff auf Bereichsassoziative Caches

"Line matching" und Wortselektion

- muss das "tag" in jeder gültigen Zeile im ausgewählten Bereich ("set") vergleichen



Intel Pentium Cache Hierarchie



Abschließende Bemerkungen

Der Programmierer kann sein Programm für maximale Cacheleistung optimieren

- durch entsprechende Organisation der Datenstrukturen
- durch Steuerung des Zugriffs auf die Daten
  - Geschachtelte Schleifenstruktur
  - Blockbildung ist eine übliche Technik

Abschließende Bemerkungen (Forts.)

Alle Systeme bevorzugen einen "Cache-freundlichen Code"

- Erreichen der optimalen Leistung ist sehr plattformspezifisch
  - ✦ Cachegrößen, Zeilengrößen, Assoziativität etc.
- einen großen Teil der optimalen Leistung erhält man mit allgemeinem Code, wenn man
  - ✦ "working set" angemessen klein hält (zeitliche Lokalität)
  - ✦ kleine Adressfortschaltungen ("strides") benutzt (räumliche Lokalität)

Motivation für einen virtuellen Speicher

Benutzung von Hauptspeicher als "Cache" für die Festplatte

- Addressraum eines Prozesses kann physikalische Speichergröße übersteigen
- Die Summe der Adressräume mehrerer kleiner Prozesse kann physikalischen Speicher übersteigen

Vereinfachung der Speicherverwaltung

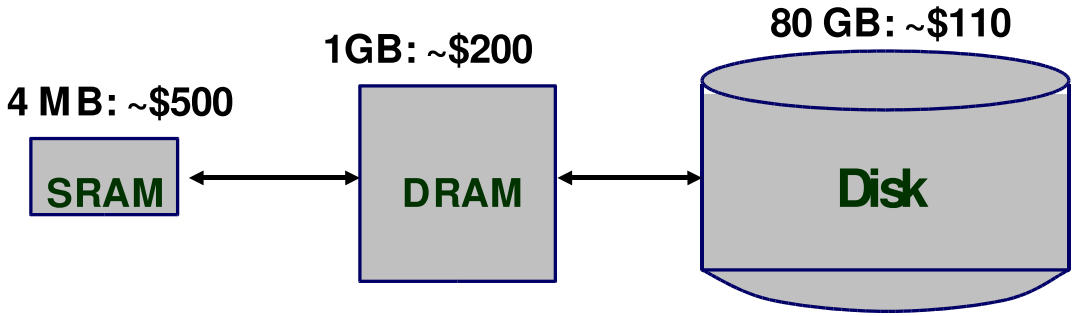
- Viele Prozesse liegen im Hauptspeicher
  - ✦ Jeder Prozess mit seinem eigenen Adressraum
- Nur "aktiver" Code und Daten sind tatsächlich im Speicher
  - ✦ Teile dem Prozess bei Bedarf mehr Speicher zu

Motivation für einen virtuellen Speicher (Forts.)

Bereitstellung von Schutzmechanismen

- Ein Prozess kann einem anderen nicht ins Gehege kommen
  - ✦ da sie in verschiedenen Adressräumen operieren
- Benutzerprozess kann nicht auf privilegierte Informationen zugreifen
  - ✦ die verschiedenen Abschnitte der Adressräume haben verschiedene Zugriffsrechte

Motivation Nr.1: DRAM ist ein "Cache" für die Festplatte



Vollständiger Adressraum ist ziemlich groß:

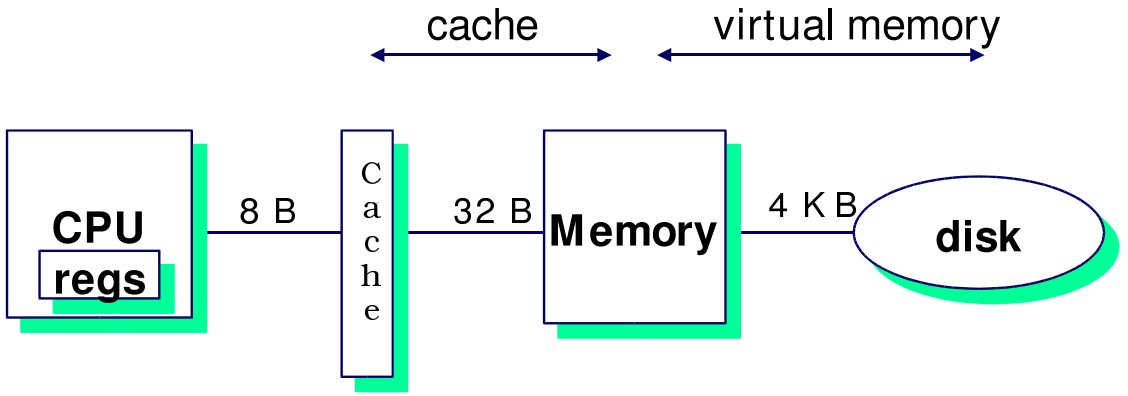
- 32-Bit Adressen: $\approx$ 4.000.000.000 (4 Milliarden) Bytes
- 64-Bit Adressen: $\approx$ 16.000.000.000.000.000.000 (16 Quintillionen) Bytes

Speichern auf der Festplatte ist $\approx$ 300x billiger als im DRAM

- 80 GB DRAM: $\approx$ \$33.000
- 80 GB Festplatte: $\approx$ \$110

Um kostengünstig auf große Datenmengen zuzugreifen, muss der Hauptanteil an Daten auf der Festplatte gespeichert werden.

Ebenen in der Speicherhierarchie



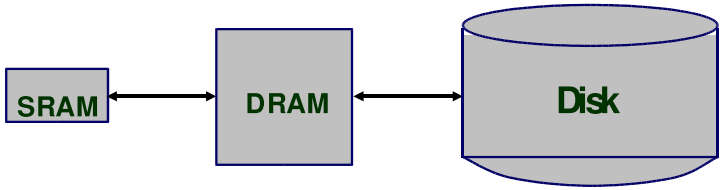
	Register	Cache	Memory	Disk Memory
size:	32 B	32 KB-4MB	1024 MB	100 GB
speed:	1 ns	2 ns	30 ns	8 ms
$/Mbyte:		$125/MB	$0.20/MB	$0.001/MB
line size:	8 B	32 B	4 KB	

larger, slower, cheaper

DRAM versus SRAM als "Cache"

DRAM versus Festplatte ist extremer als SRAM versus DRAM

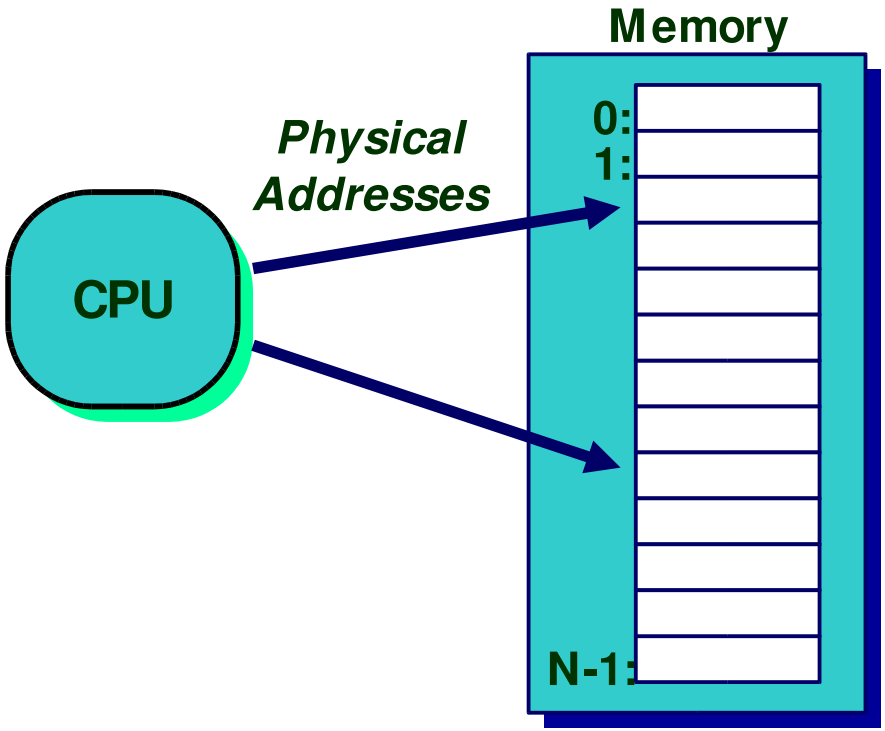
- Zugriffswartezeiten
  - DRAM $\approx$ 10x langsamer als SRAM
  - Festplatte $\approx$ 100.000x langsamer als DRAM
- Wichtigkeit der Nutzung der räumlichen Lokalität:
  - Erstes Byte ist $\approx$ 100.000x langsamer als nachfolgende Bytes
    $\rightarrow$ deutlich dramatischer als z. B. 4malige Verbesserung durch Seitenmodus ("Page-Mode") gegenüber regulärem Zugriff auf RAM
- Schlussfolgerung
  - Designentscheidung für DRAM als Cache hängt von den enormen Kosten bei Fehlzugriff ("Cache miss") ab



Ein System mit ausschließlich physikalischem Speicher

Beispiele:

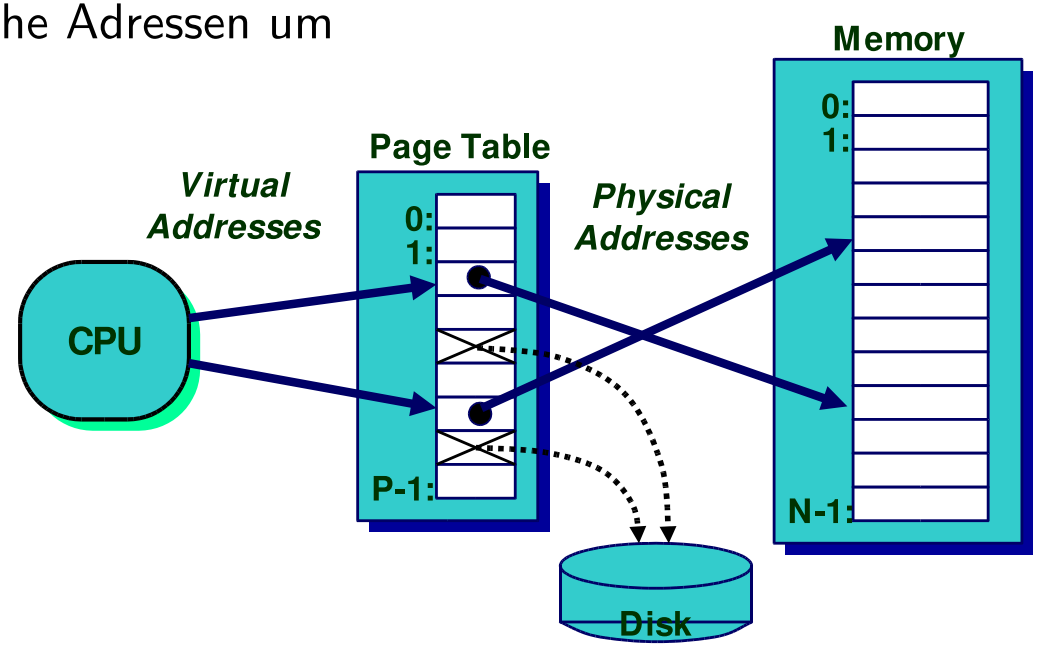
- die meisten Cray Maschinen, frühe PCs, fast alle eingebetteten Systeme etc.



Ein System mit virtuellem Speicher

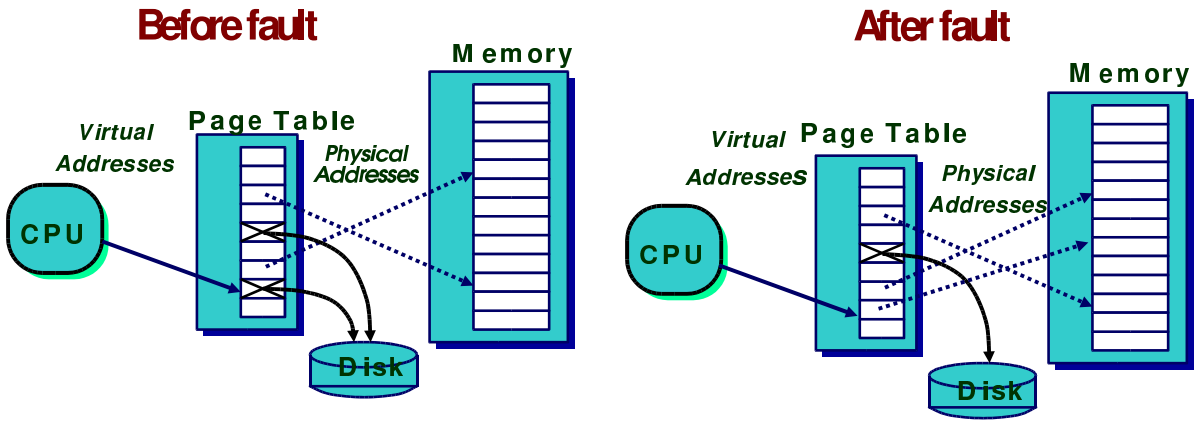
Beispiele:

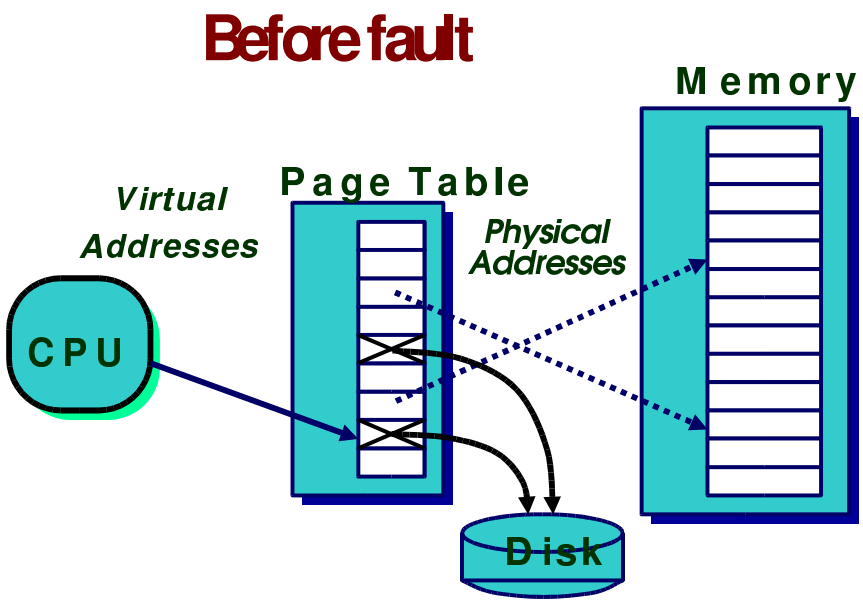
- Workstations, Server, moderne PCs etc.
- Adressübersetzung: Hardware wandelt via vom Betriebssystem (OS) verwalteter "lookup table" (Seitentabelle) virtuelle Adressen in physikalische Adressen um

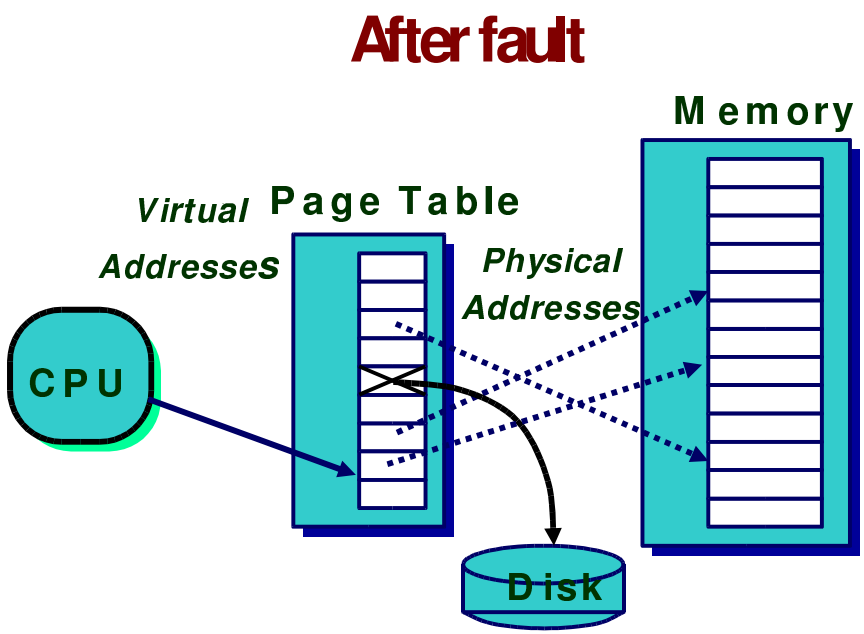


Seitenfehler (ähnlich wie “Cache Misses”)

- Seiten-Tabelleneintrag gibt virtuelle Adresse für eine Seite an, die nicht im Speicher liegt
- OS “exception handler” wird aufgerufen, um Daten von der Festplatte in den Speicher zu laden
  - ✦ laufender Prozess wird unterbrochen, andere können weiterlaufen
  - ✦ OS hat die volle Kontrolle über die Plazierung der neuen Seite im Hauptspeicher (Ersetzungsstrategien) etc.




Before fault
Memory
Page Table
Virtual
Addresses
Physical
Addresses
CPU
Disk


After fault
Memory
Page Table
Virtual
Addresses
Physical
Addresses
CPU
Disk

Einen Seitenfehler behandeln

Prozessor meldet dem Controller

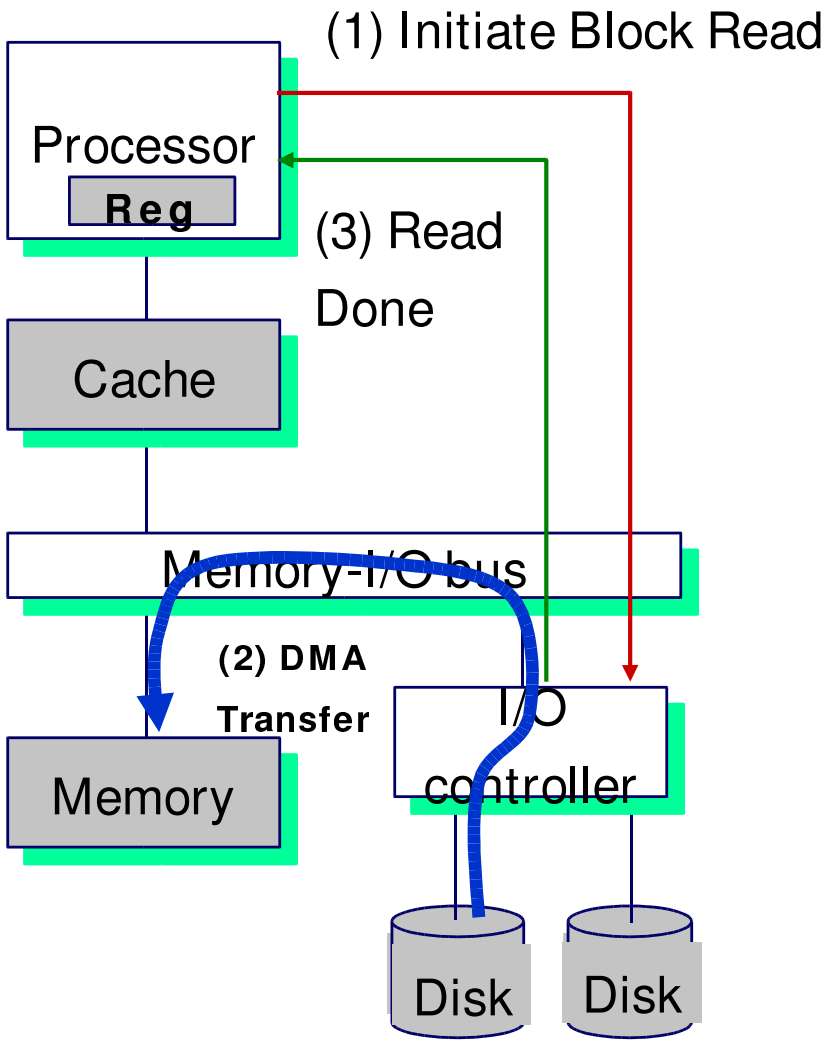
- Lies Block der Länge P ab der Festplattenadresse X und speichere sie ab Speicheradresse Y ab.

Lesezugriff erfolgt in Form von:

- Direct Memory Access (DMA)
- Kontrolle durch I/O Controller

I/O Controller meldet Abschluss

- Gibt Interrupt an den Prozessor
- OS lässt unterbrochenen Prozess weiterlaufen


(1) Initiate Block Read
Processor
Reg
(3) Read
Done
Cache
Memory-I/O bus
(2) DMA
Transfer
I/O
controller
Memory
Disk
Disk

Ergänzende Literatur

Zur Rechnerarchitektur (Teil 2, 7. Termin):

Literatur

[1] Randal E. Bryant and David O'Hallaron. Computer systems. pages 454–531, 690–700. Pearson Education, Inc., New Jersey, 2003.