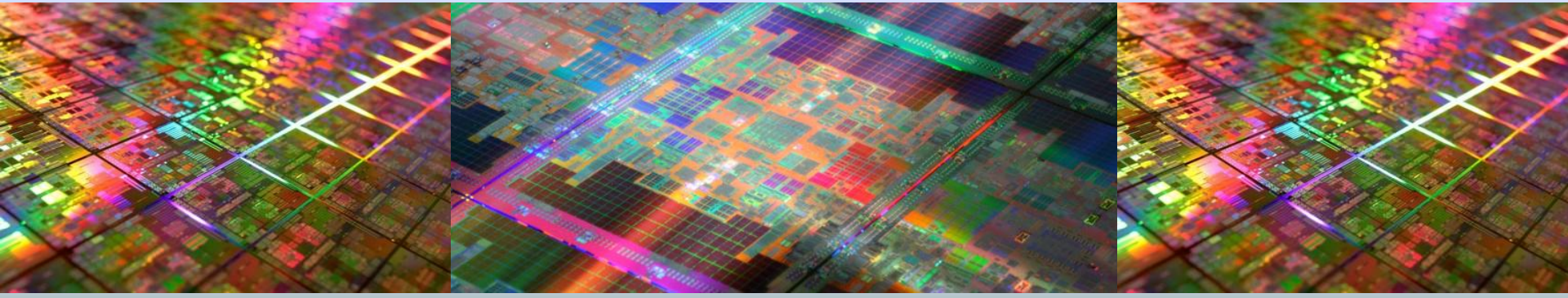
Speicherhierarchie

Technische Grundlagen der Informatik für Wirtschaftsinformatik

Stefan Podlipnig

TU Wien



Lernziele

- Aufbau der Speicherhierarchie und Lokalitätsprinzip kennen lernen
- Aufbau, Organisation und Arbeitsweise von statischem und dynamischem Speicher verstehen
- Vorteile und Eigenschaften einer Cache-Hierarchie kennenlernen
- Aufbau, Arbeitsweise und Unterschiede von vollassoziativem Cache, satzassoziativem Cache und direkt abgebildetem Cache kennenlernen
- Grundprinzip des virtuellen Speichers kennenlernen
- Theoretisches Wissen auf praktische Beispiele anwenden können

SPEICHERHIERARCHIE UND LOKALITÄTSPRINZIP

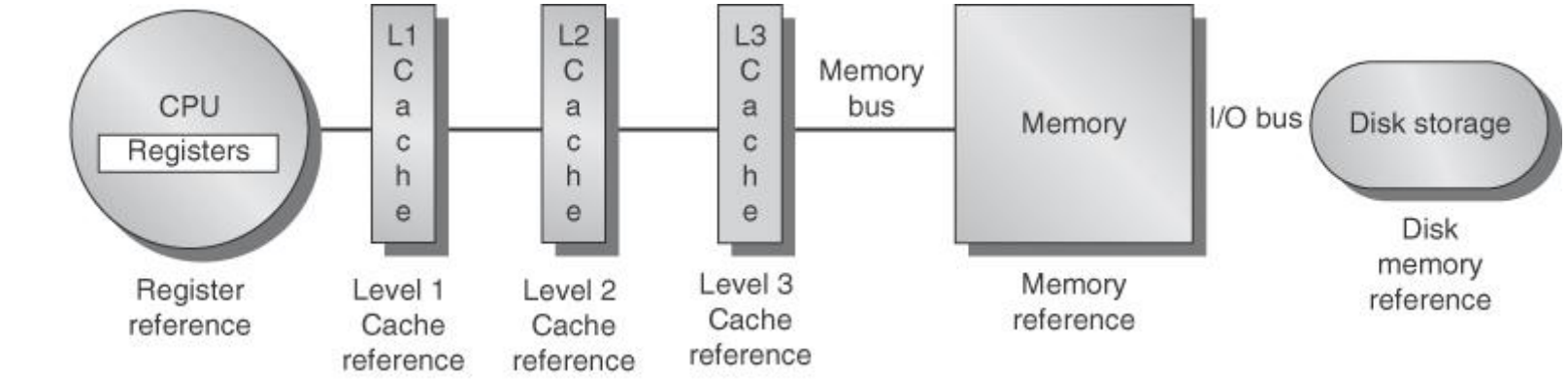
Lokalitätsprinzip

- Programme greifen zu jedem beliebigen Zeitpunkt jeweils nur auf einen relativ kleinen Teil ihres Adressraumes zu
- Temporale Lokalität (*temporal locality*)
  - Wenn ein Zugriff auf eine Adresse erfolgt, ist die Wahrscheinlichkeit hoch, dass bald wieder ein Zugriff darauf erfolgt
  - Z.B. Befehle in einer Schleife
- Räumliche Lokalität (*spatial locality*)
  - Nach einem Zugriff auf eine Adresse ist es wahrscheinlich, dass auch bald ein Zugriff auf in der Nähe befindliche Adressen erfolgt
  - Z.B. Sequentieller Zugriff auf Arrayelemente

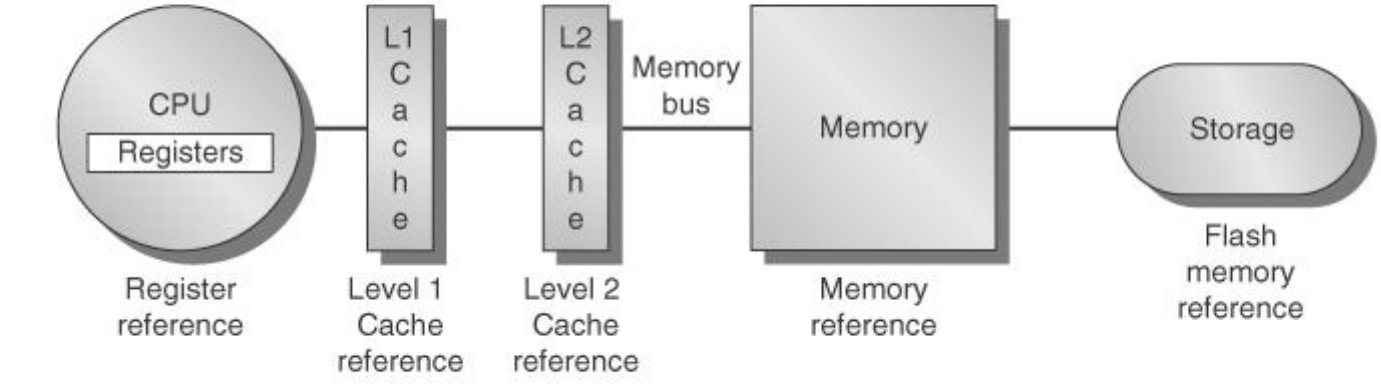
Speicherhierarchie

- Speicherhierarchie (*memory hierarchy*)
  - Eine Struktur, die mehrere Speicherebenen verwendet
  - Je **größer** die Distanz zur CPU wird, desto **größer** werden die Speicher und desto **länger** wird die Zugriffszeit
- In heutigen Rechnersystemen findet man eine mehrstufige Speicherhierarchie (sortiert nach zunehmender Kapazität/ abnehmender Geschwindigkeit)
  - Prozessorregister
  - Level-1-, Level-2-, Level-3-Caches
  - Hauptspeicher (Arbeitsspeicher)
  - Sekundärspeicher (z.B. Festplatten mit permanenter Speicherung)
  - Archivspeicher (Magnetbänder, CD-ROMs, DVDs usw.)

Speicherhierarchie (typische Werte)



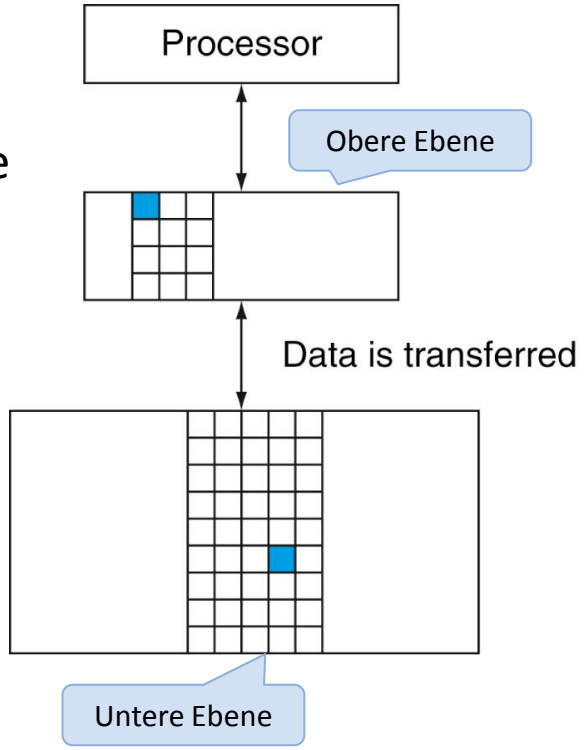
(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Speicherhierarchie - Begriffe

- Block oder Zeile (*line*)
  - Kleinste Informationseinheit für das Kopieren
  - Kann aus mehreren Datenwörtern bestehen
- Treffer (*hit*)
  - Angefragte Daten befinden sich in der oberen Ebene
  - Trefferrate (*hit rate*)
    - Anteil der Speicherzugriffe, die auf der oberen Ebene erfüllt werden (Treffer/Anfragen)
- Fehlzugriff (*miss*)
  - Block mit Daten wird aus der unteren Ebene in die obere Ebene kopiert
    - Danach können die Daten aus der oberen Ebene benutzt werden
  - Fehlzugriffsrate (*miss rate*)
    - Fehlzugriffe/Anfragen = 1 - Trefferrate



SPEICHERTECHNOLOGIEN – EIN KURZER ÜBERBLICK

Speichertechnologien

- Vier primäre Technologien für den Aufbau von Speicherhierarchien
  - SRAM (Static Random Access Memory)
    - Caches
  - DRAM (Dynamic Random Access Memory)
    - Hauptspeicher
  - Flash-Speicher
    - Für größere Speicher (vor allem) im mobilen Bereich
  - Festplatten
    - Für größere Speicher

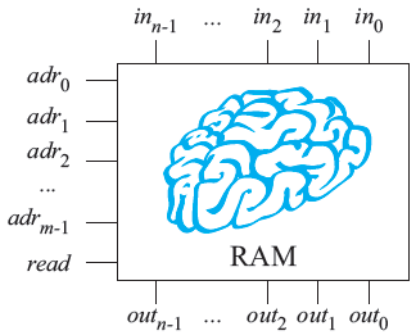
RAM (Random Access Memory)

- Direkter, adressbezogener Zugriff auf einzelne Speicherzellen
  - Um einzelne Befehle zu lesen
  - Um einzelne Daten zu lesen oder zu schreiben
- RAM (Random Access Memory = Speicher mit wahlfreiem Zugriff)
  - Inhalte können sowohl gelesen als auch verändert (beschrieben) werden
  - Das Gegenstück dazu sind ROM (Read Only Memory)-Speicher
    - Können nur gelesen, aber nicht beschrieben werden
- Bei herkömmlichen RAM-Bausteinen ist der Inhalt „flüchtig“
  - Sie behalten nur so lange den gespeicherten Wert, wie sie mit Strom versorgt werden

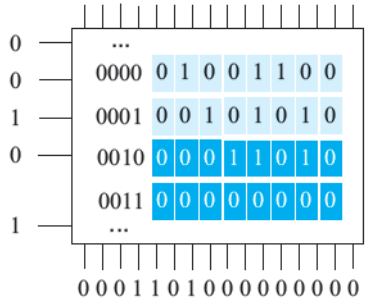
RAM – Allgemeines Schema

- Adressleitungen (adr_i)
  - Für das Ansprechen gespeicherter Datenwörter
  - Typische Adressierungsgranularität von 1 Byte
  - Meist können gleichzeitig mehrere Bytes gelesen bzw. geschrieben werden
    - Unterschied zu blockweise zu beschreibenden Speichern (Flash-Speichern)
- Eingangs- bzw. Ausgangsleitungen (in_i bzw. out_i)
  - read-Signal bestimmt ob gelesen oder geschrieben wird
- Beispiel
  - Adressierungsgranularität von 2 Byte
  - Jeweils 16 Bit einlesen bzw. ausgeben

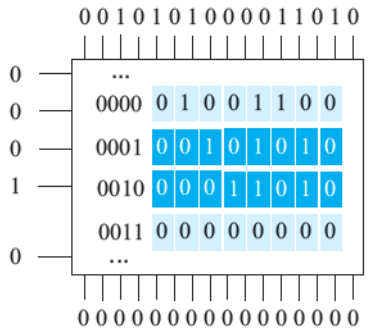
Allgemeines Schema



Lesender Zugriff (Beispiel)

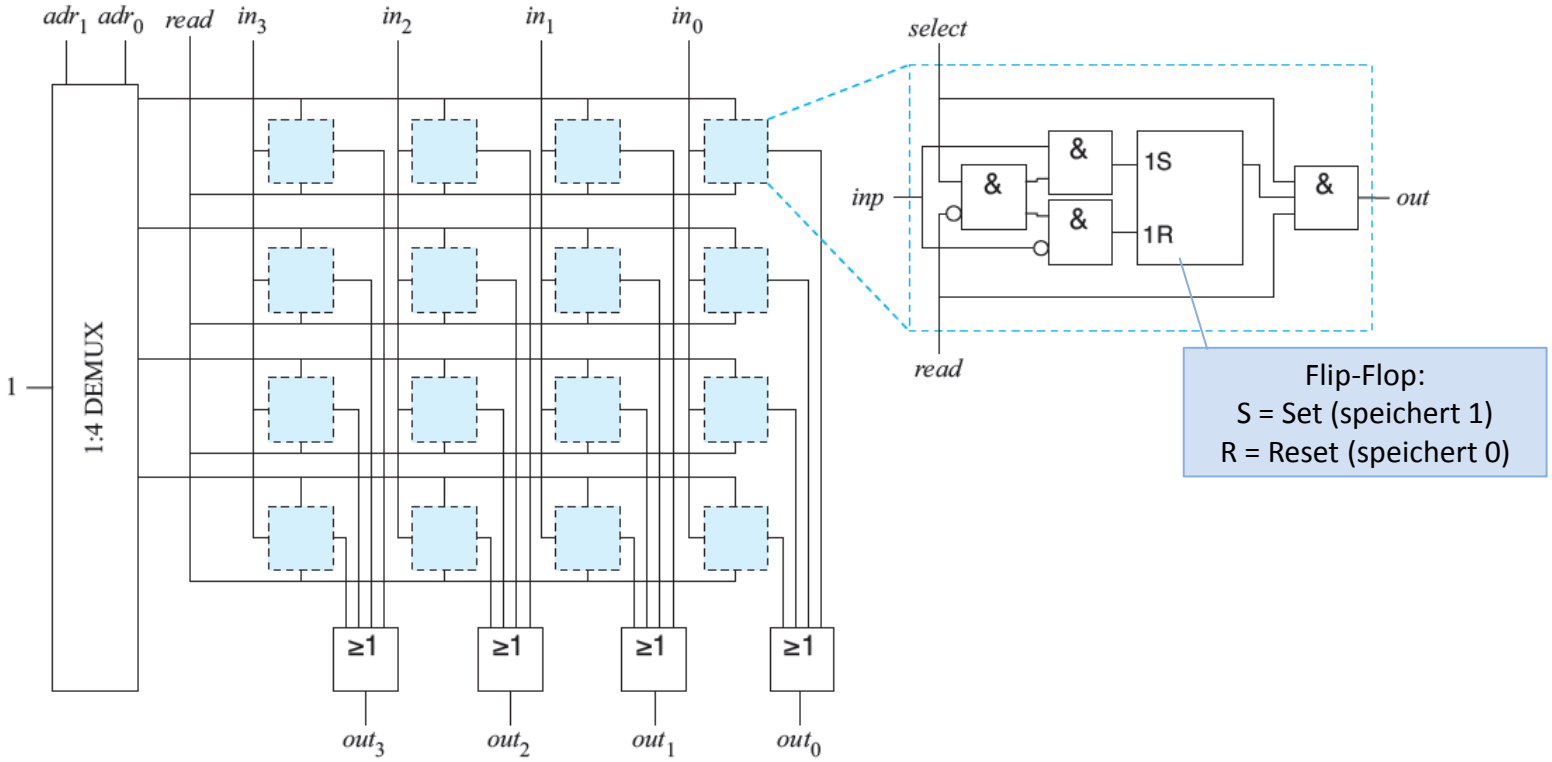


Schreibender Zugriff (Beispiel)



SRAM (1)

- Einzelne Speicherzelle
  - Benötigt nur Anliegen einer Spannung
  - Wird üblicherweise mit 4 bis 10 Transistoren pro Bit realisiert
  - Wird für Prozessor-Register oder schnelle Cache-Speicher verwendet
- Anordnung als Speichermatrix (einfaches Beispiel)

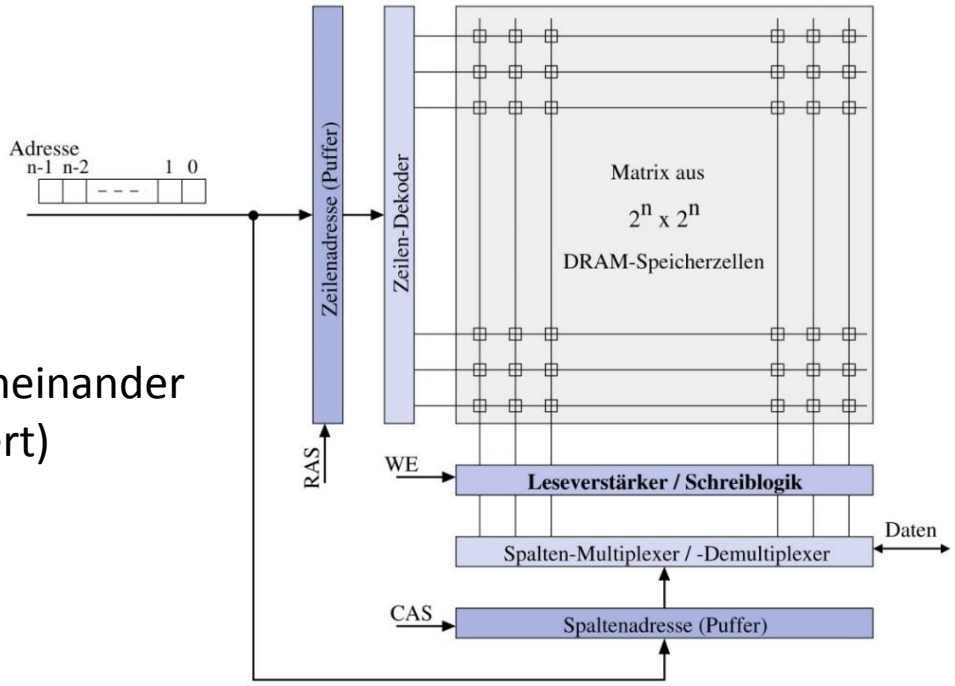


SRAM (2)

- Vorteile
  - Schneller Zugriff
  - Unempfindlich gegen elektromagnetische Strahlung
- Nachteile
  - Geringe Datendichte auf dem Chip
    - Hoher Flächenbedarf
  - Energiebedarf hoch (bei vielen Zugriffen), niedrig (im Standby)
  - Hoher Preis
- Typischer Einsatz
  - Mobile Geräte
  - Netzwerkkomponenten (z.B. Switches, Router)
  - Weltraumgeräte
  - Höchstgeschwindigkeitsrechner
  - L1, L2 und L3 Cachespeicher

DRAM (1)

- Einzelne Speicherzelle
  - Speicherung durch einen Transistor und einen Kondensator je Bit
  - Refresh notwendig
    - Kondensator verliert im Verlauf der Zeit (ca. einige ms) seine Ladung
    - Auch nach dem Auslesen
  - Hohe Datendichte
- Speicheraufbau
  - Speichermatrix
    - Zeilen und Spalten werden nacheinander separat angesprochen (adressiert)
    - Unterschiedliche Signale
      - RAS (Row Adress Strobe)
      - CAS (Column Adress Strobe)
      - WE (Write Enabled)



DRAM (2)

- Vorteile
  - Höhere Datendichte als bei SRAM
  - Geringer Preis
- Nachteile
  - Periodischer Refresh erforderlich
    - Kostet Energie, auch bei Nichtbenutzung des Speichers
  - Hohe Zugriffszeit
- Typischer Einsatz von DRAMs
  - Hauptspeicher in PCs und Workstations
- „Einfache“ DRAMs sind heute nicht mehr erhältlich, sondern nur noch die schnelleren DRAM-Varianten ...

DRAM - Beschleunigung

- Überlappung
  - Auslesen eines Datenwortes erfolgt simultan zum Anlegen der Adresse für den nächsten Zugriff
- Burst-Modus
  - Startadresse bereitstellen
  - Eine festgelegte Anzahl von Daten aus den folgenden Spaltenadressen lesen oder schreiben
- Pipelining
  - Synchrone Arbeitsweise mit dem Systemtakt
  - Je Taktzyklus ein neuer Spaltenzugriff initialisiert bzw. abgeschlossen

DDR-SDRAM

- Double Data Rate Synchronous Dynamic Random Access Memory
  - Synchron mit Systemtakt
  - Datenübertragung bei steigender und fallender Taktflanke
  - Varianten davon sind DDR2, DDR3, DDR4
- Als DIMM(Dual Inline Memory Module)-Module für Arbeitsspeicher
  - Unterschiedliche Signale auf den Anschlusskontakten auf der Vorder- und auf der Rückseite

[http://de.wikipedia.org/wiki/Dual_Inline_Memory_Module , 22.03.2016]

ROM

- ROM-Speicher
  - Inhalt geht nicht verloren, wenn er nicht mit Strom versorgt wird
- Anwendung
  - Bereits beim Einschalten eines Rechners muss z.B. ein Programm abgearbeitet werden können (z.B. Laden des Betriebssystems von einer Festplatte)
  - Ablage konstanter Daten

Arten von ROMs

- ROM
  - Wird bei der Herstellung mit einer festen Funktionalität versehen, die nicht mehr geändert werden kann
  - Bitmuster werden fix in die Hardware „einprogrammiert“
- PROM (Programmable ROM)
  - Kann einmal mit einem speziellen PROM-Brenner programmiert werden
- EPROM (Erasable PROM)
  - Kann mit UV-Licht gelöscht und erneut beschrieben werden
  - EEPROM (Electrically Erasable PROM) kann elektrisch gelöscht werden
- Flash-EEPROM (Flash-Speicher)
  - Kann ohne speziellen EPROM-Brenner mit spezieller Software erneut beschrieben werden

Flash (1)

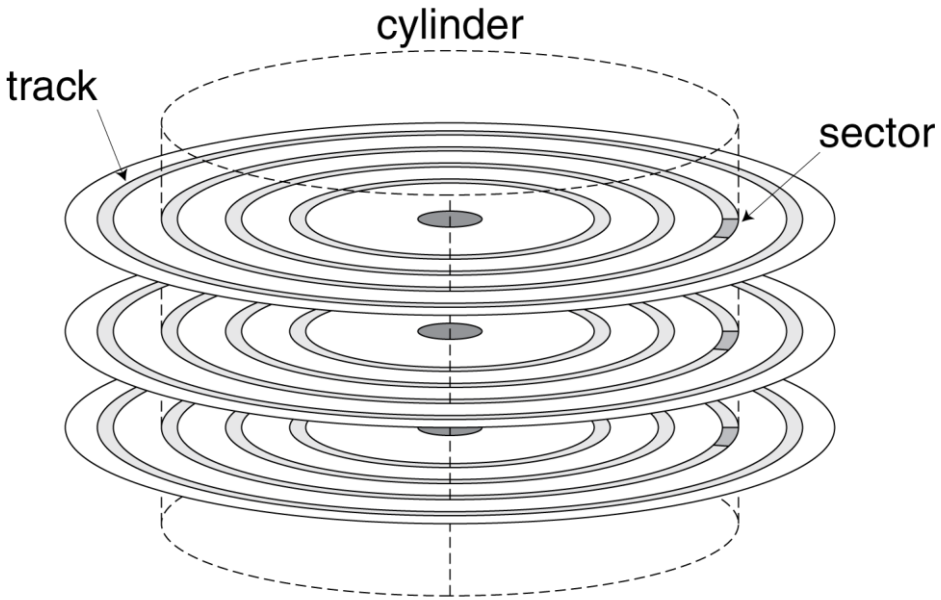
- Flash-Speicher basiert auf speziellen Halbleiterspeichertechniken
- Nichtflüchtig
  - Typische Speicherdauer: ca. 10 Jahre
- Vorteile
  - Klein
  - Geringer Energieverbrauch
  - Robust
  - Keine mechanischen Teile
- Nachteile
  - Teuer

Flash (2)

- NOR-Flash
  - Beliebige Lesezugriffe
  - Blockweise schreiben und löschen
- NAND-Flash
  - Dichter (Bits/Fläche), aber es kann nur blockweise zugegriffen werden
  - Billiger
  - USB-Sticks, Multimediaspeicher, …
- Flash Speicherzellen können nicht beliebig oft benutzt werden
  - Sind daher nicht als direkter Ersatz für RAM gedacht

Festplatte (1)

- Nichtflüchtiger Speicher
- Rotierende Magnetscheiben mit beweglichem Lese-/Schreibkopf



Festplatte (2)

- Jeder Sektor speichert
  - Sektor ID
  - Daten (Z.B. 512 Bytes)
  - Code zur Fehlererkennung
  - Synchronisationsfelder
- Zugriff auf einen Sektor
  - Verzögerung, falls vorherige Anfragen noch bearbeitet werden
  - Suchtzeit (Kopf bewegen)
  - Umdrehungslatenz
    - Bis sich der entsprechende Sektor unter dem Lesekopf befindet
  - Datentransfer

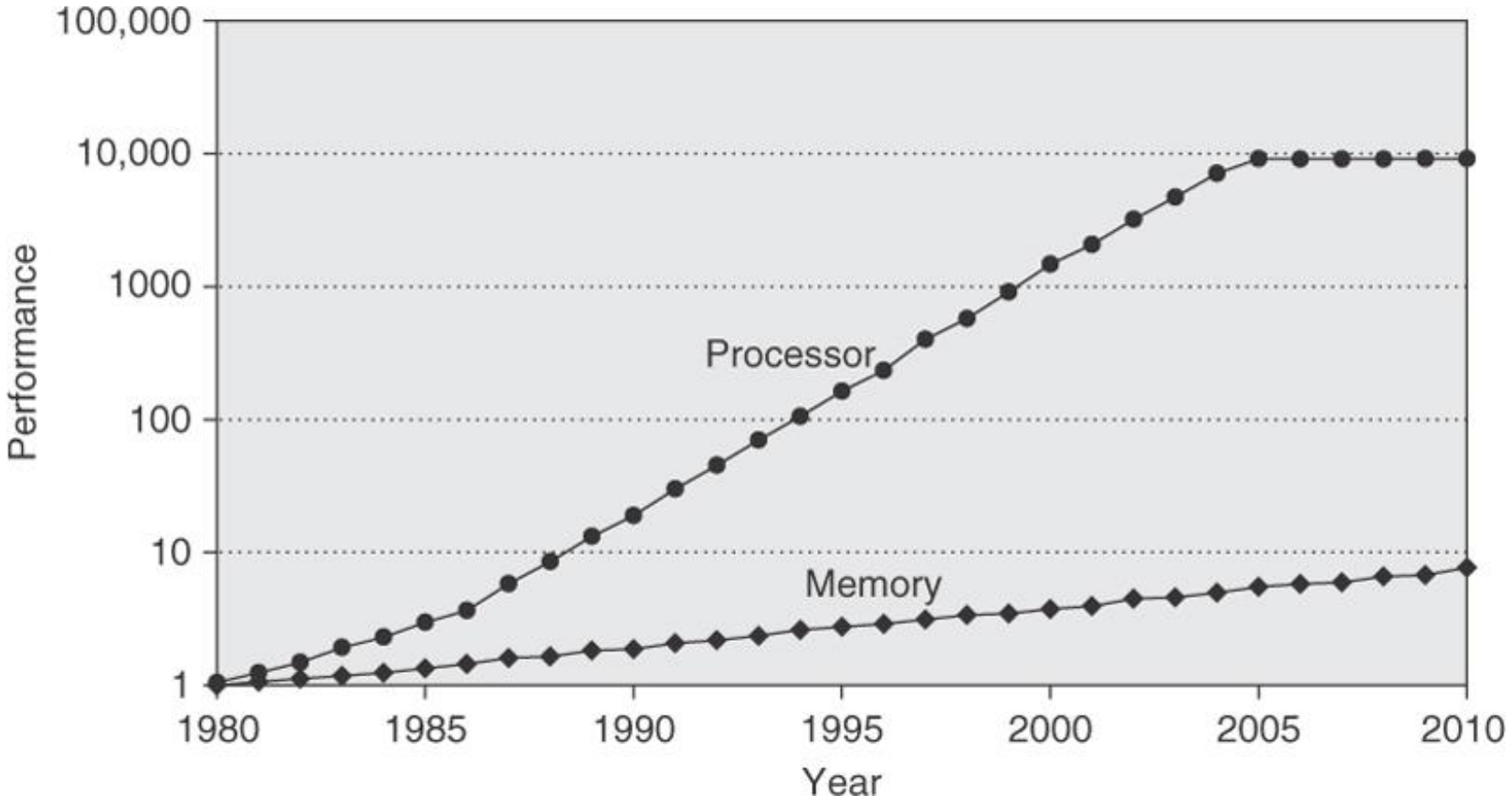
Festplatte (3)

- Vorteile
  - Billig (im Vergleich zu Hauptspeicher)
  - Große Datendichte
- Nachteile
  - Langsam (im Vergleich zu Hauptspeicher)
    - Im Bereich von Millisekunden
    - Aber: Beschleunigung durch Caching, Betriebssystemunterstützung etc.
  - Mechanischer Aufbau
    - Anfällig bei Erschütterungen etc.
    - Verschleiß

CACHES

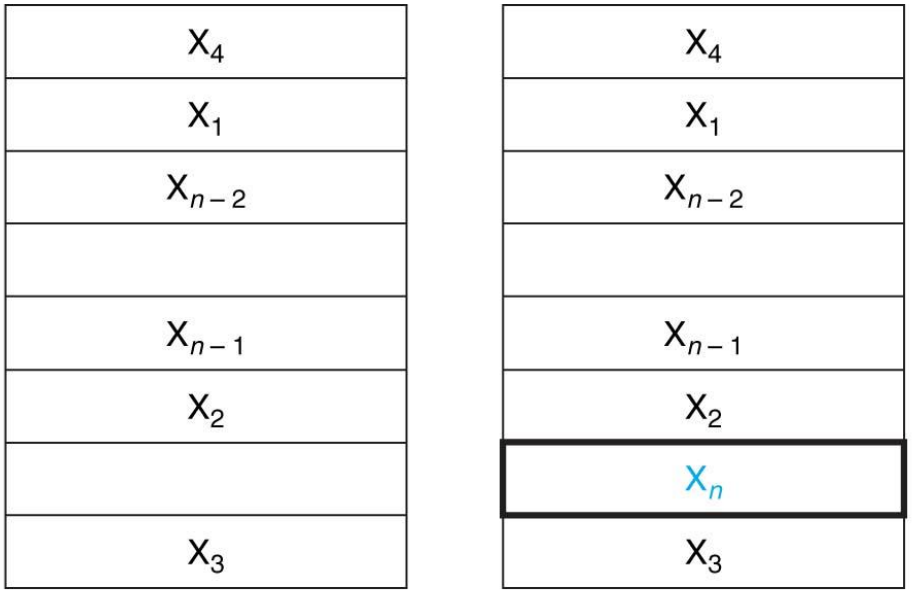
Historische Entwicklung

- Entwicklung der relativen Leistung von CPU und Speicher
  - Prozessor: ca. 55% pro Jahr (Verdopplung alle 1.5 Jahre bis 2005)
  - Speicher: ca. 7% pro Jahr (Verdopplung alle 10 Jahre)



Caches

- Cache-Speicher
  - Zwischen Prozessor und Hauptspeicher
- Es sei folgende Zugriffssequenz gegeben: X_1, … ,X_{n-1},X_n
  - Nachfolgend ein einfacher Cache (Block = 1 Wort)
  - Vor und nach dem Zugriff auf ein Wort X_n, das sich noch nicht im Cache befindet



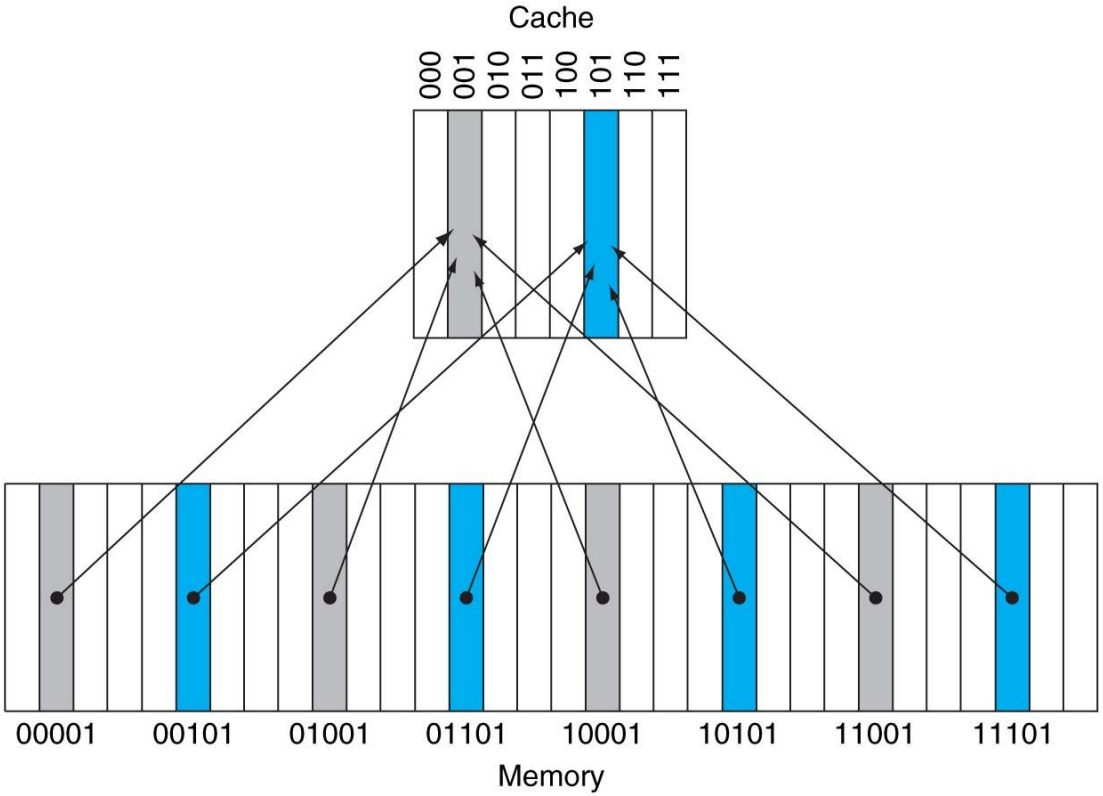
a. Before the reference to X_n b. After the reference to X_n

Wie erkennt man, ob die Daten vorhanden sind?

Wo sehen wir nach?

Direkt abgebildeter Cache

- Direkt abgebildeter Cache (*direct-mapped cache*)
  - Jede Speicheradresse wird auf genau eine Position im Cache abgebildet
  - (Blockadresse) modulo (Anzahl der Cache-Blöcke im Cache)
  - Anzahl der Cache-Blöcke ist eine Zweierpotenz
  - Die unteren (rechten) Bits der Adresse betrachten



Tags und Gültigkeits-Bit

- Welcher Block aus dem Hauptspeicher befindet sich in einem bestimmten Cache-Block?
  - Blockadresse und Daten speichern
  - Man benötigt nur den oberen Teil der Adresse
  - Wird als Tag bezeichnet
- Erkennen ob Cache-Block keine gültigen Informationen enthält
  - Gültigkeits-Bit
    - 1 = gültige Daten, 0 = nicht gültig (noch keine Daten vorhanden)
    - Am Anfang auf 0 gesetzt
  - Z.B. beim Programmstart

Cache-Beispiel (1)

- 8 Cache-Blöcke, 1 Wort pro Block, direkt abgebildet
- Anfangszustand

Index	V	Tag	Daten
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache-Beispiel (2)

Wortadresse	Binäradresse	Hit/Miss	Cache-Block
22	10 110	Miss	110

Index	V	Tag	Daten
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache-Beispiel (3)

Wortadresse	Binäradresse	Hit/Miss	Cache-Block
26	11 010	Miss	010

Index	V	Tag	Daten
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache-Beispiel (4)

Wortadresse	Binäradresse	Hit/Miss	Cache-Block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Daten
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache-Beispiel (5)

Wortadresse	Binäradresse	Hit/Miss	Cache-Block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

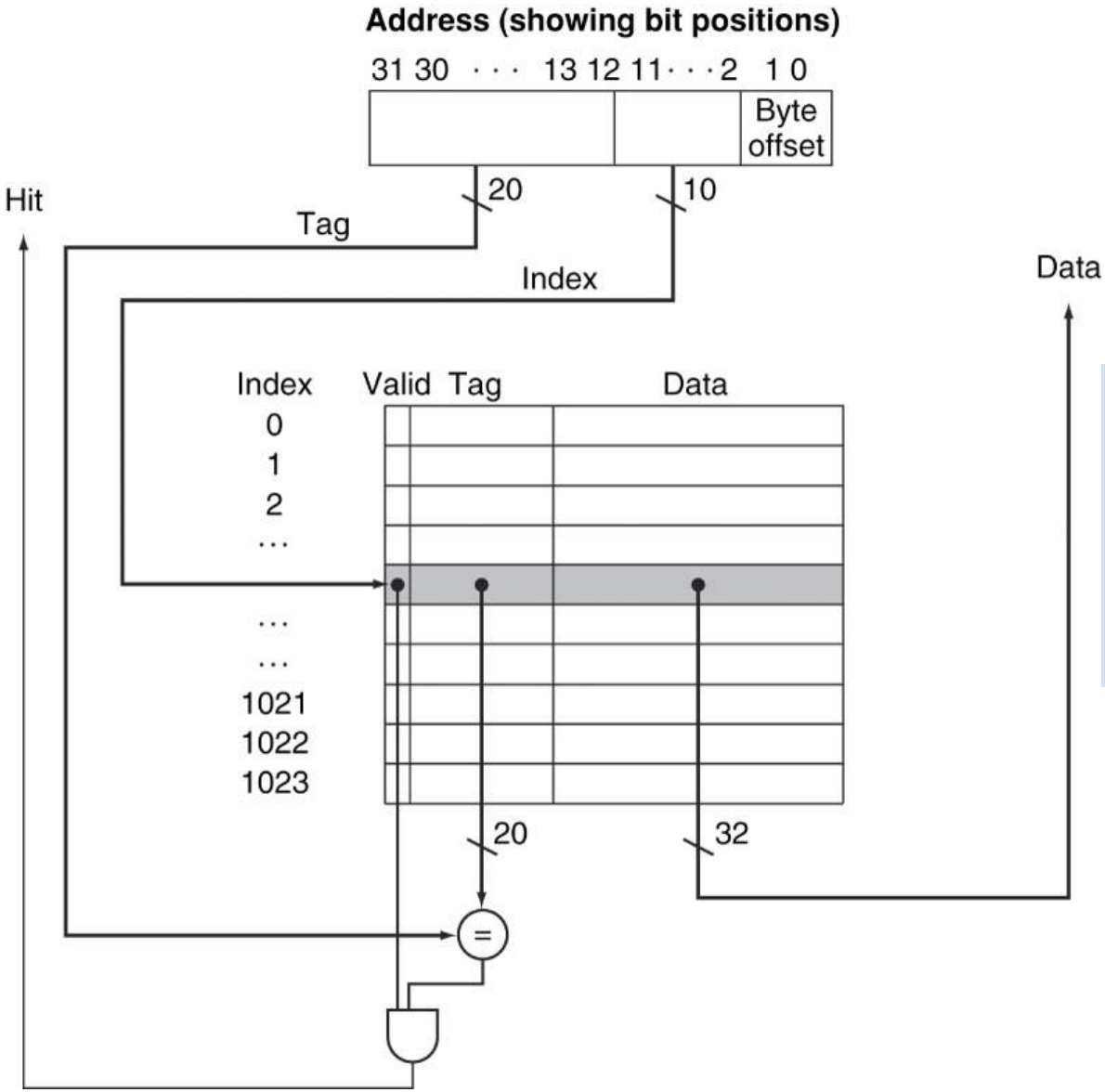
Index	V	Tag	Daten
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache-Beispiel (6)

Wortadresse	Binäradresse	Hit/Miss	Cache-Block
18	10 010	Miss	010

Index	V	Tag	Daten
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

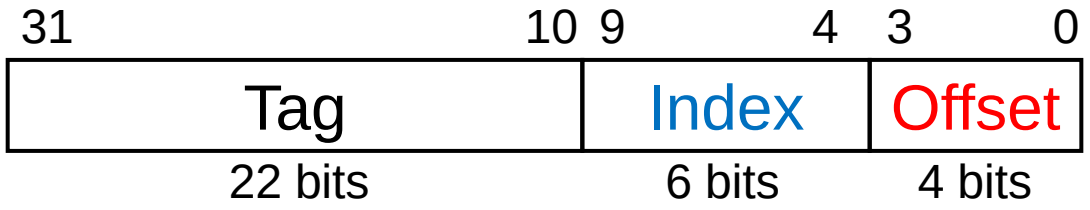
Adressaufteilung (32 Bit Adresse, 4 Bytes pro Block)



4 KiB-Cache (1024 Blöcke zu 4 Byte)
+
1024 × 21 Bits für Information (Valid, Tag)

Beispiel (32 Bit Adresse)

- Gegeben: 64 Blöcke, 16 Bytes/Block
  - In welchem Block befindet sich Adresse 1200?
  - 64 Blöcke = 6 Bits für den Index
  - 16 Bytes/Block = 4 Bits für den Offset
- Blockadresse = $\lfloor 1200/16 \rfloor = 75$
- Blocknummer = 75 modulo 64 = 11
- Binärmuster für Adresse 1200
  - 0000 0000 0000 0000 0000 01**00 1011 0000**



Blockgröße

- Größere Blöcke können die Fehlzugriffsrate reduzieren
  - Bessere Ausnutzung der räumlichen Lokalität
- Aber
  - In einem Cache mit fixer Größe gibt es dann weniger Blöcke
    - Unterschiedliche Blöcke werfen sich gegenseitig aus dem Cache
    - Kann bei zu großen Blöcken wiederum zu einer erhöhten Fehlzugriffsrate führen
  - Größere Verzögerung bei einem Fehlzugriff
    - Es müssen mehr Daten aus dem Hauptspeicher geladen werden
    - Kann mit Tricks verringert aber nicht komplett eliminiert werden
- Blockgröße ist daher ein wichtiger Designparameter

Cache Miss

- Bei einem Hit arbeitet die CPU normal weiter
- Bei einem Miss
  - CPU-Pipeline muss angehalten werden
  - Der entsprechende Block muss von der nächsten Stufe in der Speicherhierarchie angefordert werden
  - Miss bei einem Befehl
    - IF-Phase in der Pipeline wieder starten
  - Miss bei Daten
    - Warten, bis die Daten aus dem Speicher eingelesen wurden

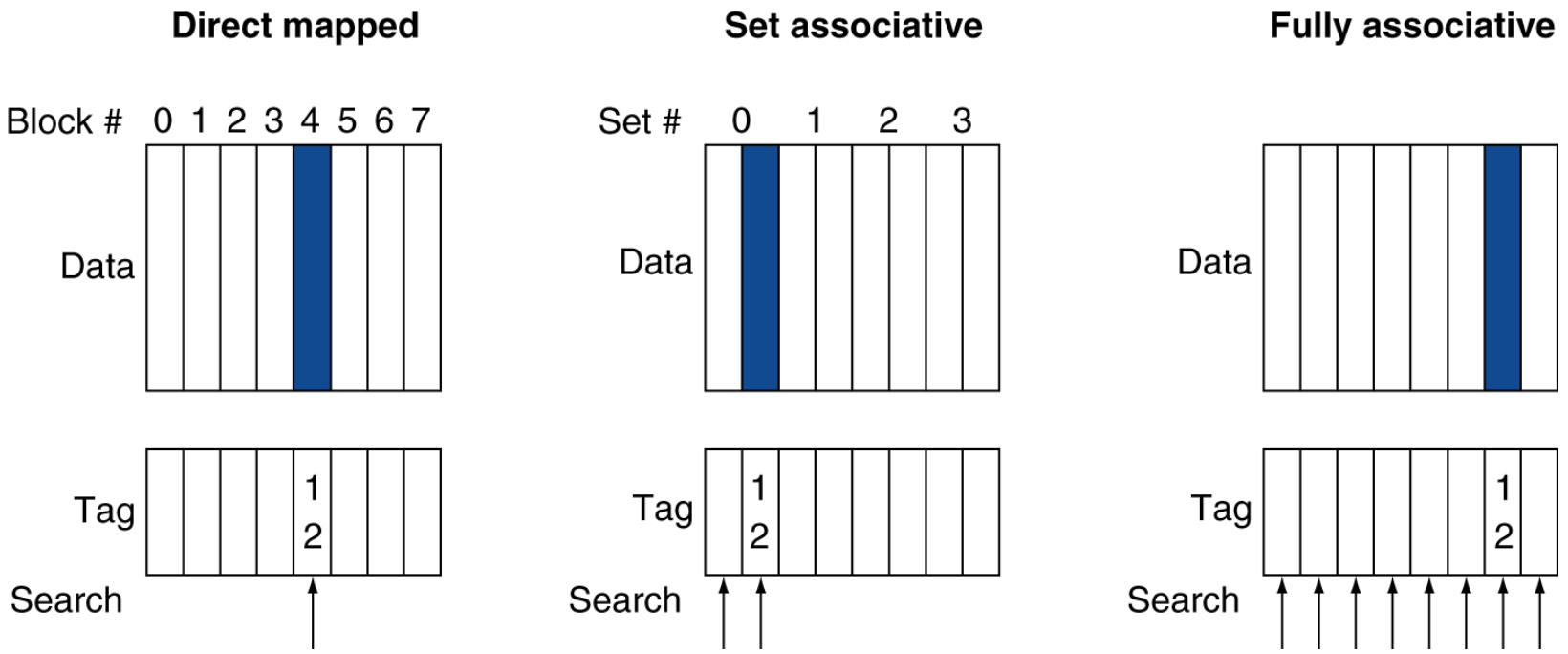
Schreiboperationen

- Bei einem Hit
  - Durchschreibetechnik (*write-through*)
    - Cache und Speicher aktualisieren
    - Daten im Speicher und Cache sind immer konsistent
  - Rückschreibetechnik (*write-back*)
    - Zunächst nur den Cache aktualisieren
    - Der veränderte Block wird erst dann in den Speicher geschrieben, wenn er ausgetauscht werden muss
    - Zusätzliches Bit (*dirty bit*) für einen Block signalisiert Veränderung
- Bei einem Miss
  - Block in den Cache laden und ändern (*write-allocate*)
  - Daten werden direkt in den Speicher geschrieben (*write-around*)
- Typische Kombinationen
  - write-back mit write-allocate
  - write-through mit write-around

Assoziative Caches

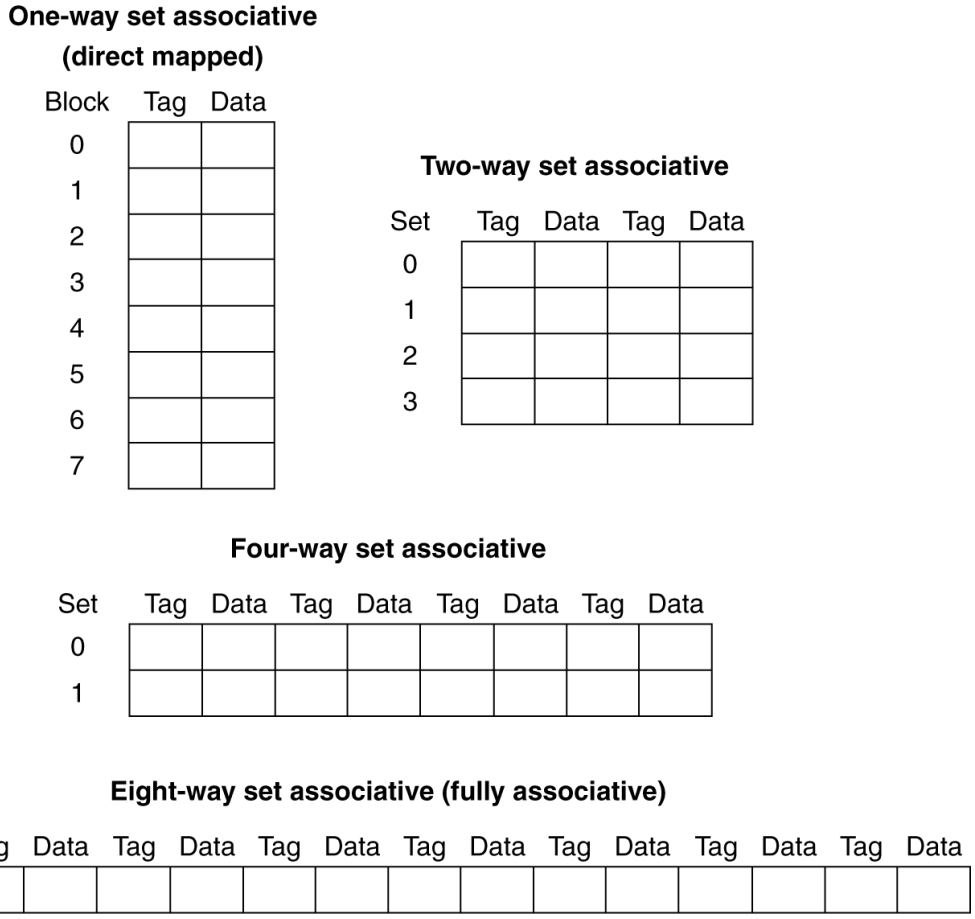
- Keine fixe Zuordnung von Speicheradresse und Position im Cache
- Zwei Ansätze
  - Vollassoziativer Cache (*fully associative cache*)
    - Block kann an jeder beliebigen Position im Cache platziert werden
    - Bei einem Zugriff müssen immer alle Cache-Einträge überprüft werden
  - Satzassoziativer Cache (*n-way set-associative cache*)
    - Block kann auf eine feste Anzahl von Plätzen (Satz) gespeichert werden
    - Jeder Satz hat n Einträge
      - Block kann einem beliebigen freien Eintrag im Satz zugeordnet werden
    - Blocknummer bestimmt den Satz
      - (Blocknummer) modulo (Anzahl der Sätze im Cache)
    - Bei einem Zugriff müssen alle Einträge in einem Satz untersucht werden

Cache-Organisationen



Spektrum der Assoziativität

- Für einen Cache mit 8 Einträgen (Tag + Daten pro Block)
  - Varianten: Direkt abgebildet, zweifach satzassoziativ, vierfach satzassoziativ, vollassoziativ



Assoziativität – Beispiel (1)

- Cache mit 4 Blöcken
  - Vereinfachung: 1 Byte pro Block
- Varianten
  - Direkt abgebildet
  - Zweifach satzassoziativ
  - Vollassoziativ
- Zugriffssequenz (Blocknummer): 0, 8, 0, 6, 8
  - Zeigt Vorteile assoziativer Caches

Assoziativität – Beispiel (2)

- Direkt abgebildet

Blockadresse	Index	Hit/Miss	Cache-Inhalt nach Zugriff			
			0	1	2	3
0	0	Miss	**Mem[0]**			
8	0	Miss	**Mem[8]**			
0	0	Miss	**Mem[0]**			
6	2	Miss	Mem[0]		**Mem[6]**	
8	0	Miss	**Mem[8]**		Mem[6]	

- Zweifach satzassoziativ

Blockadresse	Index	Hit/Miss	Cache-Inhalt nach Zugriff			
			Satz 0		Satz 1	
0	0	Miss	**Mem[0]**			
8	0	Miss	Mem[0]	**Mem[8]**		
0	0	Hit	**Mem[0]**	Mem[8]		
6	0	Miss	Mem[0]	**Mem[6]**		
8	0	Miss	**Mem[8]**	Mem[6]		

- Vollassoziativ

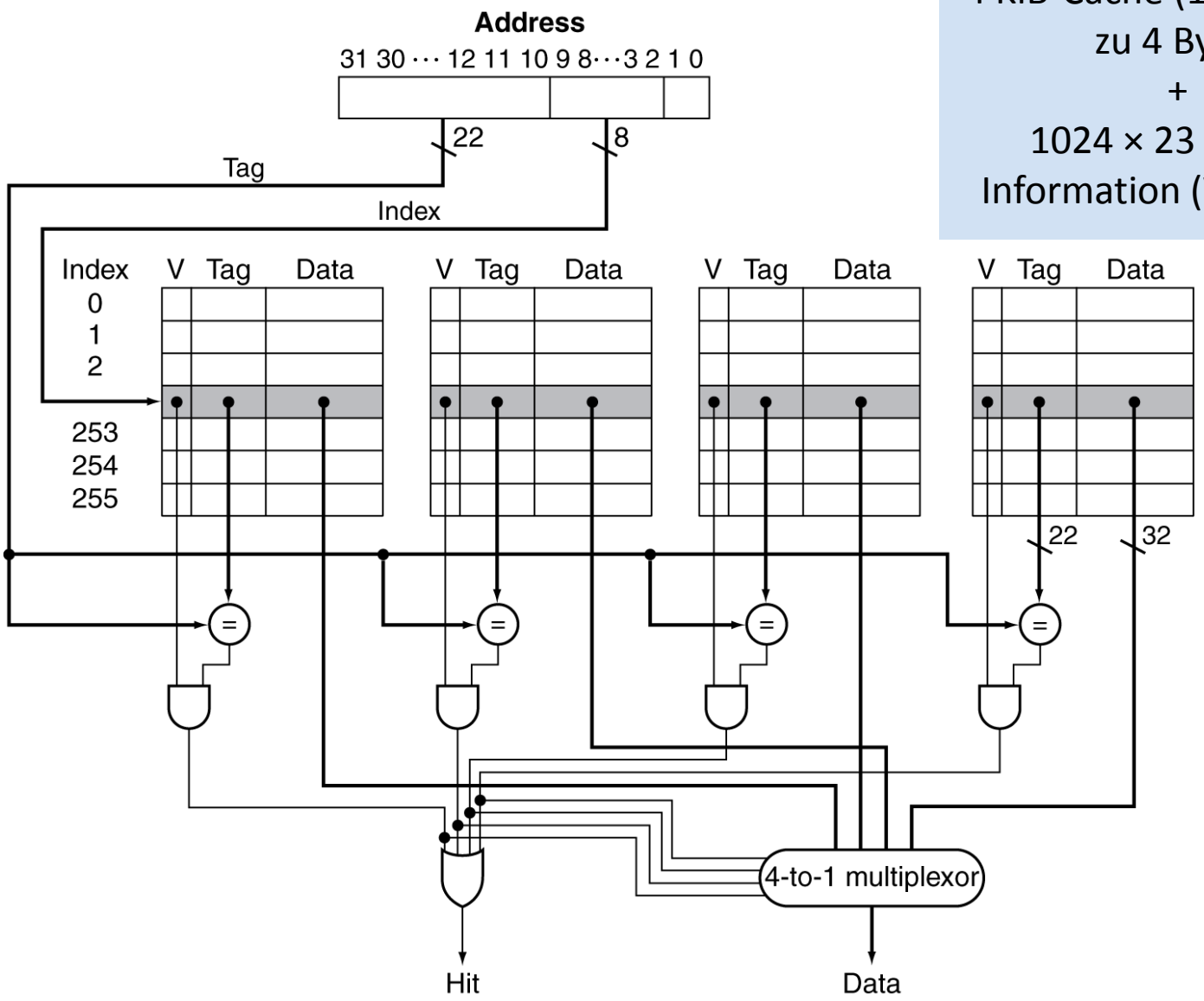
Blockadresse	Index	Hit/Miss	Cache-Inhalt nach Zugriff			
0		Miss	**Mem[0]**			
8		Miss	Mem[0]	**Mem[8]**		
0		Hit	**Mem[0]**	Mem[8]		
6		Miss	Mem[0]	Mem[8]	**Mem[6]**	
8		Hit	Mem[0]	**Mem[8]**	Mem[6]	

Assoziativität erhöhen

- Erhöhte Assoziativität verringert die Fehlzugriffsrate
  - Aber: Je höher die Assoziativität, desto kleiner die Verringerung
- Faustregeln
  - Ein zweifach satzassoziativer Cache hat typischerweise eine Fehlzugriffsrate wie ein doppelt so großer direkt abgebildeter Cache
  - Ein achtfach satzassoziativer Cache weist für die meisten Anwendungen ungefähr eine Fehlzugriffsrate wie ein vollassoziativer Cache auf

Vierfach satzassoziativer Cache – Organisation (Beispiel)

4 KiB-Cache (1024 Blöcke zu 4 Byte)
+
1024 × 23 Bits für Information (Valid, Tag)



Ersetzungsstrategie

- Direkt abgebildeter Cache
  - Keine Wahlmöglichkeit
  - Ein neu zu holender Block ersetzt den Block an gleicher Stelle
- Satzassoziativ
  - Bevorzuge Blöcke mit Valid-Bit auf 0 (nicht gültige Einträge)
  - Ansonsten wähle zwischen den Einträgen eines Satzes
- Auswahl eines Eintrages (Beispiele)
  - LRU (*least recently used*)
    - Wähle jenen Eintrag, auf den am längsten nicht mehr zugegriffen wurde
    - Hardwareseitig nur für zweifach oder vierfach satzassoziative Caches sinnvoll
  - Zufällig
    - Wähle einen zufälligen Block
    - Bei hoher Assoziativität ist die Leistung ähnlich zu LRU

Ersetzungsstrategie – Beispiel (LRU)

- Vollassoziativer Cache mit 4 Blöcken
- Zugriffssequenz (blau = Hit)

Zeitpunkt	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***
Blockadresse	1	4	8	3	2	1	2	3	1	6	5	4	1	2	1	5

- Cacheinhalt (zeitlich, vereinfacht, rot = Ersetzung)

Block 1	1	1	1	1	2	2	2	2	2	2	5	5	5	5	5	5
Block 2		4	4	4	4	1	1	1	1	1	1	1	1	1	1	1
Block 3			8	8	8	8	8	8	8	6	6	6	6	2	2	2
Block 4				3	3	3	3	3	3	3	3	4	4	4	4	4

- LRU-Ablauf (zeitliche Folge dargestellt – schematisch!)

Cacheinhalt (nach Zugriff geordnet, aktuellster Zugriff zuerst)	1	4	8	3	2	1	2	3	1	6	5	4	1	2	1	5
		1	4	8	3	2	1	2	3	1	6	5	4	1	2	1
			1	4	8	3	3	1	2	3	1	6	5	4	4	2
				1	4	8	8	8	8	2	3	1	6	5	5	4

Cache-Speicherhierarchie (1)

- Cache-Speicherhierarchie (*multilevel cache*)
  - Eine Speicherhierarchie mit mehreren Cache-Ebenen
- L1-Cache (*level one cache*)
  - Klein, schnell
- L2-Cache
  - Größer, langsamer (aber schneller als Hauptspeicher), wird nur bei Fehlzugriffen im L1-Cache kontaktiert
- L3-Cache
  - Noch größer, funktioniert zu L2 wie L2 zu L1
- Hauptspeicher
  - Wird benutzt, wenn die Daten nicht im L2 oder, falls vorhanden, nicht im L3-Cache gefunden werden

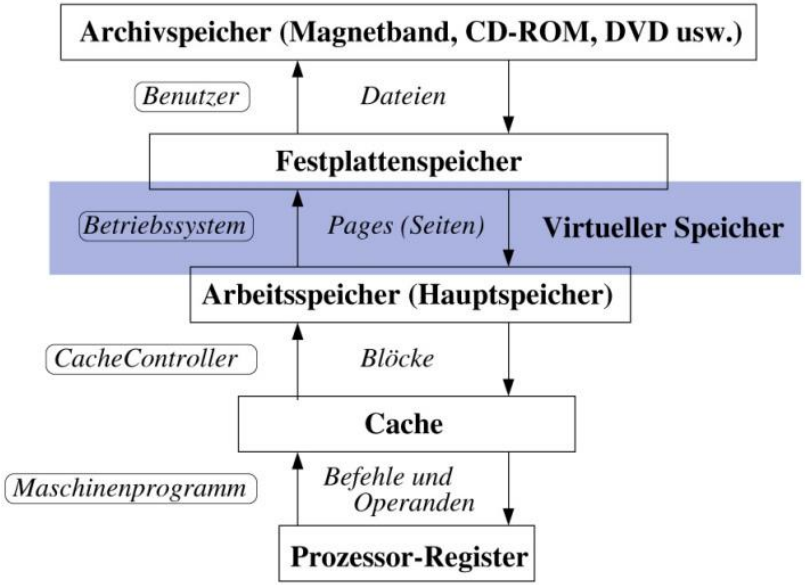
Cache-Speicherhierarchie (2)

- L1-Cache
  - Fokus auf minimaler Zugriffszeit
- L2-Cache (oder heute auch L3-Cache)
  - Fokus auf geringer Fehlzugriffsrate
  - Hauptspeicherzugriffe vermeiden
  - Zugriffszeit auf den Cache ist weniger wichtig
- Design
  - L1-Cache kleiner als L2-Cache
  - Kleinere Blockgröße im L1-Cache als im L2-Cache

VIRTUELLER SPEICHER

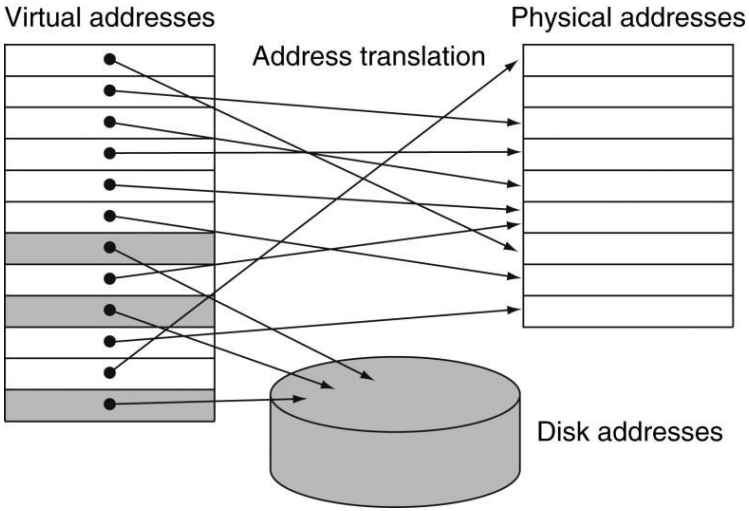
Virtueller Speicher (1)

- Hauptspeicher wird als „Cache“ für den Sekundärspeicher (z.B. Festplattenspeicher) benutzt
  - Wird von der CPU-Hardware und dem Betriebssystem gemeinsam verwaltet
  - Ein Block im virtuellen Speicher wird als Seite (*page*) bezeichnet
  - Ein Seitenfehler (*page fault*) tritt auf, wenn eine Seite, auf die zugegriffen wird, nicht im Hauptspeicher vorhanden ist



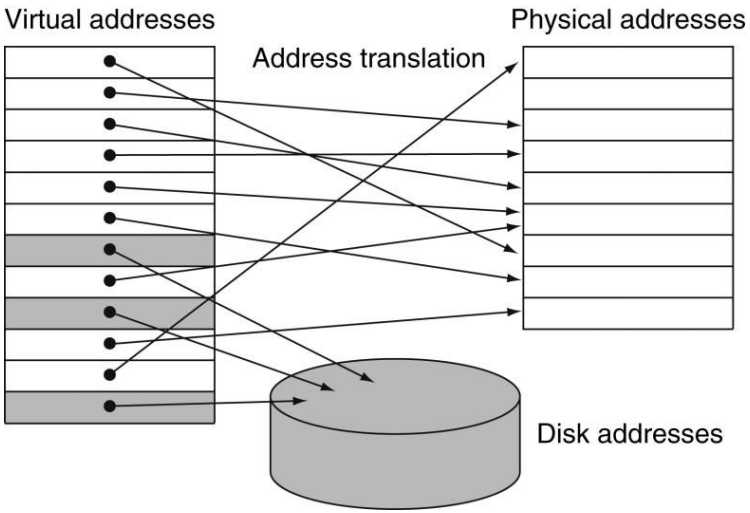
Virtueller Speicher (2)

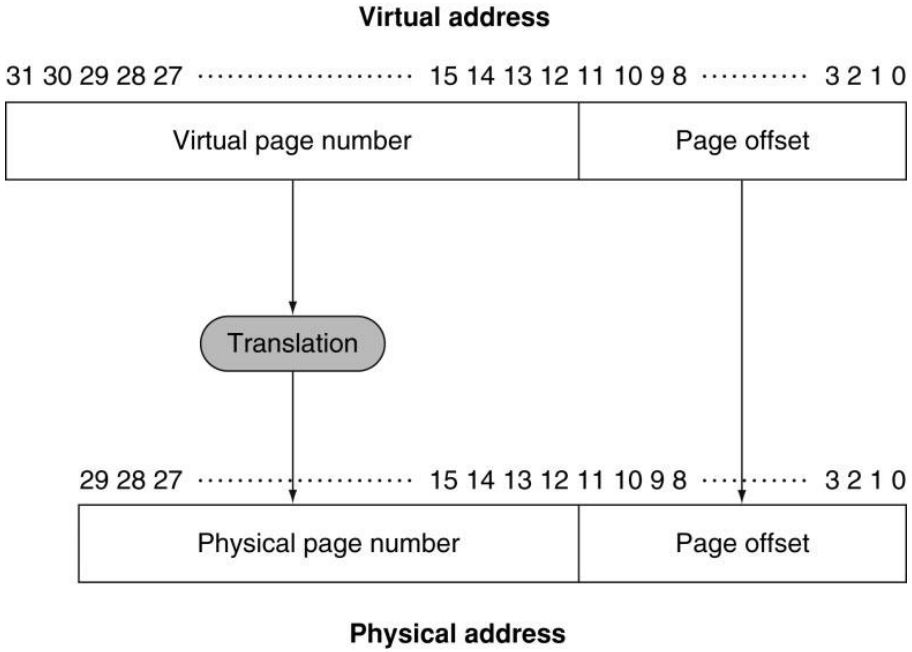
- Programme teilen sich Hauptspeicher
- Jedes Programm hat seinen privaten virtuellen Speicher (*virtual memory*)
  - Virtuelle Adresse (*virtual address*) entspricht einem Speicherplatz im virtuellen Speicher
  - Physikalische Adresse (*physical address*) entspricht einer Adresse im Hauptspeicher
  - Virtueller Speicher ist größer als tatsächlicher physikalischer Speicher



Adressübersetzung

- Adressübersetzung (*address translation*)
  - Für Seiten fixer Größe
- Beispiel (Seiten mit 4096 Bytes, d.h. Offset von 12 Bits)
  - Virtuelle Seitennummer hat 20 Bits
    - D.h. es kann 2^{20} virtuelle Seiten geben (entspricht 4 GiB)
  - Physikalische Seitennummer hat 18 Bits
    - D.h. es kann 2^{18} physikalische Seiten geben (entspricht 1 GiB)





Seitenfehler

- Bei einem Seitenfehler
  - Seite von der Festplatte holen
  - Benötigt Millionen von Taktzyklen
  - Wird vom Betriebssystem übernommen
- Designansatz
  - Seitenfehler möglichst vermeiden
  - Vollassoziativer Ansatz
  - Intelligente Ersetzungsalgorithmen

Seitentabelle

- Seitentabelle (*page table*)
  - Die Tabelle, die die Übersetzung von virtuellen in physikalische Adressen enthält
  - Wird im Speicher abgelegt
  - Als Index wird die virtuelle Adresse verwendet
  - Spezielles Register in der CPU enthält die Adresse der Seitentabelle
  - Jedes Programm besitzt eine eigene Seitentabelle
- Falls die Seite sich im Speicher befindet
  - Eintrag enthält die physikalische Seitennummer für die virtuelle Seite
  - Zusätzliche Status-Bits gespeichert
- Wenn die Seite sich nicht im Speicher befindet
  - Dann kann die Seitentabelle auf einen Speicherort auf der Festplatte zeigen
  - Dieser Austauschspeicher (*swap space*) wird für den vollständigen virtuellen Speicher eines Programms reserviert

Benutzung der Seitentabelle

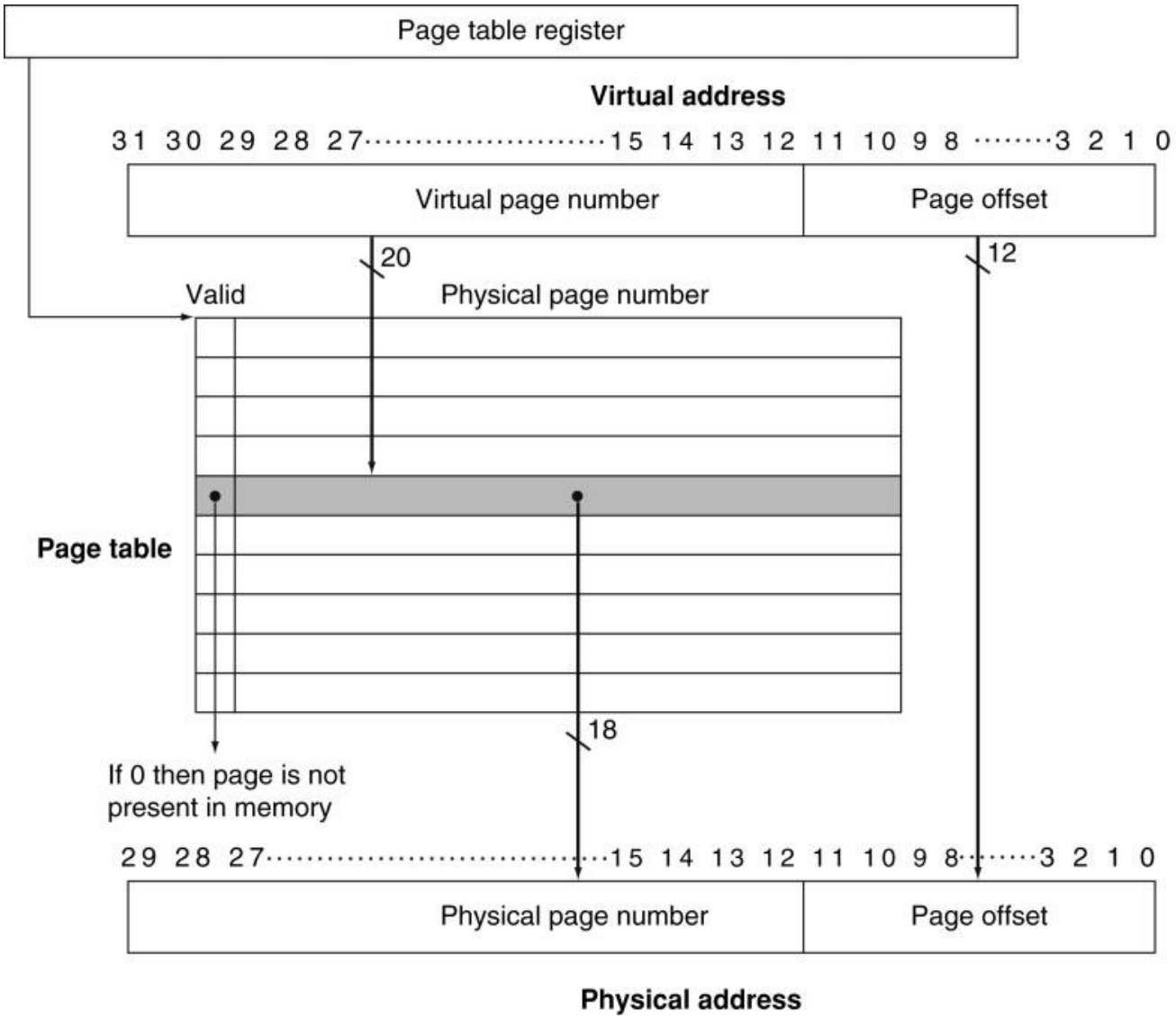
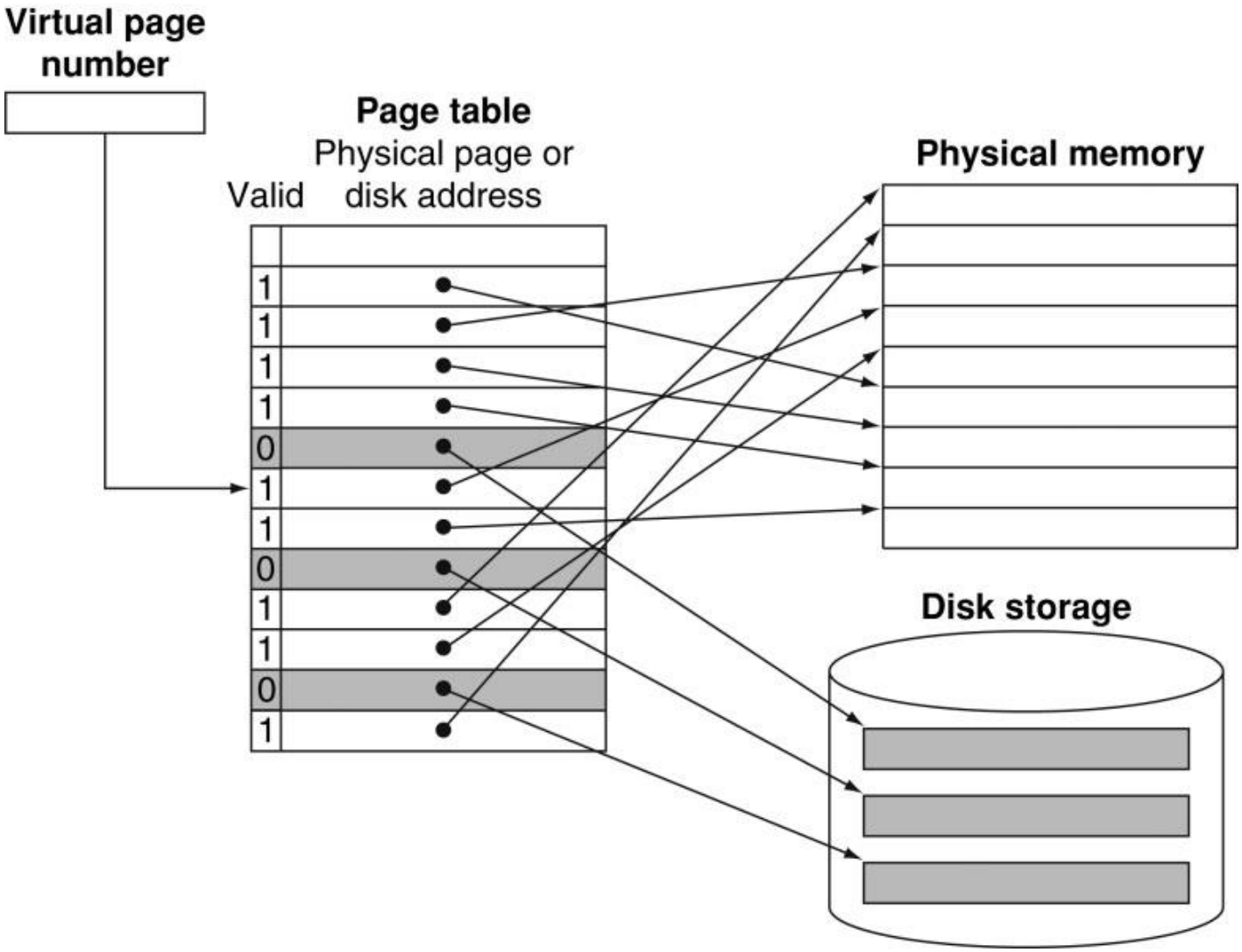


Abbildung von Seiten auf Speicher



Ersetzungsstrategie und Schreibstrategie

- LRU-Variante
  - Use-Bit im Seitentabelleneintrag wird auf 1 gesetzt
  - Dieses Bit wird periodisch auf 0 gesetzt
  - Seiten mit diesem Bit auf 0 werden bei der Ersetzung bevorzugt
- Schreibzugriffe benötigen sehr lange
  - Die Zugriffe werden geblockt und dann als Ganzes geschrieben
  - Durchschreibetechnik nicht praktikabel
  - Rückschreibetechnik verwenden
  - Wird in eine Seite geschrieben, dann wird ein entsprechendes Bit (dirty bit) gesetzt

SPEICHERHIERARCHIE - VERGLEICH

Speicherhierarchie - Vergleich

- Bestimmte Prinzipien treten auf den verschiedenen Stufen der Speicherhierarchie immer wieder auf
- Auf jeder Stufe in der Hierarchie müssen Entscheidungen getroffen werden
  - Wo kann ein Block platziert werden?
  - Wie findet man einen Block?
  - Welcher Block soll bei einem Fehlzugriff ersetzt werden?
  - Was passiert bei einer Schreiboperation?

Speicherhierarchie – Block platzieren

- Hängt von der Assoziativität ab
  - Direkt abgebildet
    - Nur eine Möglichkeit
  - Satzassoziativ (n Einträge)
    - n Möglichkeiten innerhalb eines Satzes
  - Vollassoziativ
    - Möglichkeiten entsprechen Anzahl der Blöcke
- Höhere Assoziativität reduziert die Fehlzugriffsrate
  - Vergrößert aber die Komplexität, Kosten und Zugriffszeiten

Speicherhierarchie – Block finden

Assoziativität	Suchmethode	Erforderliche Vergleiche
Direkt abgebildet	Index	1
Satzassoziativ	Indizierung des Satzes Suche innerhalb des Satzes	Assoziativitätsgrad
Vollassoziativ	Durchsuchen aller Einträge	Anzahl der Einträge

- Hardware-Caches
  - Möglichst wenige Vergleiche machen um Kosten zu reduzieren
- Virtueller Speicher
  - Voller Lookup (vollassoziativ) möglich
  - Geringere Fehlzugriffsrate

Speicherhierarchie – Ersetzung

- Auswahl jenes Eintrags, der ersetzt werden soll
  - LRU
    - Komplex
    - Erfordert mehr Hardware für höhere Assoziativitätsgrade
  - Zufällig
    - Ähnlich LRU
    - Einfacher zu realisieren
- Virtueller Speicher
  - LRU bevorzugt
  - LRU approximiert mit Hilfe von Hardware-Unterstützung

Speicherhierarchie – Schreiboperationen

- Durchschreibetechnik
  - Update erfolgt auf beide betroffenen Ebenen in der Speicherhierarchie (z.B. Cache und Hauptspeicher)
  - Vereinfacht Ersetzung
- Rückschreibetechnik
  - Update erfolgt nur auf erste betroffene Stufe in der Speicherhierarchie
  - Update erfolgt auf zweite betroffenen Stufe, wenn der entsprechende Block ersetzt wird
  - Muss mehr Statusinformationen speichern
- Virtueller Speicher
  - Nur Rückschreibetechnik sinnvoll

Ursachen für Cache-Fehlzugriffe (3-C-Modell)

- Kaltstart-Fehlzugriff (*compulsory miss*)
  - Wenn zum ersten Mal auf einen Block zugegriffen wird
- Speicherüberlastungs-Fehlzugriff (*capacity miss*)
  - Wenn wegen zu geringer Kapazität Verdrängungen benötigter Blöcke auftreten
- Adresskonflikt-Fehlzugriff (*conflict miss*)
  - Wenn benötigte Blöcke wegen Konflikten verdrängt werden

Cache-Design

Designänderung	Auswirkung auf die Fehlzugriffsrate	Mögliche negative Auswirkungen auf die Leistung
Steigerung der Cache-Größe	Senkt die Speicherüberlastungs-Fehlzugriffe	Kann die Zugriffszeit erhöhen
Steigerung der Assoziativität	Senkt die Fehlzugriffsrate von Adresskonflikt-Fehlzugriffen	Kann die Zugriffszeit erhöhen
Steigerung der Blockgröße	Senkt die Fehlzugriffsrate für viele Blockgrößen aufgrund räumlicher Lokalität	Erhöht den Fehlzugriffsaufwand Sehr große Blöcke können die Fehlzugriffsrate wieder erhöhen

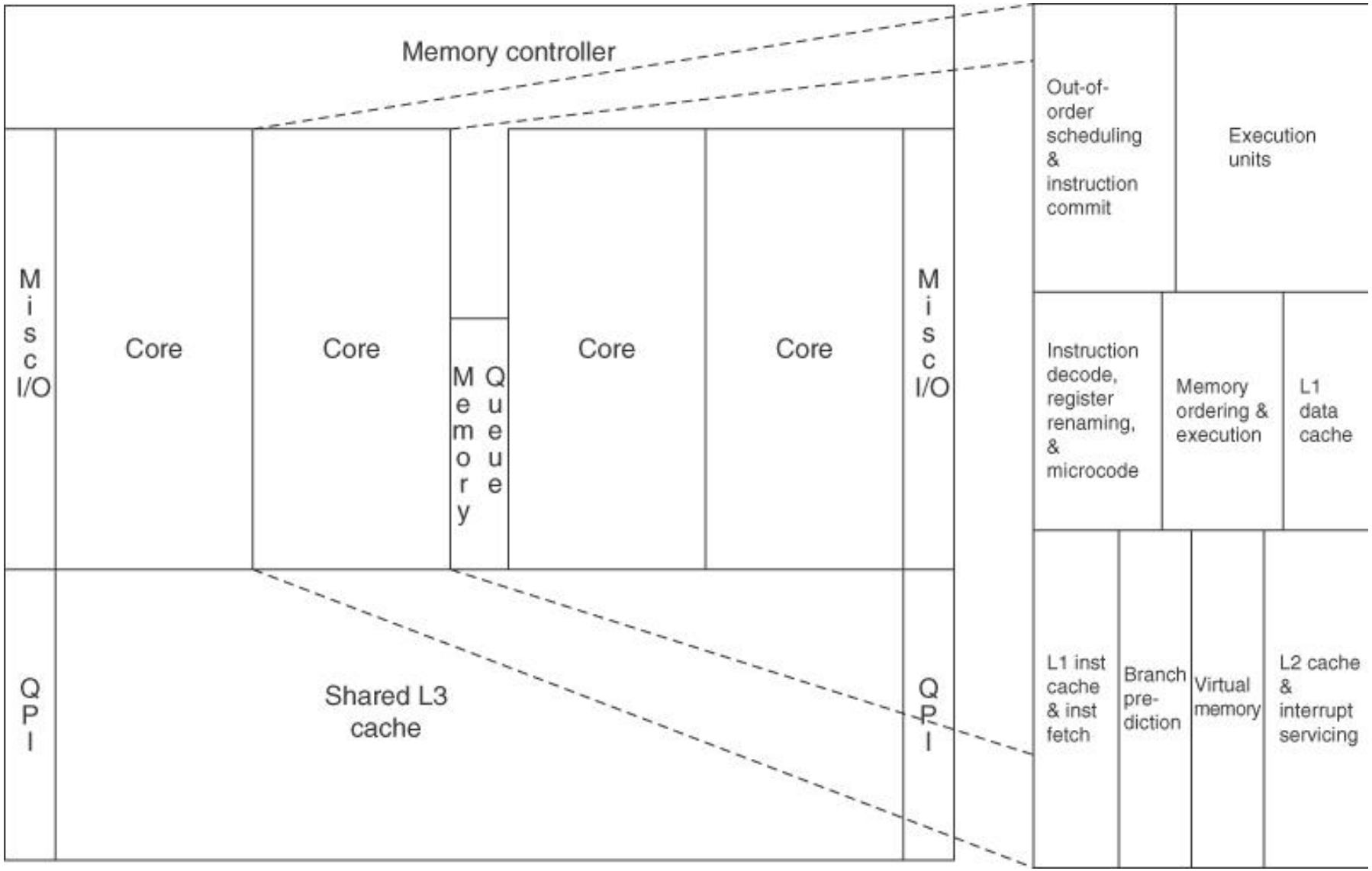
Ausblick – Multicore-Prozessoren

- Moderne Prozessoren haben mehrere Kerne (Cores)
  - Die Prozessorkerne benutzen einen gemeinsamen physischen Adressraum und haben lokale Caches
- Problem
  - Cache-Einträge in den unterschiedlichen Caches für den gleichen Block im Hauptspeicher müssen konsistent bleiben
- Beispiel
  - 2 Kerne A und B, Problem bei Durchschreibetechnik

Zeit	Ereignis	Cache für A	Cache für B	Speicherinhalt für X
0				0
1	Core A liest X	0		0
2	Core B liest X	0	0	0
3	Core A schreibt 1 in X	1	0	1

- Cache-Kohärenz-Protokolle
  - Sicherstellen, dass alle Kerne die gleichen Daten benutzen

Moderne Prozessoren – Beispiel Core-i7



ZUSAMMENFASSUNG

Zusammenfassung

- Schnelle Speicher sind klein, große Speicher sind langsam
- Man möchte große schnelle Speicher haben
  - Caching erzeugt diese Illusion
- Lokalitätsprinzip
  - Programme benutzen hauptsächlich einen kleinen Teil ihres Adressraums
- Speicherhierarchie
  - Register ↔ L1-Cache ↔ L2-Cache ↔ … ↔ Hauptspeicher ↔ Festplatte
- Für eine gute Leistung ist intelligentes Speicherdesign wichtig

Fallstrick

Es wird beim Programmieren versäumt, das Verhalten des Speichersystems zu berücksichtigen

- Erhöhung der Lokalität beim Zugriff auf Daten
  - Gemeinsame Daten zusammenfassen (Objekte in Java, Strukturen in C)
  - Lokalität bei Schleifen berücksichtigen

    Beispiel: Loop Interchange

```
/* vorher: */
for (j=0;j<100;j=j+1)
  for (i=0;i<5000;i=i+1)
    x[i][j] = 2*x[i][j];


/* nachher: höhere Lokalität*/
for (i=0;i<5000;i=i+1)
  for (j=0;j<100;j=j+1)
    x[i][j] = 2*x[i][j];
```

Literatur

- Grundliteratur
  - D. A. Patterson, J. L. Hennessy: **Computer Organization and Design**, 4. Auflage, Morgan Kaufmann, 2013 – Kapitel 5
  - J. L. Hennessy, D. A. Patterson: **Computer Architecture: A Quantitative Approach** , 5. Auflage, Morgan Kaufmann, 2011 – Kapitel 2
  - H. Herold, B. Lurz, J. Wohlrab, **Grundlagen der Informatik**, 2. Auflage, Pearson, 2012 – Kapitel 16
  - D. W. Hoffmann: **Grundlagen der Technischen Informatik**, 4. Auflage, Hanser, 2014 – Kapitel 9