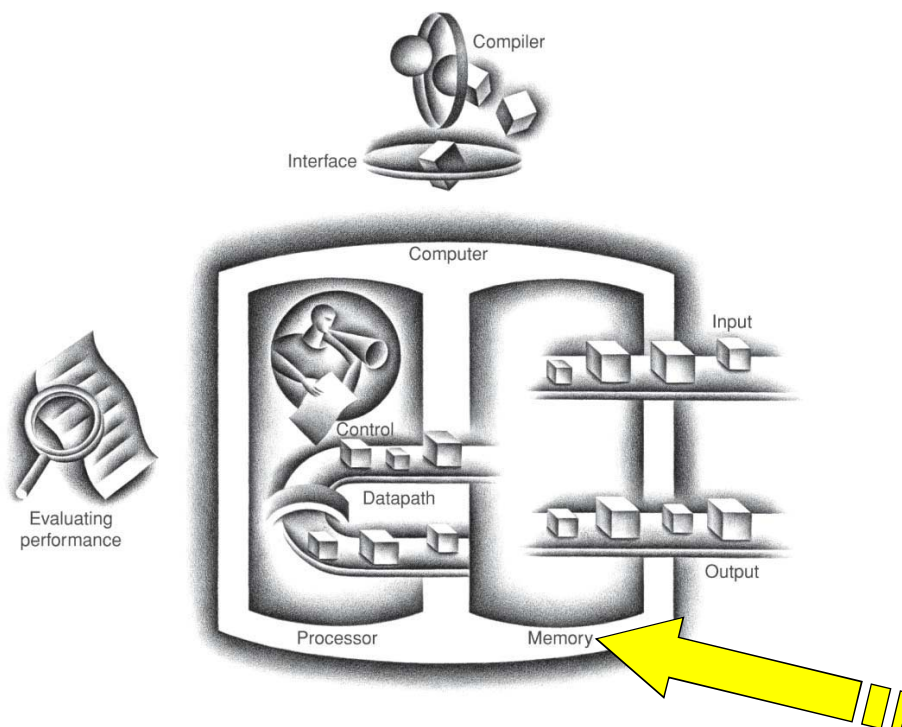
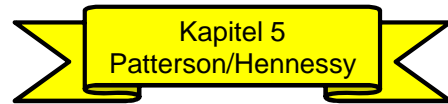


Technische Informatik 1

11. Speicherhierarchie

© Lothar Thiele

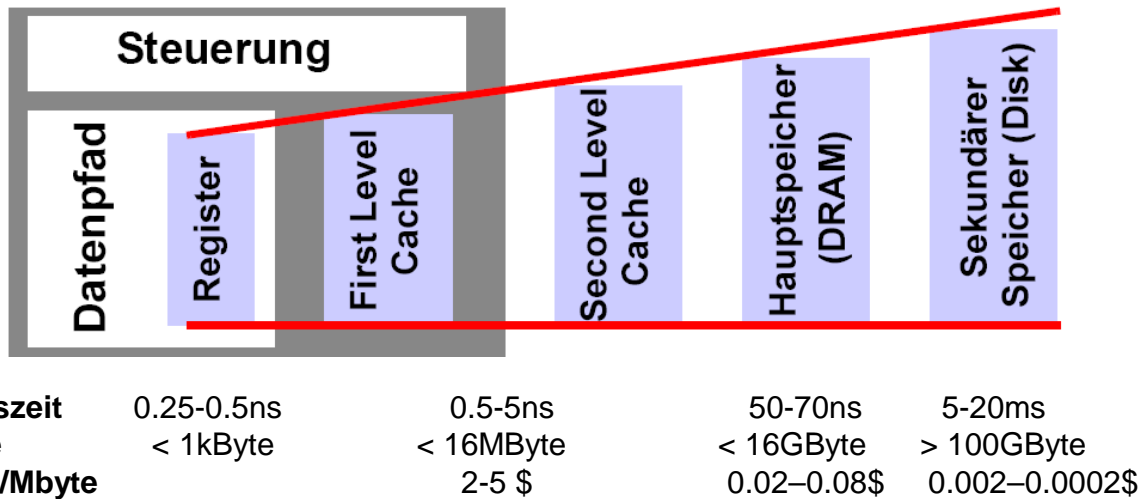
Wo sind wir ?



Übersicht

► **Ziele:**

- Dem Benutzer *möglichst viel* Speicherkapazität zur Verfügung stellen bei möglichst *kleiner Zugriffszeit*.
- Möglichst *kostengünstig*.



Technologische Trends

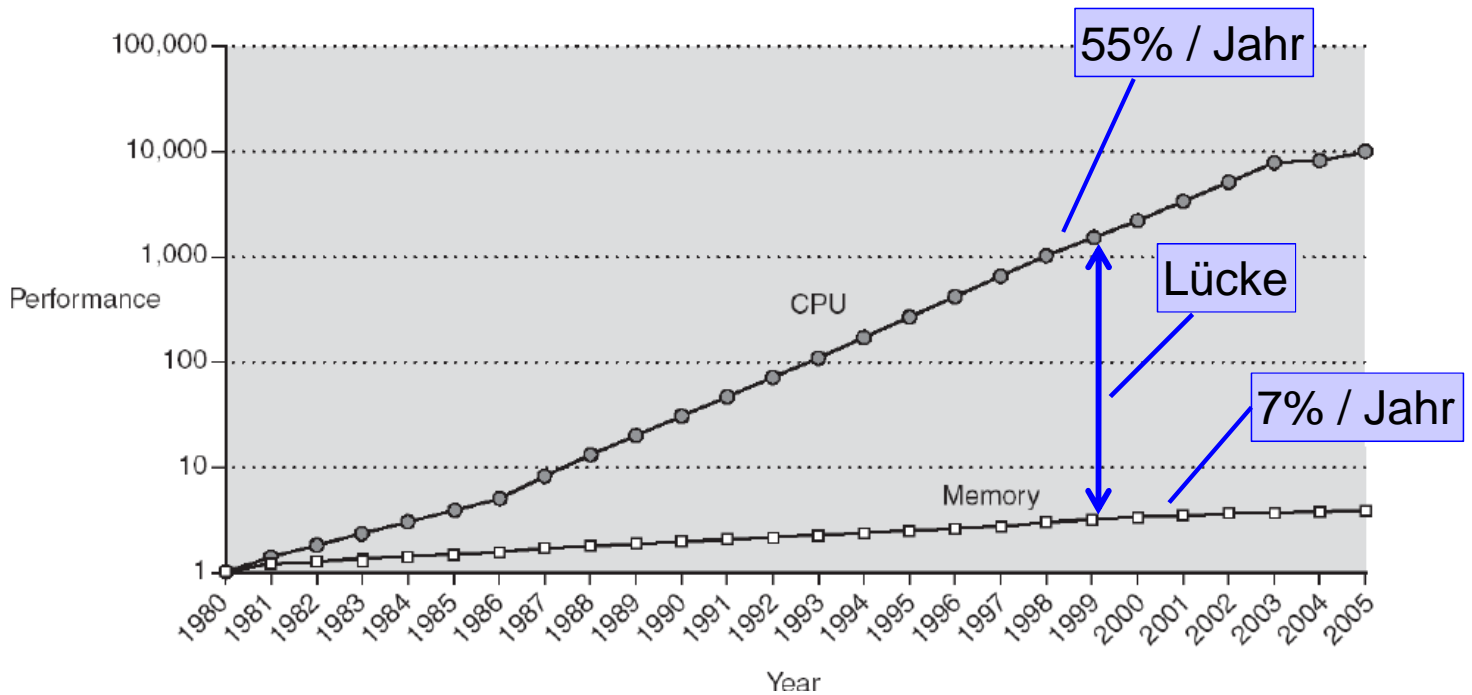
► DRAM (dynamic random access memory) Speicherbausteine:

Year introduced	Chip size	\$ per GB	Total access time to a new row/column	Column access time to existing row
1980	64 Kbit	\$1,500,000	250 ns	150 ns
1983	256 Kbit	\$500,000	185 ns	100 ns
1985	1 Mbit	\$200,000	135 ns	40 ns
1989	4 Mbit	\$50,000	110 ns	40 ns
1992	16 Mbit	\$15,000	90 ns	30 ns
1996	64 Mbit	\$10,000	60 ns	12 ns
1998	128 Mbit	\$4,000	60 ns	10 ns
2000	256 Mbit	\$1,000	55 ns	7 ns
2004	512 Mbit	\$250	50 ns	5 ns
2007	1 Gbit	\$50	40 ns	1.25 ns

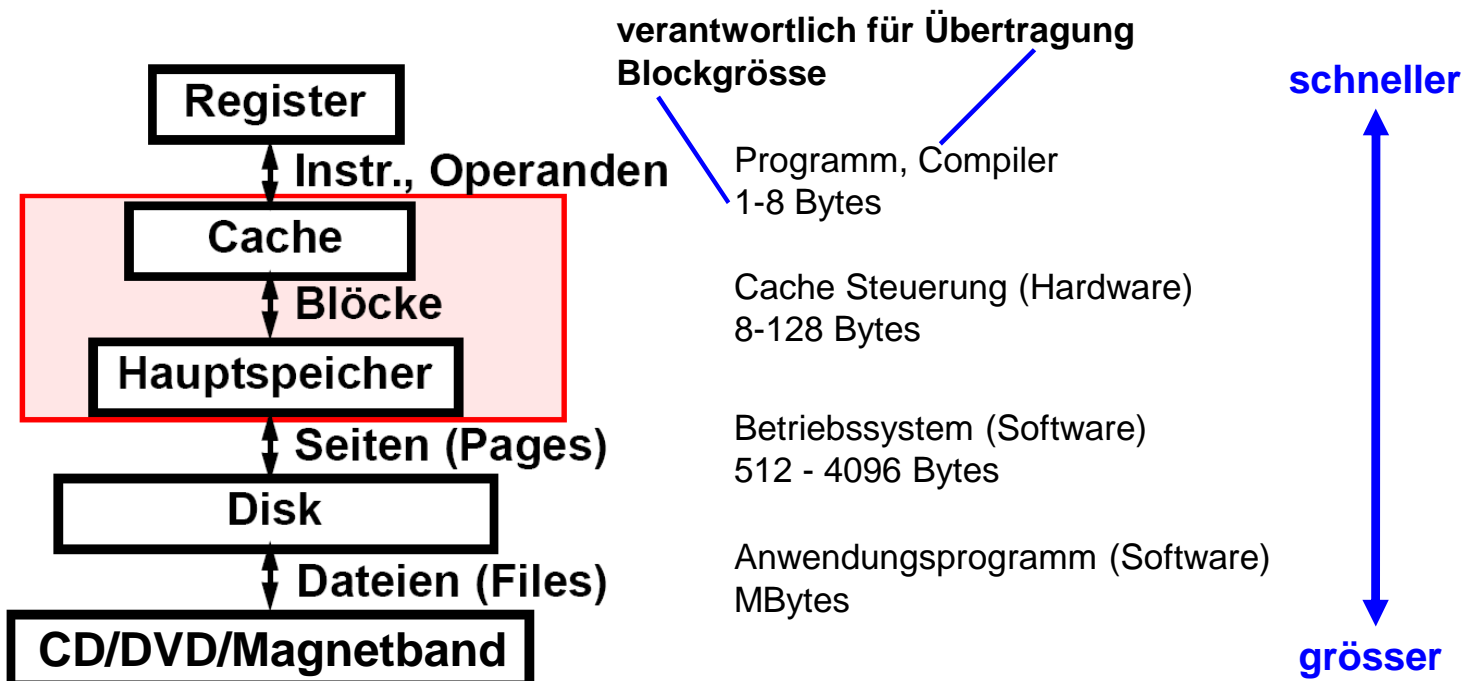
Zugriffszeit sinkt mit etwa 7% pro Jahr
Speicherkapazität steigt mit etwa 55% pro Jahr

Technologische Trends

Zugriffszeit DRAM vs. Zykluszeit Prozessor



Ebenen der Speicherhierarchie



Lokalität

- ▶ Programme greifen in einem kleinen Zeitintervall auf einen relativ kleinen, noch nicht referenzierten Teil des Adressraumes zu. Dies gilt sowohl für Instruktionen als auch für Daten.
- ▶ Zwei unterschiedliche Arten der Lokalität:
 - *Zeitliche Lokalität*: Falls ein Datum oder eine Instruktion referenziert wird, so wird sie bald wieder referenziert.
 - *Örtliche Lokalität*: Falls ein Datum oder eine Instruktion referenziert wird, werden bald Daten oder Instruktionen mit nahgelegenen Adressen referenziert.

Arbeitsprinzipien

- ▶ *Zeitliche Lokalität*: Falls ein Datum oder eine Instruktion referenziert wird, so wird sie bald wieder referenziert ->

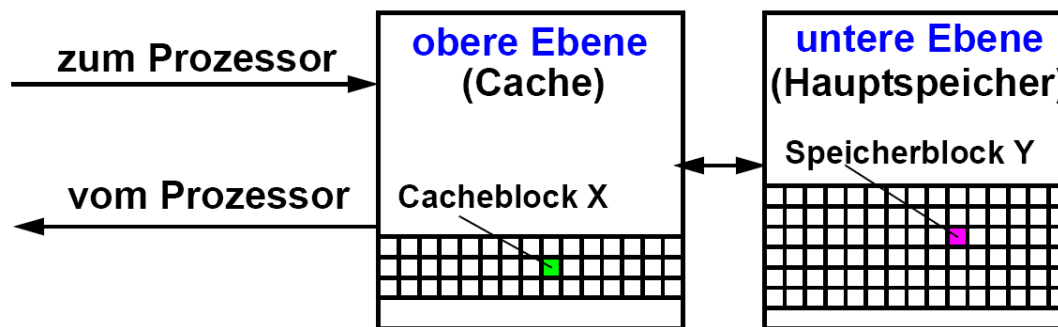
Speichern von kürzlich benutzten Informationen näher am Prozessor, d.h. in einer oberen Ebene.

- ▶ *Örtliche Lokalität*: Falls ein Datum oder eine Instruktion referenziert wird, werden bald Daten oder Instruktionen mit nahgelegenen Adressen referenziert ->

Bewege Blöcke aus aufeinanderfolgenden Wörtern näher zum Prozessor, falls eines von Ihnen benutzt wird.

Arbeitsprinzipien

- ▶ Daten werden nur zwischen aufeinanderfolgenden Ebenen kopiert.
- ▶ *Übertragungseinheit oder Blockgrösse* (z.B. Datenblock, Seite): Die kleinste Informationseinheit, die in den zwei benachbarten Hierarchieebenen vorhanden oder nicht vorhanden sein kann.
- ▶ Beispiel: Cache / Hauptspeicher



Speicherzugriff

- ▶ *Treffer (hit)*: Daten sind in der oberen Ebene vorhanden
- ▶ *Fehlzugriff (miss)*: Daten müssen aus der unteren Ebene geholt werden:
 - Anhalten der Prozessor-Pipeline (Instruktionen: IF-Phase; Daten: MEM-Phase)
 - Besorgen des gesuchten Blocks von der nächsten Ebene
- ▶ Typische Daten:

Feature	Typical values for L1 caches	Typical values for L2 caches
Total size in blocks	250–2000	15,000–50,000
Total size in kilobytes	16–64	500–4000
Block size in bytes	16–64	64–128
Miss penalty in clocks	10–25	100–1000
Miss rates (global for L2)	2%–5%	0.1%–2%

Terminologie

- ▶ **Hit-Rate:** Relativer Anteil der Speicherzugriffe, die in der oberen Ebene erfolgreich sind

- ▶ **Hit-Zeit:** Zugriffszeit zur oberen Ebene

$$\text{Hit_Zeit} = \text{Cache_Zugriffszeit} + \text{Zeit_zur_Bestimmung_von_hit_miss}$$

- ▶ **Miss-Rate:** Relativer Anteil der Speicherzugriffe, die in der oberen Ebene nicht erfolgreich sind

$$\text{Miss_Rate} = 1 - (\text{Hit_Rate})$$

- ▶ **Miss-Strafe:**

$$\text{Miss_Strafe} = \text{Zeit_zum_Finden_eines_Blocks_in_der_unteren_Ebene} + \text{Zeit_zum_Übertragen_zur_oberen_Ebene}$$

- ▶ **Mittlere Zugriffszeit:**

$$\text{Mittlere_Zugriffszeit} = \text{Hit_Zeit} + \text{Miss_Strafe} \cdot \text{Miss_Rate}$$

Implementierungsalternativen

- ▶ **Wo kann ein Block plaziert werden, wie wird er gefunden ?**

- direkte Abbildung
- teilassoziativ
- vollassoziativ

- ▶ **Welcher Block sollte bei einem Miss ersetzt werden ?**

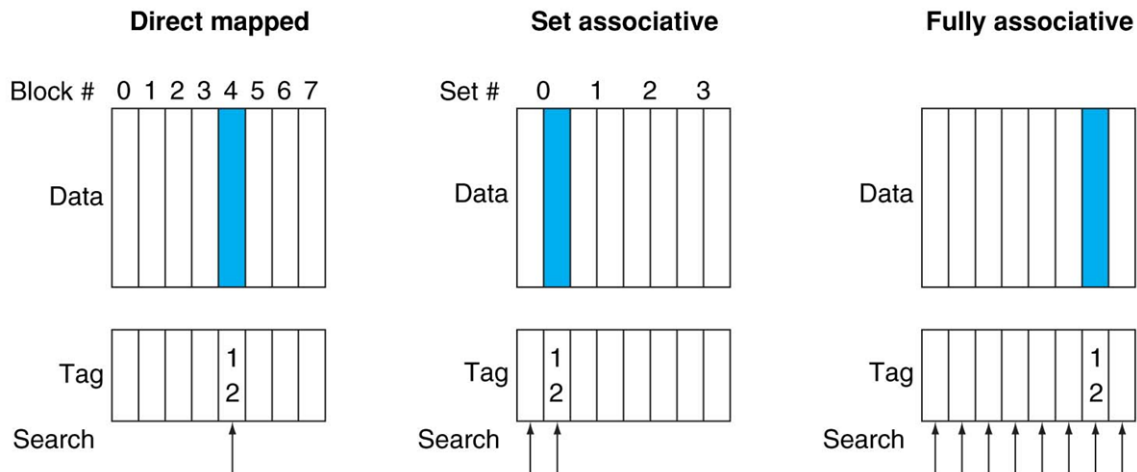
- zufällig
- am längsten unbenutzt

- ▶ **Was geschieht beim Schreiben eines Blocks ?**

- durchgängiges Schreiben
- zurückkopieren

Wo kann ein Block platziert werden?

- Platzierung eines Speicherblocks mit der Adresse 12:



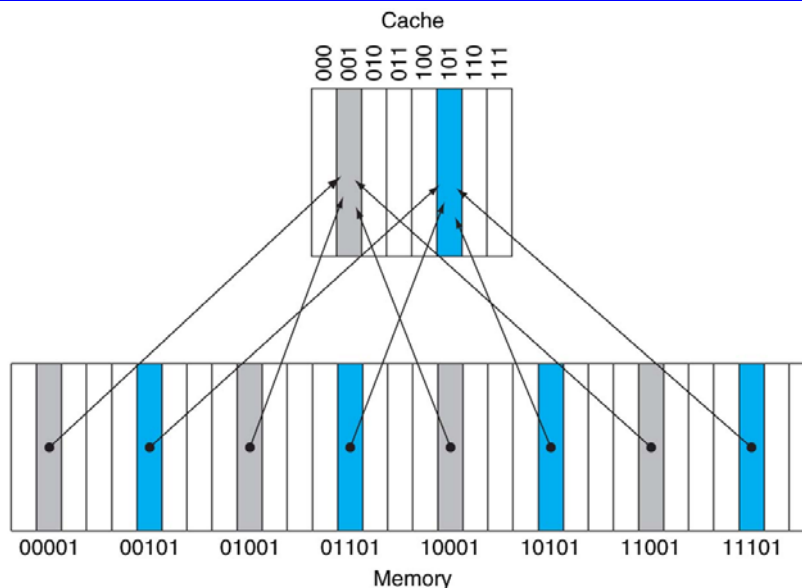
Cacheblock, in dem der Speicherblock 12 liegt:

direkte Abbildung: $(12 \bmod 8) = 4$

teillassoziativ: $(12 \bmod 4) = 0$

vollassoziativ: beliebig

Direkte Abbildung

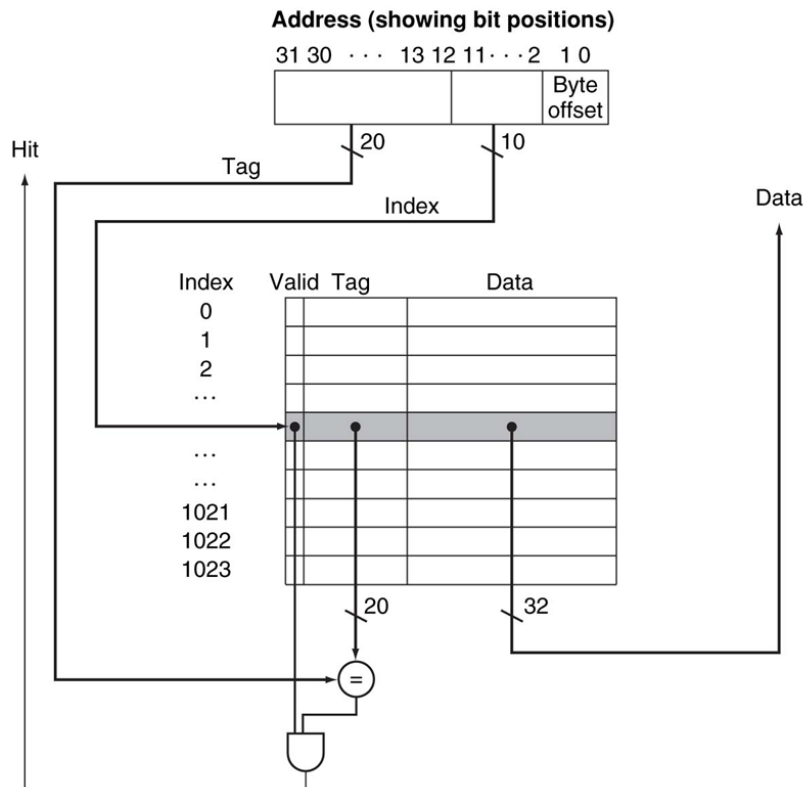


4 Bit lange Speicheradresse [4 ... 0] : 32 Daten im Hauptspeicher

Teiladresse [2 ... 0] : Cache Index (Adresse im Cache Speicher)

Teiladresse [4 ... 3] : Cache Tag (Marke zur Identifikation der Datenadresse)

Direkte Abbildung



Beispiele direkte Abbildung

- Cachegrösse:
 - Wie gross ist ein direkter Cache für 64 kByte Daten, Blockgrösse = 1 Wort = 4 Byte, byteweise Adressierung, Adresslänge 32 Bit?
 - 64 kByte = 2^{16} Byte = 2^{14} Worte
Cachegrösse = $2^{14} (32 + (32 - 2 - 14) + 1)$ Bit = 803 kBit \approx 100 kByte
- Zugriffssequenz (8 Cacheblöcke, 32 Speicherblöcke):

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Beispiele direkte Abbildung

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

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	Y	11 _{two}	Memory (11010 _{two})
011	N		
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10 _{two}	Memory (10000 _{two})
001	N		
010	Y	11 _{two}	Memory (11010 _{two})
011	Y	00 _{two}	Memory (00011 _{two})
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	Y	10 _{two}	Memory (10000 _{two})
001	N		
010	Y	11 _{two}	Memory (11010 _{two})
011	N		
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10 _{two}	Memory (10000 _{two})
001	N		
010	Y	10 _{two}	Memory (10010 _{two})
011	Y	00 _{two}	Memory (00011 _{two})
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

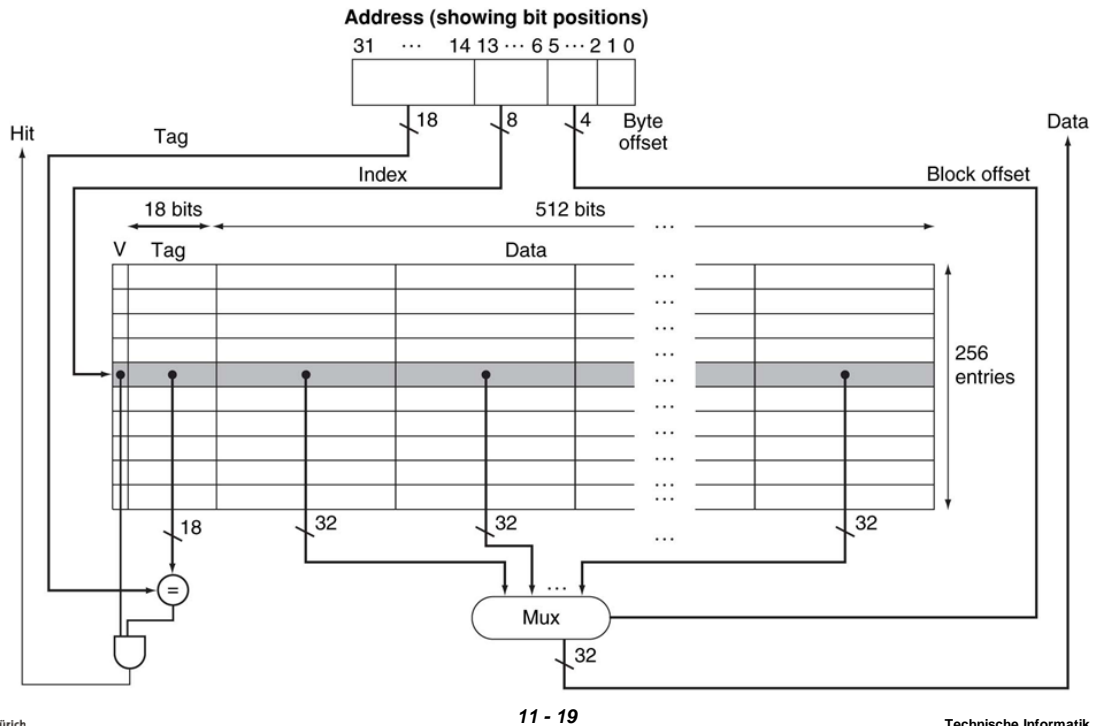
f. After handling a miss of address (10010_{two})

Vergrößerung Cacheblöcke

- ▶ **Cacheblock:** Cashedaten, die ihren eigenen Tag besitzen.
- ▶ **Vergrößerung der Blockgrösse:**
 - Vorteil aus örtlicher Lokalität ziehen
 - effizientere Speicherung
 - höhere Miss-Strafe (Zeit zum Ersetzen eines Blocks ist grösser)
- ▶ **Speichergrösse:**
 - L Bit breite Adresse, byteweise Adressierung, Cache mit 2^N Byte Nutzdaten, 2^M Byte pro Block
 - Teiladressen:
[L-1 ... N] Tag, [N-1 ... M] Cache Index, [M-1 ... 0] Block offset
 - Cachegrösse:
 $(1 + (L-N) + 8 \cdot 2^M) \cdot 2^{(N-M)}$ Bit = 2^N Byte + $(1 + L - N) \cdot 2^{(N-M)}$ Bit

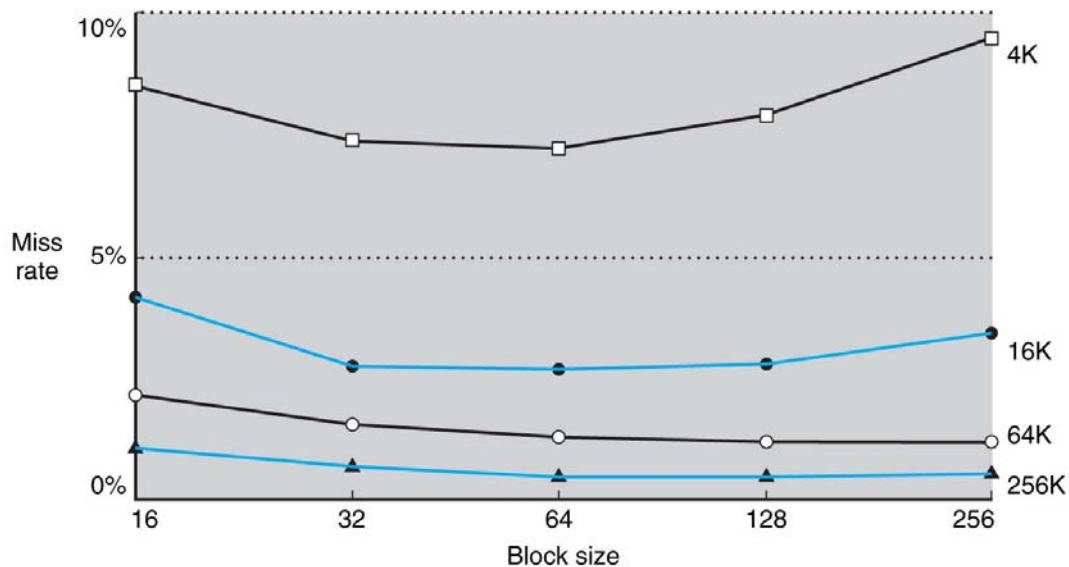
Direkte Abbildung

- ▶ Diagramme eines Caches mit direkter Abbildung, 16 kByte Nutzdaten, Blockgrösse 16 Worte, byteweise Adressierung, 32 Bit Adresslänge:



Vorteile und Nachteile

- ▶ Grosse Blöcke nutzen die örtliche Lokalität besser. Auf der anderen Seite gibt es bei grossen Blockgrösse nur wenige Blöcke im Cache; sie werden ersetzt, bevor viele der Daten im Block tatsächlich benutzt wurden.



Assoziativer Cache

► *Probleme bei direkter Abbildung:*

- Häufiger Miss aufgrund eines Konfliktes (mehrere Speicherblöcke werden auf einen Cacheindex abgebildet).
- Ungünstige festgelegte Ersetzungsstrategie bei einem Miss.
- *Abhilfe:* grösserer Cache oder mehrere Einträge pro Index (assoziativer Cache).

► *Assoziativer Cache:*

- K Einträge pro Index (K-fache Assoziativität), K direkte Caches arbeiten parallel
- Der Cacheindex selektiert eine *Menge von Blöcken*, die Tags werden mit dem entsprechenden Adressanteil parallel verglichen, die Daten werden entsprechend dem Resultat des Vergleiches selektiert.

Vergleich von Cachstrukturen

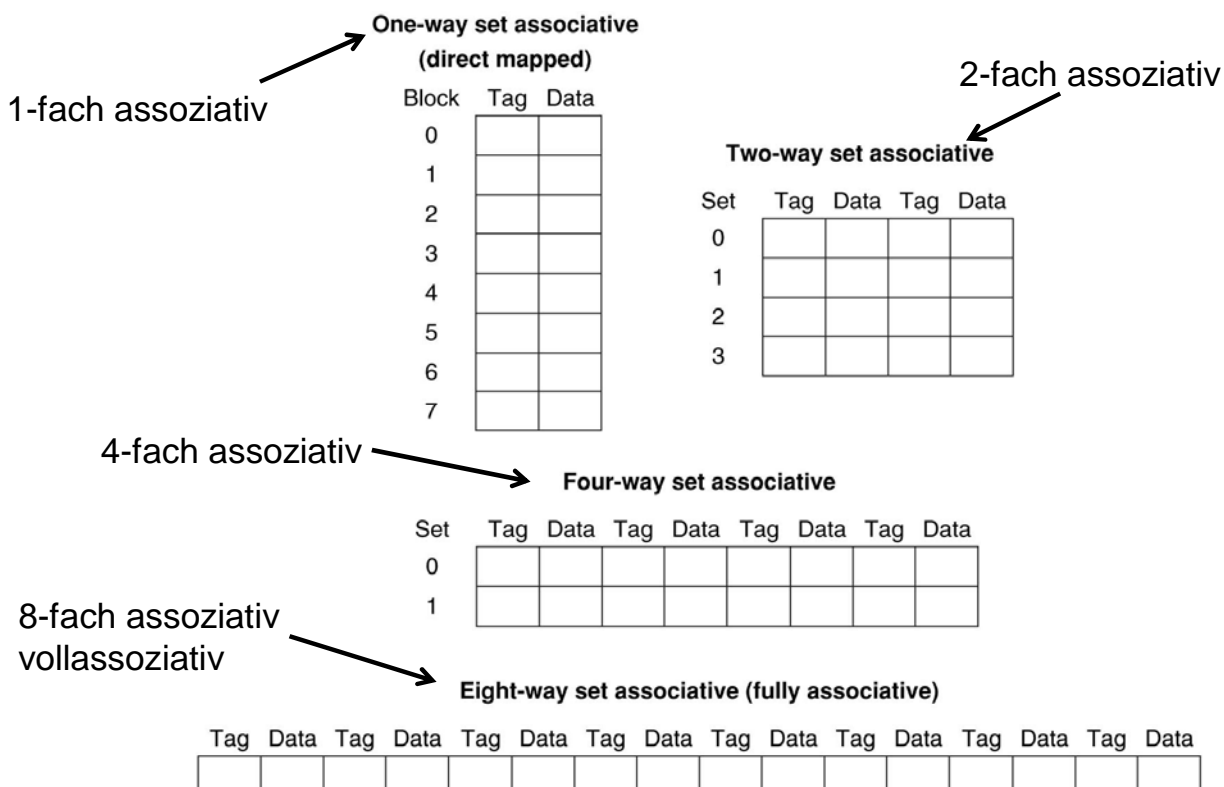
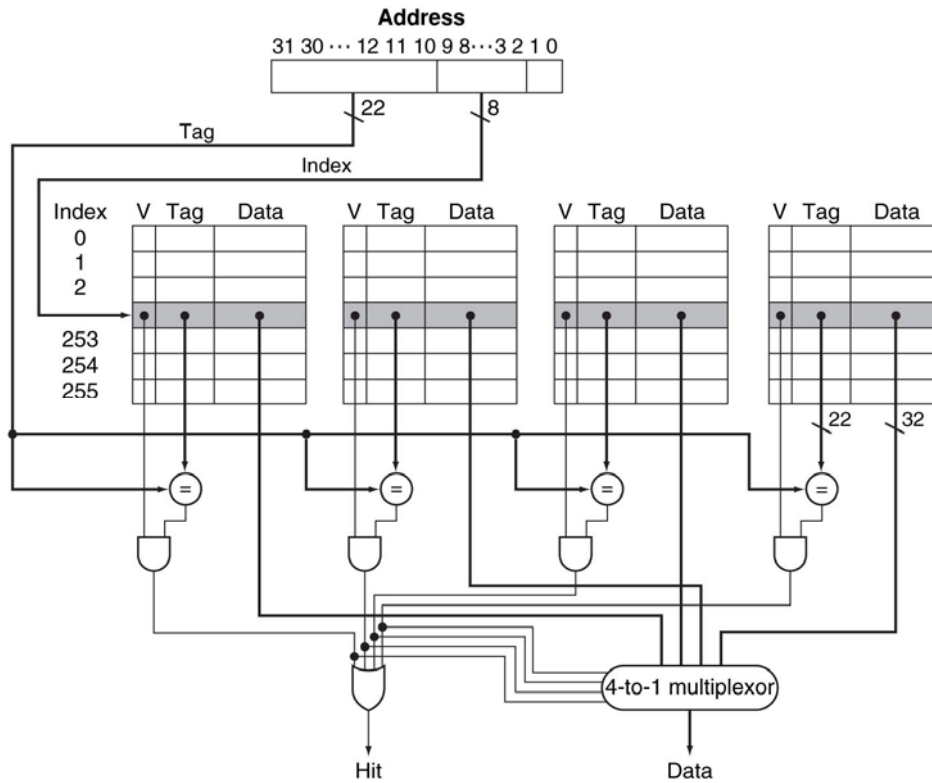
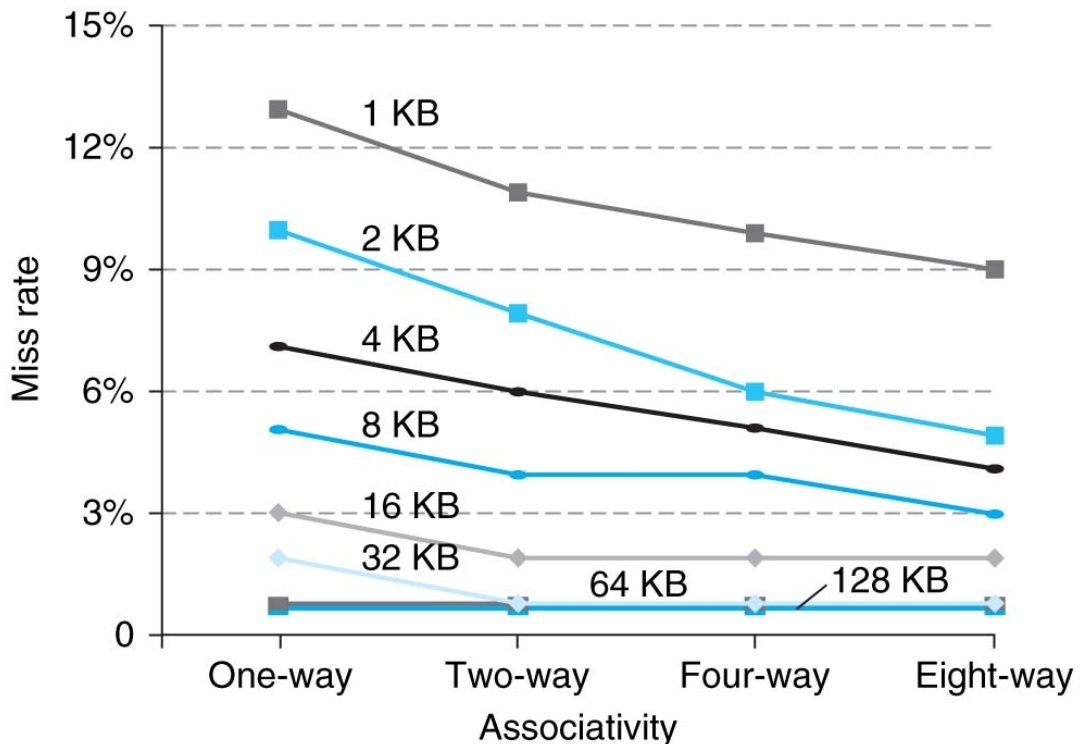


Diagramm eines teilassoziativen Caches



Vorteil der Assoziativität



Ersetzungstrategien

- ▶ *Direkte Abbildung:*
 - Jeder Speicherblock kann nur auf einen Cacheblock abgebildet werden -> keine Auswahl möglich
- ▶ *Assoziativer Cache:*
 - Jeder Speicherblock kann auf einen von K Cacheblöcken abgebildet werden; bei vollassoziativem Cache: in irgendeinen Cacheblock.
- ▶ *Einige Ersetzungsstrategien:*
 - Zufällige Auswahl des ersetzten.
 - Der am längsten unbenutzte Block wird ersetzt. Hardware speichert die jeweiligen Zugriffszeiten und ermittelt den ältesten Block (aufwendig).

Schreibalternativen

Was passiert beim Schreiben vom Cache?
Wie werden Informationen im Cache und im Hauptspeicher konsistent gehalten?

- ▶ *Zurückkopieren (write back):*
 - Prozessor schreibt nur in den Cache; falls der Block bei einem Miss ersetzt wird, wird er in den Hauptspeicher kopiert.
 - Ein "dirty bit" für jeden Cacheblock zeigt an, ob der Block zurückkopiert werden muss.
 - Aufwändige Steuerung aber kleiner Bandbreitenbedarf
 - Cache und Hauptspeicher können für lange Zeit inkonsistent sein (problematisch für Ein/Ausgabe von Daten und Mehrprozessor- und Multicoresysteme).

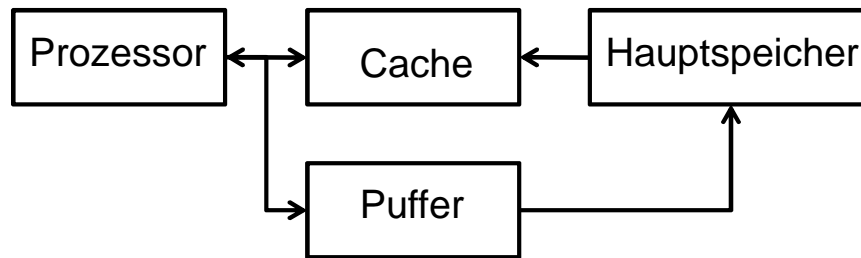
Schreibalternativen

► Durchgängiges Schreiben (*write through*):

- Mit jedem Schreibvorgang wird auch der Hauptspeicher beschreiben.
- Verlieren wir nicht den Vorteil eines Caches?
- Verwendung eines Pufferspeicher zwischen Prozessor und Speicher: Prozessor schreibt in Cache und Pufferspeicher; Pufferspeicher wird in Hauptspeicher übertragen.

▪ Voraussetzung:

$$\text{mittlere_Speicherrate} < 1/\text{Hauptspeicher_Schreibzykluszeit}$$



Leistungsberechnungen

► Grundlegende Gleichungen:

$$\text{CPU_Zeit} = (\text{CPU_Instruktionszyklen} + \text{CPU_Wartezyklen}) \cdot \text{Taktperiode}$$

$$\text{CPU_Wartezyklen} = \text{Lese_Wartezyklen} + \text{Schreib_Wartezyklen}$$

$$\text{Lese_Wartezyklen} =$$

$$\text{Leseinstruktionen/Programm} \cdot \text{Lese_Miss_Rate} \cdot \text{Lese_Miss_Strafe/Taktperiode}$$

$$\text{Schreib_Wartezyklen} =$$

$$(\text{Schreibinstruktionen/Programm} \cdot \text{Schreib_Miss_Rate} \cdot \text{Schreib_Miss_Strafe/Taktperiode}) + \text{Schreibpuffer_Wartezyklen}$$

► Vereinfachte Gleichungen:

$$\text{CPU_Wartezyklen} =$$

$$\text{Speicherinstruktionen/Programm} \cdot \text{Miss_Rate} \cdot \text{Miss_Strafe/Taktperiode} = \text{Instruktionen/Programm} \cdot \text{Miss/Instruktion} \cdot \text{Miss_Strafe/Taktperiode}$$

Beispielberechnungen

► **Gegeben:**

- Für ein bestimmtes Programm sind 33% aller Instruktionen Speicherinstruktionen, Miss-Rate bei Instruktionen 5%, Miss-Rate bei Daten 10%, $CPI_{\text{perfekt}} = 4$, Miss_Strafe/Taktperiode 12 Zyklen.

► **Frage:**

- Um wieviel schneller wäre ein Rechner mit perfektem Cache?

► **Antwort:**

$\text{Instruktions_Wartezyklen} = \text{Instruktionen} \cdot 5\% \cdot 12 = 0.6 \cdot \text{Instruktionen}$

$\text{Daten_Wartezyklen} = \text{Instruktionen} \cdot 33\% \cdot 10\% \cdot 12 = 0.4 \cdot \text{Instruktionen}$

$\text{Wartezyklen} = 1.0 \cdot \text{Instruktionen}$

$CPI_{\text{warten}} = (4 + 1) \text{ Zyklen/Instruktion}$

CPU-Zeit mit perfektem Cache ist um Faktor 5/4 besser.

Beispielberechnungen

► **Gegeben** sind die gleichen Randbedingungen wie zuvor.

► **Frage:**

- Wie steigt die Leistung des Rechners bei K-facher Taktrate.
- Annahme: Speicher wird nicht schneller.

► **Antwort:**

$\text{Wartezyklen} = K \cdot 1.0 \cdot \text{Instruktionen} = K \text{ Instruktionen}$

$CPI_{\text{schnell}} = (4 + K) \text{ Zyklen/Instruktion}$

$\text{CPU_Zeit}_{\text{langsam}} / \text{CPU_Zeit}_{\text{schnell}} =$

$(\text{Instruktionen} \cdot CPI_{\text{warten}} \cdot \text{Taktrate}) / (\text{Instruktionen} \cdot CPI_{\text{schnell}} \cdot \text{Taktrate} / K)$

$= K \cdot 5 / (4 + K)$

Beispiel: $K = 6$, Beschleunigungsfaktor = 3

Mehrere Cacheebenen

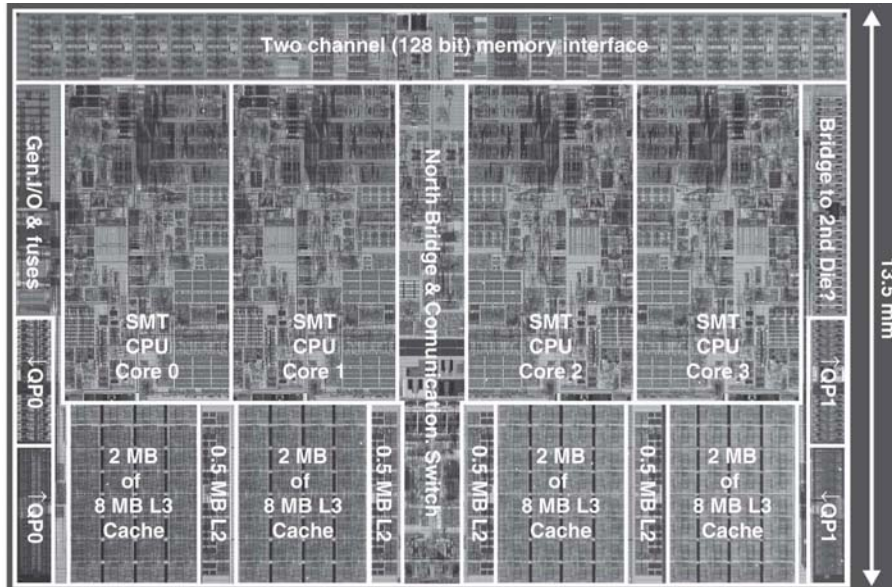
- ▶ **L1 Cache:**
 - klein, schnell, möglichst kleine Hit_Time
- ▶ **L2 Cache:**
 - behandelt Misses vom L1 Cache
 - grösser, langsamer, schneller als Hauptspeicher
 - möglichst kleine Miss_Rate um Hauptspeicherzugriff zu vermeiden
 - Blockgrösse im L2 Cache ist grösser als die im L1 Cache
- ▶ **L3 Cache:**
 - findet sich vor allem in Multicore-Prozessoren

Mehrere Cacheebenen

- ▶ **Berechnungsbeispiel:**
 - nur Instruktions-Cache (in diesem Beispiel)
Takttrate 4 GHz; CPI = 1 (ohne Caches);
L1 Miss_Rate = 2%
Zugriffszeit Hauptspeicher = 100ns
Zugriffszeit L2 Cache = 5ns
L2 Miss_Rate = 10 %
 - **Mit L1 Cache:**
 - $Miss_Strafe/Taktperiode = 100ns/0.25ns = 400$ Zyklen
 - $CPI = 1 + 0.02 \cdot 400 = 9$
 - **Mit L1 und L2 Cache:**
 - L1 Miss mit L2 Hit: $Miss_Strafe/Taktperiode = 5ns / 0.25ns = 20$ Zyklen
 - L1 Miss mit L2 Miss:
 $Miss_Strafe/Taktperiode = 105ns / 0.25ns = 420$ Zyklen
 - $CPI = 1 + 0.02 \cdot 0.9 \cdot 20 + 0.02 \cdot 0.1 \cdot 420 = 2.2$

Speicherhierarchie Beispiel

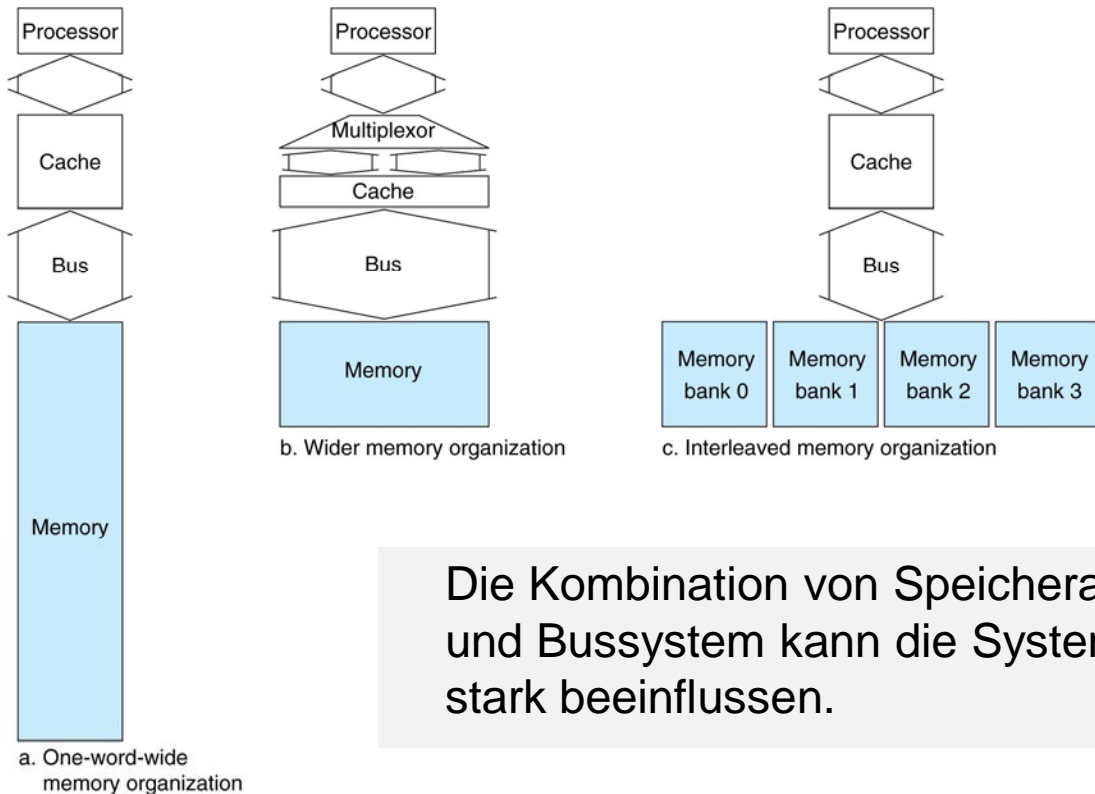
Intel Nehalem Processor Photo: Grösse 13.5 x 19.6 mm, 731 Millionen Transistoren. 4 Prozessorkerne mit jeweils 32 KB L1 Instruktionen- und 32 KB L1 Daten-Cache sowie einem 512 KB L2 Cache. Die 4 Prozessorkerne teilen sich einen gemeinsamen 8 MB L3 Cache.



Speicherhierarchie Beispiel

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

Speicher und Bus



Die Kombination von Speicherarchitektur und Bussystem kann die Systemleistung stark beeinflussen.

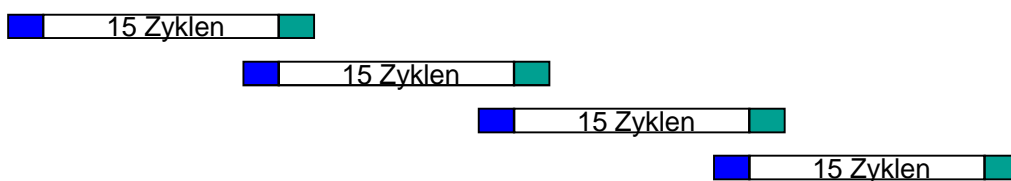
Speicher und Bus

► Annahmen:

- 1 Buszyklus zum Übertragen der Adresse
- 15 Buszyklen für jeden Hauptspeicherzugriff
- 1 Buszyklus pro Datentransfer

► Beispiel 1 (Organisation a.):

- Blockgrösse 4 Worte, Speicher- und Busbreite 1 Wort
- $Miss_Strafe = (1 + 4 \cdot 15 + 1) \text{ Buszyklen} = 62 \text{ Buszyklen}$



- $Bandbreite = (4 \cdot 4) / 62 \text{ Bytes/Zyklus} = 0.258 \text{ Bytes/Zyklus}$

Speicher und Bus

► Beispiel 2 (Organisation b.):

- Blockgrösse 4 Worte, Speicher- und Bus-breite 4 Worte
- $Miss_Strafe = (1 + 15 + 1) \text{ Buszyklen} = 17 \text{ Buszyklen}$
- $Bandbreite = (4 \cdot 4) / 17 \text{ Bytes/Zyklus} = 0.94 \text{ Bytes/Zyklus}$

► Beispiel 3 (Organisation c.):

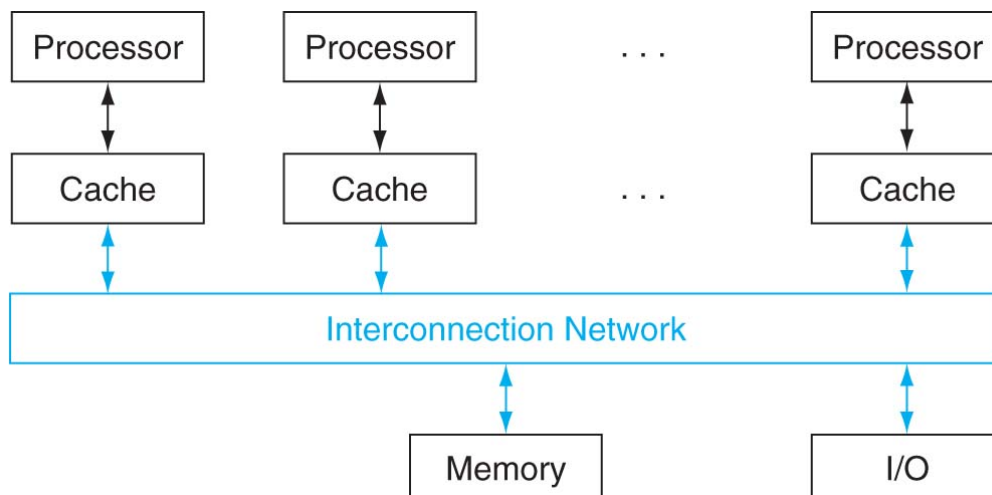
- Blockgrösse 4 Worte, 4 Speicherbänke mit einer Breite von je einem Wort, Busbreite 1 Wort
- $Miss_Strafe = (1 + 15 + 4) \text{ Buszyklen} = 20 \text{ Buszyklen}$
- $Bandbreite = (4 \cdot 4) / 20 \text{ Bytes/Zyklus} = 0.8 \text{ Bytes/Zyklus}$



Cache Kohärenz

► Inkohärente Daten in Caches und Hauptspeicher:

- Problem bei Multiprozessorsystemen mit gemeinsamem Speicher, z.B. Multicoresystemen.

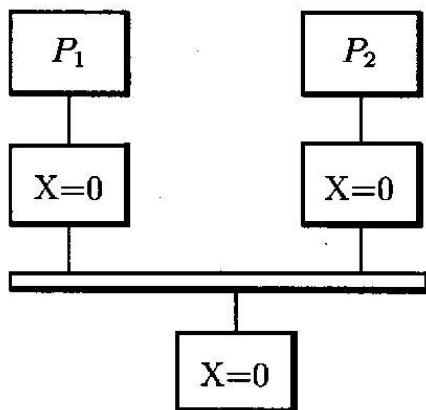


Cache Kohärenz

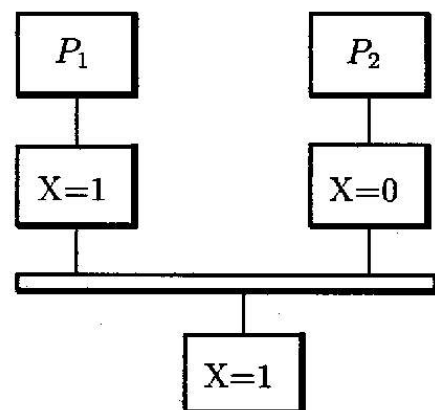
- ▶ Falls ein Prozessor P den Inhalt einer Speicherzelle X liest, nachdem er sie mit dem Wert w beschrieben hat, dann ist der gelesene Wert w , sofern die Speicherzelle in der Zwischenzeit nicht durch einen anderen Prozessor beschrieben wurde.
- ▶ Falls ein Prozessor P den Inhalt einer Speicherzelle X zur Zeit t_1 liest, nachdem ein anderer Prozessor die Speicherzelle mit dem Wert w zur Zeit t_0 beschrieben hat, dann ist der gelesene Wert w , sofern $t_1 - t_0$ genügend gross ist, und die Speicherzelle in der Zeit zwischen t_0 und t_1 nicht neu beschrieben wurde.
- ▶ Alle Prozessoren beobachten die gleiche Reihenfolge zweier Schreibbefehle auf die gleiche Speicherposition.

Cache Kohärenz

Beispiel 1:



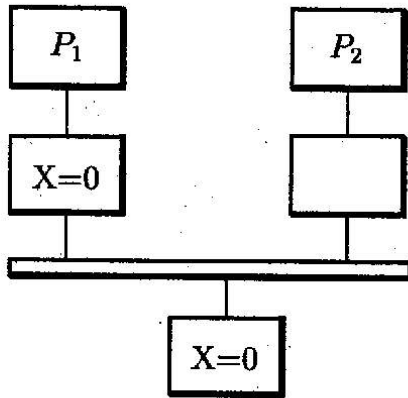
Die Variable X befindet sich in den Caches von P_1 und P_2 sowie im Hauptspeicher.



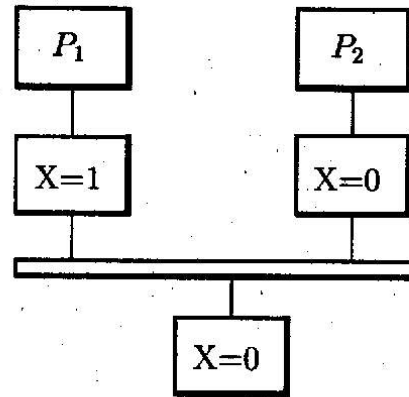
P_1 schreibt $X=1$.
Der Hauptspeicher wird beim write through cache ebenfalls verändert. P_2 liest den alten (inkohärenten) Wert aus dem eigenen Cache.

Cache Kohärenz

Beispiel 2:



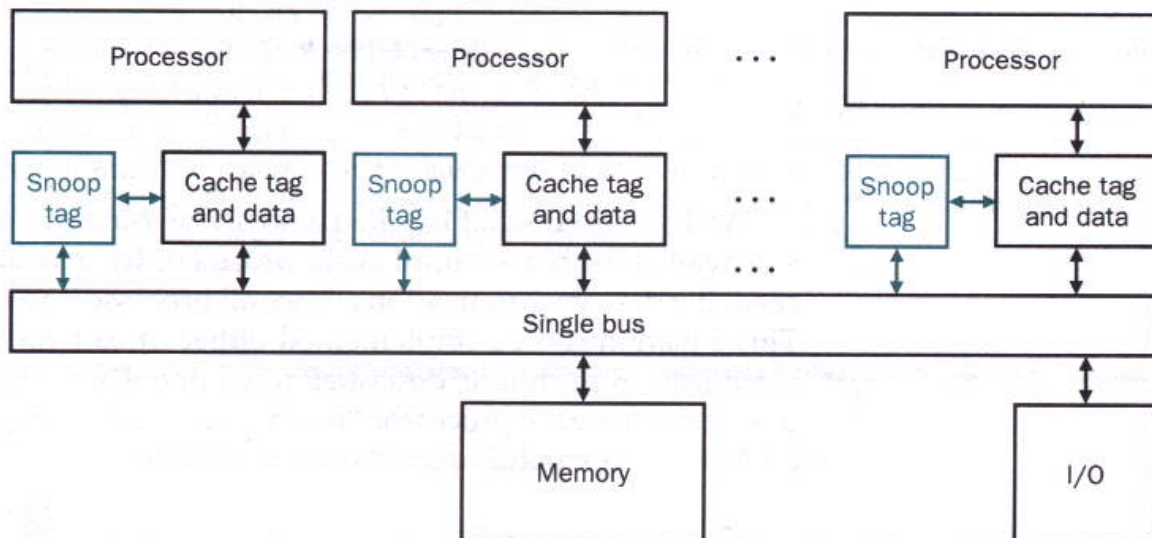
Die Variable X befindet sich im Cache von P_1 und im Hauptspeicher.



P_1 schreibt $X=1$. Der Hauptspeicher wird nicht sofort verändert (write back cache). P_2 liest nun den alten inkohärenten Wert aus dem Hauptspeicher.

Cache Kohärenz: Snoopy Protokolle

- Alle Prozessoren beobachten Datenübertragungen zwischen jedem Cache und dem gemeinsamen Speicher.



Cache Kohärenz: Snoopy Protokolle

- ▶ Erweiterung der Status-Bits jeder Cachezeile.
- ▶ Zusätzliche Cache-Controller, die das jeweilige Cachekohärenz-Protokoll implementieren.
- ▶ Zur Vermeidung von Zugriffskonflikten zwischen CPU und werden Adressen-Tags und Status-Bits dupliziert (Snoop tag).
- ▶ Protokolle werden üblicherweise durch endliche Automaten dargestellt. Die Zustände sind den Cachezeilen zugeordnet und repräsentieren die aktuelle Situation.
- ▶ Protokollbeispiele:
 - *Write invalidate für Write through (durchgängiges Schreiben)*
 - Write invalidate für Write back
 - MESI (Modified, Exclusive, Shared Invalid)

Cache Kohärenz: Write invalidate/Write through

- ▶ Das Zustandsdiagramm ist auf eine Cachezeile von Prozessor *P* bezogen.

