Technische Grundlagen der Informatik 2

Rechnerorganisation

Kapitel 7: Die Speicherhierarchie
Teil A: Caches

Prof. Dr. Ben Juurlink

Fachgebiet: Architektur eingebetteter Systeme
Institut für Technische Informatik und Mikroelektronik
Fak. IV – Elektrotechnik und Informatik

SS 2014

Ziele

Nach dieser Vorlesung sind Sie in der Lage sein:

- Folgende Begriffe zu erklären: Cache, Block/Zeile, temporale und räumliche Lokalität, Treffer, Fehlzugriff, *Tag*, Fehlzugriffsaufwand, Durch-/Rückschreibe-technik, Gültigkeits- und *dirty* Bit, satz-assoziativer Cache, ...
- Gegeben Beschreibung eines Caches, Sachen wie Größe des Caches, der Index, des Tags, ... zu berechnen
- Gegeben Beschreibung eines Caches und Reihe Speicherzugriffe, für jeden Zugriff anzugeben ob Treffer oder Fehlzugriff
- CPU-Zeit zu berechnen gegeben Fehlzugriffsrate und –aufwand
- Durchschnittliche Zugriffszeit zu berechnen und optimale Blockgröße
- Zu berechnen, wie viele Bit man braucht um einen Cache zu implementieren
- CPU-Zeit zu berechnen gegeben eine Cache-Speicherhierarchie

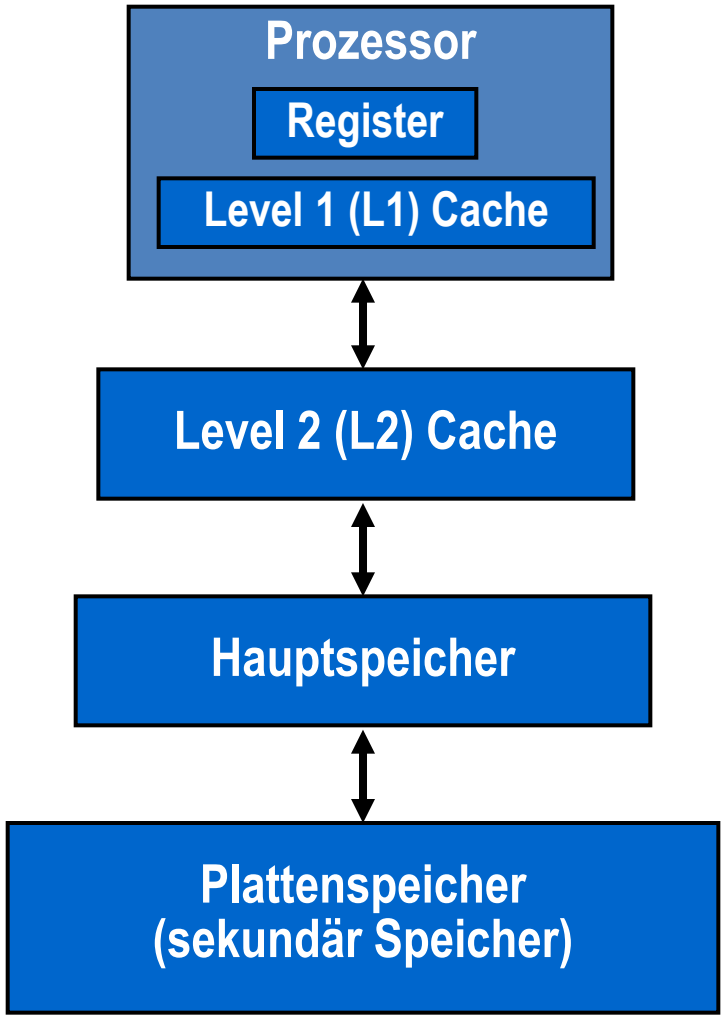
Warum Speicherhierarchie?

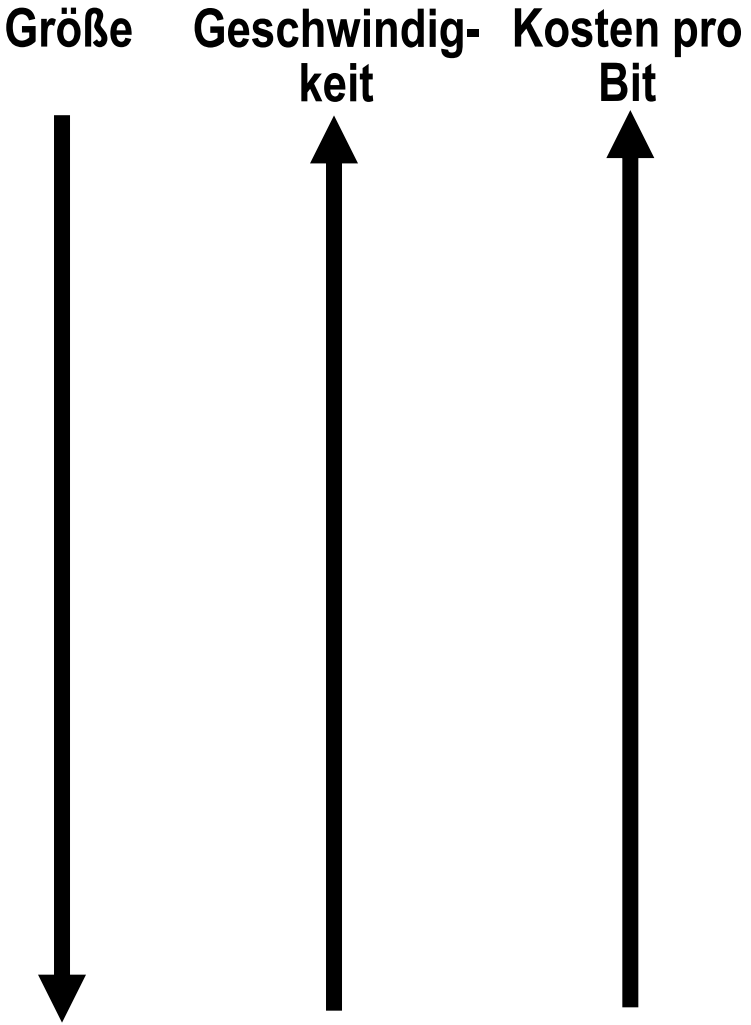
- Benutzer wollen unbegrenzt großen und unendlich schnellen Speicher
  - schnellere Speicher sind teurer und brauchen mehr Platz auf dem Chip

Technologie	Zugriffszeit	$ pro GB (2008)
SRAM	0.5-2.5 ns (1-5 cc)	$2k-$5k
DRAM	50-70 ns (50-150 cc)	$20-$75
Festplatte	5M-20M ns	$0.20-$2

- Aufbau einer Speicherhierarchie:
  - mehrere Speicherebenen mit verschiedenen Geschwindigkeiten und Größen
  - schnellere Speicher sollten häufig genutzte Daten enthalten

Aufbau einer Speicherhierarchie



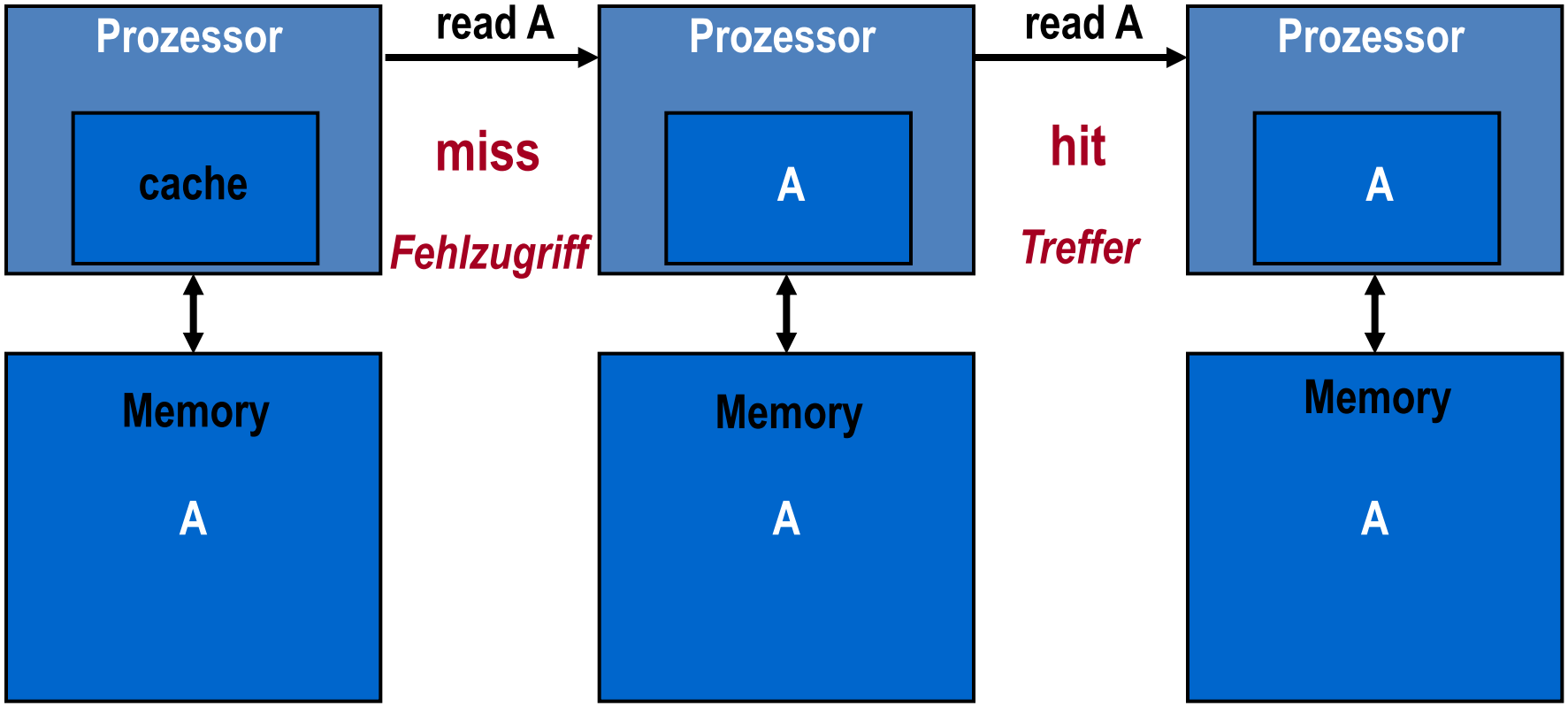


Cache ($)

- Cache ($) = Hochgeschwindigkeitsspeicher zwischen Hauptspeicher und Prozessor (CPU)
- Hardware-kontrolliert
  - Hardware bestimmt welcher Code / Daten im Cache gespeichert werden
  - Auf Abruf (*on demand*): wenn auf Code/Daten zugegriffen wird, die sich nicht im Cache befinden, werden sie in den Cache abgelegt

- Software kontrollierte Hochgeschwindigkeitsspeicher werden oft als Notizblockspeicher (*scratchpad memory*) bezeichnet
  - IBM/Sony/Toshiba Cell Prozessor: Lokaler Speicher (LS)

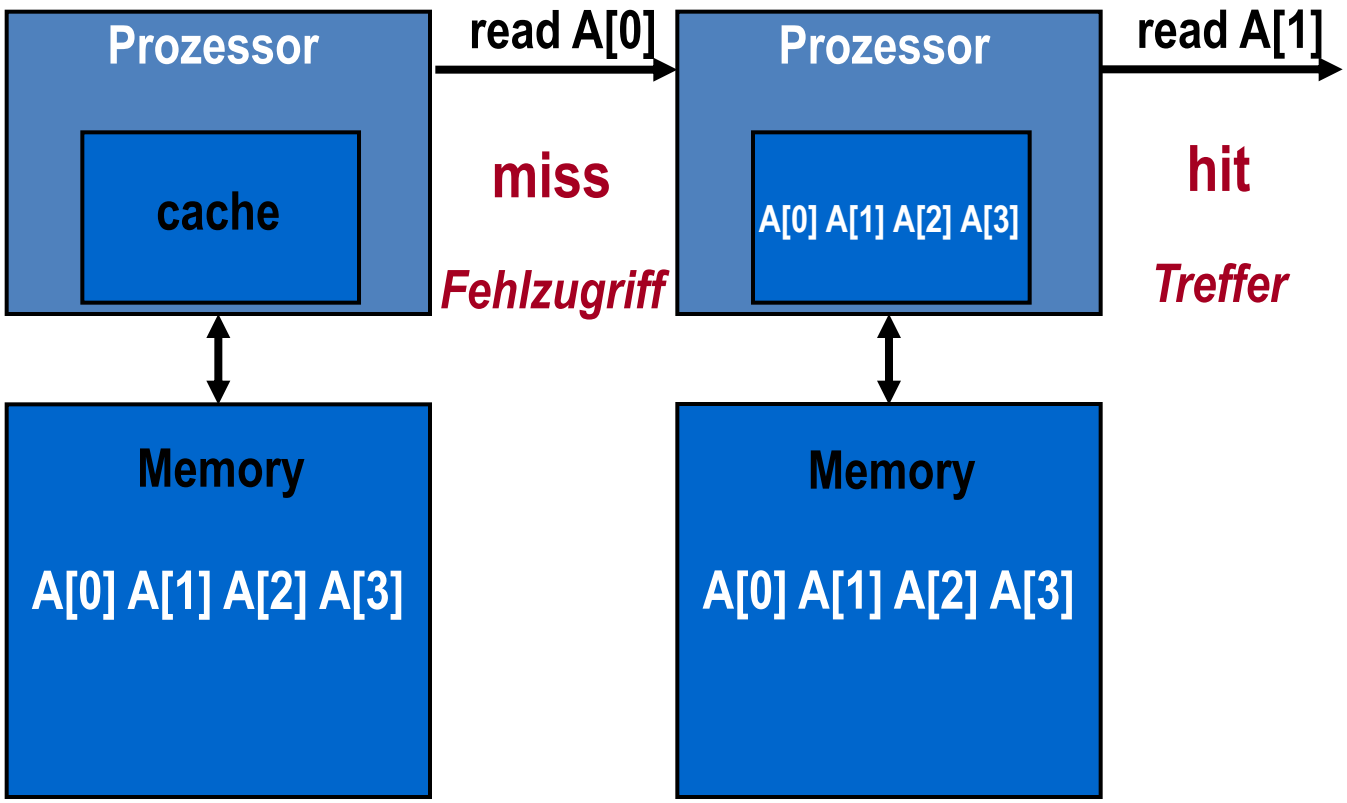
Treffer und Fehlzugriffe (hit and miss)



Lokalitätsprinzip

- Typische Computerprogramme:
  - Instruktionen werden der Reihe nach ausgeführt, nur unterbrochen durch Sprünge hauptsächlich in Schleifen oder als Prozeduraufruf
  - Dadurch wiederholtes Holen kürzlich geholter Instruktionen
  - Dadurch auch wiederholter Zugriff kürzlich zugegriffener Befehle.
- Zeitliche (temporale) Lokalität (*temporal locality*)
  - Wenn auf ein Element zugegriffen wird, erfolgt bald wieder Zugriff darauf
- Räumliche Lokalität (*spatial locality*)
  - Wenn auf ein Element zugegriffen wird, erfolgt bald Zugriff auf in der Nähe befindliche Elemente
- Prinzip gilt für Daten und Instruktionen gleichermaßen.

```
for (i=0; i<n; i++)
    sum += a[i];
```

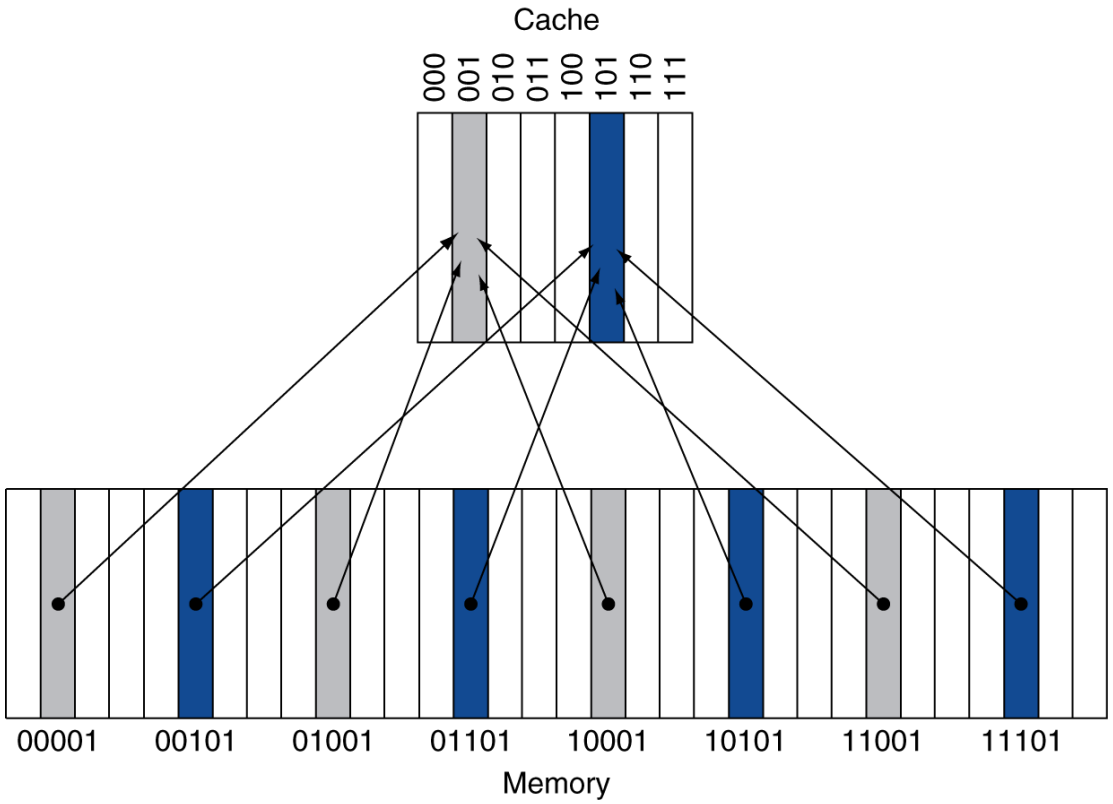


- Effizienter ein Block oder Zeile von *n* Worten zu übertragen als *n* x 1 Wort.

- Block oder Zeile (*line*): Informationseinheit, die zwischen 2 Ebenen transportiert wird
- Treffer (*hit*): angeforderte Daten werden auf der oberen Ebene gefunden
- Fehlzugriff (*miss*): angeforderte Daten nicht gefunden
- Trefferrate (*hit rate*): # Treffer / # Zugriffe
- Fehlzugriffsrate (*miss rate*): # Fehlzugriffe / # Zugriffe = 1 – Trefferrate
- Zugriffszeit bei Treffer (*hit time*): Zeit für einen Treffer (inkl. feststellen ob Treffer)
- Fehlzugriffsaufwand (*miss penalty*): Zeit benötigt um Block von unteren Ebene in höhere Ebene zu laden.
- Durchschnittliche Zugriffszeit (*average memory access time*)
  - AMAT = *hit time* + *miss rate* x *miss penalty*

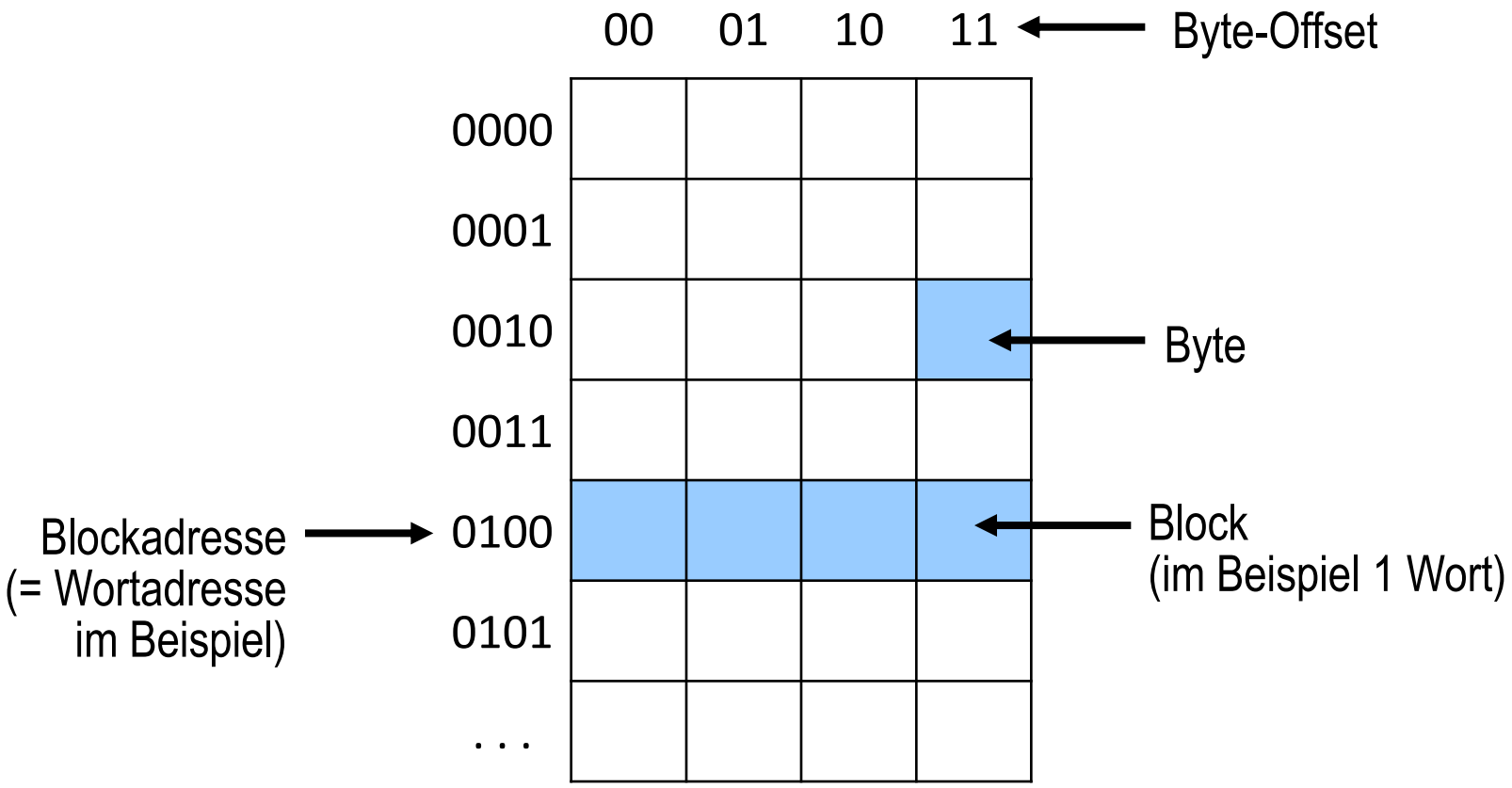
Finden von Daten im Cache / 1

- Einfachste Methode: Cache-Position abhängig von der Adresse des Blocks im Speicher
  - Direkt abgebildeter Cache (*direct-mapped cache*): jede Speicheradresse wird auf genau eine Cache-Position abgebildet



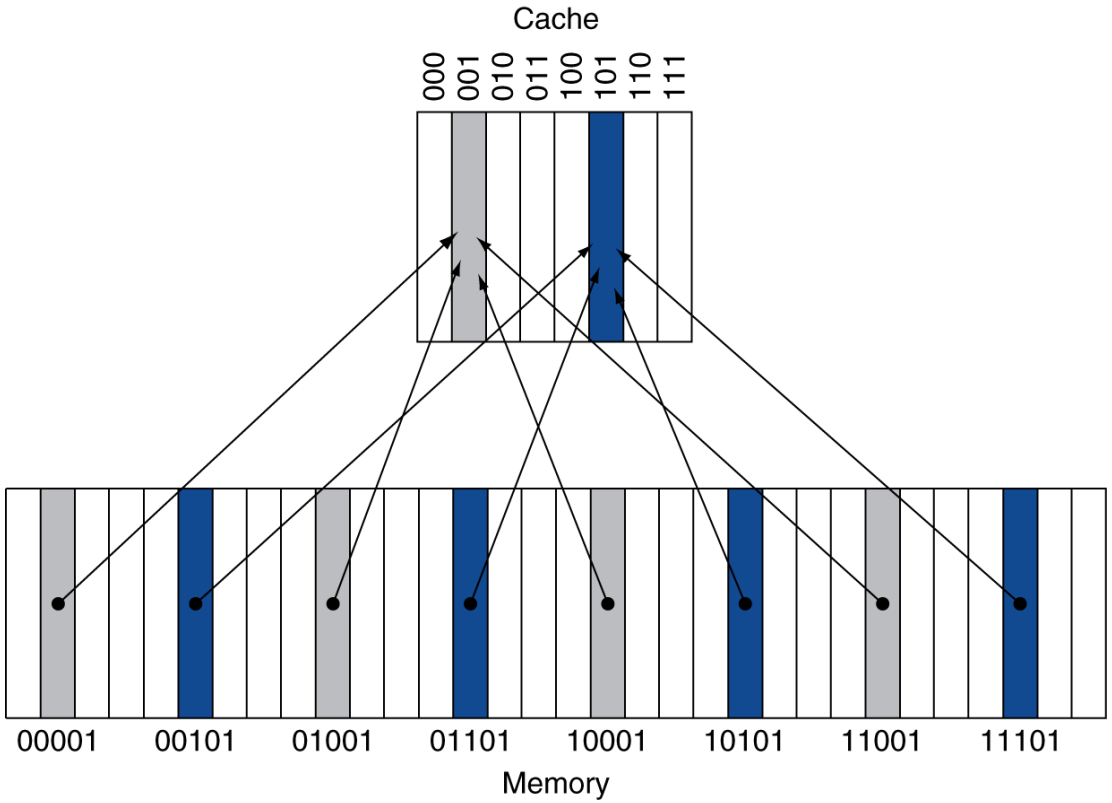
Finden von Daten im Cache / 2

- Blockadresse = (Byteadresse) / (Blockgröße in Bytes) [/=Integer Division]
- Beispiel (für 4-Byte (= 1 Wort) Blöcke):

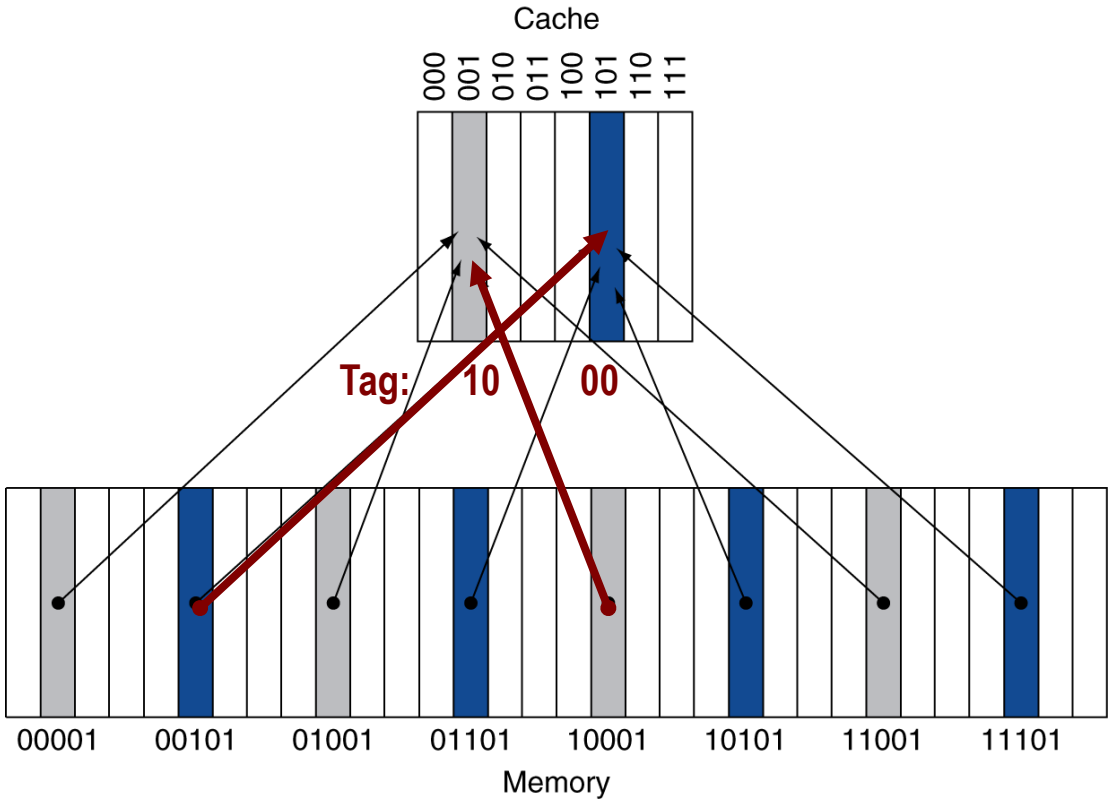


Finden van Daten im Cache / 3

- Direkt abgebildete Caches verwenden Abbildung
  - Cache-Position = (Blockadresse) modulo (# Blöcke im Cache)
- Wenn *n*=(# Blöcke im Cache) Zweierpotenz
  - Cache-Position $\log_2 n$ unteren Bits der Blockadresse



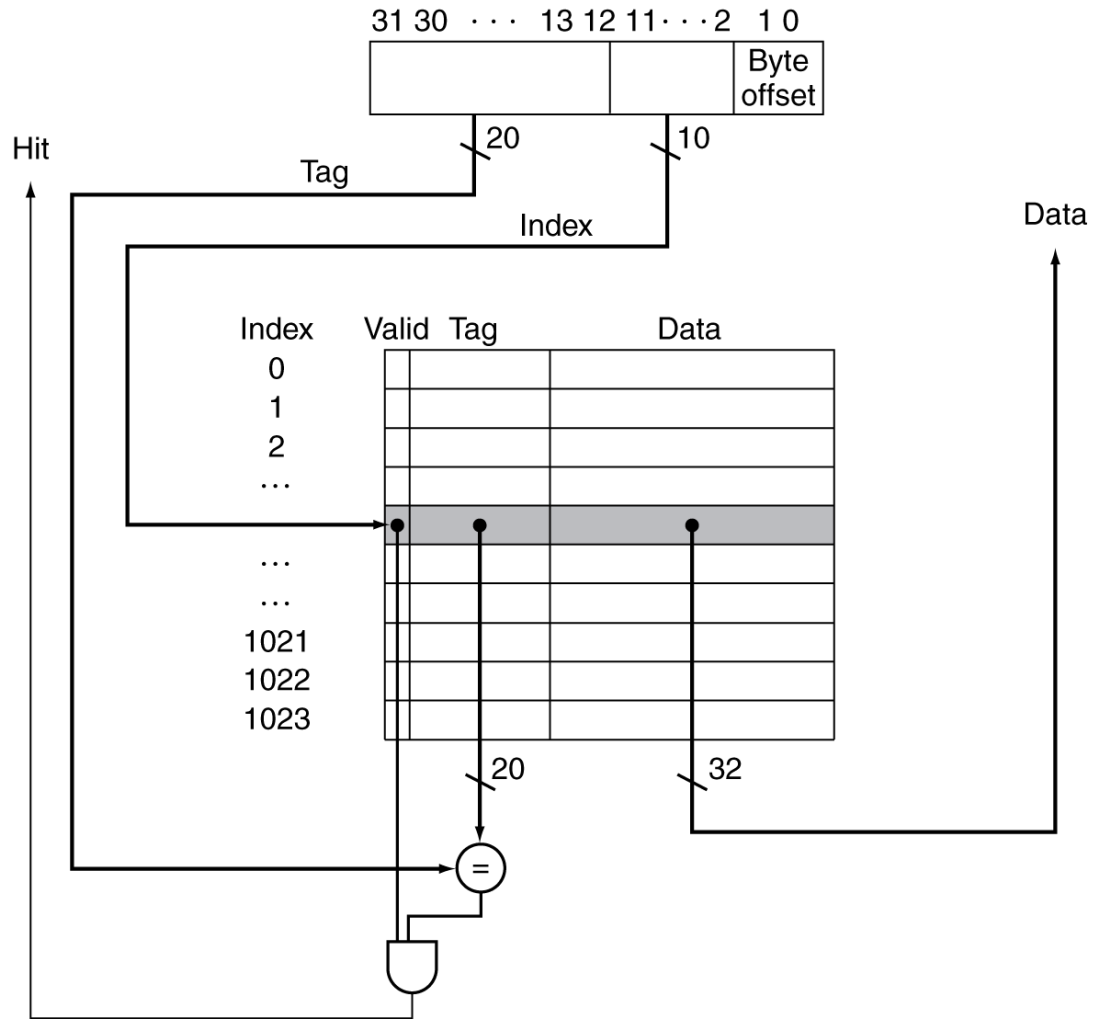
- Jede Cache-Position kann mehrere Speicherblöcke enthalten
- Wie wissen wir, ob Daten im Cache angeforderte Daten entsprechen?
  - **Tag**: Adressinformation nötig um zu erkennen ob Daten im Cache angeforderte Daten entsprechen
  - Obere Teil der Adresse, der nicht als Index für den Cache verwendet wird



Finden von Daten im Cache / 5

- Müssen erkennen können, ob Cache-Block gültige Information enthält
  - Z. B. beim Programmstart
- Jeder Cache-Block hat Gültigkeits-Bit (*valid bit*)
  - gibt an, ob Block gültige Daten enthält

Direkt abgebildeter Cache



- Blockgröße = 2^2 = 4 Bytes
- # Cache-Positionen = # Blöcke = 2^{10} = 1024
- Cachegröße = 1024 x 4 = 4096 = 4KB (1K = 1024)
- Tag-Größe = 32 – 10 – 2 = 20 Bit

Beispiel / 1

- Direkt abgebildeter Cache, anfänglich leer
- 8 Wörter groß, Blockgröße = 1 Wort
- Gib für nächste Folge von Speicherzugriffen an
  - ob Treffer oder Fehlzugriff
  - Cache-Inhalt nach jeder Zugriff

Byteadresse	Blockadresse	Blockadresse Binär	Cache-position
88	88/4 = 22	10**110**	22 mod 8 = 6
104	104/4 = 26	11**010**	26 mod 8 = 2
88	88/4 = 22	10**110**	22 mod 8 = 6
104	104/4 = 26	11**010**	26 mod 8 = 2
64	64/4 = 16	10**000**	16 mod 8 = 0
12	12/4 = 3	00**011**	3 mod 8 = 3
64	64/4 = 16	10**000**	16 mod 8 = 0
72	74/4 = 18	10**010**	18 mod 8 = 2

Beispiel / 2

- Anfangszustand:

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

- Nächste Zugriffe:

Byteadr.	Blockadr.	Binär	Cache-position	T/F
88	88/4 = 22	10**110**	22 mod 8 = 6	F
104	104/4 = 26	11**010**	26 mod 8 = 2	F

Beispiel / 3

- Inhalt:

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

- Nächste Zugriffe:

Byteadr.	Blockadr.	Binär	Cache-position	T/F
88	88/4 = 22	10**110**	22 mod 8 = 6	T
104	104/4 = 26	11**010**	26 mod 8 = 2	T

Beispiel / 4

- Inhalt:

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

- Nächste Zugriffe:

Byteadr.	Blockadr.	Binär	Cache-position	T/F
64	64/4 = 16	10**000**	16 mod 8 = 0	F
12	12/4 = 3	00**011**	3 mod 8 = 3	F

Beispiel / 5

- Inhalt:

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

- Nächste Zugriffe:

Byteadr.	Blockadr.	Binär	Cache-position	T/F
64	64/4 = 16	10**000**	16 mod 8 = 0	T
72	74/4 = 18	10**010**	18 mod 8 = 2	F

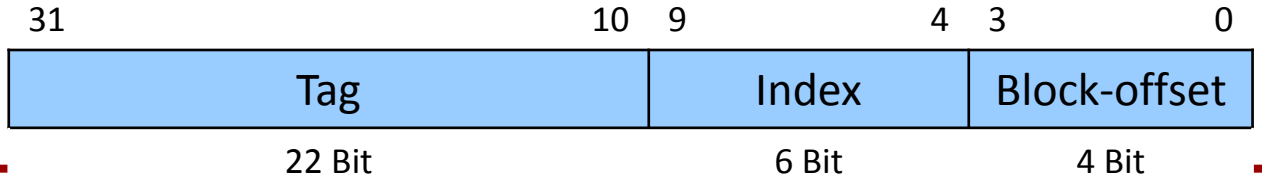
Beispiel / 6

- Inhalt:

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

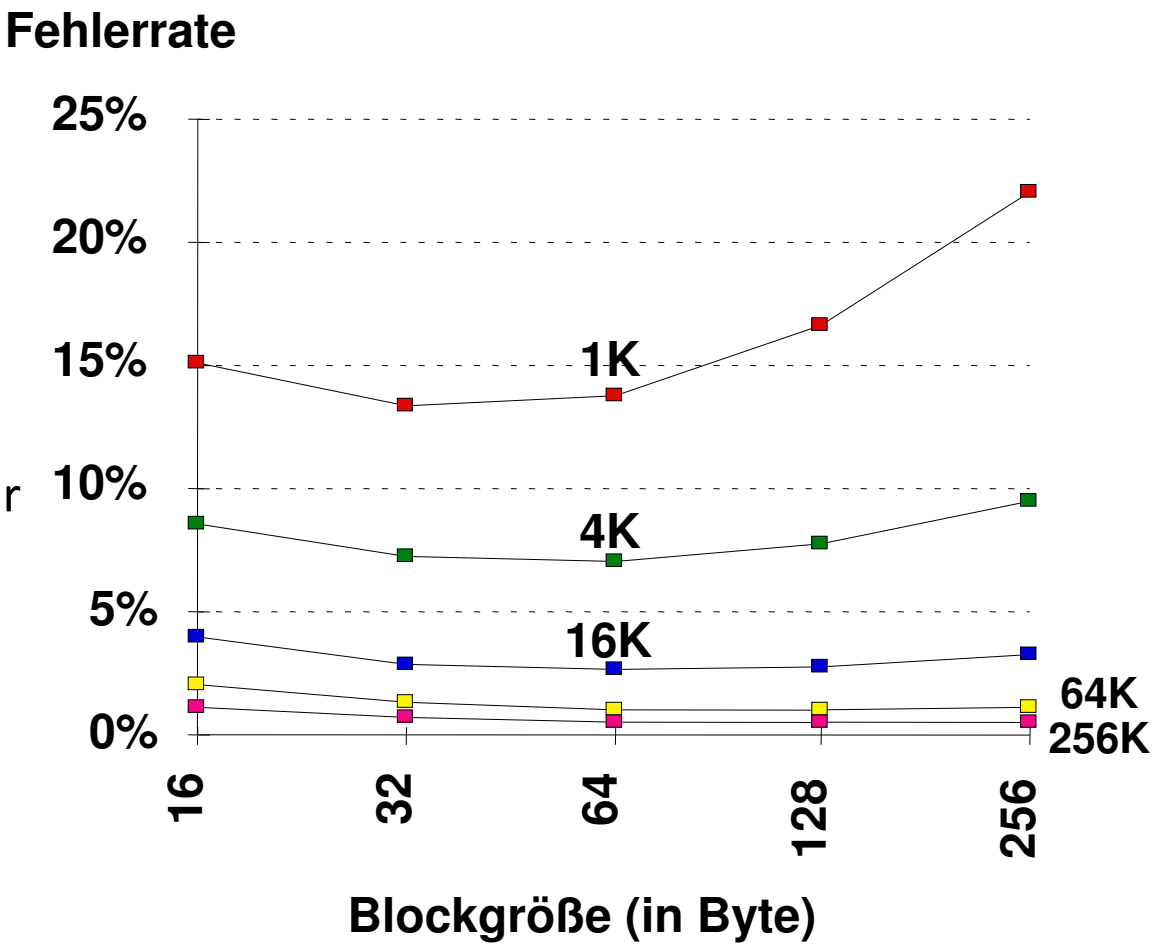
Größere Blocke

- Blöcke von 1 Wort nutzen räumliche Lokalität **nicht** aus
- Größerer Blöcke nutzen räumliche Lokalität um Fehlzugriffsrate zu senken
  - Typische Blockgröße: 32-64 Bytes
  - Block von *n* Wörtern auf einmal laden kostet weniger Zeit als *n* x 1 Wort
- Formel ändern sich nicht:
  - Blockadresse = (Byteadresse) / (Bytes pro Block)
  - Cache-Index = (Blockadresse) Modulo (#Cache-Blöcke)
- Beispiel: direkt abgebildeter Cache mit 64 16-Byte Blöcken
  - Auf welchen Cache-Index wird Byteadresse 1200 abgebildet?
  - Blockadresse = 1200 / 16 = 75
  - Cache-Index = 75 modulo 64 = 11



Blockgröße – Abwägungen / 1

- Größere Blöcke senken Fehlerrate
  - Aufgrund räumliche Lokalität
- Aber wenn zu groß im Verhältnis zu Cache-Größe
  - Größere Blöcke → weniger Blöcke → mehr Wettbewerb → erhöhte Fehlerrate
- Größter Nachteil ist, dass Fehlzugriffsaufwand steigt



Blockgröße – Abwägungen / 2

- Größere Blöcke steigern Fehlzugriffsaufwand
- Annahmen
  - hit time = 1 Taktzyklus
  - miss penalty = 10 + (Wörter pro Block) Taktzyklen
  - miss rate:

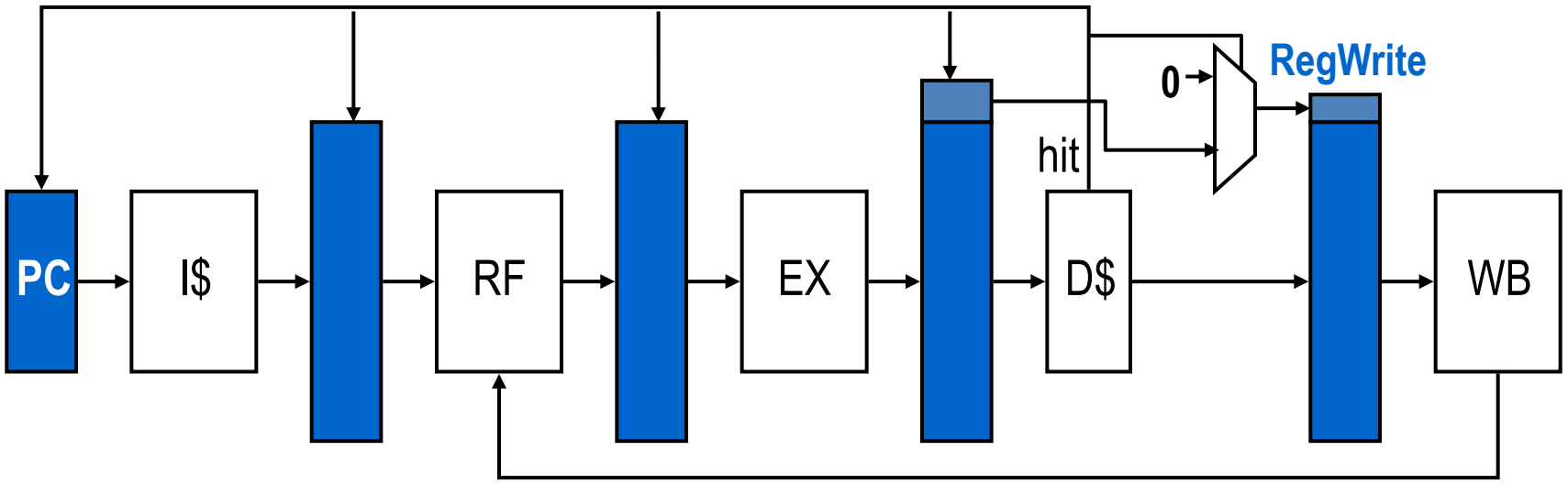
Blockgröße	4B	8B	16B	32B	64B	128B
Fehlzugriffsrate	10%	6%	4%	3%	2.5%	2.3%

- Welche Blockgröße hat geringster
  - AMAT = (hit time) + (miss rate) x (miss penalty)?

Blockgröße	4B	8B	16B	32B	64B	128B
miss penalty	11	12	14	18	26	42
AMAT	2.1	1.72	1.56	**1.54**	1.65	1.97

Verarbeitung von Fehlzugriffen

- Pipeline verzögern, Block aus nächster Hierarchieebene holen
- Befehls-Cache-Fehlzugriff
  - Ursprünglicher PC (aktueller PC-4) an Speicher senden, IF/ID Register löschen
  - Warten bis Speicher Leseoperation abschließt
  - Cache-Block füllen
  - Befehl erneut laden
- Fehlerhafter Datenzugriff → Pipeline verzögern bis Datenzugriff vollendet

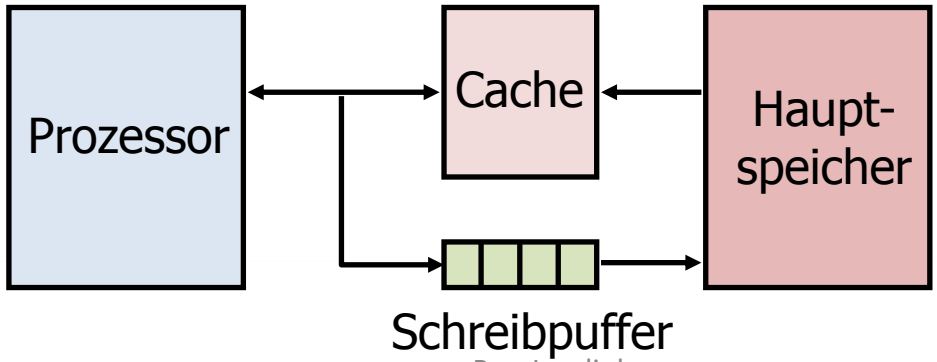


Schreiboperationen

- Schreiben wir neue Daten nur in den Cache oder auch in den Speicher?
- Beide Methoden werden implementiert
  - Durchschreibetechnik (*write-through*)
    - sowohl in den Cache als in den Speicher
    - Vorteil: Cache und Speicher sind konsistent
    - Nachteil: keine gute Leistung
  - Rückschreibetechnik (*write-back*)
    - neue Daten werden nur in den Cache geschrieben
    - in untere Hierarchieebene erst wenn Block ersetzt wird
    - Vorteil: bessere Leistung da mehrere Schreiboperationen zum gleichen Block kombiniert werden
    - Nachteile: Cache und Speicher inkonsistent, Implementierung komplexer

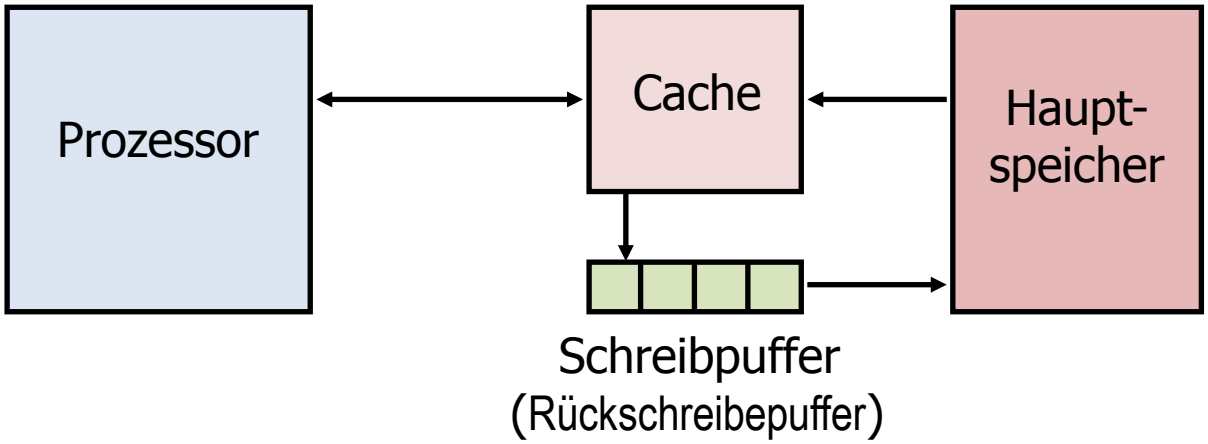
Durchschreibetechnik - Schreibpuffer

- Durchschreibetechnik keine gute Leistung
- Beispiel:
  - CPI ohne Cache-Fehlzugriffe: 1,0
  - Schreiboperation: 100 Taktzyklen
  - 10% der Befehle Schreiboperationen (SPEC2000)
  - CPI = 1,0 + 0,1x100 = 11
- Schreibpuffer (*write buffer*):
  - speichert Daten bis sie in den Speicher geschrieben werden können
  - CPU macht weiter und hält nur an wenn Schreibpuffer voll



Rückschreibetechnik – *dirty bit*

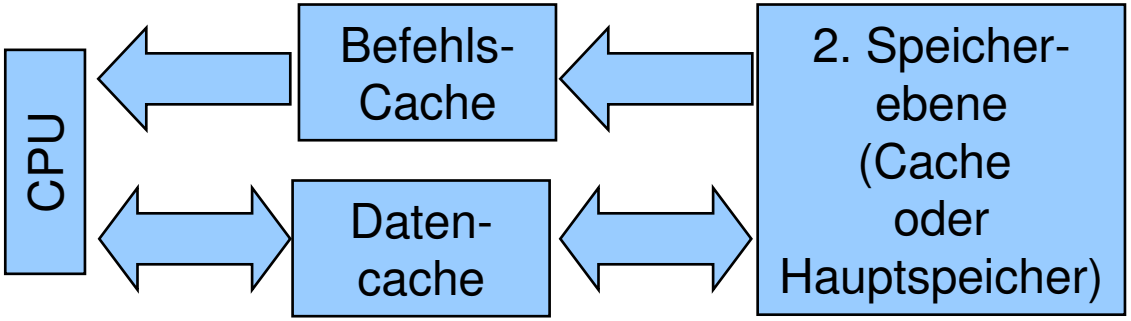
- Brauchen Block nur zurückzuschreiben, wenn er verändert wurde
- *Dirty* Bit: gibt an ob Block geschrieben wurde
- Rückschreibe-Caches haben auch Schreibpuffer für ausgetauschte „dirty-Blöcke“
  - auch Rückschreibepuffer (*write-back buffer*) genannt



- Durchschreibe-Cache kann geschrieben werden während Tag-Vergleich
- Rückschreibe-Cache nicht
  - Speicherpuffer (*store buffer*)

Beispiel: Intrinsity FastMATH / 1

- Eingebetteter MIPS-Prozessor mit 12-Stufige Pipeline
- Kann in jedem Takt ein Befehl und ein Datenwort anfordern
  - getrennte Befehls- und Daten Cache (*split instruction/data cache*)
  - Befehls-Cache könnte größere Blöcke nutzen



Fehlzugriffsrate für Befehle	Fehlzugriffsrate für Daten	Effektive Fehlzugriffsrate
0,4%	11,4%	3,2%

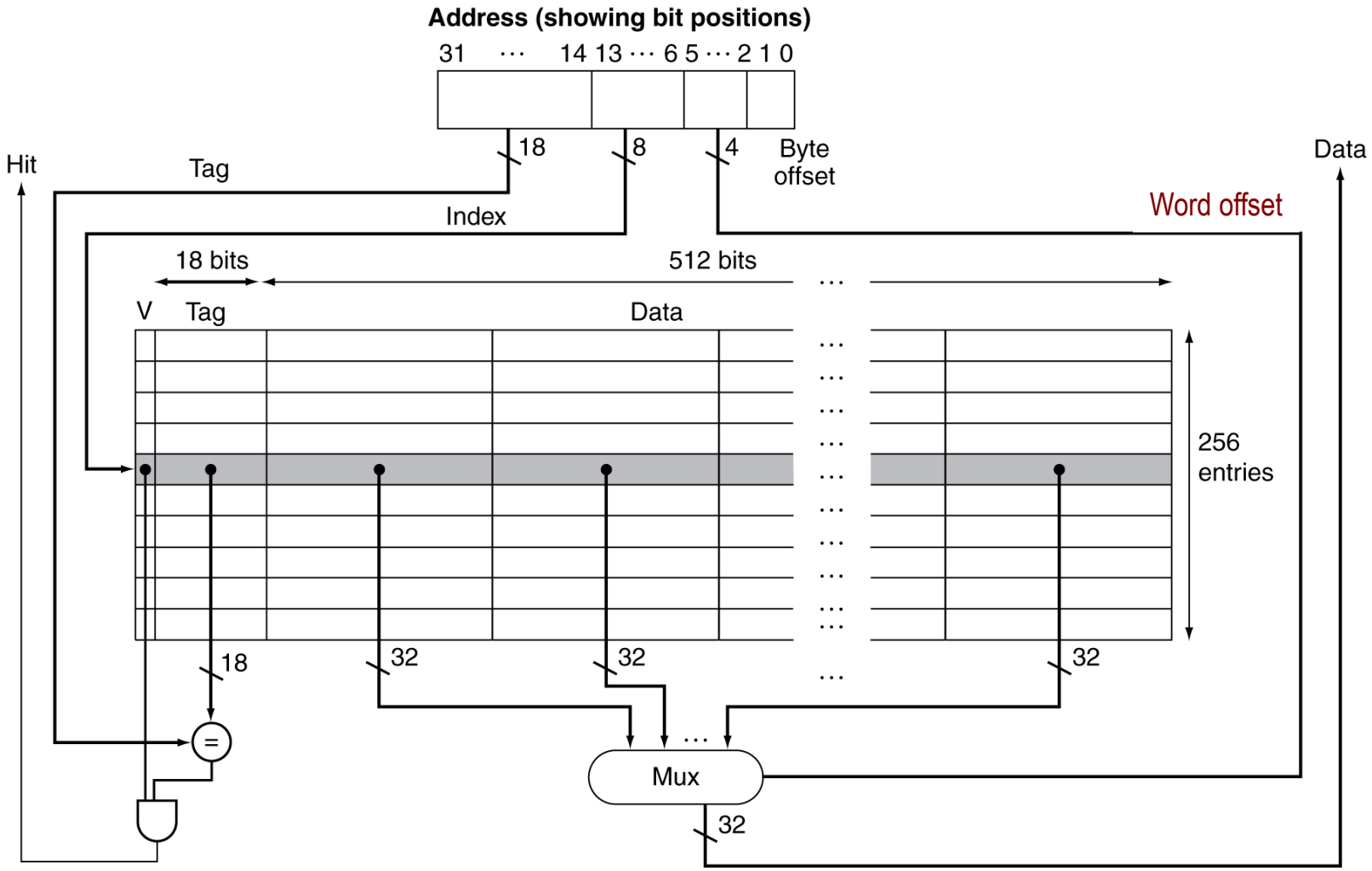
Beispiel: Intrinsity FastMATH / 2

- Sowohl Befehls- als Daten Cache 16 KB groß
- Direkt abgebildet
- Blockgröße 64 Byte (16 Wörter)
- Adresslänge 32 Bit

- Wie groß ist der Index und wie groß der Tag?

- #Blöcke = (Cache-Größe)/Blockgröße = 16KB/64B = 16x1024/64 = 256
  - Index = $\log_2(256) = 8$ Bit
- Tag-Größe = Adresslänge – (Index-Größe) – Block-Offset = 32 – 8 – 6 = 18
  - Block-Offset = $\log_2$(Blockgröße) = $\log_2(64) = 6$

Beispiel: Intrinsity FastMATH / 3



Komponenten einer Adresse

Blockadresse		Block-offset	
Tag	Index	Wort-offset	Byte-offset

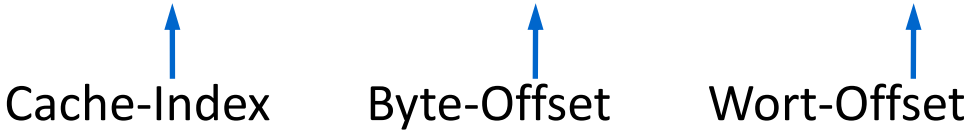
- Byte-offset: letzte 2 Bit, die einen Byte im Wort adressieren
- Wort-offset: nächste $\log_2$(Blockgröße in Wörter) Bit, die ein Wort im Block adressieren
- Block-offset: Wort-offset und Block-offset zusammen ($\log_2$(Blockgröße in Byte) Bit)
- Index: gibt Position im Cache an
- Tag: übrige Bit

- Buch undeutlich über Block-Offset
  - manchmal was wir Wort-Offset nennen (Abb. 7.8)
  - manchmal was wir Block-Offset nennen (Abb. 7.12)

- Wie viele Bits braucht man insgesamt für einen direkt abgebildeten Cache mit 16 KB Daten und Blöcke von 4 Wörtern bei 32-Bit-Adressen?

  - Cache : 16x1024 / (4x4) = 1024 Blöcken
  - Block: 4x32 = 128 Bit Daten (+ Tag und Gültigkeits-Bit)

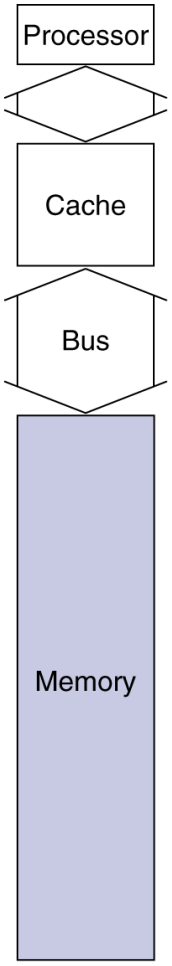
  - Tag: 32 – 10 – 2 – 2 = 18 Bit



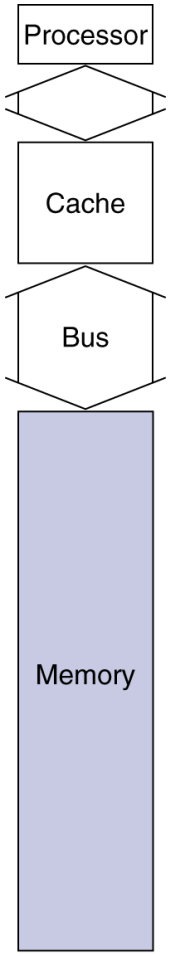
  - Bits insgesamt: 1024 x (128 + 18 + 1) = 147kBit = 18,4 KB

Hauptspeicher

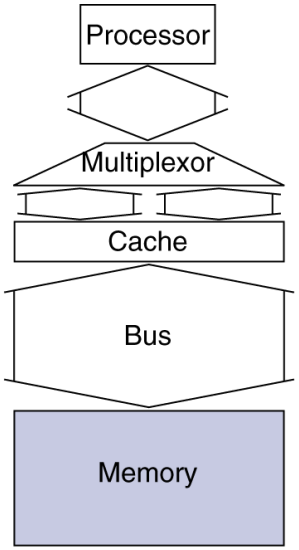
- Hauptspeicher aus DRAM-Bausteinen aufgebaut
- DRAMs auf Dichte und nicht auf Zugriffszeit ausgelegt
- Latenz reduzieren schwierig, Bandbreite zwischen Cache und Speicher erhöhen möglich
- Cache und Speicher sind über einen Bus verbunden
  - Bus-Takt langsamer als CPU-Takt (bis zu 10x)
- Beispiel:
  - 1 Bus-Takt für Senden der Adresse
  - 15 Bus-Takte für jeden DRAM-Zugriff
  - 1 Bus-Takt für Senden eines Datenworts
- Blöcke von 4 Wörtern und 1-Wort breiten DRAM:
  - Fehlzugriffswand= 1 + 4×15 + 4×1 = 65 Bus-Takte
  - Bandbreite = 16 Byte / 65 Bus-Zyklen = 0.25 Byte/Takt



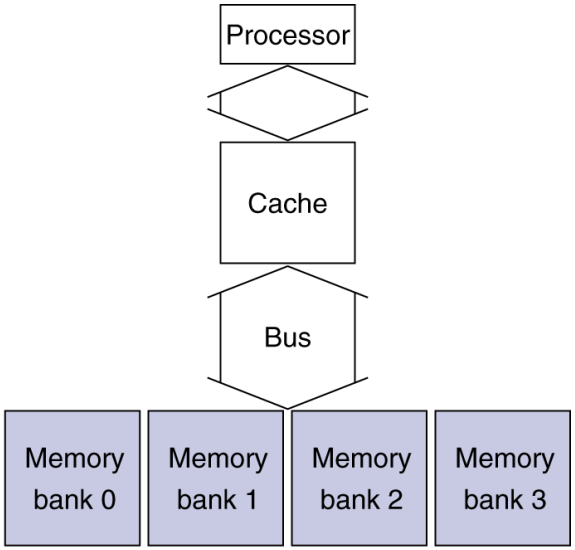
Speicherbandbreite erhöhen



1-Wort breite Speicherorganisation



breite Speicherorganisation



Speicherorganisation mit Verschränkung (*interleaved memory organization*)

- Fehlzugriffsaufwand von
  - 2-Wort-breiter Speicher: 1 + 2x15 + 2x1 = 33 Bus-Takte
  - 4-Wort-breiter Speicher: 1 + 15 + 1 = 17 Bus-Takte
  - Verschränkter Speicher mit 4 Banken: 1 + 1x15 + 4x1 = 20 Bus-Takte

- Kann man eigentlich nur durch Ausführen oder Simulation bestimmen

Einfaches Modell:

- CPU-Zeit = (CPU-Ausführungszyklen + Speicherstillstands-Zyklen) x Taktzeit
  - *CPU time* = (*CPU execution cycles* + *memory stall cycles*) x *cycle time*
- Speicherstillstands-Zyklen = Lesestillstands-Zyklen + Schreibestillstands-Zyklen
- Lesestillstands-Zyklen = #Leseoperationen x Lese-Fehlrate x Lese-Fehlaufwand
- Schreibestillstands-Zyklen = #Schreiboperationen x Schreib-Fehlrate x Schreib-Fehlaufwand + Schreibpuffer-Stillstand-Zyklen
- Annahmen:
  - Schreib-Fehlzugriff erfordert, dass der Block geladen wird
  - Schreibpuffer-Stillstand-Zyklen nur durch detaillierte Simulation zu bestimmen und meist sehr gering, wir nehmen an 0
  - Lese-Fehlaufwand = Schreib-Fehlaufwand
- Speicherstillstands-Zyklen = #Speicherzugriffe x Fehlzugriffsrate x Fehlaufwand
  = #Befehle x Fehlzugriffe/Befehl x Fehlaufwand

- Gegeben:
  - Fehlzugriffsrate Befehlscache: 2%
  - Fehlzugriffsrate Datencache: 4%
  - CPI ohne Speicherstillstände (perfekter Cache): 2
  - Fehleraufwand: 100 Zyklen
  - 36% Lade/Speicher-Befehle (SPECint2000)
- Um wie viel ist der Prozessor langsamer im Vergleich zu einem Prozessor mit perfektem Cache?

Einfluss Cache-Leistung auf CPU-Zeit (Beispiel)

- Gegeben:
  - Fehlzugriffsrate Befehlscache: 2%
  - Fehlzugriffsrate Datencache: 4%
  - CPI ohne Speicherstillstände (perfekter Cache): 2
  - Fehleraufwand: 100 Zyklen
  - 36% Lade/Speicher-Befehle (SPECint2000)
- Um wie viel ist der Prozessor langsamer im Vergleich zu einem Prozessor mit perfektem Cache?
- Lösung (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = $N_{instr} \times 0.02 \times 100 = 2 \times N_{instr}$

Einfluss Cache-Leistung auf CPU-Zeit (Beispiel)

- Gegeben:
  - Fehlzugriffsrate Befehlscache: 2%
  - Fehlzugriffsrate Datencache: 4%
  - CPI ohne Speicherstillstände (perfekter Cache): 2
  - Fehleraufwand: 100 Zyklen
  - 36% Lade/Speicher-Befehle (SPECint2000)
- Um wie viel ist der Prozessor langsamer im Vergleich zu einem Prozessor mit perfektem Cache?
- Lösung (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = $N_{instr} \times 0.02 \times 100 = 2 \times N_{instr}$
  - Datenfehlzugriffs-Zyklen = $N_{instr} \times 0.36 \times 0.04 \times 100 = 1.44 \times N_{instr}$

Einfluss Cache-Leistung auf CPU-Zeit (Beispiel)

- Gegeben:
  - Fehlzugriffsrate Befehlscache: 2%
  - Fehlzugriffsrate Datencache: 4%
  - CPI ohne Speicherstillstände (perfekter Cache): 2
  - Fehleraufwand: 100 Zyklen
  - 36% Lade/Speicher-Befehle (SPECint2000)
- Um wie viel ist der Prozessor langsamer im Vergleich zu einem Prozessor mit perfektem Cache?
- Lösung (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = $N_{instr} \times 0.02 \times 100 = 2 \times N_{instr}$
  - Datenfehlzugriffs-Zyklen = $N_{instr} \times 0.36 \times 0.04 \times 100 = 1.44 \times N_{instr}$
  - CPI mit Speicherstillstand = 2 + 3.44 = 5.44

- Gegeben:
  - Fehlzugriffsrate Befehlscache: 2%
  - Fehlzugriffsrate Datencache: 4%
  - CPI ohne Speicherstillstände (perfekter Cache): 2
  - Fehleraufwand: 100 Zyklen
  - 36% Lade/Speicher-Befehle (SPECint2000)
- Um wie viel ist der Prozessor langsamer im Vergleich zu einem Prozessor mit perfektem Cache?
- Lösung (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = $N_{instr} \times 0.02 \times 100 = 2 \times N_{instr}$
  - Datenfehlzugriffs-Zyklen = $N_{instr} \times 0.36 \times 0.04 \times 100 = 1.44 \times N_{instr}$
  - CPI mit Speicherstillstand = 2 + 3.44 = 5.44
  - Prozessor mit perfektem Cache ist Faktor 5.44/2 = 2.72 schneller.

- Angenommen wir verdoppeln die Taktgeschwindigkeit, aber die Geschwindigkeit des Hauptspeichers bleibt unverändert.
  - Fehleraufwand vergrößert sich auf 200 Zyklen
- Um wie viel schneller ist der Prozessor mit dem schnelleren Takt?
- Antwort (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = $N_{instr} \times 0.02 \times 200 = 4 \times N_{instr}$
  - Datenfehlzugriffs-Zyklen = $N_{instr} \times 0.36 \times 0.04 \times 200 = 2.88 \times N_{instr}$

Einfluss Hauptspeichergeschwindigkeit

- Angenommen wir verdoppeln die Taktgeschwindigkeit, aber die Geschwindigkeit des Hauptspeichers bleibt unverändert.
  - Fehleraufwand vergrößert sich auf 200 Zyklen
- Um wie viel schneller ist der Prozessor mit dem schnelleren Takt?
- Antwort (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = N_{instr} x 0.02 x 200 = 4 x N_{instr}
  - Datenfehlzugriffs-Zyklen = N_{instr} x 0.36 x 0.04 x 200 = 2.88 x N_{instr}
  - CPI (schneller Takt) = 2 + 6.88 = 8.88
  - CPI (langsamer Takt) = 5.44

Einfluss Hauptspeichergeschwindigkeit

- Angenommen wir verdoppeln die Taktgeschwindigkeit, aber die Geschwindigkeit des Hauptspeichers bleibt unverändert.
  - Fehleraufwand vergrößert sich auf 200 Zyklen
- Um wie viel schneller ist der Prozessor mit dem schnelleren Takt?
- Antwort (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = N_{instr} x 0.02 x 200 = 4 x N_{instr}
  - Datenfehlzugriffs-Zyklen = N_{instr} x 0.36 x 0.04 x 200 = 2.88 x N_{instr}
  - CPI (schneller Takt) = 2 + 6.88 = 8.88
  - CPI (langsamer Takt) = 5.44
  - Leistungssteigerung = 5.44 / (8.88/2) = 1.23

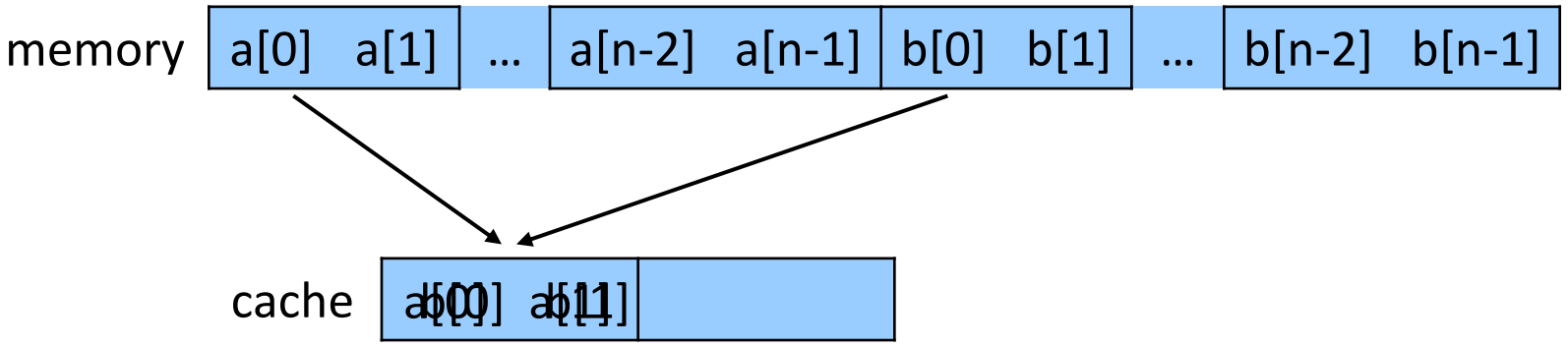
Cache-Leistung verbessern

- AMAT = (hit time) + (miss rate) x (miss penalty)
  - Trefferzeit verbessern
  - Fehlzugriffsrate reduzieren
  - Fehlzugriffsaufwand reduzieren

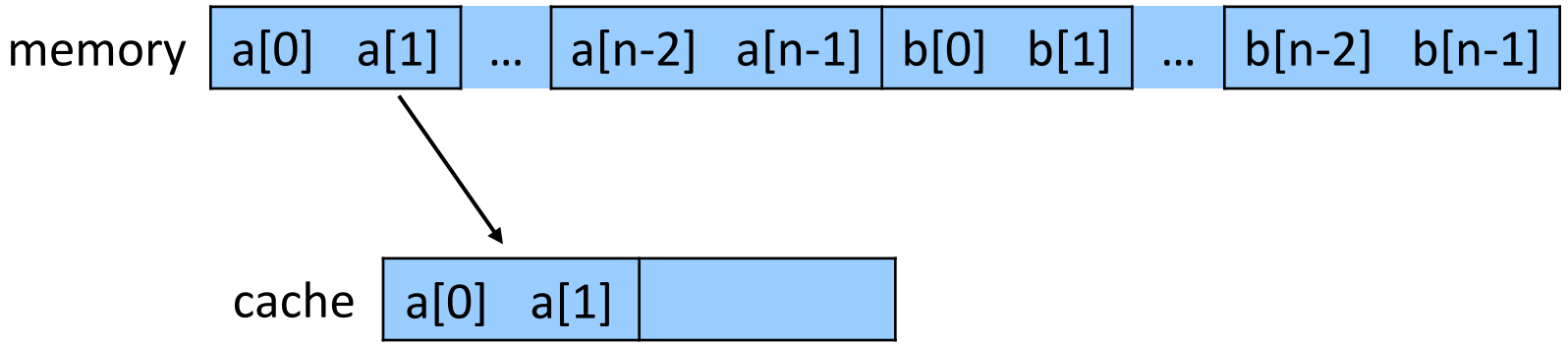
- Was passiert, wenn wir die Blöcke größer machen?

- Wir besprechen 2 Methoden um Cache-Leistung zu verbessern
  - assoziative Caches: reduzieren Fehlzugriffsrate durch flexiblere Platzierung von Blöcken
  - Cache-Speicherhierarchie (*multilevel cache*): reduziert Fehlzugriffsaufwand

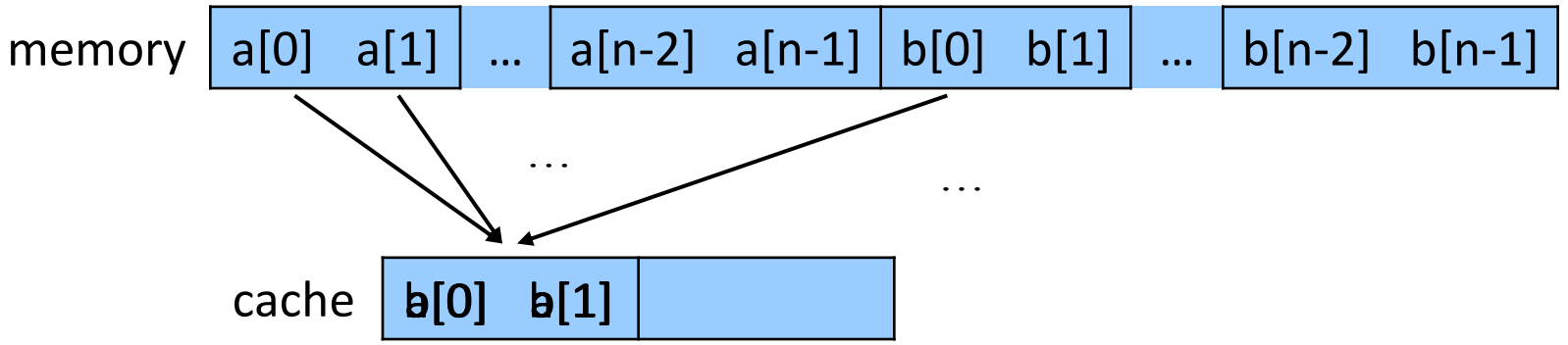
- Beispiel:
```
for (i=0; i<n; i++)
    dotprod += a[i]*b[i];
```
- Angenommen `a[i]` und `b[i]` werden auf gleichen Cache-Block abgebildet
  - Nicht unüblich, wenn `a` und `b` hintereinander im Speicher allokiert wurden und ihre Größe ein Vielfaches der Cachegröße ist
- Jeder Zugriff auf `a` und `b` generiert Fehlzugriff:



- Beispiel:
```
for (i=0; i<n; i++)
    dotprod += a[i]*b[i];
```
- Angenommen `a[i]` und `b[i]` werden auf gleichen Cache-Block abgebildet
  - Nicht unüblich, wenn `a` und `b` hintereinander im Speicher allokiert wurden und ihre Größe ein Vielfaches der Cachegröße ist
- Jeder Zugriff auf `a` und `b` generiert Fehlzugriff



- Beispiel:
```
for (i=0; i<n; i++)
    dotprod += a[i]*b[i];
```
- Angenommen `a[i]` und `b[i]` werden auf gleichen Cache-Block abgebildet
  - Nicht unüblich, wenn `a` und `b` hintereinander im Speicher allokiert wurden und ihre Größe ein Vielfaches der Cachegröße ist
- Jeder Zugriff auf `a` und `b` generiert Fehlzugriff



- Lösung: Flexiblere Platzierung von Blöcken → assoziativer Cache

Assoziative Caches

- vollassoziativer Cache (*fully associative cache*)
  - Block kann an jeder Stelle im Cache platziert werden
  - Alle Einträge müssen parallel durchsucht werden
  - 1 Vergleicher pro Cache-Eintrag → teuer
  - Auch Inhaltsadressierbarer Speicher (*Content Addressable Memory*) genannt
- n-fach satzassoziativer Cache (*n-way set associative cache*)
  - Besteht aus einer Menge von Sätzen (*sets*)
  - die aus jeweils *n* Blöcke bestehen (*n ways*)
  - Blockadresse bestimmt auf welchen Satz Block abgebildet wird
    - (Cache-Index) = (Blockadresse) modulo (#Sätze)
  - Gleichzeitiges Durchsuchen aller Einträge eines Satzes
  - *n* Vergleicher benötigt

Mögliche Organisationen für Cache mit 8 Blöcken

1-fach satzassoziativ (direkt abgebildet)

Satz	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

2-fach satzassoziativ

Satz	Tag	Data	Tag	Data
0				
1				
2				
3				

4-fach satzassoziativ

Satz	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

8-fach satzassoziativ (vollassoziativ)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Vergleich Caches mit verschiedener Assoziativität / 1

- Vergleich dreier 4-Block-Caches
  - Direct mapped, 2-way set associative, fully associative
- Sequenz von Blockadressen: 0, 8, 0, 6, 8
- Wie viele Treffer sind möglich?

Direkt abgebildeter Cache:

Block - adresse	Cache-Index	Hit/miss	Cache-Inhalt nach Zugriff			
			0	1	2	3
0	0 mod 4 = 0	miss	**Mem[0]**			
8	8 mod 4 = 0	miss	**Mem[8]**			
0	0 mod 4 = 0	miss	**Mem[0]**			
6	6 mod 4 = 2	miss	Mem[0]		**Mem[6]**	
8	8 mod 4 = 0	miss	**Mem[8]**		Mem[6]	

Vergleich Caches / 2

2-fach satz-assoziativer Cache:

Block-adresse	Cache-Index	Hit/miss	Cache-Inhalt nach Zugriff			
			Set 0		Set 1	
0	0 mod 2 = 0	miss	**Mem[0]**			
8	8 mod 2 = 0	miss	Mem[0]	**Mem[8]**		
0	0 mod 2 = 0	hit	**Mem[0]**	Mem[8]		
6	6 mod 2 = 0	miss	Mem[0]	**Mem[6]**		
8	8 mod 2 = 0	miss	**Mem[8]**	Mem[6]		

Ersetzungsschema: *least recently used*

voll-assoziativer Cache:

Block - adresse		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]	

Ersetzungsschema

- keine Wahl bei direkt abgebildeter Cache

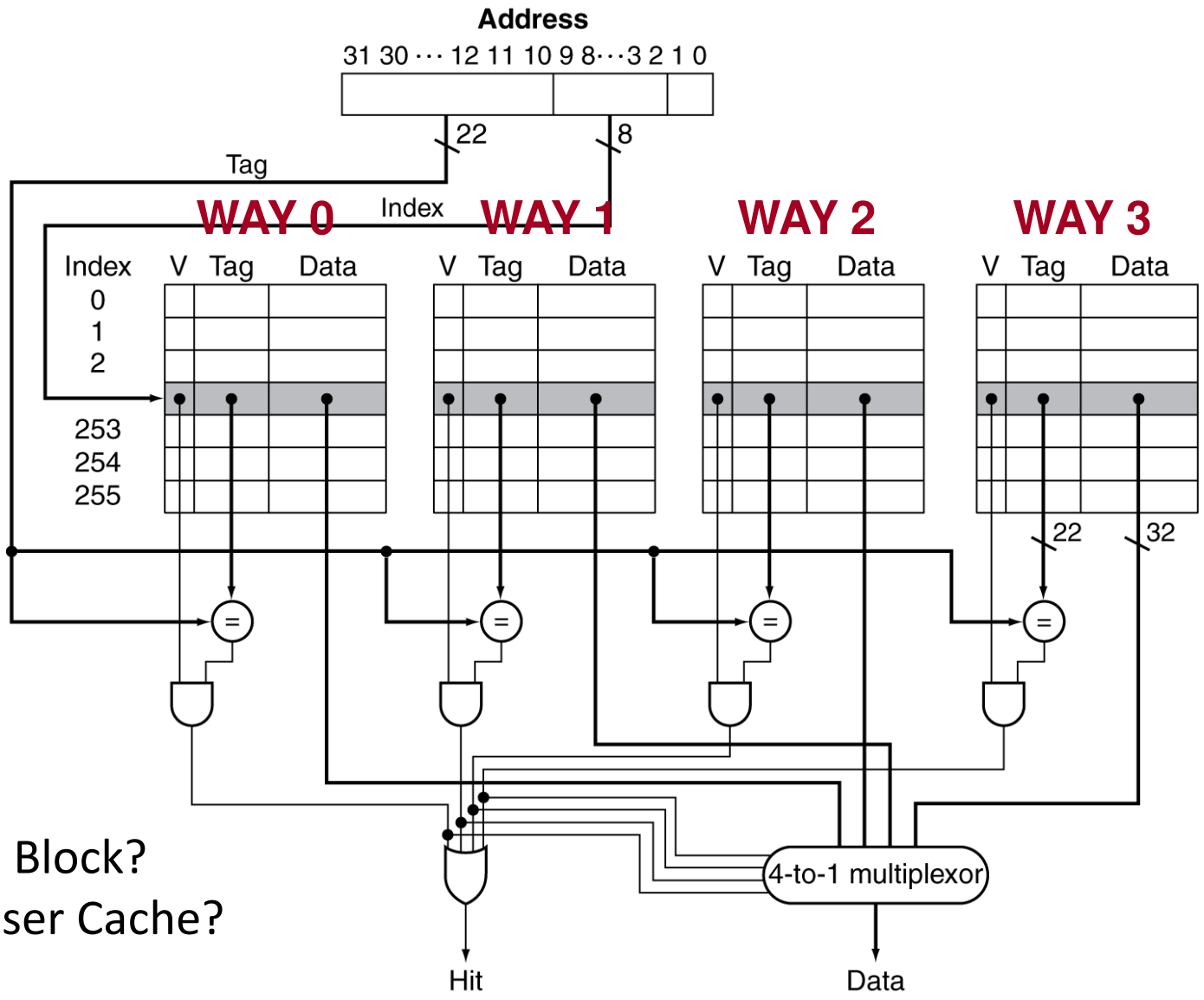
- Random
  - Ersetze einen zufälligen Block im Satz (einfache Hardware)
- First-In-First-Out (FIFO)
  - Ersetze Block, welcher als erster im Cache geladen ist
- Least Recently Used (LRU)
  - Ersetze Block, auf den am längsten nicht mehr zugegriffen wurde
  - Komplizierte Hardware (> 4-way: Approximation)
- Optimaler Algorithmus (OPT oder MIN)
  - Ersetze Block, auf den am längsten nicht mehr zugegriffen wird

Wie assoziativ?

- Erhöhung der Assoziativität reduziert Fehlerrate
  - Aber mit abnehmende Rendite (*diminishing returns*)

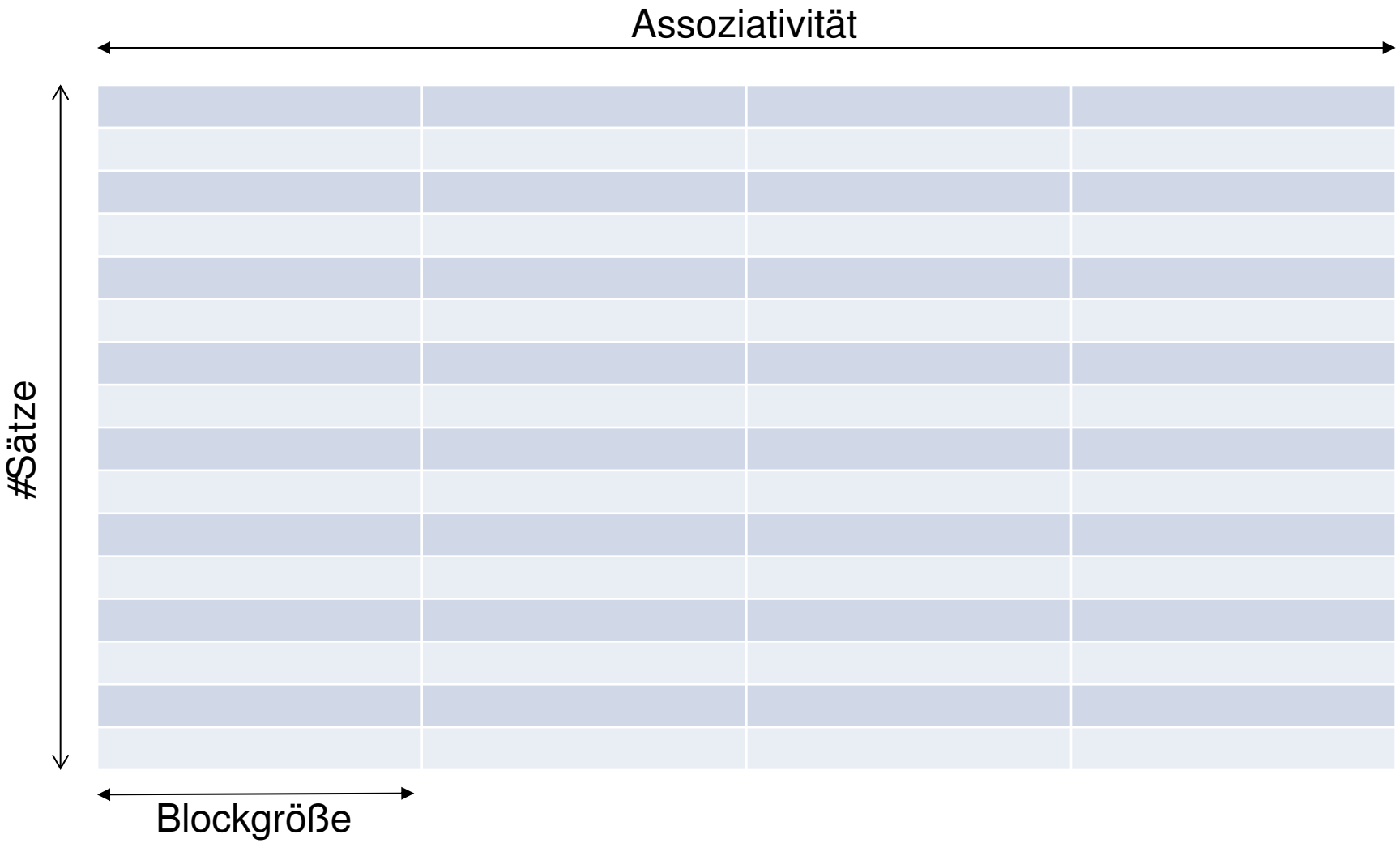
- Simulation eines 64KB Datencaches, 64-Byte Blöcke, SPEC2000:
  - 1-fach: 10.3%
  - 2-fach: 8.6%
  - 4-fach: 8.3%
  - 8-fach: 8.1%

Organisation eines 4-fach satzassoziativen Caches



- Wie groß ist ein Block?
- Wie groß ist dieser Cache?

Größe einer satzassoziativen Cache



Cache-Größe = #Sätze x Assoziativität x Blockgröße

Cache Übungen

Cache-Größe = #Sätze x Assoziativität x Blockgröße

- Gegeben:
  - Cache-Größe = 4 KB
  - 4-fach satzassoziativ
  - Blockgröße = 4 Wörter, Wort = 4 Bytes

Frage	Lösung
Wie viel Sätze?	
Wie viele Bits für Index?	
Wie viele Bits für Tag?	
Auf welchen Satz wird Byteadresse 4196 abgebildet?	

Cache-Größe = #Sätze x Assoziativität x Blockgröße

- Gegeben:
  - Cache-Größe = 4 KB
  - 4-fach satzassoziativ
  - Blockgröße = 4 Wörter, Wort = 4 Bytes

Frage	Lösung
Wie viel Sätze?	(4 x 1024) / (4 x 16) = 64
Wie viele Bits für Index?	
Wie viele Bits für Tag?	
Auf welchen Satz wird Byteadresse 4196 abgebildet?	

Cache Übungen

Cache-Größe = #Sätze x Assoziativität x Blockgröße

- Gegeben:
  - Cache-Größe = 4 KB
  - 4-fach satzassoziativ
  - Blockgröße = 4 Wörter, Wort = 4 Bytes

Frage	Lösung
Wie viel Sätze?	(4 x 1024) / (4 x 16) = 64
Wie viele Bits für Index?	$\log_2(64)$ = 6 Bits
Wie viele Bits für Tag?	
Auf welchen Satz wird Byteadresse 4196 abgebildet?	

Cache Übungen

Cache-Größe = #Sätze x Assoziativität x Blockgröße

- Gegeben:
  - Cache-Größe = 4 KB
  - 4-fach satzassoziativ
  - Blockgröße = 4 Wörter, Wort = 4 Bytes

Frage	Lösung
Wie viel Sätze?	(4 x 1024) / (4 x 16) = 64
Wie viele Bits für Index?	$\log_2(64)$ = 6 Bits
Wie viele Bits für Tag?	32 – 6 (Index) – 4 (Block Offset) = 22 Bits
Auf welchen Satz wird Byteadresse 4196 abgebildet?	

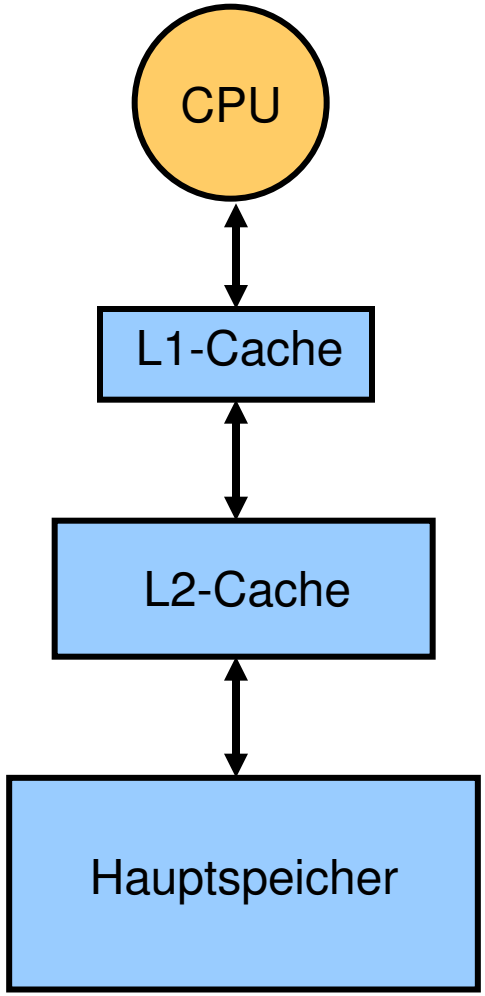
Cache Übungen

Cache-Größe = #Sätze x Assoziativität x Blockgröße

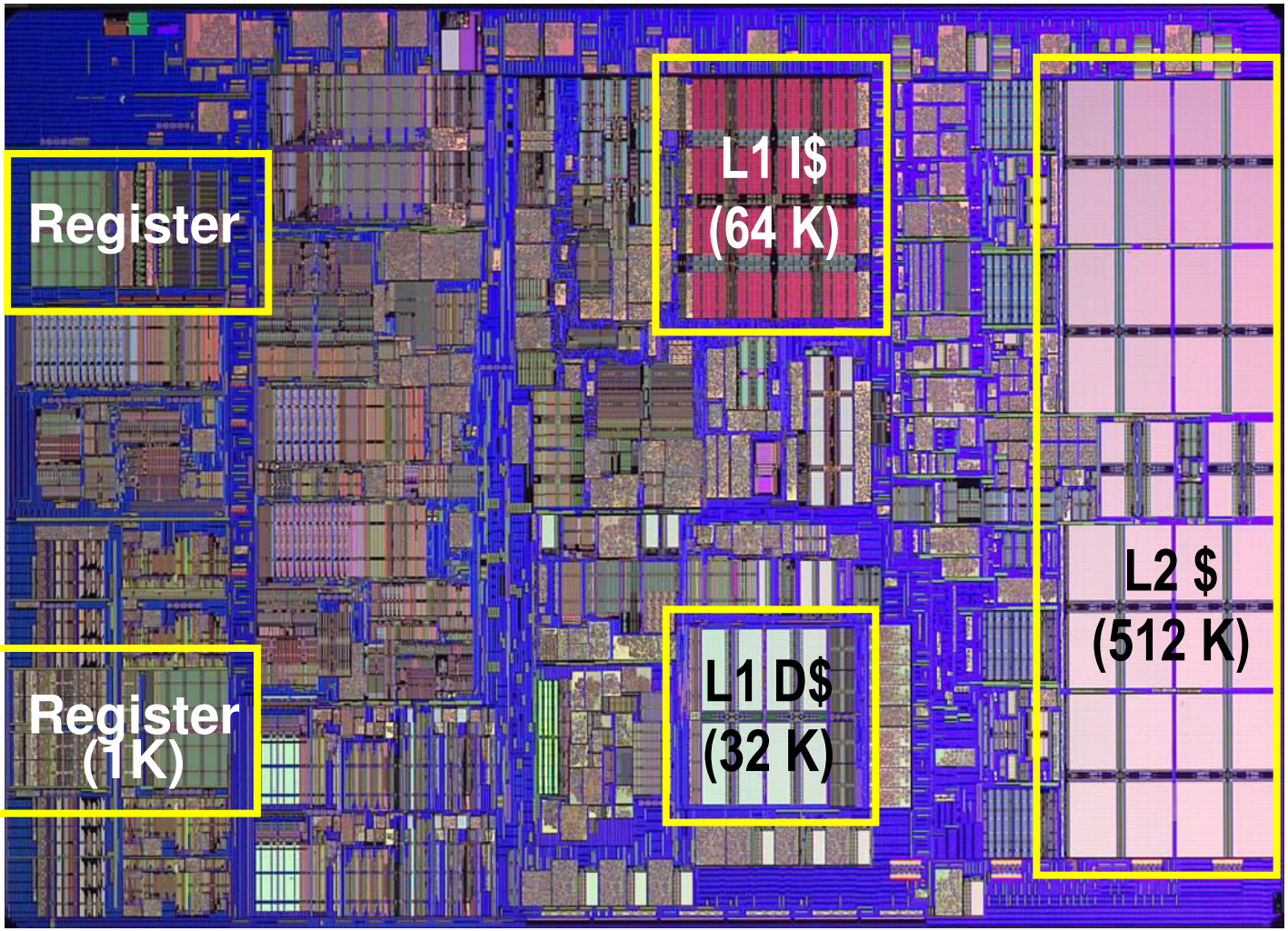
- Gegeben:
  - Cache-Größe = 4 KB
  - 4-fach satzassoziativ
  - Blockgröße = 4 Wörter, Wort = 4 Bytes

Frage	Lösung
Wie viel Sätze?	(4 x 1024) / (4 x 16) = 64
Wie viele Bits für Index?	$\log_2(64)$ = 6 Bits
Wie viele Bits für Tag?	32 – 6 (Index) – 4 (Block Offset) = 22 Bits
Auf welchen Satz wird Byteadresse 4196 abgebildet?	Blockadresse= 4196/16 = 262 Index = 262 mod 64 = 6

- Fehleraufwand reduzieren durch mehrere Cache-Ebenen (*cache levels*)
  - Cache-Speicherhierarchie
- Primärer Cache (L1-Cache) klein und schnell
  - Trefferzeit 1-3 Taktzyklen
- Sekundärer Cache (L2-Cache) größer und langsamer als L1, aber immer noch viel schneller als Hauptspeicher
  - Trefferzeit L2: 10-50 Taktzyklen
  - Hauptspeicher: 100-500 Taktzyklen
- Einige Hochleistungsprozessoren haben sogar L3-Cache



iMAC PowerPC: Alle Cache auf dem Chip



- Wir haben Prozessor mit
  - Grund-CPI von 1,0 (wenn alle Zugriffe im primären Cache treffen)
  - Taktgeschwindigkeit von 4 GHz (0,25ns Taktzykluszeit)
  - Hauptspeicher mit Zugriffszeit von 100ns (400 Taktzyklen)
  - Fehlrate_{L1} = 2% (Befehle und Daten)
- Nur L1-Cache:
  - CPI = Grund-CPI + Fehlrate_{L1} x Fehlaufwand = 1,0 + 0,02x400 = 9,0
- Nun L2-Cache hinzufügen mit
  - Zugriffszeit von 5ns (20 Taktzyklen)
  - Fehlrate_{L2} = L2 lokaler Fehlrate (#Fehler in L2 / #Zugriffe auf L2) = 25%
- CPI = Grund-CPI + Fehlrate_{L1} x Fehlaufwand_{L1}
- Fehlaufwand_{L1} = Trefferzeit_{L2} + Fehlrate_{L2} x Fehlaufwand_{L2}
- CPI = 1,0 + 0,02 x (20 + 0,25 x 400) = 3,4
- Prozessor mit L2-Cache ist 9,0/3,4 = 2,6 x schneller

Überlegungen zum Multilevel-Cache

- L1-Cache
  - zielt auf minimale Trefferzeit (*hit time*) ab
- L2-Cache:
  - zielt auf geringe Fehlzugriffsrate ab, um Hauptspeicher-Zugriffe zu vermeiden
  - Trefferzeit hat weniger Einfluss
- Fazit:
  - L1-Caches meist kleiner als bei Systeme, bei denen nur eine Cache-Ebene verwendet wird
  - L1-Blöcke kleiner als L2-Blöcke
  - L2 höhere Assoziativität als L1

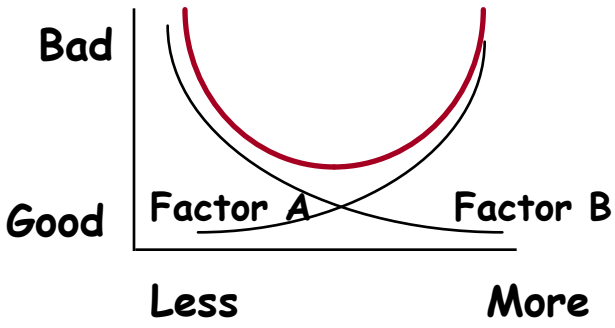
- Durchschnittliche Speicherzugriffszeit (*Average memory access time* (AMAT)):

  AMAT = Hit Time + Miss Rate x Miss Penalty

  - Um AMAT zu reduzieren, Trefferzeit, Fehlzugriffsrate und/oder Fehlzugriffsaufwand verringern.

Technik	Trefferzeit	Fehlzugriffsrate	Fehlzugriffsaufwand
Größere Blöcke		+	-
Höhere Assoziativität	-	+	
L2 Cache			+

- Alles im Bereich der Computer-Architektur ist ein Kompromiss (trade-off).



Fazit

- Um Prozessor/DRAM-Leistungslücke zu schließen, wird Cache-Speicherhierarchie verwendet
- Meiste Instruktionen/Daten werden im Cache gefunden wegen temporale und räumliche Lokalität
- Cache kann direkt abgebildet, satz-assoziativ oder voll-assoziativ sein.
- Moderne Hochleistungssysteme haben mehrere Cache-Ebenen
- Hilfreiche Cache-Formeln:
  - Cache-Größe = (# Sätze) x Assoziativität x (Blockgröße)
  - Blockadresse = (Byteadresse) / (Blockgröße in Bytes)
  - Cache-Index = (Blockadresse) modulo (# Sätze)
- Nicht merken, sondern verstehen!!!

<table>
<tr><td colspan="3">Blockadresse</td><td colspan="2">Block-offset</td></tr>
<tr><td colspan="2">Tag</td><td>Index</td><td>Wort-offset</td><td>Byte-offset</td></tr>
</table>