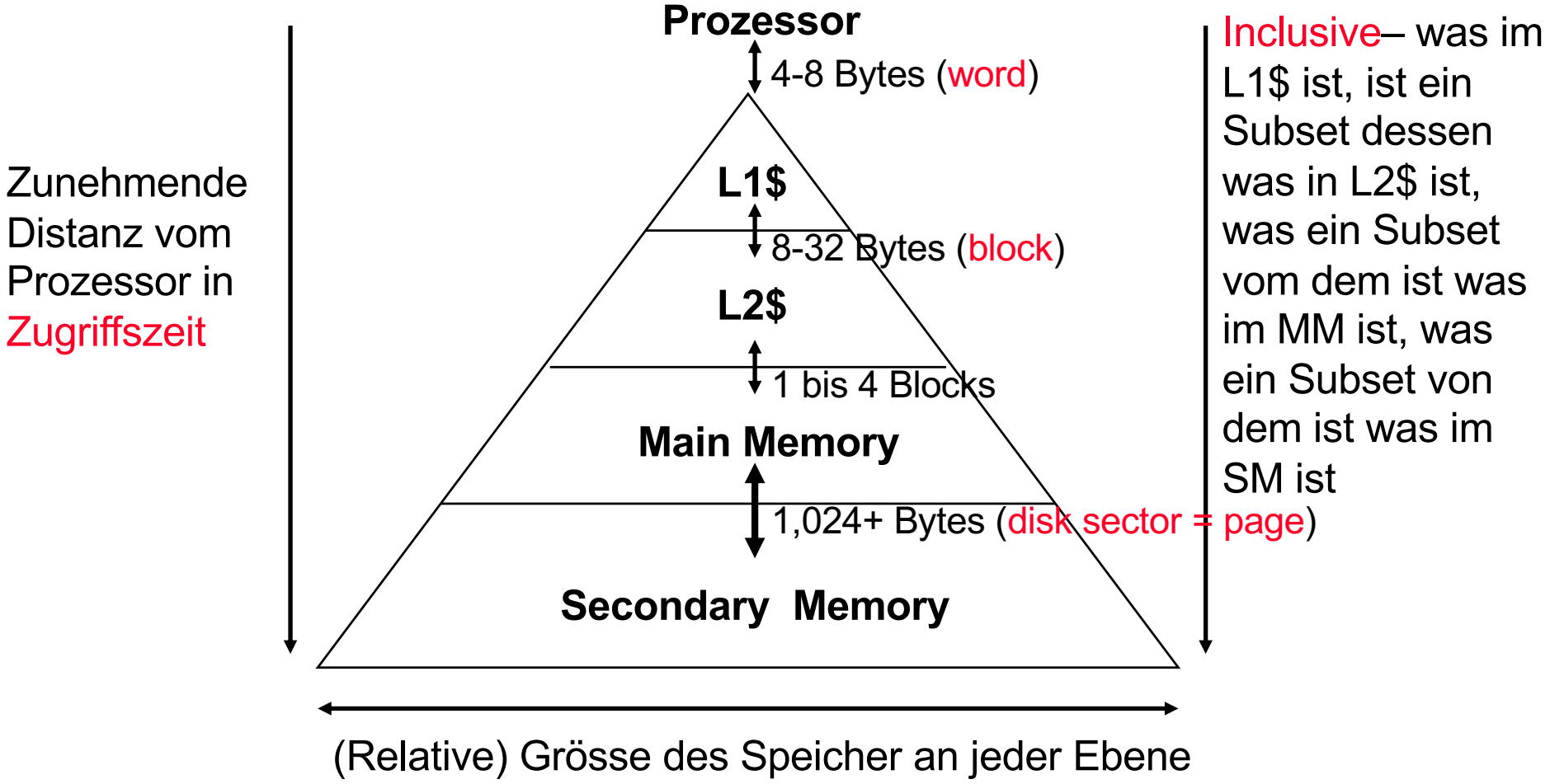
Cache Performance verbessern

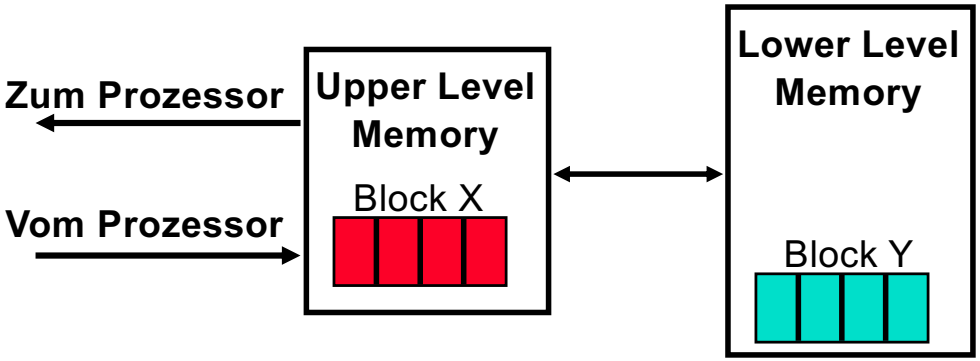
Review: Eigenschaften der Speicherhierarchie

- Nutzte den Vorteil des Lokalitätsprinzips um dem User so viel Speicher zur Verfügung zu stellen wie von der günstigsten Speichertechnologie verfügbar ist. Das mit der Geschwindigkeit der schnellsten Technologie



Review: Lokalitätsprinzips

- Zeitliche Lokalität (Lokalität in der Zeit):
  - Behalte die als letzte adressierten Daten näher am Prozessor
- Räumliche Lokalität
  - Verschiebe Blöcke aus benachbarten Wörter in höhere Levels



- Hit Time << Miss Penalty
  - Hit: Datenwert ist in einem Block im höherem Level (Blk X)
    - Hit Rate: Der Anteil von Daten gefunden im höheren Level
    - Hit Time: RAM Zugriffszeit + Zeit Bestimmung für hit/miss
  - Miss: Daten müssen aus tieferem Level geholt weren (Blk Y)
    - Miss Rate = 1 - (Hit Rate)
    - Miss Penalty: Time to replace a block in the upper level with a block from the lower level + Time to deliver this block's word to the processor
    - Miss Types: Compulsory, Conflict, Capacity

Messen der Cache Performance

- Annahme: Cache Hit Kosten sind Teil des CPU Ausführungszyklus, dann

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{CC}$$

$$= \text{IC} \times \underbrace{(\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles})}_{\text{CPI}_{\text{stall}}} \times \text{CC}$$

- Memory-Stall Zyklen kommen von cache misses (Summe von read-stalls und write-stalls)

$$\text{Read-stall Zyklen} = \text{reads/program} \times \text{read miss rate} \times \text{read miss penalty}$$

$$\text{Write-stall Zyklen} = (\text{writes/program} \times \text{write miss rate} \times \text{write miss penalty}) + \text{write buffer stalls}$$

- Für write-through Caches, können wir vereinfachen:

$$\text{Memory-stall cycles} = \text{readsWrites/program} \times \text{miss rate} \times \text{miss penalty}$$

Einfluss auf Cache Performance

- Relative «cache penalty» steigt mit der Prozessor-performance (schnellere Taktrate und/oder kleinere CPI)
  - Die Speichergeschwindigkeit verbessert sich nicht so schnell wie die Taktrate von Prozessoren. Beim Berechnen der CPI_{stall}, wird die «cache miss penalty» gemessen in Anzahl *Prozessor Taktraten* benötigt um den miss zu behandeln.
  - Je tiefer die CPI_{ideal}, je ausgeprägter der Einfluss von Stalls
- Ein Prozessor mit einer CPI_{ideal} von 2, einer 100 cycle miss penalty, 36% load/store Befehle, und 2% I$ und 4% D$ miss rate

$$\text{Memory-stall Zyklen} = 2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$$

$$\text{Daher:} \quad CPI_{stalls} = 2 + 3.44 = 5.44$$

- Was wenn die CPI_{ideal} auf 1 reduziert wird? 0.5? 0.25?
- Was wenn die Prozessor Taktrate verdoppelt wird (verdoppeln der miss penalty)?

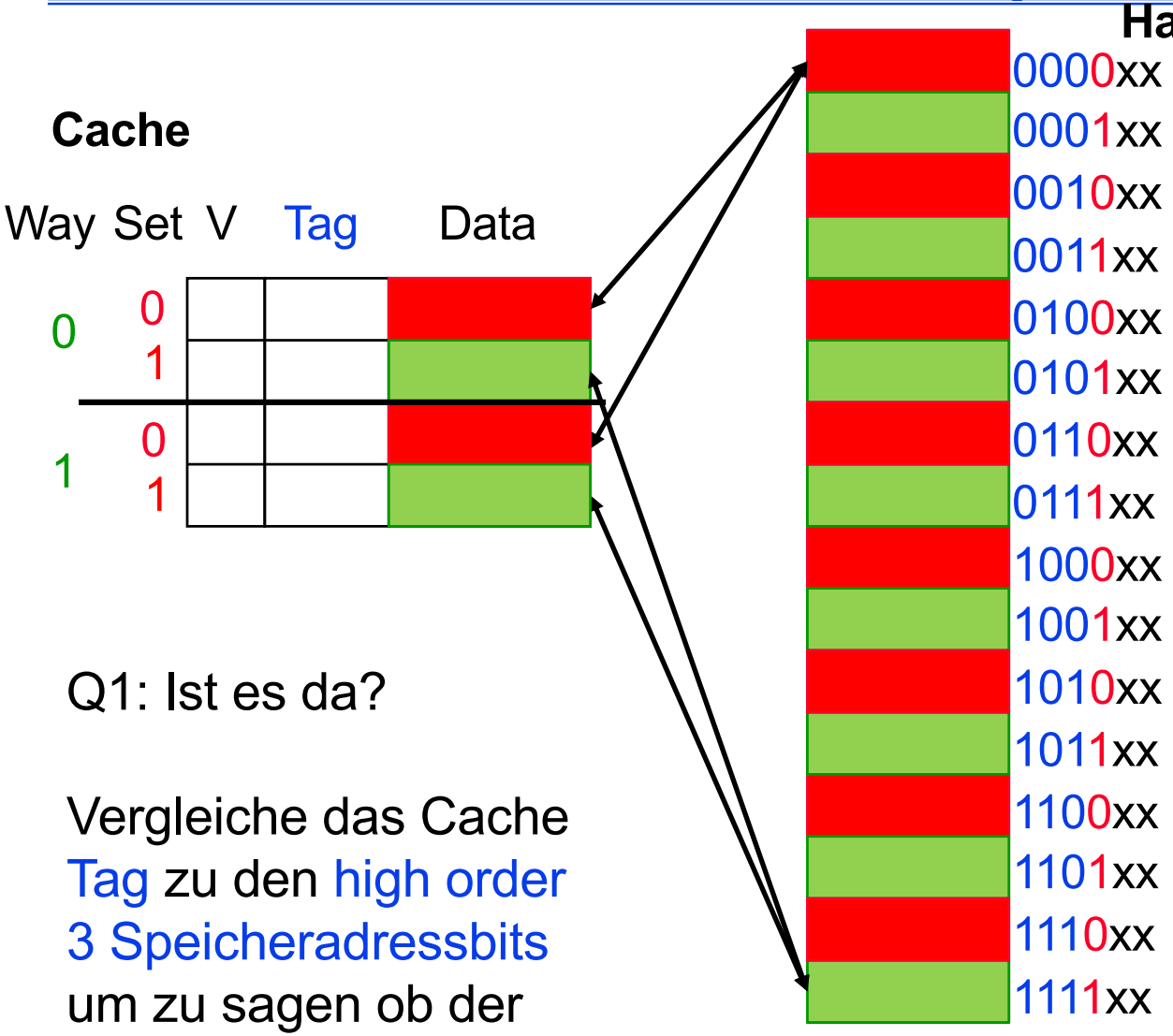
Reduzieren der Cache Miss Rate #1

1. Erlauben wir eine flexiblere Block Platzierung

- In einem direct mapped cache wird ein Speicherblock auf einen spezifischen einzelnen Cache Block abgebildet
- Das andere Extrem: Erlauben, dass ein Speicherblock auf jeden Cache Block abgebildet werden kann – fully associative cache
- Ein Kompromiss: Den Cache in Sets unterteilen. Jedes Set besteht aus n “ways” (n-way set associative). Ein Speicherblock wird dann auf ein spezifisches Set abgebildet (spezifiziert durch das Indexfeld) und kann in jedem „way“ auf diesem Set platziert werden (ergibt n Möglichkeiten)

(Blockadresse) modulo (# Sets im Cache)

Set Associative Cache Beispiel



Die beiden *low order* Bits ‘xx’ definieren das Byte im Wort (32 Bit Worte)
Ein Wort Blöcke

Q1: Ist es da?

Vergleiche das Cache Tag zu den high order 3 Speicheradressbits um zu sagen ob der Speicherblock im Cache ist

Q2: Wie finden wir es?

Nutze nächstes *low order* Speicheradressbit um das Cache Set zu bestimmen (i.e., modulo der Anzahl Sets im Cache)

Noch ein Sequenz Mapping

- Annahme: Hauptspeicher Wort Adress-Sequenz

Starten mit leerem Cache – alle Blöcke sind initial als *not valid* markiert

0 4 0 4 0 4 0 4

0 miss

000	Mem(0)

4 miss

000	Mem(0)
010	Mem(4)

0 hit

000	Mem(0)
010	Mem(4)

4 hit

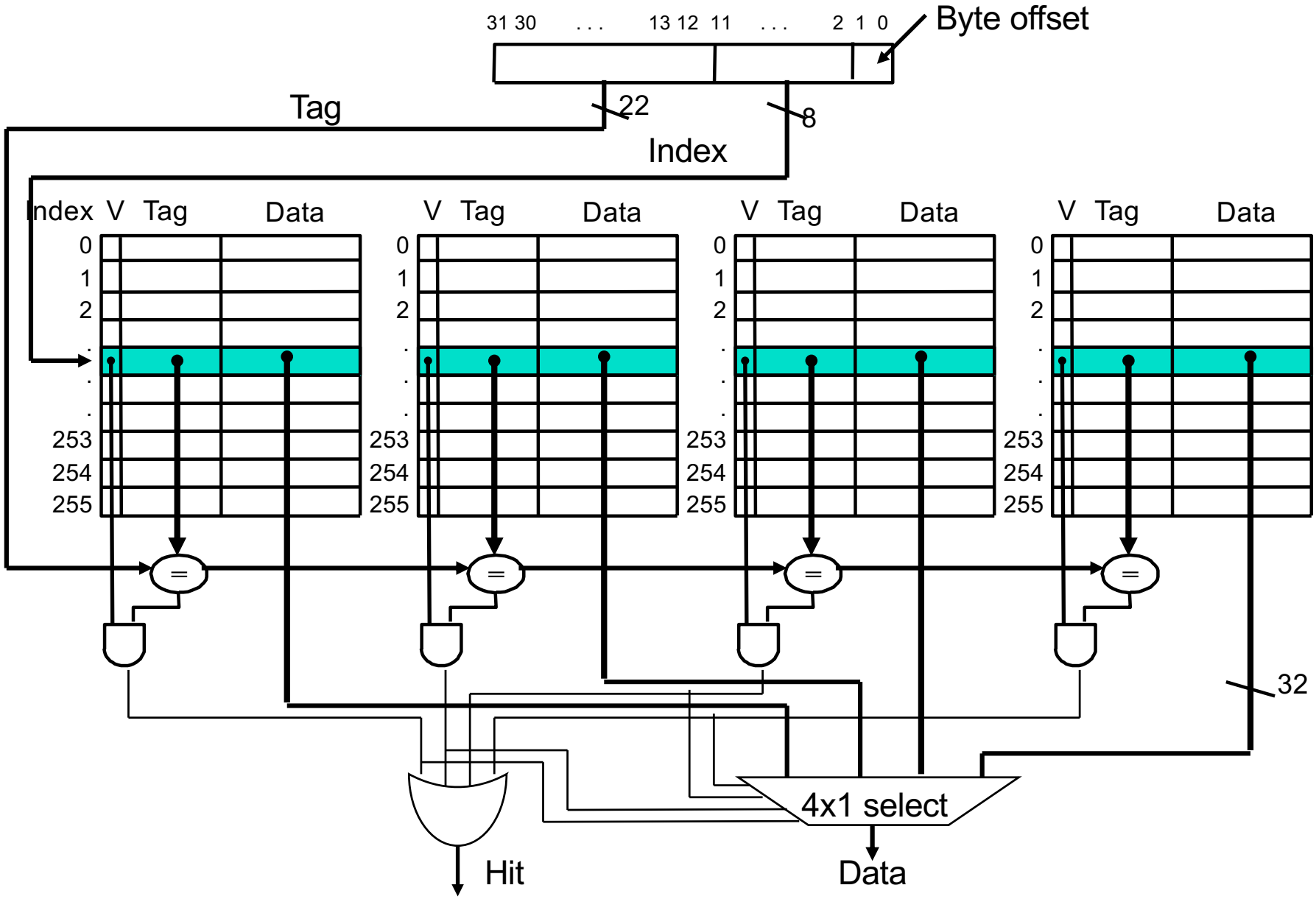
000	Mem(0)
010	Mem(4)

- 8 requests, 2 misses

- Behebt den Ping Pong Effekt in einem «direct mapped cache» durch «conflict misses» weil jetzt zwei Speicherorte in das selbe Cache Set zeigen

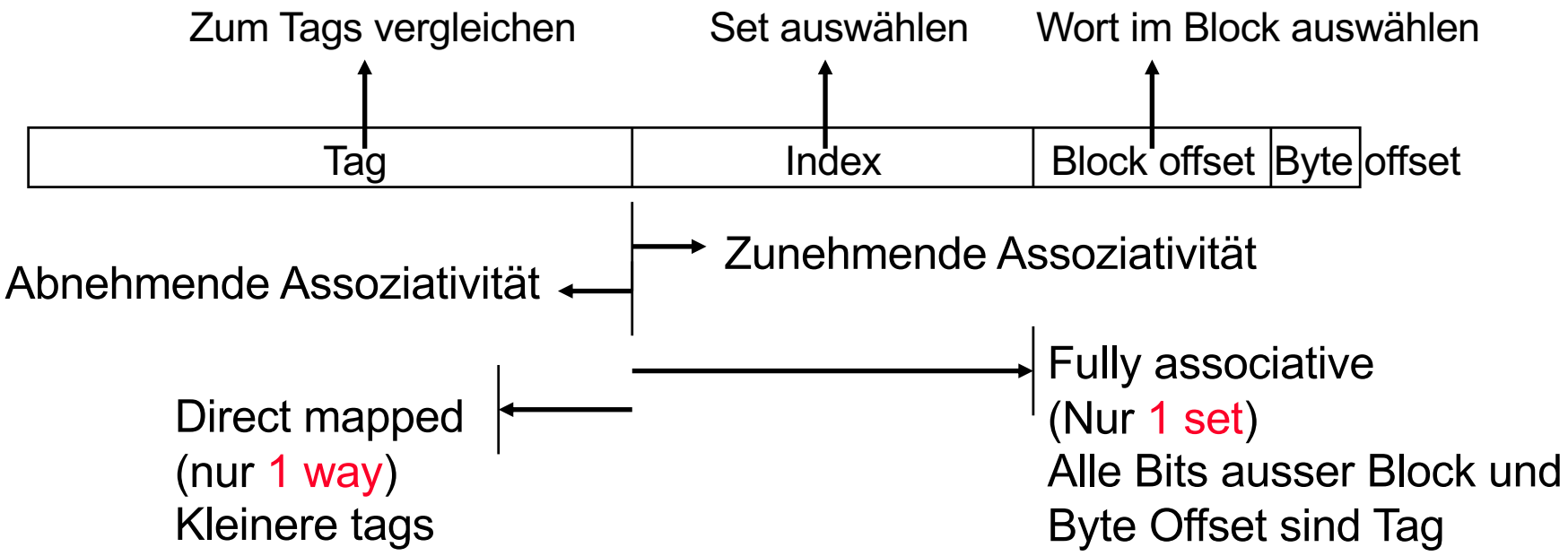
4-Way Set Associative Cache

- 2^8 = 256 Sets, jedes mit vier ways (jedes mit einem Block)



Spannbreiten eines Set Associative Caches

- Wir erhöhen die Assoziativität um den Faktor 2, bei einem Cache mit fixer Grösse
  - Verdoppelt Anzahl Blöcke in einem Set
  - Halbiert Anzahl Sets
  - Index: -1 Bit; Tag: +1 Bit

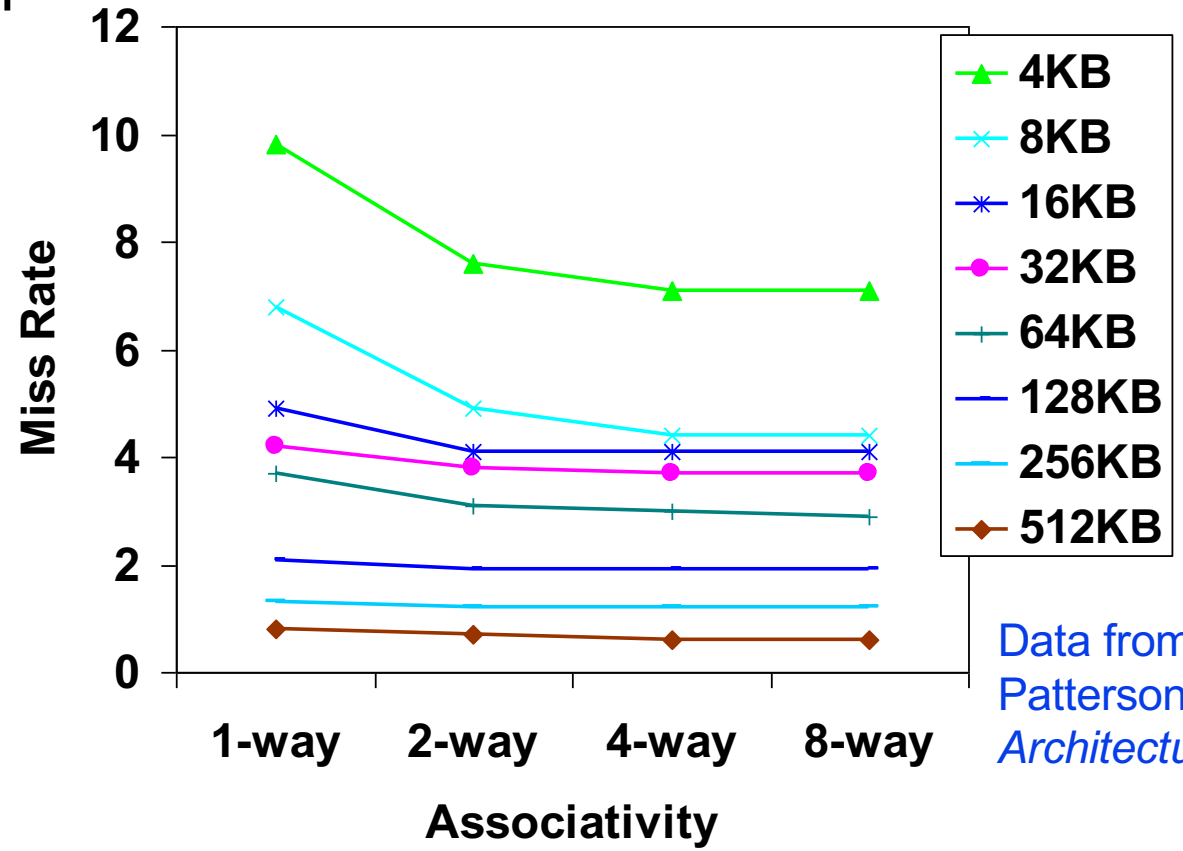


Kosten von Associative Caches

- Wie wählen wir bei einem Miss den Block aus welchen wir ersetzten?
  - Least Recently Used (LRU): Der Block der an längsten nicht benutzt wurde wird ersetzt
    - Benötigt Hardware für den Überblick jedes Blocks eines «Way» wie er im Bezug zu den anderen Blocks im Set benutzt wurde
    - Für 2-way set associative, benötigen wir ein Bit pro Set → setze das Bit wenn der eine Block referenziert wird
      (und setzte es auf 0 wenn der Andere way referenziert wird)
- N-way set associative cache Kosten
  - N Komparatoren (Erzeugen Delay und benötigen Platz)
  - MUX erzeugt Delay bevor Daten verfügbar sind (Set Auswahl)
  - Daten verfügbar **nach** Set Auswahl (und Hit/Miss Entscheidung).
    In einem «direct mapped cache» ist der Cacheblock vor der Hit/Miss Entscheidung verfügbar
    - Deshalb ist es nicht möglich einen Hit nur anzunehmen, weiterzufahren und erst später, bei einem miss, den korrekten Ablauf wiederherzustellen

Vorteile von Set Associative Caches

- Die Wahl von «direct mapped» oder «set associative» ist Abhängig von den Kosten eines Miss versus den Kosten der Implementation



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Grösste Gewinne gehen vom direct mapped zum 2-way (20%+ Reduktion in miss rate)

Reduzieren der Cache Miss Rate #2

2. Nutze mehrere Cache Level

- Mit der technologischen Weiterentwicklung haben wir nun mehr als genügen Platz auf einem Chip für einen grösseren L1 Cache oder einen Second-Level Cache – normalerweise ein unified L2 Cache (enthält Befehle und Daten) und immer öfter einen einen unified L3 Cache

- Beispiel: CPI_{ideal} von 2, 100 «cycle miss penalty» (zum Hauptspeicher), 36% load/stores, eine 2% (4%) L1 I$ (D$) miss rate. Dazu ein UL2$ mit 25 «cycle miss penalty» und einer 0.5% miss rate

$$CPI_{stalls} = 2 + .02 \times 25 + .36 \times .04 \times 25 + .005 \times 100 + .36 \times .005 \times 100 = 3.54$$

(im Vergleich zu 5.44 ohne L2$)

Überlegungen zu Multilevel Cache Design

- Design Überlegungen bei L1 und L2 Caches sind sehr Unterschiedlich
  - Primary Cache: Soll sich auf die «hit time» Minimierung fokussieren. Ziel: Erreichen einer kurzen Taktperiode
    - Kleiner mit kleineren Blockgrössen
  - Secondary Cache(s): Soll sich auf die «miss rate» Reduzierung fokussieren. Ziel: Reduzieren der grossen «Penalty» bei Hauptspeicherzugriffen
    - Grösser mit grösseren Blockgrössen
- Die «miss penalty» des L1 Cache wird durch die Gegenwart eines L2 Cache signifikant reduziert – daher kann er kleiner sein (i.e., schneller), hat aber eine höhere «miss rate»
- Für den L2 Cache, ist die «hit time» weniger wichtig als die «miss rate»
  - Die L2$ «hit time» determiniert die L1$I/L1$D «miss penalty»

Cache Design Schlüsselparameter

	L1 typical	L2 typical
Total Grösse (Blöcke)	250 bis 2000	4000 bis 250,000
Total Grösse (KB)	16 bis 64	500 bis 8000
Blockgrösse(B)	32 bis 64	32 bis 128
Miss penalty (Takte)	10 bis 25	100 bis 1000
Miss rates (global für L2)	2% bis 5%	0.1% bis 2%

Cache Parameter zweier Maschinen

	Intel P4	AMD Opteron
L1 organization	Geteilter I$ und D$	Geteilter I$ und D$
L1 cache size	8KB für D$, 96KB für trace cache (~I$)	64KB für jeden I$ und D$
L1 block size	64 Bytes	64 Bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 Bytes	64 Bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

4 Fragen zur Speicherhierarchie

- Q1: Wo kann ein Block im upper Level platziert werden?
  (Block placement)
- Q2: Wie findet man einen Block wenn er im upper Level ist?
  (Block identification)
- Q3: Welcher Block soll bei einem Miss ersetzt werden?
  (Block replacement)
- Q4: Was passiert bei einem write?
  (Write strategy)

Q1&Q2: Wo kann Block platziert/gefunden werden?

	# Sets	Blocks pro Set
Direct mapped	# Blöcke im Cache	1
Set associative	(# Blöcke im Cache)/ Assoziativität	Assoziativität (typischer-weise 2 bis 16)
Fully associative	1	# Blöcke im Cache

	Lokalisierungsmethode	# Vergleiche
Direct mapped	Index	1
Set associative	Index des Set; vergleiche Tags der Sets	Grad der Assoziativität
Fully associative	Vergleiche Blocks Tags	# of blocks

Q3: Welcher Block soll bei Miss ersetzt werden?

- Einfach für «direct mapped» – nur eine Möglichkeit
- «set associative oder «fully associative»
  - Zufall
  - LRU (Least Recently Used)
- Für ein «2-way set associative cache», hat zufälliges Ersetzen eine «miss rate» um die 1.1 mal höher als LRU.
- LRU ist zu teuer für höhere «Levels of associativity» (> 4-way). Das nachverfolgen der Nutzung ist aufwendig/teuer.

Q4: Was passiert bei einem write?

- *Write-through* – Die Informationen werden in den Block im Cache und in den Block des nächsttieferen Cache der Speicherhierarchie geschrieben
  - Write-through wird immer mit einem Schreibpuffer kombiniert. Verzögerungen durch langsames schreiben können so eliminiert (so lange sich der Schreibpuffer nicht füllt)
- *Write-back* – Die Informationen werden nur in den Block im Cache geschrieben. Der modifizierte Cacheblock wird erst beim Ersetzten in den Hauptspeicher geschrieben.
  - Benötigt ein «dirty bit» um modifizierte Blöcke nachzuverfolgen
- Vor- und Nachteile der Strategien?
  - Write-through: read misses führen nicht zu einem write (damit einfacher und günstiger)
  - Write-back: wiederholtes schreiben benötigt nur ein write in den tieferen Level

Verbessern der Cache Performance

0. Reduzieren der hit time im Cache

- Kleinerer Cache
- Direct mapped cache
- Kleinere Blöcke
- für writes
  - no write allocate – kein “hit” im Cache, nur in den Schreibpuffer schreiben
  - write allocate – um zwei Zyklen zu vermeiden (Erst auf hit prüfen, dann schreiben) Pipeline schreibt via einem verzögernden Schreibpuffer in den Cache

1. Reduzieren der miss rate

- Grösserer Cache
- Flexibleres platzieren (erhöhen der Associativität)
- Grössere Blöcke (typischerweise: 16 bis 64 Bytes)
- Victim Cache – kleiner Puffer mit den letzten verworfenen Blöcken

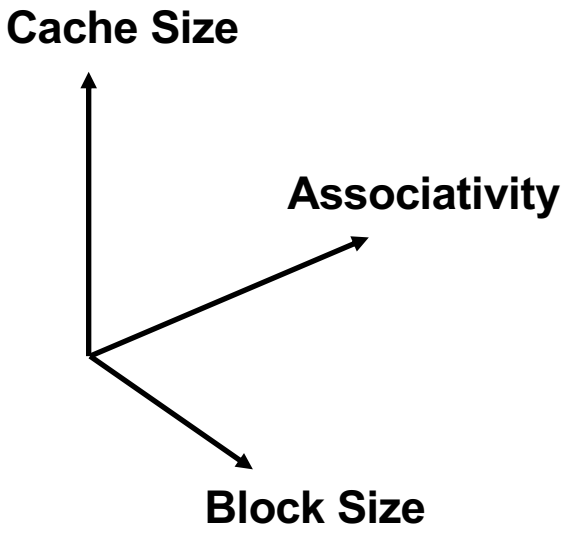
Verbessern der Cache Performance

2. Reduzieren der miss penalty

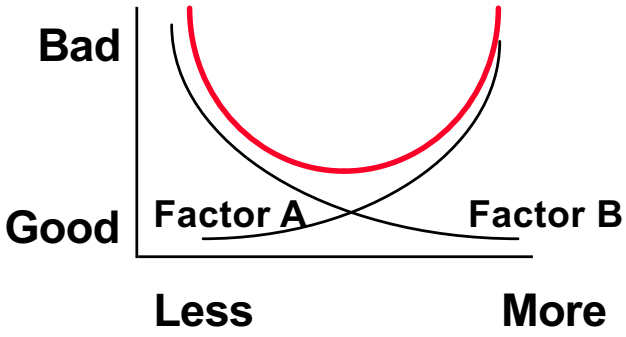
- Kleinere Blöcke
- Nutze einen Schreibpuffer für das Ersetzten von «Dirty blocks». So muss man mit den lesen nicht warten bis das Schreiben beendet ist.
- Prüfe Schreibpuffer (und/oder victim cache) bei read miss – vielleicht hat man Glück
- Für grosse Blöcke nimm das kritische Wort zuerst
- Nutze mehrere Cache Levels – L2 Cache ist nicht an die CPU Taktrate gebunden
- Schnellerer «backing store»/bessere Speicherbandbreite
  - breitere Buse
  - Speicherverschränkung (memory interleaving), Page Mode DRAMs

Zusammenfassung: Cache Design Space

- Mehrere wechselwirkende Dimensionen
  - Cachegrösse
  - Blockgrösse
  - Assoziativität
  - Replacement Strategie
  - Write-through vs write-back
  - Write allocation



- Die optimale Wahl ist ein Kompromiss
  - Abhängig von der Zugriff Charakteristik
    - Auslastung
    - Nutzung (I-cache, D-cache, TLB)
  - Abhängig von Technologie / Kosten



- Simplizität gewinnt oft