

Rechnerarchitektur (RA)

Sommersemester 2016

Verbesserung der Leistungsfähigkeit von Caches

Jian-Jia Chen

Informatik 12

[jian-jia.chen@tu-..](mailto:jian-jia.chen@tu-dortmund.de)

<http://ls12-www.cs.tu-dortmund.de/daes/>

Tel.: 0231 755 6078

2016/06/21

Cache-Performanz (1)

- *CPU-Ausführungszeit* =
 $(\# \text{ CPU-Takte} + \# \text{ Speicher-Wartezyklen}) \times \text{Taktzeit}$
- Einbeziehung von *Speicher-Wartezyklen* (*memory stall cycles*) = Takte, in denen CPU auf Speicher wartet
- Annahmen:
 - *CPU-Takte* enthalten Zeit für erfolgreichen Cache-Zugriff
 - *Speicher-Wartezyklen* treten nur bei Fehlzugriffen auf

Cache-Performanz (2)

Speicher-Wartezyklen abhängig von:

- Anzahl der Fehlzugriffe (*cache misses*) und
- „Kosten“ pro *cache miss* (= *miss penalty*)
- *Instruction count IC*

Speicher-Wartezyklen =

$$= \# \text{ Cache misses} \times \text{miss penalty}$$

$$= IC \times (\text{misses} / \text{Befehl}) \times \text{miss penalty}$$

$$= IC \times (\# \text{ Speicherzugriffe} / \text{Befehl}) \times \\ \times \text{Fehlzugriffsrate} \times \text{miss penalty}$$

- $\# \text{ Speicherzugriffe} / \text{Befehl} > 1$, da immer 1x Befehlsholen
- *miss penalty* ist eigentlich Mittelwert
- Alle Größen messbar (ggf. durch Simulation)

Cache-Performanz (3)

Bewertungsmaß für die Leistungsfähigkeit einer Speicherhierarchie:

$$\text{Mittlere Zugriffszeit} = \text{Hit Time} + \text{Miss Rate} \times \text{Miss penalty}$$

- *Hit Time* = Zeit für erfolgreichen Zugriff (in CPU-Leistungsgleichung Teil der „normalen“ Ausführungszeit)
- *Miss Rate* = Anteil der Fehlzugriffe
- *Miss Penalty* = Wartezeit der CPU auf Speicher, die pro Fehlzugriff entsteht

Maßeinheiten: physikalische Zeit oder Taktzyklen möglich

Verbesserung der Leistungsfähigkeit von Caches: Übersicht

- Beeinflussende Größen (Kapitel 5.2, Hennessy, 3./4. Aufl.):
 - *Hit Time*
 - *Miss Rate*
 - *Miss Penalty*
 - *Cache Bandwidth*

Wir betrachten daher im folgenden Verfahren zur:

- Reduktion der Rate fehlgeschlagener *Cache*-Zugriffe,
- Reduktion der Zugriffszeit im Erfolgsfall,
- Erhöhung der *Cache*-Bandbreite,
- Reduktion der Wartezyklen, die ein fehlgeschlagener Zugriff erzeugt.

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Miss Rate ↓:

Klassische Methode zur Cache-Leistungssteigerung.

Typen von *Cache-Misses*:

- *Compulsory*
Erster Zugriff auf Speicherbereich kann nie im Cache sein.
- *Conflict*
Bei *direct mapped* und *n*-Wege assoziativem \$ können Konflikte bzgl. des Ablegens eines Blocks auftreten
(d.h. zu viele Blöcke werden auf gleiches „Set“ abgebildet)
- *Capacity*
Misses, die bei unbegrenzter Assoziativität auftreten würden

Bei Kapazitätsüberschreitung: Blöcke verwerfen, später wieder holen

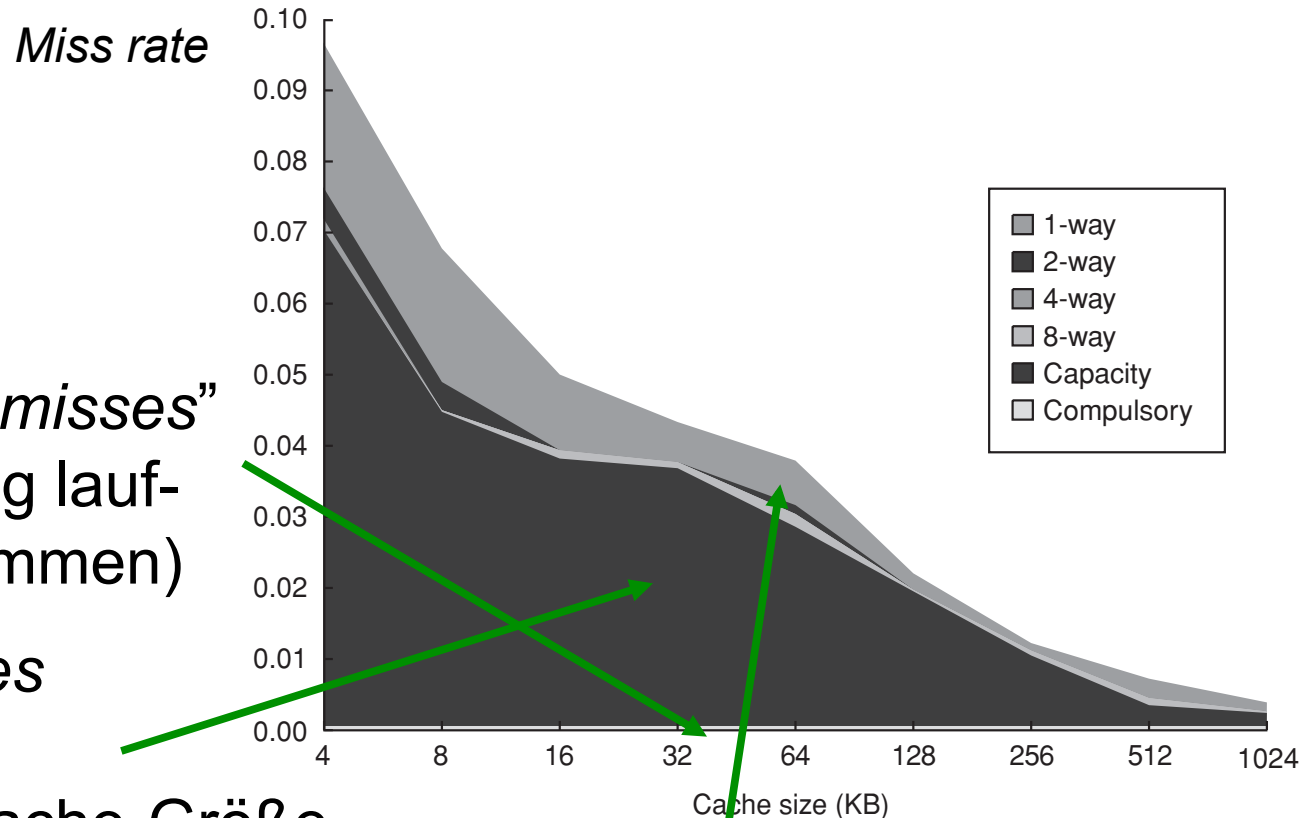
Worst Case: Cache viel zu klein

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Miss Rate ↓: (2)

Anteil der Fehlzugriffstypen (SPEC2000)

- Extrem wenige zwangsweise „*misses*“ (normal bei lang laufenden Programmen)
- *Capacity misses* reduziert mit wachsender Cache-Größe



© 2006 Elsevier Science

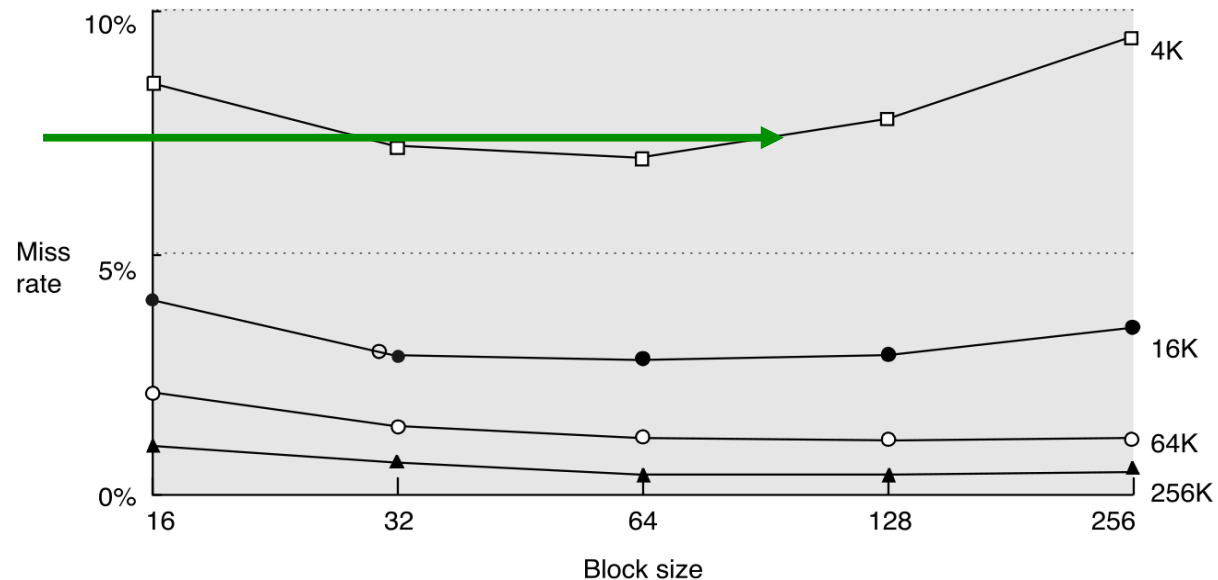
- Weitere Fehlzugriffe durch fehlende Assoziativität (Konflikte; bzgl. voll assoziativem *Cache*)

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Miss Rate ↓: Größere Cache-Blöcke

Reduziert zwangsweise Fehlzugriffe wegen örtlicher Lokalität

- längere Ladezeiten
- # Blöcke reduziert, evtl. mehr Konflikt- und Kapazitätsfehlszugriffe



© 2003 Elsevier Science

- Kosten des Fehlzugriffs dürfen Vorteile nicht überwiegen
- Abhängig von Latenz und Bandbreite des Speichers

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Uebung

	Cache Size			
Block Size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Assume that the memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles. Thus, it can supply 16 bytes in 82 clock cycles and 32 bytes in 84 clock cycles. Which block size has the smallest average memory access time for each cache size above?

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Miss Rate ↓: Größere Caches

Nachteile:

- Höhere Kosten
- Längere Cache-Zugriffszeit



Bei 2-stufigen Caches ist Zugriffszeit der 2. Ebene nicht primär ausschlaggebend

☞ Hier größere Caches verwenden (ggf. langsamer)

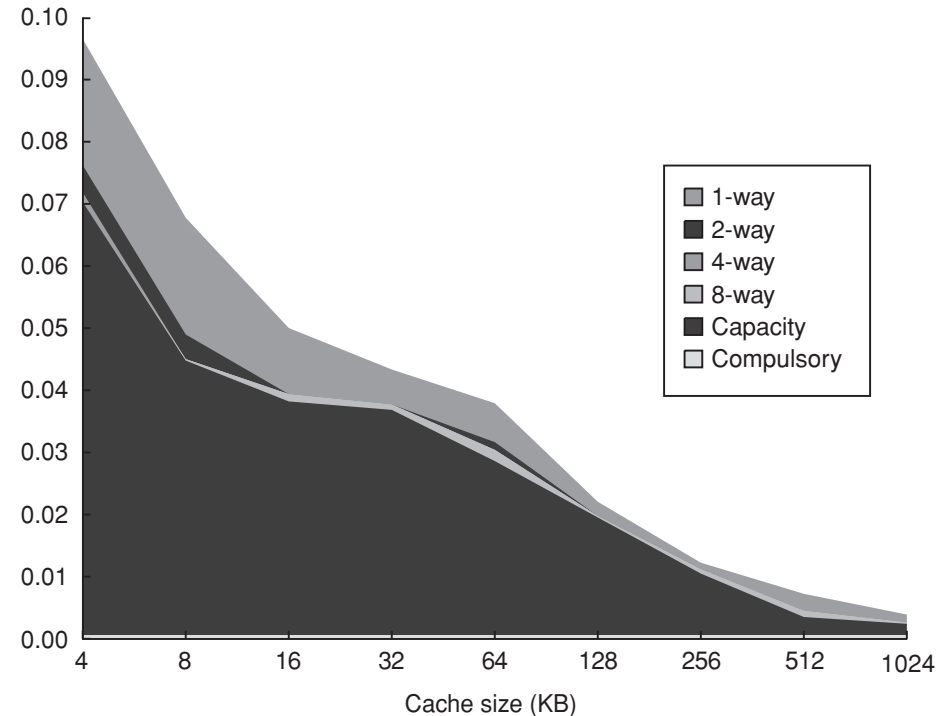
Verbreitete Lösung: Große Caches als 2nd- oder 3rd-level Caches haben heute Kapazitäten wie Hauptspeicher früher!

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Miss Rate ↓: Höhere Assoziativität

Assoziativität reduziert
Fehlzugriffsrate

- In der Praxis 8-Wege assoziativ \cong voll assoziativem Cache (aber: deutlich geringere Komplexität!)!
- „2:1-*Cache*-Daumenregel“
direct mapped Cache der Größe N hat ca. Fehlzugriffsrate von 2-Wege assoziativem Cache halber Größe
- Hohe Assoziativität erhöht Zykluszeit, da Cache (1. Ebene) direkt mit CPU gekoppelt!



- Miss Rate
- **Hit Time**
- Cache Bandwidth
- Miss Penalty

Hit Time ↓: Kleine und einfache Caches

Zugriffszeit im Erfolgsfall (*hit time*) kritisch für Leistung von Caches, da sie CPU-Takt begrenzt

- Indizierung des *Tag*-Speichers (via Index) und Vergleich mit angefragter Adresse ➡ schnell, wenn *Cache* klein
- Kleiner Cache kann *on-chip* (der CPU) realisiert werden ➡ kürzere Signallaufzeiten, effizienterer Datentransfer
- Einfacher, d.h. *direct mapped*:
➡ Überlappung von Datentransfer und *Tag*-Prüfung möglich

➡ Größe des L1-Caches ggf. nicht erhöht, evtl. sogar reduziert (Pentium III: 16KB, Pentium 4: 8KB)

Schwerpunkt auf höherer Taktrate, Kombination mit L2 *Cache*

- Miss Rate
- **Hit Time**
- Cache Bandwidth
- Miss Penalty

Hit Time ↓: Kleine und einfache Caches

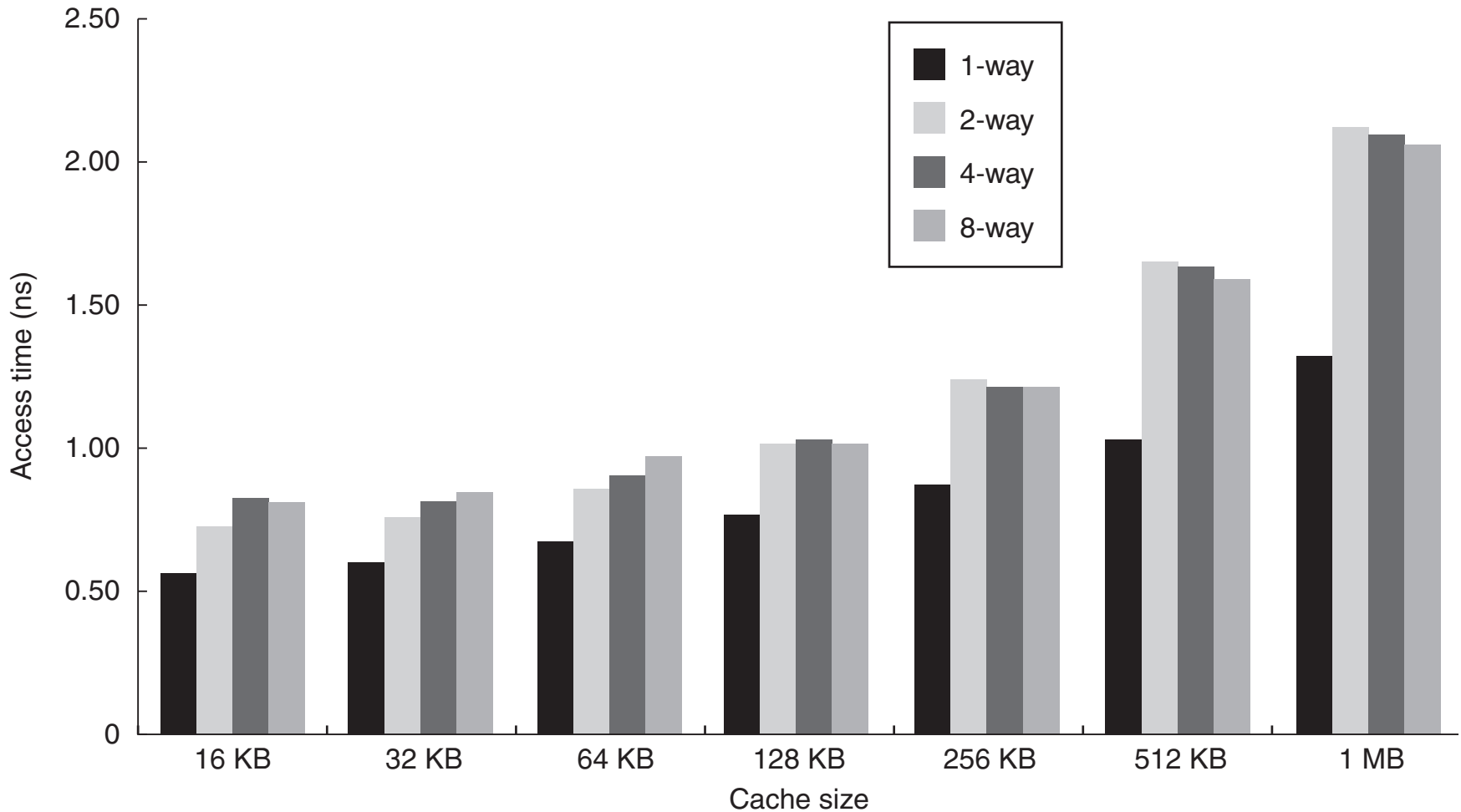


Abb. 5.4. aus HP07, © Elsevier

- *Miss Rate*
- ***Hit Time***
- *Cache Bandwidth*
- *Miss Penalty*

***Hit Time* ↓: Keine Adress- umrechnung bei Cache-Zugriff**

Aktuelle PCs verwenden virtuellen Speicher:

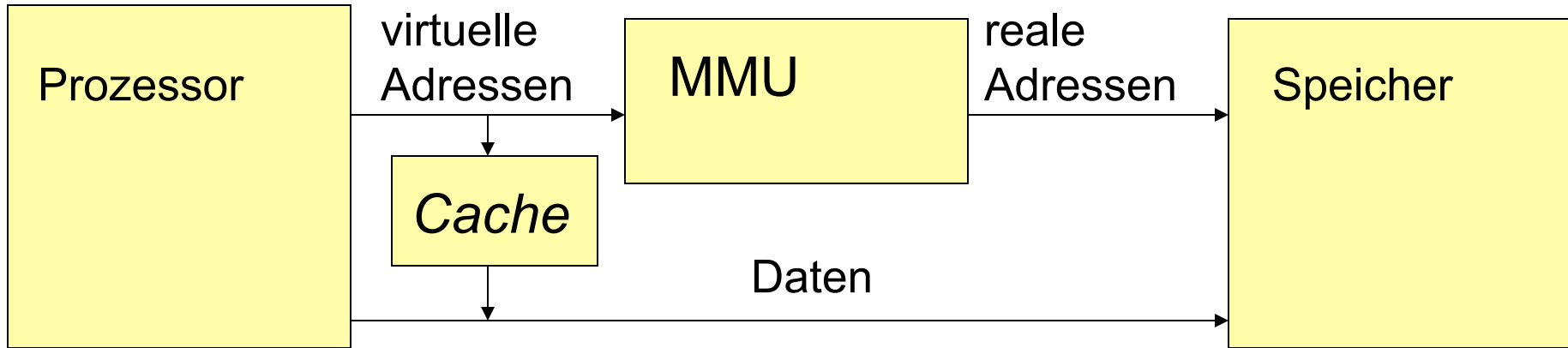
Virtuelle oder reale Caches?

Siehe Kurs „Rechnerstrukturen“

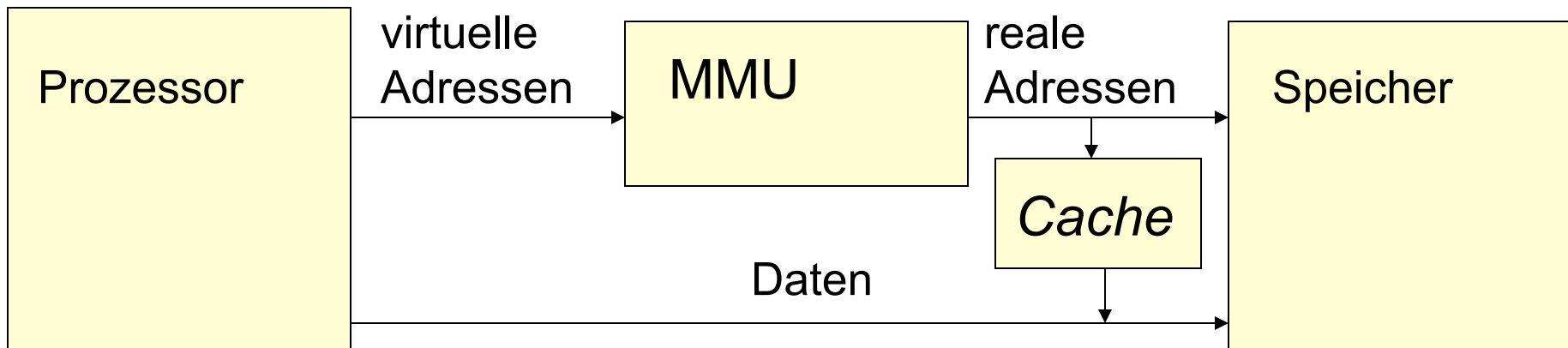
Hier: Zusätzliche Betrachtung von Mischformen

Zur Erinnerung aus RS

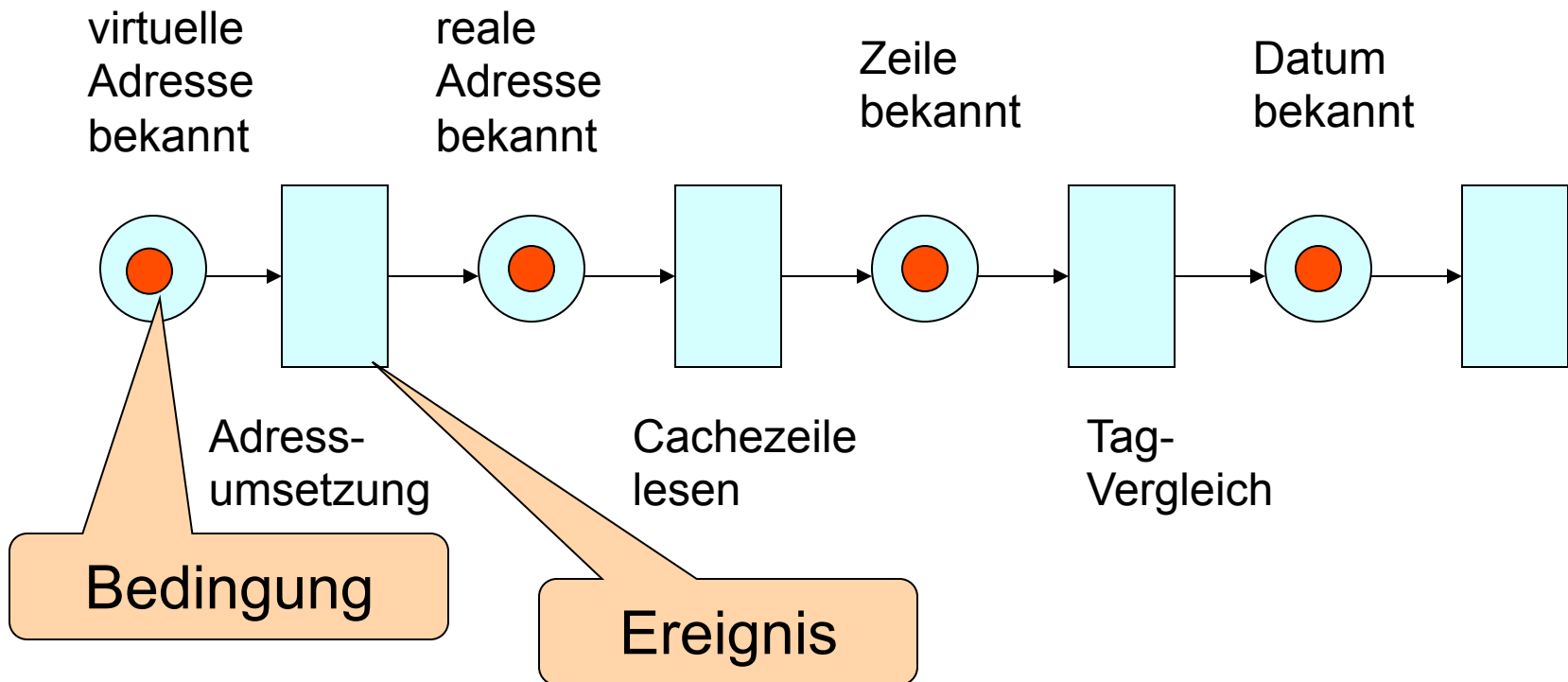
Virtuelle Caches (schnell)



Reale Caches (langsamer)

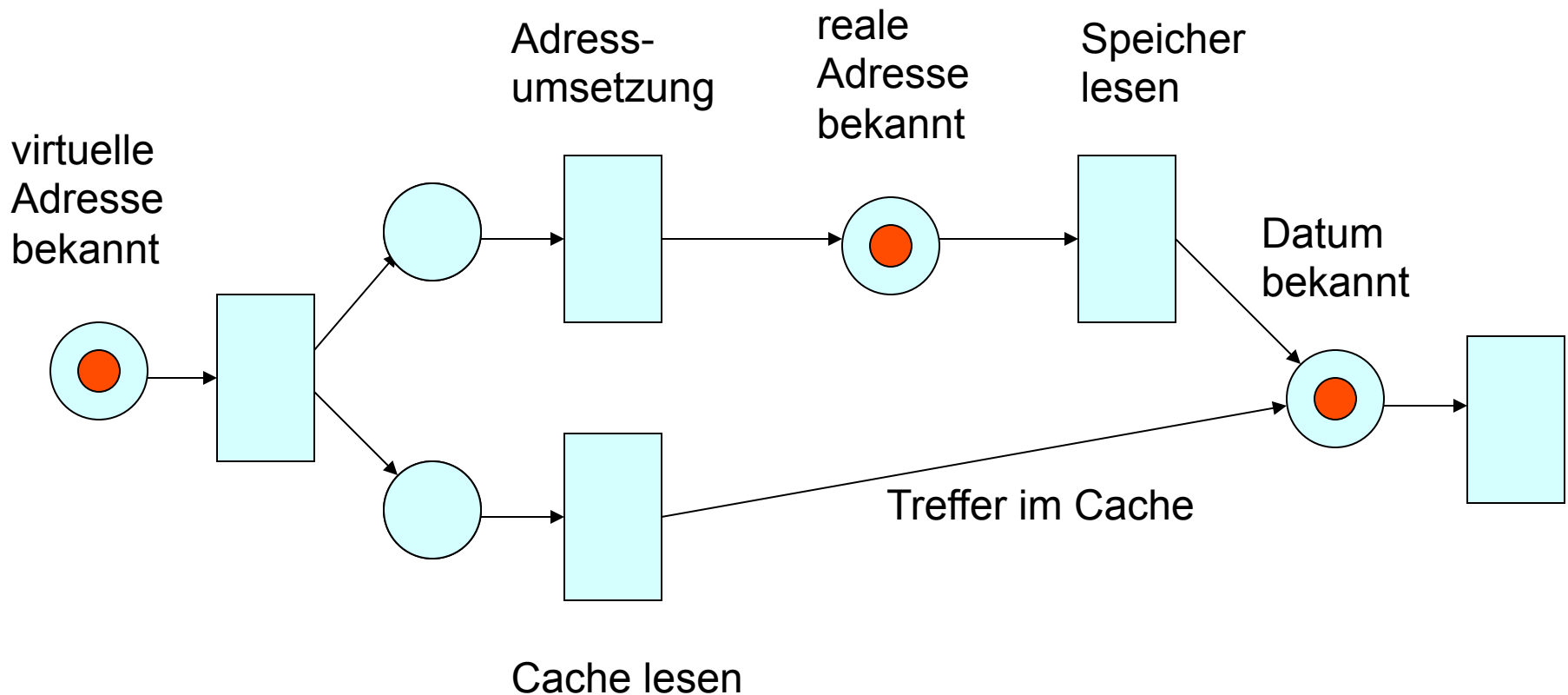


Modell des Ablaufs bei realen Caches



Adressumsetzung und Lesen des Caches nacheinander

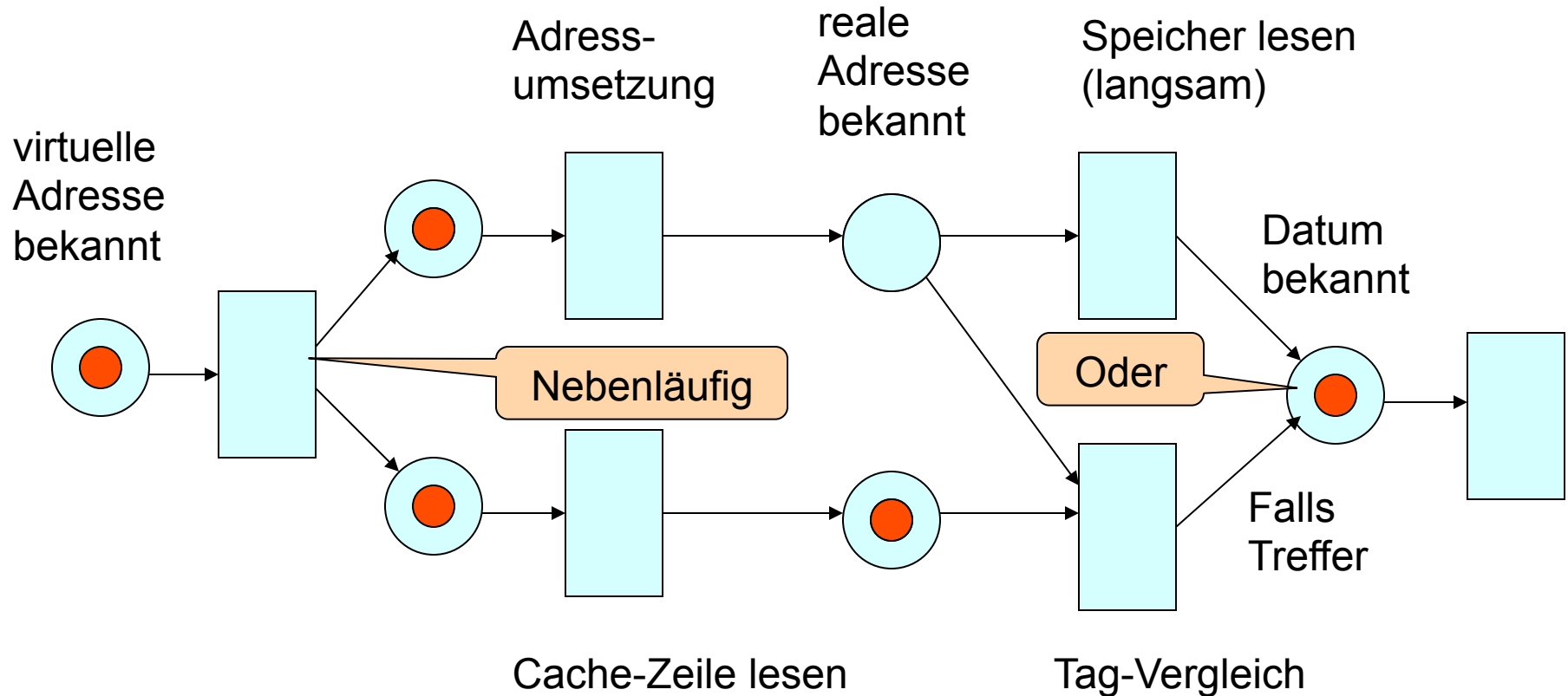
Modell des Ablaufs bei virtuellen Caches



Lesen des Caches und Adressumsetzung „parallel“

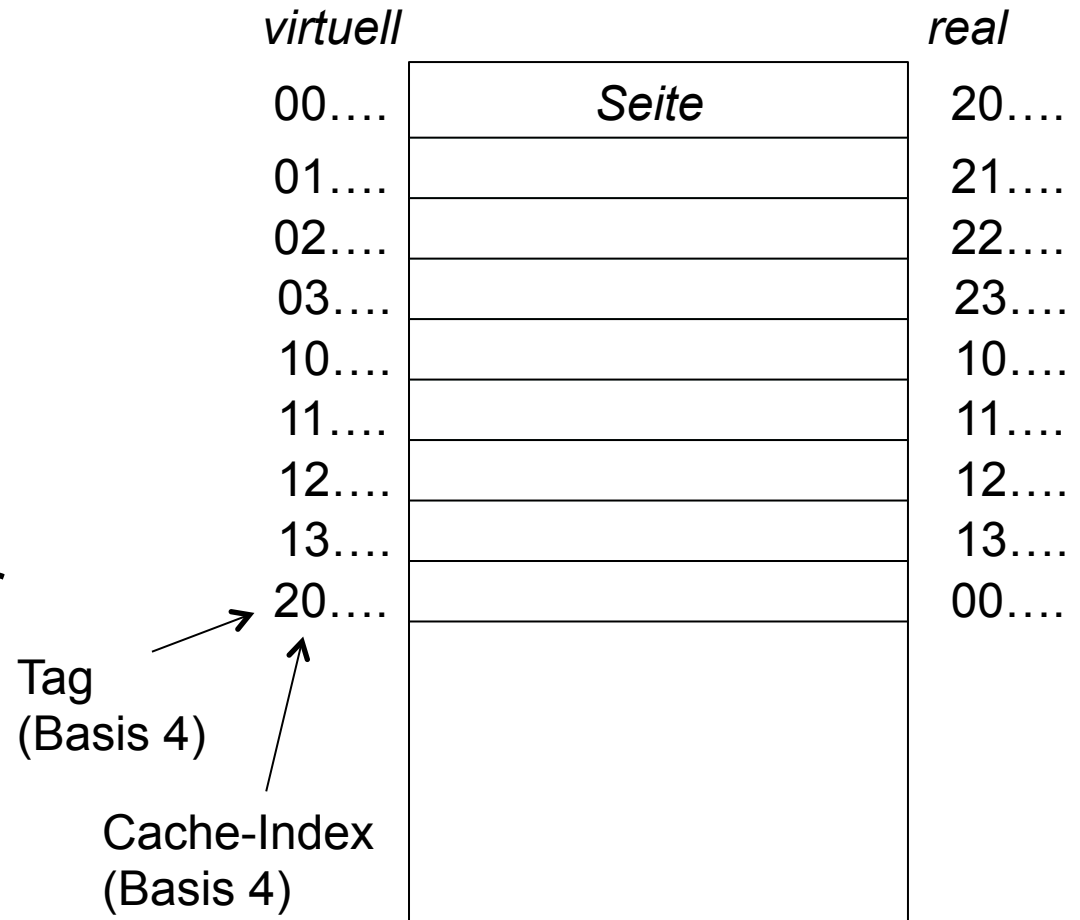
Mischform „*virtually indexed, real tagged*“

Indexwerte für virtuelle und reale Adressen identisch
☞ Zugriff auf realen Cache kann bereits während der Umrechnung auf reale Adressen erfolgen.



Mischform „*virtually indexed, real tagged*“ (2)

- Sei eine „*Meta-Page*“ der Speicherbereich, der zu einem bestimmten Tag-Feld gehört (unsere Bezeichnung)
- Innerhalb jeder „*Meta-Page*“ müssen virtueller und realer Index übereinstimmen.
- Meta-Page-Zuordnung beliebig

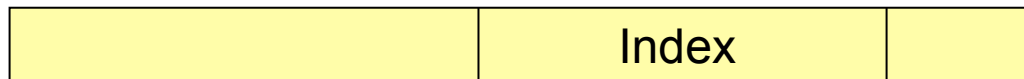


Mischform „*virtually indexed, real tagged*“ (3)

Nachteil:

- De facto: statt Seiten „*Meta-Pages*“ als Einheiten der Speicherzuteilung.
Sehr grob-granulare Speicherzuteilung.
- Beispiel: *direct mapped* \$, 256 kB=2¹⁸ B, 64 Bytes /Eintrag
☞ 12 Bits für den Index, Meta-Pages von 256 kB.

Seitennummer



Virtually indexed, virtually tagged

Fast, but there are some problems with this type

- Protection: page-level protection is checked as part of the virtual to physical address translation
- Process switches: the virtual addresses refer to different physical addresses, requiring the cache to be flushed, or to tag with additional bits for the process-identifier tag
- Aliases in virtual memory systems: antialiasing from HW or SW has to be supported
- I/O: I/O typically uses physical addresses and thus would require mapping to virtual addresses to interact with a virtual cache.

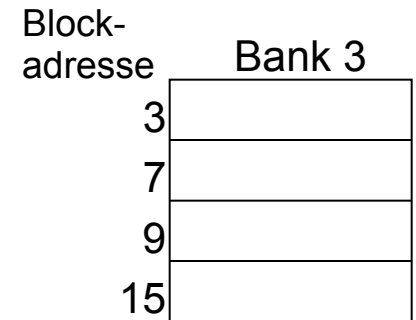
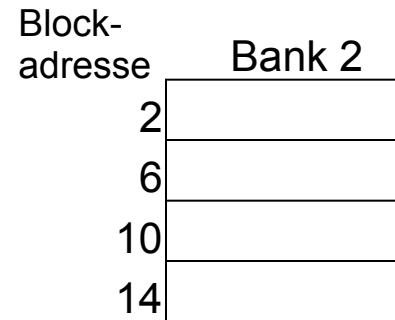
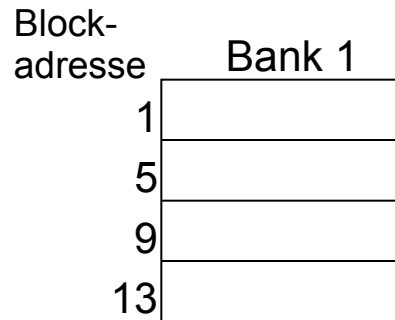
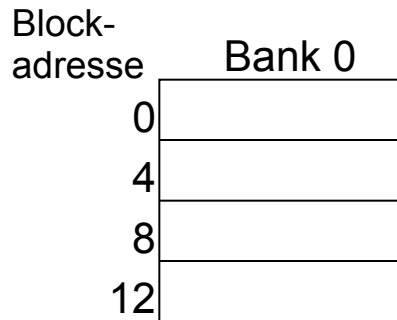
Unterschiedliche Kombinationsmöglichkeiten

Index/ Tag	Virtuell	Real
Virtuell	<i>Virtually indexed, virtually tagged (VIVT)</i> Schnell, Kohärenz- probleme	<i>Physically (real)-indexed, virtually tagged</i> Theoretisch möglich, aber sinnlos
Real	<i>Virtually indexed/ physically (real)- tagged (VIPT)</i> Mittelschnell	<i>Physically (real)-indexed, physically (real)-tagged (PIPT)</i> Langsam, keine Kohärenzprobleme

- Miss Rate
- Hit Time
- **Cache Bandwidth**
- Miss Penalty

Cache Bandwidth ↑: Multibanked Caches

Blöcke werden über mehrere Speicherbänke verteilt



- Miss Rate
- Hit Time
- **Cache Bandwidth**
- Miss Penalty

Cache Bandwidth ↑: Pipelined Cache Access

Cache Zugriff über mehrere Stufen der Pipeline verteilen

- ☞ Latenzzeiten des L1-Caches können größer sein als bei Zugriff in einem Takt (Pentium 4: 4 Zyklen)

Konsequenz mehrerer Pipelinestufen

- Größere Verzögerung für Sprungbefehle (*branch penalty*) bei falscher Sprungvorhersage
- Längere Verzögerung zwischen Laden und Verwenden von Operanden (*load delay*), siehe MIPS R4000!
- *Cache*-Latenz ändert sich nicht, aber sie wird versteckt.

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- ***Miss Penalty***

Miss Penalty ↓: Nonblocking Caches

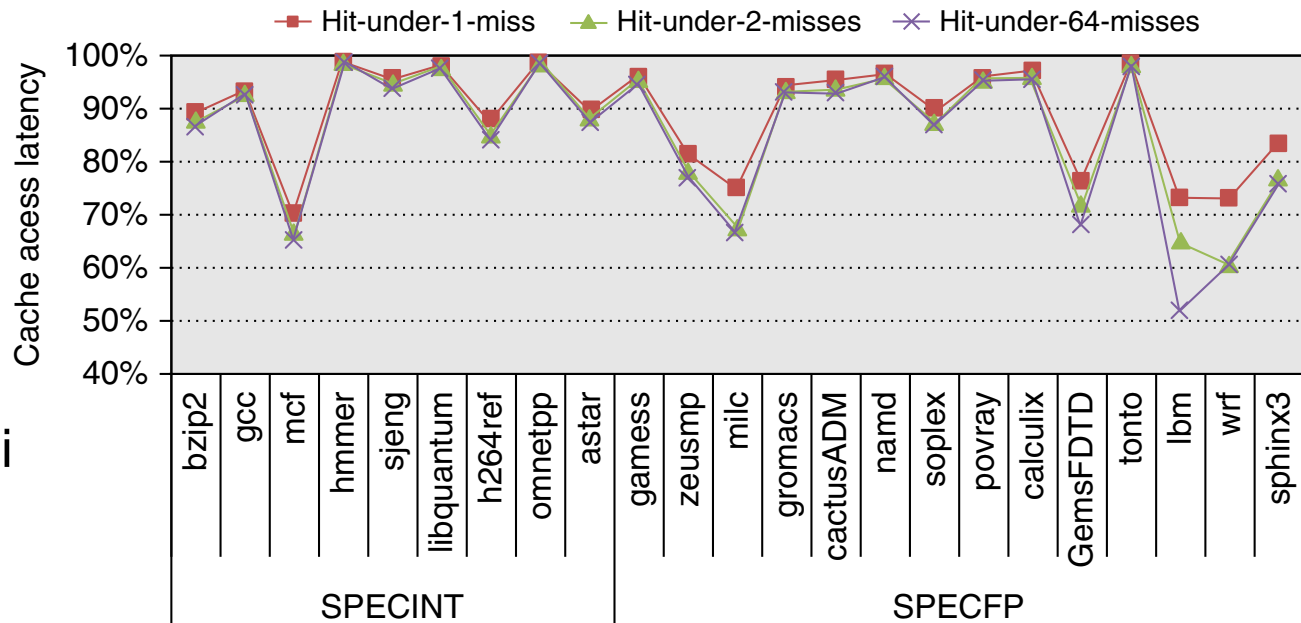
- Überlappen der Ausführung von Befehlen mit Operationen auf der Speicherhierarchie
- CPUs mit *out-of-order completion/execution*
 - ☞ Anhalten der CPU bei Fehlzugriff auf Cache nicht erforderlich (z.B. weiteren Befehl während Wartezeit holen)
- Notwendig: *Cache* kann trotz vorangegangenem (und ausstehenden) Fehlzugriffs nachfolgende erfolgreiche Zugriffe (*hits*) befriedigen
 - ☞ bezeichnet als “*hit under miss*”
- Auch: “*hit under multiple misses*” bzw. “*miss under miss*”

- Miss Rate
- Hit Time
- Cache Bandwidth
- Miss Penalty

Miss Penalty ↓: Nonblocking Caches (2)

Nicht-blockierende Caches „hit under (multiple) misses“:
Vorteil gegen blockierenden Cache (SPECINT/SPECFP)

- hit under 64 misses = 1 miss pro 64 Registern
- Verbesserung durch hit under multiple misses hauptsächlich bei FP-Benchmarks



© 2003 Elsevier Science

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- ***Miss Penalty***

Miss Penalty ↓: Zusammenfassung von Schreibpuffern

- Betrachte *write through Cache*
- Schreibzugriffe in (kleinem) Puffer gespeichert und bei Verfügbarkeit des Speicherinterfaces ausgeführt
Hier: Daten wortweise geschrieben!
- Annahme: Schreiben größerer Datenblöcke effizienter möglich
(trifft auf heutige Speicher in der Regel zu!)
- ☞ Schreibpuffer im *Cache* gruppieren, falls zusammenhängender Transferbereich entsteht

Miss Penalty ↓:

Zusammenfassung von Schreibpuffern (2)

Write buffer merging:

- Oben: ohne
- Unten: mit

Write address	V		V		V		V
100	1	Mem[100]	0		0		0
108	1	Mem[108]	0		0		0
116	1	Mem[116]	0		0		0
124	1	Mem[124]	0		0		0

Beachte:
Beispiel
extrem
optimistisch!

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Falls *write buffer merging* nicht vorgesehen,
ist Schreibpuffer auch nicht mehrere Worte breit!

© 2003 Elsevier Science

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- ***Miss Penalty***

Miss Penalty ↓:

Einführung mehrerer *Cache*-Ebenen

- Unabhängig von der CPU
- Fokus auf Schnittstelle *Cache*/Hauptspeicher

Dilemma: Leistungsfähigkeit des *Caches* steigern durch

- Beschleunigung des *Caches* (um mit CPU „mitzuhalten“)
[☞ *Cache* muss kleiner werden] oder
- Vergrößerung des *Caches* (um viele Anfragen an Speicher „befriedigen“ zu können) [☞ *Cache* langsamer]

☞ Eigentlich beide notwendig, aber unvereinbar!

Lösung: 2 *Cache*-Ebenen

- *1st-level Cache* (L1): schnell, klein, angepasst an CPU-Takt
- *2nd-level Cache* (L2): groß genug, viele Zugriffe zu puffern, kann langsamer sein

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- ***Miss Penalty***

Miss Penalty ↓: Einführung mehrerer ***Cache-Ebenen*** (2)

Leistungsbewertung jetzt komplexer:

Interaktion von L1 und L2-Cache muss berücksichtigt werden!

- (1) *Mittl. Zugriffszeit* = *Hit Time*(L1) + *Miss Rate*(L1) × *Miss Penalty* (L1)
mit Charakteristik des L2-Caches
- (2) *Miss Penalty*(L1) = *Hit Time*(L2) + *Miss Rate*(L2) × *Miss Penalty*(L2)
- (3) *Mittl. Zugriffszeit* = *Hit Time*(L1) + *Miss Rate*(L1) ×
(*Hit Time*(L2) + *Miss Rate*(L2) × *Miss Penalty*(L2))

Beachte:

- *Miss Rate*(L2) gemessen auf von L1 „übrigen“ Zugriffen
- Unterscheidung von lokaler und globaler Fehlzugriffsrate (lokal groß für L2-Cache, da nur auf Fehlzugriffen von L1)

Miss Penalty ↓: Einführung mehrerer *Cache*-Ebenen (3)

Parameter von 2-Ebenen-Cache-Systemen:

- Geschwindigkeit von L1-Cache beeinflusst CPU-Takt
- Geschwindigkeit von L2-Cache beeinflusst nur Verzögerung bei Fehlzugriff auf Daten in L1-Cache ☞ (relativ?) selten

☞ Viele Alternativen für L2, die bei L1 nicht anwendbar sind

Ziel: L2-Cache (wesentlich) größer als L1-Cache!

Miss Penalty ↓: Einführung mehrerer Cache-Ebenen (4)

Verhältnis von L1- zu L2-Cache-Inhalten:

- *Multilevel **inclusion*** (Daten in L1 immer in L2 vorhanden)
 - ☞ Konsistenz(prüfung) einfach!
 - Beachte: Setzt deutlich größeren L2-Cache voraus als s.u.
- *Multilevel **exclusion*** (Daten von L1 nie in L2 vorhanden)
 - L1-Fehlzugriff bewirkt Vertauschung von Blöcken zw. L1 u. L2
 - ☞ speicherökonomisch, falls L2 nur gering größer als L1 (z.B. AMD Athlon: L1 = 64kB, L2 = 256kB)

Miss Penalty ↓:

Critical Word First & Early Restart

CPU benötigt häufig nur einen Wert aus dem *Cache-Block*

☞ nicht warten, bis kompletter Block nachgeladen

2 Realisierungsmöglichkeiten:

■ *Critical Word First*

- Zuerst angefragtes Datenwort aus Speicher holen
- CPU-Ausführung sofort fortsetzen lassen
- Weitere Inhalte dann parallel zur CPU holen

■ *Early Restart*

- Daten in normaler Reihenfolge in Cache-Block laden
- CPU-Ausführung fortsetzen, sobald angefragtes Wort geladen

Vorteile nur bei großen Cache-Blöcken

Mit örtlicher Lokalität Wahrscheinlichkeit groß für Zugriff im aktuellen Block!

Miss Penalty ↓:

Priorisierung von Lese- über Schreibzugriffen

Idee: Lesezugriffe behandeln, bevor Schreiboperation abgeschlossen



Problem bei *write through*:

Wichtigste Optimierung: Schreibpuffer

☞ kann Wert enthalten, der von Lesezugriff angefragt ist

Lösungsmöglichkeiten:

- Warten bis Schreibpuffer leer (kontraproduktiv)
- Inhalte des Schreibpuffers prüfen und mit Lesezugriff fortfahren, falls keine Konflikte (üblich in *Desktop/Server*)

Vorgehen bei *write back*:

- „dirty“ Block in (kleinen) Puffer übernehmen
- Zuerst neue Daten lesen, dann „dirty“ Block schreiben

***Miss Penalty* ↓: *Prefetching* (durch HW oder SW)**

(deutsch etwa „Vorabruf“, „vorher abholen“)

Elemente in *Cache* laden, bevor sie gebraucht werden

- *Prefetching* versucht Speicherbandbreite zu verwenden, die anderweitig ungenutzt bliebe
- Kann mit „regulären“ Anfragen/Zugriffen interferieren!

Miss Penalty ↓: *Prefetching* (durch **HW** oder SW)

a) Hardware-*Prefetching* für Instruktionen/Daten

- Bei *cache miss*: angefragter Block → *Cache*, sequentieller Nachfolger → “*stream buffer*”
- Falls bei späterer Anfrage Block im *stream buffer*: ursprüngliche *Cache*-Anfrage verwerfen, aus *stream buffer* nachladen und neuen *prefetch* starten

Miss Penalty ↓: Prefetching (durch HW oder SW) (2)

b) Prefetching kontrolliert durch Compiler

- Verwendet spezielle Befehle moderner Prozessoren
- Möglich für Register und *Cache*
- *Prefetch*-Instruktionen dürfen keine Ausnahmen (insbes. Seitenfehler) erzeugen ☞ *non-faulting instructions*
- Cache darf nicht blockierend sein (damit Prefetch-Lade-vorgänge und weitere Arbeit der CPU parallel möglich)
- *Prefetching* erzeugt *Overhead* ☞ Kompromiss erforderlich (nur für die Daten *prefetch* Instruktionen erzeugen, die mit großer Wahrscheinlichkeit Fehlzugriffe erzeugen)
- In Alpha 21264 auch *prefetch* für Schreibzugriffe (falls angeforderter Block komplett geschrieben wird: *write hint* Befehl alloziert diesen im *Cache*, aber Daten nicht gelesen!)

- *Miss Rate*
- *Hit Time*
- *Cache Bandwidth*
- *Miss Penalty*

Zusammenfassung

Betrachten von Verfahren zur

- Reduktion der Rate fehlgeschlagener *Cache*-Zugriffe
- Reduktion der Zugriffszeit im Erfolgsfall
- Erhöhung der Speicherbandbreite
- Reduktion der Wartezyklen, die ein fehlgeschlagener Zugriff erzeugt