Rechnerstrukturen im SS2006

Caches und Cache-Kohärenz

Dr. Jie Tao

tao@ira.uka.de
http://itec.uka.de/~tao

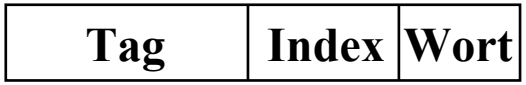
13. 07.2006

Überblick

- Cache-Funktionsweise
- Cache Leistung
- Ermittlung der Hits/Miss durch Berechnung
- Caches in MP-Systemen
  - Konsistenz und Kohärenz
    - Das MESI-Protokoll

Begrifflichkeit

- Cache Hierarchie (L1, L2, L3)
- Cache-Typ (I-Cache, D-Cache, Victim-Cache)
  - Victim-Cache: ein kleiner vollassoziativer Cache zum Speichern kürzlich ersetzter Blöcken
- Cache-Organisation
  - Direkt zugeordnet (direct-mapped)
  - Vollassoziativ (fully associative)
  - Satzassoziativ (n-set associative)



- Update-Strategie
  - Write-through, write-back
  - Write-allocation, no write-allocation

Aufgabe1: Cache-Funktionsweise

Ein Cache-Speicher werde durch 32 Bit adressiert und eine Kapazität von 16 Blöcken. Pro Block können 4 Datenbytes gespeichert werden. Die Verdrängungsstrategie sei LRU. Der Cache sei zu Beginn einer Programm-abarbeitung gelöscht, d.h. alle Blöcke sind unbesetzt.

Aufgabe1 (fort.)

- Wie viele Adressbits werden bei den Organisationsformen direkt abgebildet, 4-fach satzassoziativ und vollassoziativ jeweils für den Tag- und Indexteil benötigt? Wie breit ist die Wortadresse?
  - Es gilt: Länge (L) der Wortadresse = log_2(Blockkapazität), d.h. es werden bei 4 Datenbytes pro Block 2 Bit für die Wortadresse benötigt
  - Satzauswahl geschieht durch den Indexteil, d.h. $L_{index} = log_2(\#\ Sätze)$

Aufgabe1 (fort.)

- Für den Tag-Teil gilt somit $L_{Tag} = L_{Adresse} - L_{Index} - L_{Wort}$; der Adreßraum ist mit 32 Bit angegeben, wodurch sich folgende Aufteilungen ergeben:

Cache	#Sätze	#Blöcke/ Satz	Index Länge	Tag Länge
Direct-mapped	16	1	4	26
4-way s.a.	4	4	2	28
Fully assoc.	1	16	0	30

Aufgabe1 (fort.)

- Der Zugriff vom Programm auf einzelne Cache-Blöcke sei wie folgt (alle Adressen in hexadezimaler Schreibweise):

  1F296FFA, 378CF121, 1F296FFB, 378D1831,
  1F296FFC, 378D3F41, 1F296FFD, 378D6651,
  1F296FFE, 378D8D61, 1F296FFF, 378DB471,
  1F297000, 378DDB81, 1F297001, 378E0291,
  1F297002, 378E29A1, 1F297003, 378E50B1

Bestimmen Sie für alle drei Organisationsformen, bei welchen Zugriffen es sich um Cache-Treffer handelt. Protokollieren Sie die Belegung der einzelnen Cache-Blöcke und ermitteln Sie die Trefferraten.

Direct-mapped Cache

Adresse	Tag	Index	Treffer
1F296FFA	1F296F,11	1110	-
378CF121	378CF1,00	1000	-
1F296FFB	1F296F,11	1110	+
378D1831	378D18,00	1100	-
1F296FFC	1F296F,11	1111	-
378D3F41	378D3F,01	0000	-
1F296FFD	1F296F,11	1111	+
378D6651	378D66,01	0100	-
1F296FFE	1F296F,11	1111	+
378D8D61	378D8D,01	1000	-
1F296FFF	1F296F,11	1111	+
378DB471	378DB4,01	1100	-
1F297000	1F2970,00	0000	-
378DDB81	378DDB,10	0000	-
1F297001	1F2970,00	0000	-

6 Treffer
hit rate =
6/20 = 0,3

4-way Set-associative Cache

Adresse	Tag	Index	Treffer
1F296FFA	1F296FF	10	-
378CF121	378CF12	00	-
1F296FFB	1F296FF	10	+
378D1831	378D183	00	-
1F296FFC	1F296FF	11	-
378D3F41	378D3F4	00	-
1F296FFD	1F296FF	11	+
378D6651	378D665	00	-
1F296FFE	1F296FF	11	+
378D8D61	378D8D6	00	-
1F296FFF	1F296FF	11	+
378DB471	378DB47	00	-
1F297000	1F29700	00	-
378DDB81	378DDB8	00	-
1F297001	1F29700	00	+

7 Treffer
hit rate =
7/20 = 0,35

Fully Associative Cache

Adresse	Tag	Treffer
1F296FFA	1F296FF,10	-
378CF121	378CF12,00	-
1F296FFB	1F296FF,10	+
378D1831	378D183,00	-
1F296FFC	1F296FF,11	-
378D3F41	378D3F4,00	-
1F296FFD	1F296FF,11	+
378D6651	378D665,00	-
1F296FFE	1F296FF,11	+
378D8D61	378D8D6,00	-
1F296FFF	1F296FF,11	+
378DB471	378DB47,00	-
1F297000	1F29700,00	-
378DDB81	378DDB8,00	-
1F297001	1F29700,00	+

7 Treffer
hit rate =
7/20 = 0,35

Leistung von Caches

- Cache charakterisiert durch
  - Hit Rate r_H
  - Miss Rate $r_M = 1 - r_H$
  - Zugriffszeit bei Hit: t_H
  - Zugriffszeit bei Miss: t_{mem}
- Mittlere Zugriffszeit (nur L1 Cache):

$$t_a = r_H * t_H + r_M * t_{mem}$$

- Hierarchie fließt in jeweiligen Miss-Zweit ein
  - z. B Hierarchie mit L1 und L2

$$t_a = r_{H1} * t_{L1} + r_{M1} * (r_{H2} * t_{L2} + r_{M2} * t_{mem})$$

Aufgabe2: Cache-Leistung

- Ein System verfüge über eine Speicherhierarchie mit L1-Cache. Für den Cache gelte t_{L1}=4ns und r_H=0,75. Entsprechend betrage die Zugriffszeit für den Hauptspeicher 100ns. Berechnen Sie die mittlere Zugriffszeit t_a

$$t_a = r_H * t_{L1} + r_M * t_{mem} = r_H * t_{L1} + r_M * t_{mem}$$

$$= 0{,}75 * 4ns + (1\text{-}0{,}75) * 100ns$$

$$= 28ns$$

Aufgabe2 (fort.)

- Ergänzen Sie das System um einen L2-Cache für welche gelte, dass t_{L2}=20ns und r_{H2}=0,8.
  Berechnen Sie die mittlere Zugriffszeit für das neue System

$$t_a = r_{H1} * t_{L1} + r_{M1} * (r_{H2} * t_{L2} + r_{M2} * t_{mem})$$
$$= 0{,}75 * 4 + (1\text{-}0{,}75) * (0{,}8*20+0{,}2*100)\ ns$$
$$= 12\ ns$$

Aufgabe2 (fort.)

- Basierend auf den in der Zentralübung gegebenen Formeln: Wie sähe die Berechnung für t_a für ein L3-Cache-System aus?

$$t_a = r_{H1} * t_{L1} + r_{M1} * (r_{H2} * t_{L2} + r_{M2} * (r_{H3} * t_{H3} + r_{M3} * t_{mem}))$$

Aufgabe2 (fort.)

- Kostengründe sprechen gegen den Aufbau des gesamten Hauptspeichers aus schnellen Cache-Speichern. Welcher technische Grund spricht dagegen?

Die endliche Geschwindigkeit von Elektronen in leitfähigem Material setzt der Größe schneller Speicher eine natürliche Grenze

Ermittlung der Hits/Miss durch Berechnung

- Definition
  - Reuse Distance
    - Anzahl der unterschiedlichen Hauptspeicherblöcke zwischen zwei Zugriffen auf denselben Hauptspeicherblock
  - Set Reuse Distance
    - Anzahl der in denselben Cache-Satz geladenen unterschiedlichen Hauptspeicherblöcke zwischen zwei Zugriffen auf denselben Hauptspeicherblock
- Beispiel

Der Cache hat vier Sätze zu jeweils vier Zeile mit 32 Bytes

Zugriffe: 0x0080, 0x0040, 0x0140, 0x0100, 0x01C0, 0x0040, 0x0100

RD 3, SRD 2

- Berechnung
  - Hit/Miss (LRU)
    - SRD ≥ Assoziativität → Miss
    - SRD < Assoziativität → Hit

Aufgabe3: Leistung und Funktionsweise

- Die Cachehierarchie eines Itanium2 Prozessors besteht aus drei Cache-Ebenen, die all durch 64 Bit adressiert sind. Der L1D (Daten-Cache) hat eine Größe von 16 Kbyte, ist 4-fach assoziative und hat eine Cache-Zeilengröße von 64 Byte. Die Zugriffszeit bei Hit von L1D beträgt nur einen CPU-Zyklus. Der L2-Cache weist eine Größe von 256 Kbyte auf, ist 8-fach assoziative und besitzt eine Cache-Zeilengröße von 128 Byte. Die Zugriffszeit bei Hit von L2 beträgt 5--7 Zyklen. Der L3-Cache besitzt eine Größe von 3 MByte, ist 12-fach assoziative und hat eine Cache-Zeilengröße von 128 Byte. Die Zugriffszeit bei Hit von L3 beträgt 14--16 Zyklen.

Aufgabe3 (fort.)

- Wie viele Adressbit werden bei allen drei Caches jeweils für den Tag- und Indexteil benötigt? Wie breit ist die Wortadresse?
- Es wurden bei der Ausführung eines Programms 16560 L1D-Hits aus gesamten 18000 Speicherzugriffen ermittelt. Aus den Misses wurden 80% mit 5-Zyklen, 5% mit 7-Zyklen, 5% mit 14-Zyklen und der Rest mit 26-Zyklen erfüllt.
  - Berechnen Sie Die Missrate von L2 und L3.
  - Berechnen Sie die Mittlere Zugriffszeit für das Cache-System

Lösung

- Die mit 5 (80%) und 7 (5%) Zyklen erfüllten L1-Misses seien L2-Hits
  - $r_{H2} = 0{,}80 + 0{,}05 = 0{,}85$; $r_{M2}=1-0{,}85=0{,}15$
- $$r_{H3} = \frac{\#\text{ hits-in-L3}}{\#\text{ Zugriffe-auf-L3}} = \frac{5\%*(18000-16560)}{\text{Misses des L2}}$$

$$= \frac{5\%*(18000-16560)}{r_{M2}*(18000-16560)} = \frac{5\%}{0{,}15} = 0{,}33$$

$$r_{M3} = 1 - r_{H3} = 0{,}67$$

Lösung: Mittlere Zugriffszeit

$$t_a = r_{H1} * t_{L1} + r_{M1} * t_a'$$

$t_a' = 0{,}8*5+0{,}05*7+0{,}05*14+0{,}1*26=7{,}65$ Zyklen

$t_a = 0{,}92*1+(1-0{,}92)*7{,}65=1{,}532$ Zyklen

Aufgabe3 (fort.)

- Der Zugriff vom Programm sei wie folgt (alle Adressen in hexadezimaler Schreibweise):

  1F296FFA, 378CF121, 378D1831, 378CF131,

  002FFFCF, 13C0F10C, 263D0FDB, 1F297FE1,

  1C6654C3, 24359FD5, 1F297103, 1F296FFB

Bestimmen Sie ob bei dem letzten Zugriff es sich um Cache-Treffer (L1D und L2) handelt. Berechnen Sie dabei die Reuse Distance und die Set Reuse Distance.

Lösung

- Angegeben
  - Adressraum: 64 Bit
  - L1D: 16K, Line-size 64 Byte, 4-fach → Wort Länge: 6Bit
  - L2: 256K, Line-size 128 Byte, 8-fach → Wort Länge: 7Bit
- Es gilt:

$$\#\text{Sätze} = \frac{\text{Cache-Größe}}{\text{Blockkapazität} * \text{Assoziativität}}$$

$$\#\text{Sätze}_{L1D} = \frac{16 * 1024}{64 * 4} = 64; \quad \#\text{Sätze}_{L2} = \frac{256 * 1024}{128 * 8} = 256$$

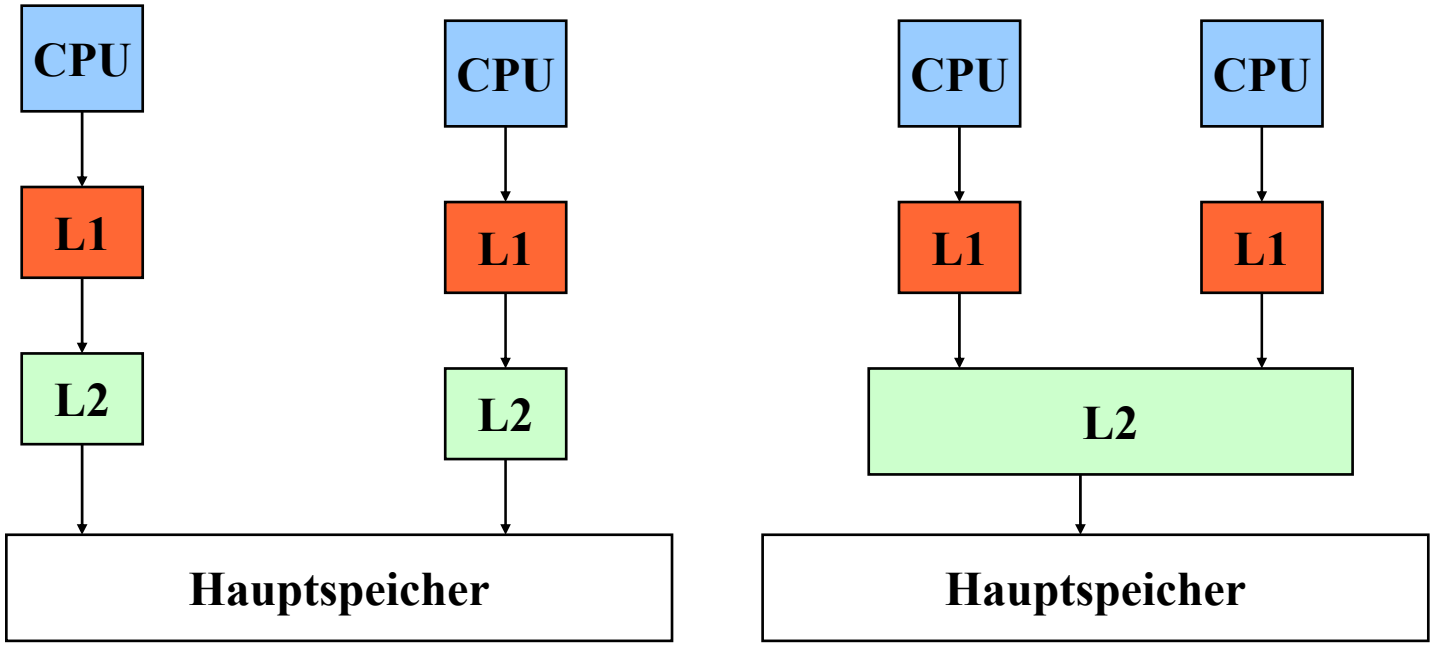
L1D Index: 6 Bit; L2 Index: 8

Lösung (fort.)

Adresse	Tag L1D	Index L1D	Tag L2	Index L2
1F296FFA	1F296	63 (3F)	1F29,0	DF
378CF121	378CF	4	378C,1	E2
378D1831	378D1	32	378D,0	30
378CF131	378CF	4	378C,1	E2
002FFFCF	002FF	63	0027,1	FF
13C0F10C	13C0F	4	13C0,1	E2
263D0FDB	263D0	63	263D,0	1F
1F297FE1	1F297	63	1F29,0	FF
1C6654C3	1C665	19	1C66,0	A9
24359FD5	24359	63	2435,1	3E
1F297103	1F297	4	1F29,0	E2
1F296FFB	1F296	63	1F29,0	DF

Die Reuse Distance des letzten Zugriffs ist 9, hinsichtlich beiden Caches. Die Set Reuse Distance ist 4 bei L1D und 0 bei L2. Es handelt sich um einen Miss in L1D und einen Hit in L2

Caches in MP-Systemen



- CPUs operieren auf lokalen Kopien
- Probleme
  - Wahrung von Konsistenz
  - Erzeugung von Cache-Kohärenz

Konsistenz und Kohärenz

- Konsistenz
  - Ein System ist konsistent, wenn alle Kopien eines Datenworte im Hauptspeicher und den verschiedenen Cache-Speichern identisch ist
- Konsistenzproblem
  - Lesen veralteter Daten bei mehreren Bus-Mastern (z.B. CPU und DMA-Controller)
- Cache-Kohärenz
  - Prozessor muss auf aktuelle Inhalte im Cache wie im Hauptspeicher zugreifen können

Bus-Snooping (1)

- Beobachtung des Busses hinsichtlich Speicherzugriffe anderer Bus-Master
- Erfordert gemeinsamen, globalen Bus: Buszugriffe jedes Masters können von allen anderen Teilnehmern gesehen werden
- Snooping wird durch Cache-Steuerung vorgenommen
- Ungültigmachen (write invalidate) oder aktualisieren (write update) des betroffenen Cache-Eintrags

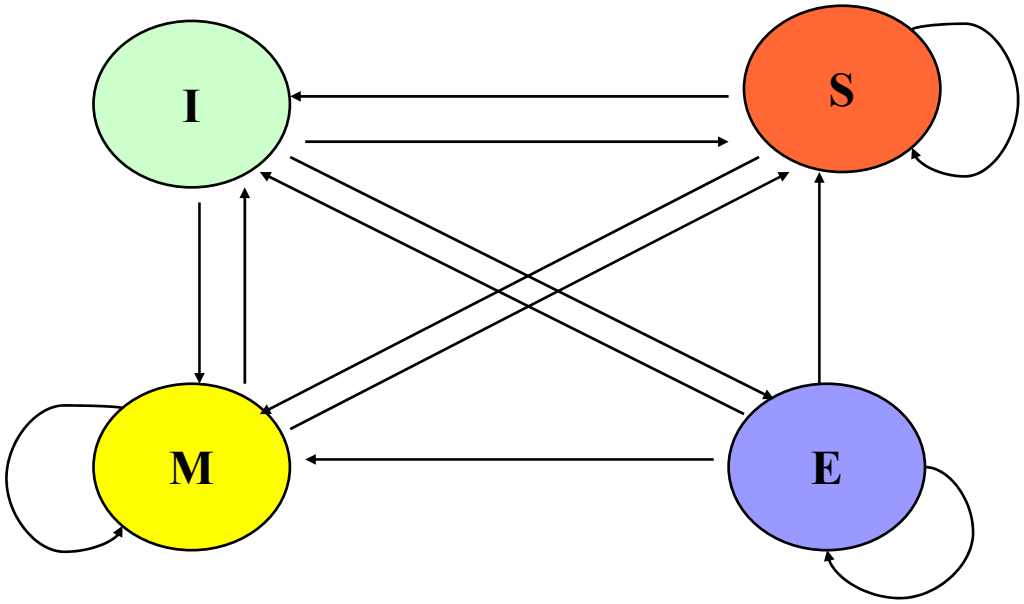
Bus-Snooping (2)

- Write Invalidate
  - Schreibbereite CPU übernimmt Bus und signalisiert write invalidate
  - Alle mitschnüffelnden Caches invalidieren die entsprechende Cache-Zeile
  - CPU schreibt Datum (write-through)
  - Lesezugriff einer anderen CPU erzeugt Cache-Miss und erzwingt Laden der aktualisierten Daten
- Write Update
  - Schreibbereite CPU übernimmt Bus und überträgt Daten über den gemeinsamen Bus (broadcast)
  - Alle mitschnüffelnden Caches aktualisieren ihre lokale Kopie

Das MESI-Protokoll

- MESI: Zustandsautomat mit 4 Zuständen
- Cache-Zeile befindet sich in einem dieser Zustände
- Erweiterung um 2 Zustandsbits
- Zustandsübergänge ausgelöst durch lokale und entfernte (durch Snooping festgestellte) Zugriffe
- Notwendige Hardwareunterstützung durch Snoop-Logik und Steuersignale

Zustände des MESI-Protokolls



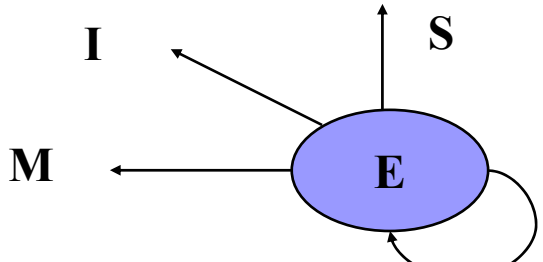
- I: invalid
- E: exclusive unmodified
- S: shared unmodified
- M: modified

Zustände des MESI-Protokolls

- Invalid
  - Betrachtete Zeile ist ungültig
  - Schreib- oder Lesezugriffe veranlassen Einladen des entsprechenden Speicherblocks in die Cache-Zeile
  - Bei Lesezugriff: Anzeige durch andere CacheController, ob Block bereits bei ihnen gespeichert ist (shared read miss) oder nicht (exclusive read miss)
  - Bei Schreibzugriff: Mitteilung an andere Cache-Controller, ob Block geschrieben werden soll (write miss)
  - Wechsel der Cache-Zeile in entsprechenden Zustand (shared, exclusive, modified)

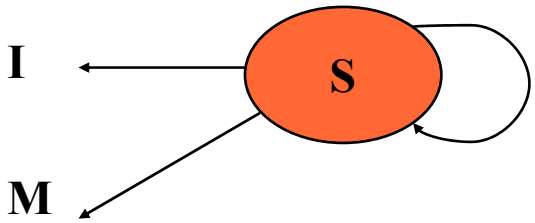
Zustände des MESI-Protokolls

- Exclusive
  - Betrachtete Zeile enthält exklusive, unveränderte Kopie des entsprechenden Speicherblocks
  - Bei Lesezugriff (read-hit) bleibt Zustand erhalten
  - Bei Lesezugriff durch anderen Prozessor (snoop-hit on read) Wechsel nach *shared*
  - Bei Schreibzugriff
    - write-hit: Wechsel nach *modified*
    - snoop-hit on write durch anderen Prozessor: Wechsel nach *invalid*



Zustände des MESI-Protokolls

- Shared
  - Speicherblock existiert als unmodifizierte Kopie in der betrachteten lokalen Cache und anderen Caches
  - Bei Lesezugriff (read-hit) bleibt Zustand erhalten
  - Bei Schreibzugriff
    - write-hit: Wechsel nach *modified* und Auslösen einer Invalidierung der anderen Cache-Kopien
    - snoop-hit on write durch anderen Prozessor: Wechsel nach *invalid*



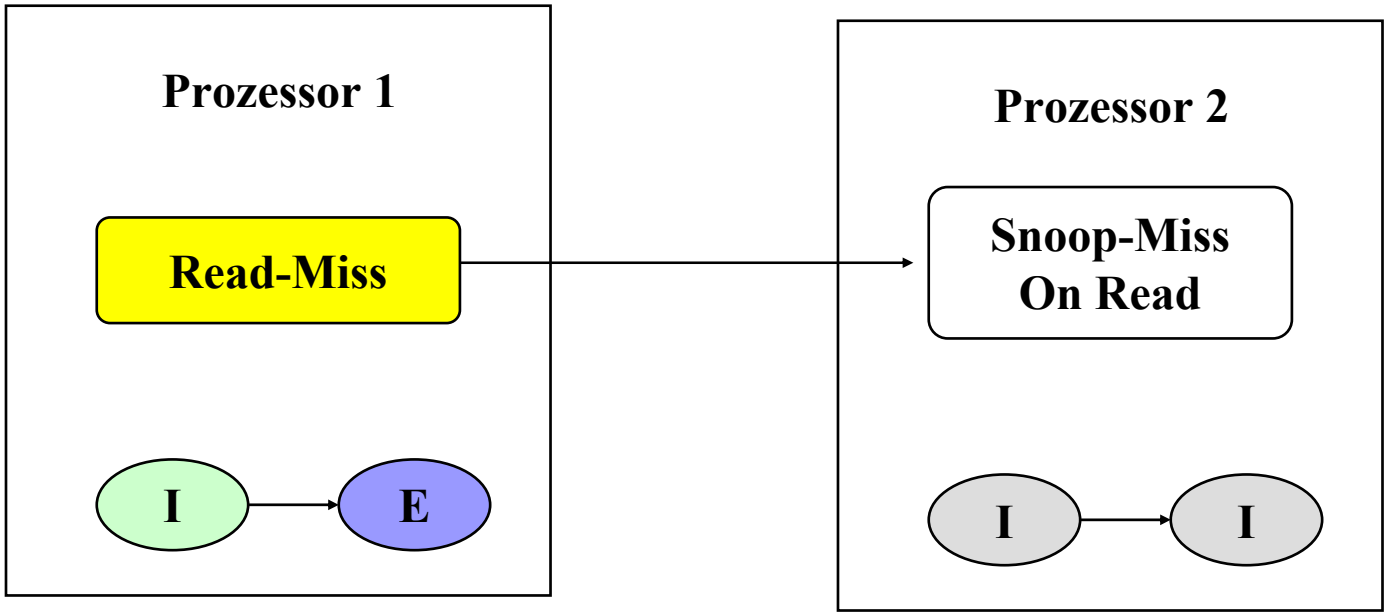
Zustände des MESI-Protokolls

- Modified
  - Betrachtete Zeile enthält exklusive, veränderte Kopie des entsprechenden Speicherblocks (dirty line)
  - Datum im Speicher veraltet, Zugriffe anderer Prozessoren überwachen, ggf. unterbrechen und Zeile zurückschreiben
  - Schreib-/Lesezugriff bleibt Zustand erhalten
  - Lesezugriff (snoop-hit on read) Wechsel nach *shared*
  - Schreibzugriff (snoop-hit on write) durch anderen Prozessor Wechsel nach *invalid*

MESI-Steuersignale

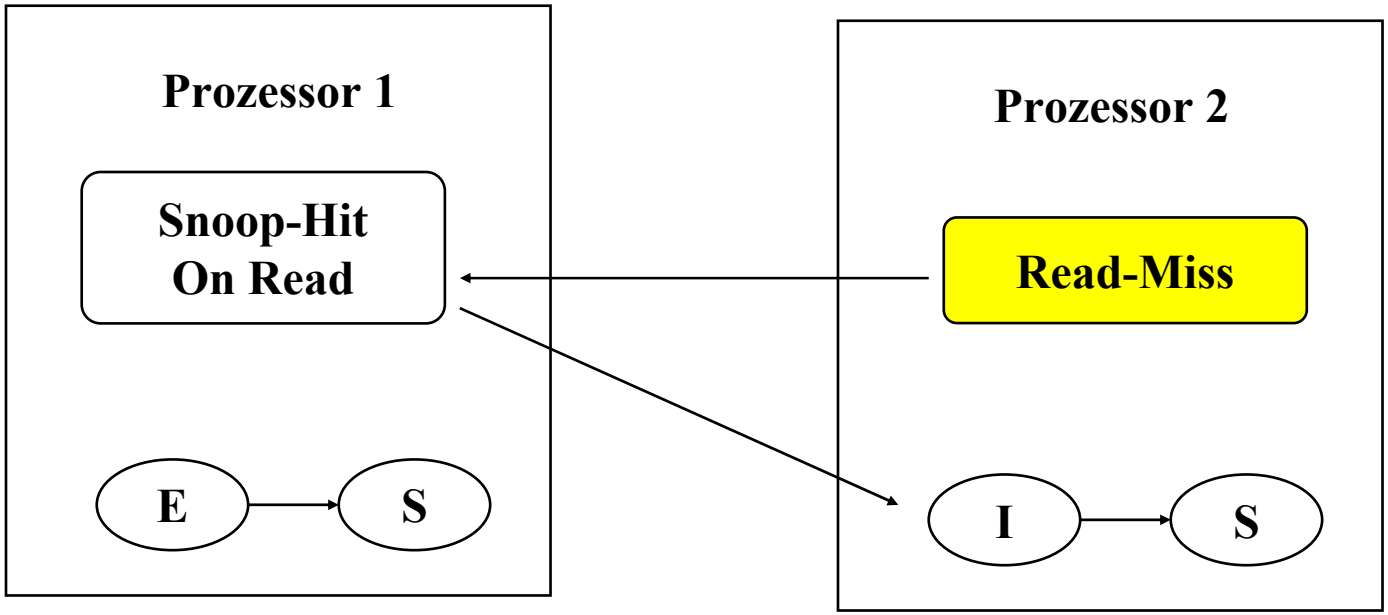
- Signale dienen zur Verständigung der Caches untereinander
- Invalid-Signal
  - Invalidierung von Einträgen in anderen Caches
- Shared-Signal
  - Anzeige, anderer Caches, ob zu ladender Block bereits als Kopie vorhanden ist
- Retry-Signal
  - Aufforderung an einen anderen Prozessor, das Laden eines Blocks abzubrechen
  - Laden wird nach Rückschreiben des Datums durch unterbrechenden Prozessor fortgesetzt

MESI-Beispiel: 2-Prozessorsystem



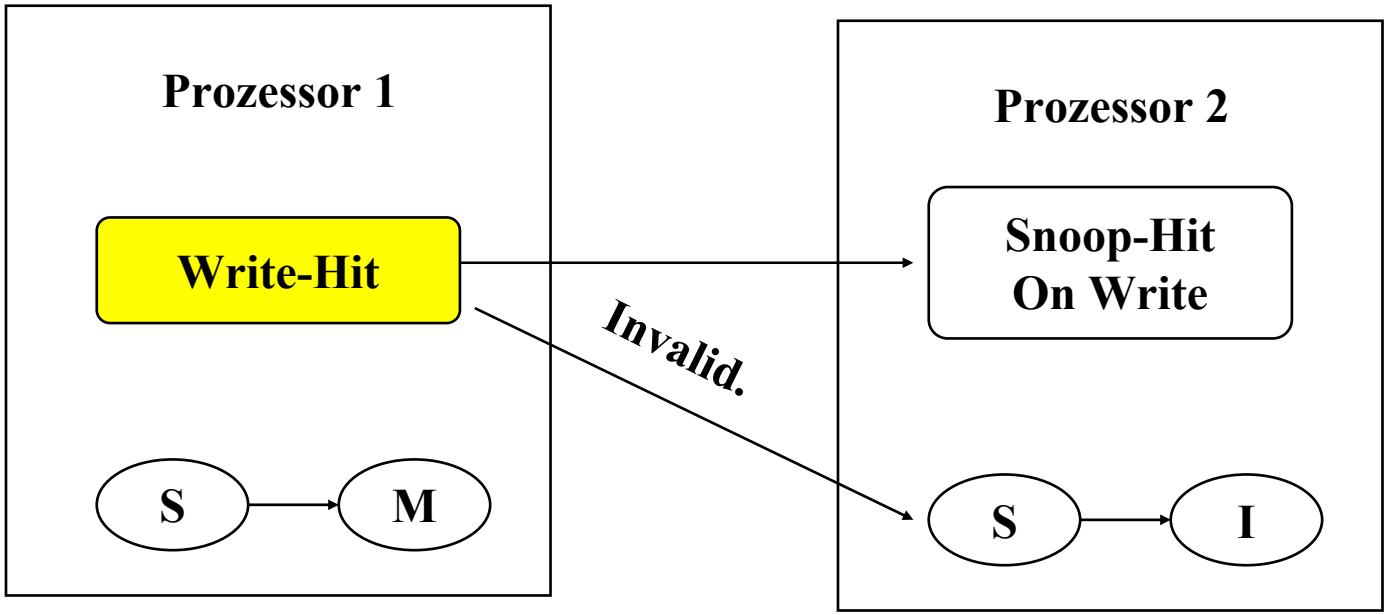
- Prozessor 1 liest Block erstmals ein (von Hauptspeicher)
- Block liegt nicht im Cache von Prozessor 2

MESI-Beispiel: 2-Prozessorsystem



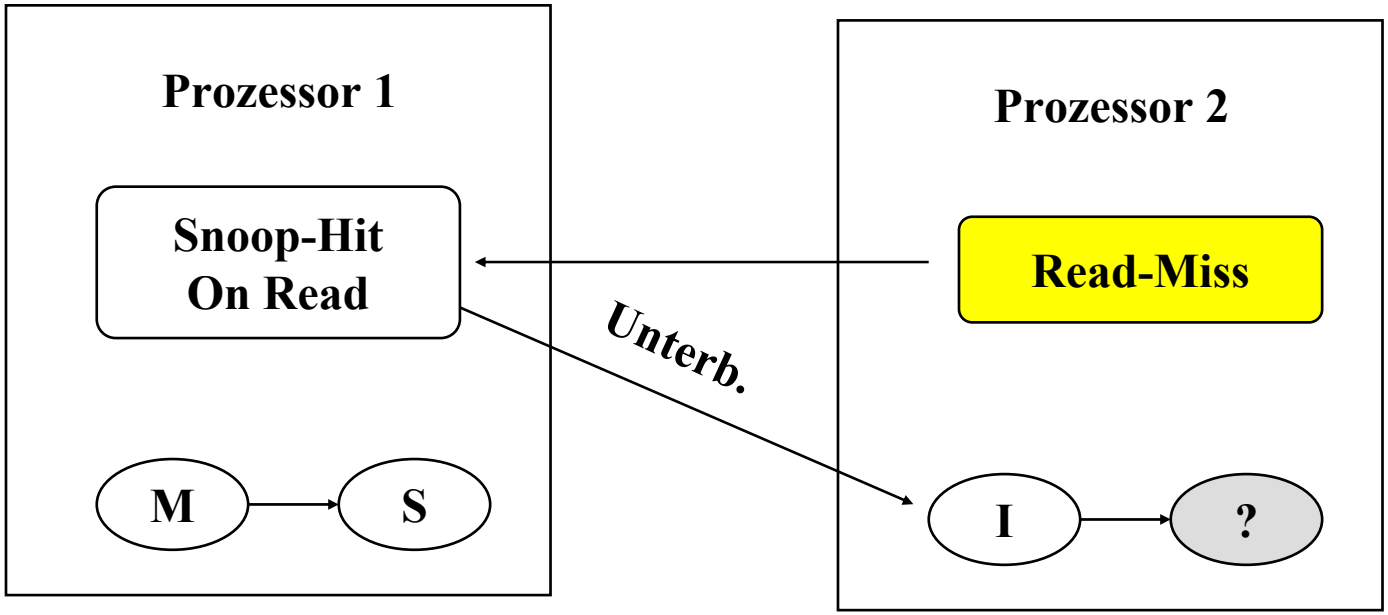
- Prozessor 2 liest Block erstmals ein (von Hauptspeicher)
- Block liegt bereits im Cache von Prozessor 1

MESI-Beispiel: 2-Prozessorsystem



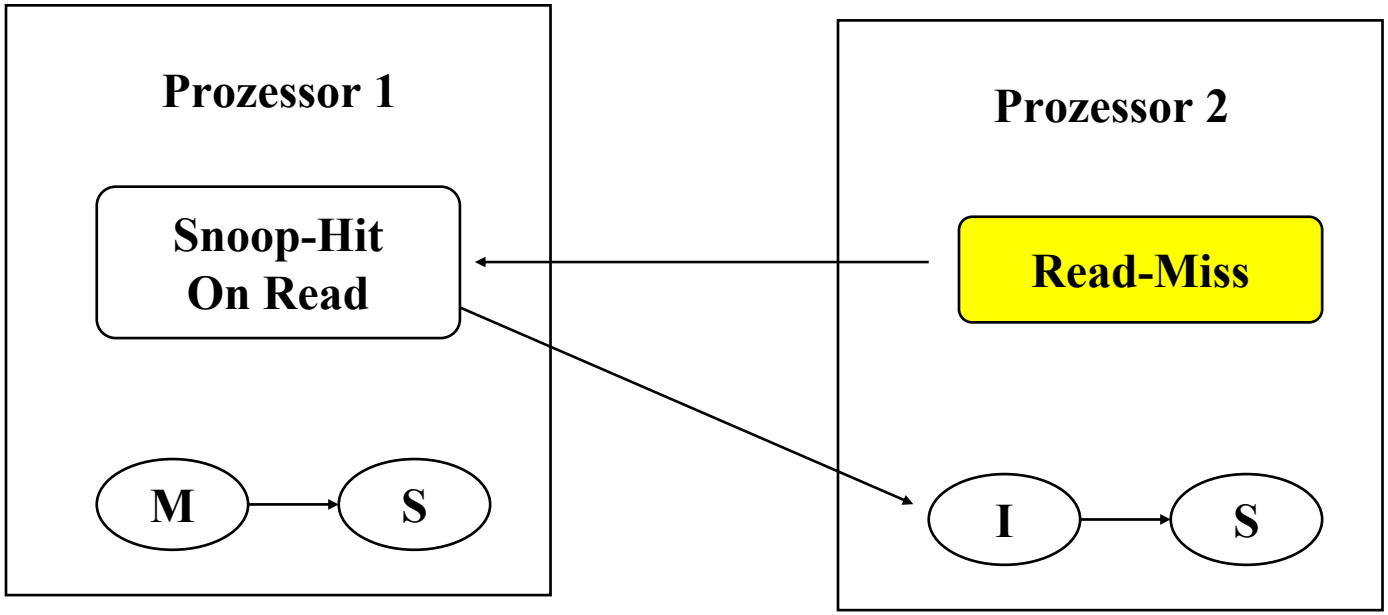
- Prozessor 1 schreibt modifizierten Block in Cache
- Block im Cache von Prozessor 2 wird invalidiert

MESI-Beispiel: 2-Prozessorsystem



- Prozessor 2 liest Block neu ein
- Prozessor 1 hält lokale, veränderte Kopie
- Unterbrechen des Lesevorgangs und Rückschreiben der Daten in Hauptspeicher

MESI-Beispiel: 2-Prozessorsystem



- Prozessor 2 startet neuen Leseversuch
- Block liegt bereits im Cache von Prozessor 1

Aufgabe4: MESI-Protokoll

Ein Zweiprozessor-System sei speichergekoppelt. Die Caches haben je eine Größe von drei Cache-Lines, welche je genau ein Speicherwort aufnehmen können. Die Füllung des Caches erfolgt von der niedrigsten Cache-Line aufwärts, sofern noch freie Lines zur Verfügung stehen. Andernfalls wird per LRU-Strategie verdrängt. Als Cache-Kohärenz-protokoll komme MESI zum Einsatz.

Vervollständigen Sie nachfolgende Tabelle; geben Sie pro Cache-Line jeweils Inhalt und MESI-Zustand an.

Prozessor	Aktion	Prozessor/Cache 1			Prozessor/Cache 2		
		Line 1	Line 2	Line 3	Line 1	Line 2	Line3
-	**(init)**	**E/8**	**E/12**	**I/-**	**E/6**	**I/-**	**I/-**
2	**rd 10**						
1	**wr 8**						
1	**rd 10**						
2	**rd 8**						
1	**wr 8**						
1	**wr 8**						
1	**rd 18**						
2	**wr 10**						
2	**wr 18**						

Prozessor	Aktion	Prozessor/Cache 1			Prozessor/Cache 2		
		Line 1	Line 2	Line 3	Line 1	Line 2	Line3
-	(init)	E/8	E/12	I/-	E/6	I/-	I/-
2	rd 10					E/10	
1	wr 8	M/8					
1	rd 10			S/10			
2	rd 8	S/8					S/8
1	wr 8	M/8					I/-
1	wr 8	M/8					
1	rd 18		E/18				
2	wr 10			I/10		M/10	
2	wr 18		I/-		M/18		