

# Rechnerarchitektur SS 2014

## Cachekohärenz

Michael Engel

TU Dortmund, Fakultät für Informatik

Teilweise basierend auf Material von Gernot A. Fink und R. Yahyapour

10. Juni 2014

# Speicher in MP-Systemen

## Zentrales Problem vom Mehrprozessorsystemen: Zugriff auf gemeinsamen Speicher

- ▶ Heutige Prozessoren erzeugen hohe Anforderungen an Speicherbandbreite
  - ⇒ Kann durch Verwendung von Caches (ggf. mit mehreren Ebenen) *deutlich* reduziert werden! Wieso?
- ✓ Durch Verwendung von Caches können auch *mehrere* Prozessoren (Haupt-)Speicher gemeinsam verwenden
- ▶ Seit 1980er viele kleine Mehrprozessorsysteme ( $<16$ ) auf der Basis von Mikroprozessoren entworfen
  - Gemeinsamer Speicher
  - Bus verbindet Speicher und alle Prozessoren (bzw. Caches)
  - ⇒ kosteneffektive Architektur
- ▶ Hardware-Realisierung
  - Früher: 1 Prozessor + Cache auf einem "Board"
  - Später: mehrere Prozessoren ( $\geq 4$ ) pro Board
  - Heute auch: mehrere Prozessoren auf einem Chip ( $\hat{=}$  Multi-Cores)

## Speicher in MP-Systemen 2

- ▶ Kleine MPs mit gemeinsamem Speicher (*shared*) unterstützen Caching für:
  - Private Daten (nur von einem Prozessor verwendet)
  - *shared data*, d.h. von mehreren Prozessoren gemeinsam verwendeten Daten ⇒ dienen zur Kommunikation
- ▶ Caching von privaten Daten
  - Inhalte in Cache transferiert
  - ✓ reduziert Zugriffszeit und Bandbreitenanforderung
  - ✓ Programmverhalten unverändert, da keine anderen Prozessoren Daten benützen
- ▶ Caching von "geshareten" Daten
  - Daten können in *mehreren* Caches repliziert werden/sein
  - ✓ reduziert neben Latenz & Bandbreitenanforderung Busbelastung bei gleichzeitigem Zugriff mehrerer Prozessoren
  - ⊘ Replikation erzeugt Problem der **Cache-Kohärenz!**

# Konsistenz & Kohärenz von Speicher

**Grundproblem:** Mehrere Prozessoren können *unterschiedliche* Sicht auf Speicher/Cache haben!

**Beispiel:**

Zeit	Ereignis	Cache CPU A	Cache CPU B	Speicher für Zelle X
0				1
1	CPU A liest X	1		1
2	CPU B liest X	1	1	1
3	CPU A schreibt Wert 0 in X	0	1	0

**Hier:** Betrachten *write-through* Cache


Wie mit *write-back*?

# Konsistenz & Kohärenz von Speicher 2

## Informeller Formulierungsversuch:

*Speichersystem ist kohärent, wenn Leseoperationen einer Zelle den letzten zuvor in diese Zelle geschriebenen Wert liefern.*

**Problem:** Definition ist zu vereinfachend und vage: *Wie Reihenfolge in MPs definieren?*

 Betrachtet werden müssen zwei Aspekte:

1. Welcher Wert wird von Leseoperation geliefert?  
⇒ Kohärenz
2. Wann wird ein geschriebener Wert von einer Leseoperation geliefert?  
⇒ Konsistenz

⇒ Betrachten zunächst (einfachere) Kohärenz

# Cache-Kohärenz

Speichersystem ist kohärent, wenn:

1. Lesen einer Zelle X durch Prozessor P nach Schreiben dieser Zelle durch P ...
    - immer geschriebenen Wert liefert, ...
    - sofern zwischen beiden Operationen kein Schreiben eines anderen Prozessors erfolgt ist.
- ⇒ Eigenschaft erhält Programmreihenfolge (bezogen auf einen Prozessor!), sollte insbes. für Uni-Prozessoren erfüllt sein.

## Cache-Kohärenz 2

(Speichersystem ist kohärent, wenn:)

2. Lesen einer Zelle X durch Prozessor P nach Schreiben von Zelle X durch anderen Prozessor P' ...
    - immer geschriebenen Wert liefert, ...
    - sofern zwischen beiden Operationen *hinreichend viel Zeit* vergeht und ...
    - kein Schreiben eines anderen Prozessors erfolgt ist.
- ⇒ Zentraler Aspekt der Kohärenz

**Beachte:** Falls beliebig lange Lesen alter Daten möglich  
⇒ Speichersystem ist klar inkohärent!

## Cache-Kohärenz 3

(Speichersystem ist kohärent, wenn:)

3. Schreiboperationen in dieselbe Zelle serialisiert werden, d.h. Schreibvorgänge verschiedener Prozessoren in dieselbe Zelle werden in derselben Reihenfolge von allen anderen Prozessoren gesehen.

**Beachte:** Beachte: Reihenfolge selbst ist *unspezifiziert*, muss nur für alle "Beobachter" gleich sein!

**Beispiel:** Zuerst 1, dann 2 in Zelle X geschrieben:

Kein Prozessor kann zuerst 2 und dann 1 lesen!

⇒ Alle Prozessoren, die zuletzt geschriebenen Wert einer Zelle anfordern, erhalten *denselben* Wert!



## Cache-Kohärenz 4

- ▶ Betrachtete Eigenschaften (1.-3.) sind hinreichend für Sicherstellung von Kohärenz
- ▶ Außerdem wichtig: Wann wird geschriebener Wert von anderen Prozessoren gesehen?  
(... sofern ... *hinreichend viel Zeit* vergeht ...)

**Beachte:** Leseoperation kann von anderem Prozessor geschriebenen Wert nie instantan sehen!

- ▶ Daten haben Prozessor ggf. noch nicht verlassen
  - ▶ Durchlaufen der Speicherhierarchie erfordert Zeit
- ⇒ **Konsistenz** definiert, *wann* geschriebener Wert von lesenden Prozessoren gesehen werden muss!  
(behandelt aber nicht absolute Zeiten, sondern *Ordnung* von Schreib- und Lesevorgängen)

## Cache-Kohärenz 5

- ▶ Kohärenz und Konsistenz sind *komplementär*:
  - Kohärenz** definiert Verhalten von Schreib-Leseoperationen auf *derselben* Speicherzelle
  - Konsistenz** definiert Verhalten von Schreib-Lese-Operationen bezogen auf Zugriffe von *anderen* Speicherzellen
- ▶ Nehmen zunächst an:
  - Schreiboperation nicht beendet bevor nicht alle Prozessoren Effekt gesehen haben
  - Prozessoren ändern Reihenfolge von Schreiboperationen bzgl. beliebiger anderer Speicherzugriffe nicht  
(Lesevorgänge können [untereinander] umgeordnet werden, Schreibvorgänge erfolgen in Programm-Reihenfolge)

# Mechanismen für Cache-Kohärenz

- ▶ Speichersystem unterstützt sowohl ...

**Migration** von Daten (Transfer in lokalen Cache) als auch ...

**Replikation** (d.h. Kopieren der Daten in mehrere Caches bei gleichzeitigem Zugriff durch mehrere Prozessoren).  
⇒ Wichtig für Performanz des Zugriffs auf shared data

- ▶ Prinzipielle Lösungsmöglichkeiten: Hardware / Software  
Wegen notwendiger Effizienz: Hardware-Realisierung

⇒ **Cache-Kohärenz Protokolle** (zw. Cache und Bus/Prozessor)

- Grundprinzip: Verfolgung des Zustands eines gemeinsam genutzten Speicherblocks
- Zwei Klassen von Protokollen:  
**Verzeichnisbasiert** und **Snooping** (dt. "Schnüffeln")

# Cache-Kohärenz: Protokollklassen

## Verzeichnisbasiert:

- ▶ Verwendungsstatus eines Speicherblocks wird über zentrale Instanz verwaltet
- ⇒ "Verzeichnis" (*directory*)
  - Beachte: Daten aber ggf. in mehreren Caches vorhanden!  
(Auch: Status dort partiell repräsentiert)

## Snooping:

- ▶ Jeder Cache, der Kopie eines Speicherblocks enthält, besitzt auch Kopie des Verwendungsstatus dieses Blocks
  - ⇒ kein zentral verwalteter Zustand
- ▶ Alle Caches kommunizieren über Bussystem (speziell: SMPs)
  - ⇒ beobachten Anfragen auf dem Bus ("schnüffeln")
- ▶ Reagieren mit Statusänderung, falls Bus-Anfragen gecacheten Speicherblock betreffen
- ⇒ dezentral repräsentierter Zustand durch Kooperation der Caches

# Cache-Kohärenz: Snooping-Protokolle

## Zwei prinzipielle Vorgehensweisen:

1. Aktualisierung aller Kopien eines Blocks beim Schreiben von Daten in den Block
  - ⇒ *write update*
  - ⚡ Problem: Hohe Anforderungen an Speicherbandbreite
    - Reduzierbar, falls sharing von Einträgen verfolgt und nur dann Kopien aktualisiert
2. Exklusiven Zugriff auf Block sicherstellen, *bevor* Prozessor diesen beschreibt
  - ⇒ *write invalidate*
  - Andere Kopien werden durch Schreiben ungültig gemacht
  - Hauptsächlicher Protokolltyp (auch bei verzeichnisbasierten Methoden)

## Cache-Kohärenz: Snooping-Protokolle 2

### Beispiel für write update Prinzip:

Prozessor- aktivität	Bus- aktivität	Cache CPU A	Cache CPU B	Speicher für Zelle X
				0
A liest X	Cache miss für X	0		0
B liest X	Cache miss für X	0	0	0
A schreibt 1	Broadcast von X	1	1	1
B liest X		1	1	1

### Beachte:

- ▶ write update nur in Kombination mit write through Cache sinnvoll
- ▶ Aktualisierung (broadcast) betrifft Speicher und alle gecachten Kopien des jeweiligen Blocks

# Cache-Kohärenz: Snooping-Protokolle 3

## Beispiel für write invalidate Prinzip:

Prozessor- aktivität	Bus- aktivität	Cache CPU A	Cache CPU B	Speicher für Zelle X
				0
A liest X	Cache miss für X	0	0	
B liest X	Cache miss für X	0	0	0
A schreibt 1	“Invalidierung” von X	1	–	0(!)
B liest X	Cache miss für X	1	1	1(!)

**Beachte:** Bei zweitem Cache miss von B für X antwortet A mit korrektem Wert; Antwort des Speichers wird verworfen!

# Cache-Kohärenz: Snooping-Protokolle 4

## Leistungsunterschiede von write update vs write invalidate:

- ▶ Mehrere Schreibzugriffe auf gleichen Block (ohne dazwischenliegendes Lesen)
  - erfordern mehrere Broadcasts,
  - aber nur eine initiale "Invalidierung"
- ▶ Falls Cache-Blöcke mehrere Speicherworte enthalten:
  - Jeder Schreibzugriff erfordert Broadcast
  - Nur erster Schreibzugriff erzeugt "Invalidierung"
- ⇒ write update arbeitet wort-weise, write invalidate auf Cache-Blöcken
- ▶ Verzögerung zwischen Schreiben und "entferntem" Lesen:
  - i.d.R. kleiner bei write update, da Daten sofort im Cache des lesenden Prozessors aktualisiert (sofern dort gecachet)
  - Bei write invalidate wird "lesender" Cache zuerst ungültig, dann Speicherzugriff verzögert, bis Daten aus Hauptspeicher nachgeladen
- ⇒ Wegen Bandbreitenanforderung: write invalidate dominiert!



# Cache-Kohärenz: Snooping-Beispiel Protokoll

- ▶ Drei Zustände pro Cache-Block:

**invalid:** Nicht belegt oder Inhalt wurde verworfen

**shared:** Cache-Block wird in mehreren Caches (lesend) gehalten

**exclusive:** Cache kann exklusiv (schreibend) auf Block zugreifen

- ▶ Kommunikation über Zustandsänderung durch Bus

⇒ Bus-Aktivität + CPU-Aktionen verändern Zustand!

**Schreiben** eines Blocks

⇒ andere existierende Kopien für ungültig erklären

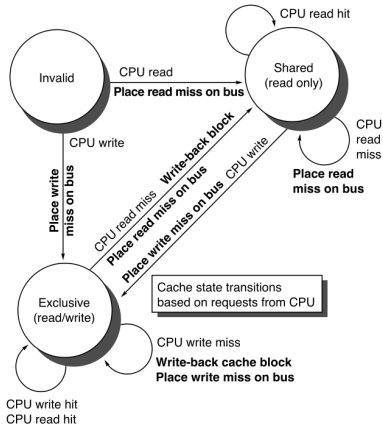
**Lesen** eines Blocks

⇒ Wird shared im akt. Cache, falls nicht modifiziert

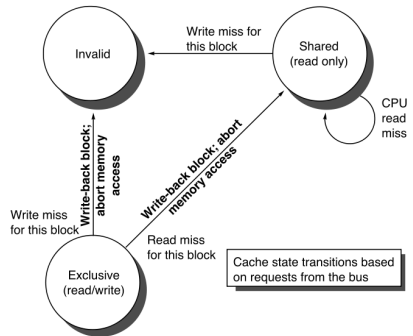
⇒ Muss aus Speicher (write through) oder anderem Cache, der aktuelle Kopie (exclusive) hat, geholt werden (write back)

# Cache-Kohärenz: Snooping-Beispiel Protokoll 2

## CPU-Aktionen



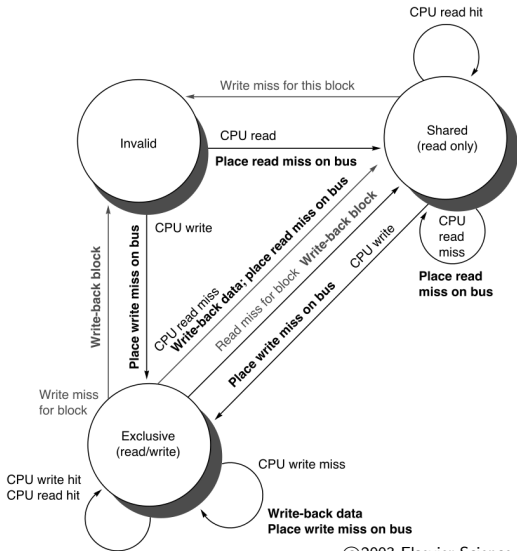
## Aktionen auf dem Bus



©2003 Elsevier Science

# Cache-Kohärenz: Snooping-Beispiel Protokoll 3

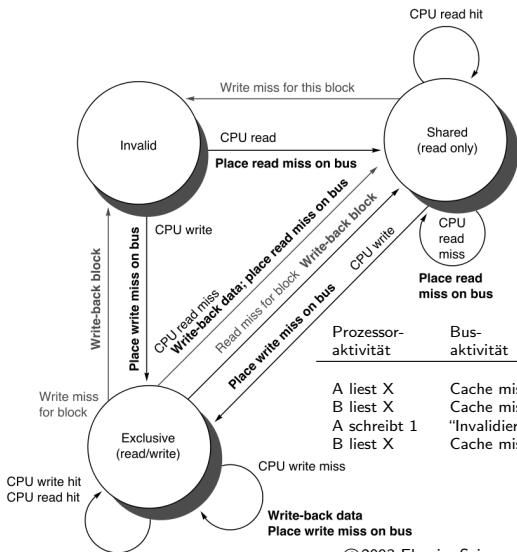
Zusammengefasstes  
Zustands-  
Diagramm des  
Cache-Blocks



# Cache-Kohärenz: Snooping-Beispiel Protokoll 4

Verhalten des Protokolls  
am Beispiel

**Beachte:** Eigentlich zwei  
Zustandsdiagramme  
notwendig: Cache-Block  
für X in CPU A / B!



Prozessor-aktivität	Bus-aktivität	Cache CPU A	Cache CPU B	Speicher für Zelle X
A liest X	Cache miss für X	0	0	0
B liest X	Cache miss für X	0	0	0
A schreibt 1	"Invalidierung" von X	1	-	0(!)
B liest X	Cache miss für X	1	1	1(!)

# Snooping-Protokolle: Erweiterungen

- ▶ Beispielprotokoll auch als MSI-Protokoll bezeichnet (wg. dreier verwendeter Zustände *modified* [hier: *exclusive*], *shared*, *invalid*)
- ▶ Erweiterung zu MESI-Protokoll:  
Einführung eines *exclusive* Zustands, d.h. betreffender Inhalt ist nur in genau einem Cache  $\Rightarrow$  Effizienzgewinn bei Lesen gefolgt von Schreiben (ohne "entfernte" Aktionen)
- ▶ Ggf. Einführung weiterer spezialisierter Zustände  
z.B. *owned*, d.h. Cacheinhalt ist einzige aktuelle Kopie der betreffenden Daten (Speicherinhalt ist ungültig)
- ▶ Auch: Erweiterung auf nicht-bus-basierte Systeme möglich!

Wo ist das Problem?

# Distributed Shared-Memory Architekturen

**Problem:** Anforderungen bzgl. Bandbreite an gemeinsamen Speicher steigt mit Anzahl der Prozessoren

⇒ MP-Systeme mit vielen (>100) Prozessoren nicht in UMA-Architektur realisierbar!

**Lösung:** MP-Systeme mit verteiltem Speicher, logisch gemeinsam nutzbar: *distributed shared memory*  
Cache Kohärenz in best. Architekturen explizit ausgeschlossen (z.B. Cray T3E, nur Caching von privaten Daten)

Wie Cache-Kohärenz in distributed shared memory sicherstellen?

**Beachte:** Caches *nicht* über gemeinsamen Bus verbunden wg. Bandbreite (Verbindung durch z.B. Gitter, Hyperkubus, ...)

## Cache-Kohärenz: Verzeichnisbasierte Protokolle

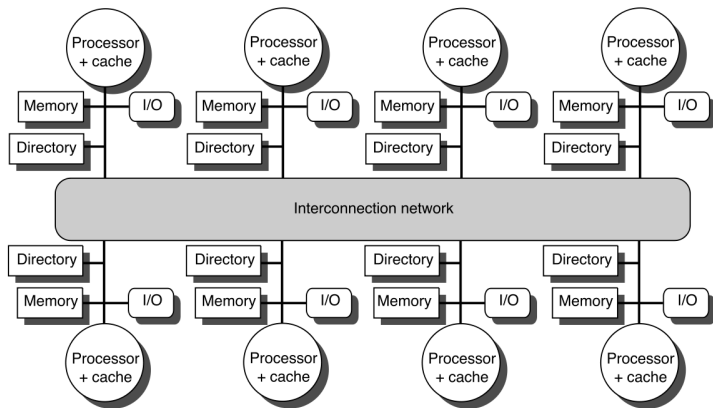
**Ziel:** Cache-Kohärenz für gemeinsame Daten sicherstellen *ohne* zentrale Kommunikation über Bussystem

**Methode:**

- ▶ Zentrale Verwaltung des Status eines Speicherblocks in Verzeichnis (directory), jedem verteiltem Speichersystem zugeordnet
- ▶ Lokale partielle Repräsentation des Status im Cache
  - ⇒ Cache und "Heimat"-Verzeichnis der gecacheten Daten kommunizieren über Nachrichten  
(Mechanismen ggf. komplexer, wenn nicht atomar, zeitl. Reihenfolge nicht eingehalten, ...)
- ▶ Zustände des Cache/Speicher-Blocks analog zu Snooping

# Cache-Kohärenz: Verz.-bas. Protokolle 2

Struktur eines DSM-MP-Systems mit verzeichnisbasiertem Cache-Kohärenzprotokoll



©2003 Elsevier Science

**Beachte:** Ein Directory pro Speicher



# Cache-Kohärenz: Verz.-bas. Beispiel-Protokoll

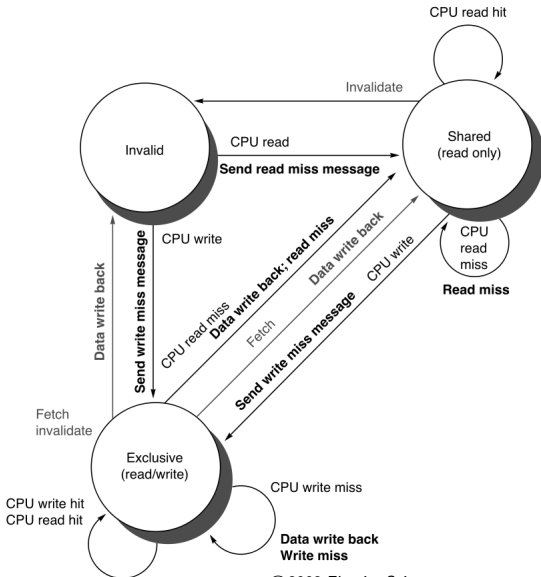
Zustandsdiagramm für Status eines Cache-Blocks

- ▶ Zustände wie Snooping
- ▶ Kommunikation mit Verzeichnis, das zentralen Zustand enthält

**Beachte:** Bei verzeichnisbasiertem Protokoll:

- ▶ Zustand des Speicherblocks  
⇒ Verzeichnis
- ▶ Zustand im Cache:  
Cache-Block  $\neq$  Speicher

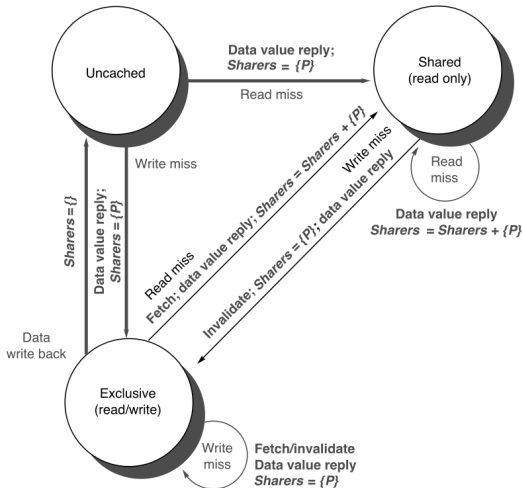
(d.h. kann über Zeit verschiedene Speicherblöcke enthalten!)



# Cache-Kohärenz: Verz.-bas. Bsp.-Protokoll 2

Zustandsdiagramm für Status eines Speicherblocks

- ▶ 3 “bekannte” Zustände (invalid  $\rightarrow$  uncached)
- ▶ Zusätzlich: Liste der Prozessoren, die Kopie der Daten des Speicherblocks besitzen (= *sharers*)
- ▶ Kommunikation mit Caches der sharer



# Cache-Kohärenz: Verz.-bas. Bsp.-Protokoll 3

## Aktionen des Verzeichnisses je nach Zustand:

**Uncached:** Daten im Speicher haben aktuellen Wert

*Read miss:*

- ▶ Anfragender Prozessor erhält Daten a.d. Speicher
- ▶ Wird zum einzigen sharer des Speicherblocks
- ▶ Speicherblock ist jetzt im Zustand shared

*Write miss:*

- ▶ Anfragender Prozessor erhält Daten, wird sharer
- ▶ Zustand des Blocks wird exclusive  
(d.h. einzige gültige Kopie der Daten jetzt von sharer gecached)

**Shared:** ...

**Exclusive:** ...

# Cache-Kohärenz: Verz.-bas. Bsp.-Protokoll 4

(Aktionen des Verzeichnisses ...)

Uncached: ...

Shared: Speicherinhalt ist aktuell, ex. gecachete Kopien

*Read miss:*

- ▶ Anfragender Prozessor erhält Daten aus Speicher
- ▶ Wird zur Menge der sharer hinzugefügt

*Write miss:*

- ▶ Anfragender Prozessor erhält Daten aus Speicher
- ▶ Alle Prozessoren in aktueller Menge der sharer erhalten invalidate-Nachrichten (Cache-Daten für ungültig erklärt)
- ▶ Anfragender Prozessor wird einziger sharer
- ▶ Zustand des Speicherblocks wird exclusive

Exclusive: ...

# Cache-Kohärenz: Verz.-bas. Bsp.-Protokoll 5

(Aktionen des Verzeichnisses ...)

Uncached: ...

Shared: ...

Exclusive: Aktueller Wert im Cache des einzigen sharers

*Read miss:* Anderer Prozessor will lesen

- ▶ “Besitzer” der Daten erhält data fetch Nachricht

Daten: “Besitzer”-Cache → Directory → Speicher

“Besitzer”-Cache: Block-Status → shared

- ▶ Anfragender Prozessor erhält Daten aus Speicher
- ▶ Wird zu Menge der sharer hinzugefügt
- ▶ Zustand des Speicherblocks wird shared

*Data write back:* Besitzer muss Cache-Block ersetzen

- ▶ ...

*Write miss:* Speicherblock hat neuen Besitzer

# Cache-Kohärenz: Verz.-bas. Bsp.-Protokoll 6

(Aktionen des Verzeichnisses ...)

Uncached: ...

Shared: ...

Exclusive: Aktueller Wert im Cache des einzigen sharers

*Read miss:* ...

*Data write back:* Besitzer muss Cache-Block ersetzen

- ▶ Speicher wird wieder aktuell, Verzeichnis wird "Besitzer"
- ▶ Block wird uncached, Menge der sharer ist leer

*Write miss:* Speicherblock hat neuen Besitzer

- ▶ Wird bei altem Besitzer ungültig
- ▶ Wert von altem Besitzer geholt, an anfragenden Prozessor
- ▶ Wird neuer Besitzer (= sharer), Status unverändert

# Cache-Kohärenz: Zusammenfassung

- ▶ Kohärenz definiert in MP-System mit Caches Verhalten von Schreib-Leseoperationen auf *derselben* Speicherzelle
- ▶ Für shared memory (bus-basiert / SMP) oder DSM (distributed shared memory) Systeme möglich (ex. gemeinsamer oder *logisch* gemeinsam genutzter Speicher)
  - bus-basiert / SMP: Snooping-Protokolle, d.h. verteilte Repräsentation / Verwaltung des Cachezustands durch Beobachtung des Buses (→ MSI, MESI, ...)
  - DSM: Verzeichnis (pro [verteilter] Speichermodul) verwaltet Caching-Zustand der Daten; Caches agieren wie bei Snooping-Protokollen

# Cache-Kohärenz: Zusammenfassung

- ▶ Kohärenz definiert in MP-System mit Caches Verhalten von Schreib-Leseoperationen auf *derselben* Speicherzelle
- ▶ Für shared memory (bus-basiert / SMP) oder DSM (distributed shared memory) Systeme möglich (ex. gemeinsamer oder *logisch* gemeinsam genutzter Speicher)
  - bus-basiert / SMP: Snooping-Protokolle, d.h. verteilte Repräsentation / Verwaltung des Cachezustands durch Beobachtung des Buses (→ MSI, MESI, ...)
  - DSM: Verzeichnis (pro [verteilter] Speichermodul) verwaltet Caching-Zustand der Daten; Caches agieren wie bei Snooping-Protokollen
- ⊘ Kohärenz definiert nicht Beziehungen zwischen Schreib-Lese-Operationen auf *unterschiedlichen* Speicherzellen  
⇒ **Konsistenz**