Rechnerstrukturen

Vorlesung im Sommersemester 2006

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.5: Multiprozessoren mit gemeinsamem Speicher

Wiederholung Cache-Speicher

- Aktualisierungsstrategie
  - Befehls-Caches:
    - Prozessor führt nur Lesezugriffe durch
      - Im Cache befindet sich immer die identische Kopie des Hauptspeichers
  - Daten-Cache:
    - Prozessor führt auch Schreibzugriffe durch.
      - Im Hauptspeicher können sich veraltete Daten befinden
    - Aktualisierungsstrategie
      - Bei Schreibzugriff (write hit)
        - » Aktualisieren des Cache-Speichers oder des Hauptspeichers oder beide (write hit):
      - Lesezugriffe dürfen nicht auf inzwischen veraltete Daten gehen;

Wiederholung: Cache-Speicher

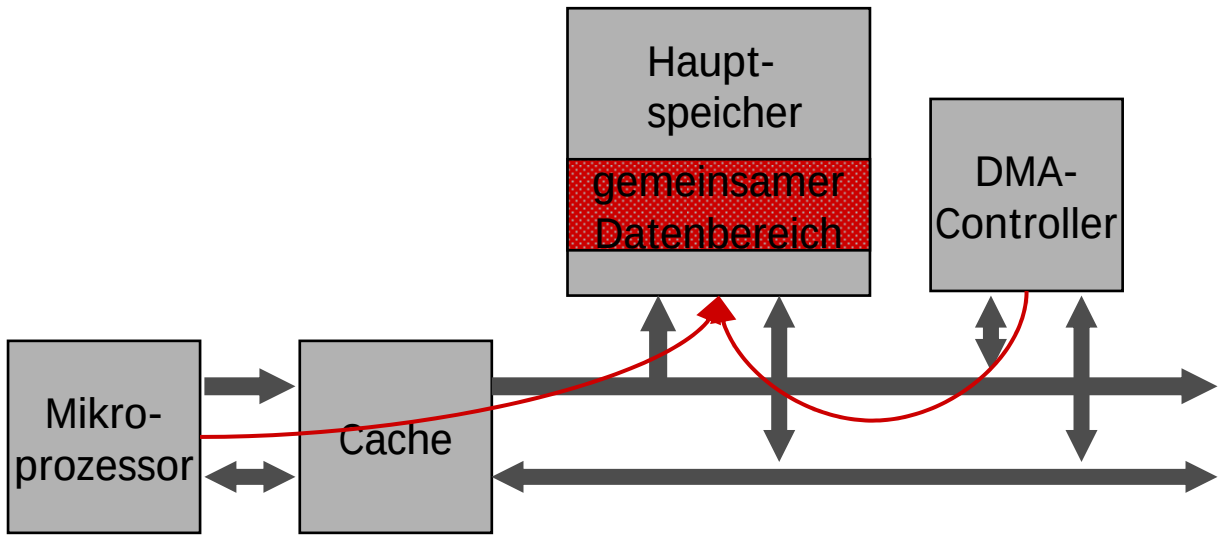
- Aktualisierungsstrategie für Caches mit je einem Valid- und einem Dirty-Bit

Cache-Zugriff	Write-Through No-Write Alloc.	Write-Through Write-Alloc.	Copy-Back
Read-Hit	Cache-Datum --> CPU	Cache-Datum --> CPU	Cache-Datum --> CPU
Read-Miss	HS-Block, Tag --> Cache HS-Datum --> CPU 1 --> V	HS-Block,Tag --> Cache HS-Datum --> CPU 1 --> V	Cache-Zeile --> HS HS-Block, Tag --> Cache HS-Datum --> CPU 1 --> V, 0 --> D
Write-Hit	CPU-Datum --> Cache,HS	CPU-Datum --> Cache,HS	CPU-Datum --> Cache 1 --> D
Write-Miss	CPU-Datum --> HS	HS-Block, Tag --> Cache, 1 --> V CPU-Datum --> Cache, HS	Cache-Zeile --> HS HS-Block, Tag --> Cache 1 --> V CPU-Datum --> Cache 1 --> D

Multiprozessor mit gemeinsamem Speicher

- Cache-Kohärenz-Problem
  - 1. Fall (I/O-Problem)
    - System mit einem Mikroprozessoren und weiteren Komponenten mit Master-Funktion (ohne Cache)
      - Zusätzlicher Master kann Kontrolle über Bus übernehmen
        - » Kann damit unabhängig vom Prozessor auf Hauptspeicher zugreifen
        - » Mikroprozessor und Master teilen sich gemeinsamen Datenbereich

- Cache-Kohärenz-Problem
  - Beispiel: Mikroprozessorsystem mit DMA-Controller



Multiprozessor mit gemeinsamem Speicher

- Cache-Kohärenz-Problem
  - Beispiel: Mikroprozessorsystem mit DMA-Controller
    - Problem: Zugriff auf veraltete Daten (stale data)
      - Problem beim Write-Through-Verfahren
        - » DMA-Controller beschreibt eine Speicherzelle, deren Inhalt im Cache als gültig eingetragen war, der Prozessor führt danach einen Lesezugriff mit der Adresse dieser Speicherzelle durch:
        - ➔Prozessor liest veraltetes Datum
      - Problem beim Copy-Back-Verfahren:
        - » der Prozessor führt Schreibzugriff mit der Adresse aus dem gemeinsamen Bereich aus und aktualisiert nur Cache; der DMA-Controller liest anschließend die Speicherzelle mit dieser Adresse:
        - ➔der DMA-Controller liest veraltetes Datum (im Hauptspeicher);

Multiprozessor mit gemeinsamem Speicher

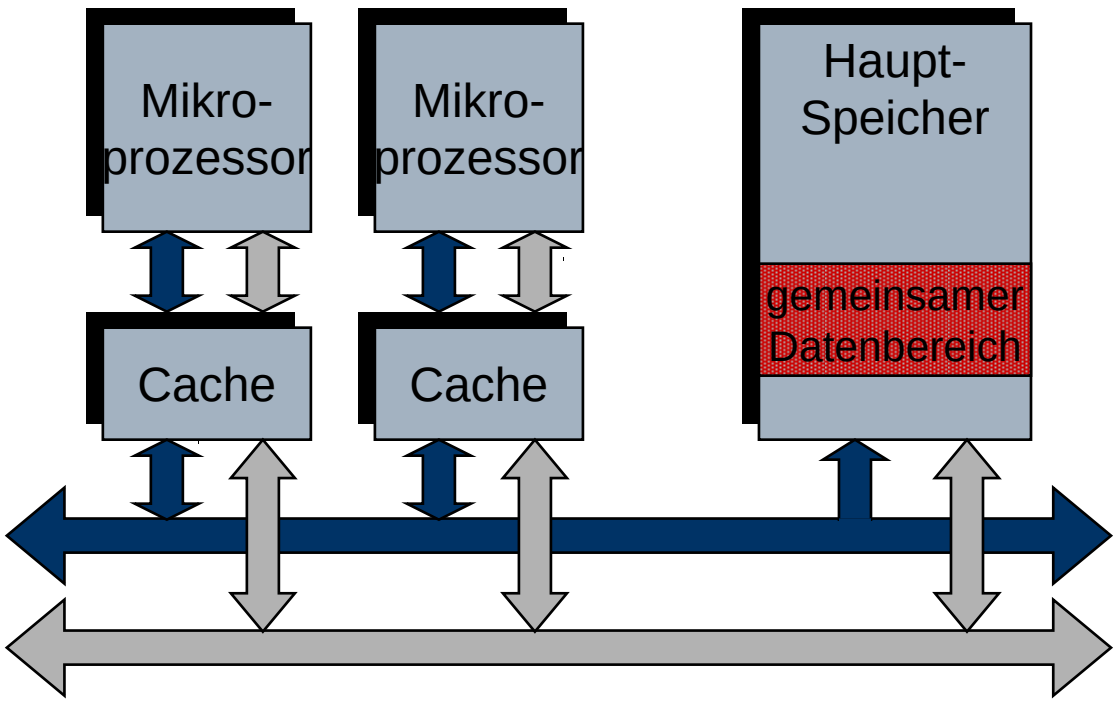
- Cache-Kohärenz-Problem
  - Beispiel: Mikroprozessorsystem mit DMA-Controller
    - Lösung des Kohärenzproblems (1):
      - Non-Cachable Data
        - » der vom Prozessor und dem zusätzlichen Master gemeinsam benutzte Speicherbereich wird von der Speicherung im Cache ausgeschlossen;
      - Aufgabe der Speicherverwaltung:
        - » Der Adressbereich wird in seinem für die Speicherverwaltungseinheit bereitgestelltem Deskriptor als „non-cacheable“ gekennzeichnet.
        - » Die Cache-Steuerung wird bei Zugriffen auf den so gekennzeichneten Bereich nicht aktiv.
        - » Es werden auch die für Schnittstellen und Controller reservierten Adressbereiche als „non-cacheable“ gekennzeichnet, um den direkten Zugriff auf deren Daten-; Steuer- und Statusregister zu gewährleisten.

Multiprozessor mit gemeinsamem Speicher

- Cache-Kohärenz-Problem
  - Beispiel: Mikroprozessorsystem mit DMA-Controller
    - Lösung des Kohärenzproblems (2)
      - Cache-Clear, Cache-Flush
        - » Die Zugriffe von Prozessor und DMA-Controller auf den gemeinsamen Datenbereich werden von zwei unterschiedlichen Tasks ausgeführt;
        - » In diesem Fall kann die Task, die den DMA-Vorgang auslöst, dafür sorgen, dass der Cache gelöscht wird, d.h nachfolgende Prozessorzugriffe führen zu einem Neuladen des Cache;
      - Write-Through: Cache-Clear:
        - » Die Cache-Einträge werden auf ungültig gesetzt.
      - Copy-Back: Cache-Flush:
        - » Alle mit „dirty“ gekennzeichneten Einträge im Cache werden in den Hauptspeicher zurückgeschrieben, danach werden Cache-Einträge auf ungültig gesetzt.

Multiprozessor mit gemeinsamem Speicher

- Cache-Kohärenz-Problem
  - 2. Fall:
    - Speichergekoppeltes Multiprozessorsystem
      - Mehrere Prozessoren mit jeweils eigenen Cache-Speichern sind über einem Systembus an einen gemeinsamen Hauptspeicher angebunden.



- Cache-Kohärenz und Konsistenz
  - Vereinfachte und intuitive Definition:
    - Ein Speichersystem ist kohärent, wenn jeder Lesezugriff auf ein Datum den aktuell geschriebenen Wert dieses Datums liefert
      - Kohärenz:
        - » definiert, welcher Wert bei einem Lesezugriff geliefert wird
      - Konsistenz:
        - » bestimmt, wann ein geschriebener Wert bei einem Lesezugriff geliefert wird

Multiprozessor mit gemeinsamem Speicher

- Kohärenz
  - Ein Speichersystem ist kohärent, wenn
    - Ein Lesezugriff eines Prozessors P auf eine Speicherstelle X, der einem Schreibzugriff von P auf die Stelle X folgt und keine Schreibzugriffe anderer Prozessoren zwischen dem Schreiben und dem Lesen von P stattfinden, liefert immer den Wert, den P geschrieben hat.
      - Einhaltung der Programmordnung
    - Ein Lesezugriff eines Prozessors P auf eine Speicherstelle X, der auf einen Schreibzugriff eines anderen Prozessors auf die Stelle X folgt, liefert den geschriebenen Wert, falls der Lese- und Schreibzugriff zeitlich ausreichend getrennt erfolgen und in der Zwischenzeit keine anderen Schreibzugriffe auf die Stelle X erfolgen.
      - Kohärente Sicht des Speichers
    - Schreibzugriffe auf die eine Speicherzelle werden serialisiert; d.h. zwei Schreibzugriffe auf eine Speicherstelle durch zwei Prozessoren werden durch die anderen Prozessoren in der selben Reihenfolge gesehen.
      - Write Serialization

Multiprozessor mit gemeinsamem Speicher

- Konsistenz
  - Frage: Wann wird ein geschriebener Wert sichtbar?
    - Warum?
      - Man kann nicht fordern, dass ein Lesezugriff auf eine Stelle X sofort den Wert liefert, der von einem Schreibzugriff auf X eines anderen Prozessors stammt
        - » Z.B.: Die Schreiboperation eines Prozessors auf die Stelle X erfolgt kurz vor dem Lesezugriff eines anderen Prozessors auf diese Stelle, dann kann nicht gesichert werden, dass die Leseoperation den richtigen Wert erhält, da möglicherweise die zu schreibenden Daten den Prozessor noch nicht verlassen haben.
  - Konsistenzmodell:
    - Strategie, wann ein Prozessor die Schreiboperationen eines anderen Prozessors sieht

Multiprozessor mit gemeinsamem Speicher

- Kohärenz und Konsistenz
  - Kohärenz:
    - Definiert das Verhalten von Lese- und Schreibzugriffen auf ein und dieselbe Speicherstelle
  - Konsistenz
    - Definiert das Verhalten von Lese- und Schreiboperationen in bezüglich Zugriffe auf andere Speicherstellen
  - Vereinfachte Sicht:
    - Wir fordern, dass eine Schreiboperation nicht abgeschlossen ist, bevor die anderen Prozessoren den Effekt sehen und dass der Prozessor die Ordnung von einem Schreibzugriffen mit einem anderen Speicherzugriff ändert
      - Präzisere Betrachtung später!

Multiprozessor mit gemeinsamem Speicher

- Erhaltung der Kohärenz
  - Ein paralleles Programm, das auf einem Multiprozessor läuft, wird üblicherweise mehrere Kopien eines Datums in mehreren Caches haben
  - Migration bei kohärenten Caches
    - Daten können zu einem lokalen Cache migrieren und dort in einer transparenten Weise verwendet werden
    - Reduziert die Latenz für einen Zugriff auf ein gemeinsames Datum, das auf einem entfernten Speicher liegt
    - Reduziert auch die erforderliche Bandbreite auf den gemeinsamen Speicher
  - Replikation bei kohärenten Caches
    - Gemeinsame Daten können in als Kopien in lokalen Caches vorliegen, wenn beispielsweise diese Daten gleichzeitig gelesen werden
    - Reduziert die Latenz der Zugriffe und die Möglichkeit einer Blockierung beim Zugriff auf das gemeinsame Datum

Multiprozessor mit gemeinsamem Speicher

- Erhaltung der Kohärenz
  - Hardware-Lösung
    - Einführung eines Cache-Kohärenzprotokolls zur Verwaltung kohärenter Caches
      - Festhalten des Zustands in dem sich gemeinsame Daten befinden
    - Tabellen-basierte Protokolle (directory-based protocols)
      - Der Zustand eines Blocks im physikalischen Speicher wird in einer Tabelle (directory) festgehalten
    - Snooping-Protokolle (Bus-Schnüffeln)
      - Jeder Cache, der eine Kopie der Daten eines Blocks des physikalischen Speichers enthält, hat ebenso eine Kopie des Zustands, in dem sich der Block befindet
      - Kein zentraler Zustand wird festgehalten
      - Caches sind an einem gemeinsamen Bus und alle Cache-Controller beobachten (oder schnüffeln) am Bus, um bestimmen zu können, ob sie eine Kopie eines Blocks enthalten, der benötigt wird

Multiprozessor mit gemeinsamem Speicher

- Erhaltung der Kohärenz
  - Möglichkeiten, die Kohärenzanforderungen zu erfüllen:
    - Write-invalidate-Protokoll:
      - Sicherstellen, dass ein Prozessor exklusiven Zugriff auf ein Datum hat, bevor er schreiben darf
      - Vor dem Verändern einer Kopie in einem Cache-Speicher müssen alle Kopien in anderen Cache-Speichern für „ungültig“ erklärt werden
    - Write-update-Protokoll:
      - Beim Verändern einer Kopie in einem Cache-Speicher müssen alle Kopien in anderen Cache-Speichern ebenfalls verändert werden, wobei die Aktualisierung auch verzögert (spätestens beim Zugriff) erfolgen kann

Multiprozessor mit gemeinsamem Speicher

- Kohärenz-Protokolle
  - Vergleich Write-invalidate-Protokoll und Write-update-Protokoll:
    - Mehrfaches Schreiben auf eine Stelle ohne dazwischen auftauchende Lesezugriffe
      - Write-Update:
        - » erfordern mehrere Broadcast-Schreiboperationen
      - Write-Invalidate:
        - » Nur eine Invalidierung
    - Cache-Zeilen mit mehreren Wörtern
      - Write-Update
        - » Arbeitet auf Wörtern
        - » Für jedes Wort in einem Block, das geschrieben wurde, ist ein Write-Broadcast notwendig
      - Write-Invalidate
        - » Die erste Schreiboperation auf ein Wort eines Cache-Blocks erfordert eine Invalidierung

Multiprozessor mit gemeinsamem Speicher

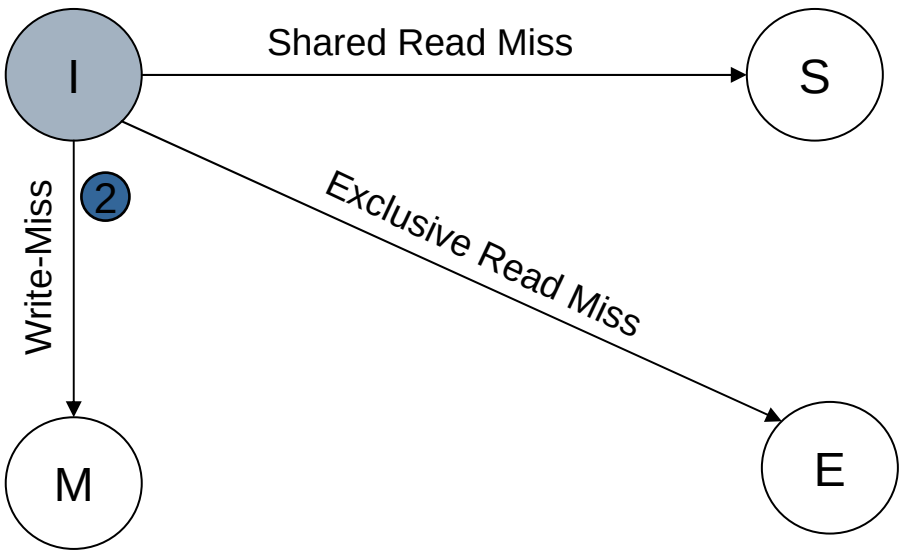
- Kohärenz-Protokolle
  - Vergleich Write-invalidate-Protokoll und Write-update-Protokoll:
    - Verzögerung zwischen dem Schreiben eines Worts und dem Lesen des geschriebenen Werts von einem anderen Prozessor ist im Allgemeinen geringer bei Write-Update
      - Die geschriebenen Daten werden sofort im Cache des lesenden Prozessors aktualisiert, wobei angenommen wird, das eine Kopie im Cache vorhanden ist
      - Bei Write-Invalidate muss beim Leser erst invalidiert werden und muss dann warten, bis eine Kopie geliefert wird
    - Write-Invalidate:
      - Generieren weniger Bus- und Speicherverkehr
      - Der am meisten verwendete Protokolltyp in Multiprozessoren

- Kohärenz-Protokolle
  - MESI-Kohärenzprotokoll
    - Jeder Cache verfügt über Snoop-Logik und Steuersignale:
      - Invalidate-Signal:
        - » Invalidieren von Einträgen in den Caches anderer Prozessoren.
      - Shared-Signal:
        - » Anzeige, ob ein zu ladender Block bereits als Kopie vorhanden ist.
      - Retry-Signal:
        - » Aufforderung für einen Prozessor, das Laden eines Blockes abzubrechen. Das Laden wird dann wieder aufgenommen, wenn ein anderer Prozessor aus dem Cache in den Hauptspeicher zurück geschrieben hat.

- MESI-Kohärenzprotokoll
  - Jede Cache-Zeile ist um zwei Statusbits erweitert
    - Zeigen Protokollzustände an:
      - Invalid (I)
      - Shared (S)
      - Exclusive (E)
      - Modified (M)
    - Beim Write-Through-Verfahren sind nur die Zustände Shared und Ivalidate relevant

- MESI-Kohärenzprotokoll
  - Bedeutung der Statusbits:
    - Invalid (I): Die betrachtete Cache-Zeile ist ungültig.
      - Lese- und Schreibzugriff auf diese Zeile veranlassen die Cache-Steuerung, den Speicherblock in die Cache-Zeile zu laden.
      - Die anderen Cache-Steuerungen, die den Bus beobachten, zeigen mit Hilfe des Shared-Signals an, ob dieser Block gespeichert ist (Shared Read Miss) oder nicht (Exclusive Read Miss).
      - Davon abhängig erfolgt der Übergang in den Zustand S oder E.
      - Bei einem Write-Miss erfolgt der Übergang in den Zustand M. Der Prozessor gibt dabei wegen der Änderung das Invalidate-Signal aus, das von den anderen Caches ausgewertet wird.

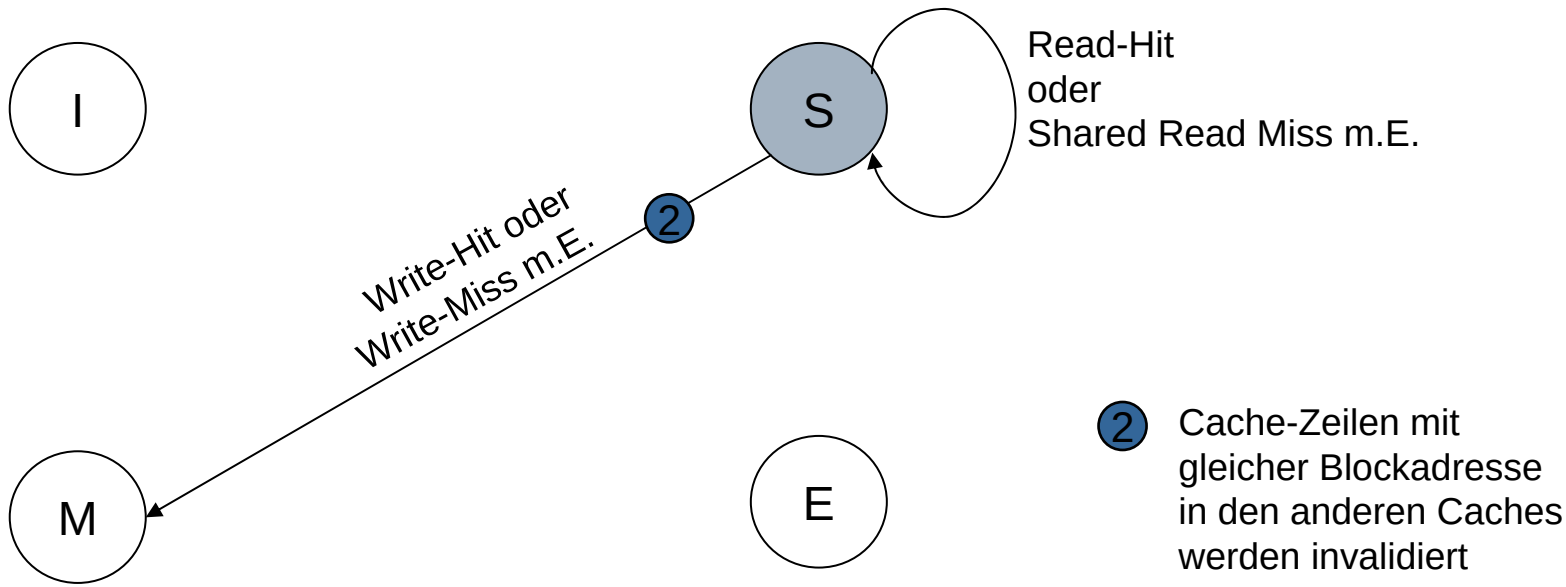
- MESI-Kohärenzprotokoll
  - Zustandsübergänge für die lokalen Schreib- und Lesezugriffe
    - Invalid (I):



2 Cache-Zeilen mit gleicher Blockadresse in den anderen Caches werden invalidiert

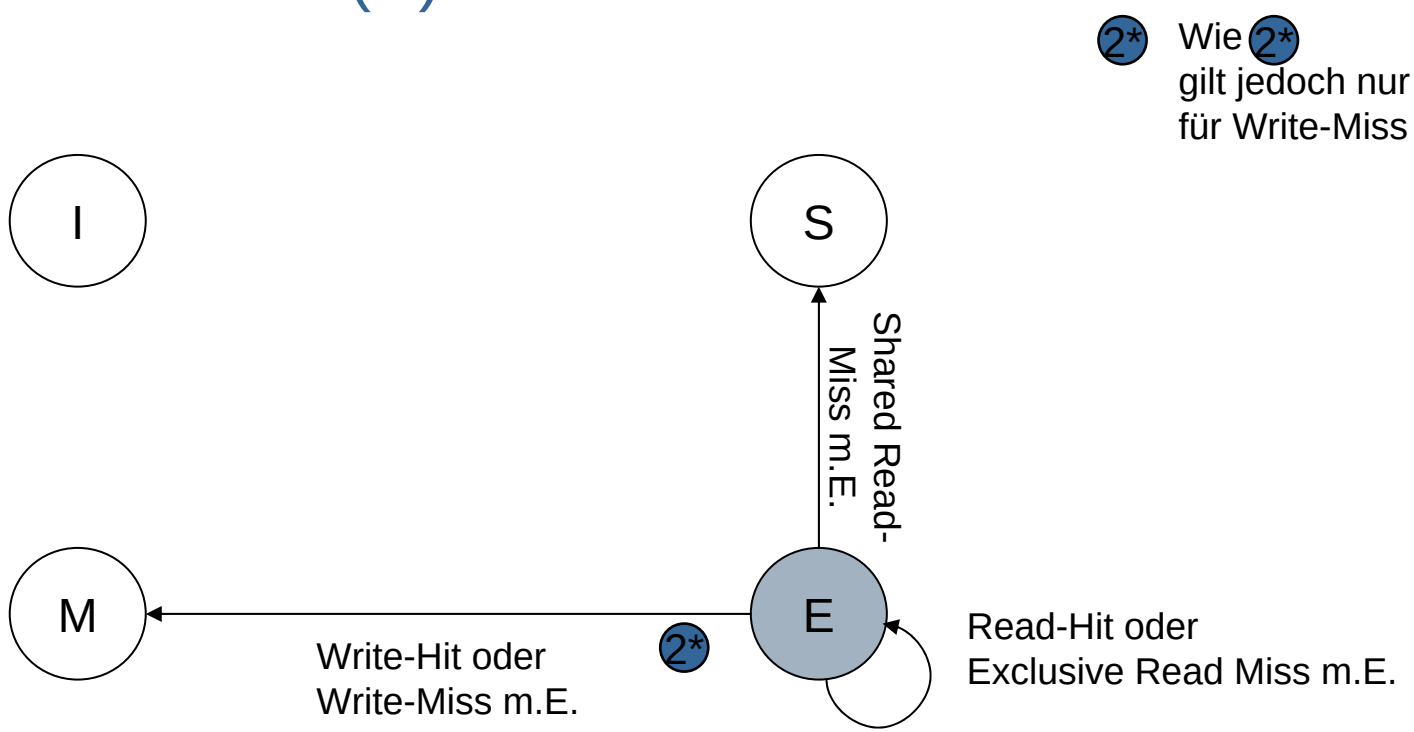
- MESI-Kohärenzprotokoll
  - Bedeutung der Statusbits:
    - Shared (S), Shared Unmodified: Der Speicherblock existiert als Kopie in der Zeile des betrachteten Caches sowie gegebenenfalls in anderen Caches.
      - Lesezugriff auf die Cache-Zeile (Read-Hit):
        - » Der Zustand wird nicht verändert.
      - Schreibzugriff auf die Cache-Zeile (Write-Hit):
        - » Die Cache-Zeile wird geändert und geht in den Zustand M über.
        - » Ausgeben des Invalidate-Signals, woraufhin die Caches, bei denen diese Cache-Zeile ebenfalls im Zustand S ist, diese als ungültig kennzeichnen (Zustand I).

- MESI-Kohärenzprotokoll
  - Zustandsübergänge für die lokalen Schreib- und Lesezugriffe
    - Shared (S):



- MESI-Kohärenzprotokoll
  - Bedeutung der Statusbits:
    - Exclusive (E), Exclusive Unmodified: Der Speicherblock existiert als Kopie nur in der Zeile des betrachteten Caches.
      - Der Prozessor kann lesend und schreiben zugreifen, ohne den Bus benützen zu müssen.
      - Schreibzugriff:
        - » Wechseln in den Zustand M.
        - » Andere Caches sind nicht betroffen.

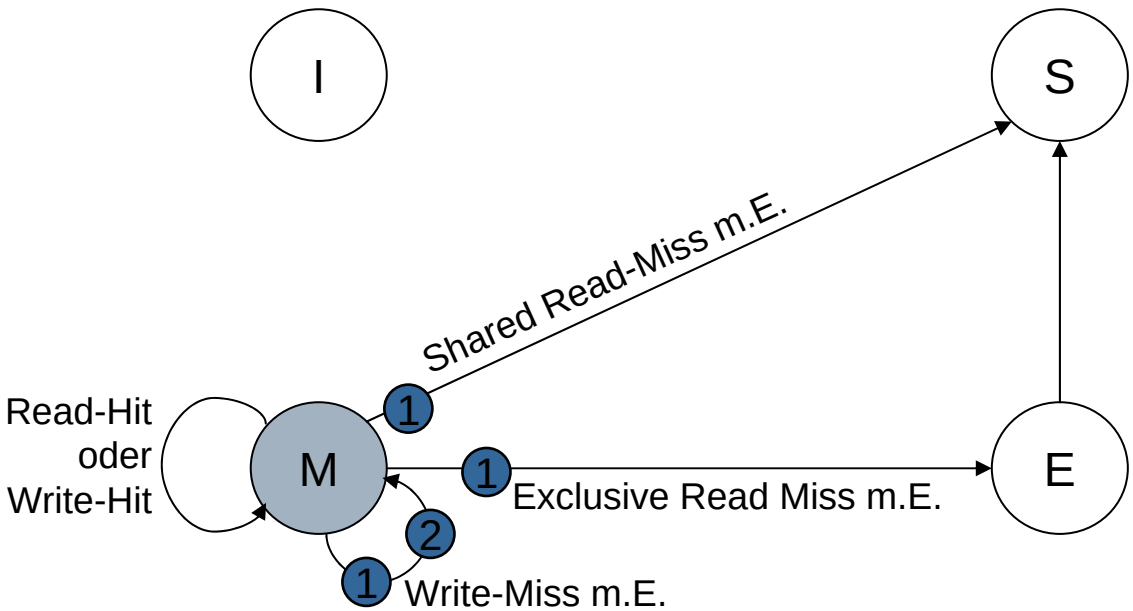
- MESI-Kohärenzprotokoll
  - Zustandsübergänge für die lokalen Schreib- und Lesezugriffe
    - Exclusive (E):



Multiprozessor mit gemeinsamem Speicher

- MESI-Kohärenzprotokoll
  - Bedeutung der Statusbits:
    - Modified (M), Exclusive Modified: Der Speicherblock existiert als Kopie nur in der Zeile des betrachteten Caches. Er wurde nach dem Laden verändert.
      - Der Prozessor kann lesend und schreiben zugreifen, ohne den Bus benützen zu müssen.
      - Bei einem Lese- oder Schreibzugriff eines anderen Prozessors auf diesen Block (Snoop-Hit) muss dieser in den Hauptspeicher zurückkopiert werden.
        - » Snoop-Hit on a Read: Übergang von M → S
        - » Snoop-Hit on a Write or Read with Intend to Modify: Übergang von M→I
      - Der Prozessor, der diesen Block aus dem Hauptspeicher holen will, wird mit Hilfe des Retry-Signals darüber informiert, dass zunächst ein Zurückschreiben erforderlich ist.

- MESI-Kohärenzprotokoll
  - Zustandsübergänge für die lokalen Schreib- und Lesezugriffe
    - Modified (M):



1 Cache-Zeile wird in Den Hauptspeicher zurückkopiert

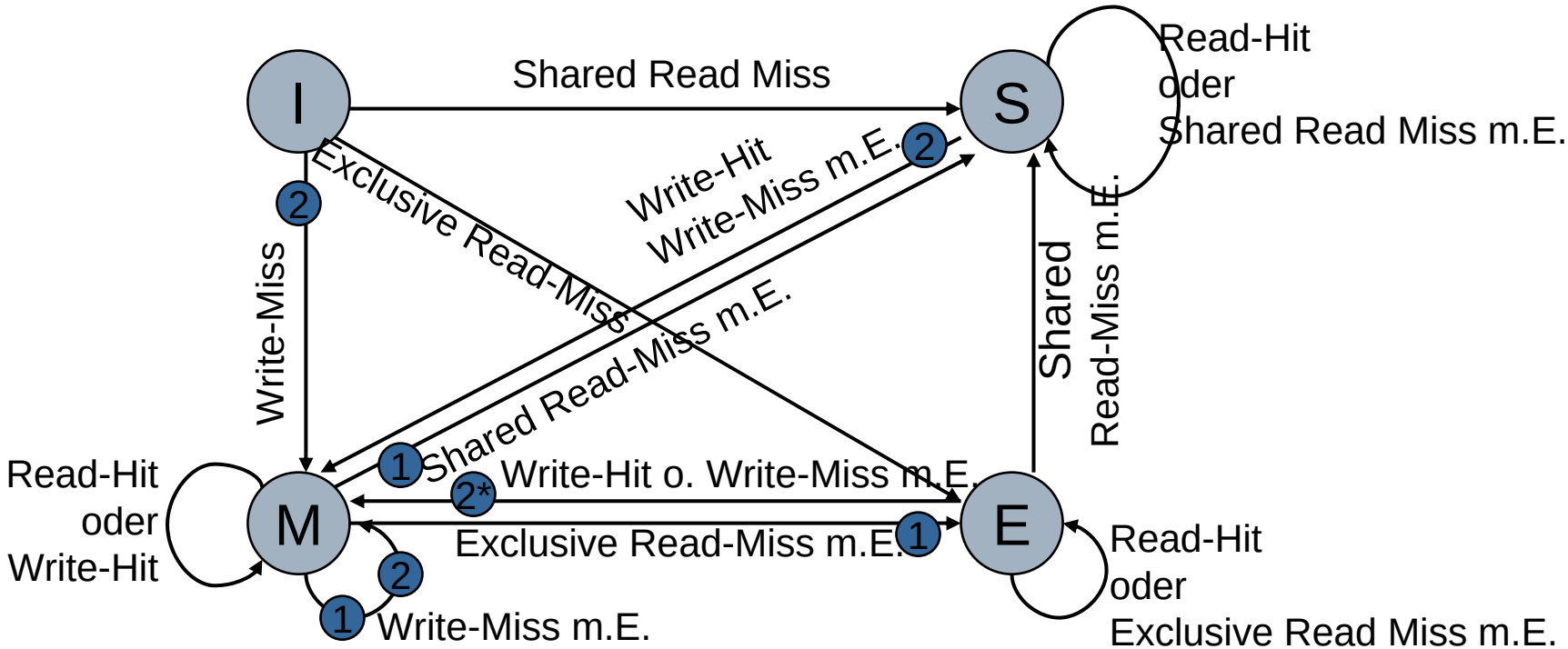
2 Cache-Zeilen mit gleicher Blockadresse in den anderen Caches werden invalidiert

Multiprozessor mit gemeinsamem Speicher

- MESI-Kohärenzprotokoll
  - Bedeutung der Statusbits:
    - Modified (M), Exclusive Modified: Der Speicherblock existiert als Kopie nur in der Zeile des betrachteten Caches. Er wurde nach dem Laden verändert.
      - Der Prozessor kann lesend und schreiben zugreifen, ohne den Bus benützen zu müssen.
      - Bei einem Lese- oder Schreibzugriff eines anderen Prozessors auf diesen Block (Snoop-Hit) muss dieser in den Hauptspeicher zurückkopiert werden.
        - » Snoop-Hit on a Read: Übergang von M → S
        - » Snoop-Hit on a Write or Read with Intend to Modify: Übergang von M→I
      - Der Prozessor, der diesen Block aus dem Hauptspeicher holen will, wird mit Hilfe des Retry-Signals darüber informiert, dass zunächst ein Zurückschreiben erforderlich ist.

Multiprozessor mit gemeinsamem Speicher

- Zustandsgraph 1 des MESI-Protokolls



m.E.: mit Ersetzung

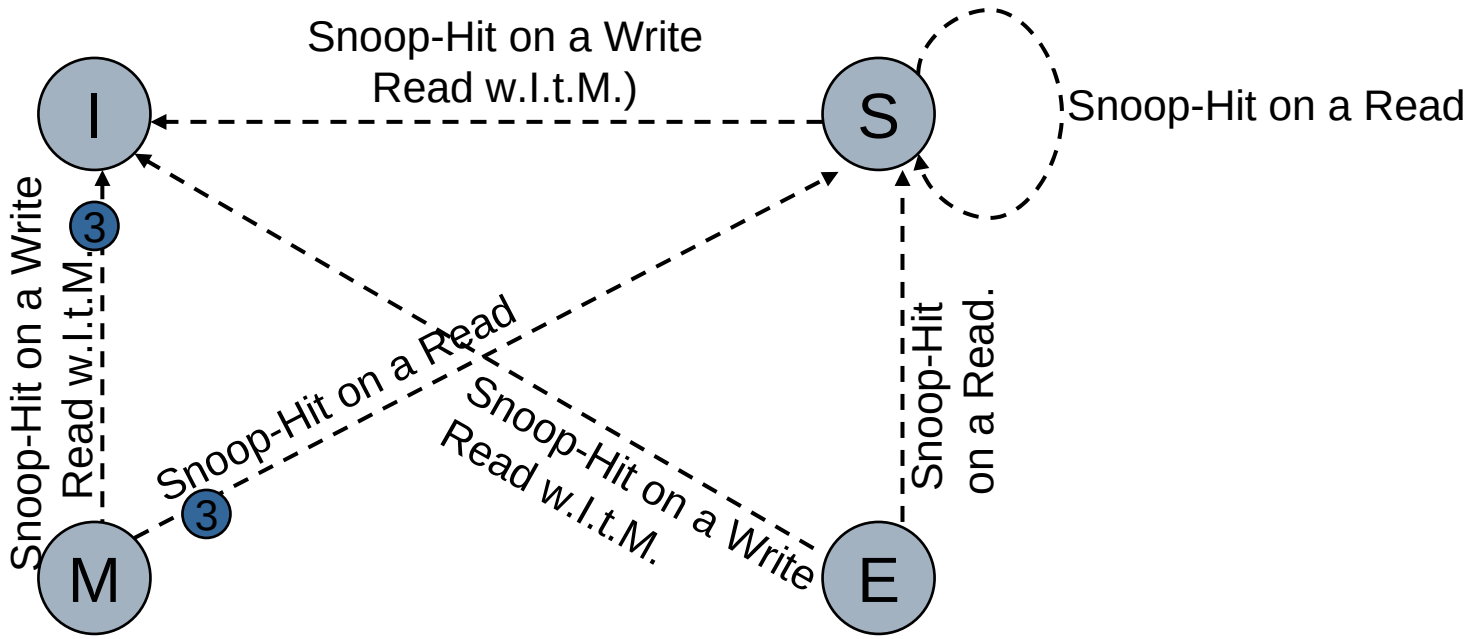
1 Cache-Zeile wird in den Hauptspeicher zurückkopiert. (Line-Flush)

2 Cache-Zeilen in den anderen Caches mit gleicher Blockadresse werden invalidiert. (Line Clear)

2* Wie 2: wie 2, gilt jedoch nur für Write-Miss mit Ersetzung

Multiprozessor mit gemeinsamem Speicher

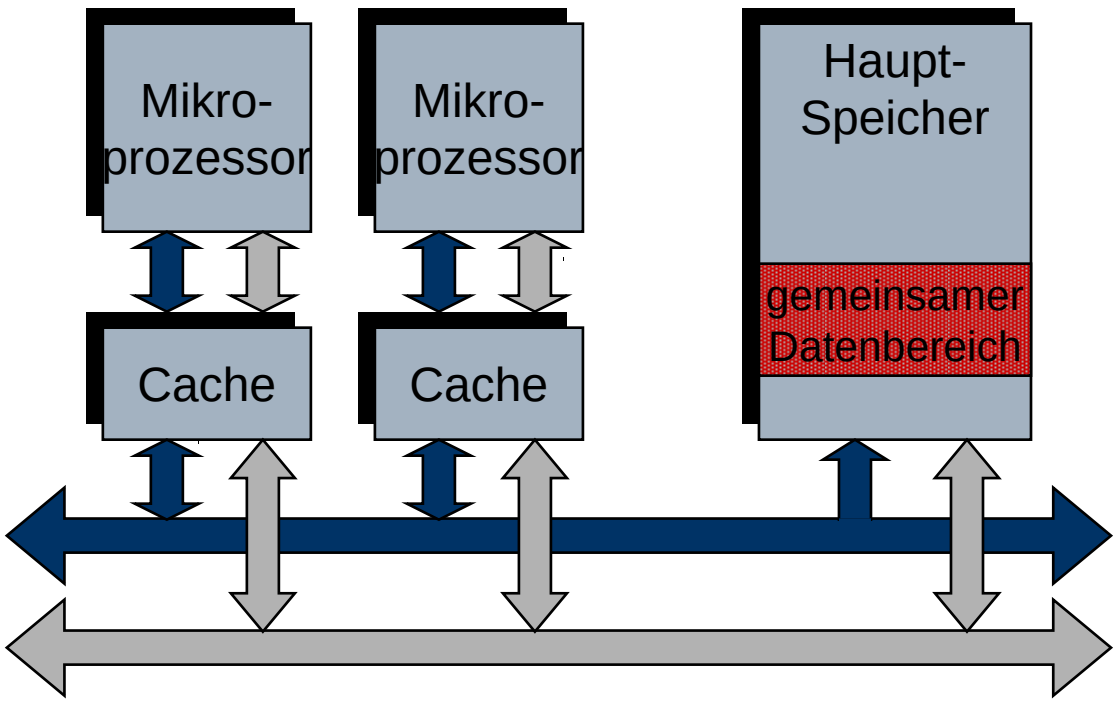
- Zustandsgraph 2 des MESI-Protokolls
  - Zustandsübergänge, die durch Aktionen, die auf dem Bus zu beobachten sind, ausgelöst werden.



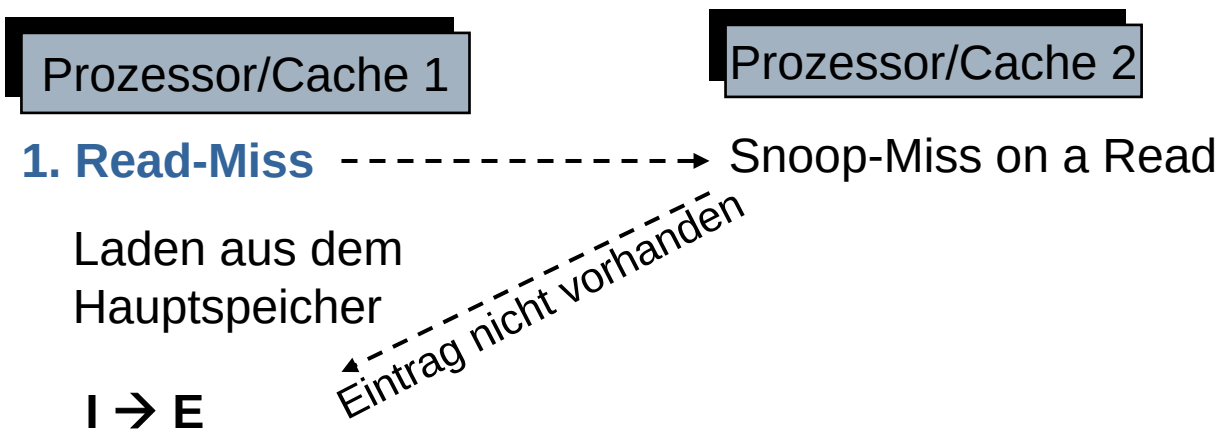
3 Retry-Signal wird aktiviert und danach wird die Cache-Zeile in den Hauptspeicher kopiert.

Multiprozessor mit gemeinsamem Speicher

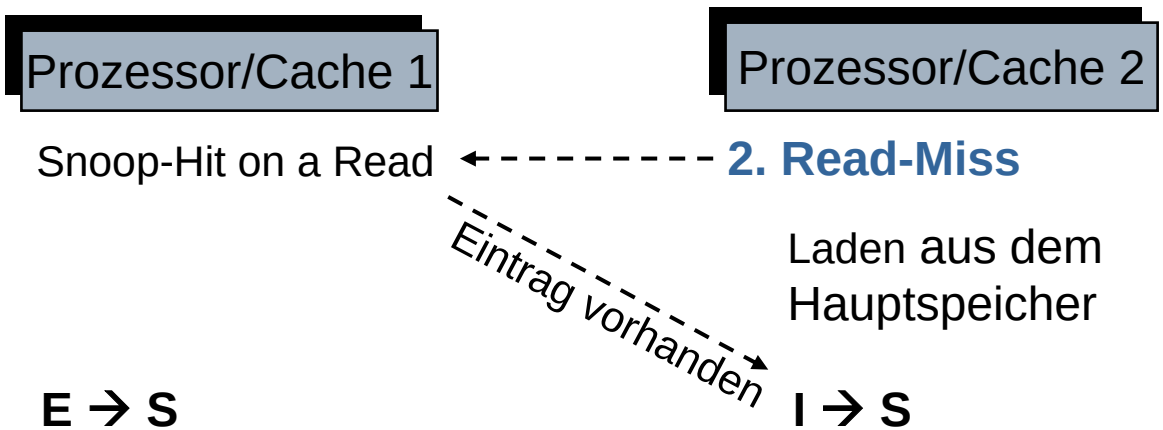
- MESI-Kohärenz-Protokoll
  - Wirkungsweise am Beispiel eines Mikroprozessorsystems mit 2 Prozessoren



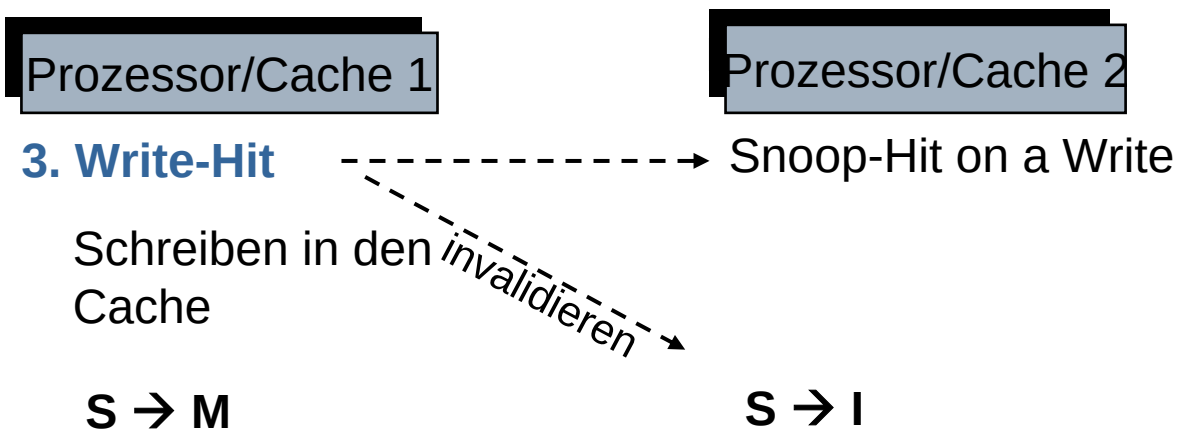
- Wirkungsweise des MESI-Protokolls
  - Beispiel: Mikroprozessorsystem mit zwei Prozessoren
    - Vier aufeinanderfolgende Zugriffe auf ein und denselben Speicherblock



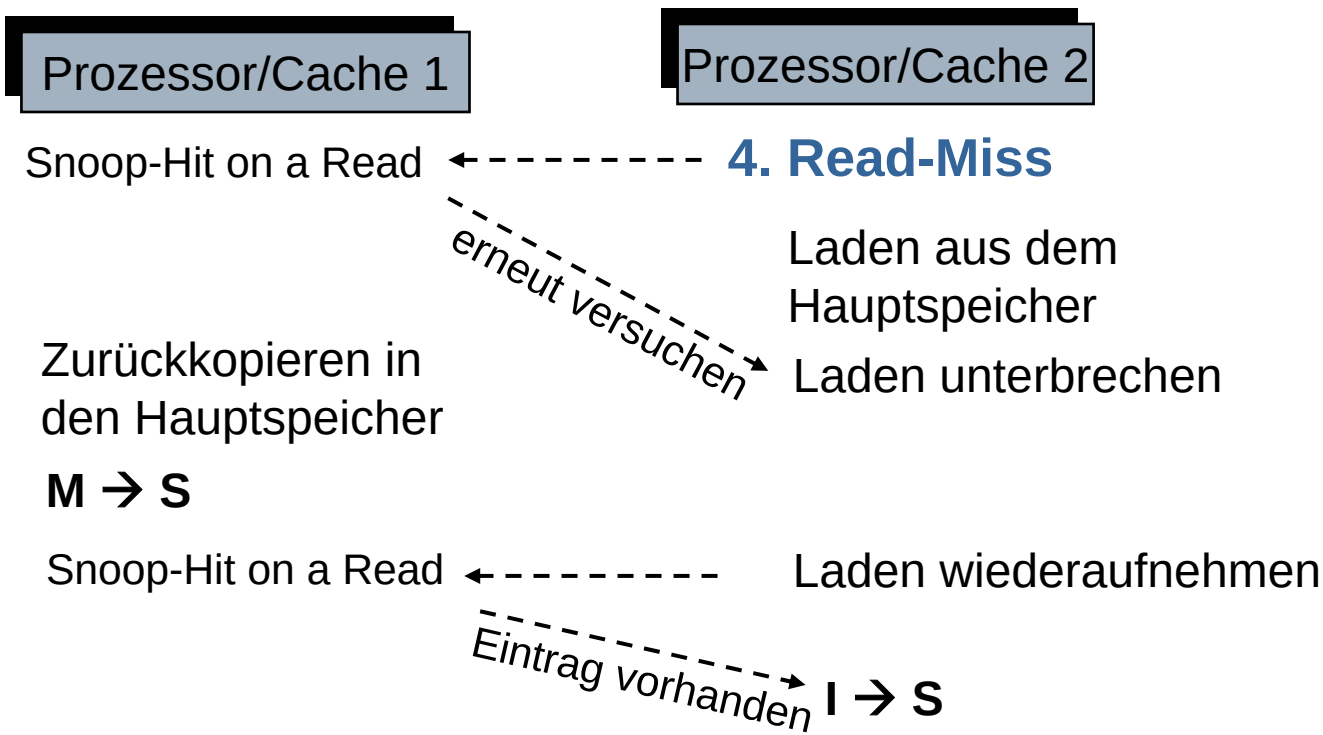
- Wirkungsweise des MESI-Protokolls
  - Beispiel: Mikroprozessorsystem mit zwei Prozessoren



- Wirkungsweise des MESI-Protokolls
  - Beispiel: Mikroprozessorsystem mit zwei Prozessoren

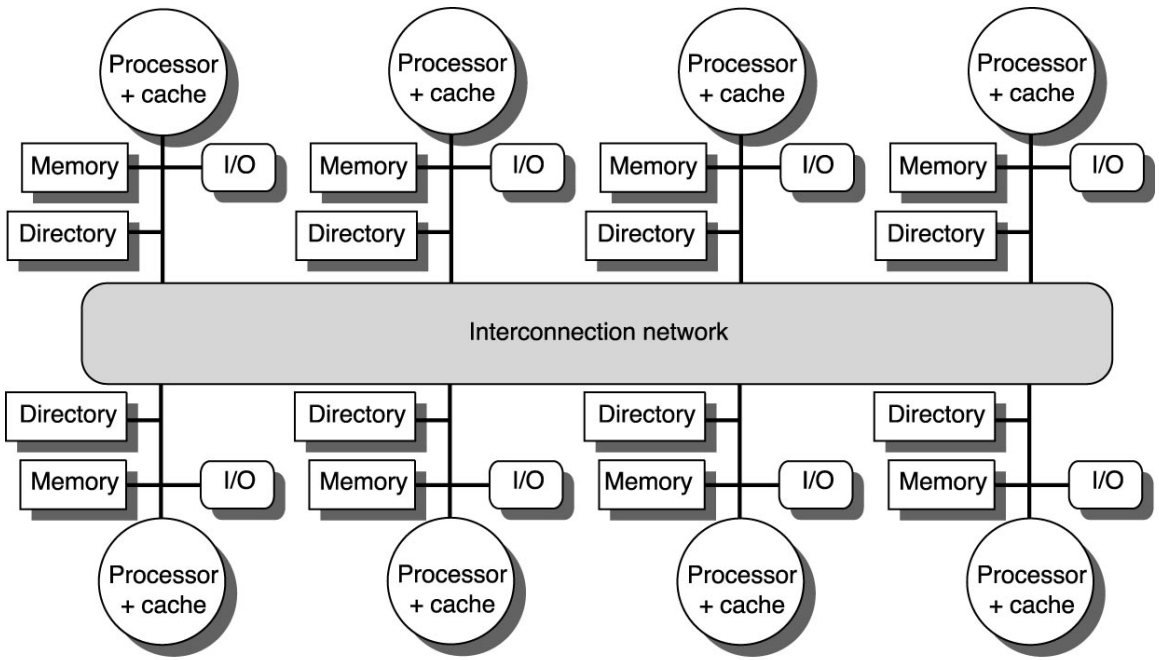


- Wirkungsweise des MESI-Protokolls
  - Beispiel: Mikroprozessorsystem mit zwei Prozessoren



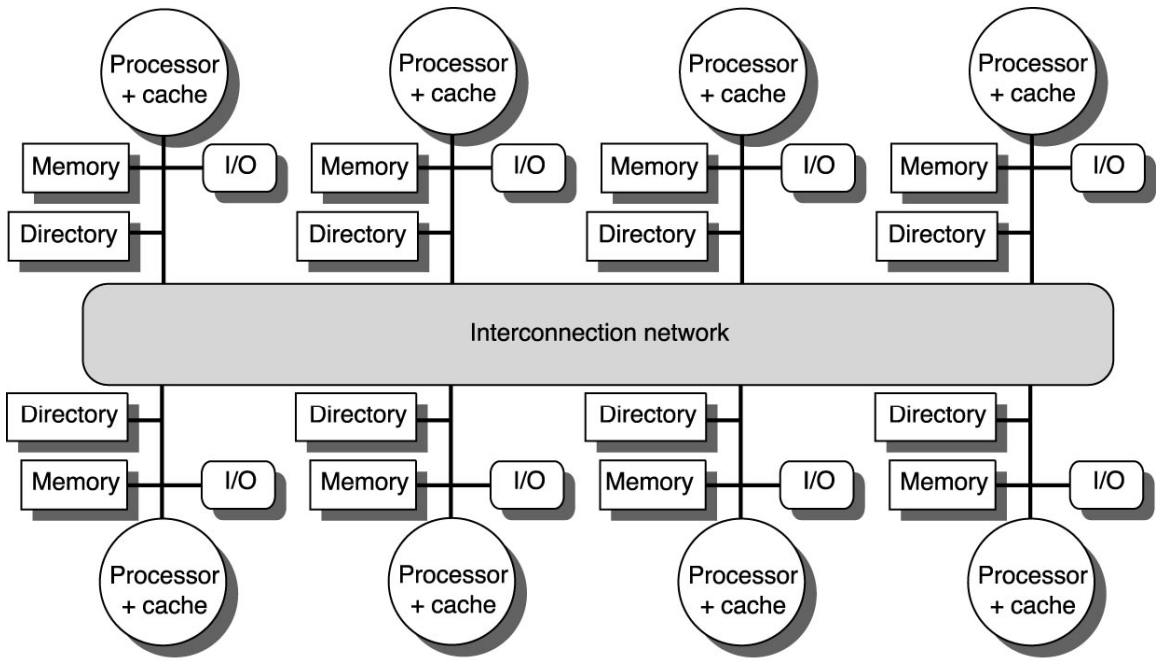
Multiprozessor mit gemeinsamem Speicher

- Kohärenzprotokolle
  - Multiprozessor mit verteiltem gemeinsamem Speicher, Distributed Shared Memory (DSM)



Multiprozessor mit gemeinsamem Speicher

- DSM-Multiprozessoren



- Kohärenz-Protokolle
  - DSM-Multiprozessoren
    - Keine Möglichkeit, die Broadcast-Eigenschaft des Busses zu nutzen
    - Verzeichnisbasierte (tabellenbasierte) Cache-Kohärenzprotokolle (directory based)
      - Herstellen der Cache-Kohärenz über Verzeichnistabelle (directory tables)
      - In Hardware oder in Software implementiert
      - Zentrale oder verteilte Verwaltung
      - Die Tabelle protokolliert für jeden Blockrahmen im lokalen Speicher, ob dieser in den lokalen oder einen entfernten Cache-Speicher als Cache-Block übertragen worden ist. Festhalten der Zustände der Kopien
      - Zustände werden ähnlich denen des MESI-Protokolls definiert
    - Beispiele: SCI, SGI Origin, DASH

- Literatur
  - Hennessy, J.; Patterson, D.: Computer Architecture A Quantative Approach. Morgan Kaufmann Publishers, San Francisco, CA, 2003, 3. Auflage: Kap. 5.12 und 6.3
  - Flik, T.; Liebig, H.: Mikroprozessortechnik. Springer-Verlag, Heidelberg, 5. Auflage, 1998: Kap.:6.2.4