Rechnerstrukturen

Vorlesung im Sommersemester 2009

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

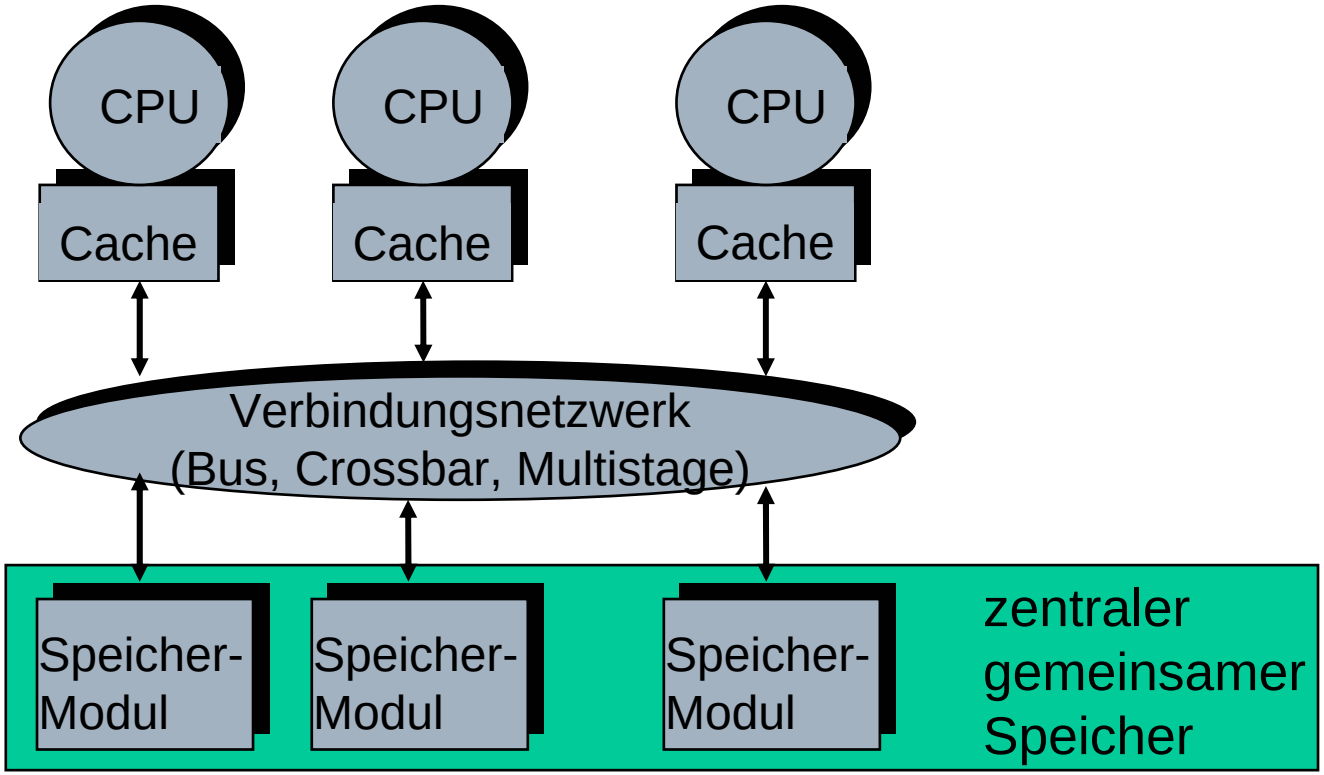
Fakultät für Informatik

Institut für Technische Informatik

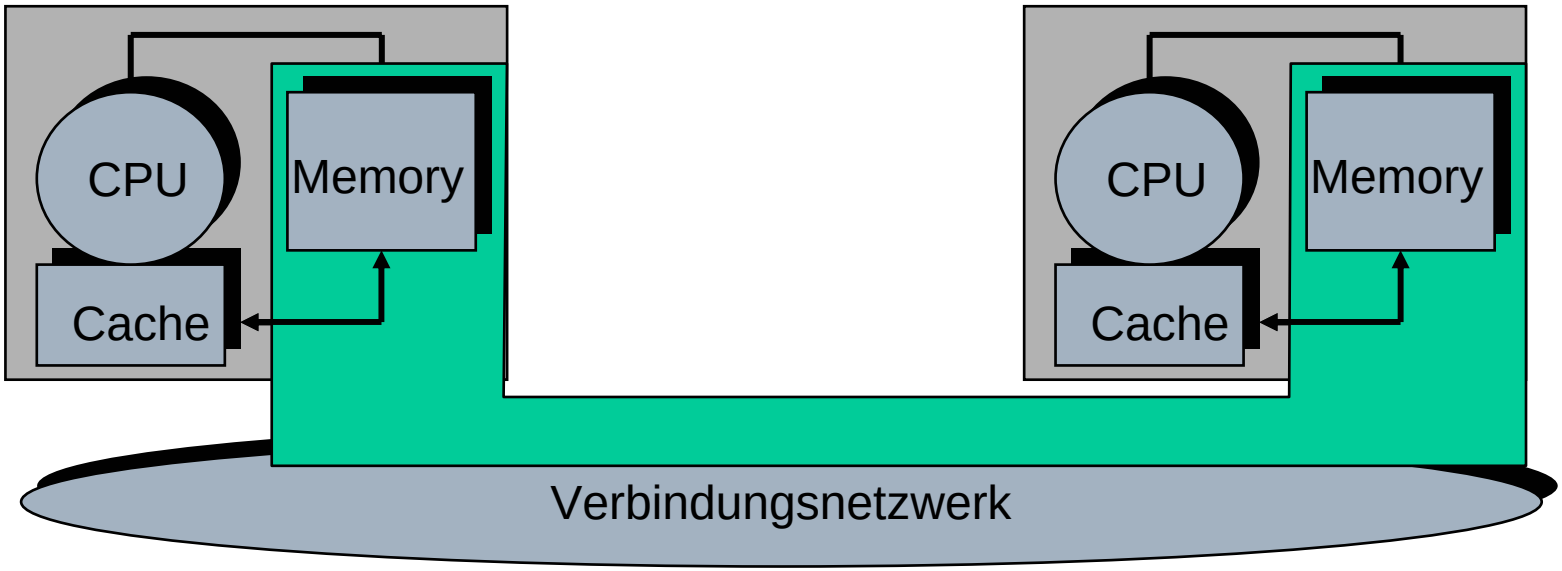
- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.6: Multiprozessoren mit gemeinsamem Speicher

- Multiprozessoren mit gemeinsamem Speicher



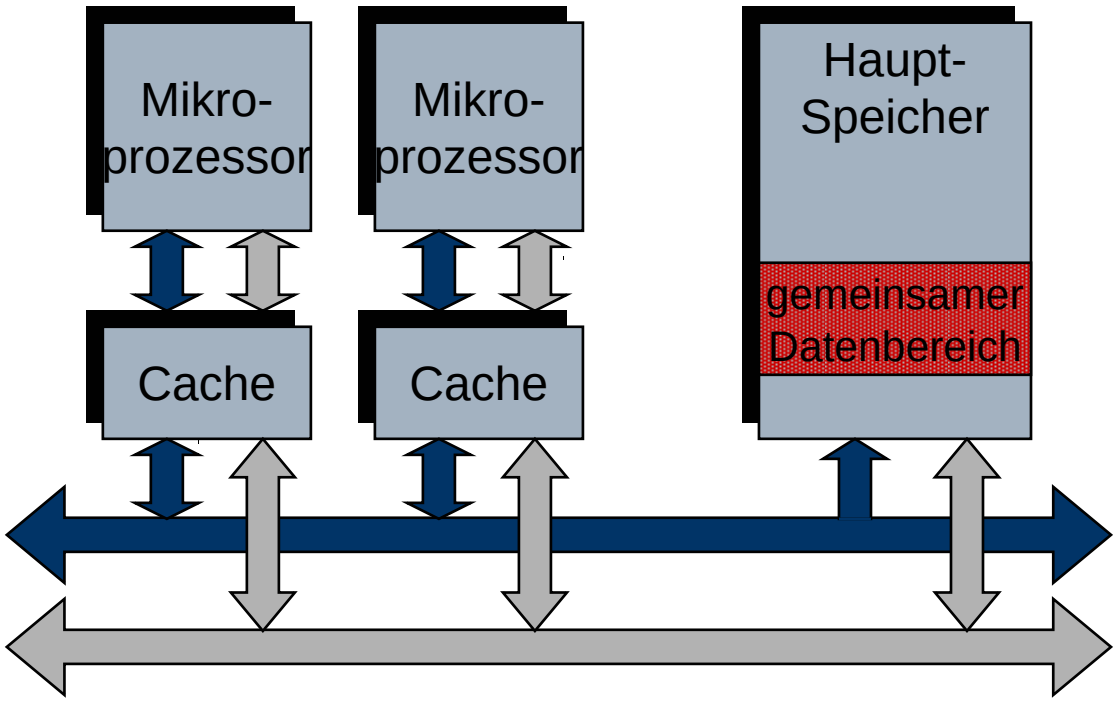
- Multiprozessoren mit verteiltem gemeinsamem Speicher



- Gültigkeitsproblem
  - wenn diese Prozessoren jeweils unabhängig voneinander auf Speicherwörter des Hauptspeichers zugreifen können.
  - Mehrere Kopien des gleichen Speicherwortes müssen miteinander in Einklang gebracht werden.
  - Eine Cache-Speicherverwaltung heißt cache-kohärent, wenn ein Lesezugriff immer den Wert des zeitlich letzten Schreibzugriffs auf das entsprechende Speicherwort liefert.

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.



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
  - 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

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

- 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.

Multiprozessor mit gemeinsamem Speicher

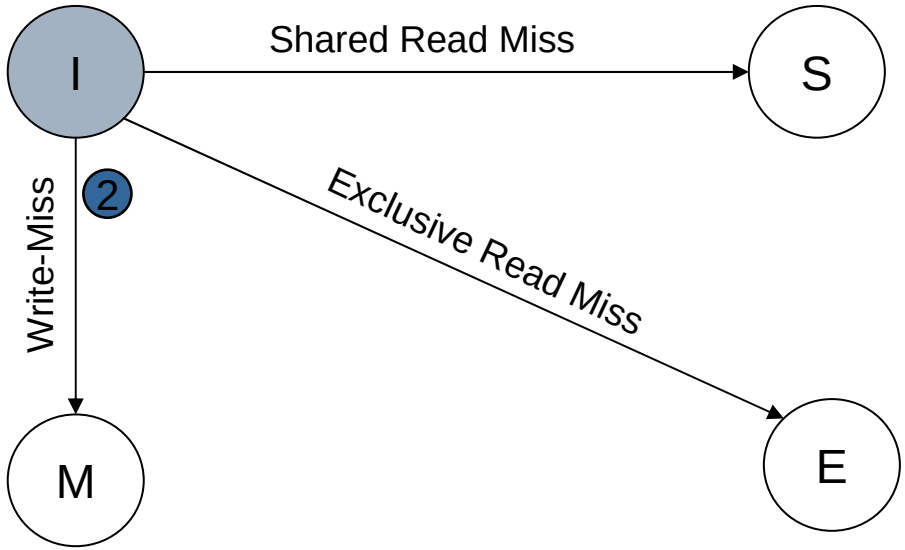
- MESI-Kohärenzprotokoll
  - Jede Cache-Zeile ist um zwei Statusbits erweitert
    - Zeigen Protokollzustände an:
      - Invalid (I)
      - Shared (S)
      - Exclusive (E)
      - Modified (M)
    - Beim Write-Through-Verfahren sind nur die Zustände Shared und Ivalidate relevant

Multiprozessor mit gemeinsamem Speicher

- 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.

Multiprozessor mit gemeinsamem Speicher

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



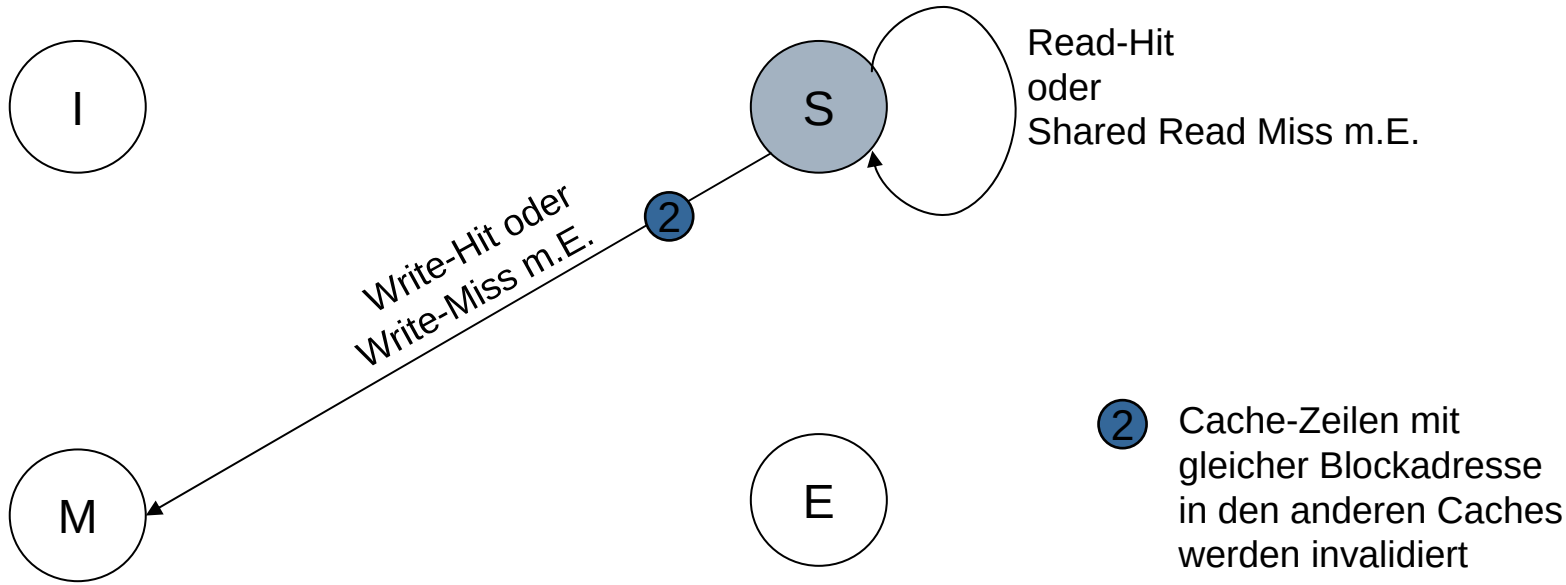
2 Cache-Zeilen mit gleicher Blockadresse in den anderen Caches werden invalidiert

Multiprozessor mit gemeinsamem Speicher

- 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).

Multiprozessor mit gemeinsamem Speicher

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

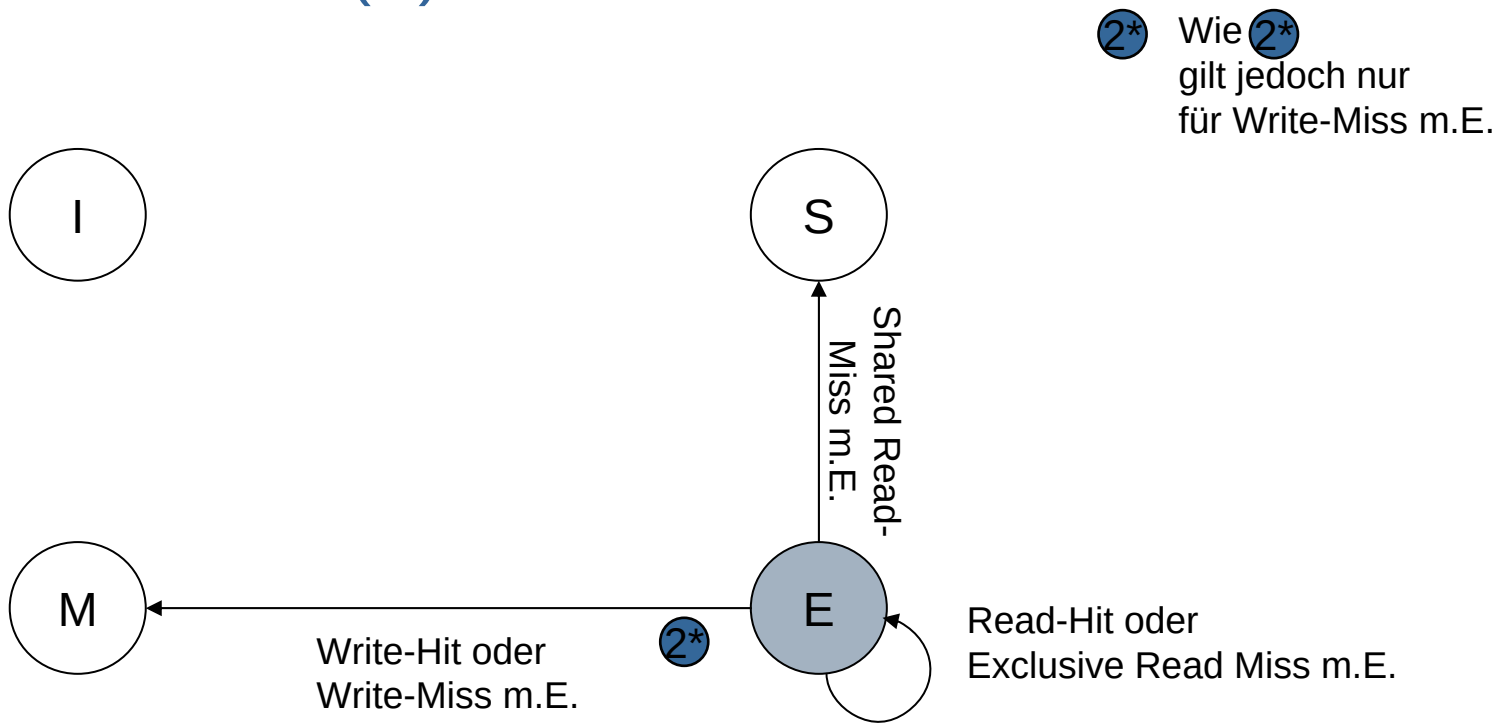


Multiprozessor mit gemeinsamem Speicher

- 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.

Multiprozessor mit gemeinsamem Speicher

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

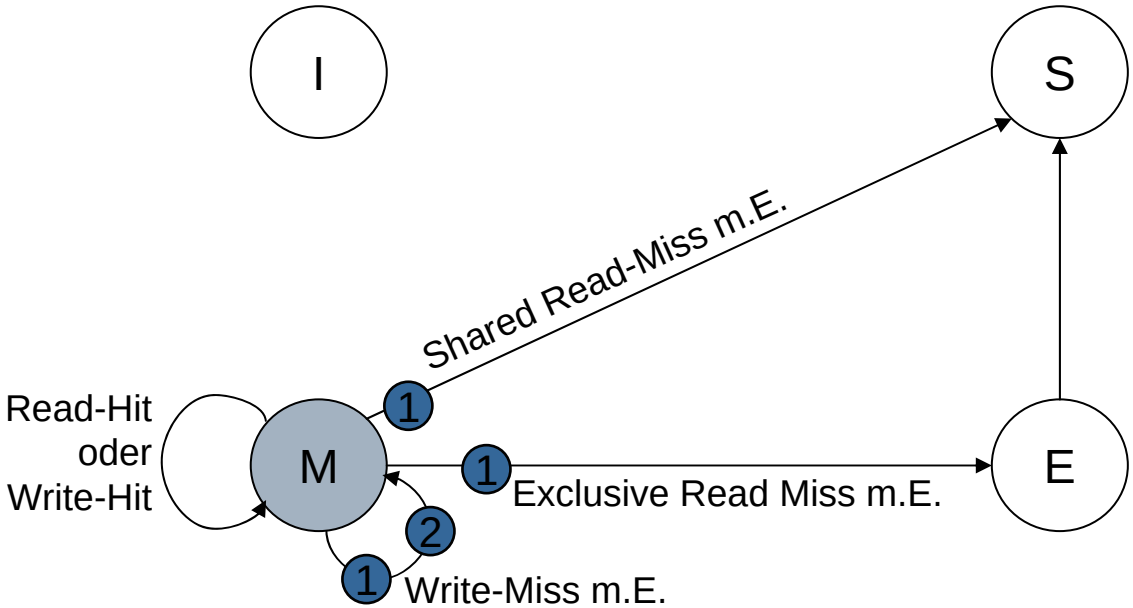


Multiprozessor mit gemeinsamem Speicher

- 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.

Multiprozessor mit gemeinsamem Speicher

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



1 Cache-Zeile wird in Den Hauptspeicher zurückkopiert

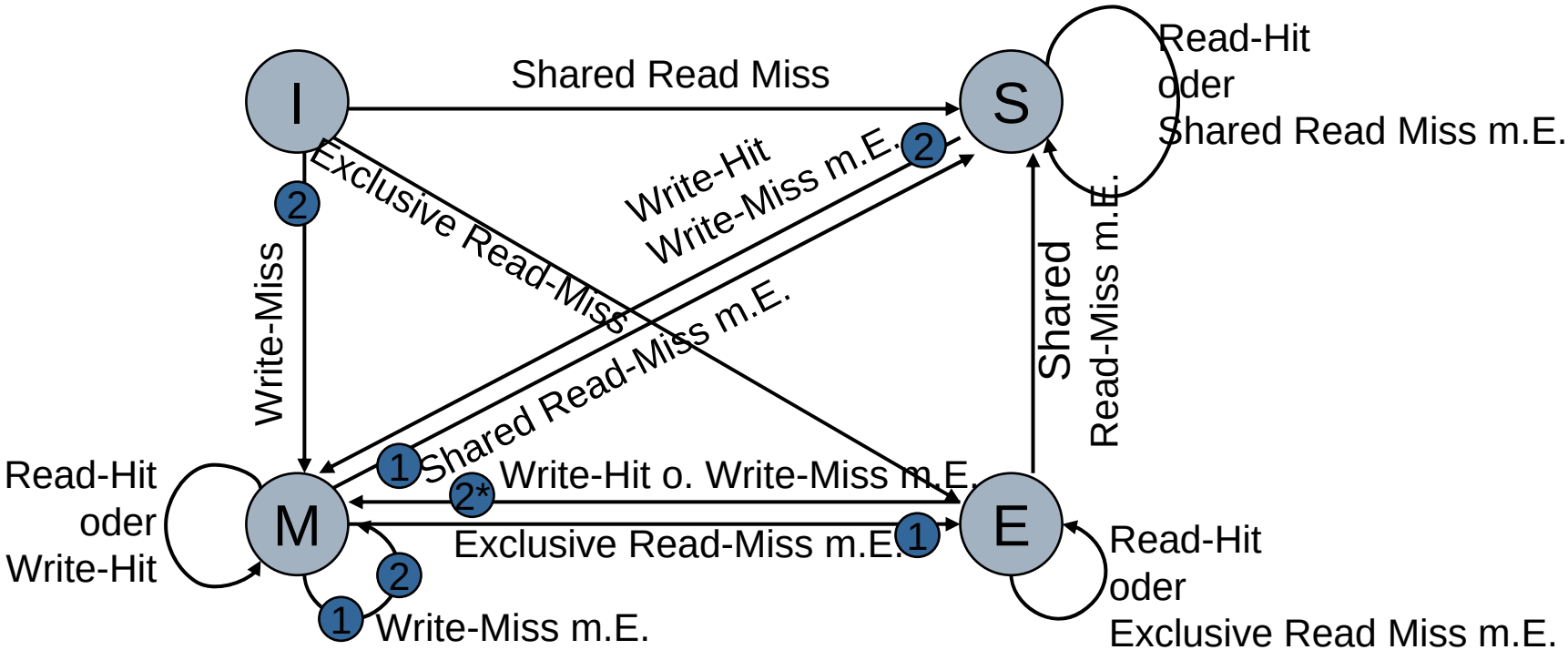
2 Cache-Zeilen mit gleicher Blockadresse in den anderen Caches werden invalidiert

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

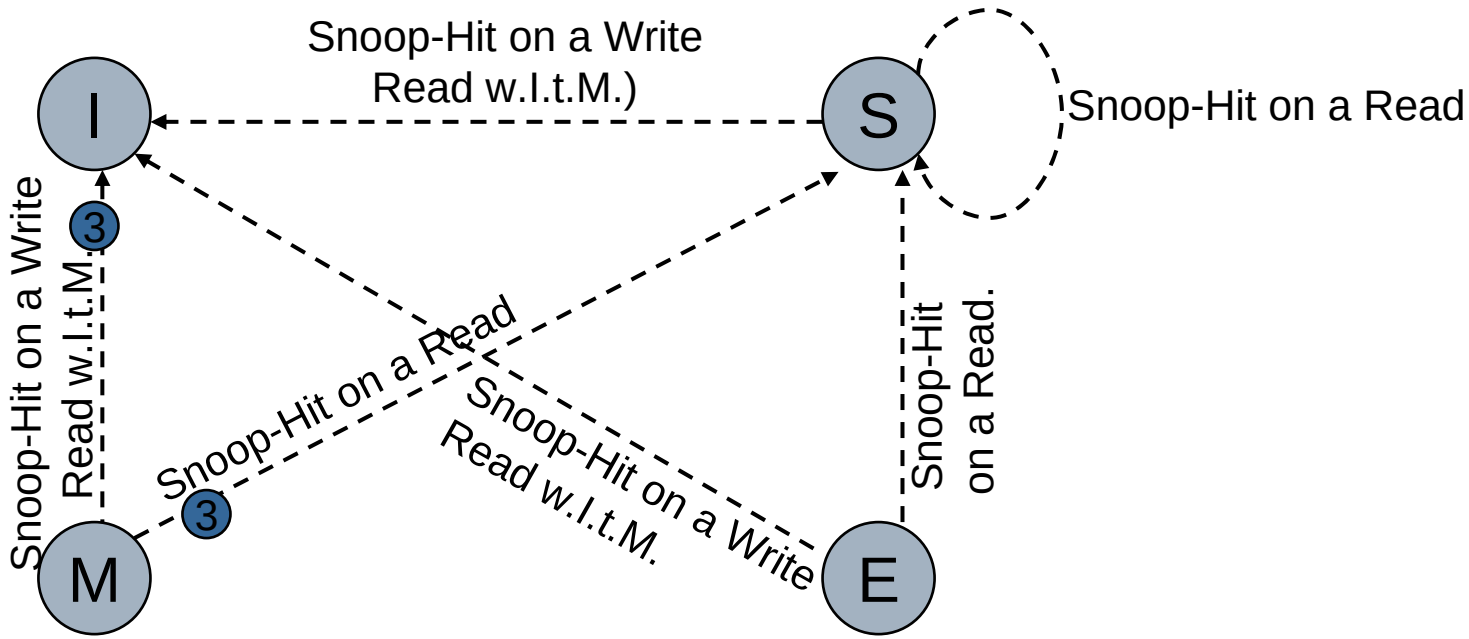
1 Cache-Zeile wird in den Hauptspeicher zurückkopiert. (Line-Flush)

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

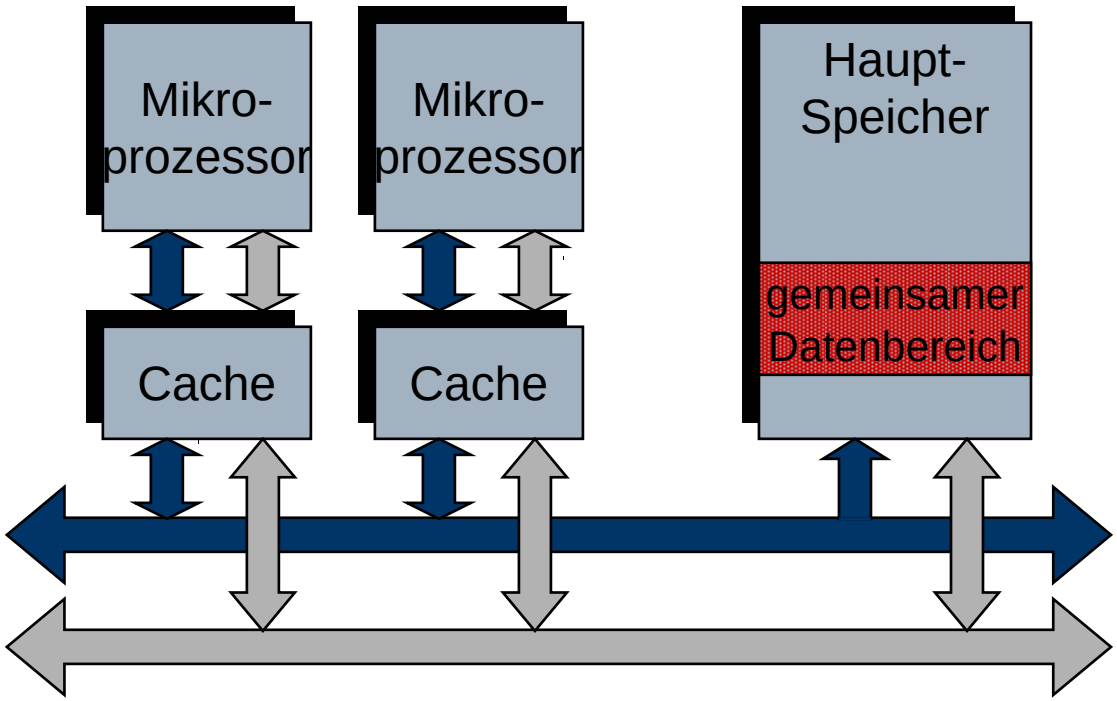
- Zustandsgraph 2 des MESI-Protokolls
  - Zustandsübergänge, die durch Aktionen, die auf dem Bus zu beobachten sind, ausgelöst werden.



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

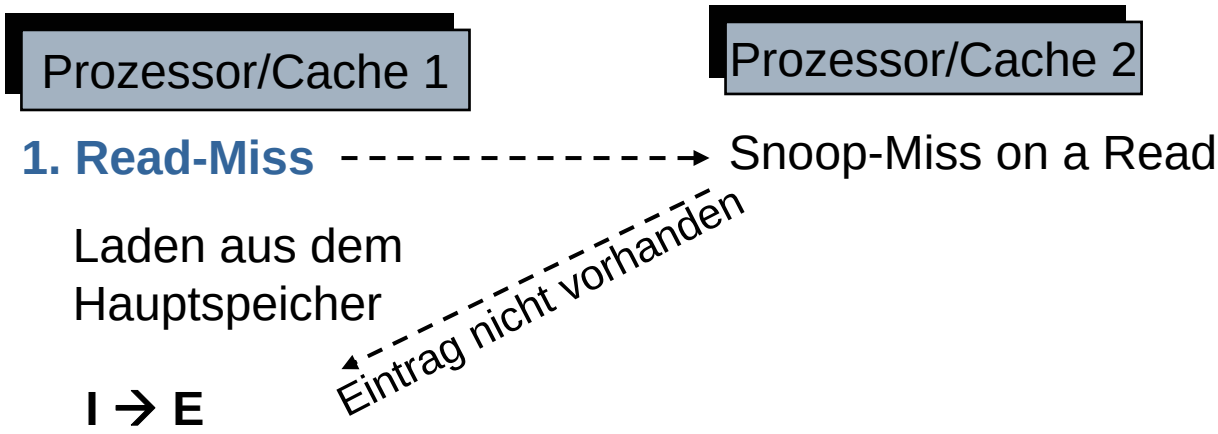
Multiprozessor mit gemeinsamem Speicher

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



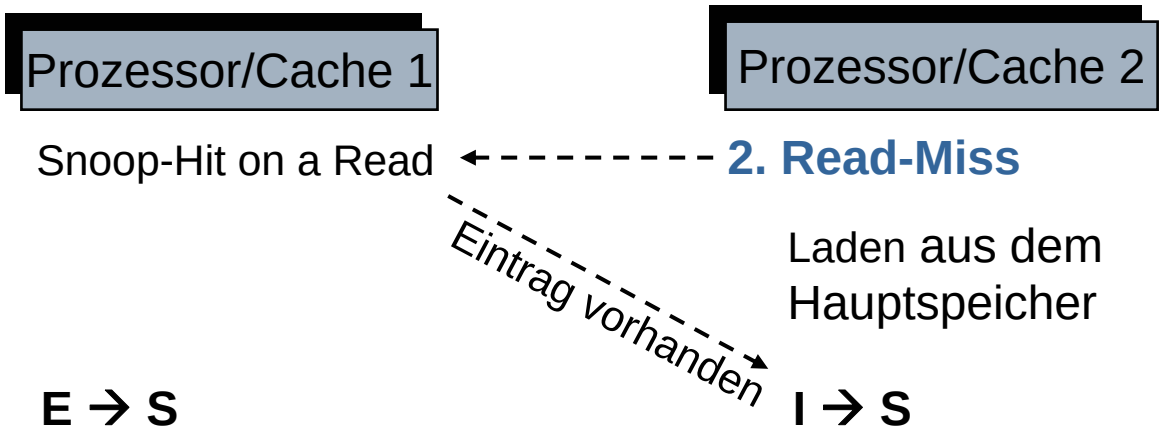
Multiprozessor mit gemeinsamem Speicher

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



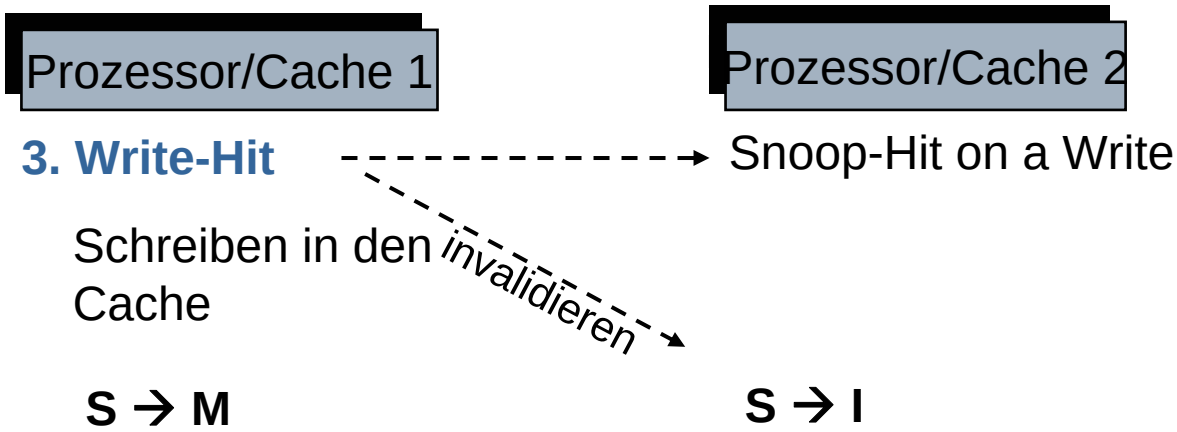
Multiprozessor mit gemeinsamem Speicher

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



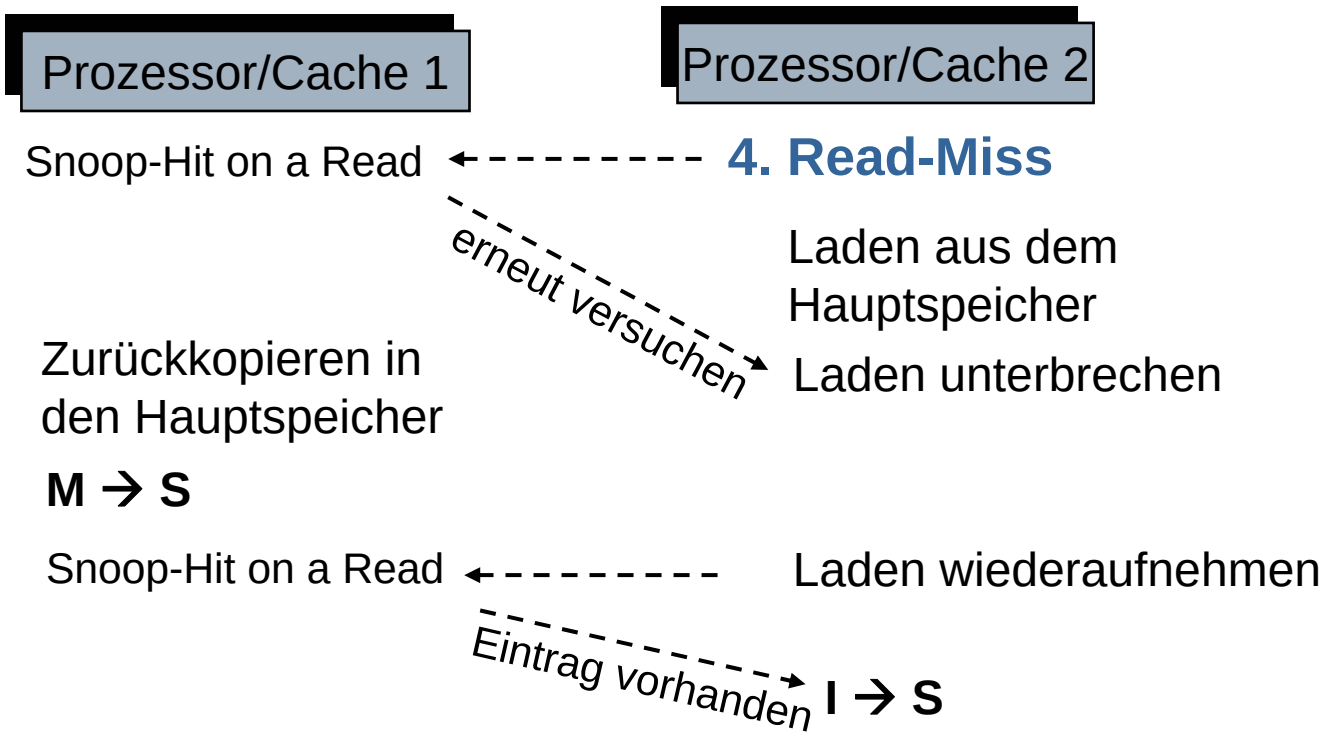
Multiprozessor mit gemeinsamem Speicher

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



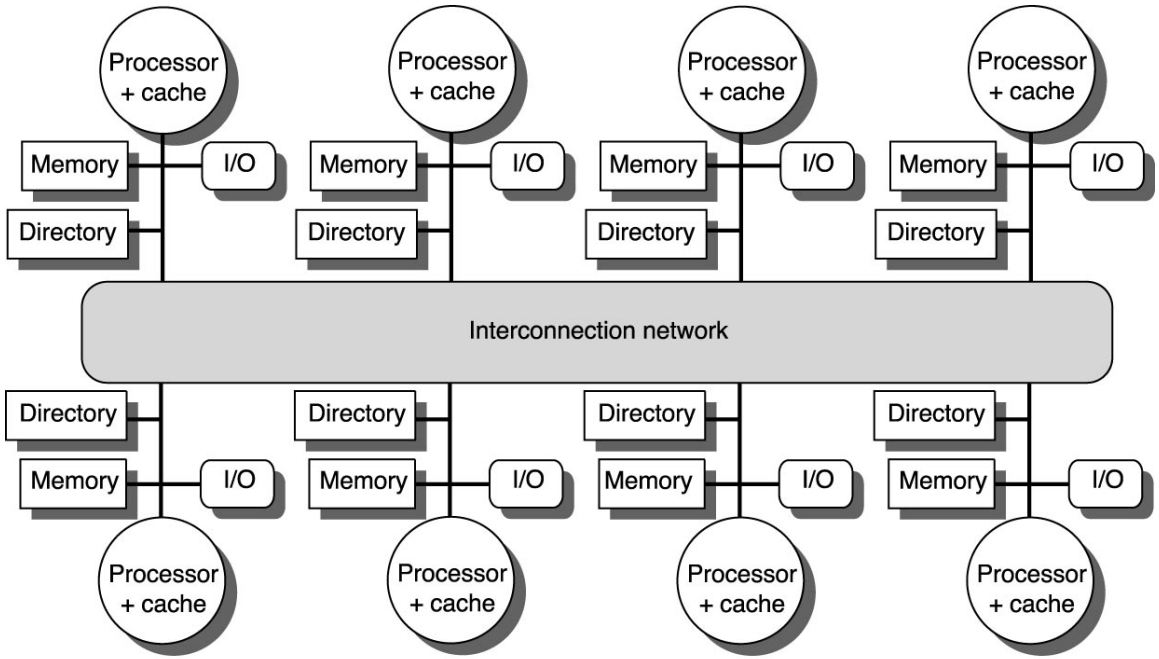
Multiprozessor mit gemeinsamem Speicher

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



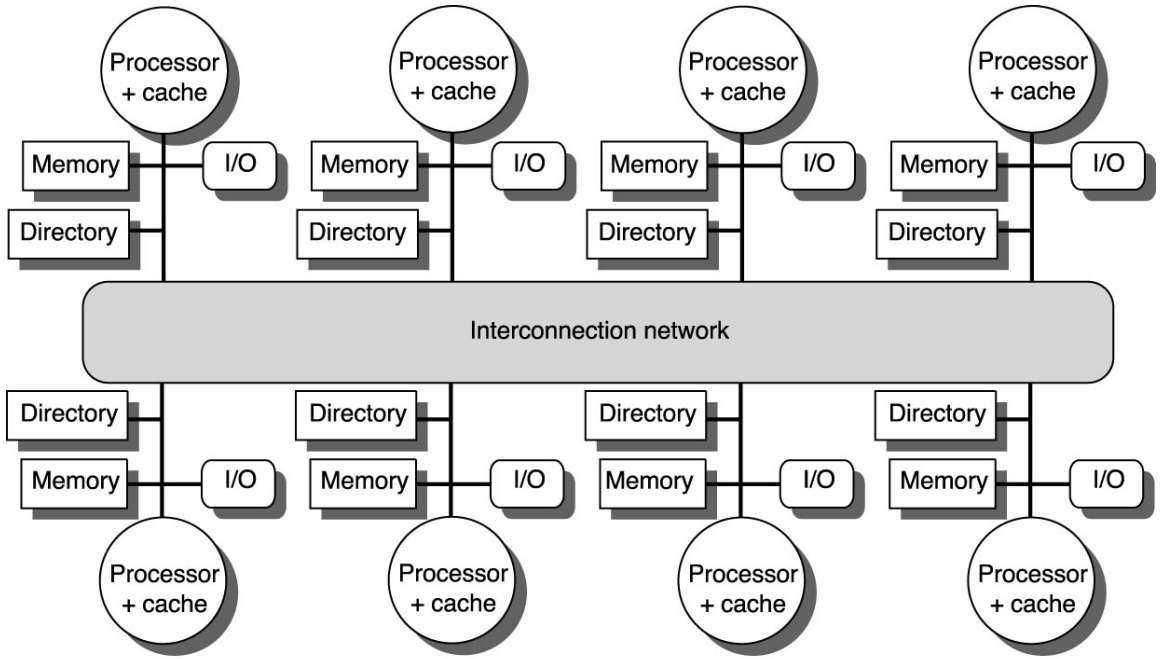
Multiprozessor mit gemeinsamem Speicher

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



Multiprozessor mit gemeinsamem Speicher

- DSM-Multiprozessoren



Multiprozessor mit gemeinsamem Speicher

- 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

Multiprozessor mit gemeinsamem Speicher

- 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

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Cache-Kohärenz
    - sichert, dass mehrere Prozessoren eine kohärente Sicht auf den Speicher haben
  - Wichtige Frage:
    - Wann muss ein Prozessor den Wert sehen, den ein anderer Prozessor aktualisiert hat?
    - Oder:
      - In welcher Reihenfolge muss ein Prozessor die Schreiboperationen eines anderen Prozessors beobachten?
    - Oder
      - Welche Bedingungen zwischen Lese- und Schreiboperationen auf verschiedene Speicherstellen durch verschiedene Prozessoren müssen gelten?

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Problem: Was kann passieren?

```
P1:     A=0;                          P2:     B=0;
        …                                     …
        A=1;                                  B=1;
L1:     if (B==0)                     L2:     if (A==0)
                …führe Aktion a2 aus                  …führe Aktion a1 aus
```

Mögliche Fälle:

a1 wird ausgeführt und a2 nicht
a2 wird ausgeführt und a1 nicht
a1 und a2 werden beide nicht ausgeführt
a1 und a2 werden beide ausgeführt

Ursache für dieses Verhalten:

- Verzögerungen der Schreiboperationen im Schreibpuffer
- Verzögerungen im Netzwerk

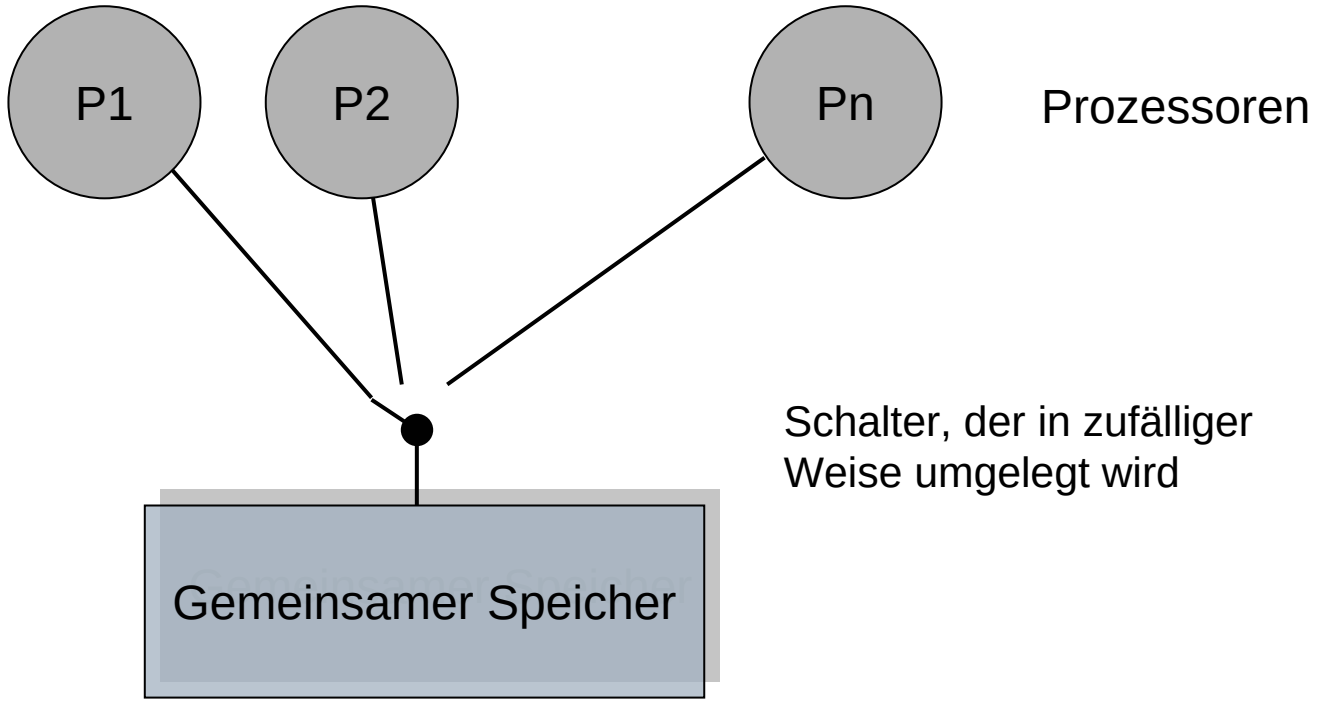
Soll dieses Verhalten erlaubt sein, und wenn ja, unter welchen Bedingungen?

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Spezifizieren die Reihenfolge, in der Speicherzugriffe eines Prozesses von anderen Prozessen gesehen werden
  - Sequentielle Konsistenz
    - Ein Multiprozessorsystem heißt sequentiell konsistent, wenn das Ergebnis einer beliebigen Berechnung dasselbe ist, als wenn die Operationen aller Prozessoren auf einem Einprozessorsystem in einer sequentiellen Ordnung ausgeführt würden. Dabei ist die Ordnung der Operationen der Prozessoren die des jeweiligen Programms.
    - Alle Lese- und Schreibzugriffe werden in einer beliebigen sequentiellen Reihenfolge, die jedoch mit den jeweiligen Programmordnungen konform ist, am Speicher wirksam.
    - Entspricht einer überlappenden sequentiellen Ausführung sequentieller Operationsfolgen anstelle einer parallelen Ausführung

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Sequentielle Konsistenz
    - Veranschaulichung der sequentiellen Konsistenz



Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Sequentielle Konsistenz
    - Programmierer geht von sequentieller Konsistenz aus
    - Führt aber zu sehr starken Einbußen bzgl. Implementierung und damit der Leistung
      - Verbietet vorgezogene Ladeoperationen, nichtblockierende Caches
  - Abgeschwächte Konsistenzmodelle
    - Konsistenz nur zum Zeitpunkt einer Synchronisationsoperation
      - Lese- und Schreiboperationen der parallel arbeitenden Prozessoren auf den gemeinsamen Speicher zwischen den Synchronisationszeitpunkten können in beliebiger geschehen.

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Definitionen
    - Ein Lesezugriff durch Prozessor Pi heißt zu einem bestimmten Zeitpunkt bezüglich Pk ausgeführt, wenn ein Schreibzugriff durch Prozessor Pk den Wert, den Pi durch den Lesezugriff auf dieselbe Adresse erhält, nicht mehr beeinflussen kann
    - Ein Schreibzugriff durch Pi heißt zu einem bestimmten Zeitpunkt bezüglich Pk ausgeführt, wenn ein Lesezugriff durch Pk auf dieselbe Adresse den Wert liefert, der von Pi geschrieben worden ist
    - Ein Zugriff gilt als ausgeführt, wenn er bezüglich aller Prozessoren im System ausgeführt ist
    - Ein Lesezugriff heißt global ausgeführt, wenn sowohl er als auch der Schreibzugriff, der den gelesenen Wert erzeugt, ausgeführt worden ist

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Definitionen
    - Unterschied zwischen ausgeführten und global ausgeführten Lesezugriff nur in Systemen, in denen der Schreibzugriff kein atomarer Vorgang ist, d.h. wenn der geschriebene Wert für alle Prozessoren des Systems nicht gleich lesbar ist
  - Sequentielle Konsistenz:
    - Hinreichende Bedingung:
      - Bevor ein Lese- oder Schreibzugriff bezüglich eines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Lesezugriffe global ausgeführt und alle vorhergehenden Schreibzugriffe ausgeführt sein
    - Reihenfolge der Operationen auf einem Prozessor wird beibehalten, und für alle anderen Prozessoren ist dieselbe Reihenfolge sichtbar.
    - Jeder Lese- und Schreibzugriff muss allen anderen Prozessoren vor einem nachfolgenden Lese- oder Aschreibzugriff bekannt gemacht werden
    - Wenig Spielraum für Optimierungen der Speicherzugriffe
      - Z.B. Puffern der Schreibzugriffe

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Prozessorkonsistenz
    - Definition (nach Goodman, 1989)
      - Ein Prozessor ist prozessorkonsistent, wenn das Ergebnis irgendeiner Ausführung dasselbe ist, als wenn die Operation eines jeden Prozessors in der sequentiellen Reihenfolge, die durch sein Programm bestimmt wird, erscheinen
    - Unterschied zur sequentiellen Konsistenz
      - Bei der Prozessorkonsistenz muss es nicht mehr für alle Prozessoren eine einheitliche Reihenfolge der Speicherzugriffe geben
      - Schreibzugriffe zweier Prozessoren können von einem dritten Prozessor in einer anderen Reihenfolge gesehen werden, als von den beiden schreibenden Prozessoren
      - Die Schreibzugriffe eines Prozessors werden jedoch von allen Prozessoren in der in seinem Programm angegebenen Reihenfolge gesehen

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Prozessorkonsistenz
    - Definition (nach Gharachorloo, 1990)
      - Bedingung der globalen Ausführung der Lesezugriffe wird fallen gelassen
      - Es gilt:
        - » Bevor ein Lesezugriff bezüglich eines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Lesezugriffe ausgeführt worden sein
        - » Bevor ein Schreibzugriff irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Schreibzugriffe ausgeführt worden sein

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Prozessorkonsistenz
    - Definition (nach Gharachorloo, 1990)
      - Konsequenz
        - » Prozessorkonsistenz führt nicht mehr unbedingt zur korrekten Sequentialisierung der Speicherzugriffe, da das Lesen schon erlaubt ist, bevor die vorhergehenden Schreibzugriffe alle ausgeführt worden sind.
        - » Lesezugriff kann zwar von den anderen Prozessoren nicht beobachtet werden, der Prozessor selbst sieht jedoch eine Reihenfolge der Speicheroperationen, die nicht seiner Programmordnung entspricht
        - » Schreibender Prozessor muss nicht auf die Ausführung des Schreibzugriffs bezüglich aller Prozessoren warten, bevor er eine weitere Schreiboperation veranlassen kann
        - » Puffern von Schreibzugriffen ist wieder erlaubt

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Schwache Konsistenz
    - Bisherige Konsistenzmodelle lassen Synchronisation paralleler Threads außer Acht
    - Konkurrierende Zugriffe auf gemeinsame Daten werden durch geeignete Synchronisationen geschützt

```
mutex m;
…
lock(m)
y=0;
x=0;
unlock(m)
…
P1:     lock(m)
        A=1;
        if (B==0)
                …führe Aktion a2 aus
        unlock(m)
```

```
P2:     lock(m)
        B=1;
        if (A==0)
                …führe Aktion a1 aus
        unlock(m)
```

Multiprozessor mit gemeinsamem Speicher

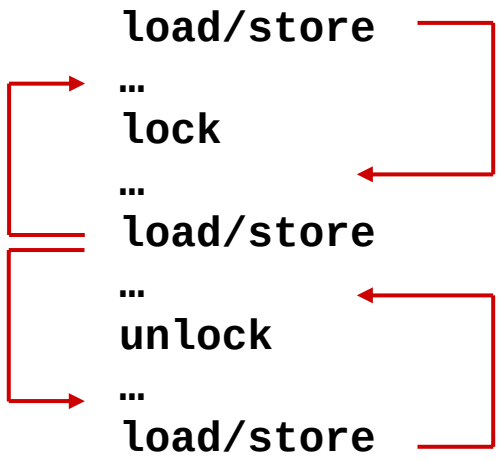
- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Idee
      - Die Konsistenz des Speicherzugriffs wird nicht mehr zu allen Zeiten gewährleistet, sondern zu bestimmten, vom Programmierer in das Programm eingesetzten Synchronisationspunkten
      - Einführung von kritischen Bereichen
        - » Innerhalb dieser Bereich wird die Inkonsistenz der gemeinsamen Daten zugelassen
        - » Voraussetzung: konkurrierende Lese-/Schreibzugriffe sind durch den kritischen Bereich unterbunden

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Bedingungen
      - Bevor ein Schreib- oder Lesezugriff bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Synchronisationspunkte erreicht worden sein
      - Bevor eine Synchronisation bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Schreib- oder Lesezugriffe ausgeführt worden sein.
      - Synchronisationspunkte müssen sequentiell konsistent sein
        - » *bevor* und *vorübergehend* beziehen sich auf die Programmordnung

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Auswirkung
      - Synchronisationsbefehle stellen Hürden dar, die von keinem Lese- oder Schreibzugriff übersprungen werden



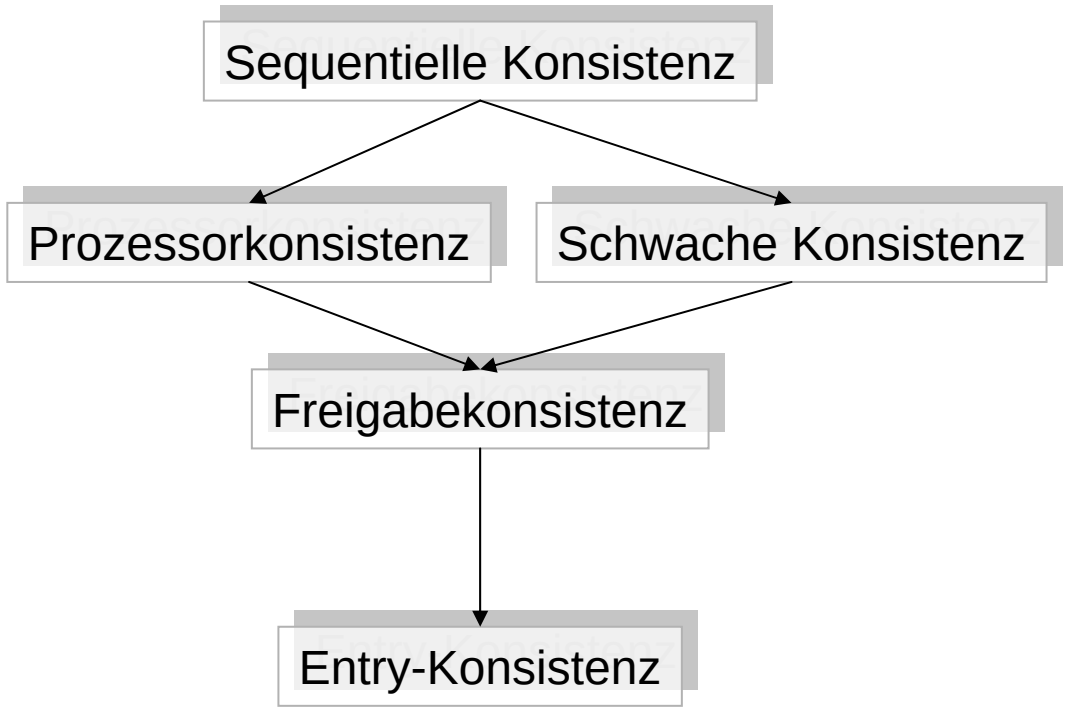
Rote Pfeile: verboten

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Auswirkung
      - Voraussetzung für die Implementierung der schwachen Konsistenz ist die hardware- und softwaremäßige Unterscheidung der Synchronisationsbefehle von den Lade- und Speicherbefehlen und eine sequentiell konsistente Implementierung der Synchronisationsbefehle
      - Das Puffern von Schreibzugriffen ist erlaubt, das von Synchronisationsbefehlen nicht!
      - „Hürdeneigenschaft“ der Synchronisationsbefehle
        - » In heutigen Mikroprozessoren mit Hilfe von Spezialbefehlen implementiert
      - Die Ordnung der Speicherbefehle wird durch die sehr viel losere Ordnung der Synchronisationsbefehle ersetzt

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Weitere Konsistenzmodelle
    - Zusammenhang



Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Weitere Konsistenzmodelle
    - definieren theoretisch weitere Abschwächungen oder
    - leiten sich aus Hardware-Implementierungen spezieller Prozessoren oder Multiprozessorsysteme ab
      - SPARC-Prozessor-spezifisches schwaches Konsistenzmodell TSO (total store order) oder das hiervon abgeleitete PSO (patial total store)