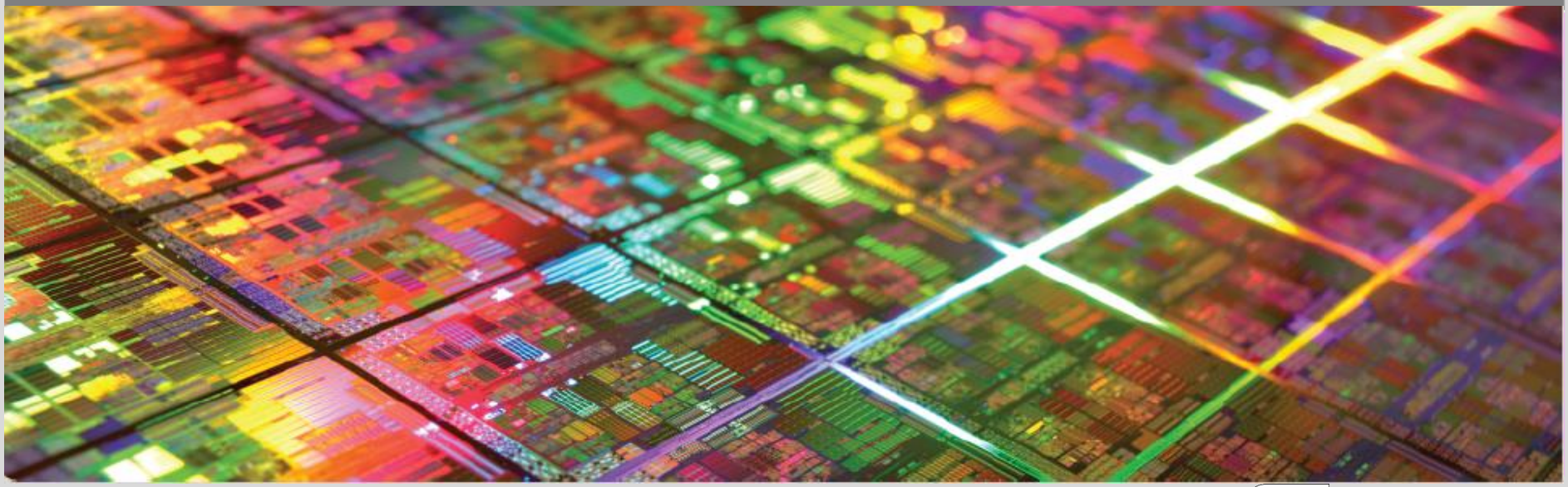
Karlsruher Institut für Technologie

Rechnerstrukturen

Vorlesung im Sommersemester 2016

Prof. Dr. Wolfgang Karl

Institut für Technische Informatik (ITEC), Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung



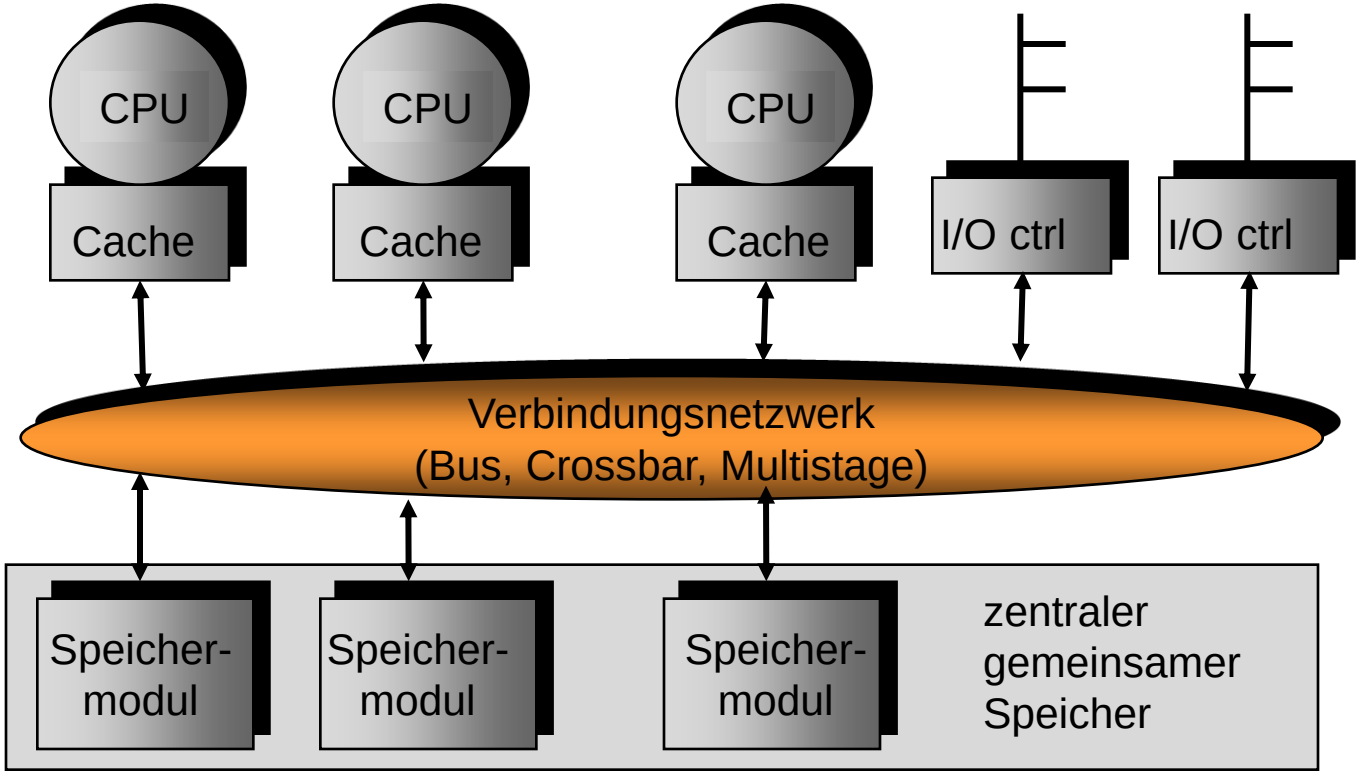
Vorlesung Rechnerstrukturen

Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation
- 3.2 Allgemeine Grundlagen
- 3.3 Parallele Programmierung
- 3.4 Quantitative Maßzahlen
- 3.5 Verbindungsstrukturen
- 3.6 Multiprozessoren mit gemeinsamem Speicher

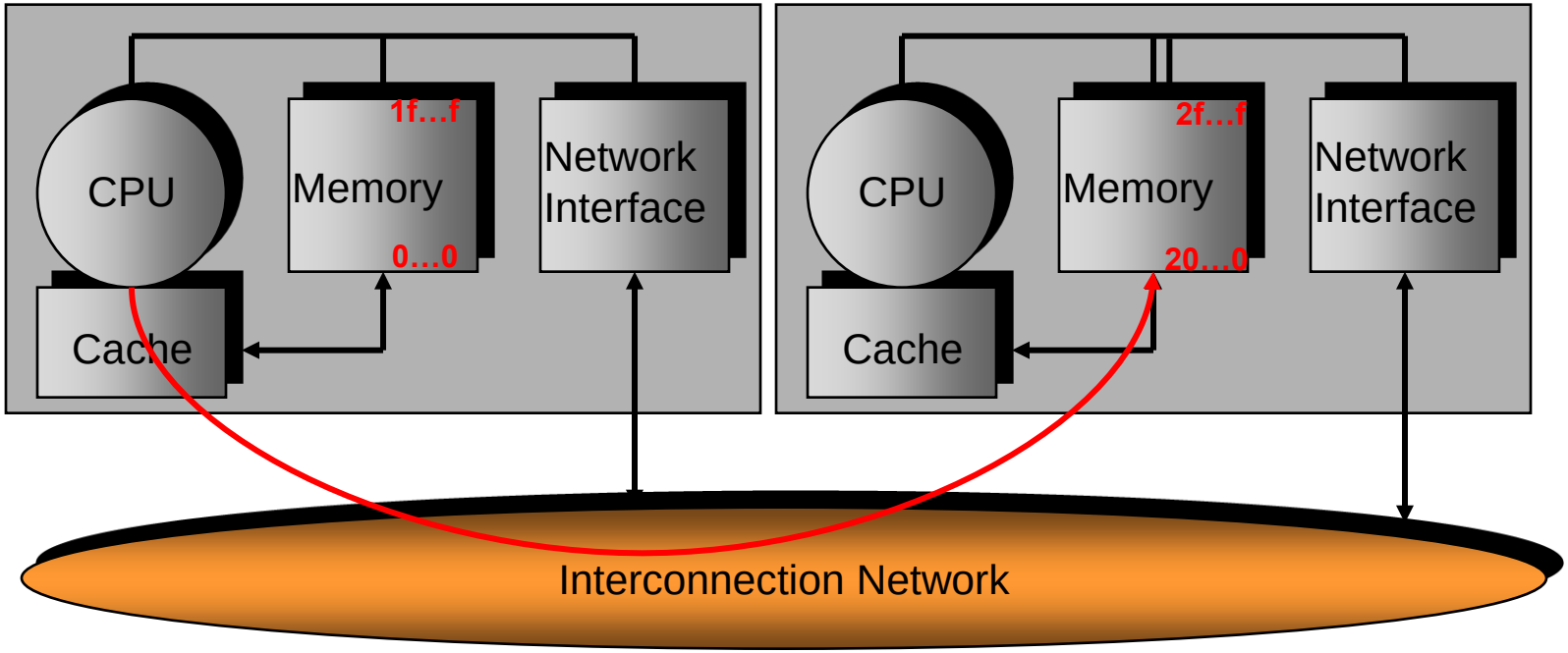
Multiprozessorsysteme

- **Multiprozessor mit gemeinsamem Speicher**
  - UMA: Uniform Memory Access
  - Beispiel: symmetrischer Multiprozessor (SMP)
    - Gleichberechtigter Zugriff der Prozessoren auf die Betriebsmittel



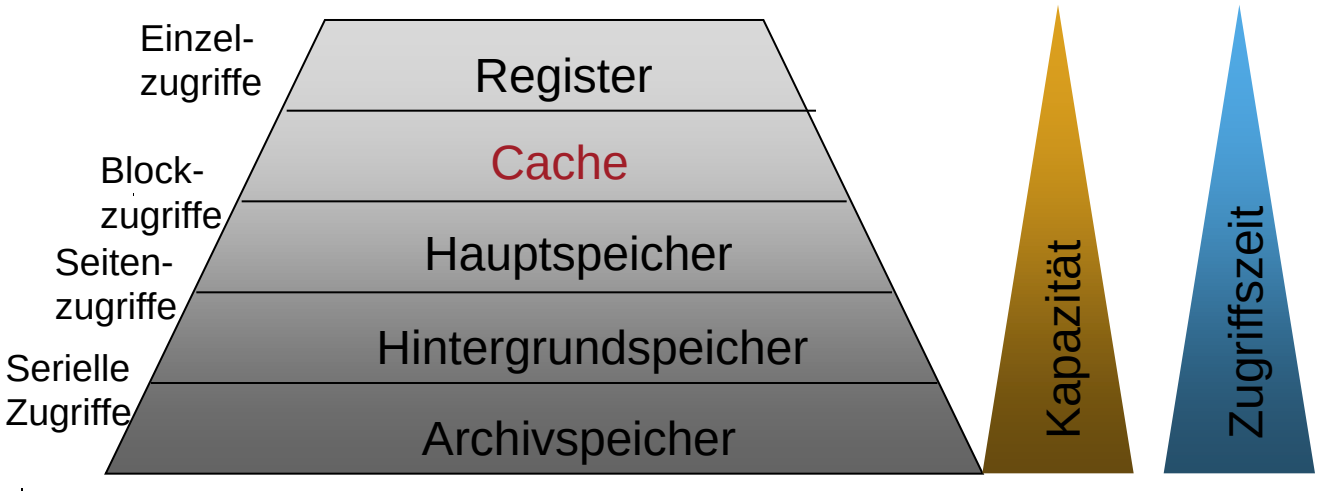
Multiprozessorsysteme

- **Multiprozessor mit verteiltem gemeinsamen Speicher**
  - NUMA: Non-Uniform Memory Access
  - Globaler Adressraum: Zugriff auf entfernten Speicher



Wiederholung: Cache-Speicher

- **Speicherhierarchie**
  - Ausnützen der Lokalitätseigenschaft von Programmen
  - Kompromiss zwischen Preis und Leistungsfähigkeit
  - Speicherkomponenten mit unterschiedlichen Geschwindigkeiten und Kapazitäten



Wiederholung: Cache-Speicher

Definitionen und Eigenschaften

- **Cache-Speicher**
  - Pufferspeicher mit schnellem Zugriff
  - Wichtigste Anwendung:
    - Pufferspeicher zwischen Hauptspeicher und Prozessor
    - Stellt die während einer Programmausführung jeweils aktuellen Hauptspeicherinhalte für Prozessorzugriffe als Kopien möglichst schnell zur Verfügung.
  - Weiteres Beispiel:
    - Translation-Lookaside Buffer (TLB)

Wiederholung: Cache-Speicher

Definitionen und Eigenschaften

- **Cache-Speicher-Steuerung /Cache-Controller**
  - Sorgt dafür, dass der Cache-Speicher in der Regel das Datum enthält, auf das der Prozessor als nächstes zugreift.
- Besondere **Strategien** für das
  - Laden
  - Aktualisieren und Adressieren des Inhalts
- Speicherung von Befehlen / Daten
  - Gemeinsam in einem Cache oder
  - Getrennt jeweils in einem Befehls- und Daten-Cache
    - Harvard-Architektur
- **Cache-Hierarchie**
  - Cache-Speicher auf mehreren Ebenen
- **Inklusionseigenschaft**
  - Inhalt des auf höchster Stufe stehenden Cache-Speichers auf in den Cache-Speichern auf niedrigerer Ebene:
  - (L1$) < (L2$) < ...

Wiederholung: Cache-Speicher

Definitionen und Eigenschaften

- **Blockrahmen, Zeile (Block-Frame, Cache-Line)**
  - Datenteil
  - Folge von Speicherwörtern im Cache-Speicher
  - Blocklänge in Bytes bestimmt die Länge der Zeile
  - Anzahl der Speicherplätze in einem Blockrahmen
- **Adressetikett (Adress-Tag):**
  - Verbunden mit Blockrahmen
  - Enthält den gemeinsamen Adressteil der in einer Zeile gespeicherten Datenkopien.
- **Statusbits**
  - Valid-Bit
    - Gültigkeitsbit zeigt an, ob Cache-Zeile gültige Kopien enthält
  - Dirty-Bit
    - Zeigt für Daten-Caches an, ob die Daten in der Cache-Zeile verändert worden sind

Wiederholung: Cache-Speicher

Arbeitsweise

- Cache-Steuerung prüft bei Speicherzugriffen des Mikroprozessors, ob
  - der zur Speicheradresse gehörende Hauptspeicherinhalt als Kopie im Cache steht (Bedingung 1) und
  - dieser Cache-Eintrag durch das Valid-Bit als gültig gekennzeichnet ist (Bedingung 2).
  - Treffer (Cache-Hit):
    - Beide Bedingungen sind erfüllt
  - Fehlzugriff (Cache-Miss):
    - Eine der beiden Bedingungen ist nicht erfüllt

Wiederholung: Cache-Speicher

Arbeitsweise

- **Cache-Fehlzugriff**
  - Aktionen bei Lesezugriffen (**read-miss**)
    - Lesen des Datums aus den niedrigeren Ebenen oder dem Hauptspeicher und Laden des Cache-Speichers
    - Kennzeichnen der Cache-Eintrages als gültig (Setzen des Valid-Bits)
    - Speichern der Adressinformation im Adressteil des Cache-Speichers
  - Aktionen bei Schreibzugriffen (**write-miss**):
    - Aktualisierungsstrategie bestimmt, ob
    - der entsprechende Block in Cache geladen und dann mit dem zu schreibenden Datum aktualisiert wird, oder ob
    - nur der Hauptspeicher aktualisiert wird und der Cache unverändert bleibt.
- **Cache-Treffer**
  - Zugriff erfolgt auf Cache

Wiederholung: Cache-Speicher

Cache-Organisation

- Wohin wird ein Block im Cache geladen?
  - **Direkte Abbildung (direct mapped)**

Hauptspeicher
Block B_j , j= 0,1, …,n-1
Kapazität: n * b = 2^{s+w} Wörter

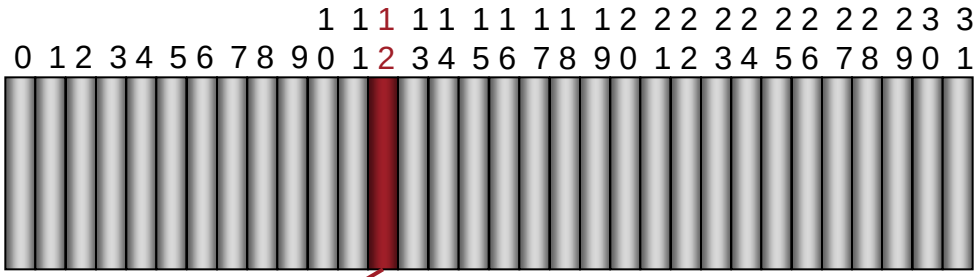
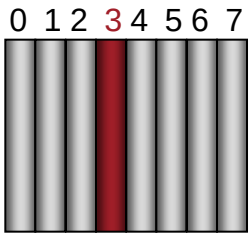


Abbildung von $\{B_j\}$ nach $\{Z_i\}$



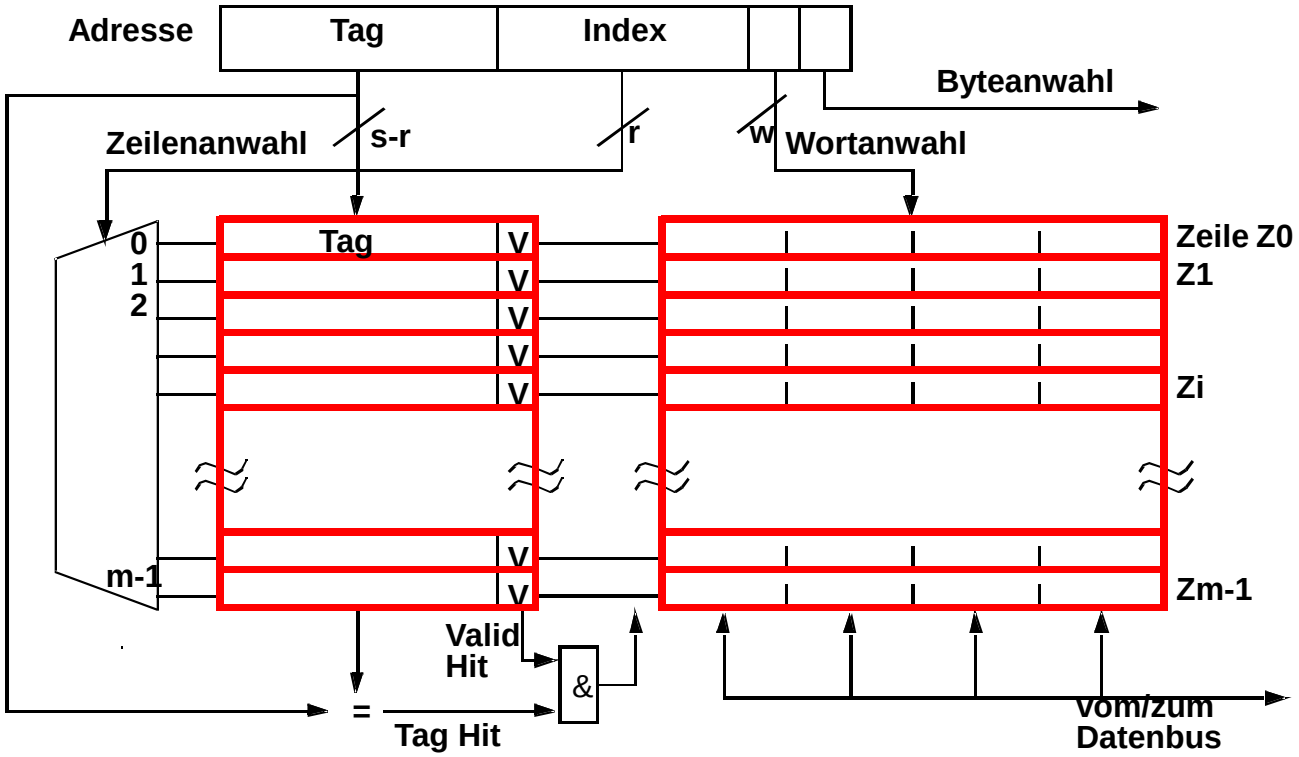
Cache
Zeile Z_i , i= 0,1, …,m-1
Kapazität: m * b = 2^{r+w} Wörter

Legende:
n >> m, n = 2^s, m= 2^r
Jeder Block enthält b Wörter mit b= 2^w

Wiederholung: Cache-Speicher

Cache-Organisation

- **Direct-Mapped Cache**



Wiederholung: Cache-Speicher

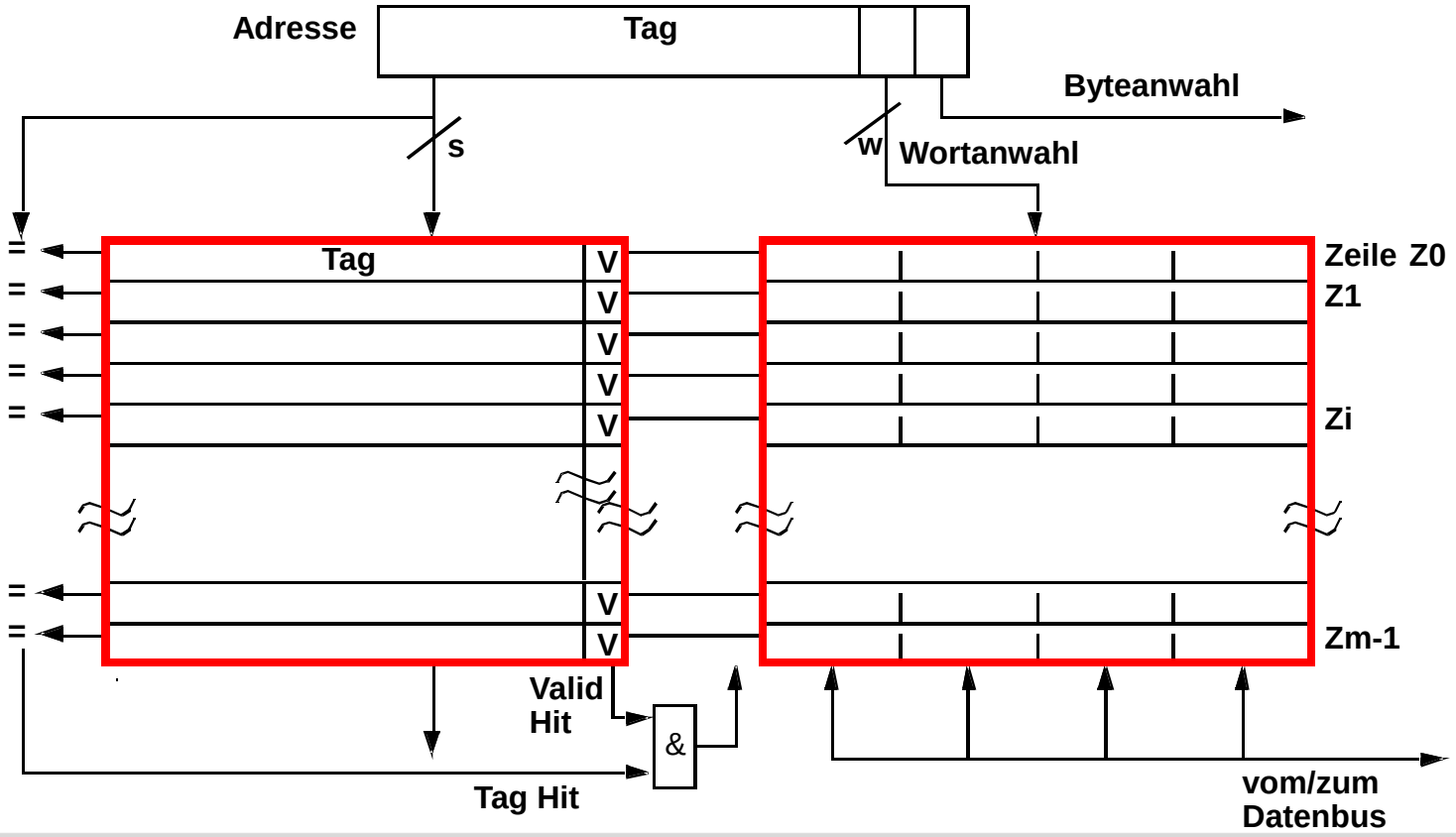
Cache-Organisation

- **Vollassoziativer Speicher**
  - jeder Block des Hauptspeichers kann auf jede Zeile des Cache-Speichers abgebildet werden (Flexibilität)
  - Ersetzungsstrategie gibt vor, welche Zeile beim Laden ersetzt werden soll (z.B. Least-Recently-Used)
  - hoher Hardware-Aufwand (Anzahl Vergleicher = Anzahl Zeilen)

Wiederholung: Cache-Speicher

Cache-Organisation

- Vollassoziativer Speicher



Wiederholung: Cache-Speicher

Cache-Organisation

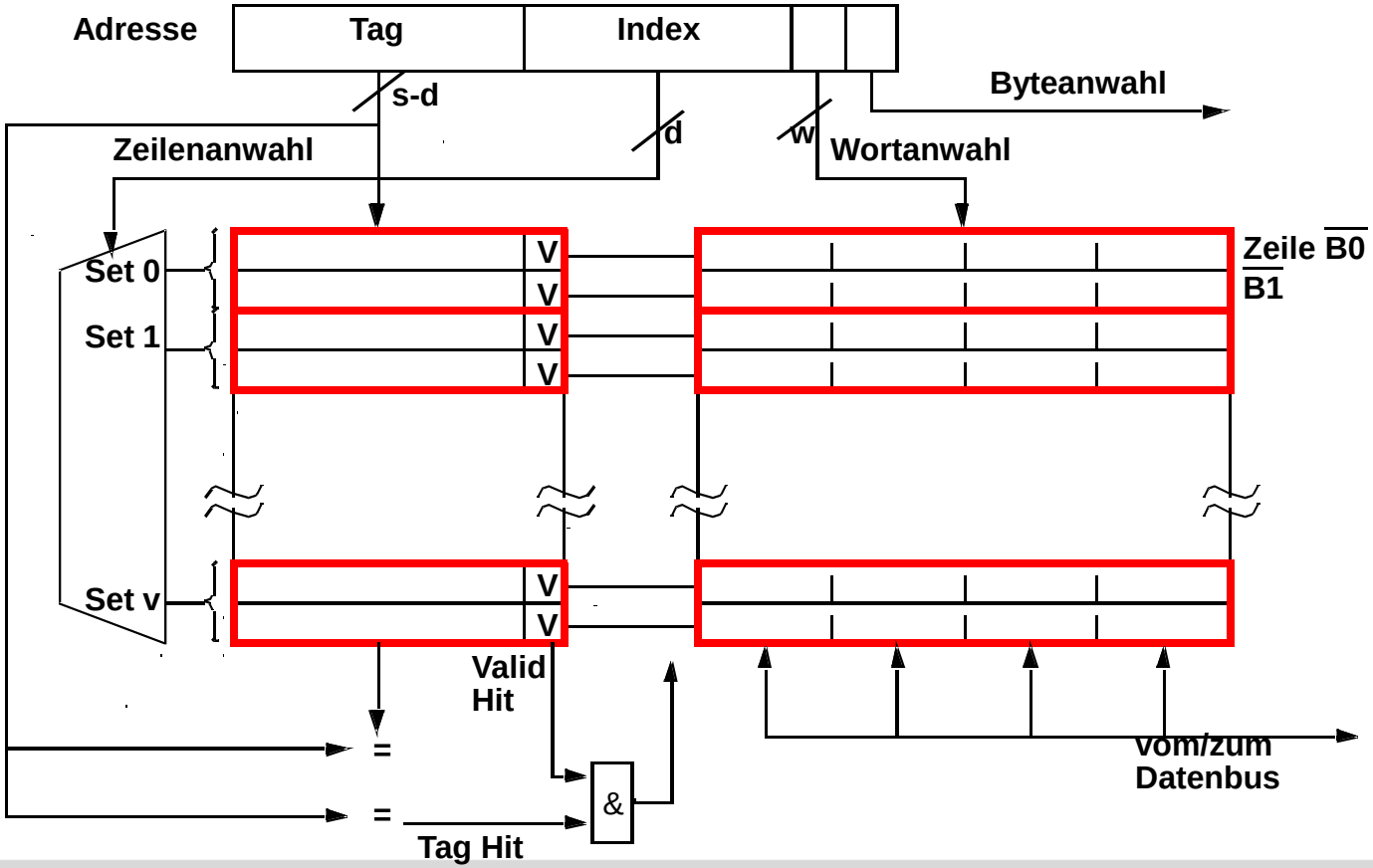
- **k-fach satzassoziativer Speicher**
  - Kompromiss zwischen Direct-Mapped- und vollassoziativem Cache;
  - Zusammenfassen von k Zeilen zu einer Menge (set)
  - Aufteilen der m Cache-Zeilen in v = m/k Mengen zu je k Zeilen;
  - jede Menge wird über eine d Bit breite Nummer identifiziert: $2^d = v$;
  - ein Block B_j kann in eine der verfügbaren Zeilen Z_f in einer Menge S_i abgebildet werden:

  - Bj --> $Z_f \varepsilon S_i$, mit j(mod v) = i.

Wiederholung: Cache-Speicher

Cache-Organisation

- k-fach satzassoziativer 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

Multiprozessorsysteme

Gültigkeitsproblem

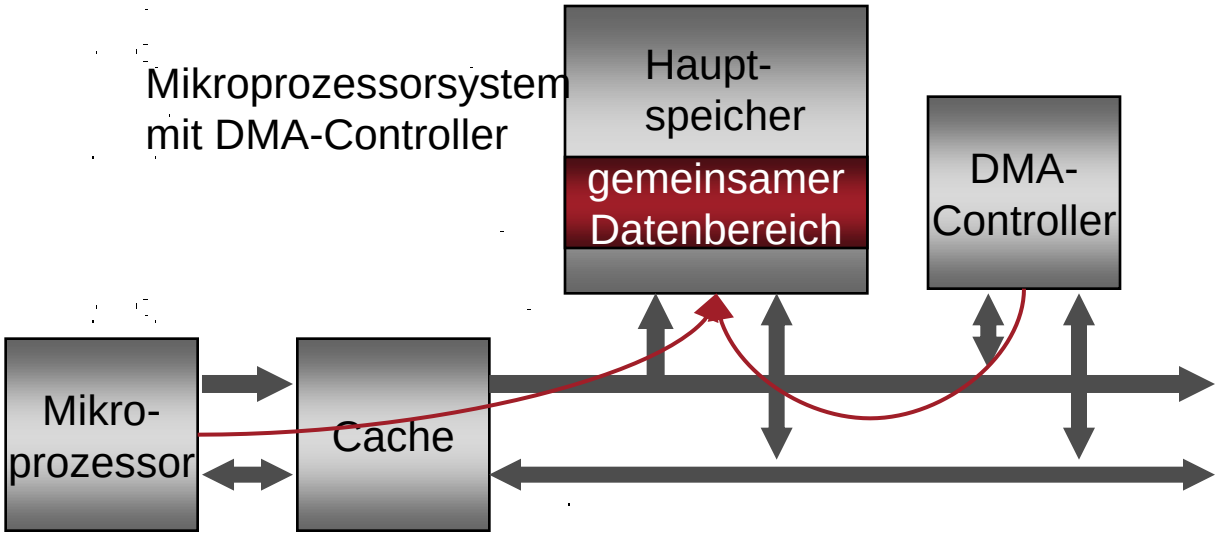
- wenn diese Prozessoren jeweils unabhängig voneinander auf Speicherwörter des Hauptspeichers zugreifen können.
- Mehrere Kopien des gleichen Speicherwortes müssen miteinander in Einklang gebracht werden.

- 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

- 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



Multiprozessor mit gemeinsamem Speicher

Cache-Kohärenz-Problem

- Mikroprozessorsystem mit DMA-Controller: 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

- Mikroprozessorsystem mit DMA-Controller: Zugriff auf veraltete Daten (stale data)
- 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

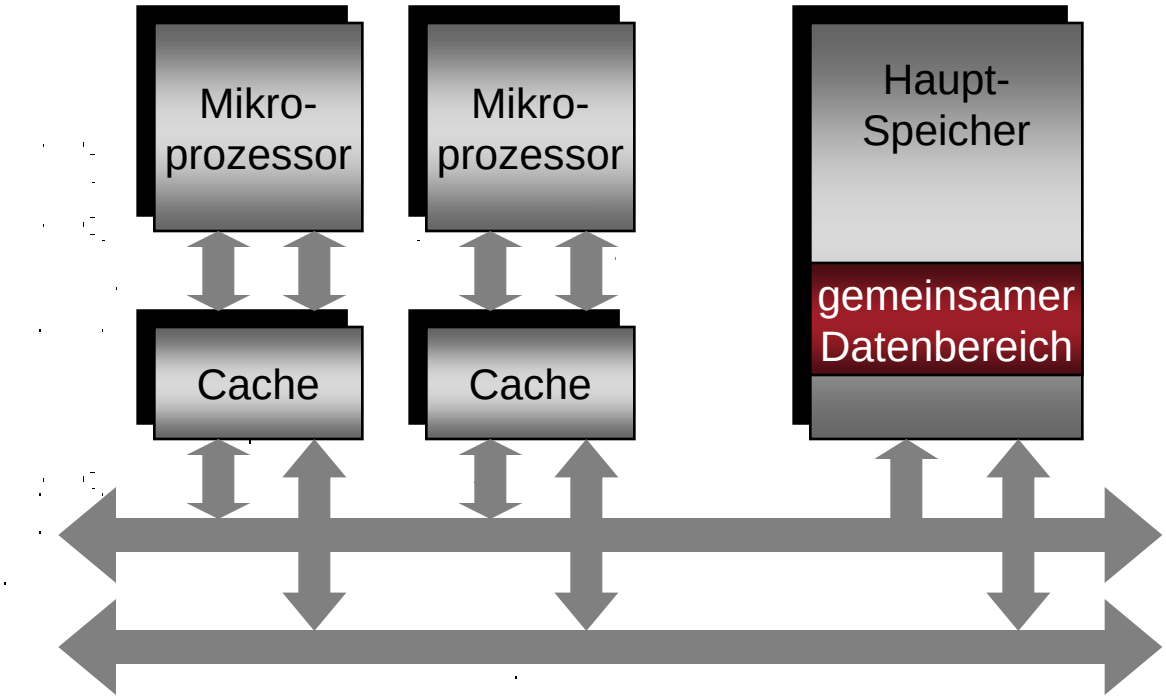
Cache-Kohärenz-Problem

- Mikroprozessorsystem mit DMA-Controller: Zugriff auf veraltete Daten (stale data)
- 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.



Multiprozessor mit gemeinsamem Speicher

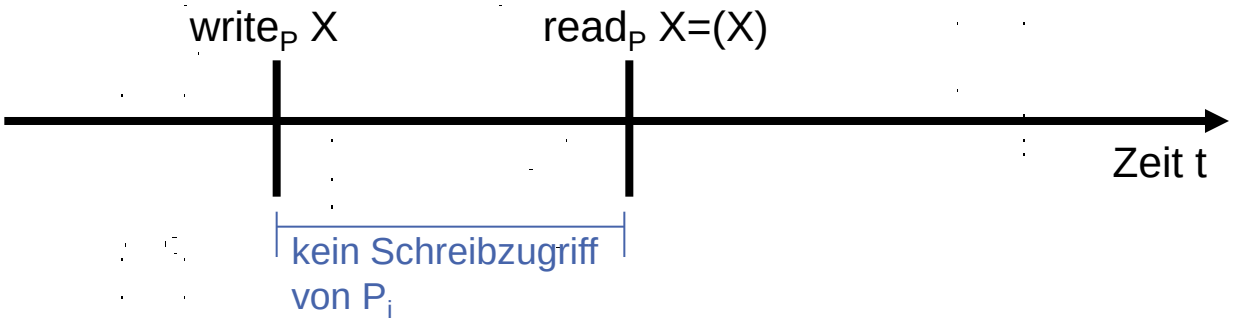
Cache-Kohärenz und Konsistenz

- 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

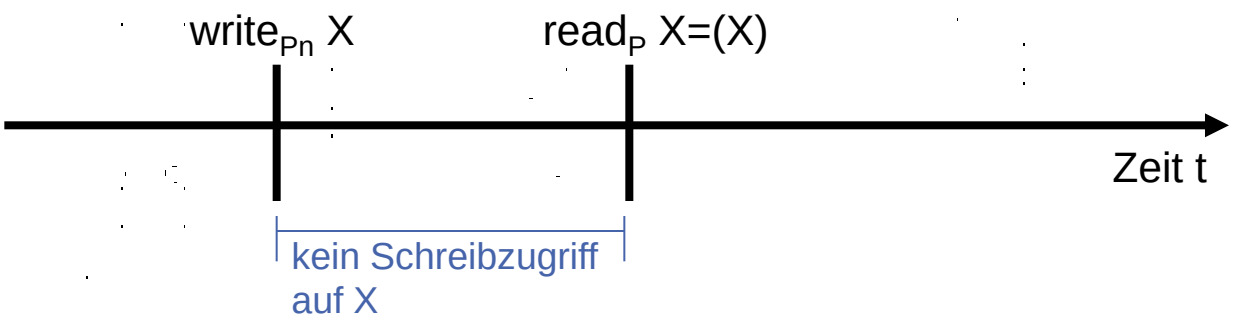
- Ein Speichersystem ist kohärent, wenn
  - Ein Lesezugriff eines Prozessors P auf eine Speicherstelle X, der einem Schreibzugriff von P auf die Stelle X folgt und keine Schreibzugriffe anderer Prozessoren zwischen dem Schreiben und dem Lesen von P stattfinden, liefert immer den Wert, den P geschrieben hat.
  - Einhaltung der Programmordnung



Multiprozessor mit gemeinsamem Speicher

Kohärenz

- Ein Speichersystem ist kohärent, wenn
  - 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



Multiprozessor mit gemeinsamem Speicher

Kohärenz

- Ein Speichersystem ist **kohärent**, wenn
  - Schreibzugriffe auf die eine Speicherzelle serialisiert werden; 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

- Ein paralleles Programm, das auf einem Multiprozessor läuft, kann 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

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:
      - erfordert 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

- **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

Kohärenz-Protokolle

- **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

Kohärenz-Protokolle

- **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

Kohärenz-Protokolle

- **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

Kohärenz-Protokolle

- **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

Kohärenz-Protokolle

- **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

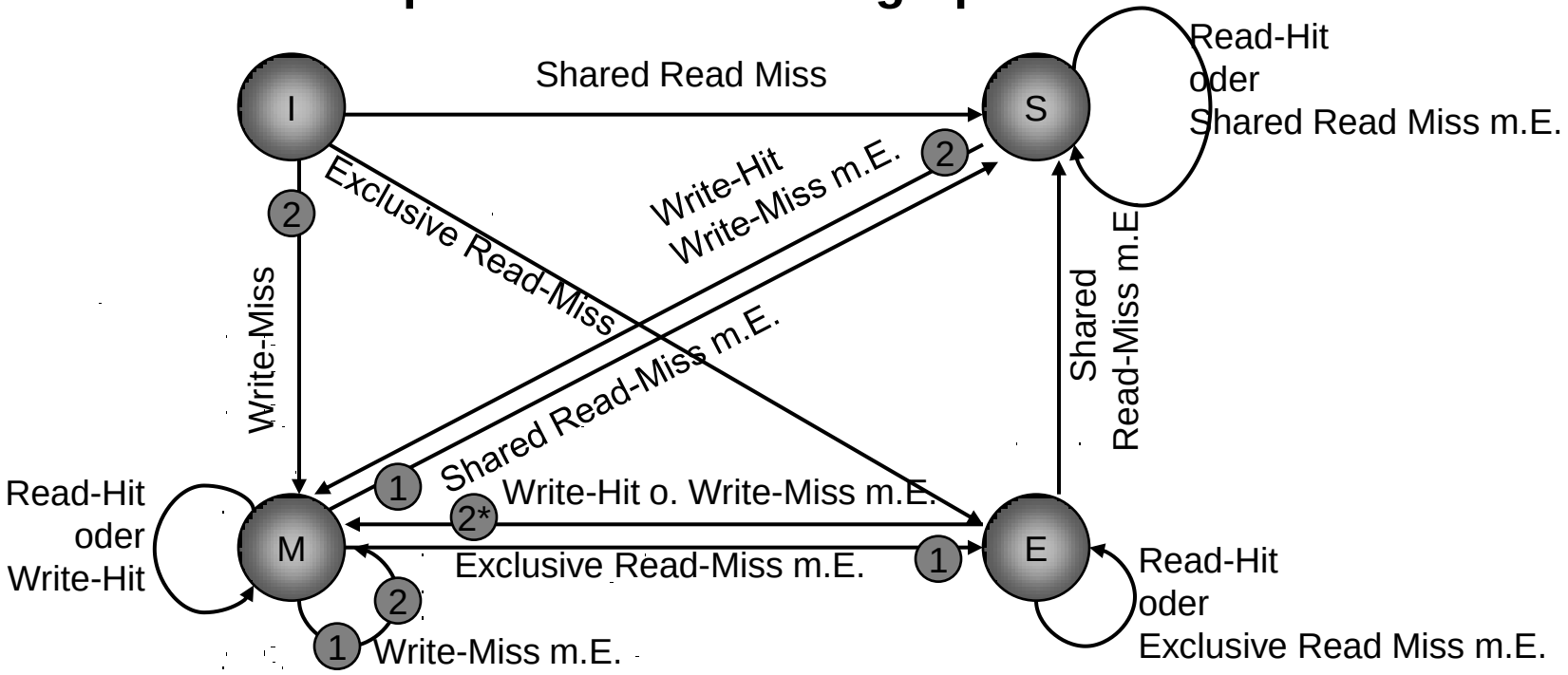
Kohärenz-Protokolle

- **MESI-Kohärenzprotokoll**
  - Darstellung durch **Zustandsgraph**
    - **Zustandsgraph 1**: Zustandsübergänge durch lokale Lese- und Schreibzugriffe, d. h. Zugriffe des Prozessors
      - Unterschiedliche Situationen
        - Der adressierte Speicherblock ist im Cache vorhanden
        - Eine gültige Zeile im Cache wird überschrieben (mit Ersetzung, m.E.)
        - Ungültige Cache-Zeile wird mit Block belegt
    - **Zustandsgraph 2**: Zustandsübergänge , die sich durch Beeinflussung von außen, von Seiten des Busses ergeben. Steuerung durch Snoop-Logik
      - Read-with intend to modify (Read w.i.t.m.) tritt bei Semaphorbefehlen auf

Multiprozessor mit gemeinsamem Speicher

Kohärenz-Protokolle

- **MESI-Kohärenzprotokoll: Zustandsgraph 1**



m.E.: mit Ersetzung

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

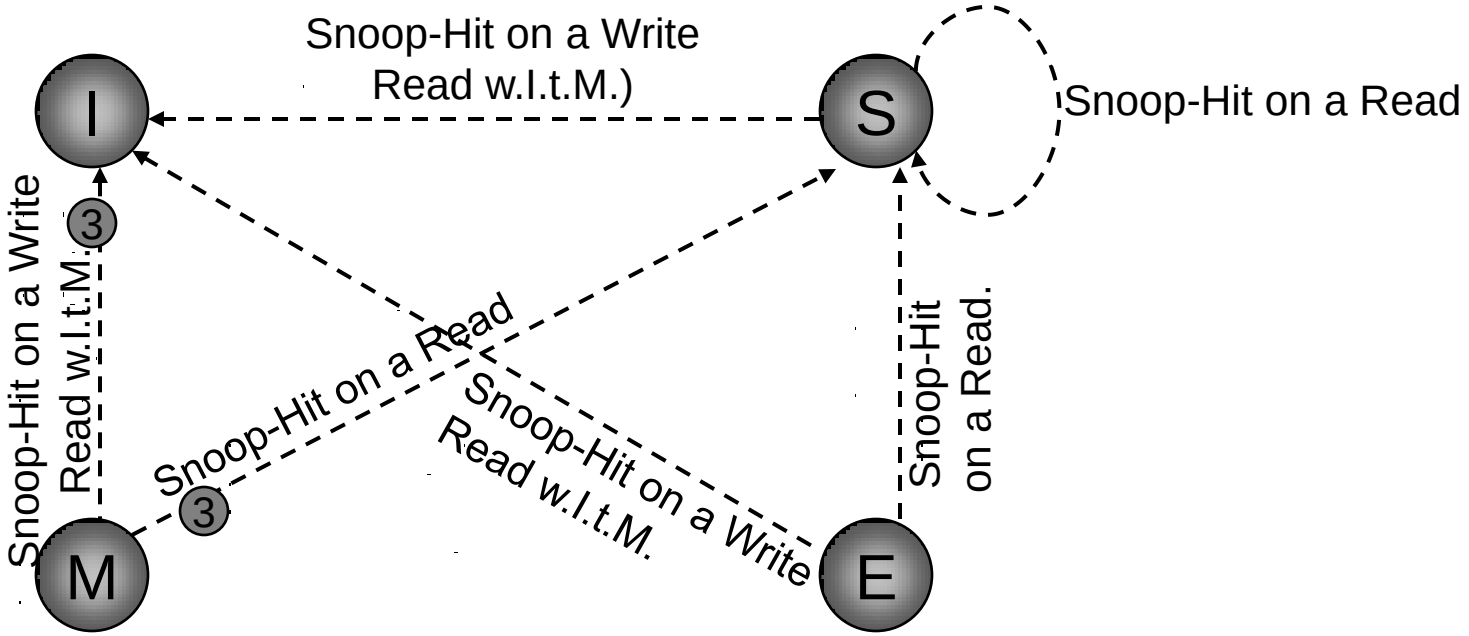
2 Cache-Zeilen in den anderen Caches mit gleicher Blockadresse werden invalidiert. (Line Clear)

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

Multiprozessor mit gemeinsamem Speicher

Kohärenz-Protokolle

- **MESI-Kohärenzprotokoll: Zustandsgraph 2**
  - Zustandsübergänge, die durch Aktionen, die auf dem Bus zu beobachten sind, ausgelöst werden.

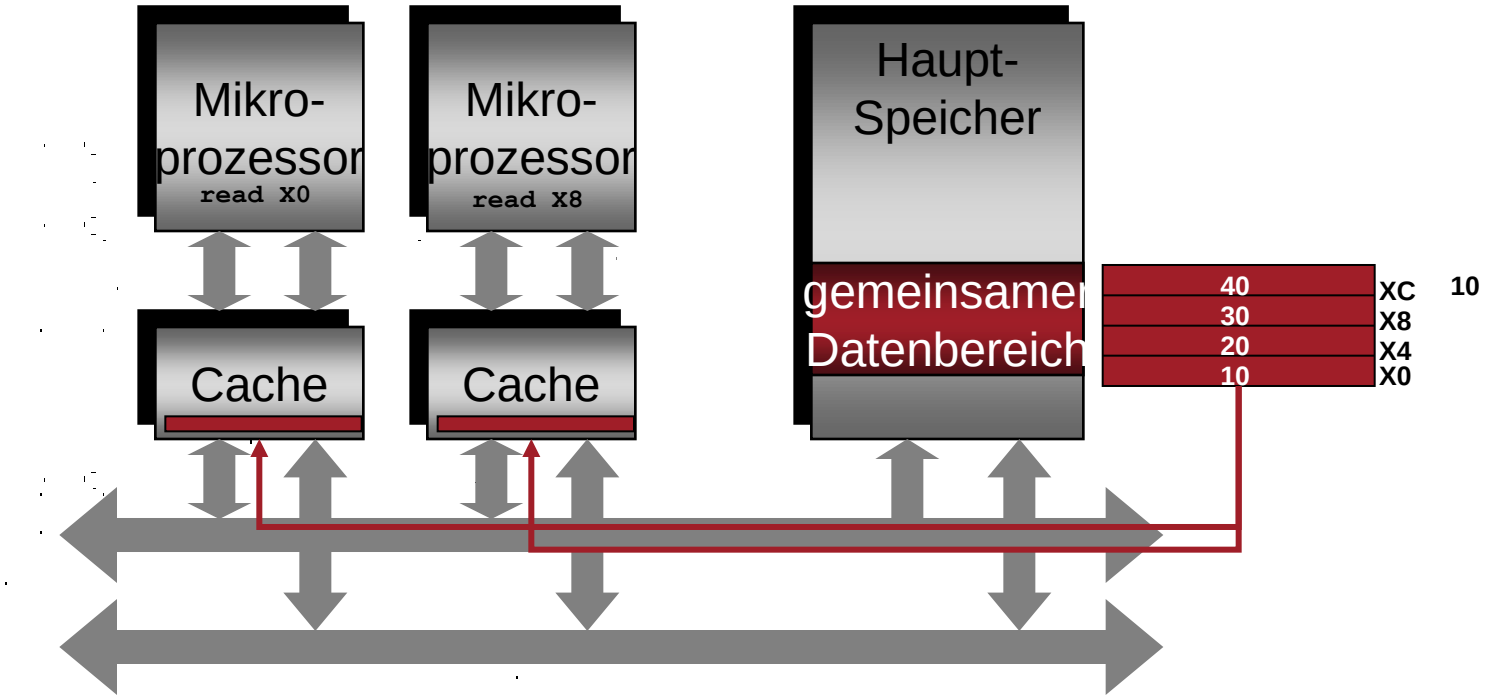


3 Retry-Signal wird aktiviert und danach wird die Cache-Zeile in den Hauptspeicher kopiert.

Multiprozessor mit gemeinsamem Speicher

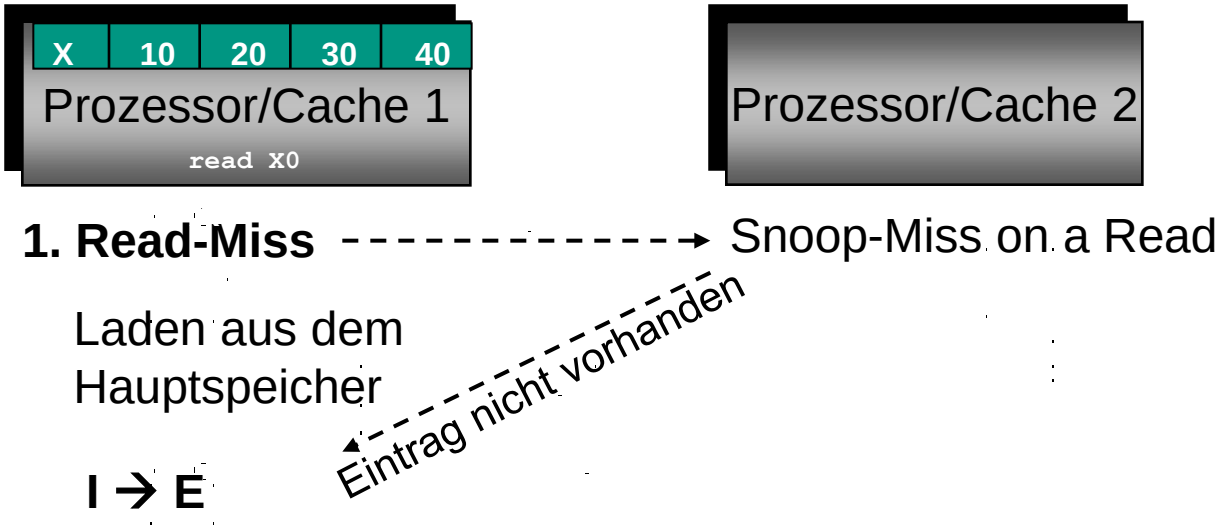
Kohärenz-Protokolle

- **MESI-Kohärenzprotokoll**:
  - Wirkungsweise am Beispiel eines Mikroprozessorsystems mit 2 Prozessoren



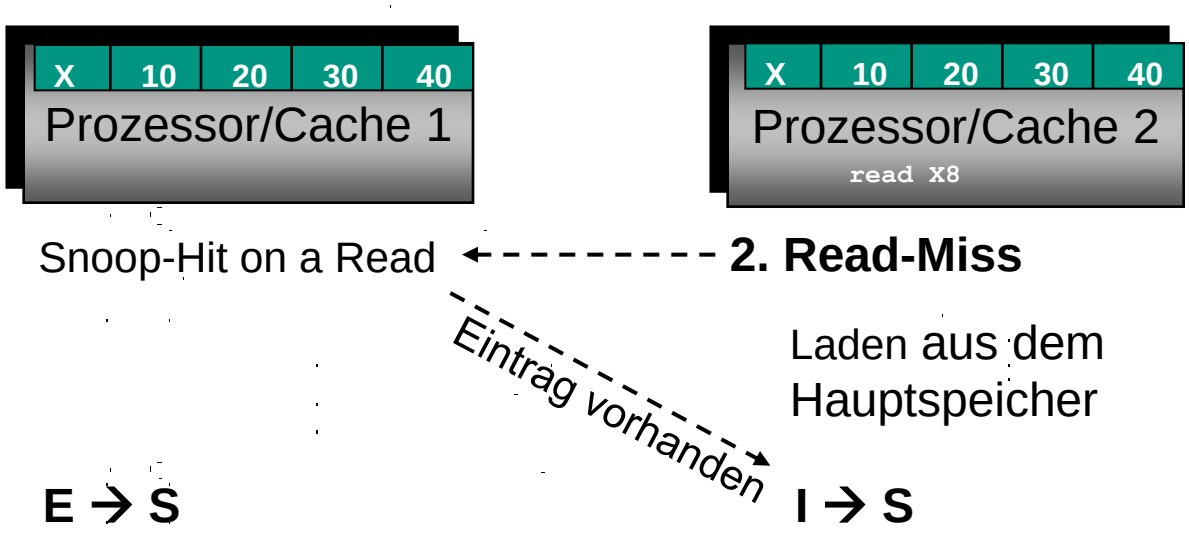
Multiprozessor mit gemeinsamem Speicher

- **Kohärenz-Protokolle**
- MESI-Kohärenzprotokoll: Wirkungsweise
  - Vier aufeinander folgende Zugriffe auf ein und denselben Speicherblock: False Sharing



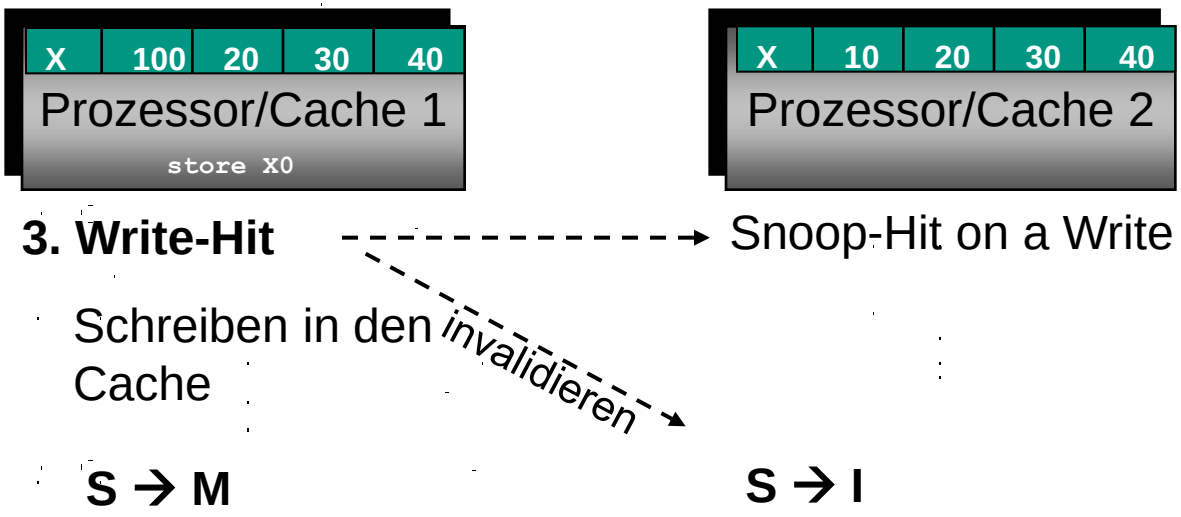
Multiprozessor mit gemeinsamem Speicher

- **Kohärenz-Protokolle**
- MESI-Kohärenzprotokoll: Wirkungsweise
  - Vier aufeinander folgende Zugriffe auf ein und denselben Speicherblock
  - False Sharing



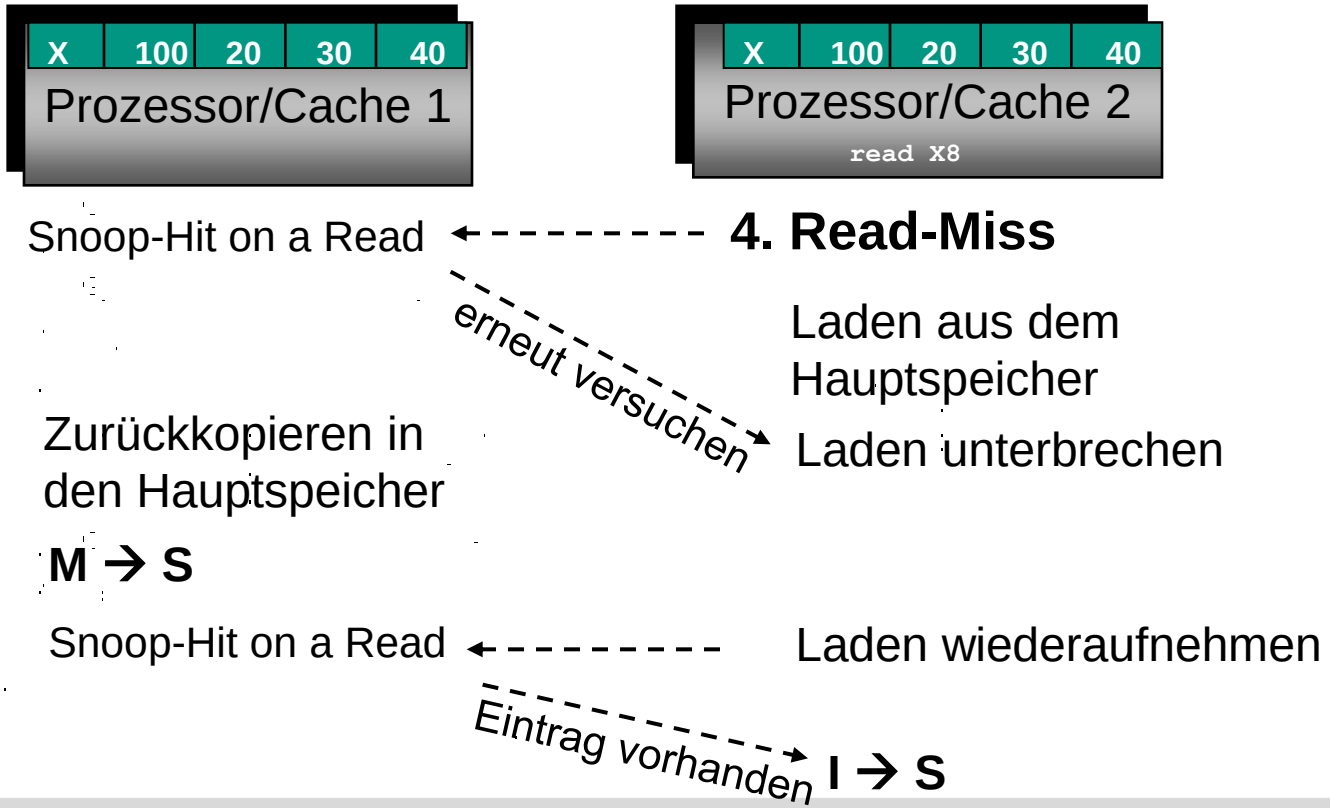
Multiprozessor mit gemeinsamem Speicher

- **Kohärenz-Protokolle**
- MESI-Kohärenzprotokoll: Wirkungsweise
  - Vier aufeinander folgende Zugriffe auf ein und denselben Speicherblock: False Sharing



Multiprozessor mit gemeinsamem Speicher

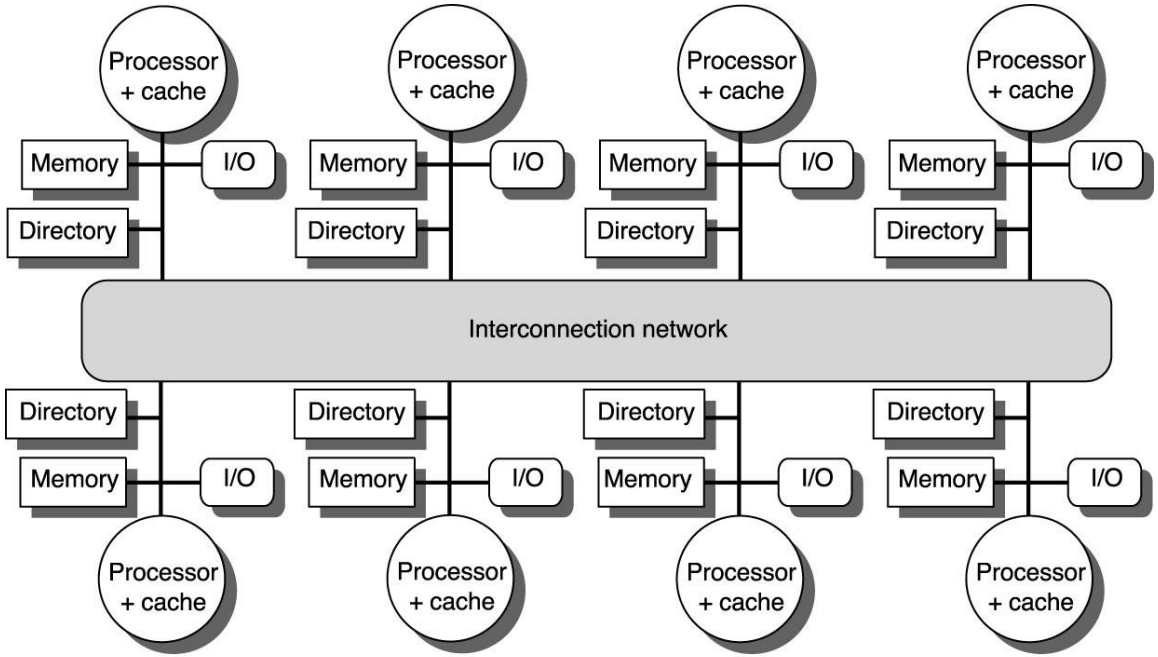
- **Kohärenz-Protokolle**
- MESI-Kohärenzprotokoll: Wirkungsweise
  - Vier aufeinander folgende Zugriffe auf ein und denselben Speicherblock: False Sharing



Multiprozessor mit gemeinsamem Speicher

Kohärenz-Protokolle

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



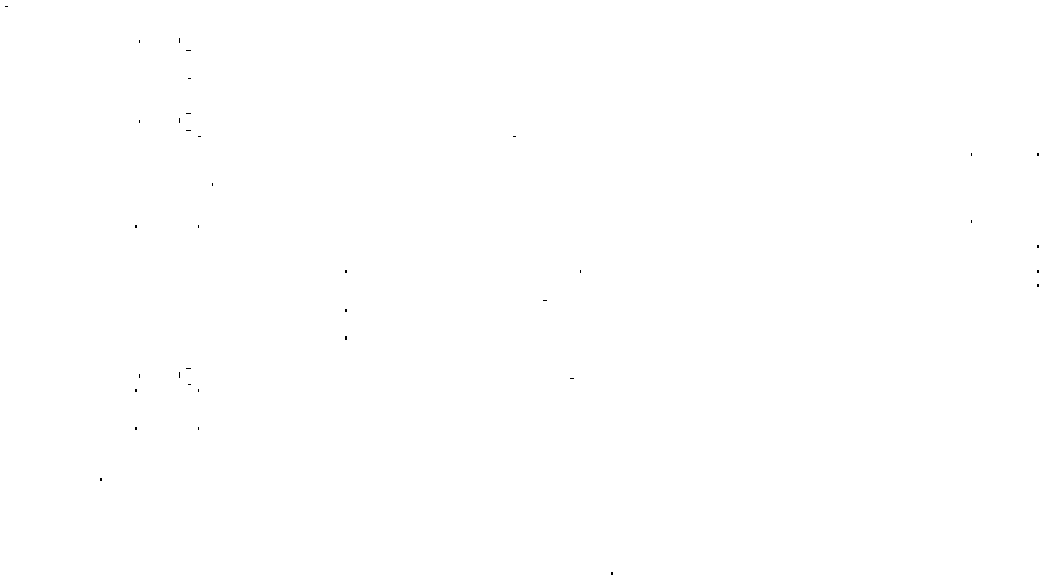
Multiprozessor mit gemeinsamem Speicher

Kohärenz-Protokolle

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

Speicherkonsistenz

- **Cache-Kohärenz**
  - sichert, dass mehrere Prozessoren eine kohärente Sicht auf den Speicher haben

- **Wichtige Fragen**:
  - Wann muss ein Prozessor den Wert sehen, den ein anderer Prozessor aktualisiert hat?
  - In welcher Reihenfolge muss ein Prozessor die Schreiboperationen eines anderen Prozessors beobachten?
  - Welche Bedingungen zwischen Lese- und Schreiboperationen auf verschiedene Speicherstellen durch verschiedene Prozessoren müssen gelten?

Multiprozessor mit gemeinsamem Speicher

Speicherkonsistenz

- Beispiel:
  - Die Prozesse P1 und P2 laufen auf verschiedenen Prozessoren
  - A=0 und B=0 sind ursprünglich in den Caches der beiden Prozessoren

```
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
```

Multiprozessor mit gemeinsamem Speicher

Speicherkonsistenz

- Beispiel: Was kann passieren?
  - a1 wird ausgeführt und a2 nicht
  - a2 wird ausgeführt und a1 nicht
  - a1 und a2 werden beide nicht ausgeführt

  - Voraussetzung: die Schreiboperationen werden unmittelbar wirksam und werden jeweils von dem anderen Prozessor gesehen

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

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

Multiprozessor mit gemeinsamem Speicher

Speicherkonsistenz

- Beispiel: Was kann passieren?
  - Write invalidate wird verzögert und die Prozessoren dürfen mit der Ausführung fortfahren
  - Dann kann es möglich sein, dass P1 und P2 die jeweiligen Invalidierung für B und A noch nicht gesehen haben bevor sie versuchen, die Werte für A und B zu lesen
    - Verzögerungen, Spekulation
  - Also: a1 und a2 werden beide ausgeführt!

```
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
```

Multiprozessor mit gemeinsamem Speicher

Speicherkonsistenzmodelle

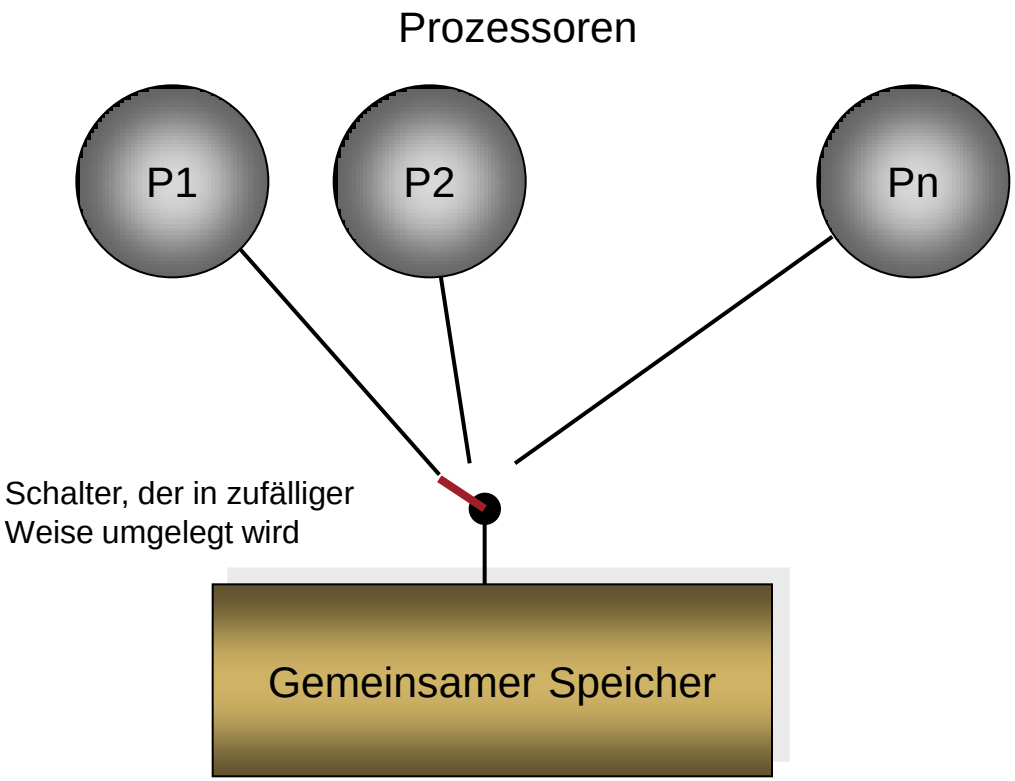
- Spezifizieren die Reihenfolge, in der Speicherzugriffe eines Prozesses von anderen Prozessen gesehen werden

- **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
  - Schreibzugriffe müssen atomisch sein, d. h. der jeweilige Wert muss überall gleichzeitig wirksam sein

Multiprozessor mit gemeinsamem Speicher

Sequentielle Konsistenz

- Veranschaulichung der sequentiellen Konsistenz



Multiprozessor mit gemeinsamem Speicher

Sequentielle Konsistenz

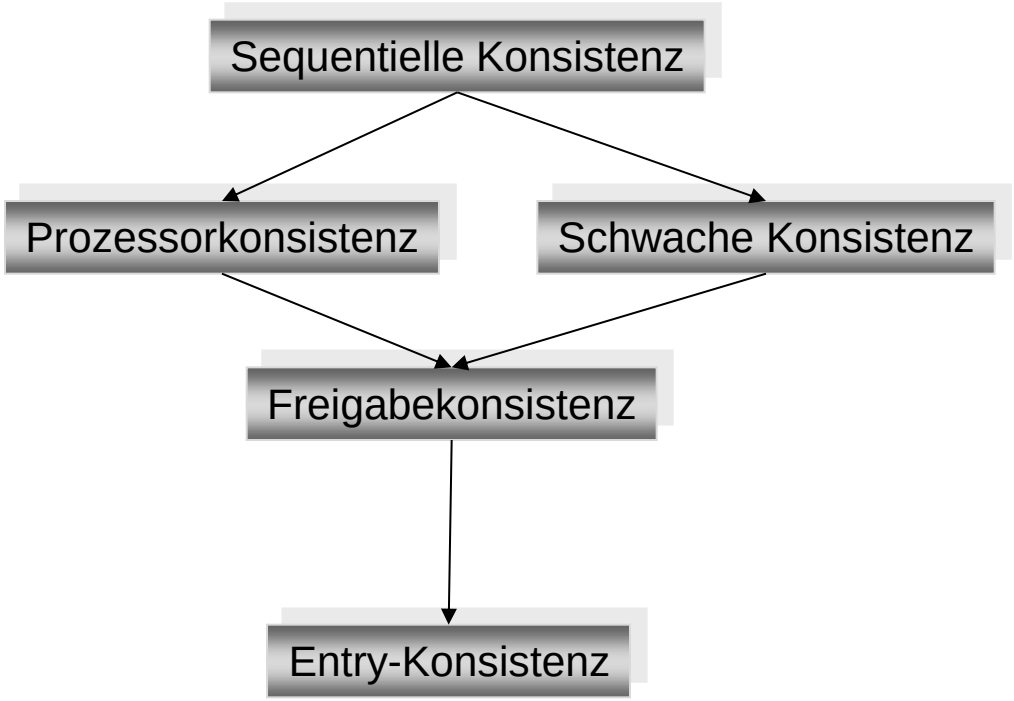
- Programmierer geht von sequentieller Konsistenz aus
- Führt aber zu sehr starken Einbußen bzgl. Implementierung und damit der Leistung
- Verbietet vorgezogene Ladeoperationen, nichtblockierende Caches

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



Multiprozessor mit gemeinsamem Speicher

Speicherkonsistenzmodelle

- **Schwache Konsistenz**
  - 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

Schwache Konsistenz

- 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 Bereiche wird die Inkonsistenz der gemeinsamen Daten zugelassen
    - Voraussetzung: konkurrierende Lese-/Schreibzugriffe sind durch den kritischen Bereiche unterbunden
    - Synchronisationspunkte sind dabei die Ein-/ und Austrittpunkte der kritischen Bereiche

Multiprozessor mit gemeinsamem Speicher

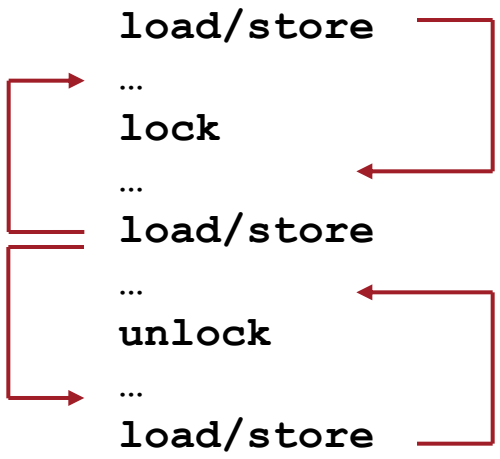
Schwache Konsistenz

- 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

Multiprozessor mit gemeinsamem Speicher

Schwache Konsistenz

- Auswirkung
  - Synchronisationsbefehle stellen Hürden dar, die von keinem Lese- oder Schreibzugriff übersprungen werden



Rote Pfeile: verboten

Multiprozessor mit gemeinsamem Speicher

Schwache Konsistenz

- 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

Synchronisation

- Wann können verschiedene Prozesse sicher auf gemeinsame Daten zugreifen?
  - Nichtunterbrechbare Operation für Lese- und Schreiboperation auf den Speicher (atomare Operation)
  - Synchronisationsoperation auf Benutzerebene
  - Frage der Implementierung wegen Leistungseinbußen

Multiprozessor mit gemeinsamem Speicher

Synchronisation

- Synchronisationsmechanismen
  - Bereitstellung über Software-Routinen (Benutzerebene)
  - Abbildung auf HW-Synchronisationsbefehle

  - Geringe Blockierung / kleine Systeme
    - Nicht unterbrechbare Instruktion(sfolge): test-and-set
    - Atomare Abarbeitung (read/modify/write)

  - Häufige Blockierung / große Systeme
    - Komplexere Synchronisationsmechanismen
    - Spin Locks, Barriers

Multiprozessor mit gemeinsamem Speicher

Synchronisation

- Hardware-Primitive
  - Unteilbarer Austausch (atomic exchange/swap)
  - Austausch von Register- mit Speicherwert
  - Anwendungsbeispiel: Aufbau einer Synchronisationsoperation

- Test and Set
  - Überprüfung eines Speicherwertes auf Bedingung
  - Setzen eines Wertes, falls Bedingung erfüllt

- Problem: Kohärenz
  - Unteilbare Operation verkompliziert Kohärenz
  - Keine andere Operation während unteilbarem Befehl erlaubt
  - Deadlock-Freiheit zwingend

Multiprozessor mit gemeinsamem Speicher

Synchronisation

- Barrier Synchronization
  - Synchronisation mehrerer Prozesse
    - Vorgabe einer Prozessgruppe/-anzahl
    - Prozesse warten an Barriere
    - Freigabe, wenn alle Prozesse angekommen
  - Abbildung auf Spin Locks
    - Schutz des Zählers für ankommender Prozesse
    - Implementierung der eigentlichen Barriere
  - Elementare Synchronisationsoperation in der Parallelen Programmierung
    - OpenMP
    - MPI