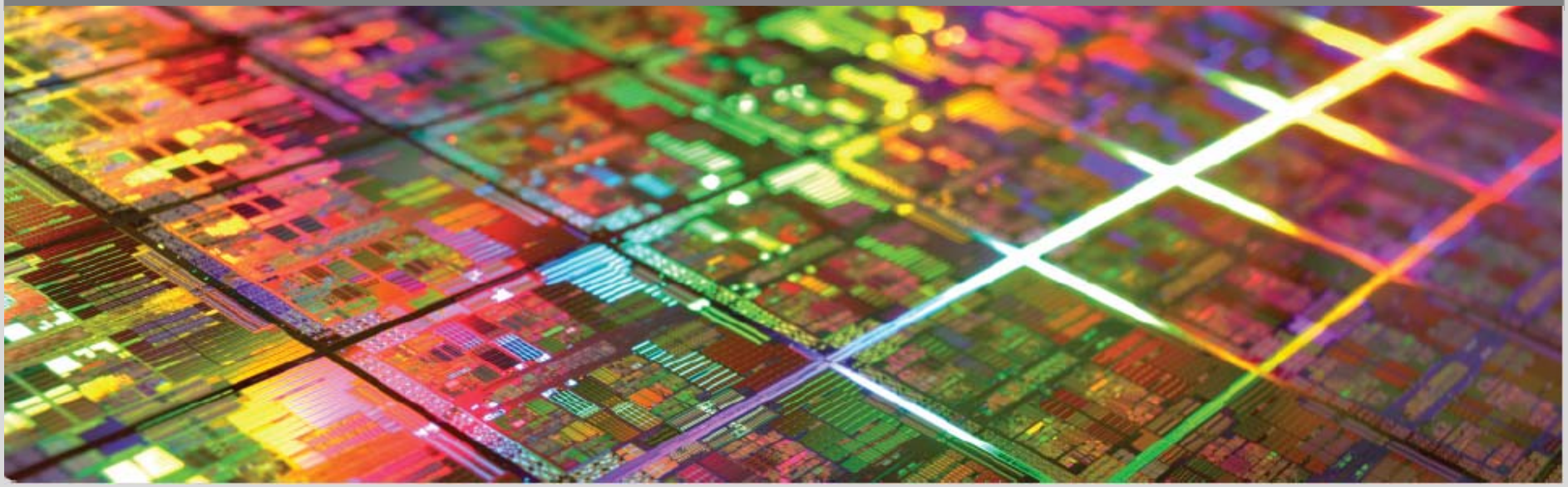
Karlsruher Institut für Technologie

Rechnerstrukturen

Vorlesung im Sommersemester 2010

Prof. Dr. Wolfgang Karl

Fakultät für Informatik – Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

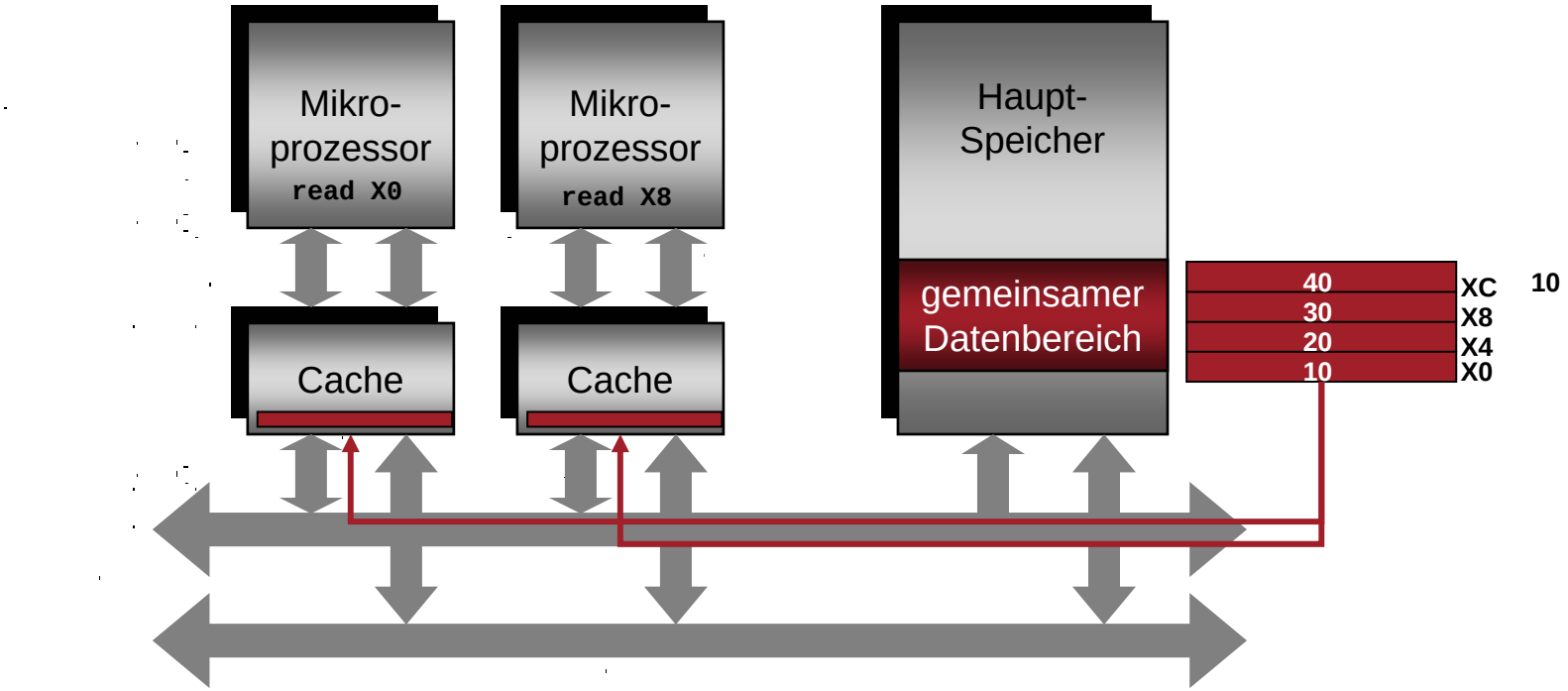


Vorlesung Rechnerstrukturen

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene**
- 3.6 Multiprozessoren mit gemeinsamem Speicher

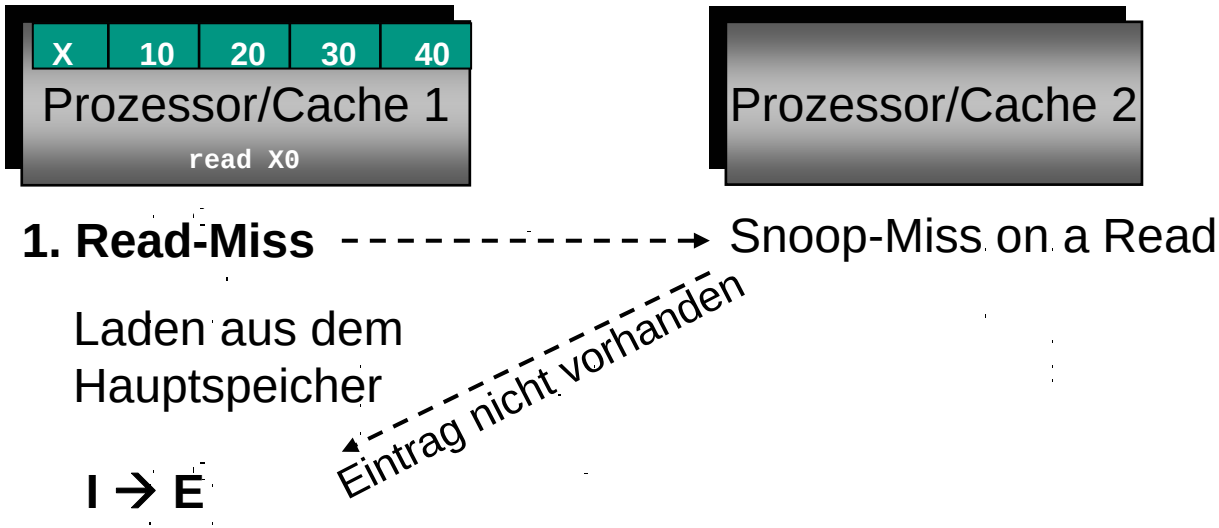
Multiprozessor mit gemeinsamem Speicher

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



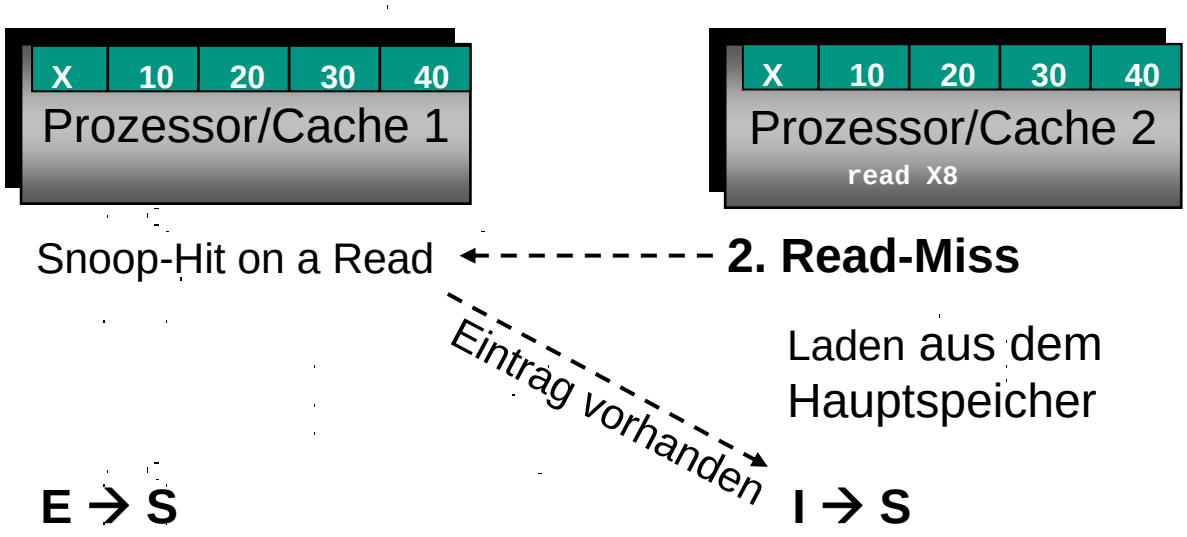
Multiprozessor mit gemeinsamem Speicher

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



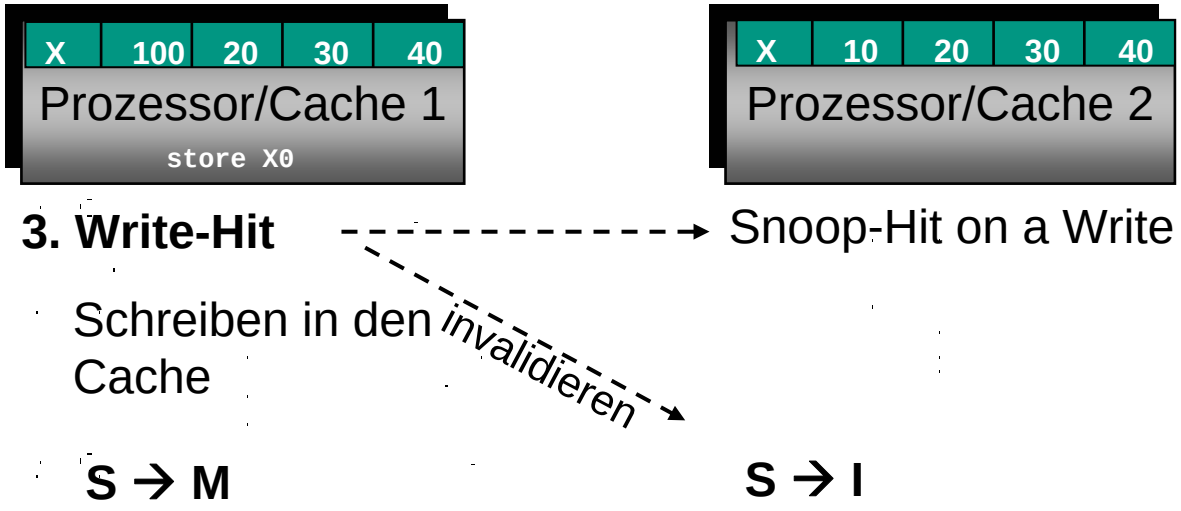
Multiprozessor mit gemeinsamem Speicher

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



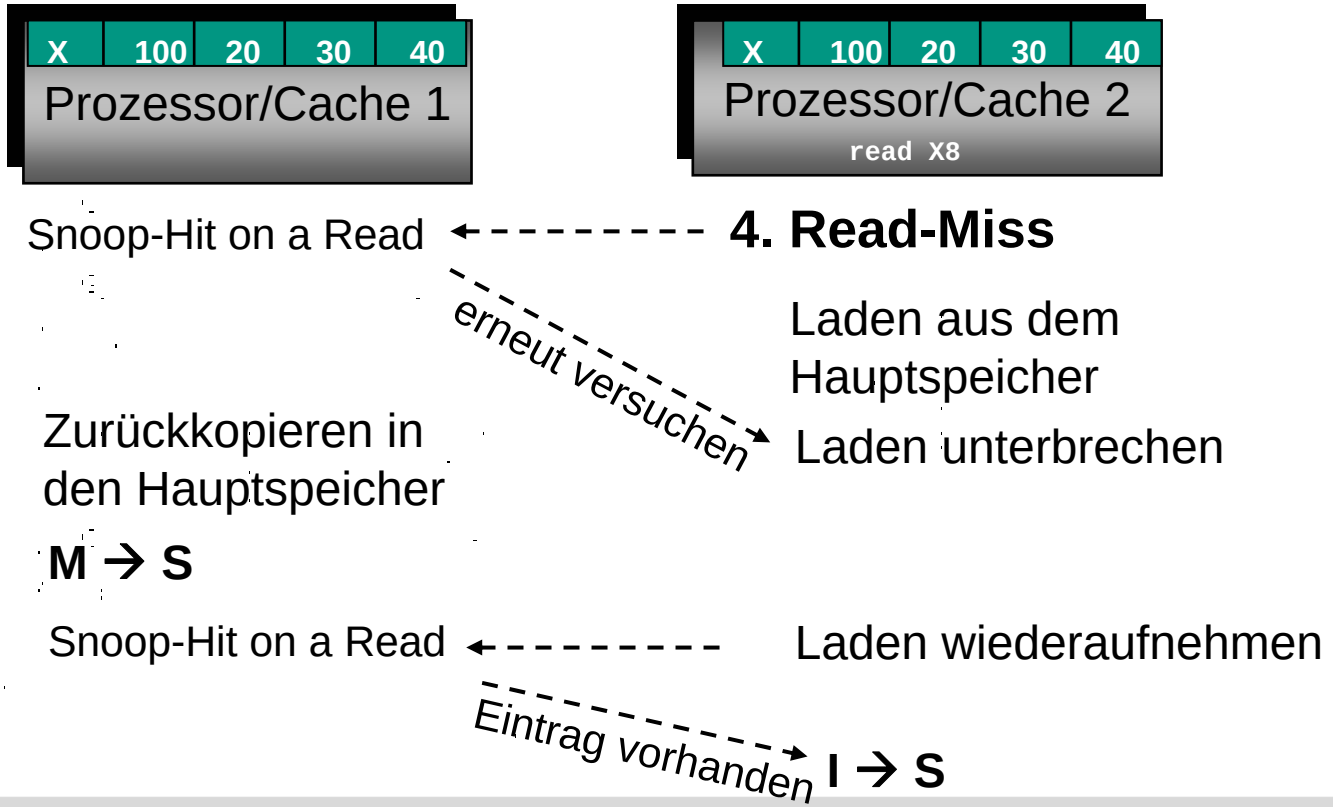
Multiprozessor mit gemeinsamem Speicher

- **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

- **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;
        …
        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?
  - 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!

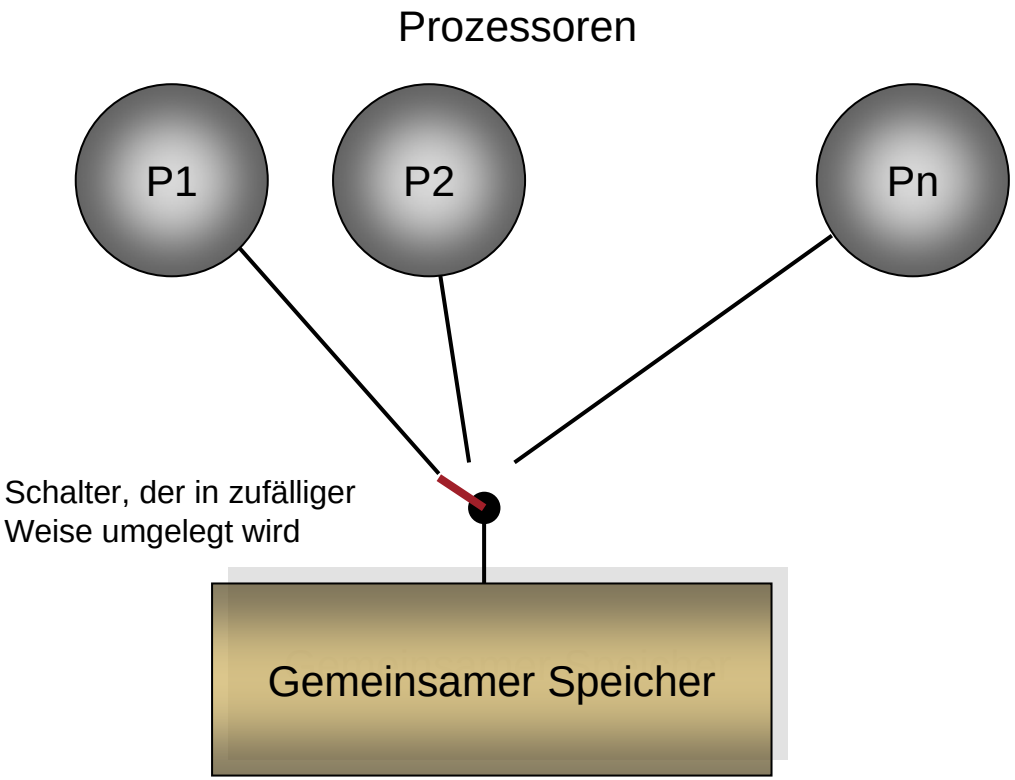
```
P1:     A=0;                    P2:     B=0;
        …                               …
        A=1;                            B=1;
L1:     if (B==0)               L2:     if (A==0)
        …führe Aktion a2 aus            …führe Aktion a1 aus
```

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**
- 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
  - 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

Multiprozessor mit gemeinsamem Speicher

- **Speicherkonsistenzmodelle**
- **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
  - Stellt die korrekte Ordnung der Schreibzugriffe sicher, nicht jedoch die Korrektheit der Zugriffe auf gemeinsam benutzte datenobjekte durch parallele Kontrollfäden
    - Synchronisation des Datenzugriffs oder Prozessablaufs

Multiprozessor mit gemeinsamem Speicher

- **Speicherkonsistenzmodelle**
- **Prozessorkonsistenz**
  - 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 Zugriffe ausgeführt worden sein

Multiprozessor mit gemeinsamem Speicher

- **Speicherkonsistenzmodelle**
- **Prozessorkonsistenz**
- 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

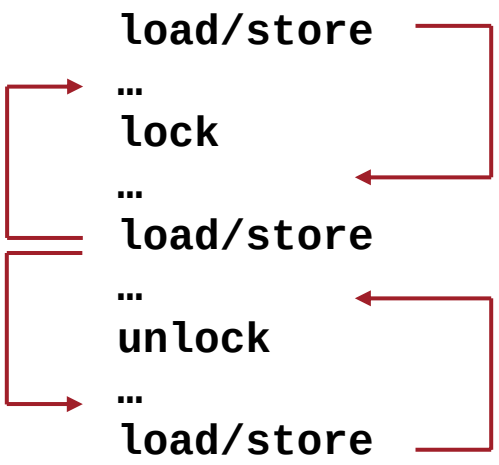
- **Schwache Konsistenz**
- Idee
  - Die Konsistenz des Speicherzugriffs wird nicht mehr zu allen Zeiten gewährleistet, sondern zu bestimmten, vom Programmierer in das Programm eingesetzten Synchronisationspunkten
  - Einführung von kritischen Bereichen
    - Innerhalb dieser Bereiche wird die Inkonsistenz 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

- **Speicherkonsistenzmodelle**