Grundlagen der Rechnerarchitektur und Betriebssysteme

Multicore-Rechner

Florian Knoch · Lehrstuhl für Praktische Informatik,
insbes. Systemnahe Programmierung, Universität Bamberg

Evaluation

Gesellschaft für Informatik

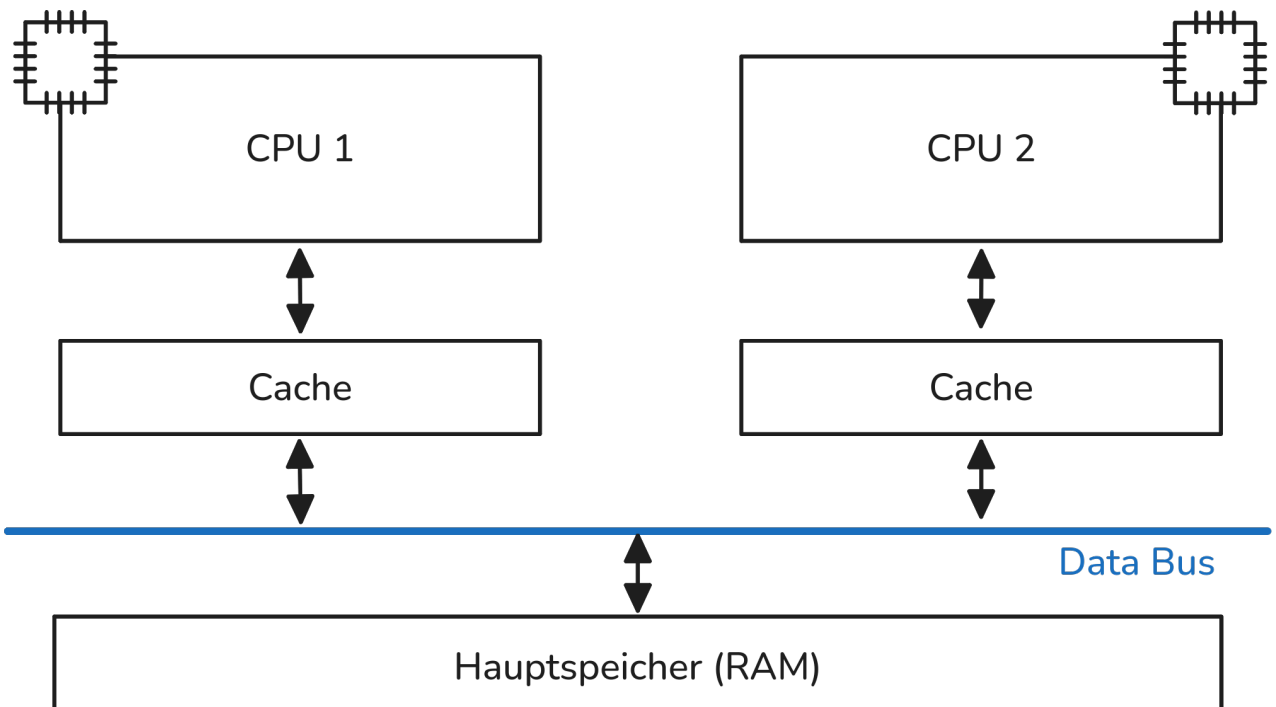
Interessensvertretung der deutschsprachigen Informatiker:innen

KORREKTUR

`satp` = Supervisor Address Translation and Protection

Multicore-Rechner

Ausgangslage



Zwei **Prozessorkerne**, die über einen geteilten Bus mit dem RAM verbunden sind

Einfacher ***Write-Back-Cache*** für jeden Kern

Anforderungen an Caches

nach D. A. Patterson und J. L. Hennessy, [1]

Migration

Daten können zu lokalen Caches verlegt und dort transparent genutzt werden.

Ziele: niedrigere Latenz, geringerer Brandbreitenbedarf

Anforderungen an Caches

nach D. A. Patterson und J. L. Hennessy, [1]

Migration

Daten können zu lokalen Caches verlegt und dort transparent genutzt werden.

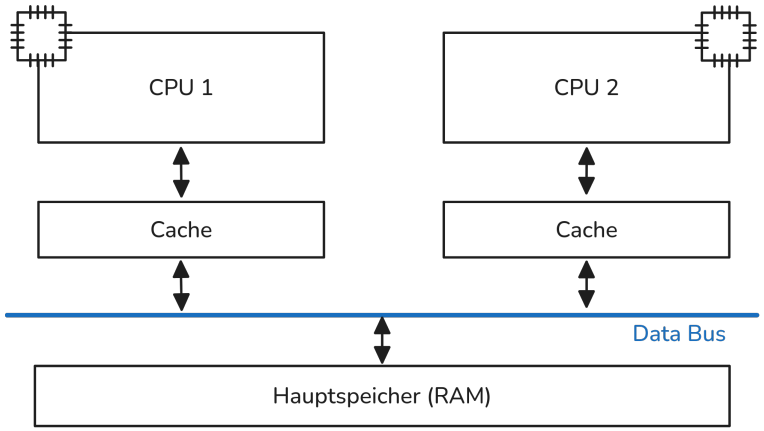
Ziele: niedrigere Latenz, geringerer Brandbreitenbedarf

Replikation

Gleichzeitig gelesene, geteilte Daten werden in den lokalen Cache kopiert.

Ziele: niedrigere Latenz, weniger Berechtigungskonflikte

Cache-Kohärenz



Wie stellen wir sicher, dass …

… höchstens ein Prozessorkern gleichzeitig in eine Speicherzelle schreibt?
(**„*Single-Writer, Multiple-Reader*“-Invariante**)

… jede Operation den Wert verwendet, der nach der letzten Speicheroperation in der Speicherzelle stand?
(**„*Data Value*“-Invariante**)

Kohärenzprotokolle

Woher weiß Kern A, was Kern B getan hat?

Snooping-basiert

Alle Prozessoren werden über alle relevanten Operationen in Kenntnis gesetzt und antworten entsprechend.

Kohärenzprotokolle

Woher weiß Kern A, was Kern B getan hat?

Snooping-basiert

Alle Prozessoren werden über alle relevanten Operationen in Kenntnis gesetzt und antworten entsprechend.

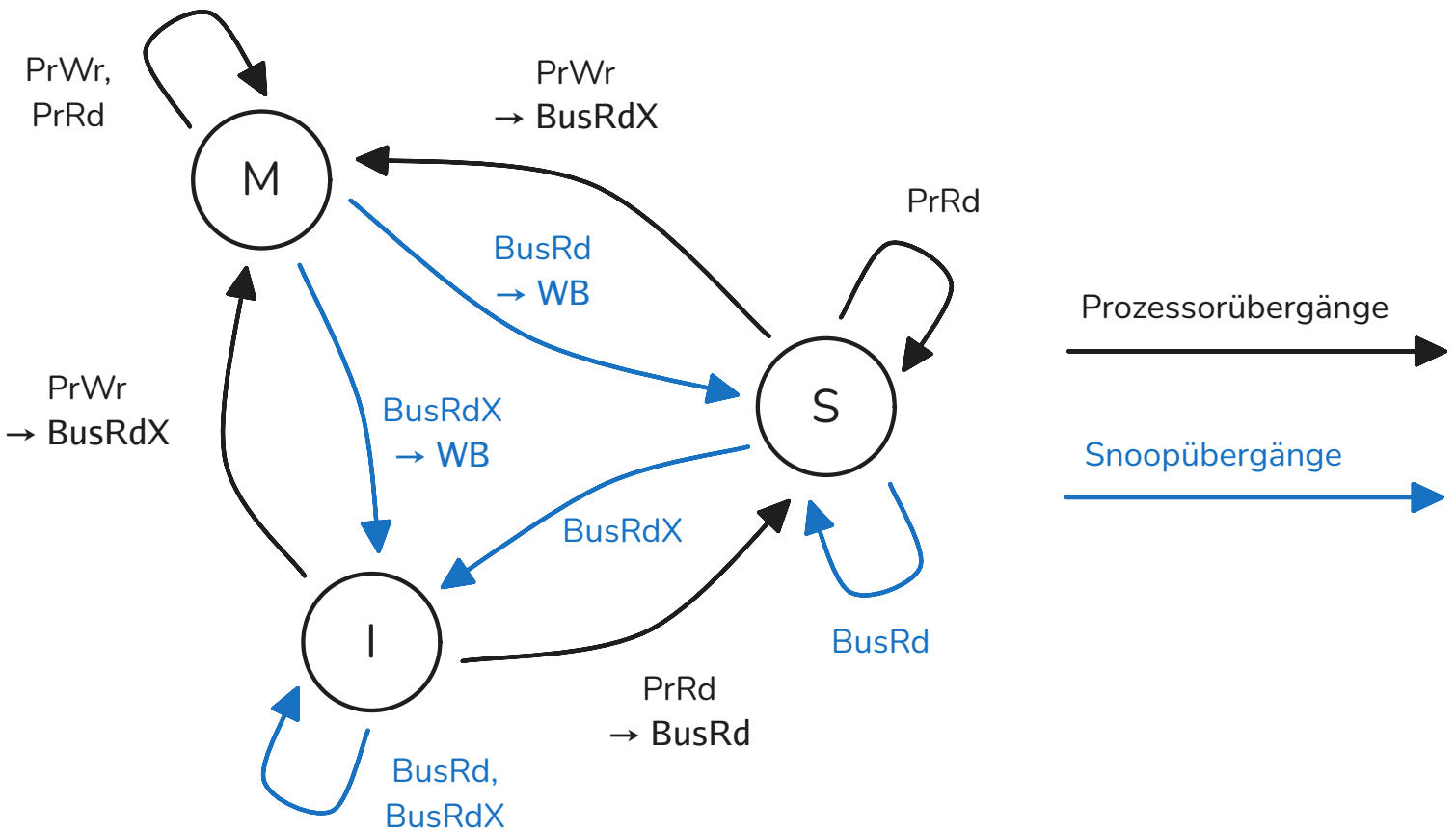
Directory-basiert

Alle Operationen werden über ein zentrales Verzeichnis koordiniert, das die „Eigentümer“ einer Cache-Zeile vorhält.

Kohärenzprotokoll *Modified-Shared-Invalid* (MSI)

Zustände
einer *Cache Line*

- **M** odified
- **S** hared
- **I** nvalid



Kohärenzprotokoll *Modified-Shared-Invalid* (MSI)

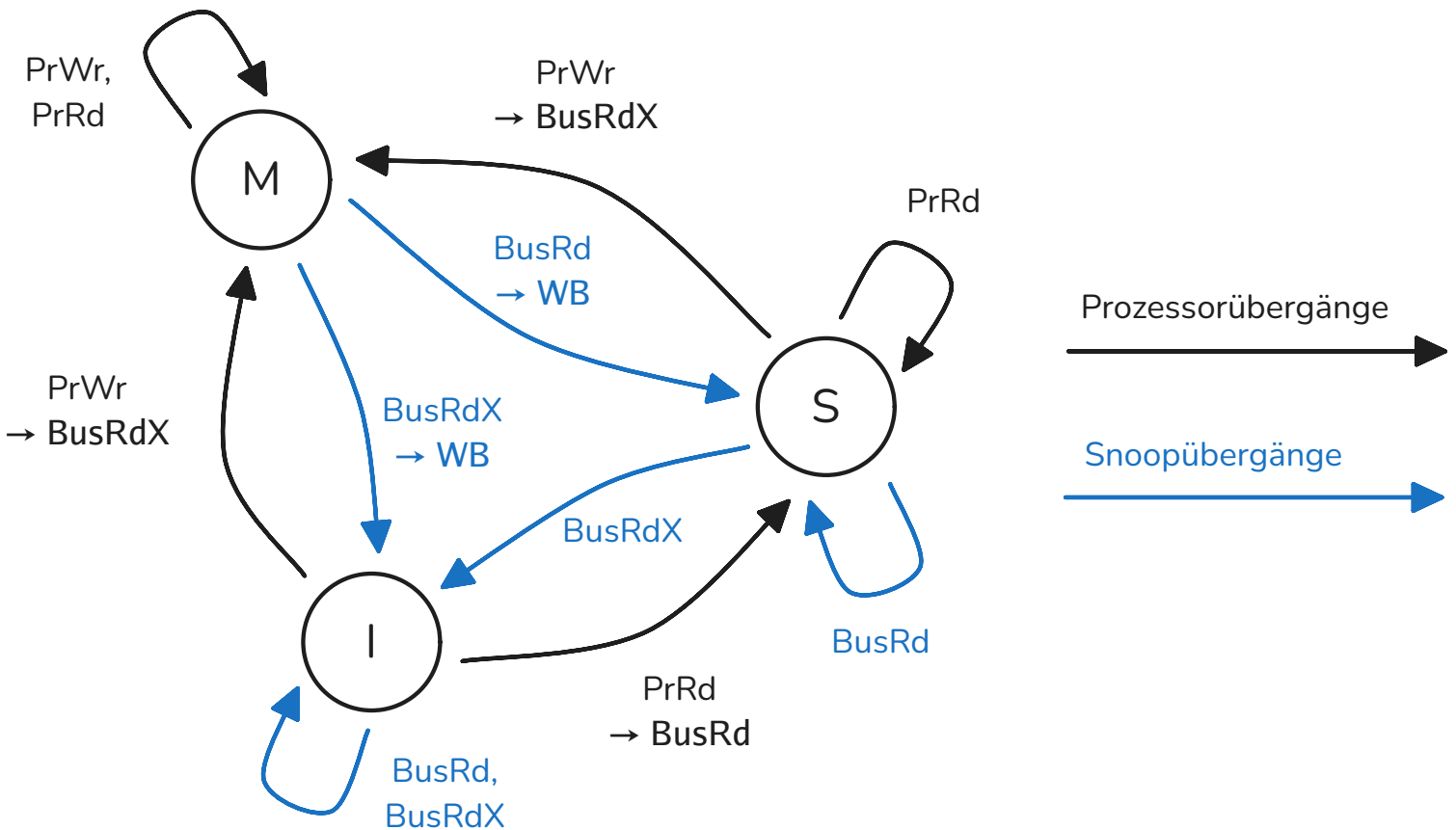
Zustände
einer *Cache Line*

- **M** odified
- **S** hared
- **I** nvalid

Akteure

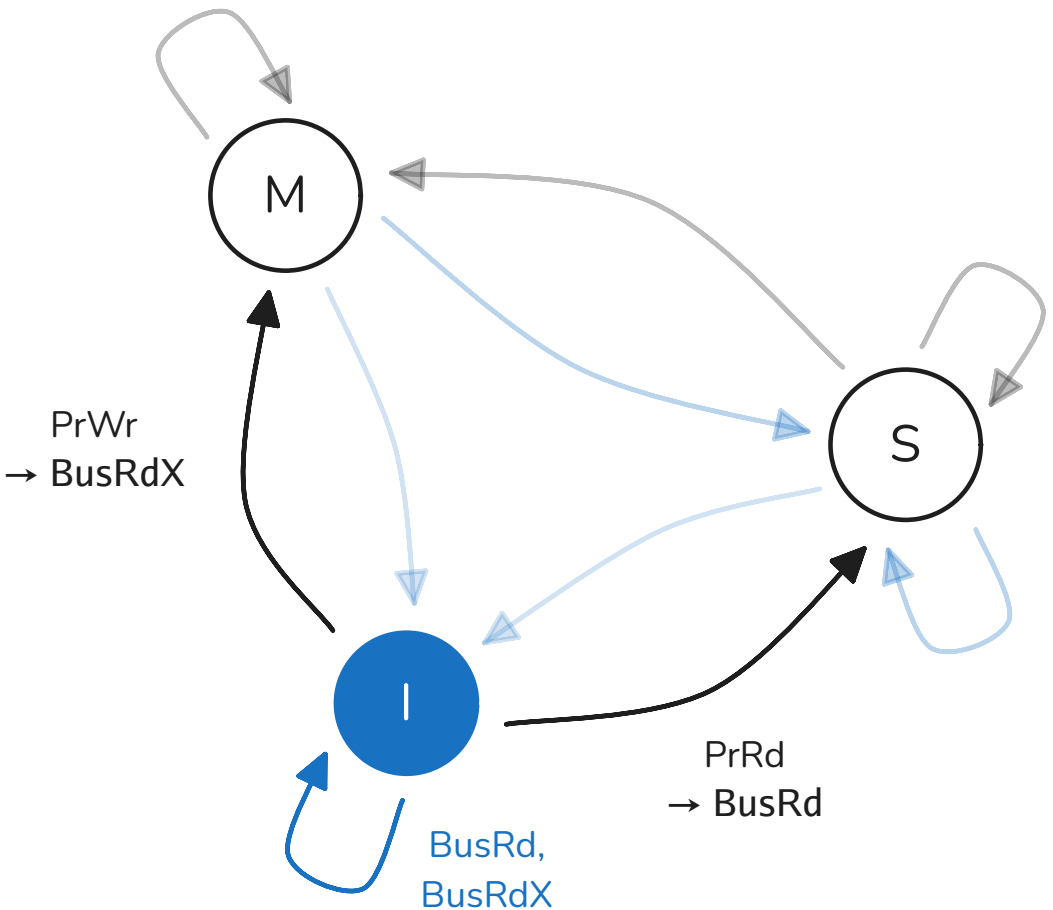
🖳 lokaler Kern („wir“)

🖳🖳🖳 die anderen Kerne, vermittelt über einen Bus



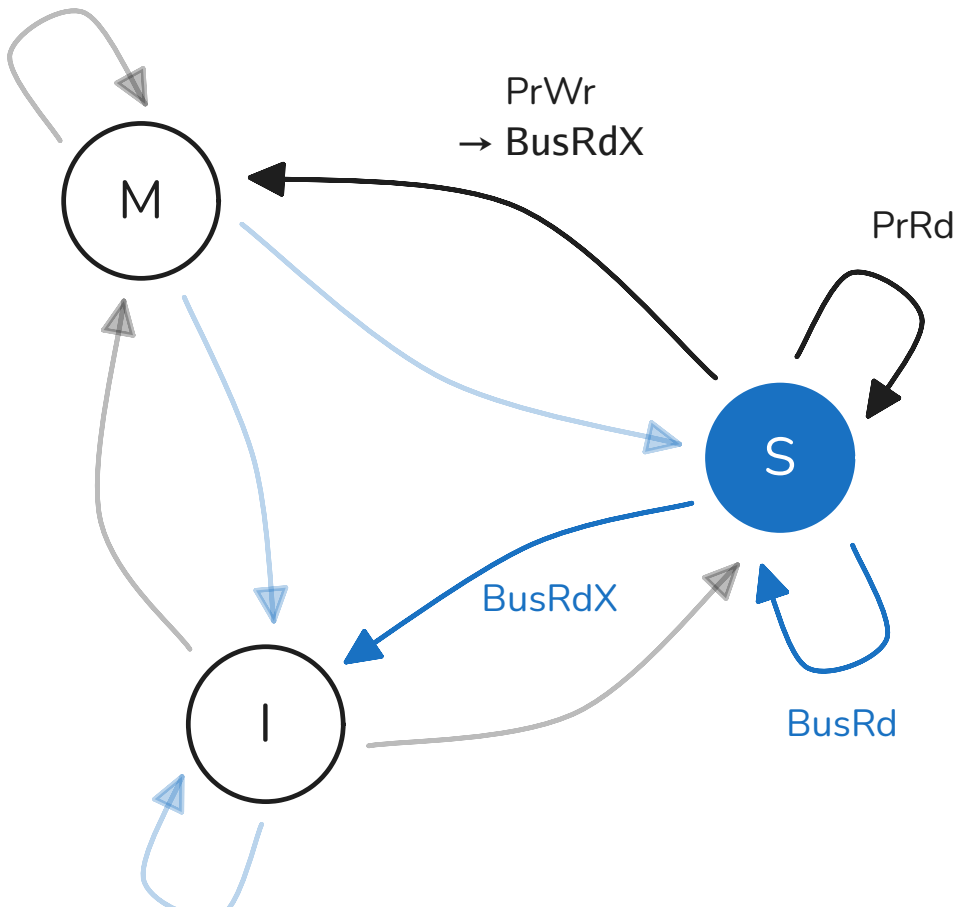
MSI-Zustand *Invalid*

- Ausgangspunkt für jeden Cache
- **andere Kerne wollen lesen** (`BusRd`): keine Änderung
- **wir wollen lesen** (`PrRd`): geteilten Besitz anfordern (`BusRd`)
- **andere Kerne wollen schreiben** (`BusRdX`): keine Änderung
- **wir wollen schreiben** (`PrWr`): exklusiven Besitz anfordern (`BusRdX`)



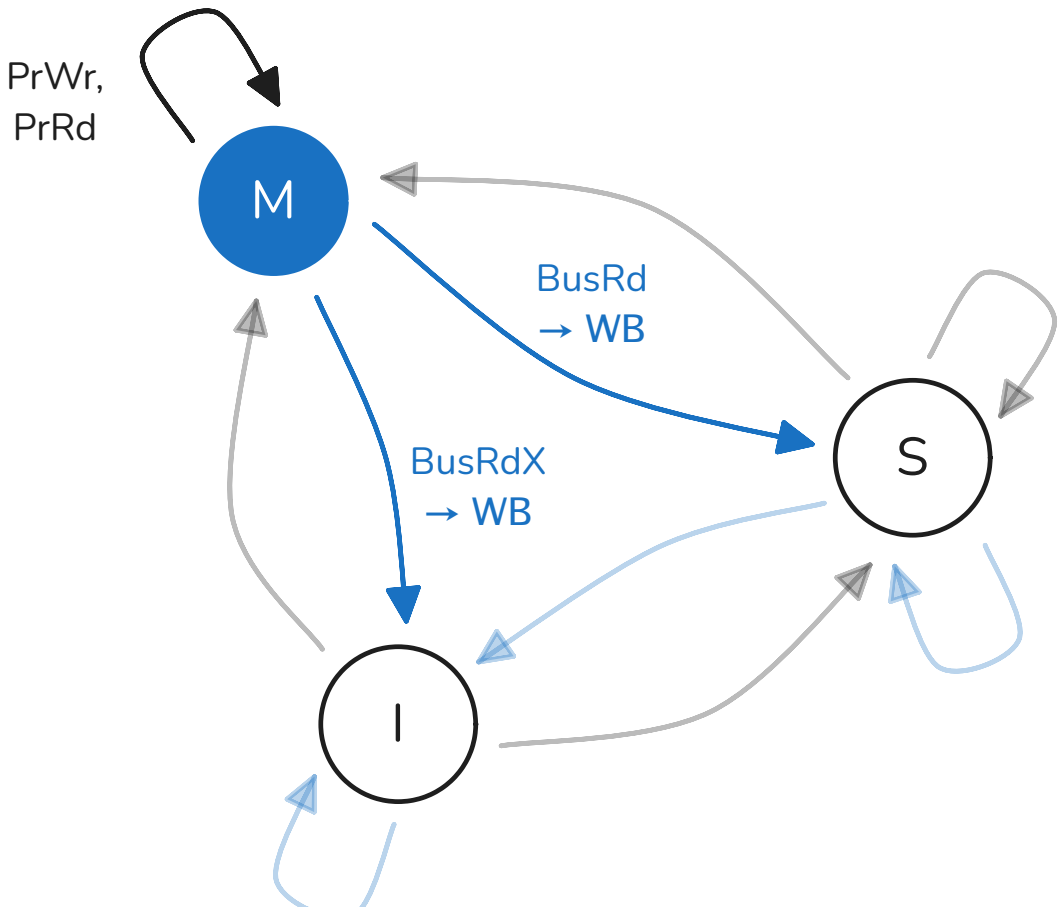
MSI-Zustand *Shared*

- mehrere Kerne haben geteilte Leseberechtigungen
- **andere Kerne wollen lesen** (BusRd): keine Änderung
- **wir wollen lesen** (PrRd): keine Änderung
- **andere Kerne wollen schreiben** (BusRdX): unsere Kopie wird invalidiert
- **wir wollen schreiben** (PrRdX): exklusiven Besitz anfordern (BusRdX)



MSI-Zustand *Modified*

- wir haben exklusive Schreibberechtigungen
- **andere Kerne wollen lesen** (`BusRd`):
  Änderungen zurückschreiben (`WB`)
- **wir wollen lesen** (`PrRd`):
  keine Änderung
- **andere Kerne wollen schreiben** (`BusRdX`):
  Änderungen zurückschreiben (`WB`)
- **wir wollen schreiben** (`PrWr`):
  keine Änderung



Herausforderung: *False Sharing*

When **two unrelated shared variables** are **located in the same cache block**, the whole block is exchanged between processors even though the processors are accessing different variables. Programmers and compilers **should lay out data carefully** to avoid false sharing.

— D. A. Patterson und J. L. Hennessy [1]

Speicherkonsistenz

Speicherkonsistenz

> Ein **Speicherkonsistenzmodell** ist eine Spezifikation des erlaubten Verhaltens eines nebenläufigen Programms, dessen Threads auf geteiltem Speicher ausgeführt werden.
>
> — übersetzt nach D. A. Patterson und J. L. Hennessy [1]

Speicherkonsistenz

> Ein **Speicherkonsistenzmodell** ist eine Spezifikation des erlaubten Verhaltens eines nebenläufigen Programms, dessen Threads auf geteiltem Speicher ausgeführt werden.
>
> — übersetzt nach D. A. Patterson und J. L. Hennessy [1]

Genügt da nicht das Cache-Kohärenzprotokoll?

Zur Erinnerung:

„*Single-Writer, Multiple-Reader*“-Invariante: höchstens ein Prozessorkern schreibt gleichzeitig in eine Speicherzelle

„*Data Value*“-Invariante: jede Operation verwendet den aktuellsten Wert in der Speicherzelle

Genügt da nicht das Cache-Kohärenzprotokoll?

Nein. Cache-Kohärenz ist nicht hinreichend für Speicherkonsistenz.

BEISPIEL

Wenn die *Pipeline* Instruktionen neu ordnet, kann das Ergebnis inkonsistent sein, auch wenn alle Anforderungen der Cache-Kohärenz eingehalten werden.

Speicherkonsistenzmodelle

Sequential Consistency

alle Instruktionen eines Kerns werden in der Programmreihenfolge ausgeführt

Speicherkonsistenzmodelle

Sequential Consistency

alle Instruktionen eines Kerns werden in der Programmreihenfolge ausgeführt

garantiert in Programmreihenfolge:

Lesen ⟶ Lesen
Lesen ⟶ Schreiben
Schreiben ⟶ Lesen
Schreiben ⟶ Schreiben

Speicherkonsistenzmodelle

Sequential Consistency

alle Instruktionen eines Kerns werden in der Programmreihenfolge ausgeführt

garantiert in Programmreihenfolge:

Lesen ⟶ Lesen
Lesen ⟶ Schreiben
Schreiben ⟶ Lesen
Schreiben ⟶ Schreiben

Total Store Order

Speicherkonsistenzmodelle

Sequential Consistency

alle Instruktionen eines Kerns werden in der Programmreihenfolge ausgeführt

garantiert in Programmreihenfolge:

Lesen ⟶ Lesen
Lesen ⟶ Schreiben
Schreiben ⟶ Lesen
Schreiben ⟶ Schreiben

Total Store Order

Lesezugriffe dürfen Schreibzugriffen an derselben Stelle im Speicher zuvorkommen

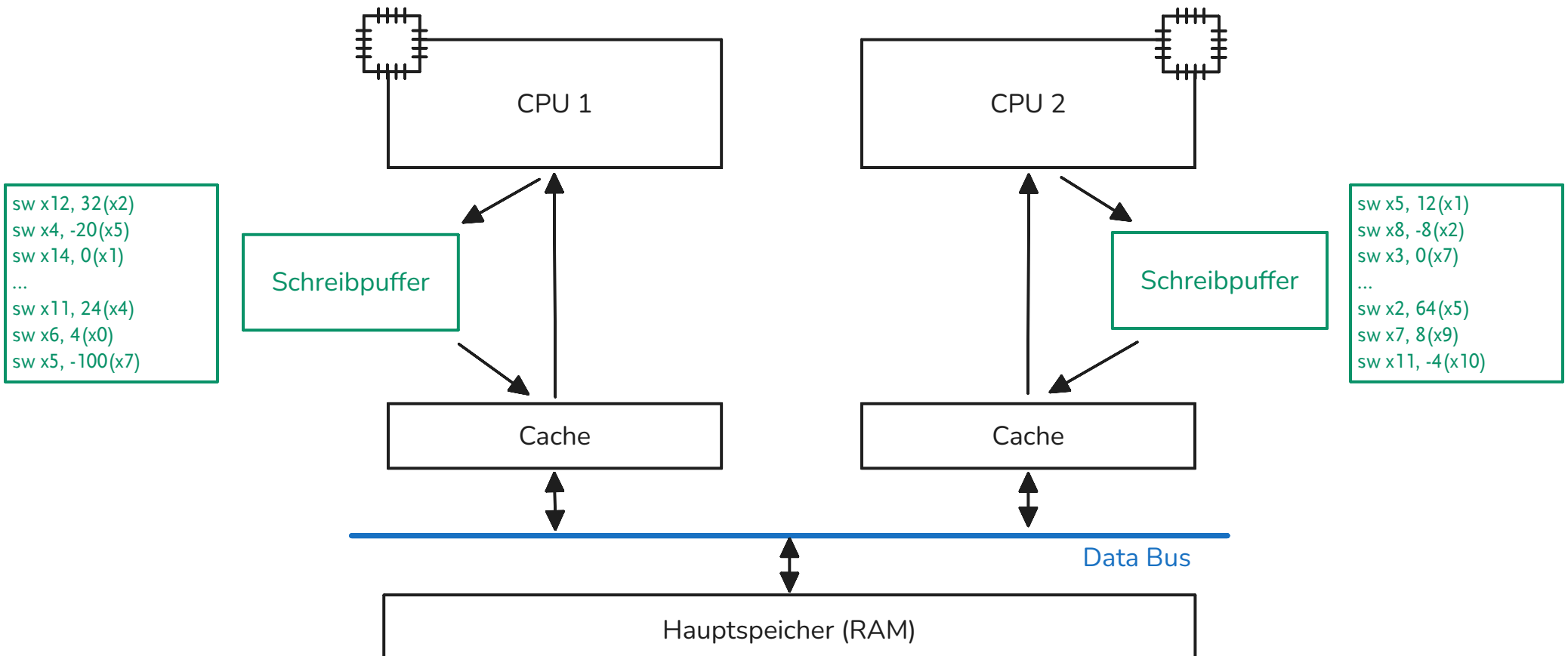
garantiert in Programmreihenfolge:

Lesen ⟶ Lesen
Lesen ⟶ Schreiben
Schreiben ⟶ Schreiben

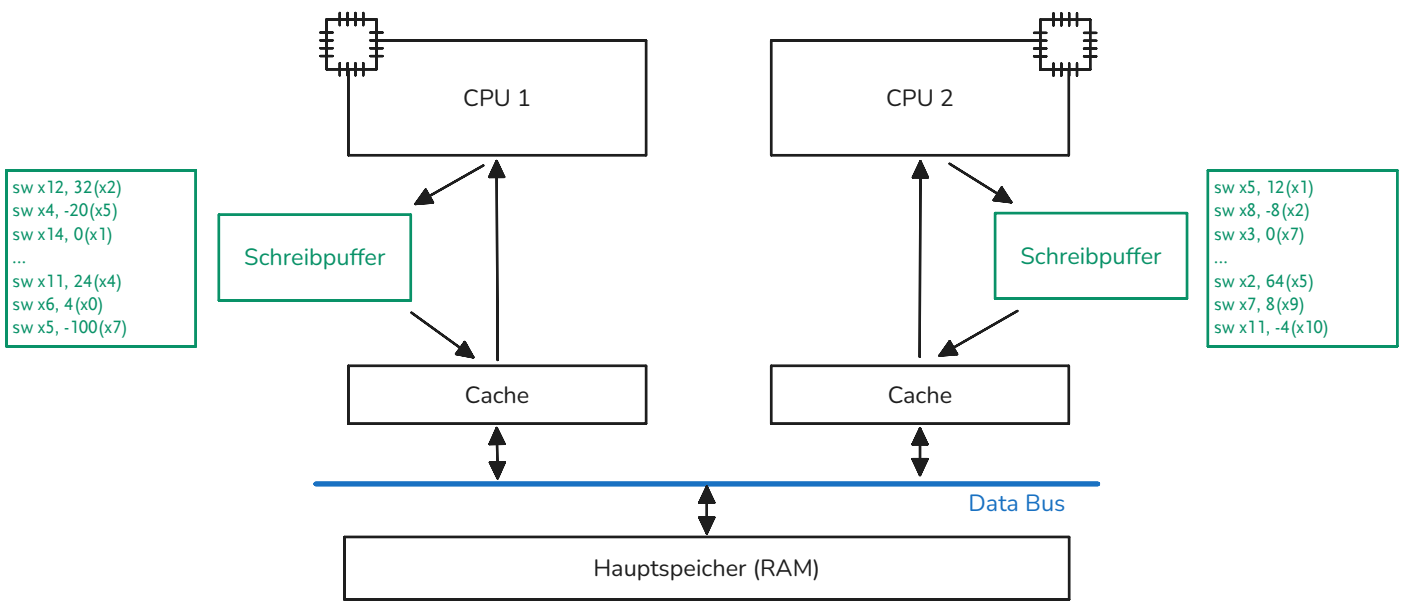
nicht garantiert sequentiell:

Schreiben ⟶ Lesen

Total Store Order mit FIFO-Schreibpuffer



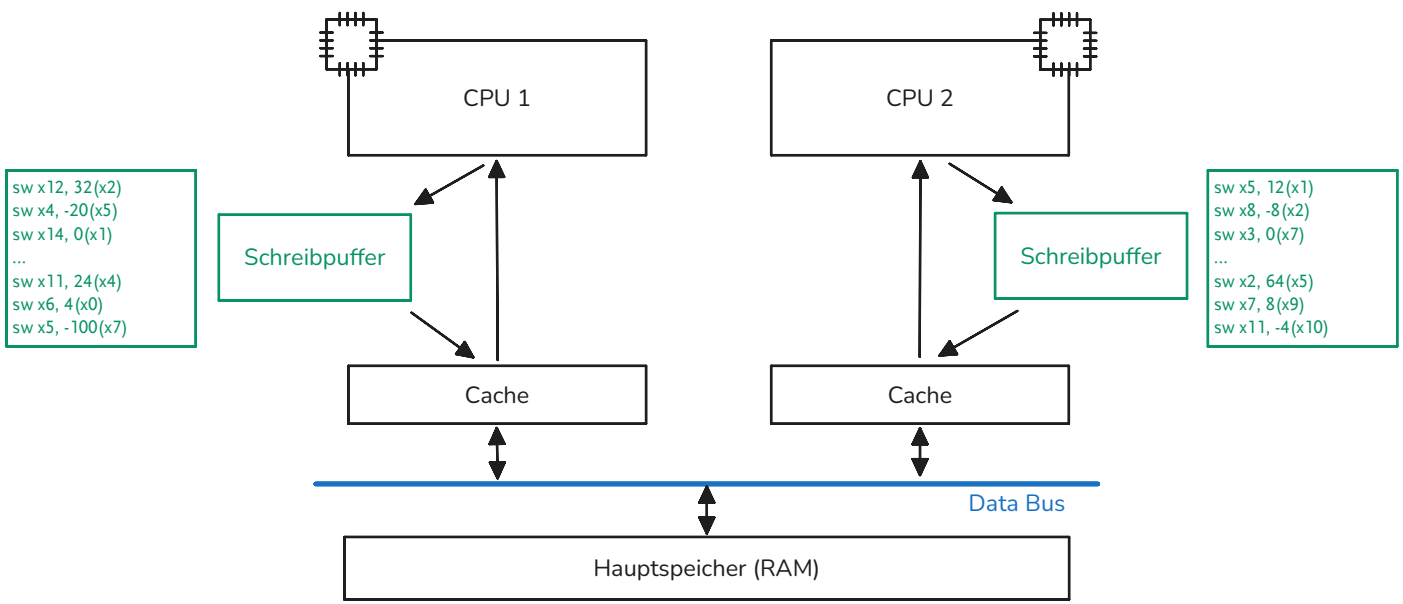
Total Store Order mit FIFO-Schreibpuffer



AUFGABE 1

Warum könnte diese Aufweichung wünschenswert sein?

Total Store Order mit FIFO-Schreibpuffer



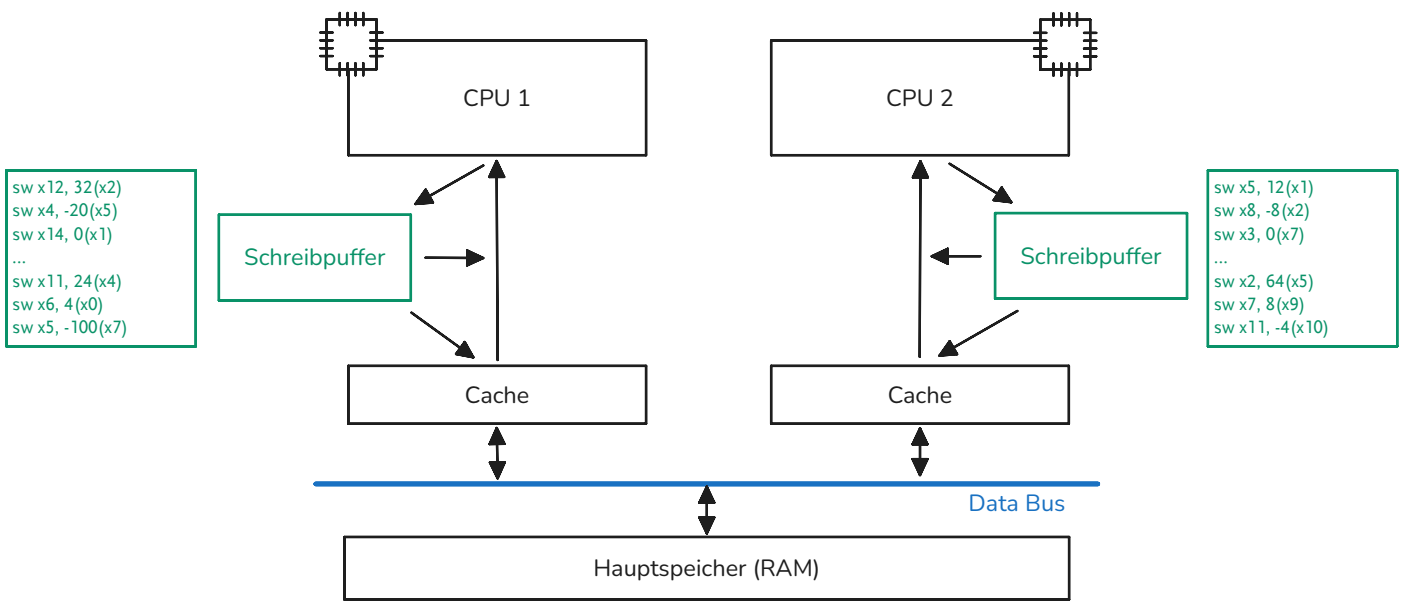
AUFGABE 1

Warum könnte diese Aufweichung wünschenswert sein?

AUFGABE 2

Welche Variationen des Schreibpuffers gibt es?

Varianten des Schreibpuffers



Schreibweiterleitung (*Store Forwarding*): lokaler Prozessor sieht ausstehende *Stores*

Keine FIFO-Reihenfolge: Aufweichung von Schreiben ⟶ Schreiben

Anwendungsbeispiel: Petersons Algorithmus

Übernommen von: Operating Systems Group, Technische Universität Dresden

```
// global variables and initial values
int interested0 = 0, interested1 = 0, turn = 0;
```

```
// process on CPU 0
interested0 = 1;
turn = 1;
while (turn == 1
  && interested1);
// critical section
interested0 = 0;
```

```
// process on CPU 1
interested1 = 1;
turn = 0;
while (turn == 0
  && interested0);
// critical section
interested1 = 0;
```

AUFGABE 3

Unter welchen Modellen funktioniert dieser Algorithmus?

Sequential Consistency

Total Store Order

Weak Ordering

Anwendungsbeispiel: Petersons Algorithmus

Übernommen von: Operating Systems Group, Technische Universität Dresden

```
// global variables and initial values
int interested0 = 0, interested1 = 0, turn = 0;
```

```
// process on CPU 0
interested0 = 1;
turn = 1;
while (turn == 1
  && interested1);
// critical section
interested0 = 0;
```

```
// process on CPU 1
interested1 = 1;
turn = 0;
while (turn == 0
  && interested0);
// critical section
interested1 = 0;
```

AUFGABE 3

Unter welchen Modellen funktioniert dieser Algorithmus?

- ✓ *Sequential Consistency*
- ✗ *Total Store Order*
- ✗ *Weak Ordering*

Fragen?

Literatur

[1] D. A. Patterson und J. L. Hennessy, *Computer organization and design the hardware/software interface*, [First edition]. Cambridge, MA: Morgan Kaufmann, 2018. [Online]. Verfügbar unter: **https://learning.oreilly.com/library/view/-/9780128122761/?ar**

[2] V. Nagarajan, D. J. Sorin, M. D. Hill, und D. A. Wood, *A primer on memory consistency and cache coherence*, Second edition. in Synthesis lectures on computer architecture. Morgan & Claypool Publishers, 2020. [Online]. Verfügbar unter: **https://doi.org/10.1007/978-3-031-01764-3**

Lizenzhinweise

Kursmaterialien · **Quelltext**

Diese Präsentation wurde zuletzt am 14.07.2026 bearbeitet. Sie basiert auf den Folien der zugehörigen Vorlesung von Prof. Dr. Michael Engel. Sofern nicht anders angegeben, sind die Inhalte unter der **Lizenz CC BY-SA 4.0** verfügbar. Das Universitätslogo sowie die Schriftart UB Scala Sans sind Eigentum der Otto-Friedrich-Universität Bamberg.

Folgende **Open-Source-Komponenten** kommen in der Präsentation zum Einsatz: **circuiteria** (Apache 2.0), **finite** (MIT), **fontawesome** (SIL), **pgf-tikz** (GPL 2.0), **typst** (Apache 2.0), **typst-ccicons** (MIT), **typst-polylux** (MIT).