Transactional Memory

Simon Schuster

Seminar: Konzepte von
Betriebssystem-Komponenten
Sommersemester 2013

27. Juni 2013

Transactional Memory: Architectural Support for Lock-Free Data Structures

Maurice Herlihy
Digital Equipment Corporation
Cambridge Research Laboratory
Cambridge MA 02139
herlihy@crl.dec.com

J. Eliot B. Moss
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
moss@cs.umass.edu

Abstract

A shared data structure is *lock-free* if its operations do not require mutual exclusion. If one process is interrupted in the middle of an operation, other processes will not be prevented from operating on that object. In highly concurrent systems, lock-free data structures avoid common problems associated with conventional locking techniques, including priority inversion, convoying, and difficulty of avoiding deadlock. This paper introduces *transactional memory*, a new multiprocessor architecture intended to make lock-free synchronization as efficient (and easy to use) as conventional techniques based on mutual exclusion. Transactional memory allows programmers to define customized read-modify-write operations that apply to multiple, independently-chosen words of memory. It is implemented by straightforward extensions to any multiprocessor cache-coherence protocol. Simulation results show that transactional memory matches or outperforms the best known locking techniques for simple benchmarks, even in the absence of priority inversion, convoying, and deadlock.

1 Introduction

A shared data structure is *lock-free* if its operations do not require mutual exclusion. If one process is interrupted in the middle of an operation, other processes will not be prevented from operating on that object. Lock-free data

structures avoid common problems assoc
ventional locking techniques in highly con

- *Priority inversion* occurs when a lo
cess is preempted while holding a
higher-priority processes.
- *Convoying* occurs when a process hol
scheduled, perhaps by exhausting its s
tum, by a page fault, or by some other
When such an interruption occurs, oth
pable of running may be unable to pr
- *Deadlock* can occur if processes att
same set of objects in different or
avoidance can be awkward if processe
tiple data objects, particularly if the
not known in advance.

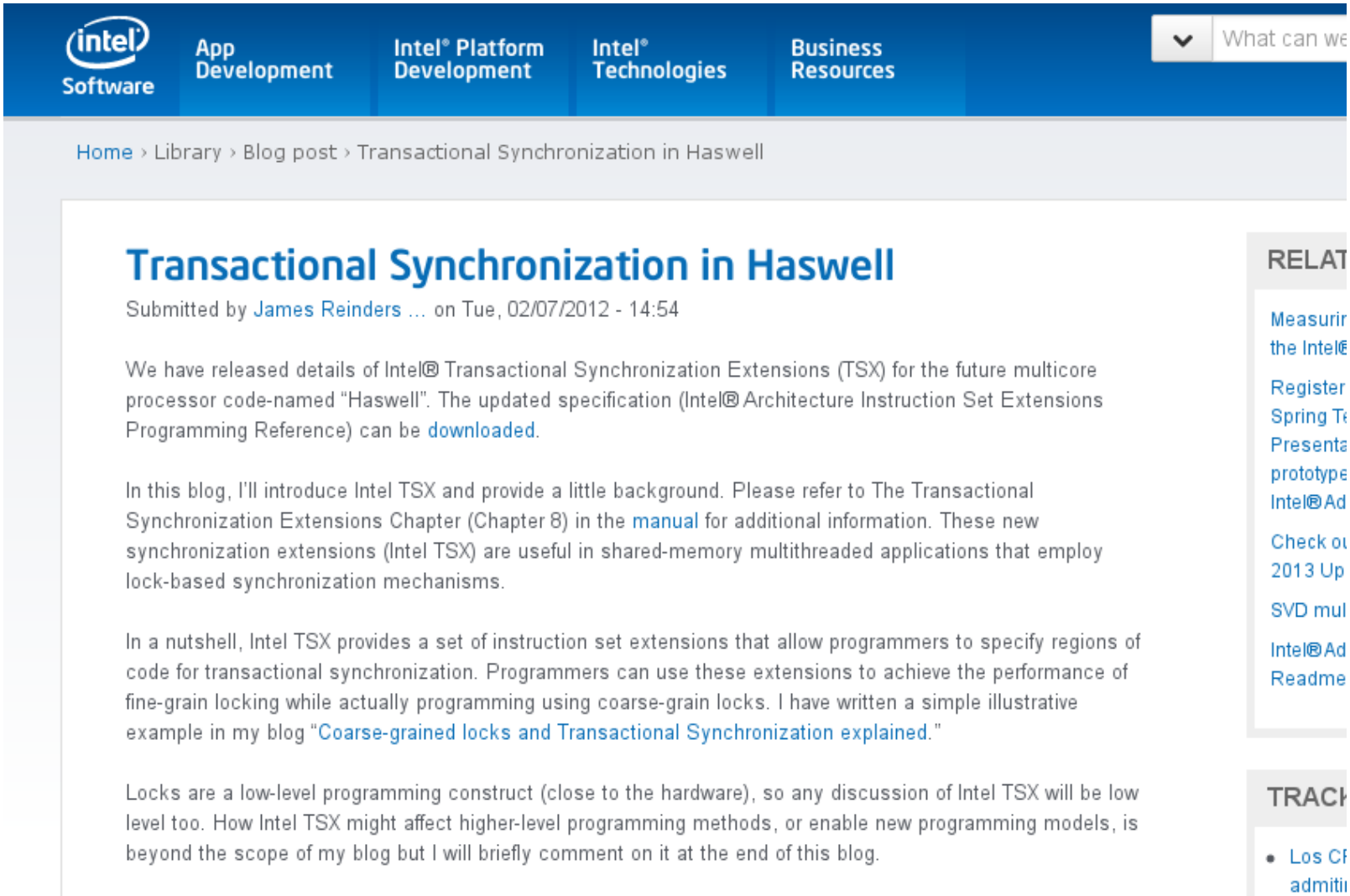
A number of researchers have investigate
implementing lock-free concurrent data
software techniques [2, 4, 19, 25, 26, 32
evidence suggests that in the absence of
voying, or deadlock, software implemen
free data structures often do not perform
locking-based counterparts.

This paper introduces *transactional me*
tiprocessor architecture intended to mak
chronization as efficient (and easy to use)
techniques based on mutual exclusion. Tra
ory allows programmers to define customi
write operations that apply to multiple,
chosen words of memory. It is implement
ward extensions to multiprocessor cache-
cols. Simulation results show that transac
competitive with the best known lock-bas
simple benchmarks, even in the absence
sion, convoys, and deadlock.

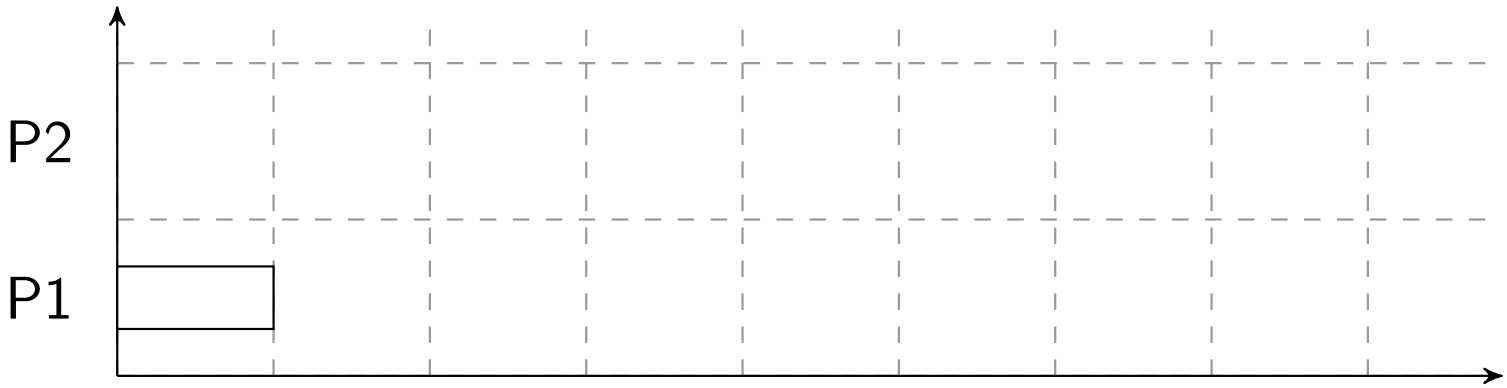
In Section 2, we describe transactional
to use it. In Section 3 we describe one w
transactional memory, and in Section 4

$2013 - 1993 = 20$

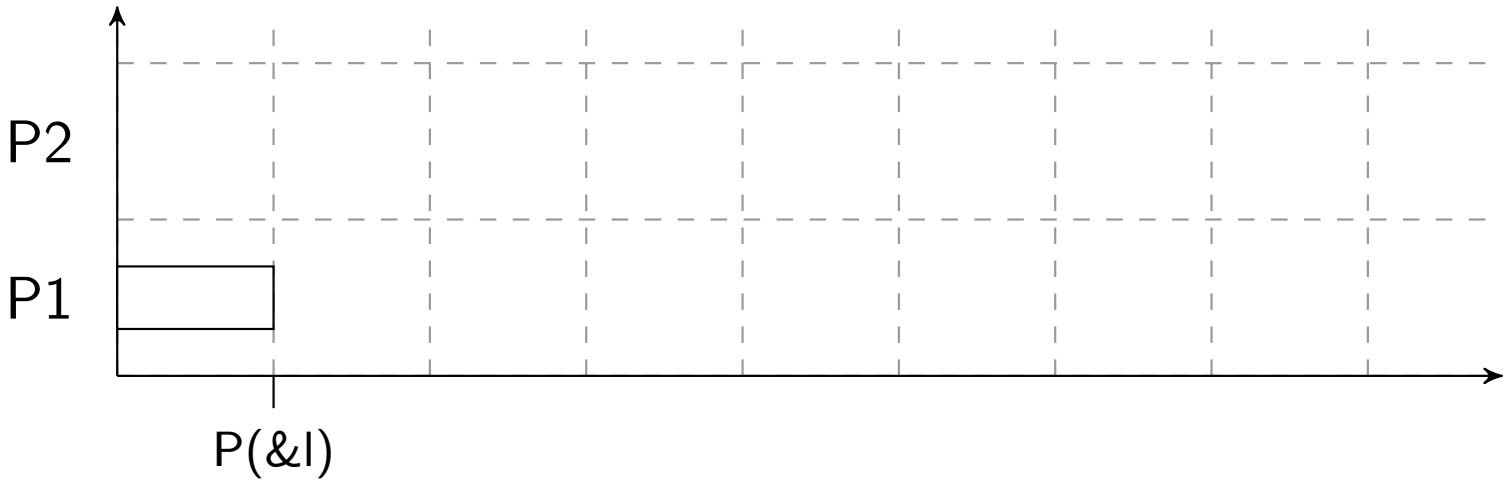
$2013 - 1993 = 20$


intel Software
App Development
Intel® Platform Development
Intel® Technologies
Business Resources
What can we
Home › Library › Blog post › Transactional Synchronization in Haswell
Transactional Synchronization in Haswell
Submitted by James Reinders ... on Tue, 02/07/2012 - 14:54
We have released details of Intel® Transactional Synchronization Extensions (TSX) for the future multicore processor code-named "Haswell". The updated specification (Intel® Architecture Instruction Set Extensions Programming Reference) can be downloaded.
In this blog, I'll introduce Intel TSX and provide a little background. Please refer to The Transactional Synchronization Extensions Chapter (Chapter 8) in the manual for additional information. These new synchronization extensions (Intel TSX) are useful in shared-memory multithreaded applications that employ lock-based synchronization mechanisms.
In a nutshell, Intel TSX provides a set of instruction set extensions that allow programmers to specify regions of code for transactional synchronization. Programmers can use these extensions to achieve the performance of fine-grain locking while actually programming using coarse-grain locks. I have written a simple illustrative example in my blog "Coarse-grained locks and Transactional Synchronization explained."
Locks are a low-level programming construct (close to the hardware), so any discussion of Intel TSX will be low level too. How Intel TSX might affect higher-level programming methods, or enable new programming models, is beyond the scope of my blog but I will briefly comment on it at the end of this blog.
RELAT
TRACK

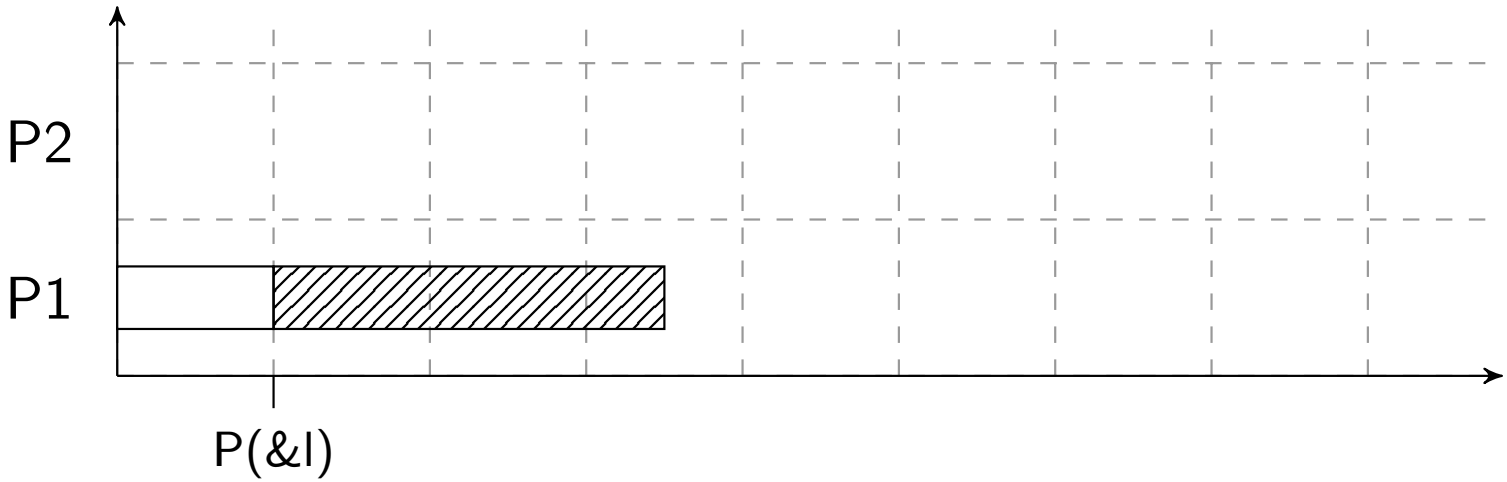
Prioritätsinversion



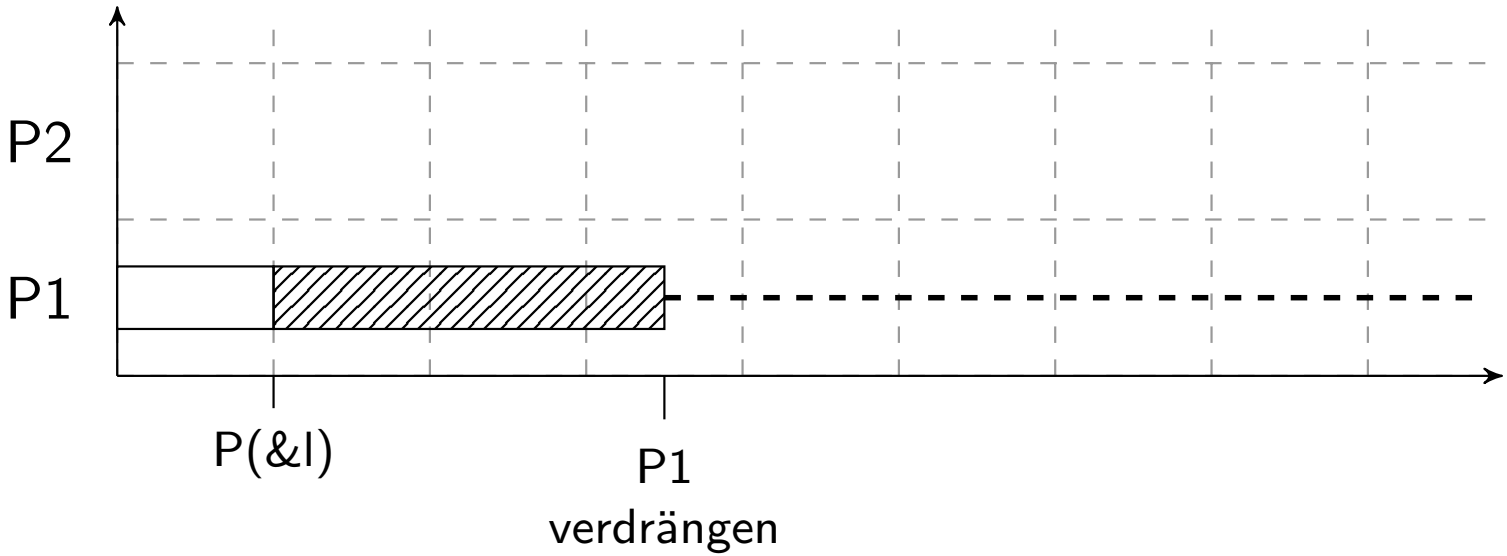
Prioritätsinversion



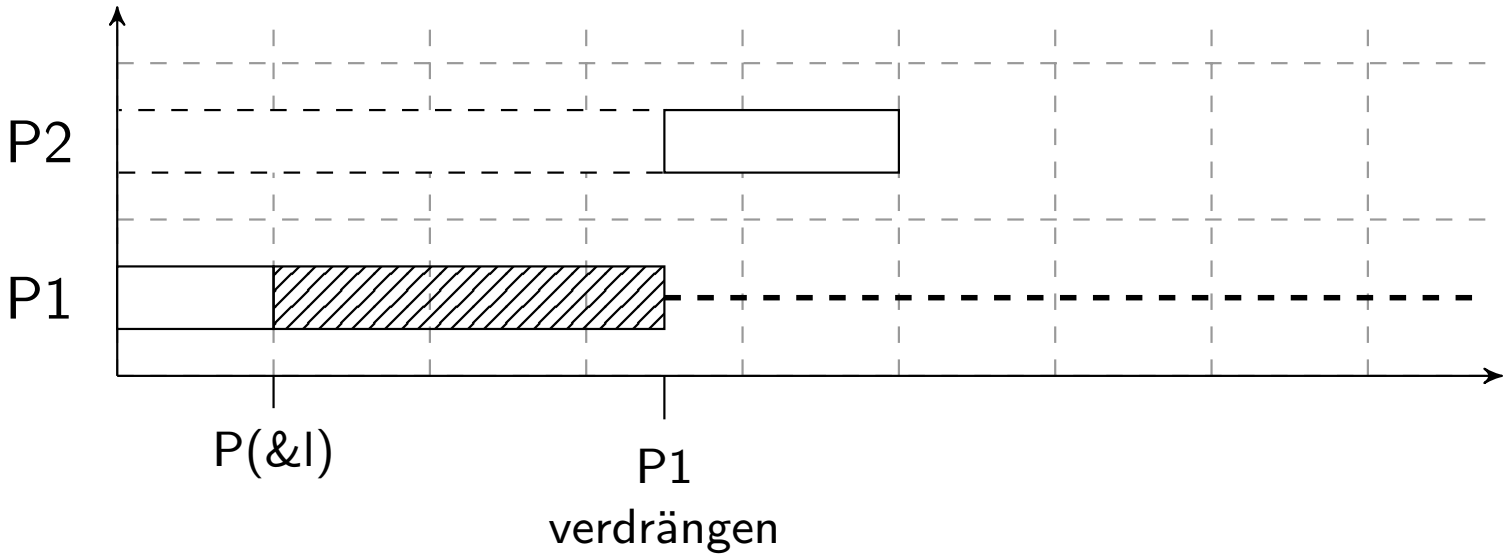
Prioritätsinversion



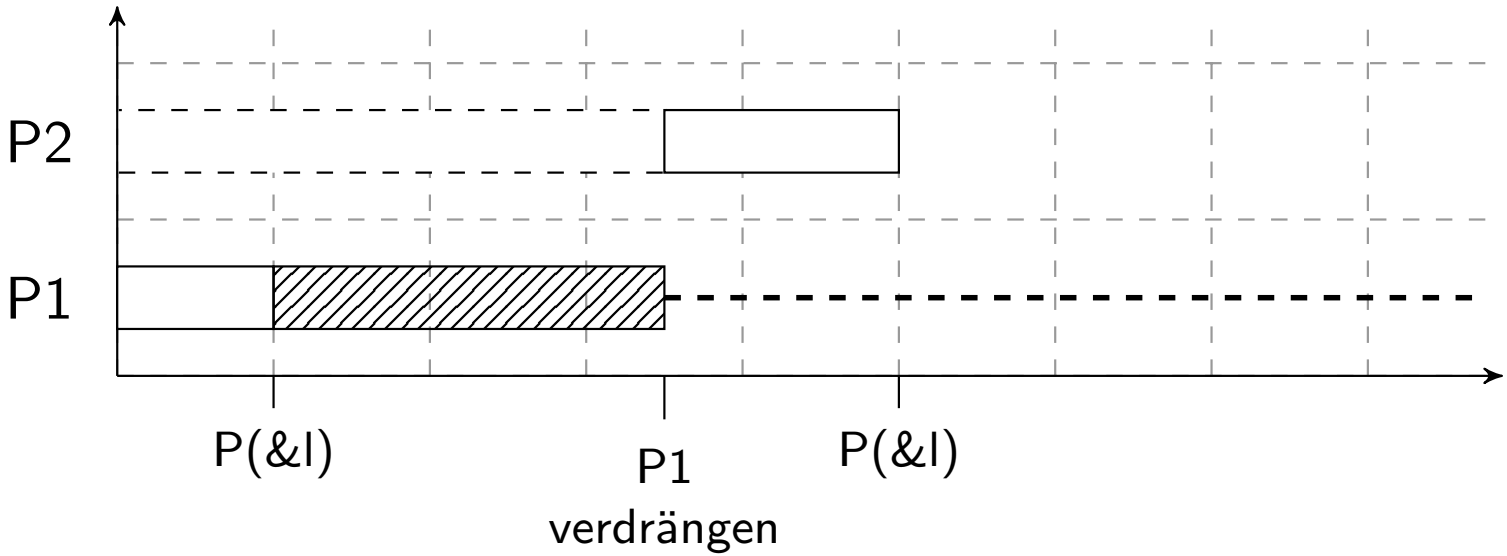
Prioritätsinversion



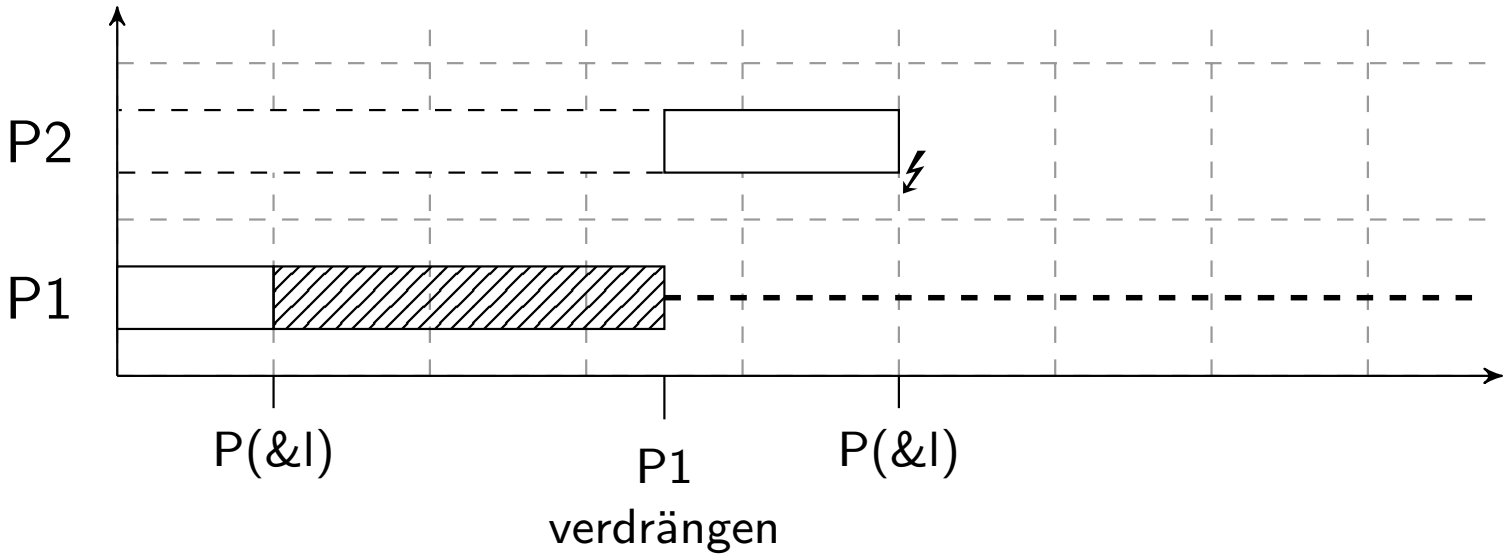
Prioritätsinversion



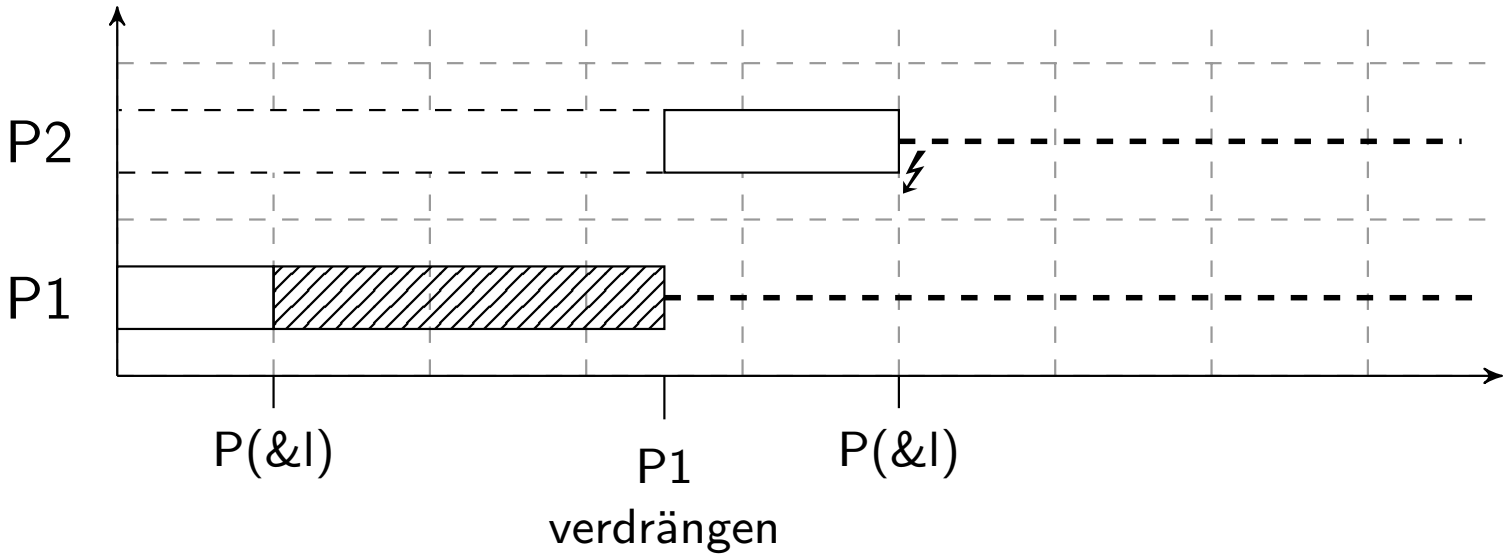
Prioritätsinversion



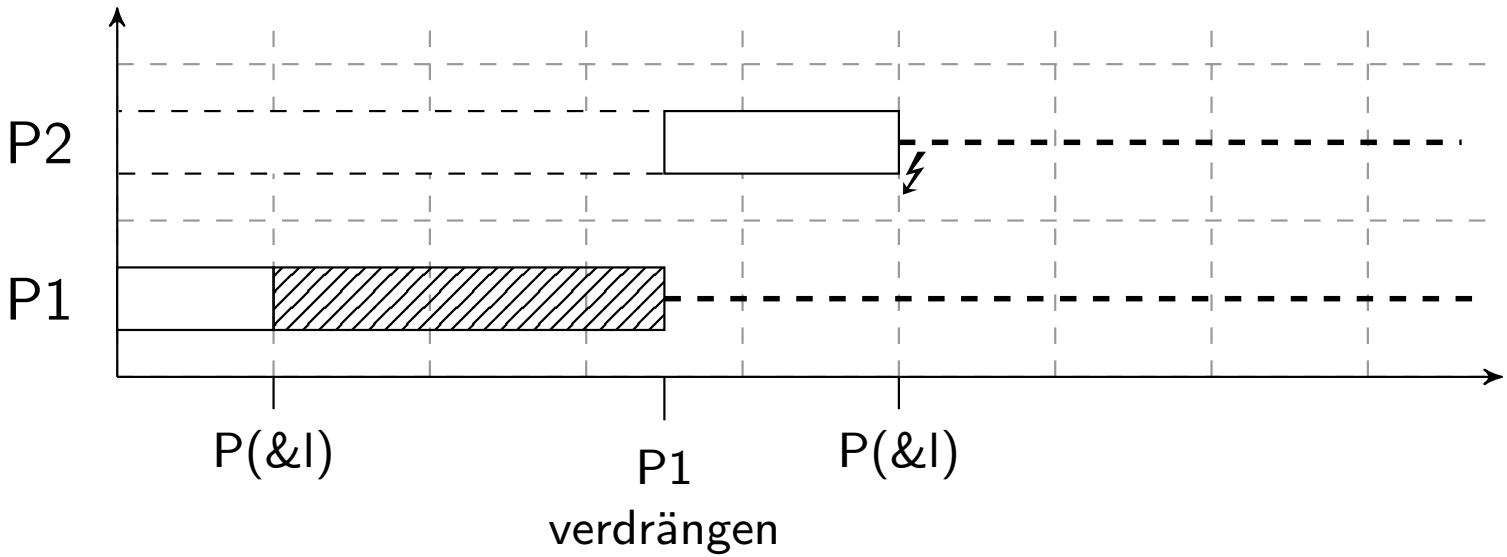
Prioritätsinversion



Prioritätsinversion



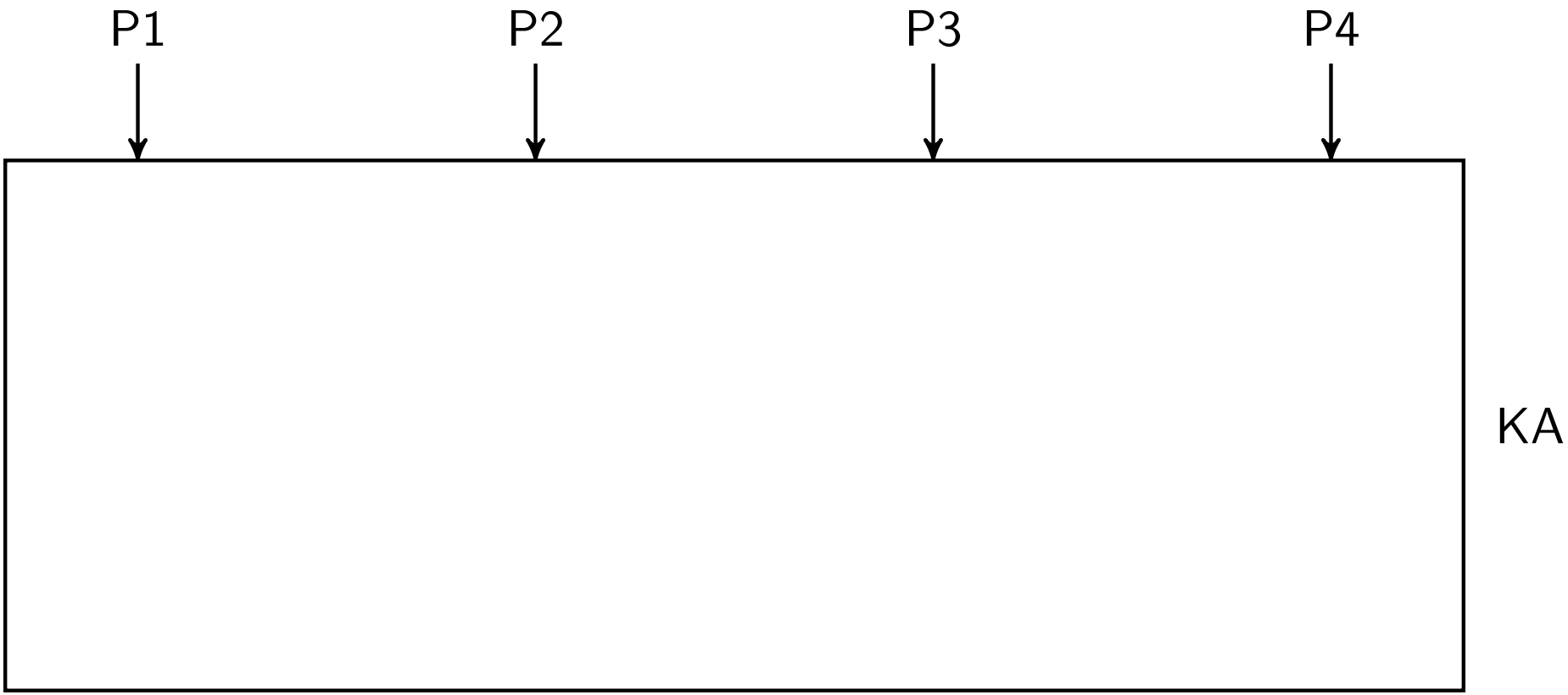
Prioritätsinversion



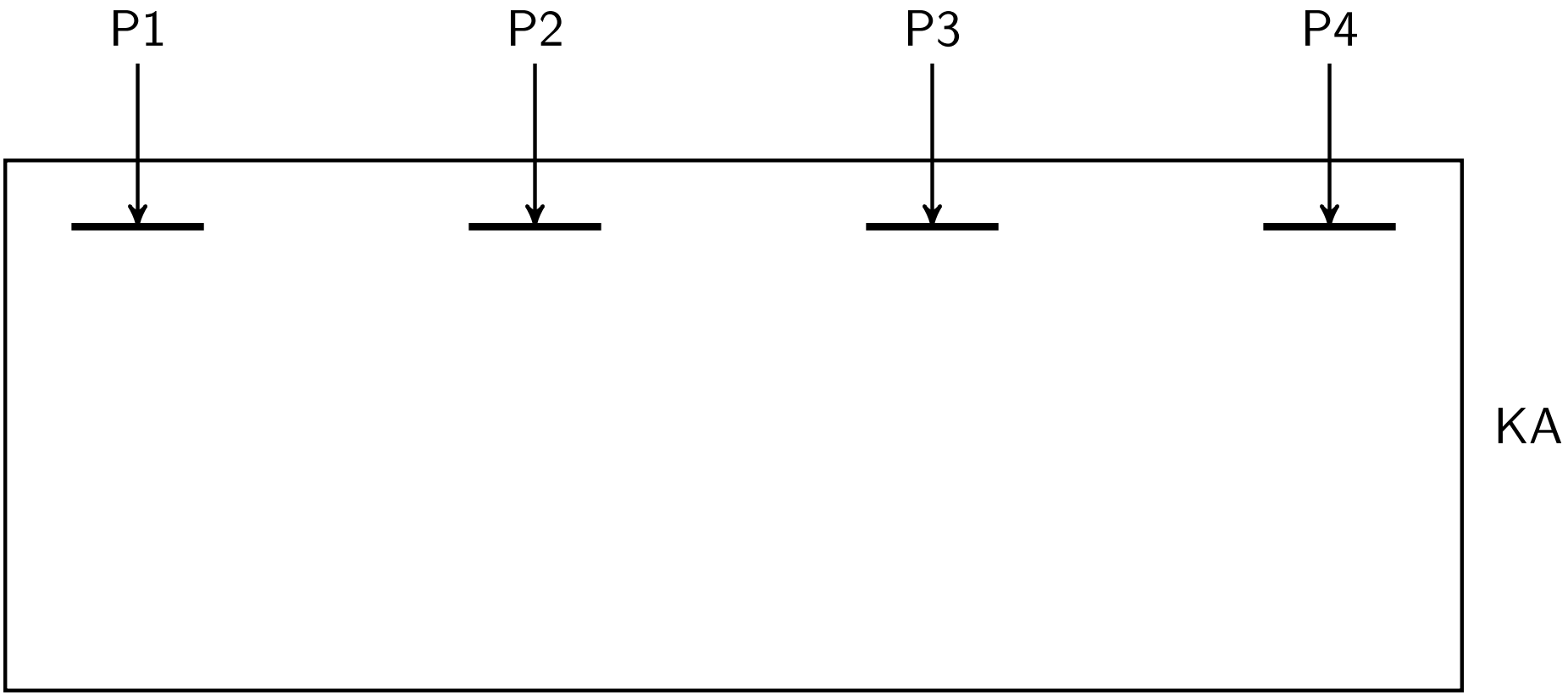
Prioritätsinversion

- Niedrigpriorer Prozess erhält eine Sperre
- Der Prozess wird verdrängt während er die Sperre besitzt
- Hochpriore, die Sperre benötigende Prozesse werden blockiert ↯

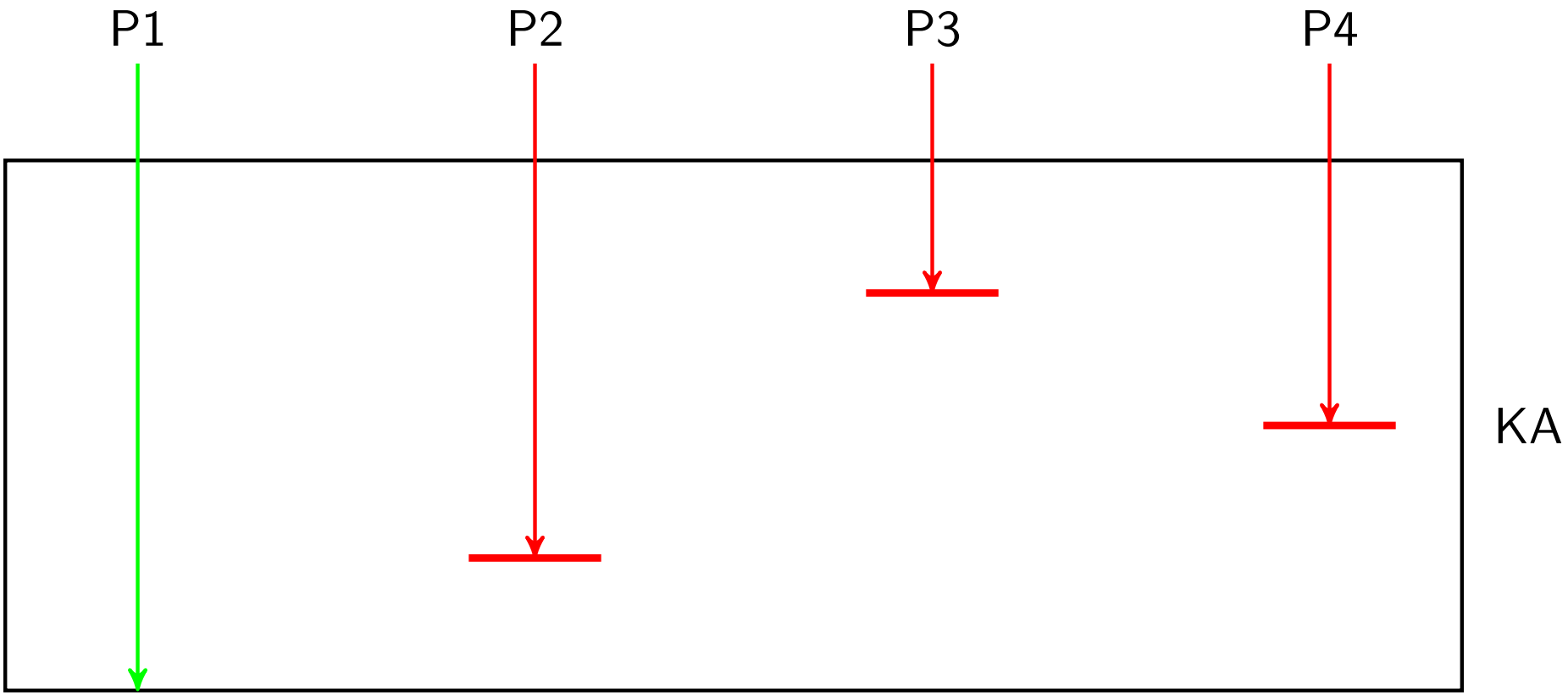
Nichtblockierende Synchronisation



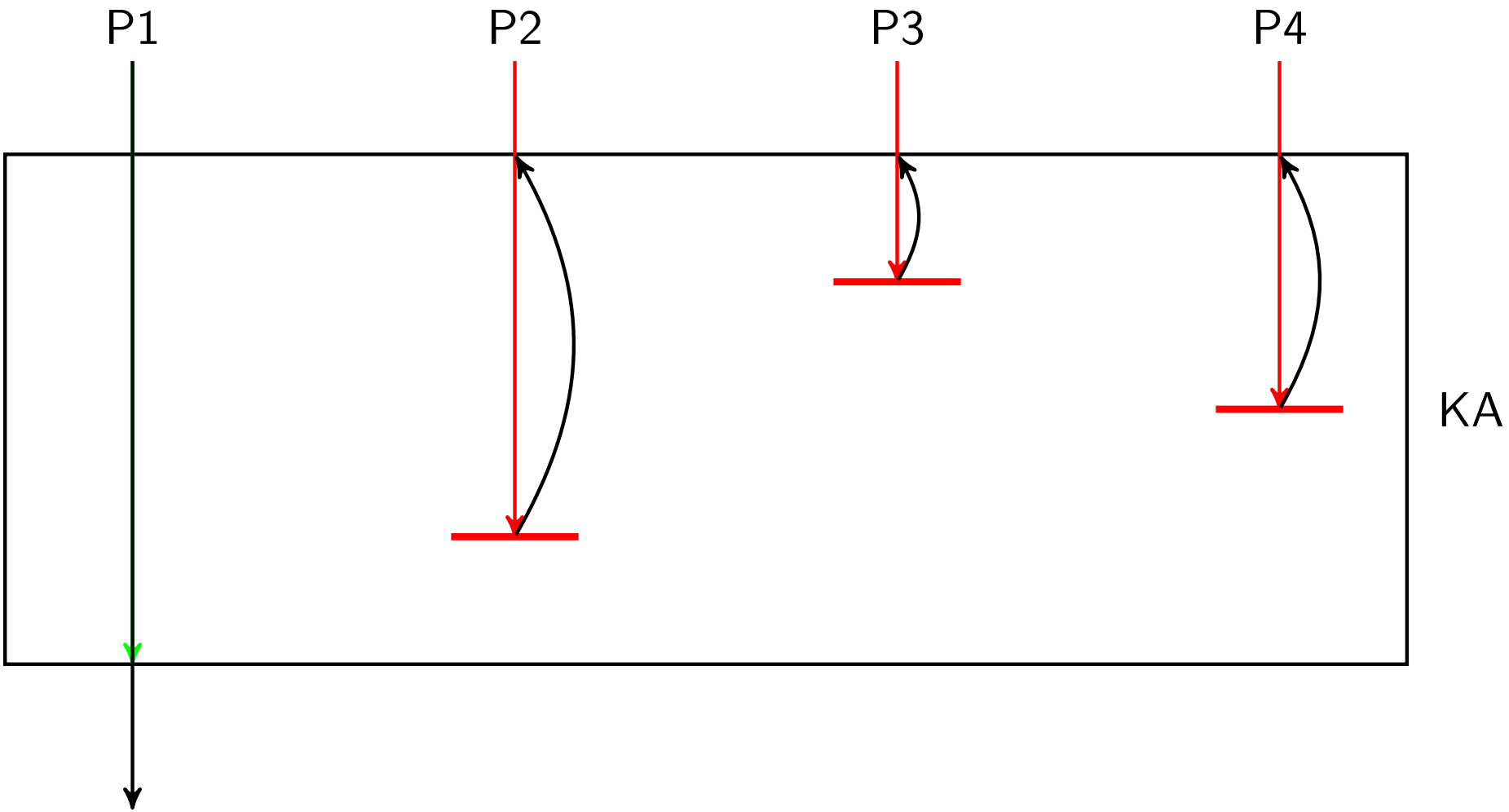
Nichtblockierende Synchronisation



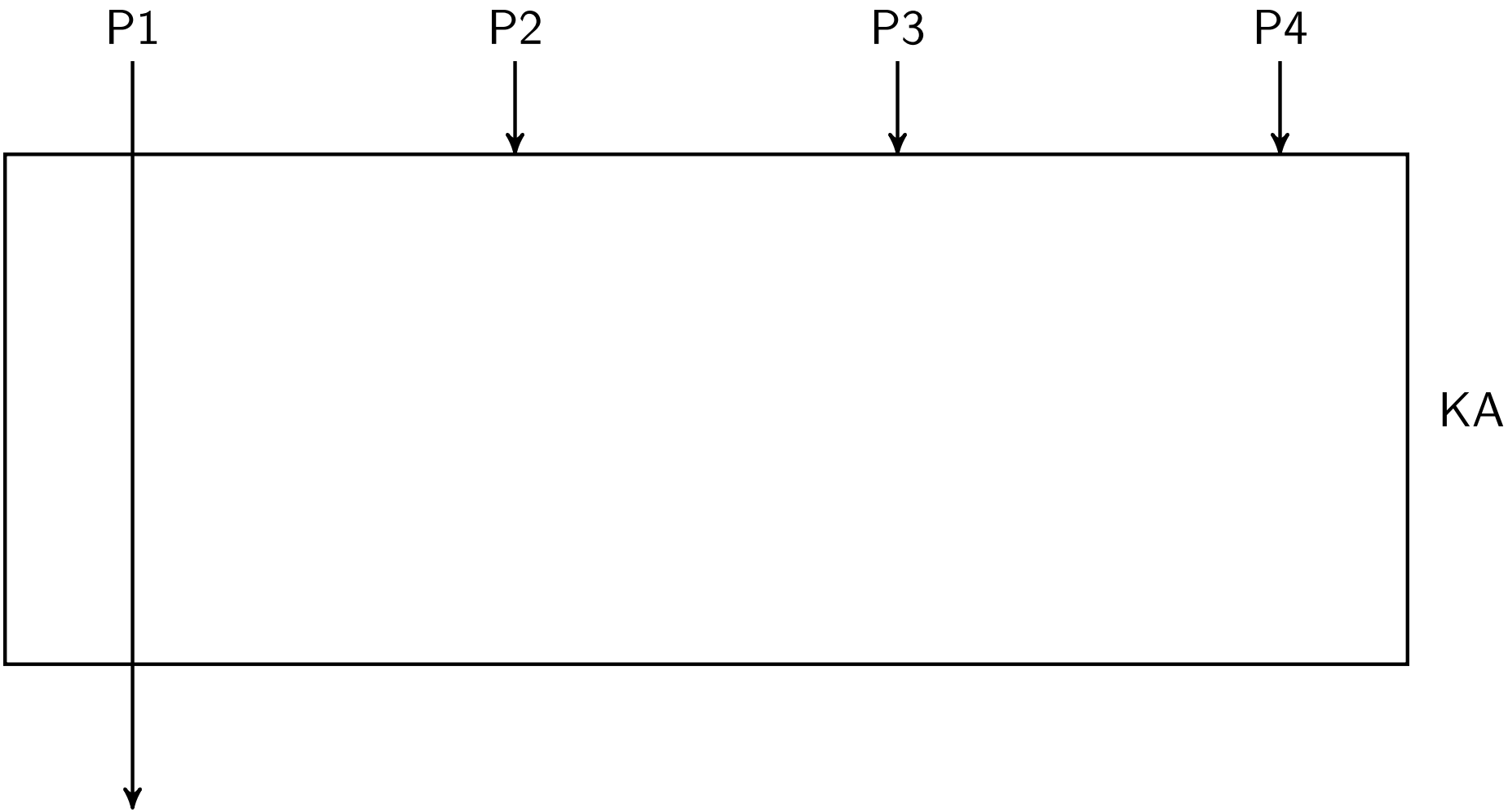
Nichtblockierende Synchronisation



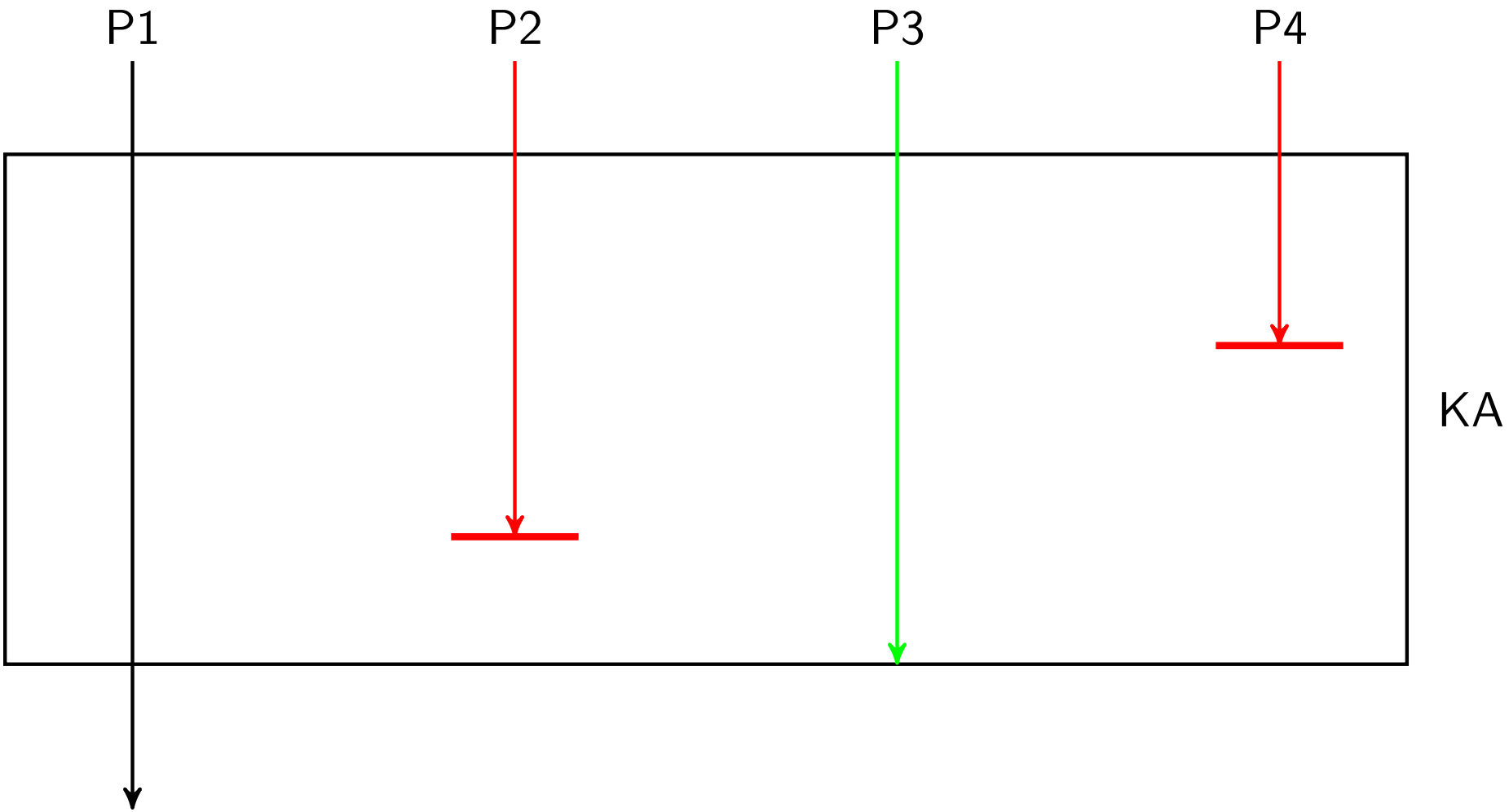
Nichtblockierende Synchronisation



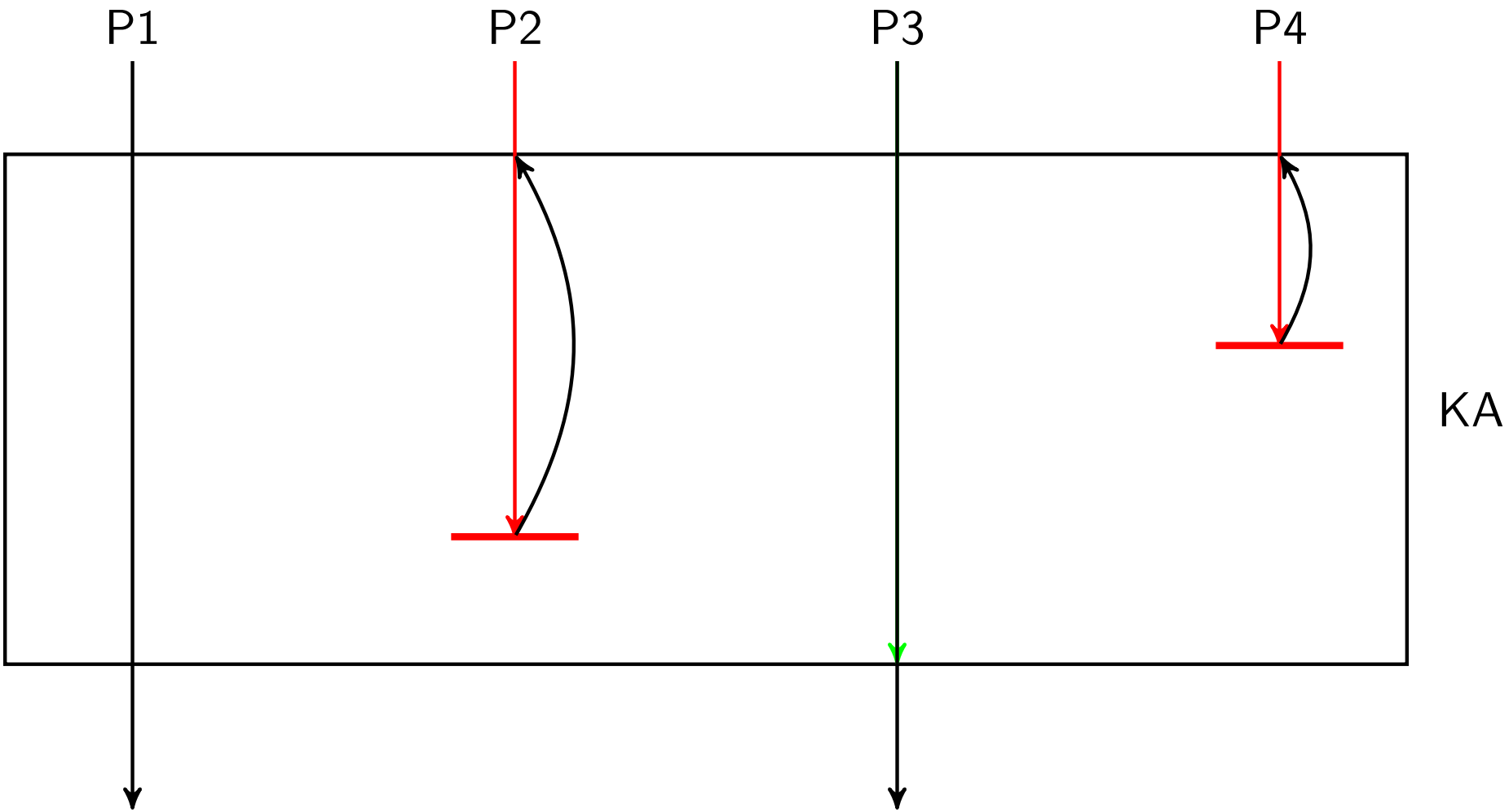
Nichtblockierende Synchronisation



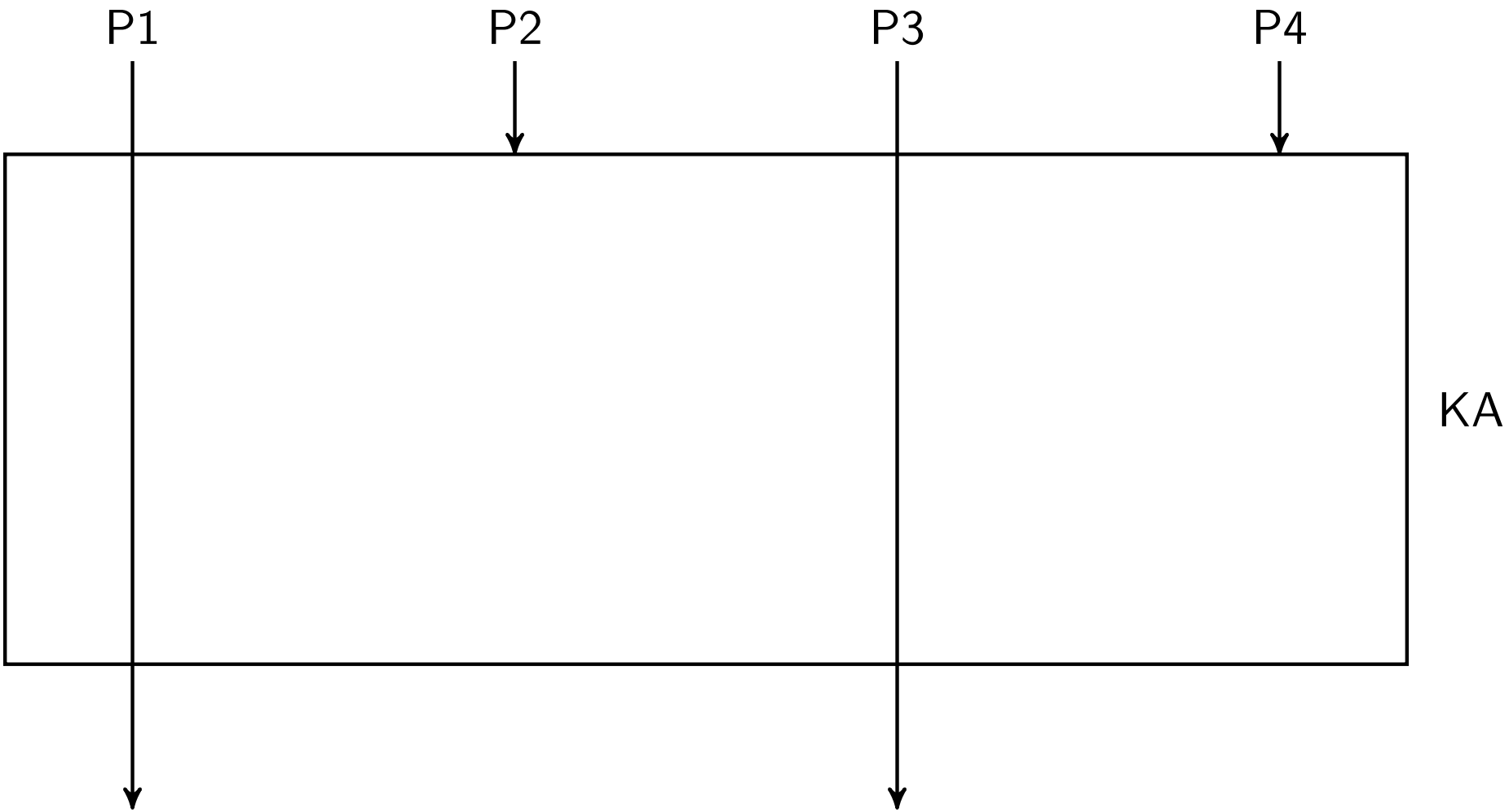
Nichtblockierende Synchronisation



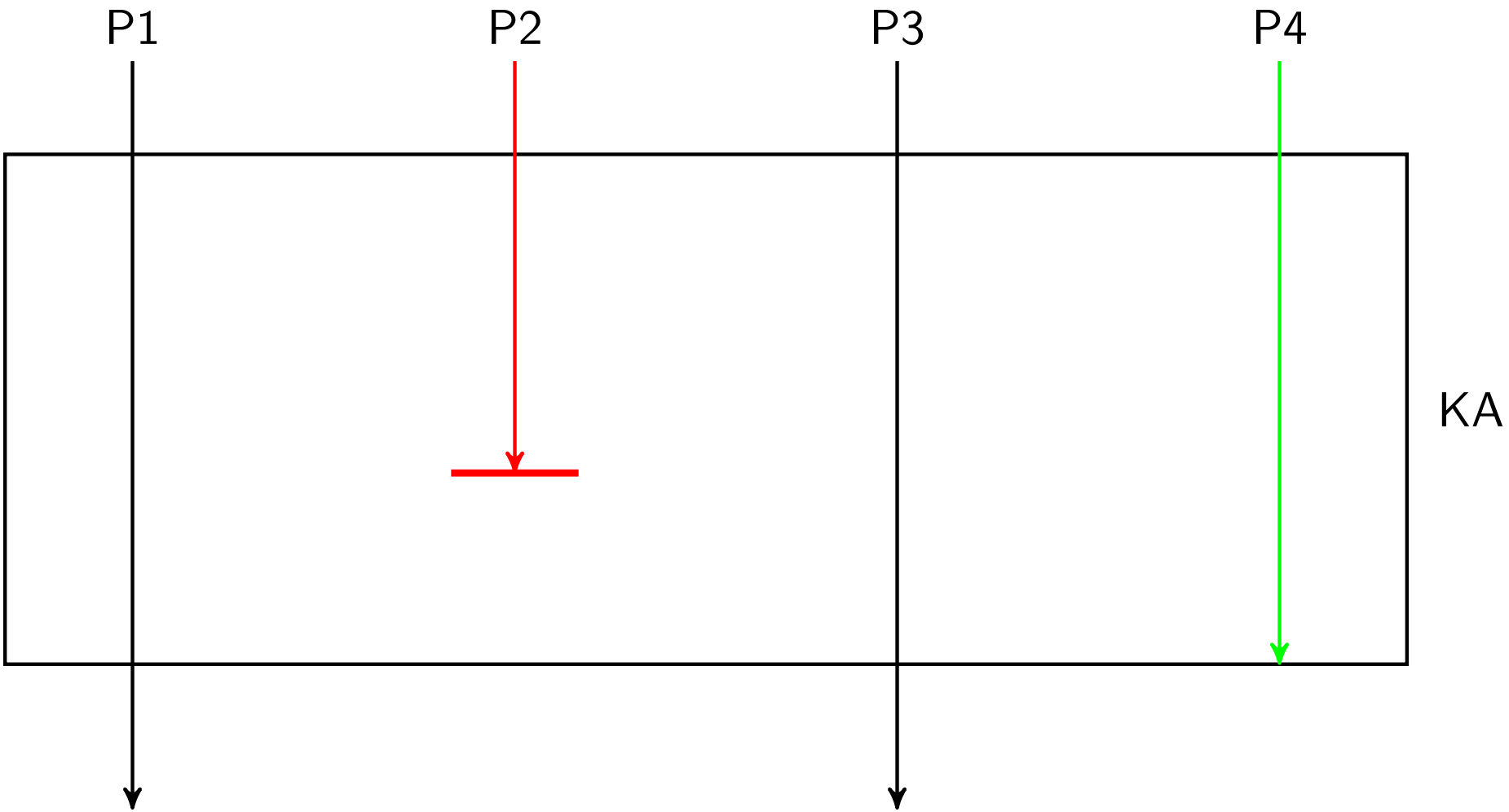
Nichtblockierende Synchronisation



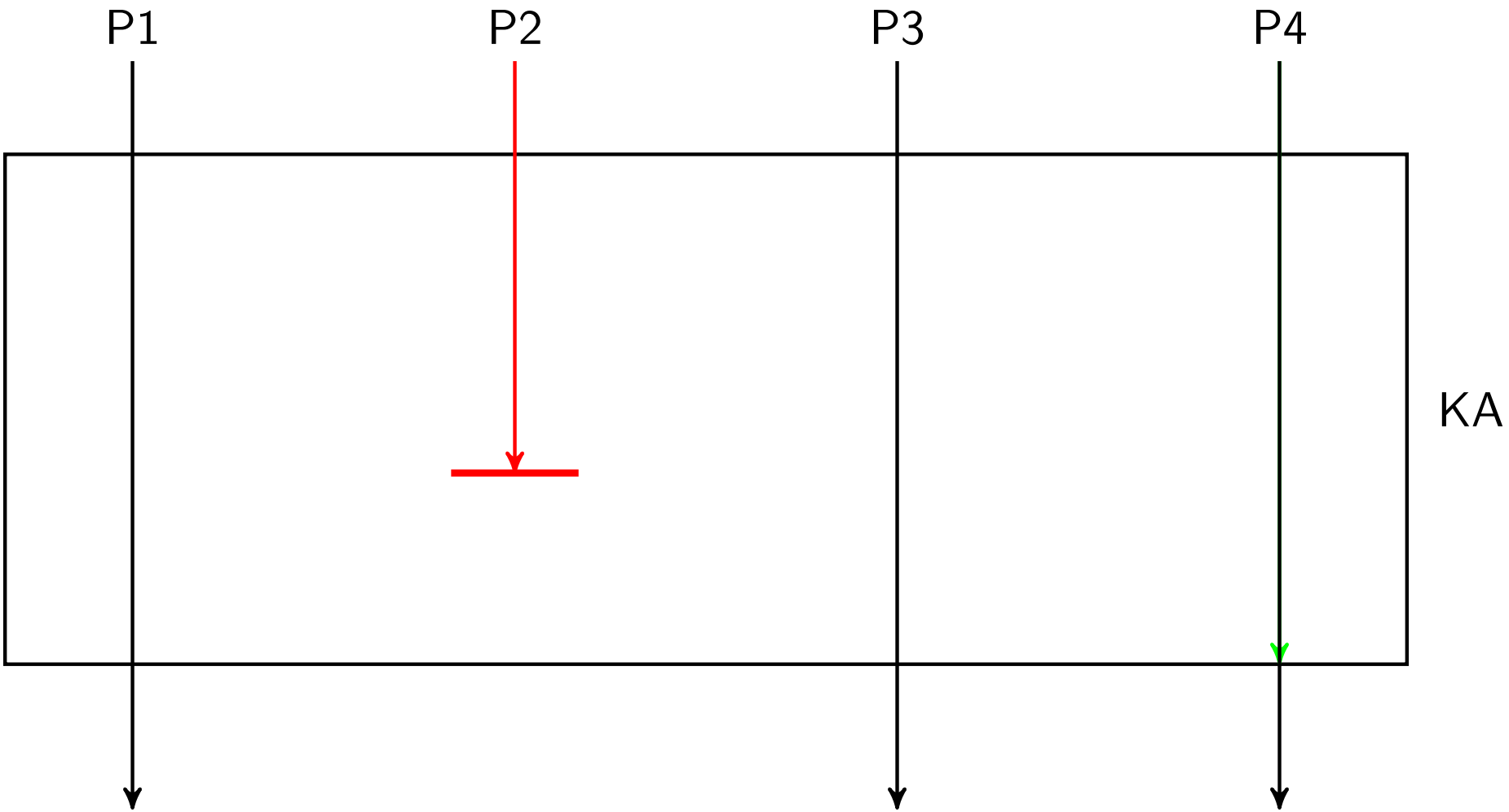
Nichtblockierende Synchronisation



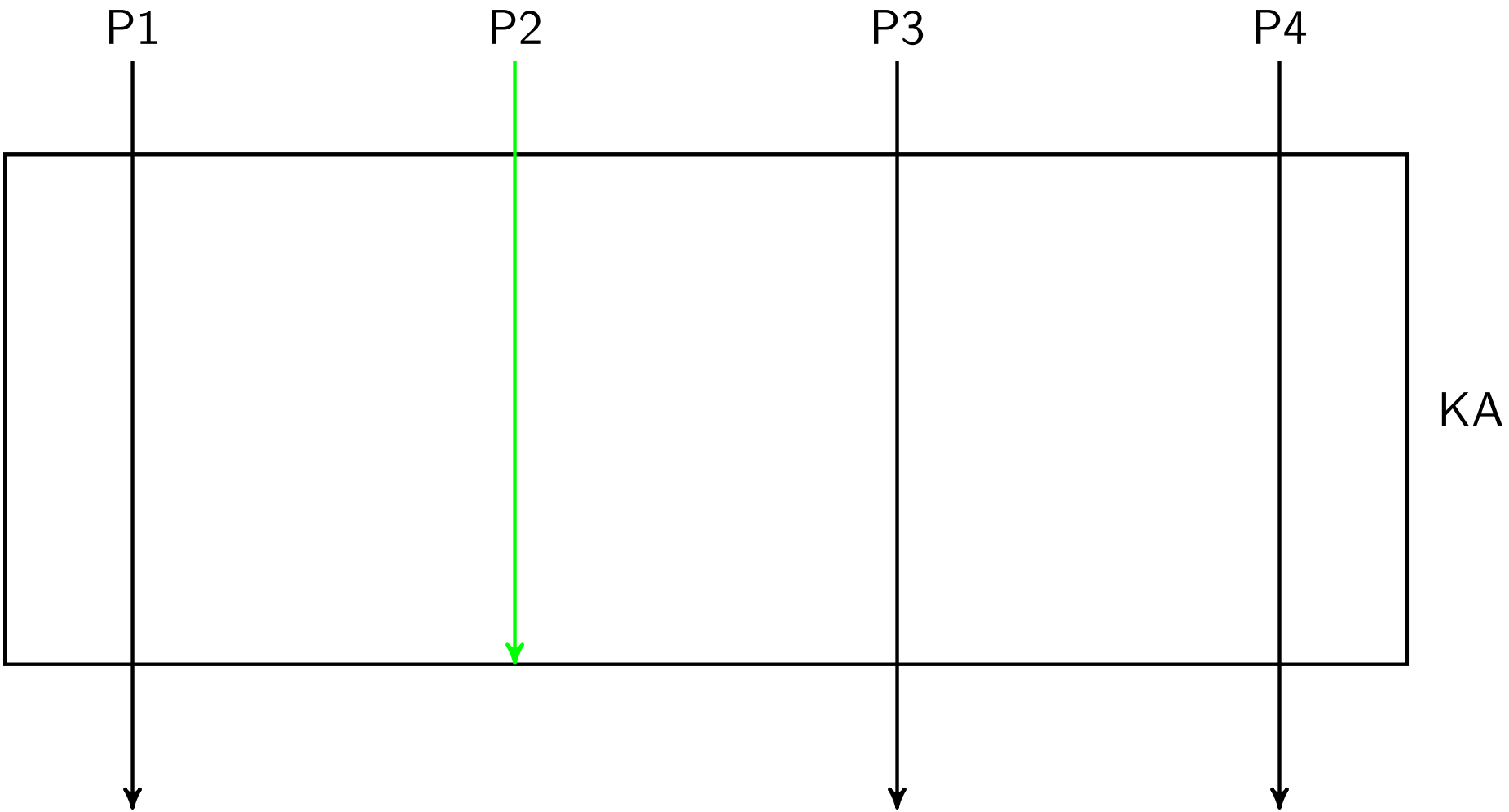
Nichtblockierende Synchronisation



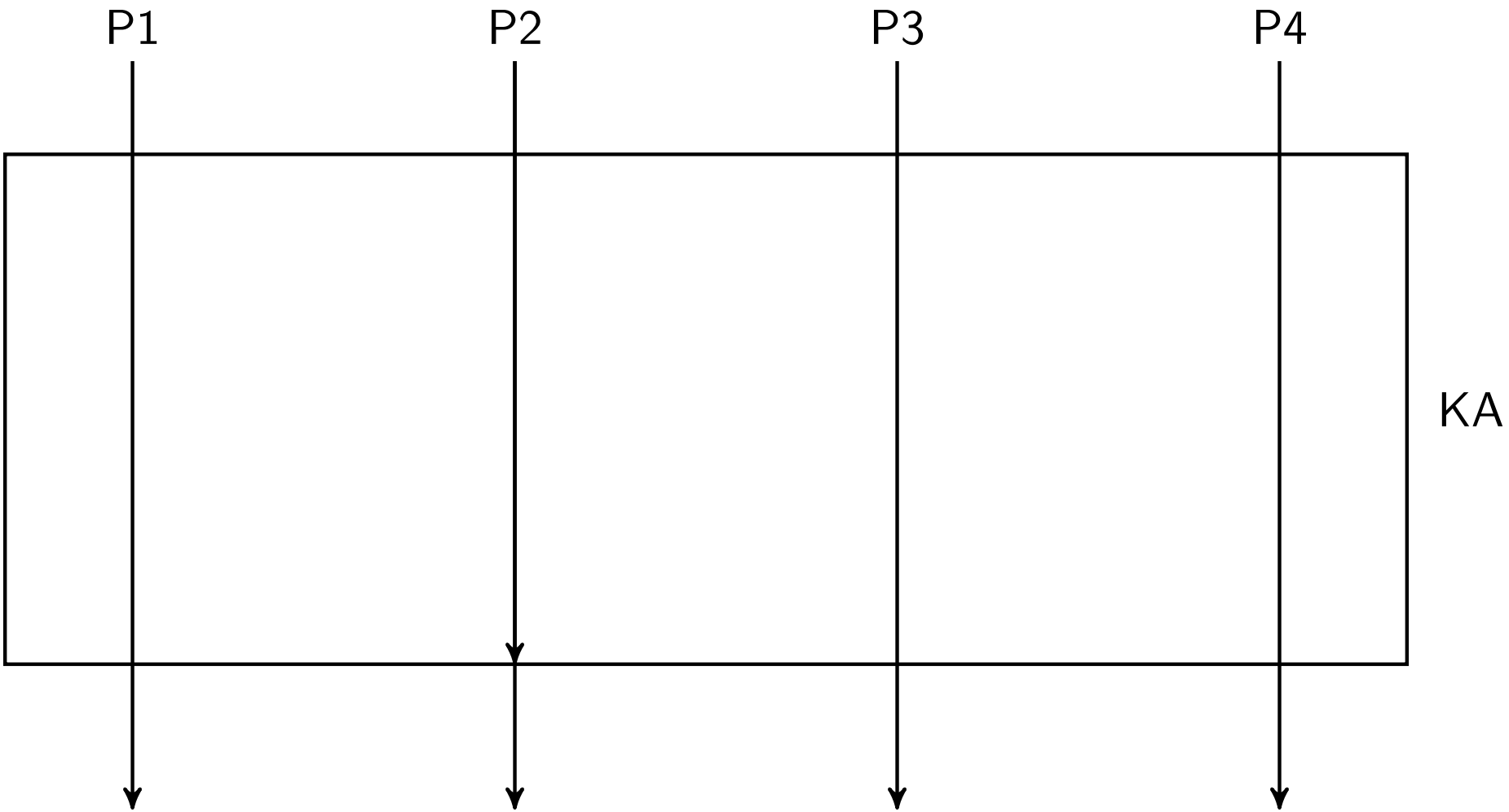
Nichtblockierende Synchronisation



Nichtblockierende Synchronisation



Nichtblockierende Synchronisation



Transaktion

ACID = **A**tomicity, **C**onsistency, **I**solation, **D**urability

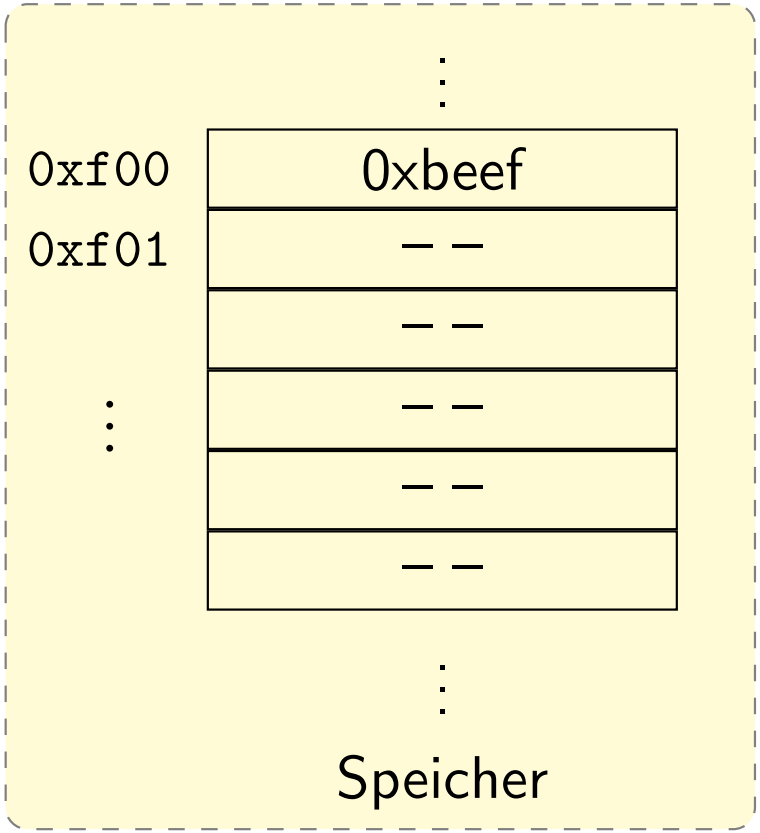
Atomarität	Transaktion ist elementare Operation
Konsistenzerhaltung	Daten sind nach Transaktionsende konsistent
Isolation	Transaktionen sind untereinander entkoppelt
Dauerhaftigkeit	Transaktion garantiert Datensicherheit

Transaktion

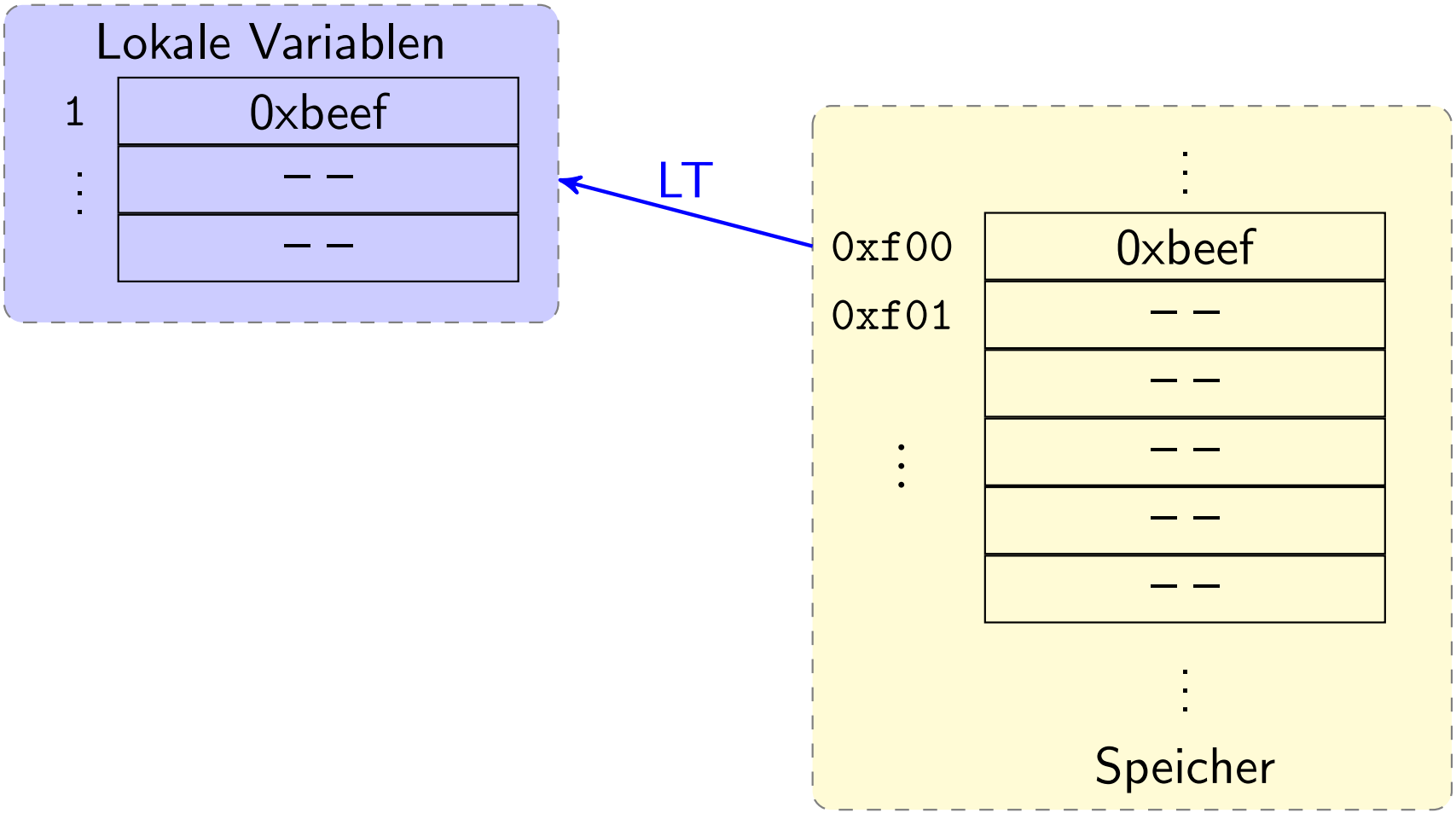
ACID = **A**tomicity, **C**onsistency, **I**solation, **D**urability

Atomarität	Transaktion ist elementare Operation
Konsistenzerhaltung	Daten sind nach Transaktionsende konsistent
Isolation	Transaktionen sind untereinander entkoppelt
~~Dauerhaftigkeit~~	~~Transaktion garantiert Datensicherheit~~

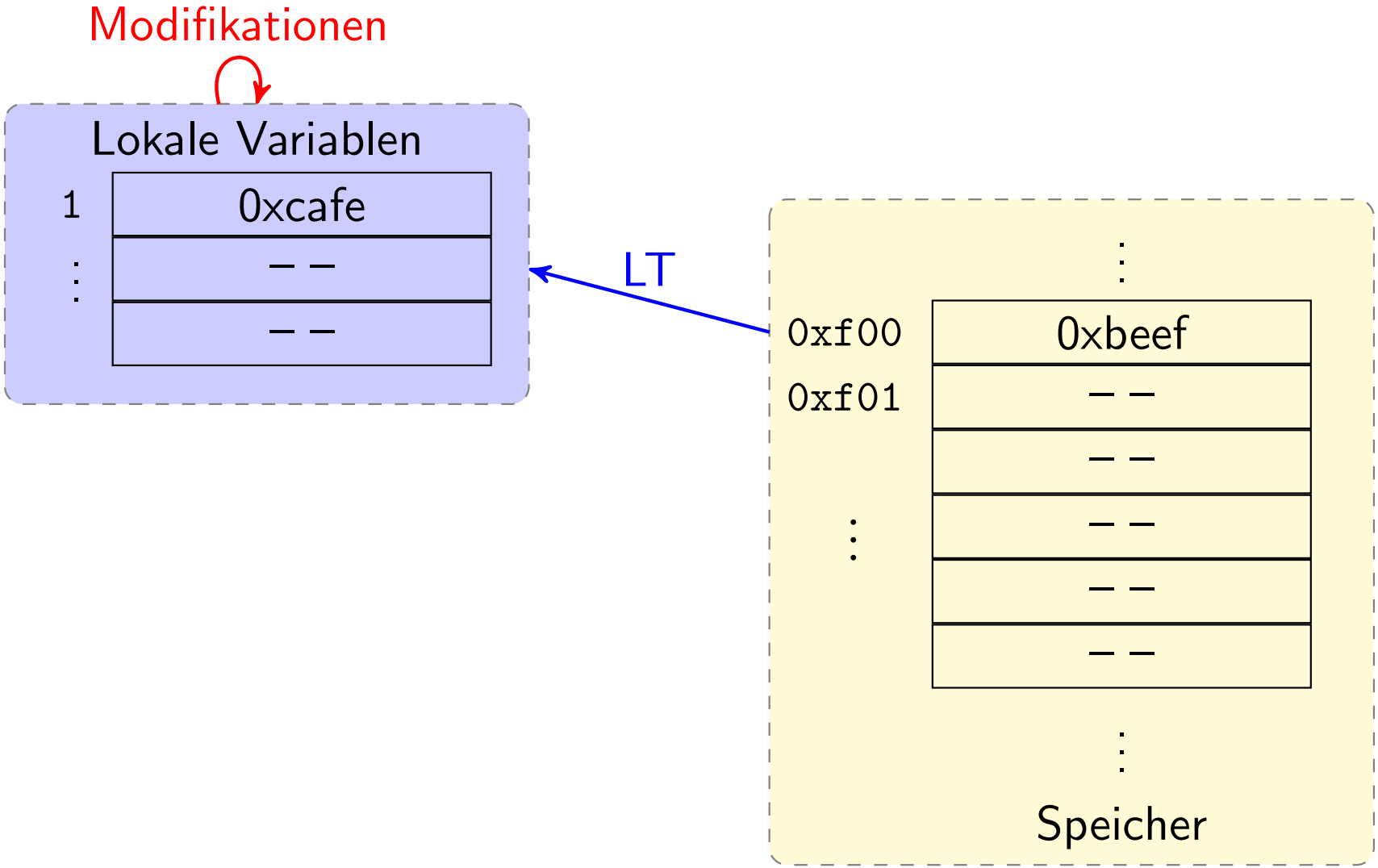
Isolation



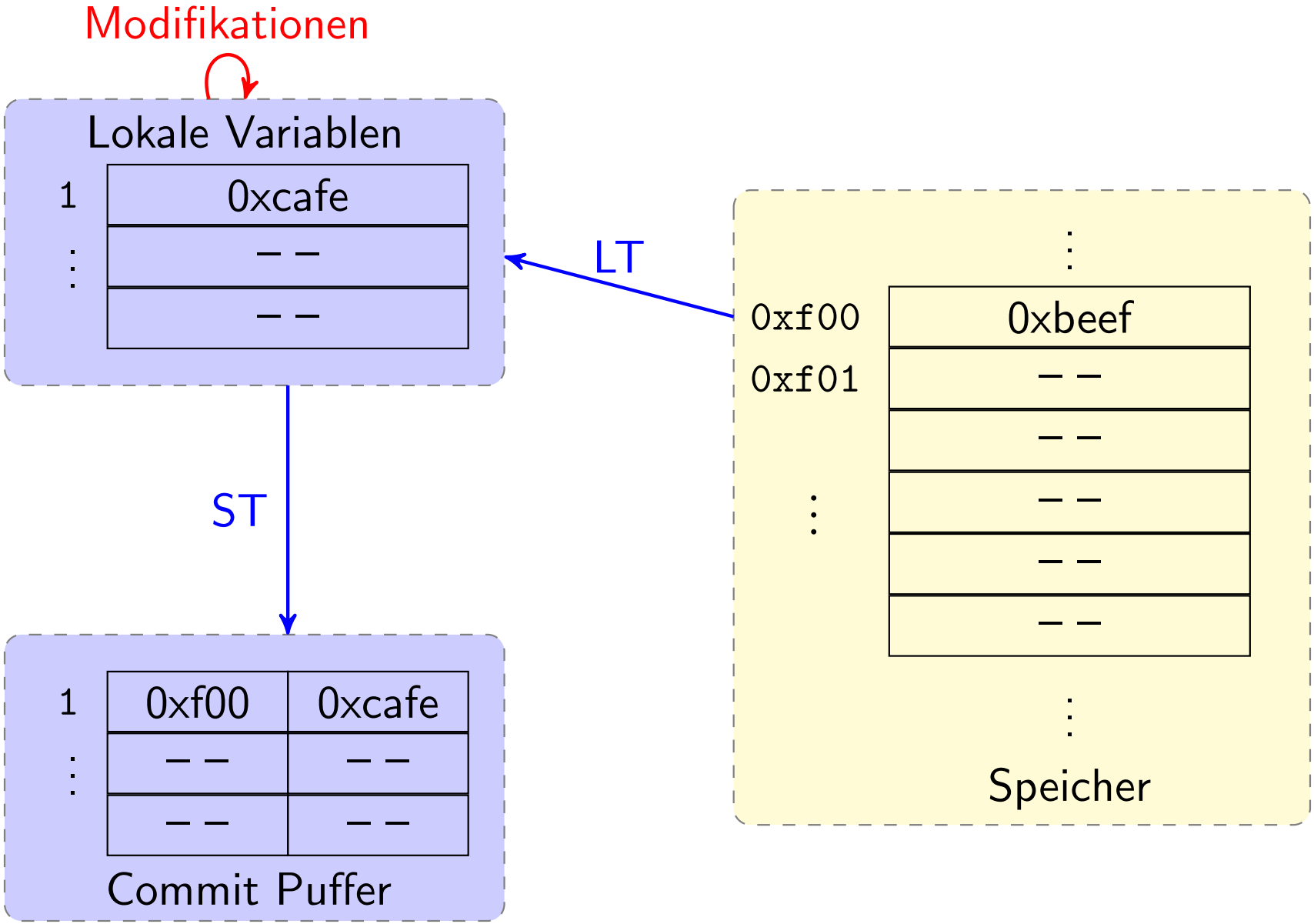
Isolation



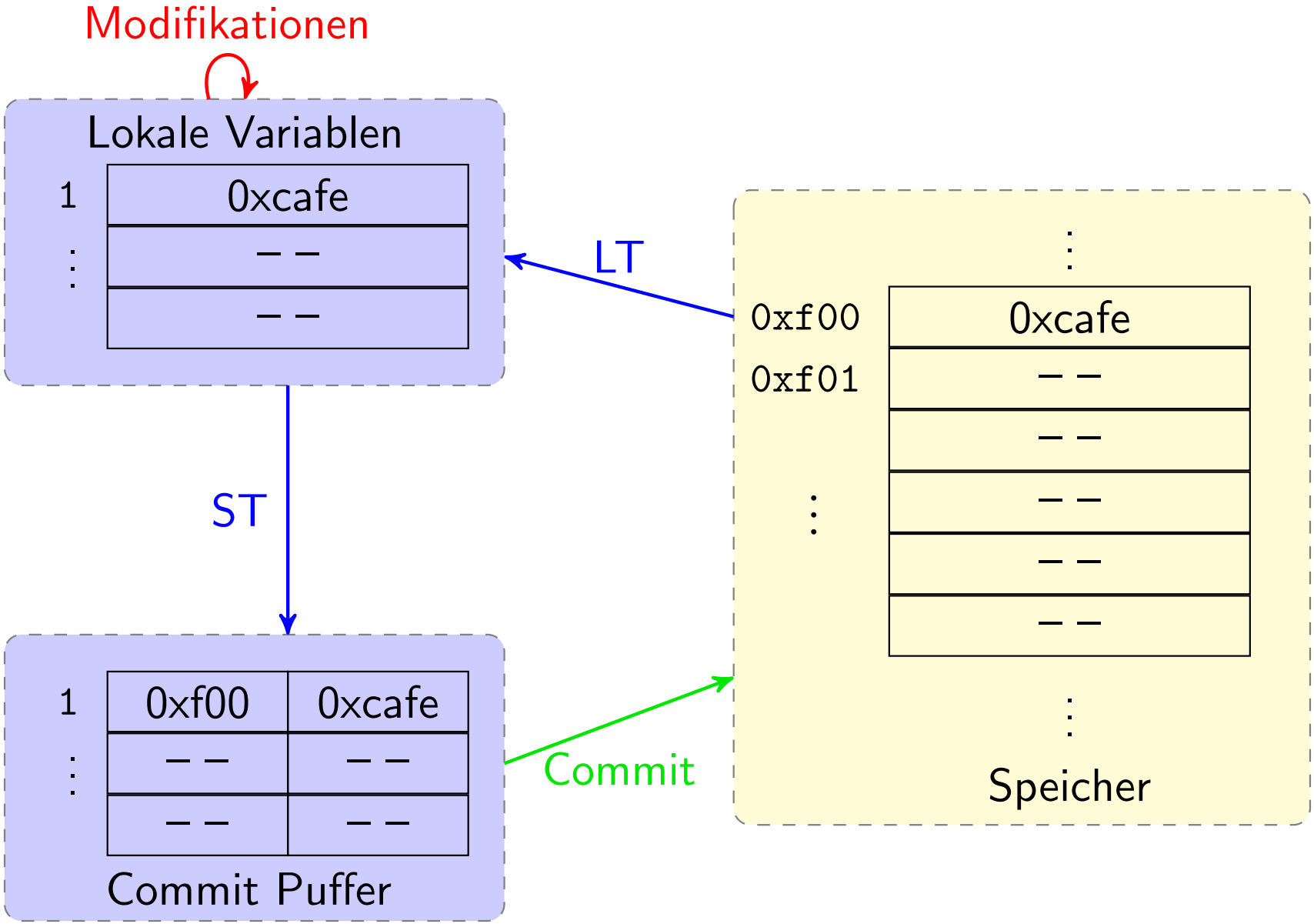
Isolation


Modifikationen
Lokale Variablen
1
0xcafe
– –
– –
LT
0xf00
0xf01
0xbeef
– –
– –
– –
– –
– –
Speicher

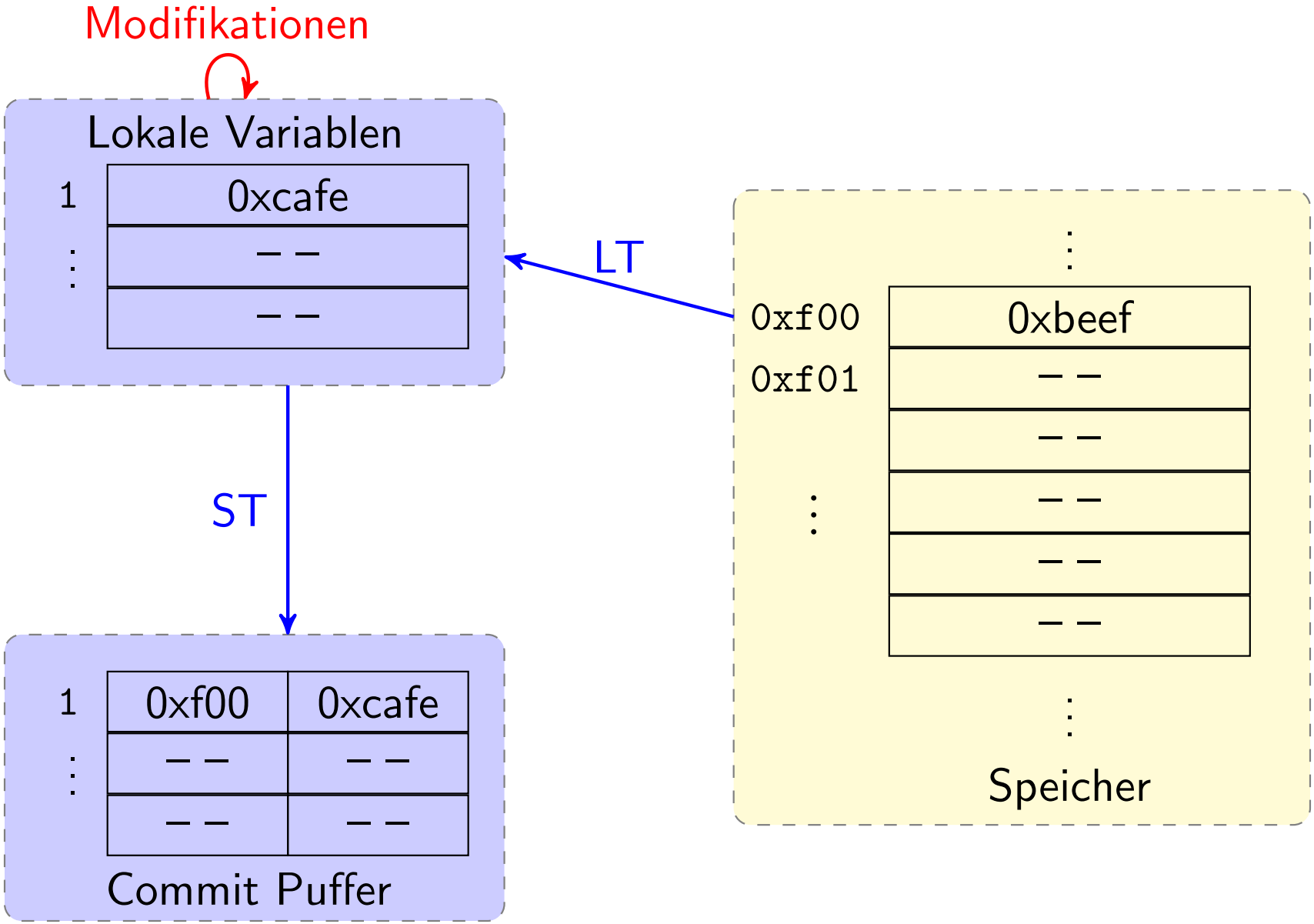
Isolation


Modifikationen
Lokale Variablen
1
0xcafe
– –
– –
LT
ST
0xf00
0xf01
0xbeef
– –
– –
– –
– –
– –
Speicher
1
0xf00
0xcafe
– –
– –
– –
– –
Commit Puffer

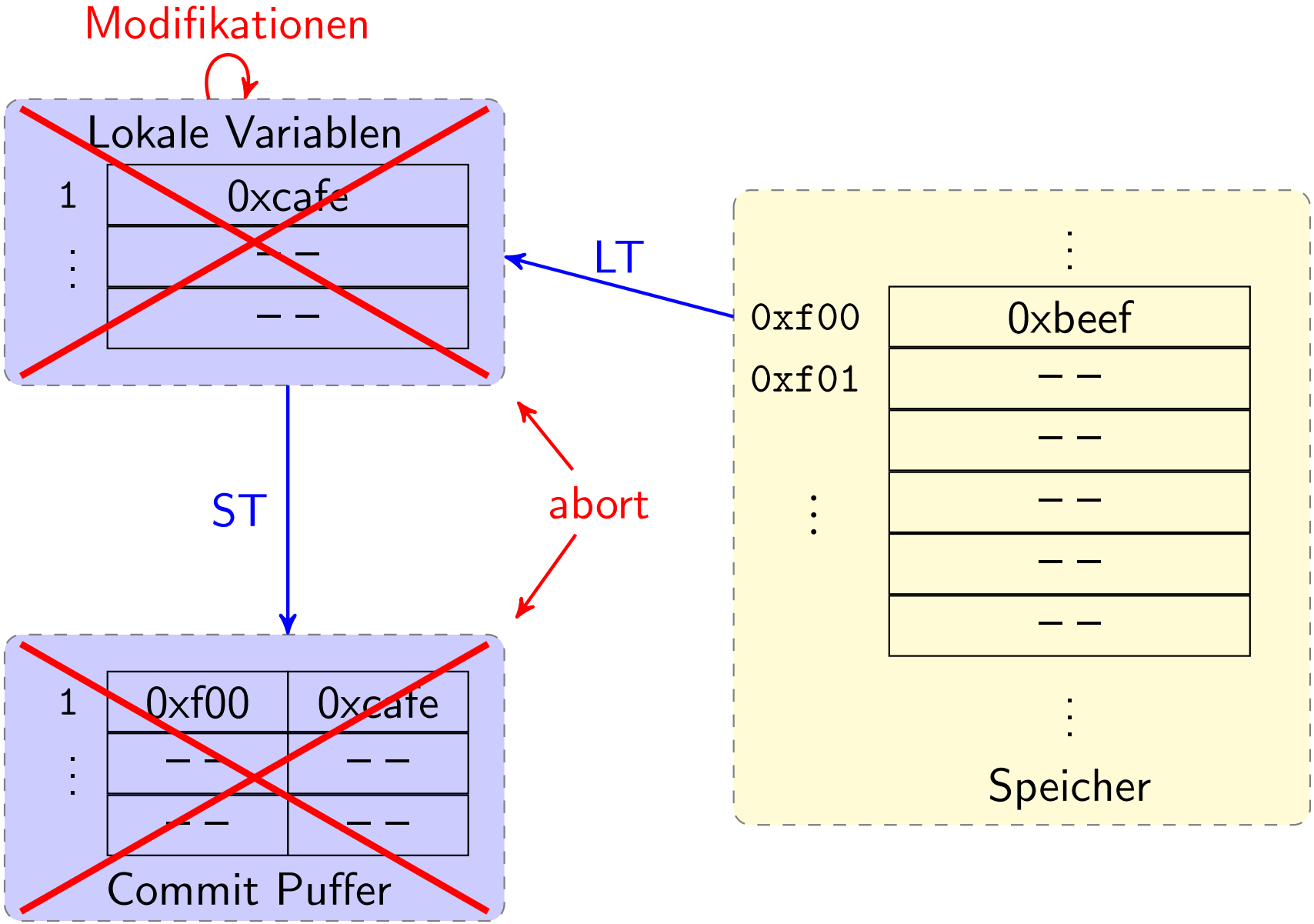
Isolation


Modifikationen
Lokale Variablen
1
0xcafe
– –
– –
LT
ST
0xf00
0xf01
0xcafe
– –
– –
– –
– –
– –
Speicher
1
0xf00
0xcafe
– –
– –
– –
– –
Commit
Commit Puffer

Isolation


Modifikationen
Lokale Variablen
1
0xcafe
– –
– –
LT
ST
0xf00
0xf01
0xbeef
– –
– –
– –
– –
– –
Speicher
1
0xf00
0xcafe
– –
– –
– –
– –
Commit Puffer

Isolation


Modifikationen
Lokale Variablen
1
0xcafe
LT
0xf00
0xbeef
0xf01
ST
abort
1
0xf00
0xcafe
Commit Puffer
Speicher

Struktur eines (TM-)Programmes

1. Daten lesen (mittels LT)
2. Berechnungen durchführen
3. Daten „schreiben“ (mittels ST)
4. Commit probieren
5. **Fehlerfall**:
    1. Abort
    2. Erneuter Versuch
        $\rightarrow$ 1

Konflikte

Lesemenge (Read-Set)

Alle im Transaktionskörper gelesenen Daten

Schreibmenge (Write-Set)

Alle im Transaktionskörper geschriebenen Daten

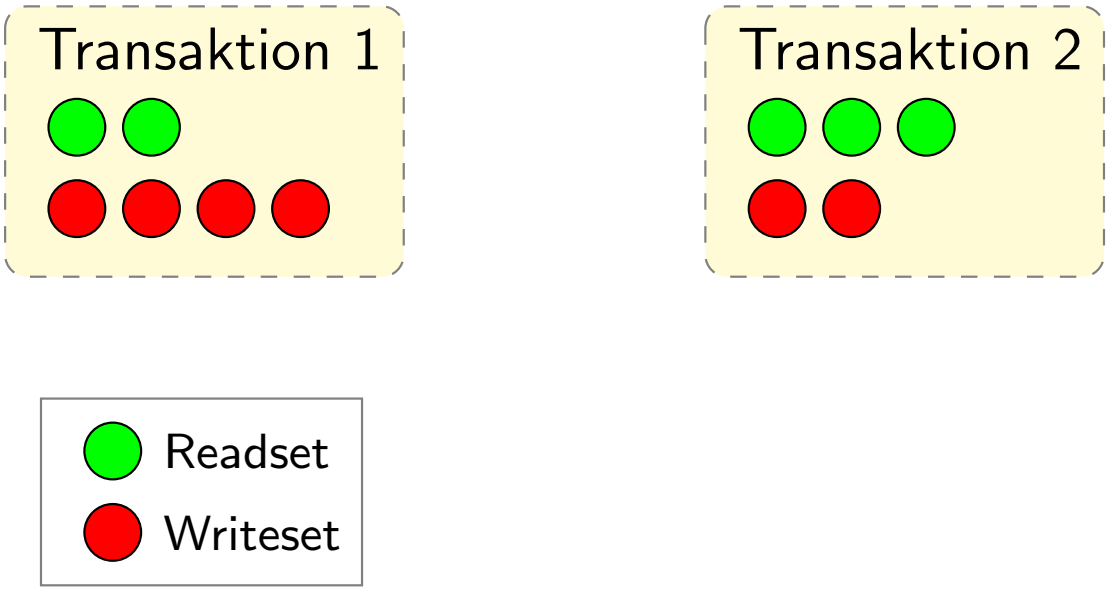
Datenmenge (Data-Set)

Lesemenge $\cup$ Schreibmenge

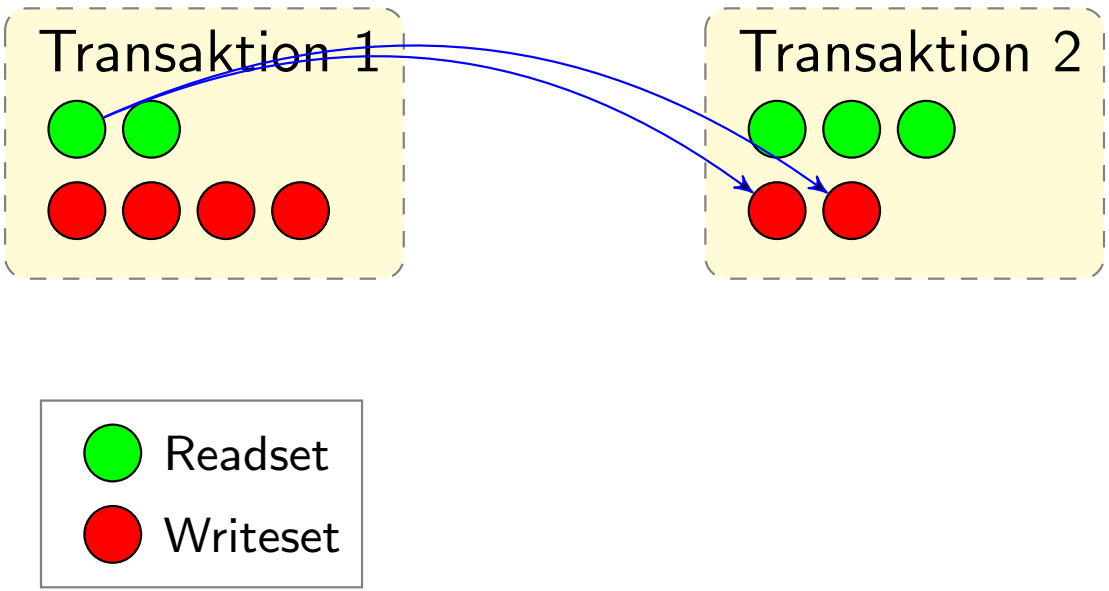
Konflikt mit anderen bereits eingetretenden Transaktionen beim Commit:

- Elemente der Datenmenge wurden (global) verändert
- Elemente der Schreibmenge wurden (global) gelesen

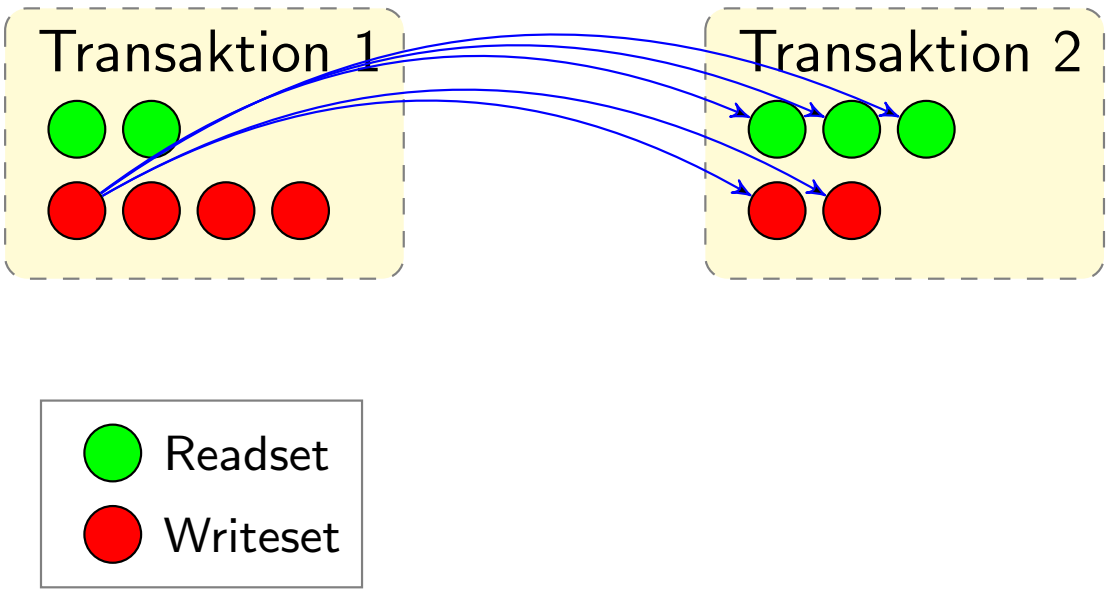
Konflikterkennung



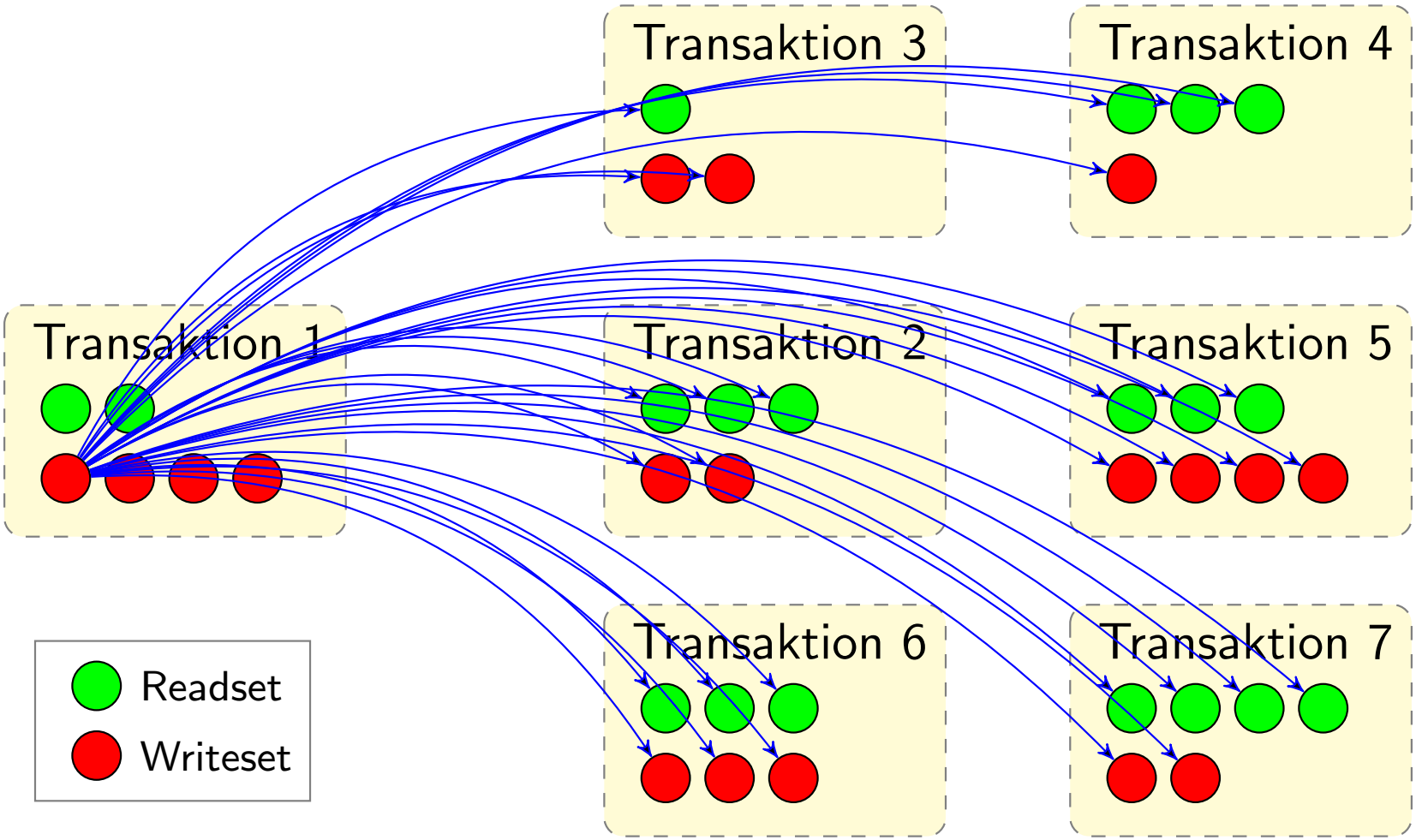
Konflikterkennung



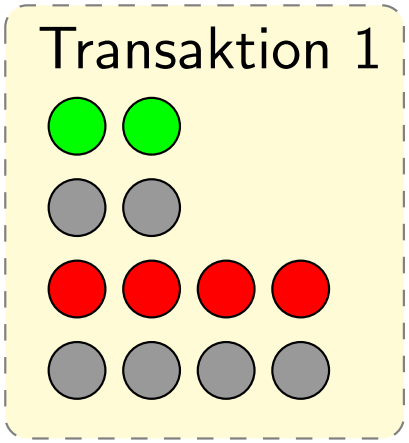
Konflikterkennung

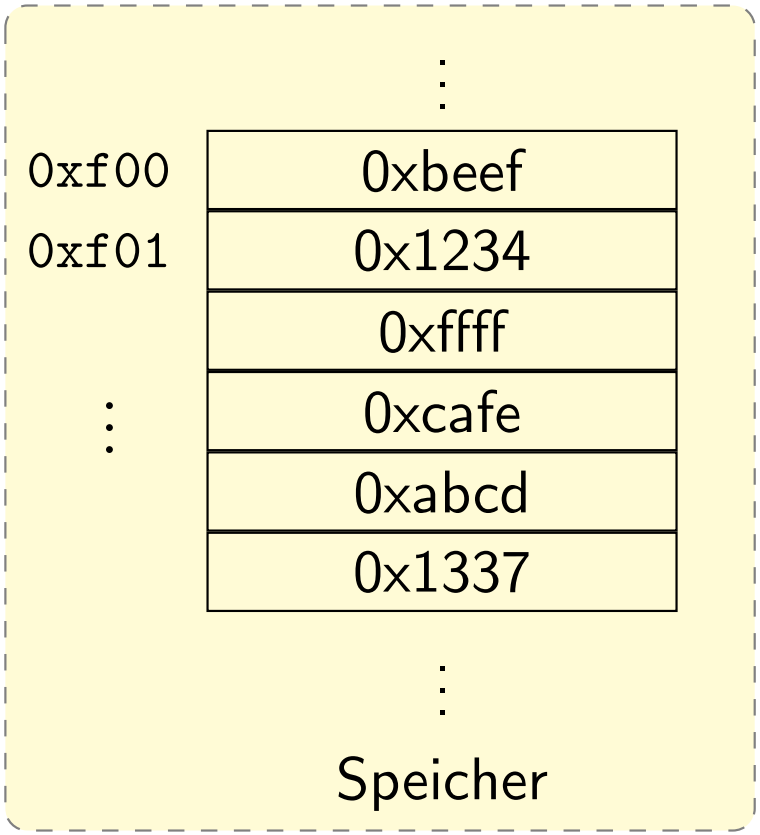


Konflikterkennung

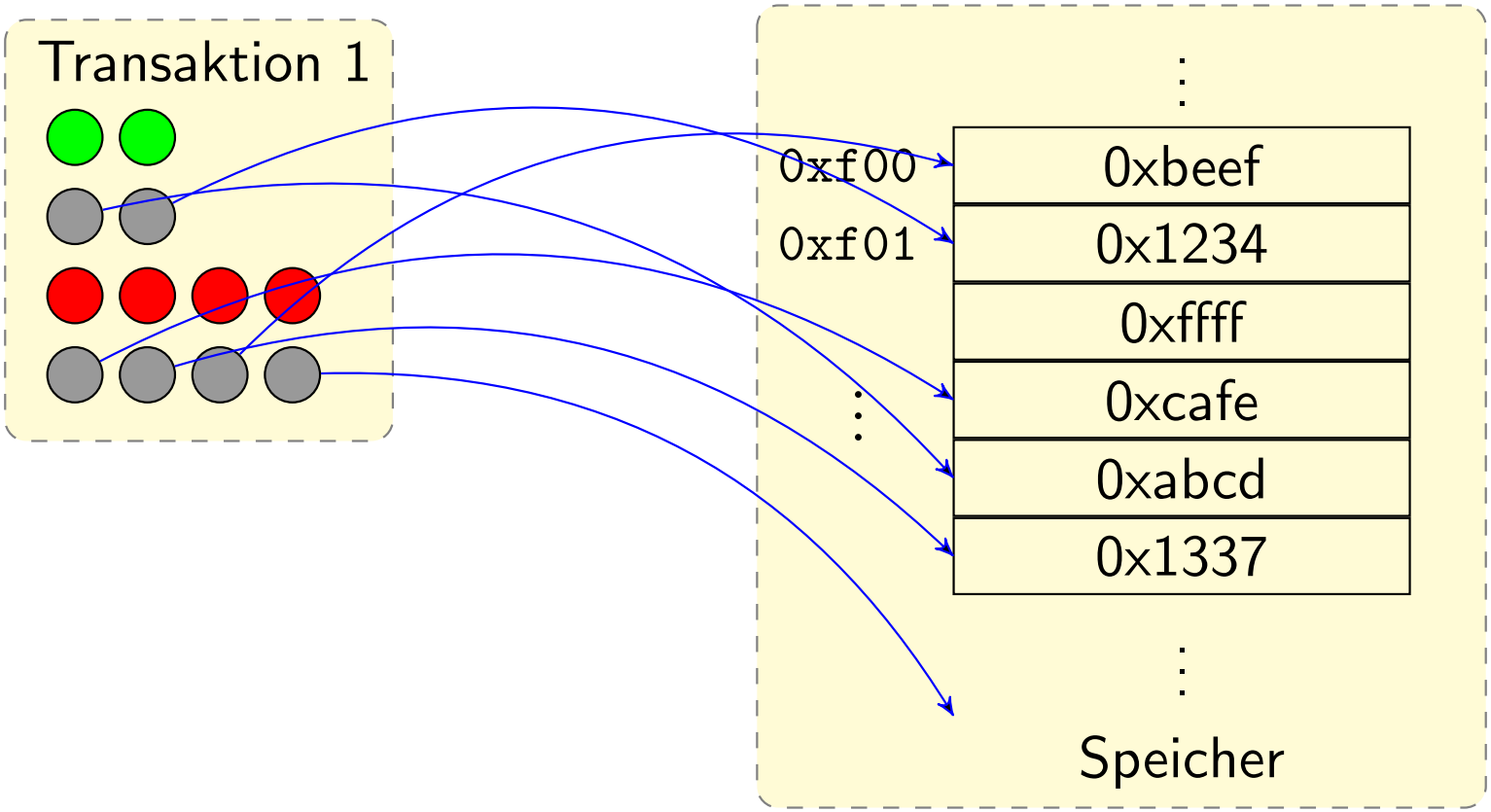


Konflikterkennung

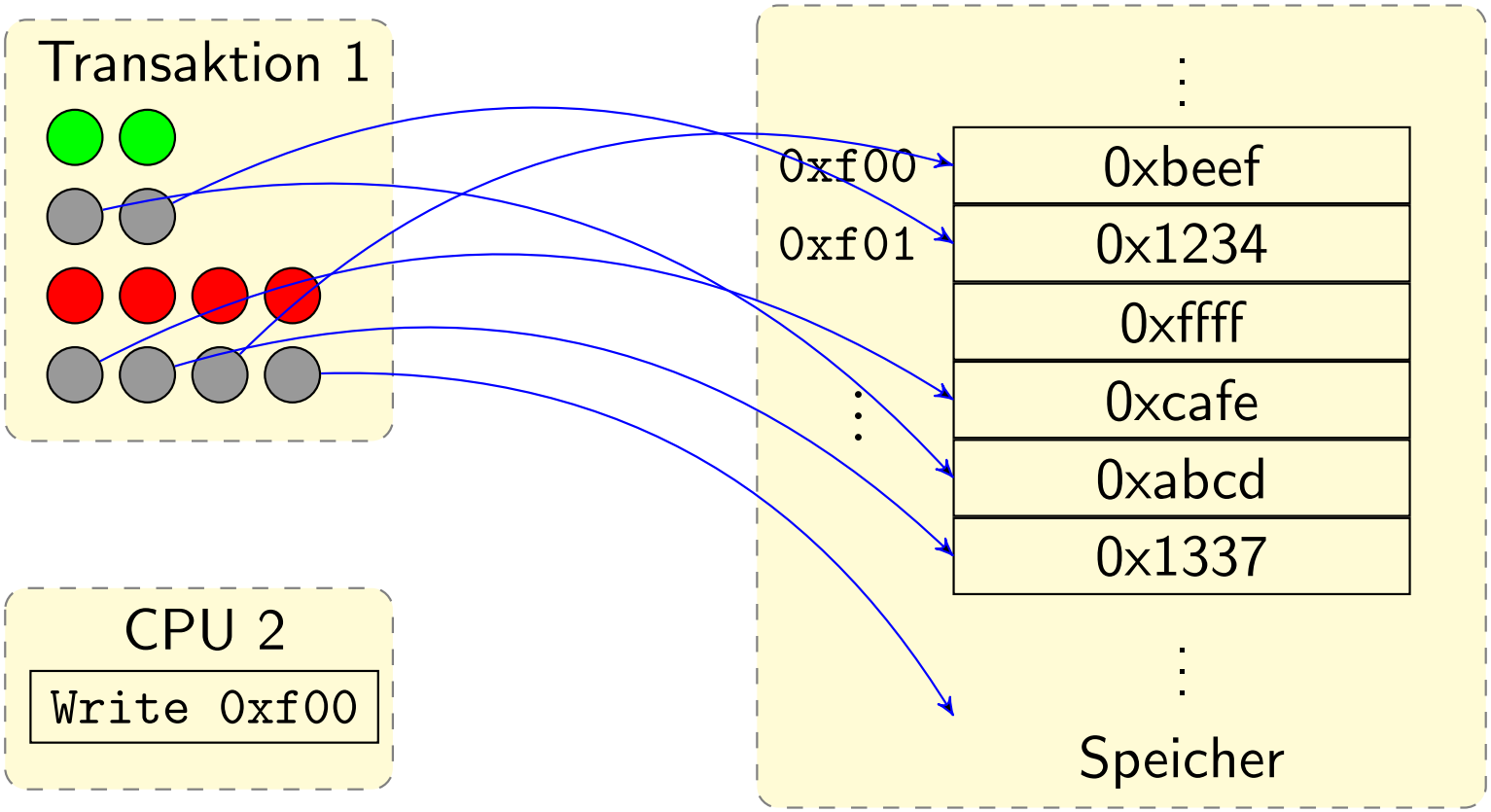




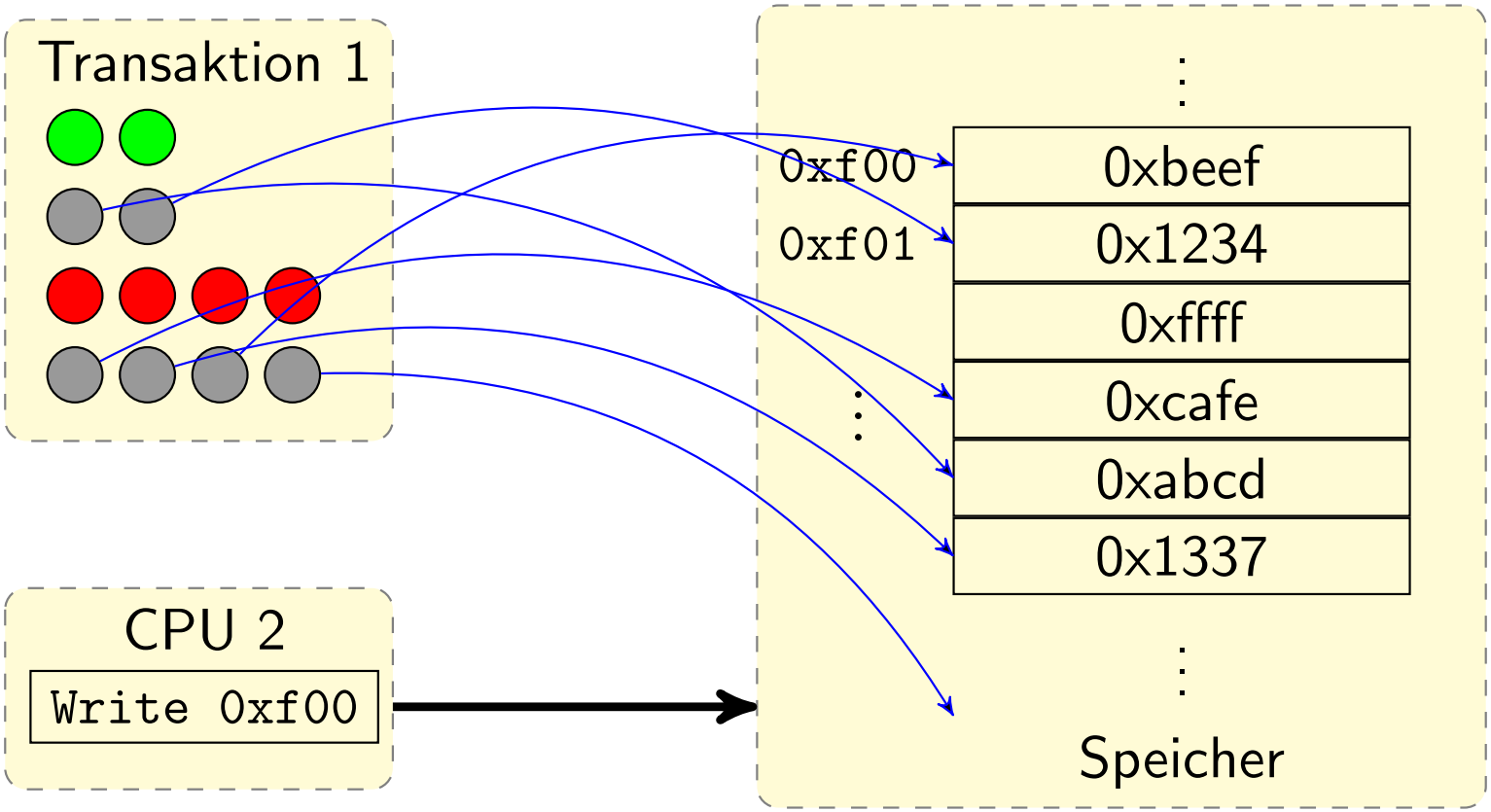
Konflikterkennung



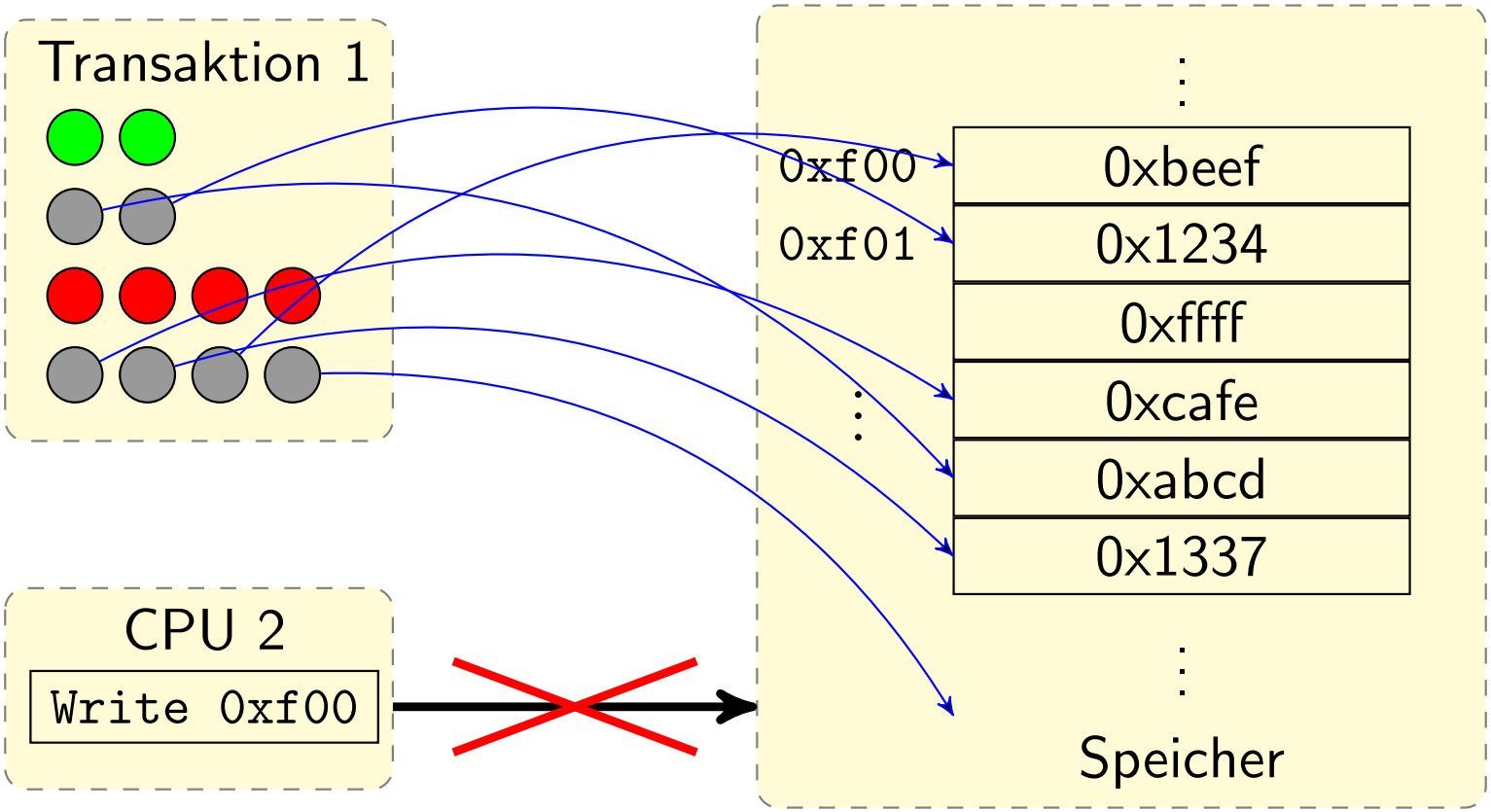
Konflikterkennung



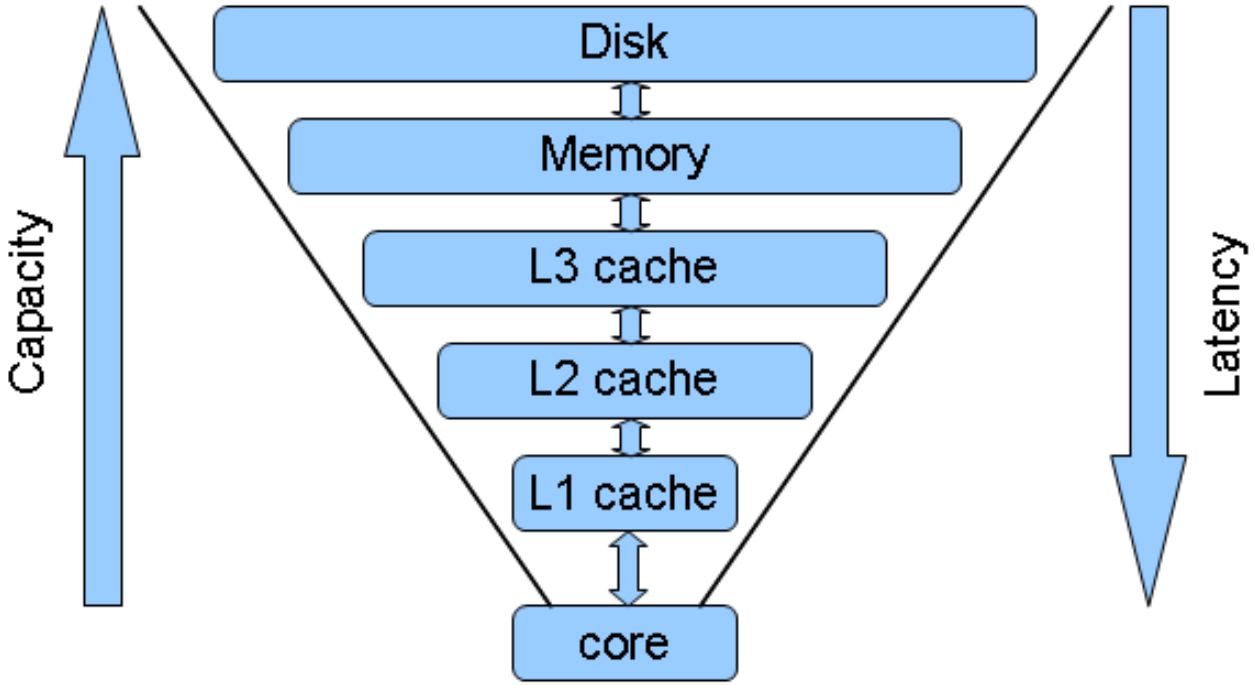
Konflikterkennung



Konflikterkennung



Cache



Cache

- Schneller Pufferspeicher in Prozessornähe
- Speichert den Inhalt kürzlich verwendeter Speicherzellen
- Zugriffszeit deutlich schneller als auf den Hauptspeicher

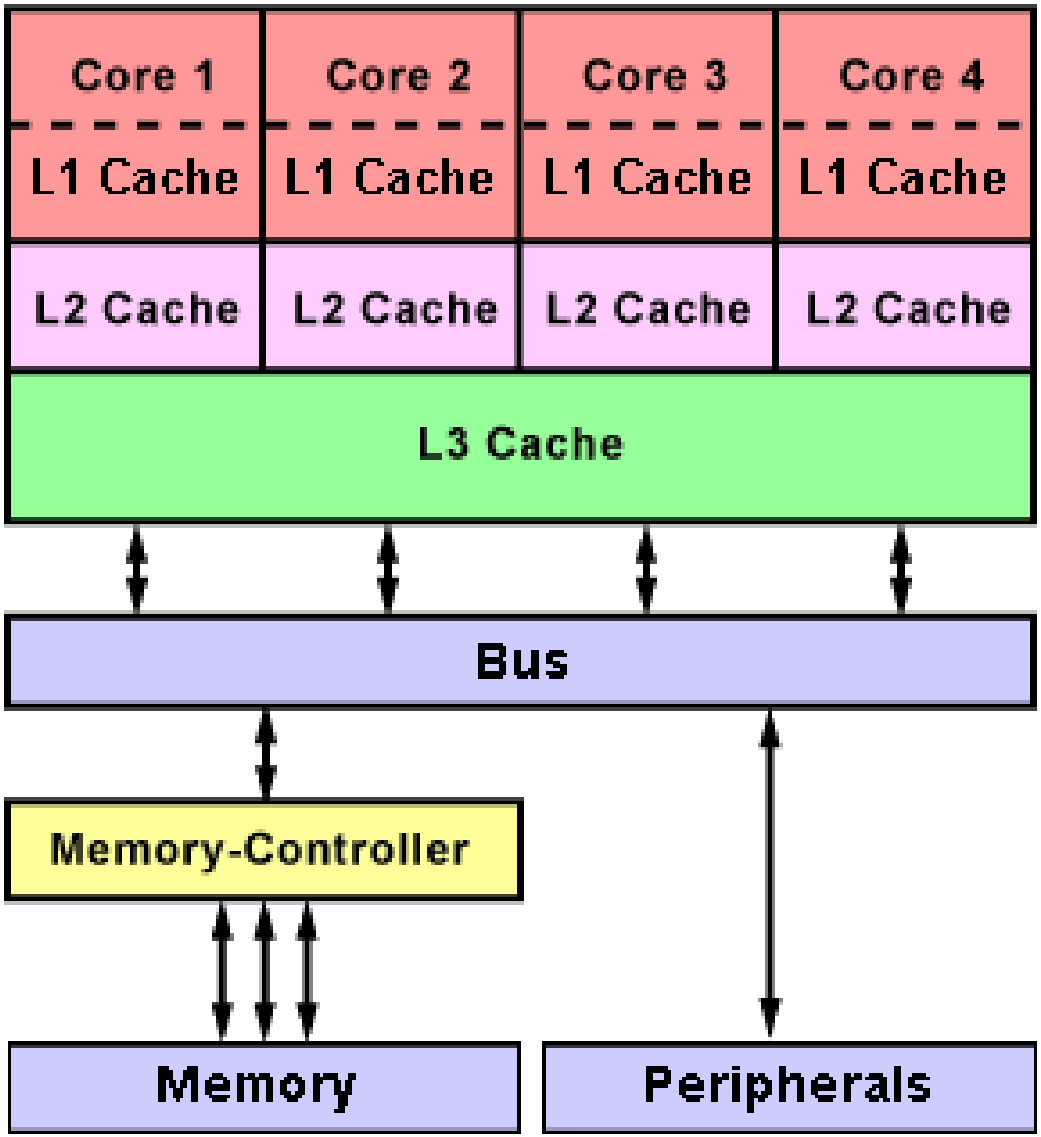
Cache-Aufbau

	Adresse	Speicherinhalt	Zustand
0	0x035	6	Dirty
1	0x634	23	Invalid
2	0x789	64	Valid

⋮

Zustand	Gültig	Verändert
Invalid		
Valid	✓	
Dirty	✓	✓

Cache-Kohärenz



Cacheeigentum

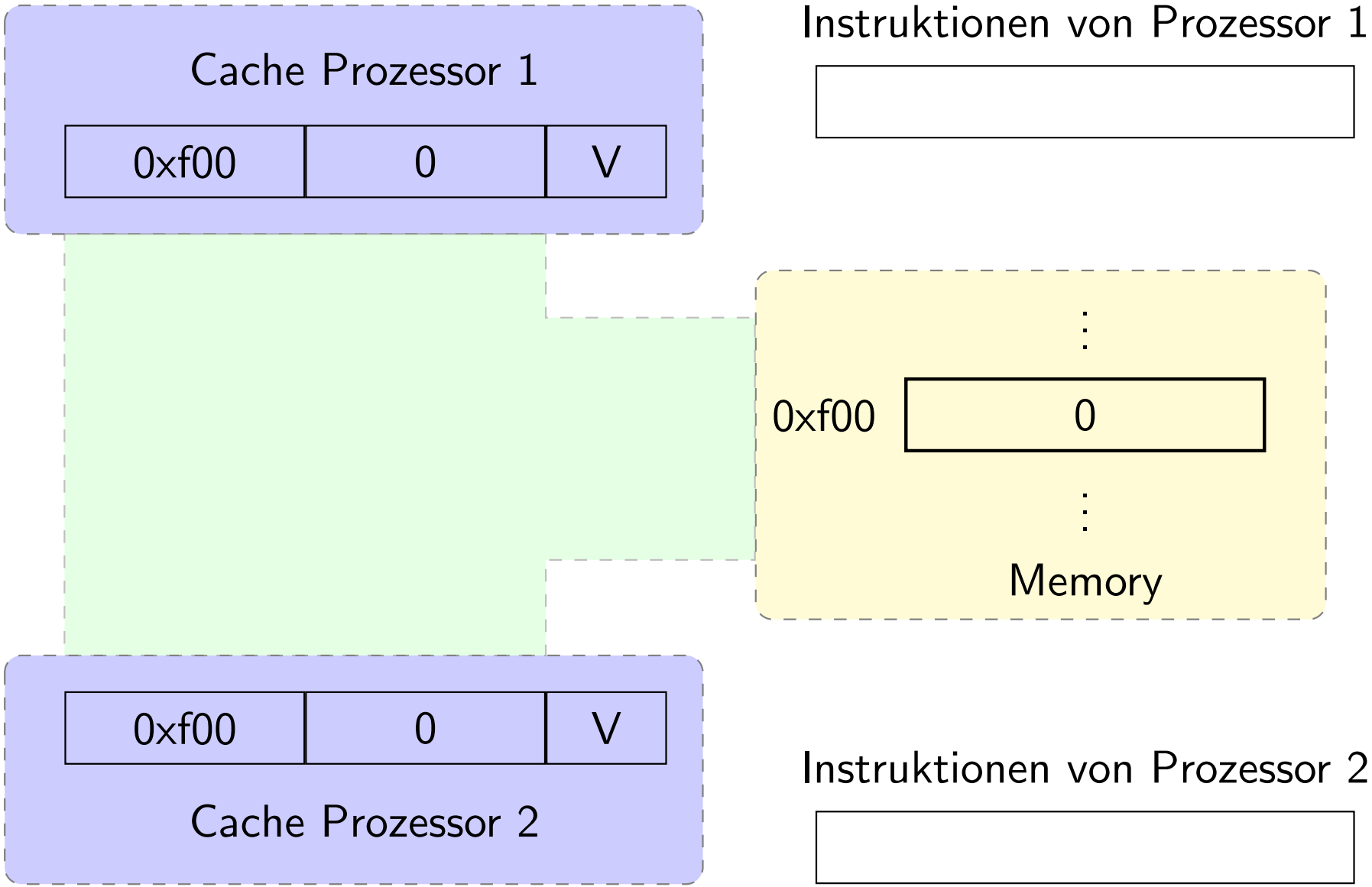
Zustand	Zugriff	Exklusiv	Unverändert
Invalid	–	–	–
Valid	R	✗	✓
Reserved	R,W	✓	✓
Dirty	R,W	✓	✗

Tabelle: Zustände

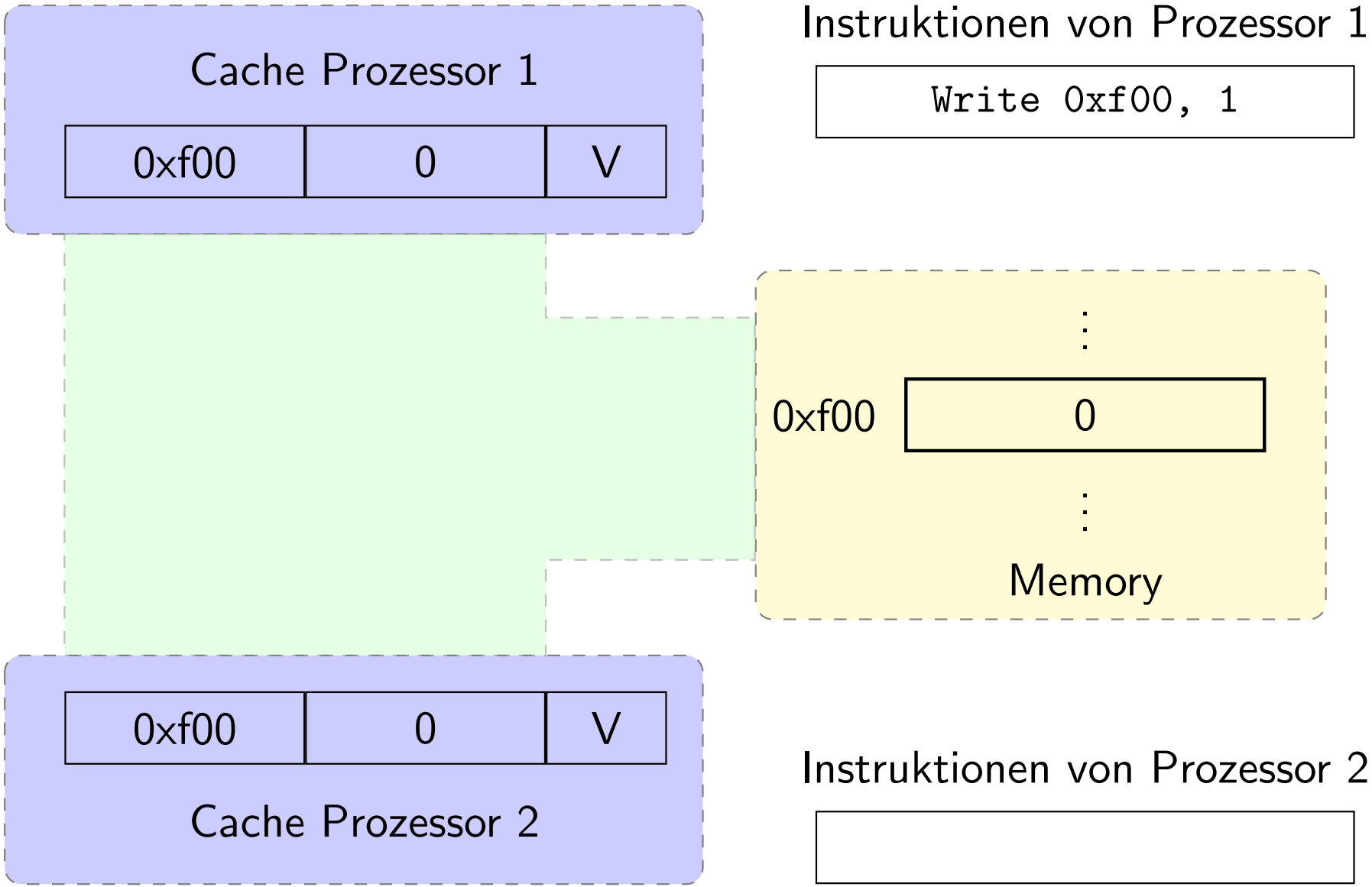
Operation	Beschreibung	Neuer Zustand
L	Wert Lesen	Valid
LX	Wert Lesen	Reserved
Write	Wert Schreiben	Dirty

Tabelle: Buszyklen

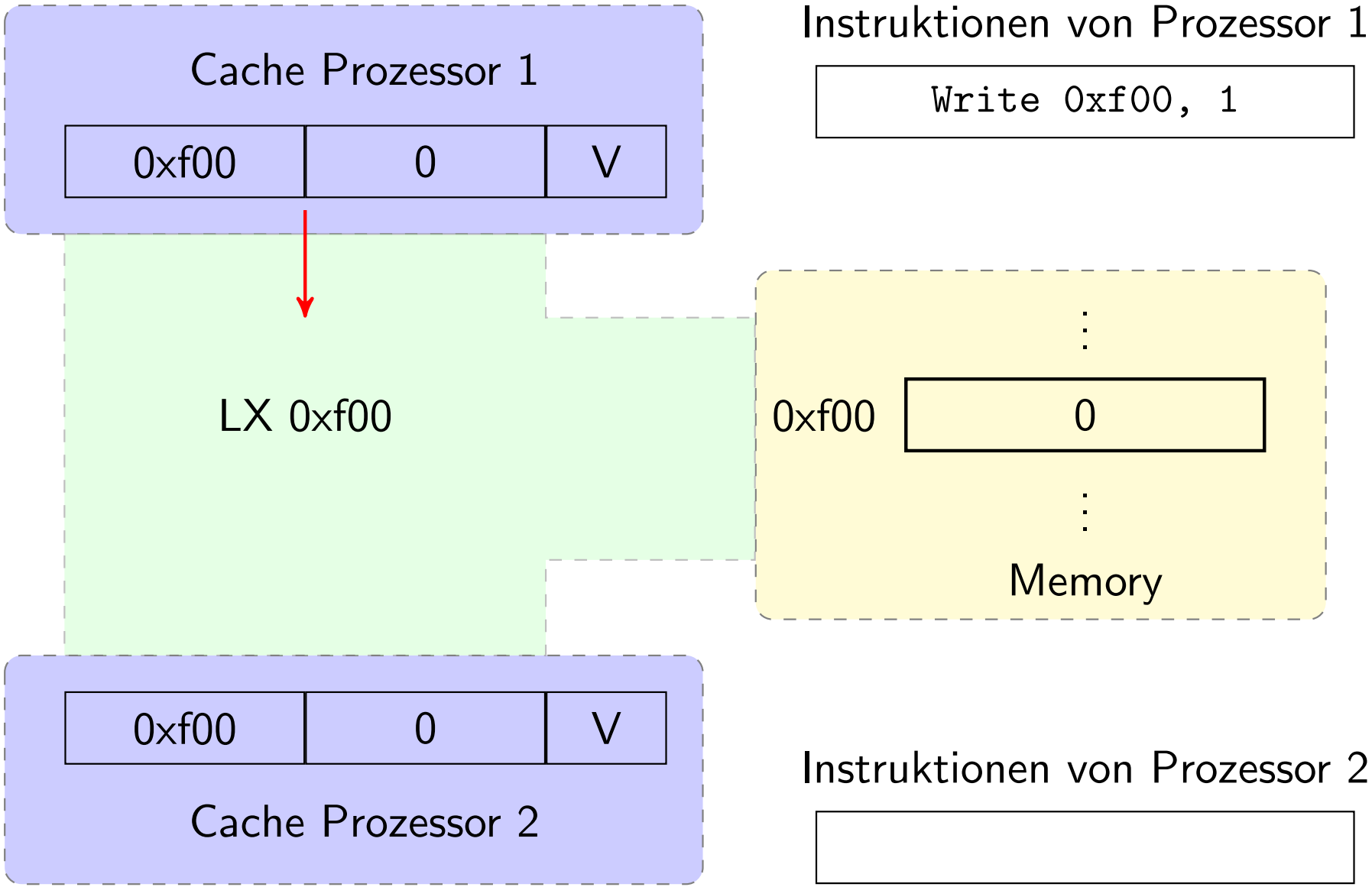
Cachekohärenzkonflikte: Fall 1



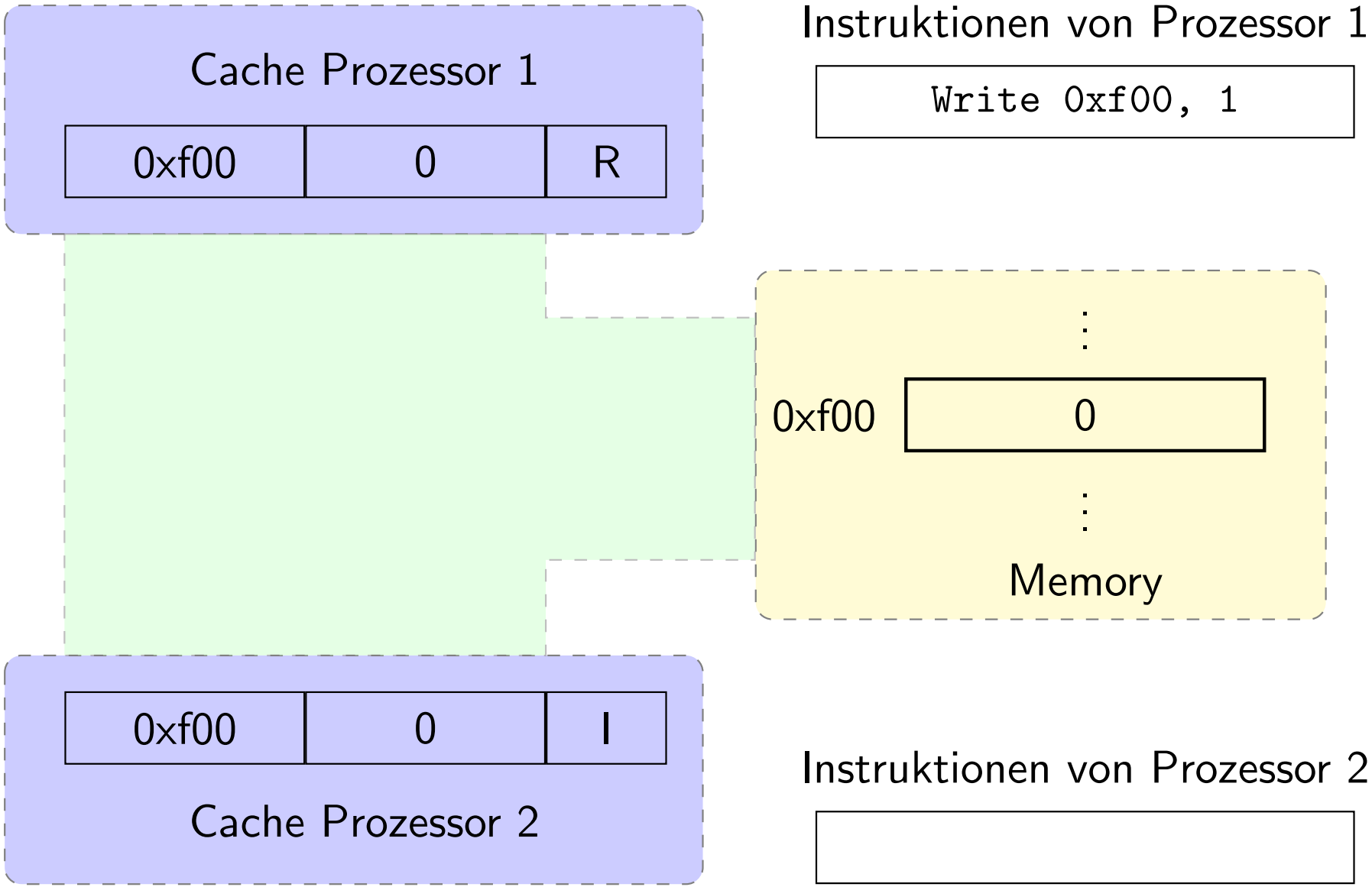
Cachekohärenzkonflikte: Fall 1



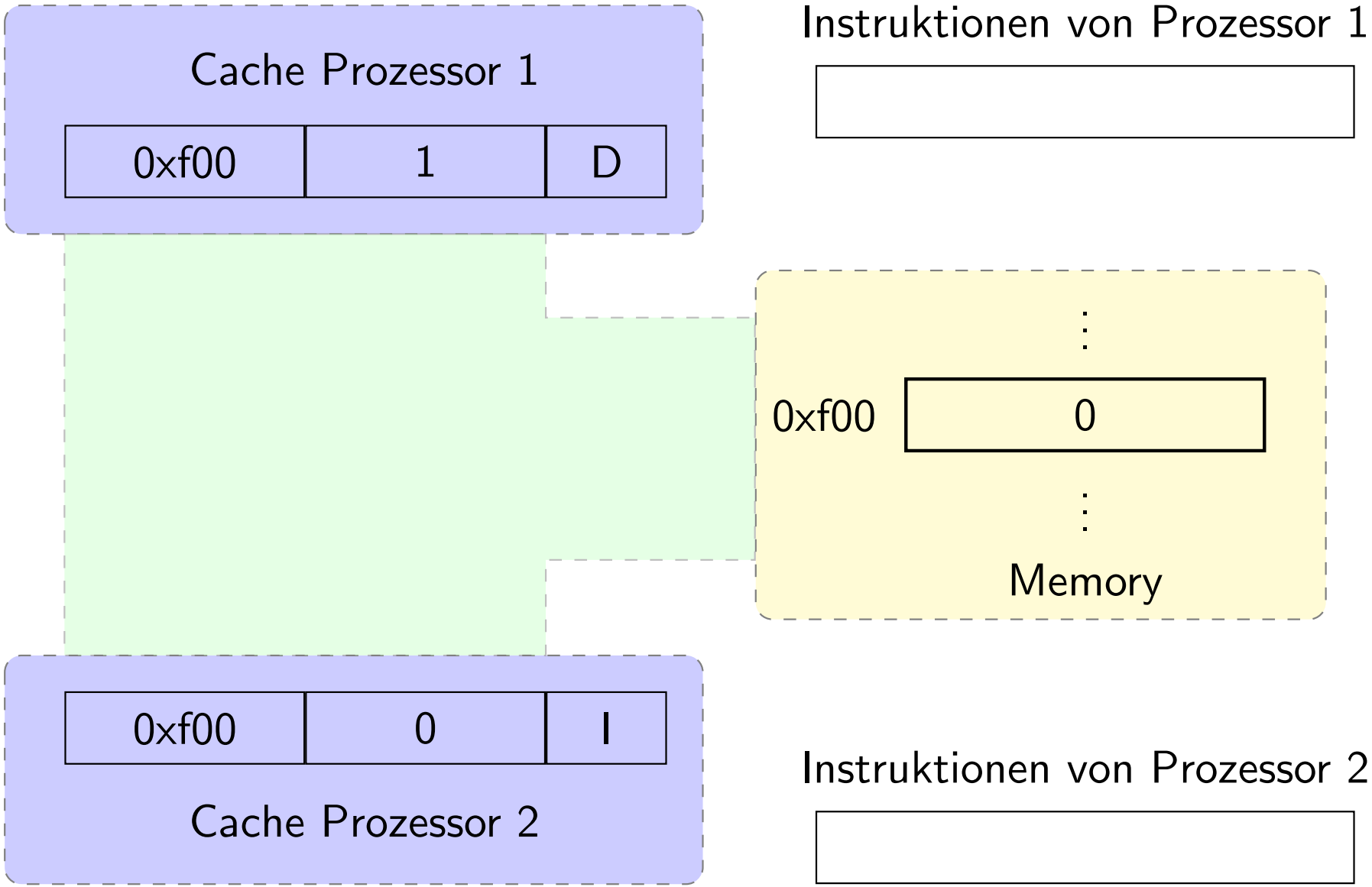
Cachekohärenzkonflikte: Fall 1



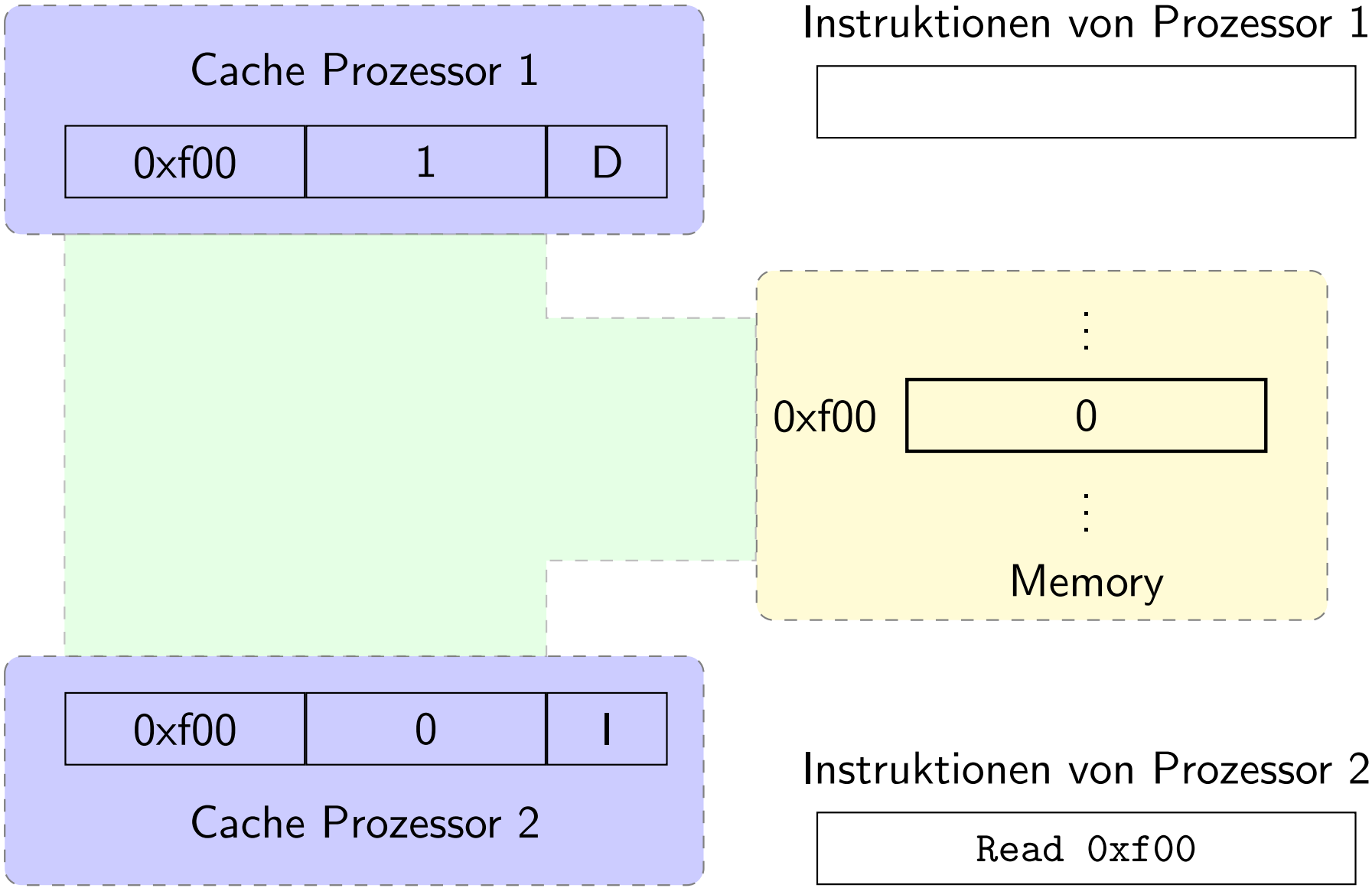
Cachekohärenzkonflikte: Fall 1



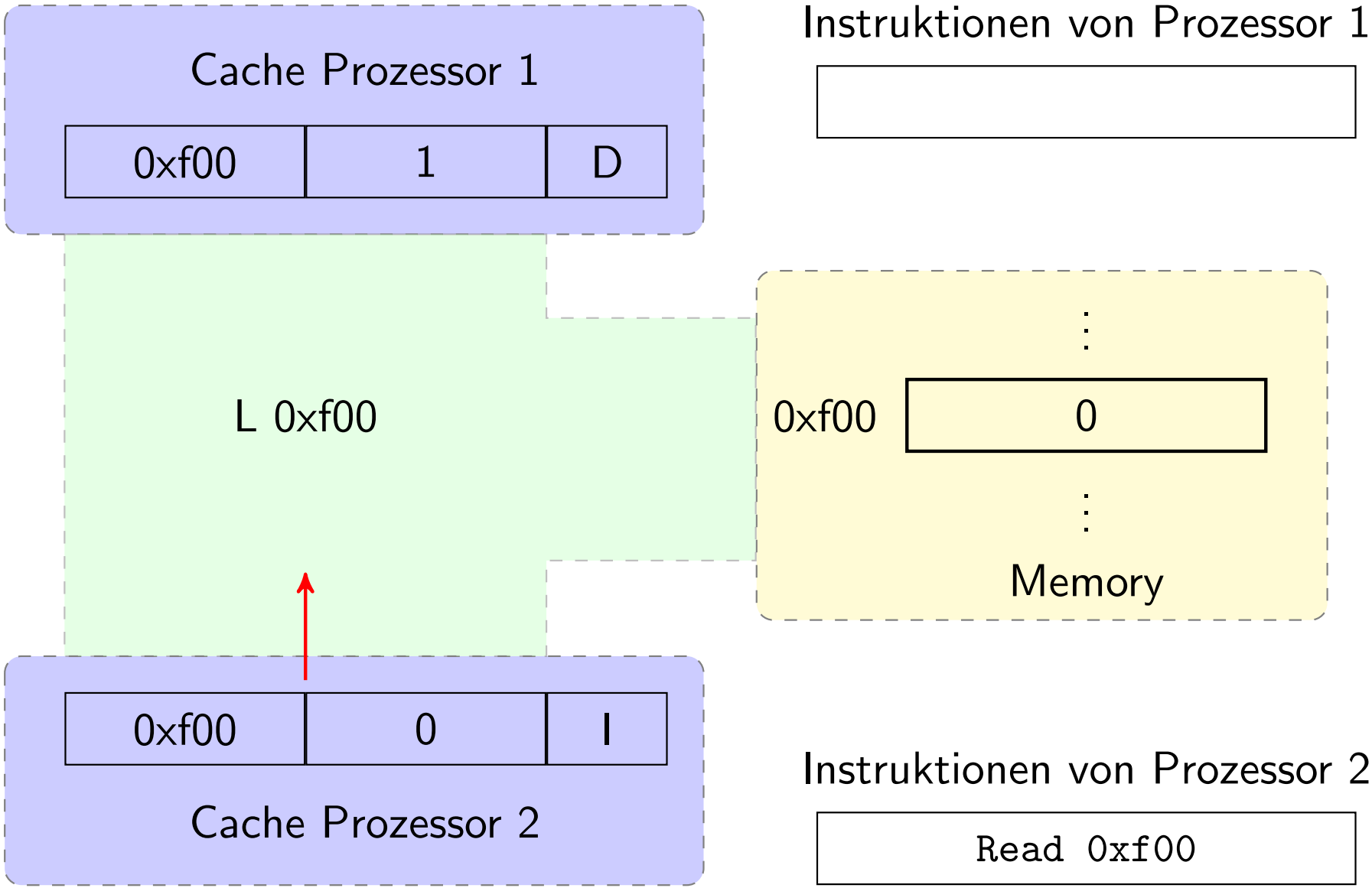
Cachekohärenzkonflikte: Fall 1



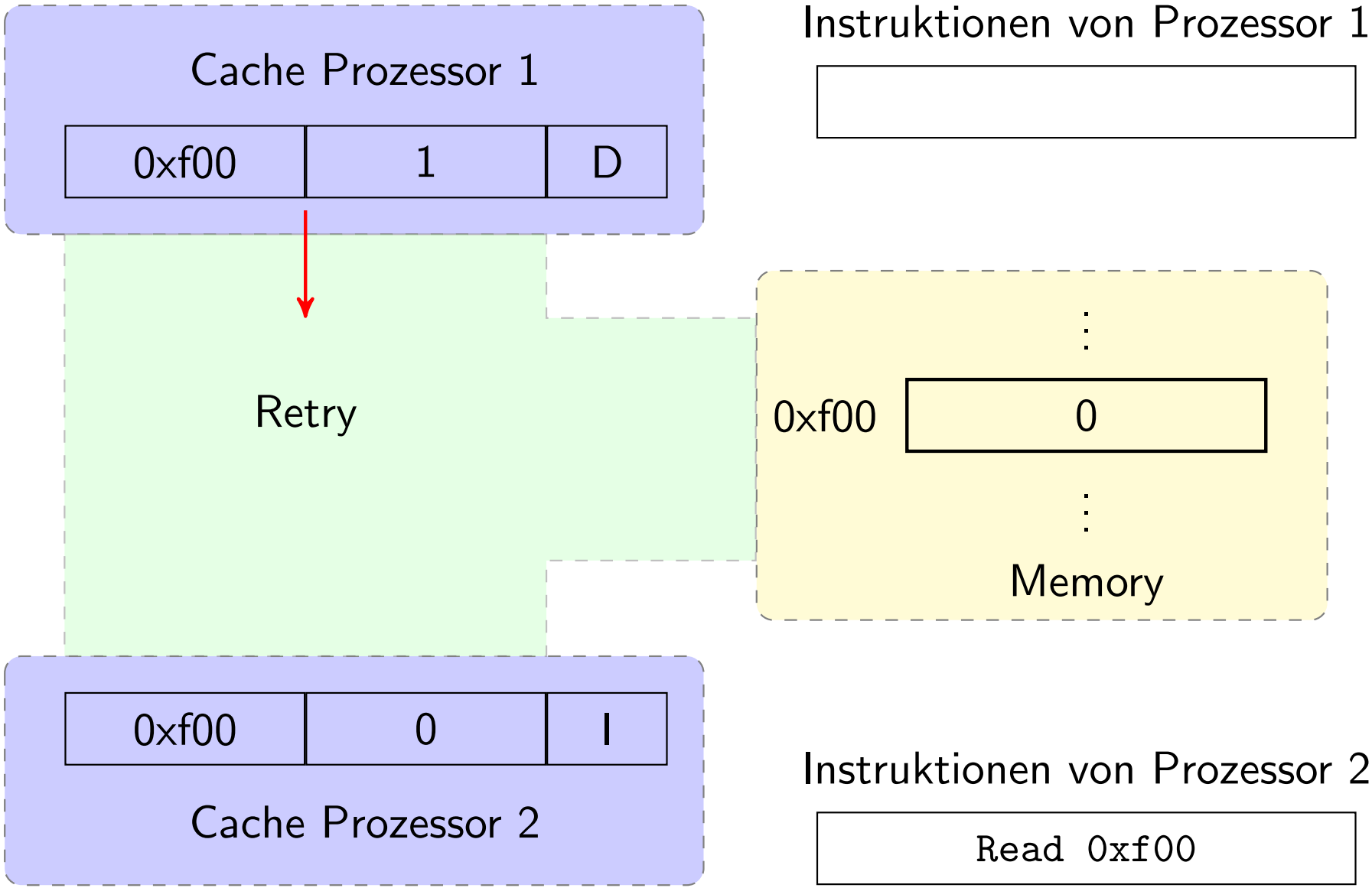
Cachekohärenzkonflikte: Fall 2



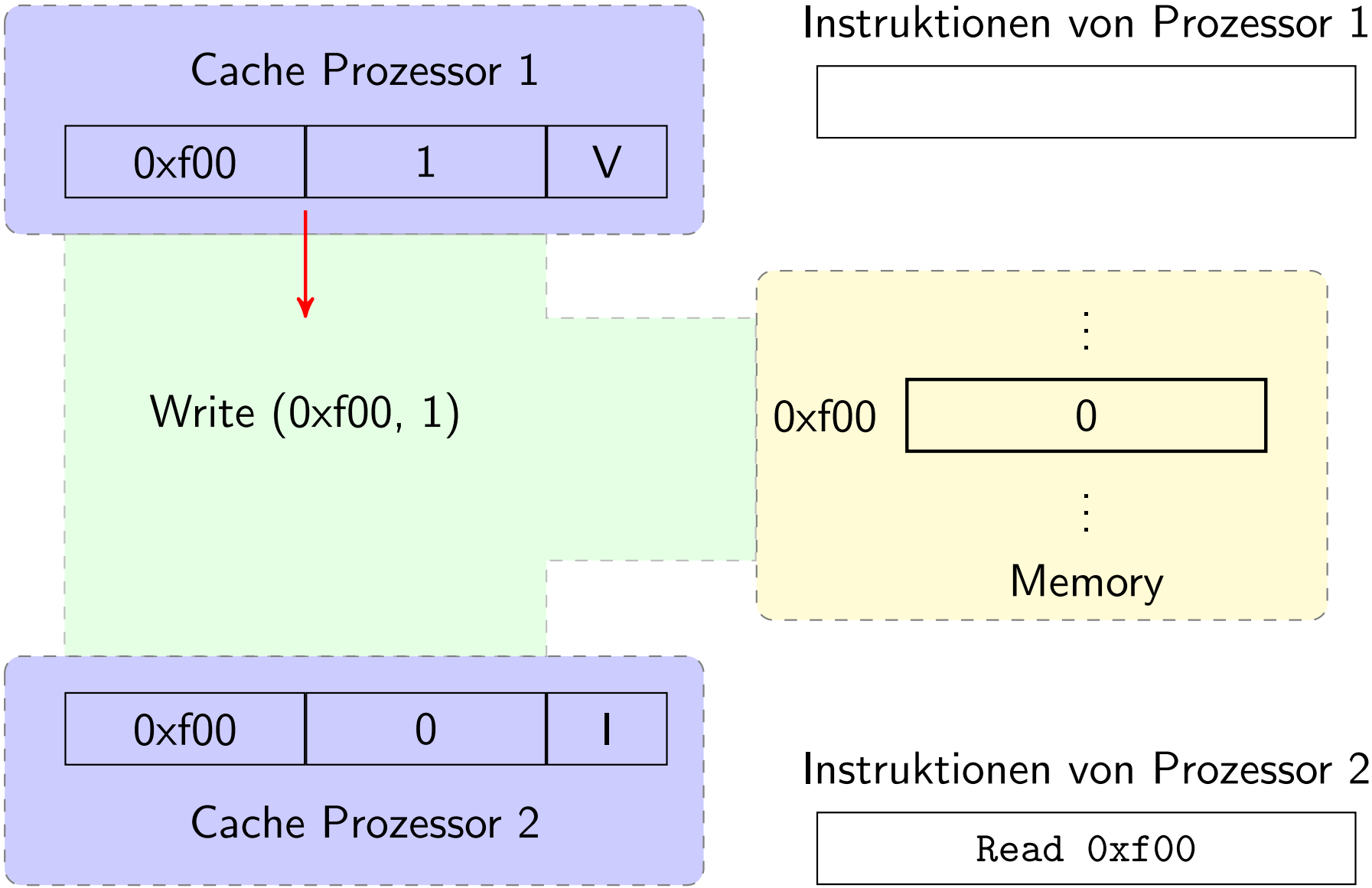
Cachekohärenzkonflikte: Fall 2



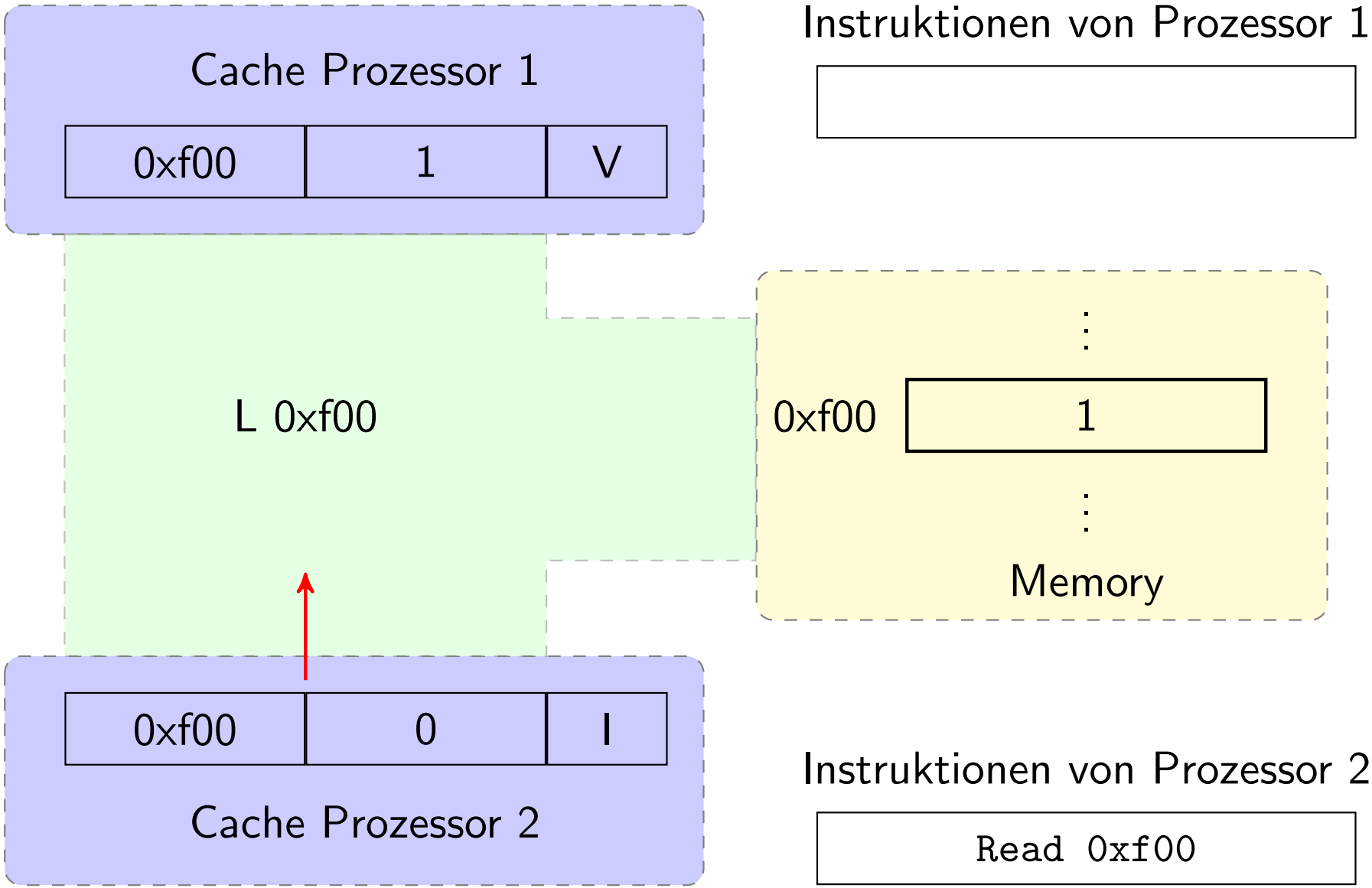
Cachekohärenzkonflikte: Fall 2



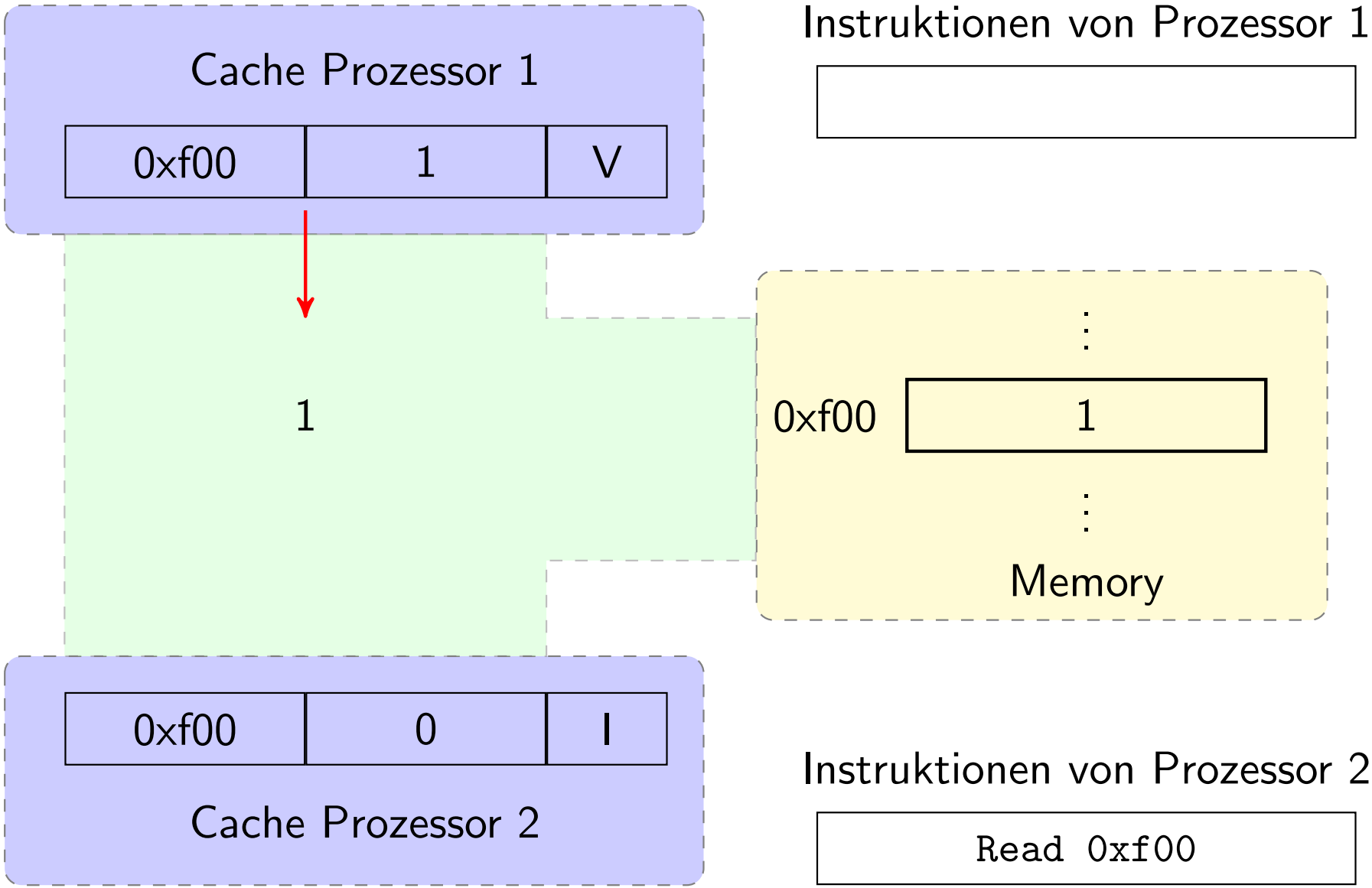
Cachekohärenzkonflikte: Fall 2



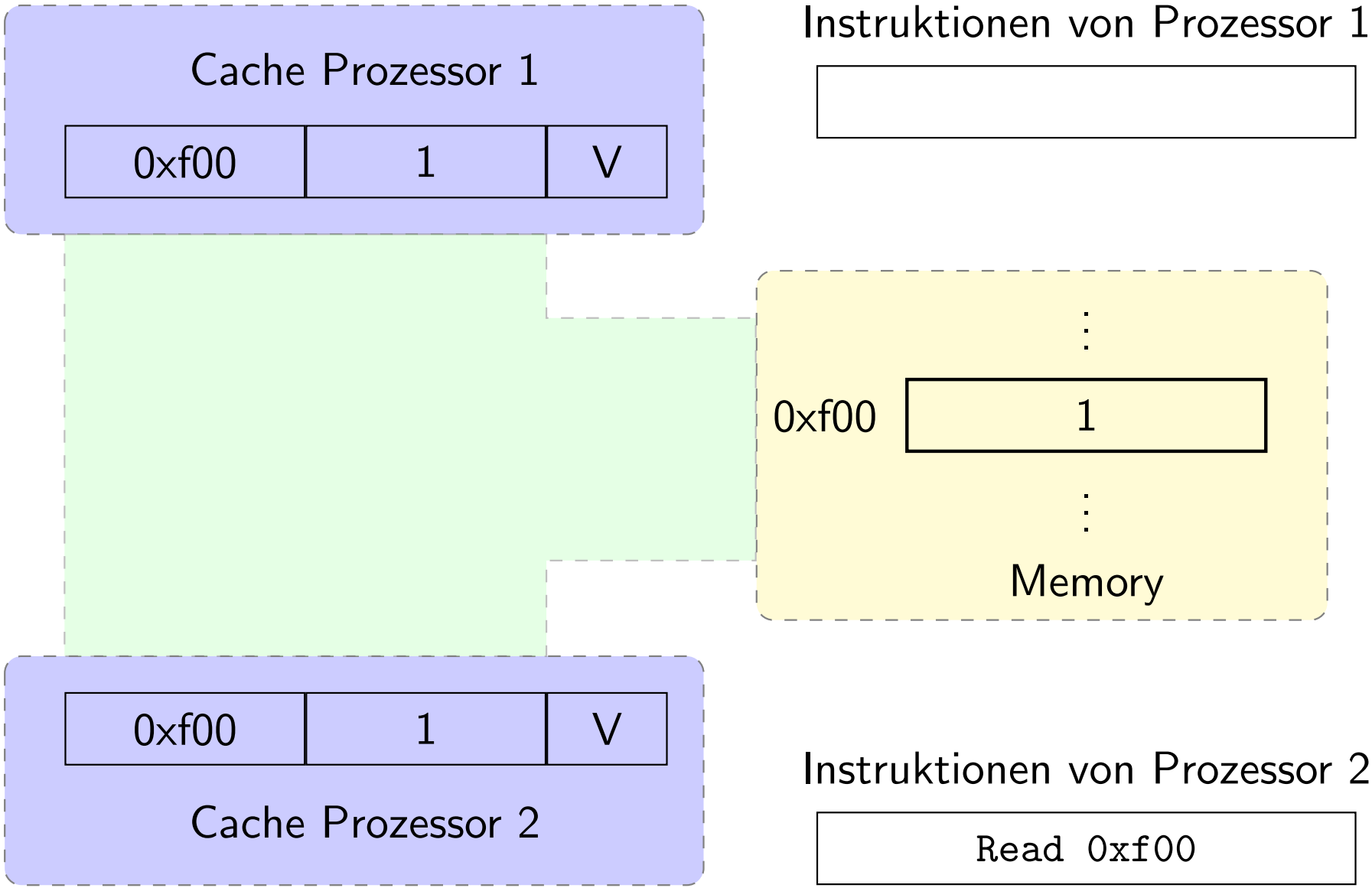
Cachekohärenzkonflikte: Fall 2



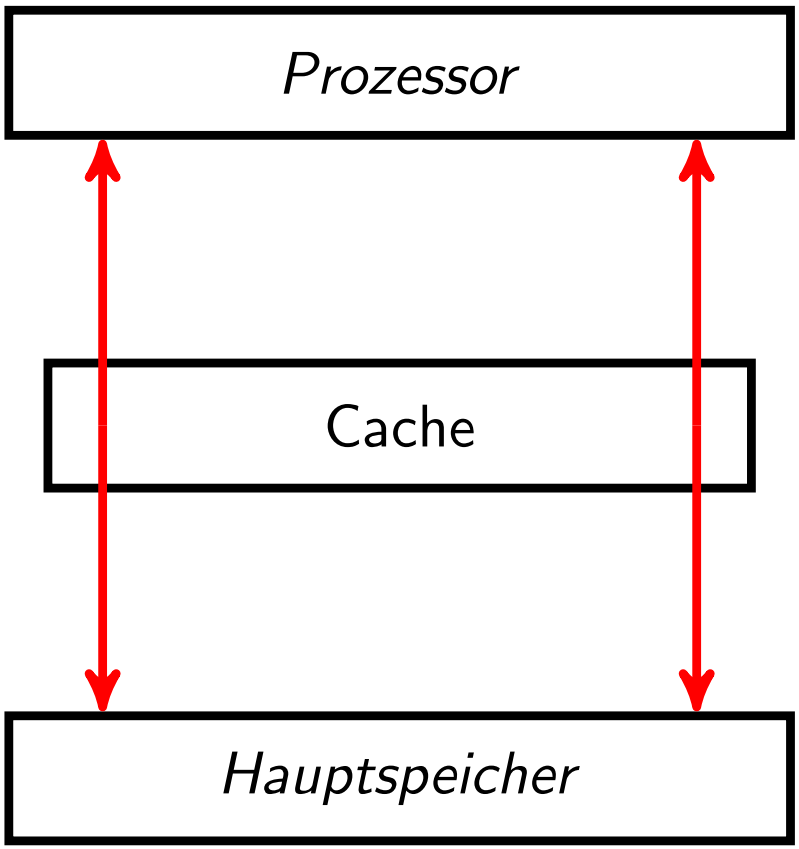
Cachekohärenzkonflikte: Fall 2



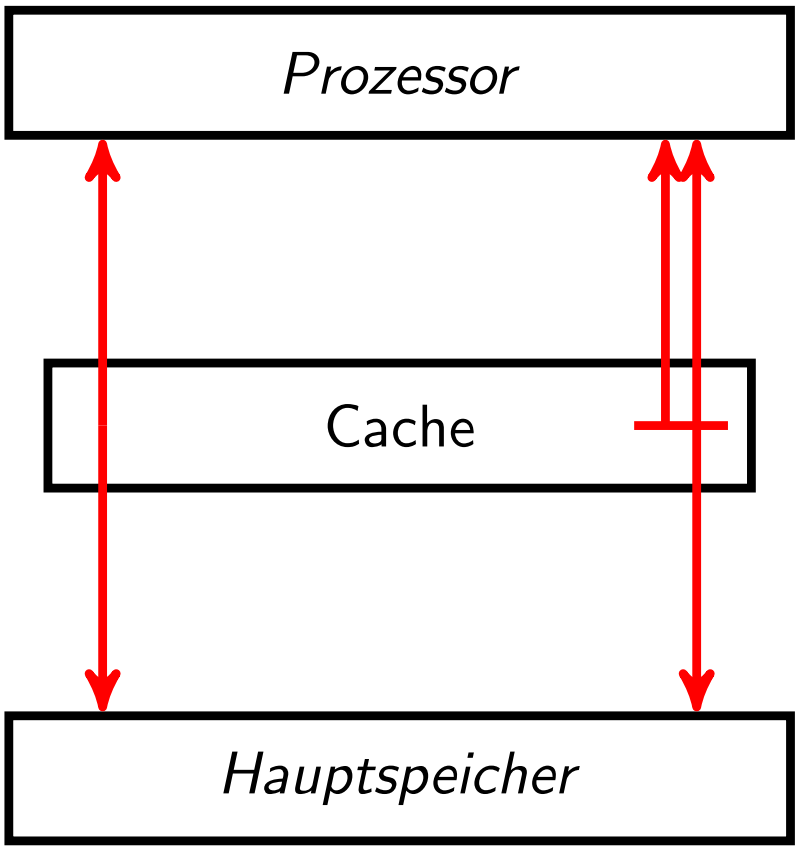
Cachekohärenzkonflikte: Fall 2



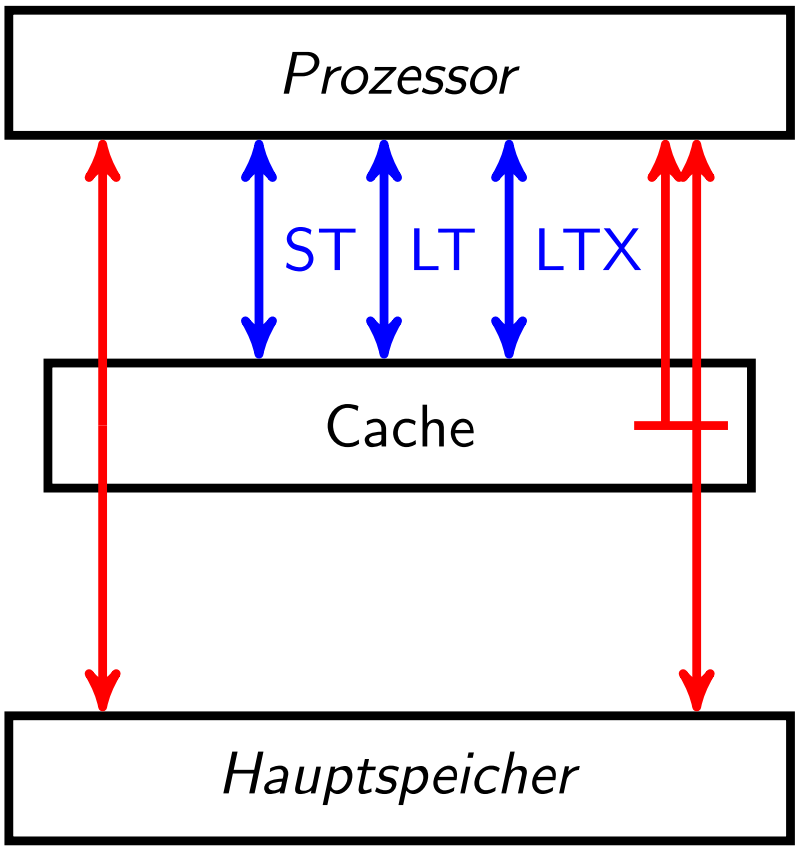
Cachekohärenz zur Konflikterkennung



Cachekohärenz zur Konflikterkennung



Cachekohärenz zur Konflikterkennung



ST: Store Transactional

Folgezustand: DIRTY (D)

LT: Load Transactional

Folgezustand: VALID (V)

LTX: Load Transactional Exclusive

Folgezustand: RESERVED (R)

Cache-basierte TM: Zurücksetzen

Transaktions-Zustand	Bedeutung
EMPTY	Ungültig
NORMAL	Regulärer Cacheeintrag, gültig
XCOMMIT	Im Erfolgsfall löschen
XABORT	Im Fehlerfall löschen

Tabelle: Transaktions-Zustand

- Pro Datum der Schreibmenge werden 2 Cacheeinträge erzeugt:
  - Eintrag 1: „XCOMMIT“, enthält immer den ursprünglichen Wert
  - Eintrag 2: „XABORT“, Modifikationen werden hier ausgeführt
- Echt paralleler COMMIT (*ABORT*) in Hardware

Buszyklen und Transaktionszustand

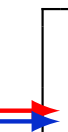
Operation	Beschreibung	Art	Neuer Zustand
L	Wert Lesen	Regulär	Valid
LX	Wert Lesen	Regulär	Reserved
Write	Wert Schreiben	Beides	Dirty
LT	Wert Lesen	Trans	Valid
LTX	Wert Lesen	Trans	Reserved
BUSY	Zugriff verweigern	Trans	Abort

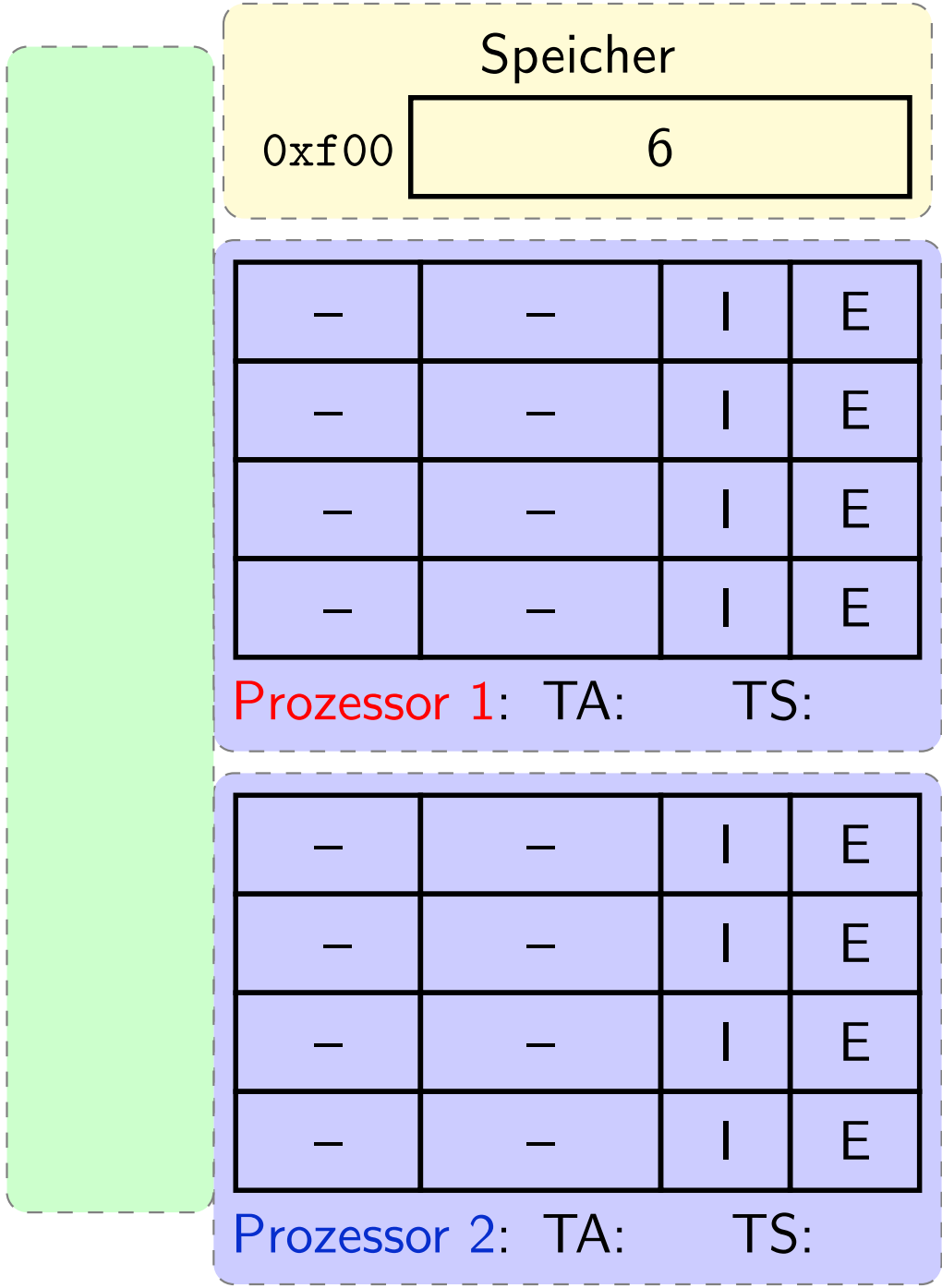
Tabelle: Buszyklen

Jeder Prozessor erhält 2 zusätzliche Flags:

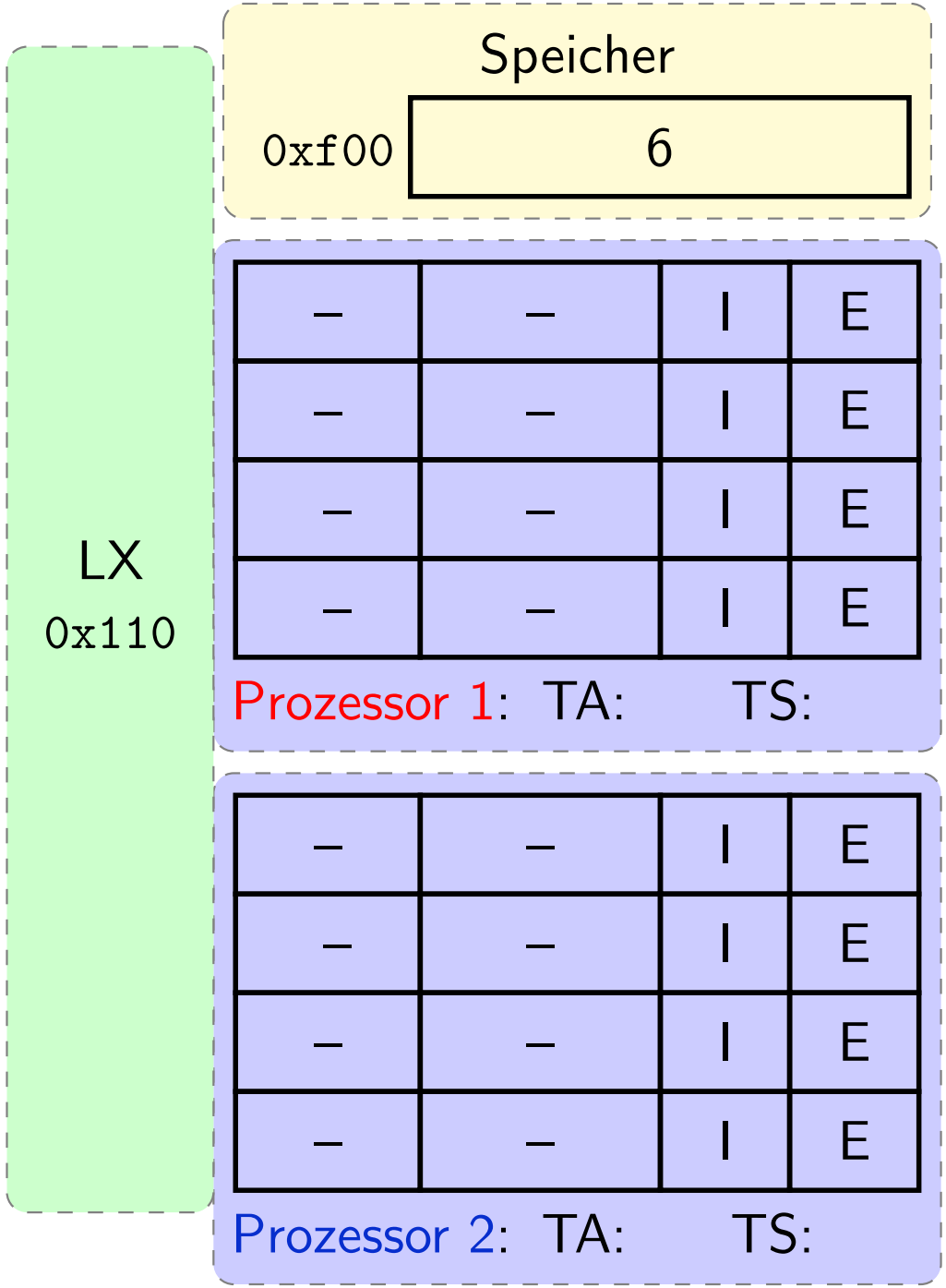
- TACTIVE: Eine Transaktion wird ausgeführt
- TSTATUS: Die Transaktion ist noch gültig

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

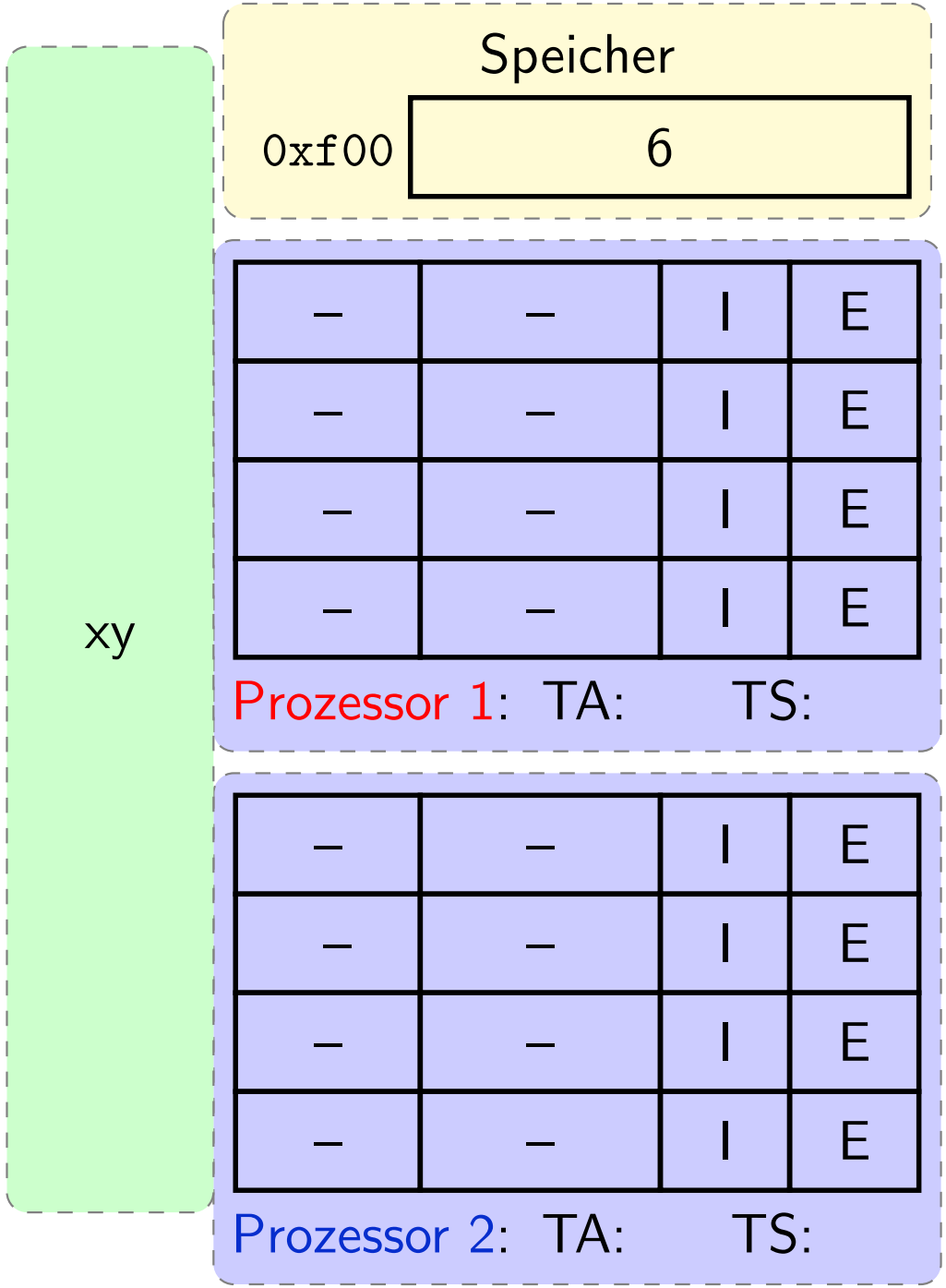




```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	xy	R	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

–	–	I	E
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

–	–	I	E
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

–	–	I	E
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

LX

0x215

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

–	–	I	E
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

z

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

–	–	I	E
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

0x215	z	R	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

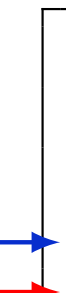
0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



LTX

0xf00

Speicher

0xf00	6

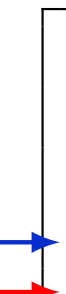
0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



6

Speicher

0xf00	6

0x110	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	0	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	6	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	6	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

LTX
0xf00

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

BUSY

Speicher

0xf00	6

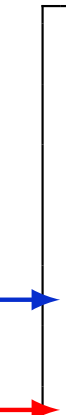
0x110	7	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	R	XC
0xf00	6	R	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	R	XC
0xf00	7	D	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	R	XC
0xf00	7	D	XA
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: ✓ TS: ✓

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

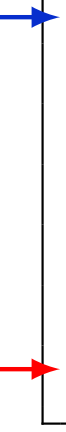
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	0	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```

Speicher

0xf00	6

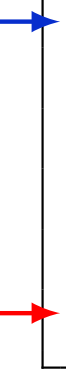
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



Speicher

0xf00	6

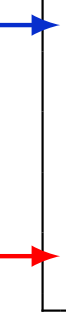
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	6

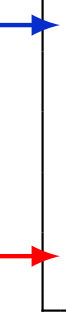
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	6

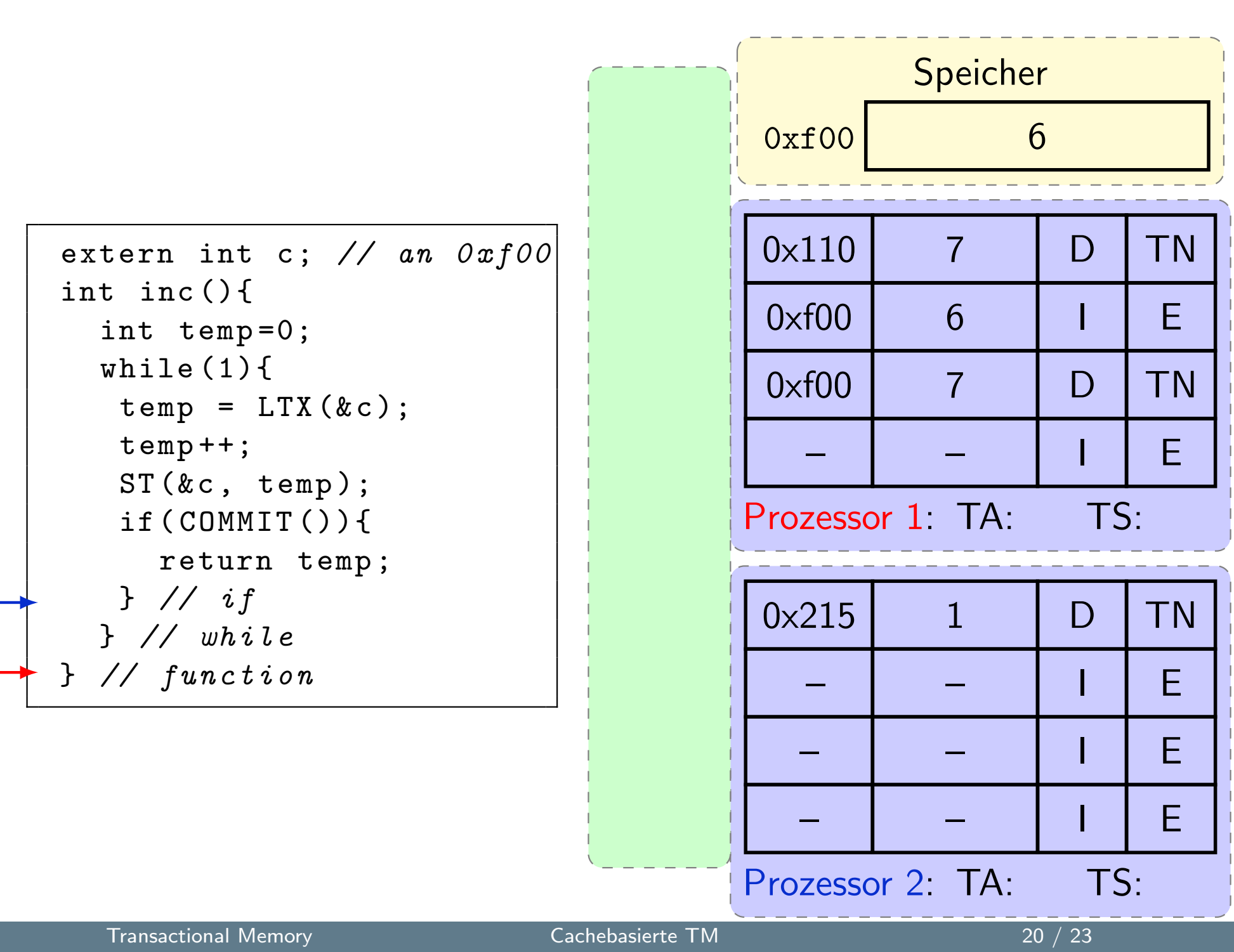
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

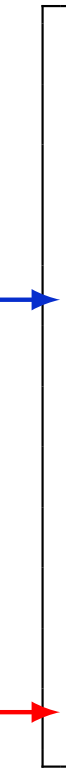
0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	6

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: TS:

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Speicher

0xf00	6

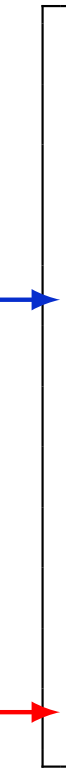
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



LTX

0xf00

Speicher

0xf00	6

Prozessor 1: TA: TS:

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

RETRY

Speicher

0xf00	6

Prozessor 1: TA: TS:

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	D	TN
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```

Write
0xf00
7

Speicher

0xf00	6

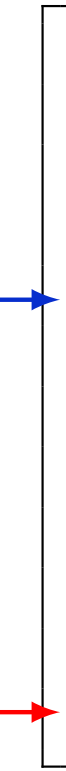
Prozessor 1: TA: TS:

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



LTX

0xf00

Speicher

0xf00	7

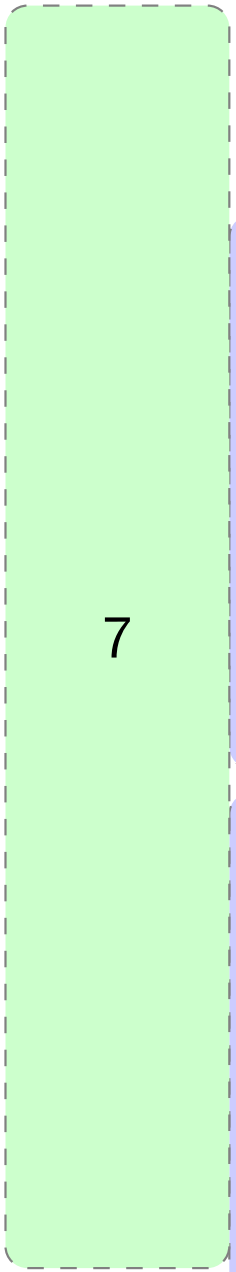
Prozessor 1: TA: TS:

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

0x215	1	D	TN
–	–	I	E
–	–	I	E
–	–	I	E

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	7

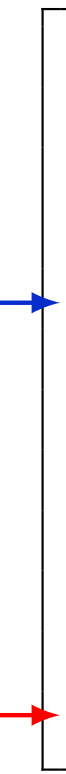
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
0xf00	7	R	XC
0xf00	7	R	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	7

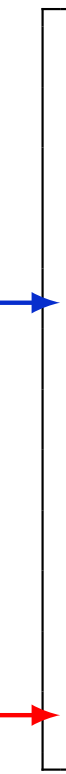
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	1	D	TN
0xf00	7	R	XC
0xf00	7	R	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	7

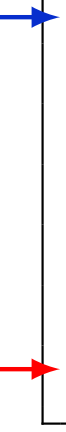
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	7	D	TN
0xf00	7	R	XC
0xf00	7	R	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



Speicher

0xf00	7

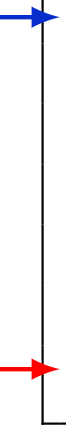
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	7	D	TN
0xf00	7	R	XC
0xf00	7	R	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	7

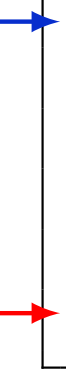
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	8	D	TN
0xf00	7	R	XC
0xf00	7	R	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



Speicher

0xf00	7

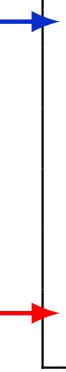
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	8	D	TN
0xf00	7	R	XC
0xf00	7	R	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



Speicher

0xf00	7

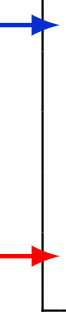
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	8	D	TN
0xf00	7	R	XC
0xf00	8	D	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
   temp = LTX(&c);
   temp++;
   ST(&c, temp);
   if(COMMIT()){
     return temp;
   } // if
  } // while
} // function
```



Speicher

0xf00	7

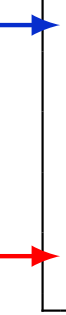
0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	8	D	TN
0xf00	7	R	XC
0xf00	8	D	XA
–	–	I	E

Prozessor 2: TA: ✓ TS: ✓

```
extern int c; // an 0xf00
int inc(){
  int temp=0;
  while(1){
    temp = LTX(&c);
    temp++;
    ST(&c, temp);
    if(COMMIT()){
      return temp;
    } // if
  } // while
} // function
```



Speicher

0xf00	7

0x110	7	D	TN
0xf00	6	I	E
0xf00	7	I	TN
–	–	I	E

Prozessor 1: TA: TS:

0x215	8	D	TN
0xf00	7	I	E
0xf00	8	D	TN
–	–	I	E

Prozessor 2: TA: TS:

Problem: Begrenzte Transaktionsgröße

- Daten einer Transaktion komplett im Cache
- Alle Cacheeinträge doppelt (XCOMMIT/XABORT)
- z.B.:*Sandy-Bride L1: 32kB*
- **Problem: Transaktionsgröße dadurch fest nach oben beschränkt**
- Lösungsansätze:
  - Hybride TM:
    Bei großen Transaktionen auf Softwaresynchronisation ausweichen
  - Hardware Accelerated TM:
    - TM-Umsetzung in Software
    - Teuere Operationen in Hardware umgesetzt

Zusammenfassung: Transactional Memory

- löst Probleme der blockierenden Synchronisation:
  Verklemmung, Prioritätsinversion, ...
- Grundideen:
  - Kritischer Abschnitt atomare Einheit
    $\rightarrow$Transaktion
  - Solange probieren bis Datenzugriff exklusiv
- Hardwareumsetzung mittels Caches vergleichsweise leicht möglich:
  Cache-Kohärenz, Eigentum, Transaktionsvermerke
- Problem: Transaktionsgröße $\leq$ Cachegröße, ...

Bildquellen

- Folie 1:
  http://www4.cs.fau.de/Lehre/SS13/PS_KVBK/papers/tm.pdf
- Folie 2:
  http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell
- Folie 11:
  http://www.1024cores.net/home/parallel-computing/cache-oblivious-algorithms
- Folie 13:
  http://www.elektronik-kompendium.de/sites/com/1308121.htm