```
[<c219ec5f>] security_sk_free+0xf/0x2
[<c2451efb>] __sk_free+0x9b/0x120
[<c25ae7c1>] ? _raw_spin_unlock_irqre
[<c2451ffd>] sk_free+0x1d/0x30
[<c24f1024>] unix_release_sock+0x174/
```

Hardware Transactional Memory (1)

Mohamed Mohamedin

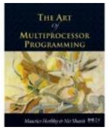
Chapter 5 of TM Book

MESI Cache Coherence Protocol

- One of the famous Cache Coherence Protocols
  - We will study it to show how we can exploit Cache Coherence Protocol to create HTM

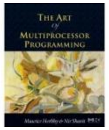
MESI

- Modified
  - Have modified cached data, must write back to memory



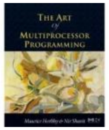
MESI

- Modified
  - – Have modified cached data, must write back to memory
- Exclusive
  - – Not modified, I have only copy



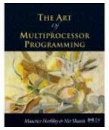
MESI

- Modified
  - Have modified cached data, must write back to memory
- Exclusive
  - Not modified, I have only copy
- Shared
  - Not modified, may be cached elsewhere

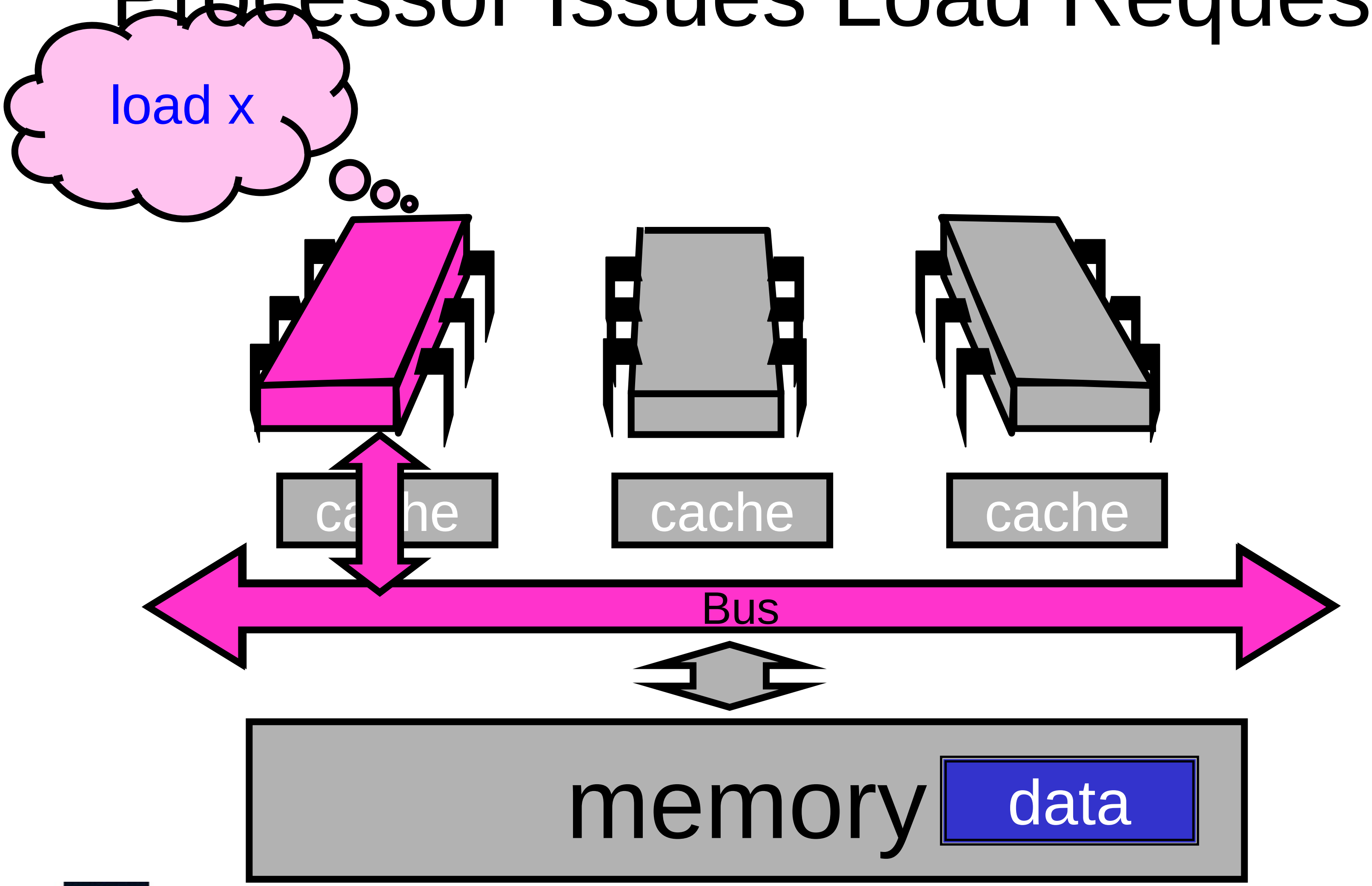


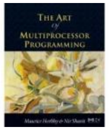
MESI

- Modified
  - Have modified cached data, must write back to memory
- Exclusive
  - Not modified, I have only copy
- Shared
  - Not modified, may be cached elsewhere
- Invalid
  - Cache contents not meaningful

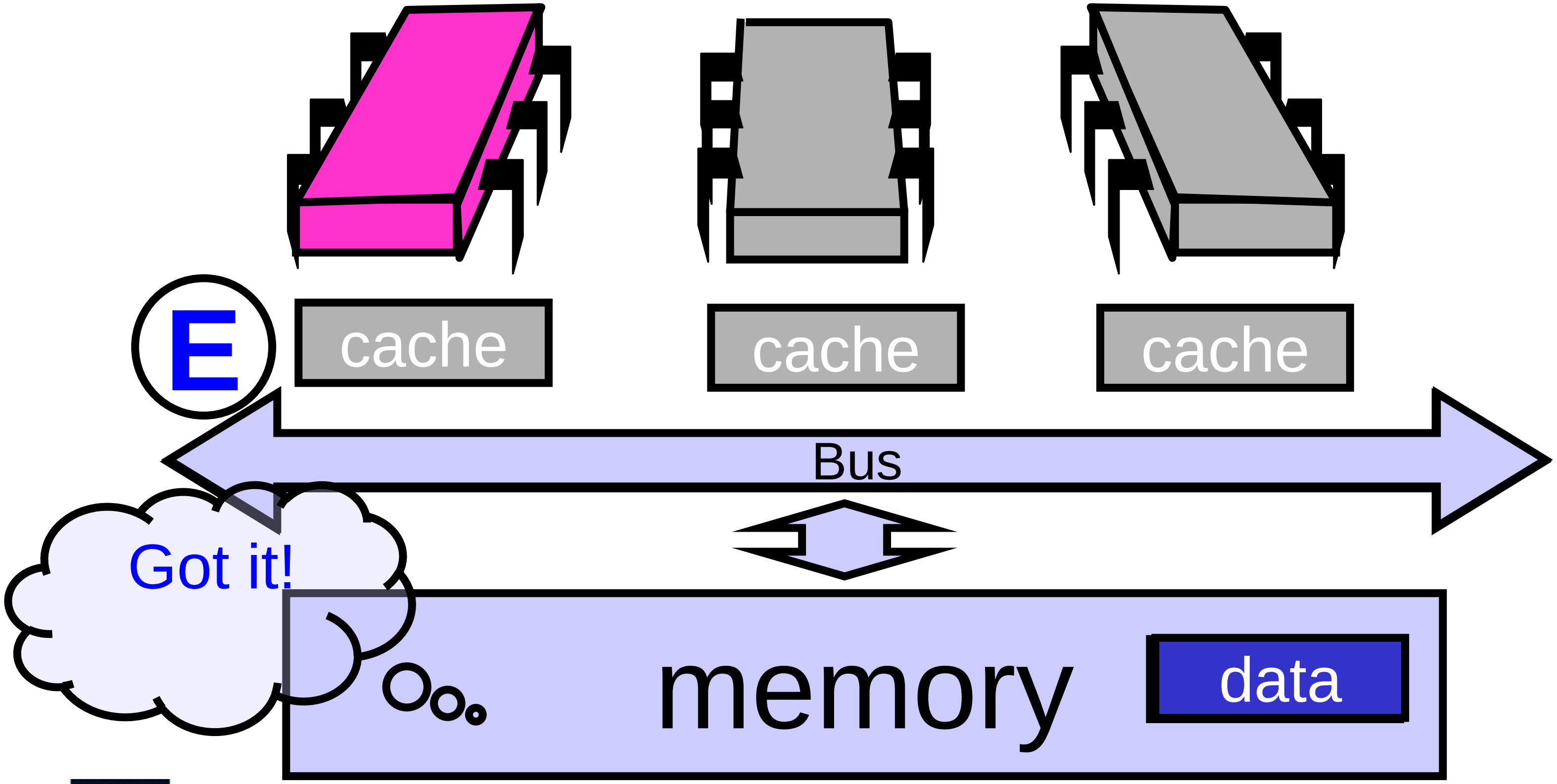


Processor Issues Load Request

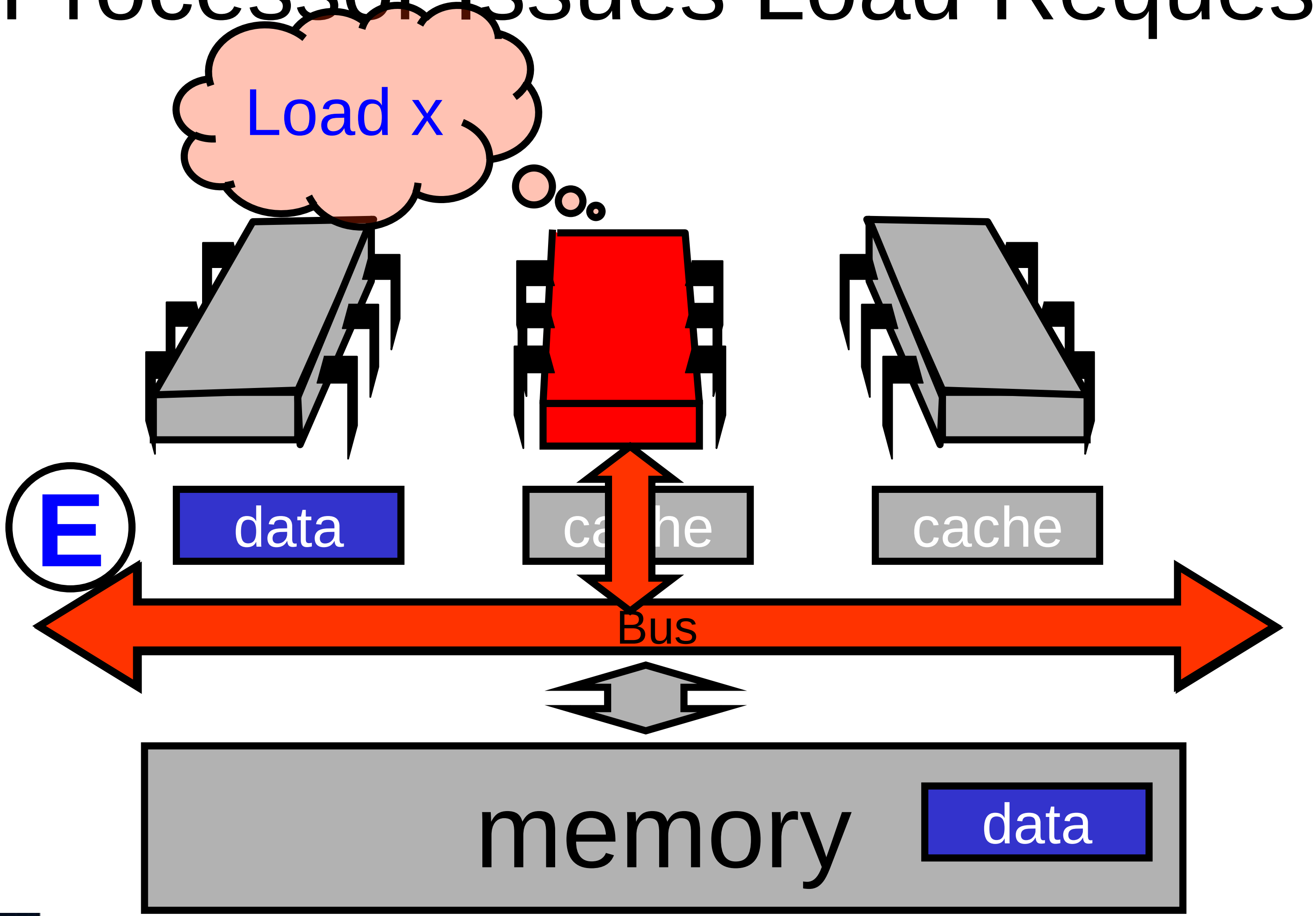


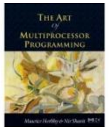


Memory Responds

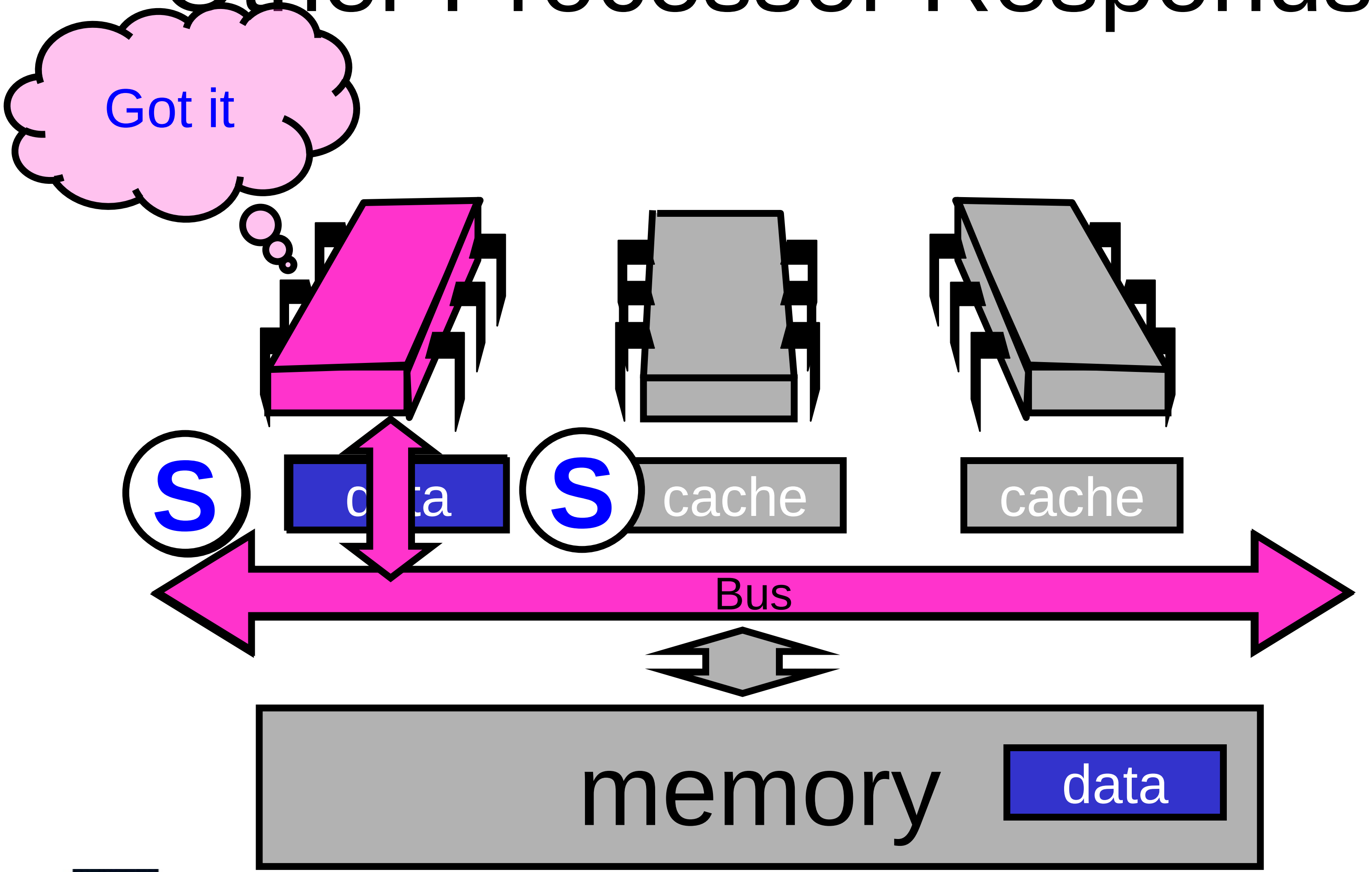


Processor Issues Load Request

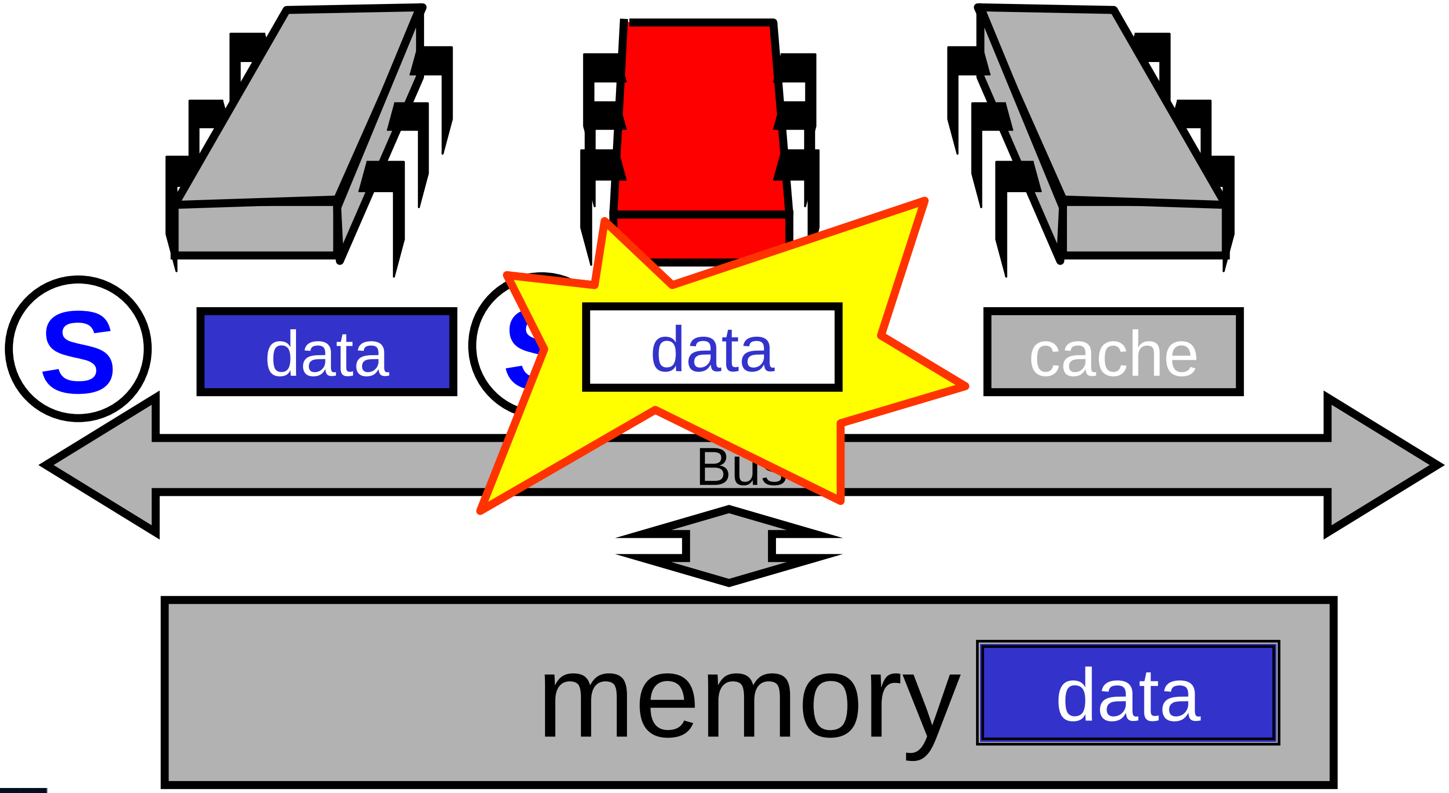




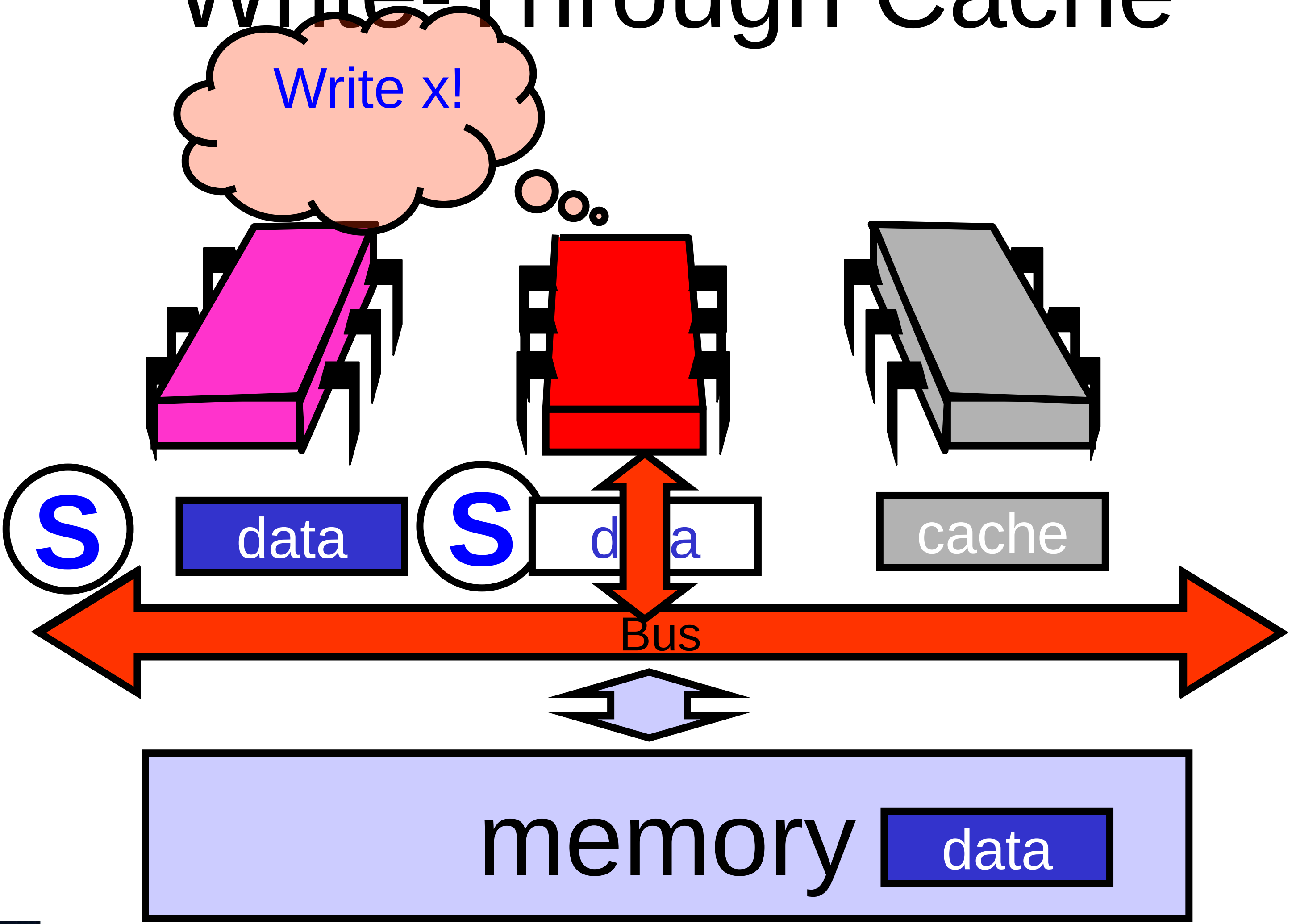
Other Processor Responds



Modify Cached Data

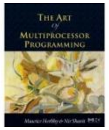


Write-Through Cache



Write-Through Caches

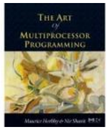
- Immediately broadcast changes
- Good
  - – Memory, caches always agree
  - – More read hits, maybe
- Bad
  - – Bus traffic on all writes
  - – Most writes to unshared data
  - – For example, loop indexes …



Write-Through Caches

- Immediately broadcast changes
- Good
  - – Memory, caches always agree
  - – More read hits, maybe
- Bad
  - – Bus traffic on all writes
  - – Most writes to unshared data
  - – For example, loop indexes …

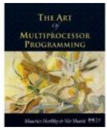
"show stoppers"



Write-Through Caches

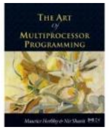
- Immediately broadcast changes
- Good
  - Memory, caches always agree
  - More read hits, maybe
- Bad
  - Bus traffic on all writes
  - Most writes to unshared data
  - For example, loop indexes …

"Also, not suitable for TM"

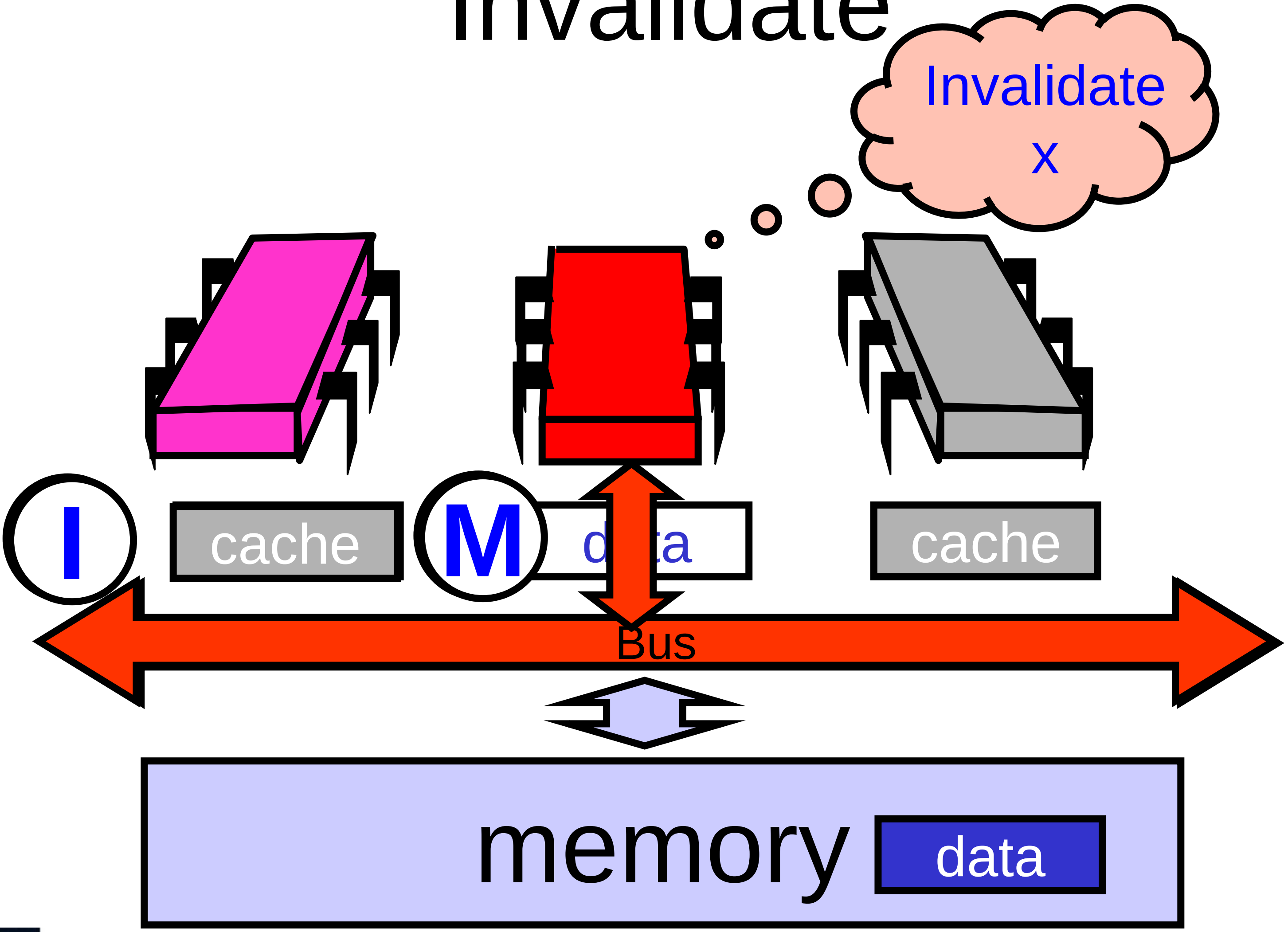


Write-Back Caches

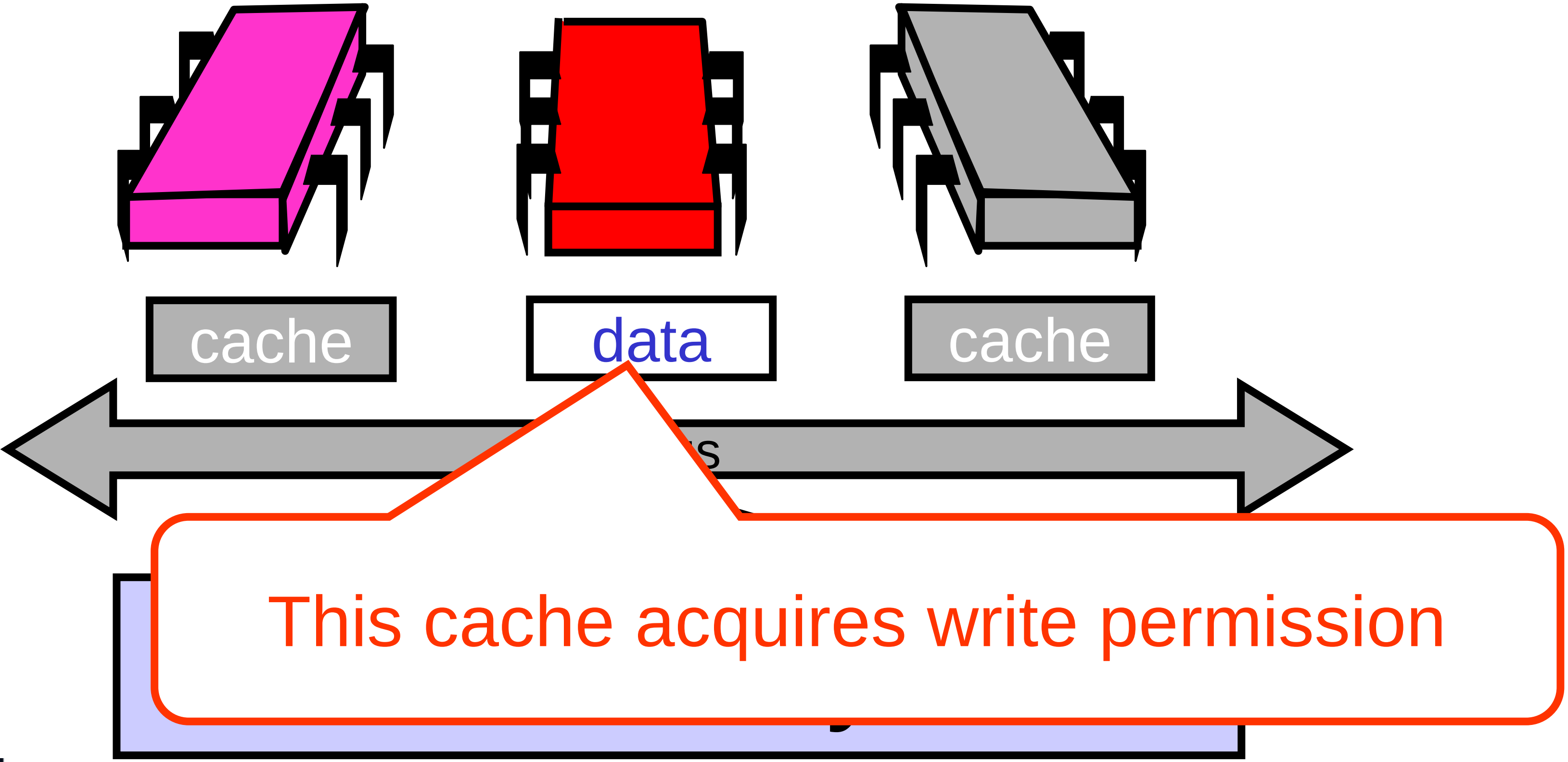
- Accumulate changes in cache
- Write back when line evicted
  - – Need the cache for something else
  - – Another processor wants it



Invalidate

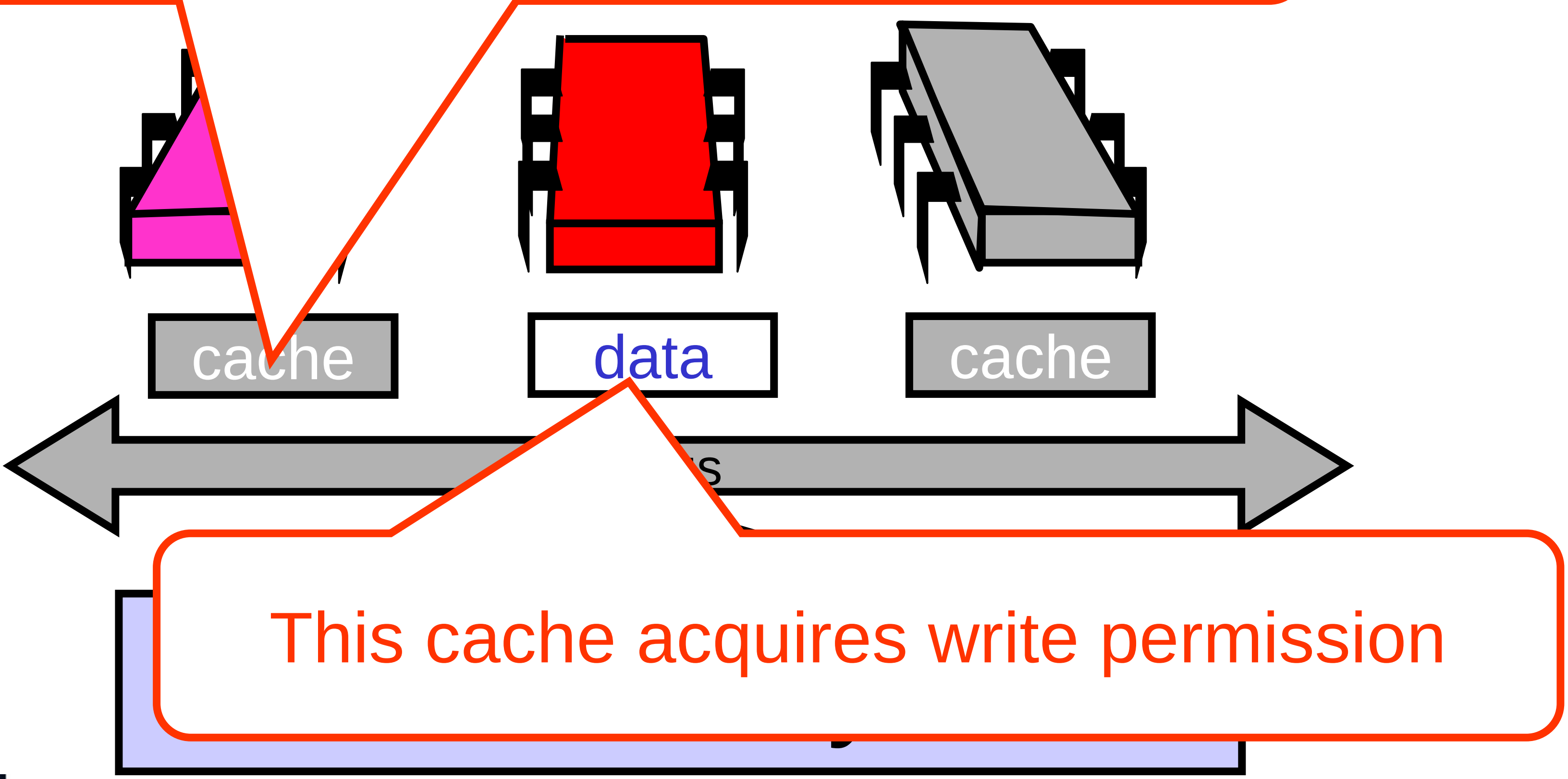


Invalidate



This cache acquires write permission

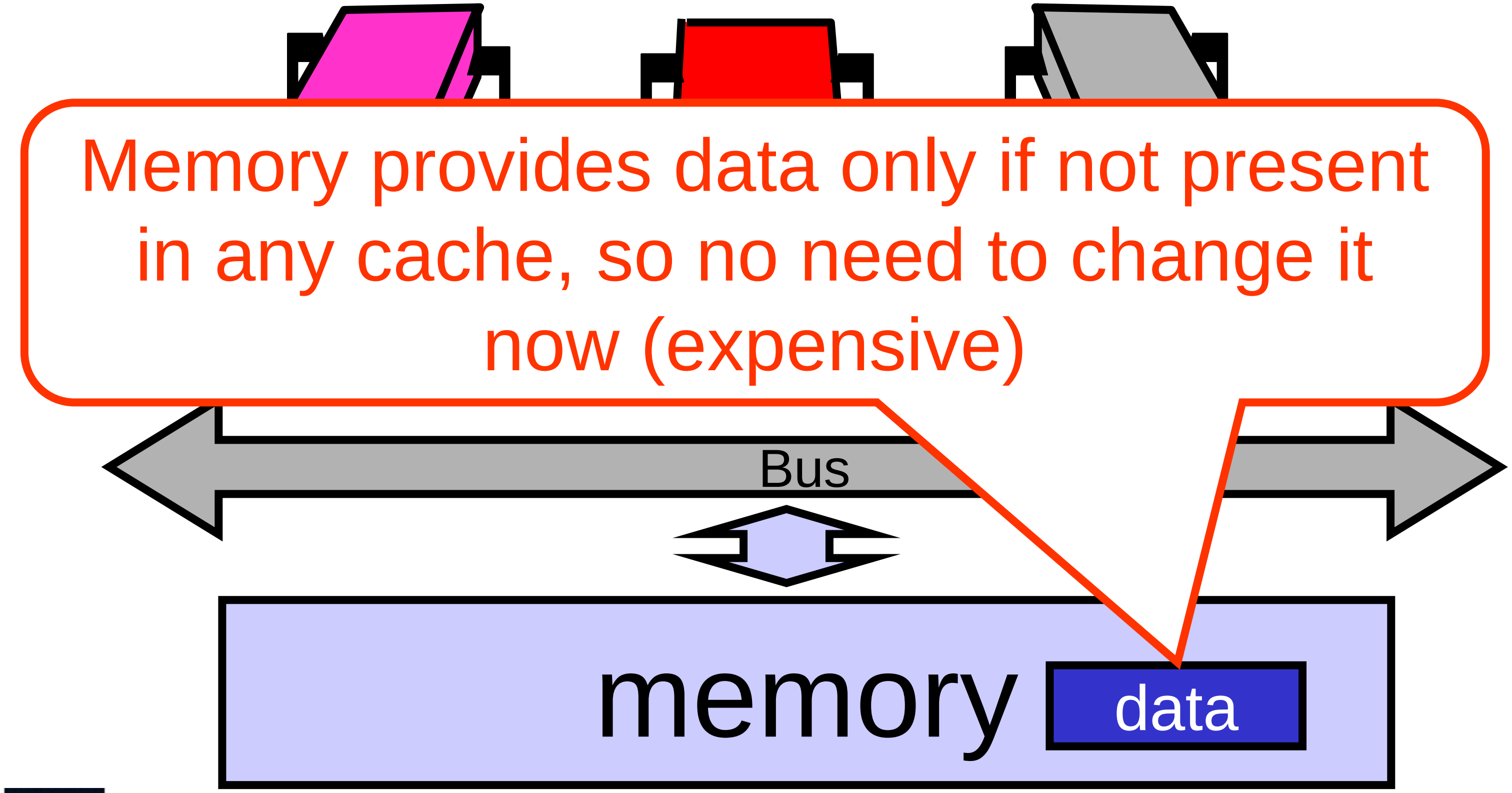
Invalidate



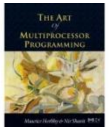
Other caches lose read permission

This cache acquires write permission

Invalidate

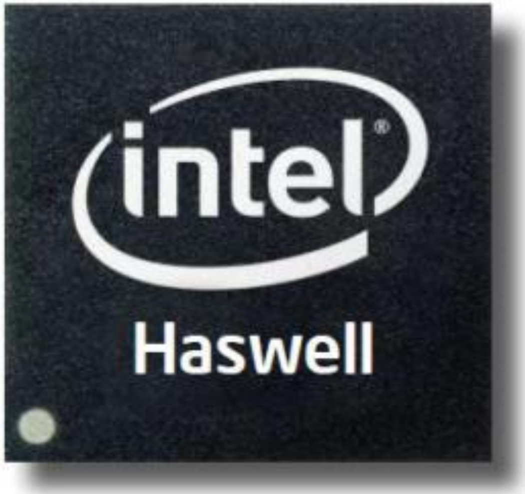


Memory provides data only if not present in any cache, so no need to change it now (expensive)

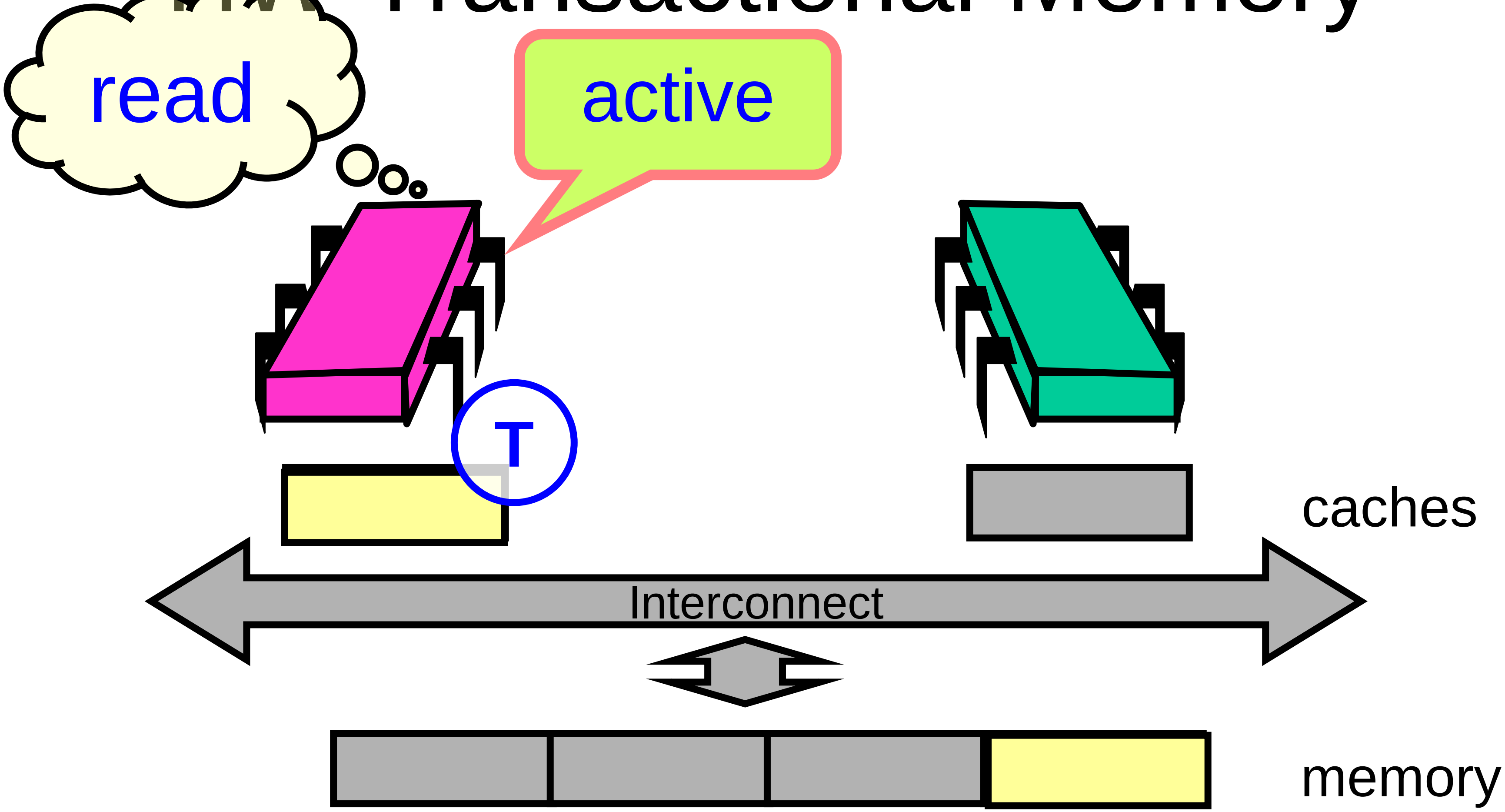


Hardware Transactional Memory

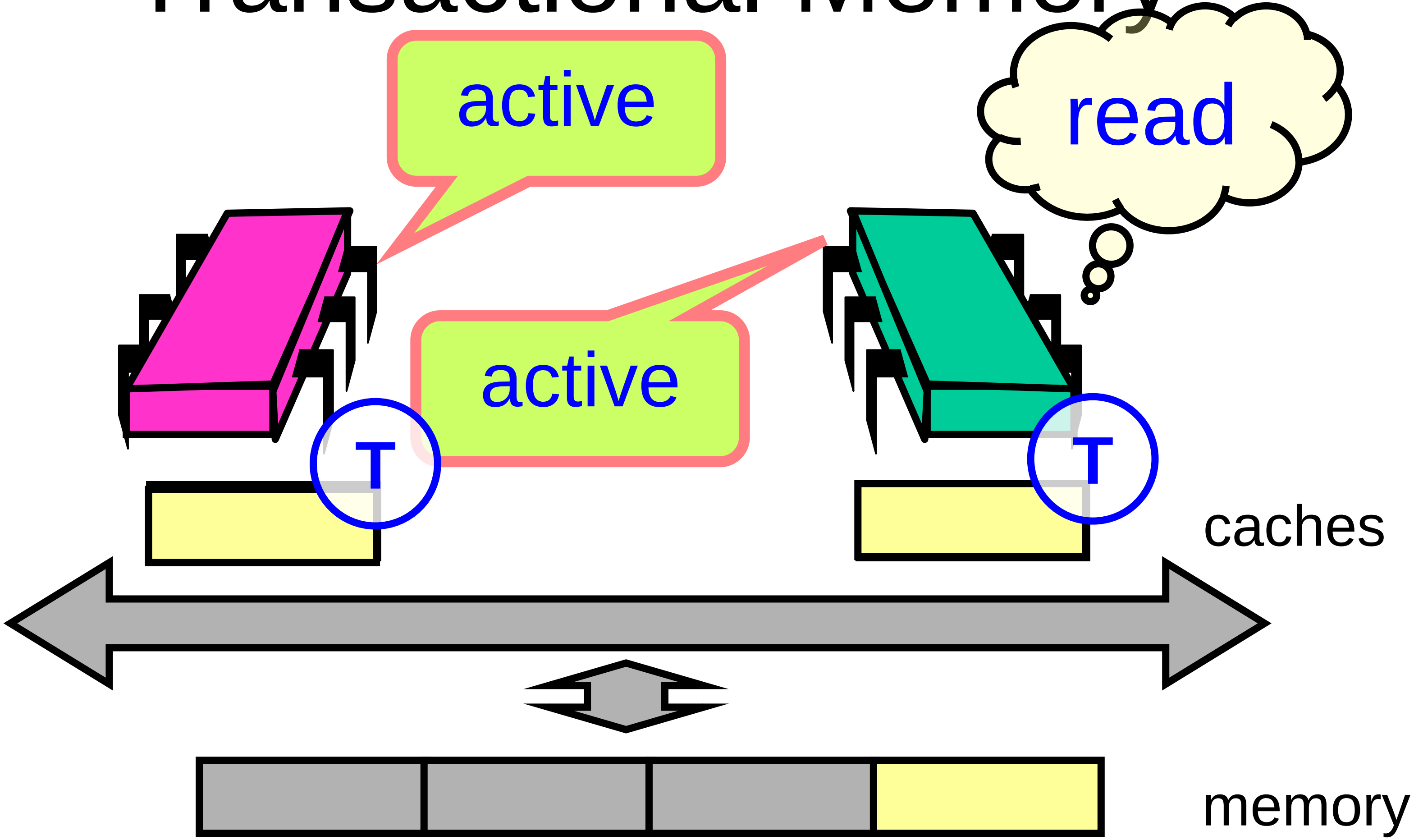
- Exploit Cache coherence
- Already almost does it
  - Invalidation
  - Consistency checking
- Speculative execution
  - Branch prediction = optimistic synch!



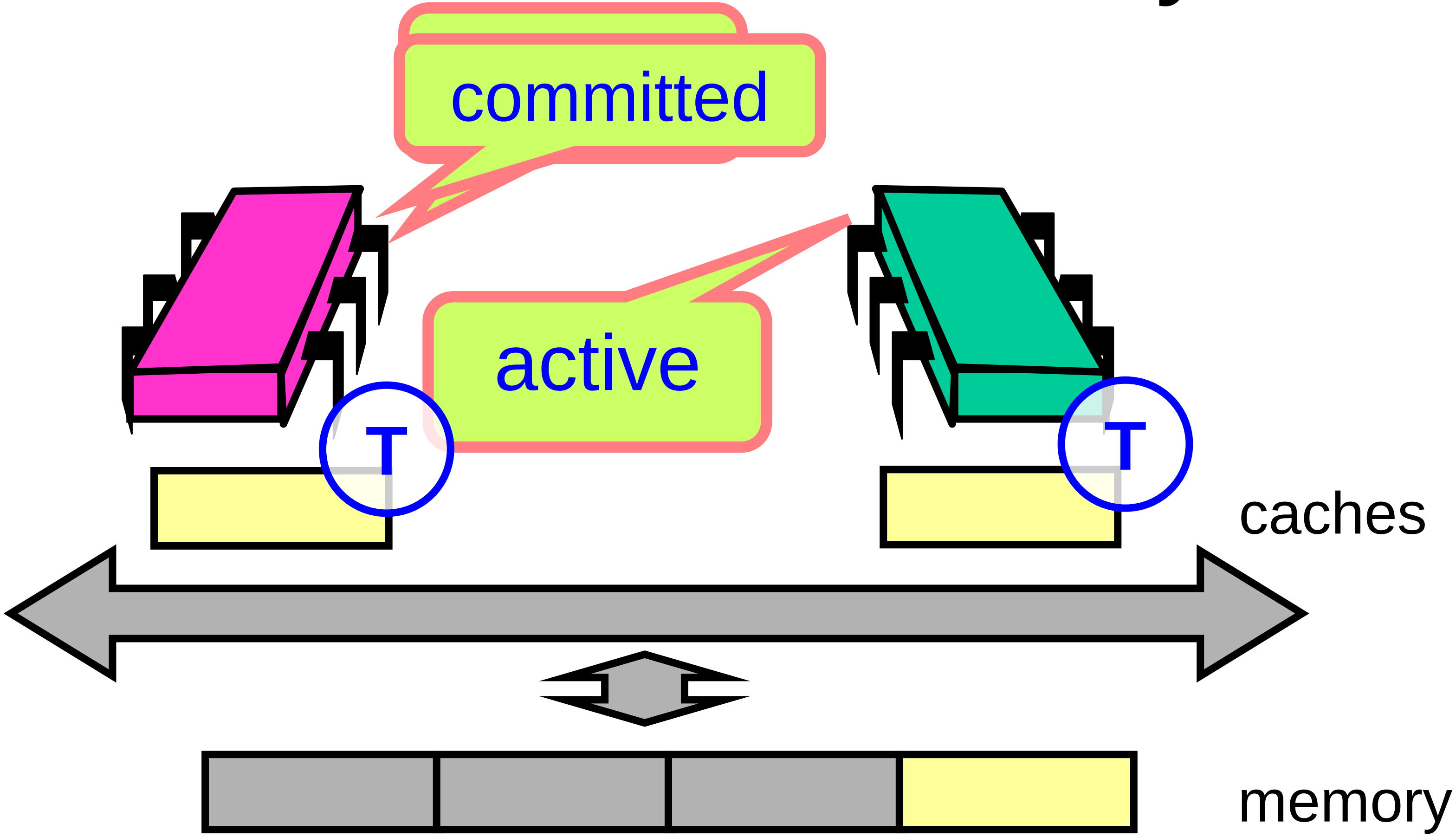
HW Transactional Memory

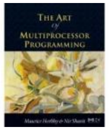


Transactional Memory

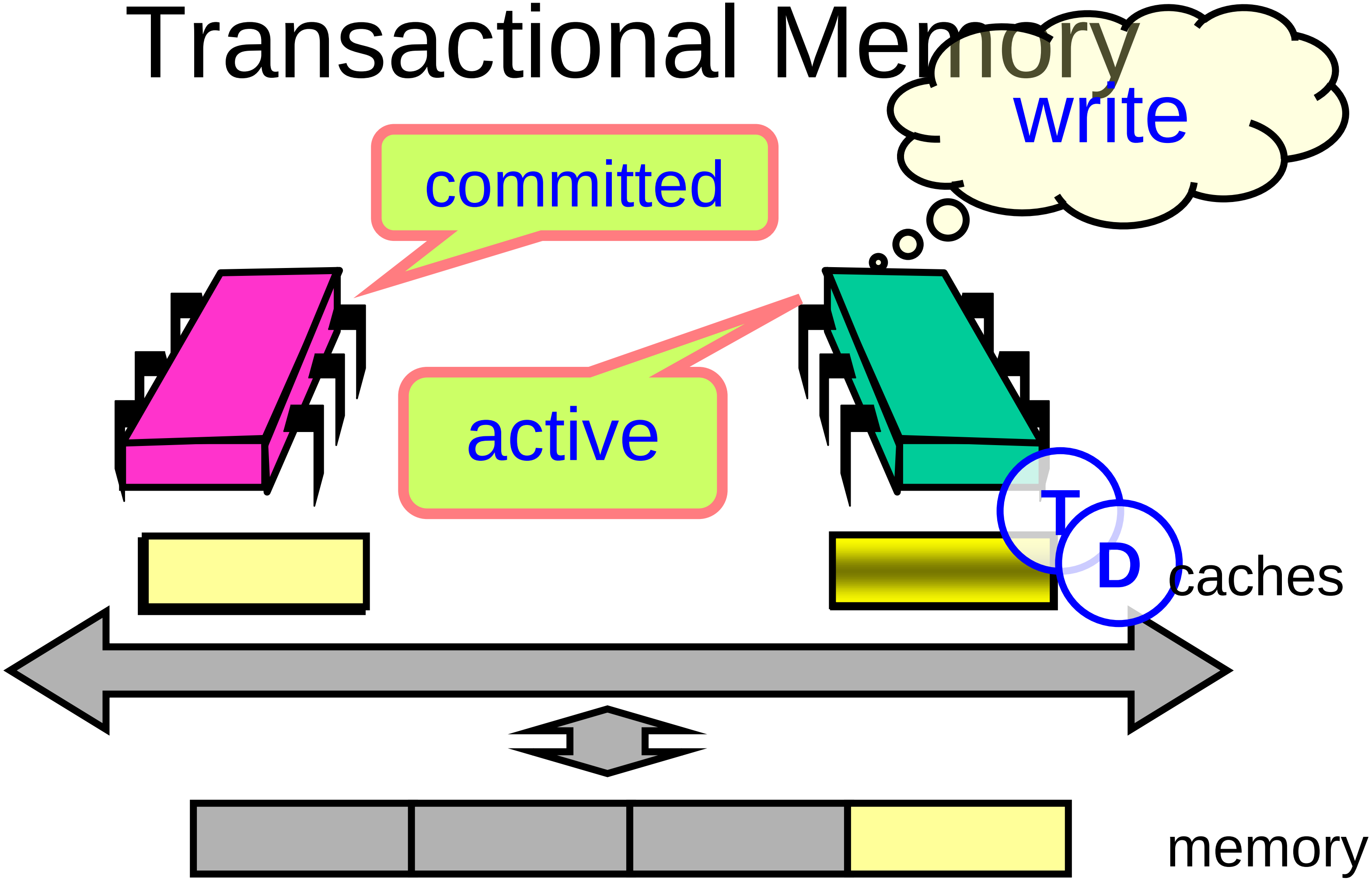


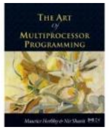
Transactional Memory



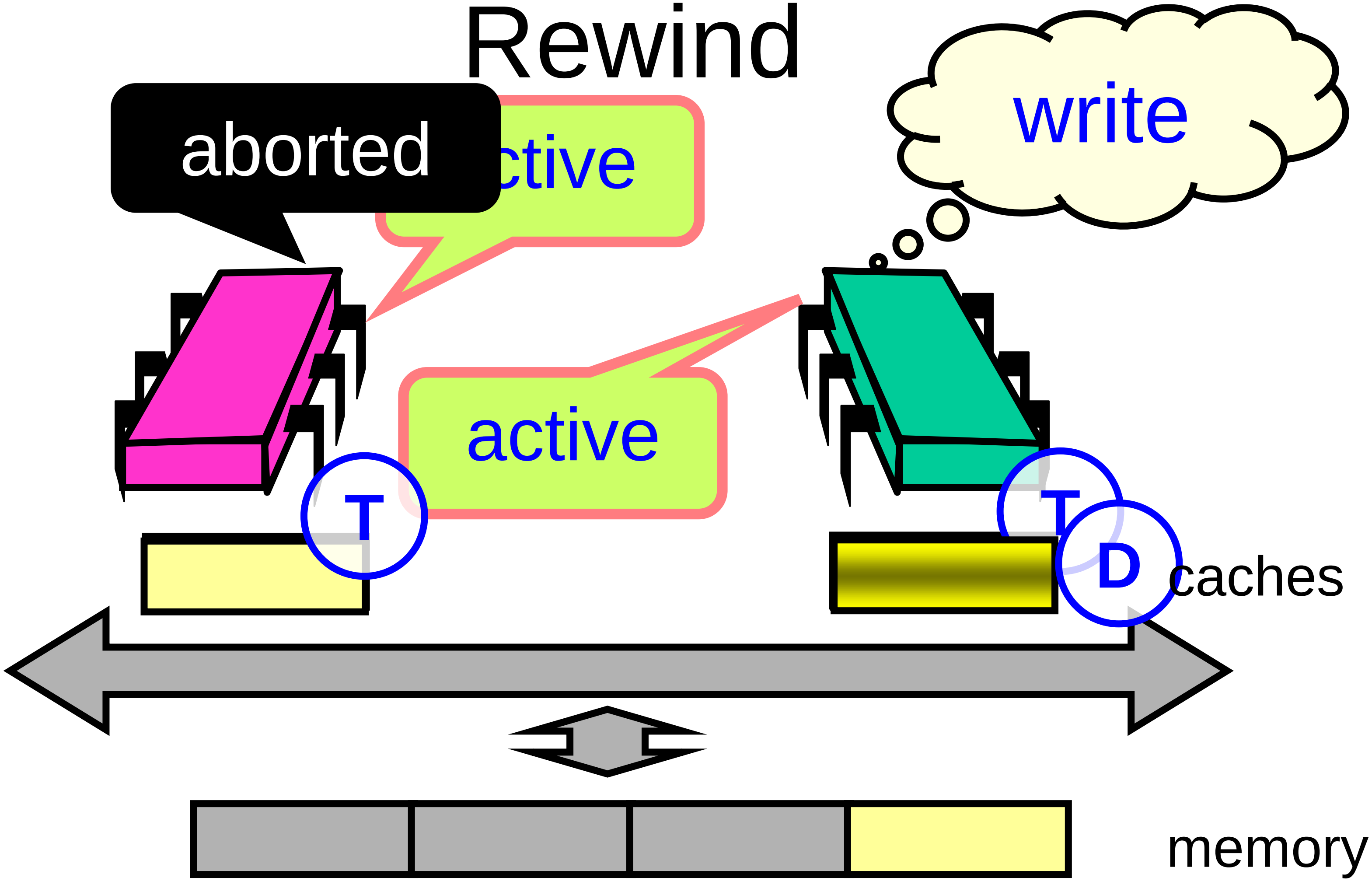


Transactional Memory



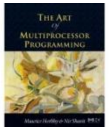


Rewind



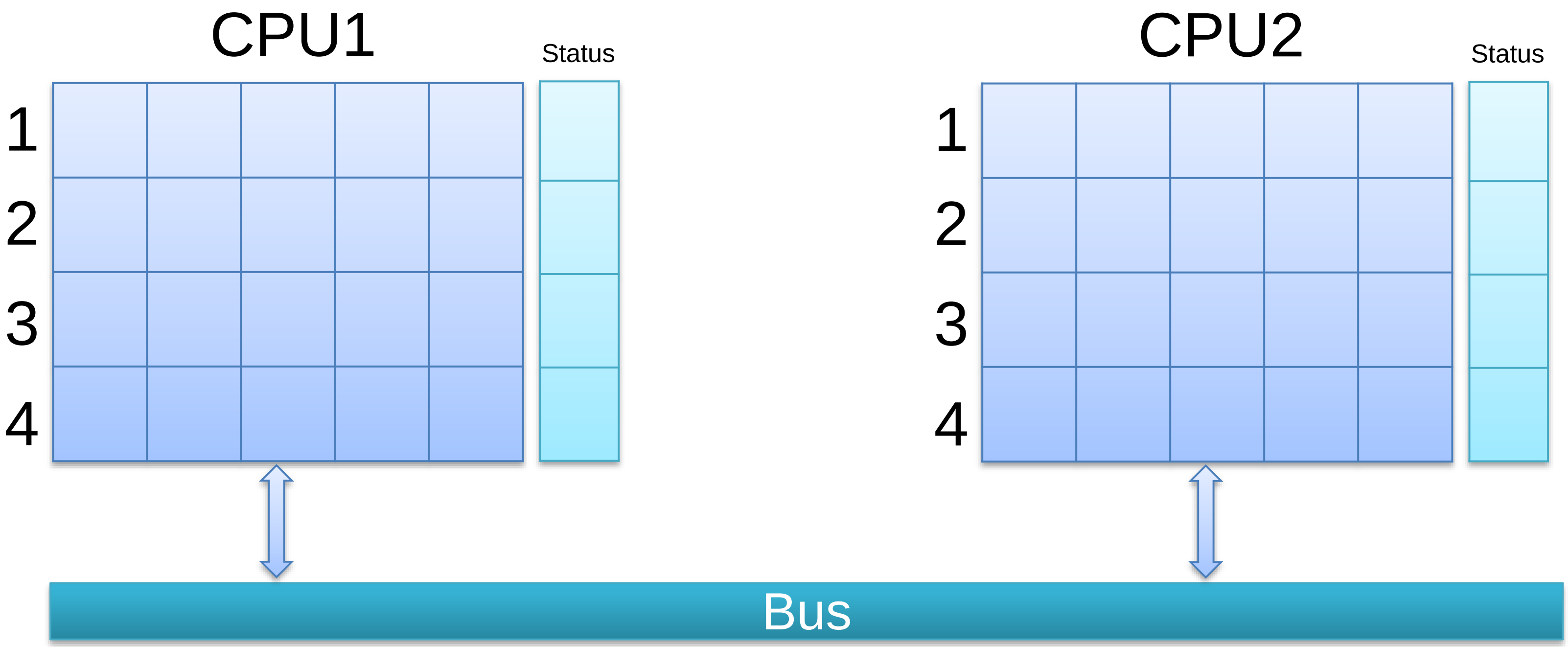
Transaction Commit

- At commit point
  - – If no cache conflicts, we win.
- Mark transactional entries
  - – Read-only: valid
  - – Modified: dirty (eventually written back)
- That’s all, folks!
  - – Except for a few details …



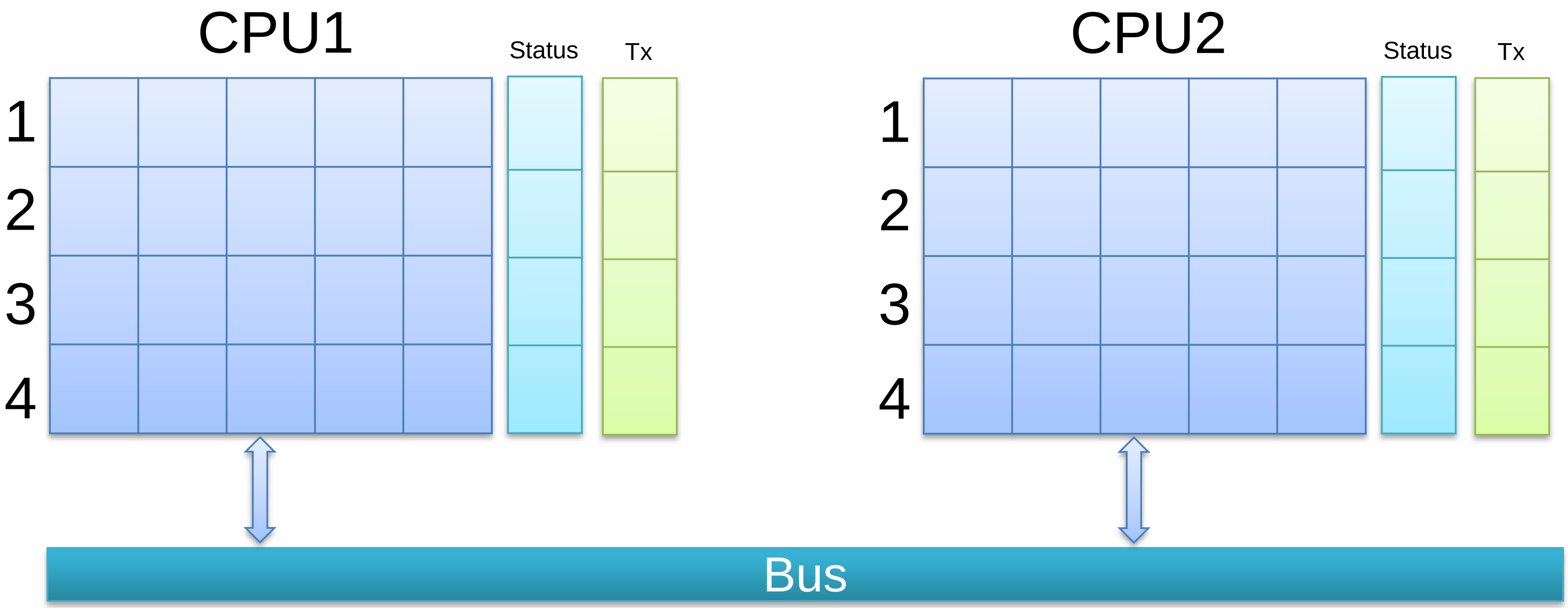
Cache coherence-based HTM

MESI Cache Coherence Protocol



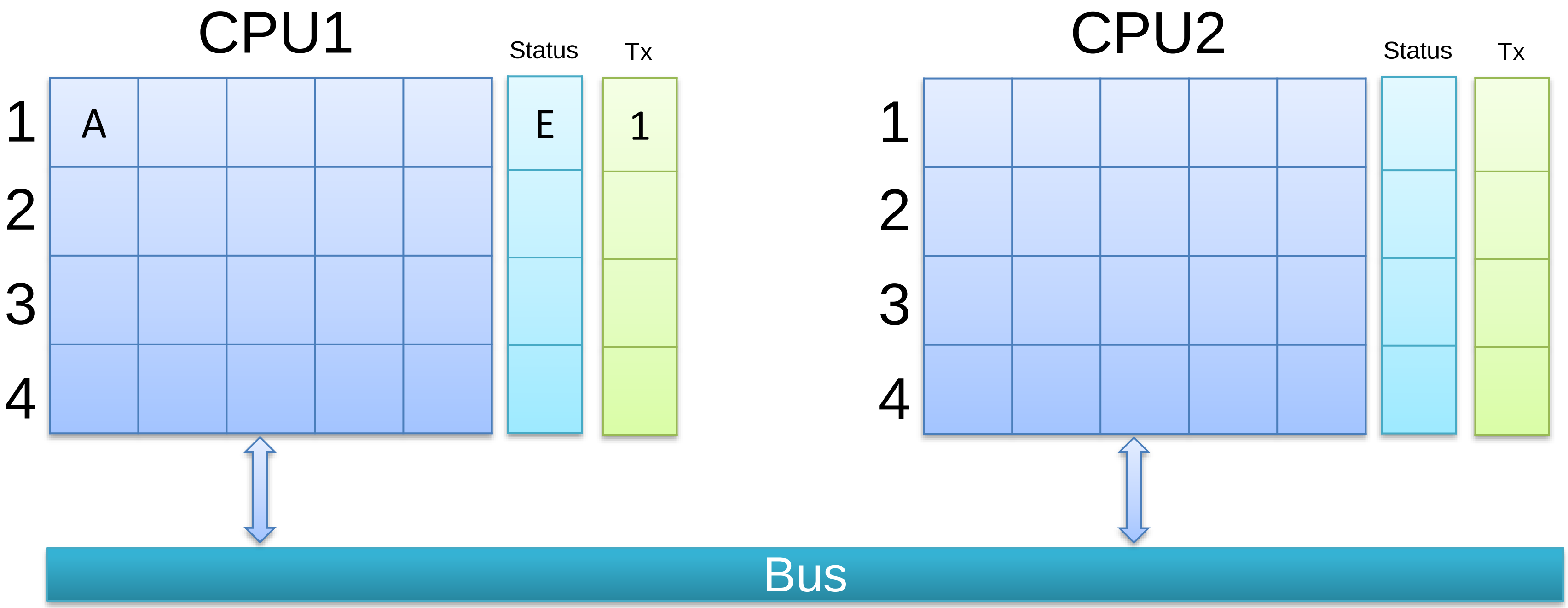
Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



CPU1

Status

Tx

1

2

3

4

A

E

1

CPU2

Status

Tx

1

2

3

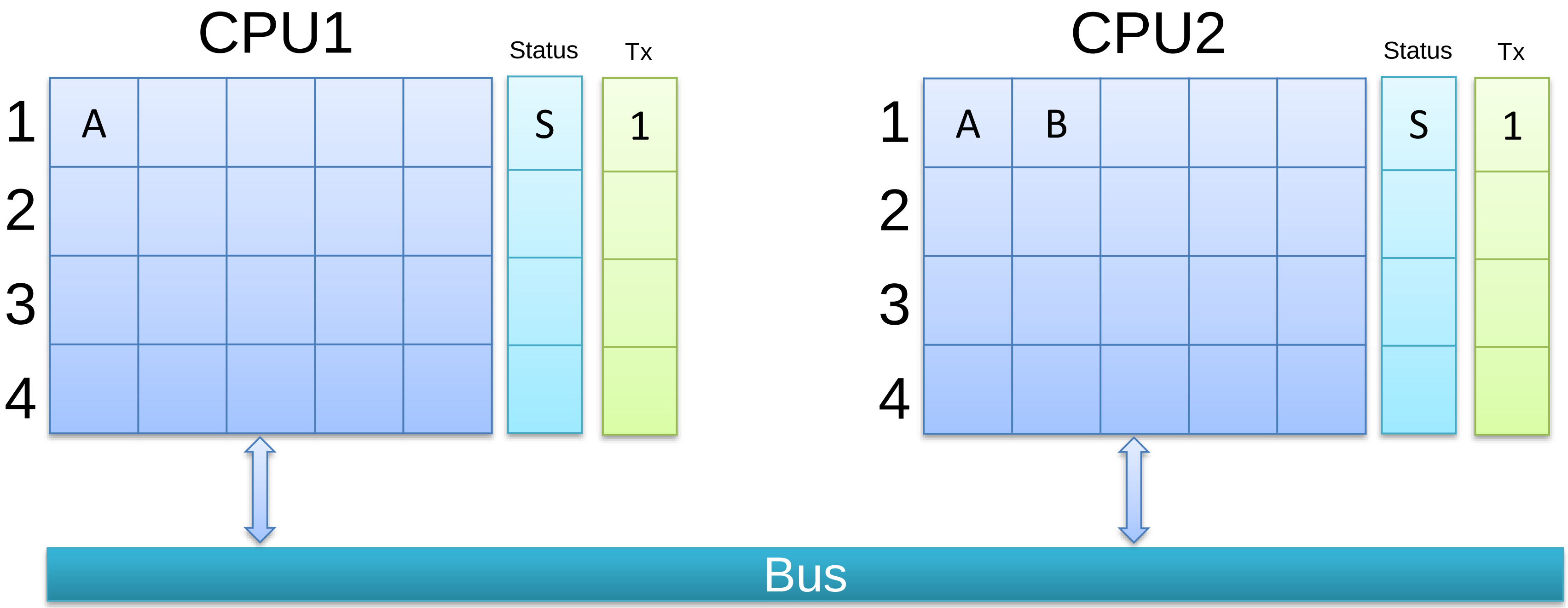
4

Bus

- _xbegin()
- read (0)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol

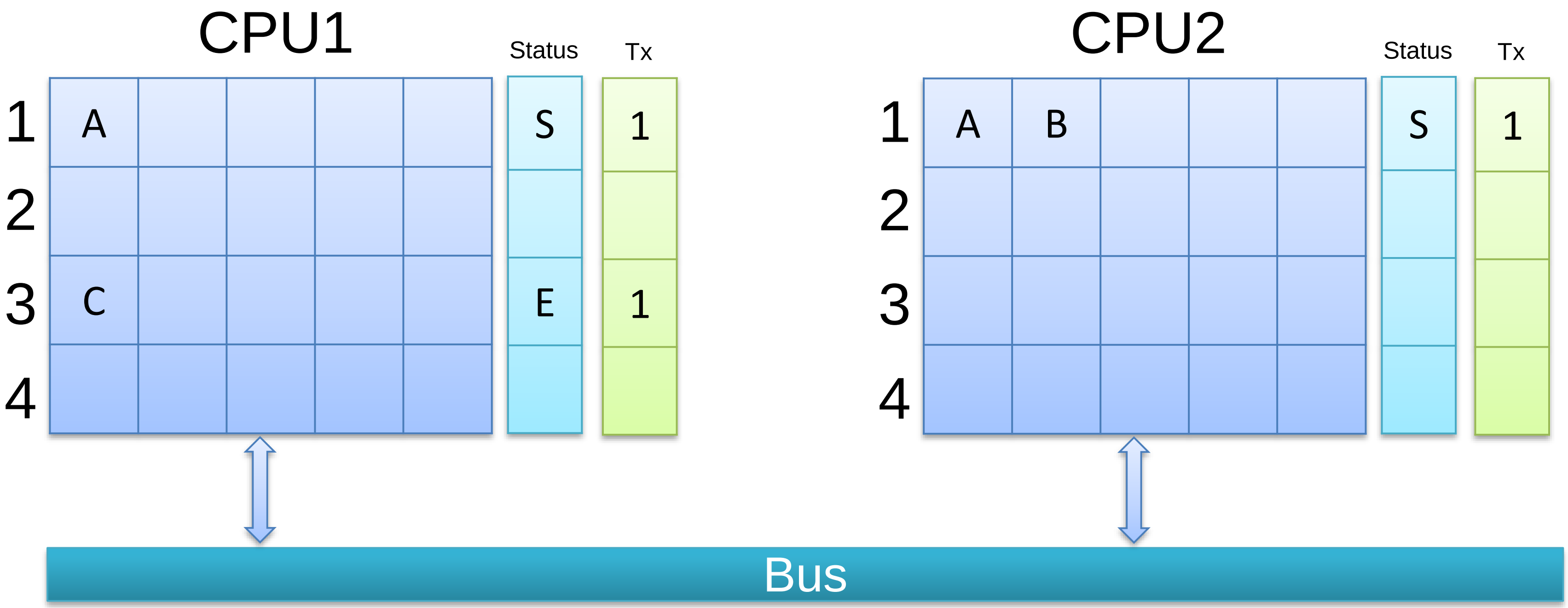


- _xbegin()
- read (0)

- _xbegin()
- read (1)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



CPU1

						Status	Tx
1	A					S	1
2							
3	C					E	1
4							

CPU2

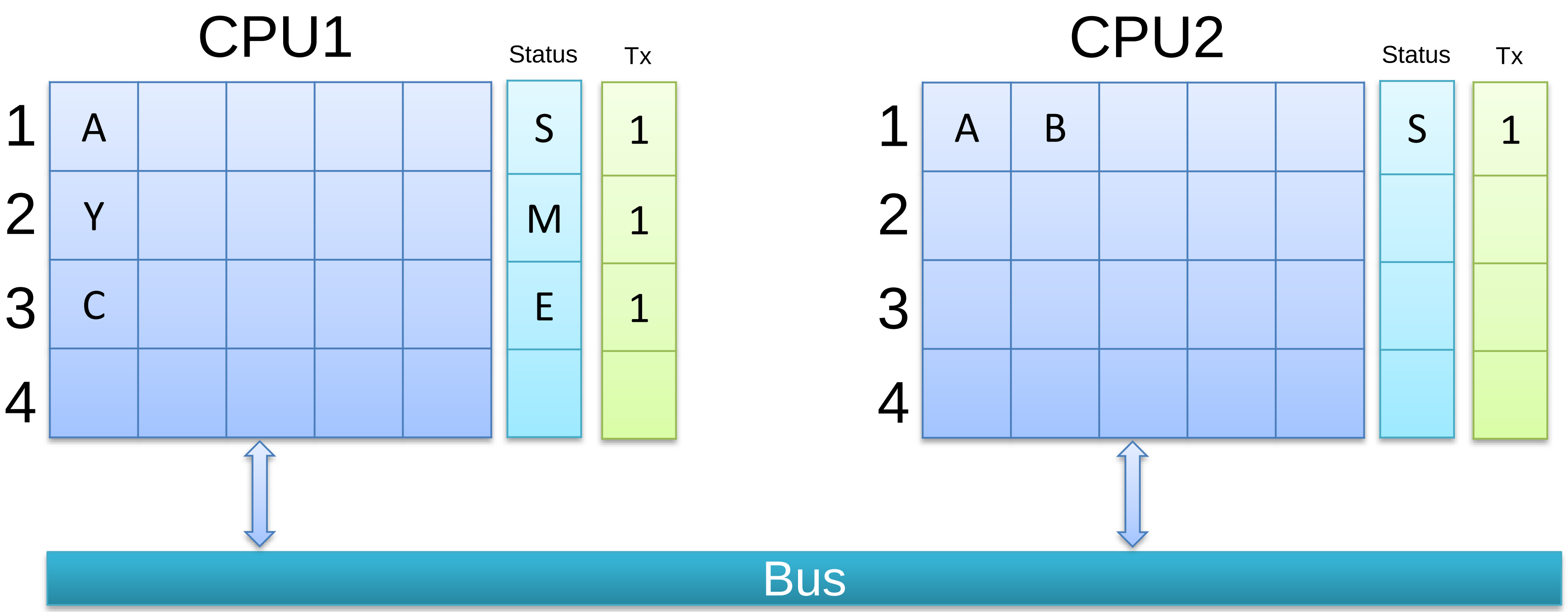
						Status	Tx
1	A	B				S	1
2							
3							
4							

- _xbegin()
- read (0)
- read (10)

- _xbegin()
- read (1)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



CPU1

						Status	Tx
1	A					S	1
2	Y					M	1
3	C					E	1
4							

CPU2

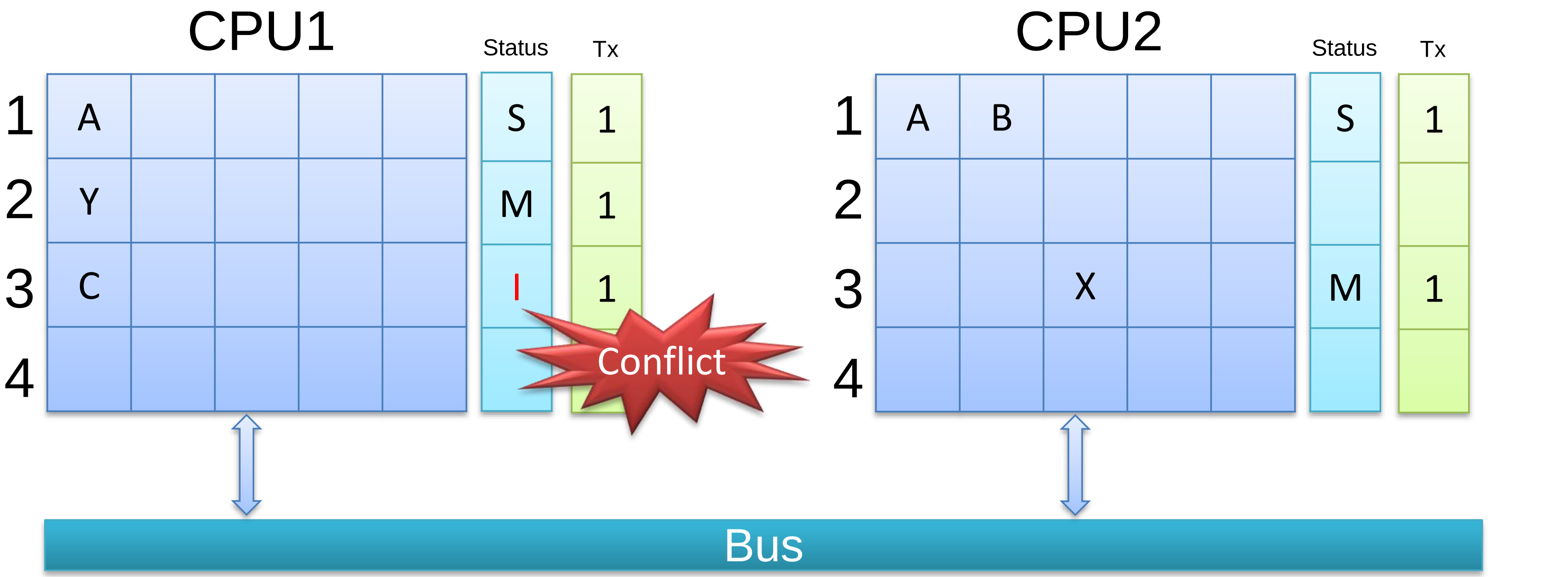
						Status	Tx
1	A	B				S	1
2							
3							
4							

- _xbegin()
- read (0)
- read (10)
- write (5, Y)

- _xbegin()
- read (1)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol

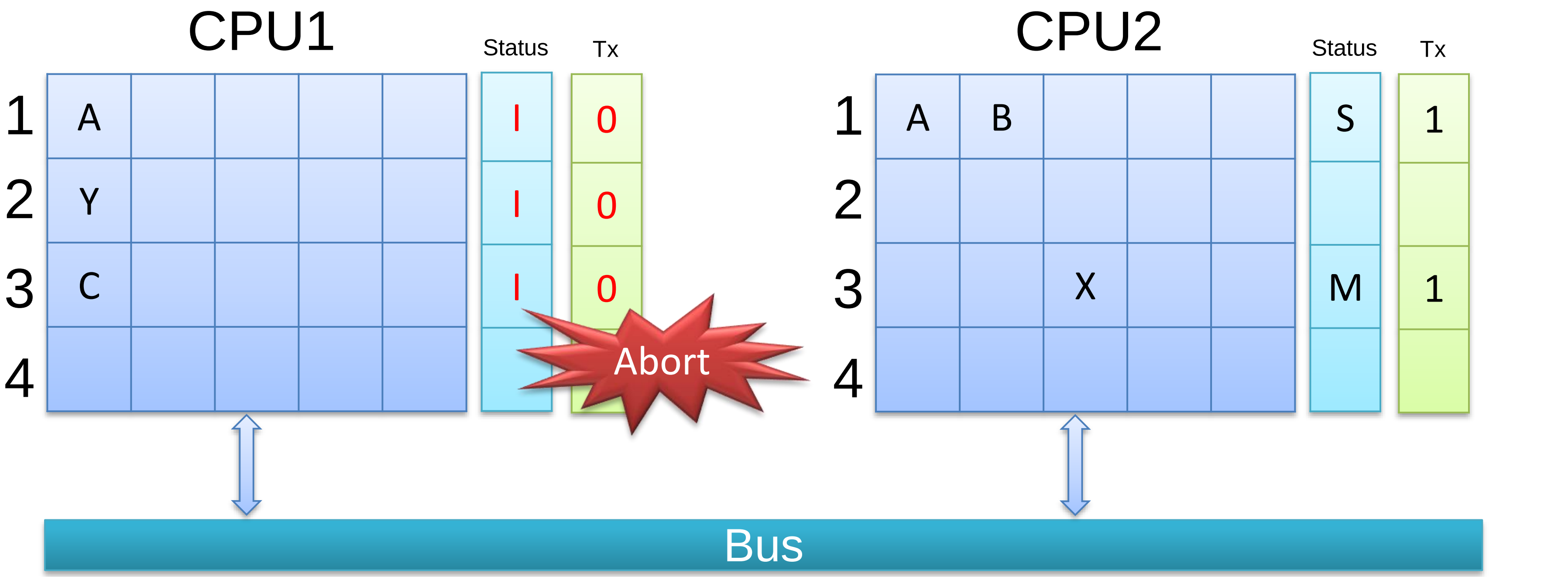


- _xbegin()
- read (0)
- read (10)
- write (5, Y)

- _xbegin()
- read (1)
- write (12, X)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



CPU1

						Status	Tx
1	A					I	0
2	Y					I	0
3	C					I	0
4							

CPU2

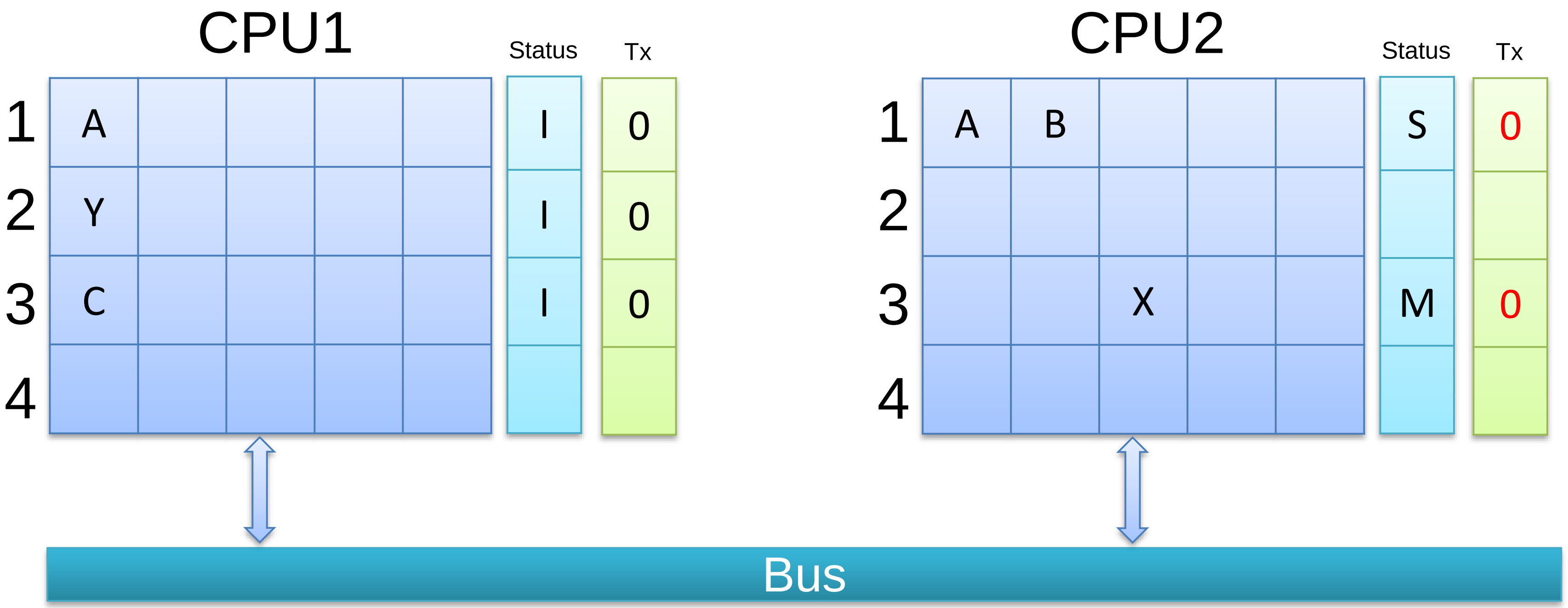
						Status	Tx
1	A	B				S	1
2							
3			X			M	1
4							

- _xbegin()
- read (0)
- read ([illegible]0)
- write ([illegible])

- _xbegin()
- read (1)
- write (12, X)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



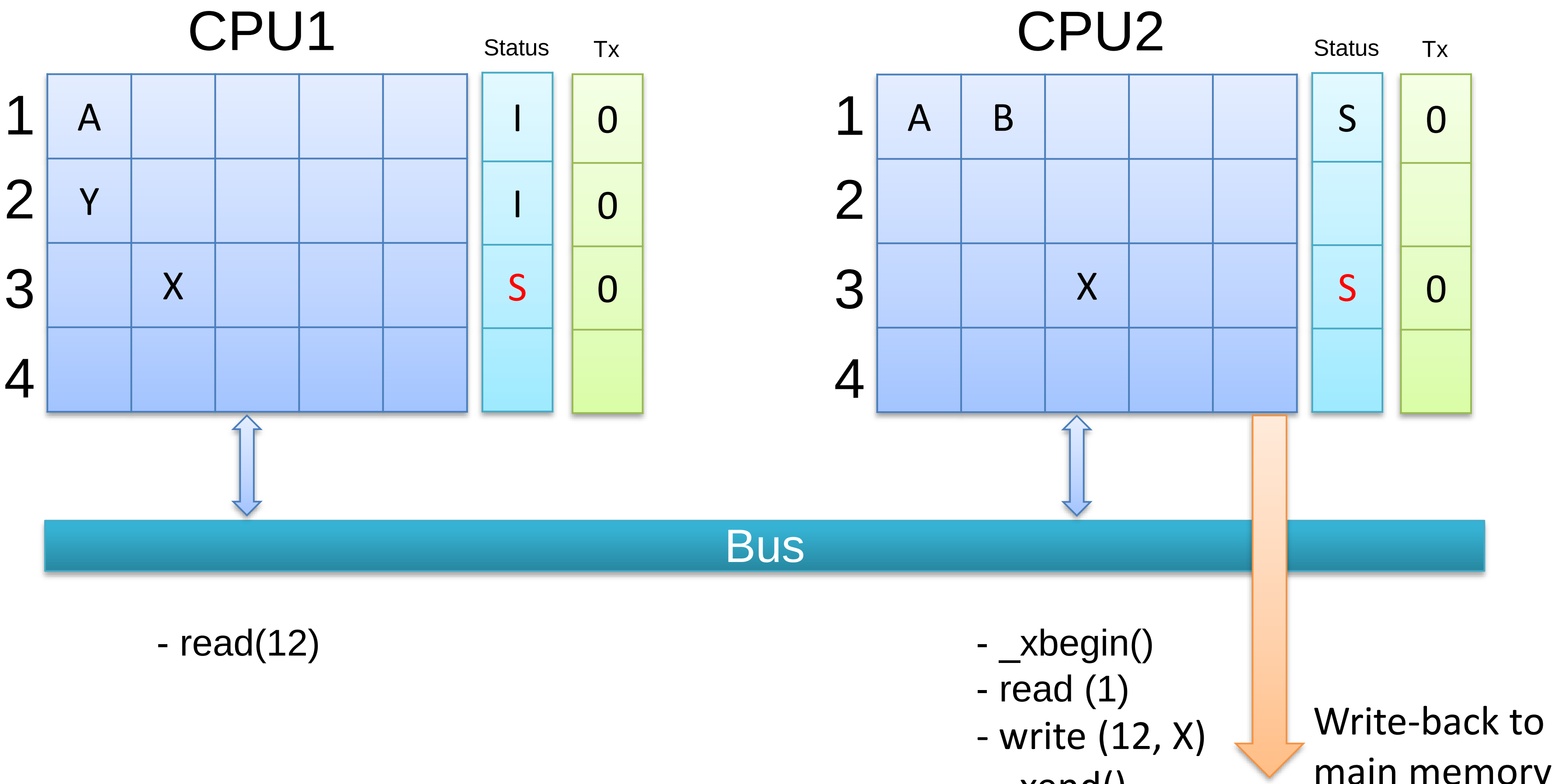
	CPU1					Status	Tx
1	A					I	0
2	Y					I	0
3	C					I	0
4							

	CPU2					Status	Tx
1	A	B				S	0
2							
3			X			M	0
4							

- _xbegin()
- read (1)
- write (12, X)
- _xend()

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



	CPU1					Status	Tx
1	A					I	0
2	Y					I	0
3		X				S	0
4							

	CPU2					Status	Tx
1	A	B				S	0
2							
3			X			S	0
4							

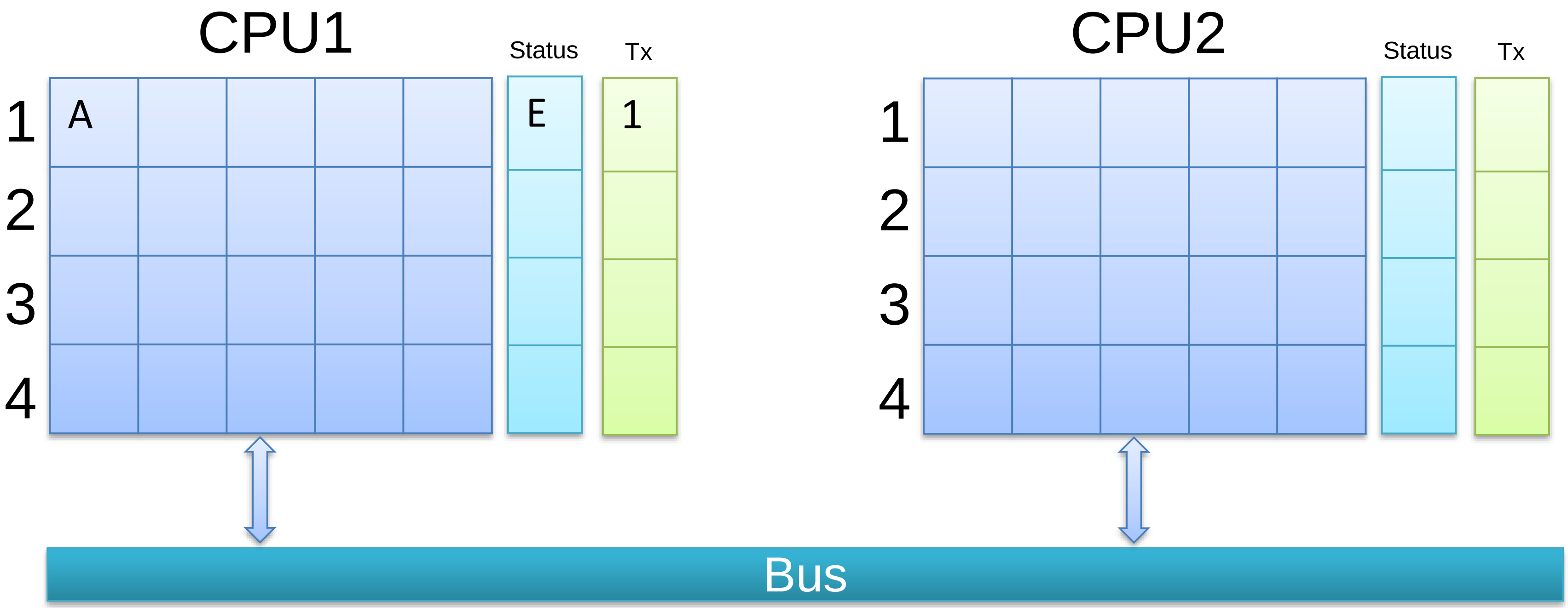
- read(12)

- _xbegin()
- read (1)
- write (12, X)
- _xend()

Write-back to main memory

HTM Limitations

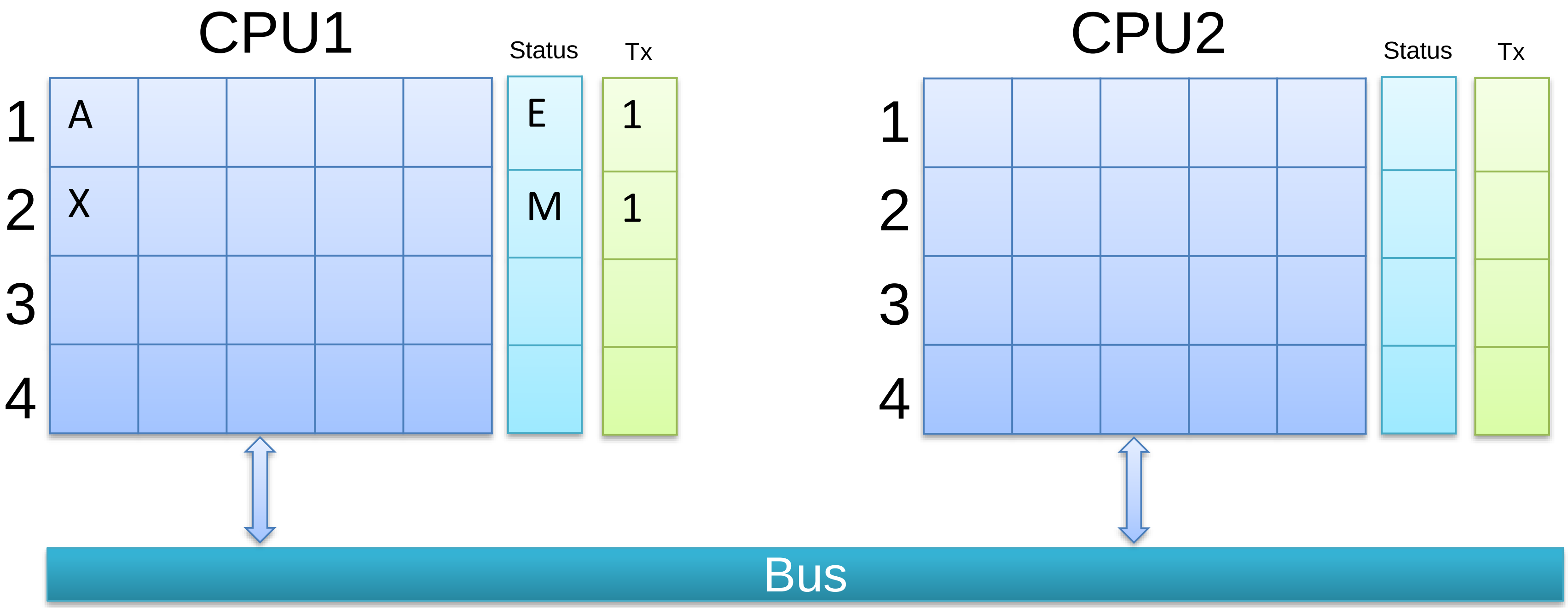
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)

HTM Limitations

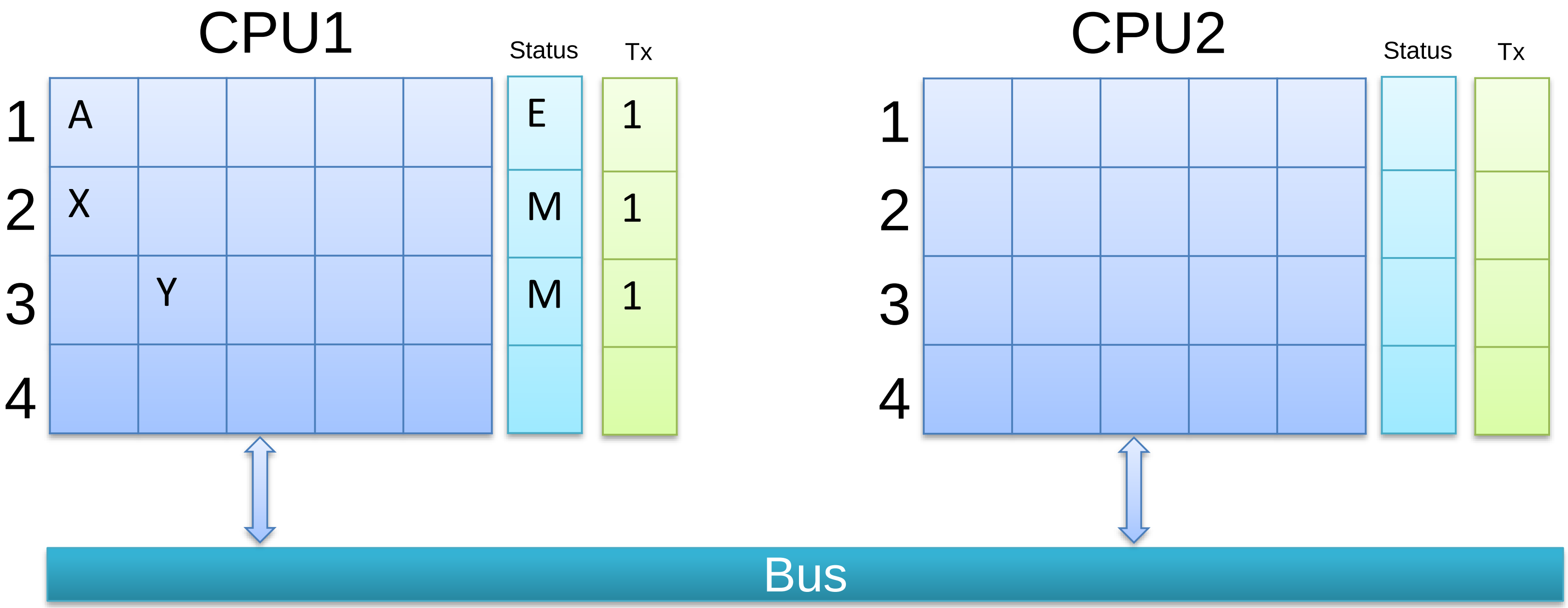
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)
- write (5, X)

HTM Limitations

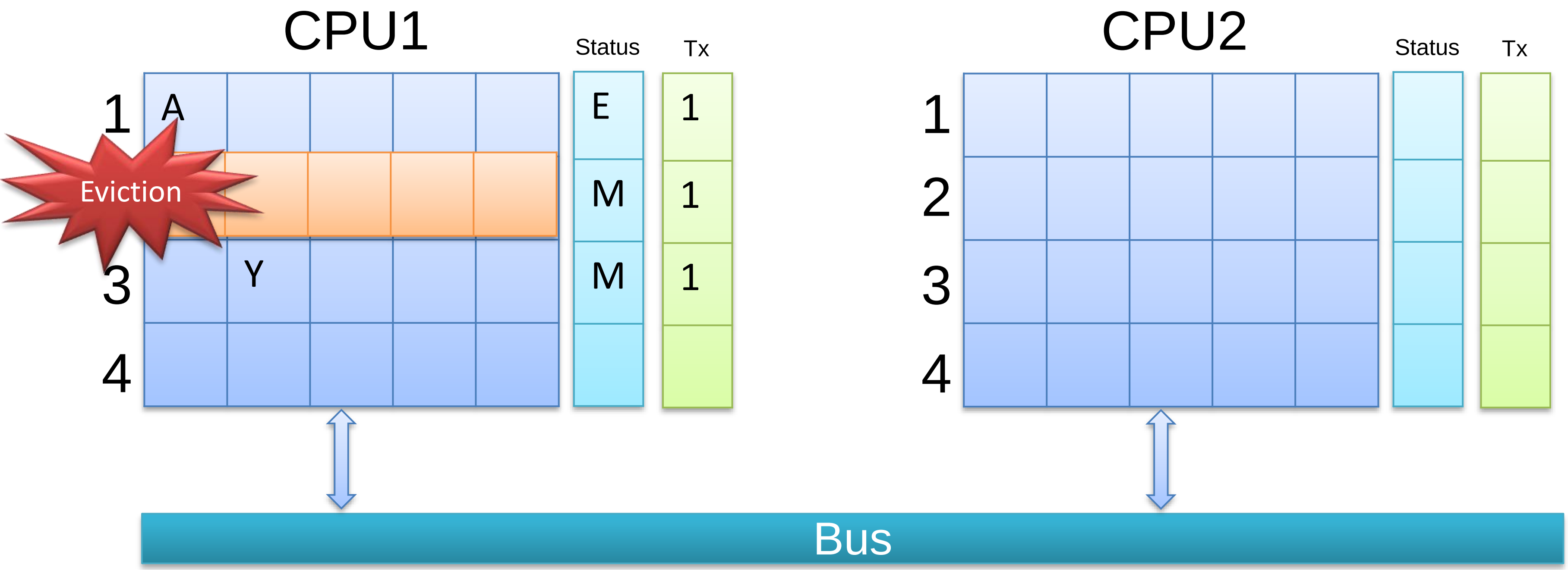
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)
- write (5, X)
- write (11, Y)

HTM Limitations

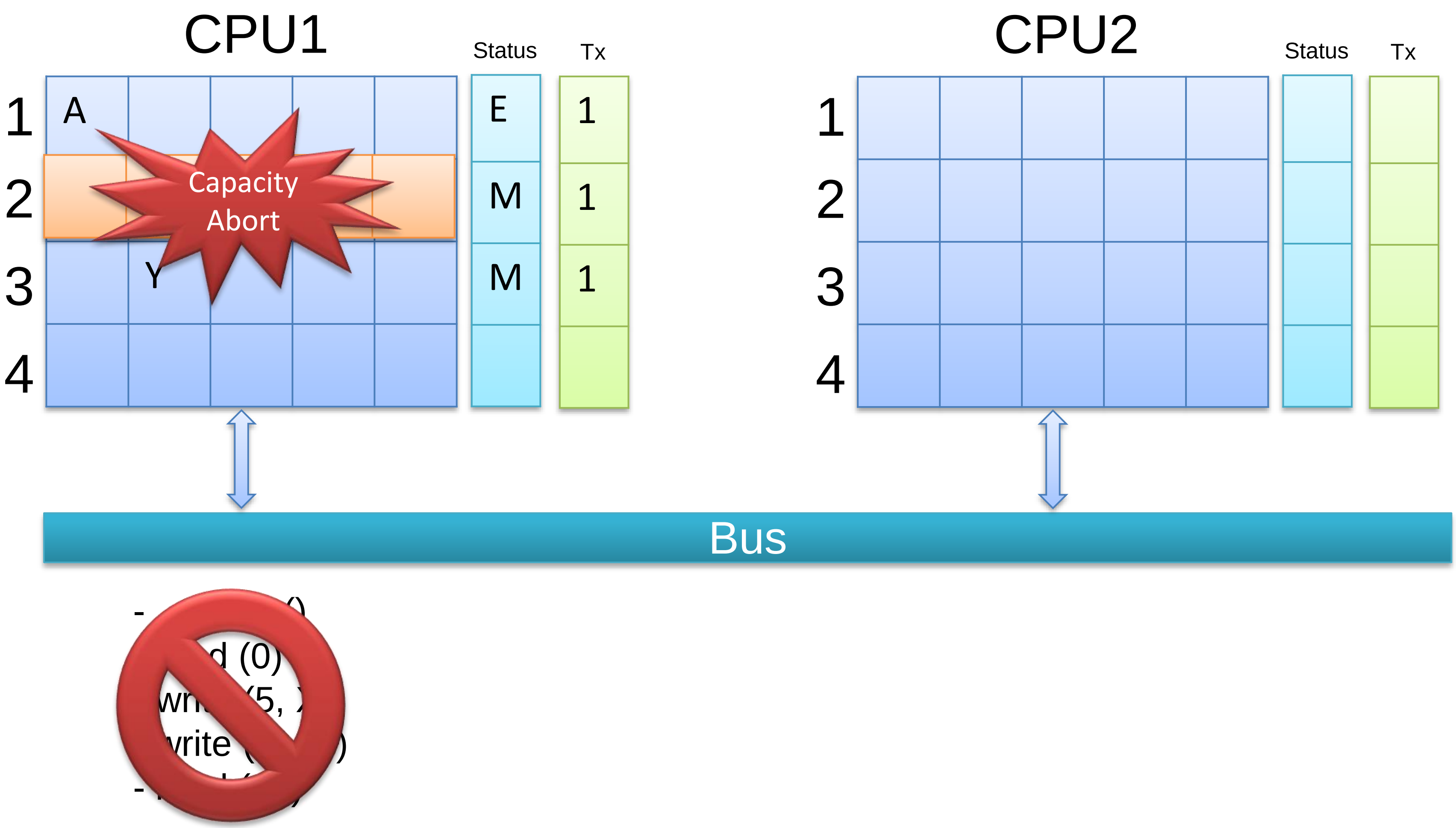
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)
- write (5, X)
- write (11, Y)
- read (**25**)

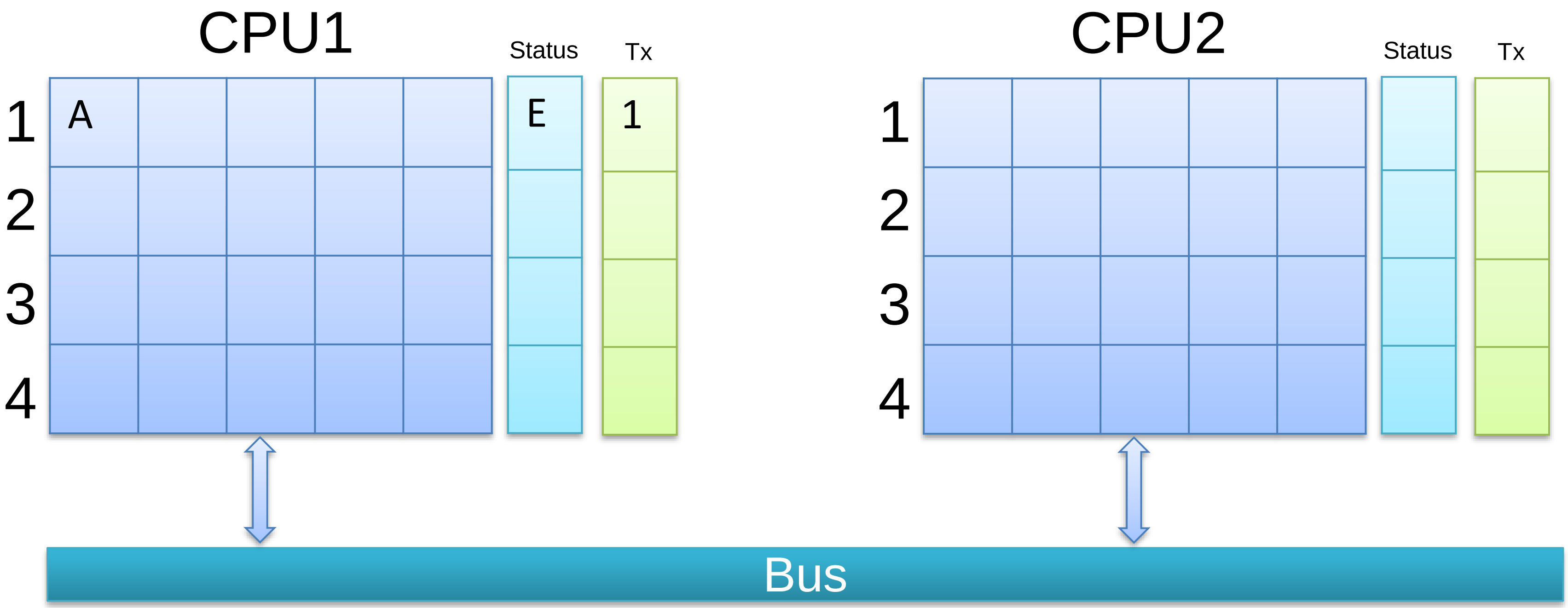
HTM Limitations

Transactional MESI Cache Coherence Protocol



HTM Limitations (2)

Transactional MESI Cache Coherence Protocol

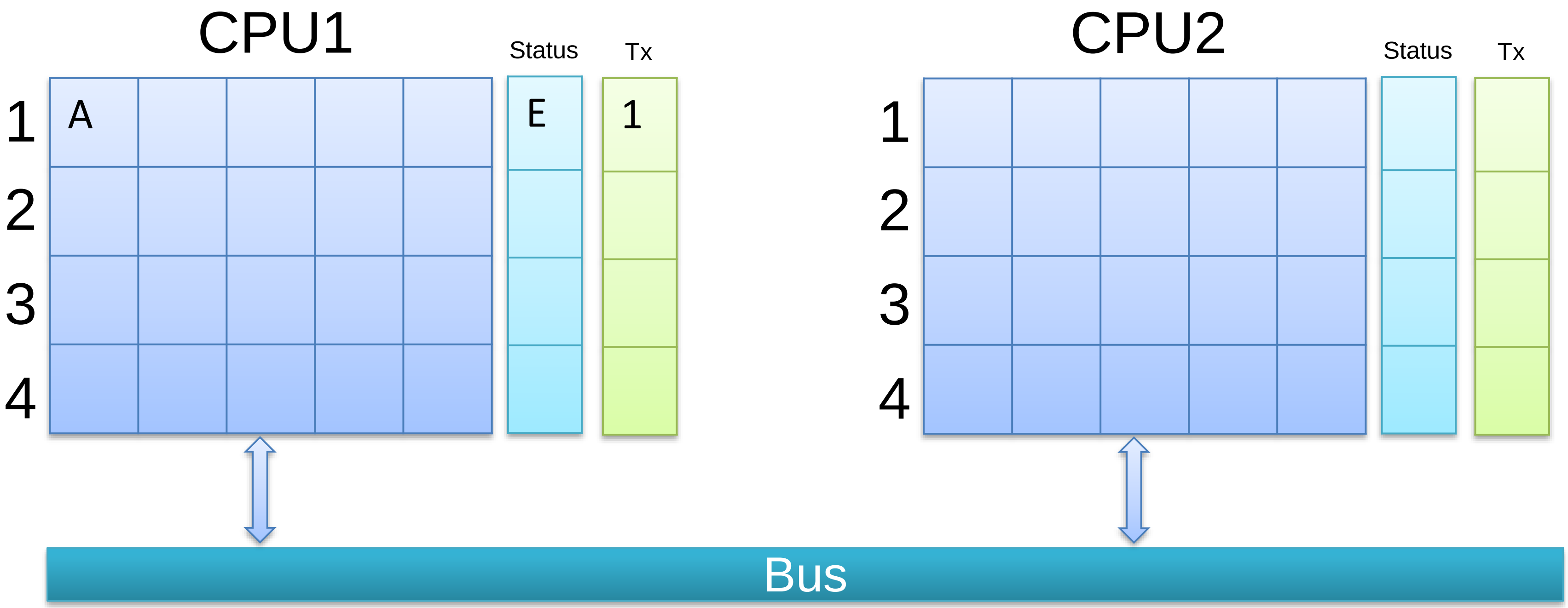


- _xbegin()
- read (0)

Time

HTM Limitations (2)

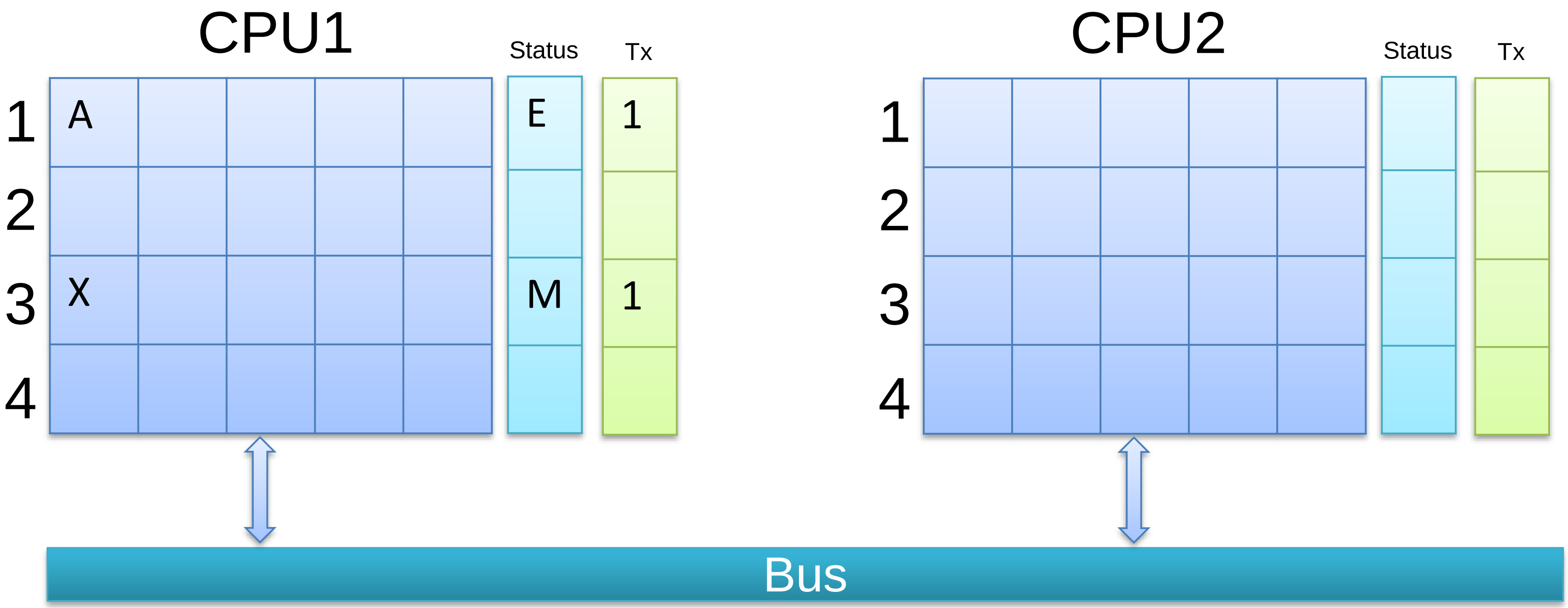
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)
- Do some long work

HTM Limitations (2)

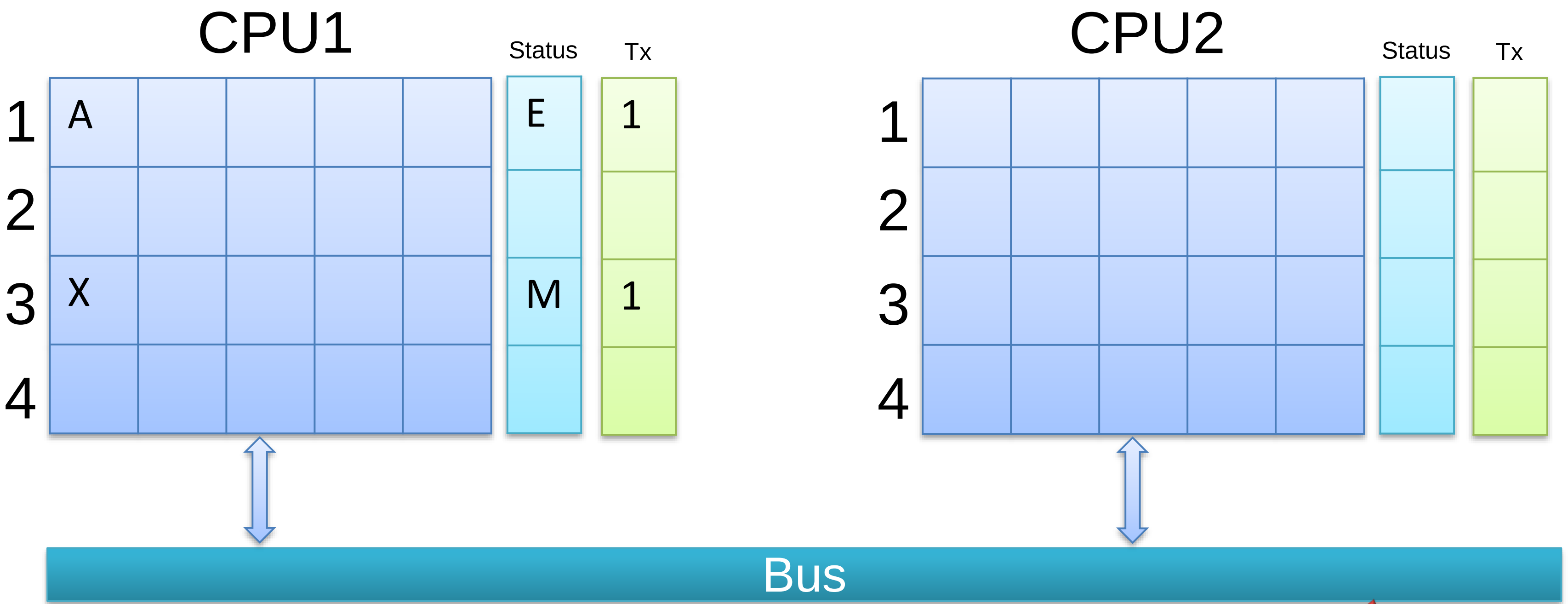
Transactional MESI Cache Coherence Protocol



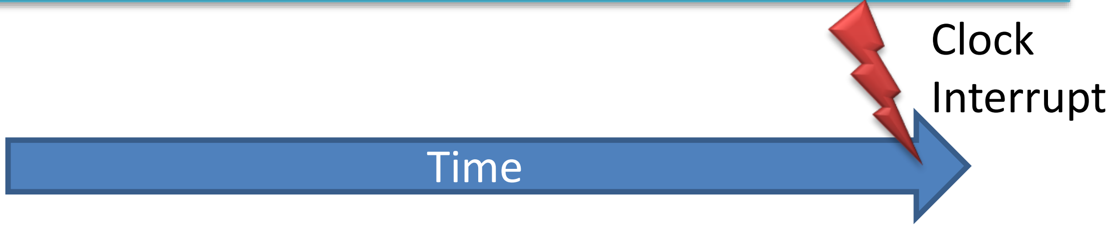
- _xbegin()
- read (0)
- Do some long work
- Write (10, X)

HTM Limitations (2)

Transactional MESI Cache Coherence Protocol

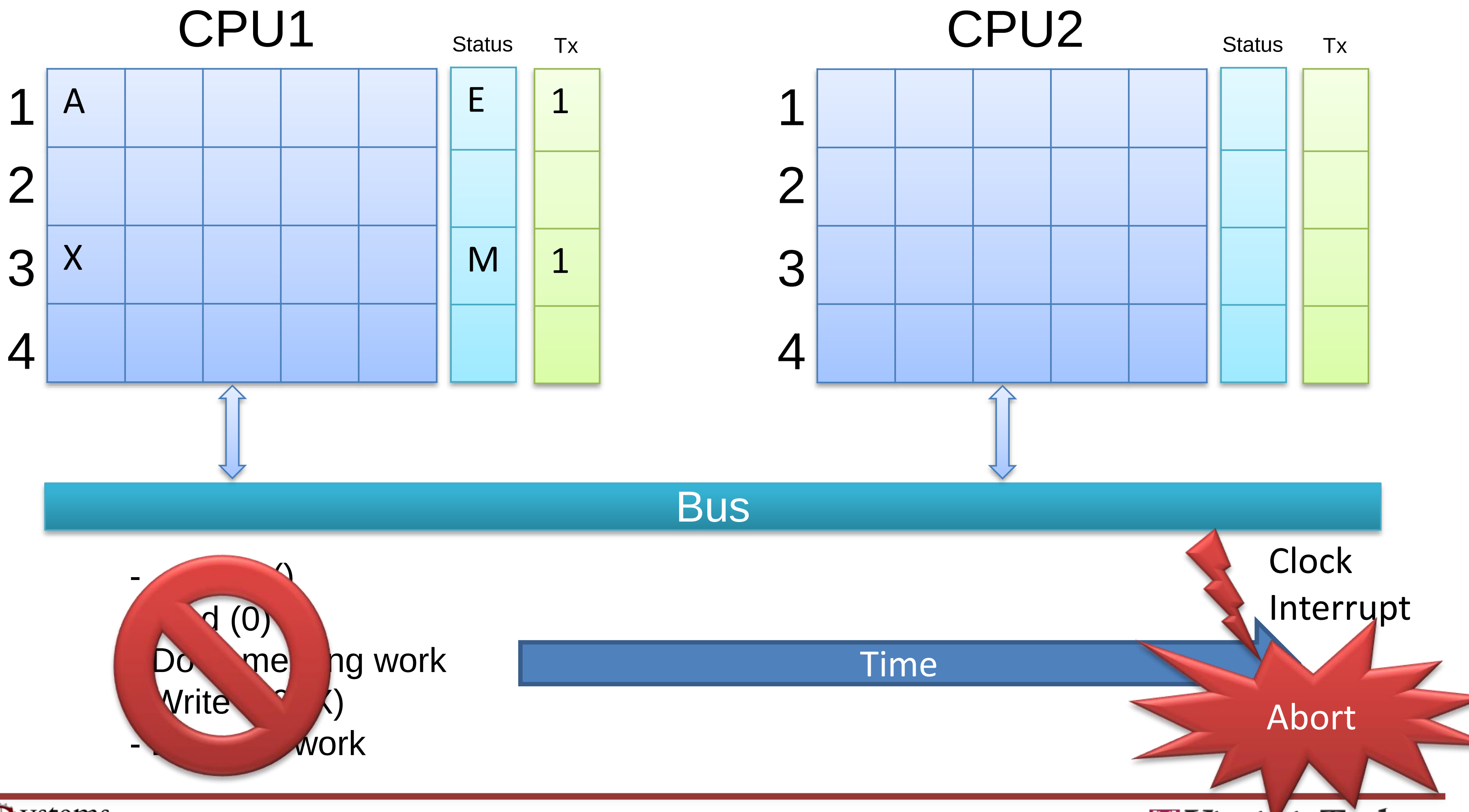


- _xbegin()
- read (0)
- Do some long work
- Write (10, X)
- Do more work



HTM Limitations (2)

Transactional MESI Cache Coherence Protocol

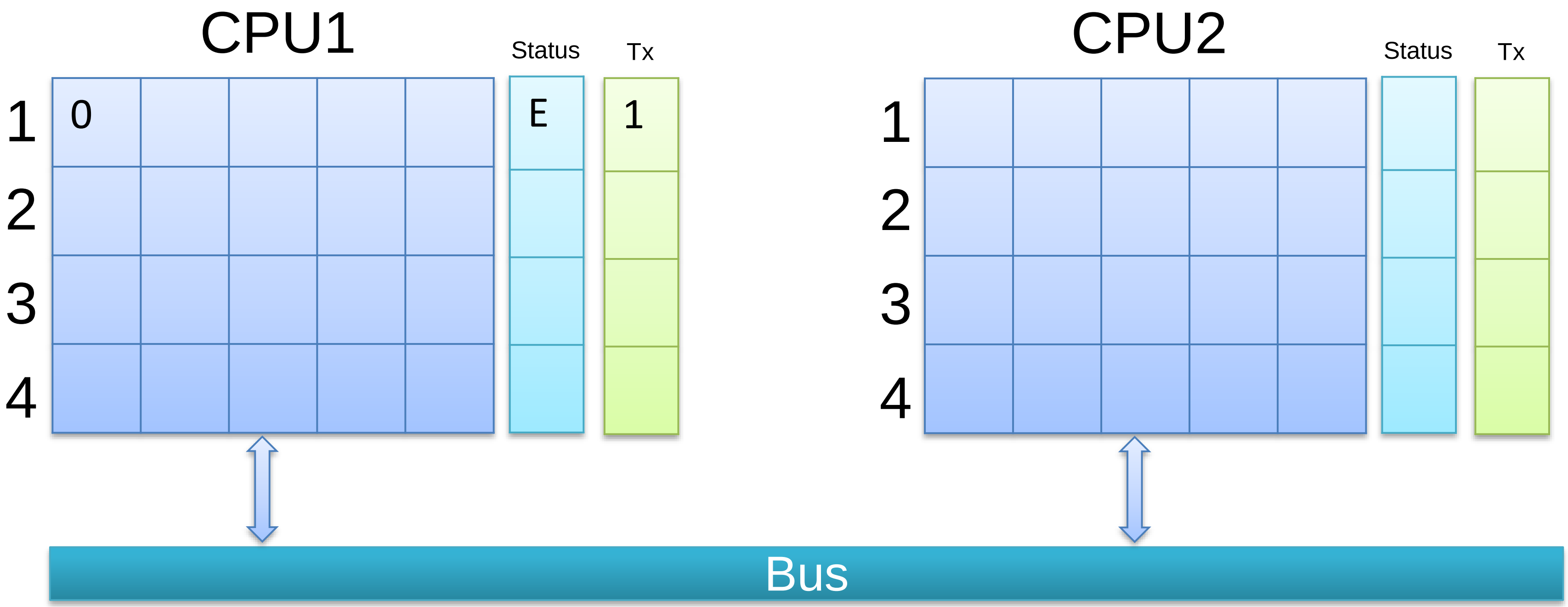


						Status	Tx
1	A					E	1
2							
3	X					M	1
4							

- ()
- d (0)
- Do me ng work
- Write (X)
- work

HTM Limitations (3)

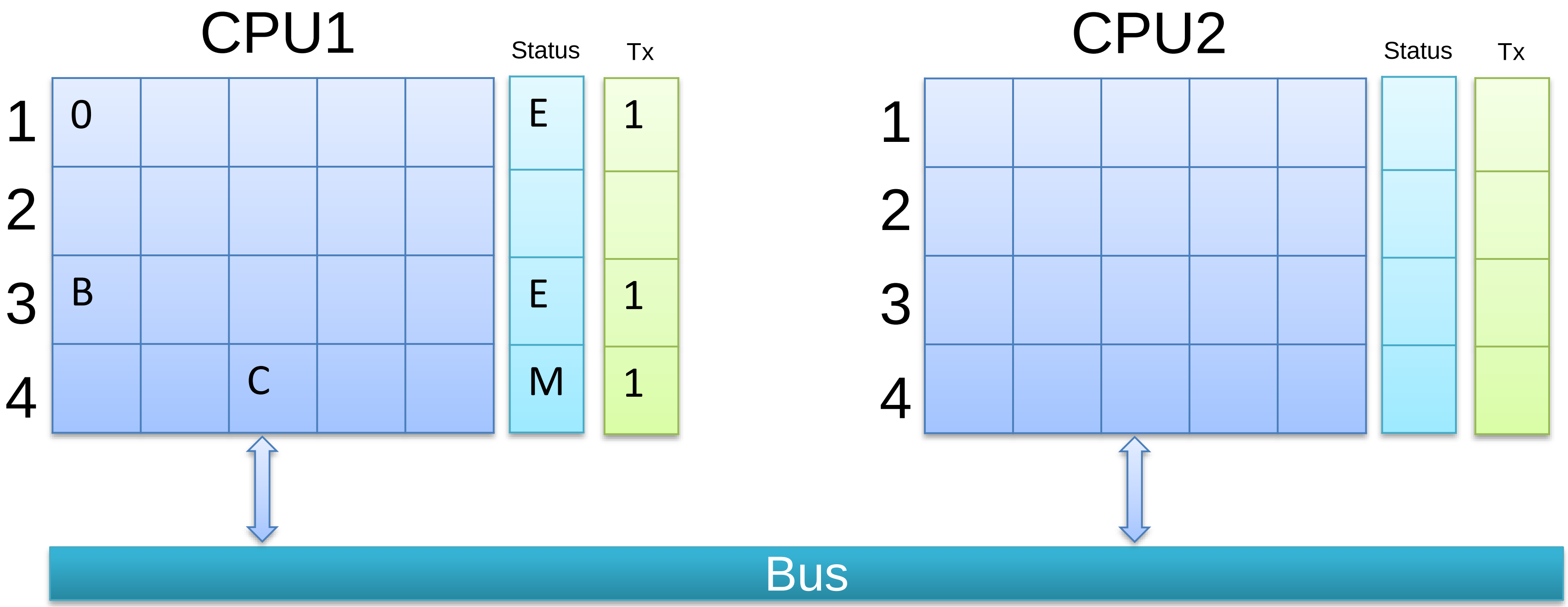
- Must define a software fallback path
  - Default is global lock



- _xbegin()
- if (read(lock)) == 1 then _xabort()

HTM Limitations (3)

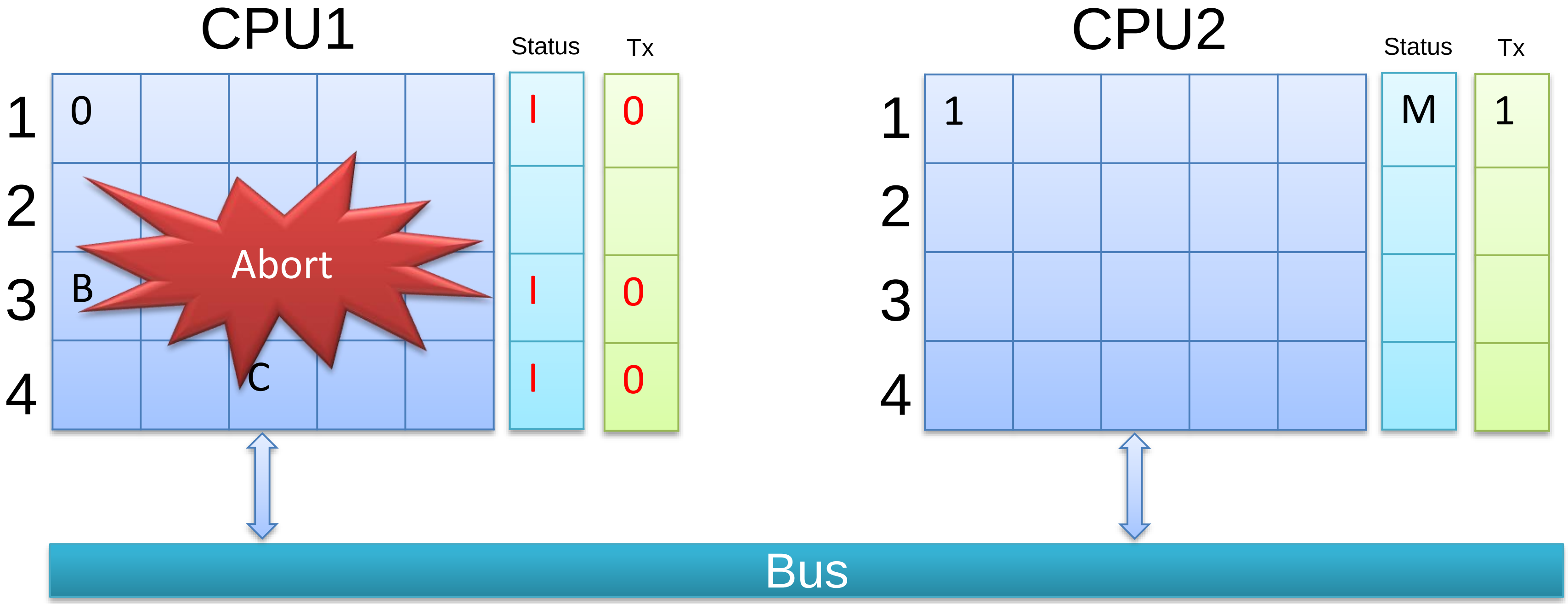
- Must define a software fallback path
  - Default is global lock



- _xbegin()
- if (read(lock)) == 1 then _xabort()
- do some work

HTM Limitations (3)

- Must define a software fallback path
  - – Default is global lock



```
-    _xbegin()
-    if (read(lock)) == 1 then _xabort()
-    do some work
```

```
//non-transactionally
CAS(lock, 0 ,1)
```

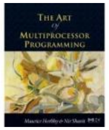
Current HTM

- Best-effort HTM has many limitation
  - Simple hardware design
  - Cache coherence protocol
  - Limited transactional resources
  - Interrupts abort transaction
  - Live-lock
- Transactions are not guaranteed to commit
  - Must provide a software fallback path
    - The standard is to use global-locking
      - Course-grained

Hardware Transactional Memory (HTM)

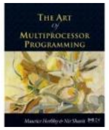
IBM’s Blue Gene/Q & System Z & Power8

Intel’s Haswell TSX extensions



Intel RTM

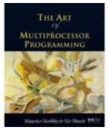
```
if (_xbegin() == _XBEGIN_STARTED) {
   speculative code
   _xend()
} else {
   abort handler
}
```



Intel RTM

```
if (_xbegin() == _XBEGIN_STARTED) {
   speculative code
   _xend()
} else {
   abort handler
}
```

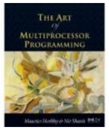
start a speculative transaction



Intel RTM

```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
  _xend()
} else {
  abort handler
}
```

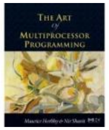
If you see this, you are inside a transaction



Intel RTM

```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
  _xend()
} else {
  abort handler
}
```

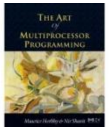
If you see anything else, your transaction aborted



Intel RTM

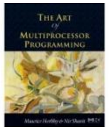
```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
  _xend()
} else {
  abort handler
}
```

you could retry the transaction, or take an alternative path



Abort codes

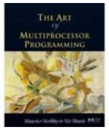
```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
} else if (status & _XABORT_EXPLICIT) {
  aborted by user code
} else if (status & _XABORT_CONFLICT) {
  read-write conflict
} else if (status & _XABORT_CAPACITY) {
  cache overflow
} else {
  …
}
```



Abort codes

```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
} else if (status & _XABORT_EXPLICIT) {
  aborted by user code
} else if (status & _XABORT_CONFLICT) {
  read-write conflict
} else if (status & _XABORT_CAPACITY) {
  cache overflow
} else {
  …
}
```

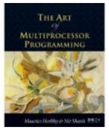
speculative code can call _xabort()



Abort codes

```
if                                 ARTED) {
  s
} else if (status & _XABORT_EXPLICIT) {
  aborted by user code
} else if (status & _XABORT_CONFLICT) {
  read-write conflict
} else if (status & _XABORT_CAPACITY) {
  cache overflow
} else {
  …
}
```

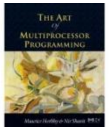
synchronization conflict occurred (maybe retry)



Abort codes

```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
} else i                             ICIT) {
  aborte
} else i                             LICT) {
  read-write conflict
} else if (status & _XABORT_CAPACITY) {
  cache overflow
} else {
  …
}
```

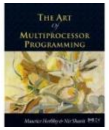
read/write set too big
(maybe don’t retry)



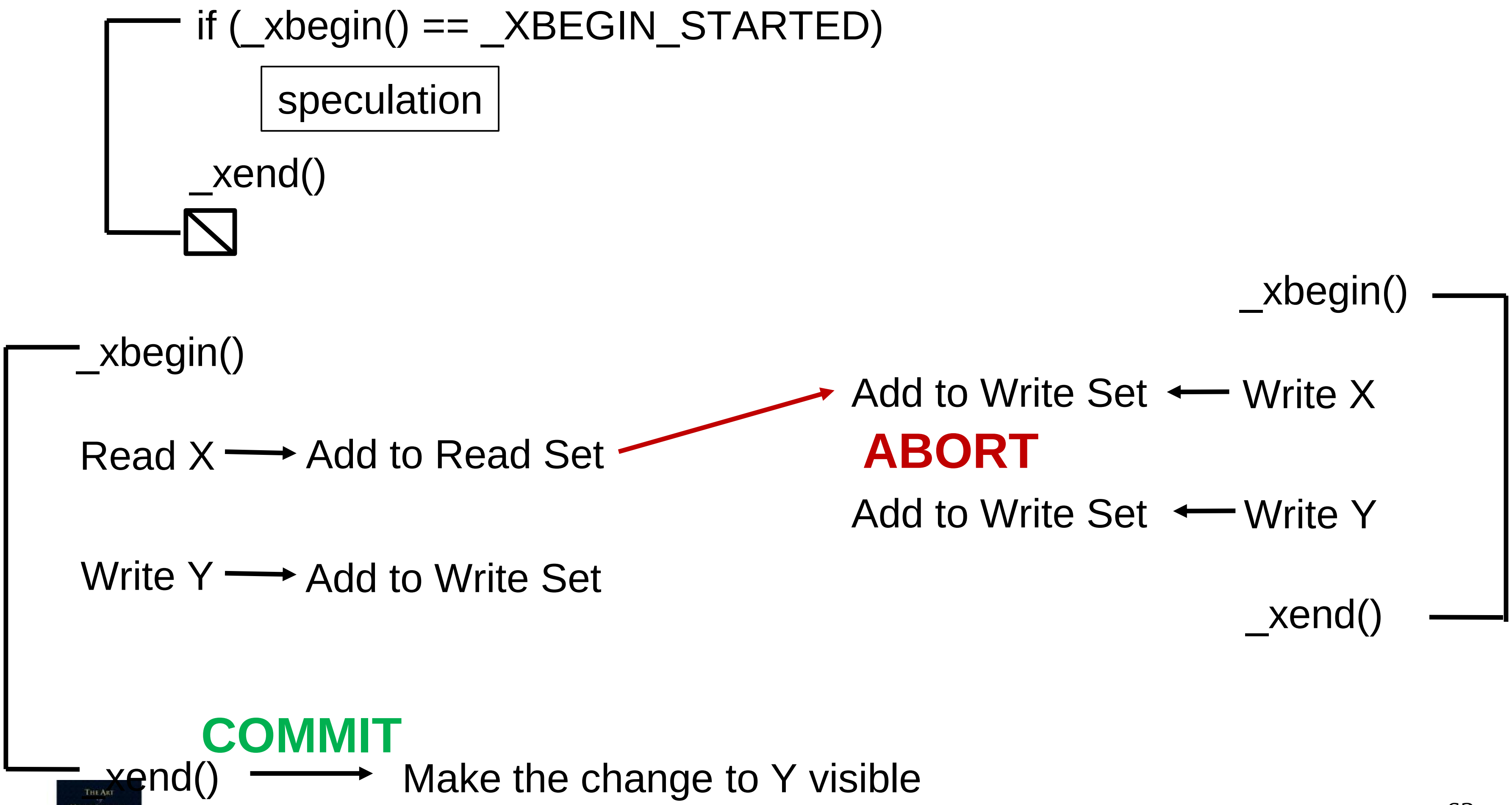
Abort codes

```
if (_xbegin() == _XBEGIN_STARTED) {
  speculative code
} else if (status & _XABORT_EXPLICIT) {
  aborted by user code
} el                                   NFLICT) {
  read write conflict
} else if (status & _XABORT_CAPACITY) {
  cache overflow
} else {
  …
}
```

other abort codes …



RTM Execution



if (_xbegin() == _XBEGIN_STARTED)

speculation

_xend()

_xbegin()

Read X → Add to Read Set

Write Y → Add to Write Set

COMMIT

_xend() → Make the change to Y visible

_xbegin()

Write X → Add to Write Set

ABORT

Write Y → Add to Write Set

_xend()

RTM

Small & Medium Transactions

Best-effort

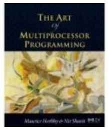
Conflicts

Overflow

Unsupported Instructions

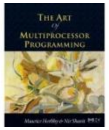
Interrupts

Needs software fallback



Non-Speculative Fallback

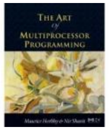
```
if (_xbegin() == _XBEGIN_STARTED) {
  read lock state
  if (lock taken) _xabort();
  work;
  _xend()
} else {
  lock->lock();
  work;
  lock->unlock();
}
```



Non-Speculative Fallback

```
if (_xbegin() == _XBEGIN_STARTED) {
  read lock state
  if (lock taken) _xabort();
  work;
  _xend()
} e
  i
  w
  i
}
```

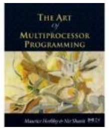
reading lock ensures that transaction will abort if another thread acquires lock



Non-Speculative Fallback

```
if (_xbegin() == _XBEGIN_STARTED) {
  read lock state
  if (lock taken) _xabort();
  work;
  _xend()
} else {
  lo
  wo
  lo
}
```

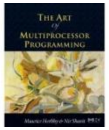
abort if another thread has acquired lock



Non-Speculative Fallback

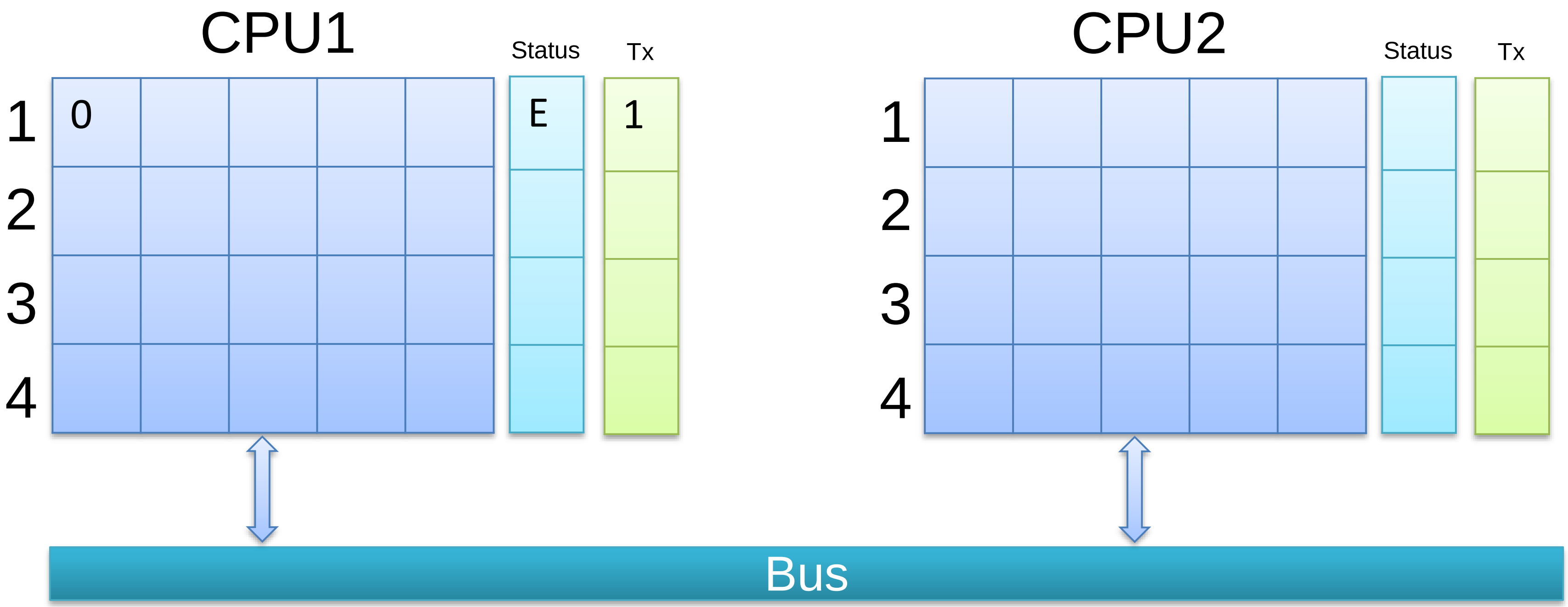
on abort, acquire lock & do work (aborting concurrent speculative transactions)

```
if
  re
  if (lock taken) _xabort();
  work;
  _xend()
} else {
  lock->lock();
  work;
  lock->unlock();
}
```



Global Lock Fallback

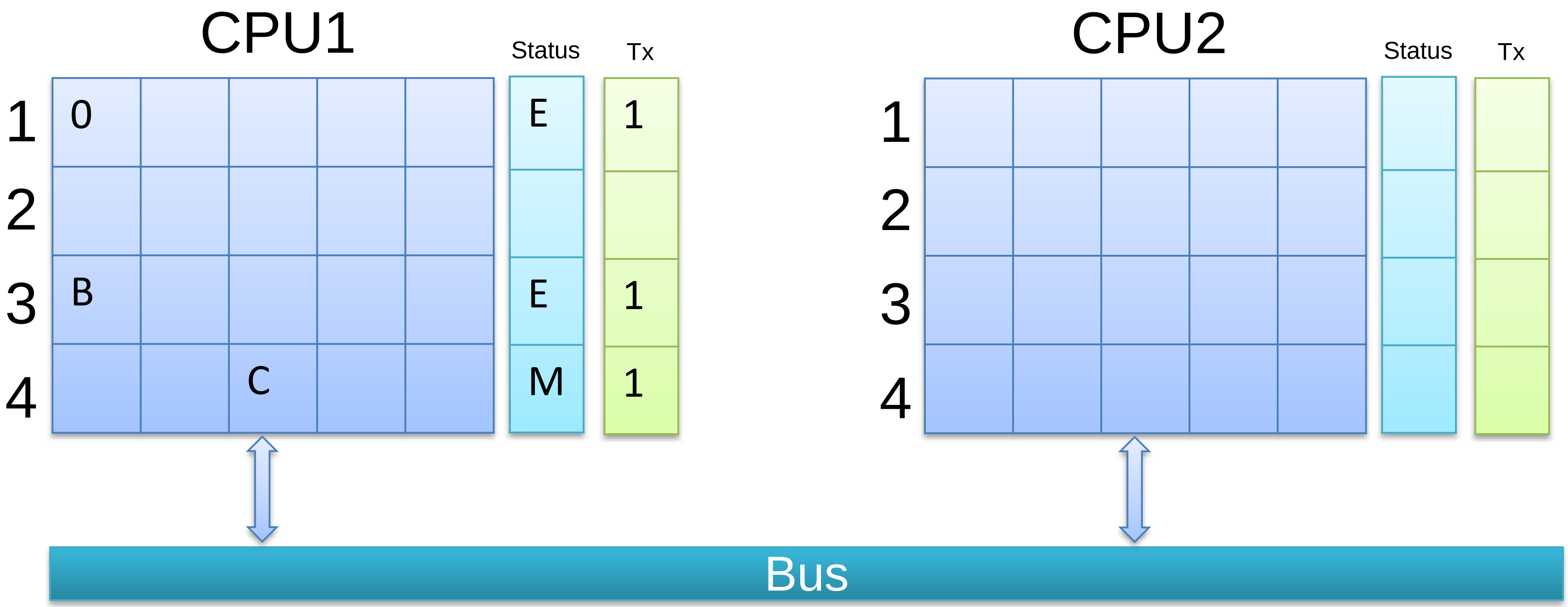
- Must define a software fallback path
  - Default is global lock



- _xbegin()
- if (read(lock)) == 1 then _xabort()

Global Lock Fallback

- Must define a software fallback path
  - Default is global lock



CPU1

						Status	Tx
1	0					E	1
2							
3	B					E	1
4			C			M	1

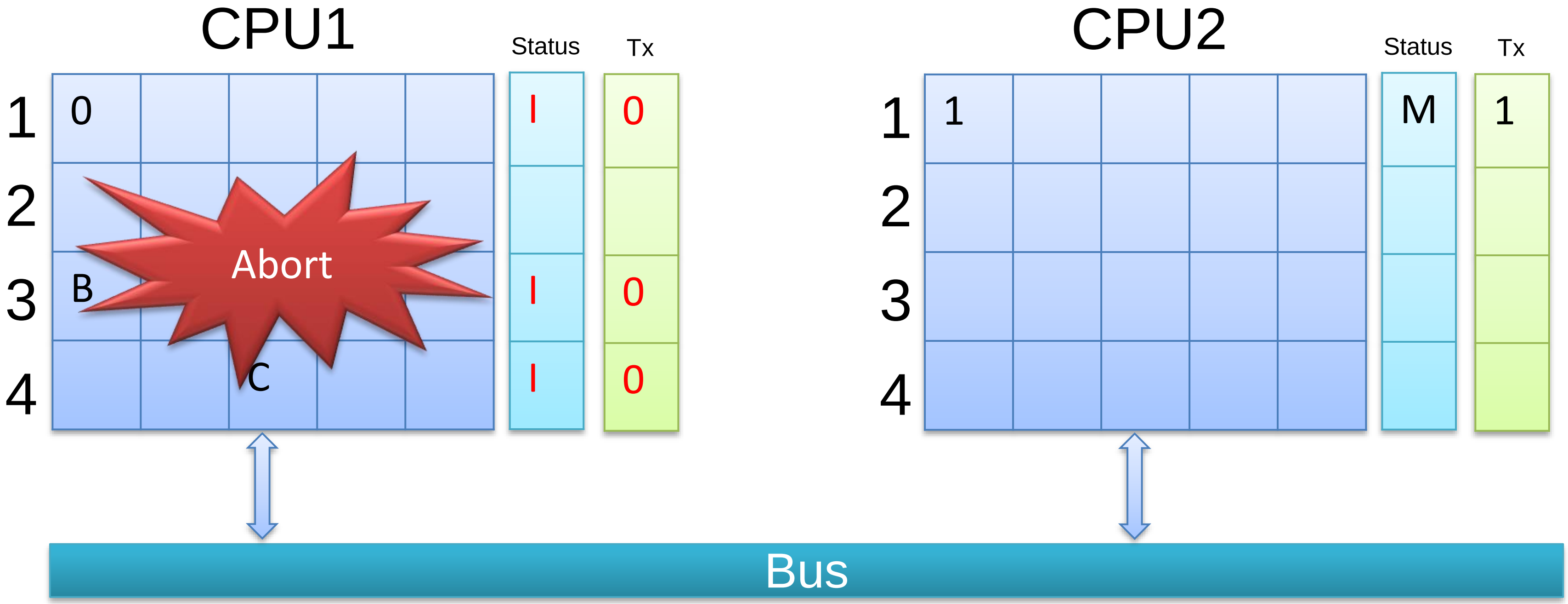
CPU2

						Status	Tx
1							
2							
3							
4							

```
-    _xbegin()
-    if (read(lock)) == 1 then _xabort()
-    do some work
```

Global Lock Fallback

- Must define a software fallback path
  - Default is global lock



```
-    _xbegin()
-    if (read(lock)) == 1 then _xabort()
-    do some work
```

```
//non-transactionally
CAS(lock, 0 ,1)
```

Lock Elision

<HLE_Aquire_Prefix> Lock(L)

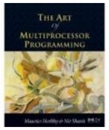
Atomic region executed as a ***transaction*** or ***mutually exclusive on L***

<HLE_Release_Prefix> Release(L)

Execute optimistically, without any locks

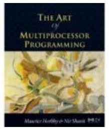
Track Read and Write Sets

Abort on memory conflict: rollback acquire lock



Lock Elision

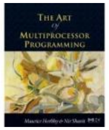
```
<HLE acquire prefix> lock();
do work;
<HLE release prefix> unlock()
```



Lock Elision

```
<HLE acquire prefix> lock();
do work;
<HLE release prefix> unlock()
```

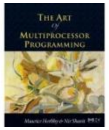
first time around, read lock and execute speculatively



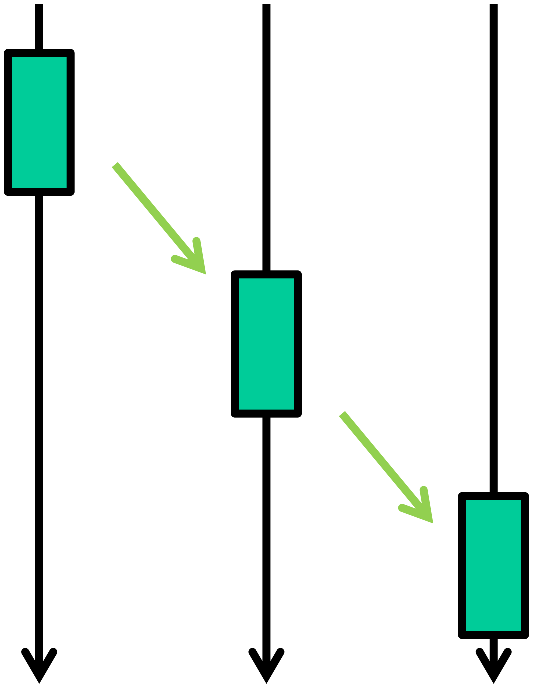
Lock Elision

```
<HLE acquire prefix> lock();
do work;
<HLE release prefix> unlock()
```

if speculation fails, no more Mr. Nice Guy, acquire the lock



Conventional Locks

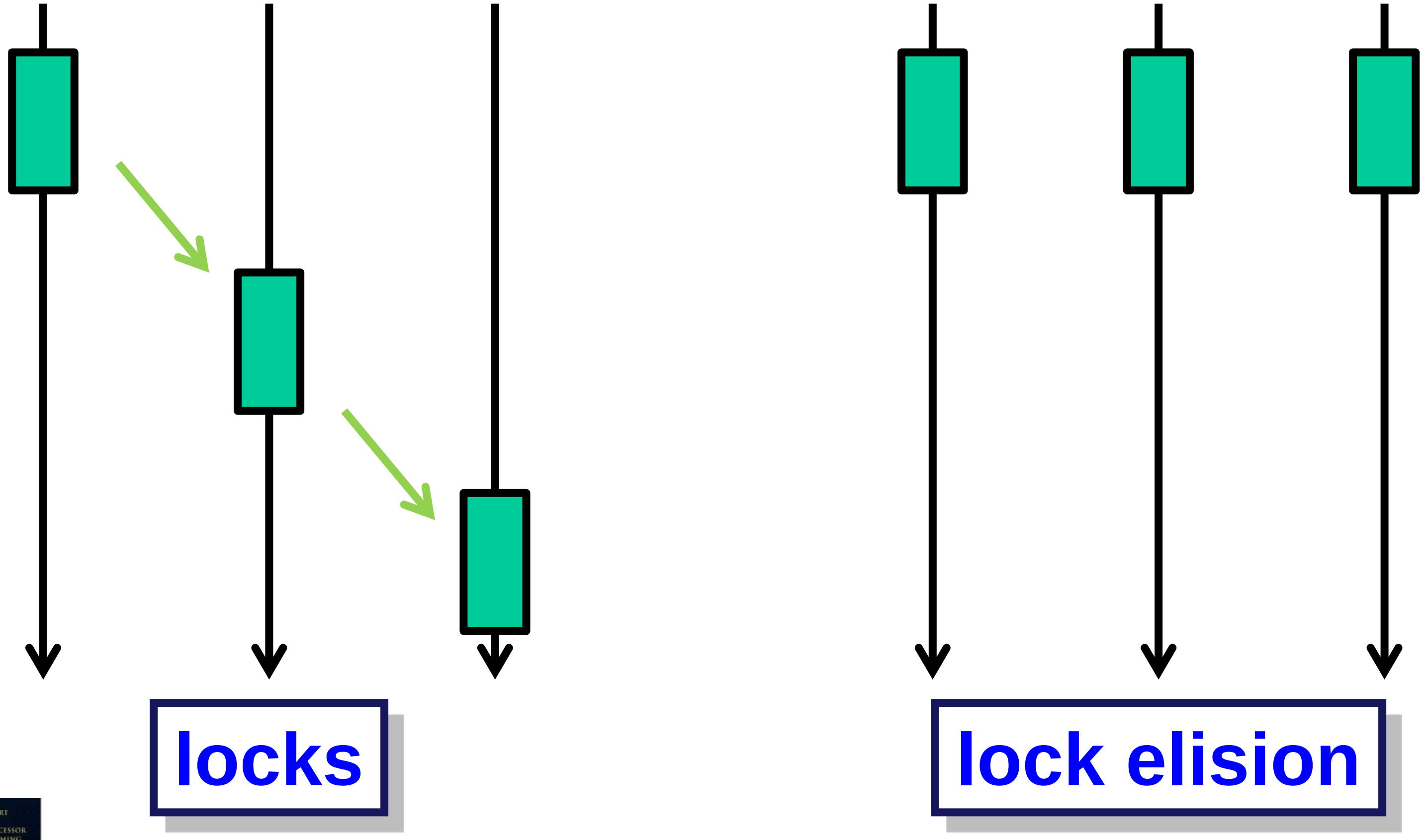


lock transfer latencies

serialized execution

locks

Lock Elision

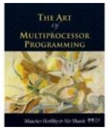


locks

lock elision

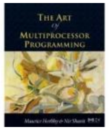
Not all and

- Limits to
  - – Transactional cache size
  - – Scheduling quantum
- Transaction cannot commit if it is
  - – Too big
  - – Too slow
  - – Actual limits platform-dependent



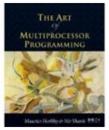
HTM Strengths & Weaknesses

- Ideal for lock-free data structures



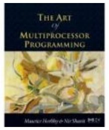
HTM Strengths & Weaknesses

- Ideal for lock-free data structures
- Practical proposals have limits on
  - Transaction size and length
  - Bounded HW resources
  - Guarantees vs best-effort



HTM Strengths & Weaknesses

- Ideal for lock-free data structures
- Practical proposals have limits on
  - Transaction size and length
  - Bounded HW resources
  - Guarantees vs best-effort
- On fail
  - Diagnostics essential
  - Retry in software?



HTM benefits

- Strong Atomicity
  - Conflicts between transactional and non-transactional code is detected
  - STM systems usually do not support strong atomicity
    - It is call Weak Atomicity
      - Only conflicts between transactions are detected
    - Very expensive to support Strong Atomicity via software
      - E.g., use a transaction of a single access in non-transactional code

HTM benefits

```
tx_begin()

tx_read(x)



tx_commit()
```

```
x = 10; //non-transactional code write
```

HTM benefits

```
tx_begin()

tx_read(x)






tx_commit()
```

```





x = 10; //non-transactional code write
```

```
tx_begin()

tx_read(x)






tx_commit()
```

```


//non-transactional code write via a
single access transaction
tx_begin()
tx_write(x, 10)
tx_commit()
```

HTM benefits

- Extremely fast compared to STM
- Implicit
  - No need for explicit tx_read and tx_write
    - Drawbacks
      - All accesses are transactional
      - Consume from the limited HTM resources
- Can be used to simplify many lock-free algorithms
  - E.g., can be used to implement Multi-CAS