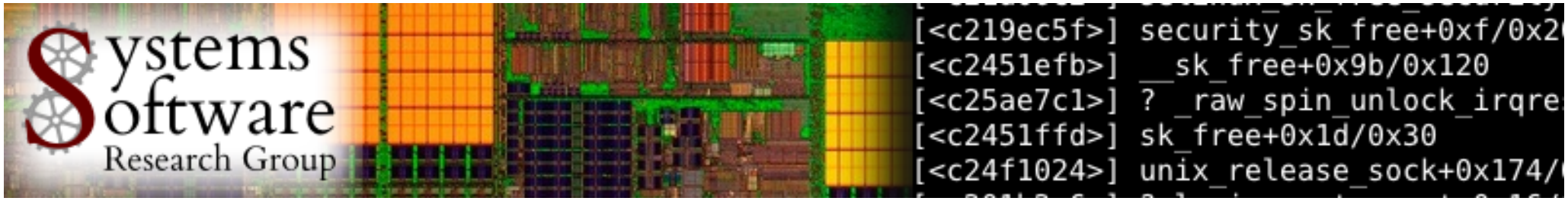


On Optimizing Transactional Memory: Transaction Splitting, Scheduling, Fine-grained Fallback, and NUMA Optimization

Mohamed Mohamedin (PhD Defense)

Binoy Ravindran, Chair

Leyla Nazhandali

Mohamed Rizk

Paul Plassmann

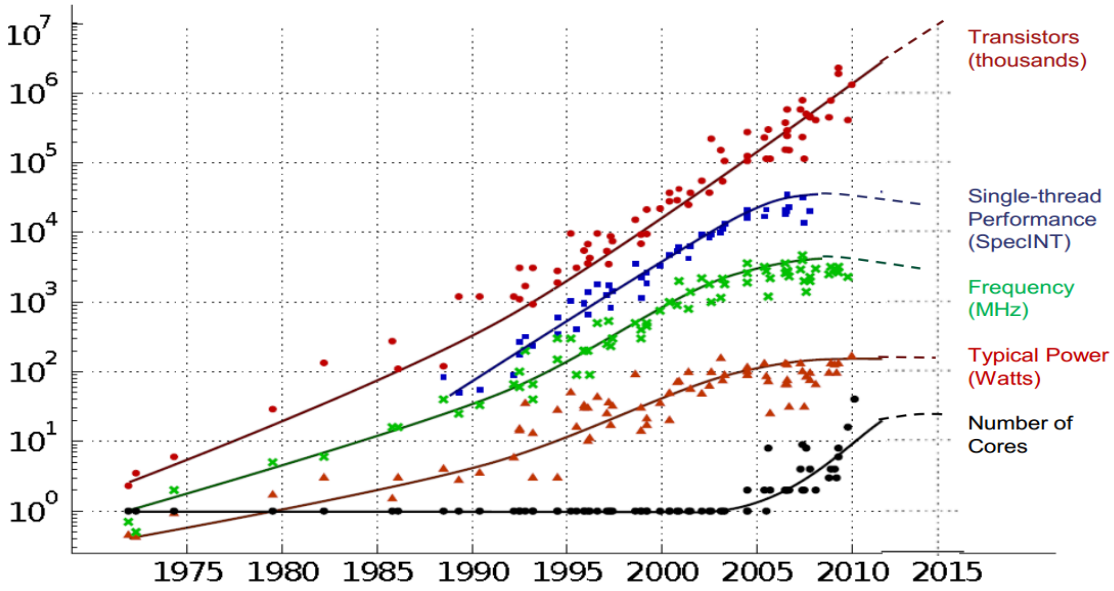
Robert P. Broadwater

Roberto Palmieri

July 30, 2015

Multi-core Architectures

- Current trend
  - Power wall
  - All devices
  - The free lunch is over
  - Parallelism



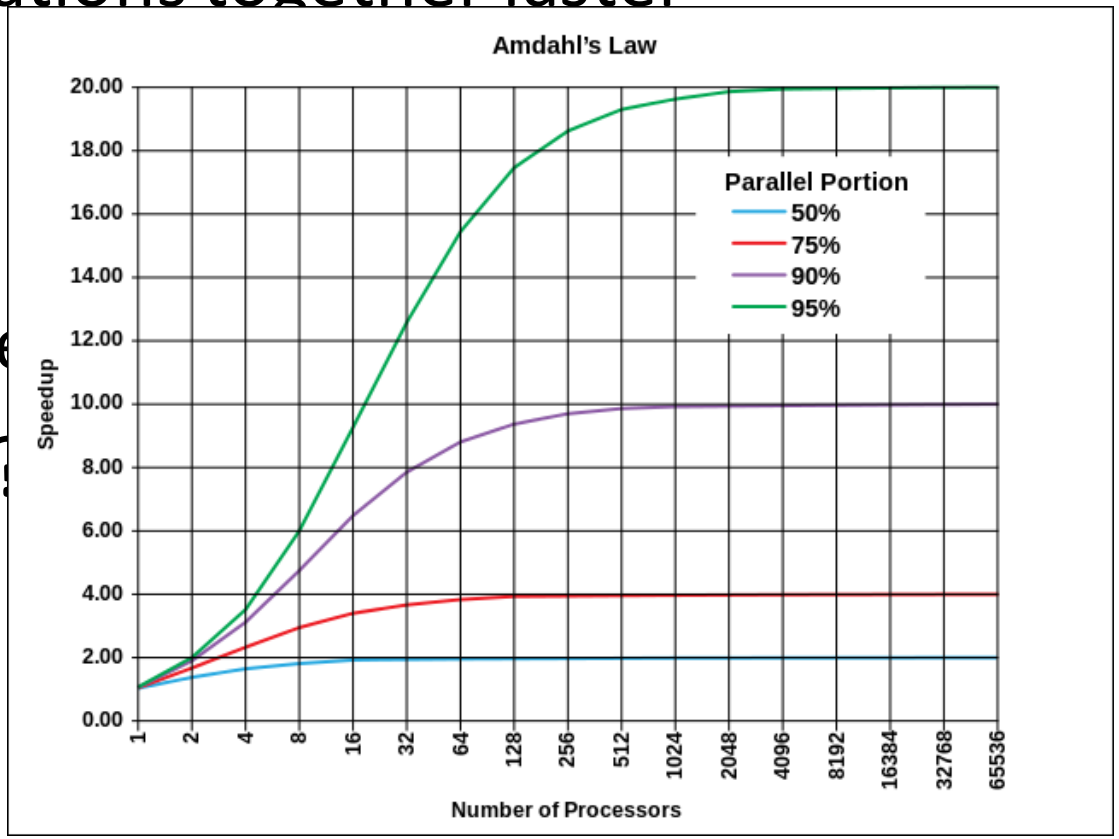
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Current Multi-core Architectures

- Chip Multi-core
  - Multiple cores on the same chip
    - E.g., 4 cores are common, up to 18 cores in Haswell-EX
  - Fast communication between cores
- Multi-socket
  - The motherboard has multiple CPU sockets.
    - Started with one core per socket
    - Now, multi-socket-multi-core
    - Memory access time (UMA, NUMA)
    - Inter-socket communication

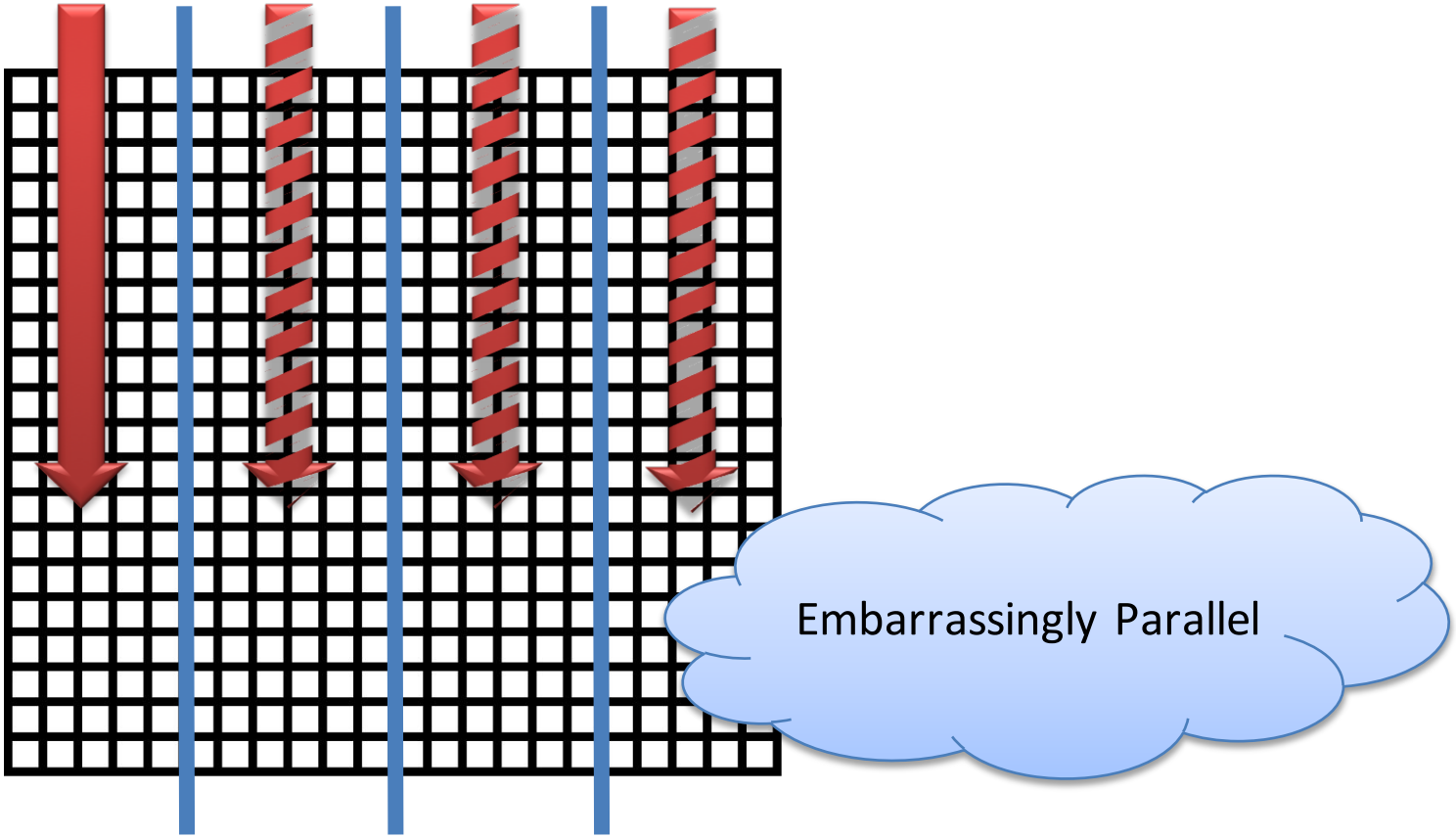
Benefits of Multi-core

- Real parallelism
  - Can run more jobs concurrently
  - Run several applications together faster
- Scalable
  - More jobs can use
  - NUMA architecture
- Is there a problem?
  - Data sharing
  - Software does not



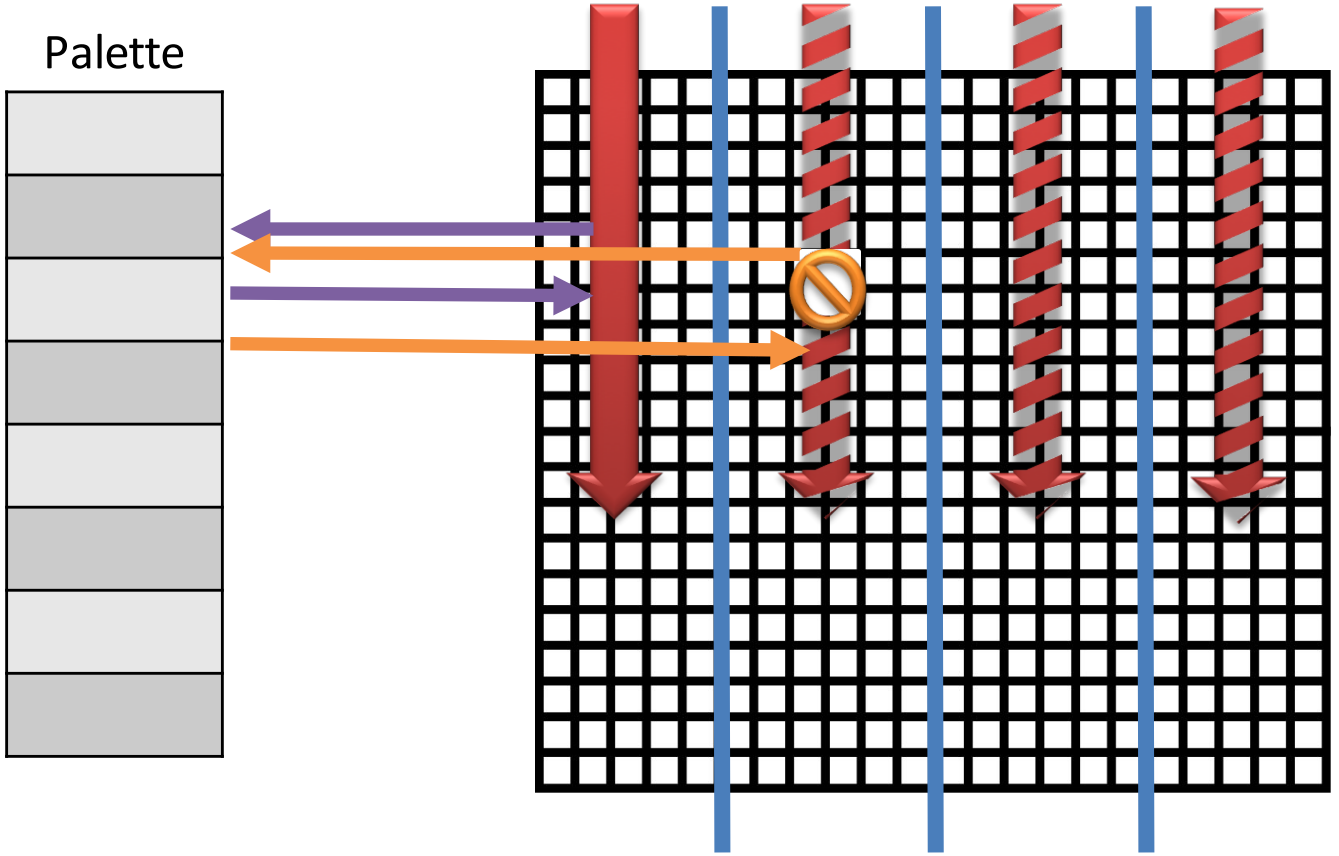
Data Sharing

- Convert a color image to grayscale
  - For each pixel, gray level = $\frac{Red+Green+Blue}{3}$



Data Sharing (2)

- Color quantization: Reduce number of distinct colors in an image (palette)

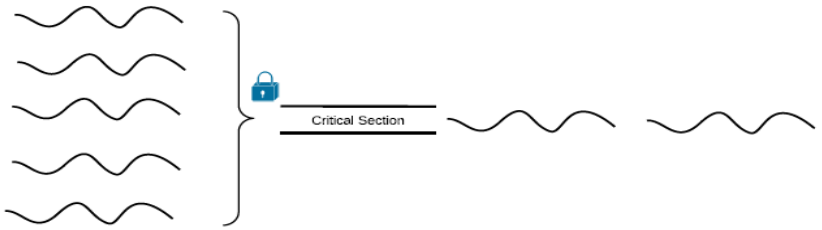


How to Exploit Parallelism?

- Data sharing → Synchronization
- Synchronization → More sequential bottleneck
- Synchronization techniques
  - Locking
    - Fine-grained locking
    - Course-grained locking
  - Transactional Memory

Lock-based Synchronization

- Coarse-grained locking
  - Easy to program
  - Limited concurrency
- Fine-gained locking
  - Hard to program
  - Algorithm specific
  - Locks issues
    - Deadlock
    - Live-lock
    - Lock-convoying
    - Priority inversion
    - Non-composable




```
public boolean add(int item) {
  Node pred, curr;
  lock.lock();
  try {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
  } finally {
    lock.unlock();
  }
}
```

```
public boolean add(int item) {
  head.lock();
  Node pred = head;
  try {
    Node curr = pred.next;
    curr.lock();
    try {
      while (curr.val < item) {
        pred.unlock();
        pred = curr;
        curr = curr.next;
        curr.lock();
      }
      if (curr.key == key) {
        return false;
      }
      Node newNode = new Node(item);
      newNode.next = curr;
      pred.next = newNode;
      return true;
    } finally {
      curr.unlock();
    }
  } finally {
    pred.unlock();
  }
}
```

- Writing fast and correct parallel code is hard

Transactional Memory (TM)

- New easier synchronization abstraction
- Easy to program
  - As easy as coarse-grained locking
- Idea
  - Run transactions concurrently
  - Serialize when there is a conflict
  - Synchronization handled by TM
  - Transparent to the programmer
- Performance comparable to fine-grained locking
- Composable
- TM Classification
  - Software TM (STM)
  - Hardware TM (HTM)
  - Hybrid TM (HyTM)

```
public boolean add(int item) {
  Node pred, curr;
  atomic {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
}
```

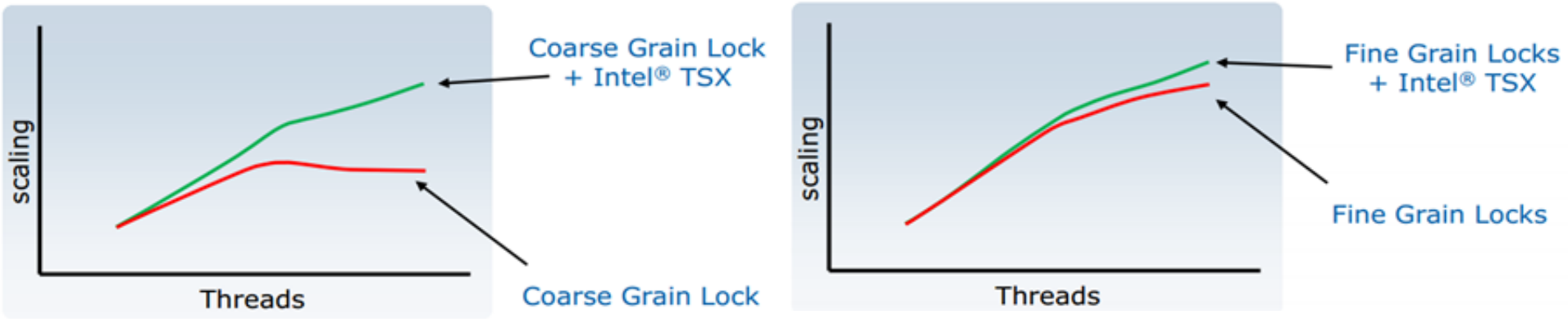
Research on TM

- Extensively studied for the past decade
- Research focus → STM
  - No special transactional hardware
  - Requires extensive instrumentation
    - Monitor each transactional access
    - Detect conflicts
- Past research on HTM and HyTM
  - Simulation
  - Different assumptions
  - Some proposals require complex hardware
  - No agreement until …

```
public boolean add(int item) {
  Node pred, curr;
  atomic {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
}
```

Commercial HTM

- Very recent (in the last 2 years)
  - IBM (Blue Gene/Q and Power 8)
  - Commodity processor (Intel Haswell)
- Best TM performance
  - Better than fine-grained locking



From: Intel Transactional Synchronization Extensions - IDF2012

HTM: Is Synchronization Problem Solved?

- Best-efforts HTM
  - Simple hardware design
  - Commit is not guaranteed
    - Software fallback path required
  - Limited capabilities
    - Data size
    - Time
    - Contention Management
- Our goal:
  - Overcome HTM limitations
    - Broaden the scope of HTM
      - HTM unsupported transactions gain more performance

Benefits of Multi-core

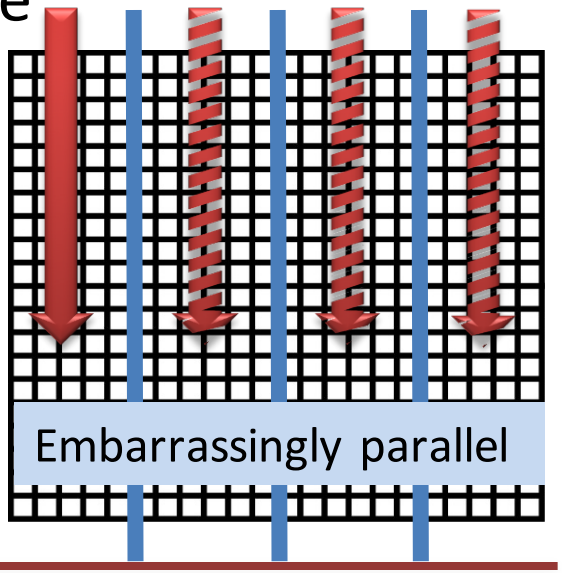
- Real parallelism
  - Can run more jobs concurrently
  - Run several applications together faster
- Scalable
  - More jobs can use more cores
  - NUMA architectures are scalable
- Is there a problem?
  - Data sharing
  - Software does not follow HW

Software does not Follow HW

- Current software is designed for older Hardware
- New Hardware gives the illusion that nothing changed
  - Same API (cache coherence, memory access, …)
- Result:
  - Current Software works, but does not exploit all HW capabilities
- Solution:
  - Redesign Software to exploit the new Hardware
    - Be aware of the underlying Hardware

How to Exploit Scalability?

- Our focus: Memory interface
  - UMA: Uniform Memory Access
  - NUMA: Non-Uniform Memory Access
    - Different memory access time impact scalability
    - Data locality is crucial
    - Synchronization techniques do not scale
      - Including TM algorithms
- Our goal:
  - Redesign TM algorithms to be NUMA-aware and scalable



In Summary

- Improving performance & scalability of synchronization techniques

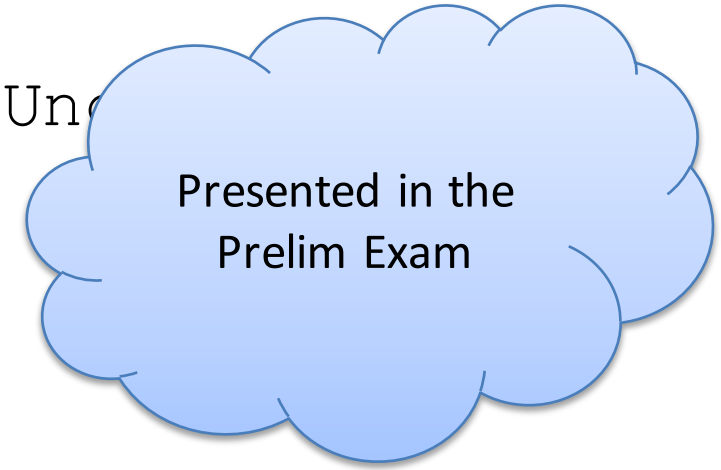
By Optimizing Transactional Memory

  - HTM has the best performance but …
    - Limited (best-effort)
    - Overcome these limitations
  - NUMA architectures are scalable but …
    - Different memory access time
    - Current TM algorithms are not scalable
    - Redesign TM to be NUMA-aware

Summary of Contributions

- Part-HTM
  - Overcome best-efforts HTM resource limitations
  - Partition large transactions in smaller sub-HTM transactions

    Published at SPAA '15, Unc
    PACT'15



- Octonauts
  - HTM-aware scheduler
  - Tackles HTM live-lock problem at high contention levels

    Published at SPAA'15

Summary of Contributions (2)

- Precise-TM
  - Fine-grained fallback technique
  - Allows more concurrency between HTM-STM

    In preparation for PPoPP'16
- Nemo
  - NUMA-aware STM algorithm
  - Achieves high scalability

    In preparation for PPoPP'16

Summary of (Contributions)* (3)

- Shield
  - State-machine replication based dependable transactional system
  - Optimized fo re architectures
    - Publis ONS'14
- SoftX
  - Redundant le transactional system
  - Achieve dependability with low synchronization overhead
    - Published at NCA'14

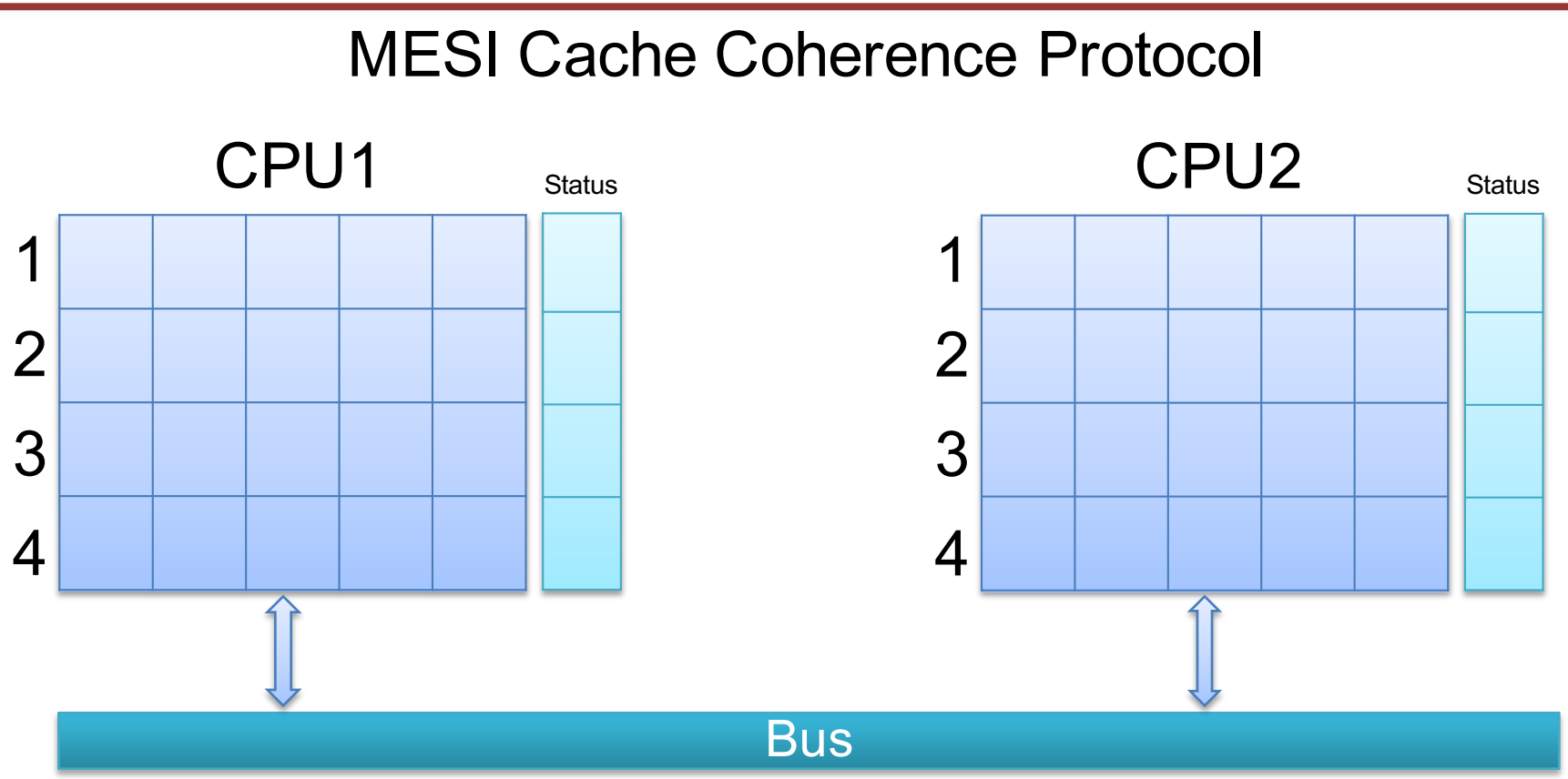
Scoped out from the dissertation

* Not in the dissertation

Fine-grained fallback path for HTM

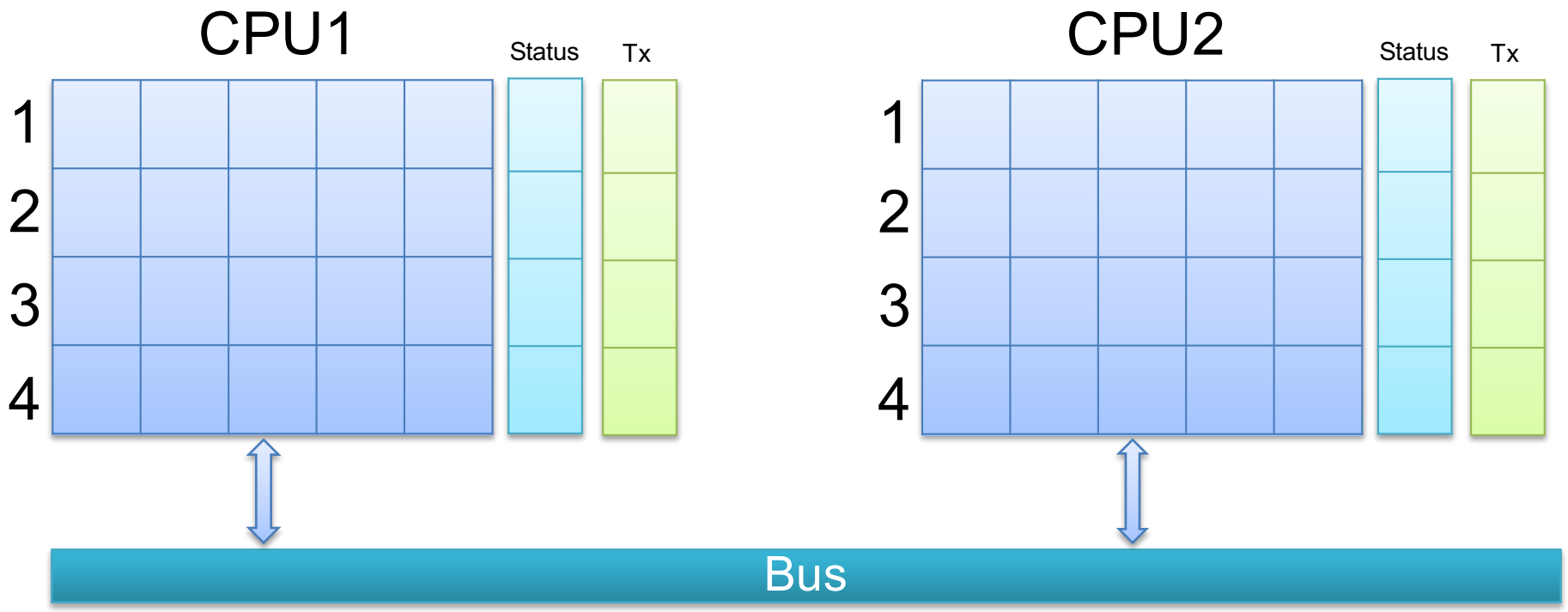
PRECISE-TM

Cache coherence-based HTM



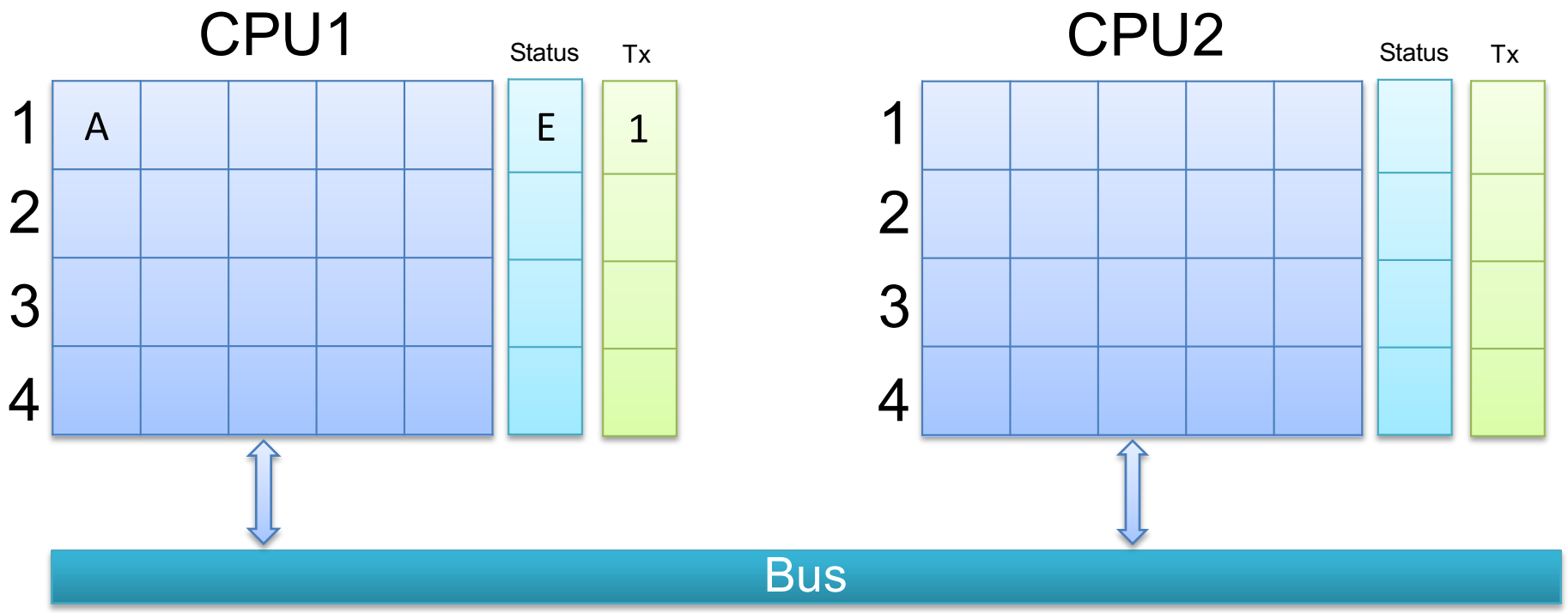
Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



Cache coherence-based HTM

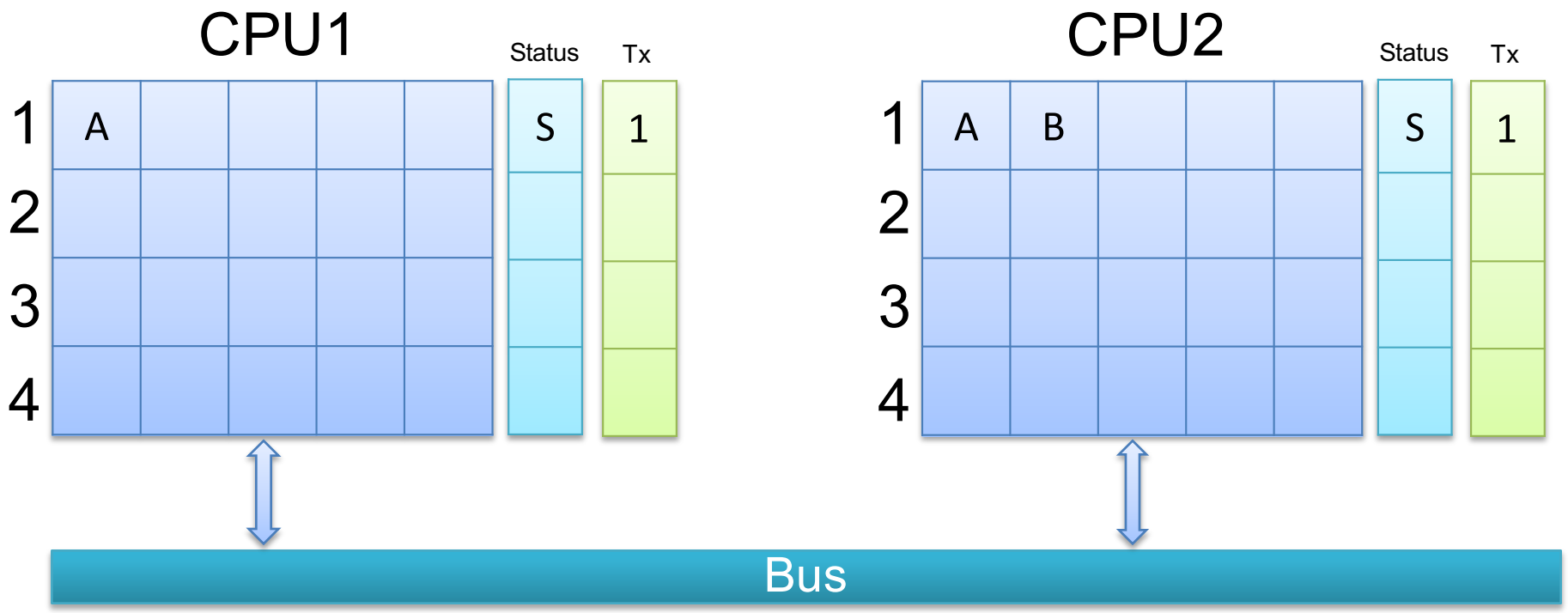
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol

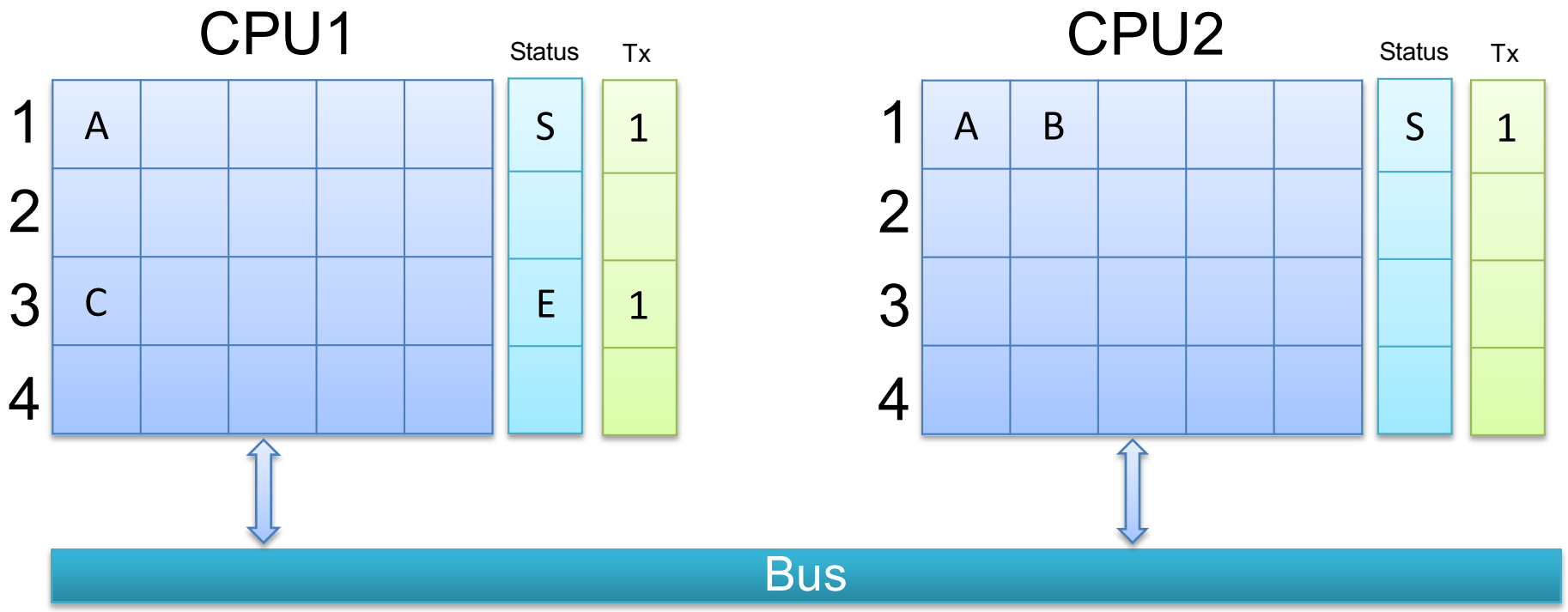


- _xbegin()
- read (0)

- _xbegin()
- read (1)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol

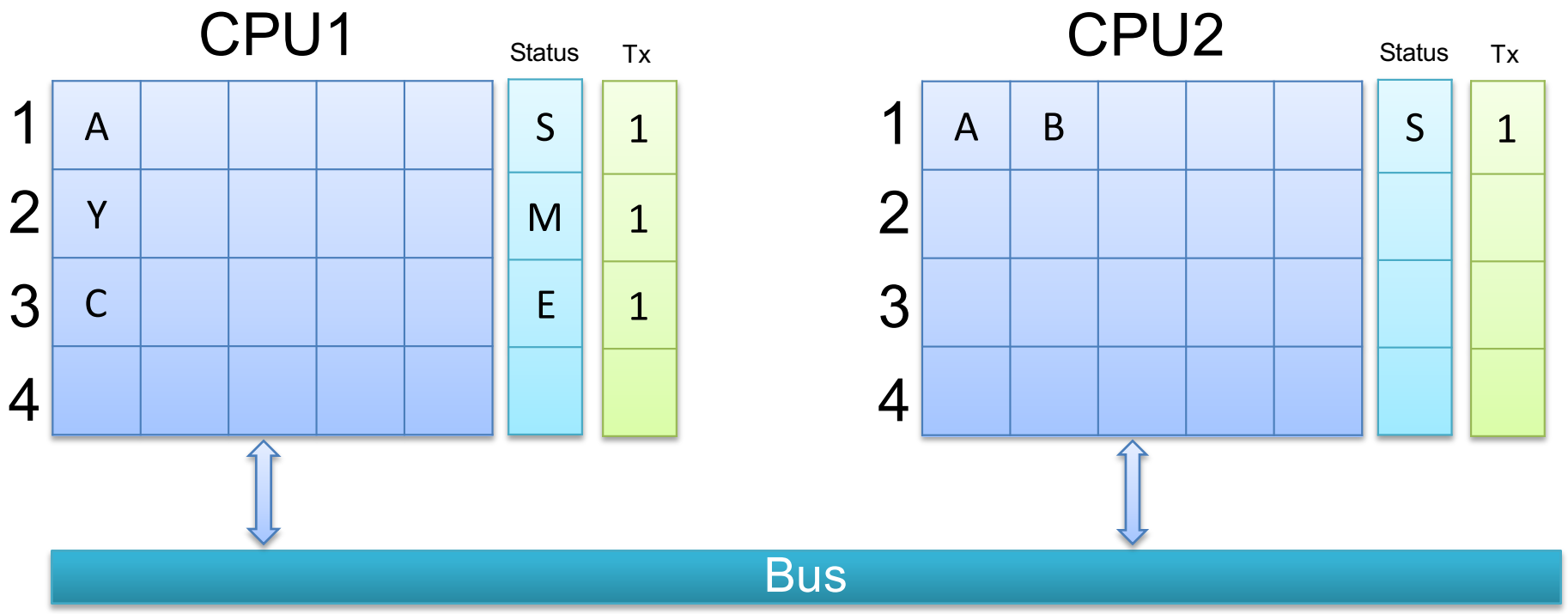


- _xbegin()
- read (0)
- read (10)

- _xbegin()
- read (1)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol

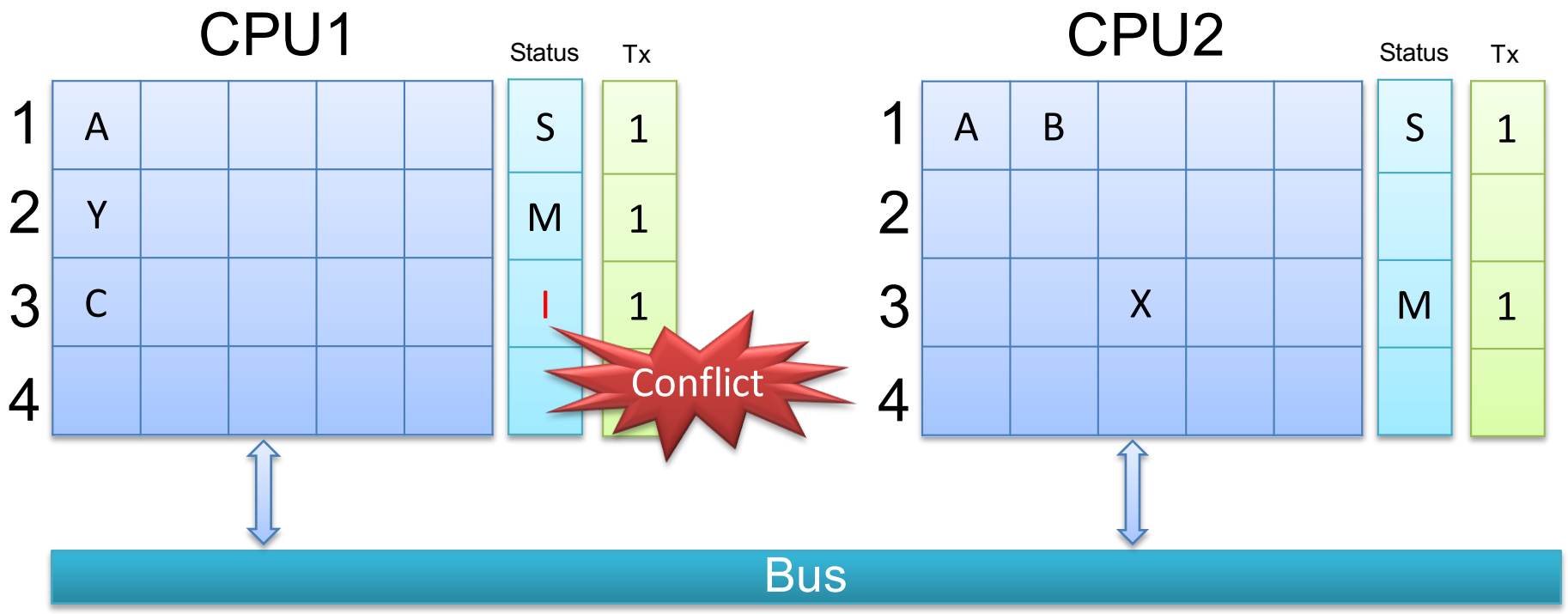


- _xbegin()
- read (0)
- read (10)
- write (5, Y)

- _xbegin()
- read (1)

Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol

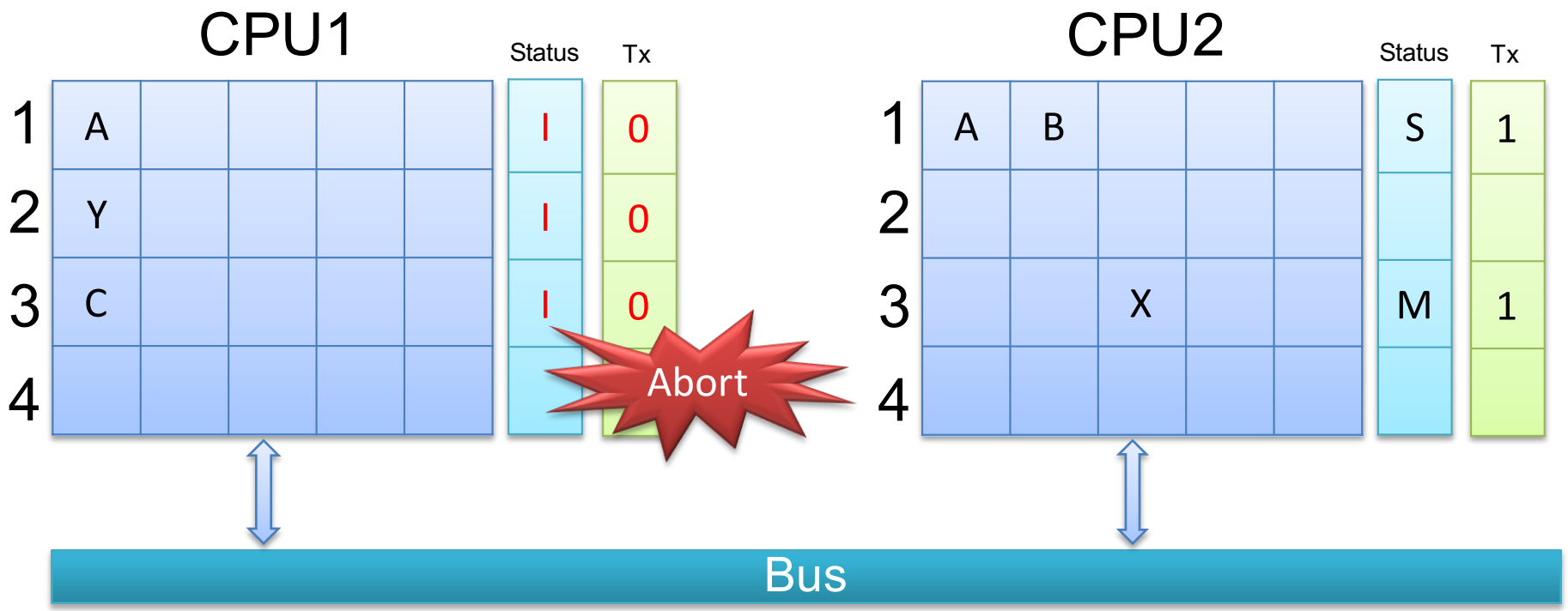


- _xbegin()
- read (0)
- read (10)
- write (5, Y)

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

Cache coherence-based HTM

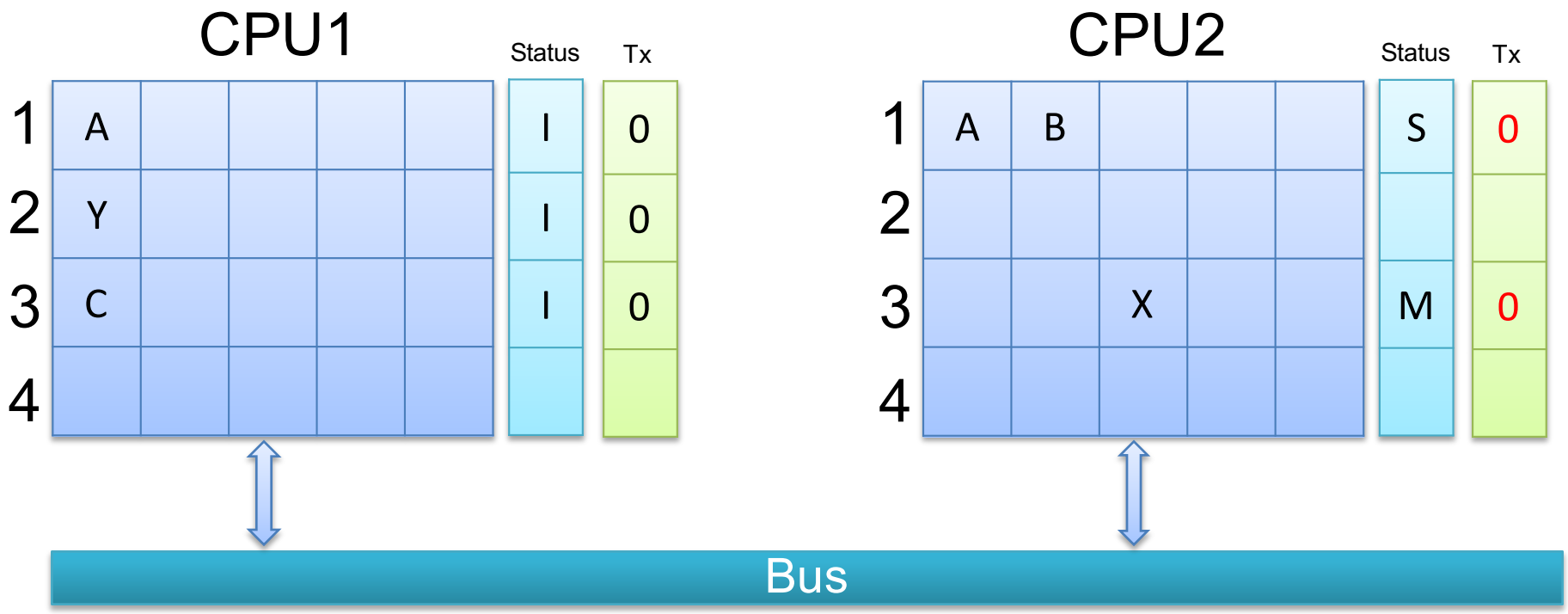
Transactional MESI Cache Coherence Protocol



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

Cache coherence-based HTM

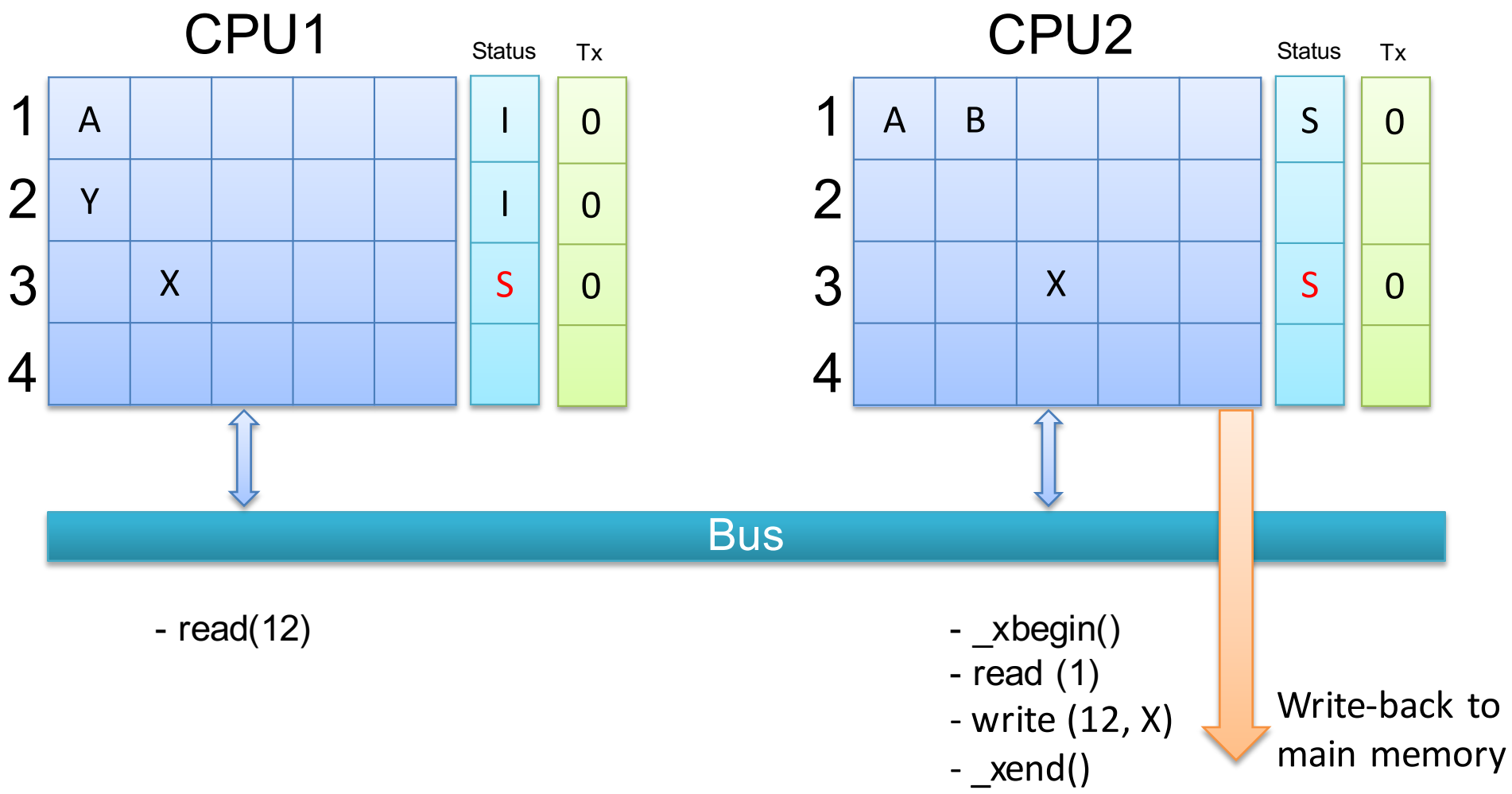
Transactional MESI Cache Coherence Protocol



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

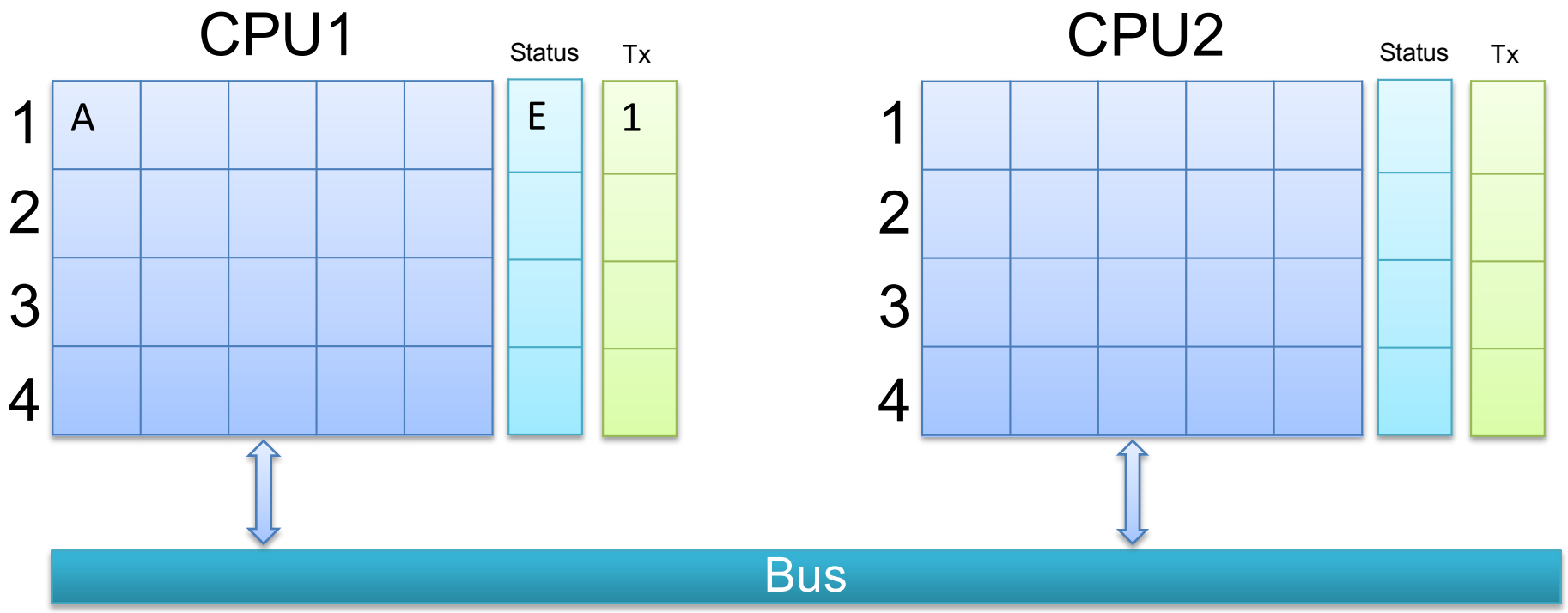
Cache coherence-based HTM

Transactional MESI Cache Coherence Protocol



HTM Limitations

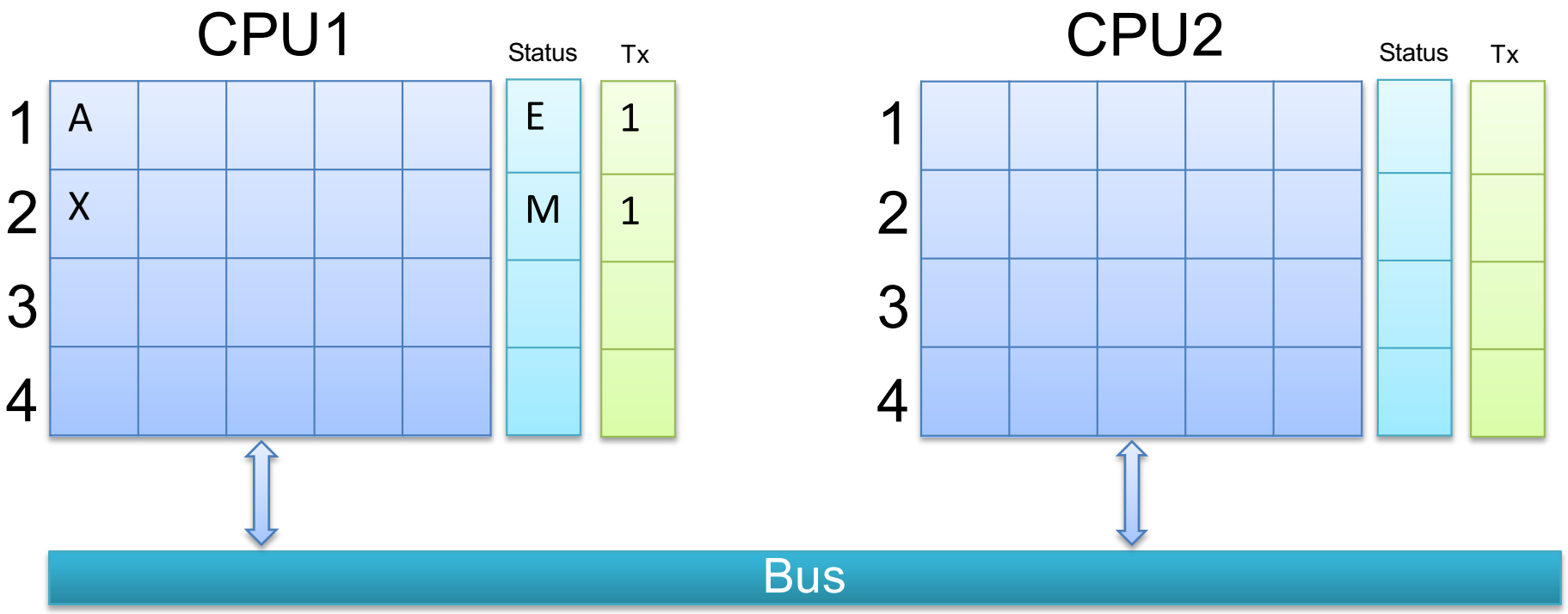
Transactional MESI Cache Coherence Protocol



- _xbegin()
- read (0)

HTM Limitations

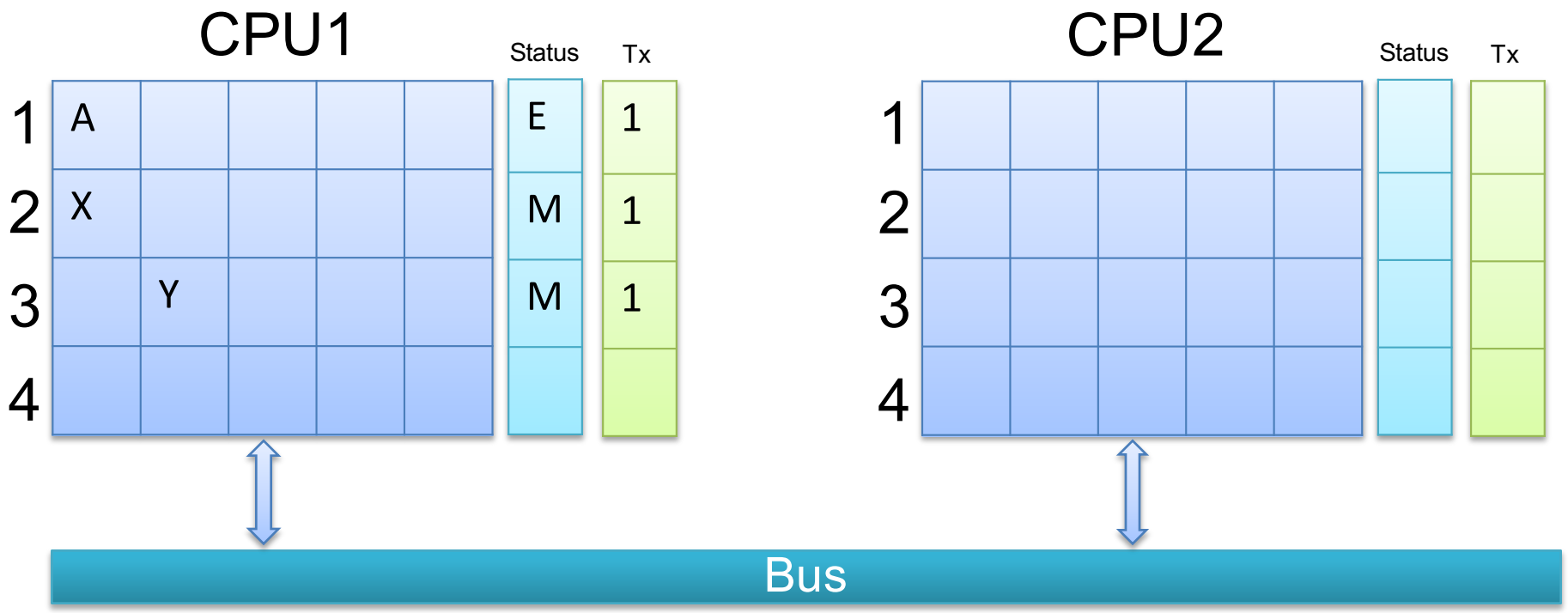
Transactional MESI Cache Coherence Protocol



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

HTM Limitations

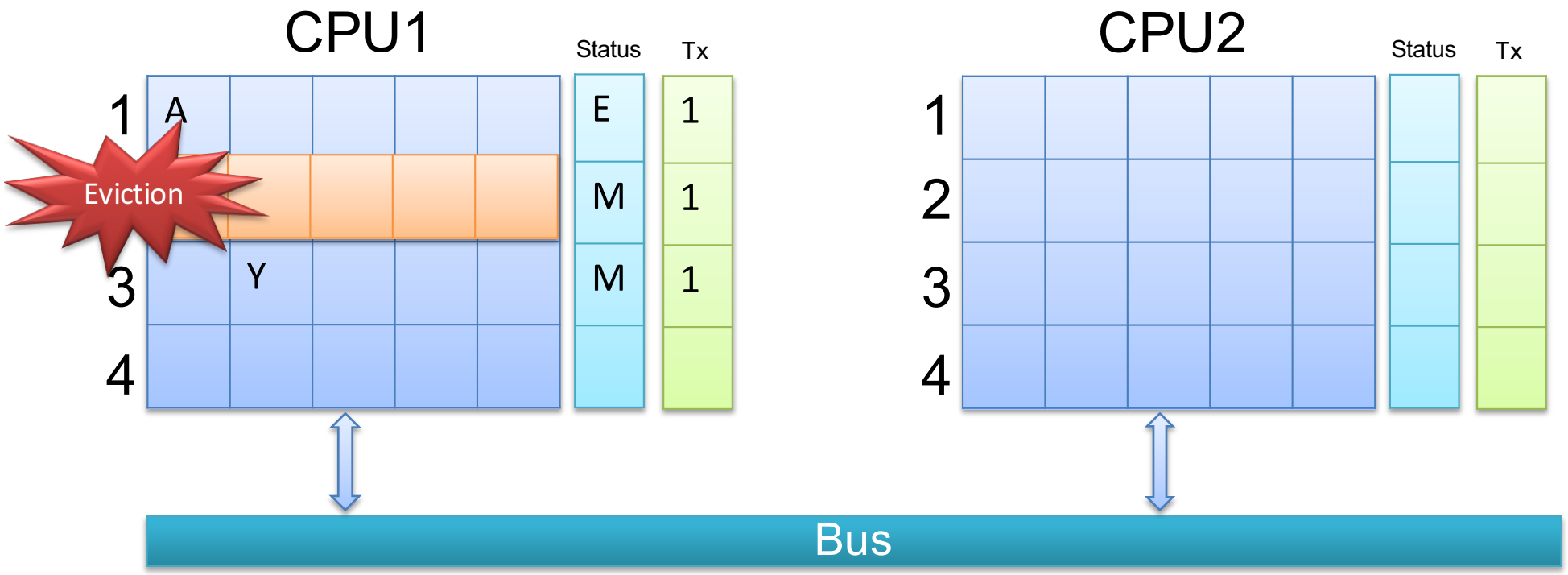
Transactional MESI Cache Coherence Protocol



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

HTM Limitations

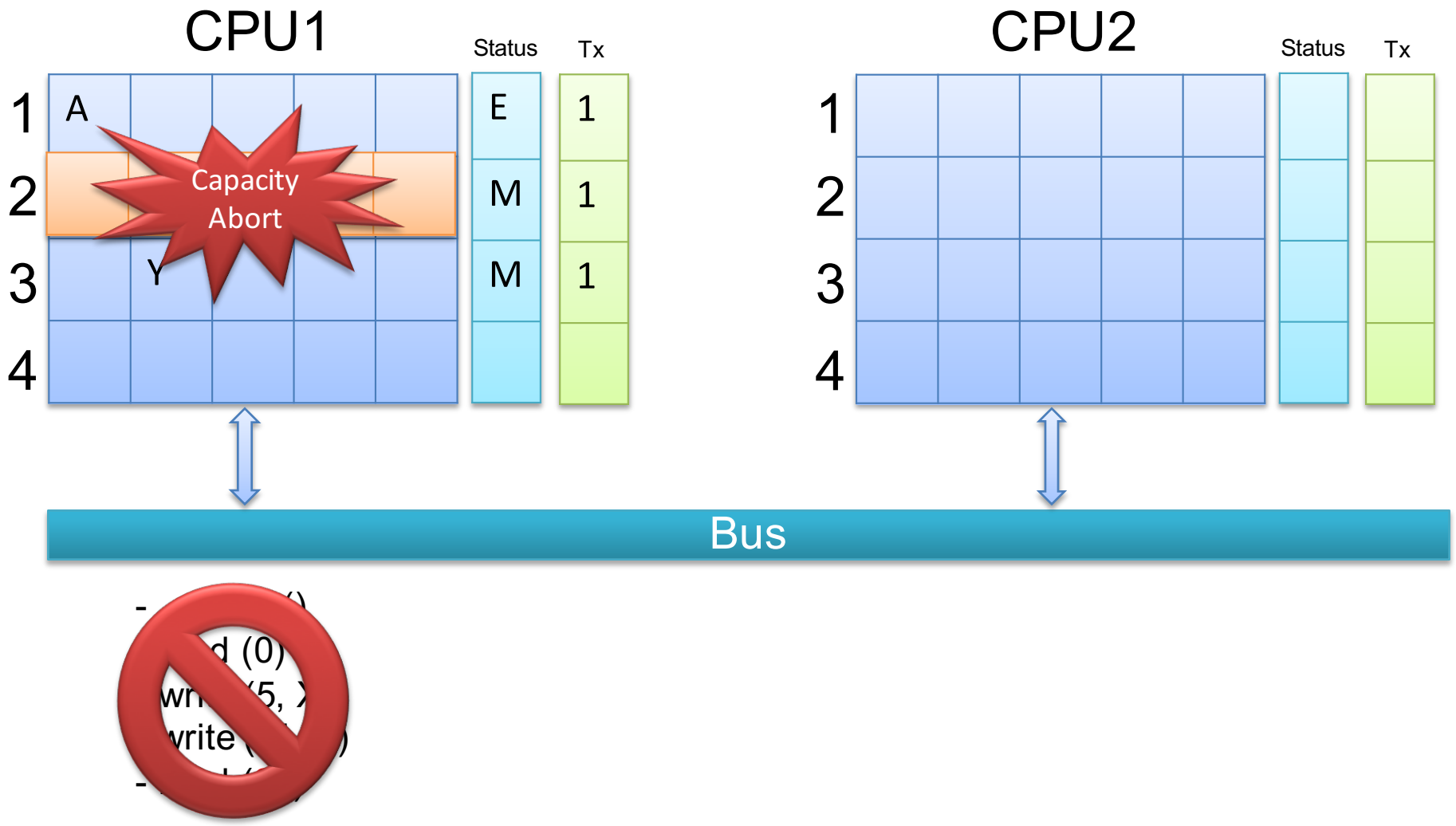
Transactional MESI Cache Coherence Protocol



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

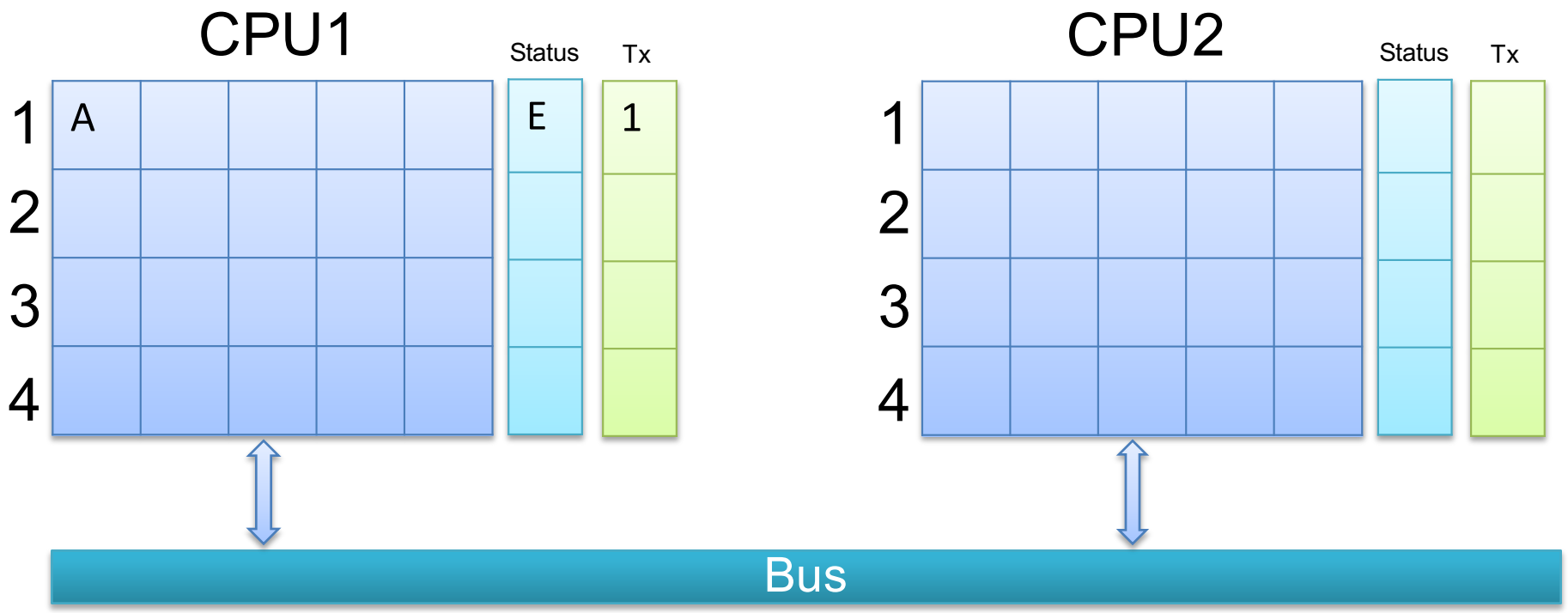
HTM Limitations

Transactional MESI Cache Coherence Protocol



HTM Limitations (2)

Transactional MESI Cache Coherence Protocol

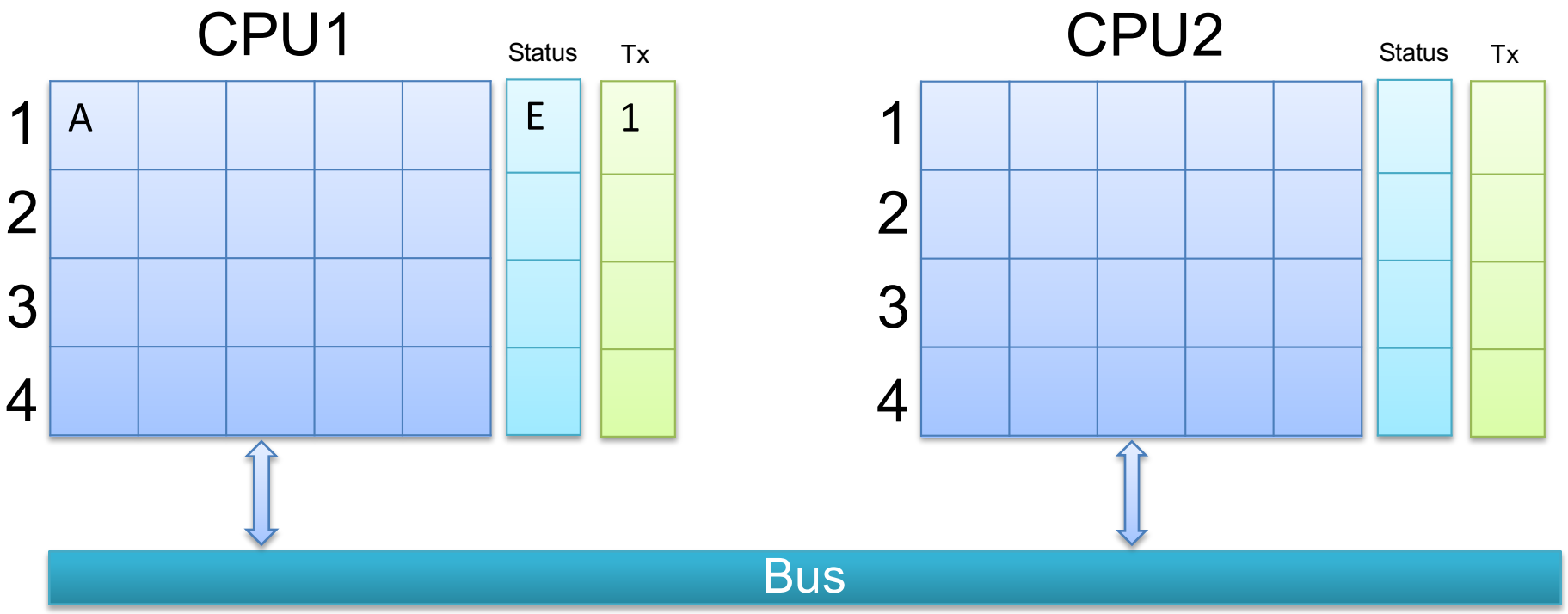


- _xbegin()
- read (0)

Time

HTM Limitations (2)

Transactional MESI Cache Coherence Protocol

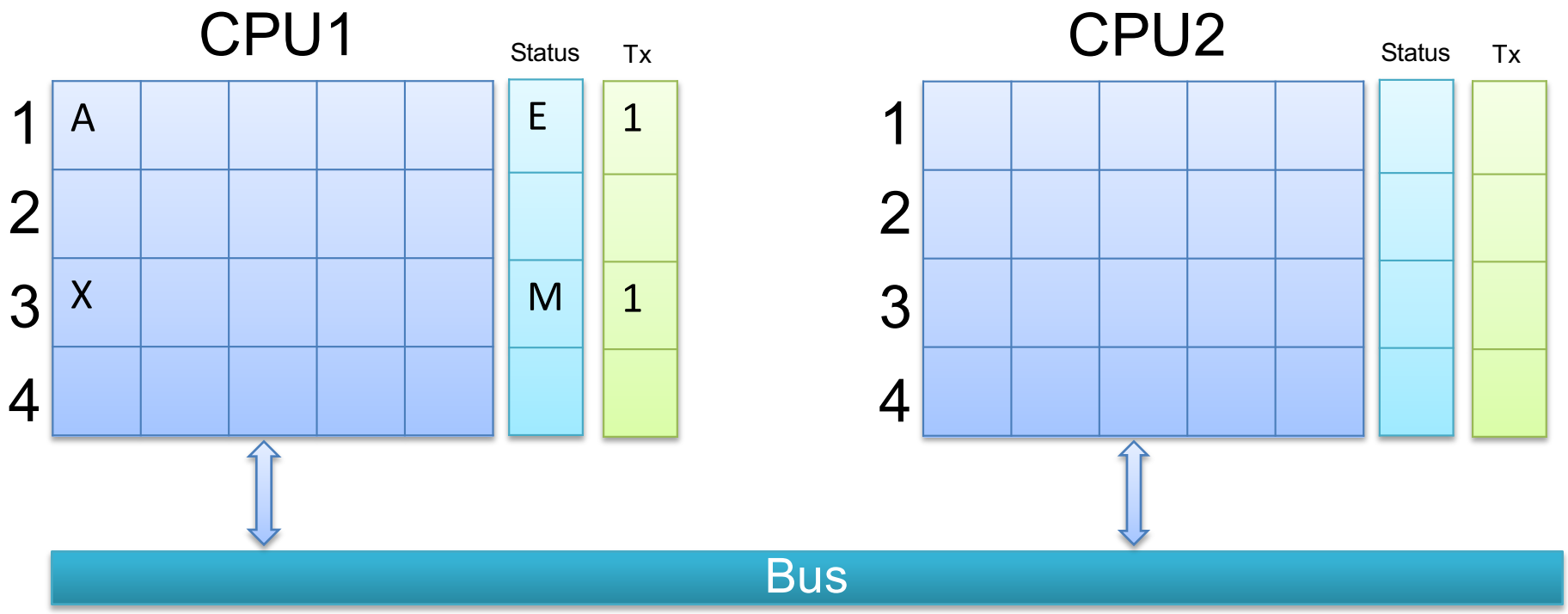


- _xbegin()
- read (0)
- Do some long work

Time

HTM Limitations (2)

Transactional MESI Cache Coherence Protocol

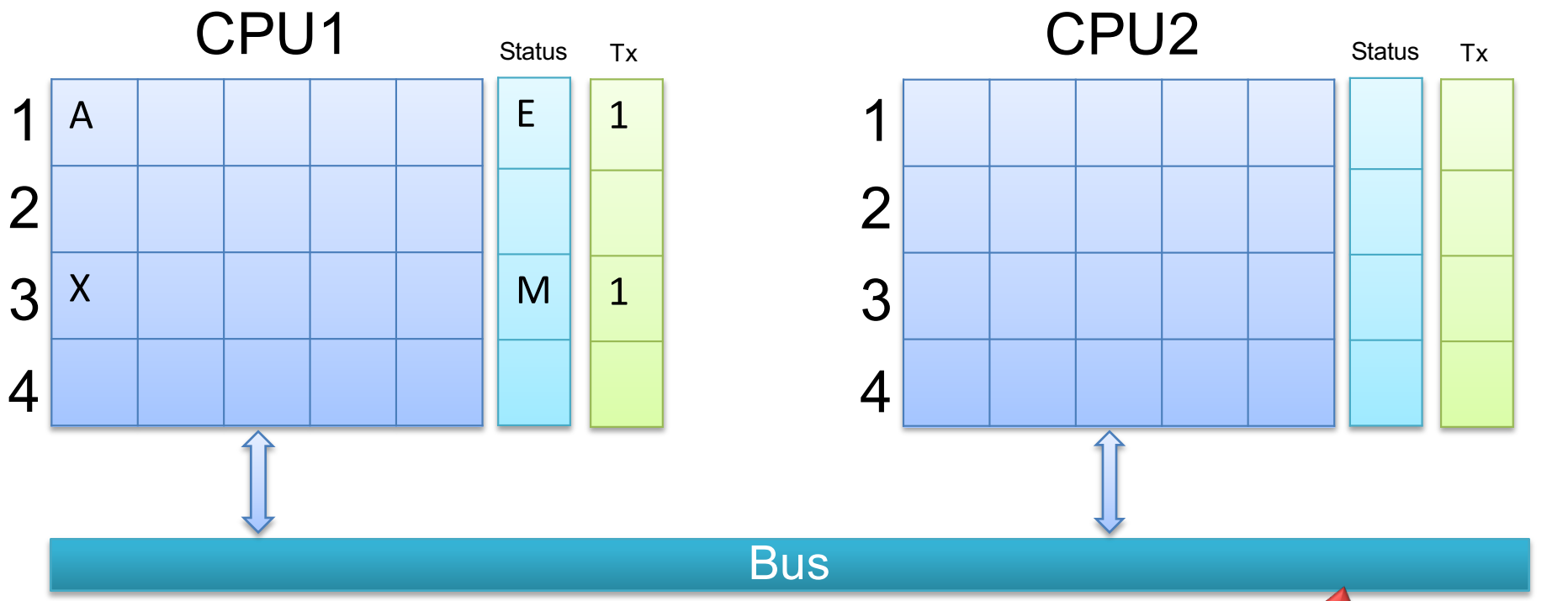


- _xbegin()
- read (0)
- Do some long work
- Write (10, X)

Time

HTM Limitations (2)

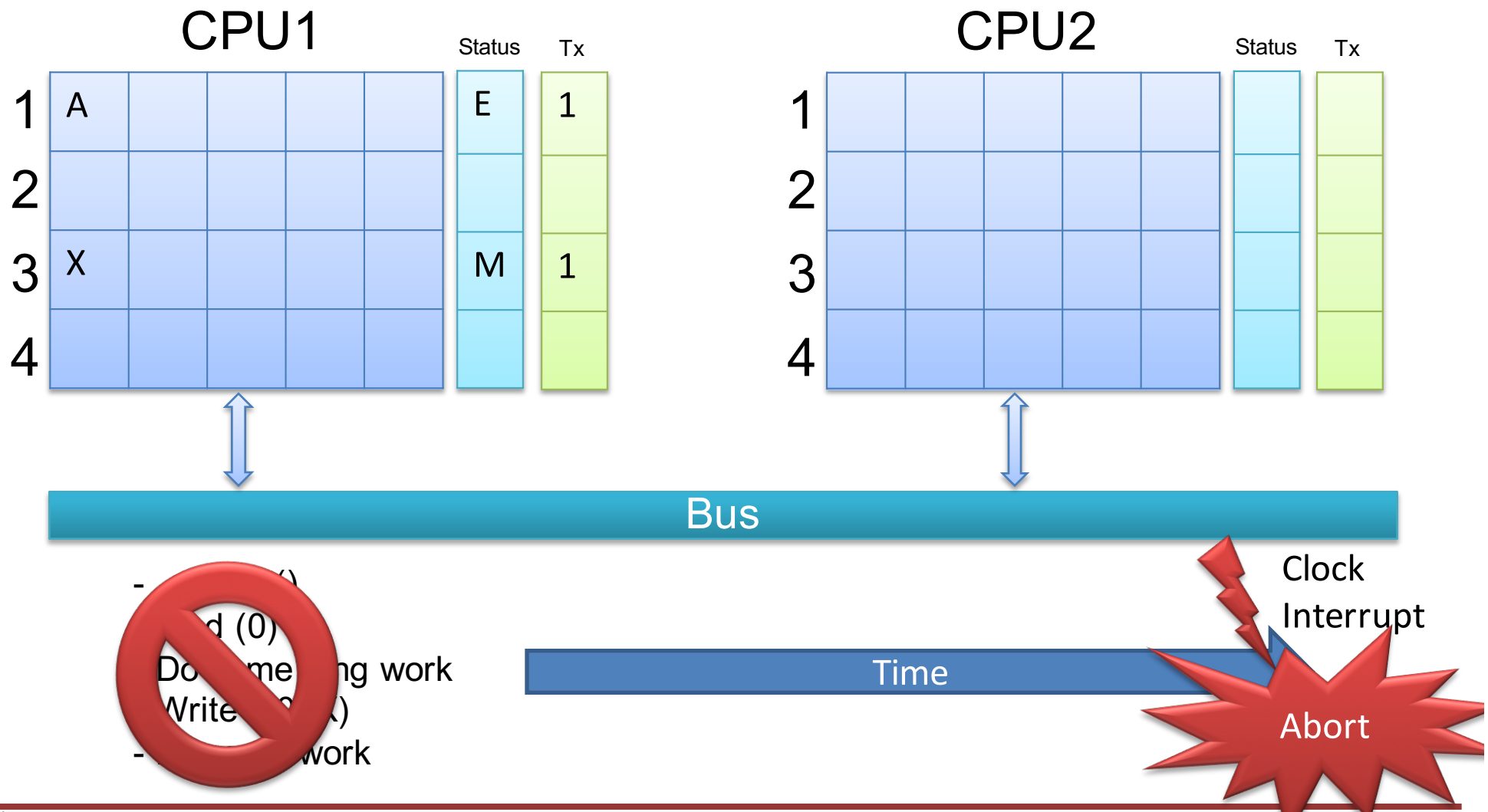
Transactional MESI Cache Coherence Protocol



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

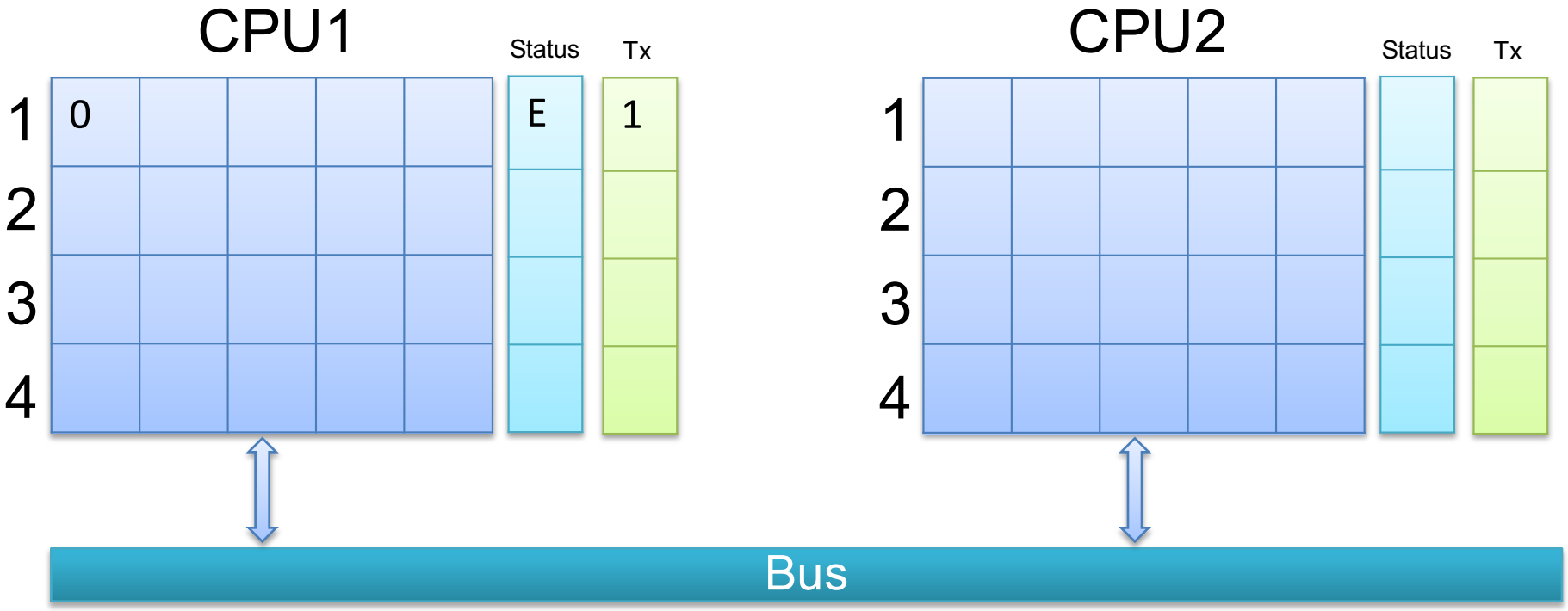
HTM Limitations (2)

Transactional MESI Cache Coherence Protocol



HTM Limitations (3)

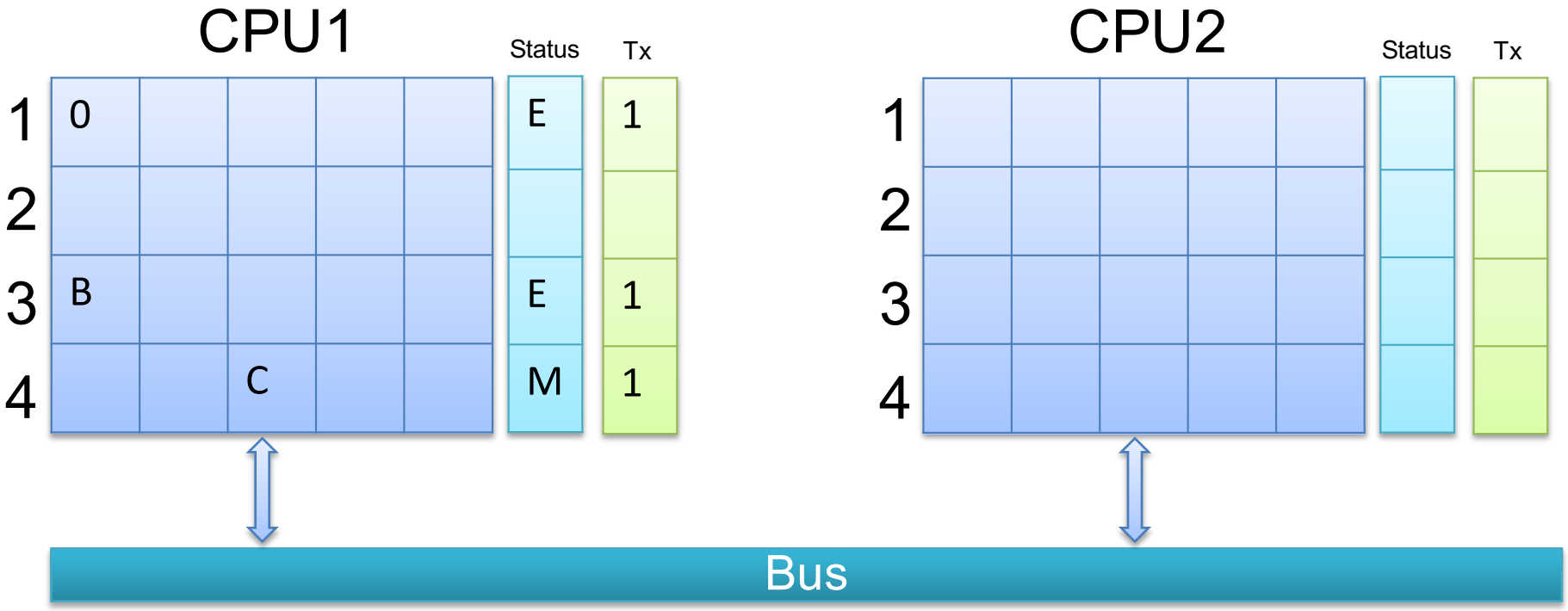
- Must define a software fallback path
  - Default is global lock



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

HTM Limitations (3)

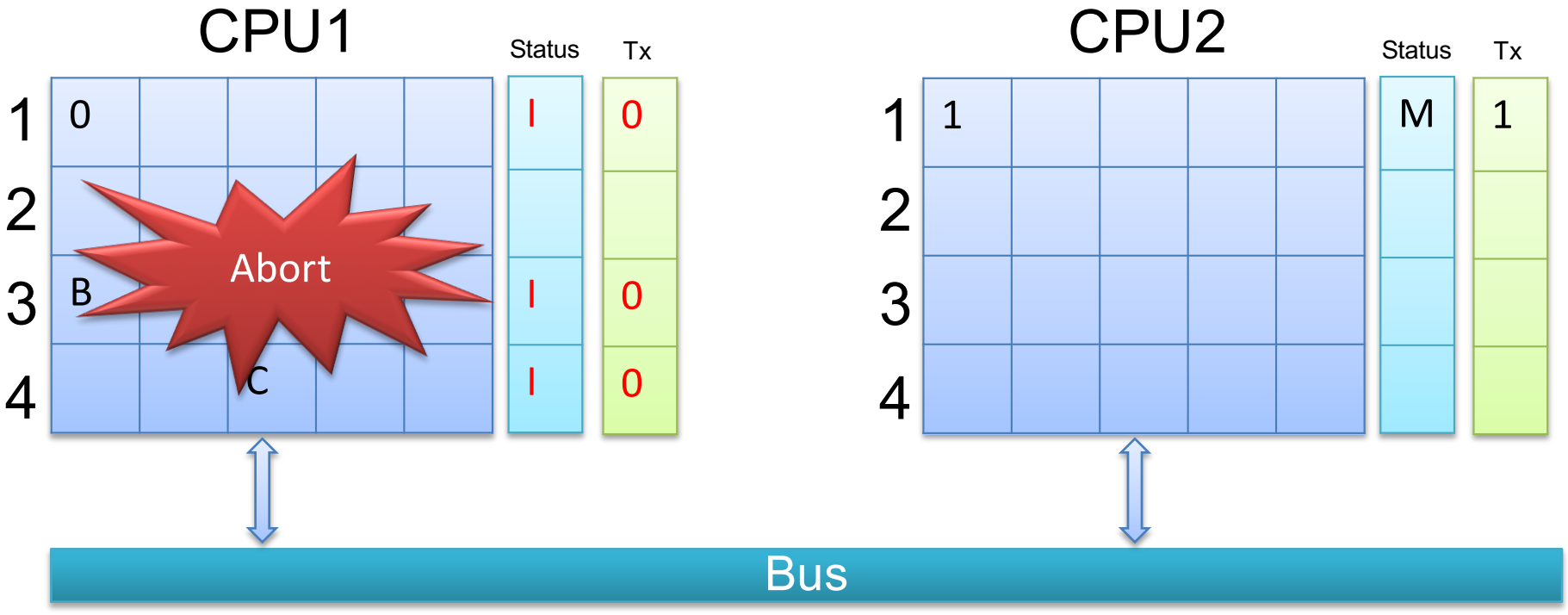
- Must define a software fallback path
  - Default is global lock



- _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)

Precise-TM

- 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

Motivation

- All HTM transactions monitors the global lock in the beginning
- Serialize software fallback path
  - Block all HTM transactions

Algorithm 1 TSX in GCC

```
 1: int attempts ← 2
 2: int status ← XBEGIN
 3: if status ≠ ok then
 4:     if attempts = 0 then
 5:         acquire(globalLock)
 6:     else
 7:         attempts−−
 8:         goto line 2
 9: if is_locked(globalLock)
10:     XABORT
11: ▷ ...transactional code
12: if attempts = 0 then
13:     release(globalLock)
14: else
15:     XEND
```

Related Work

- STM research focused on fine-grained locking
  - E.g., TL2 [35], Swiss [40]
- Initial hybrid HTM proposals tries fine-grained locking
  - Poor performance
  - E.g., RH TL2 [68]
- Thus, hybrid HTM moved towards course-grained
  - HyNOrec [81, 28]

Related Work (2)

- Refined lock-elision [34]
  - Uses lock table instead of a single lock
  - Critical section is instrumented
    - HTM reads lock table
  - One software transaction is allowed
  - Best results are with one lock only

Related Work (3)

- The problem of Precise-TM is orthogonal to other approaches targeting HTM limitations.
- Enhance the default global locking fall back
  - Lazy subscription [20]
  - Self-tuning [37]
- STM as a fall back
  - All STM instrumentation overhead
  - Examples
    - HyNOrec [28]
    - Invyswell [19]
- Reduced hardware transactions
  - Commit phase in HTM
  - Examples
    - RH TL2 [68]
    - RH NOrec [69]

Related Work

- The problem of Precise-TM is orthogonal to other approaches targetin
- Enha
  STM
  overhead
  ctions
    - RH NOrec [69]

All uses unnecessary communication between HTM-HTM and HTM-STM

```
function FAST_PATH_COMMIT(ctx)
    if HTM is read only then
      ▷ Detected by compiler static analysis
        HTM_Commit()
        return
    end if
    if num_of_fallbacks > 0 then
      ▷ Notify mixed slow-paths about the update
        if is_locked(global_clock) then
            HTM_Abort()
        end if
        global_clock ← global_clock + 1
    end if
    HTM_Commit()
end function
```

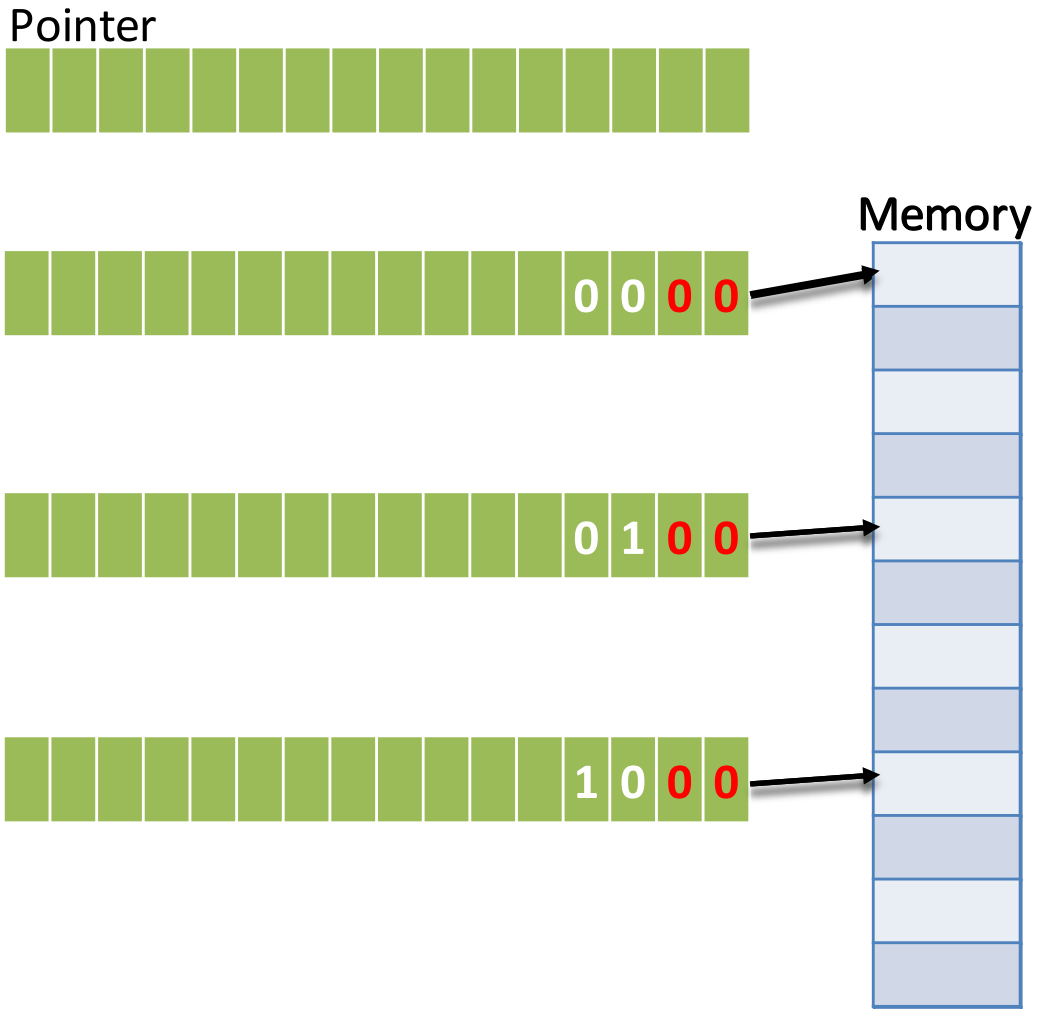
```
tx_end:
if (!commit_lock)
    ++hw_post_commit;
    _xend();
else
    if (!conflict with SW txn)
        ++hw_post_commit;
        _xend();
else _xabort();
```

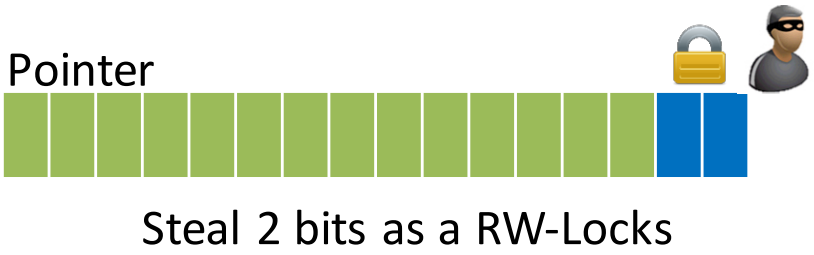
Precise-TM Idea

- Precise-TM provides a fine-grained fallback technique
  - No unnecessary interference between HTM and STM
    - Sharing common objects
  - Lightweight instrumentation
    - Time
    - Space
  - More concurrency between transactions
    - Fewer false conflicts

Precise-TM: How it works?

- Address embedded locks





Precise-TM: How it works? (2)

- Address-embedded locks used only between HTM and STM
- HTM-STM: Precise-TM
- HTM-HTM: Handled by HW
- STM-STM: Multiple approaches
  - V1: Uses global lock
  - V2: 2-Phase locking

Precise-TM V1

- Uses global lock for synchronizing STM-STM
  - More concurrency for HTM
  - Single transaction in STM

```
18: procedure READ-REFERENCE(x)
19:     if fast-path then
20:         if x & 0x0002 then
21:             _xabort();
22:     else
23:         x ← x | 0x0001
24:         add(reference-log, x)
25:     return x & !(0x0003)
26: end procedure
27:
28: procedure WRITE-REFERENCE(x, val)
29:     if fast-path then
30:         if x & 0x0003 then
31:             _xabort();
32:         x ← val
33:     else
34:         x ← val | 0x0002
35:         add(reference-log, x)
36: end procedure
```

Precise-TM V2

- More concurrency
  - STM-STM
- Exploit the address-embedded locks to synchronize STMs also

```
17: procedure READ-REFERENCE(x)
18:     if fast-path then
19:         if x & 0x0002 then
20:             _xabort();
21:     else
22:         if !isLocked(x) then
23:             if CAS(x, x & !(0x0003), x | 0x0001) then
24:                 add(reference-log, x)
25:             elseAbort-Slow-Path()
26:         else if isLockedByMe(x) then
27:             x ← x | 0x0001
28:         elseAbort-Slow-Path()
29:     return x & !(0x0003)
30: end procedure
31:
32: procedure WRITE-REFERENCE(x, val)
33:     if fast-path then
34:         if x & 0x0003 then
35:             _xabort();
36:         x ← val
37:     else
38:         if !isLocked(x) then
39:             if CAS(x, x & !(0x0003), val | 0x0002) then
40:                 add(reference-log, x)
41:             elseAbort-Slow-Path()
42:         else if isLockedByMe(x) then
43:             x ← val | 0x0002
44:         elseAbort-Slow-Path()
45: end procedure
```

With Precise-TM

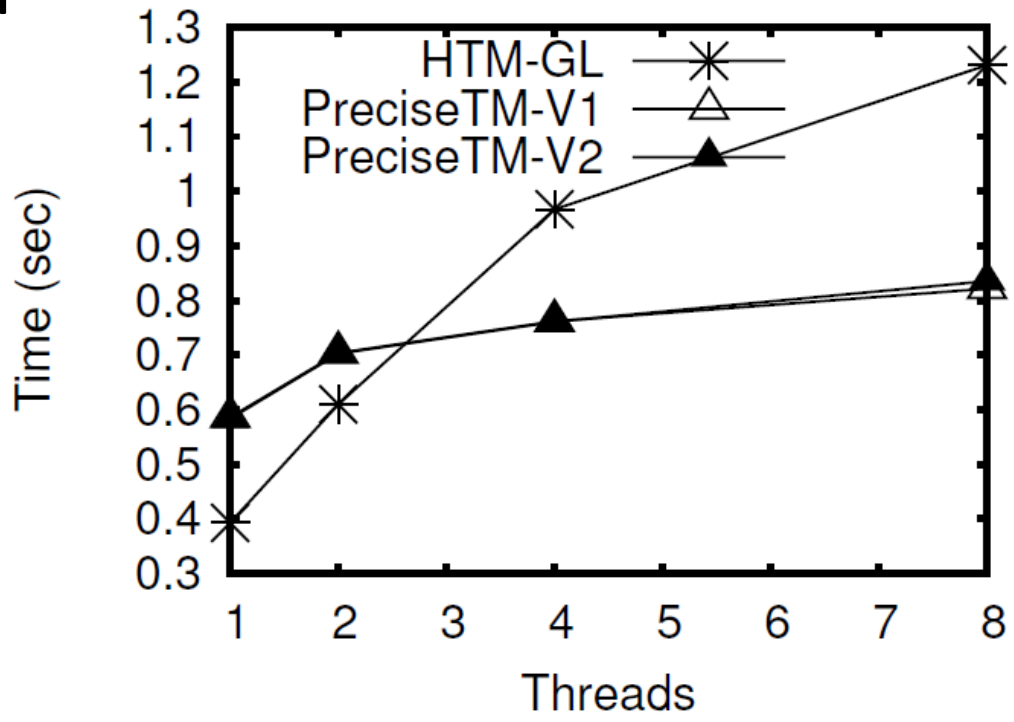
- No false aborts
- No added meta-data
- More concurrency between transactions
  - Can optimize STM-STM communication in different ways

Evaluation

- Implemented in C++
- Tested on Haswell Core i7-4770 (4 cores, 8 threads)
- Compiler: GCC 4.8.2, OS: Ubuntu 14.04 LTS
- Benchmarks: Bank, EigenBench, Linked-List
- Competitors: HTM-GL
- Available as an open-source library

Evaluation: Bank

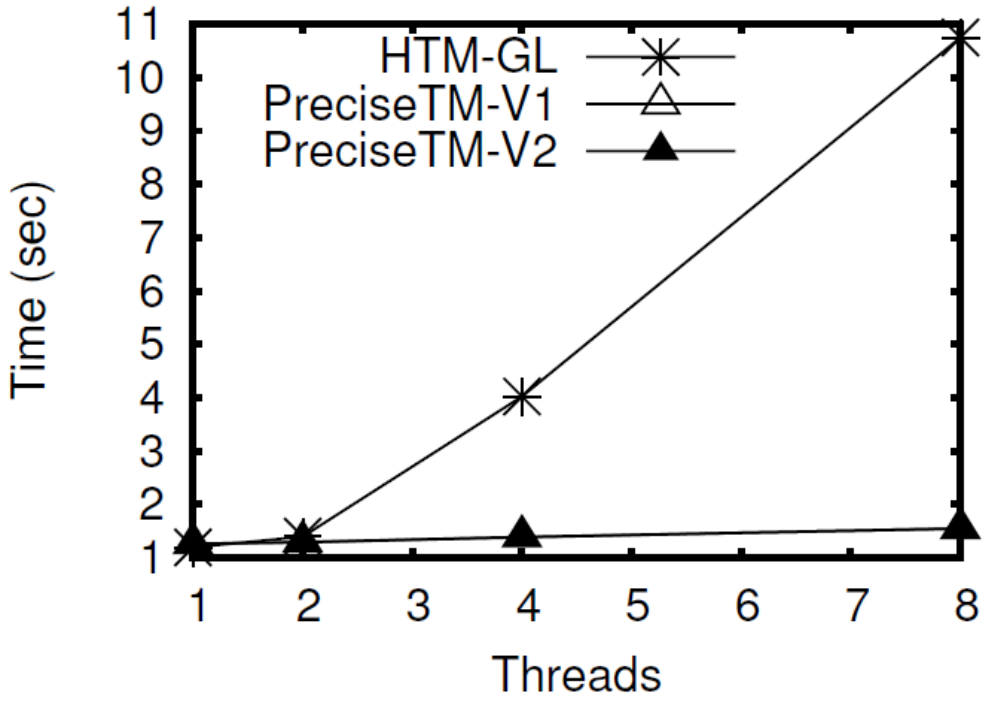
- 64K accounts → Low conflict
- In all experiments, the amount of work/thread is fixed



(b) 20% write.

Evaluation: Bank (Disjoint)

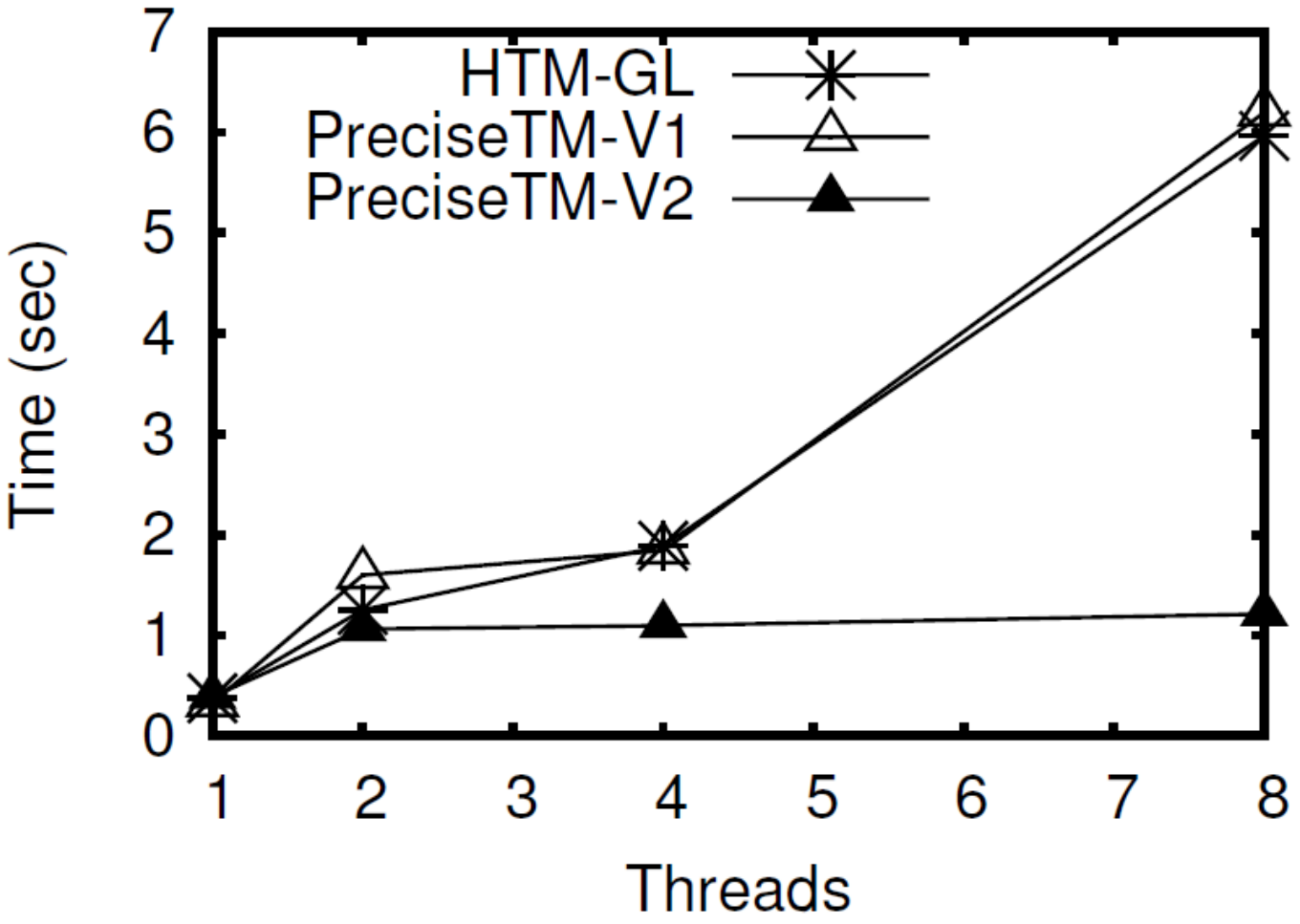
- Disjoint access → No conflict
  - Perfect for our approach



(b) 20% write.

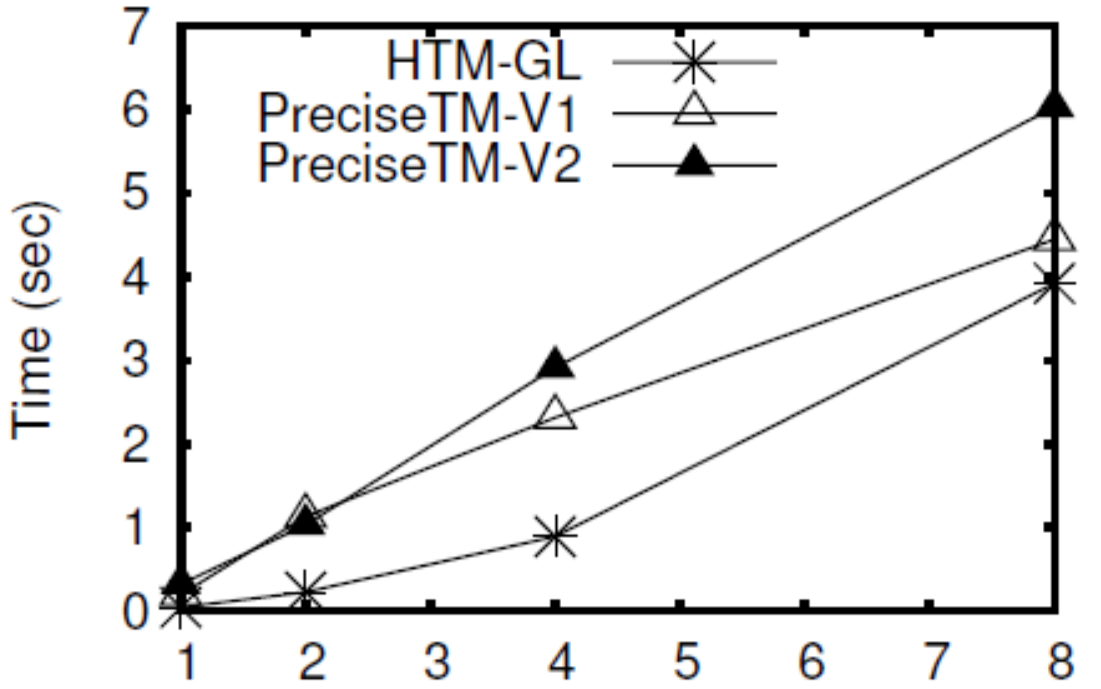
Evaluation: EigenBench

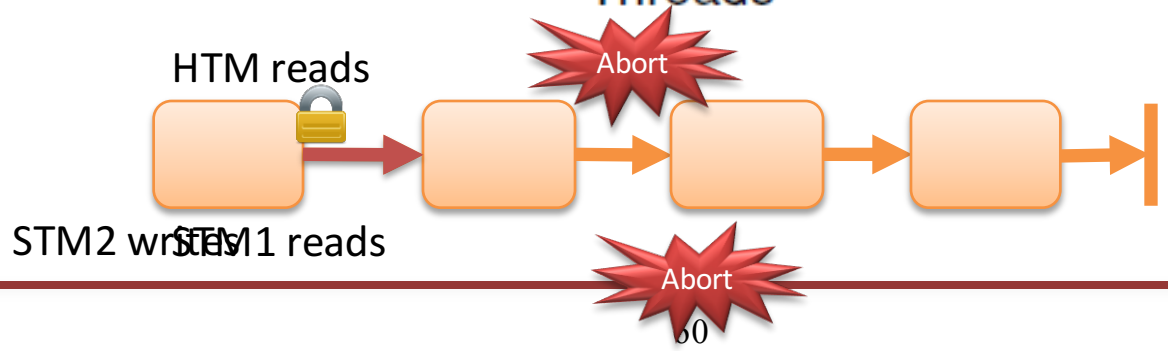
- Best case for Precise-TM V2

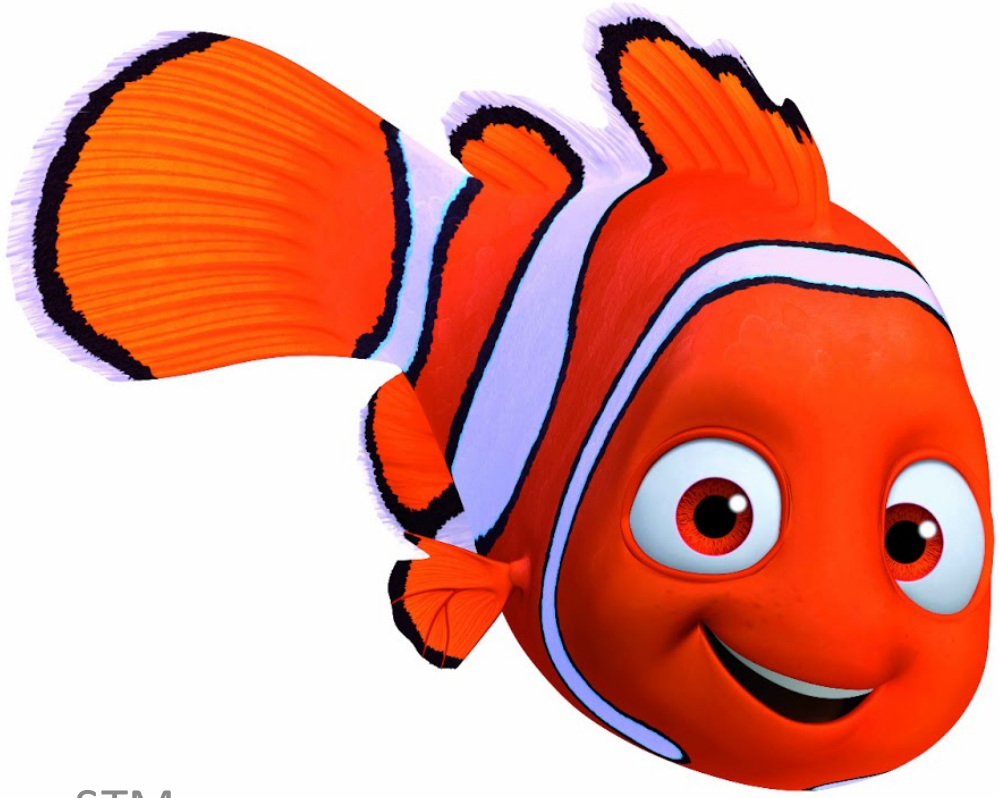


Evaluation: Linked-List

- Worst case for Precise-TM





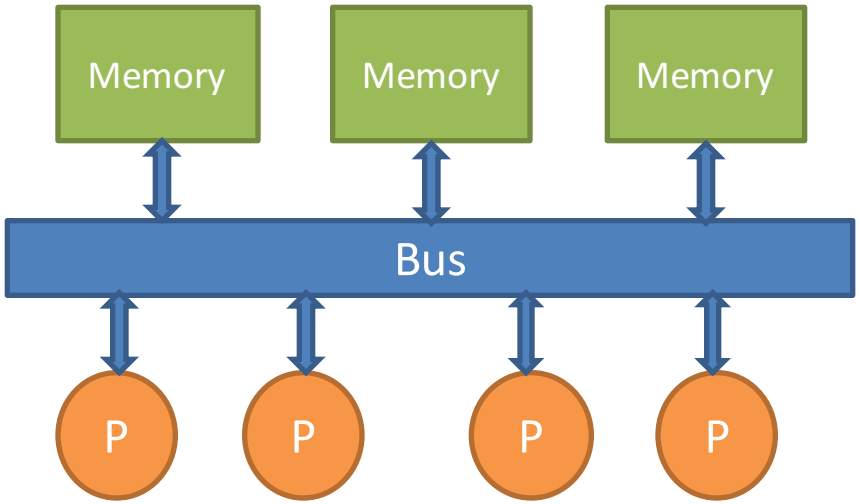


NUMA-aware STM

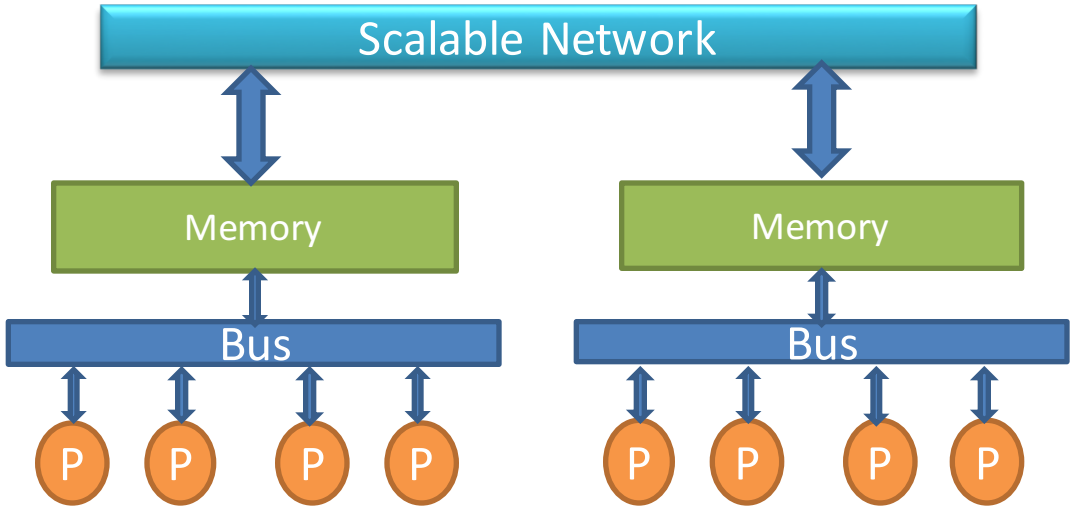
NEMO

What are NUMA Architectures?

- UMA architectures cannot scale
  - NUMA is one solution



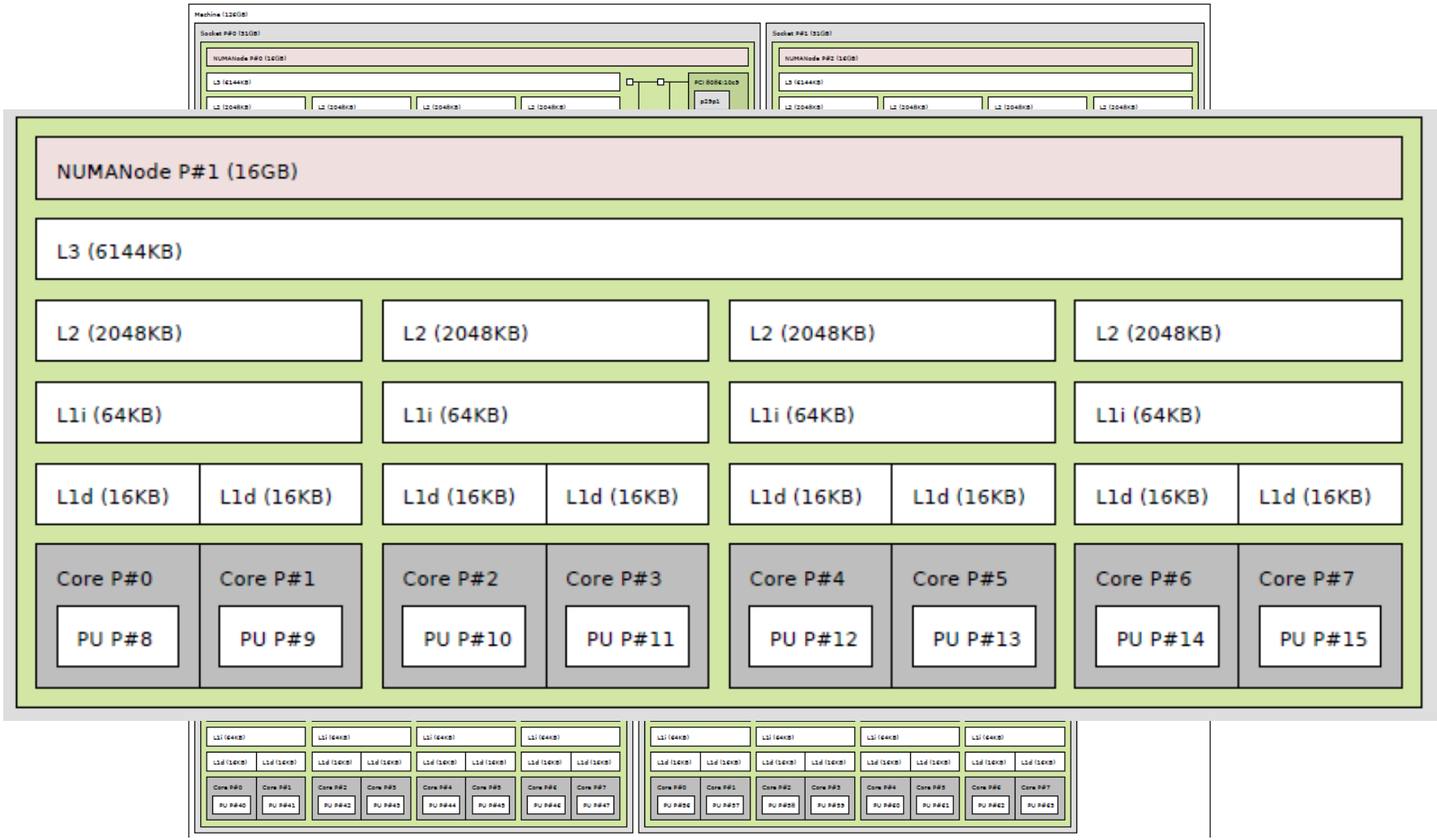
UMA



NUMA

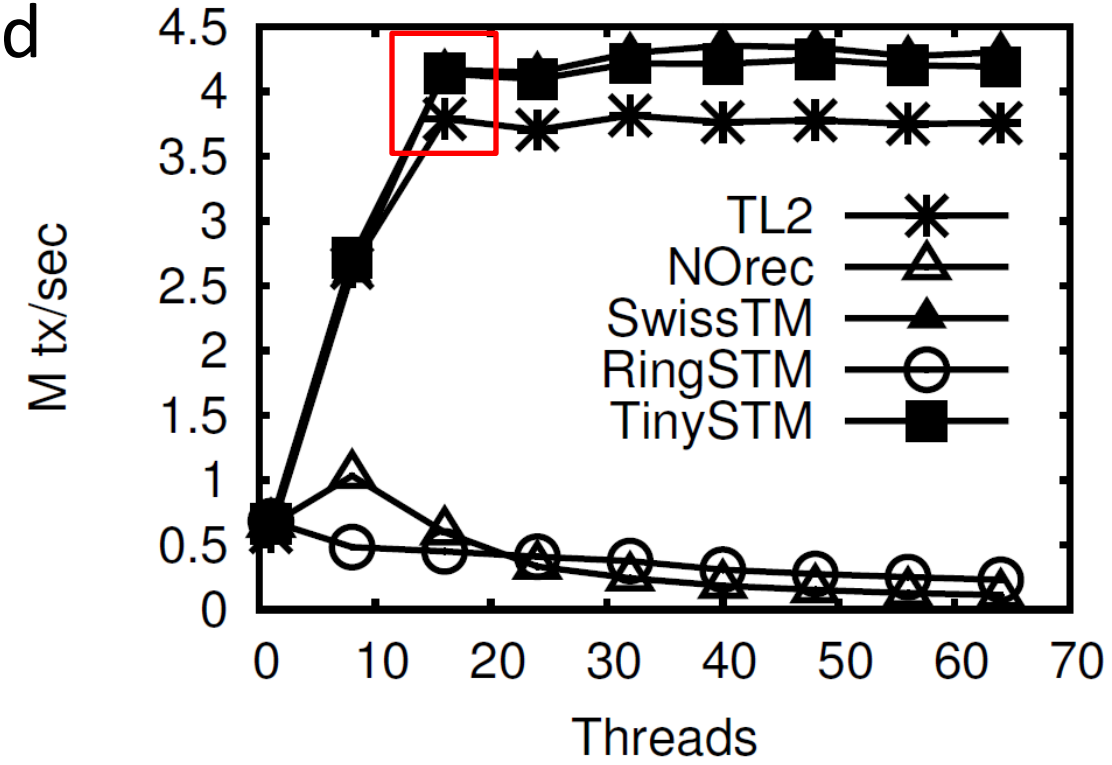
Our case study

- Our NUMA machine (64 cores)



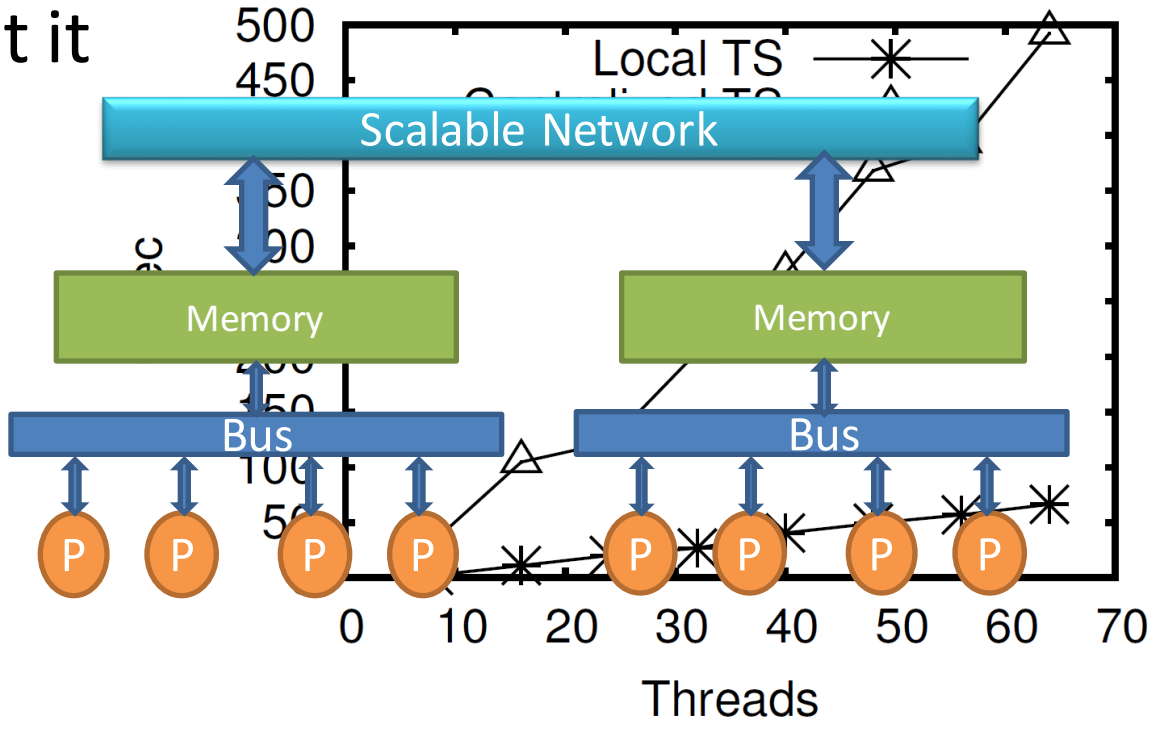
Nemo

- Most TM algorithms were designed for UMA architectures
  - Do not scale when deployed on NUMA architectures
  - Must be redesigned



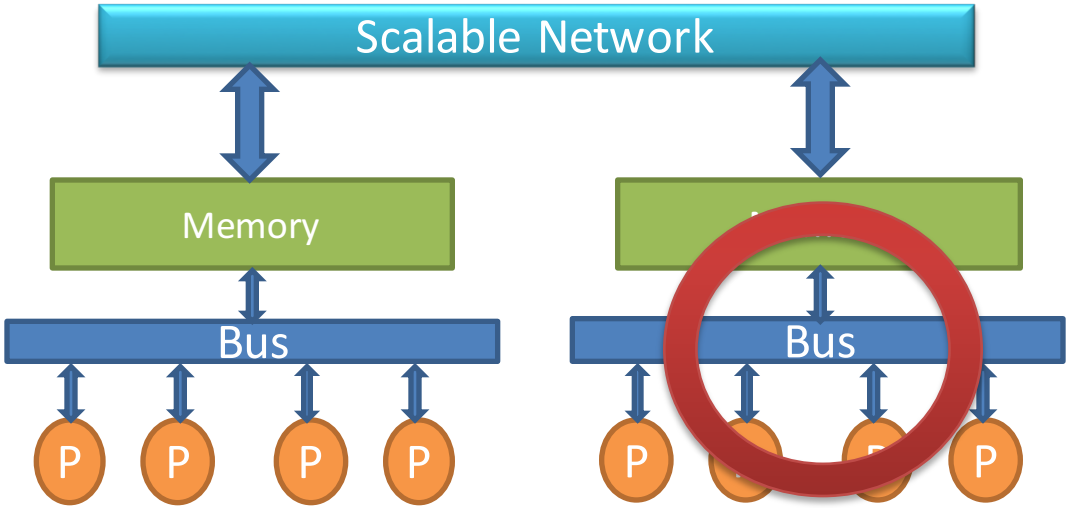
NUMA architectures properties

- Inter-NUMA communication is costly
  - Limit it
- Intra-NUMA communication is efficient
  - Exploit it



NUMA architectures properties

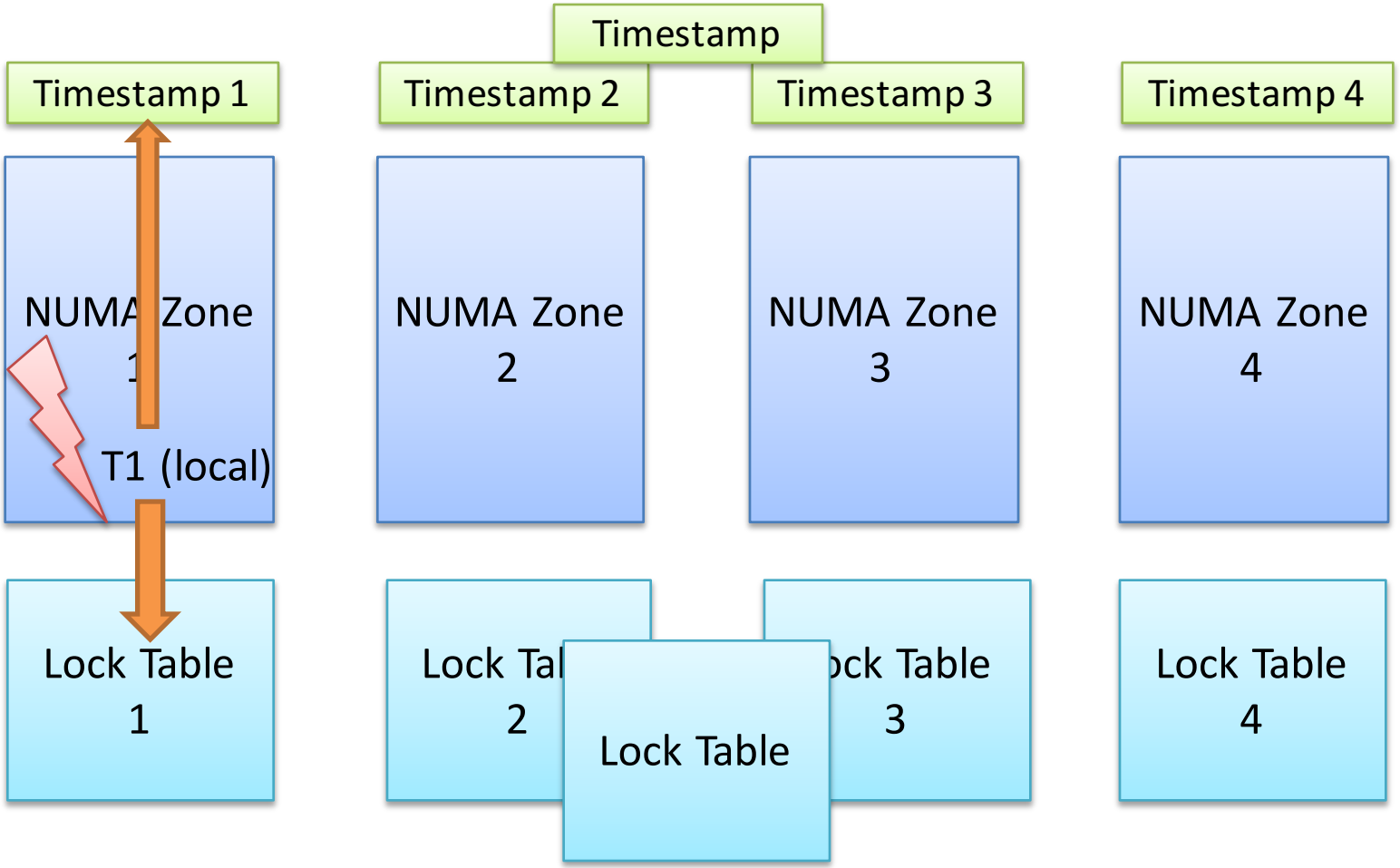
- Inter-NUMA communication is costly
  - Limit it
- Intra-NUMA communication is efficient
  - Exploit it



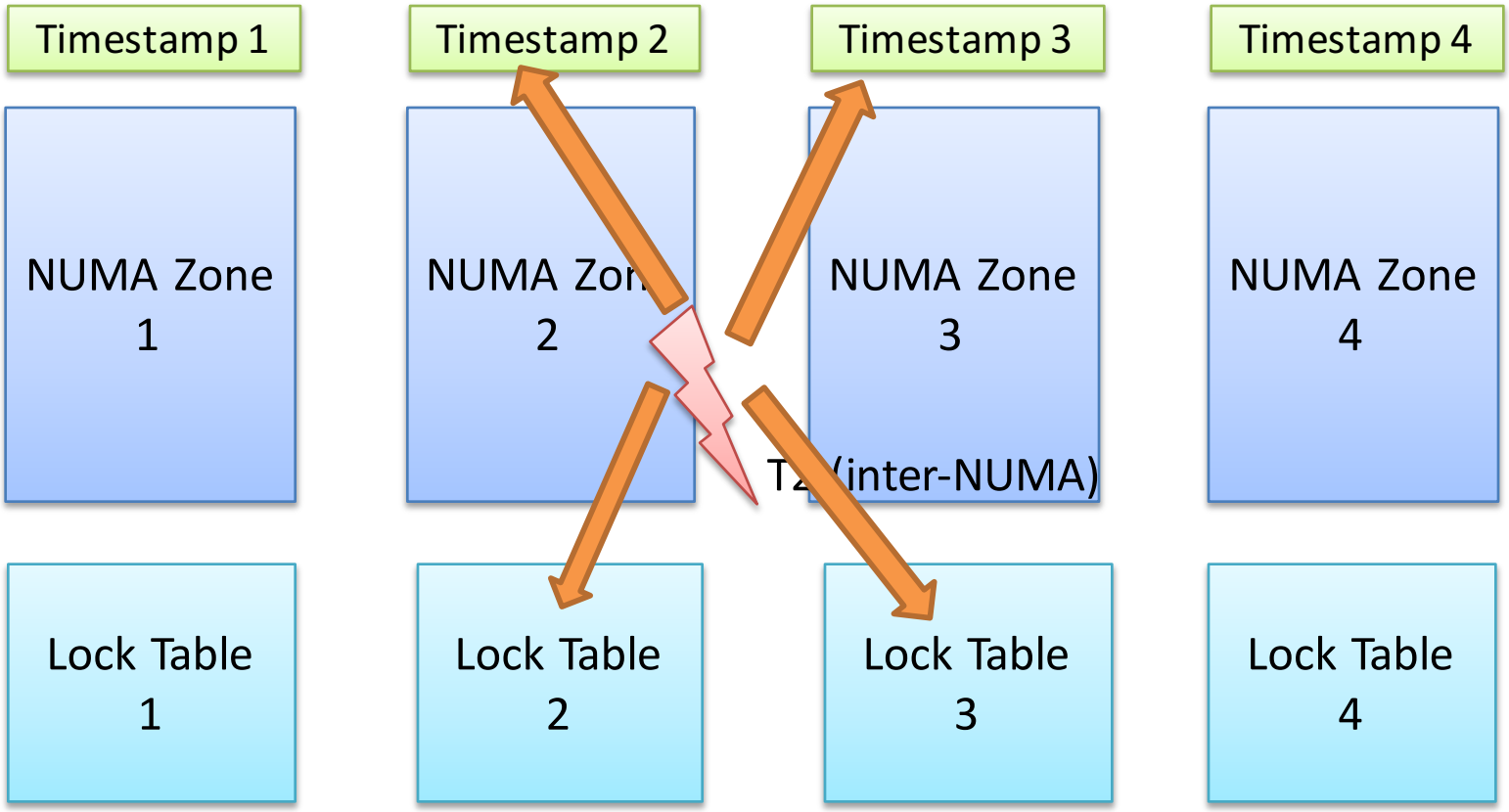
Nemo Idea

- Given a scalable workload (NUMA-local)
  - Run intra-NUMA zone transactions (common case) using efficient centralized STM
    - Can share meta-data
    - Isolated from other NUMA zones
  - Run inter-NUMA zones transactions without sharing any unnecessary meta-data
    - Threads communicate only when needed
      - Accessing common objects
    - Slower but still efficient

In Details


Timestamp
Timestamp 1
Timestamp 2
Timestamp 3
Timestamp 4
NUMA Zone 1
NUMA Zone 2
NUMA Zone 3
NUMA Zone 4
T1 (local)
Lock Table 1
Lock Table 2
Lock Table
Lock Table 3
Lock Table 4

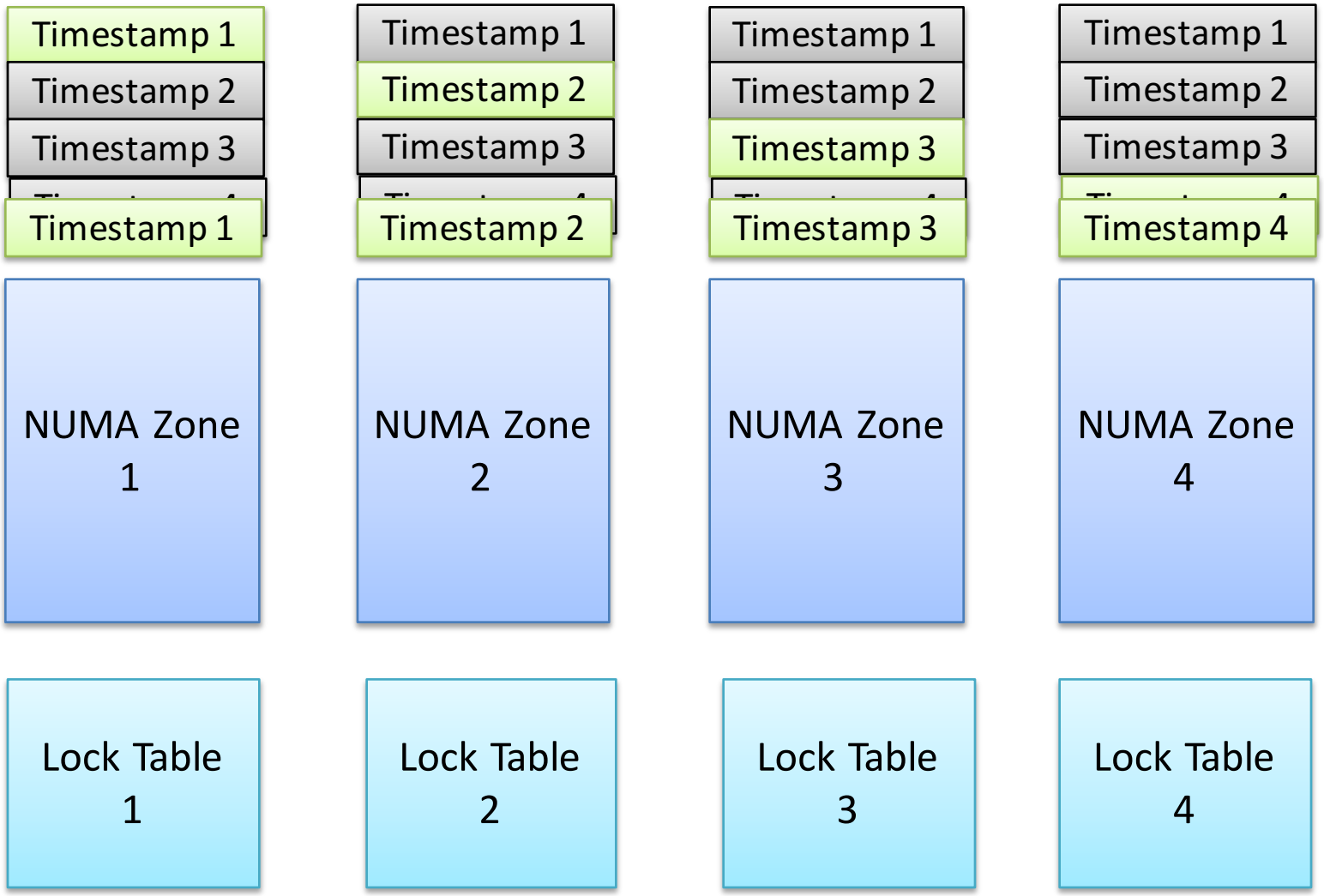
In Details


Timestamp 1
Timestamp 2
Timestamp 3
Timestamp 4
NUMA Zone 1
NUMA Zone 2
NUMA Zone 3
NUMA Zone 4
(inter-NUMA)
Lock Table 1
Lock Table 2
Lock Table 3
Lock Table 4

Nemo-TS

- One timestamp per NUMA zone
  - NUMA-local transactions
- Each thread keeps a local cache of other zones' timestamps
  - Refresh it probabilistically every 1/50
- Read object's version ≥ Local cache → Abort + Update cache
- At commit, atomically increment all accessed NUMA zones' timestamps
  - Update written objects' versions

Nemo-Vector

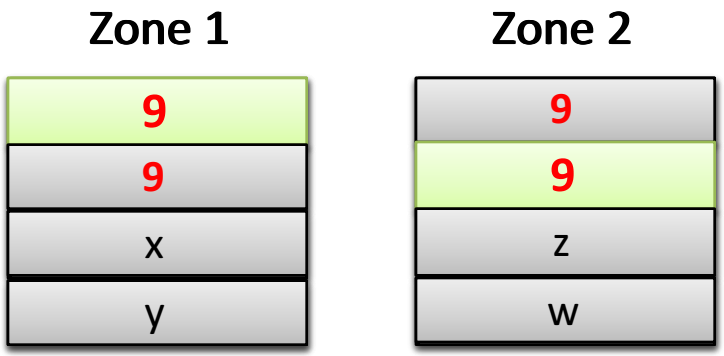

Timestamp 1
Timestamp 2
Timestamp 3
Timestamp 1
Timestamp 1
Timestamp 2
Timestamp 3
Timestamp 2
Timestamp 1
Timestamp 2
Timestamp 3
Timestamp 3
Timestamp 1
Timestamp 2
Timestamp 3
Timestamp 4
NUMA Zone 1
NUMA Zone 2
NUMA Zone 3
NUMA Zone 4
Lock Table 1
Lock Table 2
Lock Table 3
Lock Table 4

Nemo-Vector (2)

- Cache is now at the zone's level
  - Updated more frequently
- Avoid some false aborts cases
  - T_1 on z_1 Reading object X_{z_2} for the first time & $Ver(X_{z_2}) \geq ST_{T_1}[2]$ → Abort
  - But, $VCZ_2[1] \leq ST_{T_1}[1]$ means that z_2 did not invalidate any of z_1 Objects since T_1 started.
  - If this condition $VCZ_2[w] \leq ST_{T_1}[w]$ applies for all z_w written by T_1 then no abort is needed
    - Update $ST_{T_1}[2]$ and continue

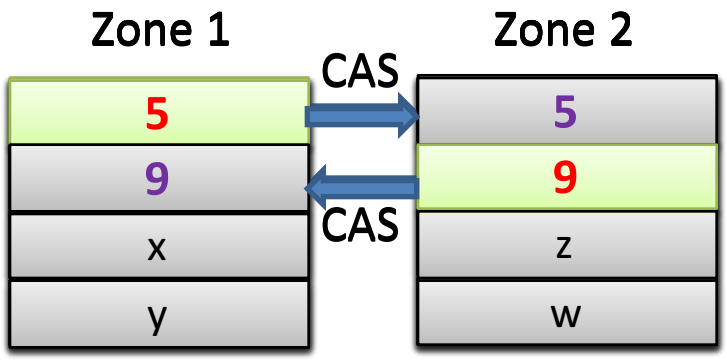
Nemo-Vector (3)

- At begin copy zone's vector to thread starting-time
- At commit, all accessed zones vector clocks are updated



Nemo-Vector (3)

- At begin copy zone's vector to thread starting-time
- At commit, all accessed zones vector clocks are updated

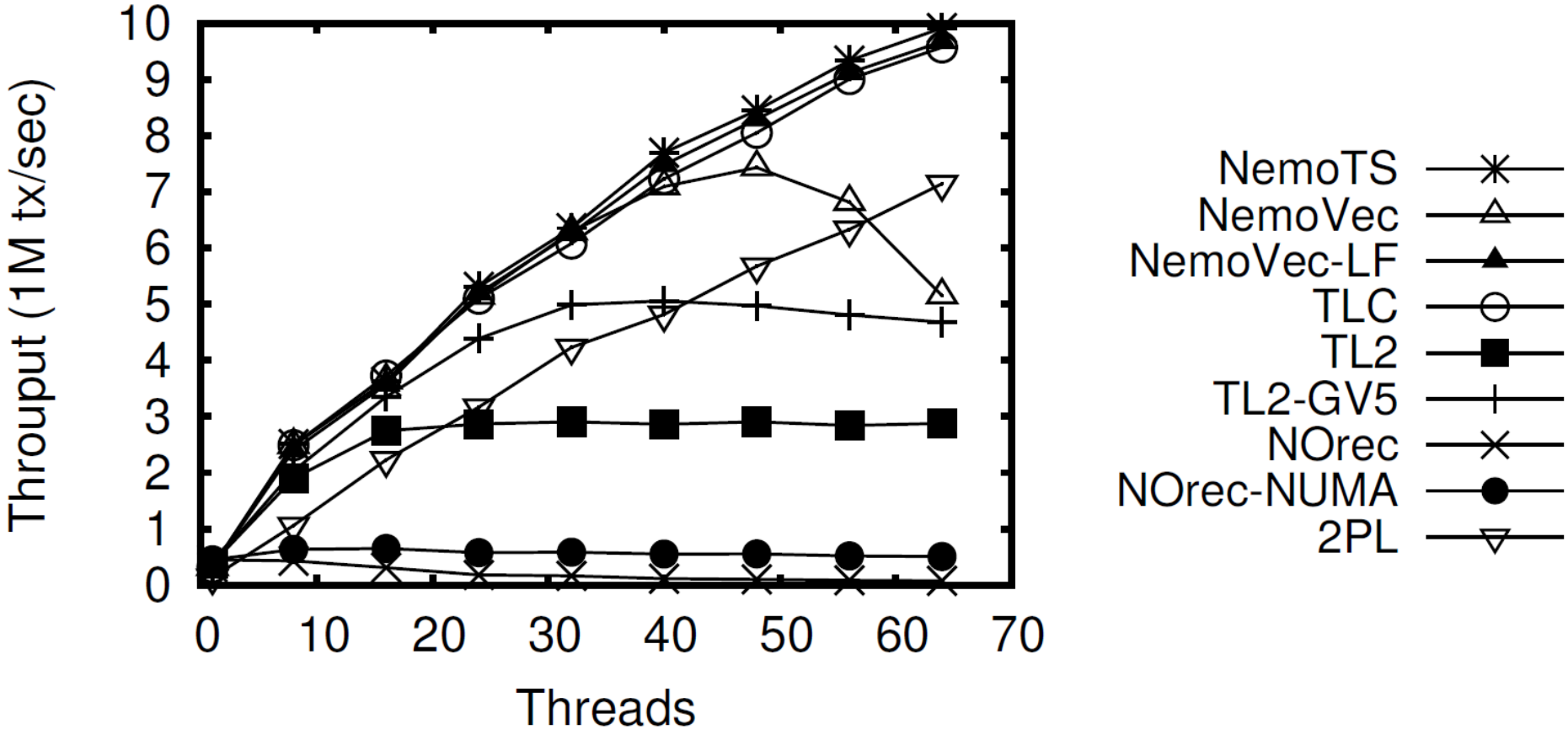


Evaluation

- Implemented in C++
- Tested on 64-cores AMD Opteron 6376 (4 sockets, 8 NUMA zones). Memory: 16 GB/zone
- Compiler: GCC 4.8.2, OS: Ubuntu 14.04 LTS (libnuma)
- Benchmarks: Bank, Linked-List, TPC-C
- Competitors: TL2, NOrec, TLC, TL2-GV5, 2-Phase locking, Norec + NUMA-aware lock
- Available as an open-source library

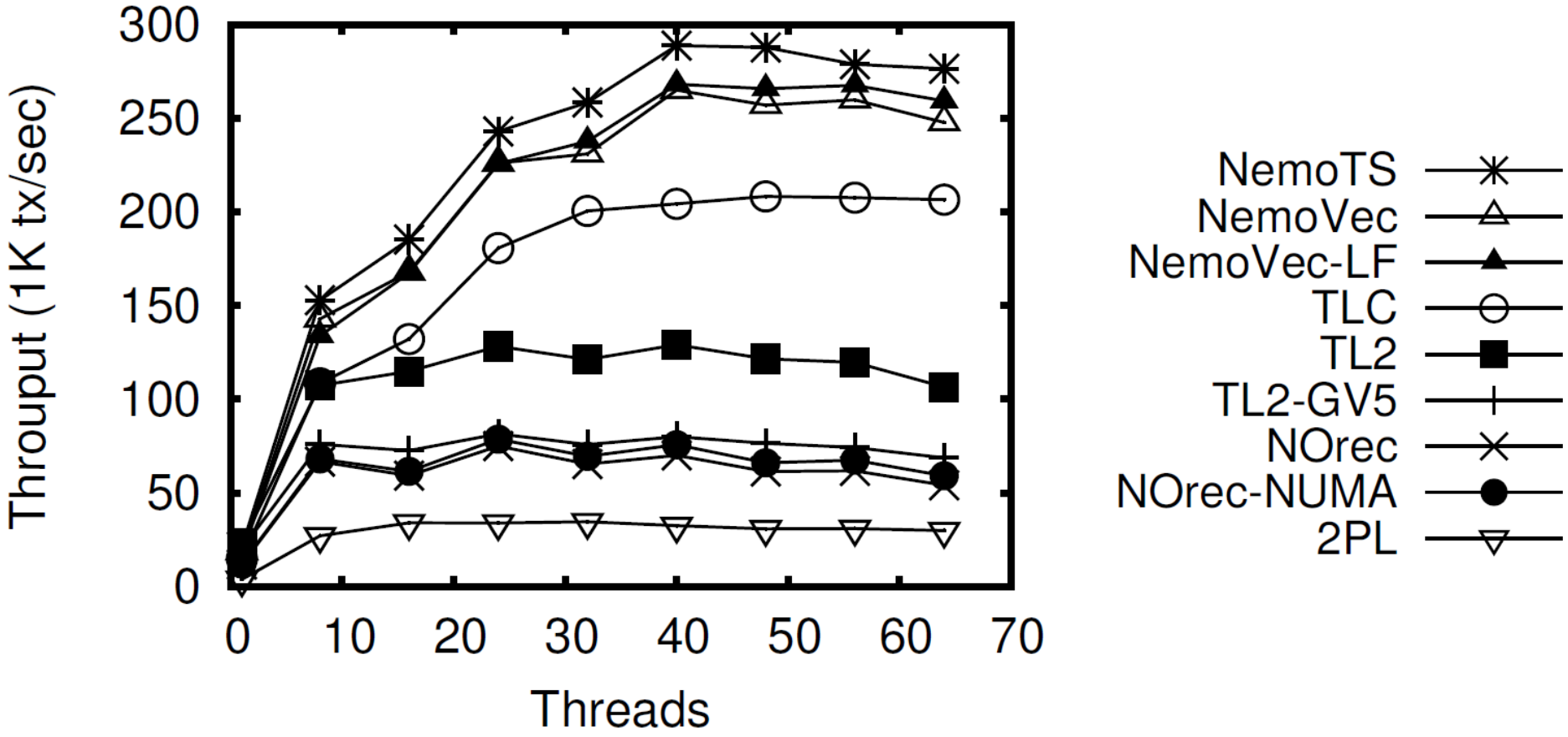
Evaluation: Bank

- 1M accounts/zone. 10% inter-NUMA transactions
- Very low contention level



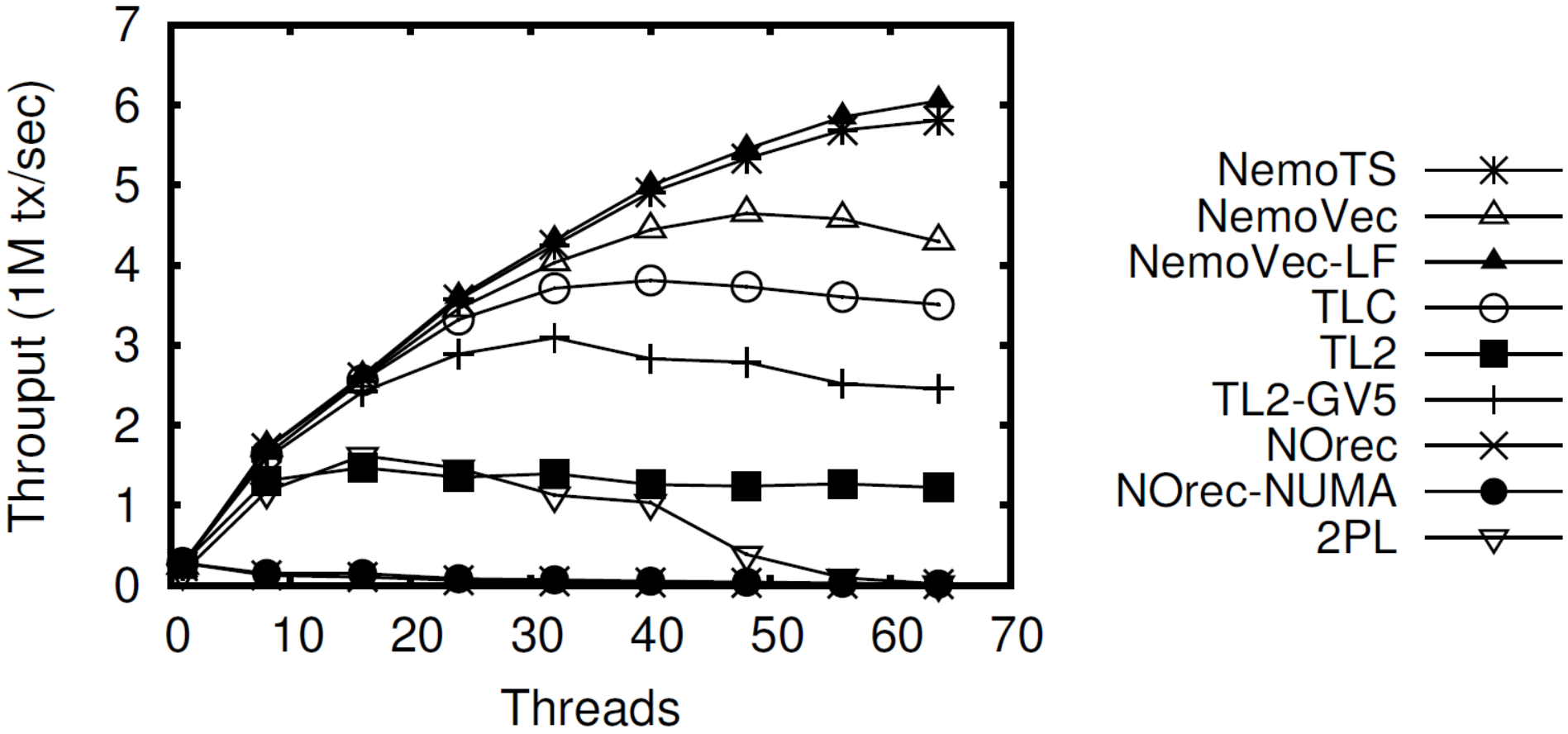
Evaluation: Linked-List

- One ordered linked-list per zone
- High level of contention



Evaluation: TPC-C

- Database is partitioned (20 warehouse/zone)
- Moderate level of contention



Nemo & HTM

- Is Nemo addressing a different problem?
  - No, but why?
- Intel released Haswell-EX
  - Supports up to 8 sockets
    - Currently, the latest configuration has **72** cores (144 threads) on 4 sockets (May 2015)
  - Still using cache coherent protocol
    - Uses NUMA architecture (Intel QuickPath Interconnect)
  - Thus, it will suffer from the same NUMA problems
    - We did the work on STM
    - Future work: Enhance the SW fallback path
      - Possible direction: Merge Precise-TM and Nemo

Conclusions

- Dissertation contributions
  - Part-HTM
    - Resource limitations of HTM
  - Octonauts
    - First HTM-aware scheduler
  - Precise-TM
    - First practical fine-grained fallback path technique
  - Nemo
    - NUMA-aware HTM
- Not in dissertation
  - Shield
  - SoftX

Future Work

- Optimize HTM software fallback bath for NUMA-HTM machines
  - Merge Precise-TM with Nemo
- Provide an efficient STM-STM synchronization technique using Precise-TM technique
  - Apply Precise-TM on existing HyTM algorithms

Thank You

- List of Publications:
  - *On Preserving Data Integrity of Transactional Applications on Multicore Architectures*, M. Mohamedin, R. Palmieri, and B. Ravindran, The 35th International Conference on Distributed Computing Systems (**ICDCS**), Short Paper, June 2015, Columbus, Ohio, USA
  - *Brief Announcement: Managing Resource Limitation of Best-Effort HTM*, M. Mohamedin, R. Palmieri, A. Hassan and B. Ravindran, 27th ACM Symposium on Parallelism in Algorithms and Architectures (**SPAA**), June 2015, Portland, Oregon, USA
  - *Brief Announcement: On Scheduling Best-Effort HTM Transactions,* M. Mohamedin, R. Palmieri and B. Ravindran, 27th ACM Symposium on Parallelism in Algorithms and Architectures (**SPAA**), June 2015, Portland, Oregon, USA
  - *On Making Transactional Applications Resilient to Data Corruption Faults*, M. Mohamedin, R. Palmieri and B. Ravindran, 13th IEEE International Symposium on Network Computing and Applications (**IEEE NCA14**), August 21-23, 2014 Boston, USA
  - *Managing Soft-errors in Transactional Systems*, M. Mohamedin, R. Palmieri and B. Ravindran, 19th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (**DPDNS 2014**), May 23, 2014, Phoenix, Arizona, USA
- Under Review:
  - *Managing Resource Limitation of Best-Effort HTM*, M. Mohamedin, R. Palmieri, A. Hassan and B. Ravindran, The 24th International Conference on Parallel Architectures and Compilation Techniques (**PACT**), October 2015, San Franciso, CA, USA
- In Preparation for Submission:
  - *Precise-TM* is planned for 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (**PPoPP**), March 2016, Barcelona, Spain
  - *Nemo* is planned for 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (**PPoPP**), March 2016, Barcelona, Spain