Improved Single Global Lock Fallback for Best-effort Hardware Transactional Memory

TRANSACT 2014

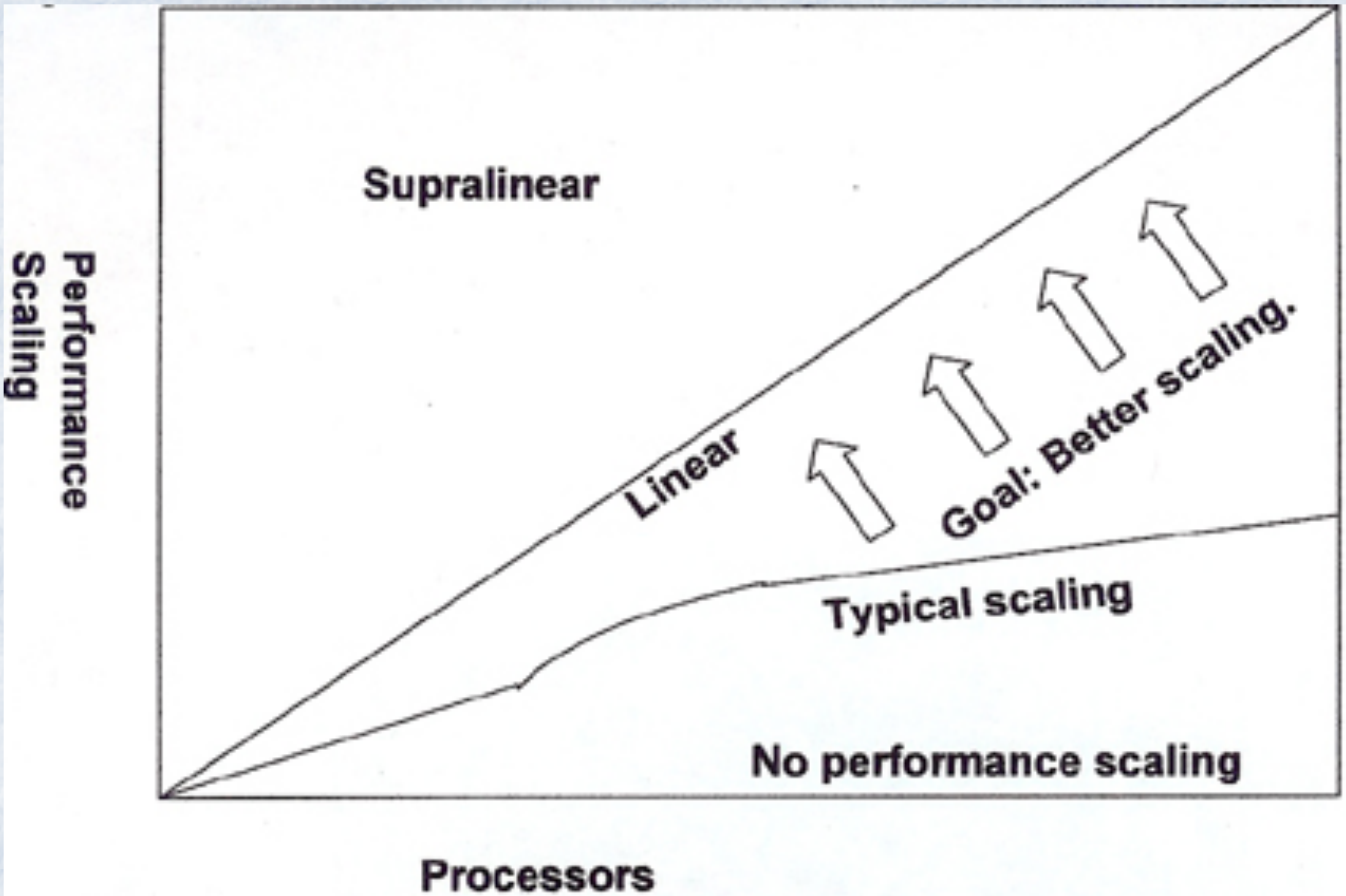
Irina Calciu

Tatiana Shpeisman

Gilles Pokam

Maurice Herlihy

Multicore Performance Scaling



Hardware Transactional Memory (HTM)

IBM's Blue Gene/Q & System Z & Power Architecture

Intel's Haswell TSX: RTM & HLE

Low overhead (cache based)

Haswell RTM

if (_xbegin() == _XBEGIN_STARTED)

Speculate Execution

_xend()

else

Abort Handler

Speculate Execution, without any locks

Read and Write Sets

Abort on memory conflict

Haswell RTM

if (_xbegin() == _XBEGIN_STARTED)

Speculate Execution

_xend()

_xbegin()

Read X → Add to Read Set

Write Y → Add to Write Set

COMMIT

_xend() → Make the change to Y visible

_xbegin()

Add to Write Set ← Write X

ABORT

Add to Write Set ← Write Y

_xend()

Lock Elision

<HLE_Aquire_Prefix> Lock(L)

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

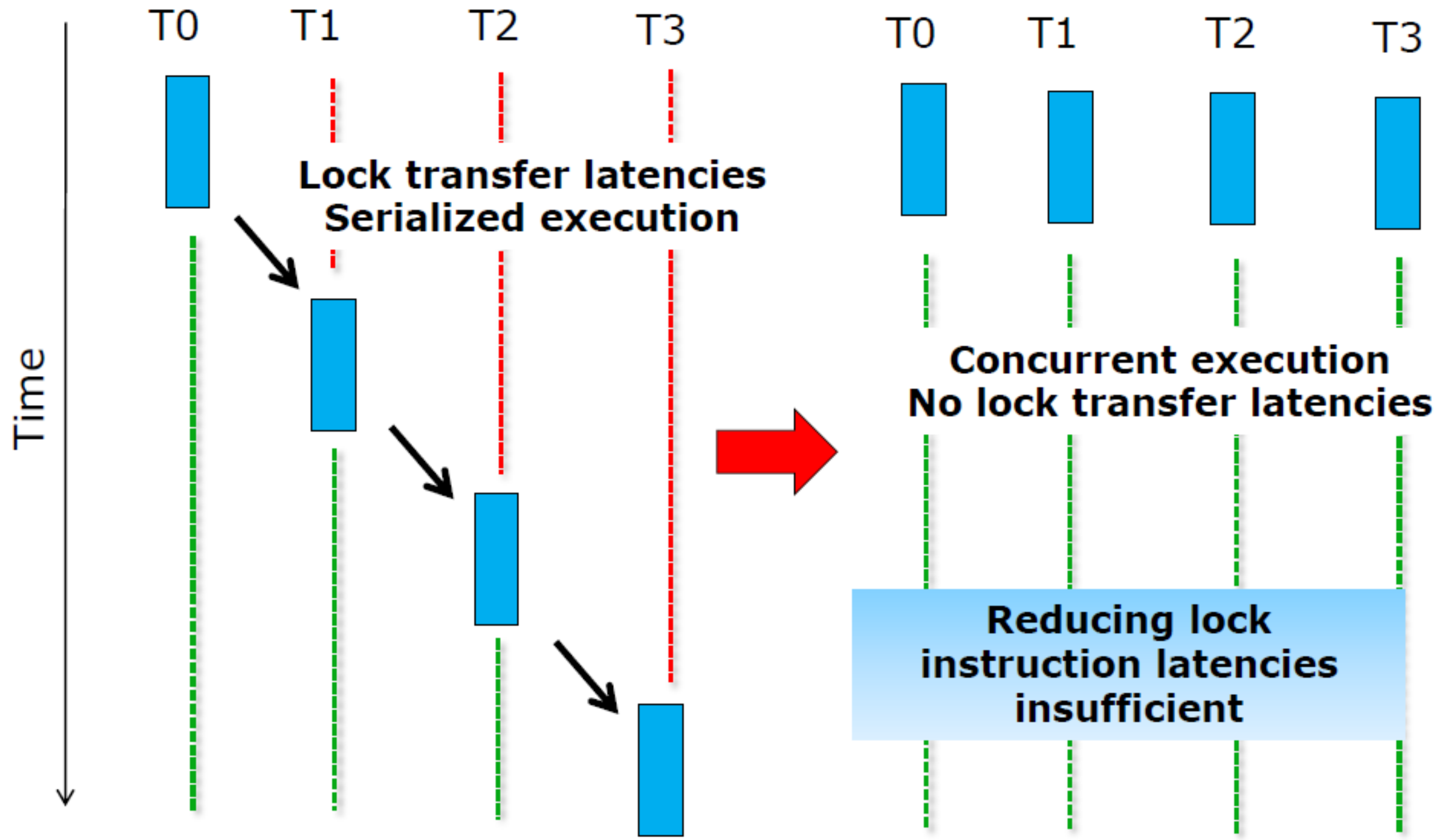
<HLE_Release_Prefix> Release(L)

Execute optimistically, without any locks

Track Read and Write Sets

Abort on memory conflict: rollback acquire lock

Lock Elision



[Anand Tech]

Haswell RTM

Small & Medium
Transactions

Best-effort

Conflicts

Overflow

Unsupported
Instructions

Interrupts

Needs software fallback

Overview

- Best-effort Hardware Transactional Memory

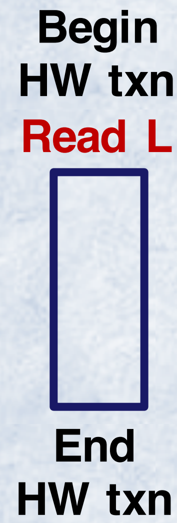
- Lazy SGL
  - Description
  - Correctness
  - Results

- Bloom Filter SGL

Single Global Lock HyTM (simple and common)

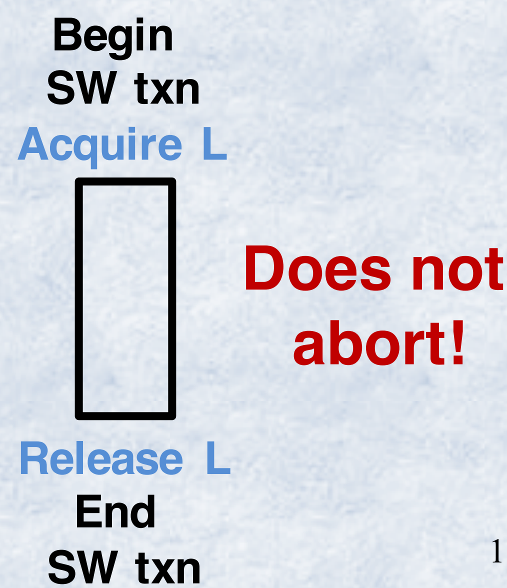
Try_SPEC:

- Wait until Lock is free
- **Transactional_Read(Lock)**
- **If Lock is taken ABORT**
- **Speculate critical section**
- End speculation

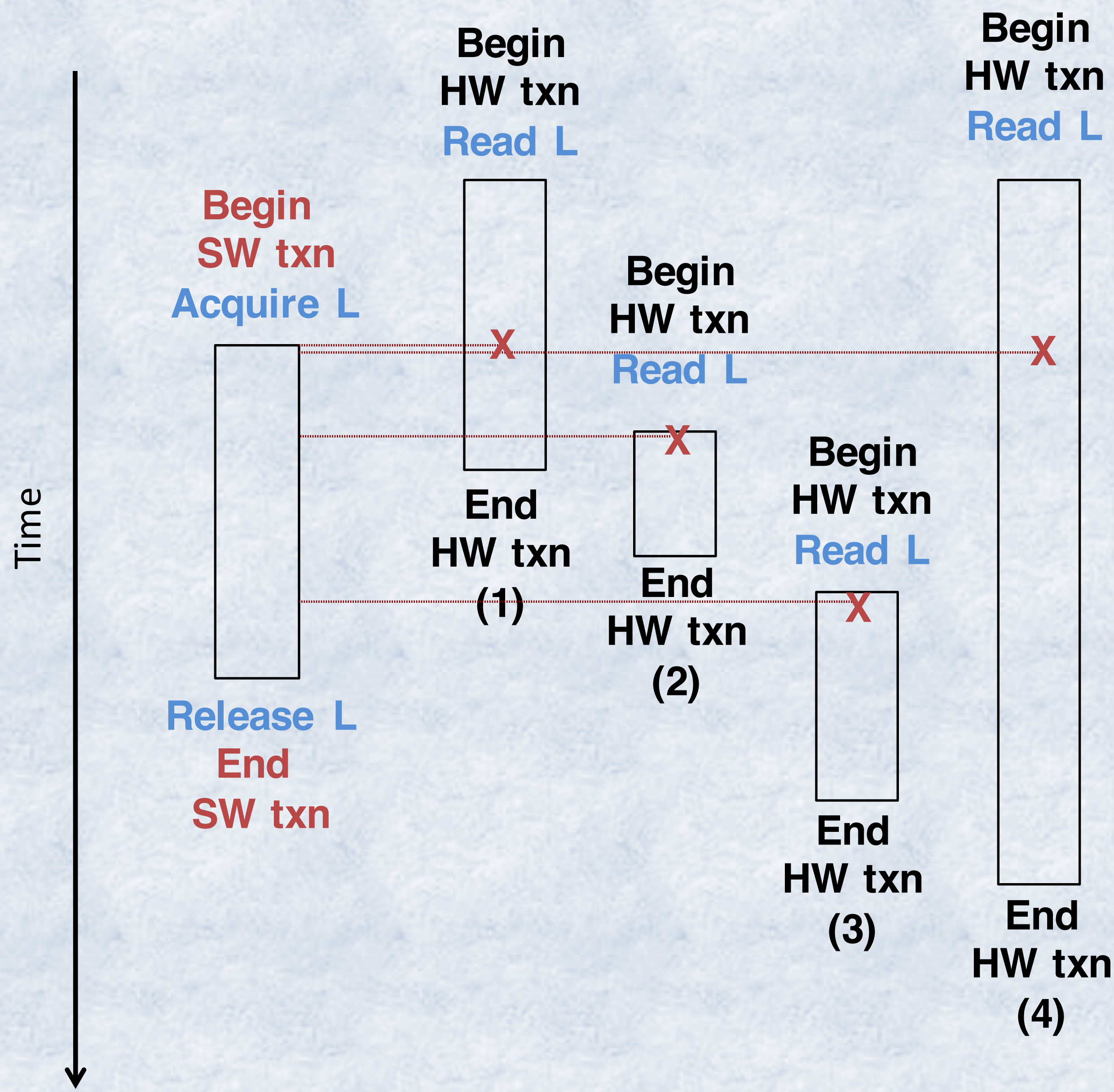


On_ABORT:

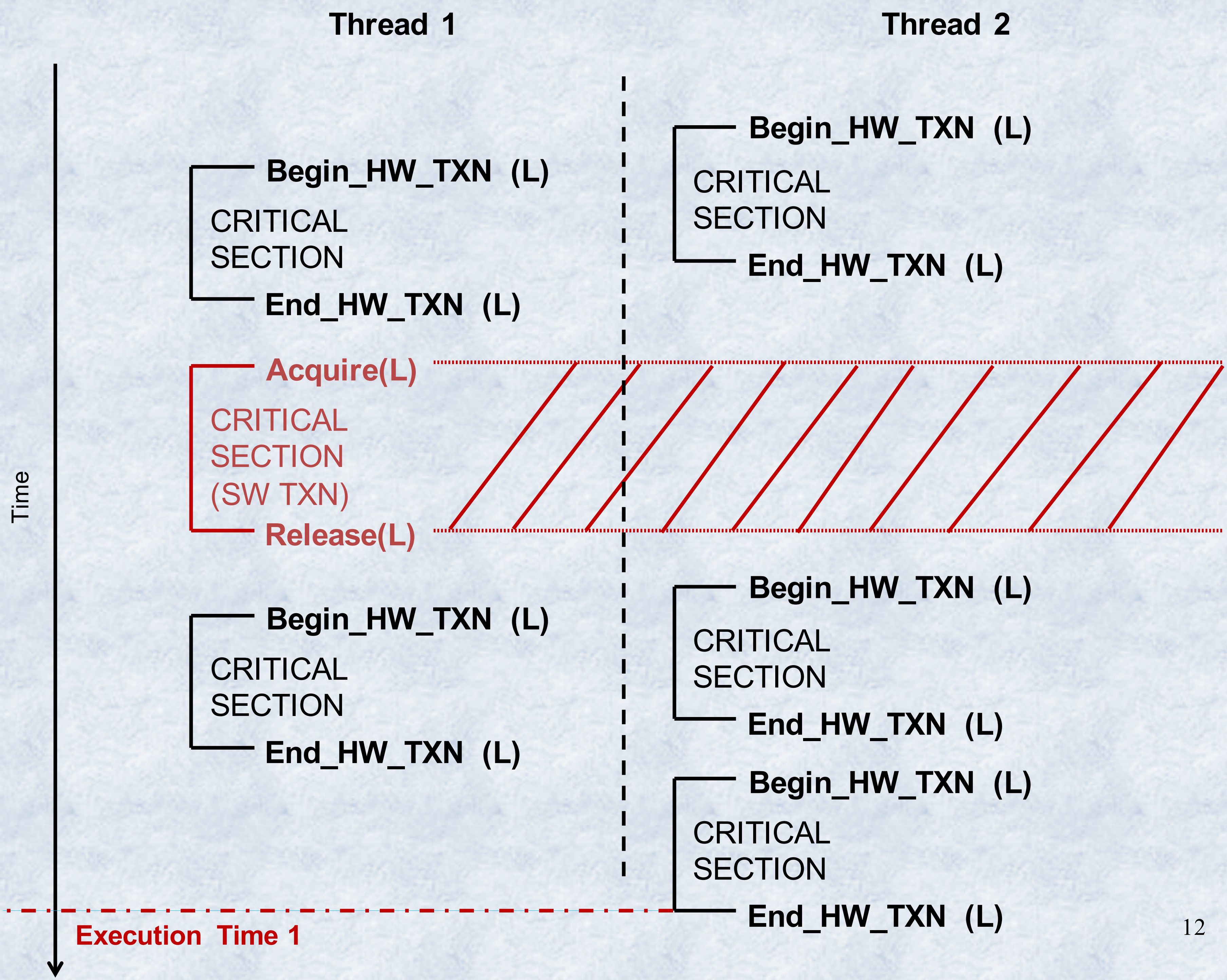
- If try_lock(Lock)
  - Critical section
  - Release(Lock)
- Else Try_SPEC

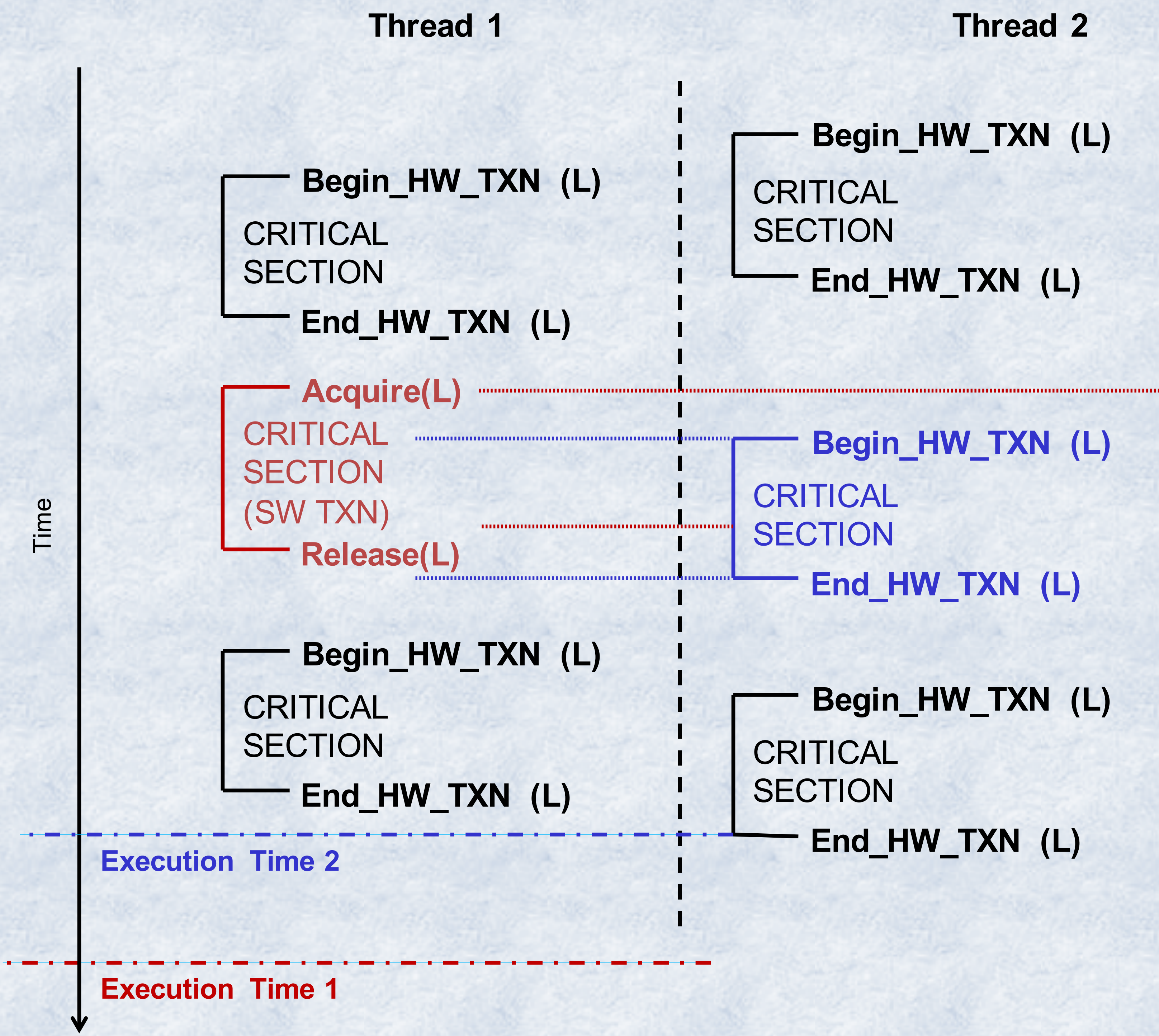


Single Global Lock HyTM (simple and common)



Legend:
X = ABORT


Thread 1
Thread 2
Time
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Acquire(L)
CRITICAL SECTION (SW TXN)
Release(L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Execution Time 1



Thread 1
Thread 2
Time
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Acquire(L)
CRITICAL SECTION (SW TXN)
Release(L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Begin_HW_TXN (L)
CRITICAL SECTION
End_HW_TXN (L)
Execution Time 2
Execution Time 1

Lazy SGL

Try_SPEC:

- **Speculate critical section**
- **Transactional_Read(Lock)**
- **If Lock is taken ABORT**
- End speculation

Begin HW txn

Read L

End HW txn

On_ABORT:

- If try_lock(Lock)
  - Critical section
  - Release(Lock)
- Else Try_SPEC

Begin SW txn

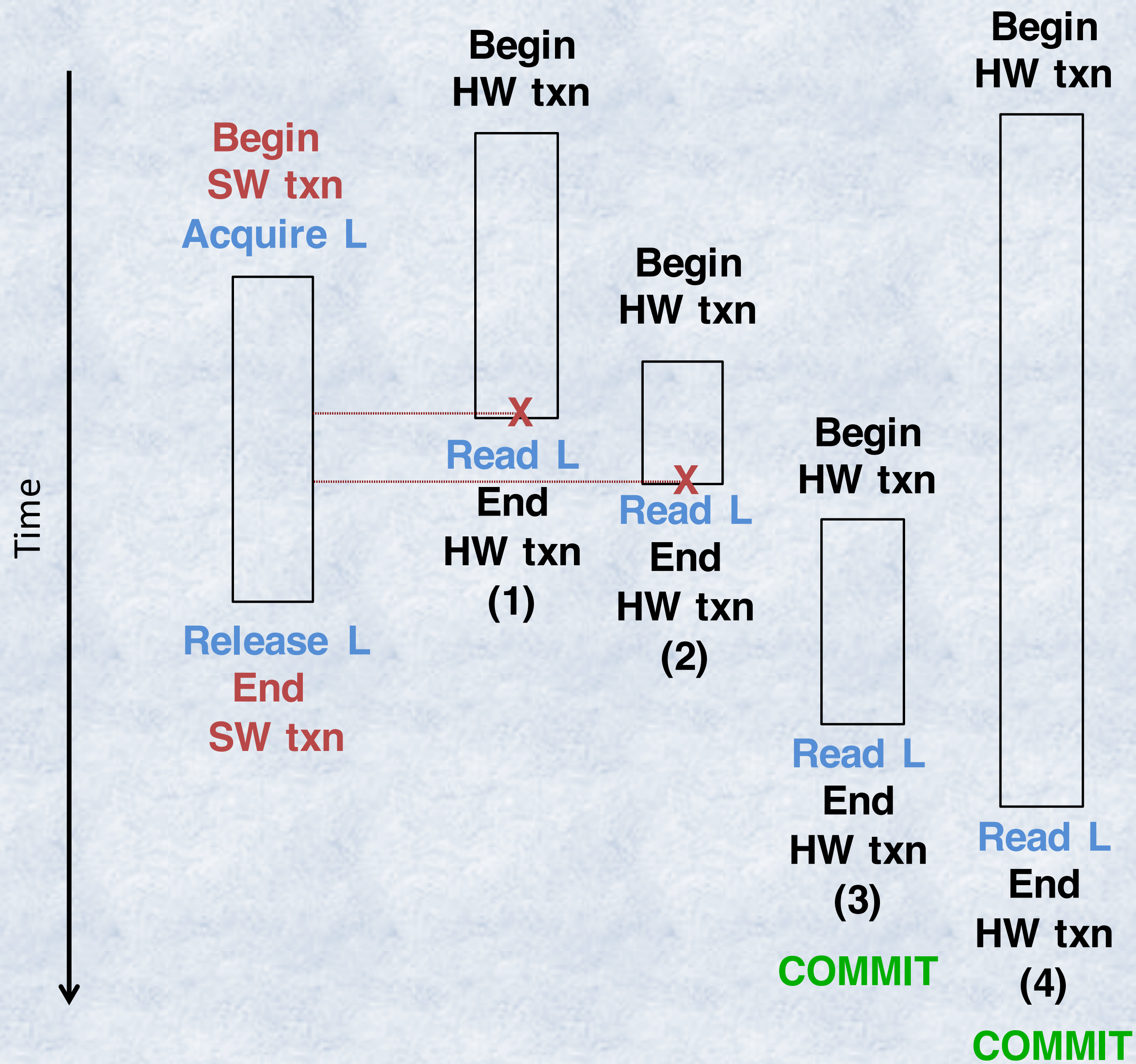
Acquire L

Release L

End SW txn

Does not abort!

Lazy SGL



Legend:
X = ABORT

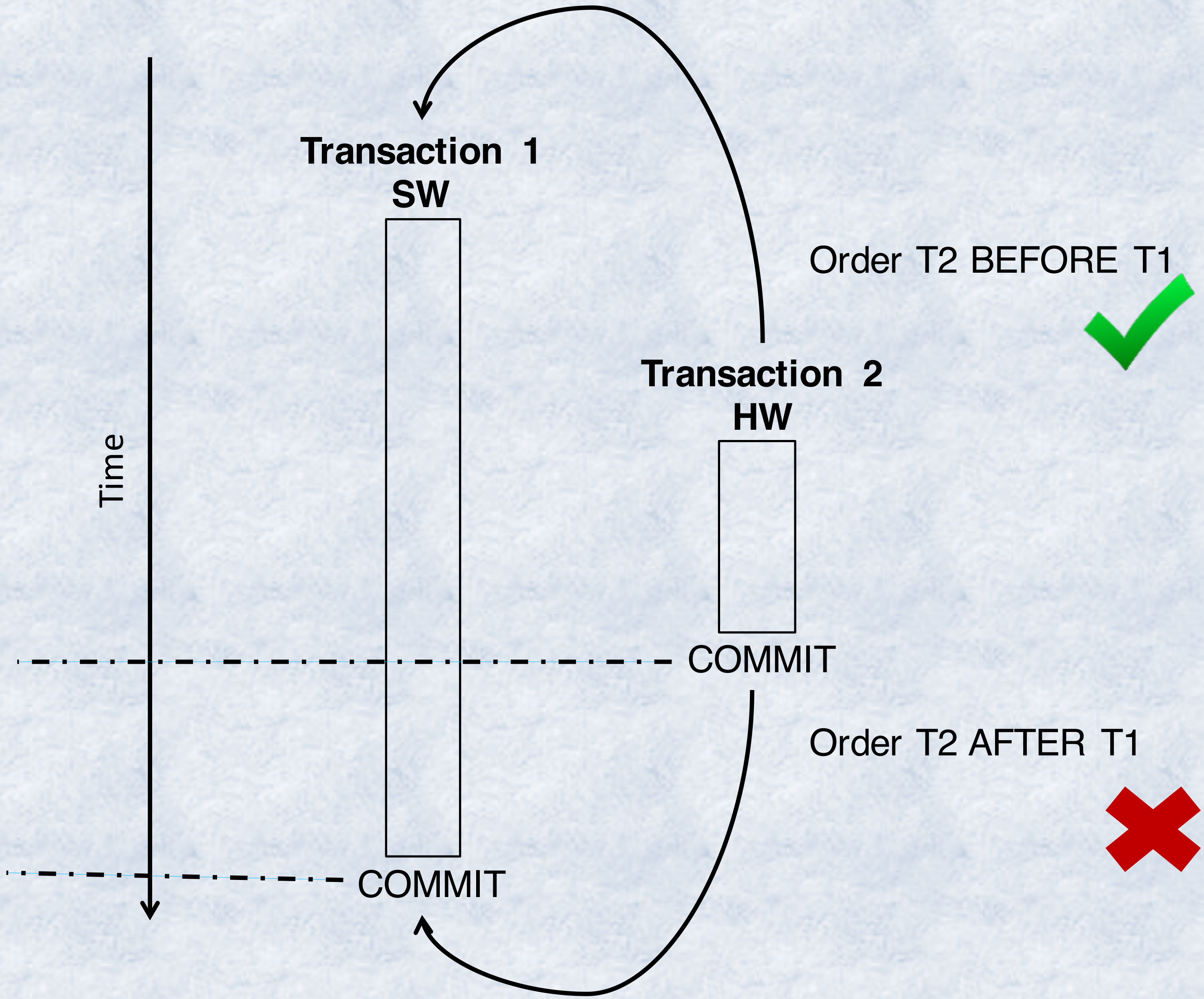
Overview

- Best-effort Hardware Transactional Memory

- Lazy SGL
  - Description
  - Correctness
  - Results

- Bloom Filter SGL

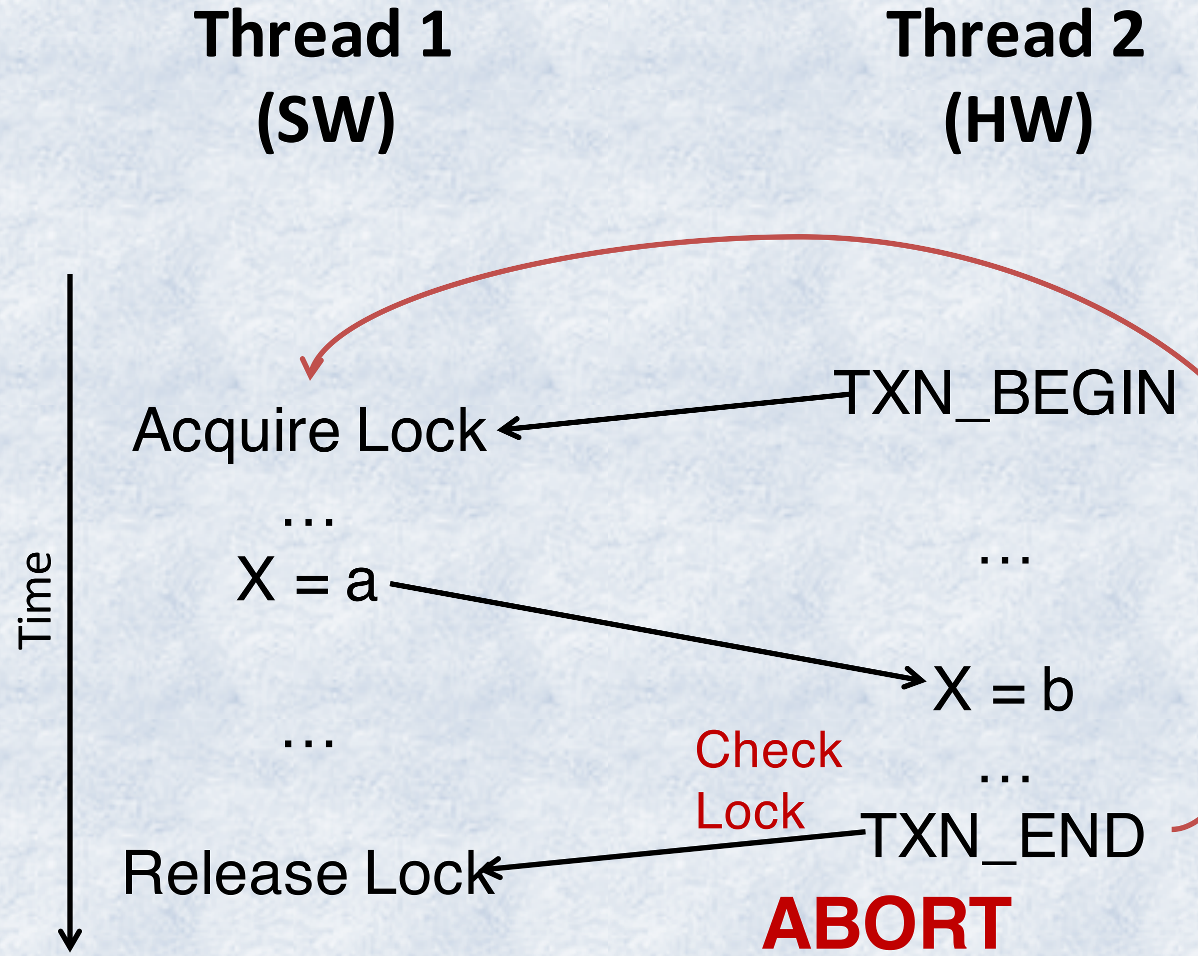
Transactional Memory Correctness



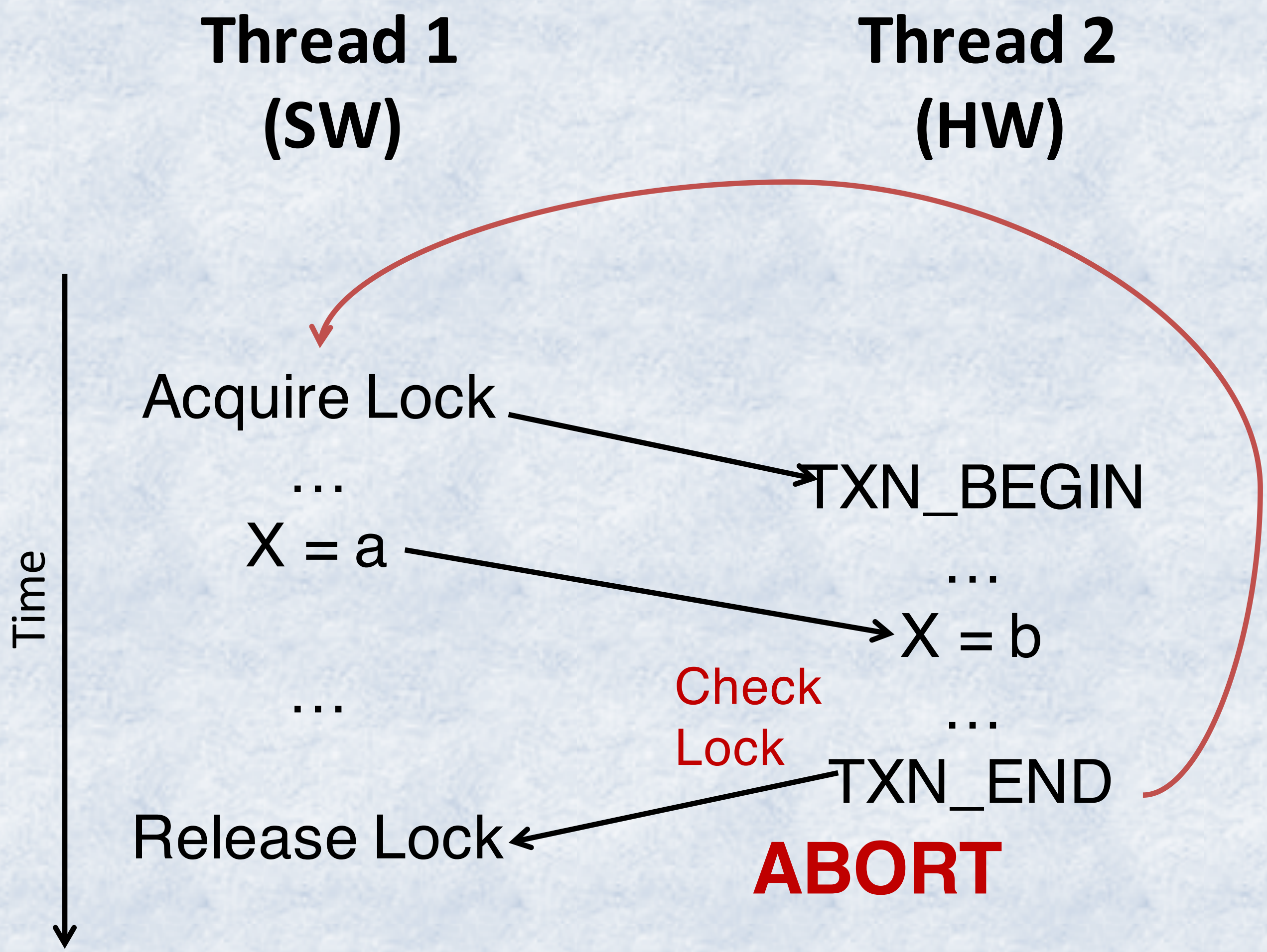
Case 1:
HW begins
SW begins
HW ends
SW ends

X value:

Correct: a
Actual: b



Case 2:
SW begins
HW begins
HW ends
SW ends



X value:

Correct: a
Actual: b ✖

Case 3:
SW begins
HW begins
SW ends
HW ends

Thread 1 (SW) | **Thread 2 (HW)**

Time

Thread 1 (SW)	Thread 2 (HW)
Acquire Lock	TXN_BEGIN
…	…
X = a	X = b
…	…
Release Lock	TXN_END

Check Lock

COMMIT

X value:

Correct: b
Actual: b

Case 4:
HW begins
SW begins
SW ends
HW ends

Thread 1 (SW)	Thread 2 (HW)
	TXN_BEGIN
Acquire Lock	
…	…
X = a	
	X = b
…	…
Release Lock	
	TXN_END

Check Lock

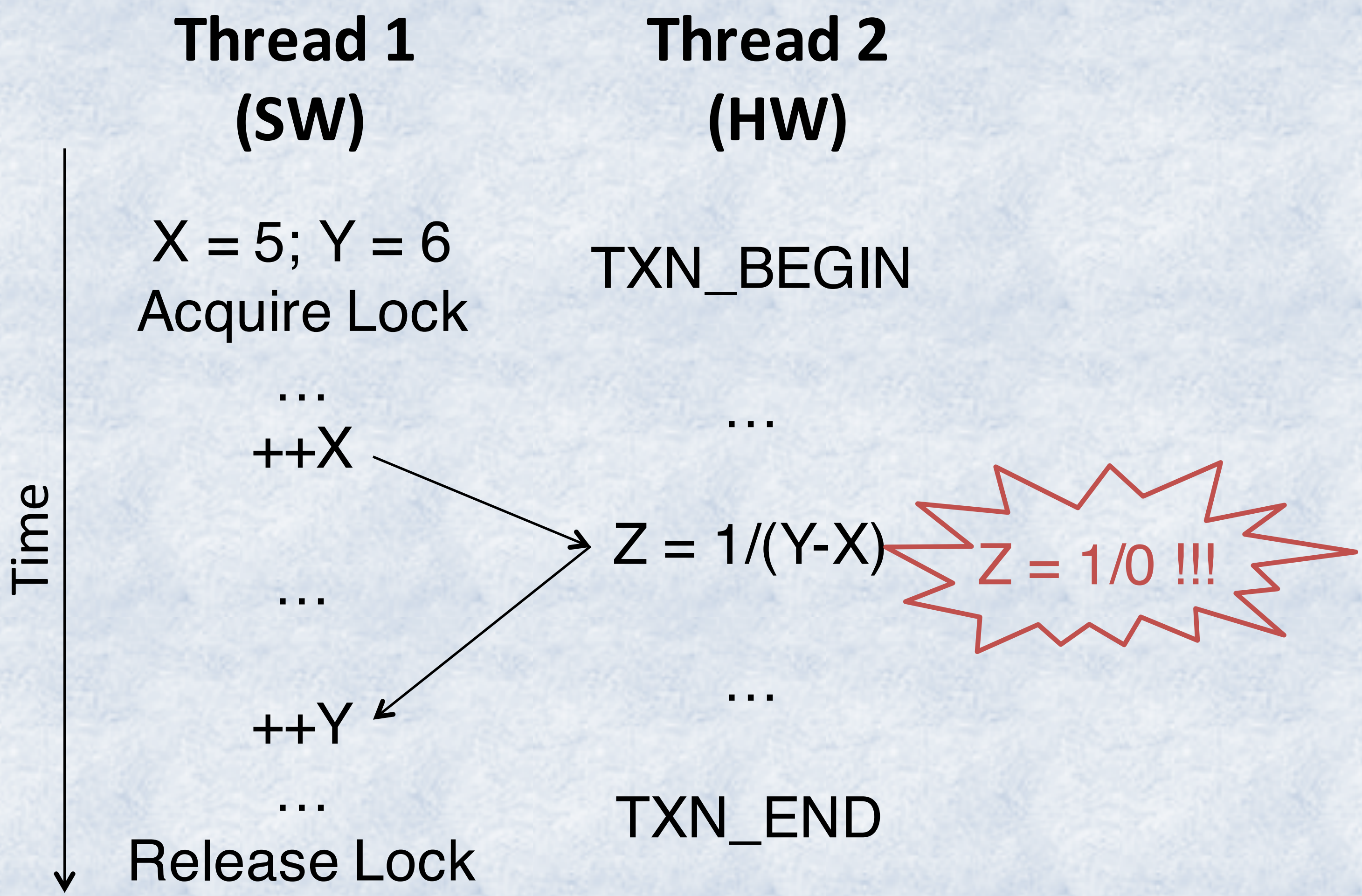
COMMIT

Time

X value:

Correct: b
Actual: b

Hardware Sandboxing



Indirect Jumps

Time

Thread 1 (SW)	**Thread 2 (HW)**
X = 5; Y = 6	_xbegin
Acquire Lock	
…	…
++X	
	if (X == Y) *p = garbage
…	p()
++Y	…
…	if (lock) abort
Release Lock	_xend

Indirect jump to garbage location

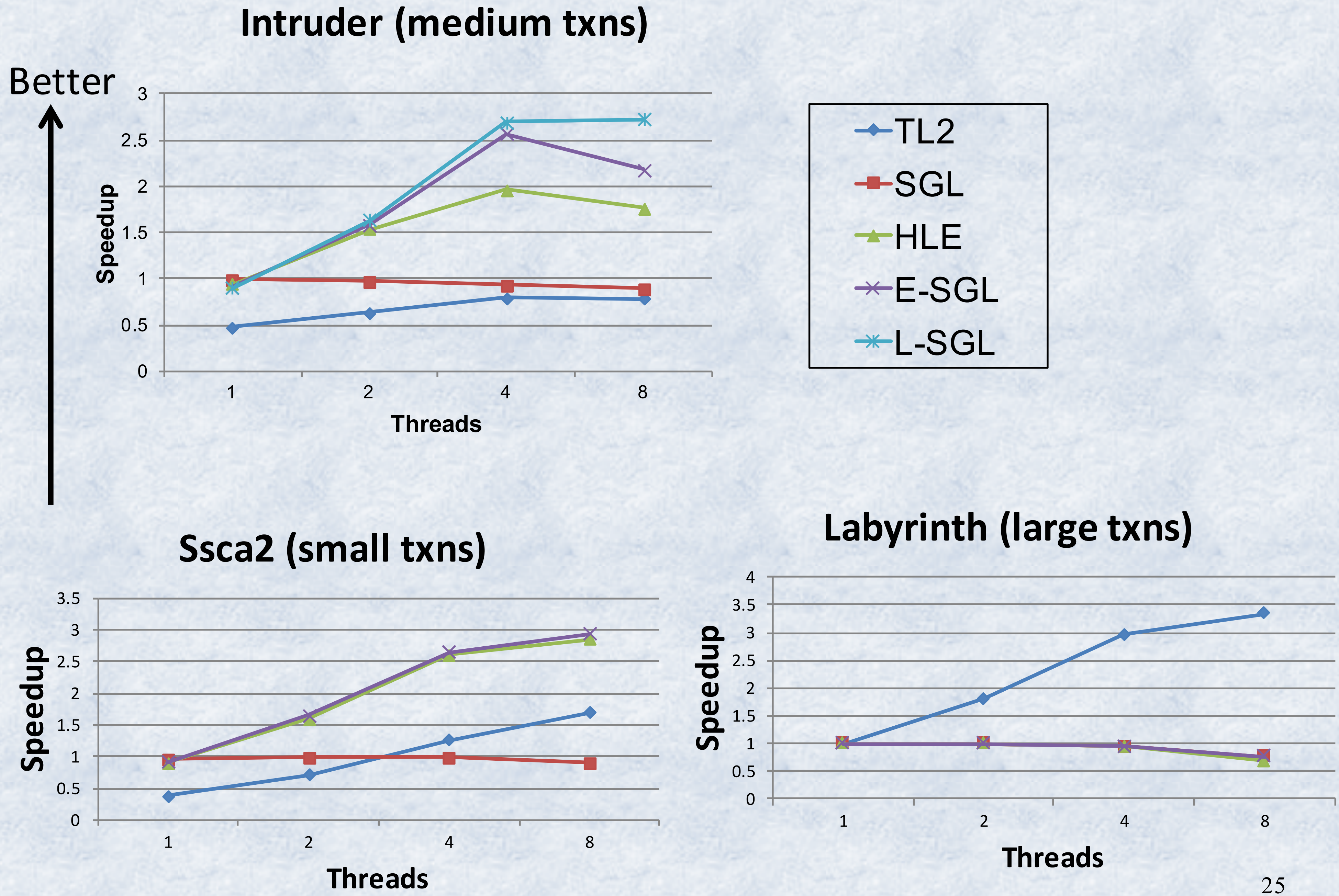
_xend

Overview

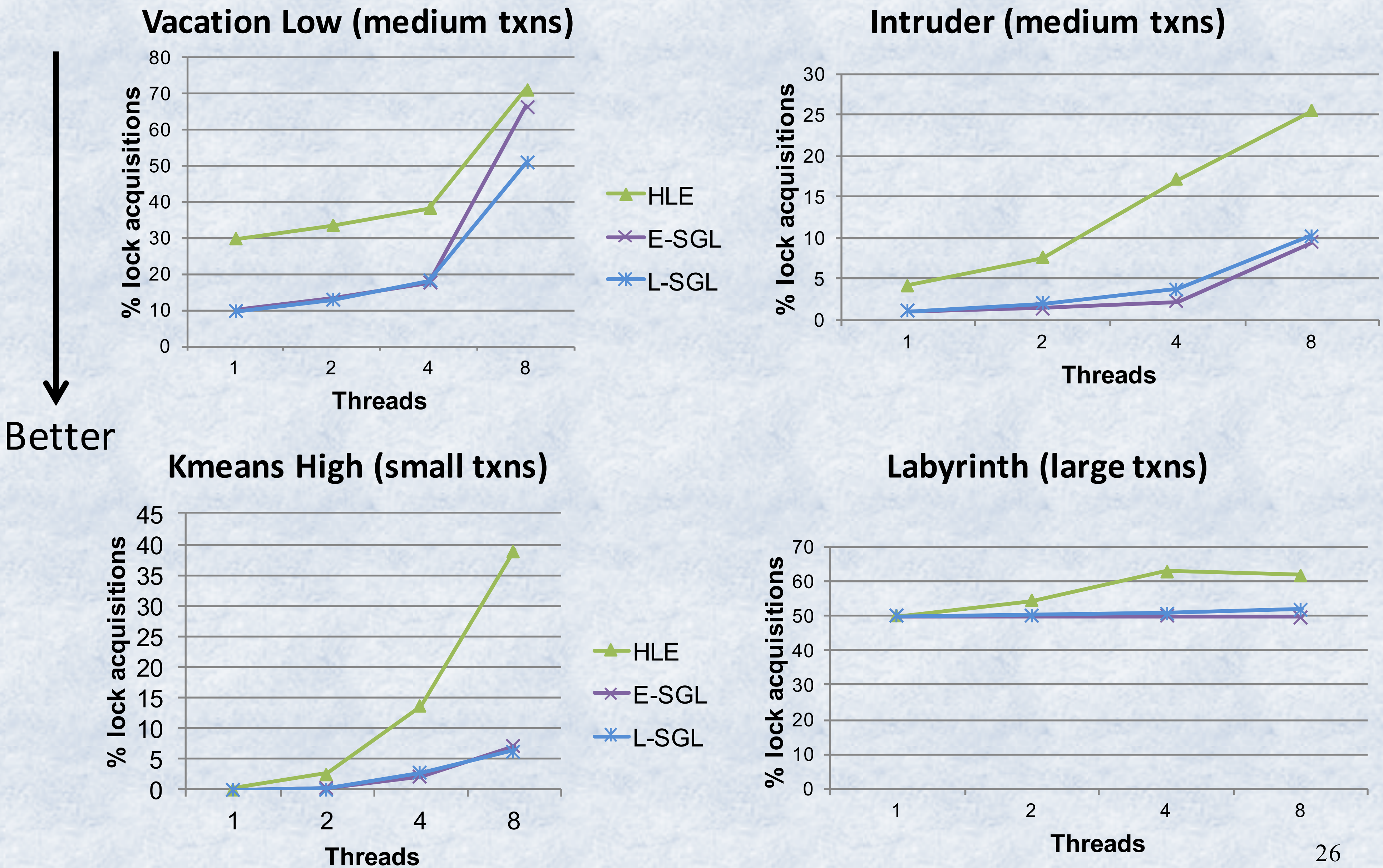
- Best-effort Hardware Transactional Memory

- Lazy SGL → Description, Correctness, Results

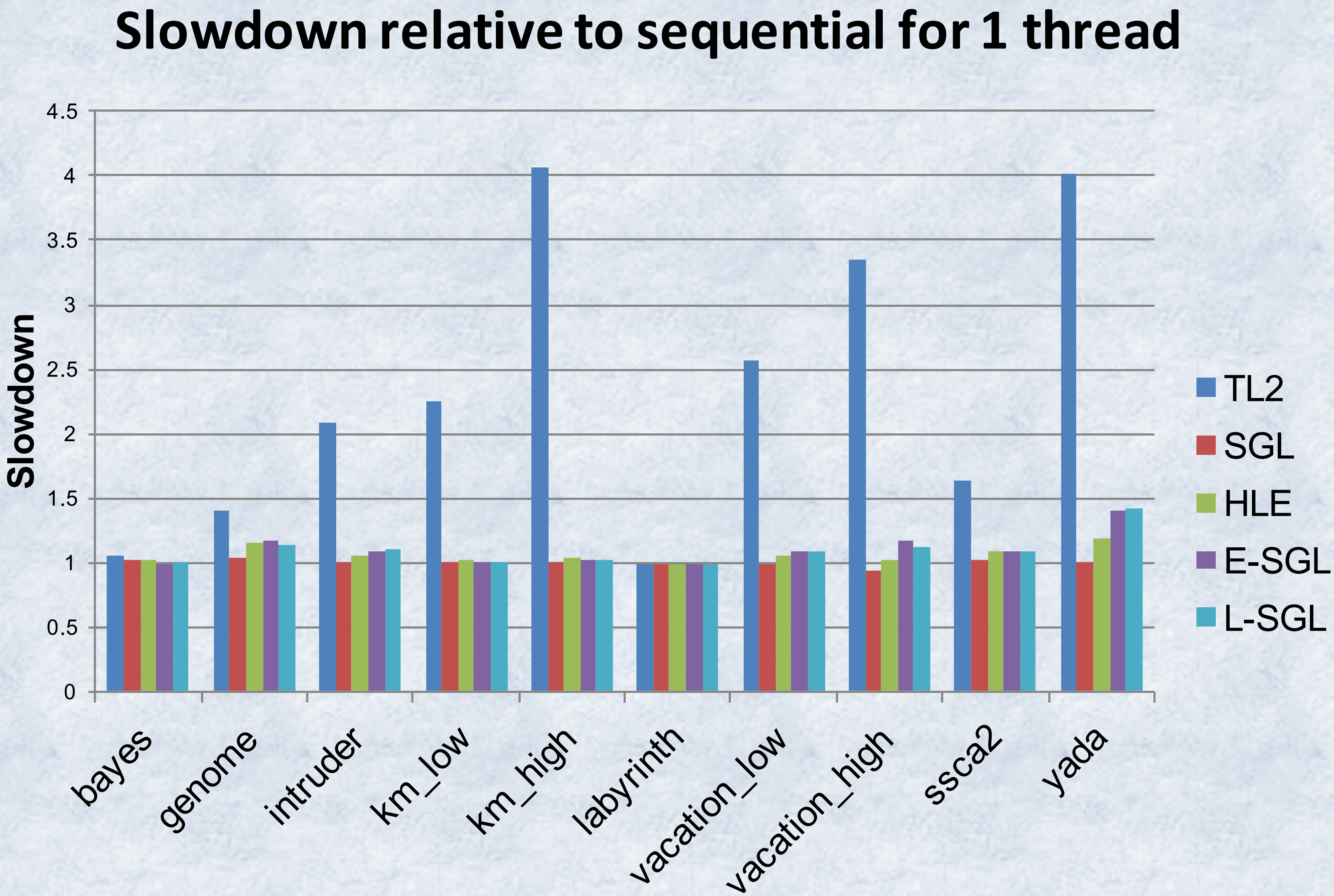
- Bloom Filter SGL


Intruder (medium txns)
Better
Speedup
Threads
TL2
SGL
HLE
E-SGL
L-SGL
Ssca2 (small txns)
Speedup
Threads
Labyrinth (large txns)
Speedup
Threads

Improved Lock Acquisition Rate



No single thread overhead



Overview

- Best-effort Hardware Transactional Memory

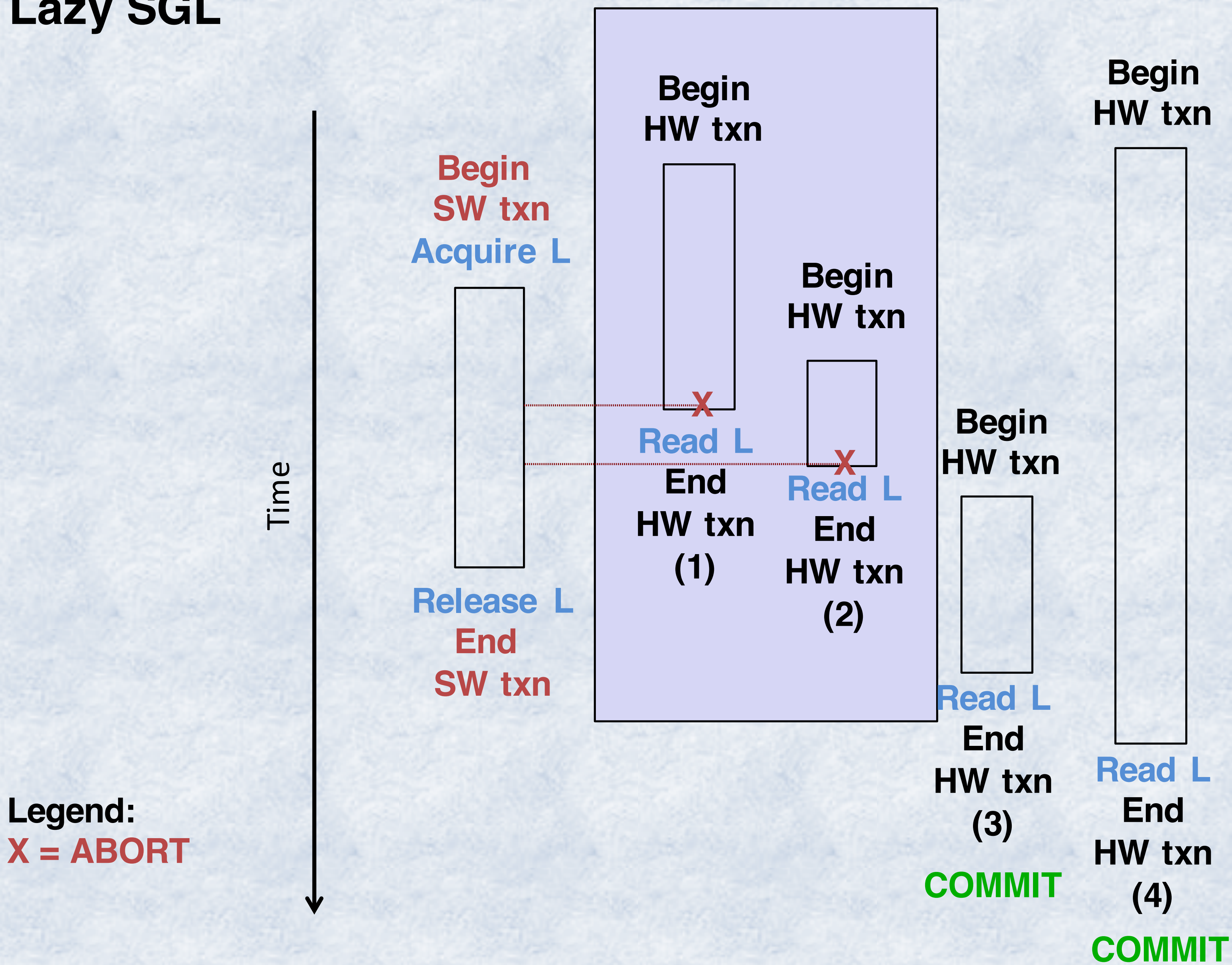
- Lazy SGL → Description
  → Correctness
  → Results

- Bloom Filter SGL

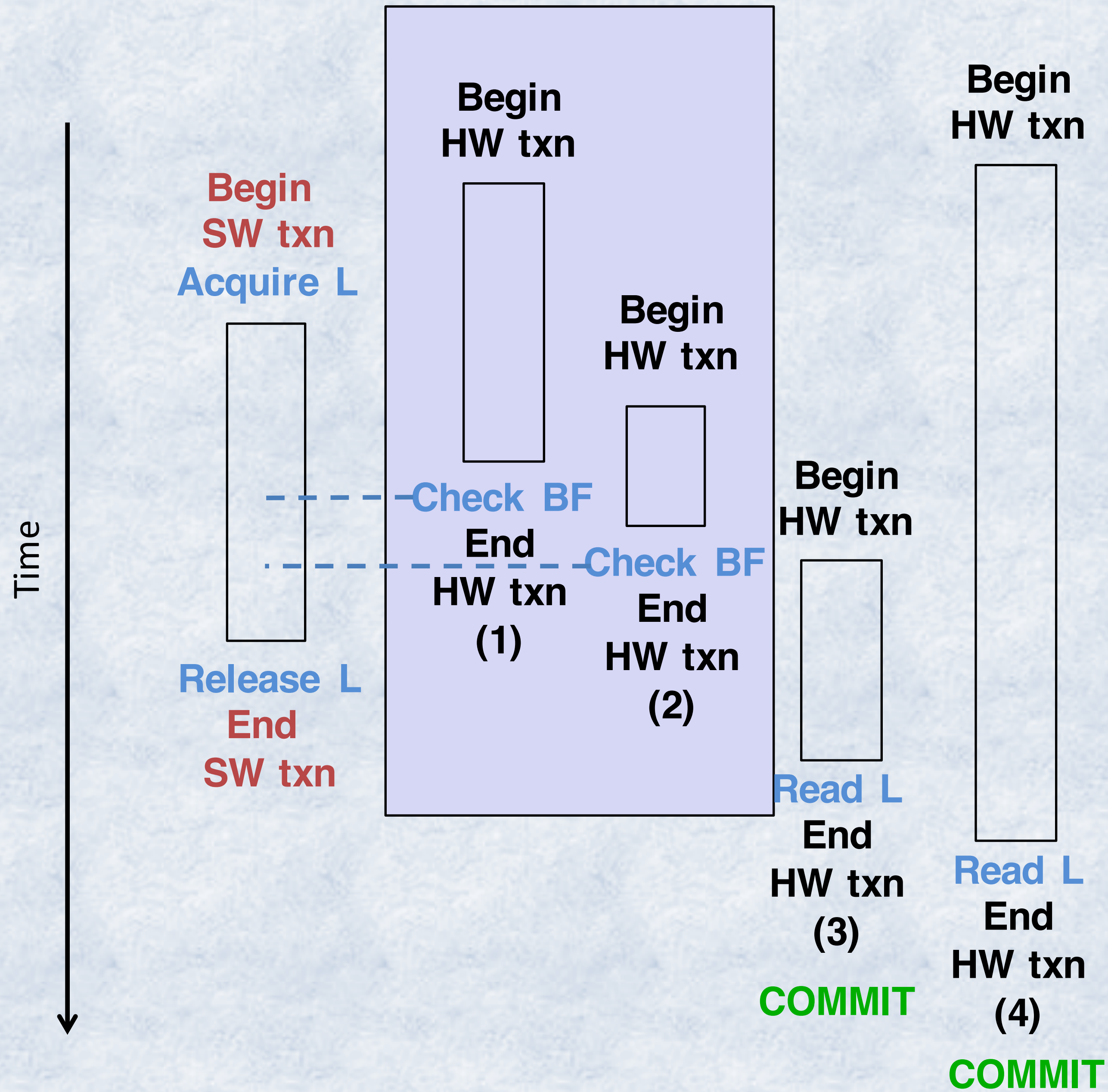
Bloom Filters

- Efficient probabilistic data structure to compute fast set intersection
- Can admit false positives
- No false negatives
- Used in TM for Conflict Detection

Lazy SGL


Time
Begin
SW txn
Acquire L
Release L
End
SW txn
Begin
HW txn
X
Read L
End
HW txn
(1)
Begin
HW txn
X
Read L
End
HW txn
(2)
Begin
HW txn
Read L
End
HW txn
(3)
COMMIT
Begin
HW txn
Read L
End
HW txn
(4)
COMMIT
Legend:
X = ABORT

BF SGL



Legend:
X = ABORT

Case 1:
HW begins
SW begins
HW ends
SW ends

Thread 1 (SW)	Thread 2 (HW)
	TXN_BEGIN
Acquire Lock	
…	
X = a	…
	X = b
…	Check BF …
	TXN_END
Release Lock	

X value:

a

Correct: a
Actual: a

Time

If BFs intersect: ABORT
Else: COMMIT

Case 2:
SW begins
HW begins
HW ends
SW ends

Thread 1 (SW)	**Thread 2 (HW)**
Acquire Lock	
…	TXN_BEGIN
X = a	…
	X = b
…	…
	TXN_END
Release Lock	

Time

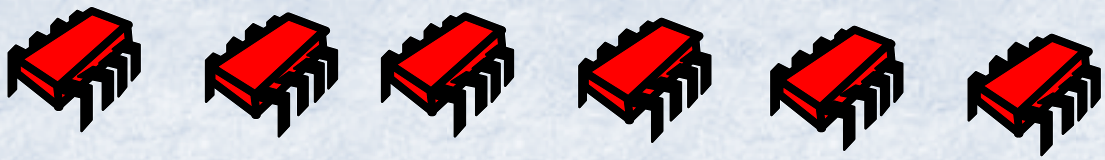
Check BF

X value:

Correct: a
Actual: b

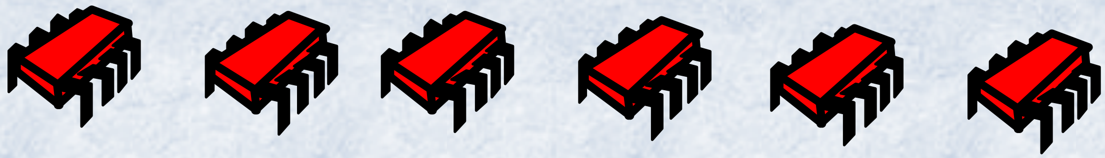
If BFs intersect: ABORT
Else: COMMIT

Conclusions



- HTMs are becoming more available
- Best-effort – need software fallback
- Eager SGL
    - simple and fast fallback,
    - often preferred to more efficient solutions

Conclusions



- Lazy SGL
  - as simple as Eager SGL
  - more efficient
- Bloom Filter SGL
  - more accurate conflict detection
  - Slower
- Can be implemented directly in hardware


time for questions