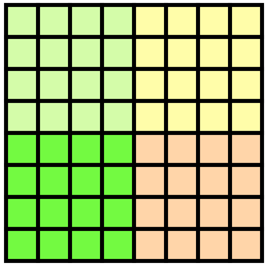
Energy Efficient GPU Transactional Memory via Space-Time Optimizations

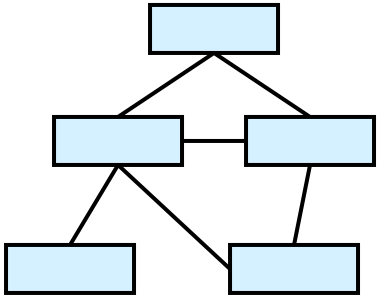
Wilson W. L. Fung
Tor M. Aamodt

a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

Why TM for GPU?

- Simple Irregular Parallelism on GPU

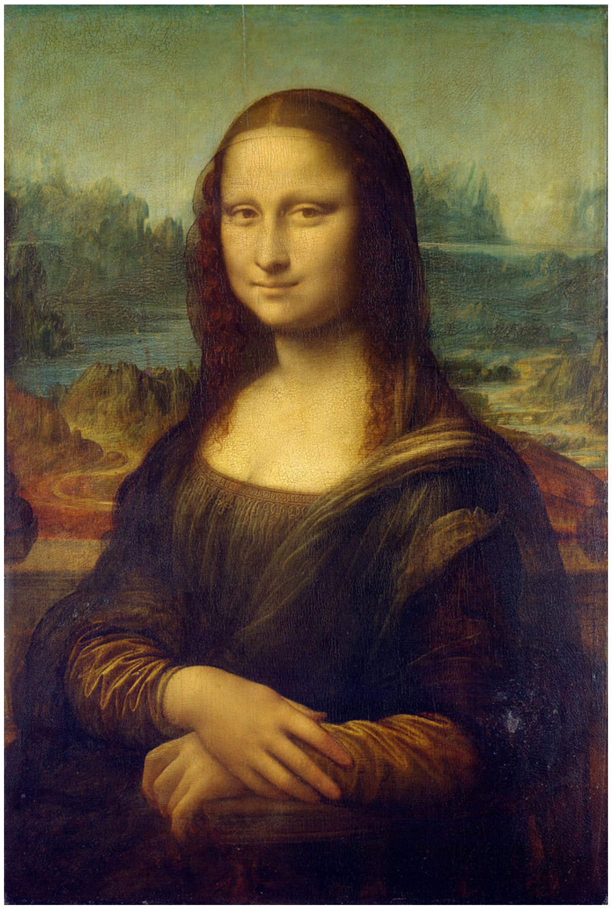




	Regular	Irregular
nBody 5M Bodies	1640s	5.2s

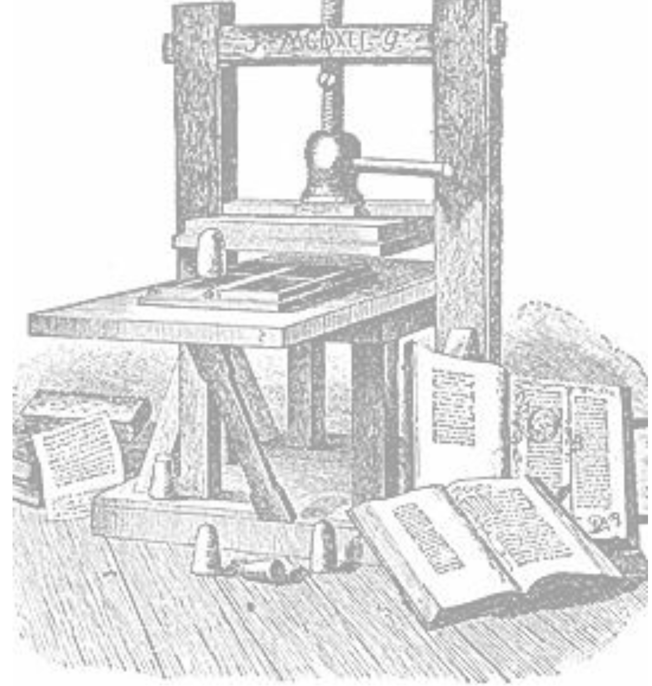
Other Applications?

Why TM for GPU?



TM on GPU

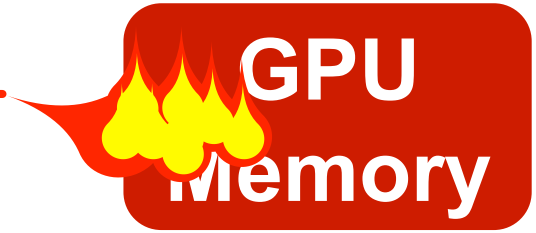
- Predictable Dev Time
- No Deadlock!
- Maintainable Code



TM for GPU: Energy Overhead

- **TM = Speculative Execution =**

- **<u>Kilo TM</u>**: First Hardware TM for GPU
  - Simple Design for Scalability
  - 1000s of Concurrent Transactions
  - **Scalar Transaction Management**
  - **Value-Based Conflict Detection**



Space

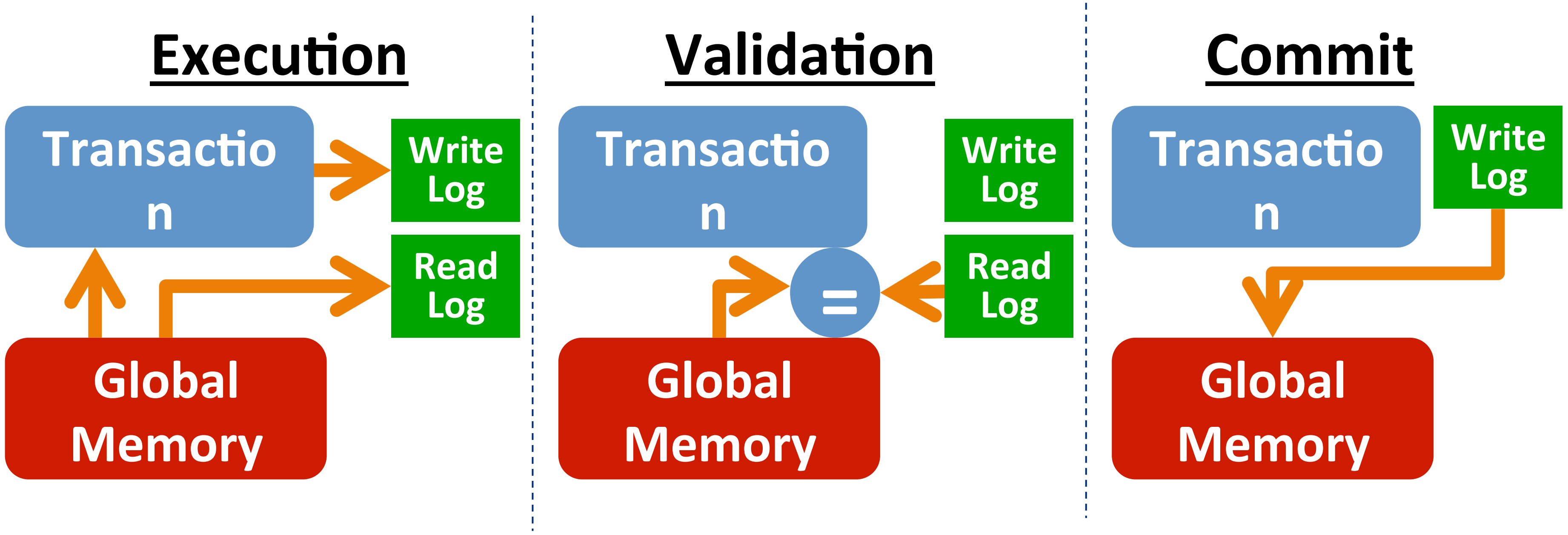
Time

65%
Speedup

2X→1.3X
Energy Usage

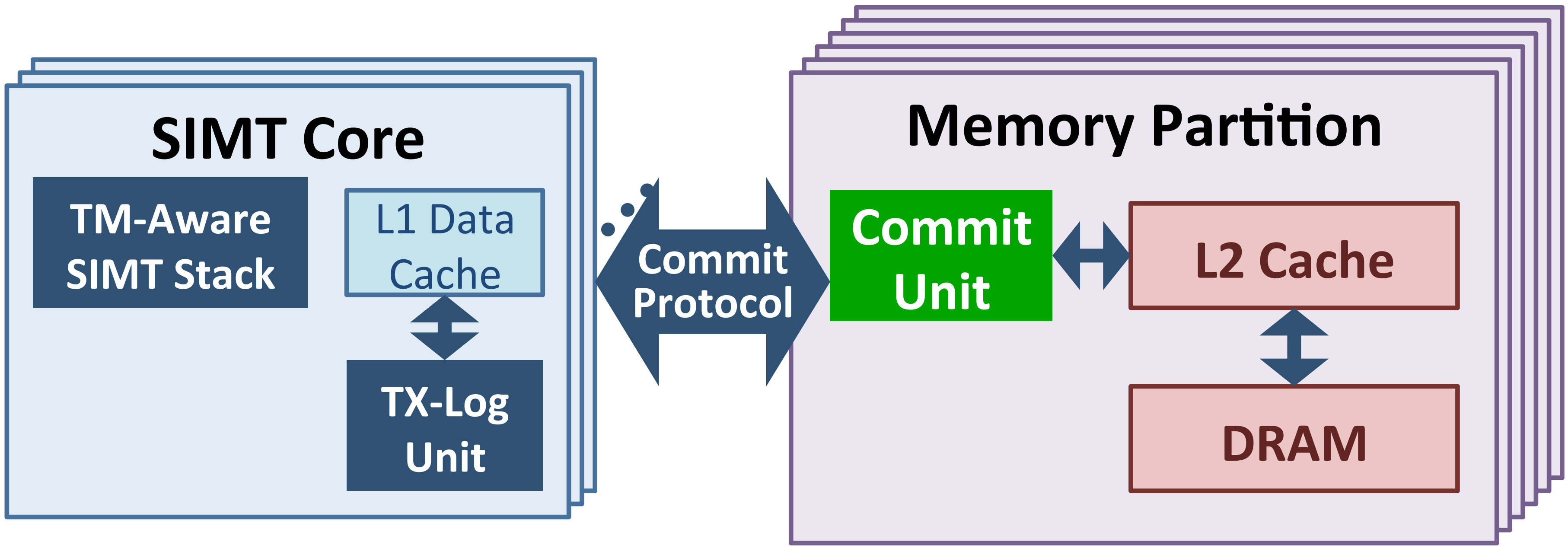
Background: Kilo TM

- 1000s of Concurrent Transactions
  - Value-based conflict detection: ~~Global Metadata~~



- Special HW to boost validation and commit parallelism

Kilo TM Implementation



Efficiency Concerns

128X Speedup over CG-Locks

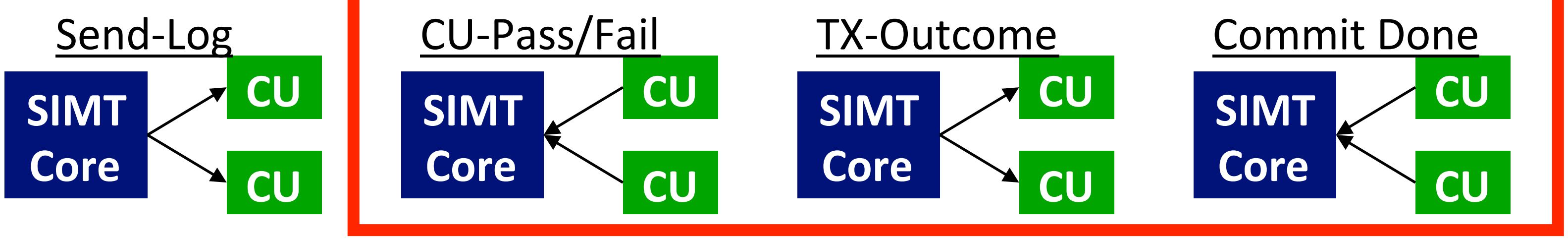
40% FG-Locks Performance

2X Energy Usage

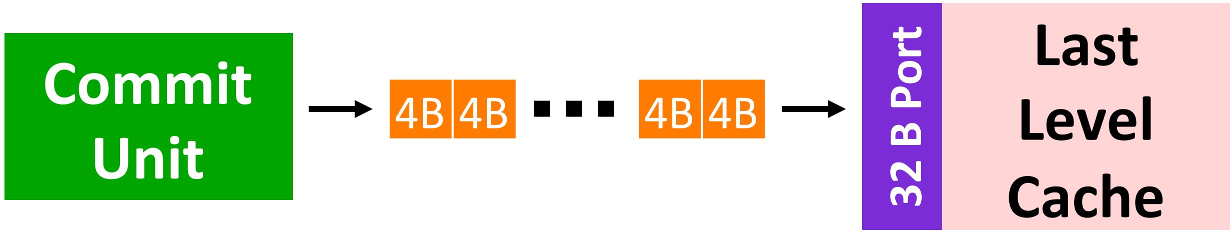
- Scalar Transaction Management
  - Scalar Transaction fits SIMT Model
  - Simple Design
  - Poor Use of SIMD Memory Subsystem
- Rereading every memory location
  - Memory access takes energy

Inefficiency from Scalar Transaction Management

- Kilo TM ignores GPU thread hierarchy
  - Excessive Control Message Traffic



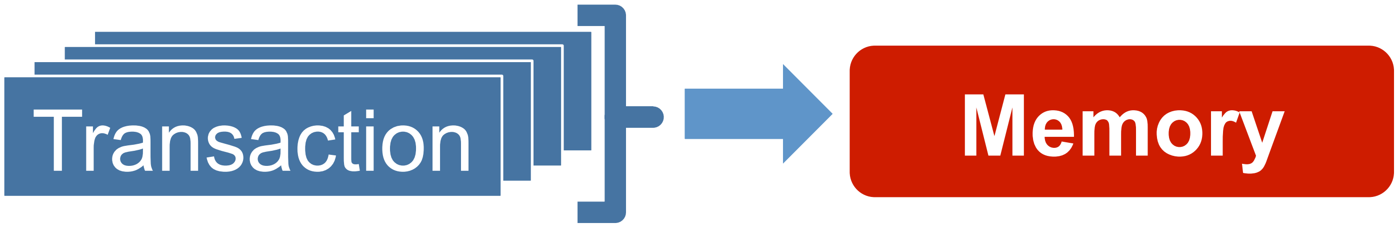
  - Scalar Validation and Commit
    → Poor L2 Bandwidth Utilization



- Simplify HW Design, but **<u>Cost Energy</u>**

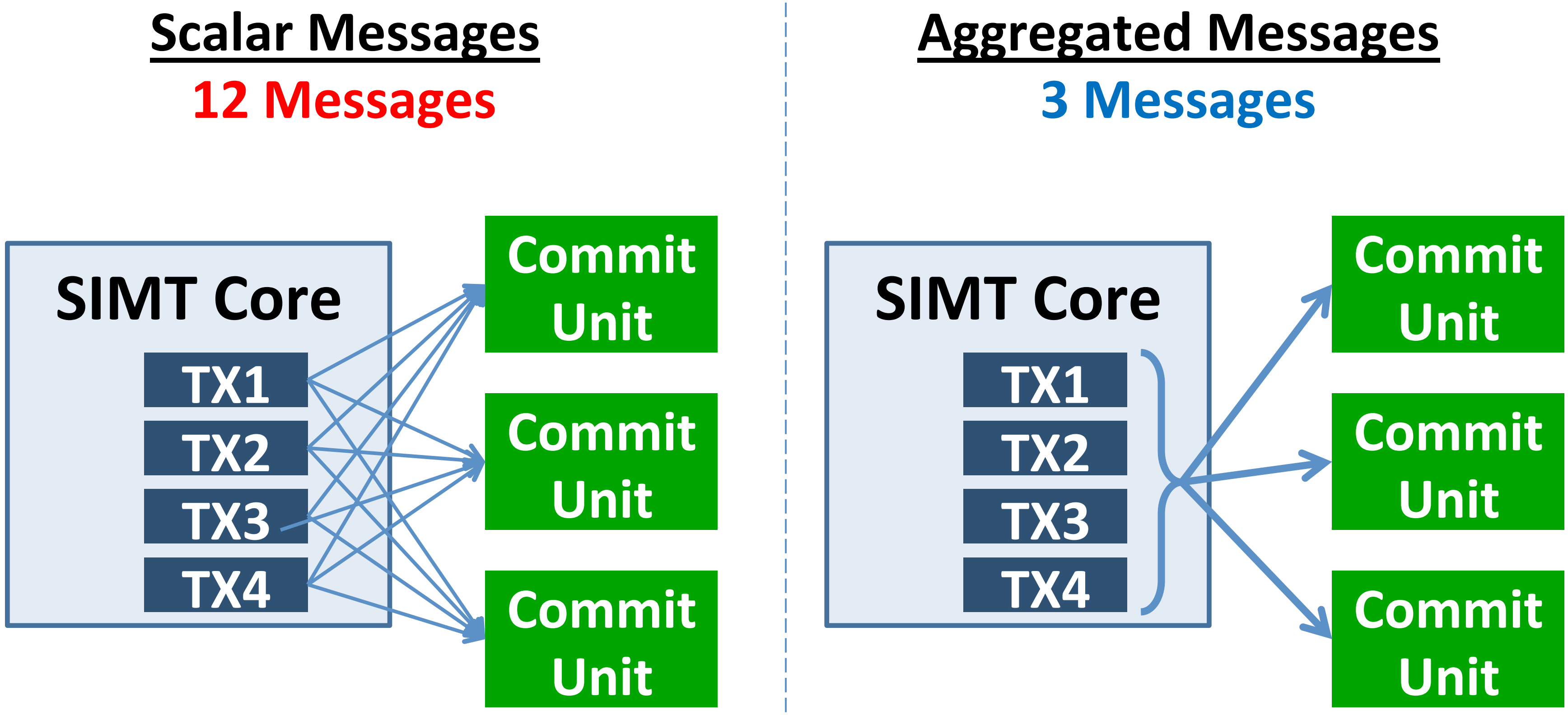
Warp Level Transaction Management

- Key Idea:
  - Manage transactions within a warp as a whole



- Enables optimizations that exploit spatial locality:
  - Aggregate Control Messages
  - Validation and Commit Coalescing
- Challenge: <u>Intra-Warp Conflicts</u>

Warp Level Transaction Management: Aggregate Control Messages

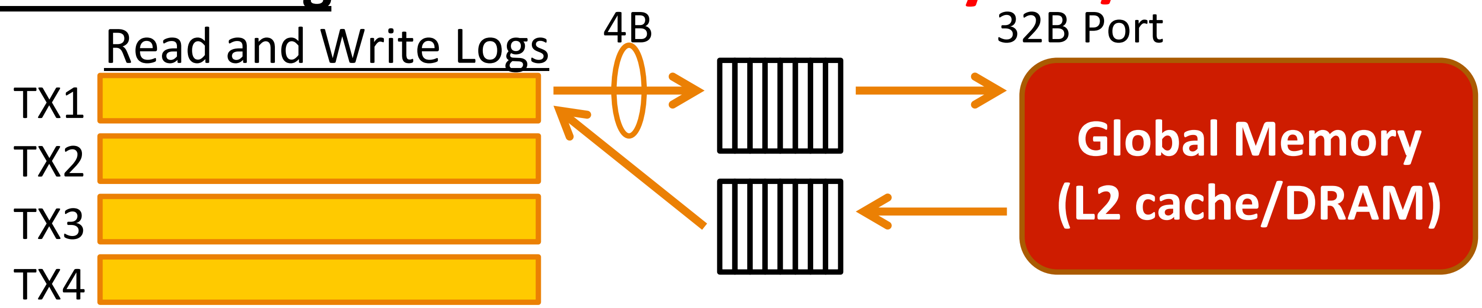


Contributes up to 40% of Interconnection Traffic

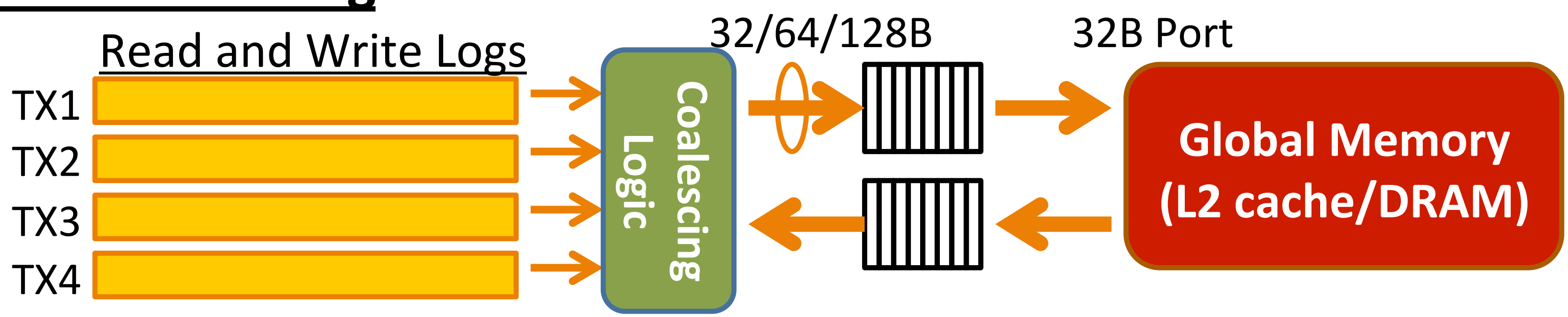
Warp Level Transaction Management: Validation and Commit Coalescing

Without Coalescing

Max Utility = 4/32 = 12.5%



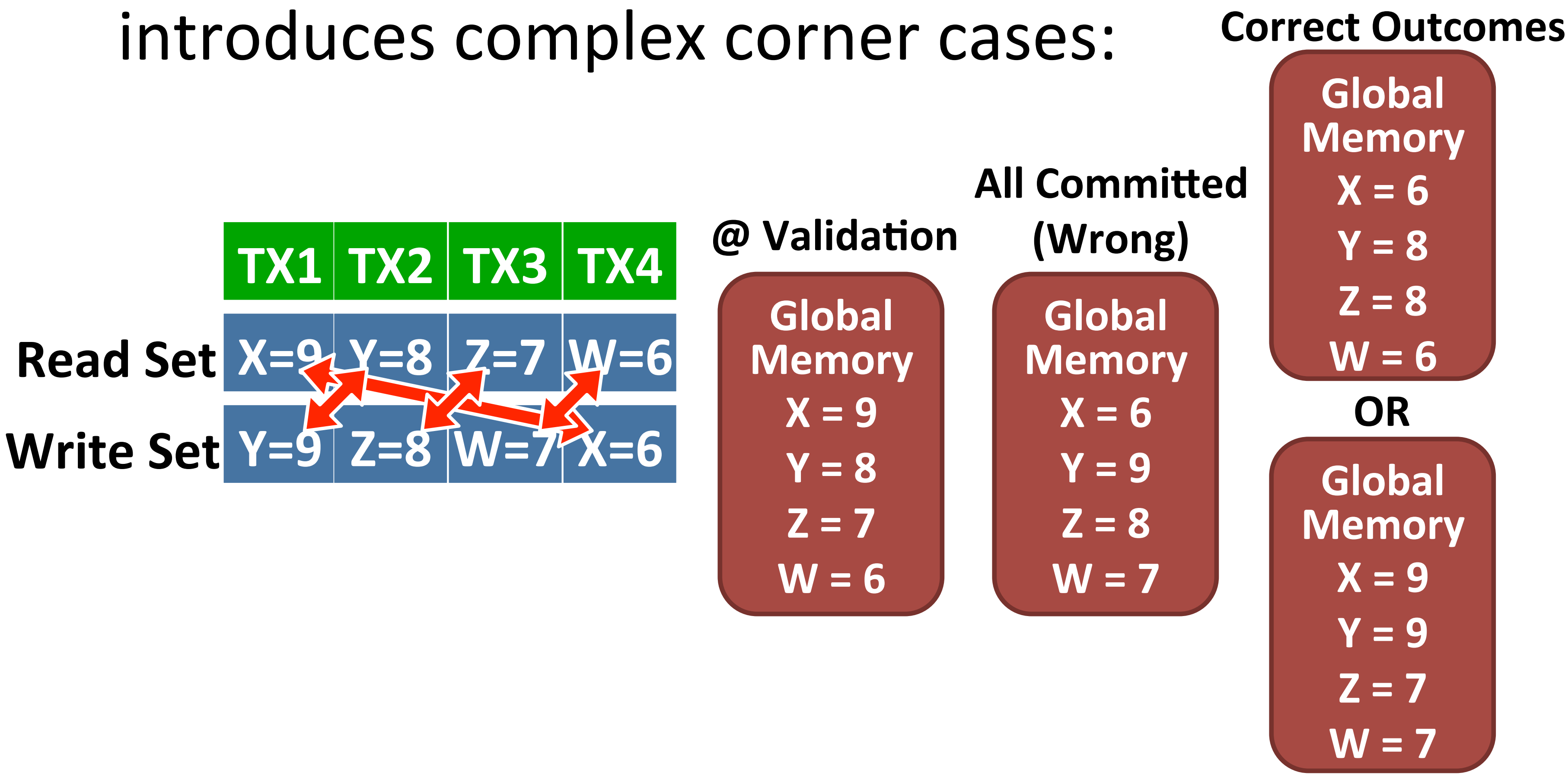
With Coalescing



Reduce 40% of Requests to L2 Cache

Intra-Warp Conflict

- Potential existence of <u>intra-warp conflict</u> introduces complex corner cases:

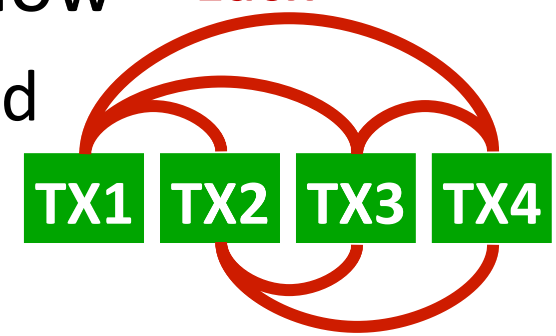


Intra-Warp Conflict Resolution

Execution | Intra-Warp Conflict Resolution | Validation | Commit

- Kilo TM stores read-set and write-set in logs
  - Compact, fits in caches
  - Inefficient for search
- Naive, pair-wise resolution too slow
  - T threads/warp, R+W words/thread
  - $O(T^2 \times (R+W)^2)$, $T \geq 32$

$O((R+W)^2)$ Comparisons Each

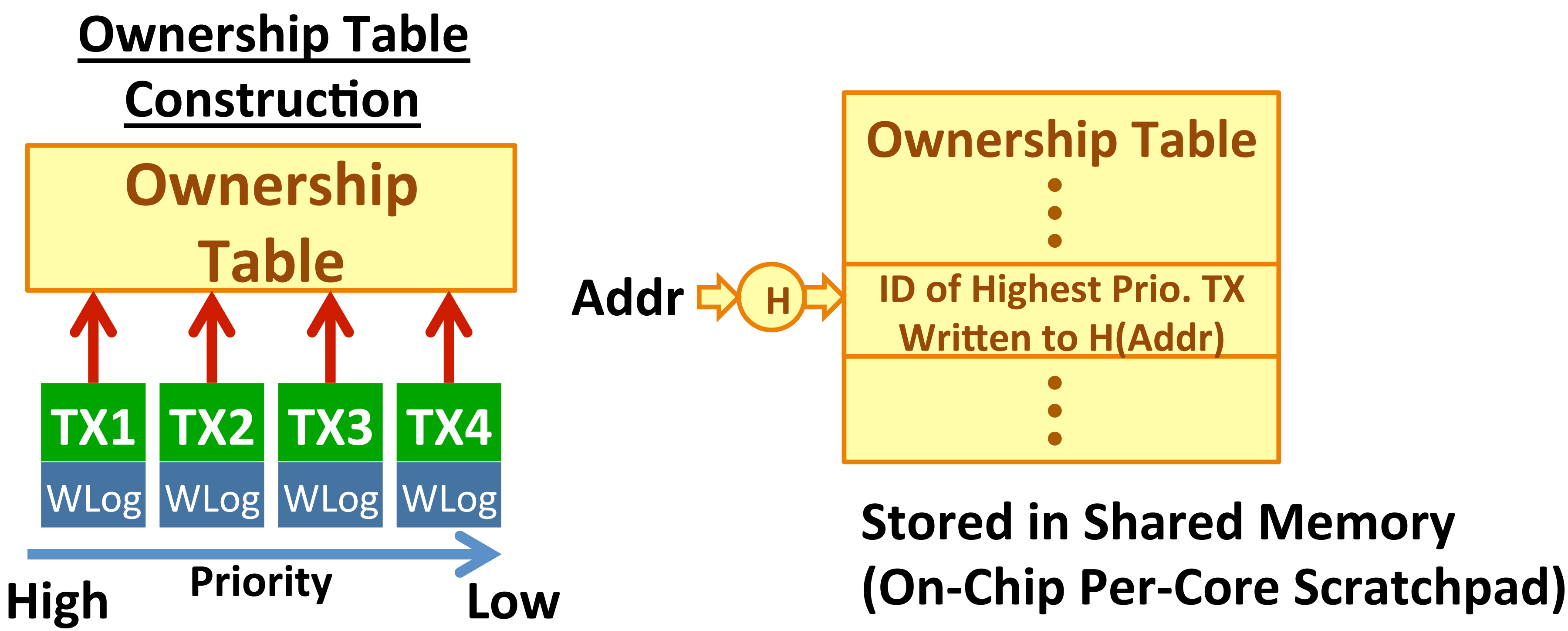


Intra-Warp Conflict Resolution: 2-Phase Parallel Conflict Resolution

- Insight: Fixed priority for conflict resolution enables parallel resolution
- O(R+W)
- Two Phases
  - Ownership Table Construction
  - Parallel Match

Intra-Warp Conflict Resolution: 2-Phase Parallel Conflict Resolution

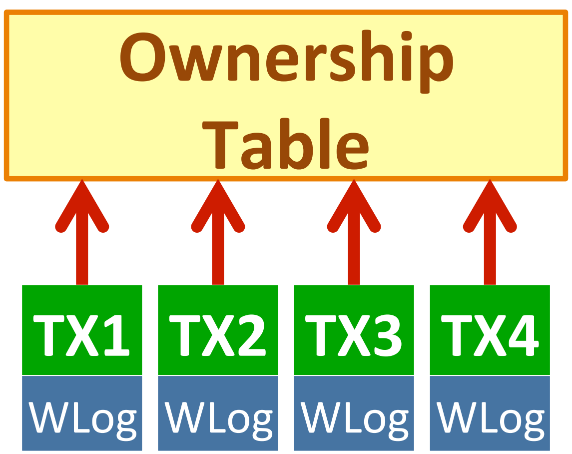
- Insight: Fixed priority for conflict resolution enables parallel resolution



Intra-Warp Conflict Resolution: 2-Phase Parallel Conflict Resolution

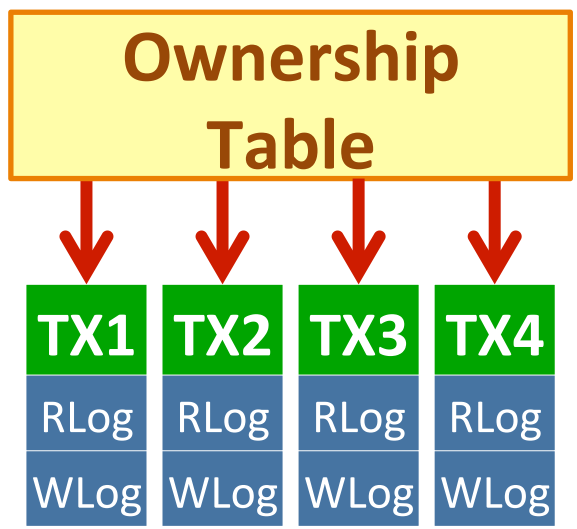
- Insight: Fixed priority for conflict resolution enables parallel resolution

Ownership Table Construction



O(W)

Parallel Match

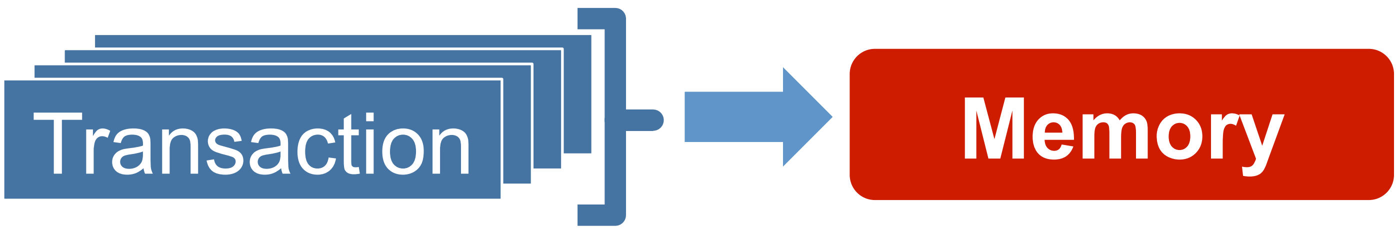


O(R+W)

Read-Log:
Owner ID < My ID → Abort
(E.g. Owner ID = 2 → Abort)

Write-Log:
OwnerID != My ID → Abort
(E.g. Owner ID = 3 → Pass)

Warp Level Transaction Management Made Practical



- Enables optimizations that exploit spatial locality:
  - Aggregate Control Messages
  - Validation and Commit Coalescing
- ~~Challenge: Intra-Warp Conflicts~~

Temporal Conflict Detection

- Motivation:
  Skip value-based conflict detection for conflict-free <u>read-only transactions</u>

Data-Dependent Control Flow

TX1

```
if (C == 0)
  B = B + 1;
```

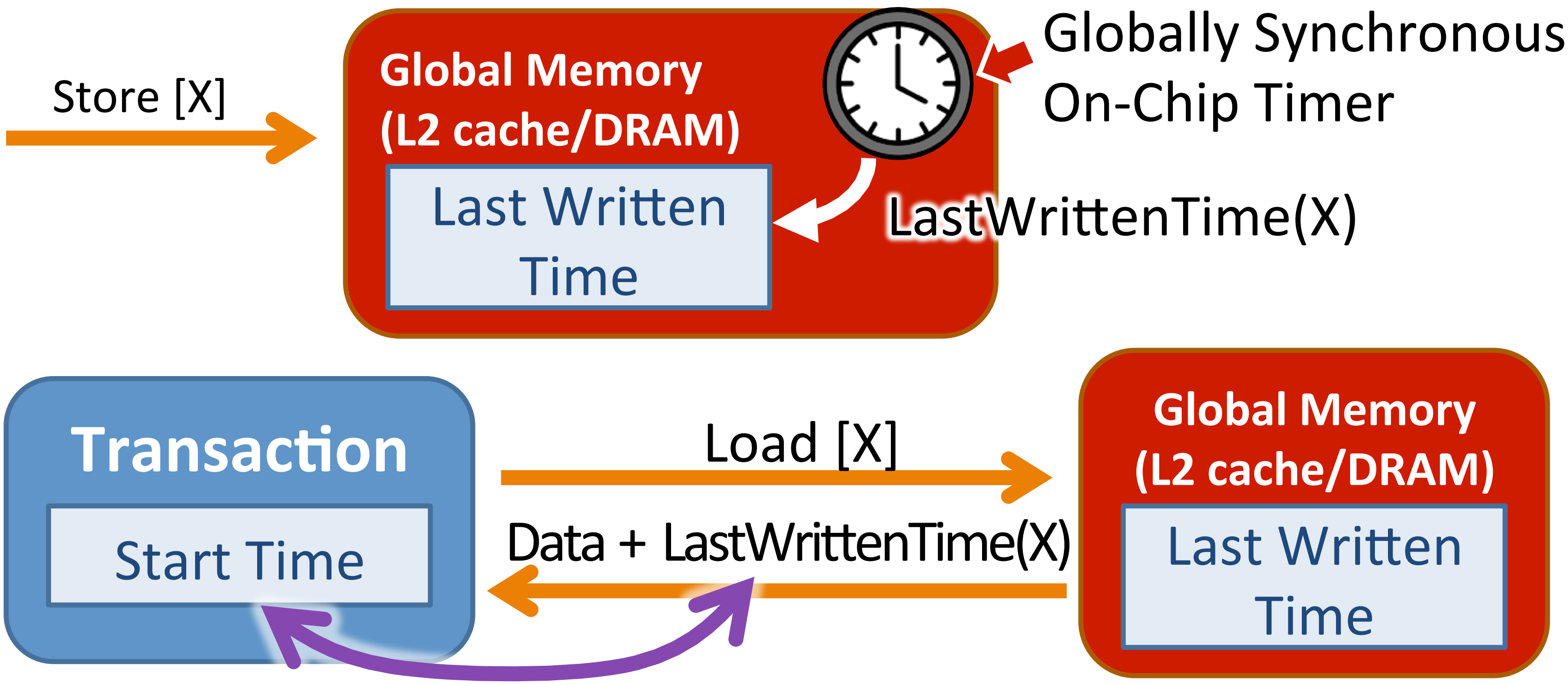
Consistent View of Memory

TX2

```
int K;
K = X + Y;
```

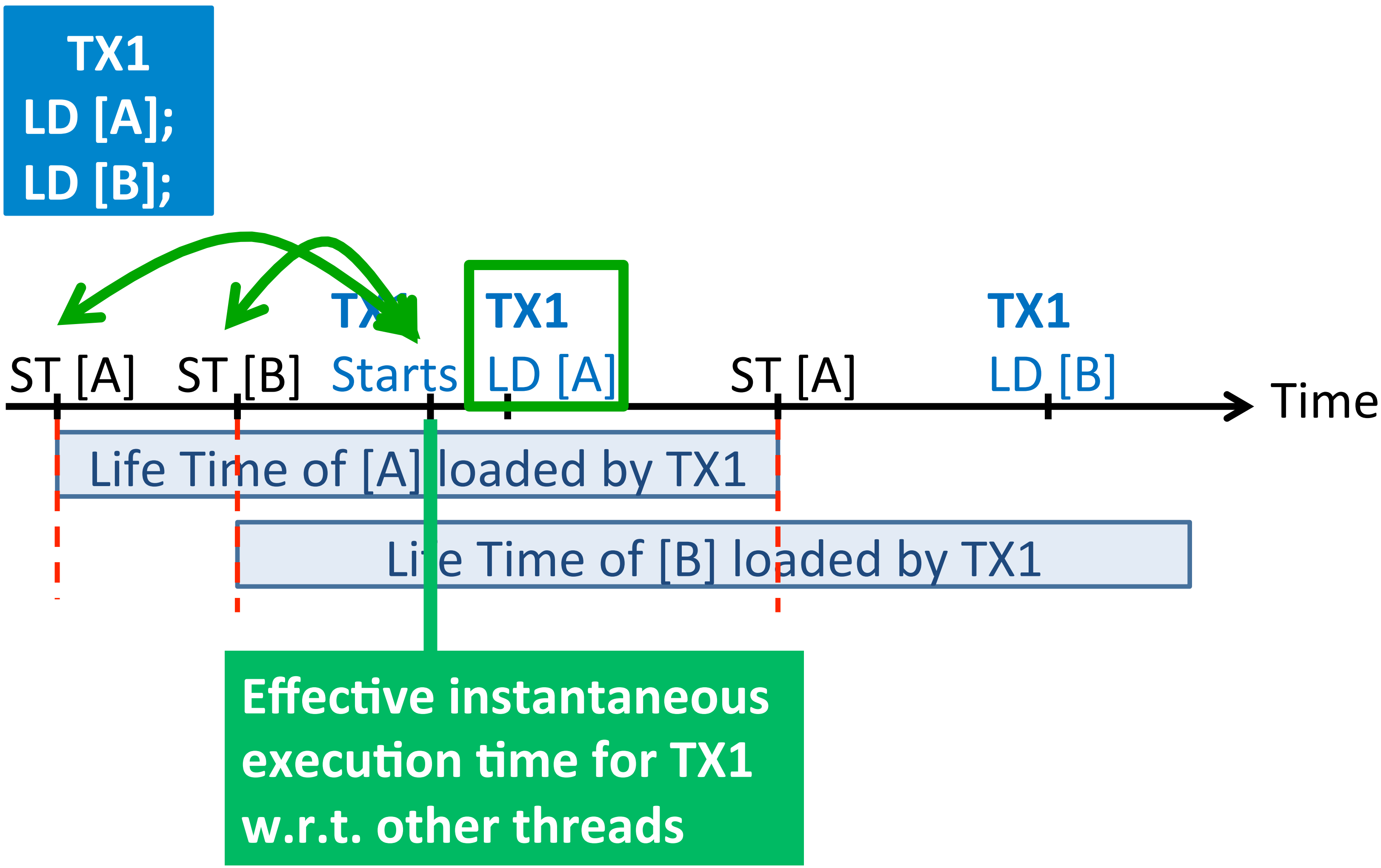
  – 40% and 85% of the transactions in two of our workloads.

Temporal Conflict Detection



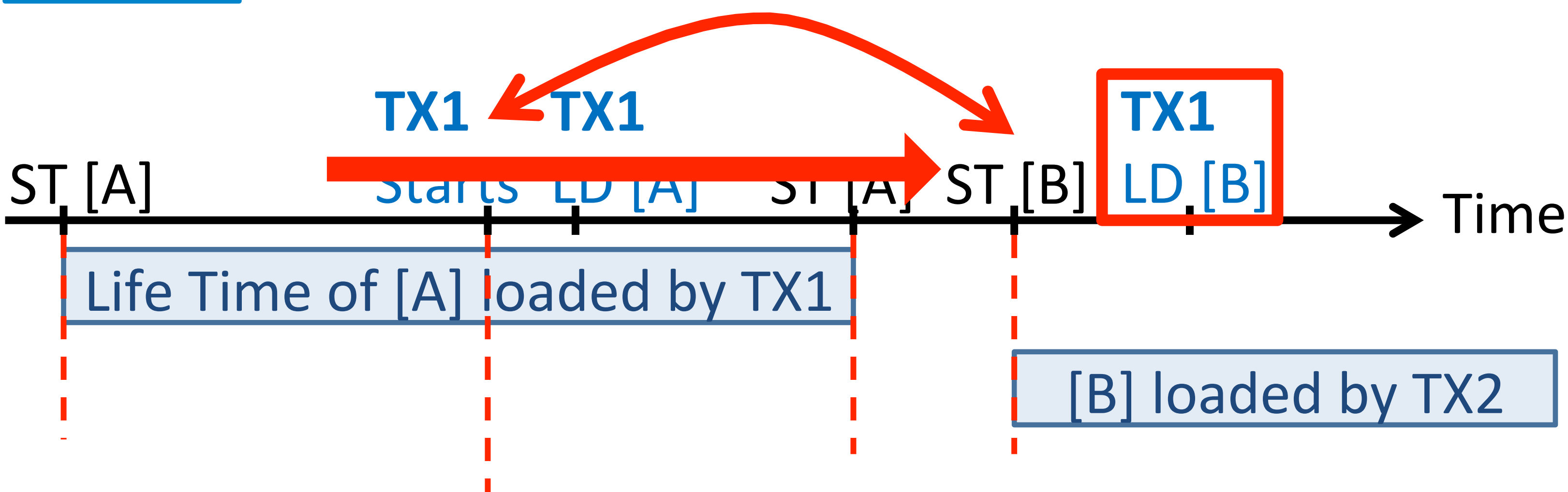
- If LastWrittenTime(X) < StartTime, **Pass**
- Otherwise, **Conflict Detected**

Temporal Conflict Detection



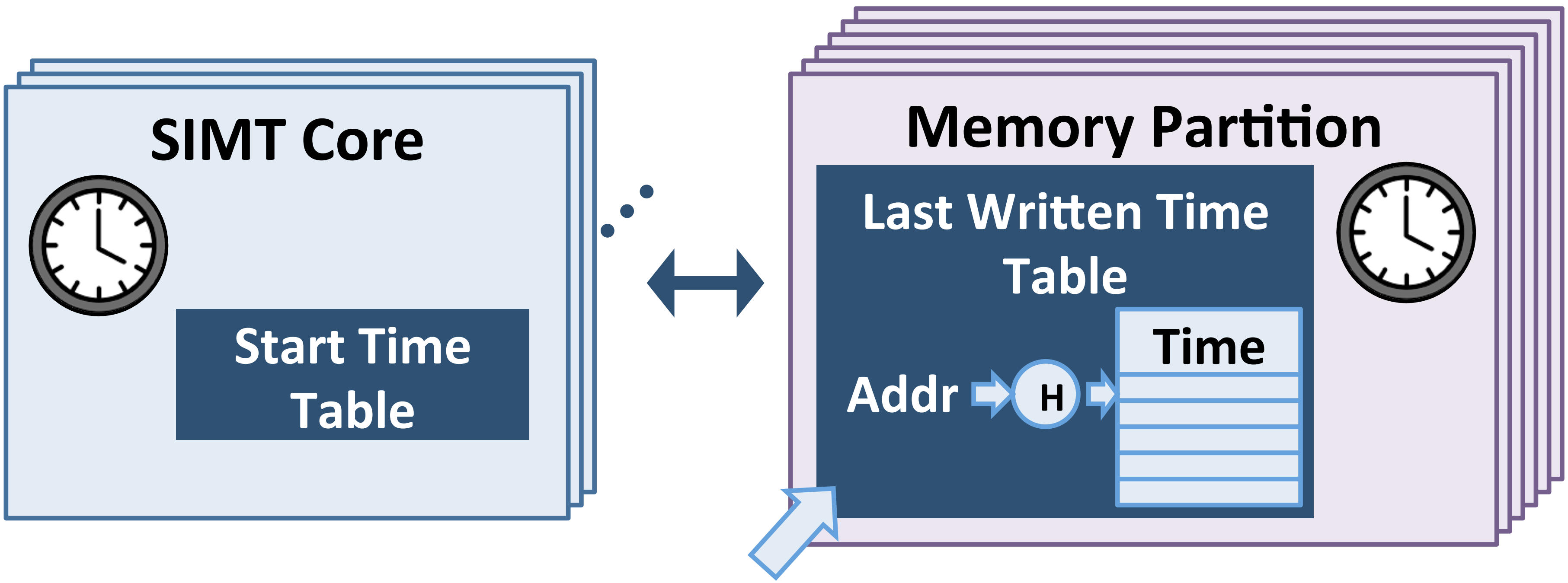
Temporal Conflict Detection

TX1
LD [A];
LD [B];



Value loaded by LD [A] and value loaded by LD [B] cannot coexists at any point of time – a detected conflict.

Temporal Conflict Detection Implementation



16kB **Recency Bloom Filter**

- Approximate but Conservative
- Aliasing two very old store is OK

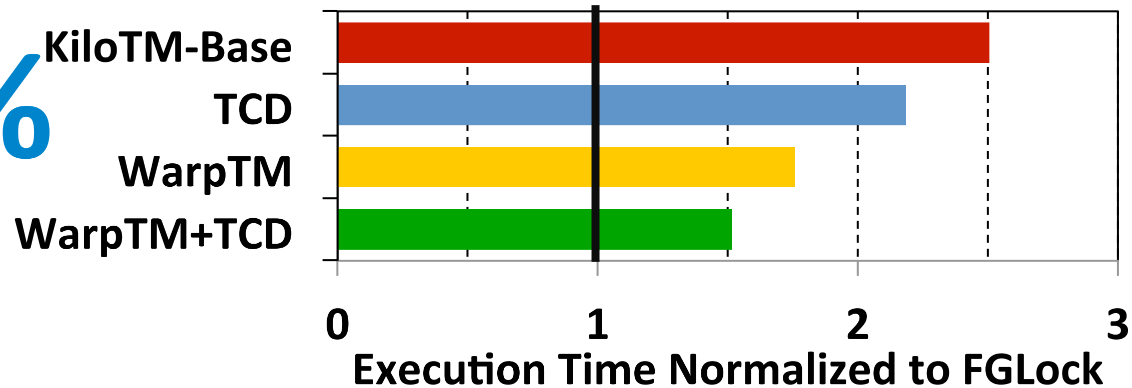
Evaluation

- GPGPU-Sim 3.2.1
  - Detailed: IPC Correlation of 0.90 vs. Fermi GPU
- Model Energy Overhead of Kilo TM
  - Extra Hardware
    - CACTI for access energy of major SRAM arrays
  - Extra Activity via GPUWattch
  - Increased Execution Time (More Leakage)
- GPU TM Applications

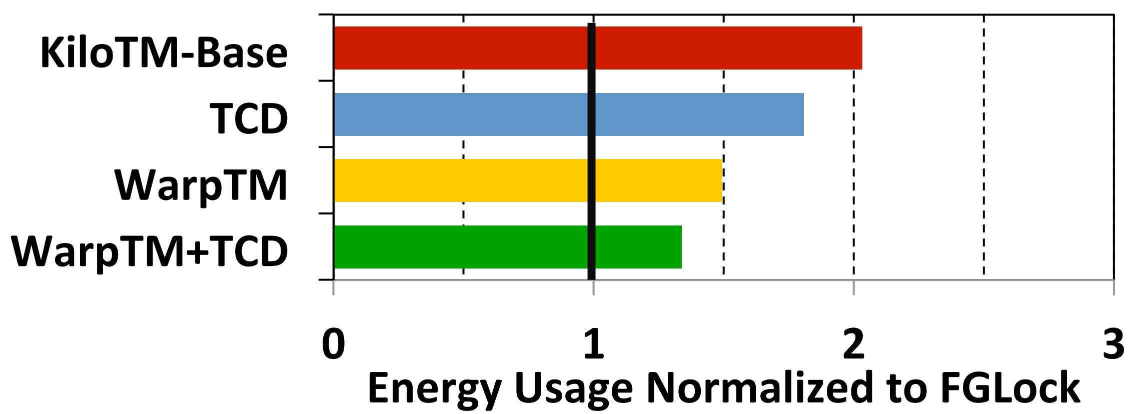
HT-[H/M/L] – Hash Table Construction	**ATM – Bank Transactions**
BH-[H/L] – Barnes Huts (N-Body)	**CL/CLto – Cloth Simulation**
CC – Maxflow/Mincut Graph	**AP – Data Mining**

Results

40%→66%
FG-Lock Performance



2X→1.3X
Energy Usage



Low Contention Workload:

Kilo TM w/ SW Optimizations on par with FG Lock

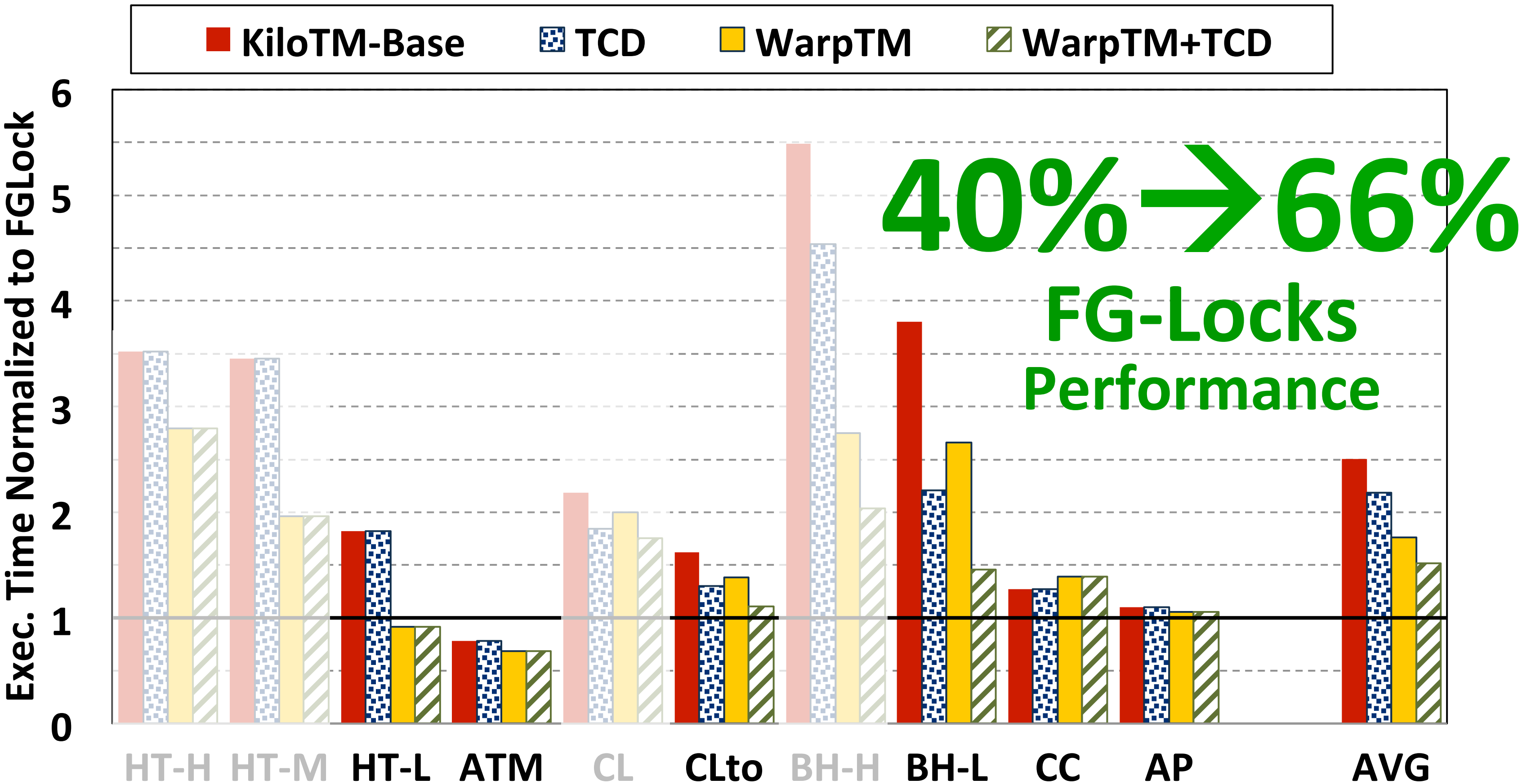
Summary

Questions?

- Two Enhancements for Kilo TM
  - Warp Level Transaction Management
    - Exploit Spatial Locality in Thread Hierarchy
  - Temporal Conflict Detection
    - Silent Commit of Read-Only Transaction
- Reduce Performance and Energy Overhead of Kilo TM
- Low Contention Workload:
  Kilo TM w/ Optimizations on par with FG Lock

BACKUP SLIDES

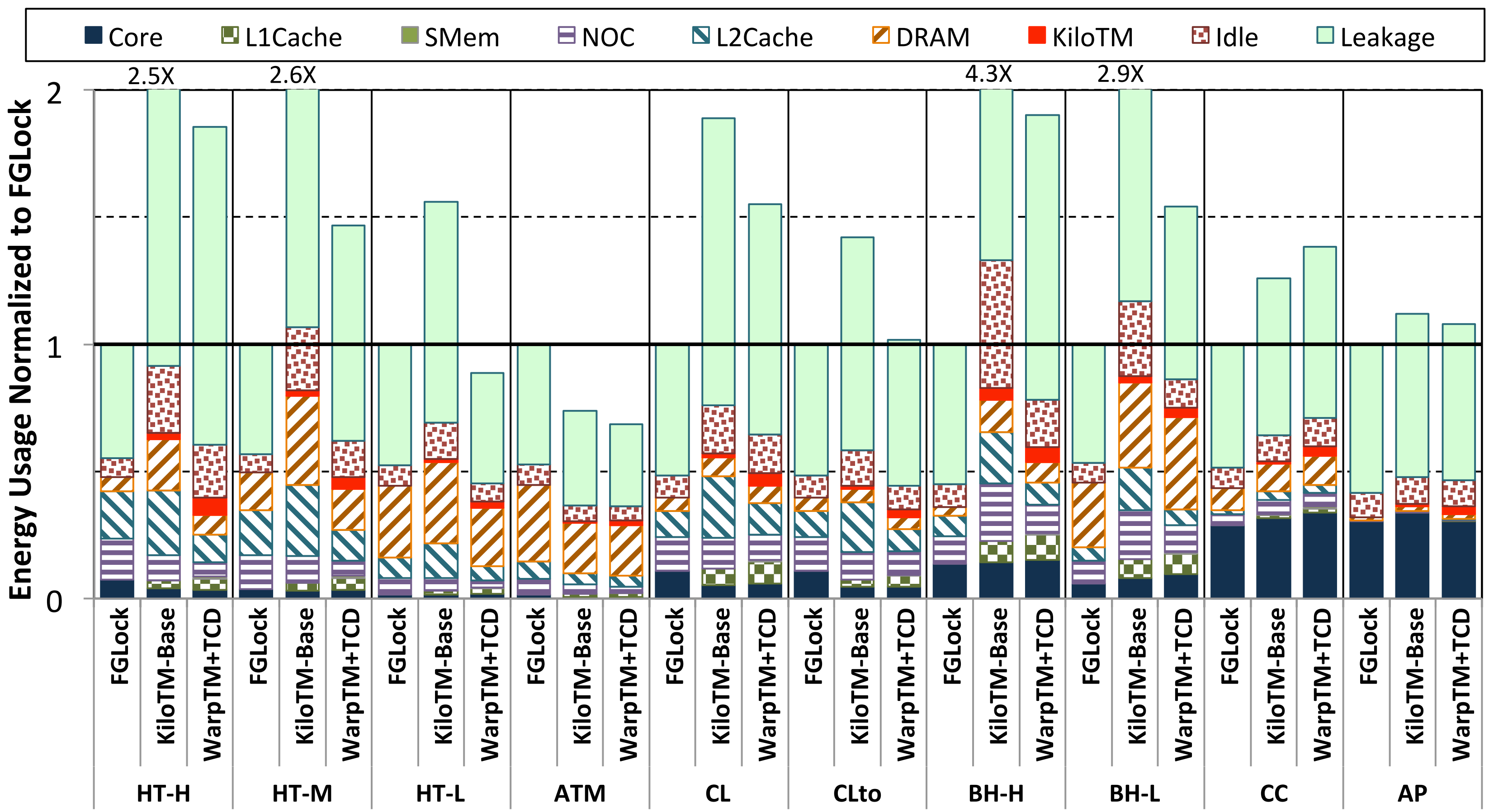
Normalized Performance



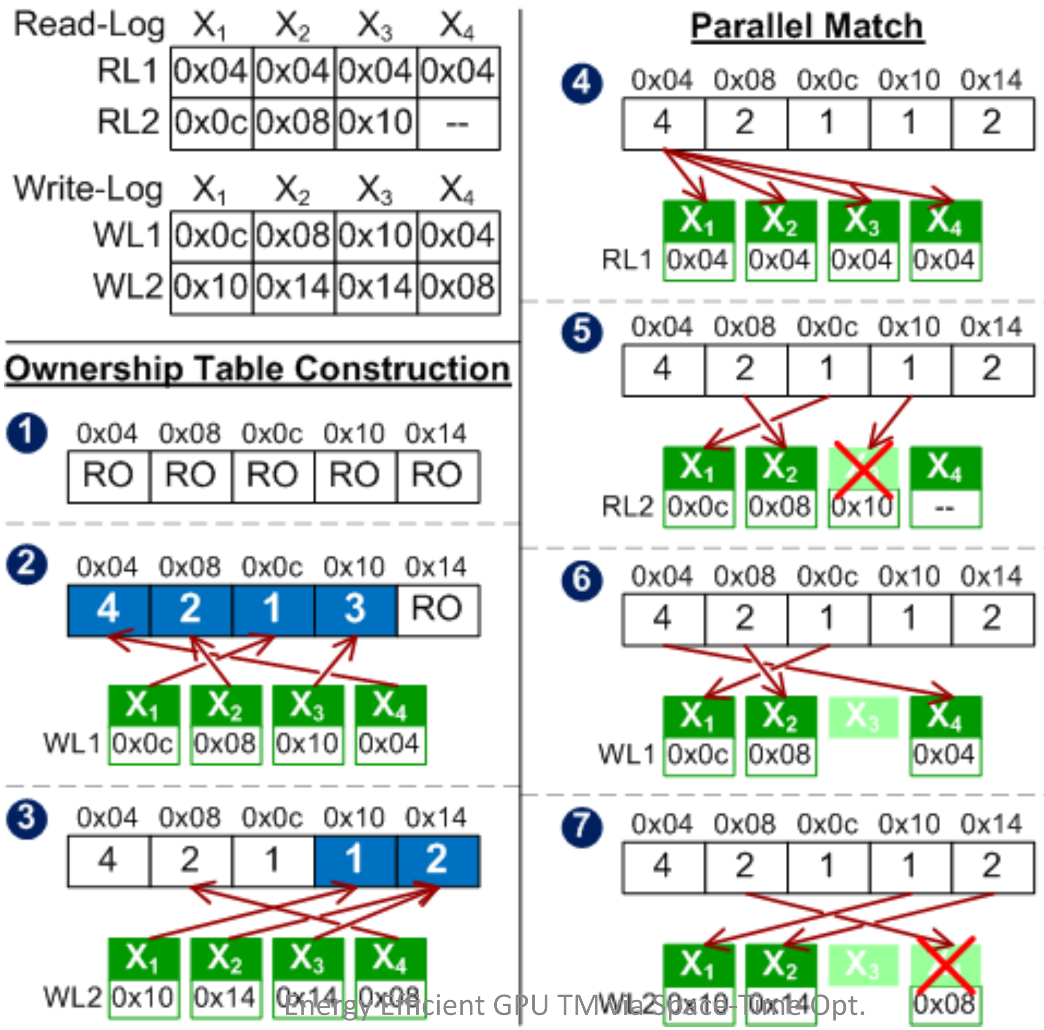
Low Contention Workload:

Kilo TM w/ SW Optimizations on par with FG Lock

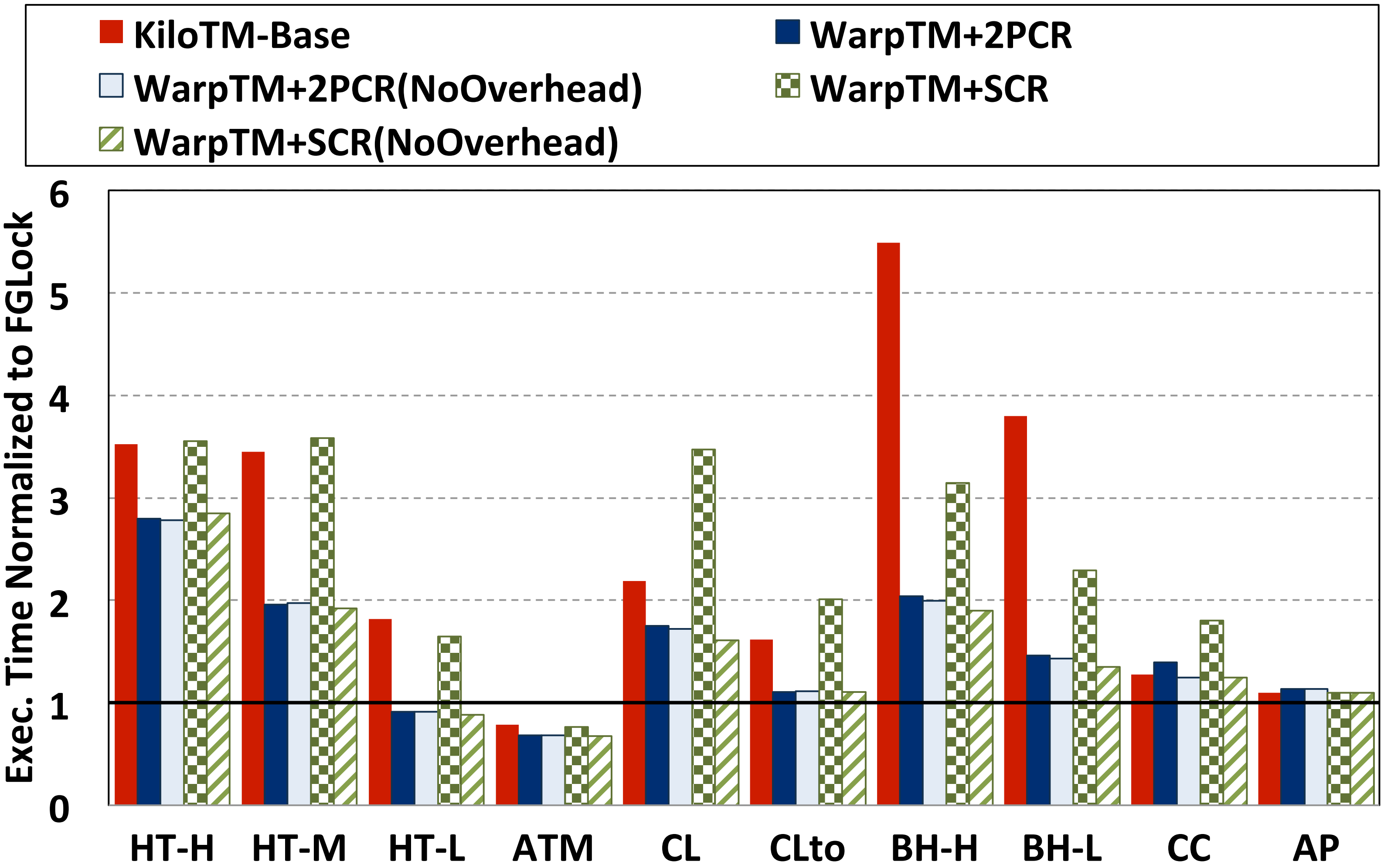
Normalized Energy Usage



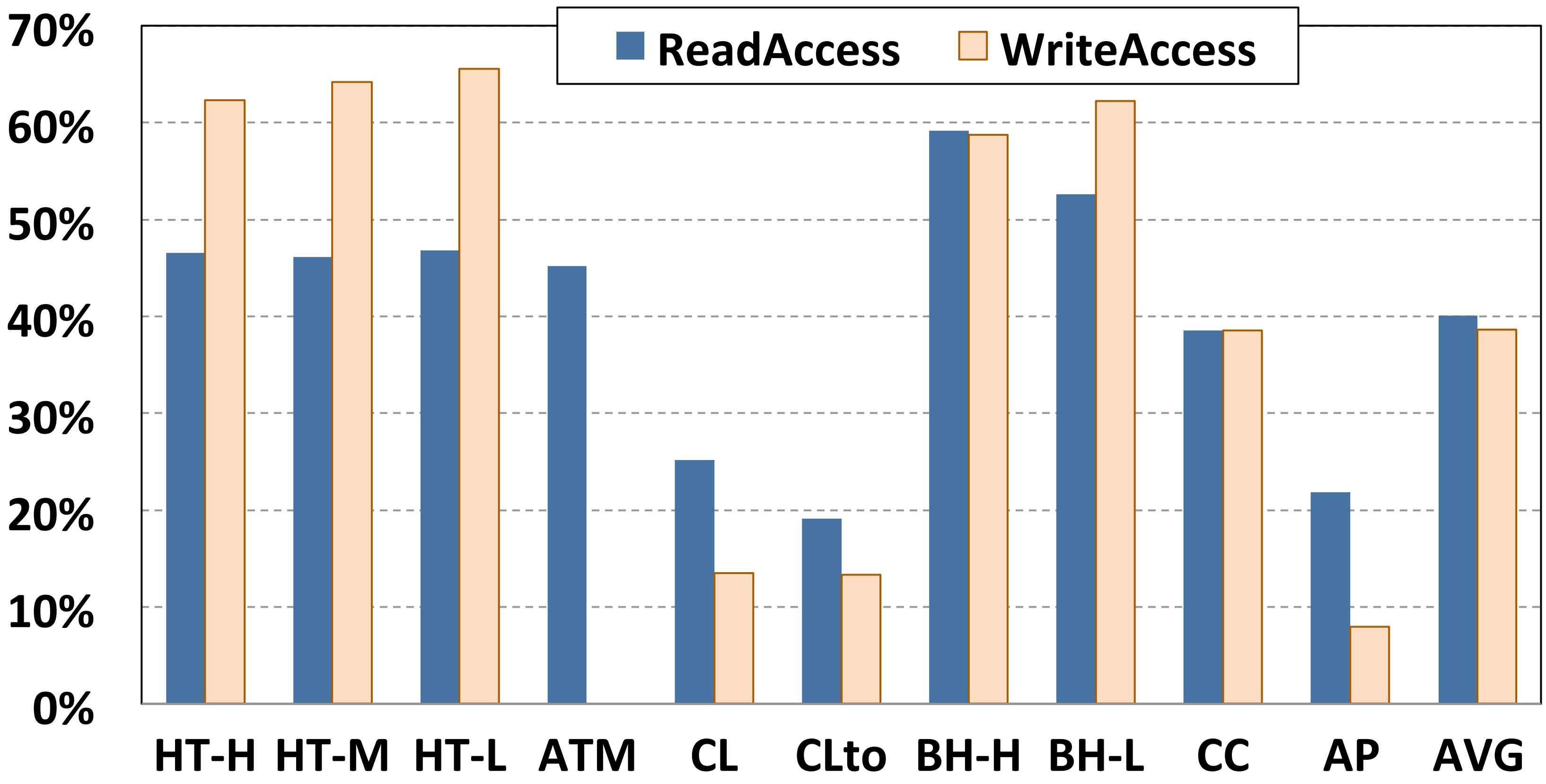
Intra-Warp Conflict Resolution: 2-Phase Parallel Conflict Resolution



2PCR vs. SCR

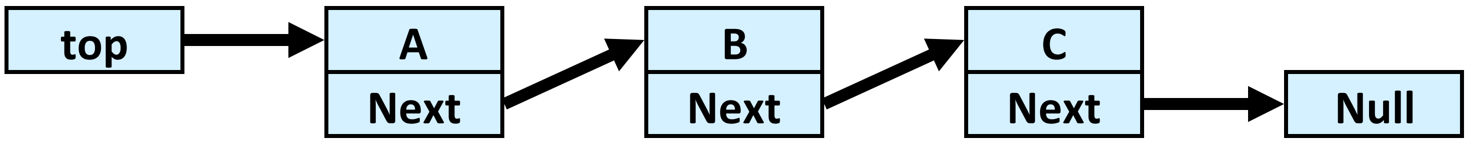

KiloTM-Base
WarpTM+2PCR
WarpTM+2PCR(NoOverhead)
WarpTM+SCR
WarpTM+SCR(NoOverhead)
Exec. Time Normalized to FGLock
0
1
2
3
4
5
6
HT-H
HT-M
HT-L
ATM
CL
CLto
BH-H
BH-L
CC
AP

Spatial Locality among Transactions



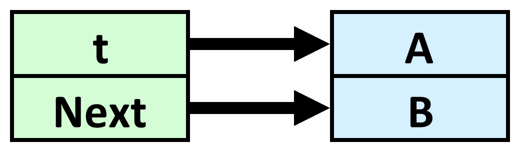
ABA Problem?

- Classic Example: Linked List Based Stack

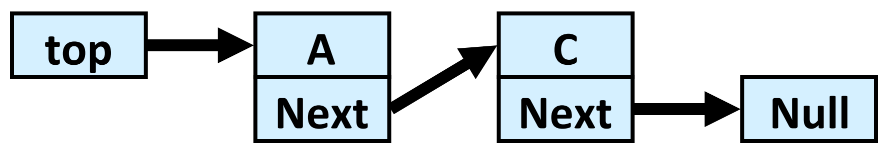


- Thread 0 – pop():

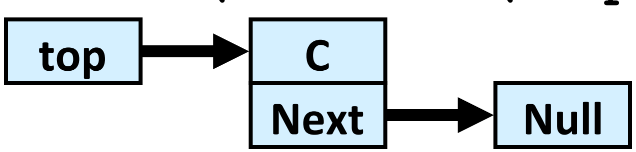
```
while (true) {
   t = top;
   Next = t->Next;
```

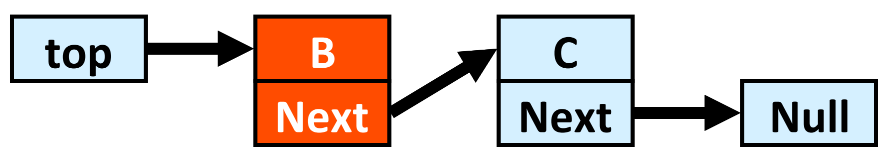



```
            // thread 2: pop A, pop B, push A
```




```
   if (atomicCAS(&top, t, next) == t) break;  // succeeds!
```

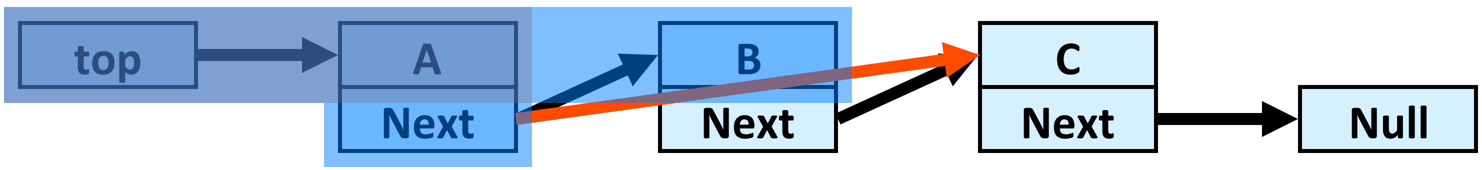





```
}
```

ABA Problem?

- atomicCAS protects <u>only a single word</u>
  - **<u>Only part of the data structure</u>**

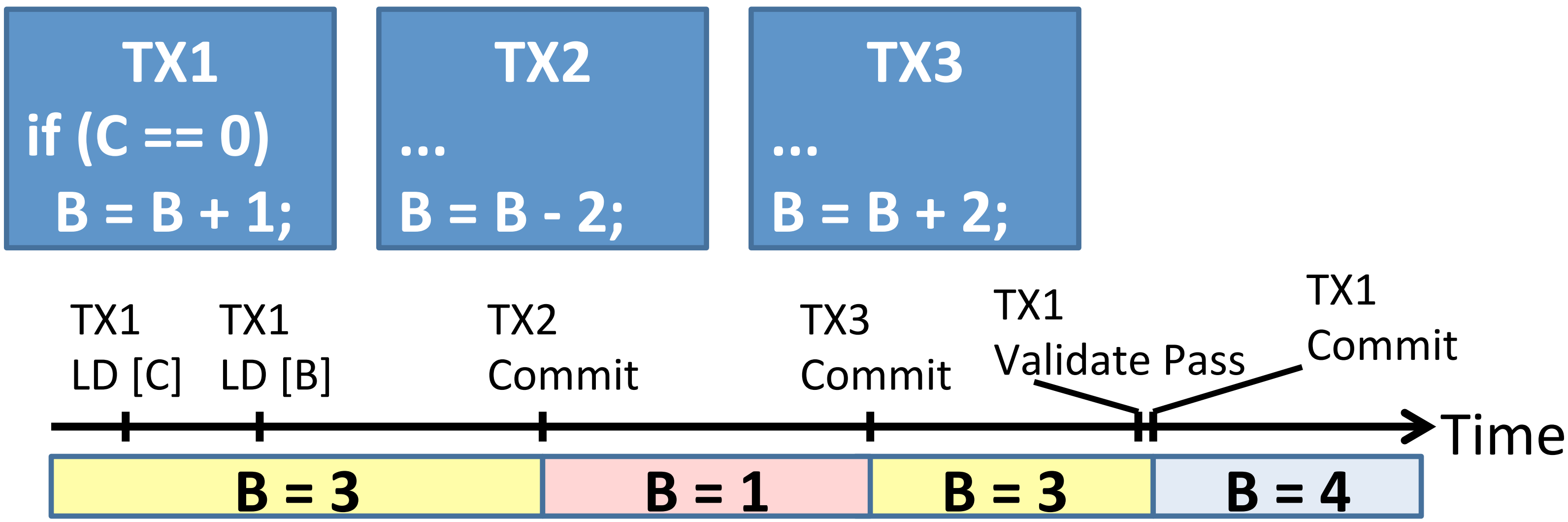



```
while (true) {
   t = top;
   Next = t->Next;
   if (atomicCAS(&top, t, next) == t) break;  // succeeds!
}
```

- **Value-based conflict detection protects *<u>all relevant parts</u>* of the data structure**

ABA Problem?

- If every memory input value is identical,
  the transaction code should generate the same output.
  - No point to re-execute transaction for ABA event.



See Tech. Report: http://www.ece.ubc.ca/~aamodt/papers/wwlfung.tr2012.pdf