

**Flex<sup>TM</sup>**

Flexible Decoupled  
Transactional Memory Support

**Arrvindh Shriraman**

Sandhya Dwarkadas

Michael L. Scott

Department of Computer Science

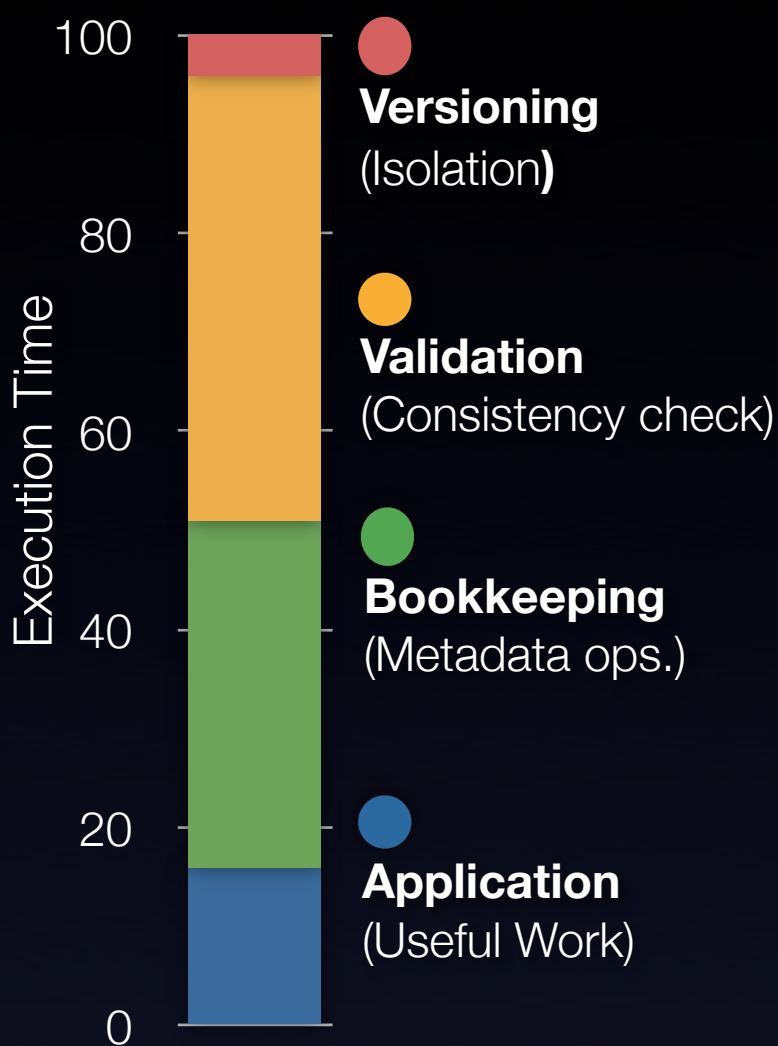


UNIVERSITY *of* ROCHESTER

# Transactions: Our Goal

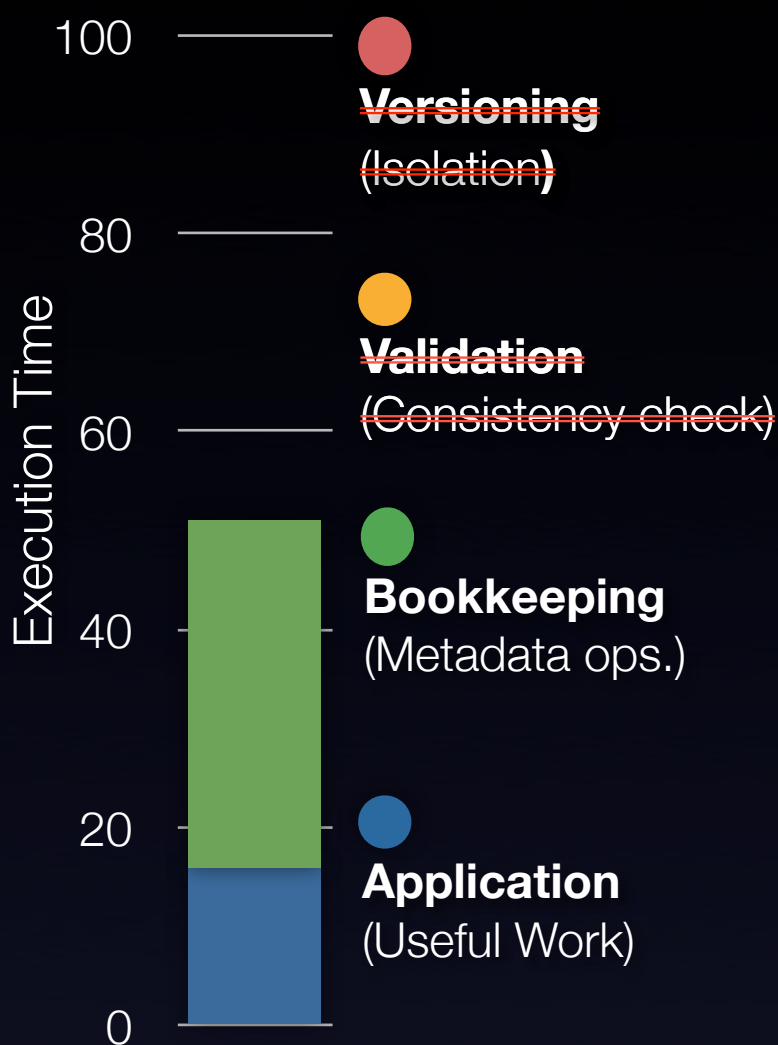
- ✦ Lazy Txs (i.e., optimistic conflict resolution)
  - more concurrency
- ✦ SW coordinates conflict management
  - when (i.e., eagerly or lazily)
  - how (i.e., stalling, who aborts)
- ✦ Limitless Txs
  - Large: cache victimization and paging
  - Long: thread switches

# Flexible Transactional Memory



- ✦ STM (e.g., RSTM)
  - all software approach

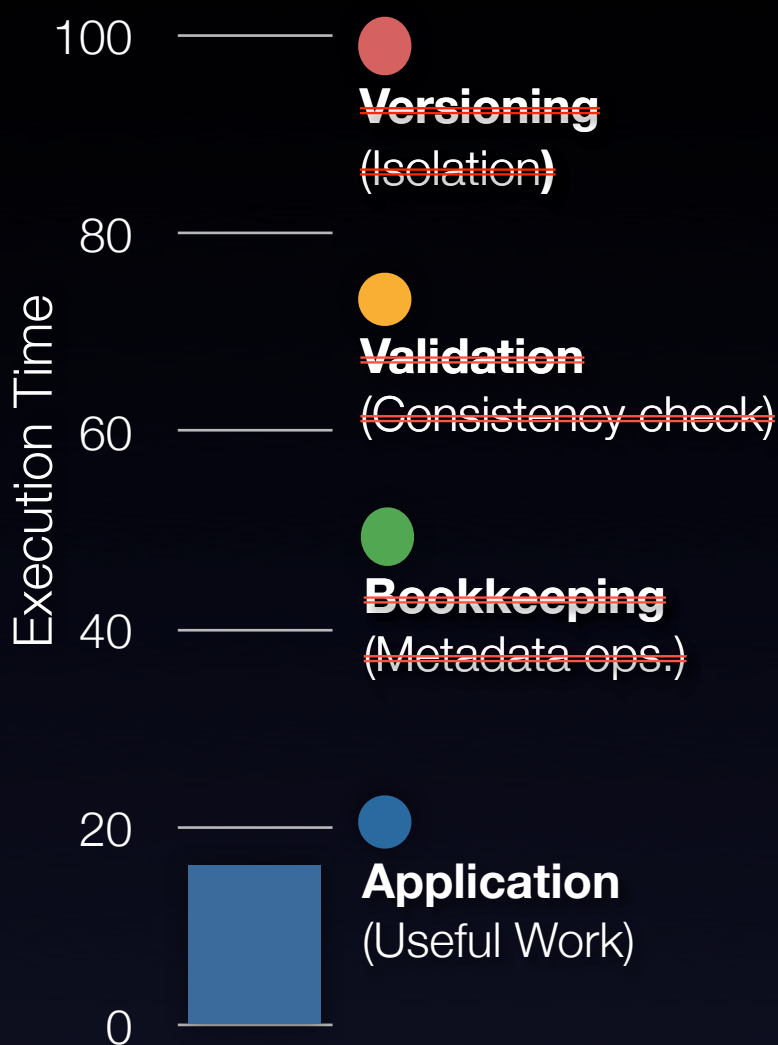
# Flexible Transactional Memory



- ✦ STM (e.g., RSTM)
  - all software approach

- ✦ RTM [ISCA' 07]
  - new cache states help bounded txs
  - software handles large & long txs

# Flexible Transactional Memory



- ✦ STM (e.g., RSTM)
  - all software approach

- ✦ RTM [ISCA' 07]
  - new cache states help bounded txs
  - software handles large & long txs

- ✦ **FlexTM** [this paper]

## **Good Performance**

No per-location software metadata

## **Simple hardware**

No bulk arbiters like lazy HTMs

## **Allows software policy**

# Decoupled Hardware Primitives (1/2)

- ✦ **Separate interchangeable** basic hardware ops. that can be coordinated by software

## Why ?

- ✦ **Minimizes hardware state**
  - small footprint, simplifies virtualization
  - reduces development time
- ✦ **Software accessible**
  - to build transactions & fine-tune policy decisions
  - to repurpose hardware for non-tx applications

# Decoupled Hardware Primitives (2/2)

## 1. Data Isolation (delaying visibility of stores)

- caches buffer speculative values, provide fast-commit
- SW allocates overflow region & HW performs access

## 2. Access Summary (tracking locations accessed)

- maintains list of locations read & written
- check on coherence messages or local memory ops.

## 3. Conflict Summary (tracking data conflict events)

- tracks conflict occurrence and type between processors

## 4. Alert-On-Update

- monitor cache-blocks and trigger handlers

# Outline

- 📌 Preview
- 📌 Data Isolation (aka. Lazy Versioning)
  - Lazy coherence
  - Overflow-Table
- 📌 Conflict Management
- 📌 FlexTM Software
- 📌 Evaluation
- 📌 Summary



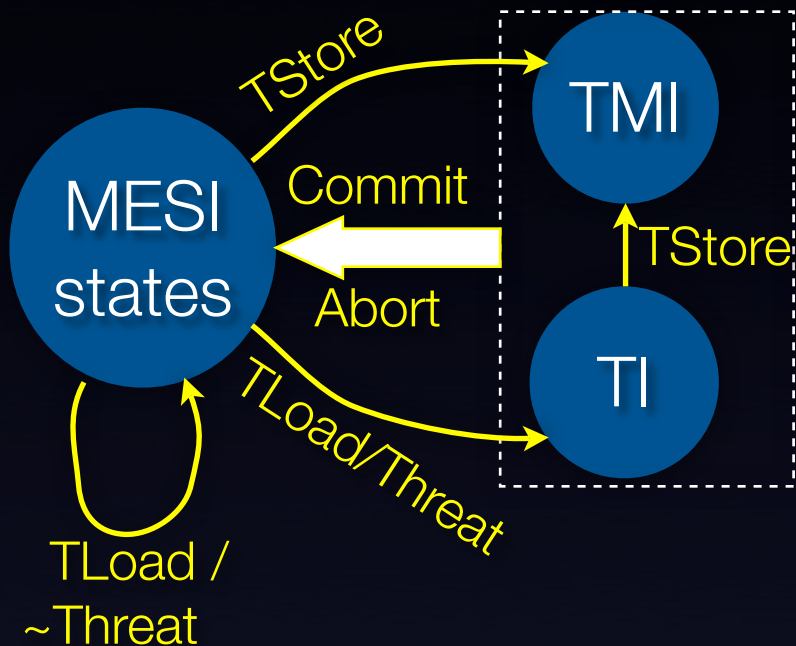
# Lazy Coherence (1/2): Approach

- ✦ Lazy coherence:
  - permit multiple readers & writers for a cache block
  - restore coherence for multiple lines simultaneously
- ✦ Current Research (e.g., TCC, Bulk)
  - bulk arbiters, bulk GetXs, bulk ops. on directory
- ✦ **Our approach: eager messages but lazy coherence**
  - look out for sharer conflicts in standard coherence msgs.
  - continue caching data, but use T-MESI states
  - simple bit-clear ops. convert T-MESI to MESI

**No bulk messages or address ops.**

# Lazy Coherence (2/2): Protocol

- Two new 'T' tagged states: TMI (T+M) and TI (T+I)
- TStores & TLoads denote speculative operations
  - ISA can include instructions or SW can tell HW the regions



- **TMI buffers TStores**
  - allows multiple writers and readers
  - no data response but threaten
  - On commit,  $T+M \Rightarrow M$
  - On abort,  $T+M \Rightarrow T+I \Rightarrow I$
- **TI caches threatened TLoads**
  - cache remotely TStored block
  - On commit/abort,  $T+I \Rightarrow I$

- + cached locations are accessed directly
- + bounded txs perform in-place update

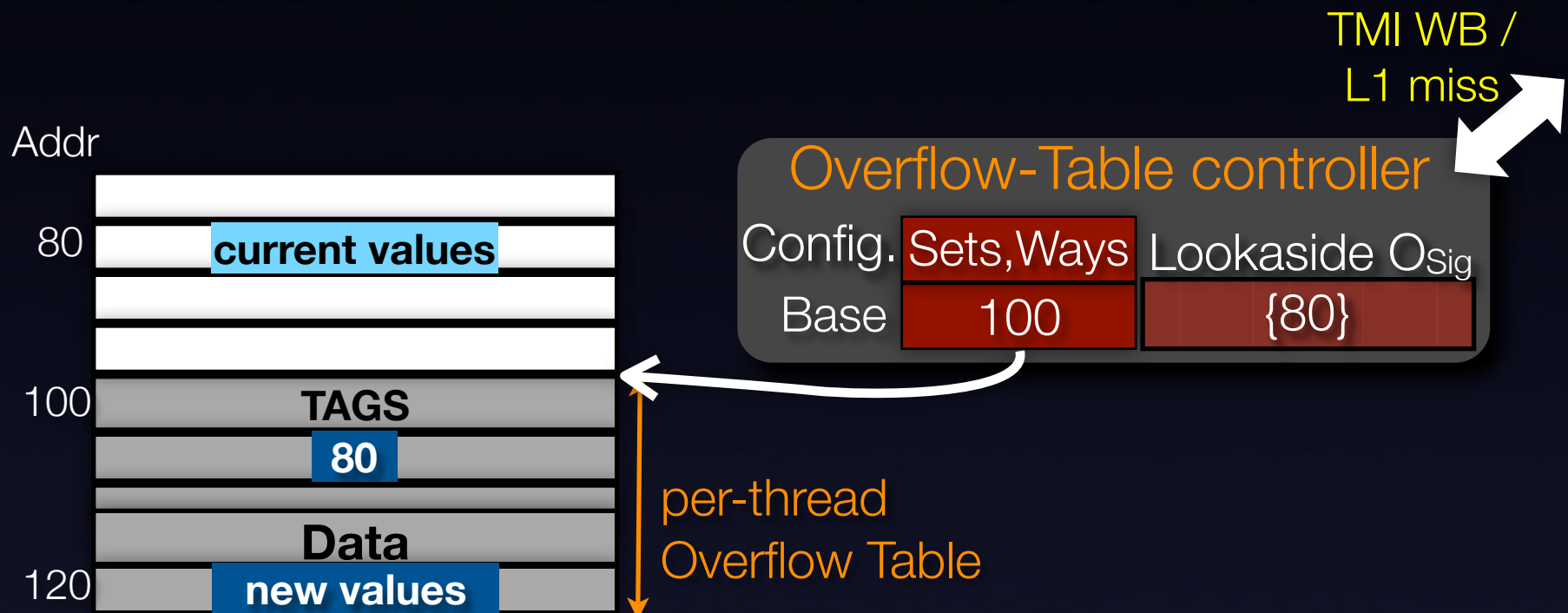
# Overflow Table

**Challenge** : Where to put evicted TMI lines?

**Solution** : Per-thread hash table (in virtual memory)

Hardware controller

- fill table with TMI lines evicted from cache
- removes table entries when reloaded into cache
- performs look-aside transparently on L1 miss in parallel with L2

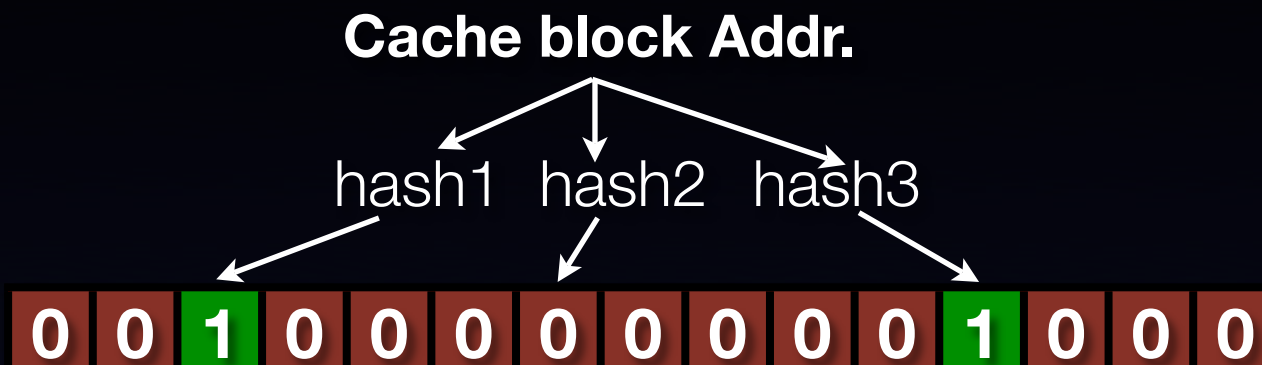


# Outline

- 📌 Preview
- 📌 Data Isolation (aka. Lazy Versioning)
- 📌 Conflict Management (flexible)
  - Access summary signatures
  - Conflict table
  - Alert-On-Update
- 📌 FlexTM Software
- 📌 Evaluation
- 📌 Summary

# Access Summary (1/2): Signatures

- ✦ Signatures [Bulk ISCA'06, LogTM-SE HPCA'07, SigTM ISCA'07]
  - Bloom filters to represent unbounded set of cache blocks
  - approx. representation with false positives



- ✦ Processor has two signatures:
  - $R_{sig}$  ( $W_{sig}$ ) summarizes locations TLoad (TStore)

**Conflict Detection:** Signatures snoop coherence messages

- responder detects conflict and overloads response
- requester picks response and resolves or notes conflict

# Access Summary (2/2): Virtualization

[details in paper]

- ✦ Required to handle long running txs & tx pauses

**Challenge** : How to detect conflicts with suspended txs ?

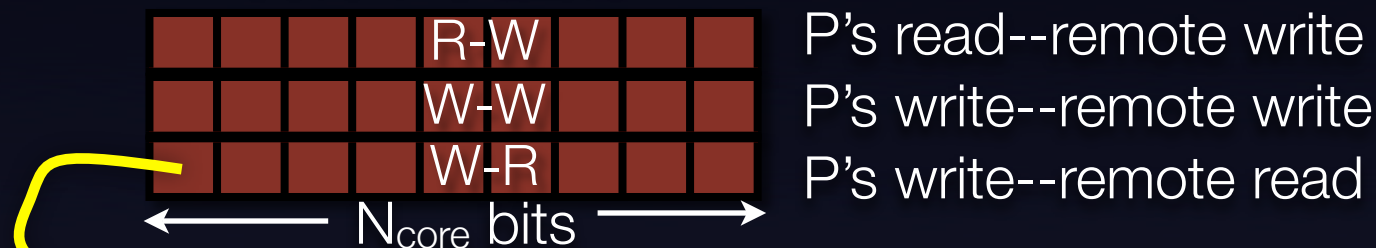
**Solution** : Read and Write summary signatures at the directory,  
(note: does not affect cache hit critical path)

- ✦ Details:
  - merge suspended txns signature with summary sig.
  - all L1 cache misses test signatures
  - **if miss, no further action** necessary
  - **if hit**, trap to software routine that mimics conflict HW

# Conflict Tables: Tracking Conflicts

- ✦ Current HTMs detect and resolve at the same time
  - Eager HTM systems perform both on a conflict
  - Lazy HTM systems perform both at commit time
- ✦ **Our approach: decouple detection from resolution**
  - HW bitmaps record conflict event & expose to SW
  - SW decides when and how to resolve conflicts
- ✦ Per-core conflict bitmap

## Core-P's table



Is there a conflict between P and core i ? Ans: Yes (1) / No (0)



# Conflict Tables: Operation

4 core machine

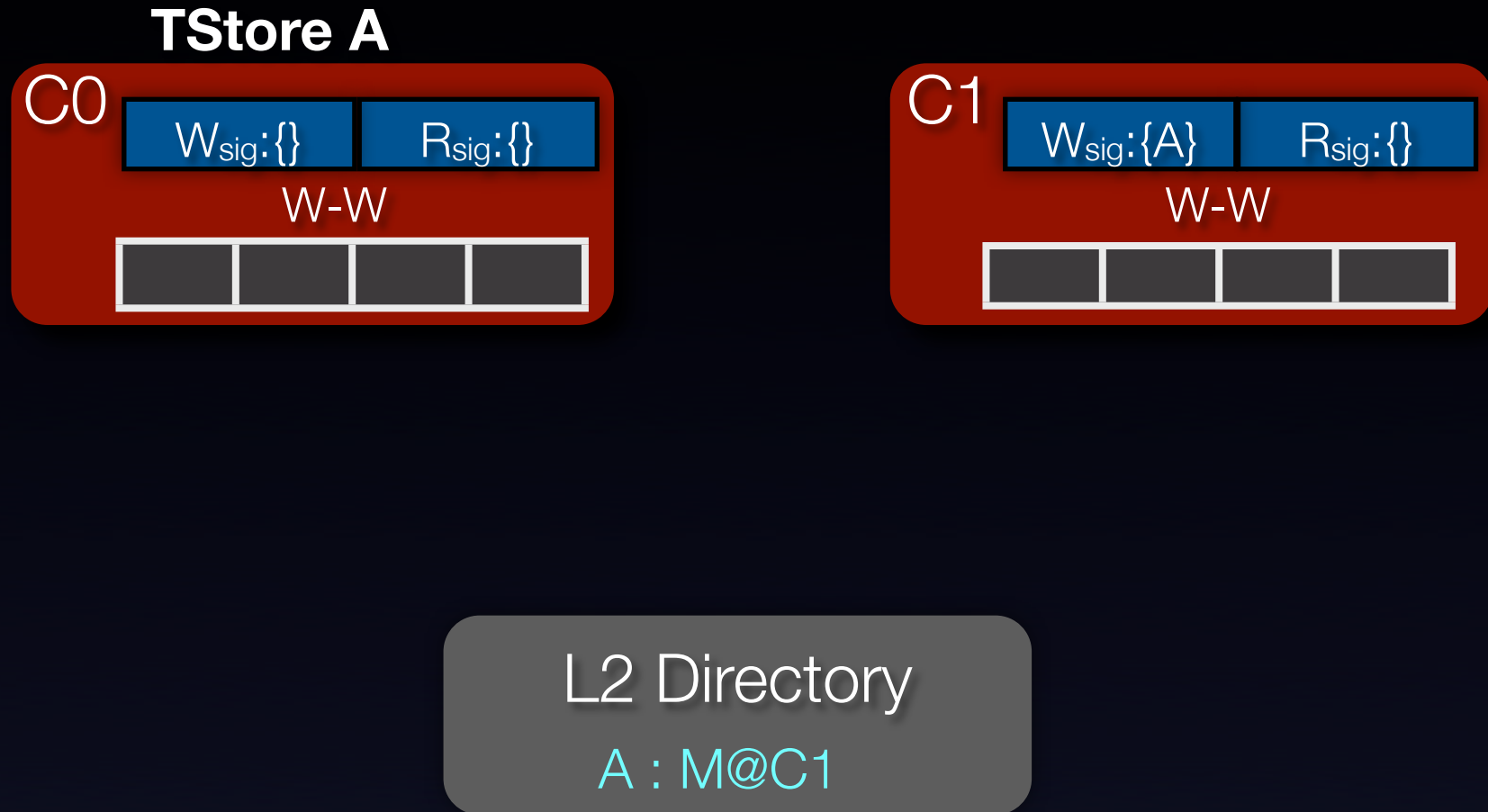


- ✦ Either processor can resolve conflict prior to commit
  - If eager, requester resolves conflict immediately
- ✦ Conflicter known, **no central arbiter** required



# Conflict Tables: Operation

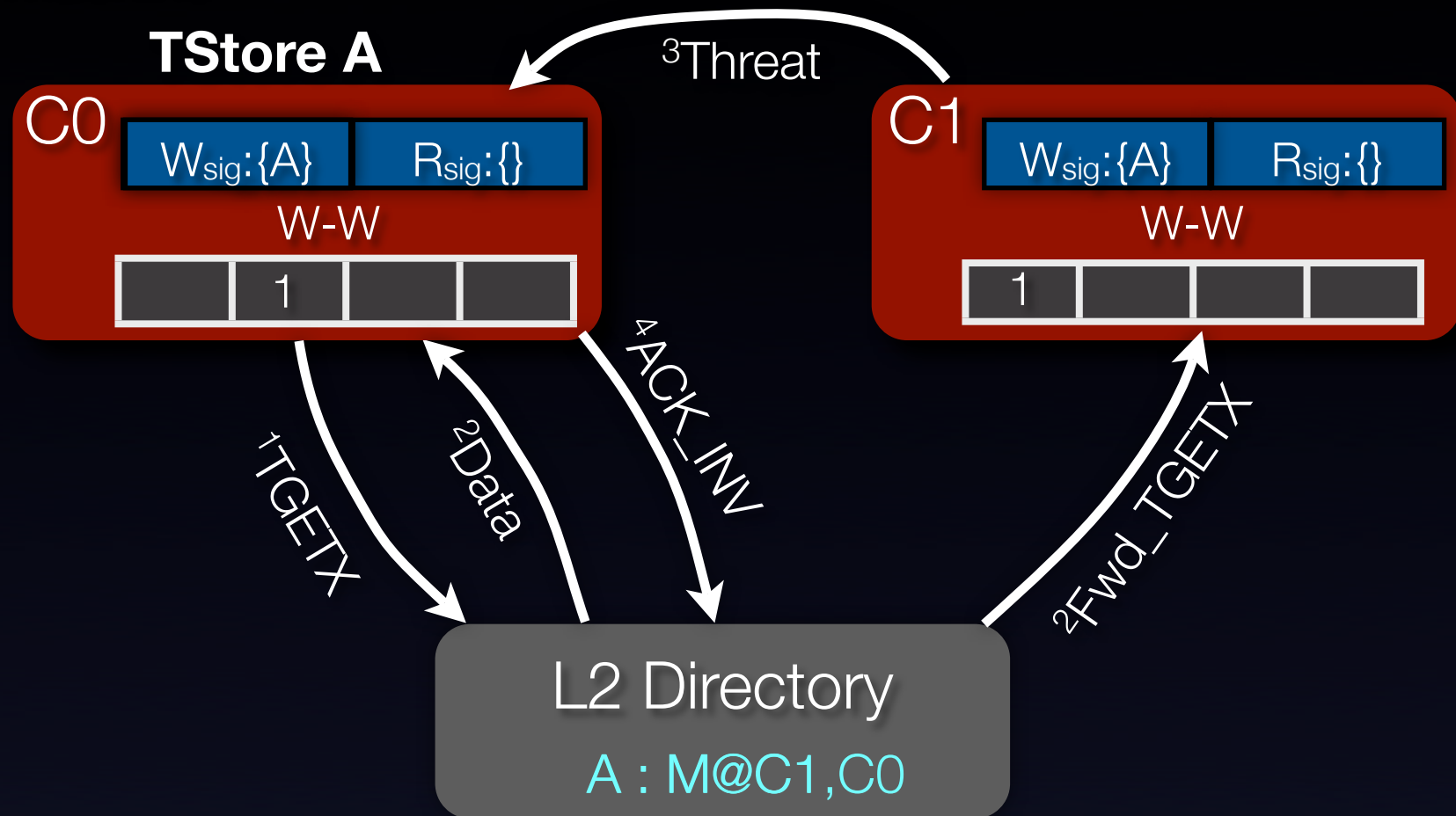
4 core machine



- ✦ Either processor can resolve conflict prior to commit
  - If eager, requester resolves conflict immediately
- ✦ Conflicter known, **no central arbiter** required

# Conflict Tables: Operation

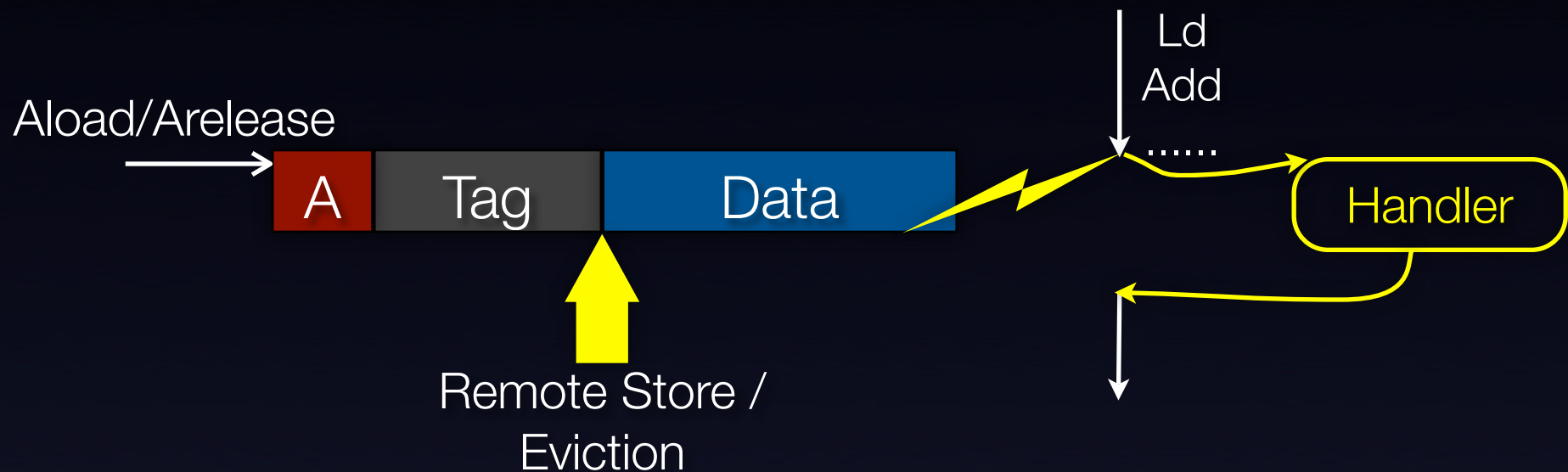
4 core machine



- ✦ Either processor can resolve conflict prior to commit
  - If eager, requester resolves conflict immediately
- ✦ Conflicter known, **no central arbiter** required

# Alert-On-Update (AOU) [ISCA'07]

- Vector specific coherence or update events to the processor in the form of a lightweight event/interrupt
  - on invalidation (capacity eviction or coherence)
  - on access/update (local event)



# Outline

- 📌 Preview
- 📌 Data Isolation (aka. Lazy Versioning)
- 📌 Conflict Management (flexible)
- 📌 FlexTM Software
  - FlexTM Transaction
  - Example
- 📌 Evaluation
- 📌 Summary

# FlexTM Transaction (1/2)

## ✦ Per-Tx descriptor

TSW	<i>active / committed / aborted</i>
State	<i>running / suspended</i>
CM <sub>PC</sub>   Abort <sub>PC</sub>	handler for conflict table events   AOU events on TSW

## ✦ FlexTM deploys

- **Signatures** for detecting and notifying conflicts
- **Conflict Tables** for tracking and managing conflicts
- **T-MESI** for in-cache buffering and OT for cache overflows
- **AOU** for propagating abort events to remote txs.

## ✦ FlexTM software

- checkpoints registers at Begin\_Tx
- manages conflicts; aborts remote tx by changing TSW
- controls commit protocol routine

# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1

**T2** | Begin\_Tx abort\_pc2



**L2 Directory**

# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
ALD TSW0

**T2** | Begin\_Tx abort\_pc2  
ALD TSW1



**L2 Directory**

TSW0 : M@C0  
TSW1 : M@C1

# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
ALD TSW0  
TSt A

**T2** | Begin\_Tx abort\_pc2  
ALD TSW1



A : M@C0

**L2 Directory**

TSW0 : M@C0

TSW1 : M@C1



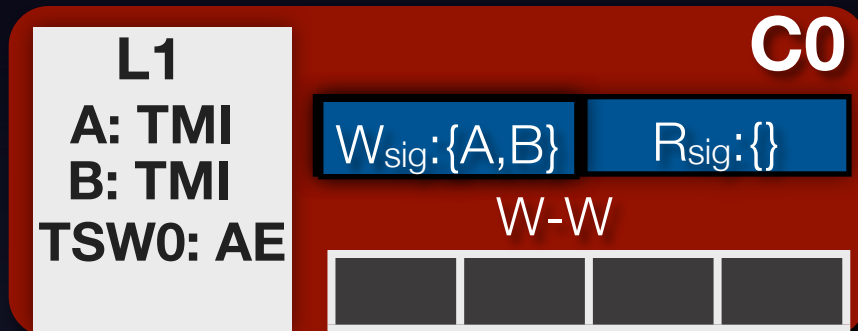
# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
 ALD TSW0  
 TSt A  
 TSt B

↓

**T2** | Begin\_Tx abort\_pc2  
 ALD TSW1

↓



# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
 ALD TSW0  
 TSt A  
 TSt B

↓

**T2** | Begin\_Tx abort\_pc2  
 ALD TSW1  
 TSt A

↓



# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
 ALD TSW0  
 TSt A  
 TSt B

↓

**T2** | Begin\_Tx abort\_pc2  
 ALD TSW1  
 TSt A  
 TSt B

↓



# Lazy Transactions: Example

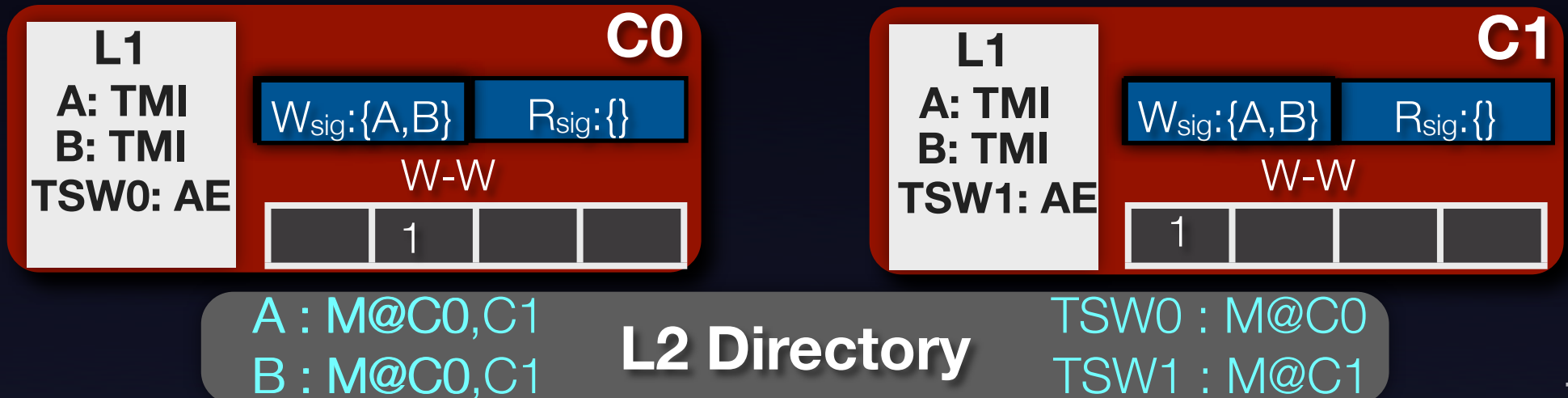
**T1** | Begin\_Tx abort\_pc1  
 ALD TSW0  
 TSt A  
 TSt B

**T2** | Begin\_Tx abort\_pc2  
 ALD TSW1  
 TSt A  
 TSt B

Conflict & Commit protocol

For-each i set in W-R or W-W  
 CAS (Status[i], ACT, ABORT)

In software, decentralized, minimal overhead  $\propto$  No. of conflicting Tx



# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
 ALD TSW0  
 TSt A  
 TSt B

Conflict & Commit protocol

For-each i set in W-R or W-W  
 CAS (Status[i], ACT, ABORT)

**T2** | Begin\_Tx abort\_pc2  
 ALD TSW1  
 TSt A  
 TSt B

 Conflict Handler!

In software, decentralized, minimal overhead  $\propto$  No. of conflicting Tx



# Lazy Transactions: Example

**T1** | Begin\_Tx abort\_pc1  
 ALD TSW0  
 TSt A  
 TSt B

Conflict & Commit protocol

For-each i set in W-R or W-W  
 CAS (Status[i], ACT, ABORT)  
 CAS-Commit Status[id]

**T2** | Begin\_Tx abort\_pc2  
 ALD TSW1  
 TSt A  
 TSt B

 Conflict Handler!

In software, decentralized, minimal overhead  $\propto$  No. of conflicting Tx



# Outline

- 📌 Preview
- 📌 Data Isolation (aka. Lazy Versioning)
- 📌 Conflict Management (flexible)
- 📌 FlexTM Software
- 📌 Evaluation
  - Speedup
  - Conflict resolution tradeoffs
  - Other results
- 📌 Summary

# Evaluation set-up

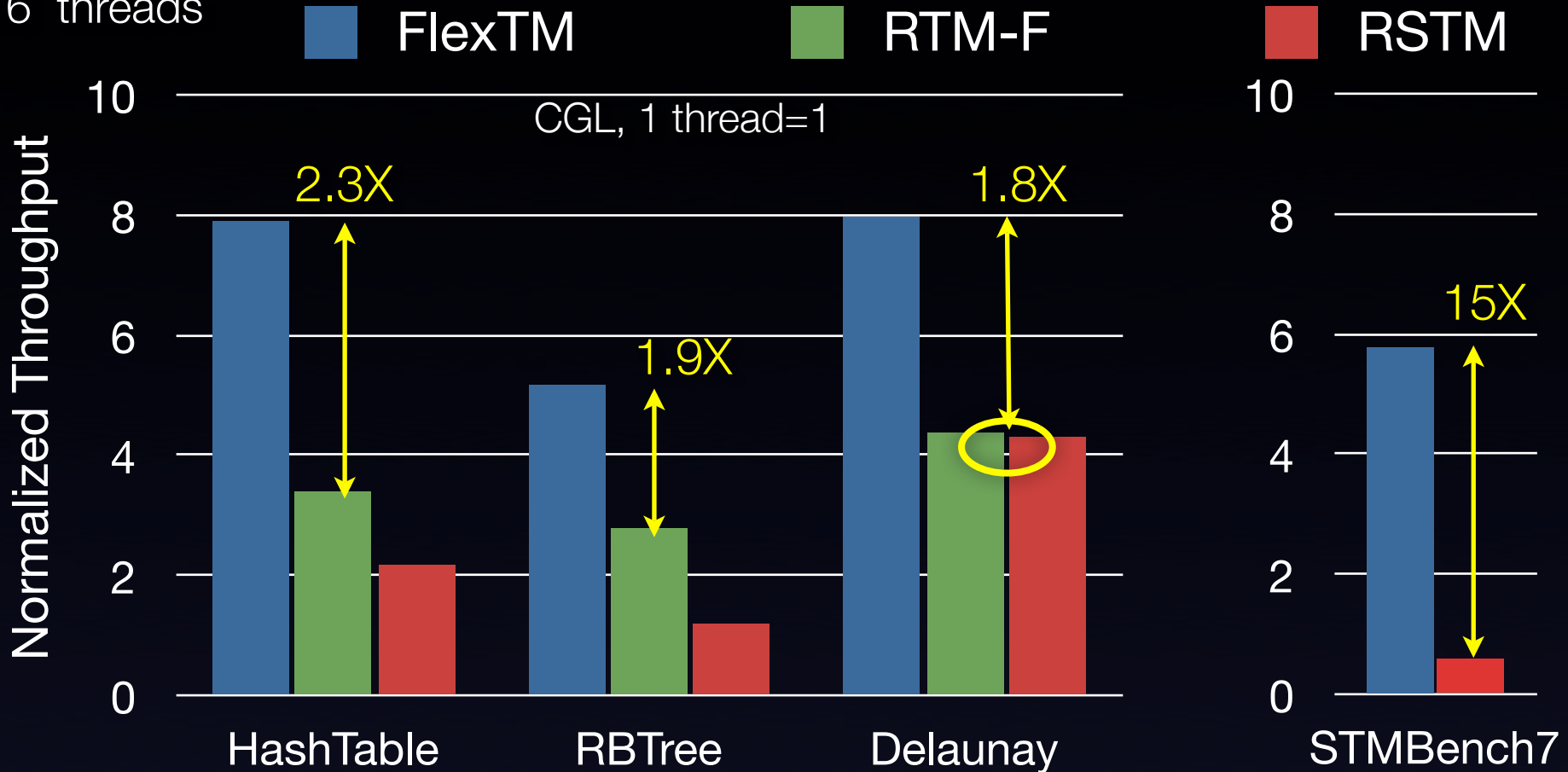
- ✦ Full system simulation, GEMS/SIMICS framework
  - 16 core CMP with shared L2
  - ORIGIN 2000 like coherence protocol (3 hop requests and silent evictions)
- ✦ Workloads
  - **Data Structures:** Hash, RBTREE, LFUCache, Graph
  - **Applications:** Scott's Delaunay, STAMP\*, STMBench7
- ✦ Runtime systems
  - CGL, FlexTM (HTM interface), RTM-F, RSTM, & TL2
  - Polka conflict manager

\* - STAMP does not (yet?) interface with RTM-F and RSTM



# FlexTM is Fast (1/2)

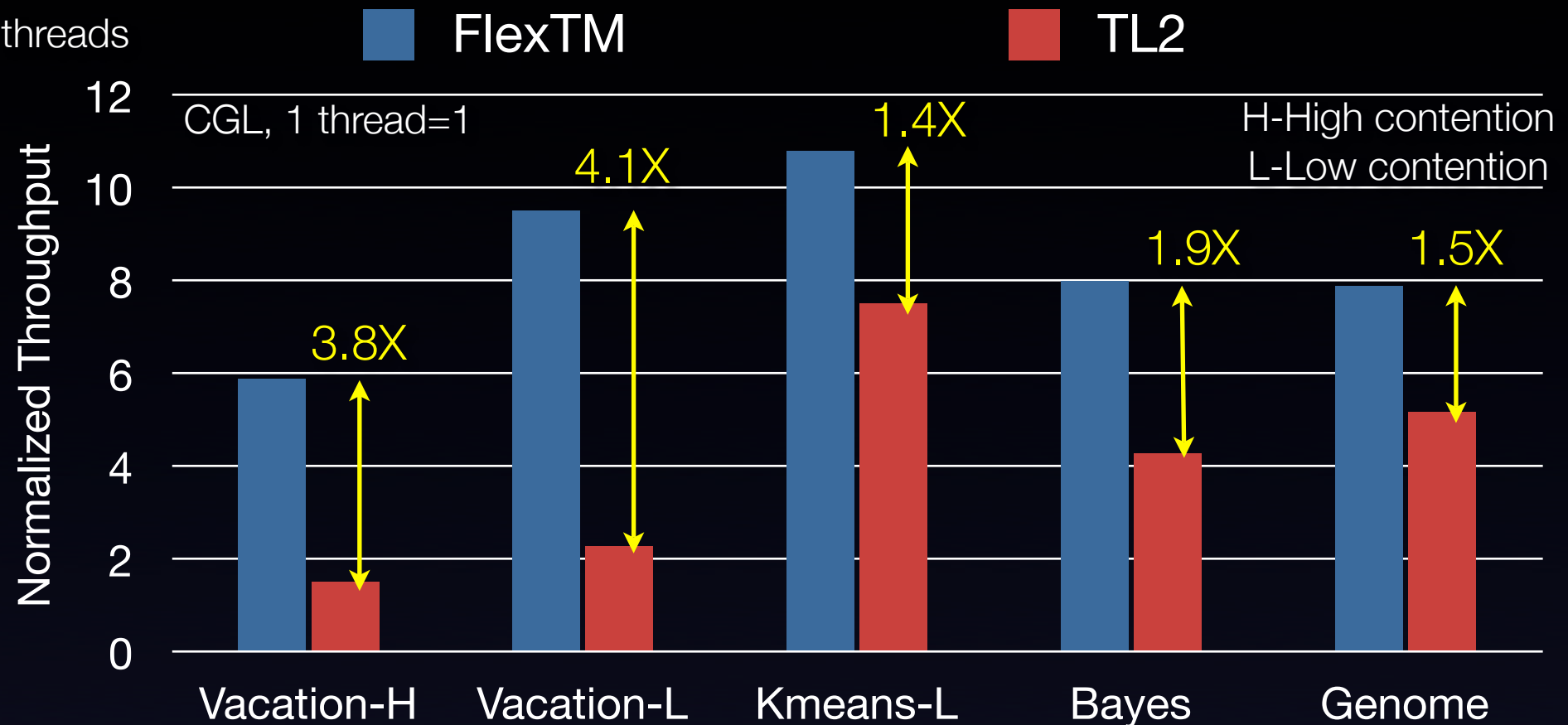
16 threads



- ✦ FlexTM gains over RTM-F proportional to SW bookkeeping overheads
  - software metadata management ~50% of tx latency
- ✦ FlexTM gains over RSTM comparable to rigid policy HTMs

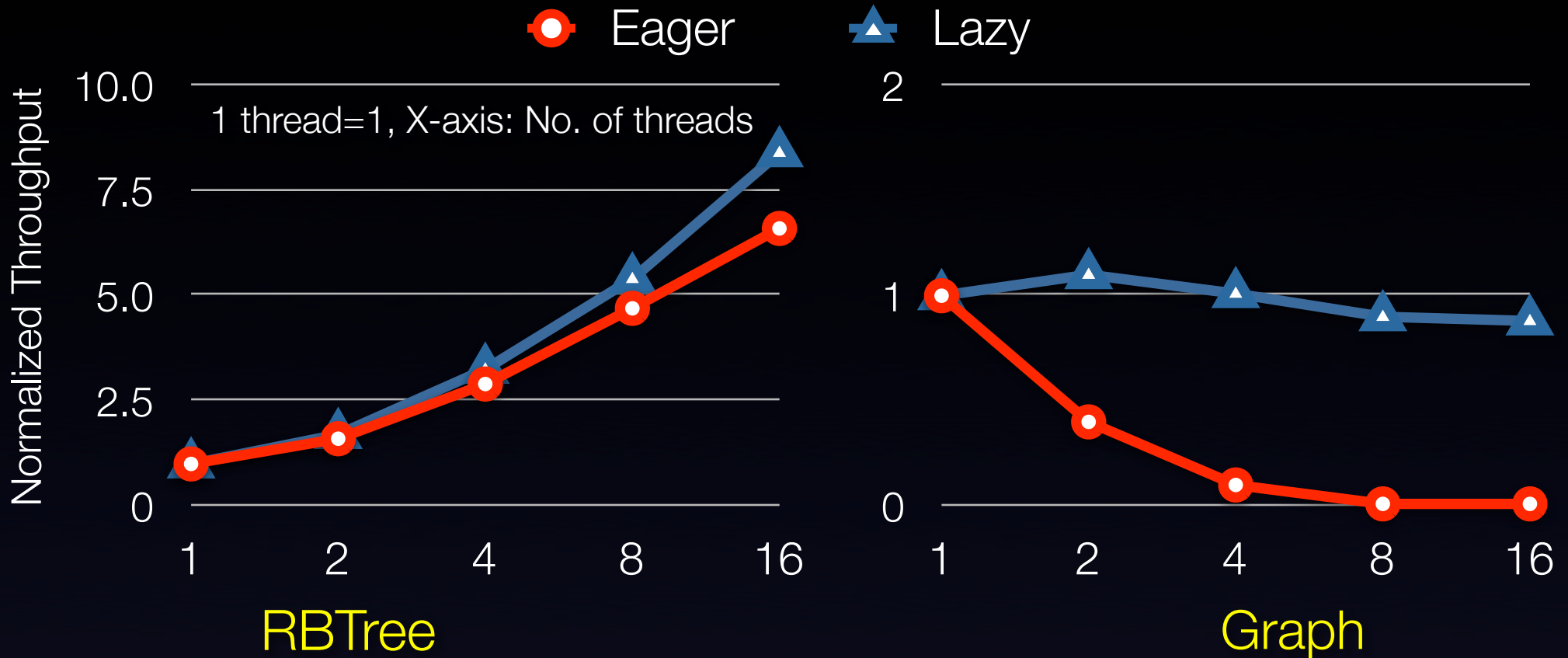
# FlexTM is Fast (2/2)

16 threads



- ✦ Kmeans-L and Genome performance gains lower
  - TL-2 per-access overheads low (i.e., high instructions / mem\_access)
- ✦ Performance gains in Vacation higher
  - lower number of instructions per memory word accessed

# Lazy mode aids progress

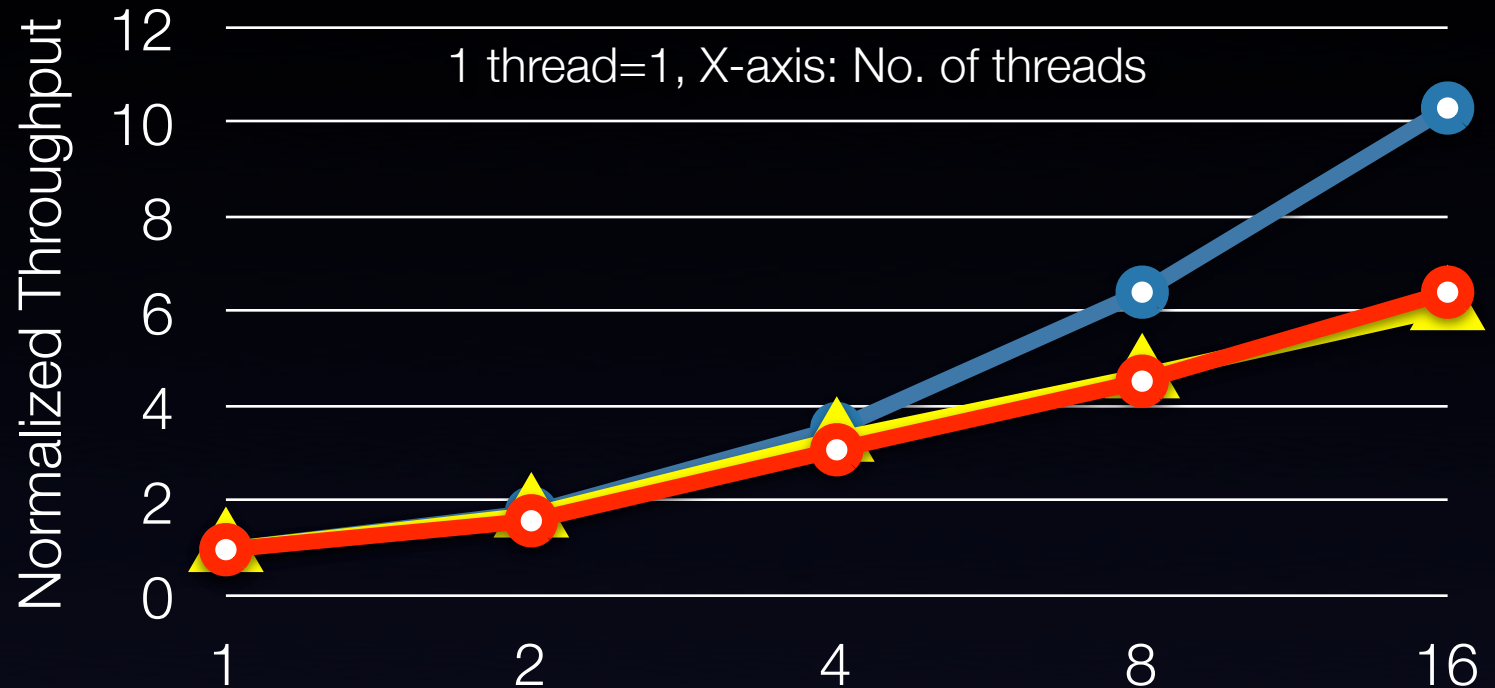


- ✦ Lazy provides more commits
- ✦ Exploits R-W sharing, allows reader & writers to commit in parallel

- ✦ Eager causes cascaded stalls and aborts
- ✦ Lazy narrows conflict window

# Mixed-mode can be better

**STMbench7**  Eager  Lazy  EagerWW-LazyRW



- Long writer (~1ms) mixed with short readers (tens thousands cycles)
  - Pair-wise conflicts between writers, conflicts with multiple readers
- Eager doesn't permit R-W sharing and reduces reader throughput
- Lazy permits W-W sharing, but wastes writer work on aborts

Best Policy: Eager-WW with Lazy-RW

# Other Results

- ✦ **Area analysis** [in paper]
  - increase in core area small, OoO (0.6%), InO (3%)
  - minimal change to pipeline, most hardware on L1 miss
- ✦ **Comparison with Central-Arbiter HTM** [in paper]
  - broadcasts and central arbiters are an overkill
  - de-centralized SW commit is efficient & important

## Non-Tx Applications

- ✦ **Watchpoints** [in TR-925]
  - Two memory monitoring primitives, AOU & Signatures
  - SW framework for detecting buffer overflows, memory leaks etc.
  - 15-50X speedup over binary instrumentation

# Summary

- ✦ Decouple TM hardware components to
  - reduce HW complexity
  - enable deployment for varied purposes
- ✦ FlexTM
  - HW manages TM operations, SW manages policy
  - decentralized conflict and commit protocol in SW
- ✦ Conflict management
  - laziness is an important design requirement
  - provides best value when left under software control

# Summary

## Questions ?

- Decouple TM hardware components to
  - reduce HW complexity
  - enable deployment for varied purposes

<http://www.cs.rochester.edu/research/cosyn>

<http://www.cs.rochester.edu/research/synchronization>

- FlexTM
  - HW manages TM operations, SW manages policy
  - decentralized conflict and commit protocol in SW

- **Acknowledgments**

– Multifacet Research group, Wisconsin

– STAMP group, Stanford

– Transaction Benchmark group, EPFL

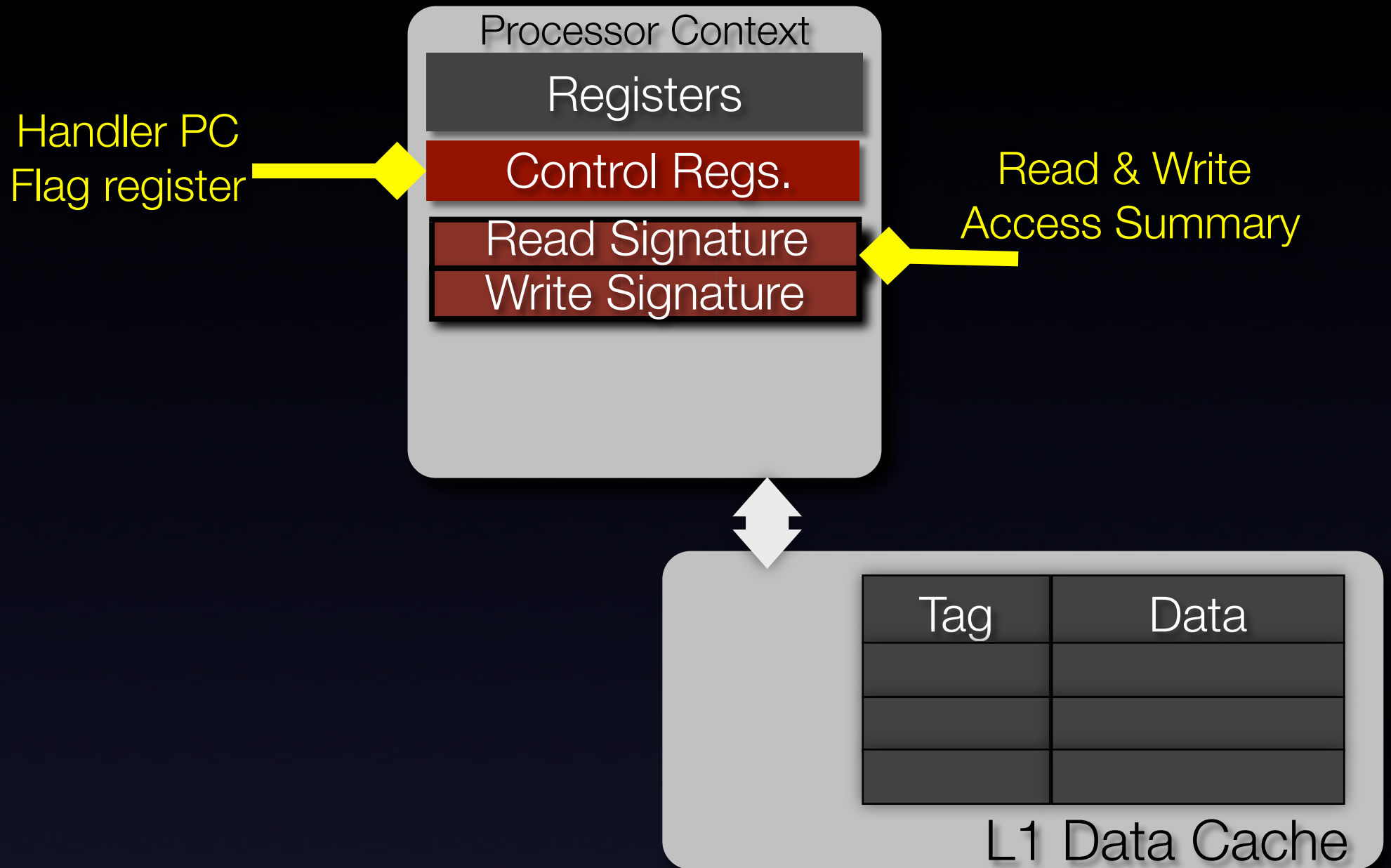
– Shan Lu, Opera group, Illinois



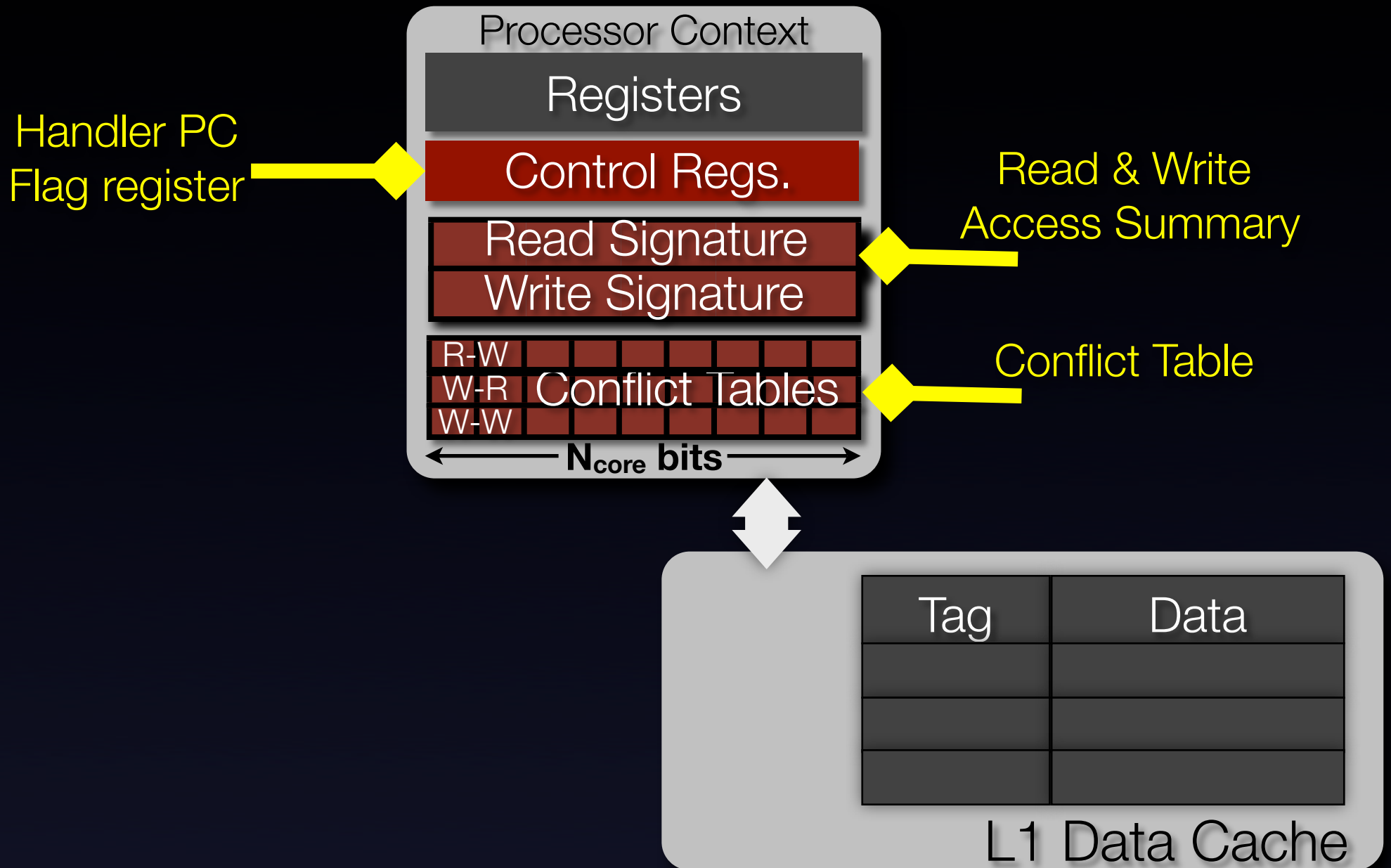




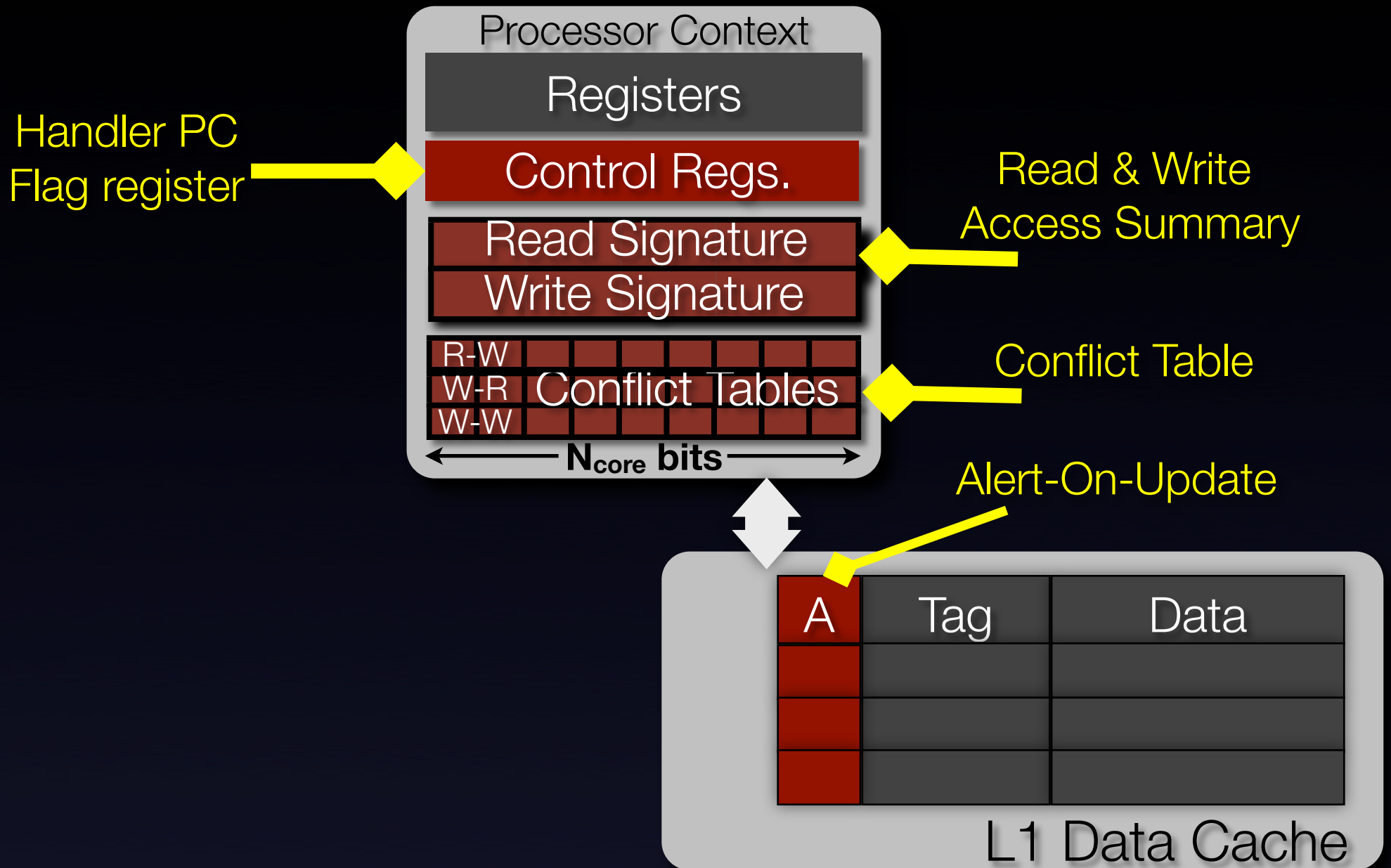
# FlexTM per-Core Hardware



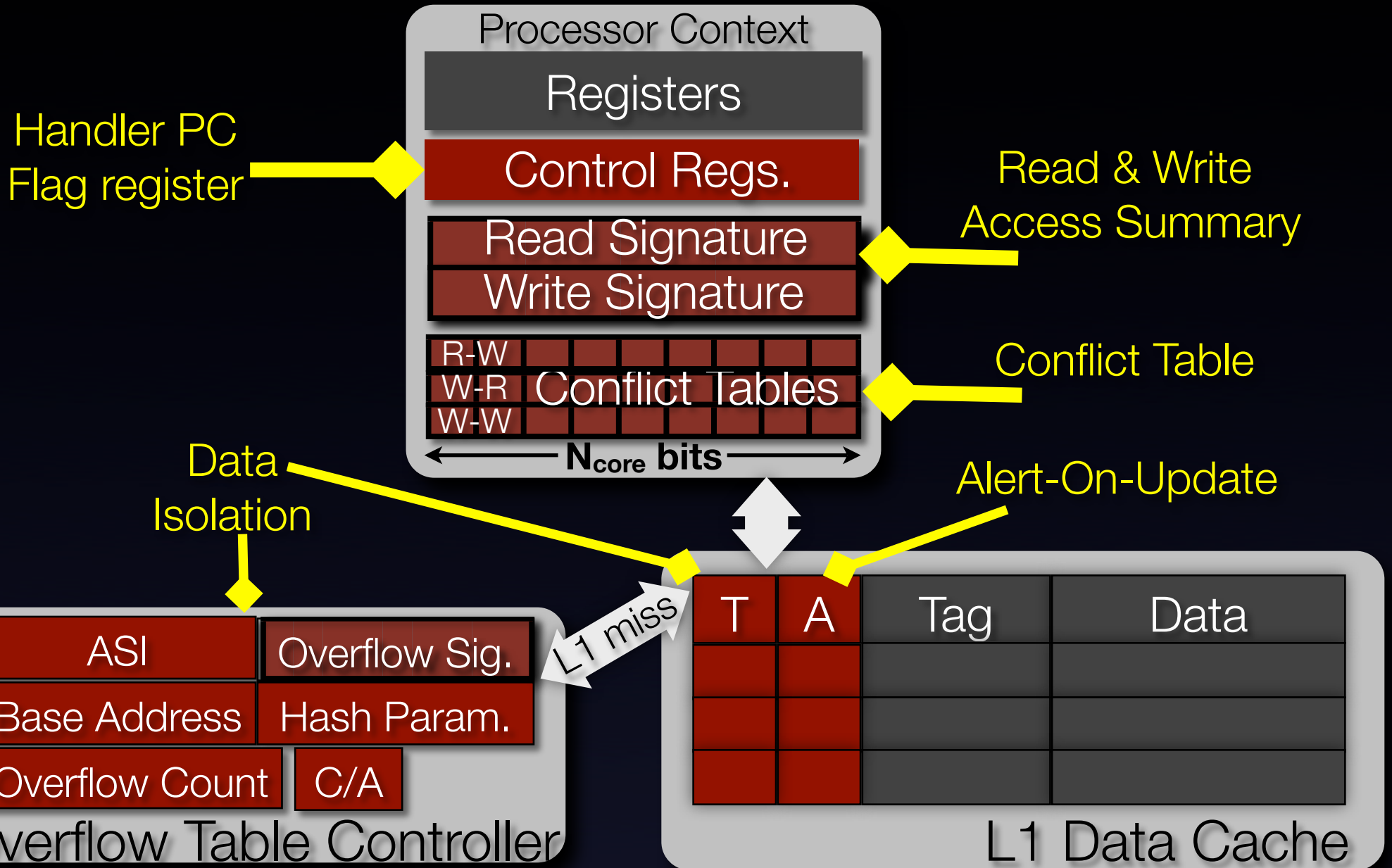
# FlexTM per-Core Hardware



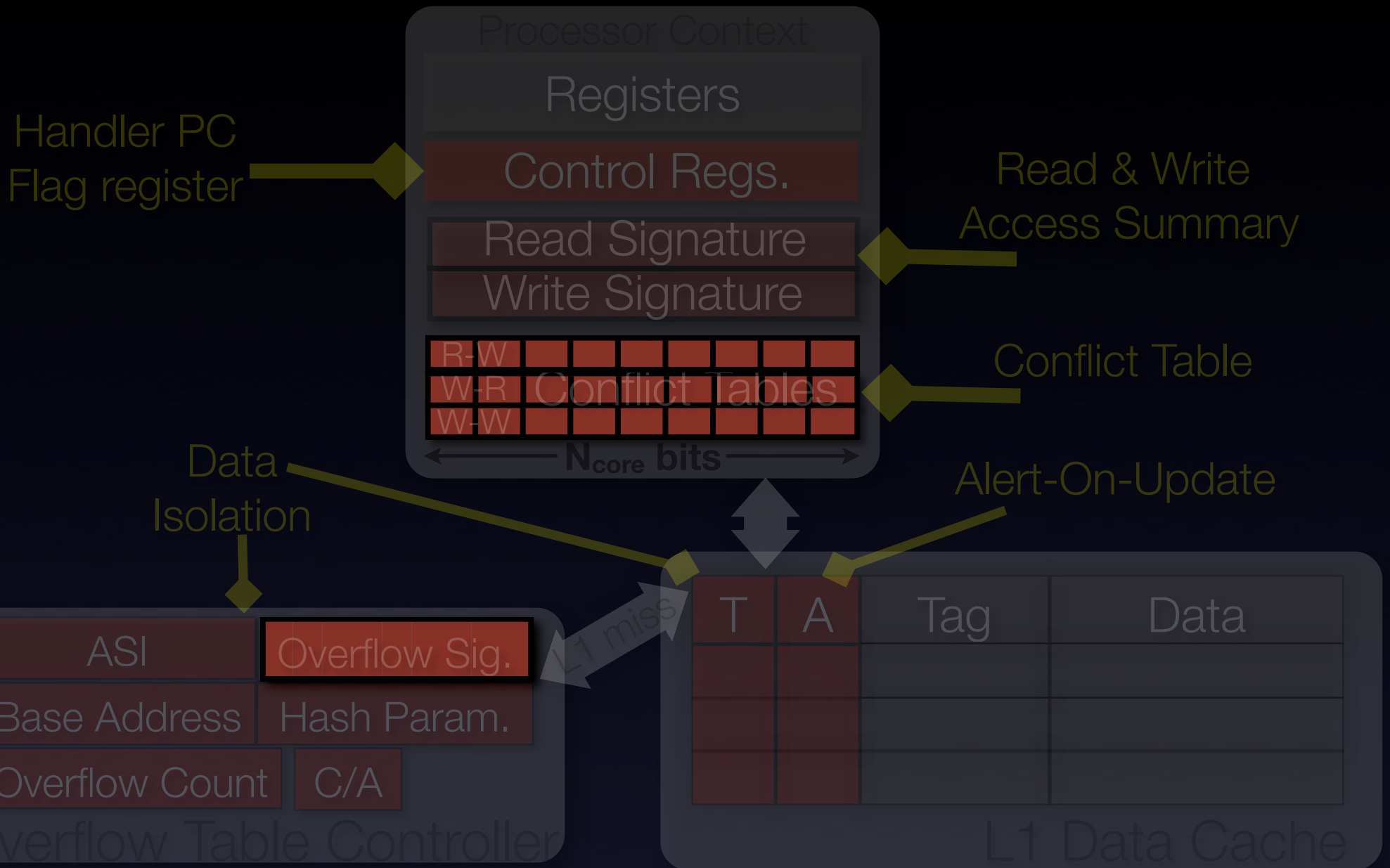
# FlexTM per-Core Hardware



# FlexTM per-Core Hardware



# FlexTM per-Core Hardware



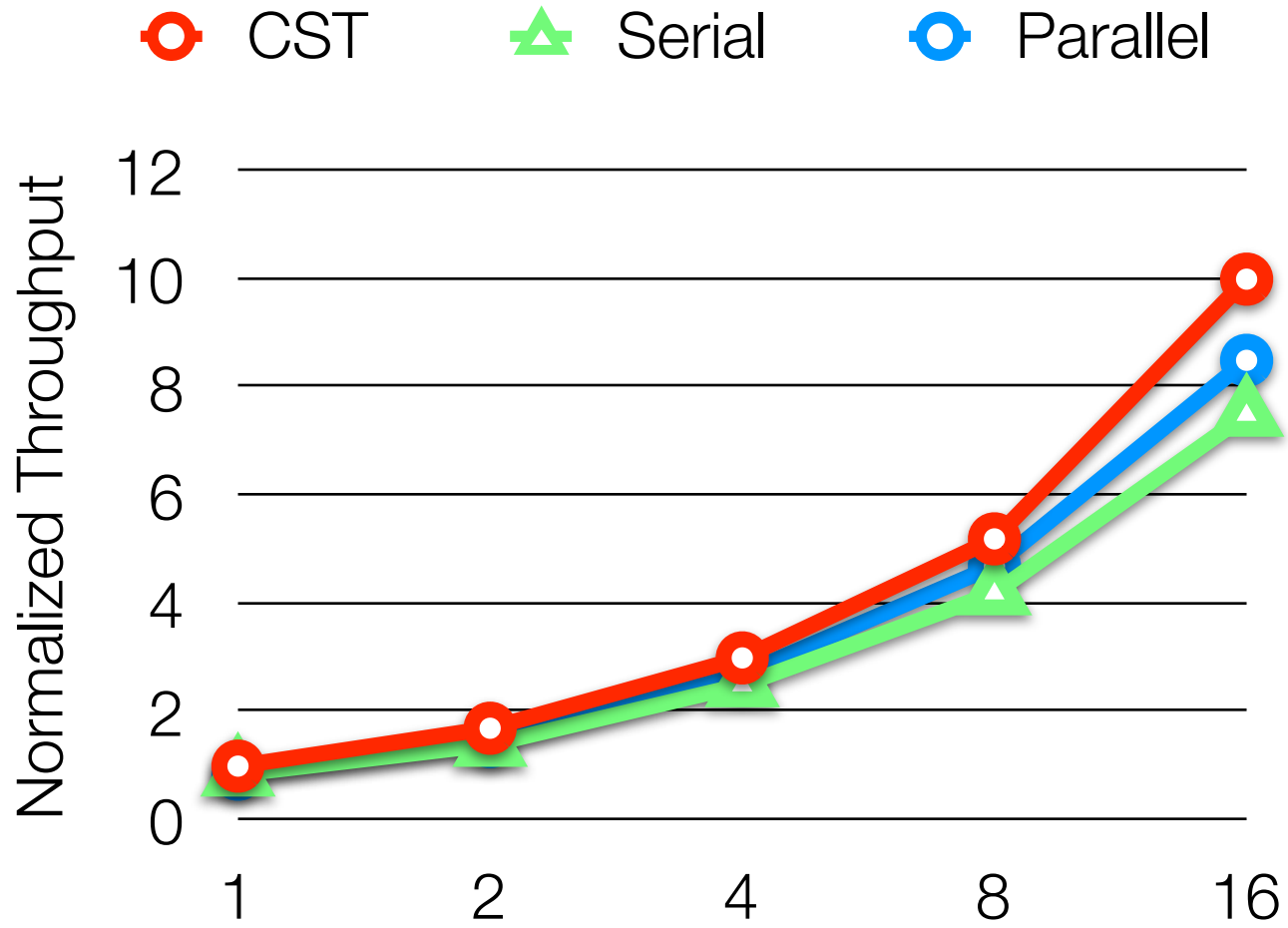


# Flex™ Area Complexity

	Core2	Power6	Niagara2
Orig. Core Area	32mm <sup>2</sup>	53mm <sup>2</sup>	12mm <sup>2</sup>
L1 area	1.8mm <sup>2</sup>	2.6mm <sup>2</sup>	0.4mm <sup>2</sup>
Signatures (2Kbit)	0.10%	0.12%	2.1%
Overflow Control	0.5%	0.45%	0.3%
%L1D area inc.	0.35%	0.3%	3.9%
% core area inc.	0.61%	0.58%	2.5%

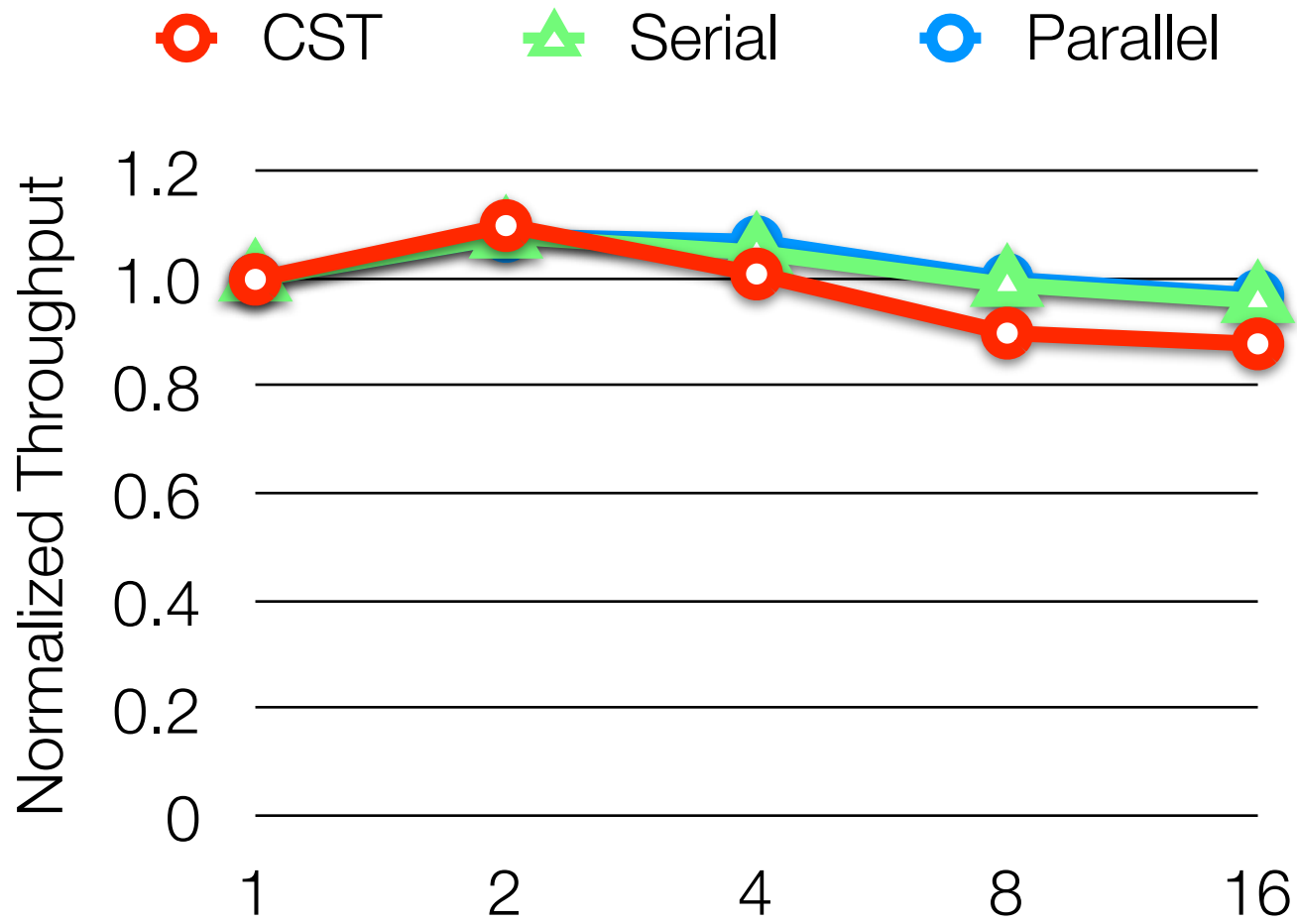
- ✦ Effect on the processor core minimal
  - OoO cores (~0.6%), In-Order (~4%)
- ✦ Negligible effect on L1 latency
  - small area effects, data array is the critical path
- ✦ Signature effects noticeable only on Niagara2
  - 8-way SMT needs 16 2Kbit signatures (4KB state)

# Hash Table





# RandomGraph






# FlexWatcher: Memory Bug Detection

- ✦ FlexTM HW provides two HW primitives for watching memory
  - AOU precisely monitors cache block aligned regions but is limited by cache size
  - Signatures provided unlimited monitoring but are vulnerable to false positives.
- ✦ Extended the ISA to support them as first-class entities
  - insert, member, read-index, activate, clear etc
- ✦ Developed a software bug detection tool
  - add required addresses to signatures
  - HW checks local & remote accesses against the signatures.
  - triggers SW trampoline on signature hits
  - handler disambiguates, if false positive return to execution



# FlexWatcher Evaluation

- ✦ BugBench from illinois, set of real-life programs with known bugs.
- ✦ Bugs  detected
  - **Buffer Overflow**  
**Solution:** Pad all heap allocated buffers with 64bytes, watch padded locations
  - **Memory Leak**  
**Solution:** Monitor all heap allocated objects and update the address's timestamp on access.
  - **Invariant Violation:**  
**Solution:** ALoad cache line of interested variable X. On AOU handler trigger assert program specific invariants.



# FlexWatcher Performance

- Compared against **Discover**, popular SPARC binary instrumentation tool from 

Benchmark	Bug	FlexWatcher	Discover
BC	BO	1.5X	75X
GZIP	BO	1.15X	17X
GZIP <sup>2</sup>	IV	1.05X	N/A
Man	BO	1.80X	65X
Squid	ML	2.50X	N/A

Execution time normalized to sequential thread performance  
FlexWatcher overheads were estimated on the simulator  
Discover overheads were estimated on a Sun T1000 server