

Tradeoffs in Transactional Memory Virtualization

JaeWoong Chung

Chi Cao Minh, Austen McDonald,
Travis Skare, Hassan Chafi, Brian D. Carlstrom,
Christos Kozyrakis, Kunle Olukotun

Computer Systems Lab
Stanford University
<http://tcc.stanford.edu>



TM Virtualization

■ Transactional Memory (TM)

- TM is a promising solution for parallel programming
- Hardware TM delivers best performance
 - But, hardware resources are limited

■ Virtualization of hardware TM systems

- What if cache capacity is exhausted?
 - Space virtualization: cache overflow, paging, thread migration, ...
- What if a transaction is interrupted?
 - Time virtualization: interrupts, context switches, ...
- What if transactions are deeply nested?
 - Depth virtualization
- It is crucial to address these issues properly for practical HTMs

Design Options for TM Virtualization

- **TM is virtualized by overflowing transactions to VM**
 - It is yet another TM system implementation
- **Granularity of data management**
 - Word Vs. cache-line Vs. page level
- **Conflict detection strategy**
 - Optimistic Vs. pessimistic
- **Implementation approach**
 - Hardware & firmware Vs. operating system Vs. user software
- **See paper for detailed discussion on options**

Previous Work

■ Hardware solutions

- UTM [HPCA'05], VTM [ISCA'05], PTM [ASPLOS'06]
 - Primarily cache-line granularity
 - Hardware manages overflowed data and metadata in virtual memory

+ Good performance for all workload cases

- Expensive, extra hardware mostly idle

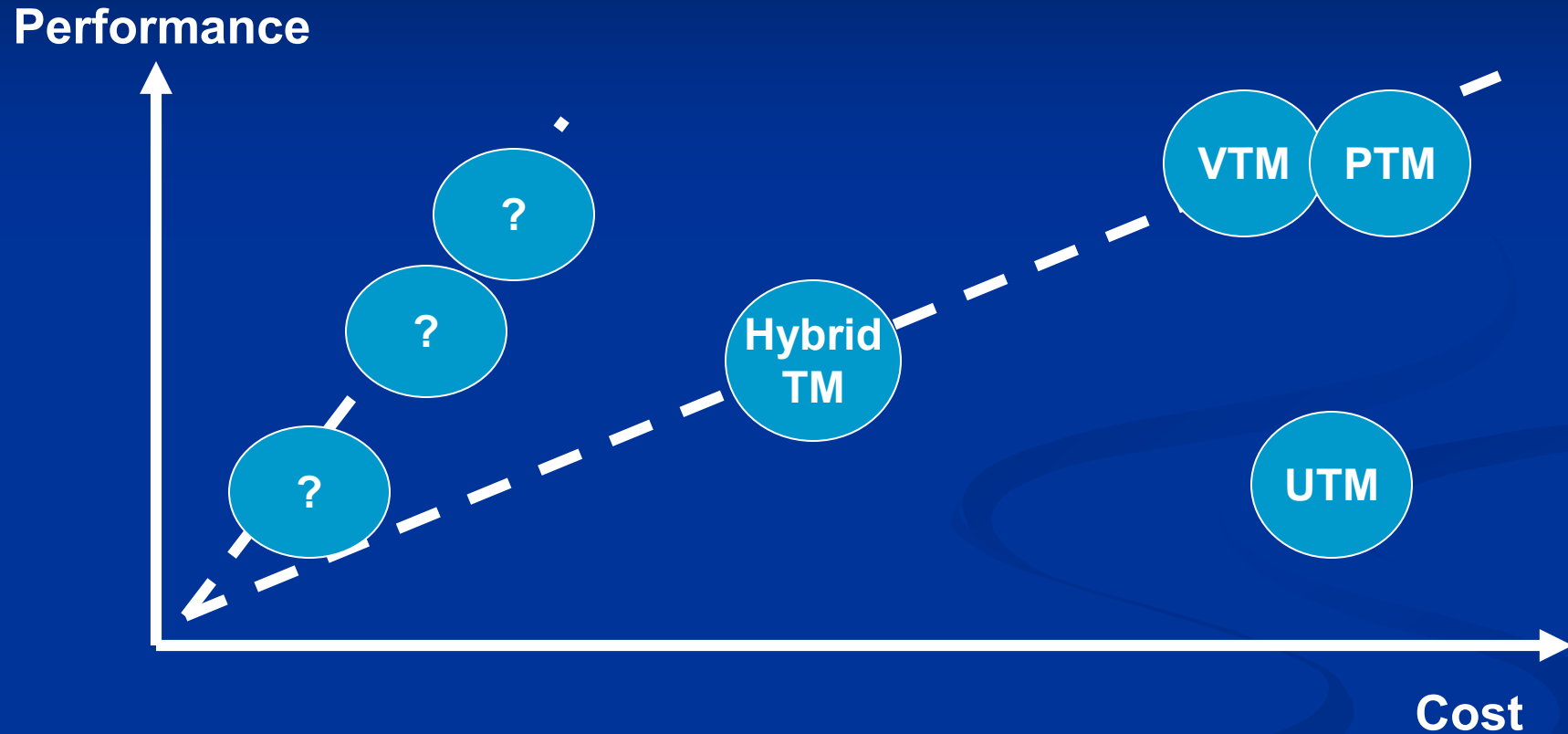
■ User software solutions

- Hybrid TM [PPoPP'06 & ASPLOS' 06]
 - Primarily object level granularity
 - Software TM for virtualization, hardware TM for acceleration

+ No additional hardware

- Two versions of code, lower performance in some cases

Virtualization Design Space



- Tradeoff: common-case performance Vs. HW/SW cost
- Overflows, interrupts, and deep nesting are rare [HPCA'06]

XTM: eXtended TM

■ Goals

- Virtualize all 3 dimensions of TM (space, time, depth)
- Low HW cost and completely transparent to user SW
- Does not slow down coexisting HW transactions

■ Assumption

- Overflows, interrupts, and deep nesting are rare [HPCA'06]

■ Key idea: virtualization through the operating system

- Builds upon existing VM support
- Data versioning & conflict detection at page granularity
- Similar to page-based software DSM systems

■ 3 designs at different performance/cost points

- XTM-base, XTM-g, XTM-e

XTM-base Overview

■ Basic operation

- On HTM overflow, rollback and restart in SW mode
- At the first access, create a copy of original (master) page
 - Change the address mapping to the copy (private page)
 - Transactional data in private page, committed data in master page
- At commit, make the private page the new master page
- All orchestrated by the operating system (no HW)

■ Conflict detection: pessimistic Vs. optimistic

- Pessimistic: use TLB shoot-downs to gain exclusive page access
- Optimistic: use snapshots & diffs before XTM commit
 - No overhead for HW transactions

■ See paper for forward progress guarantees for XTM

XTM-base Requirements

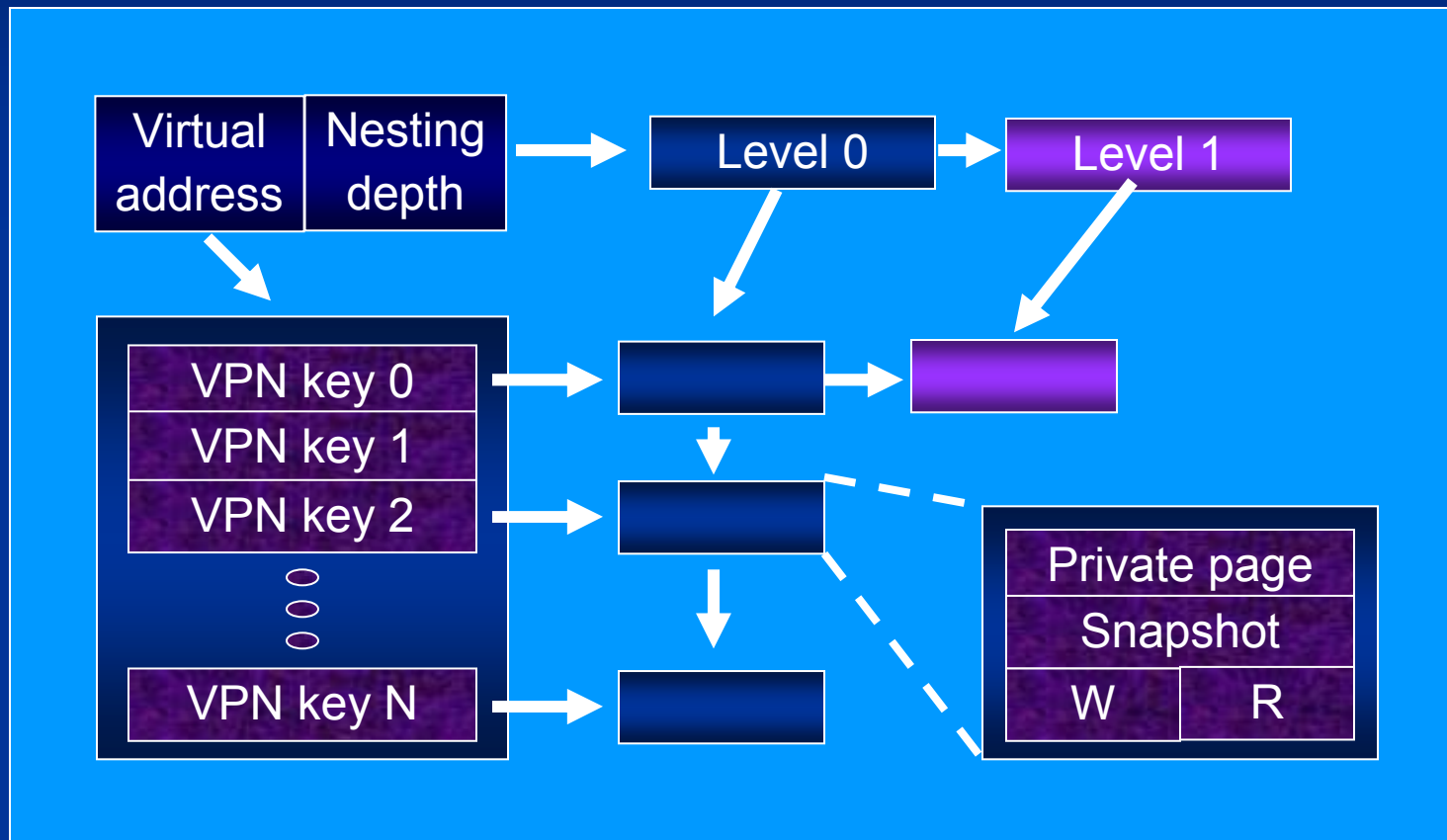
■ Hardware

- Required: overflow exception
- Optional: fast page copy mechanism (DMA, SIMD, ...)

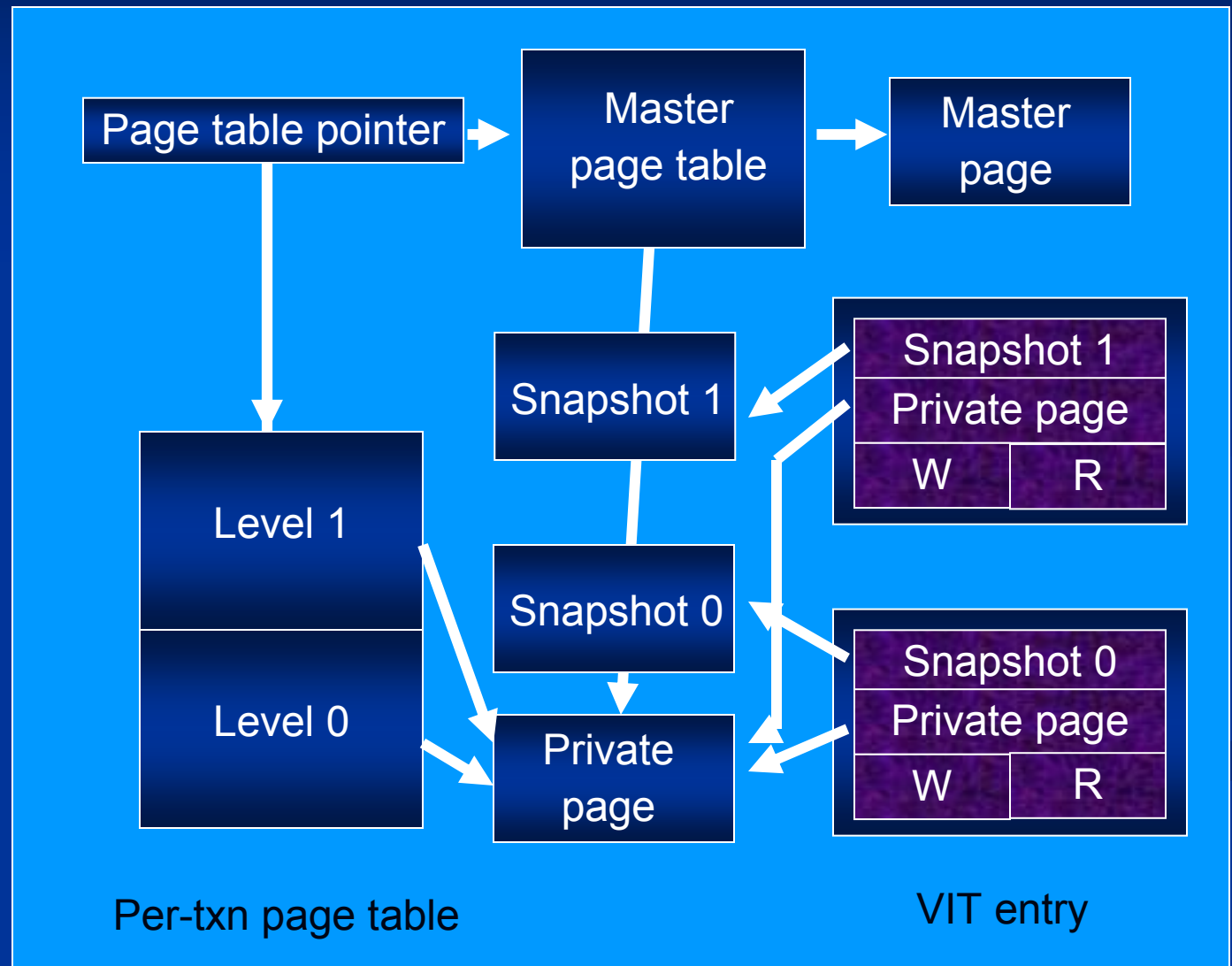
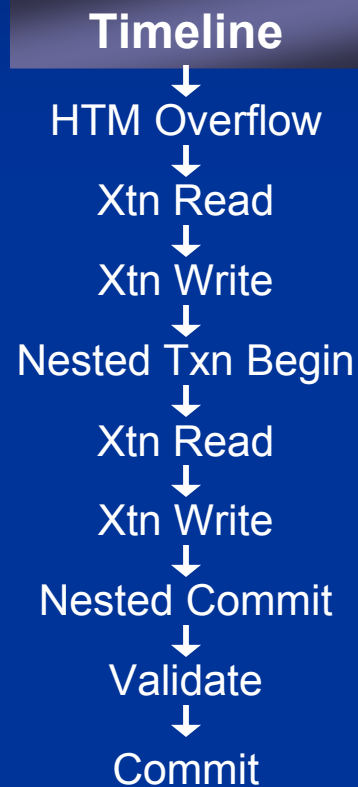
■ Data-structures (software)

- Per-transaction page table
 - Contains only mappings to private pages, not master pages
 - Populated dynamically
- Per-core virtualization information table (VIT)
 - Maintains metadata and the pointers to extra pages
- Data-structures are pre-allocated to reduce overhead

Virtualization Information Table



XTM-base Example



XTM-g: Gradual Overflow

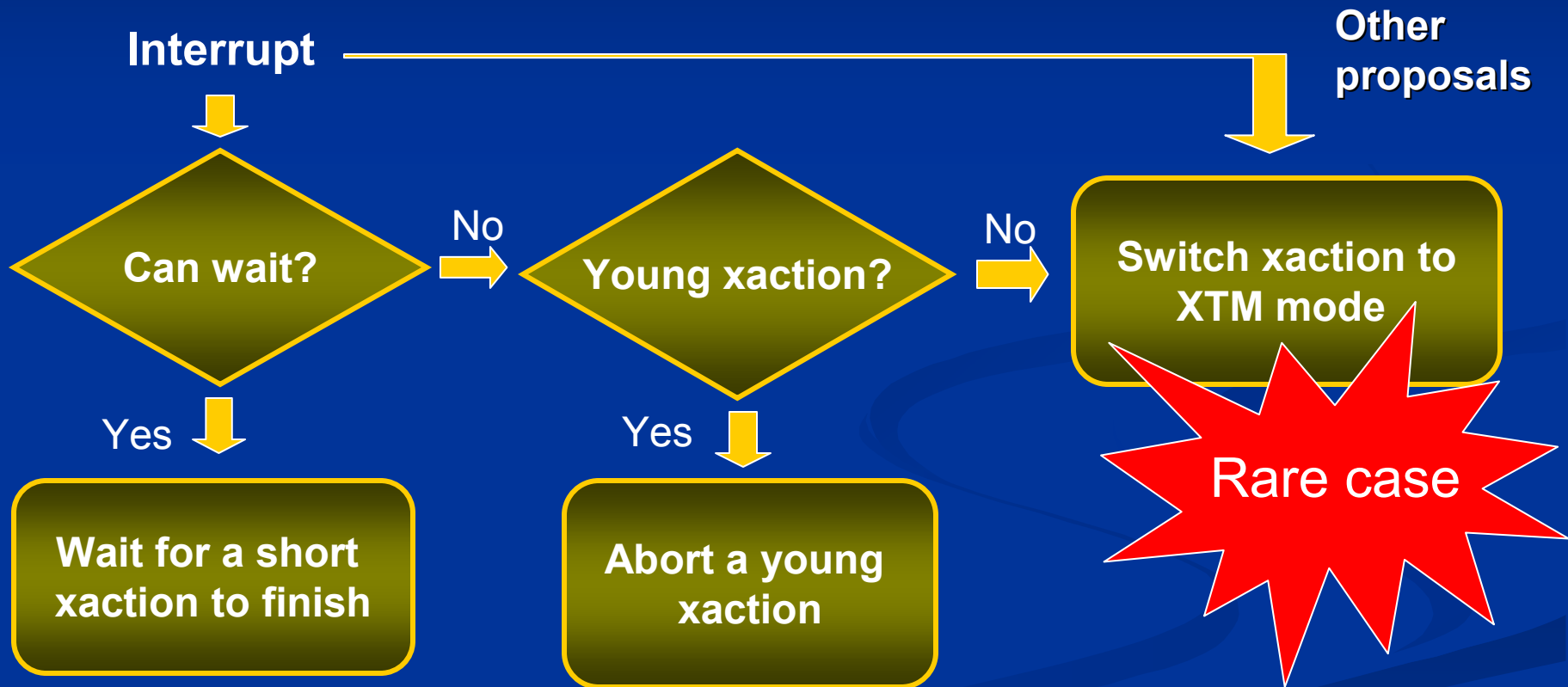
- XTM-base bottleneck: roll-back overhead on overflow
- Gradual overflow
 - On overflow, just flush one or a few pages to XTM
 - A portion of transactional data in private pages, the rest in the cache
- XTM commit
 - First validate both XTM and hardware TM data
 - Then commit the XTM and hardware TM data
 - Requires two-phase commit support [ISCA'06]
- Hardware requirements
 - Overflow bit to remember the pages that have overflowed
 - Per page-table entry, TLB entries, and cache lines

XTM-e: Fine-grain Conflict Detection

- XTM-g bottleneck: false sharing overhead at page level
- Fine-grain conflict detection
 - When flushing a cache line, record fine-grain metadata bits in VIT
 - Per cache line or per even per word
 - Use fine-grain information on validation
 - Validate only portions of each XTM page
- Requirements
 - SW: extra space in VIT entries for fine-grain metadata
 - HW: eviction buffer for metadata bits (performance enhancement)
 - Needed for cache lines that are reloaded and then evicted
 - Avoids SW handler invocation on each subsequent eviction
 - Buffer is flushed periodically

Time virtualization

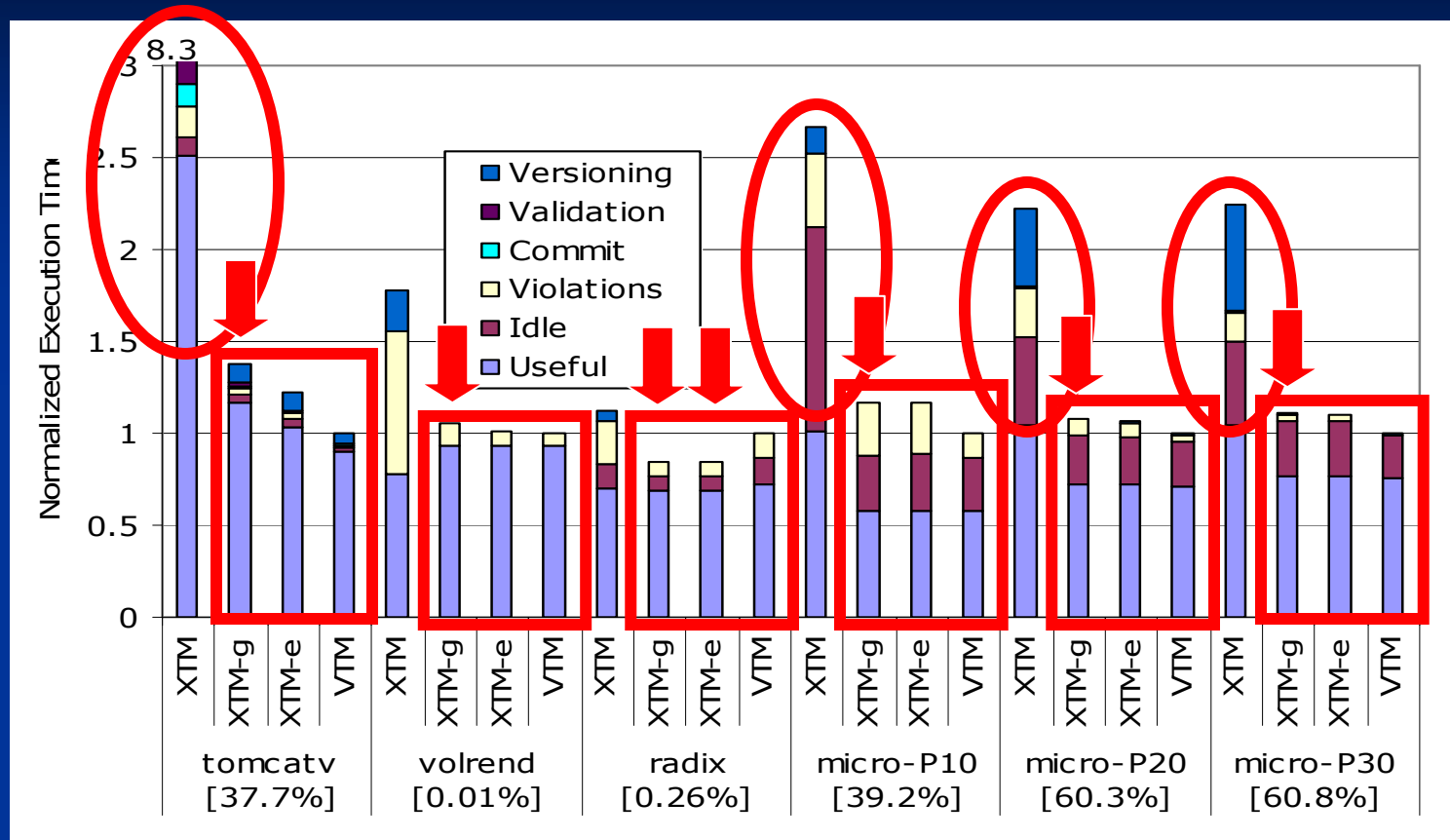
- Interrupt and context-switch procedure



Evaluation

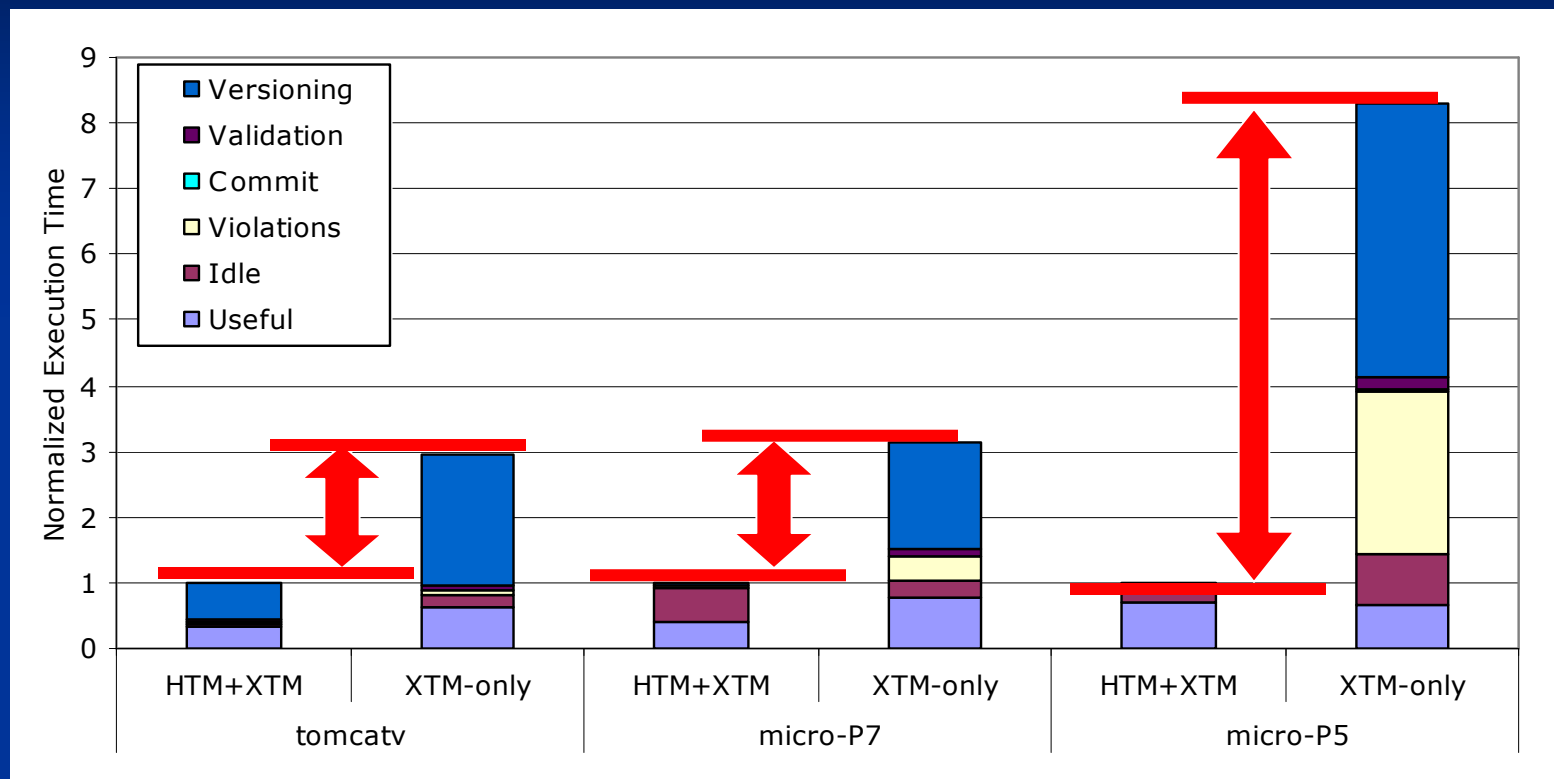
- Execution-driven hardware TM simulator (TCC)
 - XTM series and VTM are compared
 - 32KB cache for transactional buffering
- Applications
 - SPLASH-2, SPEC, and micro-benchmarks
 - **Important:** many applications did not invoke XTM at all
 - XTM introduces no HW cost or overhead for them
- Experiments
 - Overall performance analysis
 - XTM-only transactional memory
 - More results in the paper
 - Memory pressure, sensitivity to cache size, time virtualization evaluation, ...

Performance Analysis



- XTM-base showed 3x to 8x slowdown for applications with frequent overflow
 - It causes no overhead for most applications that don't overflow
- XTM-g presents a good cost/performance tradeoff point
 - 20% faster to 50% slower than VTM

XTM-only Transactional Memory?



- There is a clear performance gap between HTM + XTM and XTM-only
 - It is 3x to 8x slower than hardware TM
- Hardware support is important for transactional memory

Conclusions

- **TM is a promising solution for parallel programming**
- **Hardware TM delivers a good performance**
 - Challenges for HTM : overflows, interrupts, deep nesting, ...
 - TM virtualization is a crucial component for practical HTMs
- **XTM: virtualization through the operating system**
 - Virtualizes TM space, time, and depth
 - Low HW cost and completely transparent to user SW
 - proposes 3 designs of different tradeoff points
 - XTM-base: SW only solution
 - XTM-g: eliminates rollback overhead with Overflow bit
 - XTM-e: eliminates false sharing with more HW support
- **We hope that this work helps build practical HTM systems**

Questions?



Whew~!

Jae Woong Chung
jwchung@stanford.edu

Computer Systems Lab.
Stanford University
<http://tcc.stanford.edu>