ATLAS

Software Development Environment for Hardware Transactional Memory

Sewook Wee

Computer Systems Lab
Stanford University

The Parallel Programming Crisis

- Multi-cores for scalable performance
  - No faster single core any more
- Parallel programming is a must, but still hard
  - Multiple threads access shared memory
  - Correct synchronization is required
- Conventional: lock-based synchronization
  - Coarse-grain locks: serialize system
  - Fine-grain locks: hard to be correct

Alternative: Transactional Memory (TM)

- Memory transactions [Knight'86][Herlihy & Moss'93]
  - An atomic & isolated sequence of memory accesses
  - Inspired by database transactions
- Atomicity (all or nothing)
  - At commit, all memory updates take effect at once
  - On abort, none of the memory updates appear to take effect
- Isolation
  - No other code can observe memory updates before commit
- Serializability
  - Transactions seem to commit in a single serial order

Advantages of TM

- As easy to use as coarse-grain locks
  - Programmer declares the atomic region
  - No explicit declaration or management of locks
- As good performance as fine-grain locks
  - System implements synchronization
  - Optimistic concurrency [Kung'81]
  - Slow down only on true conflicts (R-W or W-W)
  - Fine-grain dependency detection
- No trade-off between performance & correctness

Implementation of TM

- Software TM [Harris'03][Saha'06][Dice'06]
  - Versioning & conflict detection in software
  - No hardware change, flexible
  - Poor performance (up to 8x)
- Hardware TM [Herlihy & Moss'93] [Hammond'04][Moore'06]
  - Modifying data cache hardware
  - High performance
  - Correctness: strong isolation

Software Environment for HTM

- Programming language [Carlstrom'07]
  - Parallel programming interface
- Operating system
  - Provides virtualization, resource management, …
  - Challenges for TM
    - Interaction of active transaction and OS
- Productivity tools
  - Correctness and performance debugging tools
  - Build up on TM features

Contributions

- An operating system for hardware TM
- Productivity tools for parallel programming
- Full-system prototyping & evaluation

Agenda

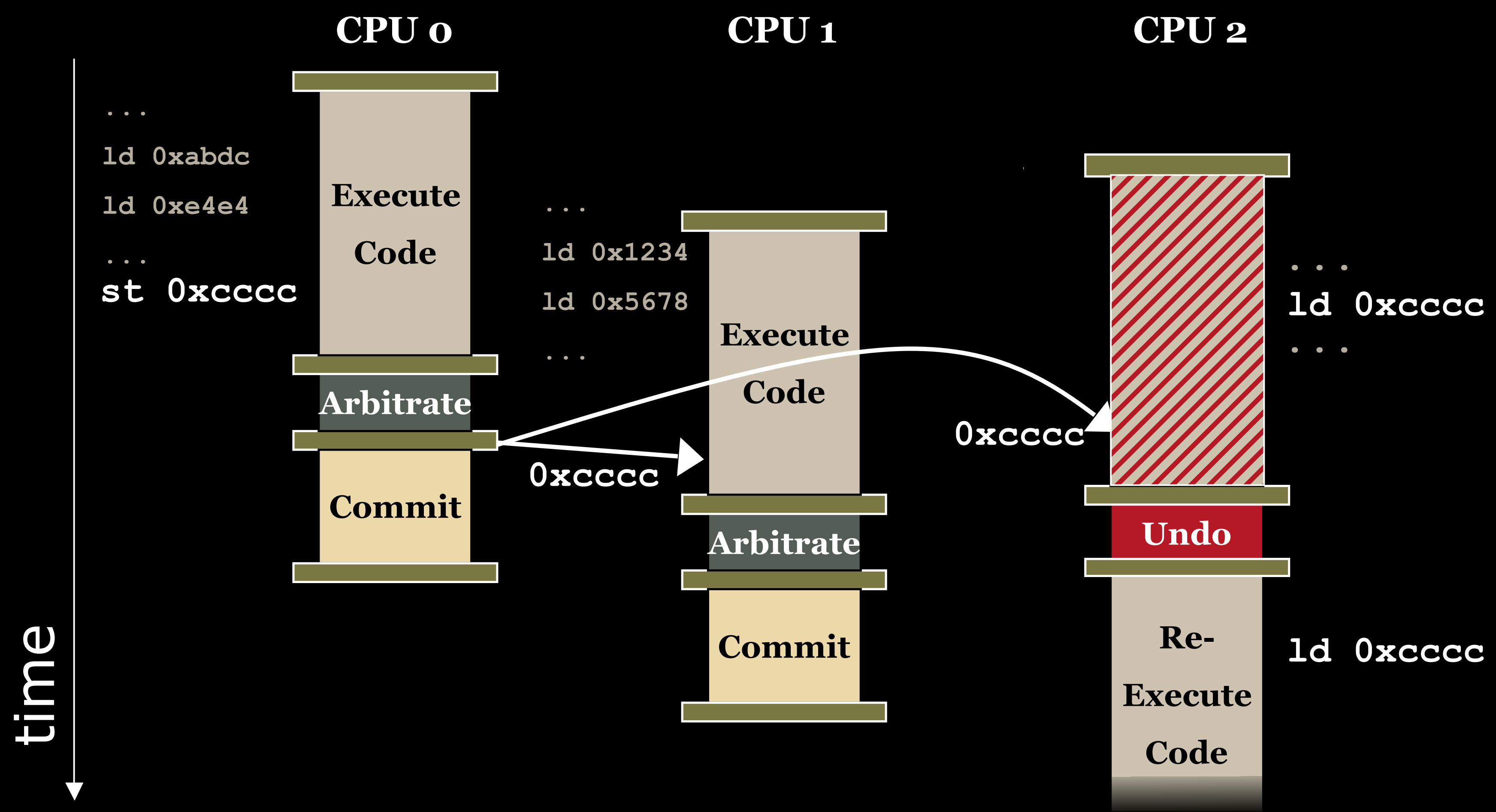
- Motivation
- Background
- Operating System for HTM
- Productivity Tools for Parallel Programming
- Conclusions

TCC: Transactional Coherence/Consistency

- A hardware-assisted TM implementation
  - Avoids overhead of software-only implementation
  - Semantically correct TM implementation

- A system that uses TM for coherence & consistency
  - Use TM to replace MESI coherence
    - Other proposals build TM on top of MESI
  - All transactions, all the time

TCC Execution Model



See [ISCA'04] for details

CMP Architecture for TCC

Transactionally Read Bits:

`ld 0xdeadbeef`

Transactionally Written Bits:

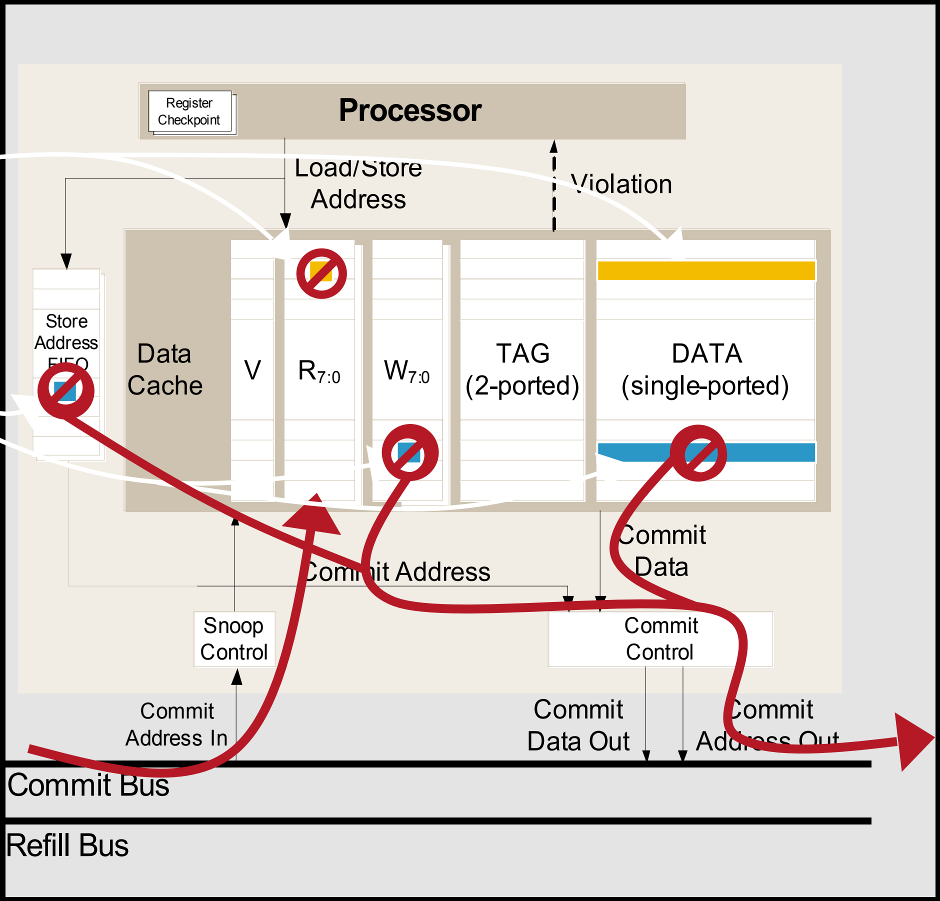
`st 0xcafebabe`

Commit:

Read pointers from Store Address FIFO, flush addresses with W bits set

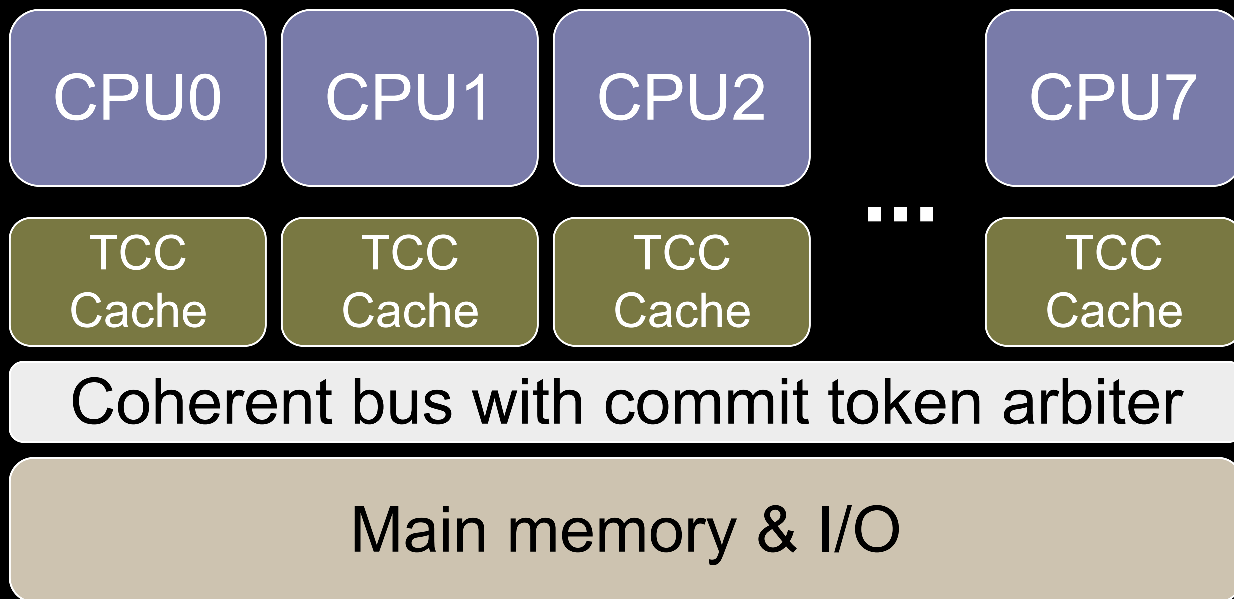
Conflict Detection:

Compare incoming address to R bits



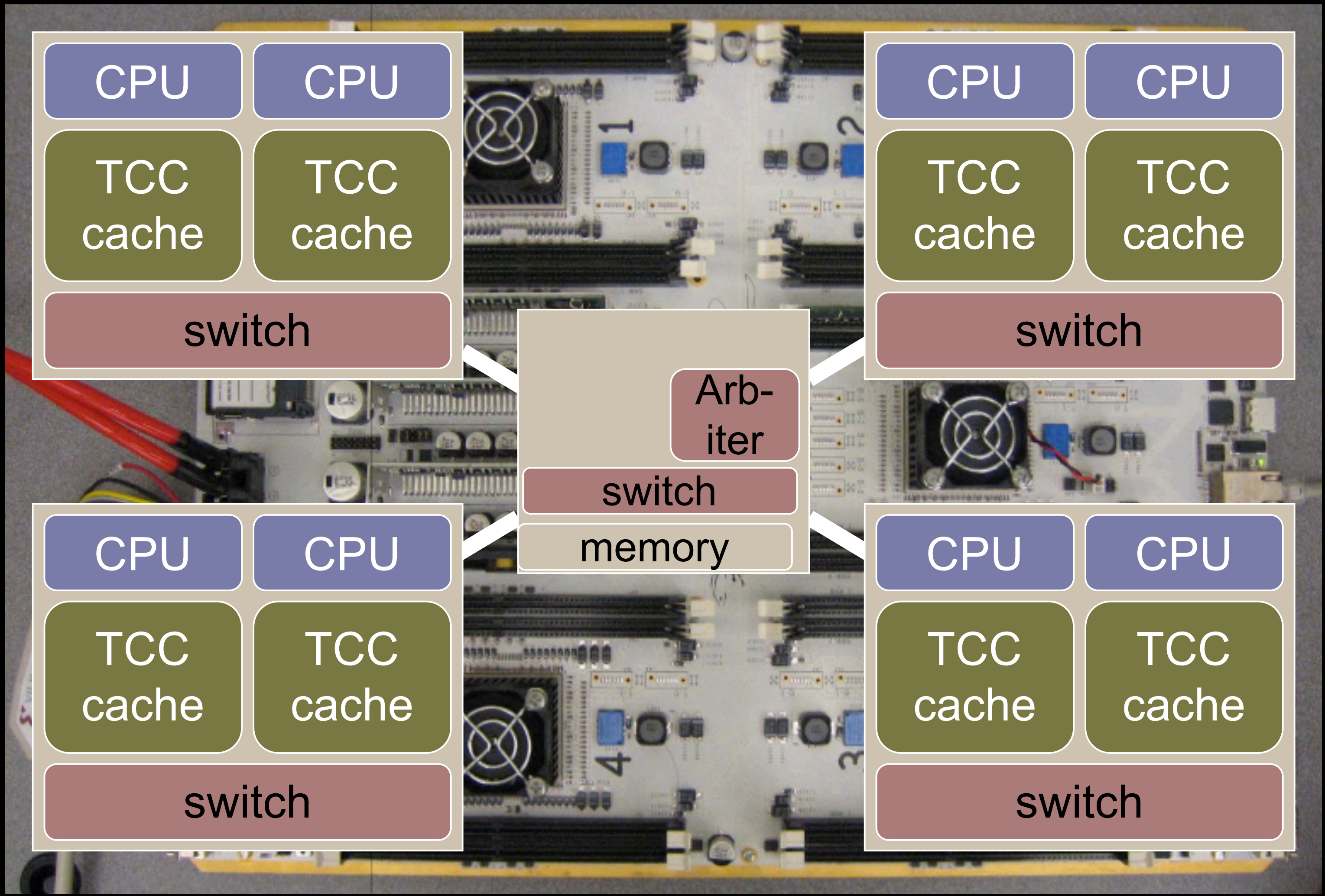
See [PACT'05] for details

ATLAS Prototype Architecture



- Goal
  - Convinces a proof-of-concept of TCC
  - Experiments with software issues

Mapping to BEE2 Board



Agenda

- Motivation
- Background
- Operating System for HTM
- Productivity Tools for Parallel Programming
- Conclusions

Challenges in OS for HTM

What should we do if OS needs to run in the middle of transaction?

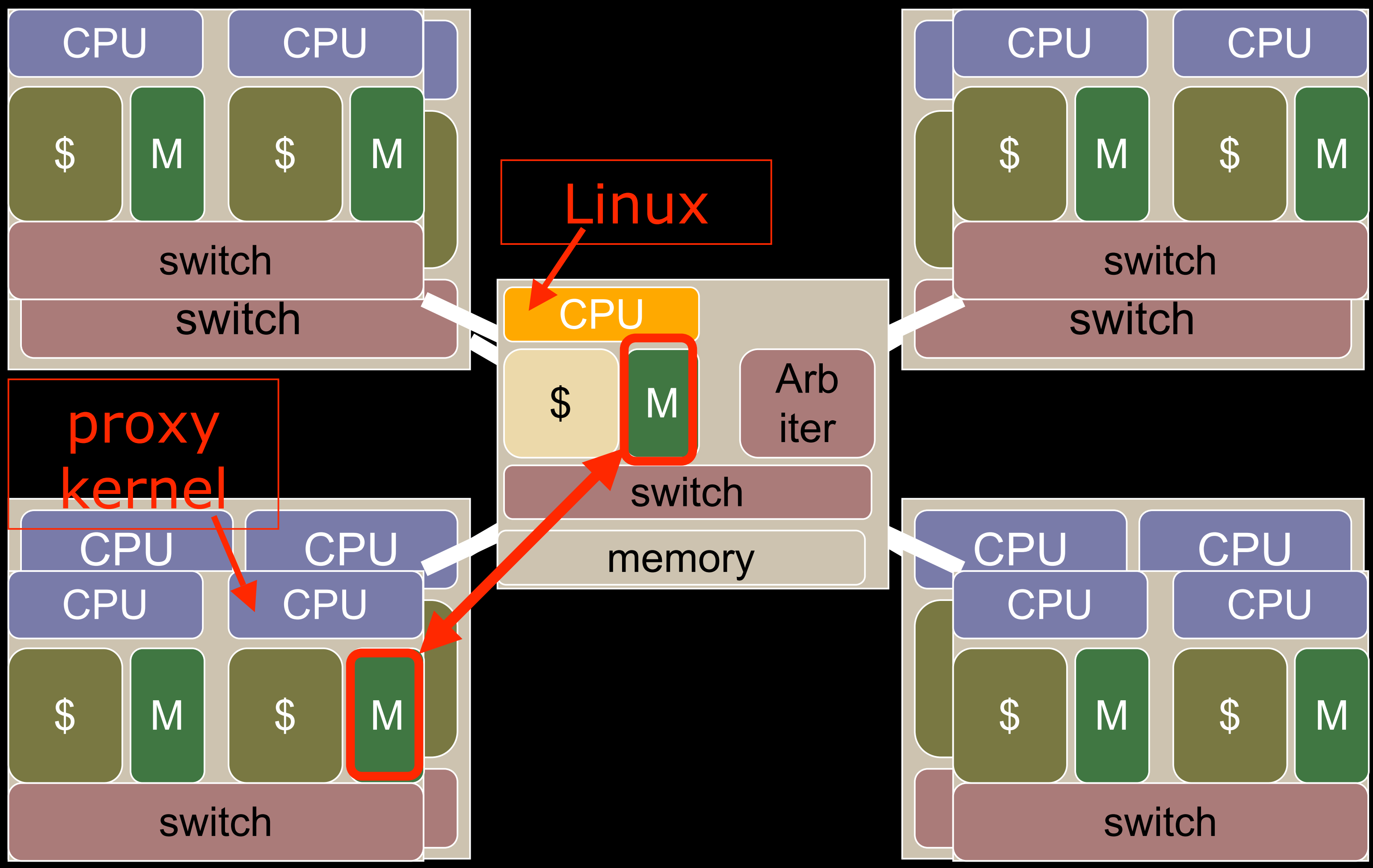
Challenges in OS for HTM

- Loss of isolation at exception
  - Exception info is not visible to OS until commit
  - I.e. faulting address in TLB miss
- Loss of atomicity at exception
  - Some exception services cannot be undone
  - I.e. file I/O
- Performance
  - OS preempts user thread in the middle of transaction
  - I.e. interrupts

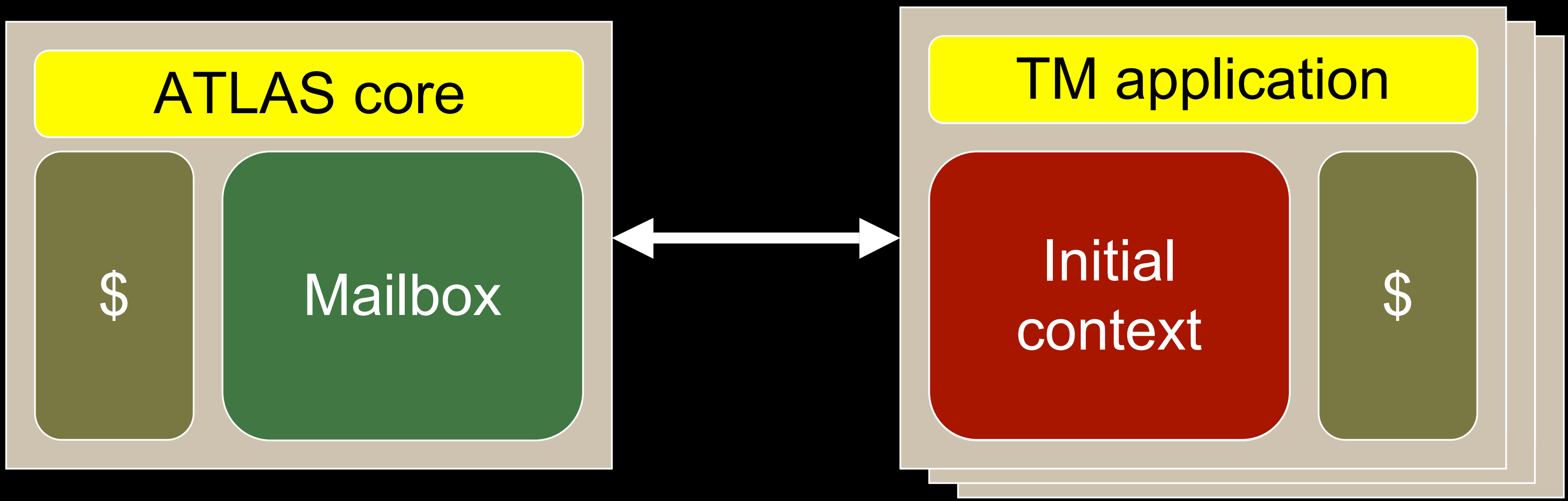
Practical Solutions

- Performance
  - A dedicated CPU for operating system
  - No need to preempt user thread in the middle of transaction
- Loss of isolation at exception
  - Mailbox: separate communication layer between application and OS
- Loss of atomicity at exception
  - Serialize system for irrevocable exceptions

Architecture Update

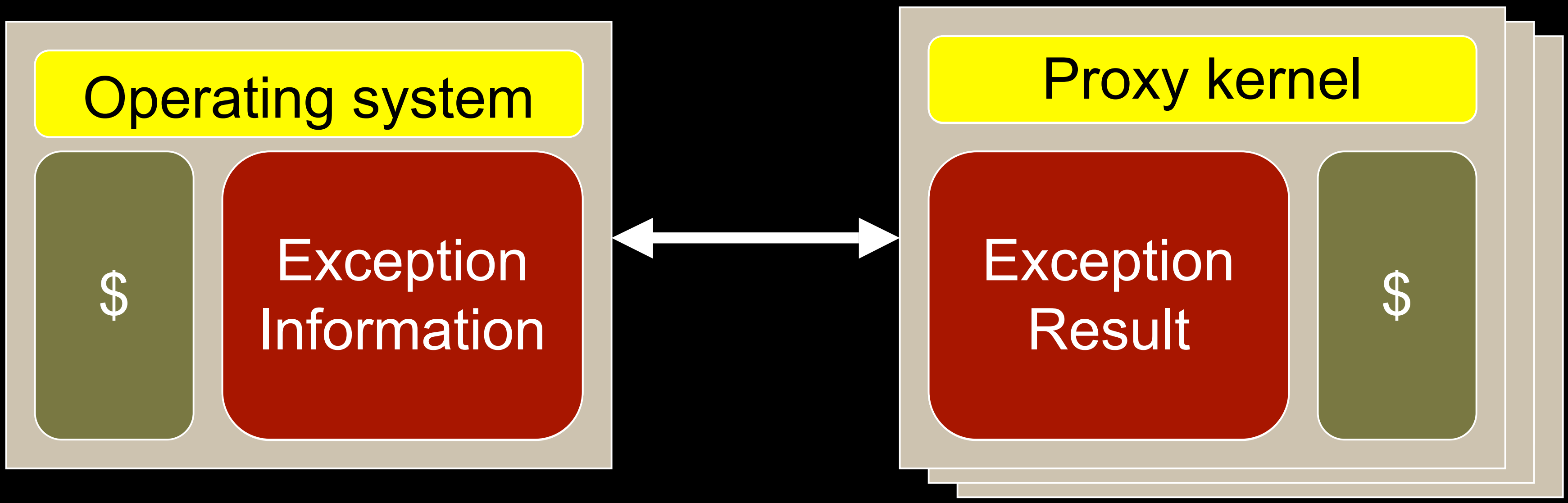


Execution overview (1) - Start of an application



- ATLAS core
  - A user-level program runs on OS CPU
  - Same address space as TM application
  - Start application & listen to requests from apps
- Initial context
  - Registers, PC, PID, …

Execution overview (2) - Exception

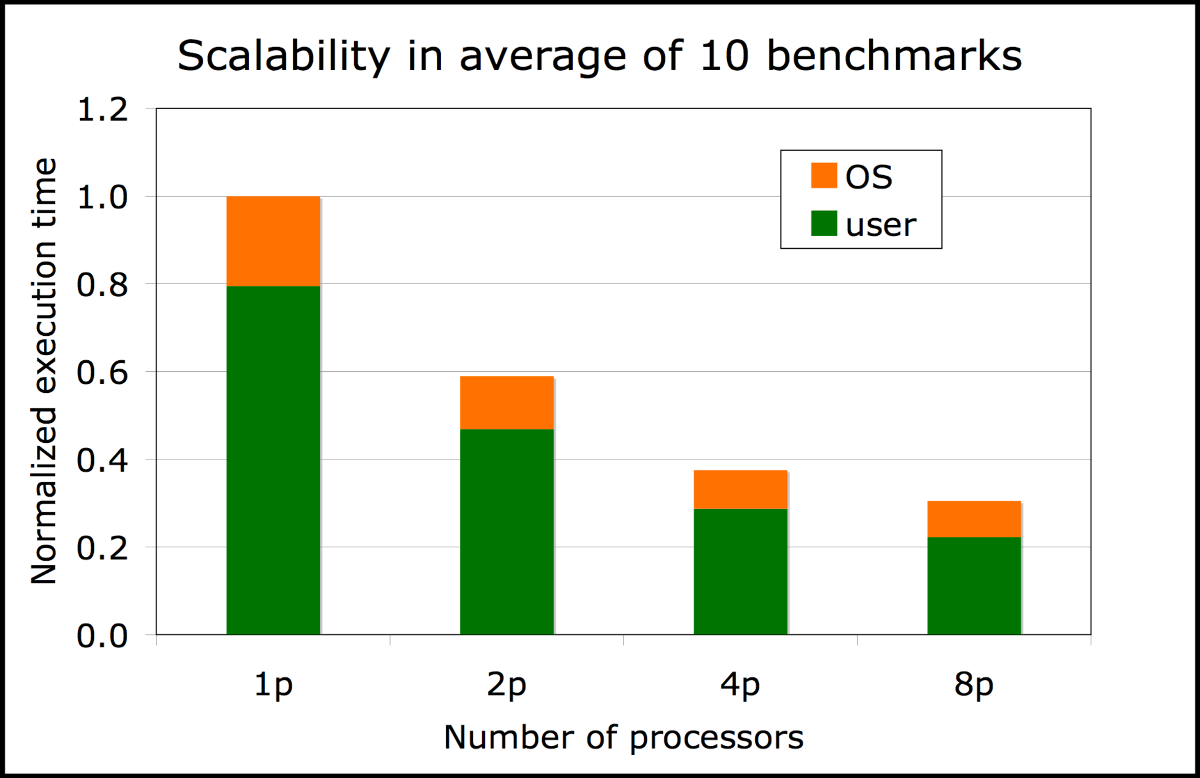


- Proxy kernel forward the exception information to OS CPU
  - Fault address for TLB misses
  - Syscall number and arguments for syscalls
- OS CPU services the request and returns the result
  - TLB mapping for TLB misses
  - Return value and error code for syscalls

Operating System Statistics

- Strategy: Localize modifications
  - Minimize the work needed to track main stream kernel development
- Linux kernel (version 2.4.30)
  - Device driver that provides user-level access to privilege-level information
  - ~1000 lines (C, ASM)
- Proxy kernel
  - Runs on application CPU
  - ~1000 lines (C, ASM)
- A full workstation for programmer's perspective

System Performance



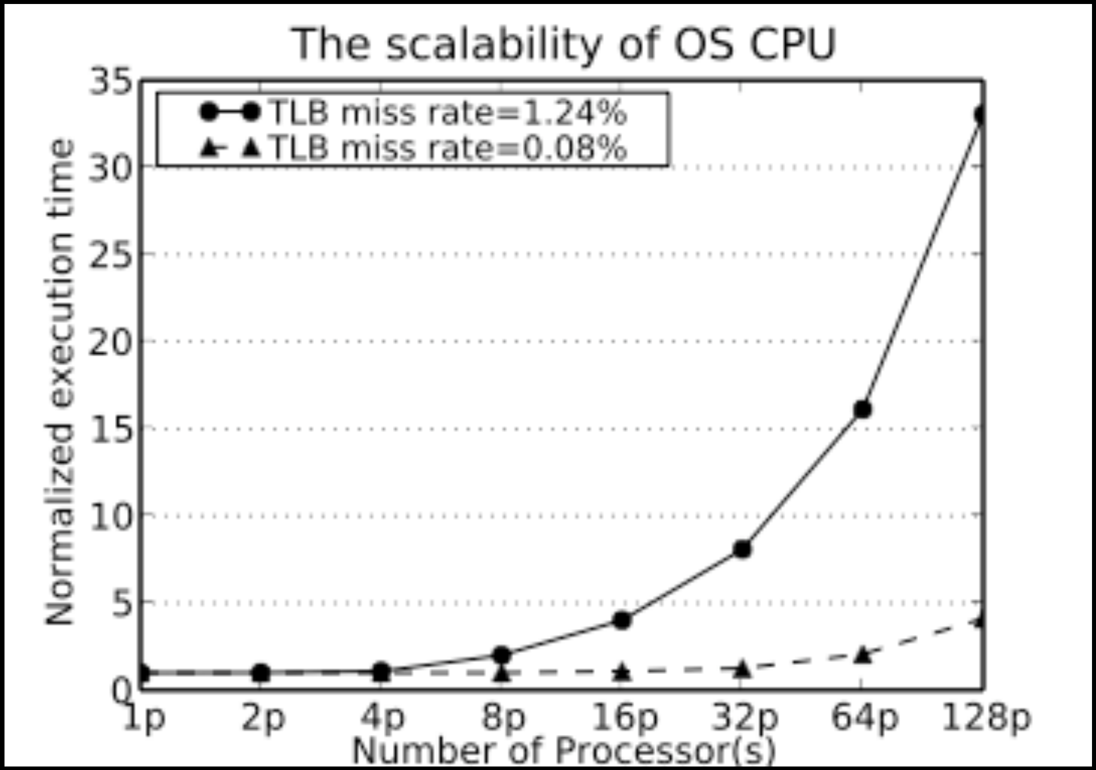
- Total execution time scales
- OS time scales, too

Scalability of OS CPU

- Single CPU for operating system
  - Eventually, it will become a bottleneck as system scales
  - Multiple CPUs for OS will need to run SMP OS

- Micro-benchmark experiment
  - Simultaneous TLB miss requests
  - Controlled injection ratio
  - Looking for the number of application CPUs that saturates OS CPU

Experiment results



- Average TLB miss rate = 1.24%
  - Start to congest from 8 CPUs
- With victim TLB (Average TLB miss rate = 0.08%)
  - Start to congest from 64 CPUs

Agenda

- Motivation
- Background
- Operating System for HTM
- Productivity Tools for Parallel Programming
- Conclusions

Challenges in Productivity Tools for Parallel Programming

- Correctness
  - Nondeterministic behavior
    - Related to a thread interleaving
  - Need to track an entire interleaving
    - Very expensive in time/space
- Performance
  - Detailed information of the performance bottleneck events
  - Light-weight monitoring
    - Do not disturb the interleaving

Opportunities with HTM

- TM already tracks all reads/writes
  - Cheaper to record memory access interleaving
- TM allows non-intrusive logging
  - Software instrumentation in TM system
  - Not in user's application
- All transactions, all the time
  - Everything in transactional granularity

Tool 1: ReplayT

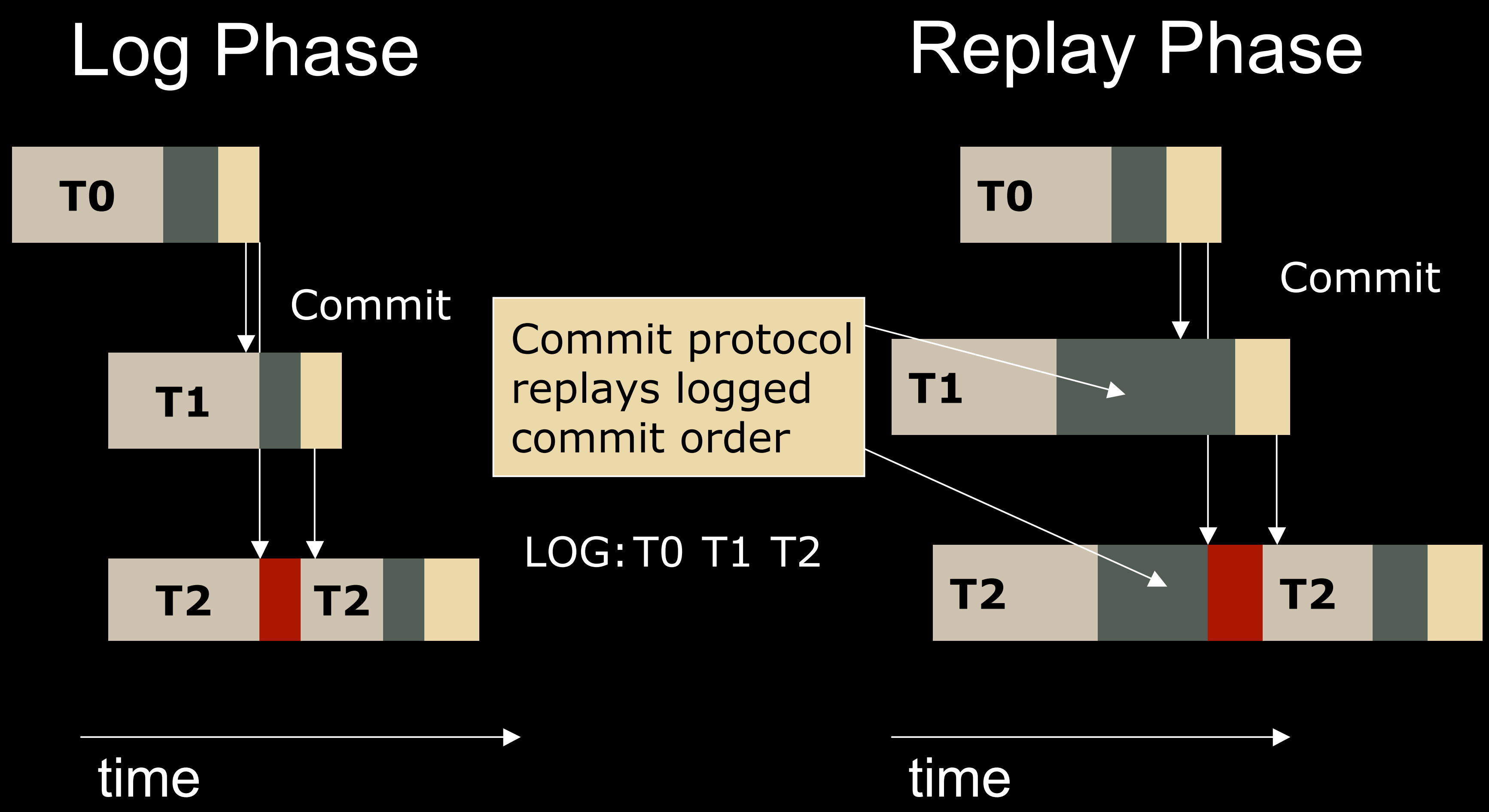
Deterministic Replay

Deterministic Replay

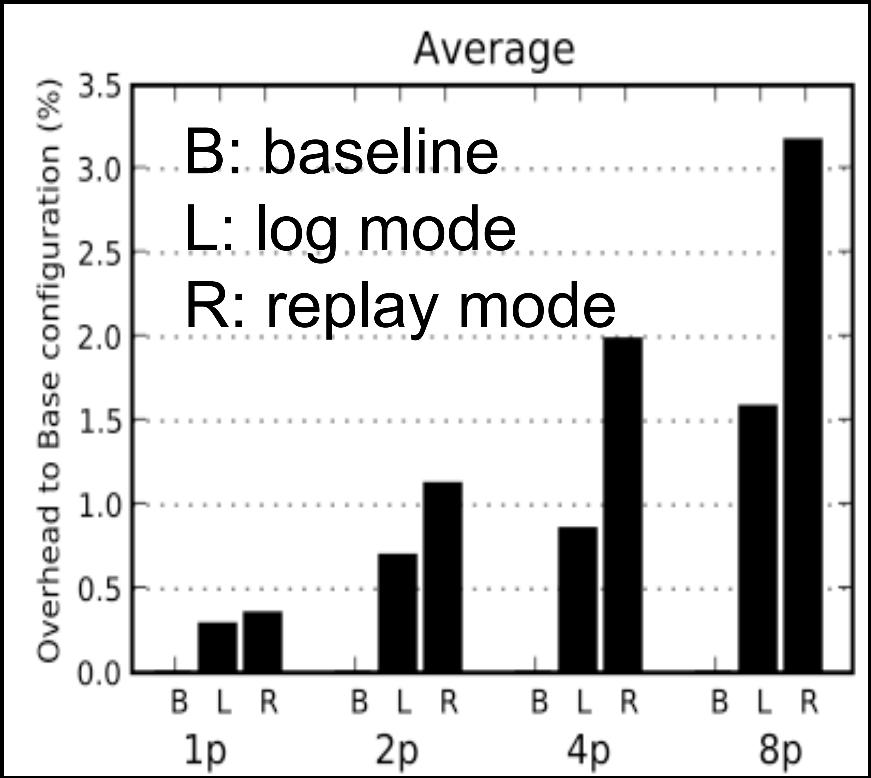
- Challenges in recording an interleaving
  - Record every single memory access
  - Intrusive
  - Large footprint

- ReplayT's approach
  - Record only a transaction interleaving
  - Minimally overhead: 1 event per transaction
  - Footprint: 1 byte per transaction (thread ID)

ReplayT Runtime



Runtime Overhead



- Average on 10 benchmarks
  - 7 STAMP,
    3 SPLASH/SPLASH2
- Less than 1.6% overhead for logging
- More overhead in replay mode
  - longer arbitration time
- 1B per 7119 insts.

➔ Minimal time & space overhead

Tool 2. AVIO-TM

Atomicity Violation Detection

Atomicity Violation

- Problem: programmer breaks an atomic task into two transactions

```
ATMDeposit:
 atomic {
   t = Balance
 }
 atomic {
  t = Balance
  Balance = t + $100
 }
```

```
 atomic {
  Balance = t + $100
 }
```

←

```
directDeposit:
 atomic {
  t = Balance
  Balance = t + $1,000
 }
```

Atomicity Violation Detection

- AVIO [Lu'06]
  - Atomic region = No unserializable interleavings
  - Extracts a set of atomic region from correct runs
  - Detects unserializable interleavings in buggy runs

- Challenges of AVIO
  - Need to record all loads/stores in global order
    - Slow (28x)
    - Intrusive - software instrumentation
    - Storage overhead
  - Slow analysis
    - Due to the large volume of data

My Approach: AVIO-TM

- Data collection in deterministic rerun
  - Captures original interleavings
- Data collection at transaction granularity
  - Eliminate repeated loggings for same address (10x)
  - Lower storage overhead
- Data analysis in transaction granularity
  - Less possible interleavings → faster extraction
  - Less data → faster analysis
  - More accurate with complementary detection tools

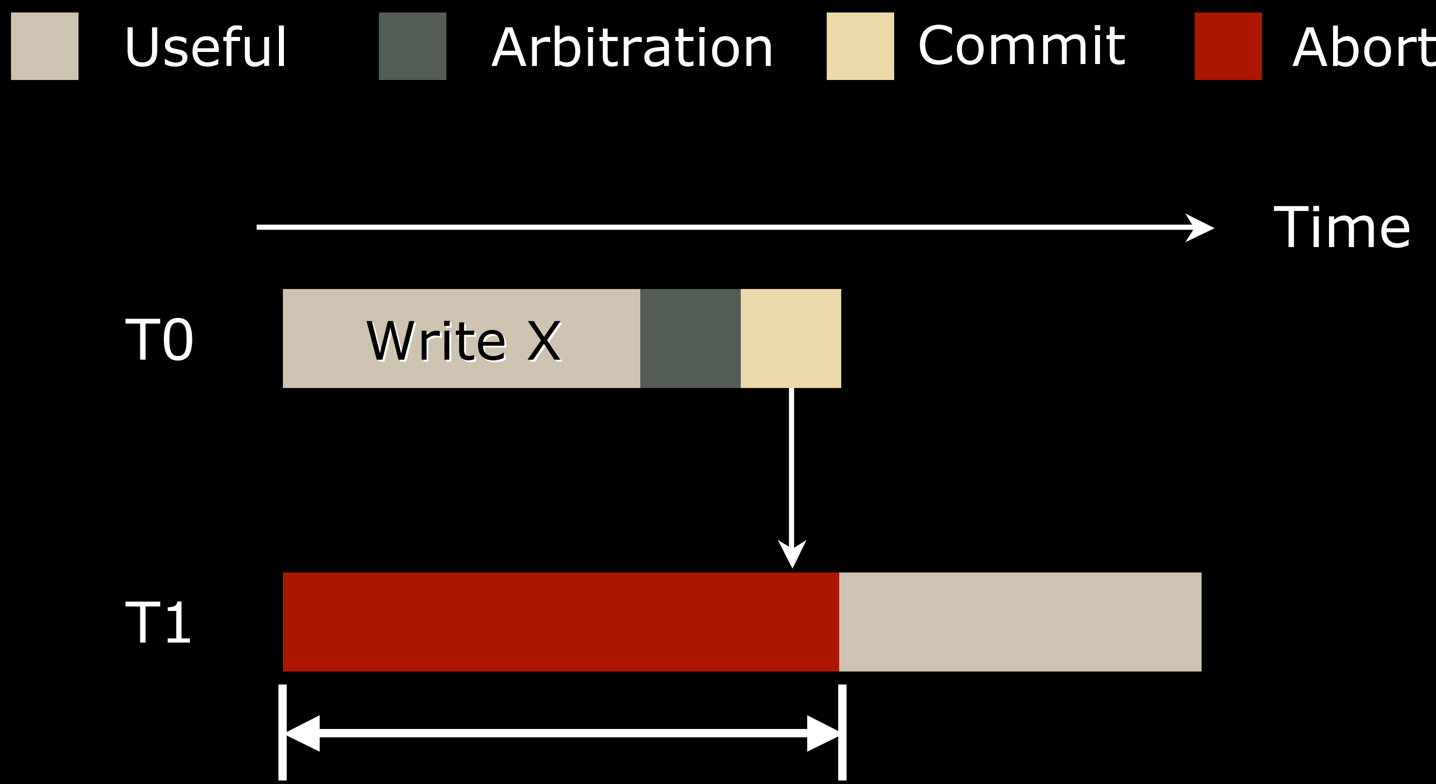
Tool 3. TAPE

Performance Bottleneck Monitor

TM Performance Bottlenecks

- Dependency conflicts
  - Aborted transactions waste useful cycles
- Buffer overflows
  - Speculative states may not fit into cache
  - Serialization
- Workload imbalance
- Transaction API overhead

Dependency Conflicts

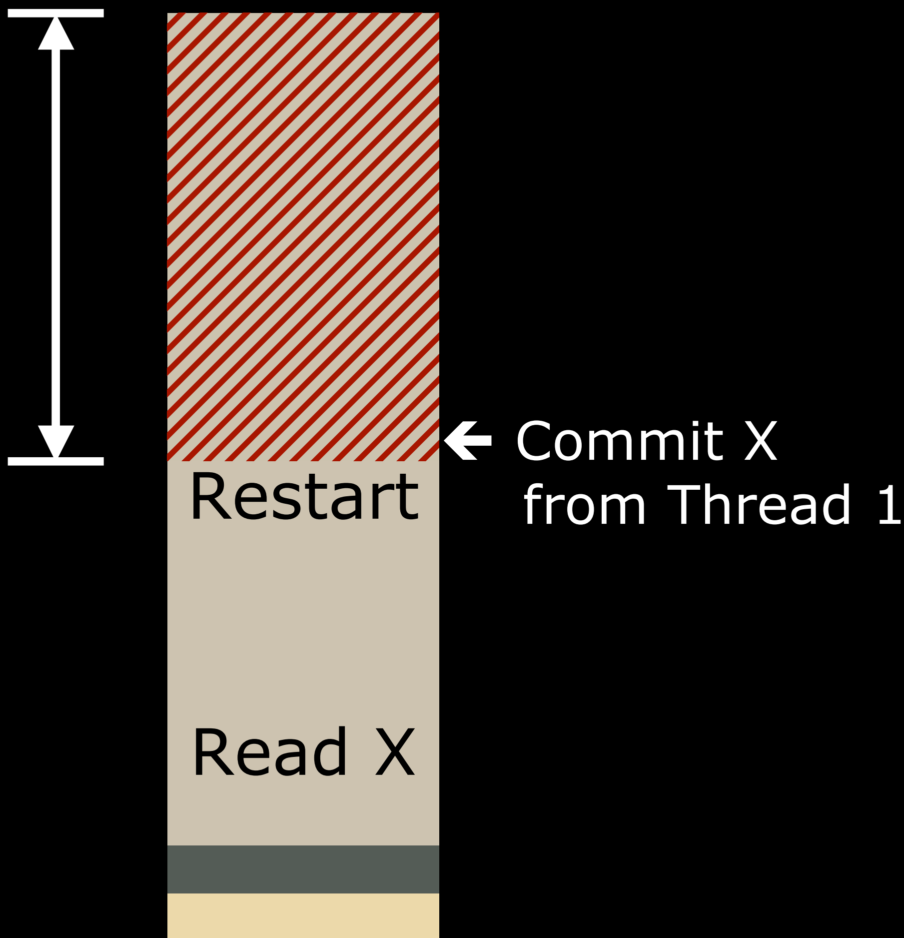


Useful cycles are wasted in T1

TAPE on ATLAS

- TAPE [Chafi, ICS2005]
  - Light weight runtime monitor for performance bottlenecks
- Hardware
  - Tracks information of performance bottleneck events
- Software
  - Collects information from hardware for events
  - Manages them through out the execution

TAPE Conflict



Per Transaction

- Object: X
- Writing Thread: 1
- Wasted cycles: 82,402
- Read PC: 0x100037FC

Per Thread

- Read PC: 0x100037FC
- …
- Occurrence: 4

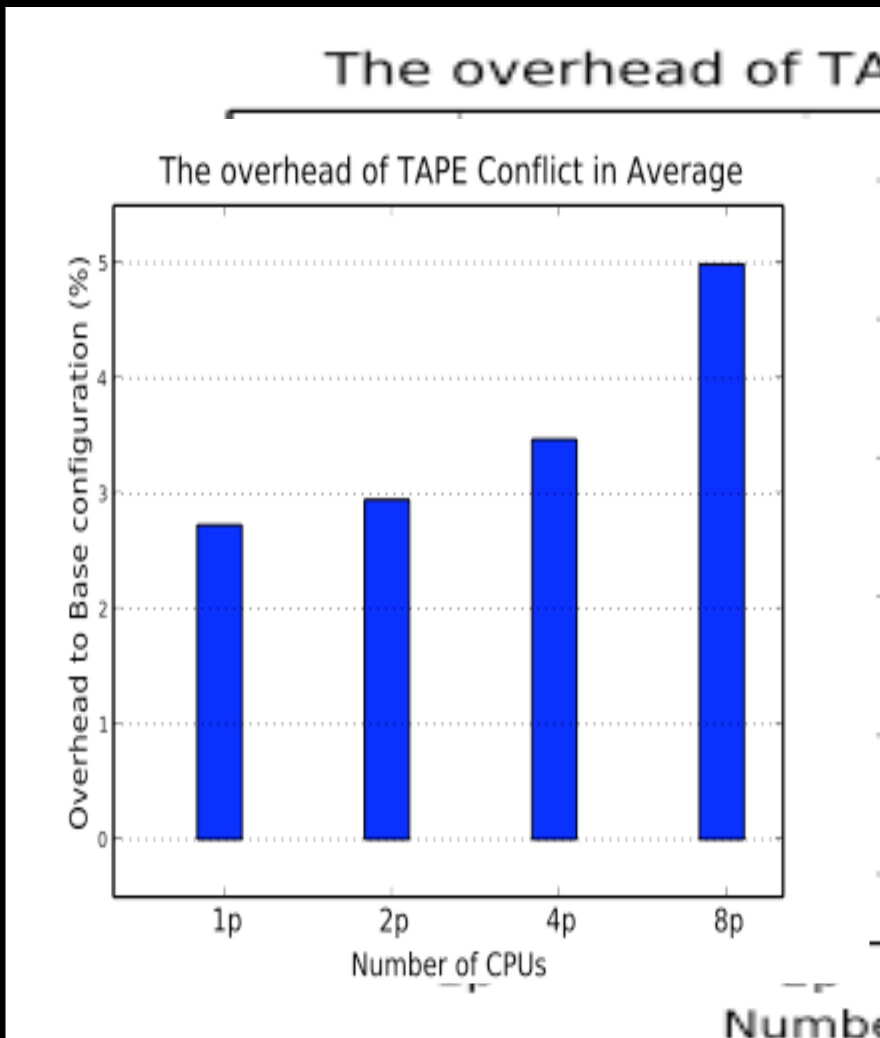
TAPE Conflict Report

Read_PC	Object_Addr	Occurence	Loss	Write_Proc	Read in source line
10001390	100830e0	30	6446858	1	..//vacation/manager.c:134
10001500	100830e0	32	1265341	3	..//vacation/manager.c:134
10001448	100830e0	29	766816	4	..//vacation/manager.c:134
10005f4c	304492e4	3	750669	6	..//lib/rbtree.c:105

- Now, programmers know,
  - Where the conflicts are
  - What the conflicting objects are
  - Who the conflicting threads are
  - How expensive the conflicts are

➔ Productive performance tuning!

Runtime Overhead



- Base overhead
  - 2.7% for 1p
- Overhead from real conflicts
  - More CPU configuration has higher chance of conflicts
  - Max. 5% in total

Conclusion

- An operating system for hardware TM
  - A dedicated CPU for the operating system
  - Proxy kernel on application CPU
  - Separate communication channel between them
- Productivity tools for parallel programming
  - ReplayT: Deterministic replay
  - AVIO-TM: Atomicity violation detection
  - TAPE: Runtime performance bottleneck monitor
- Full-system prototyping & evaluation
  - Convincing proof-of-concept

RAMP Tutorial

- ISCA 2006 and ASPLOS 2008
- Audience of >60 people (academia & industry)
  - Including faculties from Berkeley, MIT, and UIUC
- Parallelized, tuned, and debugged apps with ATLAS
  - From speedup of 1 to ideal speedup in a few minutes
  - Hands-on experience with real system

"most successful hands-on tutorial in last several decades"
- Chuck Thacker (Microsoft Research)

Acknowledgements

- My wife So Jung and our baby (coming soon)
- My parents who have supported me for last 30 years
- My advisors: Christos Kozyrakis and Kunle Olukotun
- My committee: Boris Murmann and Fouad A. Tobagi
- Njuguna Njoroge, Jared Casper, Jiwon Seo, Chi Cao Minh, and all other TCC group members
- RAMP community and BEE2 developers
- Shan Lu from UIUC
- Samsung Scholarship
- All of my friends at Stanford & my Church