Fast Memory Snapshot for Concurrent Programming without Synchronization

Jaewoong Chung

Advanced Development and Research Lab (RADL)

AMD

Woongki Baek, Christos Kozyrakis

Computer System Lab (CSL)

Stanford

Fast Memory Snapshot

- Era of multi-core processors
  - Not easy to develop concurrent software
- Memory Snapshot
  - Not a new idea : atomic multi-word read
  - But, not used for performance-critical applications
- Hardware-assisted Memory Snapshot
  - Leverage hardware transactional memory resources

Example : Memory Profiling (1)

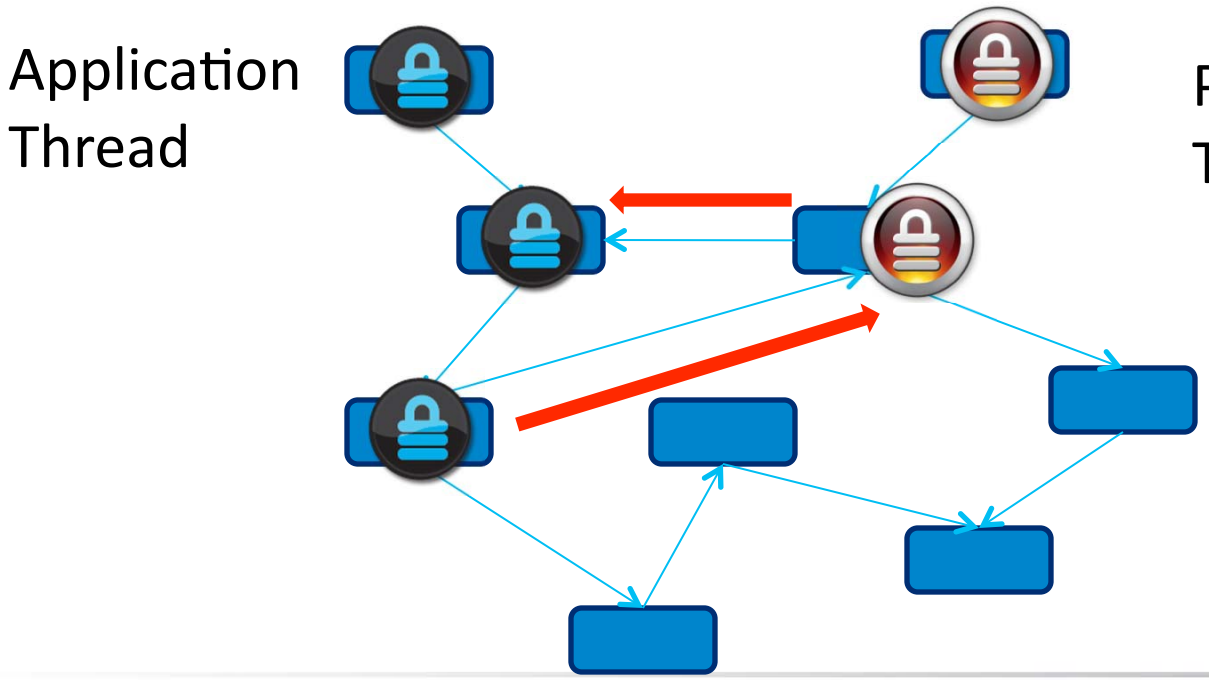
- Global Lock
  - Performance degradation

Application Thread

Profiling Thread

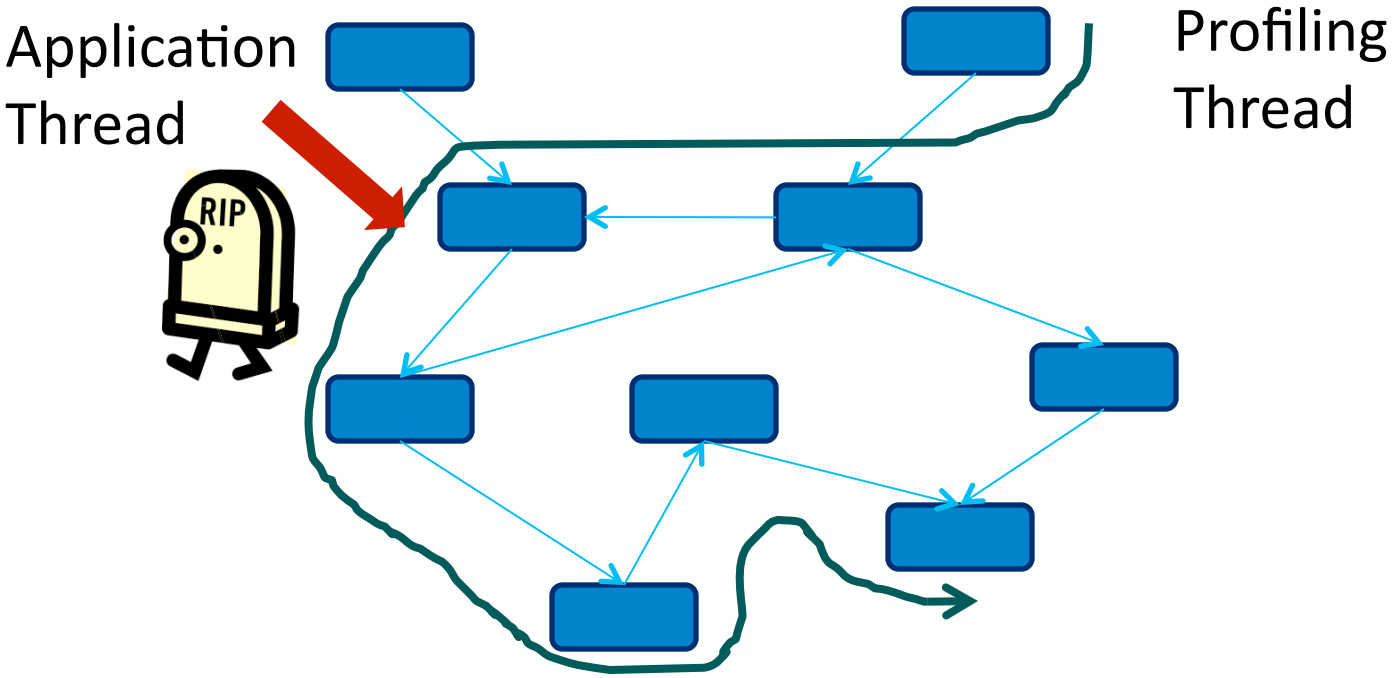
Example : Memory Profiling (2)

- Fine-grain Lock
  - Potential deadlock



Example : Memory Profiling (3)

- Transaction
  - Frequent rollback



Example : Memory Profiling (4)

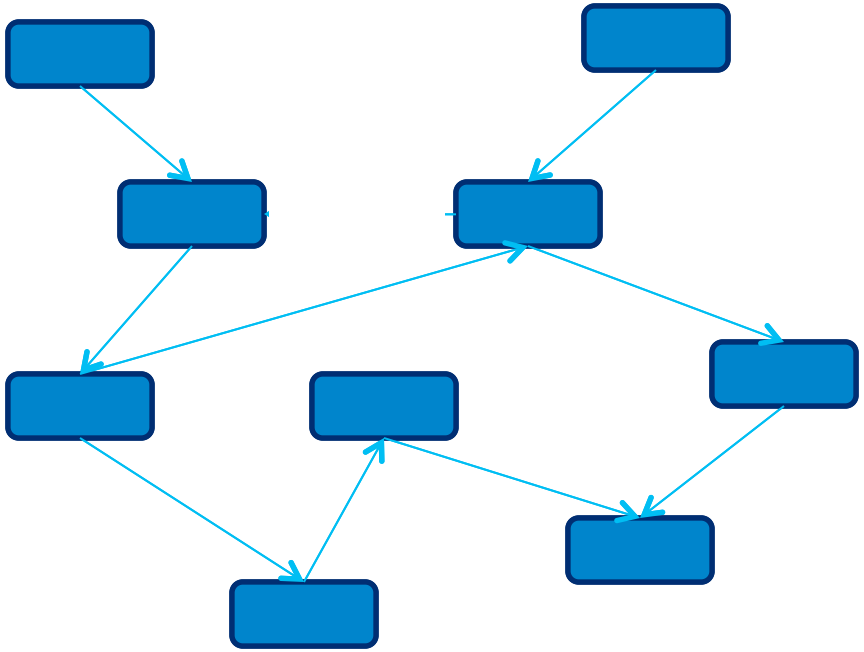
- Tri-coloring
  - White, grey, and black
  - Specific to graph traversal
  - Not general enough for atomic multiword read

- Virtual memory protection
  - Page fault, false sharing

Example : Memory Profiling (5)

- Snapshot

Application Thread



Profiling Thread

More Formally …

- Snapshot
  - Given M memory elements and P processors,
  - Scan : build a snapshot of M elements
  - Update : update each element

- Software Implementations
  - Leverages single-word atomic instructions
  - Typically, O(M) scan time and O(MP) update time
  - Slow for performance-critical applications

Our Proposal

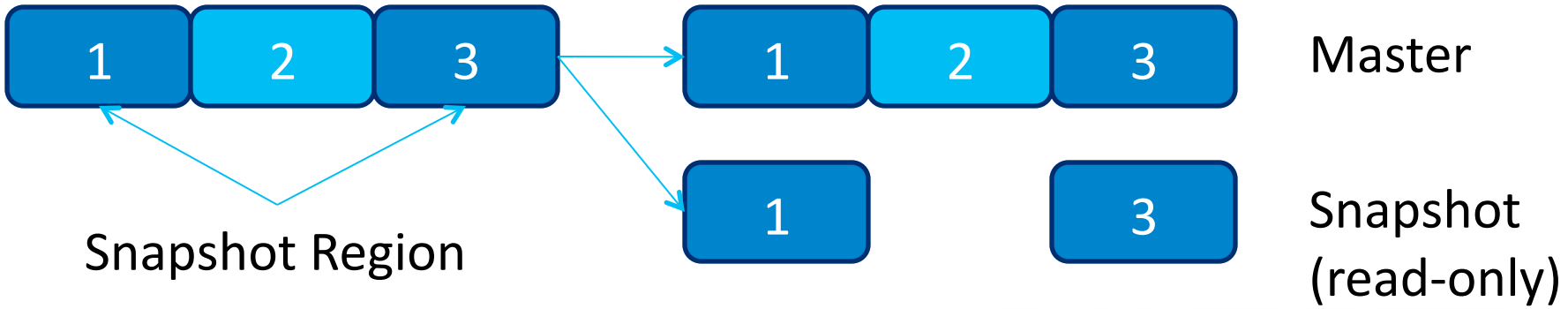
- MShot
  - Hardware-assisted snapshot
  - O(P) scan time, O(1) update time

- Use Cases
  - Fast Checkpoint : Fault tolerance, Guest OS migration
  - Concurrent Garbage Collection and Profiling
  - In-memory Database
  - More use cases in the paper

MShot Images

- Master (up-to-date) Image

- Snapshot Image
  - Multiple disjoint snapshot regions + Registers
  - Read-only



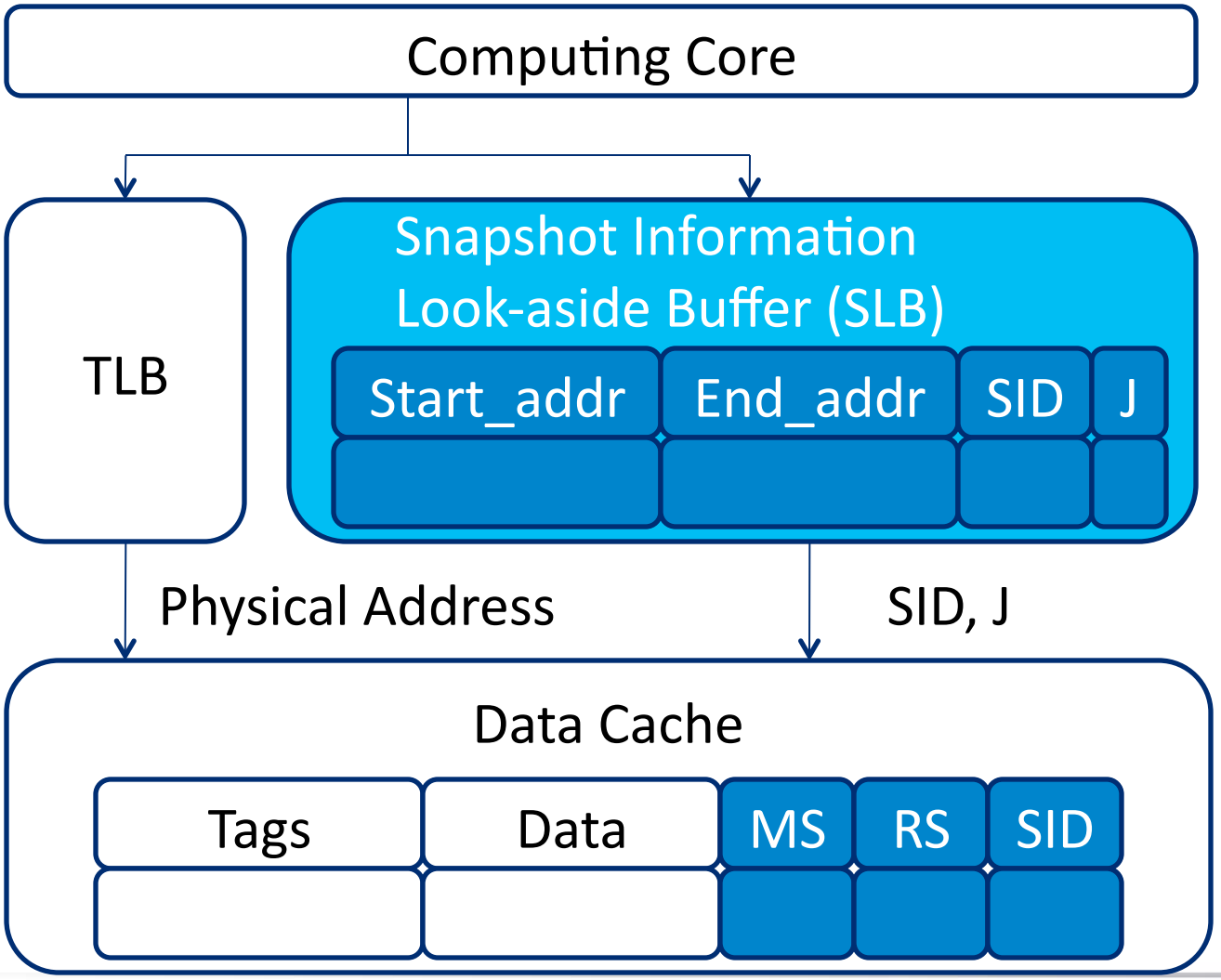
MShot API

- Take_snapshot (snapshot regions)
  - Creates a snapshot on the regions and returns snapshot ID (SID)
- Join_snapshot (SID)
  - Become *Snapshot User*
  - Load instructions start to read from snapshot images
- Leave_snapshot (SID)
  - Load instructions start to read from master images
- Destroy_snapshot (SID)

MShot components

- Software
  - Snapshot Information Table (SIT)
    - Entry per snapshot
    - SID, snapshot regions, join_list
- Hardware
  - Snapshot Information Look-aside Buffer (SLB)
    - To speed up accesses to SIT
  - MShot metadata bits per cache line
    - To distinguish snapshot data from master data

MShot Hardware Components

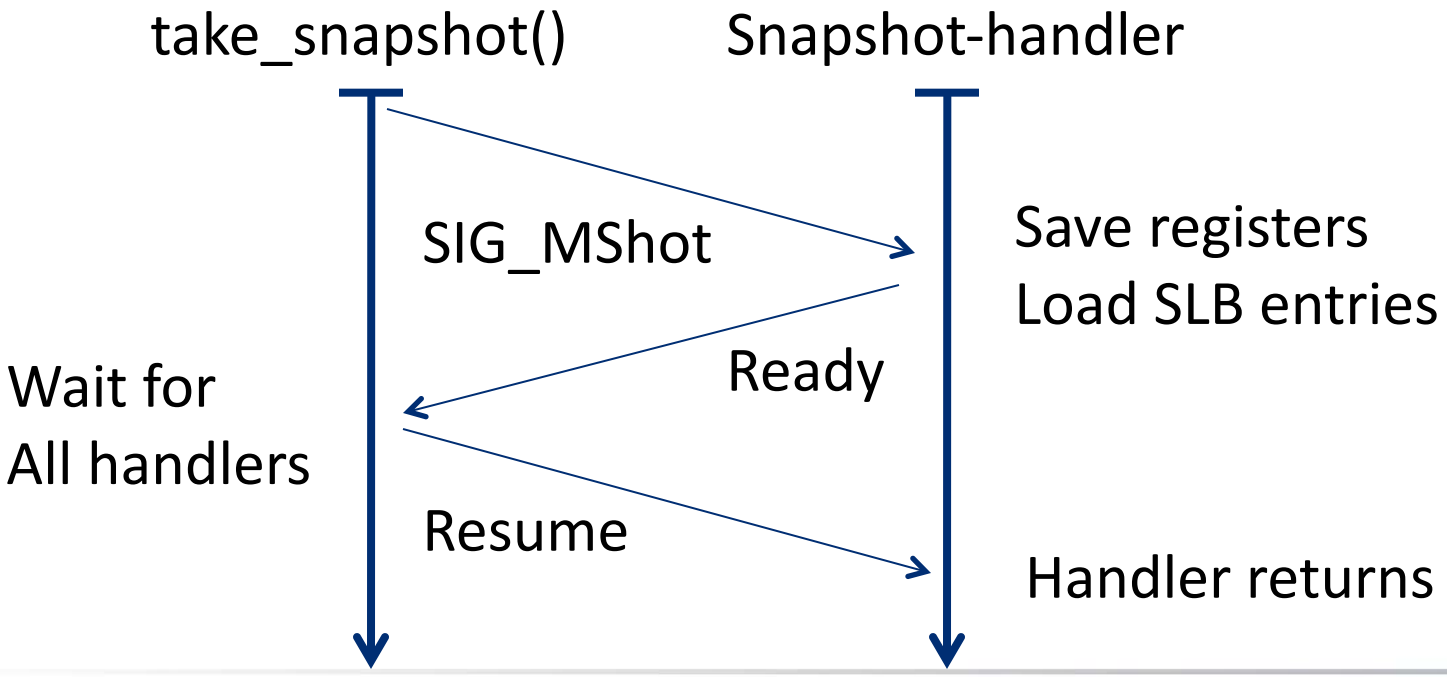


MShot Metadata Bits

- MS (Modified after Snapshot)
  - Set for master data
- RS (Read after Snapshot)
  - Set for snapshot data
- Neither bits are set if master data == snapshot data

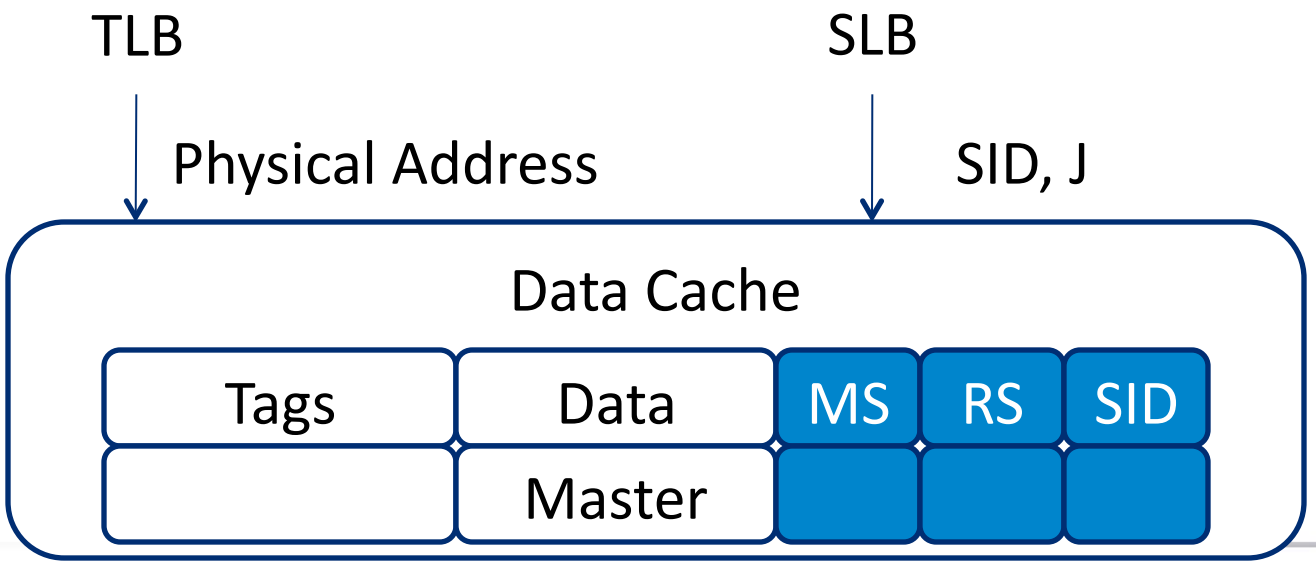
Create Snapshot = Scan

- Three-way handshaking
  - Through Inter-Processor Interrupt
    - O(P) scan time



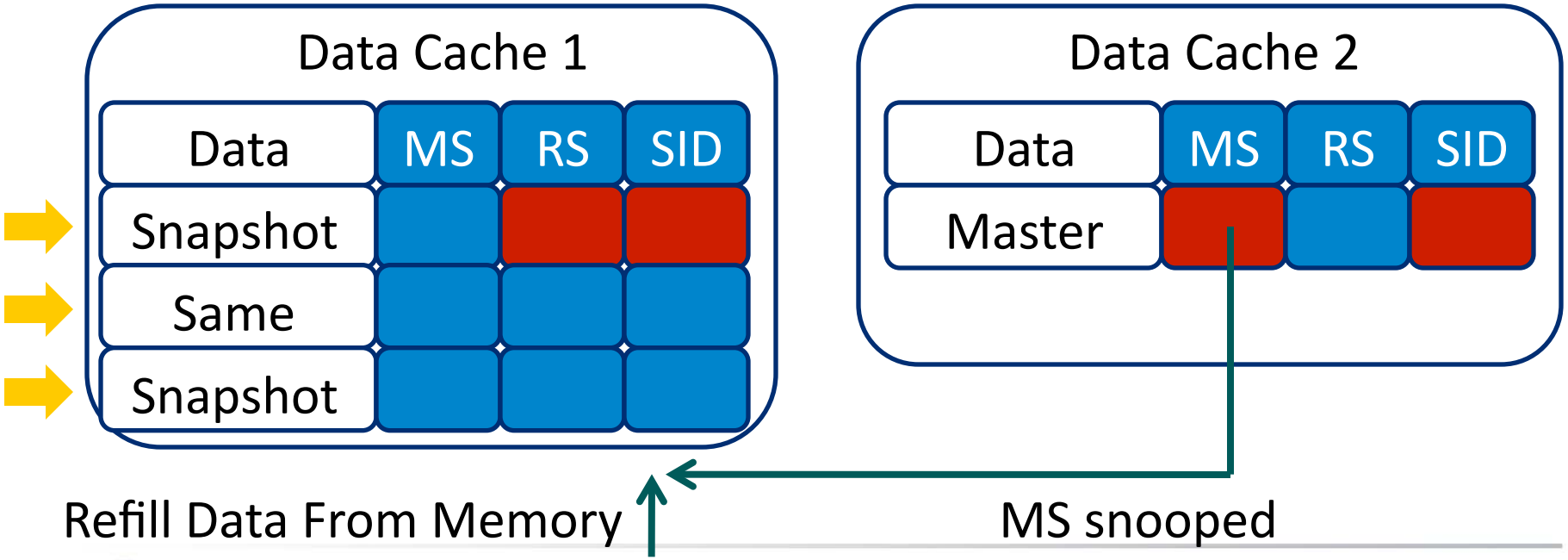
Write to Snapshot = update

- Hit if addr_tag matches and RS is not set.
  - Globally visible by invalidating other copies
    - O(1) update time
- Snapshot user can't write.



Read from Snapshot

- Snapshot user
  - Hit if addr_tag matches and MS is not set
  - RS is set if MS is piggy-backed in refill message



Destroy snapshot

- Invalidate cache lines with RS
- Gang-clear MS and RS
- Remove entry from SLB and SIT

Integration with TM

- Similarity between MShot and TM
  - Snapshot = read-only transaction that never aborts.
  - Snapshot image = last committed data
  - Master image = transactional data
  - Snapshot virtualization = transaction virtualization

- Our approach
  - Given TM resources, reuse them to low MShot implementation cost.

Baseline TM = Cache-based HTM

- Additional bits per cache line
  - TW : Transactional Write
  - TR : Transactional Read
  - TID : Transaction ID

- Page-based TM (PTM) for virtualization (ASPLOS '06)
  - Shadow page table
    - Secondary TLB to manage pages with overflowed transactional data
  - Home (original) page for transactional data
  - Shadow (new) page for last committed data

TM Resource Sharing

HTM resource	MShot resource
TM Metadata bits (TW,TR)	MShot Metadata bits (MS, RS)
TID	SID
TW/TR Gang-clear logic	MS/RS Gang-clear logic
Home/Shadow pages	Master/snapshot images
Shadow page table	Providing access to overflowed master/snapshot images

- More details in the paper

Hardware (Storage) Cost Calculation

- Assuming a baseline system
  - 16 cores, 64K L1, 1M private L2, 32B cache line, 16GB memory

Baseline System	Storage Cost
No TM Resources	2.98 %
TM with virtualization support	0.07 %
TM without virtualization support	0.43 %

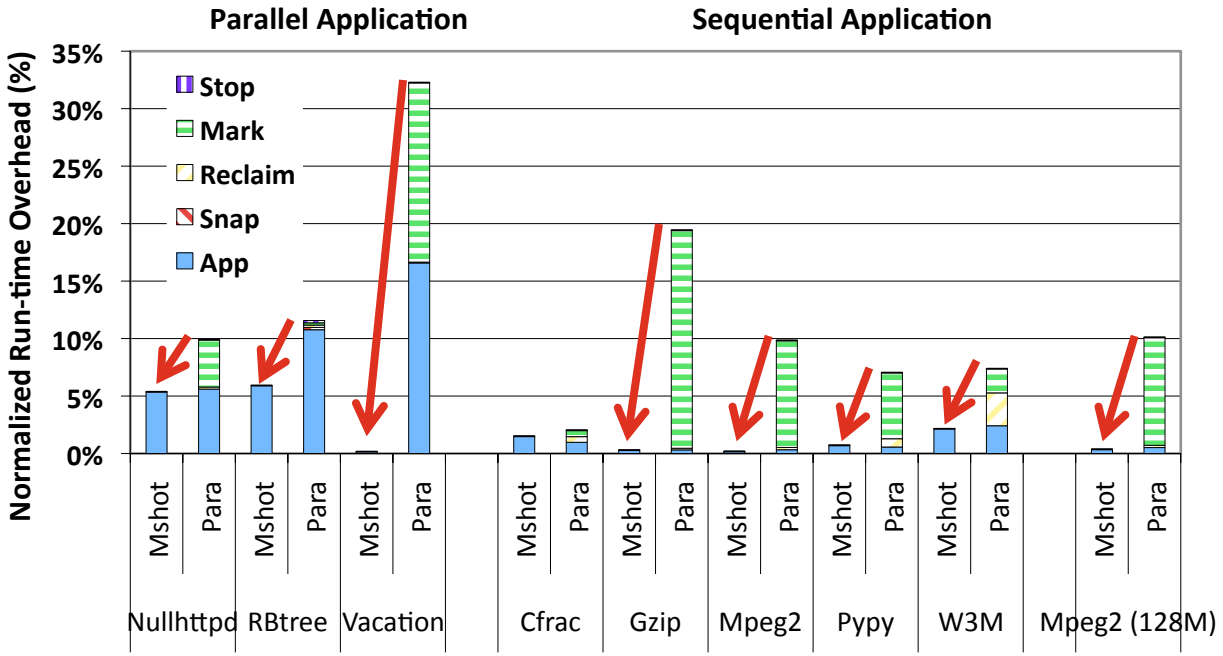
Simulation Environment

- Cycle-accurate multi-core simulator

Feature	Description
CPU	2GHz, single-issue, in-order x86 core
SLB	64 entries
Cache	64K L1, 1M private L2, 32B line, MESI
Shadow Page Table	2048 entries
Memory	4GH, 100 cycle latency
Interconnect	Tiled, 32B link, 3 cycles per hope

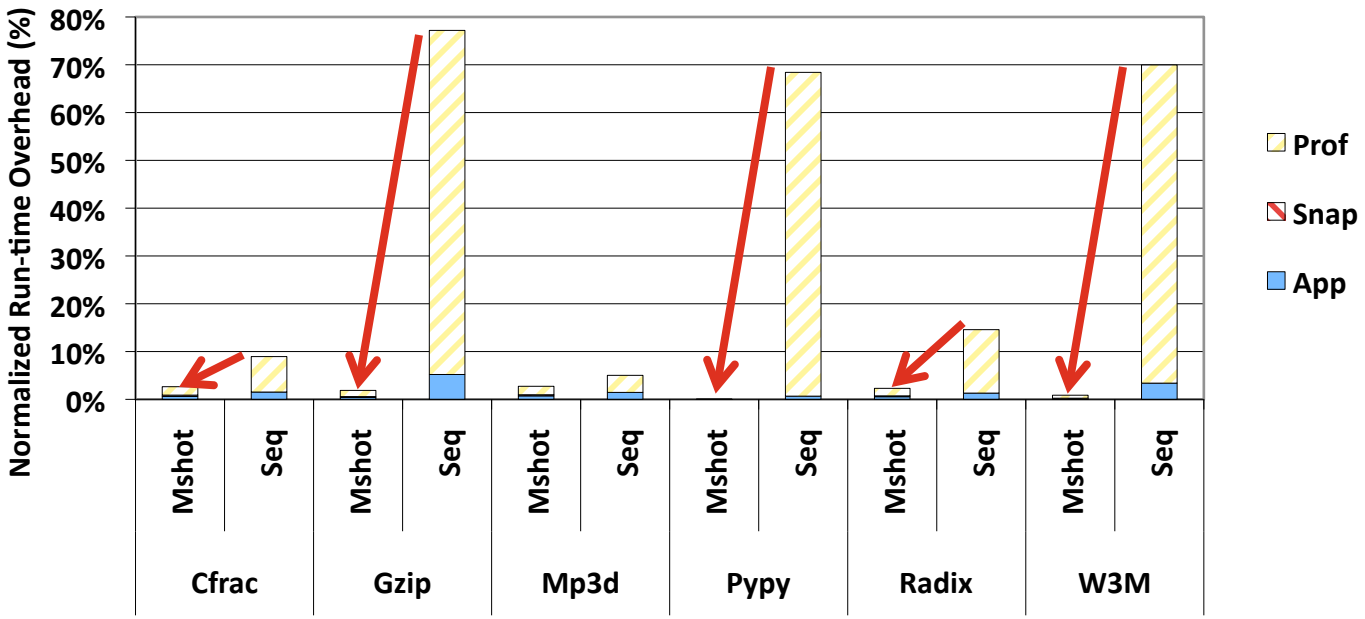
Snapshot GC

- Take a snapshot of a heap generation before GC
  - Collectors on snapshot image
  - Mutators (application threads) on master image



Snapshot Call-path Profiling

- Take a snapshot of thread stack before profiling
  - Profiler on snapshot image
  - Application threads on master image



Takeaway points

- Fast Memory Snapshot
  - Useful concurrent programming primitive
  - Atomic multi-word read operation
    - Snapshot GC, Snapshot Call-path profilers
- Cost-effective implementation with TM resources
  - Down to 0.07 % storage overhead
- AMD Advanced Synchronization Facility (ASF)
  - AMD 64 extension for Transactional Memory
  - http://developer.amd.com/CPU/ASF/Pages/default.aspx

Questions?

Jaewooong.Chung@amd.com