SW and HW support for Recording and Deterministically Replaying Concurrent Programs

Klaus Danne
Microprocessor and Programming Research, Intel Labs

Treffen der GI-Fachgruppe Betriebssysteme, 13. November 2009

Team

- Gilles Pokam
- Cristiano Pereira
- Klaus Danne
- Rolf Kassa
- Ali-Reza Adl-Tabatabai

Outline

- What is Deterministic Replay?
- Motivation and Usage Models
- Technology
  - Recording Non-determinism
  - Replaying
- Hardware Support
- Chunk-based MRR for modern CMPs
- Summary

What is Deterministic Replay?

- Definition deterministic replay:
  - **record** sufficient information during execution to enable a...
  - **replayer** to create equivalent execution (despite sources of non-determinism)
  - i.e. the replayer executes
    - the same instructions than in the original execution
    - and each dynamic instruction sees same values

- Sources of non-determinism:
  - Thread memory races
  - Inputs
  - Interrupts
  - DMA
  - Non deterministic instructions (e.g. RDTS, CPU_ID,...)

Motivation and Usage Models

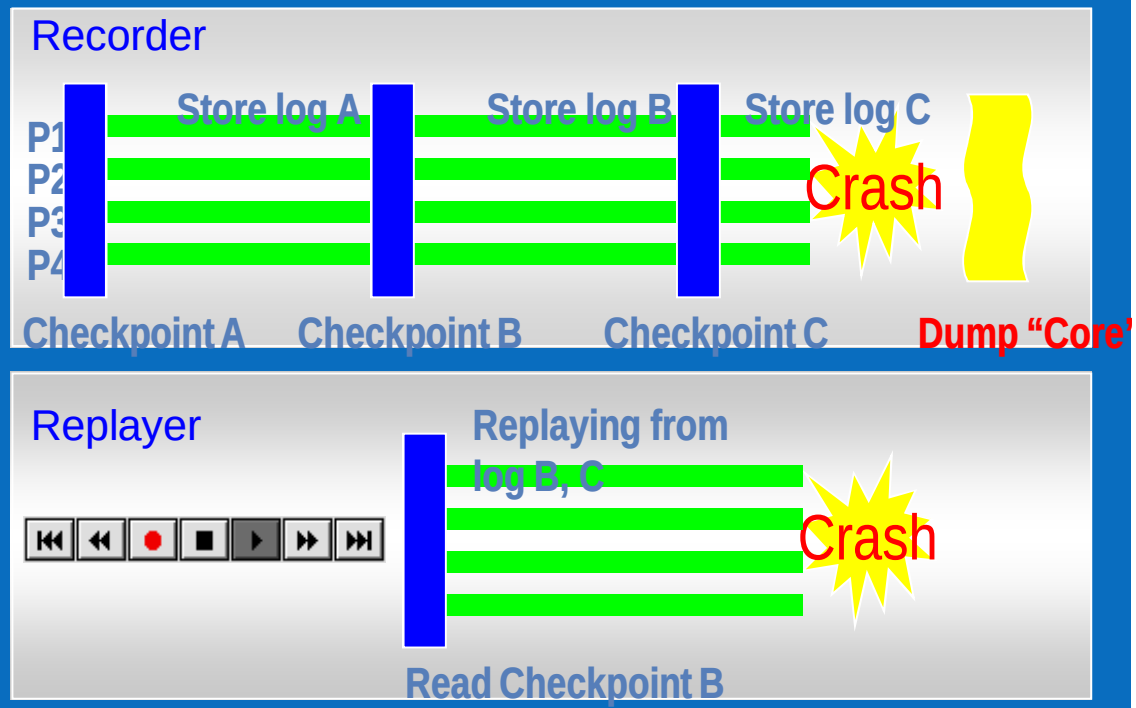
- Debugging
  - find especially concurrency bugs that are difficult to reproduce
  - enable new debugging tools
- Fault Tolerance
  - e.g. keep 2nd system in consistent state that can takeover on failure
- Security
  - e.g. analyze execution history

Debugging Scenario

- Advanced debugging of MT workloads
- Reproduce non-deterministic bugs

- Requirements:
  - Minimum impact (No masking of bugs)
  - Recording at high speed (in production run)

[src: Xu et. al., *FDR*, ISCA 03]



Fault Tolerance

- Two system in virtual lock-step
  - Primary VM
    - Records non-determinism
    - Sends log to secondary
  - Secondary VM
    - Replays equivalent execution
    - Acknowledges progress
- In case of fault
  - Secondary takes over
- Commercial product of VMware®
- Requirements:
  - Recording and replaying at high speed (always on, slower is bottleneck)

see: [VMware® white paper "Protecting Mission-Critical Workloads with VMware Fault Tolerance"]

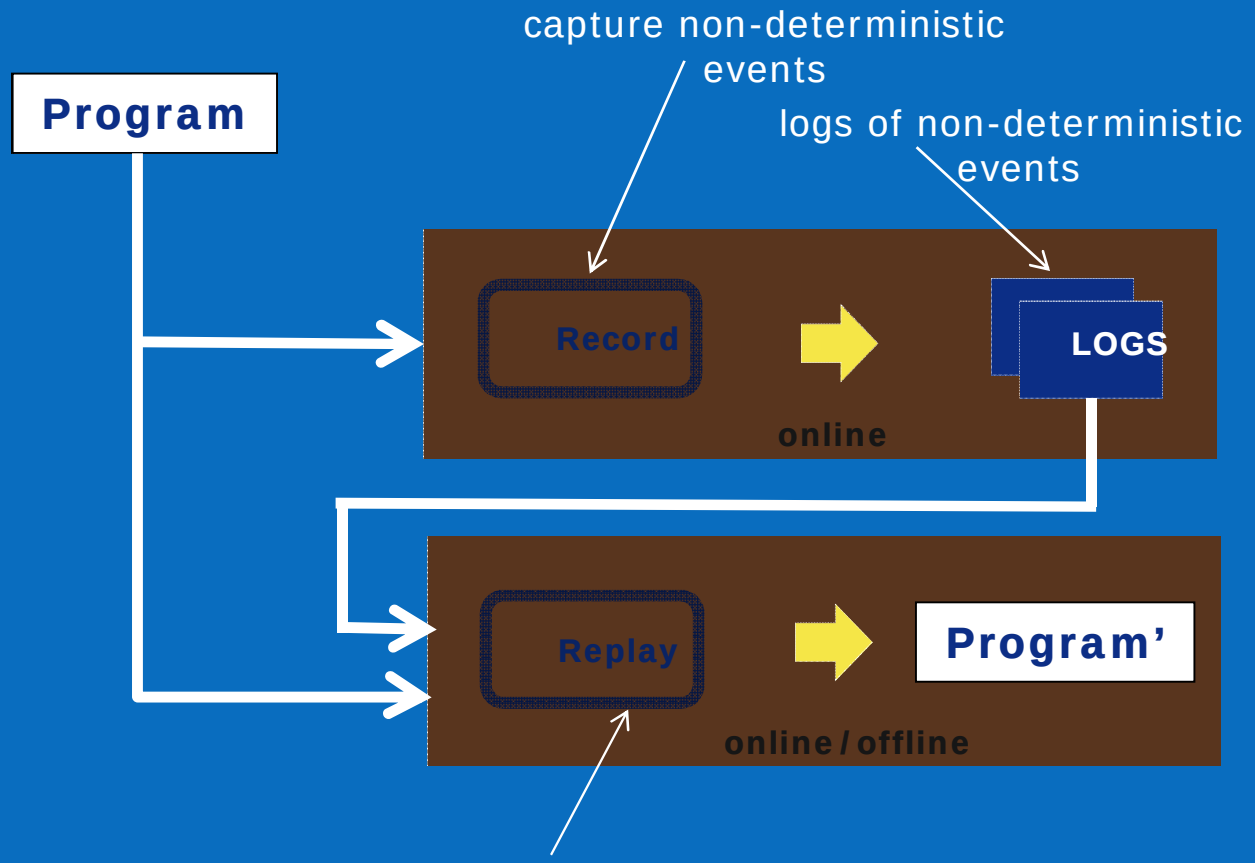
Security – example

- Critical server with sensitive data
- Recording always on, keep log for months
- security hole becomes known
  - Replay past moth, analyze if flaw was exploit
    - If yes, check intruder actions
      - E.g. what credit card numbers where stolen
- Other examples:
  - Check execution of WEB-content in sandbox before releasing it to main system
  - Forensics: employ expensive analysis on past execution to find abnormal behavior
- Requirements:
  - Recording at high speed (always on)
  - Small log size (keep for months)

Technology:

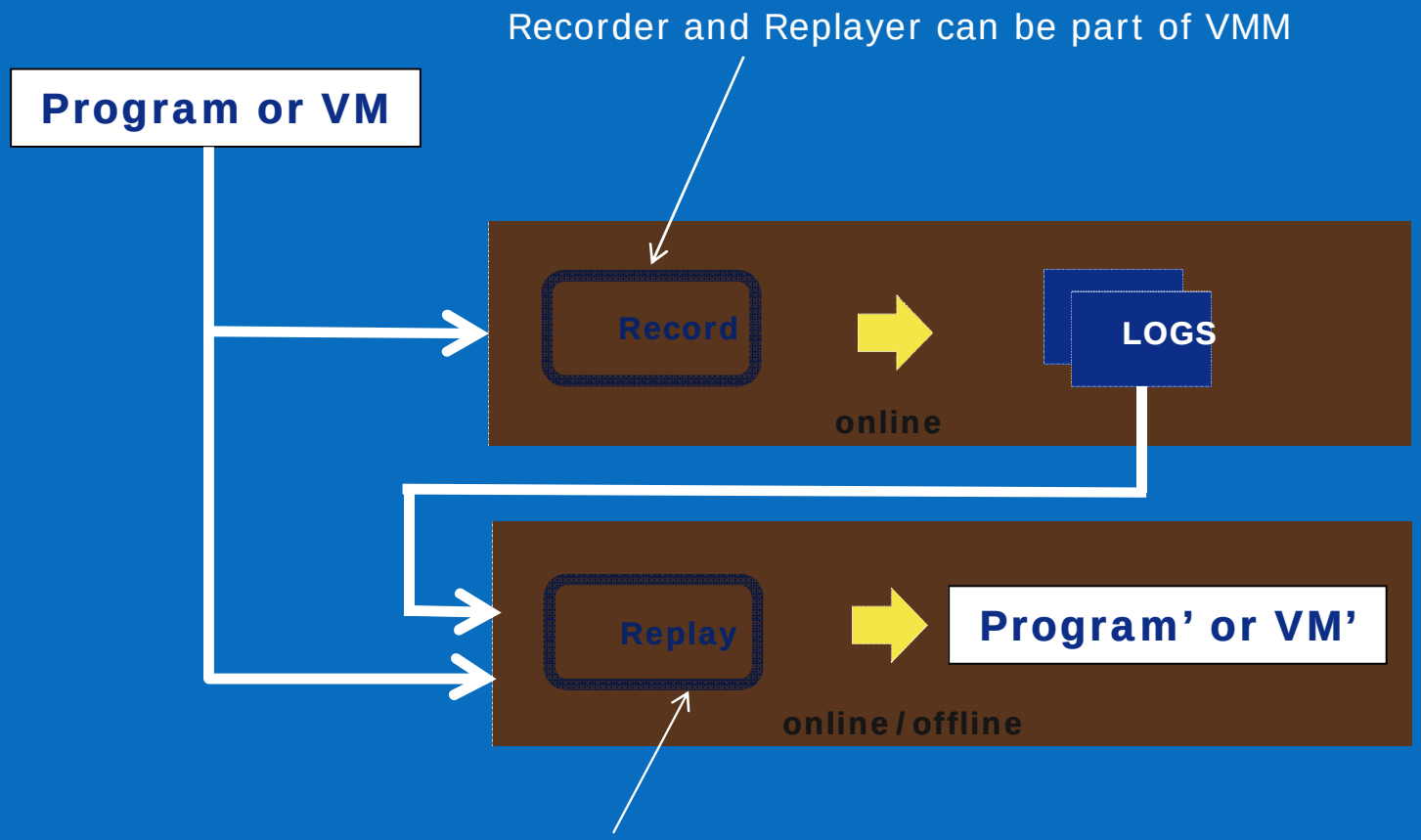
Technology

recording the non-determinism

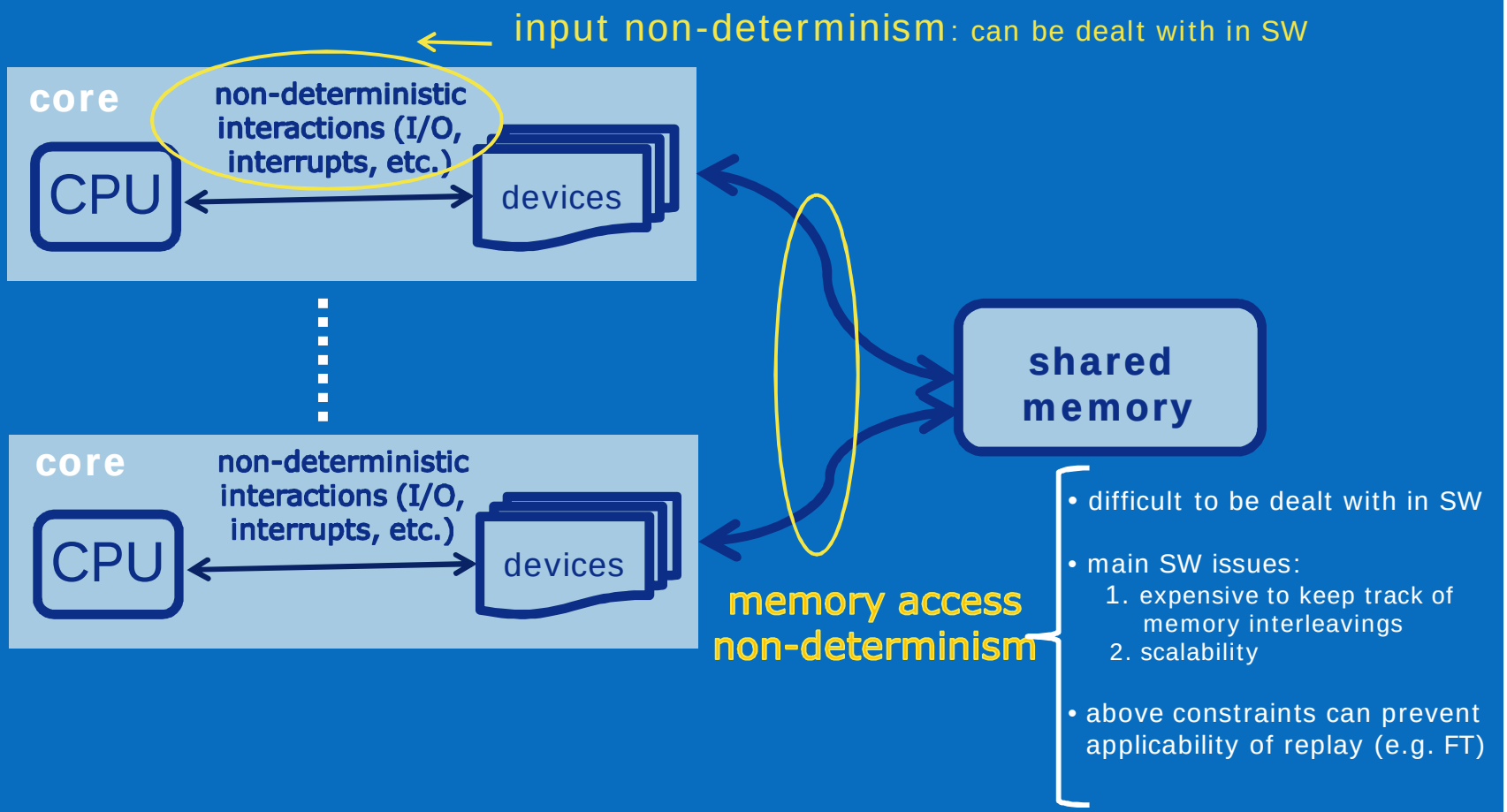


Technology

Application Replay vs. Full system



Sources of non-determinism in a CMP



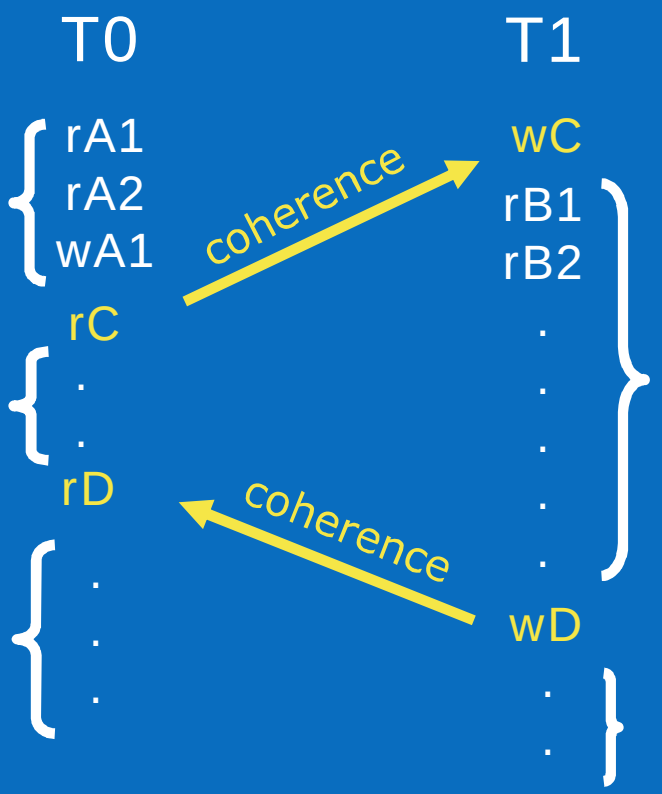
Explore HW support for fine-grained logging of memory access interleavings

Hardware Support:

Memory Race Recorder: detecting memory interleavings

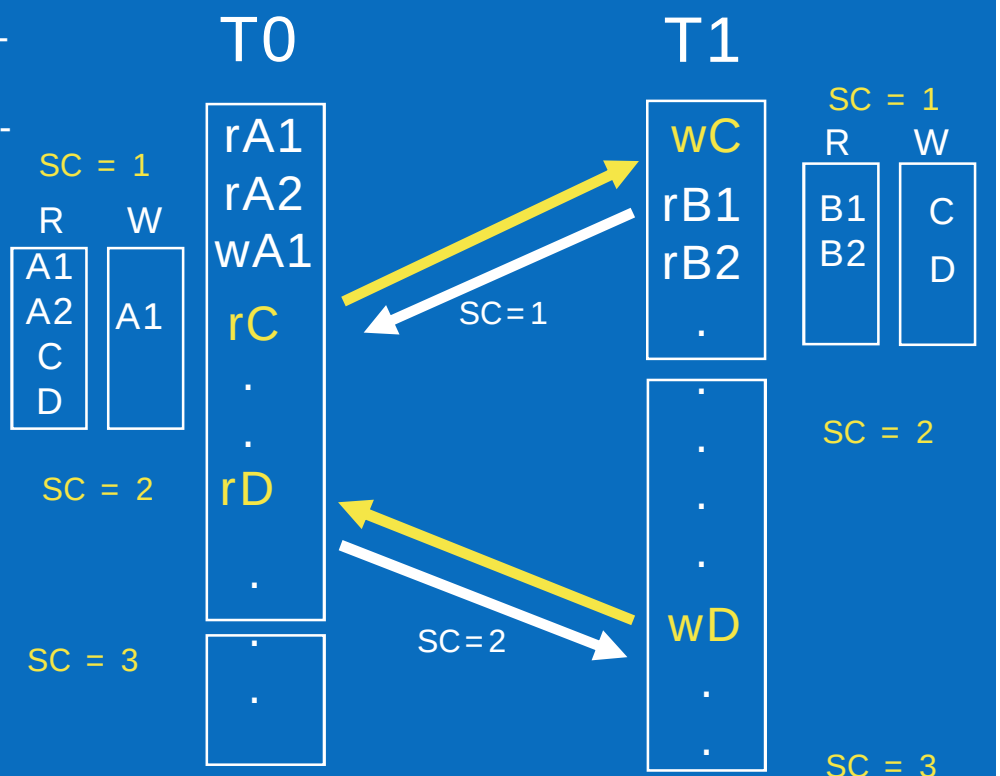
- memory interleavings are revealed using cache coherence messages

- Why not just logging coherence?
  - Much too many data if we log each dependency
  - Don't log dependencies implied by others!
  - Need to record progress of threads for replay



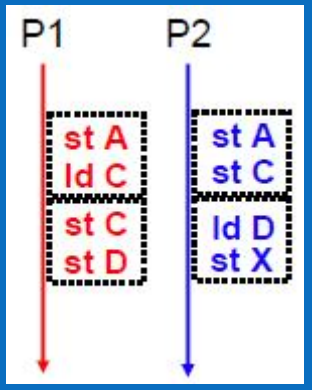
Chunk-based Memory Race Recorder:
example [RERUN, ISCA 2008]

- episode/chunk:
  - sequence of instructions without shared-memory dependencies
    - signature buffers (SB) to capture read-set **R** and write-set **W**
    - SB determines if logging is needed
  - Lamport scalar clocks to order chunks
    - SC:= SC+1 with new chunk
    - $SC_{local} := \max(SC_{local}, 1+SC_{remote})$
- Recording:
  - Order of chunks by SC
  - Length in # instr. of chunks
- Replaying:
  - Execute one chunk at a time
  - Pick chunk with smallest SC and execute #inst. of the corresponding thread



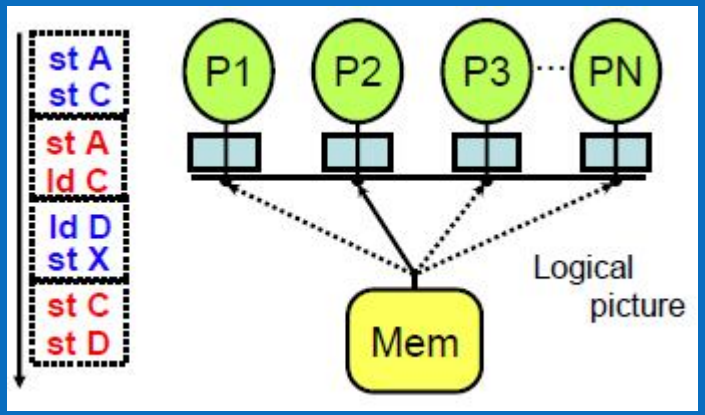
Chunk-based Memory Race Recorder: example [DeLorean, ISCA 2008]

- Chunk based “BULK-SC” execution environment:
  - similar to transactional-memory
  - makes execution environment more deterministic
  - large sequence of instructions executed atomically and in isolation (usually fix size e.g. 1000 instr.)
  - commit to state at once
  - concurrent threads are squashed, if conflict
  - signature buffers (SB) to capture read-set **R** and write-set **W**



- Recording:
  - Conflicts occur only on chunk boundaries
  - Only log order of chunk commitment
  - Log chunk-size in case of interrupted chunks

- Replaying:
  - Force HW to commit chunks in same order as during recording

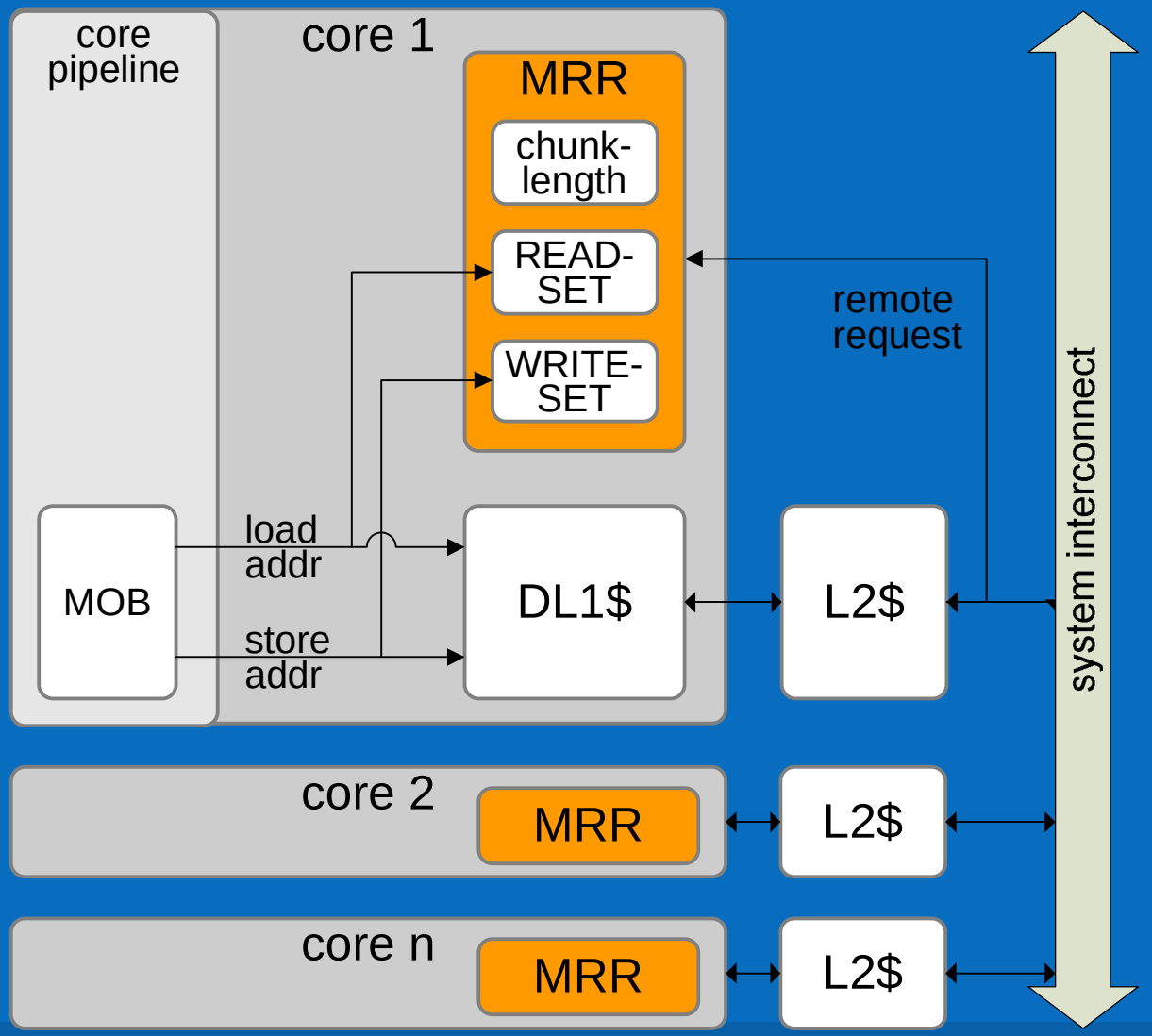


[src: BulkSC, ISCA07]

Chunk based MRR for modern CMPs:

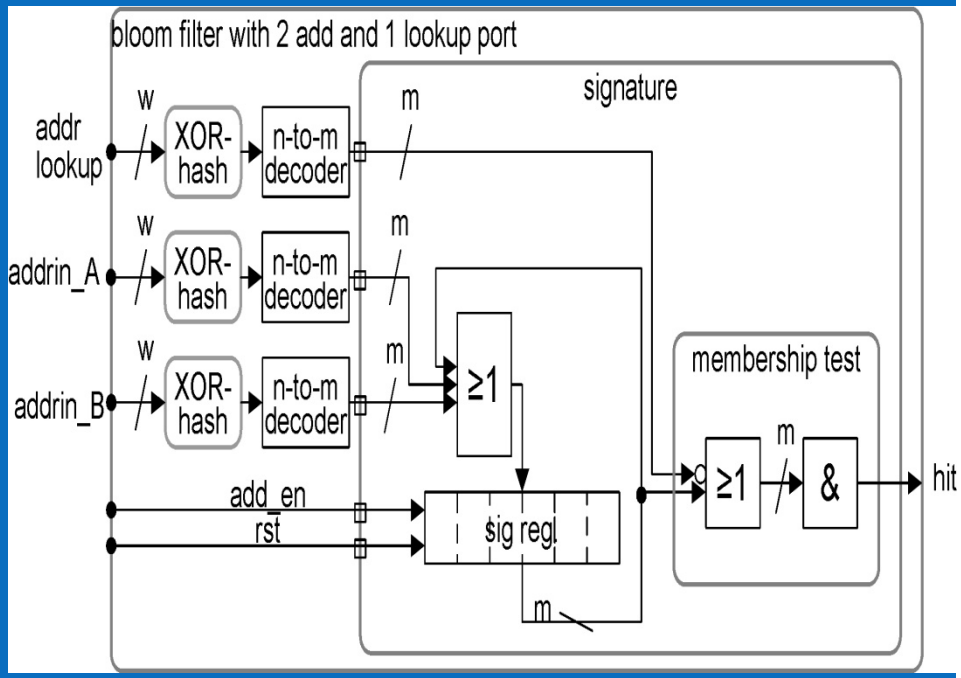
MRR: system overview

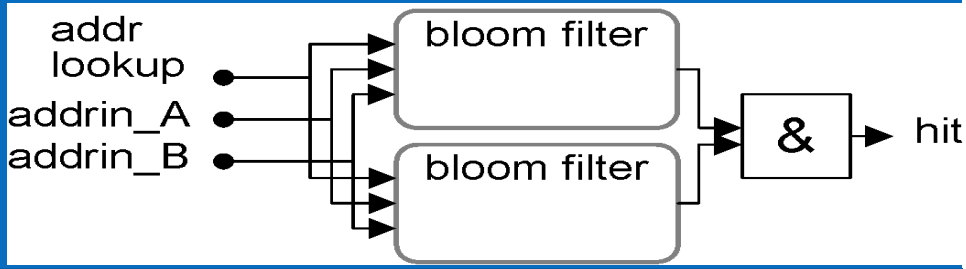
- Each core extended by MRR
- Keeps track of read/written addresses and chunk-length
- Remote read/write are lookup; if conflict -> log entry & clear state



MRR: read/write set as bloom-filter

- Bloom-filter is data-structure to approximate infinite set with finite state
- Signature (e.g. 1024 bit), represents set
- Elements (e.g. addresses) are hashed to a bit-string
  - Add: bit-string is ORed to existing signature
  - Lookup: bit-string is checked in signature
  - Reset: clears signature
- False-positives but not false-negative
  - (MMR logs some false conflicts)
- May design tradeoffs:
  - Parallel BF
  - Signature Size
  - Hash-function
  - How many ports
  - ...





MRR for modern CMPs

"*Architecting a Chunk-based Memory Race Recorder in Modern CMPs*"
at MICRO 2009

- Main contributions:
  - MRR simulation environment based on CMPSim (PIN based cache simulator)
  - New scheme to maintain causality that significantly reduces the coherence traffic overhead of previous proposals
  - New signature placement design for snoop-based CMP
  - New chunking schemes to improve replay speed

- See the MICRO paper for details, here are just some results

MRR logging results

- Most apps log less than 0.1 chunks per 1000 instructions (except ODE)

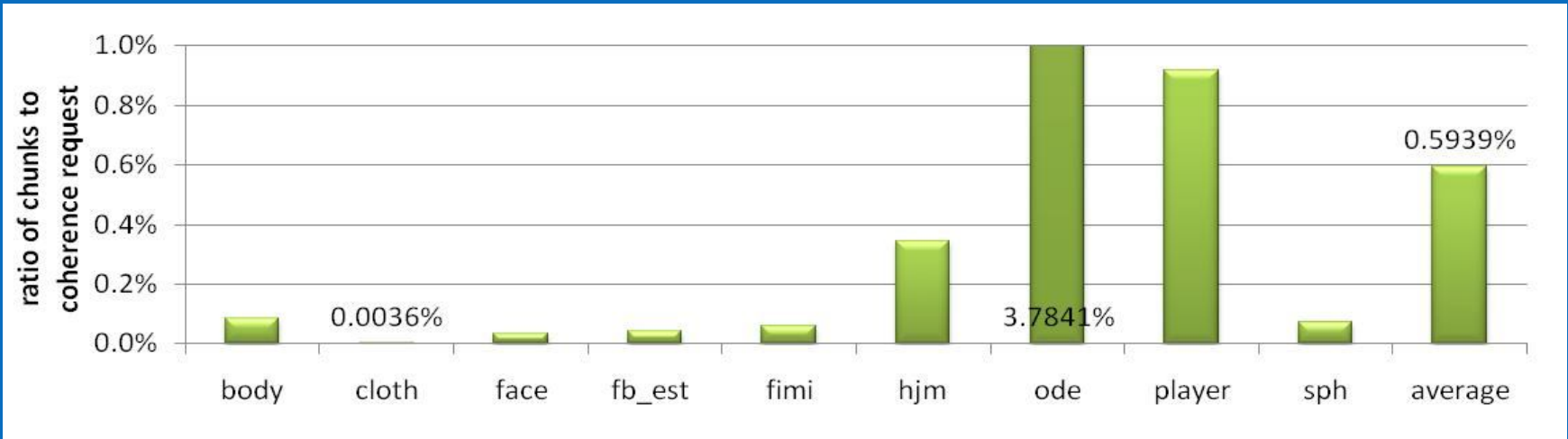
body	cloth	face	fb_est	fimi	hjm	ode	player	sph	avg
4.00E-02	1.98E-04	4.92E-02	3.93E-02	5.06E-02	7.75E-06	4.95E+00	1.94E-02	4.94E-02	6.50E-01

chunks per 1000 instr.

- Only a small fraction of coherence messages lead to log entries

body	cloth	face	fb_est	fimi	hjm	ode	player	sph
1189	28136	2828	2304	1622	291	26	109	1386

average number of coherence requests per chunk



Summary

- Deterministic Replay has valuable use cases
  - Debugging / Fault Tolerance / Security
- SW solutions available for single processor systems
- Multiprocessors need HW support for efficient logging
  - Chunk based approaches are most promising
  - Log size is affordable (some few bytes per 1000 instructions)
  - HW signatures play a key role
- Modern CMPs rise issues
  - some we attacked [MICRO2009]
  - some are subject to future work


intel
Leap ahead™


intel