

Building Practical Deterministic Replay Systems for Multiprocessors

Pablo Montesinos, Matthew D. Hicks, **Samuel T. King** and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign



Motivation

- ■ Time travel is an alluring concept in Computer Science
 - ■ Allows us visit and recreate past states and events in the computer

- ■ Time travel has a wide range of applications:
 - ■ Programmers can use it to debug their non-deterministic code
 - ■ Sysadmins use it to inspect the actions of an attacker
 - ■ System designers use it to recreate state after a system crash

- ■ Time travel can be achieved using Deterministic Replay of Execution



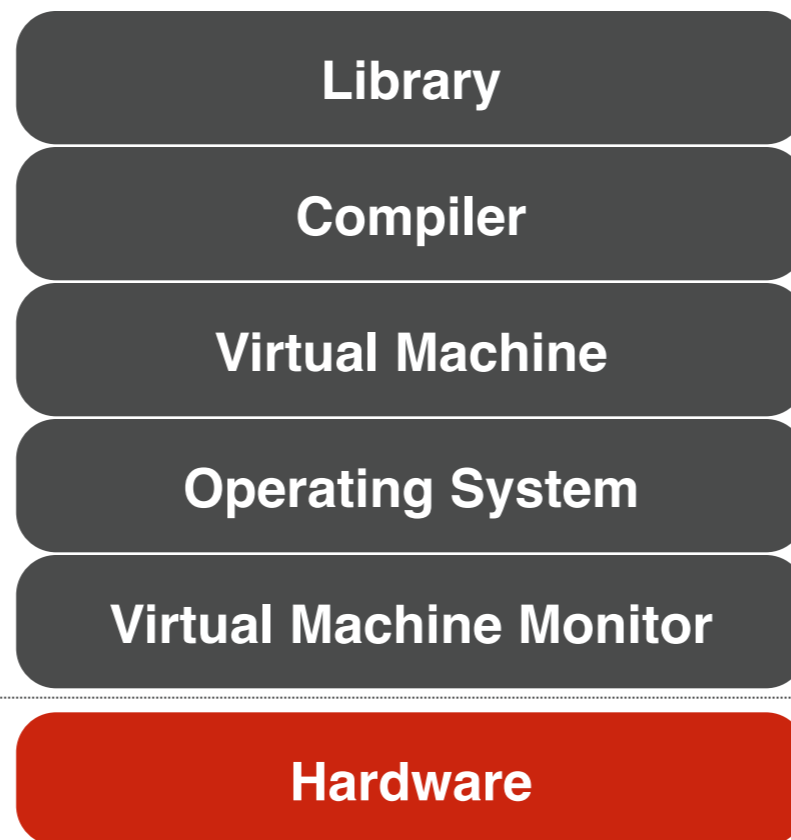
Deterministic Replay of Execution

■ ■ Intuitive concept:

1. Record key events during initial execution
2. Restore to a previous checkpoint
3. Replay recorded log to force software down the same execution path



Approaches to Deterministic Replay



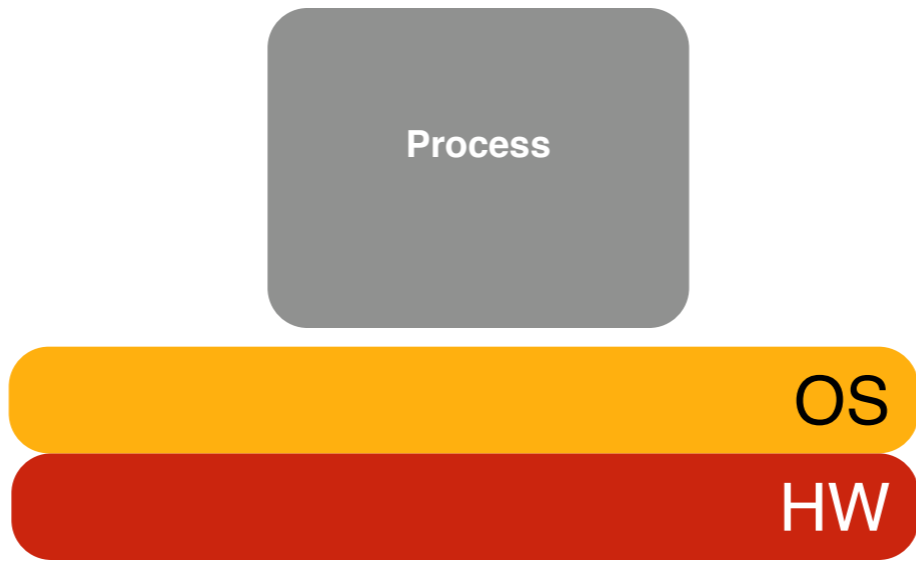
Problems with Current Deterministic Replay Schemes

- ■ SW-only schemes are flexible
 - ■ Perform slowly (or not at all) with multiprocessor systems

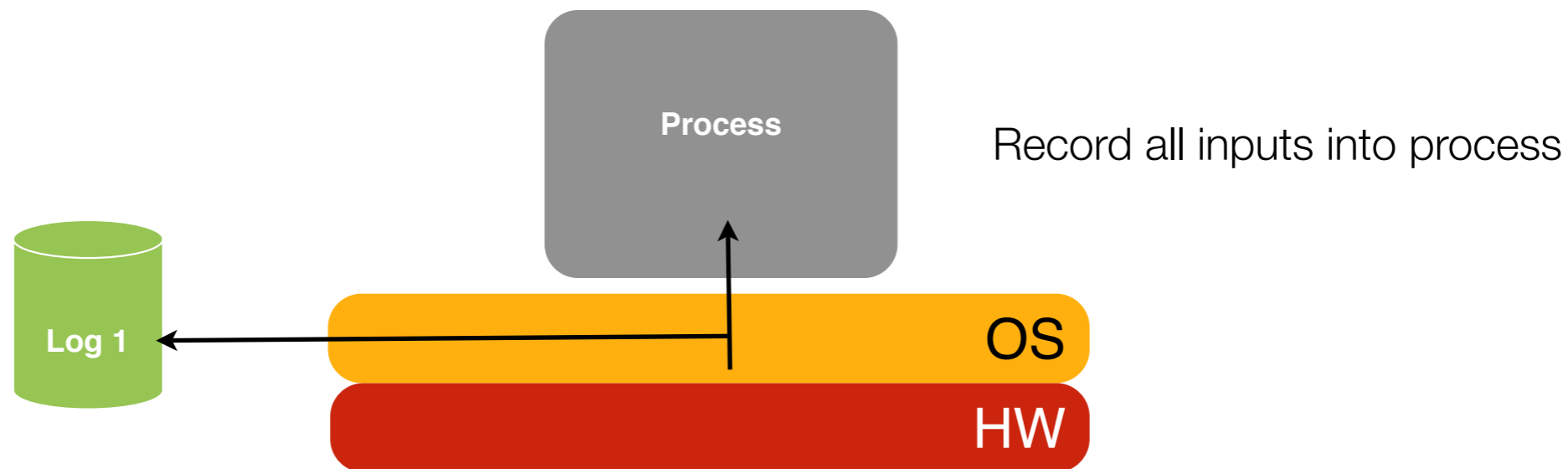
- ■ HW-based schemes offer excellent multiprocessor support
 - ■ But they are largely impractical



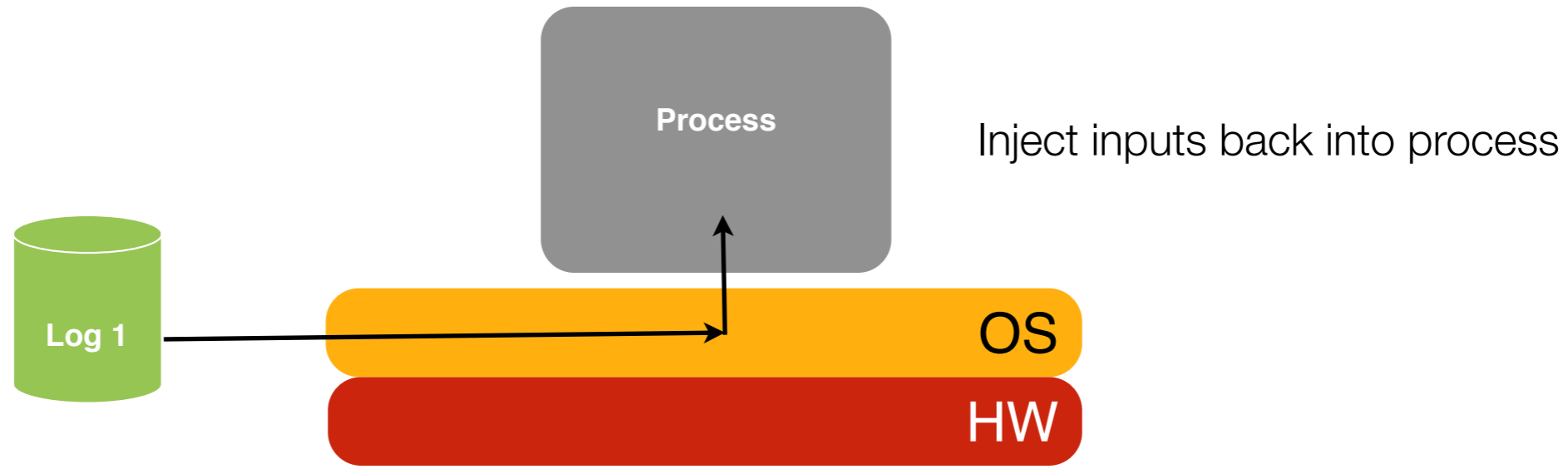
Software replay systems



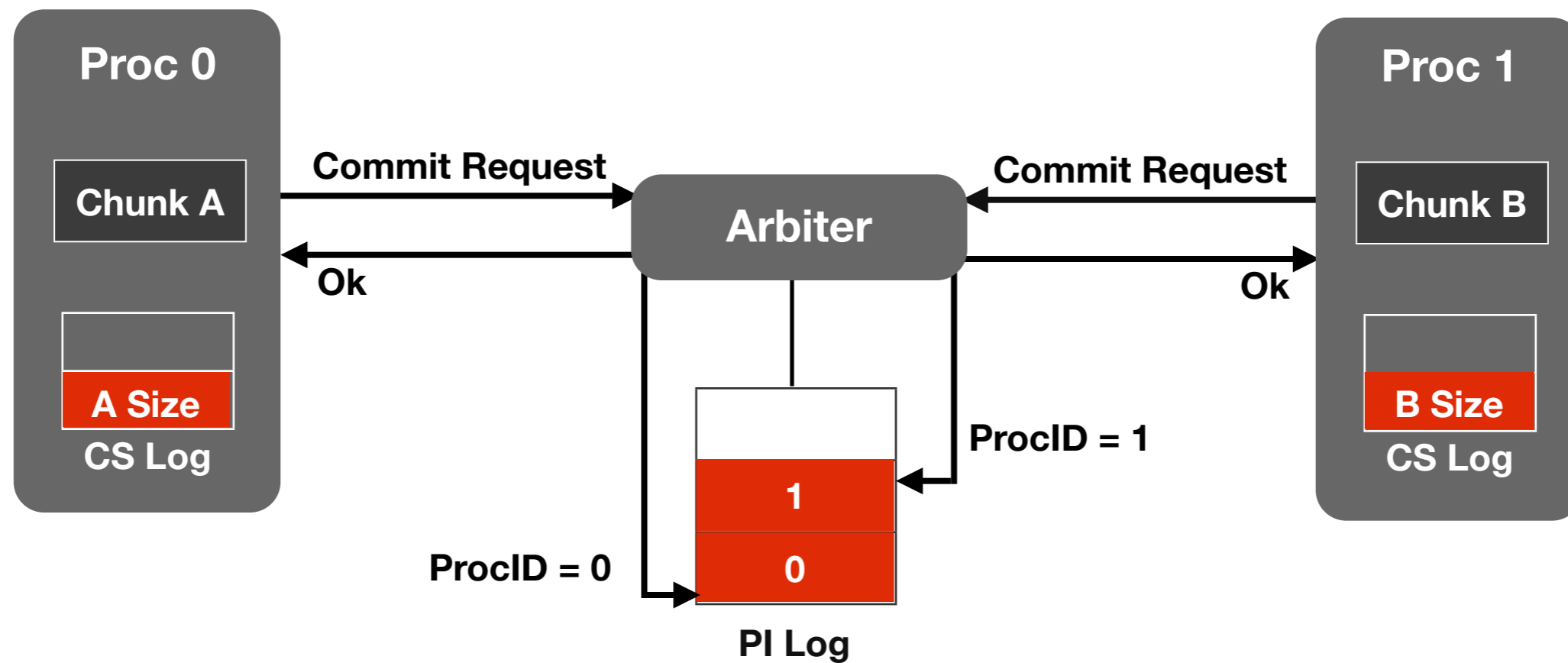
Software replay systems -- recording



Software replay systems -- replaying



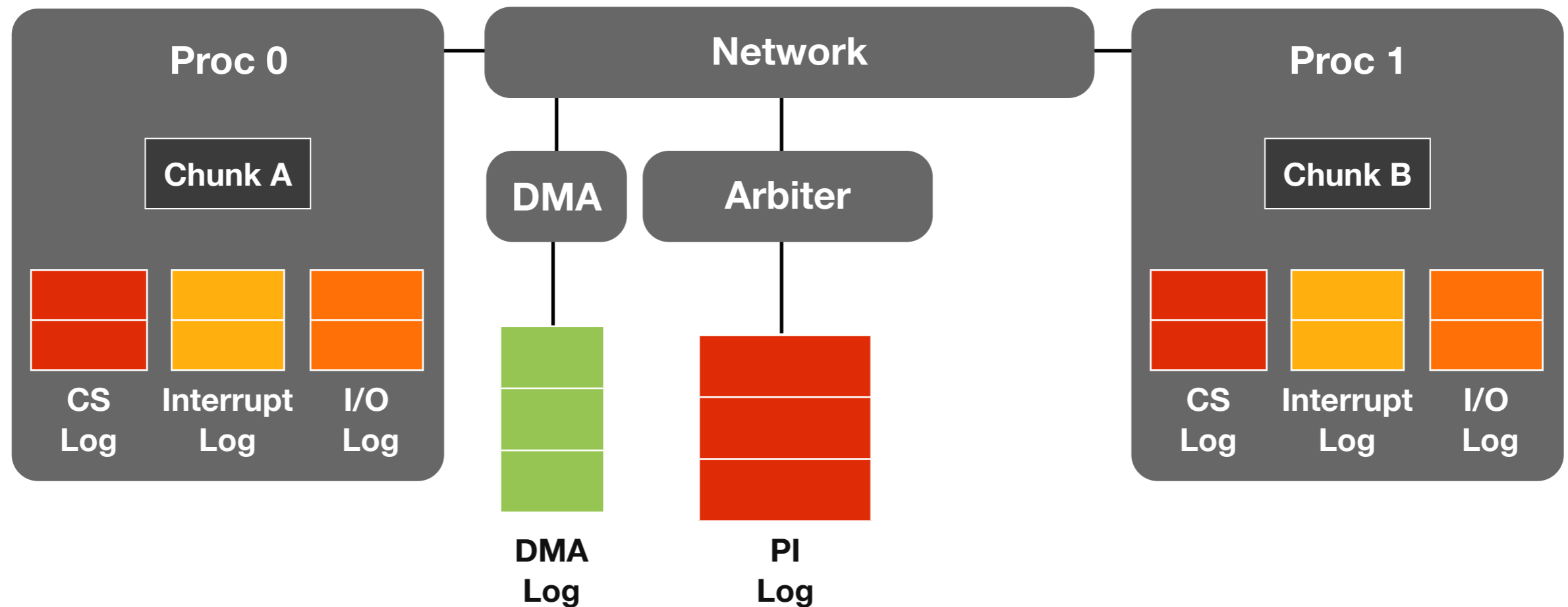
Basic DeLorean Operation



- Log implemented as two different structures:
 - Chunk Size (CS) Log
 - Processor Interleaving (PI) Log



Overall DeLorean System



- Interrupt, I/O and DMA logs are common to other HW-based schemes



Problems with Current Deterministic Replay Schemes

- ■ SW-only schemes are flexible

- ■ Perform slowly (or not at all) with multiprocessor systems

- ■

Must redesign HW mechanisms and carefully integrate them with SW in order to make HW-Assisted replay practical

- ■

- ■

- ■

- ■ Can't record and replay individual software component (e.g., process)

- ■ Require complex VMM or simulator to replay execution



Capo: Making HW-assisted Replay Practical

- Defines a set of abstractions and a SW-HW interface for practical HW-assisted deterministic replay
- Capo's key abstraction: **Replay Sphere**
 - Separates the responsibilities of the HW and the SW components
 - Separates SW that is being recorded (replayed) from the rest
- Capo's first implementation: **CapoOne**

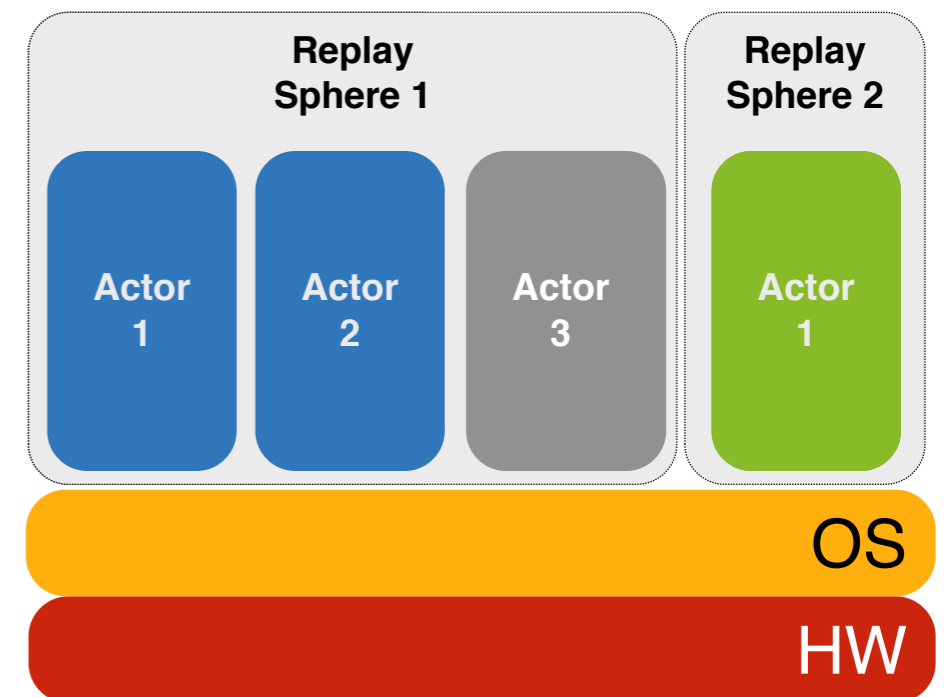


Replay Sphere

- ■ Set of threads recorded/replayed as a unit and their address space
 - ■ Only user-mode threads run inside spheres
 - ■ Threads running inside a sphere are Actors
 - ■ Threads that share memory must run within same sphere, and different applications might run within the same sphere

- ■ HW records/replays per-sphere actor interleaving

- ■ SW records/replays inputs to spheres

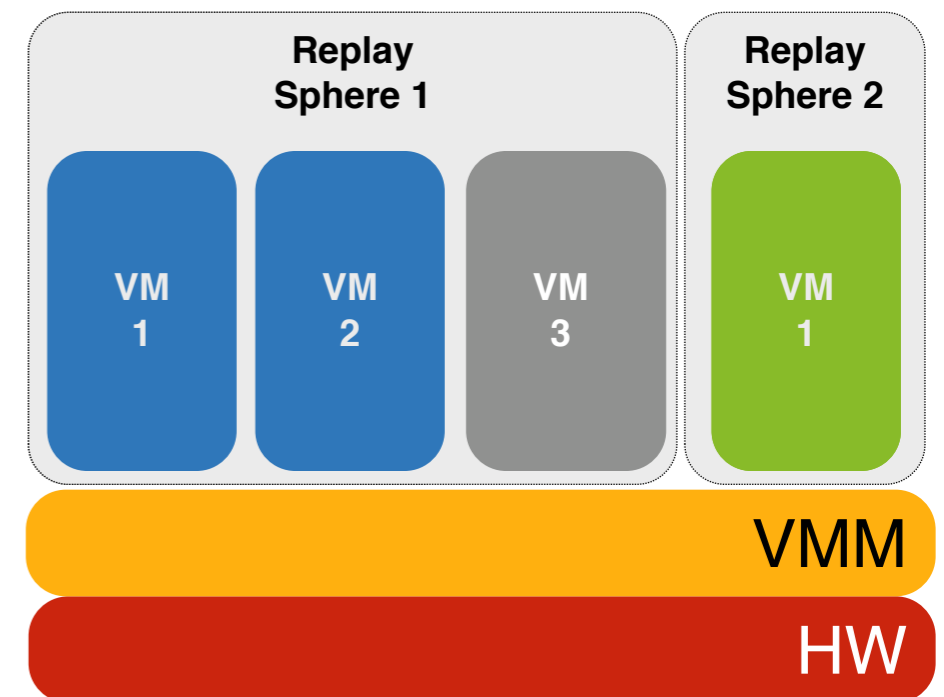


Replay Sphere

- ■ Set of threads recorded/replayed as a unit and their address space
 - ■ Only user-mode threads run inside spheres
 - ■ Threads running inside a sphere are Actors
 - ■ Threads that share memory must run within same sphere, and different applications might run within the same sphere

- ■ HW records/replays per-sphere actor interleaving

- ■ SW records/replays inputs to spheres



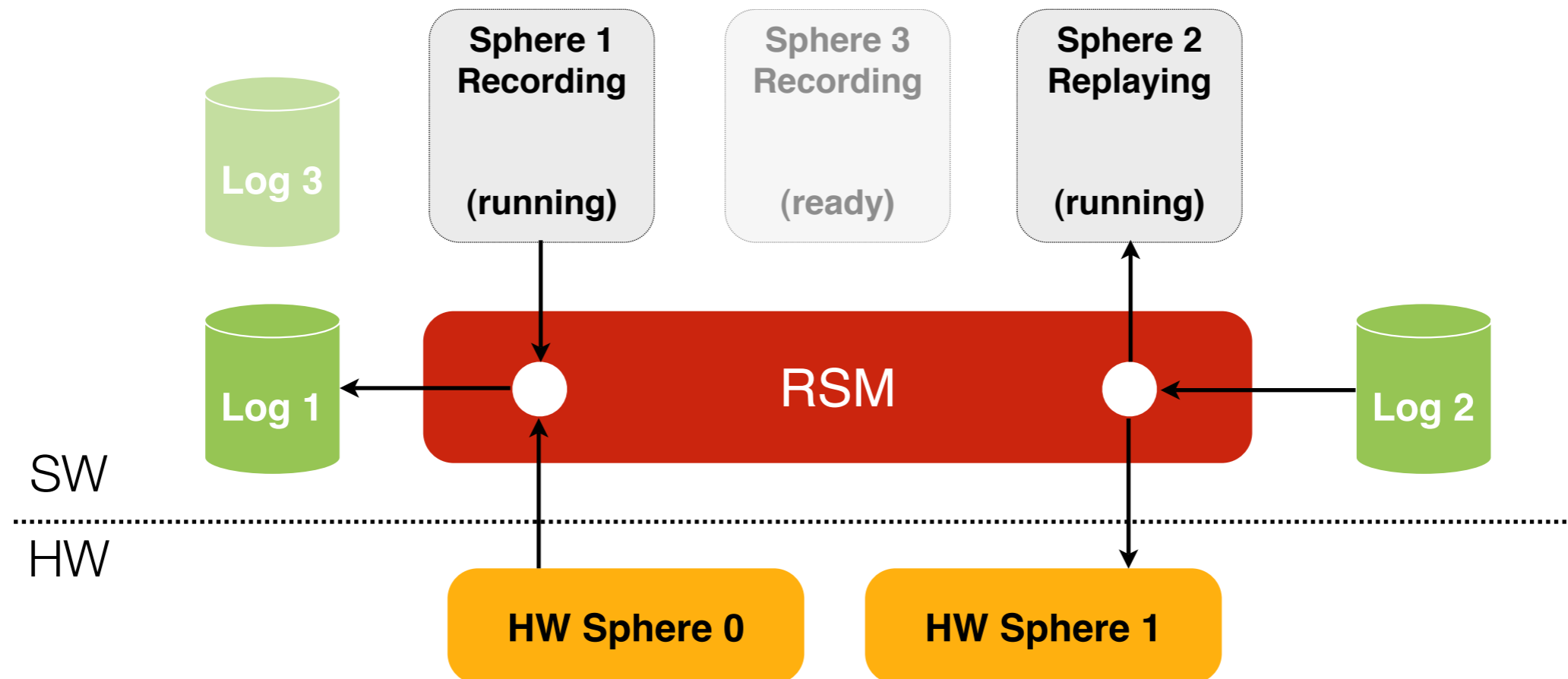
Separation of HW and SW Duties

- HW generates Actor Interleaving Log during recording and enforces same interleaving during replay

- SW is in charge of everything else:
 - Places actors that share memory in the same sphere
 - Generates Sphere Input Log during recording and injects inputs back to sphere during replay
 - Ensuring same virtual memory mapping for actors during recording and replay
 - Ensures same assignment of ActorIDs during recording and replay



The Replay Sphere Manager



- RSM provides the illusion of infinite amounts of HW Spheres
- Number of HW spheres limits number of spheres that can be recorded/replayed concurrently

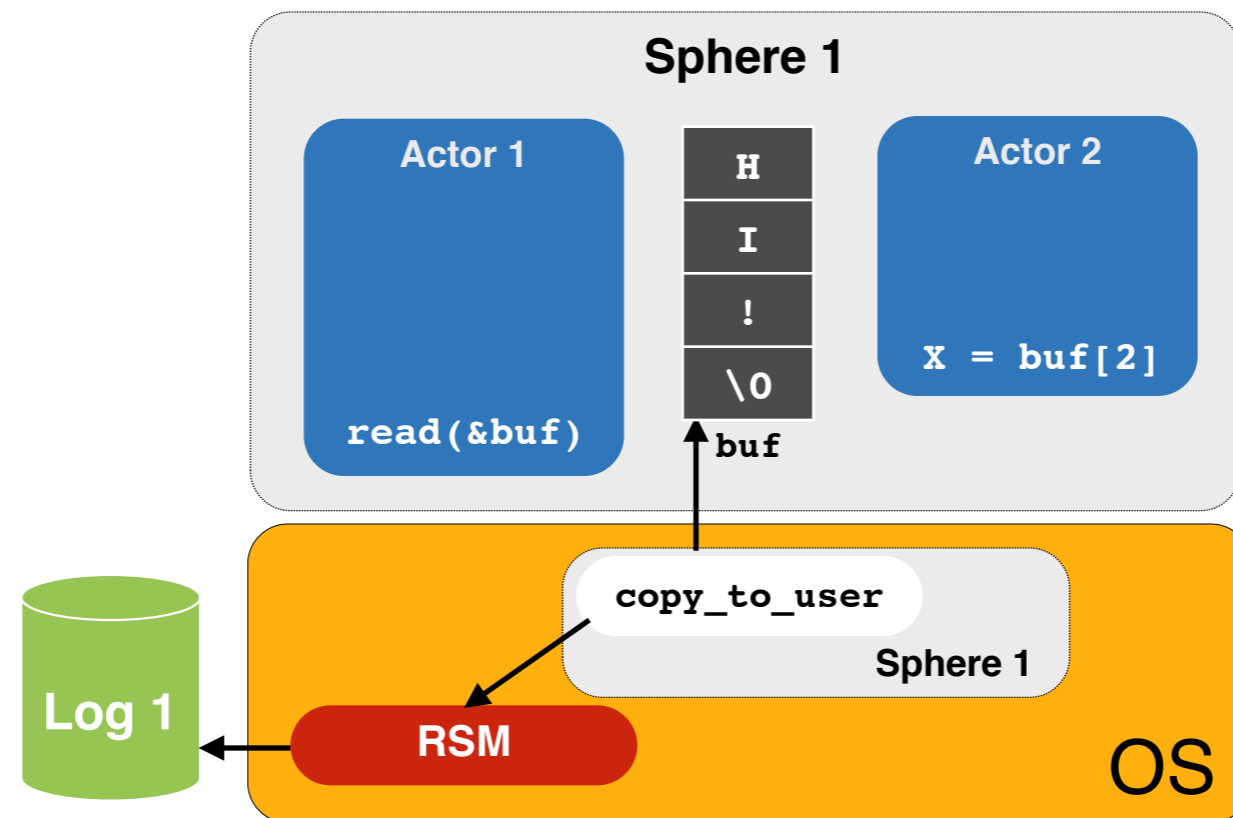


Three RSM Challenges

- ■ Maintain deterministic interleavings when copying data into a sphere
- ■ Emulating vs. re-executing system calls
- ■ Replay a sphere with fewer processors than were used during recording



Copying Data into Spheres



- Problem: copying data into sphere may cause non-replayable execution
- Solution: RSM inserts `copy_to_user` into sphere so HW records interleaving



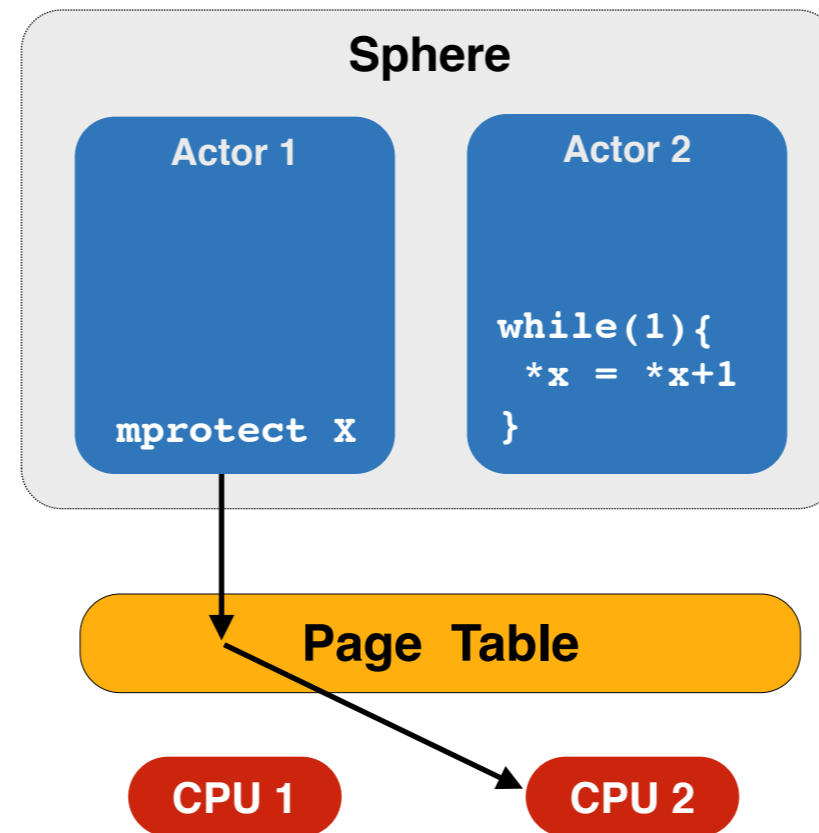
Emulating vs Re-Executing System Calls

- ■ During replay, the RSM emulates most system calls:
 - ■ RSM injects return values found in Sphere Input Log, squashes outputs
 - ■ Example: `gettimeofday`

- ■ Some have to be re-executed
 - ■ Thread management system calls. Example: `clone`
 - ■ System calls that modify address space. Example: `mprotect`



Implicit Dependencies



- Actor changes mapping or protection of address space, and another actor uses this changed address space
- RSM can express these dependencies to hardware so these interactions can be recorded



Replaying with a Lower Processor Count

- ■ Problem: next actor to replay is not scheduled

- ■ Solution 1: HW detects problem and sends interrupt to RSM
 - ■ Efficient, but it requires additional HW support

- ■ Solution 2: RSM inspects HW log and tries to prevent problem
 - ■ Not trivial, requires changes to OS scheduler

- ■ Solution 3: RSM does nothing, simply waits for OS to schedule actor
 - ■ Simple, but can hurt performance



Capo's HW Interface

- Must be independent of the HW-assisted replay system used
- Per-processor registers:
 - RSID: identifies which sphere this processor's events should be logged into
 - ActorID: used to tag events in both HW and SW logs
- Per-sphere registers:
 - Mode: specified whether the sphere is recording or replaying
 - Log pointers: used to handle each sphere's Actor Interleaving Log
- Interrupt-driven buffering interface
 - Informs SW whether Actor Interleaving Log is empty or full



CapoOne: An Implementation of Capo



- Hybrid SW-HW deterministic replay system
- Records and replays applications only
- Can mix recording, replaying and traditional execution concurrently



CapoOne: RSM Implementation

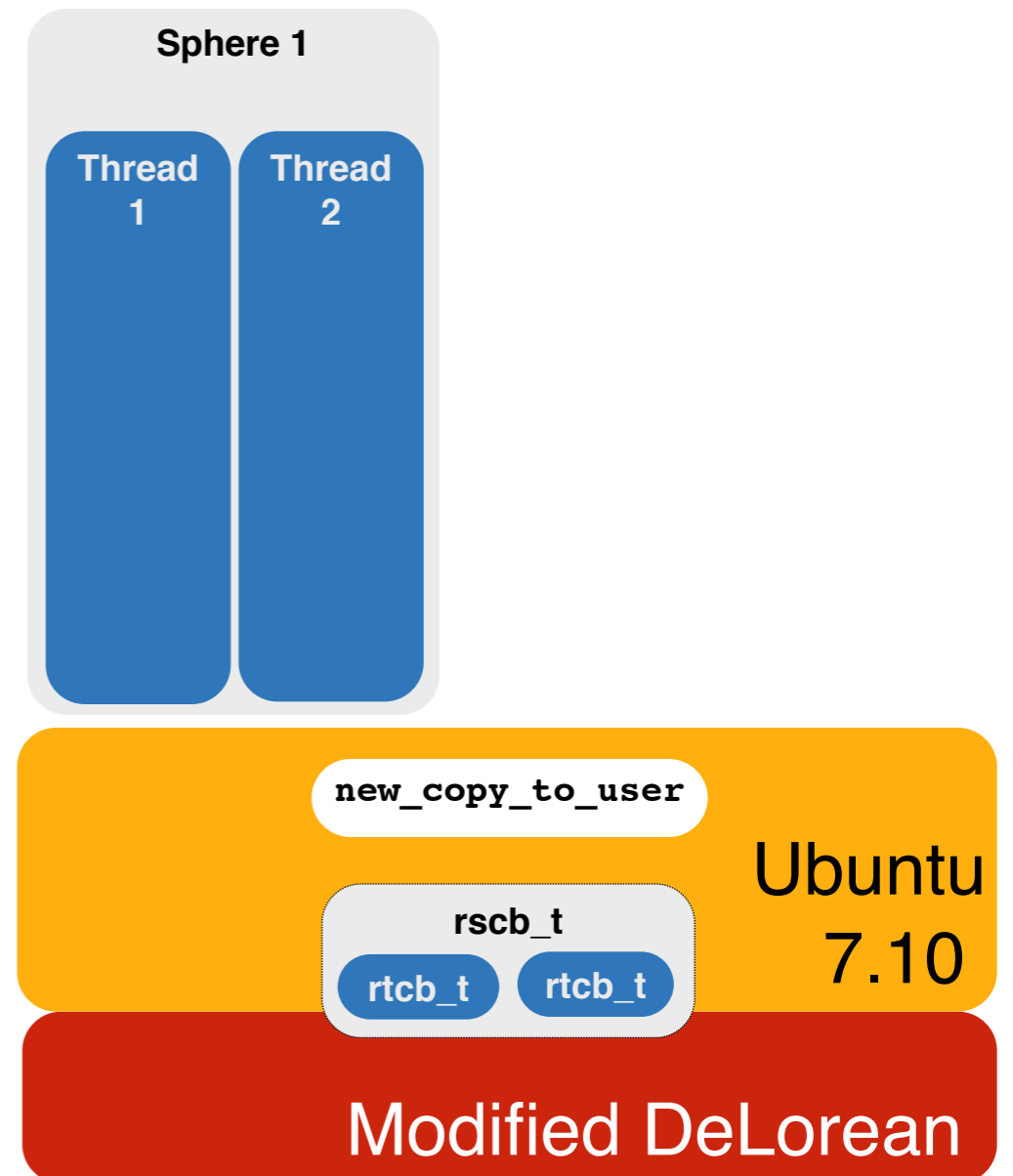


- Hybrid SW-HW deterministic replay system
- Records and replays applications only
- Can mix recording, replaying and traditional execution concurrently

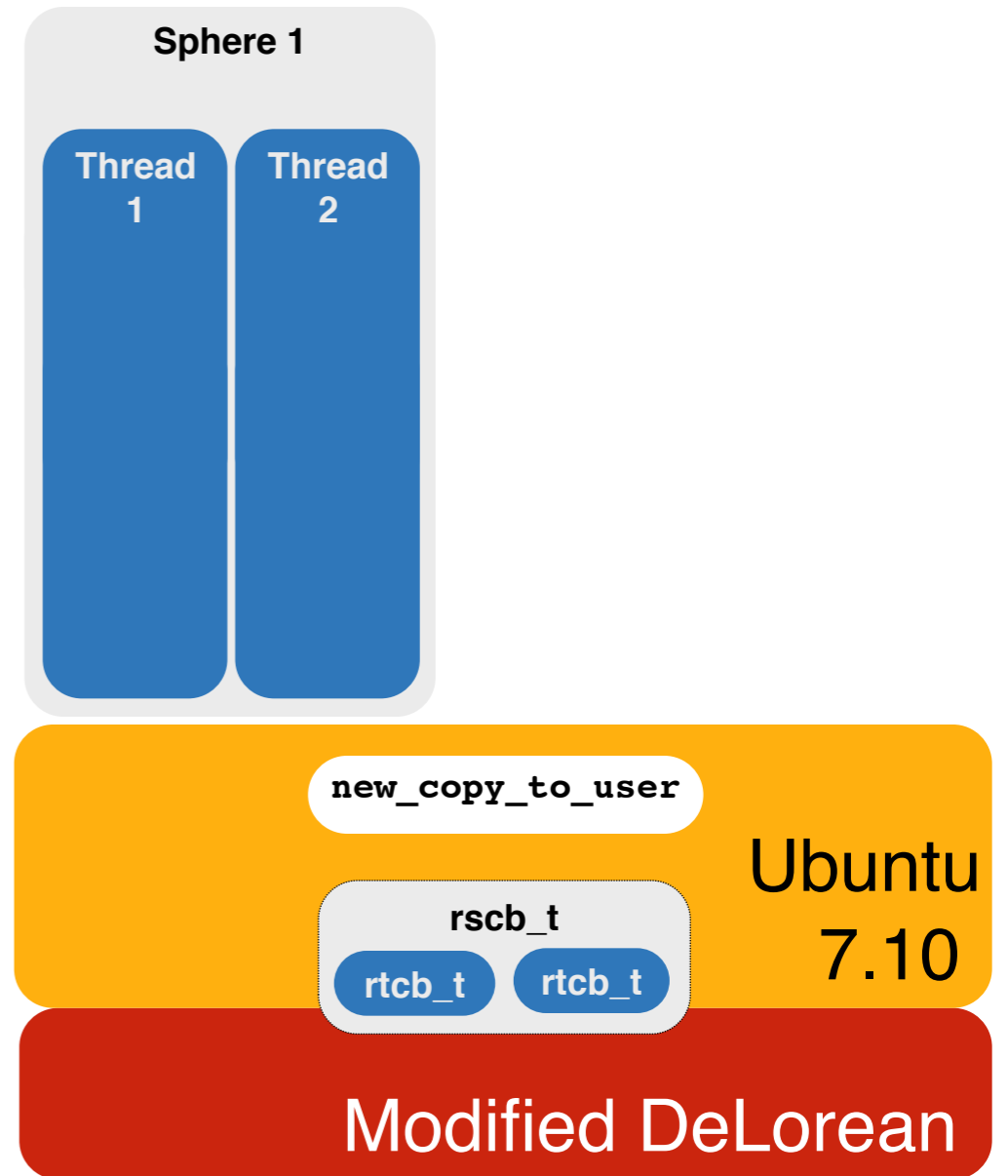


CapoOne: Linux Kernel Changes

- 2.6.24 kernel
- Added new structures:
 - `rscb_t`: stores HW-level sphere context
 - `rtcb_t`: stores per-thread sphere info
- New `copy_to_user`:
 - Saves values before copying them to sphere
 - Deterministic: ensures same chunks are created during replay

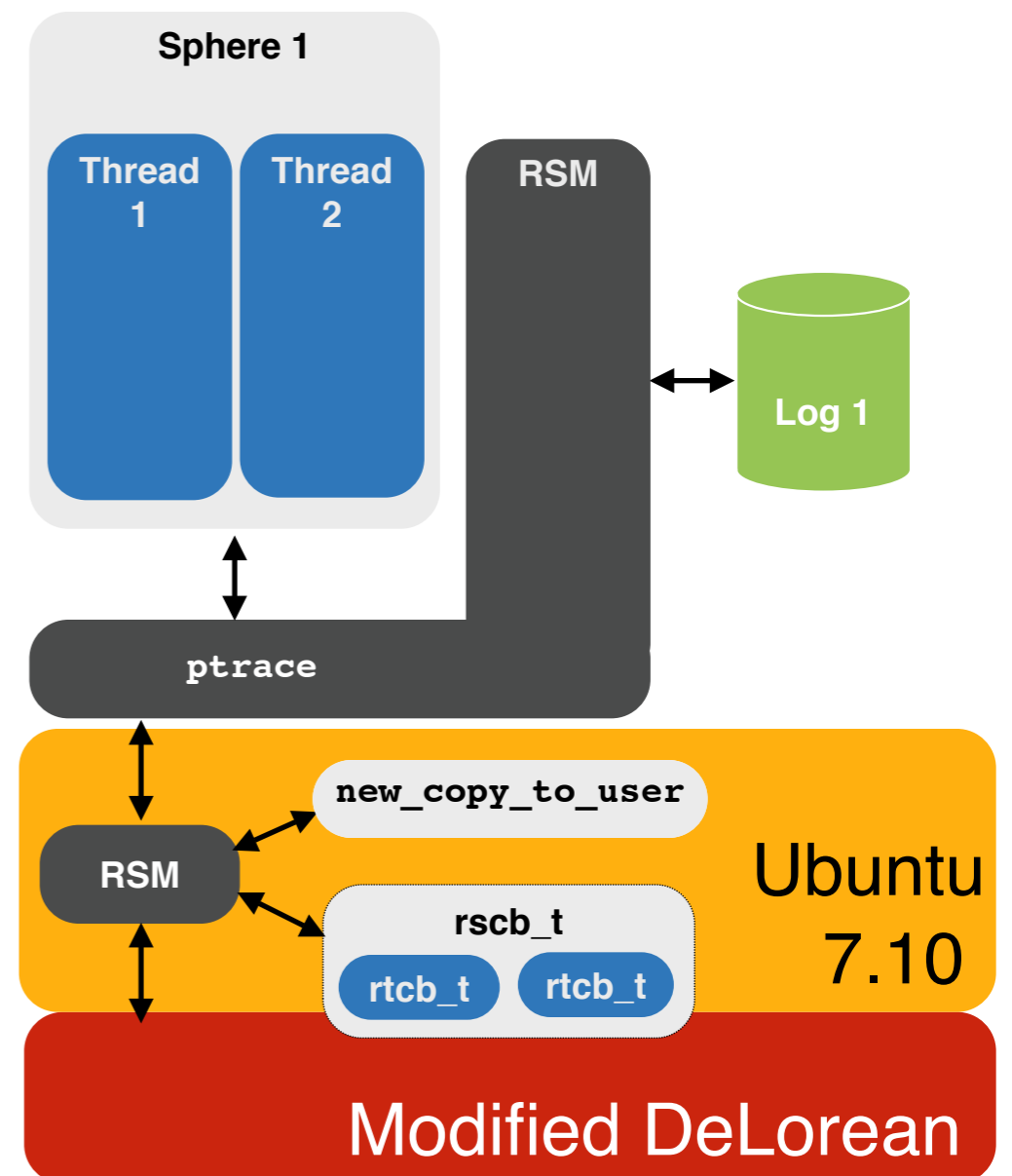


CapoOne: RSM Implementation

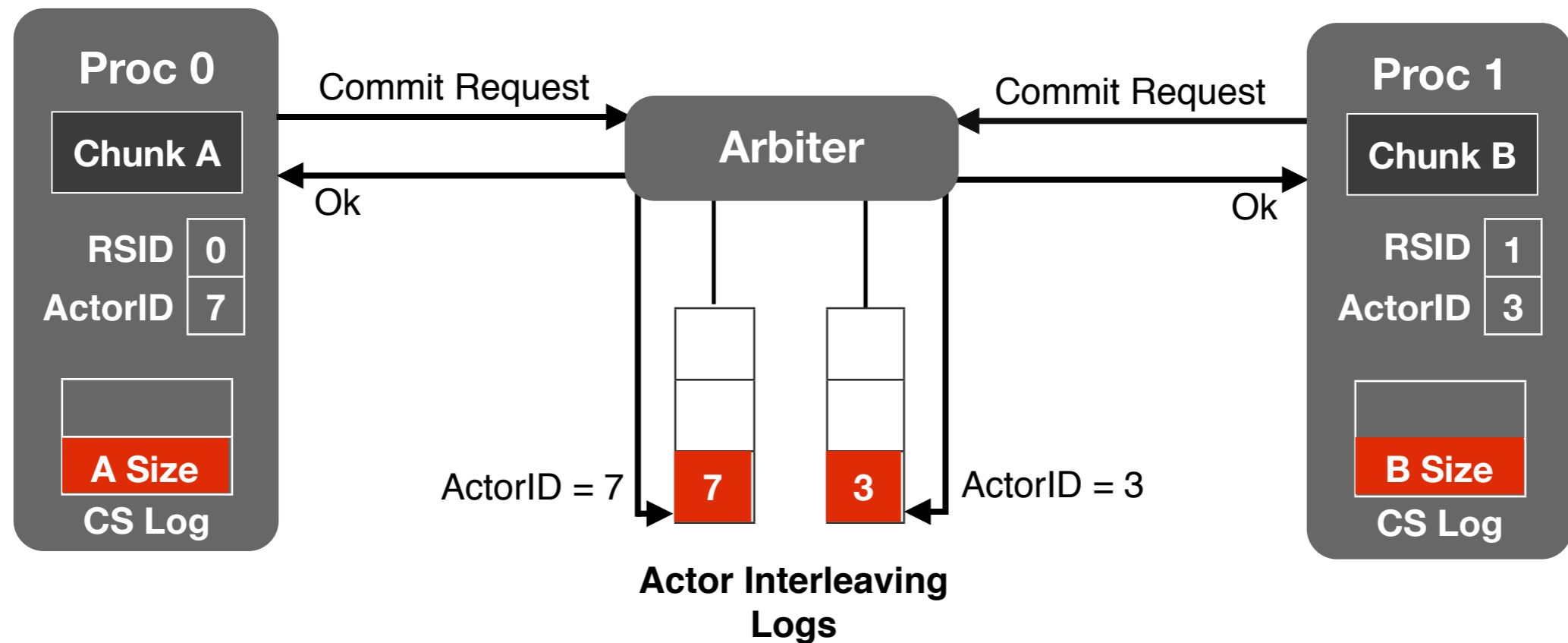


CapoOne: RSM Implementation

- Components in both kernel- and userland
- User-level RSM based on `ptrace`
 - Interposes actor's system calls
 - Stores/Retrieves logs to/from disk
 - Ensures deterministic signal delivery
- Kernel-level RSM initializes DeLorean
 - Manages `rscb_t` and `rtcb_t` structs
 - Collects `new_copy_to_user` logs
 - Inserts `new_copy_to_user` into sphere



CapoOne: Basic HW Operation



- Chunks only have instructions from one application (or the kernel)
- Two new per-processor registers: RSID, ActorID
- Arbiter now supports concurrent spheres
 - Manages an Actor Interleaving Log for each of them



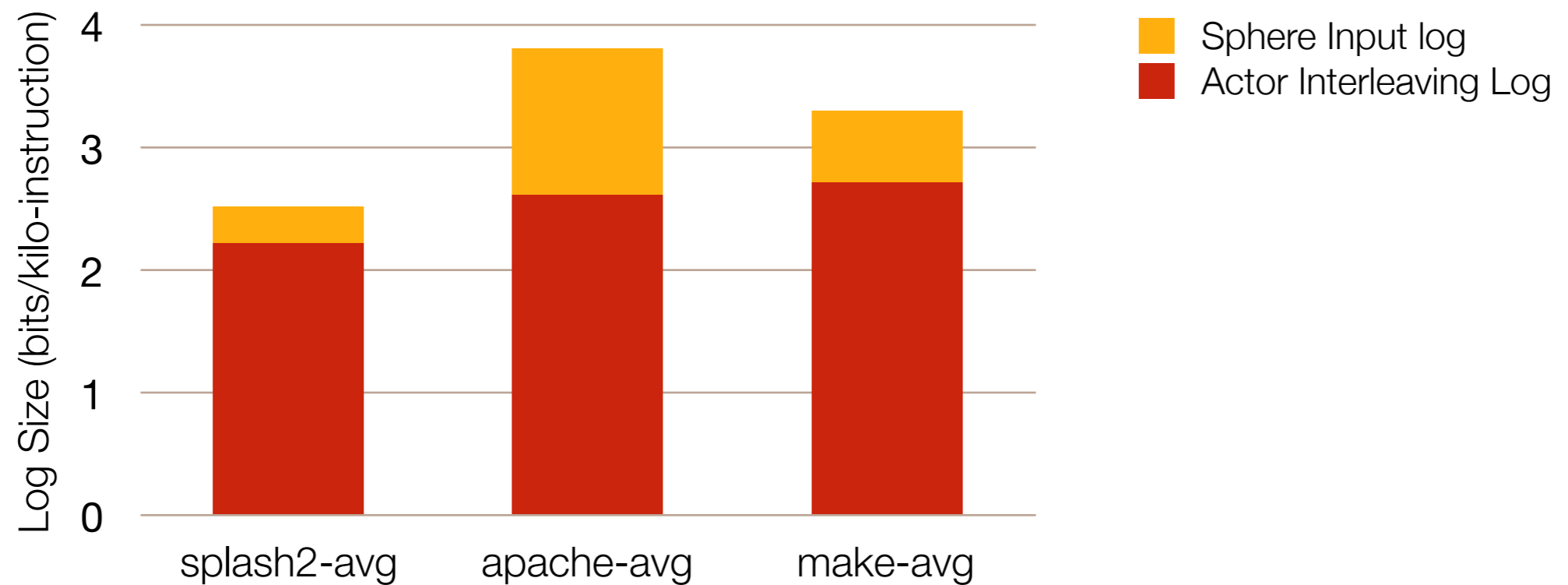
Evaluating Capo

- ■ Two environments:
 - ■ Simulated: SIMICS-based simulator.
 - ■ SMP 4 x86 processors, 2 GHz with DeLorean components
 - ■ Ubuntu 7.10 with RSM
 - ■ Real-hardware: 4-Core Intel Core 2 Quad running at 2.5 GHz
 - ■ Ubuntu 7.10 with RSM
 - ■ No DeLorean HW

- ■ Two sets of benchmarks:
 - ■ Scientific Benchmarks SPLASH-2
 - ■ System benchmarks: Apache, Compilation



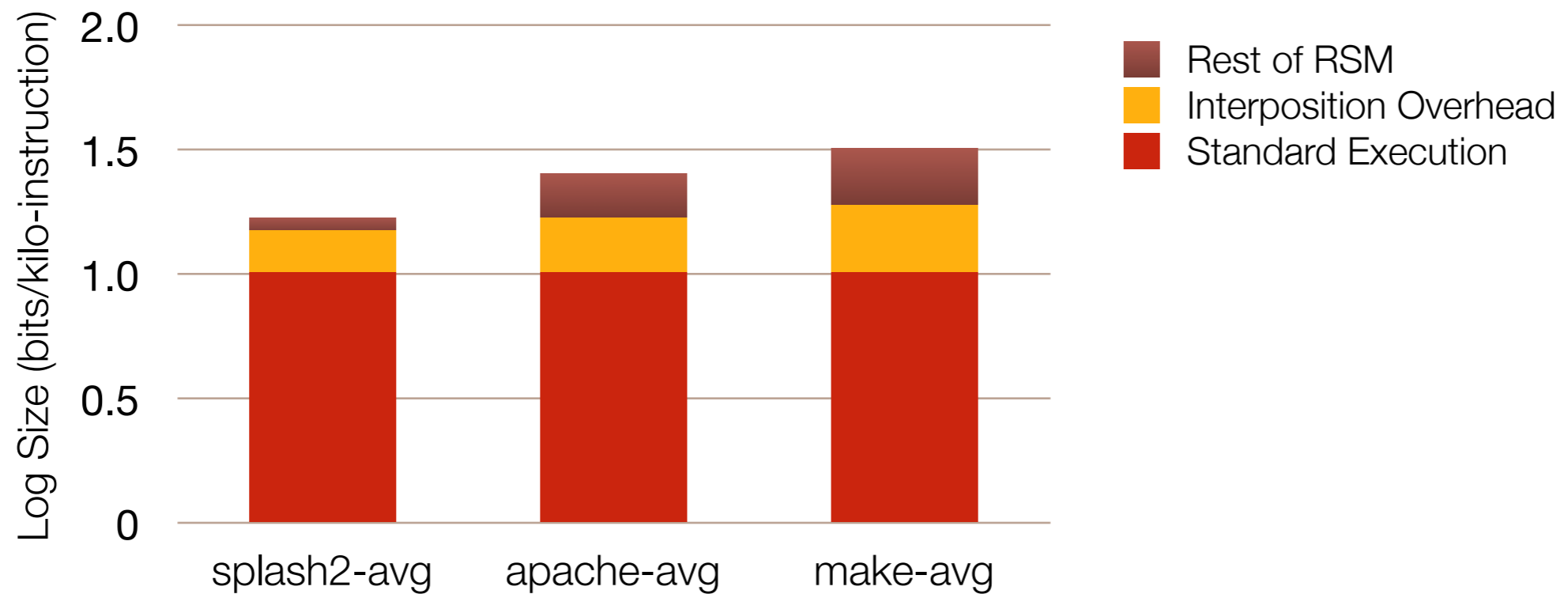
Log Size



- Actor interleaving log is slightly bigger than DeLorean's Nano
 - System calls, page faults, exceptions generate smaller chunks
- Sphere Input Log increases logging requirements only modestly



Performance Overhead



- Moderate overhead: 21% for splash2 and 41% average for system apps
- Most of the overhead comes from interposition: `ptrace`
- Separate spheres don't induce overhead on each other (see paper)



Conclusions

- ■ DeLorean is a very efficient HW-based deterministic replay scheme
 - ■ Provides High speed execution and replay with minuscule log

- ■ Capro is a promising set of abstractions for practical HW-Assisted deterministic replay
 - ■ The Replay Sphere separates SW and HW responsibilities

- ■ Combining HW and SW replay provides flexibility and performance
 - ■ Naive approach does not work, subtle and fundamental interactions
 - ■ Show how you can make this hybrid approach work



Question?

