

rr: A Deterministic Record/Replay Framework for Parallel Programs

Albert Noll
Andreas Gal
Robert O'Callahan

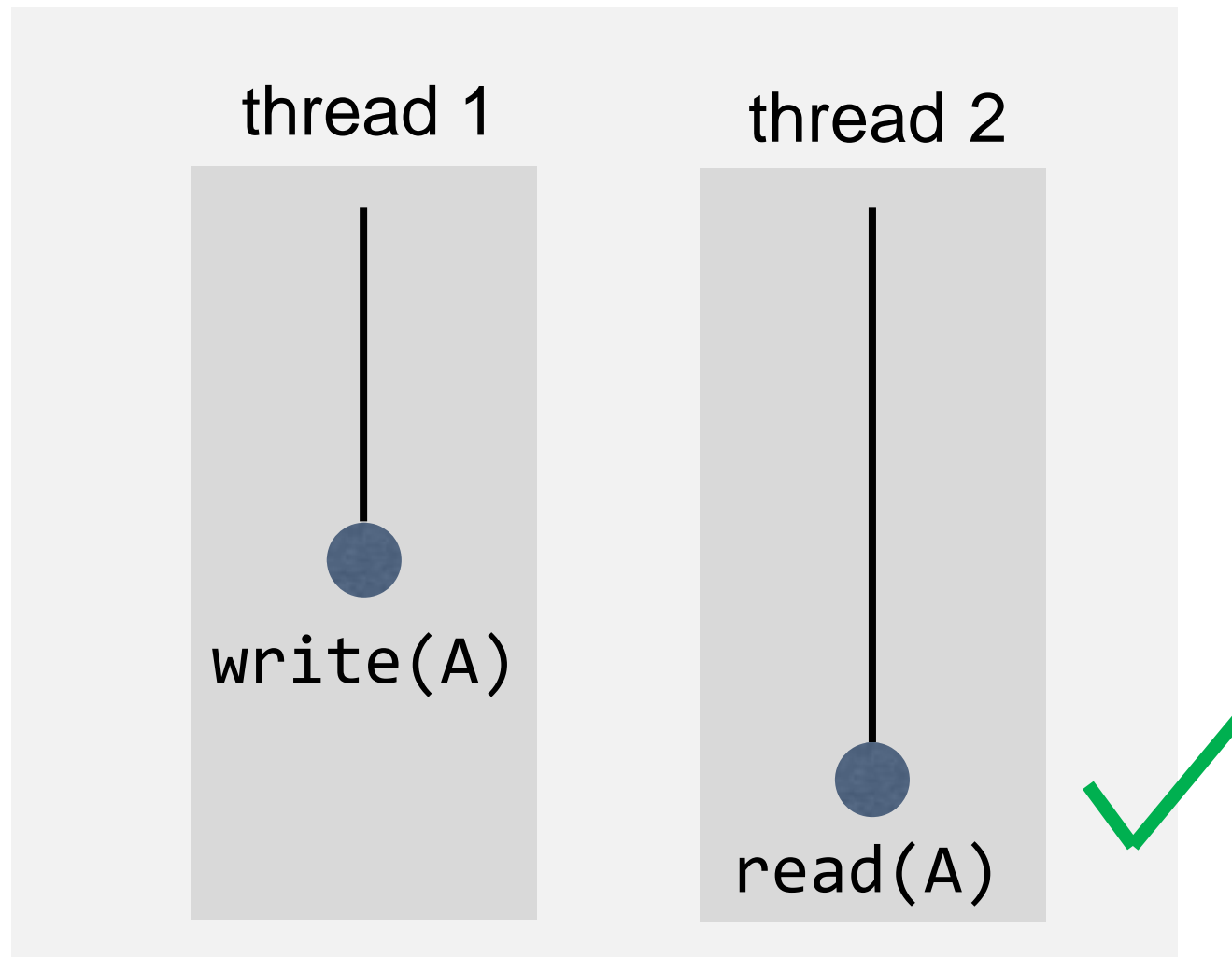
Motivation

- Software is buggy
- Program execution is non-deterministic
- Reproducing bugs is often difficult

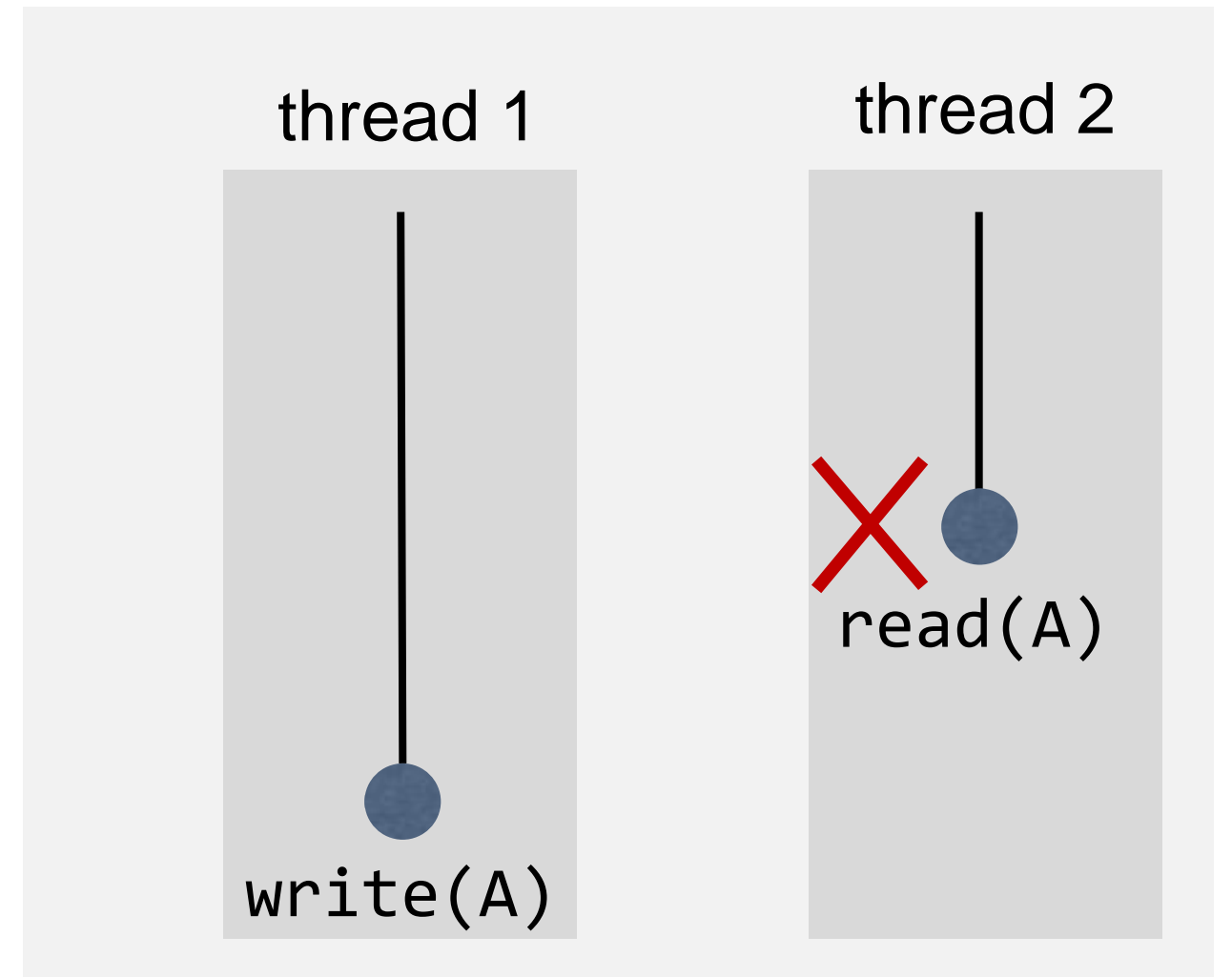
Example

Initially: $A = \text{null}$

Execution 1



Execution 2



Challenges

- Record: capture all sources of non-determinism
- Replay: construct identical program execution
- Low overhead: execution time and memory
 - Recording hours/days
- “Out of the box” components
 - No source code extensions, special hardware, libraries, etc.

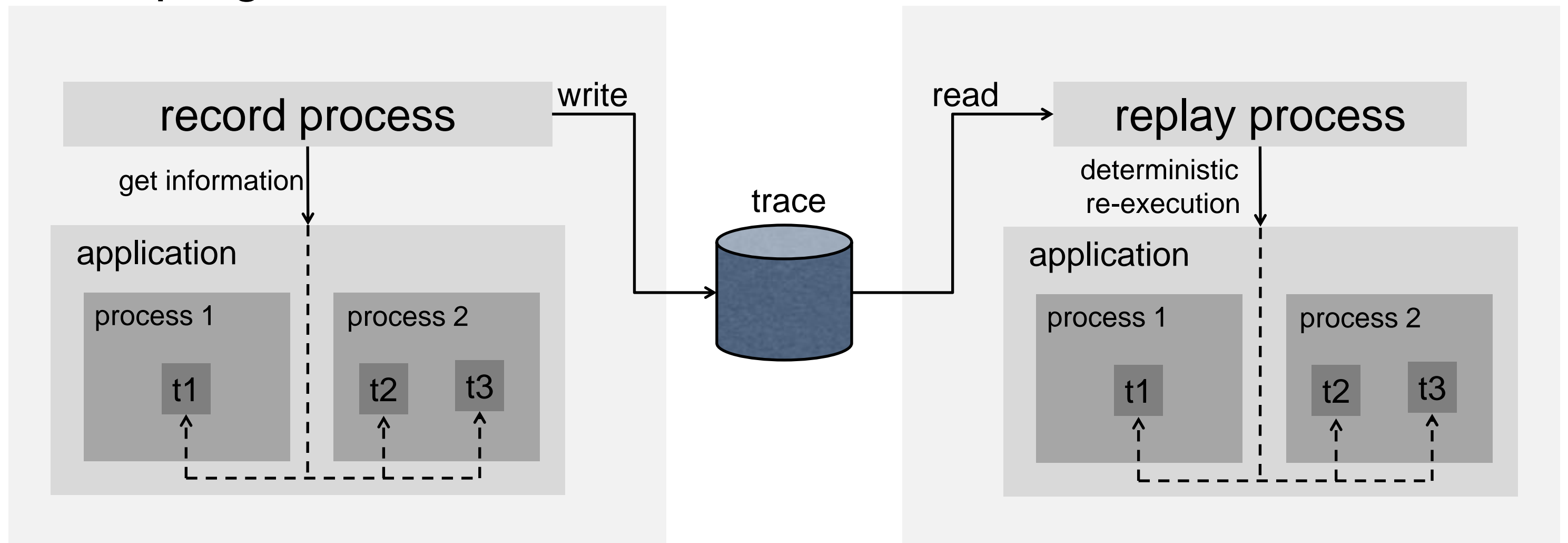
Agenda

- System Overview
- Eliminate sources of non-determinism
- Implementation
- Preliminary performance results
- Q & A

System Overview

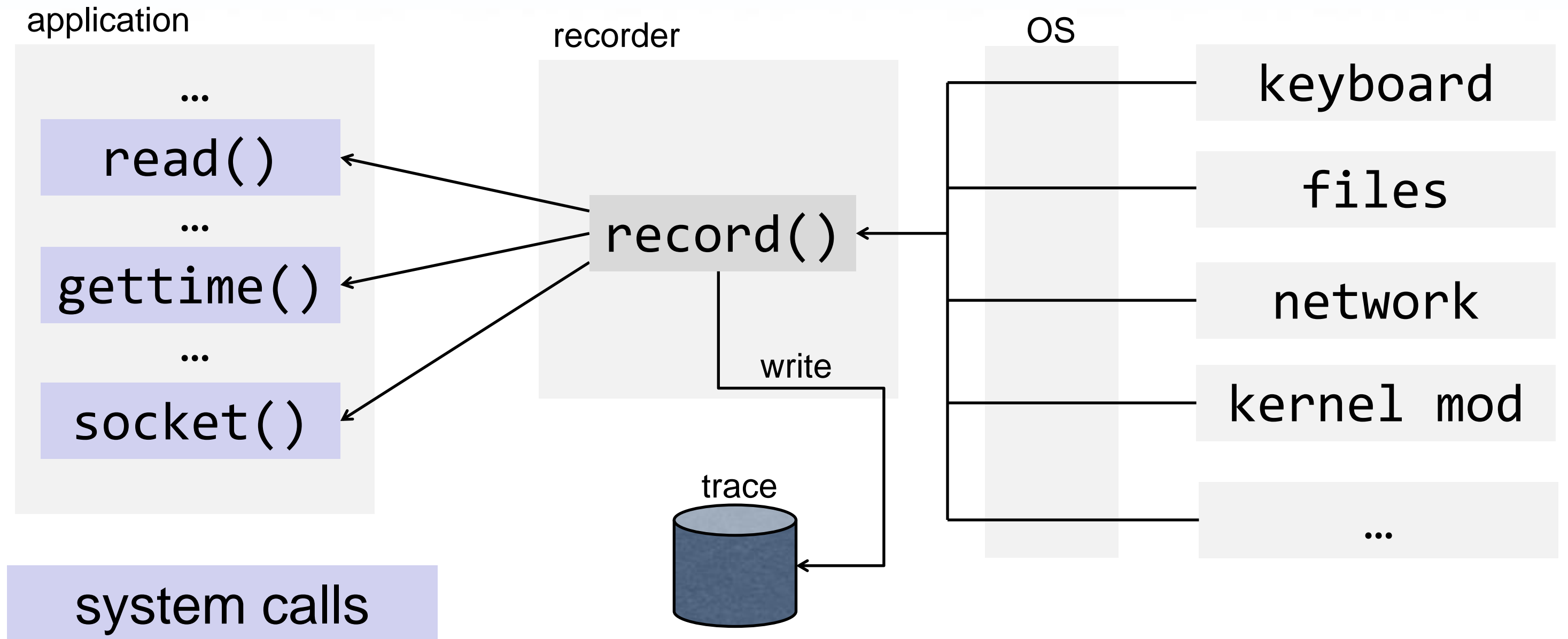
program record

program replay

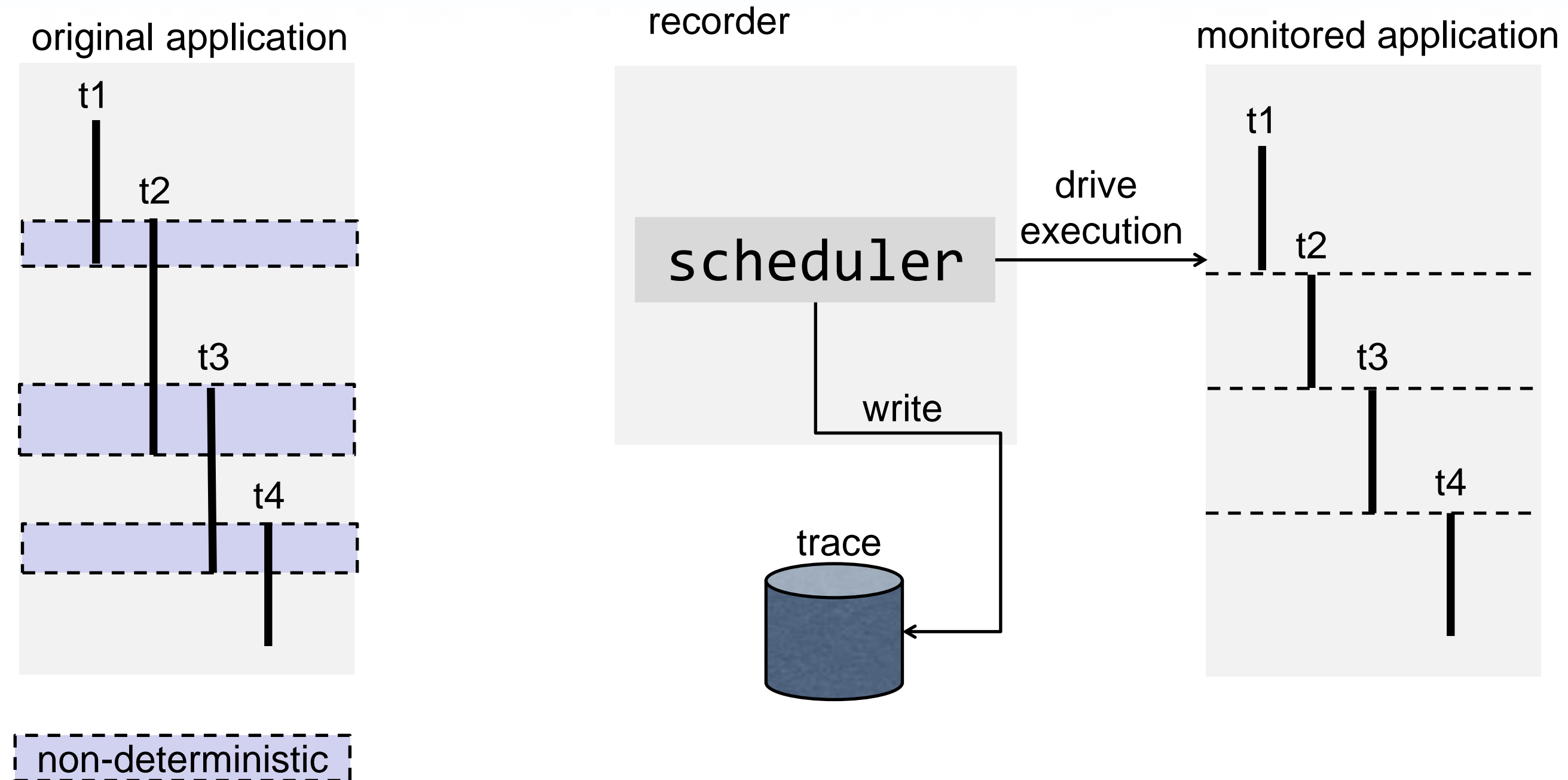


Eliminating non-determinism

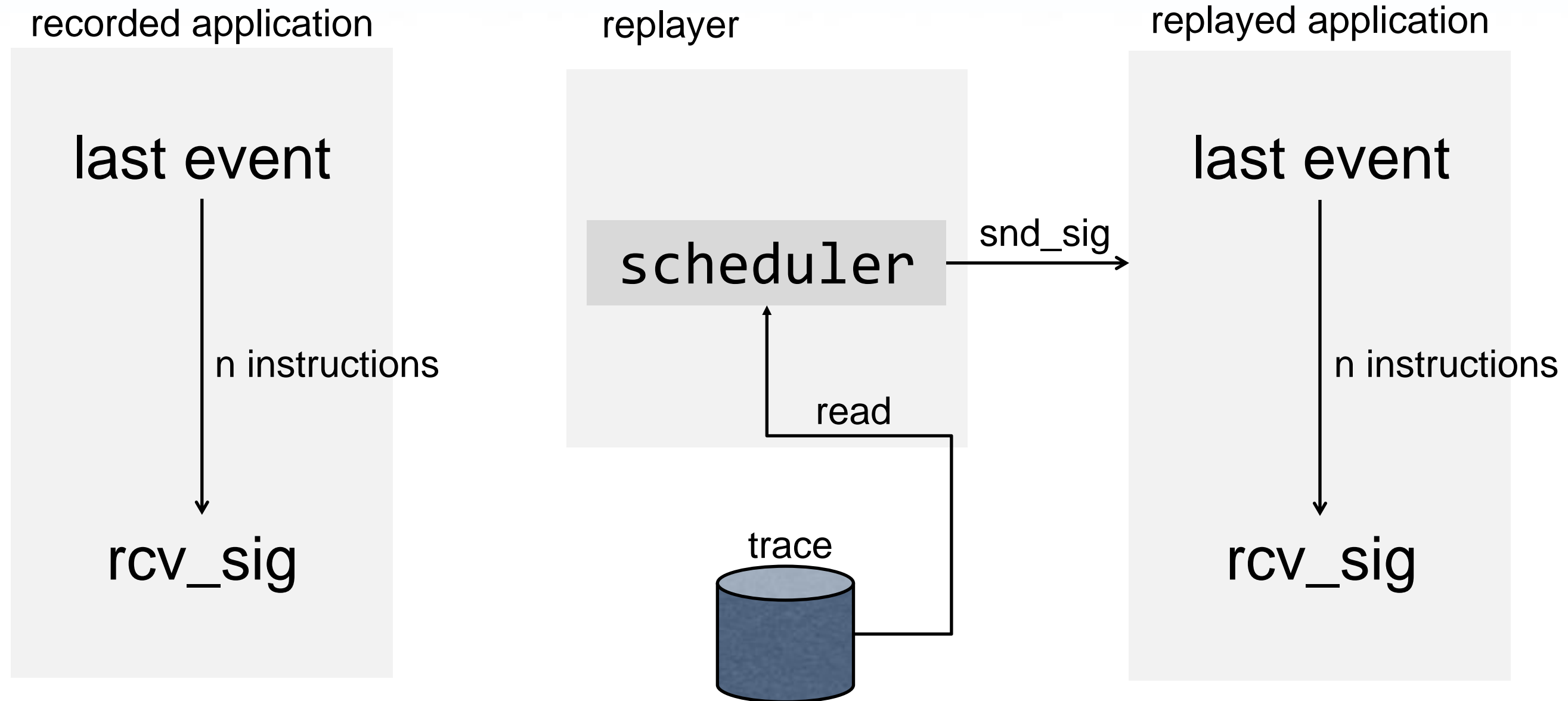
Program Input



Scheduling and Interleaving



Signals



Signals

- Serialize execution and record schedule
- Hardware performance counters
- Recorder: record number of retired instructions
- Replayer:
 - Count-down recorded instructions
 - Send signal at the exact location

Instructions: X86

- `rdstc`: read timestamp counter
 - Can be virtualized using `prctl` system call
- `rdmsr`: read model specific register
 - Must be executed at privilege level 0
- `cuid`: determine processor type and features
 - Determine processor type and features

Address Space Randomization

- Security feature that is enabled by default
- Randomizes memory layout of process
- Feature can be disabled

Implementation

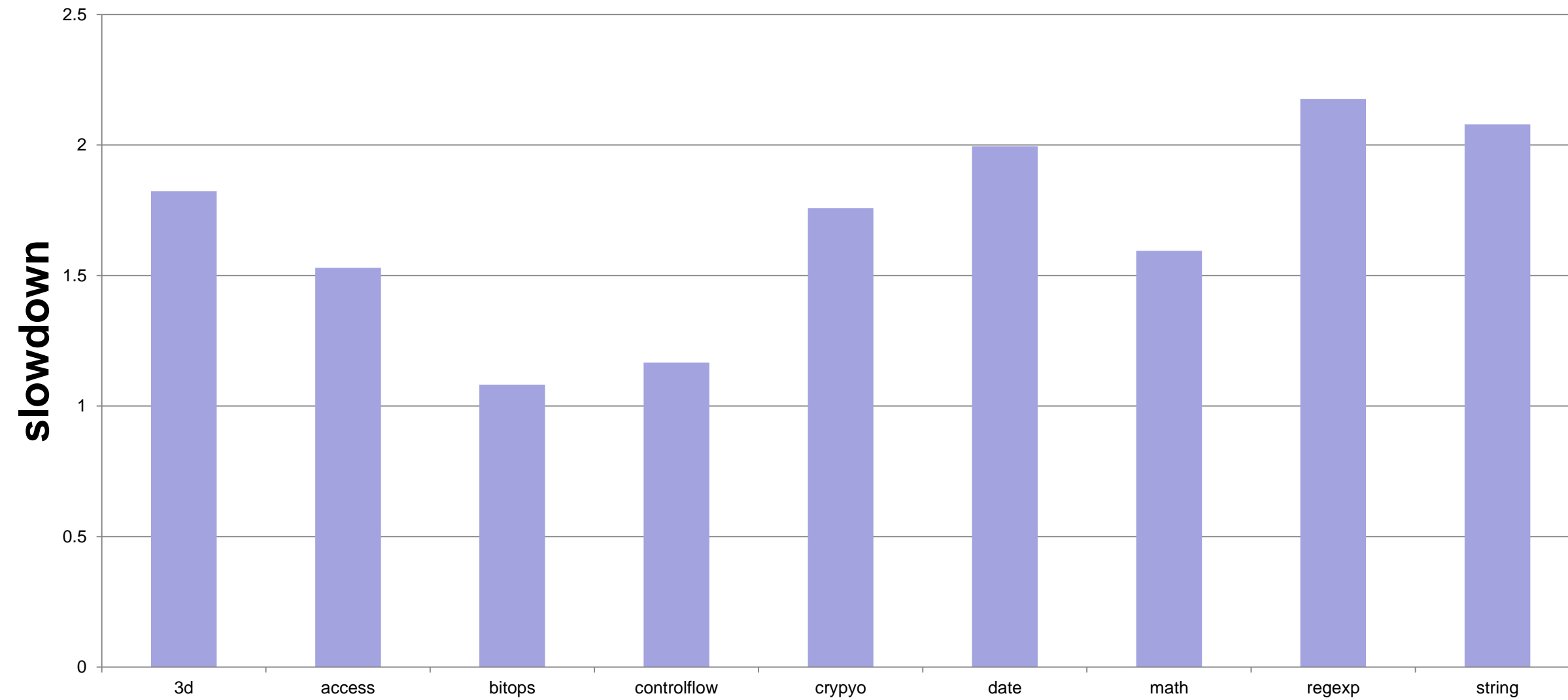
- ptrace application
 - Stop and system call entry/exit
 - Intercept signals
- Hardware performance counters
 - Deterministic signal replay
- Checkpointing
 - clone + copy-on-write

Experimental Evaluation

- Record Firefox 6
- Ubuntu with kernel 3.0 rc6
- Intel Quad-Core Core i7-2600 @ 3.4 GHz
- 4 GB RAM

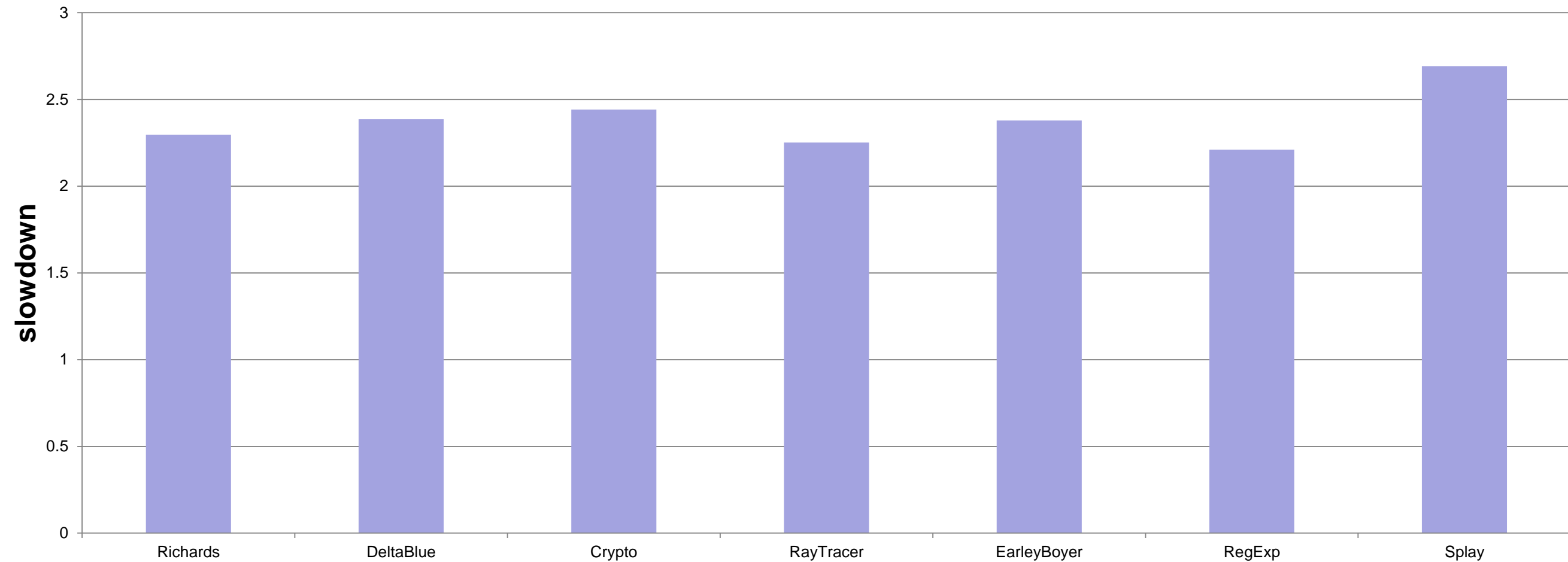
Performance Results

SunSpider



Performance Results

V8 Benchmark Suite



Questions?