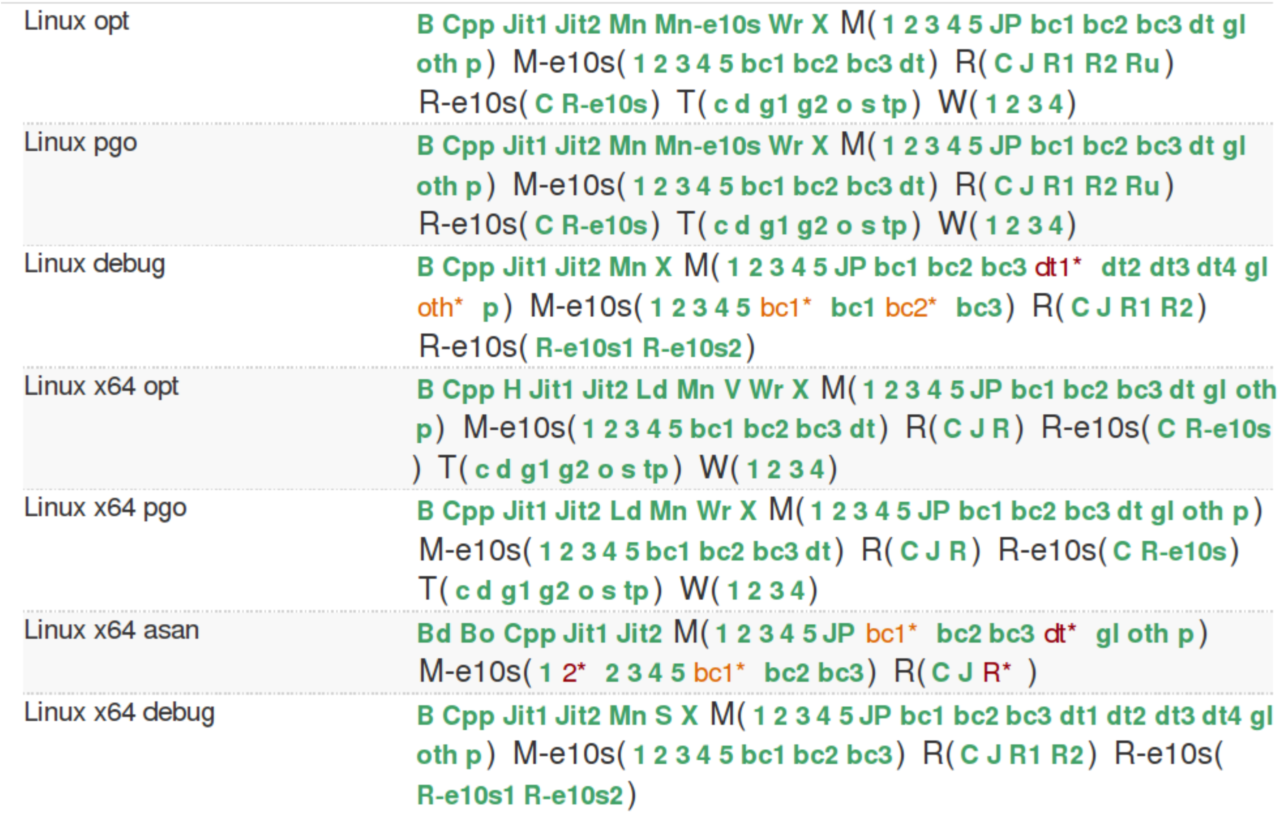
Practical Record And Replay Debugging With rr

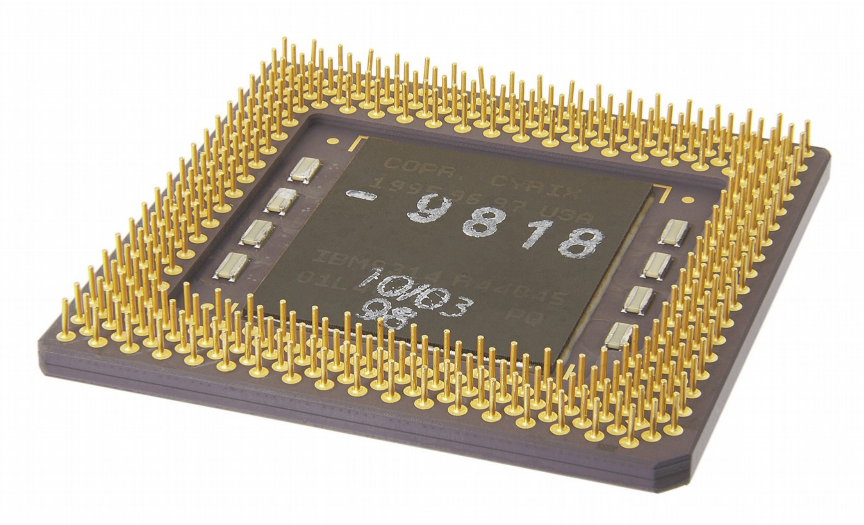
Robert O'Callahan

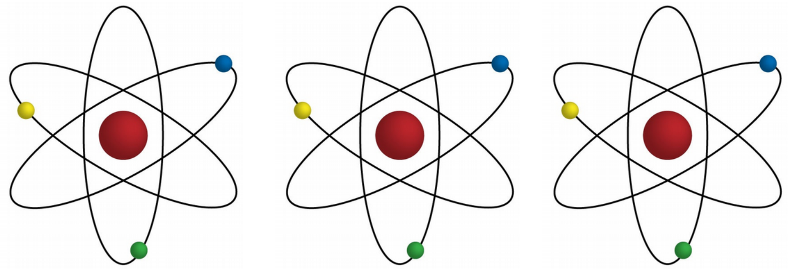
Debugging nondeterminism



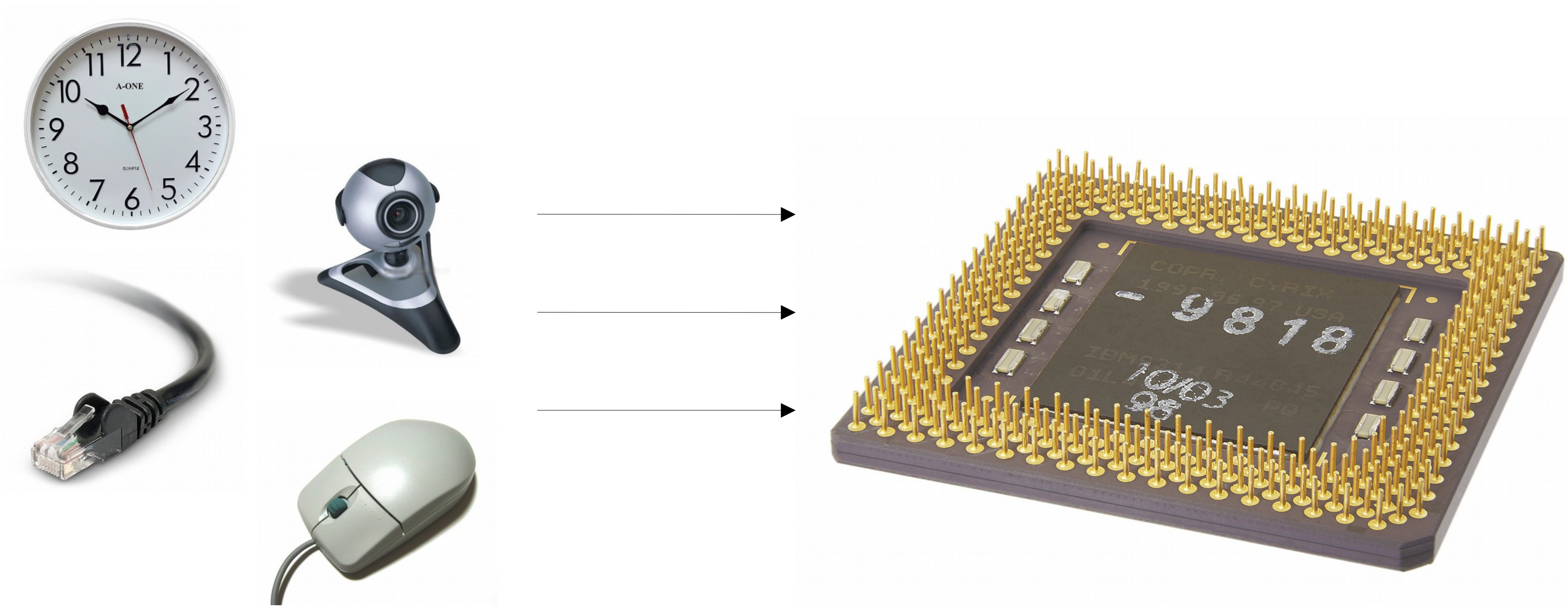
Linux opt	B Cpp Jit1 Jit2 Mn Mn-e10s Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R1 R2 Ru) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)
Linux pgo	B Cpp Jit1 Jit2 Mn Mn-e10s Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R1 R2 Ru) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)
Linux debug	B Cpp Jit1 Jit2 Mn X M(1 2 3 4 5 JP bc1 bc2 bc3 dt1* dt2 dt3 dt4 gl oth* p) M-e10s(1 2 3 4 5 bc1* bc1 bc2* bc3) R(C J R1 R2) R-e10s(R-e10s1 R-e10s2)
Linux x64 opt	B Cpp H Jit1 Jit2 Ld Mn V Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)
Linux x64 pgo	B Cpp Jit1 Jit2 Ld Mn Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)
Linux x64 asan	Bd Bo Cpp Jit1 Jit2 M(1 2 3 4 5 JP bc1* bc2 bc3 dt* gl oth p) M-e10s(1 2* 2 3 4 5 bc1* bc2 bc3) R(C J R*)
Linux x64 debug	B Cpp Jit1 Jit2 Mn S X M(1 2 3 4 5 JP bc1 bc2 bc3 dt1 dt2 dt3 dt4 gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3) R(C J R1 R2) R-e10s(R-e10s1 R-e10s2)

Deterministic hardware

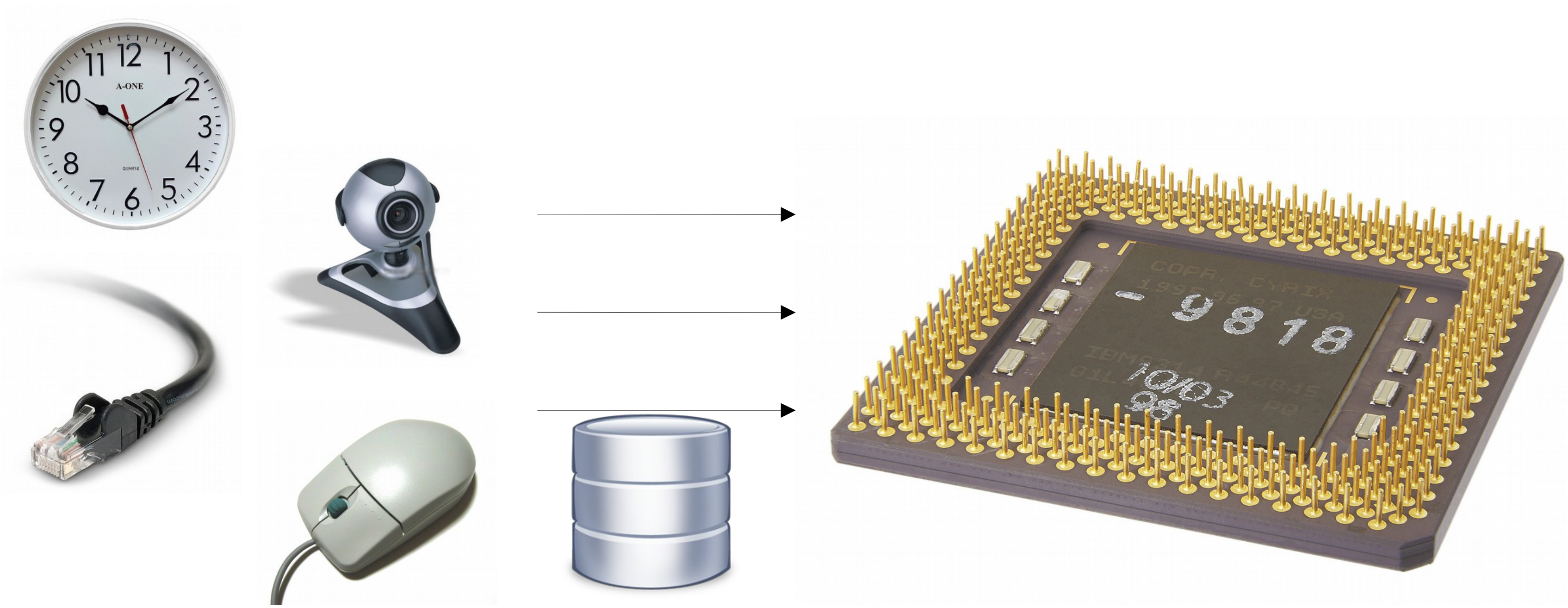




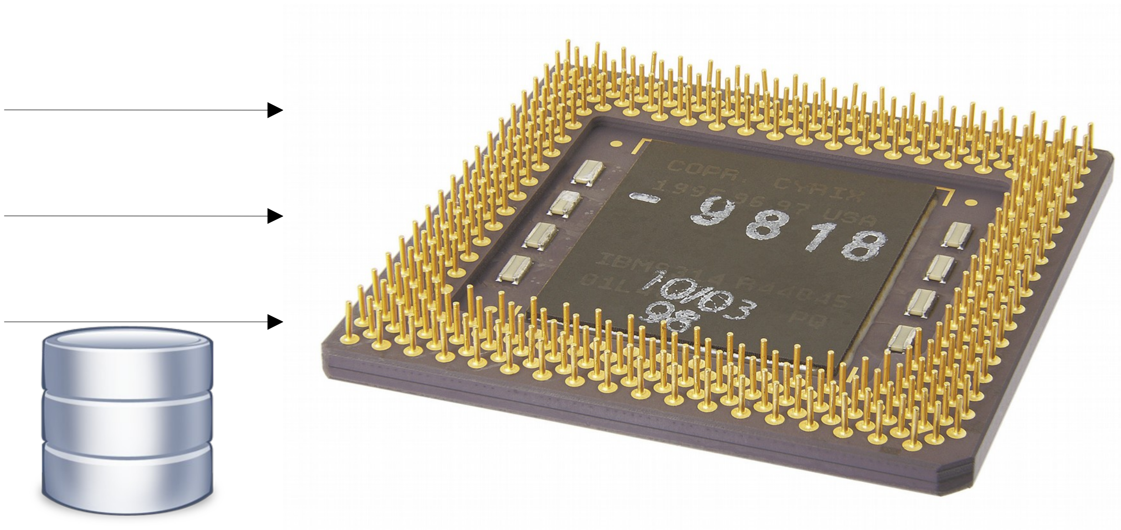
Sources of nondeterminism



Record inputs



Replay execution



“Old idea”

PinPlay

ReVirt

Nirvana

ReSpec

Jockey

Chronomancer

ODR

PANDA

Scribe

Echo

CLAP

FlashBack

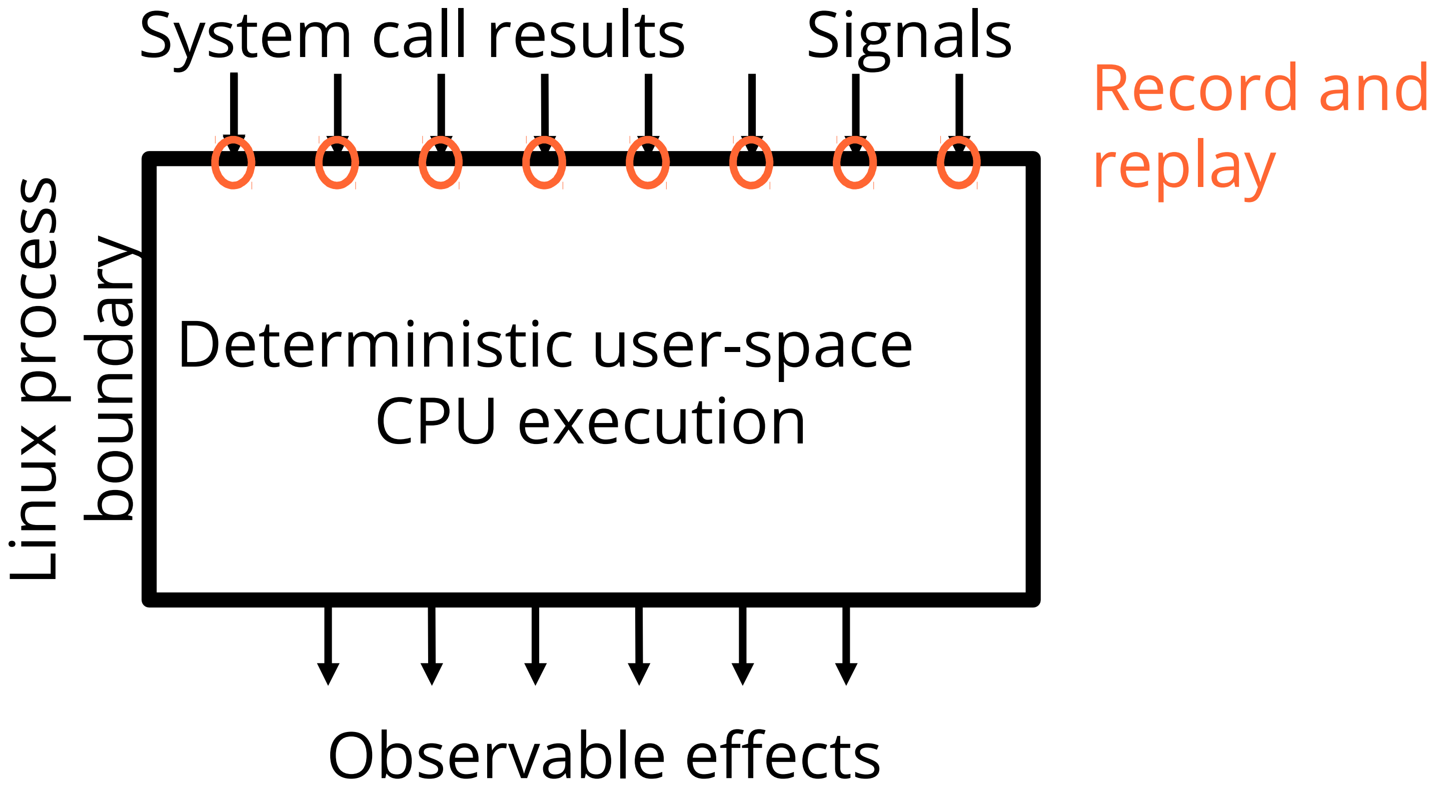
QuickRec

ReTrace

rr goals

- Easy to deploy: stock hardware, OS
- Low overhead
- Works on Firefox
- Small investment

rr design


System call results
Signals
Record and replay
Linux process boundary
Deterministic user-space CPU execution
Observable effects

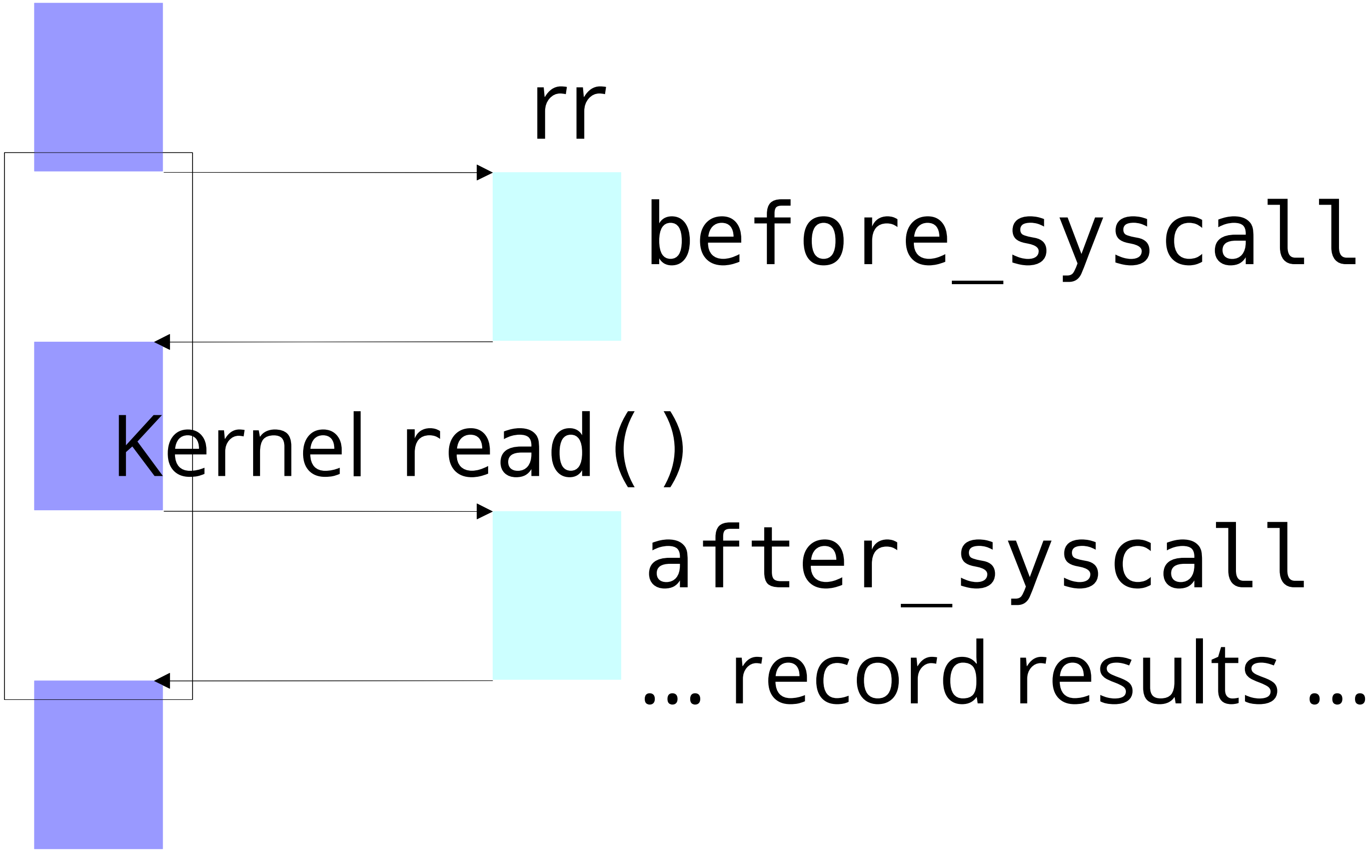
No code instrumentation

```
mov EDX, tls.ebp
mov ECX, tls
call MemReadCallback
mov EAX, [EDX]
```

Use modern HW/OS features

System call results	`ptrace`
Signals	`ptrace`
Shared memory data races	Limit to single core
Asynchronous event timing	HW performance counters
Trap on a subset of system calls	`seccomp-bpf`
Notification when system call blocks in the kernel	`DESCHED` perf events
Cheap block copies	`FIOCLONERANGE`

ptrace



rr

before_syscall

Kernel read()

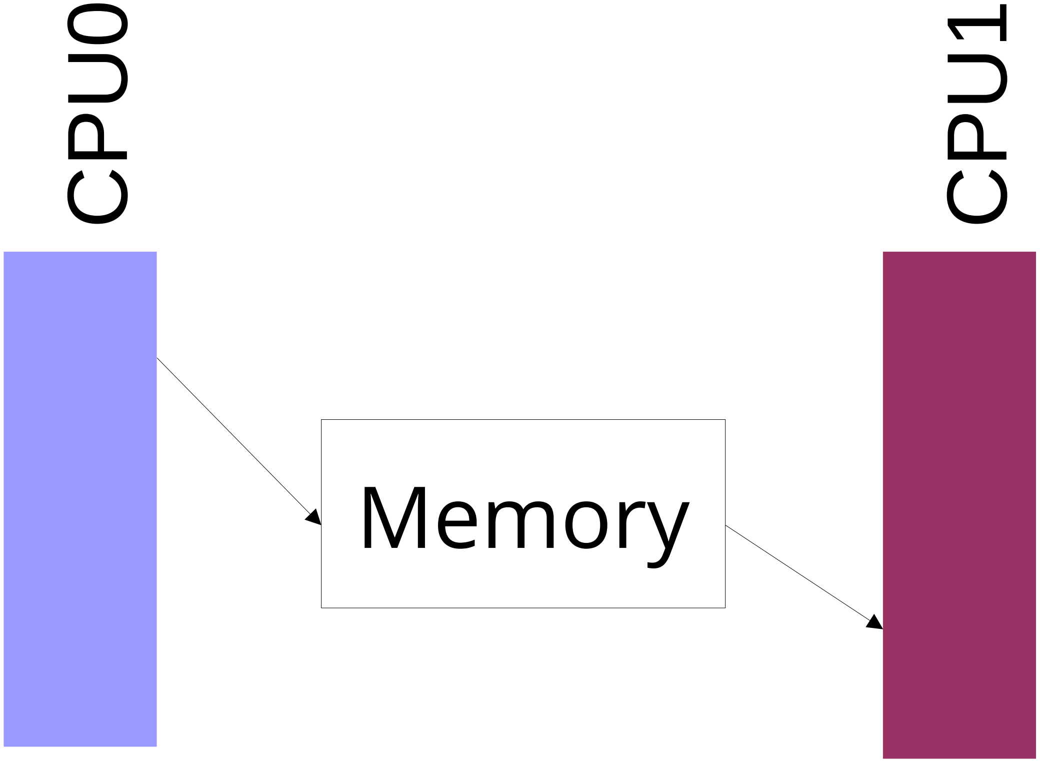
after_syscall

... record results ...

Use modern HW/OS features

System call results	`ptrace`
Signals	`ptrace`
Shared memory data races	Limit to single core
Asynchronous event timing	HW performance counters
Trap on a subset of system calls	`seccomp-bpf`
Notification when system call blocks in the kernel	`DESCHED` perf events
Cheap block copies	`FIOCLONERANGE`

Data races

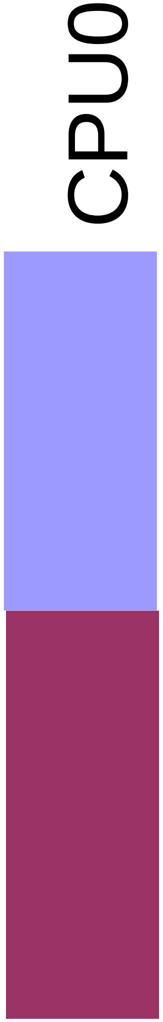


CPU0

Memory

CPU1

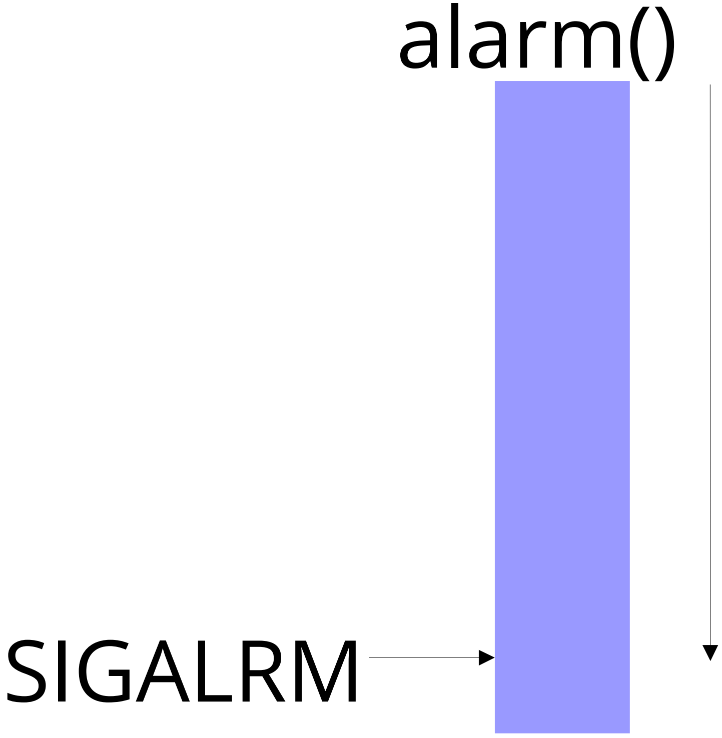
Data races



Use modern HW/OS features

System call results	`ptrace`
Signals	`ptrace`
Shared memory data races	Limit to single core
Asynchronous event timing	HW performance counters
Trap on a subset of system calls	`seccomp-bpf`
Notification when system call blocks in the kernel	`DESCHED` perf events
Cheap block copies	`FIOCLONERANGE`

Event timing: HW perf counters



Measure progress

~~Instructions executed!~~

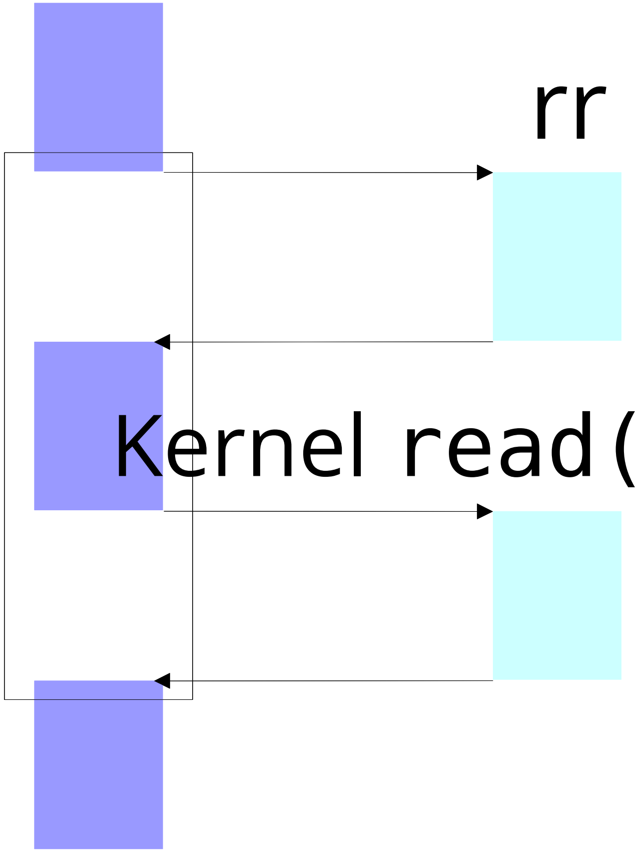
Retired conditional branches (Intel)

Zero overhead

Use modern HW/OS features

System call results	`ptrace`
Signals	`ptrace`
Shared memory data races	Limit to single core
Asynchronous event timing	HW performance counters
Trap on a subset of system calls	`seccomp-bpf`
Notification when system call blocks in the kernel	`DESCHED` perf events
Cheap block copies	`FIOCLONERANGE`

Accelerating system calls



rr

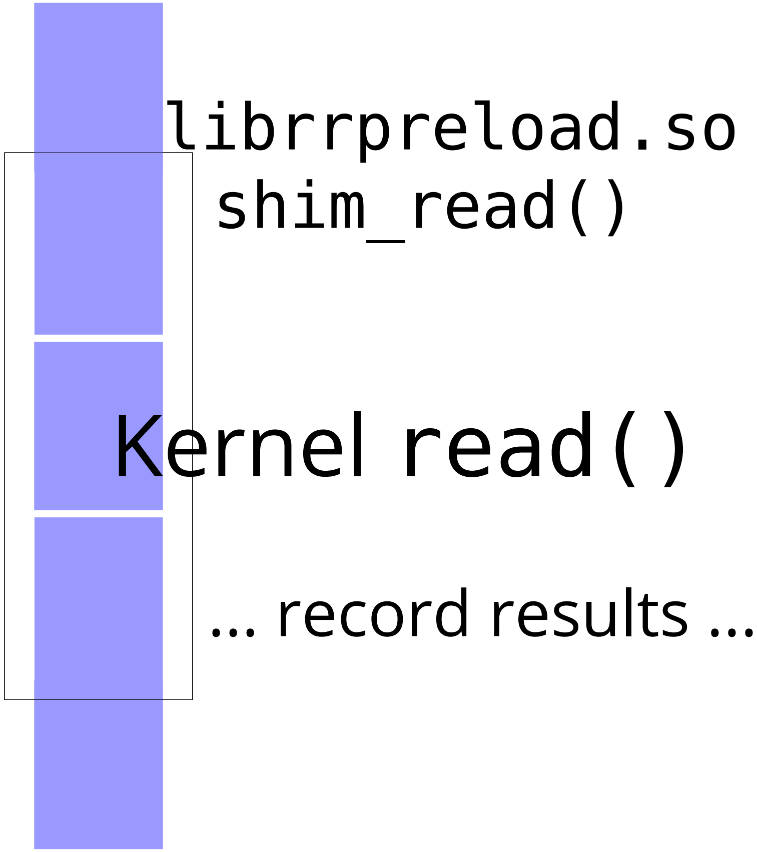
before_syscall

Kernel read()

after_syscall

... record results ...

Avoid context switches



`librrpreload.so`

`shim_read()`

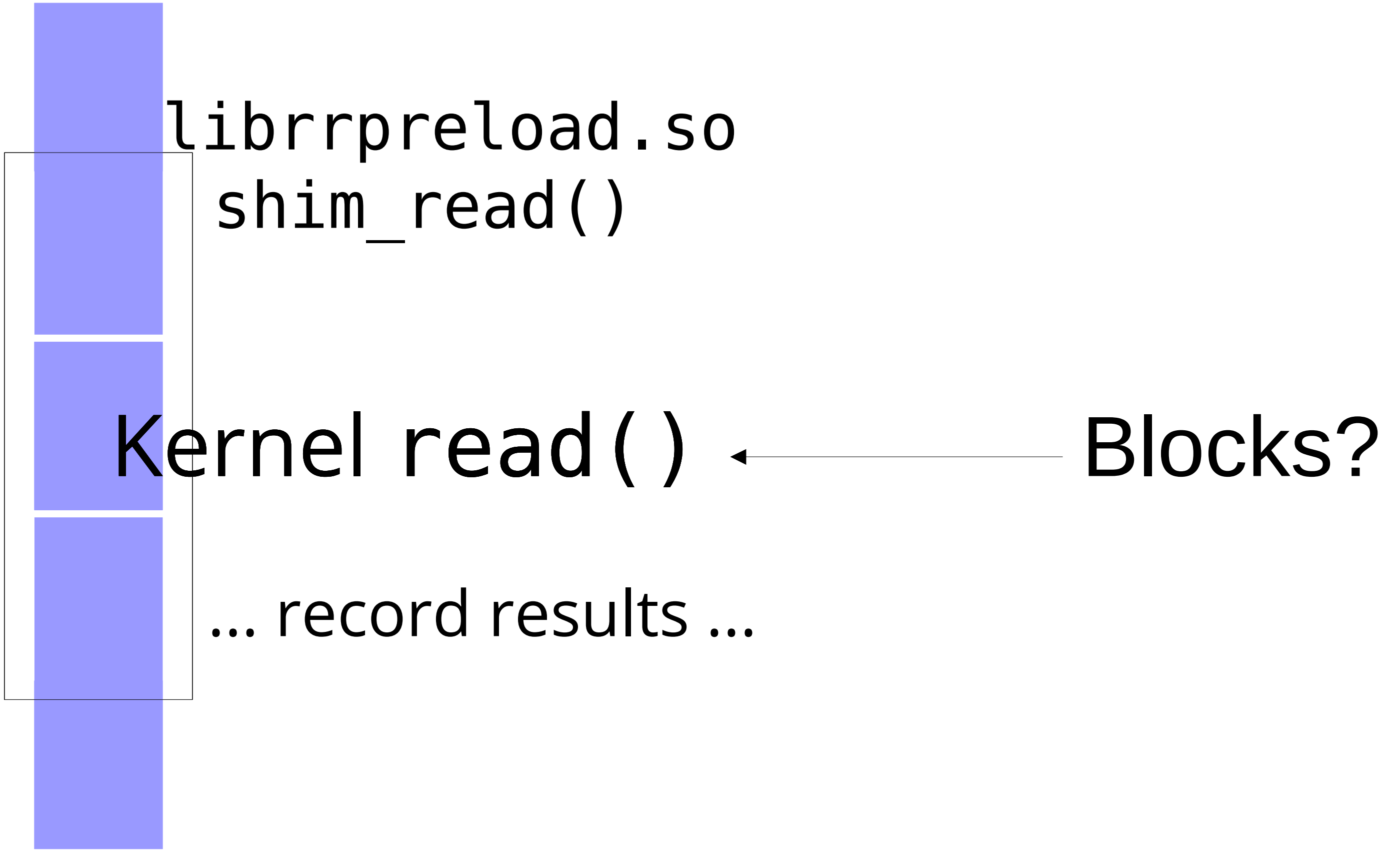
Kernel `read()`

... record results ...

Suppress `ptrace` trap

Use `seccomp-bpf` predicates

Blocking system calls



`librrpreload.so`

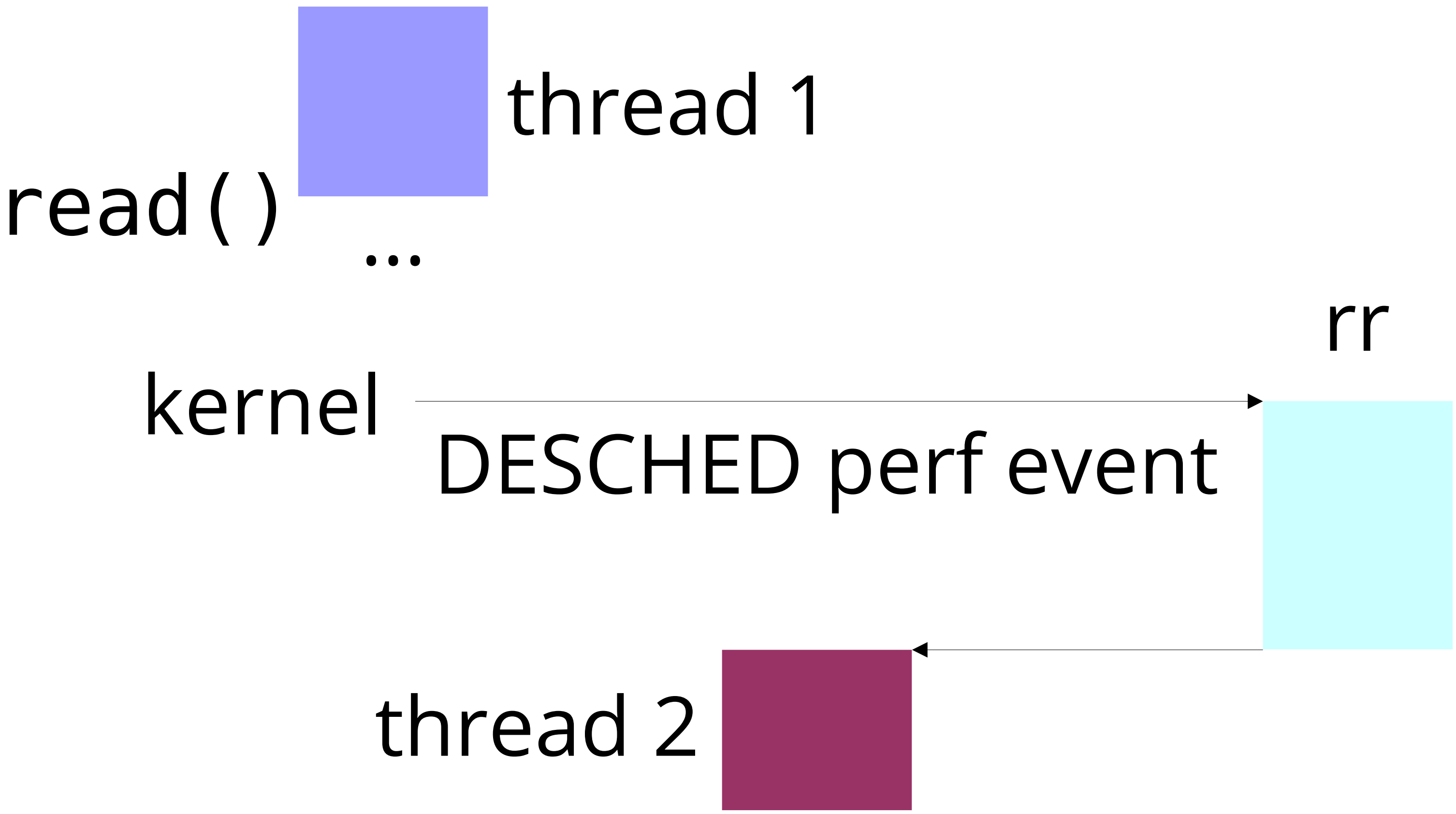
`shim_read()`

Kernel `read()`

Blocks?

... record results ...

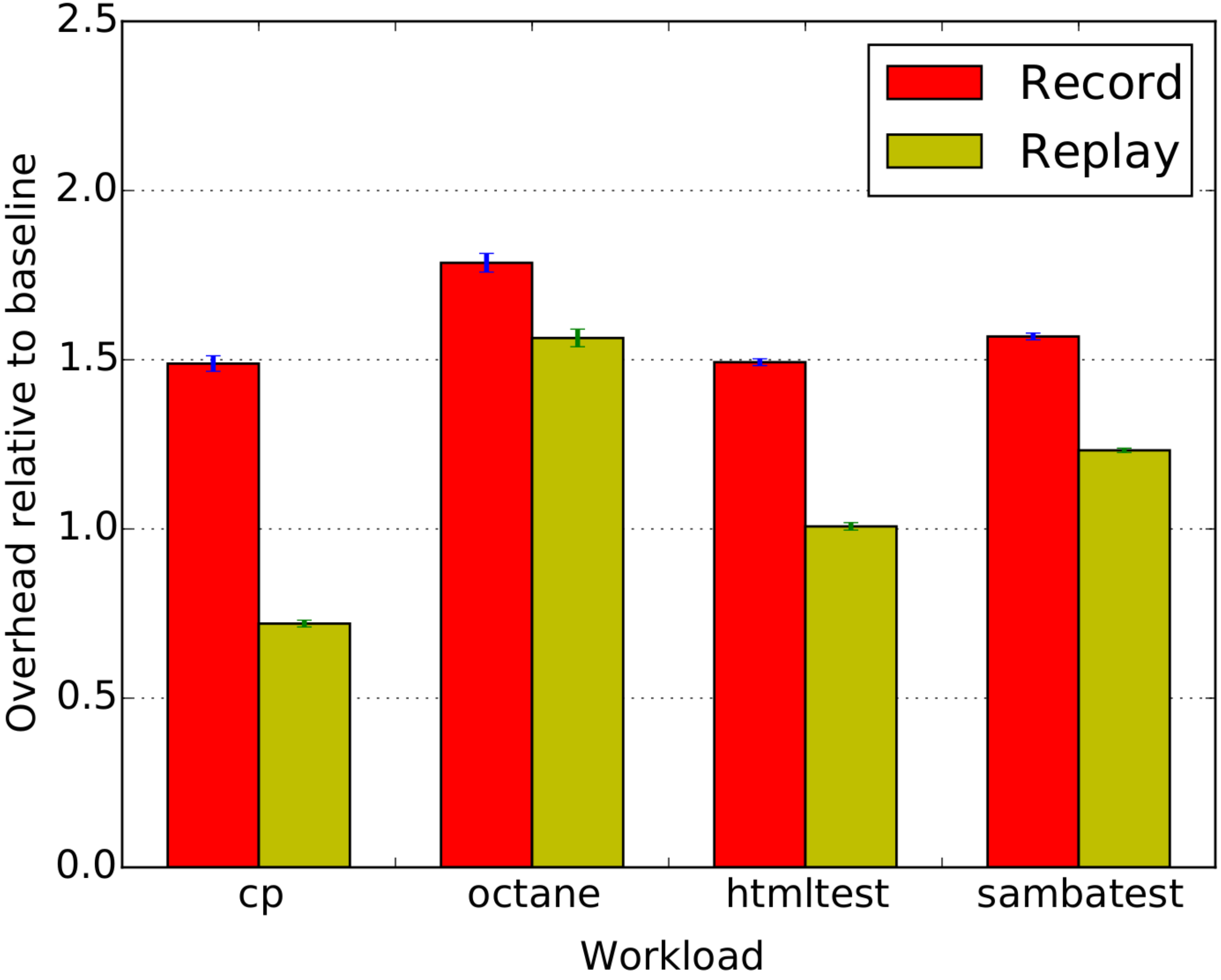
Blocking system calls



Other issues

```
RDTSC
CPUID
RDRAND
XBEGIN/XEND
```

rr Overhead

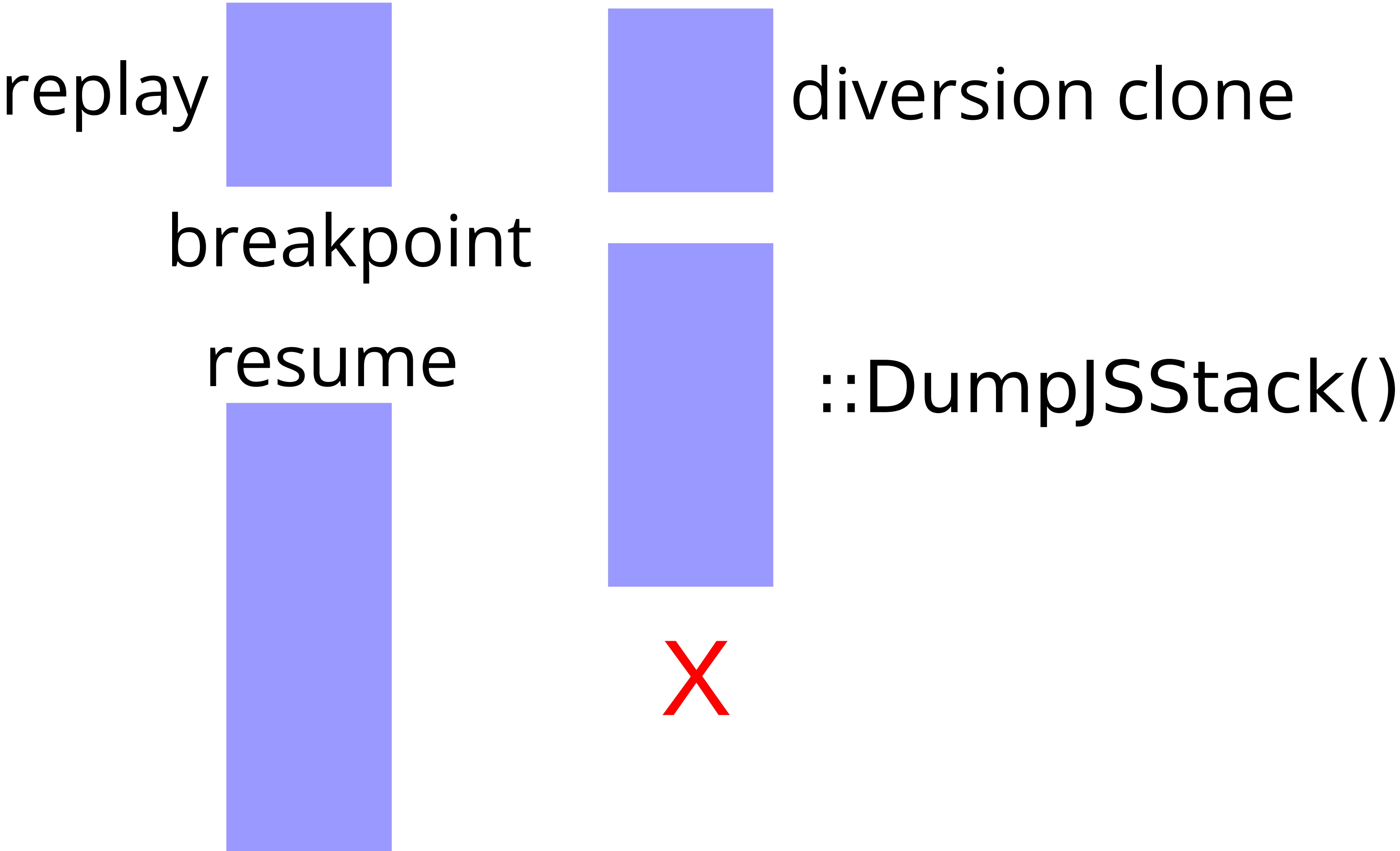

Record
Replay
Overhead relative to baseline
2.5
2.0
1.5
1.0
0.5
0.0
cp
octane
htmltest
sambatest
Workload

Debug with gdb

Running debuggee code

```
(gdb) call ::DumpJSStack()
0 _setMaxHeight()
    ["panelUI.xml":331]
    this = [object XULElement]
1 handleEvent(aEvent = [object
MouseEvent])
    ["panelUI.xml":304]
    this = [object XULElement]
```

Running debuggee code



replay

breakpoint

resume

diversion clone

::DumpJSStack()

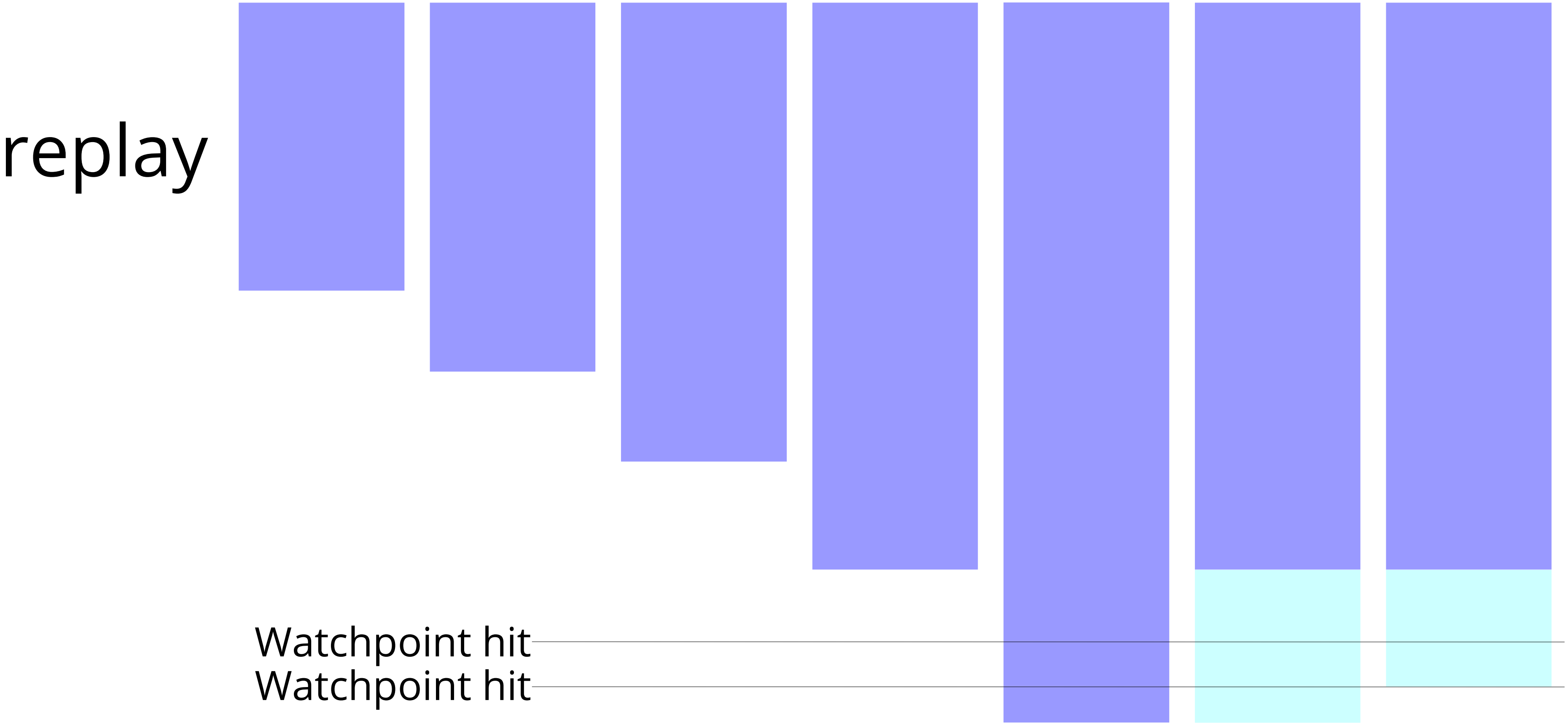
X

Reverse execution

```
(gdb) watch -l mRect.width
(gdb) reverse-continue
nsIFrame::SetRect
(this=0x2aaadd7dbeb0,
aRect=...)
718       mRect = aRect;

(gdb) reverse-next
```

Reverse execution



Results

21:38 **mstange** roc: there's somewhat of a competition going on here at the office about who can use rr the most

21:38 **mstange** roc: it's so good

21:39 **roc** :-)

21:39 **roc** who's using it?

21:39 **mstange** roc: jeff, myself, jeff's interns

21:40 **mstange** roc: and we're telling everybody else to use it whenever we get the chance

Limitations

Single-core

Recording/replaying inter-core data races
→ need HW support :-(
→ need users, to make economic argument

Find bugs in parallel programs
→ evil scheduler (*chaos mode*)

ARM

```
retry:
LDREX r0,[addr]
ADD r0,1
hardware interrupt???
STREX r1,r0,[addr]
CMP r1,0
BNE retry
```

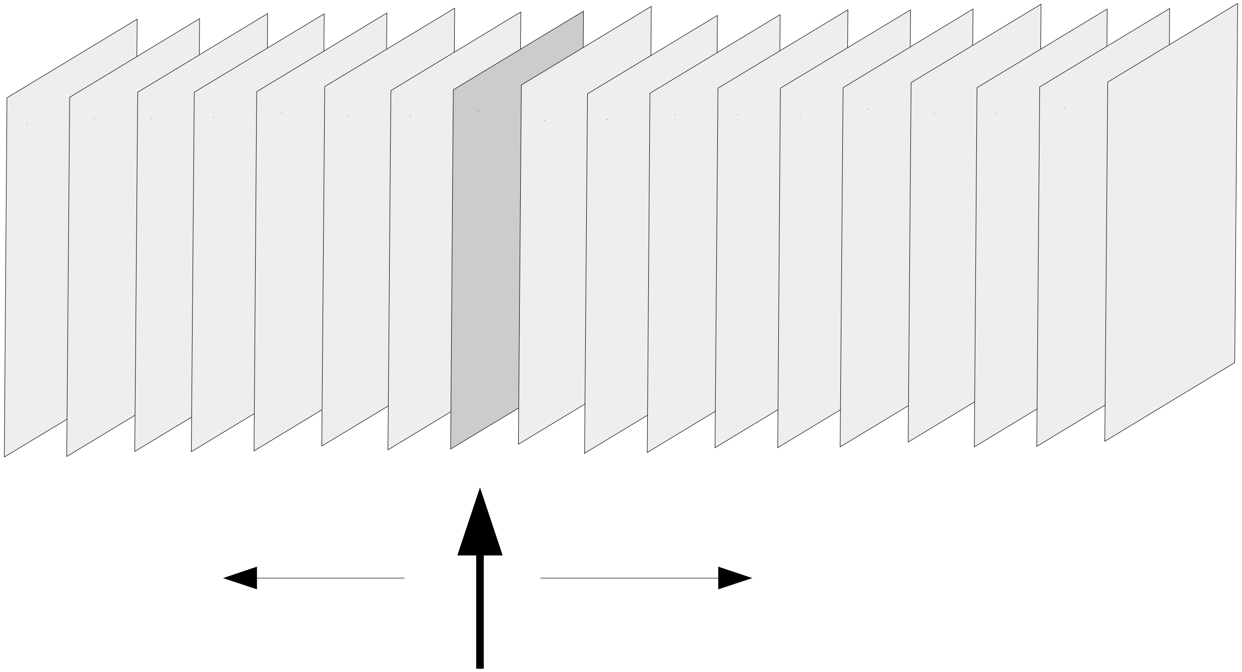
→ Need hardware support to detect/compensate
→ Or binary rewriting

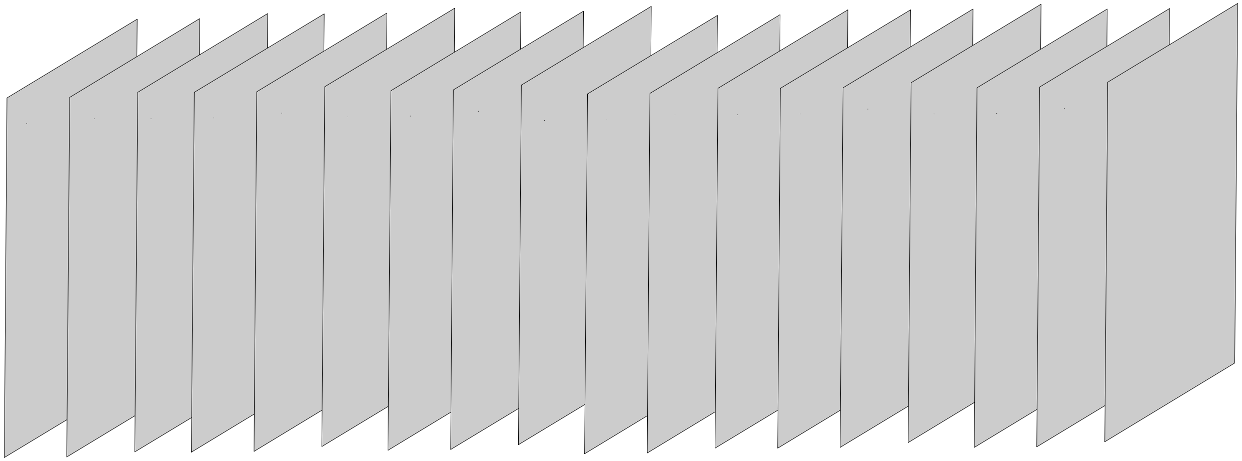
Kernel semantics

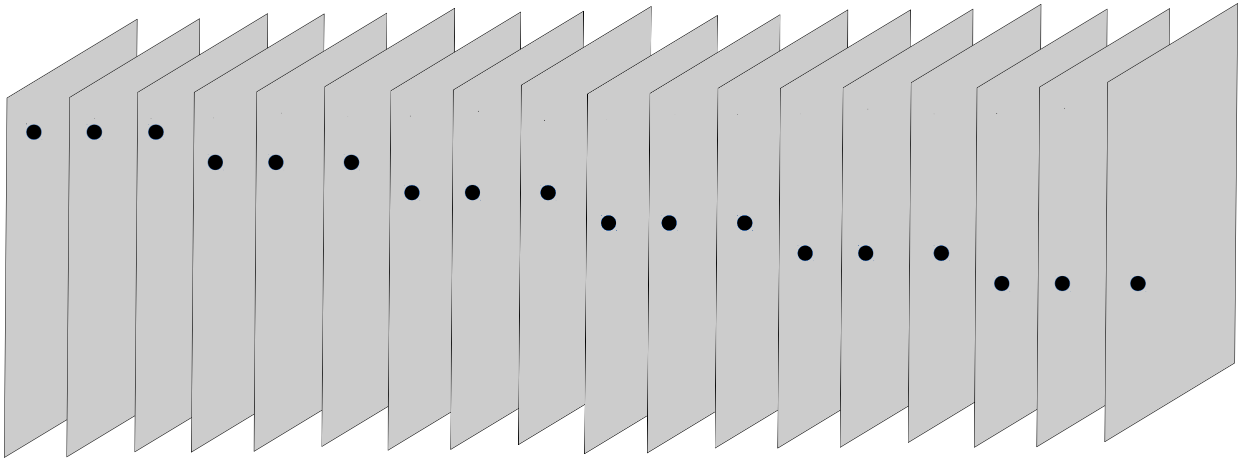
ioctls
Edge cases in system calls
Overhead of switching to supervisor process between each tracee context switch
→ Build into OS/hypervisor???

gdb

Not the ultimate debugger interface







```
static uint8_t dr_reg_to_scratch_mask(reg_id_t reg) {
  if (reg >= DR_REG_R8 && reg <= DR_REG_R11) {
    return 1 << (reg - DR_REG_R8);
  }
  if (reg >= DR_REG_R8D && reg <= DR_REG_R11D) {
    return 1 << (reg - DR_REG_R8D);
  }
  if (reg >= DR_REG_R8W && reg <= DR_REG_R11W) {
    return 1 << (reg - DR_REG_R8W);
  }
  if (reg >= DR_REG_R8L && reg <= DR_REG_R11L) {
    return 1 << (reg - DR_REG_R8L);
  }
  return 0;
}
```

Debugging
→
data analysis and visualization!

ARCANA

Research Chronomancy

REQUIRES

🟪 🟪

🟨 🟨 🟨 🟨

REWARD

4

🟪 and return 1 of your assigned ♟ to your pool.

Time magic has unlimited potential.

http://rr-project.org https://github.com/mozilla/rr