How do Time Travel Debuggers Work?

Greg Law

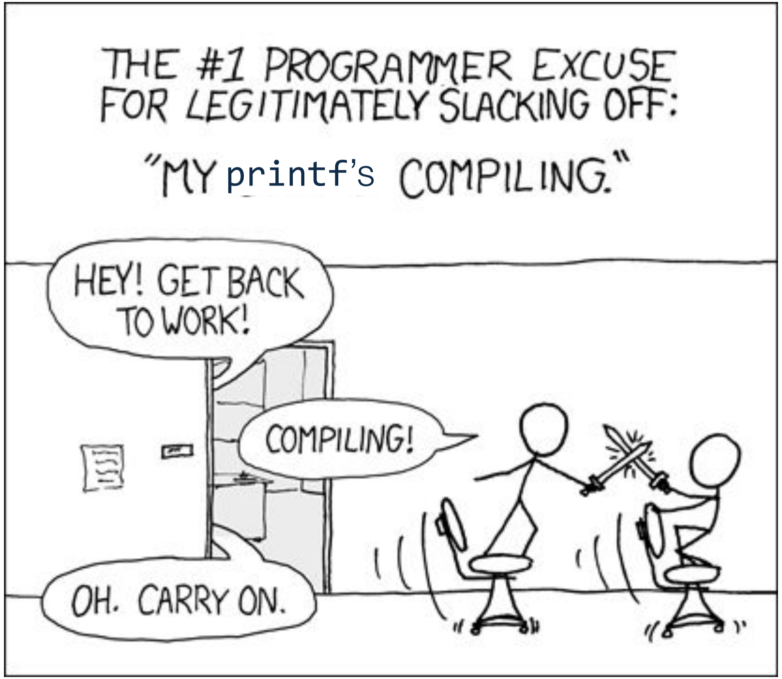
undo

Most programmers spend most of their time debugging.

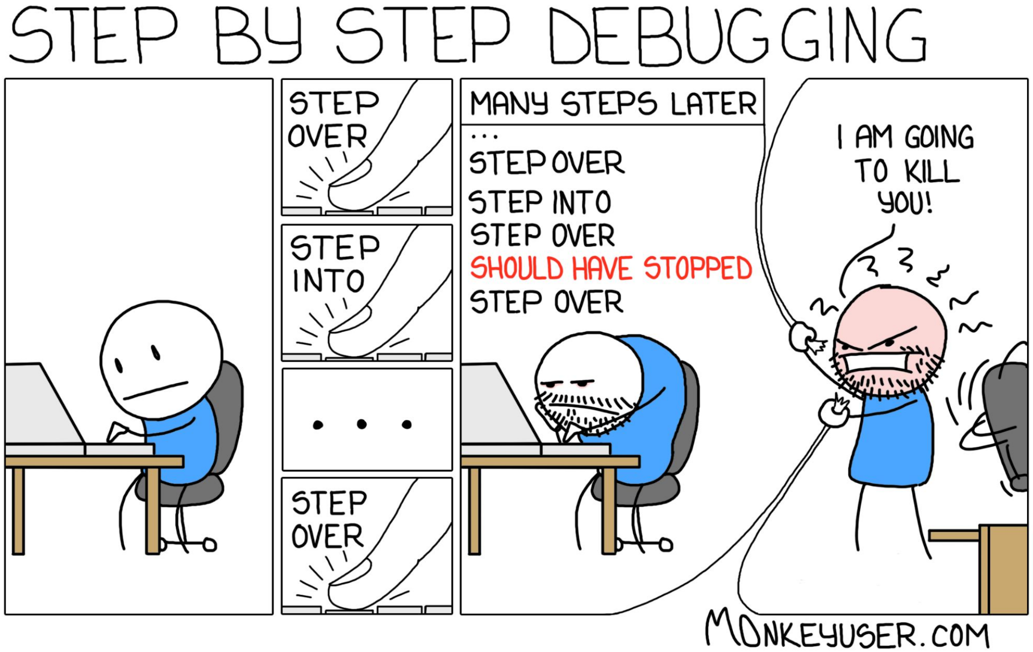
Everyone knows that debugging is twice as hard as writing a program in the first place.

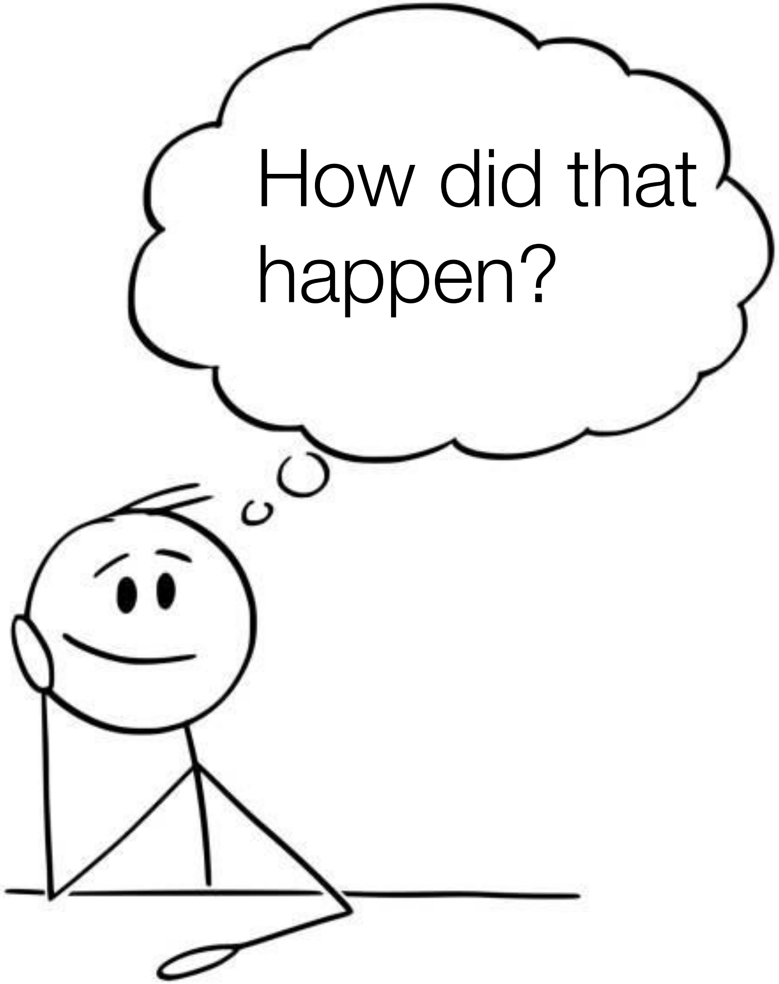
So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan

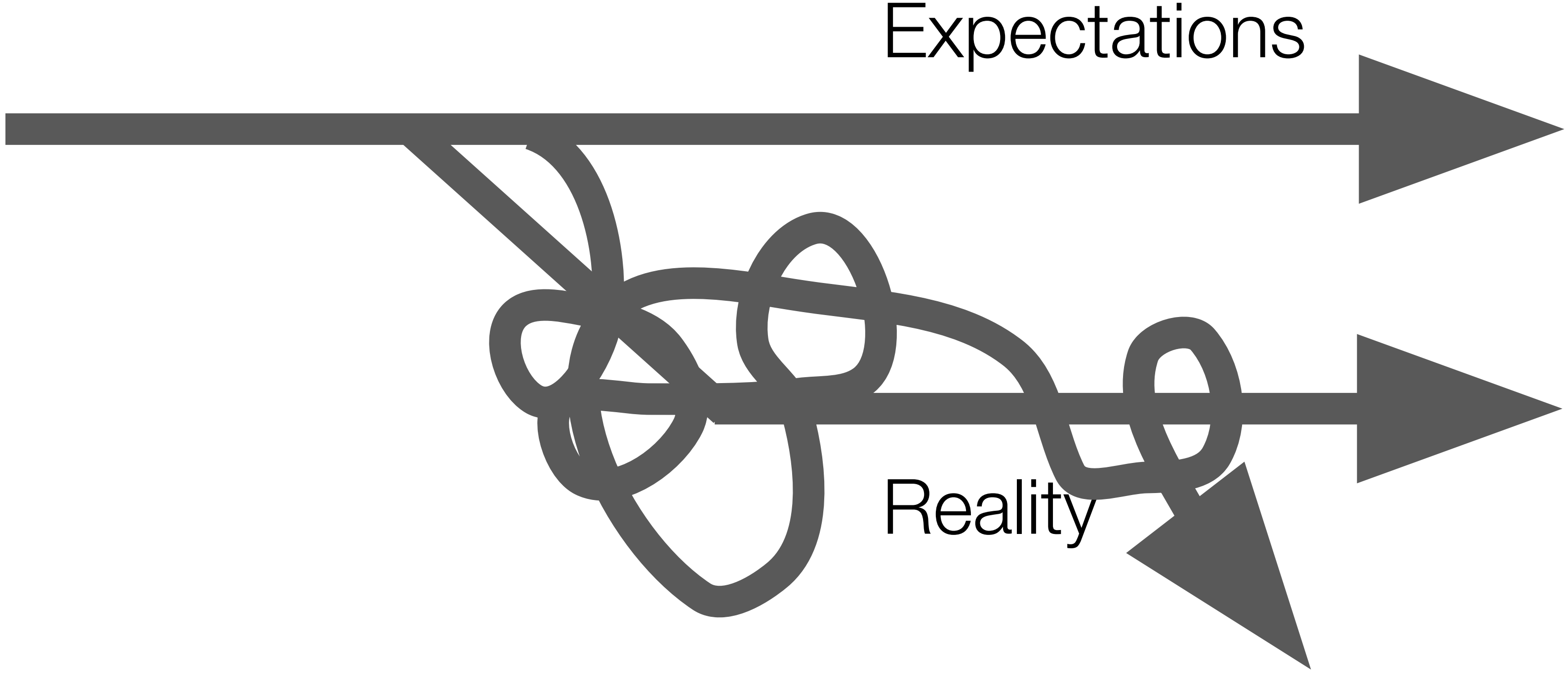

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY printf's COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

VS


STEP BY STEP DEBUGGING
STEP OVER
STEP INTO
...
STEP OVER
MANY STEPS LATER
...
STEP OVER
STEP INTO
STEP OVER
SHOULD HAVE STOPPED
STEP OVER
I AM GOING TO KILL YOU!
MONKEYUSER.COM

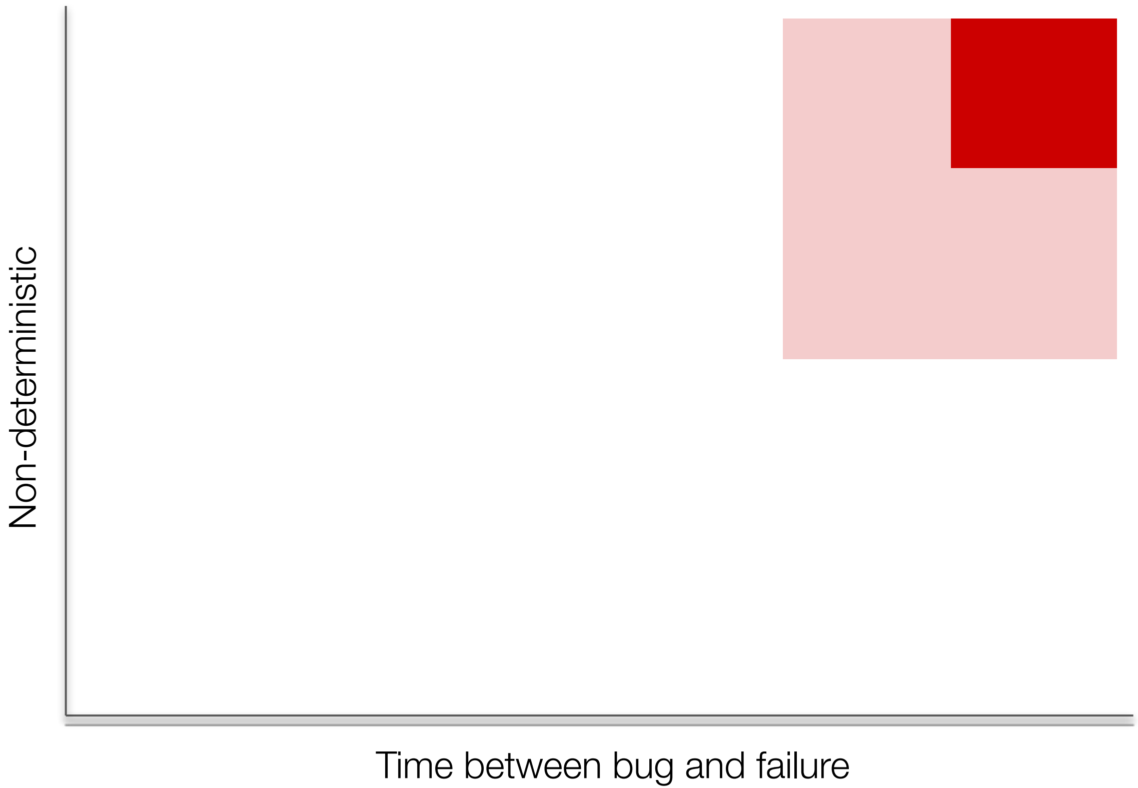

How did that happen?

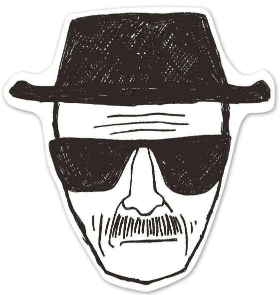

9
WCPO



Expectations
Reality

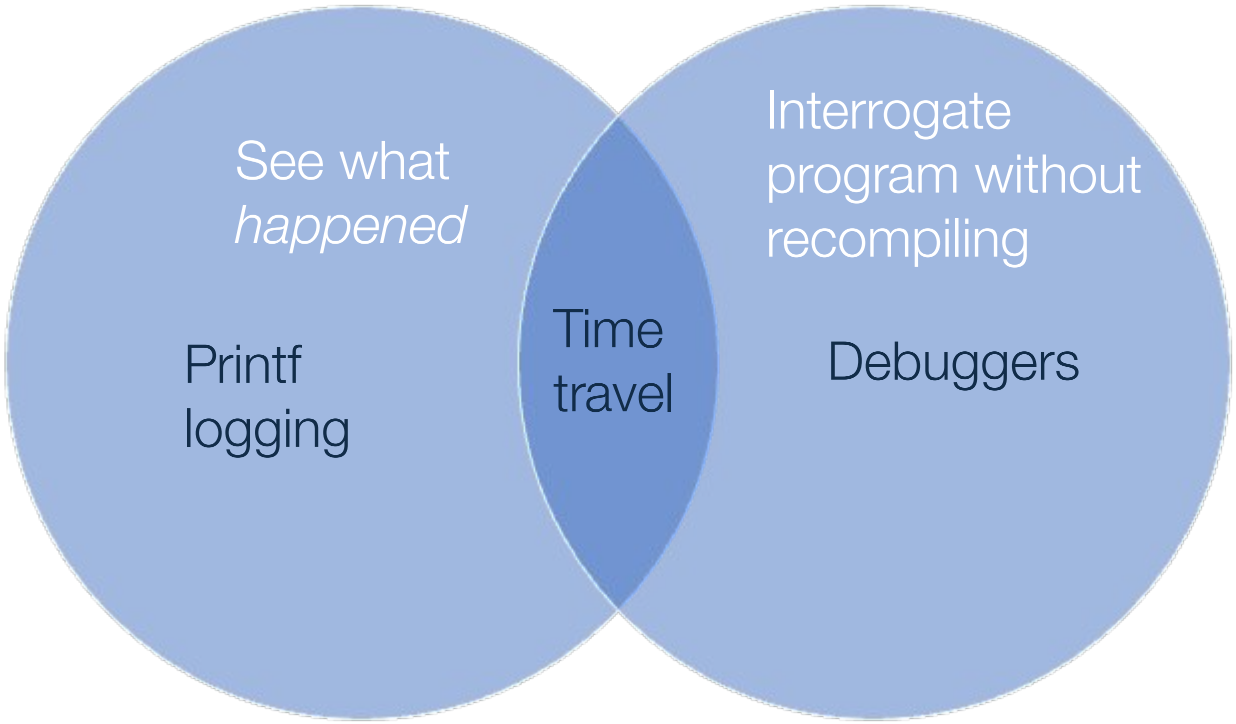
What makes bugs hard to fix?





‘Heisenbugs’

What happened… in detail



undo

C++ projects and products

Linux

- Undo - UDB & LiveRecorder
- rr (rr-project.org)
- GDB (ish)

Windows

- TTD

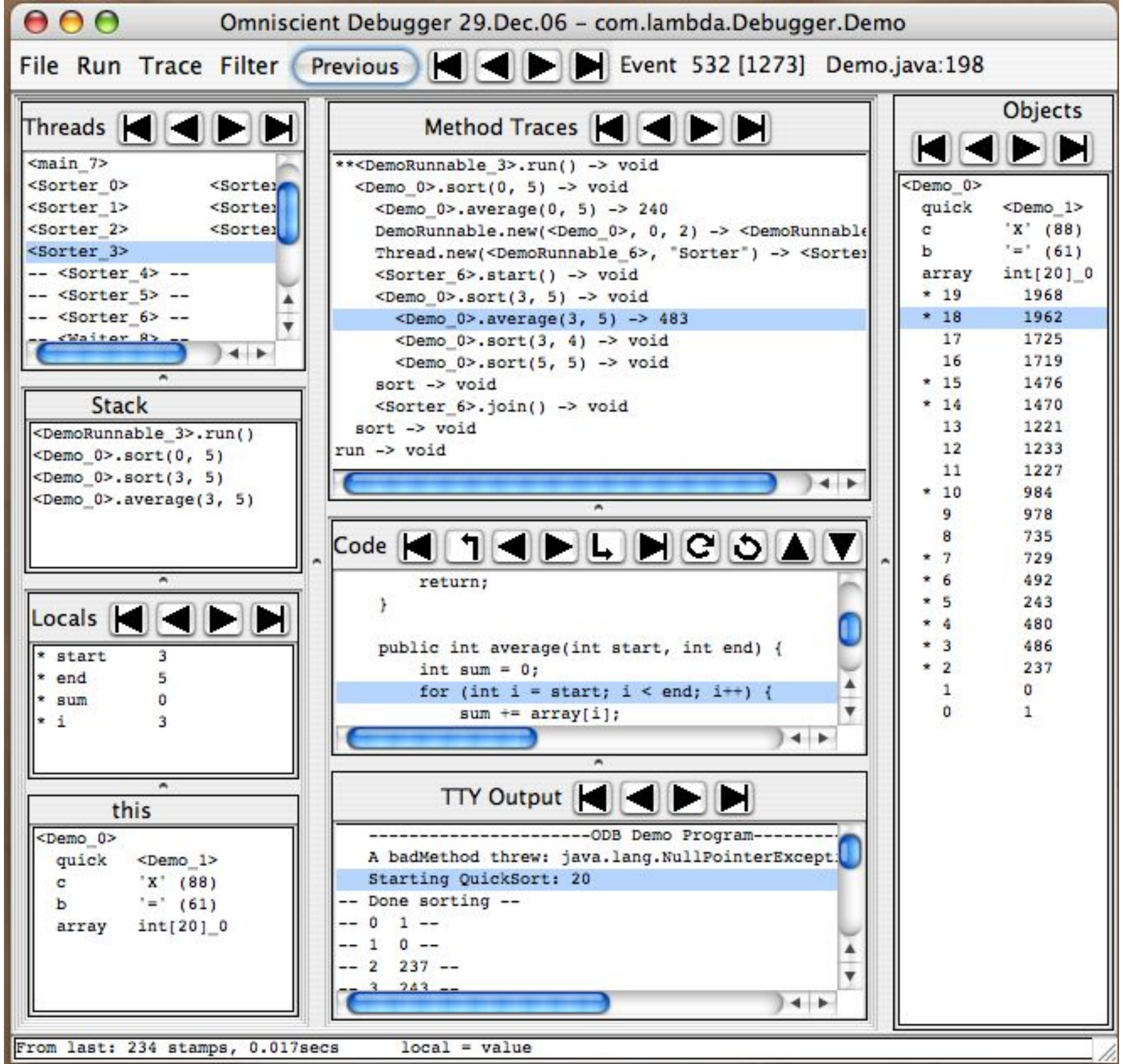
Embedded

- Lauterbach "TRACE32"
- Green Hills TimeMachine

undo

Non C++

• JavaScript / React	replay.io
• .Net	RevDebug
	Visual Studio
• Java	Undo
• Rust, Go	Undo / rr
• Python	Undo*, PyPy*


Omniscient Debugger 29.Dec.06 – com.lambda.Debugger.Demo
File Run Trace Filter Previous
Event 532 [1273] Demo.java:198
Threads
<main_7>
<Sorter_0>
<Sorter_1>
<Sorter_2>
<Sorter_3>
-- <Sorter_4> --
-- <Sorter_5> --
-- <Sorter_6> --
Method Traces
**<DemoRunnable_3>.run() -> void
<Demo_0>.sort(0, 5) -> void
<Demo_0>.average(0, 5) -> 240
DemoRunnable.new(<Demo_0>, 0, 2) -> <DemoRunnable
Thread.new(<DemoRunnable_6>, "Sorter") -> <Sorter
<Sorter_6>.start() -> void
<Demo_0>.sort(3, 5) -> void
<Demo_0>.average(3, 5) -> 483
<Demo_0>.sort(3, 4) -> void
<Demo_0>.sort(5, 5) -> void
sort -> void
<Sorter_6>.join() -> void
sort -> void
run -> void
Objects
<Demo_0>
quick <Demo_1>
c 'X' (88)
b '=' (61)
array int[20]_0
* 19 1968
* 18 1962
17 1725
16 1719
* 15 1476
* 14 1470
13 1221
12 1233
11 1227
* 10 984
9 978
8 735
* 7 729
* 6 492
* 5 243
* 4 480
* 3 486
* 2 237
1 0
0 1
Stack
<DemoRunnable_3>.run()
<Demo_0>.sort(0, 5)
<Demo_0>.sort(3, 5)
<Demo_0>.average(3, 5)
Code
return;
}
public int average(int start, int end) {
int sum = 0;
for (int i = start; i < end; i++) {
sum += array[i];
Locals
* start 3
* end 5
* sum 0
* i 3
this
<Demo_0>
quick <Demo_1>
c 'X' (88)
b '=' (61)
array int[20]_0
TTY Output
----------------------ODB Demo Program--------
A badMethod threw: java.lang.NullPointerExcept
Starting QuickSort: 20
-- Done sorting --
-- 0 1 --
-- 1 0 --
-- 2 237 --
From last: 234 stamps, 0.017secs local = value

undo

Design Decision 1.

How to remember

What was the previous state?

Two options:

1. Save it.
2. Recompute it.

$$a = a + 1$$

$$\Rightarrow a = b$$

undo

Deterministic re-execution

$$f(x) \longrightarrow x'$$

Deterministic re-execution

- Snapshots.

- Event log.

Non-determinism

- What is unpredictable?
  - ~~System calls.~~
  - Thread switches.
  - Asynchronous events (signals).
  - Shared memory accesses.
  - Some machine instructions.

Non-deterministic code

```
0x000055555555515e   mov -0x4(%rbp),%eax
0x0000555555555161   mov %eax,%esi
0x0000555555555163   lea 0xe9a(%rip),%rdi
0x000055555555516a   mov $0x0,%eax
0x000055555555516f   sub %rdx,%rsp
0x0000555555555171   mov %rcx,%r10
0x0000555555555174   mov $0x11,%eax
0x0000555555555179   syscall
```

Here’s some machine code in a program that we want to be able to **record** and **replay**.

deterministic

nondeterministic

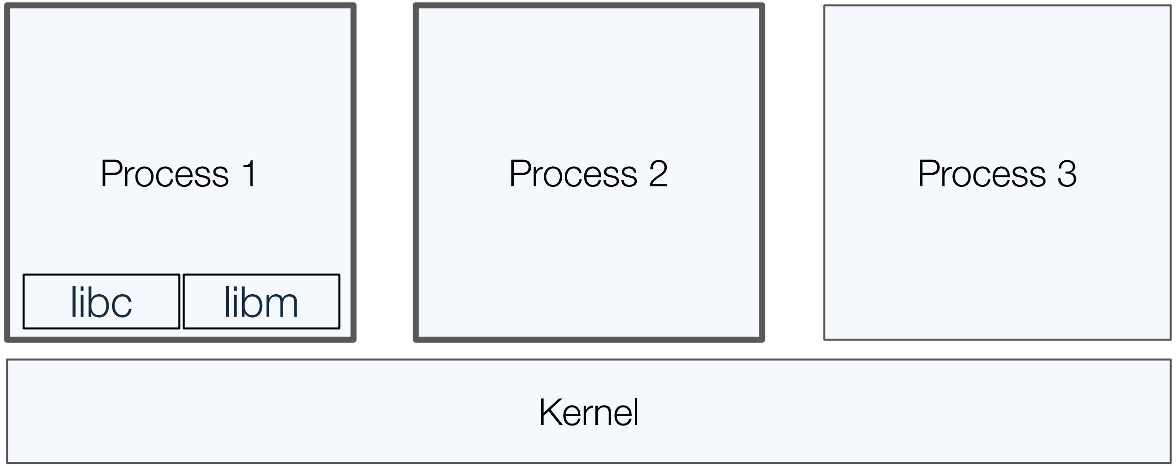
Design Decision 1

Re-execution is the winner

Design decision 2: what exactly to record?

- A process?
- A thread or function?
- A sandbox environment (e.g. JVM or v8?)
- Virtual machine?

Recording at process/OS ABI boundary



undo

Design Decision 2 - winner!

Re-execution at the process boundary is the winner

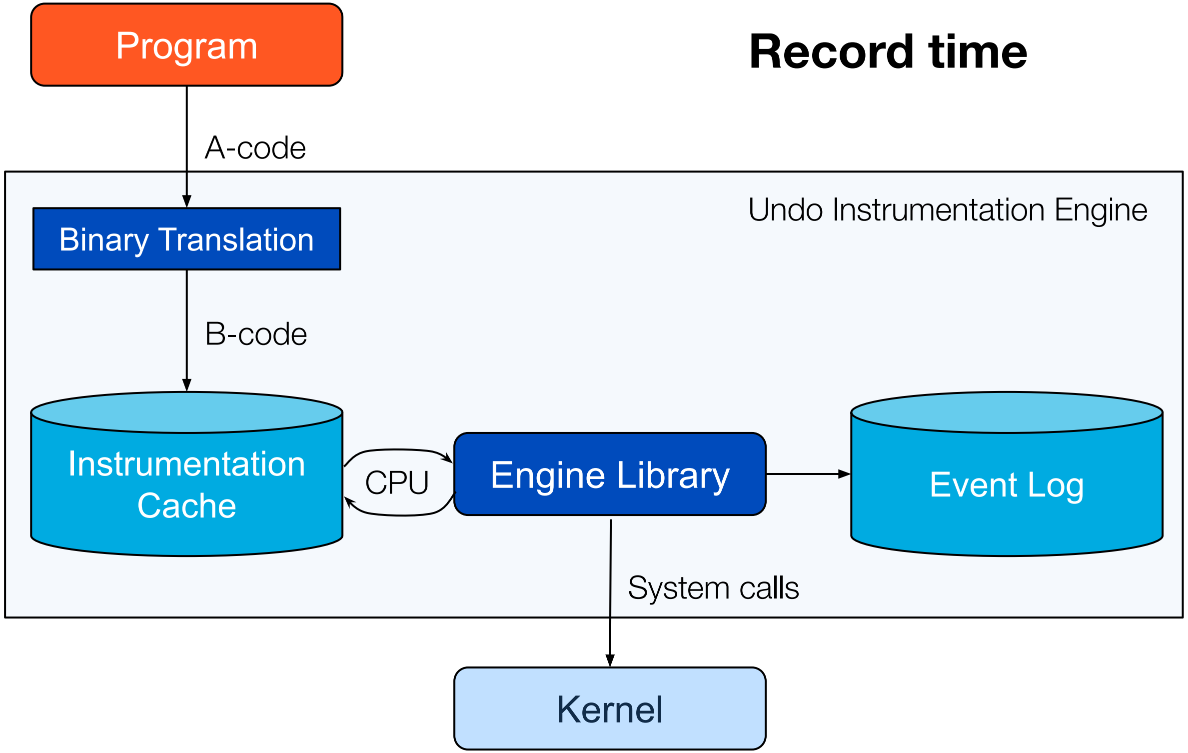
Design Decision 3: How to track time?


MADE IN CHINA

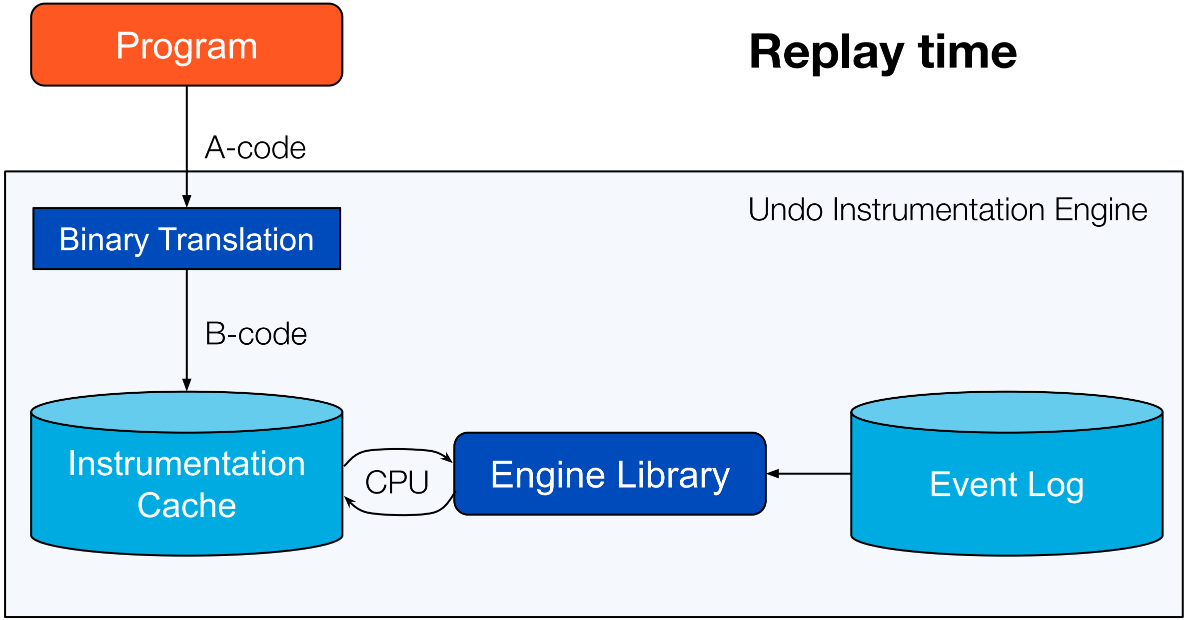
Design Decision 3: How to track time?

To JIT or not to JIT?

Translation and events



Translation and events



Perf counters

- Can be very fast.
- Simpler (less code).

JIT

- Works anywhere
- With anything*.
- Solid

undo

Design Decision 4: Rely on memory determinism?

Memory Determinism

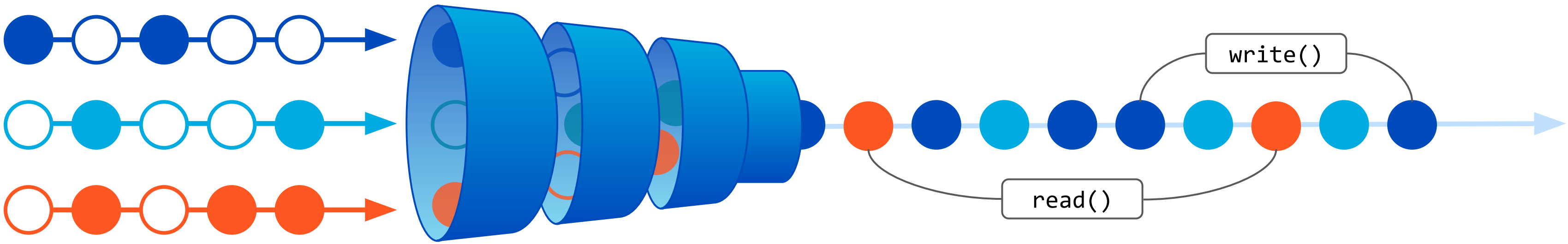
- (Much) smaller event logs
- Faster single-threaded

Not

- Unknown syscalls
- Parallel record

undo™

Thread serialization



Non-determinism

- What is unpredictable?
  - ~~System calls.~~
  - Thread switches.
  - Asynchronous events (signals).
  - ~~Shared memory accesses.~~
  - Some machine instructions.

Design decisions

	Undo	rr	WinDbg	replay.io	ODB
At what boundary to capture	proc	proc	proc	proc	JVM
Binary rewriting instrumentation	yes	no	yes	no	no
All/some/no memory recording	some	none	all*	none	all
Separate record/replay phases	yes/no	yes	yes	yes	yes
Parallel thread recording	no	no	yes	no	yes

Debuginfo

```
(gdb) print foo
<value optimized out>
(gdb)
```