Time Travel Debugging

Greg Law

Chris Croft-White

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

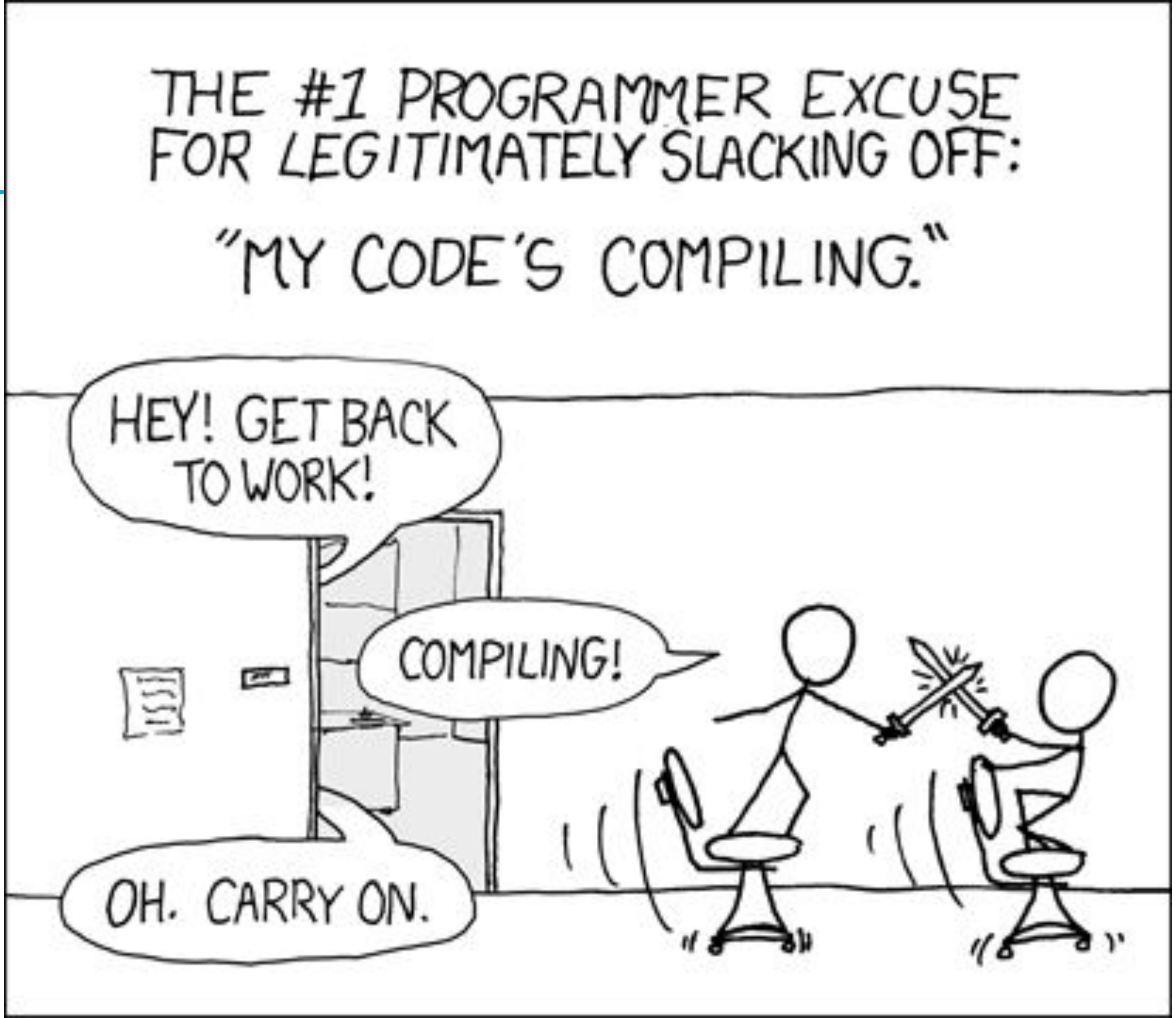
How do we debug?

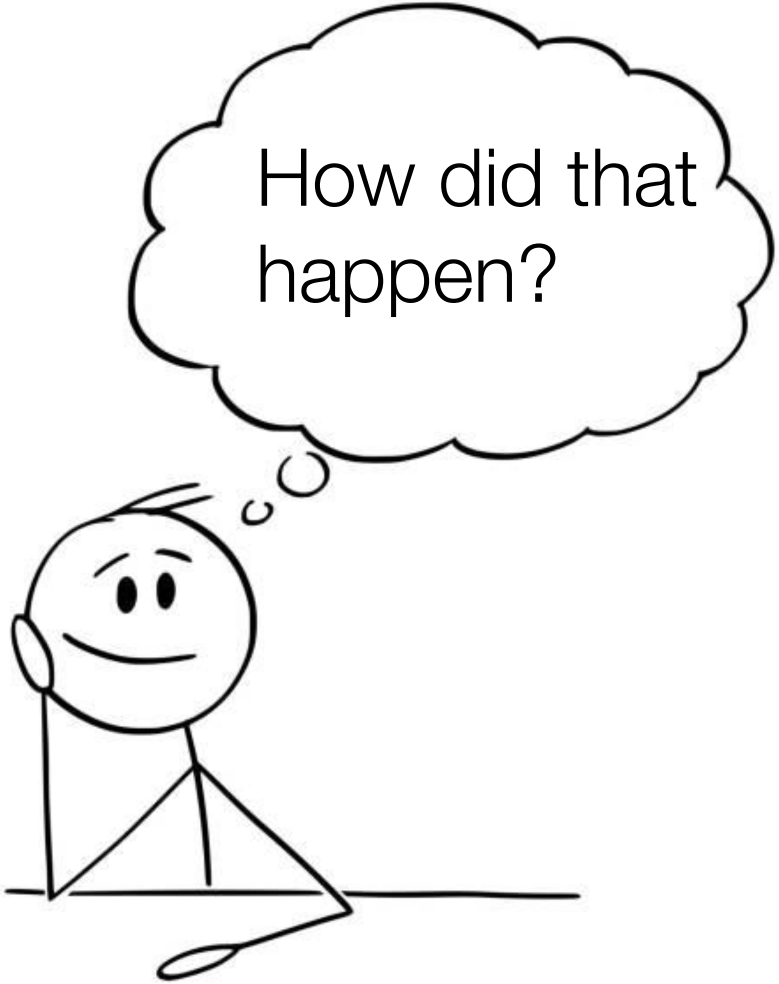
Use dynamic checkers (e.g. valgrind, ASAN)

Use a debugger (e.g. IntelliJ, GDB)

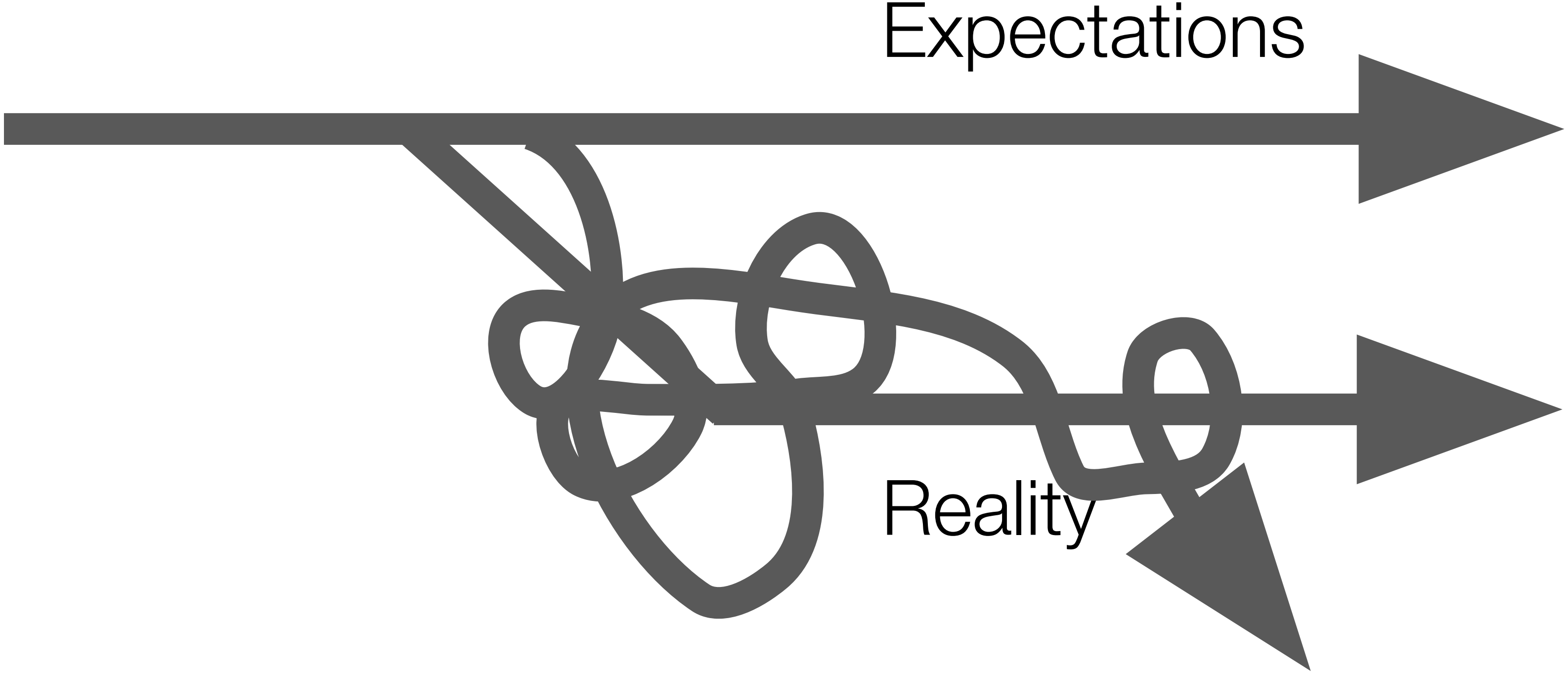
Dynamic logging (e.g. LightRun)

`logger.debug()`   **`printf()`**


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


How did that happen?

9
WCPO


Expectations
Reality

Use cases

Production*

CI/CD

“Inner loop” development

C++ projects and products

Linux

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

Windows

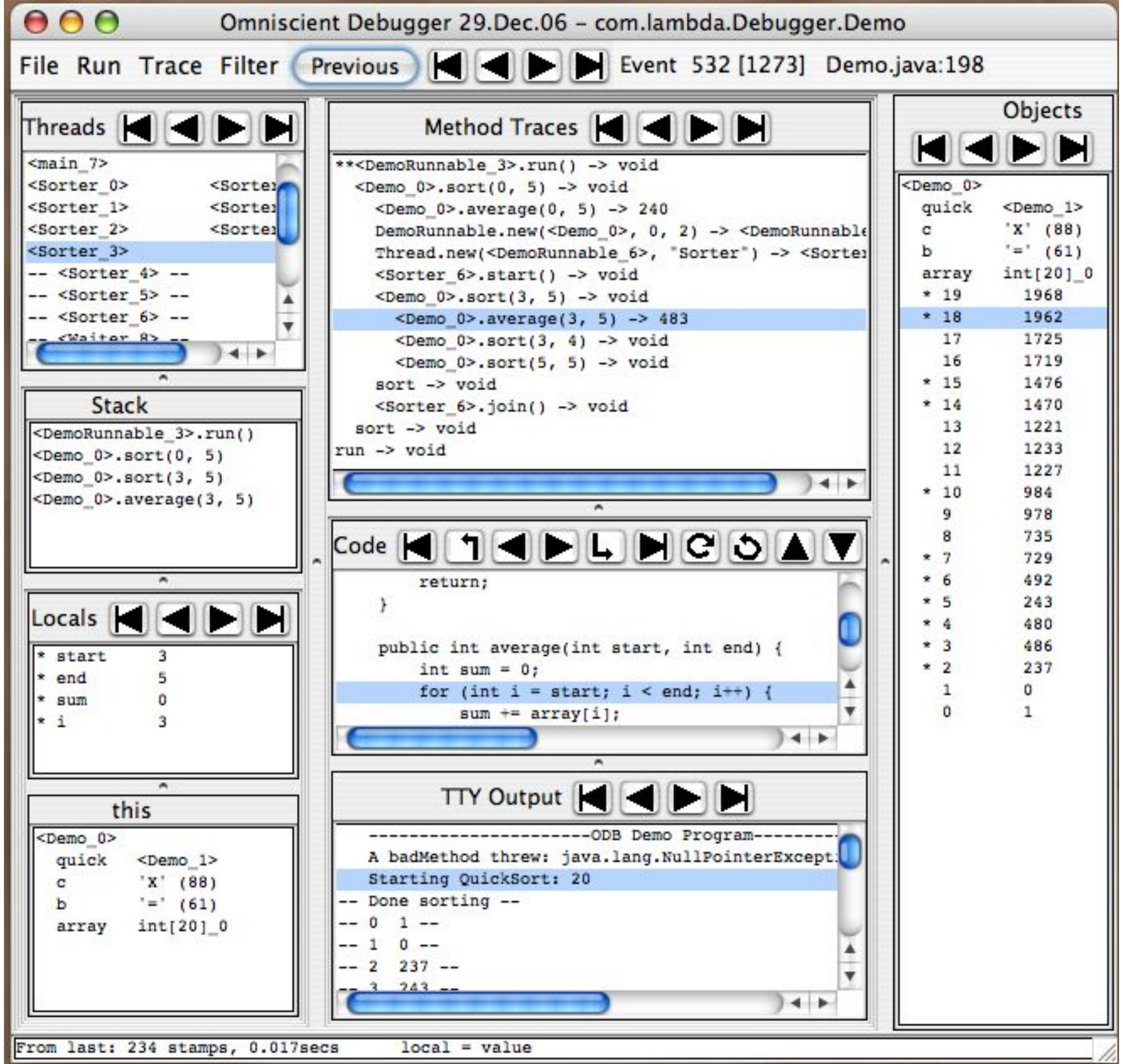
- TTD

Embedded

- Lauterbach “TRACE32”
- Green Hills TimeMachine

Non C++

● JavaScript / React	replay.io
● .Net	RevDebug Visual Studio
● Java	Undo
● Rust, Go	Undo / rr



Threads

```
<main_7>
<Sorter_0>          <Sorte
<Sorter_1>          <Sorte
<Sorter_2>          <Sorte
<Sorter_3>
-- <Sorter_4> --
-- <Sorter_5> --
-- <Sorter_6> --
-- <Waiter_8> --
```

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") -> <Sorte
    <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
```

Stack

```
<DemoRunnable_3>.run()
<Demo_0>.sort(0, 5)
<Demo_0>.sort(3, 5)
<Demo_0>.average(3, 5)
```

Locals

```
* start     3
* end       5
* sum       0
* i         3
```

this

```
<Demo_0>
  quick     <Demo_1>
  c         'X' (88)
  b         '=' (61)
  array     int[20]_0
```

Code

```
        return;
    }

    public int average(int start, int end) {
        int sum = 0;
        for (int i = start; i < end; i++) {
            sum += array[i];
```

TTY Output

```
  ---------------------ODB Demo Program--------
  A badMethod threw: java.lang.NullPointerExcept
  Starting QuickSort: 20
-- Done sorting --
-- 0  1 --
-- 1  0 --
-- 2  237 --
-- 3  243 --
```

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
```

```
From last: 234 stamps, 0.017secs       local = value
```

What was the previous state?

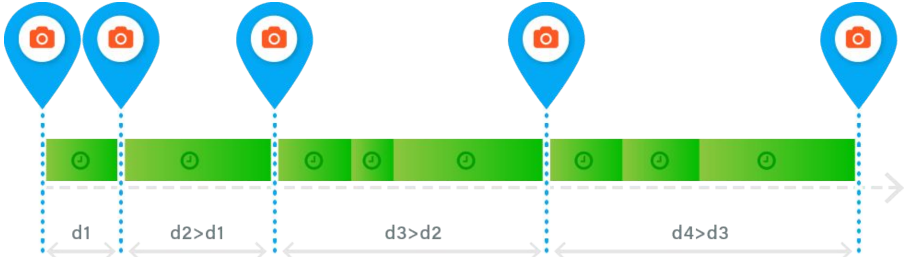
Two options:

1. Save it.
2. Recompute it.

$$a = a + 1$$ ✓

$$a = b$$ ✗

Snapshots



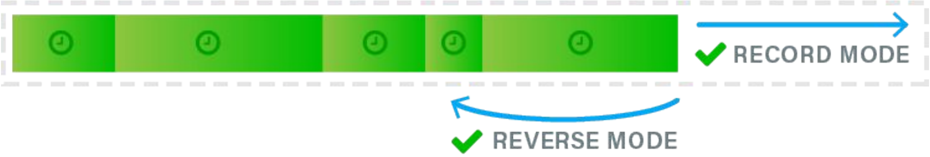
Maintain snapshots through history

Resume from these - run forward as needed

Copy-on-Write for performance & memory efficiency

Adjust spacing to anticipate user’s needs

Event log



Event Log captures non-deterministic state

Stored in memory

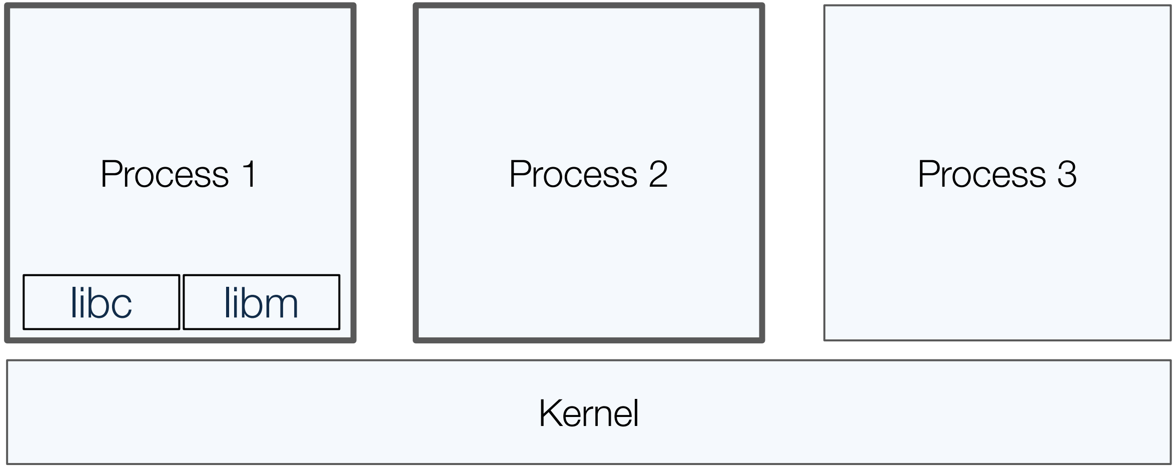
Efficient, diff-based representation

Recorded during debug (or Live Recording)

Replayed to reconstruct any point in history

Saved to create a recording file for later use

Recording at process/OS ABI boundary



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 accesses	some	none	all*	none	all
● Separate record/replay phases	yes/no	yes	yes	yes	yes

DEMO TIME!