



# Time-Travel Debugging Java applications

Greg Law, co-founder

**undo**<sup>™</sup>

# Debugging - our dirty secret

Programming *is* debugging

Most software is not truly understood by anyone

What happened?

printf, printf, printf

Why can't the computer just tell us?



In the beginning

Sir Maurice Wilkes, 1913-2010



undo





## In the beginning

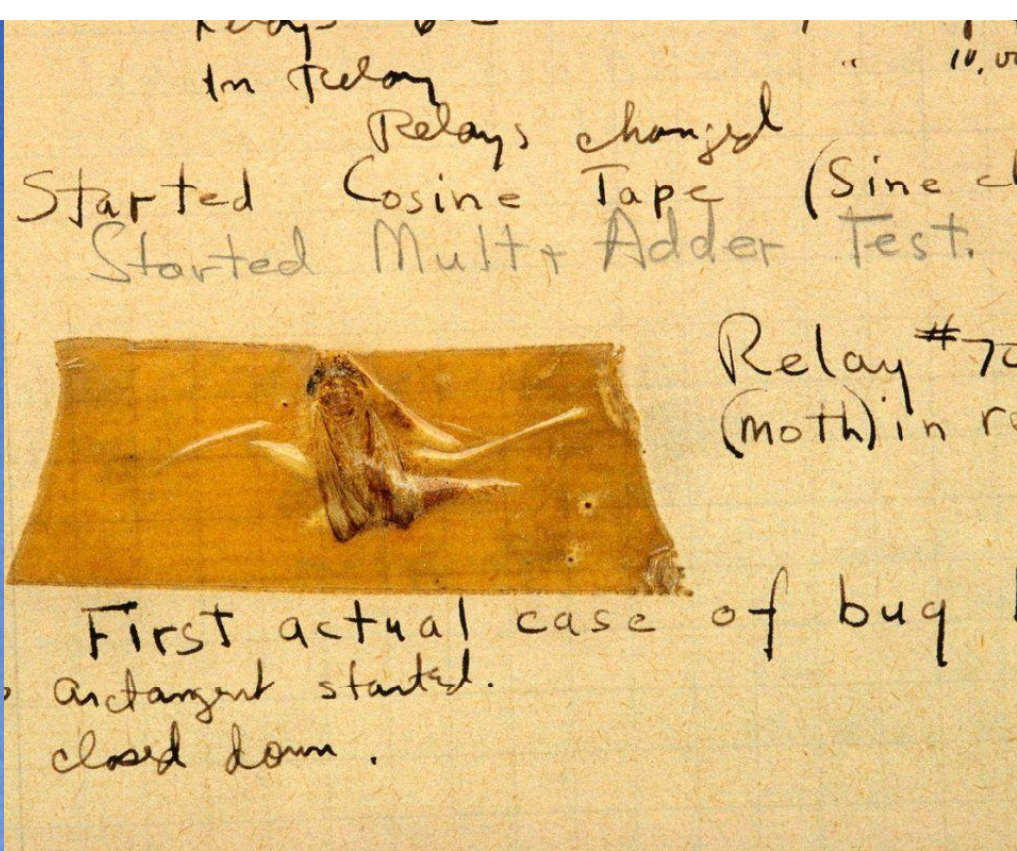
---

I well remember [...] the realization came over me with full force that a **good part of the remainder of my life was going to be spent in finding errors in my own programs**

Sir Maurice Wilkes, 1913-2010







*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?*

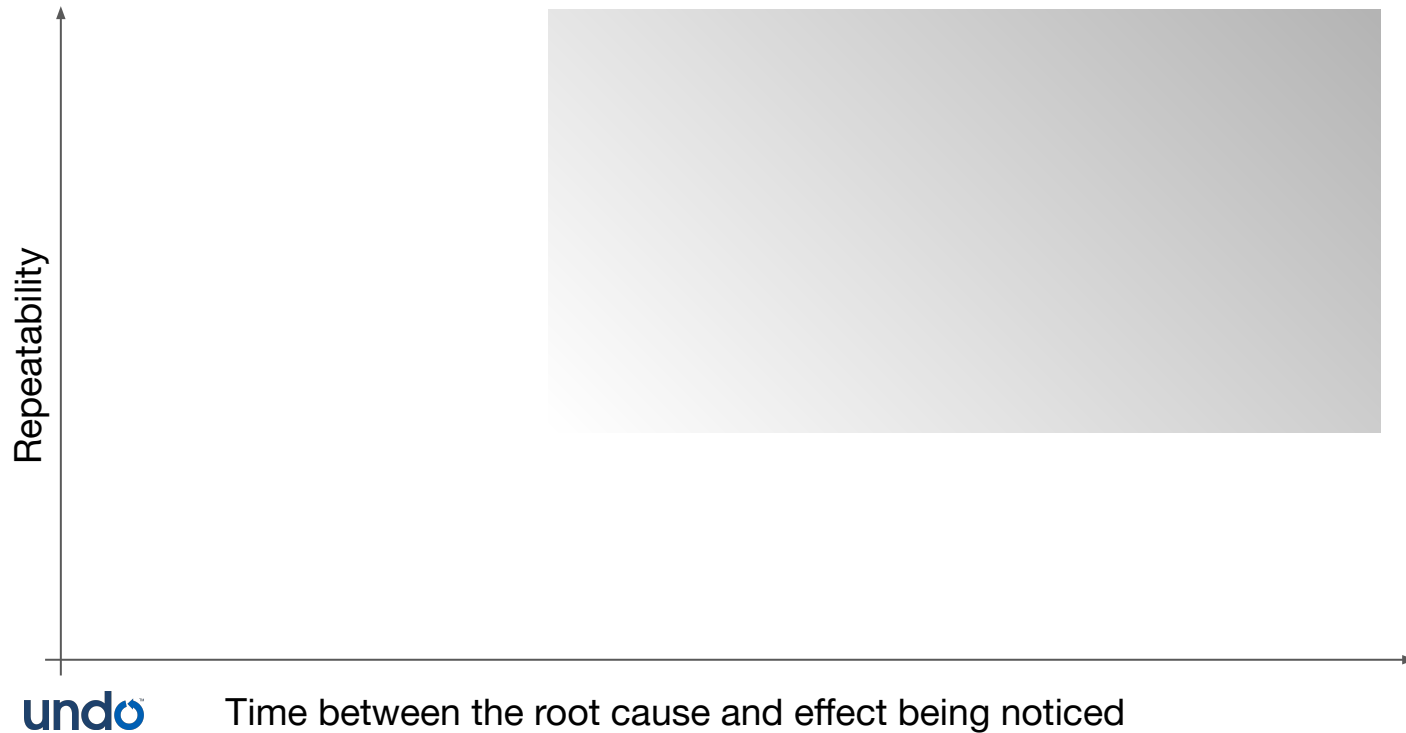
*Brian Kernighan*



# Computers are hard

# What happened?

# What makes bugs really hard?





Omniscient Debugger 29.Dec.06 com.lambda.Debugger.Demo

File Run Trace Filter Previous [Navigation Icons] Event 532 [1273] Demo.java:198

### Threads

```

<main_7>
<Sorter_0> <Sorter_0>
<Sorter_1> <Sorter_1>
<Sorter_2> <Sorter_2>
<Sorter_3> <Sorter_3>
-- <Sorter_4> --
-- <Sorter_5> --
-- <Sorter_6> --
-- <Sorter_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_6>
Thread.new(<DemoRunnable_6>, "Sorter") -> <Sorter_6>
<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.NullPointerException.
Starting QuickSort: 20
-- Done sorting --
-- 0 1 --
-- 1 0 --
-- 2 237 --
-- 3 243 --

```

From last: 234 stamps, 0.017secs local = value

# What was the previous state?

Two options:

1. Save it.
2. Recompute it.

$$a = a + 1 \quad \checkmark$$

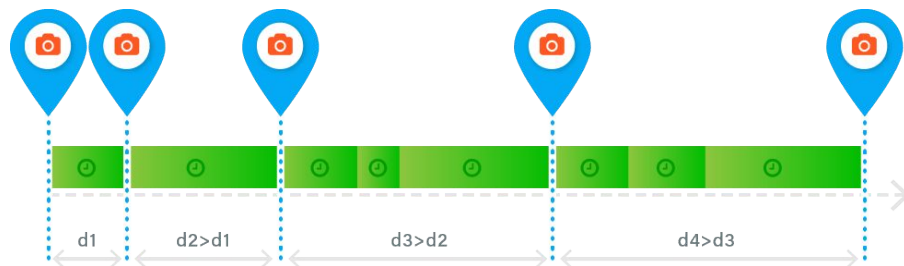
$$a = b \quad \times$$





# Snapshots

---



Maintain snapshots through history

Resume from these - run forward as needed

Copy-on-Write for 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

# Instrumentation

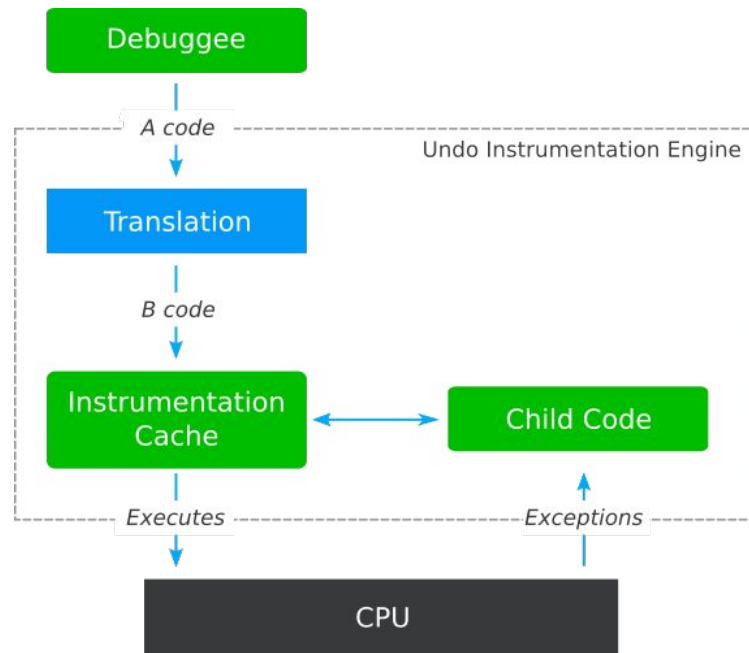
Undo Engine captures *all* non-determinism

Some machine instructions are non-deterministic

`rdtsc`, `cuid`, `syscall`, etc

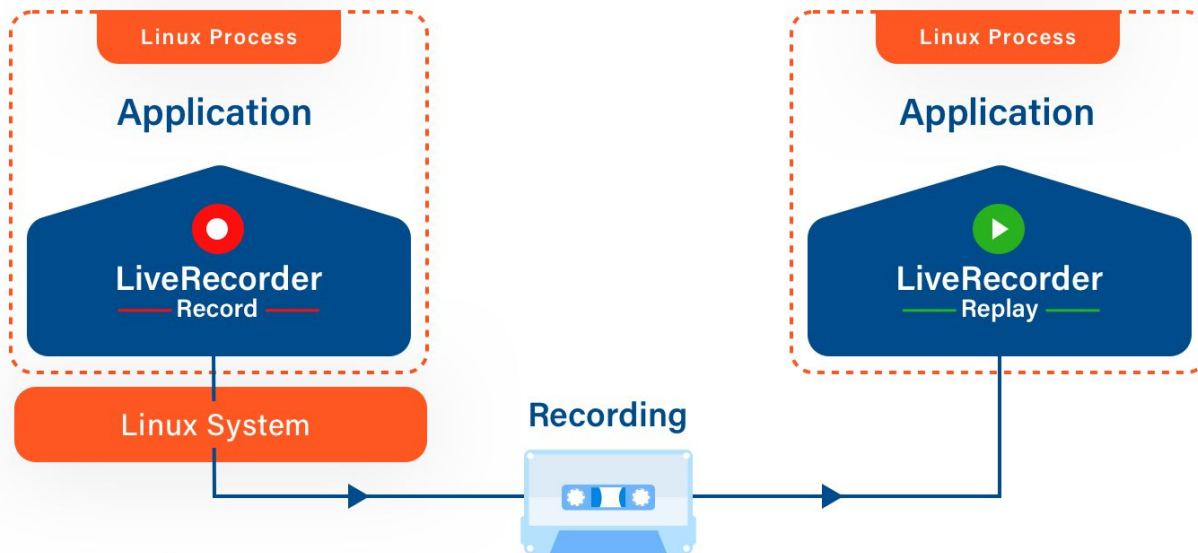
Needs to capture all this and provide precise control over execution in general

*Solution:* Runtime instrumentation





# In-process Virtualization



## Record

- Captures all state changes in a running process
- 100% reproduction of execution history

## Replay

- Replay & analyze the recording
- Detect root cause of bug
- Reverse debug and resolve

# Works well in conjunction with live logging & tracing

Logging & tracing give a high-level 'story' of a program's execution

Use it to know where to go in a recording

Apply logging to a recording

# 80/20 Rule

~~80/20 Rule~~

# Multiple implementations

---

For Linux:

- Undo LiveRecorder (C++, Go, Java)
- rr (C++, Go)
- gdb process record

For Windows:

- Microsoft's Time-Travel Debugger (C++, C#, Chakracore JS)
- RevDebug (C#, Java)

Cross platform

- Chronon debugger
- Omniscient debugger



- 
1. Computers are hard & debugging is under-served
  2. Time Travel is awesome!
  3. 80/20 rule does not always apply
-