Debugging in 2011

The traditional debugger

Traditional Debugger

- Invented in the 70s
- Design assumptions and implementation no longer useful for programs of 2011

chronon

Design assumptions

- Programs are single threaded
- Flow of execution is sequential
- Bugs are always reproducible
- Programs run for short periods of time

chronon

Designed for single thread

- ‘Stepping’ buttons assume sequential execution
- What happens in another thread when you ‘step over’ in one thread?

chronon

Short running, reproducible bugs

- Breakpoint model assumes short running programs
- Also assumes bugs are always reproducible

chronon

Not designed for mutithreading

- Hitting a breakpoint changes all thread timing.
- Just showing stack of multiple threads does not really help.

But in 2011…

Programs of 2011

- Multithreaded is the norm, not the exception
- Flow of execution not 100% sequential
- Reproducing bugs is (very) tough
- Programs run for weeks/months/years

chronon

Proof of all this….

chronon

Logging

Problems with logging

- Only real way to debug programs of 2011
- Distraction from actual coding
- Impossible to debug from huge logs for long running programs
- Fundamentally broken:
  - Trying to ‘predict’ errors
  - In real life, errors happen where you least expect
  - Thus, no logging statement where error occurred

Rethinking the debugger…

chronon

Background

- Chronon makes a ‘DVR for Java’
- Replaying the program is useless if you cant debug
- The traditional debugger didn’t work

chronon

Time Travelling Debugger

- No Breakpoints
- Jump to any point in time. No delay
- Record everything, don't reproduce
- Embrace both multi-threaded **and** sequential execution

chronon

Implementation

- Chronon ‘recordings’ are essentially indexed data files
- Read data and display state when you are debugging
- Essentially give an illusion of replaying
- All we care is to debug, not replay

chronon