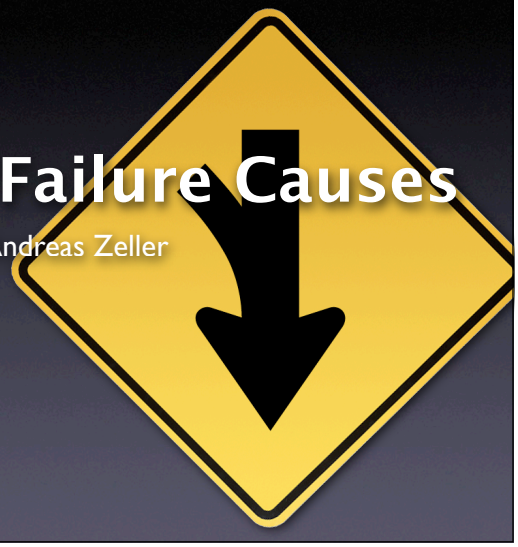
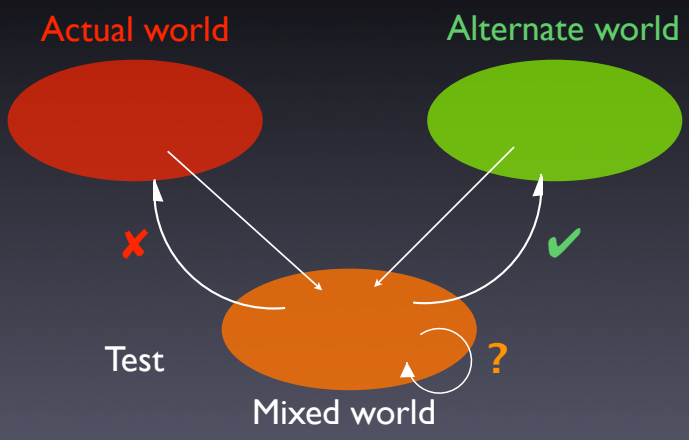
Isolating Failure Causes

Andreas Zeller



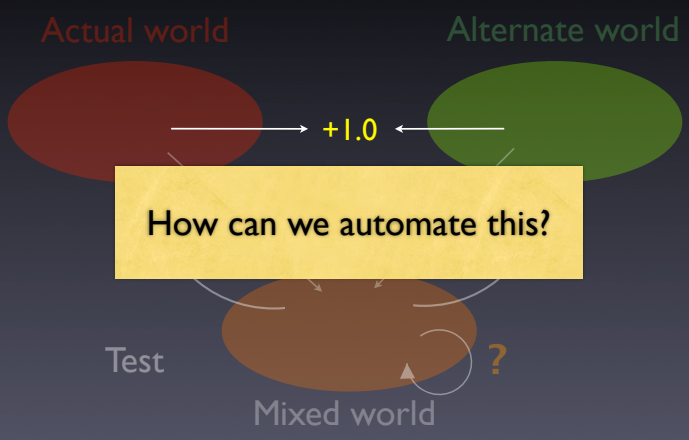
1

Isolating Causes



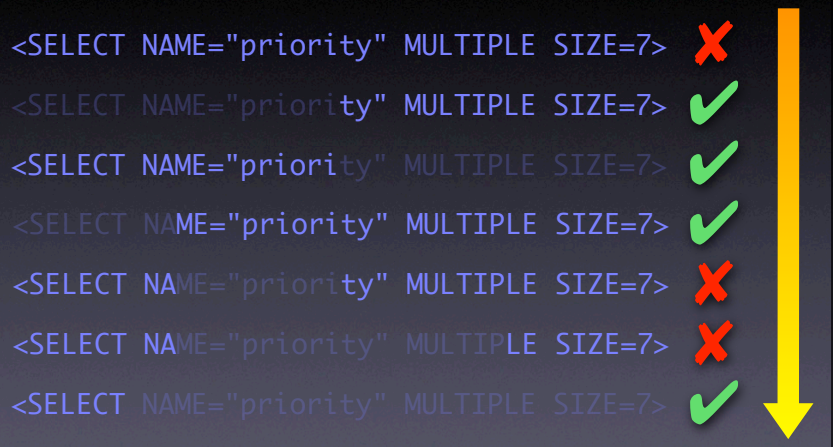
2

Isolating Causes



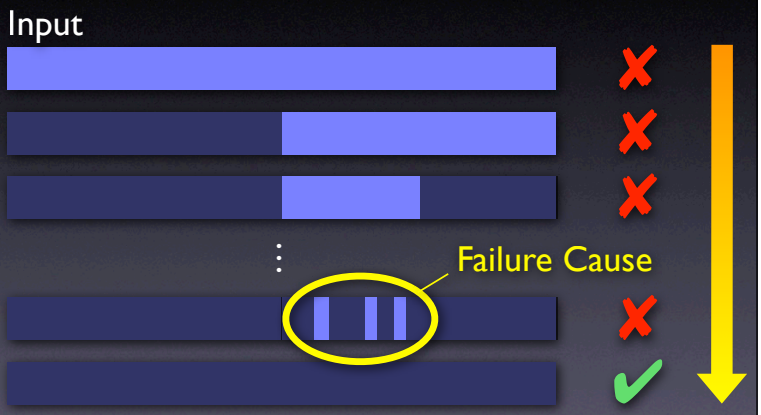
3

Simplifying Input



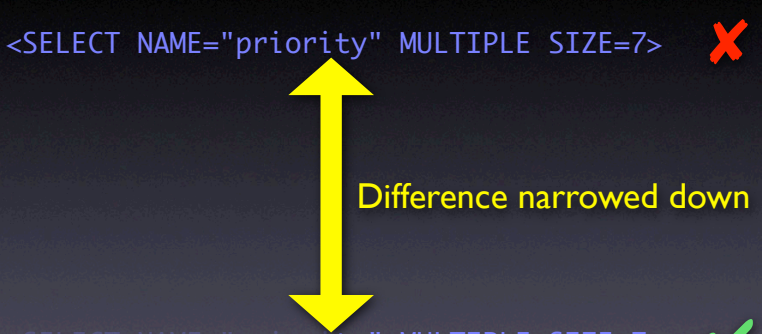
4

Simplifying



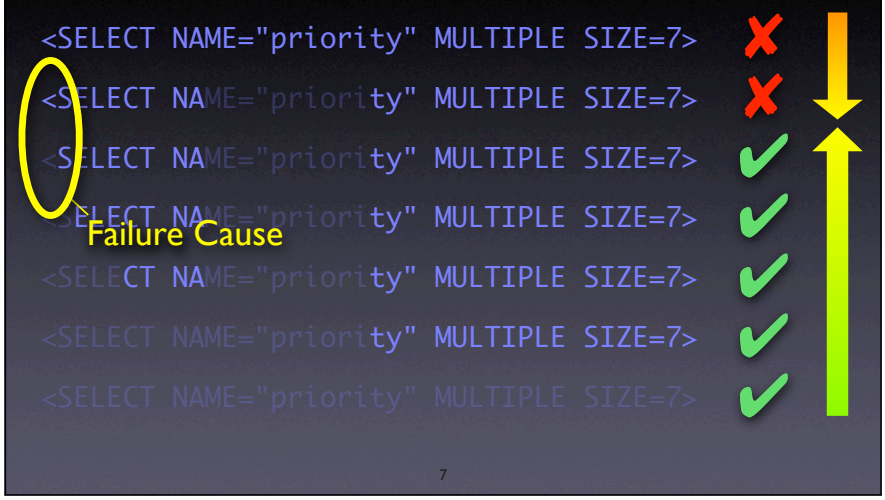
5

Isolating Input



6

Isolating Input



7

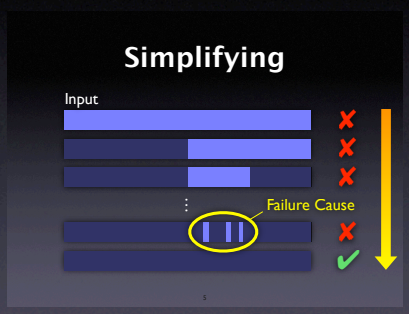
7

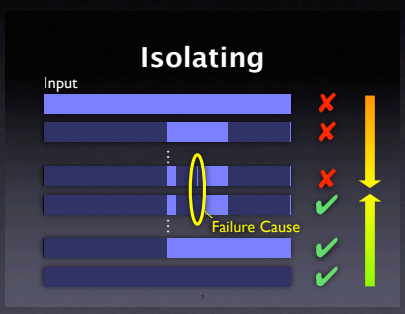
Isolating

8

8

Finding Causes





- minimal input
- minimal context

- minimal difference
- common context

9

9

Configuration

Circumstance

$$\delta$$

All circumstances

$$C = \{\delta_1, \delta_2, \dots\}$$

Configuration $c \subseteq C$

$$c = \{\delta_1, \delta_2, \dots \delta_n\}$$

Tests

Testing function

$$test(c) \in \{✔, ✘, ?\}$$

Initial configurations

$$test(c_✔) = ✔$$
$$test(c_✘) = ✘$$

Minimal Difference

Goal: Subsets $c'_✘$ and $c'_✔$

$$\varnothing = c_✔ \subseteq c'_✔ \subset c'_✘ \subseteq c_✘$$

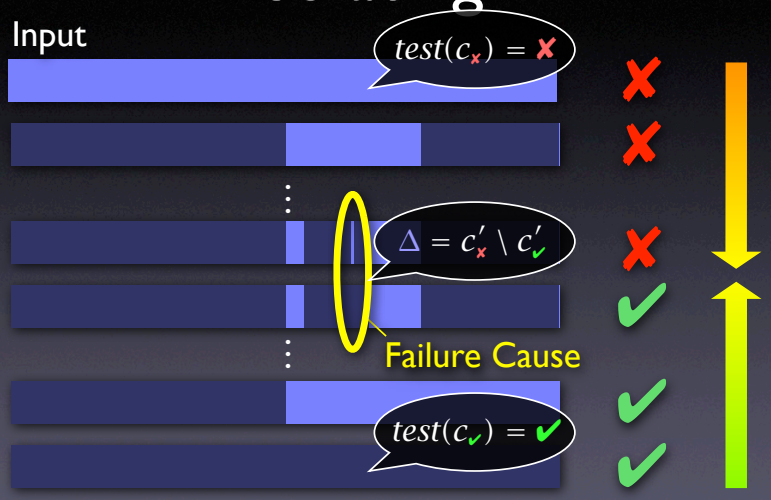
Difference

$$\Delta = c'_✘ \setminus c'_✔$$

Difference is 1-minimal

$$\forall \delta_i \in \Delta \cdot test(c'_✔ \cup \{\delta_i\}) \neq ✔ \wedge test(c'_✘ \setminus \{\delta_i\}) \neq ✘$$

Isolating



13

Algorithm Sketch

- Extend *ddmin* such that it works on *two sets at a time* – $c'_{\times}$ and $c'_{\checkmark}$
- Compute subsets

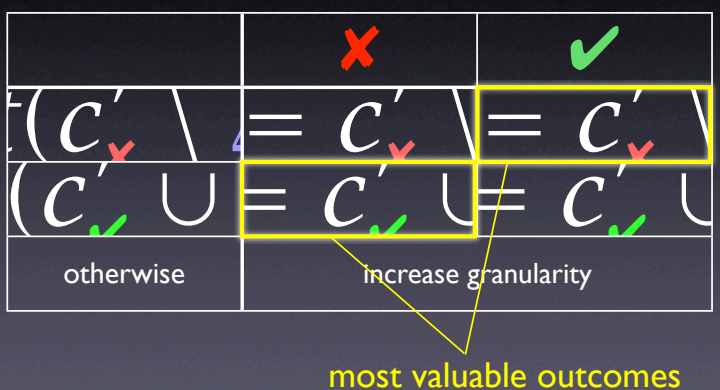
$$\Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_n = \Delta = c'_{\times} \setminus c'_{\checkmark}$$

- For each subset, test
  - the *addition* $c'_{\checkmark} \cup \Delta_i$
  - the *removal* $c'_{\times} \setminus \Delta_i$

14

14

Test Outcomes



15

15

dd in a Nutshell

$dd(c_{✔}, c_{✘}) = (c'_{✔}, c'_{✘})$ $\quad \Delta = c'_{✘} \setminus c'_{✔}$ is 1-minimal

$dd(c_{✔}, c_{✘}) = dd'(c_{✔}, c_{✘}, 2)$

$dd'(c'_{✔}, c'_{✘}, n) =$

$$\begin{cases} (c'_{✔}, c'_{✘}) & \text{if } |\Delta| = 1 \\ dd'(c'_{✘} \setminus \Delta_i, c'_{✘}, 2) & \text{if } \exists i \in \{1..n\} \cdot test(c'_{✘} \setminus \Delta_i) = ✔ \\ dd'(c'_{✔}, c'_{✔} \cup \Delta_i, 2) & \text{if } \exists i \in \{1..n\} \cdot test(c'_{✔} \cup \Delta_i) = ✘ \\ dd'(c'_{✔} \cup \Delta_i, c'_{✘}, \max(n-1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot test(c'_{✔} \cup \Delta_i) = ✔ \\ dd'(c'_{✔}, c'_{✘} \setminus \Delta_i, \max(n-1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot test(c'_{✘} \setminus \Delta_i) = ✘ \\ dd'(c'_{✔}, c'_{✘}, \min(2n, |\Delta|)) & \text{else if } n < |\Delta| \text{ (“increase granularity”)} \\ (c'_{✔}, c'_{✘}) & \text{otherwise} \end{cases}$$

16

16

```
def dd(c_pass, c_fail):
    n = 2
    while 1:
        delta = listminus(c_fail, c_pass)
        deltas = split(delta, n); offset = 0; j = 0
        while j < n:
            i = (j + offset) % n
            next_c_pass = listunion(c_pass, deltas[i])
            next_c_fail = listminus(c_fail, deltas[i])
            if test(next_c_fail) == FAIL and n == 2:
                c_fail = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_fail) == PASS:
                c_pass = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_pass) == FAIL:
                c_fail = next_c_pass; n = 2; offset = 0; break
            elif test(next_c_fail) == FAIL:
                c_fail = next_c_fail; n = max(n - 1, 2); offset = i; break
            elif test(next_c_pass) == PASS:
                c_pass = next_c_pass; n = max(n - 1, 2); offset = i; break
            else:
                j = j + 1
        if j >= n:
            if n >= len(delta):
                return (delta, c_pass, c_fail)
            else:
                n = min(len(delta), n * 2)
```

17

17

Properties

number of tests t – worst case:

$$t = |\Delta|^2 + 7|\Delta| \quad \text{where} \quad \Delta = c_{✘} \setminus c_{✔}$$

number of tests t – best case
(no unresolved outcomes):

$$t \leq \log_2(\Delta)$$

size of difference – no unresolved outcomes

$$|c'_{✘} \setminus c'_{✔}| = 1$$

18

18

Applications

Input	Code Changes	Schedules

19

19

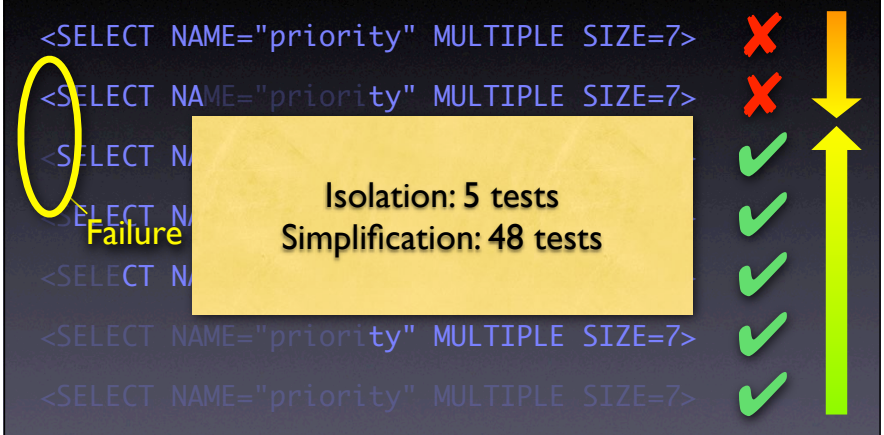
Applications

Input	Code Changes	Schedules

20

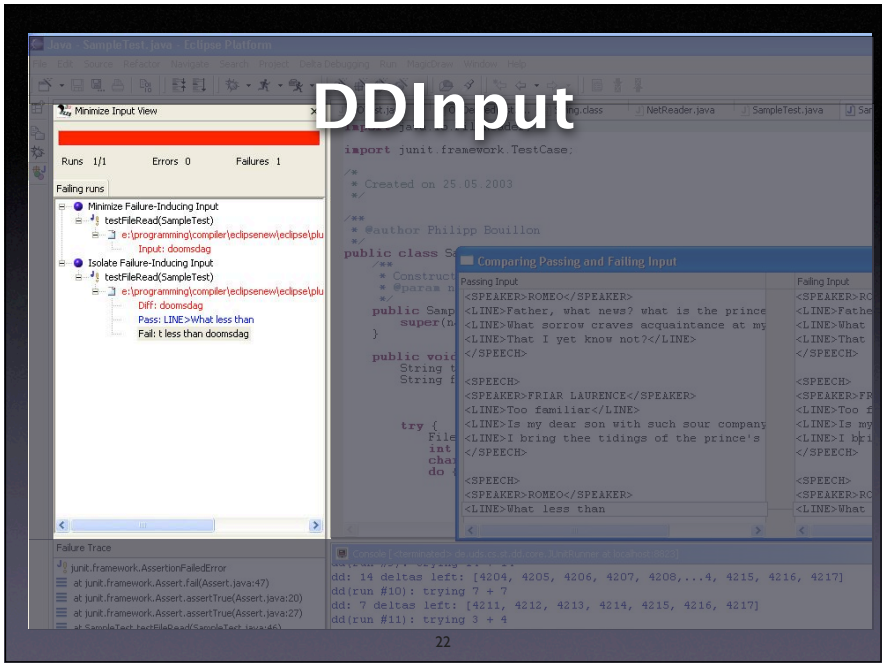
20

Isolating Input



21

21



22

Applications

Input	Code Changes	Schedules

23

23

Applications

Input	Code Changes	Schedules

24

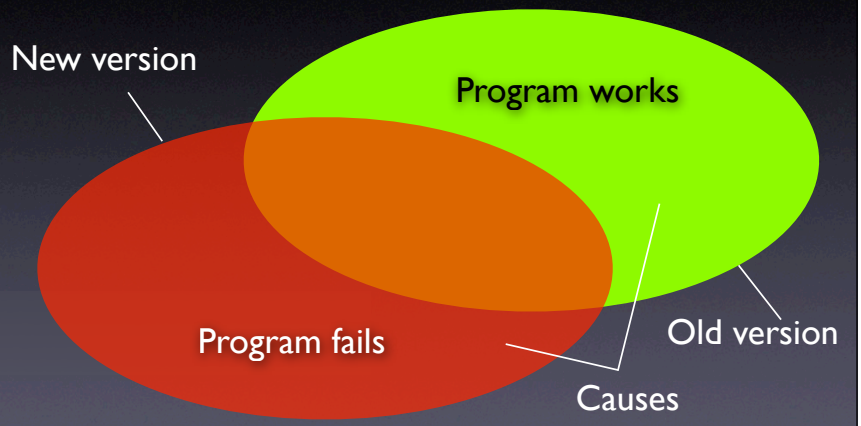
24

Code Changes

From: Brian Kahne <bkahne@ibmoto.com>
To: DDD Bug Report Address <bug-ddd@gnu.org>
Subject: Problem with DDD and GDB 4.17

When using DDD with GDB 4.16, the run command correctly uses any prior command-line arguments, or the value of "set args". However, when I switched to GDB 4.17, this no longer worked: If I entered a run command in the console window, the prior command-line options would be lost. [...]

Version Differences



What was Changed

```
$ diff -r gdb-4.16 gdb-4.17
diff -r gdb-4.16/COPYING gdb-4.17/COPYING
5c5
< 675 Mass Ave, Cambridge, MA 02139, USA
---
> 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
282c282
< Appendix: How to Apply These Terms to Your New Programs
---
> How to Apply These Terms to Your New Programs
```

...and so on for 178,200 lines (8,721 locations)

Challenges

- Granularity – within some large change, only a few lines may be relevant
- Interference – some (later) changes rely on other (earlier) changes
- Inconsistency – some changes may have to be combined to produce testable code

Delta debugging handles all this

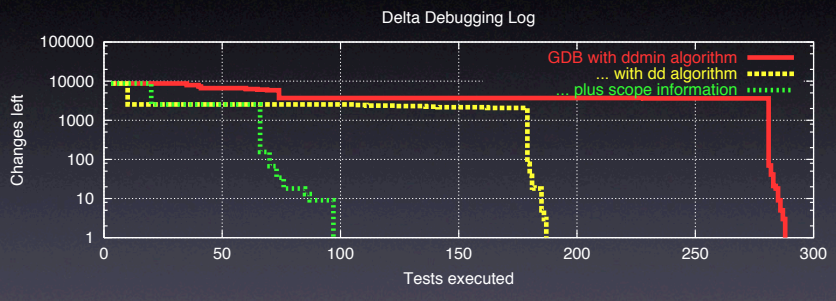
28

General Plan

- Decompose diff into changes per location (= 8,721 individual changes)
- Apply subset of changes, using PATCH
- Reconstruct GDB; build errors mean unresolved test outcome
- Test GDB and return outcome

29

Isolating Changes



- Result after 98 tests (= 1 hour)

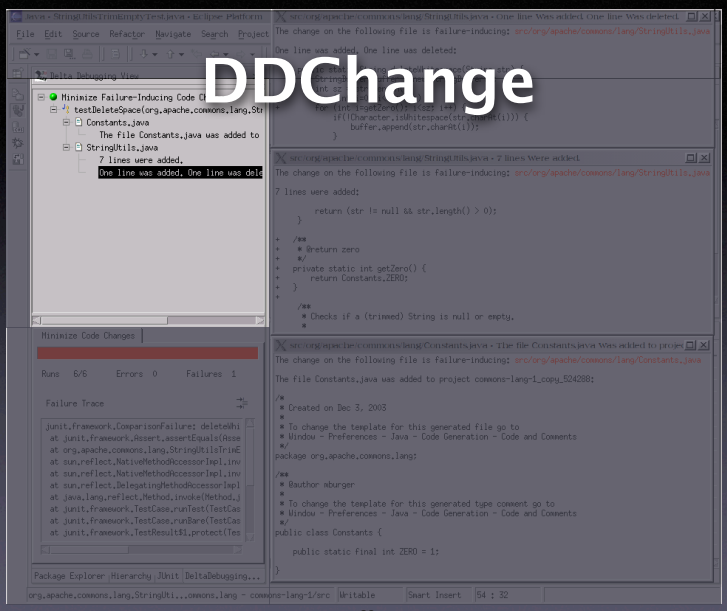
30

The Failure Cause

```
diff -r gdb-4.16/gdb/infcmd.c gdb-4.17/gdb/infcmd.c
1239c1278
< "Set arguments to give program being debugged when it is
started.\n
---
> "Set argument list to give program being debugged when
it is started.\n
```

- Documentation becomes GDB output
- DDD expects Arguments,
  but GDB outputs Argument list

DDChange



Optimizations

- History – group changes by creation time
- Reconstruction – cache several builds
- Grouping – according to scope
- Failure Resolution – scan error messages
  for possibly missing changes

Applications

Input	Code Changes	Schedules

34

34

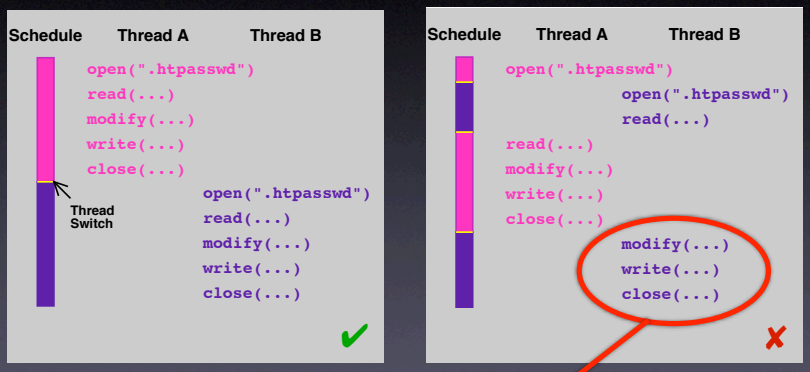
Applications

Input	Code Changes	Schedules

35

35

Thread Schedules

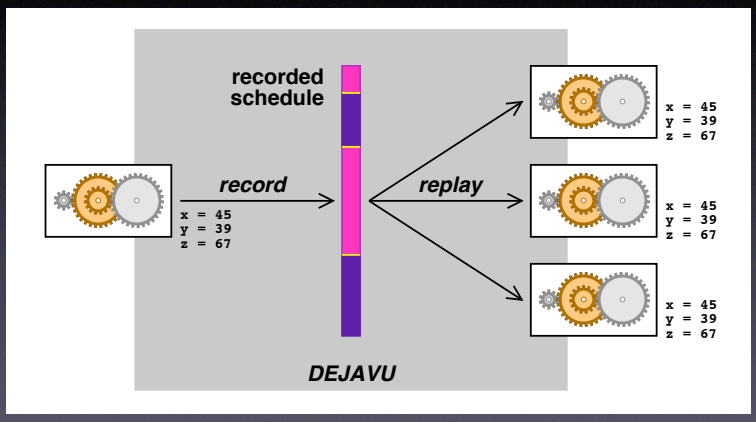


A's updates get lost!

36

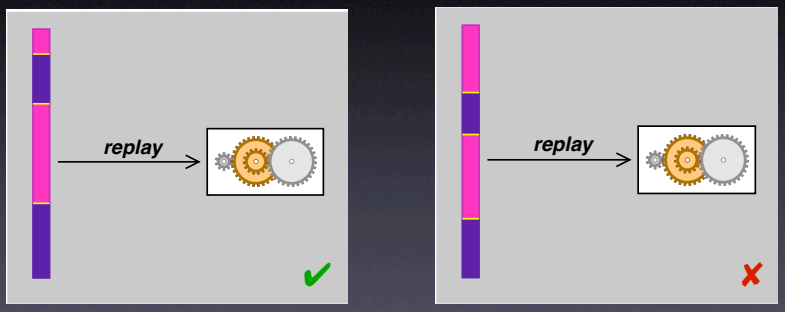
36

Record + Replay



37

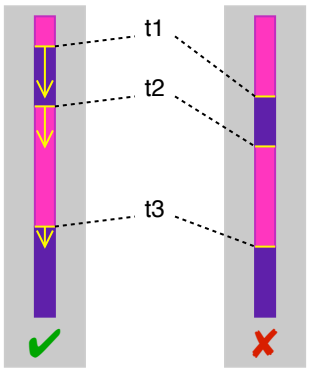
Schedules as Input



The schedule difference causes the failure!

38

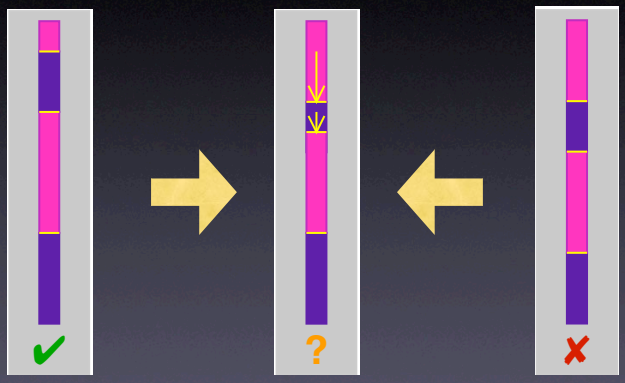
Finding Differences



- We start with runs ✔ and ✘
- We determine the differences Δ_i between thread switches t_i:
  - t_1 occurs in ✔ at "time" 254
  - t_1 occurs in ✘ at "time" 278
  - The difference $\Delta_1 = |278 - 254|$ induces a *statement interval:* the code executed between "time" 254 and 278
  - Same applies to t_2, t_3, etc.

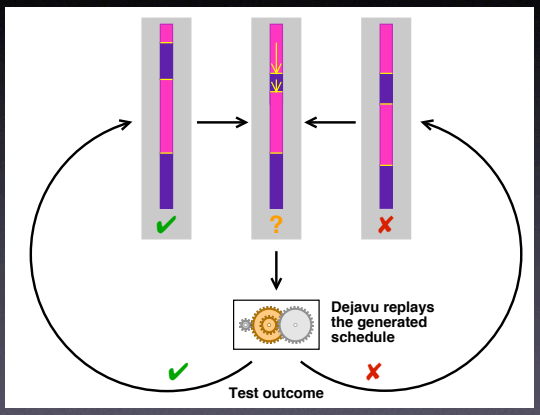
39

Isolating Differences



40

Isolating Differences



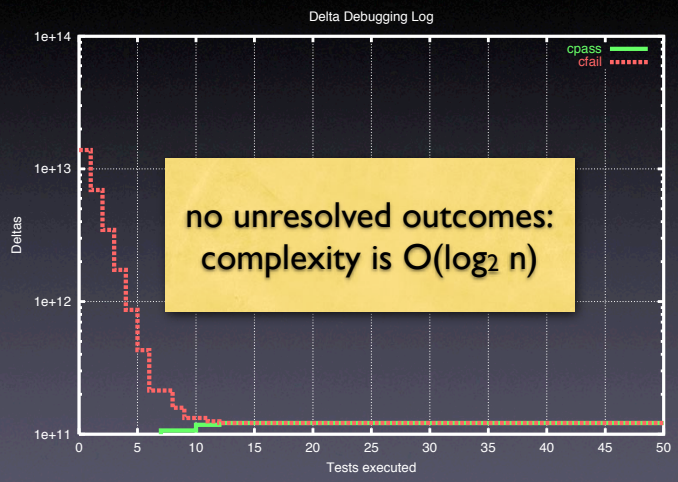
41

Example: Raytracer

- Raytracer program from Spec JVM98 suite
- Injected a simple *race condition*
- Set up *automated test* + *random schedules*
- Obtained *passing* and *failing* schedule
- 3,842,577,240 differences, each moving a thread switch by ±1 *yield point* (time unit)

42

Isolating Schedules



no unresolved outcomes:
complexity is $O(\log_2 n)$

43

The Failure Cause

```
25 public class Scene { …
44     private static int ScenesLoaded = 0;
45     (more methods…)
81     private
82     int LoadScene(String filename) {
84         int OldScenesLoaded = ScenesLoaded;
85         (more initializations…)
91         infile = new DataInputStream(…);
92         (more code…)
130        ScenesLoaded = OldScenesLoaded + 1;
131        System.out.println("" +
                 ScenesLoaded + " scenes loaded.");
132        …
134    }
135    …
733 }
```


44

General Issues

- How do we choose the *alternate world*?
- How do we *decompose* the configuration?
- How do we know *a* failure is *the* failure?
- How do we disambiguate *multiple causes?*
- How do I get to the defect?

45

Concepts

- ★ To isolate failure causes automatically, use
  - an *automated test case*
  - a means to *narrow down the difference*
  - a *strategy* for proceeding.
- ★ One possible strategy is Delta Debugging.

46

Concepts (2)

- ★ Delta Debugging can isolate failure causes
  - in the (general) *input*
  - in the *version history*
  - in *thread schedules*
- ★ Every such cause implies a *fix* – but not necessarily a correction.

47

48