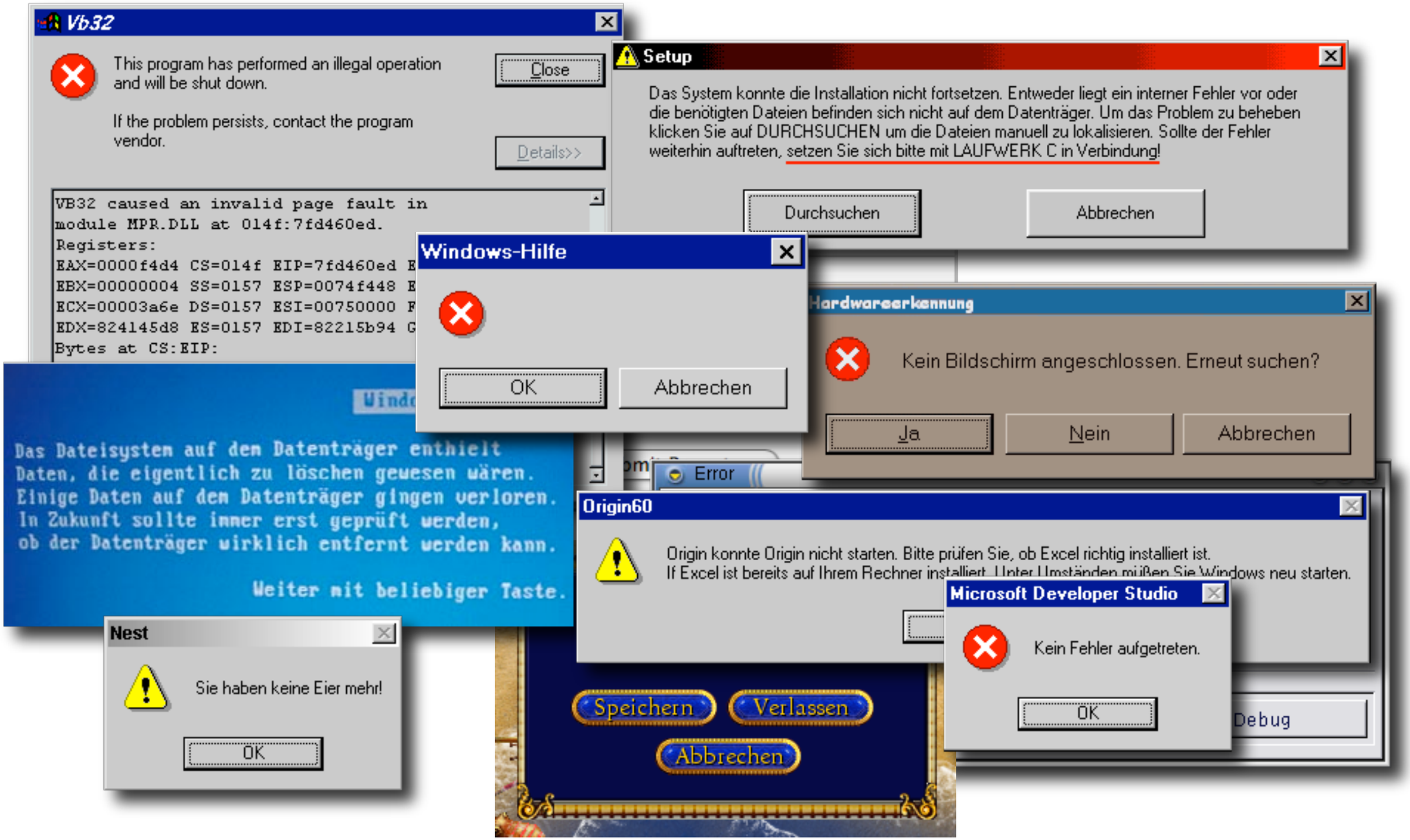
Systematic Debugging

Software Engineering

Andreas Zeller • Saarland University

The Problem


Vb32
This program has performed an illegal operation and will be shut down.
If the problem persists, contact the program vendor.
Close
Details>>
VB32 caused an invalid page fault in
module MPR.DLL at 014f:7fd460ed.
Registers:
EAX=0000f4d4 CS=014f EIP=7fd460ed E
EBX=00000004 SS=0157 ESP=0074f448 E
ECX=00003a6e DS=0157 ESI=00750000 F
EDX=824145d8 ES=0157 EDI=82215b94 G
Bytes at CS:EIP:
Setup
Das System konnte die Installation nicht fortsetzen. Entweder liegt ein interner Fehler vor oder die benötigten Dateien befinden sich nicht auf dem Datenträger. Um das Problem zu beheben klicken Sie auf DURCHSUCHEN um die Dateien manuell zu lokalisieren. Sollte der Fehler weiterhin auftreten, setzen Sie sich bitte mit LAUFWERK C in Verbindung!
Durchsuchen
Abbrechen
Windows-Hilfe
OK
Abbrechen
Hardwareerkennung
Kein Bildschirm angeschlossen. Erneut suchen?
Ja
Nein
Abbrechen
Error
Origin60
Origin konnte Origin nicht starten. Bitte prüfen Sie, ob Excel richtig installiert ist.
If Excel ist bereits auf Ihrem Rechner installiert. Unter Umständen müßen Sie Windows neu starten.
Microsoft Developer Studio
Kein Fehler aufgetreten.
OK
Debug
Das Dateisystem auf den Datenträger enthielt
Daten, die eigentlich zu löschen gewesen wären.
Einige Daten auf dem Datenträger gingen verloren.
In Zukunft sollte immer erst geprüft werden,
ob der Datenträger wirklich entfernt werden kann.
Weiter mit beliebiger Taste.
Nest
Sie haben keine Eier mehr!
OK
Speichern
Verlassen
Abbrechen

Facts on Debugging

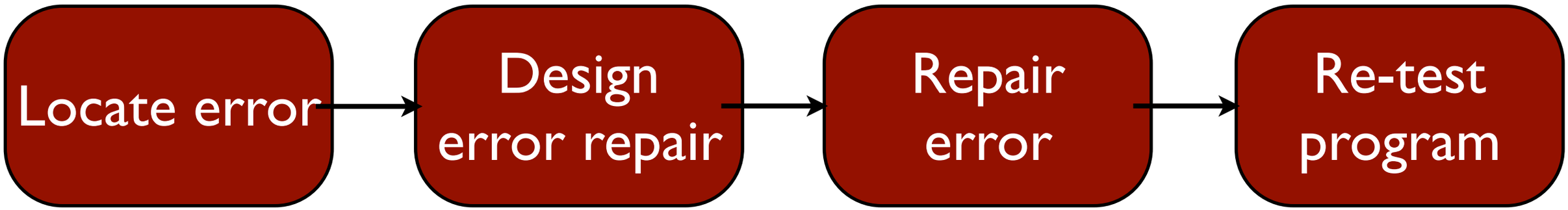
- Software bugs cost ~60 bln US$/yr in US
- Improvements could reduce cost by 30%
- Validation (including debugging) can easily take up to 50-75% of the development time
- When debugging, some people are *three times* as efficient than others

Boskoop: bug (~/tmp/bug) <zeller.zeller> — bash — 80x24 — ⌘1

```
$ ls
bug.c
$ gcc-2.95.2 -O bug.c
gcc: Internal error: program cc1 got fatal signal 11
Segmentation fault
$
```

How to Debug

(Sommerville 2004)



The Process

T rack the problem

R eproduce

A utomate

F ind Origins

F ocus

I solate

C orrect

T R A F F I C

Tracking Problems

trac
Integrated SCM & Project Management

Search

Login | Settings | Help/Guide | About Trac

Wiki | Timeline | Roadmap | Browse Source | View Tickets | New Ticket | Search

This report: Edit | Copy | Delete | New Report | Custom Query

{9} Time Tracking (7 matches)

Ticket	Planned	Spent	Remaining	Accuracy	Customer	Summary	Component	Status
#6	10h		10h	0.0	milestone1	asdf	component1	new
#5	2h	4h	0h	2.0	milestone1	234	component1	new
#4				0.0	milestone1	yxcv	component1	new
#3	4h	4h		0.0	milestone1	test3	component1	closed
#2	4h	2h	2h	0.0	milestone1	test2	component1	new
#1	8h	7.0h	3.0h	2.0	milestone1	test 1	component1	new
#7	1h			-1.0	milestone2	3452345	component1	new

Note: See TracReports for help on using and creating reports.

Download in other formats:

XML RSS Feed | Comma-delimited Text | Tab-delimited Text | SQL Query

trac POWERED

Powered by **Trac 0.9.pre**
By Edgewall Software.

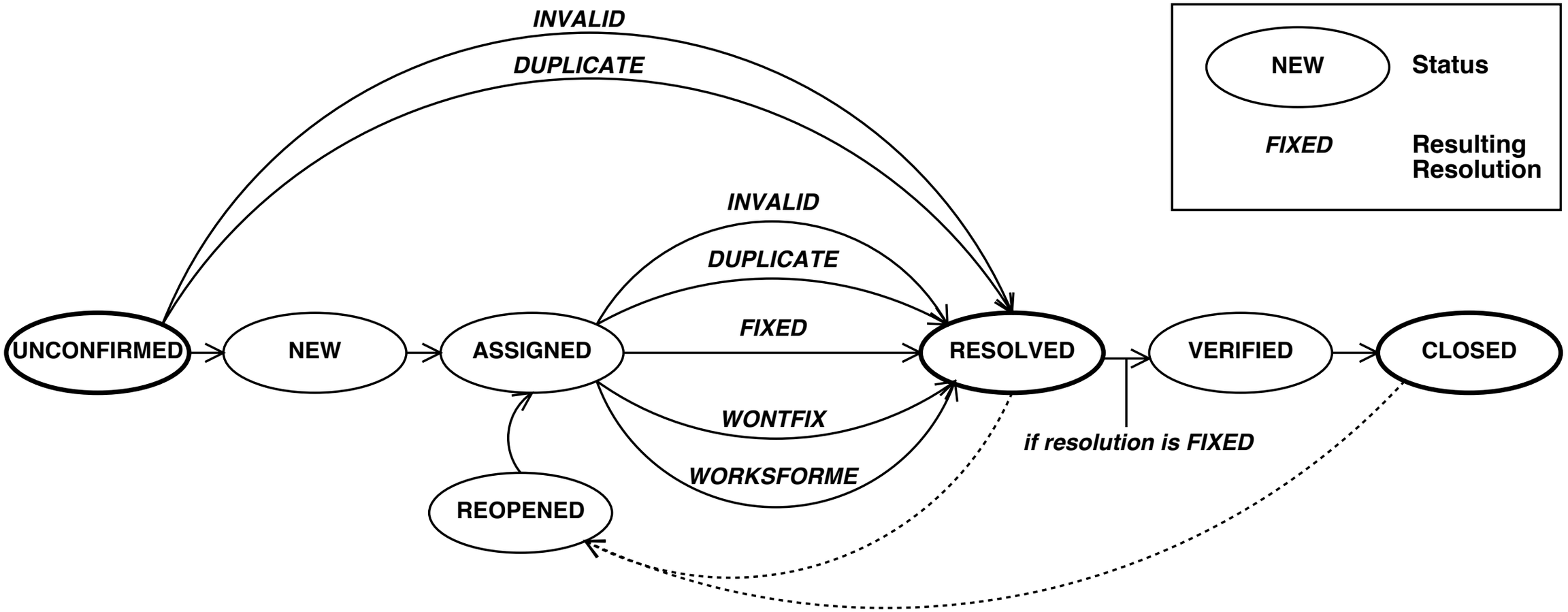
Visit the Trac open source project at
http://trac.edgewall.com/

Tracking Problems

- Every problem gets entered into a *problem database*
- The *priority* determines which problem is handled next
- The product is ready when all problems are resolved

TRAFFIC

Problem Life Cycle



TRAFFIC

Reproduce

Randomness

Operating System

Communication

Concurrency

Interaction

Physics

Data

Debugger

TRAFFIC

Automate

```
// Test for host
public void testHost() {
  int noPort = -1;
  assertEquals(askigor_url.getHost(), "www.askigor.org");
  assertEquals(askigor_url.getPort(), noPort);
}

// Test for path
public void testPath() {
  assertEquals(askigor_url.getPath(), "/status.php");
}

// Test for query part
public void testQuery() {
  assertEquals(askigor_url.getQuery(), "id=sample");
}
```

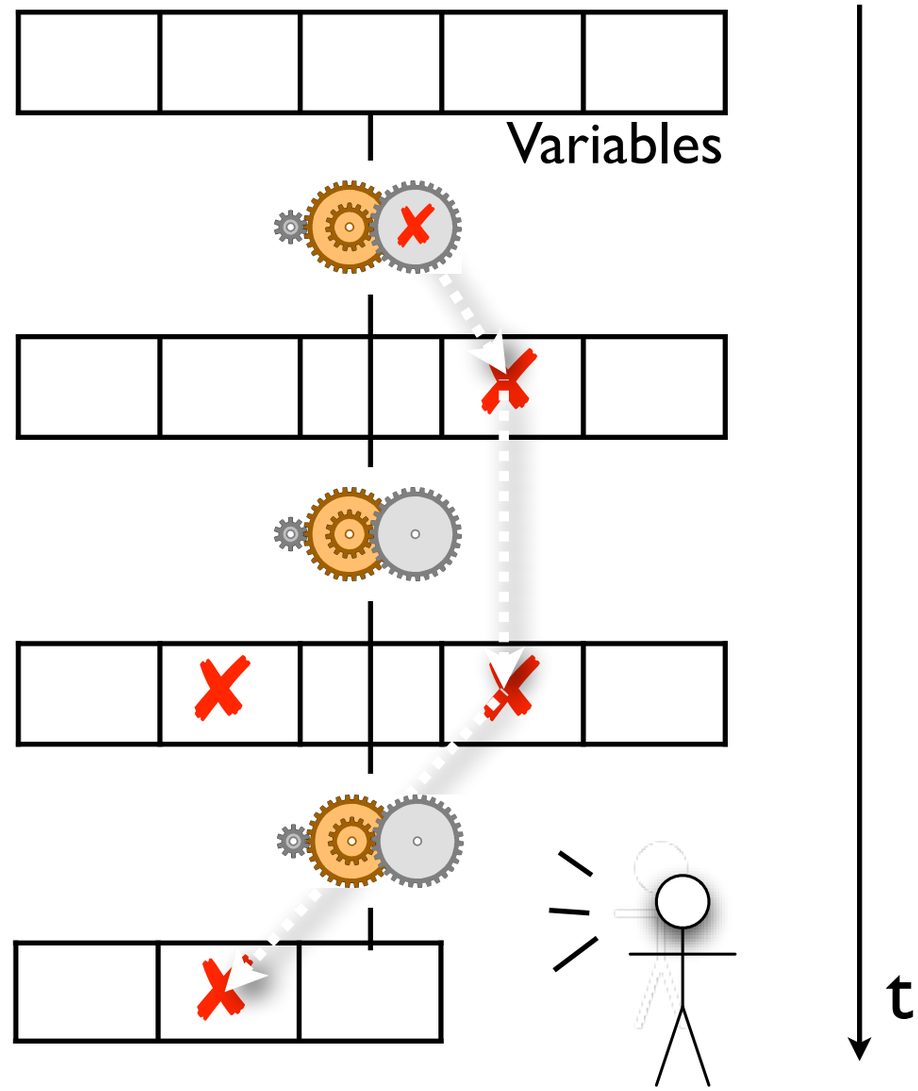
TRAFFIC

Automate

- Every problem should be *reproducible automatically*
- Achieved via appropriate (unit) tests
- After each change, we re-run the tests

Finding Origins

1. The programmer creates a *defect* in the code.
2. When executed, the defect creates an *infection*.
3. The infection *propagates*.
4. The infection causes a *failure*.

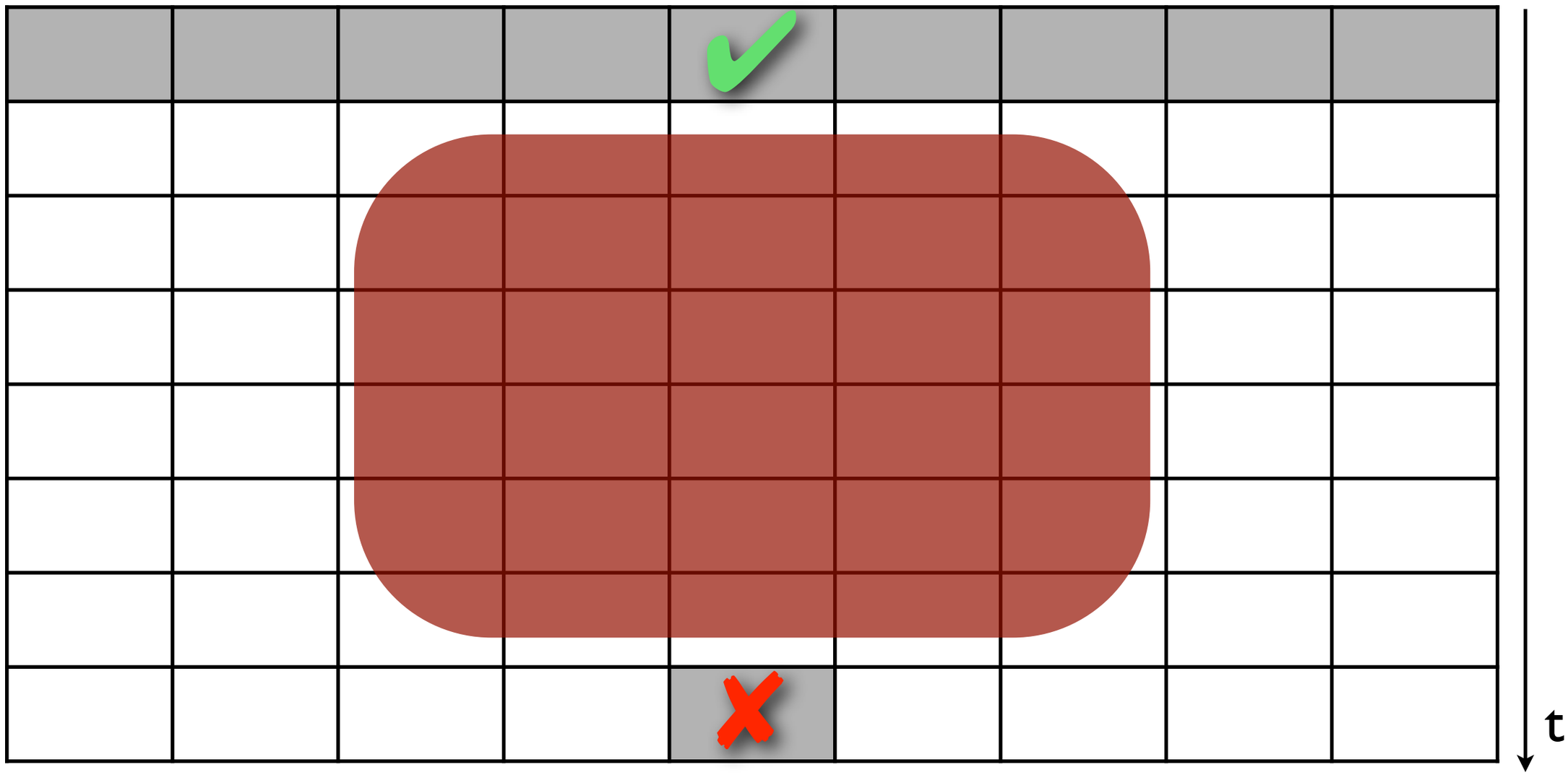


This infection chain must be traced back – and broken.

Not every defect creates an infection – not every infection results in a failure

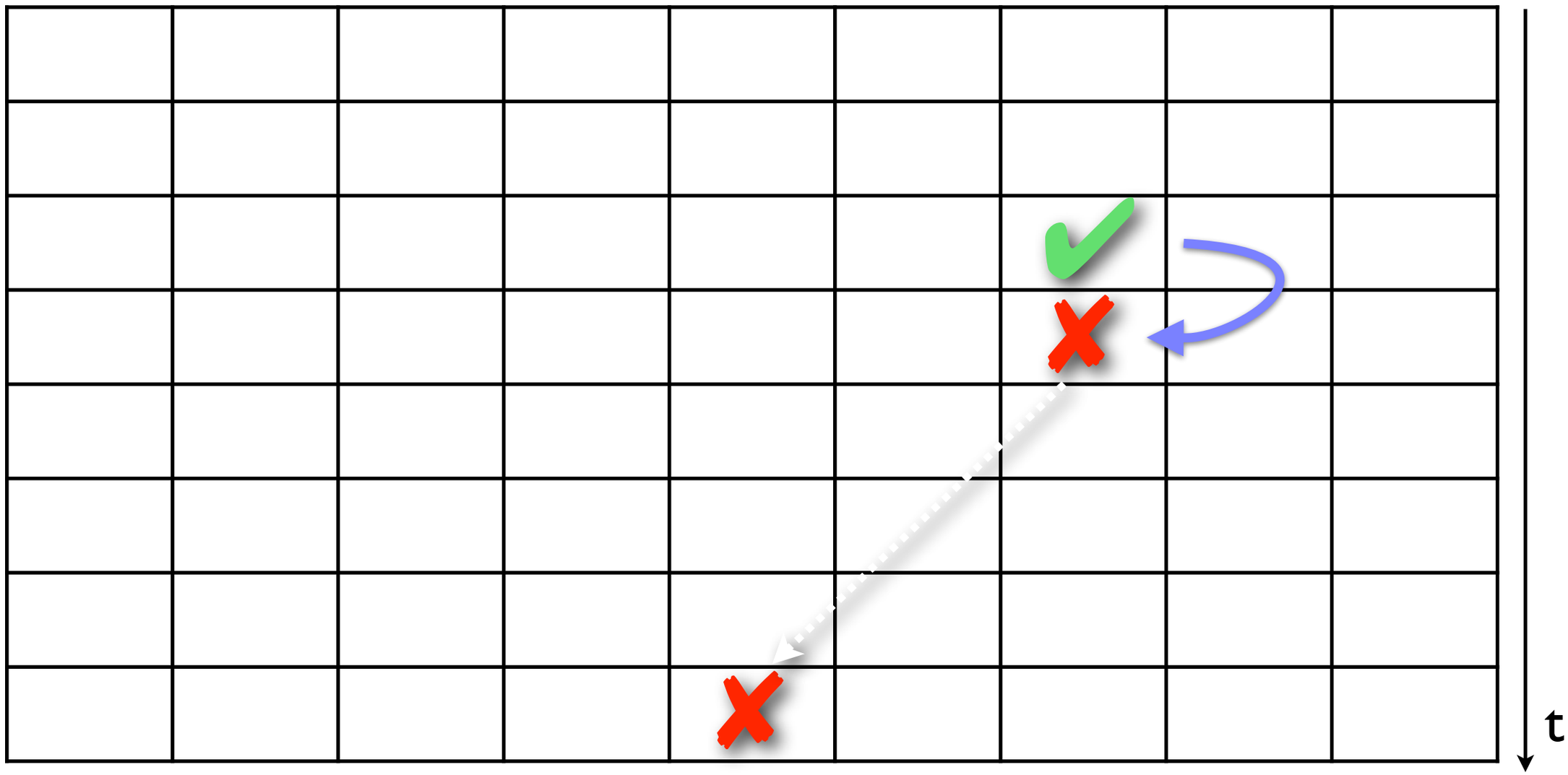
TRAFFIC

Finding Origins



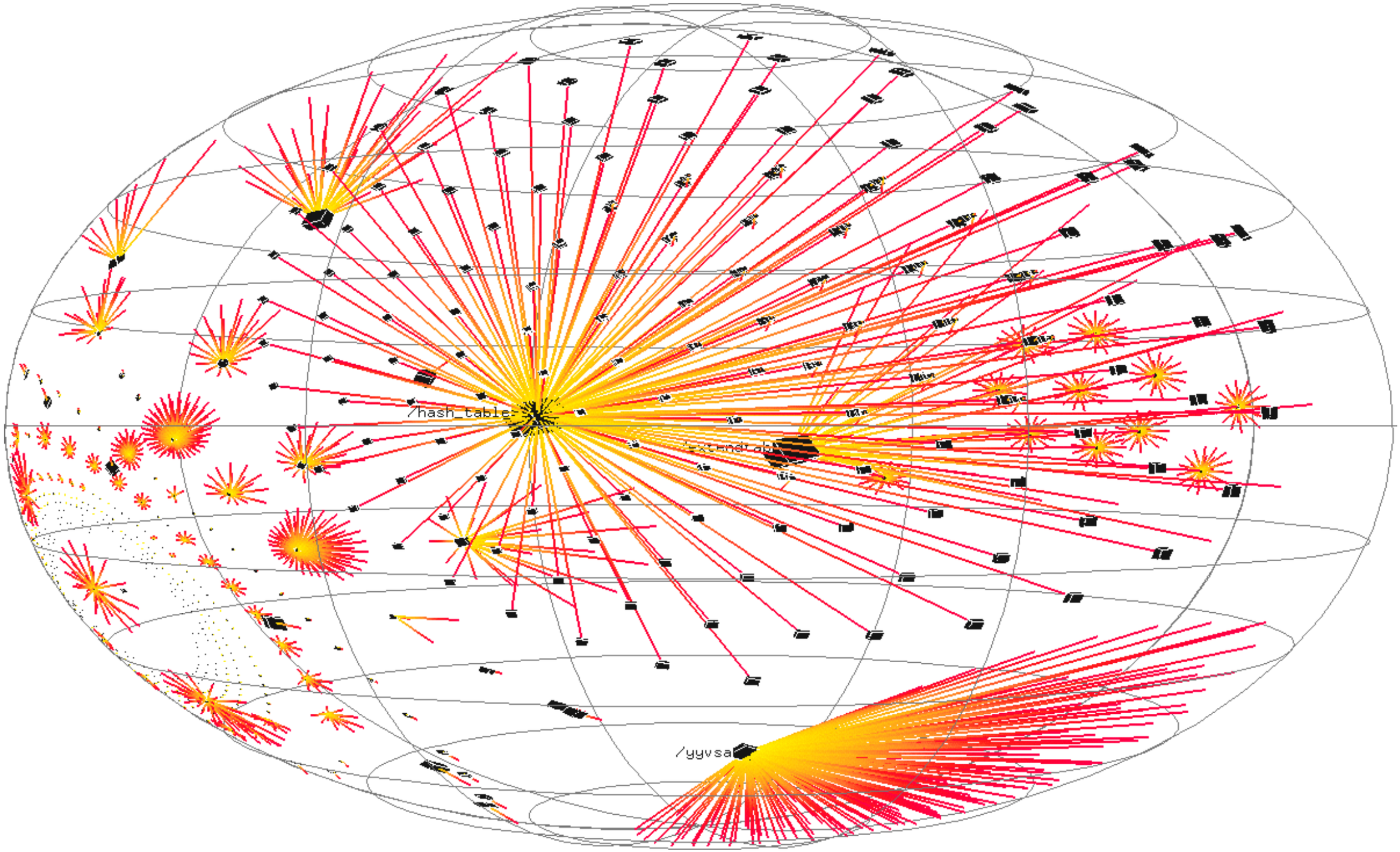
TRAFFIC

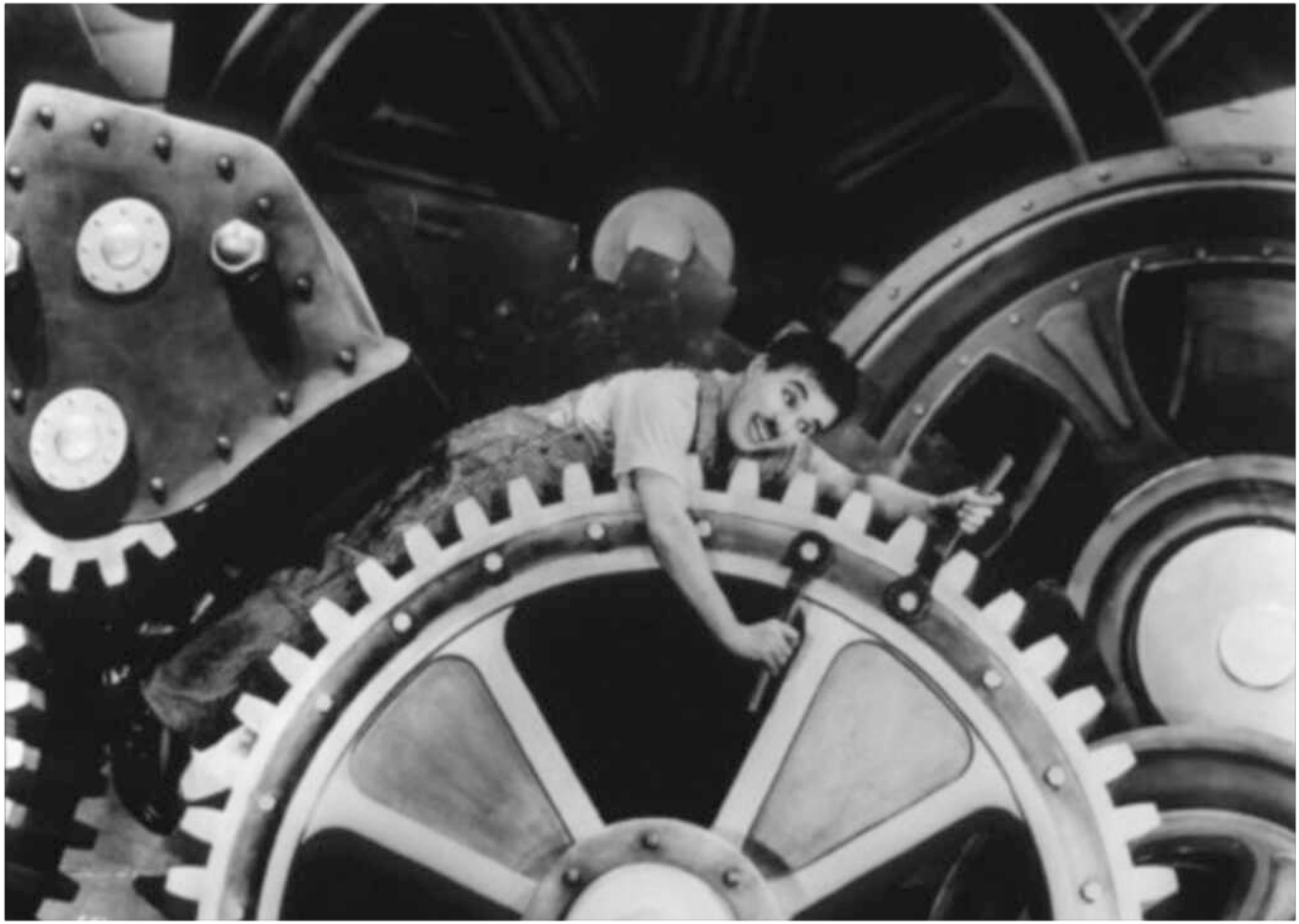
The Defect



TRAFFIC

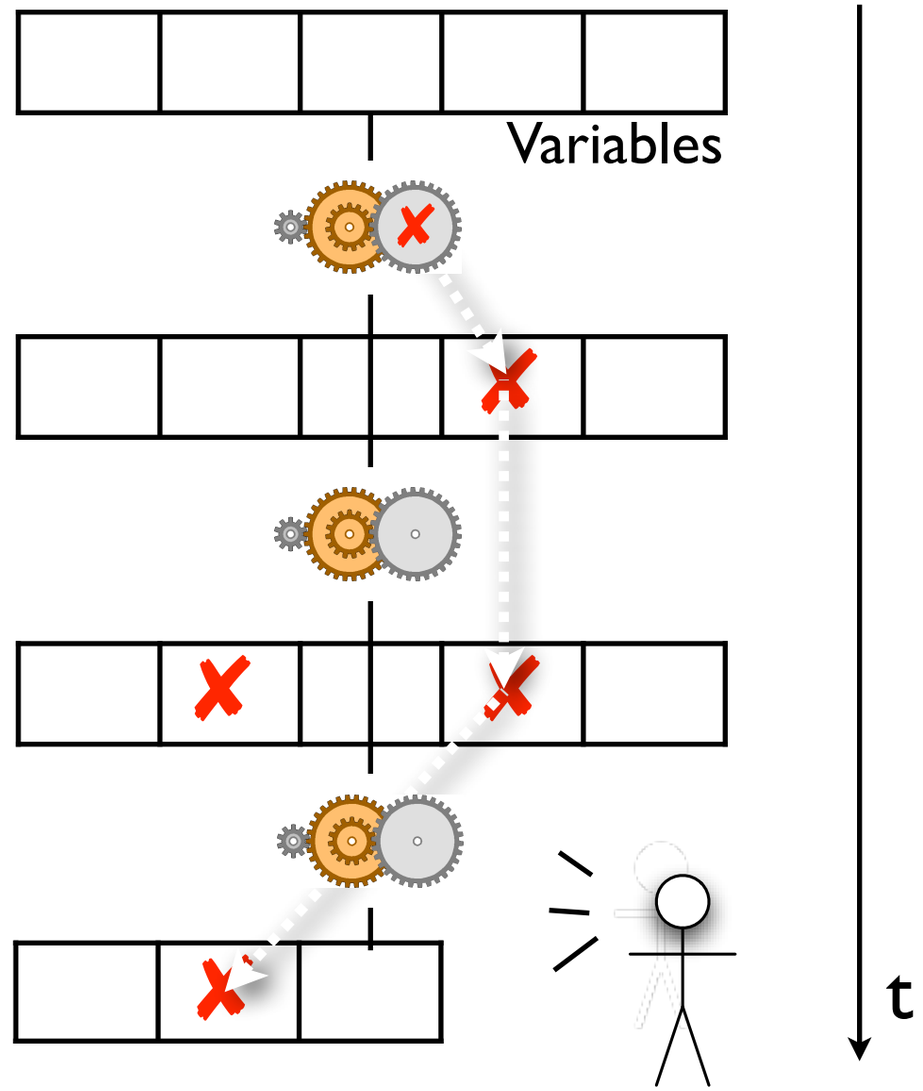
A Program State



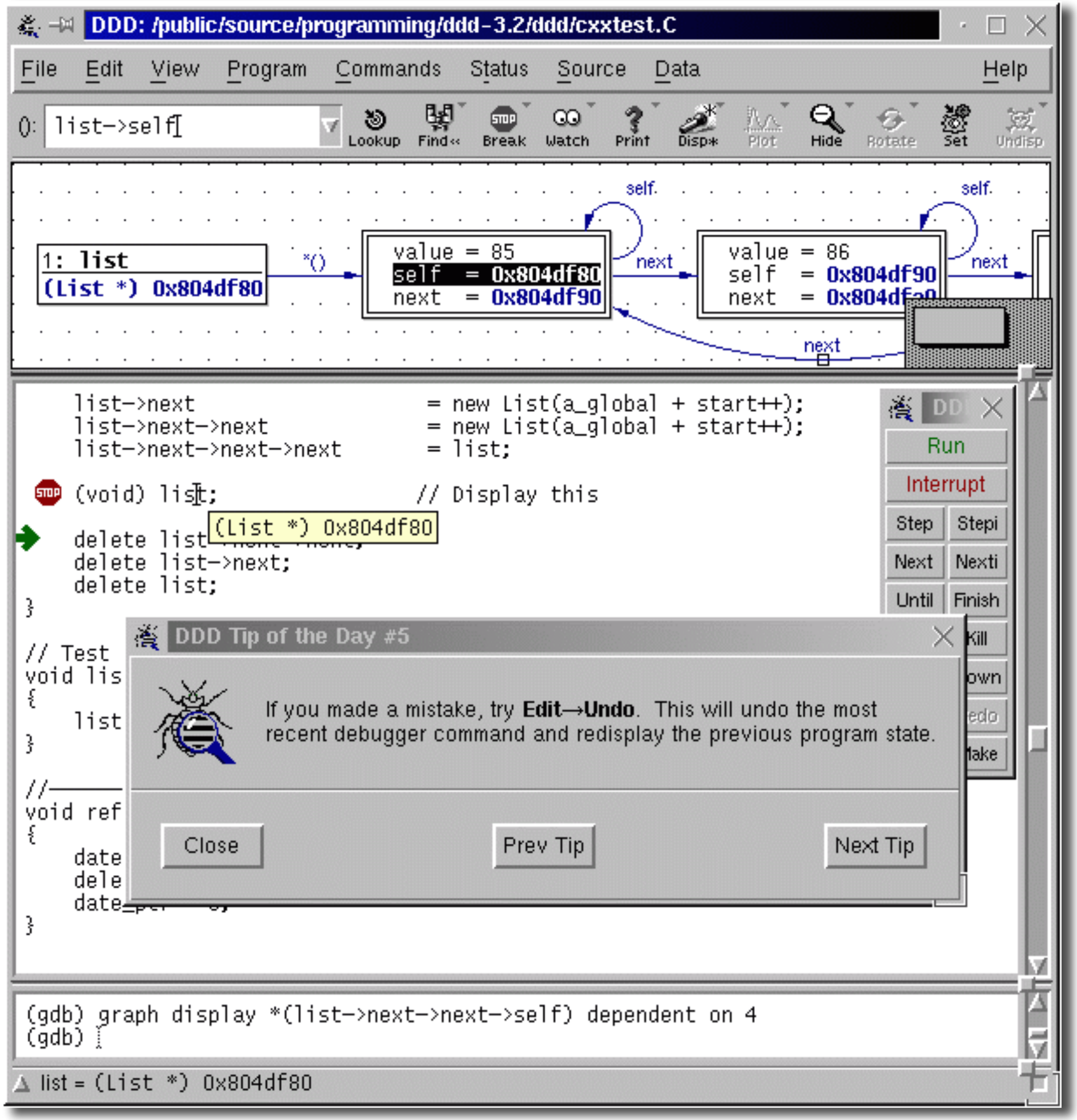


Finding Origins

1. We start with a *known infection* (say, at the failure)
2. We search the infection in the *previous state*

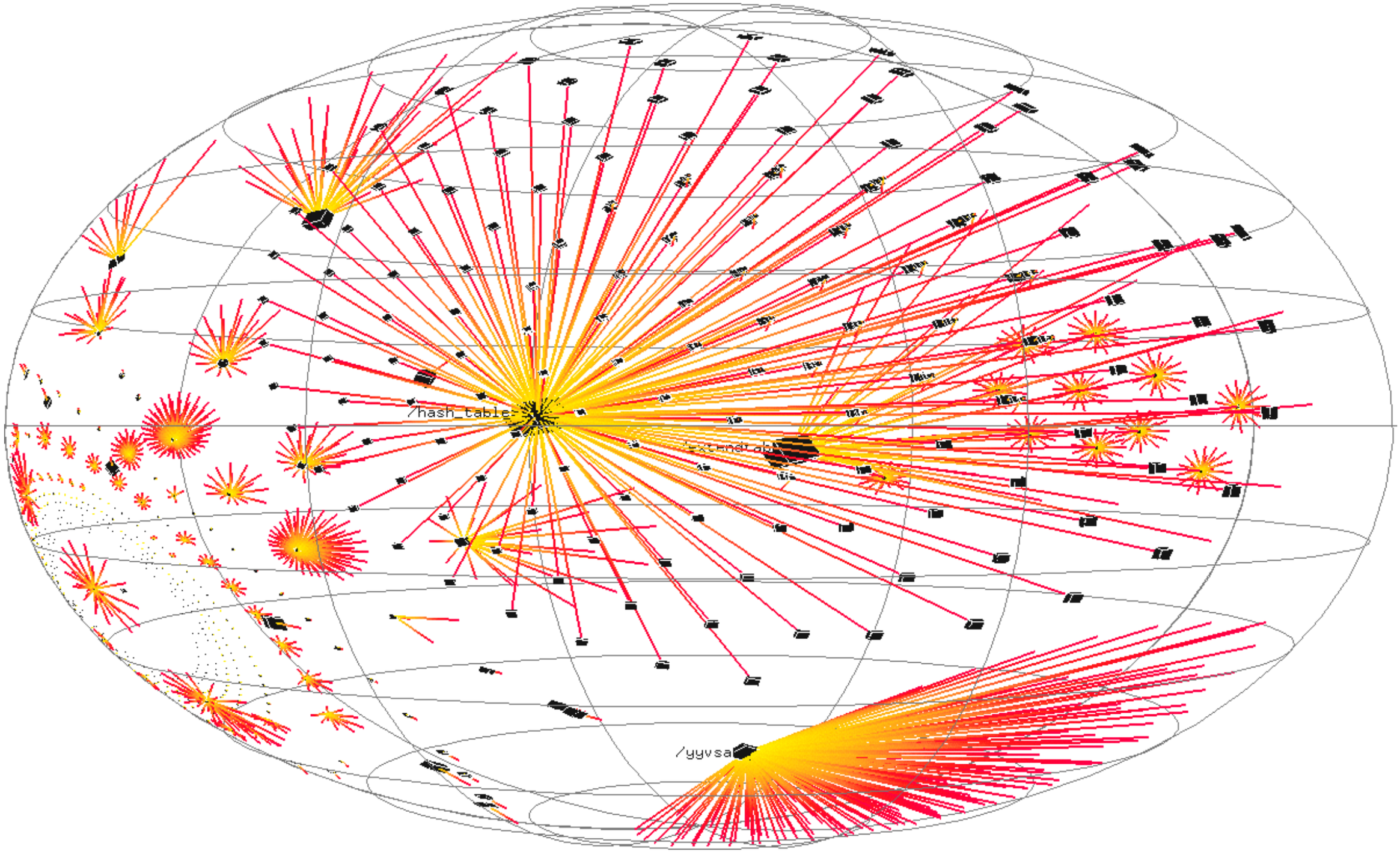


TRAFFIC


DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
(): list->self
Lookup Find« Break Watch Print Disp* Plot Hide Rotate Set Undisp
self.
self.
1: list
(List *) 0x804df80
*()
value = 85
self = 0x804df80
next = 0x804df90
next
value = 86
self = 0x804df90
next
next
list->next = new List(a_global + start++);
list->next->next = new List(a_global + start++);
list->next->next->next = list;
(void) list; // Display this
(List *) 0x804df80
delete list;
delete list->next;
delete list;
}
// Test
void lis
{
list
}
//
void ref
{
date
dele
date
}
DD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Kill
own
edo
lake
DDD Tip of the Day #5
If you made a mistake, try Edit→Undo. This will undo the most recent debugger command and redisplay the previous program state.
Close
Prev Tip
Next Tip
(gdb) graph display *(list->next->next->self) dependent on 4
(gdb)
list = (List *) 0x804df80

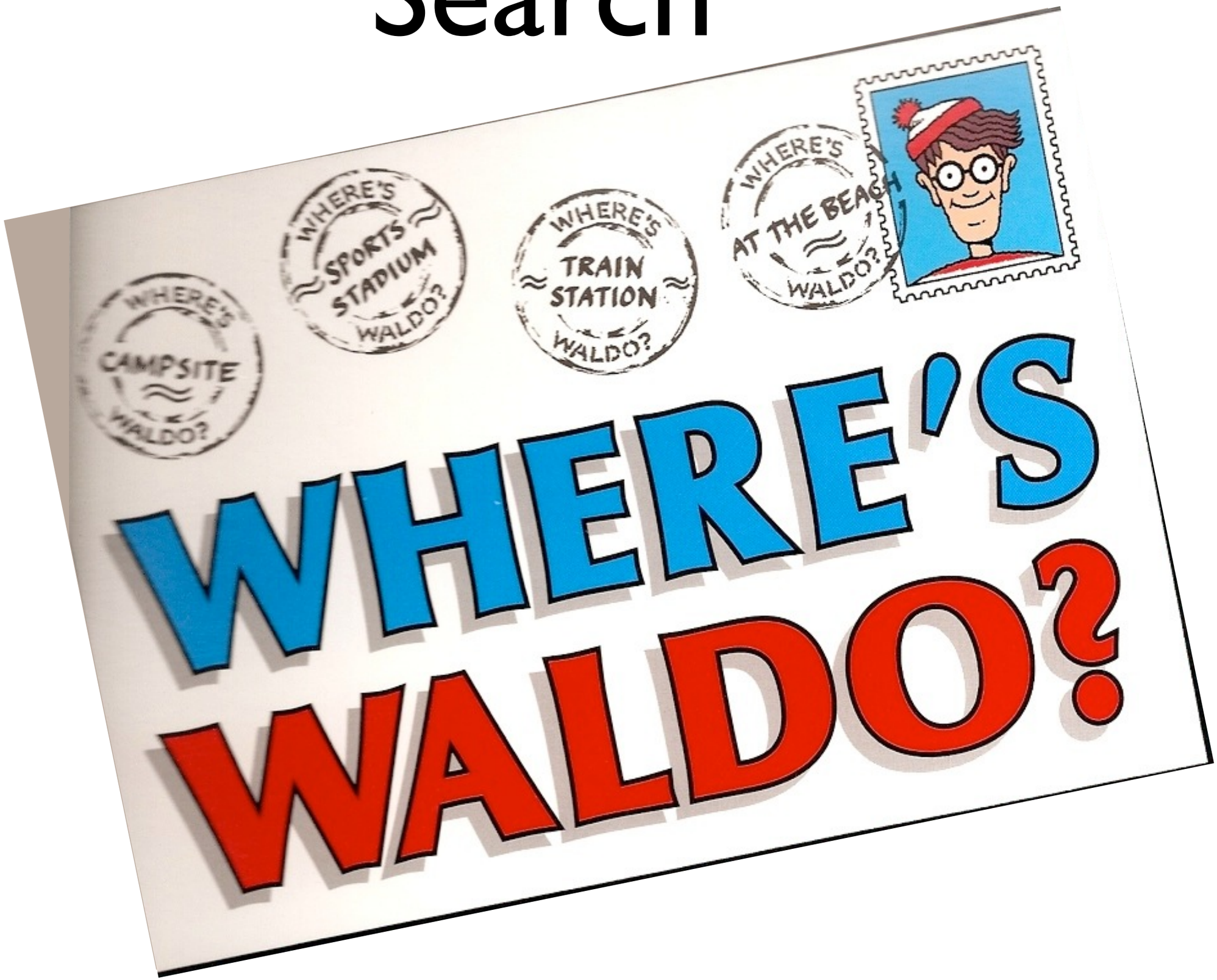
TRAFFIC

A Program State



TRAFFIC

Search



Focus

During our search for infection, we focus upon locations that

- *are possibly wrong*
  (e.g., because they were buggy before)
- *are explicitly wrong*
  (e.g., because they violate an *assertion*)

Assertions are the best way to find infections!

Finding Infections

```
class Time {
public:
     int hour();      // 0..23
     int minutes();   // 0..59
     int seconds();   // 0..60 (incl. leap seconds)

     void set_hour(int h);
     …
}
```

Every time between 00:00:00 and 23:59:60 is valid

Finding Origins

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}

void Time::set_hour(int h)
{
    assert (sane());  // Precondition
    …
    assert (sane());  // Postcondition
}
```

Finding Origins

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}
```

sane() is the *invariant* of a Time object:

- valid *before* every public method
- valid *after* every public method

Finding Origins

- Precondition fails = Infection *before* method
- Postcondition fails = Infection *after* method
- All assertions pass = no infection

```
void Time::set_hour(int h)
{
    assert (sane());  // Precondition
    …
    assert (sane());  // Postcondition
}
```

TRAFFIC

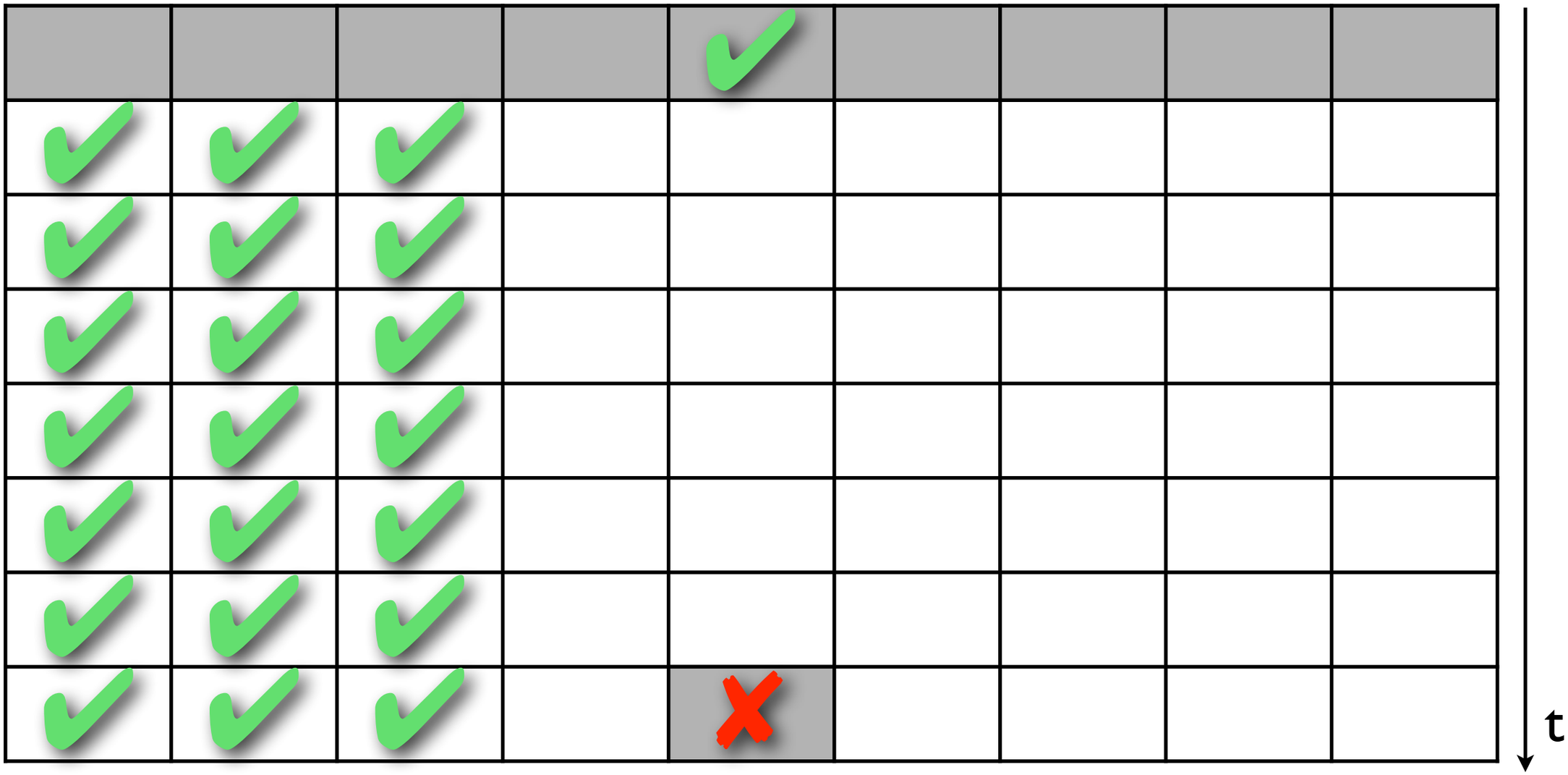
Complex Invariants

```
class RedBlackTree {
  …
  boolean sane() {
      assert (rootHasNoParent());
      assert (rootIsBlack());
      assert (redNodesHaveOnlyBlackChildren());
      assert (equalNumberOfBlackNodesOnSubtrees());
      assert (treeIsAcyclic());
      assert (parentsAreConsistent());

      return true;
  }
}
```

TRAFFIC

Assertions



Focusing

- All possible influences must be checked
- Focusing on most likely candidates
- Assertions help in finding infections fast

Isolation

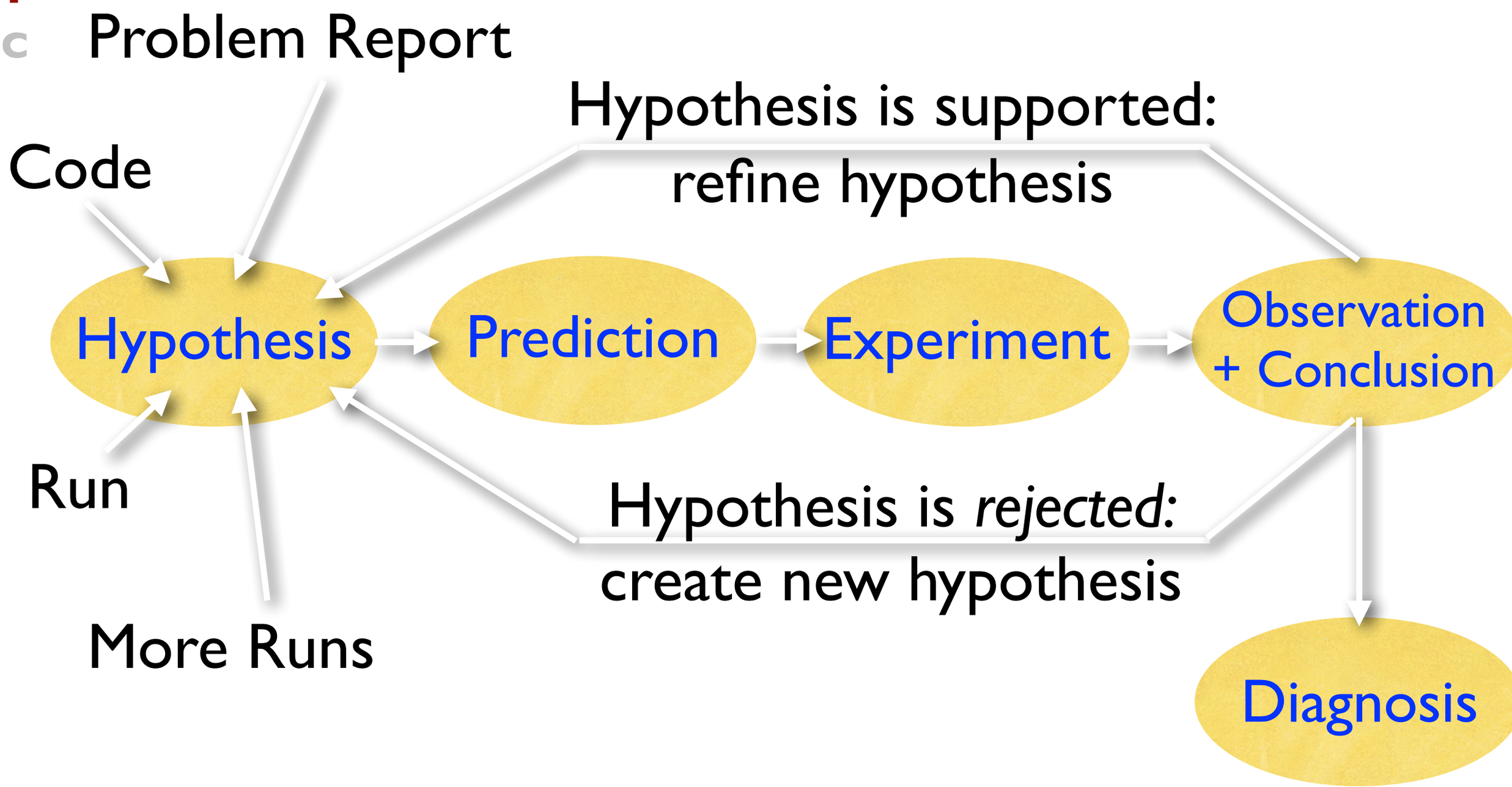
- Failure causes should be *narrowed down systematically*
- Use *observation* and *experiments*

TRAFFIC

Scientific Method

1. Observe some aspect of the universe.
2. Invent a *hypothesis* that is consistent with the observation.
3. Use the hypothesis to make *predictions*.
4. Tests the predictions by experiments or observations and modify the hypothesis.
5. Repeat 3 and 4 to refine the hypothesis.

Scientific Method



TRAFFIC


MASTERMIND
MASTERMIND
The Challenging Game of Logic and Deduction
Break the hidden code!
New sleek design with
improved built in storage
tray and code shield!
Pressman
MASTERMIND

Explicit Hypotheses

Hypothesis	The executi… uses a[0] = 0
Prediction	At Lin… ld hold.
Experiment	…e 37.
Observati…	…s as predicted.
Conclusion	…hesis is *confirmed.*

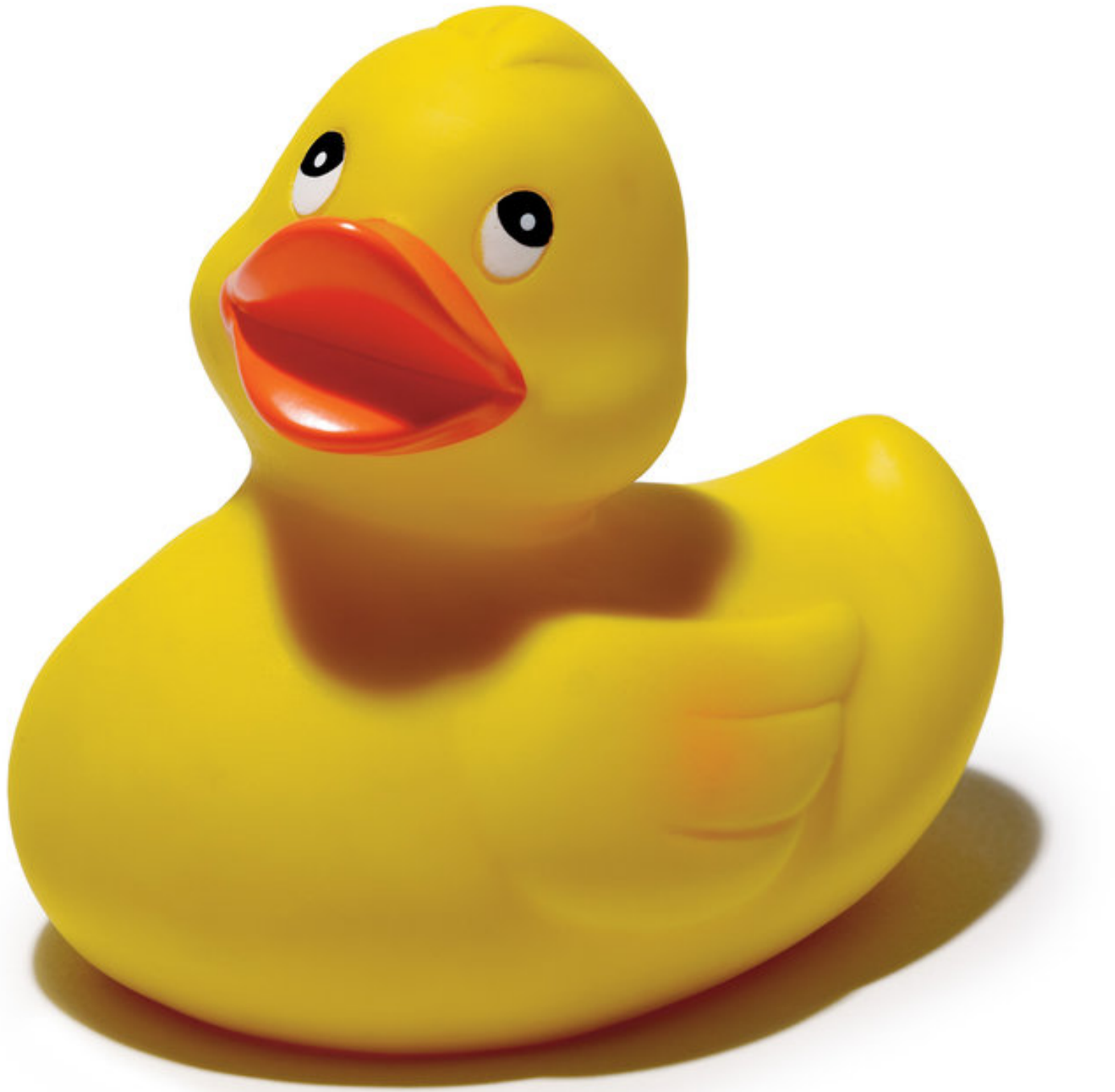
Keeping every… memory is like p… mastermind blind!

TRAFFIC

Explicit Hypotheses

TRAFFIC

Rubberducking



Isolate

- We repeat the search for infection origins until we found the defect
- We proceed *systematically* along the scientific method
- *Explicit steps* guide the search – and make it repeatable at any time

Correction

Before correcting the defect, we must check whether the defect

- actually is an *error* and
- *caus*es the failure

Only when we understood both, can we correct the defect

TRAFFIC

The Devil's Guide to Debugging

Find the defect by guessing:

- Scatter debugging statements everywhere
- Try changing code until something works
- Don't back up old versions of the code
- Don't bother understanding what the program should do

TRAFFIC

The Devil's Guide to Debugging

Don't waste time understanding the problem.

- Most problems are trivial, anyway.

TRAFFIC

The Devil's Guide to Debugging

Use the most obvious fix.

- Just fix what you see:

```
x = compute(y)
// compute(17) is wrong – fix it
if (y == 17)
    x = 25.15
```

Why bother going into compute()?

TRAFFIC

Successful Correction

Homework

- Does the failure no longer occur?
  (If it does still occur, this should come as a big surprise)
- Did the correction introduce new problems?
- Was the same mistake made elsewhere?
- Did I commit the change to version control and problem tracking?

The Process

T rack the problem

R eproduce

A utomate

F ind Origins

F ocus

I solate

C orrect

WINNER OF JOLT PRODUCTIVITY AWARD

ANDREAS ZELLER

WHY PROGRAMS FAIL

A GUIDE TO SYSTEMATIC DEBUGGING

SECOND EDITION

MORGAN KAUFMANN

Which hypotheses are consistent with our observations so far?

- [x] Double quotes are stripped from ~~tagged~~ input

input	expected	output	
"foo"	"foo"	foo	X
"bar"	"bar"	bar	X
""	""	(empty)	X

The error is due to tag being set.

Automated Debugging (Udacity)

Summary

The Process

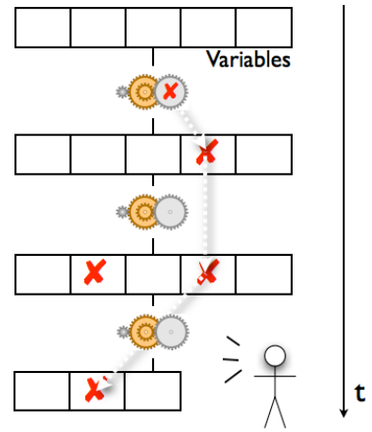
T rack the problem
R eproduce
A utomate
F ind Origins
F ocus
I solate
C orrect

Finding Origins

T R A **F** F I C

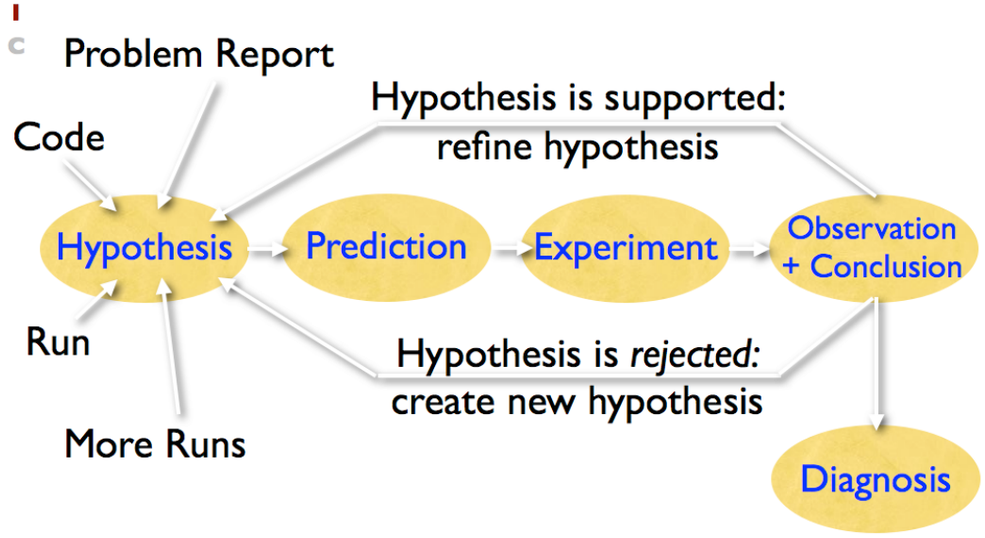
1. The programmer creates a *defect* in the code.
2. When executed, the defect creates an *infection*.
3. The infection *propagates*.
4. The infection causes a *failure*.

This infection chain must be traced back – and broken.



Scientific Method

T R A F F **I** C



Online Course on Debugging