

**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=82215b9
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 das Problem weiterhin auftreten, setzen Sie sich bitte mit LAUFWERK C in Verbindung!

Durchsuchen

Abbrechen

**Windows-Hilfe**

OK

Abbrechen

**Hardwareerkennung**

Kein Bildschirm angeschlossen. Erneut versuchen

Ja

Nein

# Fehlersuche

Software-Praktikum  
Andreas Zeller, Universität des Saarlandes

**Nest**

Sie haben keine Eier mehr!

OK

Origin konnte Origin nicht starten. Bitte prüfen Sie, ob Excel richtig installiert ist. If Excel ist bereits auf Ihrem Rechner installiert, | Inter | Umständen müssen Sie v

Speichern

Verlassen

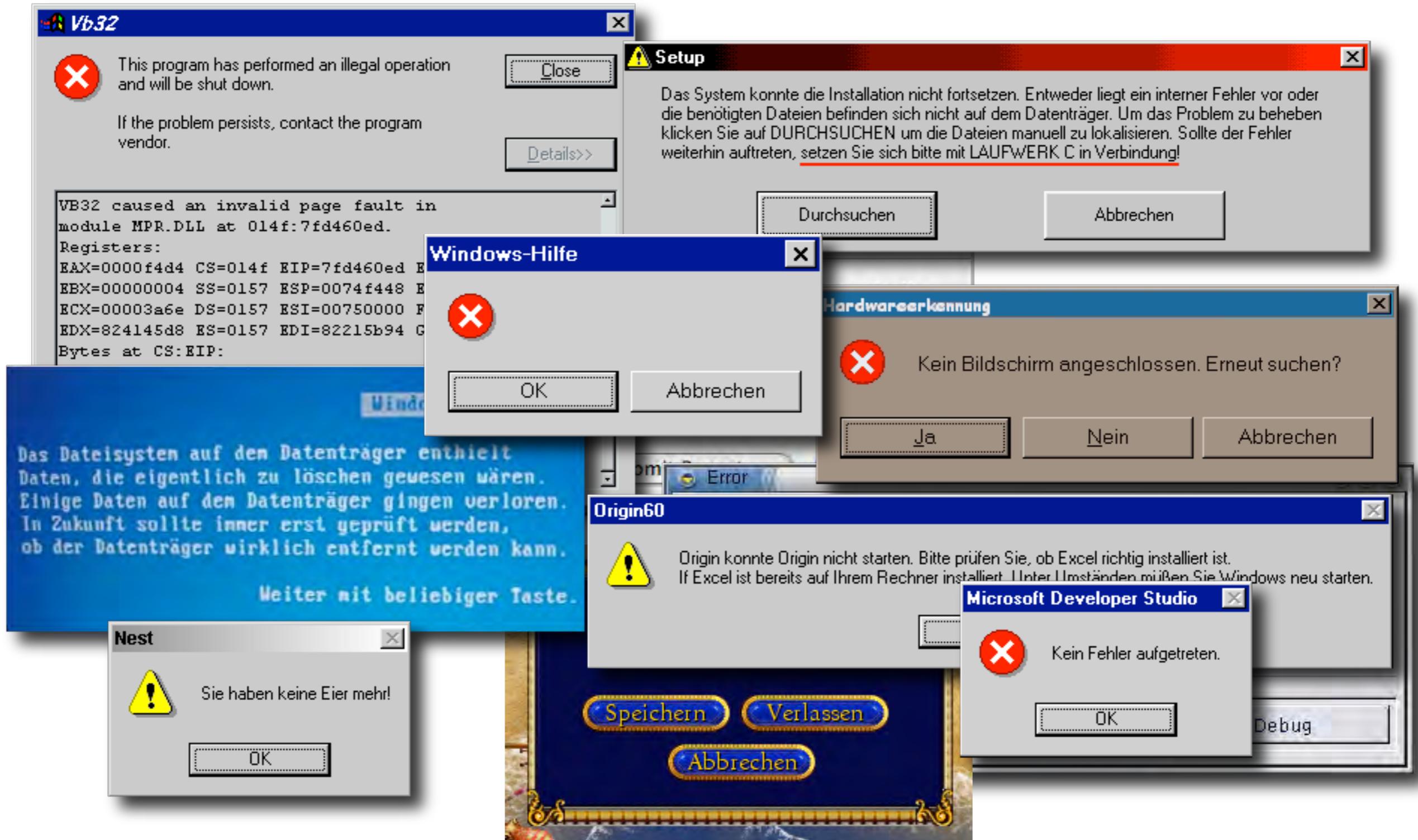
Abbrechen

**Microsoft Developer Studio**

Kein Fehler aufgetreten.

OK

# Das Problem





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

\$ █

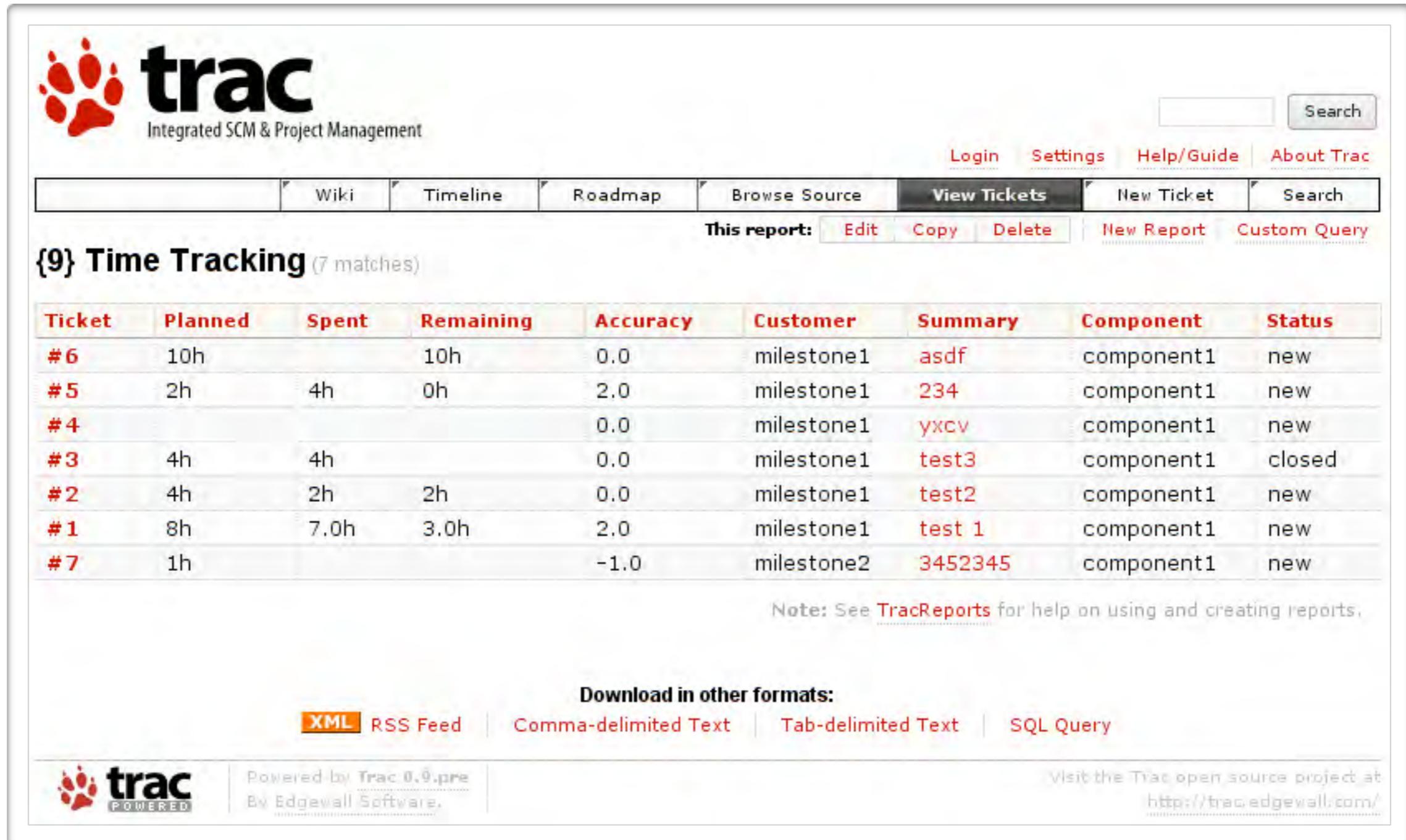
⌘



# Vorgehensweise

<b>T</b> rack the problem	<i>Problem verfolgen</i>
<b>R</b> eproduce	<i>Reproduzieren</i>
<b>A</b> utomate	<i>Automatisieren</i>
<b>F</b> ind Origins	<i>Ursprünge finden</i>
<b>F</b> ocus	<i>Fokussieren</i>
<b>I</b> solate	<i>Isolieren</i>
<b>C</b> orrect	<i>Korrigieren</i>

# Problem verfolgen



The screenshot displays the Trac web interface. At the top left is the Trac logo (a red paw print) and the text 'trac Integrated SCM & Project Management'. To the right is a search box and a 'Search' button. Below the logo are navigation links: 'Login', 'Settings', 'Help/Guide', and 'About Trac'. A secondary navigation bar contains 'Wiki', 'Timeline', 'Roadmap', 'Browse Source', 'View Tickets' (which is highlighted), 'New Ticket', and 'Search'. Below this bar, there are buttons for 'This report: Edit Copy Delete New Report Custom Query'. The main content area shows a report titled '{9} Time Tracking (7 matches)'. Below the title is a table with the following data:

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

Below the table, there is a note: 'Note: See [TracReports](#) for help on using and creating reports.'

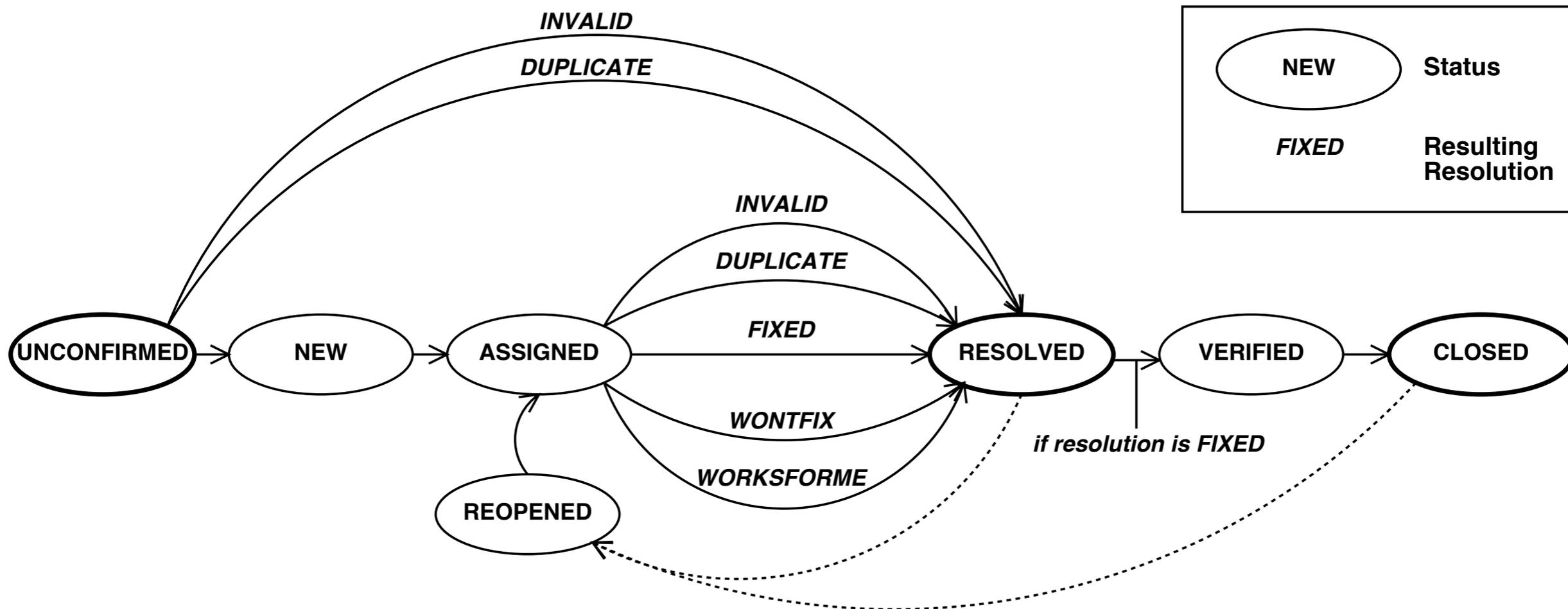
At the bottom, there is a section 'Download in other formats:' with links for 'XML RSS Feed', 'Comma-delimited Text', 'Tab-delimited Text', and 'SQL Query'.

The footer contains the Trac logo, the text 'Powered by Trac 0.9.pre By Edgewall Software.', and a link to the Trac open source project at <http://trac.edgewall.com/>.

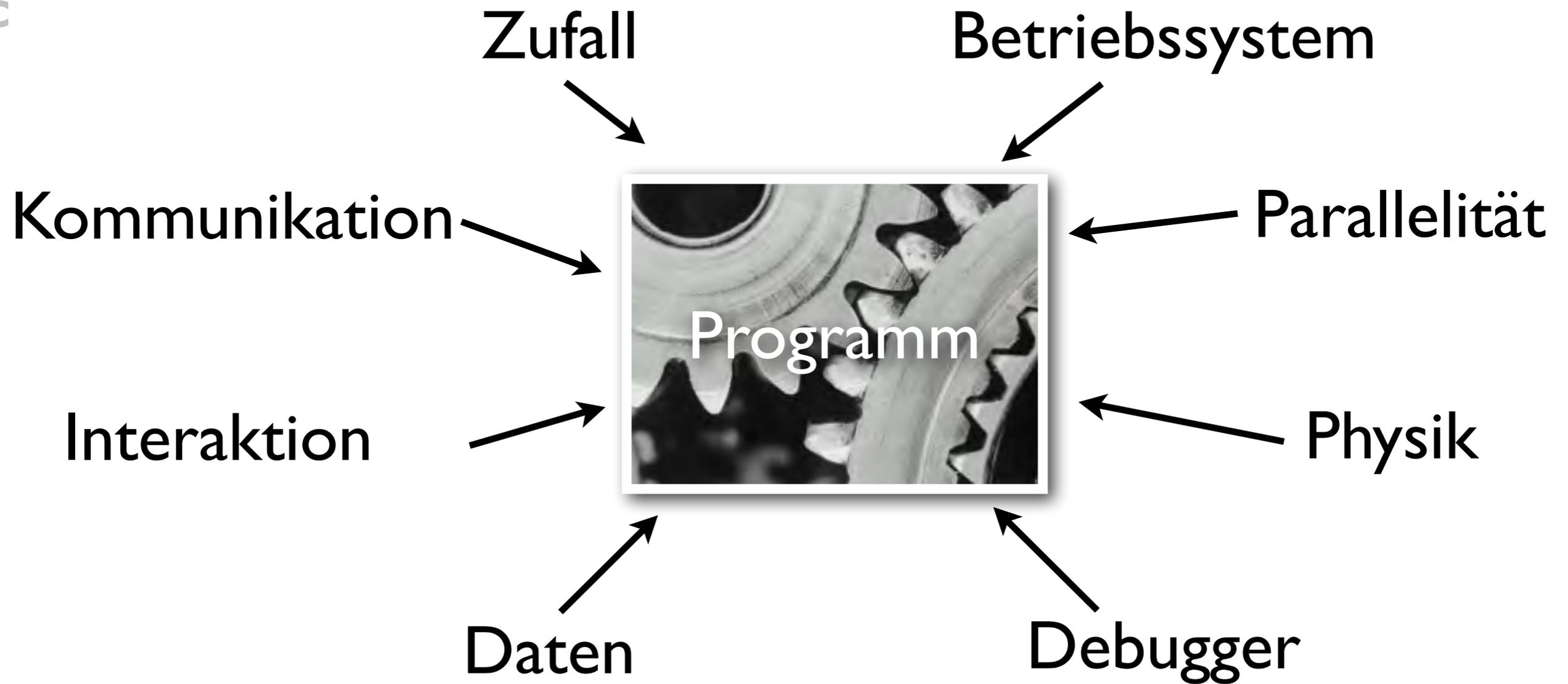
# Problem verfolgen

- Jedes Problem wird in die Fehler-Datenbank eingetragen
- Die Priorität bestimmt, welches Problem als nächstes bearbeitet wird
- Sind alle Probleme behoben, ist das Produkt fertig

# Lebenszyklus eines Problems



# Reproduzieren



# Automatisieren

```
// 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");
}
```

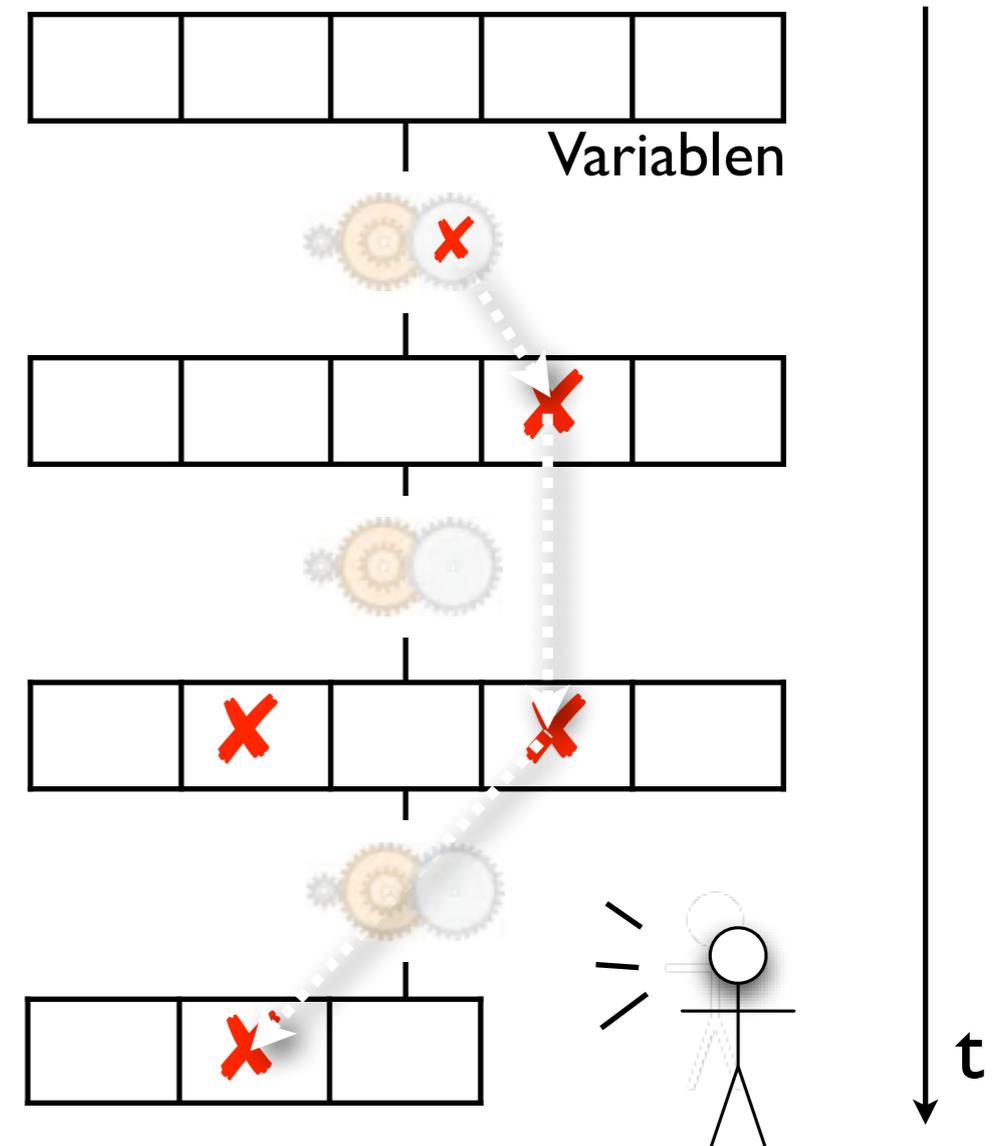
# Automatisieren

- Jedes Problem sollte automatisch reproduzierbar sein
- Dies geschieht über geeignete JUnit-Testfälle
- Nach jeder Änderung werden die Testfälle ausgeführt

# Ursprung finden

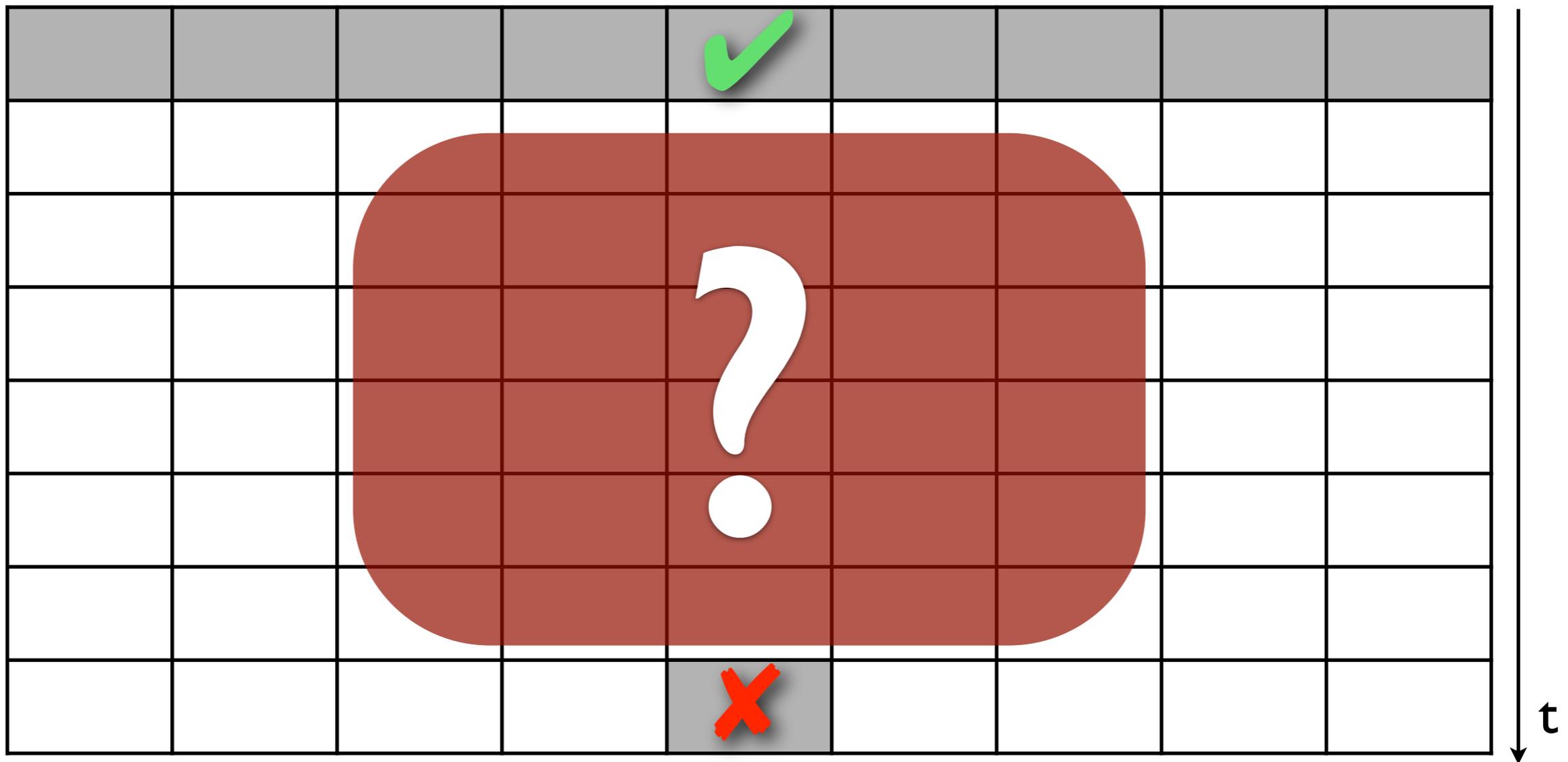
1. Der Programmierer erzeugt einen *Defekt* – einen Fehler im Code
2. Der ausgeführte Defekt erzeugt eine *Infektion* – einen Fehler im Zustand
3. Die Infektion breitet sich aus...
4. ...und wird als *Fehlverhalten* sichtbar.

Diese Infektionskette müssen wir *brechen*.



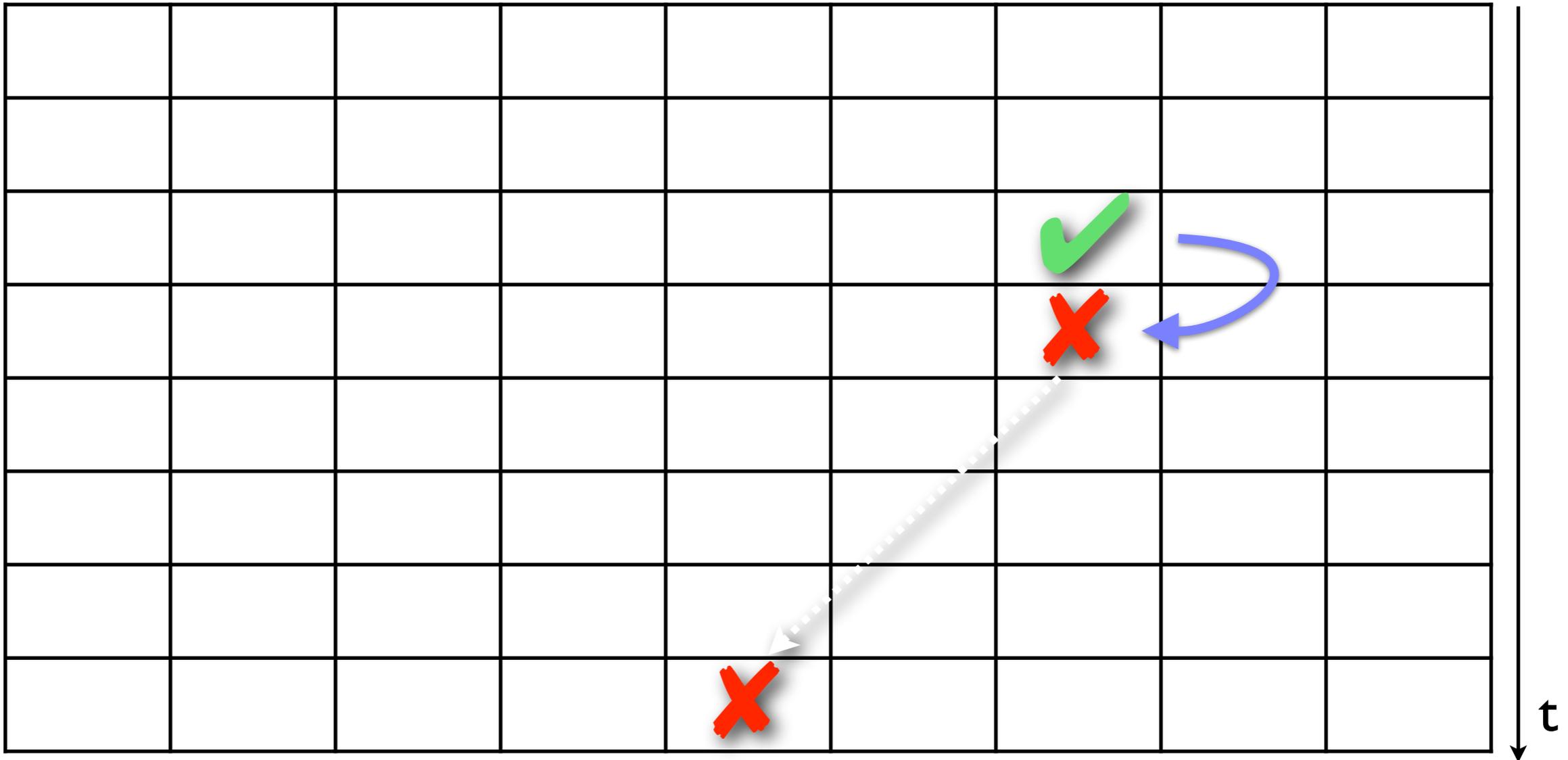
# Ursprung finden

Variablen

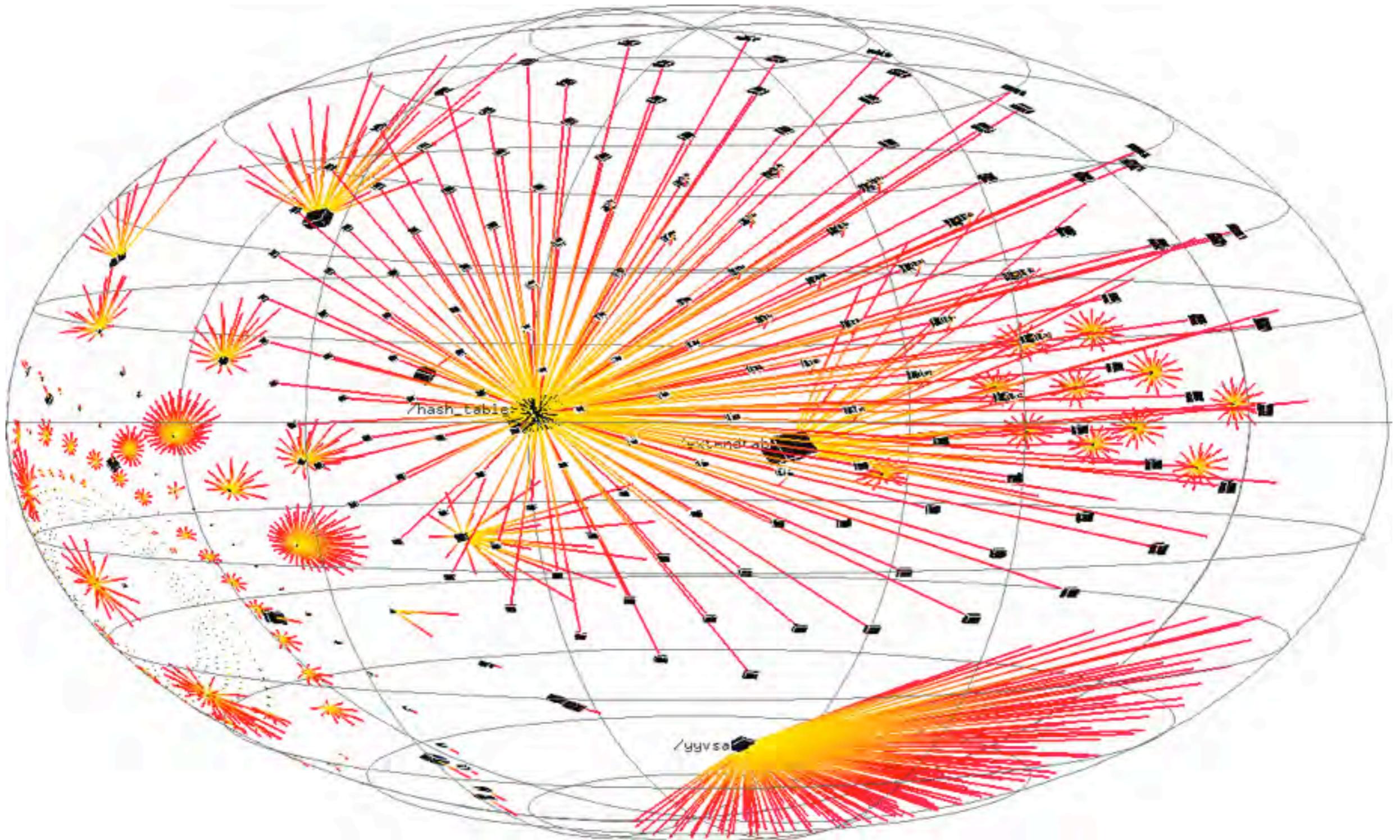


# Der Defekt

Variablen



# Ein Programmzustand

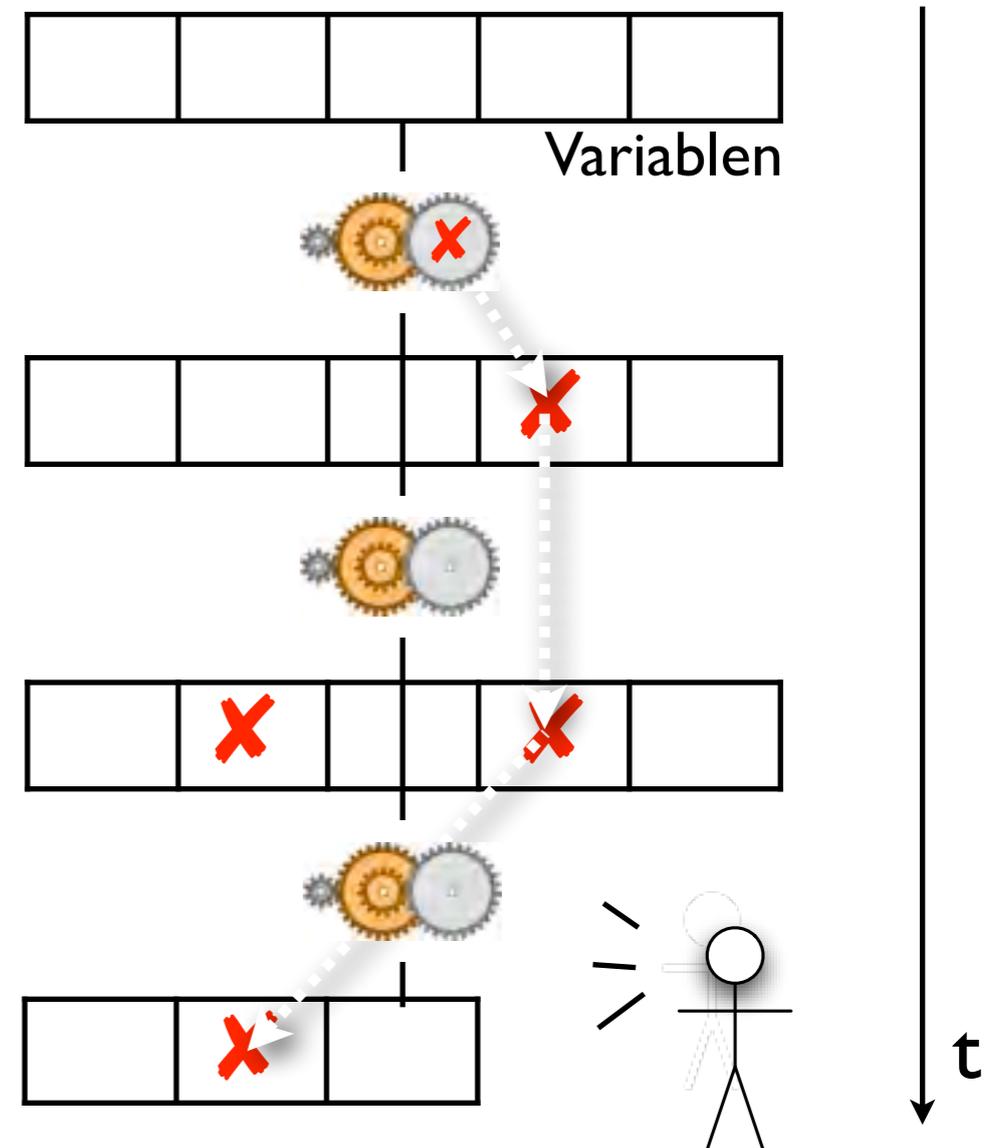


T  
R  
A  
**F**  
F  
I  
C



# Ursprung finden

1. Wir beginnen mit einer *bekanntem Infektion* (etwa am *Ende der Ausführung*)
2. Wir suchen die Infektion im *vorherigen Zustand*



DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C

File Edit View Program Commands Status Source Data Help

0: list->self

```

list->next          = new List(a_global + start++);
list->next->next     = new List(a_global + start++);
list->next->next->next = list;

```

**STOP** (void) list; // Display this

delete list; // none money  
delete list->next;  
delete list;

// Test  
void lis  
{  
list  
}

//  
void ref  
{  
date  
dele  
date

DDD Tip of the Day #5

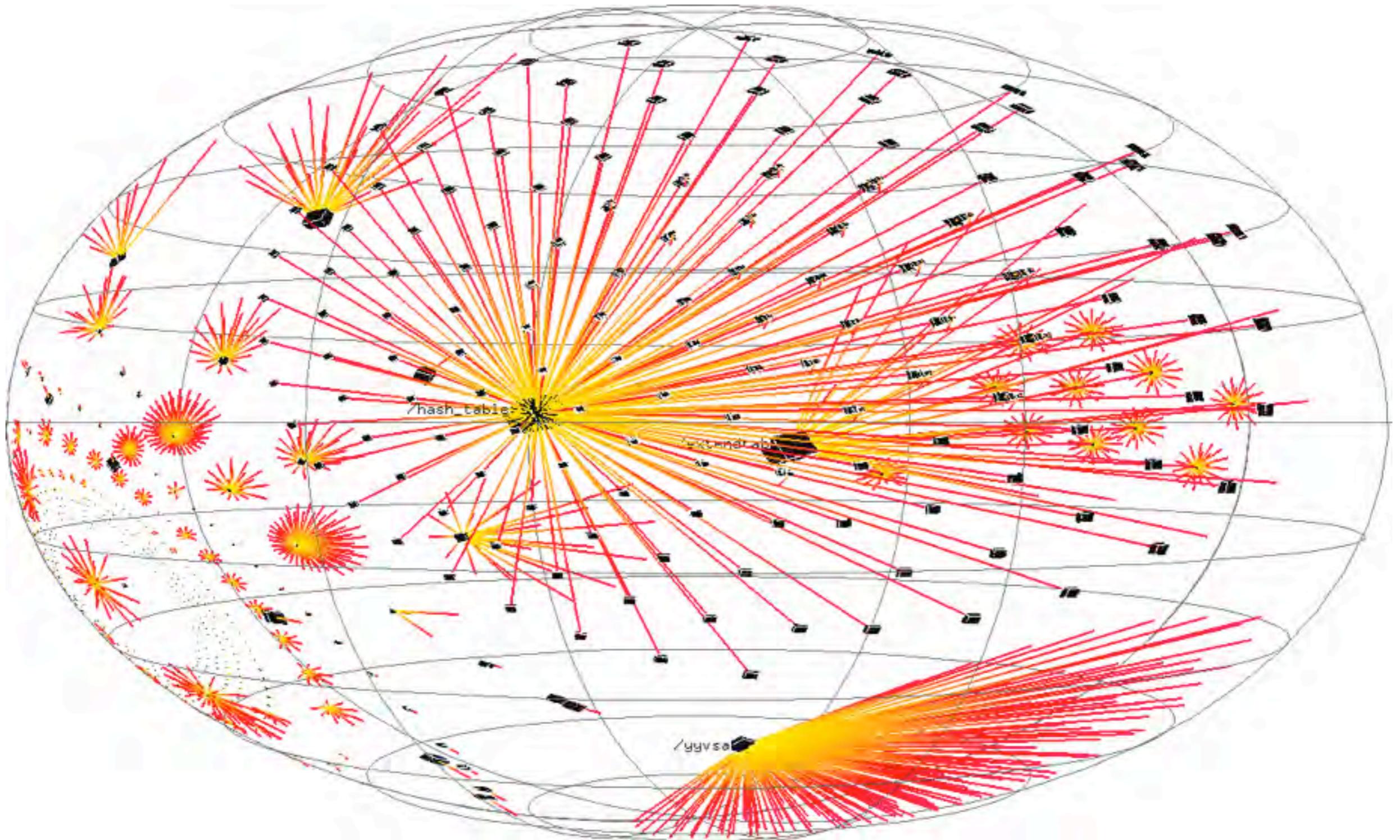
If you made a mistake, try **Edit→Undo**. This will undo the most recent debugger command and redisplay the previous program state.

Close Prev Tip Next Tip

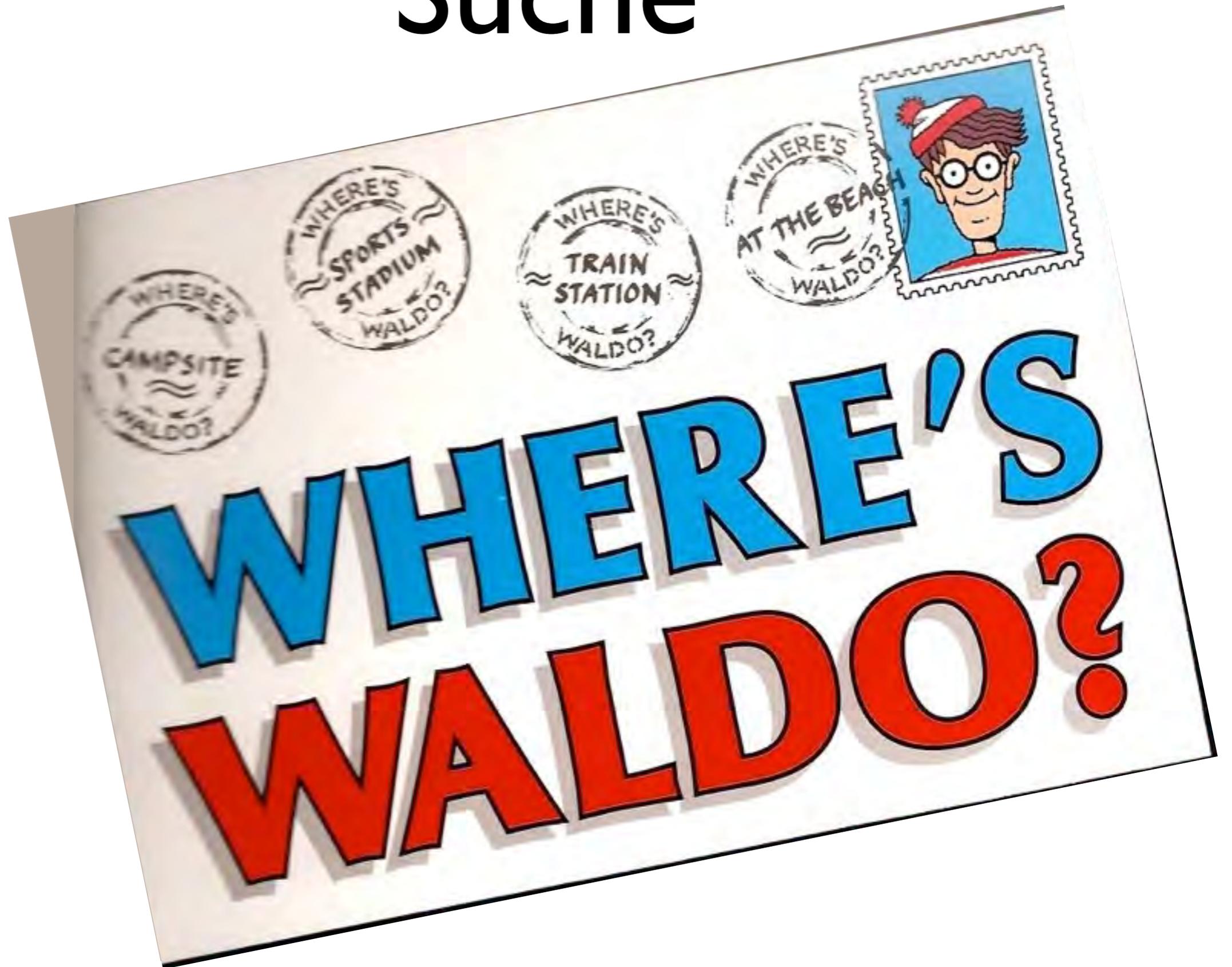
(gdb) graph display \*(list->next->next->self) dependent on 4  
(gdb) ↓

△ list = (List \*) 0x804df80

# Ein Programmzustand



# Suche





# Fokussieren

Bei der Suche nach Infektionen konzentrieren uns auf Stellen im Zustand, die

- *wahrscheinlich falsch* sind (z.B. weil hier früher Fehler aufgetreten sind)
- *explizit falsch* sind (z.B. weil sie eine *Zusicherung* verletzen)

Zusicherungen sind das effektivste Mittel, Infektionen zu finden.

# Infektionen finden

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

Jede Zeit von 00:00:00 bis 23:59:60 ist gültig

# Ursprung finden

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

void Time::set_hour(int h)
{
    assert (sane()); // Vorbedingung
    ...
    assert (sane()); // Nachbedingung
}
```

# Ursprung finden

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

sane() ist die *Invariante* eines Time-Objekts:

- gilt *vor* jeder öffentlichen Methode
- gilt *nach* jeder öffentlichen Methode

# Ursprung finden

- Vorbedingung schlägt fehl = Infektion *vor* Methode
- Nachbedingung schlägt fehl = Infektion *nach* Methode
- Alle Zusicherungen ok = keine Infektion

```
void Time::set_hour(int h)
{
    assert (sane()); // Vorbedingung
    ...
    assert (sane()); // Nachbedingung
}
```

# Komplexe Invarianten

```
class RedBlackTree {  
    ...  
    boolean sane() {  
        assert (rootHasNoParent());  
        assert (rootIsBlack());  
        assert (redNodesHaveOnlyBlackChildren());  
        assert (equalNumberOfBlackNodesOnSubtrees());  
        assert (treeIsAcyclic());  
        assert (parentsAreConsistent());  
  
        return true;  
    }  
}
```

# Zusicherungen

				✓				
✓	✓	✓						
✓	✓	✓						
✓	✓	✓						
✓	✓	✓						
✓	✓	✓						
✓	✓	✓						
✓	✓	✓		✗				

↓ t

# Fokussieren

- Alle möglichen Einflüsse müssen geprüft werden
- Konzentration auf wahrscheinlichste Kandidaten
- Zusicherungen helfen schnell, Infektionen zu finden

# Isolieren

- Fehlerursachen sollen *systematisch* eingeeengt werden – mit Beobachtungen und Experimenten.

# Wissenschaftliche Methode

1. Beobachte einen Teil des Universums
2. Erfinde eine *Hypothese*, die mit der Beobachtung übereinstimmt
3. Nutze die Hypothese, um Vorhersagen zu machen.
4. Teste die Vorhersagen durch Experimente oder Beobachtungen und passe die Hypothese an.
5. Wiederhole 3 and 4, bis die Hypothese zur *Theorie* wird.

# Wissenschaftliche Methode

Fehlerbericht

Code

Hypothese ist *bestätigt*:  
Hypothese verfeinern

Hypothese

Vorhersage

Experiment

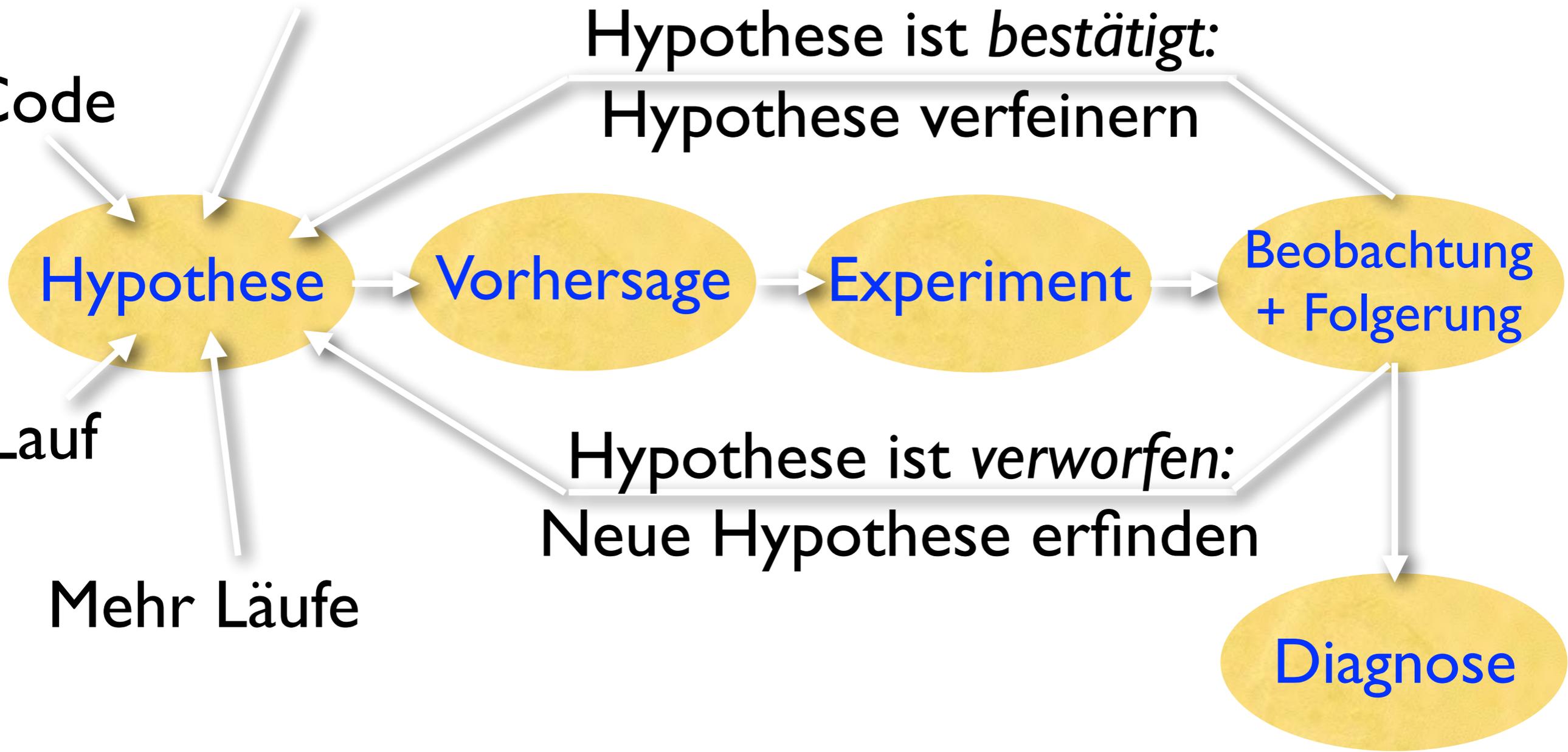
Beobachtung  
+ Folgerung

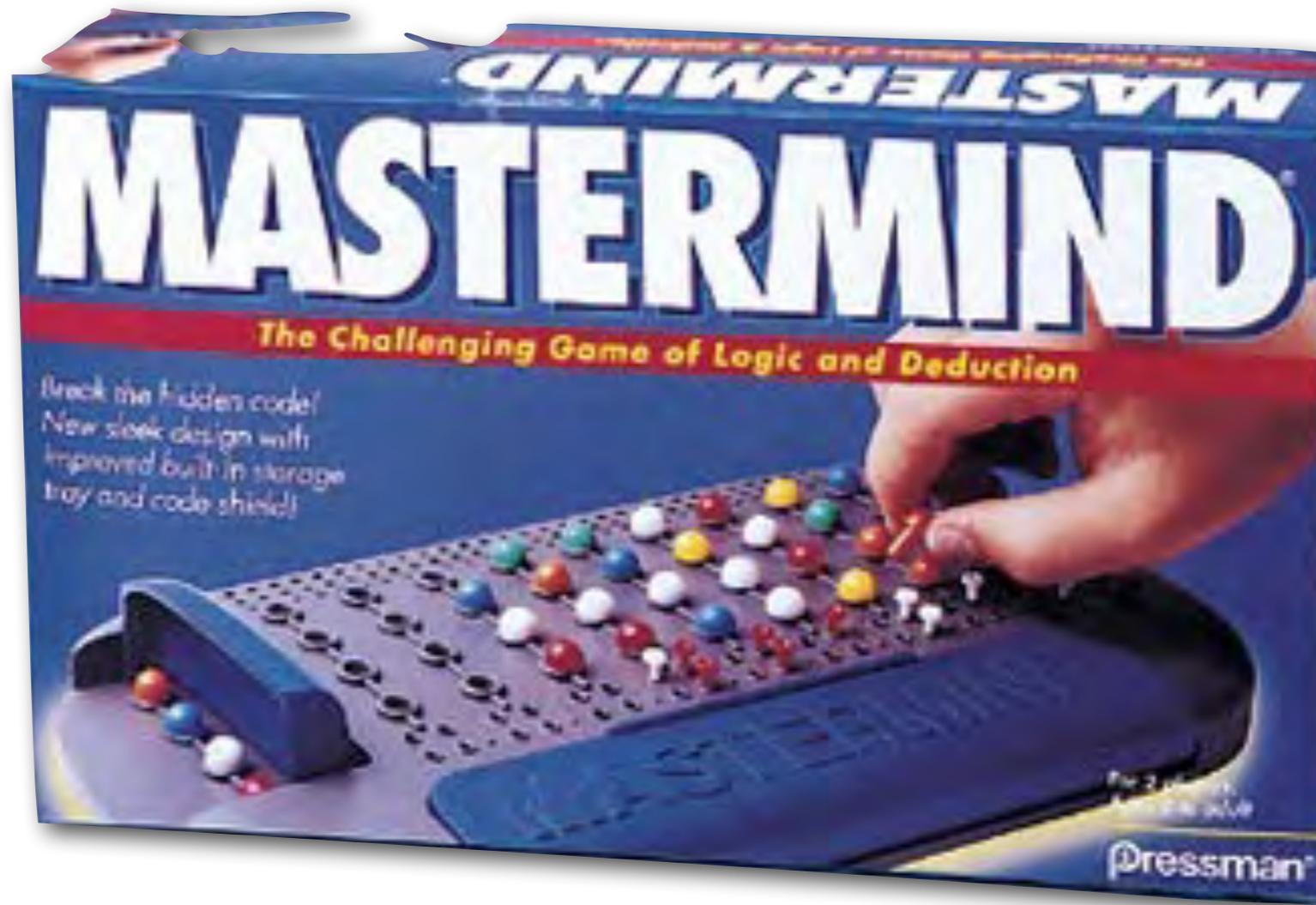
Lauf

Hypothese ist *verworfen*:  
Neue Hypothese erfinden

Mehr Läufe

Diagnose





# Explizite Hypothesen

Hypothesis	The execution causes $a[0] = 0$
Prediction	At line 37, the following should hold.
Experiment	Run the program on line 37.
Observation	The program behaves as predicted.
Conclusion	The hypothesis is confirmed.

Wer alles im Kopf behält,  
spielt Mastermind blind!

# Explizite Hypothesen



# Isolieren

- Wir wiederholen die Suche nach Infektions-Ursprüngen, bis wir den Defekt gefunden haben.
- Wir gehen *systematisch* vor – im Sinne der wissenschaftlichen Methode
- Durch *explizite* Schritte leiten wir die Suche und können sie jederzeit nachvollziehen

# Korrektur

Vor der Korrektur müssen wir prüfen, ob der Defekt

- tatsächlich ein *Fehler* ist und
- das Fehlverhalten *verursacht*

Erst wenn beides verstanden ist, dürfen wir den Fehler korrigieren.

# The Devil's Guide to Debugging

Finde den Defekt durch Raten:

- Verstreue überall Debugging-Anweisungen
- Ändere den Code, bis etwas funktioniert
- Mache keine Kopien von alten Versionen
- Versuche gar nicht erst zu verstehen, was das Programm tun soll.

# The Devil's Guide to Debugging

Verschwende keine Zeit damit, dem Problem auf den Grund zu gehen

- Die meisten Probleme sind ohnehin trivial

# The Devil's Guide to Debugging

Benutze die offensichtlichste Reparatur:

- Repariere nur das, was Du siehst:

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

Warum sich mit `compute()` beschäftigen?

# Erfolgreiche Korrektur



# Hausaufgaben

- Tritt das Fehlverhalten nicht mehr auf?  
(Falls doch, sollte dies eine große Überraschung sein)
- Könnte die Korrektur neue Fehler einführen?
- Wurde derselbe Fehler woanders gemacht?
- Ist meine Korrektur ins Versionsmanagement und Problem-Tracking eingespielt?

# Vorgehensweise

<b>T</b> rack the problem	<i>Problem verfolgen</i>
<b>R</b> eproduce	<i>Reproduzieren</i>
<b>A</b> utomate	<i>Automatisieren</i>
<b>F</b> ind Origins	<i>Ursprünge finden</i>
<b>F</b> ocus	<i>Fokussieren</i>
<b>I</b> solate	<i>Isolieren</i>
<b>C</b> orrect	<i>Korrigieren</i>

# Vorgehensweise

<b>T</b> rack the problem	<i>Problem verfolgen</i>
<b>R</b> eproduce	<i>Reproduzieren</i>
<b>A</b> utomate	<i>Automatisieren</i>
<b>F</b> ind Origins	<i>Ursprünge finden</i>
<b>F</b> ocus	<i>Fokussieren</i>
<b>I</b> solate	<i>Isolieren</i>
<b>C</b> orrect	<i>Korrigieren</i>