Technische Fakultät
Universität Bielefeld

Vorlesung

Softwaretest und -debugging

Dr. Carsten Gnörlich

Rechnerbetriebsgruppe

Kap. 8 - Wie Fehler zustande kommen und man sie managt

(= Kap. 1 und 2 aus Zeller)

Abgrenzung zum Softwaretest

Unterschiede zwischen Testen und Debuggen

Aufdecken von Fehlfunktionen

während der Entwicklung

beim Produktionssystem

Unterschiede zwischen Testen und Debuggen

Aufdecken von Fehlfunktionen

während der Entwicklung

beim Produktionssystem

Softwaretestmethoden
→ unbekannte Fehlfkt finden

Unterschiede zwischen Testen und Debuggen

Aufdecken von Fehlfunktionen

während der Entwicklung

beim Produktionssystem

Softwaretestmethoden
→ unbekannte Fehlfkt finden

Debuggen
→ bekannte Fehlfkt beheben
= zufällige Fehlfkt b. Kunden

Unterschiede zwischen Testen und Debuggen

Aufdecken von Fehlfunktionen

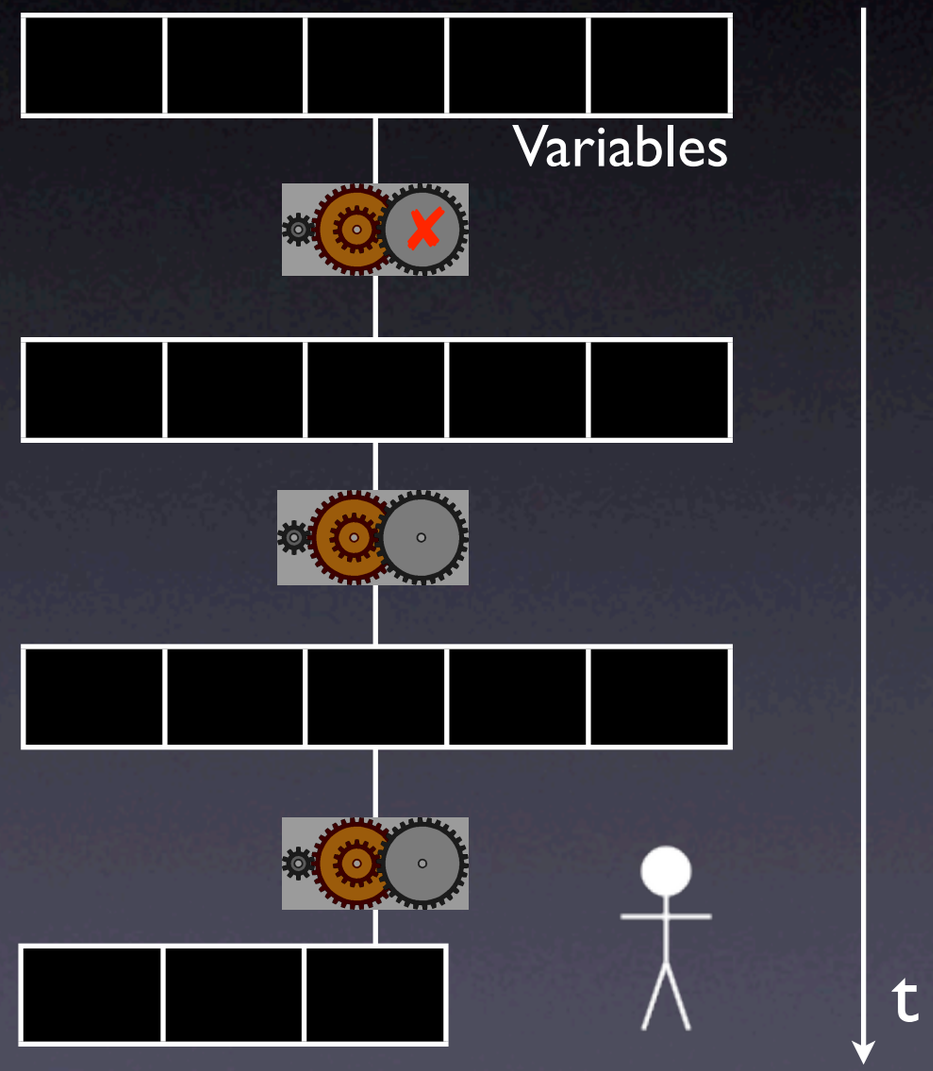
während der Entwicklung	beim Produktionssystem
Softwaretestmethoden	**Debuggen**
→ unbekannte Fehlfkt finden	→ bekannte Fehlfkt beheben
Finden der Fehlfkt. **teuer**	Finden der Fehlfkt. **gratis**
Fehlerlokalisation **billig**	Fehlerlokalisation **teuer**
→ „Nebenprodukt“;	→Hauptaufwand!
ergibt sich aus den Testfällen	Reproduktion, Lokalisierung

How Failures Come to be

Andreas Zeller

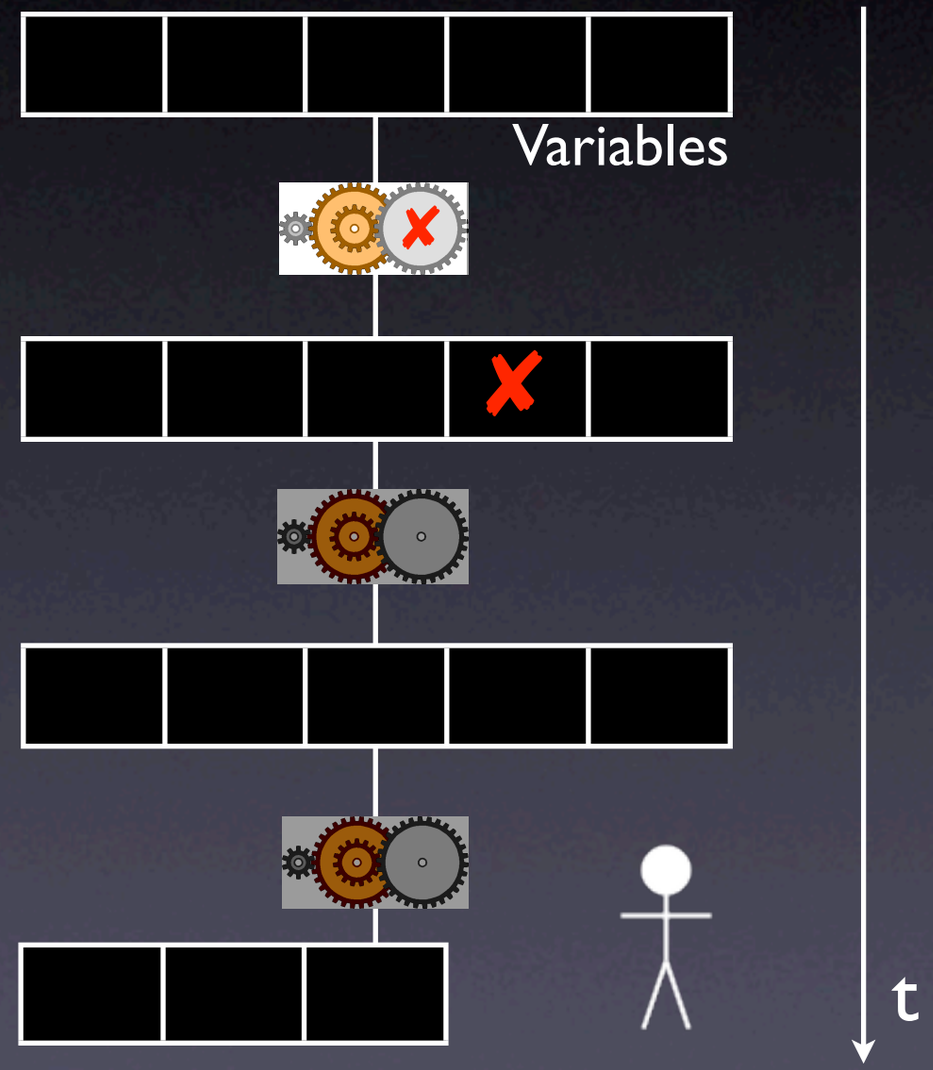
From Defect to Failure

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



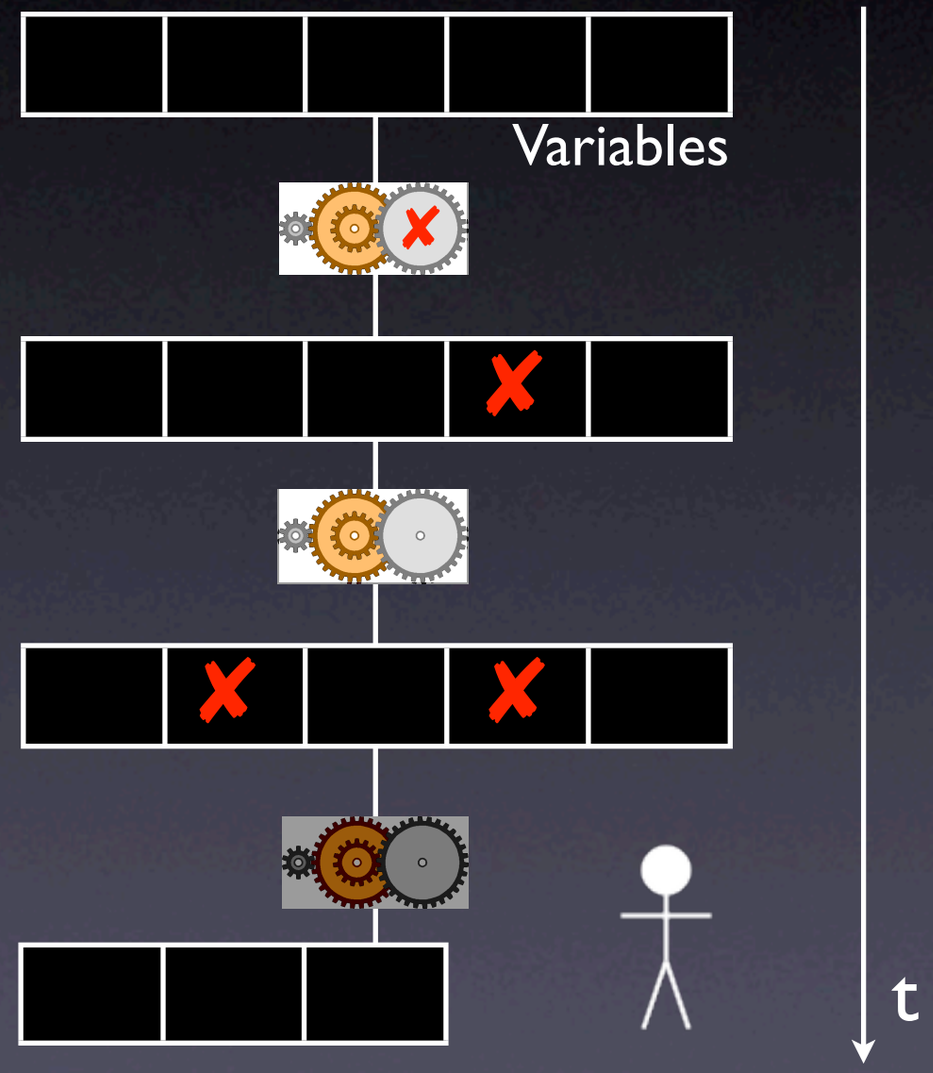
From Defect to Failure

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



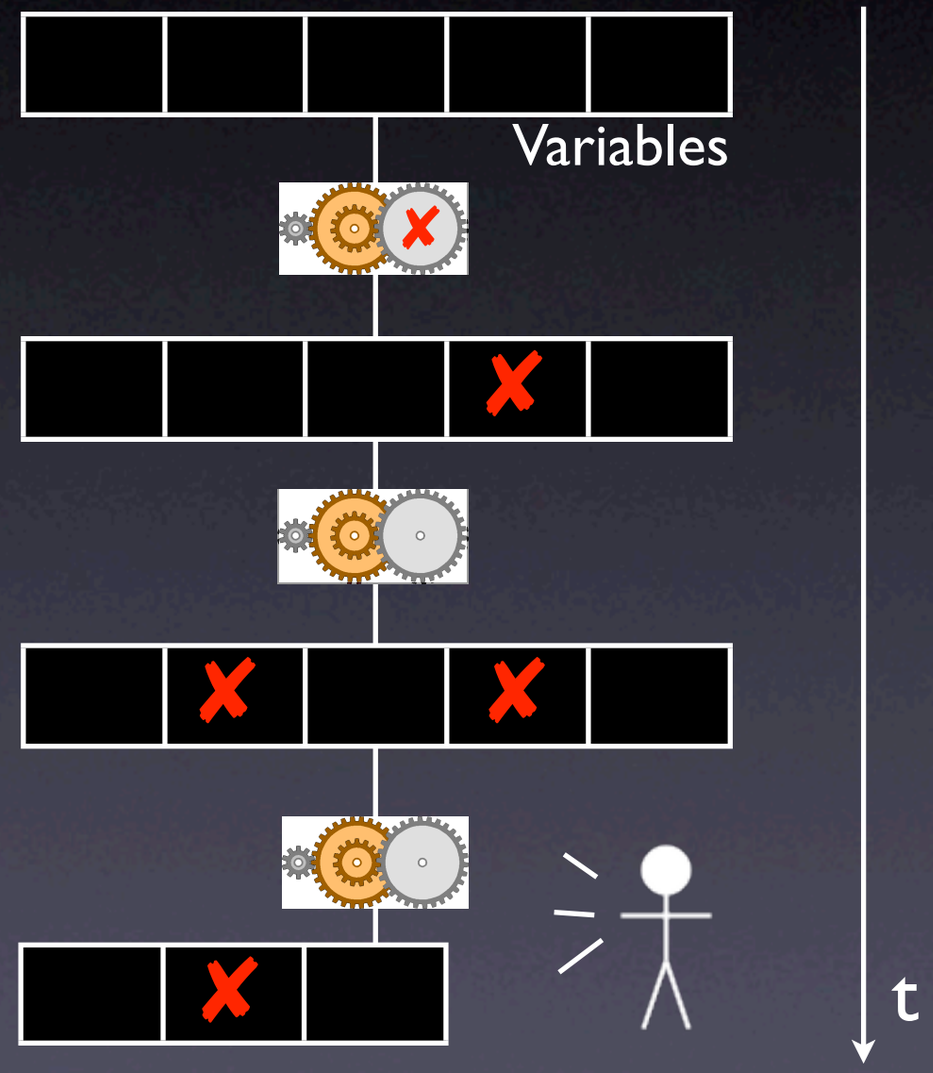
From Defect to Failure

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



From Defect to Failure

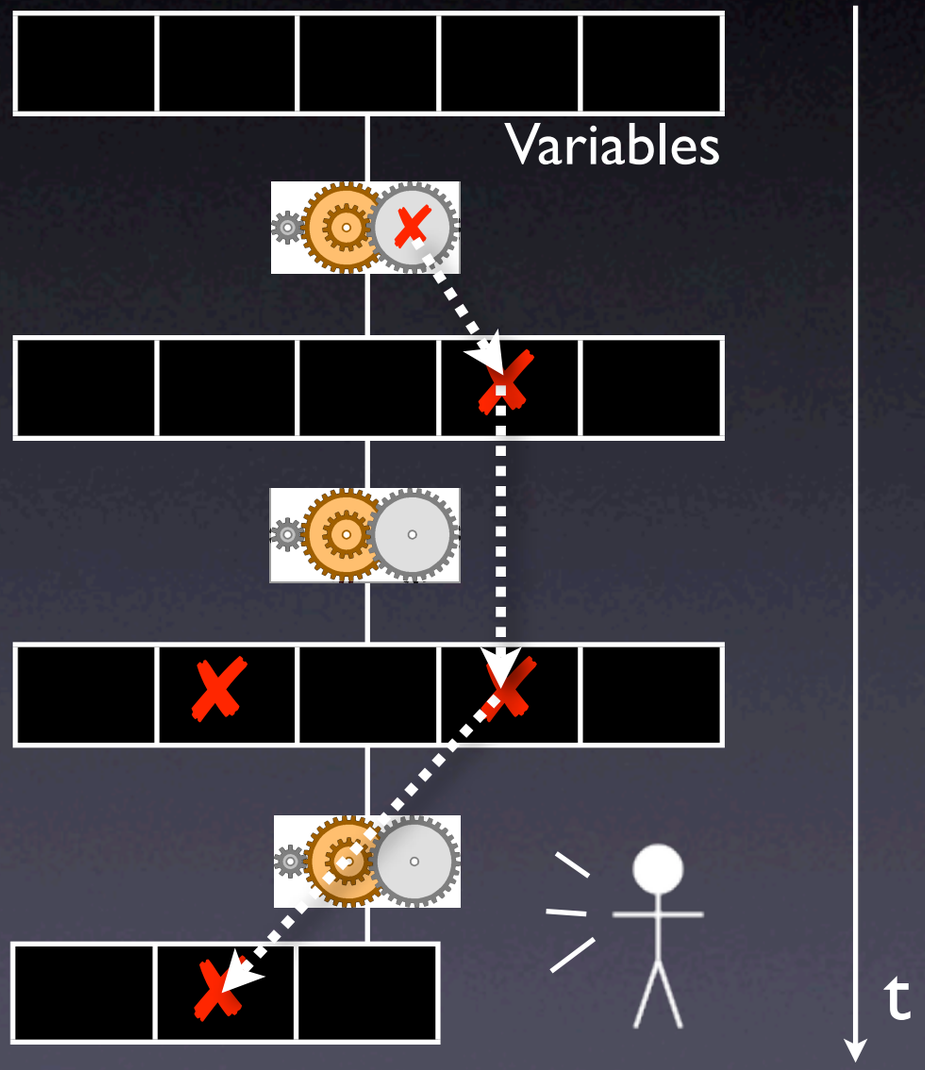
1. The programmer creates a *defect* – an error in the code.
2. When executed, the defect creates an *infection* – an error in the state.
3. The infection *propagates*.
4. The infection causes a *failure*.



From Defect to Failure

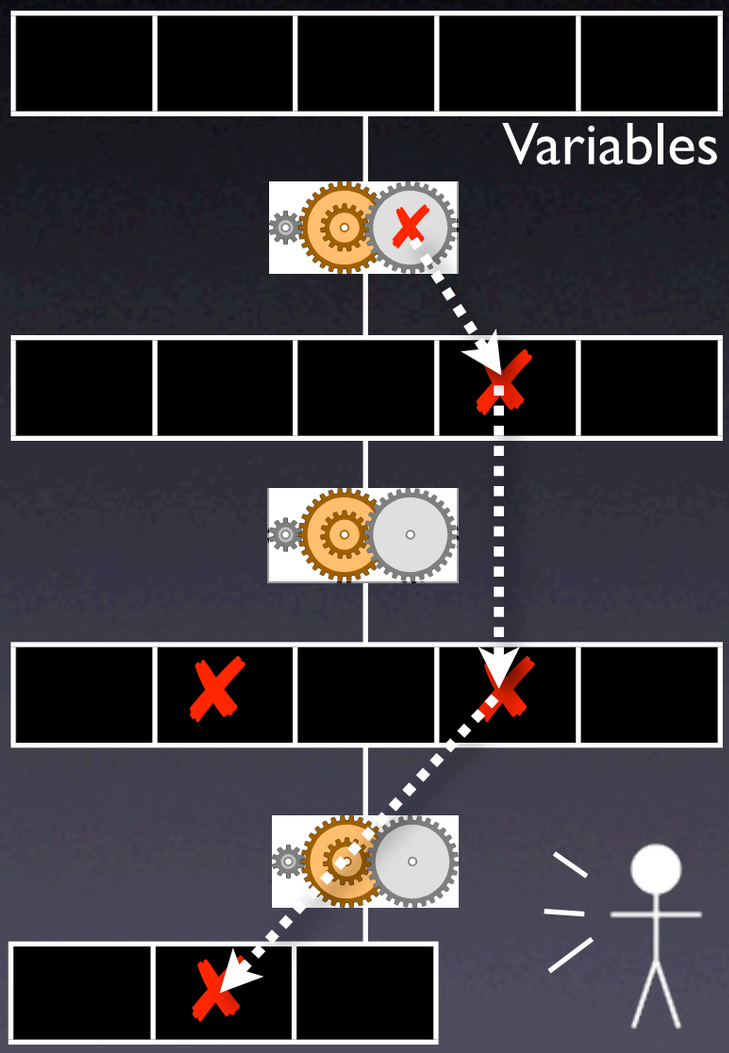
1. The programmer creates a *defect* – an error in the code.
2. When executed, the defect creates an *infection* – an error in the state.
3. The infection *propagates*.
4. The infection causes a *failure*.

This infection chain must be traced back – and broken.



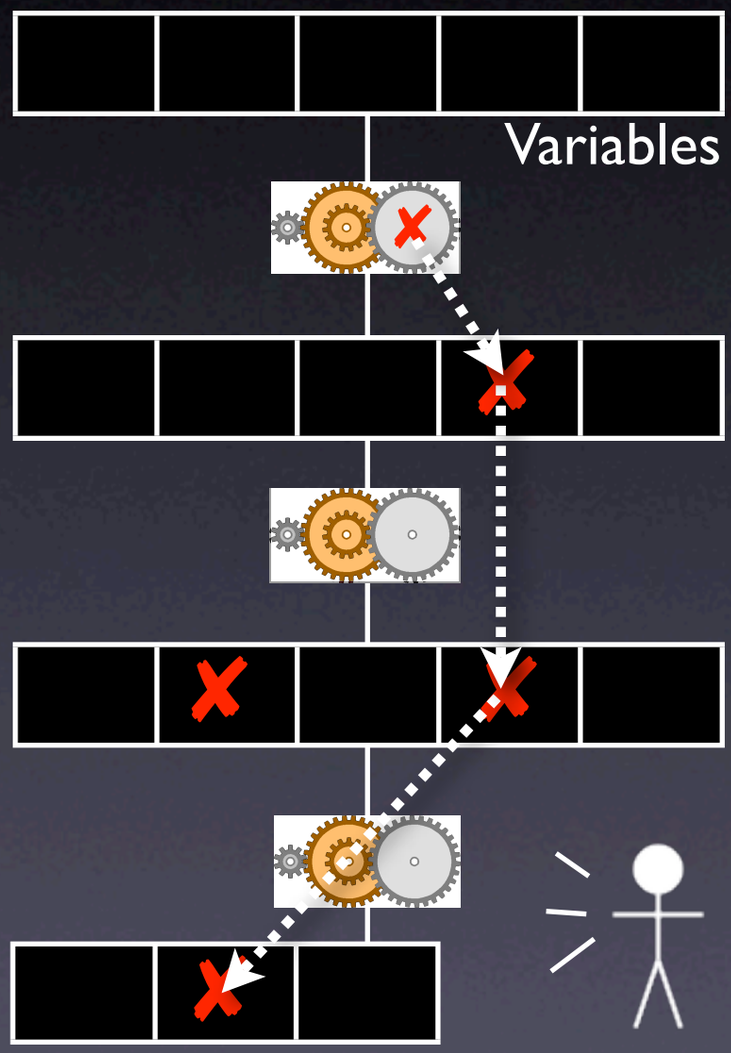
Debugging

- Every *failure* can be traced back to some *infection*, and every *infection* is caused by some *defect*.
- *Debugging* means to *relate a given failure to the defect* – and to remove the defect.



The Curse of Testing

- Not every defect causes a failure!
- *Testing can only show the presence of errors – not their absence.*
(Dijkstra 1972)



The Traffic Principle

The Traffic Principle

T

R

A

F

F

I

C

The Traffic Principle

T rack the problem

R

A

F

F

I

C

The Traffic Principle

T rack the problem

R eproduce the failure

A

F

F

I

C

The Traffic Principle

T rack the problem

R eproduce the failure

A utomate and simplify

F

F

I

C

The Traffic Principle

T rack the problem

R eproduce the failure

A utomate and simplify

F ind possible infection origins

F

I

C

The Traffic Principle

T rack the problem

R eproduce the failure

A utomate and simplify

F ind possible infection origins

F ocus on likely origins

I

C

The Traffic Principle

T rack the problem

R eproduce the failure

A utomate and simplify

F ind possible infection origins

F ocus on likely origins

I solate the infection chain

C

The Traffic Principle

T rack the problem

R eproduce the failure

A utomate and simplify

F ind possible infection origins

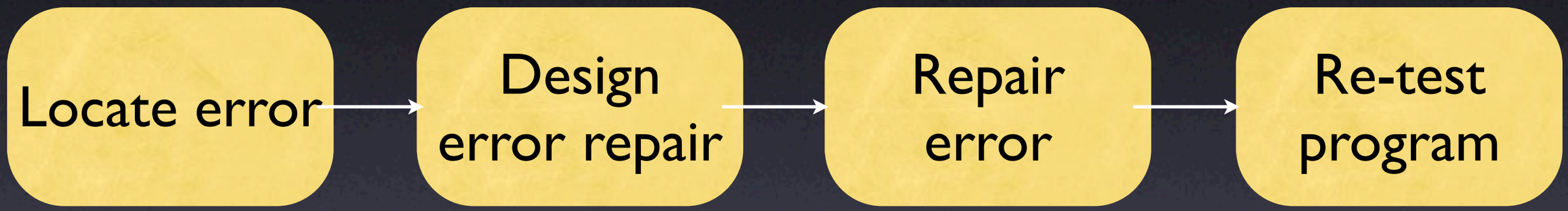
F ocus on likely origins

I solate the infection chain

C orrect the defect

How to Debug

(Sommerville 2004)



The Traffic Principle

T rack the problem

R eproduce the failure

A utomate and simplify

Komplexes Suchproblem

F ind possible infection origins

F ocus on likely origins

I solate the likely infection chain

C orrect the defect

A Sample Program

A Sample Program

$

A Sample Program

```
$ sample 9 8 7
```

A Sample Program

```
$ sample 9 8 7
Output: 7 8 9
```

A Sample Program

```
$ sample 9 8 7
Output: 7 8 9
```

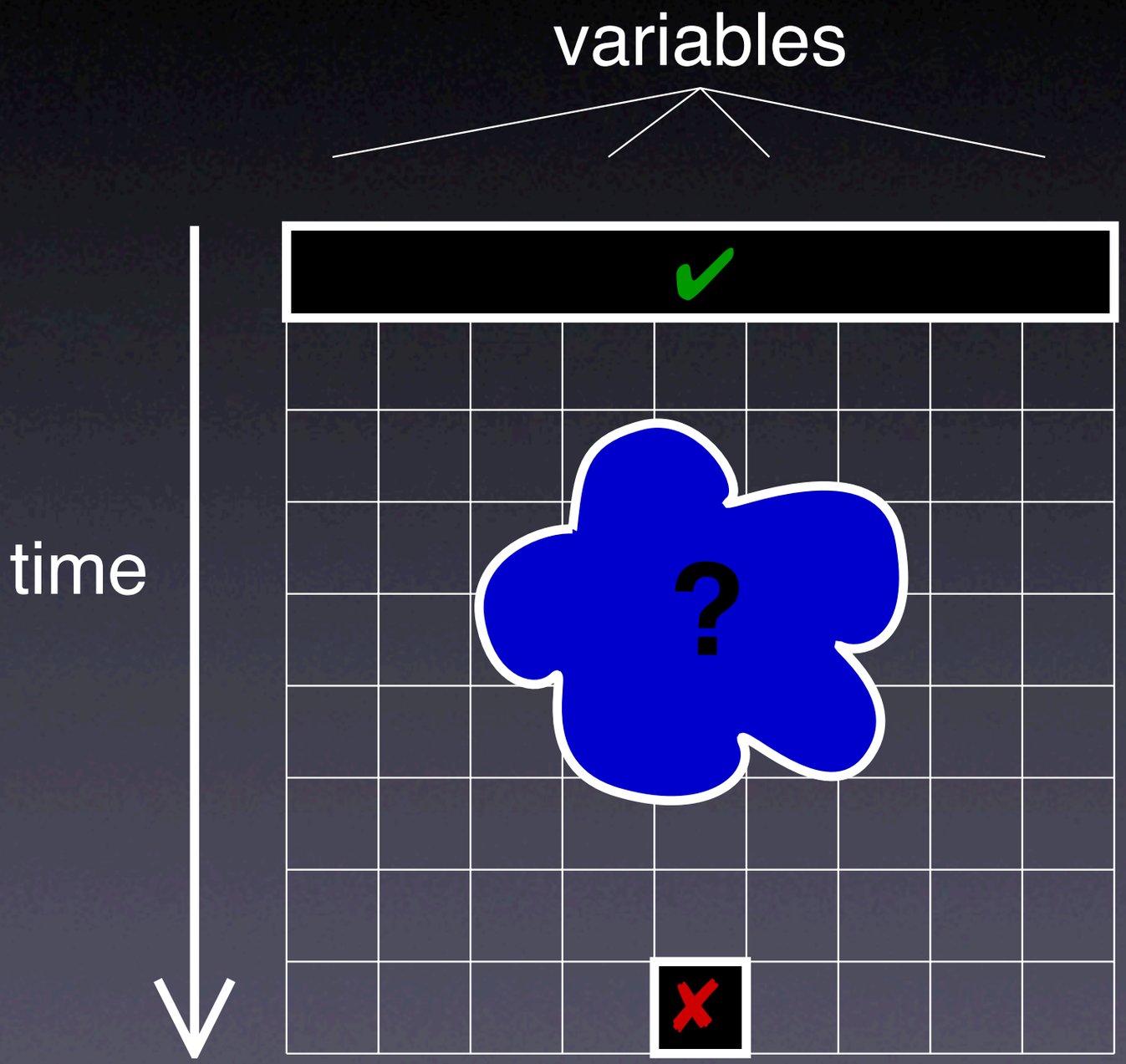
```
$ sample 11 14
```

A Sample Program

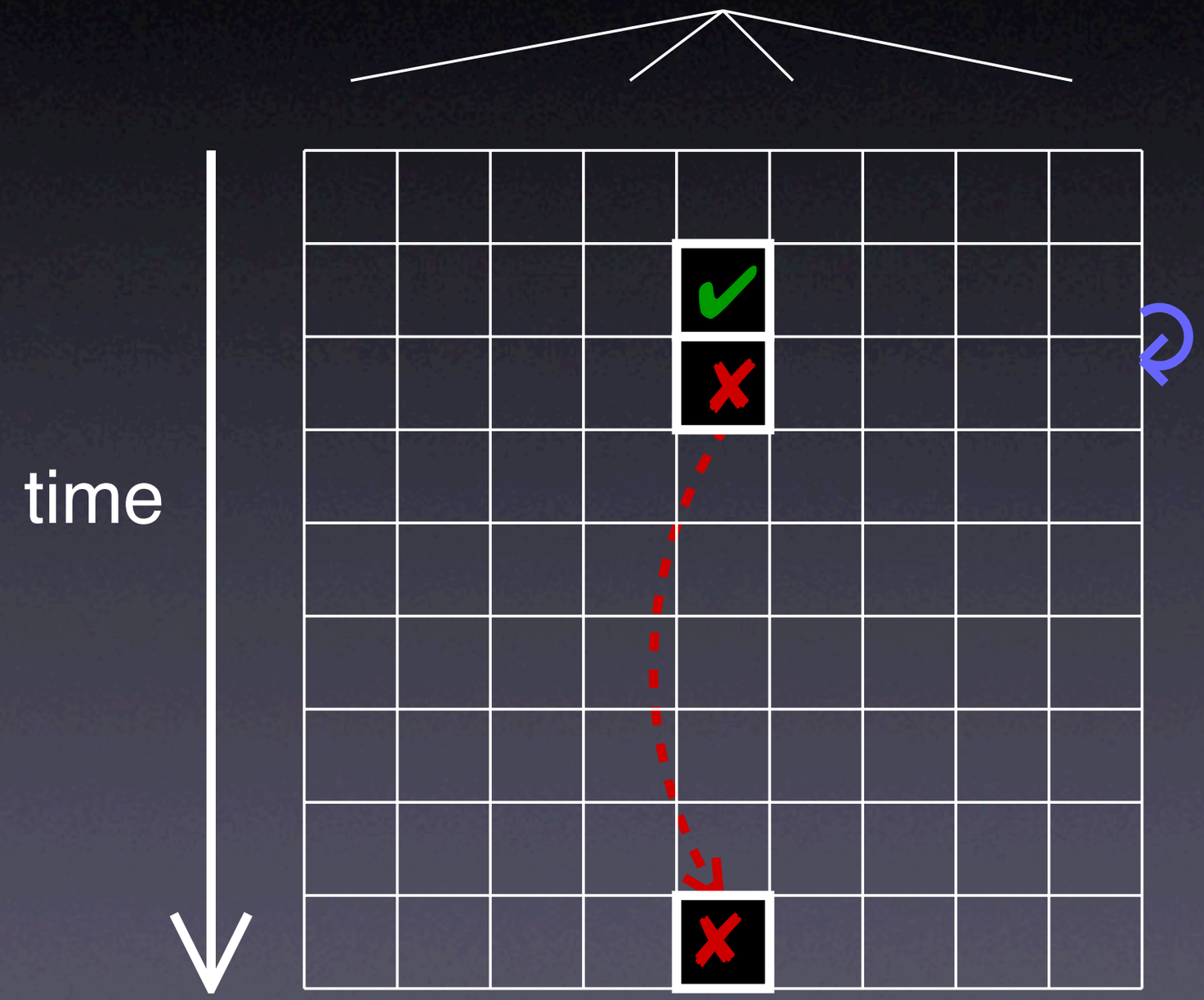
```
$ sample 9 8 7
Output: 7 8 9
```

```
$ sample 11 14
Output: 0 11
```

Search in Space + Time



The Defect



Ansatz: Divide and Conquer

a) gesunde von infizierten Zuständen trennen

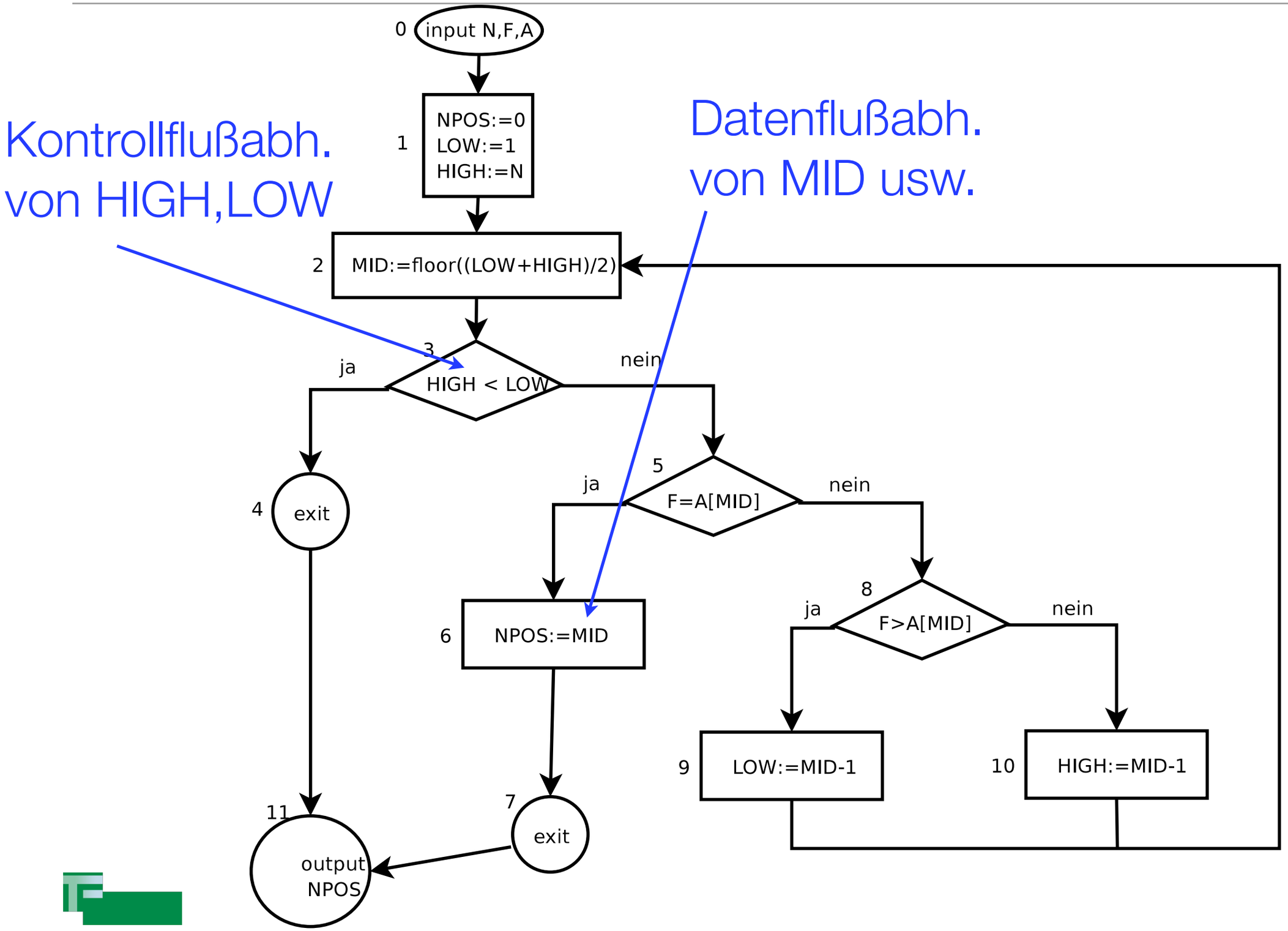
b) relevante Variablen von irrelevanten trennen

- jede Variable hängt von endlich vielen Vorgängern ab

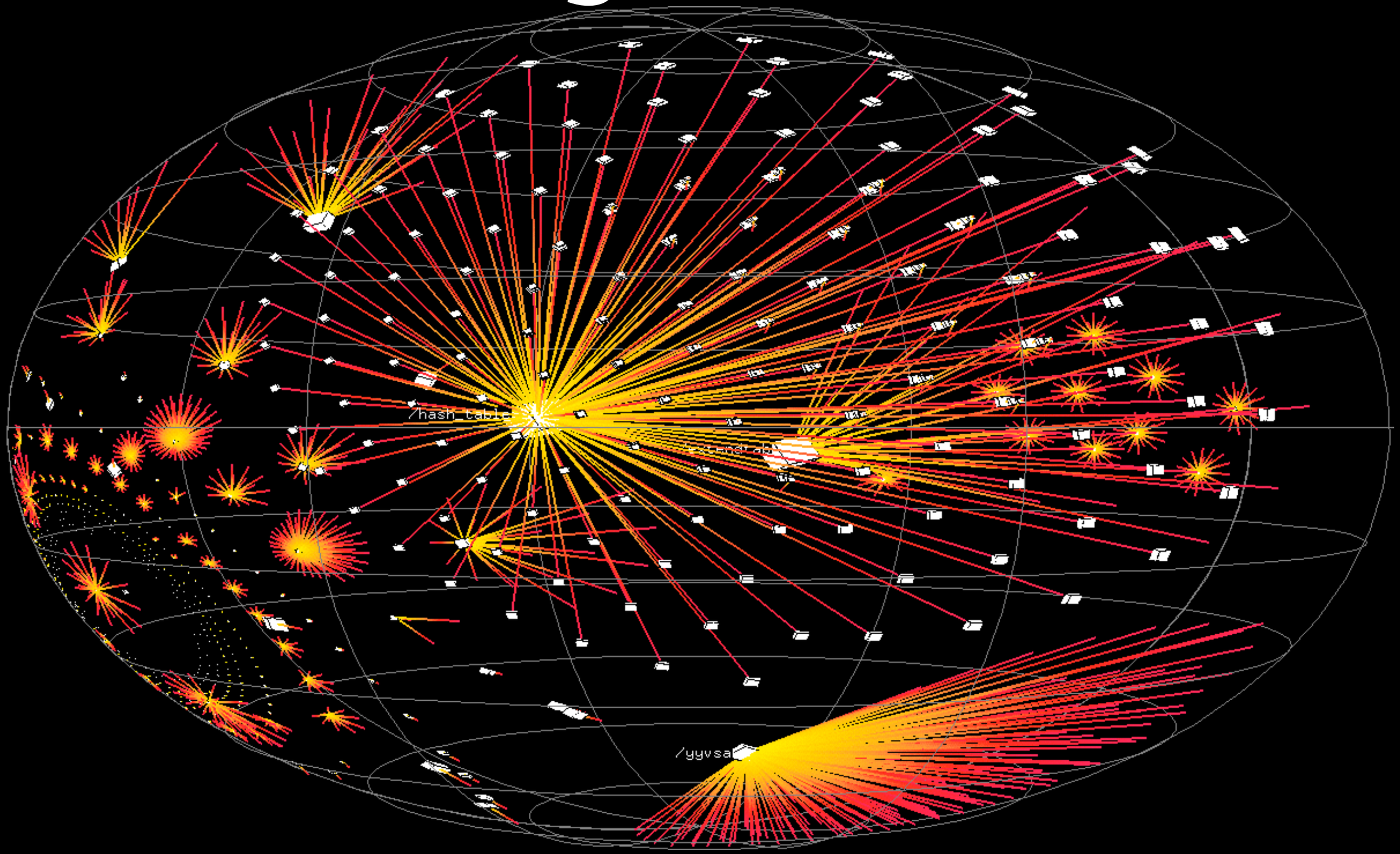
→Kontrollfluß, Datenfluß ausnutzen

c) analog für Funktionen, Module machen

Beispiel aus Kap.5



A Program State



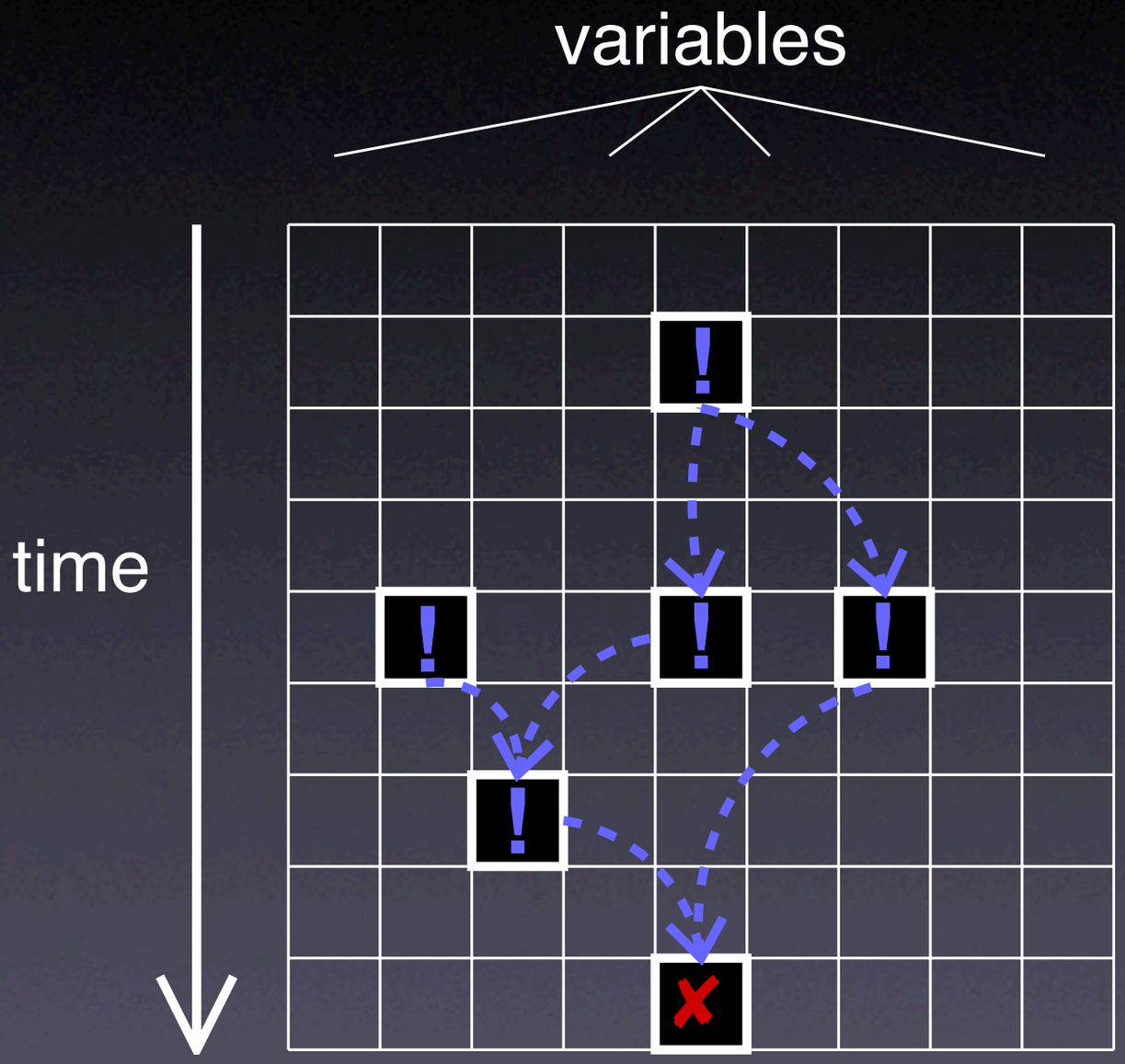
A Sample Program

```
$ sample 9 8 7
Output: 7 8 9
```

```
$ sample 11 14
Output: 0 11
```

```c
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    printf("Output: ");
    for (i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);

    return 0;
}
```

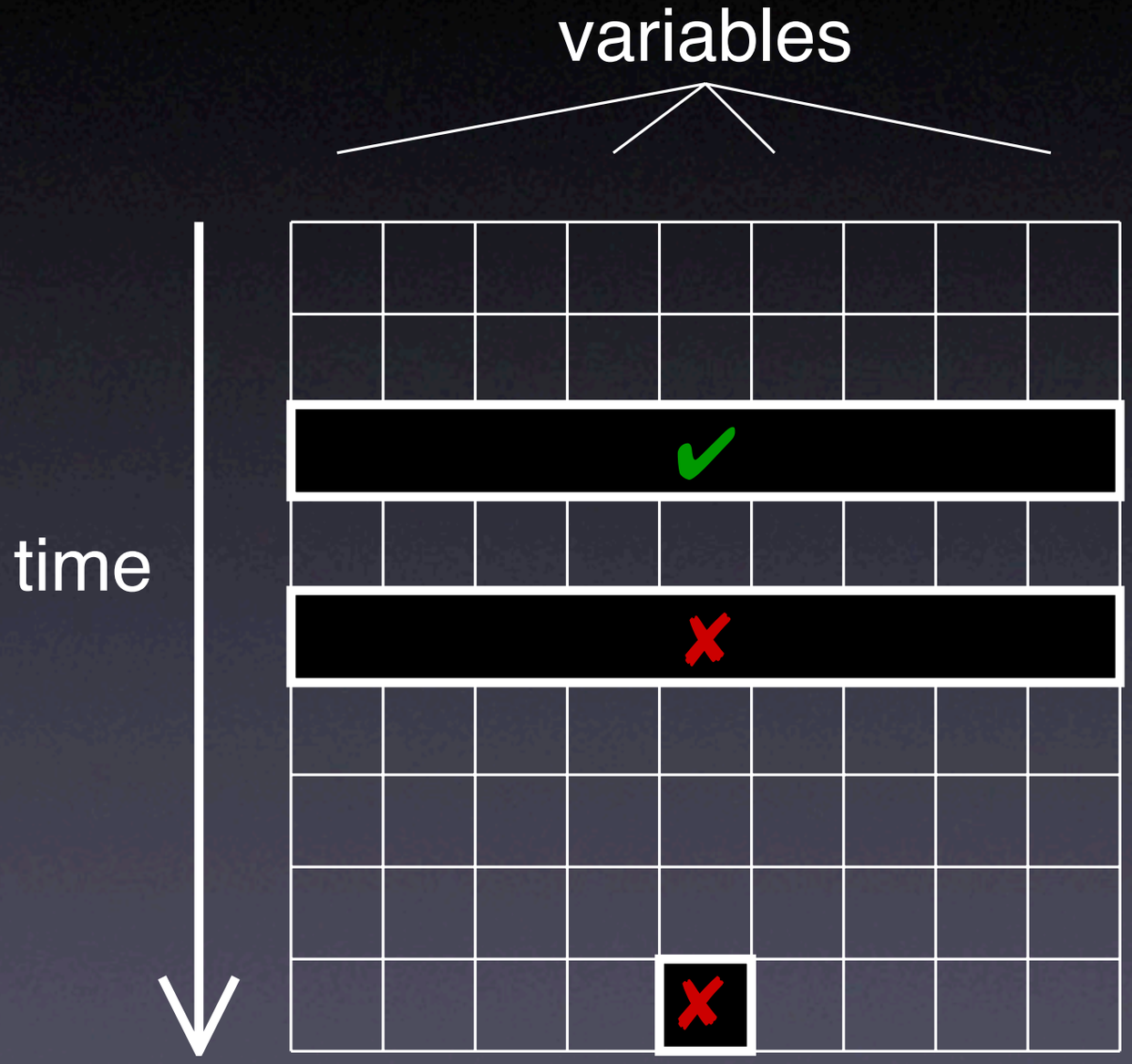
Find Origins



- The 0 printed is the value of `a[0]`. Where does it come from?
- Basic idea: Track or deduce value origins
- Separates *relevant* from *irrelevant* values
- We can trace back `a[0]` to `shell_sort`

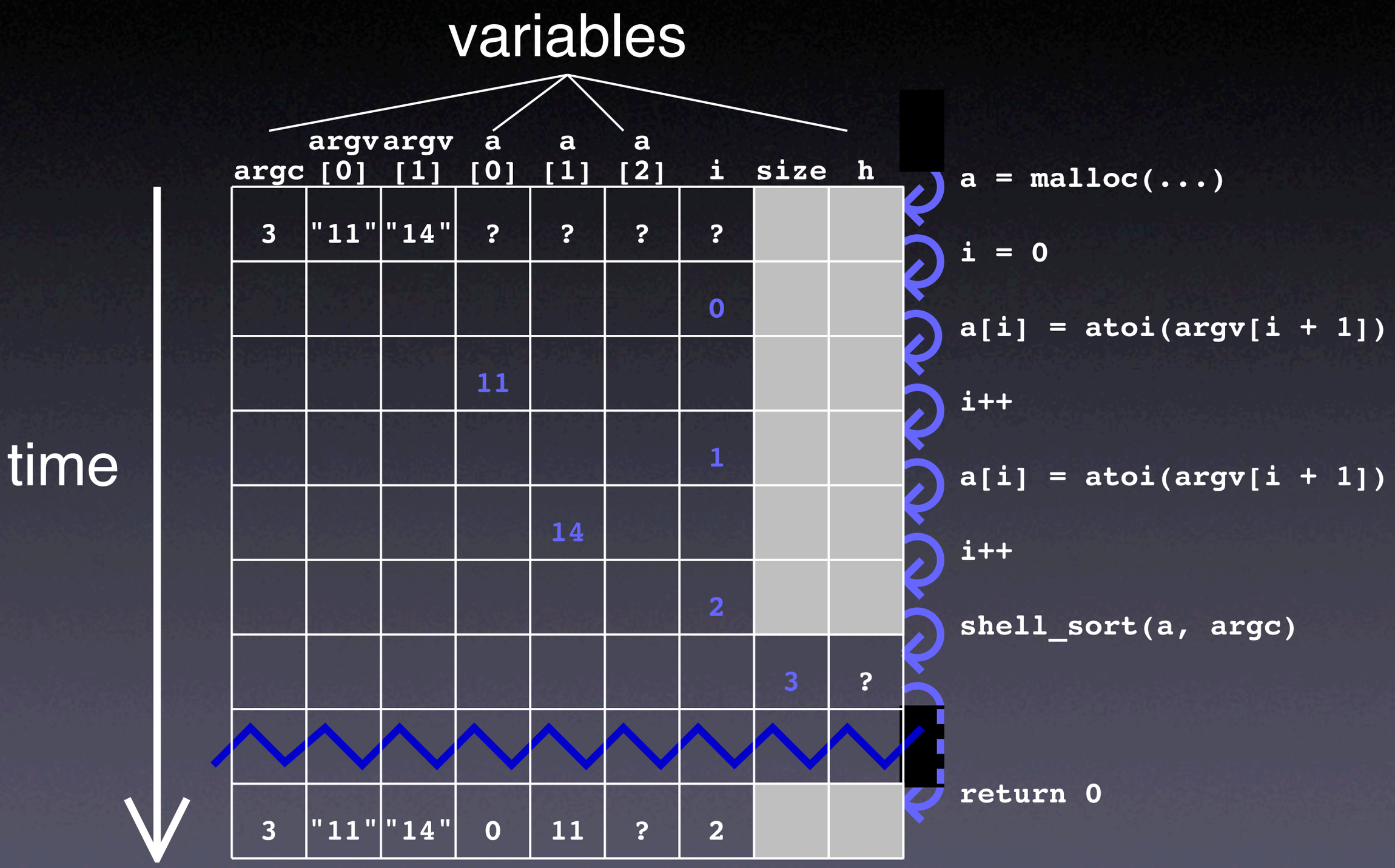
```
static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++)
        {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h)
                a[j] = a[j - h];
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}
```

Search in Time



- In shell_sort, the state must have become infected.
- Basic idea: Observe a transition from sane to infected.

Observing a Run



Specific Observation

```
static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    ...
}
```

Specific Observation

```
static void shell_sort(int a[], int size)
{
    fprintf(stderr, “At shell_sort”);
    for (i = 0; i < size; i++)
        fprintf(stderr, “a[%d] = %d\n”, i, a[i]);
    fprintf(stderr, “size = %d\n”, size);
    int i, j;
    int h = 1;
    ...
}
```

Specific Observation

```
static void shell_sort(int a[], int size)
{
    fprintf(stderr, “At shell_sort”);
    for (i = 0; i < size; i++)
        fprintf(stderr, “a[%d] = %d\n”, i, a[i]);
    fprintf(stderr, “size = %d\n”, size);
    int i, j;
    int h = 1;
    ...
}
```

```
$ sample 11 14
a[0] = 11
a[1] = 14
a[2] = 0
size = 3
Output: 0 11
```

Specific Observation

```
static void shell_sort(int a[], int size)
{
    fprintf(stderr, “At shell_sort”);
    for (i = 0; i < size; i++)
        fprintf(stderr, “a[%d] = %d\n”, i, a[i]);
    fprintf(stderr, “size = %d\n”, size);
    int i, j;
    int h = 1;
    ...
}
```

```
$ sample 11 14
a[0] = 11
a[1] = 14
a[2] = 0
size = 3
Output: 0 11
```

The state is infected at the call of shell_sort!

Fixing the Program

```
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    ...
}
```

Fixing the Program

```
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    ...
}
```

Fixing the Program

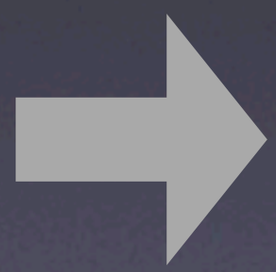
```
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);


    ...
}
```

Fixing the Program

```
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc - 1);

    ...
}
```

Fixing the Program

```
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc - 1);

    ...
}
```



```
$ sample 11 14
Output: 11 14
```

Erster Teil des TRAFFIC-Prinzips

- Track the problem
  Reproduce the failure
  Automate and simplify the test case(s)
  Find possible infection origins
  Focus on likely origins
  Isolate the infecton
  Correct the problem

Tracking Problems

Andreas Zeller

What's a problem?

- A *problem* is a questionable property of a program run
- It becomes a *failure* if it's incorrect…
- …a *request for enhancement* if missing…
- …and a *feature* if normal behavior.

It's not a bug, it's a feature!

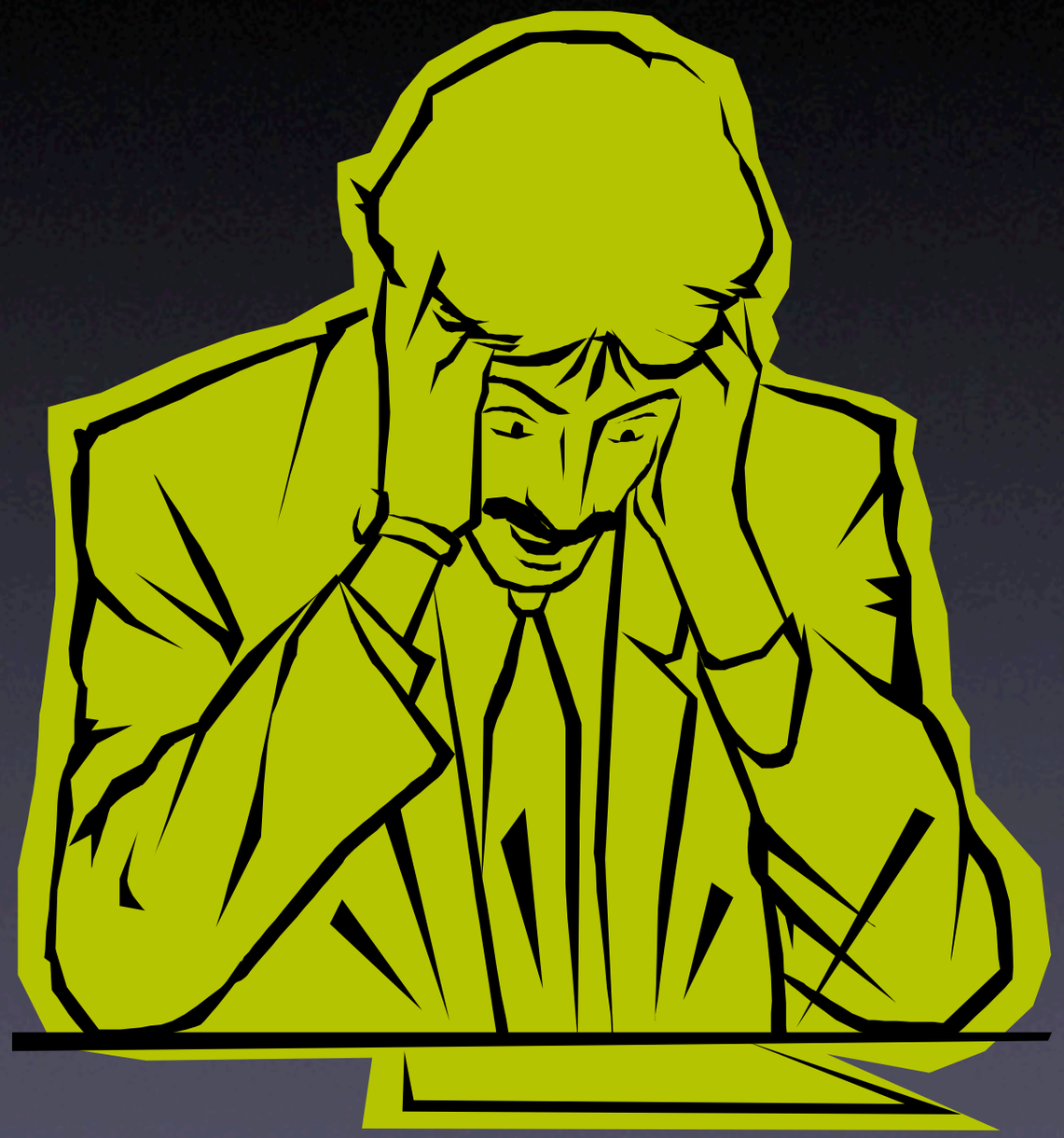
Problem Life Cycle

- The user *informs* the vendor about some problem.
- The vendor
  1. *reproduces* the problem
  2. *isolates* the circumstances
  3. *locates* and *fixes* the defect
  4. *delivers* the fix to the user.

Vendor Challenges

- How do I organize the life cycle?
- Which problems are currently open?
- Which are the most severe problems?
- Did similar problems occur in the past?

User Challenges



Solve my problem!

Problem Report

- A problem comes to life with a *problem report.*

- A problem report includes all the information the vendor needs to fix the problem.
- Also known as *change request* or *bug report.*

Problem report #1

From: me@dot.com
To: zeller@gnu.org
Subject: Crash

Your program crashed. (core dumped)

Problem report #2

From: me@dot.com
To: zeller@gnu.org
Subject: Re: Crash

Sorry, here's the core - cu

<core, 14MB>

Problem report #3

From: me@dot.com
To: zeller@gnu.org
Subject: Re: Crash

You may need that, too (just in case)


<drive_c.zip, 148GB>

What to report

- The *product release*
- The *operating environment*
- The *problem history*
- *Expected* and *experienced behavior*
- *A* one-line *summary*

Product Release

- Typically, some *version number* or otherwise unique identifier
- Required to *reproduce the exact version:*

  Perfect Publishing Program 1.1 (Build 7E47)
- Generalize: Does the problem occur only in this release?

Operating Environment

- Typically, *version information* about the operating system
- Can be simple (”Windows 98 SE”) or complex (”Debian Linux ‘Lenny’ with the following packages…”)
- Generalize: In which environments does the problem occur?

Systemressourcen

- Speicherplatz (RAM, Festplatte)
- CPU-Typ
  - unterstützte SSE-Version (Optimierung!)
  - unterstützte Virtualisierung
- Sonstige Hardware
  - Ausgabe von lspci

Problem History

- Steps needed to *reproduce* the problem:

1. Create “bug.ppp”
2. Print on the default printer…

- If the problem cannot be reproduced, it is unlikely to be fixed
- Simplify: Which steps are relevant?

Expected Behavior

- What should have happened according to the user:

  The program should have printed the document.

- Reality check: What's the understanding of the user?

Observed Behavior

- The *symptoms* of the problem — in contrast to the *expected* behavior

```
The program crashed with the following information

*** STACK DUMP OF CRASH (LemonyOS)

 Back chain  ISA  Caller
 00000000    SPC  0BA8E574
 03EADF80    SPC  0B742428
 03EADF30    SPC  0B50FDDC  PrintThePage+072FC
SnicketPC unmapped memory exception at
        0B512BD0 PrintThePage+05F50
```

A one-line summary

- Captures the essential of the problem

  PPP 1.1 crashes when printing

Things to avoid

- Humor

  PPP (oops, gotta go to the restroom :-) …

- Sarcasm

  Here's yet another "never-to-be-fixed" bug

- Attacks

  If you weren't too incompetent to grasp…

Talk back

Netscape Quality Feedback Agent Prompt - Gecko1.4

NETSCAPE

The Netscape Quality Feedback Agent has captured information that Netscape needs to help improve Communicator's quality.

Enter your email address (optional), describe how you were using Communicator (optional), then click Send.

Your Email Address (optional):

Andreas Zeller <zeller@cs.uni-sb.de>

Send me information about updates to Netscape products

If failure occured within the browser please provide URL/location address of the page you were browsing

about:config

Describe what you were doing when communicator failed (optional):

Printing "about:config" crashes.

FULL CIRCLE SOFTWARE

Netscape Quality Feedback Agent
Talkback is a Trademark of Full Circle

Send | Don't Send

Details

Application Information
+ Agent Configuration Version
+ Agent Library Version
+ Application Launch Time
+ Build Identifier
+ Deployment Identifier
+ Interface Version
+ Monitor Configuration Version
+ Platform Identifier
+ Product Identifier

Description

Identifier used to uniquely identify which application was running at the time information was captured.

Include item selected above

NetscapeGecko1.4LinuxIntel2003061711

Save As... | Close | Help

Talk Back + Privacy

- Be sure what to collect and include in an automated report:
  - Pages visited
  - Text entered
  - Images viewed…
- *Privacy* is an important issue here!

All these Problems

All these Problems

001 It's too big and too slow. [This one will never get fixed]

003 (Motif 1.1) The command window is scrolled whenever obscured.

021 (DBX) Using SunOS DBX, attempting to dereference a `(nil)' pointer
    results in an error message and no new display. However, the
    expression is entered as an ordinary display.

026 (DBX) Using SunOS DBX with PASCAL or Modula-2, selected array
    elements are not counted from the starting index of the array.

041 Starting a multi-window DDD iconified under vtwm and fvwm causes
    trouble with group iconification.

272 (LessTif) The `select' font selection method works only once.

281 In auto deiconify mode, the Debugger Console uniconifies even if
    other DDD windows are already there.

286 (Motif) Changing Cut/Copy/Paste accelerators at runtime does not work.

Managing Problems

Managing Problems

- Alternative #1: *A Problem File*

Managing Problems

- Alternative #1: *A Problem File*
  - Only one person at a time can work on it

Managing Problems

- Alternative #1: *A Problem File*
  - Only one person at a time can work on it
  - History of earlier (fixed) problems is lost

Managing Problems

- Alternative #1: *A Problem File*
  - Only one person at a time can work on it
  - History of earlier (fixed) problems is lost
  - Does not scale

Managing Problems

- Alternative #1: *A Problem File*
  - Only one person at a time can work on it
  - History of earlier (fixed) problems is lost
  - Does not scale
- Alternative #2: *A Problem Database*

Bugzilla

Search for bugs – Mozilla {Build ID: 2002072204}

File Edit View Go Bookmarks Tools Window Help Debug QA

Search

mozilla.org

Bugzilla Version 2.17

This is **Bugzilla**: the Mozilla bug system. For more information about what Bugzilla is and what it can do, see mozilla.org's **bug pages**.

Search for bugs

Summary: contains all of the words/strings | Search

Product:	Component:	Version:	Target:
Browser Bugzilla Calendar CCK Chimera	Accessibility Accessibility APIs Account Manager Address Book Addressbook/LDAP (non-UI)	1.01 1.1 1.2 1.3 1.4	--- Future 3.0 Jan M1

A comment: contains all of the words/strings

The URL: contains all of the words/strings

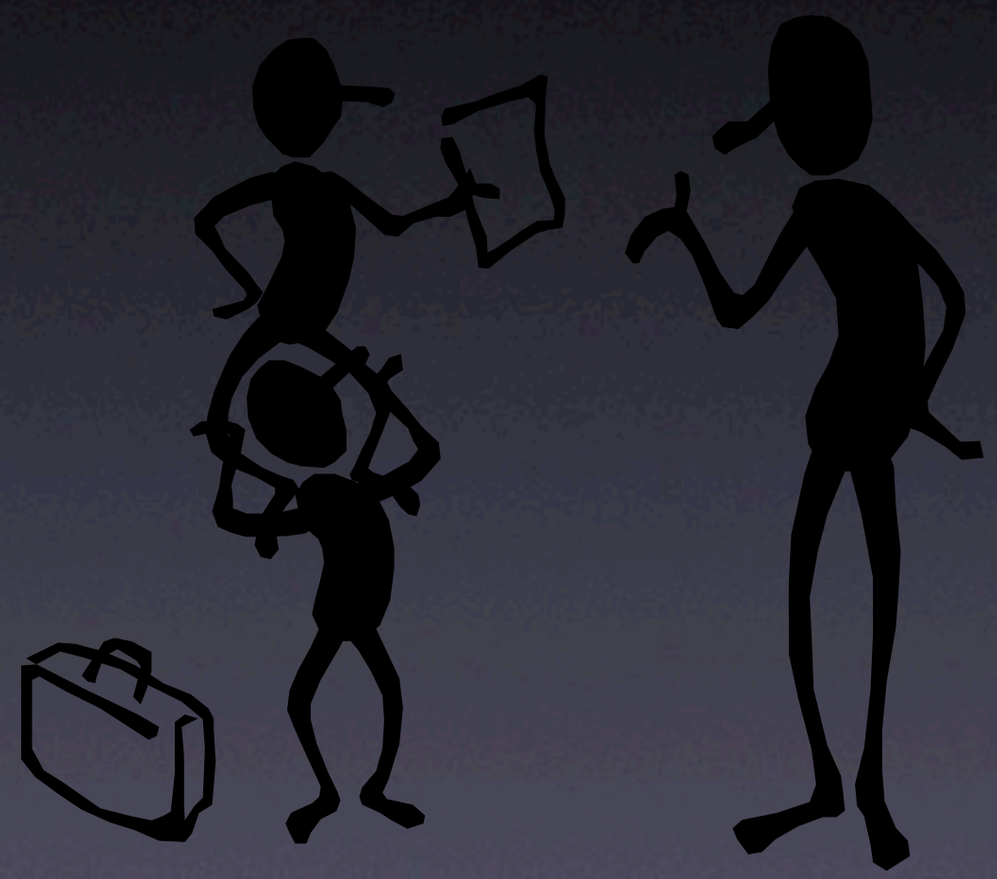
Whiteboard: contains all of the words/strings

Keywords: contains all of the keywords

Status:	Resolution:	Severity:	Priority:	Hardware:	OS:
UNCONFIRMED NEW ASSIGNED REOPENED RESOLVED VERIFIED CLOSED	FIXED INVALID WONTFIX LATER REMIND DUPLICATE WORKSFORME	blocker critical major normal minor trivial enhancement	-- P1 P2 P3 P4 P5	All DEC HP Macintosh PC SGI Sun	All Windows 3.1 Windows 95 Windows 98 Windows ME Windows 2000 Windows NT

Classifying Problems

- Severity
- Priority
- Identifier
- Comments
- Notification



Severity

Severity

Enhancement. A desired feature.

Severity

Enhancement. A desired feature.

Trivial. Cosmetic problem.

Severity

Enhancement. A desired feature.

Trivial. Cosmetic problem.

Minor. Problem with easy workaround.

Severity

Enhancement. A desired feature.

Trivial. Cosmetic problem.

Minor. Problem with easy workaround.

Normal. "Standard" problem.

Severity

Enhancement. A desired feature.

Trivial. Cosmetic problem.

Minor. Problem with easy workaround.

Normal. "Standard" problem.

Major. Major loss of function.

Severity

Enhancement. A desired feature.

Trivial. Cosmetic problem.

Minor. Problem with easy workaround.

Normal. "Standard" problem.

Major. Major loss of function.

Critical. Crashes, loss of data or memory

Severity

Enhancement. A desired feature.

Trivial. Cosmetic problem.

Minor. Problem with easy workaround.

Normal. "Standard" problem.

Major. Major loss of function.

Critical. Crashes, loss of data or memory

Showstopper. Blocks development.

Klassifikation während Fehlerbearbeitung

fragwürdige Programmeigenschaft

„Problem“

Feature

Failure

Priority

- Every new problem gets a *priority*
- The higher the priority, the sooner the problem will be addressed
- Priority is independent from severity
- Prioritizing problems is the main tool to control development and problem solving

Identity

- Every new problem gets an *identifier* (also known as *PR number* or *bug number*)
- The identifier is used in all documents during the debugging process:

```
Subject: PR #3427 is fixed?
```

Comments

- Every developer can attach *comments* to a problem:

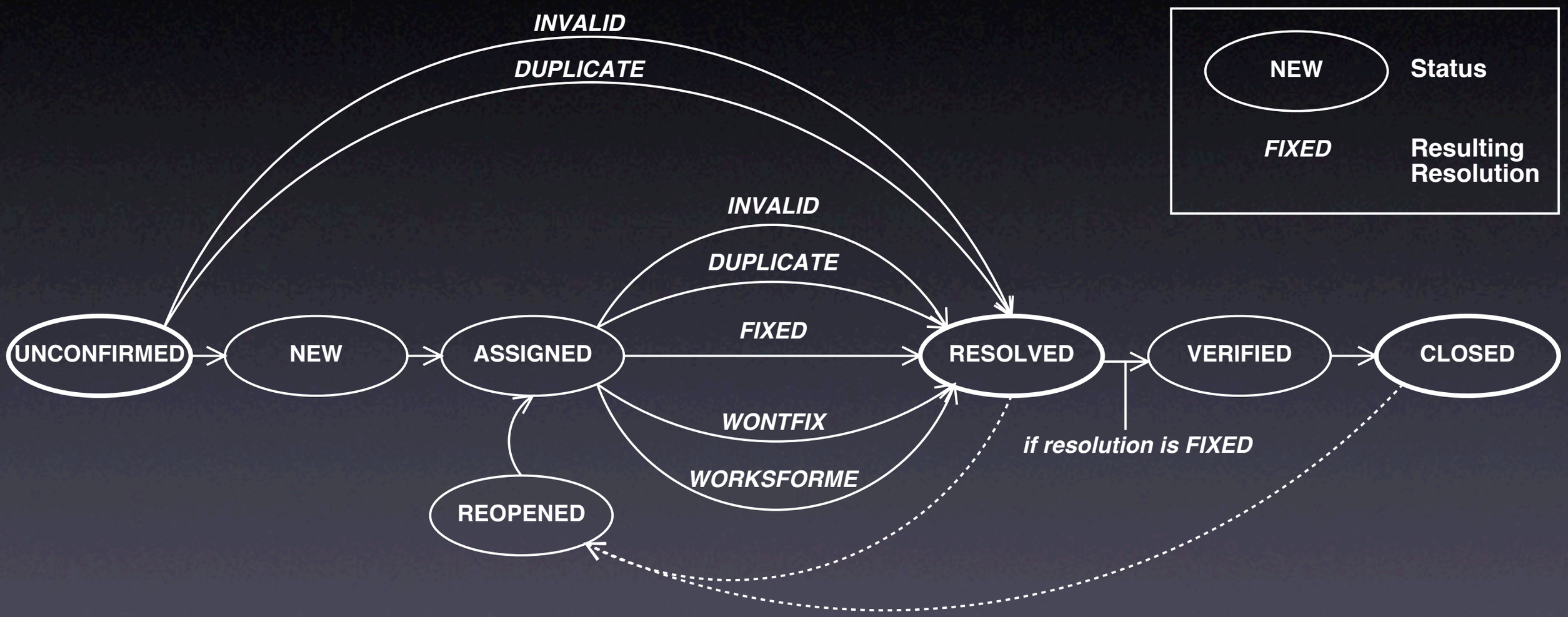
```
I have a patch for this.  It's just an
uninitialized variable but I still
need a review.
```

- Comments may also include files, documents, etc.

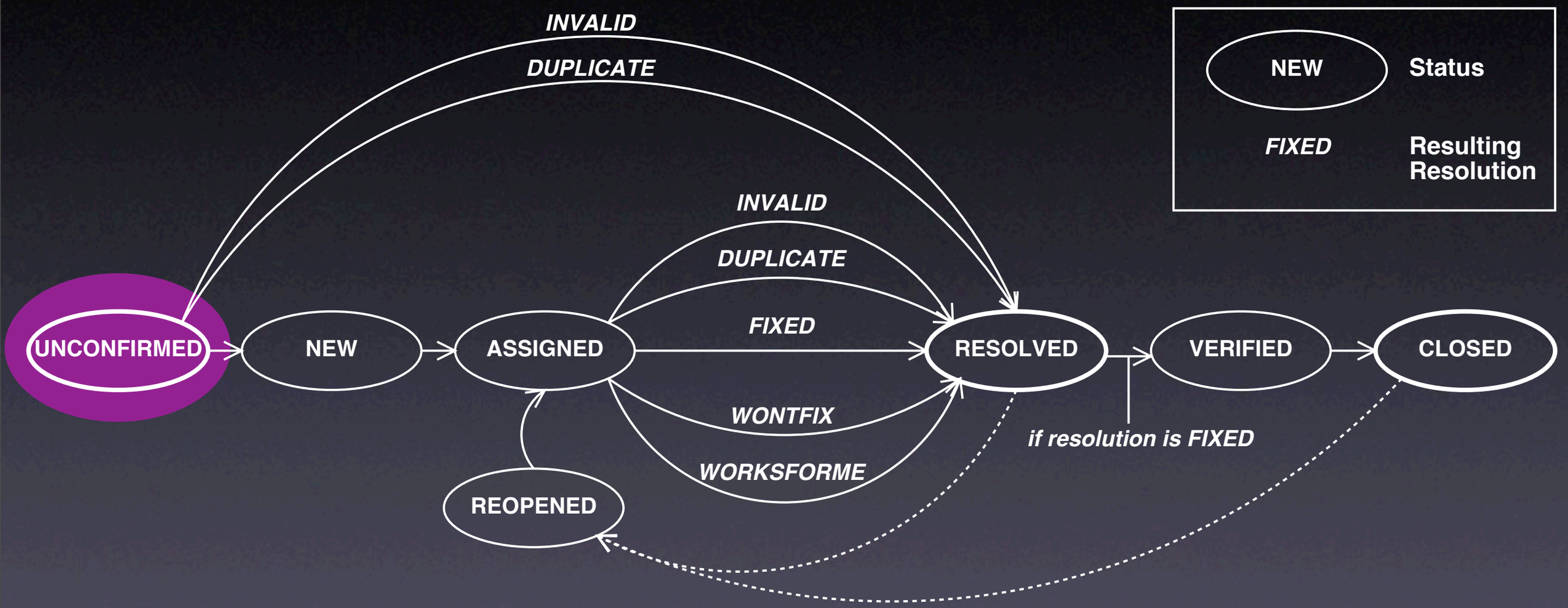
Notification

- Developers can attach an e-mail address to a problem report; they will be notified every time the report changes.
- Users can do so, too.

The Problem Lifecycle

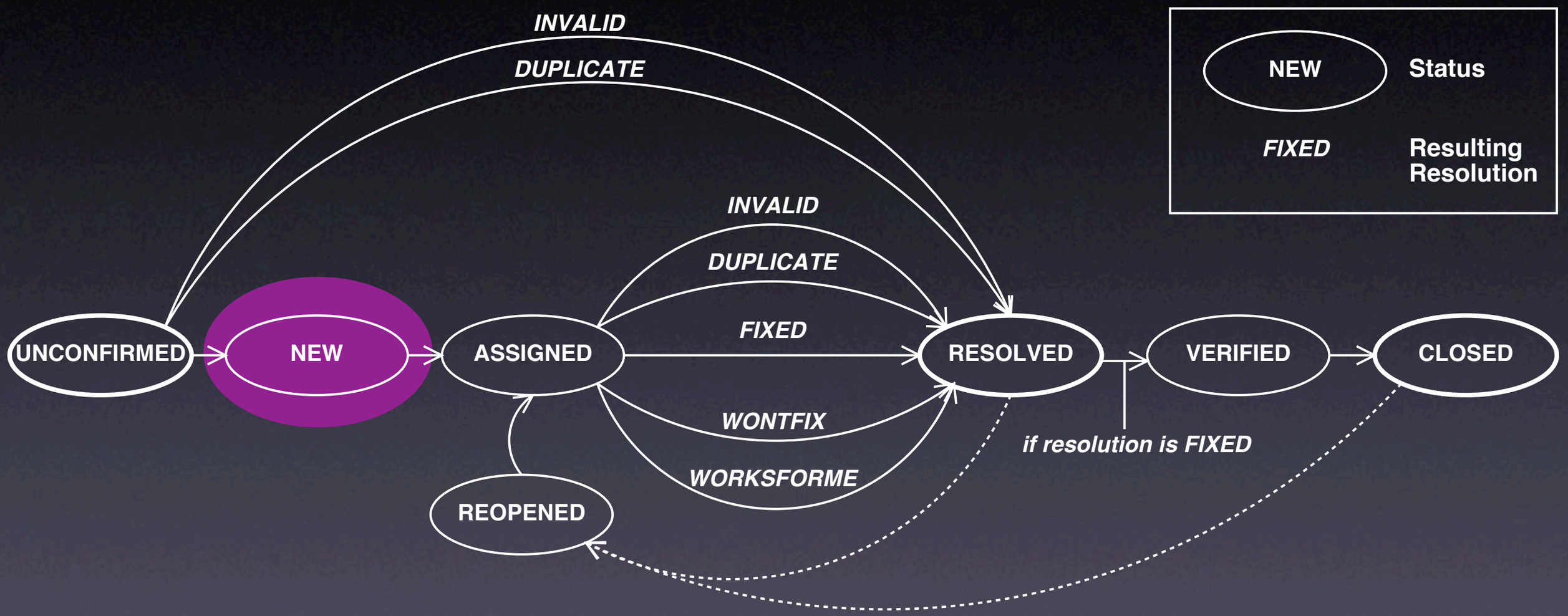


Unconfirmed Problem



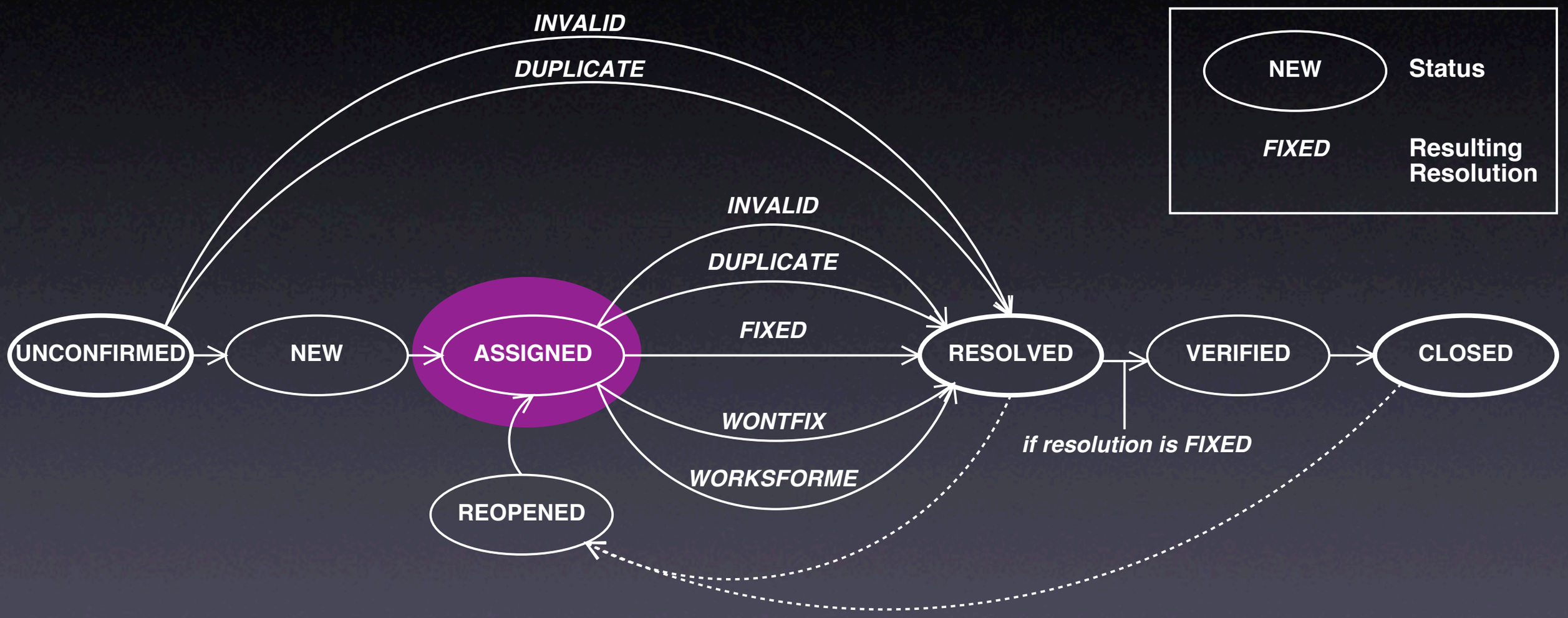
- The problem report has just been entered into the database

New Problem



- The report is *valid* and not a *duplicate*. (If not, it becomes *resolved*.)

Assigned Problem

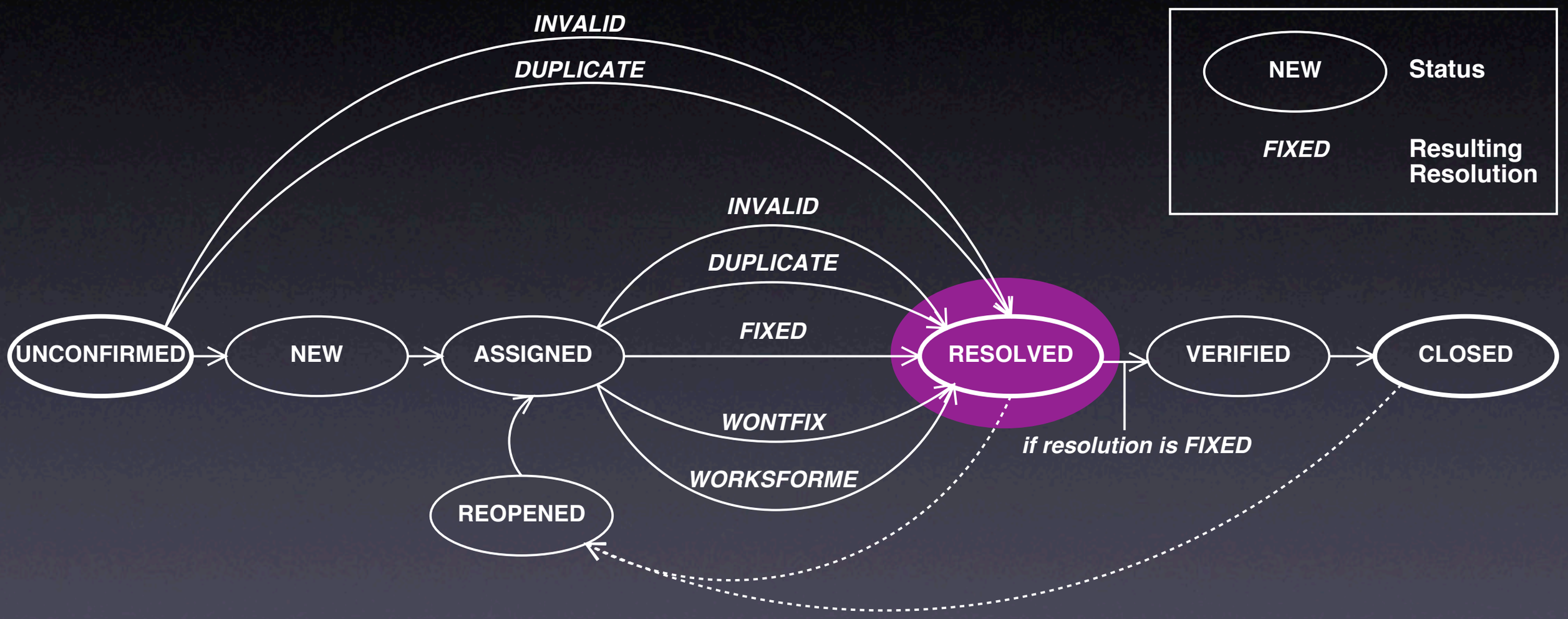


- The problem is assigned to a developer

Resolution

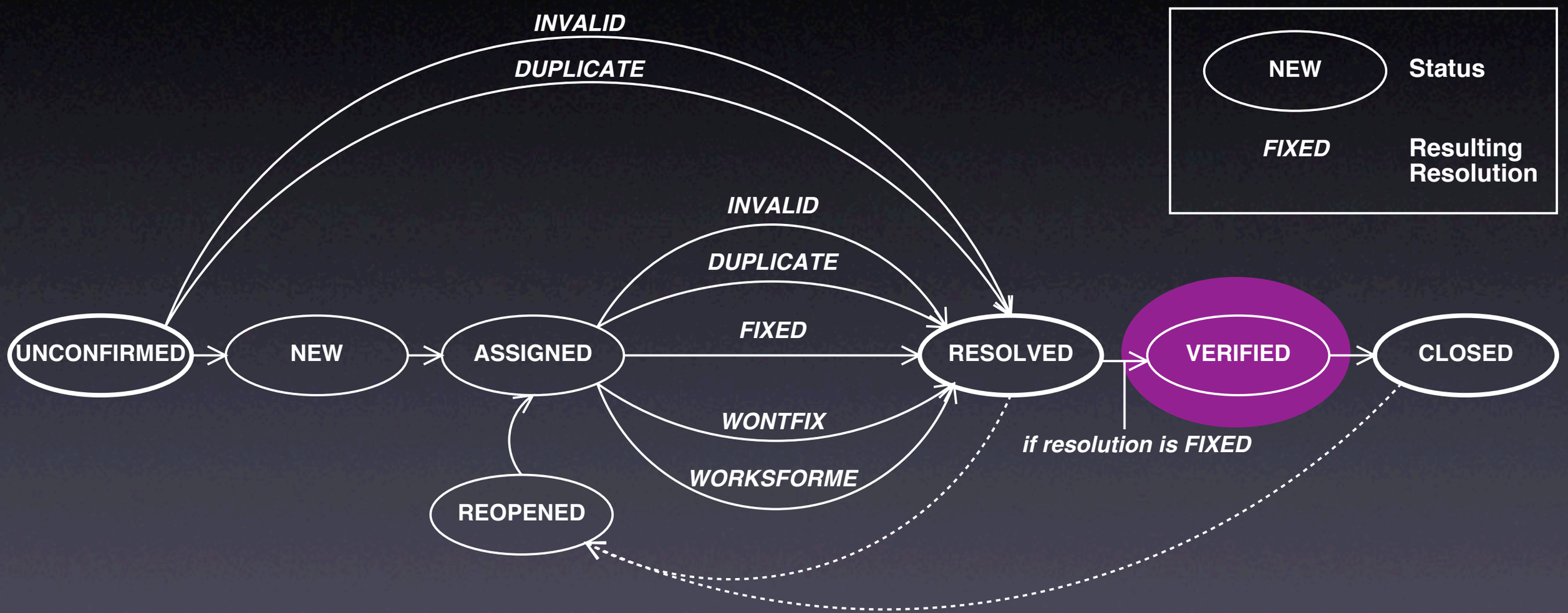
- FIXED: The problem is fixed.
- INVALID: The problem is not a problem.
- DUPLICATE: The problem already exists.
- WONTFIX: Will never be fixed (for instance, because the problem is a feature)
- WORKSFORME: Could not be reproduced.

Resolved Problem



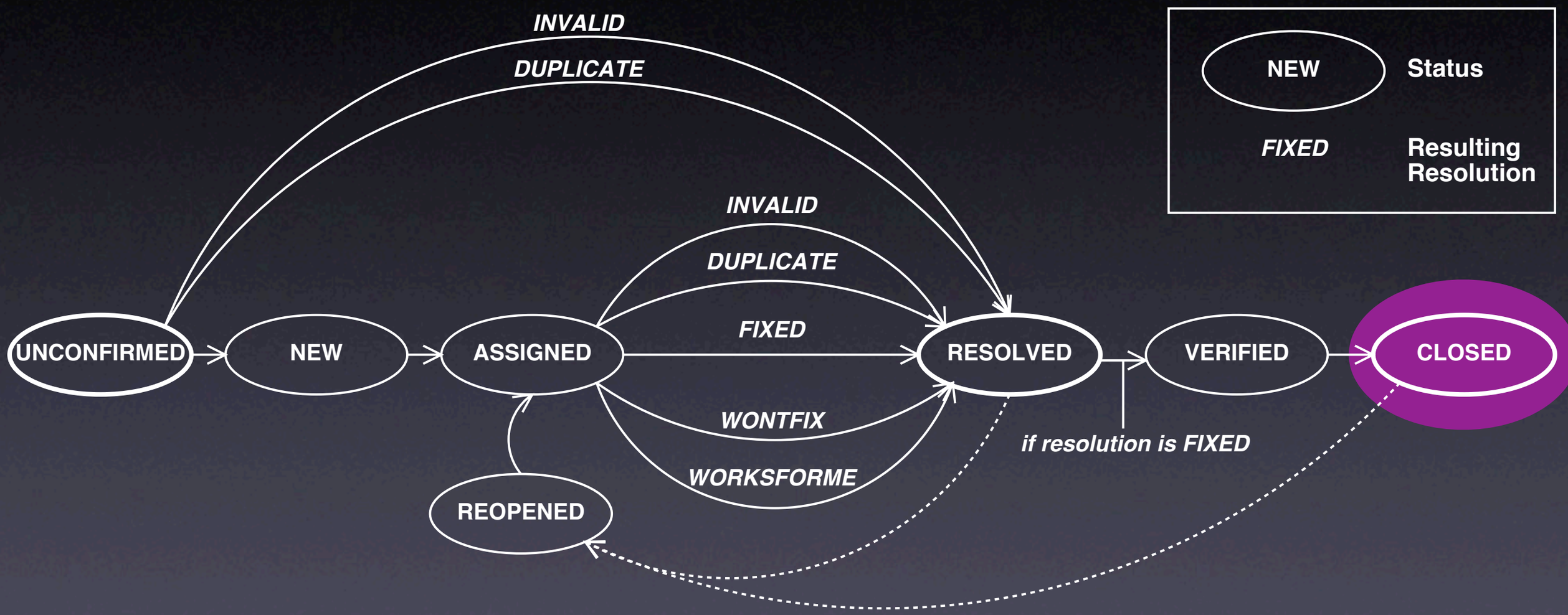
- The problem report has been processed.

Verified Problem



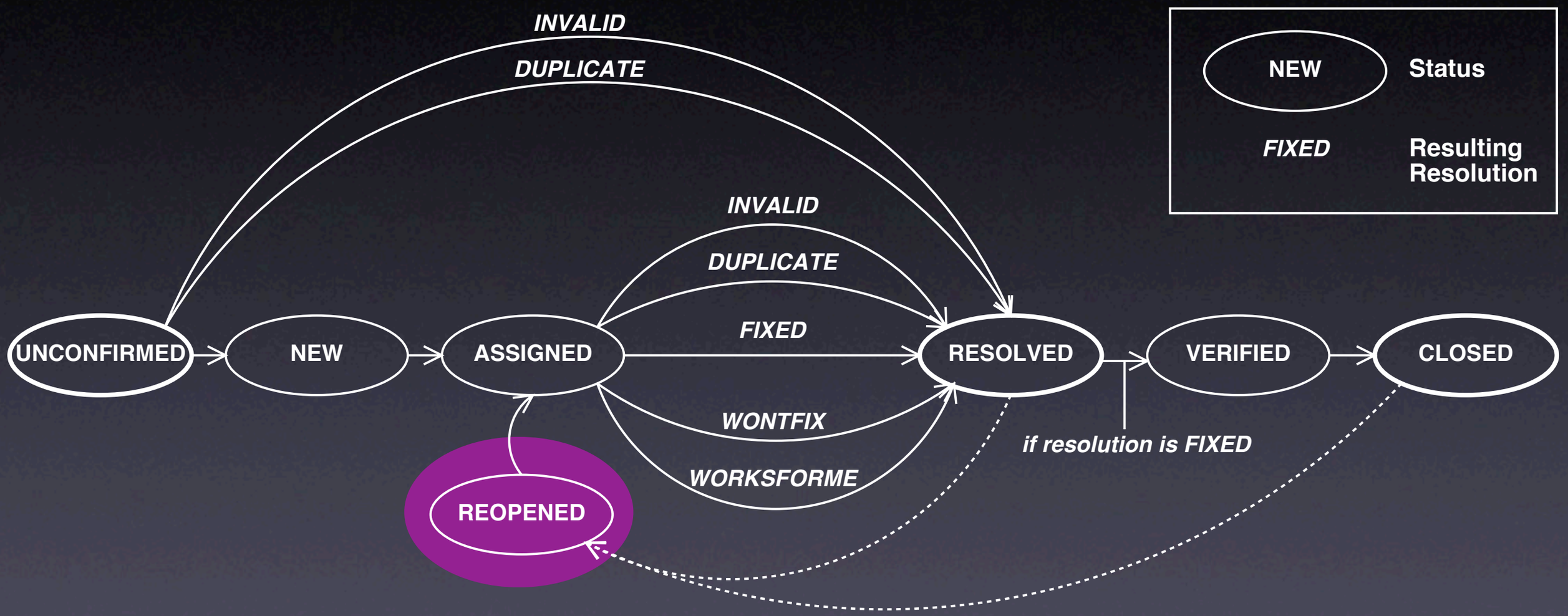
- The problem is fixed; the fix has been successful.

Closed Problem



- A new version with the fix has been released.

Reopened Problem



- Oops – there we go again :–(

Management

Management

- Who *enters* problem reports?

Management

- Who *enters* problem reports?
- Who *classifies* problem reports?

Management

- Who *enters* problem reports?
- Who *classifies* problem reports?
- Who sets *priorities*?

Management

- Who *enters* problem reports?
- Who *classifies* problem reports?
- Who sets *priorities*?
- Who takes care of the problem?

Management

- Who *enters* problem reports?
- Who *classifies* problem reports?
- Who sets *priorities*?
- Who takes care of the problem?
- Who *closes* issues?

The SCCB

- At many organizations, a *software change control board* is in charge of these questions:
  - Assess the *impact* of a problem
  - Assign tasks to developers
  - Close issues…

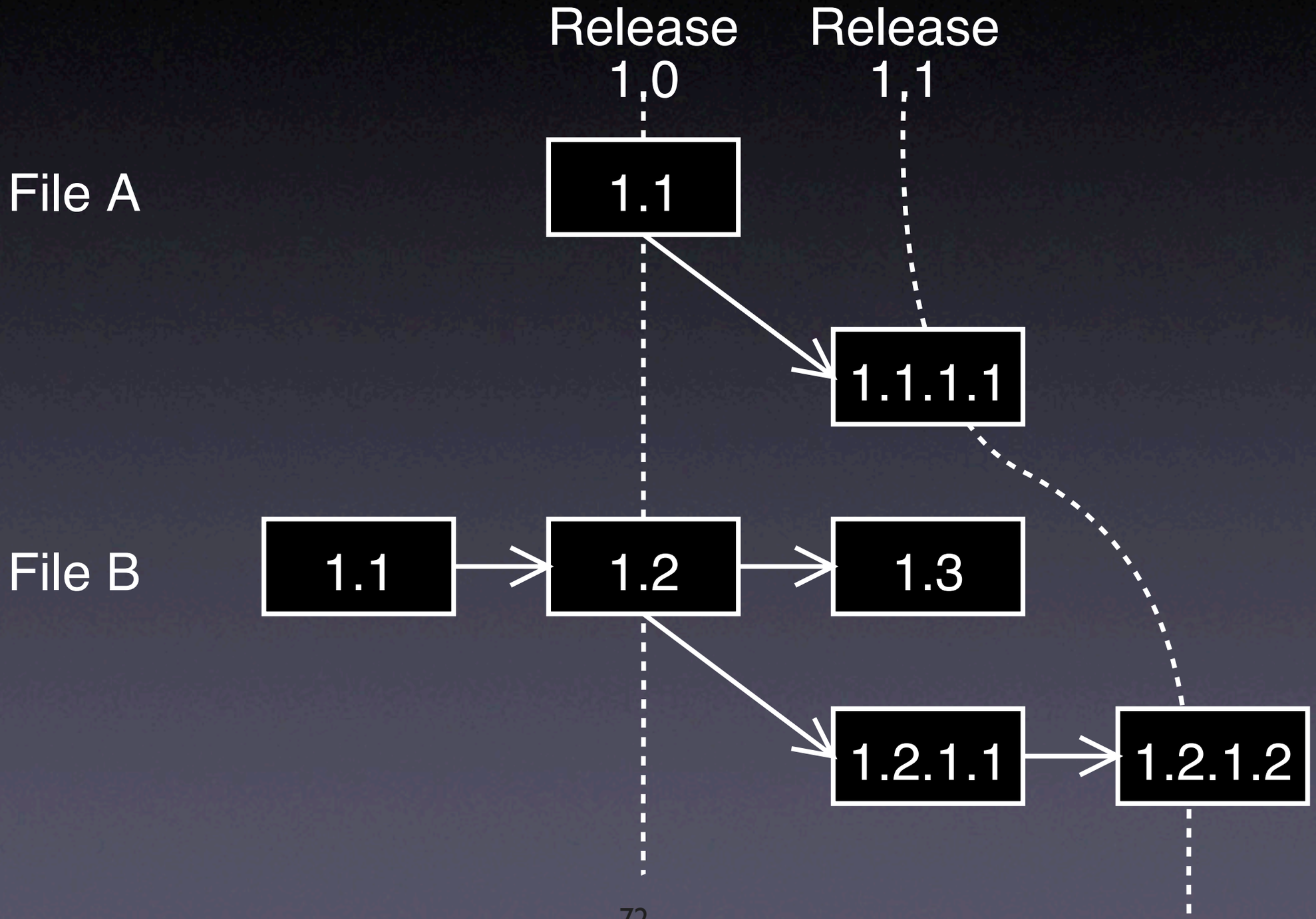
Problem-driven Development

- The whole development can be organized around the problem database:
- Start with one single problem: "The product does not exist"
- Decompose into sub-problems
- Ship when all problems are fixed

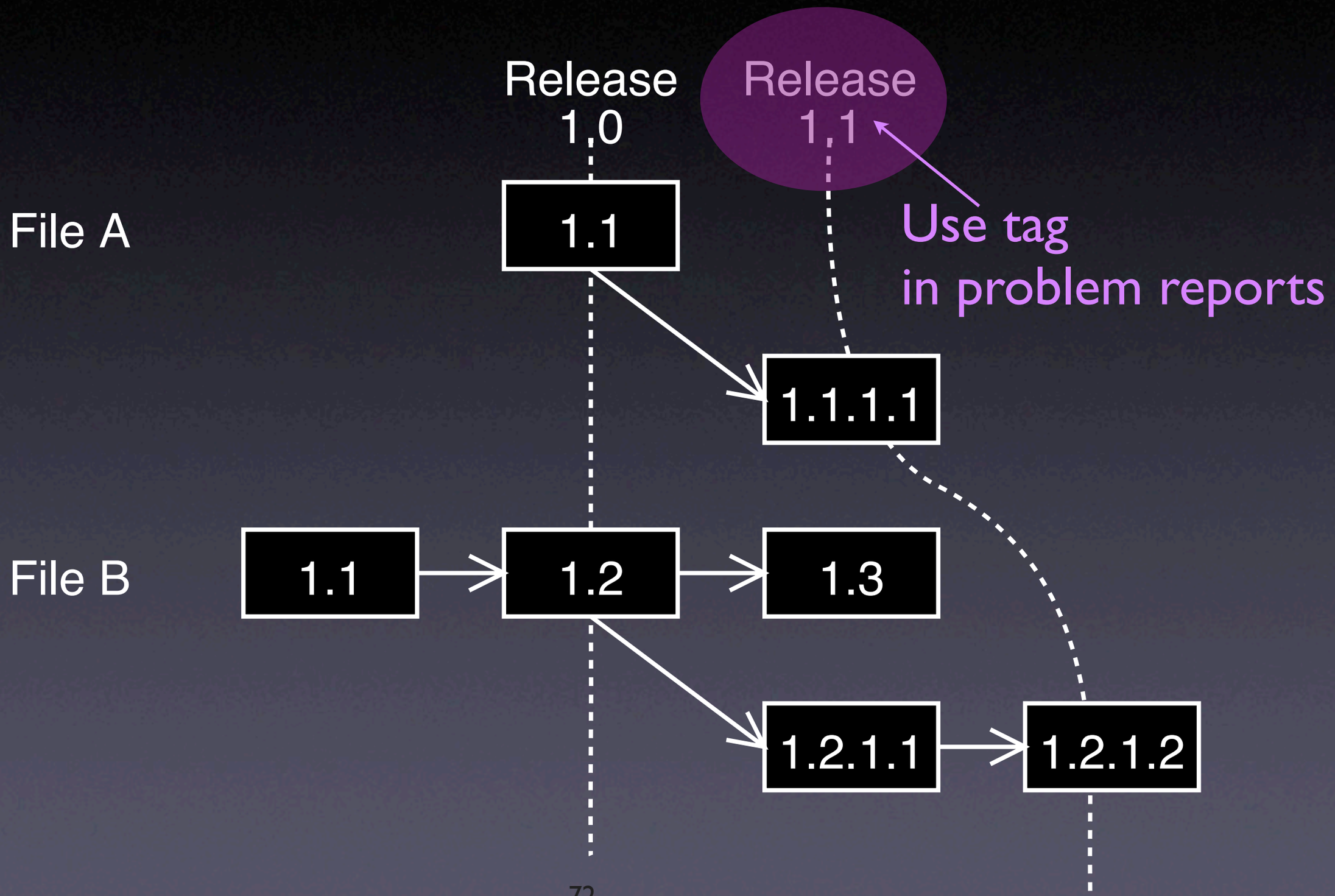
Managing Clutter

- Large problem databases contain *garbage*
- Get rid of *duplicates* by
  - simplifying bug reports
  - asking submitters to search first
- Get rid of *obsolete* problems by searching for old ones that rarely occurred

Problems and Fixes



Problems and Fixes



Problems and Tests

Problems and Tests

- Some test fails. Should we enter the problem into the database?

Problems and Tests

- Some test fails. Should we enter the problem into the database?
- *No*, because test cases make problem reports obsolete.

Problems and Tests

- Some test fails. Should we enter the problem into the database?
- *No*, because test cases make problem reports obsolete.
- Once we can repeat a problem at will, there is no need for a database entry

Danke fürs Zuhören!

Bis nächste Woche :-)