Simplifying Problems

Andreas Zeller

Simplifying

- Once one has tracked and reproduced a problem, one must find out *what's relevant*:
  - Does the problem really depend on 10,000 lines of input?
  - Does the failure really require this exact schedule?
  - Do we need this sequence of calls?

Simplifying

- For every circumstance of the problem, check whether it is relevant for the problem to occur.
- If it is not, remove it from the problem report or the test case in question.

Circumstances

- Any aspect that may influence a problem is a *circumstance:*
  - Aspects of the problem environment
  - Individual steps of the problem history

Experimentation

- By *experimentation*, one finds out whether a circumstance is relevant or not:
- Omit the circumstance and try to reproduce the problem.
- The circumstance is relevant iff the problem no longer occurs.

Mozilla Bug #24735

Ok the following operations cause mozilla to crash consistently on my machine

-> Start mozilla
-> Go to bugzilla.mozilla.org
-> Select search for bug
-> Print to file setting the bottom and right margins to .50
(I use the file /var/tmp/netscape.ps)
-> Once it's done printing do the exact same thing again on the same file (/var/tmp/netscape.ps)
-> This causes the browser to crash with a segfault

<SELECT NAME="op_sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION
VALUE="Windows 95">Windows 95<OPTION VALUE="Windows 98">Windows
98<OPTION VALUE="Windows ME">Windows ME<OPTION VALUE="Windows
2000">Windows 2000<OPTION VALUE="Windows NT">Windows NT<OPTION
VALUE="Mac System 7">Mac System 7<OPTION VALUE="Mac System 7.5">Mac
System 7.5<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1<OPTION
VALUE="Mac System 8.0">Mac System 8.0<OPTION VALUE="Mac System
8.5">Mac System 8.5<OPTION VALUE="Mac System 8.6">Mac System
8.6<OPTION VALUE="Mac System 9.x">Mac System 9.x<OPTION VALUE="MacOS
X">MacOS X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION
VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION
VALUE="BeOS">BeOS
VALUE="IRIX">IRIX
VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION VALUE="OSF/
1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION
VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>

</td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION VALUE="P5">P5</SELECT>

bugzilla.mozilla.org

What's relevant in here?

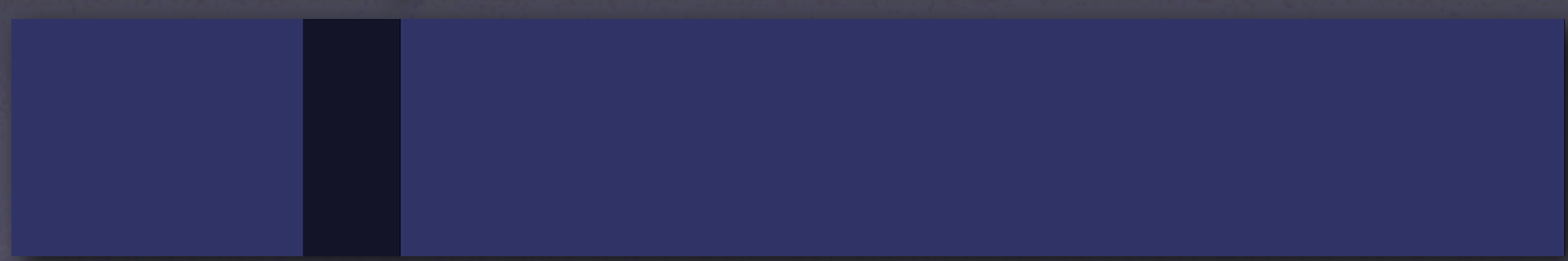
Why simplify?

- Ease of communication. A simplified test case is easier to communicate.
- Easier debugging. Smaller test cases result in smaller states and shorter executions.
- Identify duplicates. Simplified test cases *subsume* several duplicates.

Binary Search

- Proceed by binary search. Throw away half the input and see if the output is still wrong.
- If not, go back to the previous state and discard the other half of the input.

HTML input



Simplified Input

```
<SELECT NAME="priority" MULTIPLE SIZE=7>
```

- Simplified from 896 lines to one single line
- Required 12 tests only

Benefits

- Ease of communication. All one needs is "Printing <SELECT> crashes".
- Easier debugging. We can directly focus on the piece of code that prints <SELECT>.
- Identify duplicates. Check other test cases whether they're <SELECT>-related, too.

Why automate?

- Manual simplification is *tedious*.
- Manual simplification is *boring*.
- We have machines for tedious and boring tasks.

Basic Idea

- We set up an *automated test* that checks whether the failure occurs or not (= Mozilla crashes when printing or not)
- We implement a *strategy* that realizes the binary search.

Automated Test

1. Launch Mozilla
2. Replay (previously recorded) steps from problem report
3. Wait to see whether
   - Mozilla crashes (= the test *fails*)
   - Mozilla still runs (= the test *passes*)
4. If neither happens, the test is *unresolved*

Binary Search

```
<SELECT NAME="priority" MULTIPLE SIZE=7> ✗
<SELECT NAME="priority" MULTIPLE SIZE=7> ✓
<SELECT NAME="priority" MULTIPLE SIZE=7> ✓
<SELECT NAME="priority" MULTIPLE SIZE=7> ✓
<SELECT NAME="priority" MULTIPLE SIZE=7> ✗
<SELECT NAME="priority" MULTIPLE SIZE=7> ✗
<SELECT NAME="priority" MULTIPLE SIZE=7> ✓
```

What do we do if *both halves* pass?

Configuration

Circumstance

$$\delta$$

All circumstances

$$\mathcal{C} = \{\delta_1, \delta_2, \ldots\}$$

Configuration $c \subseteq \mathcal{C}$

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

Tests

Testing function

$$test(c) \in \{✔, ✘, ?\}$$

Failure-inducing configuration

$$test(c_{✘}) = ✘$$

Relevant configuration $c'_{✘} \subseteq c_{✘}$

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

Binary Strategy

Split input

$$c_{✗} = c_1 \cup c_2$$

If removing first half fails…

$$test(c_{✗} \setminus c_1) = ✗ \Longrightarrow c_{✗}{}' = c_{✗} \setminus c_1$$

If removing second half fails…

$$test(c_{✗} \setminus c_2) = ✗ \Longrightarrow c_{✗}{}' = c_{✗} \setminus c_2$$

Otherwise, increase granularity:

$$c_{✗} = c_1 \cup c_2 \cup c_3 \cup c_4$$

$$c_{✗} = c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8$$

General Strategy

Split input into *n* parts (initially 2)

$$c_{✘} = c_1 \cup c_2 \cup \cdots \cup c_n$$

If some removal fails…

$$\exists i \in \{1, \ldots, n\} \cdot test(c_{✘} \setminus c_i) = ✘ \implies \begin{array}{l} c_{✘}{}' = c_{✘} \setminus c_i \\ n' = \max(n-1, 2) \end{array}$$

Otherwise, increase granularity

$$c_{✘}{}' = c_{✘} \quad n' = 2n$$

ddmin in a Nutshell

$c'_{✘} = ddmin(c_{✘})$ is a relevant configuration

$ddmin(c_{✘}) = ddmin'(c'_{✘}, 2)$ with $ddmin'(c'_{✘}, n) =$

$$\begin{cases} c'_{✘} & \text{if } |c'_{✘}| = 1 \\ ddmin'(c'_{✘} \setminus c_i, \max(n-1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot test(c'_{✘} \setminus c_i) = ✘ \\ & \text{(“some removal fails”)} \\ ddmin'(c'_{✘}, \min(2n, |c'_{✘}|)) & \text{else if } n < |c'_{✘}| \text{ (“increase granularity”)} \\ c'_{✘} & \text{otherwise} \end{cases}$$

where $c'_{✘} = c_1 \cup c_2 \cup \cdots \cup c_n$

$$\forall c_i, c_j \cdot c_i \cap c_j = \emptyset \wedge |c_i| \approx |c_j|$$

```
def _ddmin(circumstances, n):
    while len(circumstances) >= 2:
        subsets = split(circumstances, n)

        some_complement_is_failing = 0
        for subset in subsets:
            complement = listminus(circumstances, subset)
            if test(complement) == FAIL:
                circumstances = complement
                n = max(n - 1, 2)
                some_complement_is_failing = 1
                break

        if not some_complement_is_failing:
            if n == len(circumstances):
                break
            n = min(n * 2, len(circumstances))

    return circumstances
```

ddmin at Work

Input: `<SELECT NAME="priority" MULTIPLE SIZE=7>` ⟨40 characters⟩ ✘
`<SELECT NAME="priority" MULTIPLE SIZE=7>` ⟨0 characters⟩ ✔

```
 1 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨20⟩ ✔
 2 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨20⟩ ✔
 3 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨30⟩ ✔
 4 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨30⟩ ✘
 5 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨20⟩ ✔
 6 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨20⟩ ✘
 7 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
 8 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
 9 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨15⟩ ✔
10 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨15⟩ ✔
11 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨15⟩ ✘
12 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
13 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
14 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
15 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨12⟩ ✔
16 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨13⟩ ✔
17 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨12⟩ ✔
18 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨13⟩ ✘
19 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
20 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
21 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨11⟩ ✔
22 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✘
23 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
24 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨8⟩ ✔
25 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
26 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨8⟩ ✔
27 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨9⟩ ✔
28 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨9⟩ ✔
29 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨9⟩ ✔
30 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨9⟩ ✔
31 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨8⟩ ✔
32 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨9⟩ ✔
33 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨8⟩ ✘
34 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
35 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
36 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
37 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
38 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
39 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨6⟩ ✔
40 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
41 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
42 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
43 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
44 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
45 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
46 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
47 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
48 <SELECT NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
```

Result: `<SELECT>`

Complexity

- The maximal number of *ddmin* tests is

$$\frac{(|c_{\times}|^2 + 7|c_{\times}|)}{2}$$

Worst Case Details

First phase: every test is unresolved

$$t = 2 + 4 + 8 + \cdots + 2|c_{\times}|$$
$$= 2|c_{\times}| + |c_{\times}| + \frac{|c_{\times}|}{2} + \frac{|c_{\times}|}{4} + \cdots = 4|c_{\times}|$$

Second phase: testing *last* set always fails

$$t' = (|c_{\times}| - 1) + (|c_{\times}| - 2) + \cdots + 1$$
$$= 1 + 2 + 3 + \cdots + (|c_{\times}| - 1)$$
$$= \frac{|c_{\times}|(|c_{\times}| - 1)}{2} = \frac{|c_{\times}|^2 - |c_{\times}|}{2}$$

Binary Search

If

- there is only one failure-inducing circumstance, and
- all configurations that include this circumstance fail,

the number of tests is $t \leq \log_2(|c_{\times}|)$

More Simplification

Simplified failure-inducing *fuzz* input:

- FLEX crashes on 2,121 or more non-newline characters
- NROFF crashes on “\D^J%0F” or “\302\n”
- CRTPLOT crashes on “t”

Minimal Interaction

Ok the following operations cause mozilla to crash consistently on my machine

-> Start mozilla
-> Go to bugzilla.mozilla.org
-> Select search for bug
-> Print to file setting the bottom and right margins to .50 (I use the file /var/tmp/netscape.ps)
-> Once it's done printing do the exact same thing again on the same file (/var/tmp/netscape.ps)
-> This causes the browser to crash with a segfault

Minimal Interaction

Basic idea:
Apply *ddmin* to recorded user interaction

- To reproduce the Mozilla printing crash:
  - Press *P* while holding *Alt*
  - Press *mouse button 1*
  - Release *mouse button 1*

Optimization

- Caching
- Stop Early
- Syntactic Simplification
- Isolate Differences, not Circumstances

Caching

- Basic idea: store the results of earlier test()
- Saves 8 out of 48 tests in <SELECT> example

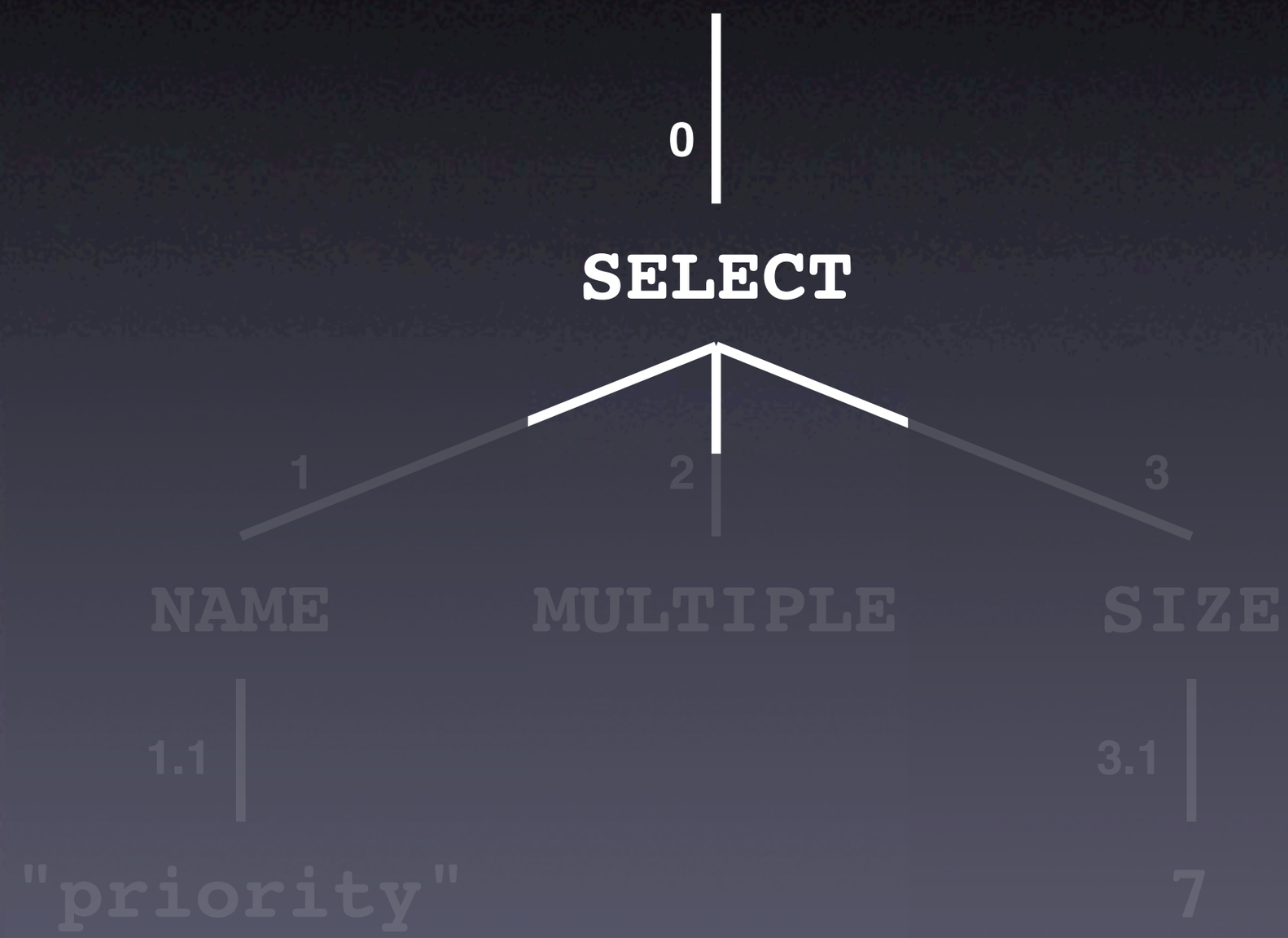
Stop Early

One may stop simplification when

- a certain *granularity* has been reached
- no *progress* has been made
- a certain *amount of time* has elapsed

Syntactic Simplification

```
<SELECT NAME="priority" MULTIPLE SIZE=7>
```



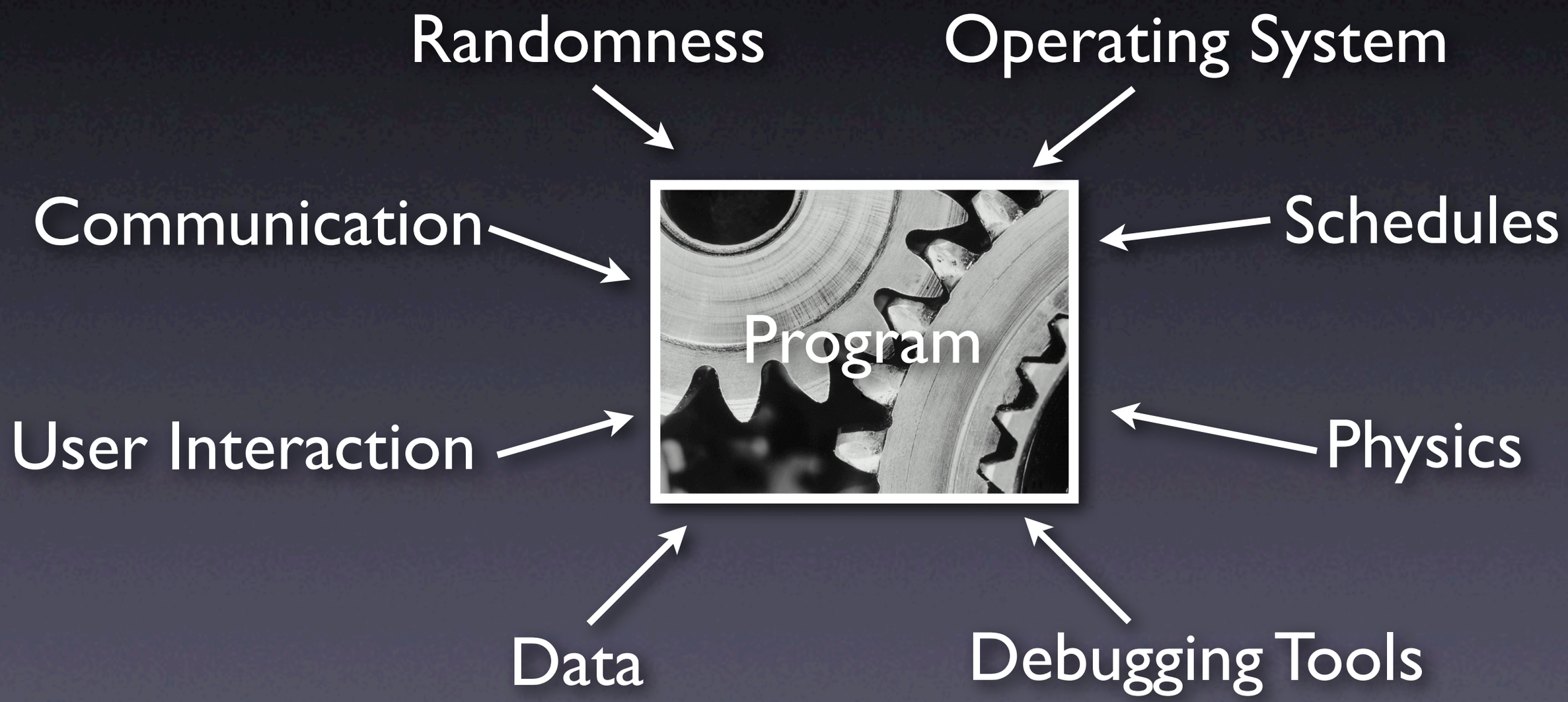
Differences

```
<SELECT NAME="priority" MULTIPLE SIZE=7>
```

The extra “<” is failure-inducing!

```
<SELECT NAME="priority" MULTIPLE SIZE=7>
```

More Circumstances



More Automation

- Failure-Inducing Input
- Failure-Inducing Code Changes
- Failure-Inducing Schedules
- Failure-Inducing Program States
- Failure-Inducing Method Calls

Failure Cause

Lenhof, Hans-Peter,, +49 681 302-64701, lenhof@cs.uni-sb.de
Lindig, Christian, +49 681 9378406, +49 681 302 7790, lindig@cs.uni-sb.de
Mehlmann, Martin,,, mehlmann@st.cs.uni-sb.de
Meyer zu Tittingdorf, Friederike, +49 6894 390032, +49 681 302-78099, meyer@cs.uni-sb.de
Mileva, Yana,, +49 681 302-64020, mileva@cs.uni-sb.de
Müller-Perich, Elisabeth,, +49 681 302-7070, sekr.techfak@rz.uni-sb.de
Nir-Bleimling, Naomi,, +49 68130264011, naomi@wjpserver.cs.uni-sb.de
Offergeld, Thilo,, +49 681 302-6794, t.offergeld@univw.uni-sb.de
PC, CC 2006,,, cc2006pc@st.cs.uni-sb.de
Paul, Wolfgang, +4968171827, +49 6813022436, wjp@cs.uni-sb.de
Premraj, Rahul, +44 7796973711, +49 681 302-64013, premraj@cs.uni-sb.de
Reindel, Erich, +49 6371 912842, +49 681 302-78091, reindel@cs.uni-sb.de
Schuler, David,, +49 681 302-64026, schuler@st.cs.uni-sb.de
Schuler, Erika,, +49 6813027069, schuler@tf.uni-sb.de
Schäfer, Christa, +49 6897 71167, +49 68130264011,
Security, AG,,, security@st.cs.uni-sb.de
Seidel, Raimund, +49 6894 383698, +49 681 302-4713, rseidel@cs.uni-sb.de
Sekretariat, Sekretariat,, +49 681 302-64011, office@st.cs.uni-sb.de
Sliwerski, Jacek, +491741333208,, sliwers@st.cs.uni-sb.de
Slusallek, Philipp, +49 6826 1 88 71 32, +49 681 302-3830, slusallek@cs.uni-sb.de
Slusallek USA, Philipp, +1 670 391 9186, +1 408 486 2788, slusallek@cs.uni-sb.de
Smolka, Gert, +49 681 782770, +49 681 302-7311, smolka@ps.uni-sb.de
Software-Evolution, AG,,, softevo@st.cs.uni-sb.de
Thiel, Frank,,, hausmeister@cs.uni-sb.de
Weiß, Cathrin,,, weiss@st.cs.uni-sb.de
Wilhelm, Reinhard,, +49 681 302-4399, wilhelm@cs.uni-sb.de
Zeller, Andreas,,, zeller@cs.uni-sb.de
Zeller, Andreas, +49 681 3710467, +49 681 302-64011, zeller@cs.uni-sb.de
Zimmermann, Tom, +49 871 71742 (Eltern), +1 403 210 9470, zimmerth@cs.uni-sb.de

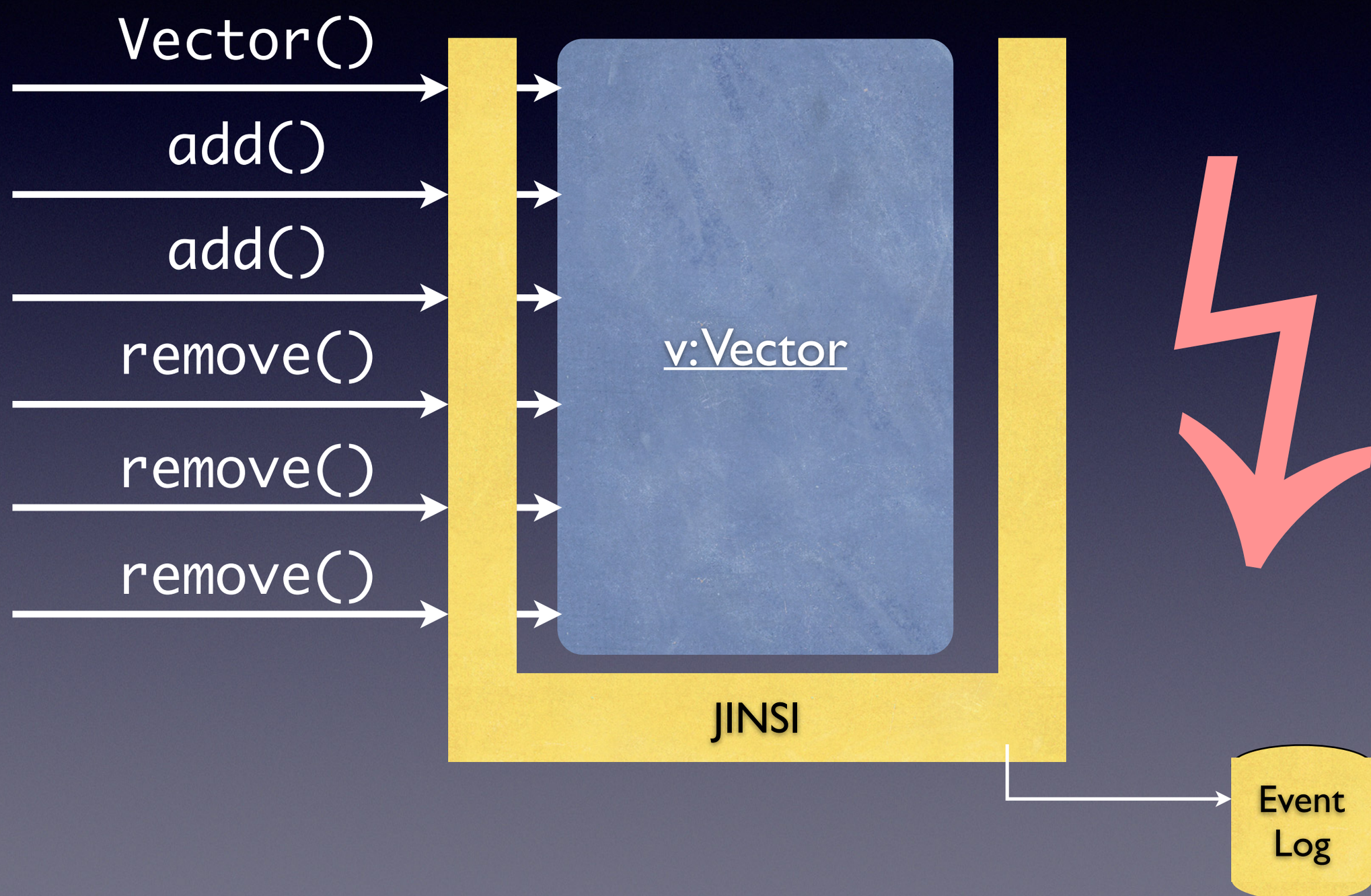
Failure Cause

Lenhof, Hans-Peter,, +49 681 302-64701, lenhof@cs.uni-sb.de
Lindig, Christian, +49 681 9378406, +49 681 302 7790, lindig@cs.uni-sb.de
Mehlmann, Martin,,, mehlmann@st.cs.uni-sb.de
Meyer zu Tittingdorf, Friederike, +49 6894 390052, +49 681 302-78099, meyer@cs.uni-sb.de
Mileva, Yana,, +49 681 302-64020, mileva@cs.uni-sb.de
Müller-Perich, Elisabeth,, +49 681 302-7070, sekr.techfak@rz.uni-sb.de
Nir-Bleimling, Naomi,, +49 68130264011, naomi@wjpserver.cs.uni-sb.de
Offergeld, Thilo,, +49 681 302-6794, t.offergeld@univw.uni-sb.de
PC, CC 2006,,, cc2006pc@st.cs.uni-sb.de
Paul, Wolfgang, +4968171827, +49 6813022436, wjp@cs.uni-sb.de
Premraj, Rahul, +44 7796973711, +49 681 302-64013, premraj@cs.uni-sb.de
Reindel, Erich, +49 6371 912842, +49 681 302-78091, reindel@cs.uni-sb.de
Schuler, David,, +49 681 302-64026, schuler@st.cs.uni-sb.de
Schuler, Erika,, +49 6813027069, schuler@tf.uni-sb.de
Schäfer, Christa, +49 6897 71167, +49 68130264011,
Security, AG,,, security@st.cs.uni-sb.de
Seidel, Raimund, +49 6894 383698, +49 681 302-4713, rseidel@cs.uni-sb.de
Sekretariat, Sekretariat,, +49 681 302-64011, office@st.cs.uni-sb.de
Sliwerski, Jacek, +491741333208,, sliwers@st.cs.uni-sb.de
Slusallek, Philipp, +49 6826 1 88 71 32, +49 681 302-3830, slusallek@cs.uni-sb.de
Slusallek USA, Philipp, +1 670 391 9186, +1 408 486 2788, slusallek@cs.uni-sb.de
Smolka, Gert, +49 681 782770, +49 681 302-7311, smolka@ps.uni-sb.de
Software-Evolution, AG,,, softevo@st.cs.uni-sb.de
Thiel, Frank,,, hausmeister@cs.uni-sb.de
Weiß, Cathrin,,, weiss@st.cs.uni-sb.de
Wilhelm, Reinhard,, +49 681 302-4399, wilhelm@cs.uni-sb.de
Zeller, Andreas,,, zeller@cs.uni-sb.de
Zeller, Andreas, +49 681 3710467, +49 681 302-64011, zeller@cs.uni-sb.de
Zimmermann, Tom, +49 871 71742 (Eltern), +1 403 210 9470, zimmerth@cs.uni-sb.de

Problem:
Simulating user interaction is cumbersome.

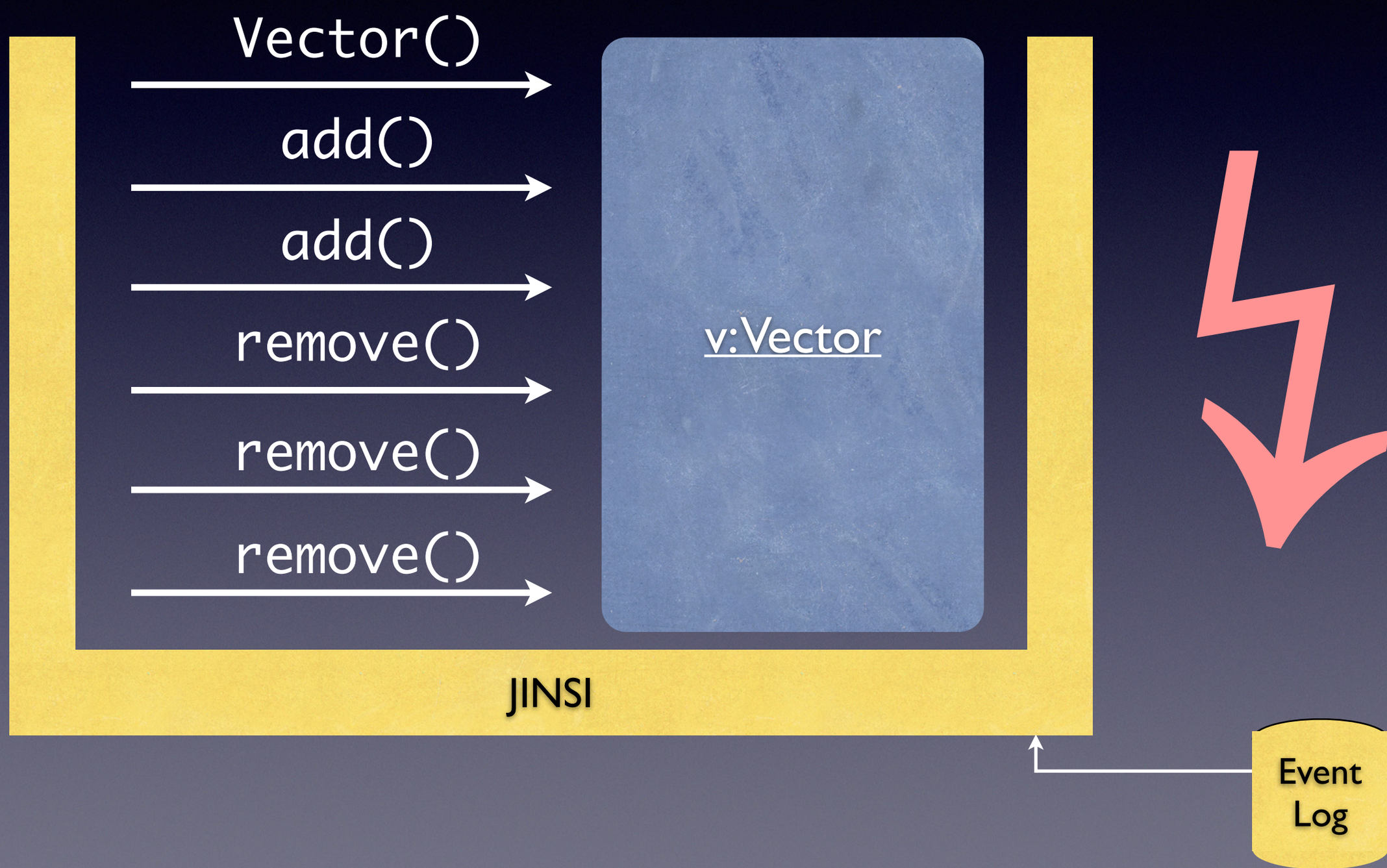
Isolating Relevant Calls

Step 1: Record



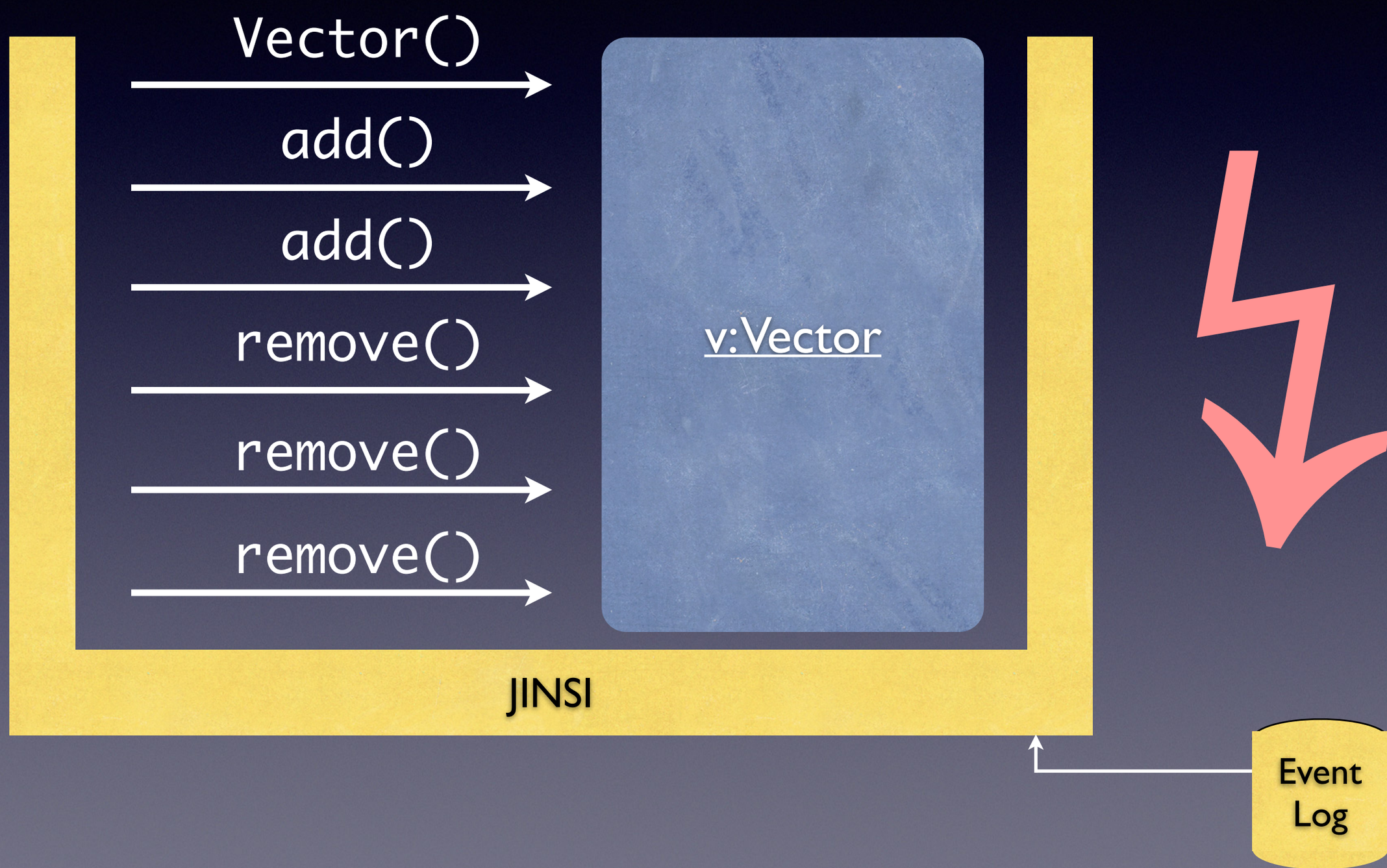
Isolating Relevant Calls

Step 2: Replay



Isolating Relevant Calls

Step 3: Simplify



Isolating Relevant Calls

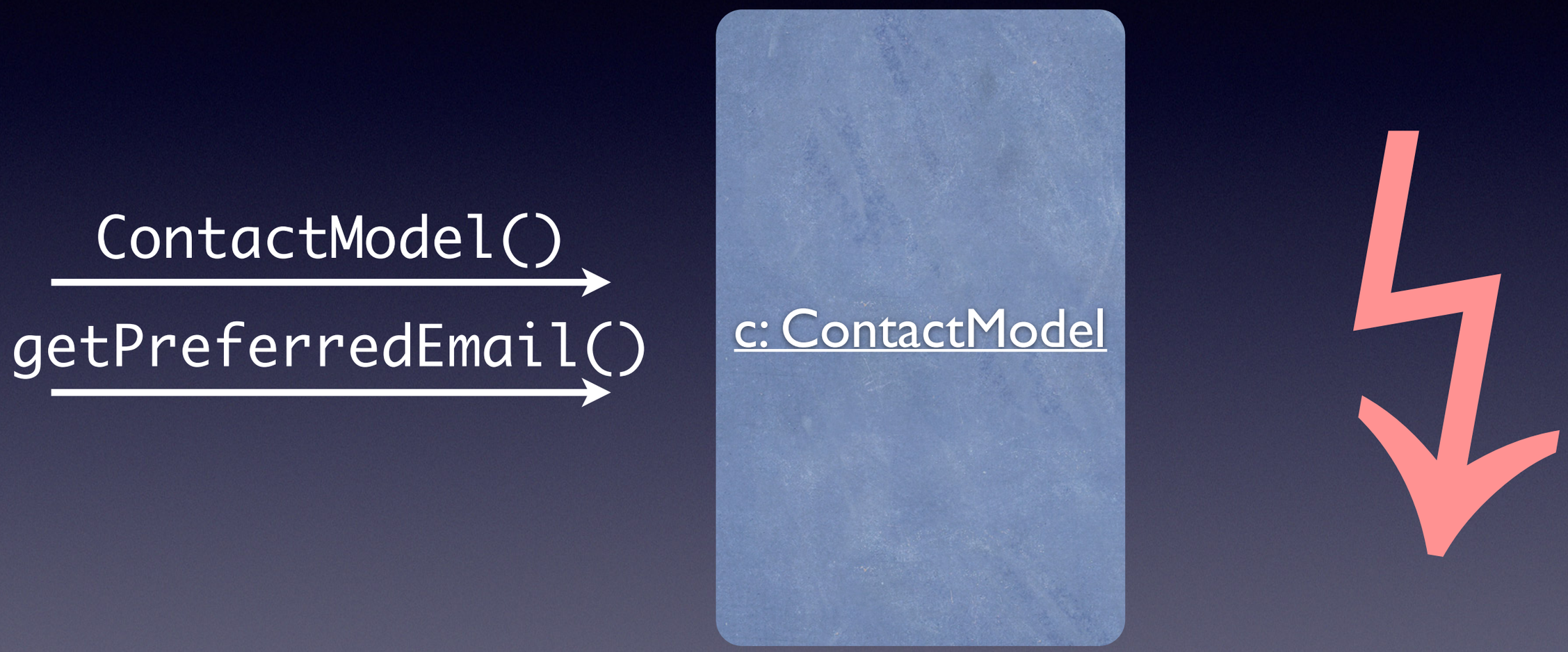
Step 4: Create Unit Test

```
testVector()
{
    Vector v = new Vector();
    v.remove(obj);
}
```

JINSI

Columba ContactModel

Columba ContactModel



Unit Test

```
testContactModel()
{
    ContactModel c = new ContactModel();
    String s = c.getPreferredEmail();
}
```

getPreferredEmail

```
public String getPreferredEmail() {
   Iterator it = getEmailIterator();

   // get first item
   IEmailModel model = (IEmailModel) it.next();

   // backwards compatiblity
   // -> its not possible anymore to create a
   // contact model without email address
   if (model == null)
      return null;

   return model.getAddress();
}
```