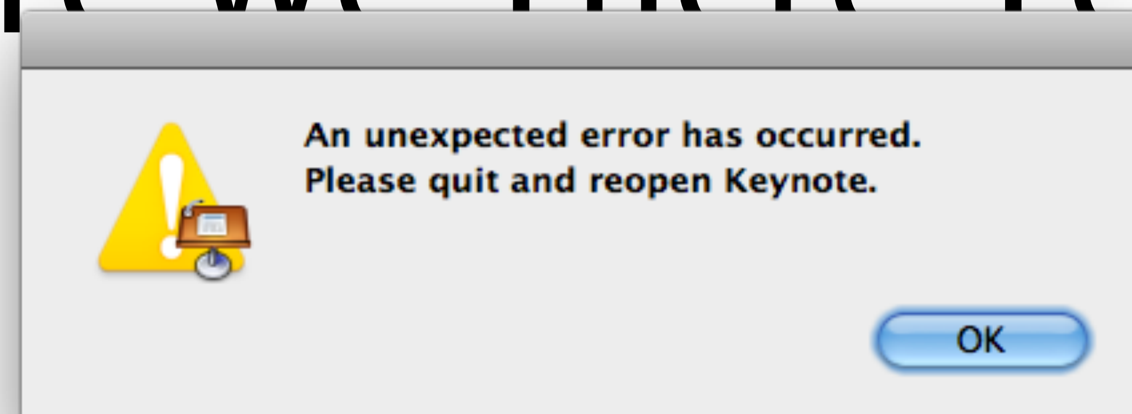
Automated Debugging: Are We There Yet?

Alessandro (Alex) Orso
School of Computer Science – College of Computing
Georgia Institute of Technology
http://www.cc.gatech.edu/~orso/

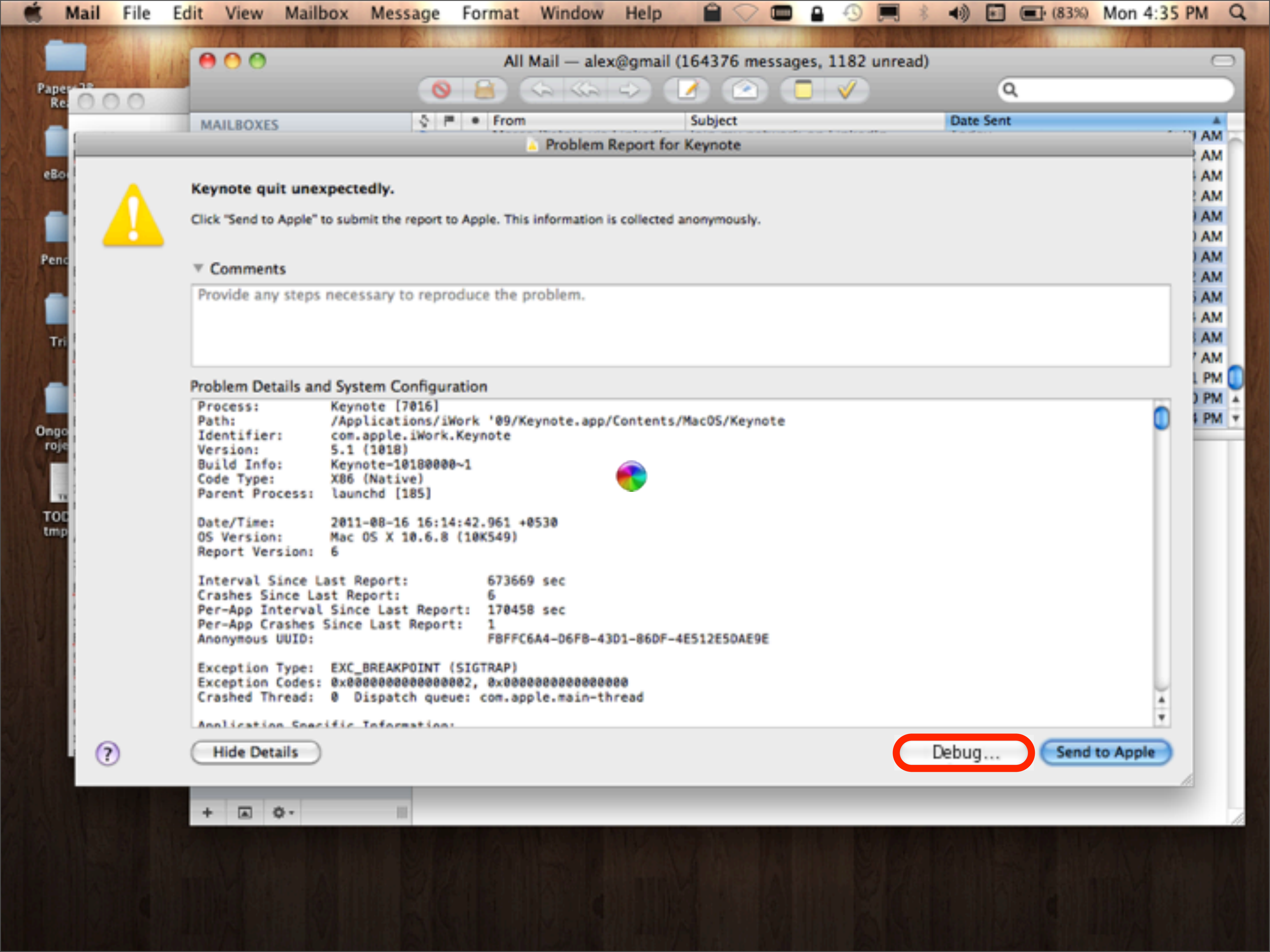
Partially supported by: NSF, IBM, and MSR

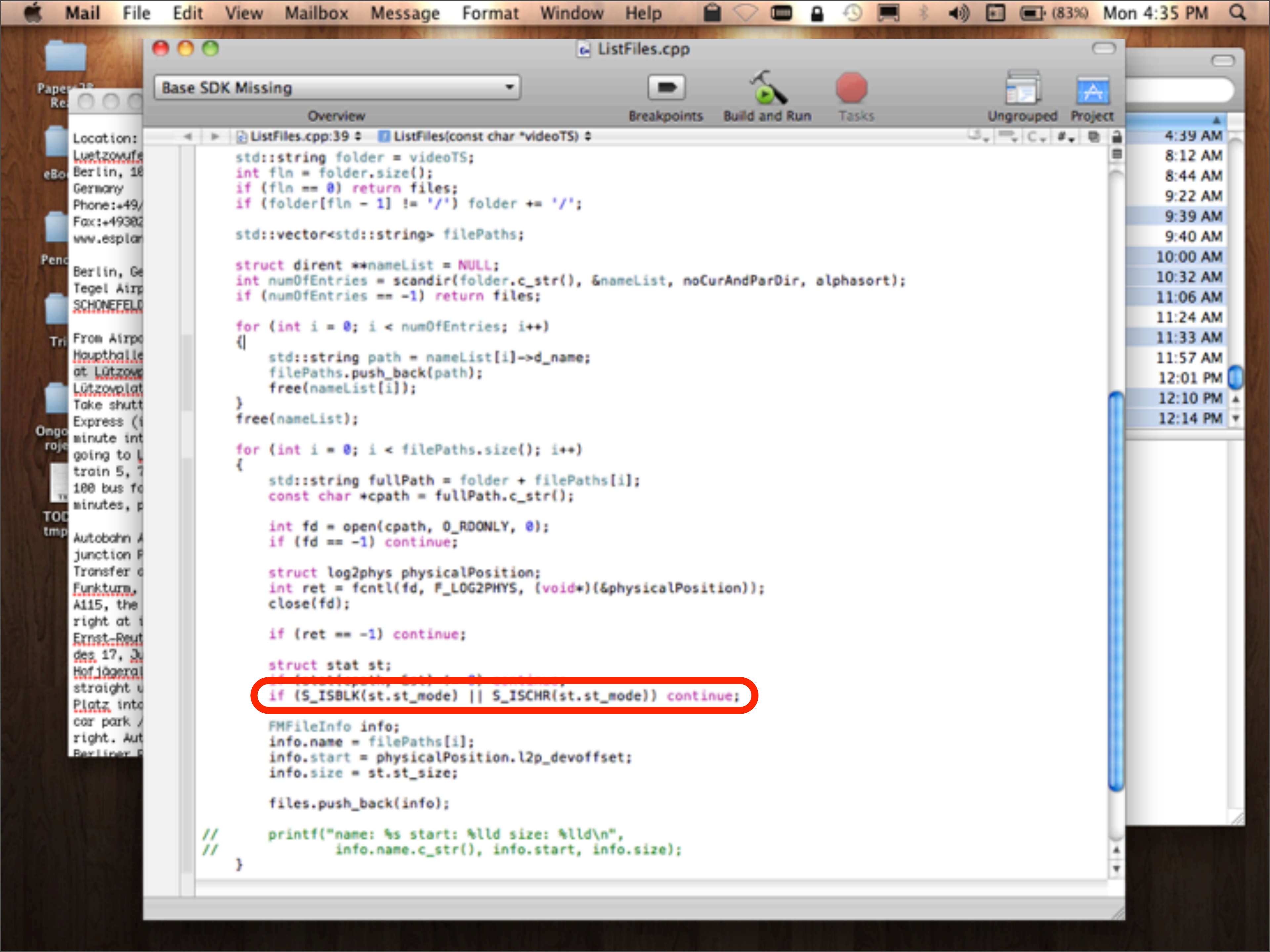
Automated Debugging: Are We There Yet?



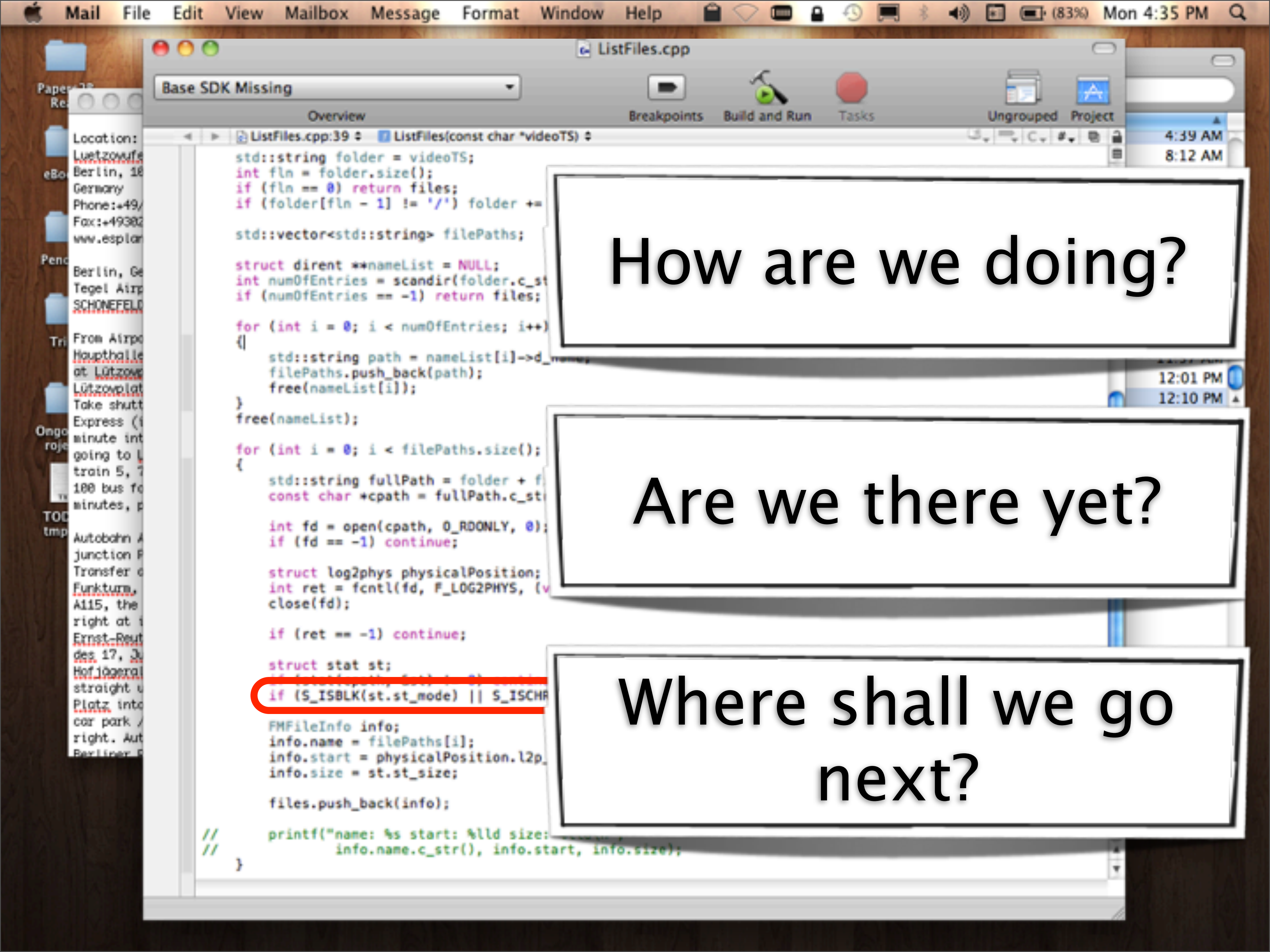
Alessandro (Alex) Orso
School of Computer Science – College of Computing
Georgia Institute of Technology
http://www.cc.gatech.edu/~orso/

Partially supported by: NSF, IBM, and MSR


Mail File Edit View Mailbox Message Format Window Help
(83%) Mon 4:35 PM
All Mail — alex@gmail (164376 messages, 1182 unread)
MAILBOXES
From
Subject
Date Sent
Problem Report for Keynote
Keynote quit unexpectedly.
Click "Send to Apple" to submit the report to Apple. This information is collected anonymously.
Comments
Provide any steps necessary to reproduce the problem.
Problem Details and System Configuration
Process: Keynote [7016]
Path: /Applications/iWork '09/Keynote.app/Contents/MacOS/Keynote
Identifier: com.apple.iWork.Keynote
Version: 5.1 (1018)
Build Info: Keynote-10180000~1
Code Type: X86 (Native)
Parent Process: launchd [185]
Date/Time: 2011-08-16 16:14:42.961 +0530
OS Version: Mac OS X 10.6.8 (10K549)
Report Version: 6
Interval Since Last Report: 673669 sec
Crashes Since Last Report: 6
Per-App Interval Since Last Report: 170458 sec
Per-App Crashes Since Last Report: 1
Anonymous UUID: FBFFC6A4-D6FB-43D1-86DF-4E512E5DAE9E
Exception Type: EXC_BREAKPOINT (SIGTRAP)
Exception Codes: 0x0000000000000002, 0x0000000000000000
Crashed Thread: 0 Dispatch queue: com.apple.main-thread
Hide Details
Debug...
Send to Apple



ListFiles.cpp
Base SDK Missing
Overview
Breakpoints
Build and Run
Tasks
Ungrouped
Project
ListFiles.cpp:39
ListFiles(const char *videoTS)
std::string folder = videoTS;
int fln = folder.size();
if (fln == 0) return files;
if (folder[fln - 1] != '/') folder += '/';
std::vector<std::string> filePaths;
struct dirent **nameList = NULL;
int numOfEntries = scandir(folder.c_str(), &nameList, noCurAndParDir, alphasort);
if (numOfEntries == -1) return files;
for (int i = 0; i < numOfEntries; i++)
{
std::string path = nameList[i]->d_name;
filePaths.push_back(path);
free(nameList[i]);
}
free(nameList);
for (int i = 0; i < filePaths.size(); i++)
{
std::string fullPath = folder + filePaths[i];
const char *cpath = fullPath.c_str();
int fd = open(cpath, O_RDONLY, 0);
if (fd == -1) continue;
struct log2phys physicalPosition;
int ret = fcntl(fd, F_LOG2PHYS, (void*)(&physicalPosition));
close(fd);
if (ret == -1) continue;
struct stat st;
if (S_ISBLK(st.st_mode) || S_ISCHR(st.st_mode)) continue;
FMFileInfo info;
info.name = filePaths[i];
info.start = physicalPosition.l2p_devoffset;
info.size = st.st_size;
files.push_back(info);
//   printf("name: %s start: %lld size: %lld\n",
//   info.name.c_str(), info.start, info.size);
}


How are we doing?
Are we there yet?
Where shall we go next?

How Are We Doing?

A Short History of Debugging

The Birth of Debugging

???

2013

First reference to software errors
Your guess?

The Birth of Debugging

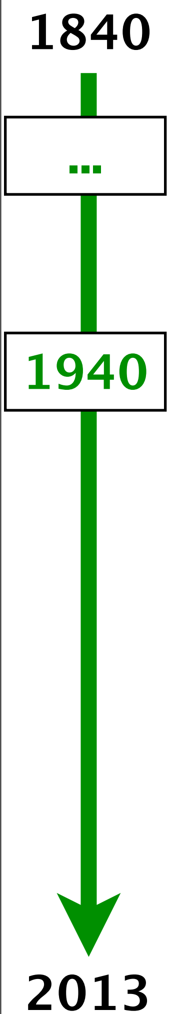
1840

1843

- Software errors mentioned in Ada Byron's notes on Charles Babbage's analytical engine

2013

The Birth of Debugging



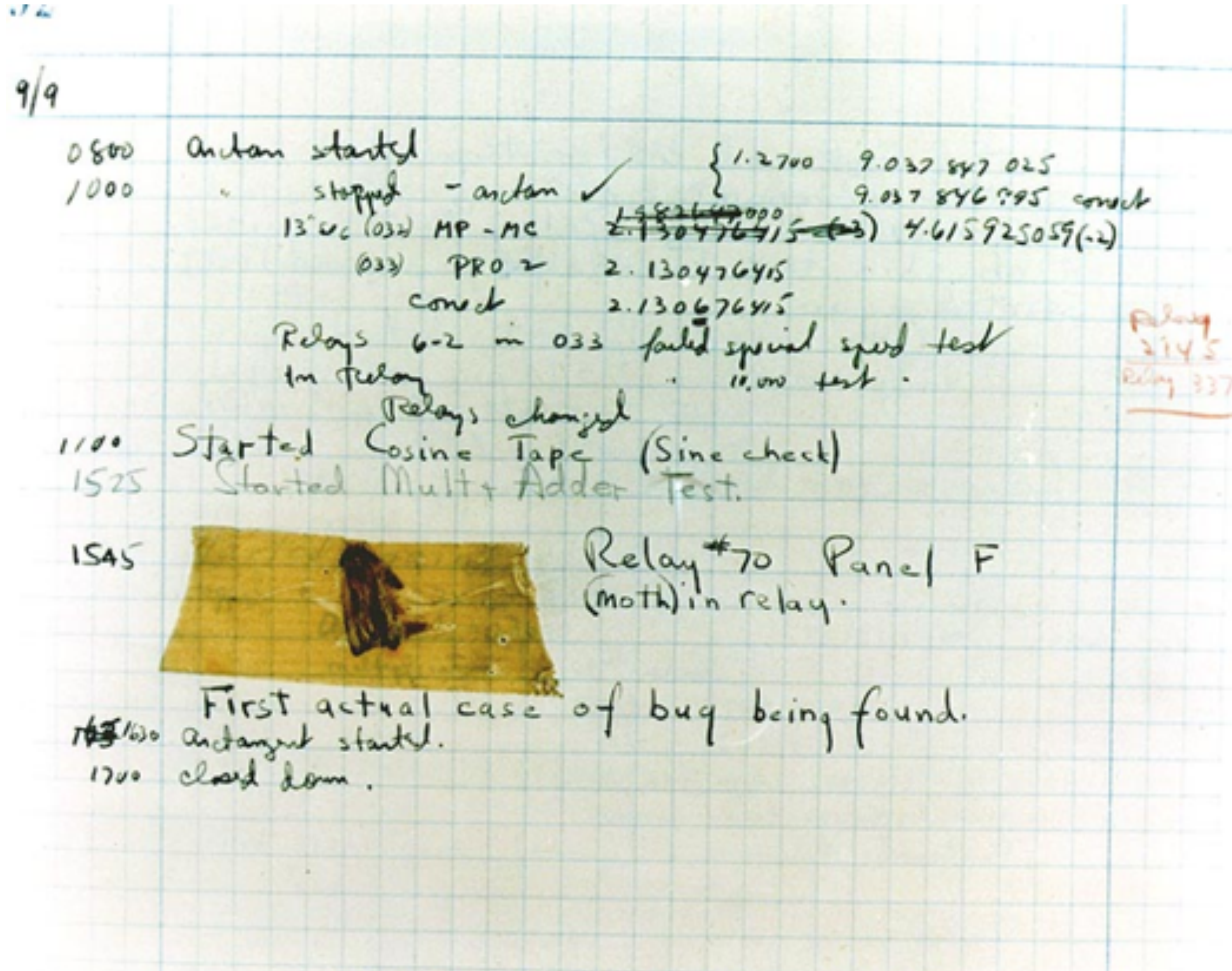
- Software errors mentioned in Ada Byron's notes on Charles Babbage's analytical engine
- Several uses of the term bug to indicate defects in computers and software

The Birth of Debugging

1840

1947

2013



Symbolic Debugging

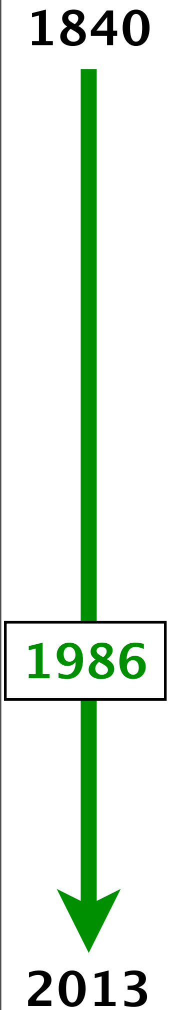
1840

1962

2013

- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)

Symbolic Debugging



- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)
- GDB

Symbolic Debugging

1840

- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)
- GDB
- DDD

1996

2013

Symbolic Debugging

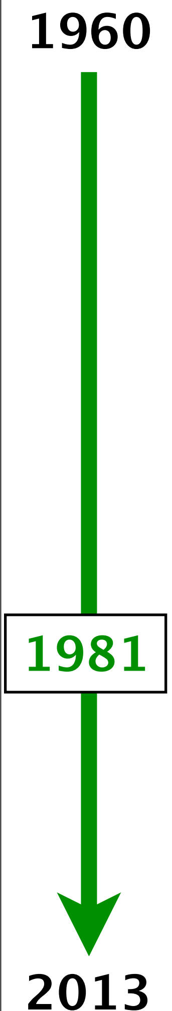
1840

1996

2013

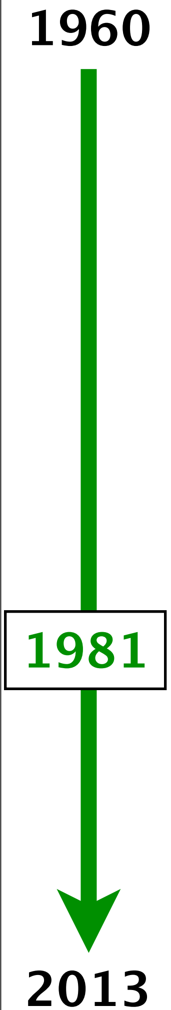
- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)
- GDB
- DDD
- ...

Program Slicing



- **Intuition**: developers “slice” backwards when debugging

Program Slicing

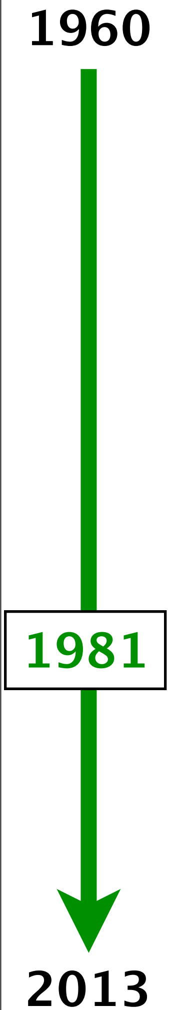


- **Intuition**: developers "slice" backwards when debugging
- Weiser's breakthrough paper

Static Slicing Example

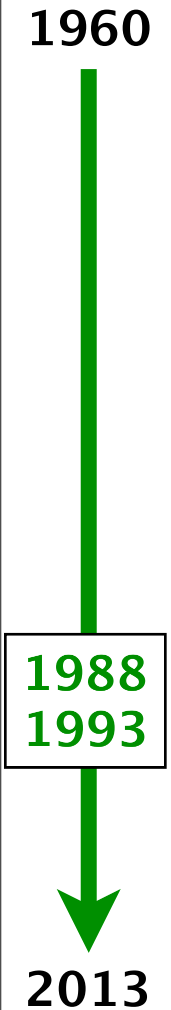
```
mid() {
 int x,y,z,m;
 read("Enter 3 numbers:",x,y,z);
 m = z;
 if (y<z)
    if (x<y)
          m = y;
    else if (x<z)
          m = y; // bug
 else
    if (x>y)
          m = y;
    else if (x>z)
          m = x;
 print("Middle number is:", m);
}
```

Program Slicing



- **Intuition**: developers "slice" backwards when debugging
- Weiser's breakthrough paper

Program Slicing



- **Intuition**: developers “slice” backwards when debugging
- Weiser’s breakthrough paper
- Korel and Laski’s dynamic slicing
- Agrawal

Dynamic Slicing Example

```
mid() {
 int x,y,z,m;
 read("Enter 3 numbers:",x,y,z);
 m = z;
 if (y<z)
    if (x<y)
          m = y;
    else if (x<z)
          m = y; // bug
 else
    if (x>y)
          m = y;
    else if (x>z)
          m = x;
 print("Middle number is:", m);
}
```

Dynamic Slicing Example

Test Cases

		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
	`mid() {`						
	`int x,y,z,m;`						
1:	`read("Enter 3 numbers:",x,y,z);`	•	•	•	•	•	•
2:	`m = z;`	•	•	•	•	•	•
3:	`if (y<z)`	•	•	•	•	•	•
4:	`if (x<y)`	•	•			•	•
5:	`m = y;`		•				
6:	`else if (x<z)`	•				•	•
7:	`m = y; // bug`	•					•
8:	`else`			•	•		
9:	`if (x>y)`			•	•		
10:	`m = y;`			•			
11:	`else if (x>z)`				•		
12:	`m = x;`						
13:	`print("Middle number is:", m);`	•	•	•	•	•	•
	`}`						
	Pass/Fail	P	P	P	P	P	F

Dynamic Slicing Example

	Test Cases					
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
`mid() {`						
`int x,y,z,m;`						
`1: read("Enter 3 numbers:",x,y,z);`	•	•	•	•	•	•
`2: m = z;`	•	•	•	•	•	•
`3: if (y<z)`	•	•	•	•	•	•
`4: if (x<y)`	•	•			•	•
`5: m = y;`		•				
`6: else if (x<z)`	•				•	•
`7: m = y; // bug`	•					•
`8: else`			•	•		
`9: if (x>y)`			•	•		
`10: m = y;`			•			
`11: else if (x>z)`				•		
`12: m = x;`						
`13: print("Middle number is:", m);`	•	•	•	•	•	•
`}`						
Pass/Fail	P	P	P	P	P	F

Program Slicing

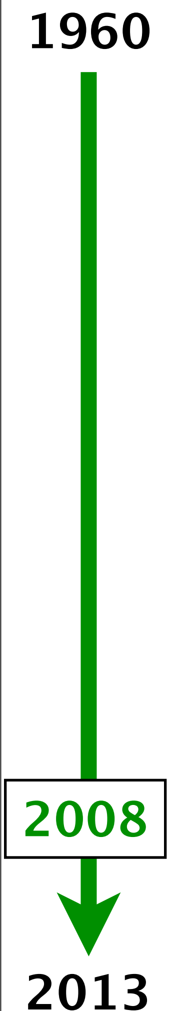
1960

1988
1993

2013

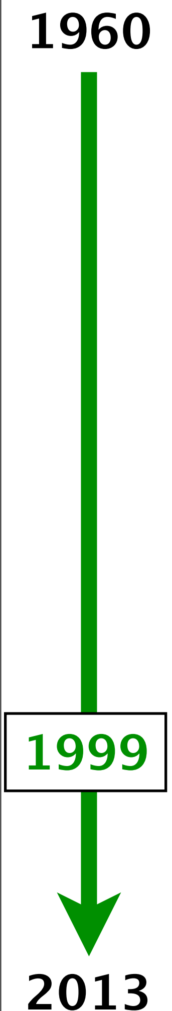
- **Intuition**: developers "slice" backwards when debugging
- Weiser's breakthrough paper
- Korel and Laski's dynamic slicing
- Agrawal

Program Slicing



- **Intuition**: developers “slice” backwards when debugging
- Weiser’s breakthrough paper
- Korel and Laski’s dynamic slicing
- Agrawal
- Ko’s Whyline

Delta Debugging



- **Intuition**: it’s all about differences!

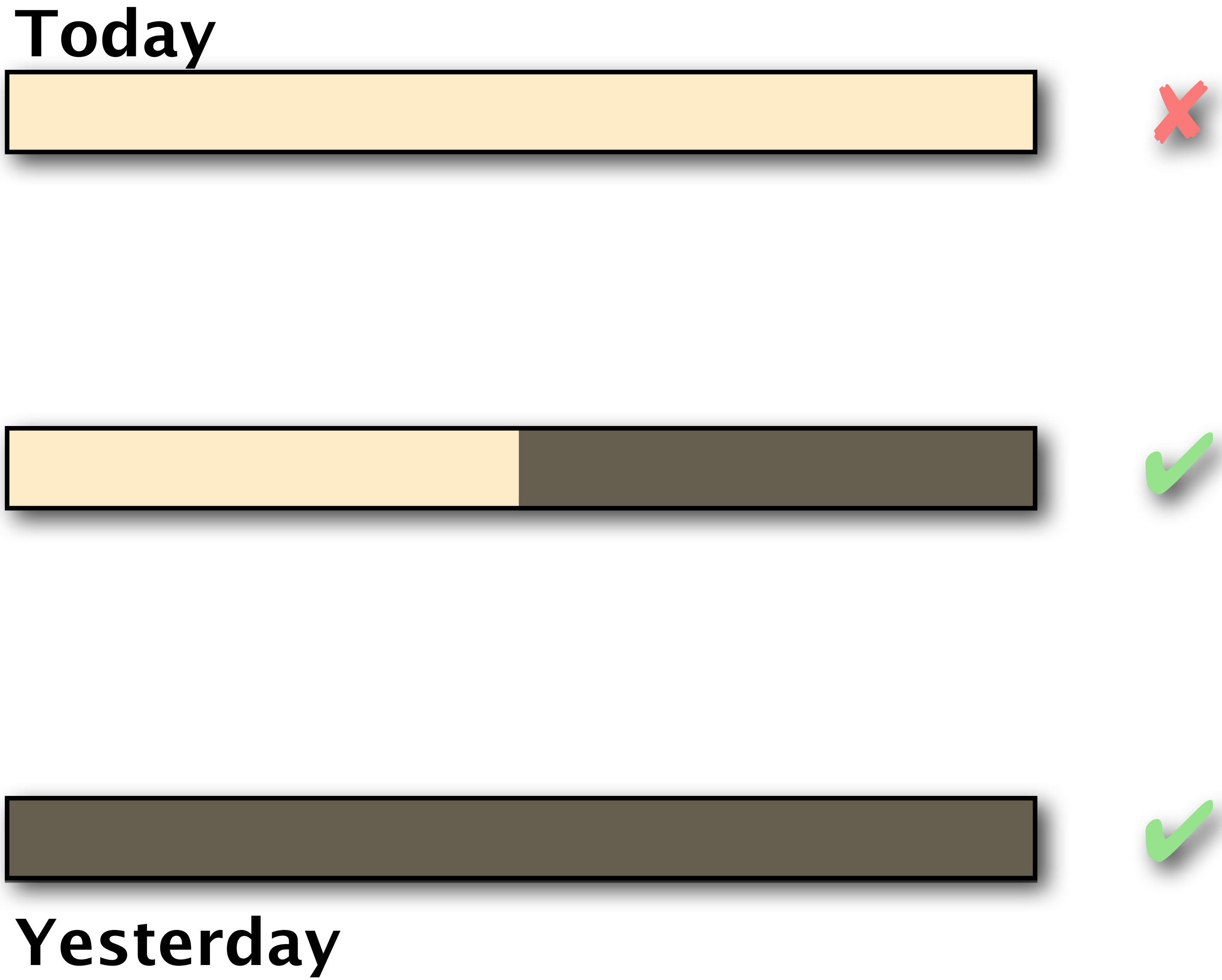
Delta Debugging

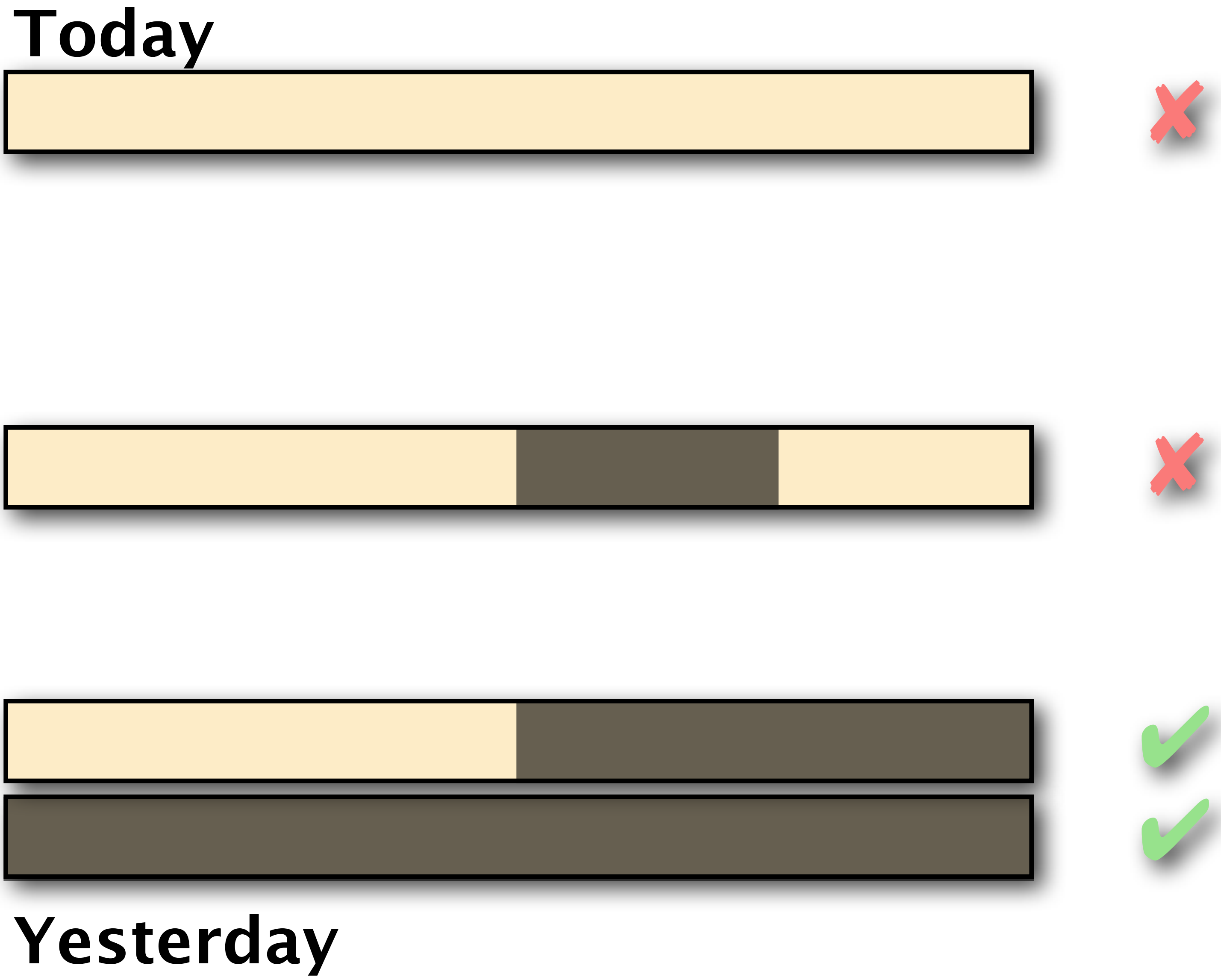
1960

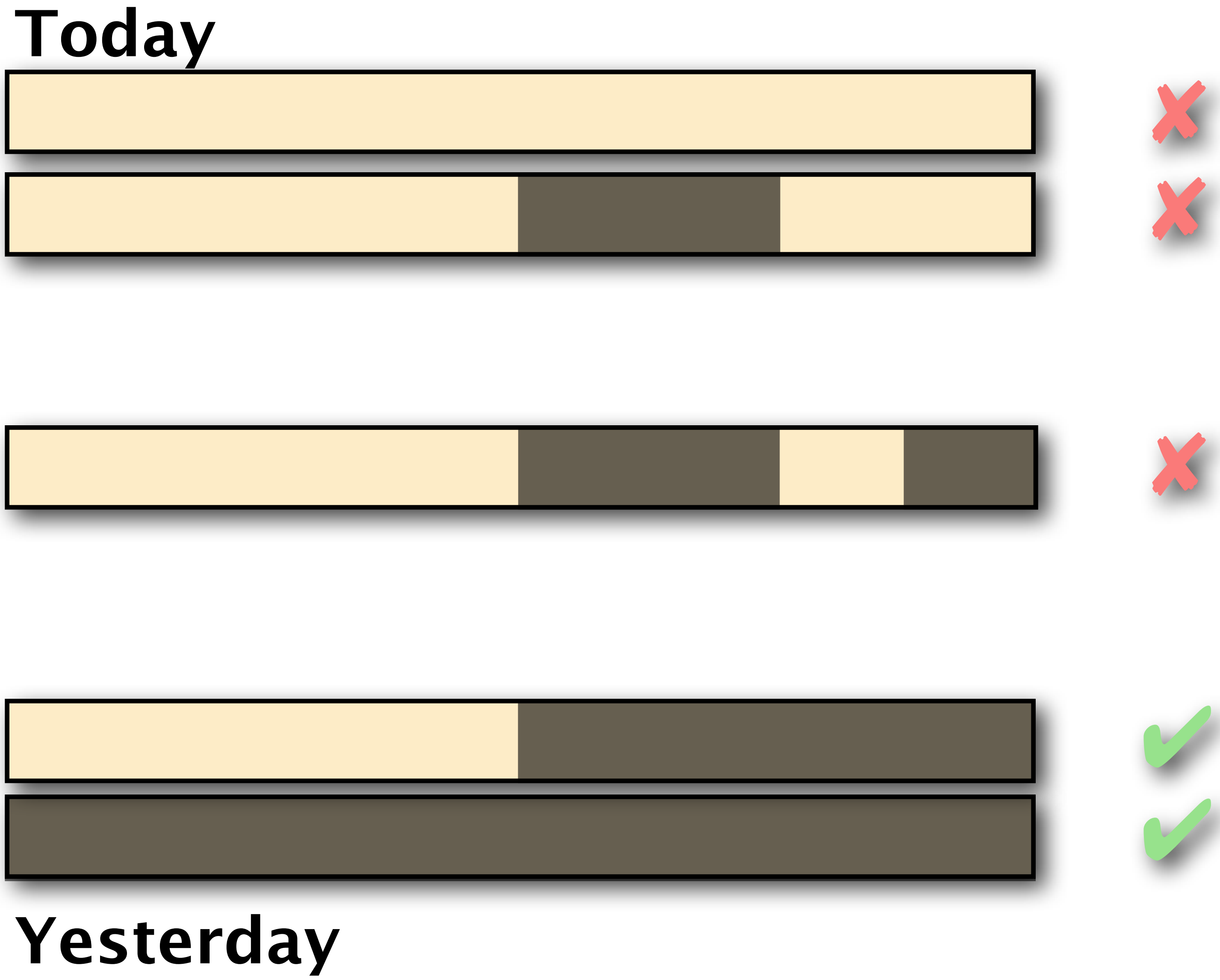
- **Intuition**: it's all about differences!
- Isolates failure causes automatically
- Zeller's "Yesterday, My Program Worked. Today, It Does Not. Why?"

1999

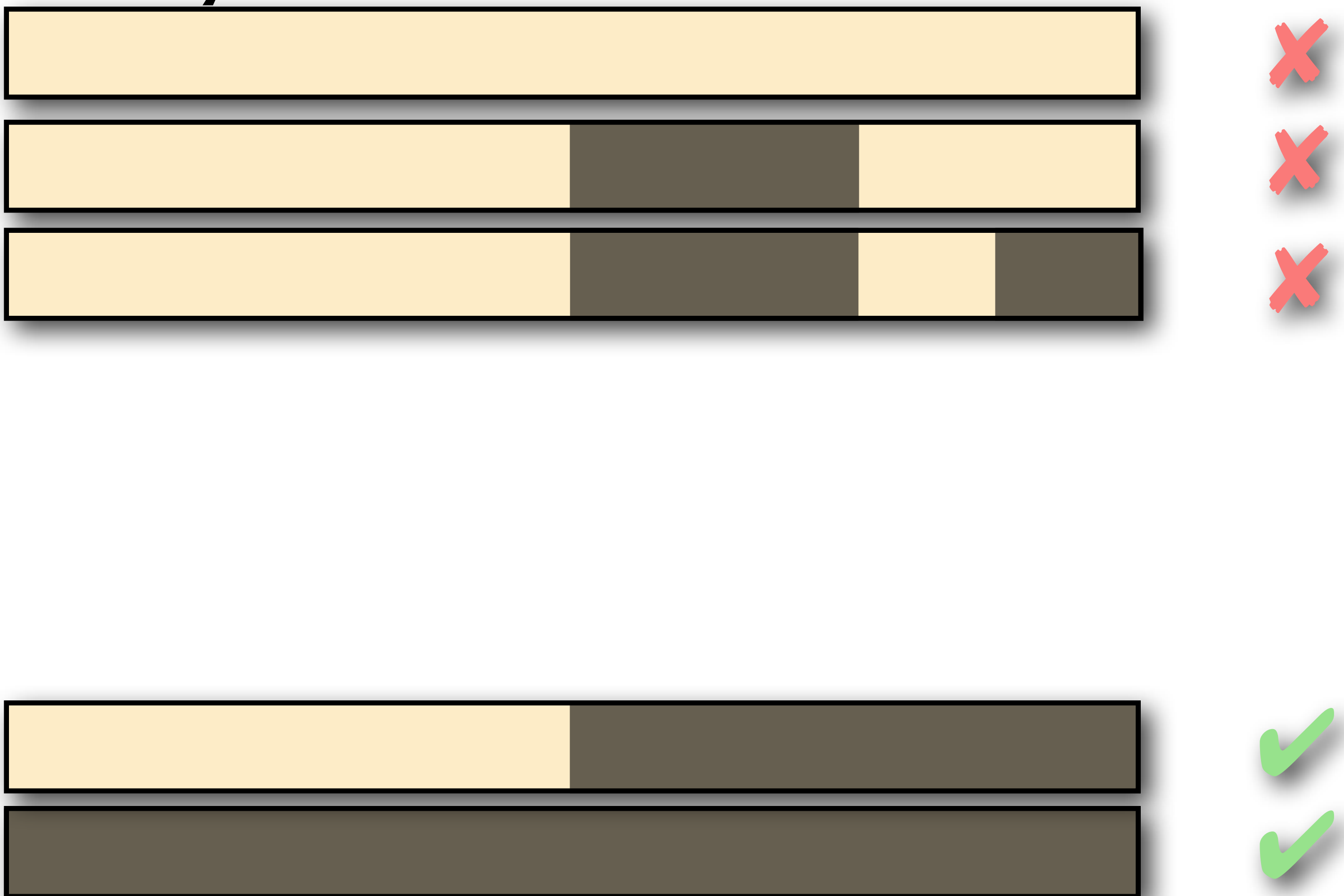
2013


Today
Yesterday

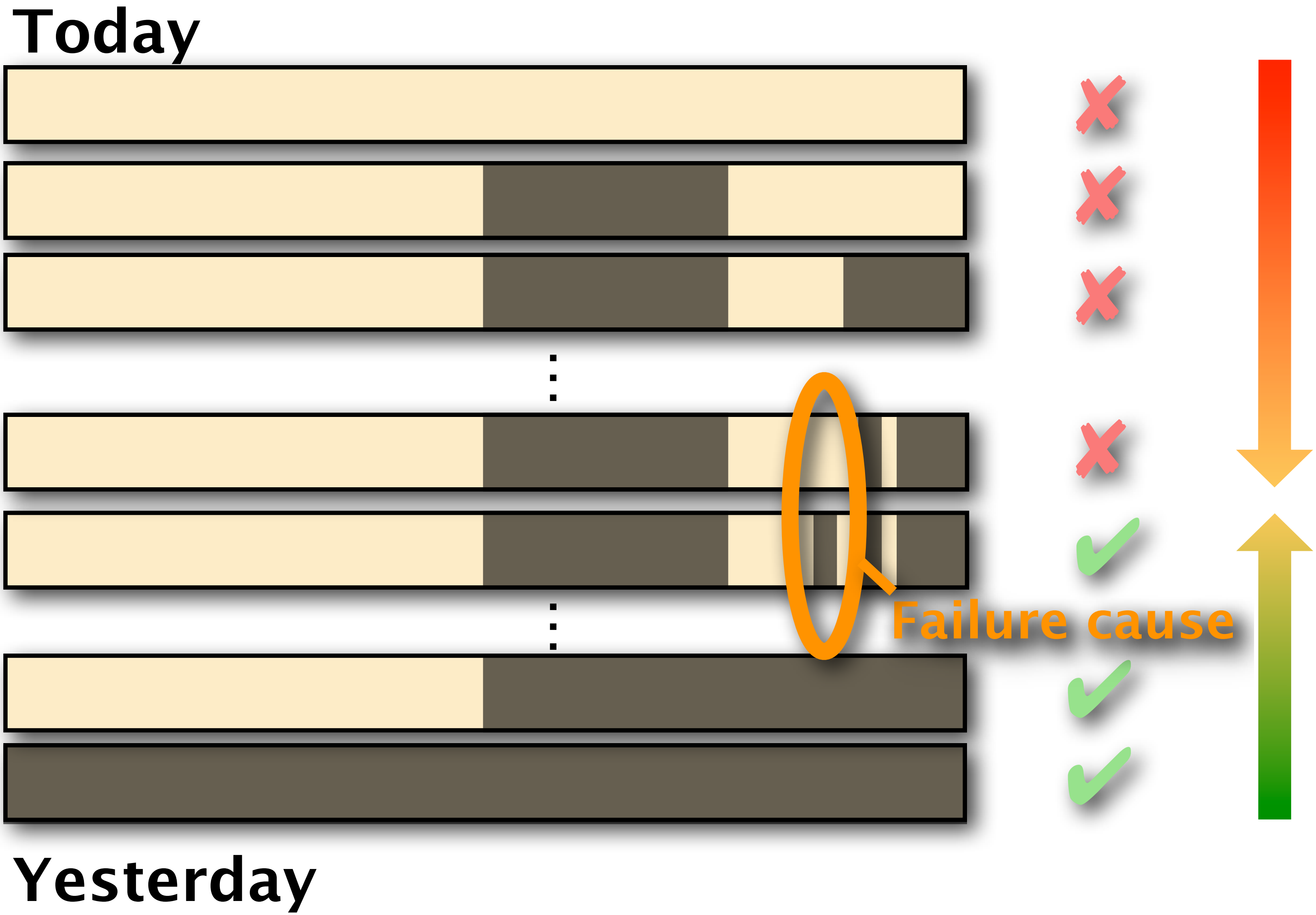

Today
Yesterday

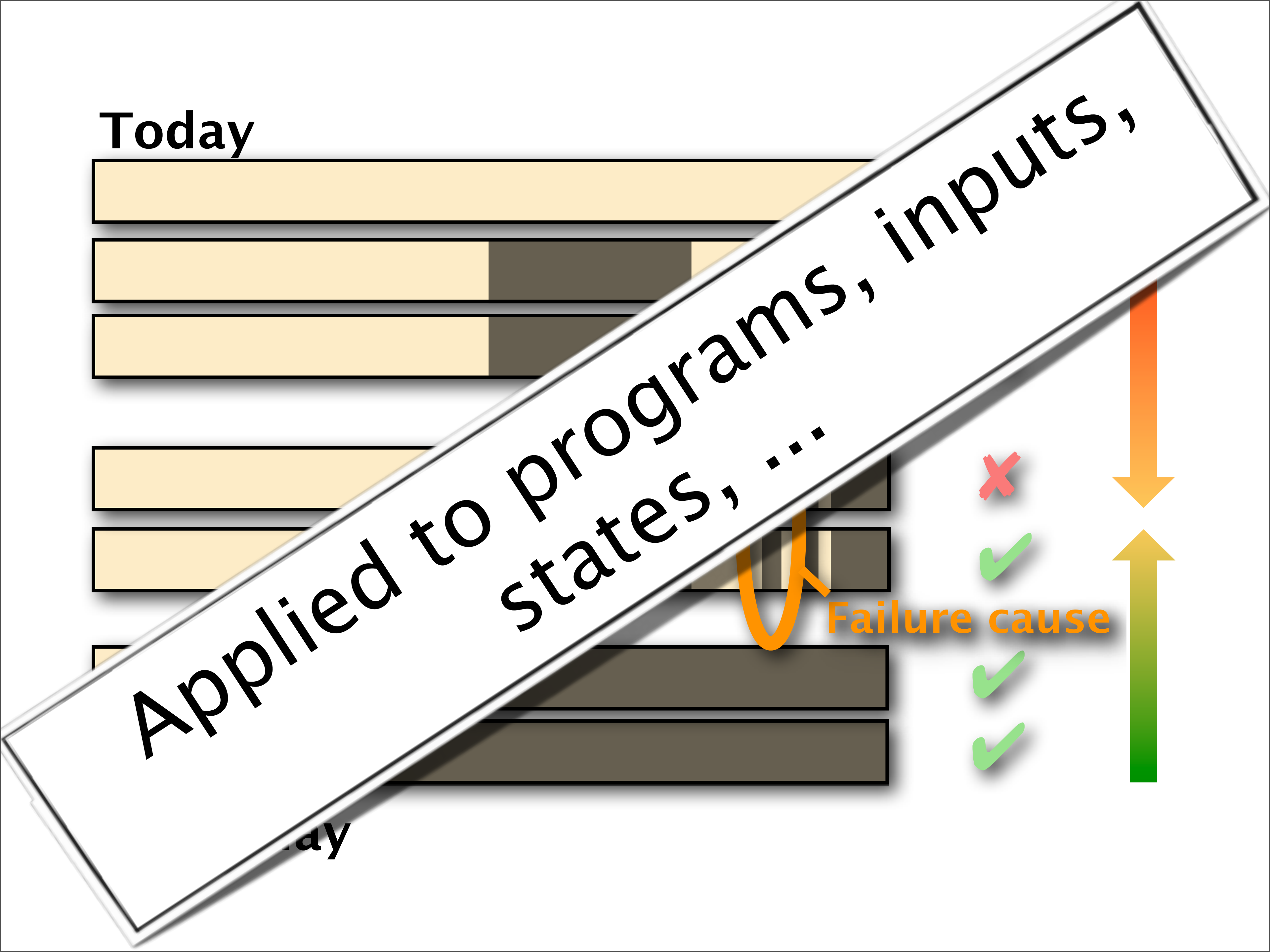

Today
Yesterday

Today

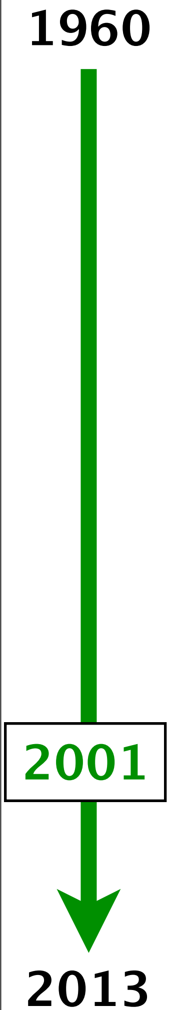


Yesterday


Today
Failure cause
Yesterday

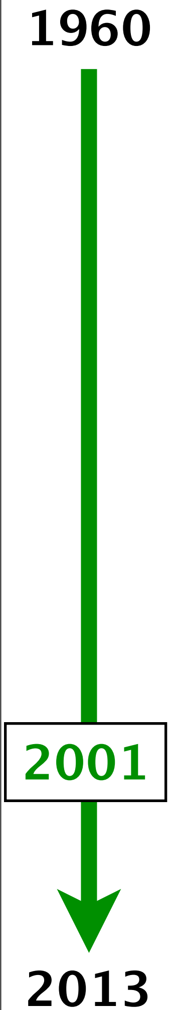

Today
Applied to programs, inputs, states, ...
Failure cause

Statistical Debugging



- **Intuition**: debugging techniques can leverage multiple executions

Statistical Debugging



- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula

Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

Test Cases

```
mid() {
 int x,y,z,m;
```

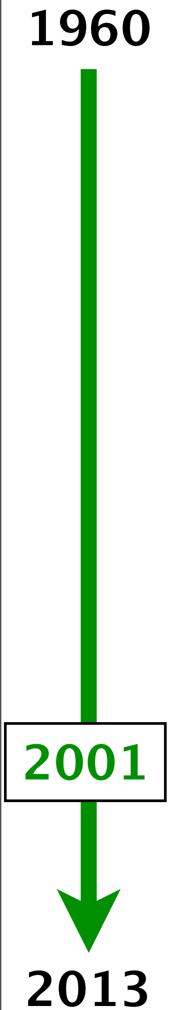
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	suspiciousness
`1: read("Enter 3 numbers:",x,y,z);`	•	•	•	•	•	•	0.5
`2: m = z;`	•	•	•	•	•	•	0.5
`3: if (y<z)`	•	•	•	•			0.5
`4:    if (x<y)`							.6
`5:          m = y;`							.0
`6:    else if (x<z)`							.7
`7:          m = y; // bug`							.8
`8: else`			•	•			0.0
`9:    if (x>y)`			•	•			0.0
`10:         m = y;`							.0
`11:   else if (x>z)`							.0
`12:         m = x;`							.0
`13: print("Middle number is:", m);`							.5
Pass/Fail	P	P	P	P	P	F	

```
}
```

$$susp(1) = \frac{\frac{1}{1}}{\frac{5}{5} + \frac{1}{1}} = 0.5$$

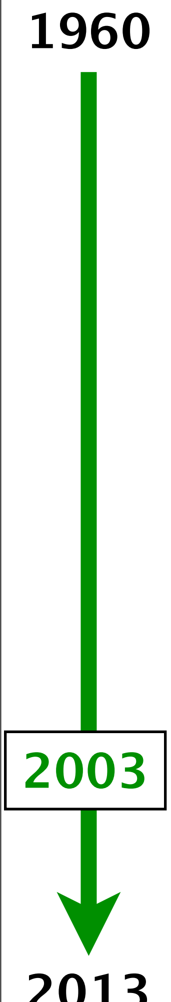
$$susp(7) = \frac{\frac{1}{1}}{\frac{1}{5} + \frac{1}{1}} = 0.8$$

Statistical Debugging



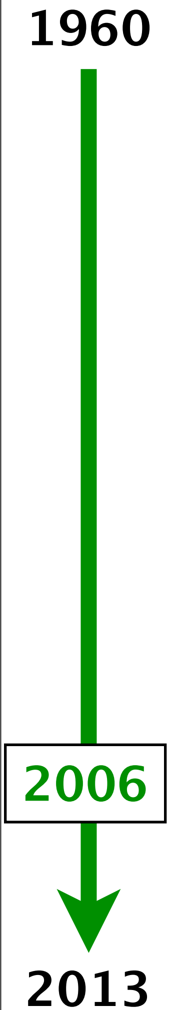
- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula

Statistical Debugging



- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula
- CBI

Statistical Debugging



- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula
- CBI
- Ochiai

Statistical Debugging

1960

- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula
- CBI
- Ochiai
- Causal inference based

2010

2013

Statistical Debugging

1960

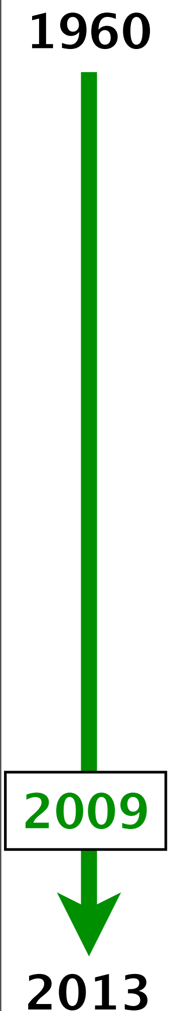
- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula
- CBI
-
- C
- Ma

...

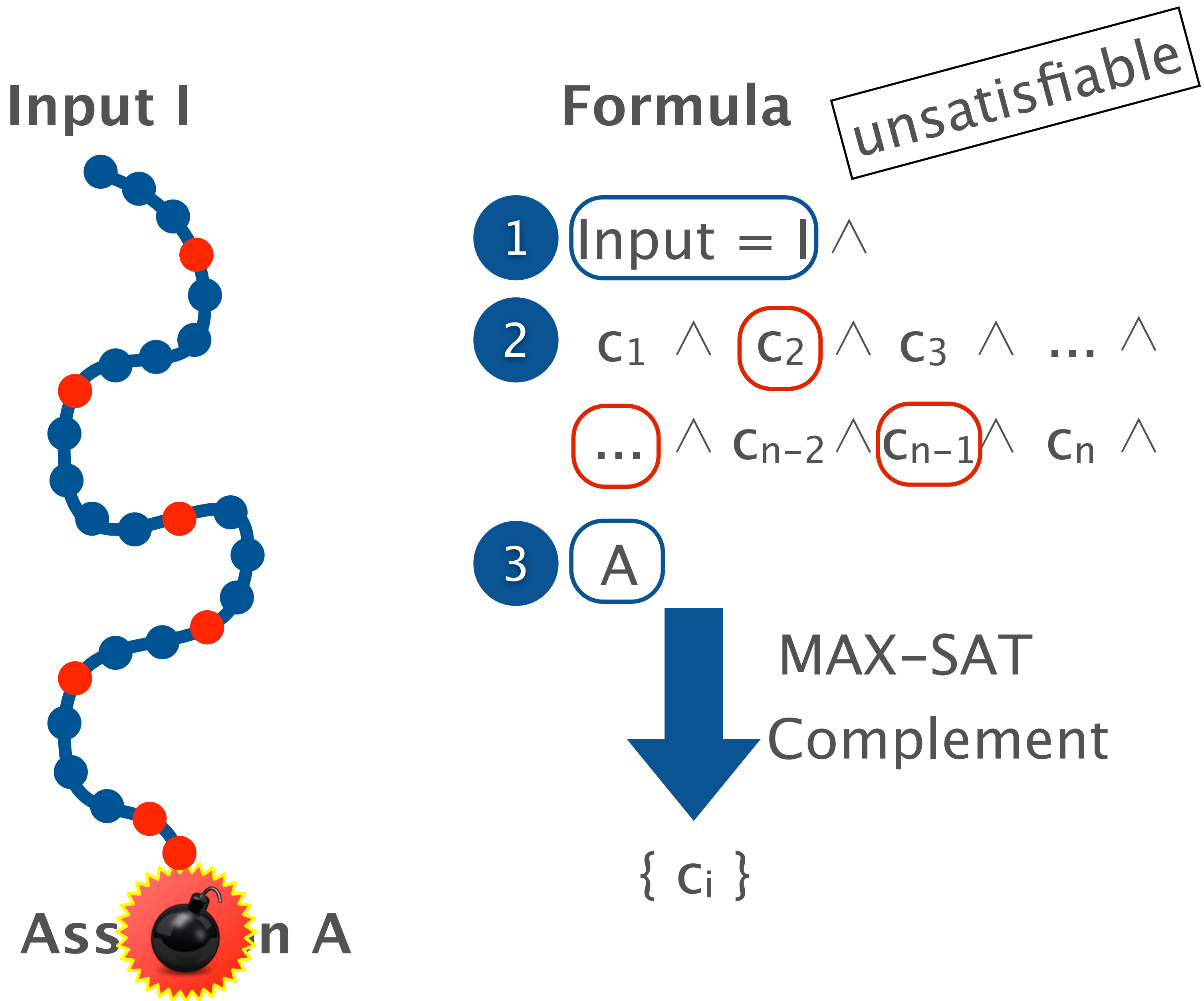
2013

Workflow integration: Tarantula, GZoltar, EzUnit, ...

Formula-based Debugging (AKA Failure Explanation)

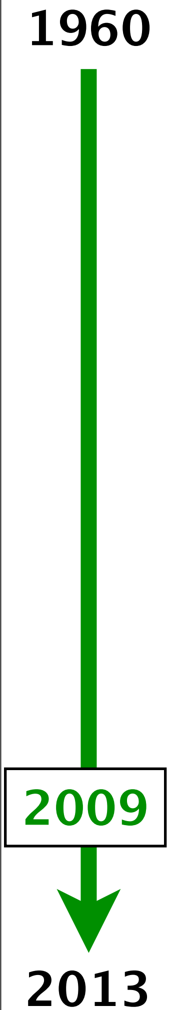


- **Intuition**: executions can be expressed as formulas that we can reason about



Input I
Formula
unsatisfiable
1
Input = I ∧
2
c_1 ∧ c_2 ∧ c_3 ∧ ... ∧
... ∧ c_{n-2} ∧ c_{n-1} ∧ c_n ∧
3
A
MAX-SAT
Complement
{ c_i }
Ass
n A

Formula-based Debugging (AKA Failure Explanation)



- **Intuition**: executions can be expressed as formulas that we can reason about
- Darwin

Formula-based Debugging (AKA Failure Explanation)

1960

- **Intuition**: executions can be expressed as formulas that we can reason about
- Darwin
- Bug Assist

2011

2013

Formula-based Debugging (AKA Failure Explanation)

1960

- **Intuition**: executions can be expressed as formulas that we can reason about
- Darwin
- Bug Assist
- Error invariants

2011

2013

Formula-based Debugging (AKA Failure Explanation)

1960

- **Intuition**: executions can be expressed as formulas that we can reason about
- Darwin
- Bug Assist
- Error invariants
- Angelic debugging

2011

2013

Additional Techniques

1960

- Contracts (e.g., Meyer et al.)
- Counterexample-based (e.g., Groce et al., Ball et al.)
- Tainting-based (e.g., Leek et al.)
- Debugging of field failures (e.g., Jin et al.)
- Predicate switching (e.g., Zhan
- Fault localization f
- Debug
- ard et al.)
- programming
- ixes
- web pages, comments, concurrency)
- Identifying workarounds/recovery strategies (e.g., Gorla et al.)
- Formula based debugging (e.g., Jose et al., Ermis et al.)
- ...

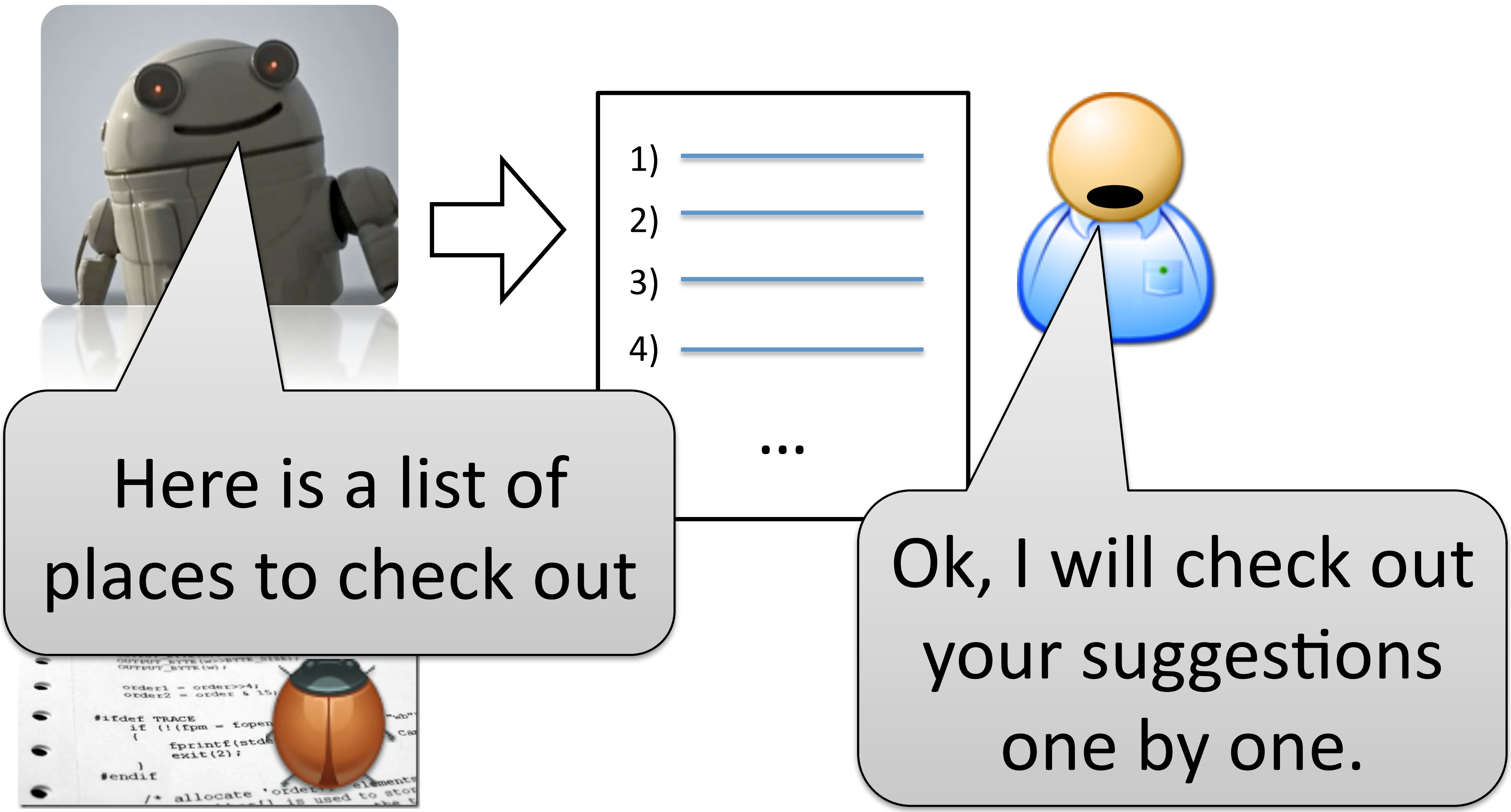
2013

Not meant to be comprehensive!

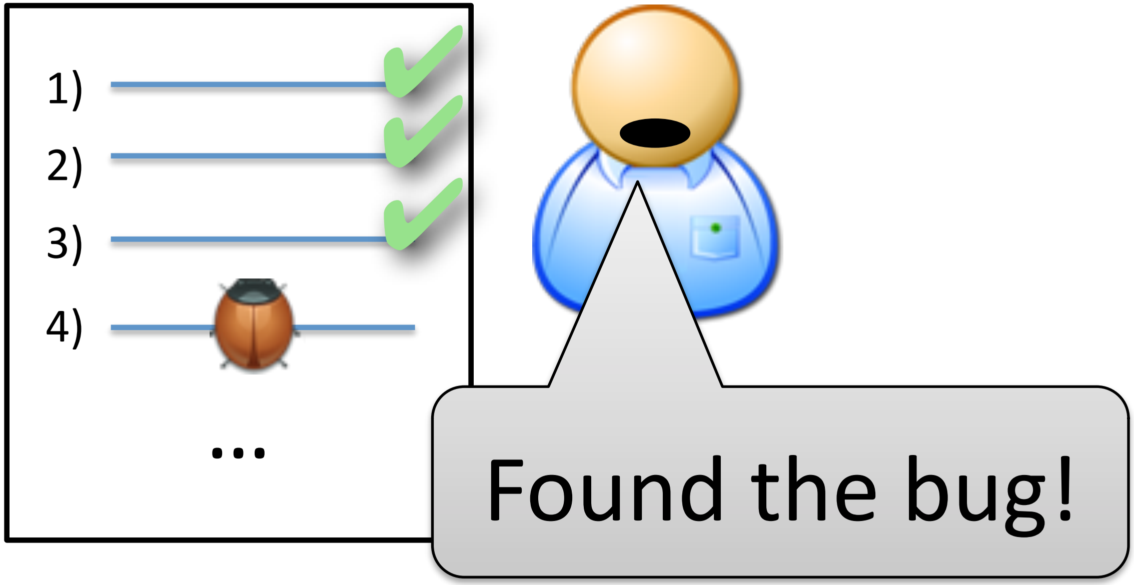
Are We There Yet?

Can We Debug at the Push of a Button?

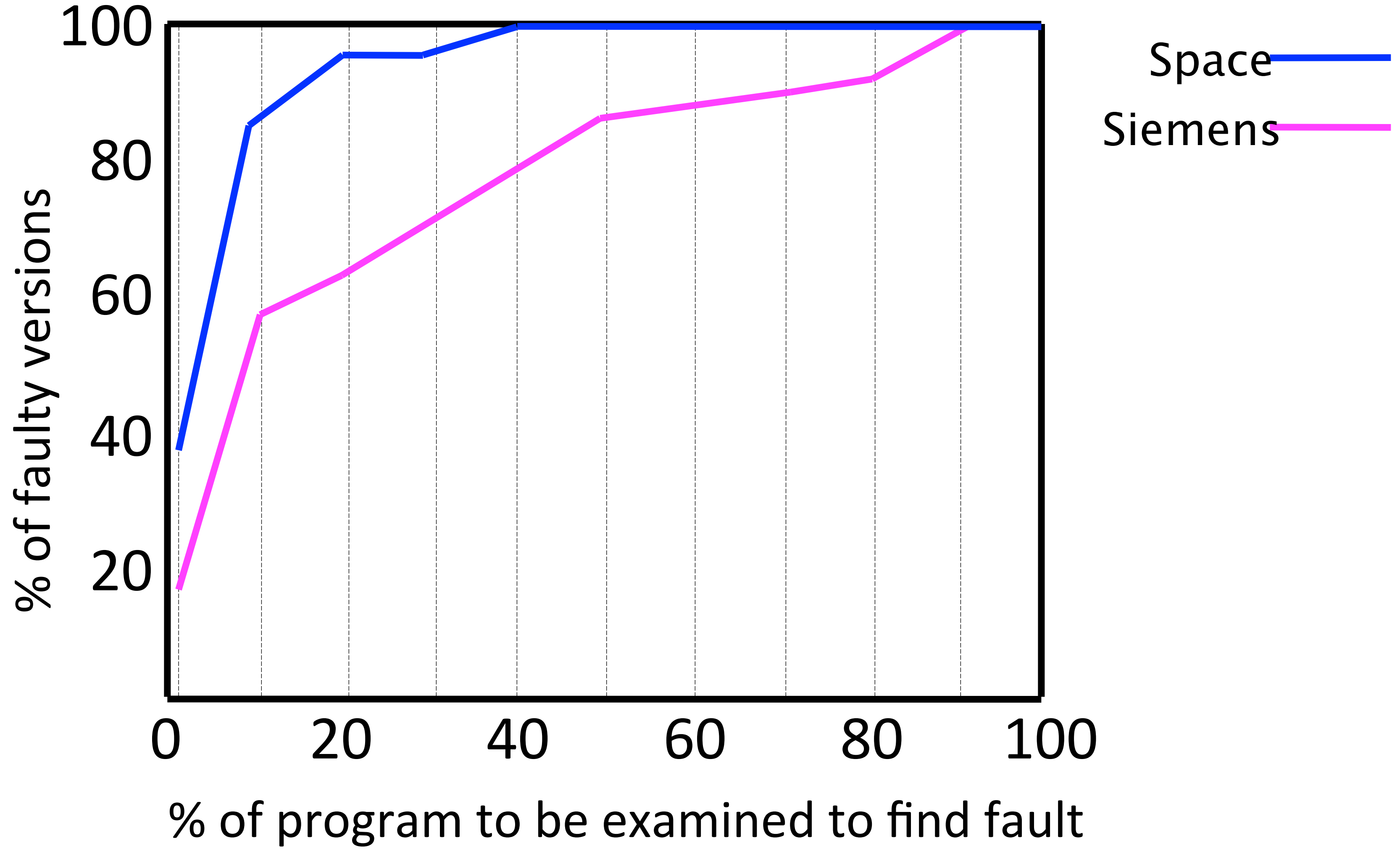
Automated Debugging
(rank based)



Automated Debugging
Conceptual Model



Performance of Automated Debugging Techniques



Mission Accomplished?

Best result: fault in 10% of the code.
Great, but...

100 LOC ➡ 10 LOC

10,000 LOC ➡ 1,000 LOC

100,000 LOC ➡ 10,000 LOC

Mission Accomplished?

Best result: fault in 10% of the code.
Great, but...

100 LOC ➡ 10 LOC

10,000 L

Moreover, strong assumptions

,000 LOC ➡ 10,000 LOC

Assumption #1: Programmers exhibit **perfect bug understanding**

Do you see a bug?

Assumption #2: Programmers inspect a list **linearly and exhaustively**

Good for comparison, but is it realistic?

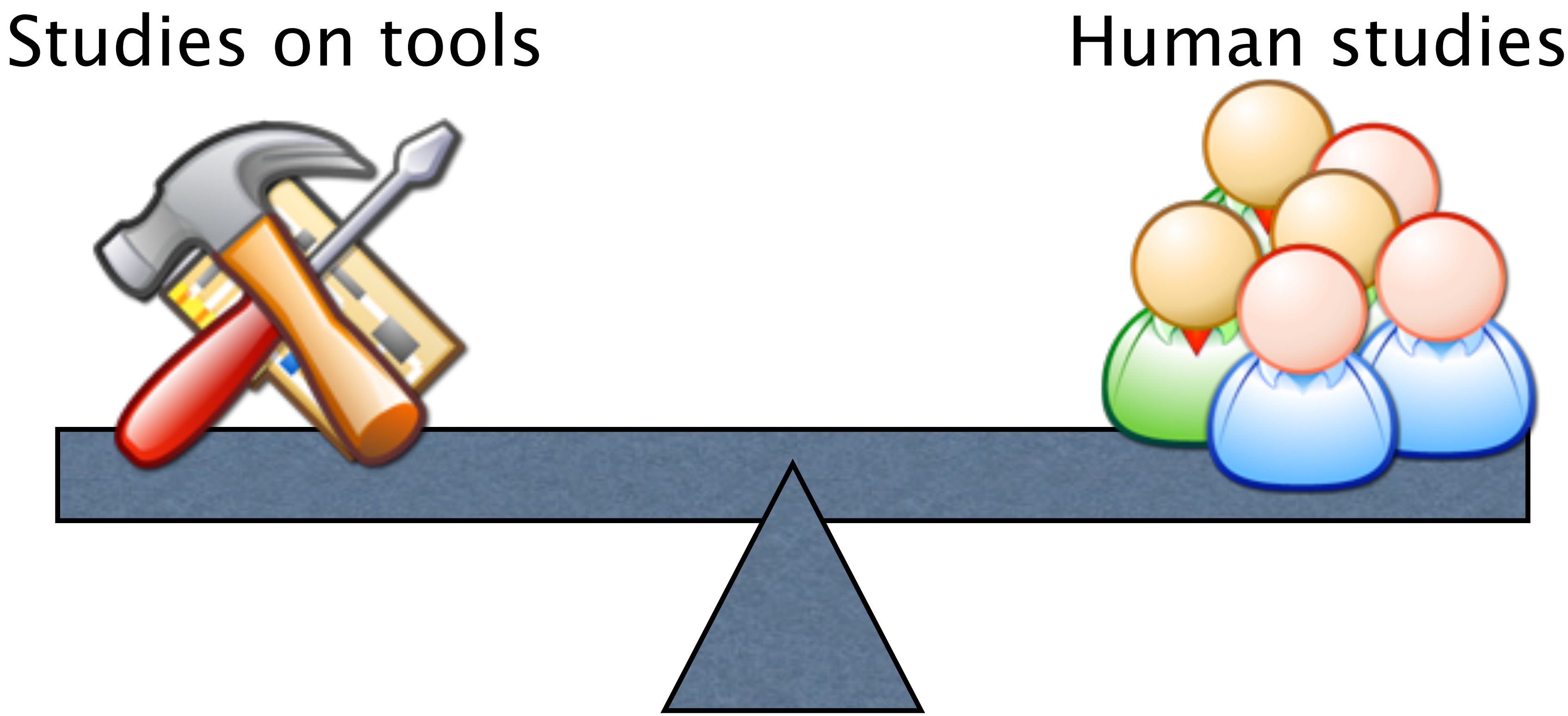
Assumption #2: Programmers inspect a list **linearly and exhaustively**

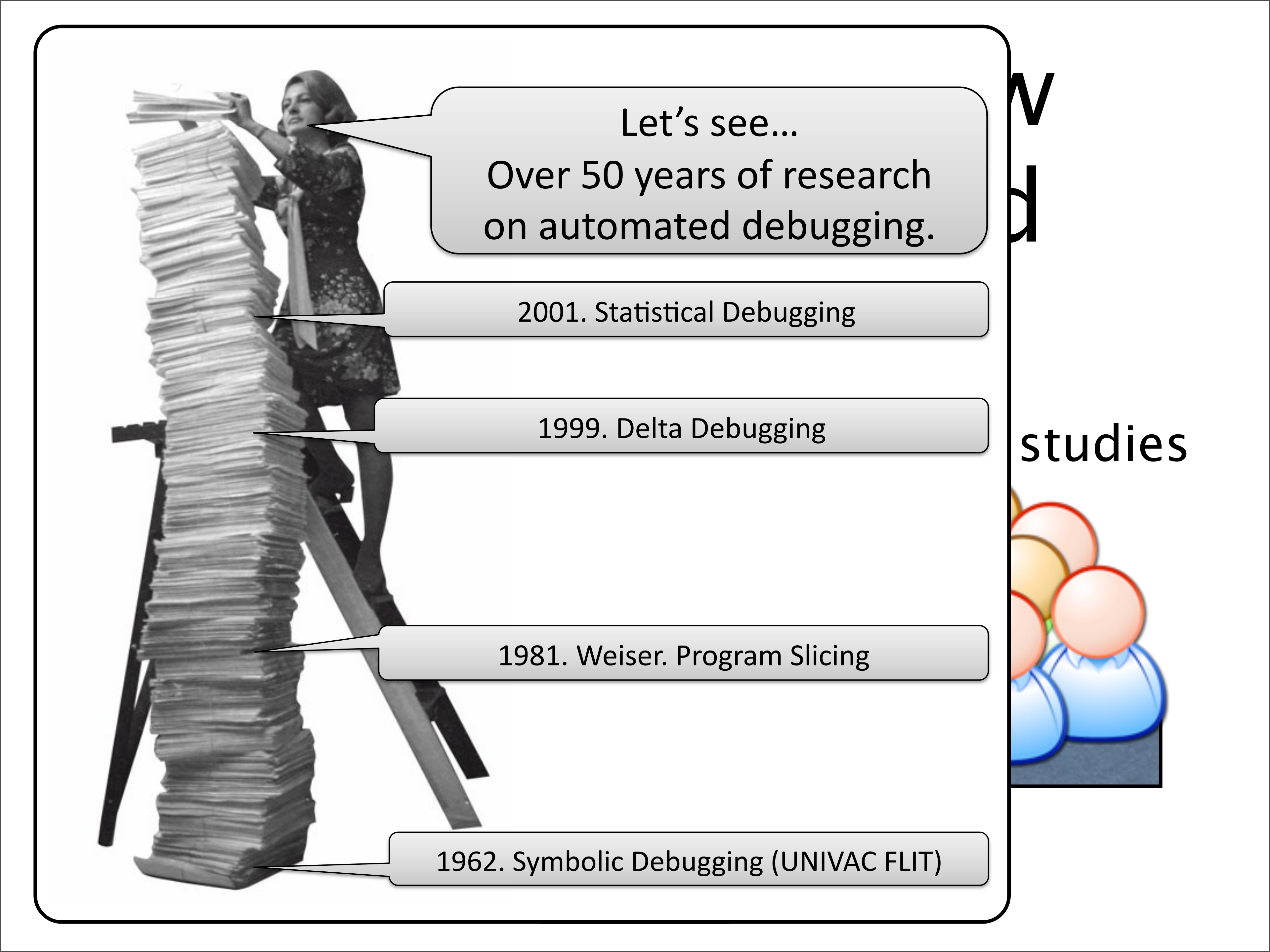
Where Shall We Go Next?

Are We Headed in the Right Direction?


AKA: "Are Automated Debugging Techniques Actually Helping Programmers?" ISSTA 2011
Chris Parnin and Alessandro Orso

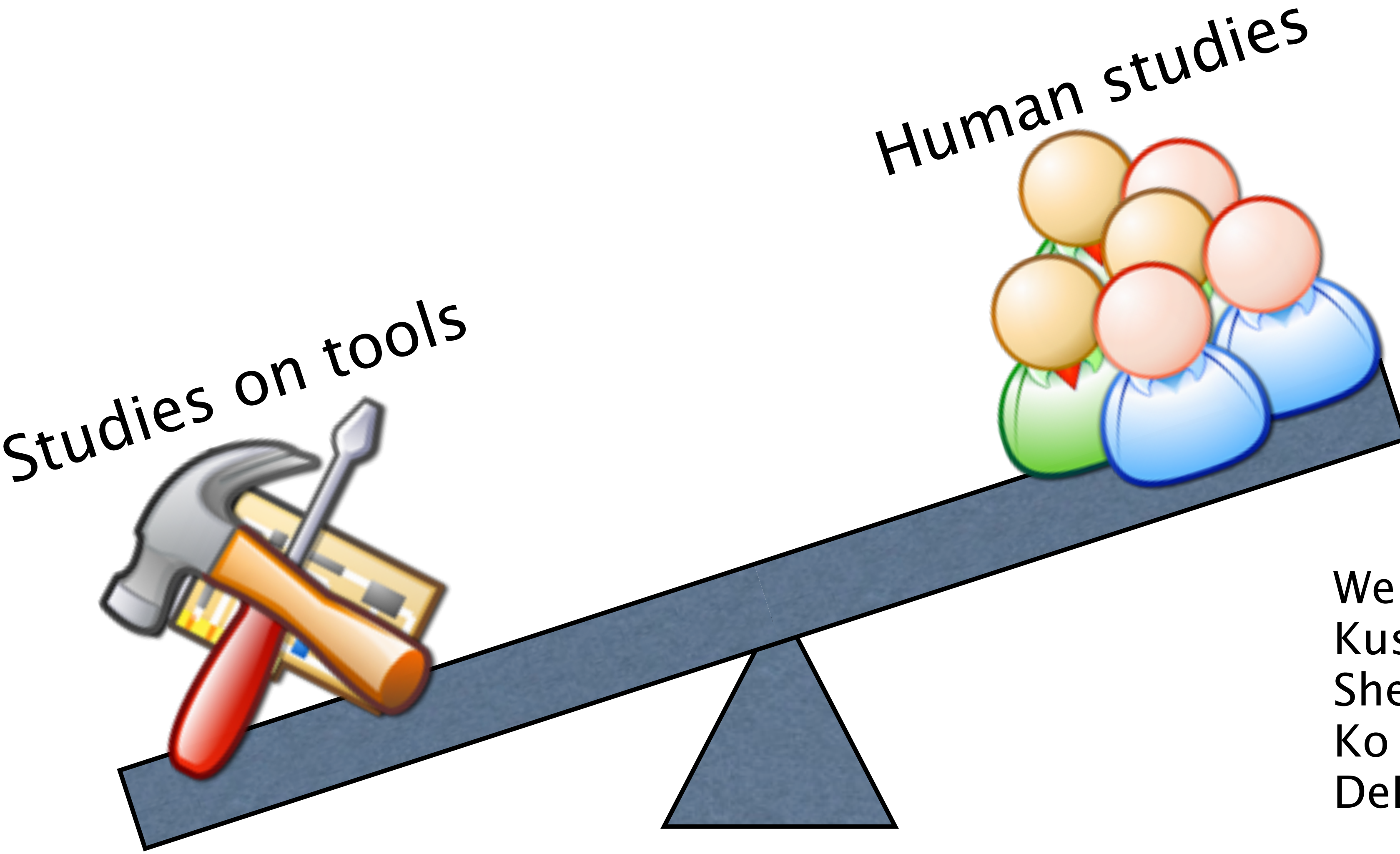
What do we know about automated





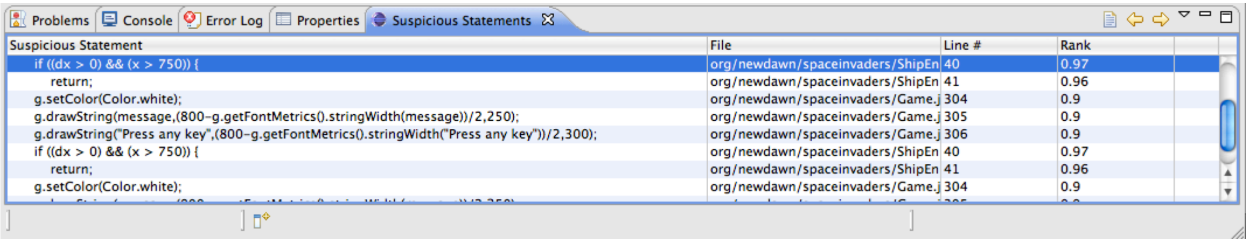
Let’s see…
Over 50 years of research
on automated debugging.
2001. Statistical Debugging
1999. Delta Debugging
1981. Weiser. Program Slicing
1962. Symbolic Debugging (UNIVAC FLIT)
studies

What do we know about automated



Weiser
Kusumoto
Sherwood
Ko
DeLine

Are these Techniques and Tools Actually Helping Programmers?



- What if we gave developers a ranked list of statements?
- How would they use it?
- Would they easily see the bug in the list?
- Would ranking make a difference?

Hypotheses

H1: Programmers who use automated debugging tools will locate bugs faster than programmers who do not use such tools

H2: Effectiveness of automated tools increases with the level of difficulty of the debugging task

H3: Effectiveness of debugging with automated tools is affected by the faulty statement's rank

Research Questions

RQ1: How do developers navigate a list of statements ranked by suspiciousness? In order of suspiciousness or jumping from one statement to the other?

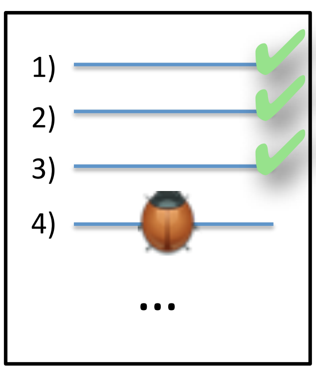
RQ2: Does perfect bug understanding exist? How much effort is involved in inspecting and assessing potentially faulty statements?

RQ3: What are the challenges involved in using automated debugging tools effectively? Can unexpected, emerging strategies be observed?

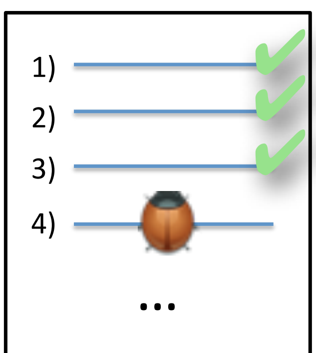
Experimental Protocol: Setup

Participants:

- 34 developers
- MS's Students
- Different levels of expertise (low, medium, high)



Experimental Protocol: Setup



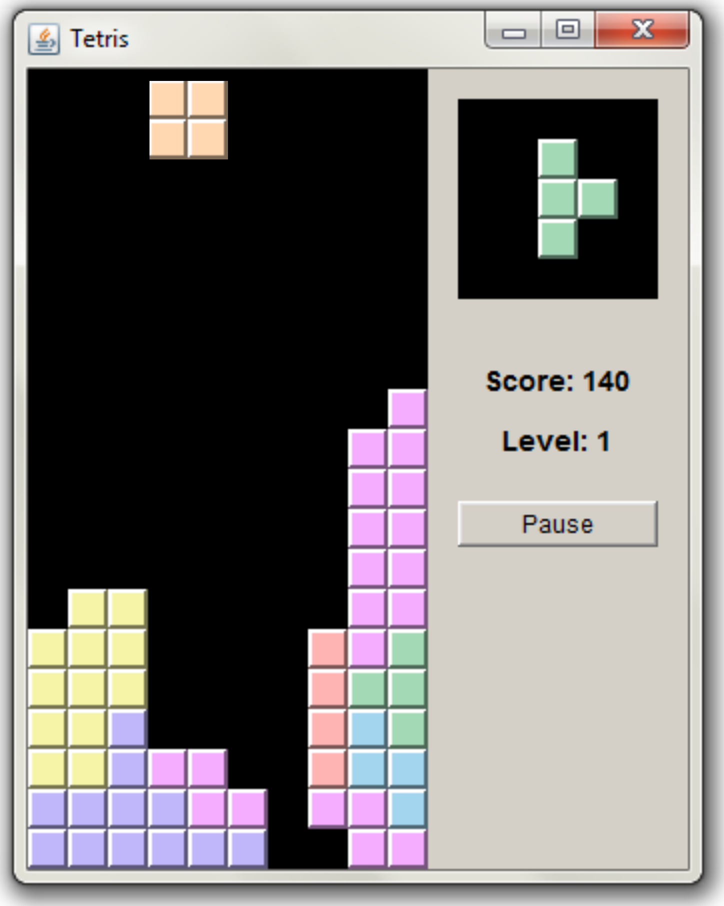
Tools
- Rank-based tool (Eclipse plug-in, logging)
- Eclipse debugger

Experimental Protocol: Setup

Software subjects:

- Tetris (~2.5KLOC)
- NanoXML (~4.5KLOC)

Tetris Bug



(Easier)

NanoXML Bug

The input, **testvm_22.xml**, contains the following input xml document:

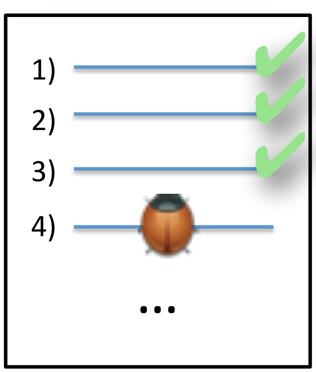
```
<Foo a="test">
  <ns:Bar>
    <Blah x="1" ns:x="2"/>
  </ns:Bar>
</Foo>
```

When running the NanoXML program (main is in class Parser1_vw_v1), the following exception is thrown:

```
Exception in thread "main" net.n3.nanoxml.XMLParseException:
XML Not Well-Formed at Line 19: Closing tag does not match opening tag: `ns:Bar' != `:Bar'
at net.n3.nanoxml.XMLUtil.errorWrongClosingTag(XMLUtil.java:497)
at net.n3.nanoxml.StdXMLParser.processElement(StdXMLParser.java:438)
        at net.n3.nanoxml.StdXMLParser.scanSomeTag(StdXMLParser.java:202)
        at net.n3.nanoxml.StdXMLParser.processElement(StdXMLParser.java:453)
        at net.n3.nanoxml.StdXMLParser.scanSomeTag(StdXMLParser.java:202)
        at net.n3.nanoxml.StdXMLParser.scanData(StdXMLParser.java:159)
        at net.n3.nanoxml.StdXMLParser.parse(StdXMLParser.java:133)
        at net.n3.nanoxml.Parser1_vw_v1.main(Parser1_vw_v1.java:50)
```

(Harder)

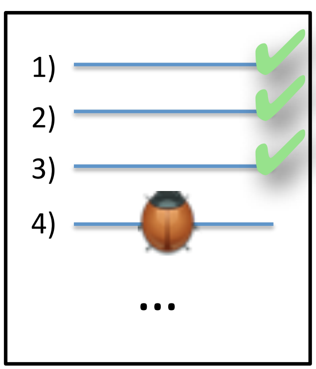
Experimental Protocol: Setup



Software subjects:

- Tetris (~2.5KLOC)
- NanoXML (~4.5KLOC)

Experimental Protocol: Setup



Tasks:

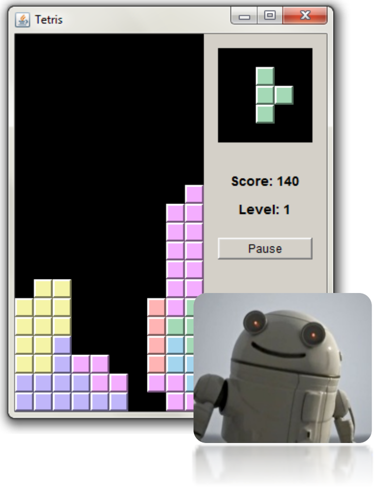
- Fault in Tetris
- Fault in NanoXML
- 30 minutes per task
- Questionnaire at the end

Experimental Protocol: Studies and Groups

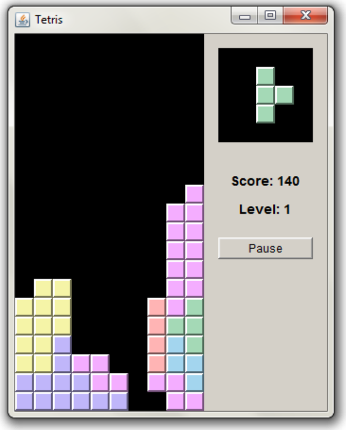
Experimental Protocol: Studies and Groups

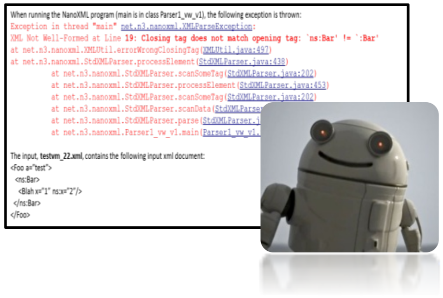
Study 1

A



B

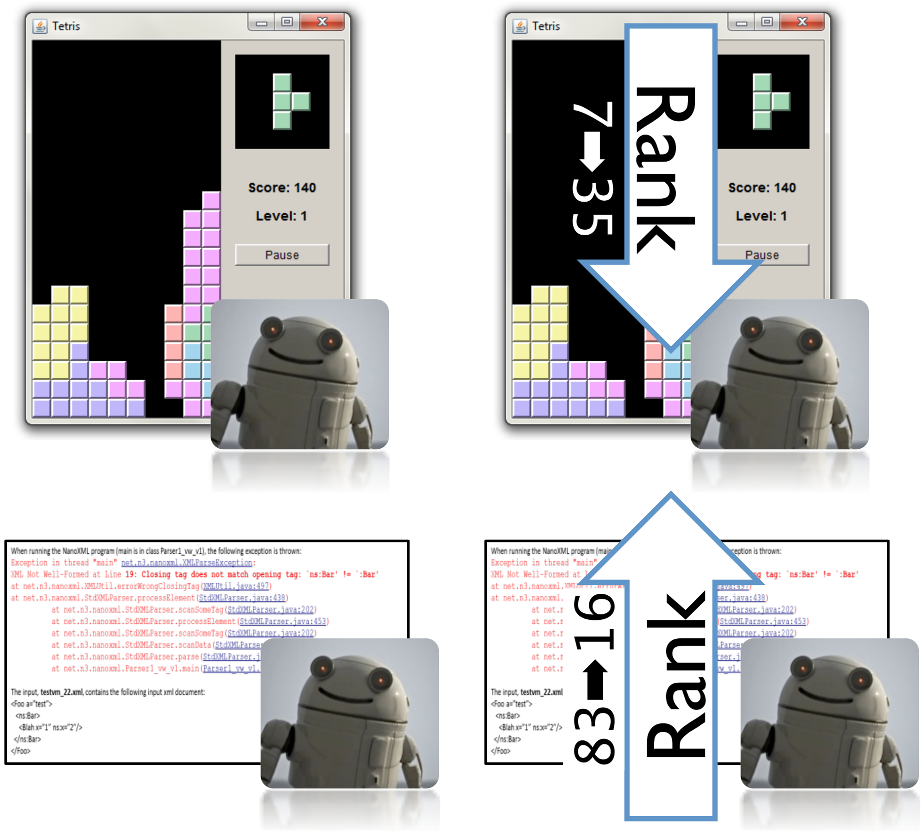




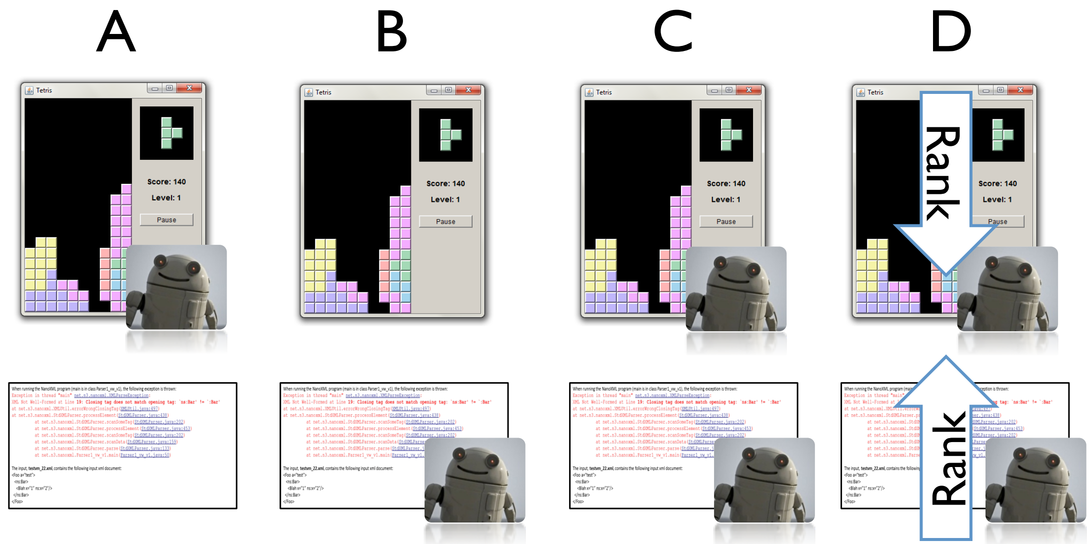
Experimental Protocol: Studies and Groups

Study 2

C D

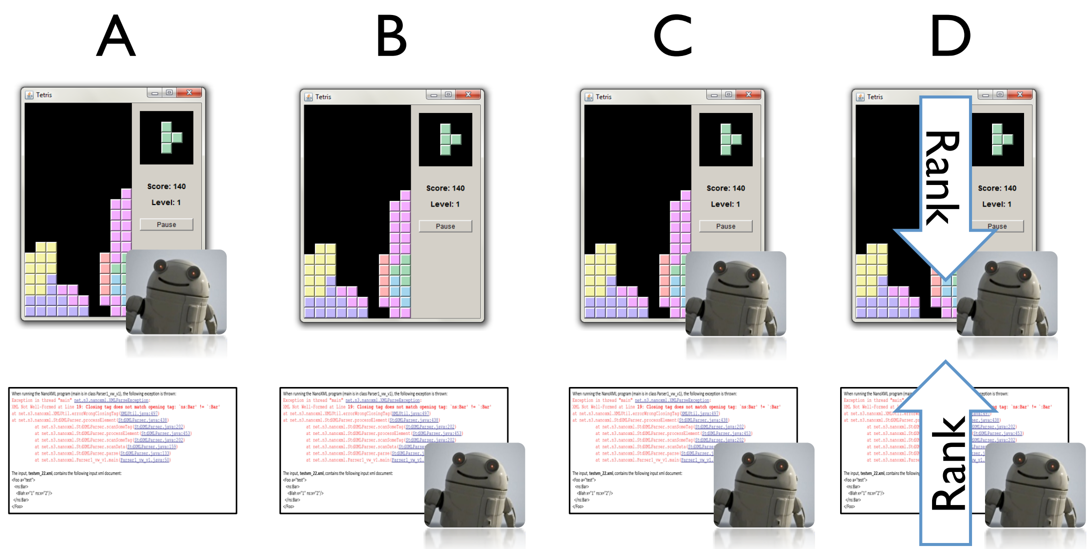


Study Results



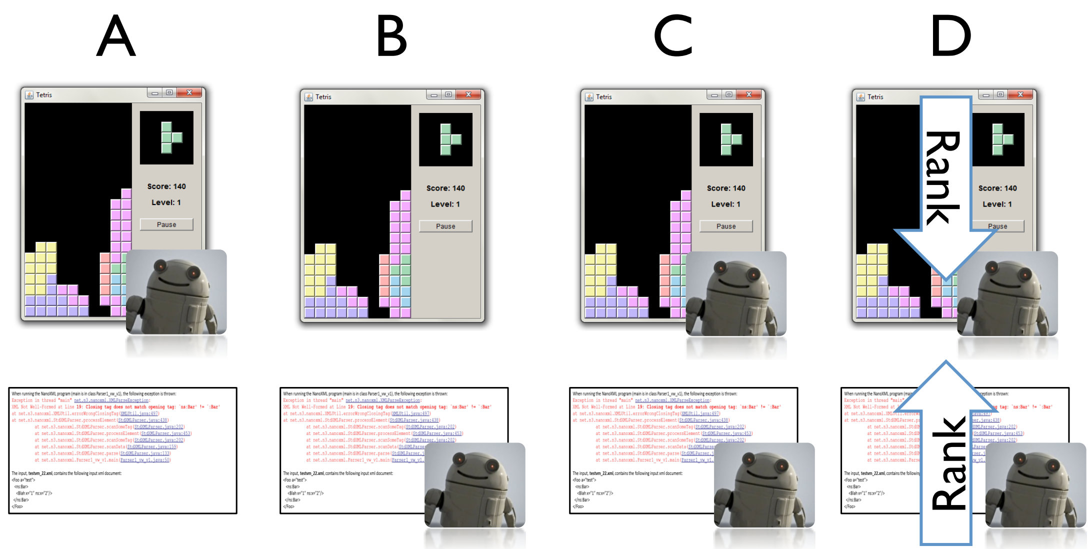
	Tetris	NanoXML
A		
B		
C		
D		

Study Results



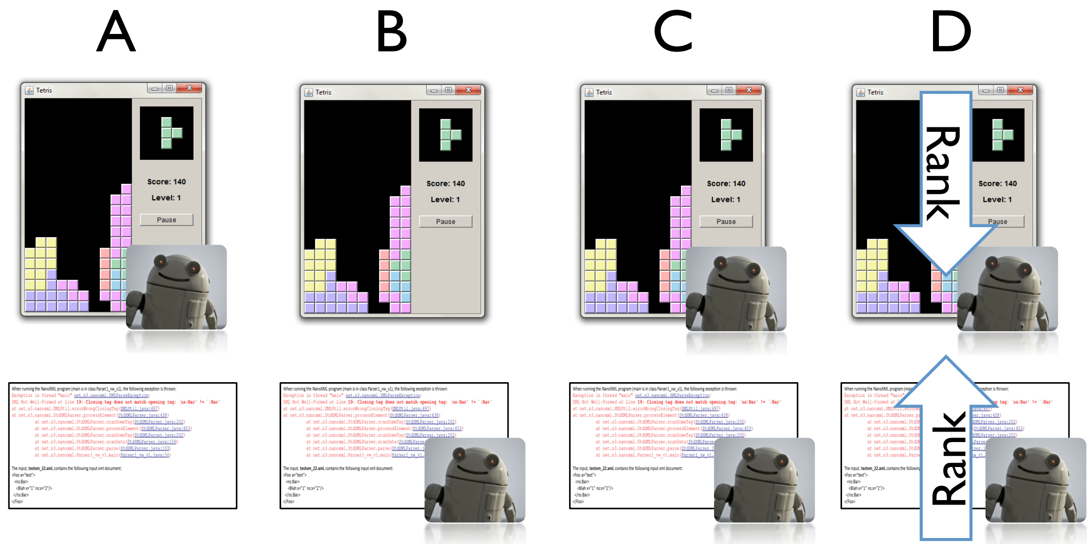
	Tetris	NanoXML
A	**Not** significantly different	
B		
C		
D		

Study Results



<table>
<tr><th></th><th>Tetris</th><th>NanoXML</th></tr>
<tr><td>A</td><td rowspan="2">Not significantly different</td><td rowspan="2">Not significantly different</td></tr>
<tr><td>B</td></tr>
<tr><td>C</td><td rowspan="2">Not significantly different</td><td rowspan="2">Not significantly different</td></tr>
<tr><td>D</td></tr>
</table>

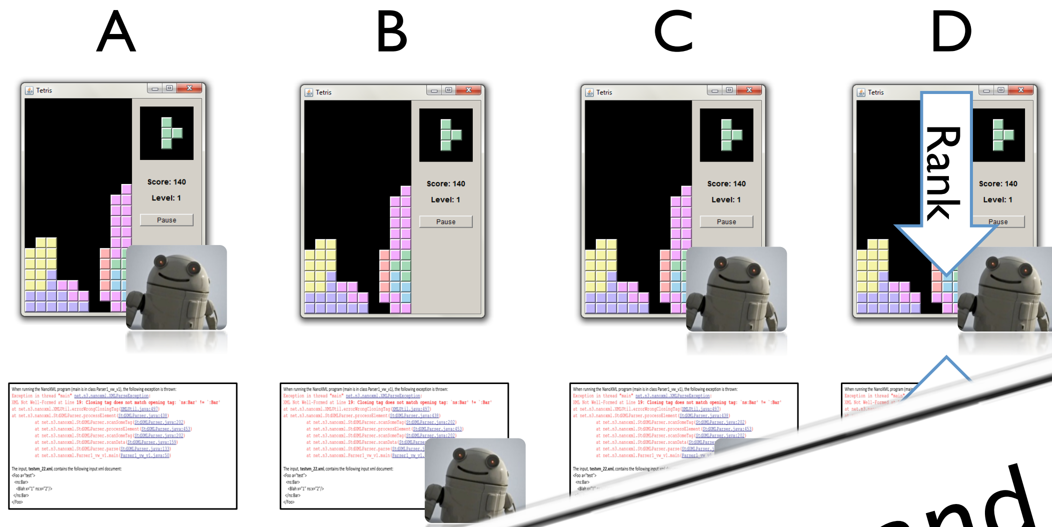
Study Results



Stratifying participants

<table>
<tr><th></th><th>Tetris</th><th>NanoXML</th></tr>
<tr><td>A</td><td rowspan="2">Significantly different for high performers</td><td rowspan="2">Not significantly different</td></tr>
<tr><td>B</td></tr>
<tr><td>C</td><td rowspan="2">Not significantly different</td><td rowspan="2">Not significantly different</td></tr>
<tr><td>D</td></tr>
</table>

Study Results



Analysis of results and questionnaires...

…icantly different

D

Not significantly different

Not significantly different

Findings: Hypotheses

H1: Programmers who use automated debugging tools will locate bugs faster than programmers who do not use such tools

Experts are faster when using the tool ➡ Support for H1 (with caveats)

H2: Effectiveness of automated tools increases with the level of difficulty of the debugging task

The tool did not help harder tasks ➡ No support for H2

H3: Effectiveness of debugging with automated tools is affected by the faulty statement's rank

Changes in rank have no significant effects ➡ No support for H3

Findings: RQs

RQ1: How do developers navigate a list of statements ranked by suspiciousness? In order of suspiciousness or jumping b/w stmts?

Programmers do not visit each statement in the list, they **search**

RQ2: Does perfect bug understanding exist? How much effort is involved in inspecting and assessing potentially faulty statements?

Perfect bug understanding is generally not a realistic assumption

RQ3: What are the challenges involved in using automated debugging tools effectively? Can unexpected, emerging strategies be observed?

1) The statements in the list were sometimes useful as starting points
2) (Tetris) Several participants preferred to search based on intuition
3) (NanoXML) Several participants gave up on the tool after investigating too many false positives

Research Implications

- Percentages will not cut it (e.g., 1.8% == 83rd position)
  ➡ **Implication 1:** Techniques should focus on improving absolute rank rather than percentage rank
- Ranking can be successfully combined with search
  ➡ **Implication 2:** Future tools may focus on searching through (or automatically highlighting) certain suspicious statements
- Developers want explanations, not recommendations
  ➡ **Implication 3:** We should move away from pure ranking and define techniques that provide context and ability to explore
- We must grow the ecosystem
  ➡ **Implication 4:** We should aim to create an ecosystem that provides the entire tool chain for fault localization, including managing and orchestrating test cases

In Summary

- We came a **long** way since the early days of debugging

...

- There is still a **long** way to go...

Where Shall We Go Next

- Hybrid, semi-automated fault localization techniques
- Debugging of field failures (with limited information)
- Failure understanding and explanation
- (Semi-)automated repair and workarounds
- User studies, user studies, user studies!
  (true also for other areas)

With much appreciated input/contributions from

- Andy Ko
- Wei Jin
- Jim Jones
- Wes Masri
- Chris Parnin
- Abhik Roychoudhury
- Wes Weimer
- Tao Xie
- Andreas Zeller
- Xiangyu Zhang