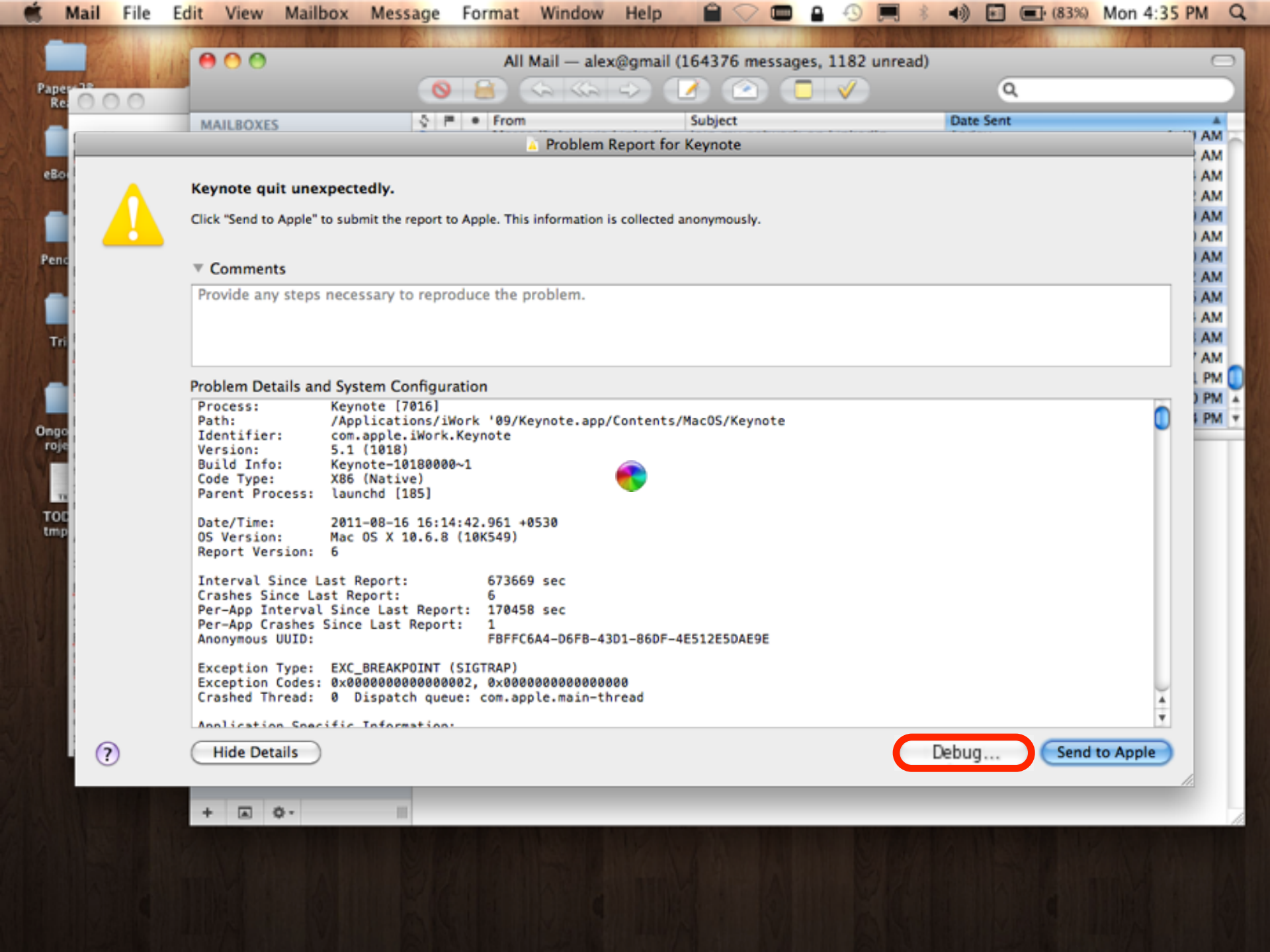
Software Debugging: Past, Present, and Future

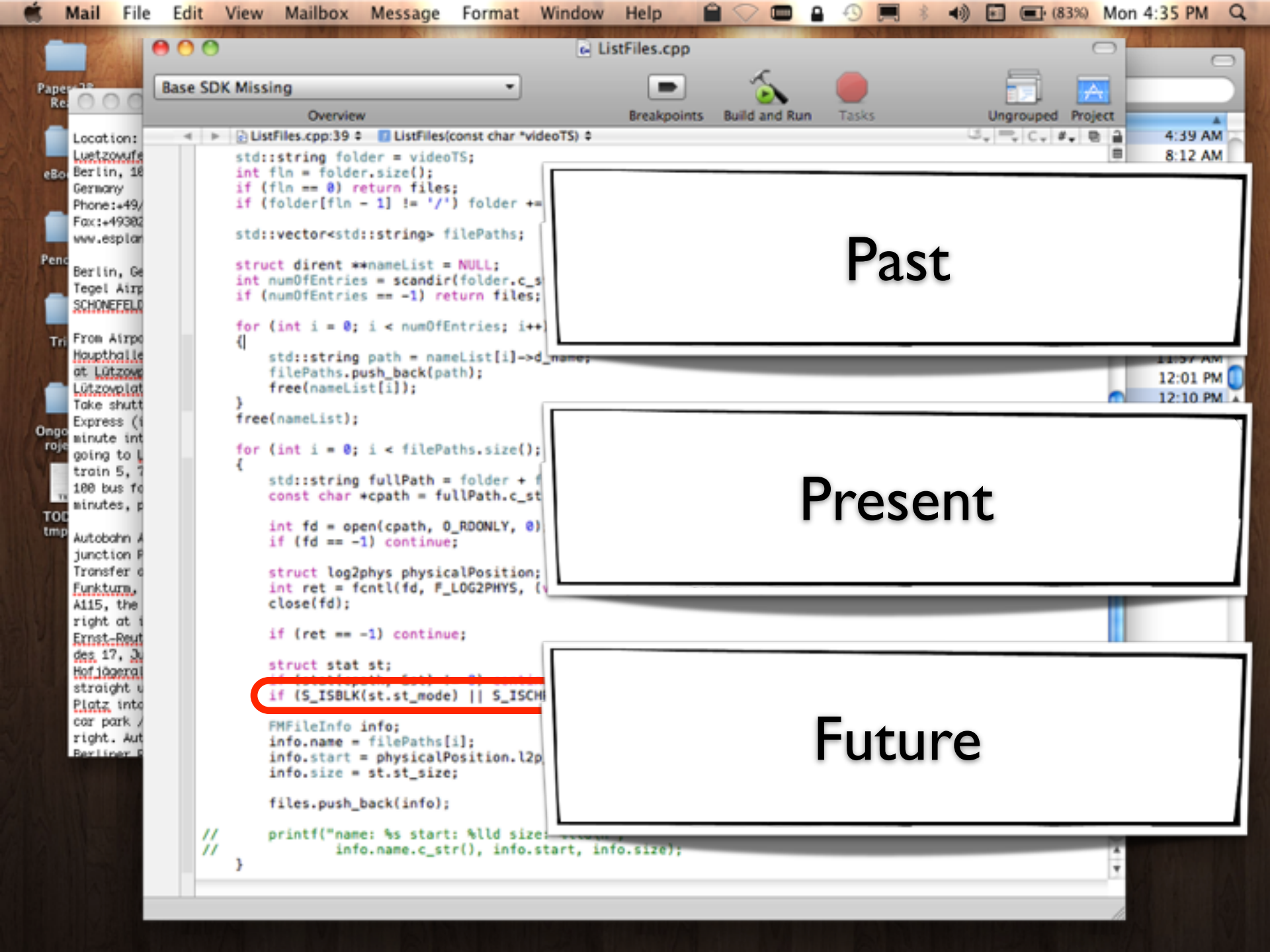
Alessandro (Alex) Orso
School of Computer Science – College of Computing
Georgia Institute of Technology
http://www.cc.gatech.edu/~orso/

Partially supported by: NSF, Google, IBM, and MSR

Automated Debugging

Dare to Dream


Mail File Edit View Mailbox Message Format Window Help
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


Mail File Edit View Mailbox Message Format Window Help
ListFiles.cpp
Base SDK Missing
Overview
Breakpoints
Build and Run
Tasks
Ungrouped
Project
ListFiles.cpp:39
ListFiles(const char *videoTS)
Past
Present
Future

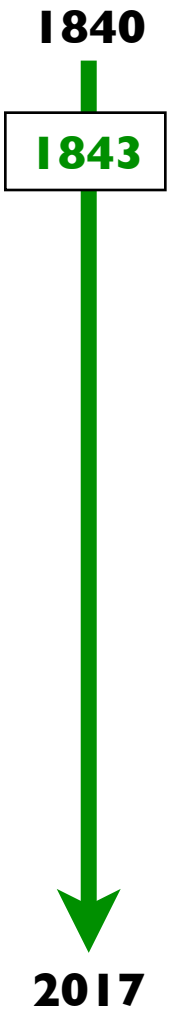
Past

A Short History of Debugging

The Birth of Debugging

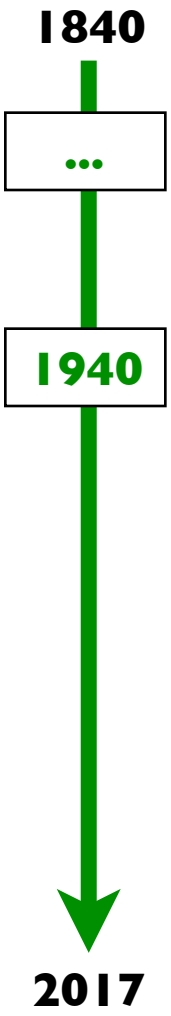
First reference to software errors
Your guess?

The Birth of Debugging



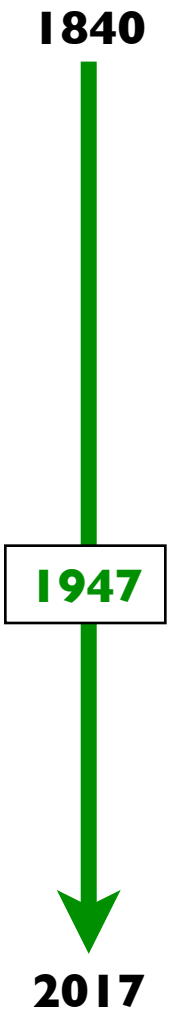
- *Software errors* mentioned in Ada Byron's notes on Charles Babbage's analytical engine

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



- *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
- First actual *bug* and actual *debugging* (Admiral Grace Hopper's associates working on Mark II Computer at Harvard University)

The Birth of Debugging

1840

1947

2017

- 9/9

  0800 Arctan started

  1000 " stopped - arctan ✓

  13" u/c (033) MP - MC

  (033) PRO 2

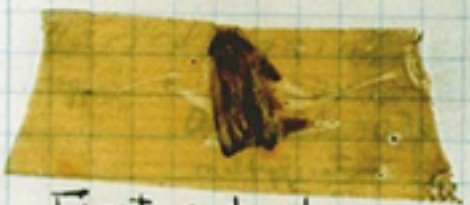
  correct

  { 1.2700 9.037 847 025
  9.037 846 ?95 correct
  ~~1.9821649000~~ ~~2.13047764715 (-3)~~ 4.615925059(-2)
  2.13047764715
  2.13067764715

  Relays 6-2 in 033 failed special speed test
  In relay . 1100 test .
  Relays changed

  1100 Started Cosine Tape (Sine check)

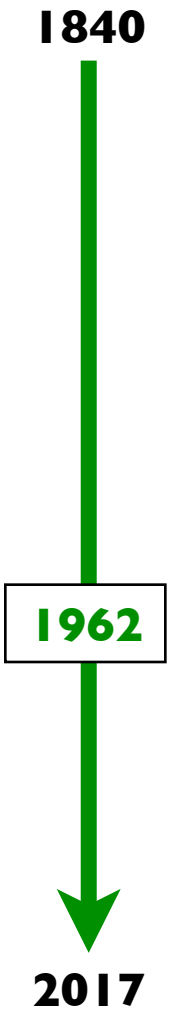
  1525 Started Multy Adder Test.

- 1545  Relay #70 Panel F
  (moth) in relay.

  First actual case of bug being found.

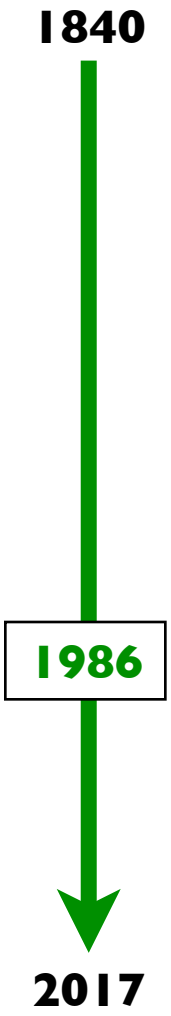
  1700 closed down.

Symbolic Debugging



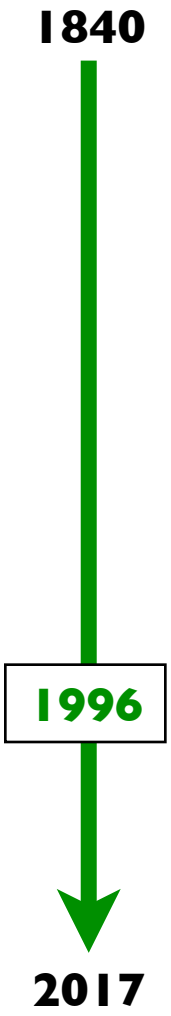
- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)

Symbolic Debugging



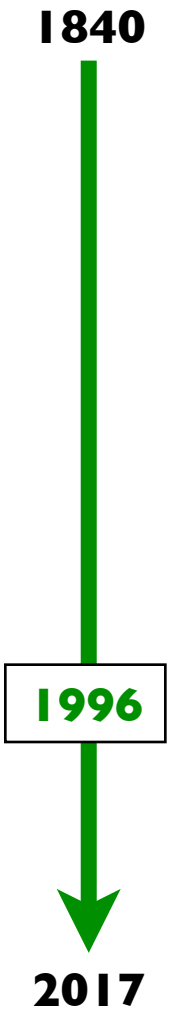
- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)
- GDB

Symbolic Debugging



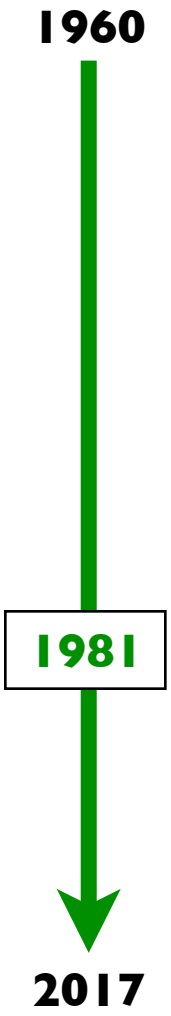
- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)
- GDB
- DDD

Symbolic Debugging



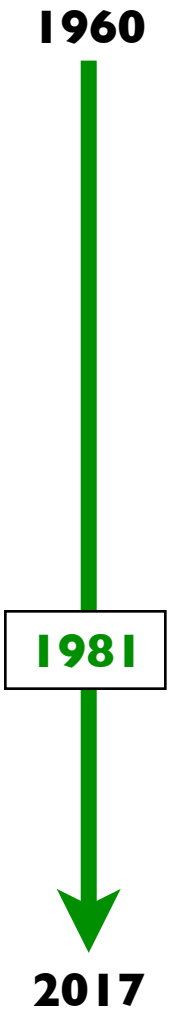
- UNIVAC 1100's FLIT
  (Fault Location by Interpretive Testing)
- GDB
- DDD
- ...

Program Slicing



- **Intuition**: developers "slice" backwards when debugging

Program Slicing

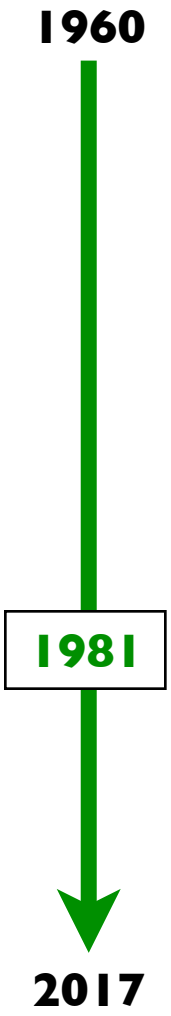


- **Intuition**: developers "slice" backwards when debugging
- Weiser's breakthrough paper

Static Slicing Example

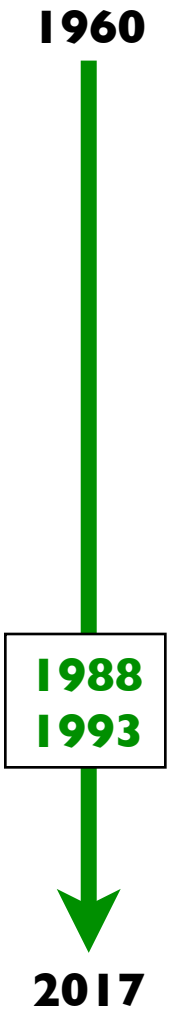
```
   mid() {
     int x,y,z,m;
1:   read("Enter 3 numbers:",x,y,z);
2:   m = z;
3:   if (y<z)
4:      if (x<y)
5:            m = y;
6:      else if (x<z)
7:            m = y; // bug
8:   else
9:      if (x>y)
10:           m = y;
11:     else if (x>z)
12:           m = x;
13:  print("Middle number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

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;
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);
   }
```

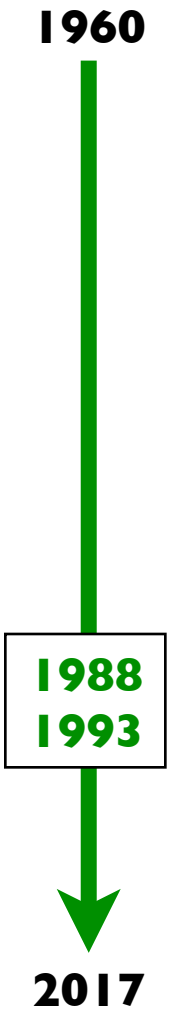
Dynamic Slicing Example

		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



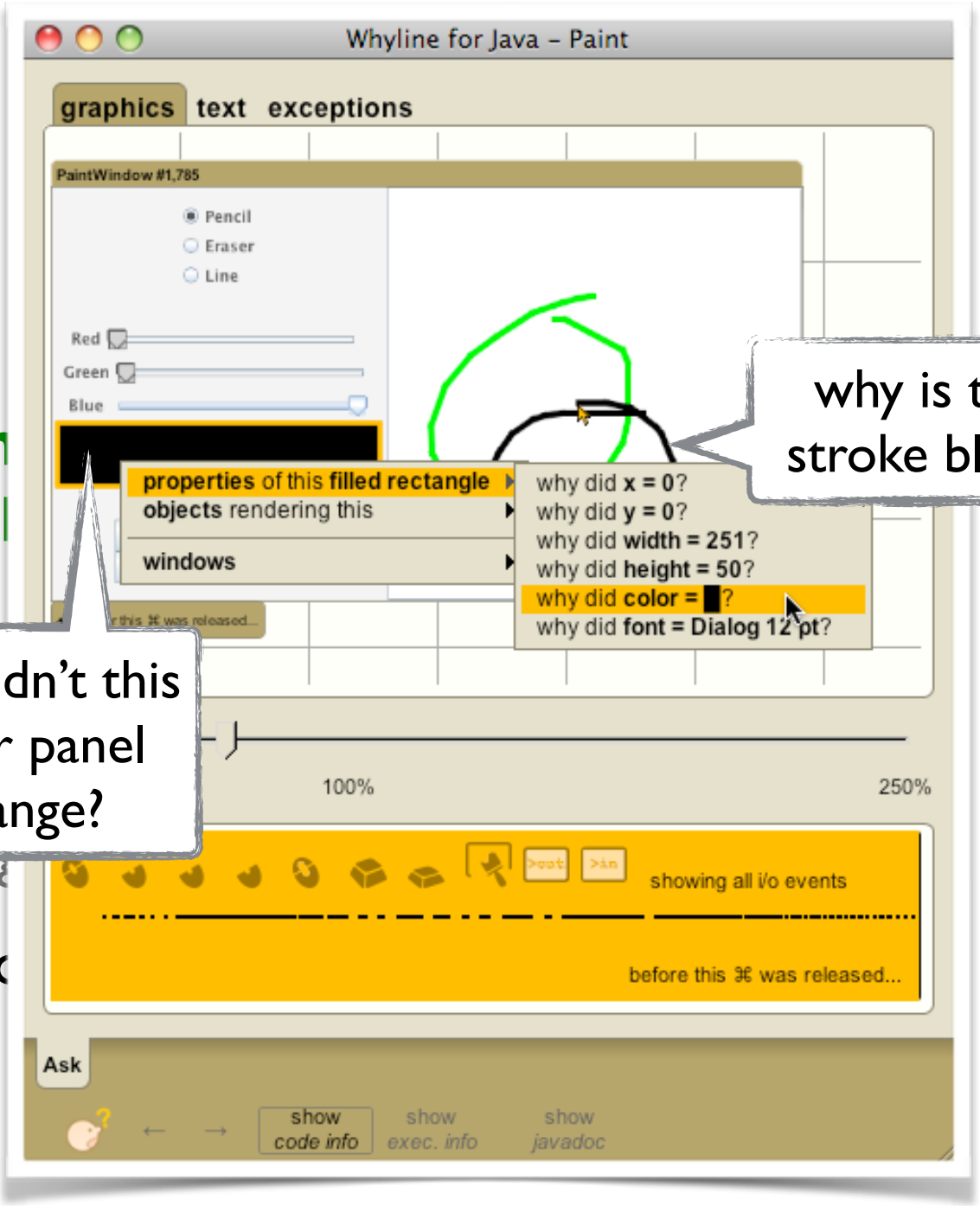
- **Intuition**: developers "slice" backwards when debugging
- Weiser's breakthrough paper
- Korel and Laski's dynamic slicing
- Agrawal

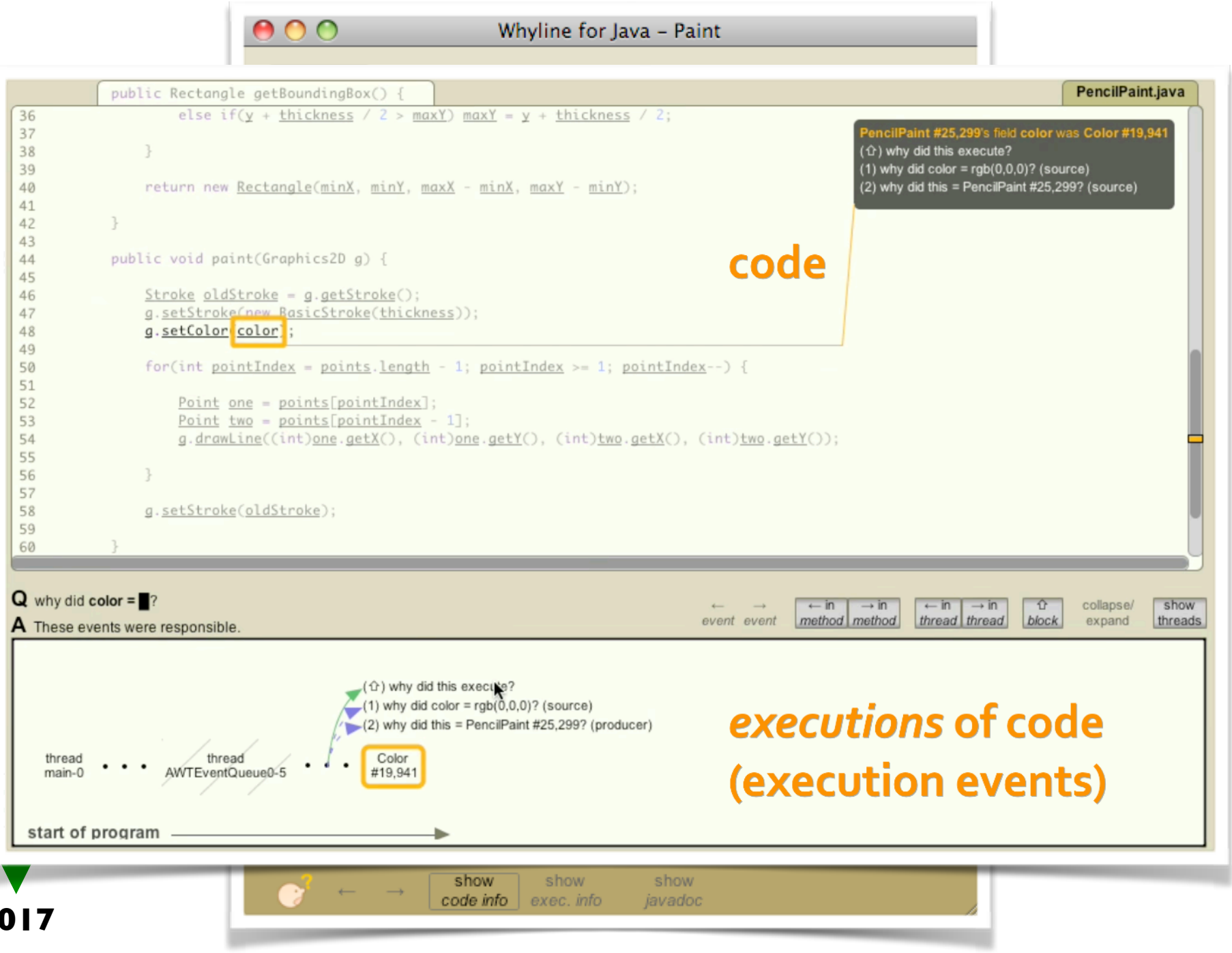
1960

2008

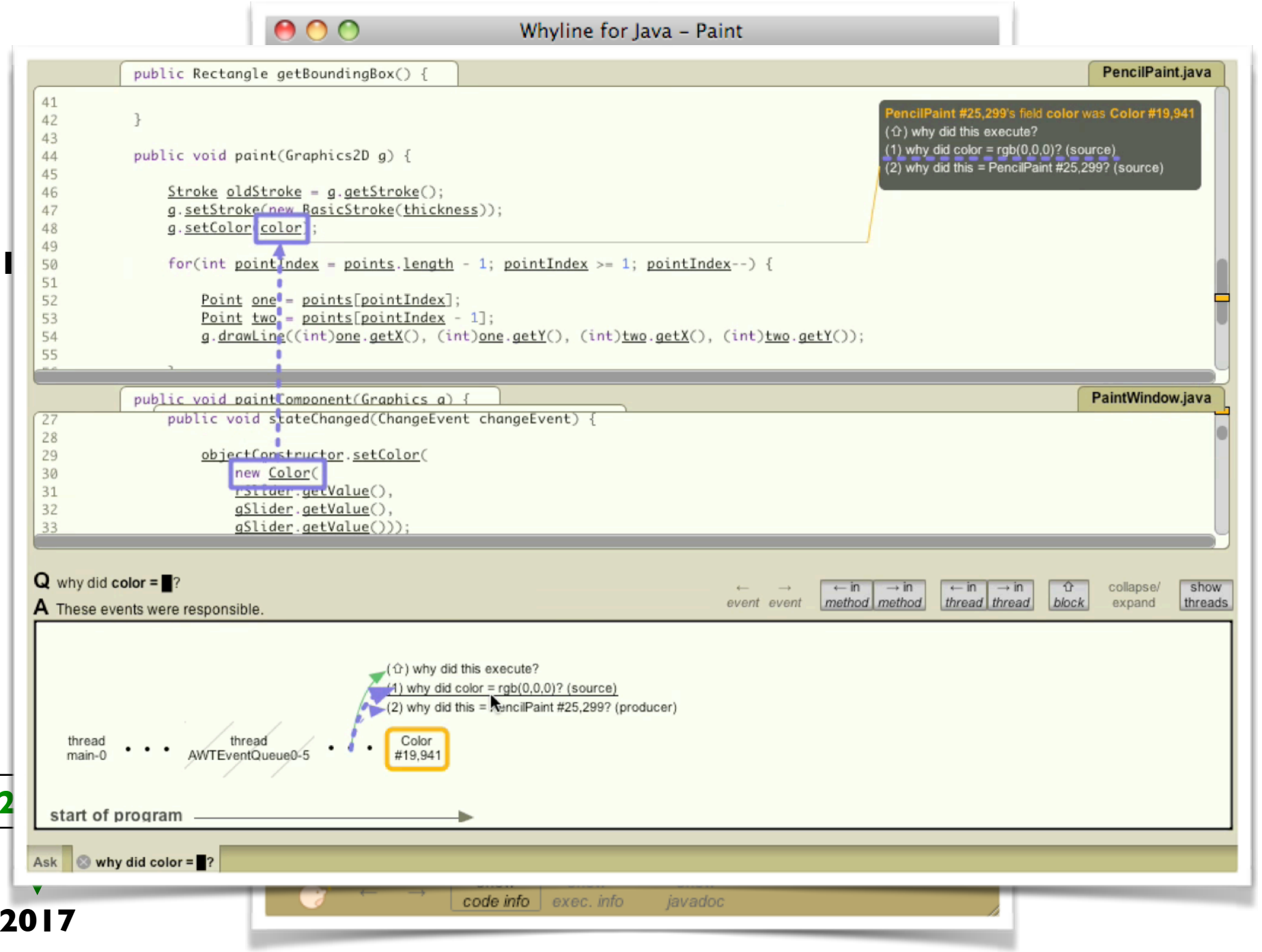
2017

- In
- wh
- K



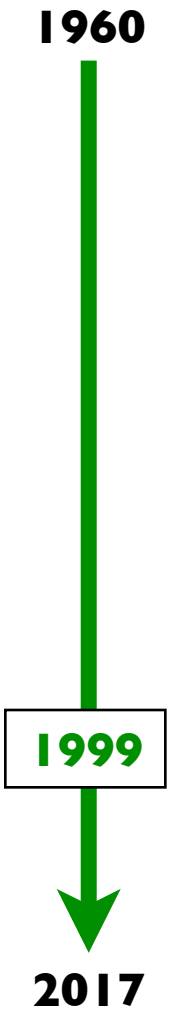

Whyline for Java – Paint
public Rectangle getBoundingBox() {
PencilPaint.java
36 else if(y + thickness / 2 > maxY) maxY = y + thickness / 2;
37
38 }
39
40 return new Rectangle(minX, minY, maxX - minX, maxY - minY);
41
42 }
43
44 public void paint(Graphics2D g) {
45
46 Stroke oldStroke = g.getStroke();
47 g.setStroke(new BasicStroke(thickness));
48 g.setColor(color);
49
50 for(int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--) {
51
52 Point one = points[pointIndex];
53 Point two = points[pointIndex - 1];
54 g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());
55
56 }
57
58 g.setStroke(oldStroke);
59
60 }
PencilPaint #25,299's field color was Color #19,941
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (source)
code
Q why did color = ?
A These events were responsible.
← event
→ event
← in method
→ in method
← in thread
→ in thread
⇧ block
collapse/ expand
show threads
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (producer)
thread main-0
thread AWTEventQueue0-5
Color #19,941
start of program
executions of code (execution events)
show code info
show exec. info
show javadoc

2017


Whyline for Java – Paint
public Rectangle getBoundingBox() {
PencilPaint.java
41
42 }
43
44 public void paint(Graphics2D g) {
45
46 Stroke oldStroke = g.getStroke();
47 g.setStroke(new BasicStroke(thickness));
48 g.setColor(color);
49
50 for(int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--) {
51
52 Point one = points[pointIndex];
53 Point two = points[pointIndex - 1];
54 g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());
55
PencilPaint #25,299's field color was Color #19,941
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (source)
public void paintComponent(Graphics g) {
PaintWindow.java
27 public void stateChanged(ChangeEvent changeEvent) {
28
29 objectConstructor.setColor(
30 new Color(
31 rSlider.getValue(),
32 gSlider.getValue(),
33 gSlider.getValue()));
Q why did color = ?
A These events were responsible.
← event
→ event
← in method
→ in method
← in thread
→ in thread
⇧ block
collapse/ expand
show threads
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (producer)
thread main-0
thread AWTEventQueue0-5
Color #19,941
start of program
Ask
why did color = ?
code info
exec. info
javadoc

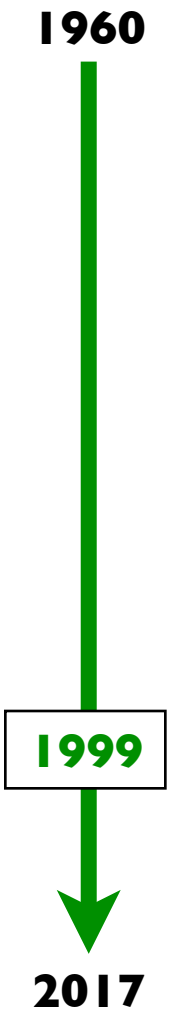
2017

Delta Debugging



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

Delta Debugging



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

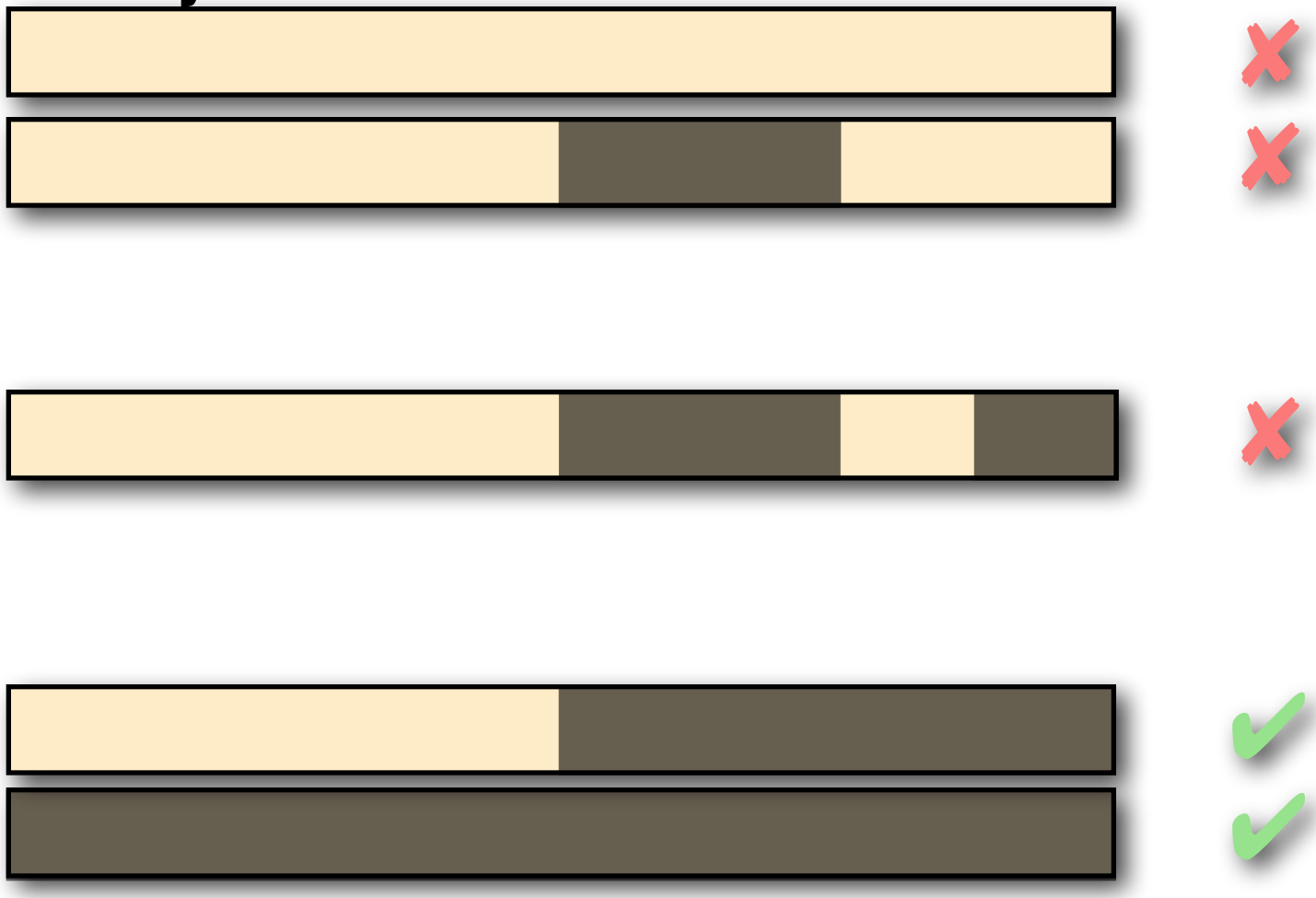
Today

Yesterday

Today

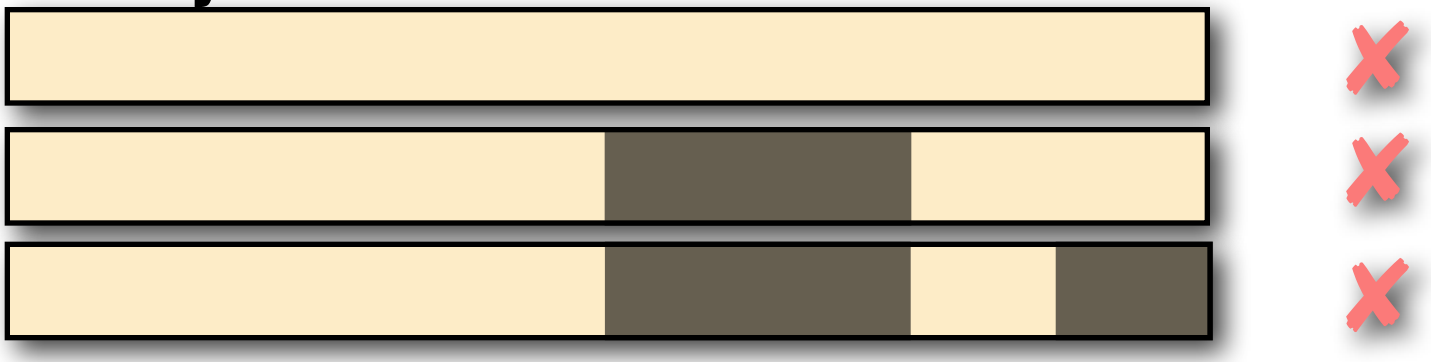
Yesterday

Today

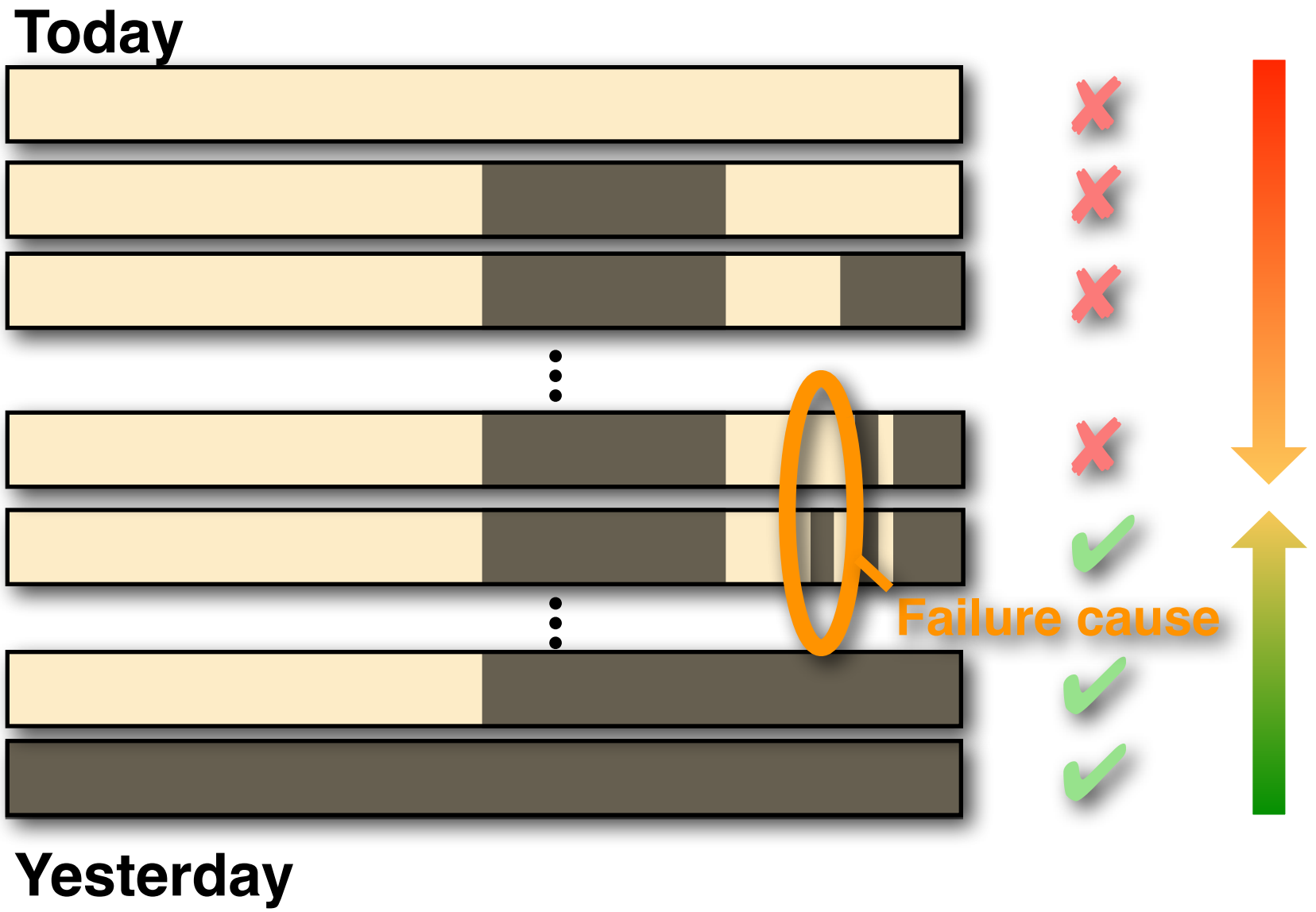


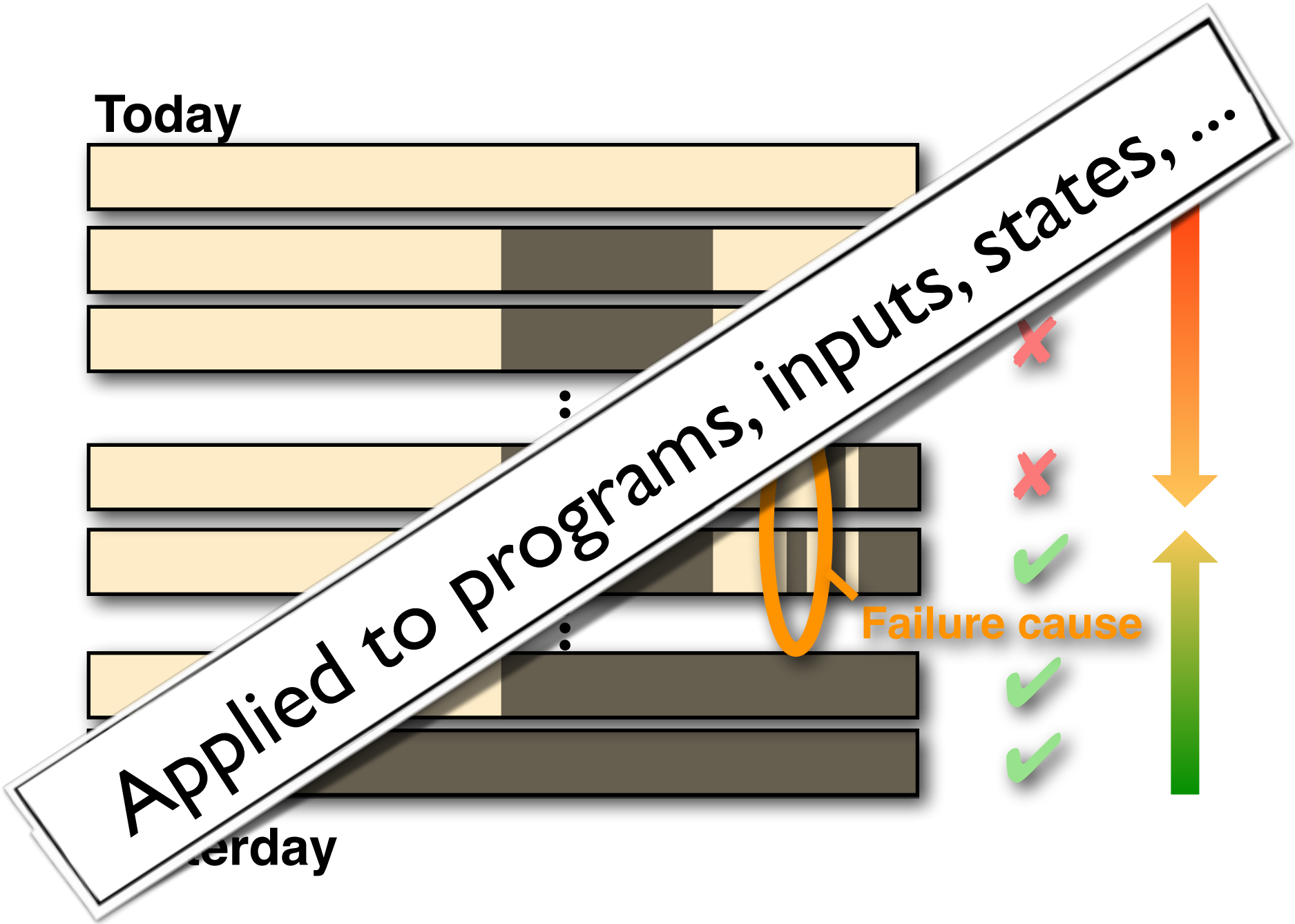
Yesterday

Today

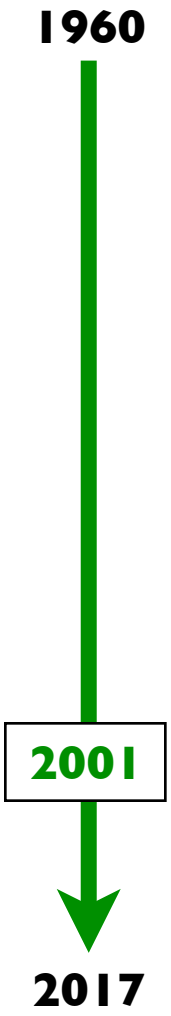


Yesterday


Today
Failure cause
Yesterday

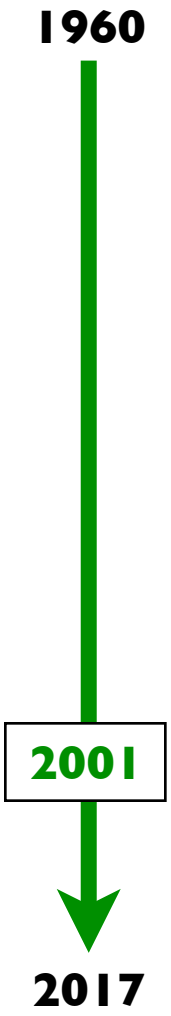

Today
Applied to programs, inputs, states, ...
Failure cause
erday

Statistical Debugging



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

Statistical Debugging

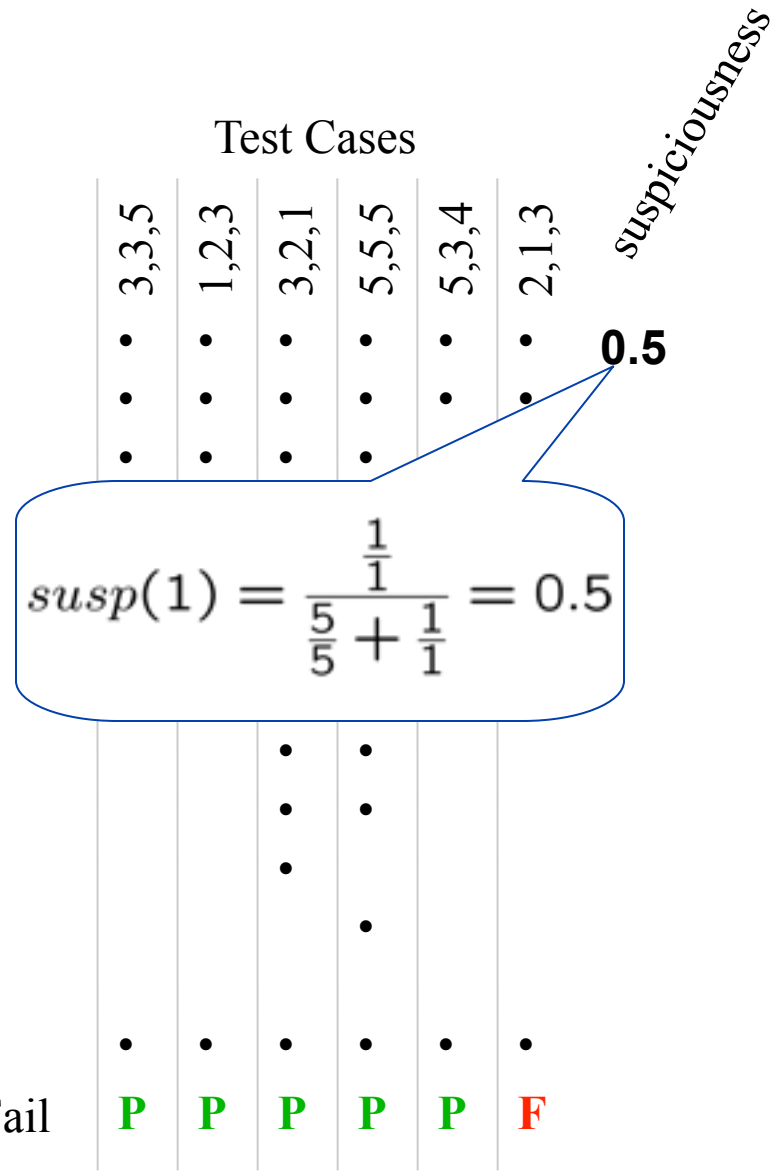


- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula

Tarantula

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

```
   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);
   }
```



Tarantula

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

```
   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);
   }
```

	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	suspiciousness
	•	•	•	•	•	•	0.5
	•	•	•	•	•	•	0.5
	•	•	•	•	•	•	0.5
	•	•			•	•	0.6
		•					0.0
	•				•	•	0.7
	•					•	0.8
			•	•			0.0
			•	•			0.0
			•				0.0
				•			0.0
							0.0
	•	•	•	•	•	•	0.5
Pass/Fail	P	P	P	P	P	F	

Test Cases

Tarantula

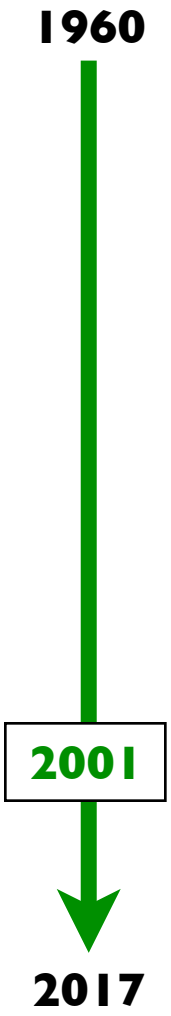
$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

```
   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);
   }
```

	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	suspiciousness
1:	•	•	•	•	•	•	0.5
2:	•	•	•	•	•	•	0.5
3:	•	•	•	•	•	•	0.5
4:	•	•			•	•	0.6
5:		•					0.0
6:	•				•	•	0.7
7:	•					•	0.8
8:			•	•			0.0
9:			•	•			0.0
10:			•				0.0
11:				•			0.0
12:							0.0
13:	•	•	•	•	•	•	0.5
Pass/Fail	P	P	P	P	P	F	

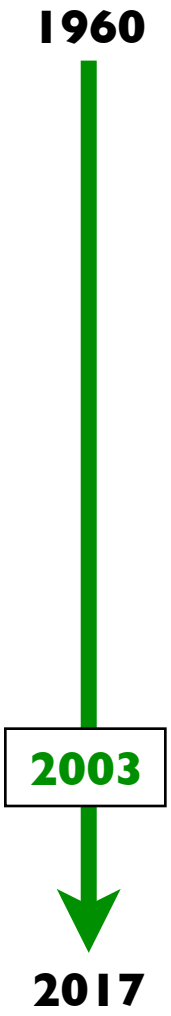
Test Cases

Statistical Debugging



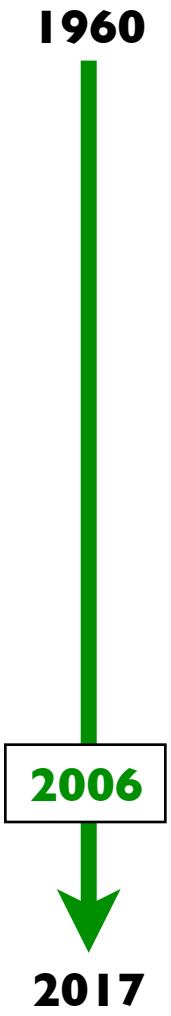
- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula

Statistical Debugging



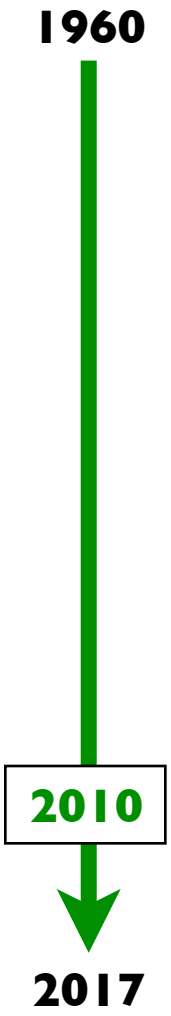
- **Intuition**: debugging techniques can leverage multiple executions
- Tarantula
- CBI

Statistical Debugging



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

Statistical Debugging



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

Statistical Debugging

1960

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

2008

2017

IR-Based Techniques

Bug ID: 90018
Summary: Native tooltips left around on CTabFolder.
Description: Hover over the PartStack CTabFolder inside eclipse until some native tooltip is displayed. For example, the maximize button. When the tooltip appears, change perspectives using the keybinding. the CTabFolder gets hidden, but its tooltip is permanently displayed and never goes away. Even if that CTabFolder is disposed (I'm assuming) when the perspective is closed.

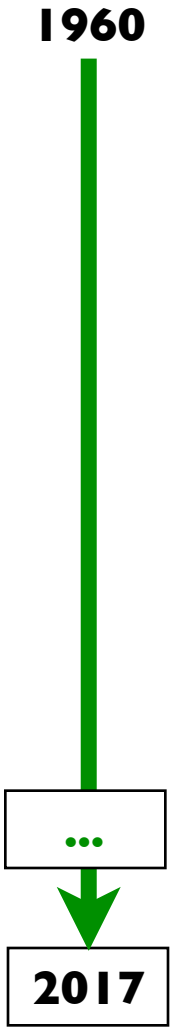
IR-Based Techniques

Bug ID: 90018
Summary: Native tooltips left around on CTabFolder.
Description: Hover over the PartStack CTabFolder inside eclipse until some native tooltip is displayed. For example, the maximize button. When the tooltip appears, change perspectives using the keybinding. the CTabFolder gets hidden, but its tooltip is permanently displayed and never goes away. Even if that CTabFolder is disposed (I'm assuming) when the perspective is closed.

Source code file: **CTabFolder**.java

```
public class CTabFolder extends Composite {
    // tooltip
    int [] toolTipEvents = new int[] {SWT.MouseExit,
    SWT.MouseHover, SWT.MouseMove,
    SWT.MouseDown, SWT.DragDetect};
    Listener toolTipListener;
    …
    / * Returns <code>true</code> if the CTabFolder
    only displys the selected tab
     * and <code>false</code> if the CTabFolder
    displays multiple tabs.
     */
    …void onMouseHover(Event event) {
      showToolTip(event.x, event.y);
    }
    void onDispose() {
        inDispose = true;
      hideToolTip();
        …
    }
}
```

Statistical Debugging



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

Additional Techniques

1960

2017

- 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., Zhang et al.)
- Fault localization for multiple faults (e.g., Steimann et al.)
- Debugging of concurrency failures (e.g., Park et al.)
- Automated data structure repair (e.g., Rinard et al.)
- Finding patches with genetic programming
- Domain specific fixes
  (tests, web pages, comments, concurrency)
- Identifying workarounds/recovery strategies (e.g., Gorla et al.)
- Formula based debugging (e.g., Jose et al., Ermis et al.)
- ...

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., Zhang et al.)
- Fault localization for mul
- Debuggi
- ming
- pages, comments, concurrency)
- Identifying workarounds/recovery strategies (e.g., Gorla et al.)
- Formula based debugging (e.g., Jose et al., Ermis et al.)
- ...

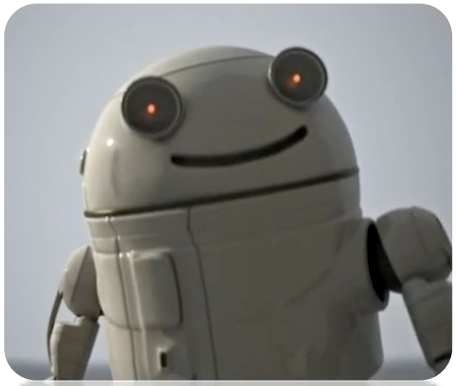
Not meant to be comprehensive!

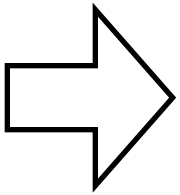
2017

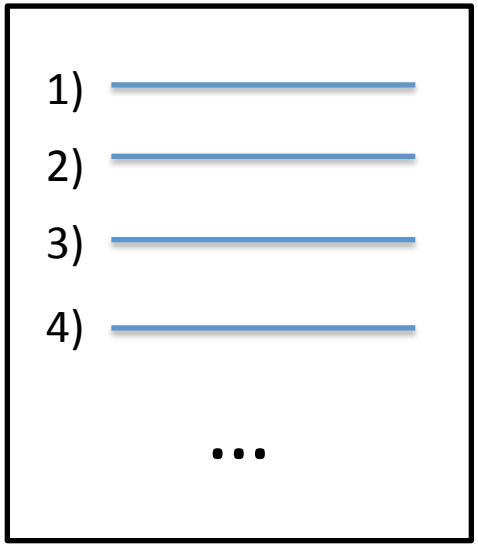
Present

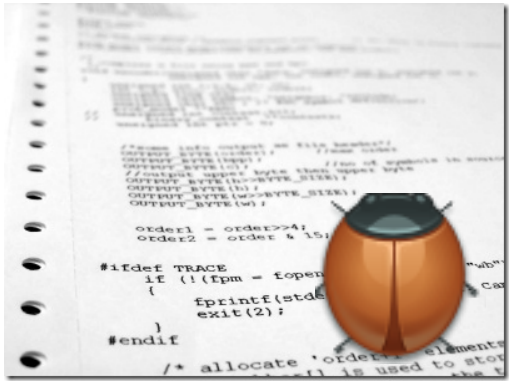
Can We Debug at the Push of a Button?

Automated Debugging (rank based)

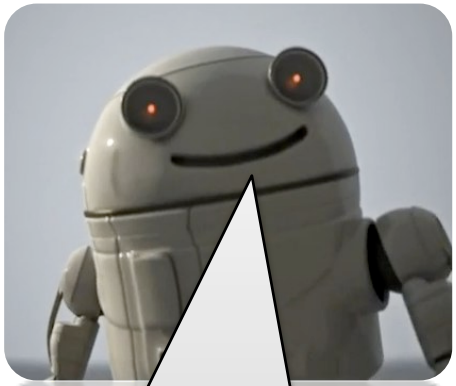


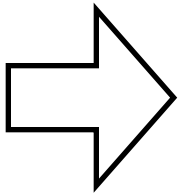


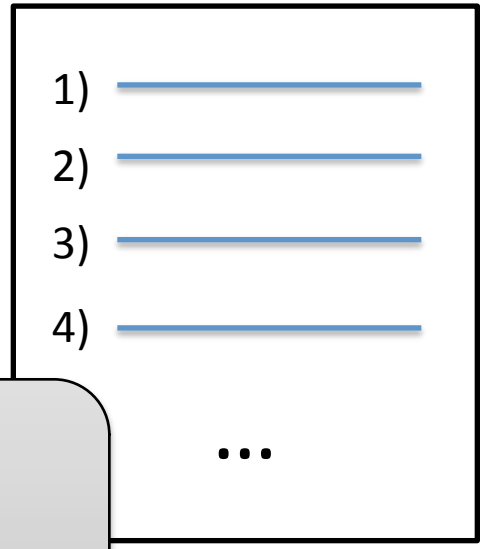




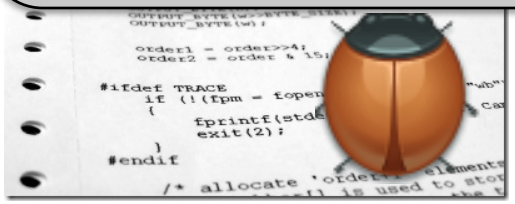
Automated Debugging (rank based)



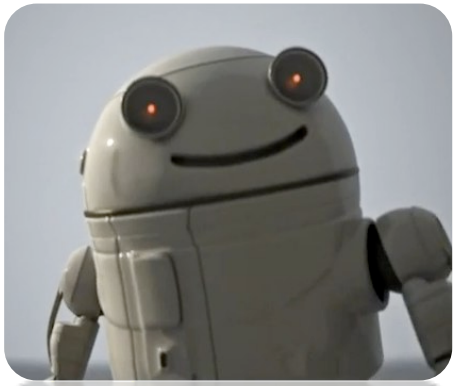




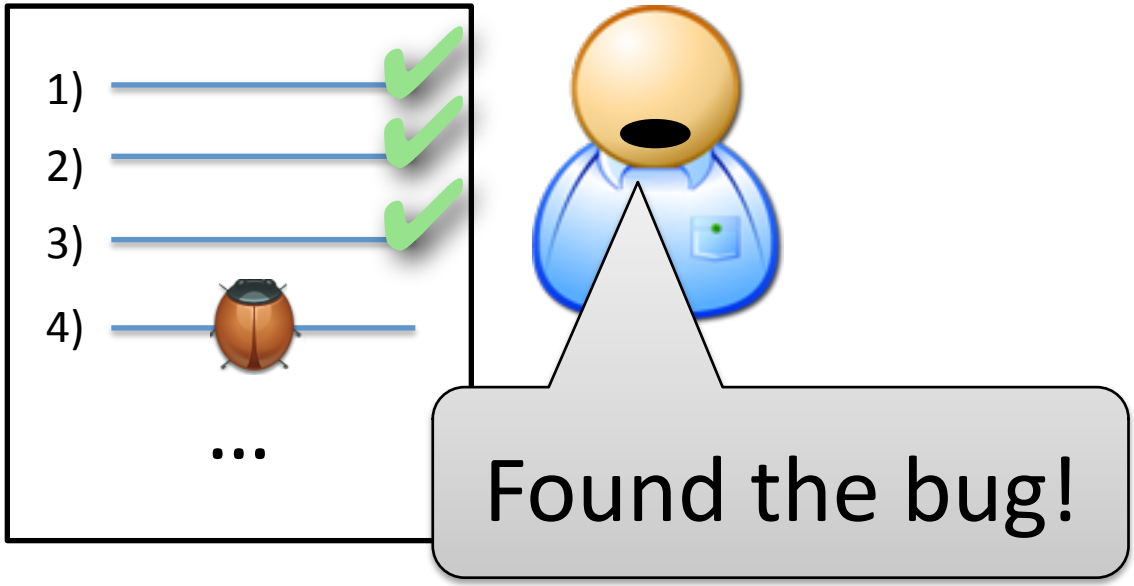
Here is a list of places to check out



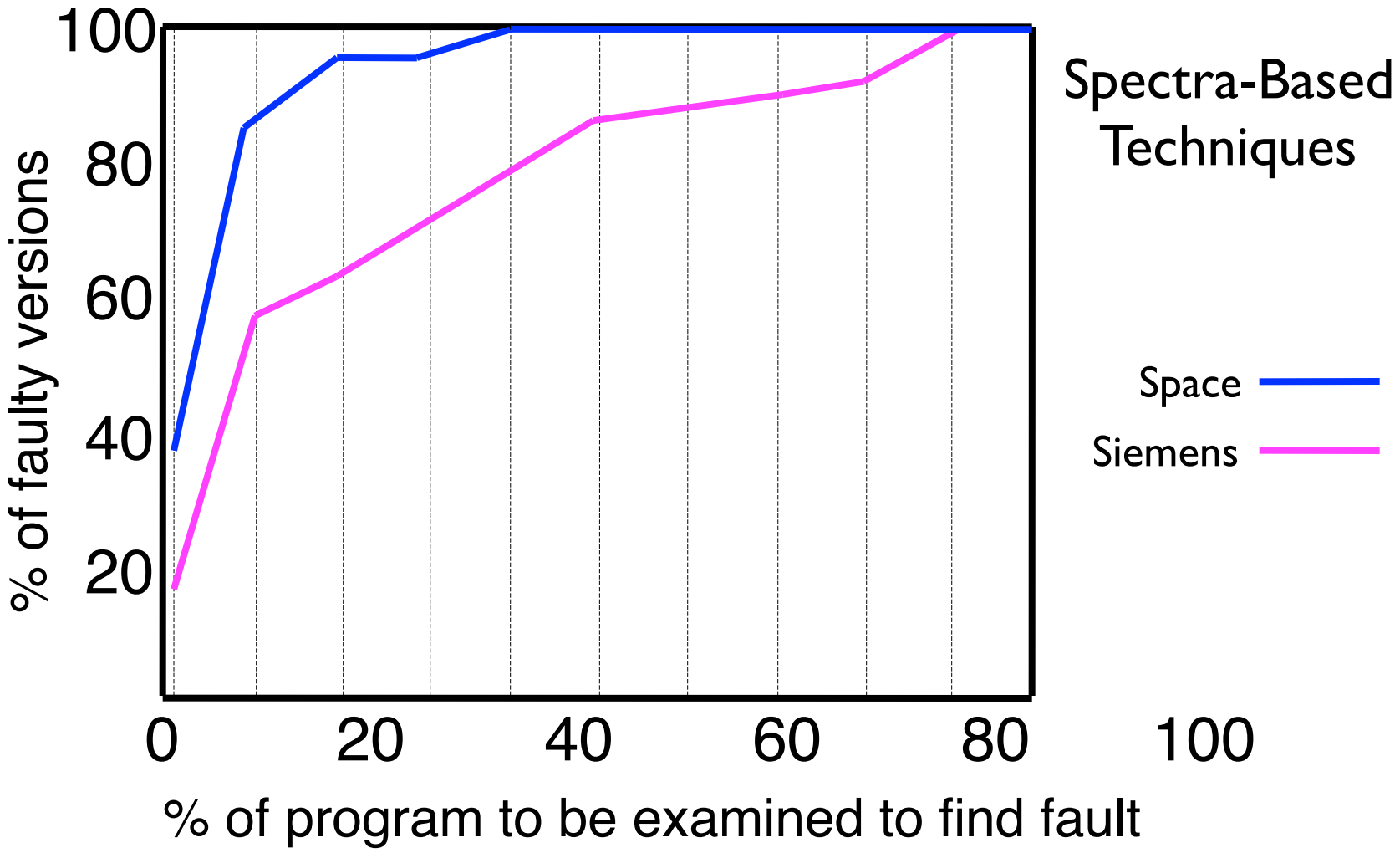
Automated Debugging
Conceptual Model



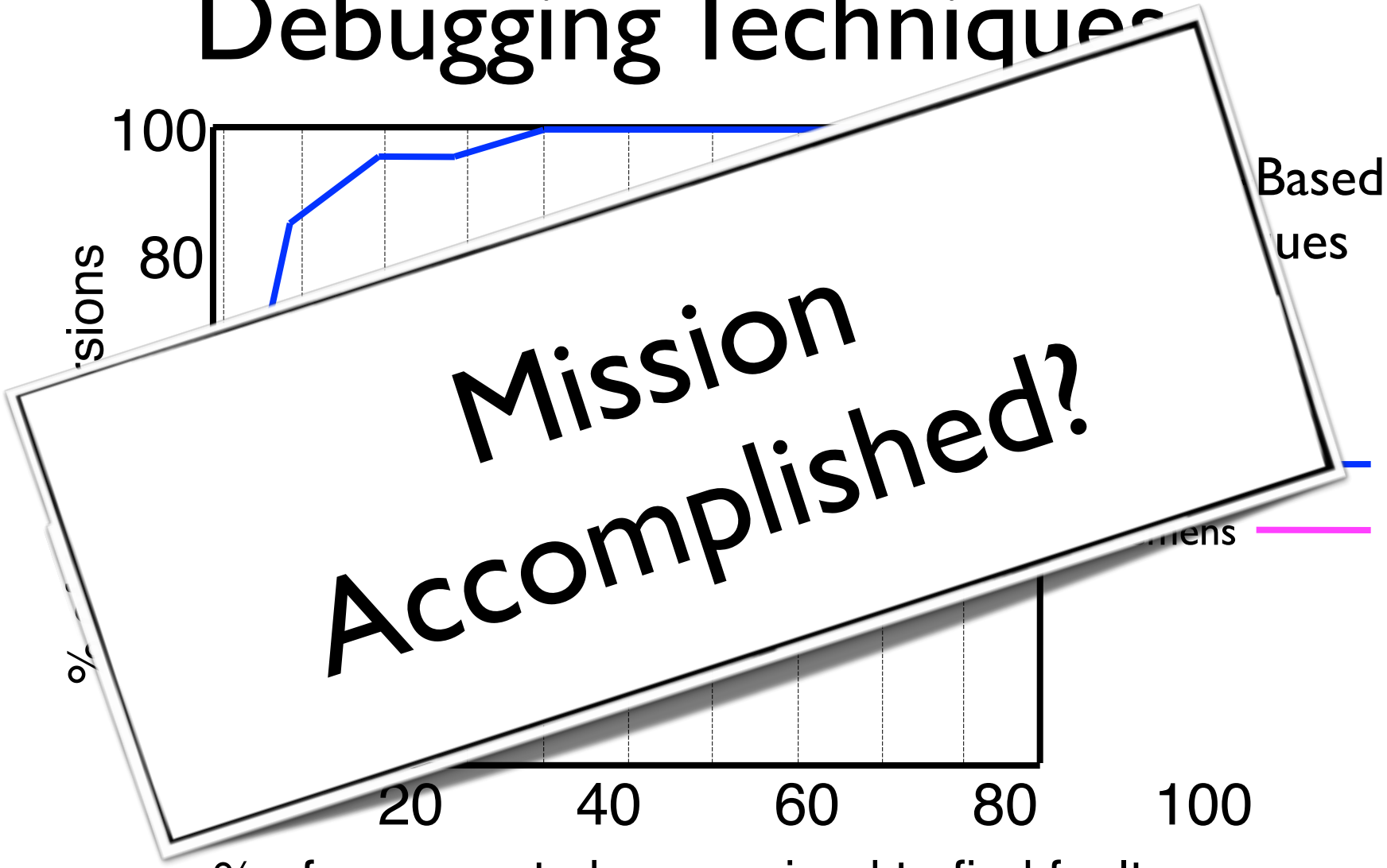
Automated Debugging
Conceptual Model



Performance of Automated Debugging Techniques



Performance of Automated Debugging Techniques



Mission Accomplished?

Assumption #1: Locating a bug in **10% of the code** is a great result

100 LOC ➡ 10 LOC

10,000 LOC ➡ 1,000 LOC

100,000 LOC ➡ 10,000 LOC

Assumption #2: Programmers exhibit **perfect bug understanding**

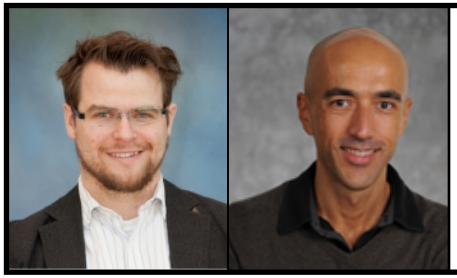
Assumption #3: Programmers inspect a list **linearly and exhaustively**

Good for comparison, but is it realistic?

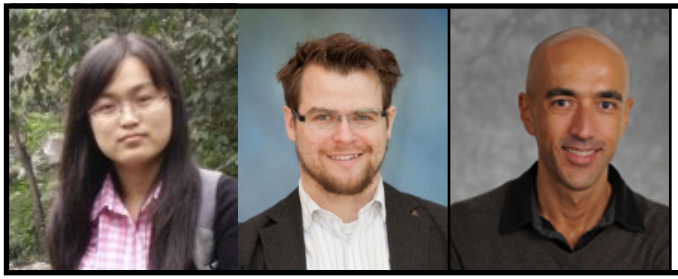
Assumption #3: Programmers inspect a list **linearly and exhaustively**

Assumption #3: Programmers inspect a list **linearly and exhaustively**

Are we headed in the right direction?



"*Are Automated Debugging Techniques Actually Helping Programmers?*" ISSTA 2011
C. Parnin and A. Orso

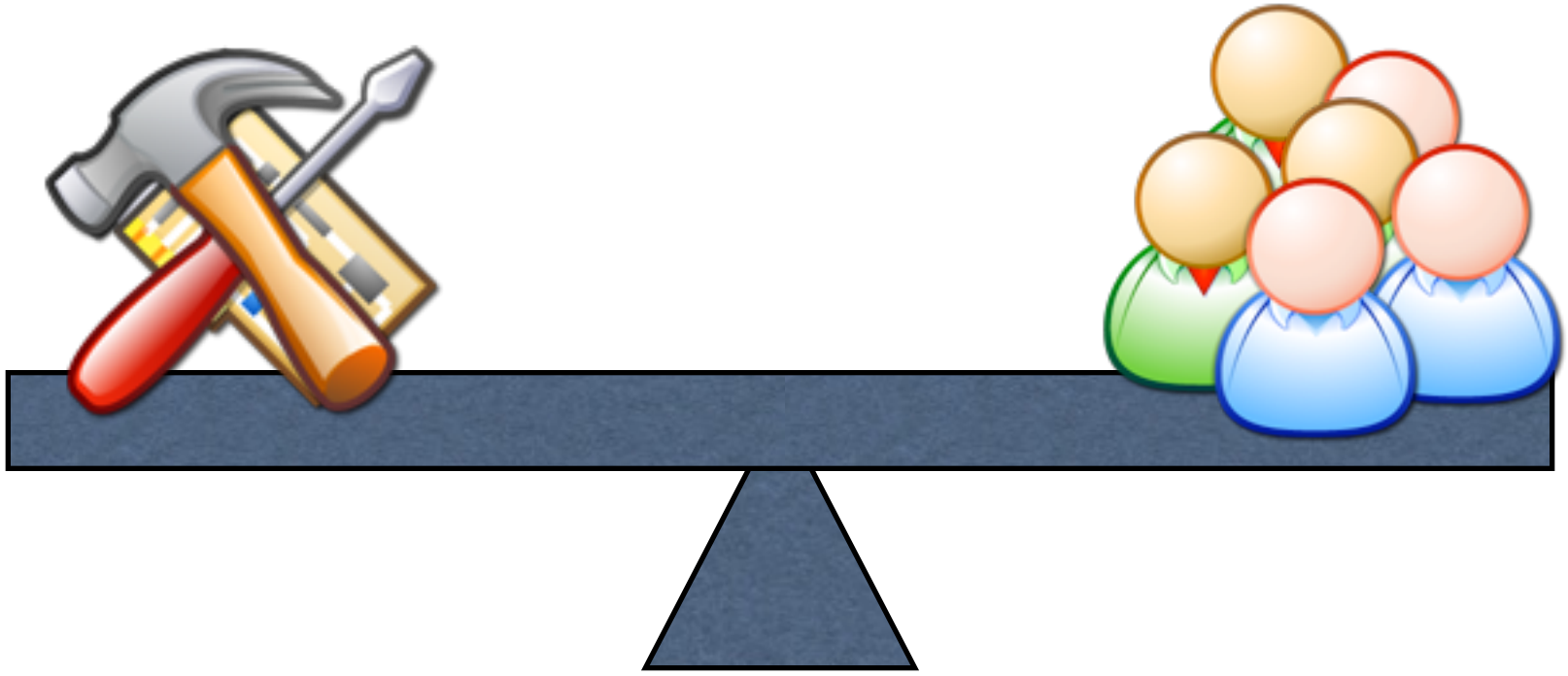


"*Evaluating the Usefulness of IR-Based Fault Localization Techniques*" ISSTA 2015
Q. Wang, C. Parnin, and A. Orso

What do we know about automated debugging?

Studies on tools

Human studies



out

g?

studies

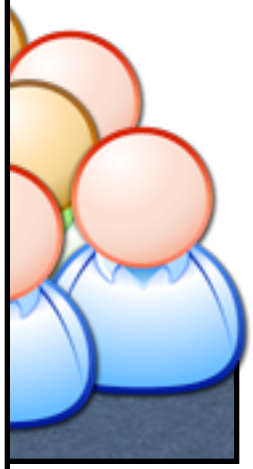
Let's see…
Over 50 years of research
on automated debugging.

2001. Statistical Debugging

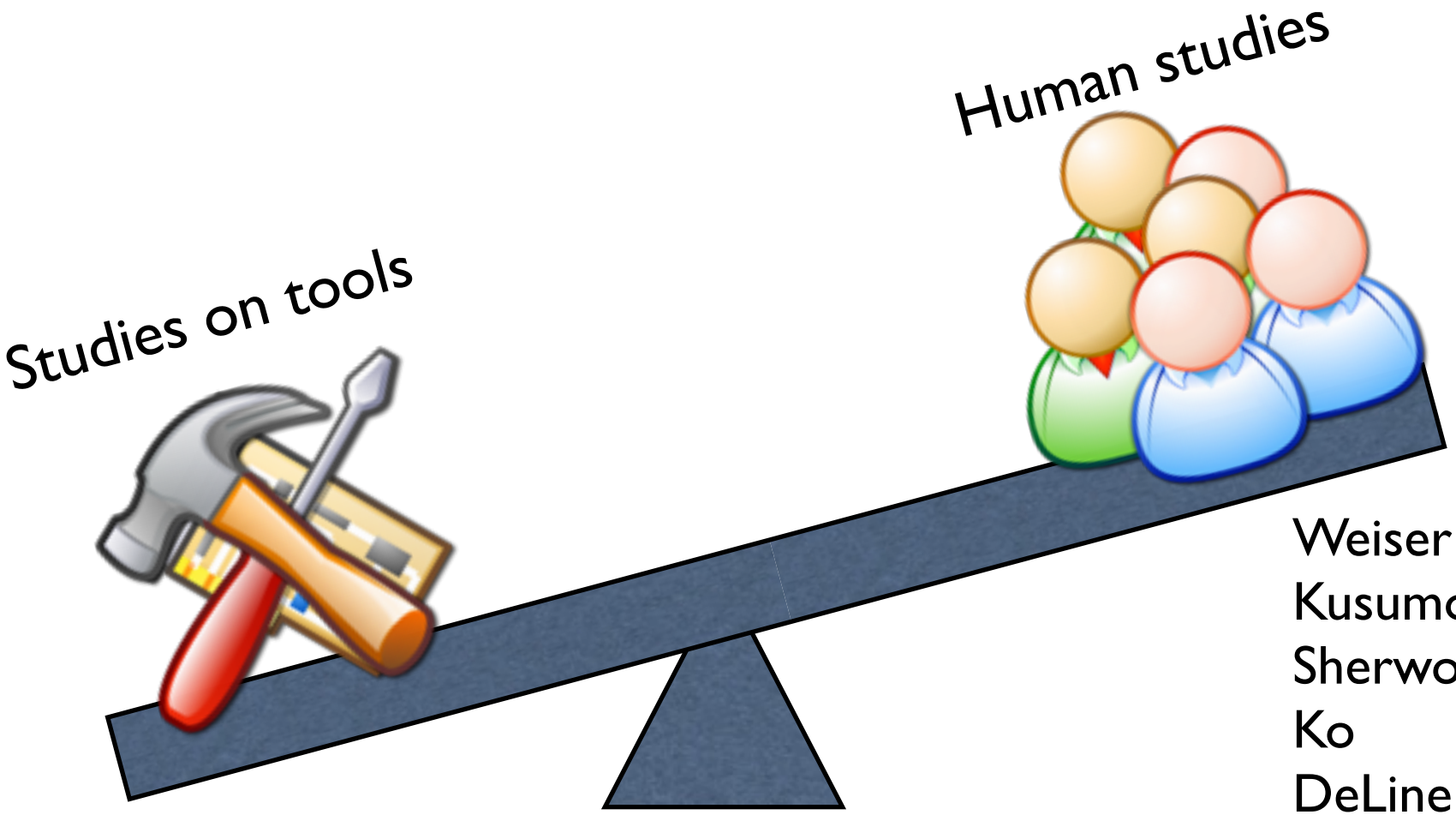
1999. Delta Debugging

1981. Weiser. Program Slicing

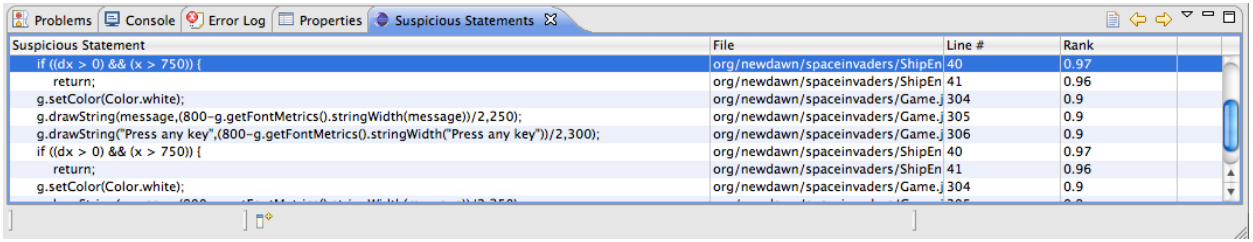
1962. Symbolic Debugging (UNIVAC FLIT)



What do we know about automated debugging?



Are these Techniques and Tools Actually Helping Programmers?



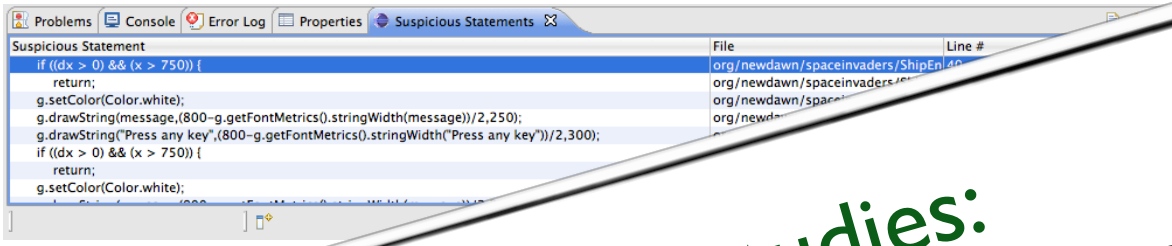
RQ1: Do programmers who use automated debugging tools locate bugs faster than programmers who do not use such tools?

RQ2: Is the effectiveness of debugging with automated tools affected by the faulty statement's rank?

RQ3: Do developers navigate a list of statements ranked by suspiciousness in the order provided?

RQ4: Does perfect bug understanding exist?

Are these Techniques and Tools Actually Helping Programmers?



User studies:

Spectra based fault localization

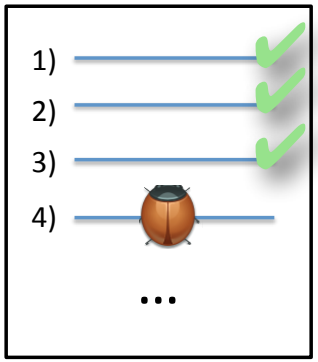
~~IR-based fault localization~~

ols affected by the

list of statements ranked by suspiciousness in

R oes perfect bug understanding exist?

Experimental Protocol: Setup



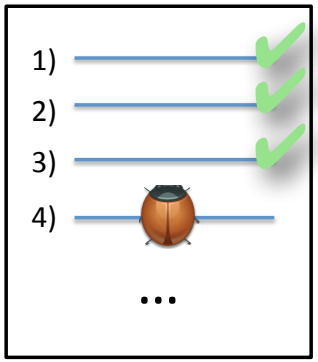
Participants:

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

Tools

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

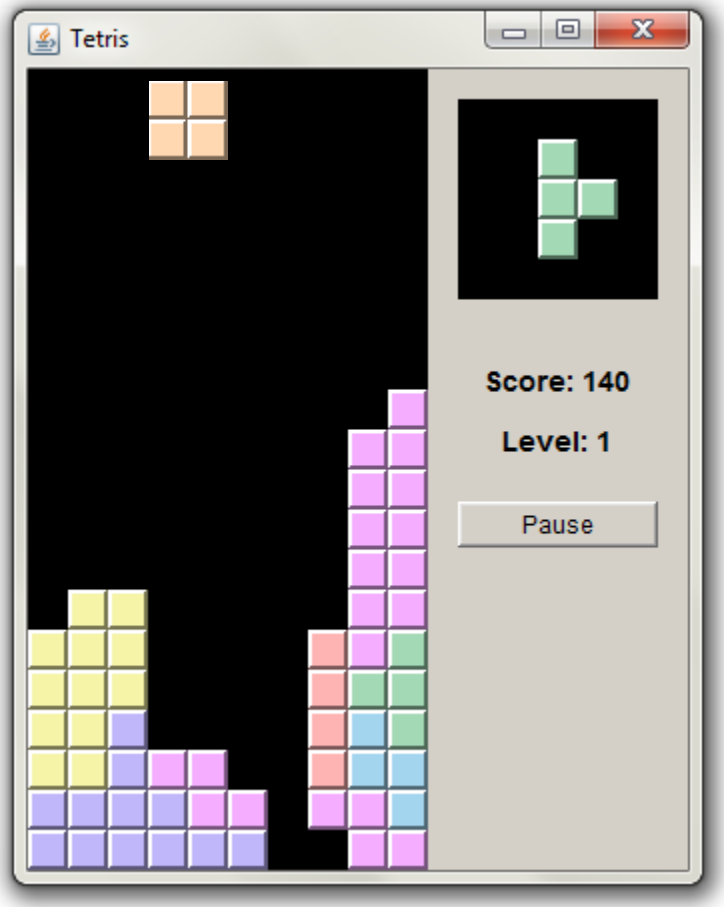
Experimental Protocol: Setup



Software subjects:

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

Tetris Bug



(Easier)

NanoXML Bug

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

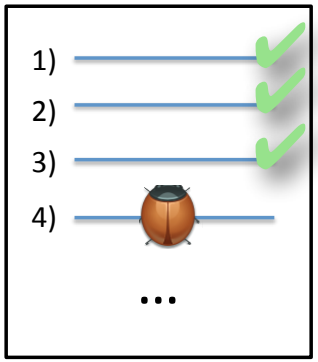
```
<Foo a=”test”>
  <ns:Bar>
    <Blah x=”1” ns:x=”2”/>
  </ns:Bar>
</Foo>
```

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



Tasks:

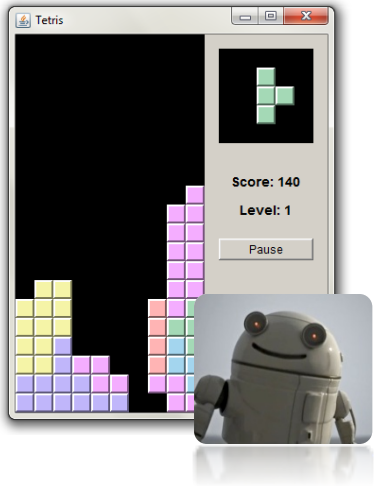
- Fault in Tetris
- Fault in NanoXML
- 30 minutes per task
- Questionnaire at the end

Experimental Protocol: Studies and Groups

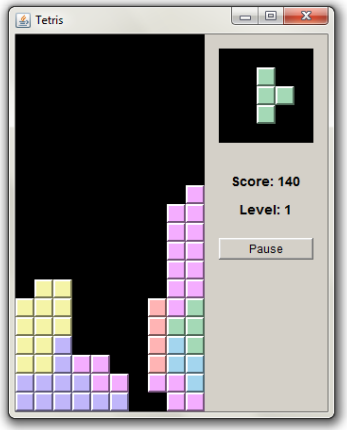
Experimental Protocol: Studies and Groups

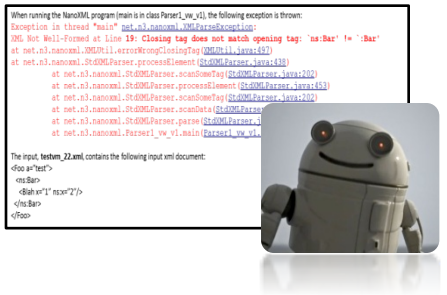
Part 1

A



B

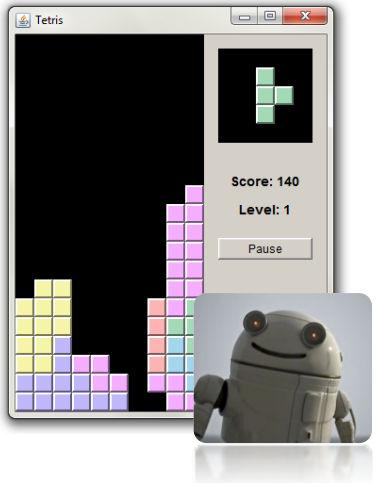


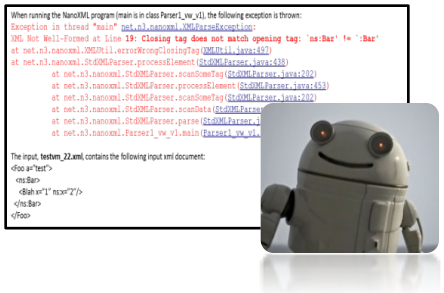


Experimental Protocol: Studies and Groups

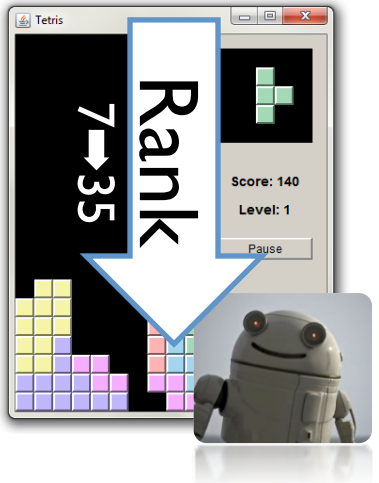
Part 2

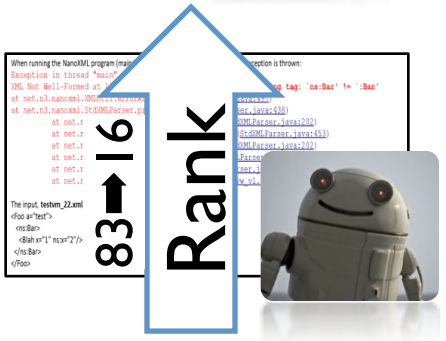
C





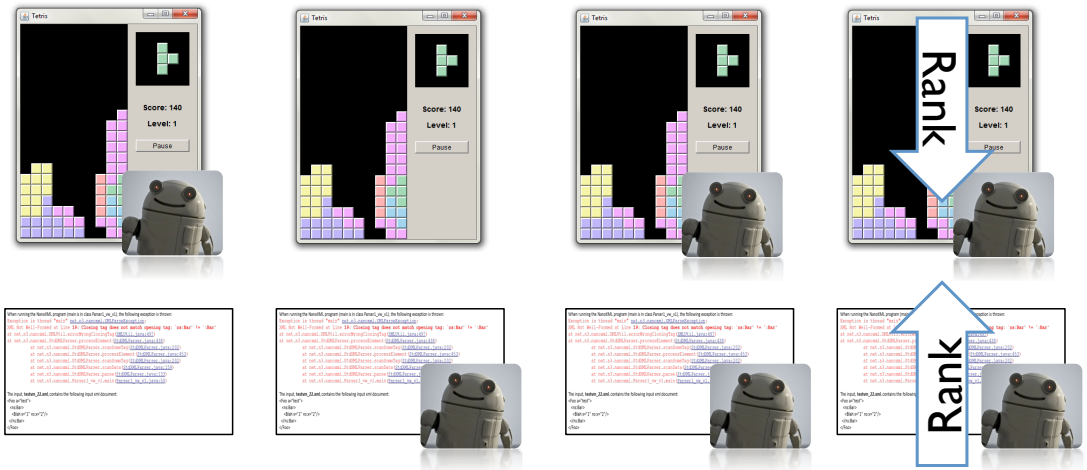
D





Study Results

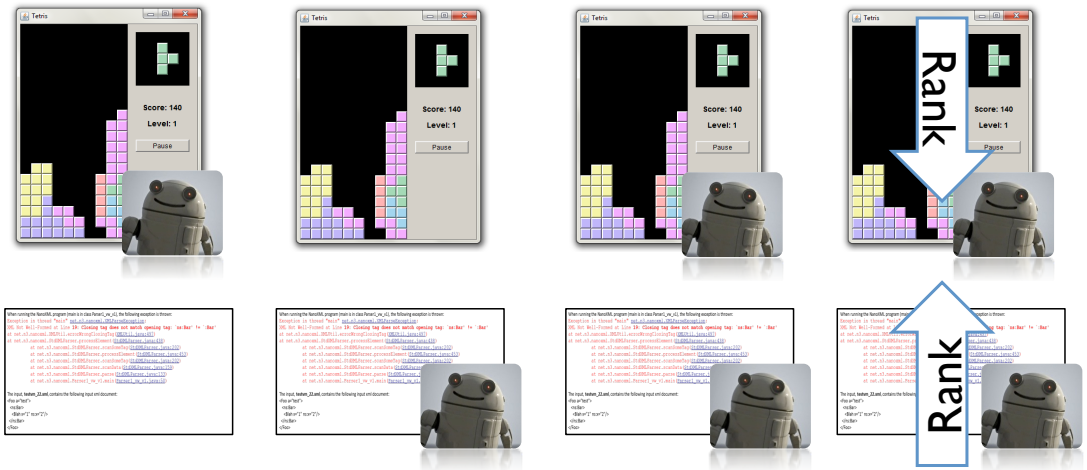
A B C D



	Tetris	NanoXML
A		
B		
C		
D		

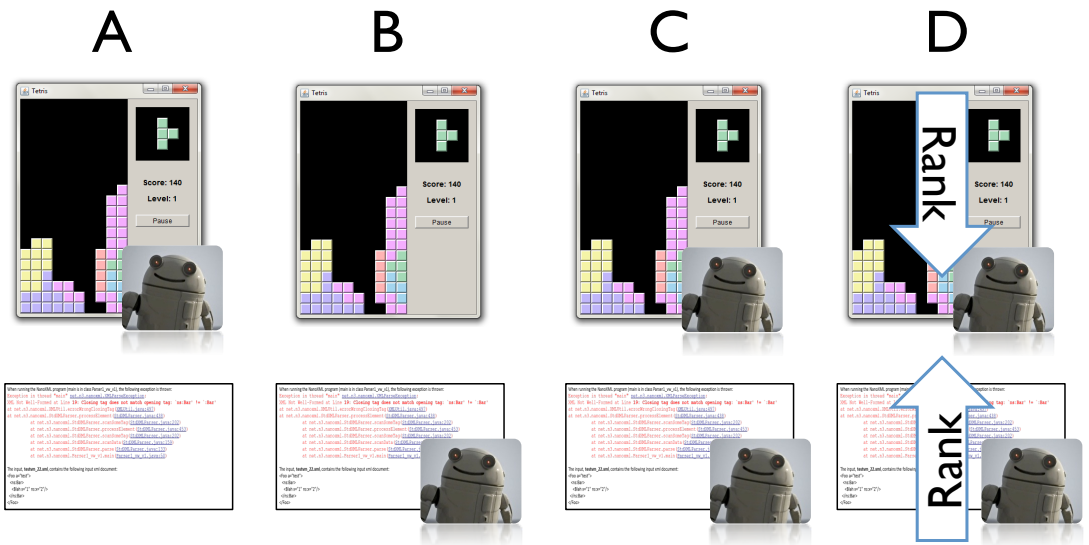
Study Results

A B C D



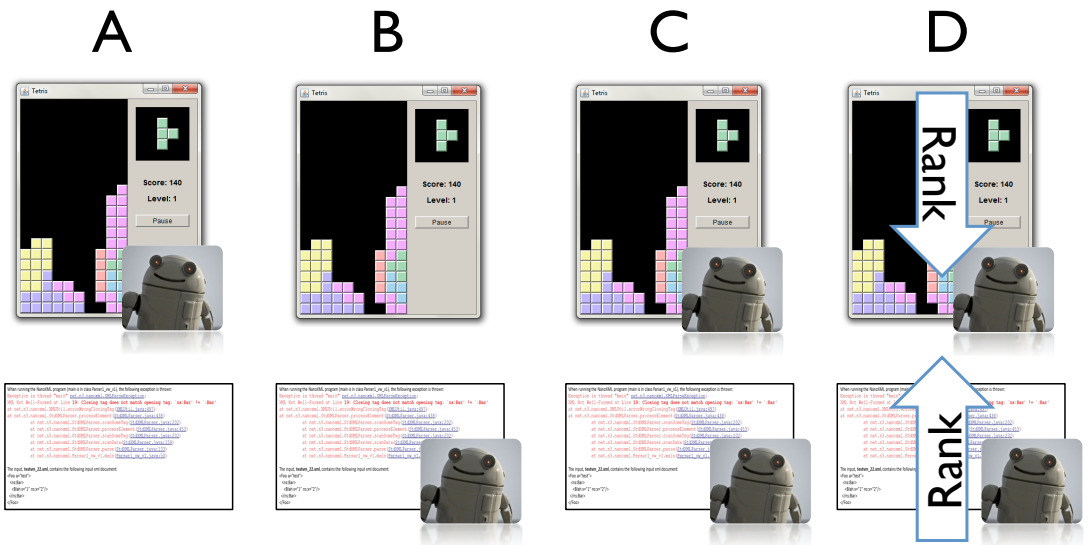
	Tetris	NanoXML
A	**Not** significantly different	
B		
C		
D		

Study Results



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

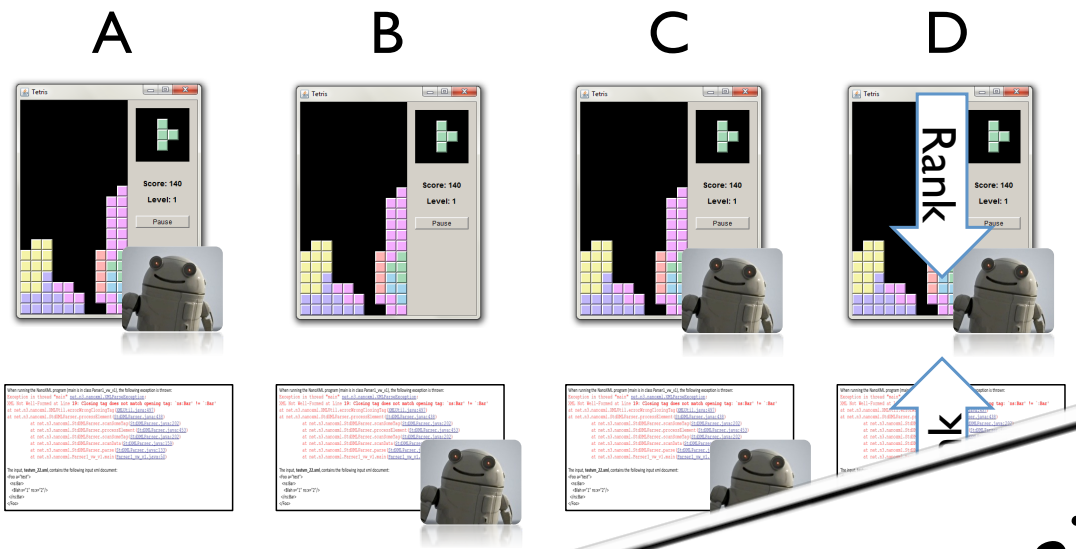
Study Results



Stratifying participants

	Tetris	NanoXML
A	Significantly different for high performers	**Not** significantly different
B		
C	**Not** significantly different	**Not** significantly different
D		

Study Results



Analysis of results and questionnaires...

Stratifyi…		
Pa…	…rmers	significantly different
C	**Not** significantly different	**Not** significantly different
D		

Findings

RQ1: Do programmers who use automated debugging tools locate bugs faster than programmers who do not use such tools?

Experts are faster when using the tool ➡ Yes (with caveats)

RQ2: Is the effectiveness of debugging with automated tools affected by the faulty statement's rank

Changes in rank have no significant effects ➡ No

RQ3: Do developers navigate a list of statements ranked by suspiciousness in the order provided?

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

RQ4: Does perfect bug understanding exist?

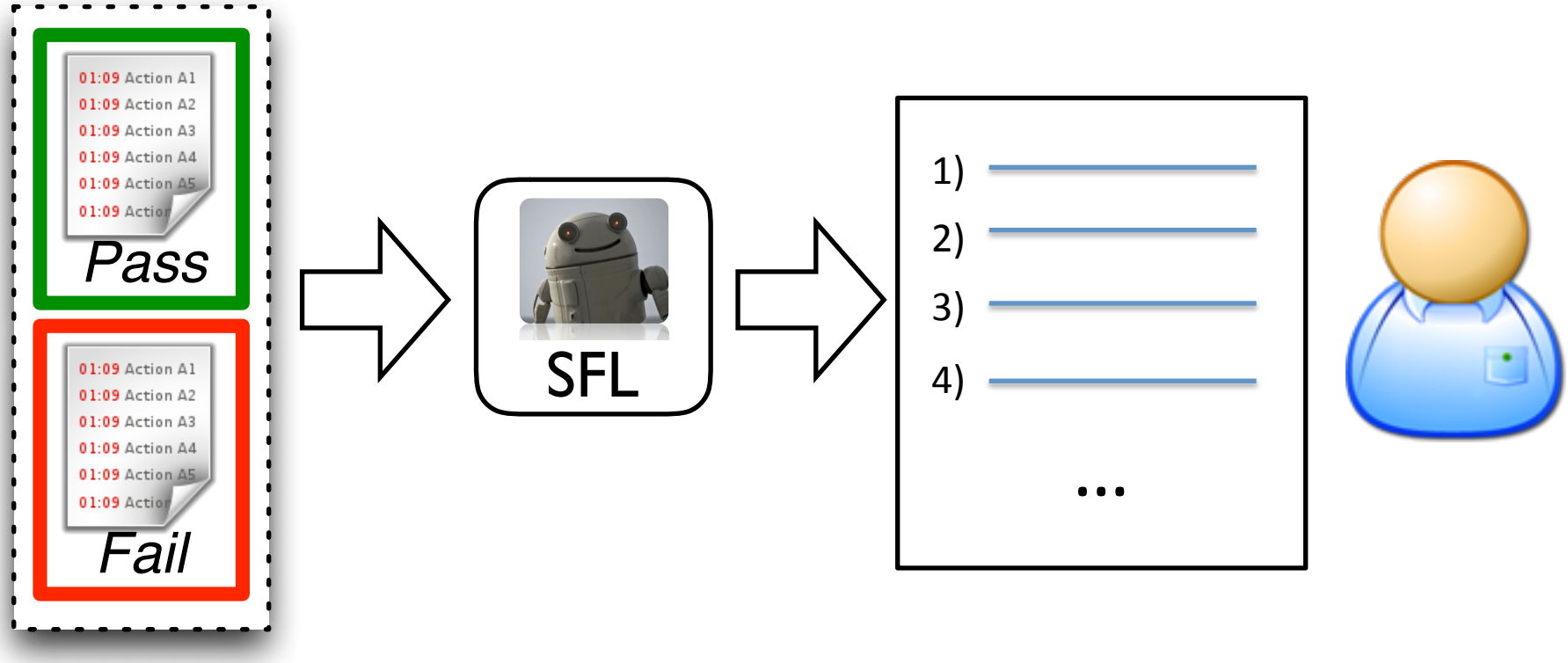
Perfect bug understanding is generally not a realistic assumption

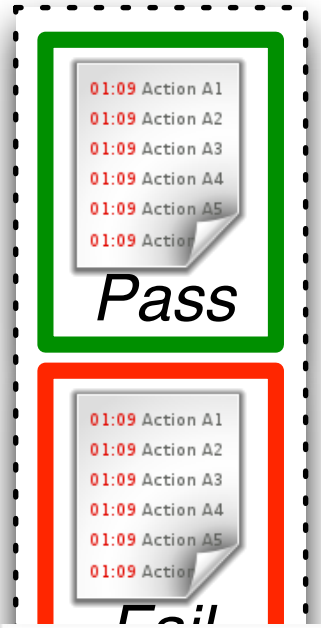
Future

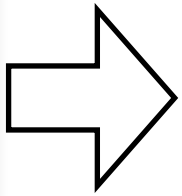
Where Shall We Go Next?

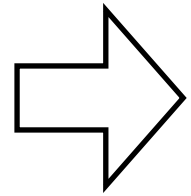
Feedback-based Debugging (Humans in the Loop)

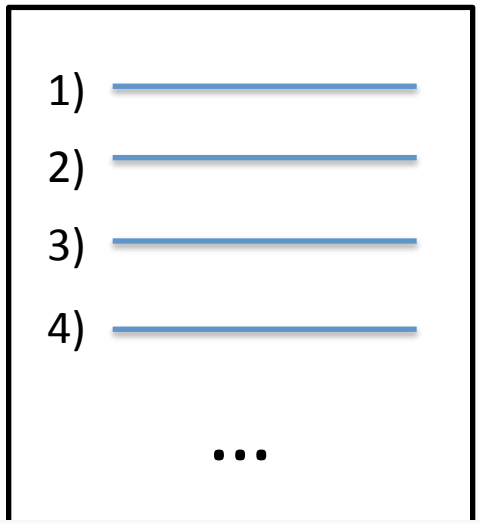
Intuition: we should amplify, rather than replace human skills


01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Pass
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Fail
SFL
1)
2)
3)
4)
...



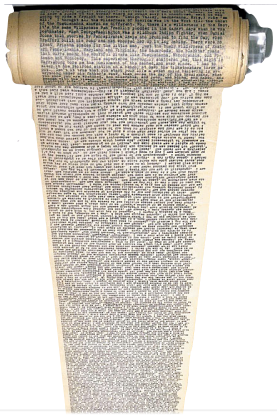


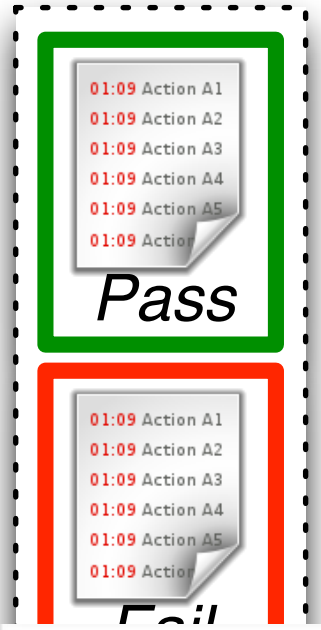


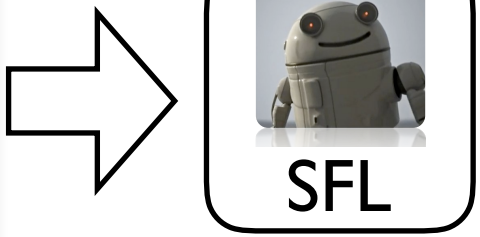


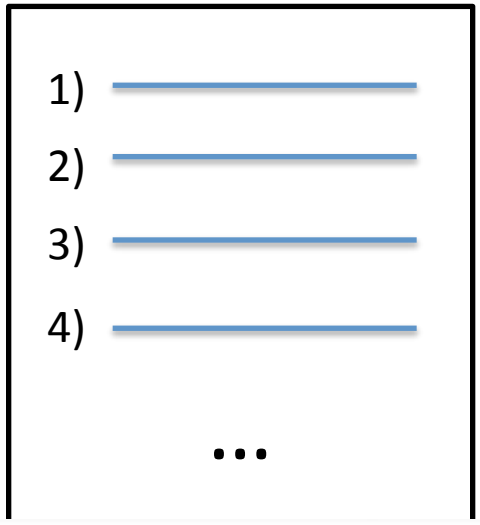
Assumption: Programmers exhibit **perfect bug understanding**

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



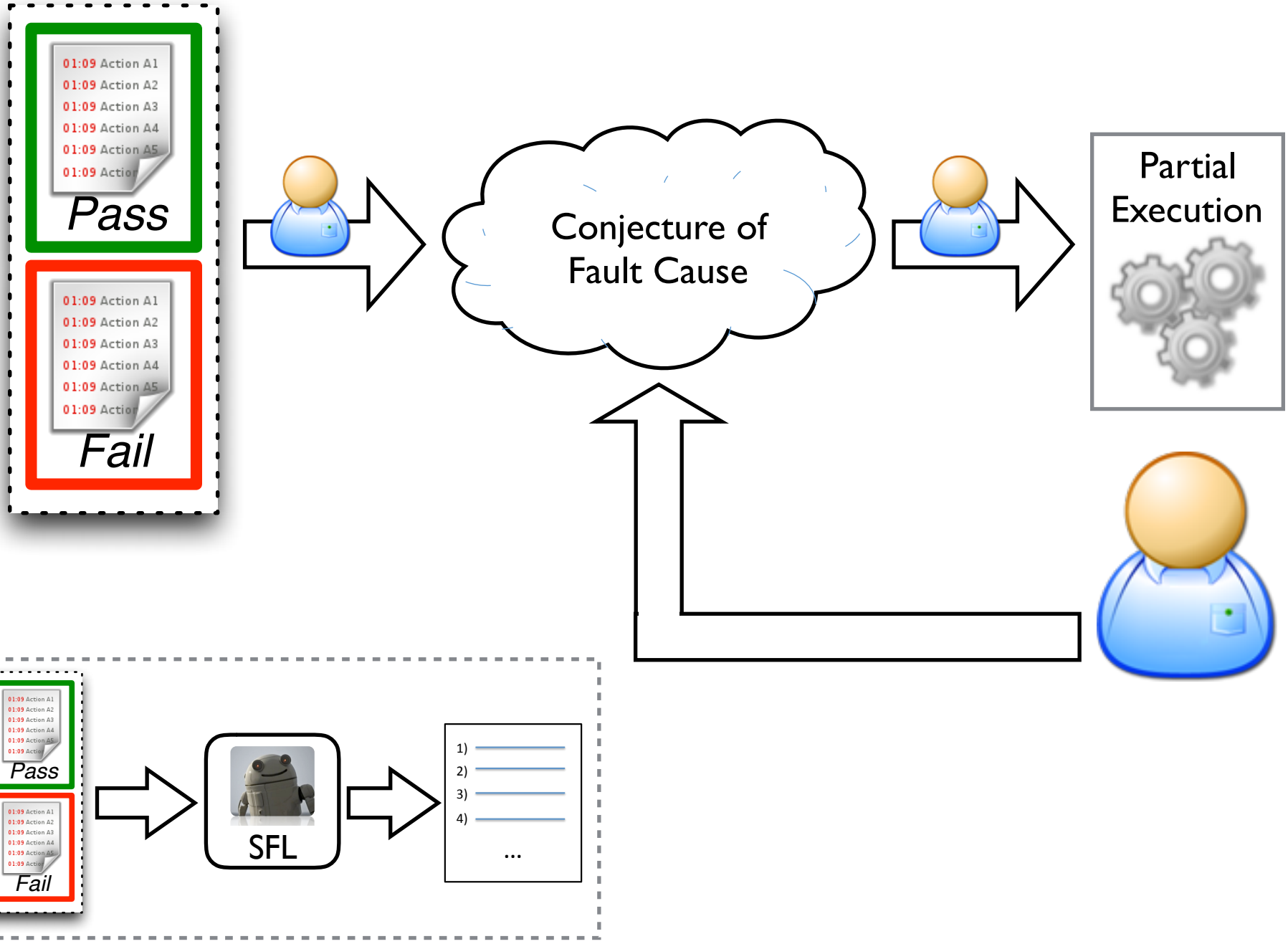


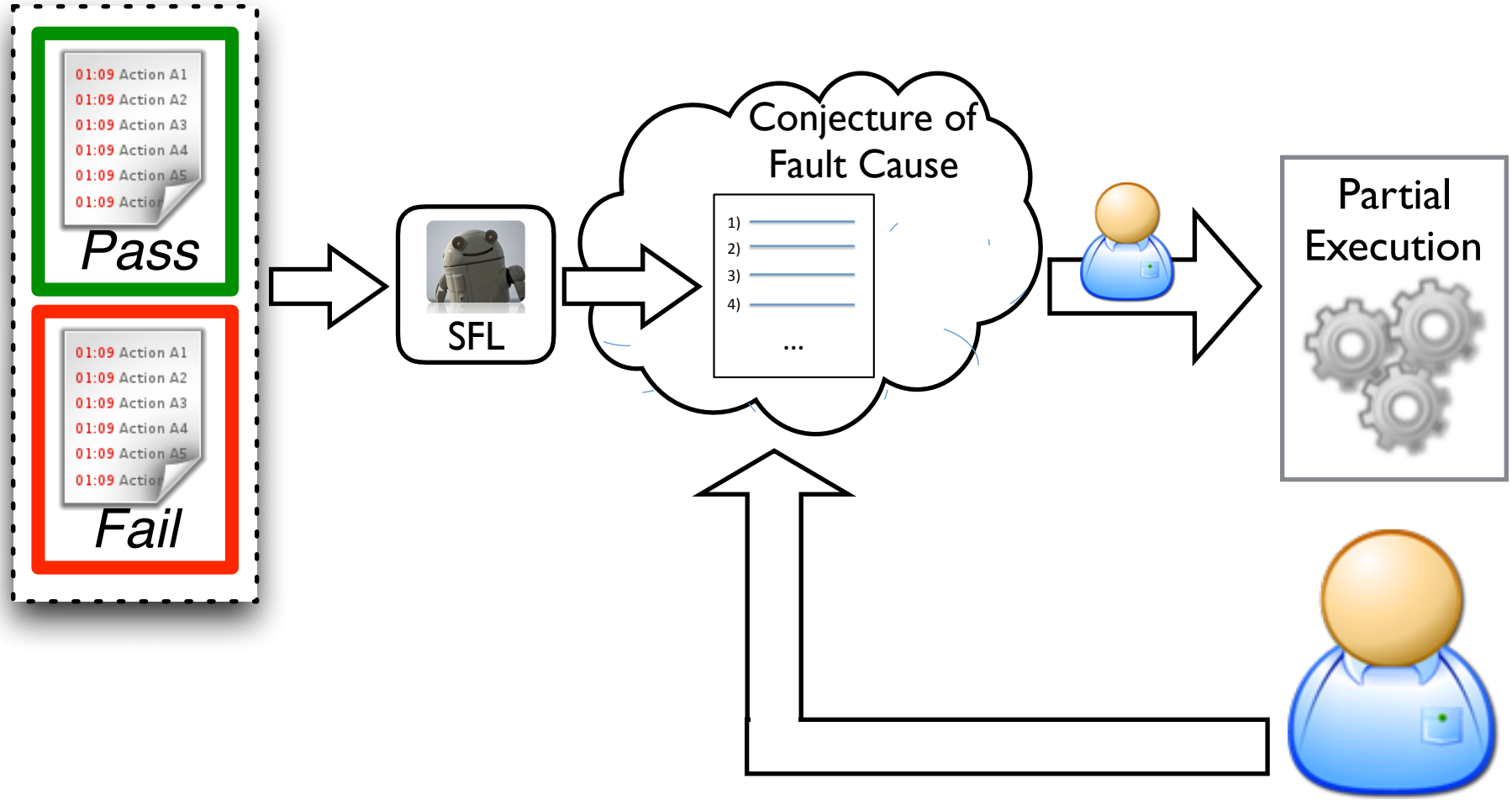


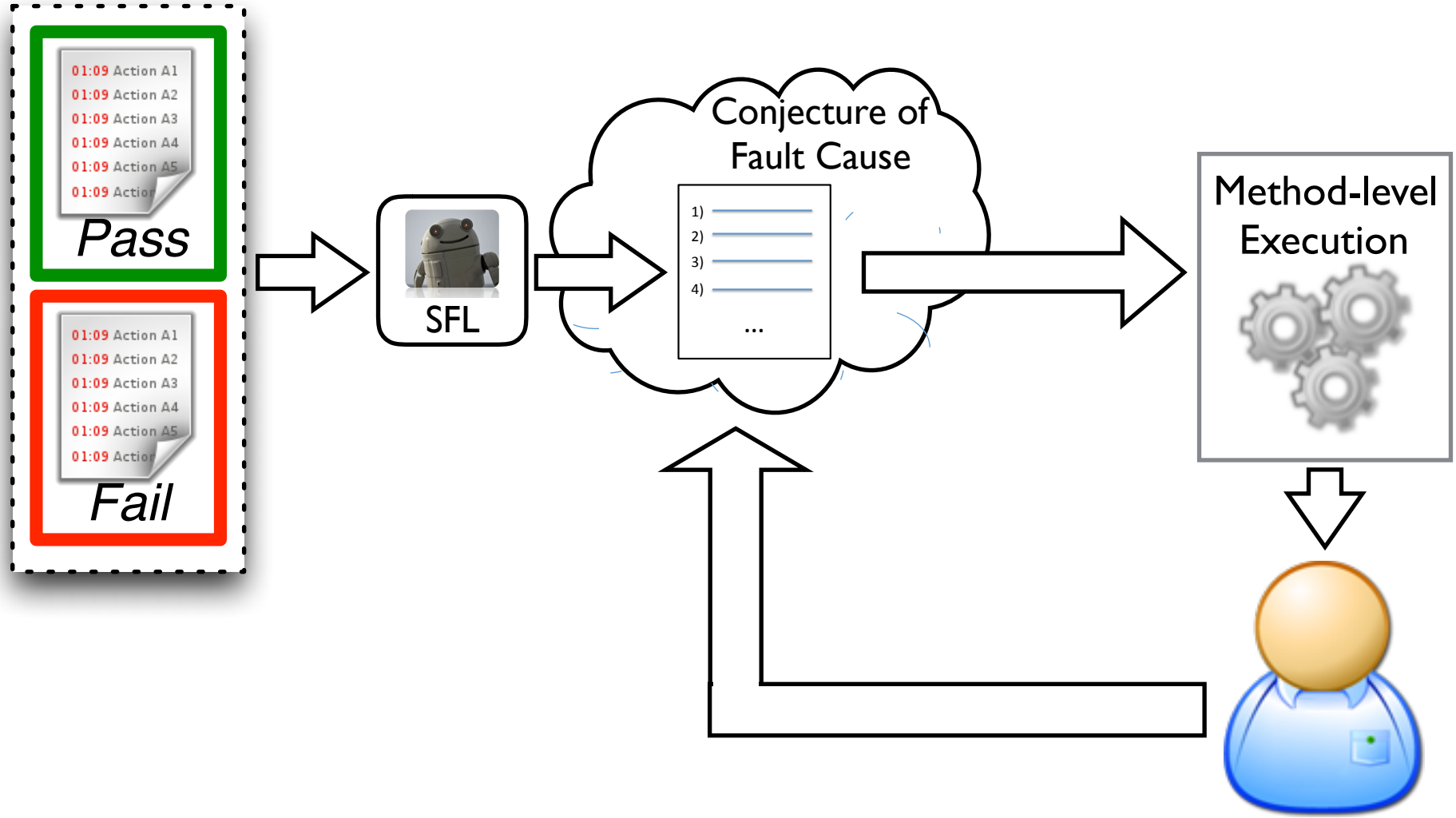


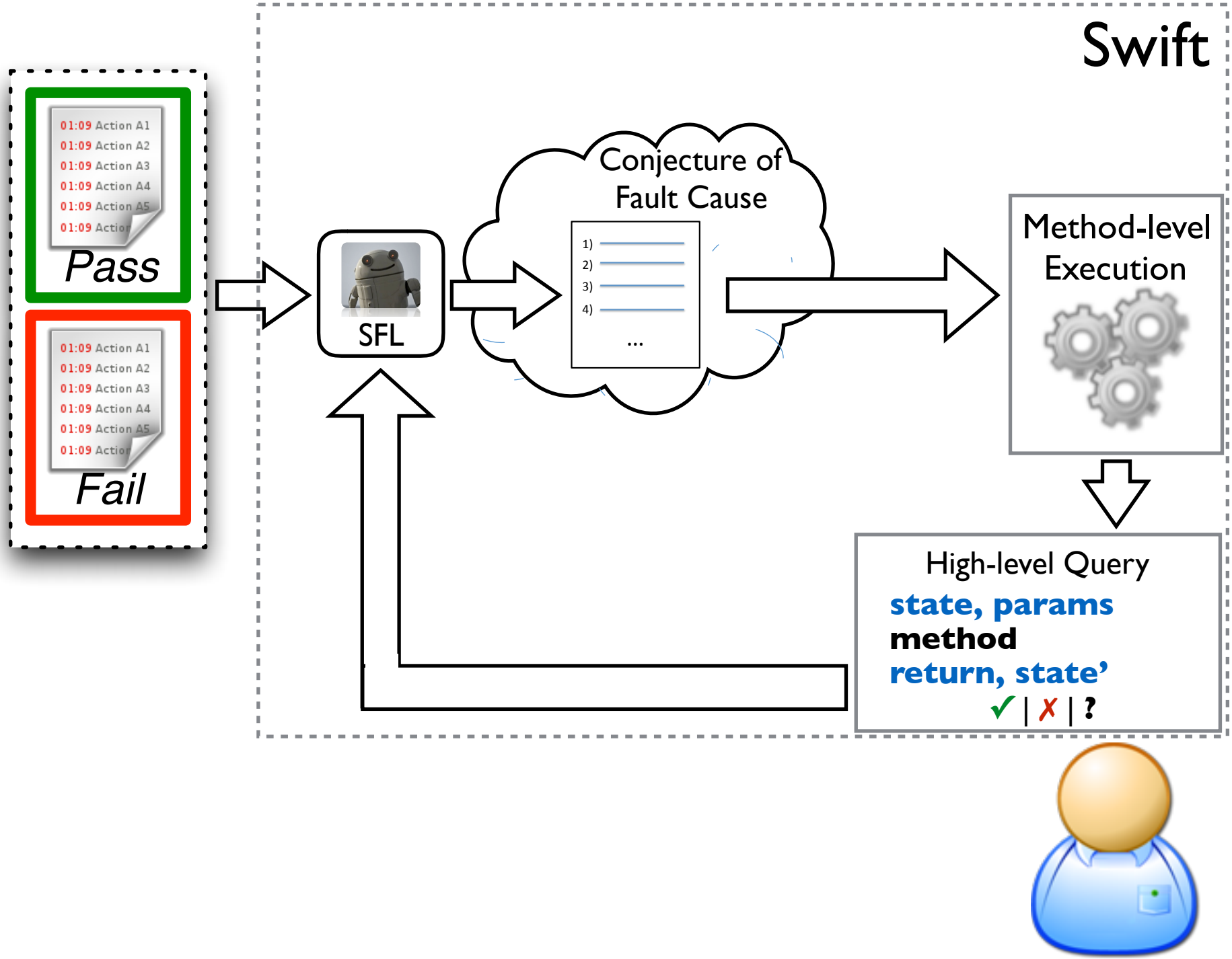
Assumption: Programmers exhibit **perfect bug understanding**

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


01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Pass
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Fail
Conjecture of Fault Cause
Partial Execution
Pass
Fail
SFL
1)
2)
3)
4)
...


01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Pass
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Fail
SFL
Conjecture of Fault Cause
1)
2)
3)
4)
...
Partial Execution


01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Pass
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Fail
SFL
Conjecture of Fault Cause
1)
2)
3)
4)
...
Method-level Execution


Swift
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Pass
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Fail
SFL
Conjecture of Fault Cause
1)
2)
3)
4)
...
Method-level Execution
High-level Query
state, params
method
return, state'
✓ | ✗ | ?

Example

```
1  public class BoundedStack {
2   Integer[] elems; int numElems;
3
4   BoundedStack(int max) {
5     elems = new Integer[max]; }
6
7   void push(Integer k) {
8    /* check size */
9    elems[numElems++] = k; }
10
11  void pop() { --numElems; }
12
13  Integer peek() {
14   if (size() == 0)
15     return null;
16   else return elems[size()-1];}
17
18  void clear() { numElems = 0; }
19
20  int size() { return numElems; }
21  ...
22 }
```

Iteration 1

Q_1: `BoundedStack.clear()`#0 in t_2	
Input:	Output:

this: {
elems: {7, 8, null}
numElems: 2
}
	this: {
	elems: {7, 8, null}
	numElems: 0
	}

✓ ✗ ?

```
1  public class BoundedStack {
2   Integer[] elems; int numElems;
3
4   BoundedStack(int max) {
5     elems = new Integer[max]; }
6
7   void push(Integer k) {
8    /* check size */
9    elems[numElems++] = k; }
10
11  void pop() { --numElems; }
12
13  Integer peek() {
14   if (size() == 0)
15     return null;
16   else return elems[size()-1];}
17
18  void clear() { numElems = 0; }
19
20  int size() { return numElems; }
21  ...
22 }
```

Iteration 2

BoundedStack.push(int)#0 in t2	
Input:	Output:

k: 7
this: {
elems: {null, null, null}
numElems: 0
}
	this: {
	elems: {7, null, null}
	numElems: 1
	}


```
1  public class BoundedStack {
2   Integer[] elems; int numElems;
3
4   BoundedStack(int max) {
5     elems = new Integer[max]; }
6
7   void push(Integer k) {
8    /* check size */
9    elems[numElems++] = k; }
10
11  void pop() { --numElems; }
12
13  Integer peek() {
14   if (size() == 0)
15     return null;
16   else return elems[size()-1];}
17
18  void clear() { numElems = 0; }
19
20  int size() { return numElems; }
21  ...
22 }
```

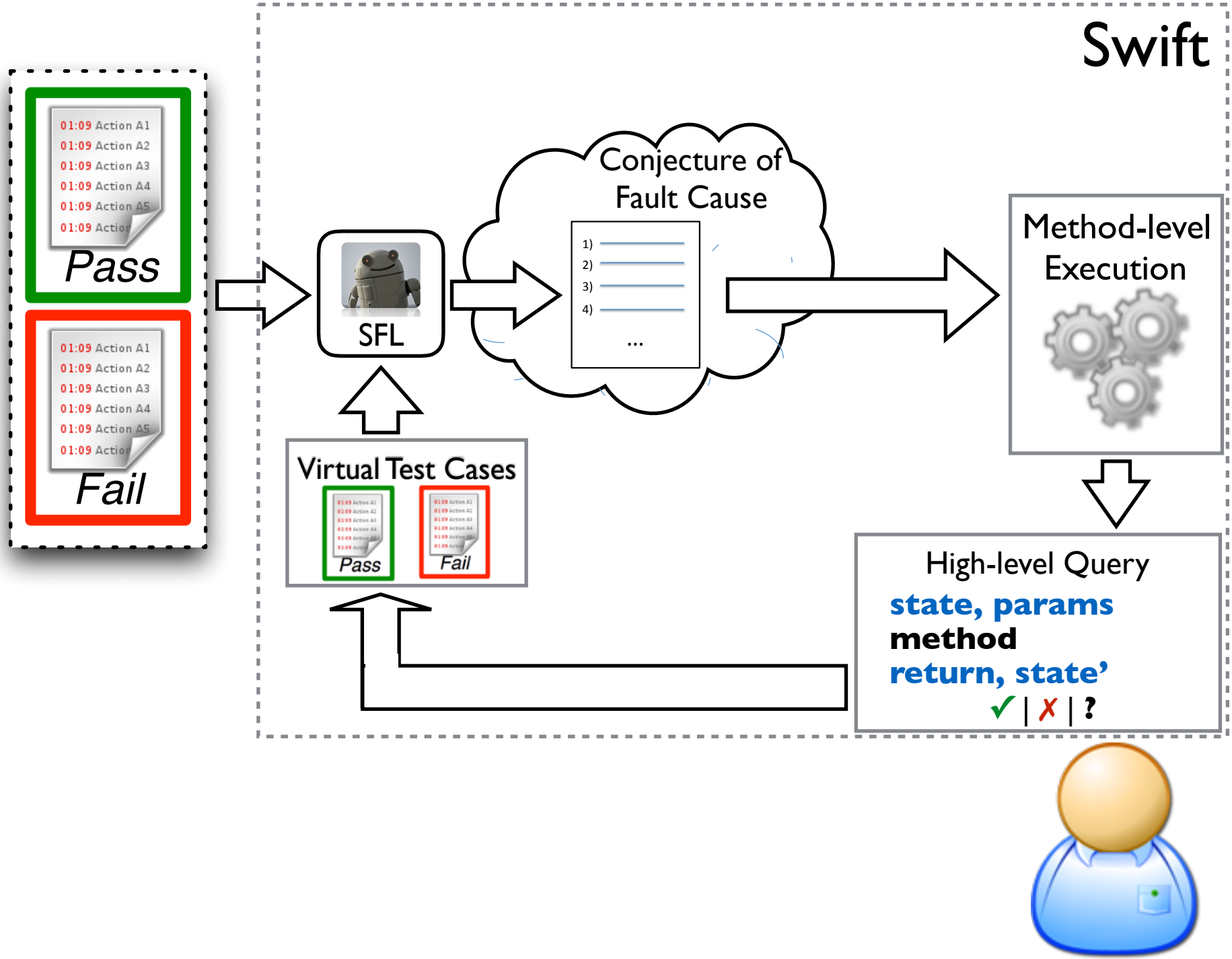
Iteration 3

Q_3: `BoundedStack.pop()`#0 in t_2	
Input:	Output:

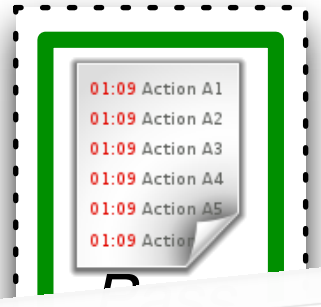
this: {
elems: {7, 8, null}
numElems: 0
}
	this: {
	elems: {7, 8, null}
	numElems: -1
	}

✓ ✗ ?

```
1  public class BoundedStack {
2   Integer[] elems; int numElems;
3
4   BoundedStack(int max) {
5     elems = new Integer[max]; }
6
7   void push(Integer k) {
8    /* check size */
9    elems[numElems++] = k; }
10
11  void pop() { --numElems; }
12
13  Integer peek() {
14   if (size() == 0)
15     return null;
16   else return elems[size()-1];}
17
18  void clear() { numElems = 0; }
19
20  int size() { return numElems; }
21  ...
22 }
```


Swift
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Pass
01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Action
Fail
SFL
Conjecture of Fault Cause
1)
2)
3)
4)
...
Method-level Execution
Virtual Test Cases
Pass
Fail
High-level Query
state, params
method
return, state'
✓ | ✗ | ?

Swift



Conjecture of
Fault Cau

evel
on

eturn, state'
✓ | ✗ | ?

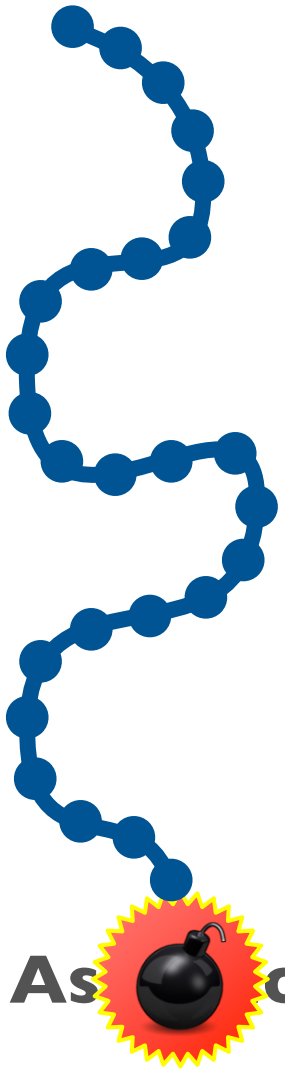
Preliminary empirical results:

- 20 faults from 3 open-source projects
- Average ranking: 66.3
- Average # of queries: 4.3

Formula-based Debugging (AKA Failure Explanation)

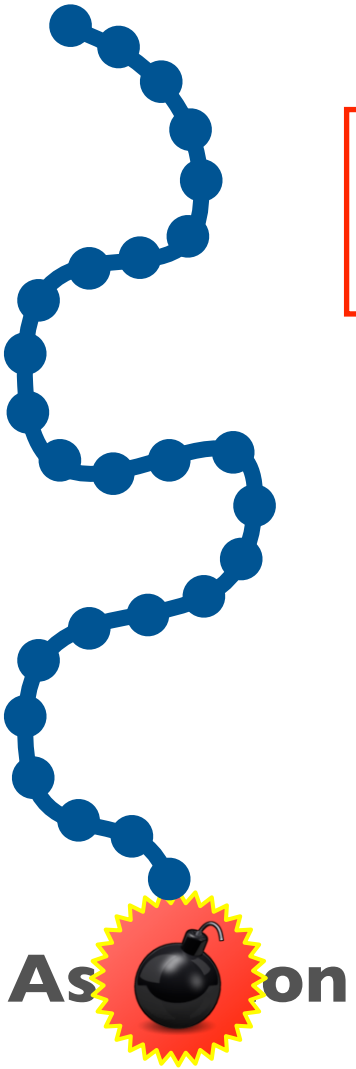
- **Intuition**: executions can be expressed as formulas that we can reason about

Input



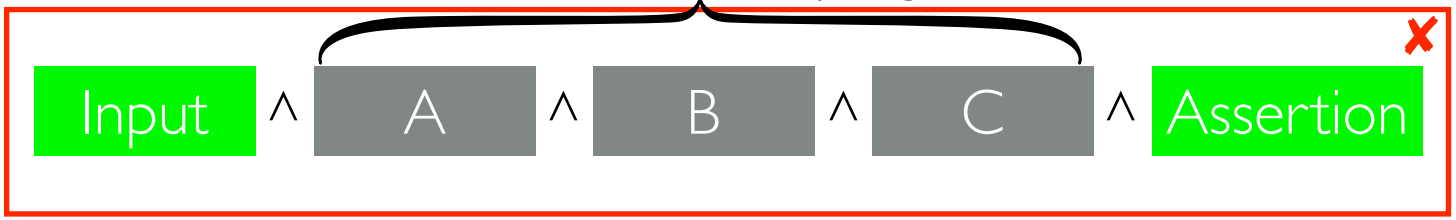
As on

Input

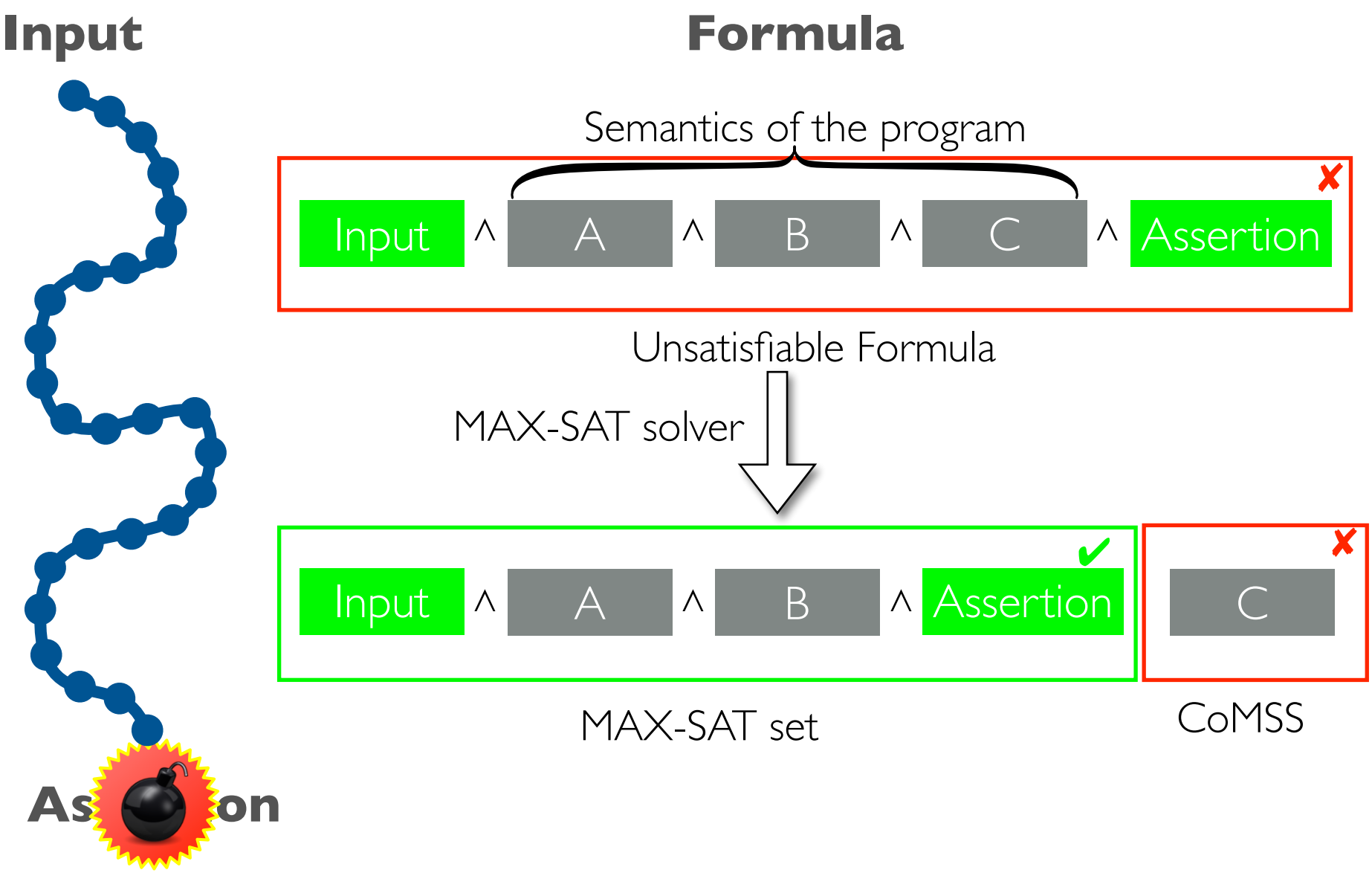


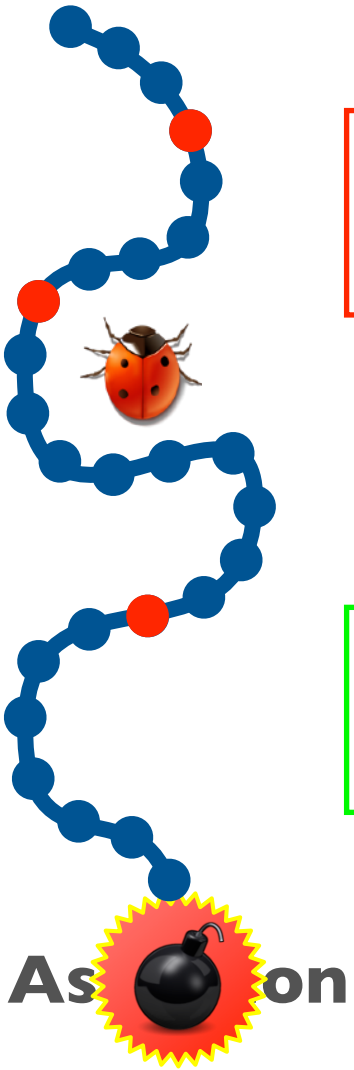
Formula

Semantics of the program



Unsatisfiable Formula


Input
Formula
Semantics of the program
Input
A
B
C
Assertion
Unsatisfiable Formula
MAX-SAT solver
Input
A
B
Assertion
C
MAX-SAT set
CoMSS



Formula

Semantics of the program

Input ∧ A ∧ B ∧ C ∧ Assertion ✘

Unsatisfiable Formula

MAX-SAT solver

Input ∧ A ∧ B ∧ Assertion ✔

C ✘

MAX-SAT set

CoMSS

Formula-based Debugging (AKA Failure Explanation)

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

Formula-based Debugging (AKA Failure Explanation)

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

Formula-based Debugging (AKA Failure Explanation)

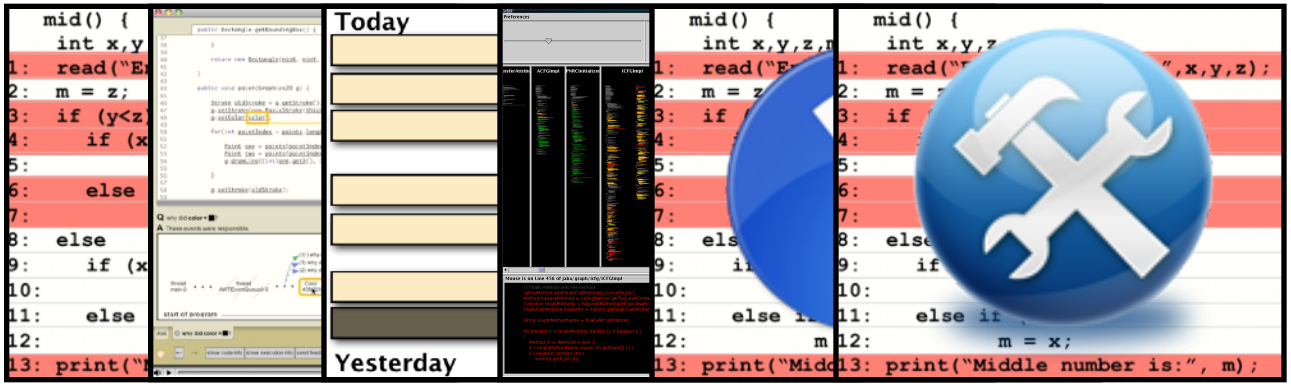
- **Intuition**: executions can be expressed as formulas that we can reason about
- Bug Assist
- Error invariants
- Angelic Debugging

Formula-based Debugging (AKA Failure Explanation)

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

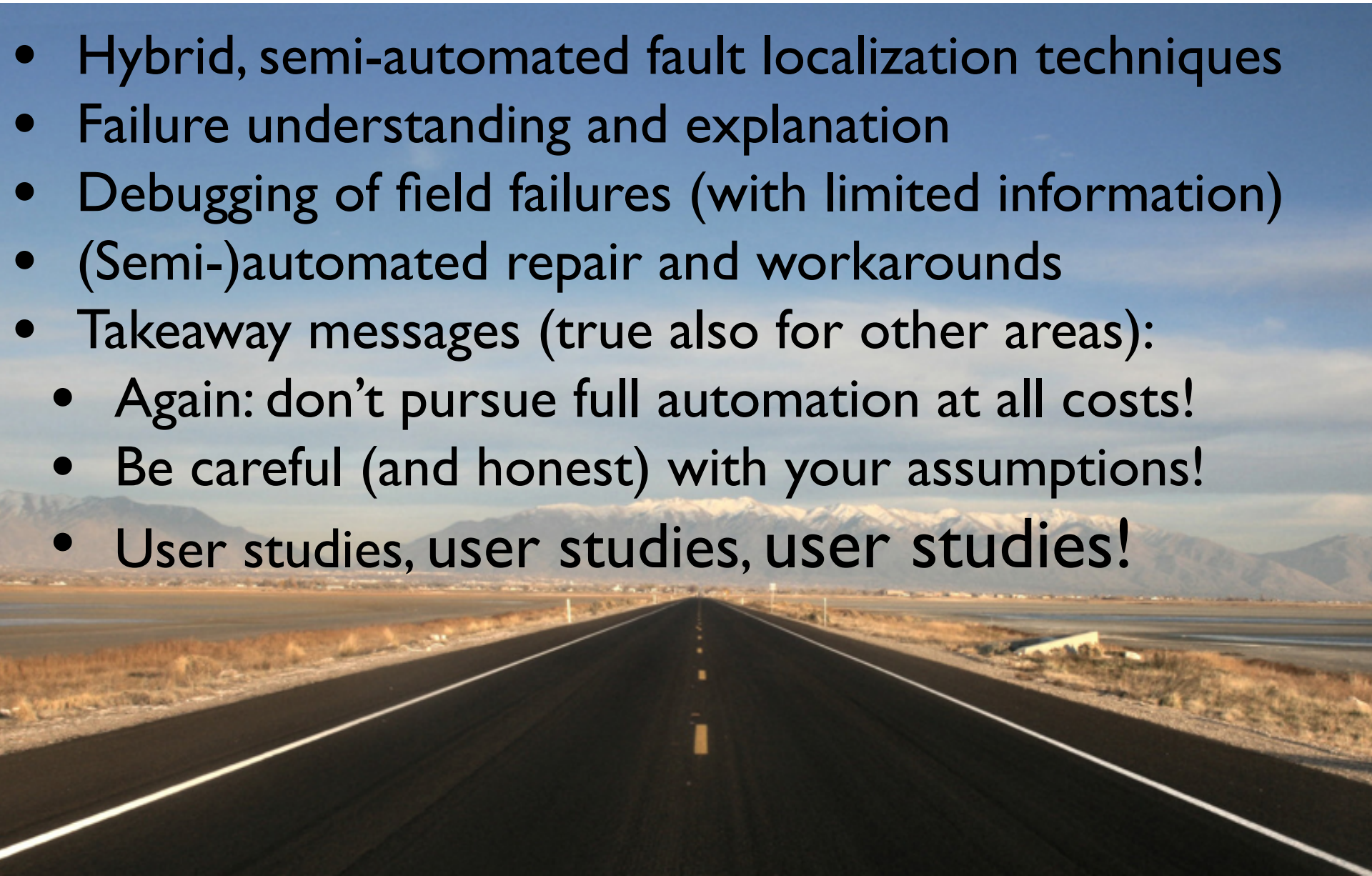
In Summary

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



- But there is still a **long** way to go...

Where Shall We Go Next?



- Hybrid, semi-automated fault localization techniques
- Failure understanding and explanation
- Debugging of field failures (with limited information)
- (Semi-)automated repair and workarounds
- Takeaway messages (true also for other areas):
  - Again: don't pursue full automation at all costs!
  - Be careful (and honest) with your assumptions!
  - User studies, user studies, user studies!

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