

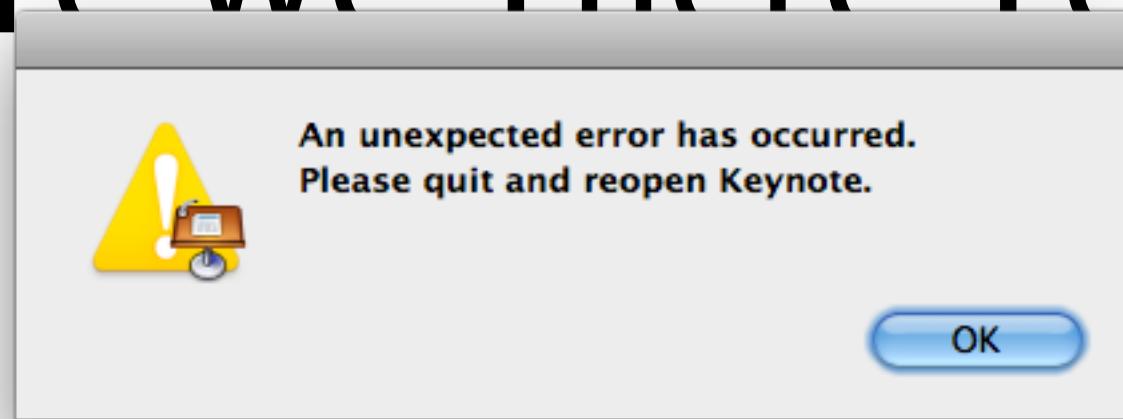
# Automated Debugging: Are We There Yet?

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/>



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-4E512E50AE9E

Exception Type: EXC\_BREAKPOINT (SIGTRAP)  
Exception Codes: 0x0000000000000002, 0x0000000000000000  
Crashed Thread: 0 Dispatch queue: com.apple.main-thread

Application Specific Information:

Hide Details Debug... Send to Apple



Location: Luetzowufer, Berlin, 10711 Germany Phone:+49/30 28493822 Fax:+4930228493822 www.esplora.de

Pending: Berlin, Germany Tegel Airport SCHONEFELD

Travel: From Airport Hauptbahnhof at Lützowplatz Lützowplatz Take shuttle Express (10 min) going to U-Bahn 5, 7, 100 bus for 10 min, p

Ongoing: Autobahn A junction R Transfer to Finkenwerder, All5, the right at Ernst-Reuter-Platz 17, 2nd Hofjägerstrasse straight on Platz into car park right. Aut Berliner P

TO DO: Autobahn A junction R Transfer to Finkenwerder, All5, the right at Ernst-Reuter-Platz 17, 2nd Hofjägerstrasse straight on Platz into car park right. Aut Berliner P

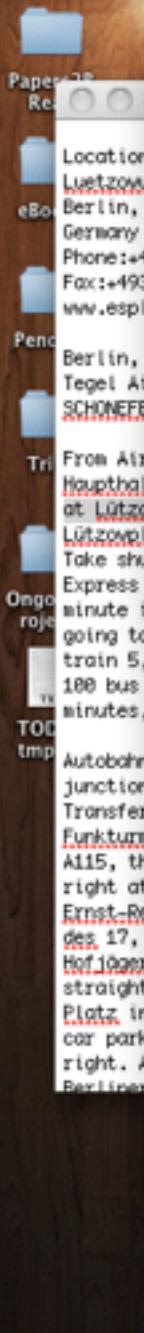
Temp: Autobahn A junction R Transfer to Finkenwerder, All5, the right at Ernst-Reuter-Platz 17, 2nd Hofjägerstrasse straight on Platz into car park right. Aut Berliner P

Base SDK Missing

Overview Breakpoints Build and Run Tasks Ungrouped Project

```
FileList.cpp
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 *cpPath = fullPath.c_str();
        int fd = open(cpPath, 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;
        fstat(fd, &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);
}
```

4:39 AM  
8:12 AM  
8:44 AM  
9:22 AM  
9:39 AM  
9:40 AM  
10:00 AM  
10:32 AM  
11:06 AM  
11:24 AM  
11:33 AM  
11:57 AM  
12:01 PM  
12:10 PM  
12:14 PM

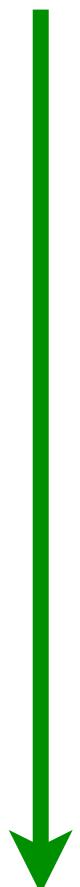


# How Are We Doing?

A Short History of Debugging

# The Birth of Debugging

???



First reference to software errors  
Your guess?

2013

# The Birth of Debugging

1840

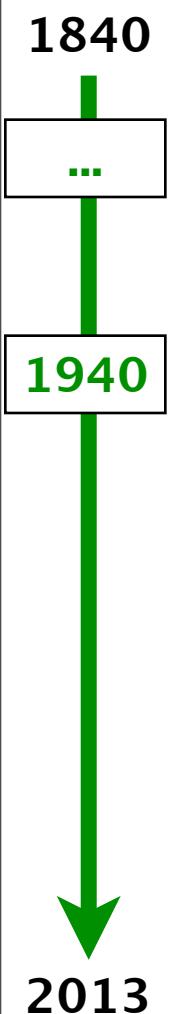
1843



- 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

9/9

0800 Arctan started  
1000 . stopped - arctan ✓  
13'00 (033) MP - MC { 1.2700 9.037 847 025  
033 PRO 2 2.130476415 9.037 846 995 correct  
correct 2.130676315

Relays 6-2 in 033 failed special speed test  
in relay 11.000 test.

Relay 2145  
Relay 3370

1100 Started Cosine Tape (Sine check)  
1525 Started Multi Adder Test.

1545



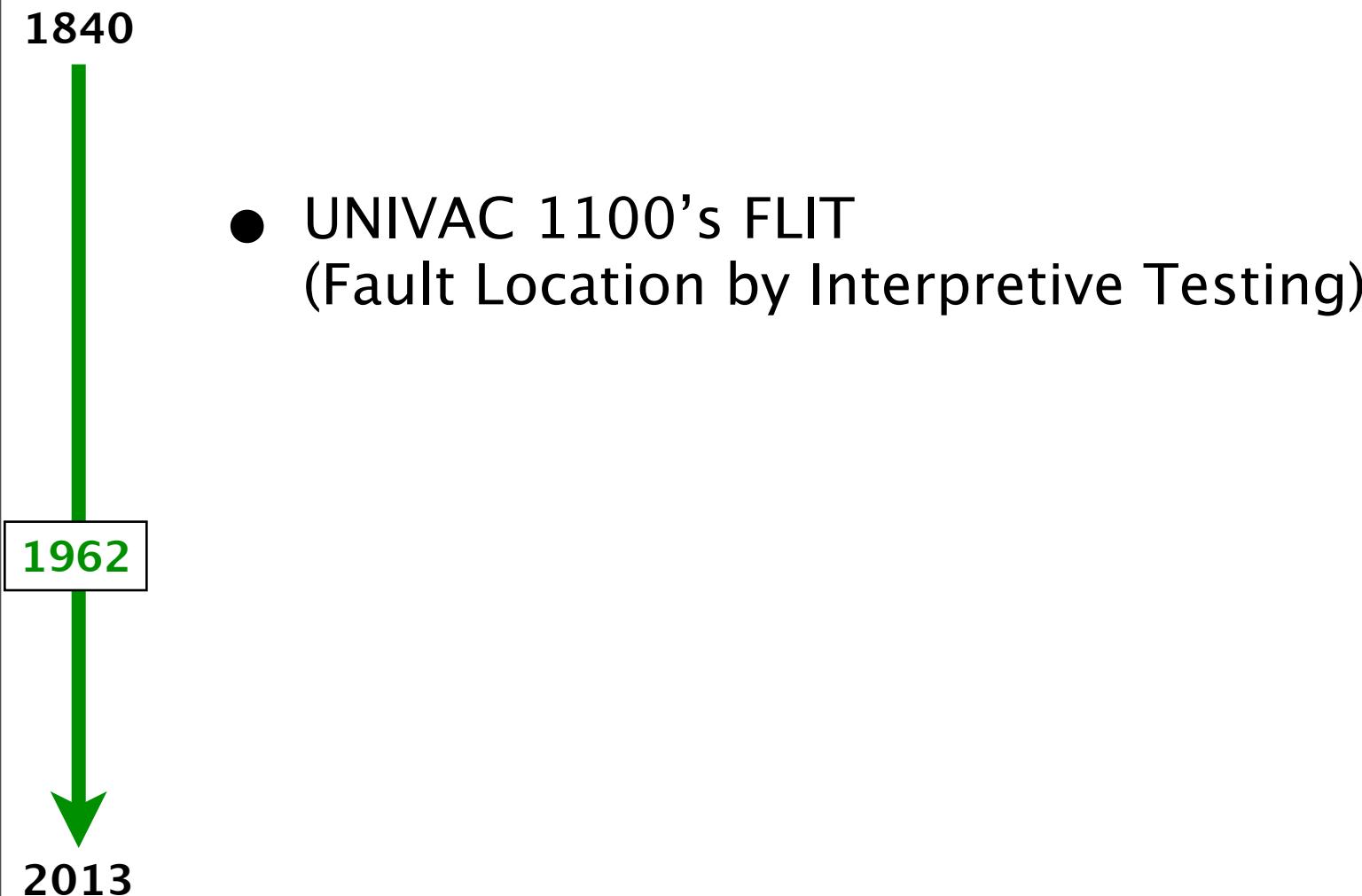
Relay #70 Panel F  
(moth) in relay.

1600 Arctan started.  
1700 closed down.

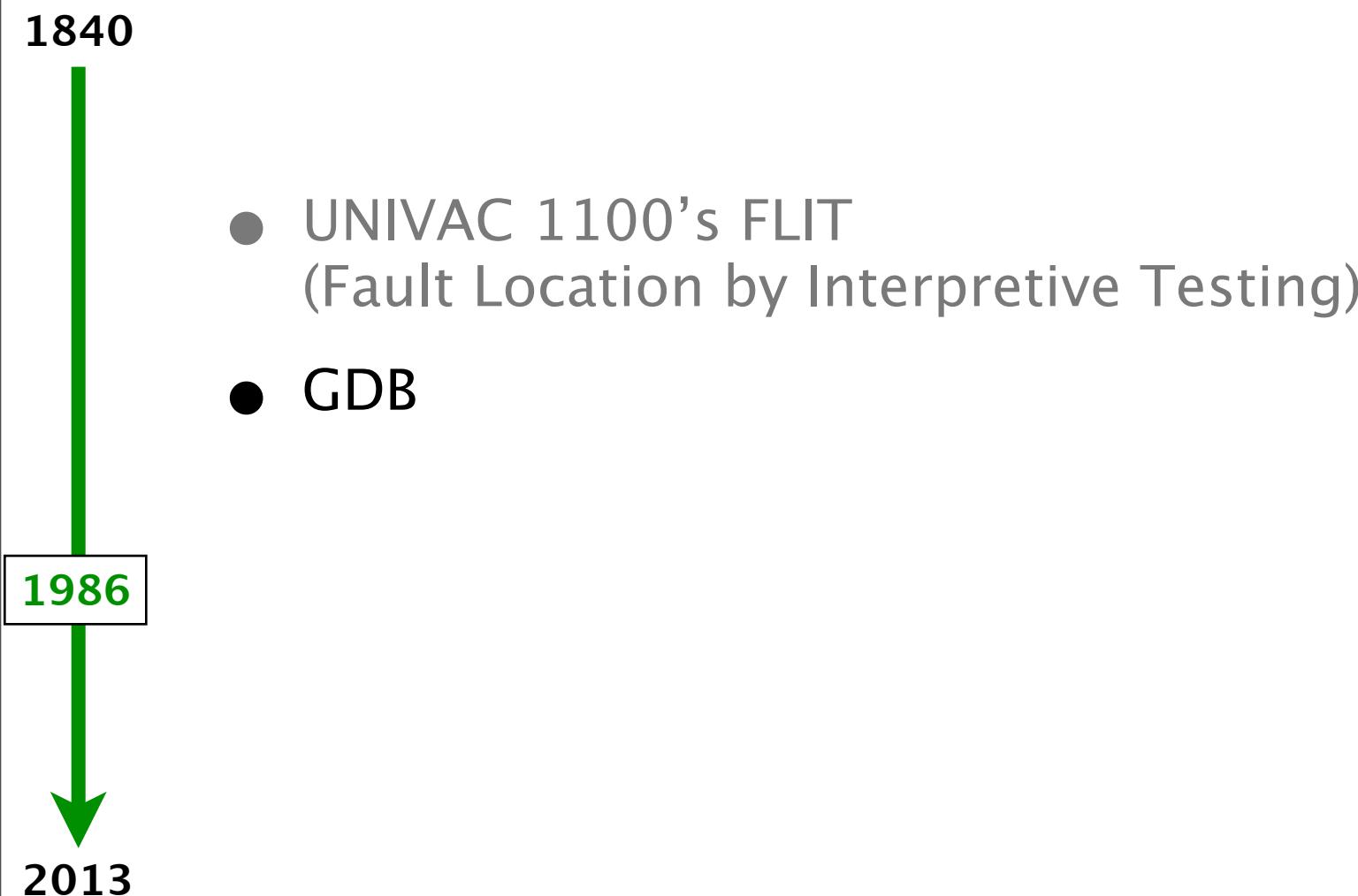
1947

2013

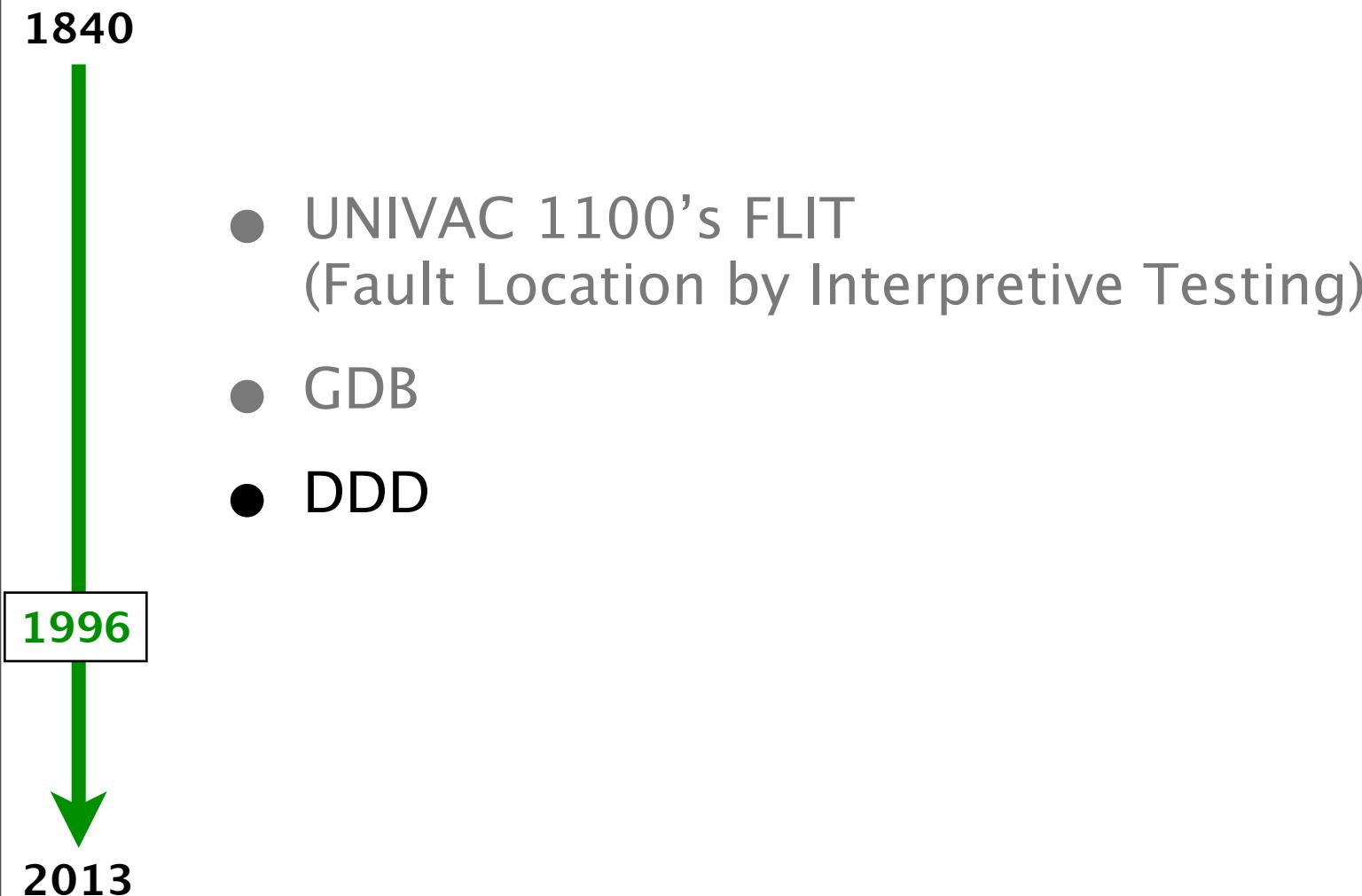
# Symbolic Debugging



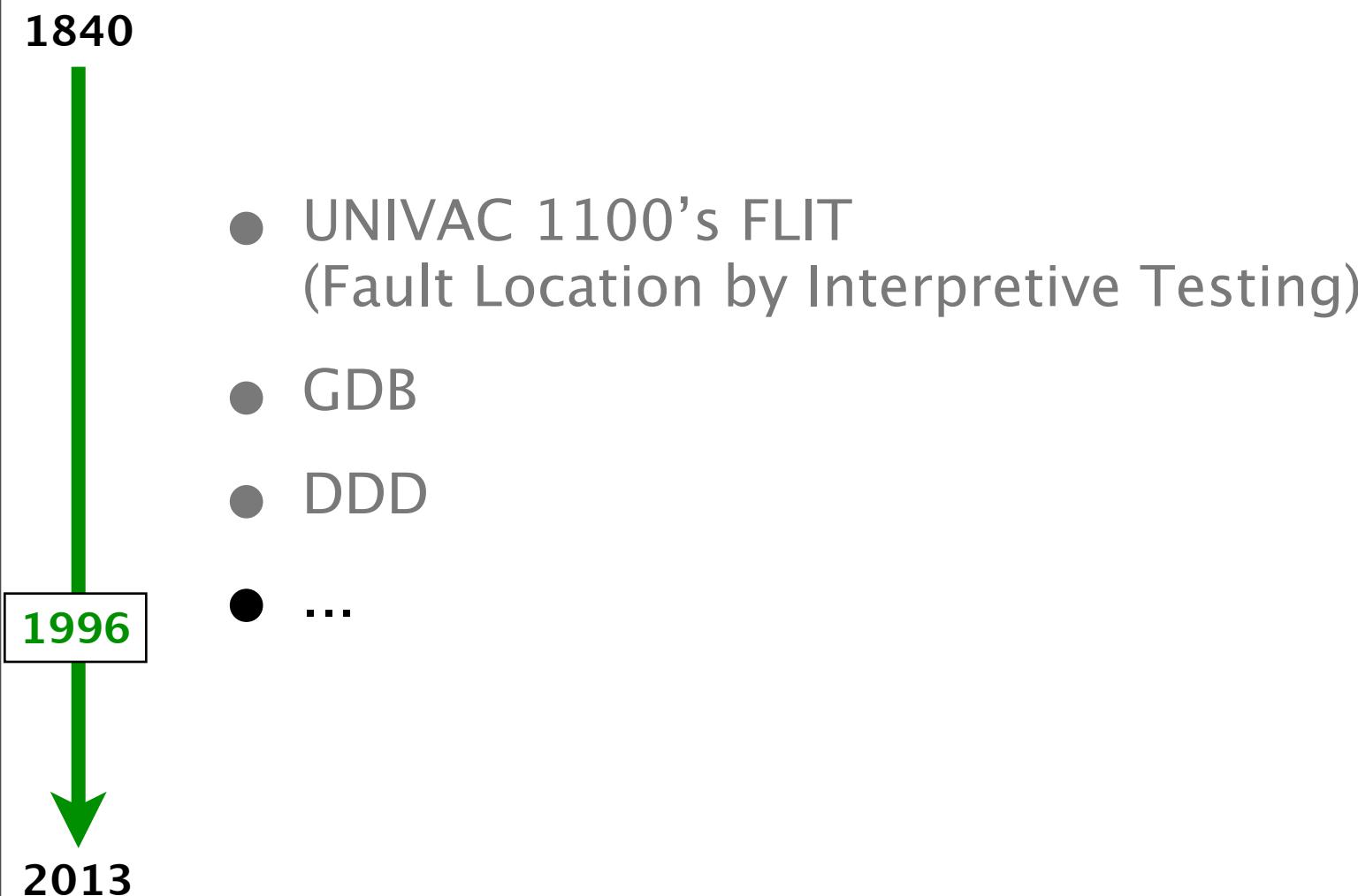
# Symbolic Debugging



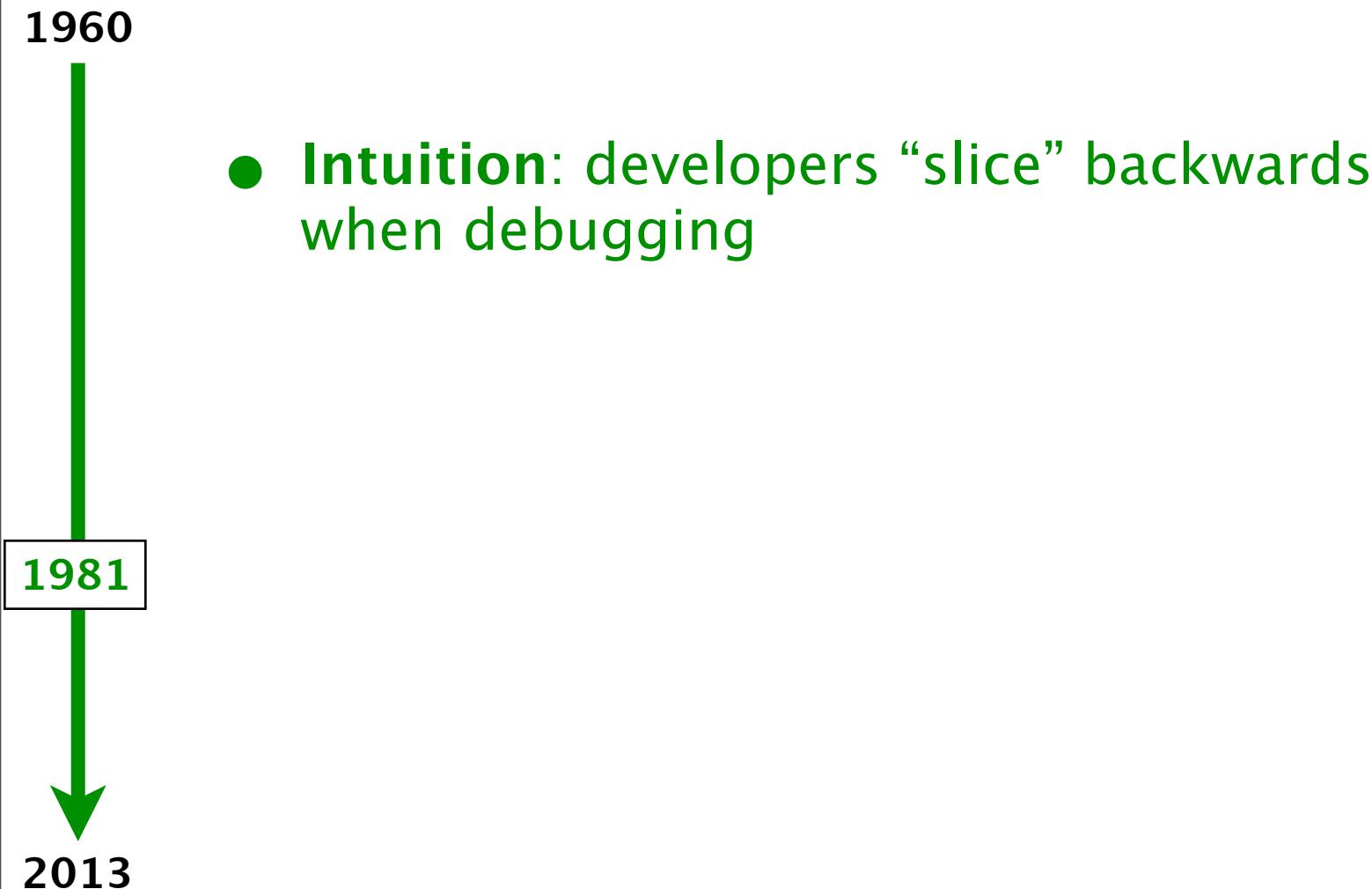
# Symbolic Debugging



# Symbolic Debugging



# Program Slicing



# Program Slicing

1960

- **Intuition:** developers “slice” backwards when debugging
- Weiser’s breakthrough paper

1981

2013

# 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

1960

- **Intuition:** developers “slice” backwards when debugging
- Weiser’s breakthrough paper

1981

2013

# Program Slicing

1960

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

1988  
1993

2013

# 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

```
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);  
}
```

Test Cases

	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
1:	•	•	•	•	•	•
2:	•	•	•	•	•	•
3:	•	•	•	•	•	•
4:	•	•			•	•
5:		•				
6:	•				•	•
7:	•					•
8:			•	•		
9:			•	•		
10:			•			
11:				•		
12:						
13:	•	•	•	•	•	•
	Pass/Fail	P	P	P	P	F

# 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);  
}
```

Test Cases

	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
1:	•	•	•	•	•	•
2:	•	•	•	•	•	•
3:	•	•	•	•	•	•
4:	•	•		•	•	
5:		•				
6:	•			•	•	
7:	•				•	
8:			•	•		
9:		•	•			
10:			•			
11:				•		
12:						
13:	•	•	•	•	•	•
	Pass/Fail	P	P	P	P	F

# Program Slicing

1960

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

1988  
1993

2013

# Program Slicing

1960

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

2008



2013

# Delta Debugging

1960

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

1999

2013

# Delta Debugging

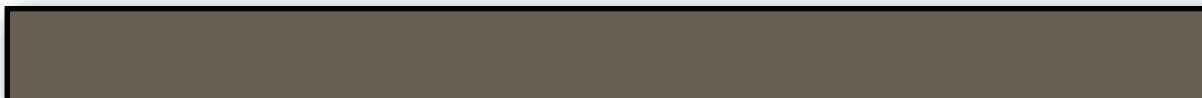
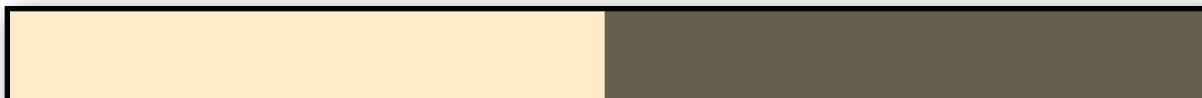
1960

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

1999

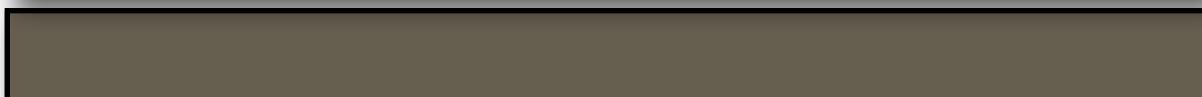
2013

# Today



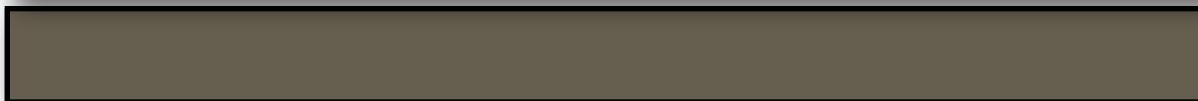
# Yesterday

# Today



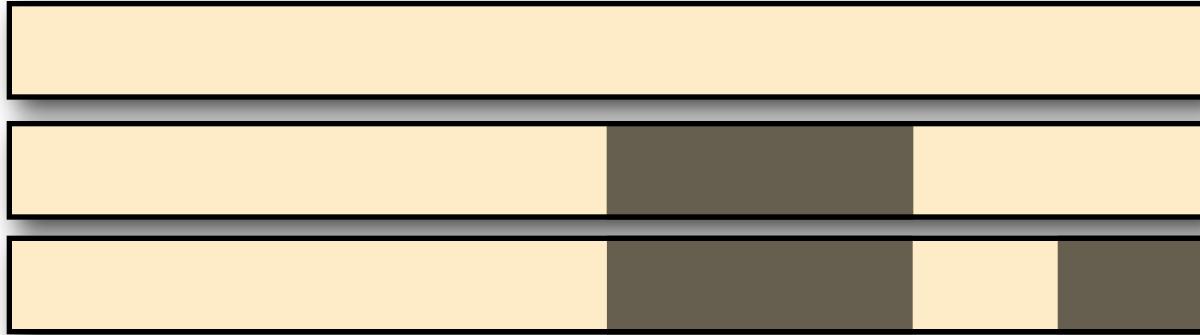
# Yesterday

# Today

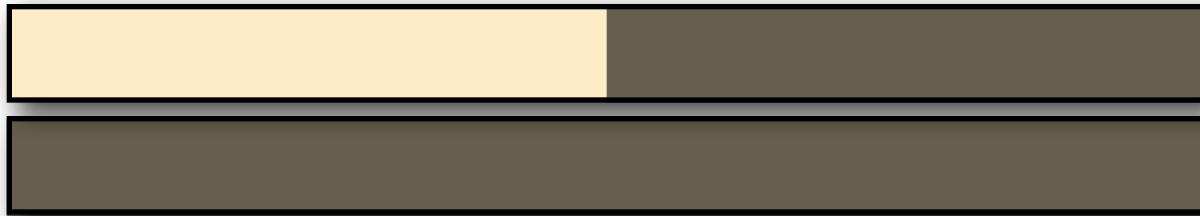


# Yesterday

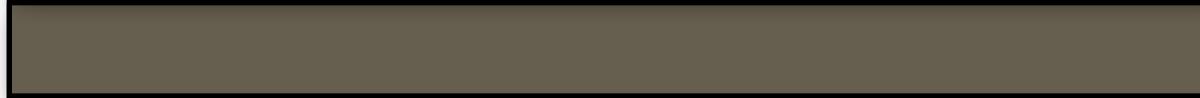
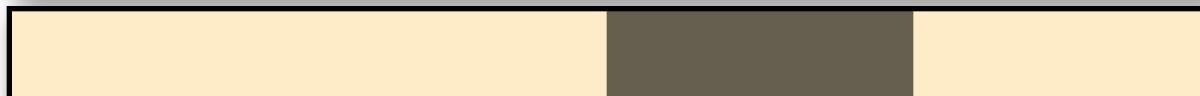
# Today



# Yesterday



# Today

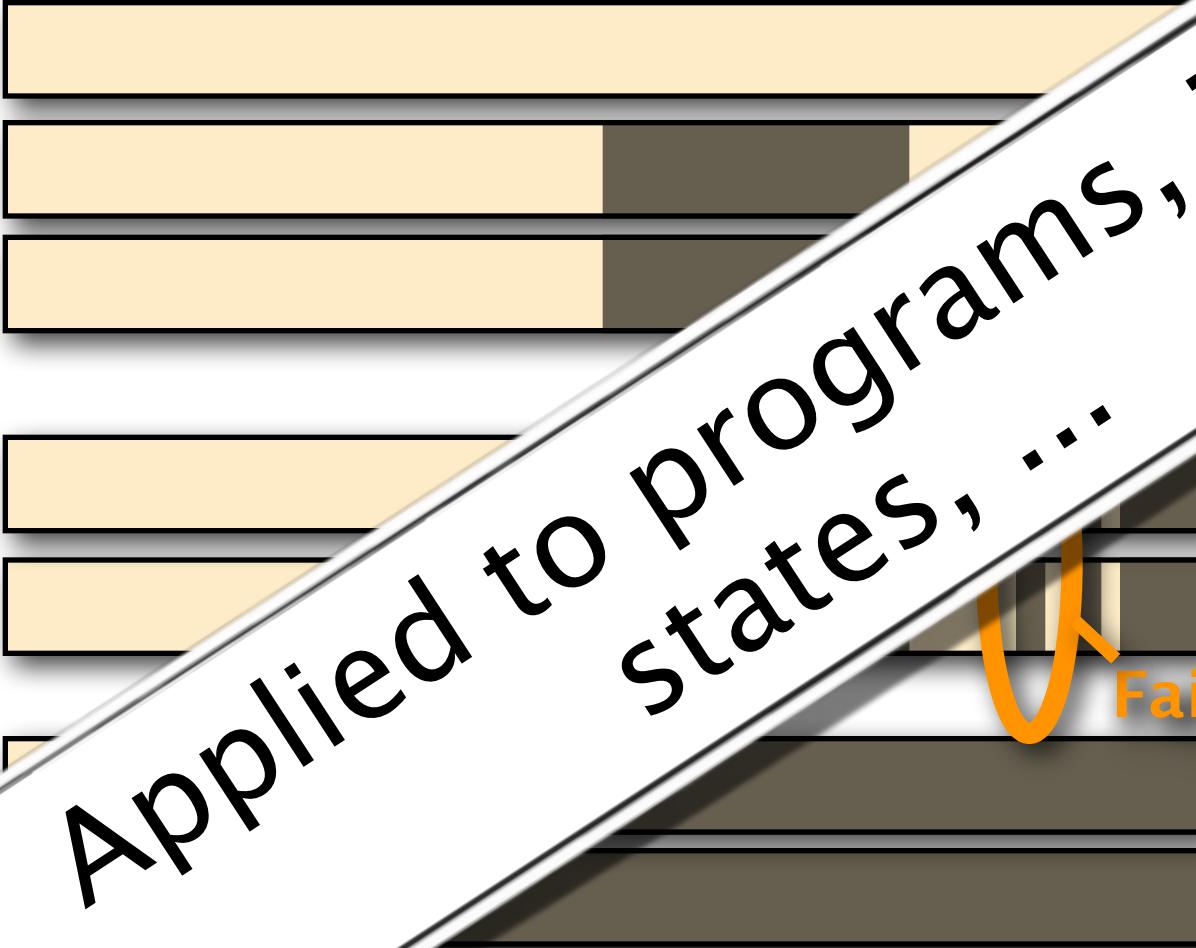


Failure cause



# Yesterday

Today



# Statistical Debugging

1960

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

2001



2013

# Statistical Debugging

1960

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

2001

2013

# Tarantula

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

```

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);
}

```

Test Cases	Pass/Fail							suspiciousness
	P	P	P	P	P	P	F	
3,3,5	•	•	•	•	•	•	•	0.5
1,2,3	•	•	•	•	•	•	•	0.5
3,2,1	•	•	•	•	•	•	•	0.5
5,5,5	•	•	•	•	•	•	•	0.5
5,3,4	•	•	•	•	•	•	•	0.5
2,1,3	•	•	•	•	•	•	•	0.5
<i>susp(1) = <math>\frac{\frac{1}{1}}{\frac{5}{5} + \frac{1}{1}} = 0.5</math></i>							0.5	
<i>susp(7) = <math>\frac{\frac{1}{1}}{\frac{1}{5} + \frac{1}{1}} = 0.8</math></i>							0.5	

# Statistical Debugging

1960

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

2001

2013

# Statistical Debugging

1960

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

2003

2013

# Statistical Debugging

1960

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

2006

2013

# 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...

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

...

2013

# Formula-based Debugging (AKA Failure Explanation)

1960

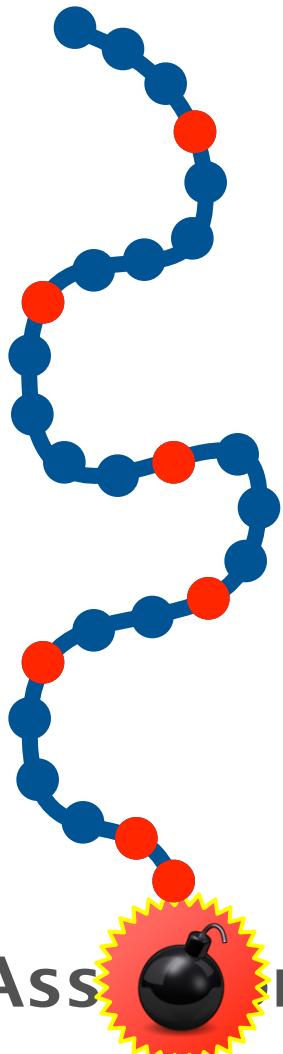
- **Intuition:** executions can be expressed as formulas that we can reason about

2009



2013

Input I



Formula

unsatisfiable

1 **Input = I**  $\wedge$

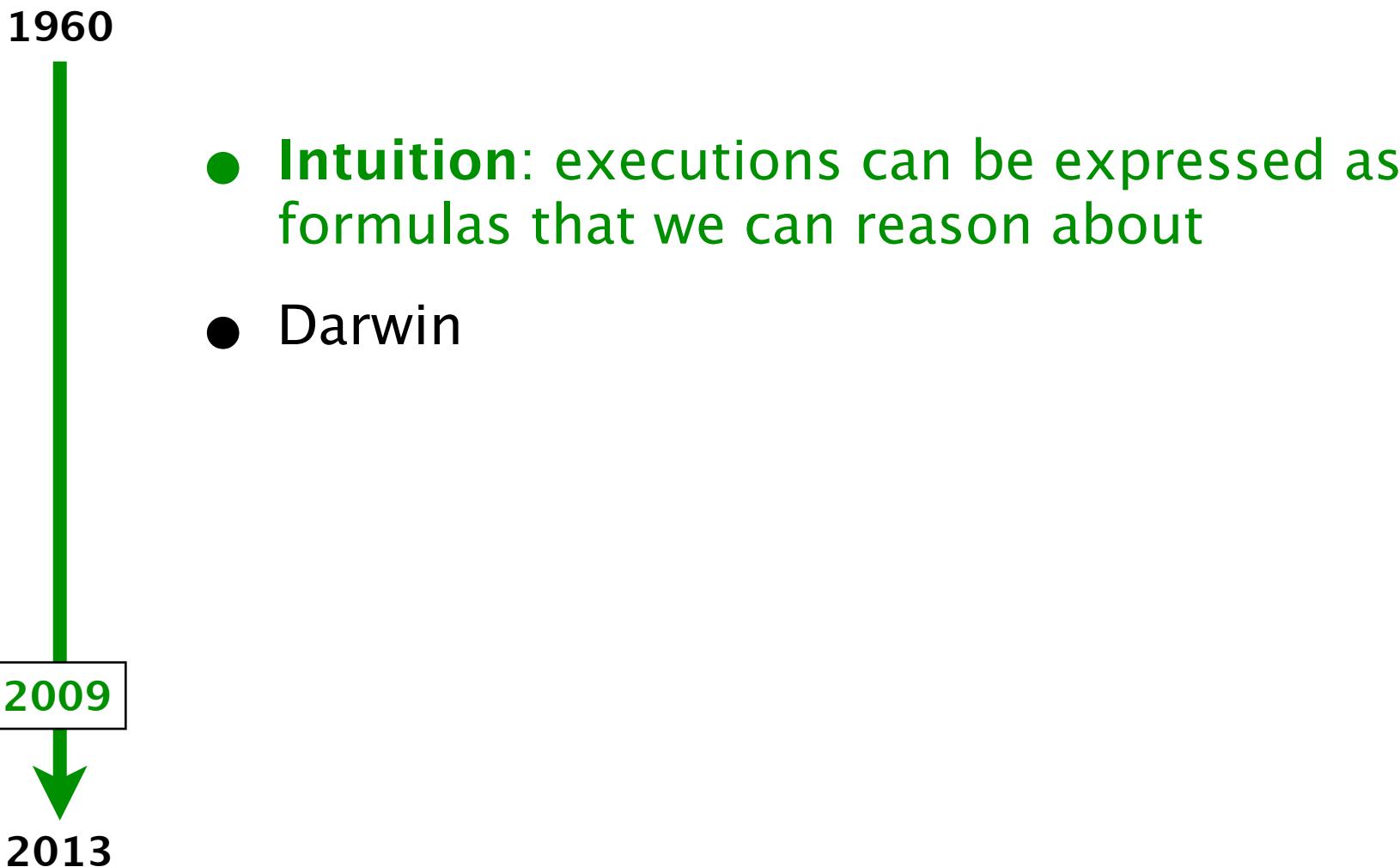
2  $c_1 \wedge c_2 \wedge c_3 \wedge \dots \wedge$   
 $\dots \wedge c_{n-2} \wedge c_{n-1} \wedge c_n \wedge$

3 **A**

MAX-SAT  
Complement

{  $c_i$  }

# Formula-based Debugging (AKA Failure Explanation)



# 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., Zhang et al.)
- Fault localization for distributed systems (e.g., Allard et al.)
- Debugging of C/C++ programs (e.g., Godefroid et al.)

Not meant to be comprehensive!

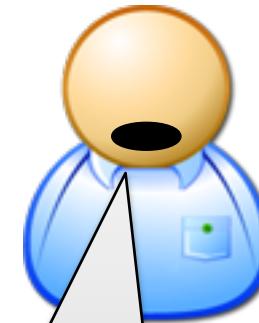
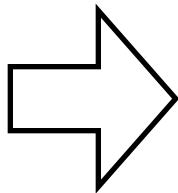
- Debugging of Linux (e.g., Godefroid et al.)
- Debugging of web pages, comments, concurrency
- Identifying workarounds/recovery strategies (e.g., Gorla et al.)
- Formula based debugging (e.g., Jose et al., Ermis et al.)
- ...

2013

# Are We There Yet?

Can We Debug at the Push of a Button?

# Automated Debugging (rank based)

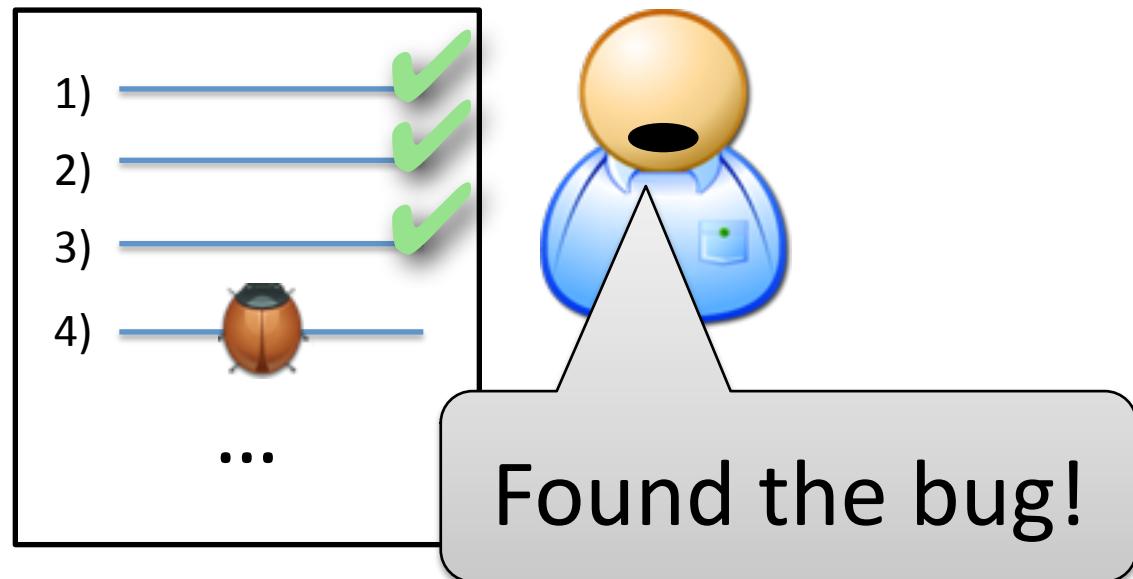


Here is a list of places to check out

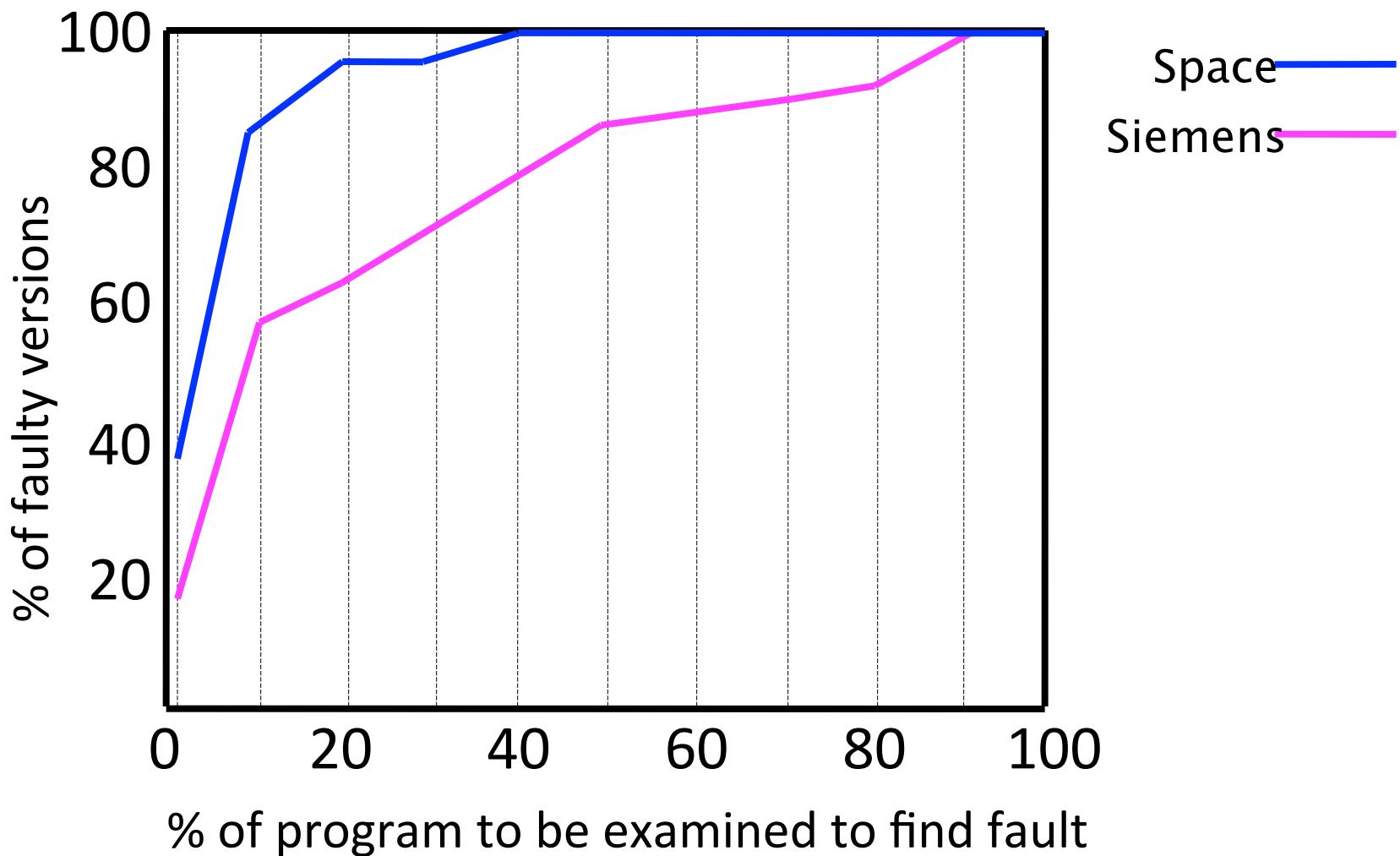
Ok, I will check out your suggestions one by one.



# 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  $\rightarrow$  10 LOC

10,000 LOC

Moreover, strong assumptions

100 LOC  $\rightarrow$  10,000 LOC

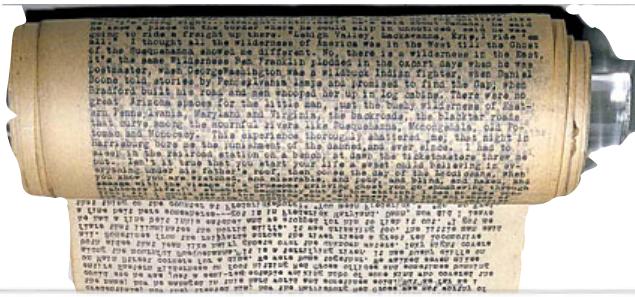


# Assumption #1: Programmers exhibit perfect bug understanding



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



Does the conceptual model make sense?  
Have we really evaluated it?

# Where Shall We Go Next?

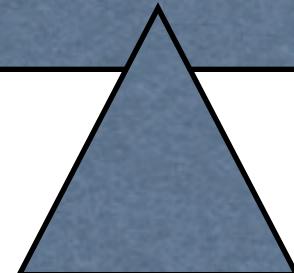
Are We Headed in the Right Direction?



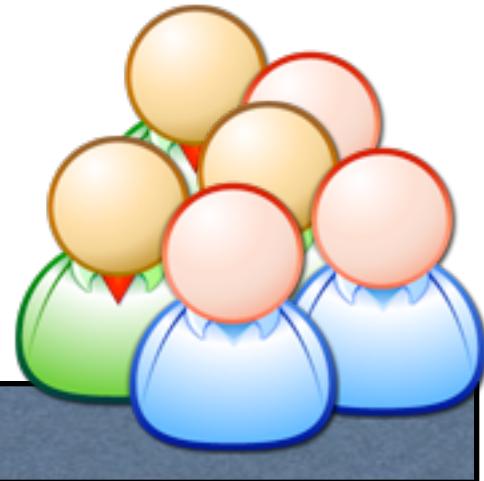
AKA: “Are Automated Debugging Techniques  
Actually Helping Programmers?” ISSTA 2011  
Chris Parnin and Alessandro Orso

# What do we know about automated

Studies on tools



Human studies





Let's see...

Over 50 years of research  
on automated debugging.

2001. Statistical Debugging

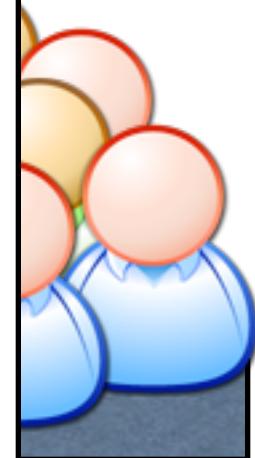
1999. Delta Debugging

1981. Weiser. Program Slicing

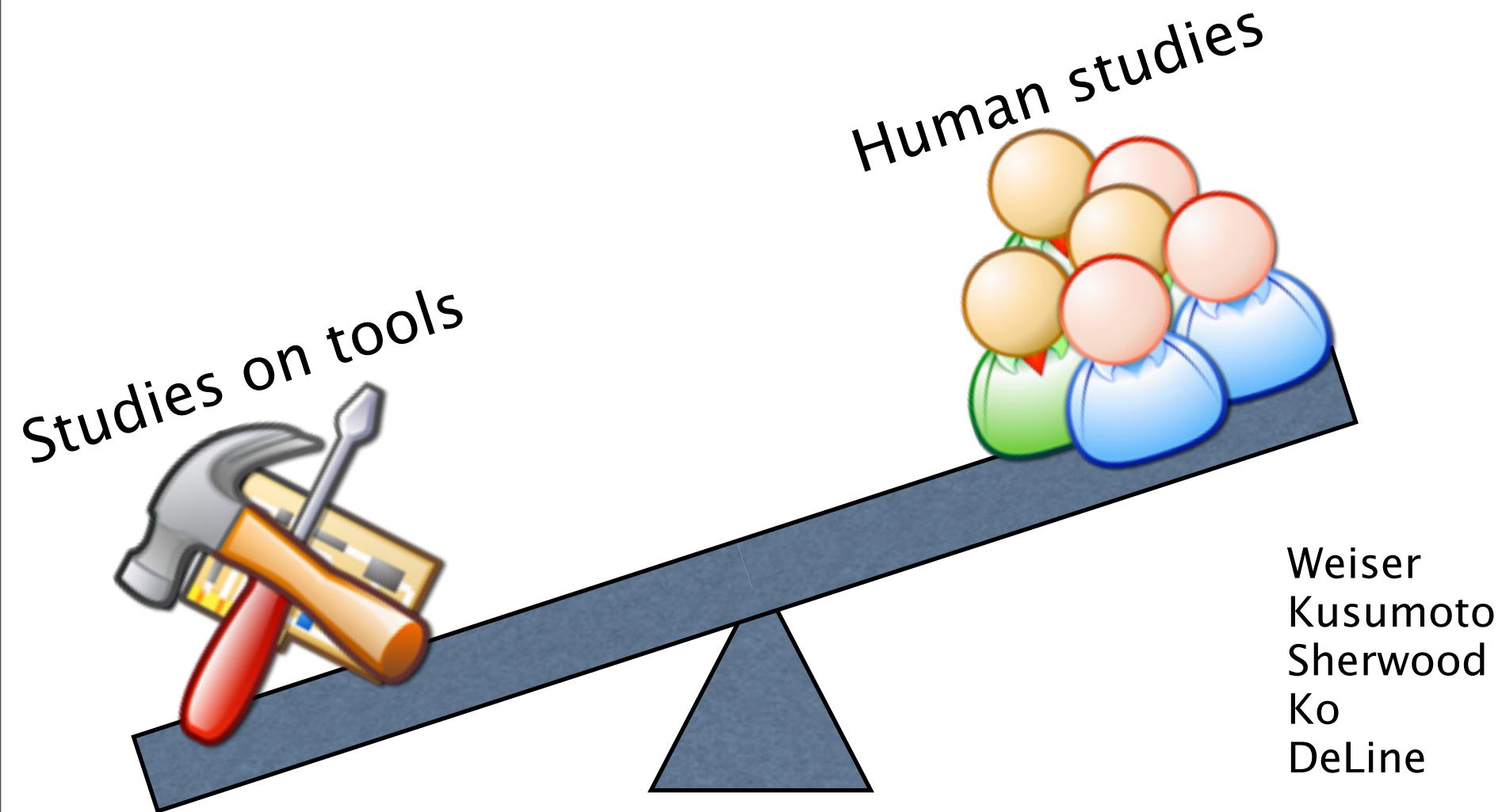
1962. Symbolic Debugging (UNIVAC FLIT)

N  
D

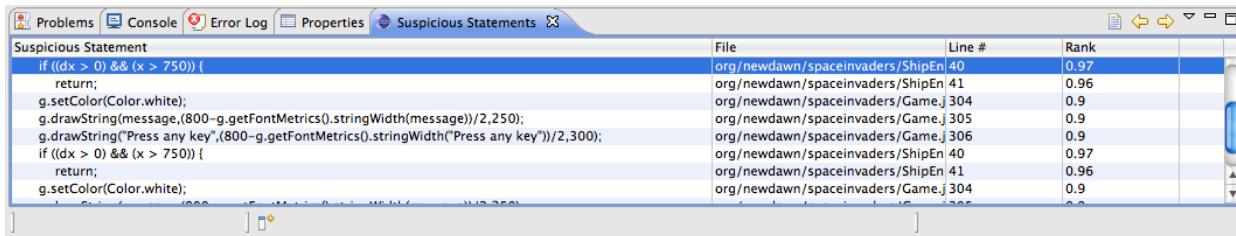
studies



# What do we know about automated



# Are these Techniques and Tools Actually Helping Programmers?



A screenshot of a software interface titled "Suspicious Statements". The interface includes tabs for "Problems", "Console", "Error Log", "Properties", and "Suspicious Statements". The "Suspicious Statements" tab is active, showing a list of code snippets and their associated file, line number, and rank. The list is sorted by rank, with the highest rank at the top. The code snippets are from a file named "ShipEn.java".

	File	Line #	Rank
1	org/newdawn/spaceinvaders/ShipEn	40	0.97
2	org/newdawn/spaceinvaders/ShipEn	41	0.96
3	org/newdawn/spaceinvaders/Game	304	0.9
4	org/newdawn/spaceinvaders/Game	305	0.9
5	org/newdawn/spaceinvaders/Game	306	0.9
6	org/newdawn/spaceinvaders/ShipEn	40	0.97
7	org/newdawn/spaceinvaders/ShipEn	41	0.96
8	org/newdawn/spaceinvaders/Game	304	0.9



- 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

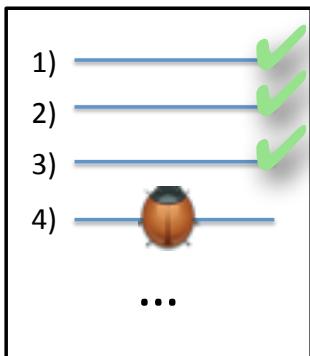


```

  (0,0,0)
  {
    return [self, self, self];
  }
}

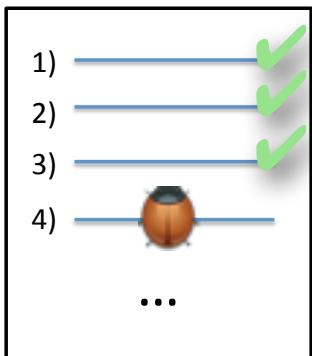
class A {
  constructor(...args) {
    this.args = args;
  }
  [Symbol.iterator]() {
    return {
      next() {
        if (this.args.length === 0) {
          return {value: undefined, done: true};
        }
        let value = this.args.shift();
        return {value, done: false};
      }
    };
  }
  [Symbol.unscopableKeys] = [
    'length',
    'Symbol.iterator'
  ];
}

```



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

# Experimental Protocol: Setup



## Tools

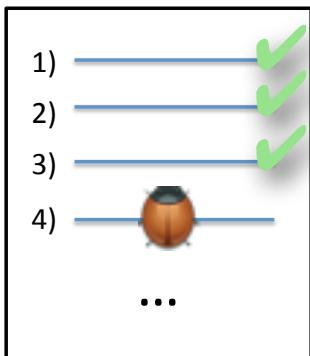
- Rank-based tool  
(Eclipse plug-in, logging)
- Eclipse debugger

# Experimental Protocol: Setup

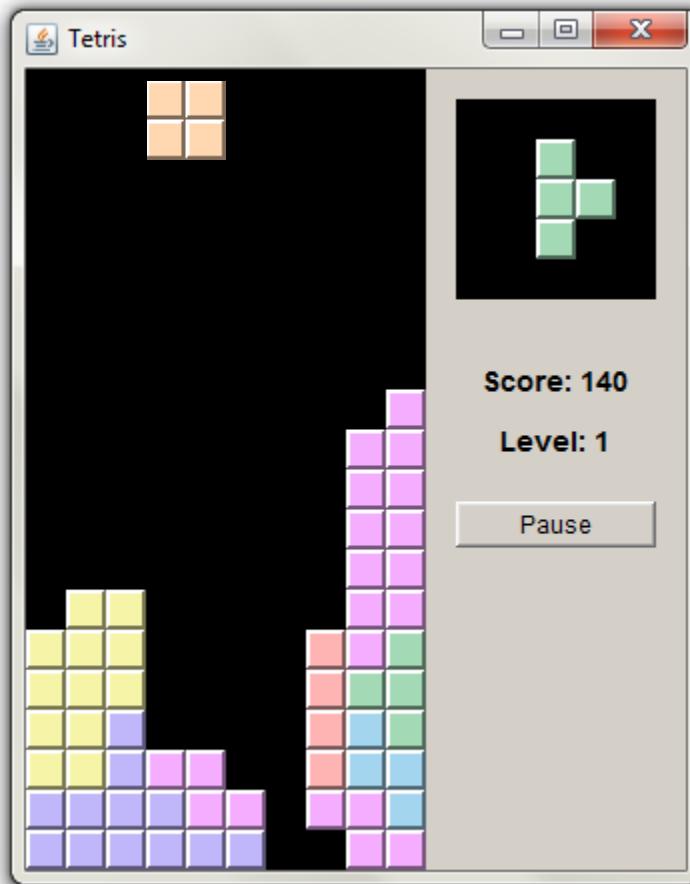


## Software subjects:

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



# Tetris Bug



(Easier)

# NanoXML Bug

The input, **testvm\_22.xml**, contains the following input xml document:

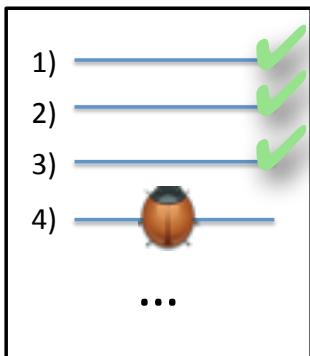
```
<Foo a="test">
  <ns:Bar>
    <Blah x="1" ns:x="2"/>
  </ns:Bar>
</Foo>
```

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)

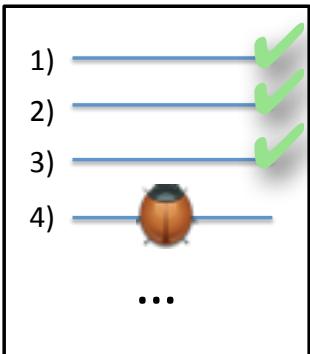
# Experimental Protocol: Setup



## Software subjects:

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

# Experimental Protocol: Setup



## Tasks:

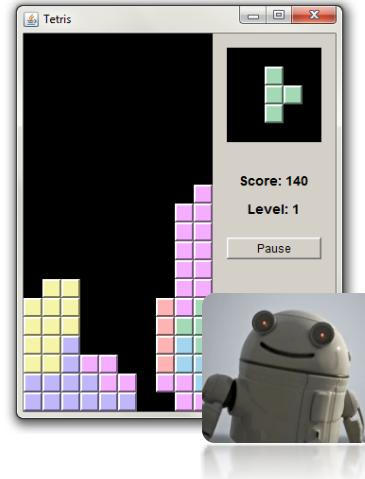
- Fault in Tetris
- Fault in NanoXML
- 30 minutes per task
- Questionnaire at the end

# Experimental Protocol: Studies and Groups

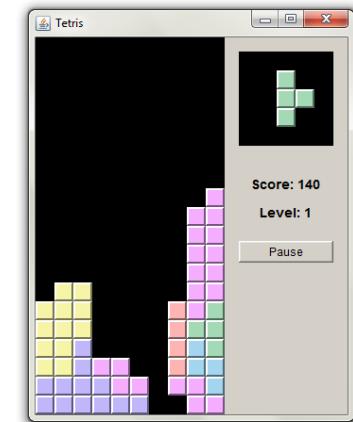
# Experimental Protocol: Studies and Groups

## Study 1

A



B



When running the NanoML program [main is in class Parser1\_ve\_v1], the following exception is thrown:  
Exception in thread "main" net.n3.nanoml.XMLParseException:  
XML Not Well-Formed at Line 19: Closing tag does not match opening tag: ns:Bar' != ':Bar'  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:145)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:138)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:202)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:153)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:202)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:159)  
at net.n3.nanoml.xml.XMLParser.parse (XMLParser.java:131)  
at net.n3.nanoml.Parser1\_ve\_v1.main (Parser1\_ve\_v1.java:150)

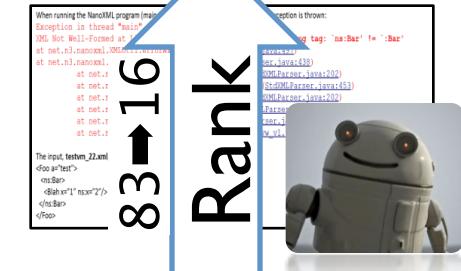
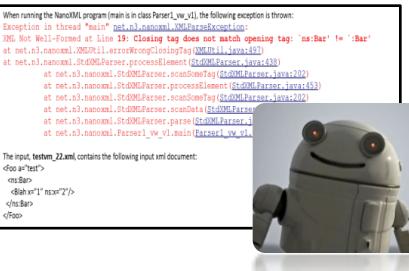
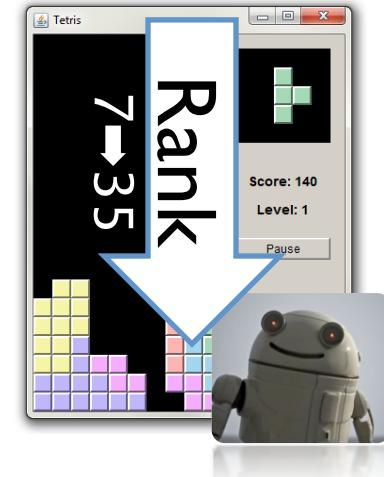
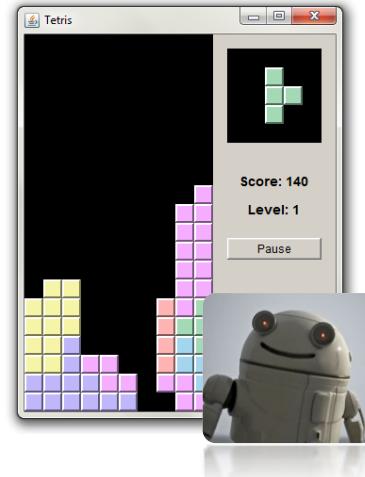
The input, testum\_22.xml, contains the following input xml document:  
<foo ar="test">  
<ns:Bar>  
<ns:Bar>1<ns:Bar>2</ns:Bar>  
</ns:Bar>  
</foo>

When running the NanoML program [main is in class Parser1\_ve\_v1], the following exception is thrown:  
Exception in thread "main" net.n3.nanoml.XMLParseException:  
XML Not Well-Formed at Line 19: Closing tag does not match opening tag: 'ns:Bar' != ':Bar'  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:145)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:138)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:202)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:153)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:202)  
at net.n3.nanoml.xml.XMLParser.readEndElementTag (XMLParser.java:159)  
at net.n3.nanoml.xml.XMLParser.parse (XMLParser.java:131)  
at net.n3.nanoml.Parser1\_ve\_v1.main (Parser1\_ve\_v1.java:150)

The input, testum\_22.xml, contains the following input xml document:  
<foo ar="test">  
<ns:Bar>  
<ns:Bar>1<ns:Bar>2</ns:Bar>  
</ns:Bar>  
</foo>

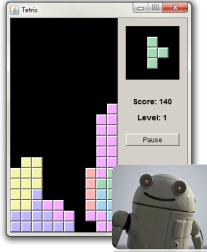
# Experimental Protocol: Studies and Groups

# Study 2

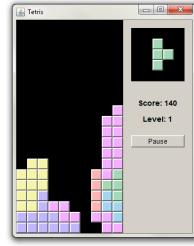


# Study Results

A



B



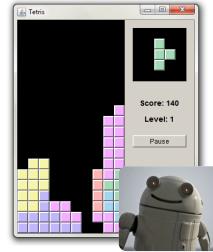
Run the command below to install the `awscli` package. The following message is shown:

```
aws configure
AWS Access Key ID [None]: AKIAJ43GK3C6W273ZGQ
AWS Secret Access Key [None]: 3L1qKJGqB4XWzWzWzWzWzWzWzWzWzWz
Default region name [None]: us-east-1
Default output format [None]: json
```

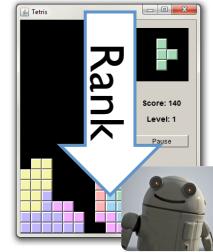
The next `aws` command contains the following environment document:

```
aws configure
[{"name": "aws_access_key_id", "value": "AKIAJ43GK3C6W273ZGQ", "type": "String"}, {"name": "aws_secret_access_key", "value": "3L1qKJGqB4XWzWzWzWzWzWzWzWzWzWz", "type": "String"}, {"name": "aws_session_token", "value": null, "type": "String"}, {"name": "aws_unsigned_url", "value": null, "type": "String"}, {"name": "aws_region", "value": "us-east-1", "type": "String"}, {"name": "aws_output", "value": "json", "type": "String"}, {"name": "aws_profile", "value": null, "type": "String"}]
```

8



D



One strategy I usually employ is to add a `Rank` column to the end of my `columns` block. This allows me to sort the results of my query by rank. For example, the following query will return the top 10 results for `SELECT`ing `name` and `age` from `people`, ordered by `age` in descending order:

```
SELECT name, age
  FROM people
 ORDER BY age DESC
  LIMIT 10;
```

The result `name,Rank` contains the following:

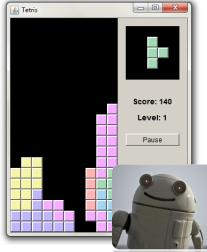
name	Rank
John	1
Bob	2
Mike	3
Paul	4
Steve	5
David	6
Tom	7
Mark	8
Jeff	9
Chris	10

As you can see, the results are ordered by `age` in descending order, and the top 10 results are returned. The `Rank` column is also included in the results, so you can see the rank of each result.

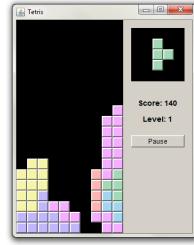
	Tetris	NanoXML
A		
B		
C		
D		

# Study Results

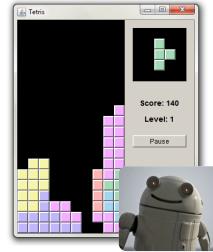
A



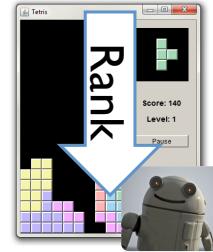
B



8



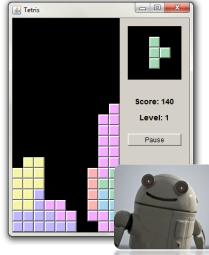
D



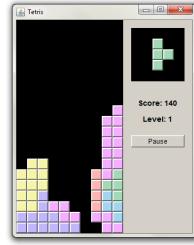
	Tetris	NanoXML
A	Not significantly different	
B		
C		
D		

# Study Results

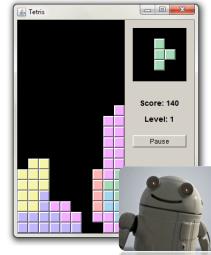
A



B

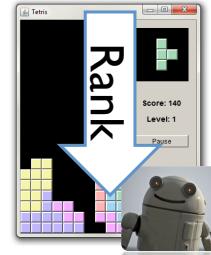


8



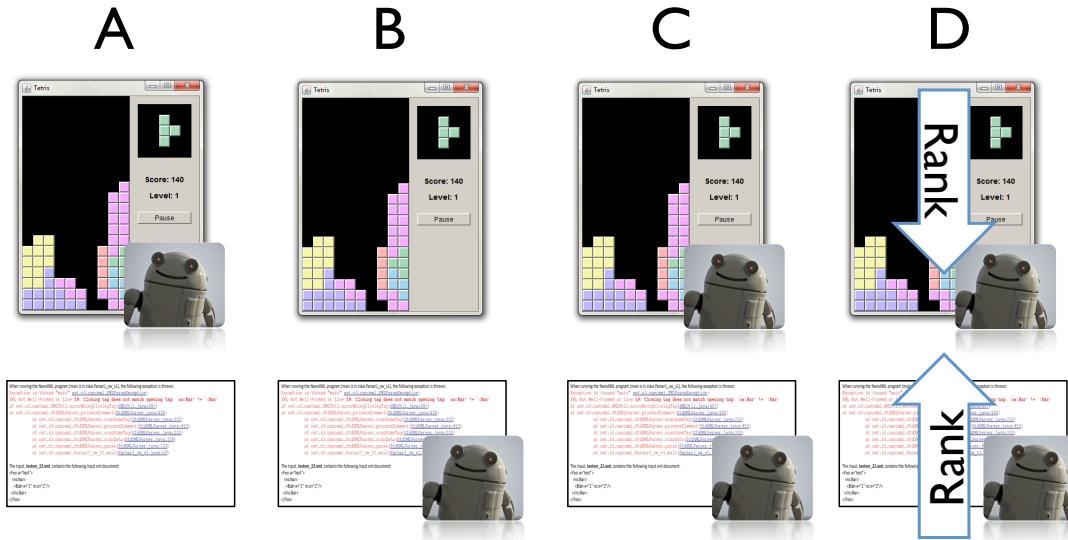
```
#!/usr/bin/python
# Author: www.pythontutor.com, www.pythontutor.com/visualize.html#mode=edit
# License: www.pythontutor.com/visualize.html#mode=edit
# Description: www.pythontutor.com/visualize.html#mode=edit
```

D



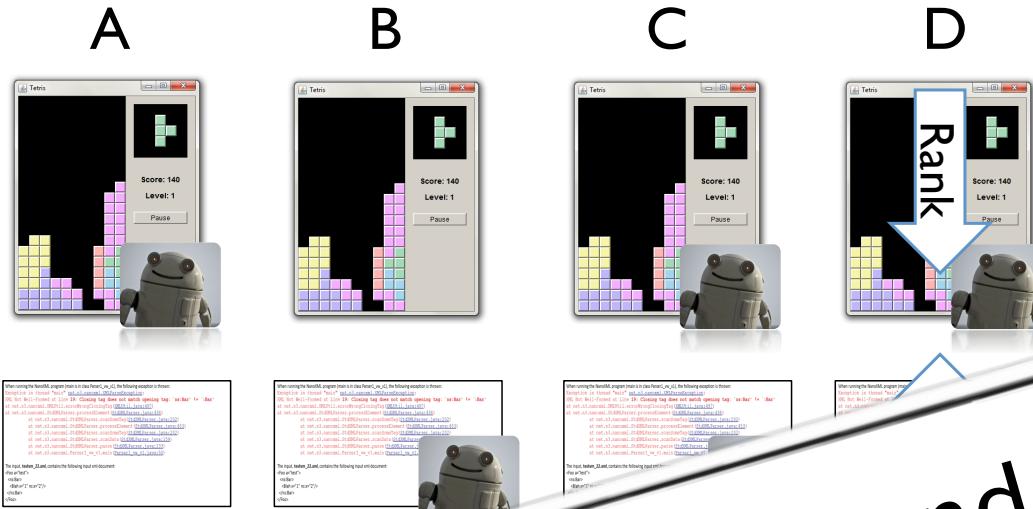
	Tetris	NanoXML
A	Not significantly different	Not significantly different
B	Not significantly different	Not significantly different
C	Not significantly different	Not significantly different
D	Not significantly different	Not significantly different

# Study Results



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

# Study Results



# Analysis of results and questionnaires...

significantly  
different

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?

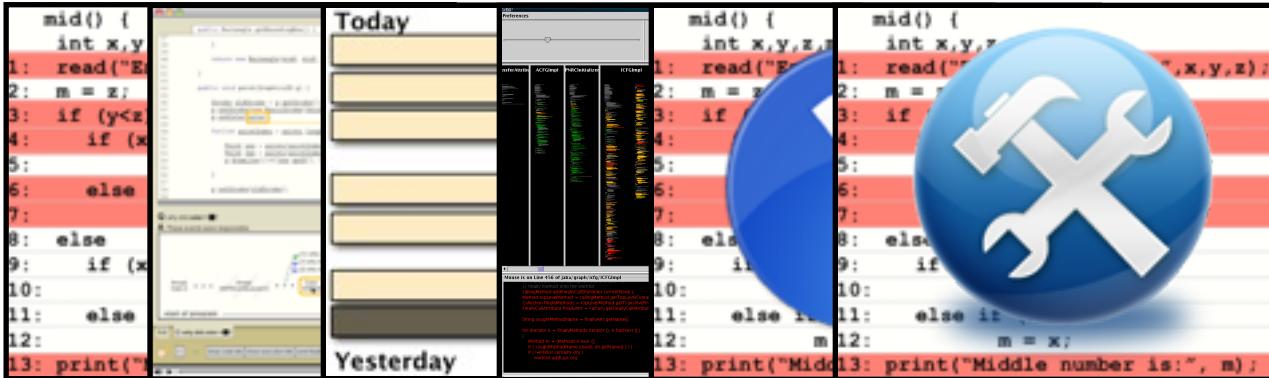
- 1) The statements in the list were sometimes useful as starting points
- 2) (Tetris) Several participants preferred to search based on intuition
- 3) (NanoXML) Several participants gave up on the tool after investigating too many false positives

# Research Implications

- Percentages will not cut it (e.g., 1.8% == 83<sup>rd</sup> 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