

SOFT620020.01  
Advanced Software  
Engineering

Bihuan Chen, Pre-Tenure Assoc. Prof.

[bhchen@fudan.edu.cn](mailto:bhchen@fudan.edu.cn)

<https://chenbihuan.github.io>

# Course Outline

| Date    | Topic                   | Date    | Topic              |
|---------|-------------------------|---------|--------------------|
| Sep. 09 | Introduction            | Nov. 04 | Mobile Testing     |
| Sep. 16 | Testing Overview        | Nov. 11 | Delta Debugging    |
| Sep. 23 | Guided Random Testing   | Nov. 18 | Presentation 2     |
| Sep. 30 | Search-Based Testing    | Nov. 25 | Bug Localization   |
| Oct. 12 | Performance Analysis    | Dec. 02 | Automatic Repair   |
| Oct. 14 | Presentation 1          | Dec. 09 | Symbolic Execution |
| Oct. 21 | Security Testing        | Dec. 16 | Big Code Analysis  |
| Oct. 28 | <b>Compiler Testing</b> | Dec. 23 | Presentation 3     |

# Discussion – Do You Trust Compilers?



```
int main () {  
    // do something  
    ...  
    return 0;  
}
```



```
...  
call puts  
movl $0, %eax  
popq %rbp  
ret
```

developers' **belief**: compilers are **faithful** translators

# Reflect on This Trust – llvm bug 14972

```
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;    removed by optimization
    foo(s);
    return 0;
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

# Reflect on This Trust – gcc bug 58731

```
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        int f = 2147483647 - b; integer overflow
        for (; c++)
            for (;;) {
                e = a > f;
                if (d) break;
            }
    return 0;
}
```

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O3 test.c ; ./a.out
^C
```

# Reflect on This Trust – Debatable Bug 1

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;

if (buf + len >= buf_end)
    return; // len too large


if (buf + len < buf) → if (0)
return; // overflow, buf + len wrapped around

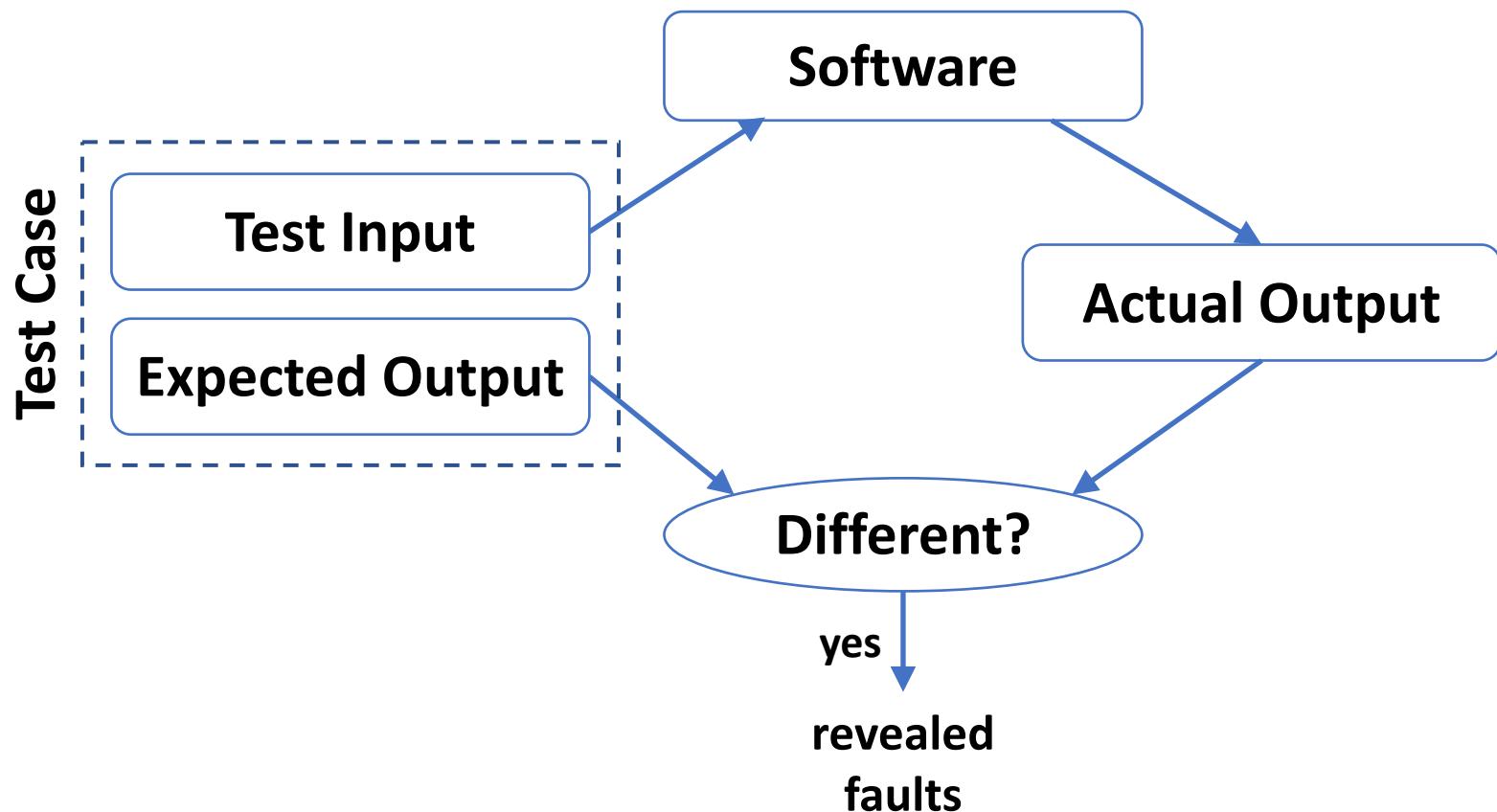
// write to buf[0..len-1]
...
```

But this is a **security threat**, i.e., use a large **len** to trigger **buffer overflow**

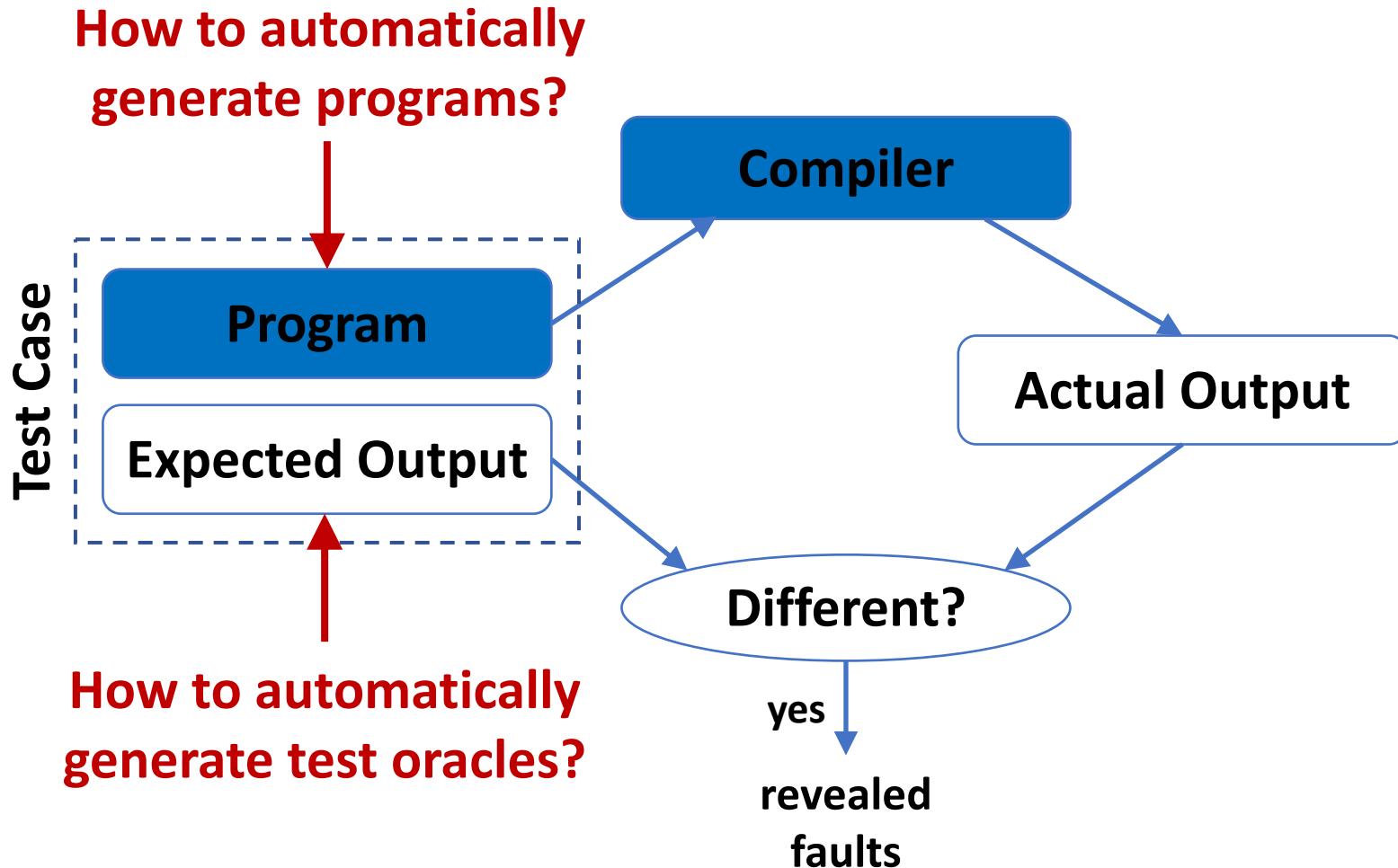
# Reflect on This Trust – Debatable Bug 2

```
bool isPasswordCorrect() {  
    bool isPasswordCorrect = false;  
    string Password("password");  
  
    if(Password == getPasswordFromUser()) {  
        isPasswordCorrect = true;  
    }  
    // removed from the optimized code  
    // secure the code by wiping the password from memory  
    memset(Password, 0, sizeof(Password));  
  
    return isPasswordCorrect;  
}
```

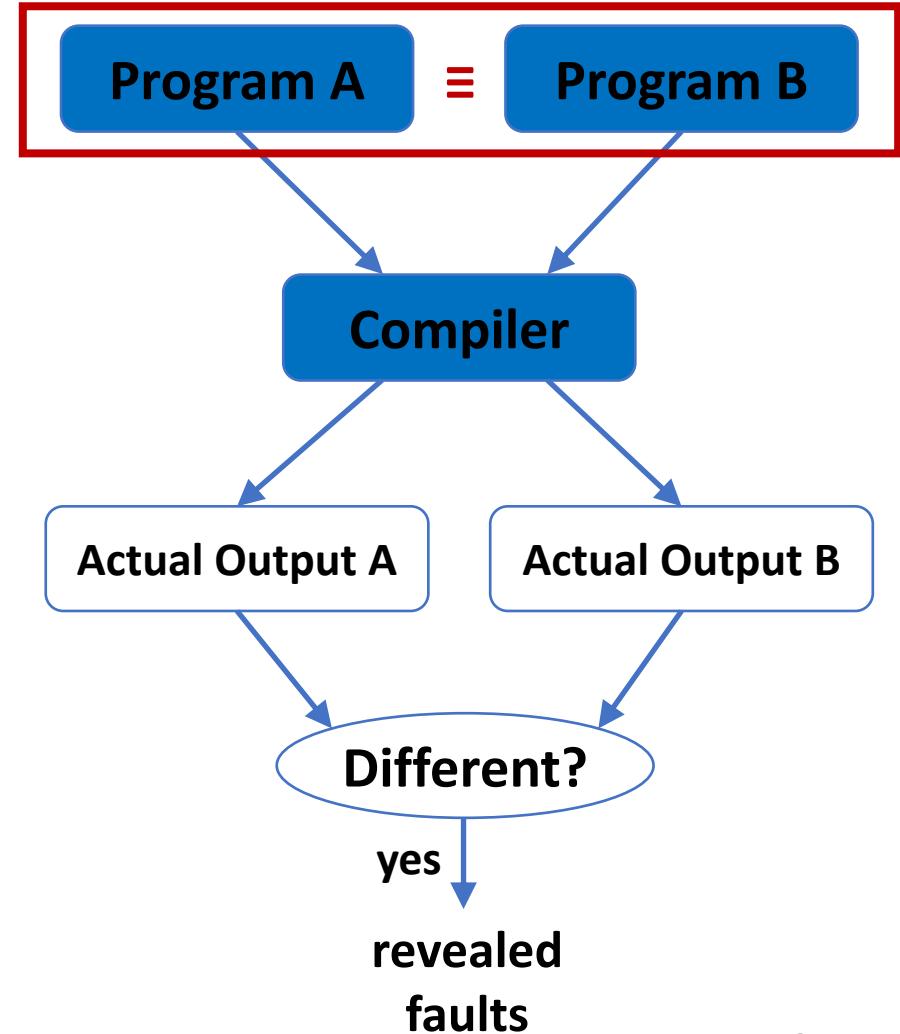
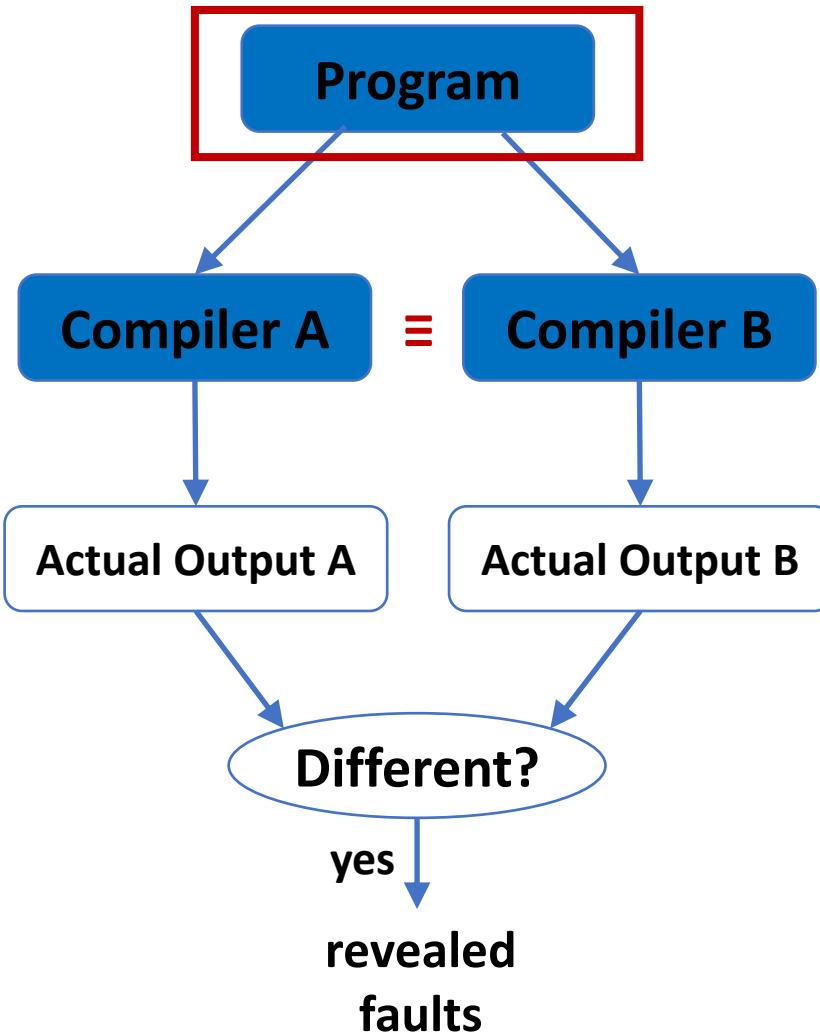
# Typical Software Testing



# Compiler Testing



# Differential Testing (Solving Oracle Problem)

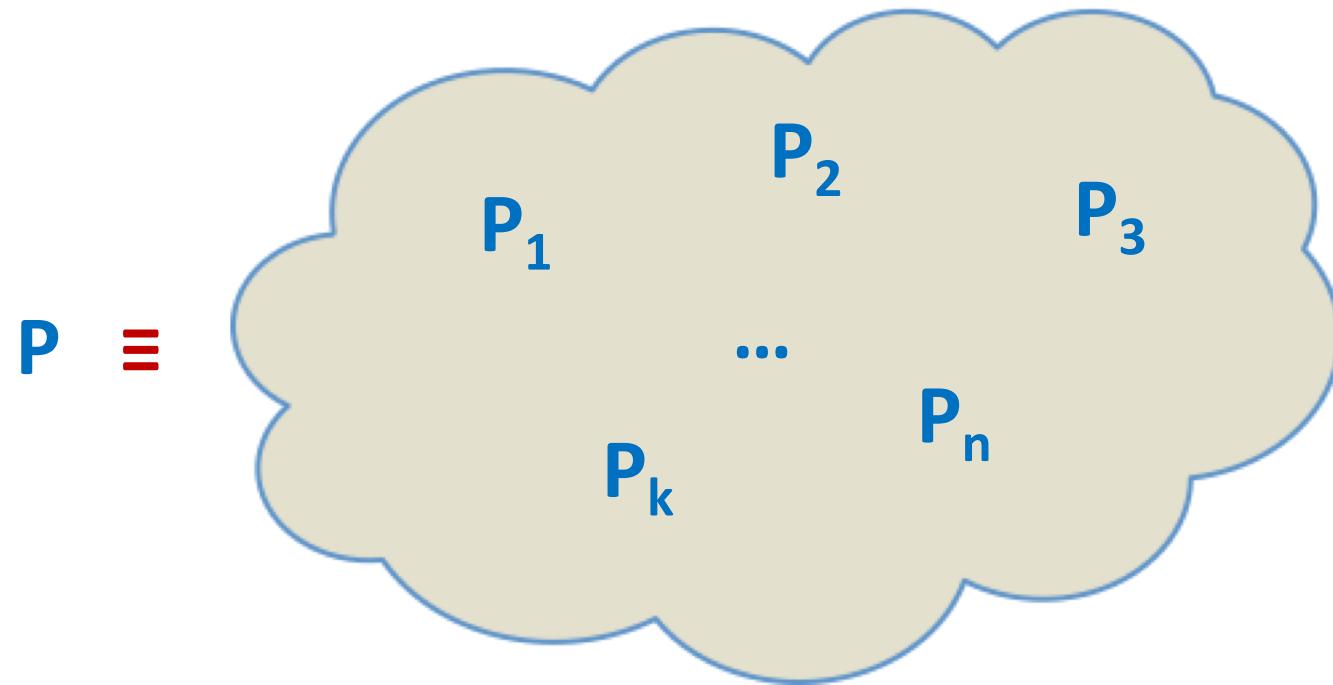


# Compiler Validation via Equivalence Modulo Inputs

Vu Le, Mehrdad Afshari, Zhendong Su

PLDI 2014, Distinguished Paper Award

# Vision for Equivalent Programs

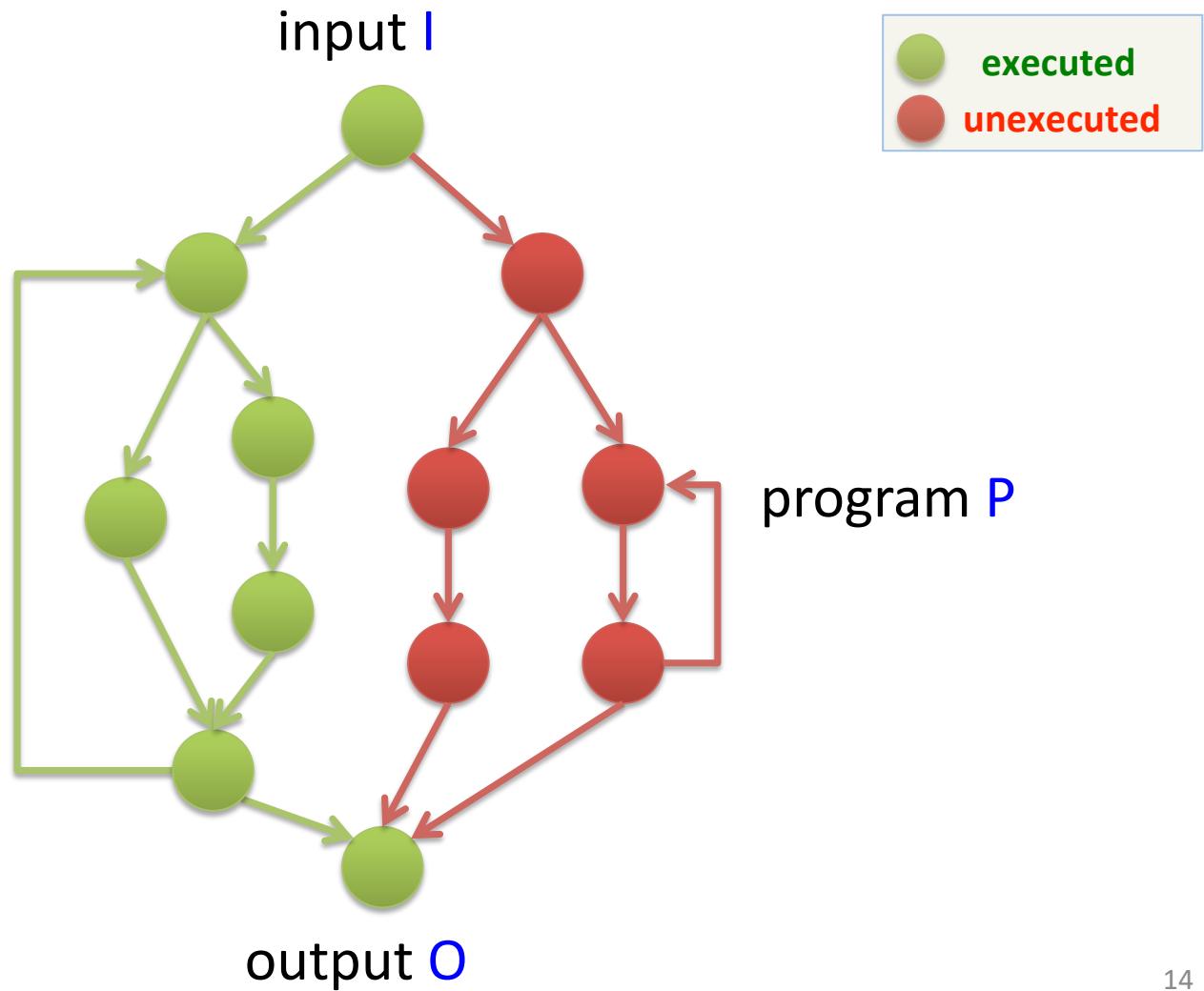


**Take existing real-world code and transform it into a different but equivalent variants of the original code**

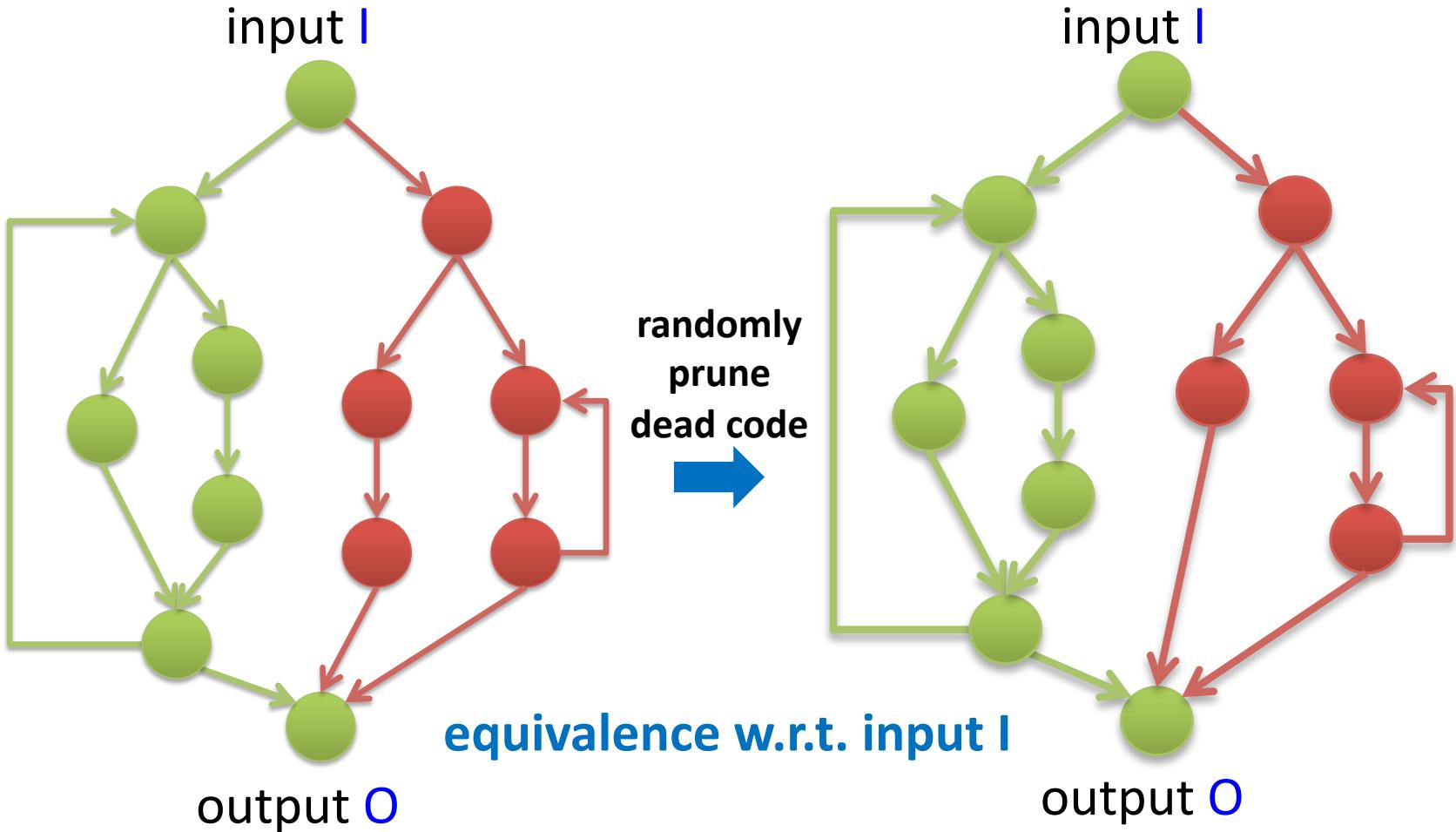
# Equivalence Modulo Inputs (EMI)

- Relax equivalence w.r.t. a **given input**
  - Variants must satisfy  $P(i) = P_k(i)$  on input  $i$
  - Variants may differ on other input  $j$ :  $P(j) \neq P_k(j)$
- Exploit the **interplay** between
  - **Dynamic** program execution on **some input**
  - **Static** compilation for **all inputs**

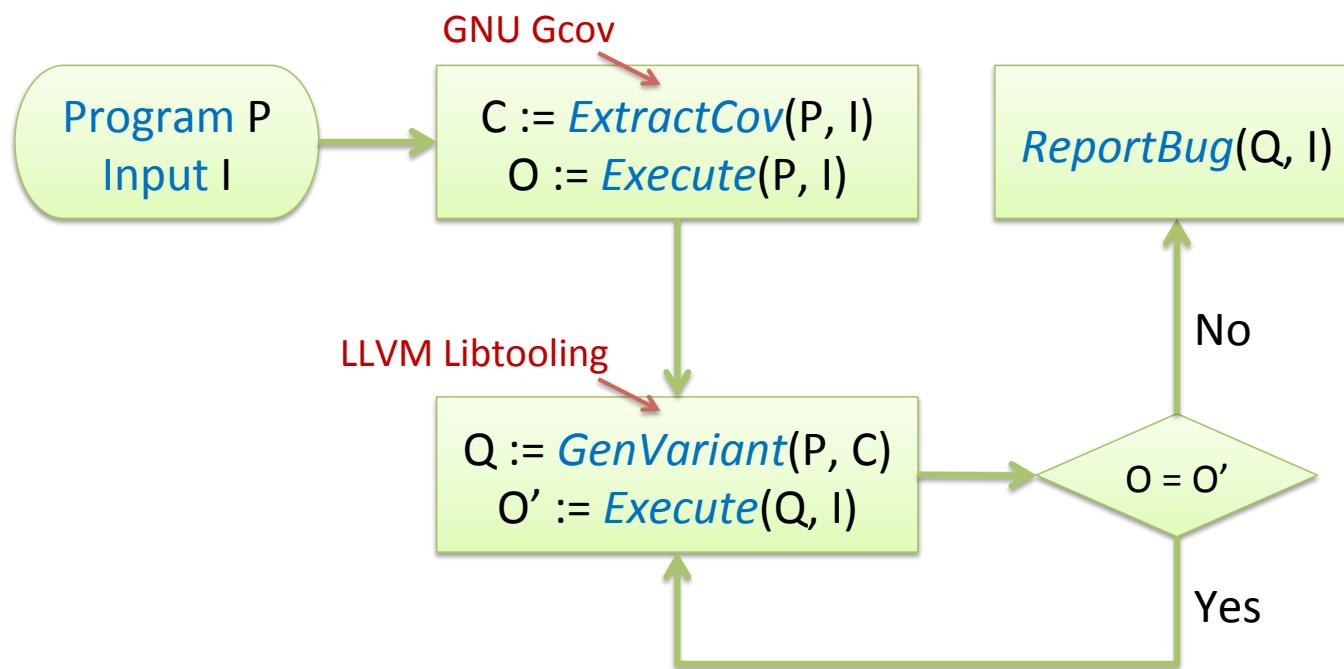
# Profile



# Mutation



# Orion Overview



# Examples

```
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) abort();
    if (x.d != 20) abort();
    if (x.e != 30) abort();
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) abort();
    if (z.c != 12) abort();
    if (z.d != 22) abort();
    if (z.e != 32) abort();
    if (l != 123) abort();
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

unexecuted

```
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) /* deleted */;
    if (x.d != 20) abort();
    if (x.e != 30) /* deleted */;
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) /* deleted */;
    if (z.c != 12) abort();
    if (z.d != 22) /* deleted */;
    if (z.e != 32) abort();
    if (l != 123) /* deleted */;
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

GCC test suite

```
clang -m32 -O0 test.c ; ./a.out
clang -m32 -O1 test.c ; ./a.out
```

```
clang -m32 -O0 test.c ; ./a.out
clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

# Examples (cont.)

```
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        for (; c++)
            for (;;) {
                b++;
                e = a > 2147483647 - b;
                if (d) break;
            }
    return 0;          unexecuted
}
```

Generated by CSmith

```
gcc -O0 test.c ; ./a.out
gcc -O3 test.c ; ./a.out
```

```
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        for (; c++)
            for (;;) {
                e = a > 2147483647 - b;
                if (d) break;
            }
    return 0;
}
```

```
gcc -O0 test.c ; ./a.out
gcc -O3 test.c ; ./a.out
^C
```

# Why Effective?

- Compilers must produce correct code for **all inputs**
- Variants have different data & control flow
  - Exercise **various** optimization strategies
  - Demand exact **same output** on the given input

# Evaluation

- Two multi-core Ubuntu machines
- April 2013 – March 2014
- Seed programs
  - Compiler test suites
  - Open-source projects
  - CSmith-generated programs

# Bug Counts and Types

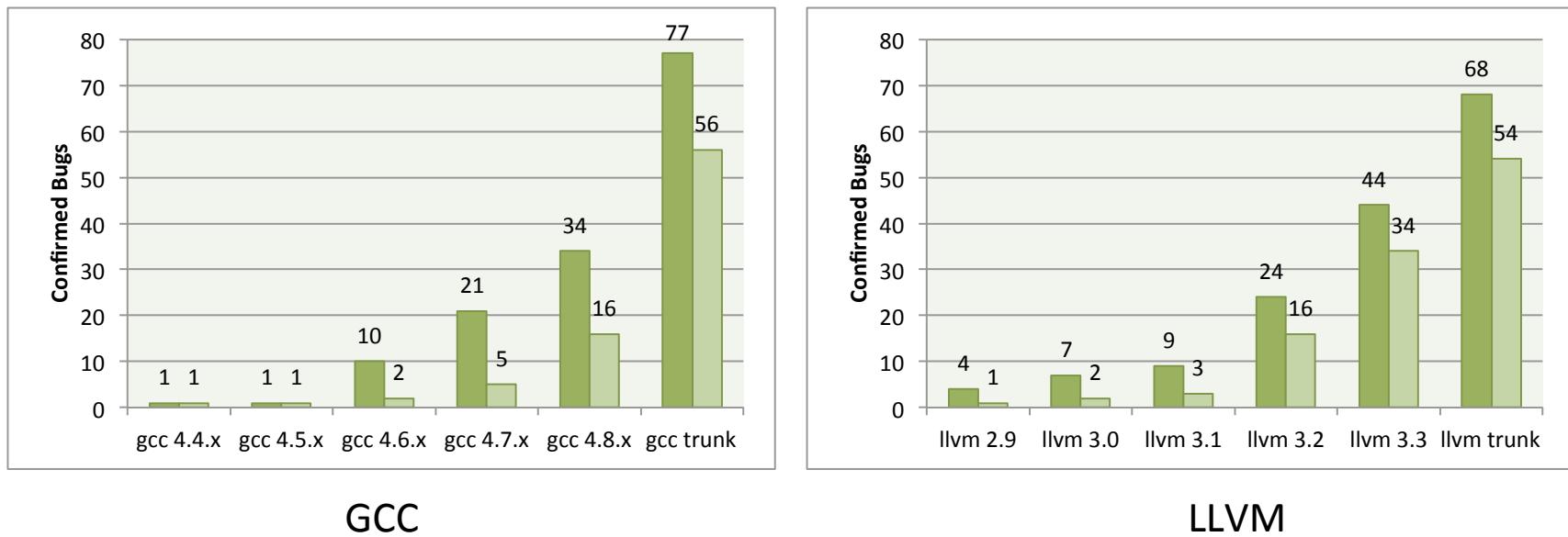
**Bug Counts**

|                  | <b>GCC</b> | <b>LLVM</b> | <b>TOTAL</b> |
|------------------|------------|-------------|--------------|
| Reported         | 111        | 84          | <b>195</b>   |
| Marked Duplicate | 28         | 7           | <b>35</b>    |
| Confirmed        | 79         | 68          | <b>147</b>   |
| Fixed            | 56         | 54          | <b>110</b>   |

**Bug Types**

|             | <b>GCC</b> | <b>LLVM</b> | <b>TOTAL</b> |
|-------------|------------|-------------|--------------|
| Wrong code  | 46         | 49          | <b>95</b>    |
| Crash       | 23         | 10          | <b>33</b>    |
| Performance | 10         | 9           | <b>19</b>    |

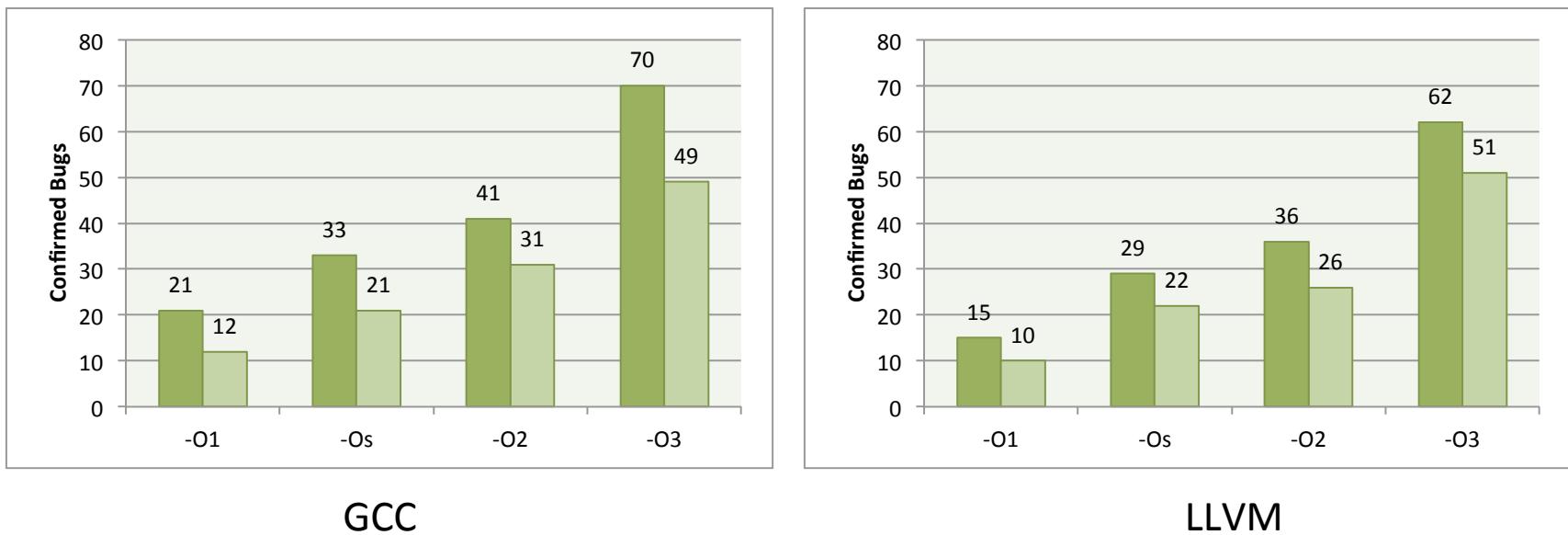
# Affected Versions



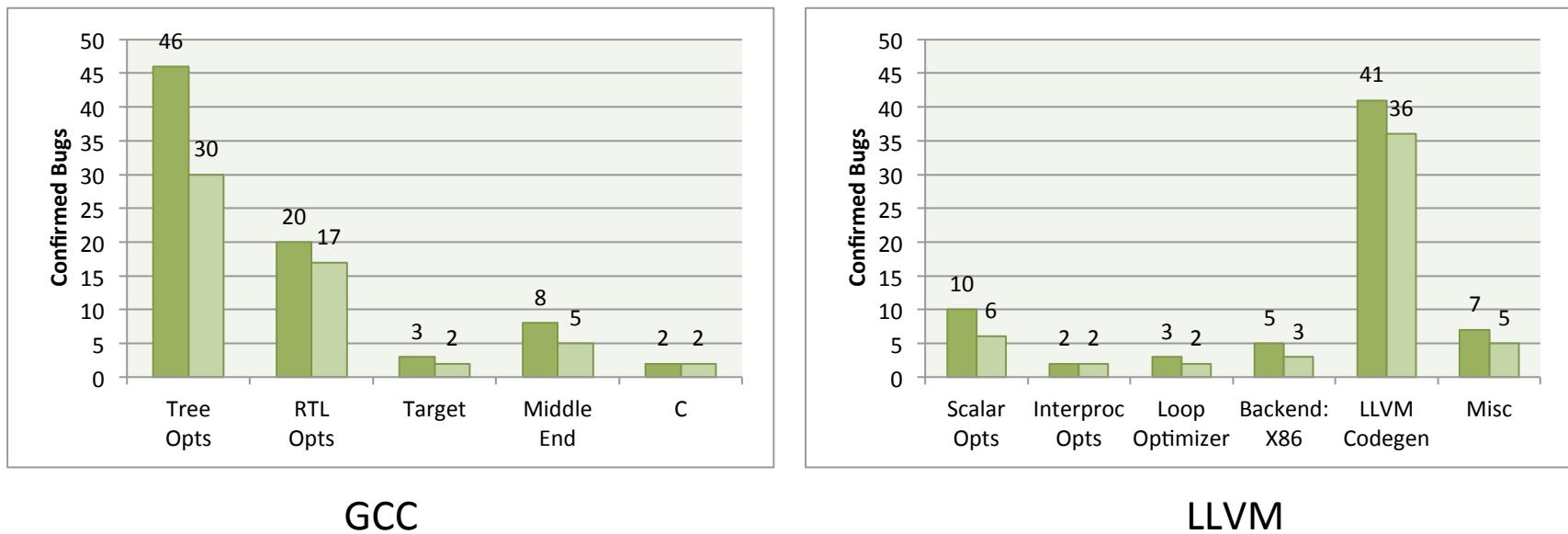
GCC

LLVM

# Affected Optimization Levels



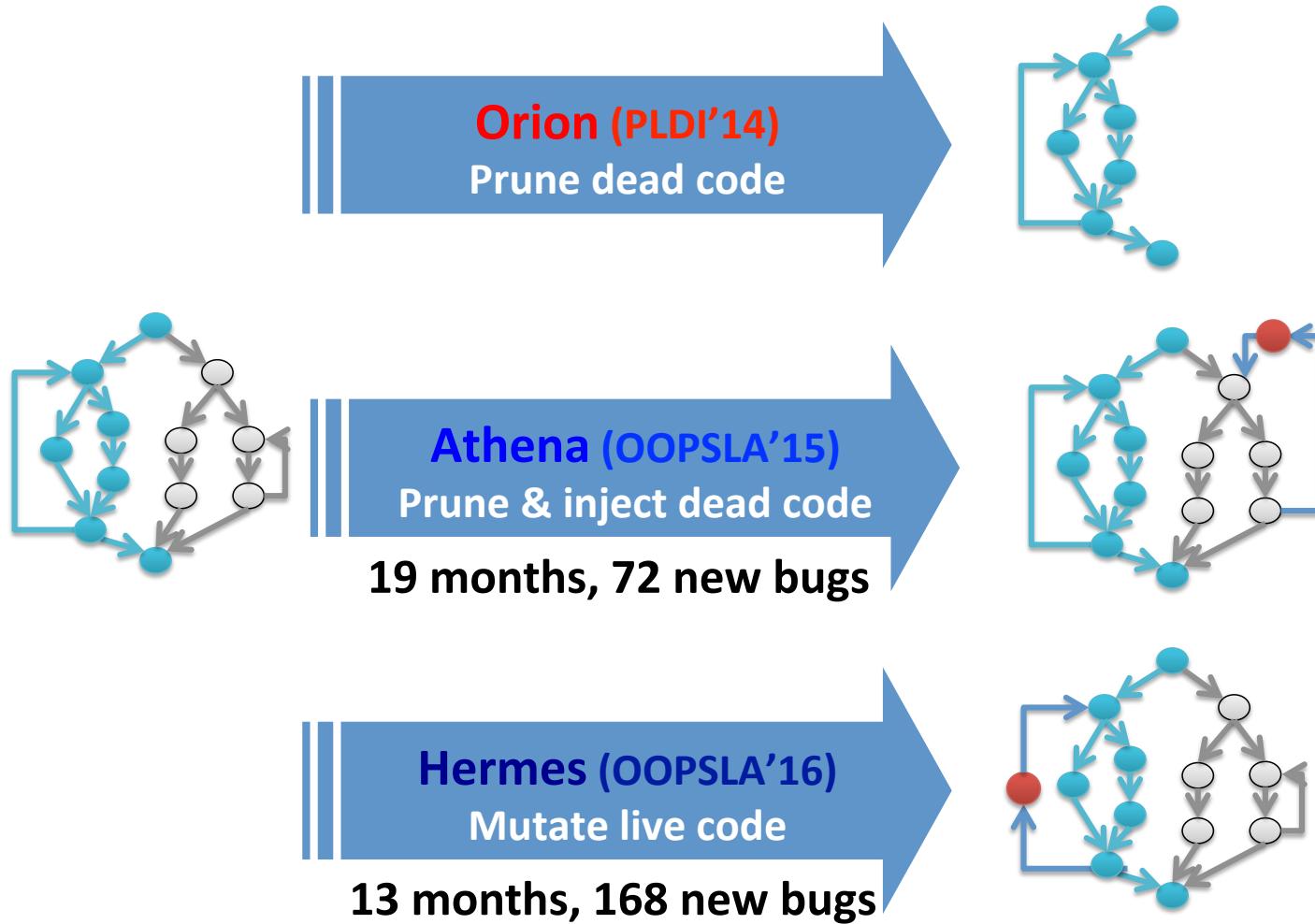
# Affected Components



# Conclusions

- EMI is **general** and **widely applicable**
  - Can test compilers, analysis and transformation tools
  - Generates real-world tests
  - Requires no reference compilers
- Orion is very **effective**
  - Has uncovered **200+ bugs** in GCC and LLVM
  - Majority of the bugs were **miscompilations**
- Plan to investigate other **mutation strategies**

# More Mutation Strategies



# Bug Counts (till Recently)

|          | <b>GCC</b> | <b>LLVM</b> | <b>TOTAL</b> |
|----------|------------|-------------|--------------|
| Reported | 643        | 498         | <b>1141</b>  |
| Fixed    | 397        | 239         | <b>636</b>   |

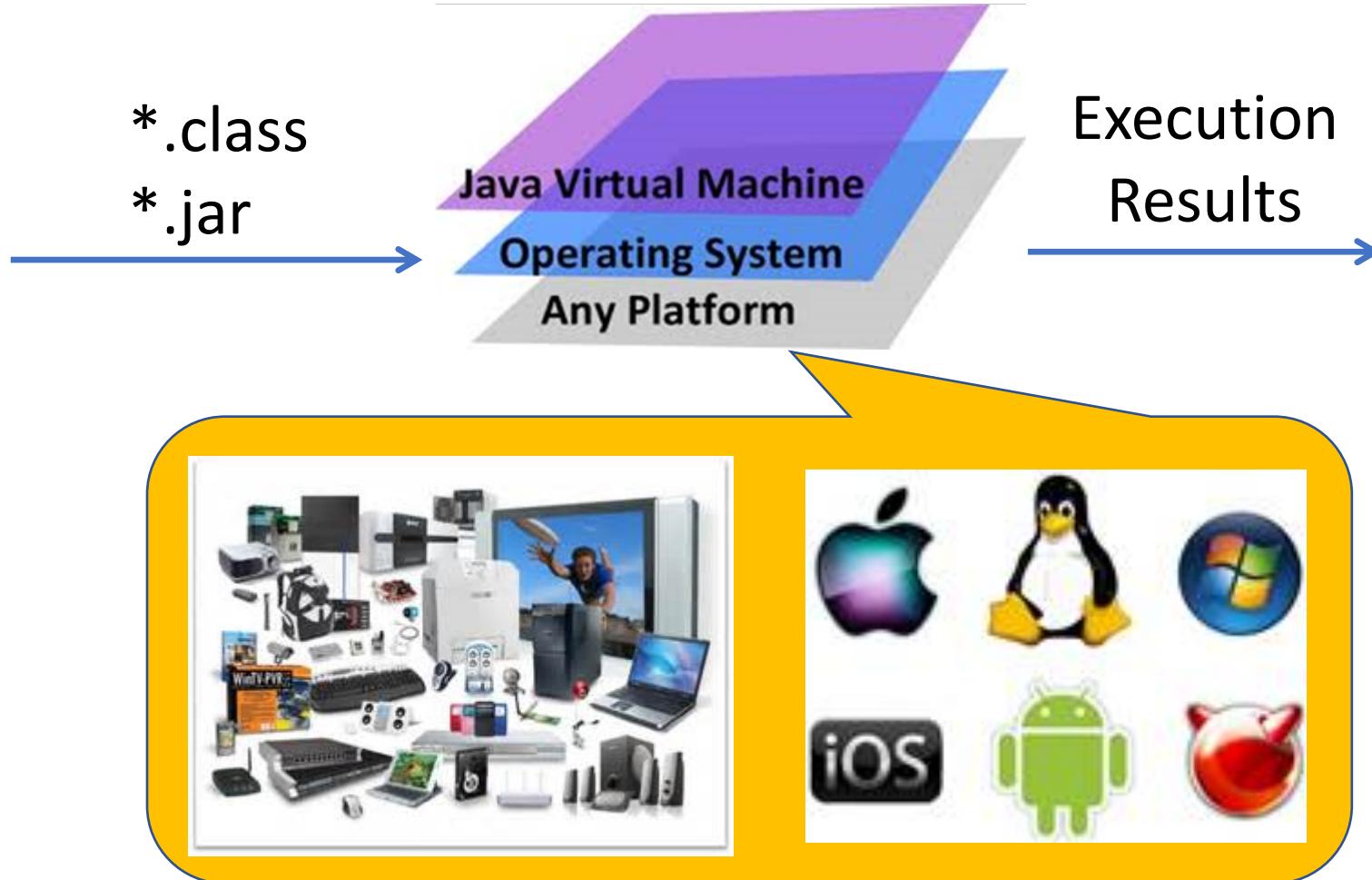
# Coverage-Directed Differential Testing of JVM Implementations

Yuting Chen, Ting Su, Chengnian Sun, Zhendong Su, Jianjun Zhao

PLDI 2016

# Java Virtual Machine (JVM)

Oracle's HotSpot, IBM's J9, Jikes RVM, Azul's Zulu, and GNU's GIJ



# Reflect on This Trust – JVM Discrepancy 1

```
1 ...  
2 MD5 checksum 8  
3 ...  
4 class M1436188543  
5 minor version: 0  
6 major version: 51  
7 flags: ACC_SUPER  
8 Constant pool:  
9 ...  
10 #7 = Utf8 <clinit>  
11 #8 = Utf8 ()V  
12 #9 = Class #19 // java/lang/System  
13 #10 = Utf8 Code  
14 #11 = Utf8 main  
15 ...  
16 {  
17     public abstract {};  
18     flags: ACC_PUBLIC, ACC_ABSTRACT  
19     public static void main(java.lang.String[]);  
20     flags: ACC_PUBLIC, ACC_STATIC  
21     Code:  
22         stack=2, locals=1, args_size=1  
23         0: getstatic #12 // Field java/lang/System.  
             out:Ljava/io/PrintStream;  
24         3: ldc #4 // String Completed!  
25         5: invokevirtual #21 // Method java/io/  
             PrintStream.println:(Ljava/lang/String  
             ;)V  
26         8: return}
```

**clinit method**

**public abstract {};**

- HotSpot takes it as an **ordinary** method
- J9 reports a **class format error**

**Cause:** the JVM specification says that “other methods named <clinit> in a class file are **of no consequence**”



A class or interface initialization method needs to be strictly defined

# Reflect on This Trust – JVM Discrepancy 2

```
1 //The Jimple code of M1433982529
2 public class M1433982529 extends java.lang.
3     Object{
4         protected void internalTransform(java.lang.
5             String) {
6             java.util.Map r0;
7             r0 := @parameter0: java.util.Map;
8             staticinvoke <java.lang.Object: boolean
9                 getBoolean(java.util.Map)>(r0);
10            return;
11        }
12    }
```

GIJ throws a **verification error** because it is unsafe to perform a type casting between `java.lang.String` and `java.util.Map`, while HotSpot does not report any error



JVMs take their own classfile verification and type checking policies

# Reflect on This Trust – JVM Discrepancy 3

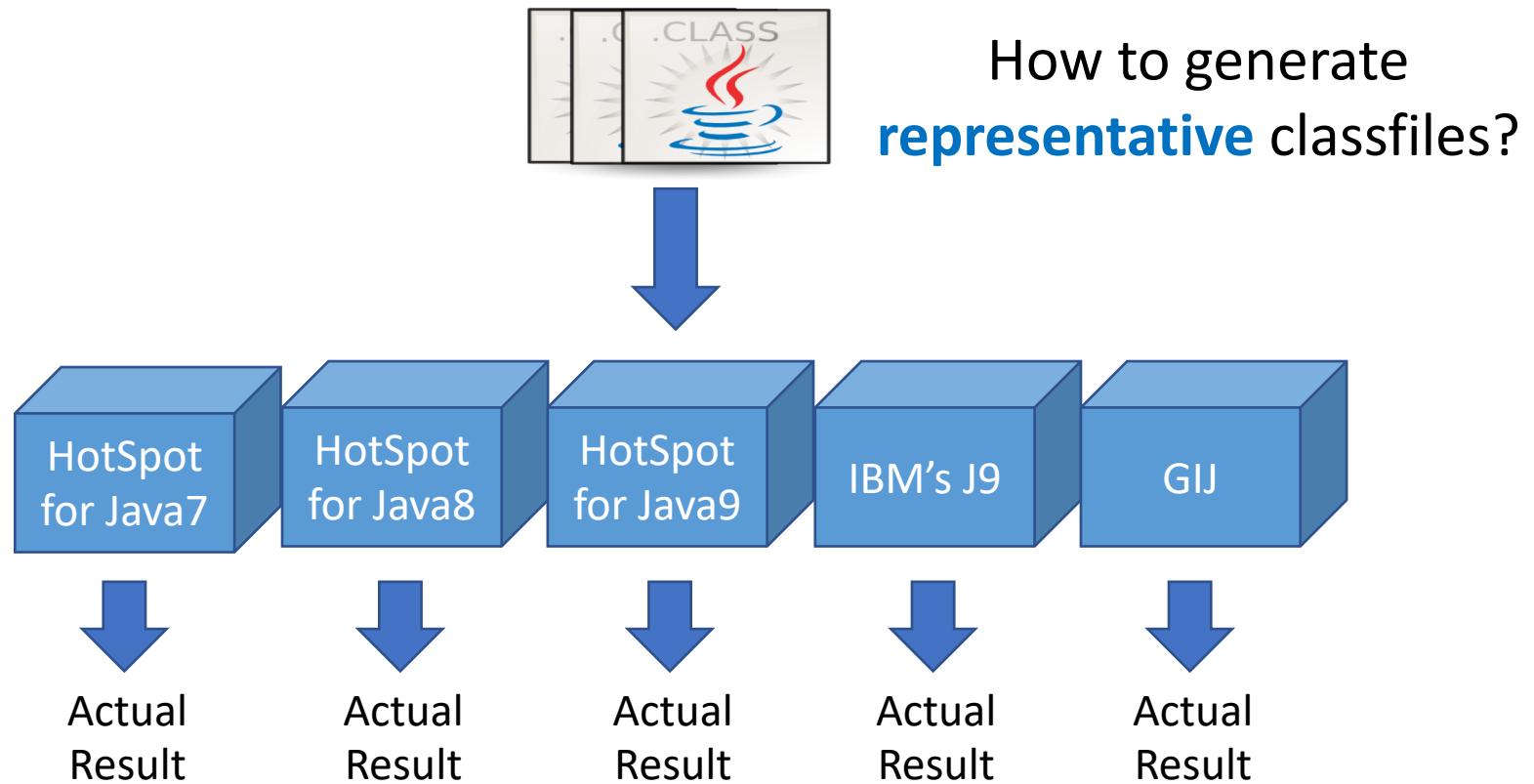
```
1 //The source code of sun.java2d.pisces.  
2 PiscesRenderingEngine  
3 public class PiscesRenderingEngine extends  
4     RenderingEngine {  
5     ...  
6     private static enum NormMode {OFF, ON_NO_AA  
7         , ON_WITH_AA};  
8     ...  
9 }  
//The Jimple code of M1437121261  
10 public class M1437121261 {  
11     public static void main (String[] r0)  
12         throws sun.java2d.pisces.  
13             PiscesRenderingEngine$2{  
14     ...  
15 } }
```

HotSpot reports a `java.lang.IllegalAccessException`, while J9 and GIJ do not



JVMs are not compatible to access some classes

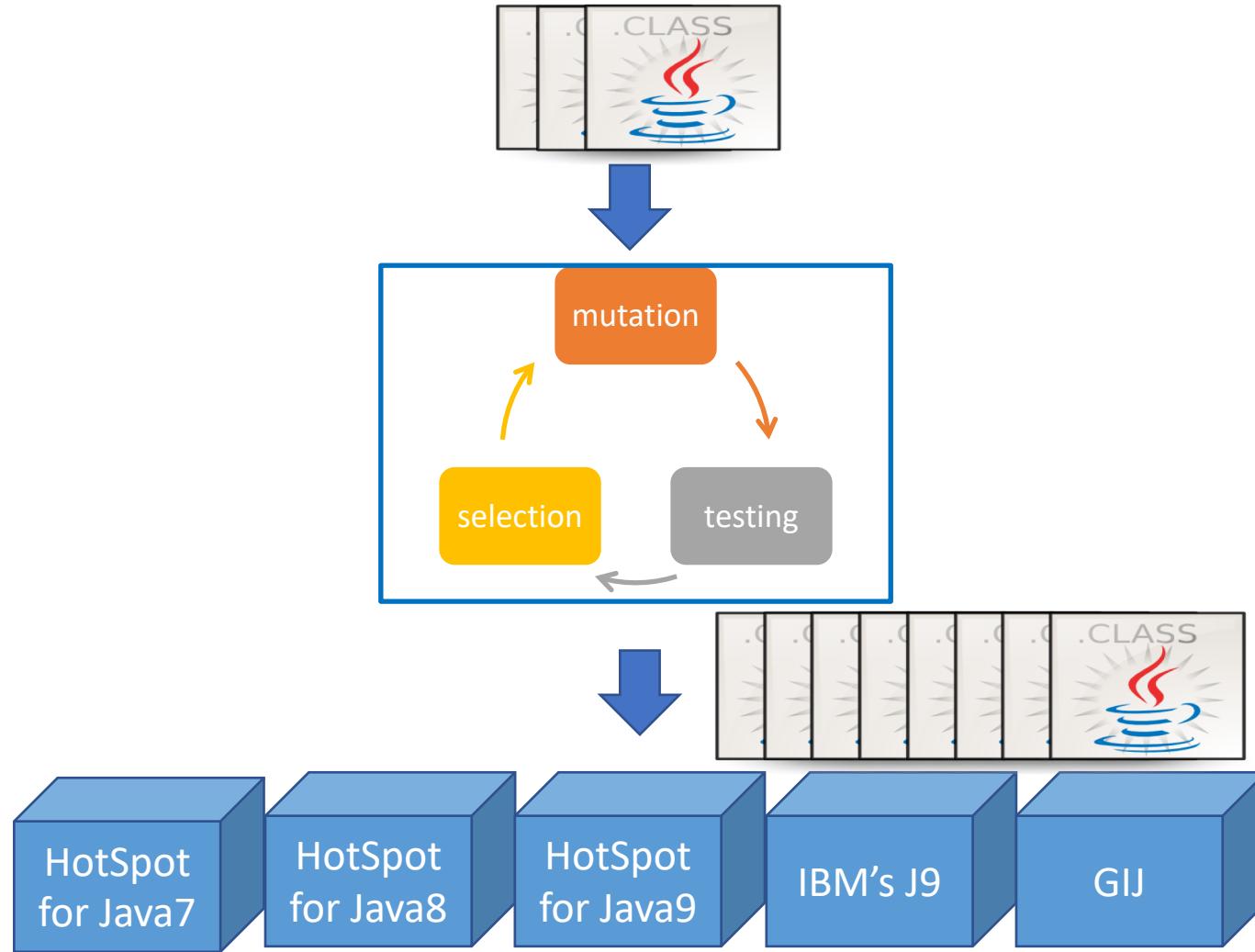
# Differential JVM Testing



# Two Key Observations

- A classfile can encompass intricate constraints
  - Corner cases can be created through mutating seeds
- Equivalence class partition removes redundancy
  - ECP works only if we can decide whether two tests belong to the same partition

# ClassFuzz Overview

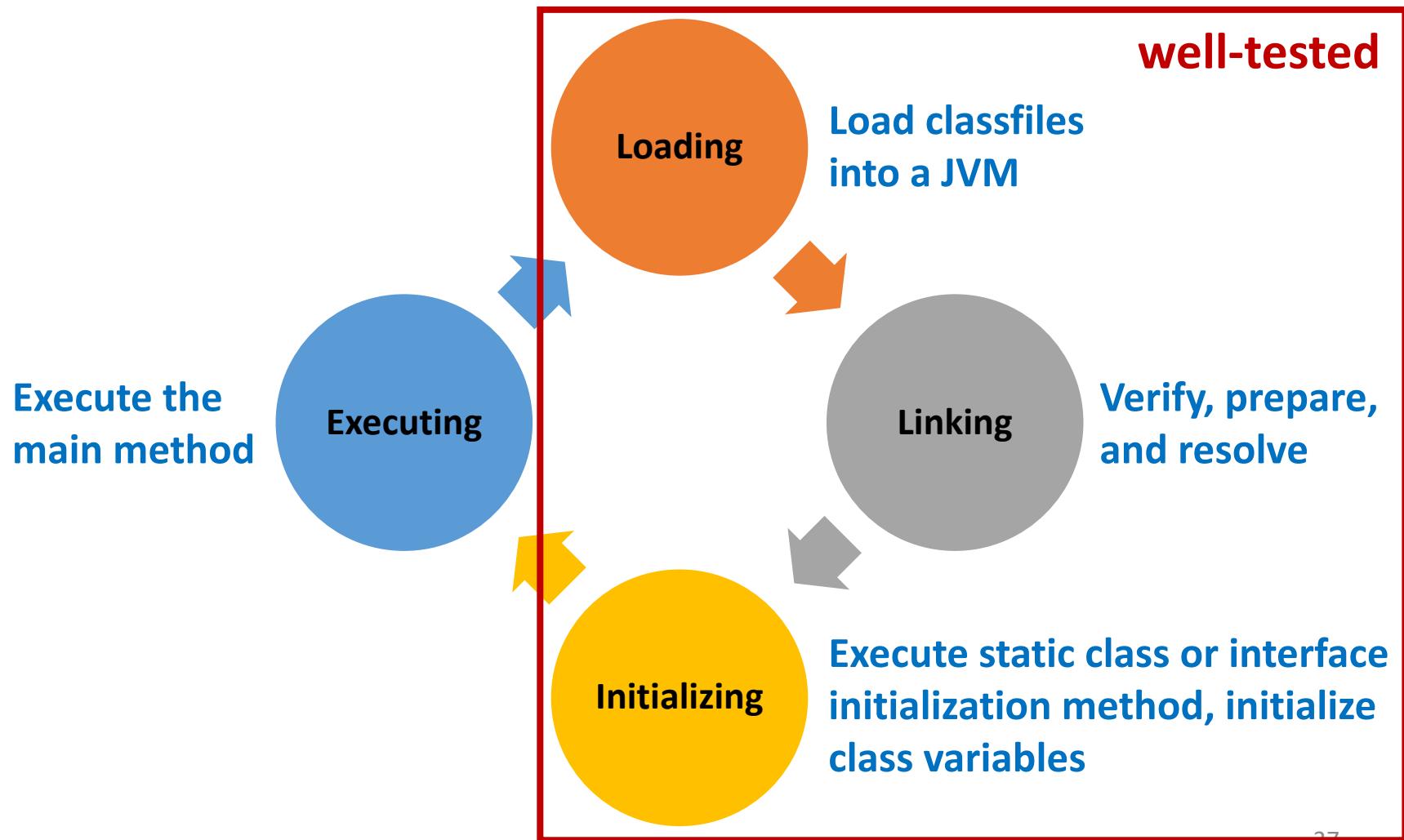


# Mutating Classfiles

| What to mutate | Typical mutators   | Example (Jimple code before mutation<br>→ Jimple code after mutation)  |
|----------------|--|--|
| Class          | Reset its attributes ( <i>e.g.</i> , modifiers, name, package name, superclass), <i>etc.</i>   | class M1437185190 extends java.lang.Object<br>→ <b>private</b> class M1437185190 extends java.lang.Thread  |
| Interface      | Insert one or more class-implementing interfaces, delete one or more interfaces, <i>etc.</i>   | class M1437185190 extends java.lang.Object<br>→ class M1437185190 extends java.lang.Object<br><b>implements</b> java.security.PrivilegedAction   |
| Field          | Insert one or more class fields, delete one or more fields, choose a field and set its attribute(s) ( <i>e.g.</i> , name, modifiers), <i>etc.</i>  | ...protected final java.util.Map MAP; ...<br>→ ...protected final java.util.Map MAP;<br><b>public</b> java.lang.Object MAP; ...  |
| Method         | Insert one or more class methods, delete one or more methods, choose a method and set its attribute(s) ( <i>e.g.</i> , name, modifiers, return type), <i>etc.</i>                                | public void <init>(){...}<br>→ public <b>static</b> void <init>(){...}   |
| Exception      | Select a method and insert one or more exceptions thrown, delete one or more exceptions thrown, <i>etc.</i>  | public static void main(java.lang.String[]){<br>→ public static void main(java.lang.String[])<br>throws sun.java2d.pisces.PiscesRenderingEngine\$2 {   |
| Parameter      | Select a method and insert one or more parameters, delete one or more parameters, <i>etc.</i>  | public static void main(java.lang.String[])<br>→ public static void main( <b>java.lang.Object</b> ,<br>java.lang.String[])   |
| Local variable | Select a method and insert one or more local variables, delete one or more local variables, choose a local variable and set its attribute(s) ( <i>e.g.</i> , type, name, modifiers), <i>etc.</i> | public static void main(java.lang.String[]){<br>int \$i0; ...}<br>→ public static void main(java.lang.String[]){<br><b>java.lang.String</b> \$i0; ...}   |
| Jimple file    | Insert one or more program statements, delete one or more program statements, <i>etc.</i>  | r0:=parameter0: java.lang.String[];<br>r1:=<java.lang.System: java.io.PrintStream out>;<br>virtualinvoke \$r1.<java.io.PrintStream:void<br>println(java.lang.String)>("Executed");<br>→ r0:=parameter0: java.lang.String[];<br>virtualinvoke \$r1.<java.io.PrintStream:void<br>println(java.lang.String)>("Executed");<br>r1:=<java.lang.System: java.io.PrintStream out>; |

129 mutators

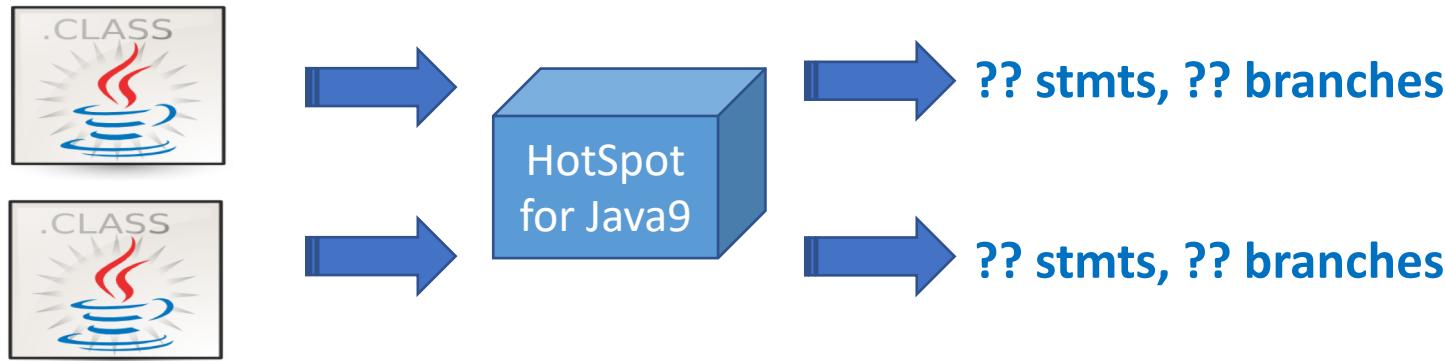
# JVM Startup Process



# Selecting Mutators

- **Fact:** mutators are designed arbitrarily; and some are effective, while some others are not effective
- **Goal:** create representative classfiles as many and fast as possible
- **Heuristic:** the more number of representative classfiles have been created by a mutator, the more chance should be given for that mutator

# Selecting Representative Classfiles



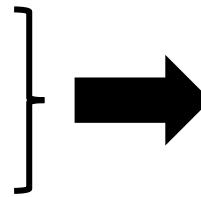
- Several comparison criteria are introduced
  - Statement coverage
  - Branch coverage
  - Trace coverage

# Differential Testing of JVMs

- check whether the main method can be normally invoked or during which phase an error/exception is thrown
  - (0) “normally invoked”
  - (1) “rejected during the loading phase”
  - (2) “rejected during the linking phase”
  - (3) “rejected during the initialization phase”
  - (4) “rejected at runtime”

# Evaluation Setup

- Coverage collection
  - HotSpot for Java9
  - GCOV + LCOV



At each run the coverage can be conveniently collected

HotSpot (**260K** LOCs)  
Cost for cov. analysis: **30+ mins**

share/vm/classfile/ (**11977** LOCs)  
Cost for cov. analysis: **90 secs**

- Seeds
  - 1216 classfiles in JRE 7
- Target JVMs

| JVM implementation     | Java version supported | Description  | Release date  |
|------------------------|------------------------|--|---------------|
| HotSpot for Java 9 [3] | 1.9.0-internal         | Open-source reference implementation of the JVM specification, Java SE 9 Edition | Late 2016     |
| HotSpot for Java 8 [3] | 1.8.0                  | Open-source reference implementation of the JVM specification, Java SE 8 Edition | March 2014    |
| HotSpot for Java 7 [3] | 1.7.0                  | Open-source reference implementation of the JVM specification, Java SE 7 Edition | July 2011     |
| J9 for IBM SDK8 [4]    | 1.8.0                  | IBM's fully compatible JVM with the libraries of Oracle's Java SE 8 platform     | February 2015 |
| GIJ 5.1.0 [2]          | 1.5.0                  | The GNU Interpreter for Java (GIJ), a Java bytecode interpreter                  | April 2015    |

# Evaluated Setup (cont.)

- ClassFuzz supplemented with a uniqueness analysis
  - [st], [stbr], [tr]
- Randfuzz, Greedyfuzz, Uniquefuzz

|                     | ClassFuzz        | randfuzz | greedyfuzz | uniquefuzz |
|---------------------|------------------|----------|------------|------------|
| Mutation-based      | ✓                | ✓        | ✓          | ✓          |
| Cov. Analysis       | ✓                | ✗        | ✓          | ✓          |
| Uniqueness Analysis | [st] [stbr] [tr] | ✗        | [stbr]     | [stbr]     |
| Mutator Selection   | ✓                | ✗        | ✗          | ✗          |

# How Many Test Classfiles can be Generated?

|  | Classfuzz |       |       | Uniquefuzz | Greedyfuzz | Randfuzz |
|--|-----------|-------|-------|------------|------------|----------|
|  | [stbr]    | [st]  | [tr]  |            |            |          |
| #iterations                                  | 2130      | 2108  | 1971  | 1898       | 1911       | 46318    |
| GenClasses                                   | 1539      | 1543  | 1450  | 1436       | 1432       | 29523    |
| TestClasses                                  | 898       | 494   | 774   | 628        | 98         | 29523    |
| succ   | 42.2%     | 23.4% | 39.3% | 33.1%      | 5.1%       | 63.7%    |
| Average time for each generated class (sec.) | 168.4     | 168.0 | 178.8 | 180.5      | 181.0      | 8.78     |
| Average time for each test class (sec.)      | 288.6     | 524.7 | 334.9 | 412.7      | 2644.9     | 8.78     |

$$\text{succ} = |\text{testClasses}| / \#iterations * 100\%$$

# How Effective are the Test Classfiles?

|                                | JRE7<br>classes | seeding<br>classfiles | Classfuzz |       |       | Uniquefuzz | Greedyfuzz | Randfuzz |
|--------------------------------|-----------------|-----------------------|-----------|-------|-------|------------|------------|----------|
|                                |                 |                       | [stbr]    | [st]  | [tr]  |            |            |          |
| GenClasses                     | 21736           | 1216                  | 1539      | 1543  | 1450  | 1436       | 1432       |          |
| all invoked                    | 77              | 17                    | 197       | 199   | 226   | 203        | 195        |          |
| all rejected at the same stage | 21295           | 1162                  | 1100      | 1104  | 984   | 1063       | 980        | —        |
| Discrepancies                  | 364             | 37                    | 242       | 240   | 240   | 170        | 257        |          |
| Distinct_Discrepancies         | 16              | 9                     | 17        | 14    | 13    | 10         | 15         |          |
| diff                           | 1.7%            | 3.0%                  | 15.7%     | 15.6% | 16.5% | 11.8%      | 17.9%      |          |
| TestClasses                    |                 |                       | 898       | 494   | 774   | 628        | 98         | 29523    |
| all invoked                    |                 |                       | 121       | 62    | 122   | 109        | 11         | 3914     |
| all rejected at the same stage |                 |                       | 670       | 374   | 564   | 462        | 72         | 20072    |
| Discrepancies                  | —               | —                     | 107       | 58    | 88    | 57         | 15         | 5537     |
| Distinct_Discrepancies         |                 |                       | 17        | 11    | 12    | 9          | 7          | 14       |
| diff                           |                 |                       | 11.9%     | 11.7% | 11.4% | 9.1%       | 15.3%      | 18.8%    |

$$\text{diff} = |\text{Discrepancies}| / |\text{TestClasses}| * 100\%$$

# Can the Test Classfiles Find JVM Defects?

- 62 new JVM discrepancies

```
1 ...  
2 MD5 checksum 8  
3 fb69050bbcb9a83ddd90ae393368c5e  
4 ...  
5 class M1436188543  
6 minor version: 0  
7 major version: 51  
8 flags: ACC_SUPER  
9 Constant pool:  
10 ...  
11 #7 = Utf8 <clinit>  
12 #8 = Utf8 ()V  
13 #9 = Class #19 // java/lang/System  
14 #10 = Utf8 Code  
15 #11 = Utf8 main  
16 rename abstract method as <clinit>  
17 {  
18     public abstract {};  
19     flags: ACC_PUBLIC, ACC_ABSTRACT  
20     public static void main(java.lang.String[]);  
21     flags: ACC_PUBLIC, ACC_STATIC  
22     Code:  
23         stack=2, locals=1, args_size=1  
24         0: getstatic #12 // Field java/lang/System.  
25             out:Ljava/io/PrintStream;  
26         3: ldc #4 // String Completed!  
27         5: invokevirtual #21 // Method java/io/  
28             PrintStream.println:(Ljava/lang/String  
29             ;)V  
30         8: return  
31 }
```

set the parameter type from Map to String

```
1 //The Jimple code of M143982929  
2 public class M143982929 extends java.lang.  
3     Object{  
4         protected void internalTransform(java.lang.  
5             String) {  
6             java.util.Map r0;  
7             r0 := @parameter0: java.util.Map;  
8             staticinvoke <java.lang.Object: boolean  
9                 getBoolean(java.util.Map)>(r0);  
10            return;  
11        }  
12    }
```

add a throw exception to the main method

```
1 //The source code of sun.java2d.pisces.  
2 PiscesRenderingEngine  
3 public class PiscesRenderingEngine extends  
4     RenderingEngine {  
5     ...  
6     private static enum NormMode {OFF, ON_NO_AA  
7         , ON_WITH_AA};  
8     ...  
9     }  
10    //The Jimple code of M1437121261  
11    public class M1437121261 {  
12        public static void main (String[] r0)  
13            throws sun.java2d.pisces.  
14                PiscesRenderingEngine$2{  
15                    ...  
16                }  
17            }
```

# Conclusions

- **Problem**
  - Testing JVMs requires painstaking effort in designing **test classfiles** along with their **test oracles**
- **Proposal: coverage-directed fuzz testing**
  - **Test classfile generation**
    - Mutating classes and selectively applying mutators
    - Deciding the representativeness of a classfile mutant
  - **Differential JVM testing**

# Reading Materials

- V. Le, M. Afshari, and Z. Su. Compiler validation via equivalence modulo inputs. In PLDI, page 216-226, 2014.
- V. Le, C. Sun, and Z. Su. Finding deep compiler bugs via guided stochastic program mutation. In OOPSLA, pages 386– 399, 2015.
- C. Sun, V. Le, and Z. Su. Finding compiler bugs via live code mutation. In OOPSLA, pages 849–863, 2016.
- Q. Zhang, C. Sun, and Z. Su. Skeletal program enumeration for rigorous compiler testing. In PLDI, page 347-361, 2017.
- X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. In PLDI, pages 283–294, 2011.
- Y. Chen, A. Groce, C. Zhang, W. Wong, X. Fern, E. Eide, and J. Regehr. Taming compiler fuzzers. In PLDI, pages 197–208, 2013.
- C. Sun, V. Le, and Z. Su. Finding and analyzing compiler warning defects. In ICSE, pages 203–213, 2016.
- Y. Chen, T. Su, C. Sun, Z. Su, and J. Zhao. Coverage-directed differential testing of JVM implementations. In PLDI, page 85-99, 2016.

# Q&A?

Bihuan Chen, Pre-Tenure Assoc. Prof.

[bhchen@fudan.edu.cn](mailto:bhchen@fudan.edu.cn)

<https://chenbihuan.github.io>