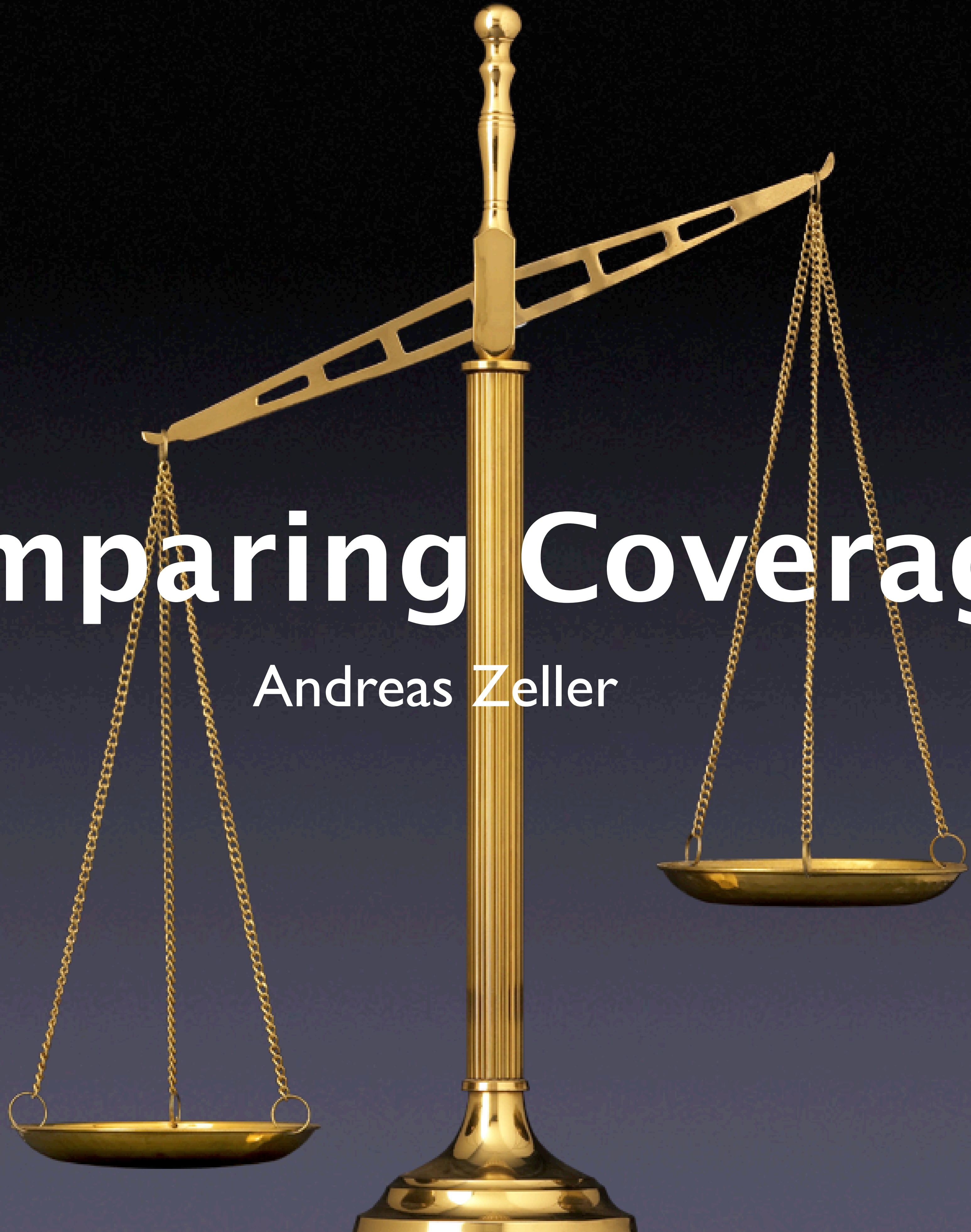
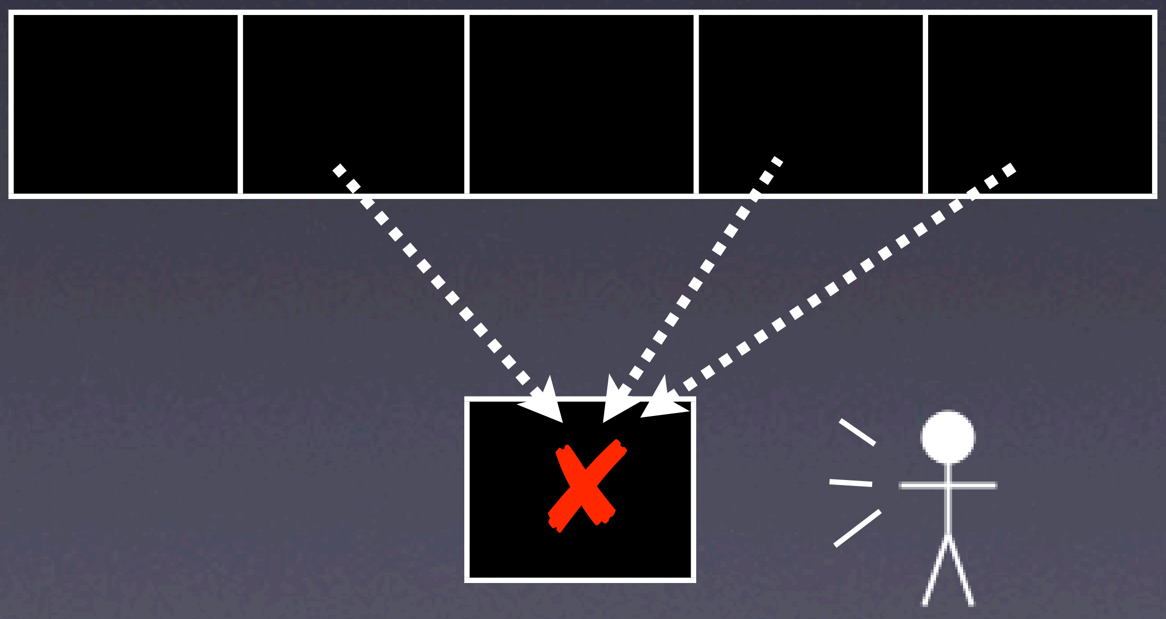
Comparing Coverage

Andreas Zeller

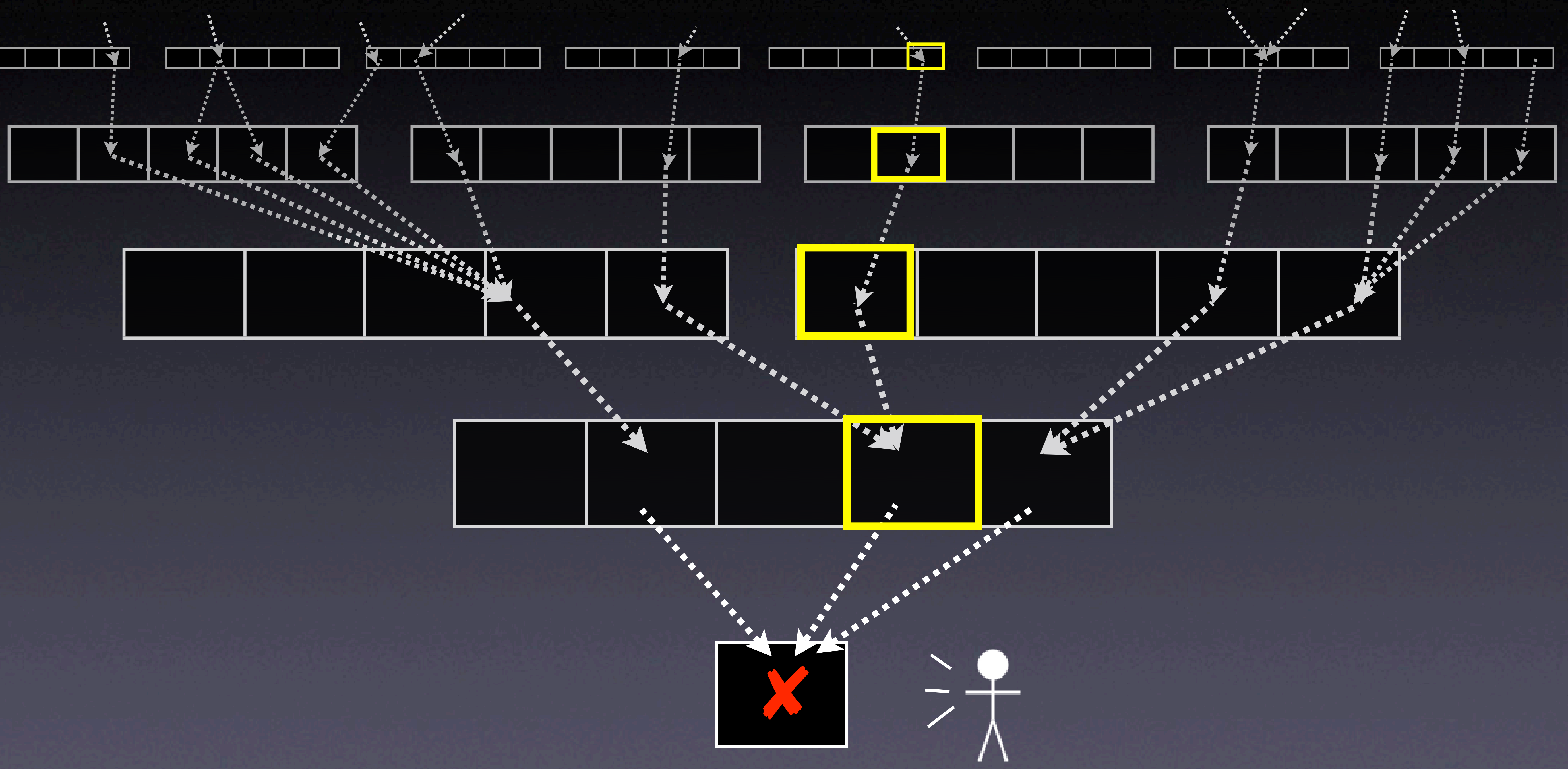


Tracing Infections

- For every infection, we must find the *earlier infection* that *causes* it.
- Which origin should we focus upon?

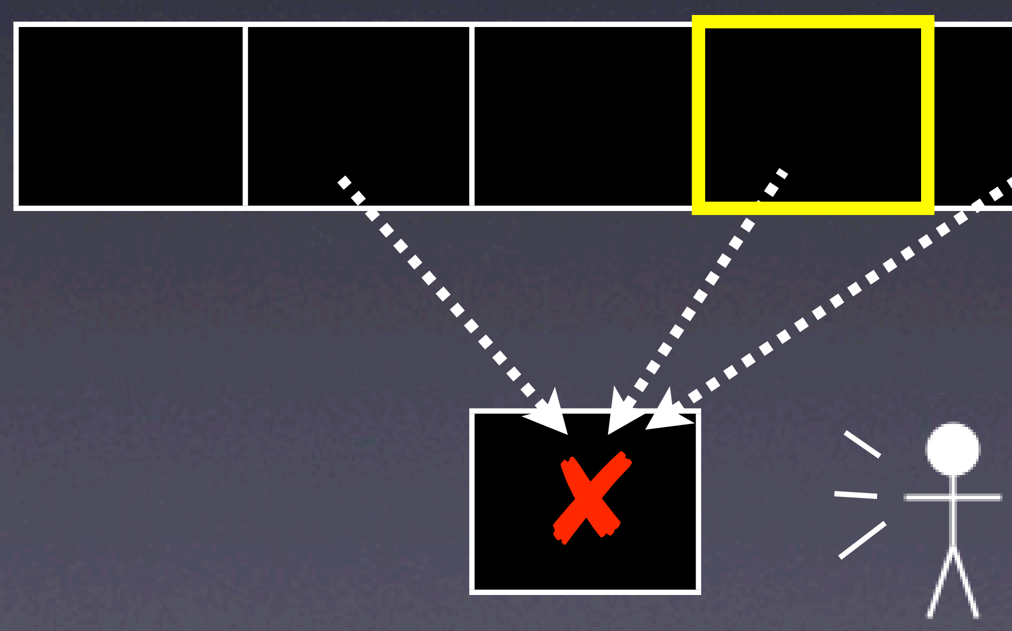


Tracing Infections



Focusing on Anomalies

- Examine origins and locations where something *abnormal* happens



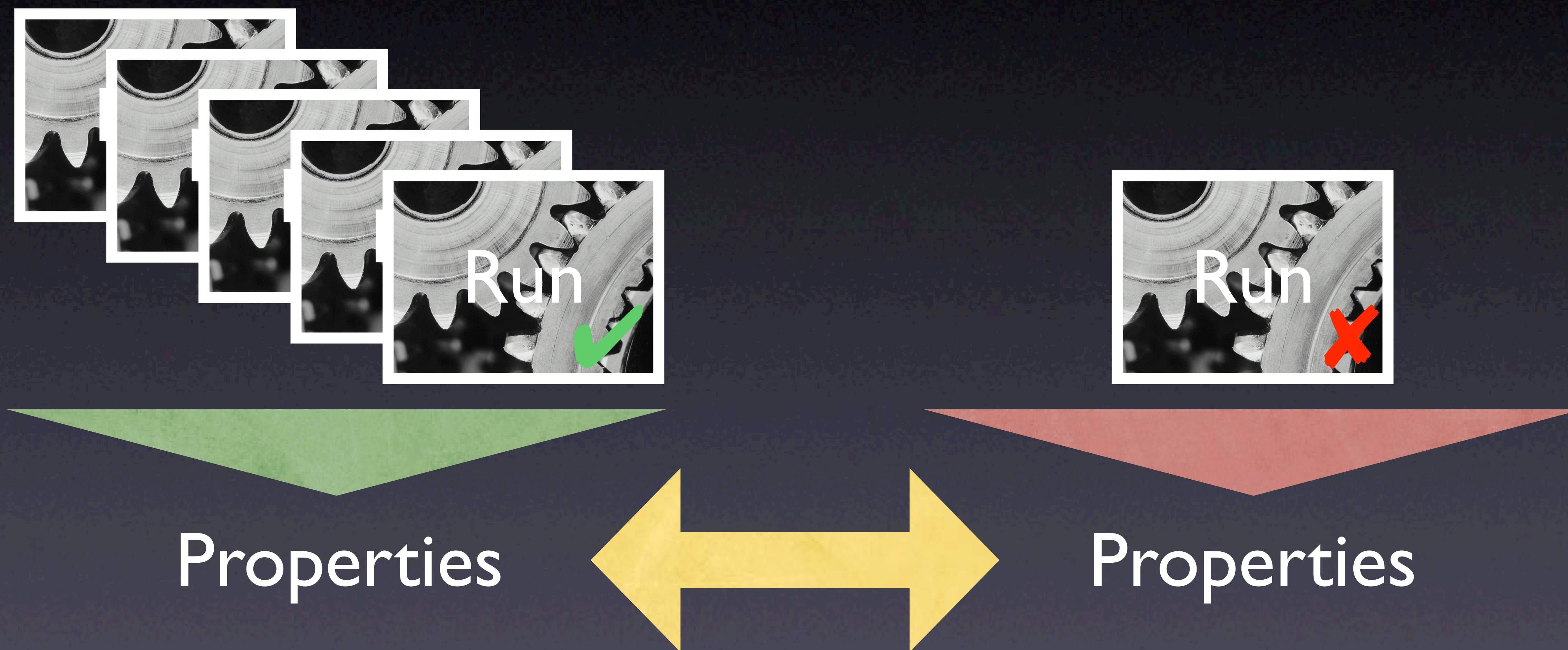
What’s normal?

- General idea: Use *induction* – reasoning from the particular to the general
- Start with a *multitude* of runs
- Determine *properties* that are common across all runs

What's abnormal?

- Suppose we determine common properties of all *passing* runs.
- Now we examine a run which *fails* the test.
- Any difference in properties *correlates with failure* – and is likely to hint at failure causes

Detecting Anomalies



Differences correlate with failure

Properties

Data properties that hold in all runs:

- "At f(), x is odd"
- "$0 \leq x \leq 10$ during the run"

Code properties that hold in all runs:

- "f() is always executed"
- "After open(), we eventually have close()"

Comparing Coverage

1. Every failure is caused by an infection, which in turn is caused by a defect
2. The defect must be *executed* to start the infection
3. Code that is executed *in failing runs only* is thus likely to cause the defect

The middle program

```
$ middle 3 3 5
middle: 3

$ middle 2 1 3
middle: 1
```

```
int main(int arc, char *argv[])
{
    int x = atoi(argv[1]);
    int y = atoi(argv[2]);
    int z = atoi(argv[3]);
    int m = middle(x, y, z);

    printf("middle: %d\n", m);

    return 0;
}
```

```
int middle(int x, int y, int z) {
    int m = z;
    if (y < z) {
        if (x < y)
            m = y;
        else if (x < z)
            m = y;
    } else {
        if (x > y)
            m = y;
        else if (x > z)
            m = x;
    }
    return m;
}
```

Obtaining Coverage

for C programs

Obtaining Coverage

for Python programs

```
if __name__ == "__main__":
    sys.settrace(tracer)
    x = sys.argv[1]
    y = sys.argv[2]
    z = sys.argv[3]
    m = middle(x, y, z)

    print "middle:", m
```

Obtaining Coverage

for Python programs

```
def tracer(frame, event, arg):
    code = frame.f_code
    function = code.co_name
    filename = code.co_filename
    line = frame.f_lineno
    print filename + ":" + `line` + \
        ":" + function + "():", \
        event, arg
    return tracer
```

Obtaining Coverage

for Python programs

```
$ ./middle.py 3 3 5
./middle.py:13:middle(): call None
./middle.py:14:middle(): line None
./middle.py:15:middle(): line None
./middle.py:16:middle(): line None
./middle.py:18:middle(): line None
./middle.py:19:middle(): line None
./middle.py:26:middle(): line None
./middle.py:26:middle(): return 3
middle: 3
```

For remaining steps,
see new project

x	3	1	3	5	5	2
y	3	2	2	5	3	1
z	5	3	1	5	4	3
`int middle(int x, int y, int z) {`	●	●	●	●	●	●
`    int m = z;`	●	●	●	●	●	●
`    if (y < z) {`	●	●	●	●	●	●
`        if (x < y)`		●				
`            m = y;`		●				
`        else if (x < z)`	●				●	●
`            m = y;`	●					●
`    } else {`	●		●	●		
`        if (x > y)`			●			
`            m = y;`			●			
`        else if (x > z)`						
`            m = x;`						
`    }`						
`    return m;`	●	●	●	●	●	●
`}`	✔	✔	✔	✔	✔	✘

Discrete Coloring

executed only in failing runs
highly suspect

executed in passing and failing runs
ambiguous

executed only in passing runs
likely correct

x	3	1	3	5	5	2
y	3	2	2	5	3	1
z	5	3	1	5	4	3
`int middle(int x, int y, int z) {`	●	●	●	●	●	●
`    int m = z;`	●	●	●	●	●	●
`    if (y < z) {`	●	●	●	●	●	●
`        if (x < y)`		●				
`            m = y;`		●				
`        else if (x < z)`	●				●	●
`            m = y;`	●					●
`    } else {`	●		●	●		
`        if (x > y)`			●			
`            m = y;`			●			
`        else if (x > z)`						
`            m = x;`						
`    }`						
`    return m;`	●	●	●	●	●	●
`}`	✔	✔	✔	✔	✔	✘

	x	3	1	3	5	5	2
	y	3	2	2	5	3	1
	z	5	3	1	5	4	3
int middle(int x, int y, int z) {		●	●	●	●	●	●
int m = z;		●	●	●	●	●	●
if (y < z) {		●	●	●	●	●	●
if (x < y)			●				
m = y;			●				
else if (x < z)		●				●	●
m = y;		●					●
} else {		●		●	●		
if (x > y)				●			
m = y;				●			
else if (x > z)							
m = x;							
}							
return m;		●	●	●	●	●	●
}		✔	✔	✔	✔	✔	✘

Continuous Coloring

executed only in failing runs

passing and failing runs

executed only in passing runs

Hue

$$hue(s) = red\ hue + \frac{\%passed(s)}{\%passed(s) + \%failed(s)} \times hue\ range$$

0% passed  100% passed

Brightness

frequently executed

$$bright(s) = \max(\%passed(s), \%failed(s))$$

rarely executed

x	3	1	3	5	5	2
y	3	2	2	5	3	1
z	5	3	1	5	4	3
`int middle(int x, int y, int z) {`	●	●	●	●	●	●
`int m = z;`	●	●	●	●	●	●
`if (y < z) {`	●	●	●	●	●	●
`if (x < y)`		●				
`m = y;`		●				
`else if (x < z)`	●				●	●
`m = y;`	●					●
`} else {`	●		●	●		
`if (x > y)`			●			
`m = y;`			●			
`else if (x > z)`						
`m = x;`						
`}`						
`return m;`	●	●	●	●	●	●
`}`	✔	✔	✔	✔	✔	✘

Source: Jones et al., ICSE 2002

Source: Jones et al., ICSE 2002

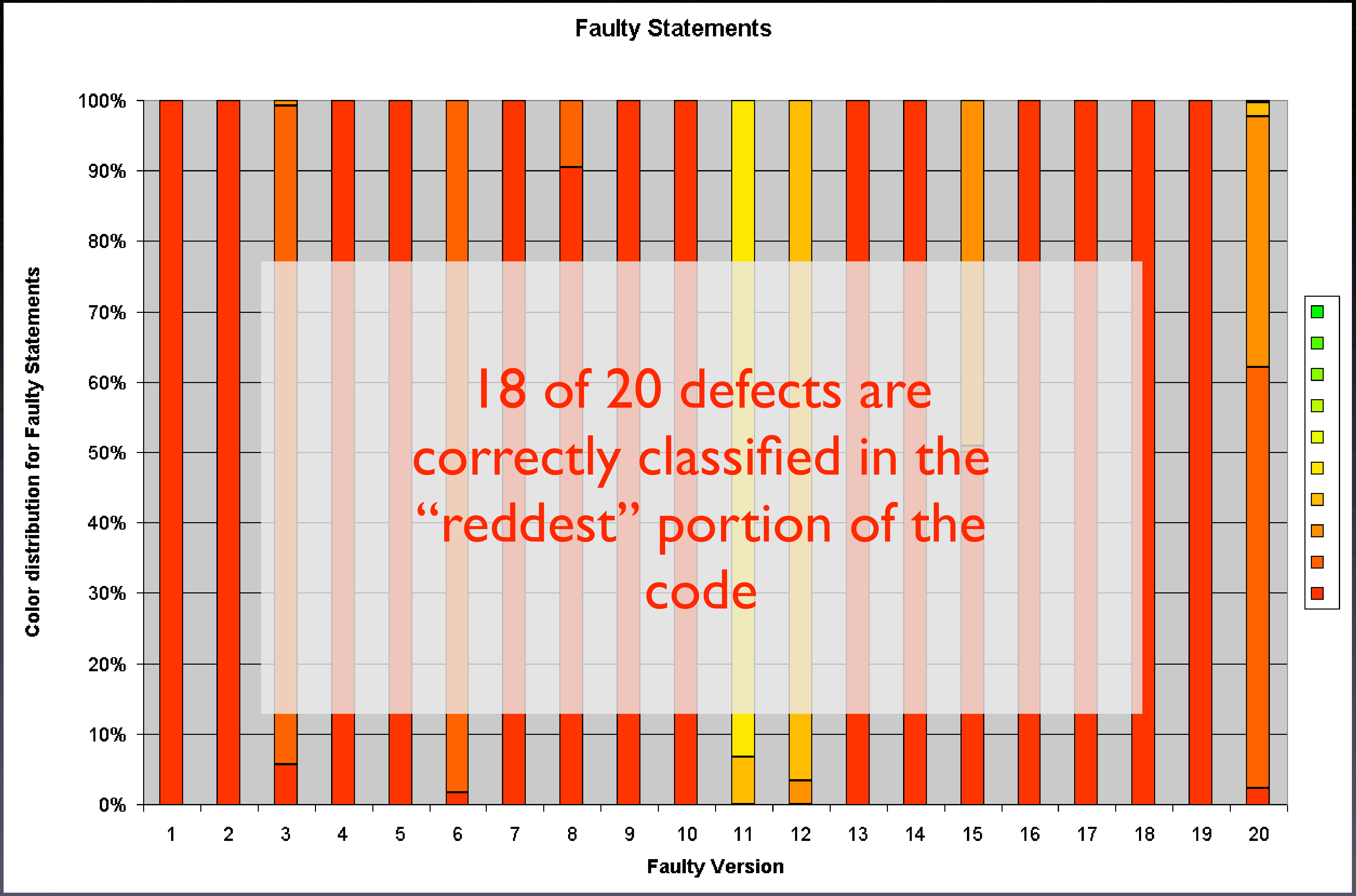
Evaluation

How well does comparing coverage detect anomalies?

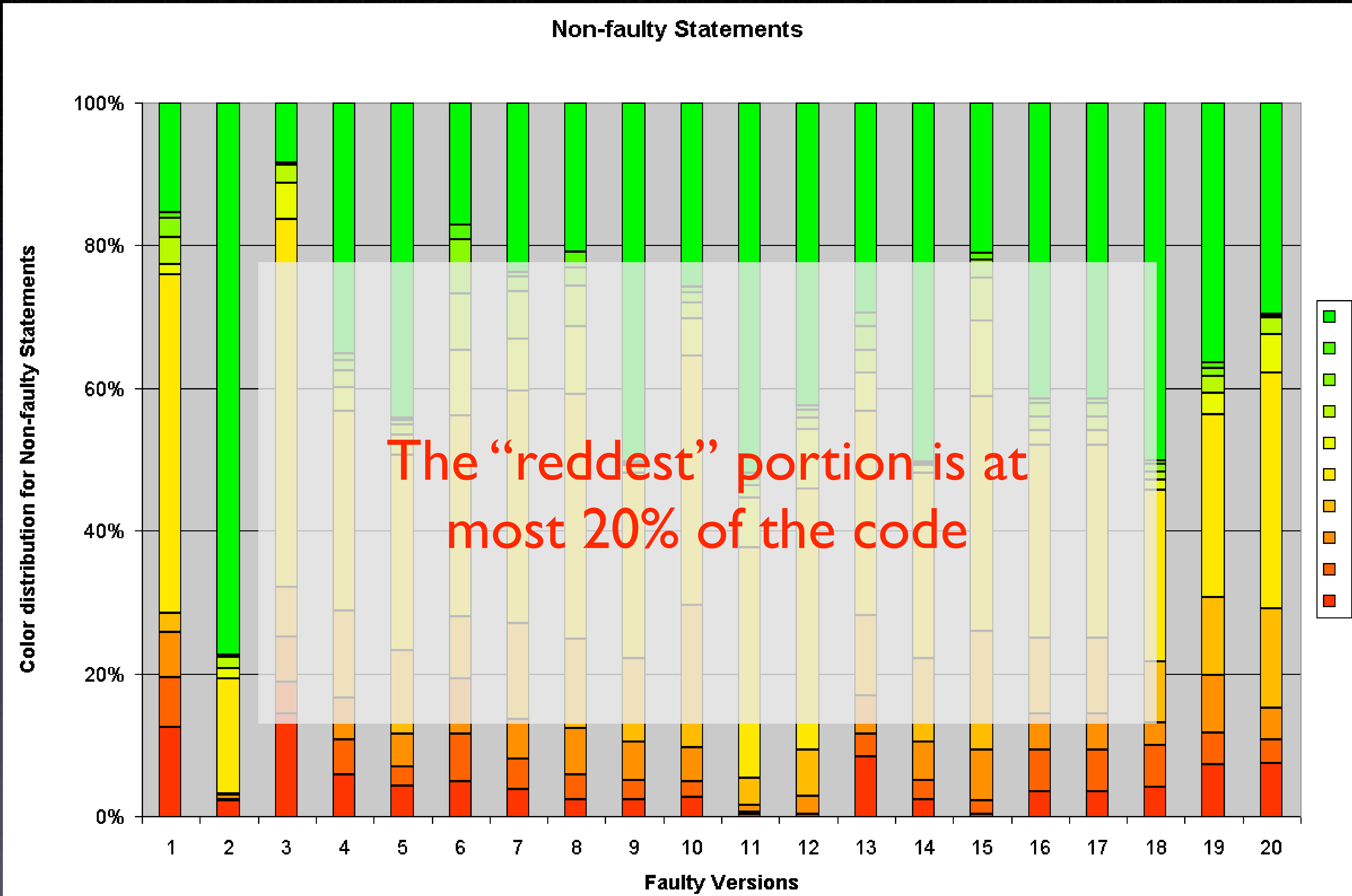
- How green are the defects? *(false negatives)*
- How red are non-defects? *(false positives)*

Space

- 8000 lines of executable code
- 1000 test suites with156–4700 test cases
- 20 defective versions with one defect each (corrected in subsequent version)



Source: Jones et al., ICSE 2002



Source: Jones et al., ICSE 2002

Siemens Suite

- 7 C programs, 170–560 lines
- 132 variations with one defect each
- 108 all yellow (i.e., useless)
- 1 with one red statement (at the defect)

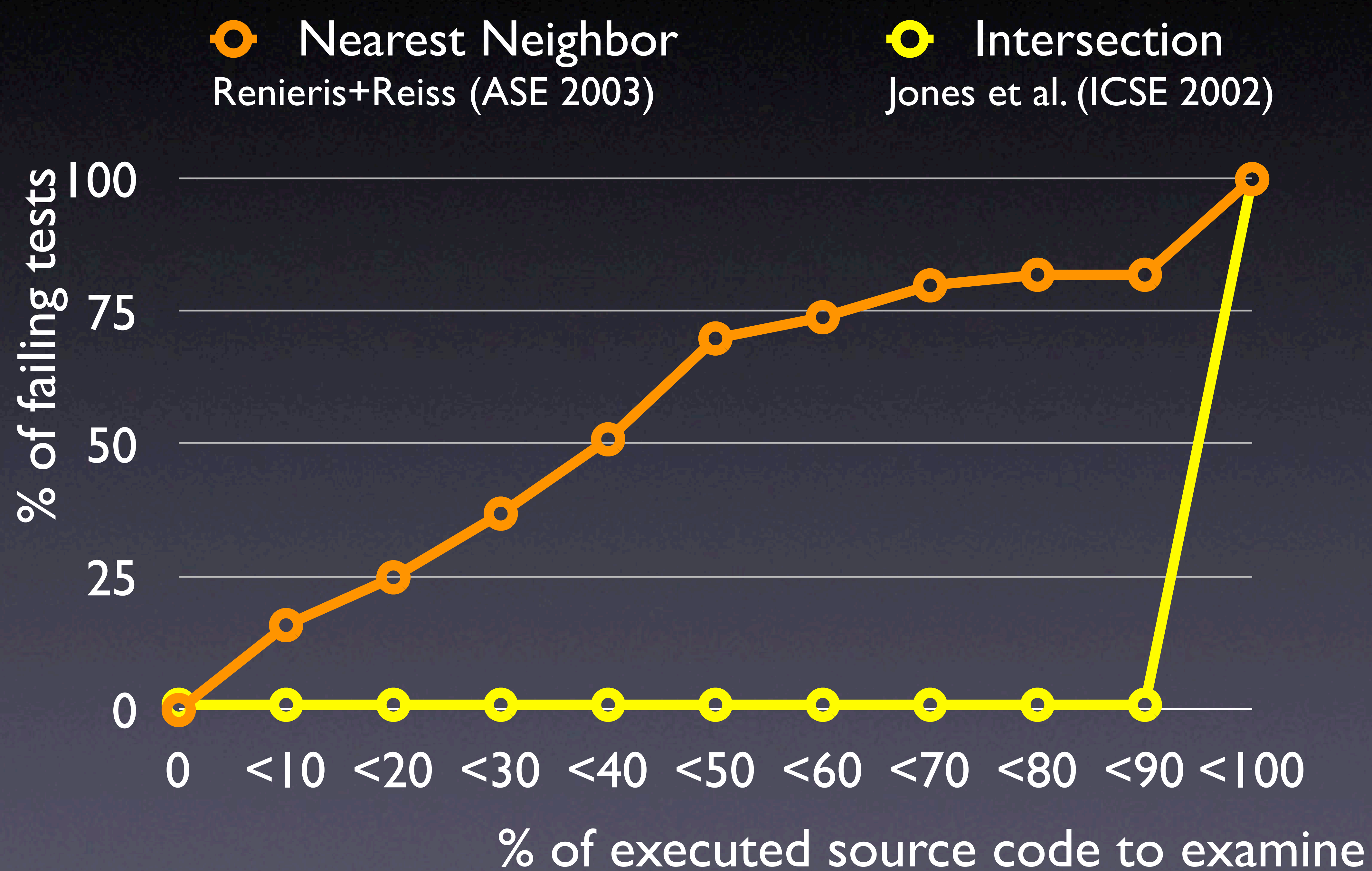
Source: Renieris and Reiss, ASE 2003

Nearest Neighbor

Nearest Neighbor

Compare with the single run
that has the most similar coverage

Locating Defects



Results obtained from Siemens test suite; can not be generalized

Sequences

Sequences of locations can correlate with failures:

open() read() close()	✔
open() close() read()	✘
close() open() read()	✘

…but all locations are executed in both runs!

The AspectJ Compiler

```
$ ajc Test3.aj
$ java test.Test3
test.Test3@b8df17.x Unexpected Signal : 11
occurred at PC=0xFA415A00
Function name=(N/A) Library=(N/A) ...
Please report this error at http://java.sun.com/
...
$
```

Coverage Differences

- Compare the failing run with passing runs
- `BcelShadow.getThisJoinPointVar()` is invoked in the failing run only
- Unfortunately, this method is correct

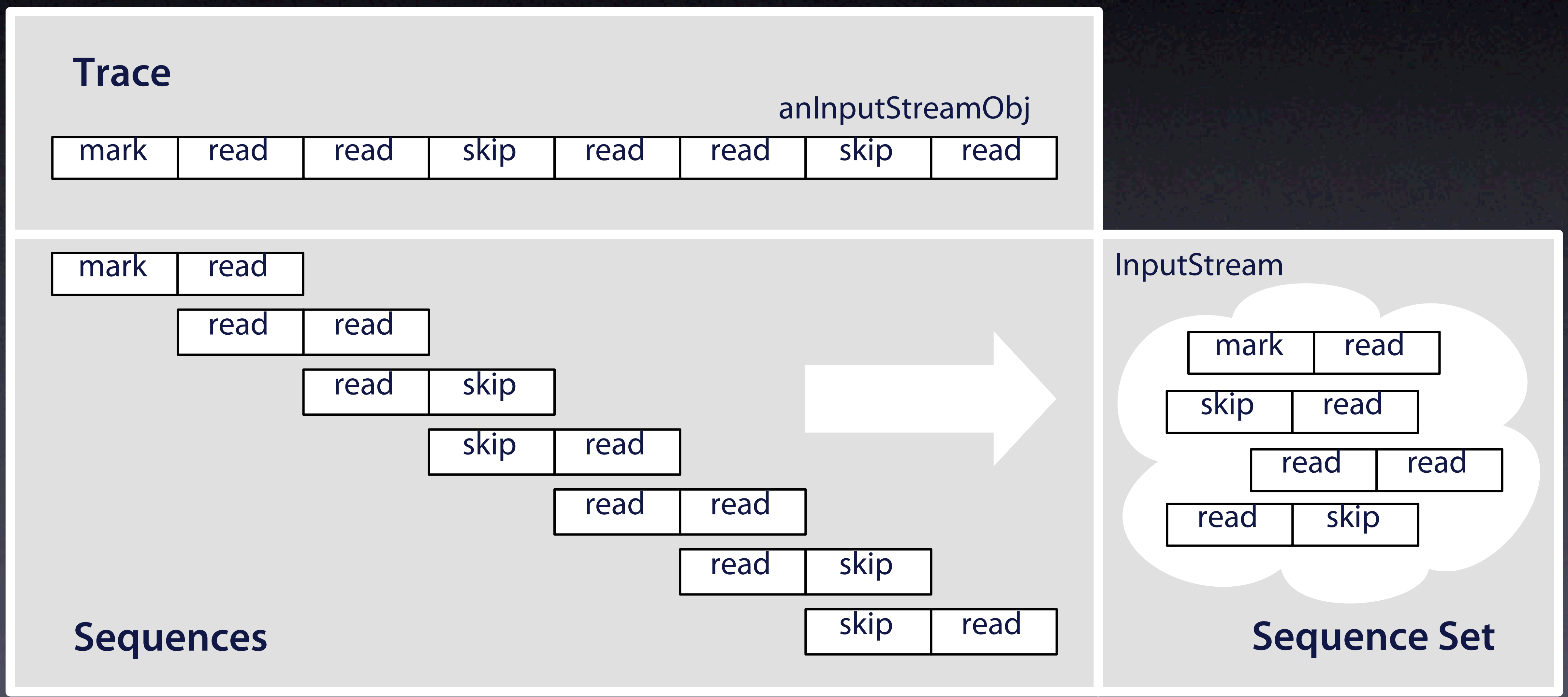
Sequence Differences

This *sequence* occurs only in the failing run:

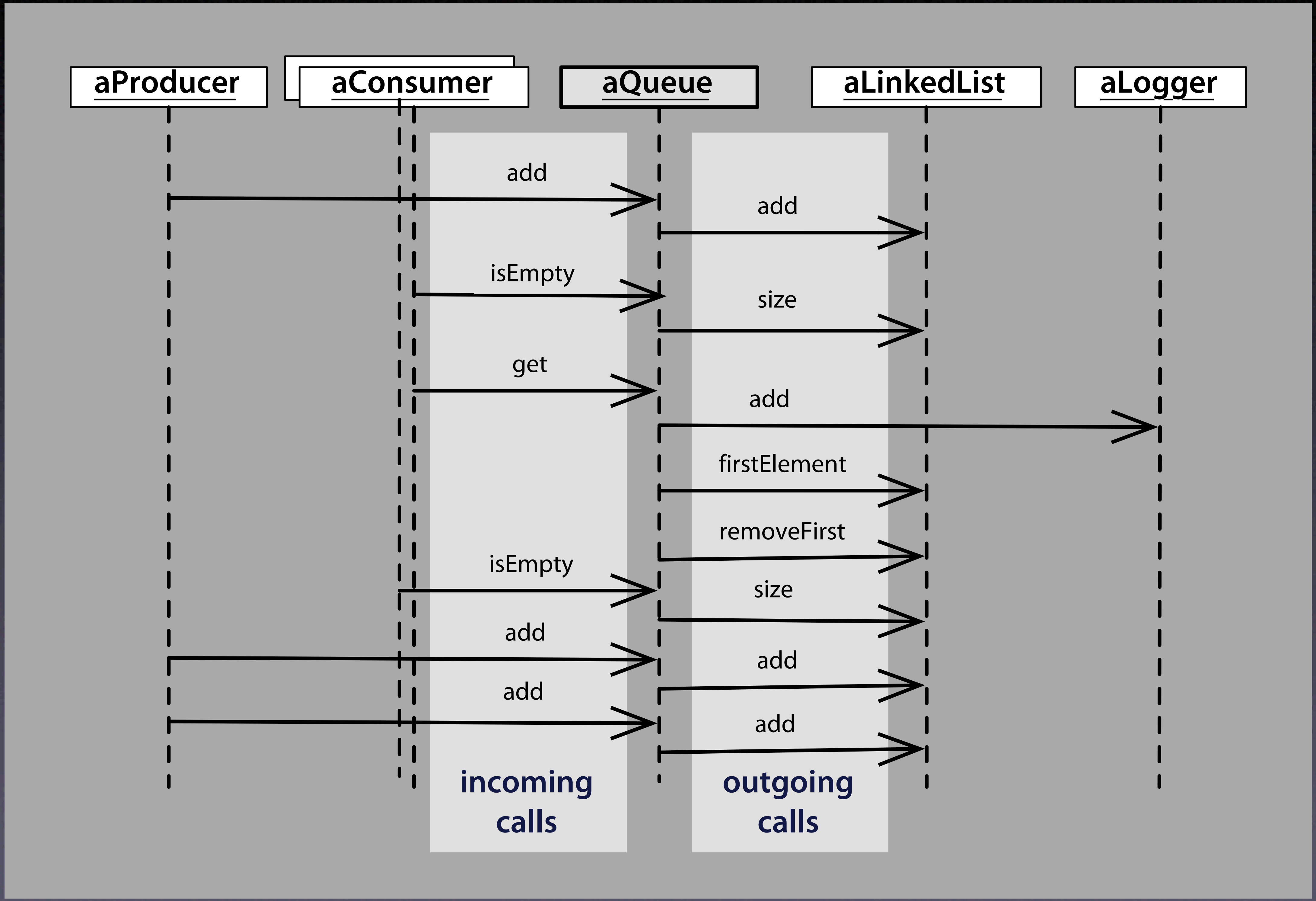
```
⟨ ThisJoinPointVisitor.isRef(),
  ThisJoinPointVisitor.canTreatAsStatic(),
  MethodDeclaration.traverse(),
  ThisJoinPointVisitor.isRef(),
  ThisJoinPointVisitor.isRef() ⟩
```

Defect location

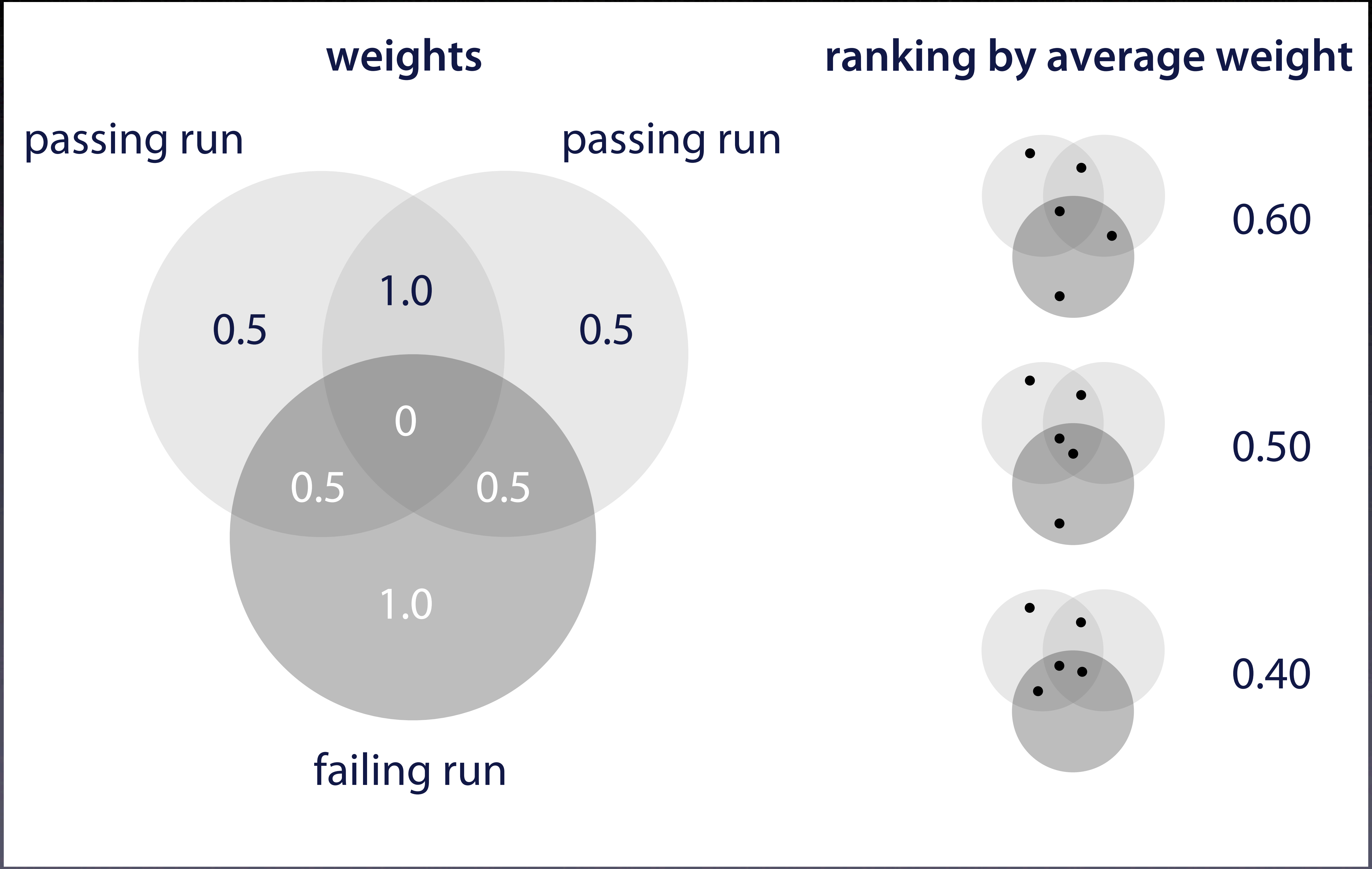
Collecting Sequences



Ingoing vs. Outgoing



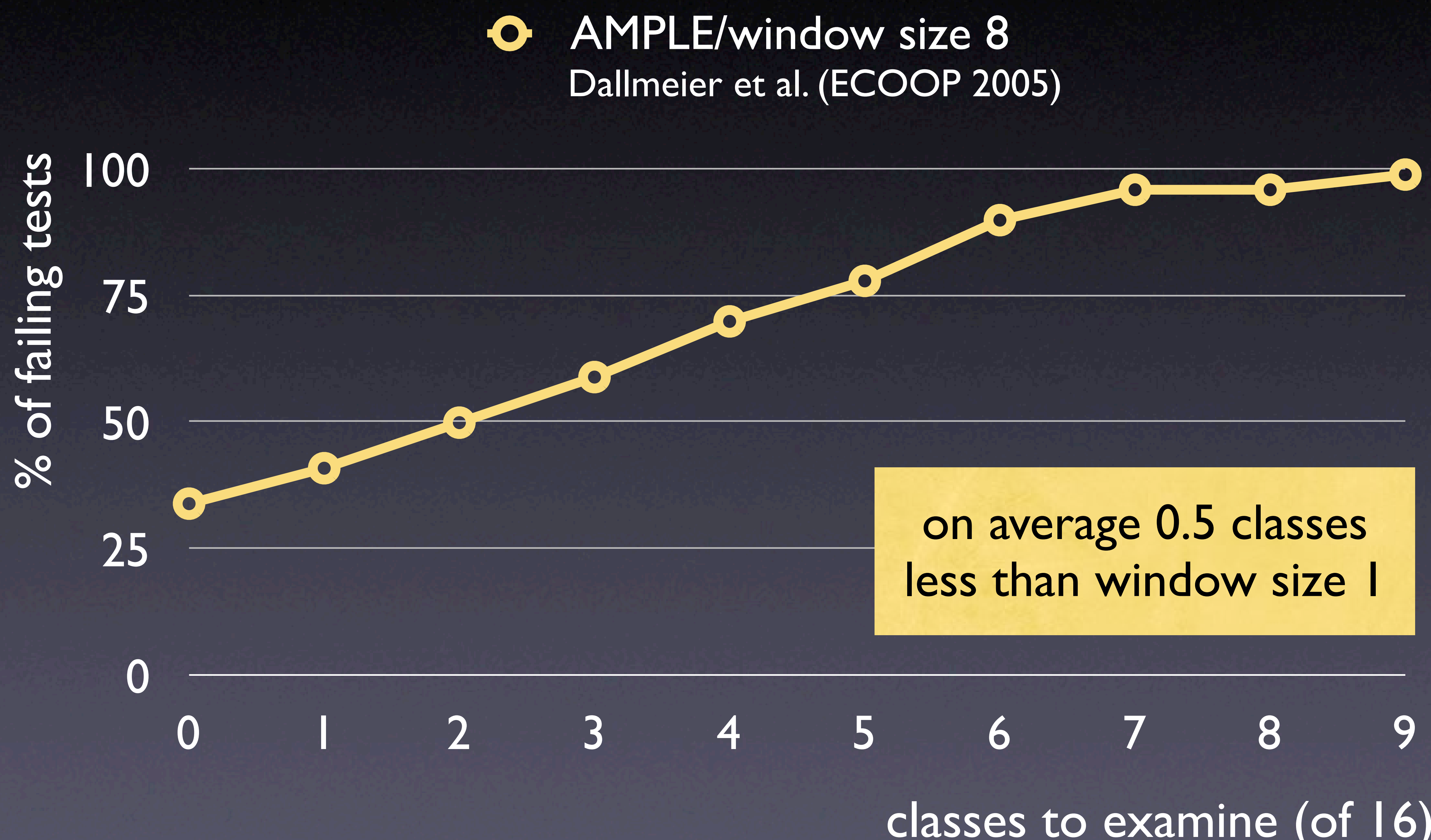
Anomalies


weights
ranking by average weight
passing run
passing run
0.5
1.0
0.5
0
0.5
0.5
1.0
failing run
0.60
0.50
0.40

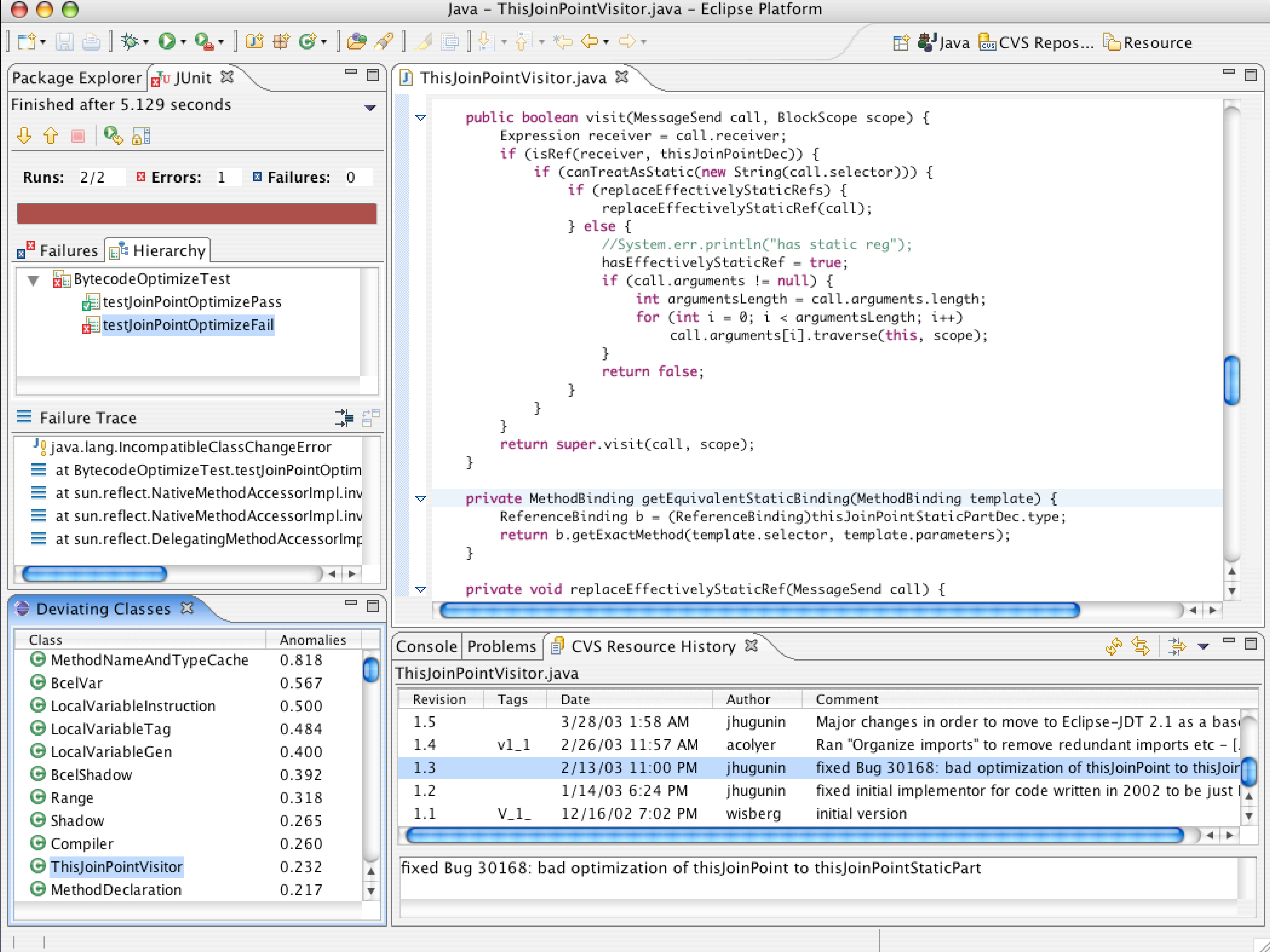
NanoXML

- Simple XML parser written in Java
- 5 revisions, each with 16–23 classes
- 33 errors discovered or seeded

Locating Defects



Results obtained from NanoXML; can not be generalized


Java – ThisJoinPointVisitor.java – Eclipse Platform
Java
CVS Repos...
Resource
Package Explorer
JUnit
Finished after 5.129 seconds
Runs: 2/2
Errors: 1
Failures: 0
Failures
Hierarchy
BytecodeOptimizeTest
testJoinPointOptimizePass
testJoinPointOptimizeFail
Failure Trace
java.lang.IncompatibleClassChangeError
at BytecodeOptimizeTest.testJoinPointOptim
at sun.reflect.NativeMethodAccessorImpl.inv
at sun.reflect.NativeMethodAccessorImpl.inv
at sun.reflect.DelegatingMethodAccessorImp
Deviating Classes
Class
Anomalies
MethodNameAndTypeCache 0.818
BcelVar 0.567
LocalVariableInstruction 0.500
LocalVariableTag 0.484
LocalVariableGen 0.400
BcelShadow 0.392
Range 0.318
Shadow 0.265
Compiler 0.260
ThisJoinPointVisitor 0.232
MethodDeclaration 0.217
ThisJoinPointVisitor.java
public boolean visit(MessageSend call, BlockScope scope) {
Expression receiver = call.receiver;
if (isRef(receiver, thisJoinPointDec)) {
if (canTreatAsStatic(new String(call.selector))) {
if (replaceEffectivelyStaticRefs) {
replaceEffectivelyStaticRef(call);
} else {
//System.err.println("has static reg");
hasEffectivelyStaticRef = true;
if (call.arguments != null) {
int argumentsLength = call.arguments.length;
for (int i = 0; i < argumentsLength; i++)
call.arguments[i].traverse(this, scope);
}
return false;
}
}
}
return super.visit(call, scope);
}
private MethodBinding getEquivalentStaticBinding(MethodBinding template) {
ReferenceBinding b = (ReferenceBinding)thisJoinPointStaticPartDec.type;
return b.getExactMethod(template.selector, template.parameters);
}
private void replaceEffectivelyStaticRef(MessageSend call) {
Console
Problems
CVS Resource History
ThisJoinPointVisitor.java
Revision
Tags
Date
Author
Comment
1.5 3/28/03 1:58 AM jhugunin Major changes in order to move to Eclipse-JDT 2.1 as a bas
1.4 v1_1 2/26/03 11:57 AM acolyer Ran "Organize imports" to remove redundant imports etc – [
1.3 2/13/03 11:00 PM jhugunin fixed Bug 30168: bad optimization of thisJoinPoint to thisJoir
1.2 1/14/03 6:24 PM jhugunin fixed initial implementor for code written in 2002 to be just
1.1 V_1_ 12/16/02 7:02 PM wisberg initial version
fixed Bug 30168: bad optimization of thisJoinPoint to thisJoinPointStaticPart

Concepts

- Comparing coverage (or other features) shows anomalies correlated with failure
- Nearest neighbor or sequences locate errors more precisely than just coverage
- Low overhead + simple to realize