Fork me on Github

Unit Test Virtualization with VMVM

Jonathan Bell and Gail Kaiser
Columbia University

ICSE 2014

@_jon_bell_

June 5, 2014

Good news: We have tests!

No judgement on whether they are complete or not,
but we sure have a lot of them

	Number of tests
Apache Tomcat	1,734
Closure Compiler	7,949
Commons I/O	1,022

Bad news: We have to run a lot of tests!

Bad news: We have to run a lot of tests!

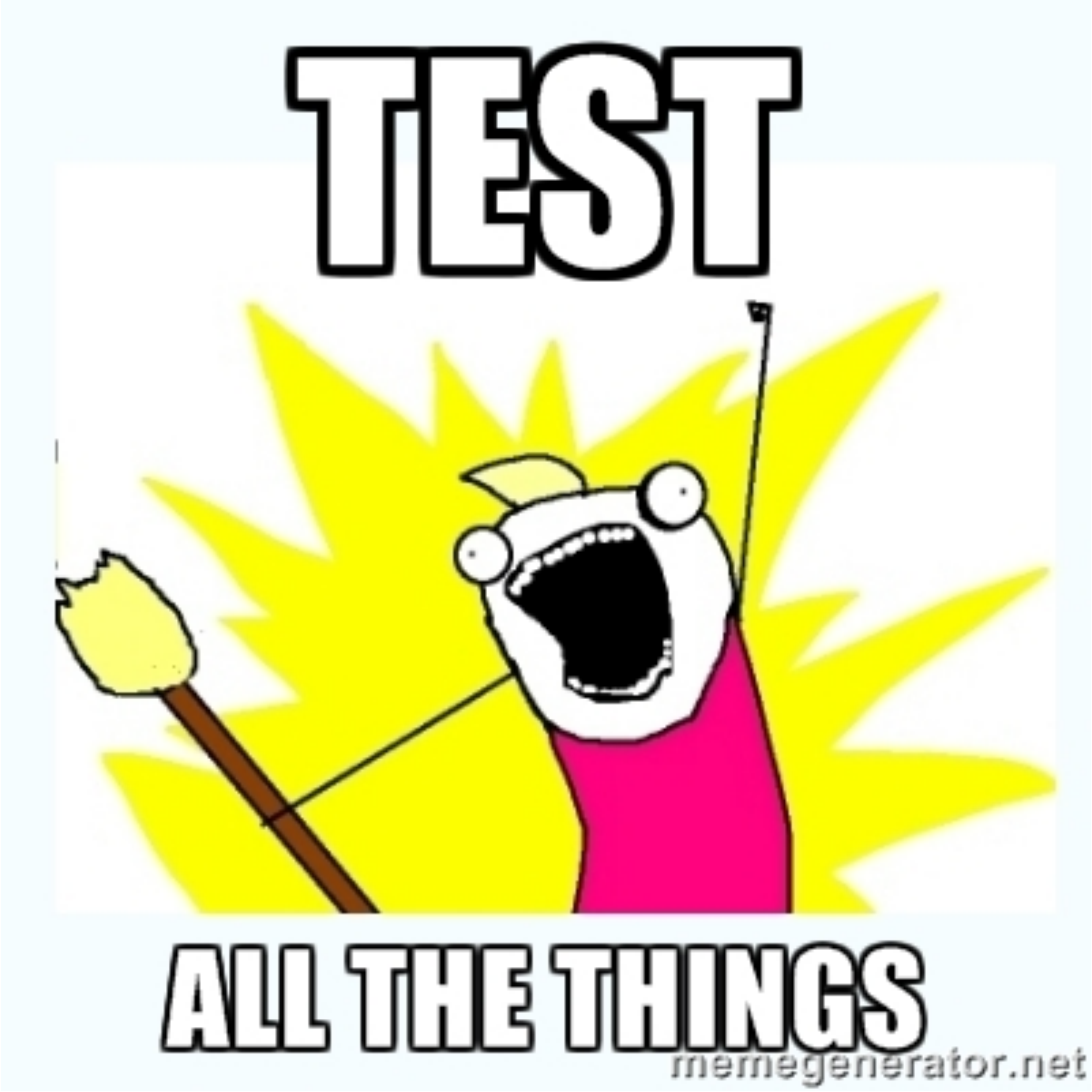
- Much work has focused on improving the situation:
- Test Suite Prioritization
  - E.g. Wong [ISSRE ’97], Rothermel [ICSM ’99]; Elbaum [ICSE ’01]; Srivastava [ISSTA ’02] and more

Bad news: We have to run a lot of tests!

- Much work has focused on improving the situation:
- Test Suite Prioritization
  - E.g. Wong [ISSRE ’97], Rothermel [ICSM ’99]; Elbaum [ICSE ’01]; Srivastava [ISSTA ’02] and more
- Test Suite Minimization
  - E.g. Harrold [TOSEM ’93]; Wong [ICSE ’95]; Chen [IST ’98]; Jones [TOSEM ’03]; Tallam [PASTE ’05]; Jeffrey [TSE ’07]; Orso [ICSE ’09] Hao [ICSE ’12] and more

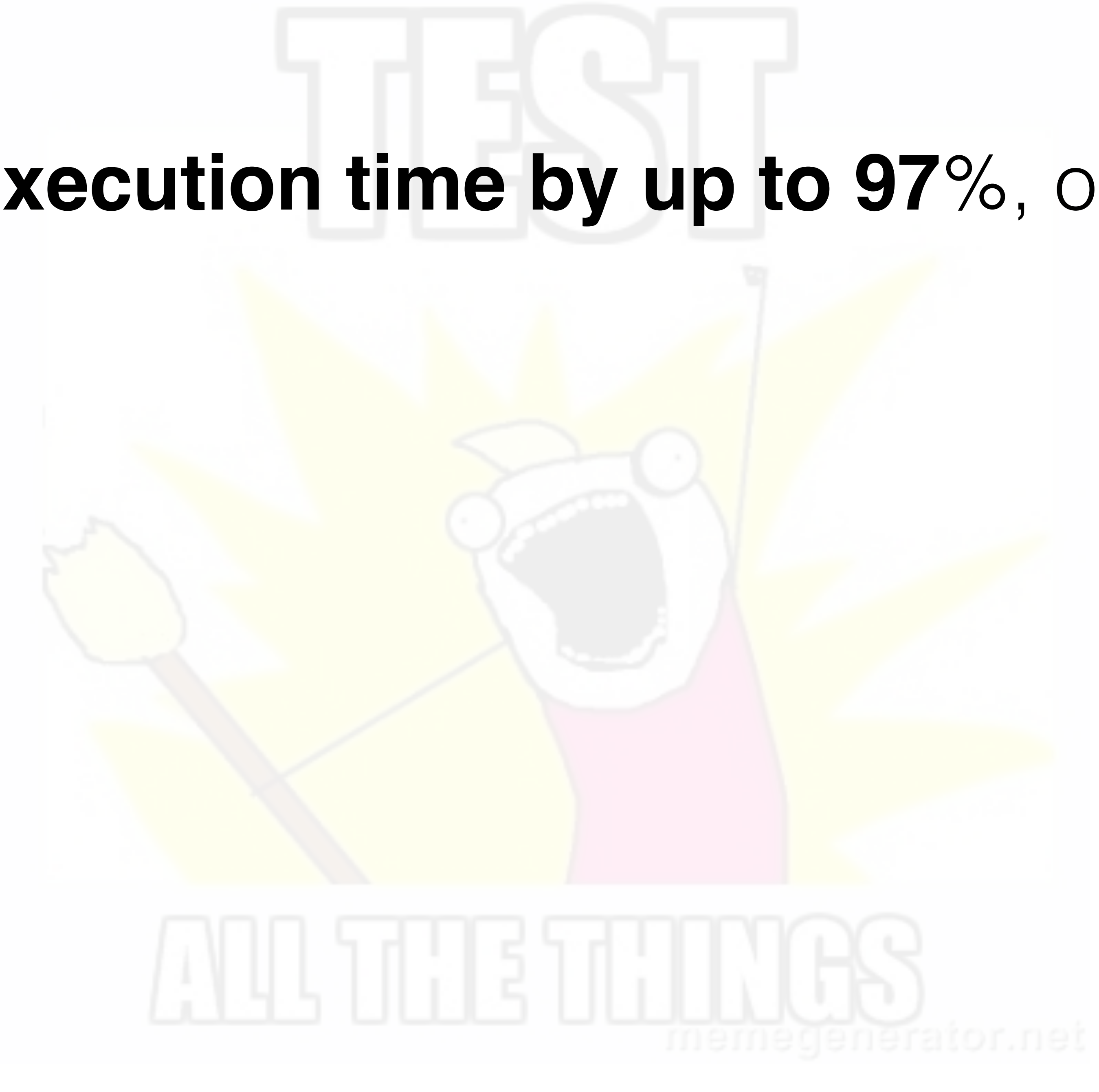
Testing still takes too long.

Our Approach: Unit Test Virtualization

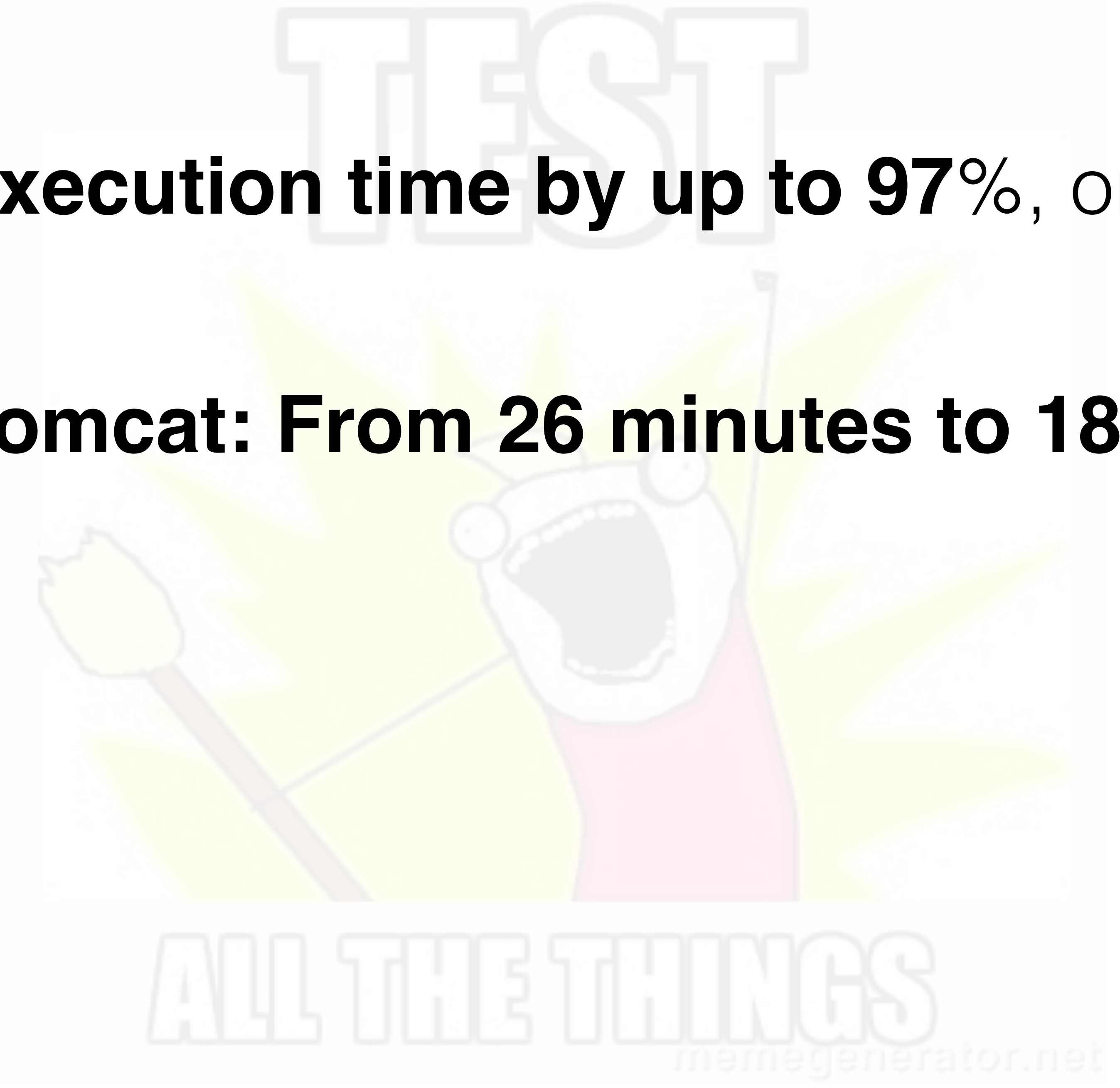


Our Approach:
Unit Test Virtualization

Reduces test execution time by up to 97%, on average 62%



Our Approach:
Unit Test Virtualization



Reduces test execution time by up to 97%, on average 62%

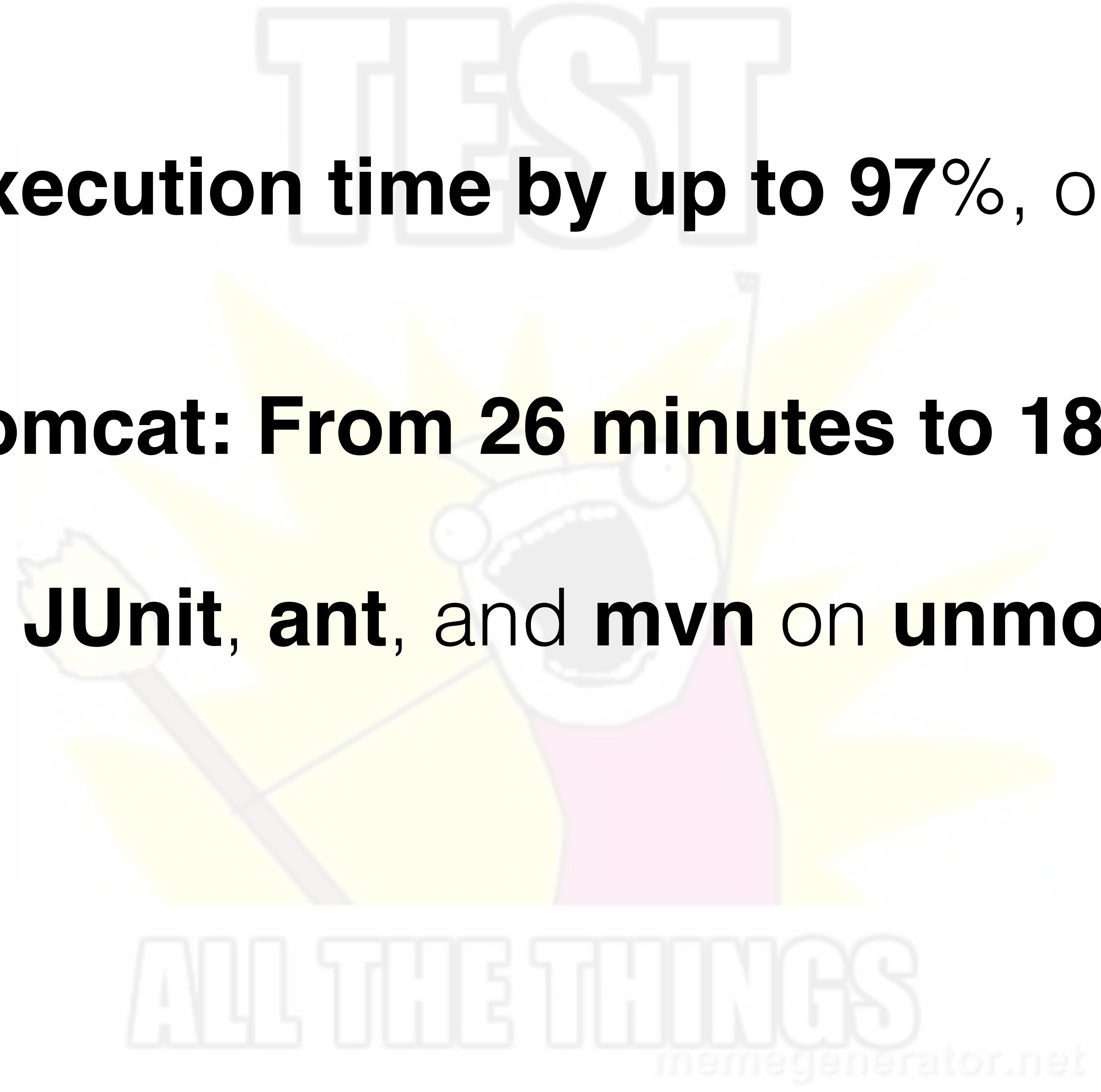
Apache Tomcat: From 26 minutes to 18 minutes

Our Approach:
Unit Test Virtualization

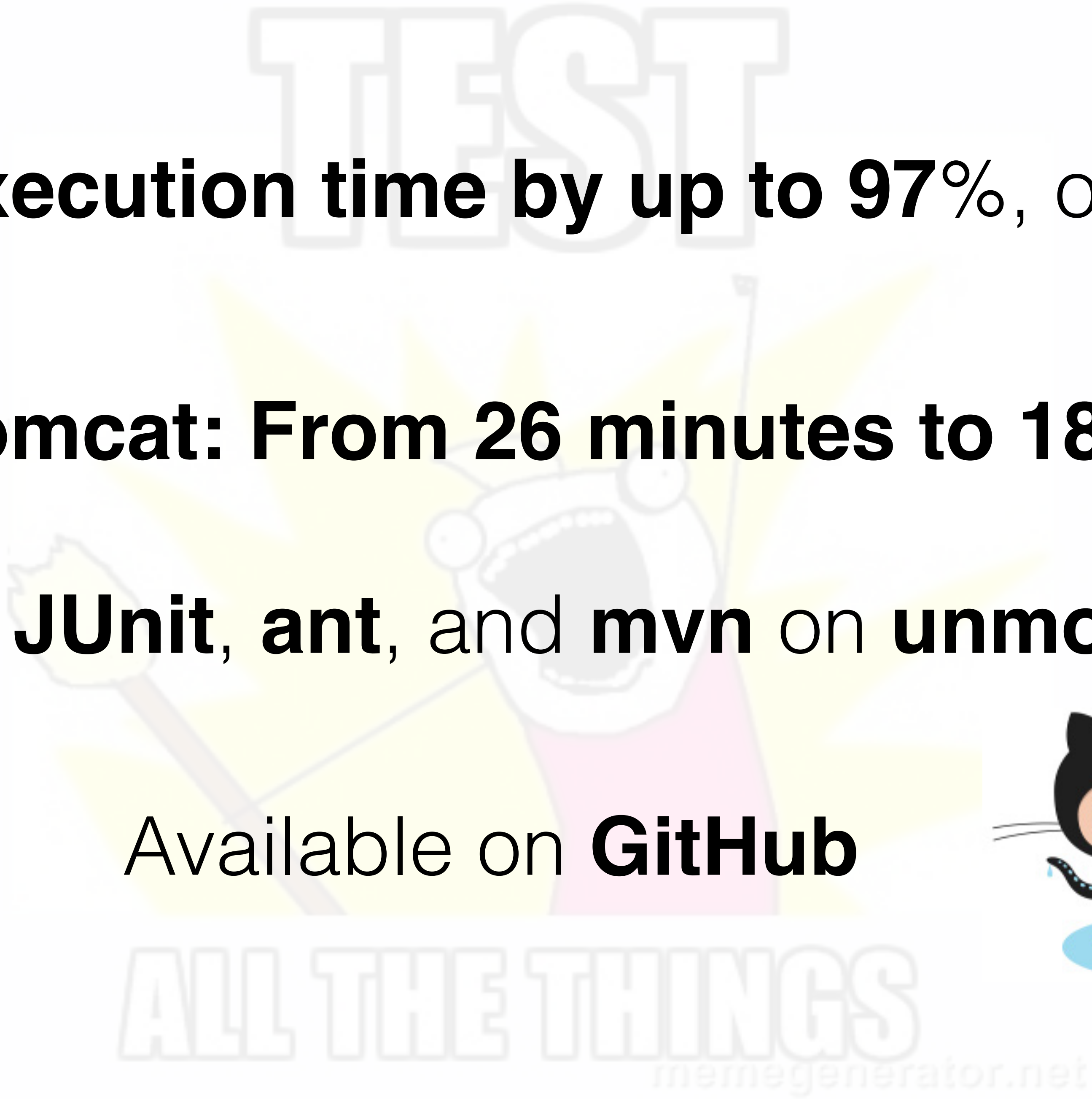
Reduces test execution time by up to 97%, on average 62%

Apache Tomcat: From 26 minutes to 18 minutes

Integrates with **JUnit**, **ant**, and **mvn** on **unmodified JVMs.**



Our Approach:
Unit Test Virtualization



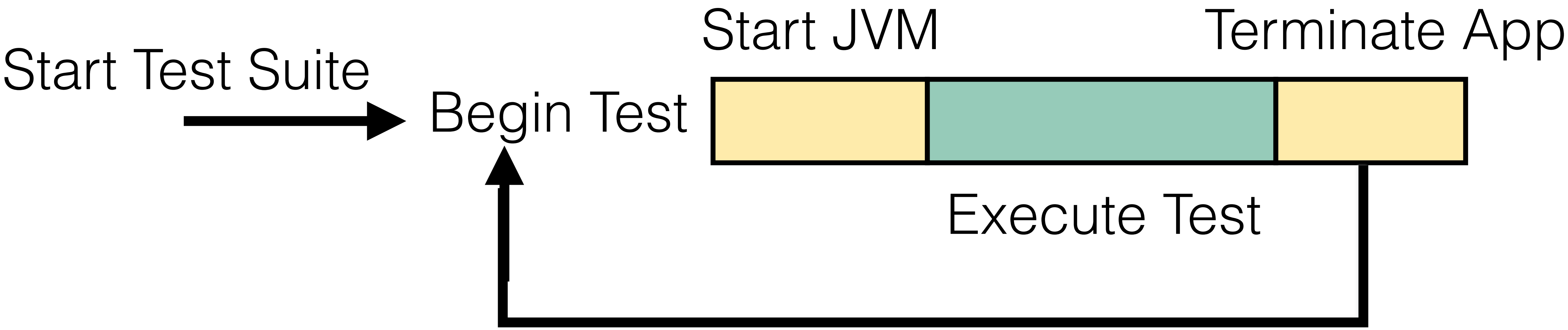
Reduces test execution time by up to 97%, on average 62%

Apache Tomcat: From 26 minutes to 18 minutes

Integrates with **JUnit**, **ant**, and **mvn** on **unmodified JVMs.**

Available on **GitHub**

JUnit Test Execution



Start Test Suite

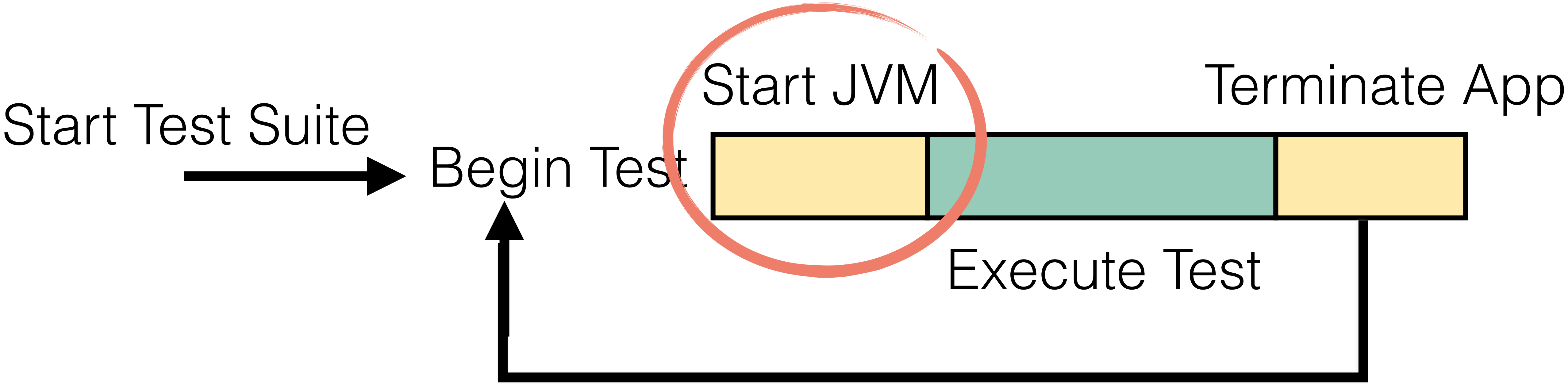
Begin Test

Start JVM

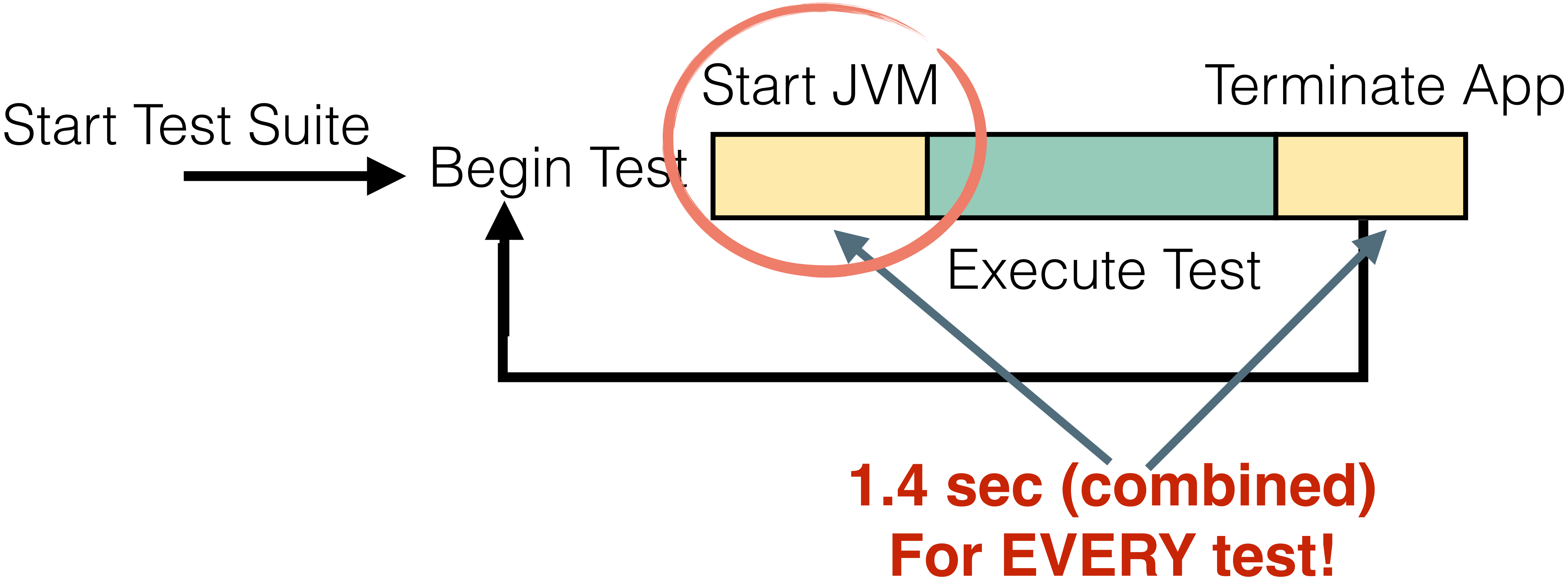
Execute Test

Terminate App

JUnit Test Execution

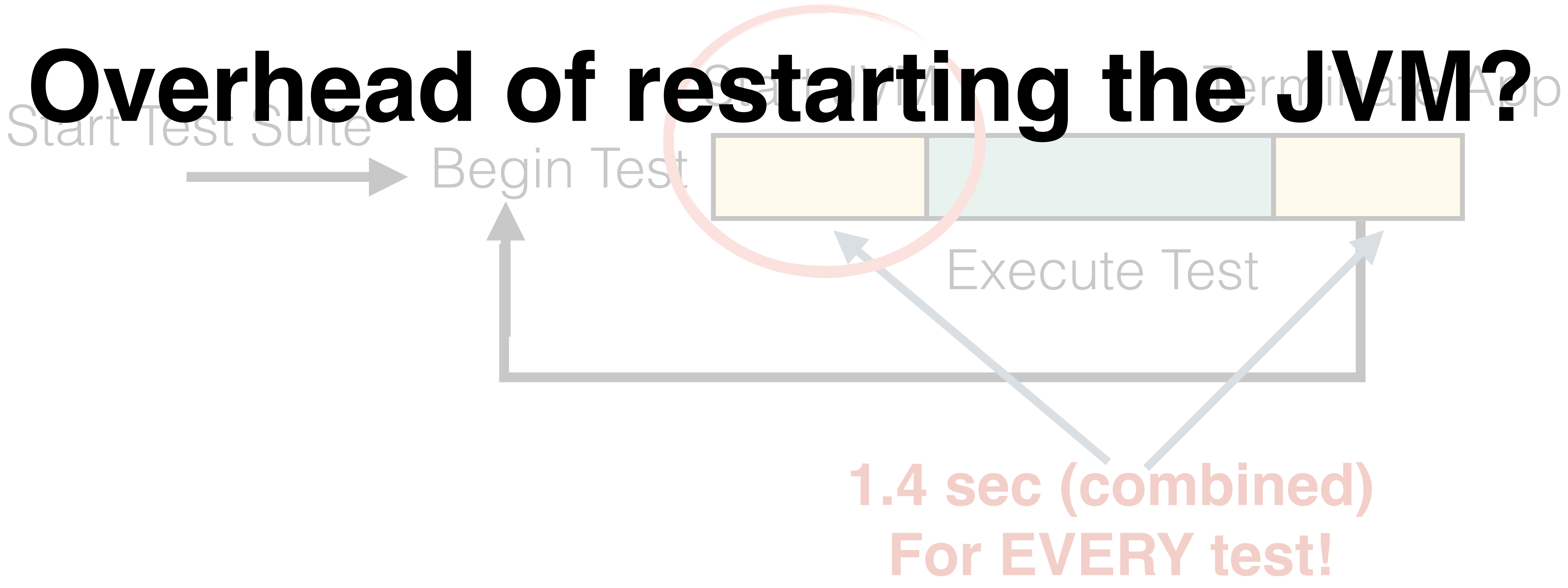


JUnit Test Execution



**1.4 sec (combined)
For EVERY test!**

JUnit Test Execution

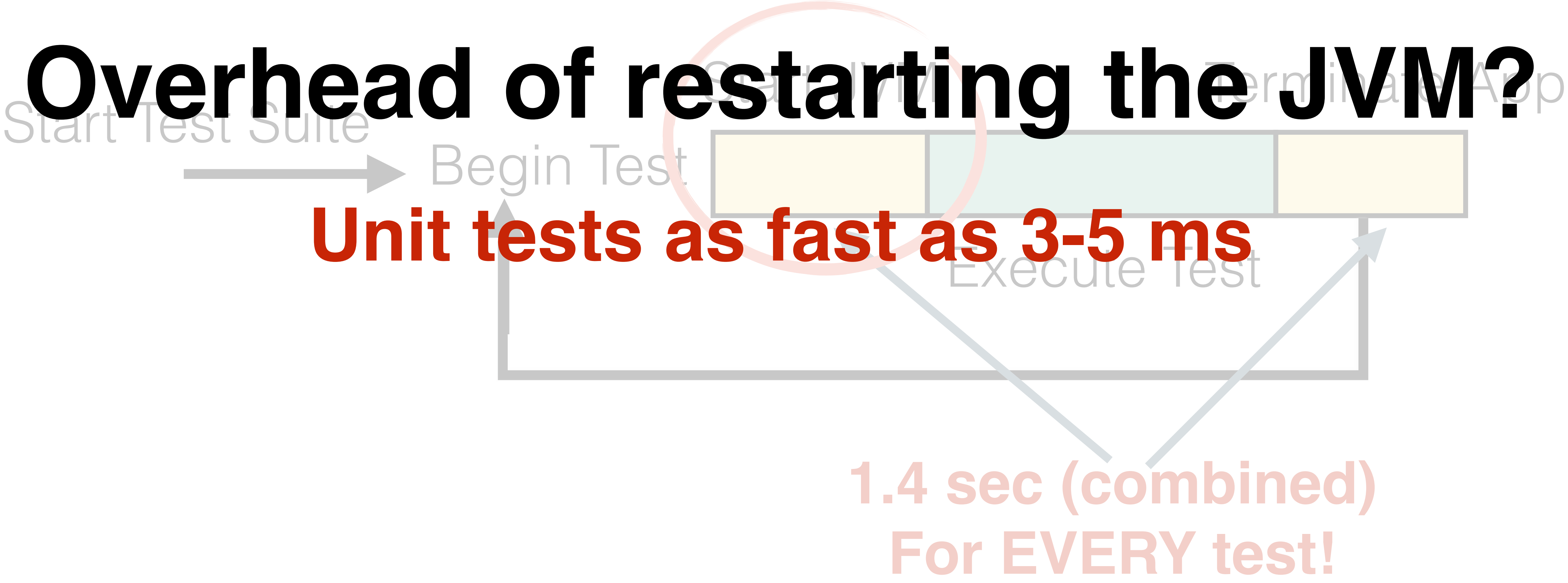


Overhead of restarting the JVM?

JUnit Test Execution

Overhead of restarting the JVM?

Unit tests as fast as 3-5 ms



Start Test Suite

Begin Test

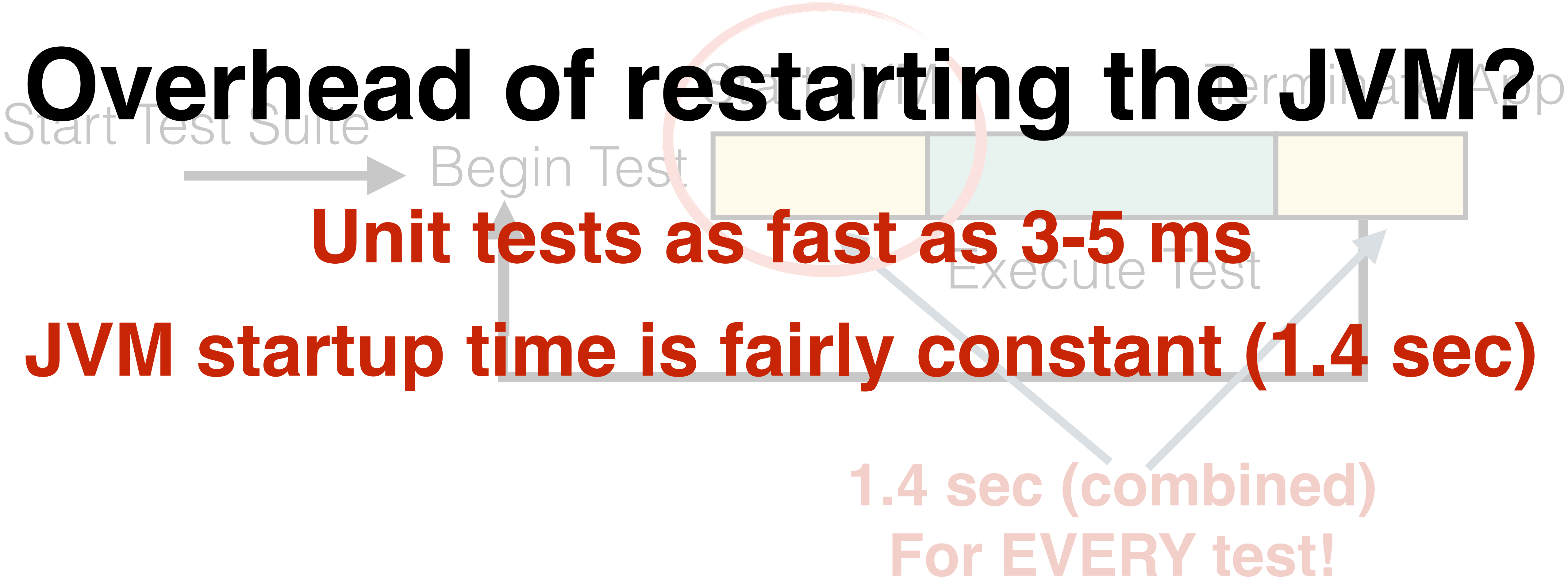
Start JVM

Terminate App

Execute Test

1.4 sec (combined)
For EVERY test!

JUnit Test Execution

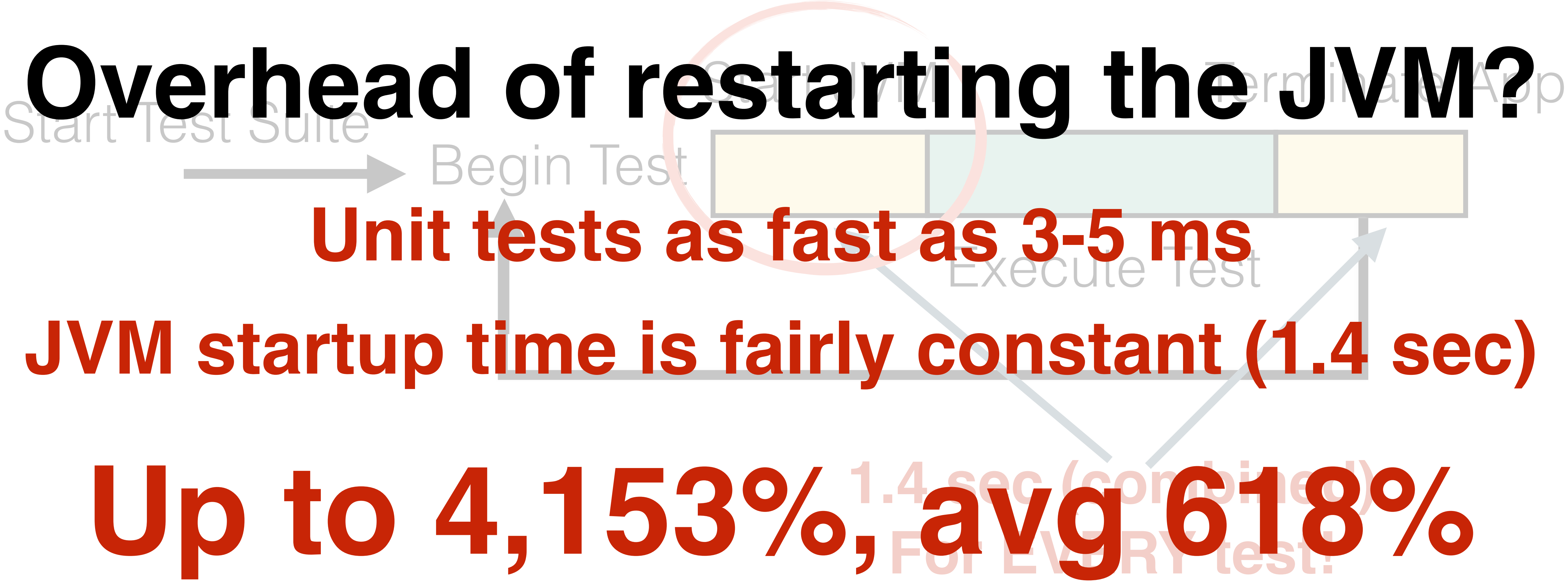


Overhead of restarting the JVM?

Unit tests as fast as 3-5 ms

JVM startup time is fairly constant (1.4 sec)

JUnit Test Execution



Overhead of restarting the JVM?

Unit tests as fast as 3-5 ms

JVM startup time is fairly constant (1.4 sec)

Up to 4,153%, avg 618%

***From our study of 20 popular FOSS apps**

Do applications really use a new JVM for each test?

- Checked out the 1,000 largest Java projects from Ohloh
- 81% of projects with more than 1,000 tests do it
- 71% of projects with more than 1 million LOC do it
- Overall: 41% of all of the projects do

Test Independence

- We typically assume that tests are *order-independent*

Test Independence

- We typically assume that tests are *order-independent*
- Might rely on developers to completely reset the system under test between tests
  - Who tests the tests?

Test Independence

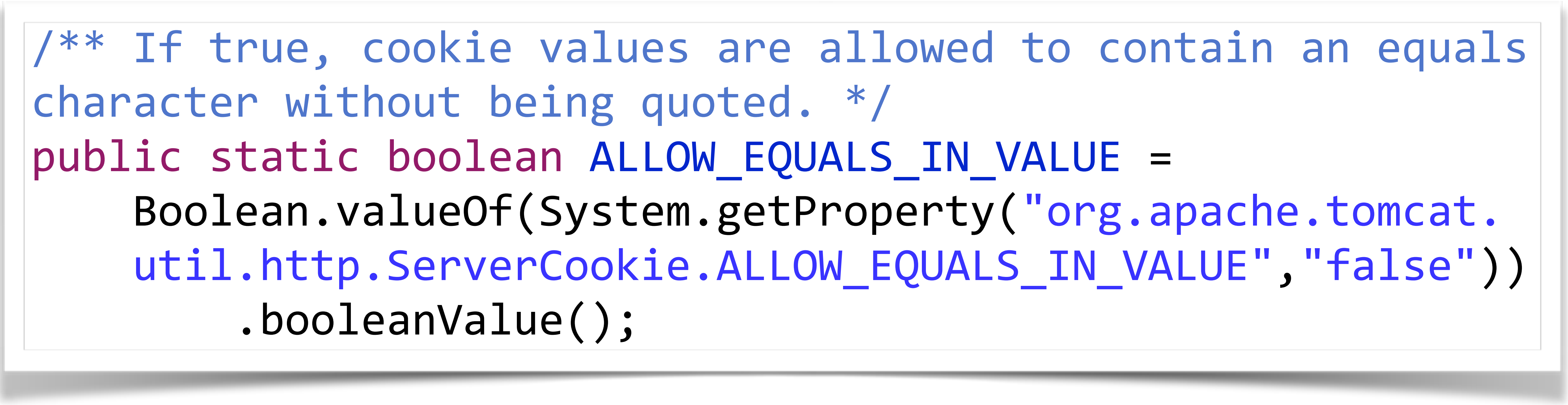
- We typically assume that tests are *order-independent*

- Might rely on developers to completely reset the system under test between tests

    - Who tests the tests?

- Dangerous: If wrong, can have false positives or false negatives (Muşlu [FSE ’11], Zhang [ISSTA ’14])

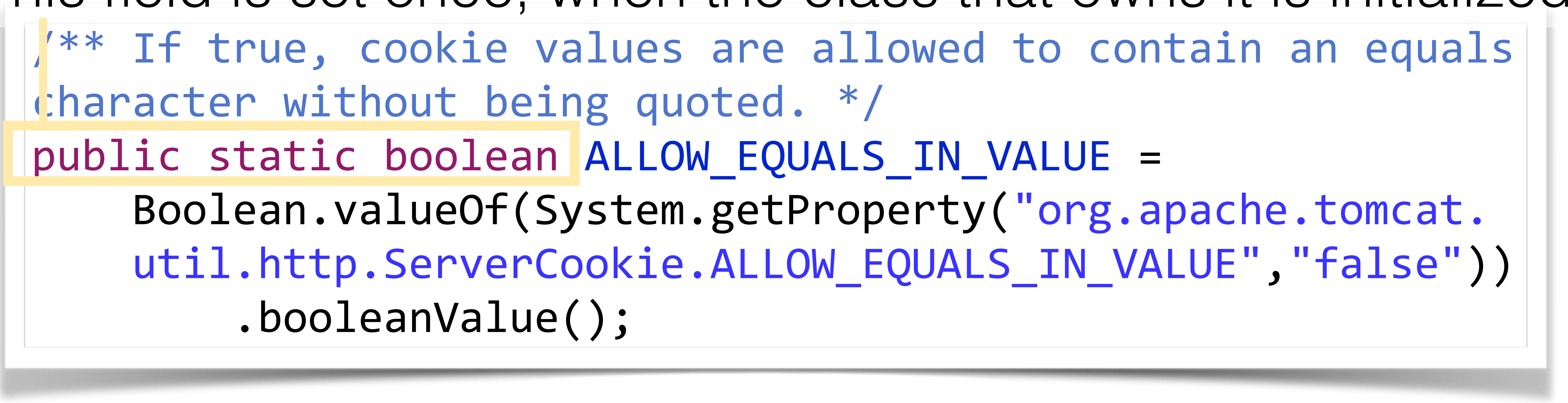
Test Independence



```
/** If true, cookie values are allowed to contain an equals
character without being quoted. */
public static boolean ALLOW_EQUALS_IN_VALUE =
    Boolean.valueOf(System.getProperty("org.apache.tomcat.
    util.http.ServerCookie.ALLOW_EQUALS_IN_VALUE","false"))
        .booleanValue();
```

Test Independence

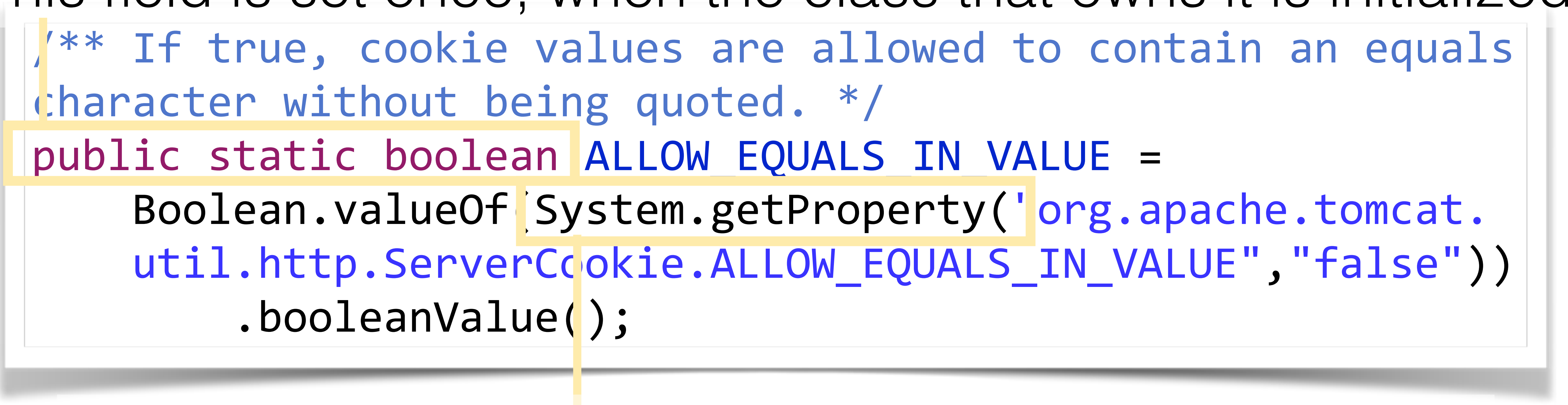
This field is set once, when the class that owns it is initialized



```
/** If true, cookie values are allowed to contain an equals
character without being quoted. */
public static boolean ALLOW_EQUALS_IN_VALUE =
    Boolean.valueOf(System.getProperty("org.apache.tomcat.
    util.http.ServerCookie.ALLOW_EQUALS_IN_VALUE","false"))
        .booleanValue();
```

Test Independence

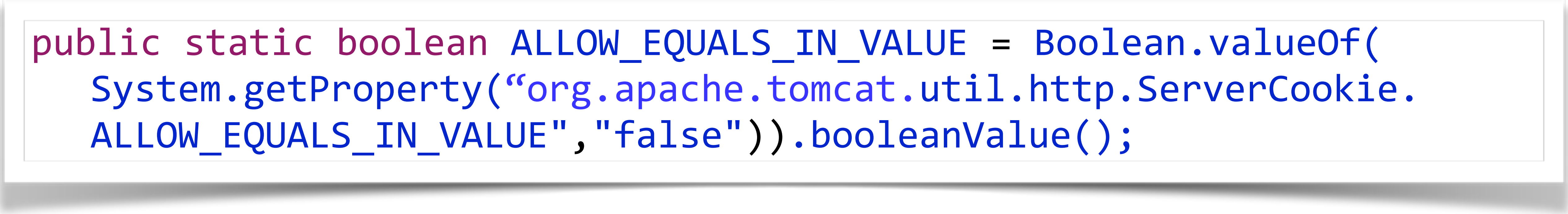
This field is set once, when the class that owns it is initialized



```
/** If true, cookie values are allowed to contain an equals
character without being quoted. */
public static boolean ALLOW_EQUALS_IN_VALUE =
    Boolean.valueOf(System.getProperty("org.apache.tomcat.
    util.http.ServerCookie.ALLOW_EQUALS_IN_VALUE","false"))
        .booleanValue();
```

This field's value is dependent on an external property

A Tale of Two Tests

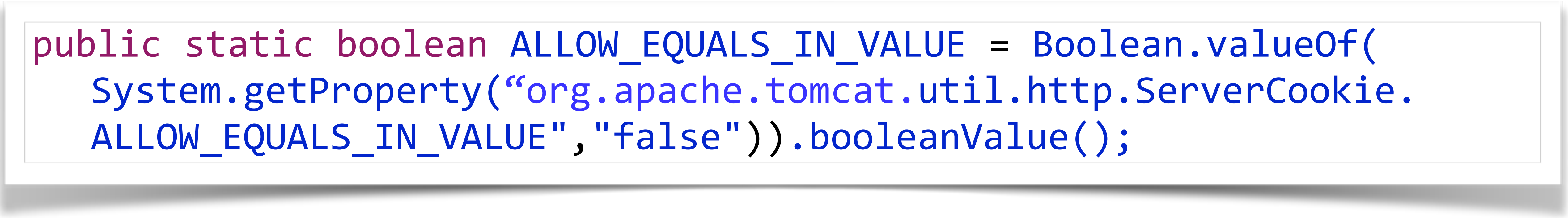


```
public static boolean ALLOW_EQUALS_IN_VALUE = Boolean.valueOf(
   System.getProperty(“org.apache.tomcat.util.http.ServerCookie.
   ALLOW_EQUALS_IN_VALUE","false")).booleanValue();
```

TestAllowEqualsInValue

TestDontAllowEqualsInValue

A Tale of Two Tests



```
public static boolean ALLOW_EQUALS_IN_VALUE = Boolean.valueOf(
   System.getProperty(“org.apache.tomcat.util.http.ServerCookie.
   ALLOW_EQUALS_IN_VALUE","false")).booleanValue();
```

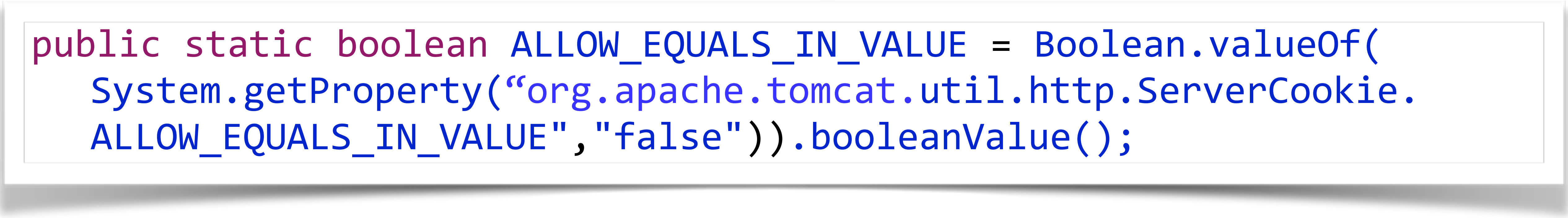
TestAllowEqualsInValue

TestDontAllowEqualsInValue

Sets environmental variable to `true`

Start Tomcat, run test

A Tale of Two Tests



```
public static boolean ALLOW_EQUALS_IN_VALUE = Boolean.valueOf(
   System.getProperty(“org.apache.tomcat.util.http.ServerCookie.
   ALLOW_EQUALS_IN_VALUE","false")).booleanValue();
```

TestAllowEqualsInValue

Sets environmental variable to `true`

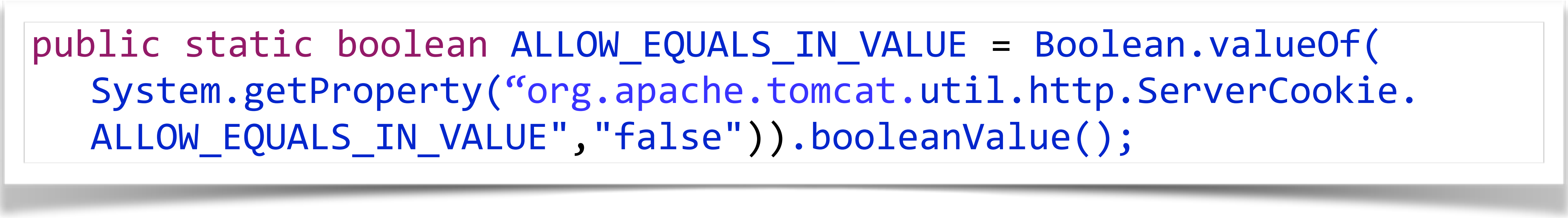
Start Tomcat, run test

TestDontAllowEqualsInValue

Sets environmental variable to `false`

Start Tomcat, run test

A Tale of Two Tests



```
public static boolean ALLOW_EQUALS_IN_VALUE = Boolean.valueOf(
   System.getProperty(“org.apache.tomcat.util.http.ServerCookie.
   ALLOW_EQUALS_IN_VALUE","false")).booleanValue();
```

TestAllowEqualsInValue

Sets environmental variable to `true`

Start Tomcat, run test

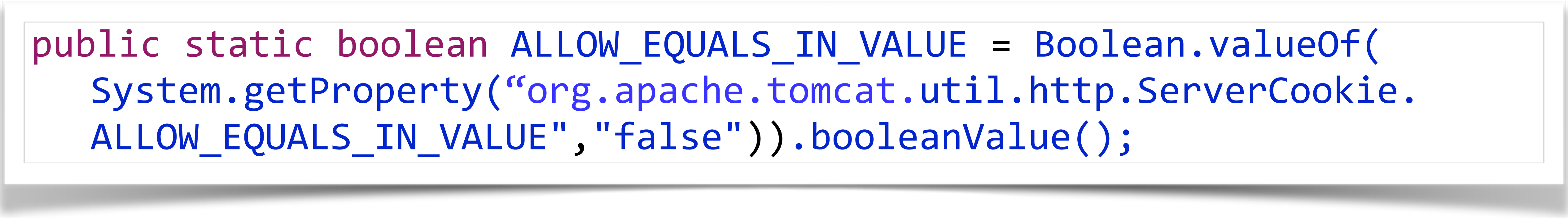
TestDontAllowEqualsInValue

Sets environmental variable to `false`

Start Tomcat, run test

But our static field is stuck!

A Tale of Two Tests



```
public static boolean ALLOW_EQUALS_IN_VALUE = Boolean.valueOf(
   System.getProperty(“org.apache.tomcat.util.http.ServerCookie.
   ALLOW_EQUALS_IN_VALUE","false")).booleanValue();
```

TestAllowEqualsInValue

Sets environmental variable to `true`

Start Tomcat, run test

TestDontAllowEqualsInValue

Sets environmental variable to `false`

Start Tomcat, run test

Our Approach

Unit Test Virtualization: Allow tests to leave side-effects.
But efficiently contain them.

How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

We think in terms of *object graphs*

How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

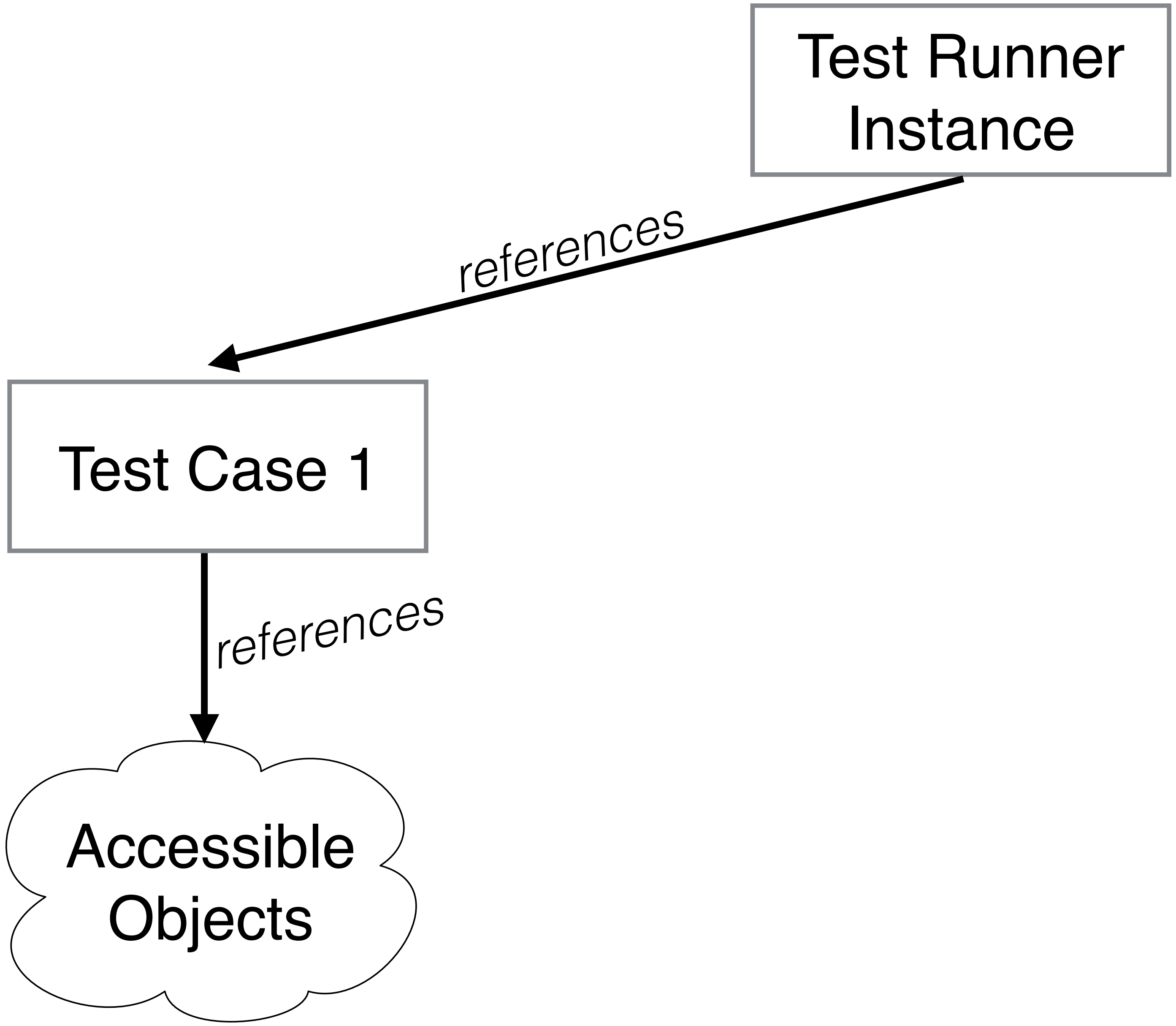
We think in terms of *object graphs*

Test Runner Instance

How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

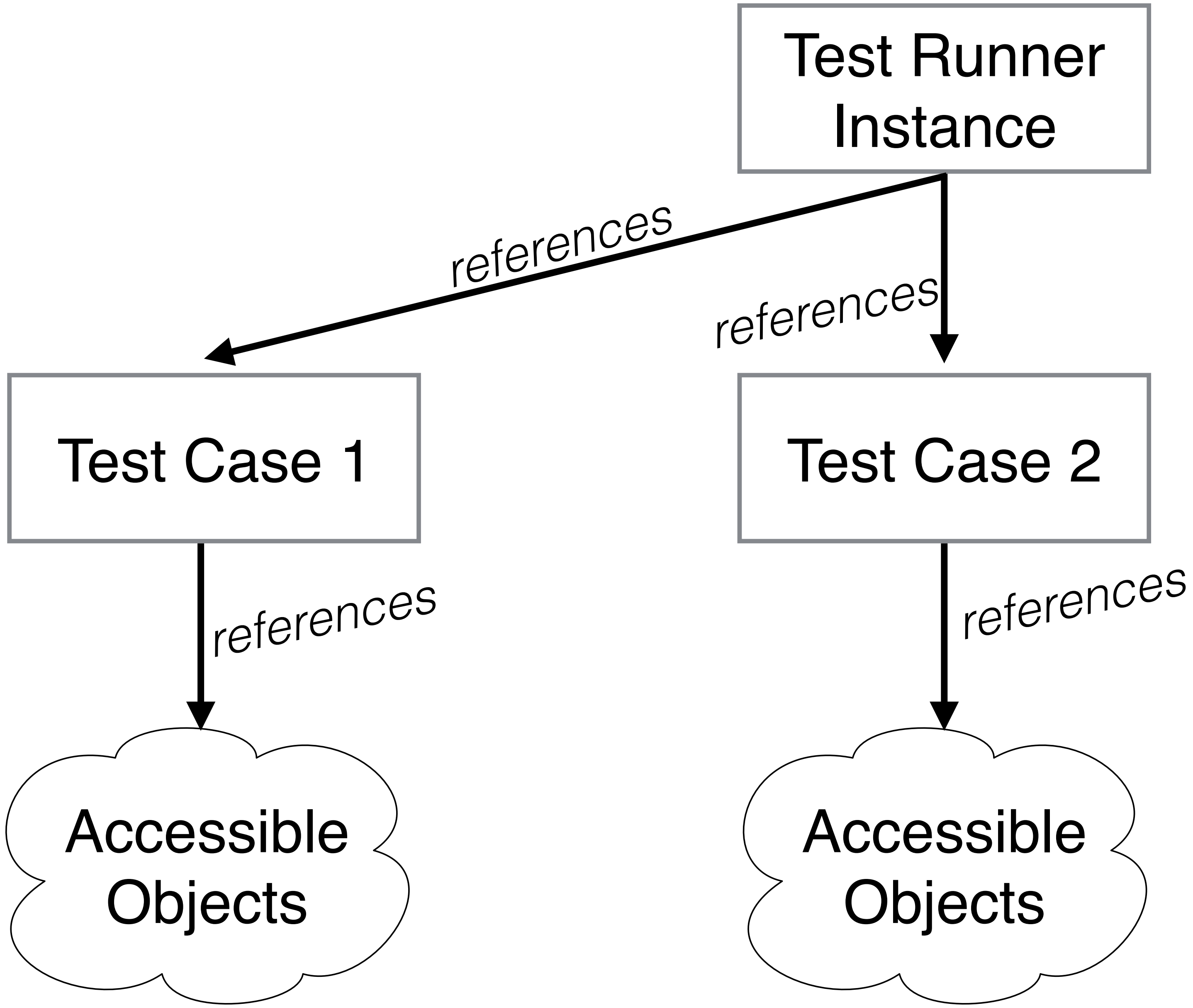
We think in terms of *object graphs*



How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

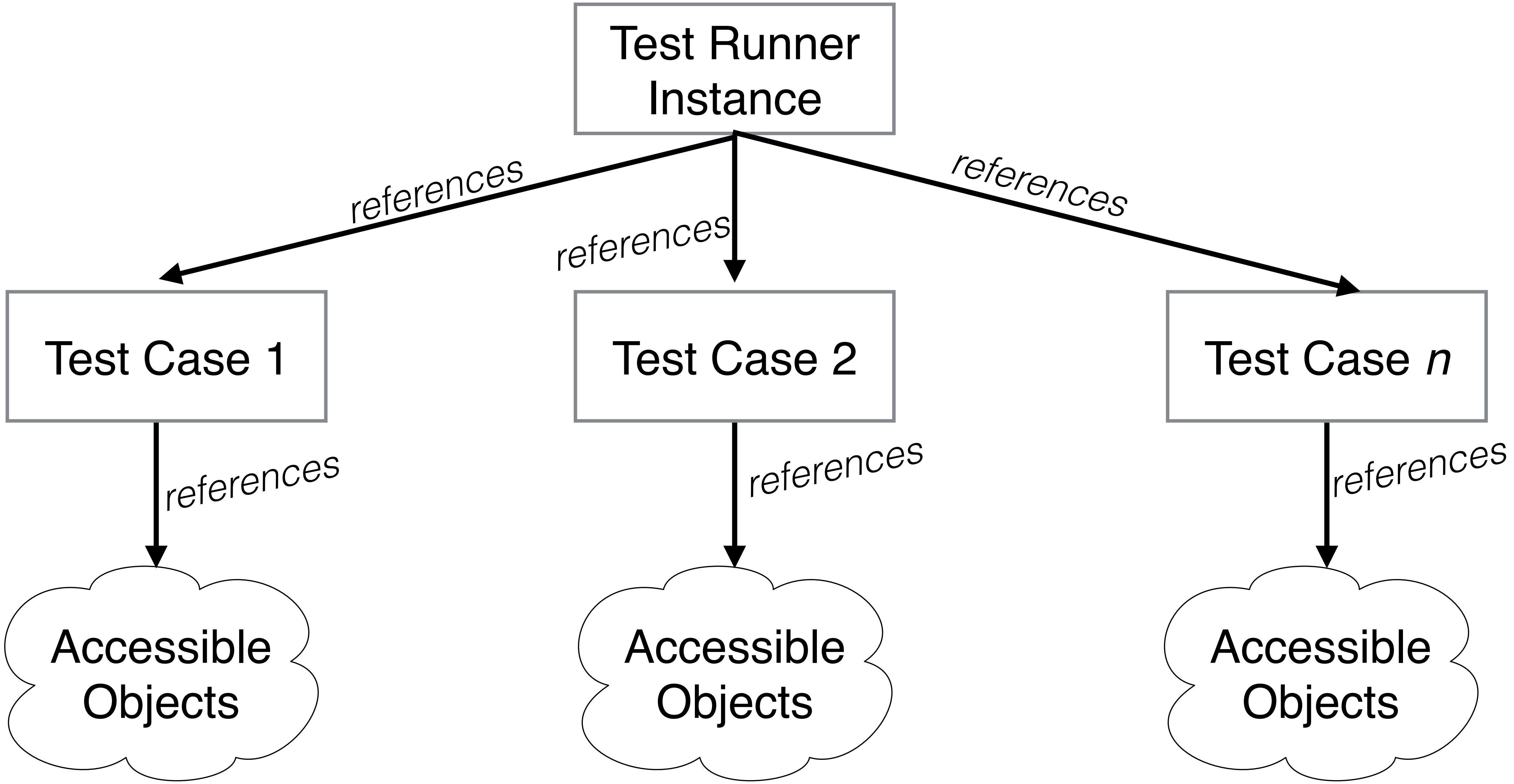
We think in terms of *object graphs*



How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

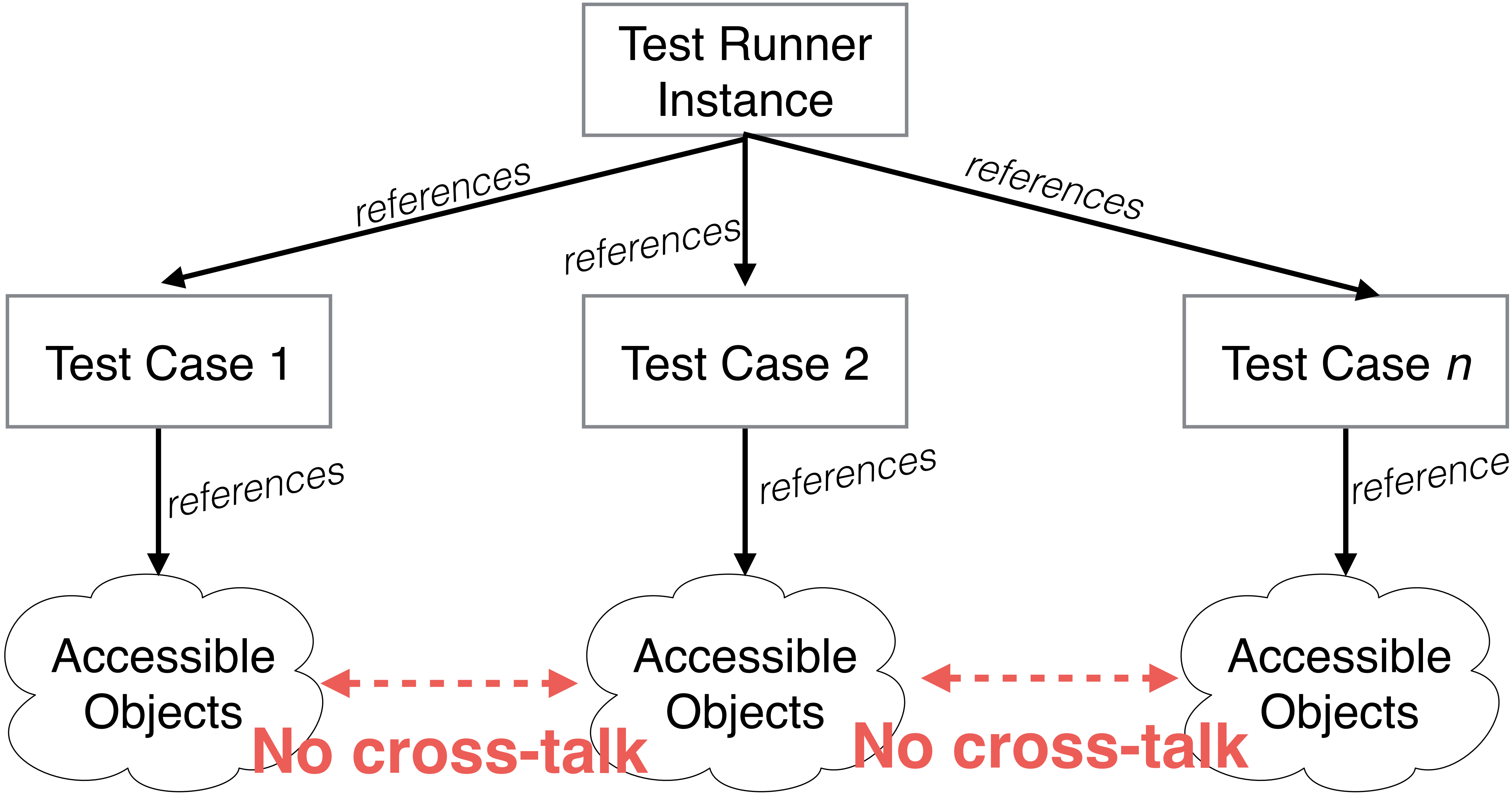
We think in terms of *object graphs*



How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

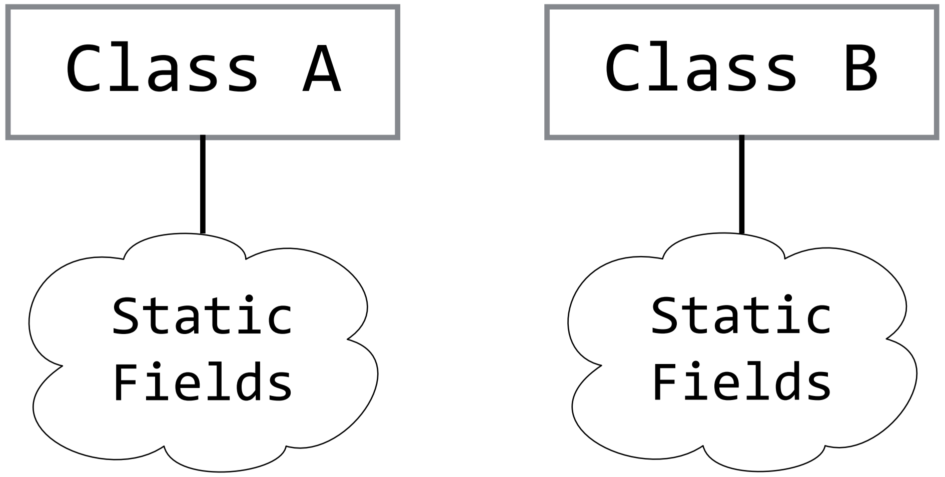
We think in terms of *object graphs*



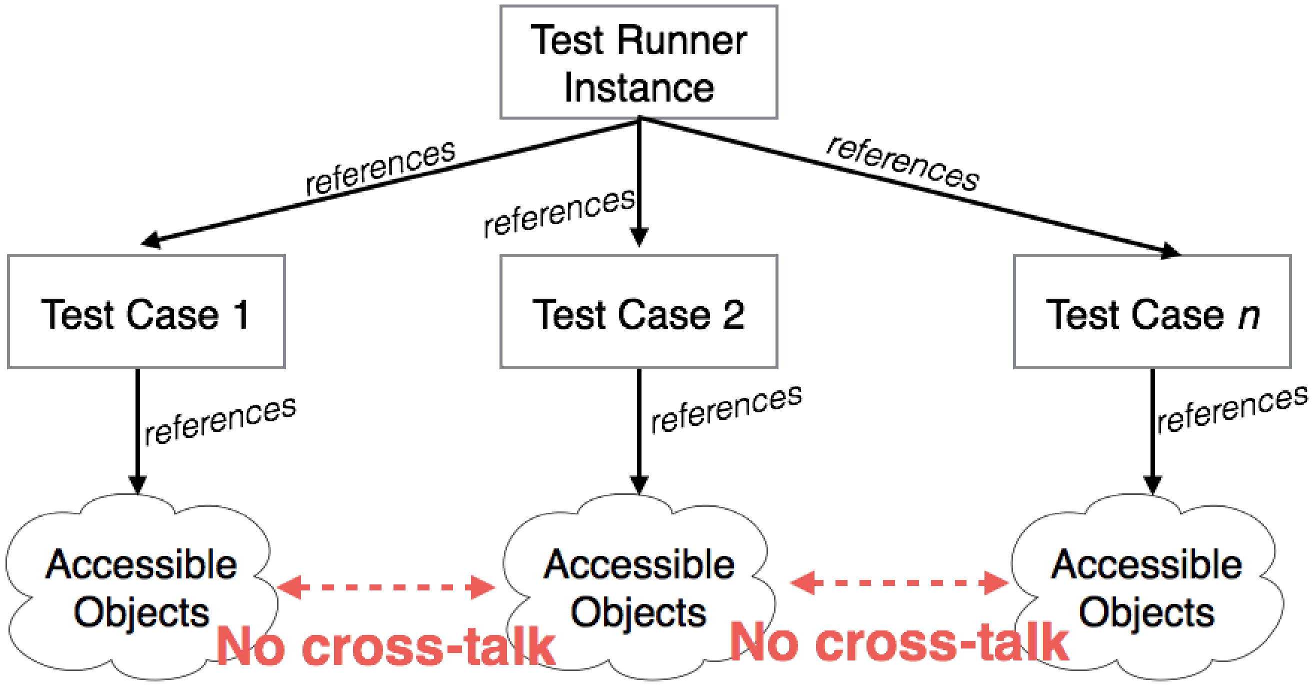
How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

We think in terms of *object graphs*



Static fields: owned by a class, NOT by an instance



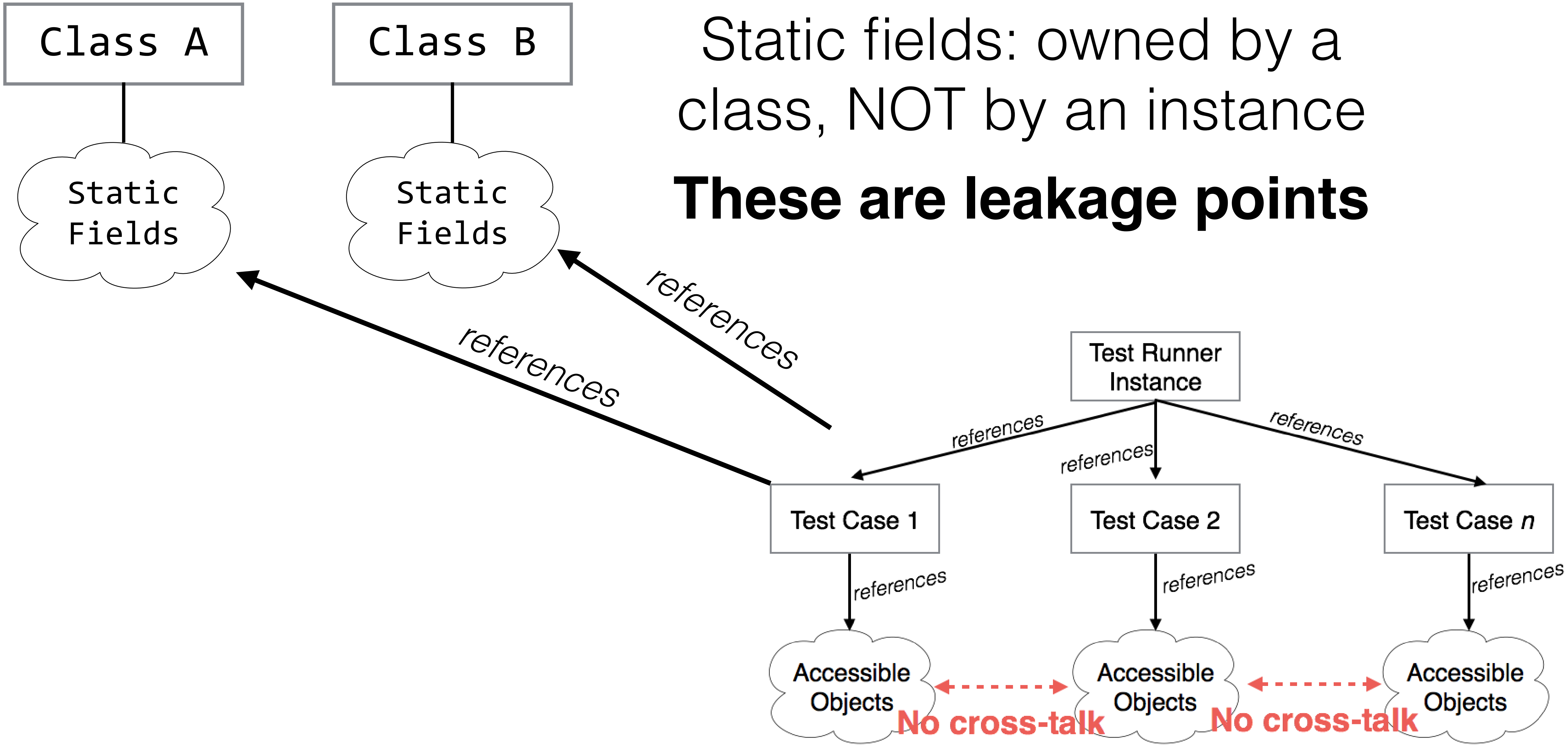
How do Tests Leak Data?

Java is **memory-managed**, and **object oriented**

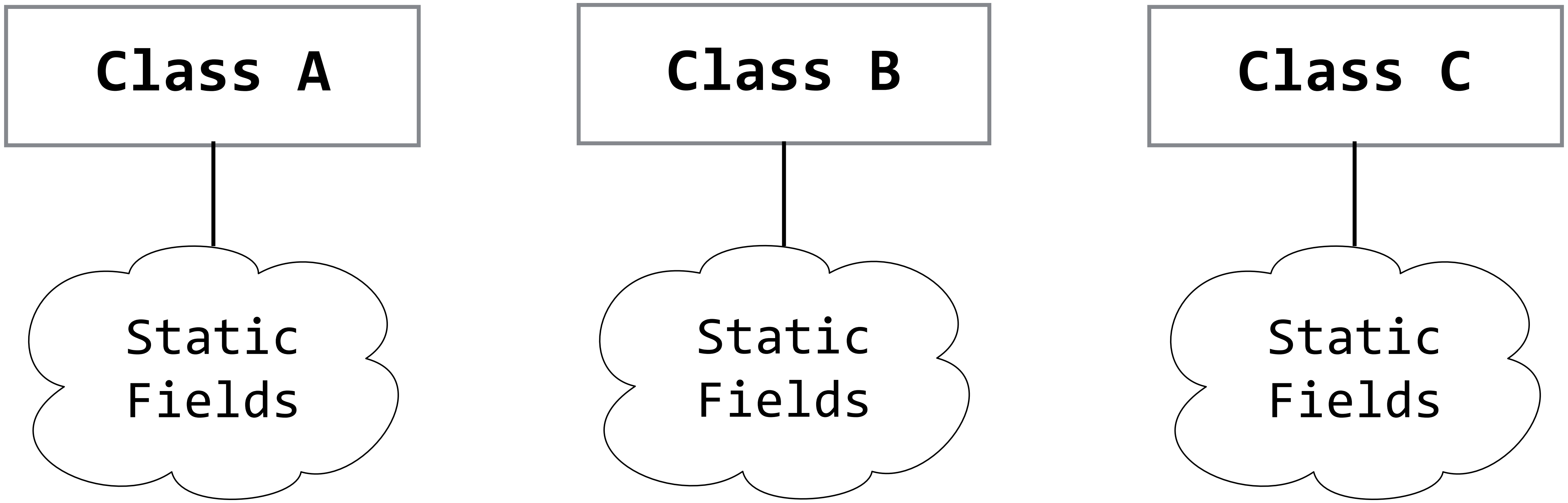
We think in terms of *object graphs*

Static fields: owned by a class, NOT by an instance

These are leakage points



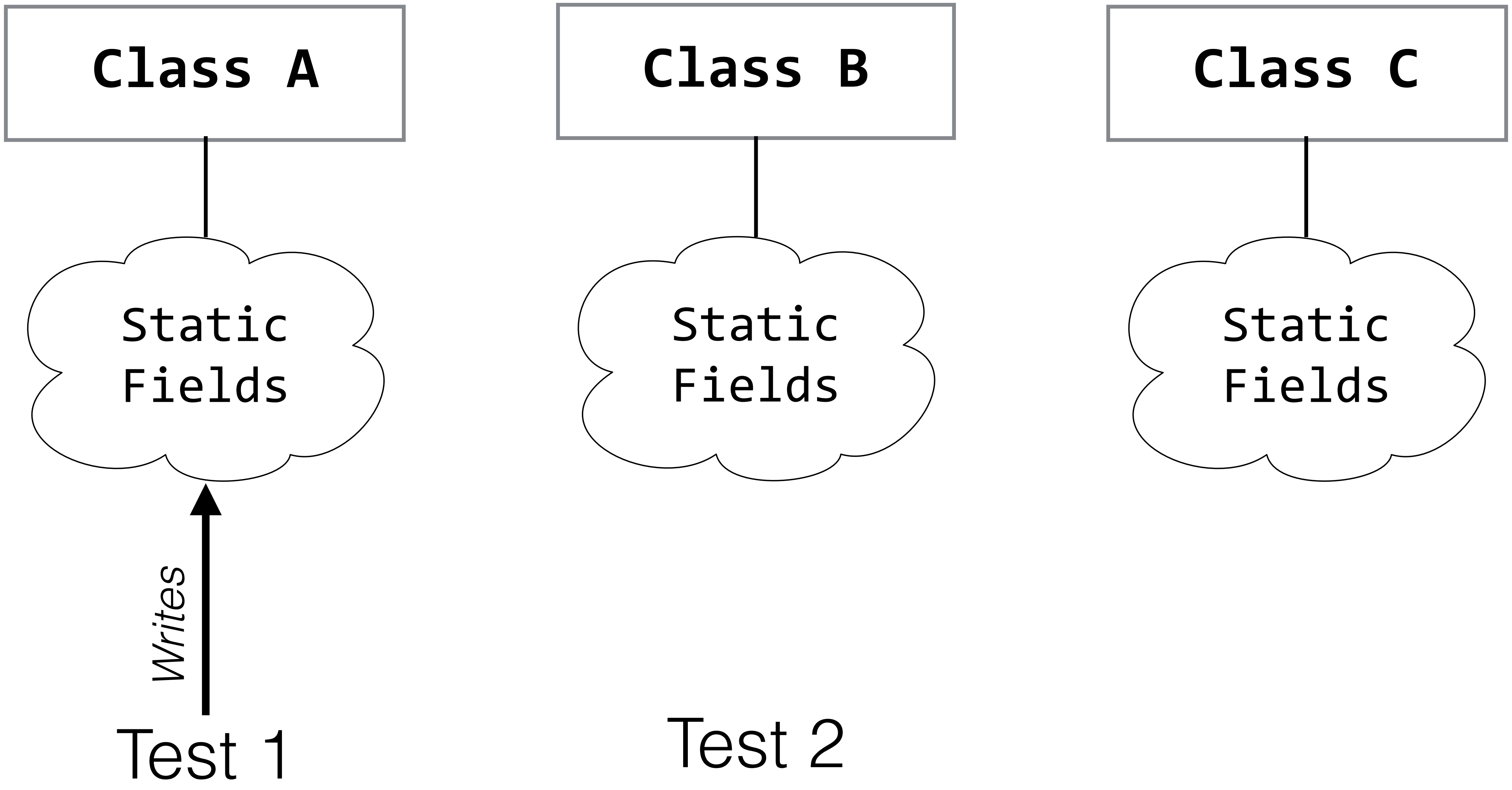
Isolating Side Effects



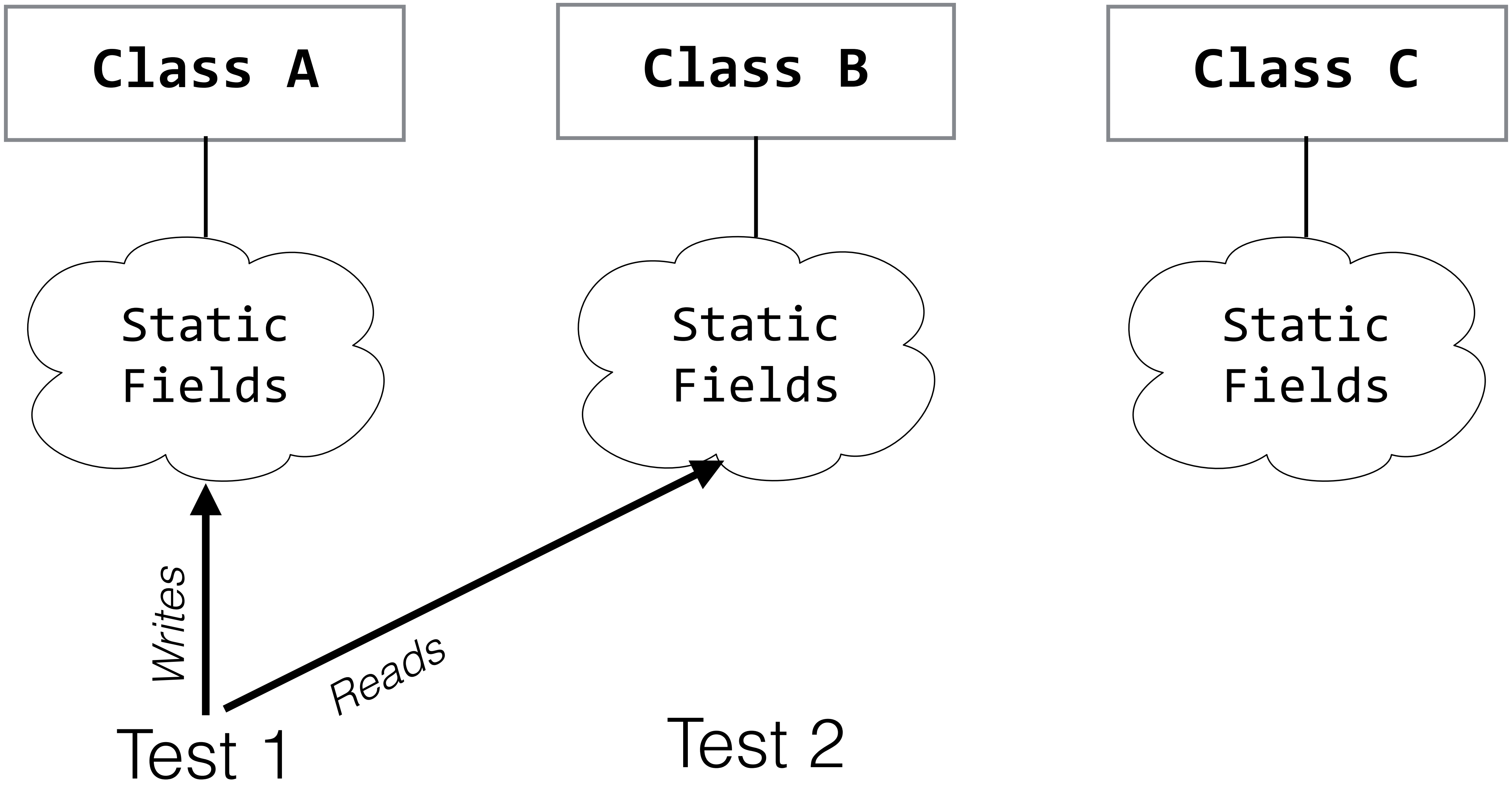
Test 1

Test 2

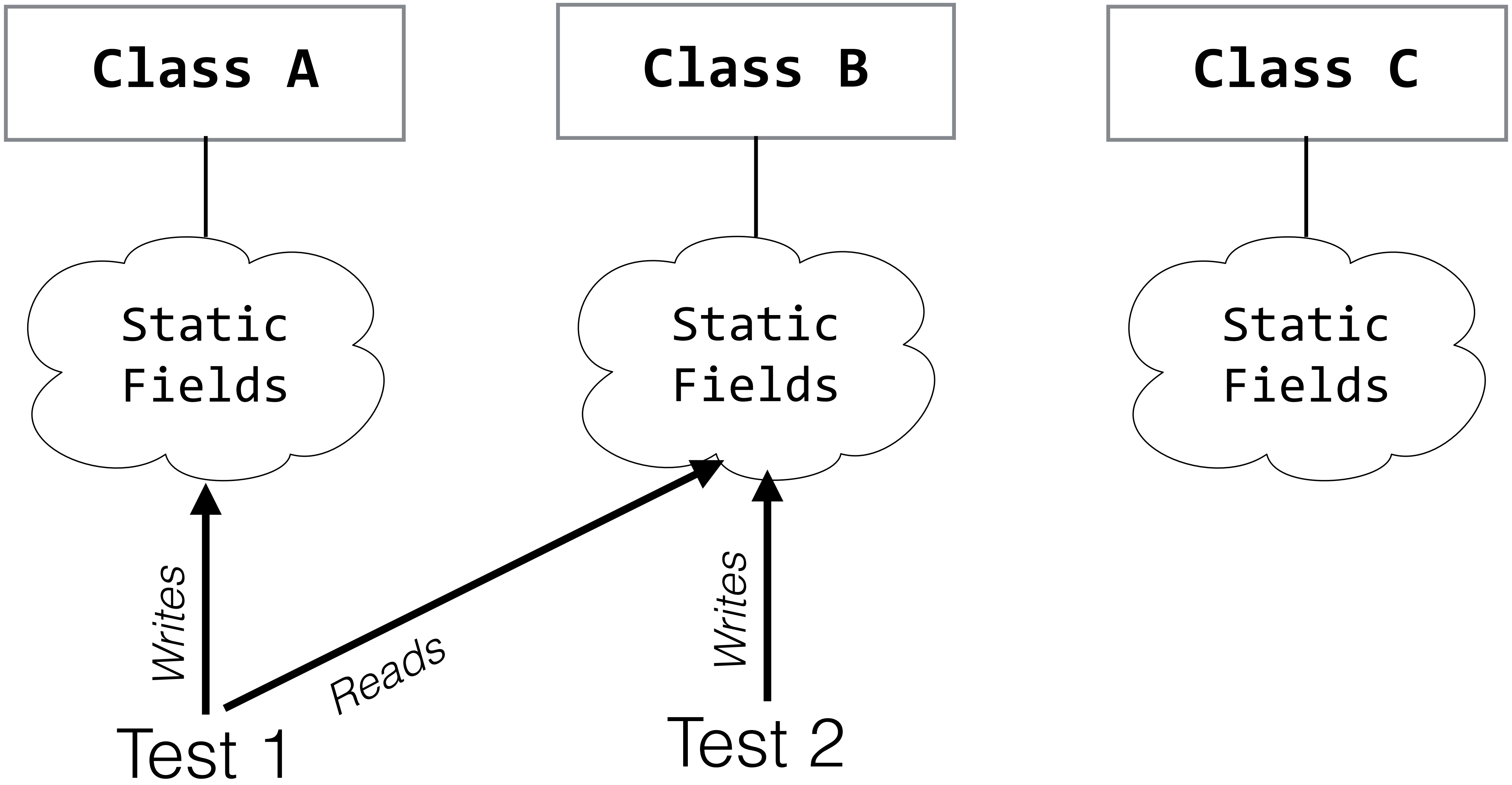
Isolating Side Effects



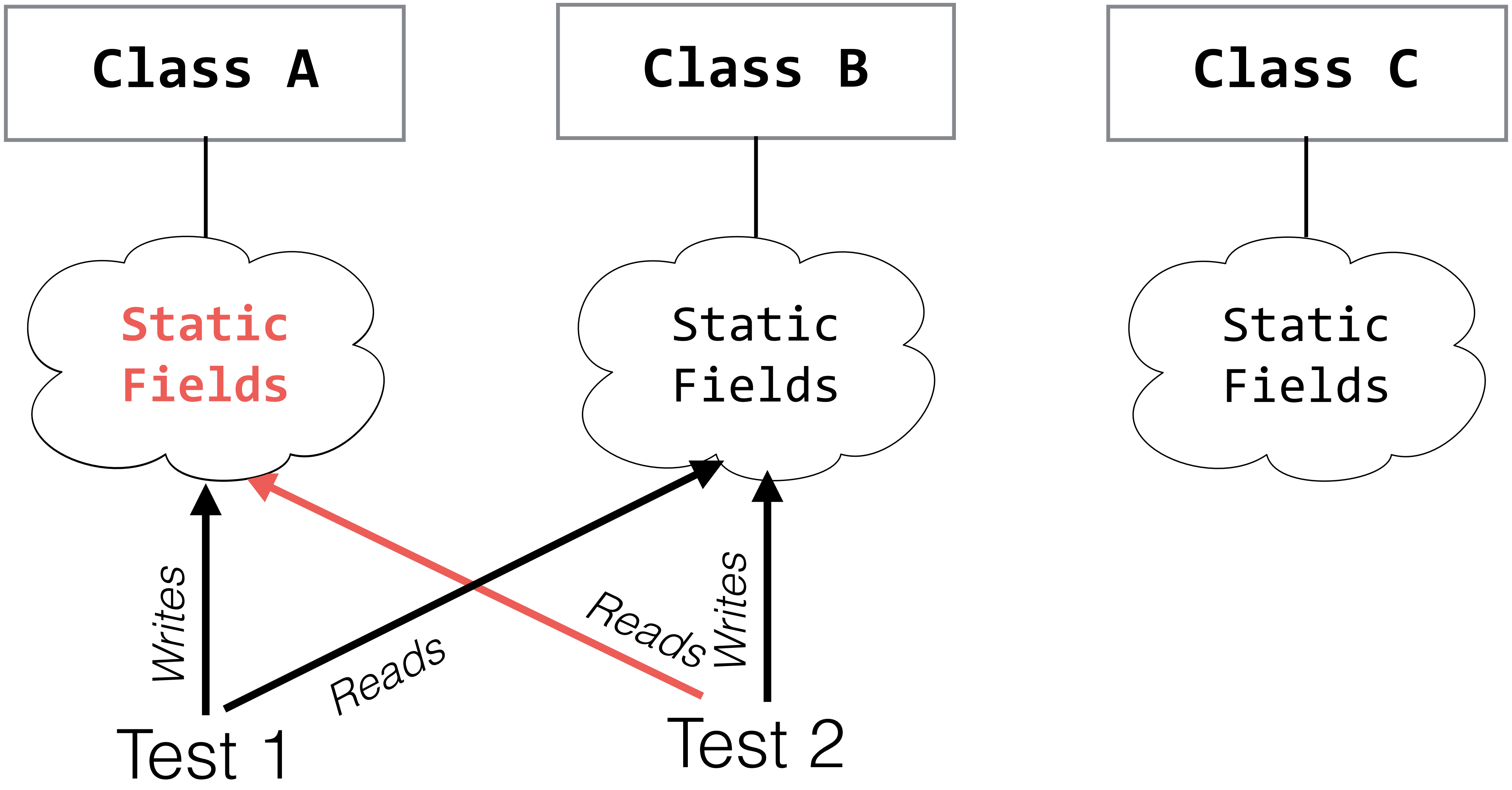
Isolating Side Effects



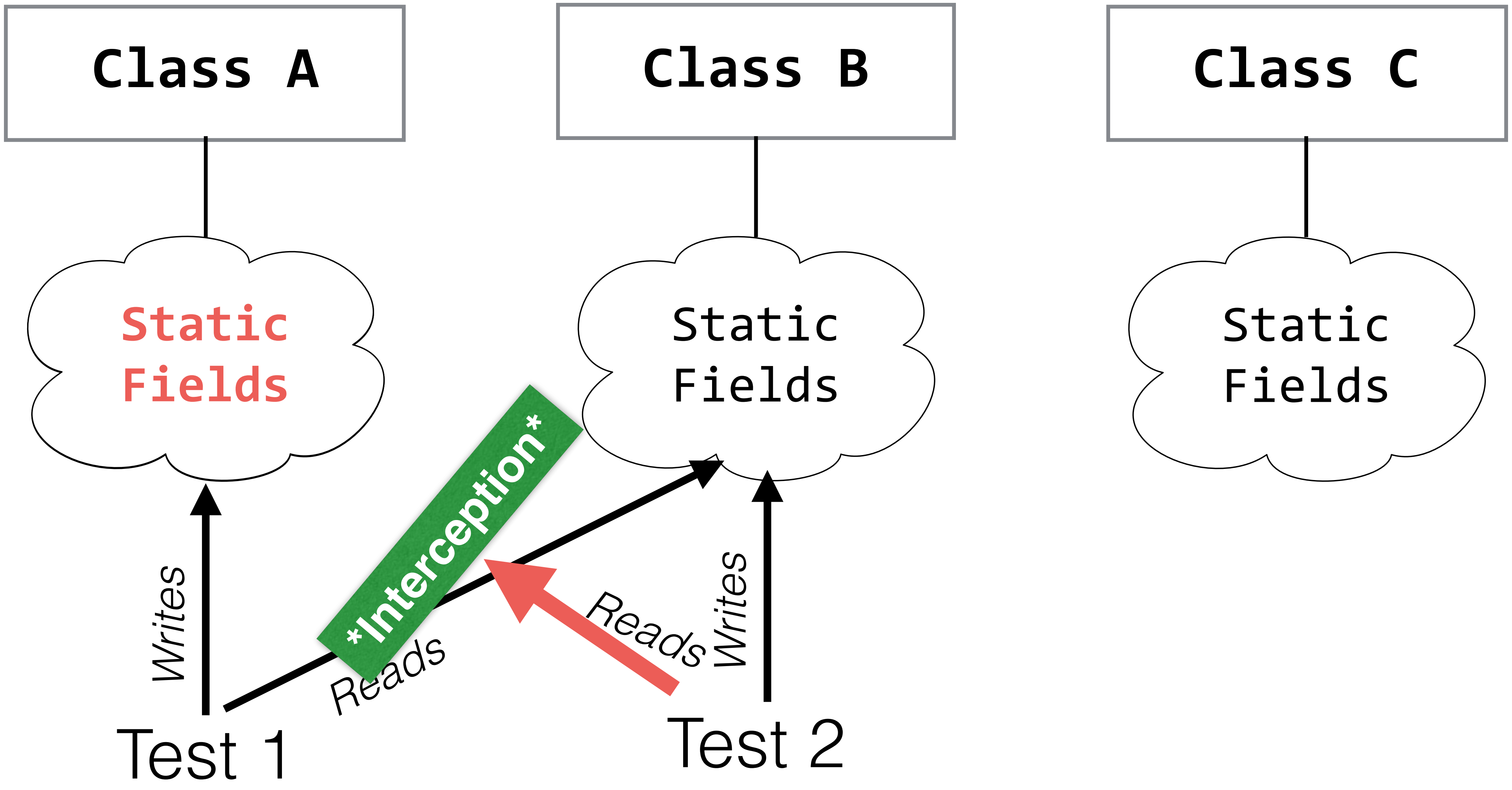
Isolating Side Effects



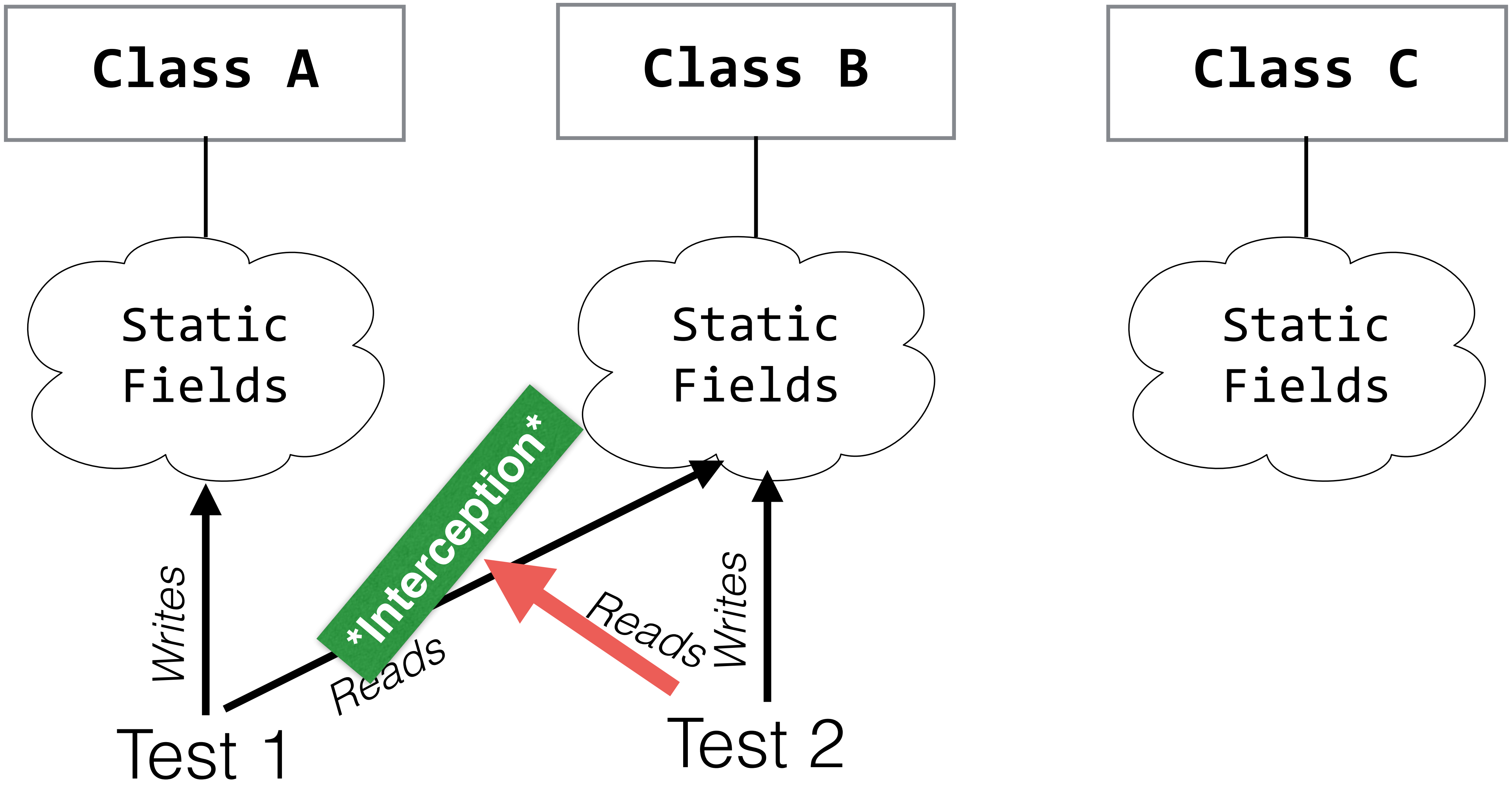
Isolating Side Effects



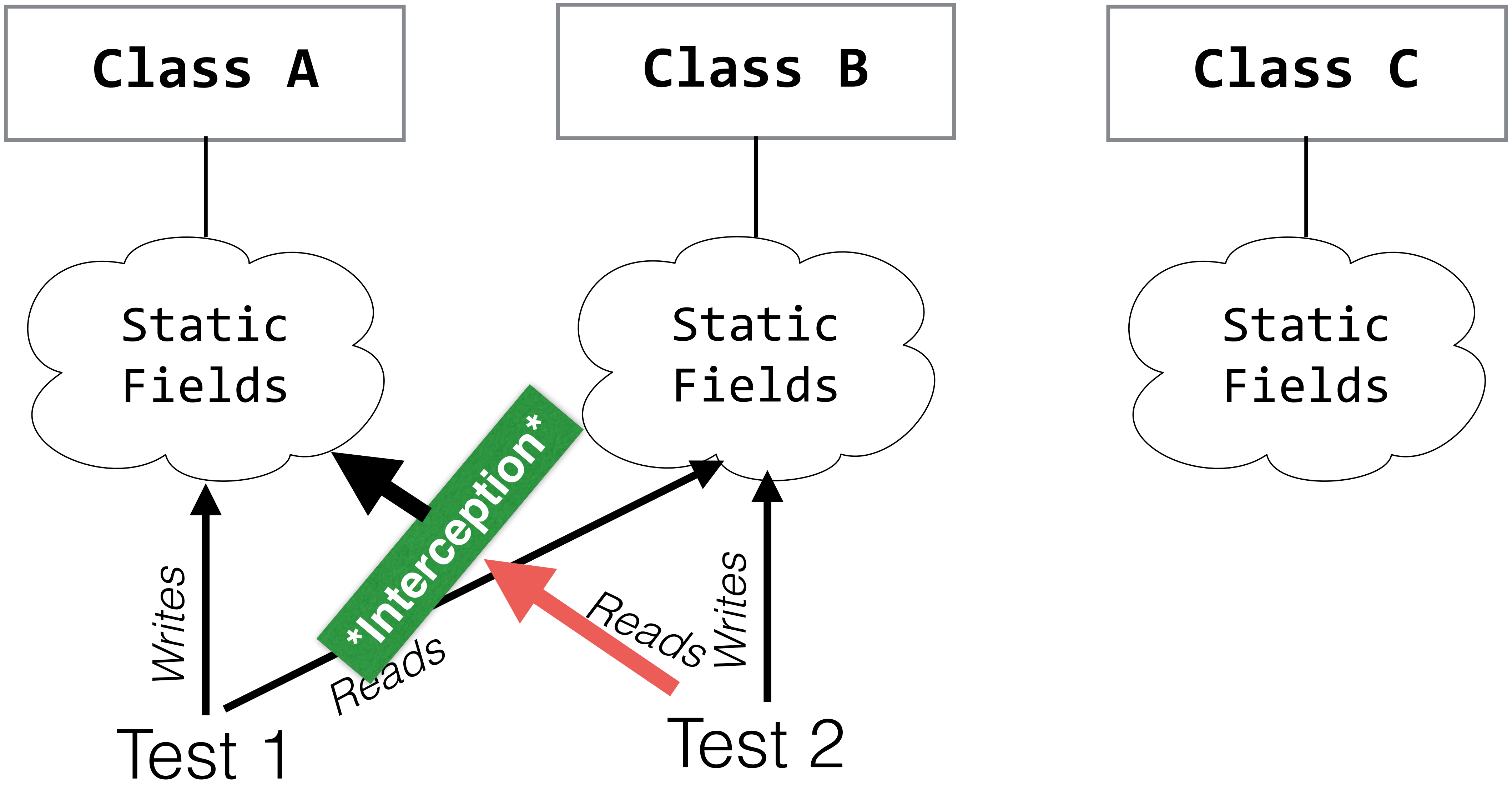
Isolating Side Effects



Isolating Side Effects



Isolating Side Effects



Class A

Class B

Class C

Static Fields

Static Fields

Static Fields

Writes

Reads

Interception

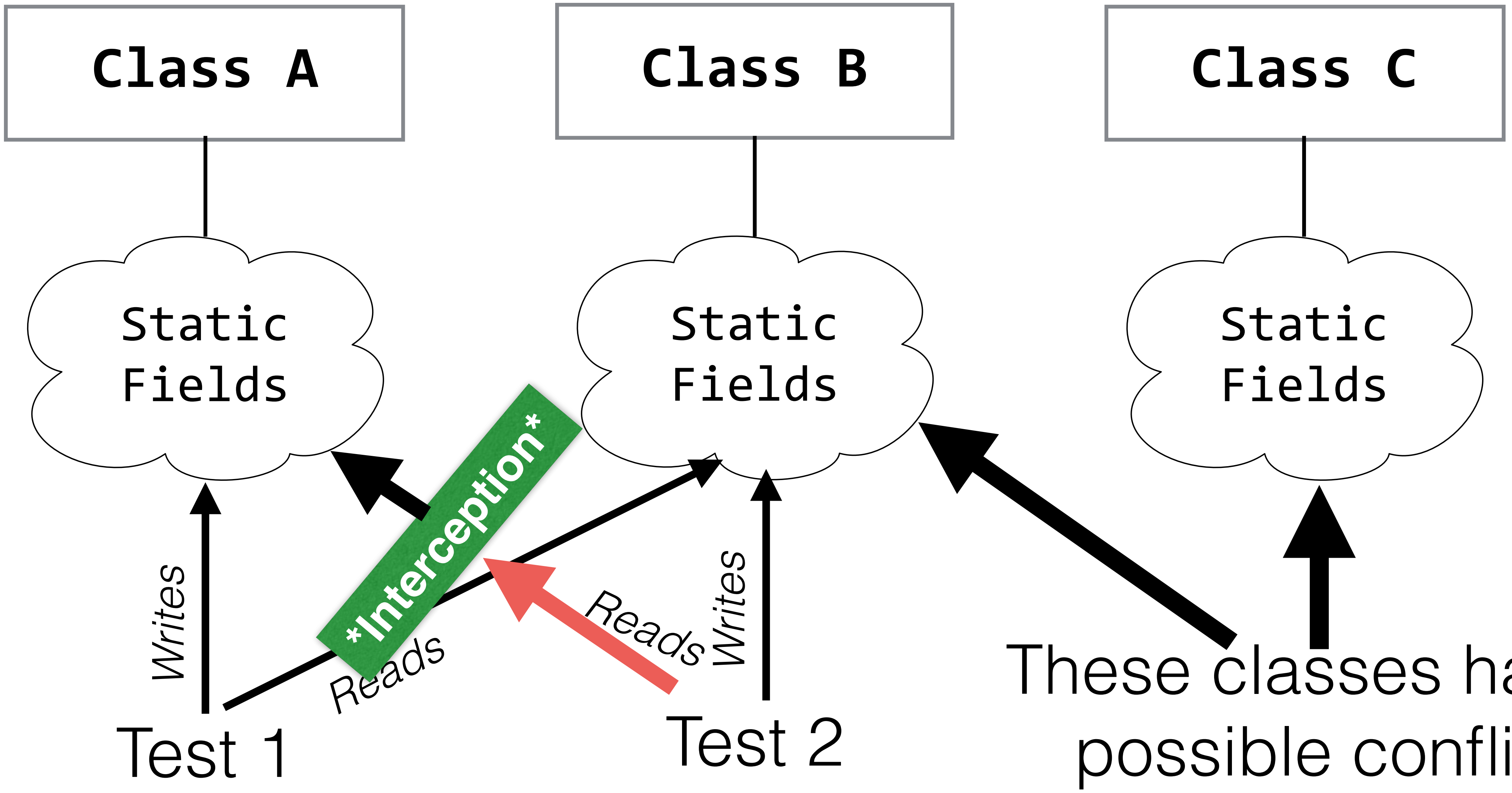
Reads

Writes

Test 1

Test 2

Isolating Side Effects

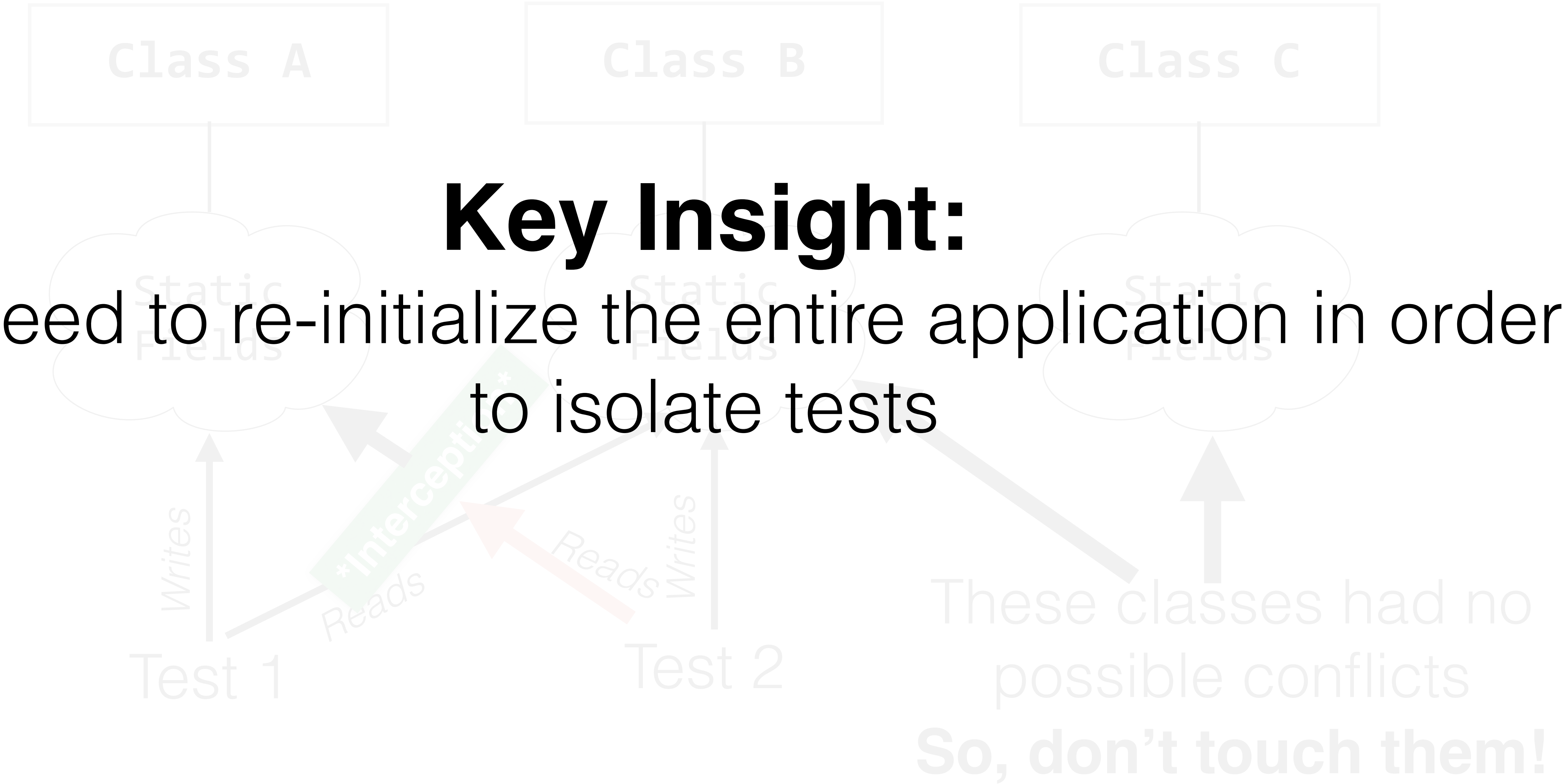


These classes had no possible conflicts
So, don't touch them!

Isolating Side Effects

Key Insight:

No need to re-initialize the entire application in order to isolate tests



VMVM: Unit Test Virtualization

- Isolates in-memory side effects, just like restarting JVM

VMVM: Unit Test Virtualization

- Isolates in-memory side effects, just like restarting JVM
- Integrates easily with ant, maven, junit

VMVM: Unit Test Virtualization

- Isolates in-memory side effects, just like restarting JVM
- Integrates easily with ant, maven, junit
- Implemented completely with application byte code instrumentation

VMVM: Unit Test Virtualization

- Isolates in-memory side effects, just like restarting JVM
- Integrates easily with ant, maven, junit
- Implemented completely with application byte code instrumentation
- No changes to JVM, no access to source code required

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time

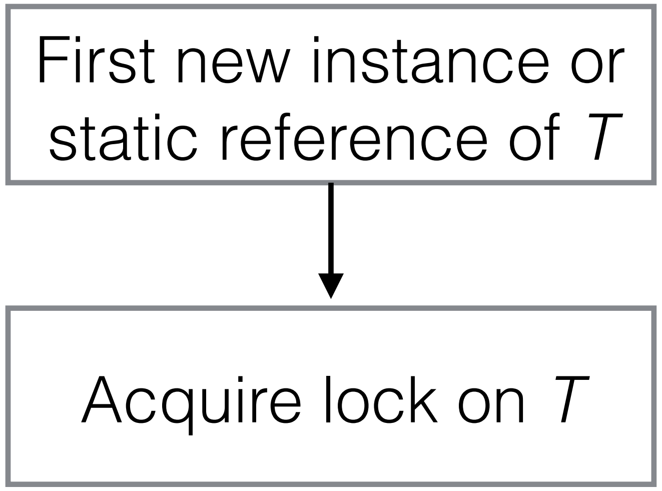
Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time

First new instance or static reference of *T*

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time

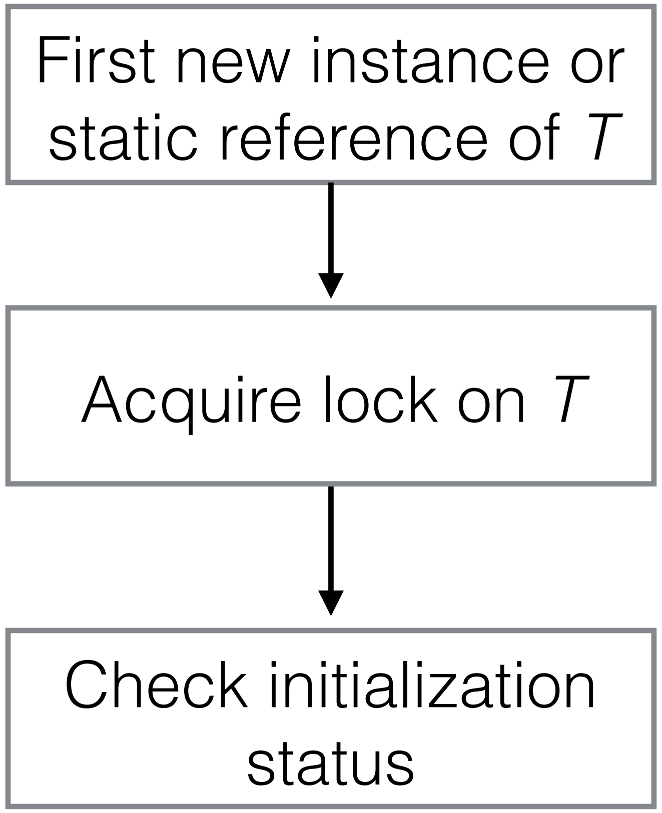


First new instance or static reference of *T*

Acquire lock on *T*

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time

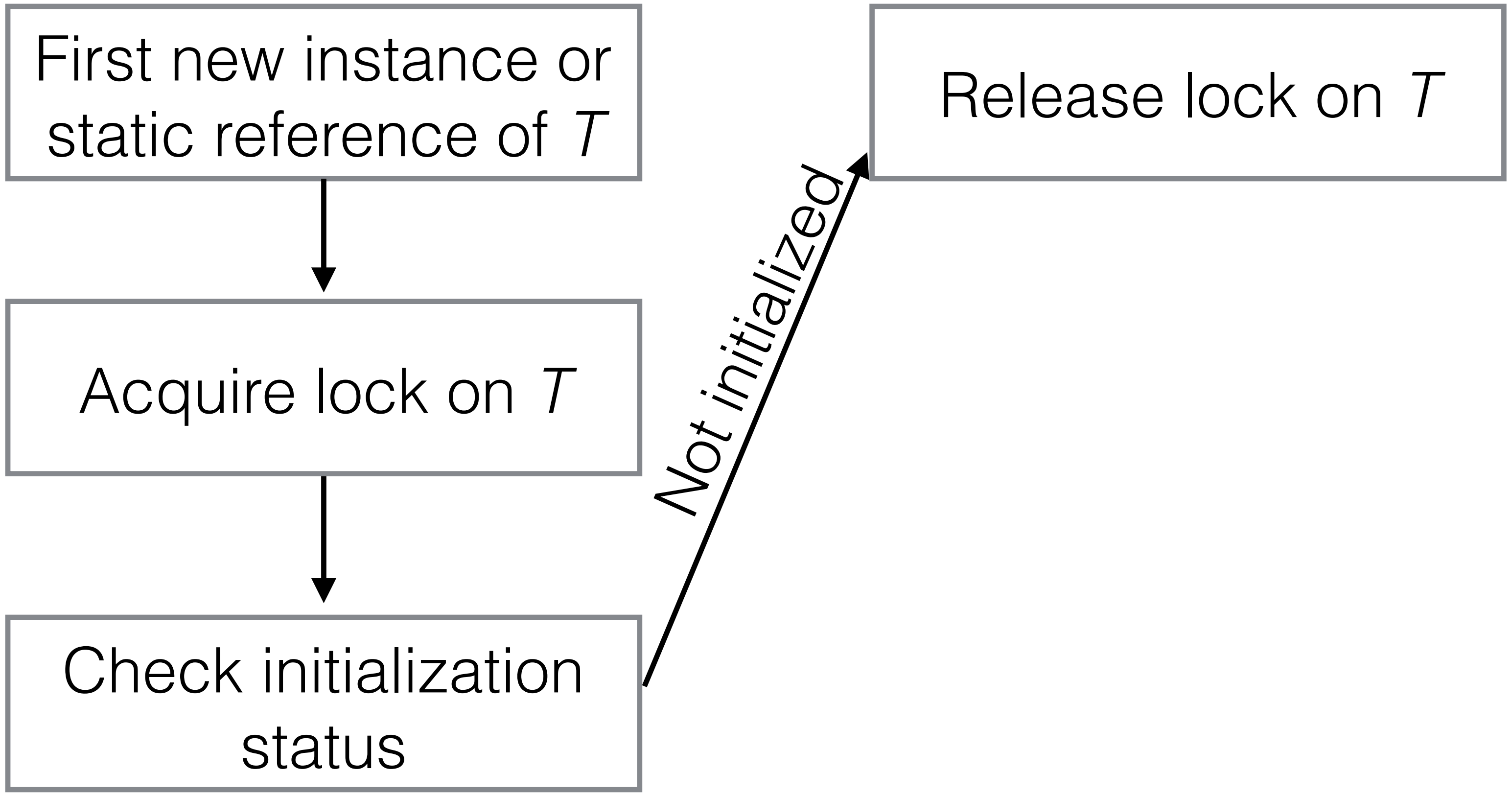

First new instance or static reference of *T*

Acquire lock on *T*

Check initialization status

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



First new instance or static reference of *T*

Acquire lock on *T*

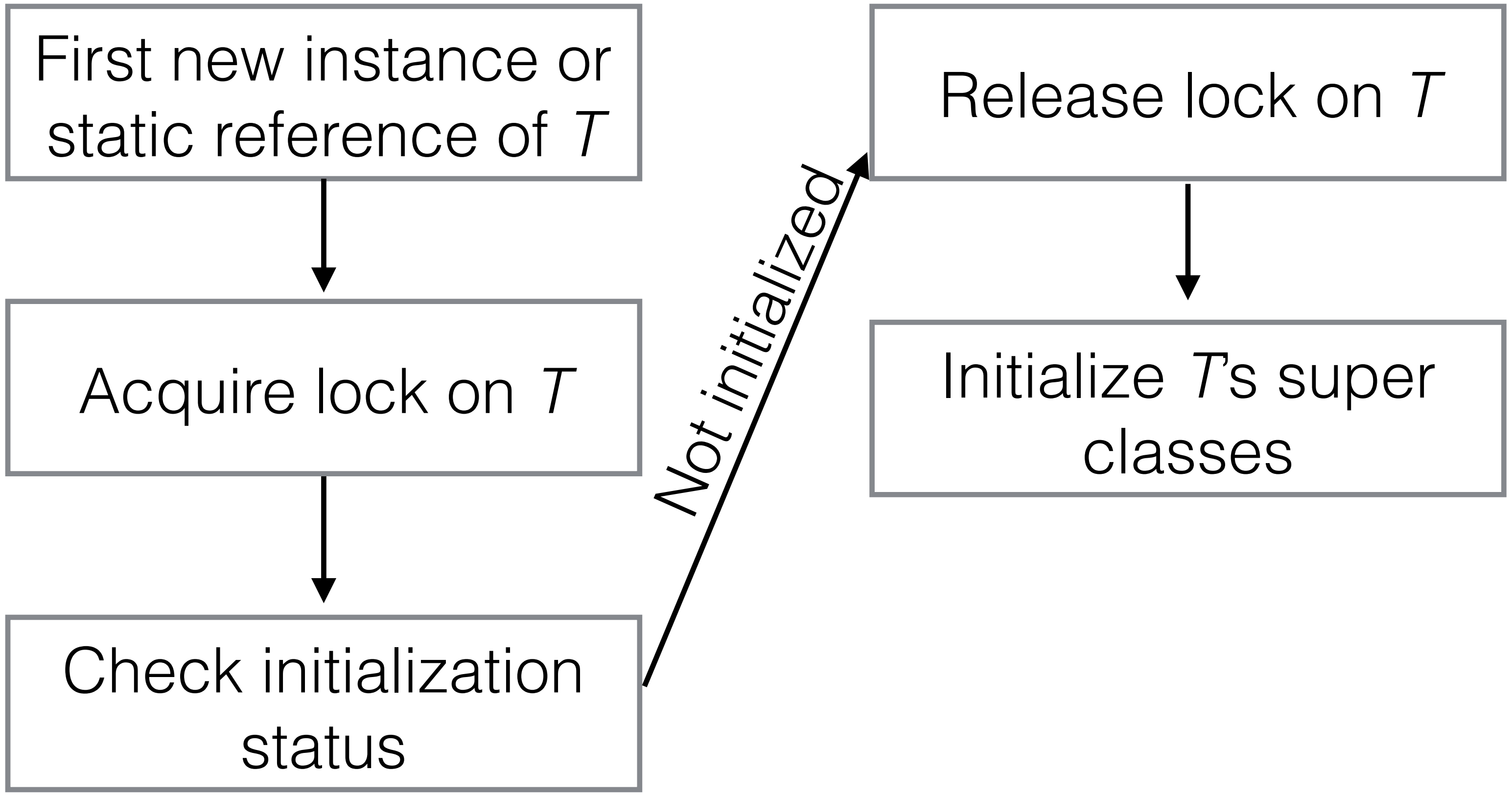
Check initialization status

Not initialized

Release lock on *T*

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



First new instance or static reference of *T*

Acquire lock on *T*

Check initialization status

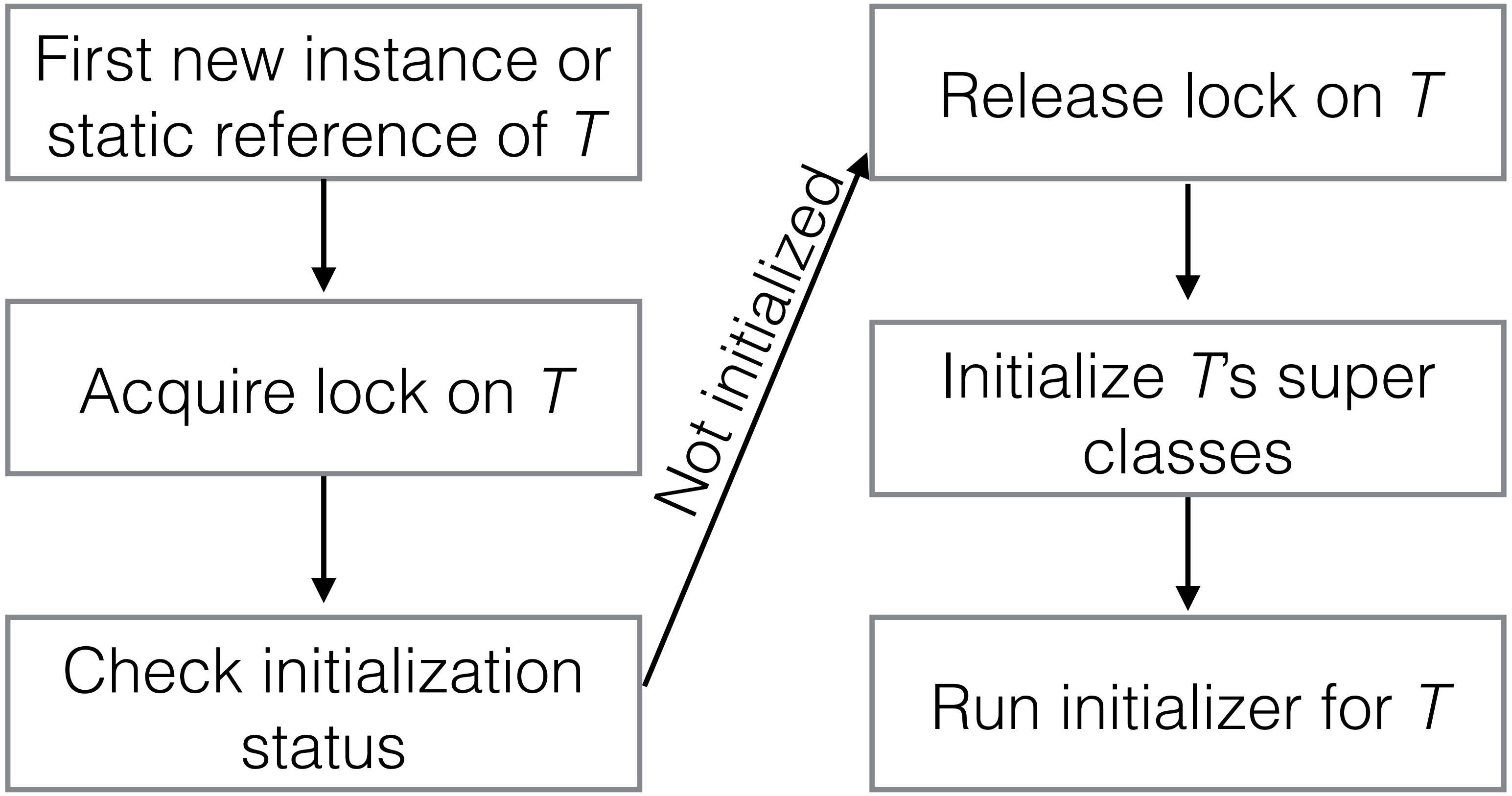
Not initialized

Release lock on *T*

Initialize *T*'s super classes

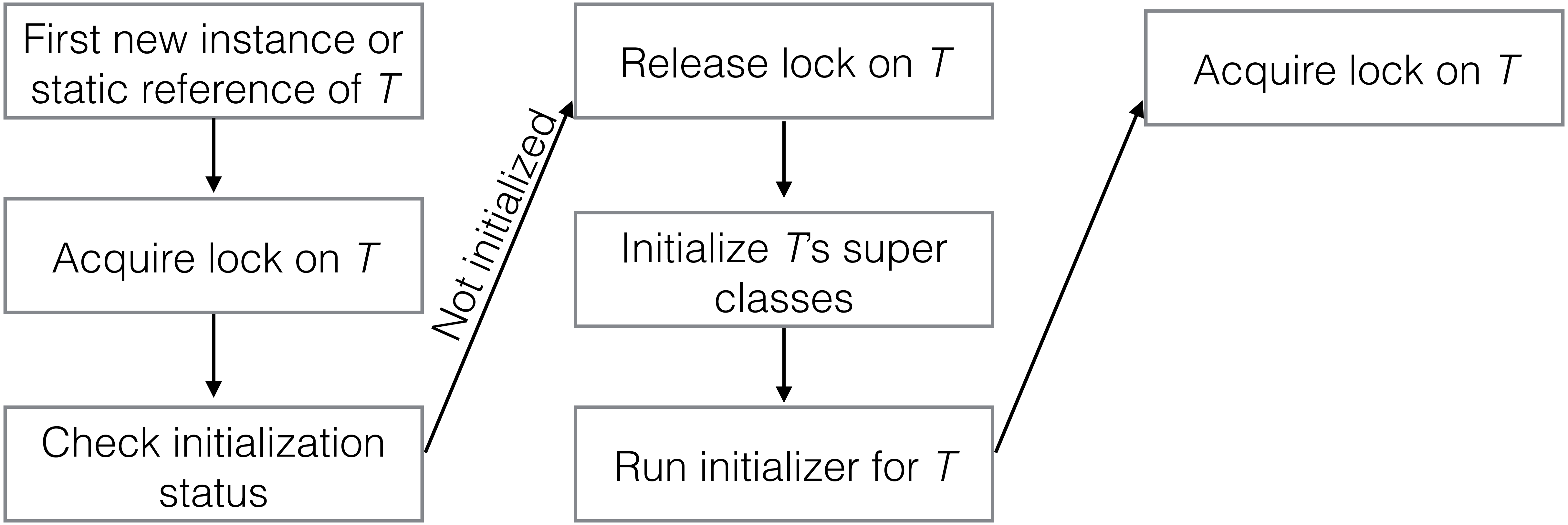
Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time


First new instance or static reference of *T*
Acquire lock on *T*
Check initialization status
Not initialized
Release lock on *T*
Initialize *T*'s super classes
Run initializer for *T*

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



First new instance or static reference of *T*

Acquire lock on *T*

Check initialization status

Not initialized

Release lock on *T*

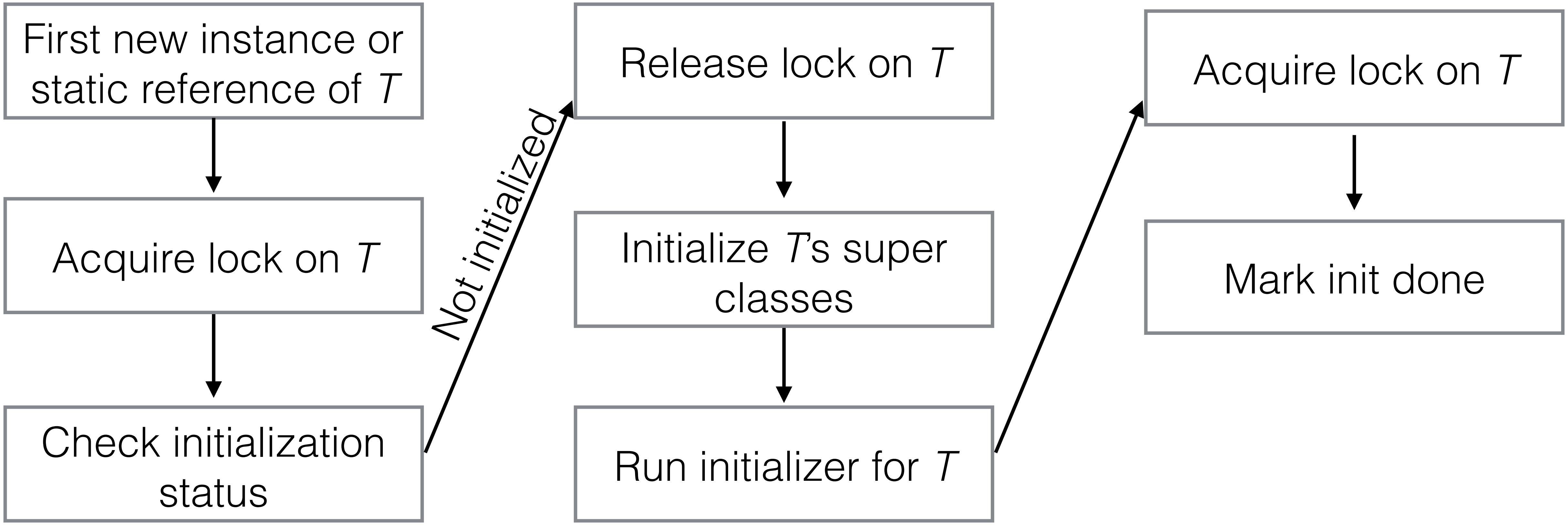
Initialize *T*'s super classes

Run initializer for *T*

Acquire lock on *T*

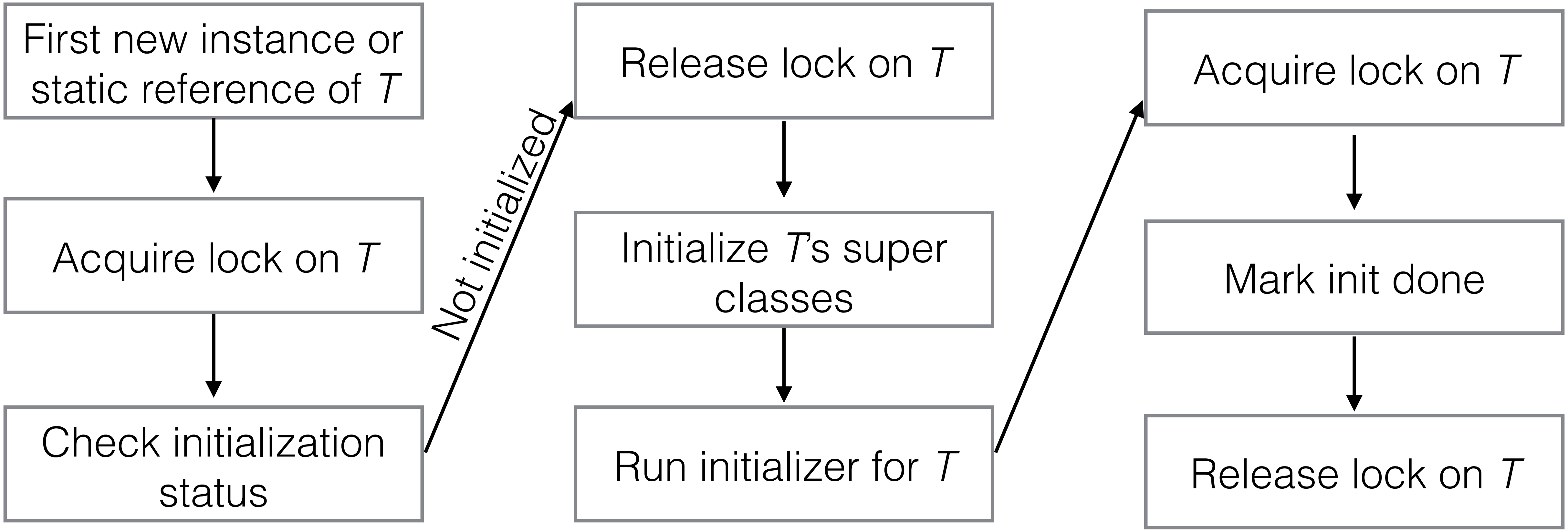
Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



First new instance or static reference of *T*

Acquire lock on *T*

Check initialization status

Release lock on *T*

Initialize *T*'s super classes

Run initializer for *T*

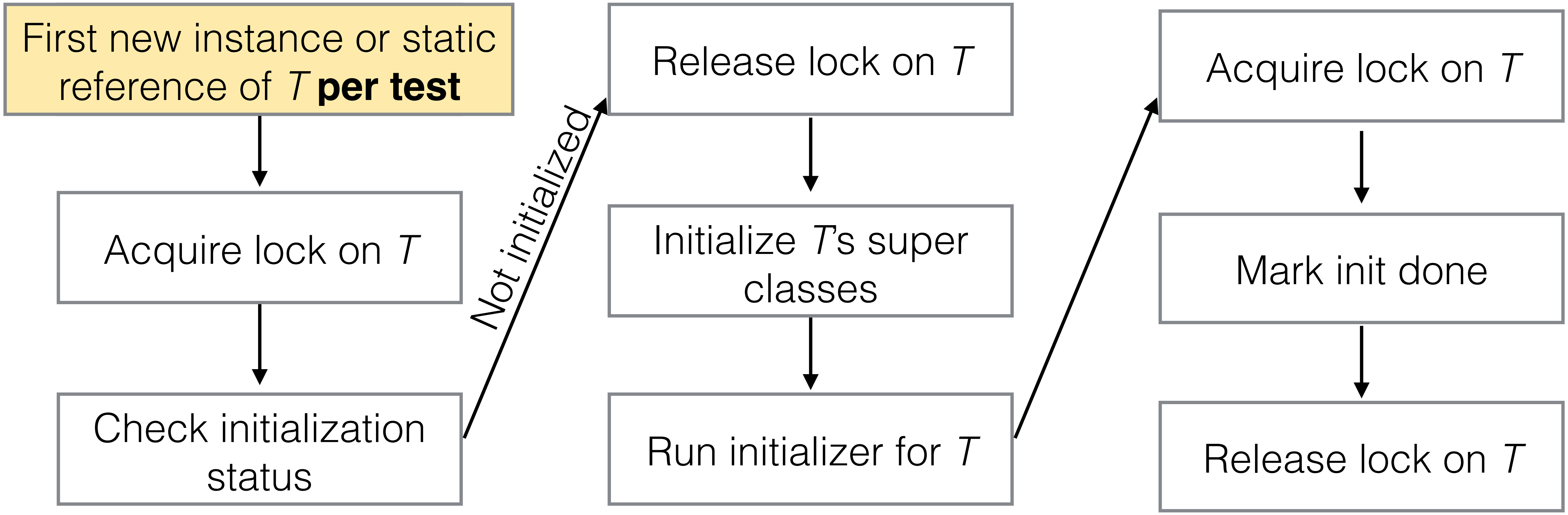
Acquire lock on *T*

Mark init done

Release lock on *T*

Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



First new instance or static reference of *T* **per test**

Acquire lock on *T*

Check initialization status

Release lock on *T*

Initialize *T*'s super classes

Run initializer for *T*

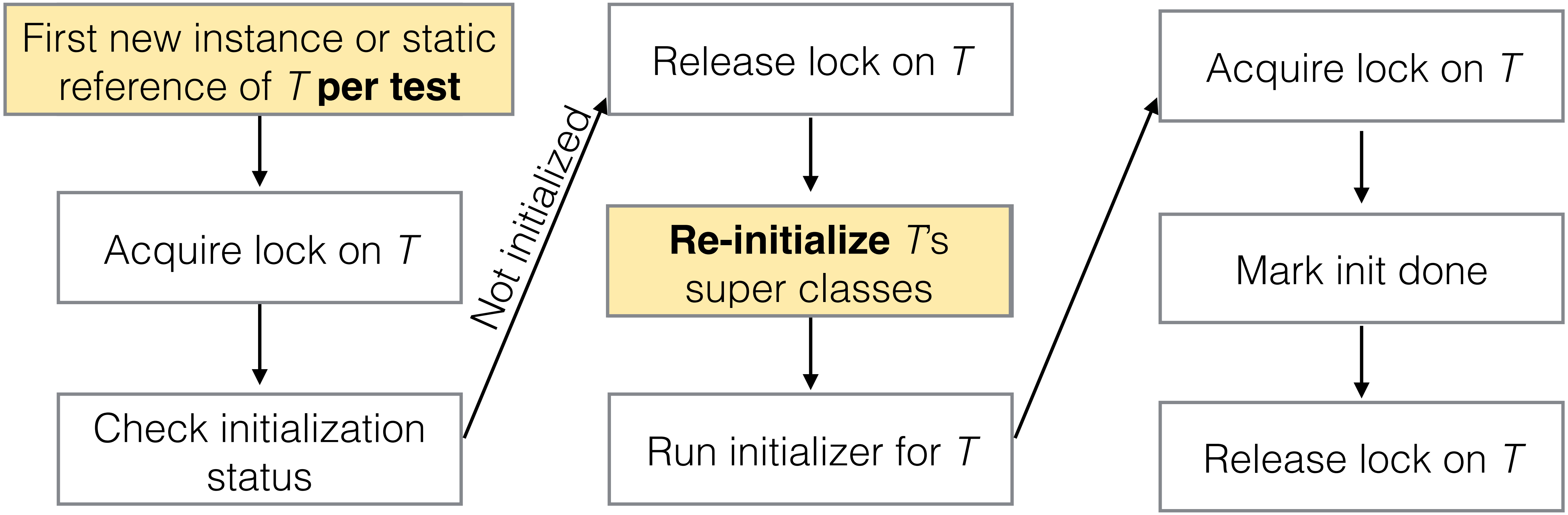
Acquire lock on *T*

Mark init done

Release lock on *T*

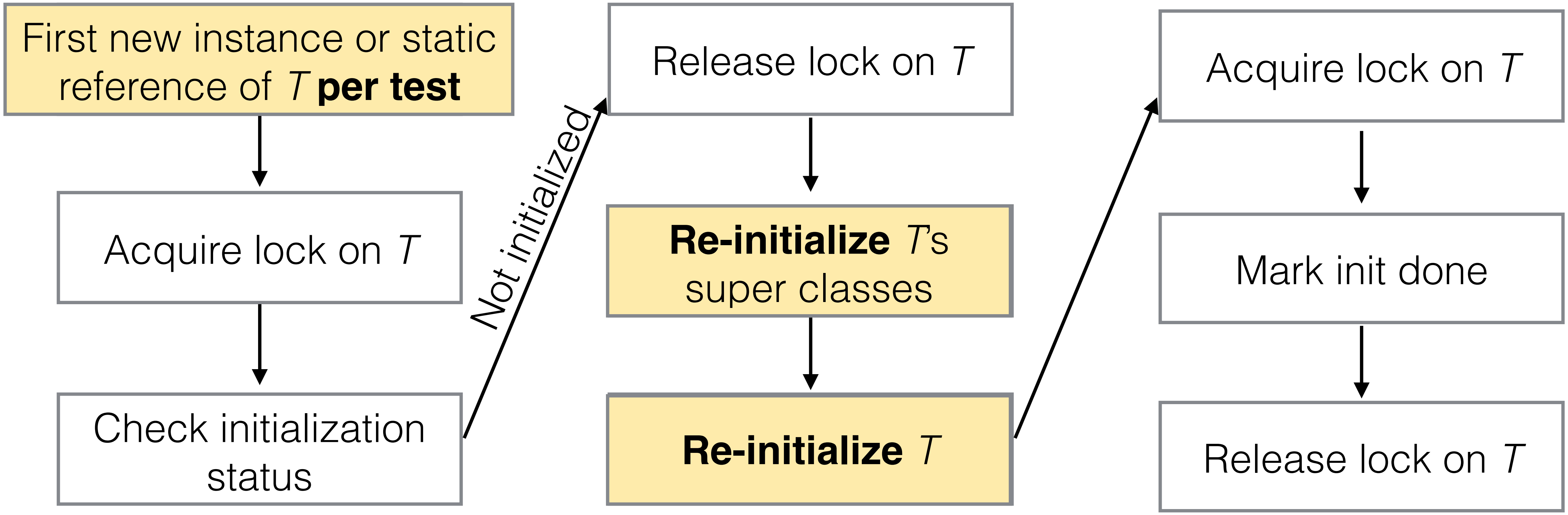
Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



Efficient Reinitialization

Emulate *exactly* what happens when a class is initialized the first time



First new instance or static reference of *T* **per test**

Acquire lock on *T*

Check initialization status

Not initialized

Release lock on *T*

Re-initialize *T*'s super classes

Re-initialize *T*

Acquire lock on *T*

Mark init done

Release lock on *T*

Efficient Reinitialization

- Does not require any modifications to the JVM and runs on commodity JVMs

Efficient Reinitialization

- Does not require any modifications to the JVM and runs on commodity JVMs
- The JVM calls a special method, <clinit> to initialize a class

Efficient Reinitialization

- Does not require any modifications to the JVM and runs on commodity JVMs
- The JVM calls a special method, <clinit> to initialize a class
- We do the same, entirely in Java

Efficient Reinitialization

- Does not require any modifications to the JVM and runs on commodity JVMs

- The JVM calls a special method, <clinit> to initialize a class

- We do the same, entirely in Java

- Add guards to trigger this process

Efficient Reinitialization

- Does not require any modifications to the JVM and runs on commodity JVMs
- The JVM calls a special method, <clinit> to initialize a class
- We do the same, entirely in Java
- Add guards to trigger this process
- Register a hook with test runner to tell us when a new test starts

Experiments

- RQ1: How does VMVM compare to Test Suite Minimization?

Experiments

- RQ1: How does VMVM compare to Test Suite Minimization?
- RQ2: What are the performance gains of VMVM?

Experiments

- RQ1: How does VMVM compare to Test Suite Minimization?

- RQ2: What are the performance gains of VMVM?

- RQ3: Does VMVM impact fault finding ability?

RQ1: VMVM vs Test Minimization

- Study design follows Zhang [ISSRE ‘11]’s evaluation of four minimization approaches

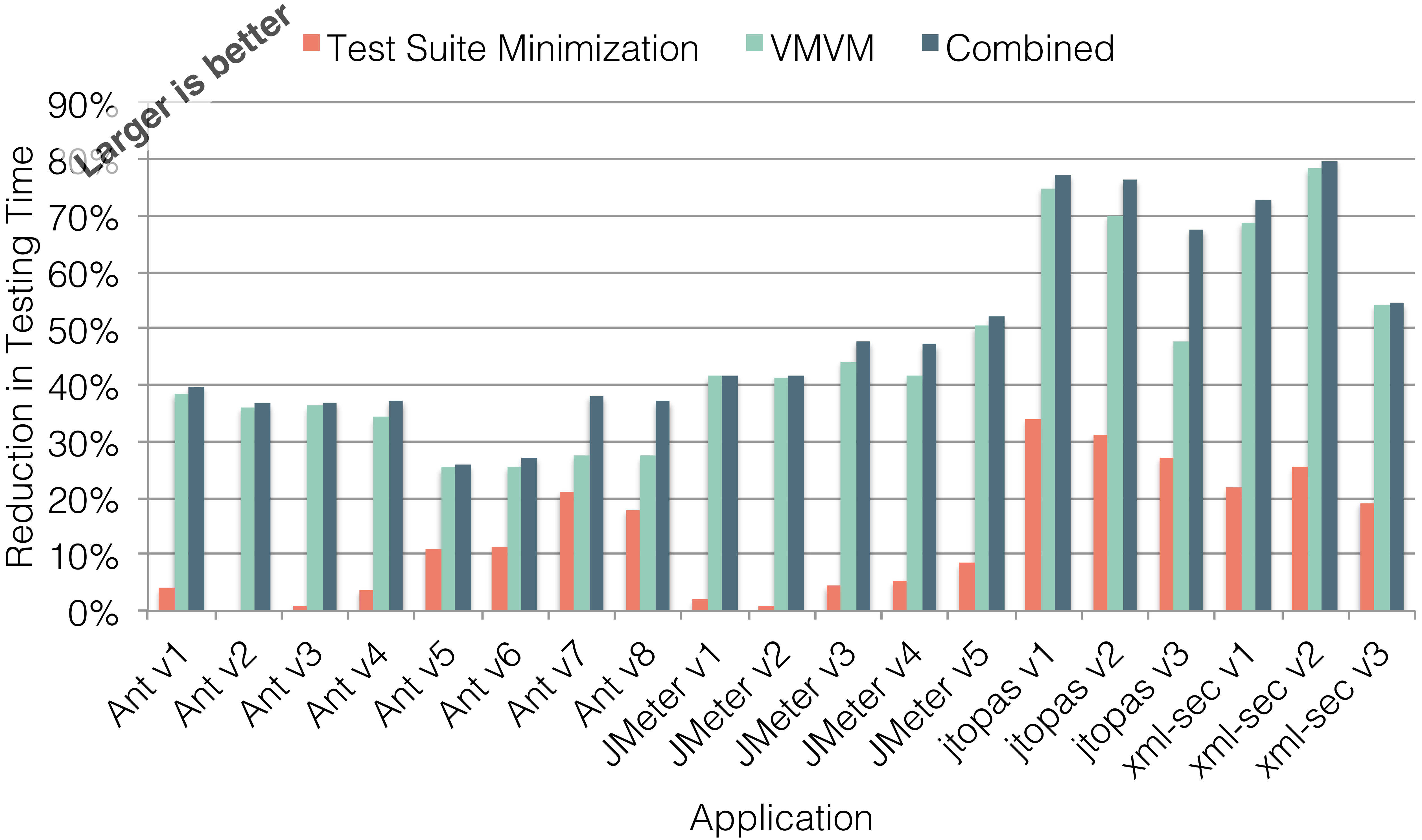
RQ1: VMVM vs Test Minimization

- Study design follows Zhang [ISSRE ‘11]’s evaluation of four minimization approaches
- Compare to the minimization technique with least impact on fault finding ability, Harrold [TOSEM ‘93]'s technique

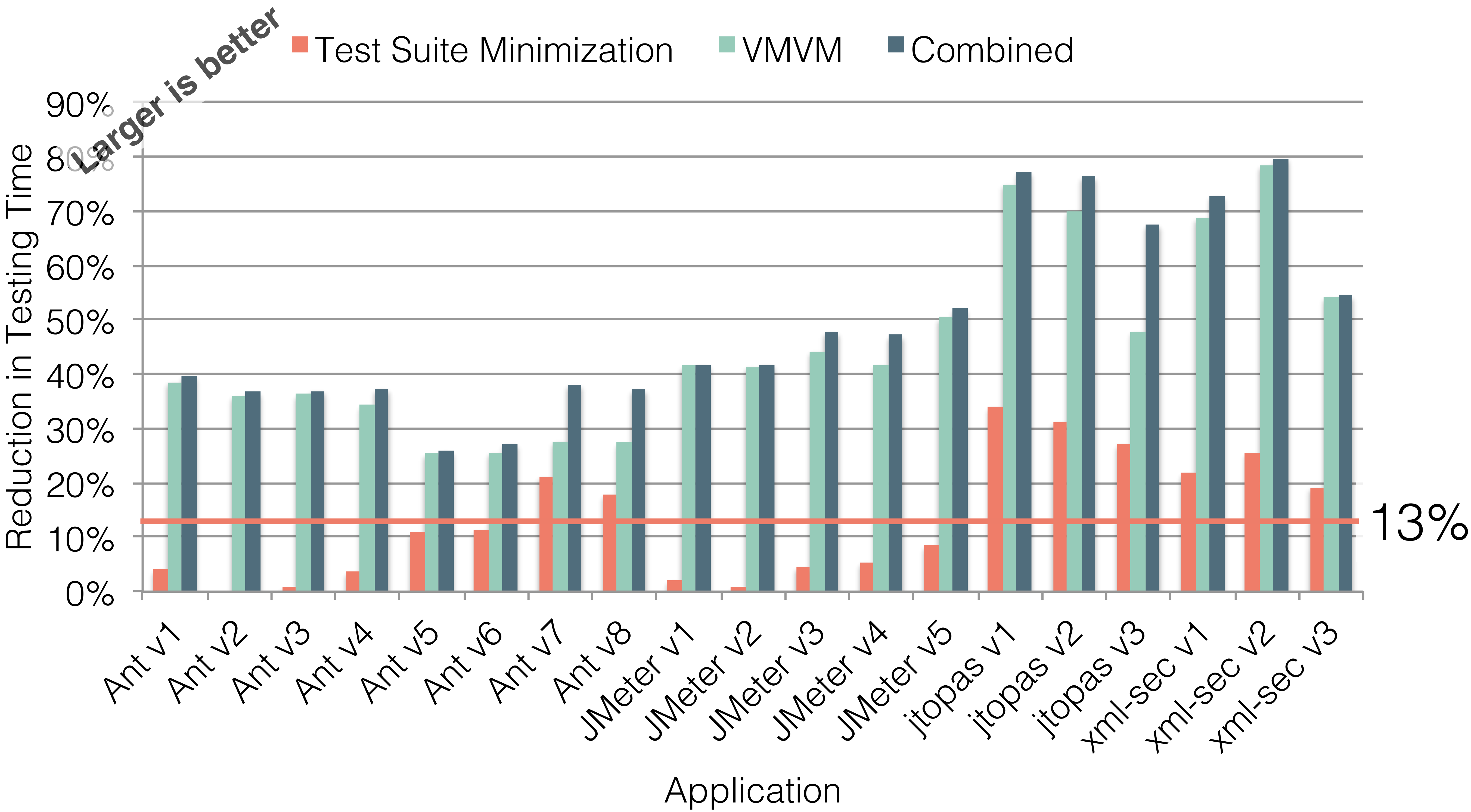
RQ1: VMVM vs Test Minimization

- Study design follows Zhang [ISSRE ‘11]’s evaluation of four minimization approaches
- Compare to the minimization technique with least impact on fault finding ability, Harrold [TOSEM ‘93]'s technique
- Study performed on the popular Software Infrastructure Repository dataset

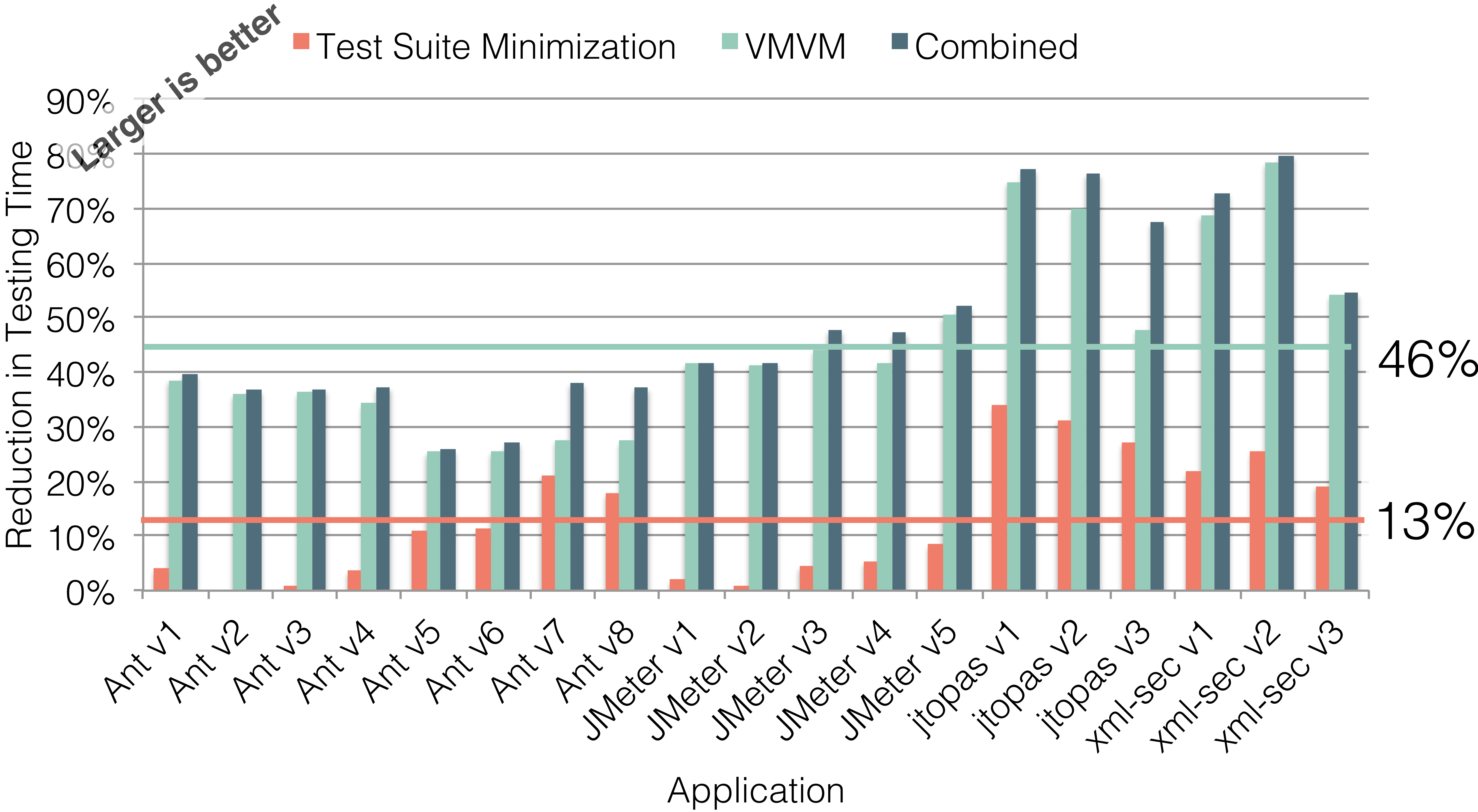
RQ1: VMVM vs Test Minimization



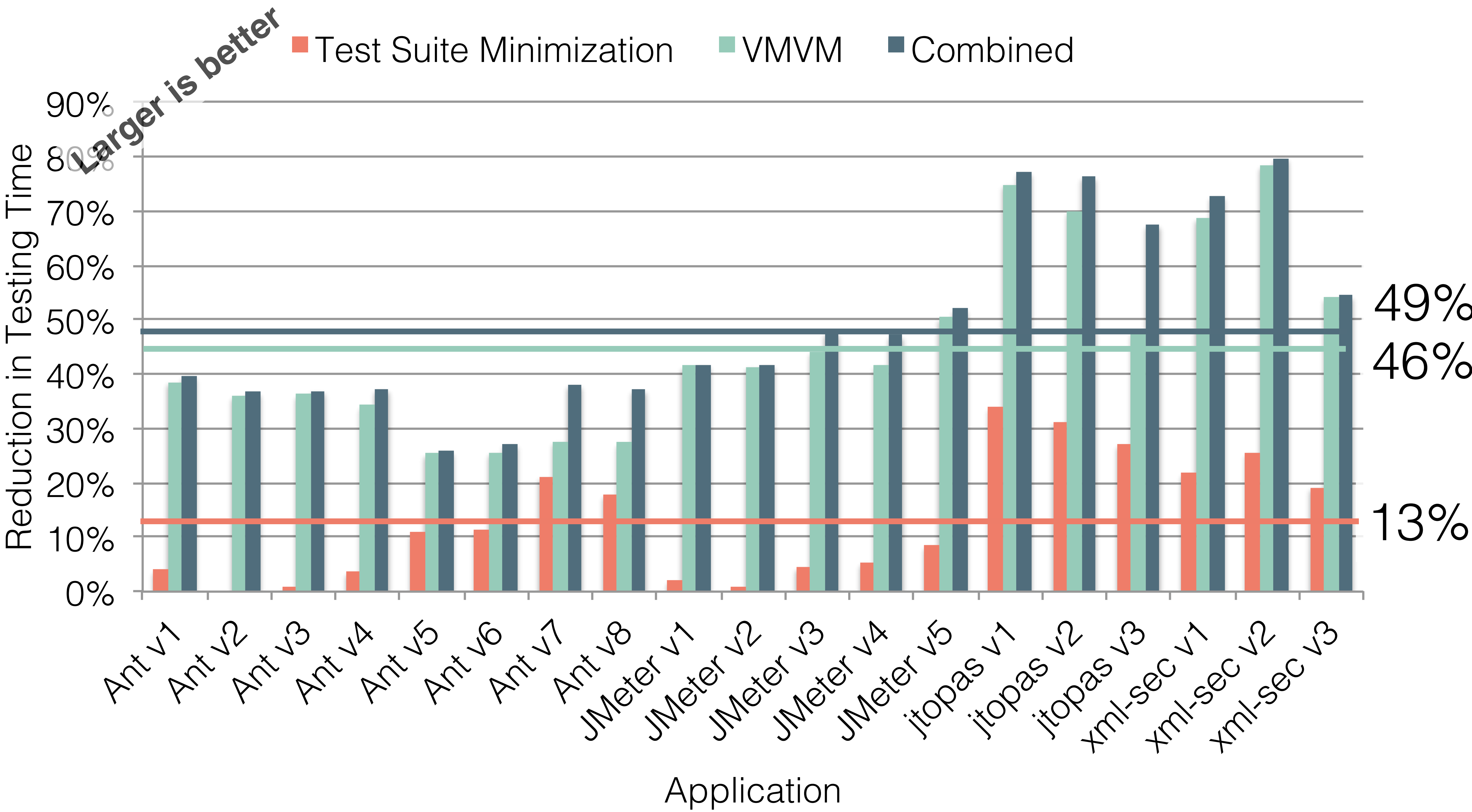
RQ1: VMVM vs Test Minimization



RQ1: VMVM vs Test Minimization



RQ1: VMVM vs Test Minimization



RQ2: Broader Evaluation

- Previous study: well-studied suite of 4 projects, which average 37,000 LoC and 51 test classes

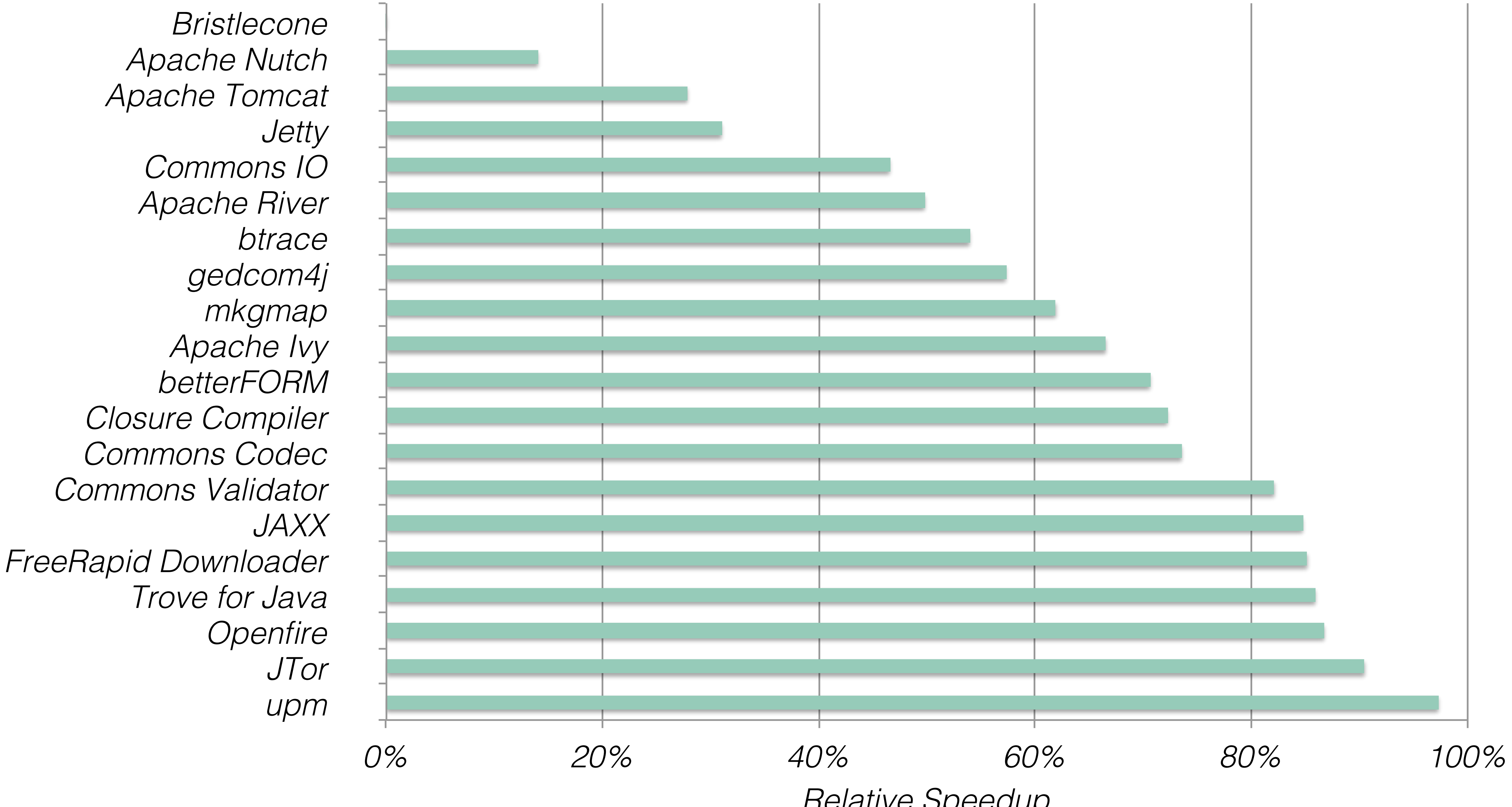
RQ2: Broader Evaluation

- Previous study: well-studied suite of 4 projects, which average 37,000 LoC and 51 test classes
- This study: manually collected repository of 20 projects, average 475,000 LoC and 56 test classes

RQ2: Broader Evaluation

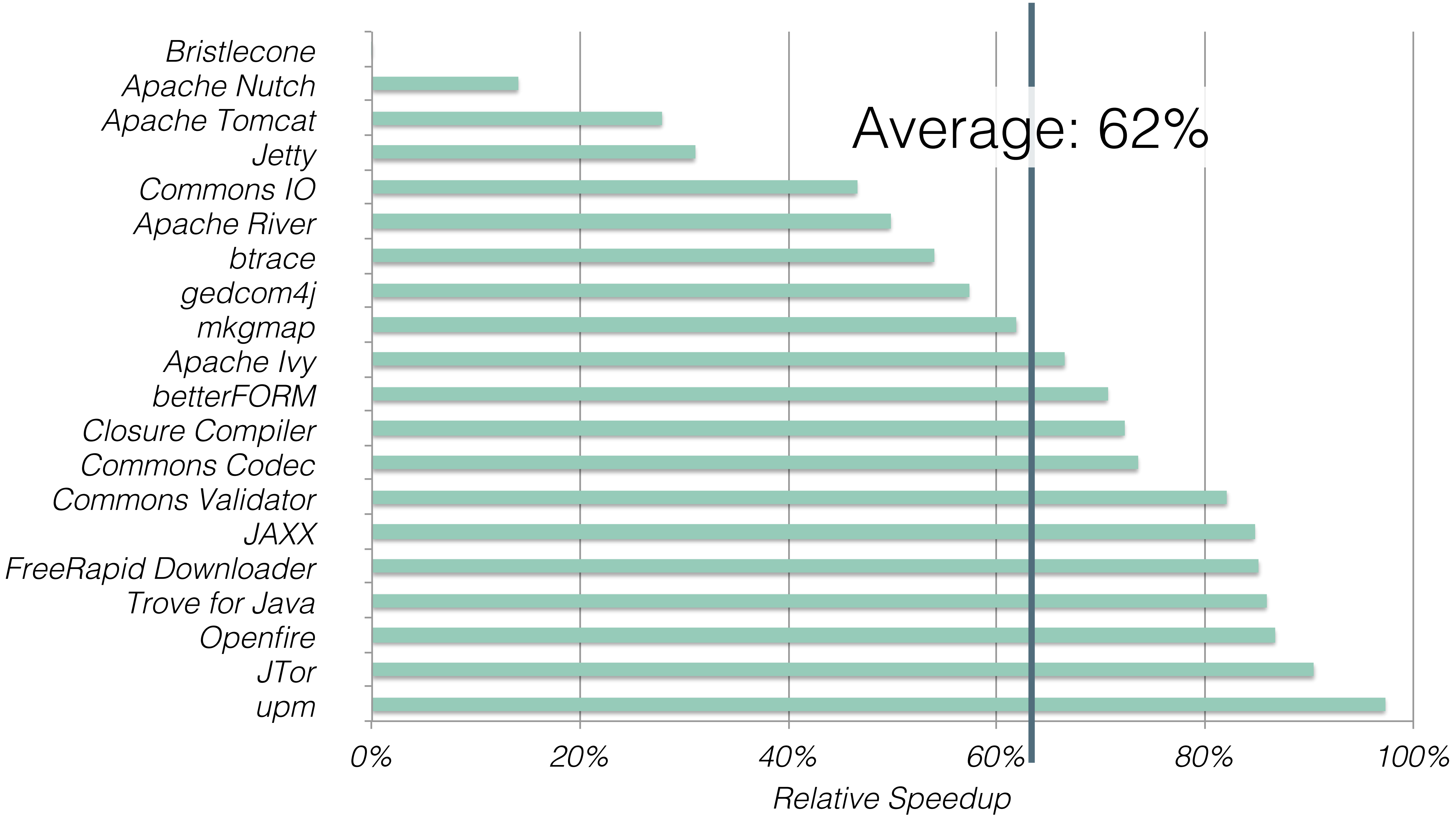
- Previous study: well-studied suite of 4 projects, which average 37,000 LoC and 51 test classes
- This study: manually collected repository of 20 projects, average 475,000 LoC and 56 test classes
  - Range from 5,000 LoC - 5,692,450 LoC; 3 - 292 test classes; 3.5-15 years in age

RQ2: Broader Evaluation



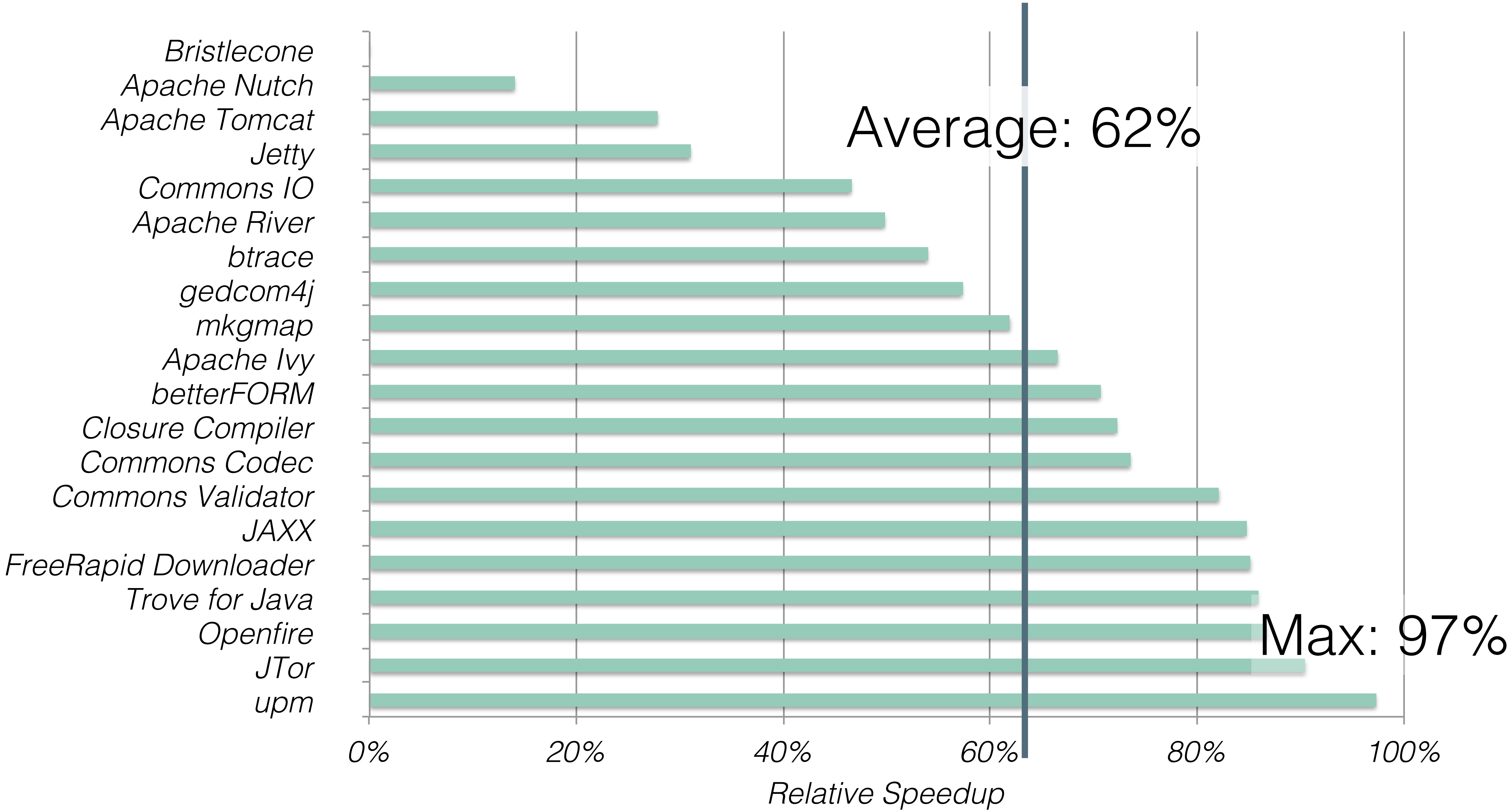
Larger is better

RQ2: Broader Evaluation



Larger is better

RQ2: Broader Evaluation



Larger is better

Factors that impact reduction

- Looked for relationships between number of tests, lines of code, age of project, total testing time, time per test, and VMVM's speedup

Factors that impact reduction

- Looked for relationships between number of tests, lines of code, age of project, total testing time, time per test, and VMVM's speedup
- Result: Only average time per test is correlated with VMVM's speedup (in fact, quite strongly; $p < 0.0001$)

RQ3: Impact on Fault Finding

- **No impact on fault finding** from seeded faults (SIR)

RQ3: Impact on Fault Finding

- **No impact on fault finding** from seeded faults (SIR)
- Does VMVM correctly isolate tests though?

RQ3: Impact on Fault Finding

- **No impact on fault finding** from seeded faults (SIR)
- Does VMVM correctly isolate tests though?
- Compared false positives and negatives between un-isolated execution, traditionally isolated execution, and VMVM-isolated execution for these 20 complex applications

RQ3: Impact on Fault Finding

- **No impact on fault finding** from seeded faults (SIR)
- Does VMVM correctly isolate tests though?
- Compared false positives and negatives between un-isolated execution, traditionally isolated execution, and VMVM-isolated execution for these 20 complex applications
- Result: **False positives occur when not isolated. VMVM shows no false positives or false negatives.**

Conclusions

- Most large applications isolate their test cases

Conclusions

- Most large applications isolate their test cases
- VMVM provides up to a 97% reduction in testing time through more efficient isolation (average 62%)

Conclusions

- Most large applications isolate their test cases
- VMVM provides up to a 97% reduction in testing time through more efficient isolation (average 62%)
- VMVM does not risk a reduction in fault finding

Fork me on Github

Unit Test Virtualization with VMVM

Jonathan Bell and Gail Kaiser
Columbia University

https://github.com/Programming-Systems-Lab/vmvm

See a demo of VMVM at 2:30 today! Room MR G1-3

ICSE 2014 @_jon_bell_ June 5, 2014