SpeedGun: Performance Regression Testing of Concurrent Classes

Michael Pradel[1], Markus Huggler[2], and Thomas R. Gross[2]

[1] University of California, Berkeley

[2] ETH Zurich

Motivation

Writing concurrent software is difficult

Correctness:
Synchronize concurrent accesses to shared data

Performance:
Avoid unnecessary synchroniza-tion

Motivation

Writing concurrent software is difficult

Correctness:
Synchronize concurrent accesses to shared data

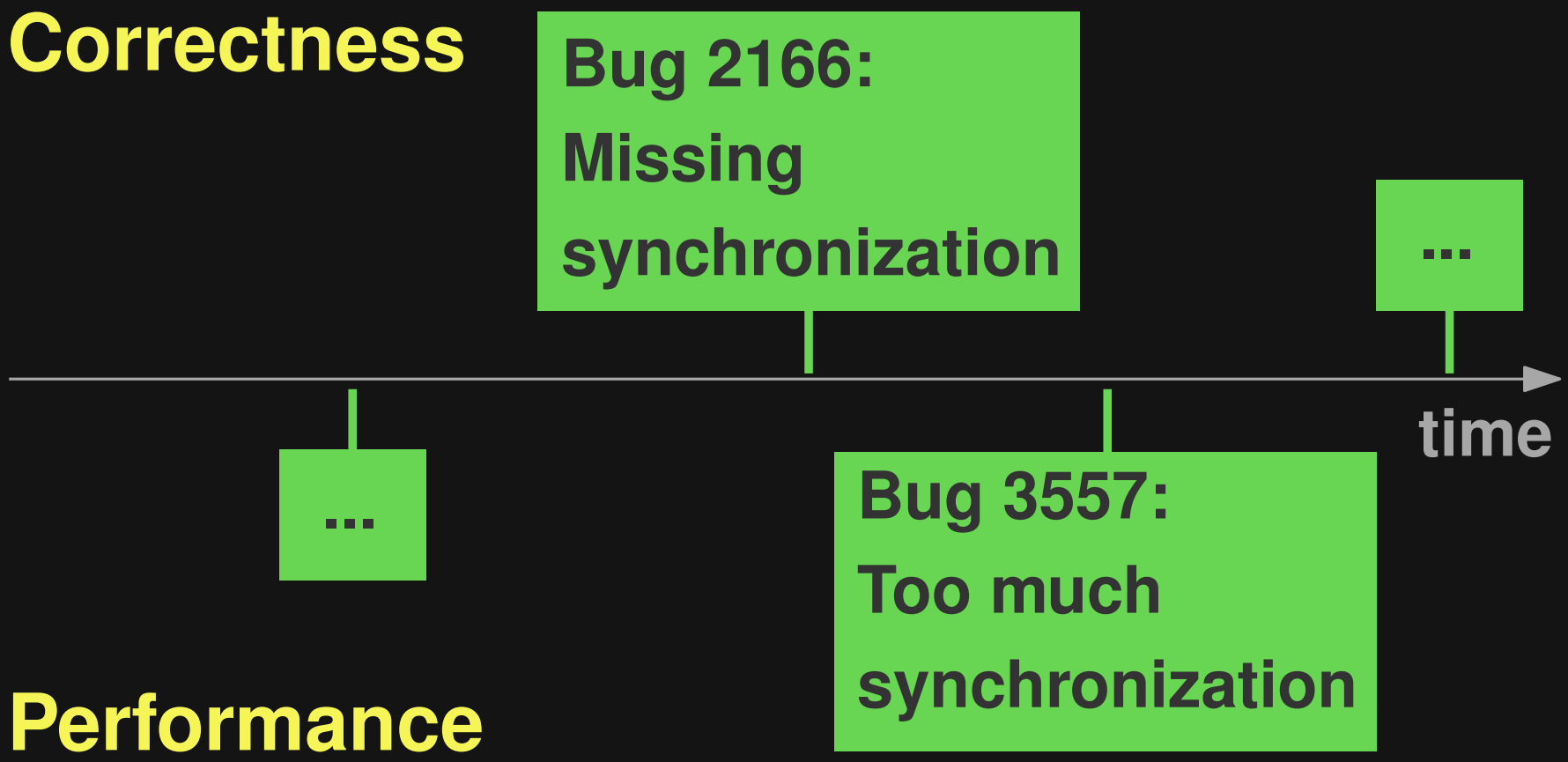
Performance:
Avoid unnecessary synchroniza-tion

Data races
Atomicity violations
Thread safety
Schedule exploration

?

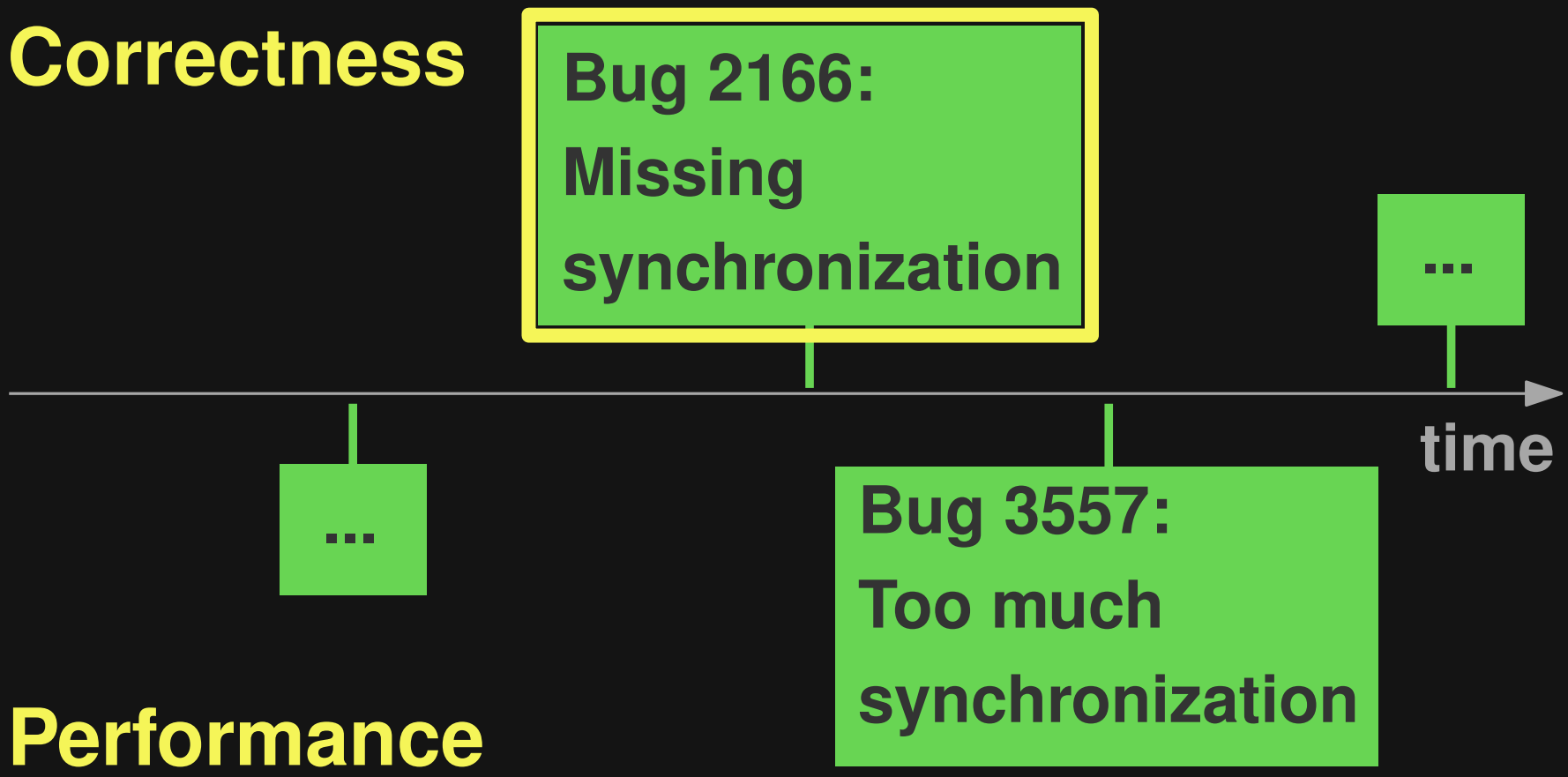
Real-World Example

History of Groovy's `ExpandoMetaClass`



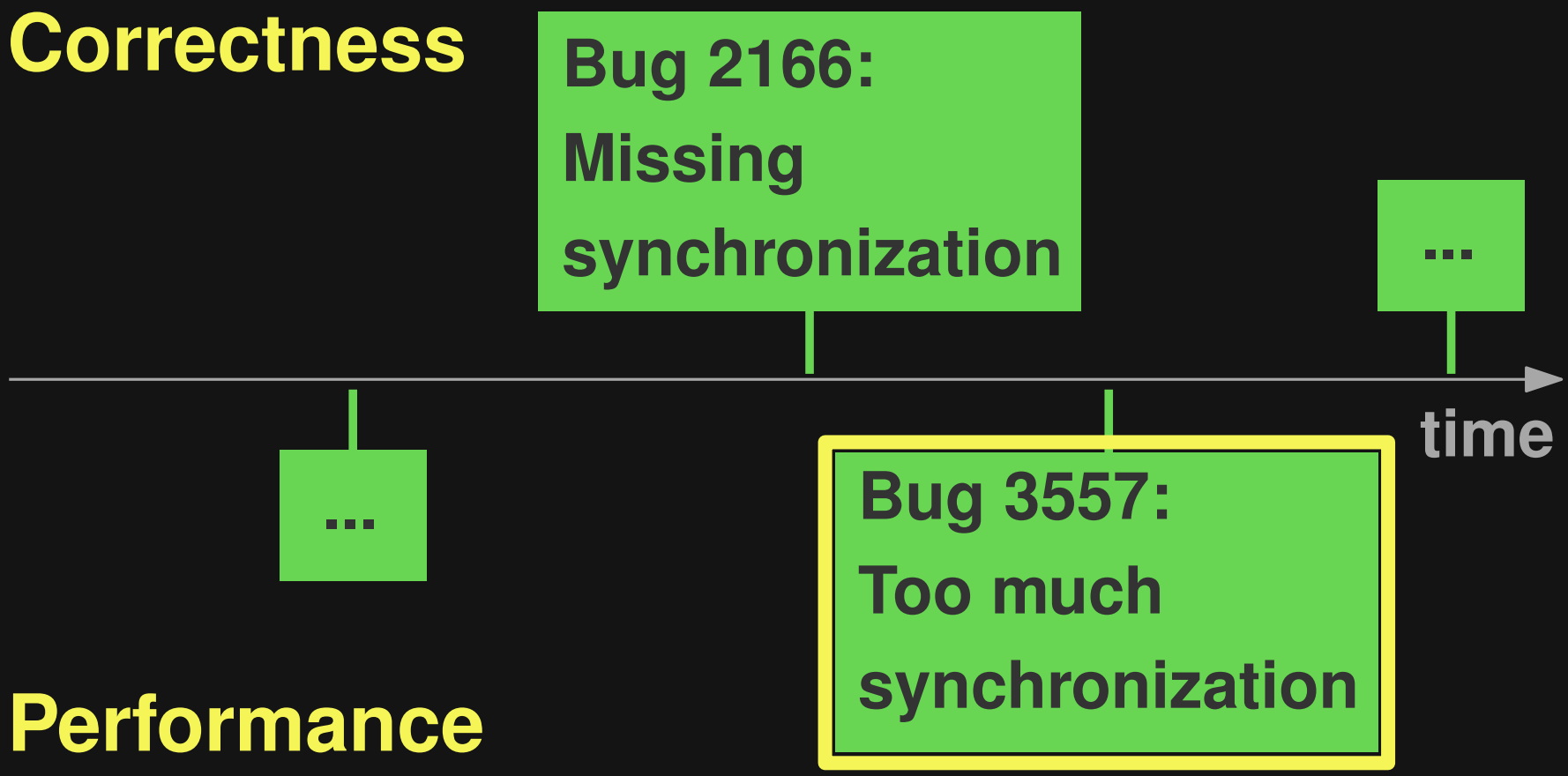
Real-World Example

History of Groovy's `ExpandoMetaClass`



Real-World Example

History of Groovy's `ExpandoMetaClass`



Real-World Example (2)

```
class ExpandoMetaClass {
  private boolean initialized;
  synchronized void initialize() {
    if (!this.initialized) {
      this.initialized = true;
    }
  }
  boolean isInitialized() {
    return this.initialized;
  }


}
```

Before bug 2166

Real-World Example (2)

```
class ExpandoMetaClass {
  private boolean initialized;
  synchronized void initialize() {
    if (!isInitialized()) {
      setInitialized(true);
    }
  }
  synchronized boolean isInitialized() {
    return this.initialized;
  }
  synchronized void setInitialized
      (boolean b) {
    this.initialized = b;
  }
}
```

Fix for bug 2166

Real-World Example (2)

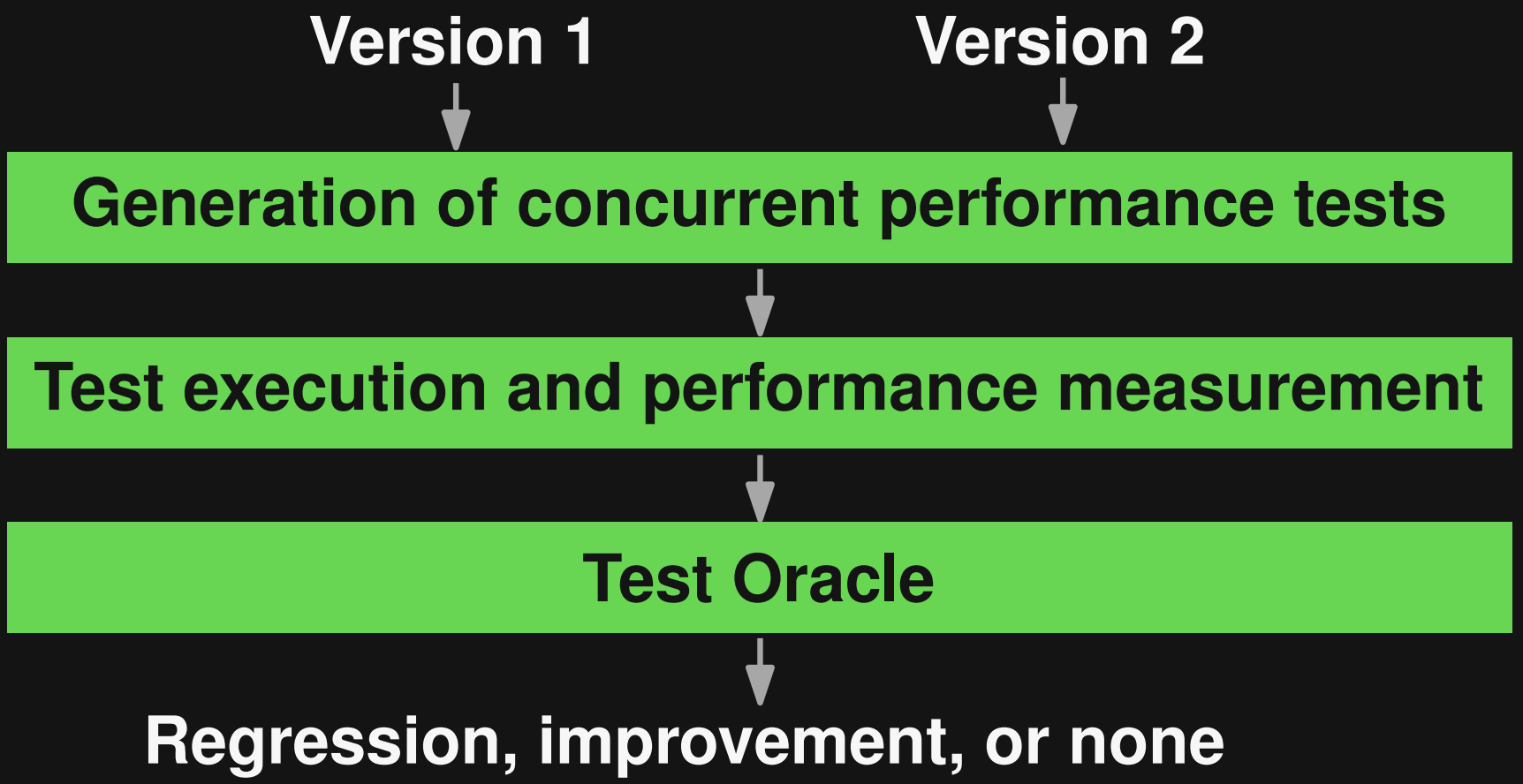
```
class ExpandoMetaClass {
  private volatile boolean initialized;
  synchronized void initialize() {
    if (!isInitialized()) {
      setInitialized(true);
    }
  }
  boolean isInitialized() {
    return this.initialized;
  }
  void setInitialized(boolean b) {
    this.initialized = b;
  }

}
```

Fix for bug 3557

SpeedGun: Overview

Automated performance regression testing for thread-safe classes



Challenges

Measuring performance ain’t easy

Challenges

Measuring performance of concurrent software ain't easy at all

Challenges

Measuring performance of concurrent software ain't easy at all

- Measurement accuracy
  - → Minimum measurable timespan

Challenges

Measuring performance of concurrent software ain't easy at all

- Measurement accuracy
  → Minimum measurable timespan
- Thread scheduling
  → Repeated execution

Challenges

Measuring performance of concurrent software ain't easy at all

- Measurement accuracy
  → Minimum measurable timespan
- Thread scheduling
  → Repeated execution
- Just-in-time compilation
  → Warm up + steady state

Challenges

Measuring performance of concurrent software ain't easy at all

- Measurement accuracy
  → Minimum measurable timespan
- Thread scheduling
  → Repeated execution
- Just-in-time compilation
  → Warm up + steady state
- Garbage collection
  → Invoke before measurements

Concurrent Tests: Example

Sequential prefix + concurrent suffixes

```
ExpandoMetaClassInit v0 = new ExpandoMetaClassInit();
ExpandoMetaClass v1 = v0.unInitalizedExpandoMetaClass();
Class v2 = v1.getJavaClass();
ExpandoMetaClass x = new ExpandoMetaClass(v2, true);
x.getExpandoMethods();
```


```
String v4 = x.toString();
x.respondsTo(v4, v4, null);
x.isModified();
...
```

```
x.initialize();
x.getClassNode();
x.getProperties();
...
```

Test Generation

Feedback-directed random generation of **concurrent** tests [PLDI'12]

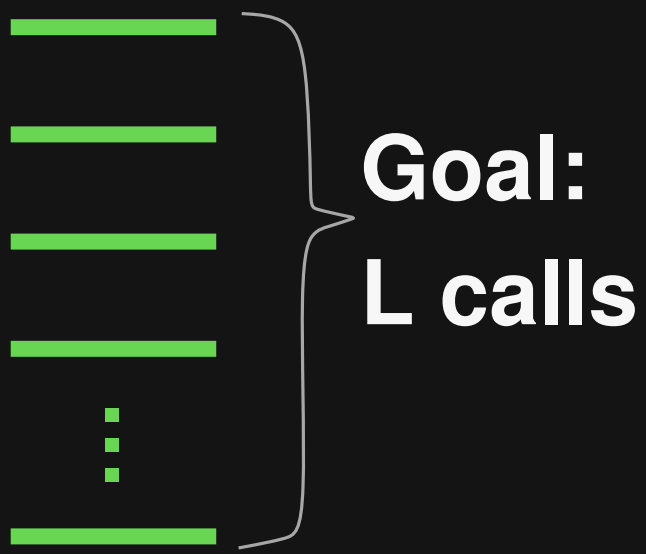
Here: Long tests with many suffixes

Exceed minimum measurable timespan

High degree of concurrency

Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests



Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Test Generation: Scalability

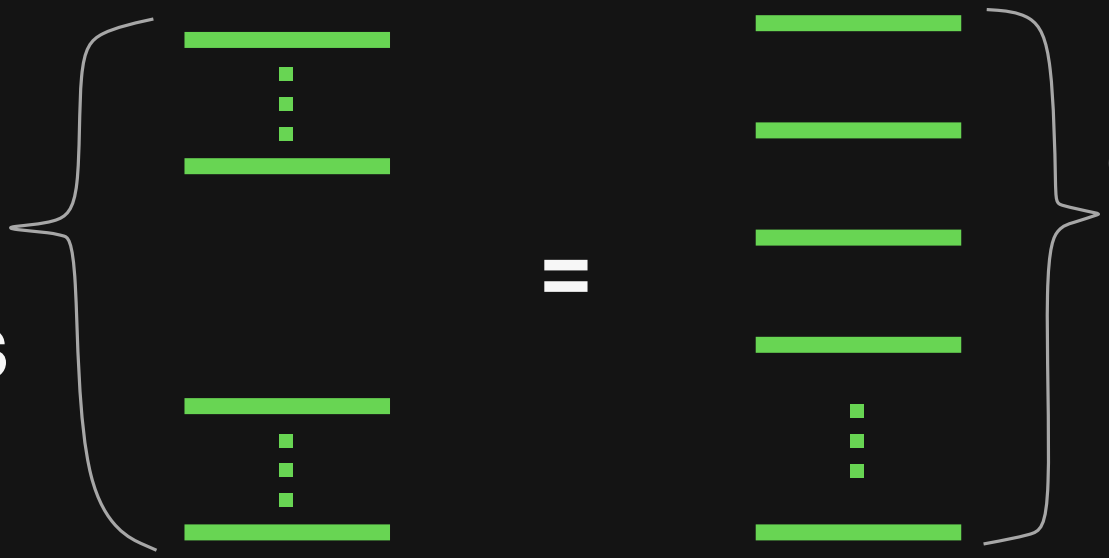
Challenge: Scaling feedback-directed test generation to large tests

Execute at least $1 + .. + L = \frac{L^2 + L}{2}$ **calls**

Test Generation: Scalability

Approach: Generate smaller sequences and repeat them

Repeat $\sqrt{L}$ calls $\sqrt{L}$ times

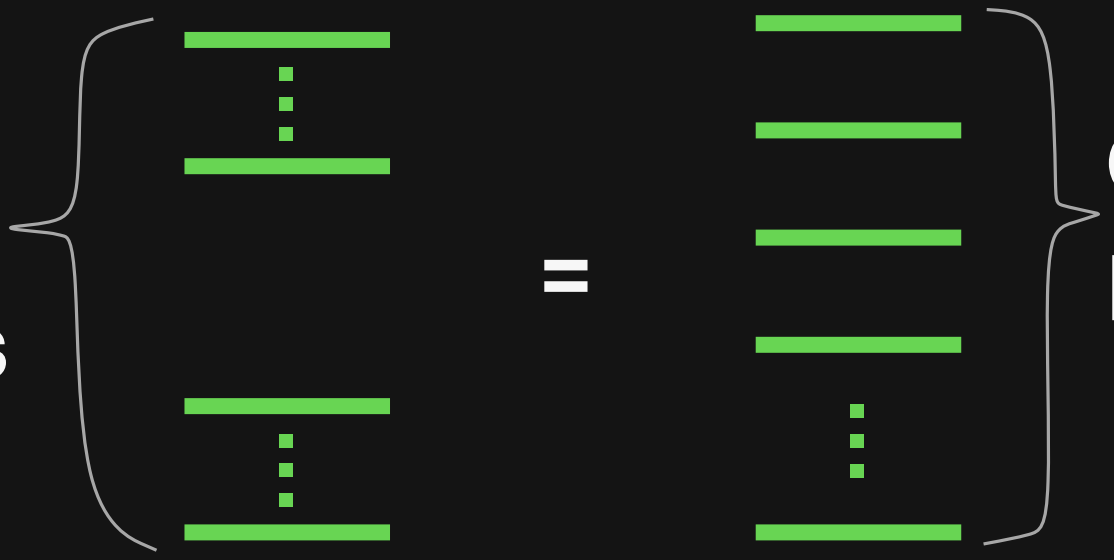


Goal: L calls

Test Generation: Scalability

Approach: Generate smaller sequences and repeat them

Repeat $\sqrt{L}$ calls $\sqrt{L}$ times



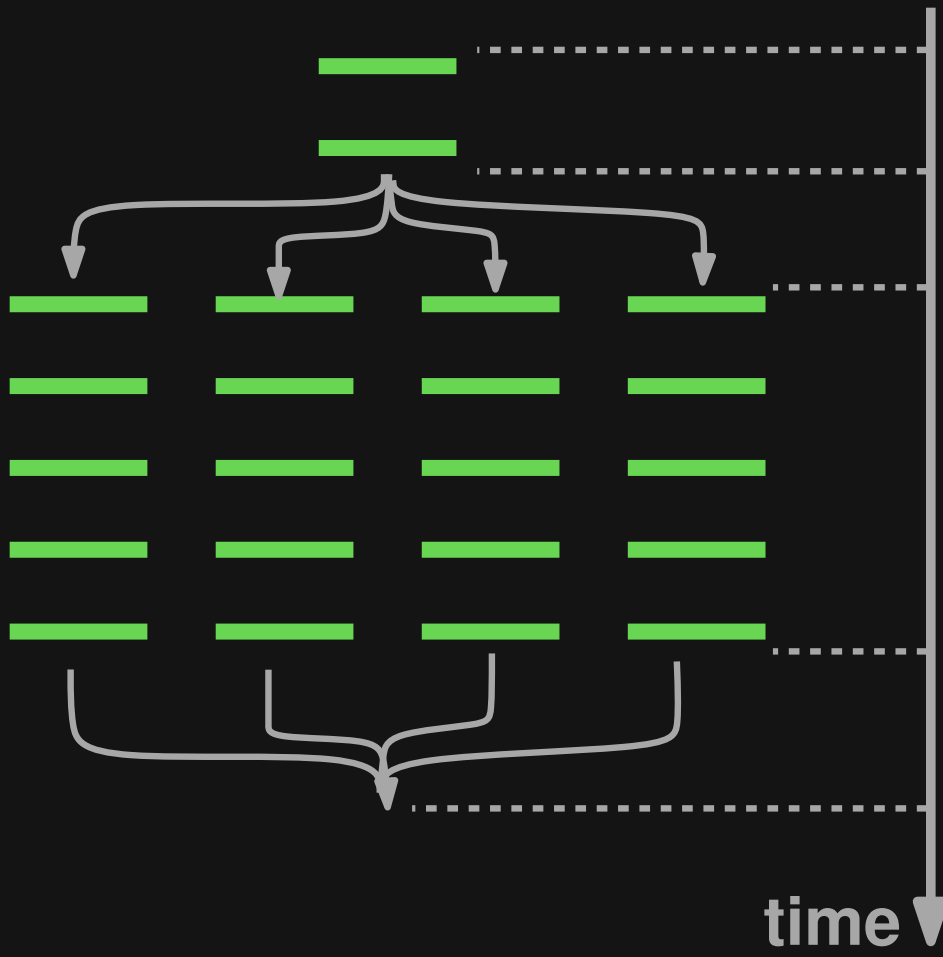
=

Goal: L calls

Execute at least $1 + .. + \sqrt{L} = \frac{L + \sqrt{L}}{2}$ calls

Test Execution (Single)

How to measure test execution time?



Prefix

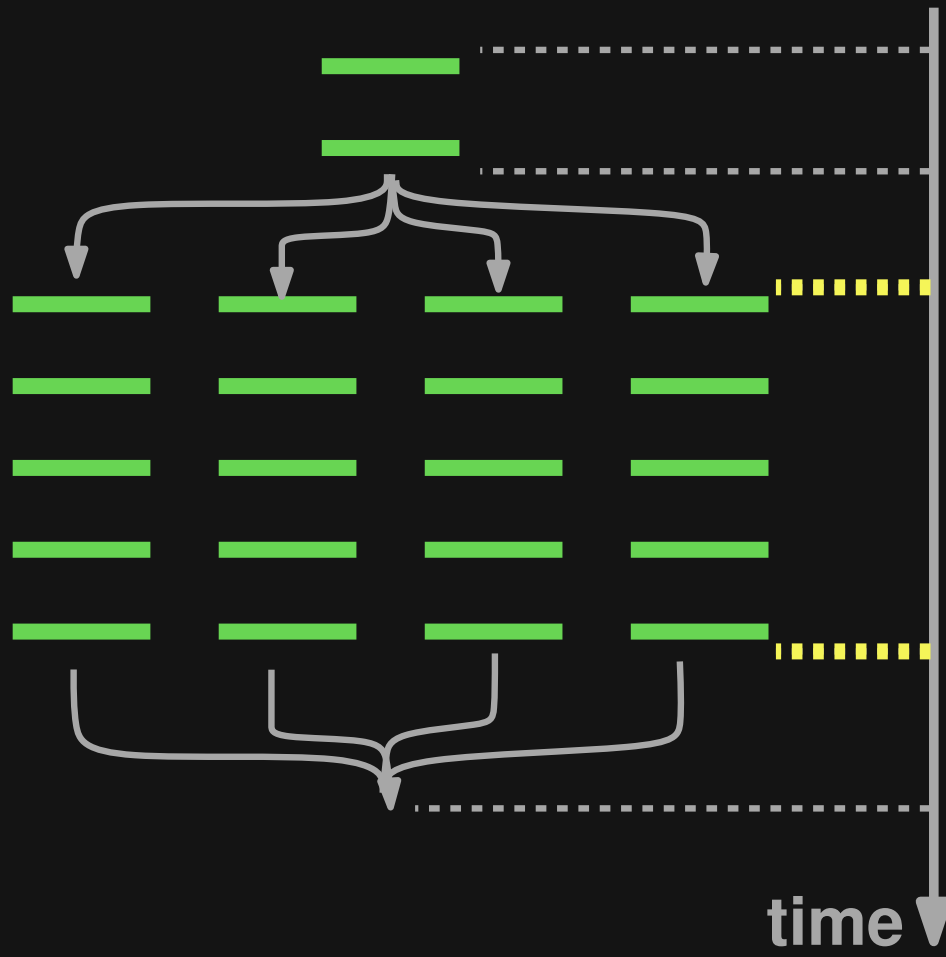
Thread setup

Suffixes

Thread cleanup

Test Execution (Single)

How to measure test execution time?



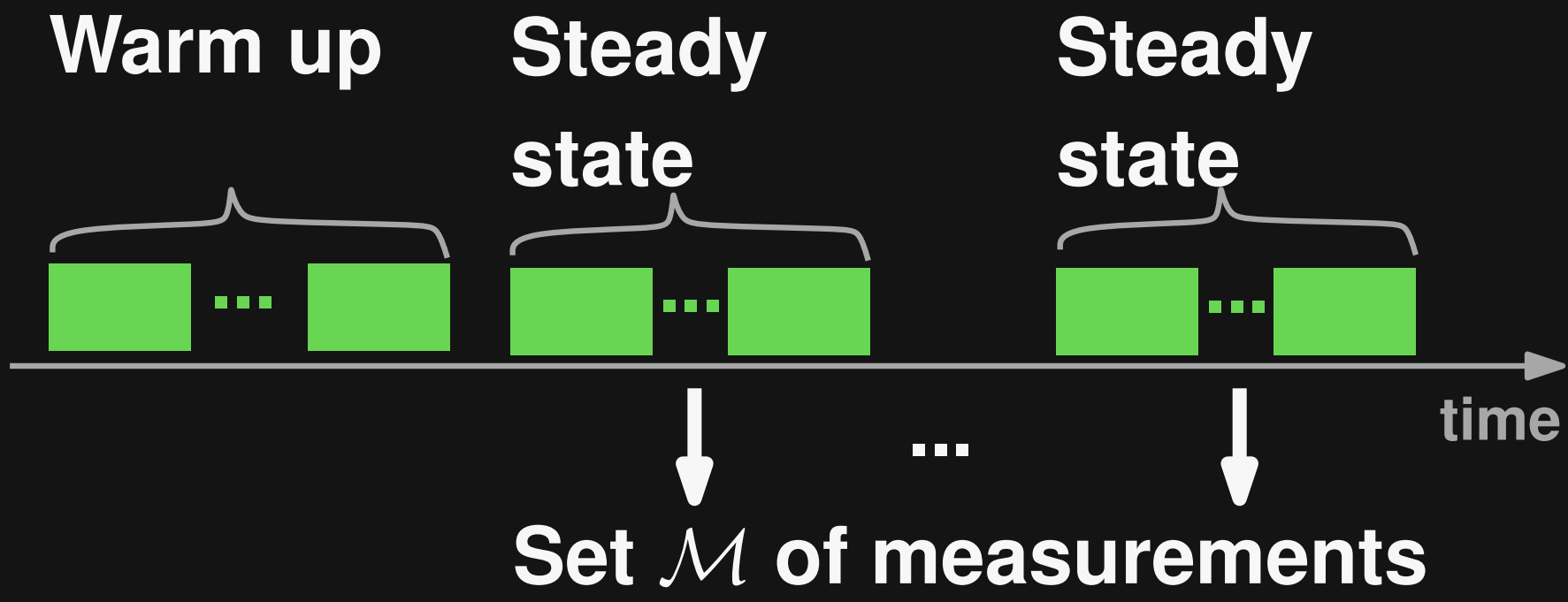
Prefix

Thread setup

Suffixes =
Measured time

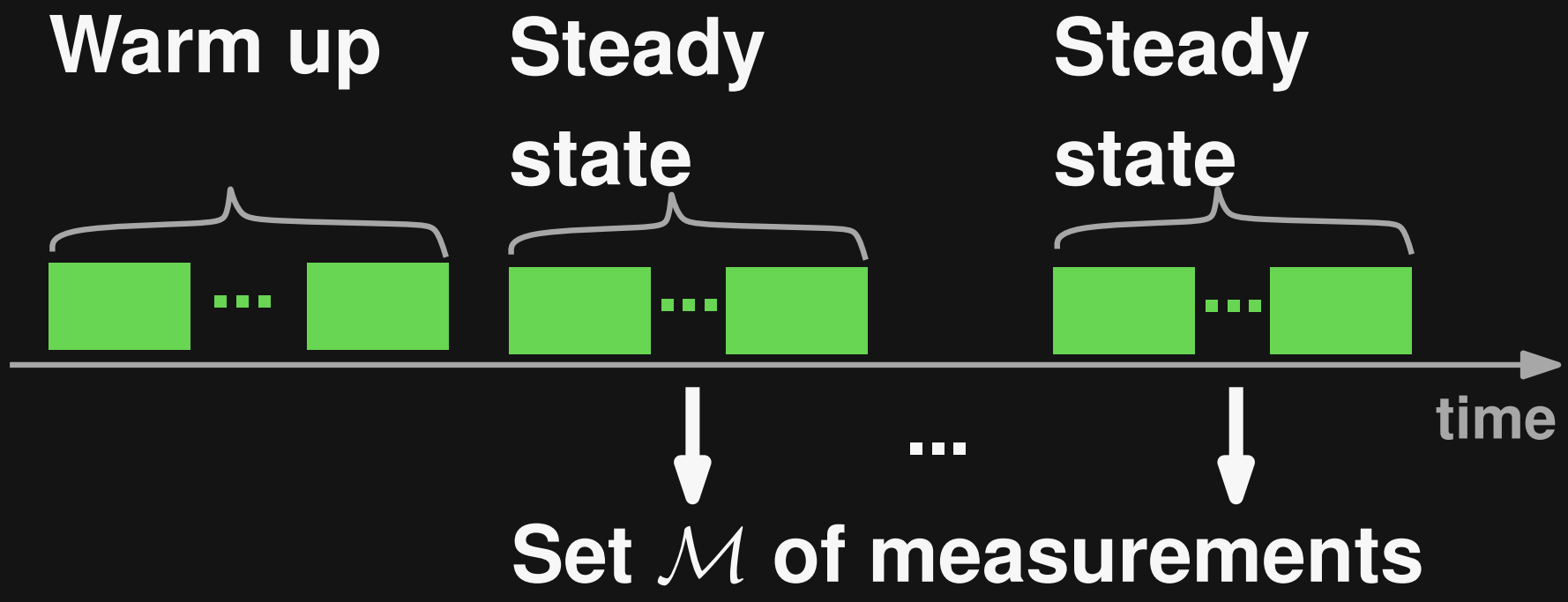
Thread cleanup

Test Execution (Repeated)



= test execution

Test Execution (Repeated)



Add measurements until variance is within fixed bounds: $\sigma(\mathcal{M}) \leq \overline{\mathcal{M}} \cdot \beta$

- $\sigma(\mathcal{M})$: standard deviation
- $\overline{\mathcal{M}}$: mean
- β: default: 0.01

Length of Tests

How long should tests be?

Length of Tests

How long should tests be?

Average execution time of test T: t_T

Repetitions: r_T

Constraints:

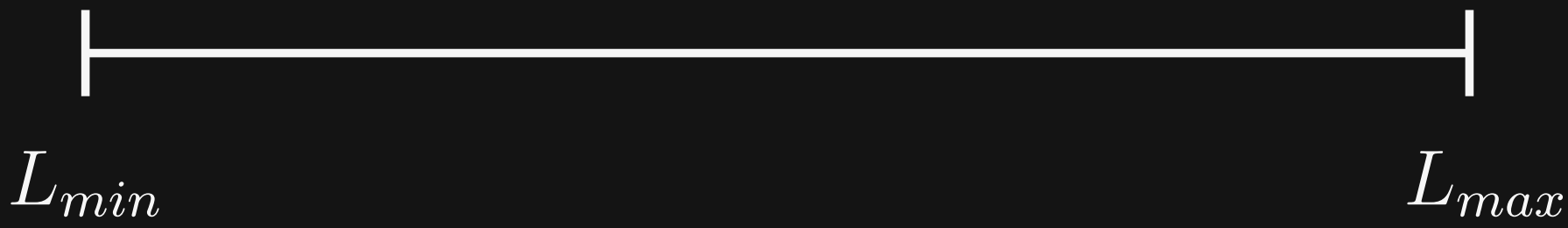
- $t_T >$ minimum measurable timespan
- $r_T >$ minimum number of repetitions

Length of Tests (2)

Approach: Binary search

Repeat until constraints fulfilled:

- **Generate a test T**
- **Execute and measure t_T and r_T for both versions**

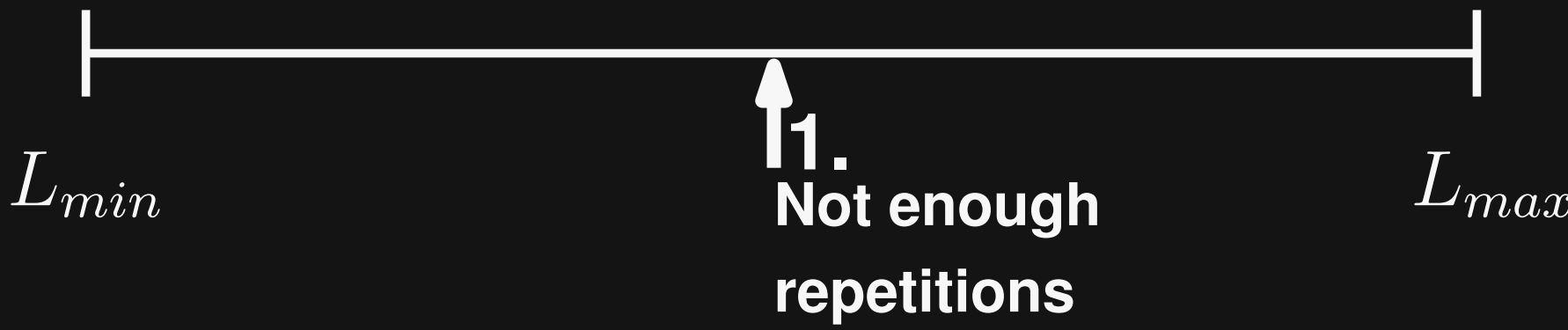


Length of Tests (2)

Approach: Binary search

Repeat until constraints fulfilled:

- **Generate a test T**
- **Execute and measure t_T and r_T for both versions**

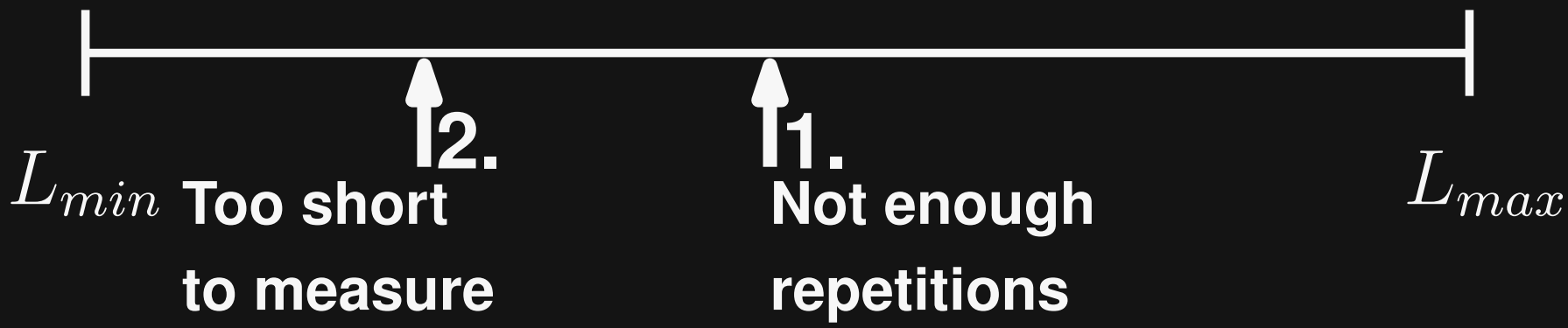


Length of Tests (2)

Approach: Binary search

Repeat until constraints fulfilled:

- **Generate a test T**
- **Execute and measure t_T and r_T for both versions**



Length of Tests (2)

Approach: Binary search

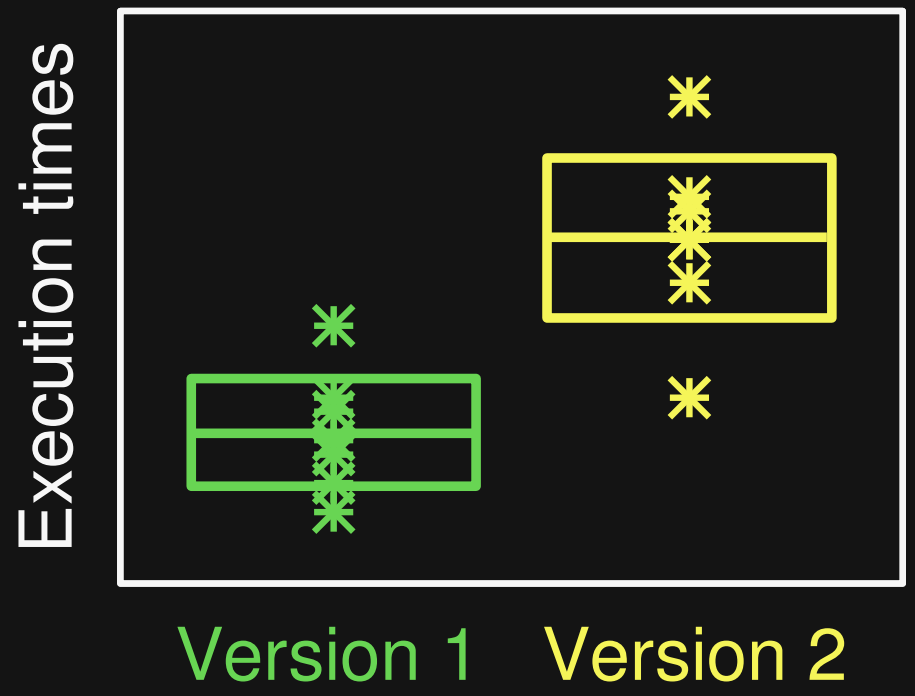
Repeat until constraints fulfilled:

- **Generate a test T**
- **Execute and measure t_T and r_T for both versions**

Test Oracle

Does one version outperform the other?

1) Decide winner of each test

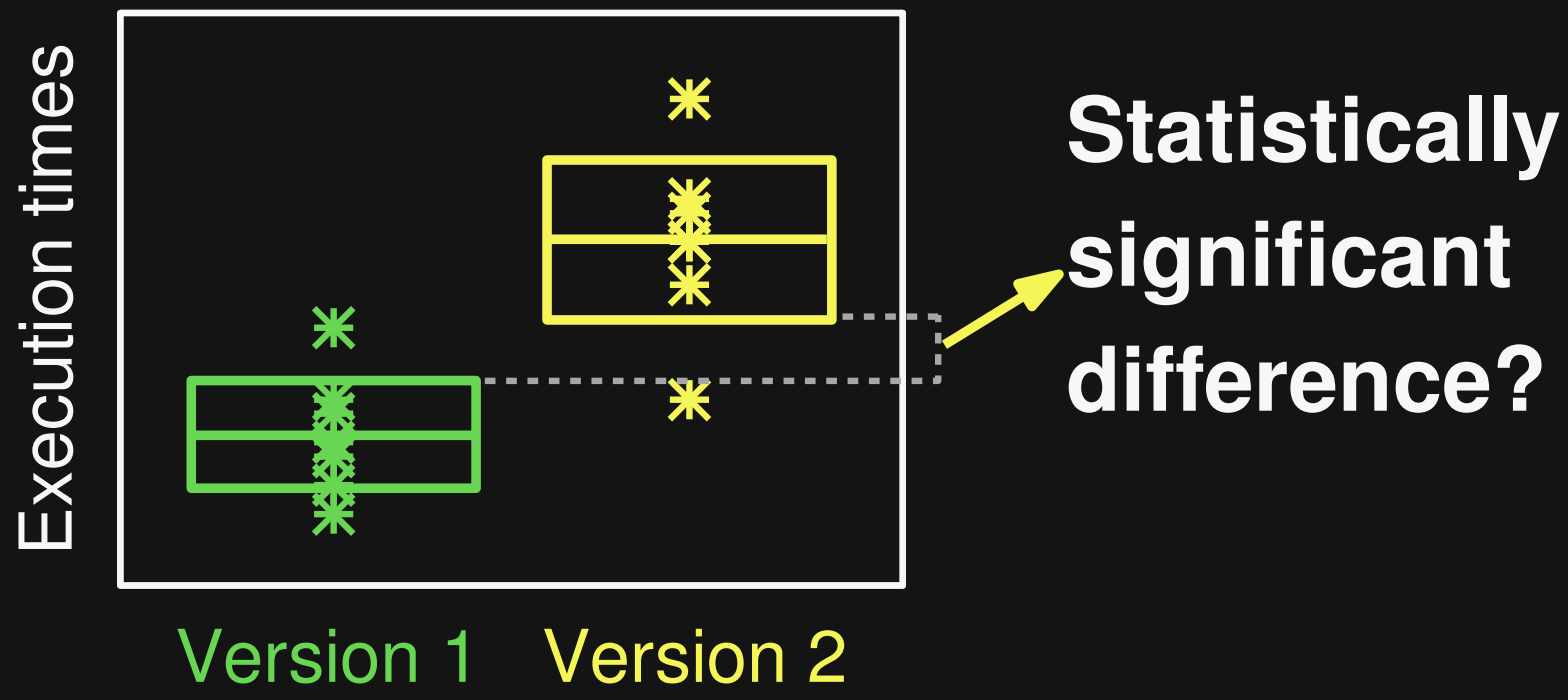


.. mean and confidence interval

Test Oracle

Does one version outperform the other?

1) Decide winner of each test

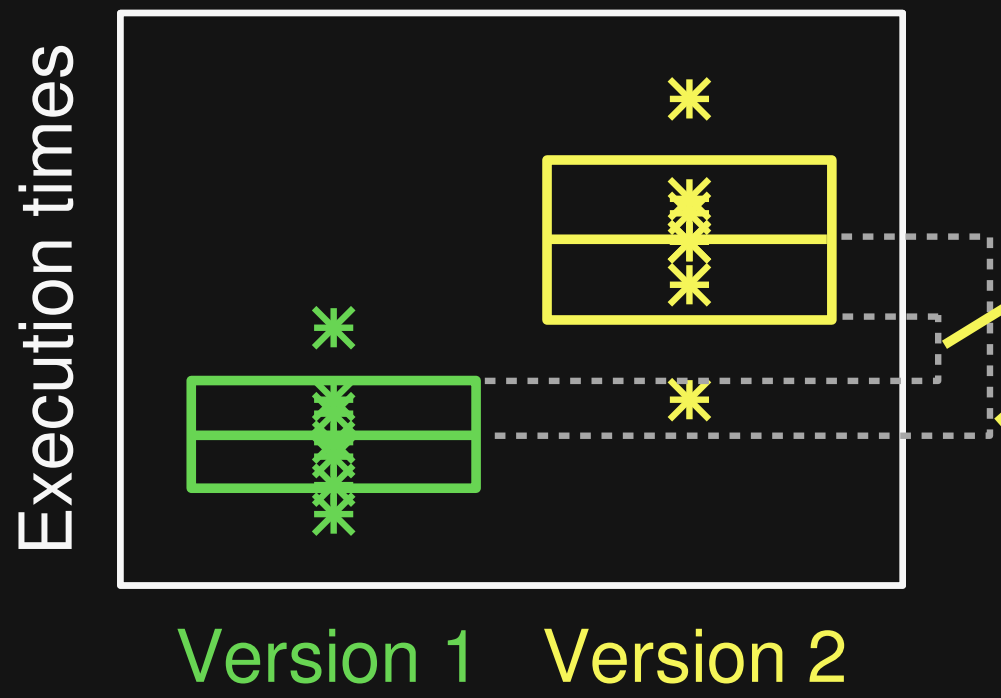


.. mean and confidence interval

Test Oracle

Does one version outperform the other?

1) Decide winner of each test



Statistically significant difference?

Difference above threshold?

.. **mean and confidence interval**

Test Oracle

Does one version outperform the other?

2) Decide overall winner

Test Oracle

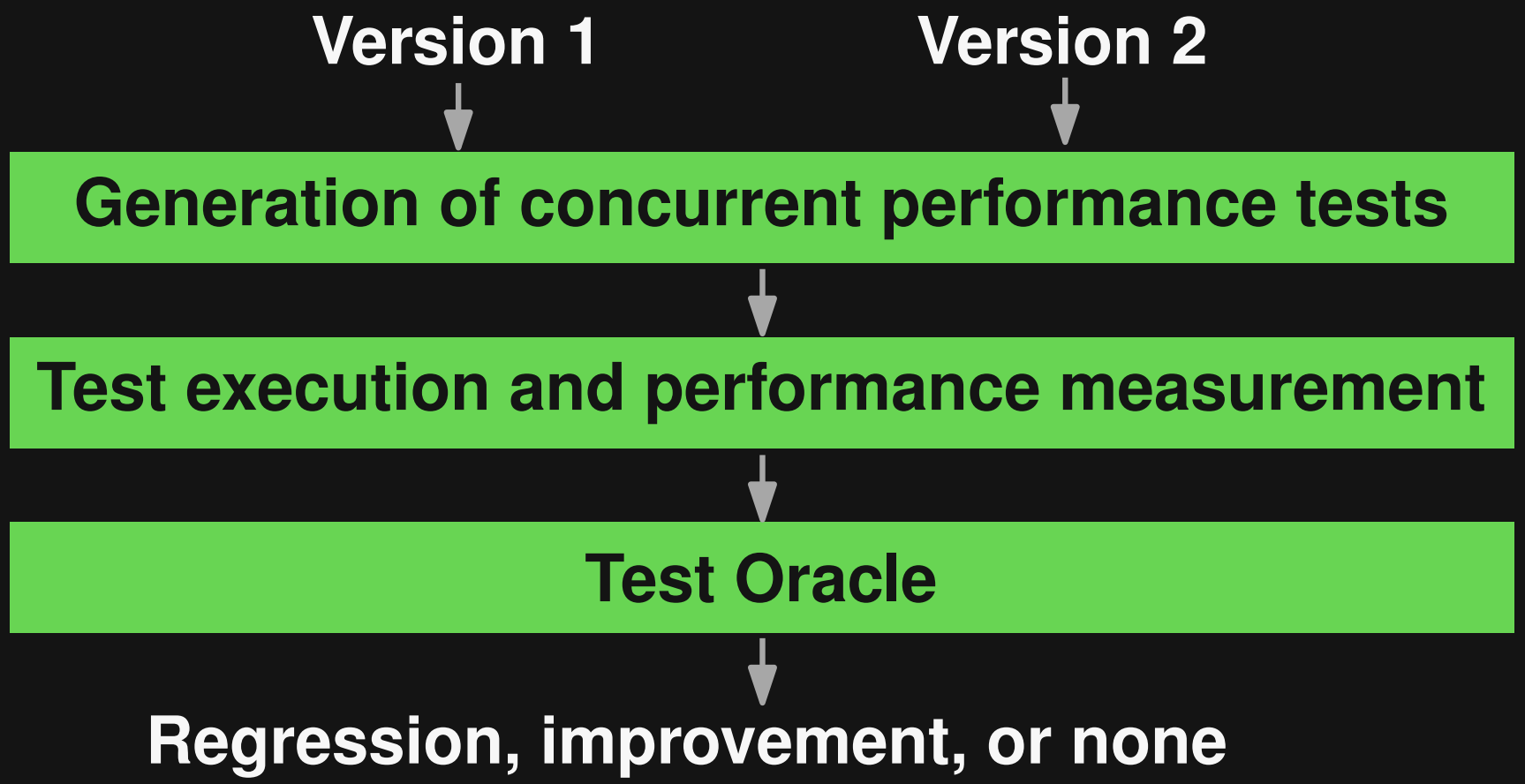
Does one version outperform the other?

2) Decide overall winner

- **Group tests by winner:**
  $\mathcal{T}_{V1}$, $\mathcal{T}_{V2}$, $\mathcal{T}_{None}$
- **Report regression if**
  $|\mathcal{T}_{V1}| > |\mathcal{T}_{V2}|$ **and** $|\mathcal{T}_{V1}| > |\mathcal{T}_{None}|$
- **Report improvement if**
  $|\mathcal{T}_{V2}| > |\mathcal{T}_{V1}|$ **and** $|\mathcal{T}_{V2}| > |\mathcal{T}_{None}|$

SpeedGun: Overview

Automated performance regression testing for thread-safe classes



Evaluation

Does SpeedGun identify performance regressions and improvements?

Setup:

- 5 classes from 4 projects
- Full version history of 3 classes
- 113 pairs of classes

Baseline:

- Comments from developers
- Manual inspection

Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

Unexpected difference is reported (2)

Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

Unexpected difference is reported (2)

Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

Unexpected difference is reported (2)

Results

113 pairs of classes

No warning (96)

Expected difference
is reported (11)

Expected difference
is not reported (4)

Unexpected difference
is reported (2)

Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

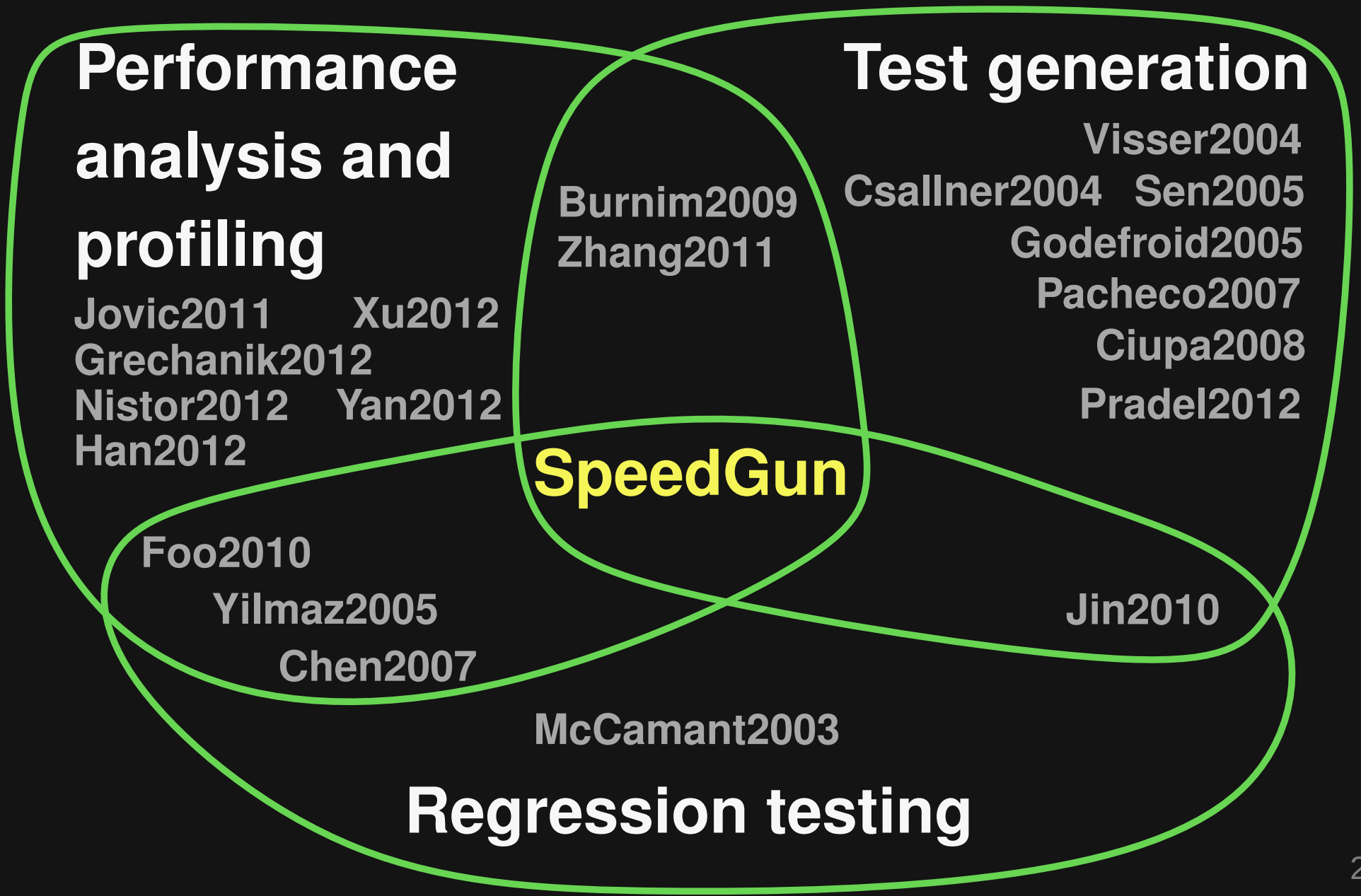
Unexpected difference is reported (2)

Examples

Program	Change		Speedup
Groovy	Synchronize methods	↘	0.92x
Groovy	Volatile instead of synchronized	↗	1.50x
Collections	Fix correctness bug by adding synchronization	↘	0.64x
Pool	Finer-grained locking to avoid deadlocks	↗	1.52x

Intel Core i7 CPU, 8 threads

Related Work



Conclusion

SpeedGun: Automated performance regression testing for thread-safe classes

- Generation of concurrent performance tests
- Systematically avoid pitfalls of measuring concurrent performance

A step towards reliable and efficient concurrent software

SpeedGun: Performance Regression Testing of Concurrent Classes

Michael Pradel[1], Markus Huggler[2], and Thomas R.Gross[2]

[1] ~~University of California, Berkeley~~

[2] ETH Zurich

I'm looking for students to join my group at TU Darmstadt!