Fully Automatic and Precise Detection of Thread Safety Violations

Michael Pradel and Thomas R. Gross

Department of Computer Science
ETH Zurich

thread-safe.org

Motivation

Thread-safe classes:

Building blocks for concurrent programs

Motivation

Thread-safe classes:

Building blocks for concurrent programs

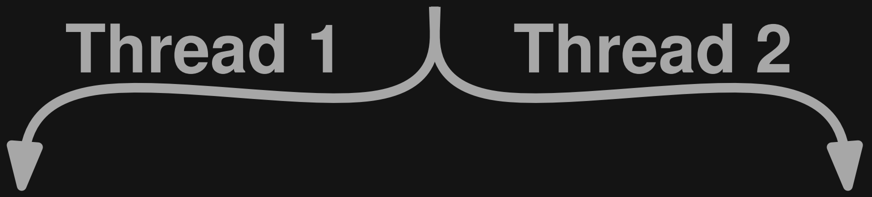
Motivation

Thread-safe classes:

Building blocks for concurrent programs

Example from JDK

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



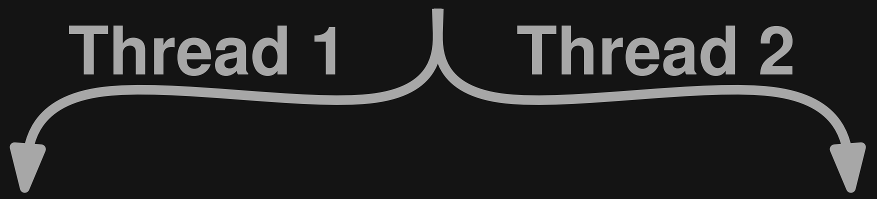
Thread 1

Thread 2

```
b.insert(1, b)        b.deleteCharAt(1)
```

Example from JDK

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



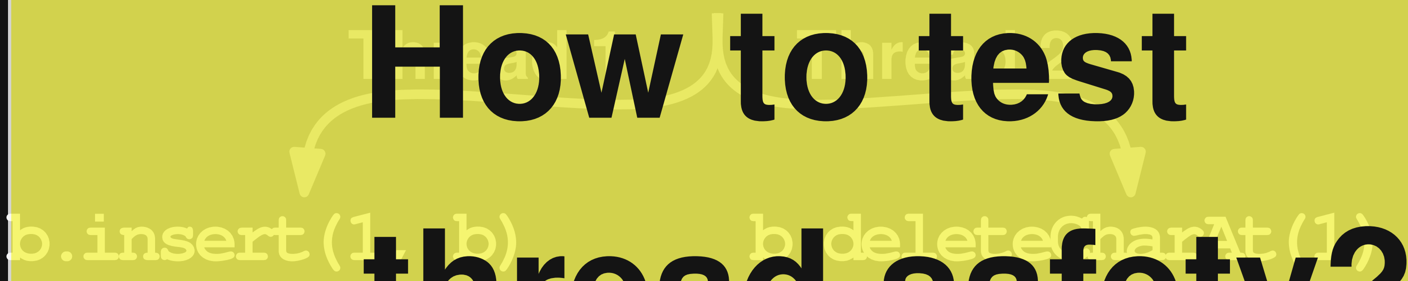
```
b.insert(1, b)        b.deleteCharAt(1)
```

IndexOutOfBoundsException

Confirmed as bug: Issue #7100996

Example from JDK

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



Thread 1

Thread 2

```
b.insert(1, b)
```

```
b.deleteCharAt(1)
```

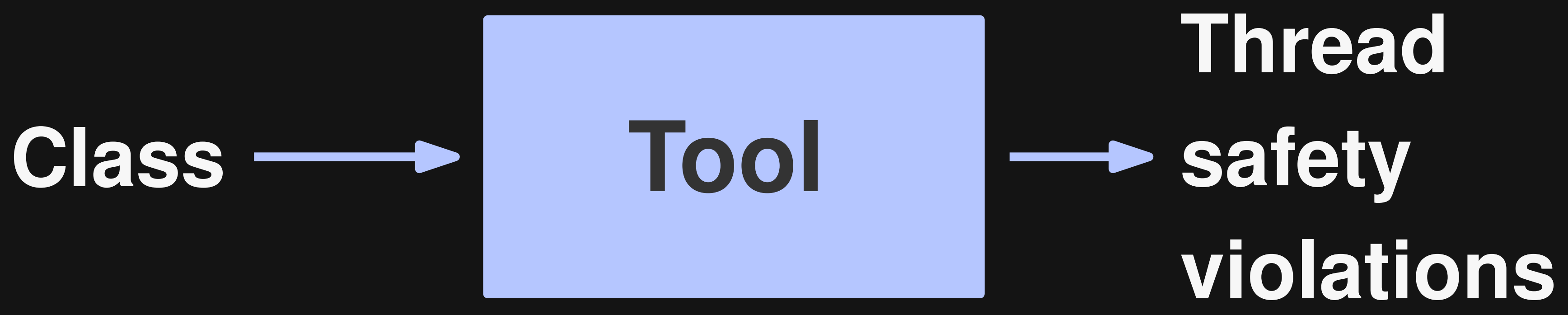
How to test thread safety?

IndexOutOfBoundsException

Confirmed as bug: Issue #7100996

Goal

Automatic and precise bug detection



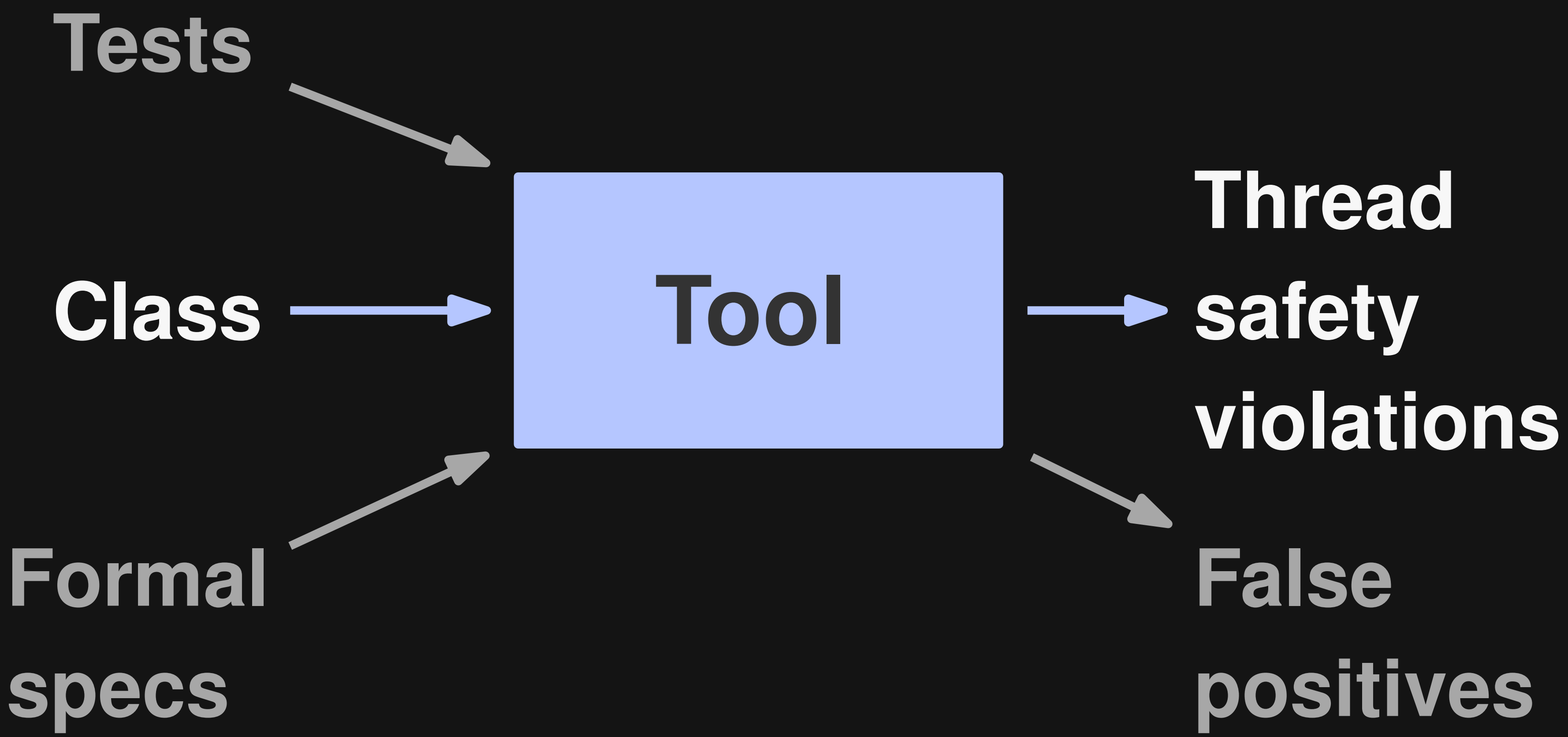
Class

Tool

Thread safety violations

Goal

~~Automatic~~ and ~~precise~~ bug detection



Tests

Class

Formal specs

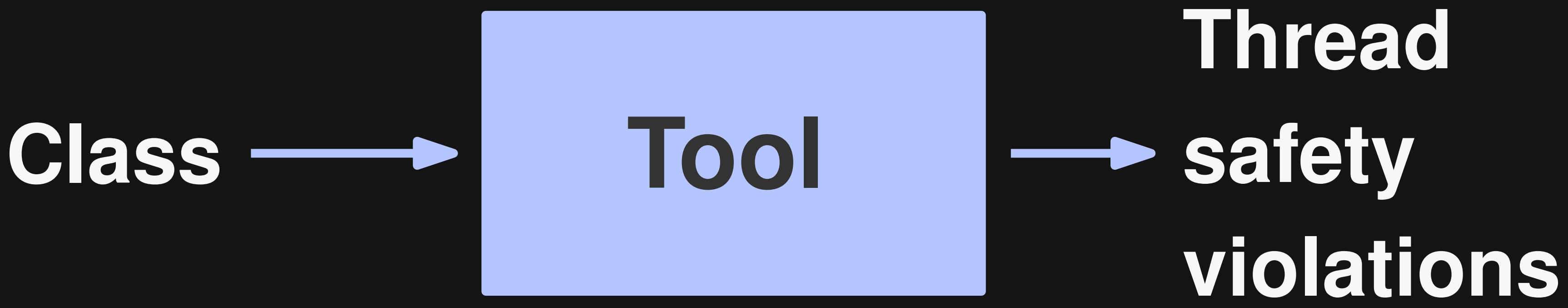
Tool

Thread safety violations

False positives

Goal

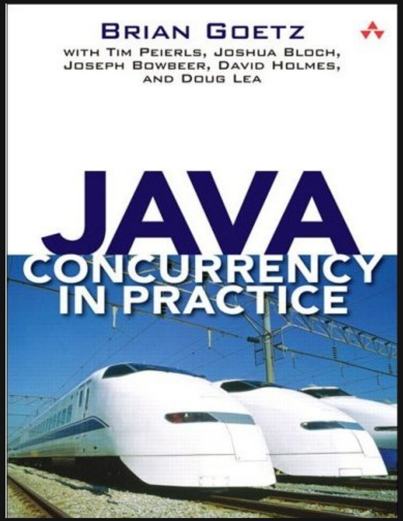
Automatic and precise bug detection



Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”

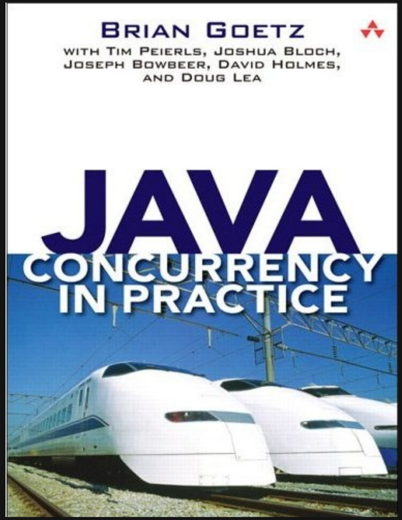
page 18


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY IN PRACTICE

Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”

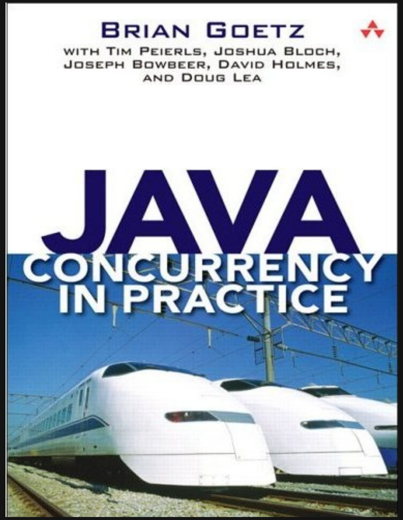
page 18


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH,
JOSEPH BOWBEER, DAVID HOLMES,
AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”

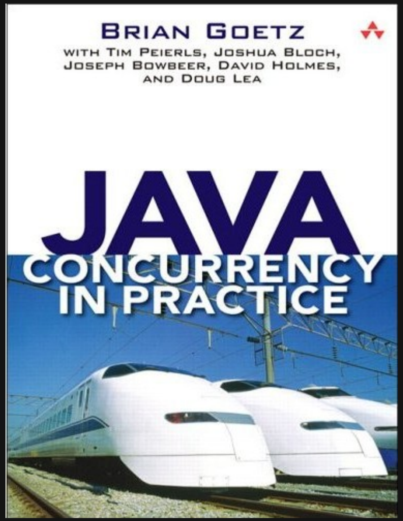
page 18


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY IN PRACTICE

Thread-Safe Classes

"behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code"

page 18


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY IN PRACTICE

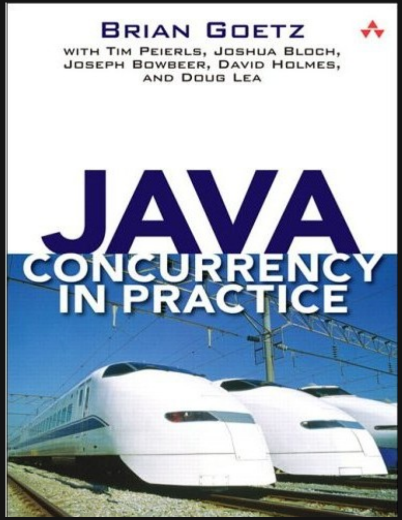
"operations ... behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads"

StringBuffer API documentation, JDK 6

Thread-Safe Classes

"behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code"

page 18

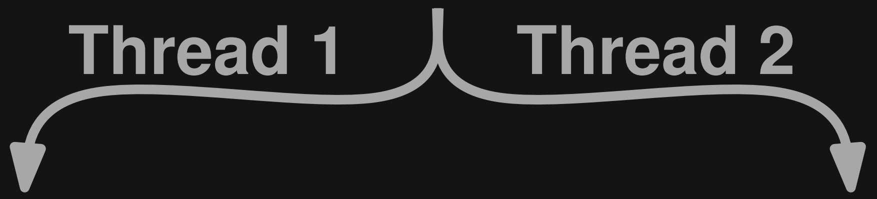

BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH,
JOSEPH BOWBEER, DAVID HOLMES,
AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

"operations ... behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads"

StringBuffer API documentation, JDK 6

Example

```
StringBuffer b = new StringBuffer()
```



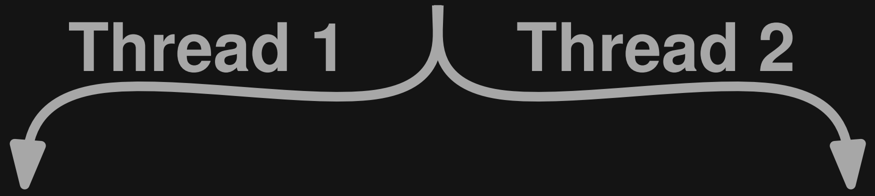
Thread 1

Thread 2

```
b.append("a")        b.append("c")
b.append("b")
```

Example

```
StringBuffer b = new StringBuffer()
```



Thread 1

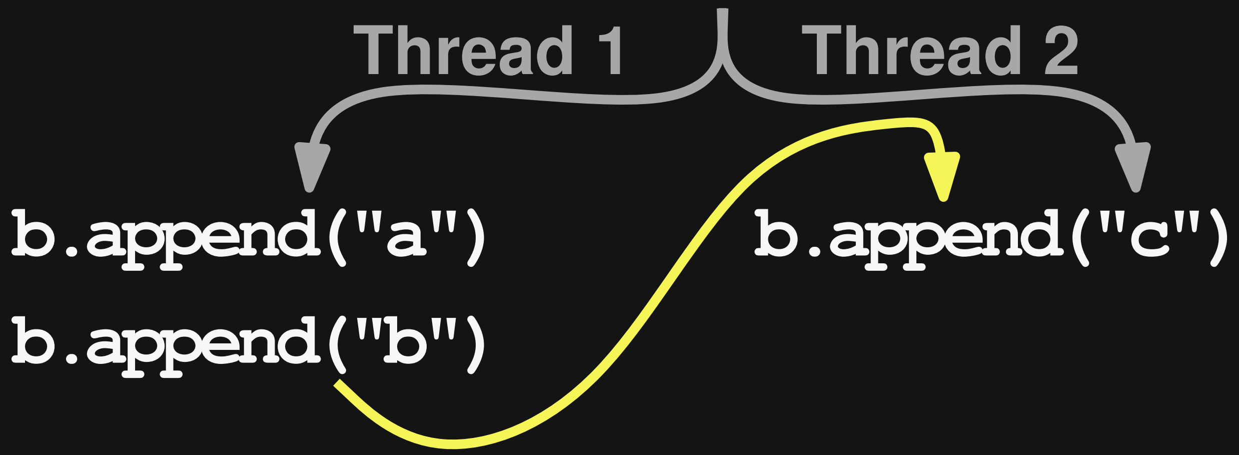
Thread 2

```
b.append("a")         b.append("c")
b.append("b")
```

"abc" ✓   "cab" ✓   "acb" ✓   "ac" ✗

Example

```
StringBuffer b = new StringBuffer()
```



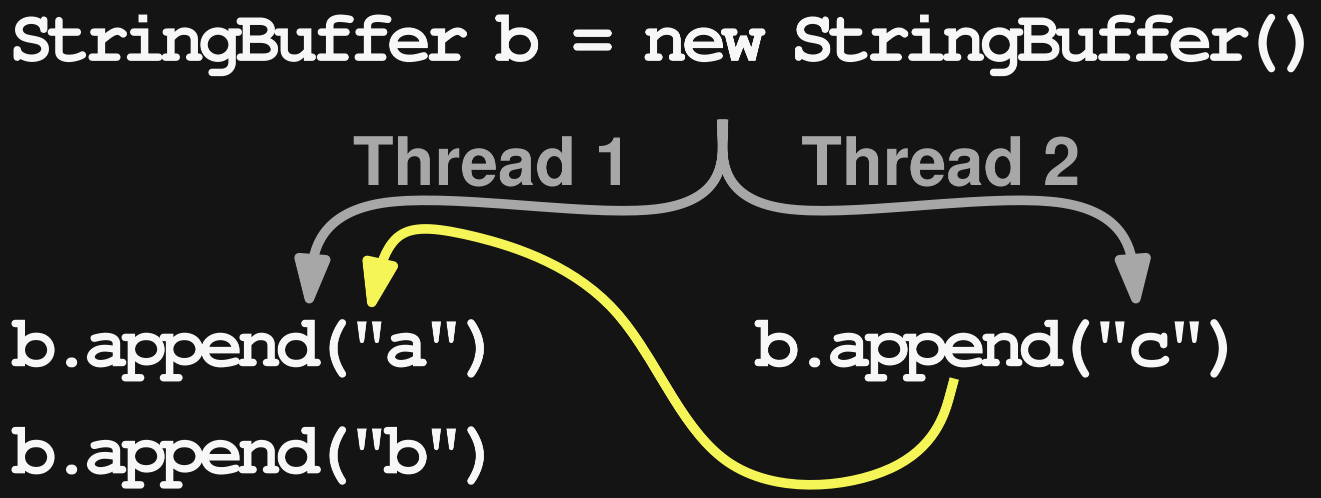
Thread 1

Thread 2

```
b.append("a")          b.append("c")
b.append("b")
```

"abc" ✓ "cab" ✓ "acb" ✓ "ac" ✗

Example



```
StringBuffer b = new StringBuffer()
```

```
b.append("a")                b.append("c")
b.append("b")
```

"abc" ✓

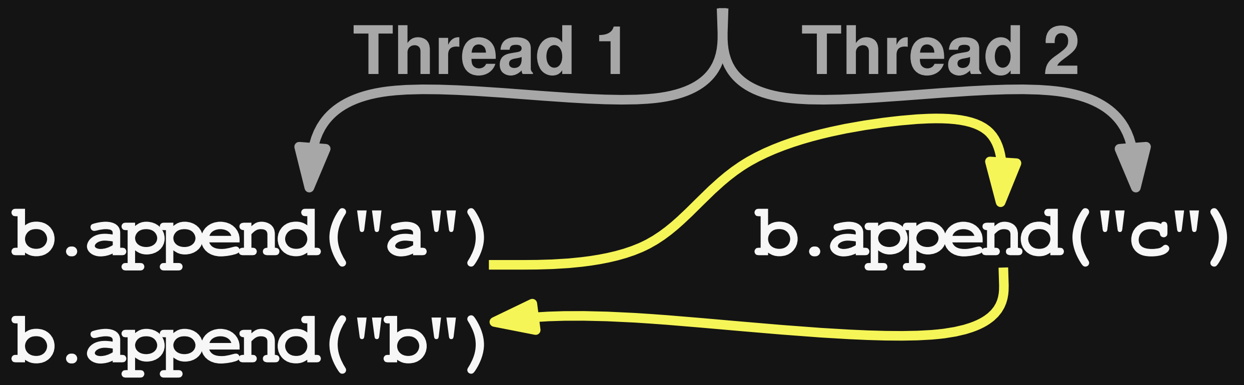
"cab" ✓

"acb" ✓

"ac" ✗

Example

```
StringBuffer b = new StringBuffer()
```



Thread 1

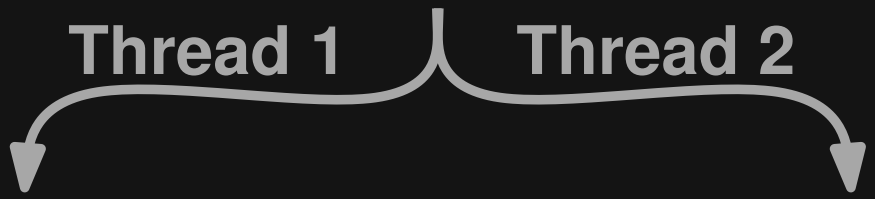
Thread 2

```
b.append("a")        b.append("c")
b.append("b")
```

"abc" ✓ "cab" ✓ "acb" ✓ "ac" ✗

Example

```
StringBuffer b = new StringBuffer()
```



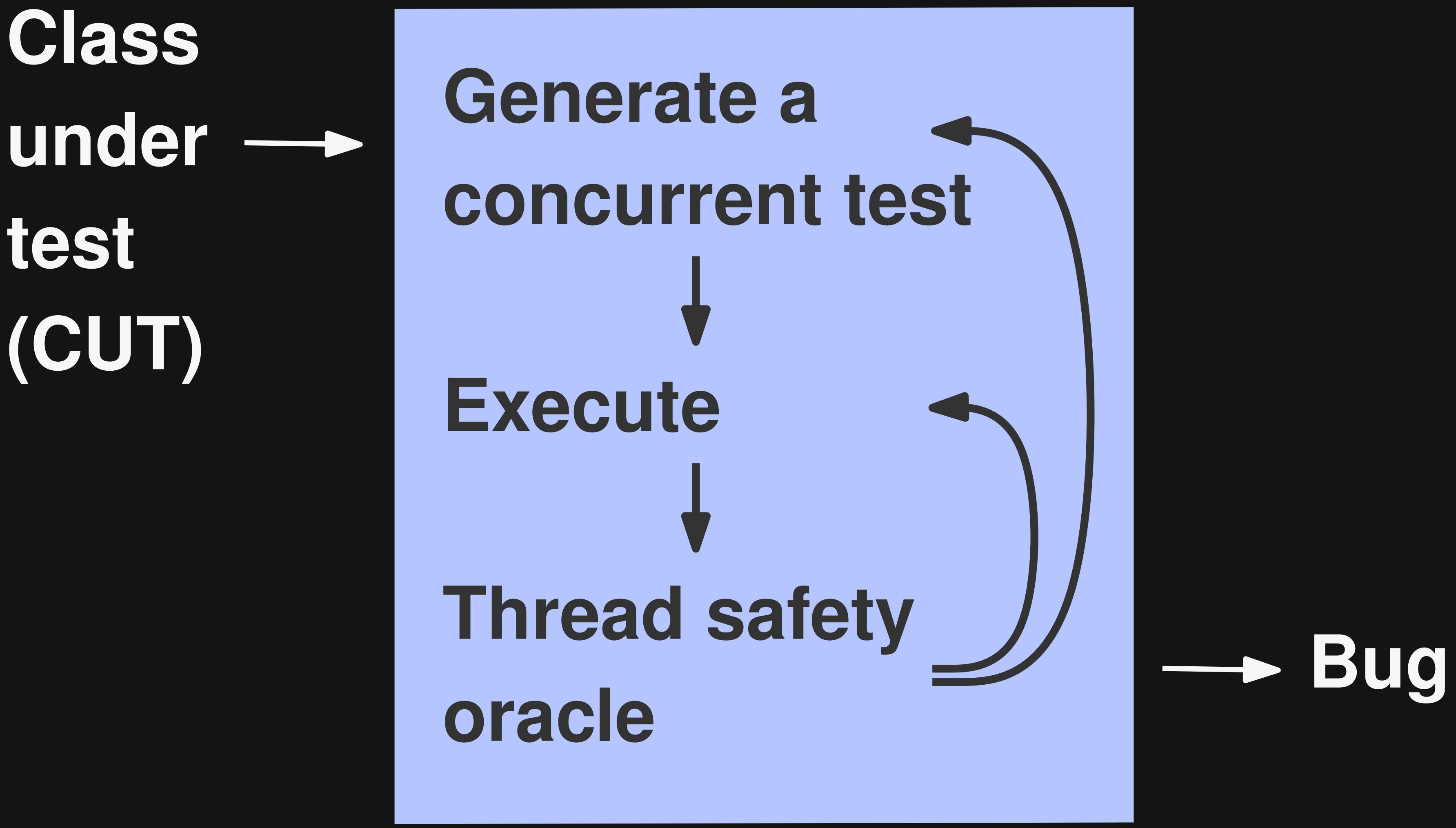
Thread 1

Thread 2

```
b.append("a")            b.append("c")
b.append("b")
```

"abc" ✓ "cab" ✓ "acb" ✓ "ac" ✗

Approach



Class under test (CUT)

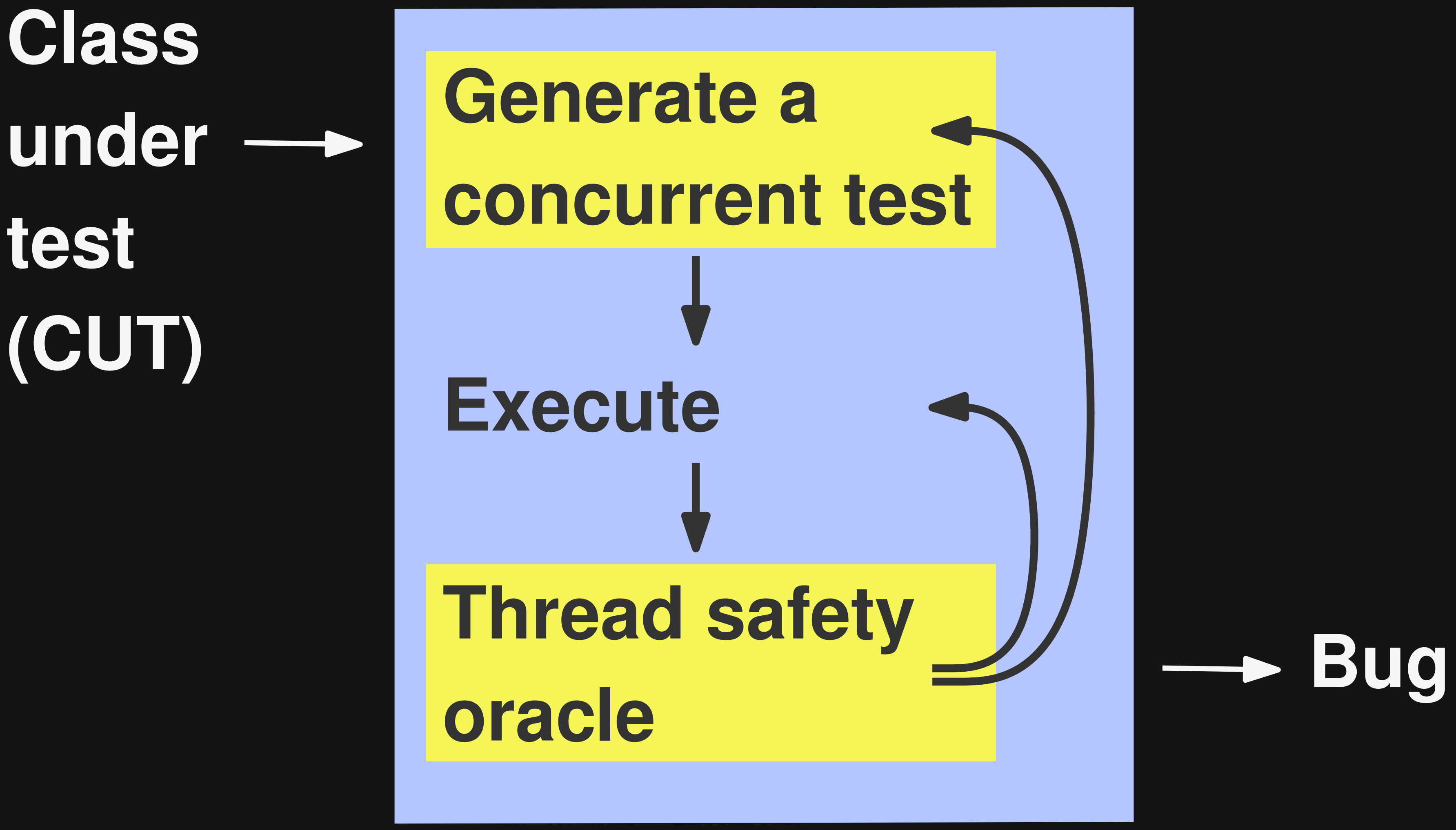
Generate a concurrent test

Execute

Thread safety oracle

Bug

Approach



Class under test (CUT)

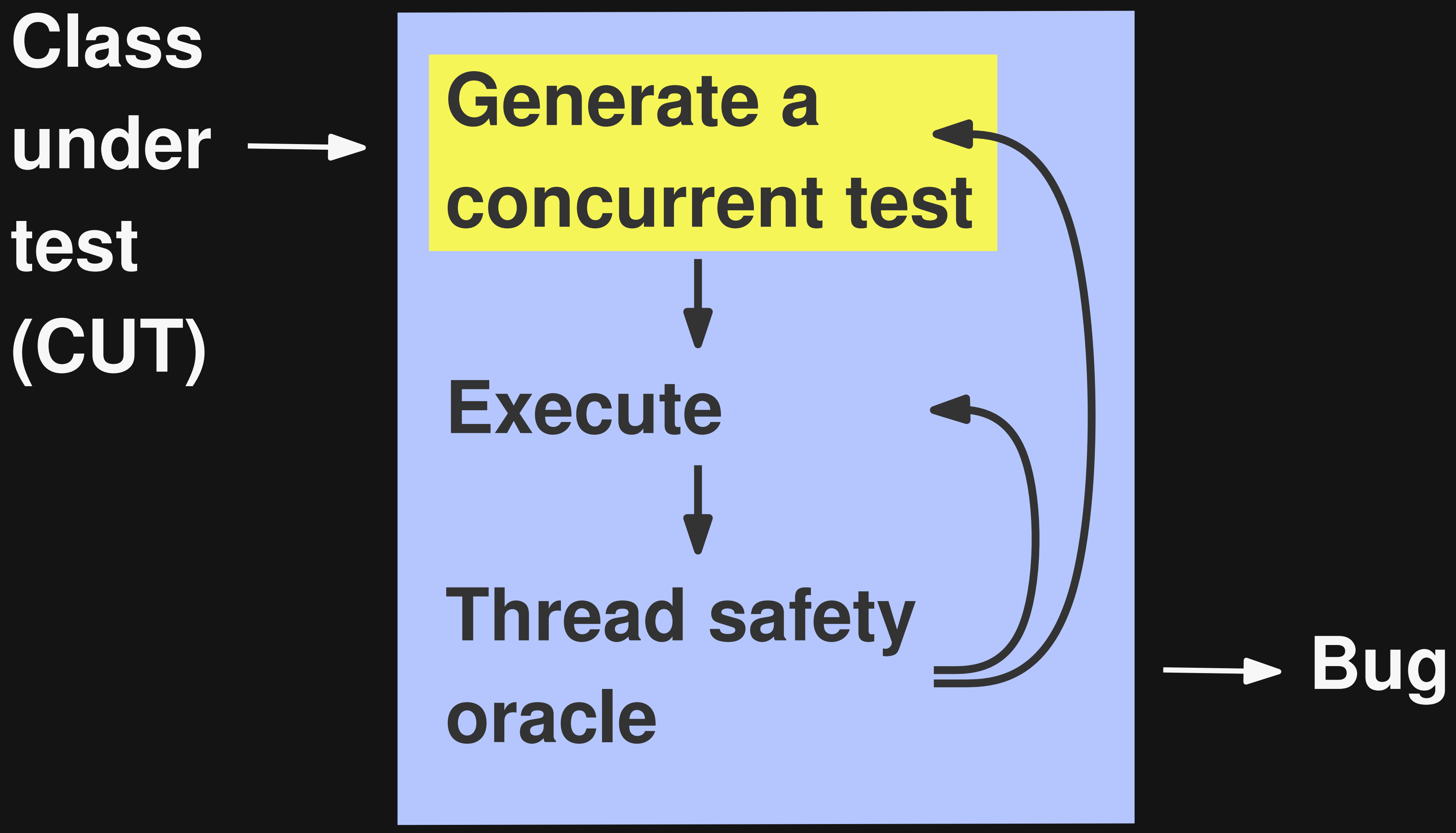
Generate a concurrent test

Execute

Thread safety oracle

Bug

Approach



Class under test (CUT)

Generate a concurrent test

Execute

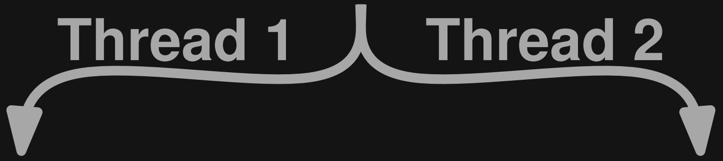
Thread safety oracle

Bug

Generating Concurrent Tests

Example:

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



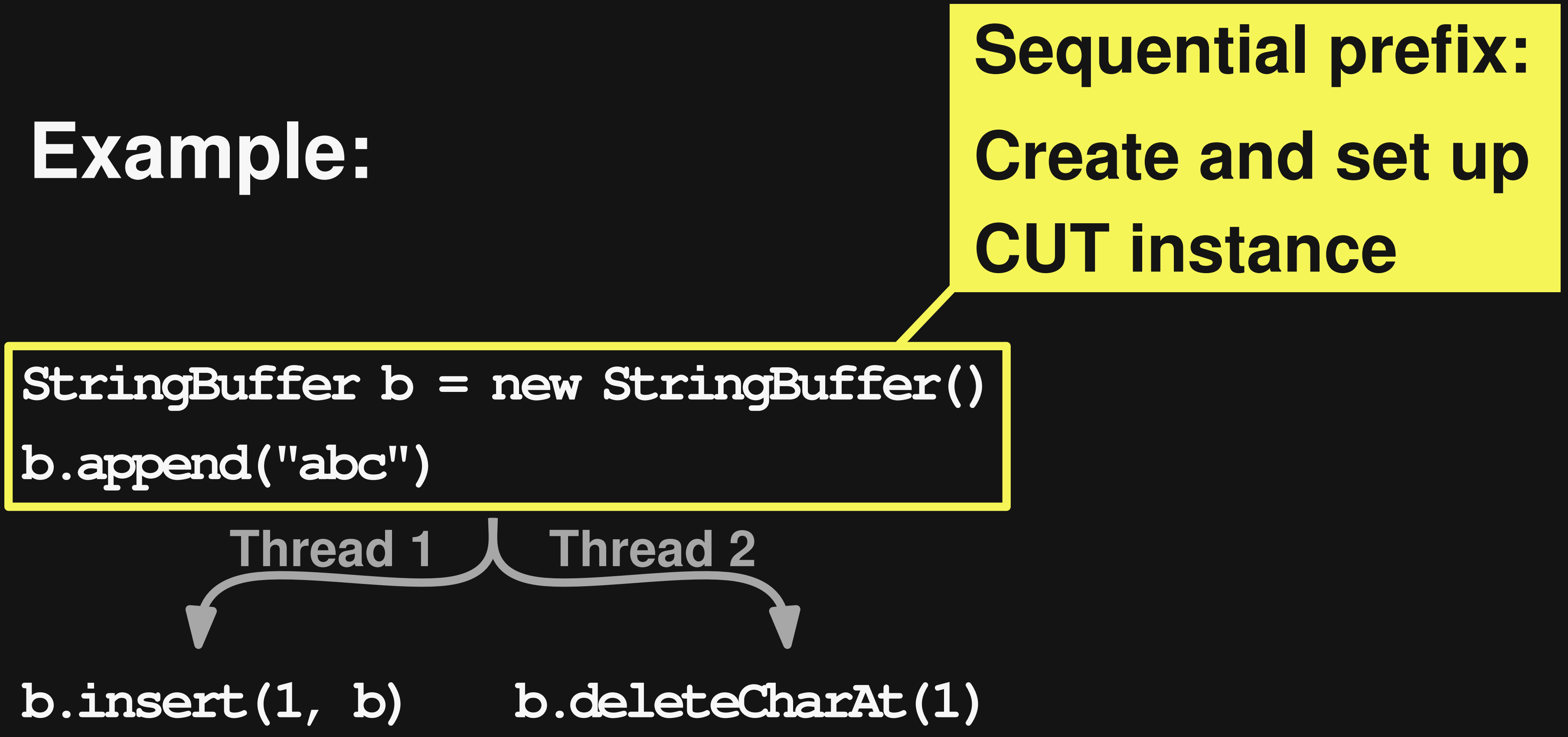
Thread 1

Thread 2

```
b.insert(1, b)    b.deleteCharAt(1)
```

Generating Concurrent Tests

Example:



Sequential prefix:

Create and set up CUT instance

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

Thread 1

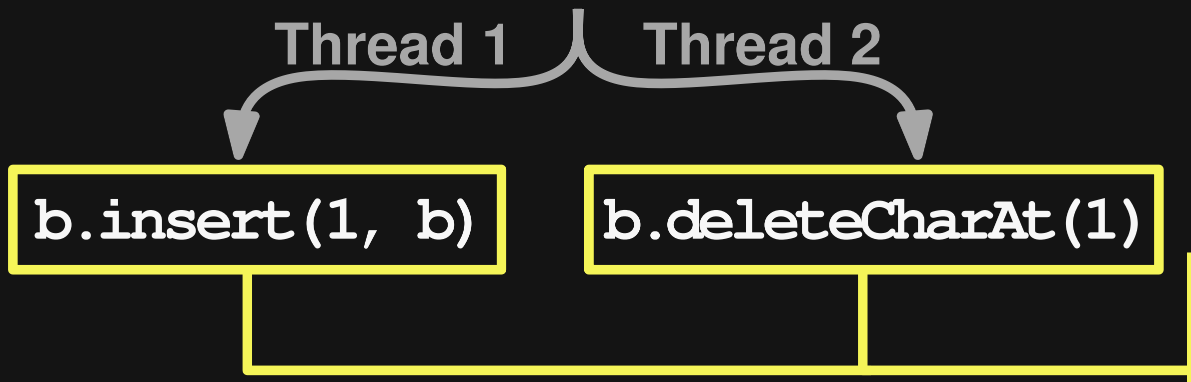
Thread 2

```
b.insert(1, b)    b.deleteCharAt(1)
```

Generating Concurrent Tests

Example:

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



```
b.insert(1, b)
```

```
b.deleteCharAt(1)
```

Concurrent suffixes: Use shared CUT instance

Test Generation Algorithm

1. **Create prefix**
    - Instantiate CUT
    - Call methods

2. **Create suffixes for prefix**
    - Call methods on shared CUT instance

3. **Prefix + two suffixes = test**

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Randomly select a constructor

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Randomly select a constructor

```
StringBuffer b = new StringBuffer()
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
```

✓

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Randomly select a method

```
StringBuffer b = new StringBuffer()
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Randomly select a method

```
StringBuffer b = new StringBuffer()
b.append(/* String */)
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Arguments:

a) Take available object

b) Call method returning required type

c) Random value

```
StringBuffer b = new StringBuffer()
b.append(/* String */)
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

Arguments:

a) Take available object

b) Call method returning required type

c) Random value

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

✓

Creating a Prefix

1. **Create prefix**
   - Instantiate CUT
   - Call methods

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

Creating Suffixes

2. **Create suffixes for prefix**
    - Call methods on shared CUT instance

Creating Suffixes

2. **Create suffixes for prefix**
    - Call methods on shared CUT instance

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

Randomly select a method

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

Randomly select a method

```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(/* int */, /* CharSequence */)
```

Creating Suffixes

2. Create suffixes for prefix
   - Call methods on shared CUT instance

Arguments:

a) Take available object

b) Call method returning required type

c) Random value

```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(/* int */, /* CharSequence */)
```

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

Arguments:

a) Take available object

b) Call method returning required type

c) Random value

```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(-5, b)
```

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
b.append("abc")



b.insert(-5, b)
```

Creating Suffixes

2. **Create suffixes for prefix**

- Call methods on shared CUT instance

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
b.append("abc")



b.insert(-5, b)
```

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

Arguments:
a) Take available object
b) Call method returning required type
c) Random value

```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(/* int */, /* CharSequence */)
```

Creating Suffixes

2. **Create suffixes for prefix**

- Call methods on shared CUT instance

Arguments:

a) Take available object

b) Call method returning required type

c) Random value

```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(1, b)
```

Creating Suffixes

2. **Create suffixes for prefix**

- Call methods on shared CUT instance

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
b.append("abc")



b.insert(1, b)
```

Creating Suffixes

2. **Create suffixes for prefix**

   - Call methods on shared CUT instance

After adding a call: Execute

```
StringBuffer b = new StringBuffer()
b.append("abc")



b.insert(1, b)
```

✓

Creating Suffixes

2. **Create suffixes for prefix**
    - Call methods on shared CUT instance

```
StringBuffer b = new StringBuffer()
b.append("abc")



b.insert(1, b)
```

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

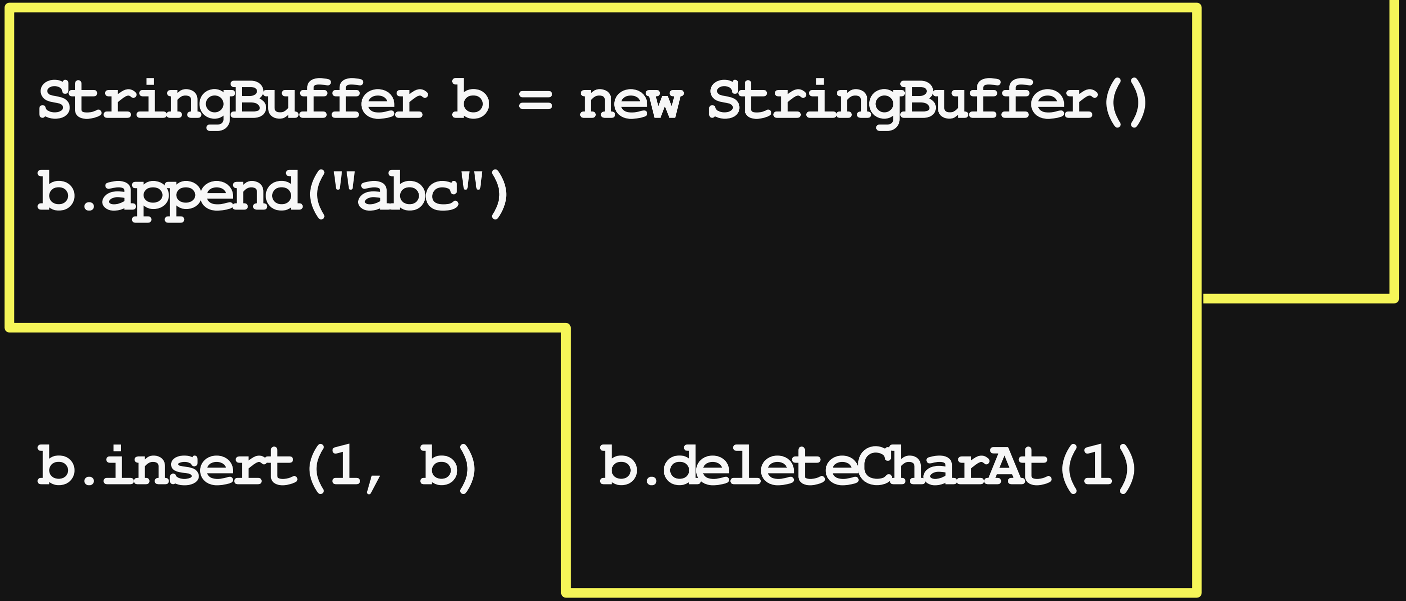
```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(1, b)    b.deleteCharAt(1)
```

Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

After adding a call: Execute



```
StringBuffer b = new StringBuffer()
b.append("abc")
```

```
b.insert(1, b)
```

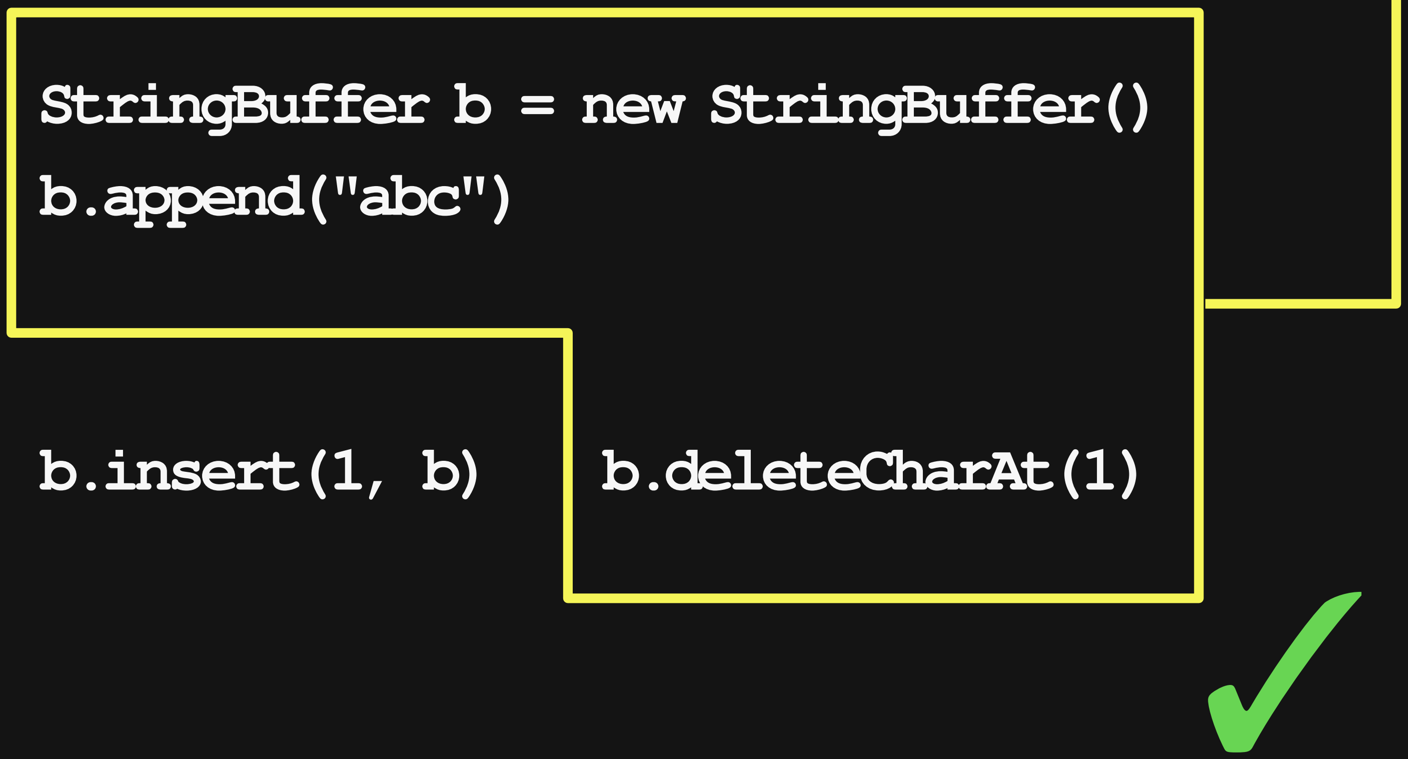
```
b.deleteCharAt(1)
```

Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

After adding a call: Execute



```
StringBuffer b = new StringBuffer()
b.append("abc")

b.insert(1, b)          b.deleteCharAt(1)
```

✓

Creating Suffixes

2. **Create suffixes for prefix**
   - Call methods on shared CUT instance

```
StringBuffer b = new StringBuffer()
b.append("abc")
```

```
b.insert(1, b)    b.deleteCharAt(1)
```

Creating a Test

3. Prefix + two suffixes = test

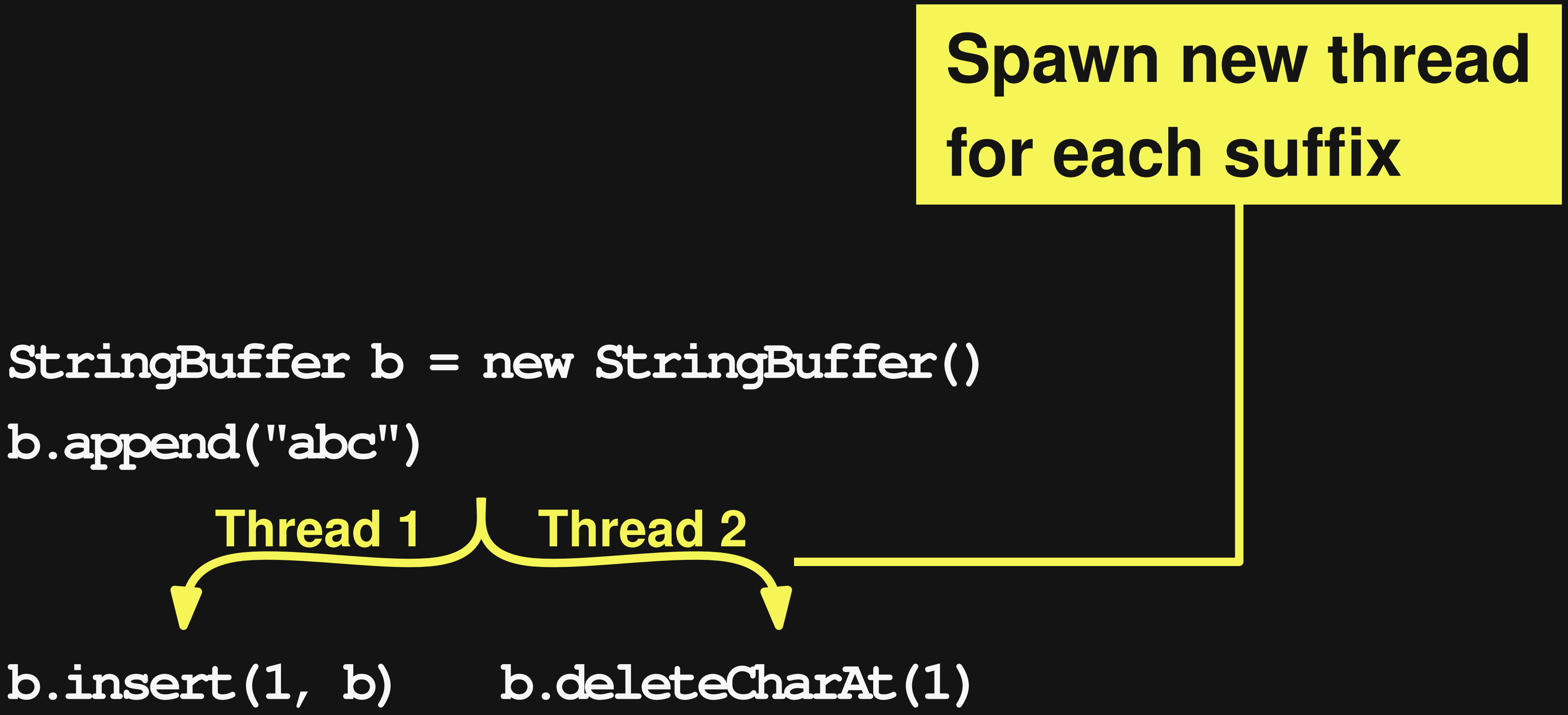
Creating a Test

3. Prefix + two suffixes = test

```
StringBuffer b = new StringBuffer()
b.append("abc")


b.insert(1, b)    b.deleteCharAt(1)
```

Creating a Test

3. Prefix + two suffixes = test

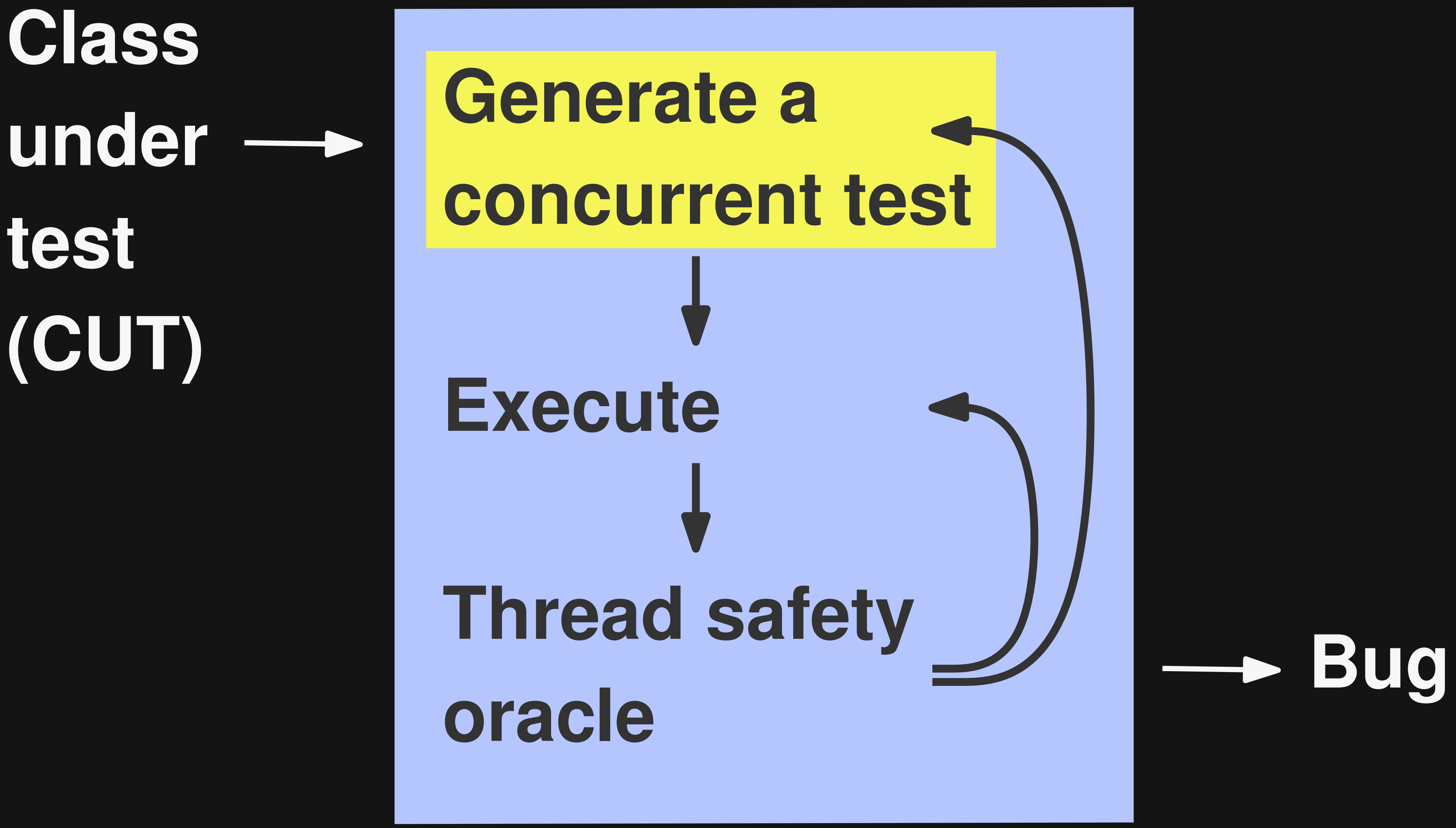


Spawn new thread for each suffix

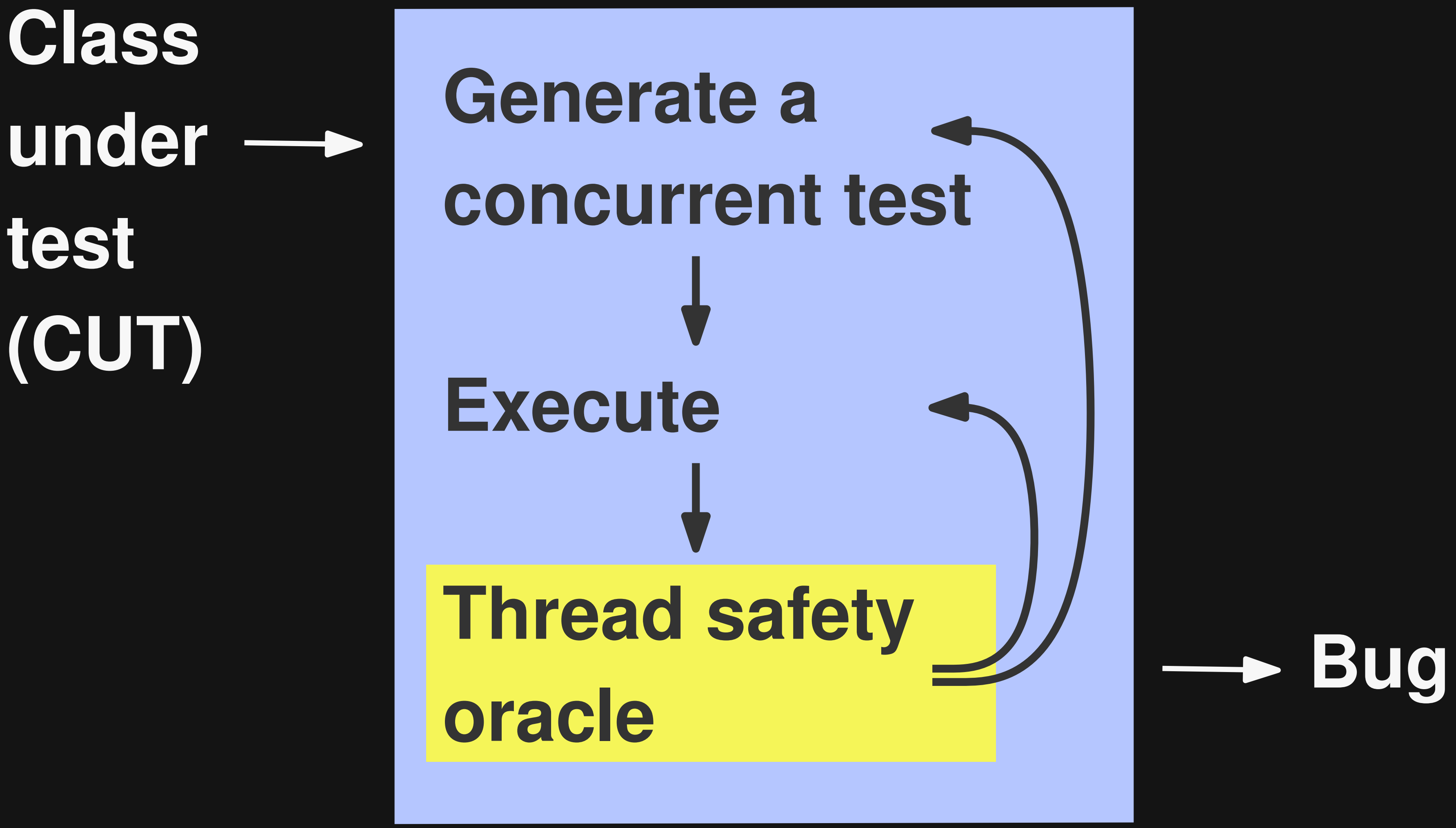
```
StringBuffer b = new StringBuffer()
b.append("abc")
```

```
b.insert(1, b)    b.deleteCharAt(1)
```

Approach


Class under test (CUT)
Generate a concurrent test
Execute
Thread safety oracle
Bug

Approach



Class under test (CUT)

Generate a concurrent test

Execute

Thread safety oracle

Bug

Thread Safety Oracle

Does the test execution expose a thread safety violation?

- Focus on exceptions and deadlocks
- Compare concurrent execution to linearizations

Assumptions

Concurrency-only crashes are undesired

- Matches definition of thread safety

Control over all input to tests

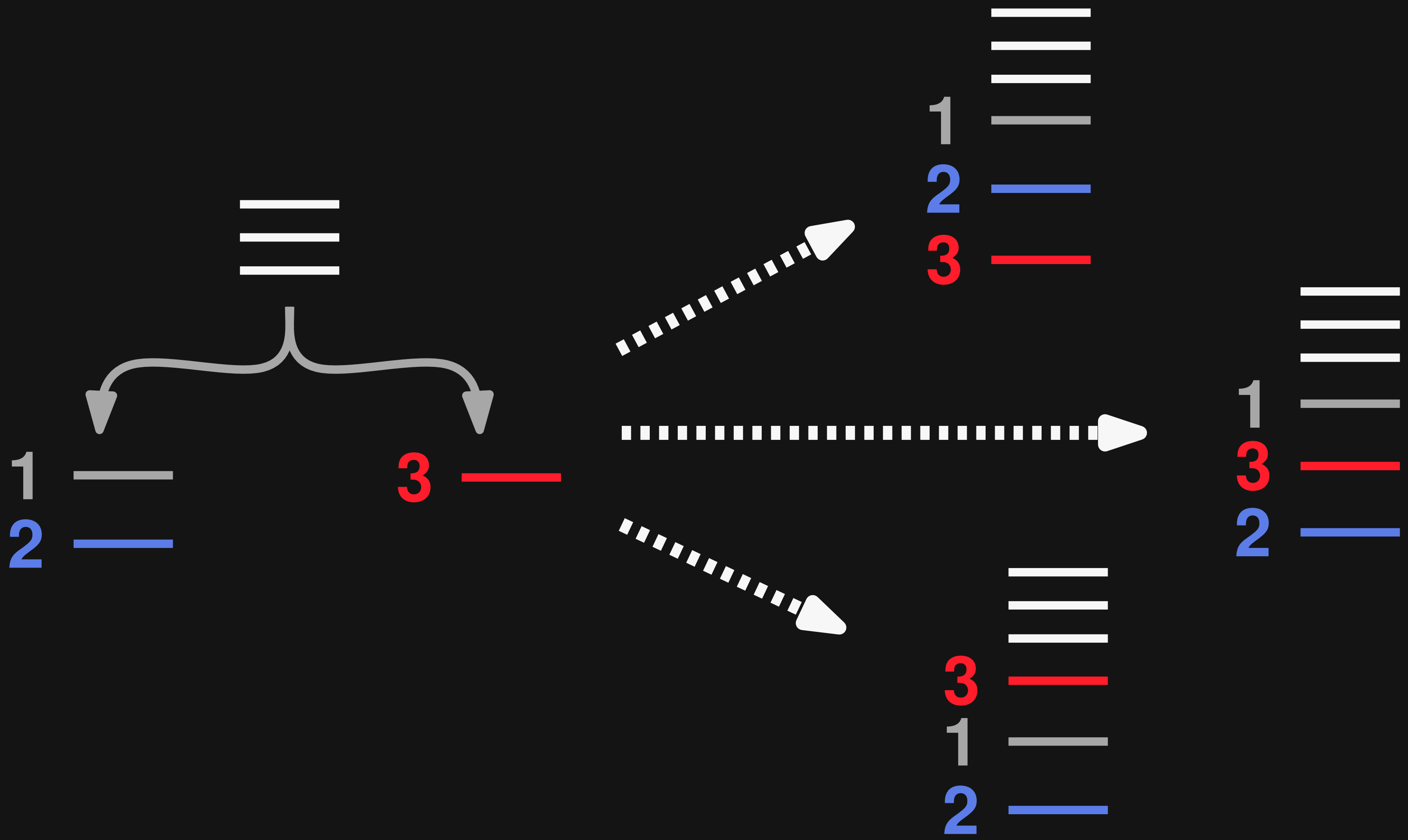
- Sequential execution: Deterministic

Linearizations

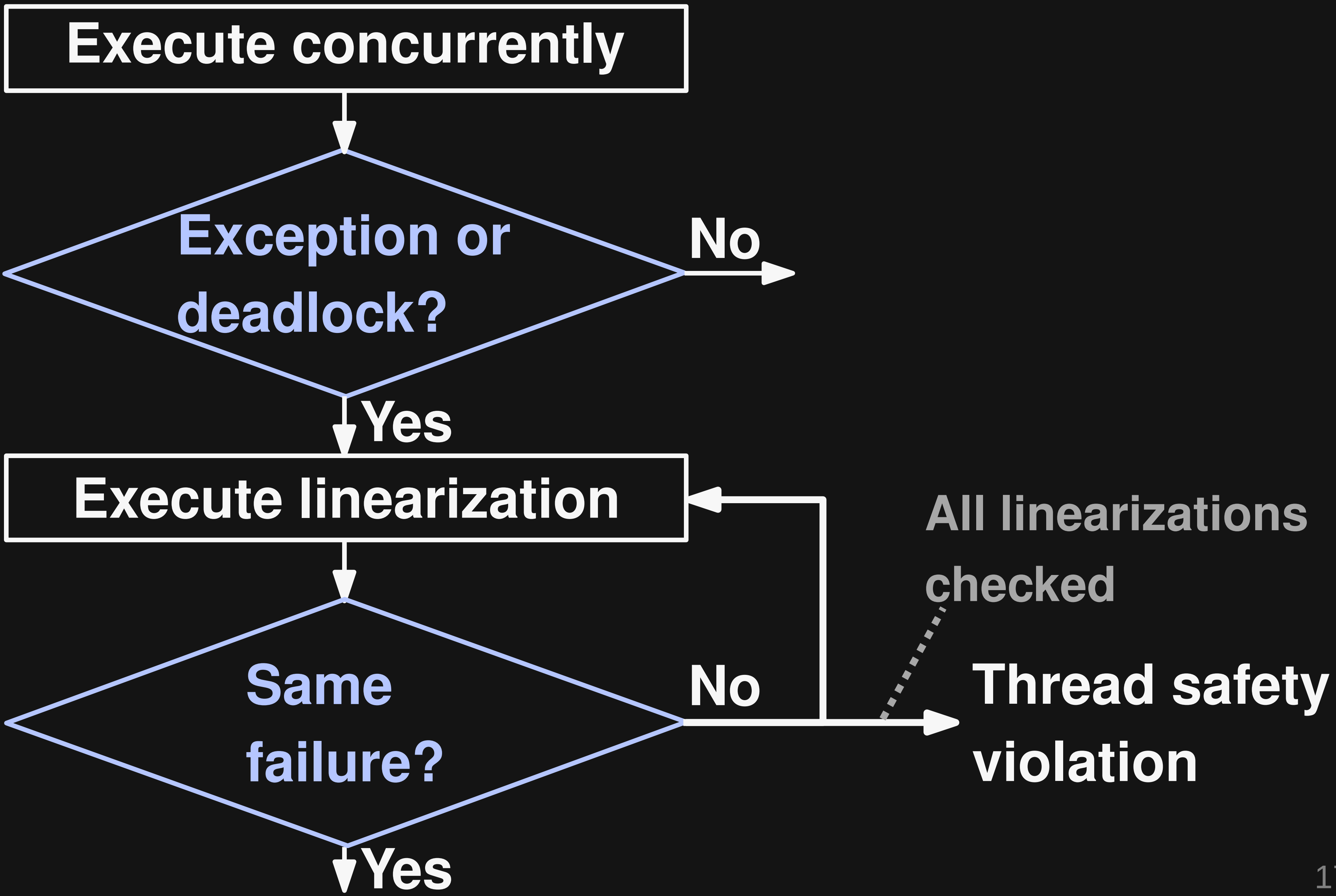
- Put all calls into one thread
- Preserve order of calls within a thread

Linearizations

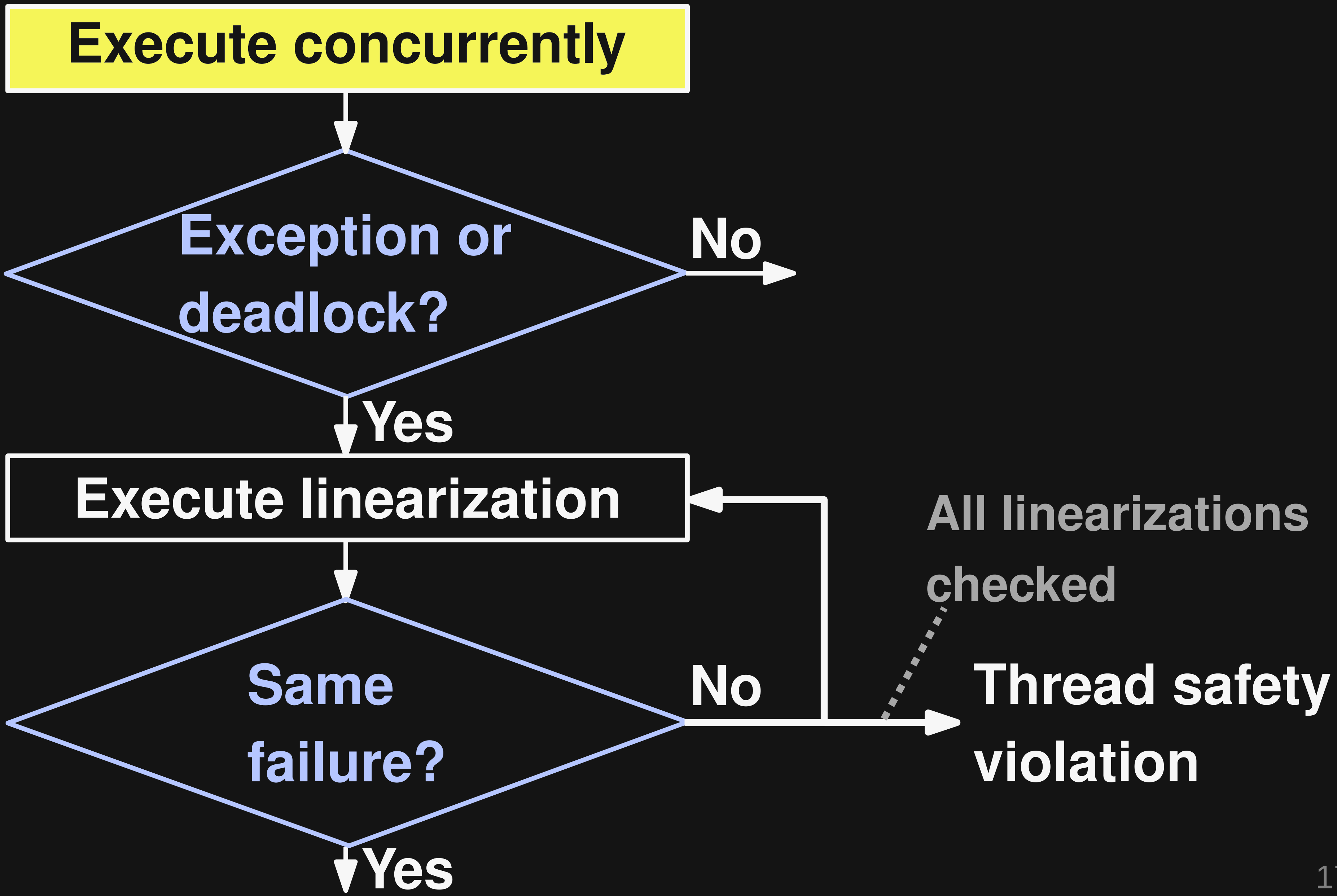
- Put all calls into one thread
- Preserve order of calls within a thread



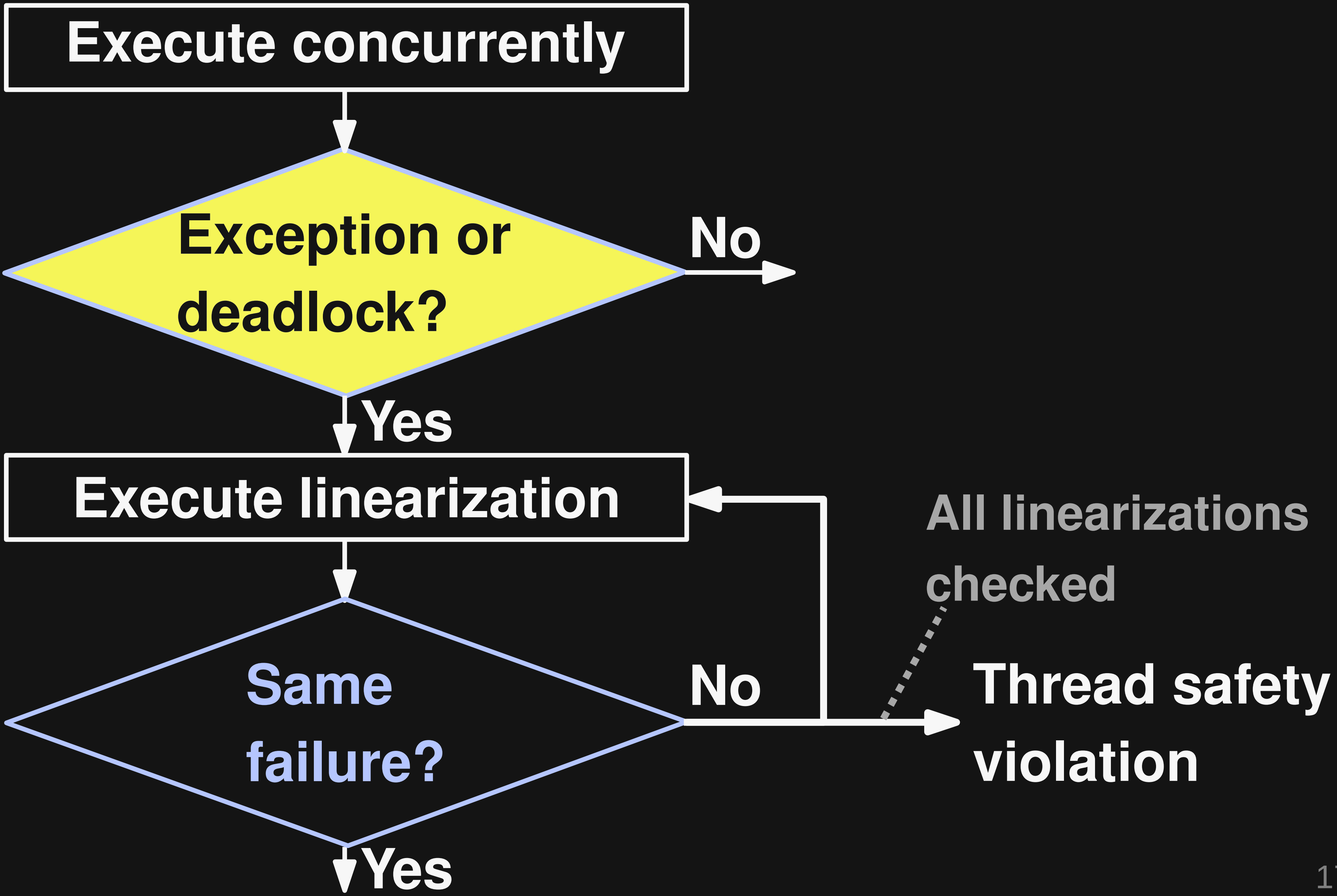
The Oracle


Execute concurrently
Exception or deadlock?
No
Yes
Execute linearization
All linearizations checked
Same failure?
No
Thread safety violation
Yes

The Oracle


Execute concurrently
Exception or deadlock?
No
Yes
Execute linearization
All linearizations checked
Same failure?
No
Thread safety violation
Yes

The Oracle



Execute concurrently

Exception or deadlock?

No

Yes

Execute linearization

Same failure?

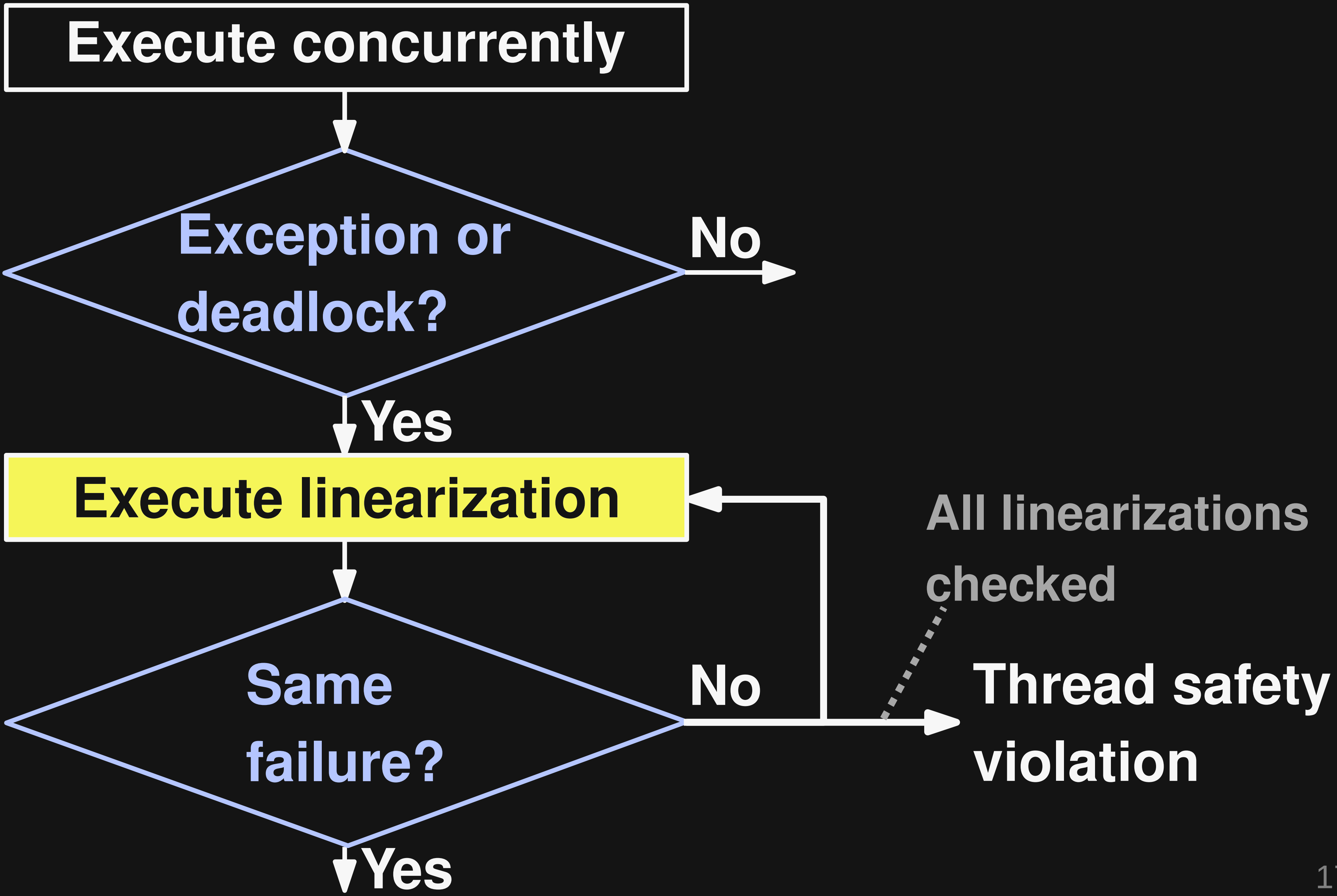
No

Yes

All linearizations checked

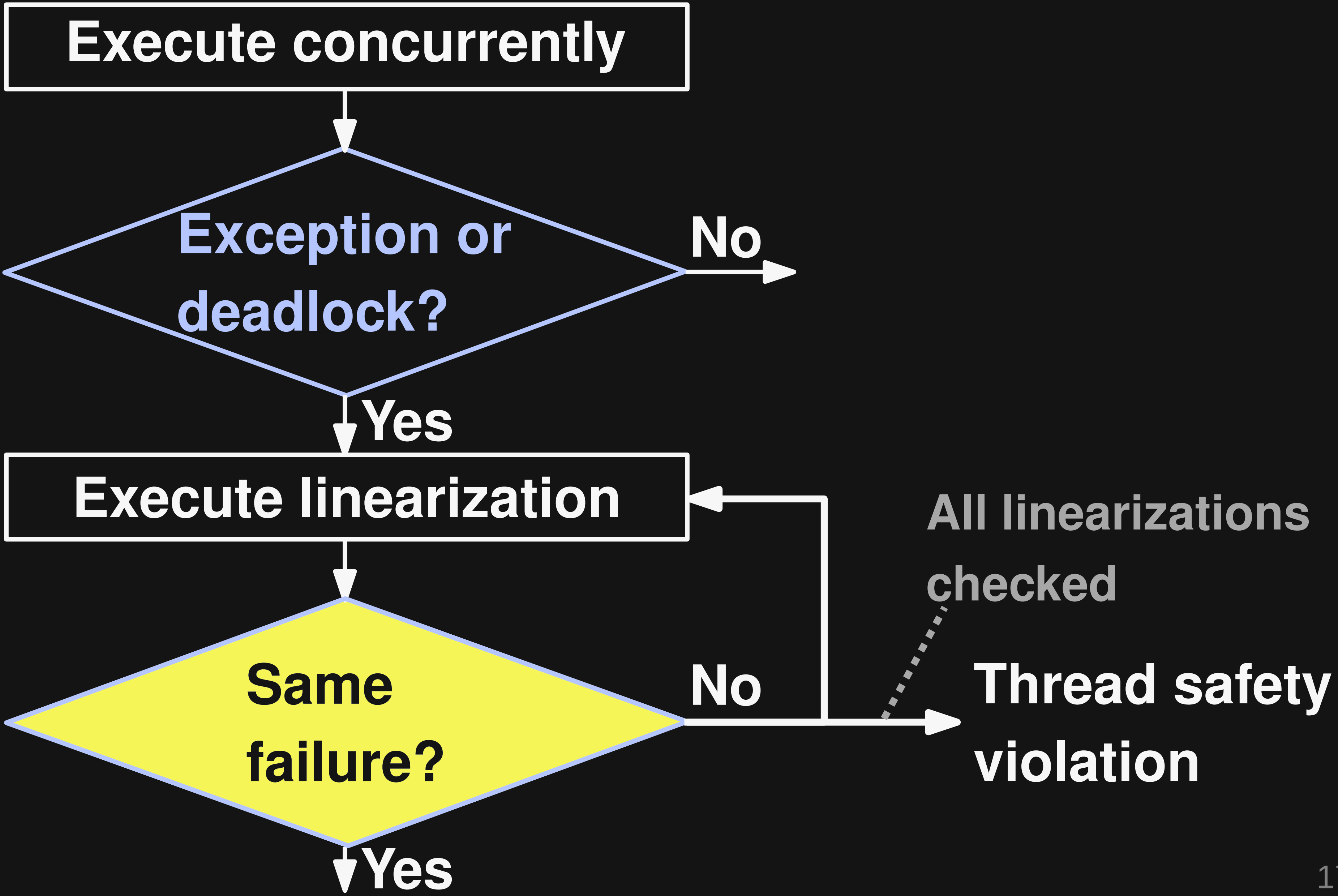
Thread safety violation

The Oracle

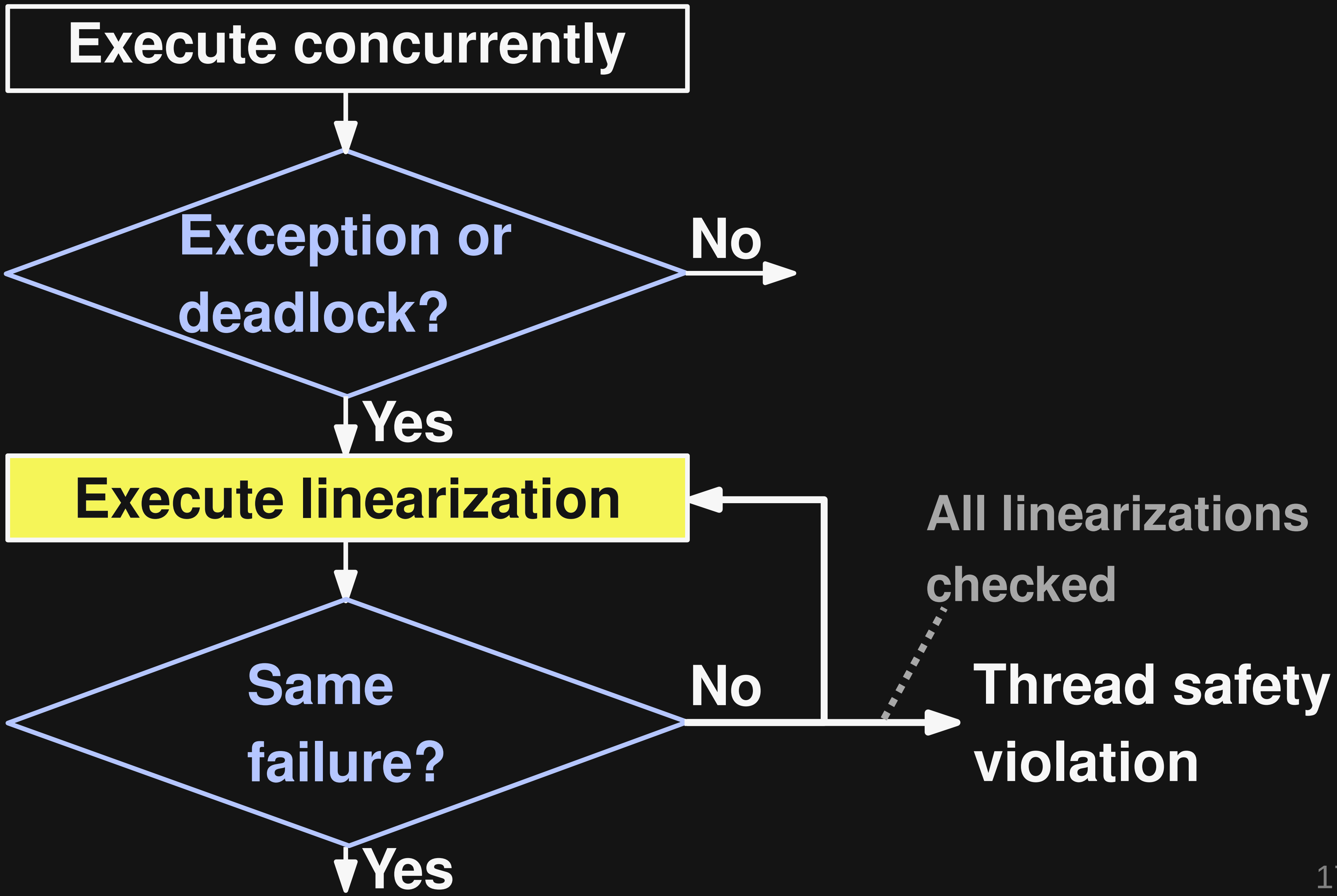


Execute concurrently
Exception or deadlock?
No
Yes
Execute linearization
All linearizations checked
Same failure?
No
Thread safety violation
Yes

The Oracle


Execute concurrently
Exception or deadlock?
No
Yes
Execute linearization
All linearizations checked
Same failure?
No
Thread safety violation
Yes

The Oracle



Execute concurrently

Exception or deadlock?

No

Yes

Execute linearization

All linearizations checked

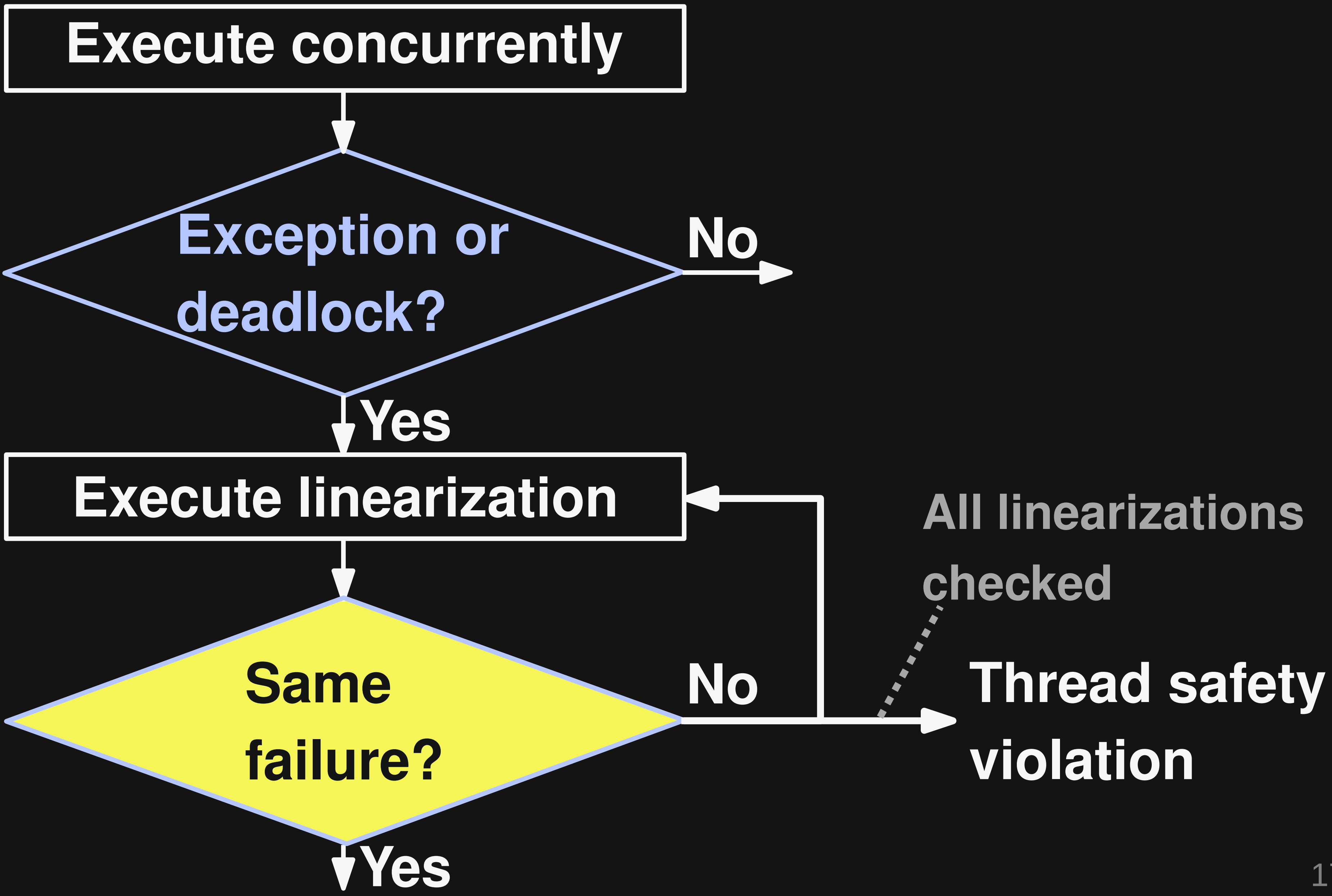
Same failure?

No

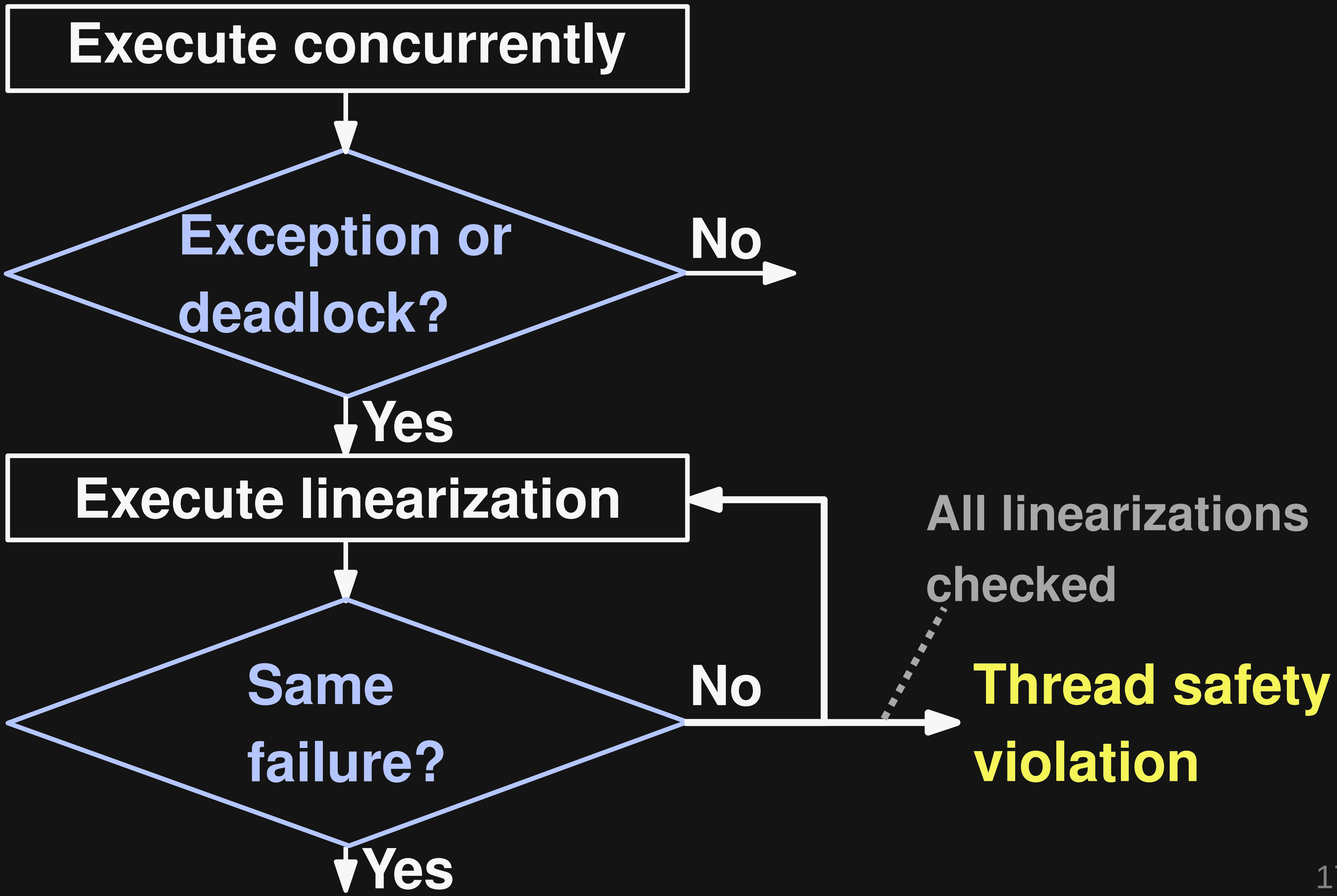
Thread safety violation

Yes

The Oracle


Execute concurrently
Exception or deadlock?
No
Yes
Execute linearization
All linearizations checked
Same failure?
No
Thread safety violation
Yes

The Oracle



Execute concurrently

Exception or deadlock?

No

Yes

Execute linearization

Same failure?

No

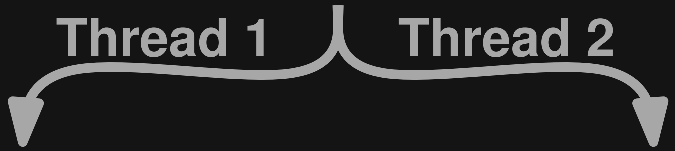
Yes

All linearizations checked

Thread safety violation

Example

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



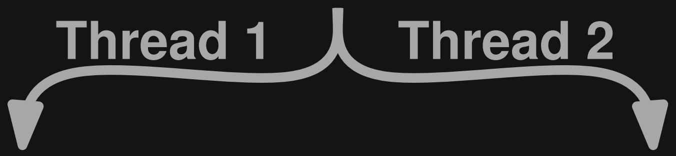
Thread 1

Thread 2

```
b.insert(1, b)    b.deleteCharAt(1)
```

Example

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



Thread 1

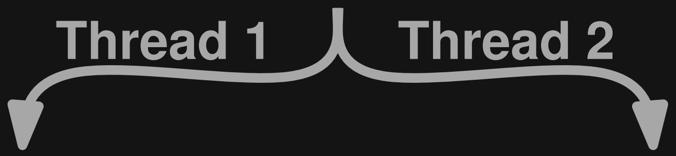
Thread 2

```
b.insert(1, b)   b.deleteCharAt(1)
```

⚠

Example

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



Thread 1 Thread 2

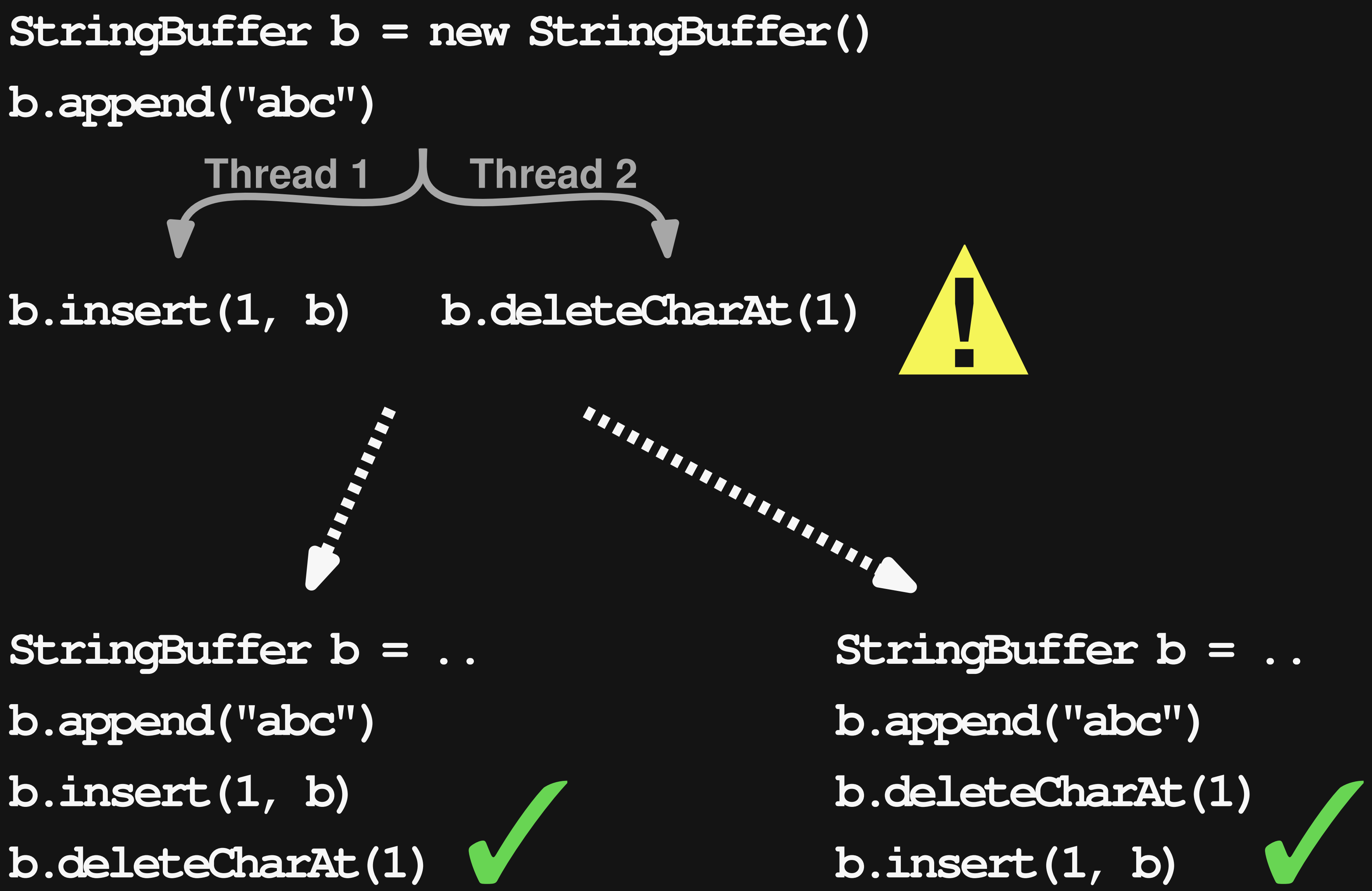
```
b.insert(1, b)    b.deleteCharAt(1)
```

⚠

```
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

✓

Example



```
StringBuffer b = new StringBuffer()
b.append("abc")
```

```
b.insert(1, b)    b.deleteCharAt(1)
```

⚠

```
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

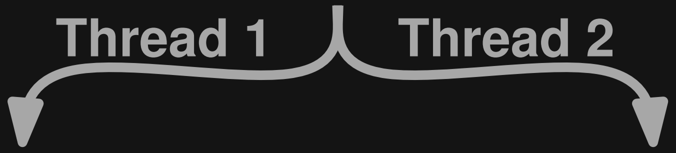
✓

```
StringBuffer b = ..
b.append("abc")
b.deleteCharAt(1)
b.insert(1, b)
```

✓

Example

```
StringBuffer b = new StringBuffer()
b.append("abc")
```



Thread 1

Thread 2

```
b.insert(1, b)    b.deleteCharAt(1)
```

⚠

Thread safety violation

```
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

✓

```
StringBuffer b = ..
b.append("abc")
b.deleteCharAt(1)
b.insert(1, b)
```

✓

Properties of the Oracle

Sound but incomplete *

- All reported violations are real
- Cannot guarantee thread safety

Independent of bug type

- Data races
- Atomicity violations
- Deadlocks

* with respect to incorrectness

Implementation

thread-safe.org

Automatic and precise thread safety checking

Evaluation

1. Effectiveness in finding bugs

2. Performance

Setup:

- Thread-safe classes from six Java libraries (e.g., JDK, Apache DBCP)
- Intel Xeon (8x3GHz)

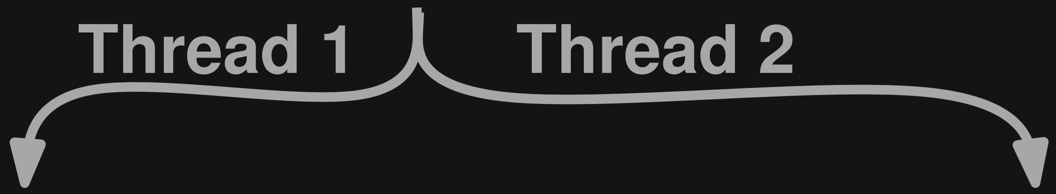
Bugs

Found 15 bugs and 0 false positives

- **9 known bugs**
- **6 previously unknown bugs**
  - E.g., in JDK and Apache DBCP

Example: Apache DBCP

```
DataSource ds = new DataSource()
```

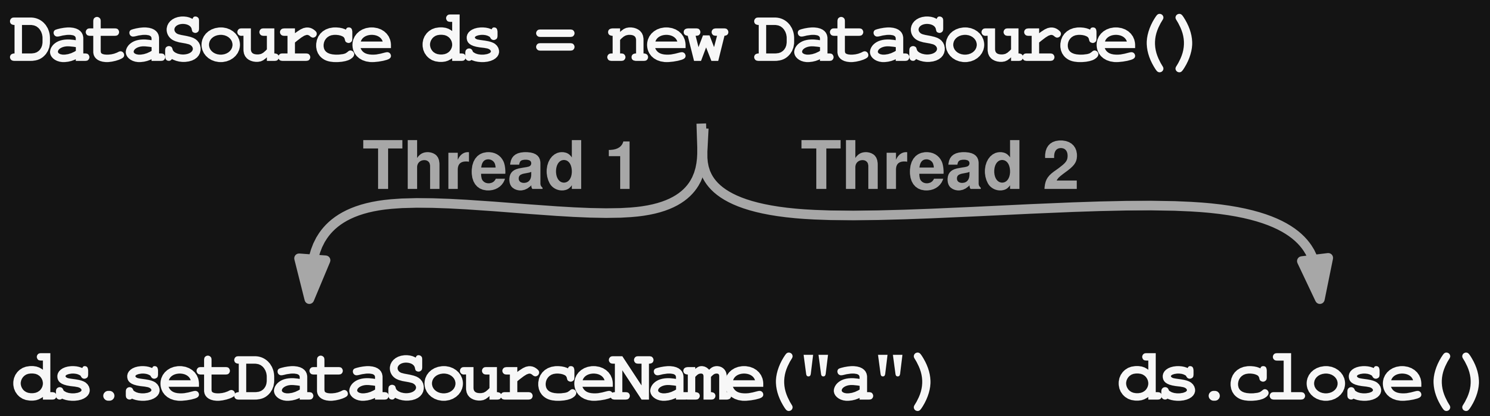


Thread 1

Thread 2

```
ds.setDataSourceName("a")      ds.close()
```

Example: Apache DBCP



```
DataSource ds = new DataSource()
```

Thread 1

Thread 2

```
ds.setDataSourceName("a")     ds.close()
```

⚠ **ConcurrentModificationException**

Reason: Unsynchronized use of thread-unsafe collection

Confirmed as bug: Issue #369

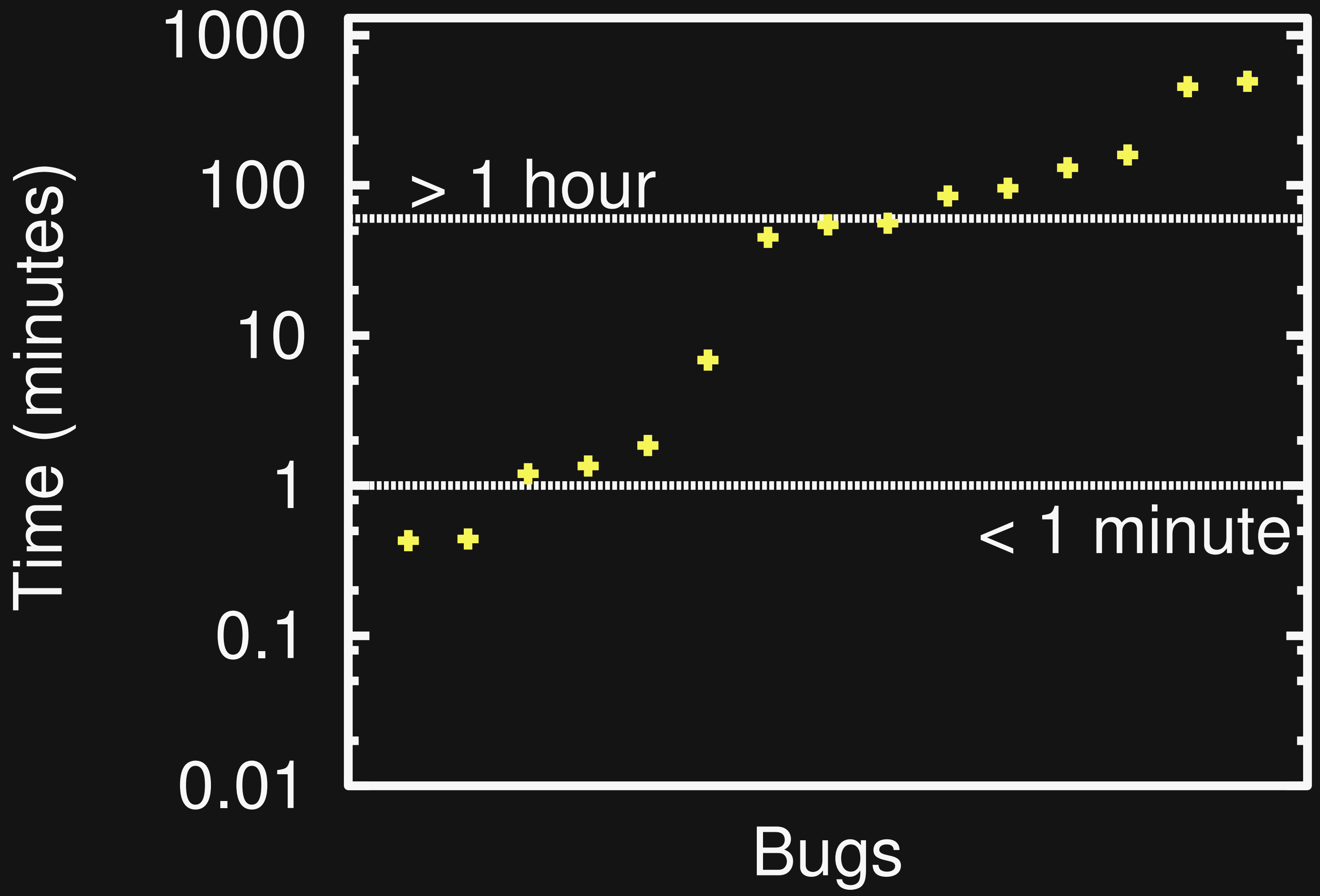
Kinds of Failures

12 of 15 failures are implicit (VM or JDK)

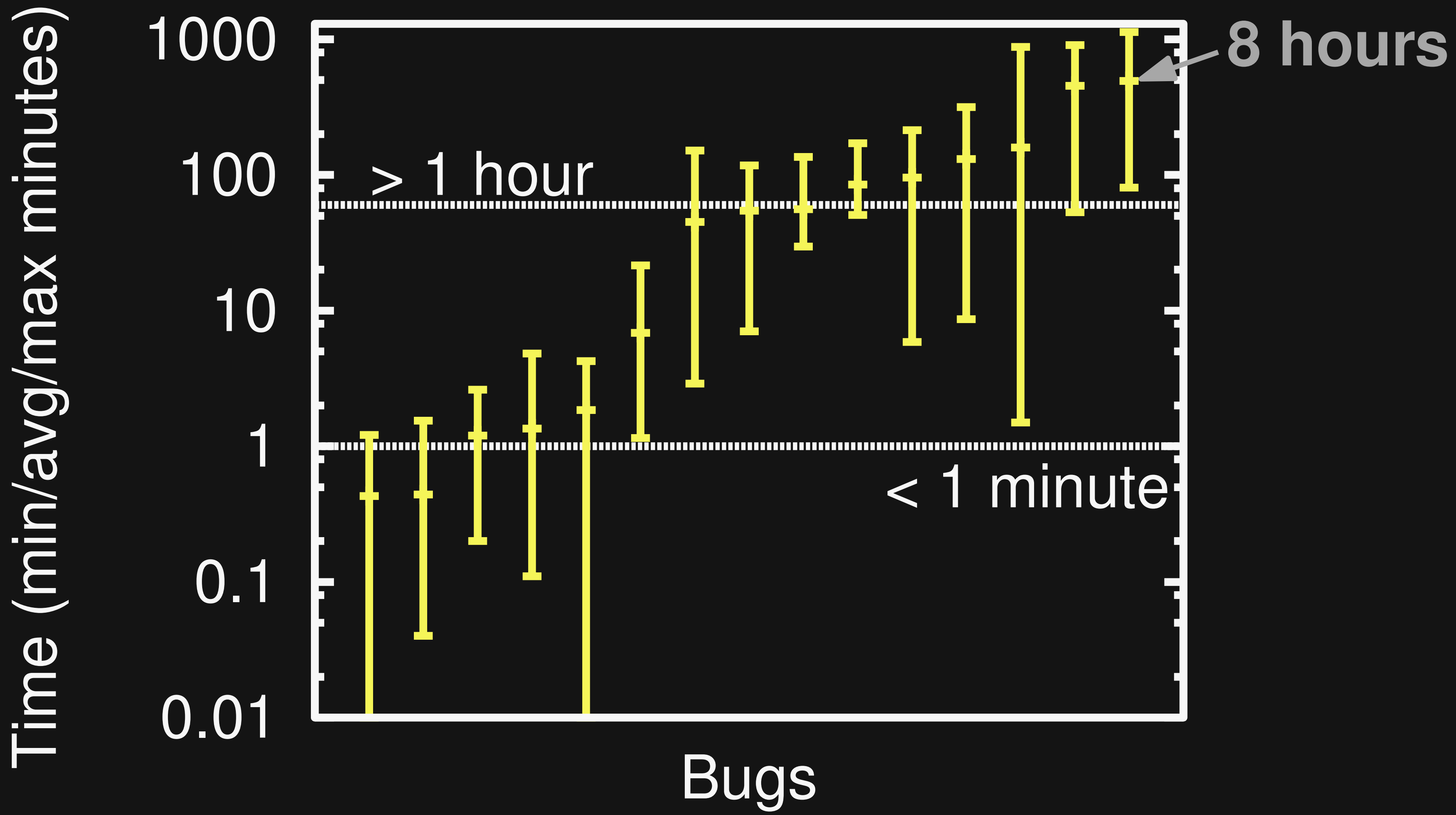
Most common:

- **NullPointerException**
- **ConcurrentModificationException**

Performance



Performance



Conclusion

Concurrency: More and more important

Need tools to test thread-safe classes

This work:

- Fully automatic testing
- Only real bugs reported

Thank you!

Try it:

thread-safe.org

Fully Automatic and Precise Detection of Thread Safety Violations
Michael Pradel and Thomas R. Gross, ETH Zurich