Program Analysis

Analyzing Concurrent Programs (Part 3)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2020/2021

Outline

1. **Introduction**
2. **Dynamic Data Race Detection**
3. **Testing Thread-Safe Classes** 
4. **Exploring Interleavings**

Mostly based on these papers:

- *Eraser: A Dynamic Data Race Detector for Multithreaded Programs*, Savage et al., ACM TOCS, 1997
- *Fully Automatic and Precise Detection of Thread Safety Violations*, Pradel and Gross, PLDI 2012
- *Finding and Reproducing Heisenbugs in Concurrent Programs*, Musuvathi et al., USENIX 2008

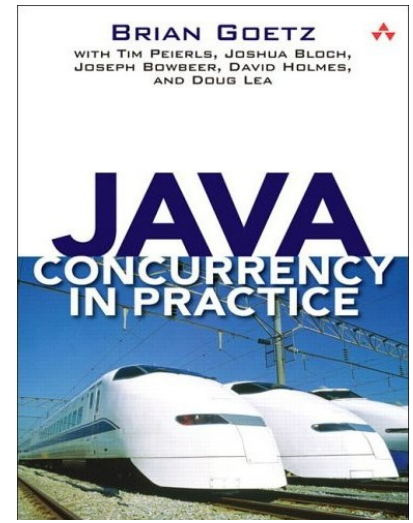
Thread Safety

- Popular way to encapsulate the challenges of concurrent programming: Thread-safe classes
- Class ensures correct synchronization
- Clients can use instances as if they were alone
- Rest of program can treat implementation of thread-safe class as a blackbox

Thread Safety (2)

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

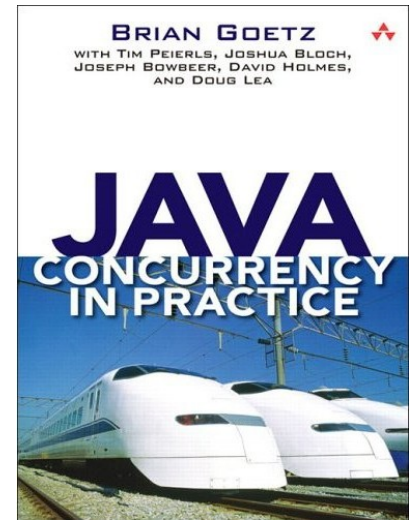
page 18



Thread Safety (2)

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

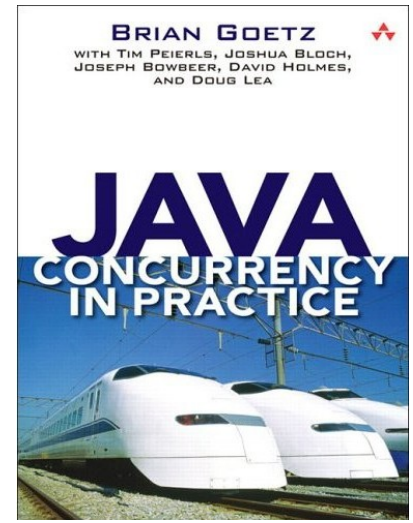
page 18



Thread Safety (2)

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

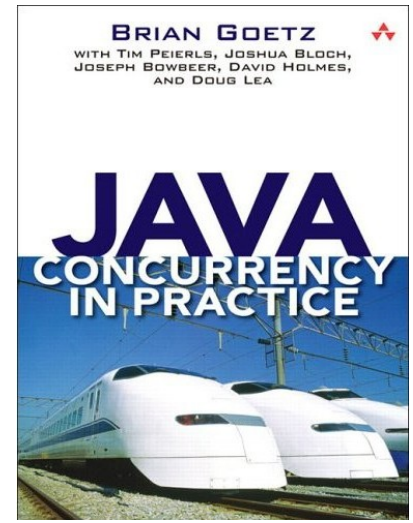
page 18



Thread Safety (2)

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

page 18



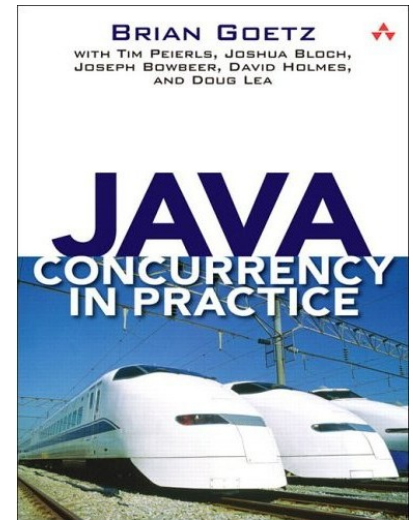
"operations ... behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads"

StringBuffer API documentation, JDK 6

Thread Safety (2)

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

page 18



"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 from JDK

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


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

Example from JDK

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


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

Quiz: What can be the content of `b` if `StringBuffer` is thread-safe?

Example from JDK

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


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

"abc" ✓ "cab" ✓ "acb" ✓ "ac" ✗ "bac" ✗

Testing Thread-Safe Classes

- Correctness of program relies on thread safety of specific classes
- But: What if the class is actually not thread-safe?

- ConTeGe = Concurrent Test Generator
- Creates multi-threaded unit tests
- Detects thread safety violations by comparing concurrent behavior against linearizations

Example Bug from JDK

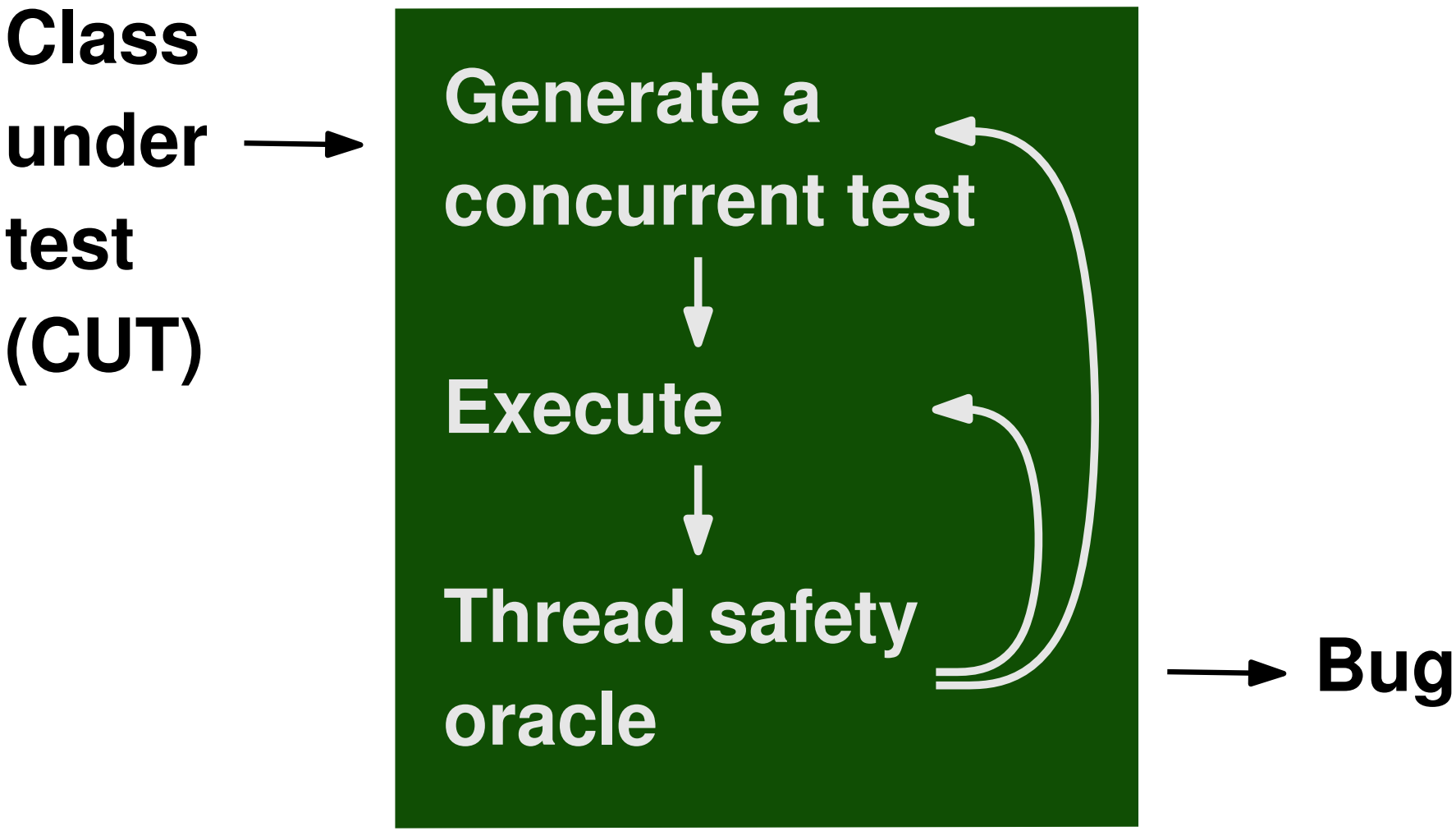
```
StringBuffer b = new StringBuffer()
b.append("abc")
```


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

IndexOutOfBoundsException

Confirmed as bug: Issue #7100996

ConTeGe


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

Generating Concurrent Tests

Example:

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

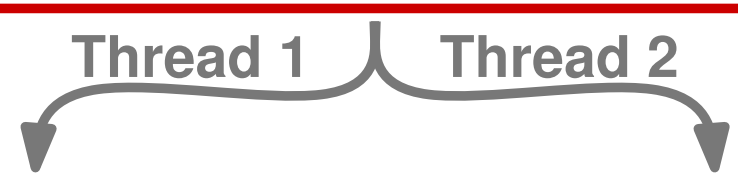

```
b.insert(1, b)      b.deleteCharAt(1)
```

Generating Concurrent Tests

Example:

Sequential prefix:
Create and set up CUT instance

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

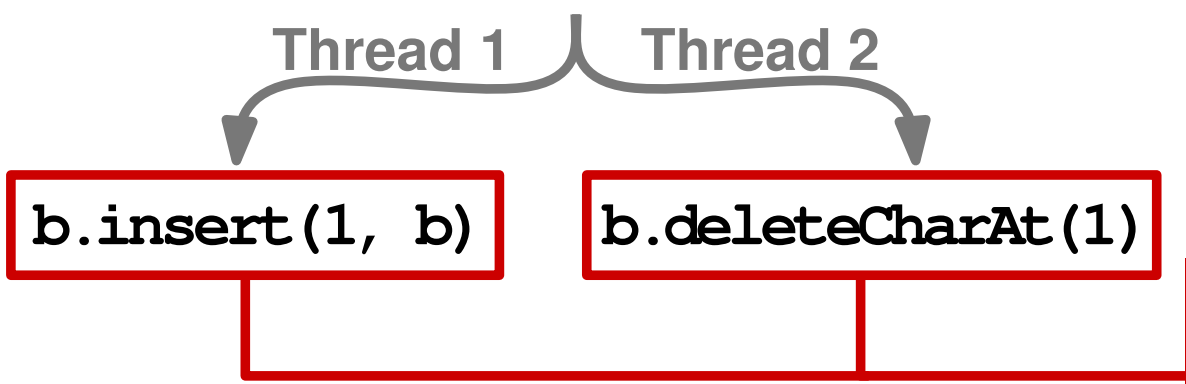



```
b.insert(1, b)      b.deleteCharAt(1)
```

Generating Concurrent Tests

Example:

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



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**

Selection of methods via feedback-directed test generation

Creating a Prefix

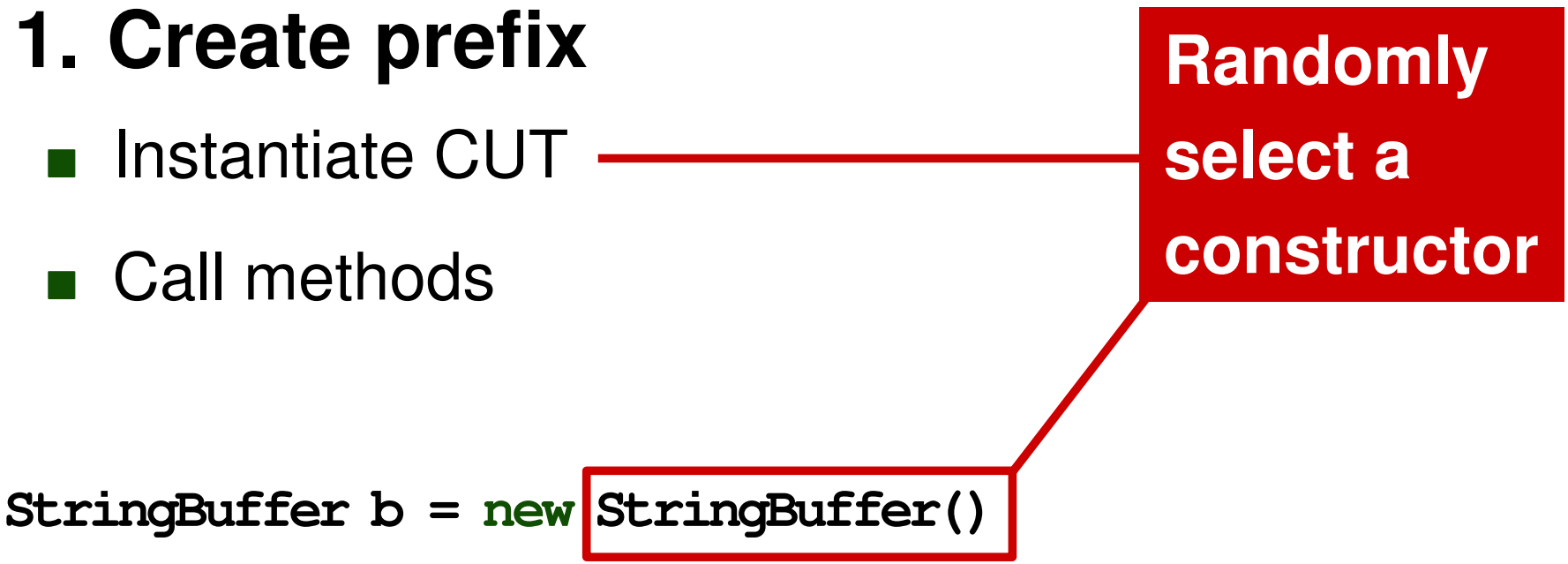
1. Create prefix

- Instantiate CUT
- Call methods

Creating a Prefix

1. Create prefix

- Instantiate CUT
- Call methods



Creating a Prefix

1. Create prefix

- Instantiate CUT
- Call methods

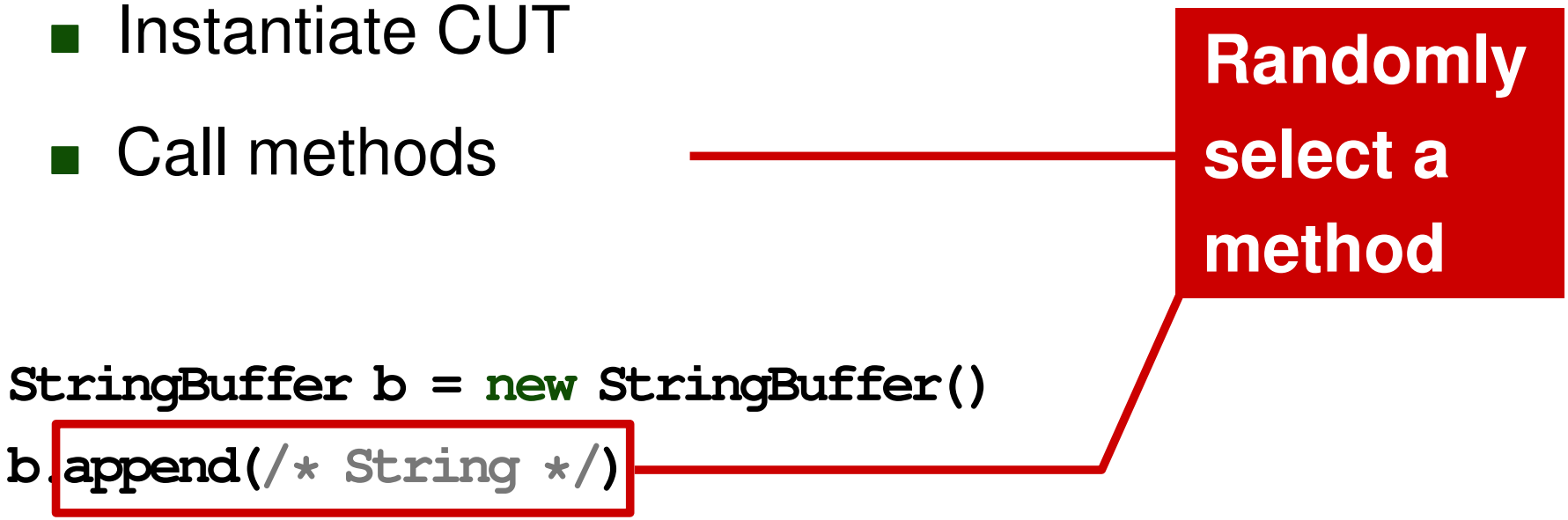
After adding a call: Execute

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

Creating a Prefix

1. Create prefix

- Instantiate CUT
- Call methods



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 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")
```

```
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

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

```
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

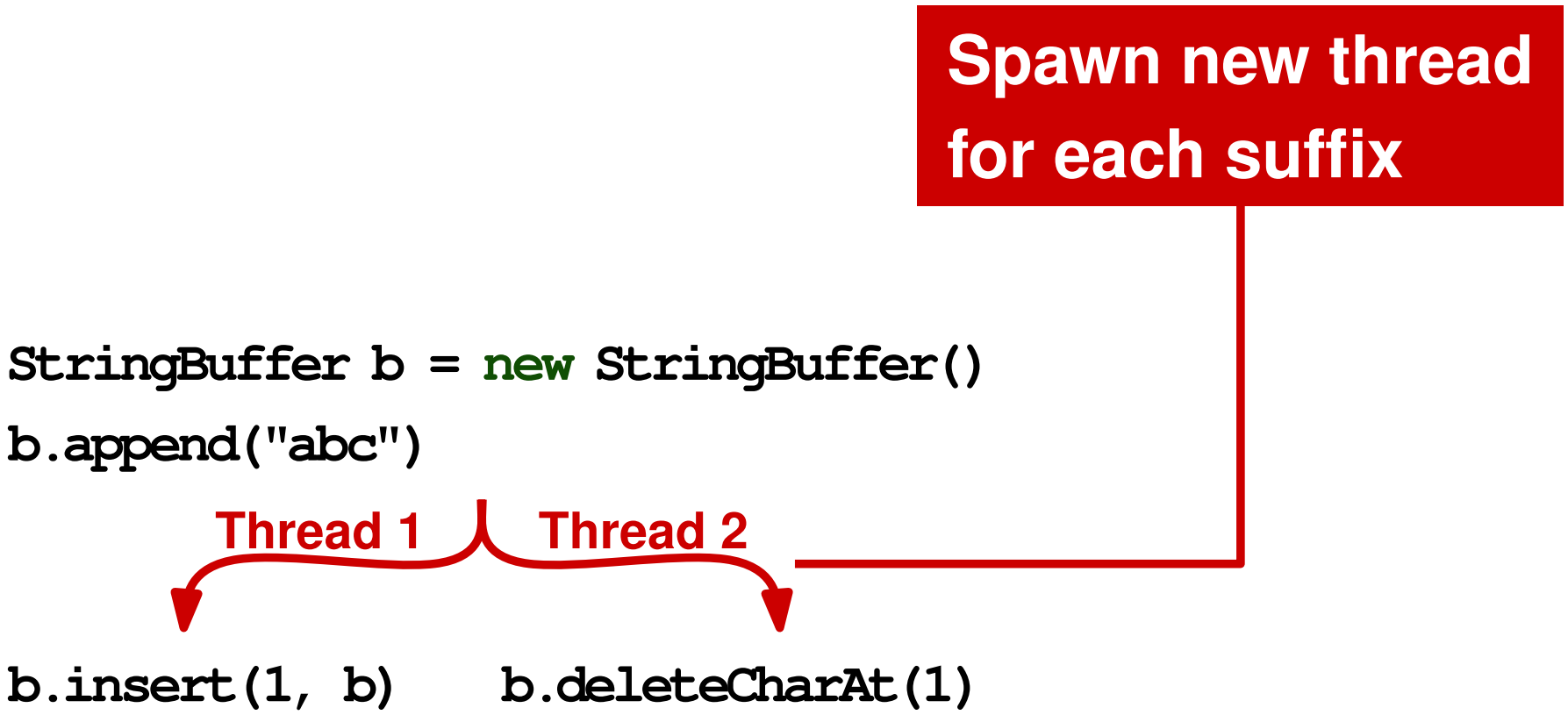
```
StringBuffer b = new StringBuffer()
b.append("abc")
```

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

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

Creating a Test

3. Prefix + two suffixes = test



Thread Safety Oracle

Does the test execution expose a thread safety violation?

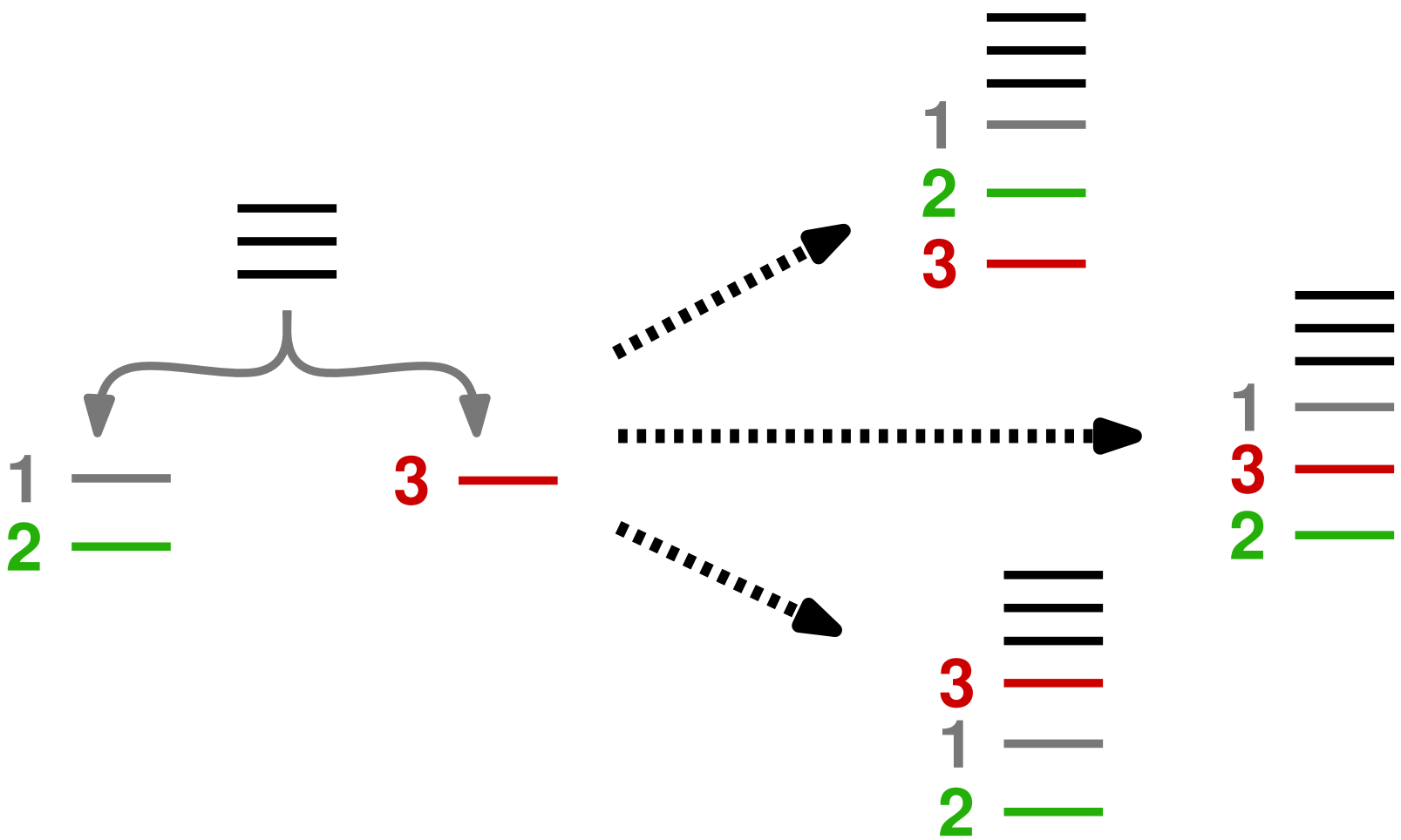
- Focus on exceptions and deadlocks
- Compare concurrent execution to linearizations

Linearizations

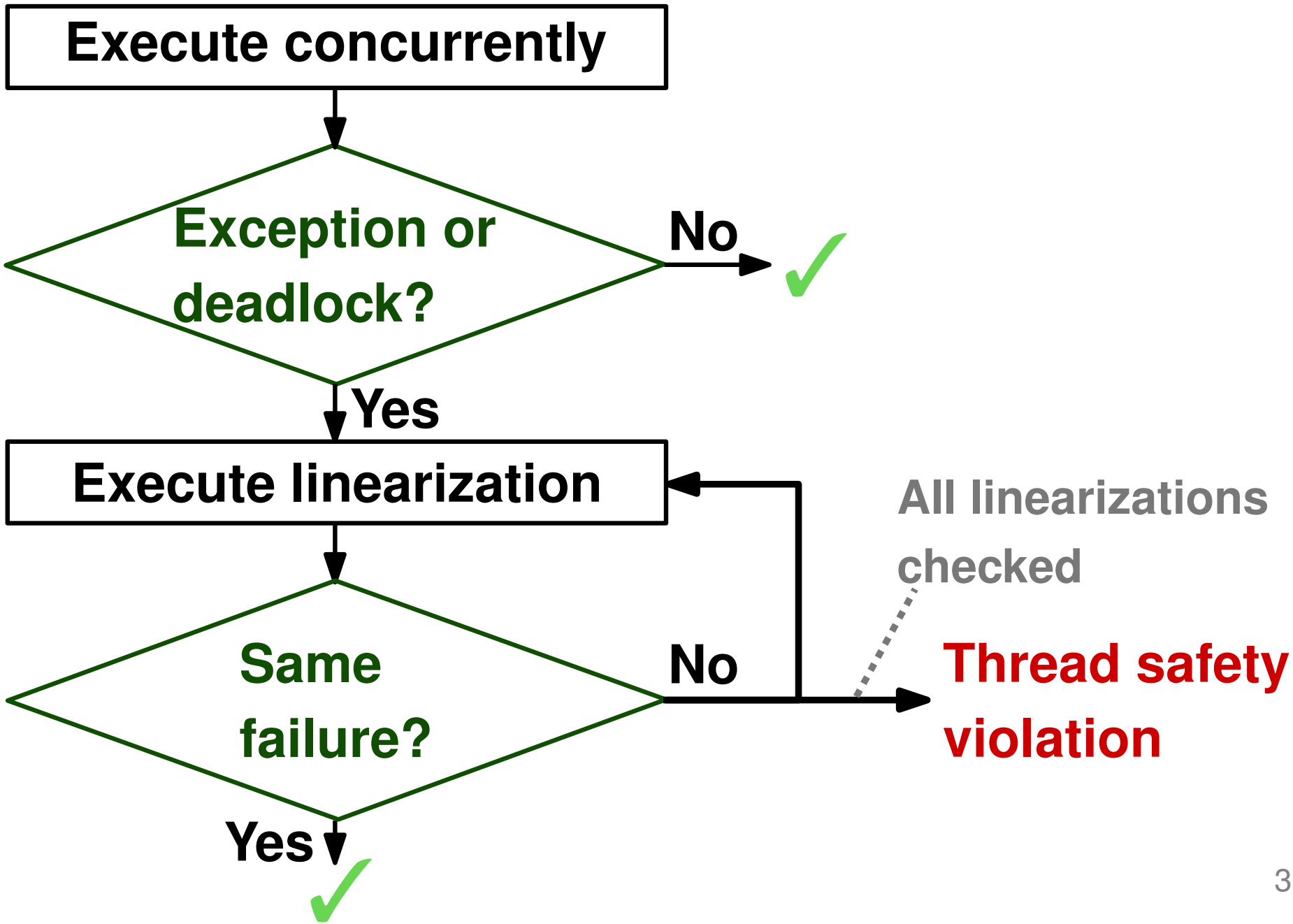
- Put all calls into one thread
- Preserve order of calls within a thread

Linearizations

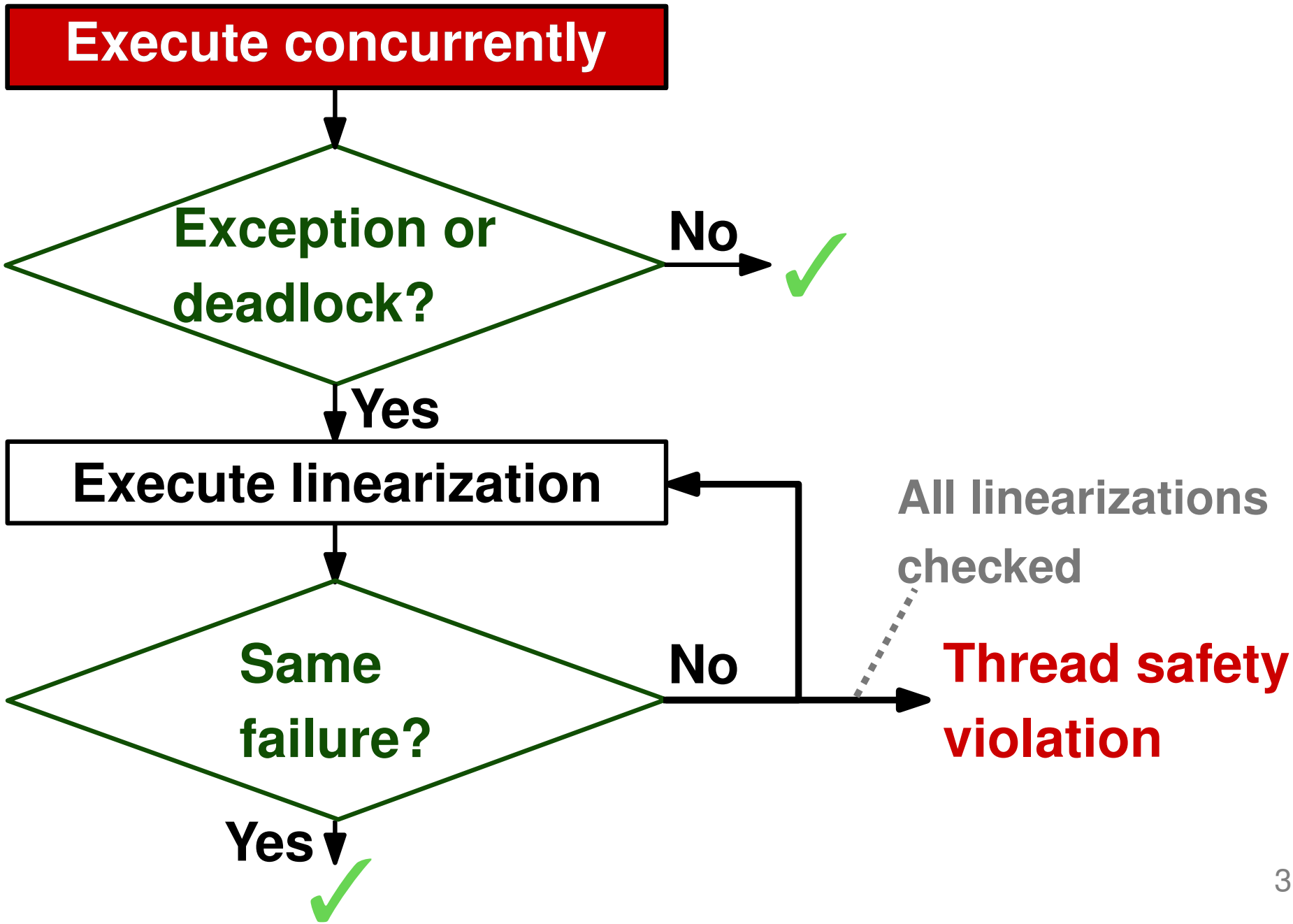
- Put all calls into one thread
- Preserve order of calls within a thread



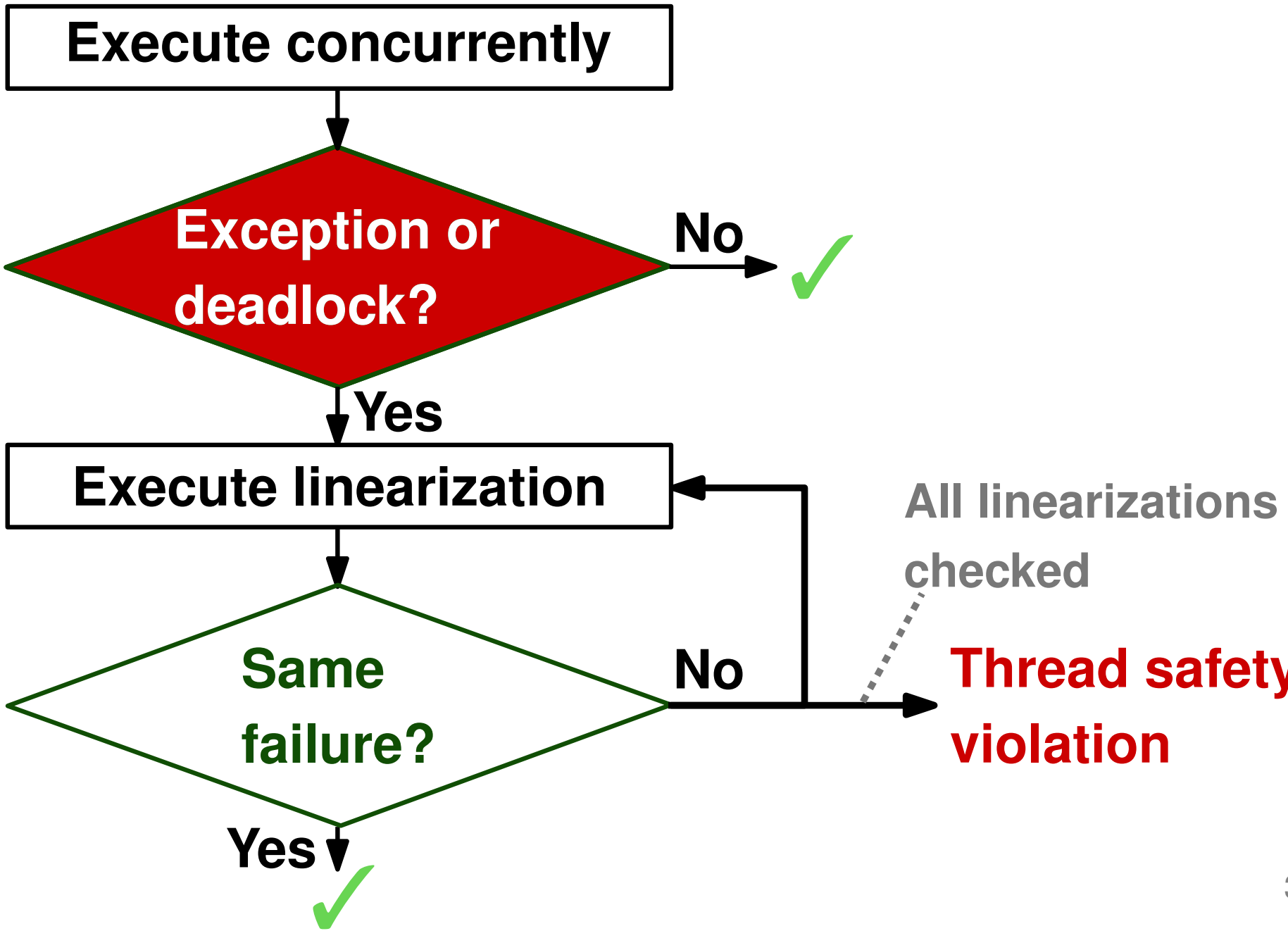
The Oracle


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

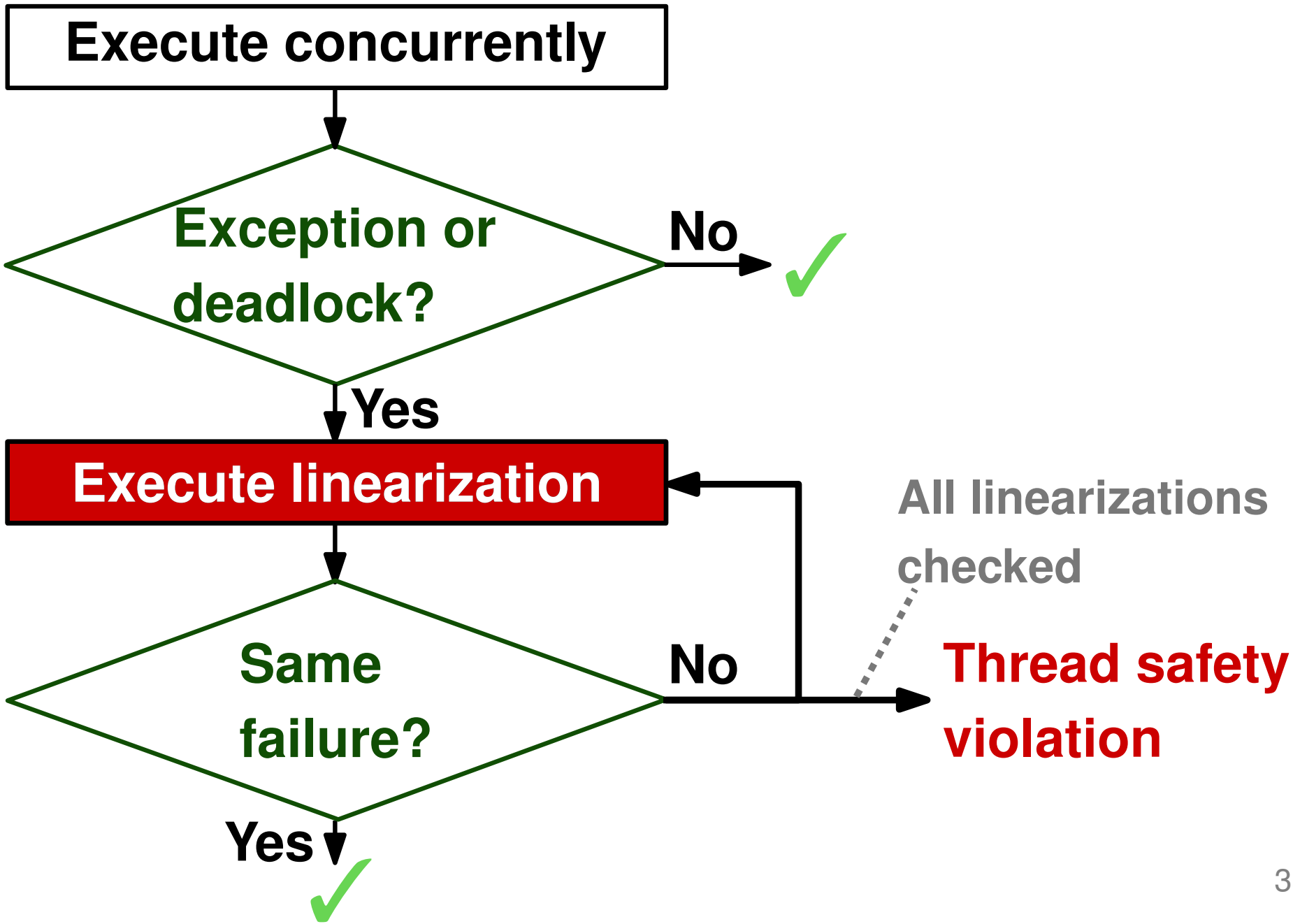
The Oracle


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

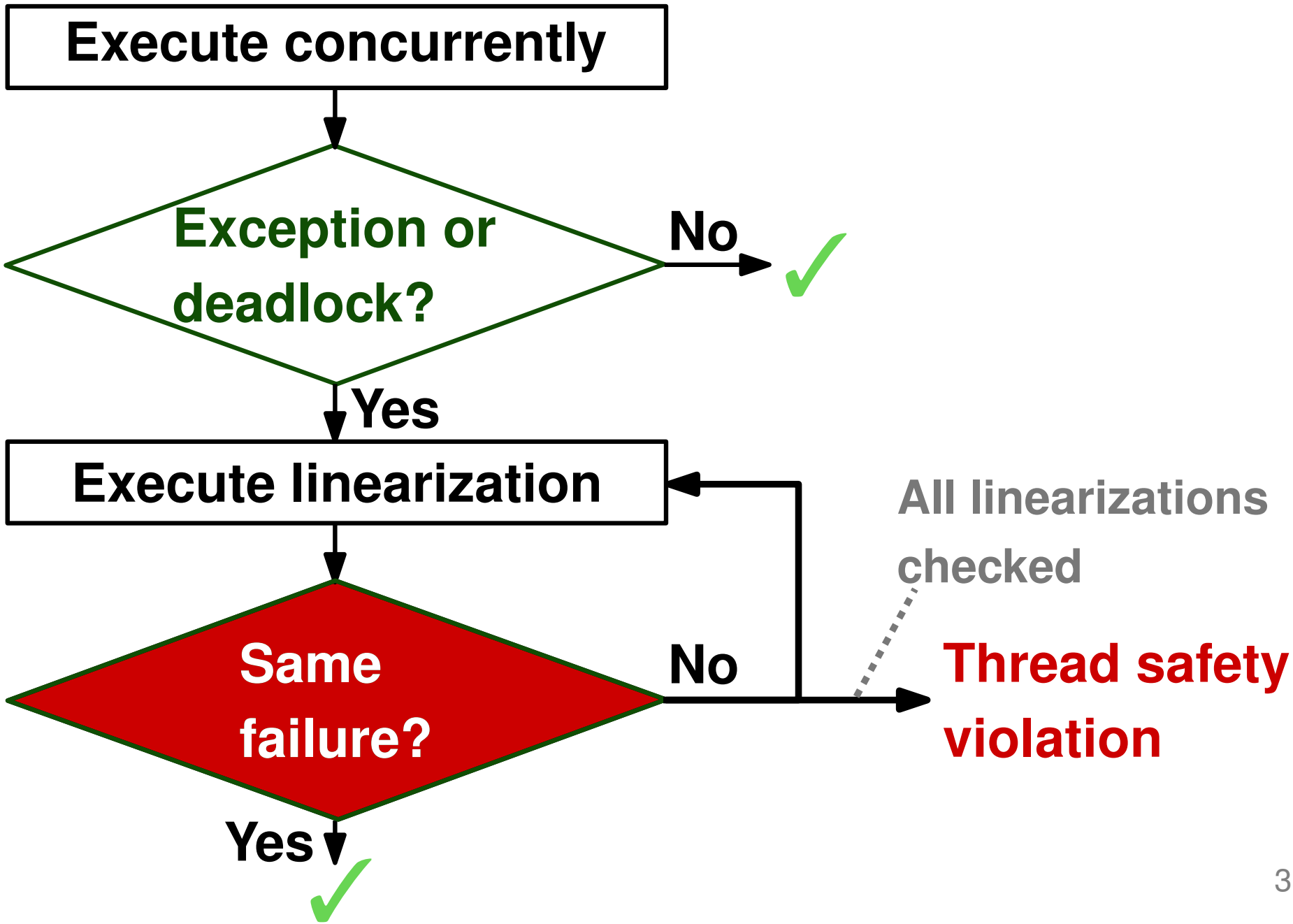
The Oracle


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

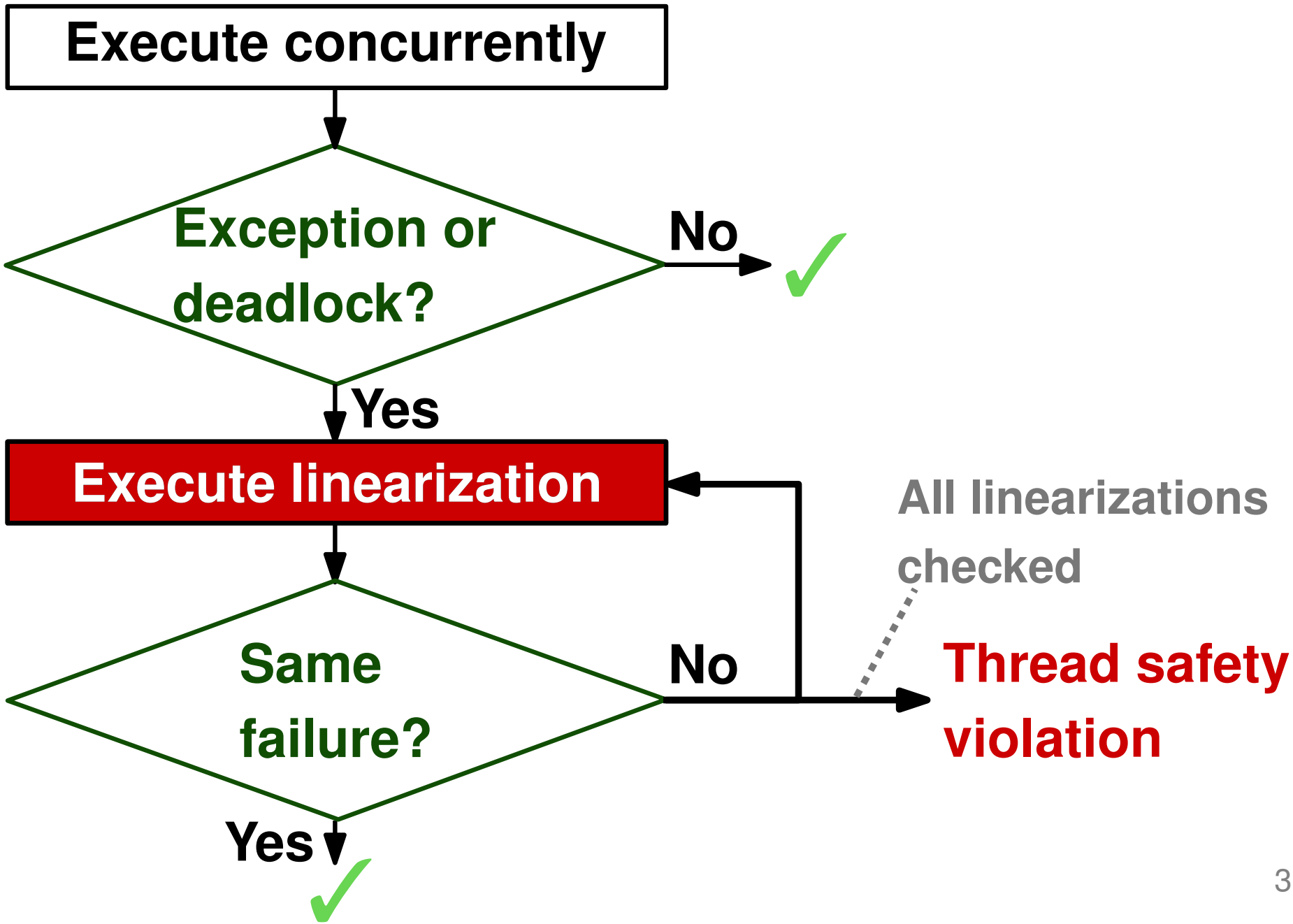
The Oracle


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

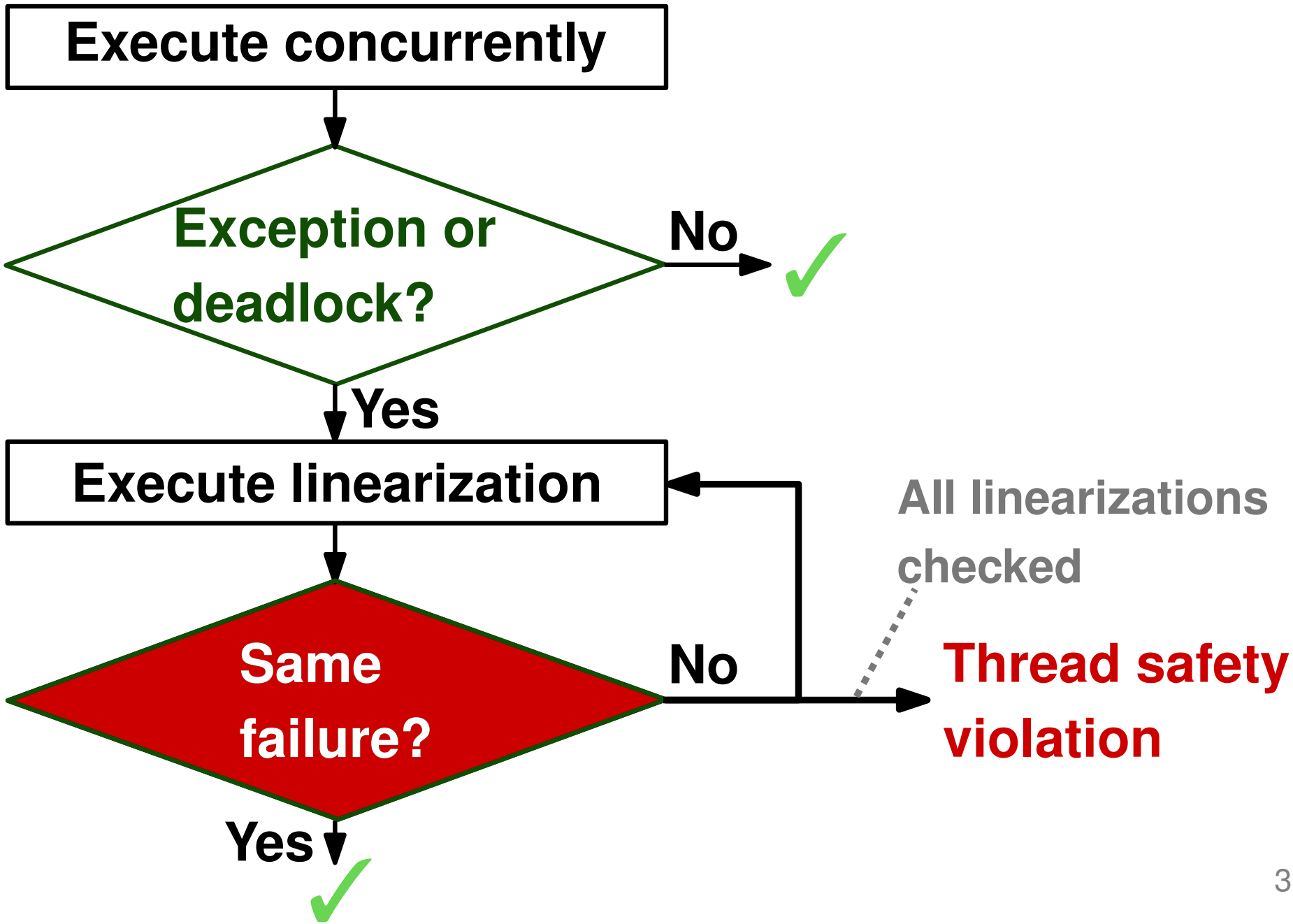
The Oracle


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

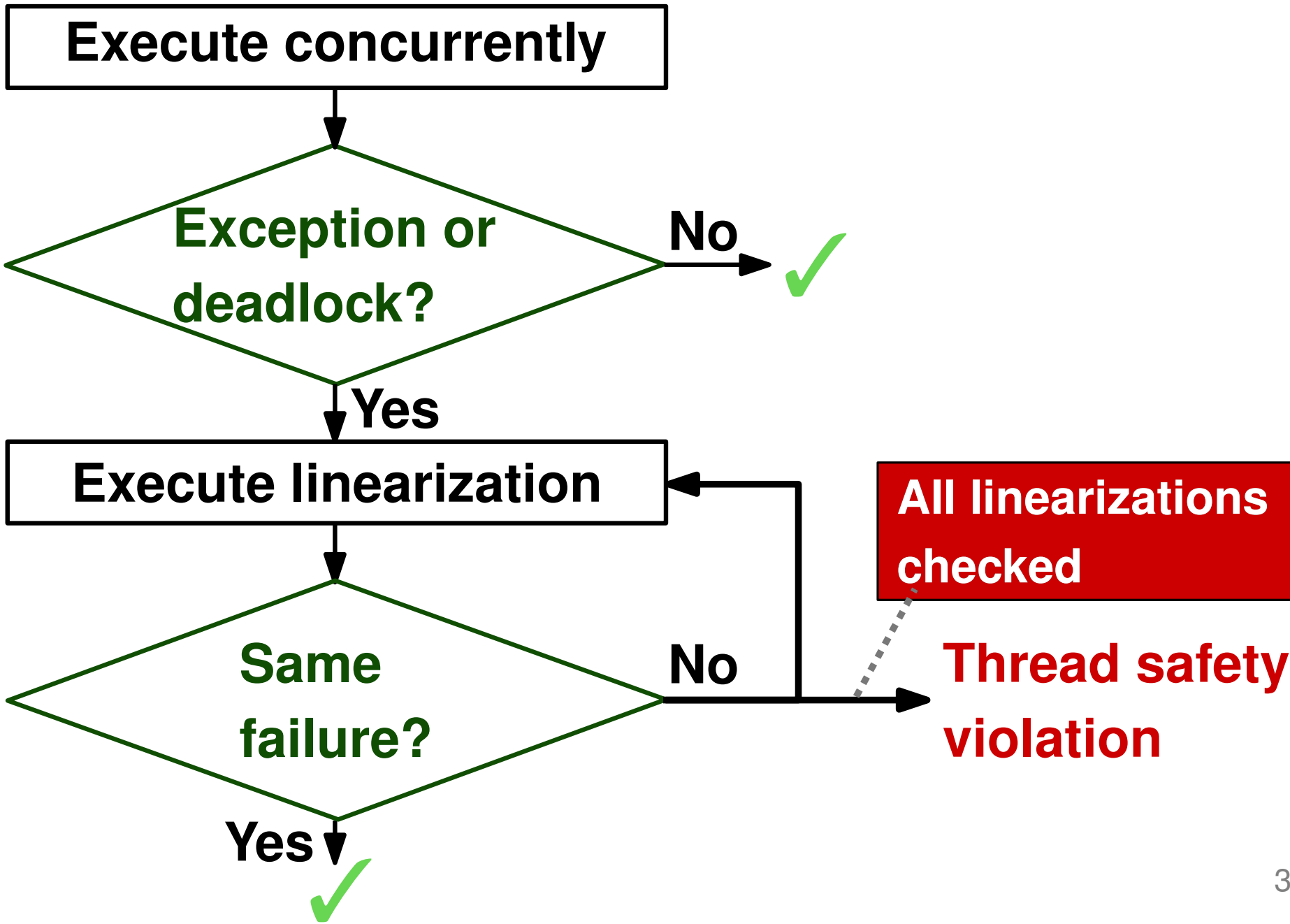
The Oracle


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

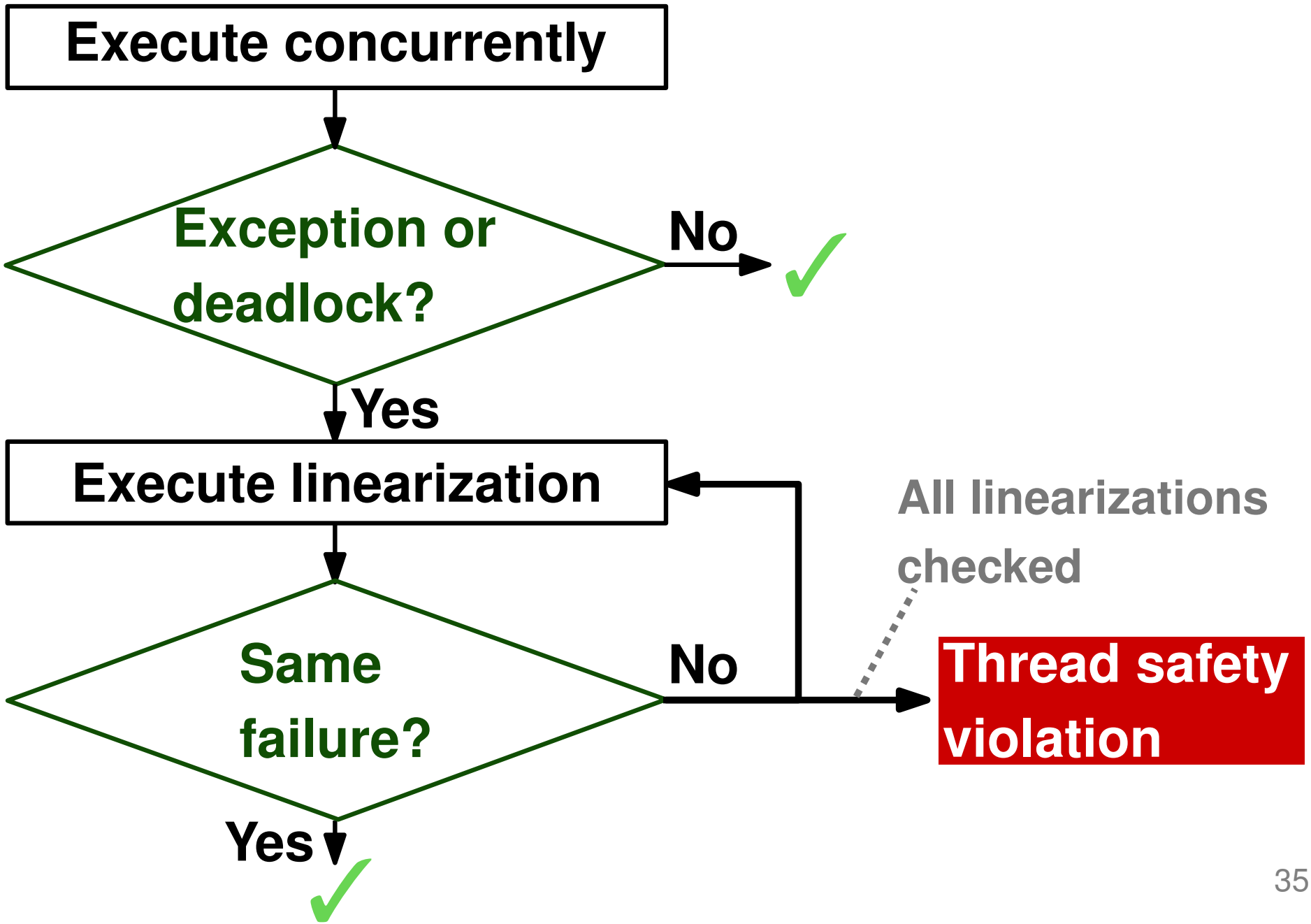
The Oracle


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

Example

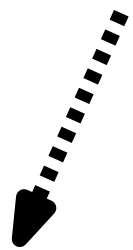
```
StringBuffer b = new StringBuffer()
b.append("abc")
```


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

Example

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


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



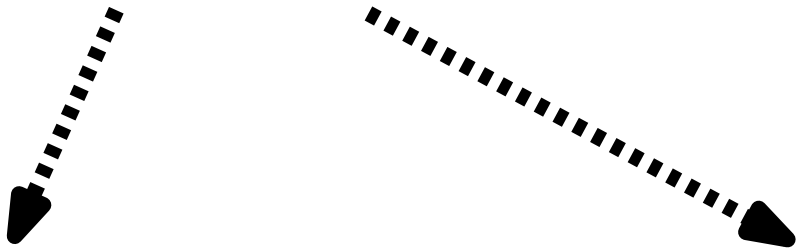
```
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

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")
```


```
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 & Results

- **Implemented for Java classes**
- **Applied to popular thread-safe classes from JDK, Apache libraries, etc.**
- **Found 15 concurrency bugs, including previously unknown problems in JDK**
- **Takes between several seconds and several hours (worst-case: 19 hours)**
  - Coverage-guided approach (ICSE'17): Worst-case time reduced to several minutes