

Synthesizing Tests for Detecting Atomicity Violations*

Malavika Samak and Murali Krishna Ramanathan

Indian Institute of Science, Bangalore

* Replication package evaluated

Introduction



Indian Institute of Science, Bangalore



Introduction

- ❖ Developers use multithreaded libraries as building blocks.

Introduction

- ❖ Developers use multithreaded libraries as building blocks.
- ❖ Do the client applications need to acquire additional locks to avoid *atomicity violations*?



Introduction

- ❖ Developers use multithreaded libraries as building blocks.
- ❖ Do the client applications need to acquire additional locks to avoid ***atomicity violations***?
- ❖ If YES,
 - ❖ what locks need to be acquired?
 - ❖ which method invocations require these acquisitions?

Introduction

- ❖ Developers use multithreaded libraries as building blocks.
- ❖ Do the client applications need to acquire additional locks to avoid ***atomicity violations***?
- ❖ If YES,
 - ❖ what locks need to be acquired?
 - ❖ which method invocations require these acquisitions?

Dynamic Atomicity Violation Detection



What is an atomicity violation?

T_1



Previous (P)



Current (C)

T_2



Remote (R)

1. No atomicity violation
2. No intervening access between P & C

What is an atomicity violation?

T_1



Previous (P)

T_2



Remote (R)



Current (C)

1. Atomicity violation
2. R intervenes between P & C

What is an atomicity violation?

T₁



T₂



1. Atomicity violation
2. R intervenes between P & C

Dynamic Atomicity Violation Detection



Indian Institute of Science, Bangalore



Dynamic Atomicity Violation Detection

Library



Indian Institute of Science, Bangalore

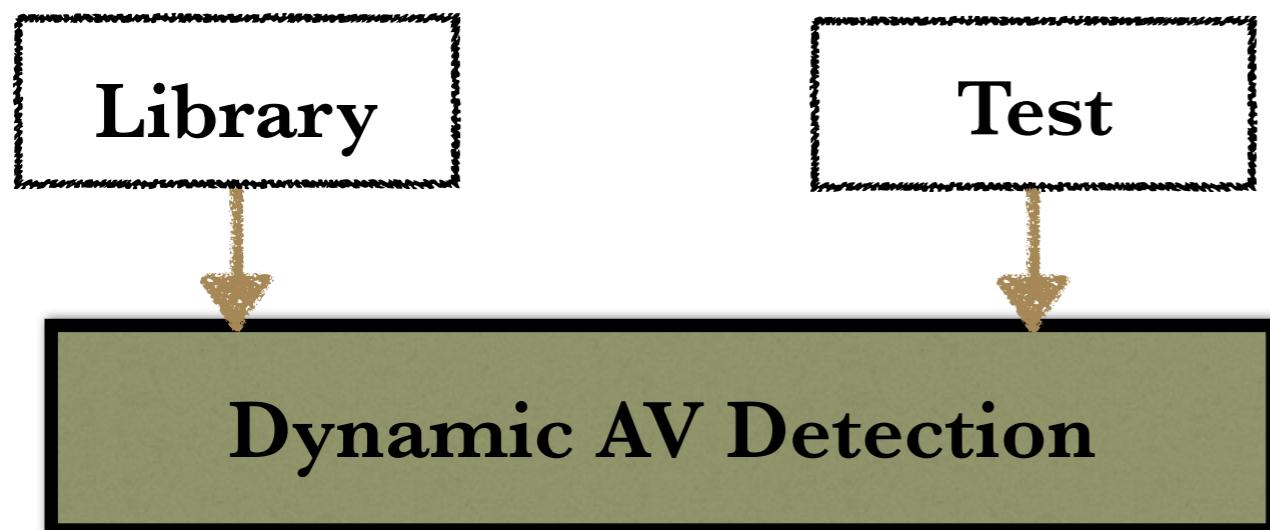


Dynamic Atomicity Violation Detection

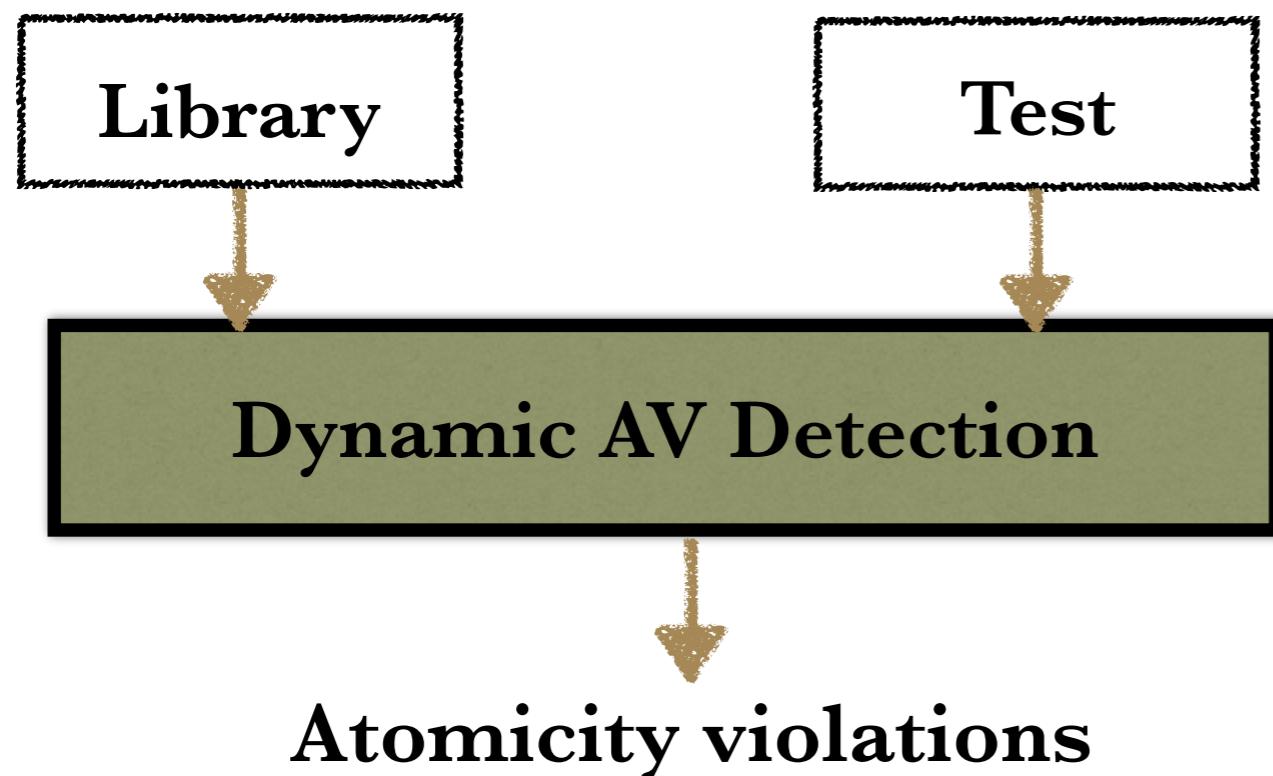
Library

Test

Dynamic Atomicity Violation Detection

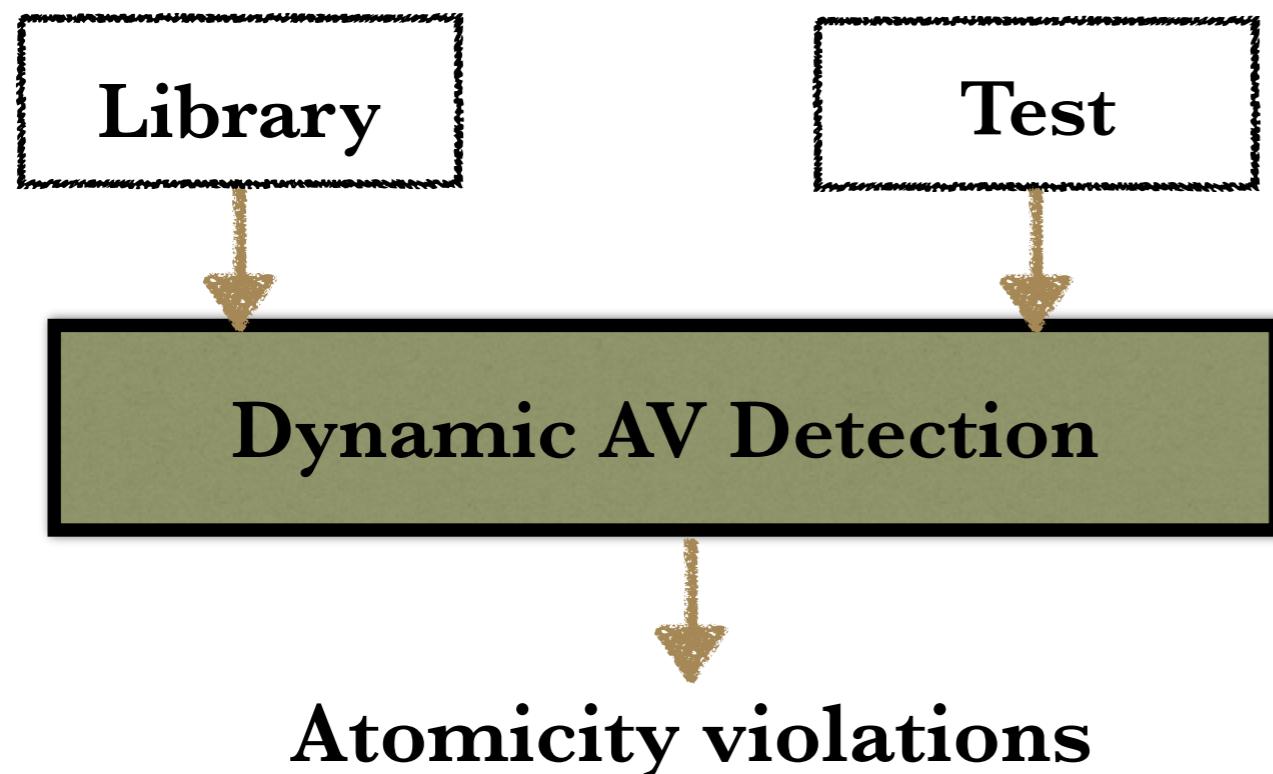


Dynamic Atomicity Violation Detection



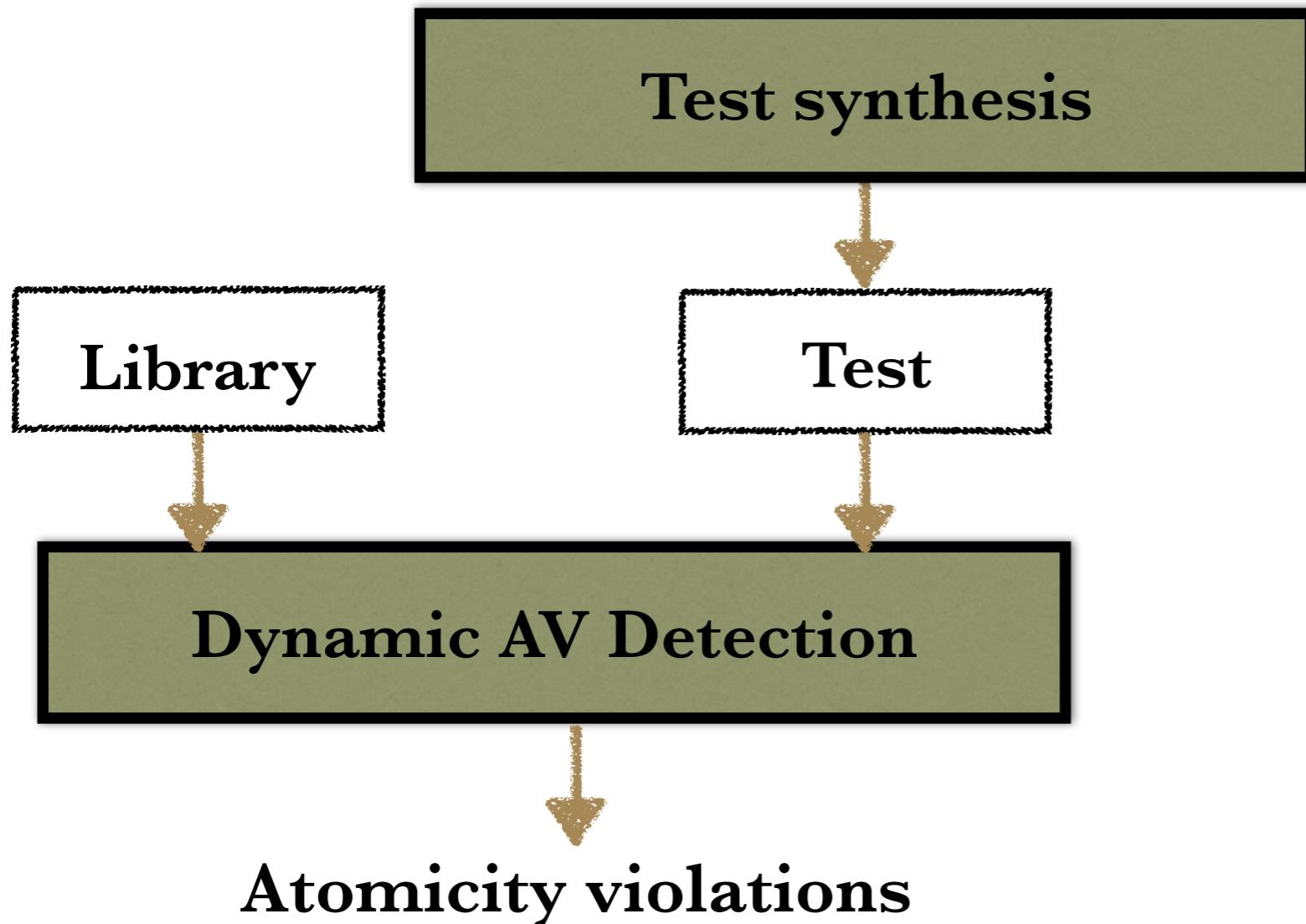
- Atomizer, POPL'04
- AtomFuzzer, FSE'08
- Velodrome, PLDI'08
- CTrigger, ASPLOS'09
- DoubleChecker, PLDI'14

Dynamic Atomicity Violation Detection

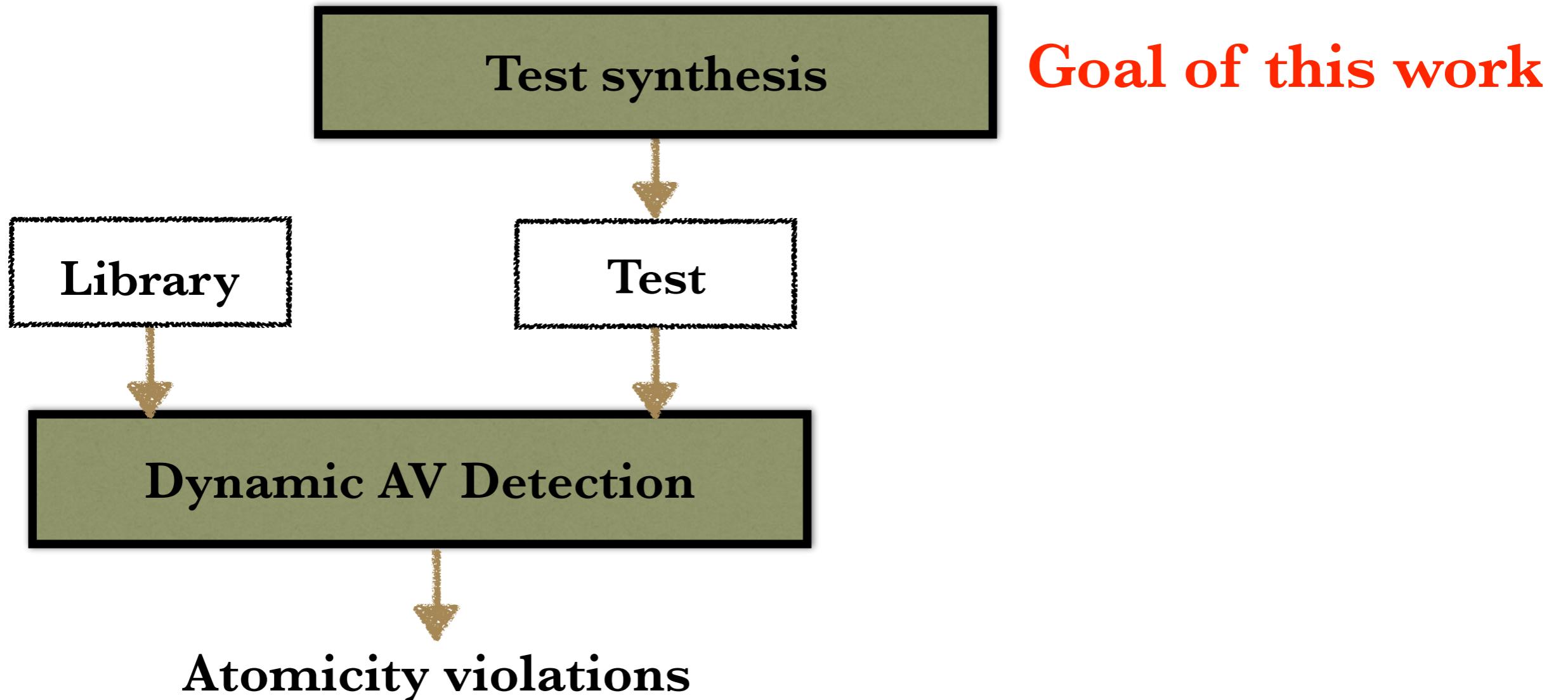


- Significant manual effort
- Higher false negatives

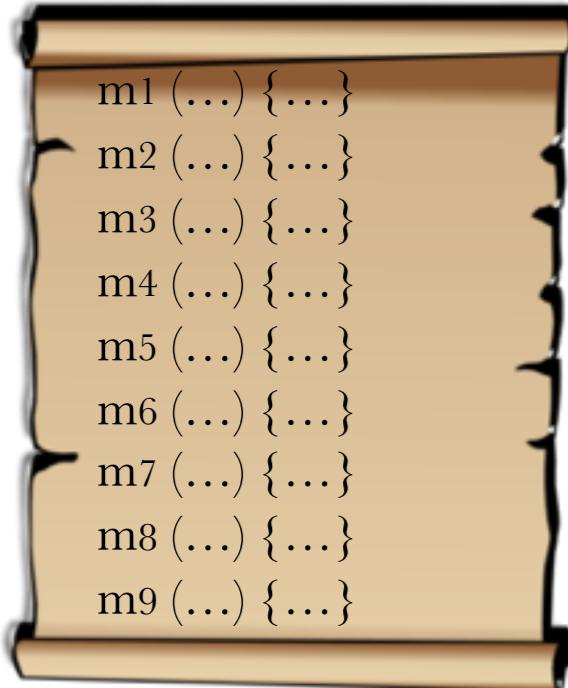
Dynamic Atomicity Violation Detection



Dynamic Atomicity Violation Detection



Randomized Test Generation

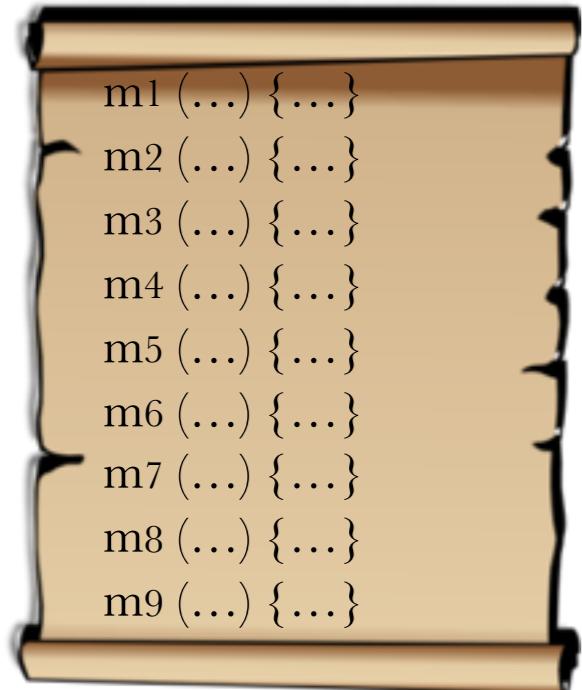


Library



Possible Parameters

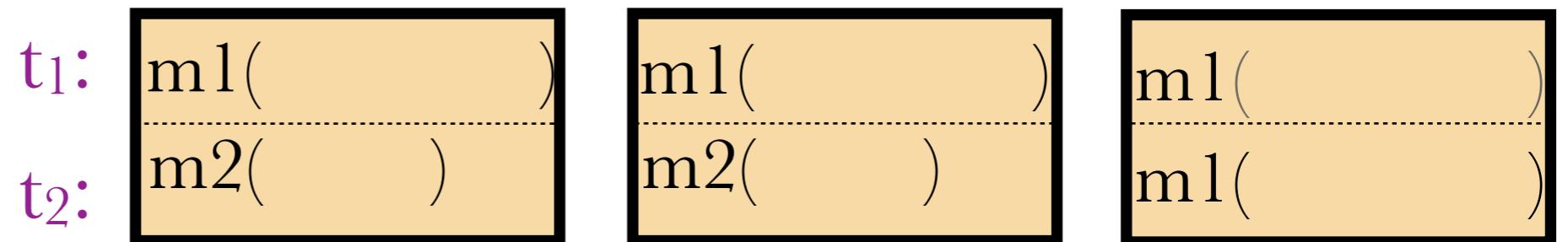
Randomized Test Generation



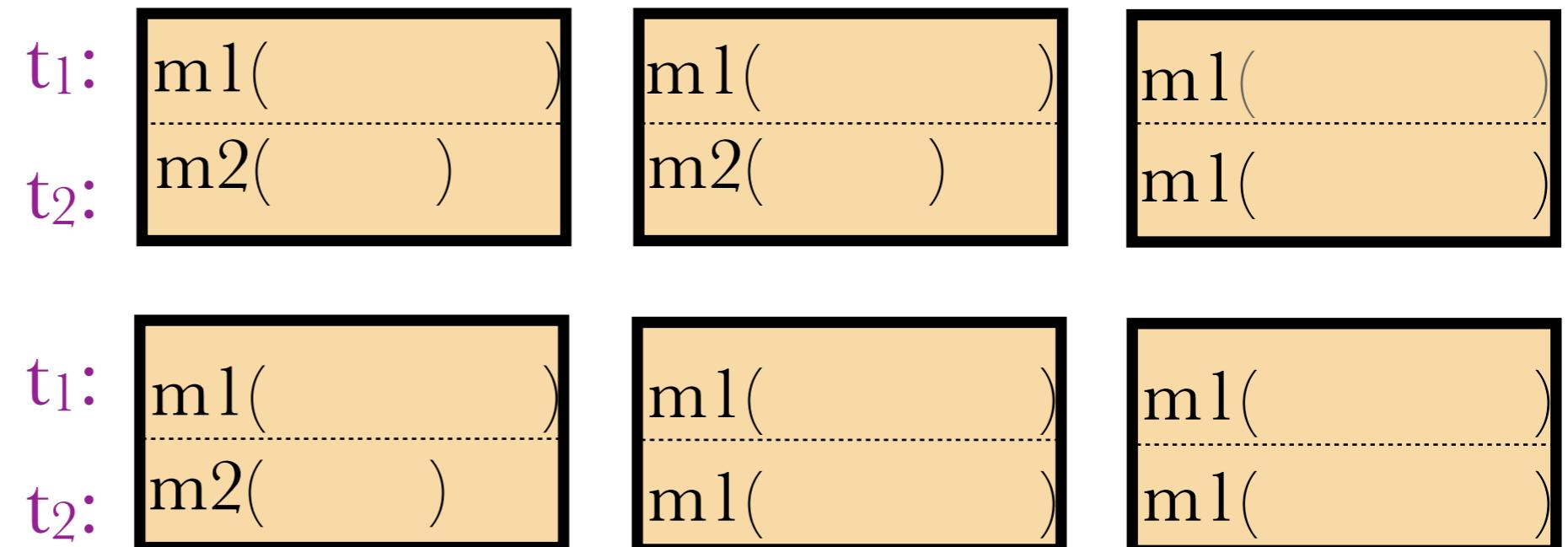
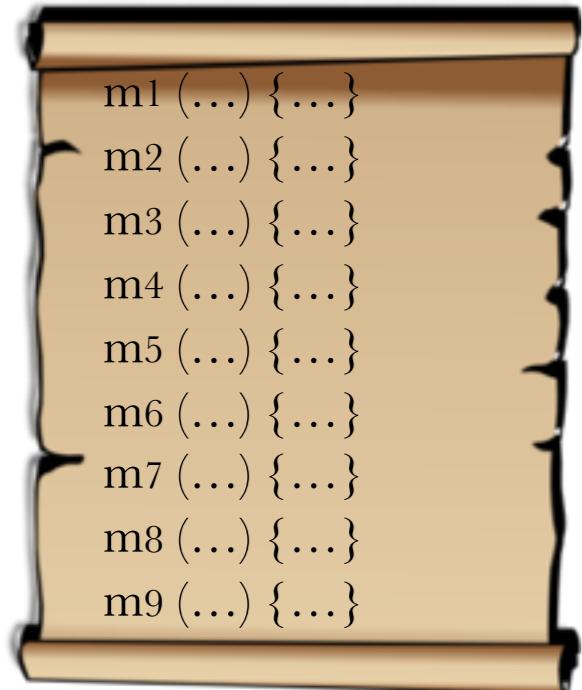
Library



Possible Parameters



Randomized Test Generation

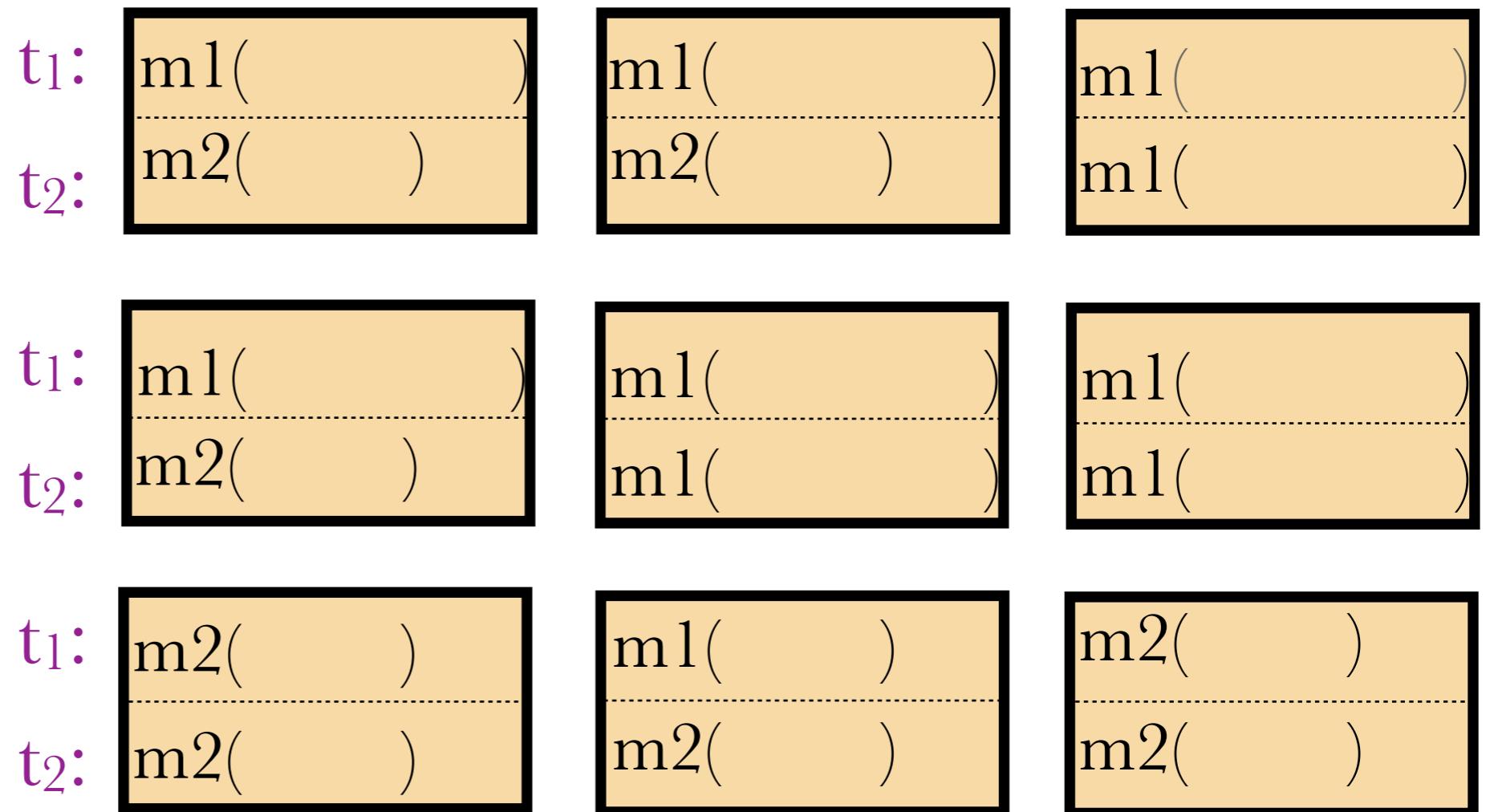
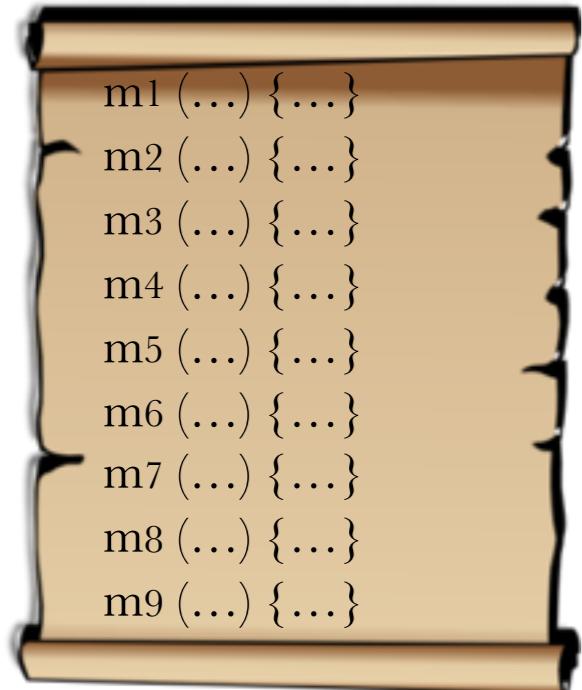


Library



Possible Parameters

Randomized Test Generation

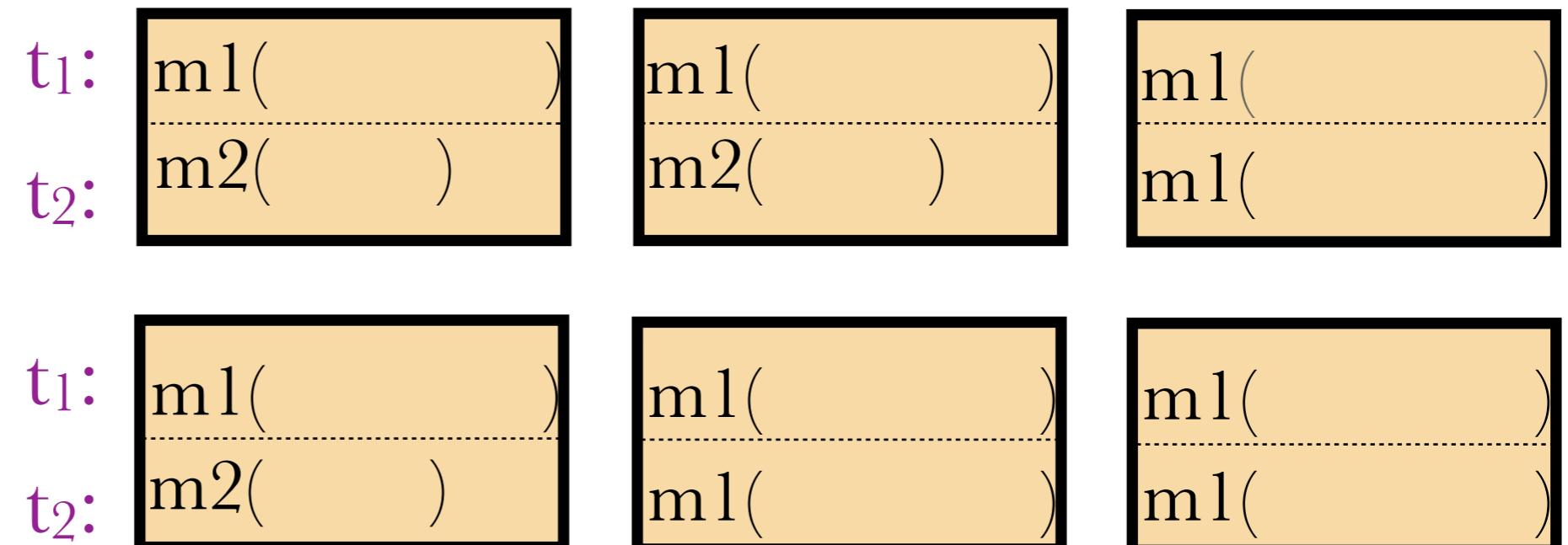
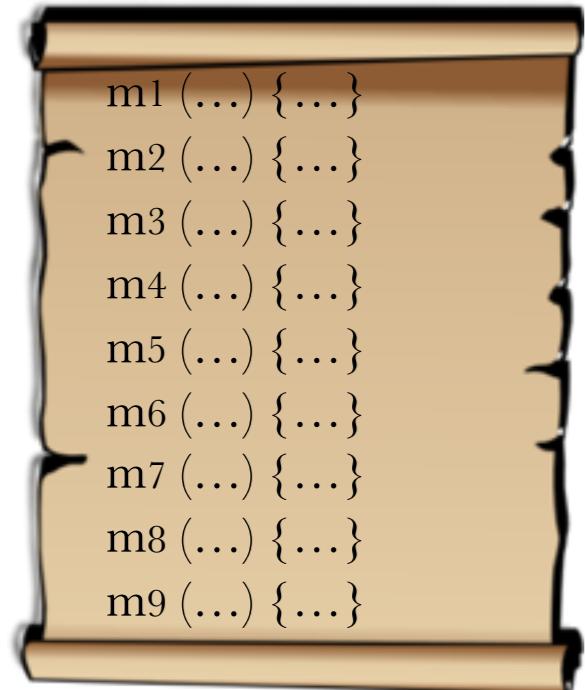


Library



Possible Parameters

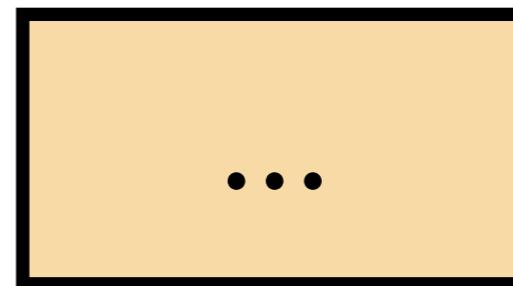
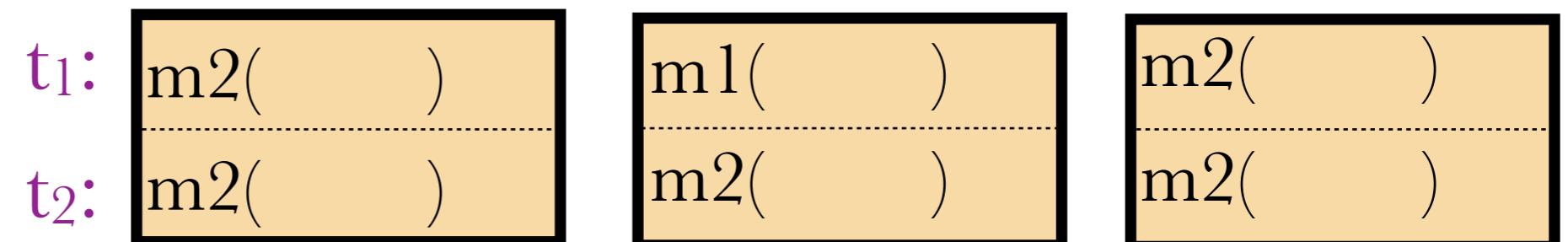
Randomized Test Generation



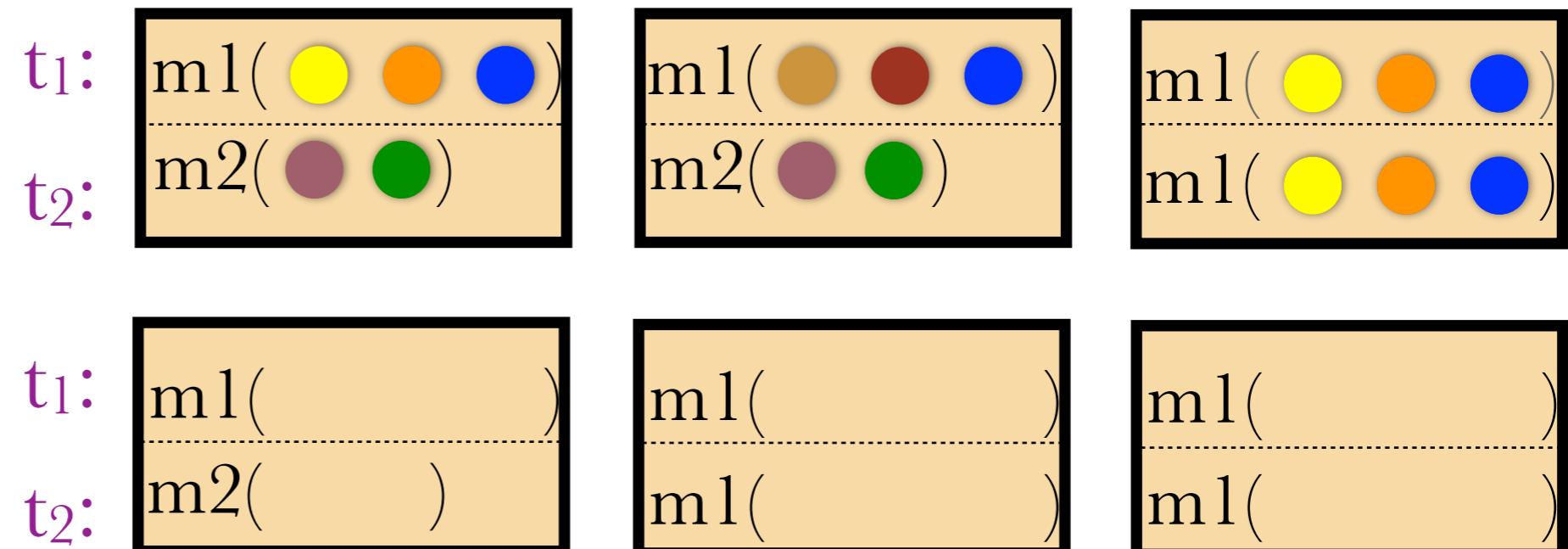
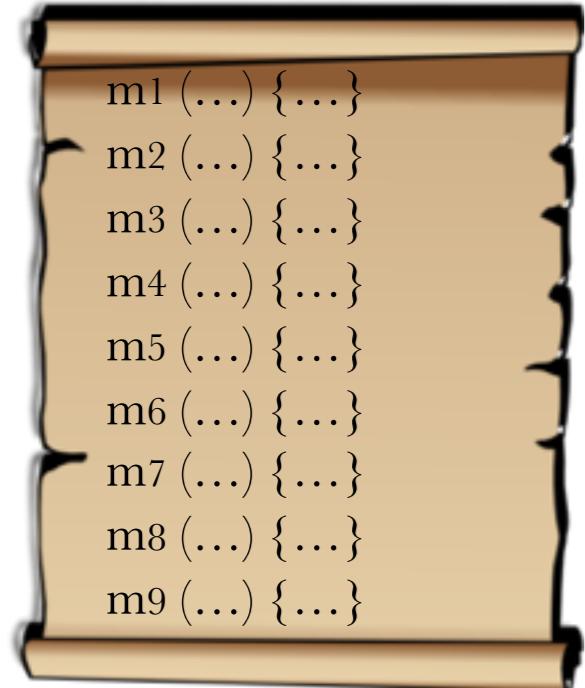
Library



Possible Parameters



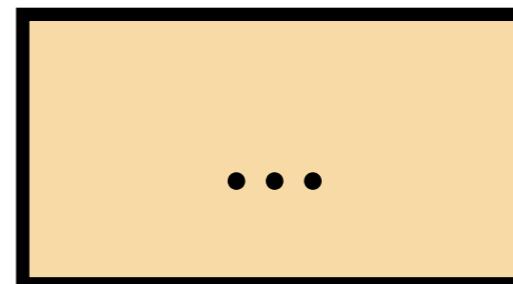
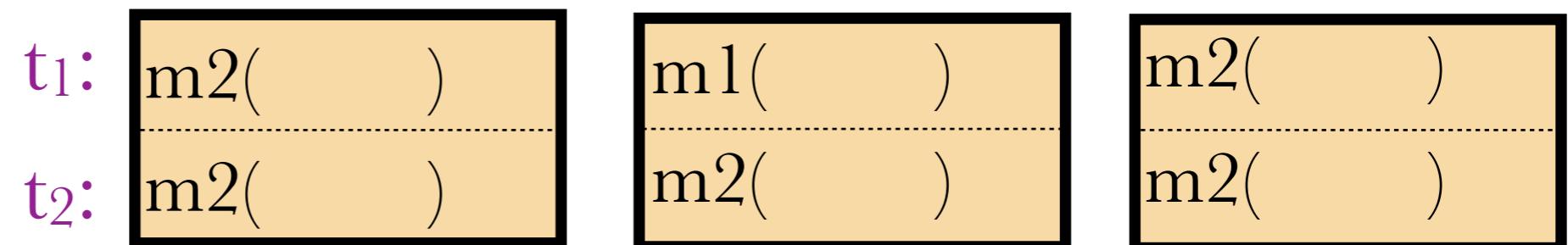
Randomized Test Generation



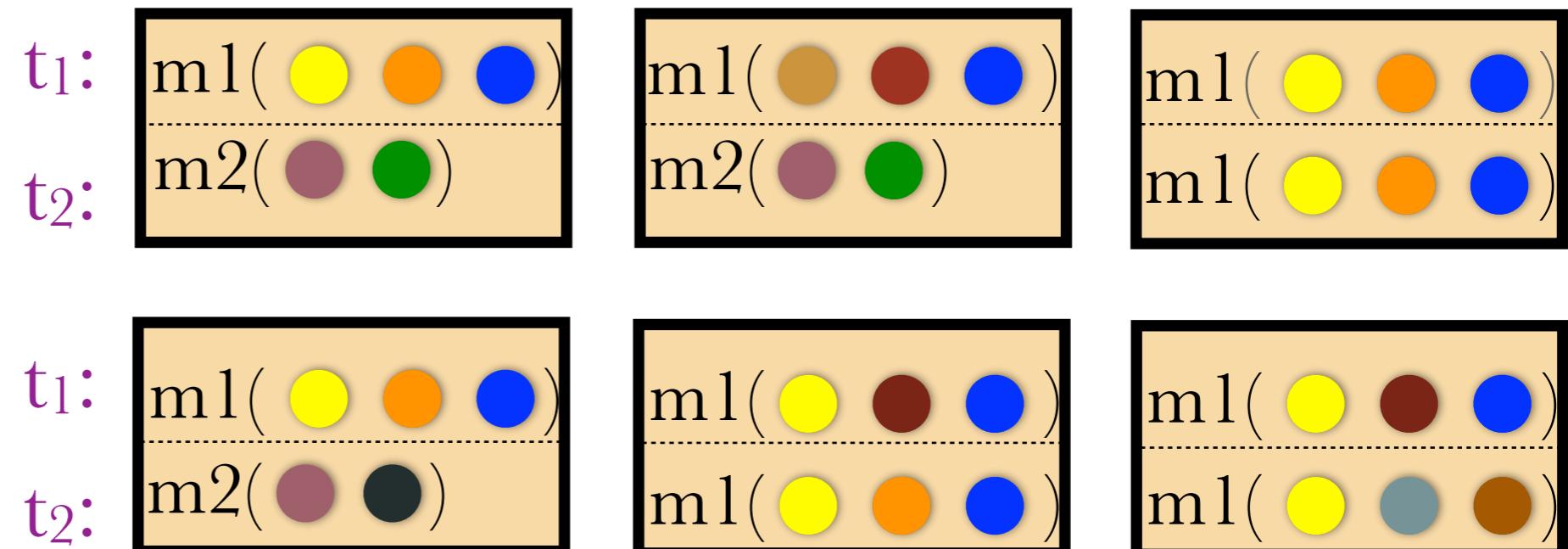
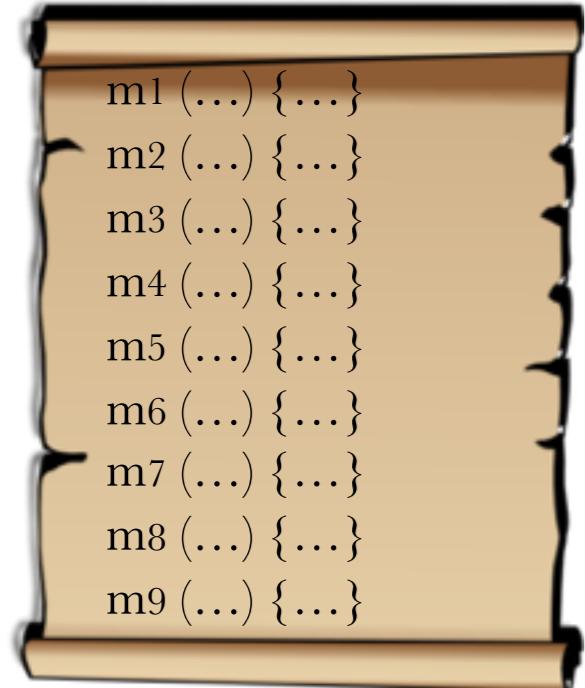
Library



Possible Parameters



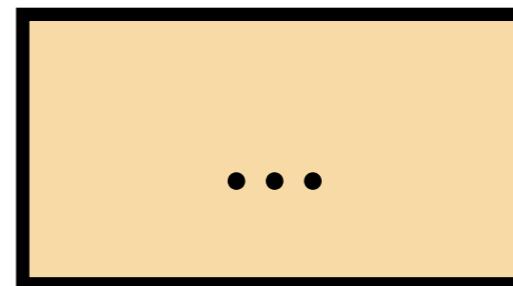
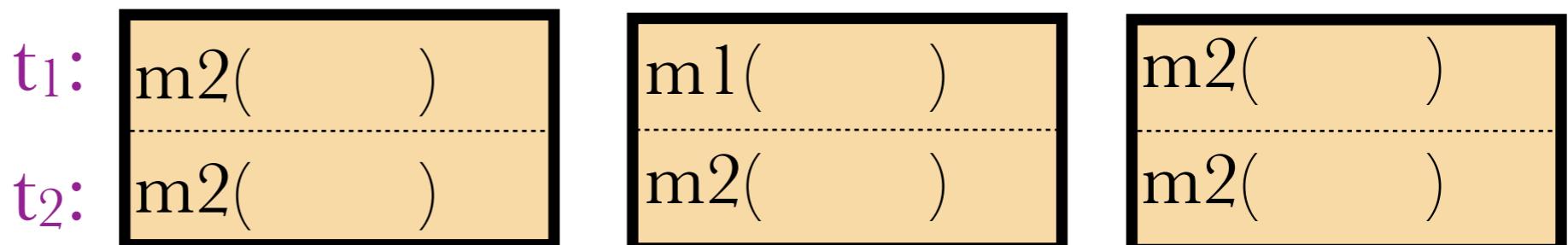
Randomized Test Generation



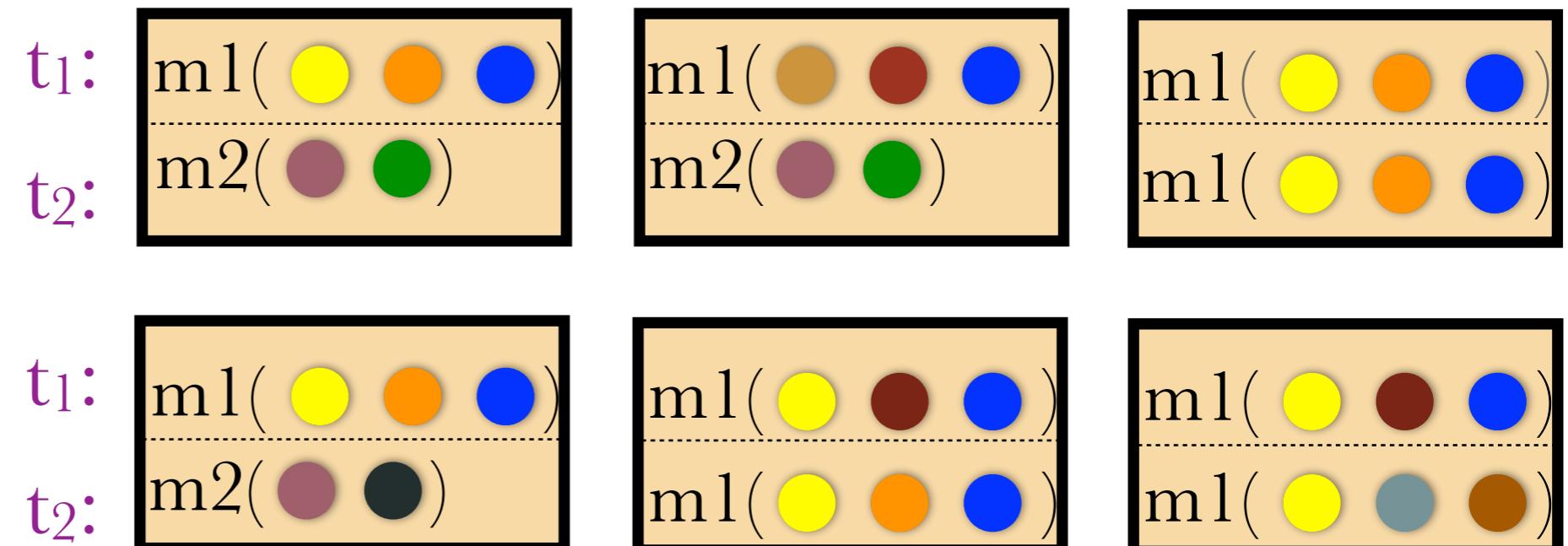
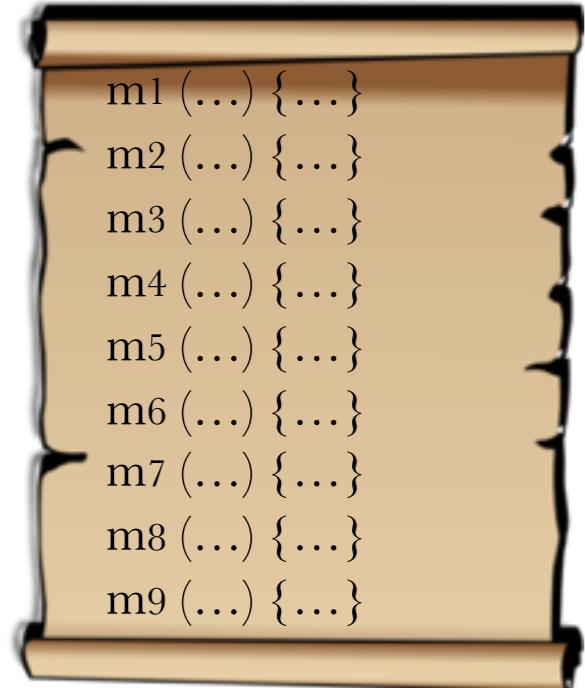
Library



Possible Parameters



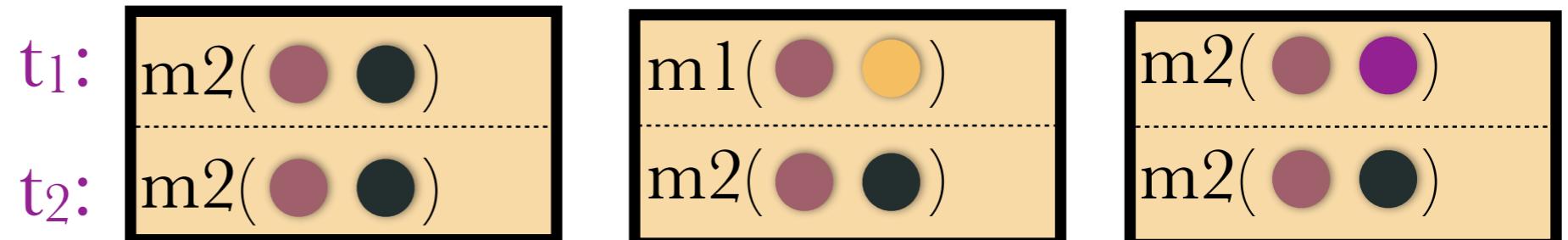
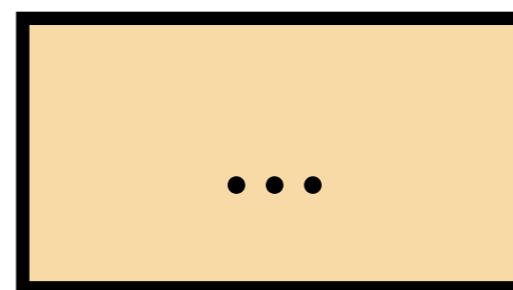
Randomized Test Generation



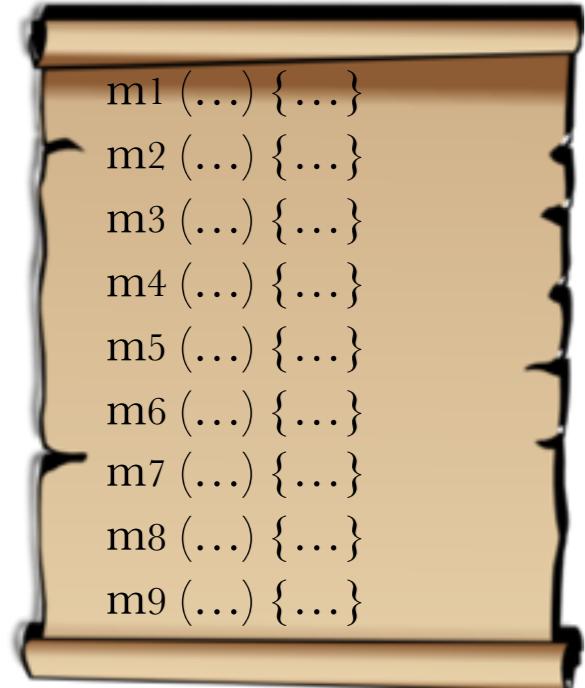
Library



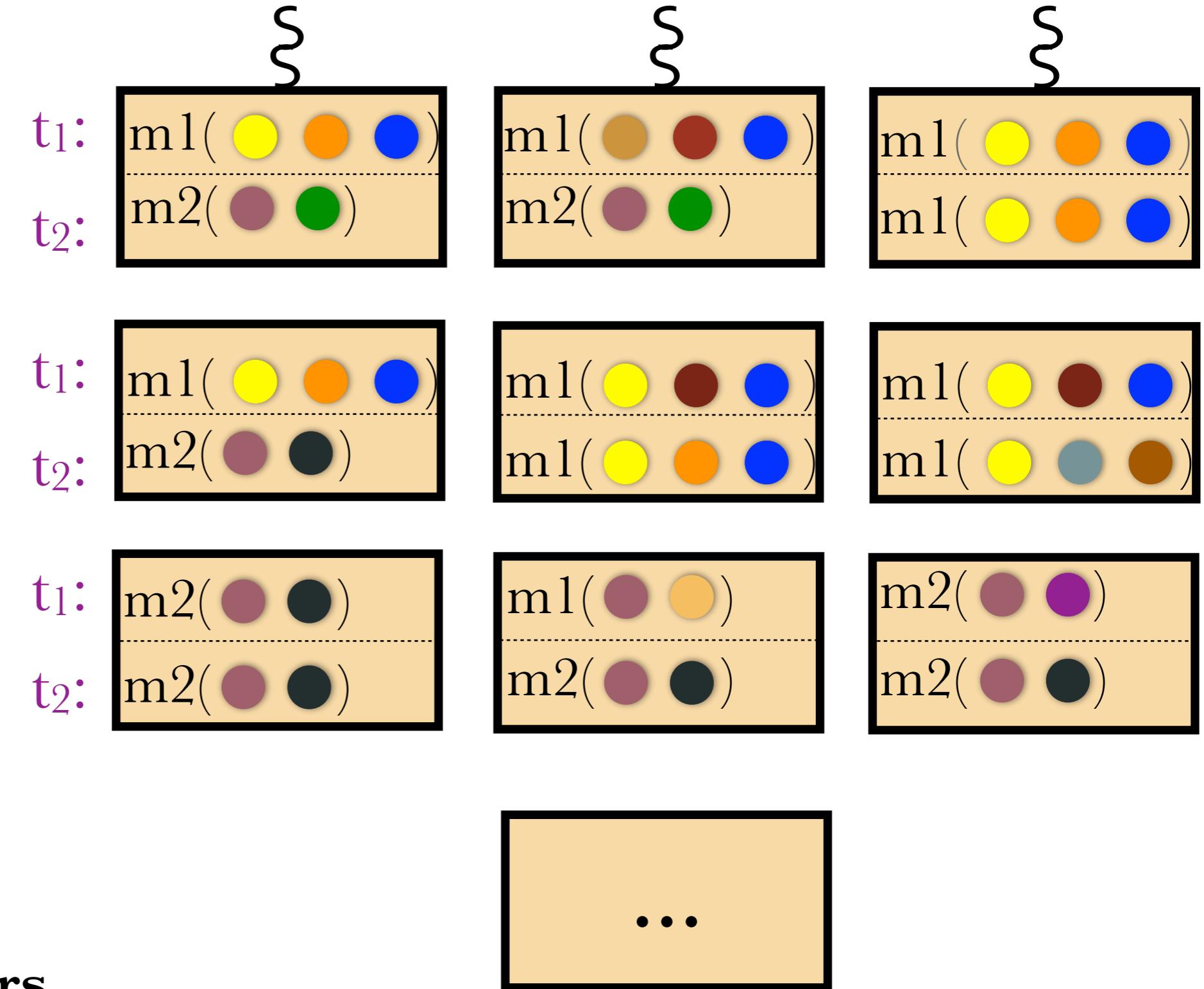
Possible Parameters



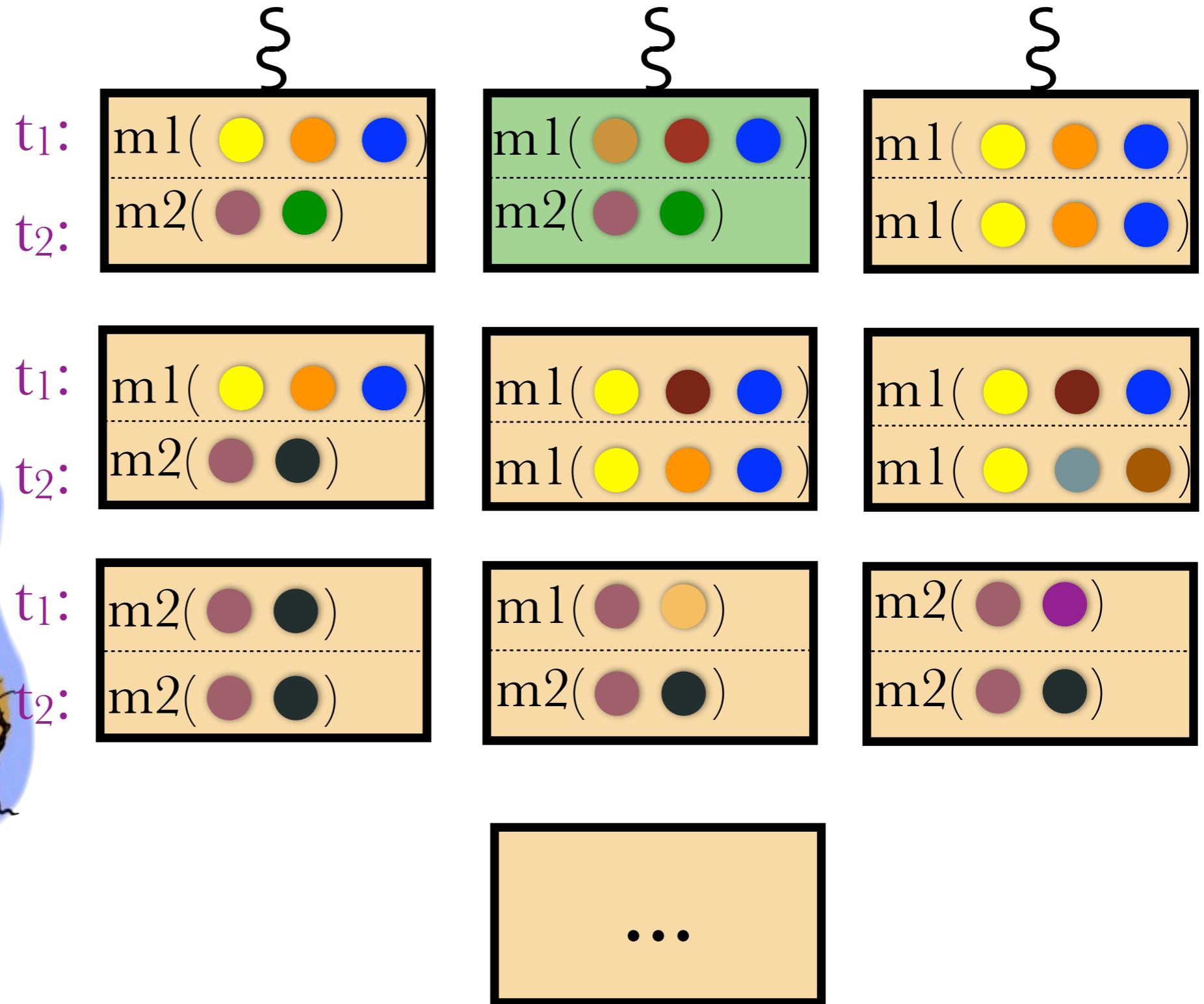
Randomized Test Generation



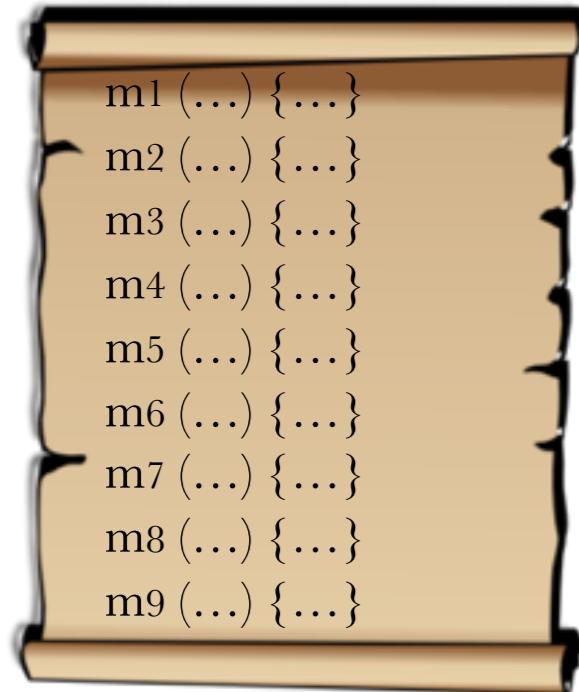
Library
Possible Parameters



Randomized Test Generation



Targeted Test Synthesis

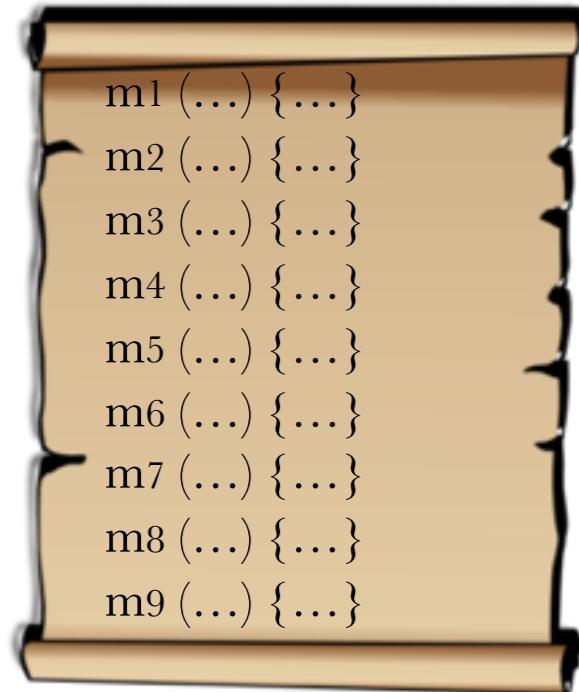


Library



Possible Parameters

Targeted Test Synthesis



Library

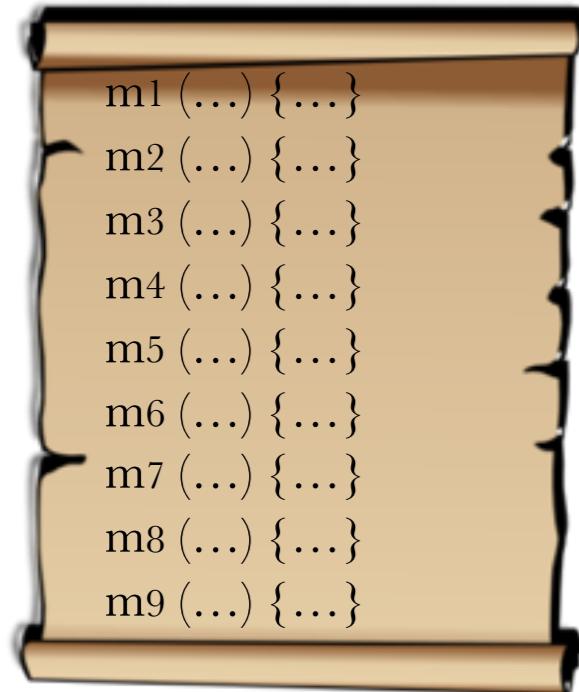


Possible Parameters

t_1

t_2

Targeted Test Synthesis



Library



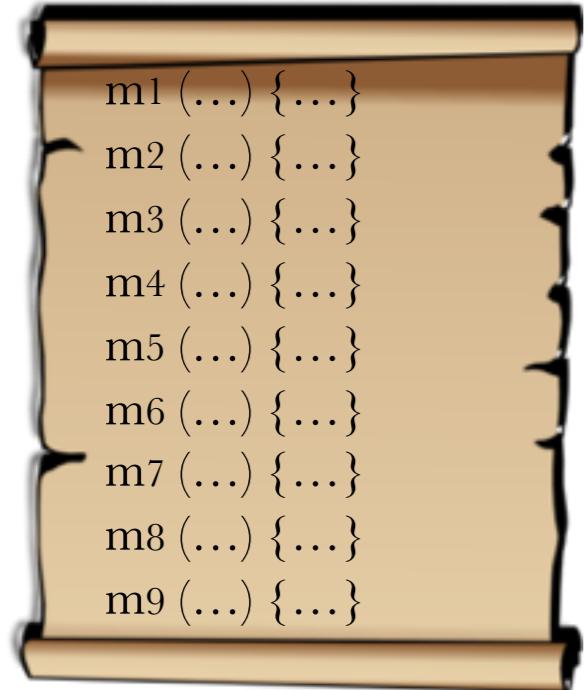
Possible Parameters

m1()

t_1

t_2

Targeted Test Synthesis



Library

$m1(\quad)$ $m4(\quad)$

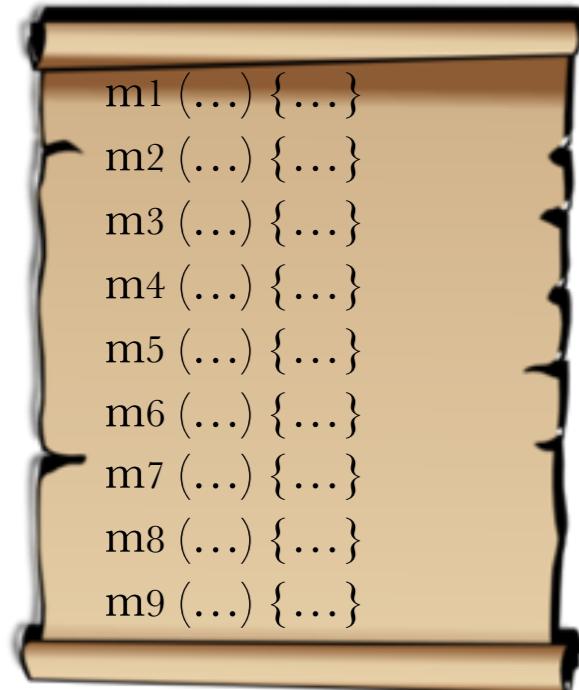
t_1

t_2



Possible Parameters

Targeted Test Synthesis



Library



Possible Parameters

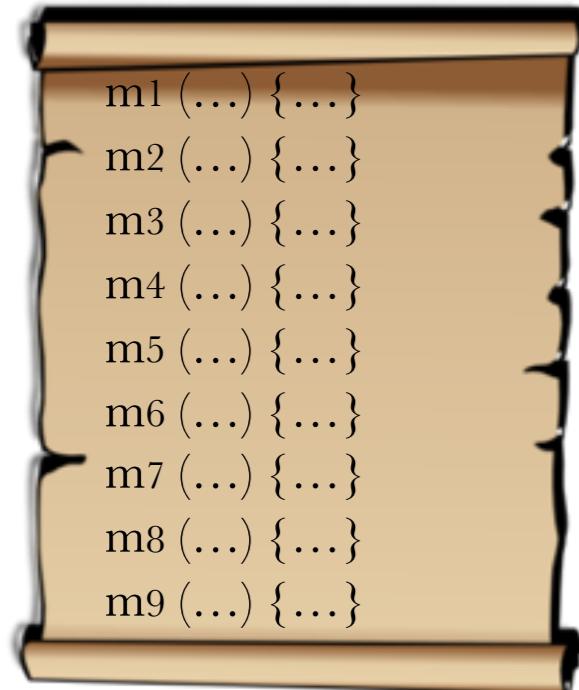
t_1

m1(**)**

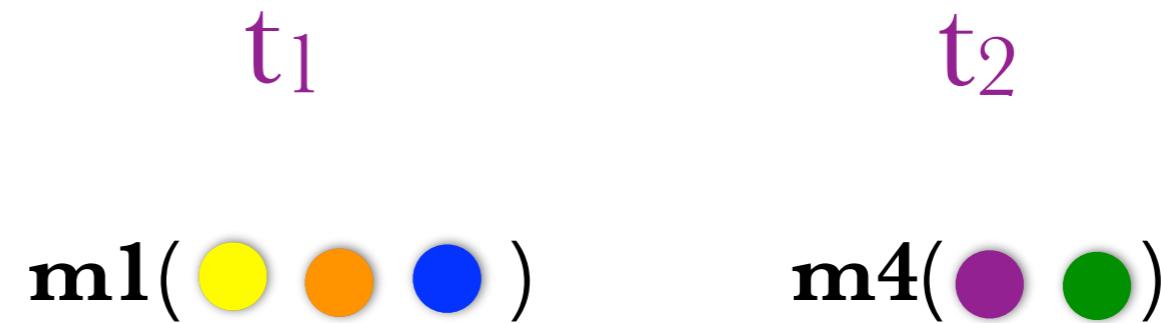
t_2

m4(**)**

Targeted Test Synthesis

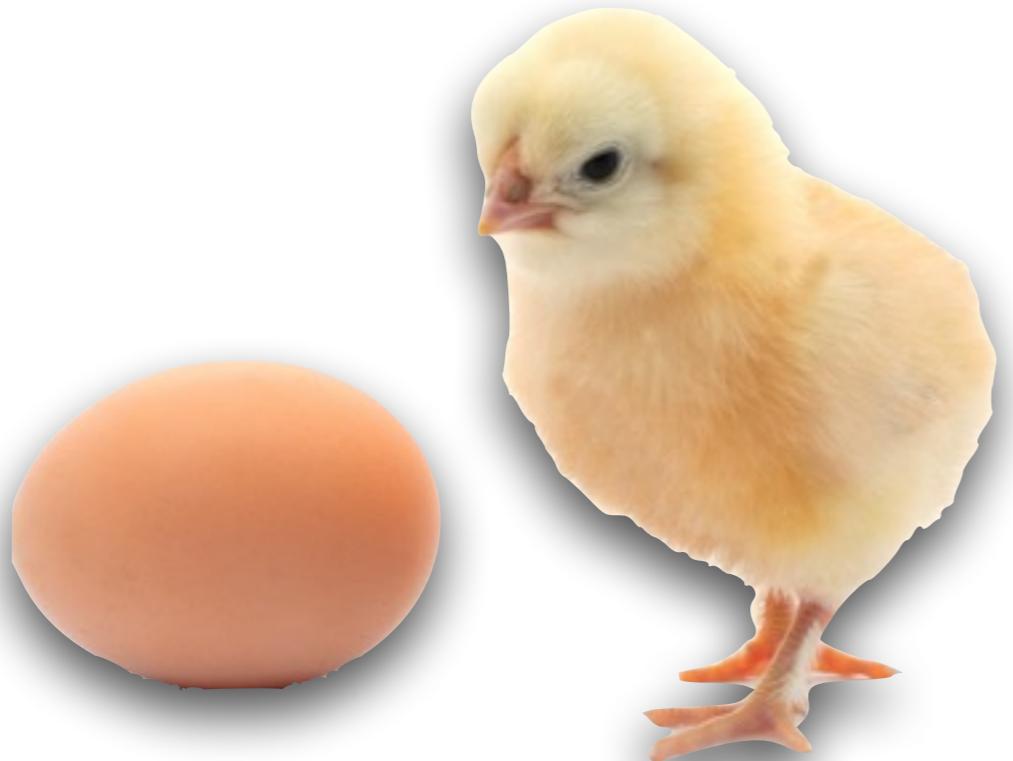


Library

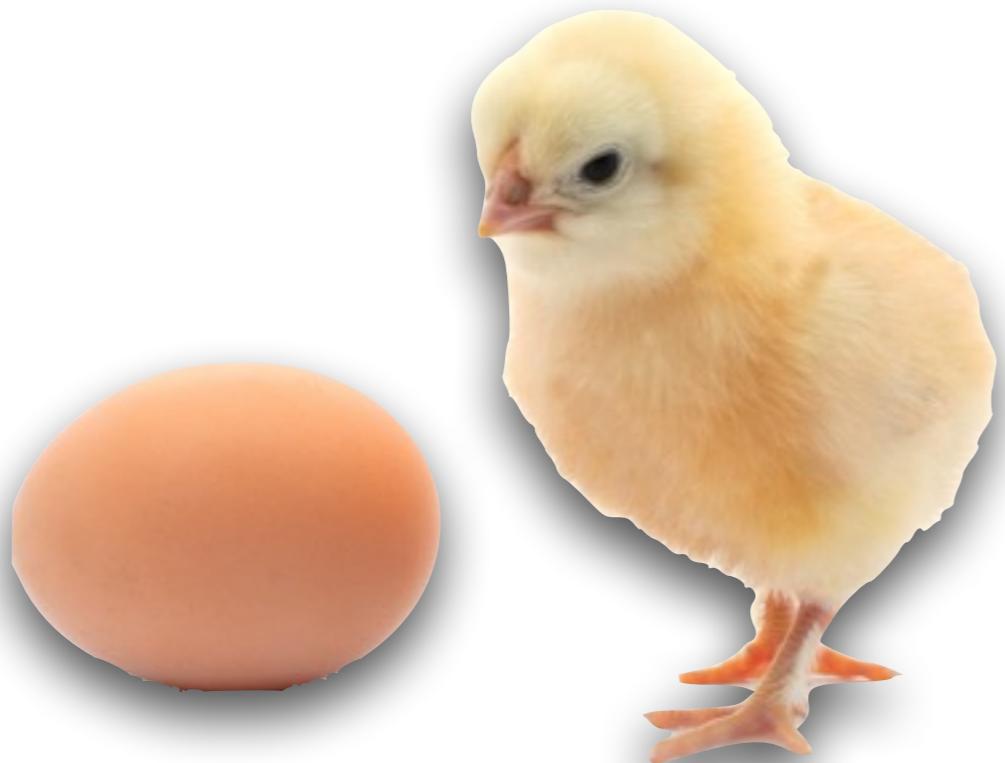


Possible Parameters

Conundrum

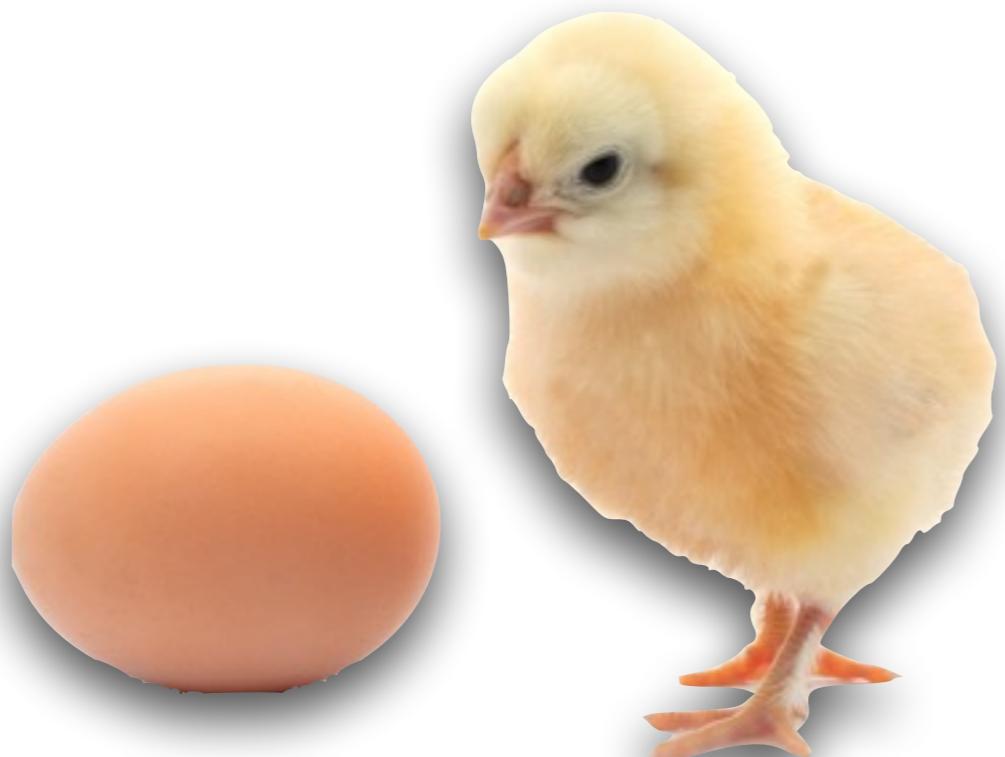


Conundrum



- ❖ Dynamic defect detection requires effective tests

Conundrum



- ❖ Dynamic defect detection requires effective tests
- ❖ Test synthesis requires some knowledge of defects

Problem Statement

How to synthesize multi-threaded tests that
expose *atomicity violations*?



Key Insight

Use data from sequential test execution to construct multithreaded tests

Overview

INTRUDER



Overview

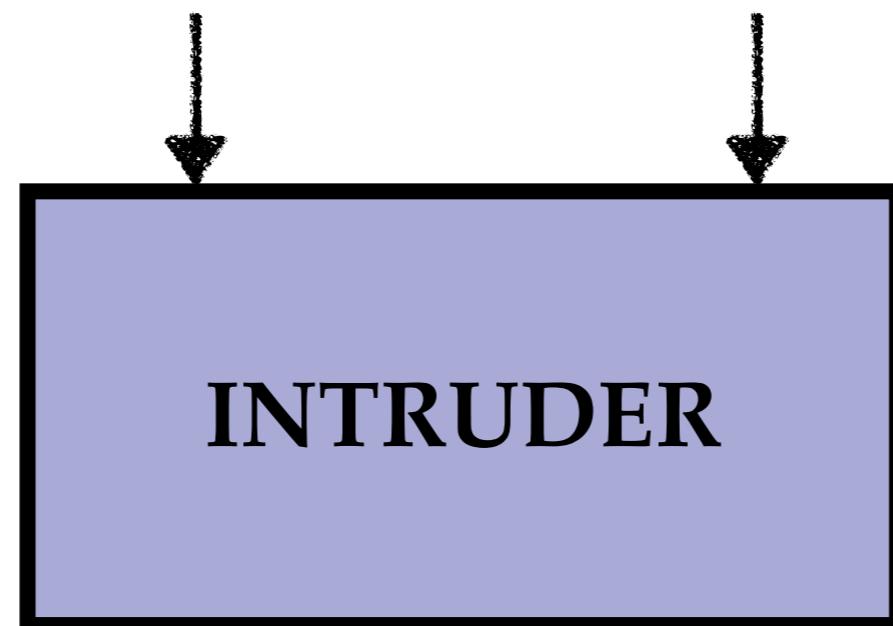
Library Sequential Tests

INTRUDER



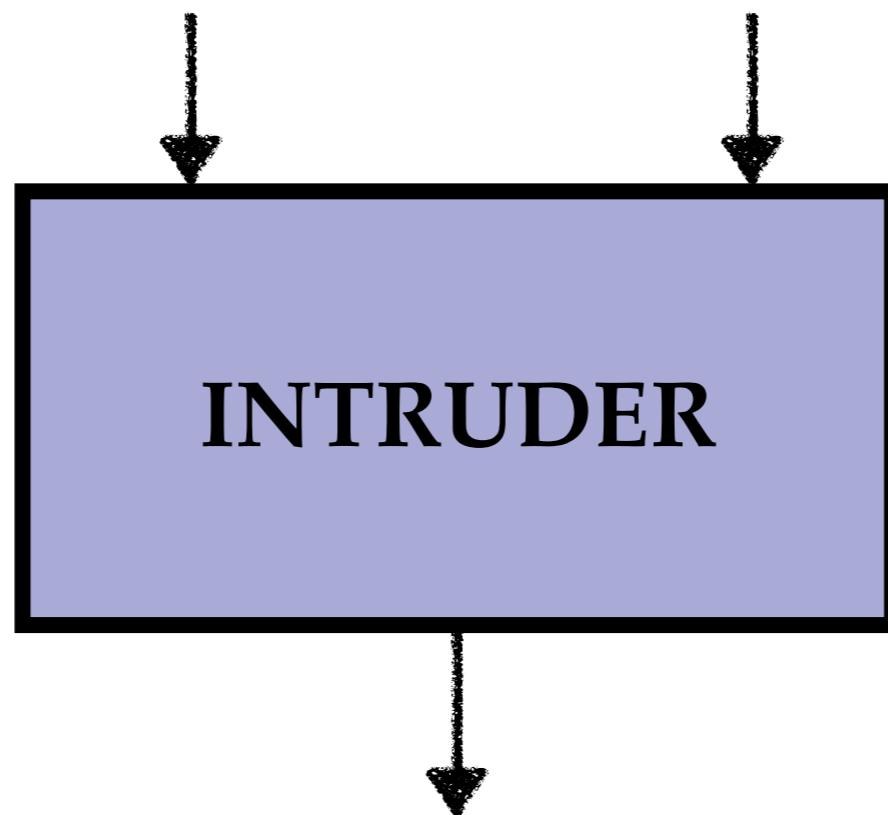
Overview

Library Sequential Tests



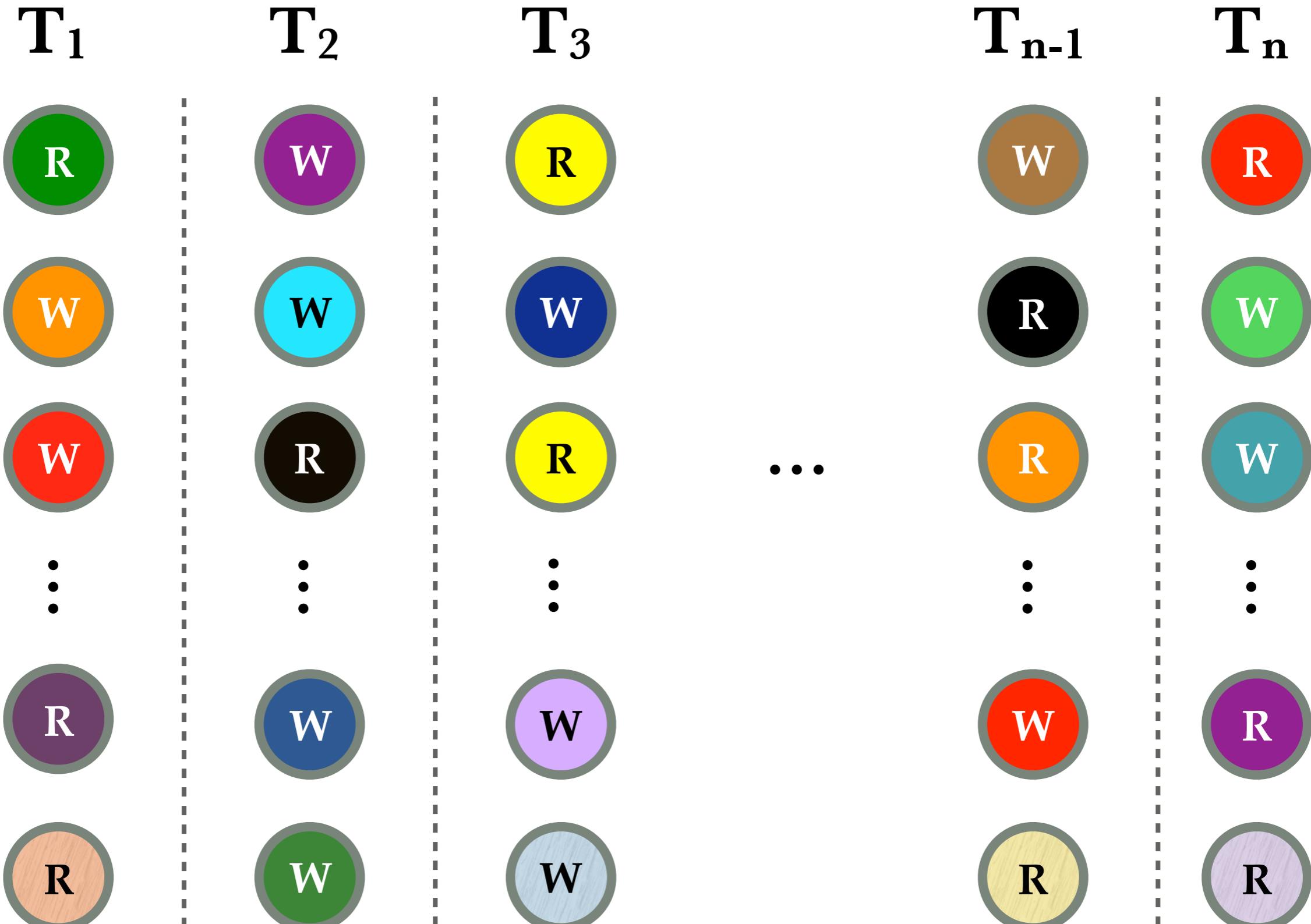
Overview

Library Sequential Tests

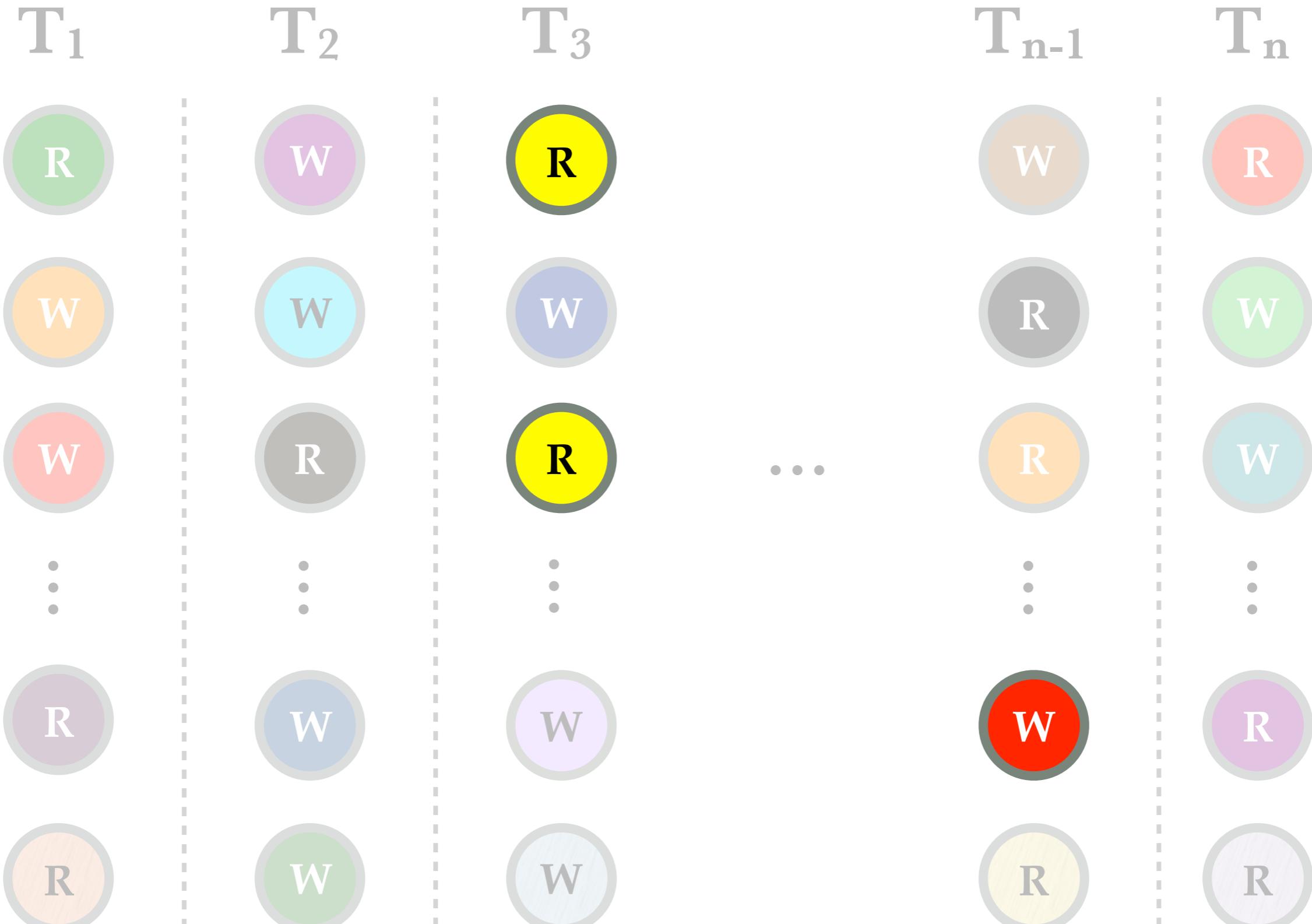


Multithreaded Tests

Challenge 1 : Identify accesses triplets



Challenge 1 : Identify accesses triplets



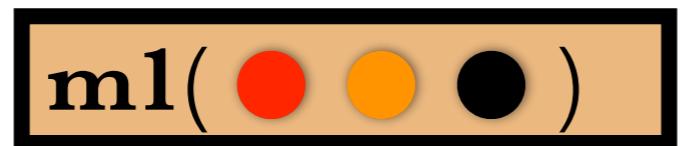
Challenge 2: Access Reachability



Indian Institute of Science, Bangalore

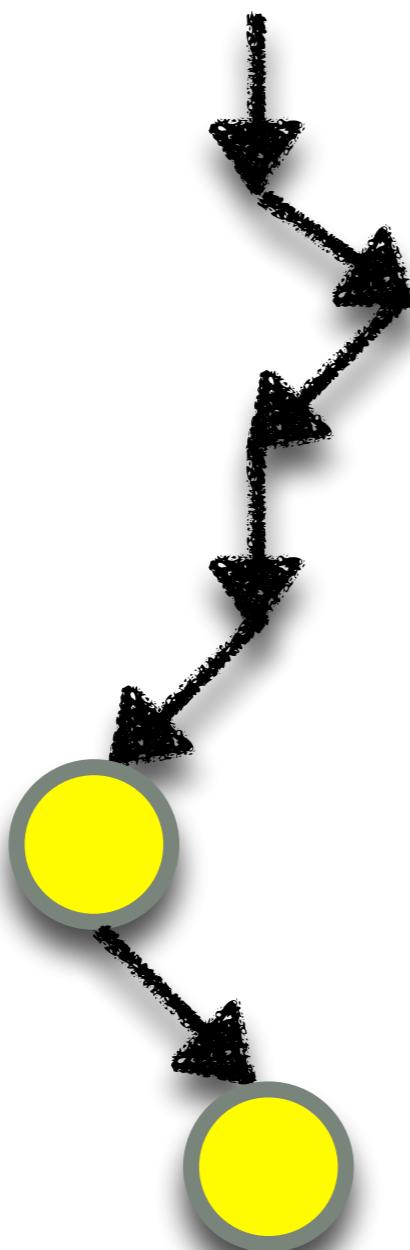


Challenge 2: Access Reachability



Challenge 2: Access Reachability

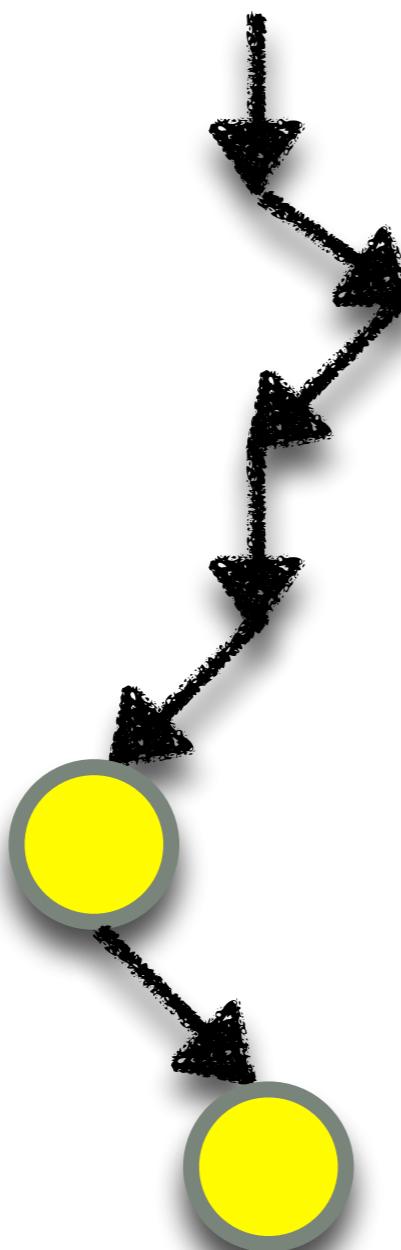
ml(● ● ●)



Challenge 2: Access Reachability

m1(●●●)

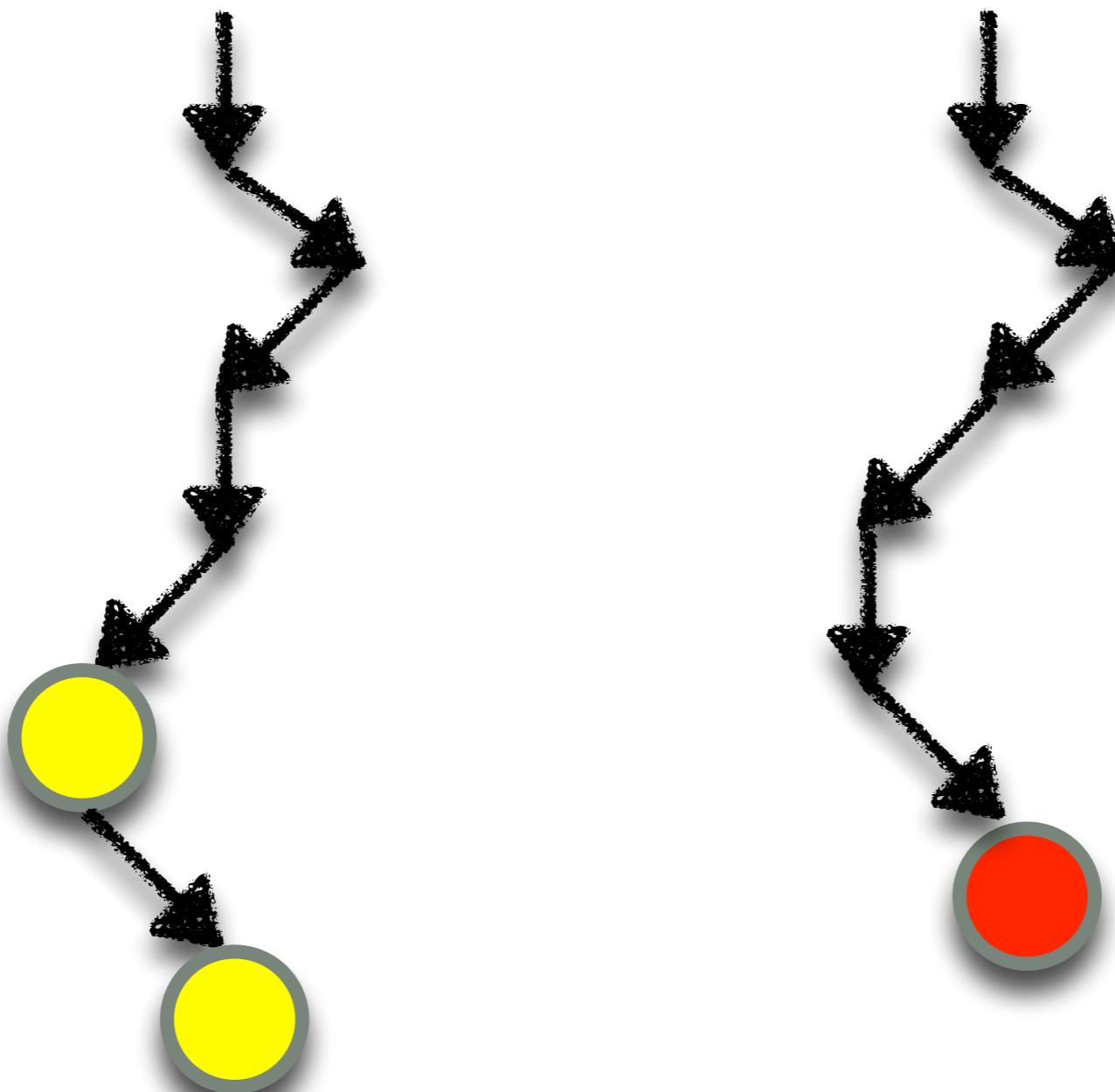
m4(●●●)



Challenge 2: Access Reachability

m1(●●●)

m4(●●●)

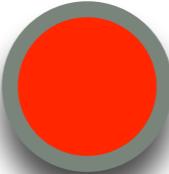
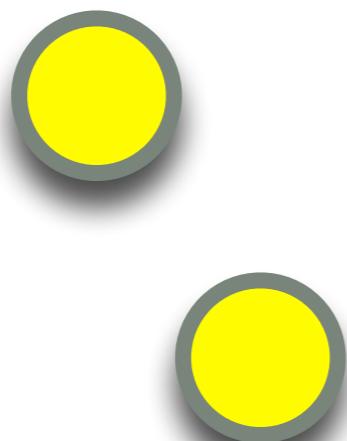


Challenge 3 : Setting Objects

Setter

m1(● ● ●)

m4(● ● ●)

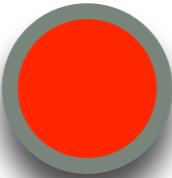
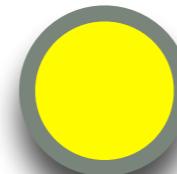


Challenge 3 : Setting Objects

Setter

m1()

m4(● ● ●)

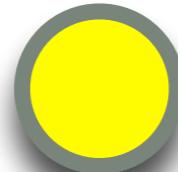
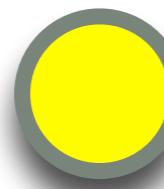


Challenge 3 : Setting Objects

Setter

m1()

m4()

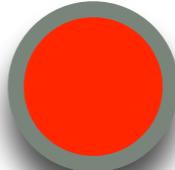
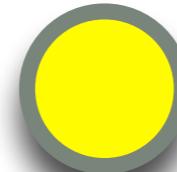


Challenge 3 : Setting Objects

Setter

m1(● ● ●)

m4()

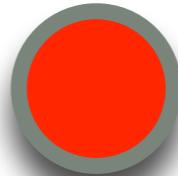
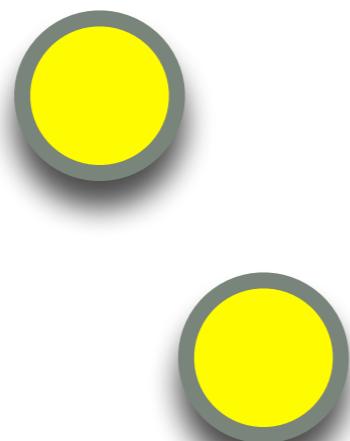


Challenge 3 : Setting Objects

Setter

m1(● ● ●)

m4(● ● ●)

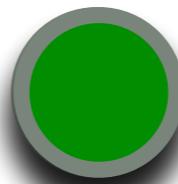
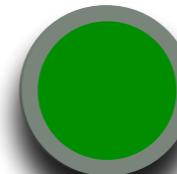


Challenge 3 : Setting Objects

Setter

m1(● ● ●)

m4(● ● ●)



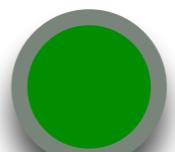
Challenge 4 : Generating Objects

Initializer

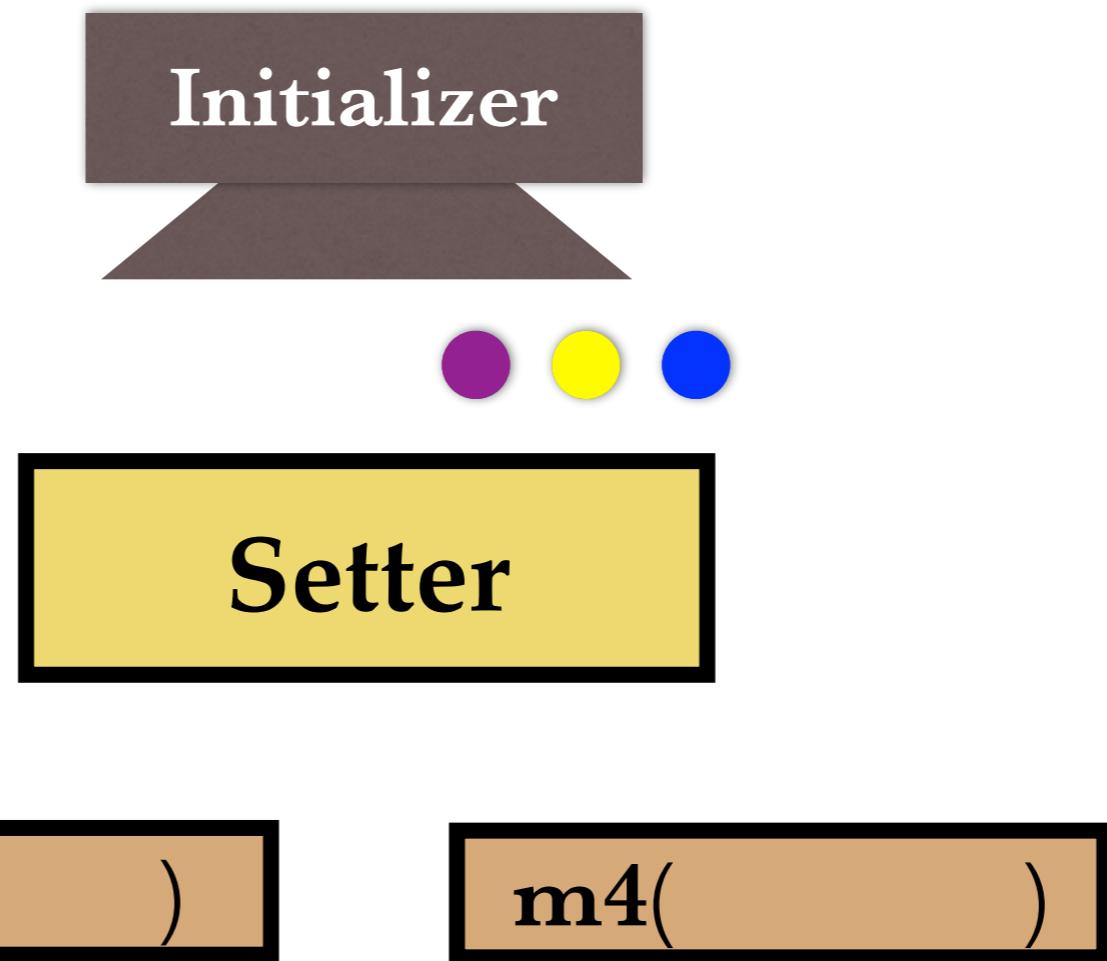
Setter

m1()

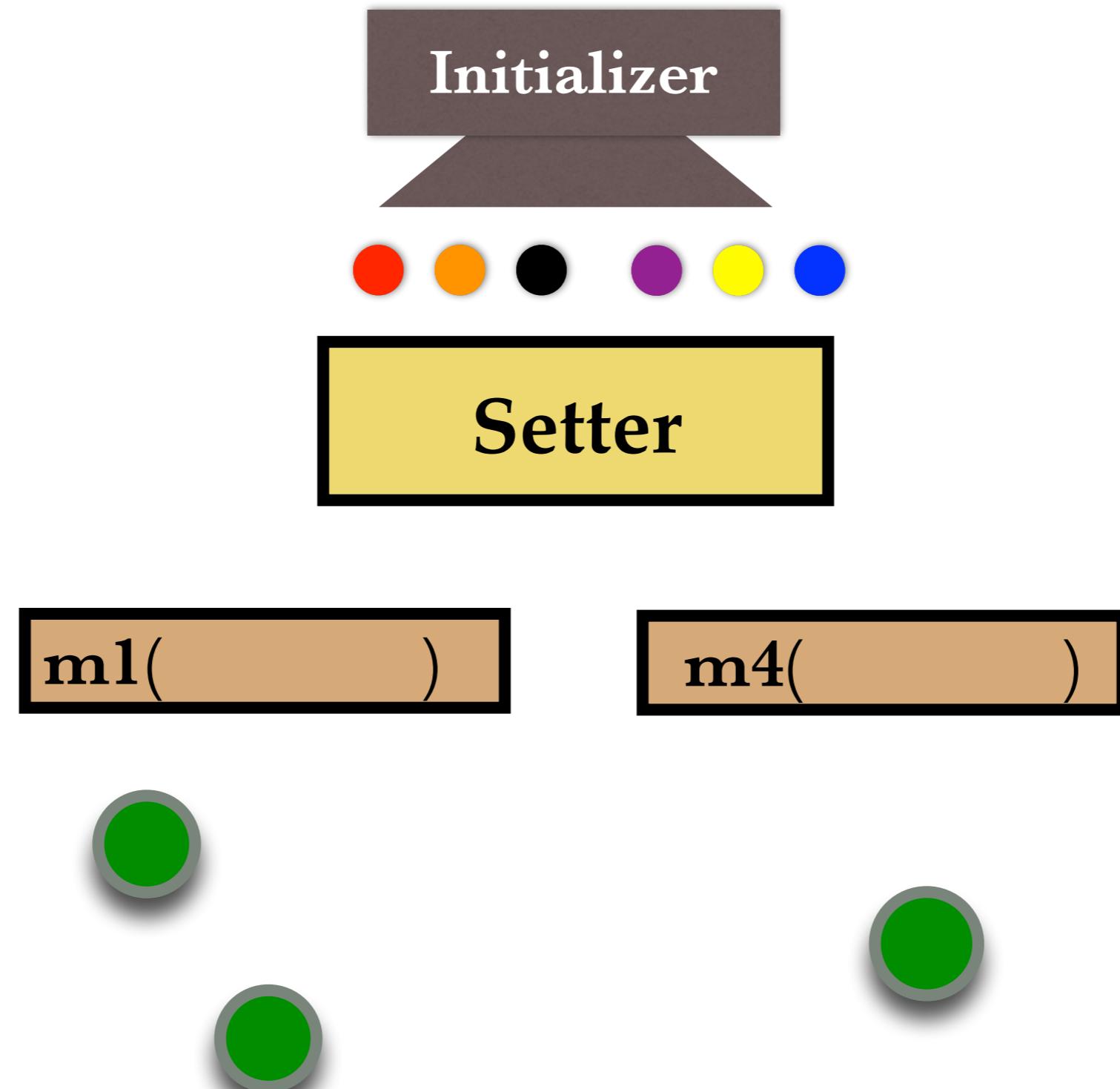
m4()



Challenge 4 : Generating Objects



Challenge 4 : Generating Objects



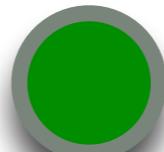
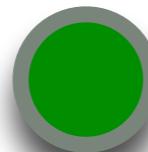
Challenge 4 : Generating Objects

Initializer

Setter

m1(● ● ● **)**

m4(● ● ● **)**



Summary of Challenges

1. Identify access triplets
2. Identify the APIs to be invoked concurrently
3. Set the context
4. Use legal object instances



Identify access triplets

Identify access triplets

Input Sequential Test

```
a1.evaluate ( ... )  
a2.reset ( ... )  
a3.set (f1)
```

Identify access triplets



```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
}
```



Input Sequential Test

```
a1.evaluate ( ... )  
a2.reset ( ... )  
a3.set (f1)
```



Identify access triplets



```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
}
```



Input Sequential Test

```
a1.evaluate ( ... )  
a2.reset ( ... )  
a3.set (f1)
```



a₁



a₁.f

Identify access triplets

```
evaluate ( ... ) {  
    temp = this.f  
    flag = (temp.g != null)  
    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```

Input Sequential Test

```
a1.evaluate ( ... )  
a2.reset ( ... )  
a3.set (f1)
```

 a₁  a₁.f
 a₂  a₃

Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

1. Access same field

Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

1. Access same field
2. P & C are consecutive accesses

Identify access triplets

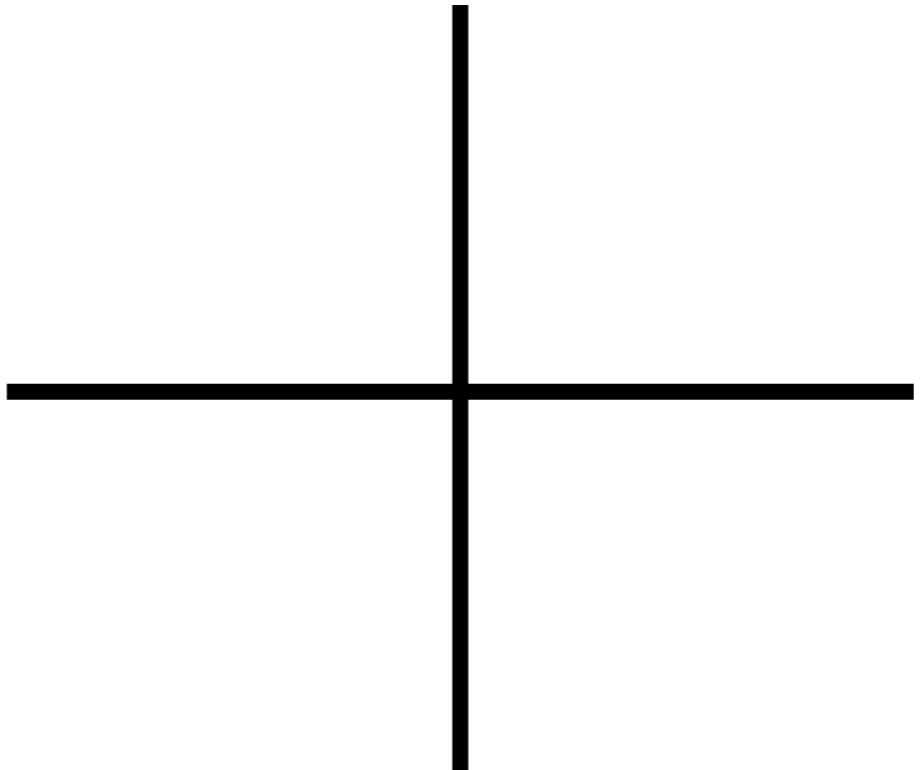
```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :

Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

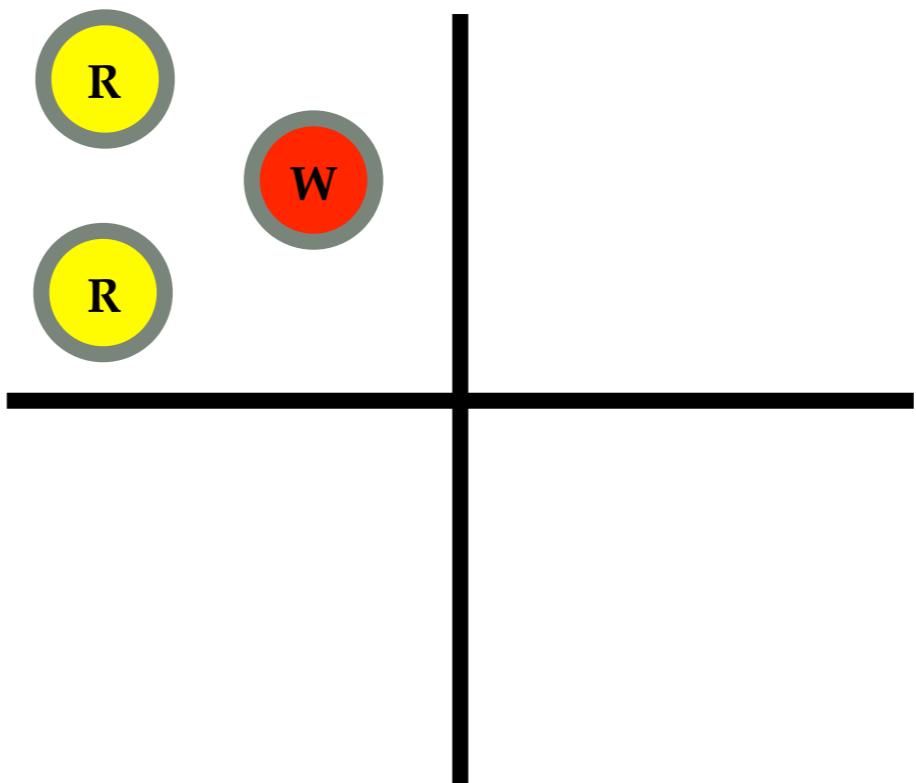
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

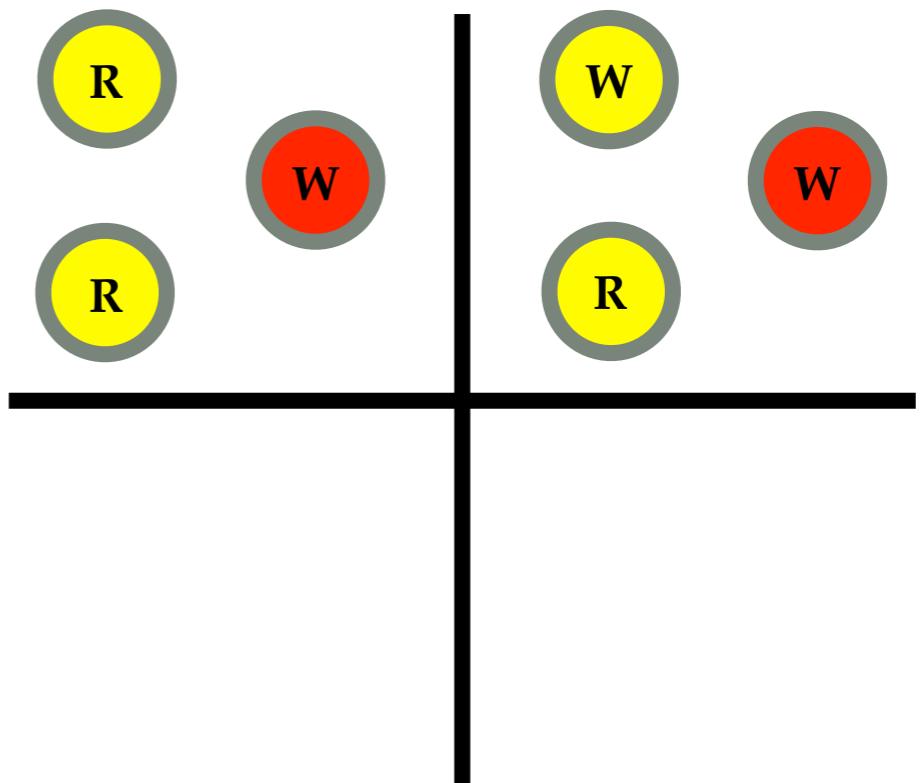
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

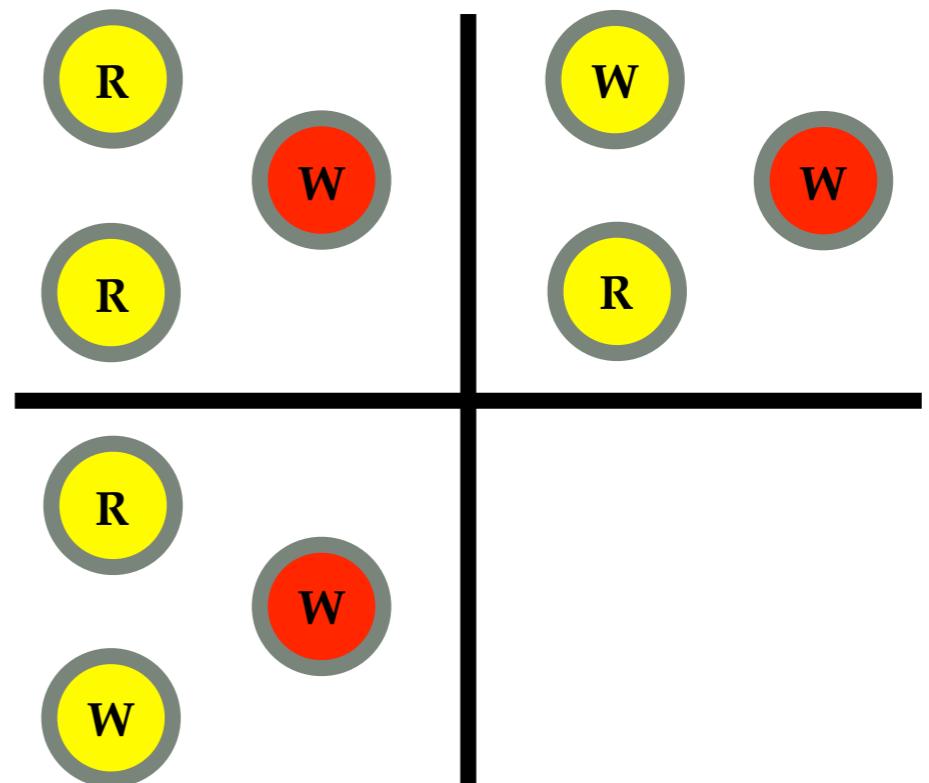
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

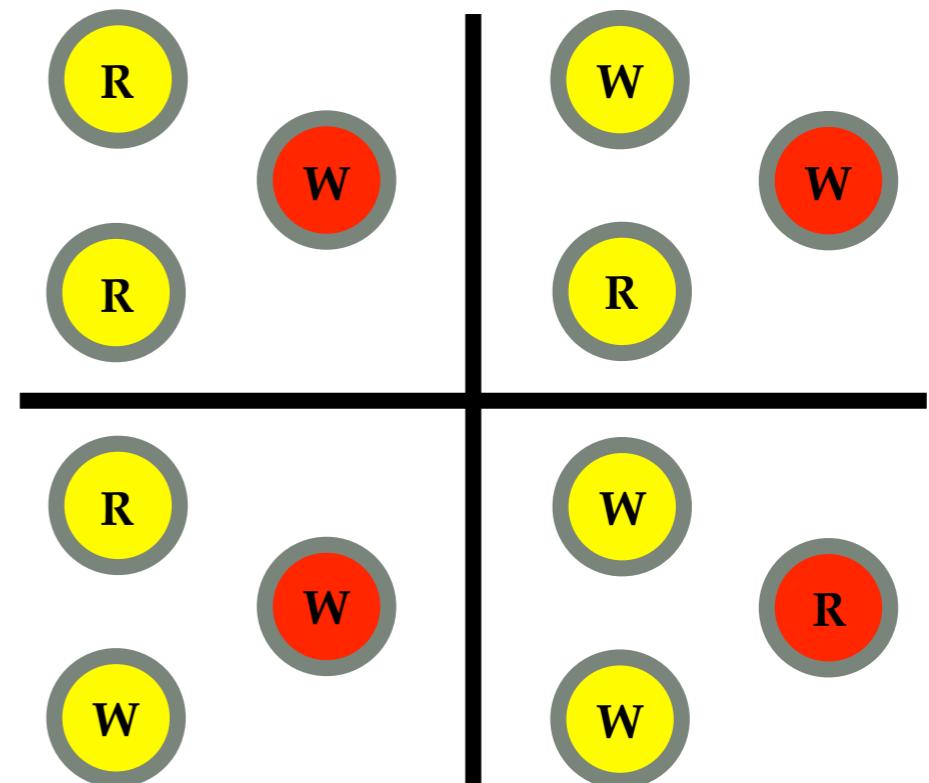
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

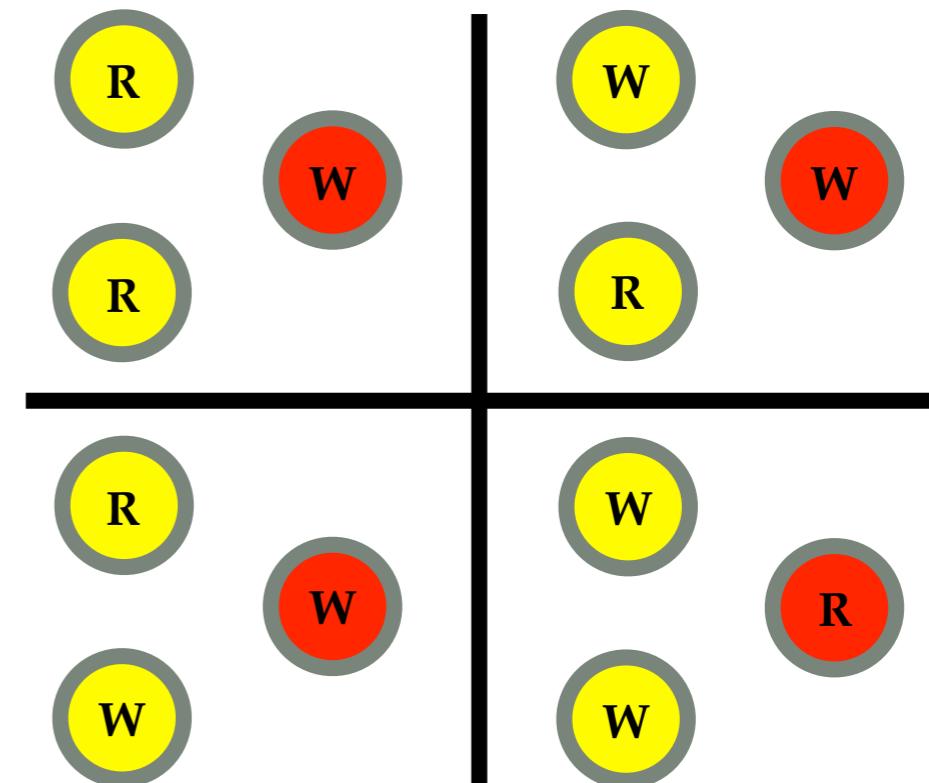
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
    temp = this.f  
P    flag = (temp.g != null)  
C    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
R    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```

1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :

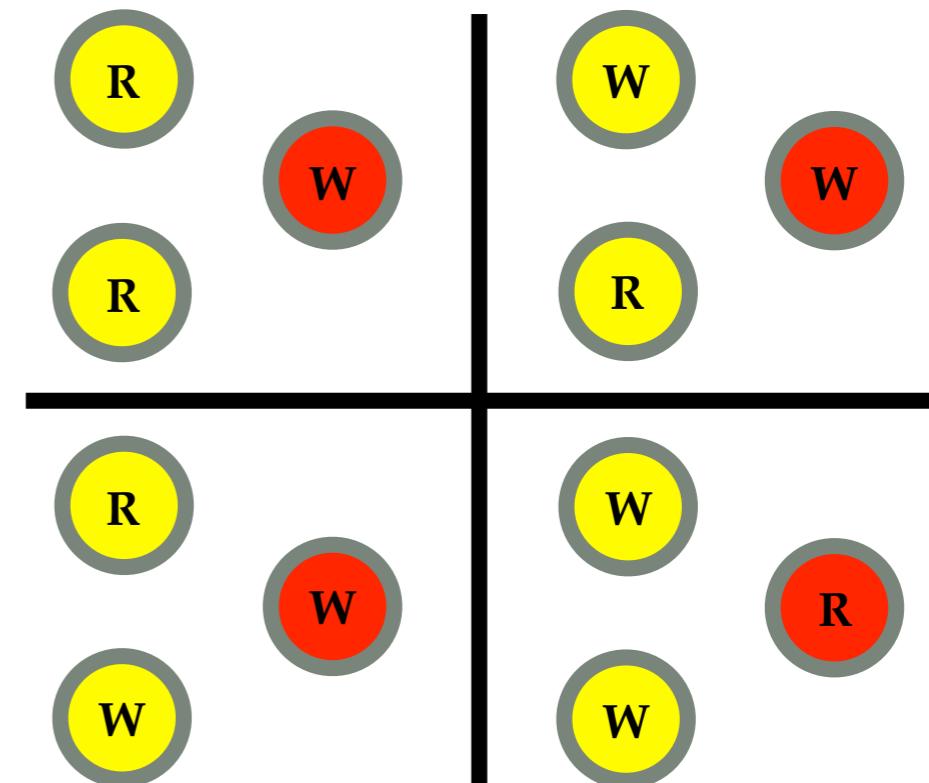


Identify access triplets

```
evaluate ( ... ) {  
    temp = this.f  
P    flag = (temp.g != null)  
C    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
R    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```



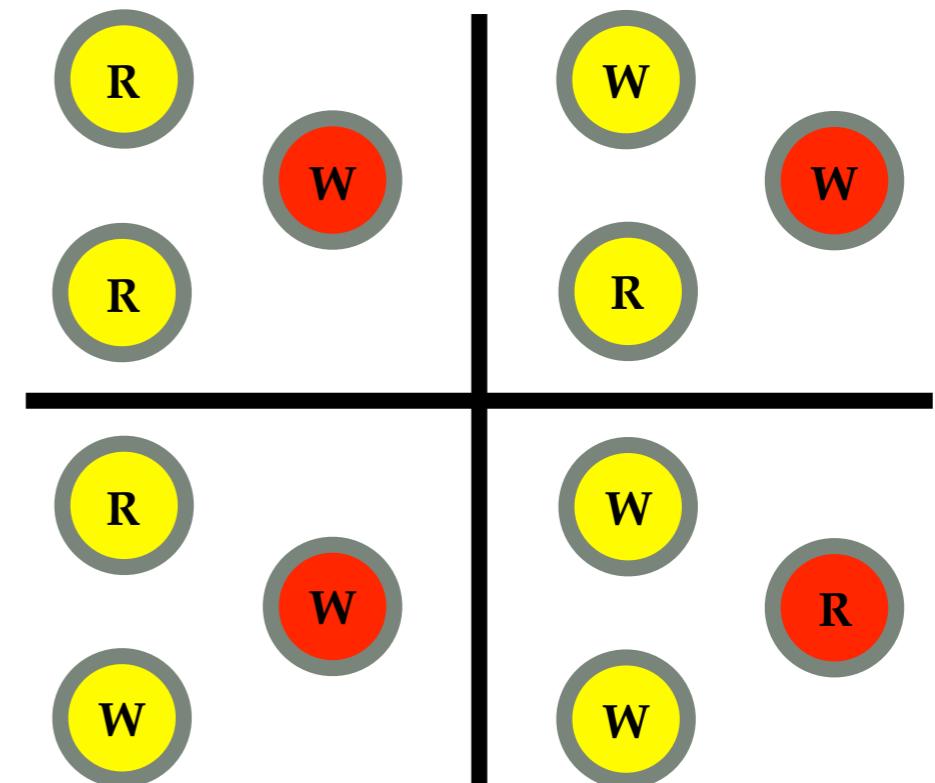
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
    temp = this.f  
P    flag = (temp.g != null)  
C    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
R    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```

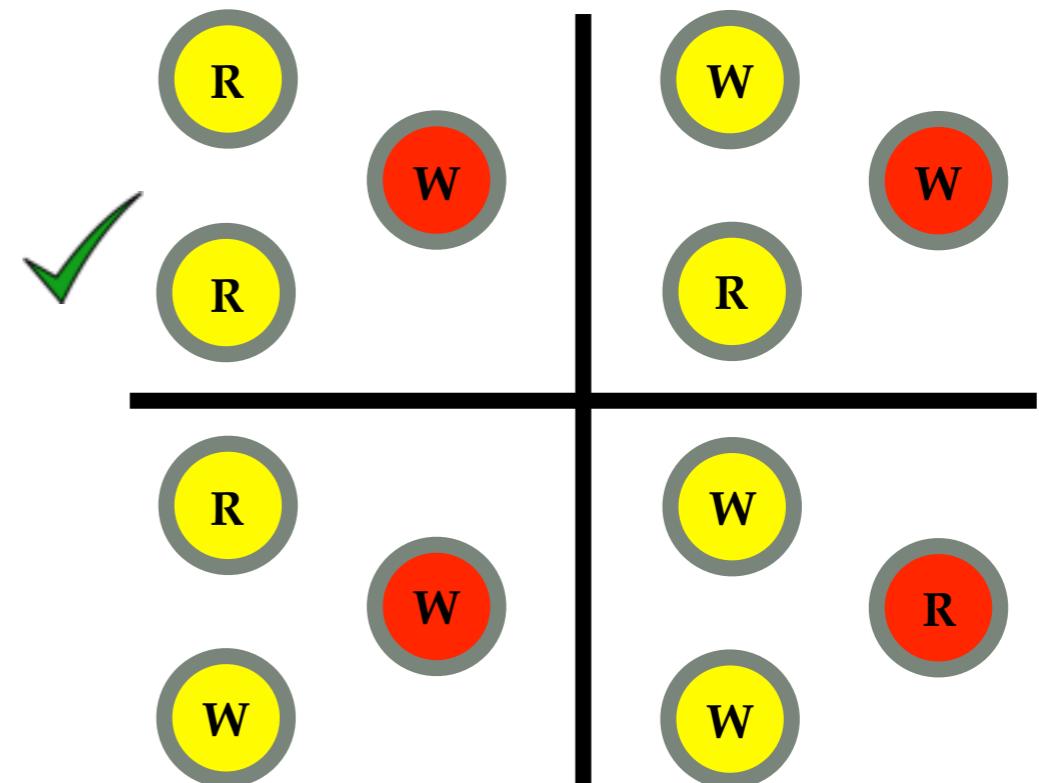
- ✓ 1. Access same field
- ✓ 2. P & C are consecutive accesses
- 3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
    temp = this.f  
P    flag = (temp.g != null)  
C    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
R    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```

- ✓ 1. Access same field
- ✓ 2. P & C are consecutive accesses
- 3. P - C - R accesses are :



Identify access triplets

```
evaluate ( ... ) {  
  
    temp = this.f  
  
P    flag = (temp.g != null)  
  
C    if(flag) assert(temp.g != null)  
  
}  
  
  
reset ( ... ) {  
  
    temp = this.f  
  
R    temp.g = null  
  
}  
  
  
set (F f) {  
    this.f = f  
}
```



Identify access triplets



```
evaluate ( ... ) {
```

```
    temp = this.f
```

P flag = (temp.g != null)

C if(flag) **assert**(temp.g != null)



```
reset ( ... ) {
```

```
    temp = this.f
```

R temp.g = null



```
set (F f) {
```

```
    this.f = f
```

```
}
```

a₁

a_{1.f}

a₂

Identify access triplets



```
evaluate ( ... ) {
```

```
    temp = this.f
```

P

```
    flag = (temp.g != null)
```

C

```
    if(flag) assert(temp.g != null)
```



```
}
```



```
reset ( ... ) {
```

```
    temp = this.f
```

R

```
    temp.g = null
```



```
}
```

```
set (F f) {
```

```
    this.f = f
```

a₁

a_{1.f}

a₂

Identify access triplets



```
evaluate ( ... ) {
```

```
    temp = this.f
```

P

```
    flag = (temp.g != null)
```

C

```
    if(flag) assert(temp.g != null)
```



```
}
```



```
reset ( ... ) {
```

```
    temp = this.f
```

R

```
    temp.g = null
```



```
}
```

```
set (F f) {  
    this.f = f  
}
```

1. $a_1 \neq a_2$



a_1

$a_1.f$

a_2

Identify access triplets



```
evaluate ( ... ) {
```

```
    temp = this.f
```

P

```
    flag = (temp.g != null)
```

C

```
    if(flag) assert(temp.g != null)
```



```
}
```



```
reset ( ... ) {
```

```
    temp = this.f
```

R

```
    temp.g = null
```



```
}
```

```
set (F f) {
```

```
    this.f = f
```

```
}
```

1. $a_1 \neq a_2$ \neq
2. $a_1.f.g = a_2.f.g$



Identify access triplets



```
evaluate ( ... ) {
```

```
    temp = this.f
```

P

```
    flag = (temp.g != null)
```

C

```
    if(flag) assert(temp.g != null)
```



```
}
```



```
reset ( ... ) {
```

```
    temp = this.f
```

R

```
    temp.g = null
```



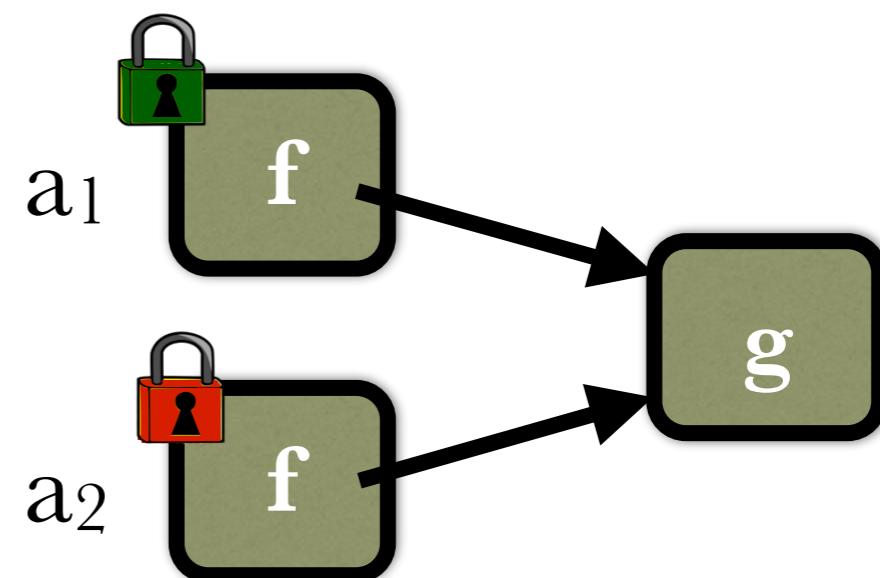
```
}
```

```
set (F f) {
```

```
    this.f = f
```

```
}
```

1. $a_1 \neq a_2$ \neq
2. $a_1.f.g = a_2.f.g$



a_1



$a_1.f$



a_2

Identify access triplets



```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)
```

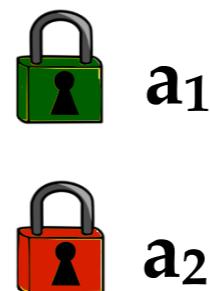


```
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
temp.g = null  
}
```



```
set (F f) {  
  
    this.f = f  
}
```

1. $a_1 \neq a_2$ =
2. $a_1.f.g = a_2.f.g$



Identify the APIs

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    P flag = (temp.g != null)  
  
    C if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    R temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
  
}
```



Identify the APIs

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    P flag = (temp.g != null)  
  
    C if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    R temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
  
}
```

evaluate



Identify the APIs

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    P flag = (temp.g != null)  
  
    C if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    R temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
  
}
```

← evaluate

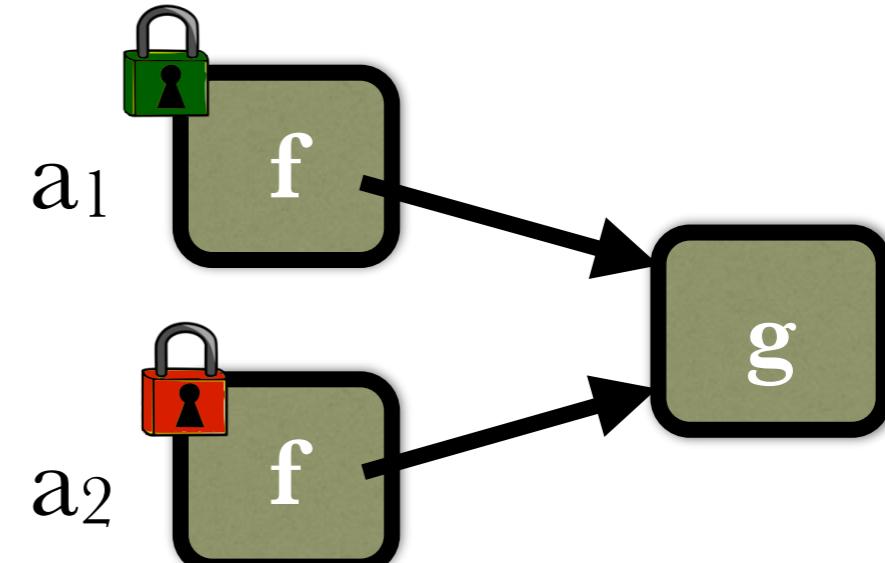
→ reset

Set the context

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
temp.g = null  
  
}  
  
set (F f) {  
  
    this.f = f  
}
```

Set the context

```
evaluate ( ... ) {  
    temp = this.f  
    flag = (temp.g != null)  
    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```

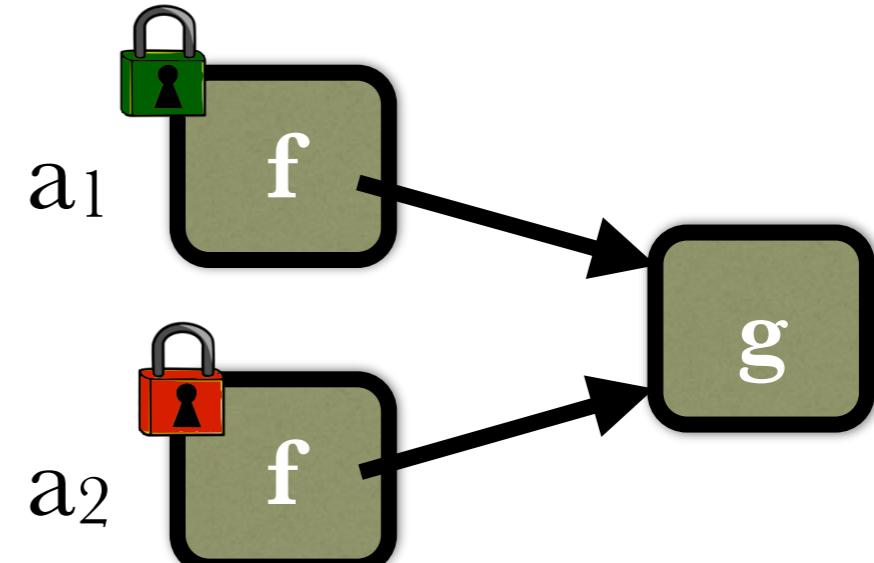


Desired object state

Set the context

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}
```

```
set (F f) {  
    this.f = f  
}
```

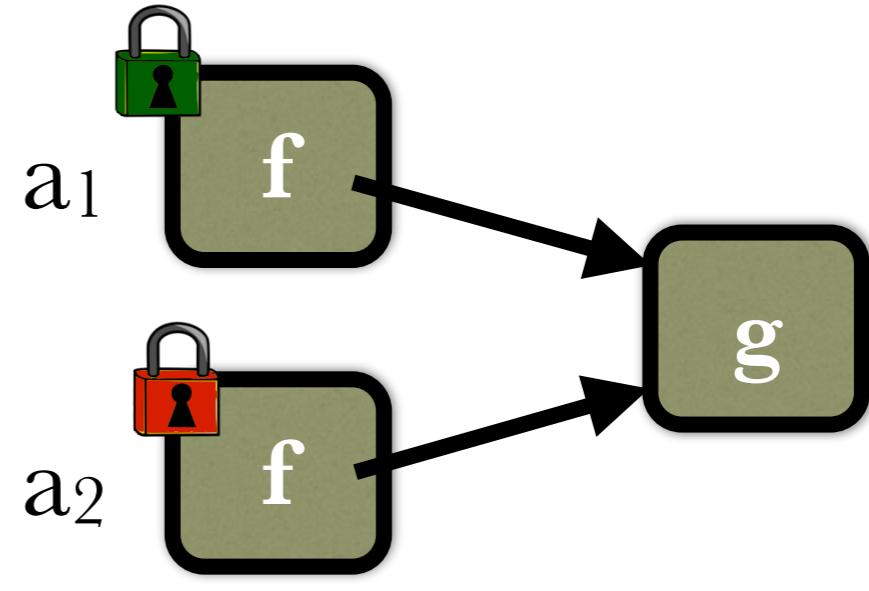


Desired object state

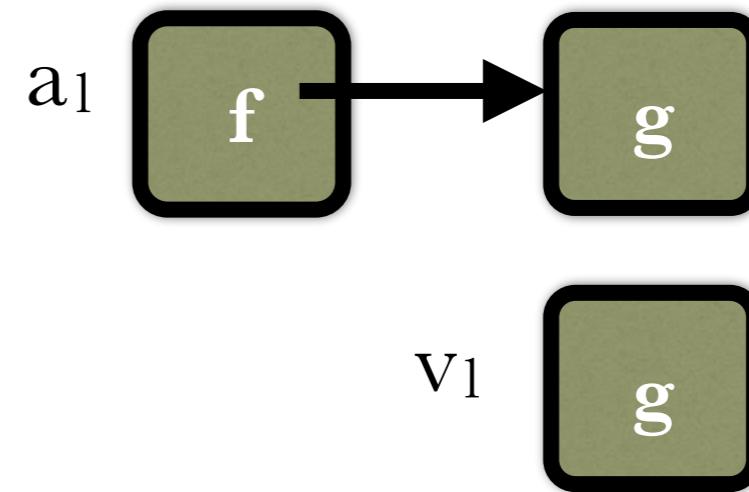
Set the context

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}
```

```
set (F f) {  
    this.f = f  
}
```



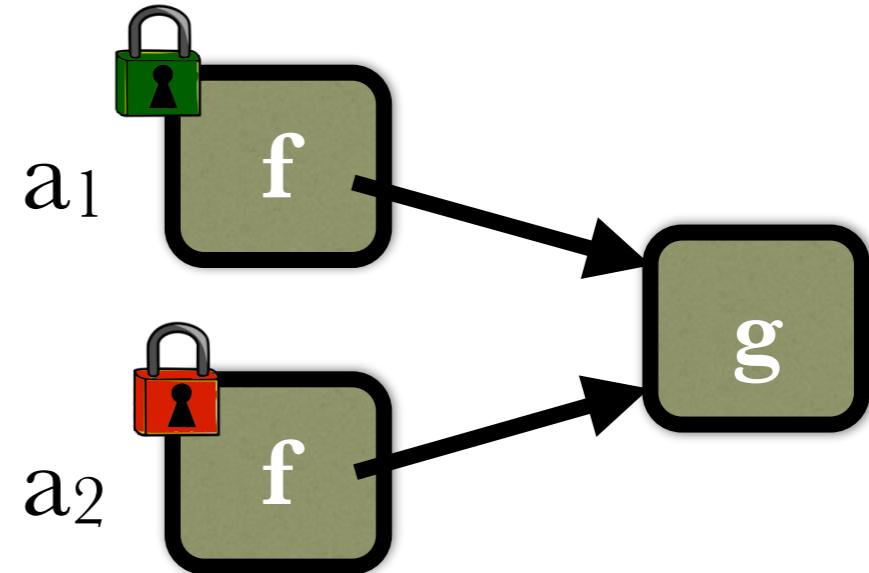
Desired object state



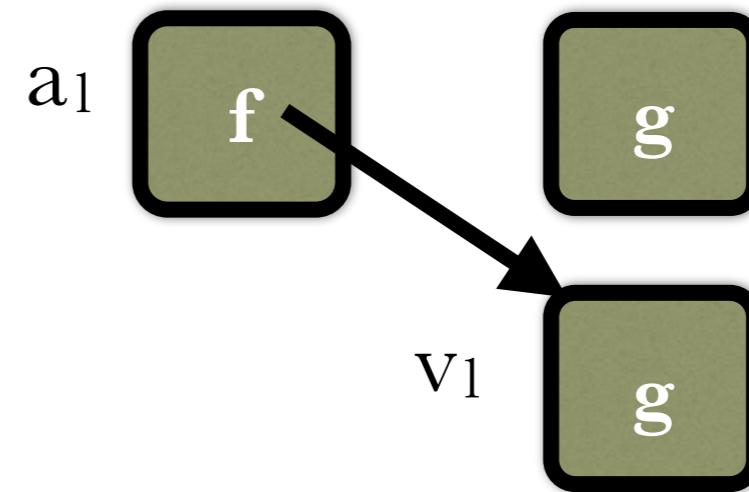
Set the context

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}
```

```
set (F f) {  
    this.f = f  
}
```

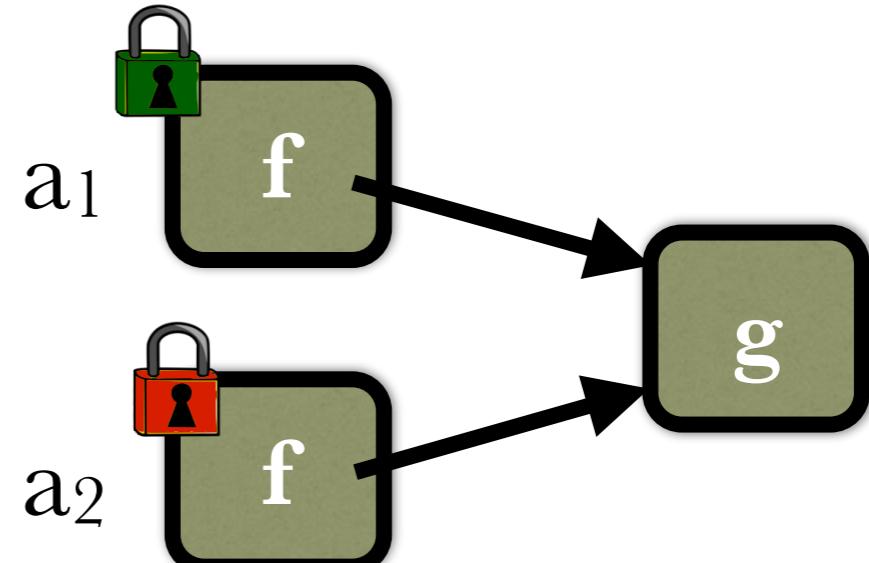


Desired object state

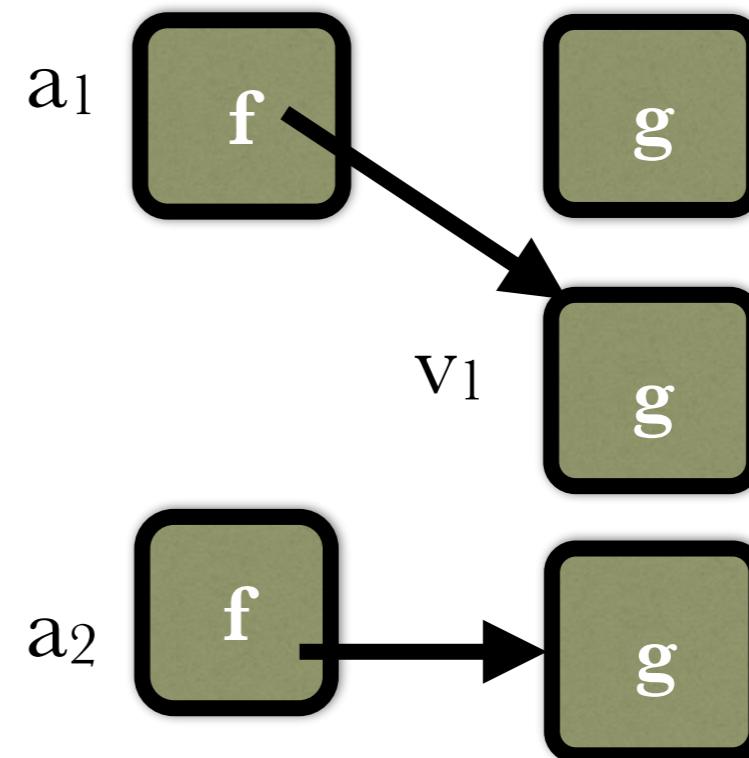


Set the context

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}  
  
set (F f) {  
    this.f = f  
}
```

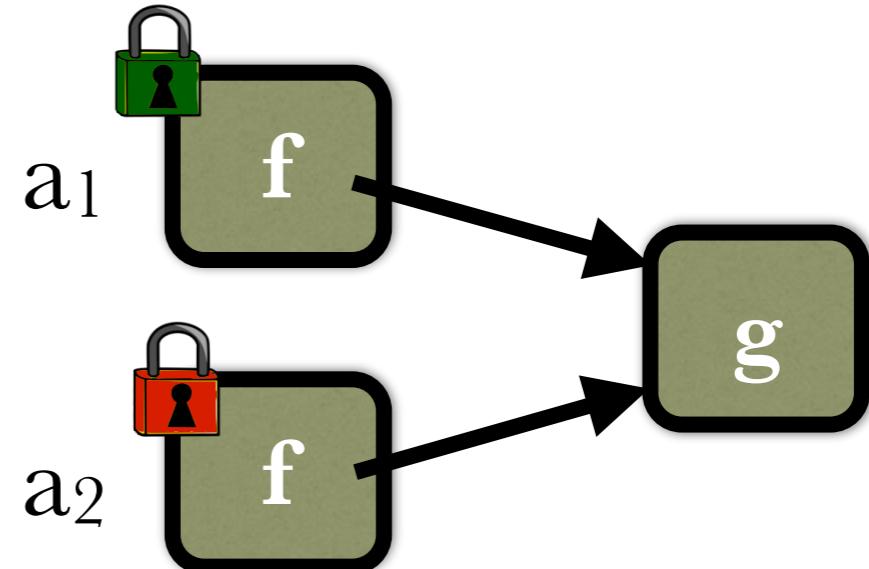


Desired object state

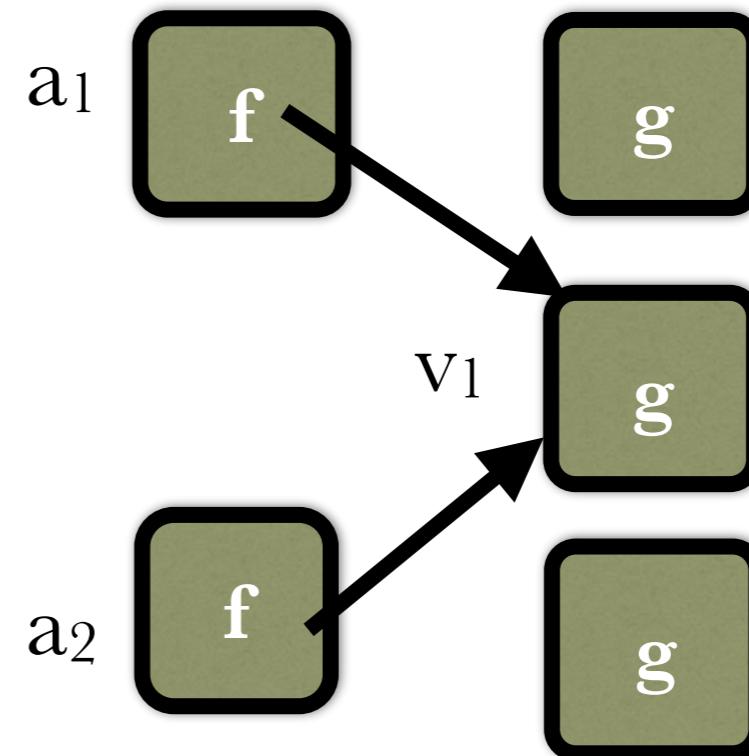


Set the context

```
evaluate ( ... ) {  
    temp = this.f  
    flag = (temp.g != null)  
    if(flag) assert(temp.g != null)  
}  
  
reset ( ... ) {  
    temp = this.f  
    temp.g = null  
}  
  
set (F f) {  
    this.f = f  
}
```



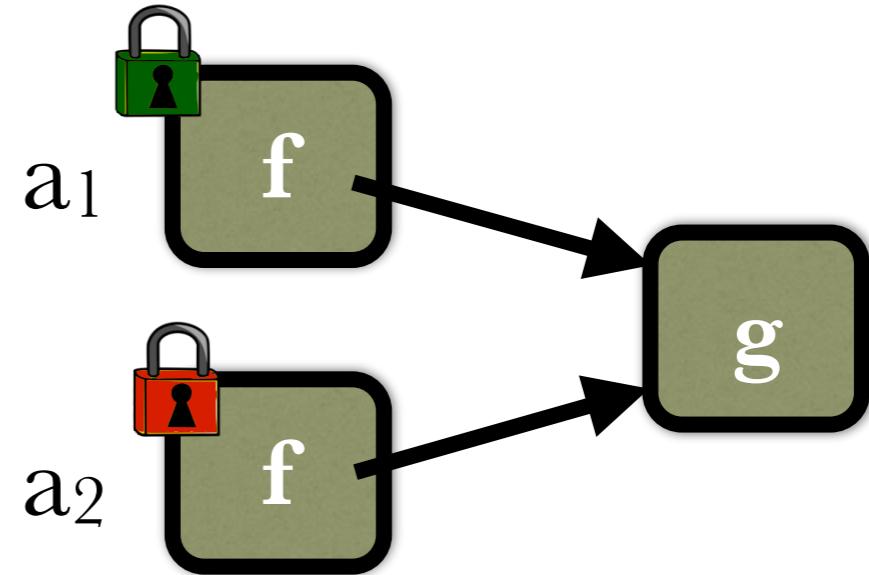
Desired object state



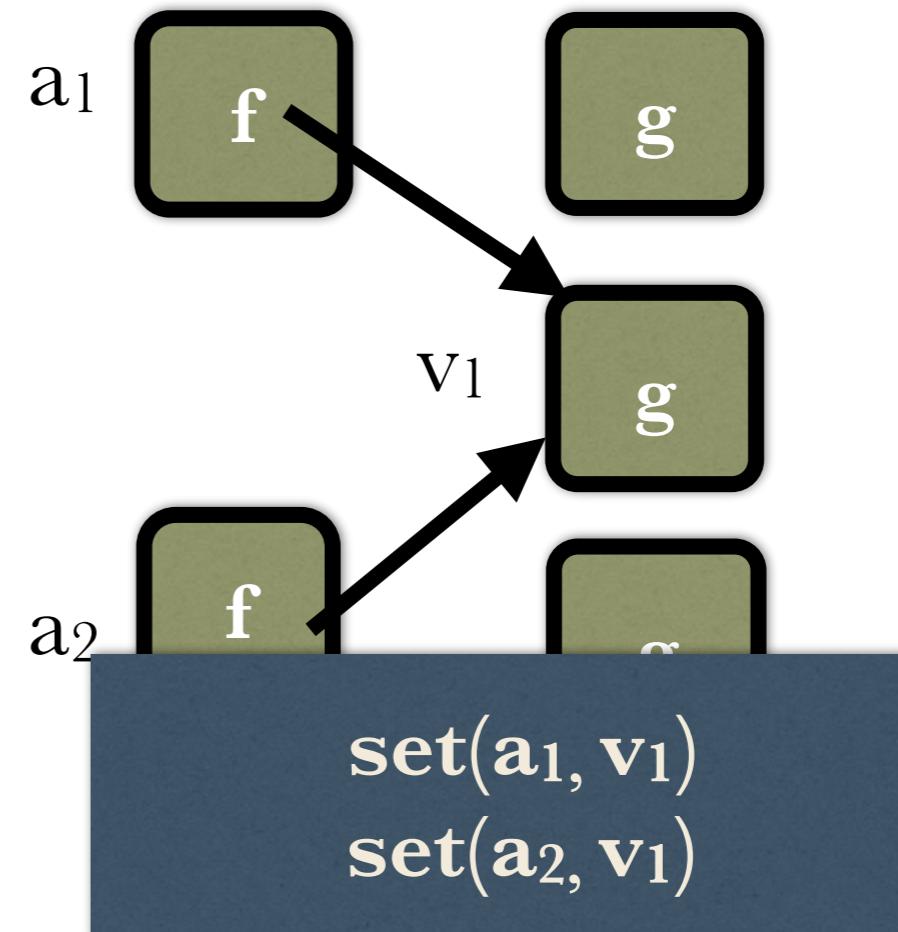
Set the context

```
evaluate ( ... ) {  
  
    temp = this.f  
  
    flag = (temp.g != null)  
  
    if(flag) assert(temp.g != null)  
  
}  
  
reset ( ... ) {  
  
    temp = this.f  
  
    temp.g = null  
  
}
```

```
set (F f) {  
    this.f = f  
}
```



Desired object state



Object Generation

Execute sequential tests for obtaining object instances

Object Generation

Execute sequential tests for obtaining object instances

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

set :

set :

evaluate:

reset :

Object Generation

Execute sequential tests for obtaining object instances

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

set : (o₁, o₂)

set :

evaluate:

reset :

Object Generation

Execute sequential tests for obtaining object instances

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

set : (o₁, o₂)

set : (o₃, o₄)

evaluate:

reset :

Object Generation

Execute sequential tests for obtaining object instances

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

set : (o₁, o₂)

set : (o₃, o₄)

evaluate: (o₅, ...)

reset :

Object Generation

Execute sequential tests for obtaining object instances

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

```
a1.evaluate (...)  
a2.reset ( ... )  
a3.set (f1)
```

set : (o₁, o₂)

set : (o₃, o₄)

evaluate: (o₅, ...)

reset : (o₆, ...)

Generated Test

o₁.set(o₂)

o₃.set(o₂)

o₁.evaluate (...)

o₃.reset (...)

Generated Test

O₁.set(O₂)

O₃.set(O₂)

O₁.evaluate (...)

O₃.reset (...)

Required Test Case !



Experimental Validation

* Replication package evaluated



Indian Institute of Science, Bangalore



Experimental Validation

* **Replication package evaluated**

- ◆ Implemented using the SOOT bytecode analysis framework

Experimental Validation

* **Replication package evaluated**

- ◆ Implemented using the SOOT bytecode analysis framework
- ◆ Evaluated on open source Java libraries



Experimental Validation

* **Replication package evaluated**

- ◆ Implemented using the SOOT bytecode analysis framework
- ◆ Evaluated on open source Java libraries
- ◆ Invoked each method in a class once with random objects



Experimental Validation

* Replication package evaluated

- ◆ Implemented using the SOOT bytecode analysis framework
- ◆ Evaluated on open source Java libraries
- ◆ Invoked each method in a class once with random objects
- ◆ Used our implementation of CTrigger (Park et al., ASPLoS'09) for JAVA to detect atomicity violations

Comparison

- ◆ **ConTeGe** : Pradel and Gross, PLDI'12
- ◆ **Omen** : Samak and Ramanathan, OOPSLA'14
- ◆ **Narada** : Samak, Ramanathan, Jagannathan, PLDI'15



Tests Generated

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	9k	10	6	11
StringBuffer	1.3k	-	-	8
Vector	31k	15	-	6
SkipCursor	274	-	-	3
TimesyncClientExtension	174	-	-	1
ApplicationStatistic	172	-	-	1
PortalStatistic	165	-	-	1
CompositeGraphicsNode	10k	-	4	7
PerformanceMonitor	1.2k	-	-	2
	53k	25	10	40

Tests Generated

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	9k	10	6	11
StringBuffer	1.3k	-	-	8
Vector	31k	15	-	6
SkipCursor	274	-	-	3
TimesyncClientExtension	174	-	-	1
ApplicationStatistic	172	-	-	1
PortalStatistic	165	-	-	1
CompositeGraphicsNode	10k	-	4	7
PerformanceMonitor	1.2k	-	-	2
	53k	25	10	40

Tests Generated

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	9k	10	6	11
StringBuffer	1.3k	-	-	8
Vector	31k	15	-	6
SkipCursor	274	-	-	3
TimesyncClientExtension	174	-	-	1
ApplicationStatistic	172	-	-	1
PortalStatistic	165	-	-	1
CompositeGraphicsNode	10k	-	4	7
PerformanceMonitor	1.2k	-	-	2
	53k	25	10	40

Tests Generated

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	9k	10	6	11
StringBuffer	1.3k	-	-	8
Vector	31k	15	-	6
SkipCursor	274	-	-	3
TimesyncClientExtension	174	-	-	1
ApplicationStatistic	172	-	-	1
PortalStatistic	165	-	-	1
CompositeGraphicsNode	10k	-	4	7
PerformanceMonitor	1.2k	-	-	2
	53k	25	10	40

Tests Generated

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	9k	10	6	11
StringBuffer	1.3k	-	-	8
Vector	31k	15	-	6
SkipCursor	274	-	-	3
TimesyncClientExtension	174	-	-	1
ApplicationStatistic	172	-	-	1
PortalStatistic	165	-	-	1
CompositeGraphicsNode	10k	-	4	7
PerformanceMonitor	1.2k	-	-	2
	53k	25	10	40

Defects Detected

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	-	-	-	24
StringBuffer	-	-	-	8
Vector	-	-	-	6
SkipCursor	-	-	-	-
TimesyncClientExtension	-	-	-	2
ApplicationStatistic	-	-	-	1
PortalStatistic	-	-	-	1
CompositeGraphicsNode	-	-	20	36
PerformanceMonitor	-	-	-	1
	-	-	20	79

Defects Detected

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	-	-	-	24
StringBuffer	-	-	-	8
Vector	-	-	-	6
SkipCursor	-	-	-	-
TimesyncClientExtension	-	-	-	2
ApplicationStatistic	-	-	-	1
PortalStatistic	-	-	-	1
CompositeGraphicsNode	-	-	20	36
PerformanceMonitor	-	-	-	1
	-	-	20	79

Defects Detected

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	-	-	-	24
StringBuffer	-	-	-	8
Vector	-	-	-	6
SkipCursor	-	-	-	-
TimesyncClientExtension	-	-	-	2
ApplicationStatistic	-	-	-	1
PortalStatistic	-	-	-	1
CompositeGraphicsNode	-	-	20	36
PerformanceMonitor	-	-	-	1
	-	-	20	79

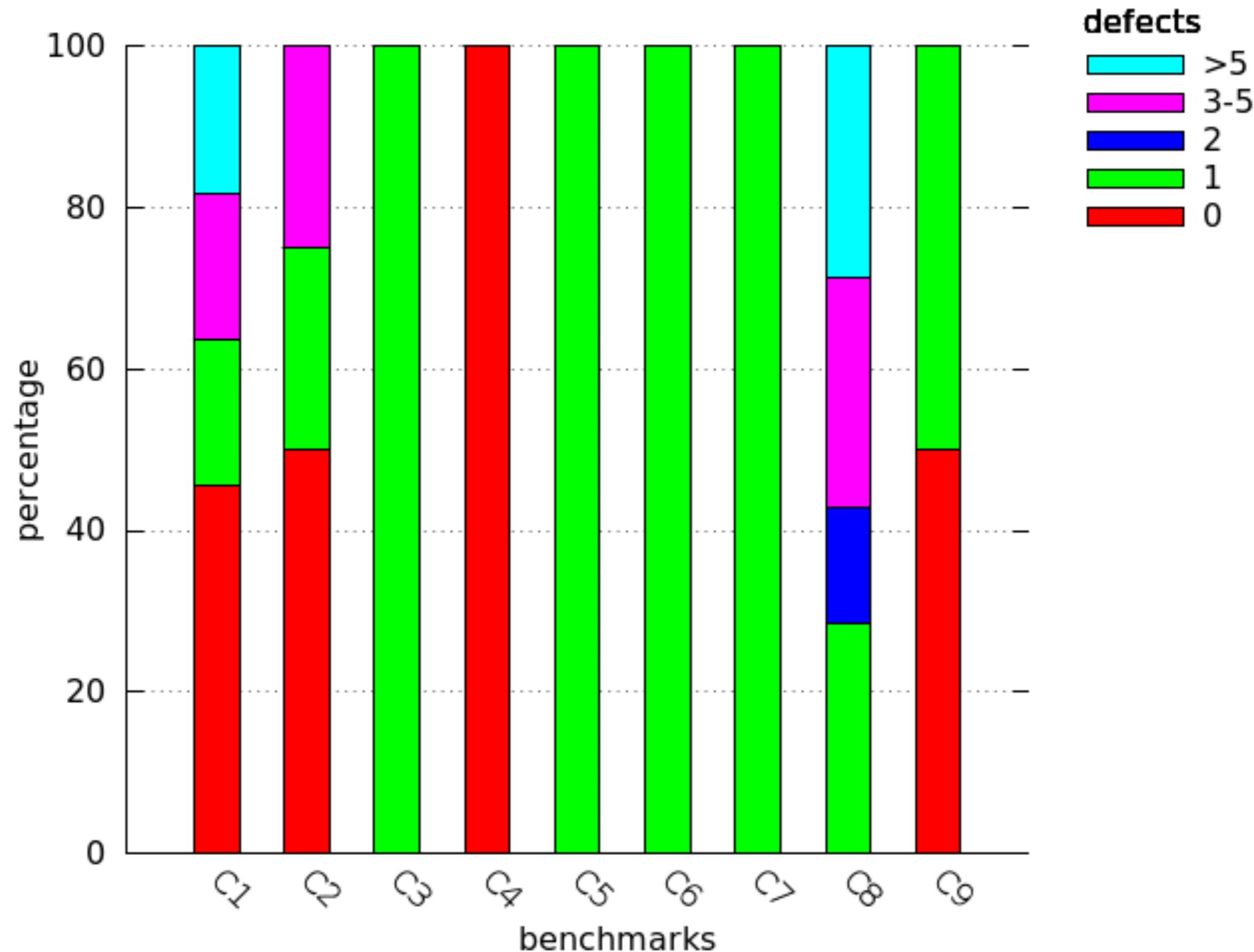
Defects Detected

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	-	-	-	24
StringBuffer	-	-	-	8
Vector	-	-	-	6
SkipCursor	-	-	-	-
TimesyncClientExtension	-	-	-	2
ApplicationStatistic	-	-	-	1
PortalStatistic	-	-	-	1
CompositeGraphicsNode	-	-	20	36
PerformanceMonitor	-	-	-	1
	-	-	20	79

Defects Detected

Class	ConTeGe	Omen	Narada	Intruder
DynamicBin1D	-	-	-	24
StringBuffer	-	-	-	8
Vector	-	-	-	6
SkipCursor	-	-	-	-
TimesyncClientExtension	-	-	-	2
ApplicationStatistic	-	-	-	1
PortalStatistic	-	-	-	1
CompositeGraphicsNode	-	-	20	36
PerformanceMonitor	-	-	-	1
	-	-	20	79

Distribution of Tests



Replication Package



Introduction

Thread-safe libraries can help programmers avoid the complexities of multithreading. However, designing libraries that guarantee Detecting and eliminating atomicity violations when methods in the libraries are invoked concurrently is vital in building reliable c libraries. While there are dynamic analyses to detect atomicity violations, these techniques are critically dependent on effective i designing such tests is non-trivial.

We have designed a novel and scalable tool named INTRUDER for synthesizing multithreaded tests that help detect atomicity vi implementation of the library and a sequential seed testsuite that invokes every method in the library with random parameters. If the sequential tests, generates variable lock dependencies and constructs a set of three accesses which when interleaved suit cause an atomicity violation. Subsequently, it identifies pairs of method invocations that correspond to these accesses and invo threads with appropriate objects to help expose atomicity violations.

Intruder is able to synthesize 33 multithreaded tests across nine classes in less than two minutes to detect 58 harmful atomicity unknown violations in thread-safe classes. We also demonstrate the effectiveness of Intruder by comparing the results with other synthesizing multithreaded tests.

Download

[Download three tools \(omen+, narada and intruder\) as VM image here](#)
[Download the VM image here](#)

Replication Package



Introduction

Thread-safe libraries can help programmers avoid the complexities of multithreading. However, designing libraries that guarantee Detecting and eliminating atomicity violations when methods in the libraries are invoked concurrently is vital in building reliable c libraries. While there are dynamic analyses to detect atomicity violations, these techniques are critically dependent on effective i designing such tests is non-trivial.

We have designed a novel and scalable tool named INTRUDER for synthesizing multithreaded tests that help detect atomicity vi implementation of the library and a sequential seed testsuite that invokes every method in the library with random parameters. If the sequential tests, generates variable lock dependencies and constructs a set of three accesses which when interleaved suit cause an atomicity violation. Subsequently, it identifies pairs of method invocations that correspond to these accesses and invo threads with appropriate objects to help expose atomicity violations.

Intruder is able to synthesize 33 multithreaded tests across nine classes in less than two minutes to detect 58 harmful atomicity unknown violations in thread-safe classes. We also demonstrate the effectiveness of Intruder by comparing the results with other synthesizing multithreaded tests.

Download

[Download three tools \(omen+, narada and intruder\) as VM image here](#)
[Download the VM image here](#)

Replication Package



Introduction

Thread-safe libraries can help programmers avoid the complexities of multithreading. However, designing libraries that guarantee Detecting and eliminating atomicity violations when methods in the libraries are invoked concurrently is vital in building reliable c libraries. While there are dynamic analyses to detect atomicity violations, these techniques are critically dependent on effective i designing such tests is non-trivial.

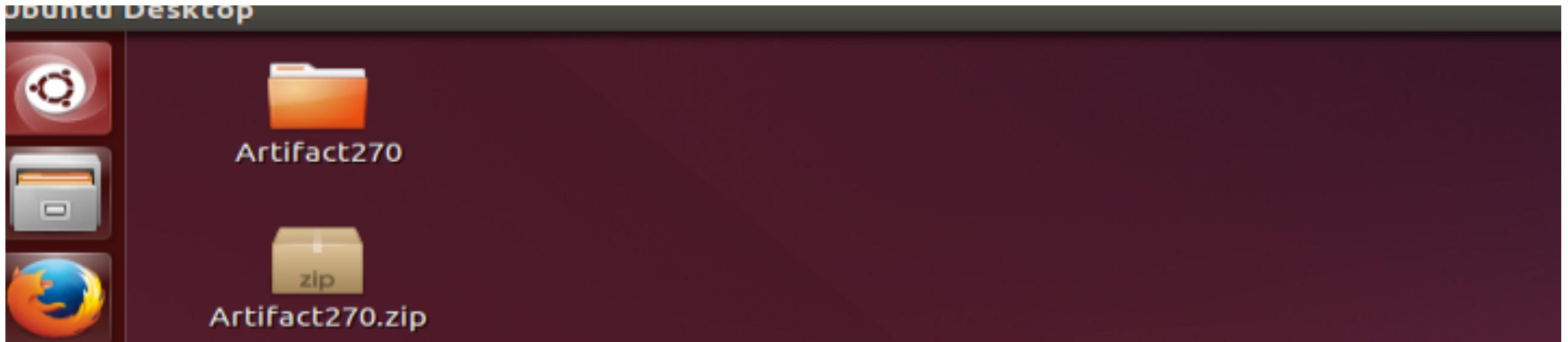
We have designed a novel and scalable tool named INTRUDER for synthesizing multithreaded tests that help detect atomicity vi implementation of the library and a sequential seed testsuite that invokes every method in the library with random parameters. If the sequential tests, generates variable lock dependencies and constructs a set of three accesses which when interleaved suit cause an atomicity violation. Subsequently, it identifies pairs of method invocations that correspond to these accesses and invo threads with appropriate objects to help expose atomicity violations.

Intruder is able to synthesize 33 multithreaded tests across nine classes in less than two minutes to detect 58 harmful atomicity unknown violations in thread-safe classes. We also demonstrate the effectiveness of Intruder by comparing the results with other synthesizing multithreaded tests.

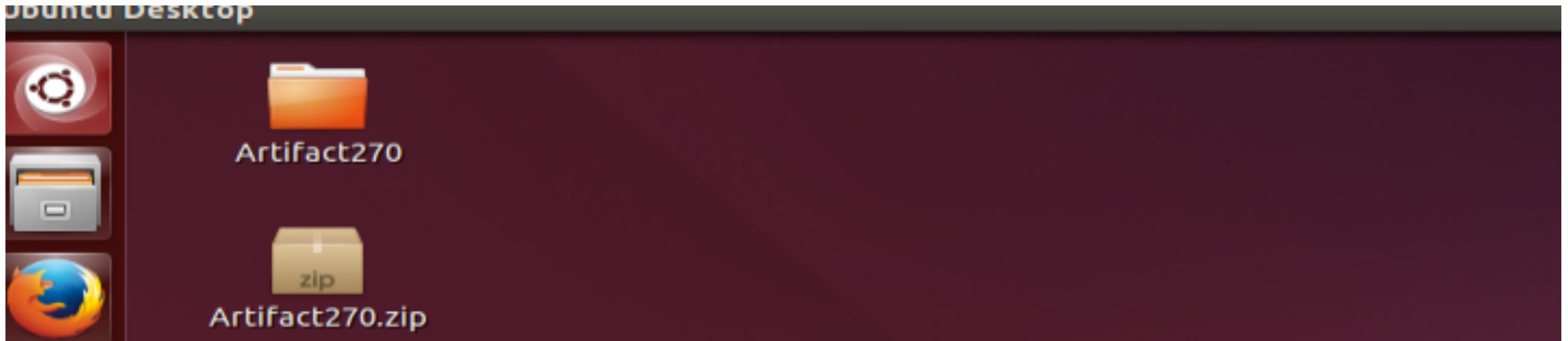
Download

Download three tools (omen+, narada and intruder) as VM image [here](#)
Download the VM image [here](#)

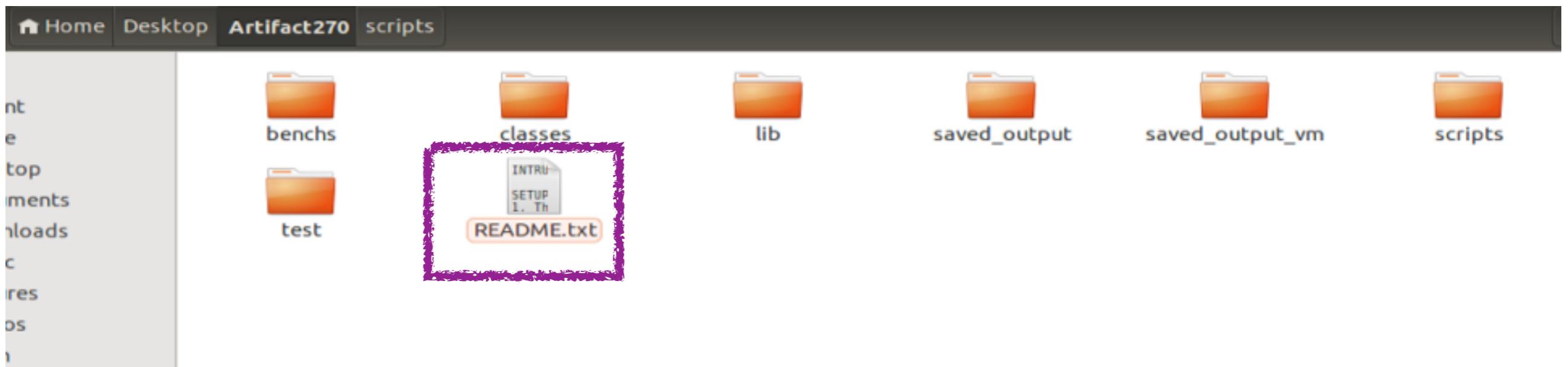
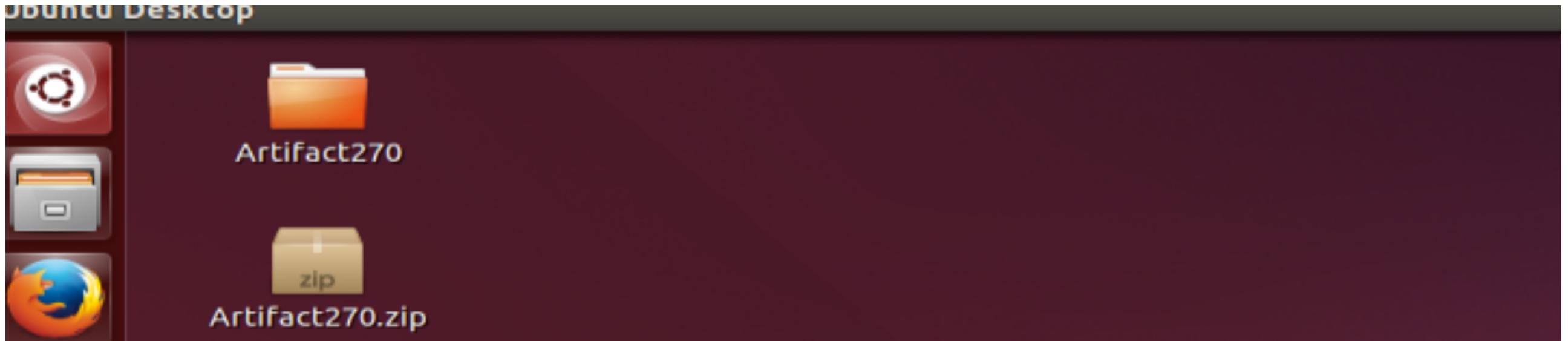
Replication Package



Replication Package



Replication Package



Replication Package



The screenshot shows a terminal window titled "ubuntu64_1 [Running]" with the command "gedit (~/Desktop/Artifact270/scripts) - gedit". The file "init.sh" is open, containing the following code:

```
export CTRIGGER=OFF
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:. ./lib/concurrent.jar:. /jsr305-1.3.9.jar:. ./lib/junit-4.11.jar:. ./lib/amen.jar:. ./lib/geotools_r.jar
```

Replication Package



ubuntu64_1 [Running]
(~Desktop/Artifact270/scripts) - gedit
init.sh x

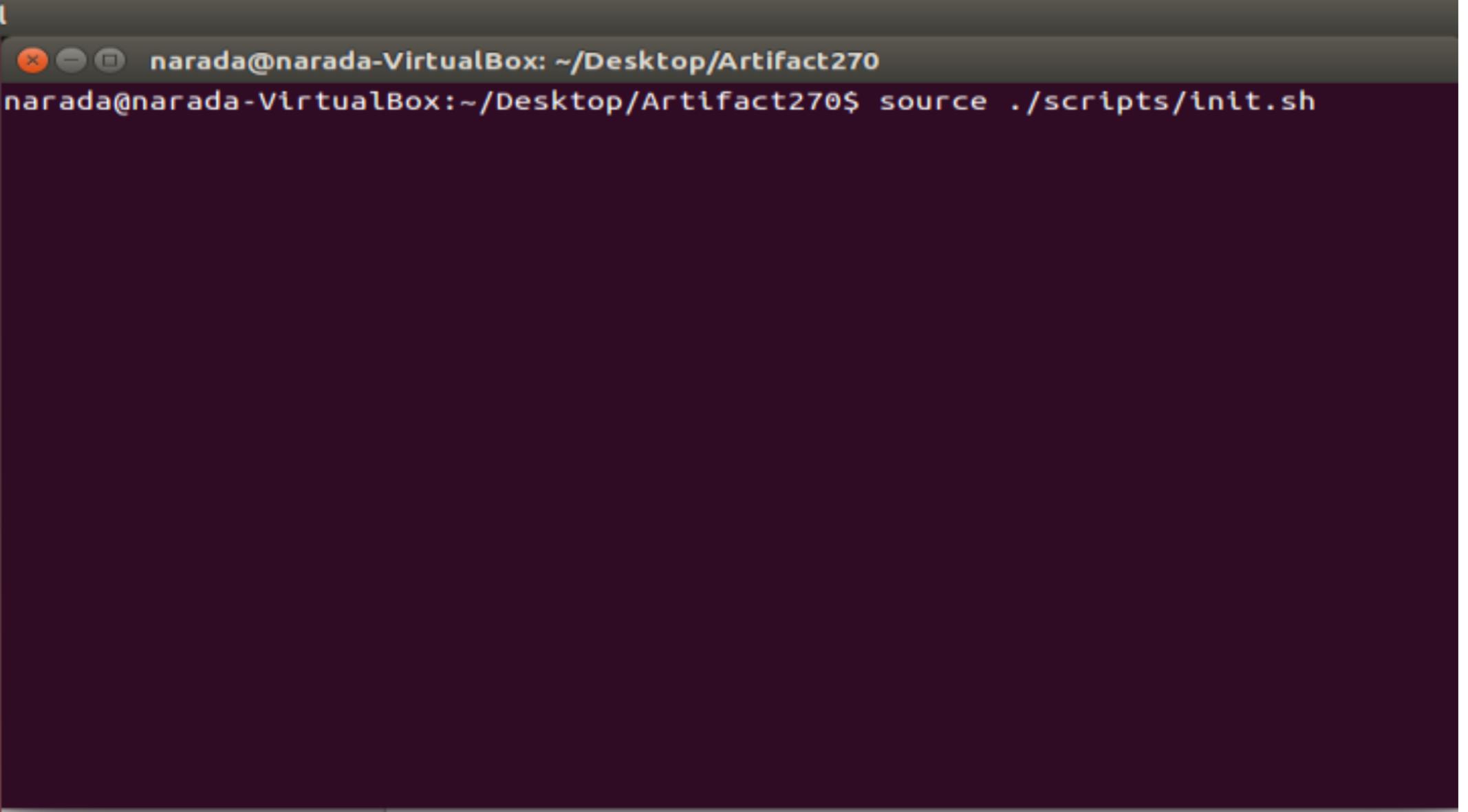
```
export CTRIGGER=OFF
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:. ./lib/concurrent.jar:. ./jsr305-1.3.9.jar:. ./lib/junit-4.11.jar:. ./lib/amen.jar:. ./lib/geotools_r.jar
```



ubuntu64_1 [Running]
(~Desktop/Artifact270/scripts) - gedit
init.sh x

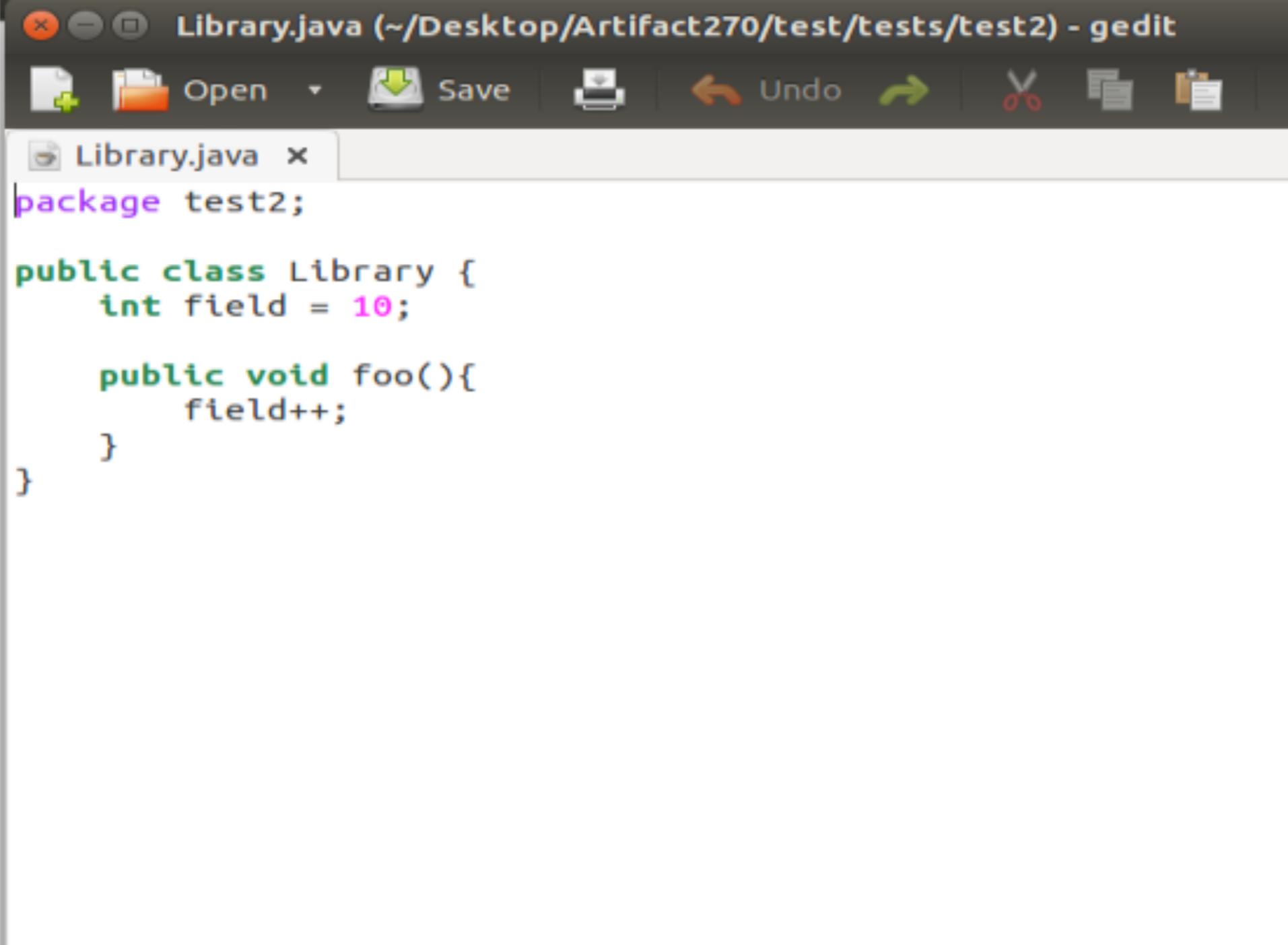
```
export CTRIGGER=OFF
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:. ./lib/concurrent.jar:. ./lib/rand
jsr305-1.3.9.jar:. ./lib/junit-4.11.jar:. ./lib/amen.jar:. ./lib/geotools_r.jar:. ./lib/
```

Replication Package



A screenshot of a terminal window titled 'narada@narada-VirtualBox: ~/Desktop/Artifact270'. The window contains the command 'source ./scripts/init.sh' which has been entered at the prompt.

Replication Package



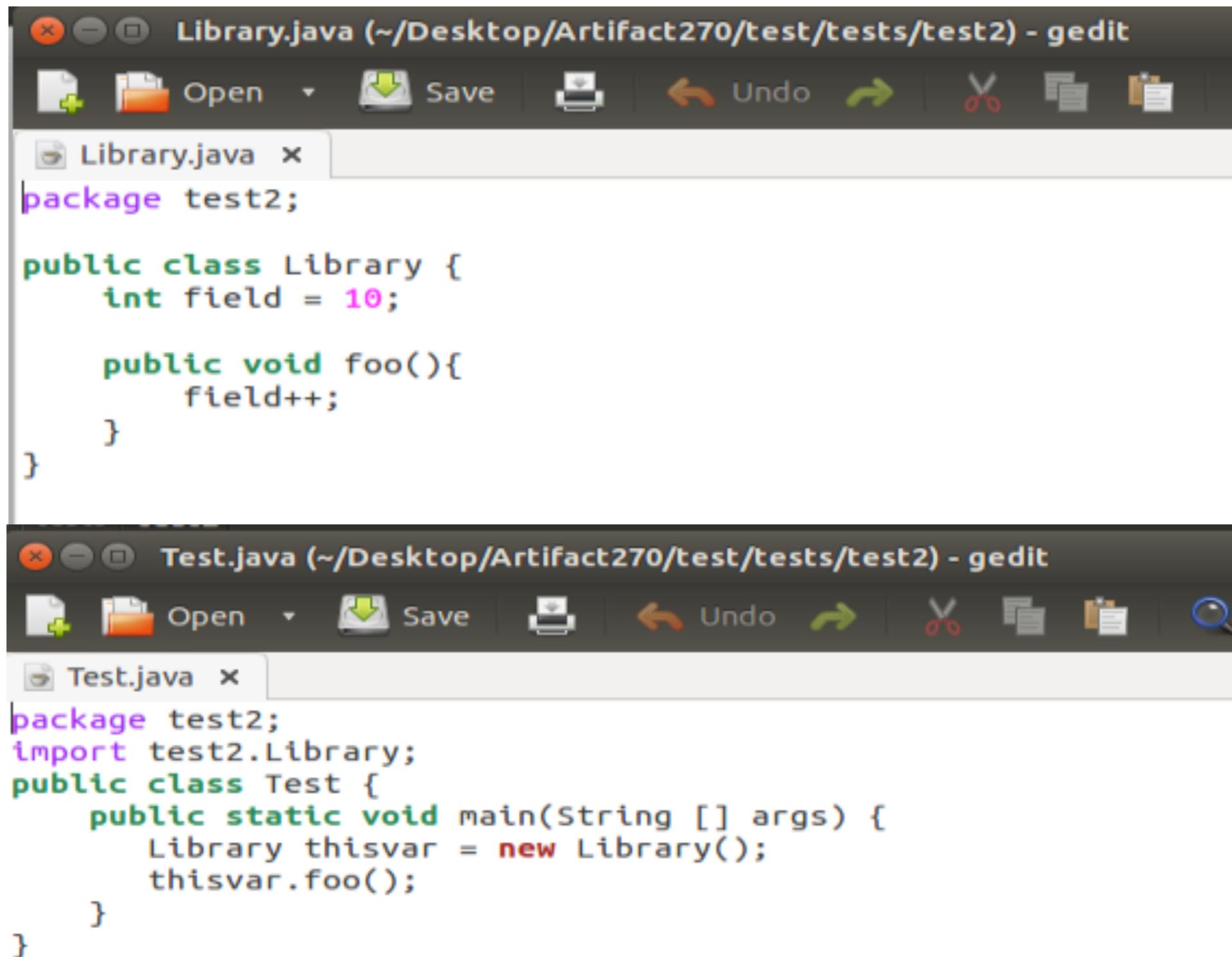
The screenshot shows a gedit window titled "Library.java (~/Desktop/Artifact270/test/tests/test2) - gedit". The window contains the following Java code:

```
package test2;

public class Library {
    int field = 10;

    public void foo(){
        field++;
    }
}
```

Replication Package



The image shows two windows of the gedit text editor side-by-side. Both windows have a dark interface with light-colored code.

Library.java (~/Desktop/Artifact270/test/tests/test2) - gedit

```
package test2;

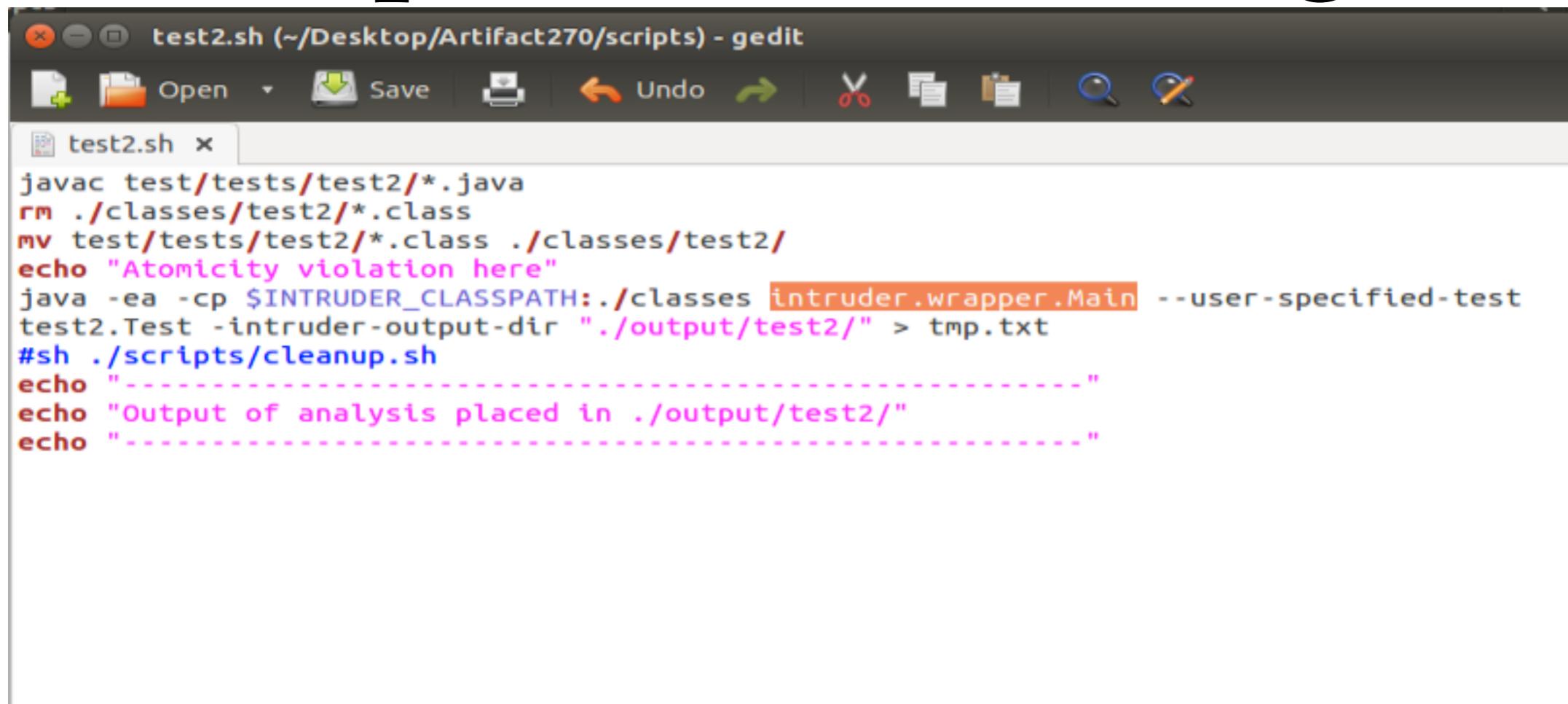
public class Library {
    int field = 10;

    public void foo(){
        field++;
    }
}
```

Test.java (~/Desktop/Artifact270/test/tests/test2) - gedit

```
package test2;
import test2.Library;
public class Test {
    public static void main(String [] args) {
        Library thisvar = new Library();
        thisvar.foo();
    }
}
```

Replication Package



A screenshot of the gedit text editor showing a file named "test2.sh". The file contains a shell script with the following content:

```
javac test/tests/test2/*.java
rm ./classes/test2/*.class
mv test/tests/test2/*.class ./classes/test2/
echo "Atomicity violation here"
java -ea -cp $INTRUDER_CLASSPATH:./classes intruder.wrapper.Main --user-specified-test
test2.Test -intruder-output-dir "./output/test2/" > tmp.txt
#sh ./scripts/cleanup.sh
echo -----
echo "Output of analysis placed in ./output/test2/"
echo -----
```

Replication Package

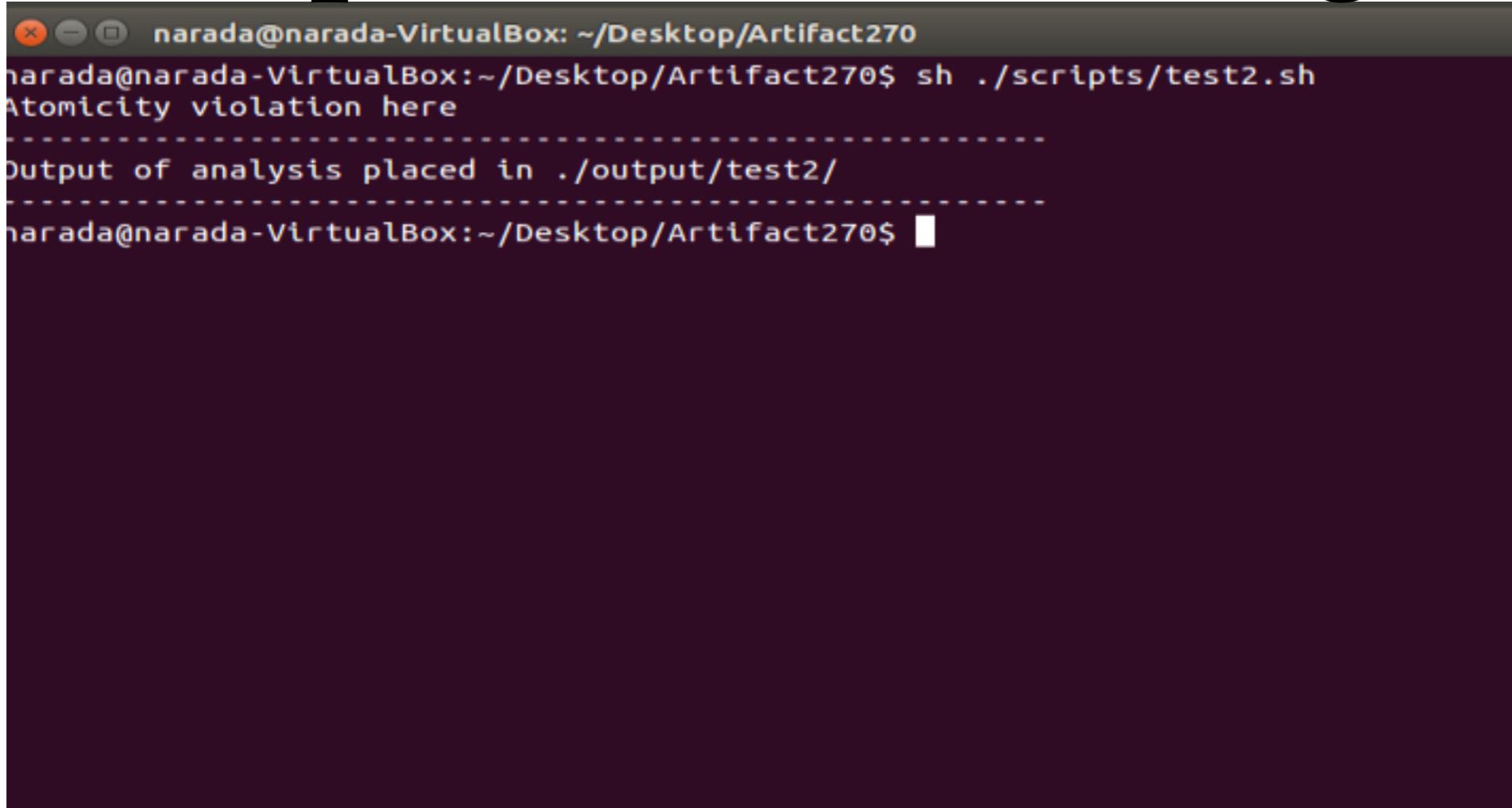
The screenshot shows a terminal window with the title "test2.sh (~/Desktop/Artifact270/scripts) - gedit". The window contains the following shell script:

```
javac test/tests/test2/*.java
rm ./classes/test2/*.class
mv test/tests/test2/*.class ./classes/test2/
echo "Atomicity violation here"
java -ea -cp $INTRUDER_CLASSPATH:./classes intruder.wrapper.Main --user-specified-test
test2.Test -intruder-output-dir "./output/test2/" > tmp.txt
#sh ./scripts/cleanup.sh
echo -----
echo "Output of analysis placed in ./output/test2/"
echo -----
```

The screenshot shows a terminal window with the title "test2.sh (~/Desktop/Artifact270/scripts) - gedit". The window contains the same shell script as the previous one:

```
javac test/tests/test2/*.java
rm ./classes/test2/*.class
mv test/tests/test2/*.class ./classes/test2/
echo "Atomicity violation here"
java -ea -cp $INTRUDER_CLASSPATH:./classes intruder.wrapper.Main --user-specified-test
test2.Test -intruder-output-dir "./output/test2/" > tmp.txt
#sh ./scripts/cleanup.sh
echo -----
echo "Output of analysis placed in ./output/test2/"
echo -----
```

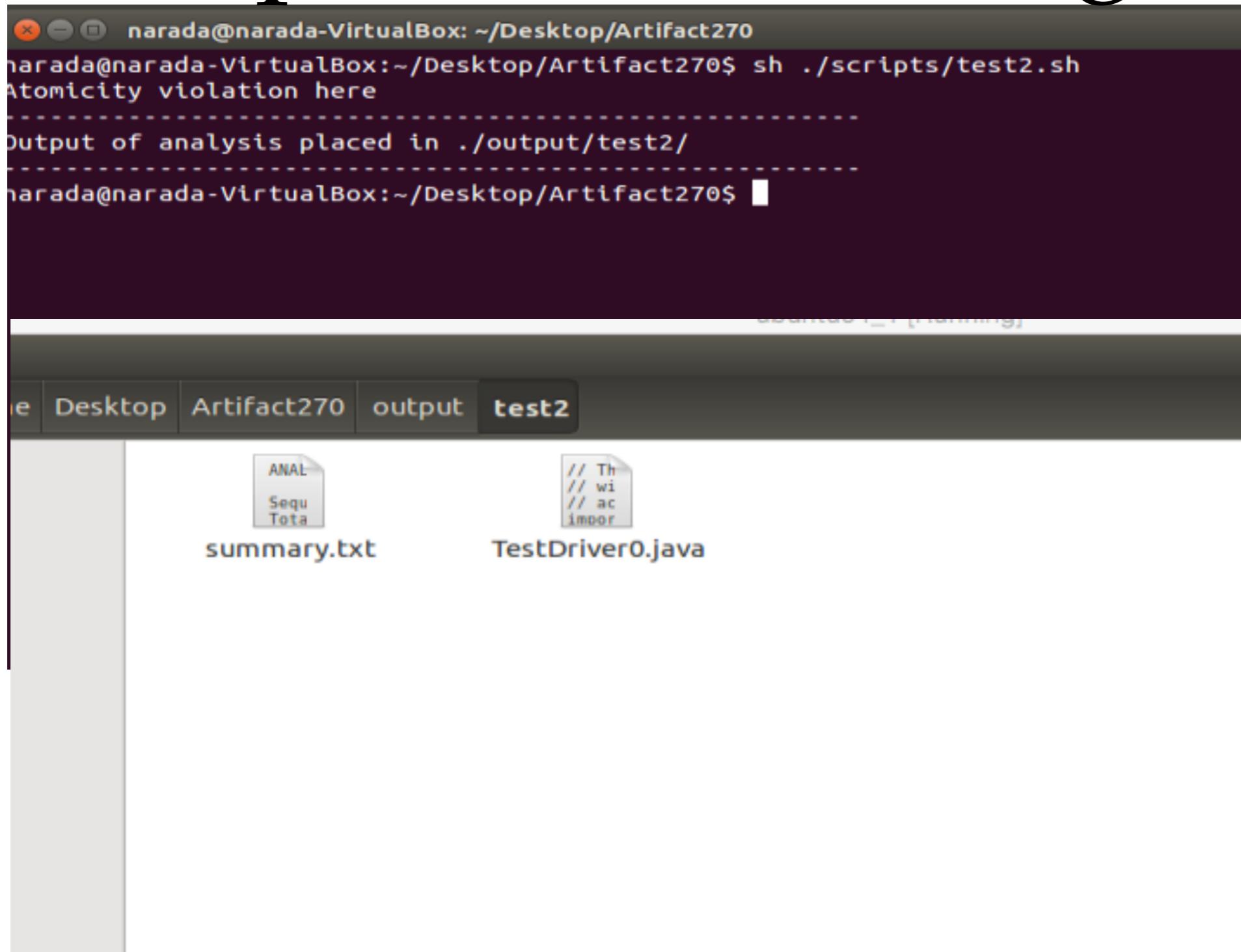
Replication Package



The screenshot shows a terminal window with the following text:

```
narada@narada-VirtualBox: ~/Desktop/Artifact270
narada@narada-VirtualBox:~/Desktop/Artifact270$ sh ./scripts/test2.sh
Atomicity violation here
-----
Output of analysis placed in ./output/test2/
-----
narada@narada-VirtualBox:~/Desktop/Artifact270$ █
```

Replication Package



Replication Package

The screenshot shows a Linux desktop environment with several windows open:

- A terminal window titled "narada@narada-VirtualBox: ~/Desktop/Artifact270" displays the command "sh ./scripts/test2.sh" and its output, which includes an atomicity violation message and analysis output.
- A file manager window titled "test2" shows a directory structure with files "summary.txt" and "TestDriver0.java".
- A gedit window titled "summary.txt (~/Desktop/Artifact270/output/test2) - gedit" displays the contents of the "summary.txt" file, which contains an "ANALYSIS SUMMARY:" section with synthesis time and test count information.

```
narada@narada-VirtualBox: ~/Desktop/Artifact270$ sh ./scripts/test2.sh
Atomicity violation here
-----
Output of analysis placed in ./output/test2/
-----
narada@narada-VirtualBox:~/Desktop/Artifact270$
```

File Manager:

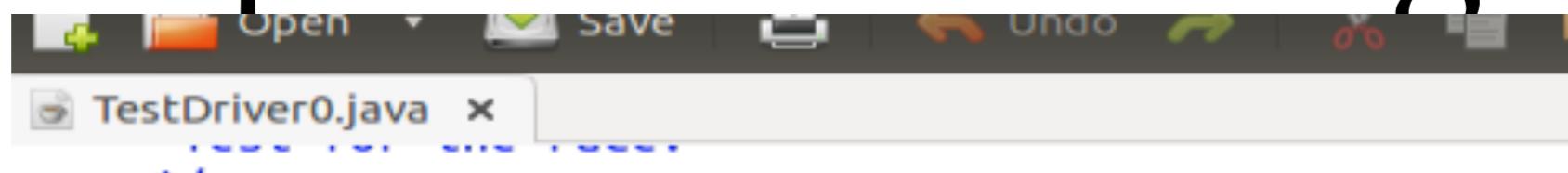
- summary.txt
- TestDriver0.java

gedit:

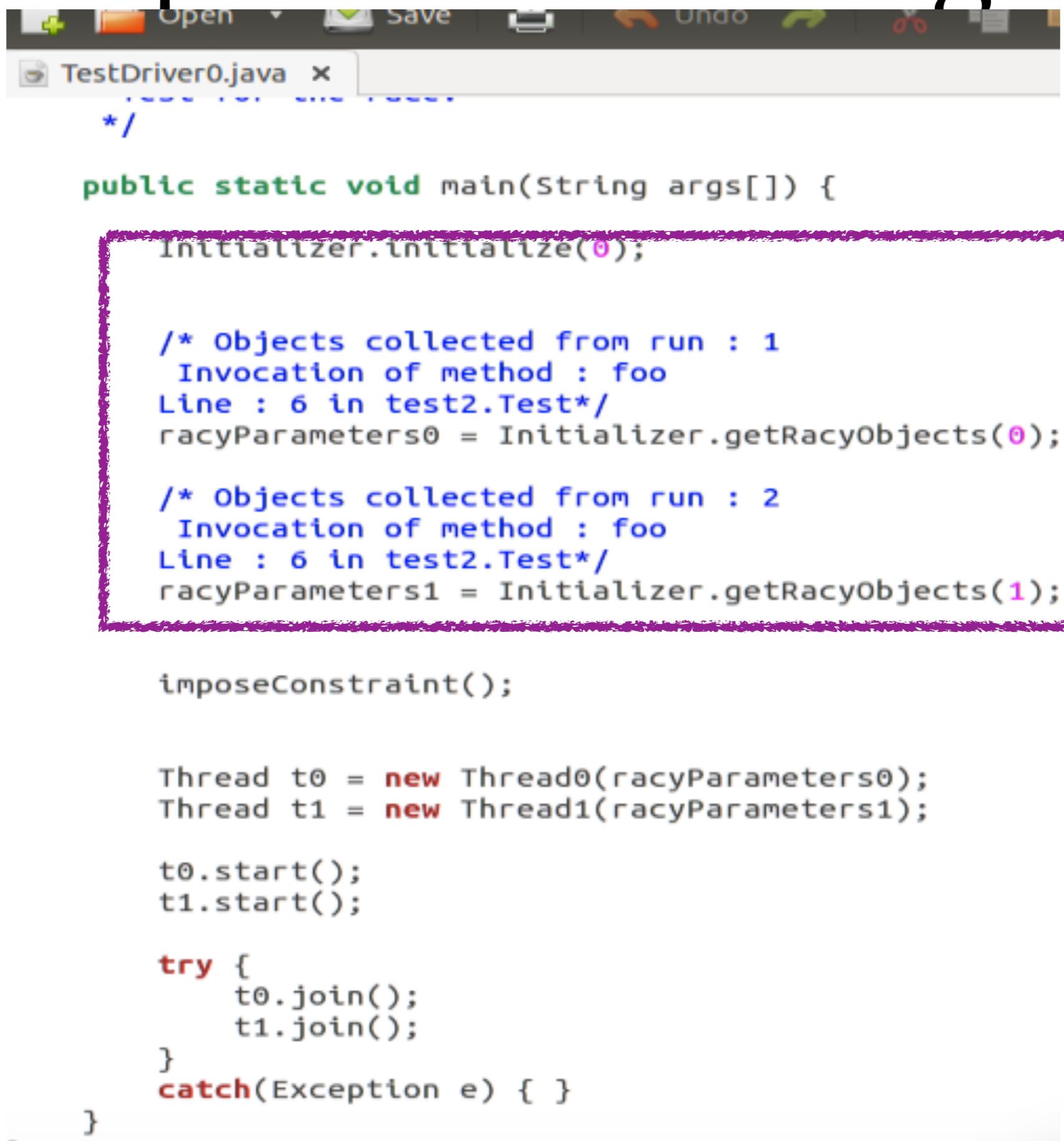
```
ANALYSIS SUMMARY:

Sequential test synthesis time : 5150 msec
Total number of tests : 1
```

Replication Package



Replication Package



The screenshot shows a Java IDE interface with a menu bar and toolbars at the top. Below the toolbar, a tab bar displays "TestDriver0.java". The main editor area contains Java code. A purple rectangular box highlights a specific section of the code, which is also shown in a larger, separate window below.

```
/*
 * TestDriver0.java
 */
public static void main(String args[]) {
    Initializer.initialize(0);

    /* Objects collected from run : 1
       Invocation of method : foo
       Line : 6 in test2.Test*/
    racyParameters0 = Initializer.getRacyObjects(0);

    /* Objects collected from run : 2
       Invocation of method : foo
       Line : 6 in test2.Test*/
    racyParameters1 = Initializer.getRacyObjects(1);

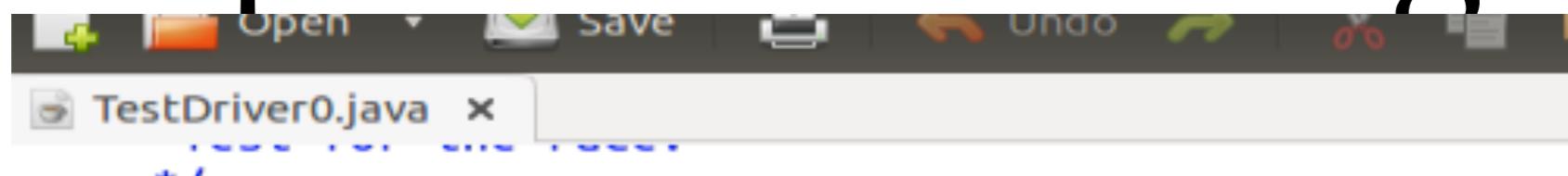
    imposeConstraint();

    Thread t0 = new Thread0(racyParameters0);
    Thread t1 = new Thread1(racyParameters1);

    t0.start();
    t1.start();

    try {
        t0.join();
        t1.join();
    }
    catch(Exception e) { }
}
```

Replication Package

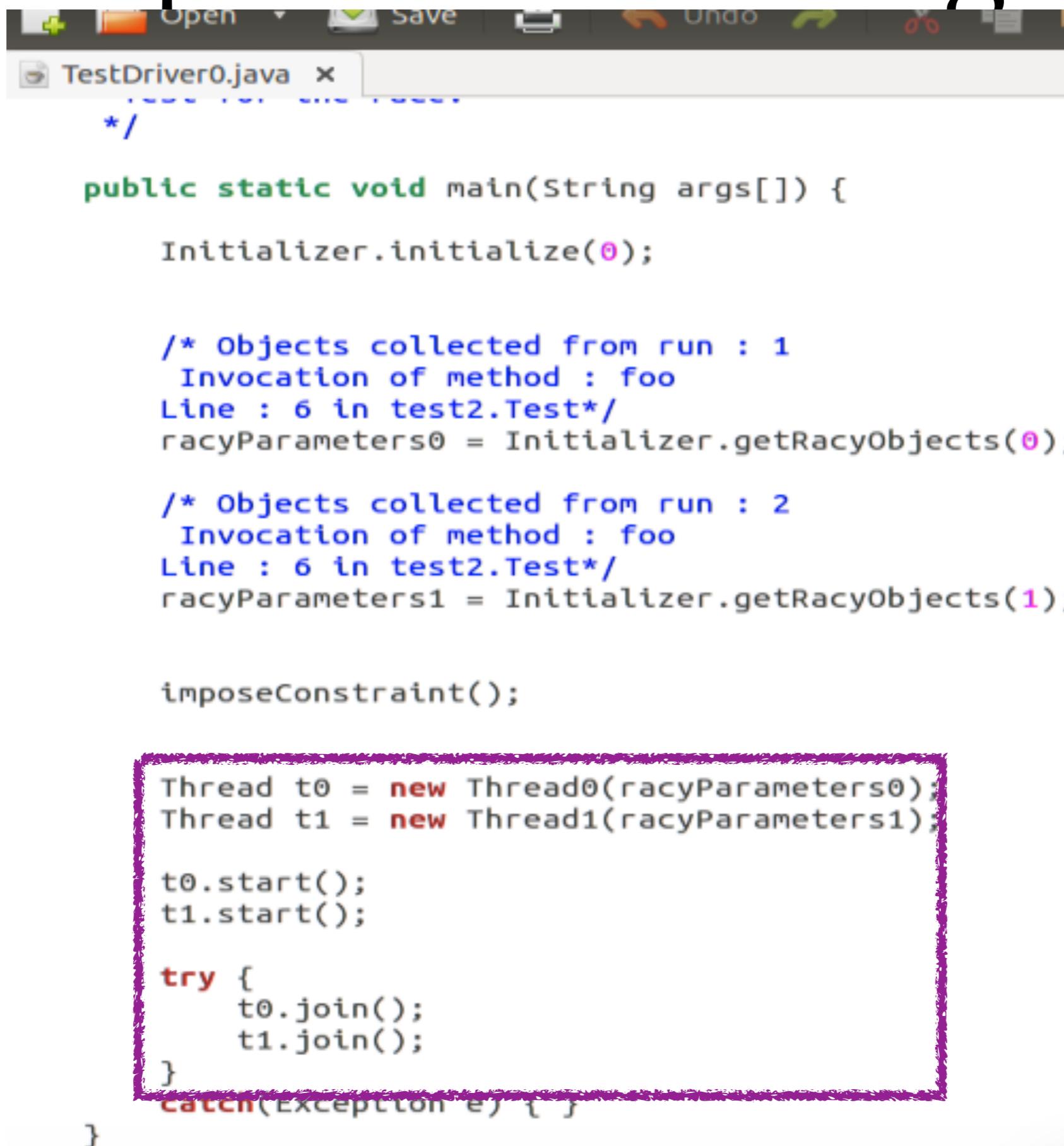


```
Thread t0 = new Thread0(racyParameters0);  
Thread t1 = new Thread1(racyParameters1);
```

```
t0.start();  
t1.start();
```

```
try {  
    t0.join();  
    t1.join();  
}  
catch(Exception e) { }
```

Replication Package



The screenshot shows a Java IDE interface with a menu bar and toolbars at the top. Below the toolbar, a tab bar displays "TestDriver0.java". The main editor area contains Java code. A portion of the code is highlighted with a purple rectangular border.

```
/*
 * TestDriver0.java
 */
public static void main(String args[]) {
    Initializer.initialize(0);

    /* Objects collected from run : 1
     * Invocation of method : foo
     * Line : 6 in test2.Test*/
    racyParameters0 = Initializer.getRacyObjects(0);

    /* Objects collected from run : 2
     * Invocation of method : foo
     * Line : 6 in test2.Test*/
    racyParameters1 = Initializer.getRacyObjects(1);

    imposeConstraint();

    Thread t0 = new Thread0(racyParameters0);
    Thread t1 = new Thread1(racyParameters1);

    t0.start();
    t1.start();

    try {
        t0.join();
        t1.join();
    }
    catch(Exception e) { }
}
```

Replication Package

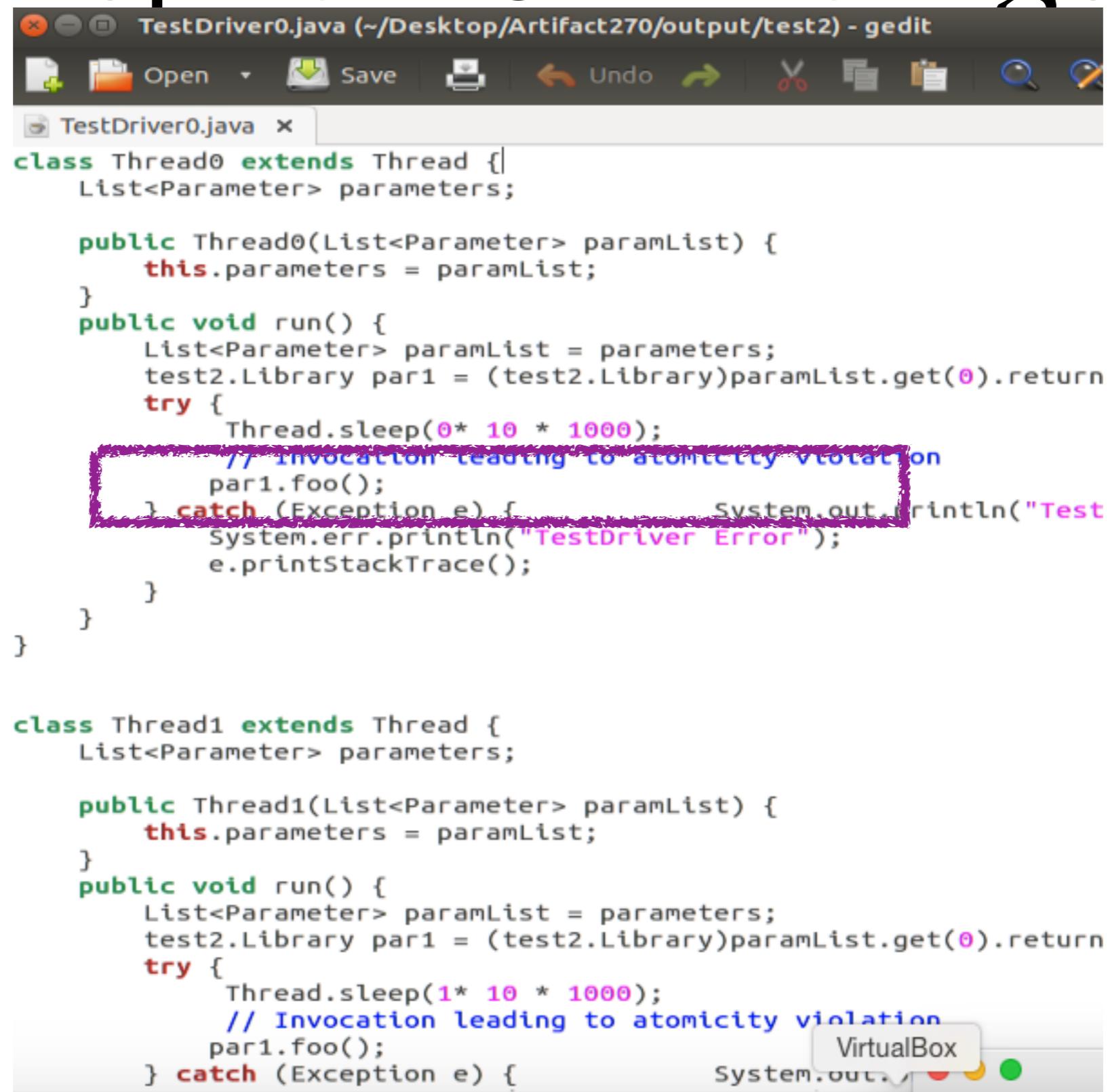
```
TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit
File Open Save Print Undo Redo Cut Copy Find Replace
TestDriver0.java ×
class Thread0 extends Thread {
    List<Parameter> parameters;

    public Thread0(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(0* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("Test");
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}

class Thread1 extends Thread {
    List<Parameter> parameters;

    public Thread1(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(1* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("Test");
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}
```

Replication Package



```
TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit
File Open Save Print Undo Redo Cut Copy Find Replace
TestDriver0.java ×
class Thread0 extends Thread {
    List<Parameter> parameters;

    public Thread0(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(0* 10 * 1000);
            // invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("Test");
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}

class Thread1 extends Thread {
    List<Parameter> parameters;

    public Thread1(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(1* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("Test");
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}
```

Replication Package

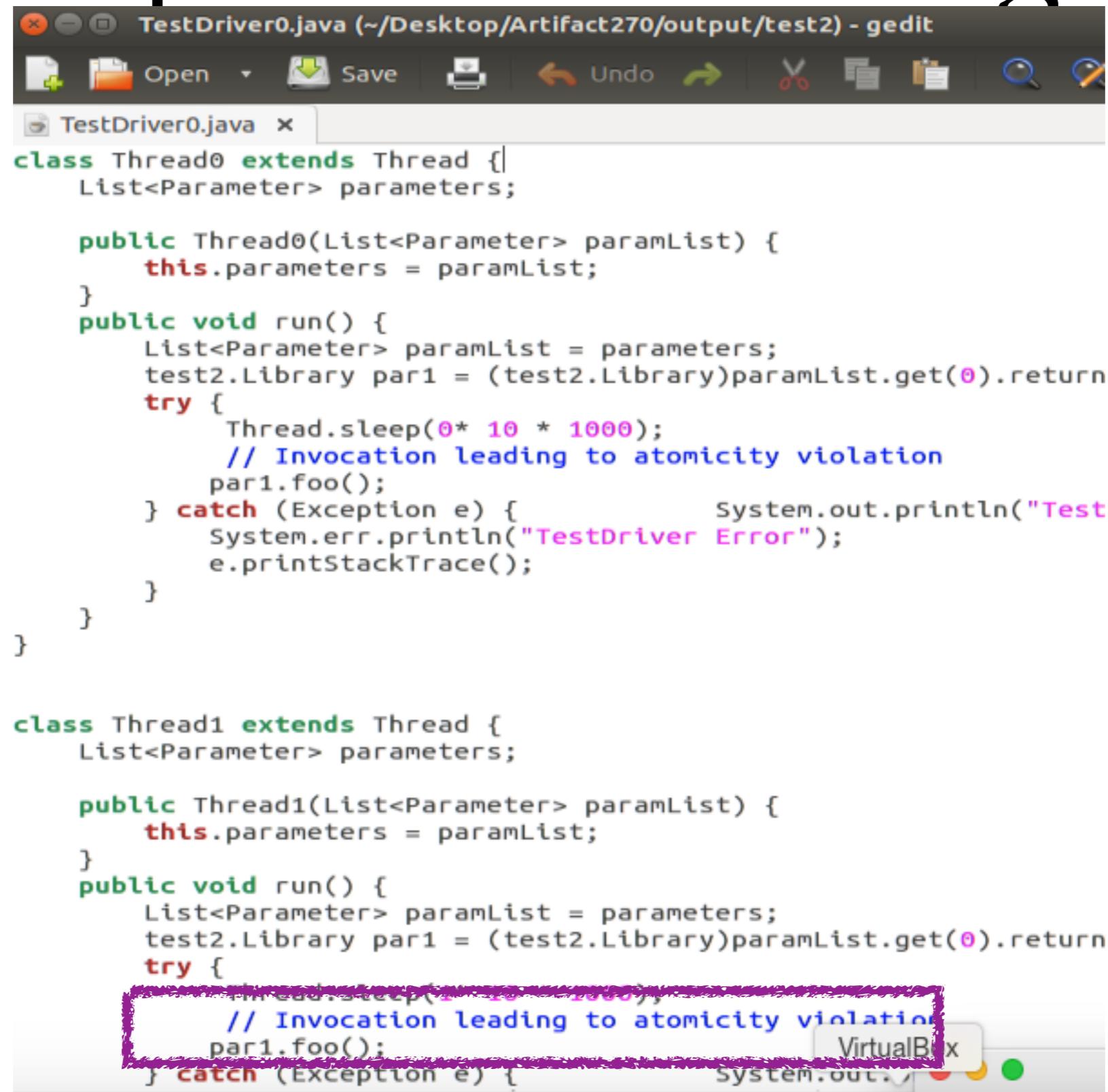
```
TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit
File Open Save Print Undo Redo Cut Copy Find Replace
TestDriver0.java ×
class Thread0 extends Thread {
    List<Parameter> parameters;

    public Thread0(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(0* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {           System.out.println("Test
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}

class Thread1 extends Thread {
    List<Parameter> parameters;

    public Thread1(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(1* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}
```

Replication Package



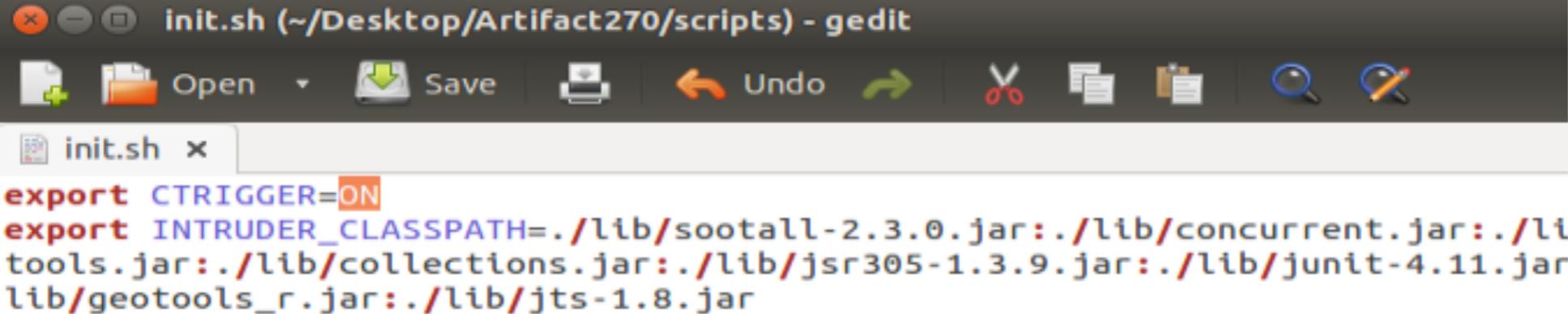
```
TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit
File Open Save Print Undo Redo Cut Copy Find Replace
TestDriver0.java x
class Thread0 extends Thread {
    List<Parameter> parameters;

    public Thread0(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(0* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("Test");
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}

class Thread1 extends Thread {
    List<Parameter> parameters;

    public Thread1(List<Parameter> paramList) {
        this.parameters = paramList;
    }
    public void run() {
        List<Parameter> paramList = parameters;
        test2.Library par1 = (test2.Library)paramList.get(0).return
        try {
            Thread.sleep(0* 10 * 1000);
            // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {
            System.out.println("Test");
            System.err.println("TestDriver Error");
            e.printStackTrace();
        }
    }
}
```

Replication Package



The screenshot shows a terminal window titled "init.sh (~/Desktop/Artifact270/scripts) - gedit". The window contains the following code:

```
export CTRIGGER=ON
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:../lib/concurrent.jar:./
tools.jar:../lib/collections.jar:../lib/jsr305-1.3.9.jar:../lib/junit-4.11.jar
lib/geotools_r.jar:../lib/jts-1.8.jar
```

Replication Package

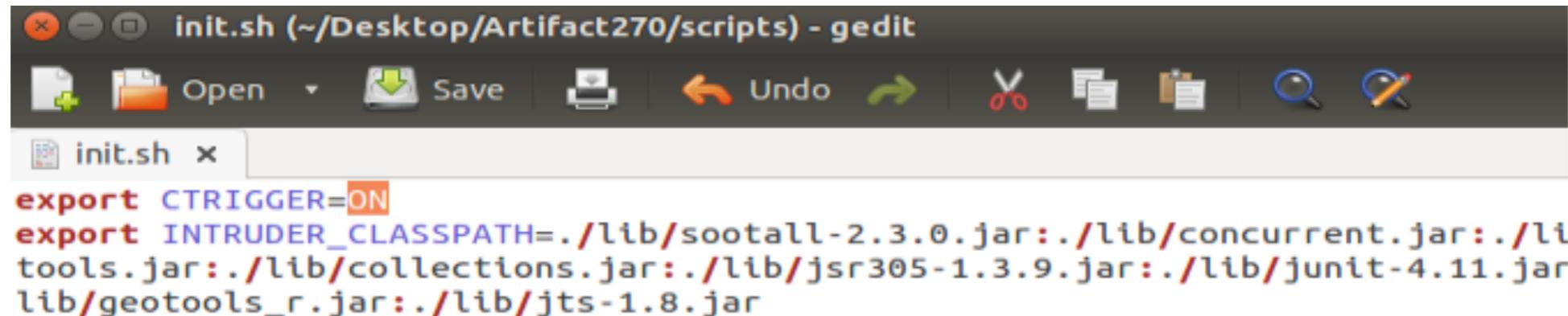
The image shows a Linux desktop environment with two windows open. The top window is a terminal window titled "init.sh (~/Desktop/Artifact270/scripts) - gedit". It contains the following shell script code:

```
export CTRIGGER=ON
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:../lib/concurrent.jar:./
tools.jar:../lib/collections.jar:../lib/jsr305-1.3.9.jar:../lib/junit-4.11.jar
lib/geotools_r.jar:../lib/jts-1.8.jar
```

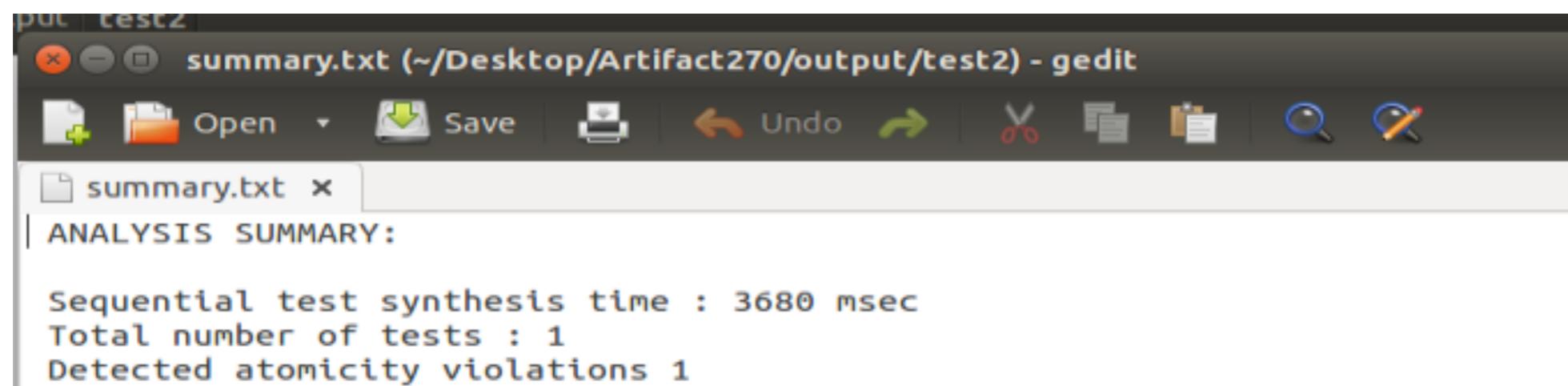
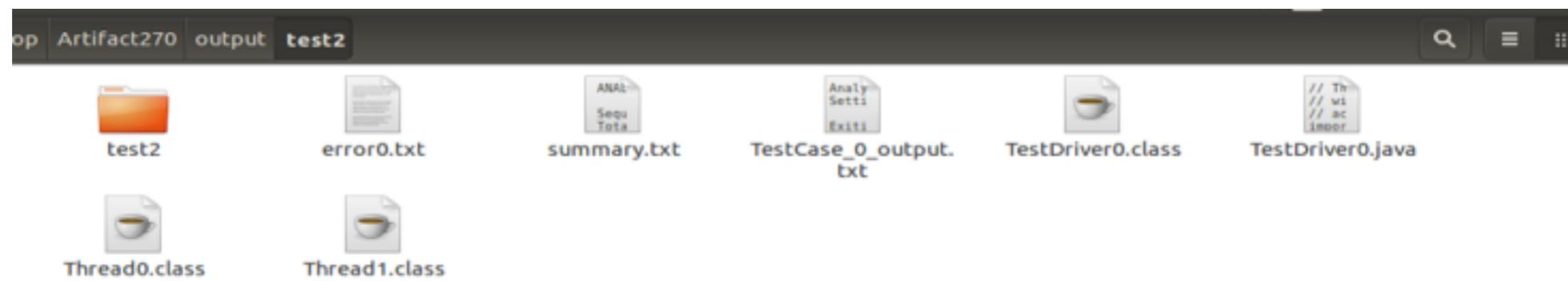
The bottom window is a file manager showing the contents of a directory named "test2". The directory structure is as follows:

- test2 (folder)
- error0.txt
- summary.txt
- TestCase_0_output.txt
- TestDriver0.class
- TestDriver0.java
- Thread0.class
- Thread1.class

Replication Package



```
init.sh (~/Desktop/Artifact270/scripts) - gedit
File Open Save Undo Redo Cut Copy Find Replace
init.sh x
export CTRIGGER=ON
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:./lib/concurrent.jar:./lib/tools.jar:./lib/collections.jar:./lib/jsr305-1.3.9.jar:./lib/junit-4.11.jar
lib/geotools_r.jar:./lib/jts-1.8.jar
```



```
put test2
summary.txt (~/Desktop/Artifact270/output/test2) - gedit
File Open Save Undo Redo Cut Copy Find Replace
summary.txt x
ANALYSIS SUMMARY:

Sequential test synthesis time : 3680 msec
Total number of tests : 1
Detected atomicity violations 1
```

Replication Package

```
TestCase_0_output.txt x
Analysis class javato.activetesting.CTrigger-1
Setting current invocation to foo to be collected at line 6

Exiting Sequential Invocation
Prinitng error stack for her java.lang.reflect.InvocationTargetException got class 1

Setting current invocation to foo to be collected at line 6

Exiting Sequential Invocation
Prinitng error stack for her java.lang.reflect.InvocationTargetException got class 1

1. MEMORY: 42949672968
P: read at: test2/Library.java#7 by (12)
C: write at: test2/Library.java#7 by (12)
R: write at: test2/Library.java#7 by (13)

-----
Total PCR pairs found: 4
No of unique PCR pairs that cause atomicity violation: 1
```

Summary

- ❖ Designed a *directed* approach to synthesize atomicity violation inducing tests
- ❖ INTRUDER: tool that incorporates the proposed design
- ❖ Validated using open source java libraries
- ❖ Helped detect harmful atomicity violations
 - ❖ Negligible time overhead
- ❖ Tool: <http://www.csa.iisc.ernet.in/~sss/tools/intruder/>