Synthesizing Tests for Detecting Atomicity Violations*

Malavika Samak *and Murali Krishna Ramanathan*
Indian Institute of Science, Bangalore

*** Replication package evaluated**

Introduction

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)

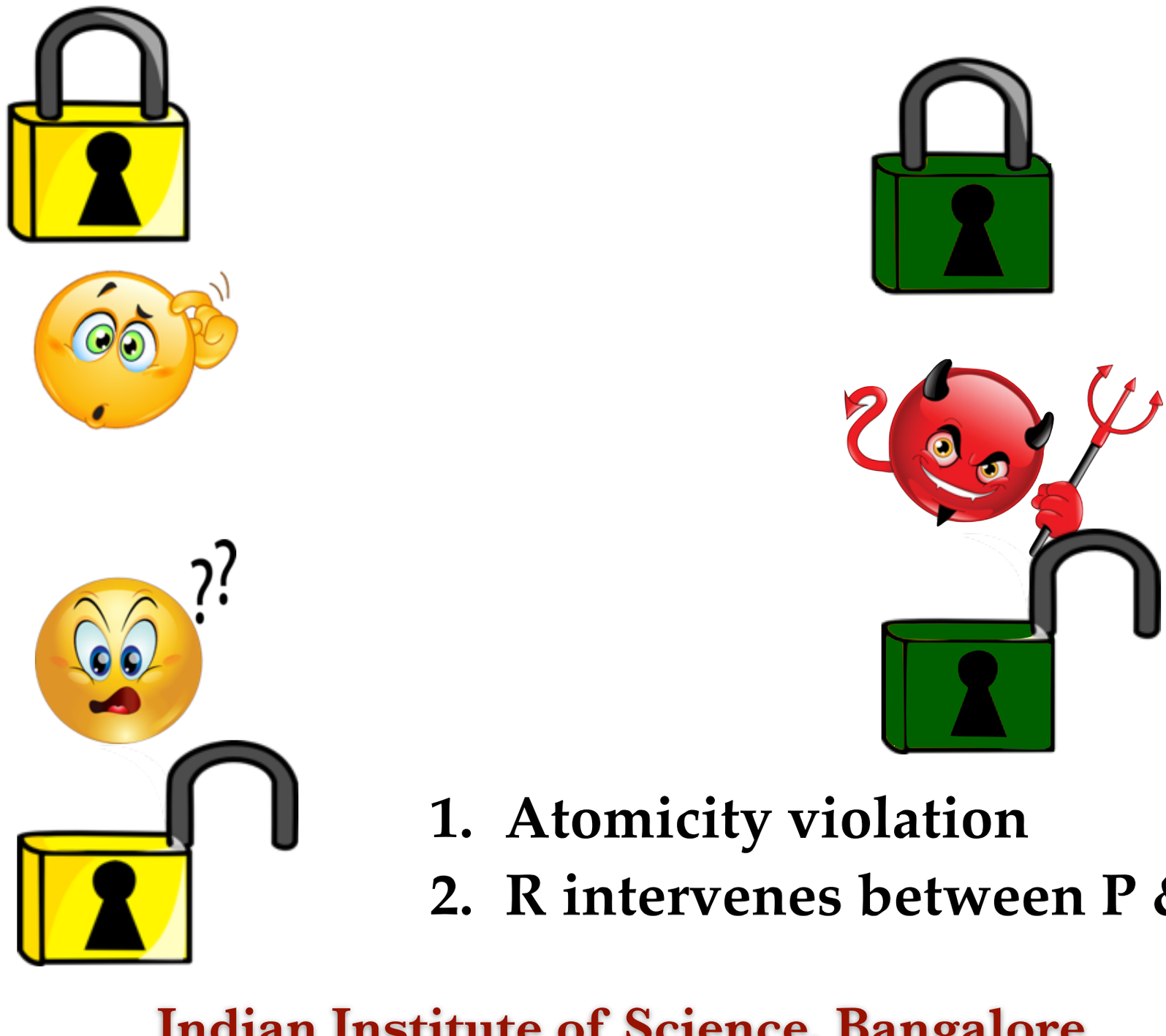
Current (C)

T_2

Remote (R)

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

What is an atomicity violation?



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

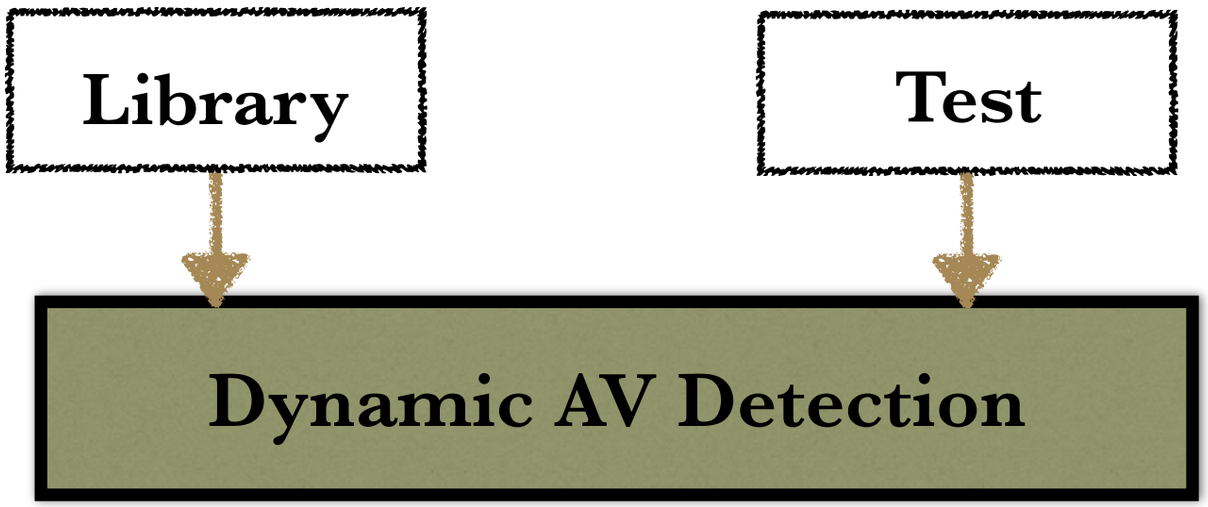
Dynamic Atomicity Violation Detection

Library

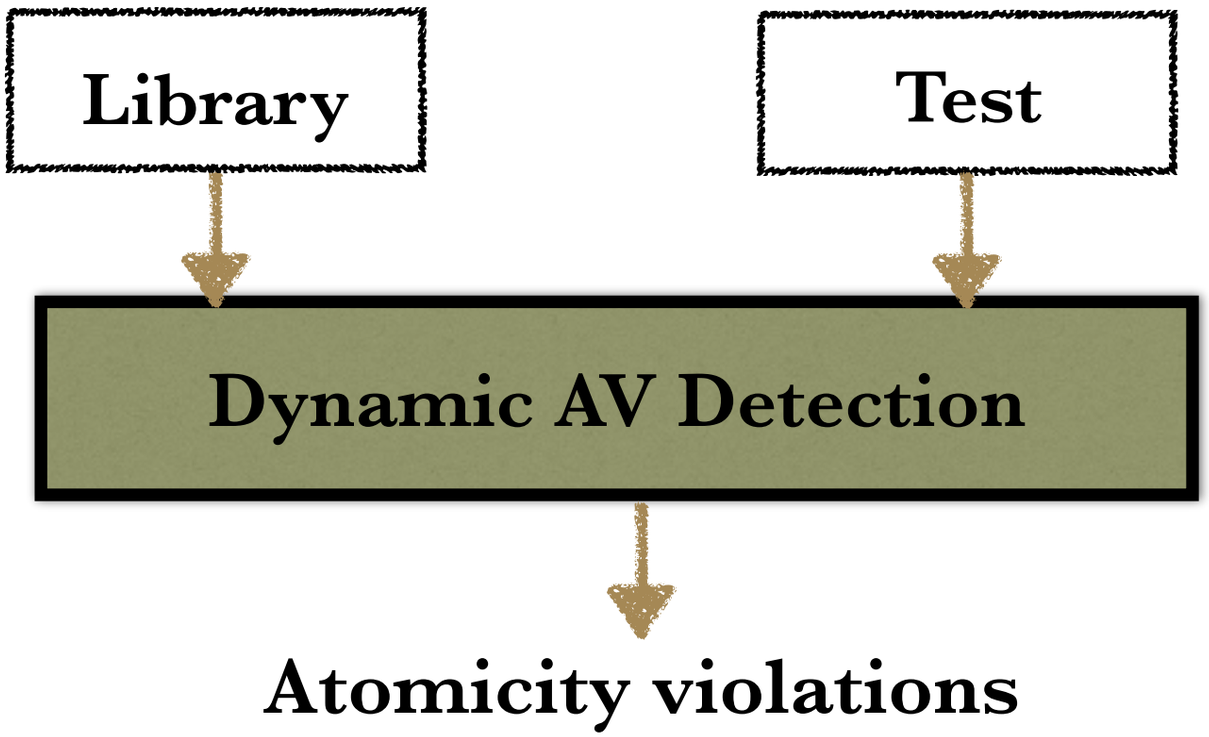
Test

Indian Institute of Science, Bangalore

Dynamic Atomicity Violation Detection

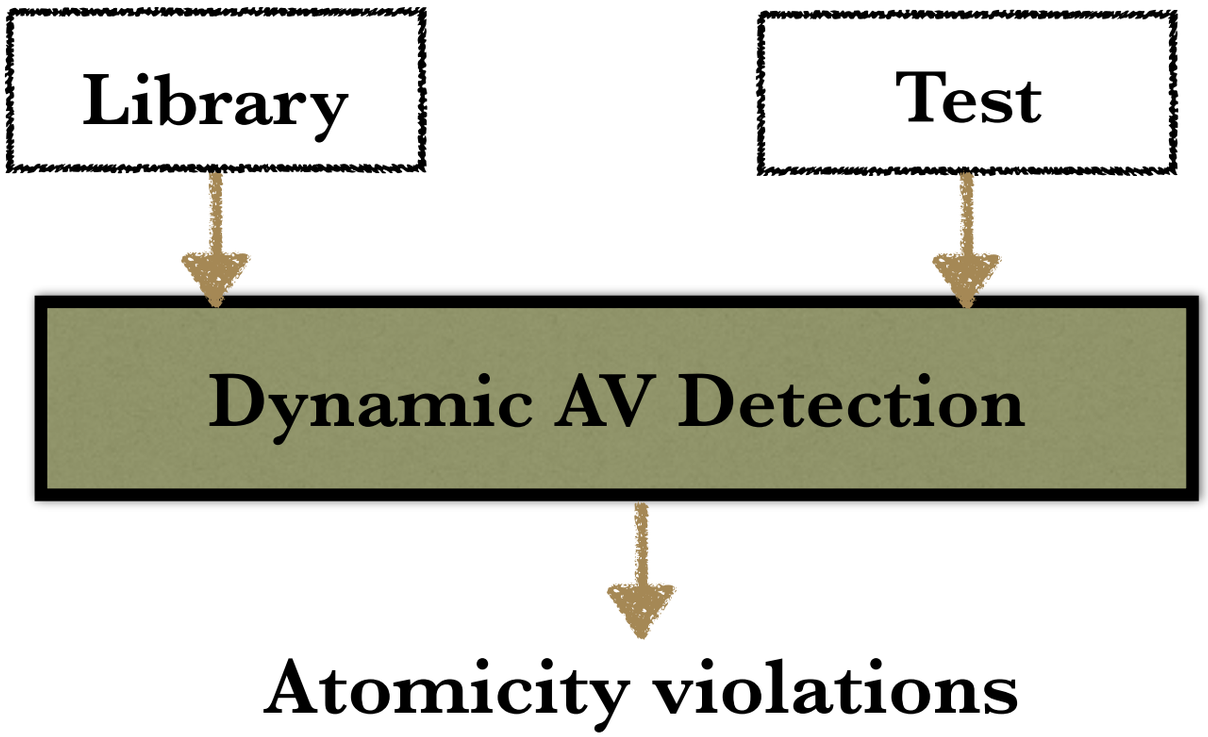


Dynamic Atomicity Violation Detection



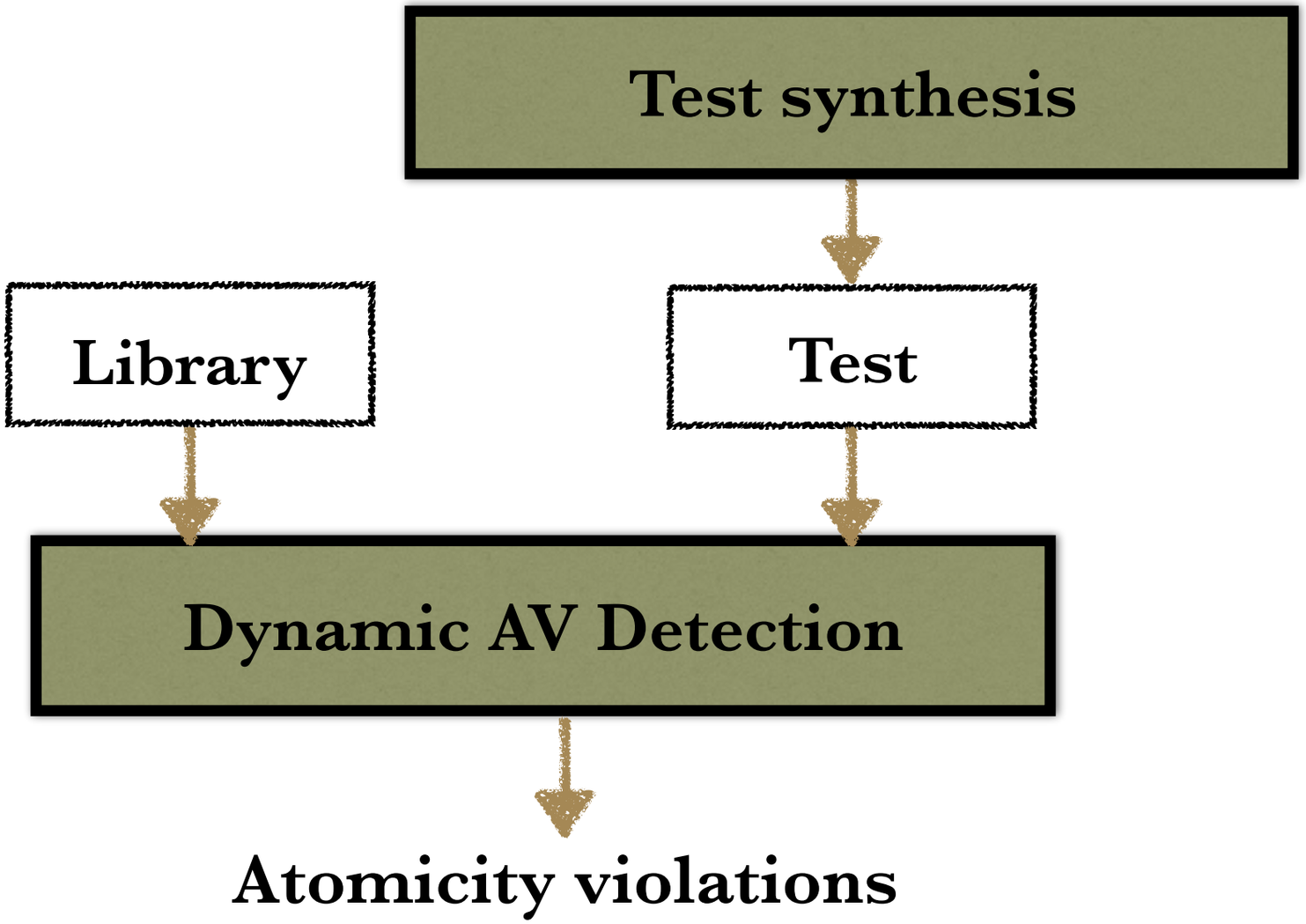
- **Atomizer,** POPL'04
- **AtomFuzzer,** FSE'08
- **Velodrome,** PLDI'08
- **CTrigger,** ASPLOS'09
- **DoubleChecker,** PLDI'14

Dynamic Atomicity Violation Detection

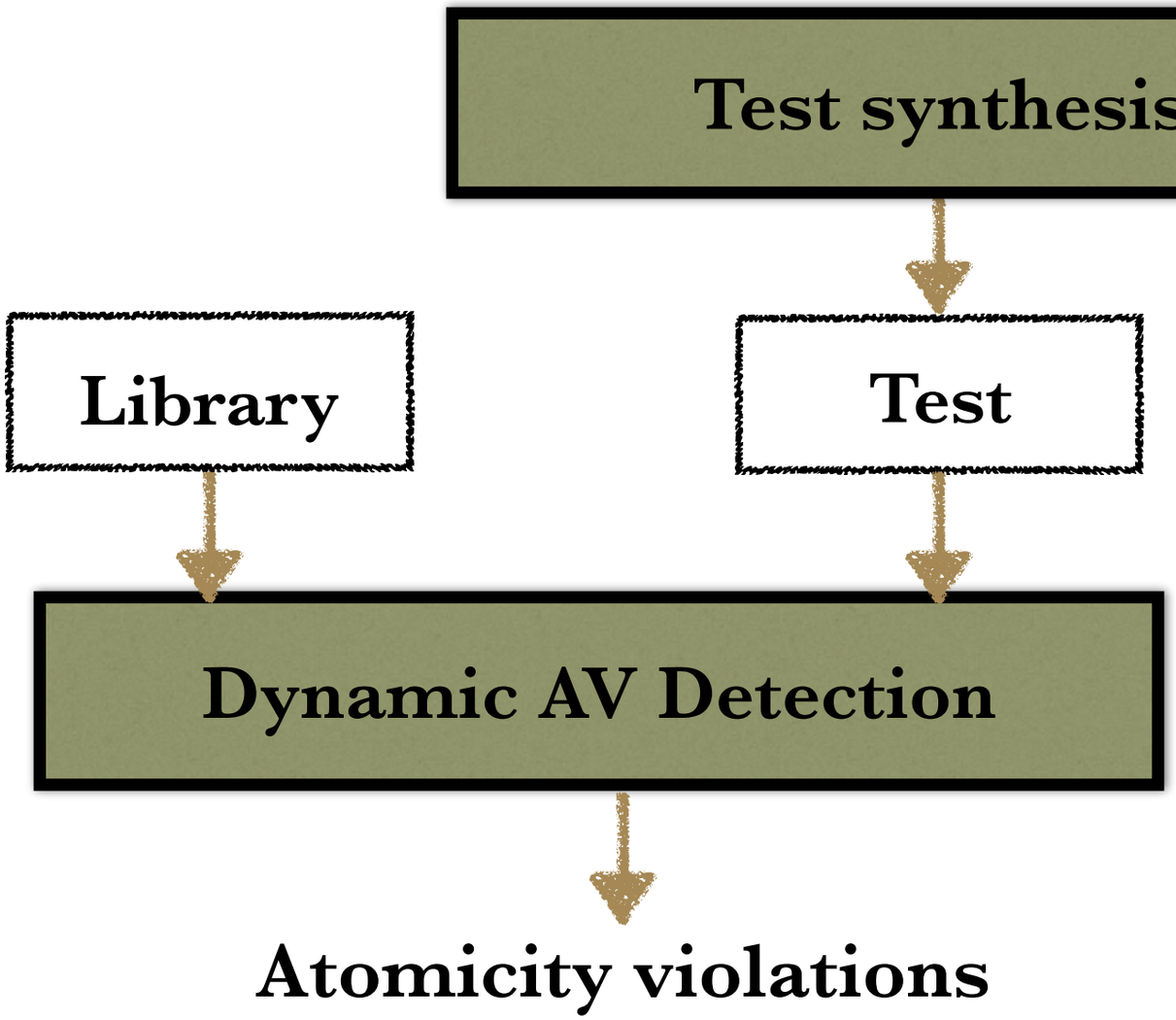


- **Significant manual effort**
- **Higher false negatives**

Dynamic Atomicity Violation Detection

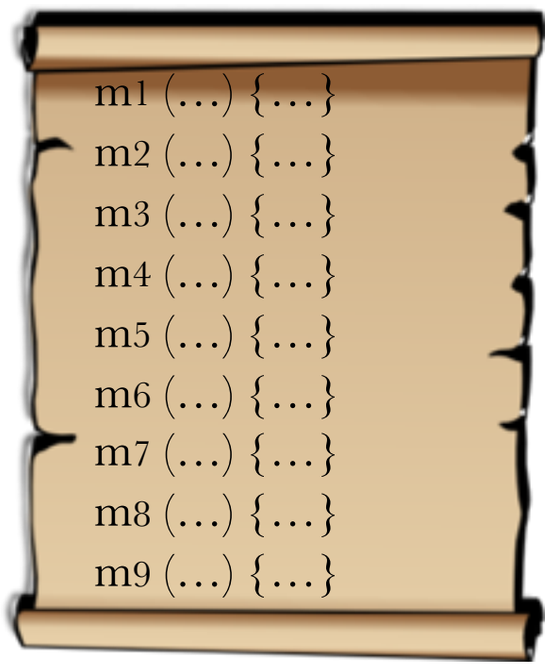


Dynamic Atomicity Violation Detection



Goal of this work

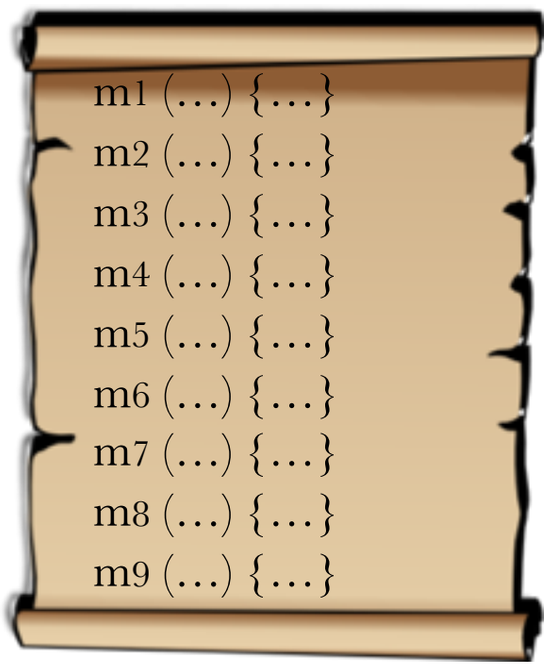
Randomized Test Generation



Library

Possible Parameters

Randomized Test Generation



Library

Possible Parameters

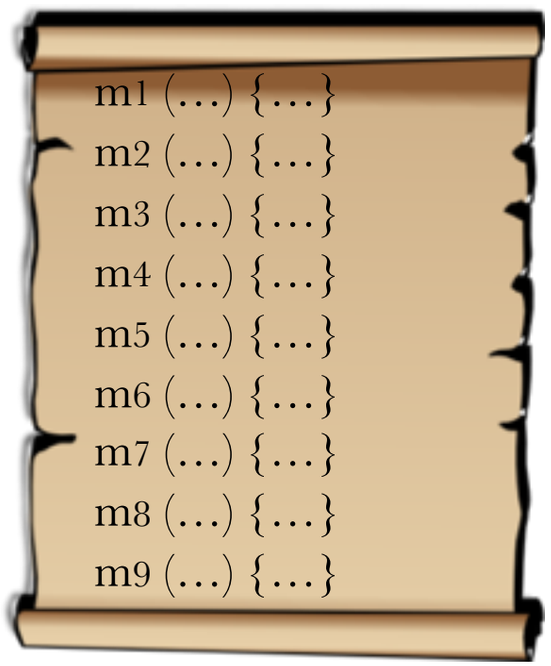
t_1: m1( )

t_2: m2( )

m1( )
m2( )

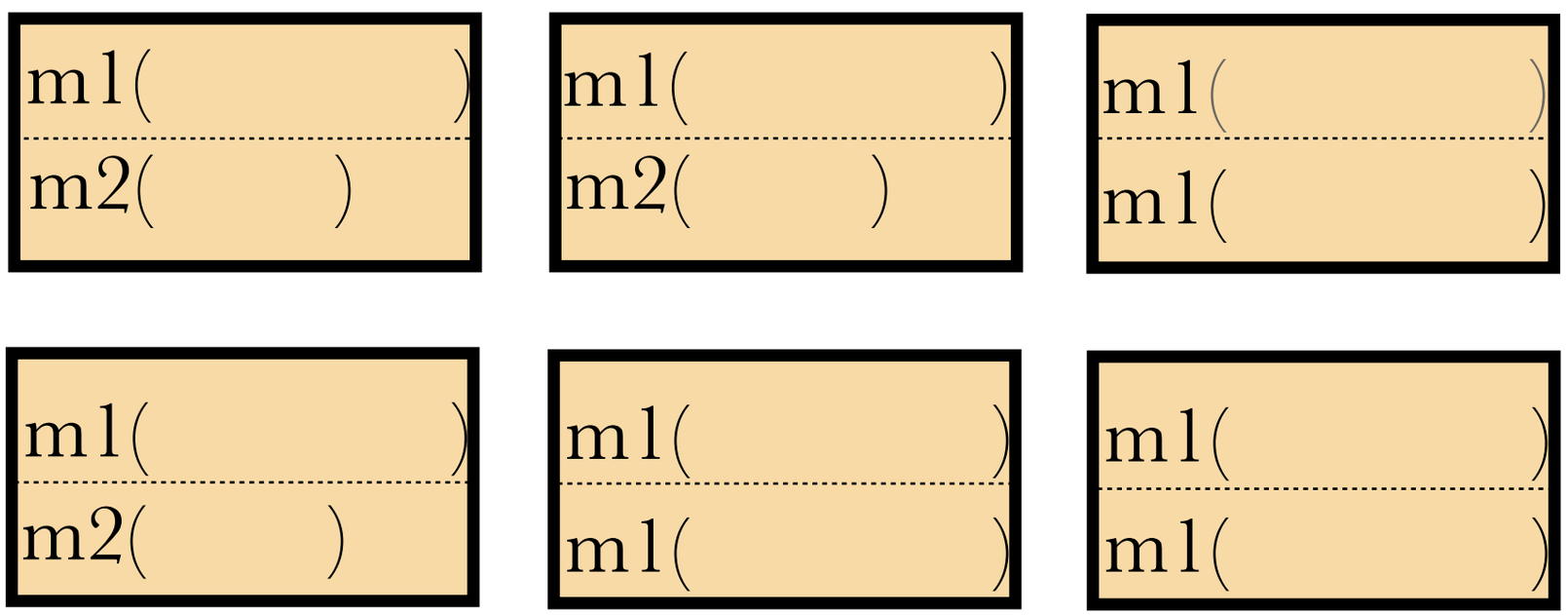
m1( )
m1( )

Randomized Test Generation

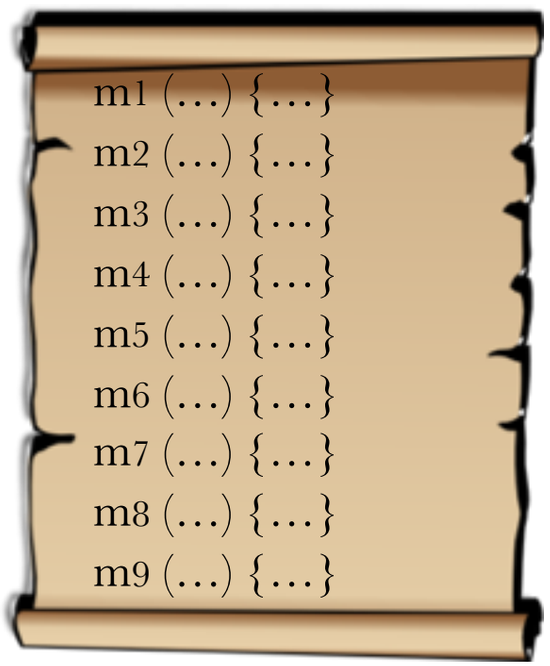


Library

Possible Parameters



Randomized Test Generation



Library

Possible Parameters

t_1: m1( ) / t_2: m2( )

t_1: m1( ) / t_2: m2( )

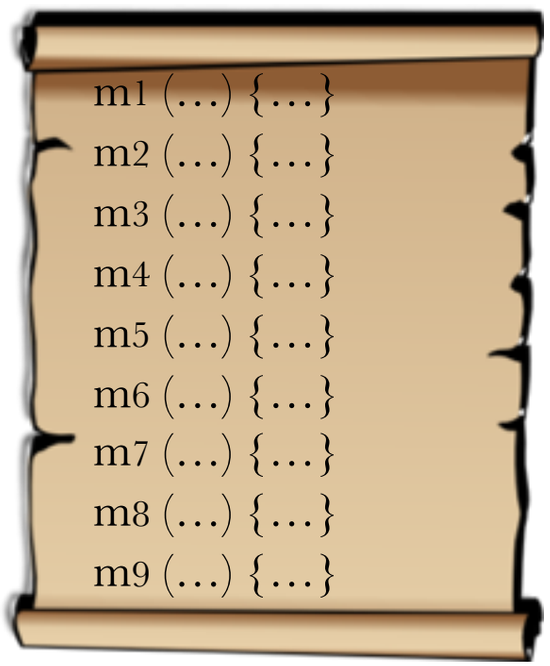
t_1: m1( ) / t_2: m1( )

t_1: m1( ) / t_2: m2( )

t_1: m1( ) / t_2: m1( )

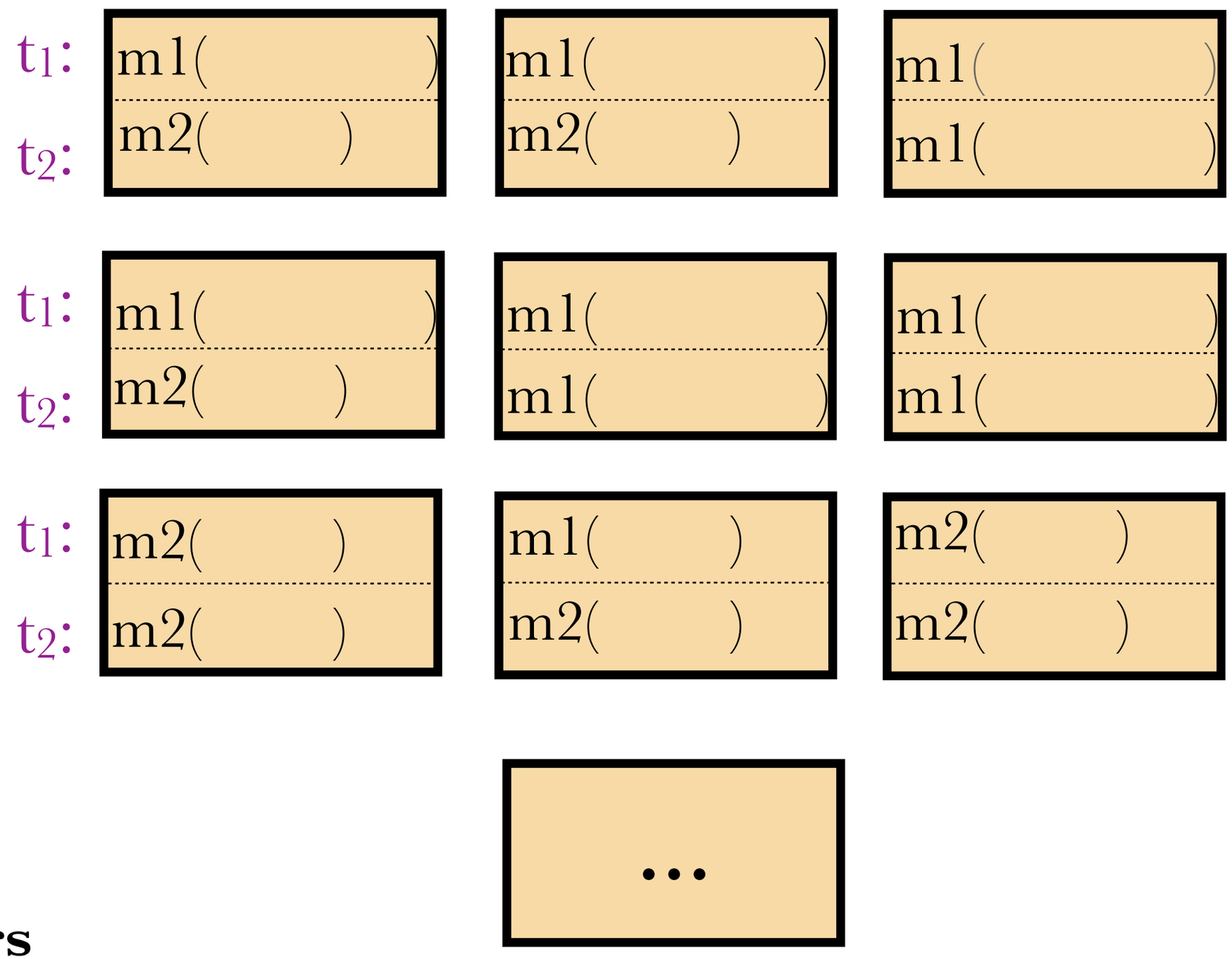
t_1: m1( ) / t_2: m1( )

Randomized Test Generation

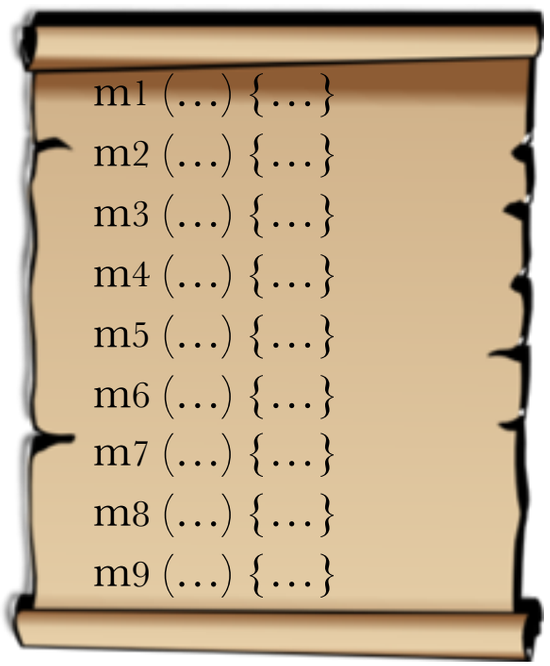


Library

Possible Parameters

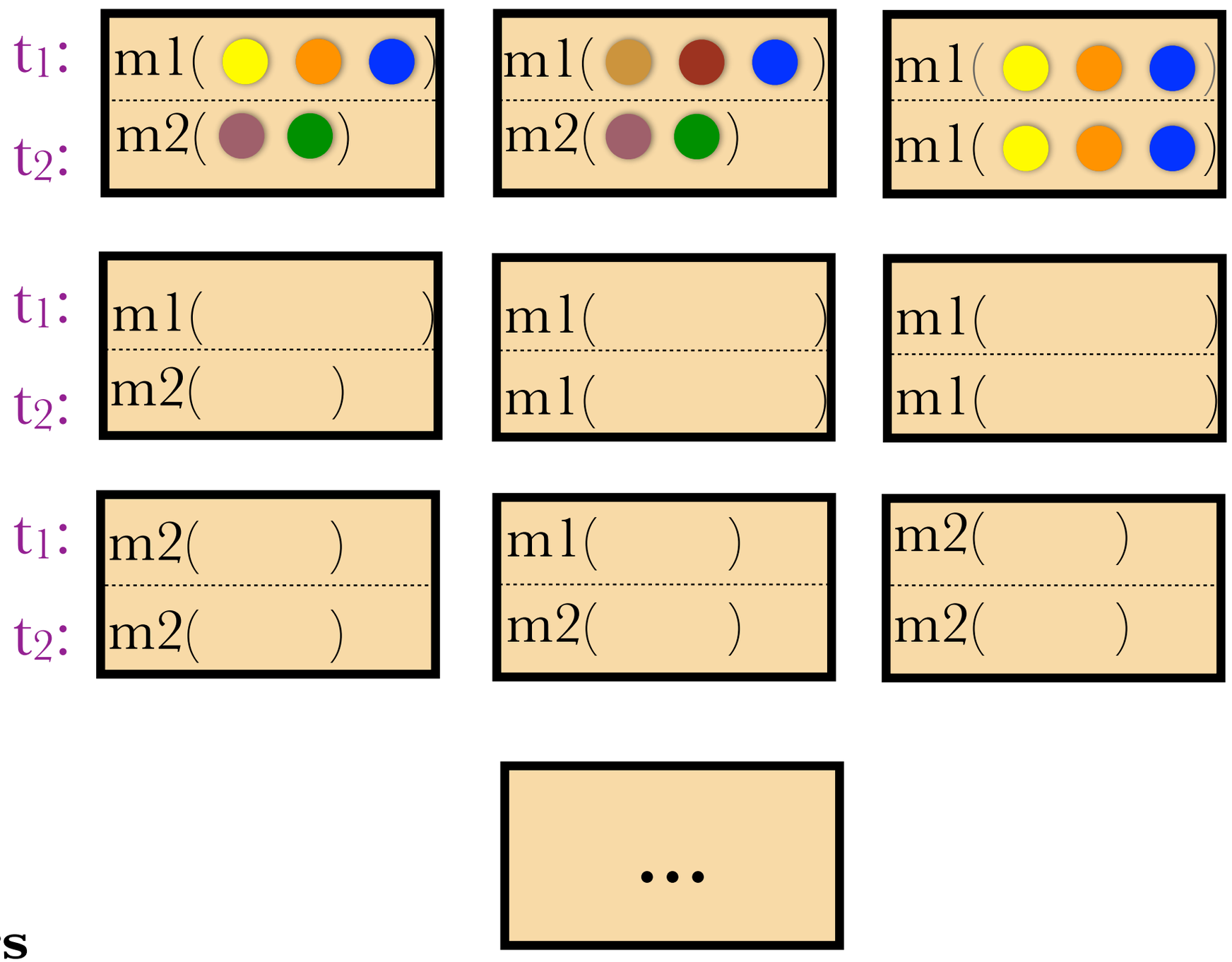


Randomized Test Generation

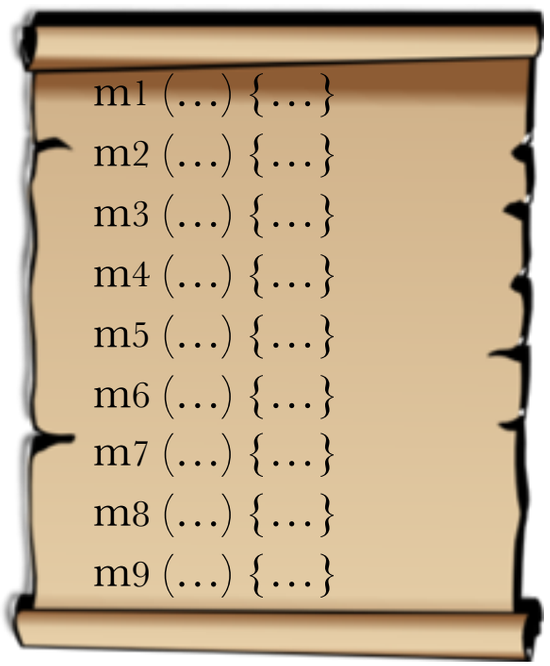


Library

Possible Parameters

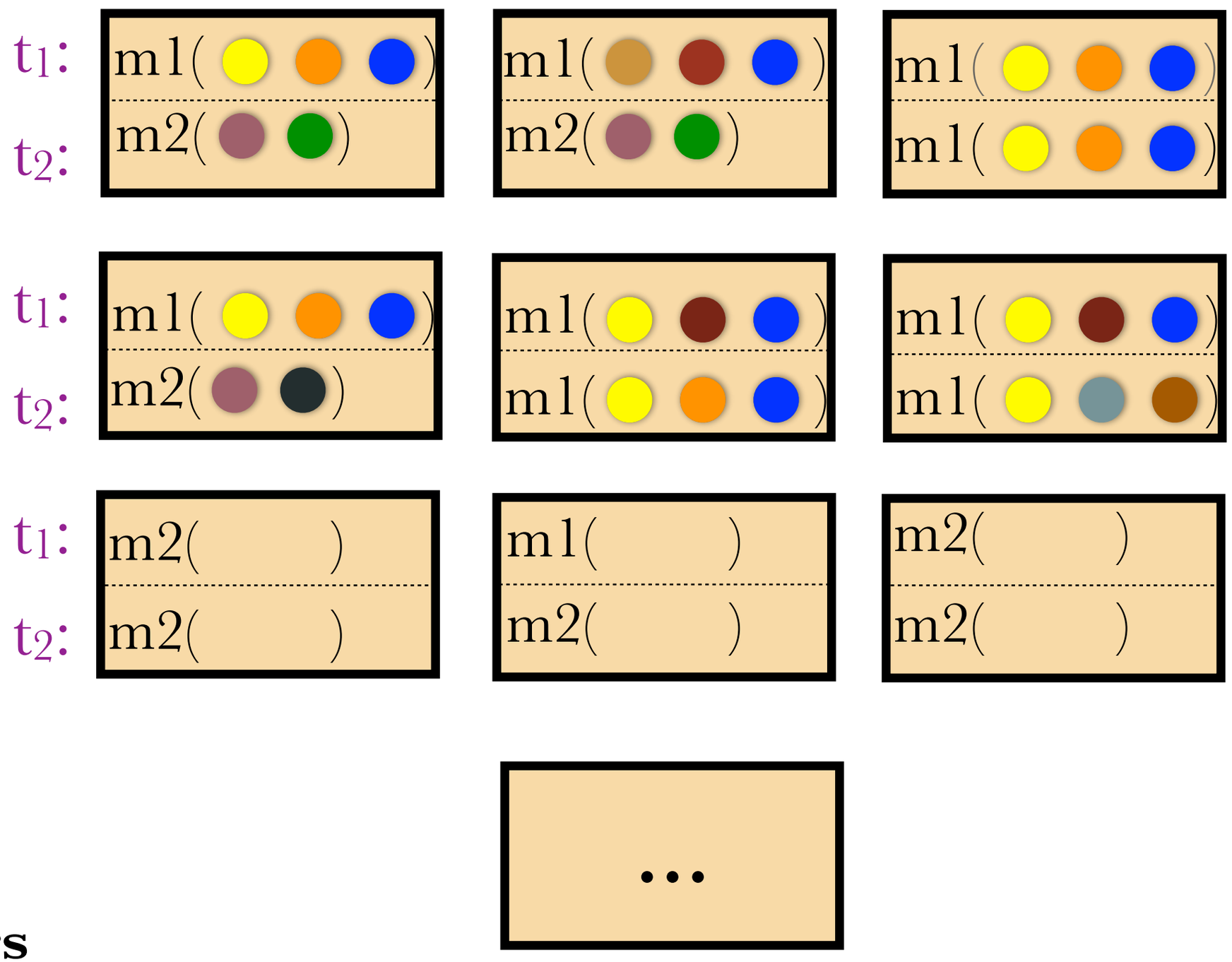


Randomized Test Generation

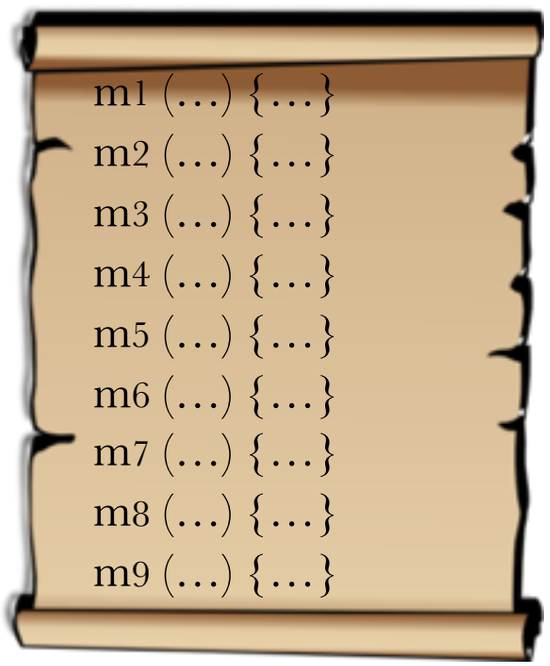


Library

Possible Parameters

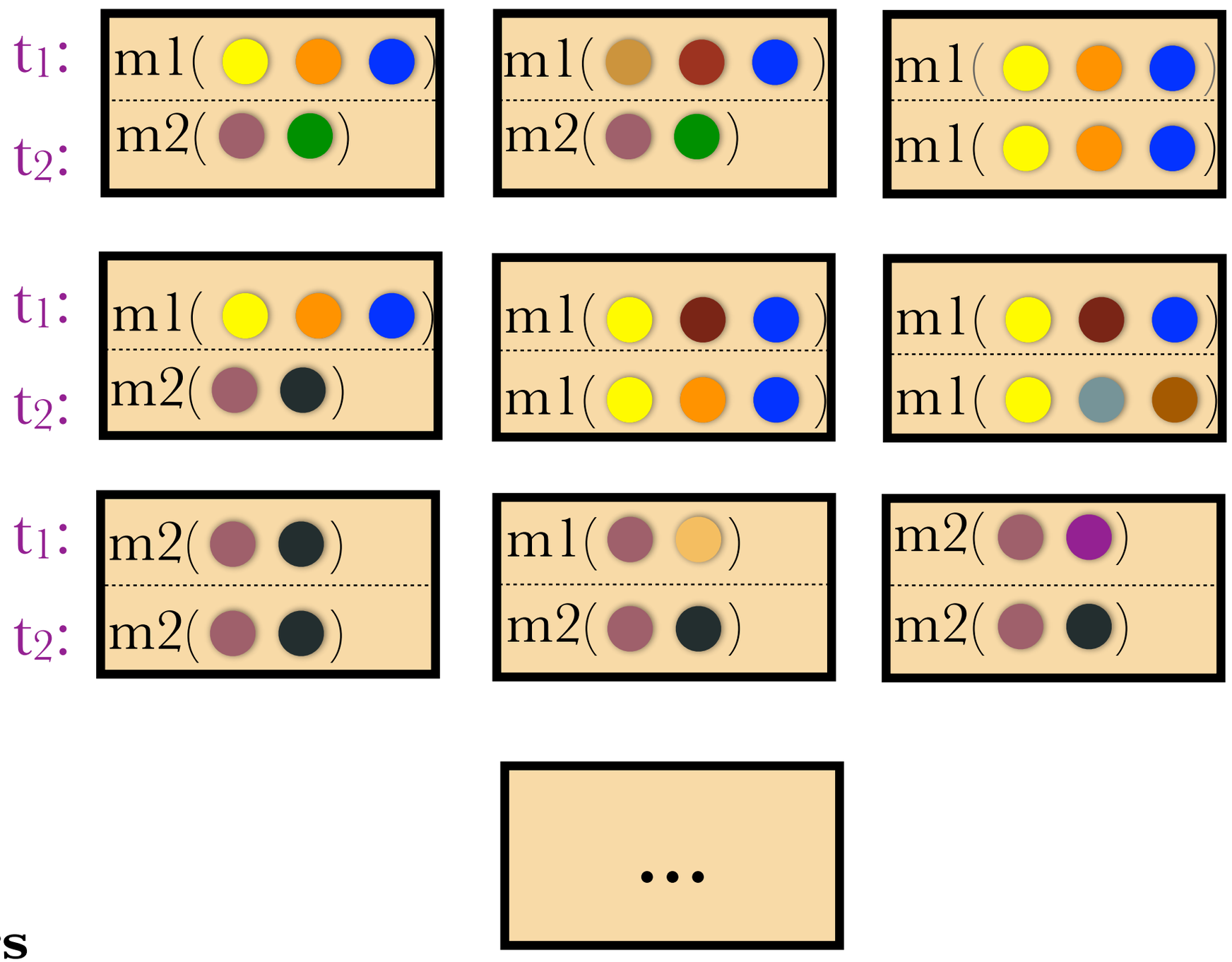


Randomized Test Generation

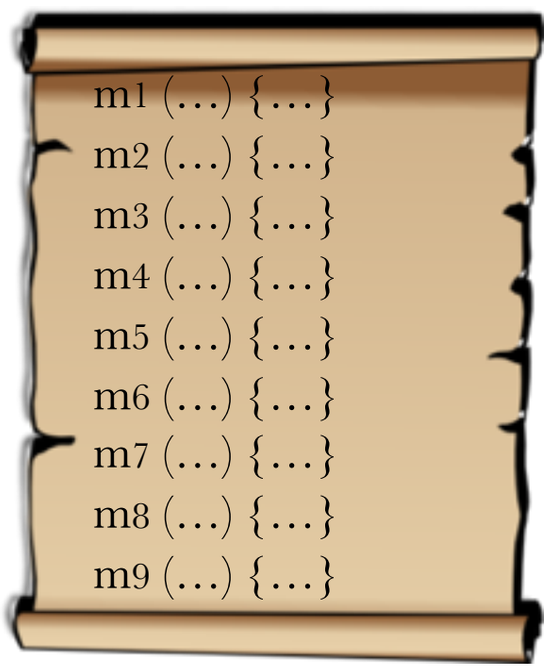


Library

Possible Parameters

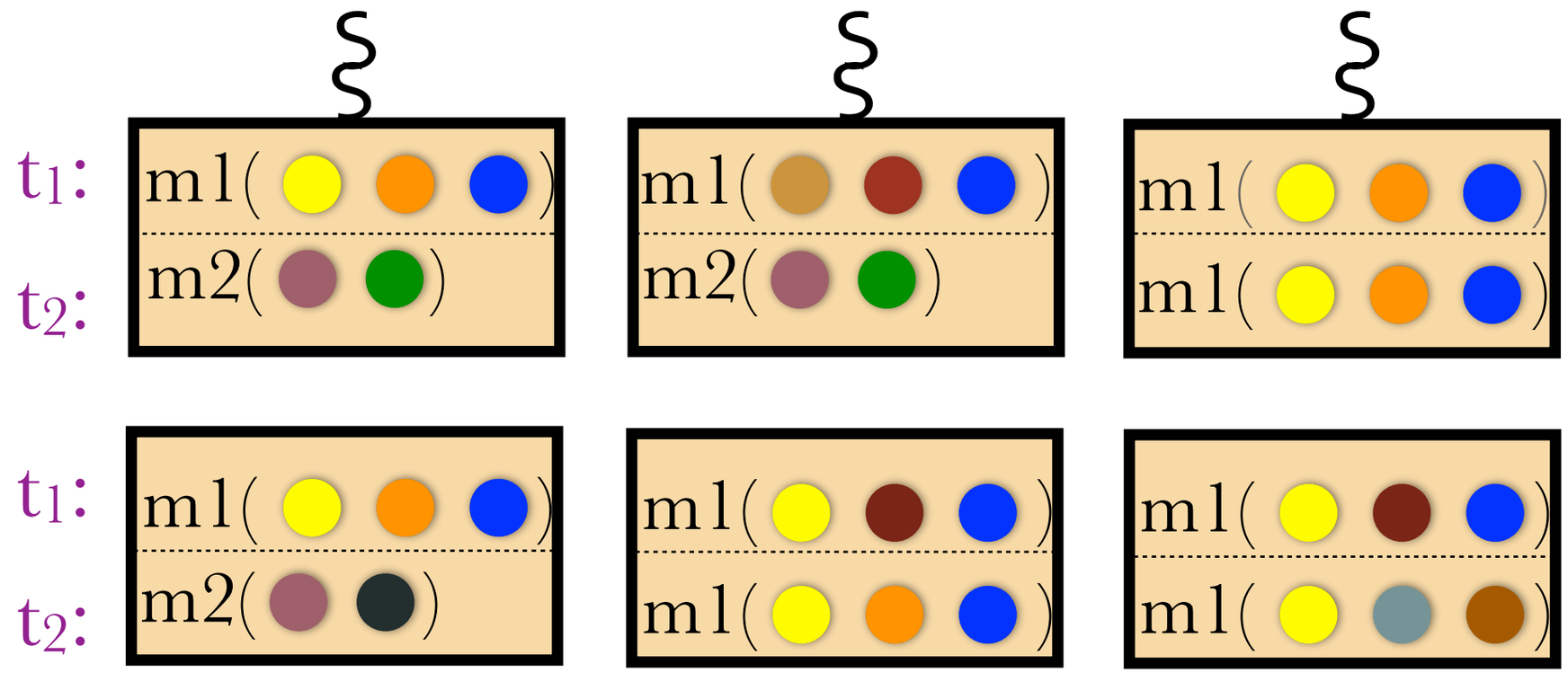


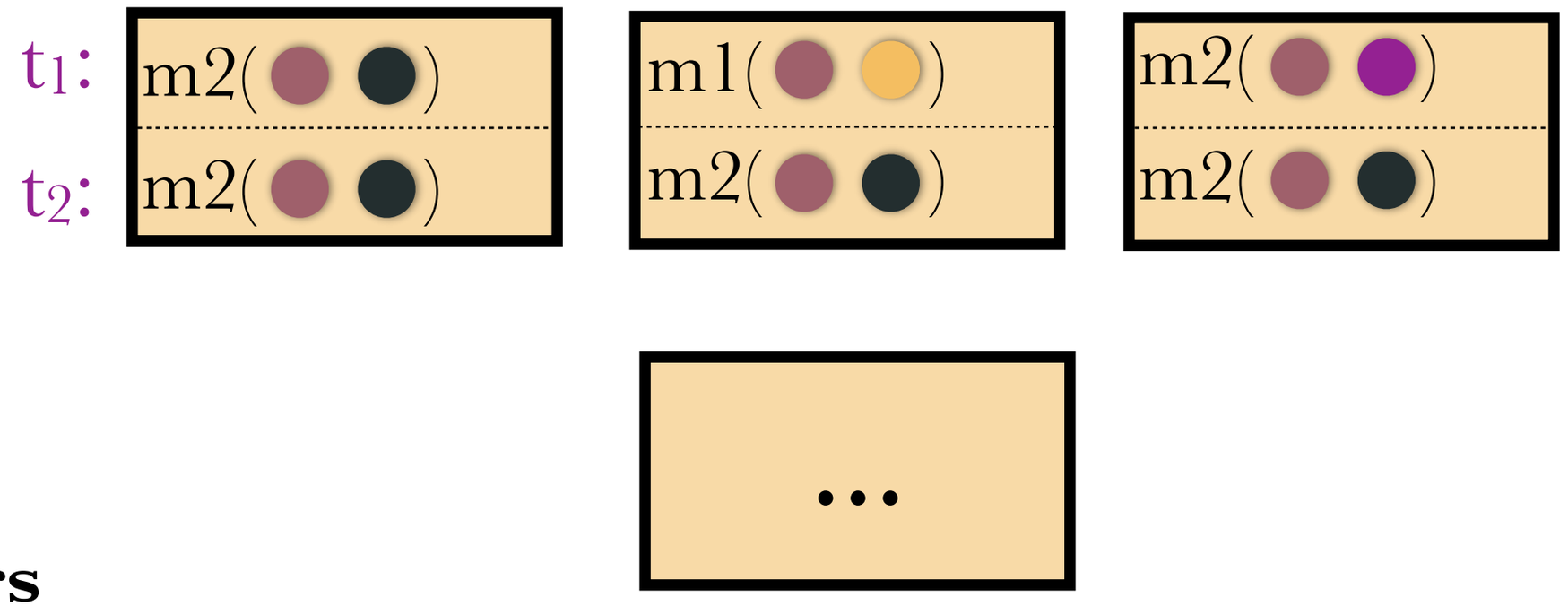
Randomized Test Generation



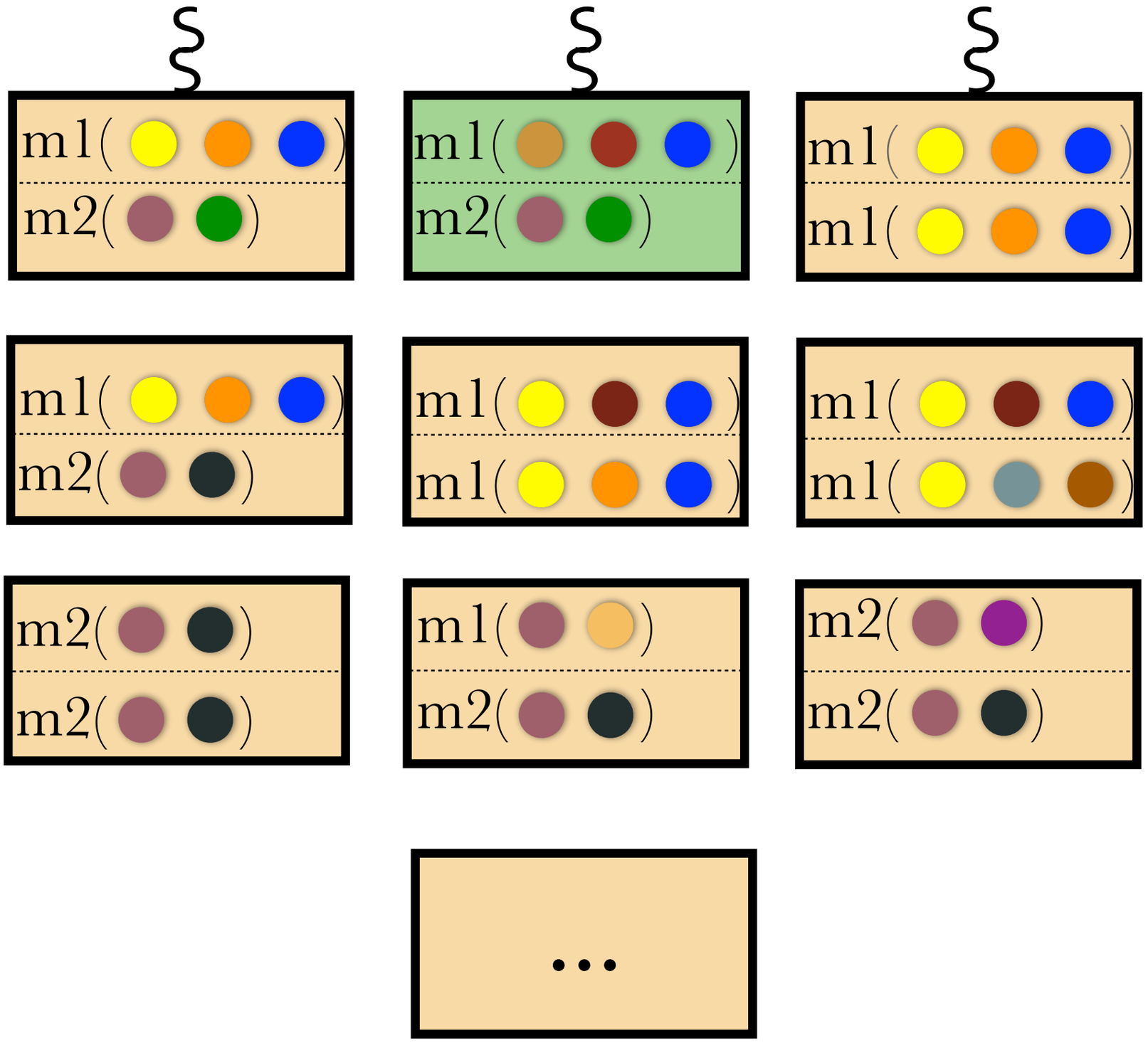
Library

Possible Parameters

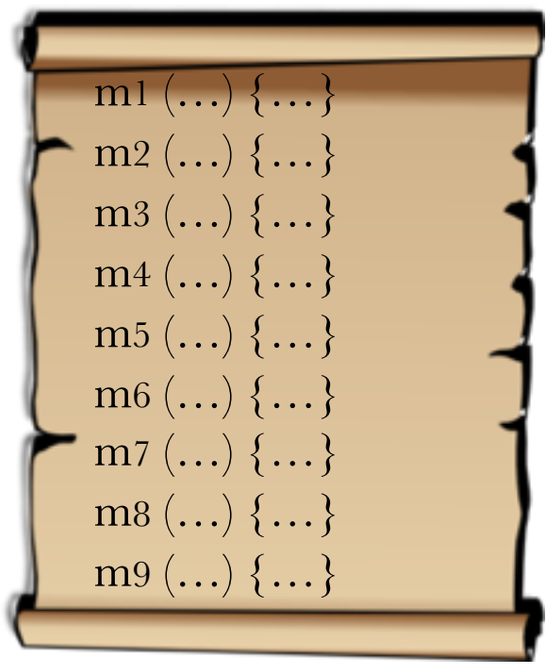




Randomized Test Generation



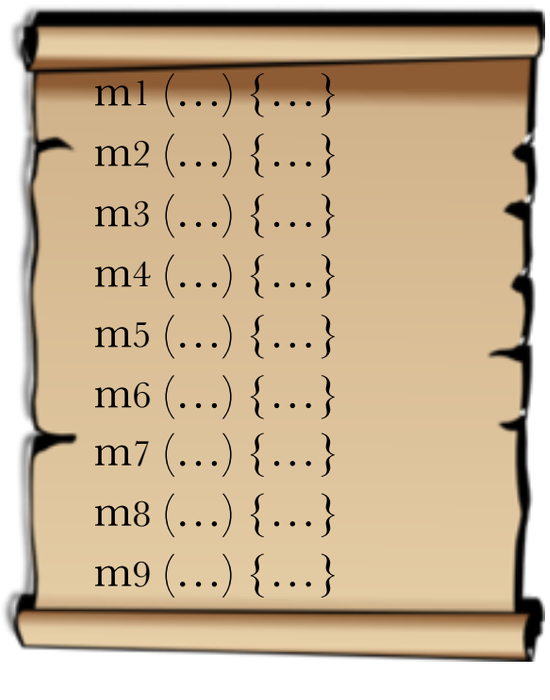
Targeted Test Synthesis



Library

Possible Parameters

Targeted Test Synthesis



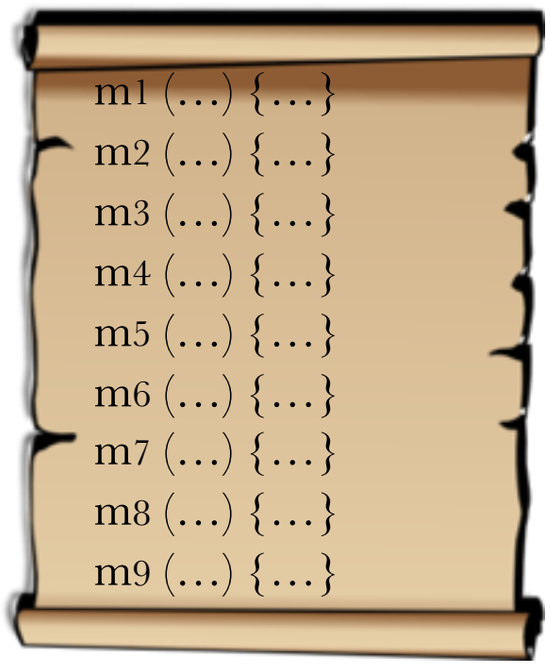
Library

Possible Parameters

t_1

t_2

Targeted Test Synthesis



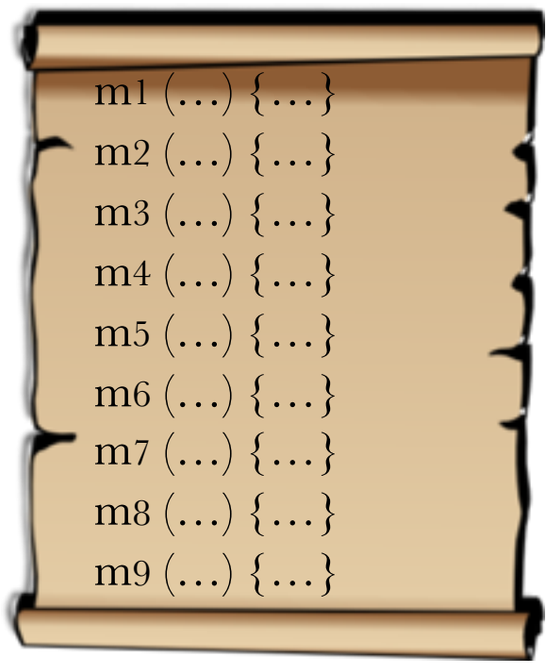
Library

Possible Parameters

t_1 t_2

m1( )

Targeted Test Synthesis



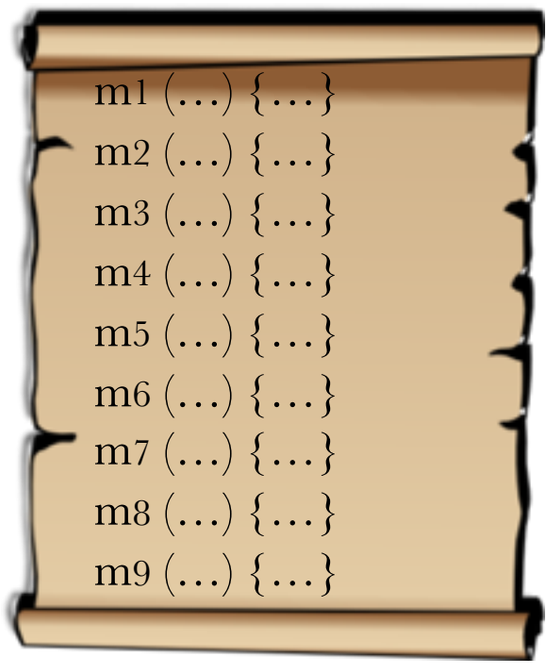
Library

Possible Parameters

t_1 t_2

m1( ) **m4**( )

Targeted Test Synthesis



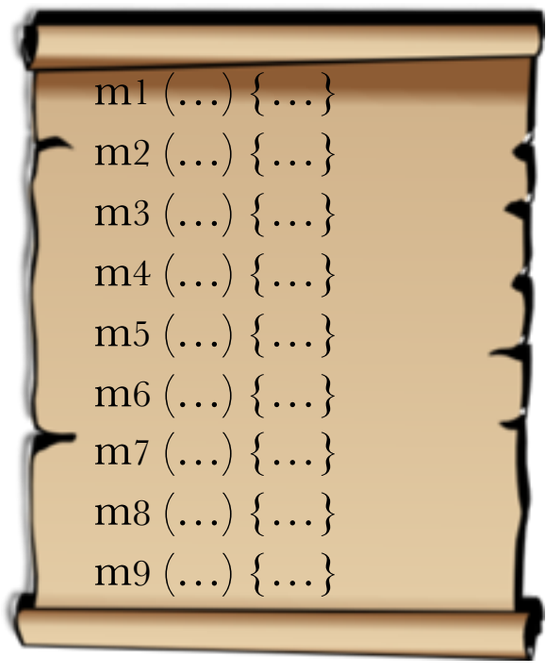
Library

Possible Parameters

t_1 t_2

m1(● ● ●) **m4**( )

Targeted Test Synthesis



Library

Possible Parameters

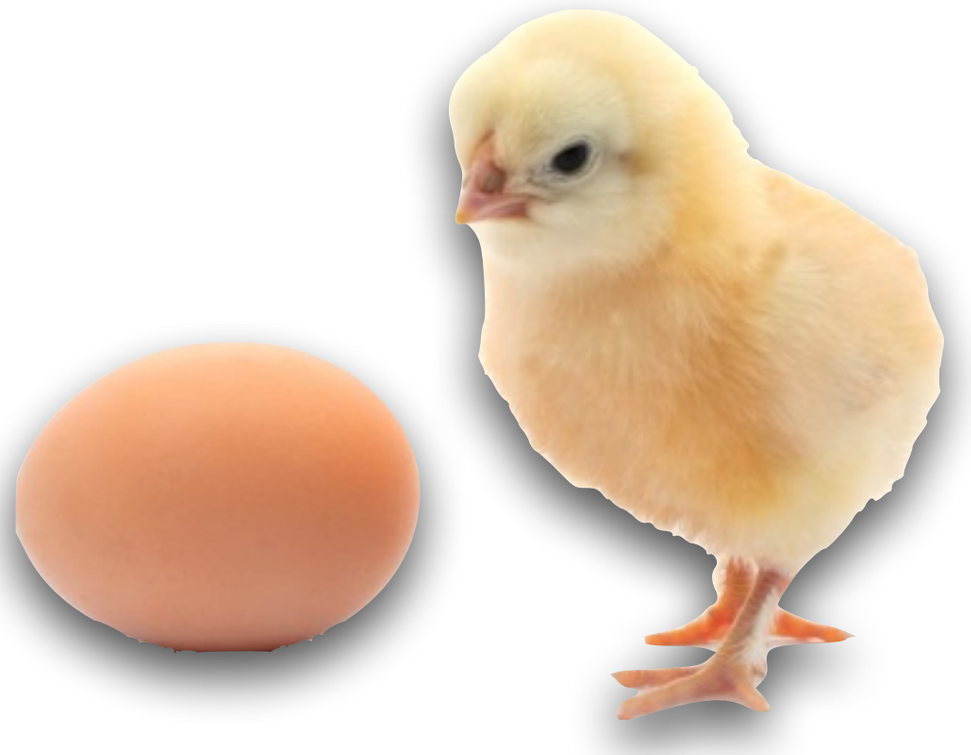
t_1

m1( )

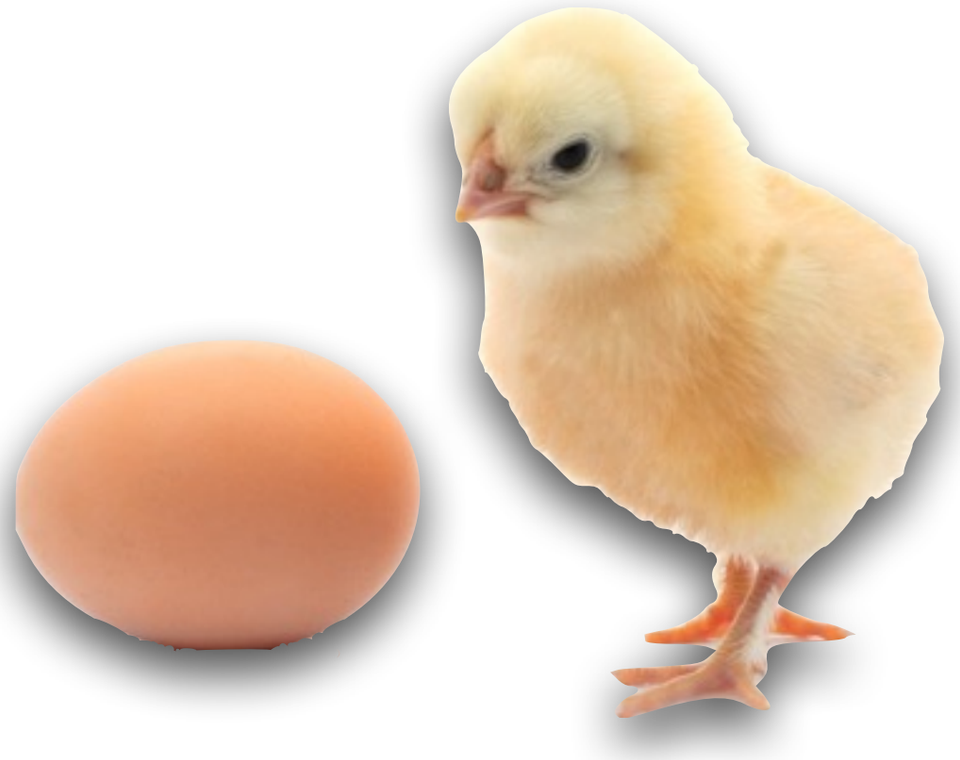
t_2

m4( )

Conundrum

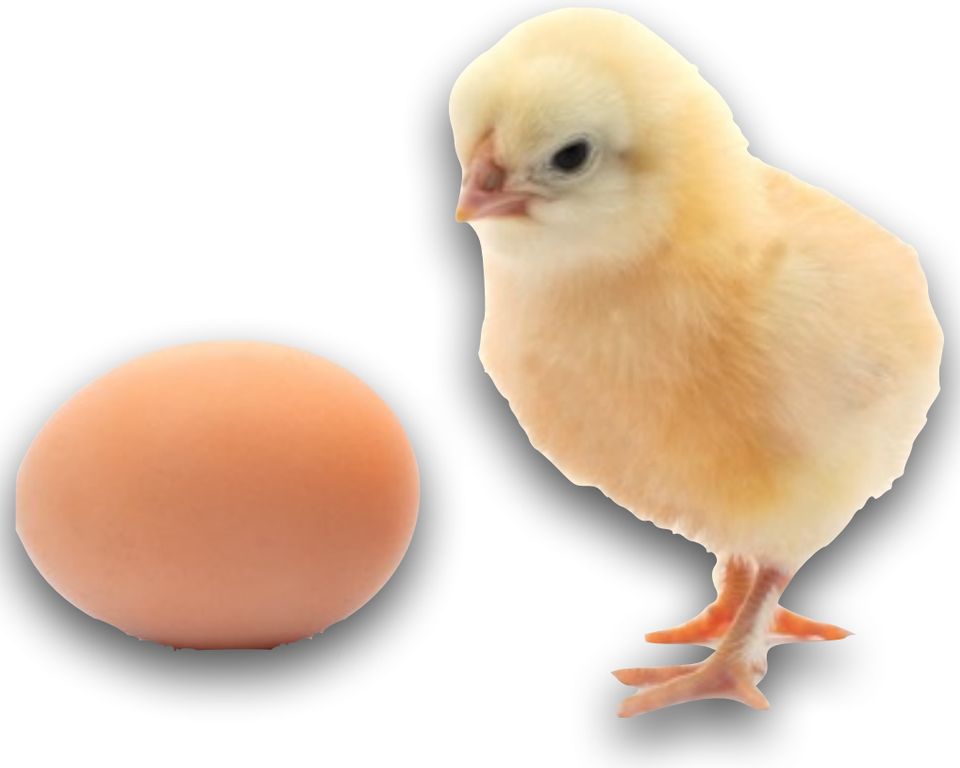


Conundrum



- 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

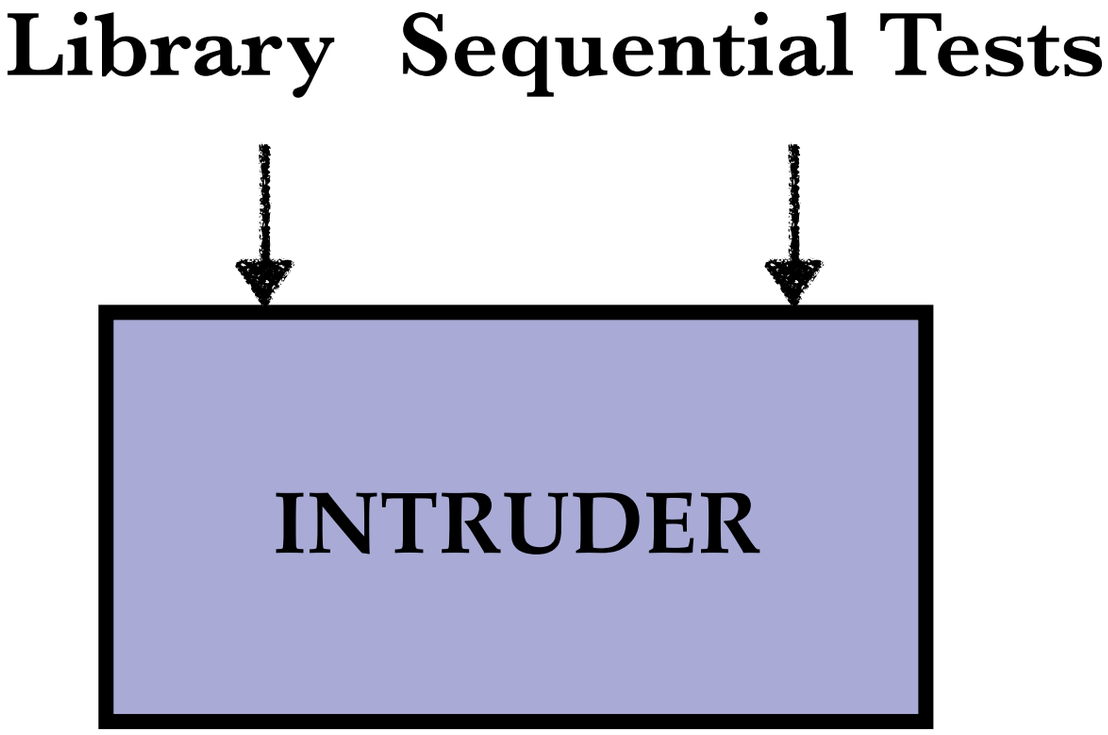
INTRUDER

Overview

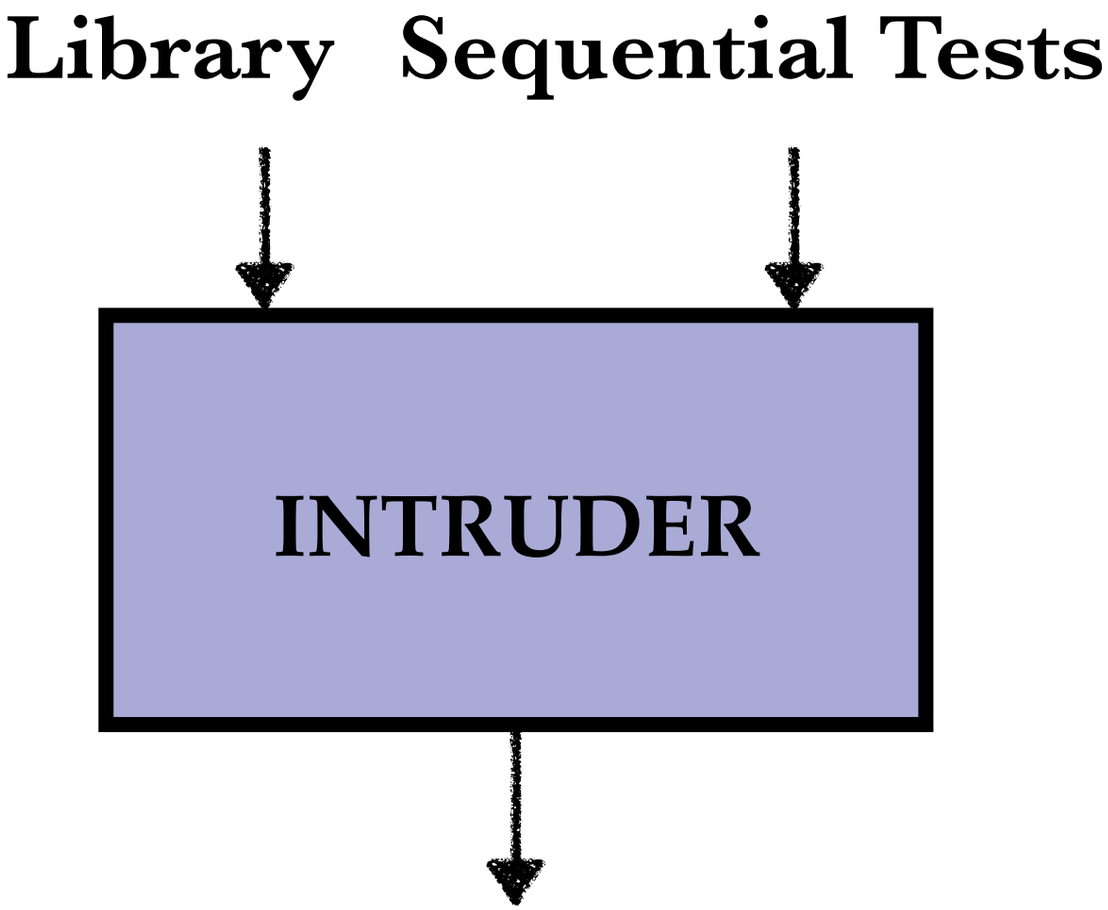
Library Sequential Tests

INTRUDER

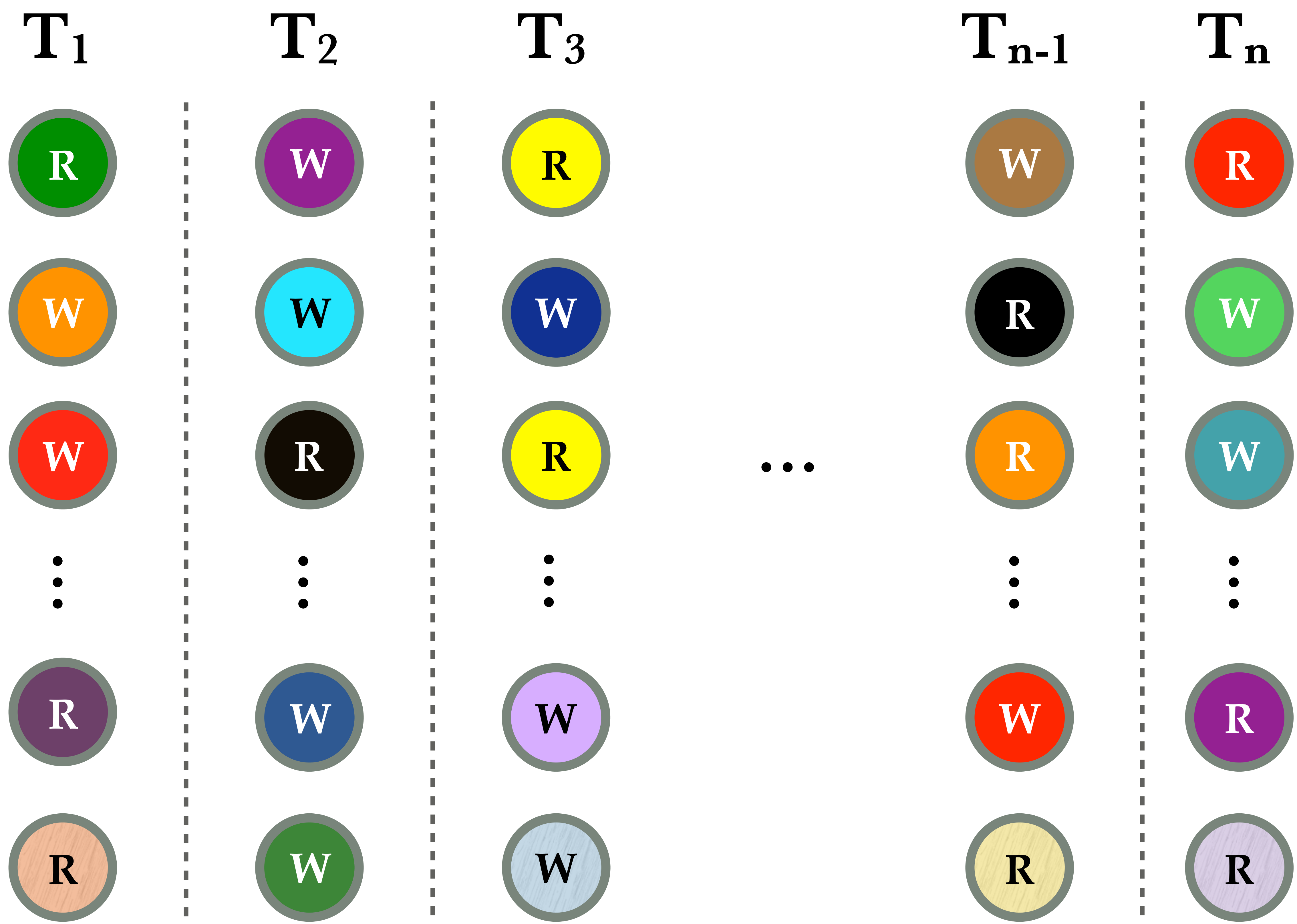
Overview



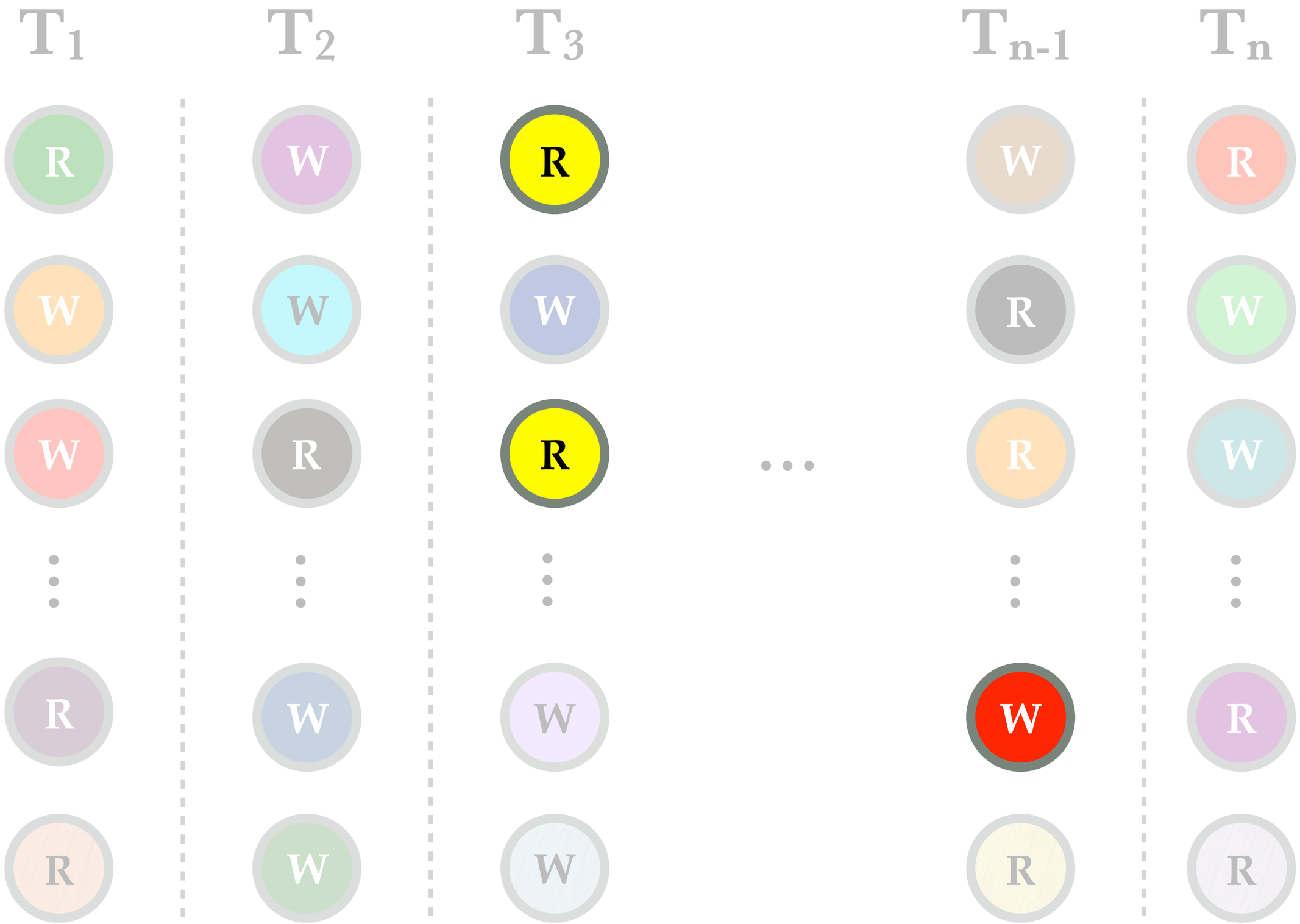
Overview



Challenge 1 : Identify accesses triplets

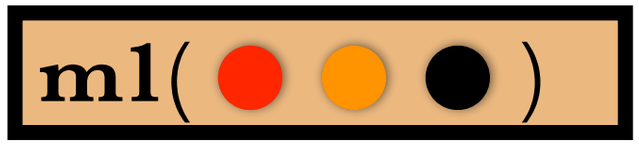


Challenge 1 : Identify accesses triplets

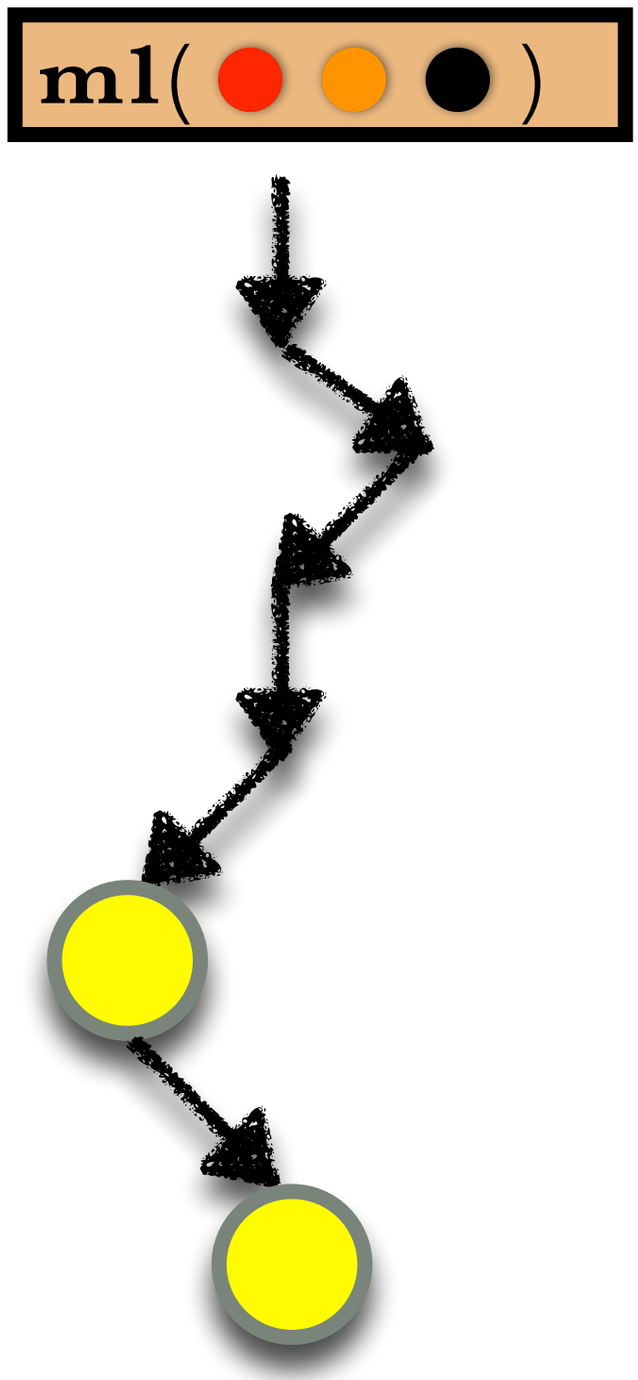


Challenge 2: Access Reachability

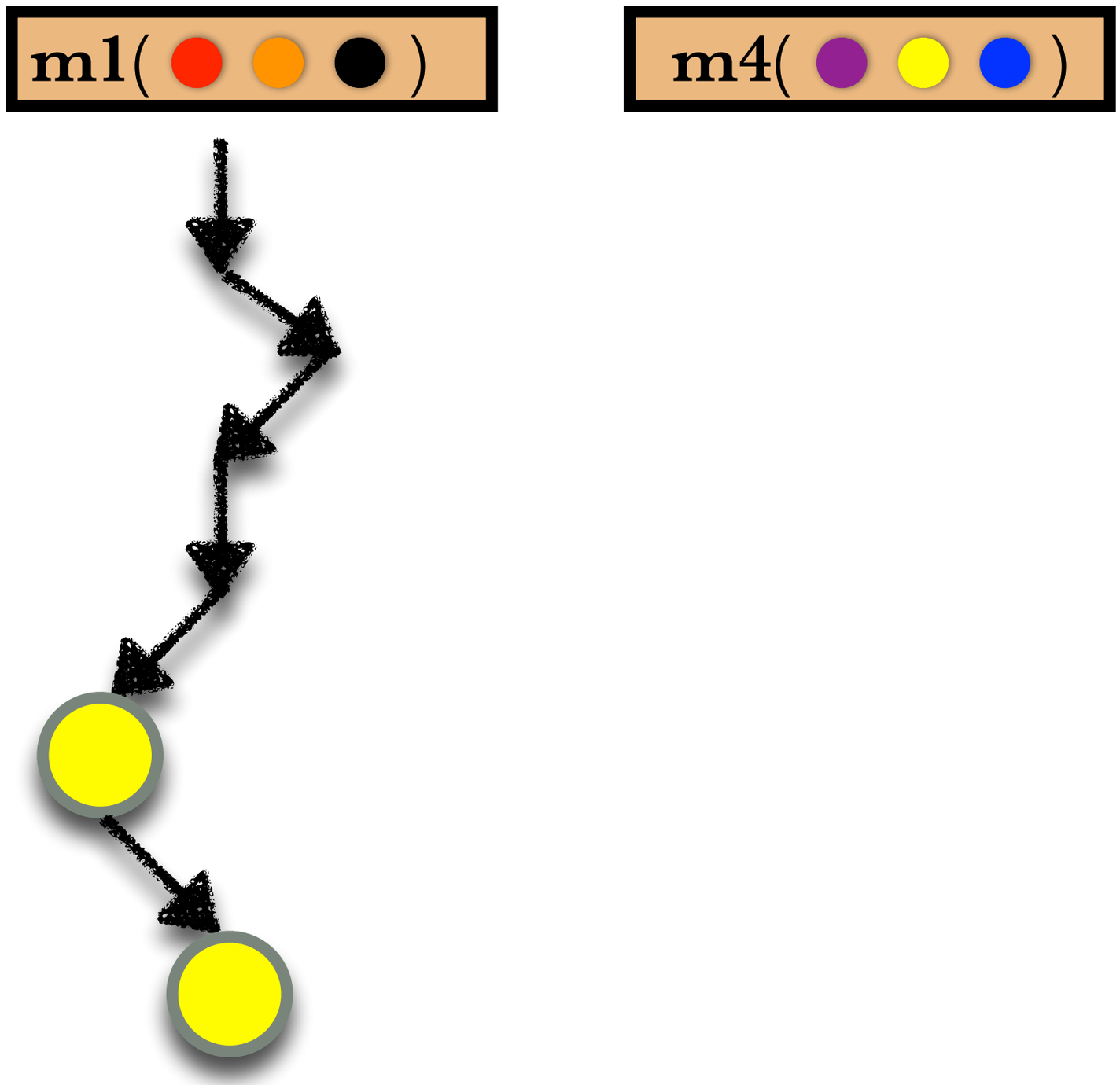
Challenge 2: Access Reachability



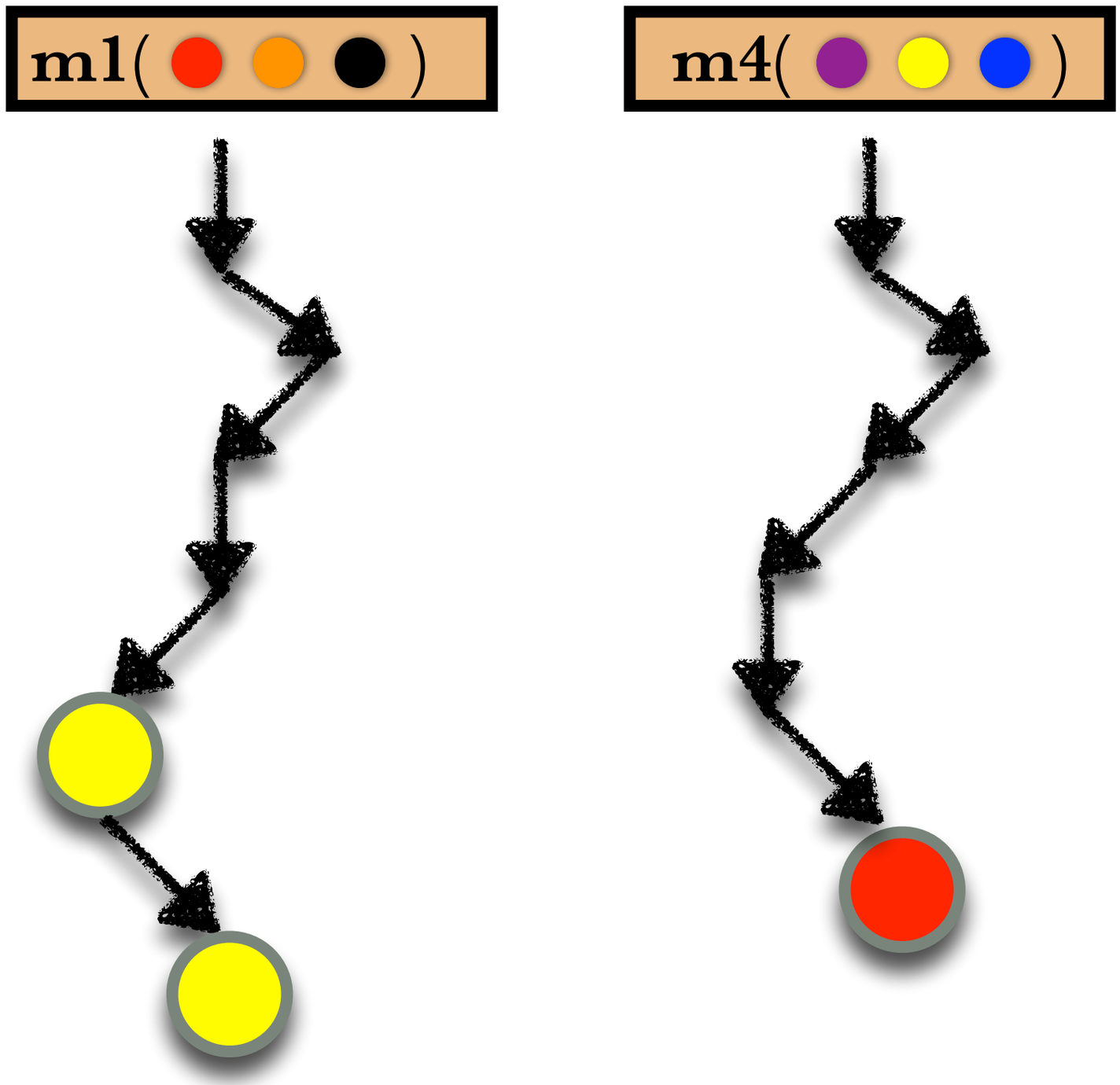
Challenge 2: Access Reachability



Challenge 2: Access Reachability



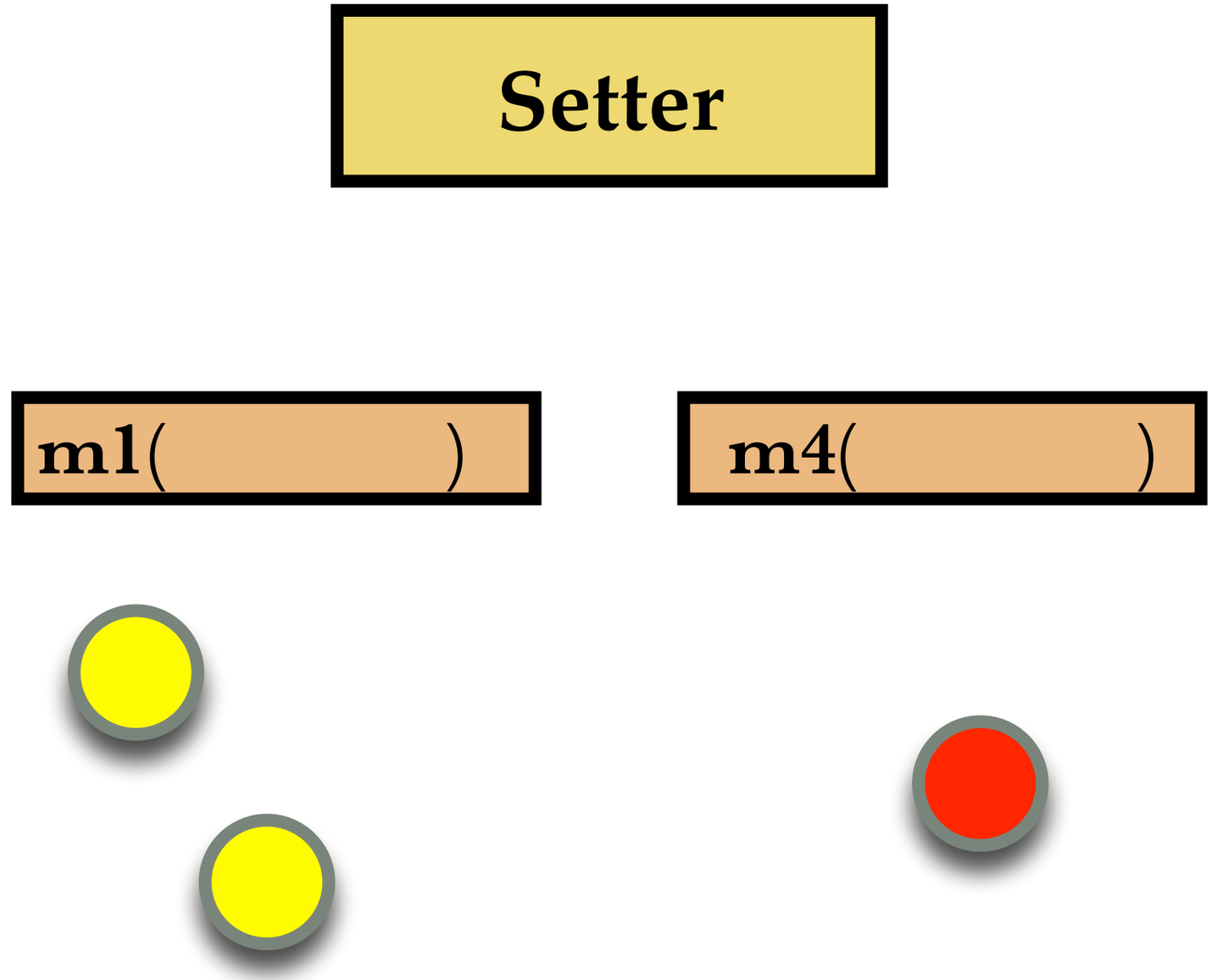
Challenge 2: Access Reachability



Challenge 3 : Setting Objects

Challenge 3 : Setting Objects

Challenge 3 : Setting Objects

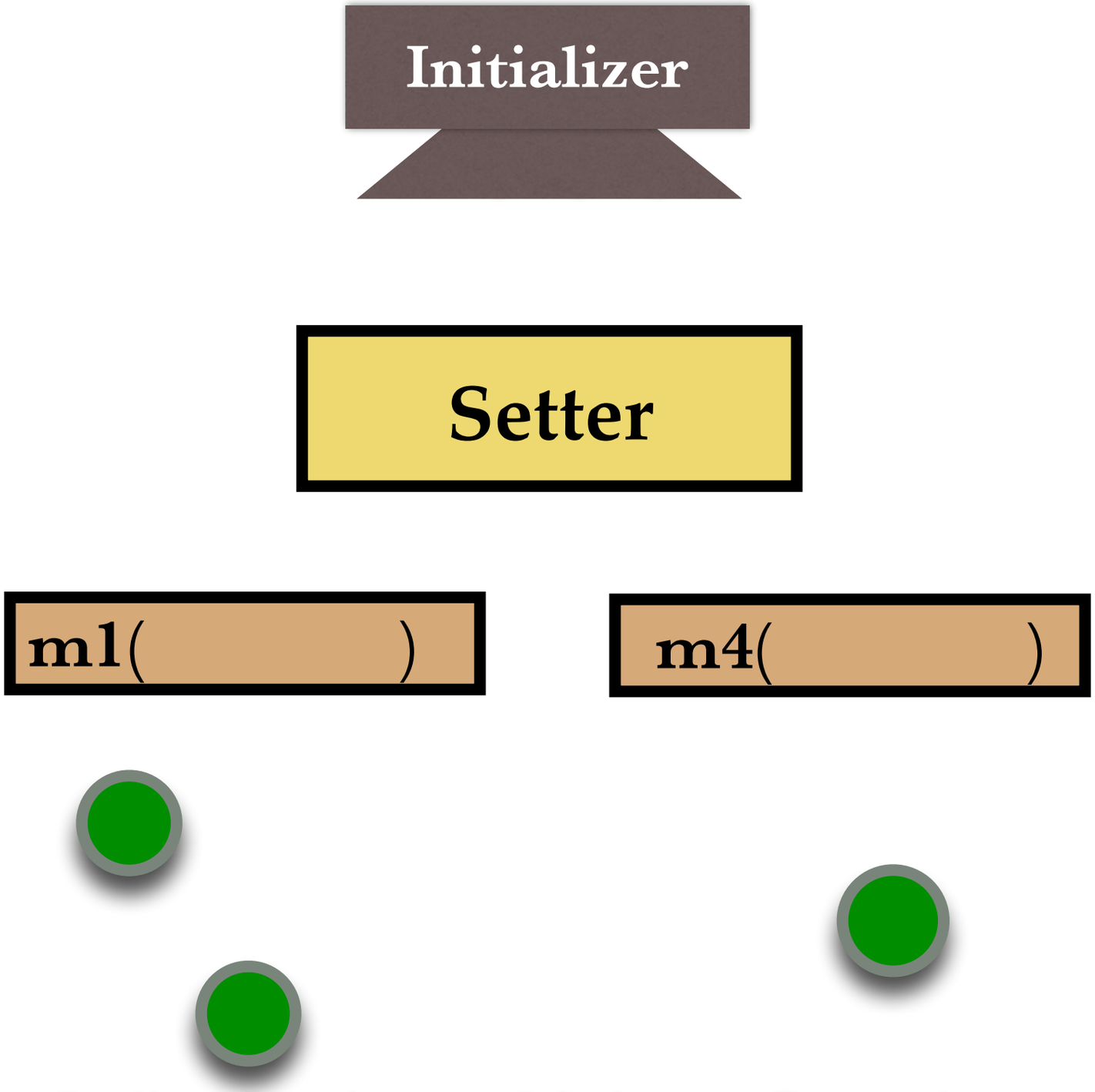


Challenge 3 : Setting Objects

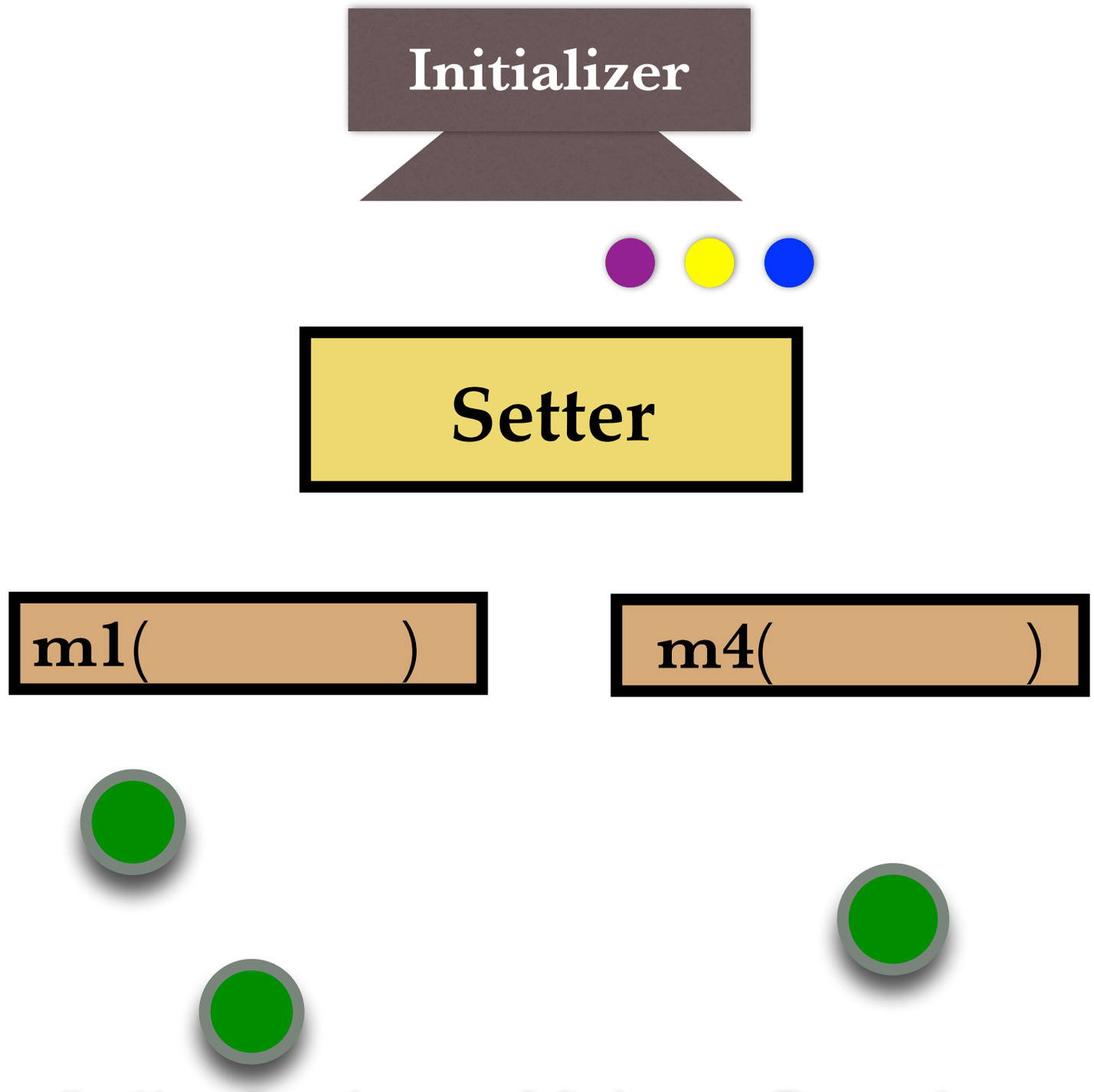
Challenge 3 : Setting Objects

Challenge 3 : Setting Objects

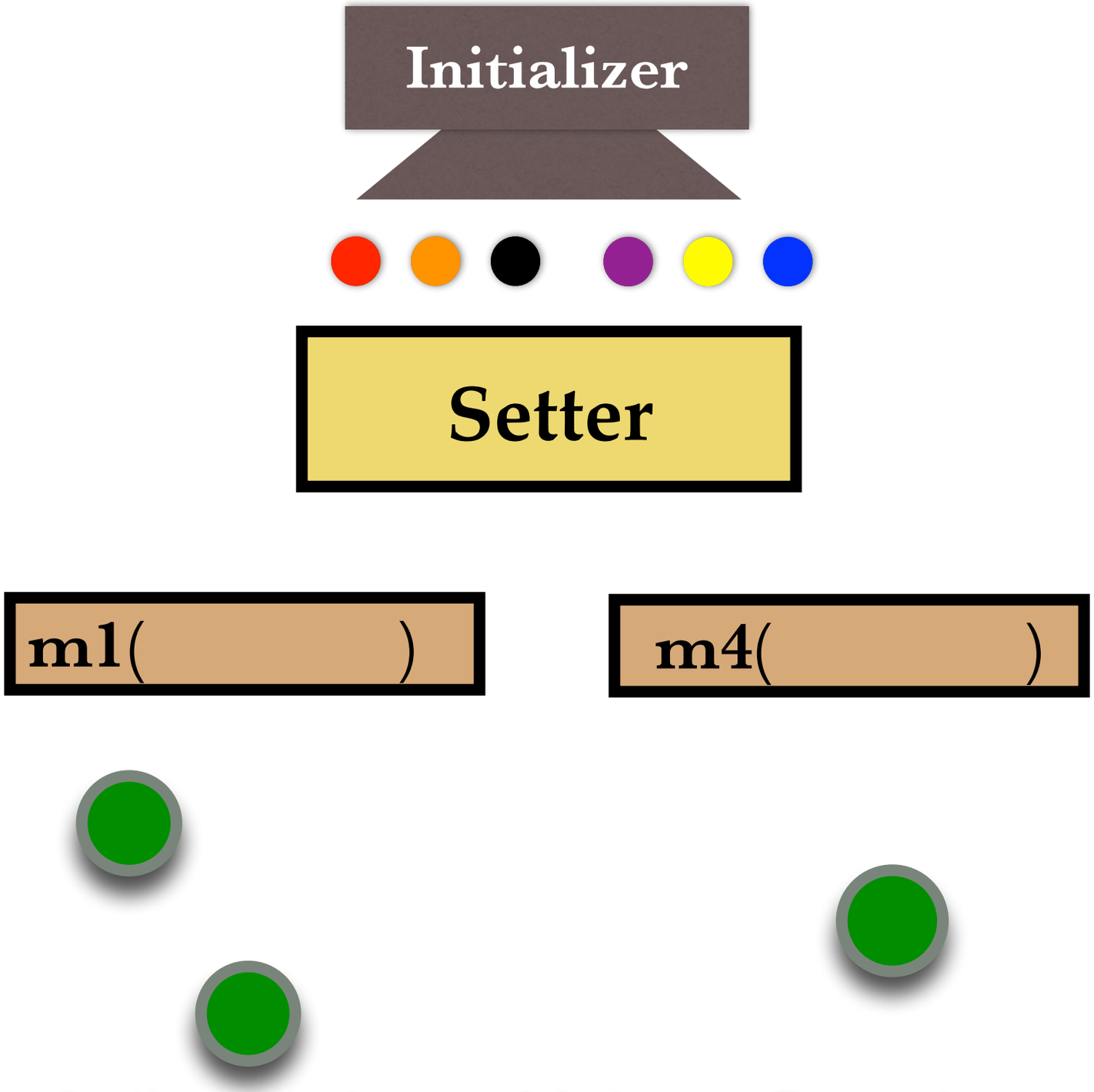
Challenge 4 : Generating Objects



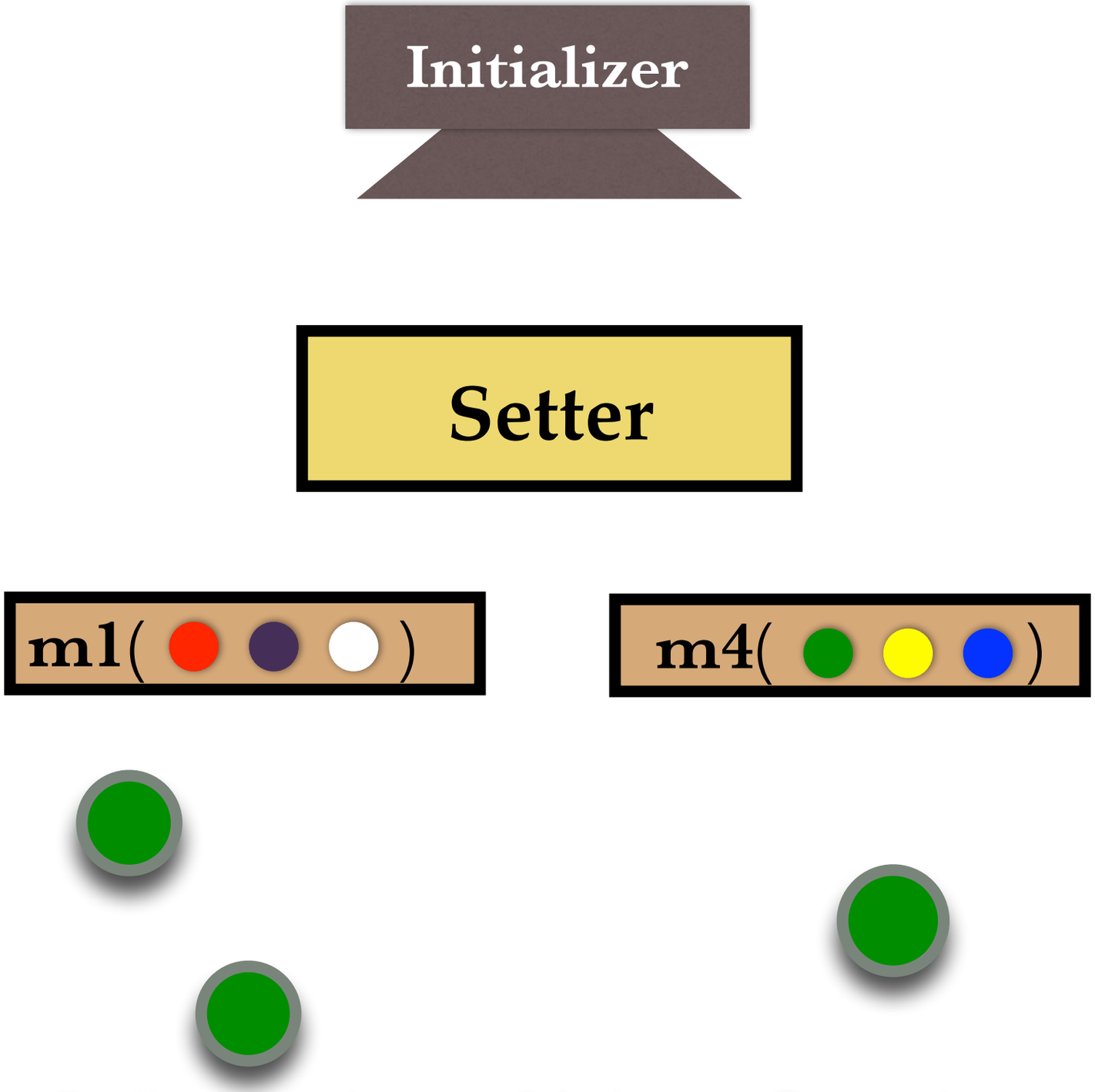
Challenge 4 : Generating Objects



Challenge 4 : Generating Objects



Challenge 4 : Generating Objects



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

a_1.evaluate (…)

a_2.reset (…)

a_3.set (f_1)

Identify access triplets

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

Input Sequential Test

a_1.evaluate (…)
a_2.reset (…)
a_3.set (f_1)

a_1 $a_1.f$

Identify access triplets

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

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

Input Sequential Test

a_1.evaluate (…)
a_2.reset (…)
a_3.set (f_1)

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
}
```

Input Sequential Test

a_1.evaluate (…)
a_2.reset (…)
a_3.set (f_1)


a_1


a_1.f


a_2


a_3

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

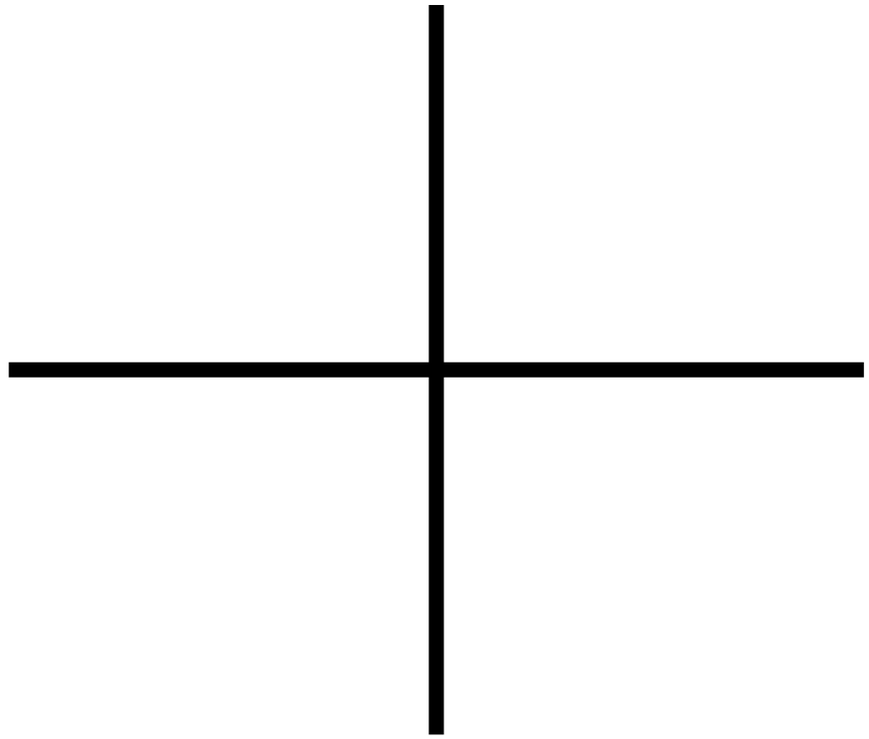
```
evaluate ( … ) {
    temp = this.f
    flag = (temp.g != null)
    if(flag)  assert(temp.g != null)
}

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
}
```

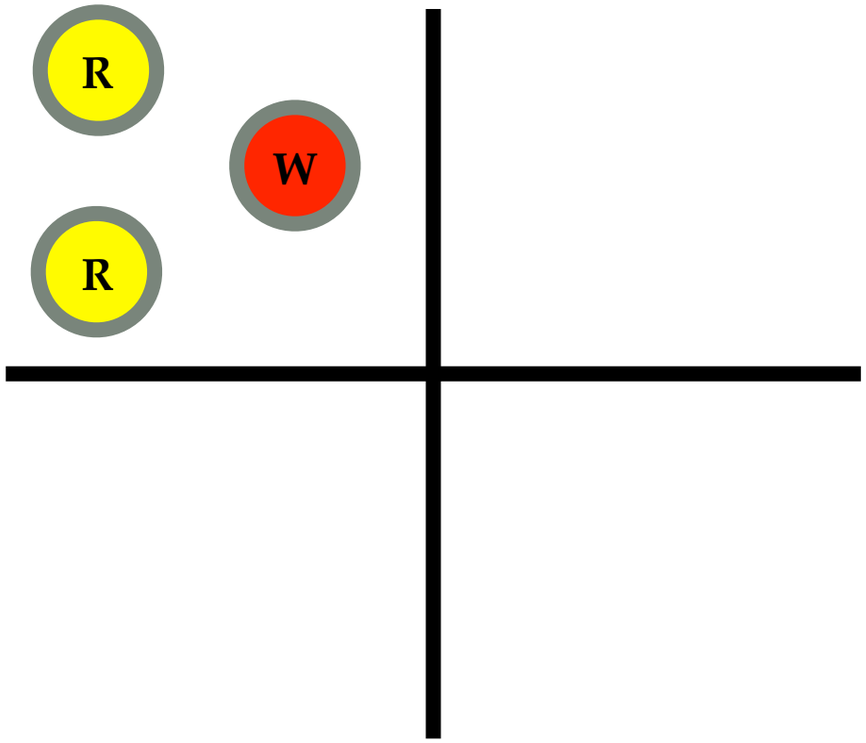
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

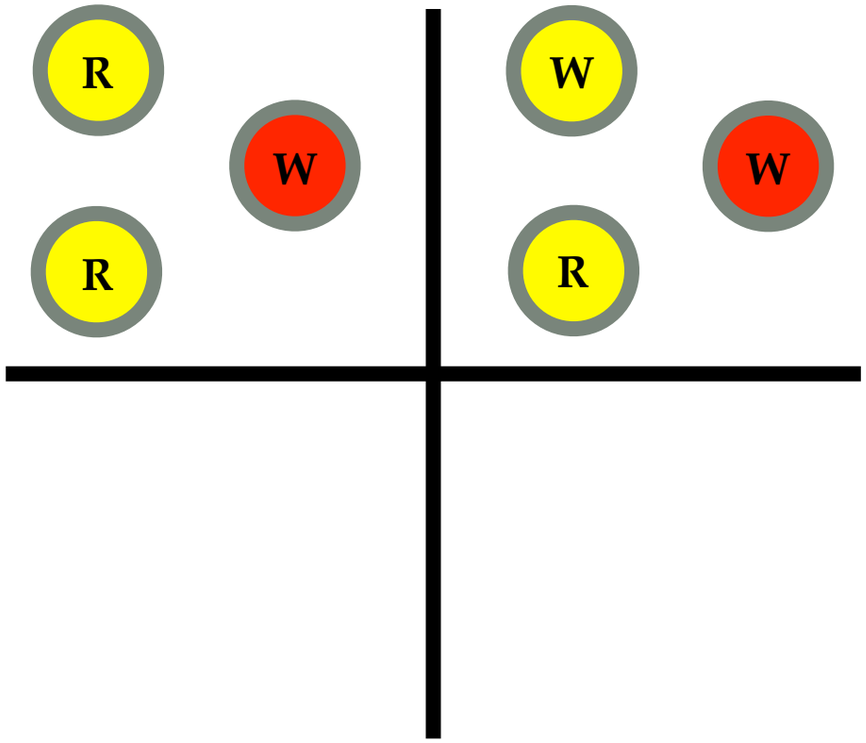
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

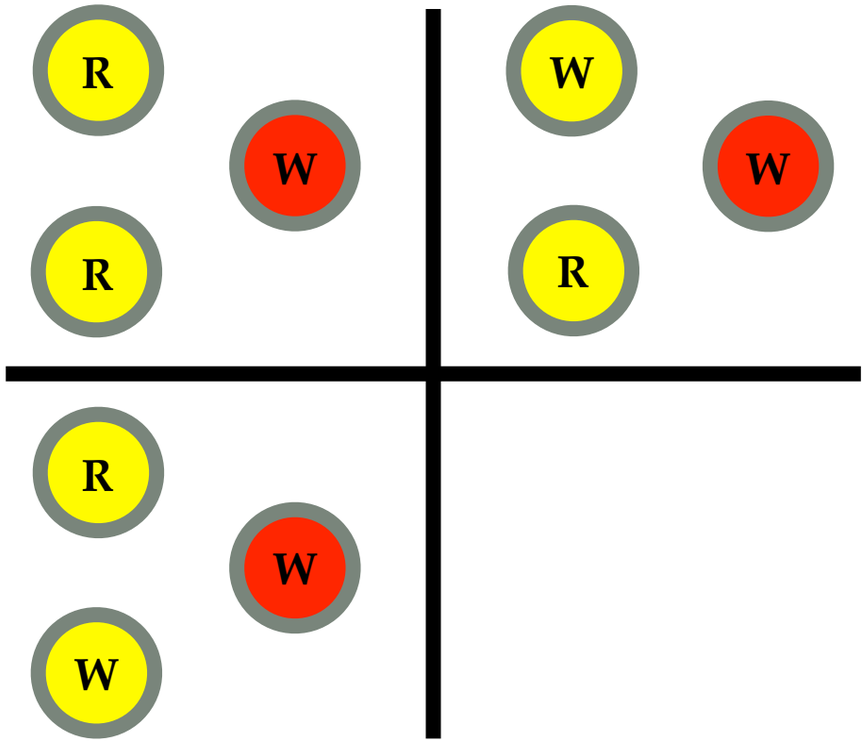
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

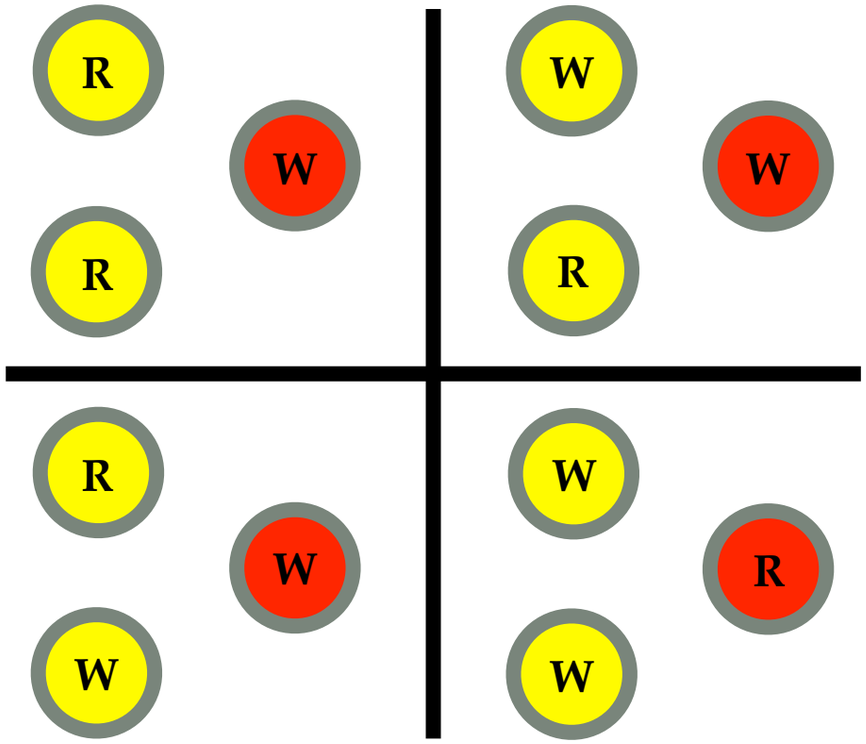
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

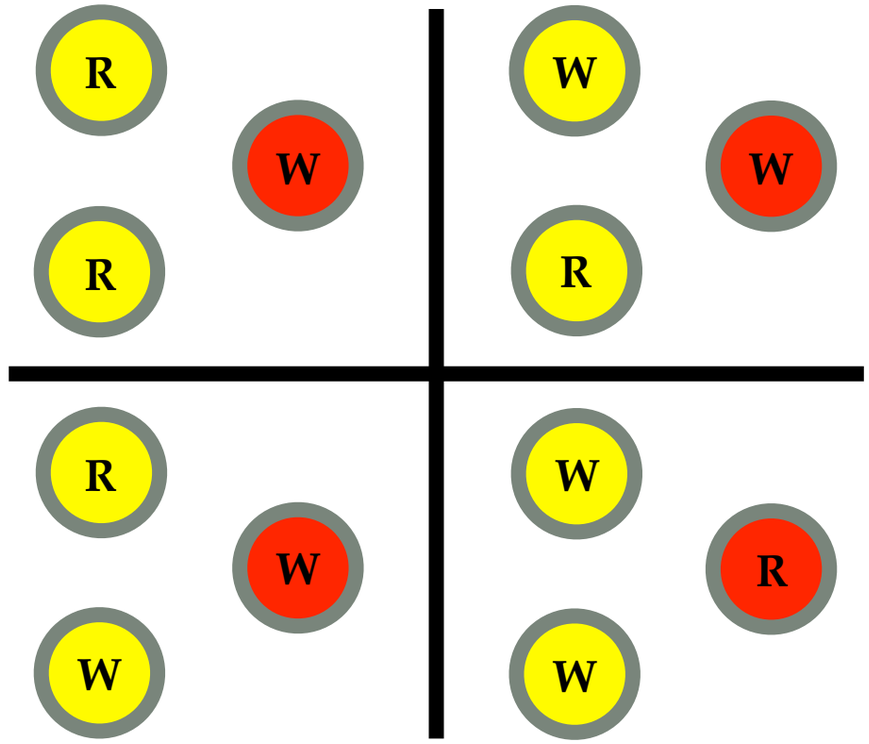
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

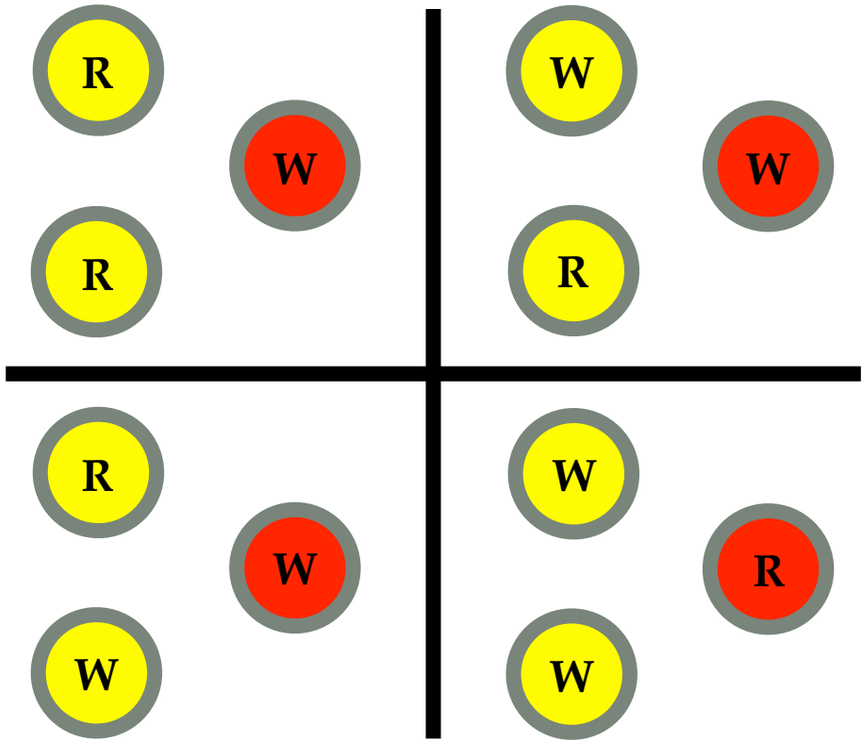
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

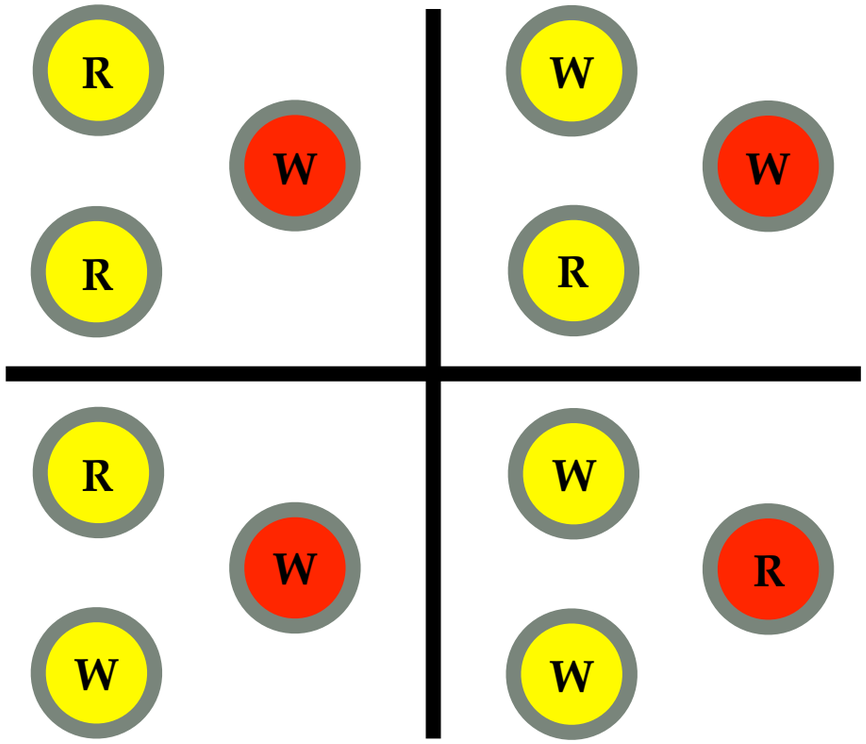
1. Access same field
2. P & C are consecutive accesses
3. P - C - R accesses are :



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
}
```

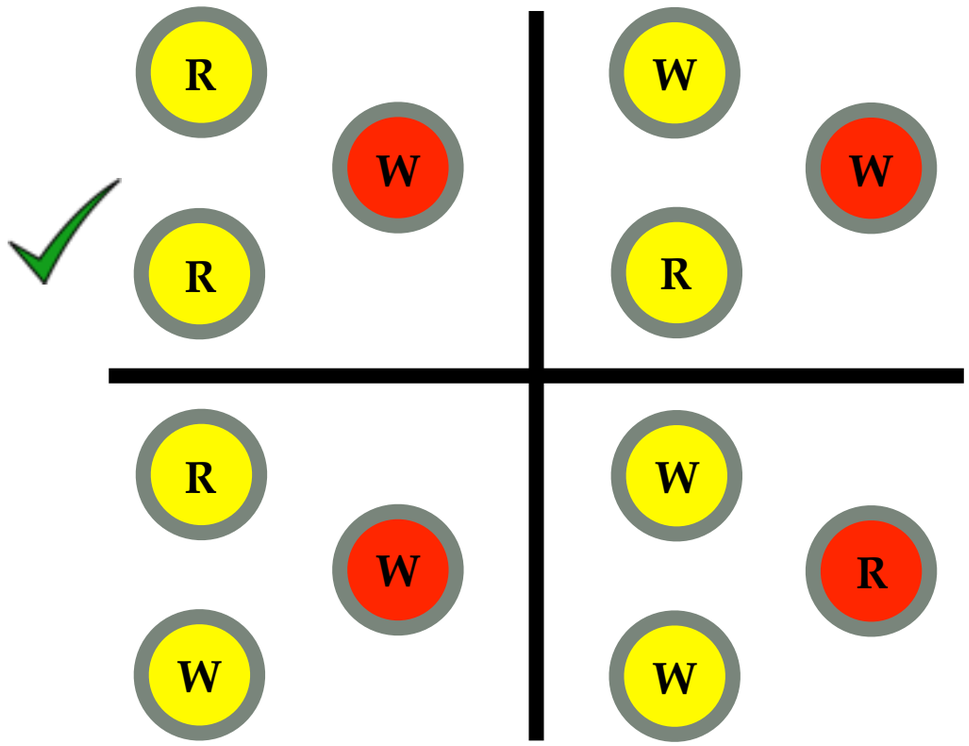
1. Access same field ✓
2. P & C are consecutive accesses ✓
3. P - C - R accesses are :



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
}
```

a_1 $a_1.f$

a_2

Identify access triplets

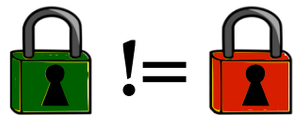
```
evaluate ( … ) {
    temp = this.f
P   flag = (temp.g != null)
C   if(flag)  assert(temp.g != null)
}

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

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

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 != 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 != a_2 !=
2. a_1.f.g = a_2.f.g

a_1

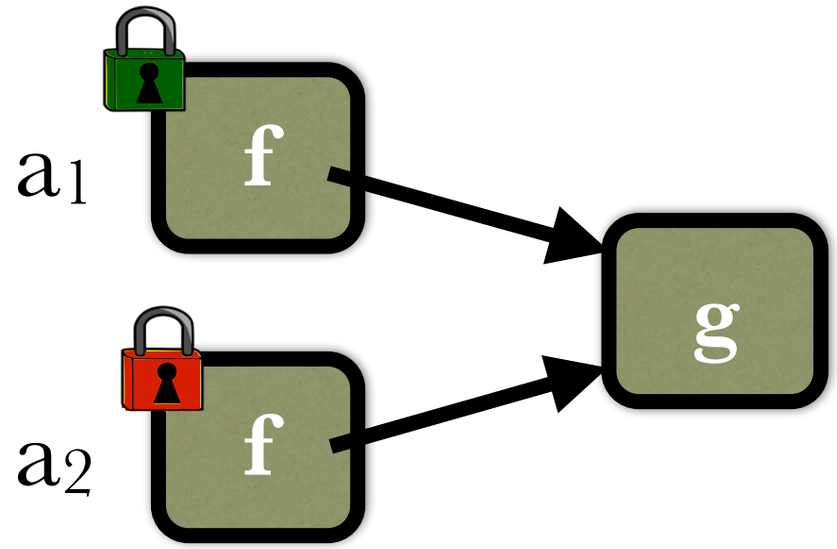
a_2

a_1.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
}
```

1. a_1 != a_2 !=
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 != a_2 =
2. $a_1.f.g = a_2.f.g$

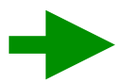
a_1

a_2

$a_1.f$

Identify the APIs

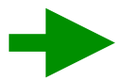
```
evaluate ( … ) {
    temp = this.f
P   flag = (temp.g != null)
C   if(flag)  assert(temp.g != null)
}

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

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



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

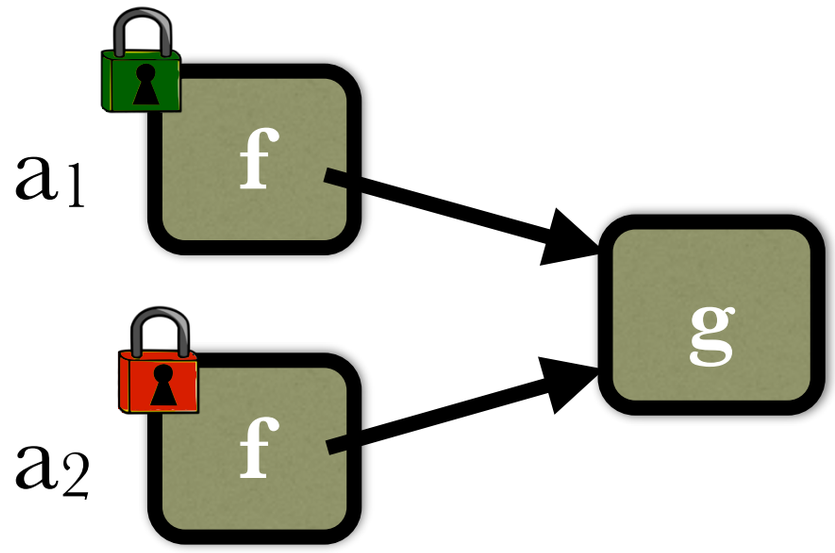
reset

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

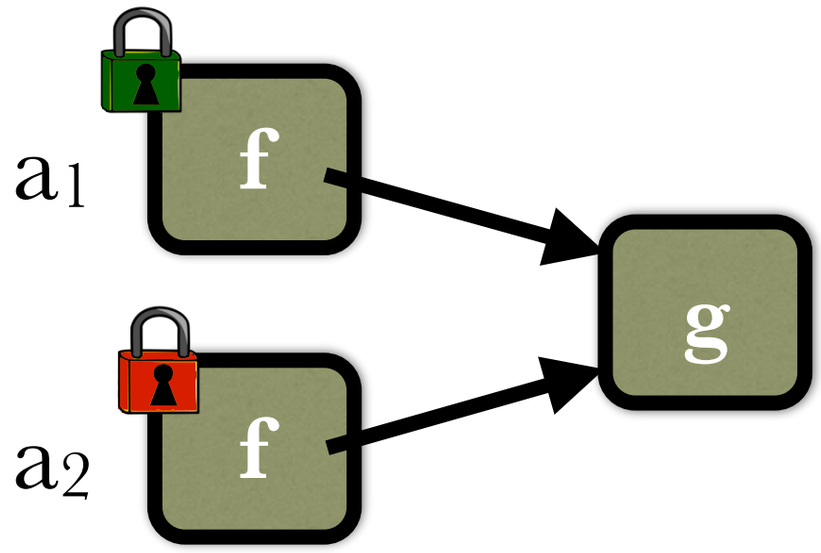
```
evaluate ( … ) {
    temp = this.f
    flag = (temp.g != null)
    if(flag)  assert(temp.g != null)
}

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
}
```

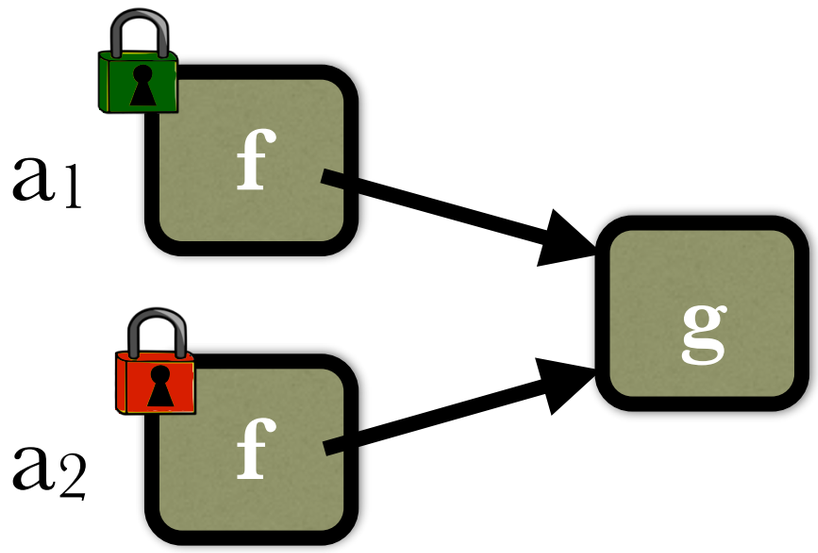


Desired object state

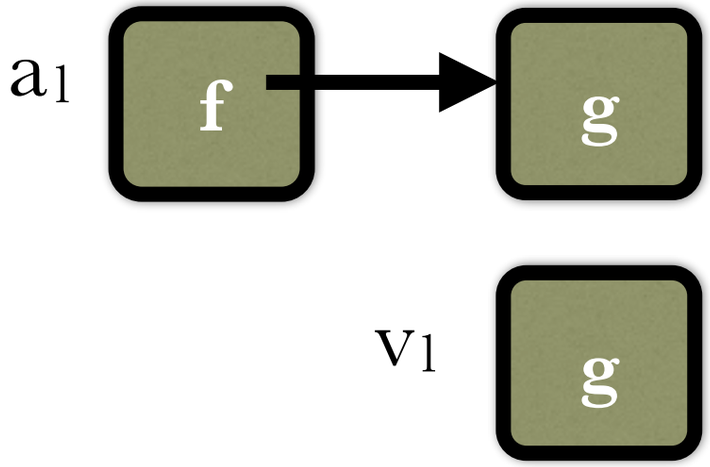
Set the context

```
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
}
```



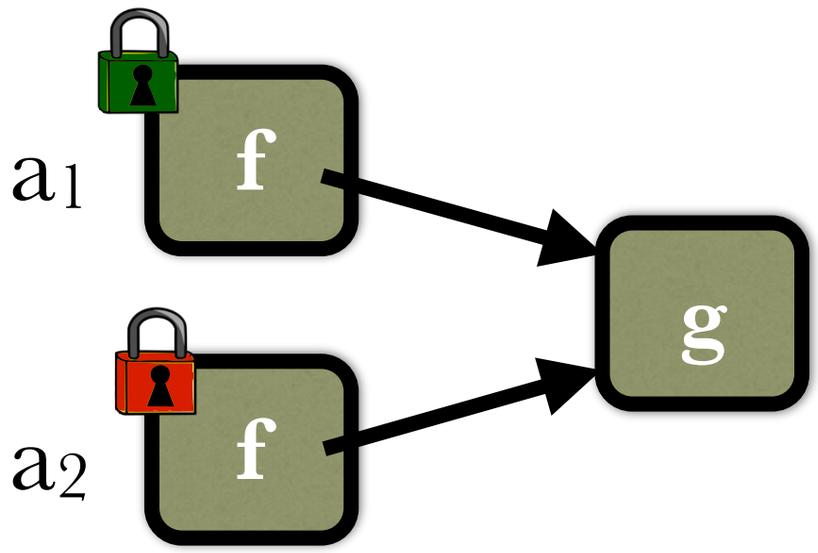
Desired object state



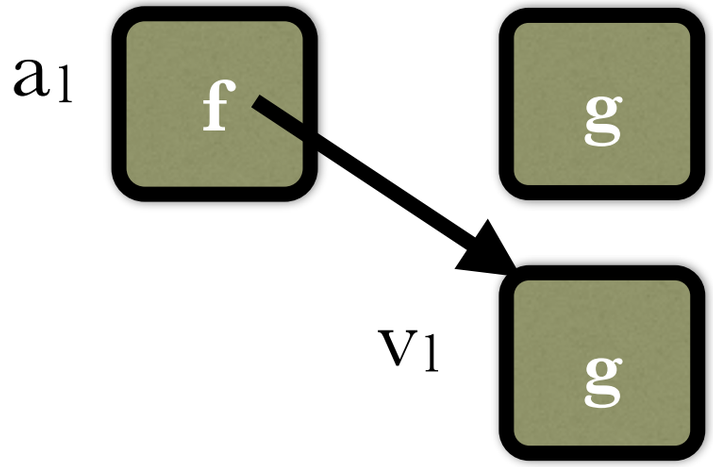
Set the context

```
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
}
```



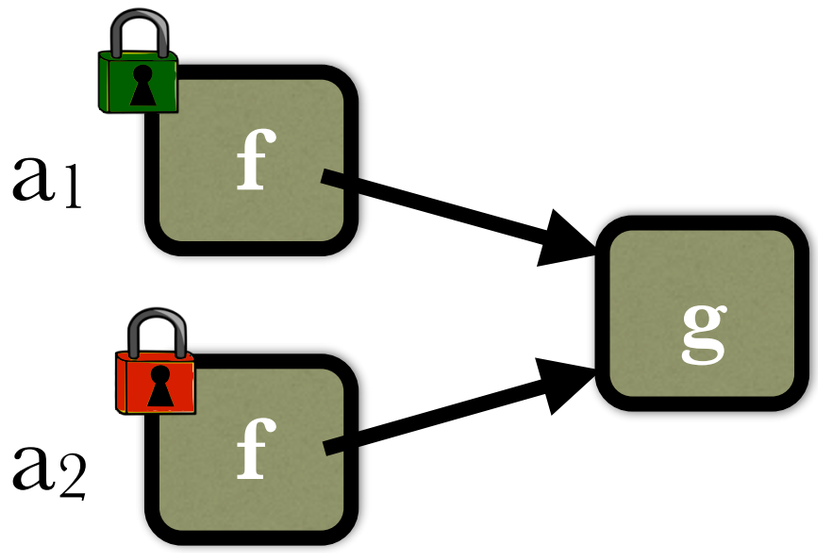
Desired object state



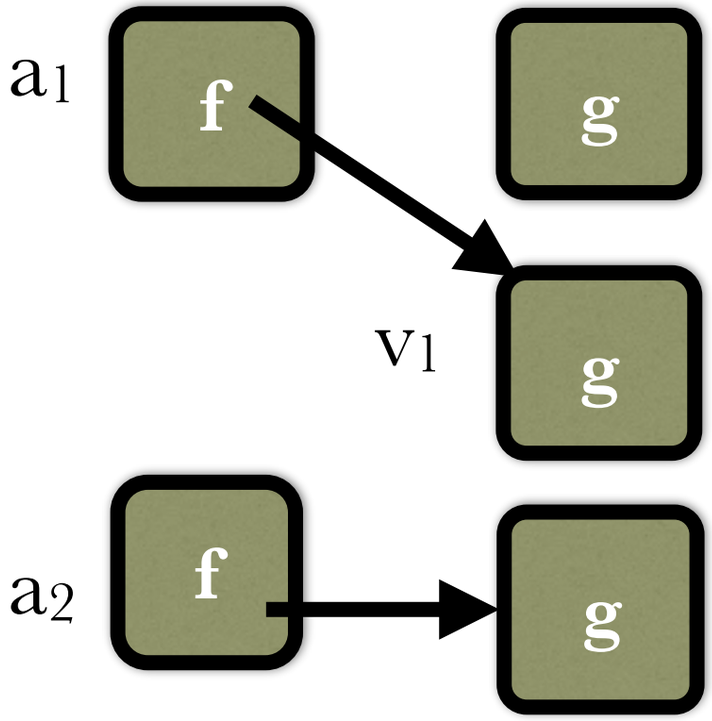
Set the context

```
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
}
```



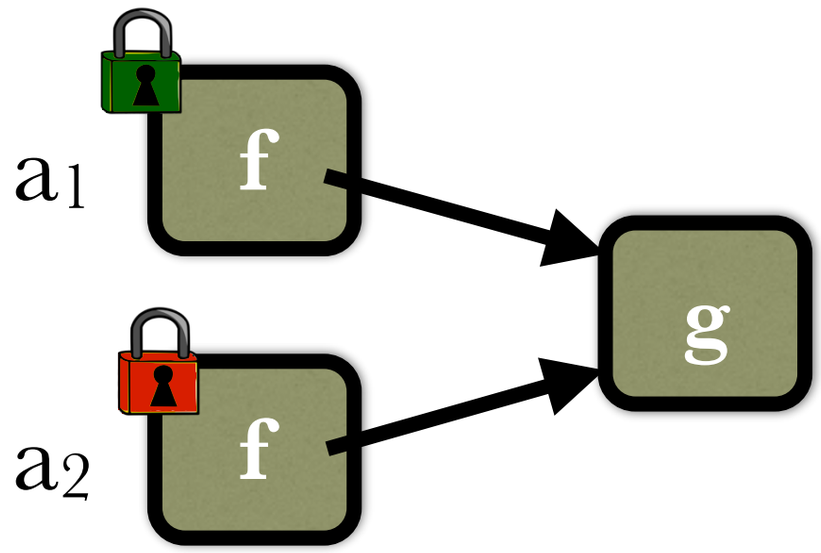
Desired object state



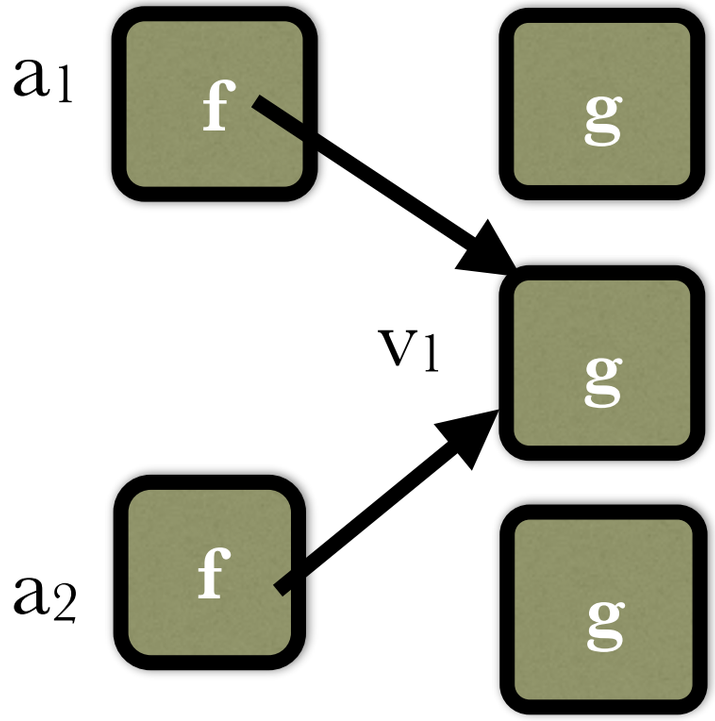
Set the context

```
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
}
```



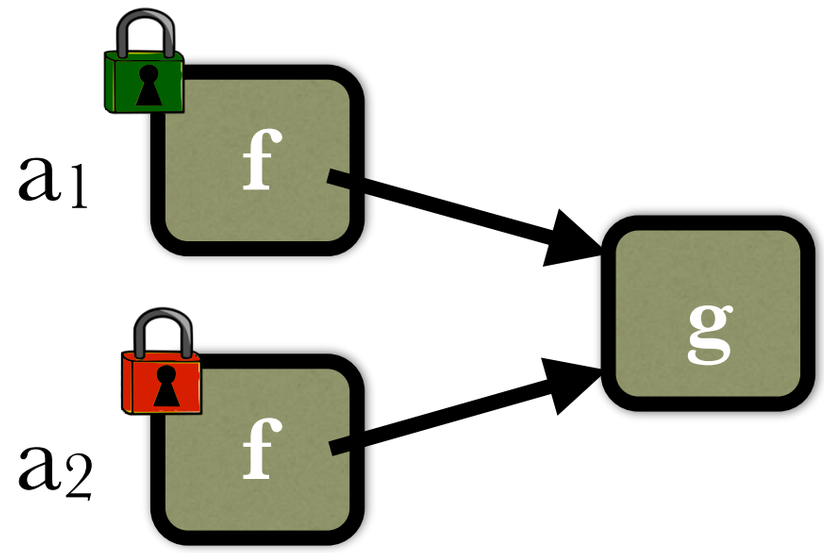
Desired object state



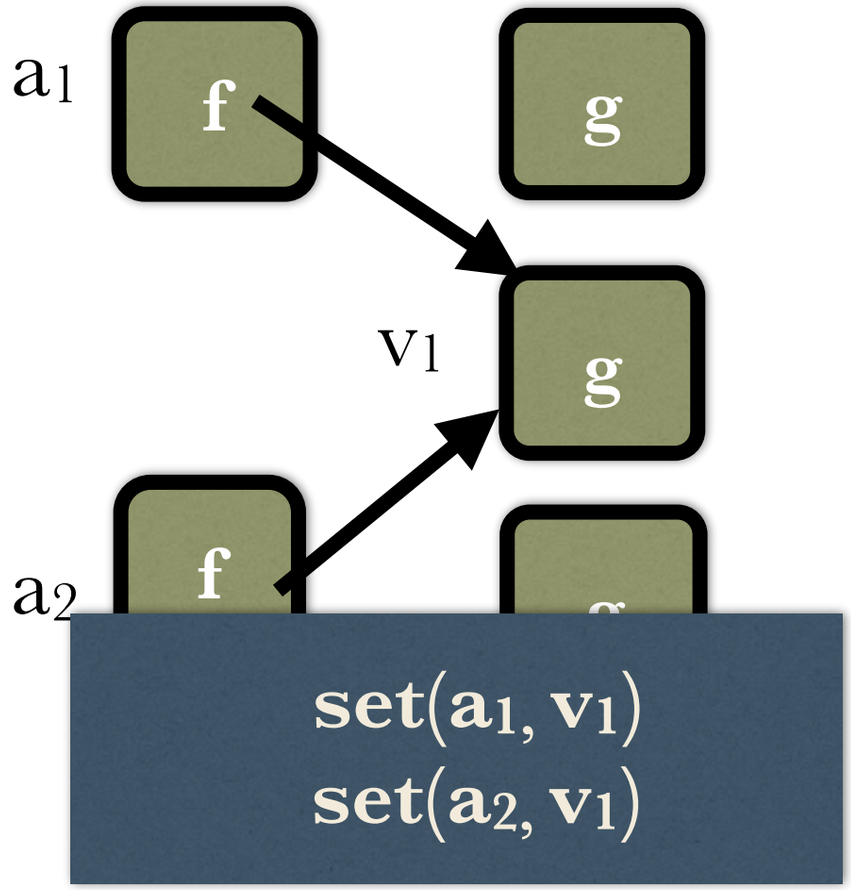
Set the context

```
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

a_1.evaluate (…)	a_1.evaluate (…)	**a_1.evaluate (…)**	a_1.evaluate (…)
a_2.reset (…)	a_2.reset (…)	a_2.reset (…)	**a_2.reset (…)**
a_3.set (f_1)	**a_3.set (f_1)**	a_3.set (f_1)	a_3.set (f_1)

set :

set :

evaluate: **reset :**

Object Generation

Execute sequential tests for obtaining object instances

a_1.evaluate (…)	a_1.evaluate (…)	**a_1.evaluate (…)**	a_1.evaluate (…)
a_2.reset (…)	a_2.reset (…)	a_2.reset (…)	**a_2.reset (…)**
a_3.set (f_1)	**a_3.set (f_1)**	a_3.set (f_1)	a_3.set (f_1)

set : (o_1, o_2)

set :

evaluate: **reset :**

Object Generation

Execute sequential tests for obtaining object instances

a_1.evaluate (…)	a_1.evaluate (…)	**a_1.evaluate (…)**	a_1.evaluate (…)
a_2.reset (…)	a_2.reset (…)	a_2.reset (…)	**a_2.reset (…)**
a_3.set (f_1)	**a_3.set (f_1)**	a_3.set (f_1)	a_3.set (f_1)

set : (o_1, o_2)

set : (o_3, o_4)

evaluate: **reset :**

Object Generation

Execute sequential tests for obtaining object instances

a_1.evaluate (…)	a_1.evaluate (…)	**a_1.evaluate (…)**	a_1.evaluate (…)
a_2.reset (…)	a_2.reset (…)	a_2.reset (…)	**a_2.reset (…)**
a_3.set (f_1)	**a_3.set (f_1)**	a_3.set (f_1)	a_3.set (f_1)

set : (o_1, o_2)

set : (o_3, o_4)

evaluate:(o_5, …) **reset :**

Object Generation

Execute sequential tests for obtaining object instances

a_1.evaluate (…)	a_1.evaluate (…)	**a_1.evaluate (…)**	a_1.evaluate (…)
a_2.reset (…)	a_2.reset (…)	a_2.reset (…)	**a_2.reset (…)**
a_3.set (f_1)	**a_3.set (f_1)**	a_3.set (f_1)	a_3.set (f_1)

set : (o_1, o_2)

set : (o_3, o_4)

evaluate:(o_5, …) **reset : (o_6, …)**

Generated Test

o_1.set(o_2)

o_3.set(o_2)

o_1.evaluate (…)

o_3.reset (…)

Generated Test

o_1.set(o_2)

o_3.set(o_2)

o_1.evaluate (…) o_3.reset (…)

Required Test Case !

Experimental Validation

*** Replication package evaluated**

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**

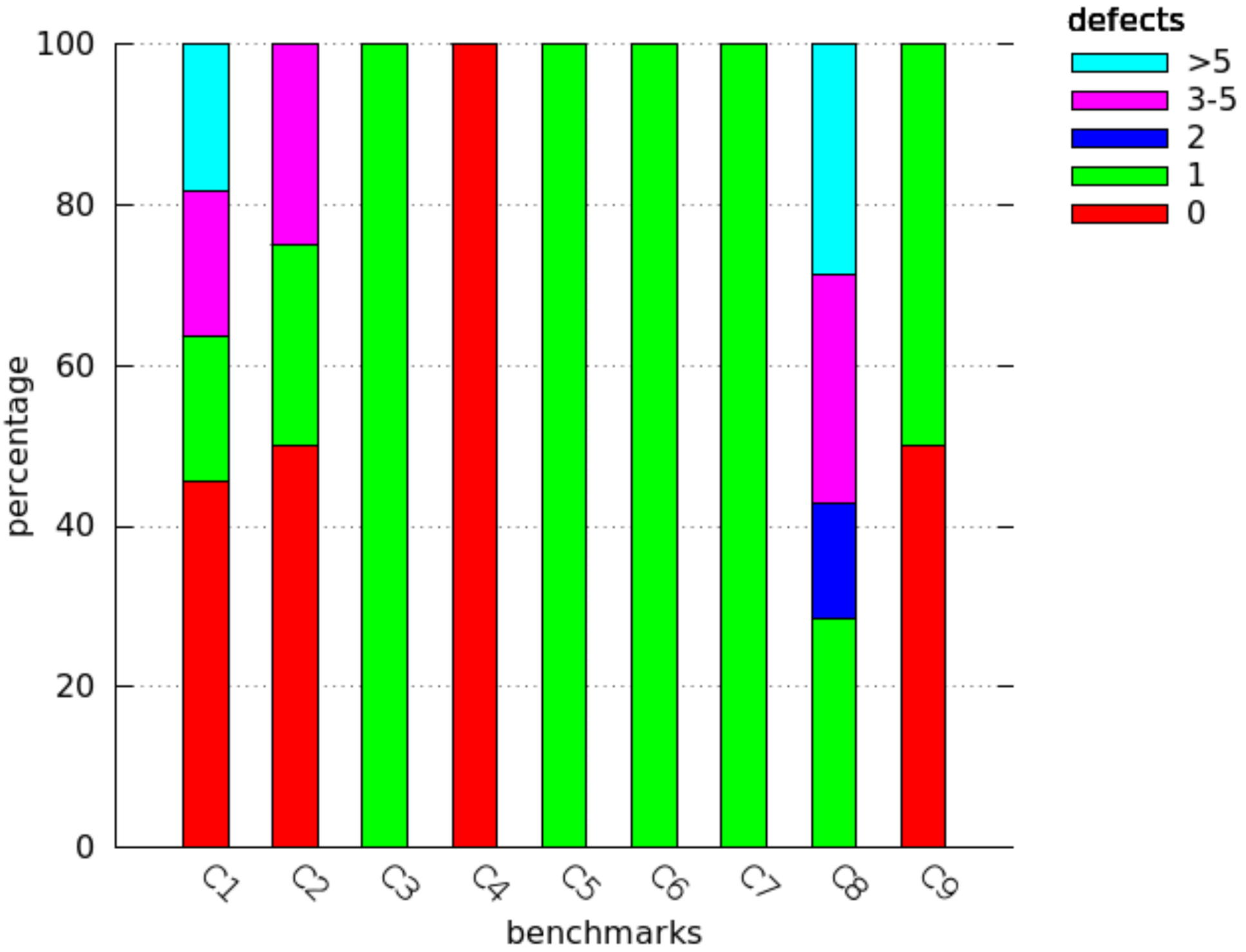
Defects Detected

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 guarante
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 t
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. It
the sequential tests, generates variable lock dependencies and constructs a set of three accesses which when interleaved suita
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 othe
synthesizing multithreaded tests.

Download

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

Indian Institute of Science, Bangalore

Replication Package

Introduction

Thread-safe libraries can help programmers avoid the complexities of multithreading. However, designing libraries that guarante
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 r
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. It
the sequential tests, generates variable lock dependencies and constructs a set of three accesses which when interleaved suita
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 othe
synthesizing multithreaded tests.

Download

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

Indian Institute of Science, Bangalore

Replication Package

Introduction

Thread-safe libraries can help programmers avoid the complexities of multithreading. However, designing libraries that guarante
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 t
designing such tests is non-trivial.

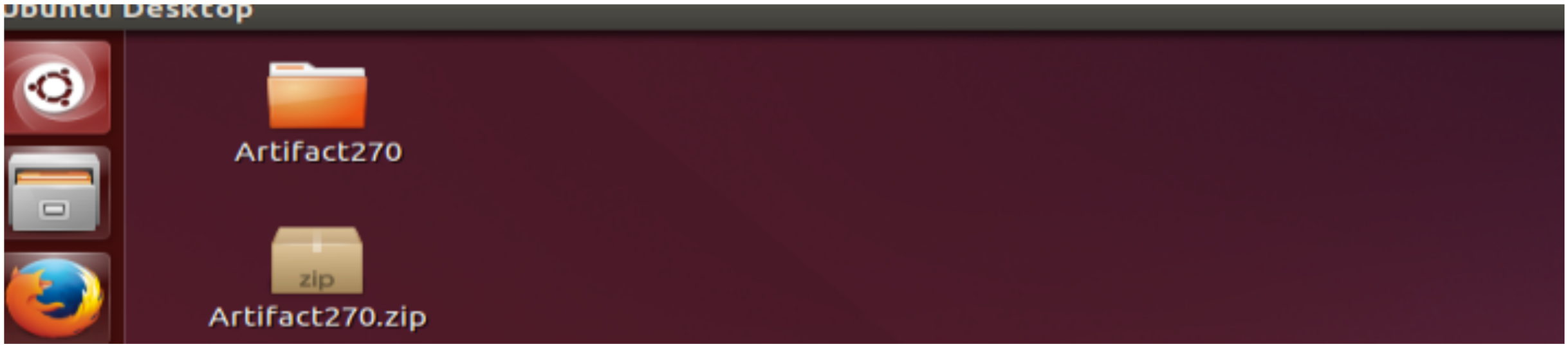
We have designed a novel and scalable tool named INTRUDER for synthesizing multithreaded tests that help detect atomicity vi
implementation of the library and a sequential seed testsuite that invokes every method in the library with random parameters. It
the sequential tests, generates variable lock dependencies and constructs a set of three accesses which when interleaved suita
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 othe
synthesizing multithreaded tests.

Download

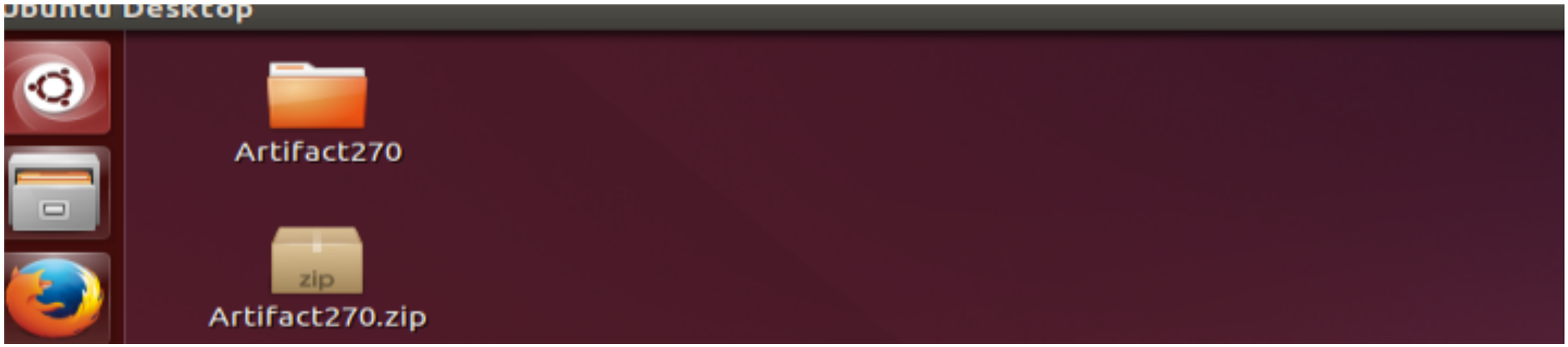
Download three tools (omen+, narada and intruder) as VM image here
Download the VM image here

Replication Package

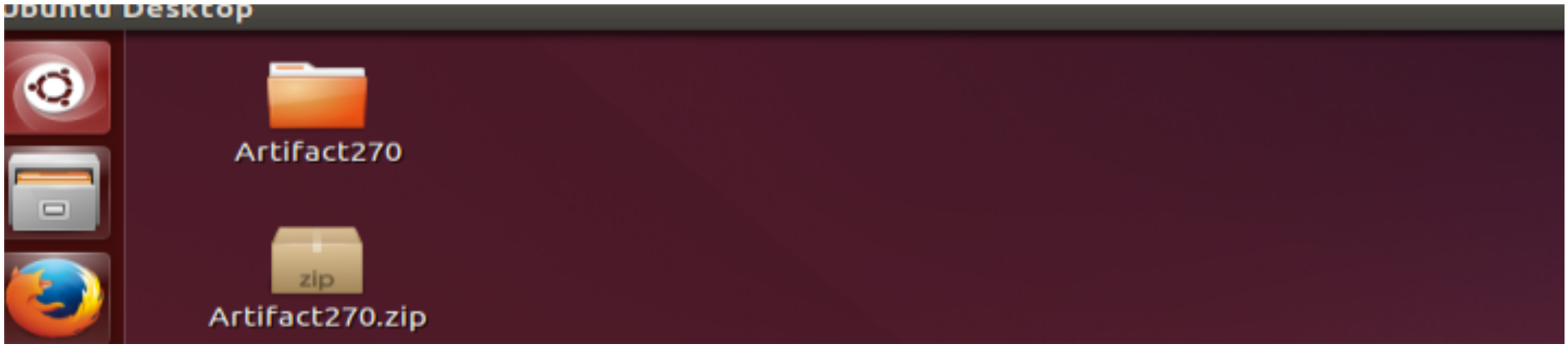


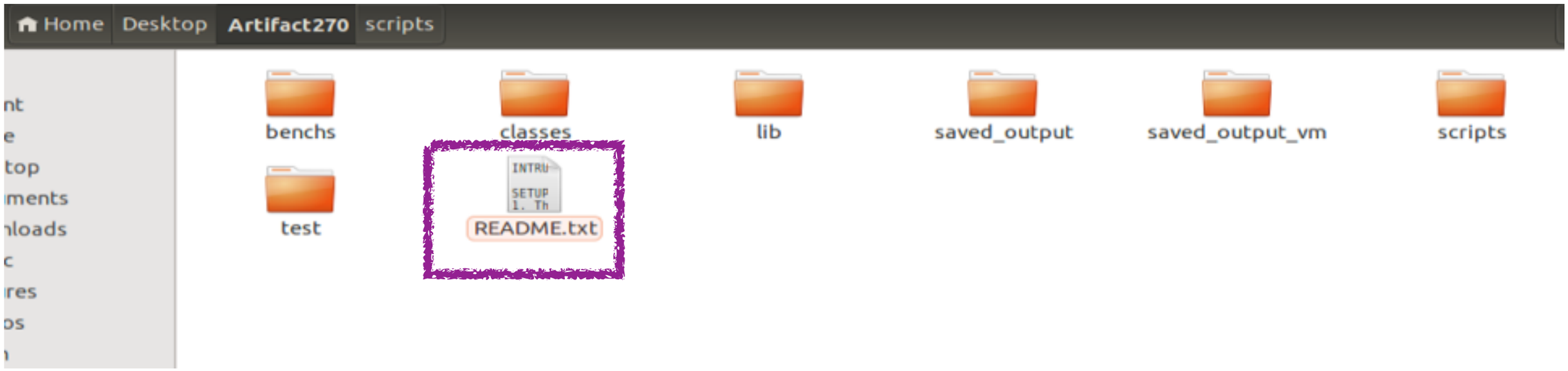
Indian Institute of Science, Bangalore

Replication Package



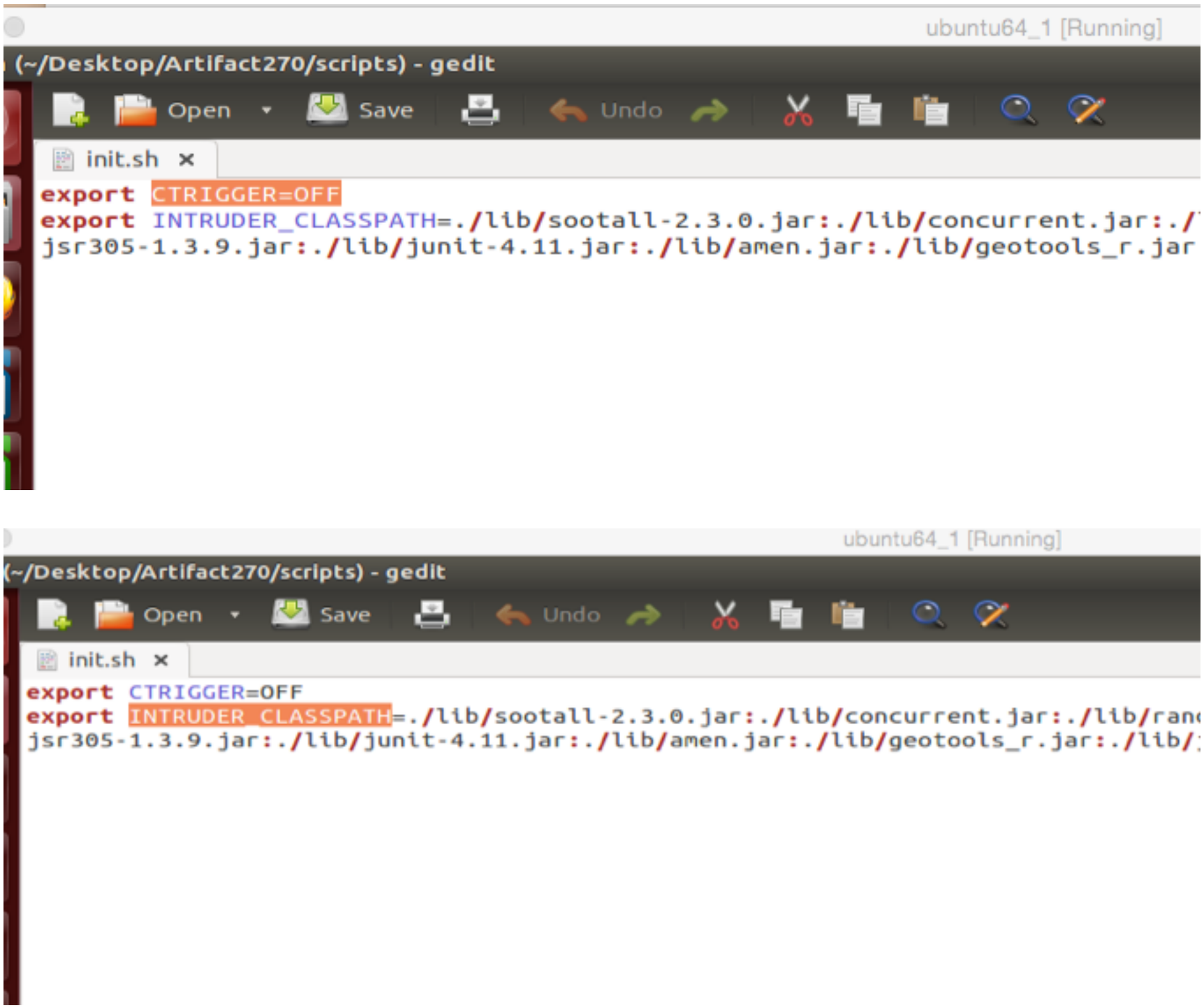
Replication Package



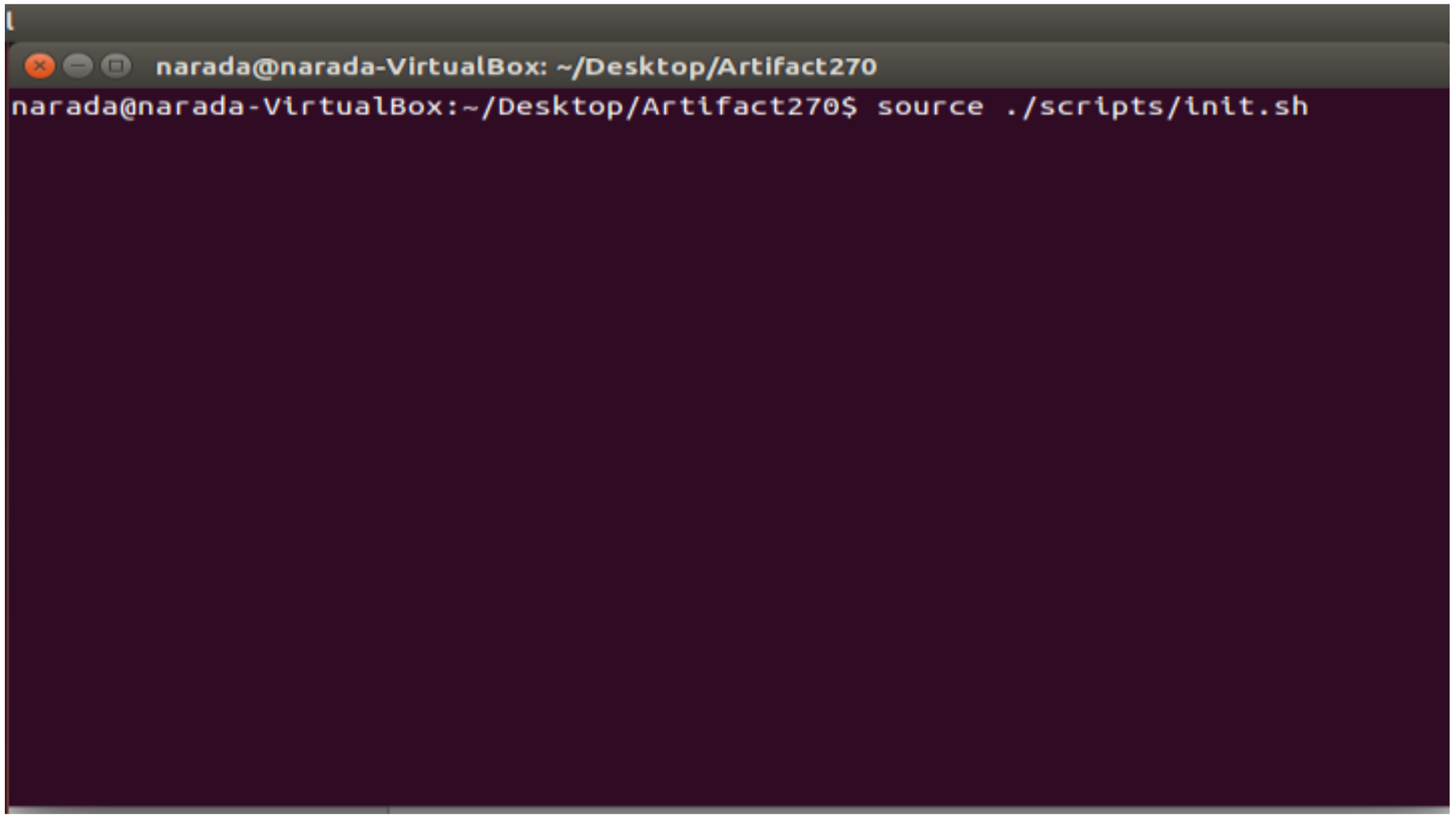


Replication Package

Replication Package



Replication Package



Replication Package

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

Open Save Undo

Library.java

```
package test2;

public class Library {
    int field = 10;

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

Replication Package

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

Open Save Undo

Library.java

```
package test2;

public class Library {
    int field = 10;

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

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

Open Save Undo

Test.java

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

Replication Package

test2.sh (~/Desktop/Artifact270/scripts) - gedit

Open Save Undo

test2.sh

```
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

test2.sh (~/Desktop/Artifact270/scripts) - gedit

Open Save Undo

test2.sh

```
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 "-----------------------------------------------------"
```

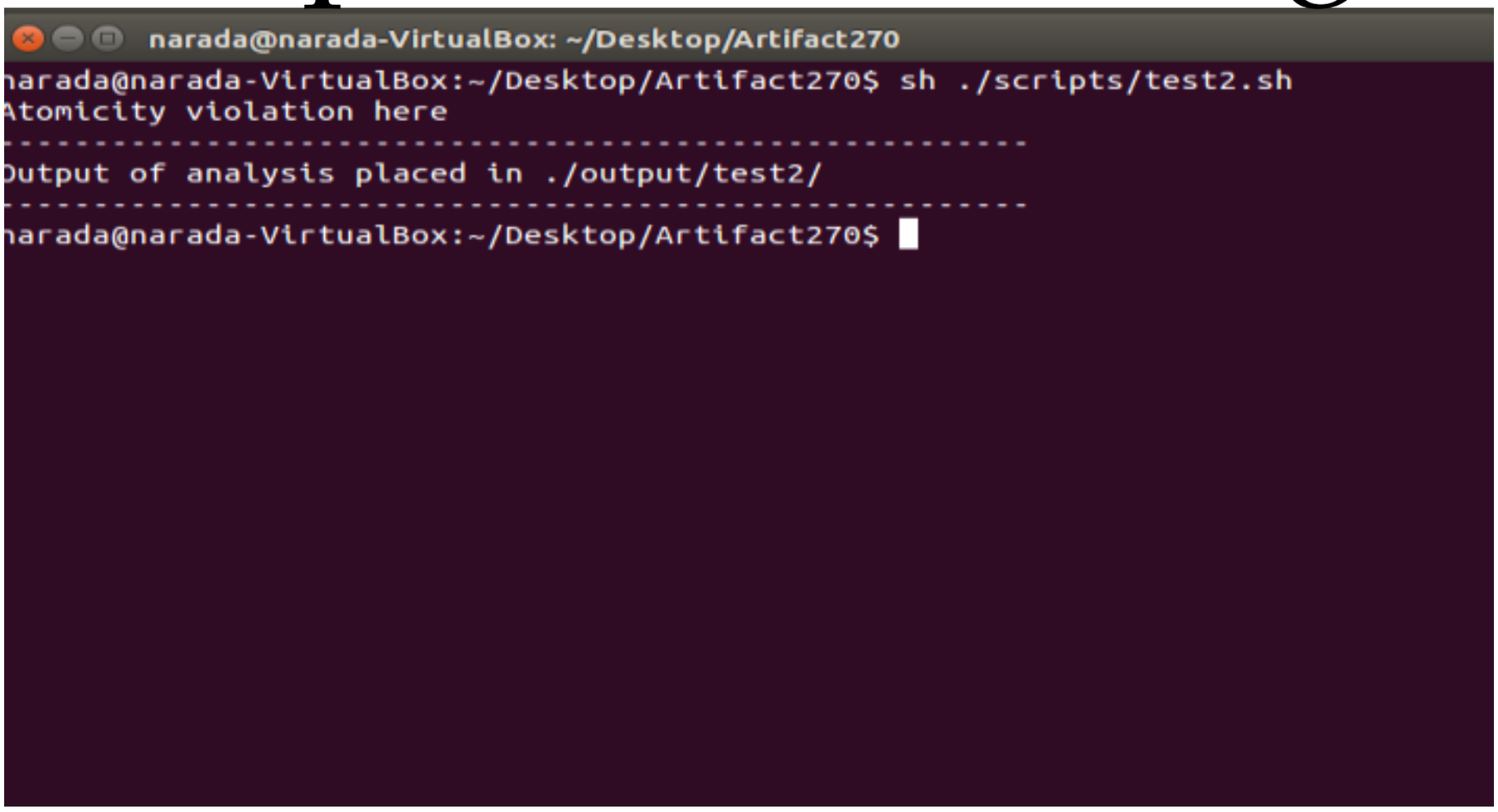
test2.sh (~/Desktop/Artifact270/scripts) - gedit

Open Save Undo

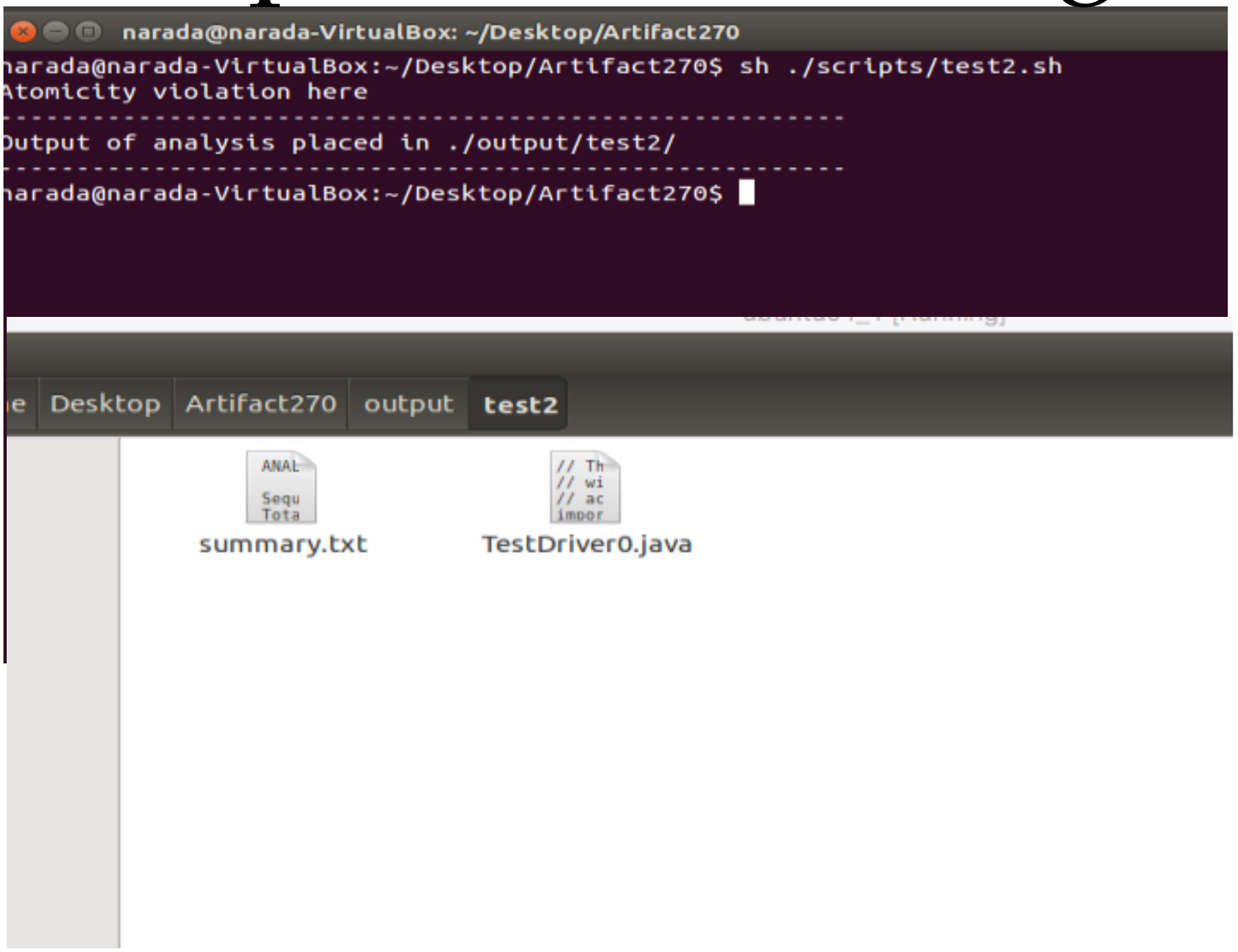
test2.sh

```
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 "-----------------------------------------------------"
```

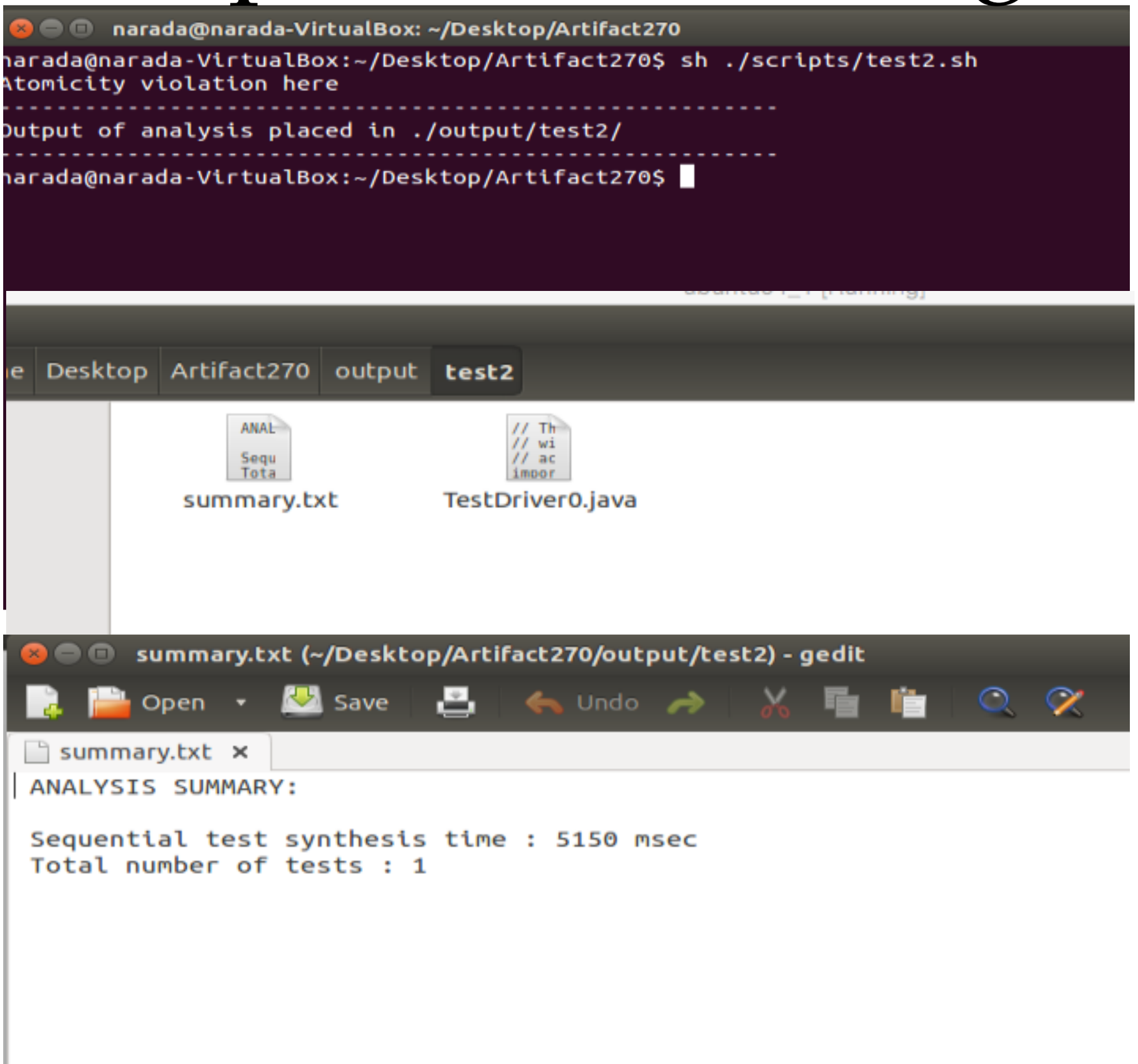
Replication Package



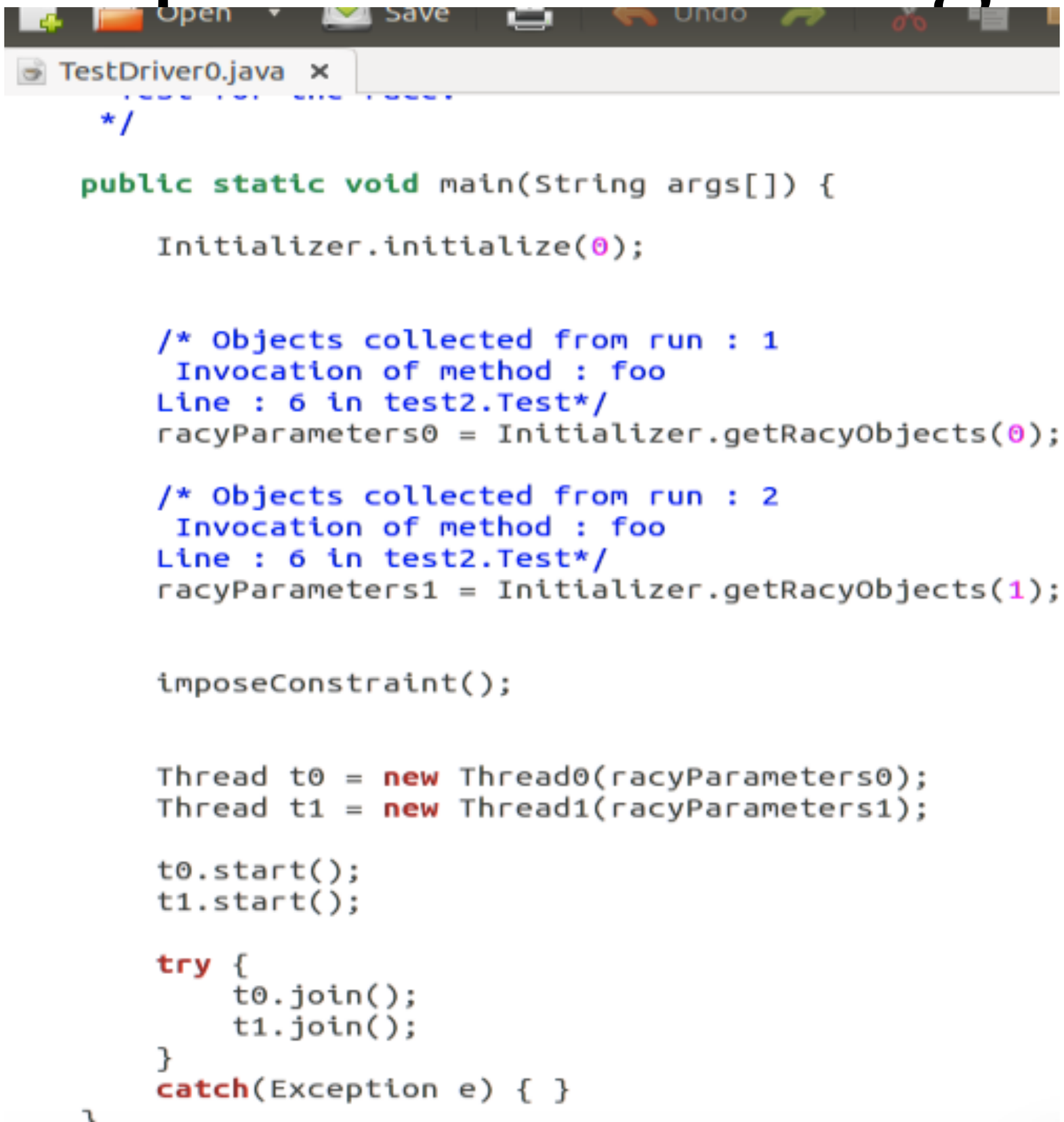
Replication Package



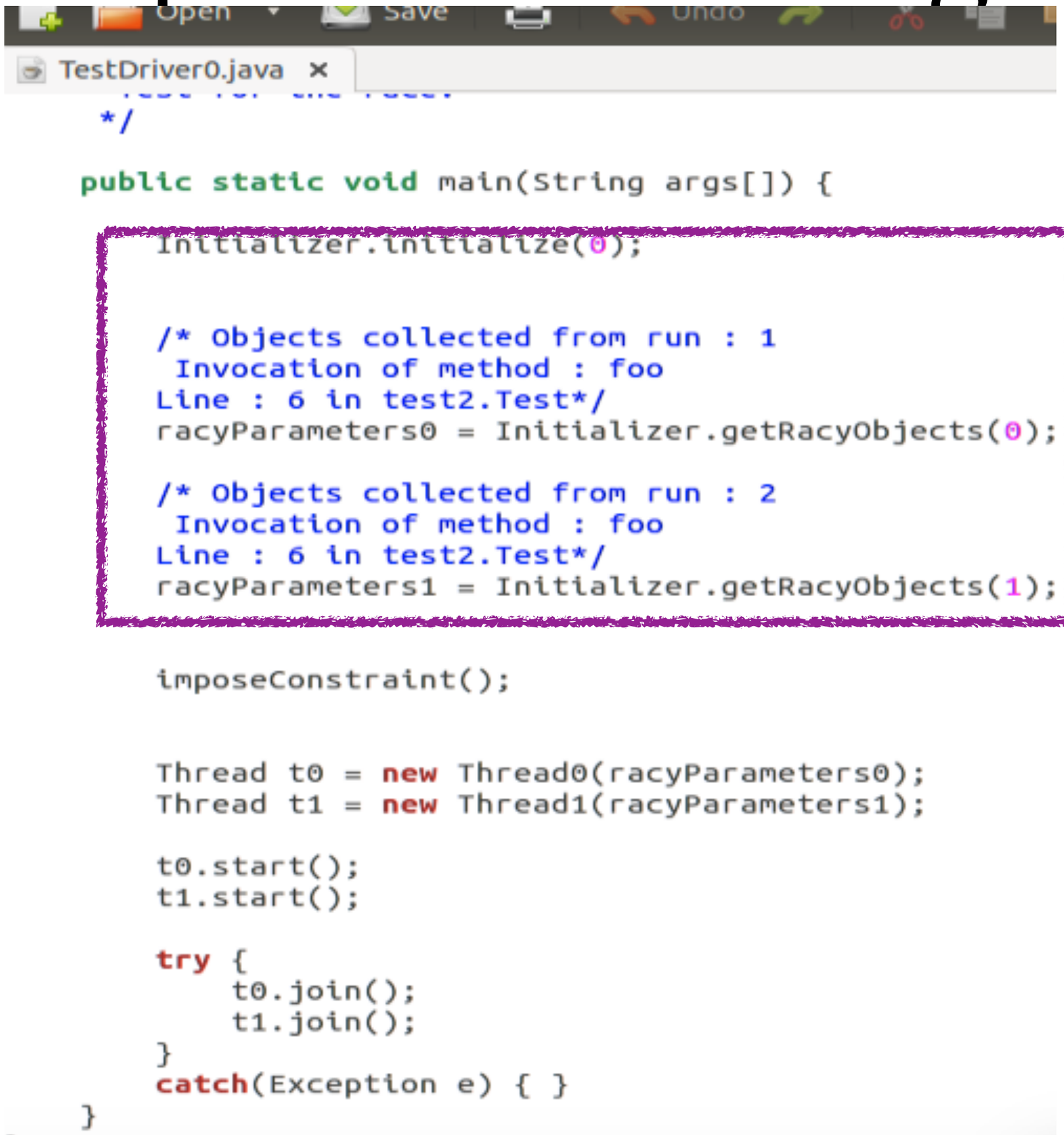
Replication Package



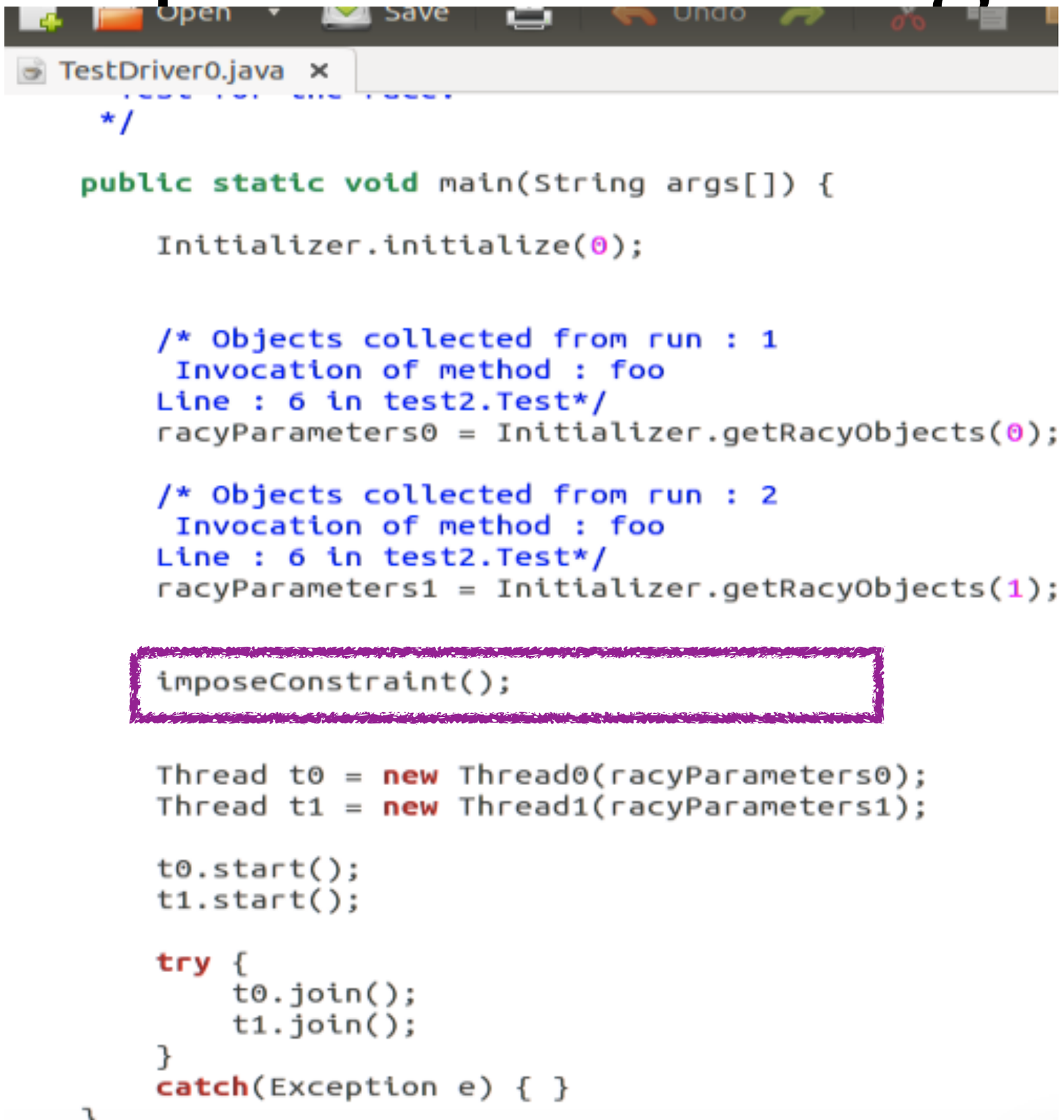
Replication Package



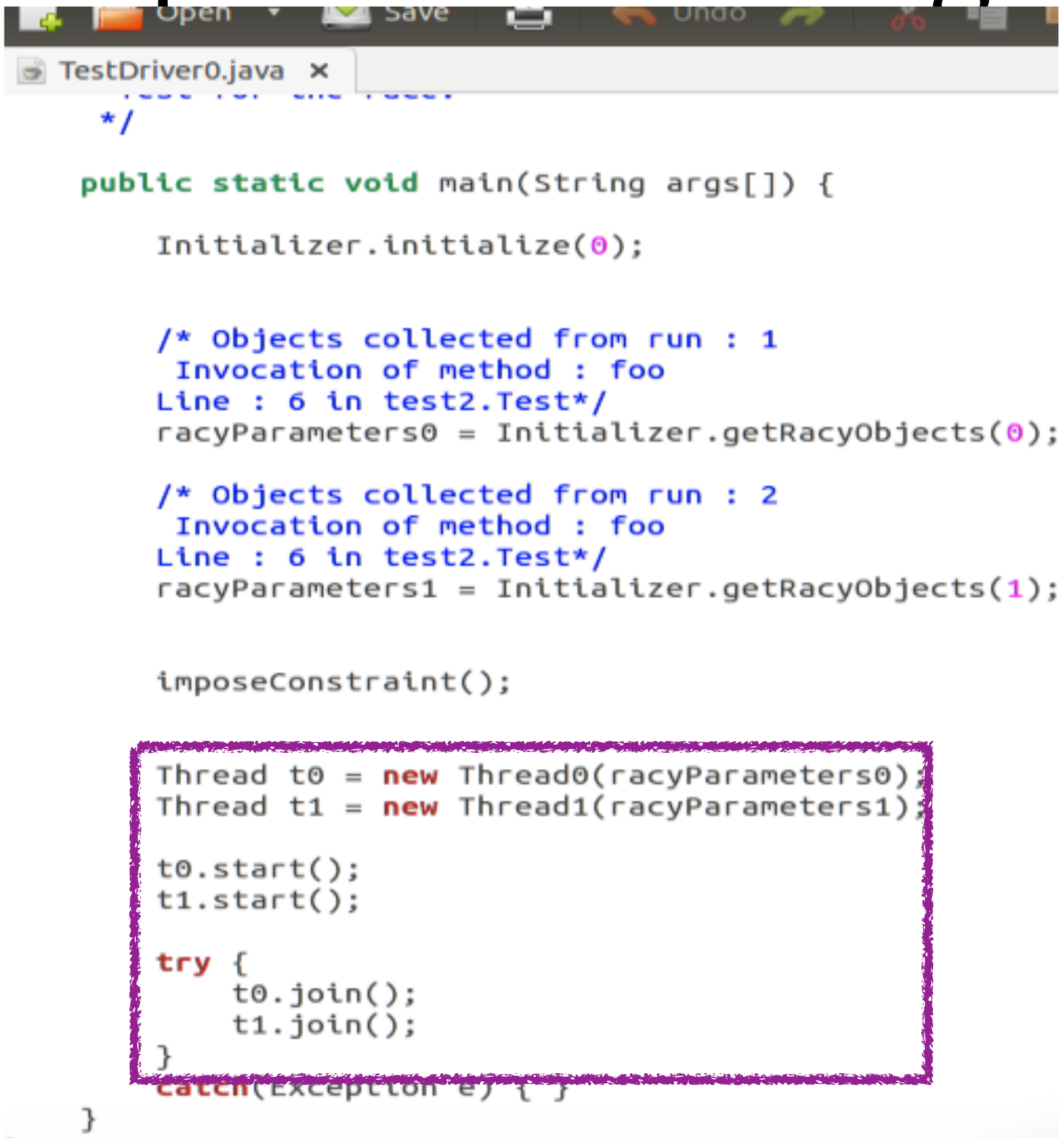
Replication Package



Replication Package



Replication Package



Replication Package

TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit

Open Save Undo

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.
```

VirtualBox

Replication Package

TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit

Open Save Undo

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.
```

VirtualBox

Replication Package

TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit

Open Save Undo

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.
```

VirtualBox

Replication Package

TestDriver0.java (~/Desktop/Artifact270/output/test2) - gedit

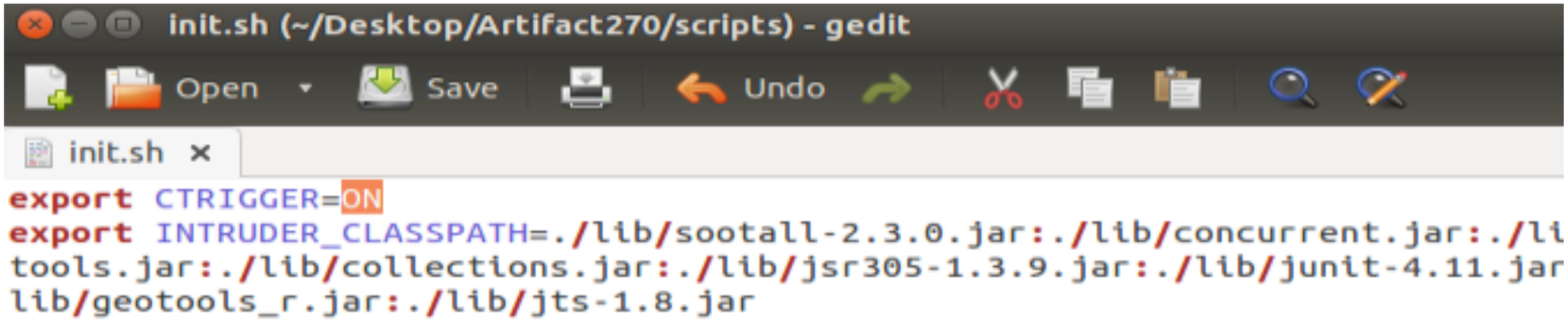
Open Save Undo

TestDriver0.java

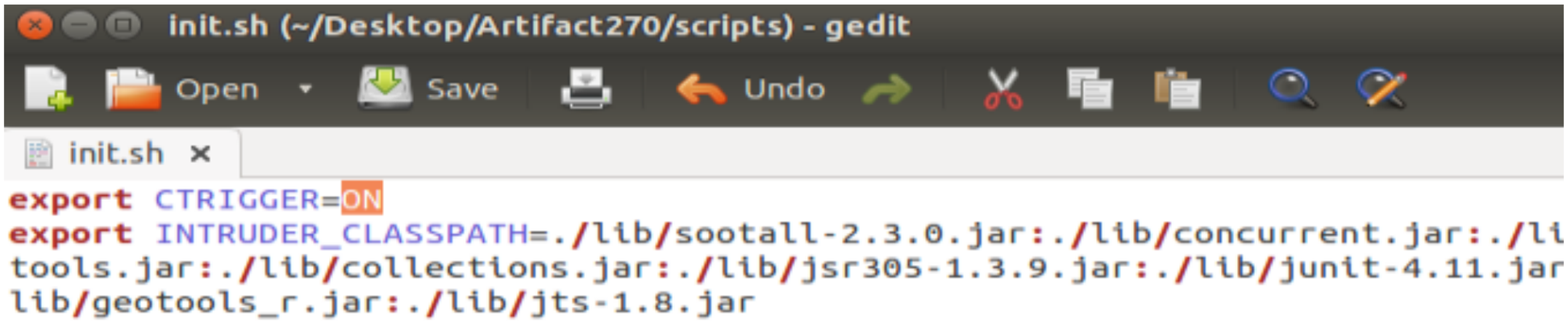
```
class Thread0 extends Thread {
    List<Parameter> parameters;

    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 {
             // Invocation leading to atomicity violation
            par1.foo();
        } catch (Exception e) {            System.out.
```

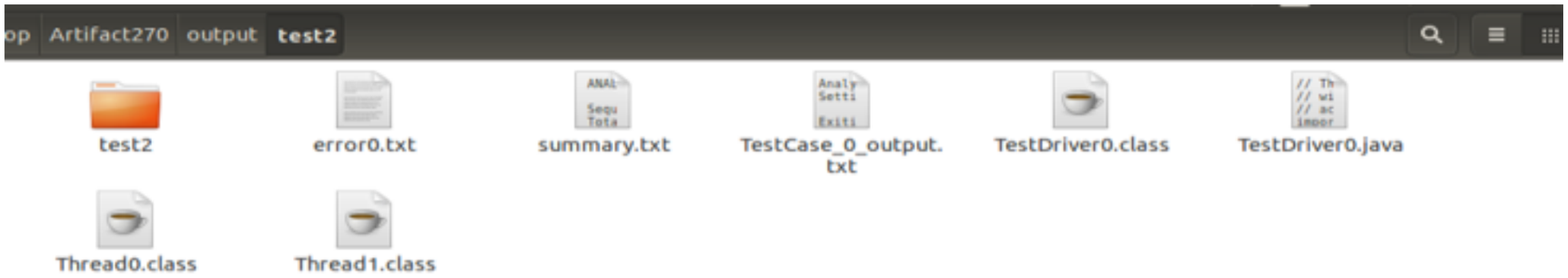
VirtualBox

Replication Package



Replication Package





Replication Package

init.sh (~/Desktop/Artifact270/scripts) - gedit

Open Save Undo

init.sh

```
export CTRIGGER=ON
export INTRUDER_CLASSPATH=./lib/sootall-2.3.0.jar:./lib/concurrent.jar:./li
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
```

op Artifact270 output test2


test2


error0.txt


summary.txt


TestCase_0_output.txt


TestDriver0.class


TestDriver0.java


Thread0.class


Thread1.class

put test2

summary.txt (~/Desktop/Artifact270/output/test2) - gedit

Open Save Undo

summary.txt

```
ANALYSIS SUMMARY:

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

Replication Package

TestCase_0_output.txt

```
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 t

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 t



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/