Synthesizing Racy Tests

Malavika Samak (IISc)
Murali Krishna Ramanathan (IISc)
Suresh Jagannathan (Purdue)

Introduction

Introduction

- Developers use multithreaded libraries as building blocks.
- Do the client applications need to acquire additional locks to avoid data races?
- 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 data races?

- If YES
  - what locks need to be acquired?
  - which method invocations require these acquisitions?

Apply Dynamic Race Detection

Dynamic Race Detection

Dynamic Race Detection

Library

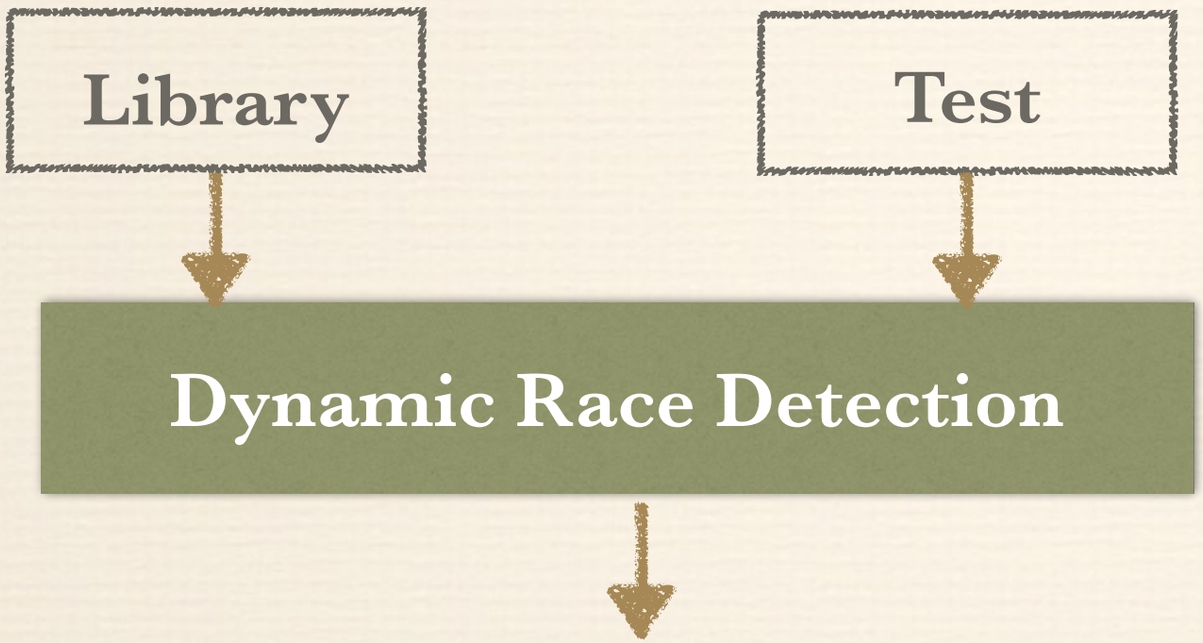
Dynamic Race Detection

Library

Test

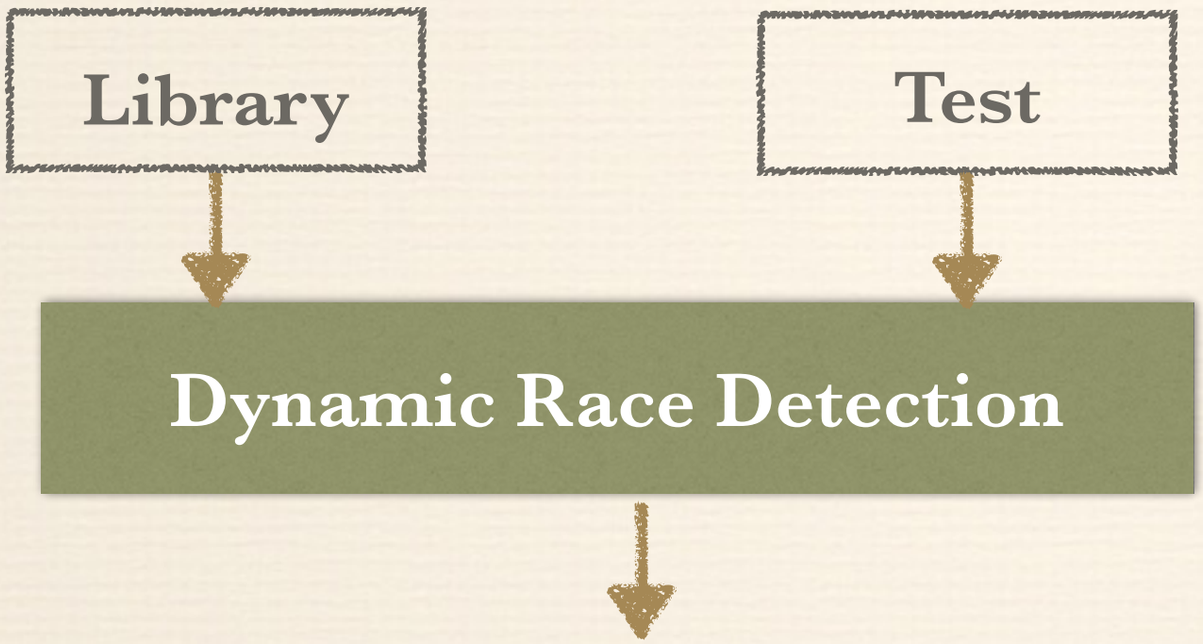
Dynamic Race Detection

Dynamic Race Detection



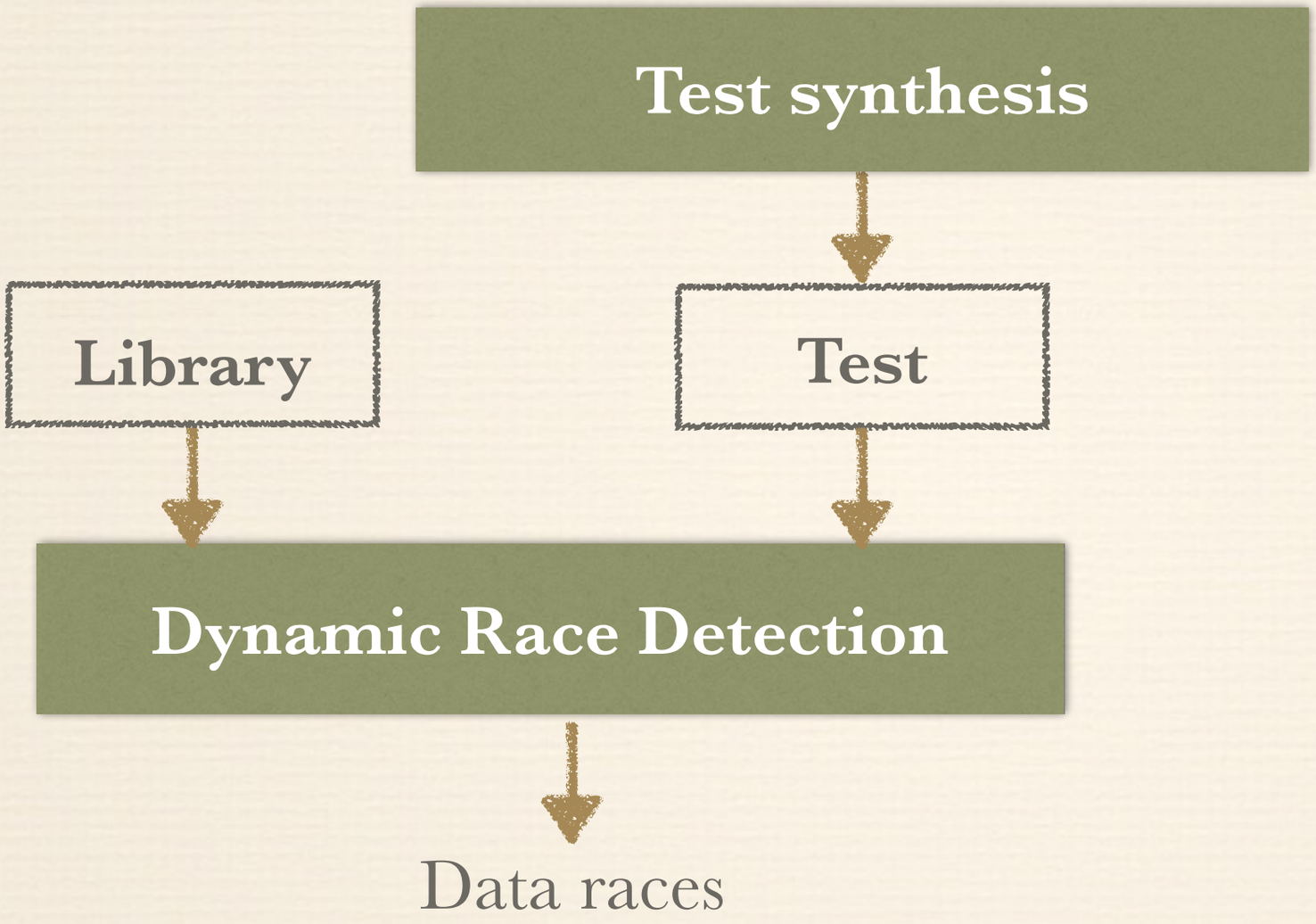
- **Eraser,** SOSP'97
- **DJIT+,** PPoPP'03
- **RaceFuzzer,** PLDI'08
- **FastTrack,** PLDI'09
- **Causally Precedes,** POPL'12

Dynamic Race Detection

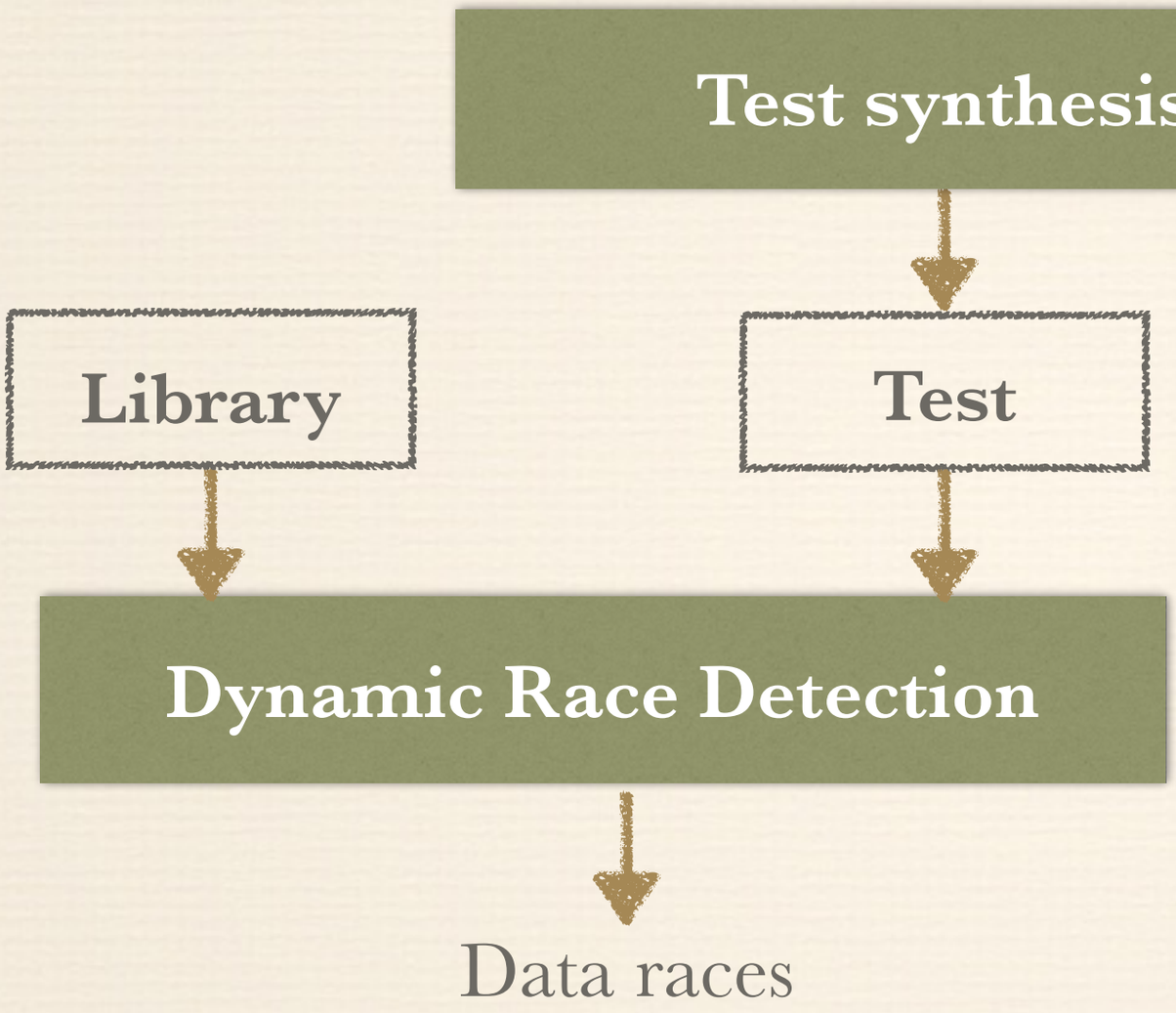


- **Significant manual effort**
- **Higher false negatives**

Dynamic Race Detection



Dynamic Race Detection



Goal of this work

Motivating example

Motivating example

- hazelcast-3.3.2
  - in-memory data grid
  - manage high data ingest rates
  - 82 contributors, 68 releases, 12K commits

Motivating example

- hazelcast-3.3.2
  - in-memory data grid
  - manage high data ingest rates
  - 82 contributors, 68 releases, 12K commits
- SynchronizedWriteBehindQueue class is declared thread-safe
  - clients need not acquire additional locks

Motivating example

- hazelcast-3.3.2
  - in-memory data grid
  - manage high data ingest rates
  - 82 contributors, 68 releases, 12K commits
- SynchronizedWriteBehindQueue class is declared thread-safe
  - clients need not acquire additional locks
- Class is **NOT** thread-safe
  - corner case leads to a race
  - requires a complex multi-threaded test

Motivating example

Motivating example

t_1: a.removeFirst()

t_2: b.removeFirst()

Motivating example

t_1: a.removeFirst()

t_2: b.removeFirst()

- Which methods need to be invoked concurrently?

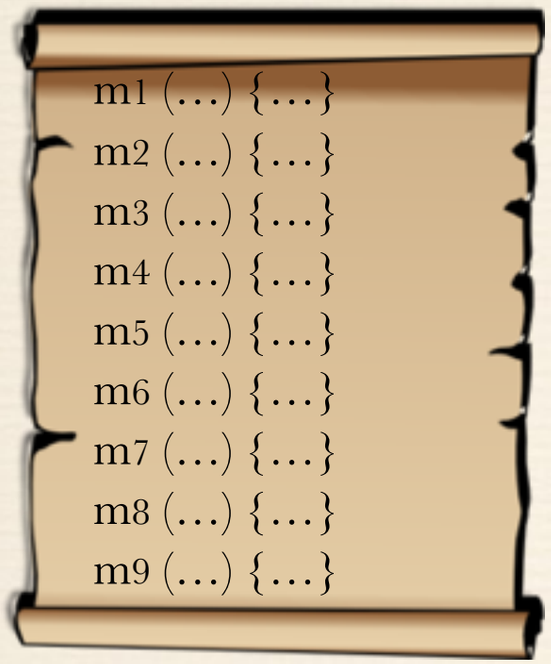
Motivating example

t_1: a.removeFirst()

t_2: b.removeFirst()

- Which methods need to be invoked concurrently?
- What parameters need to be passed to these methods?

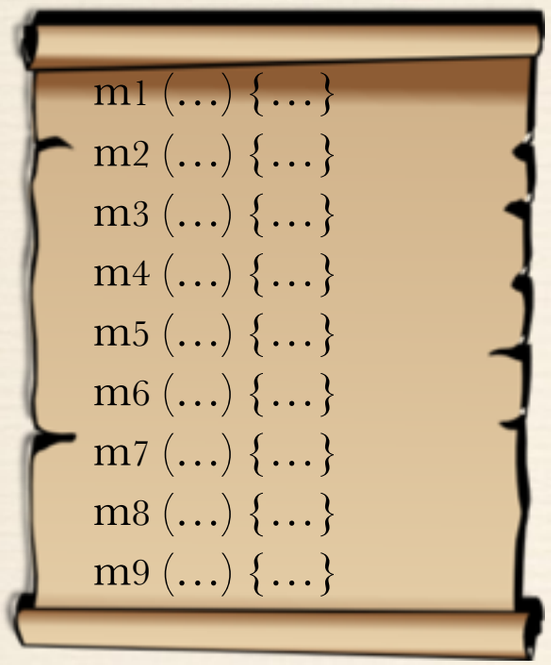
Random Test Generation



Library

Possible Parameters

Random Test Generation



Library

Possible Parameters

t_1: m1( )
t_2: m2( )

t_1: m1( )
t_2: m2( )

t_1: m1( )
t_2: m1( )

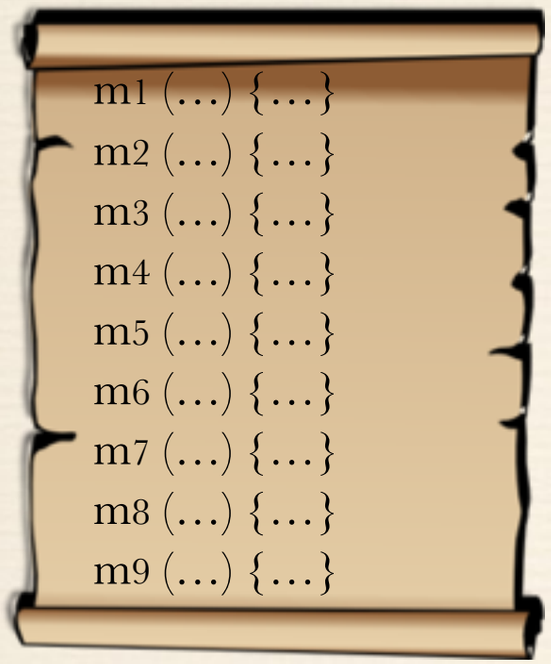
t_1: m1( )
t_2: m1( )

t_1: m2( )
t_2: m2( )

t_1: m1( )
t_2: m2( )

t_1: m2( )
t_2: m2( )

Random 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(● ● ●)

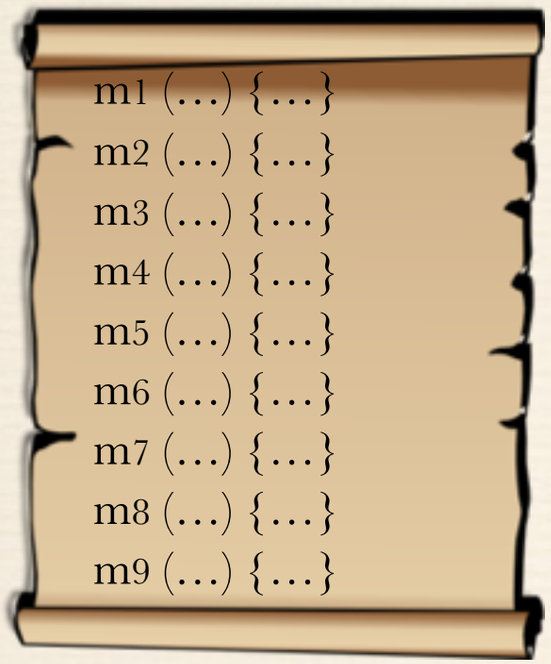
t_1: m1(● ● ●)
t_2: m1(● ● ●)

t_1: m2(● ●)
t_2: m2(● ●)

t_1: m1(● ●)
t_2: m2(● ●)

t_1: m2(● ●)
t_2: m2(● ●)

Random Test Generation



Library

Possible Parameters

t_1: m1(● ● ●)
t_2: m2(● ●)

m1(● ● ●)
m2(● ●)

m1(● ● ●)
m1(● ● ●)

t_1: m1(● ● ●)
t_2: m2(● ●)

m1(● ● ●)
m1(● ● ●)

m1(● ● ●)
m1(● ● ●)

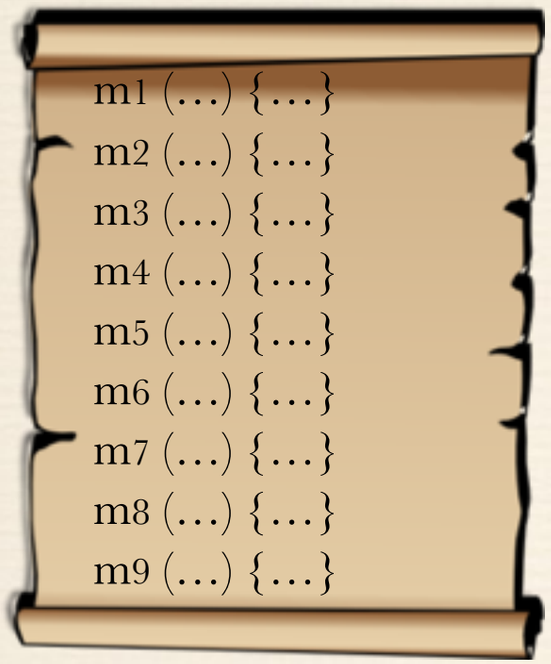
t_1: m2(● ●)
t_2: m2(● ●)

m1(● ●)
m2(● ●)

m2(● ●)
m2(● ●)

…

Random 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: m1(● ● ●)

t_1: m2(● ●)
t_2: m2(● ●)

t_1: m1(● ●)
t_2: m2(● ●)

t_1: m2(● ●)
t_2: m2(● ●)

…

Random Test Generation

§ § §

m1(● ● ●)
m2(● ●)

m1(● ● ●)
m2(● ●)

m1(● ● ●)
m1(● ● ●)

m1(● ● ●)
m2(● ●)

m1(● ● ●)
m1(● ● ●)

m1(● ● ●)
m1(● ● ●)

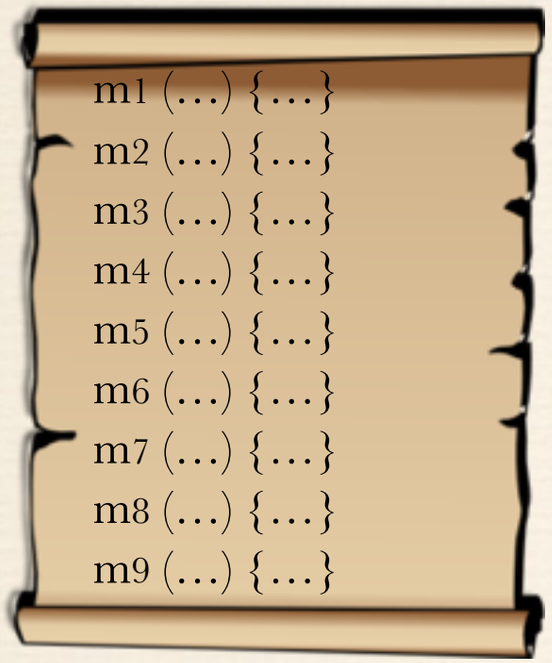
m2(● ●)
m2(● ●)

m2(● ●)
m2(● ●)

m2(● ●)
m2(● ●)

...

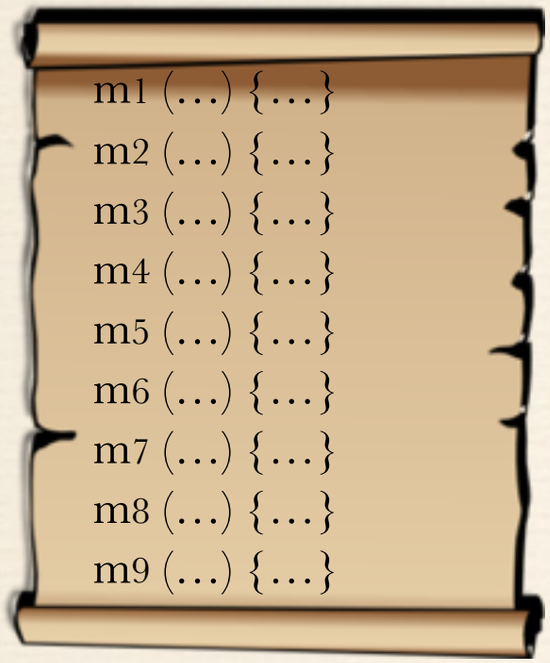
Targeted Test Synthesis



Library

Possible Parameters

Targeted Test Synthesis

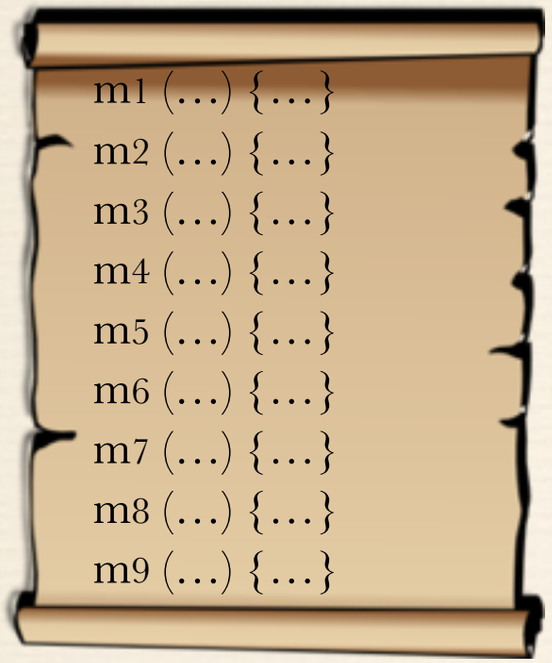


Library

Possible Parameters

m1( )

Targeted Test Synthesis

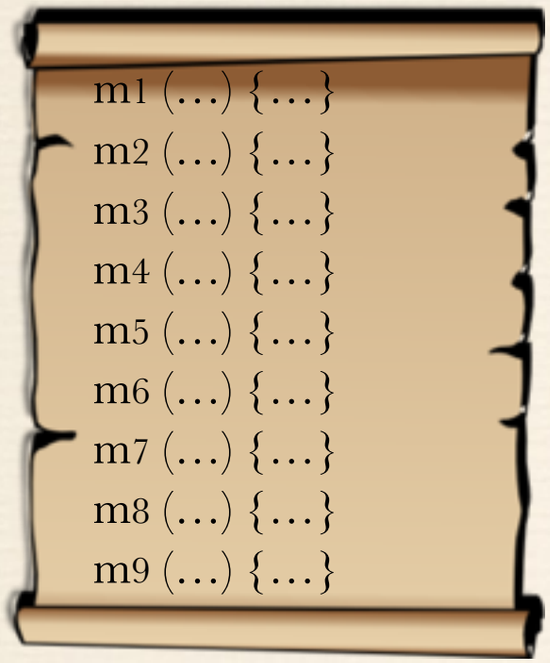


Library

Possible Parameters

m1( ) **m4**( )

Targeted Test Synthesis



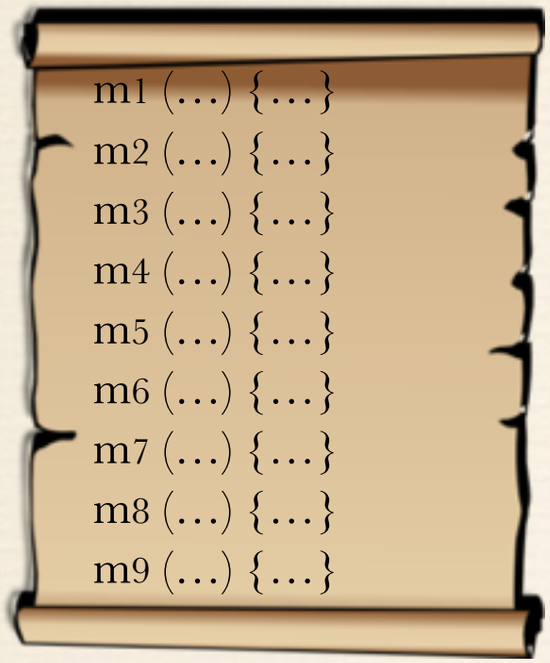
Library

Possible Parameters

m1(● ● ● **)**

m4(**)**

Targeted Test Synthesis



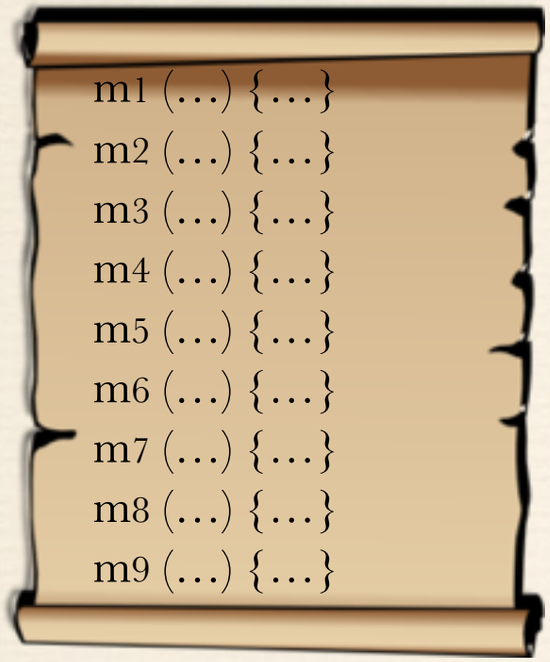
Library

Possible Parameters

m1()

m4()

Targeted Test Synthesis



Library

Possible Parameters

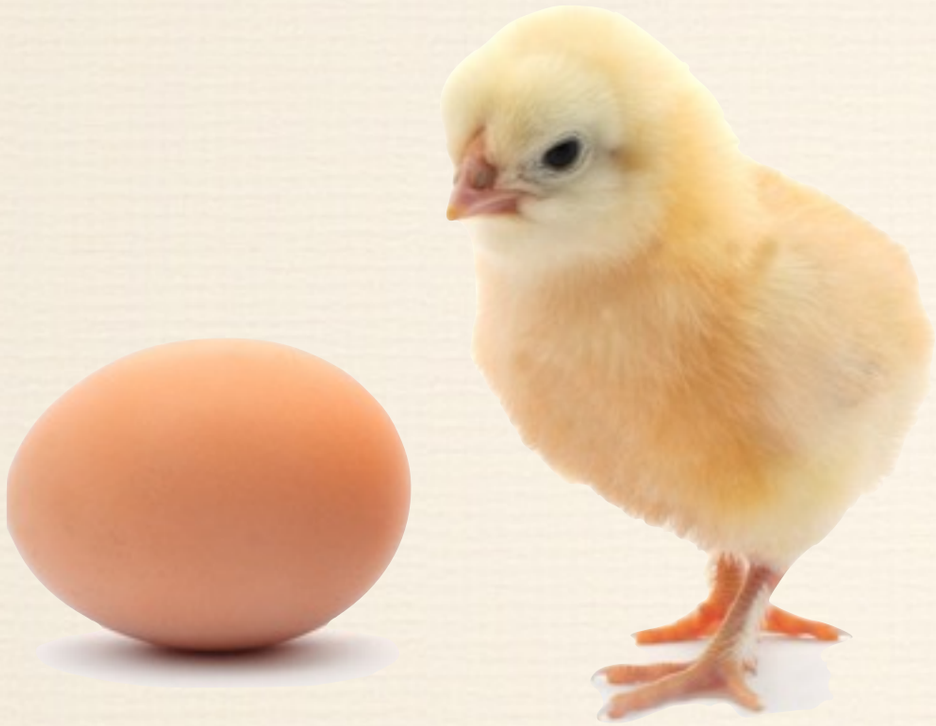
t_1

m1(● ● ●)

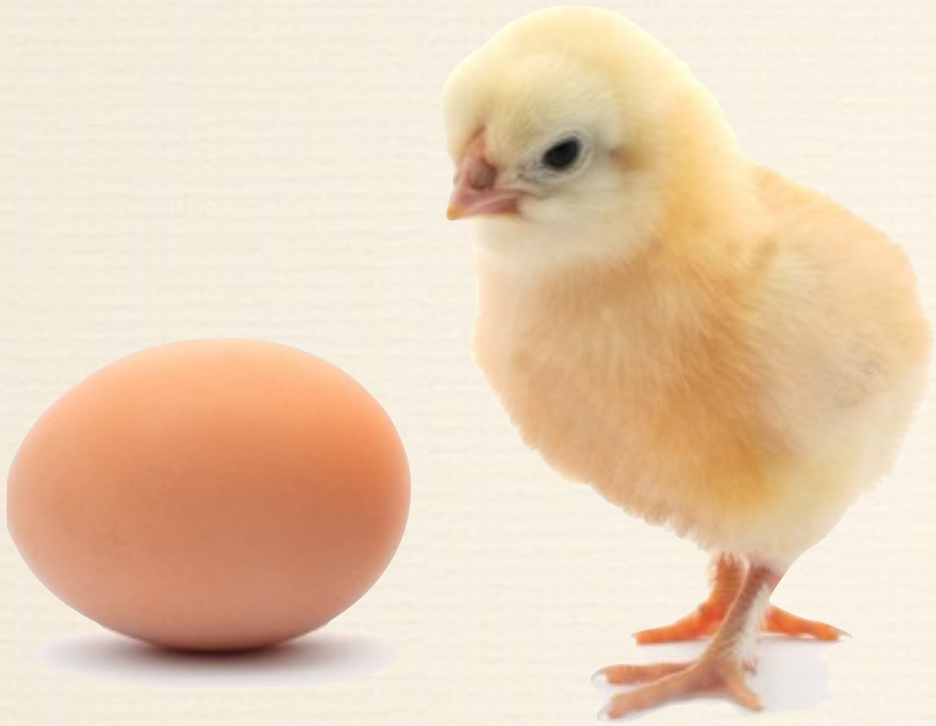
t_2

m4(● ●)

Conundrum

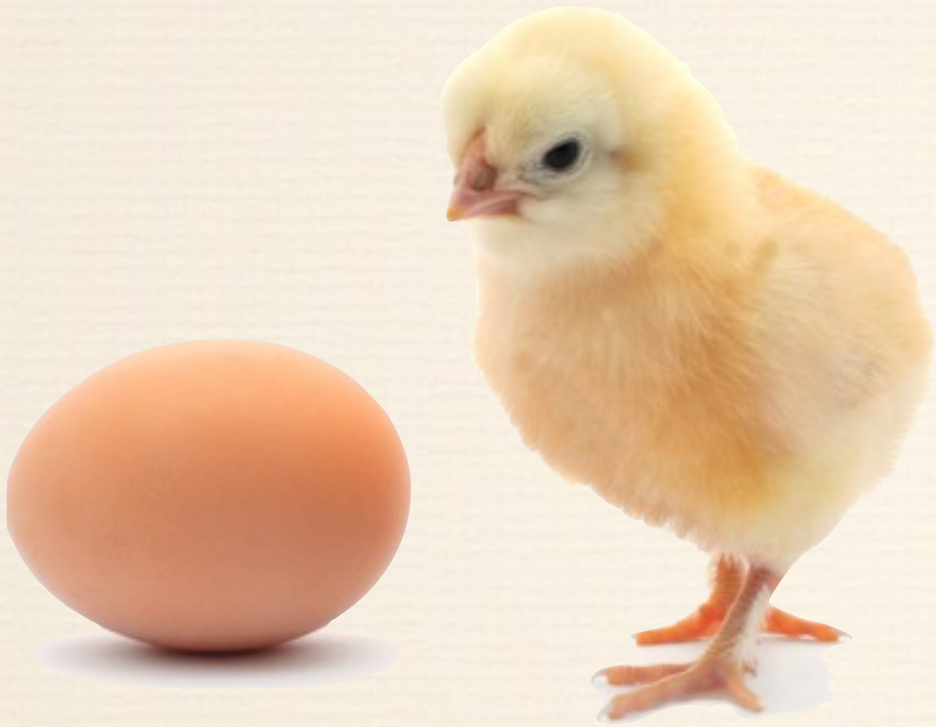


Conundrum



- Dynamic race detection requires effective tests

Conundrum



- Dynamic race detection requires effective tests
- Test synthesis requires some knowledge of races

Key Insight

Use data from sequential test execution to construct multithreaded tests

Overview

Overview

Library **Sequential Tests**

Overview

Library

Sequential Tests

Overview

Library **Sequential Tests**

Overview

Library

Sequential Tests

NARADA

Multithreaded Tests

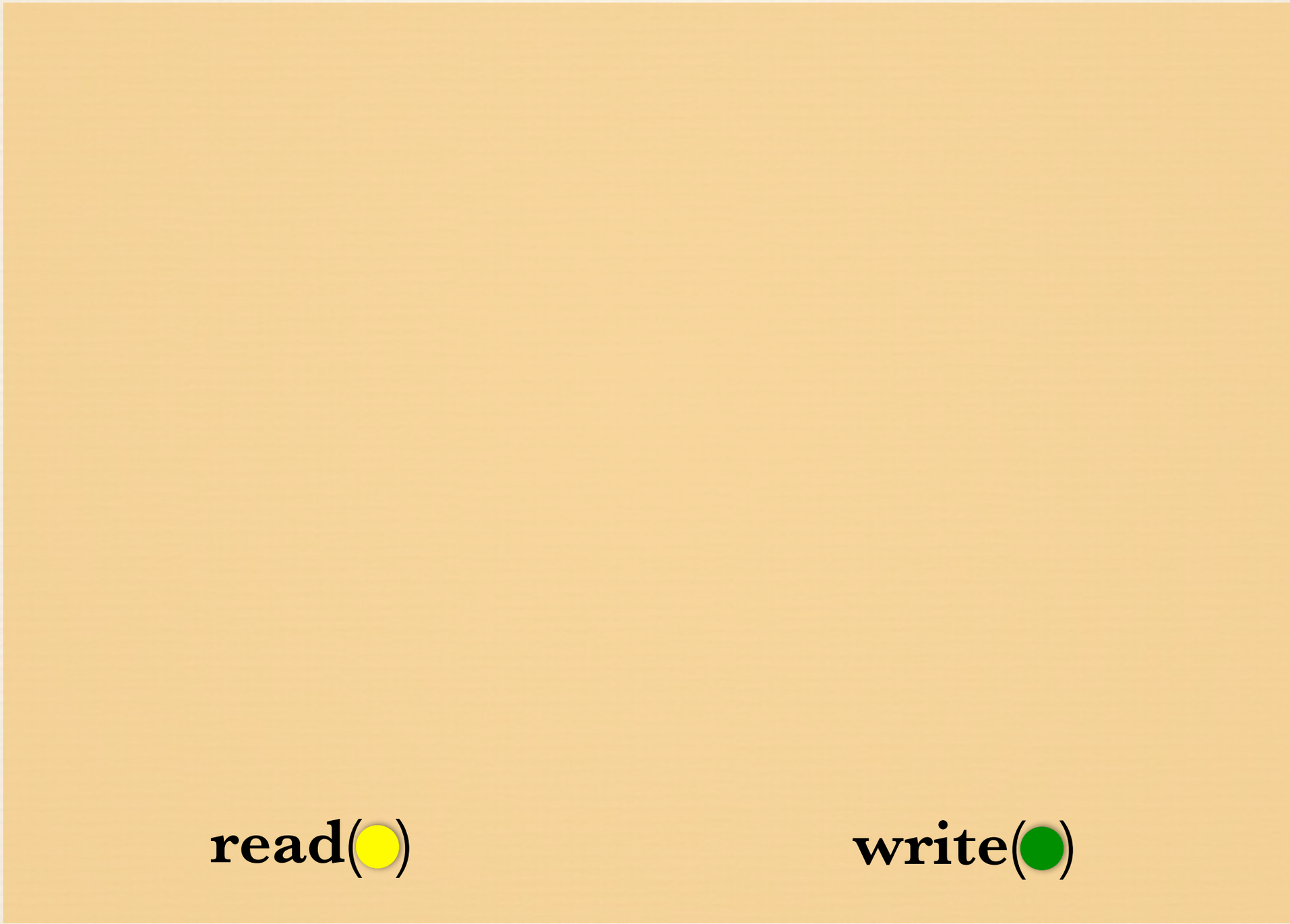
Challenge 1: Identify access pairs

T_1	**write(●)**	**read(●)**	**…**	**write(●)**
T_2	**read(●)**	**write(●)**	**…**	**read(●)**
T_3	**write(●)**	**read(●)**	**…**	**write(●)**
		…		
T_{n-1}	**write(●)**	**read(●)**	**…**	**write(●)**
T_n	**read(●)**	**read(●)**	**…**	**read(●)**

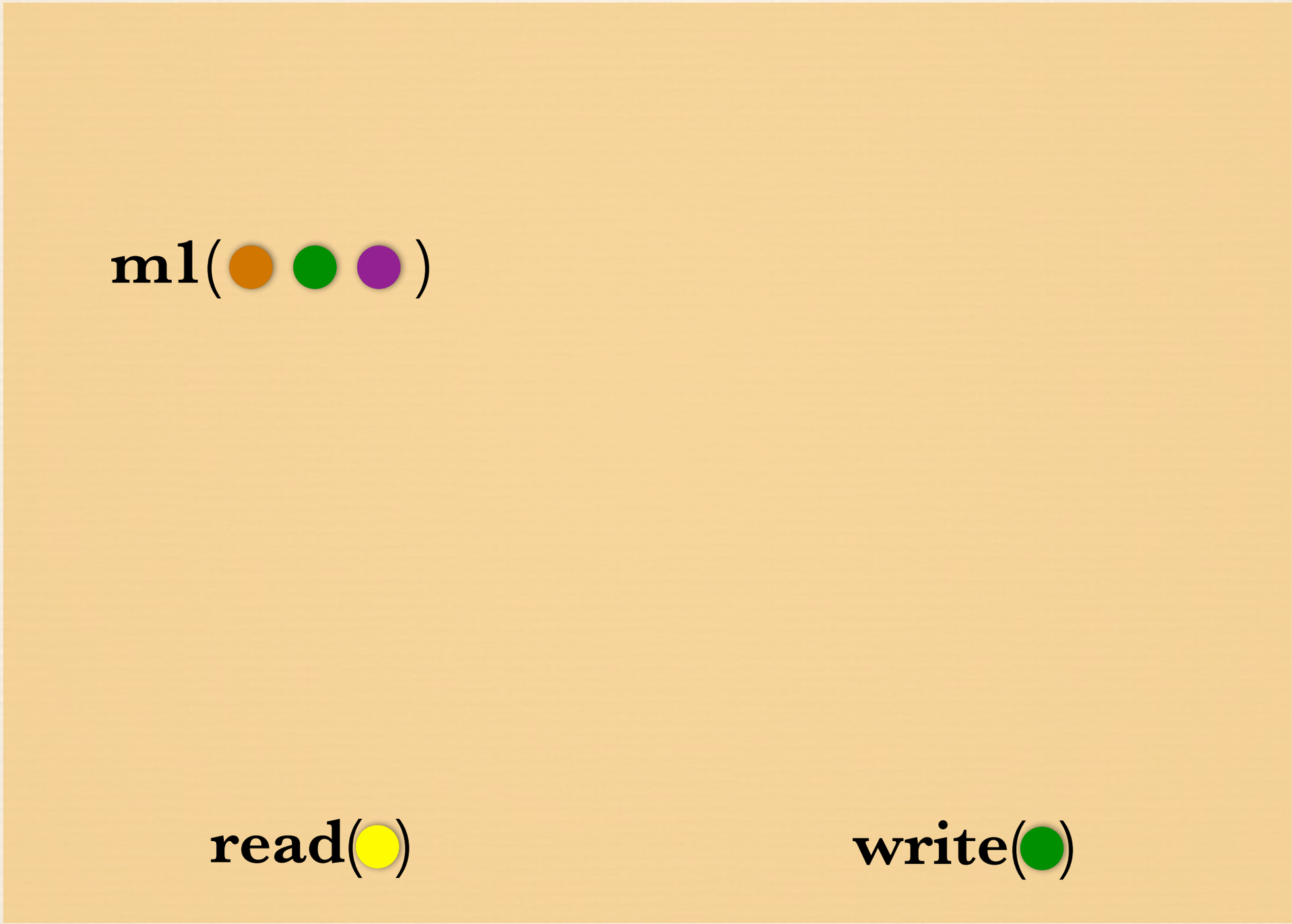
Challenge 1: Identify access pairs

T_1	**write(●)**	**read(●)**	…	**write(●)**
T_2	**read(●)**	**write(●)**	…	**read(●)**
T_3	**write(●)**	**read(●)**	…	**write(●)**
	…			
T_{n-1}	**write(●)**	**read(●)**	…	**write(●)**
T_n	**read(●)**	**read(●)**	…	**read(●)**

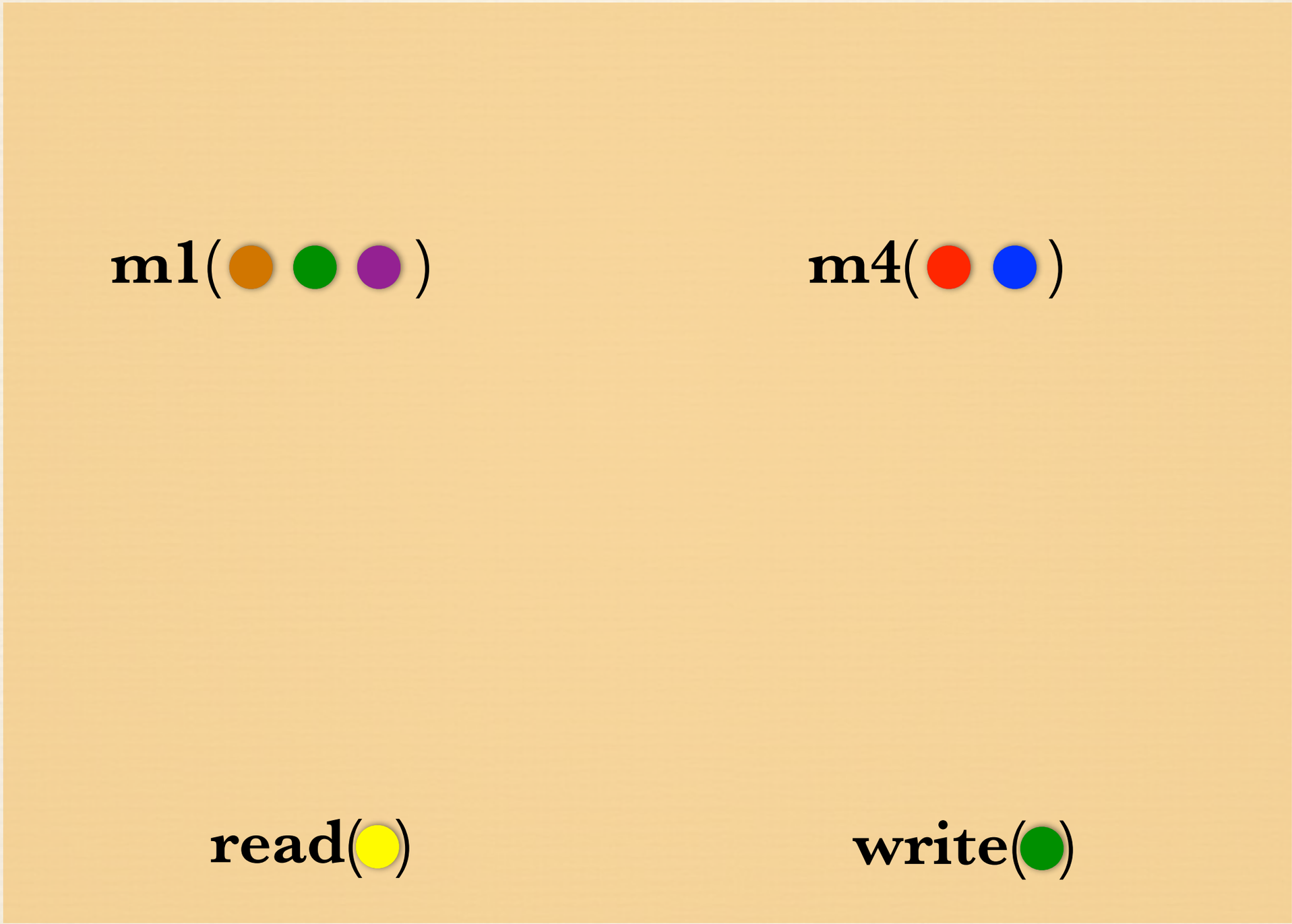
Challenge 2: Access reachability



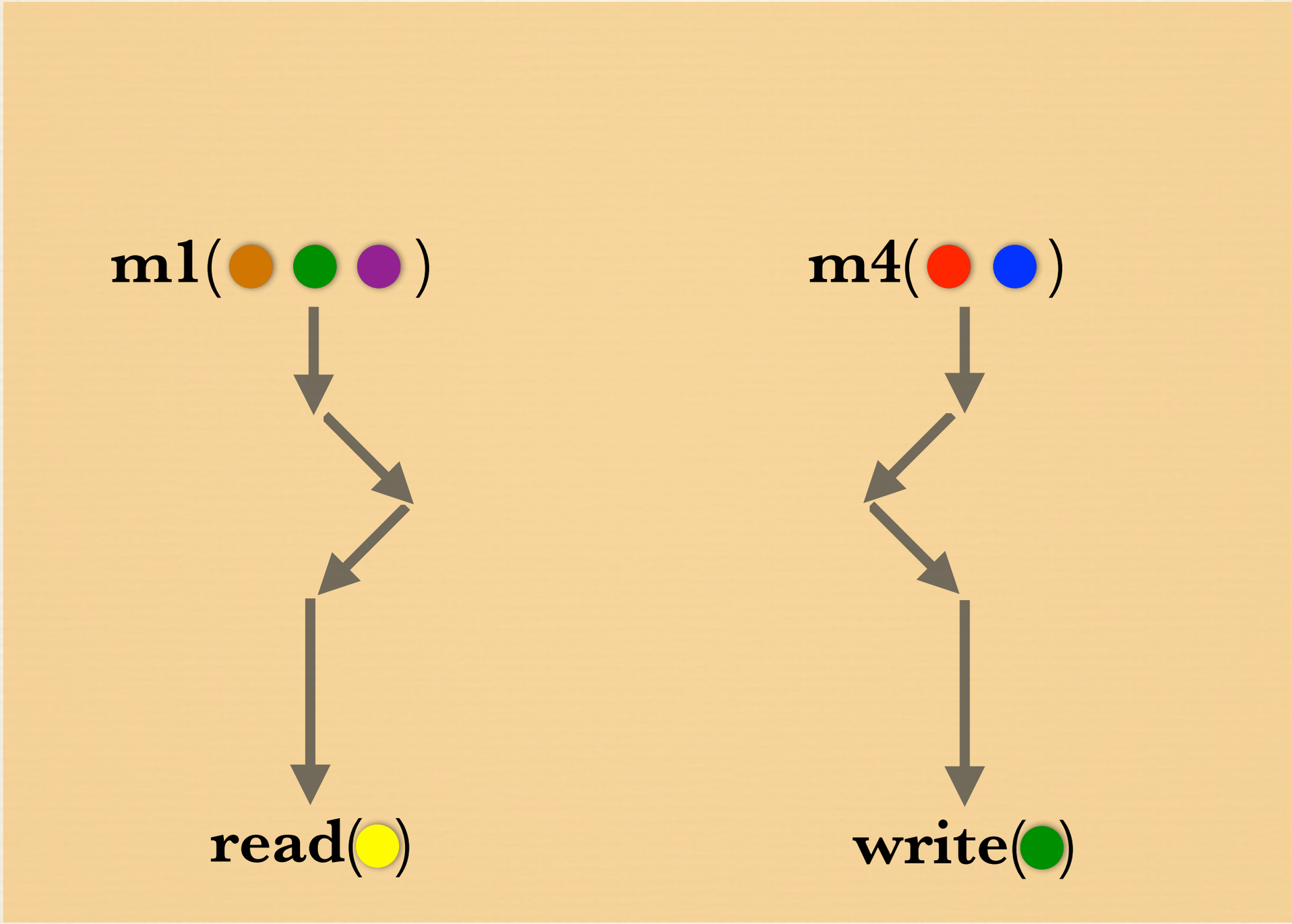
Challenge 2: Access reachability



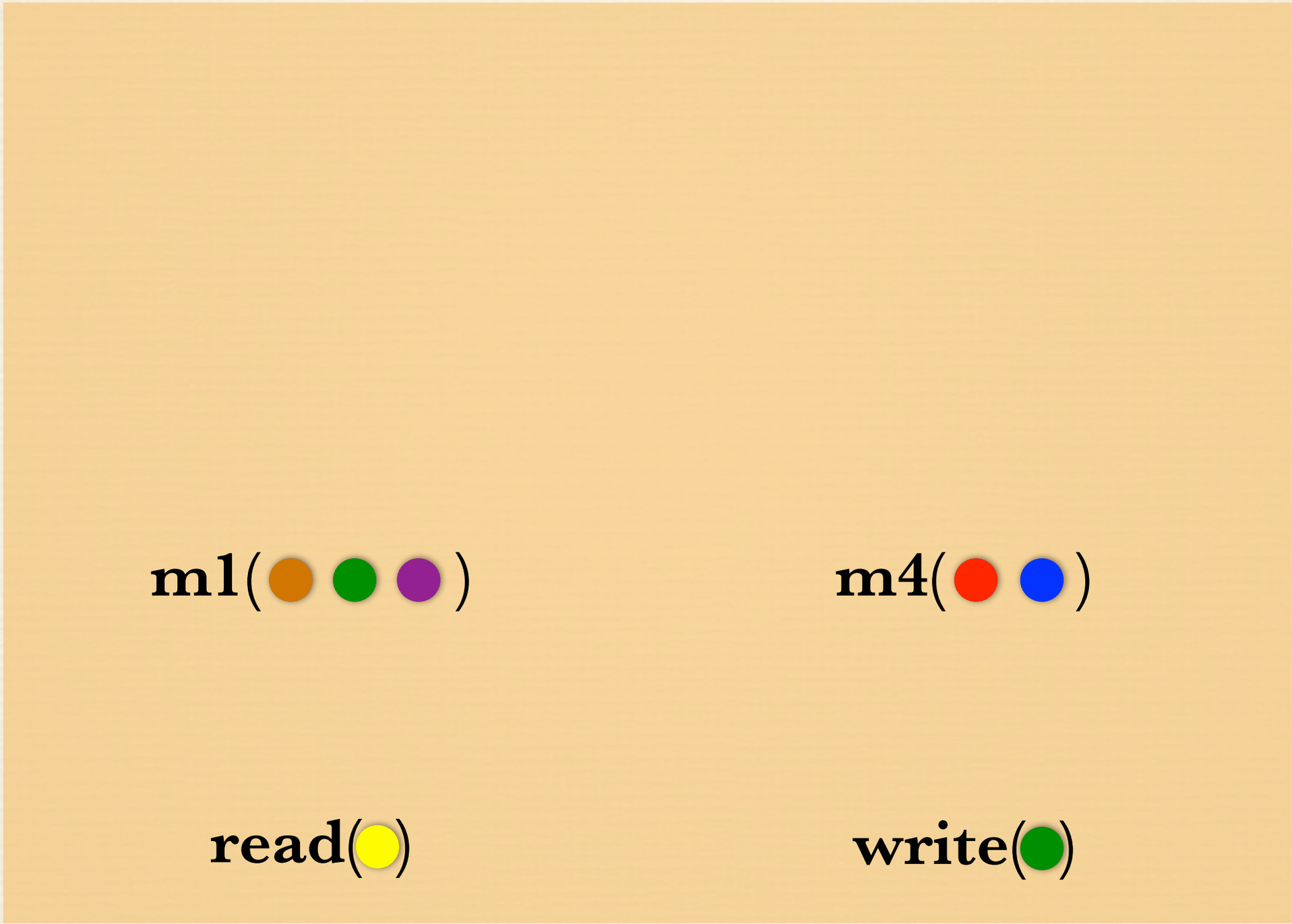
Challenge 2: Access reachability



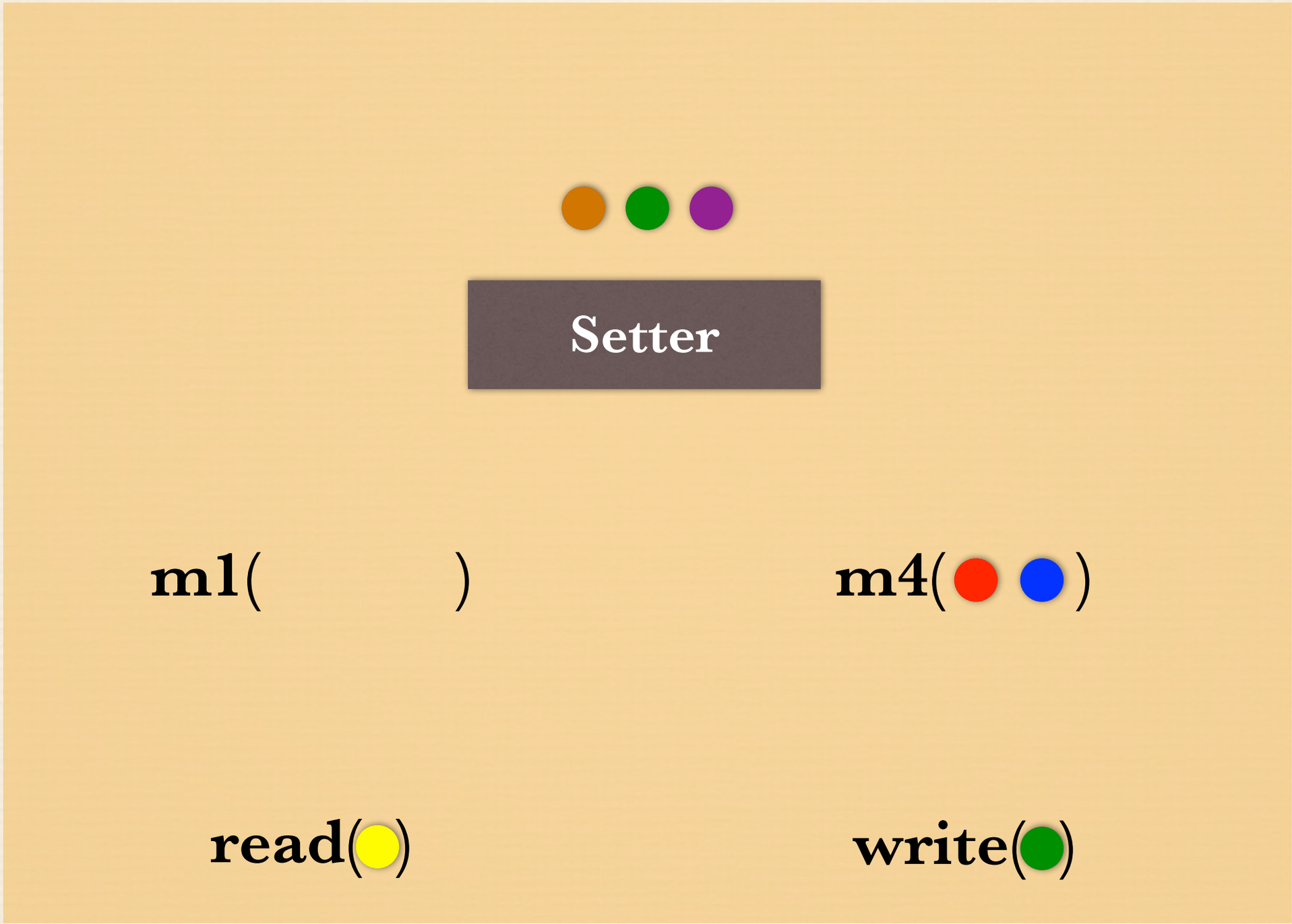
Challenge 2: Access reachability



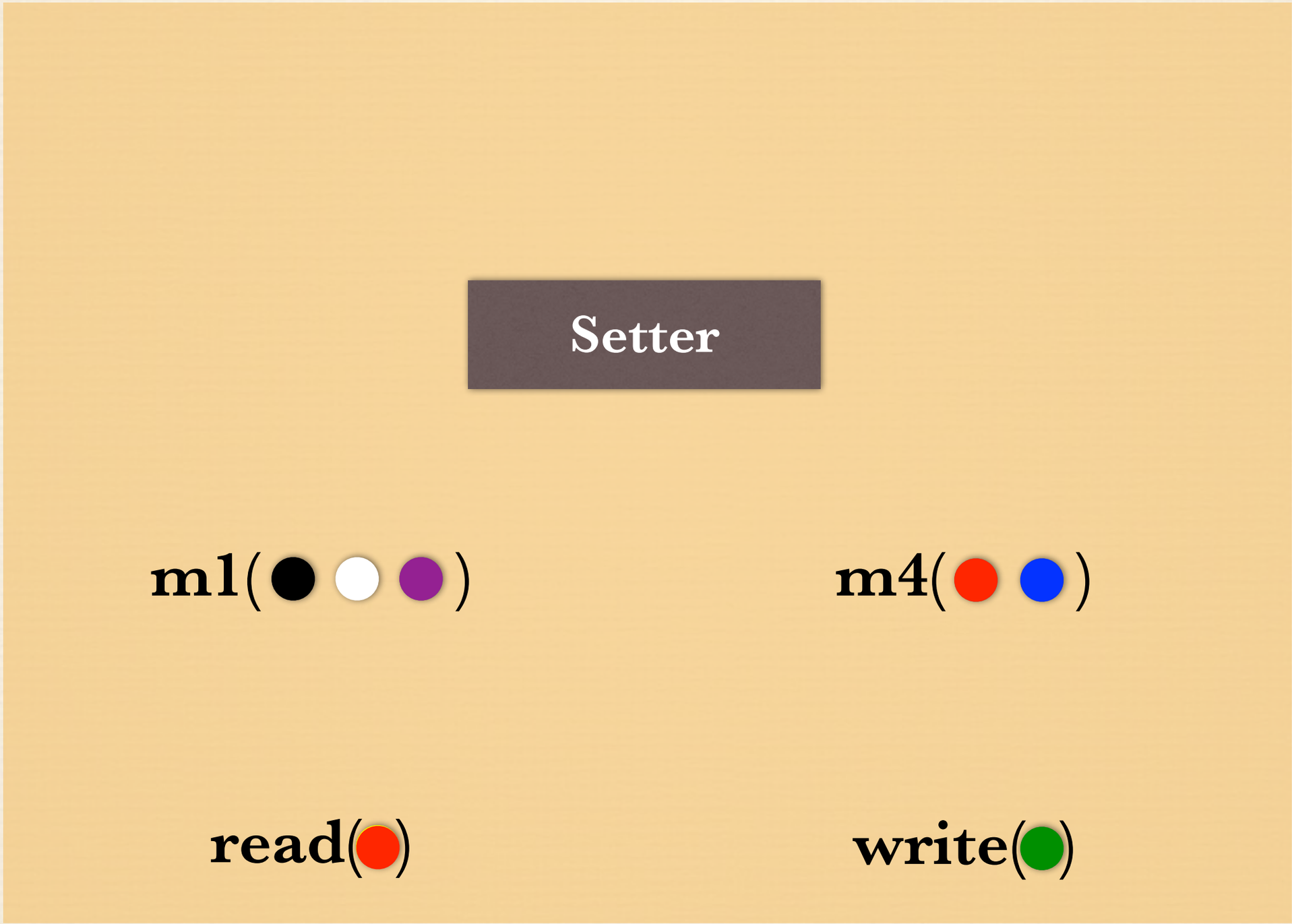
Challenge 3: Set the context



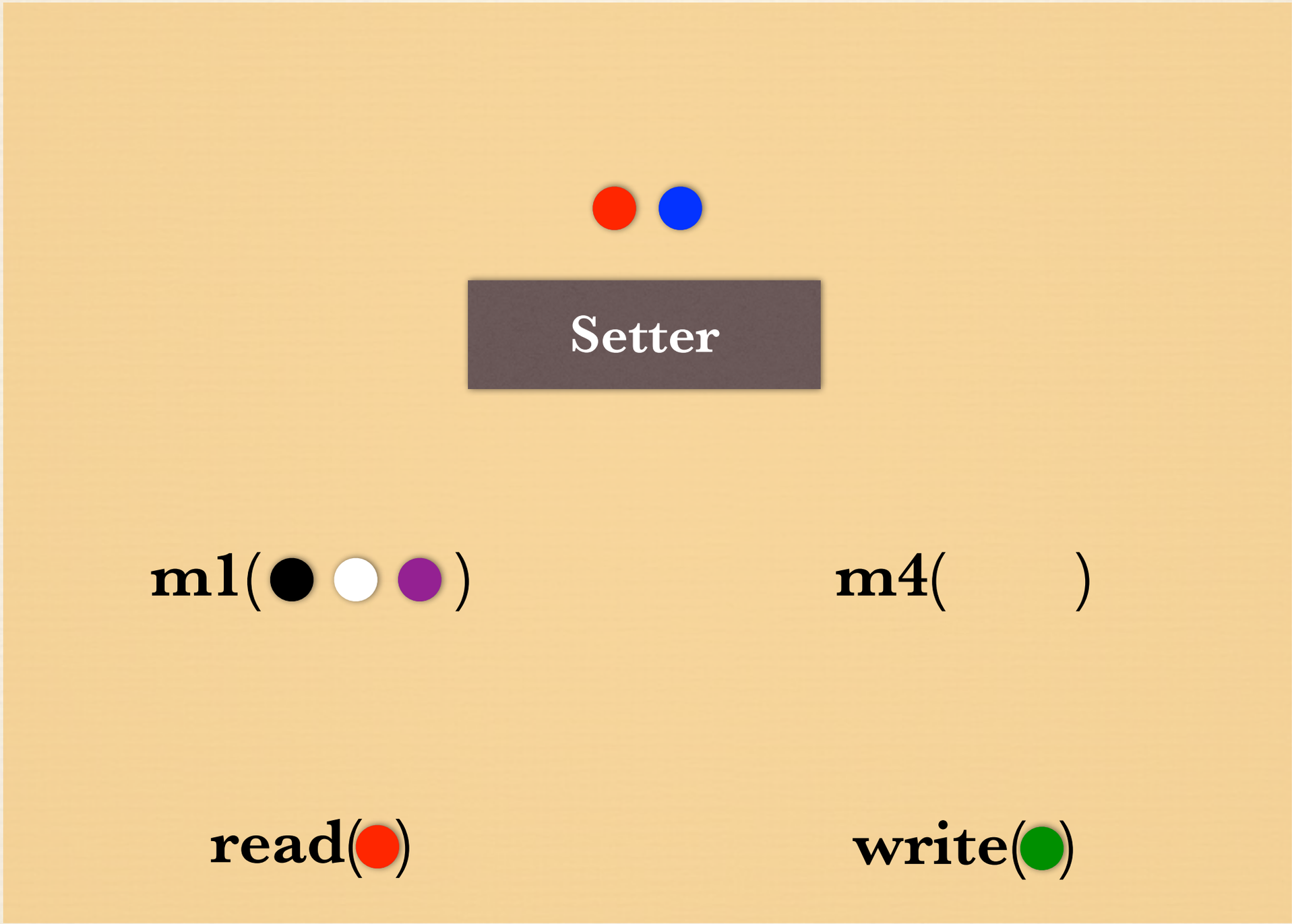
Challenge 3: Set the context



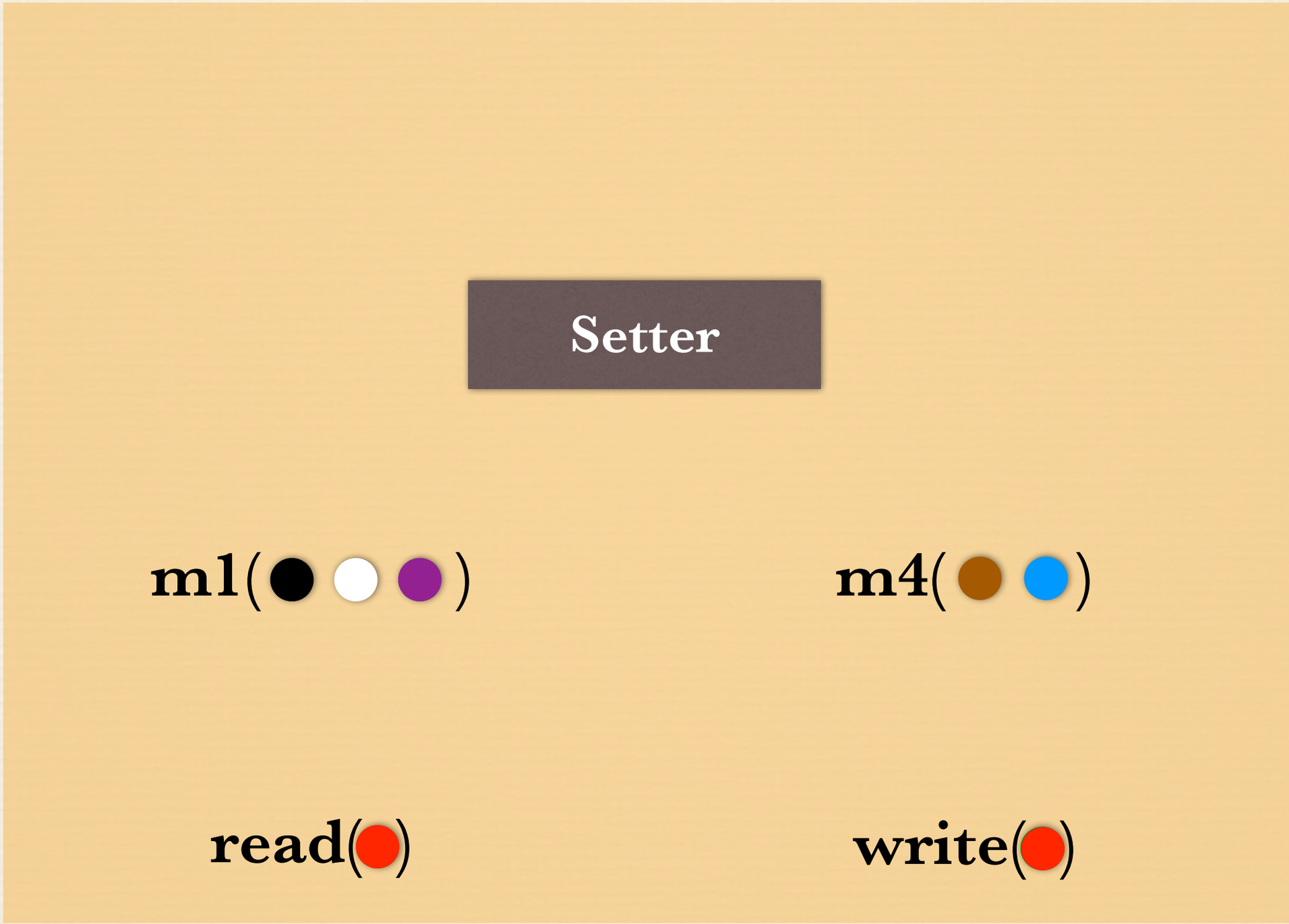
Challenge 3: Set the context



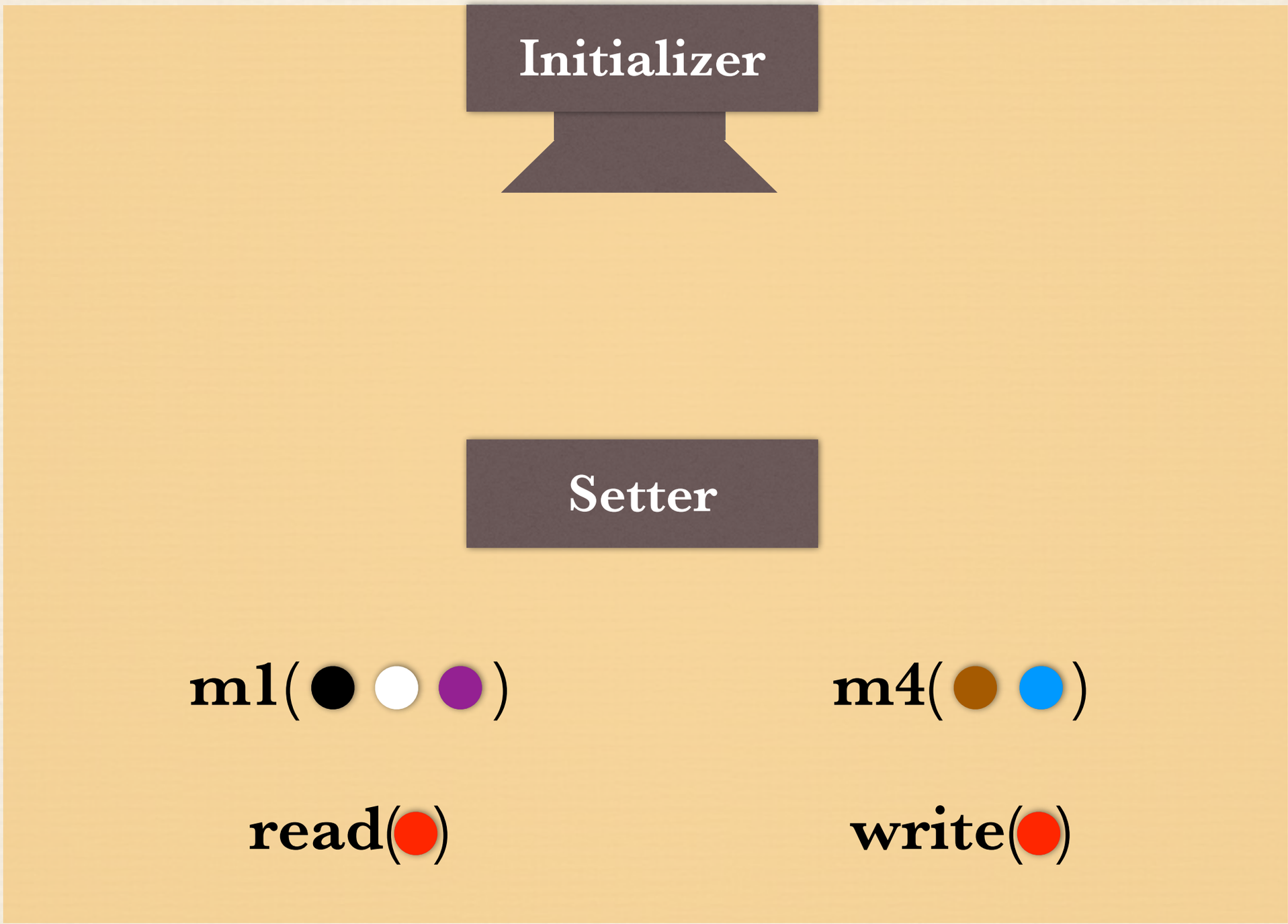
Challenge 3: Set the context



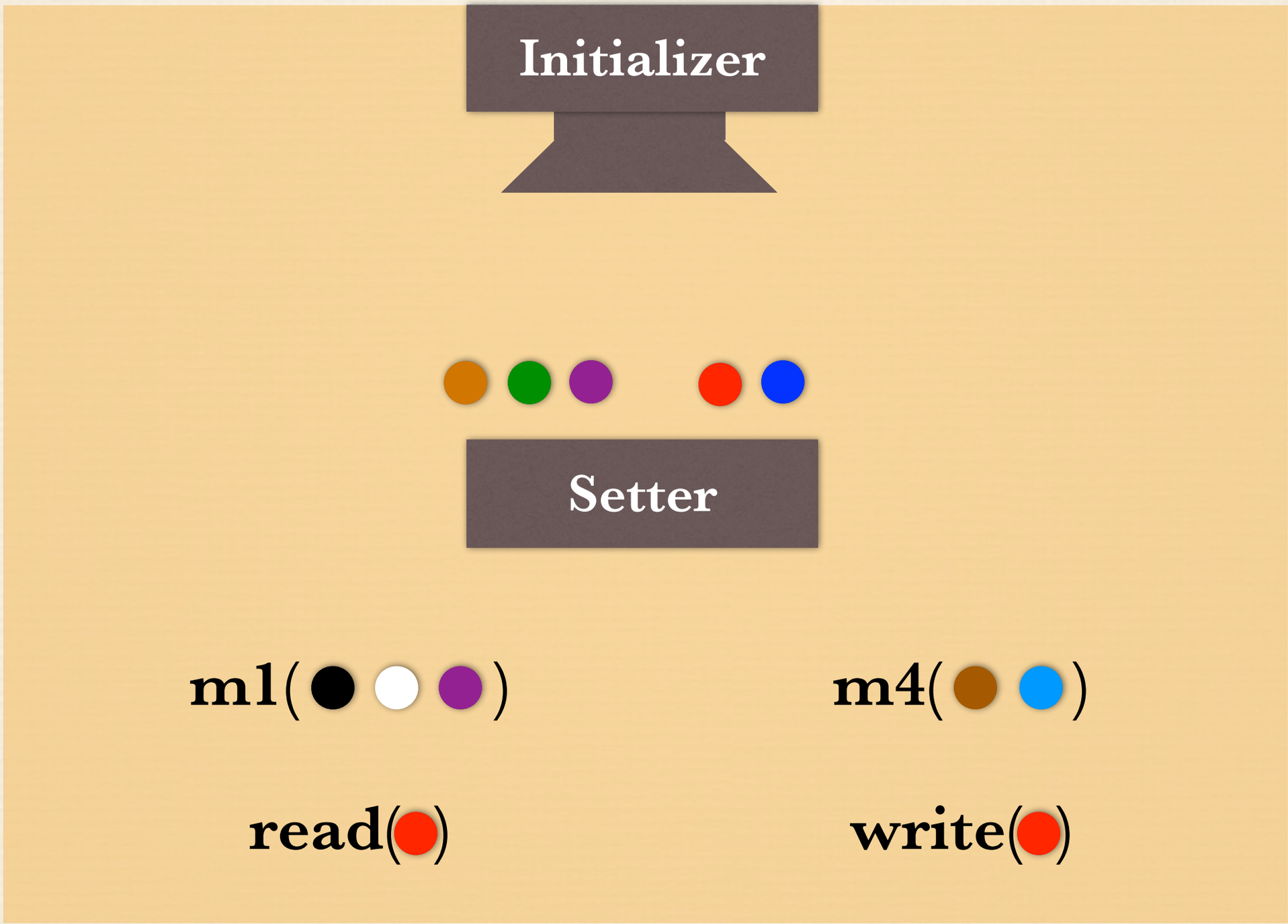
Challenge 3: Set the context



Challenge 4: Use legal object instances



Challenge 4: Use legal object instances



Summary of challenges

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

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

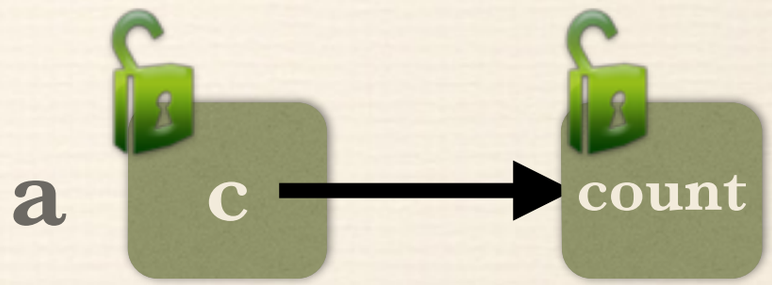
Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);



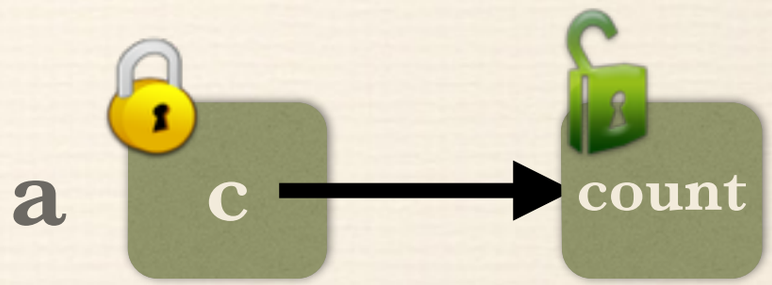
Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);



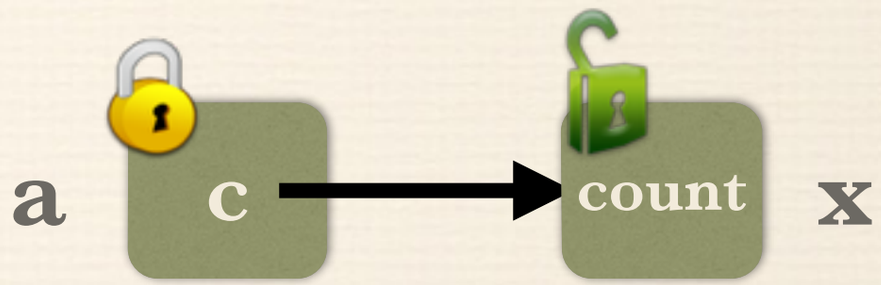
Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);



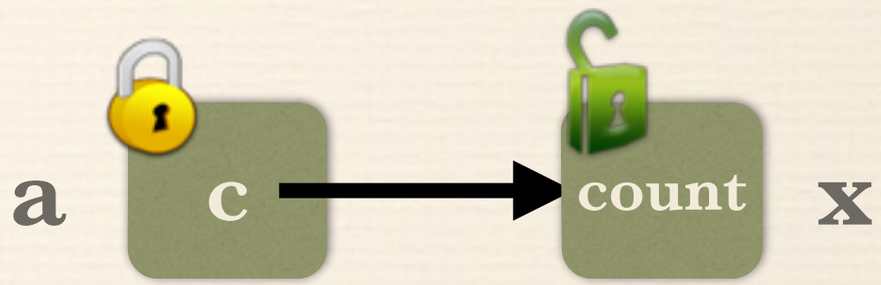
Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);



Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

Class : Lib Field : c

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

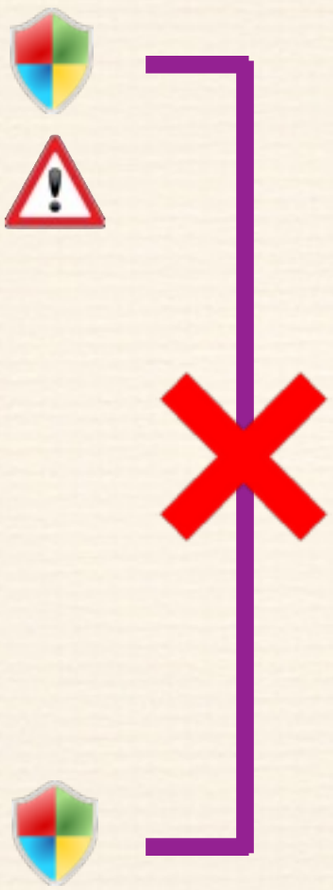
update (a_1);

set (a_1, val);

Class : Lib Field : c

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Input Sequential Test

update (a_1);

set (a_1, val);

Class : Lib Field : c

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

Class : Counter Field : count

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

Class : Counter Field : count

Identify access pairs

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Input Sequential Test

update (a_1);

set (a_1, val);

Class : Counter Field : count

Pairs :

1. **read 13 and write 13**

2. **write 13 and write 13**

Identify the APIs to be invoked concurrently

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Class : Counter **Field :** count

Pairs :

1. read 13 and write 13

2. write 13 and write 13

Identify the APIs to be invoked concurrently

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Class : Counter **Field :** count

Pairs :

1. read 13 and write 13

2. write 13 and write 13

→ **update & update**

Set the context

- Derive a sequence of method invocations
- Identify the parameter sharing between invocations

Set the context

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Set the context

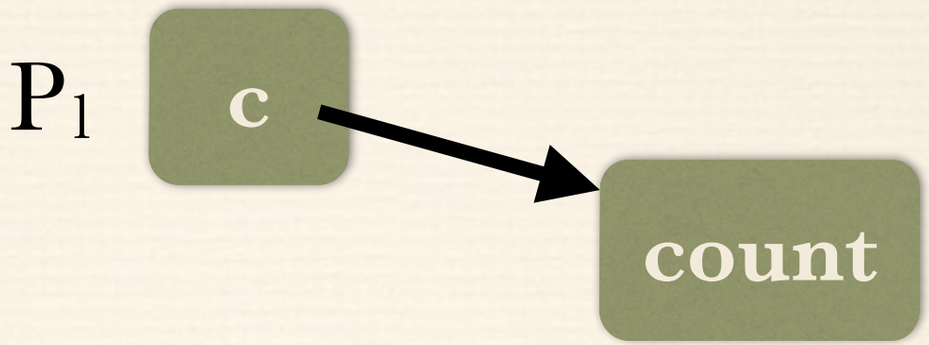
P_1.c.count to update == P_2.c.count to update

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Set the context

P_1.c.count to update == P_2.c.count to update

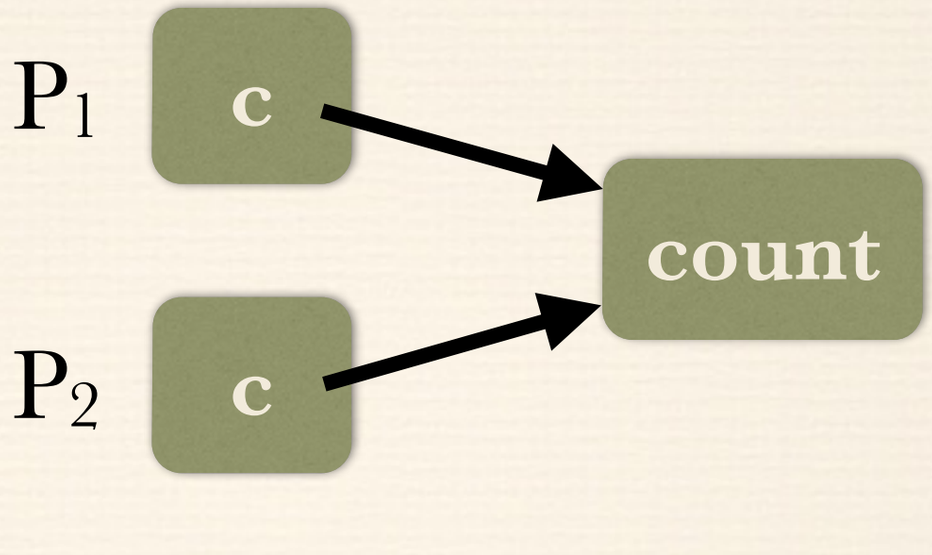
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Set the context

P_1.c.count to update == P_2.c.count to update

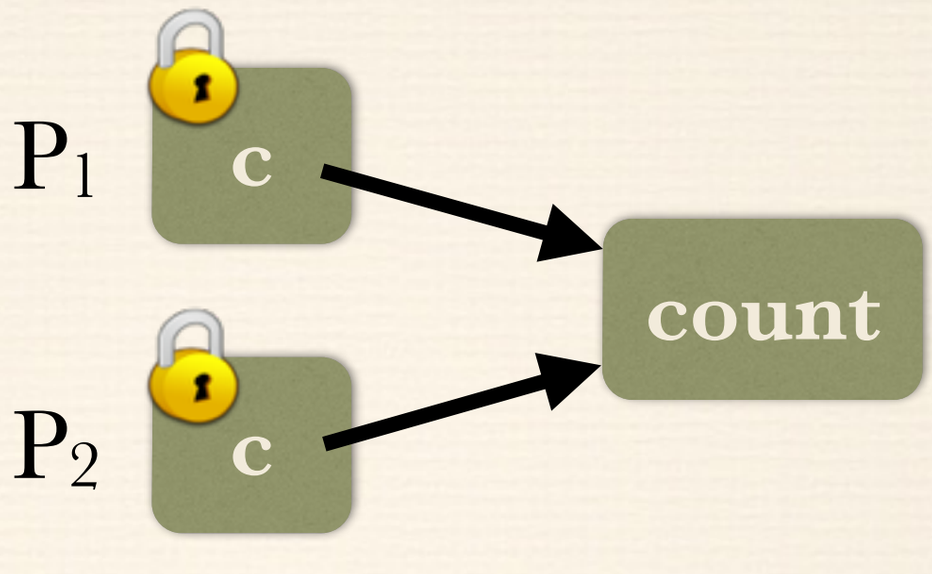
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Set the context

P_1.c.count to update == P_2.c.count to update

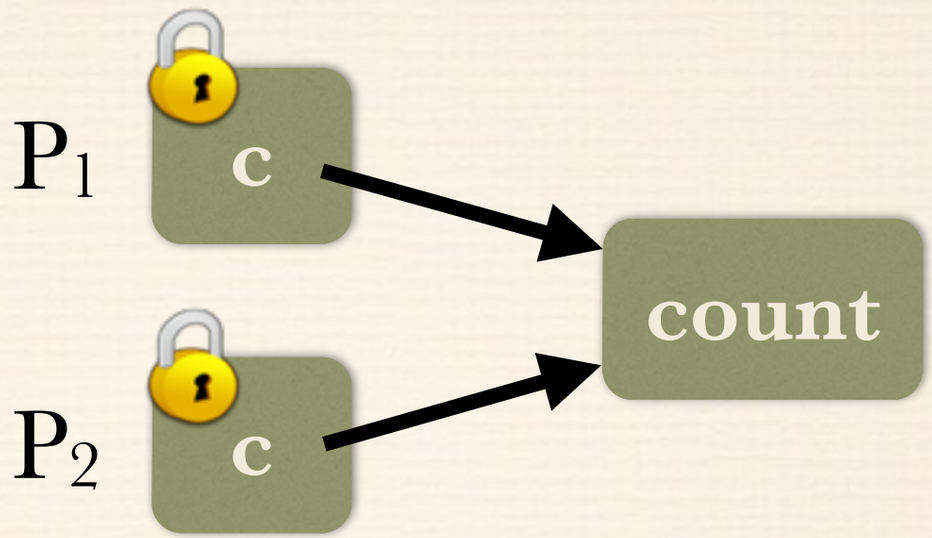
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Set the context

P_1.c.count to update == P_2.c.count to update

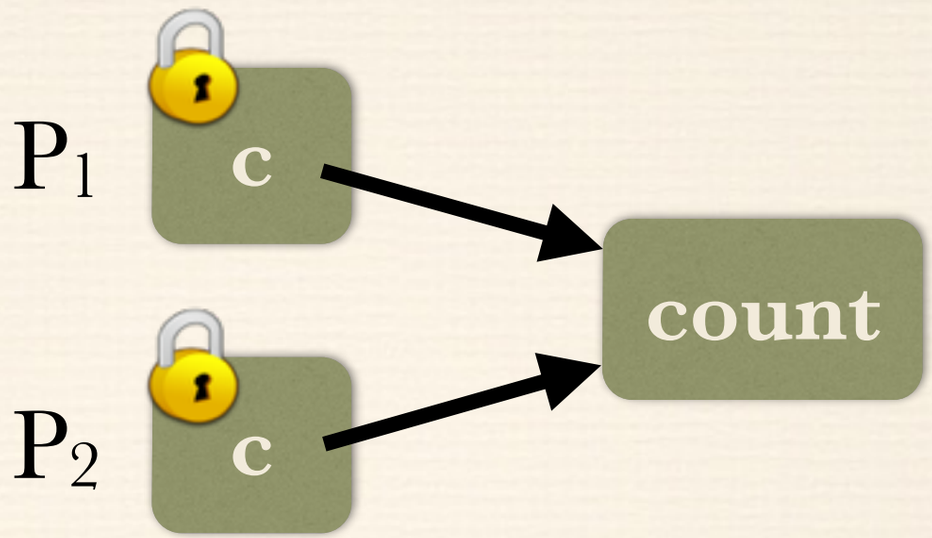
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Set the context

P_1.c.count to update == P_2.c.count to update

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

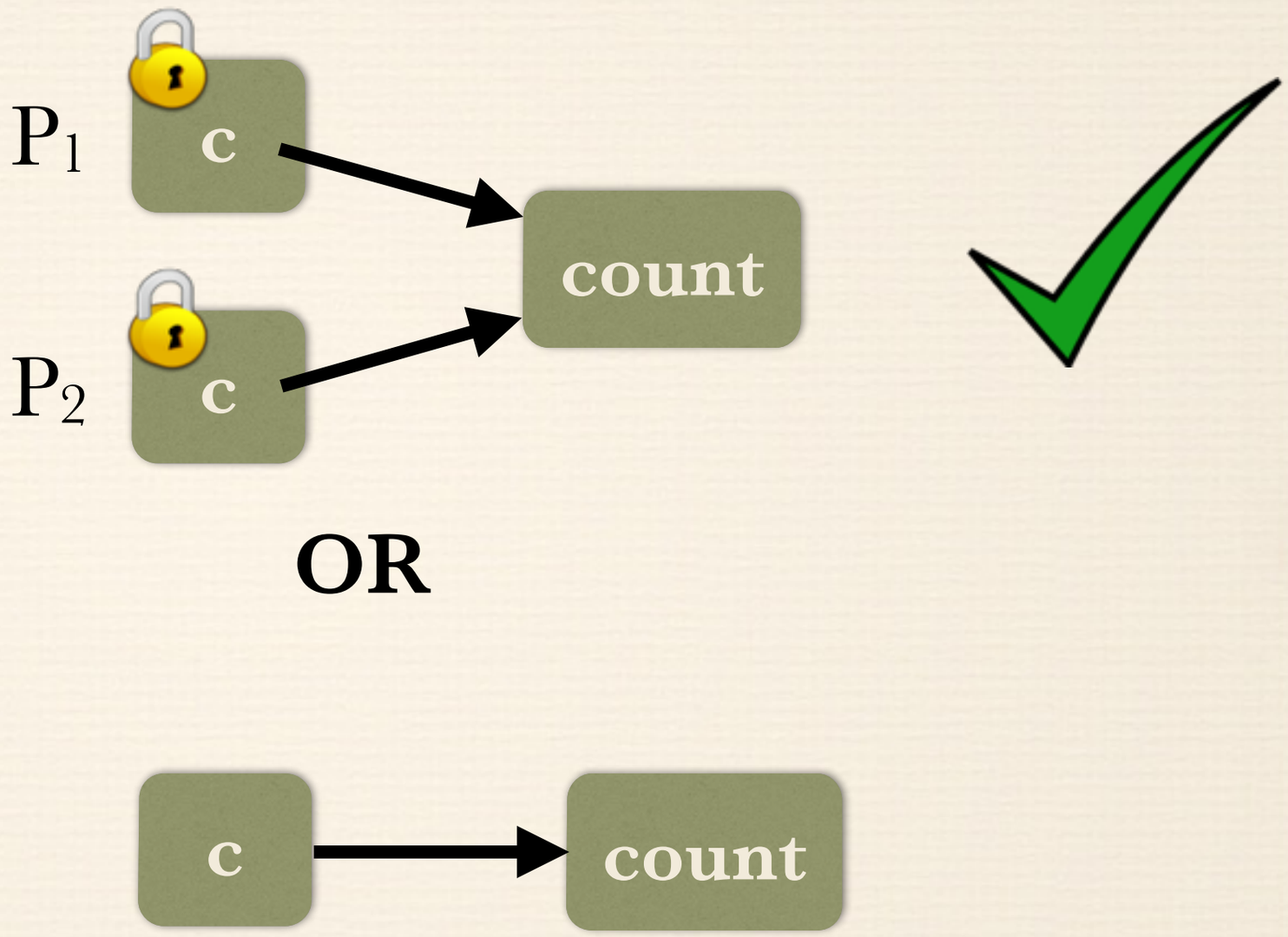


OR

Set the context

P_1.c.count to update == P_2.c.count to update

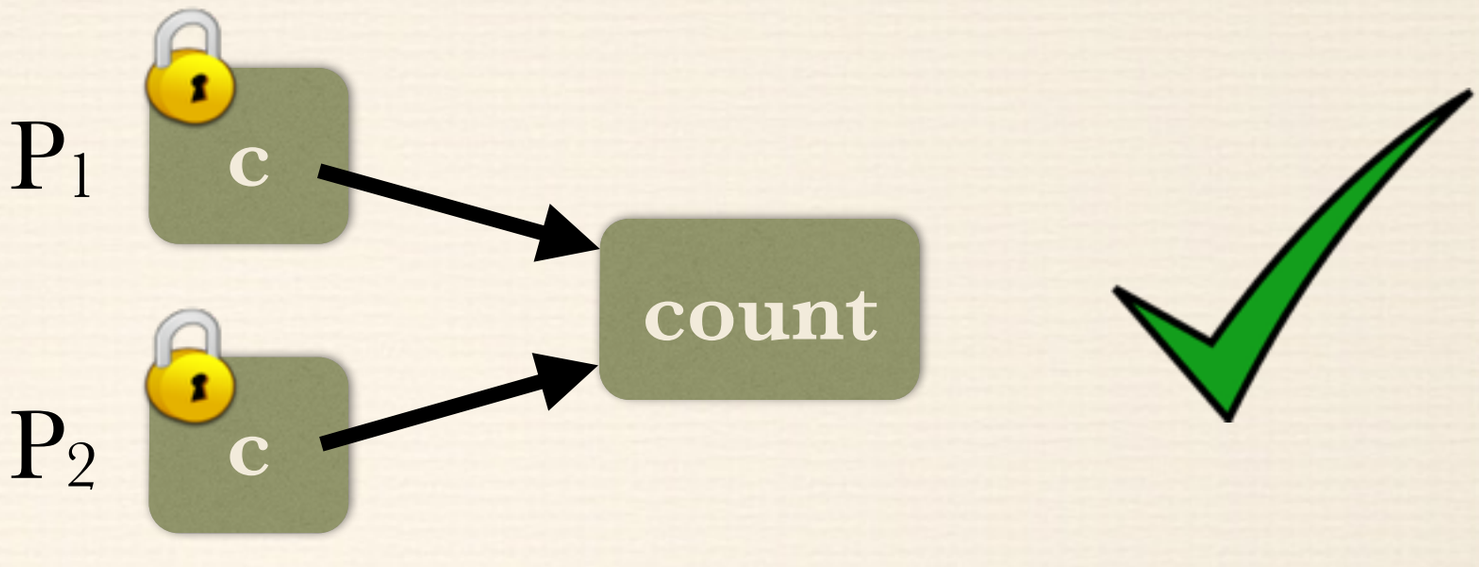
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



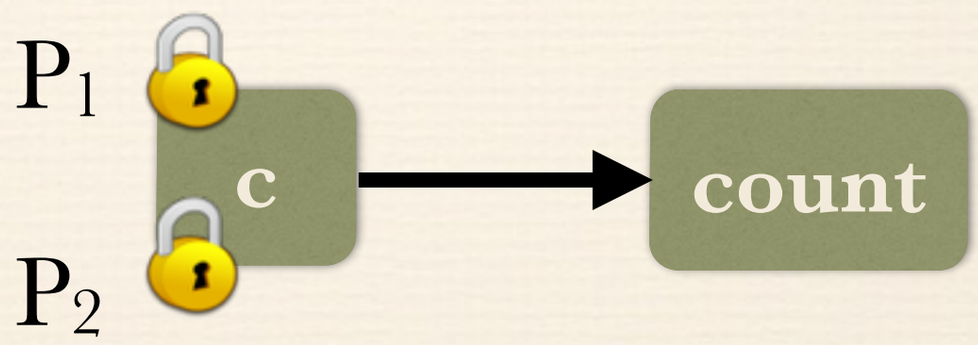
Set the context

P_1.c.count to update == P_2.c.count to update

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

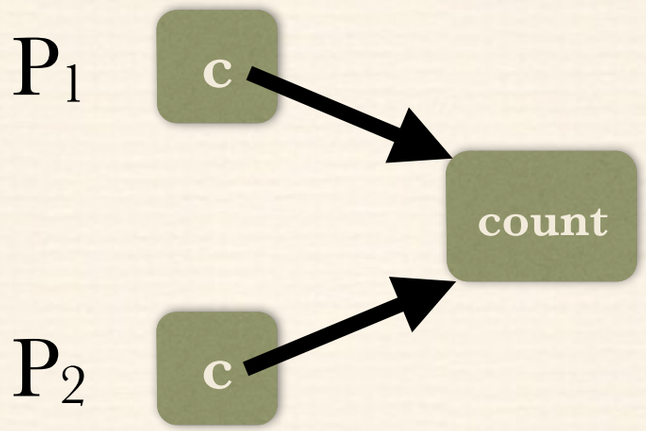


OR



Set the context

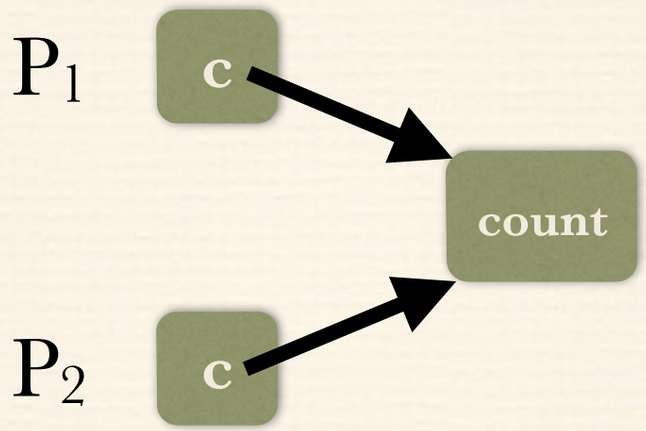
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Desired object state

Set the context

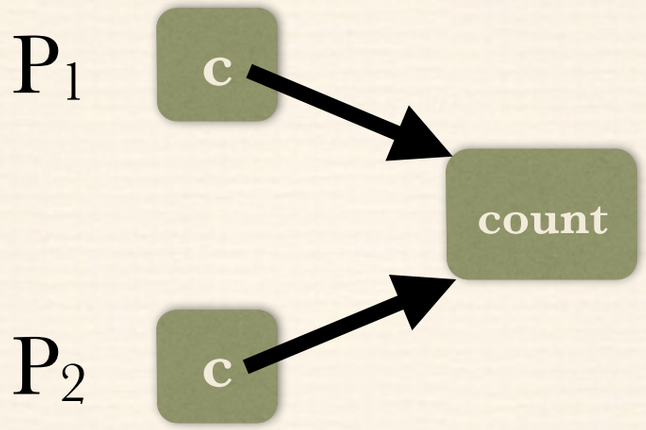
```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



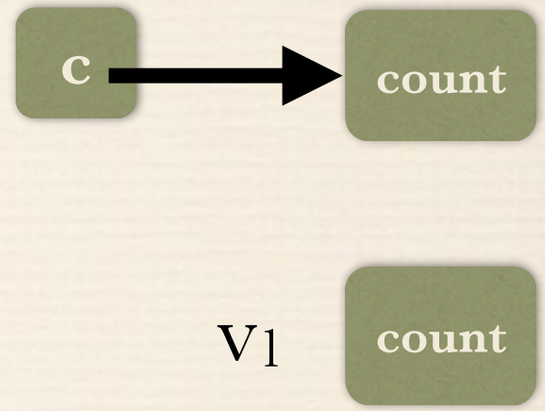
Desired object state

Set the context

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

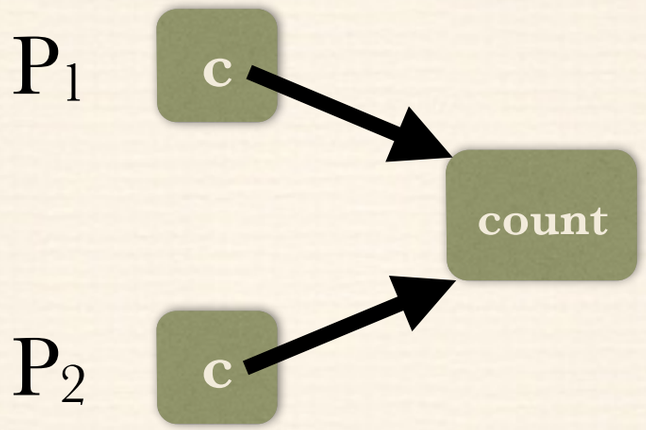


Desired object state

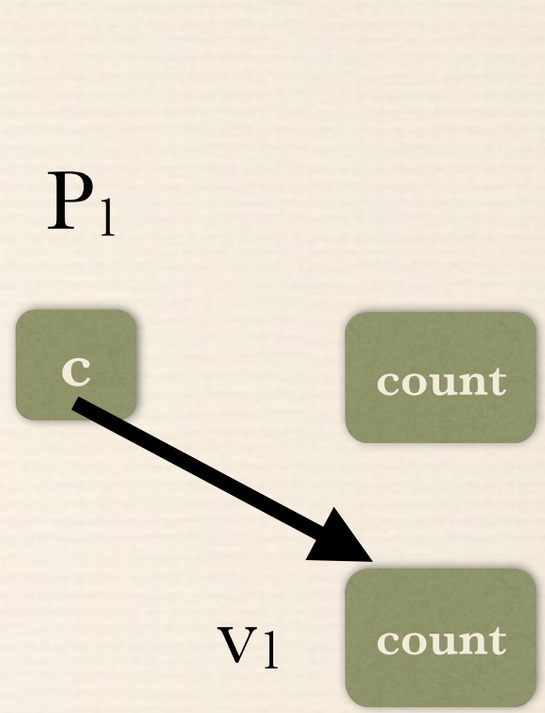


Set the context

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

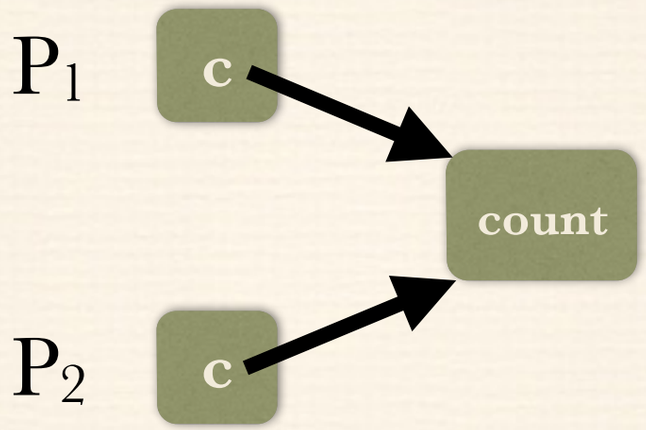


Desired object state

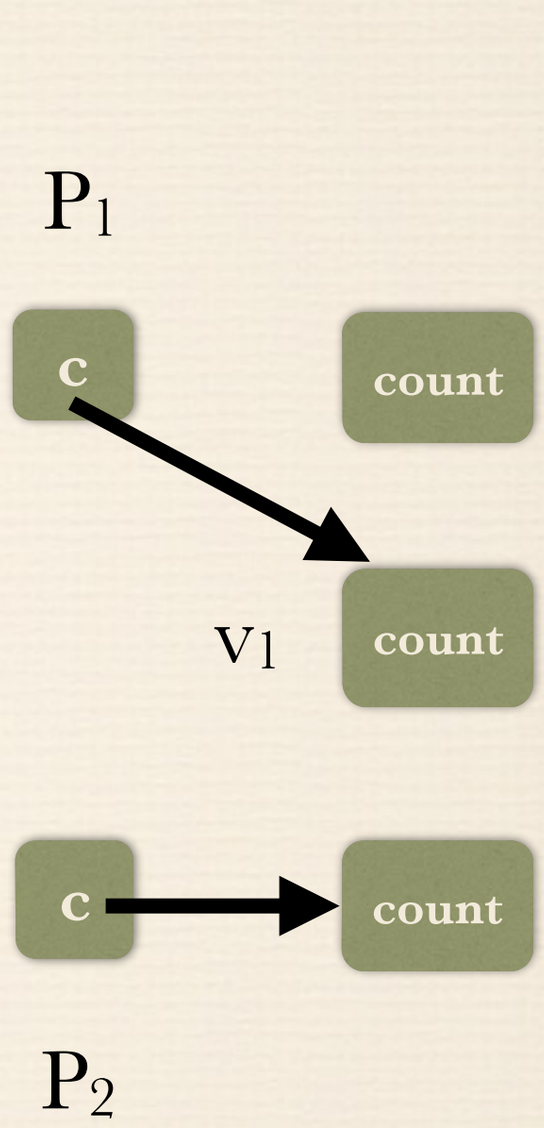


Set the context

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

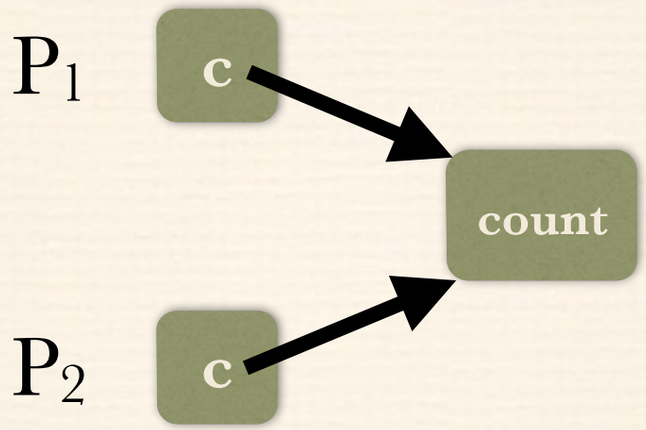


Desired object state

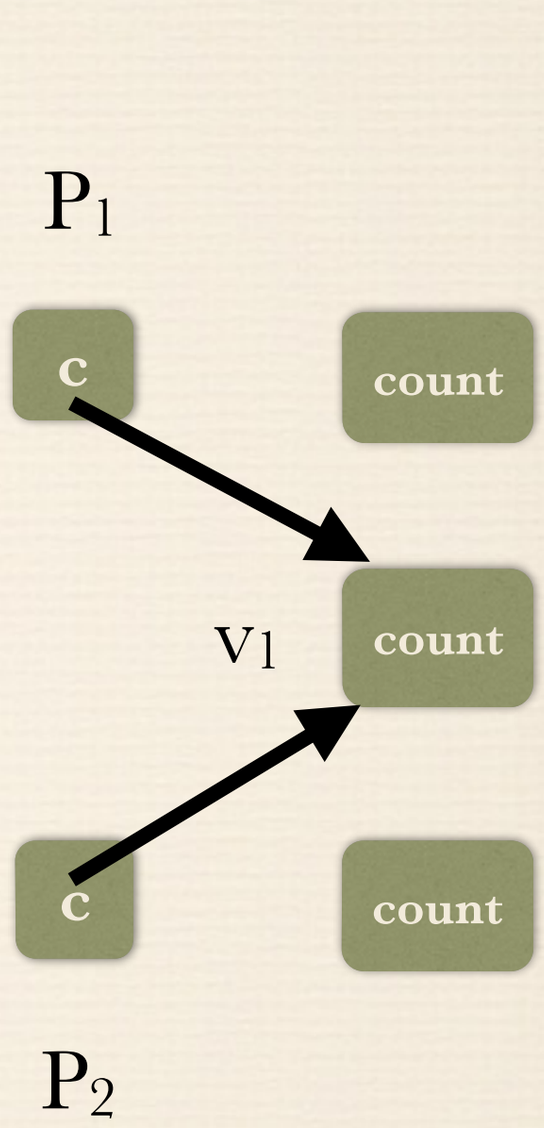


Set the context

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

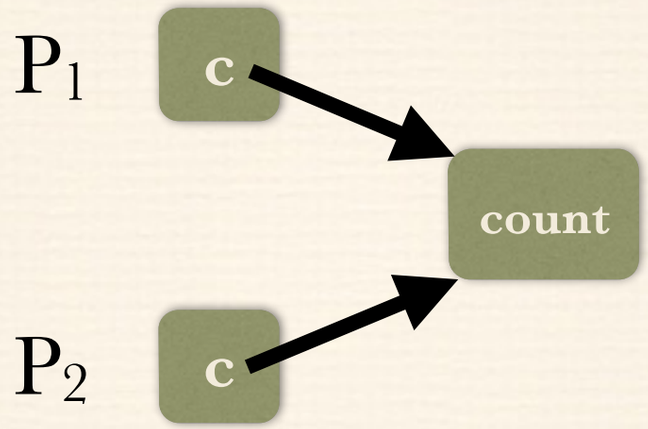


Desired object state

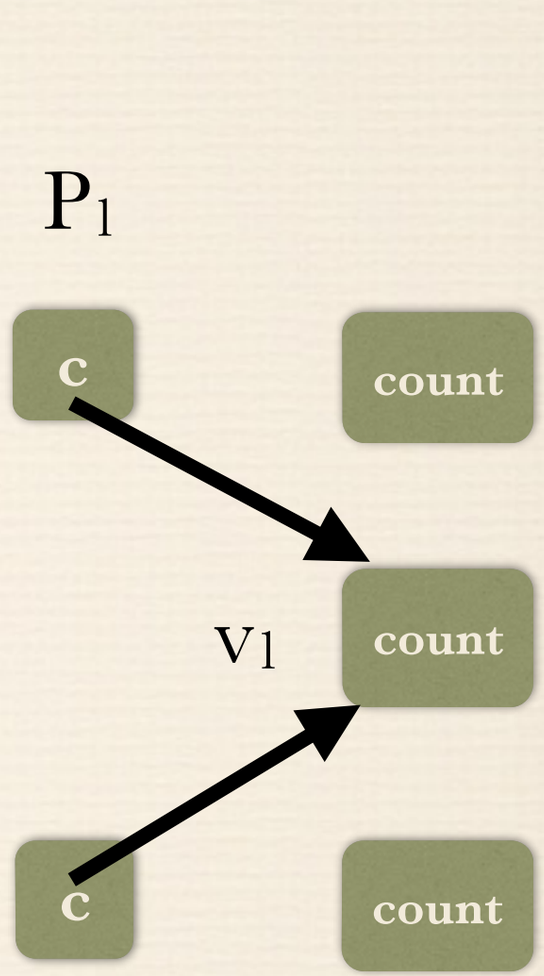


Set the context

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```



Desired object state



set(P_1, v_1)
set(P_2, v_1)

Use legal object instances

Execute sequential tests for obtaining object instances

Use legal object instances

Execute sequential tests for obtaining object instances

update (a_1);	update (a_1);	**update (a_1);**	**update (a_1);**
set (a_1, val);	**set (a_1, val);**	set (a_1, val);	set (a_1, val);

set :

set :

update : **update :**

Use legal object instances

Execute sequential tests for obtaining object instances

update (a_1); **set (a_1, *val*);**	update (a_1); **set (a_1, *val*);**	**update (a_1);** set (a_1, *val*);	**update (a_1);** set (a_1, *val*);

set : (o_1, o_2)

set :

update : **update :**

Use legal object instances

Execute sequential tests for obtaining object instances

update (a_1); **set (a_1, *val*);**	update (a_1); **set (a_1, *val*);**	**update (a_1);** set (a_1, *val*);	**update (a_1);** set (a_1, *val*);

set : (o_1, o_2)

set : (o_3, o_4)

update : **update :**

Use legal object instances

Execute sequential tests for obtaining object instances

update (a_1); **set (a_1, *val*);**	update (a_1); **set (a_1, *val*);**	**update (a_1);** set (a_1, *val*);	**update (a_1);** set (a_1, *val*);

set : (o_1, o_2)

set : (o_3, o_4)

update : (o_5) **update : (o_6)**

Generated Test

```
update (Lib a) {
    11:  lock(a)
    12: Counter x = a.c
    13:  x.count = x.count +1
    14:  unlock(a)
}

set (Lib a, Counter b) {
    15:  lock(a)
    16:    a.c = b
    17:  unlock(a)
}
```

Initializer ( )

set (o_5, o_2)

set (o_6, o_4)

t_1:	t_2:
update (o_5)	**update** (o_6)

Experimental Setup

Experimental Setup

- Implemented using the SOOT bytecode analysis framework

Experimental Setup

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

Experimental Setup

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

Experimental Setup

- Implemented using the SOOT bytecode analysis framework
- Evaluated on open source Java libraries
- Invoked each method in a class once with random objects
- Used RaceFuzzer (Sen, PLDI'08) to detect races

Experimental Validation

Experimental Validation

- Synthesized **101 racy tests**, including thread-safe classes

Experimental Validation

- Synthesized **101 racy tests**, including thread-safe classes
  - Randomized Test Generation (PLDI'12) generated 105 - 70K tests. Detected two defects

Experimental Validation

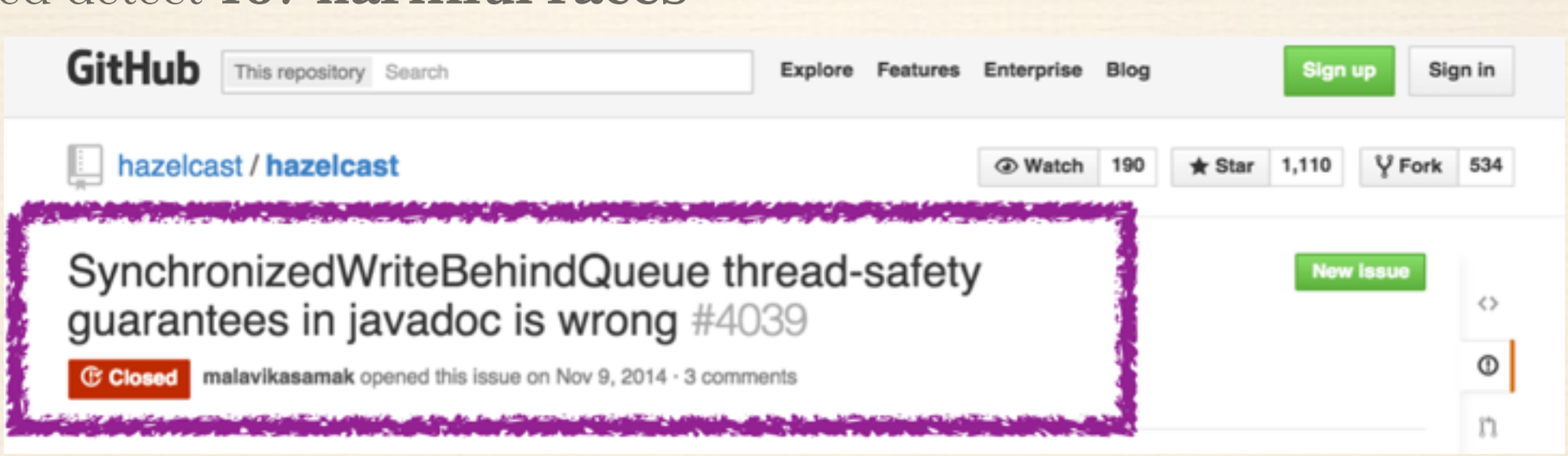
- Synthesized **101 racy tests**, including thread-safe classes
  - Randomized Test Generation (PLDI'12) generated 105 - 70K tests. Detected two defects
- Negligible time overhead (< 25s per class) for synthesizing tests

Experimental Validation

- Synthesized **101 racy tests**, including thread-safe classes
  - Randomized Test Generation (PLDI'12) generated 105 - 70K tests. Detected two defects
- Negligible time overhead (< 25s per class) for synthesizing tests
- Helped detect **187 harmful races**

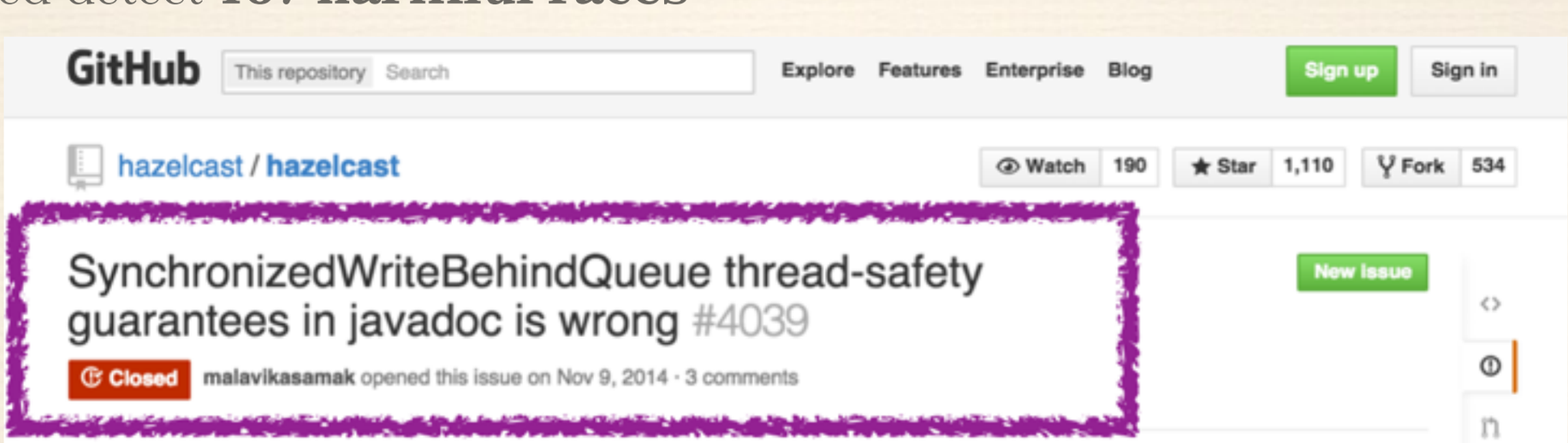
Experimental Validation

- Synthesized **101 racy tests**, including thread-safe classes
  - Randomized Test Generation (PLDI'12) generated 105 - 70K tests. Detected two defects
- Negligible time overhead (< 25s per class) for synthesizing tests
- Helped detect **187 harmful races**

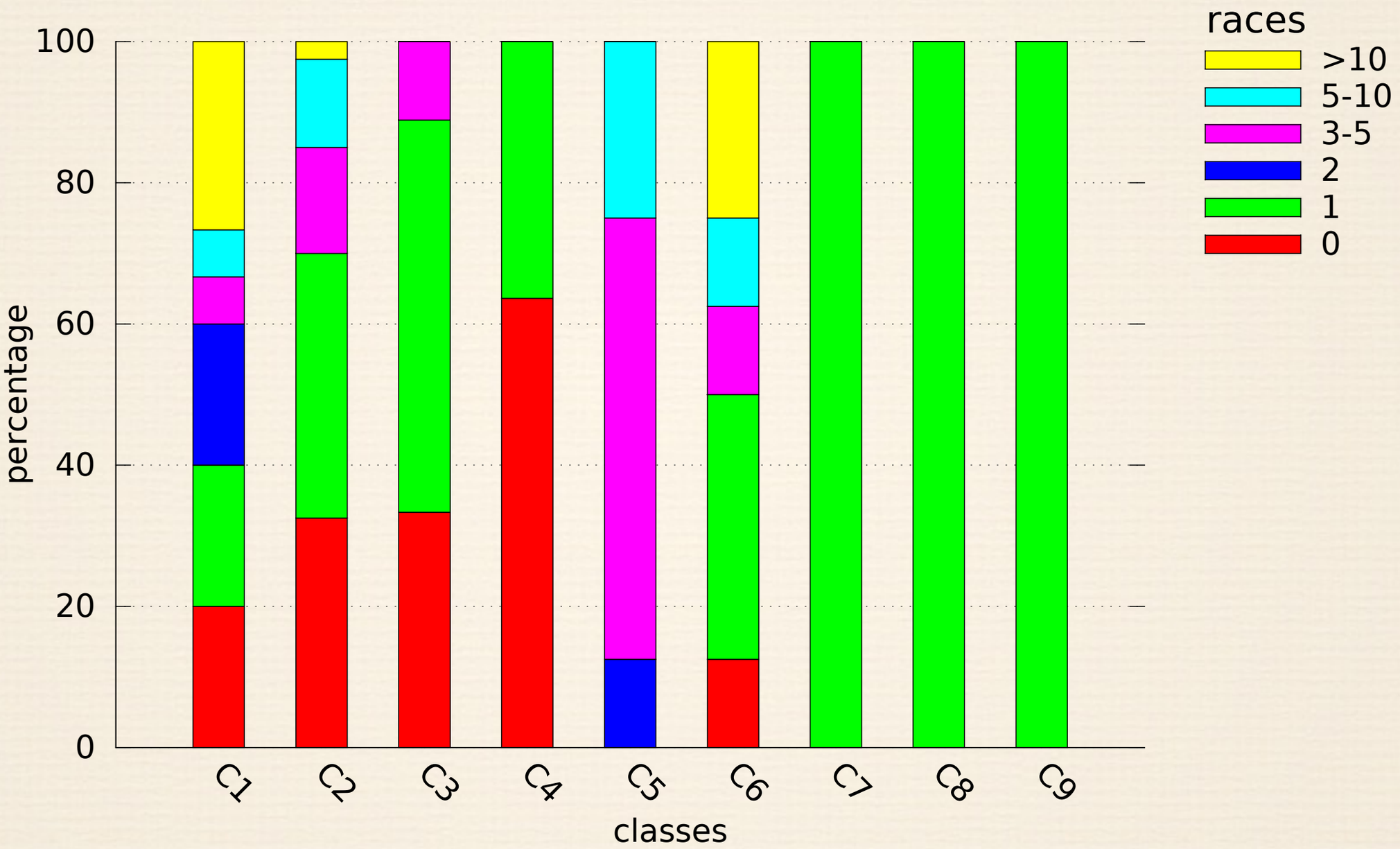


Experimental Validation

- Synthesized **101 racy tests**, including thread-safe classes
  - Randomized Test Generation (PLDI'12) generated 105 - 70K tests. Detected two defects
- Negligible time overhead (< 25s per class) for synthesizing tests
- Helped detect **187 harmful races**



Distribution of Tests



Summary

- Designed a ***directed*** approach to synthesize racy tests
- NARADA: tool that incorporates the proposed design
- Validated using open source java libraries
- Helped detect harmful races
  - Negligible time overhead
- Tool: http://www.csa.iisc.ernet.in/~sss/tools/narada/