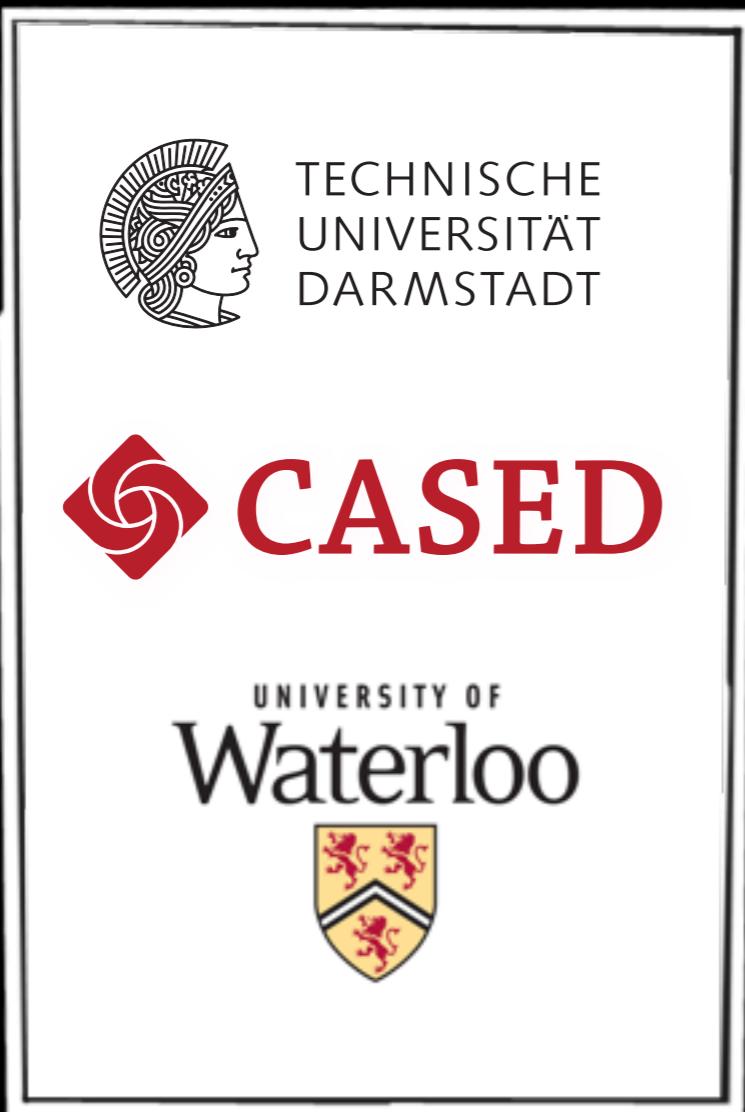


Clara: Partially Evaluating Runtime Monitors at Compile Time

Eric Bodden
Patrick Lam



Outline

- Why partial evaluation of runtime monitors?
- Runtime monitoring: existing tools and approaches
- Clara's design and architecture
- Clara's three pre-defined static analyses
- Using Clara to evaluate your monitoring aspects
- Implementing your own static analyses

Why partial
evaluation?

Integrate results of three communities

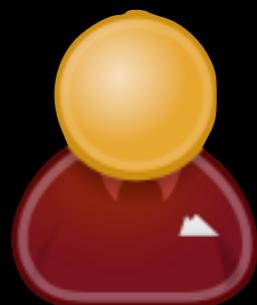
Integrate results of three communities

AOP /
AspectJ

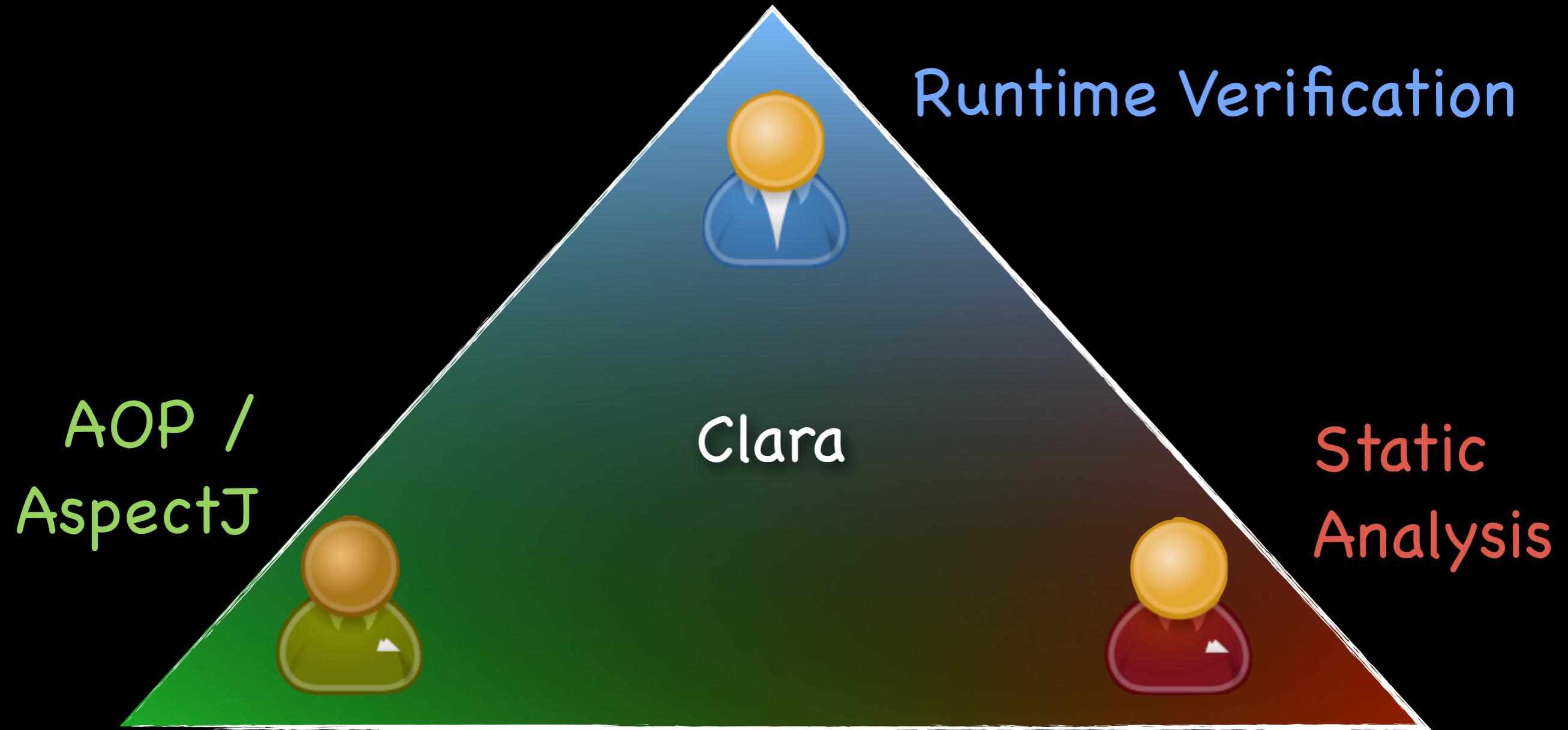


Runtime Verification

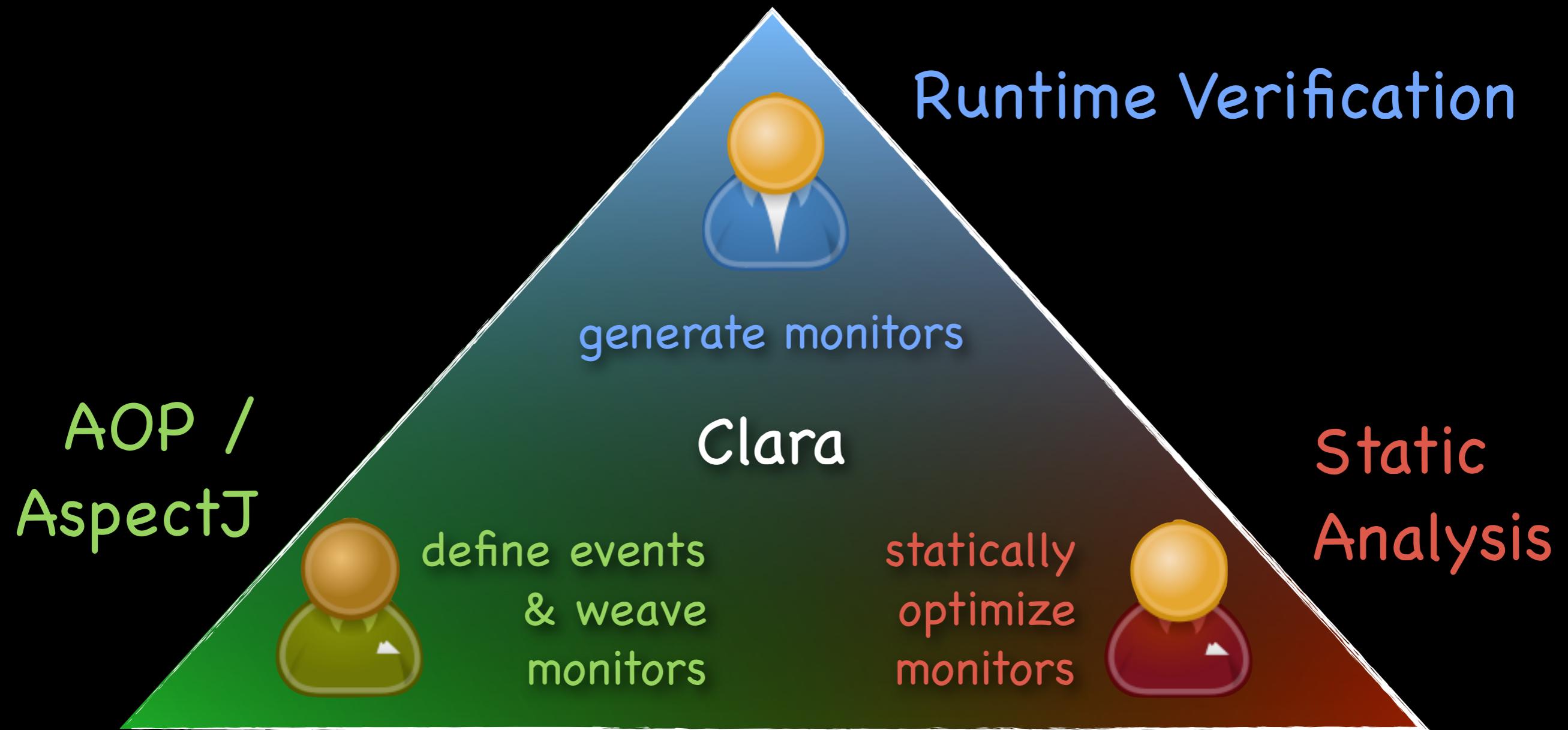
Static
Analysis



Integrate results of three communities



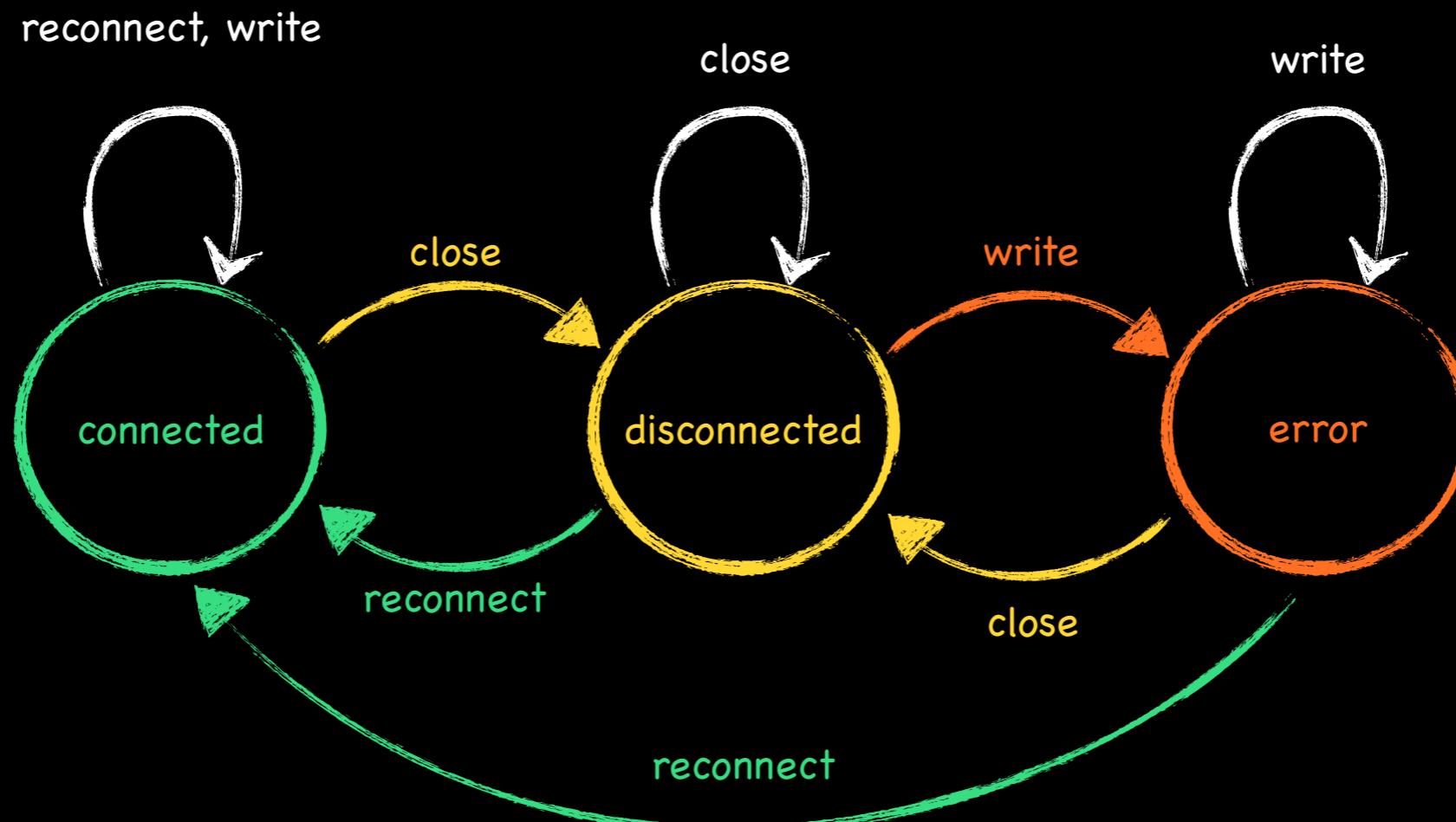
Integrate results of three communities



Finite-state properties

“After closing a connection c ,
don’t write to c until c is reconnected.”

Finite-state properties



"After closing a connection c,
don't write to c until c is reconnected."

ConnectionClosed Aspect

```
Set closed = new IdentityHashSet();
```

```
after(Connection c) returning:  
    call(* Connection.close()) && target(c) {  
        closed.add(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.reconnect()) && target(c) {  
        closed.remove(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.write(..)) && target(c) {  
        if(closed.contains(c))  
            error("May not write to "+c+", as it is closed!");  
    }
```

Aspects vs. manual monitors

- Tedious: have to add state variables to every implementation of Connection interface!
- Cluttering: error-checking code is all over the place!
- Hard to analyze: hard to see what the code actually checks
- Problematic: external hash maps/sets can lead to memory leaks

Aspects vs. manual monitors

- Tedious: have to add state variables to every implementation of Connection interface!
Aspects let you monitor interface types
- Cluttering: error-checking code is all over the place!
- Hard to analyze: hard to see what the code actually checks
- Problematic: external hash maps/sets can lead to memory leaks



Aspects vs. manual monitors

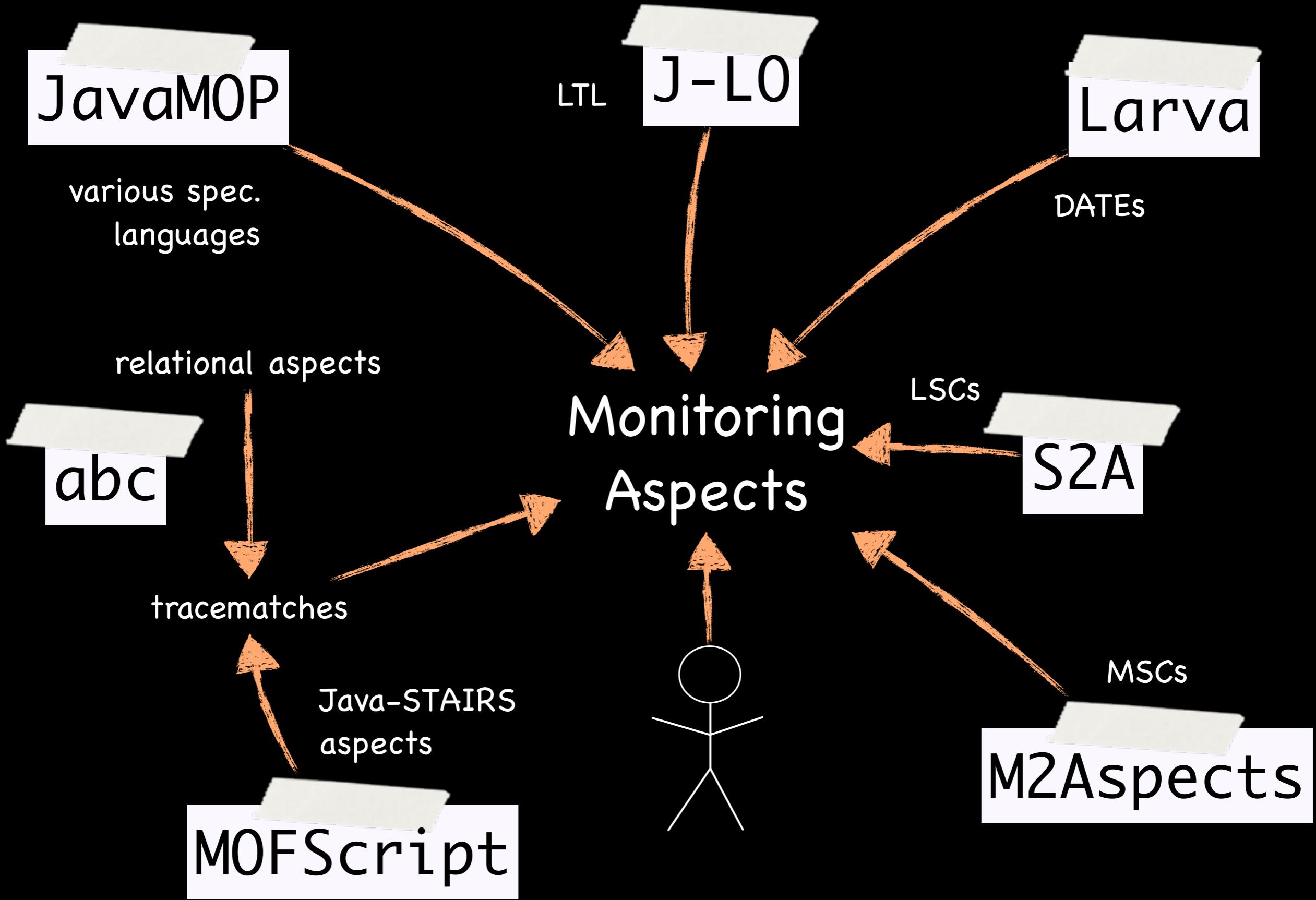
- Tedious: have to add state variables to every implementation of Connection interface!
Aspects let you monitor interface types 
- Cluttering: error-checking code is all over the place!
Aspects modularize the monitoring code 
- Hard to analyze: hard to see what the code actually checks
- Problematic: external hash maps/sets can lead to memory leaks

Aspects vs. manual monitors

- Tedious: have to add state variables to every implementation of Connection interface!
Aspects let you monitor interface types 
- Cluttering: error-checking code is all over the place!
Aspects modularize the monitoring code 
- Hard to analyze: hard to see what the code actually checks 
- Problematic: external hash maps/sets can lead to memory leaks

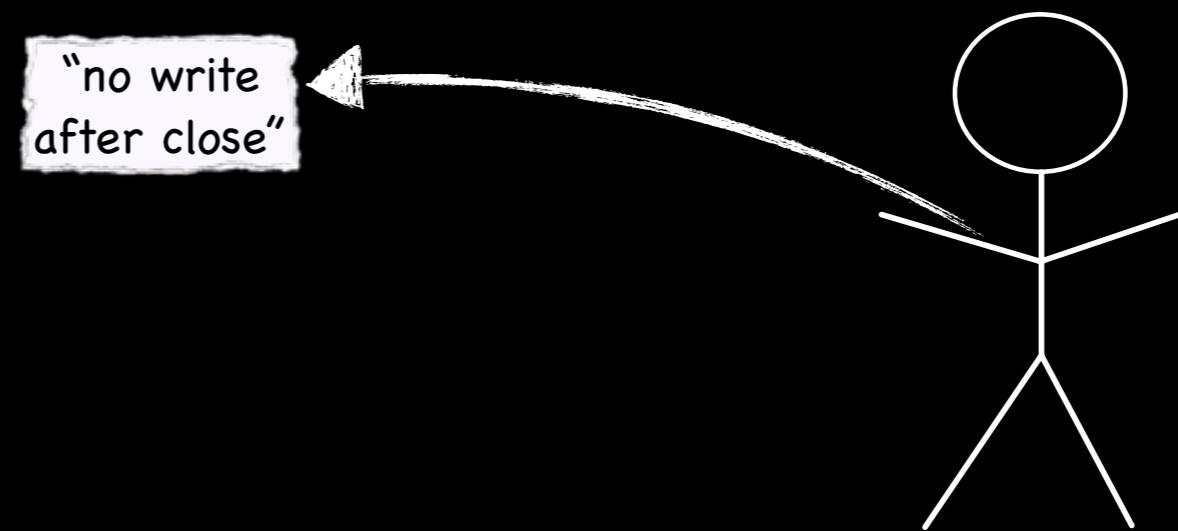
Aspects vs. manual monitors

- Tedious: have to add state variables to every implementation of Connection interface!
Aspects let you monitor interface types 
- Cluttering: error-checking code is all over the place!
Aspects modularize the monitoring code 
- Hard to analyze: hard to see what the code actually checks 
- Problematic: external hash maps/sets can lead to memory leaks 

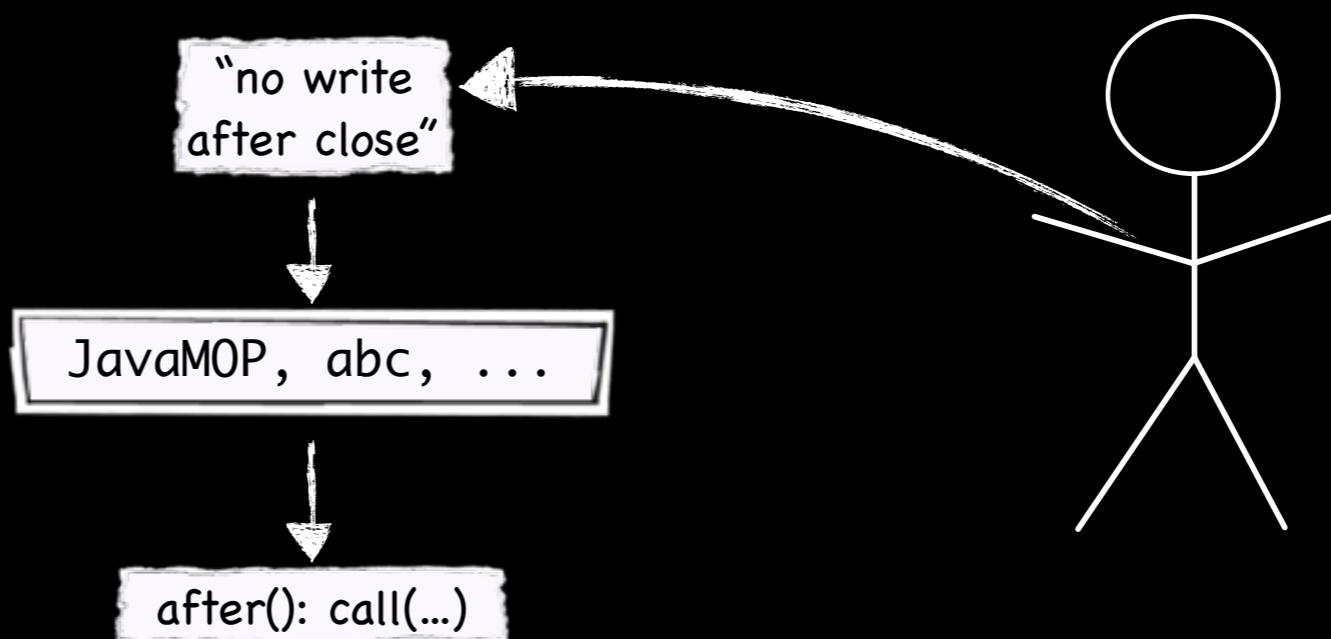


Runtime verification of finite-state properties

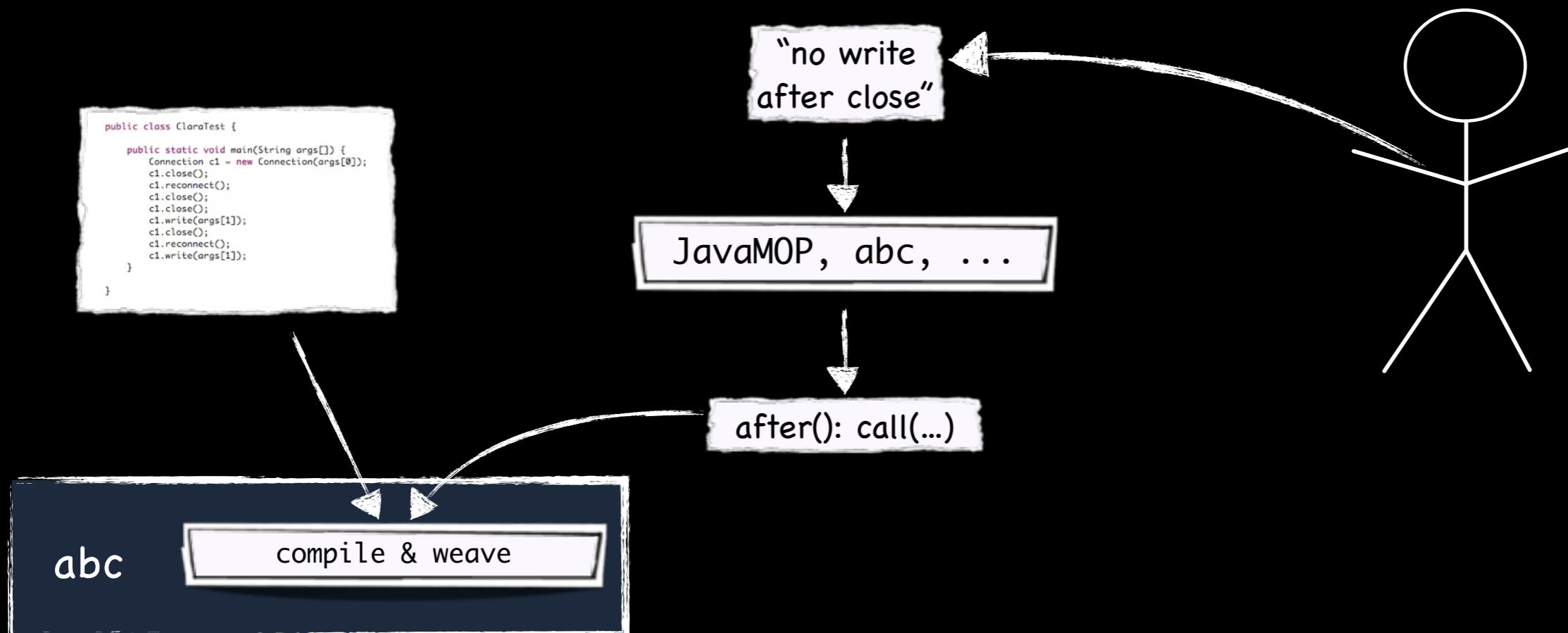
Runtime verification of finite-state properties



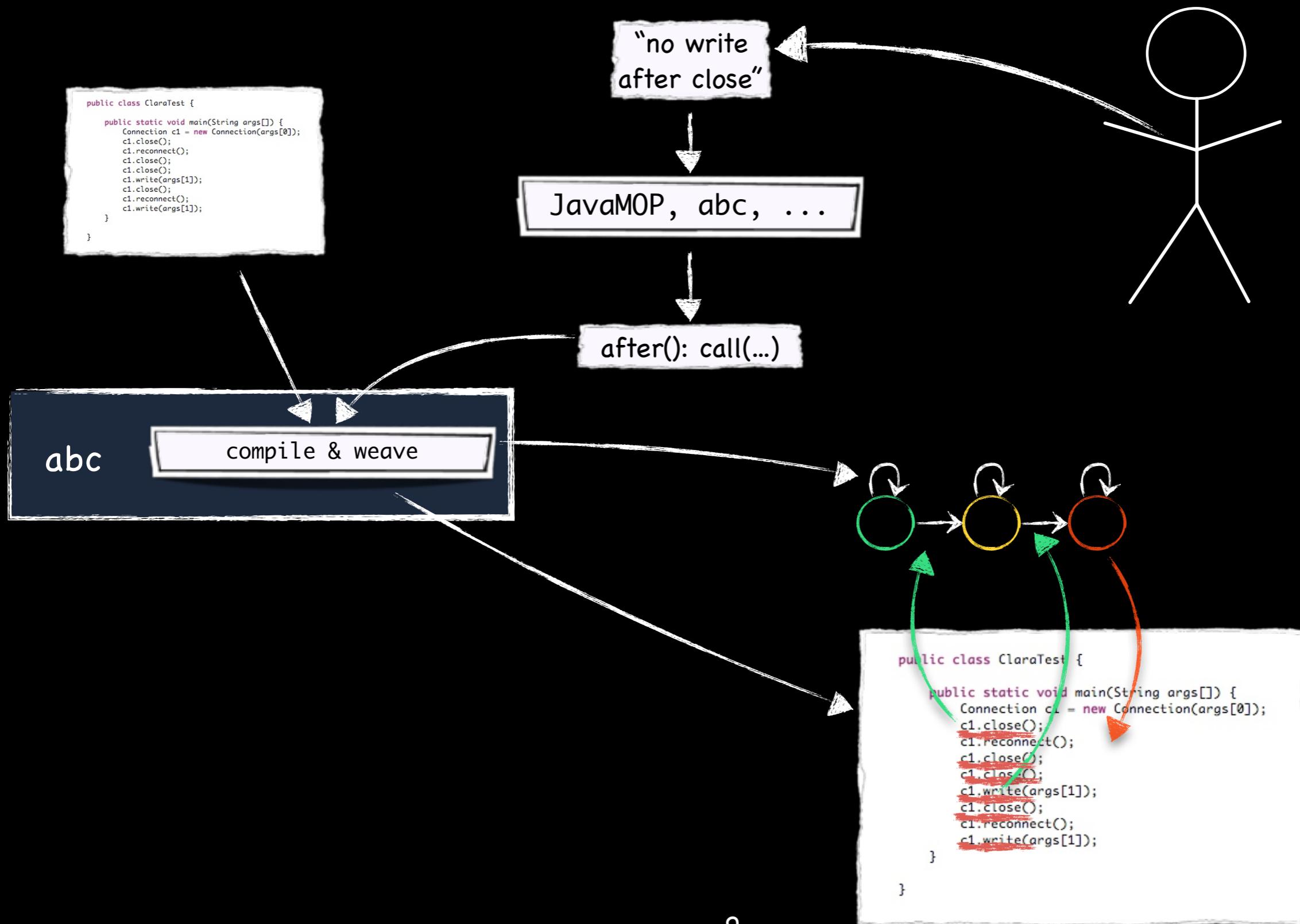
Runtime verification of finite-state properties



Runtime verification of finite-state properties



Runtime verification of finite-state properties



Runtime verification of finite-state properties



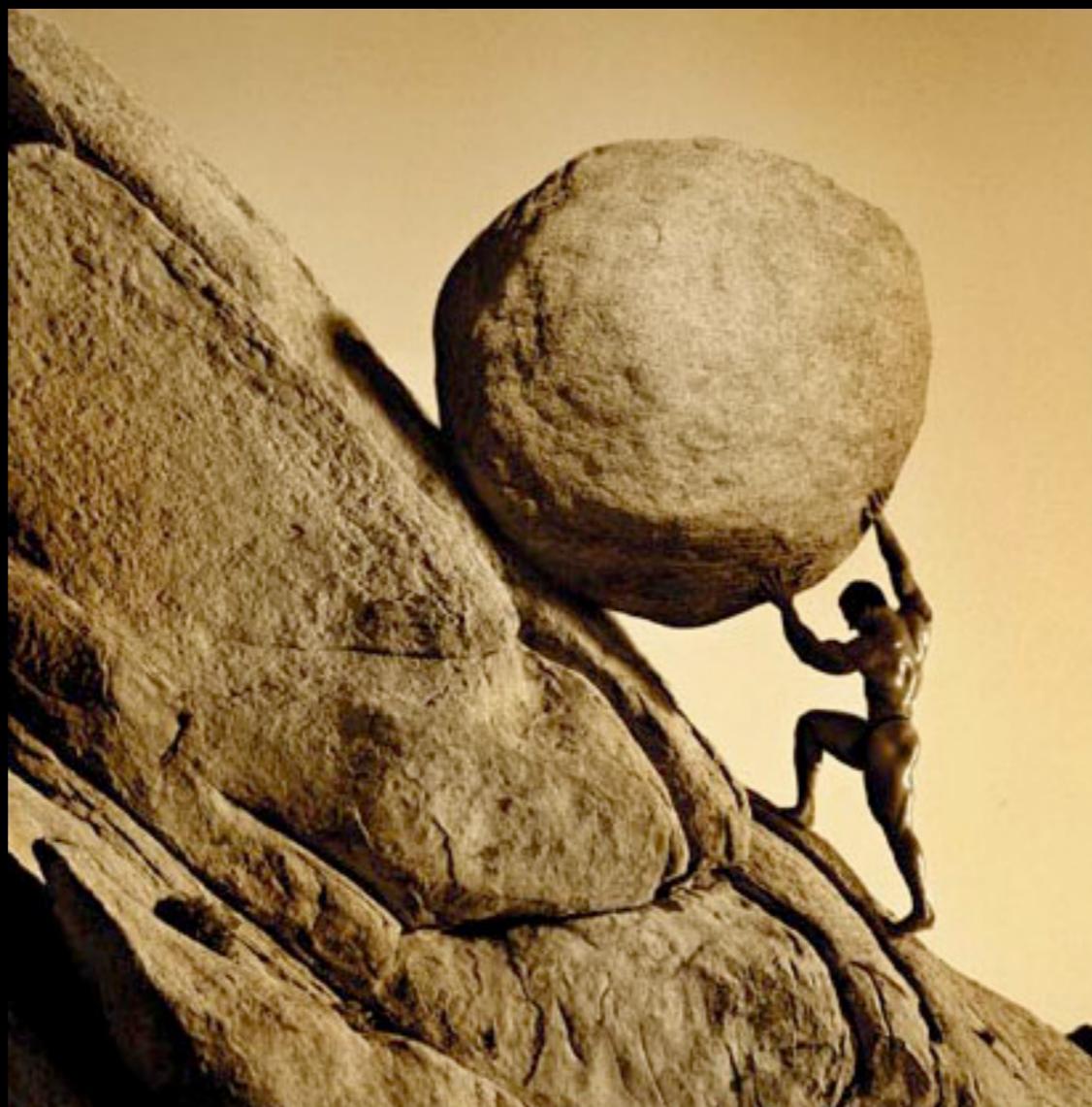
No static guarantees

Runtime verification of finite-state properties



Potentially large runtime overhead

Runtime verification of finite-state properties

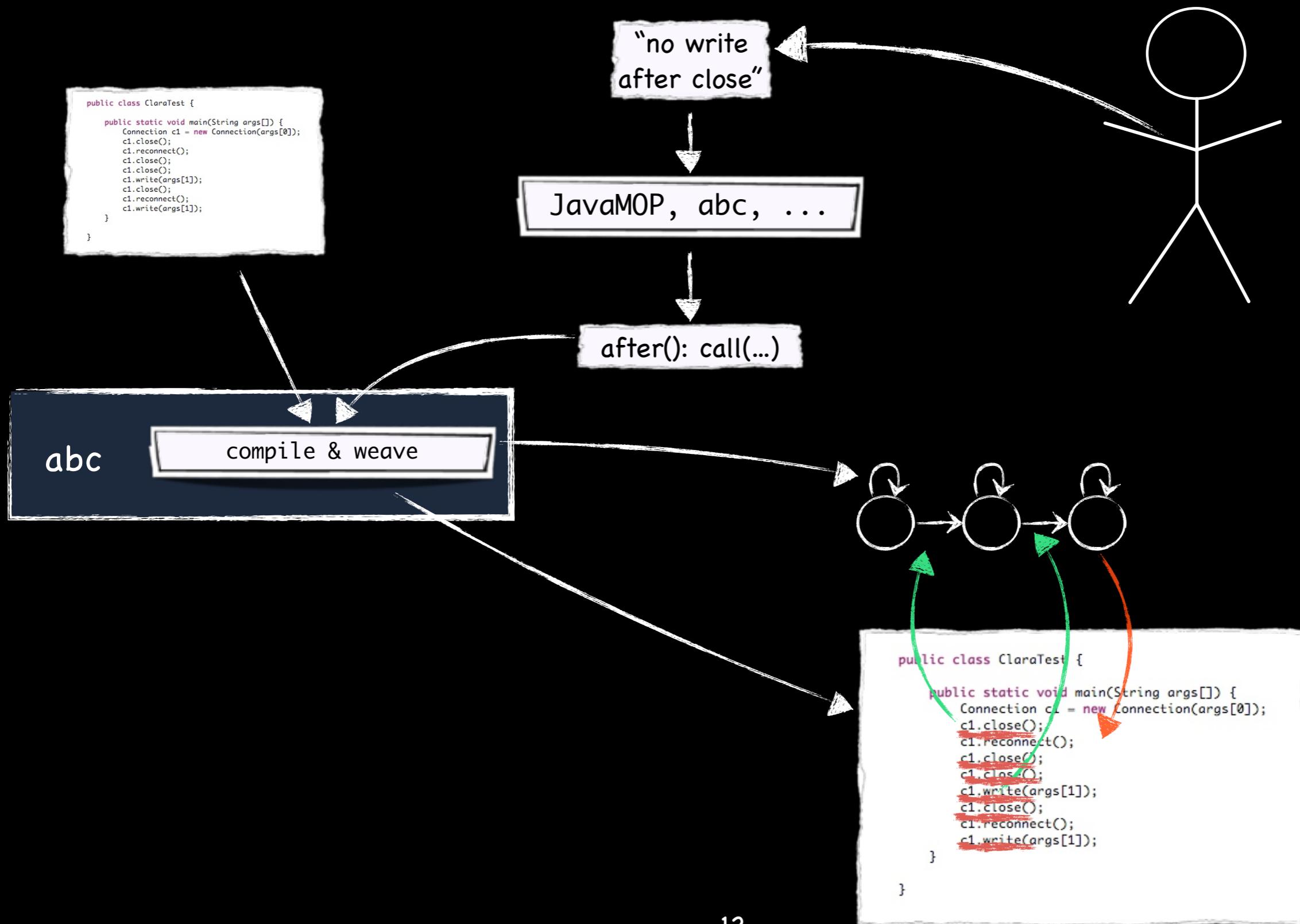


When to finish testing?

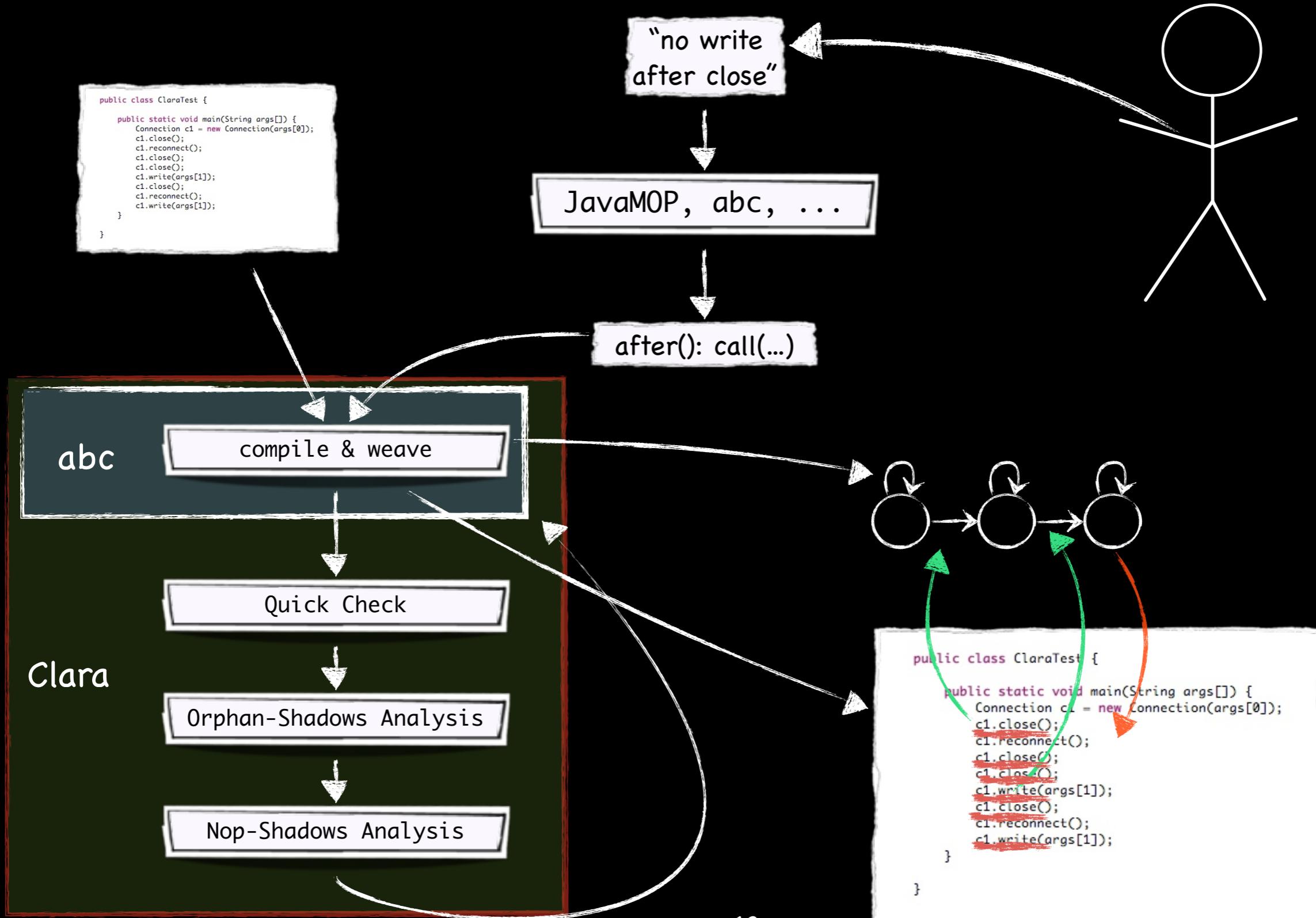
General Idea

- Use static analysis to approximate **at compile time** what the runtime monitor will do **at runtime**
- Use analysis information to **identify instrumentation points ("shadows")** that **require no monitoring**
- When no shadow requires monitoring then the program cannot violate the monitored property -> **static proof!**

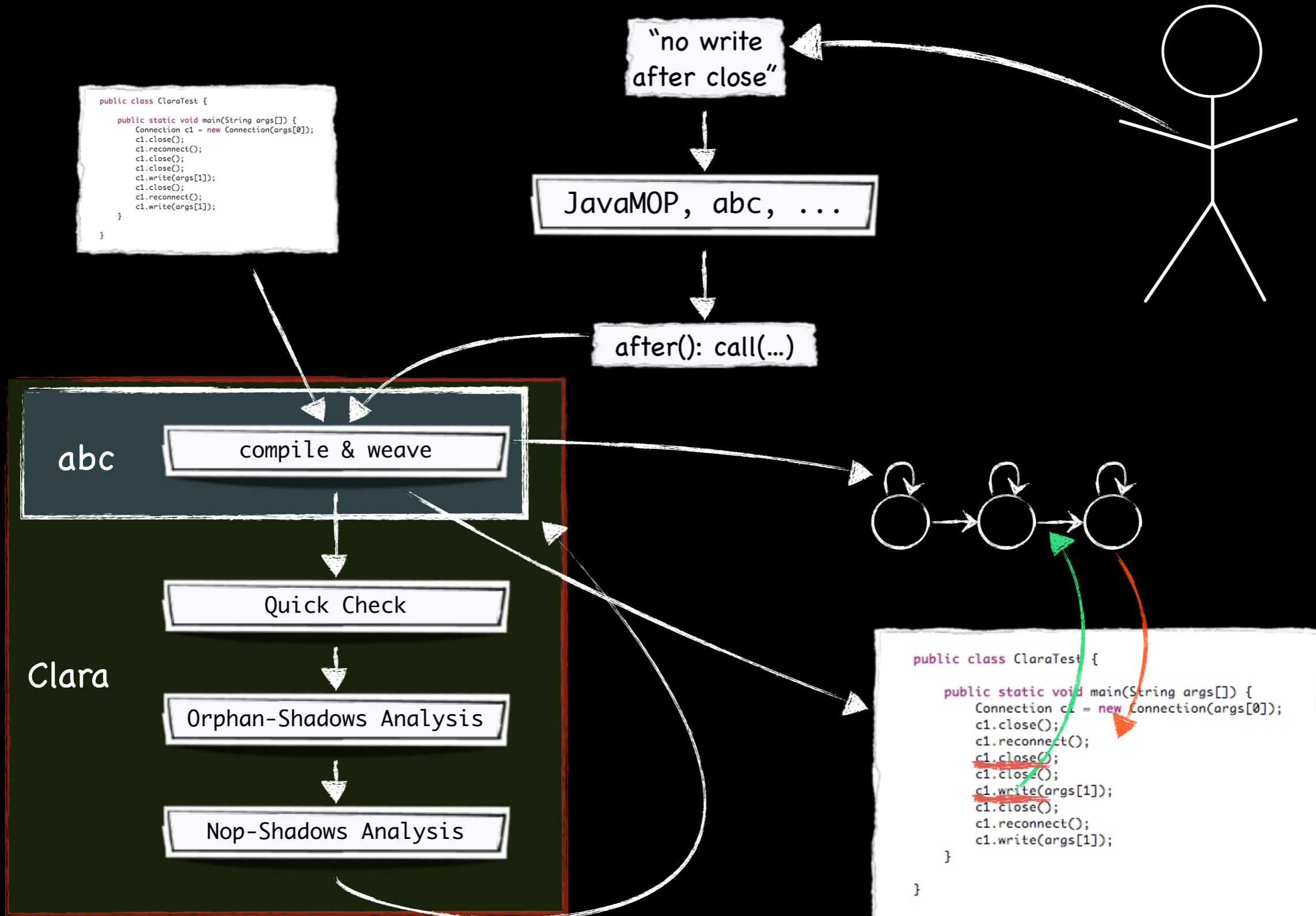
The Clara Framework



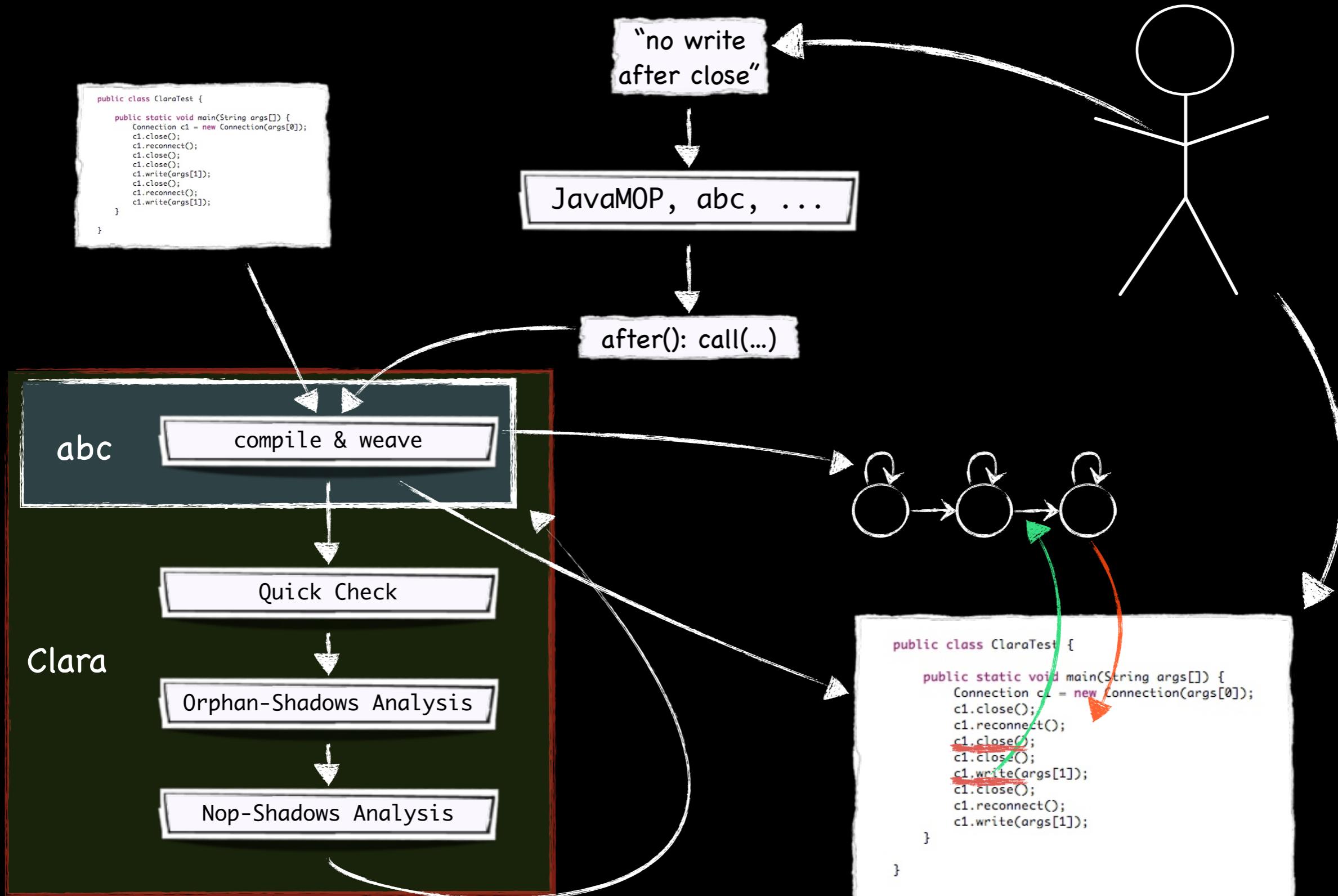
The Clara Framework



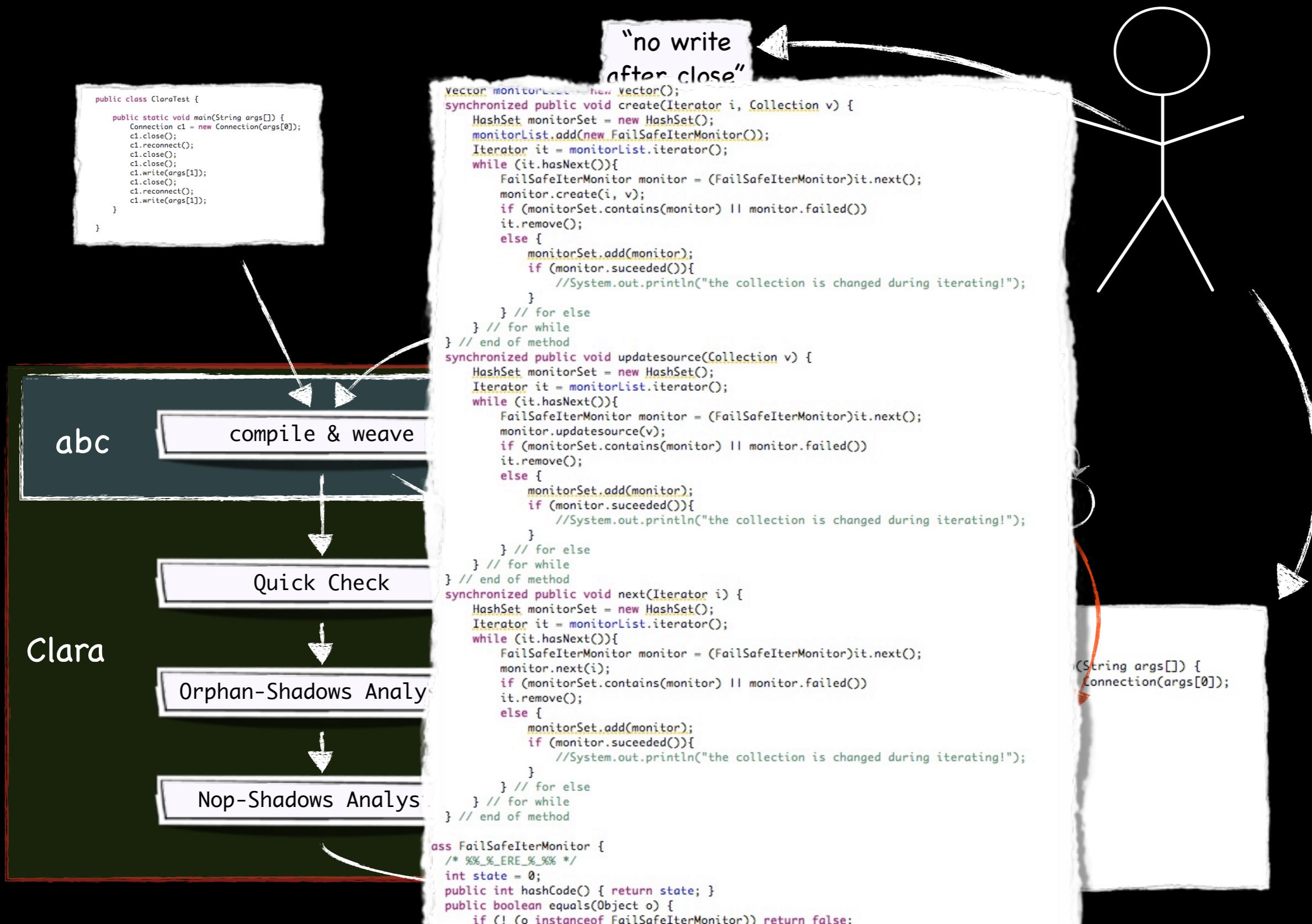
The Clara Framework



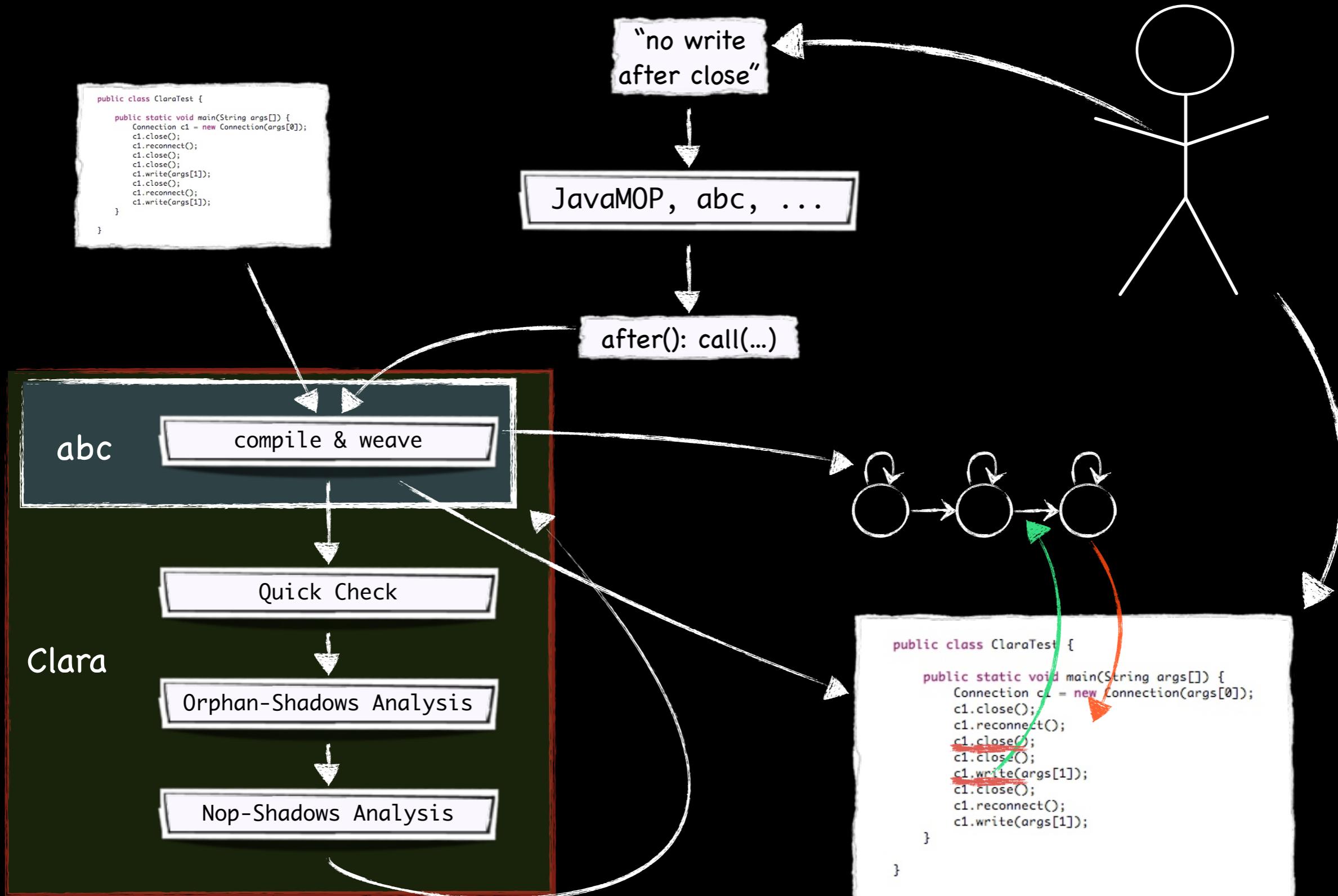
The Clara Framework



The Clara Framework



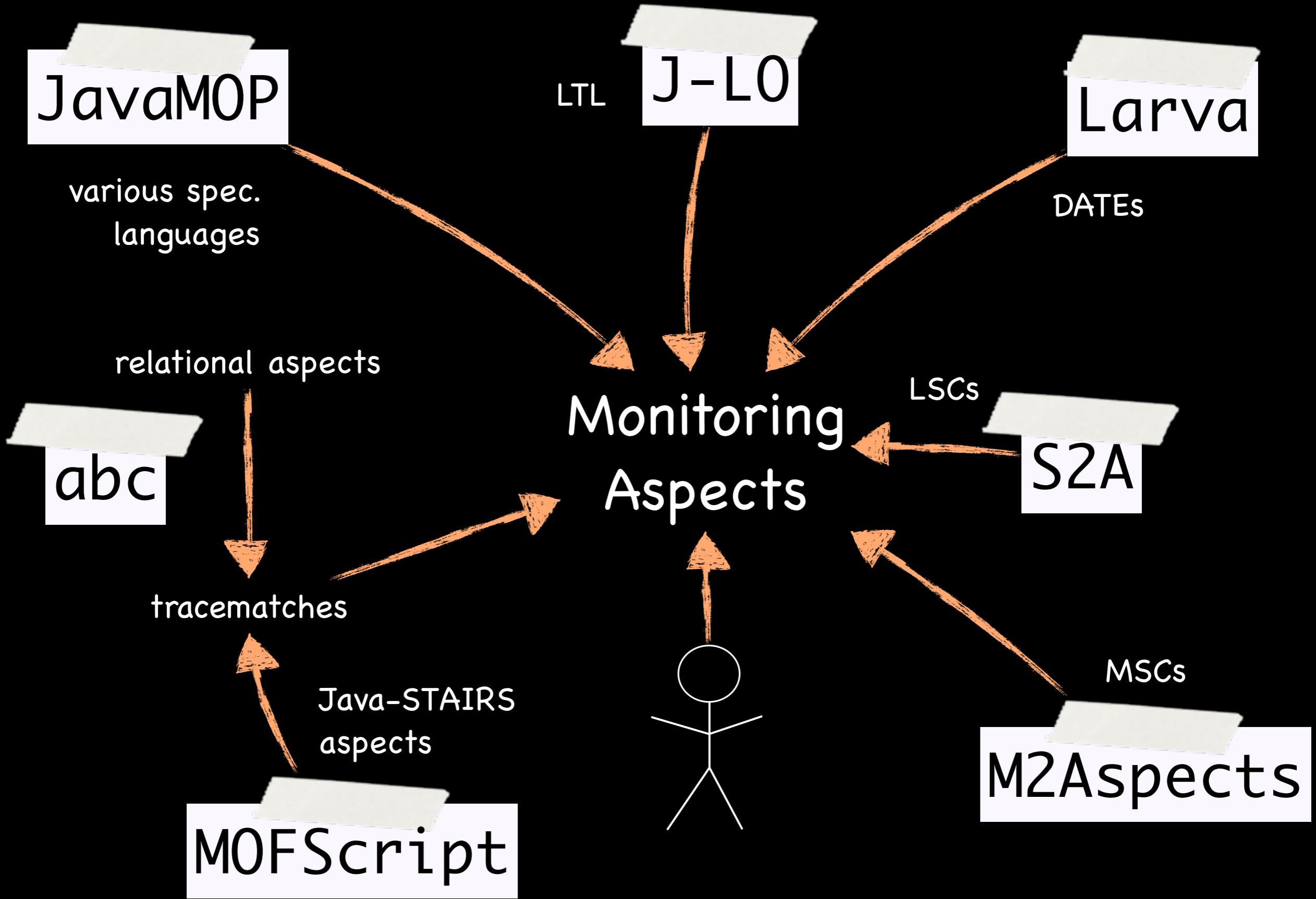
The Clara Framework



Clara - Design Goals

- ⦿ Design goals:
 - ⦿ Support a wide range of RV tools
 - ⦿ For every tool, support the same range of static analyses / optimizations
- ⦿ Challenge:
 - ⦿ Right level of abstraction

Runtime monitoring:
Existing tools and
approaches



JavaMOP

The screenshot shows the JavaMOP 2.1 website. At the top left is the Formal Systems Laboratory logo (a stylized 'f' and 'l' in a circle). The main title "JavaMOP 2.1" is centered above a navigation bar with links for Authors, Links, and a search bar. The navigation bar also includes links for Main Page, People, Projects, Publications, News, and Events. Below the navigation is a "Languages" section with a "MOP Matrix" table. The matrix has "Languages" on the left and "Logic Plugins" at the top. It contains entries for JavaMOP (highlighted in yellow) and BusMOP. The matrix table is titled "MOP Matrix: a clickable map of MOP pages." Below the matrix are download links for "javamop2.1.2.jar" and "All versions HERE". Prerequisites listed are JDK (Java Development Platform) 6 or later and AspectJ 1.5.x or later. A note states that JavaMOP is an instance of MOP for Java, with links to papers and the manual. It also provides an email address for support. A "Run JavaMOP Online" section allows users to choose an example from a dropdown menu (CFG, ERE, HasNext, HasNext2, HashSet) and view the corresponding Java code in a text editor.

JavaMOP 2.1

Authors: Feng Chen, Patrick Meredith, Dongyun Jin, Grigore Rosu

Links: MOP Syntax, JavaMOP Syntax, JavaMOP News and Logs, All Versions

Main Page | People | Projects | Publications | News | Events

Search: Go | Search

MOP Matrix: a clickable map of MOP pages.

Download: javamop2.1.2.jar | All versions HERE

Prerequisites: JDK (Java Development Platform) (6 or later), AspectJ (1.5.x or later)

JavaMOP is an instance of MOP for Java. HERE you can find our JavaMOP papers and HERE you can find the JavaMOP manual. Below you can run JavaMOP online. If you seek further help or if you want to report a bug, please send us a message at mop@cs.uiuc.edu.

Run JavaMOP Online

Choose an example:

- CFG
- ERE
 - HasNext
 - HasNext2
 - HashSet

ERE/SafeEnum

```
package mop;
import java.io.*;
import java.util.*;

// The SafeEnum property is designed
// to match a case where a Collection
// with an associated Enumeration is
```

<http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP2.1>

JavaMOP – Design

- there is no ideal specification language for temporal properties
- LTL may be good for one property, regular expressions or CFGs for another
- hence support multiple frontends, each of which supports different properties
- use AspectJ as target language
- allow for parameterized traces, independently of the frontend used

JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){})  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e) {}  
  
    ere : create next* updatesource updatesource* next  
  
    @match {  
        //issue error message  
    }  
}
```

JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){})  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e) {}  
  
    ere : create next* updatesource updatesource* next  
  
    @match {  
        //issue error message  
    }  
}
```

JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {
```

```
    event create after(Vector v)
        returning(Enumeration e) :
            call(Enumeration Vector+.elements())
            && target(v) {}

    event updatesource after(Vector v) :
        (call(* Vector+.remove*(..))
         || call(* Vector+.add*(..)) ...
            && target(v) {}

    event next before(Enumeration e) :
        call(* Enumeration+.nextElement())
        && target(e) {}
```

ere : create next* updatesource updatesource* next

```
@match {
    //issue error message
}
```

JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){}  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e){}  
  
    ere : create next* updatesource updatesource* next  
  
    @match {  
        //issue error message  
    }  
}
```

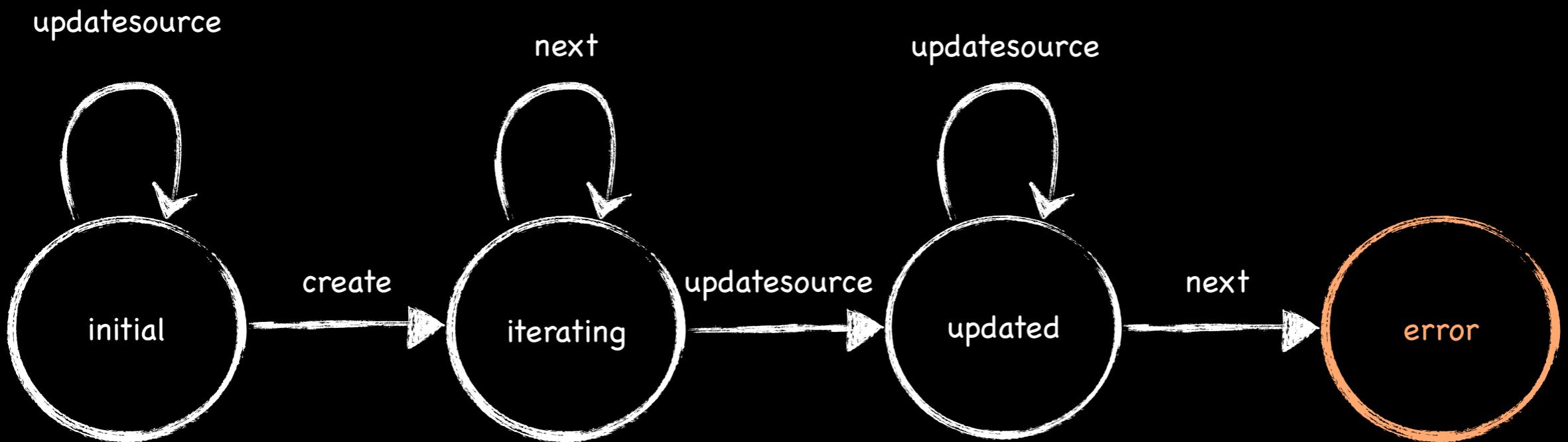
JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){}  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e){}  
  
    ere : create next* updatesource updatesource* next
```

```
@match {  
    //issue error message  
}
```

}

SafeEnum Automaton

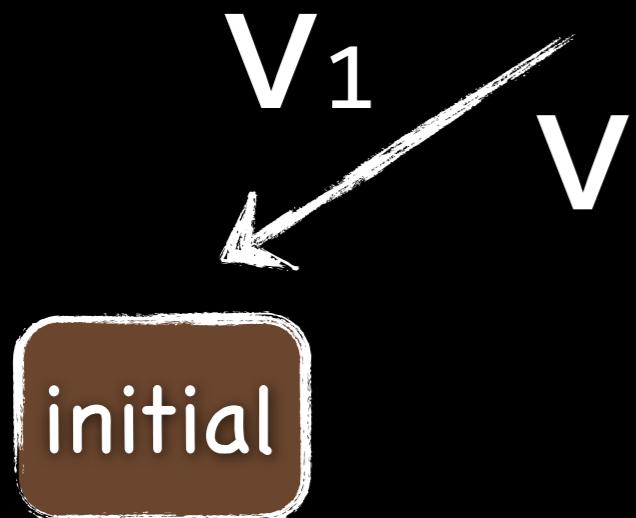


JavaMOP - Indexing

JavaMOP - Indexing

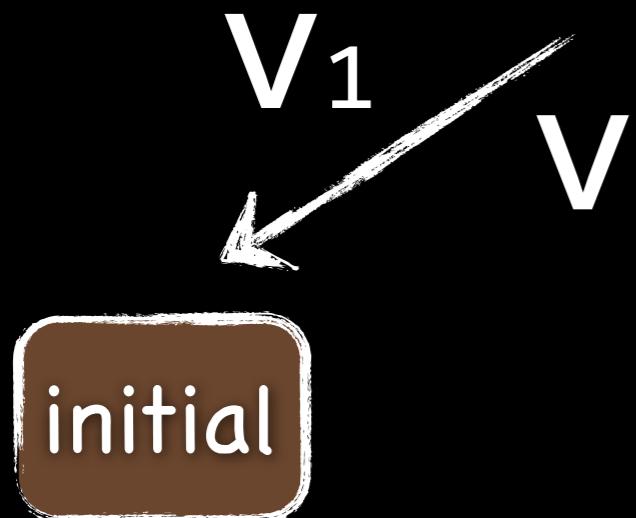
v₁.add(. . .)

JavaMOP - Indexing



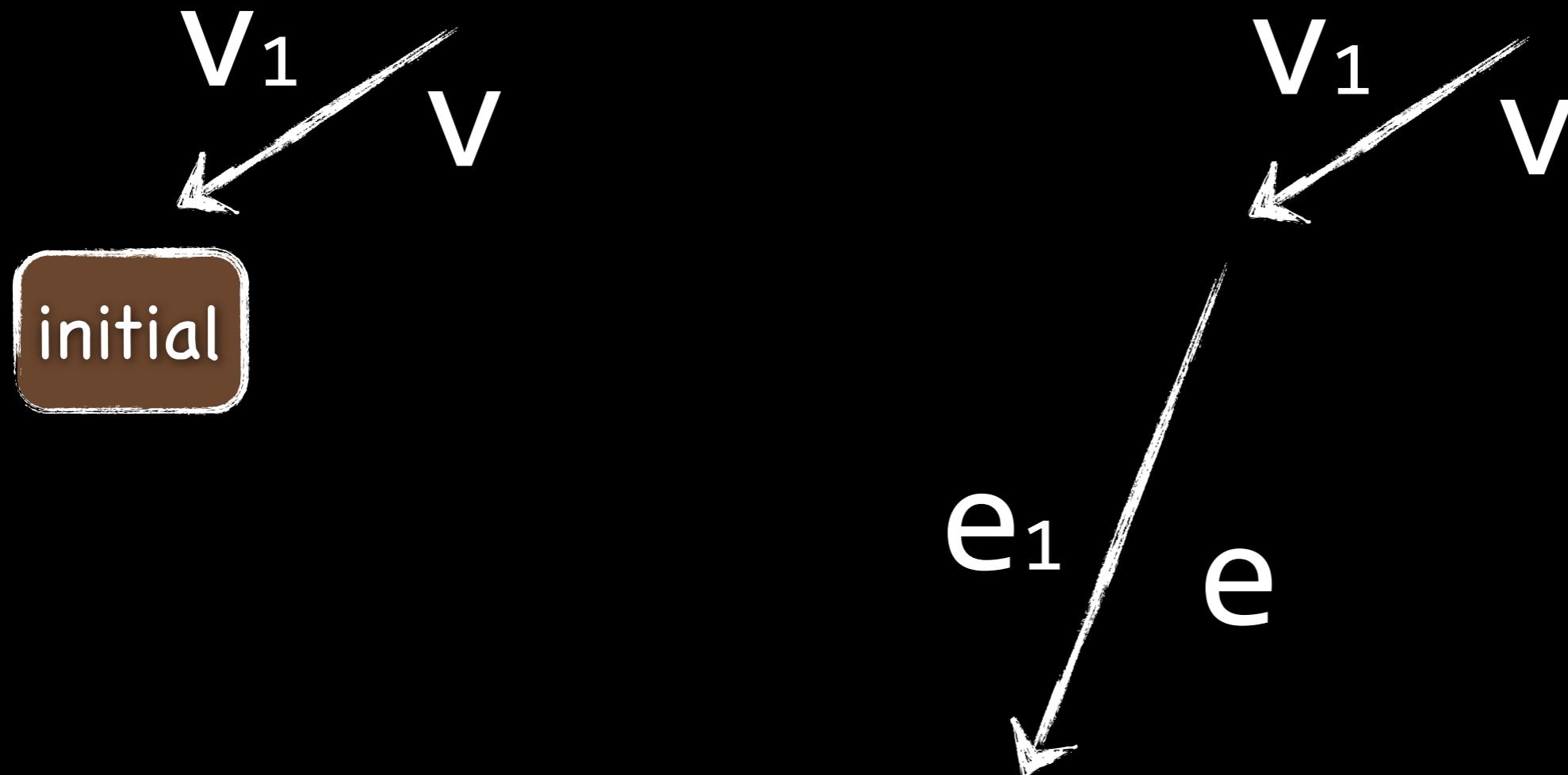
$v_1 . add(\dots)$

JavaMOP - Indexing



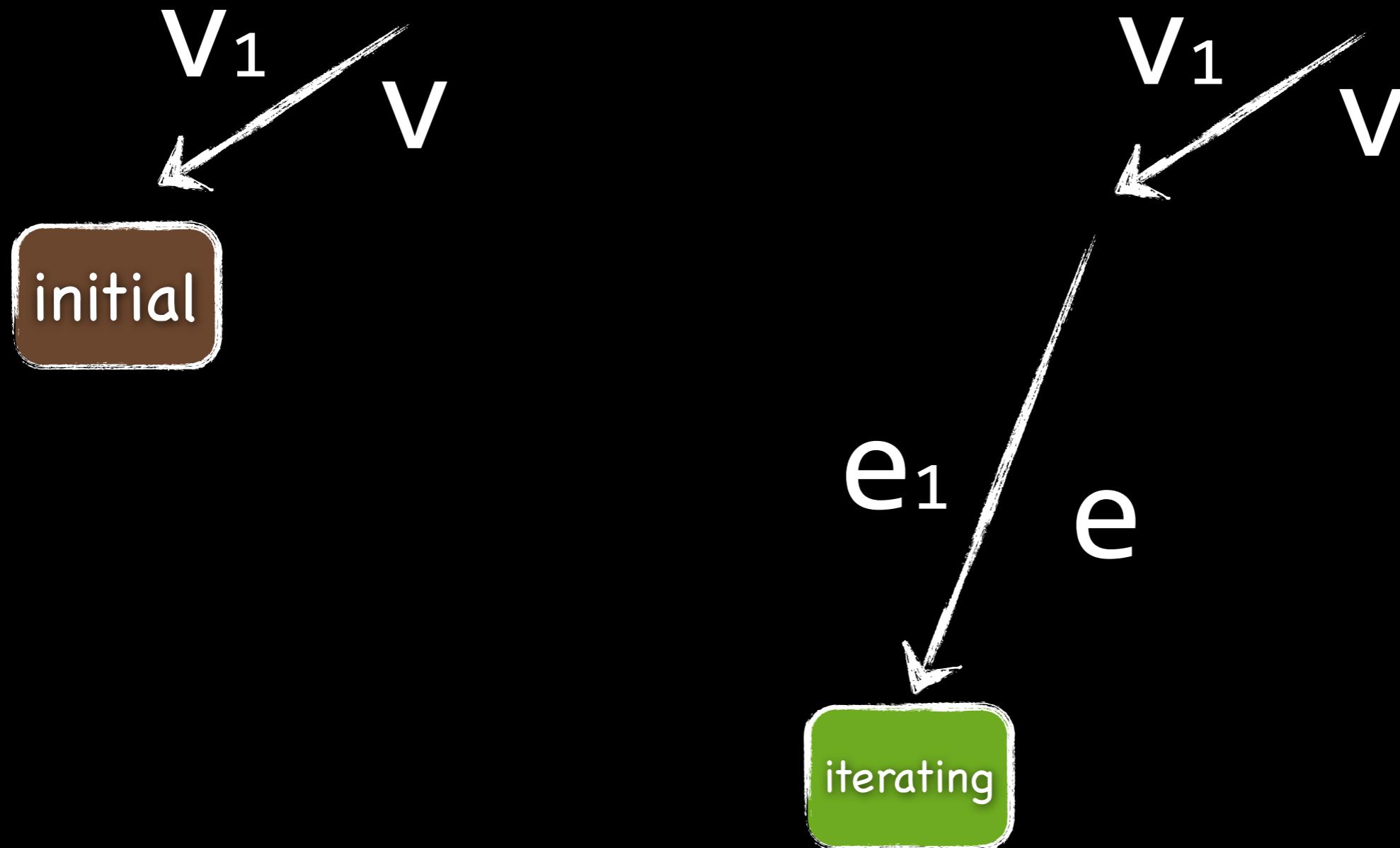
$e_1 = V_1.\text{elements}()$

JavaMOP - Indexing



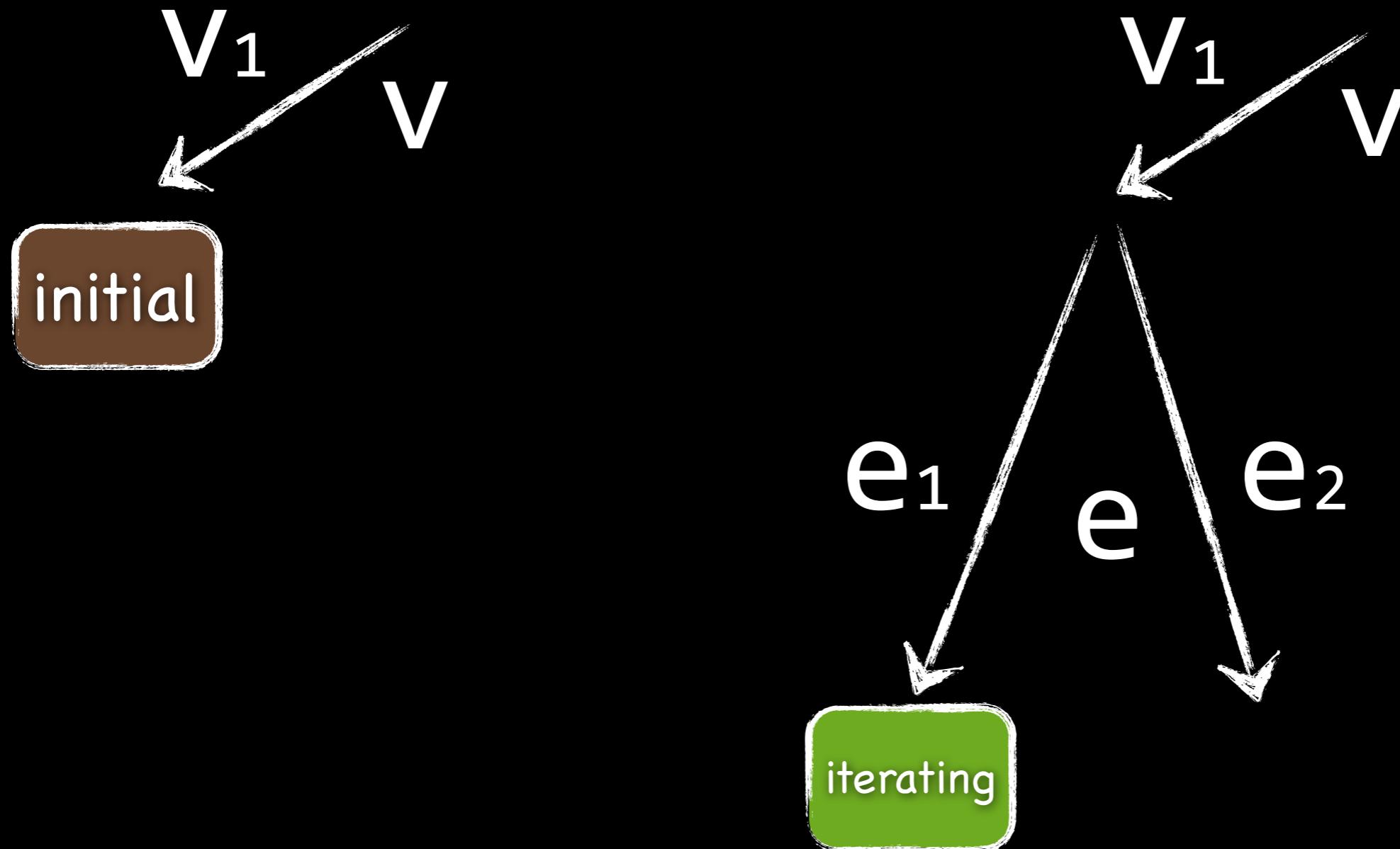
`e1 = V1.elements()`

JavaMOP - Indexing



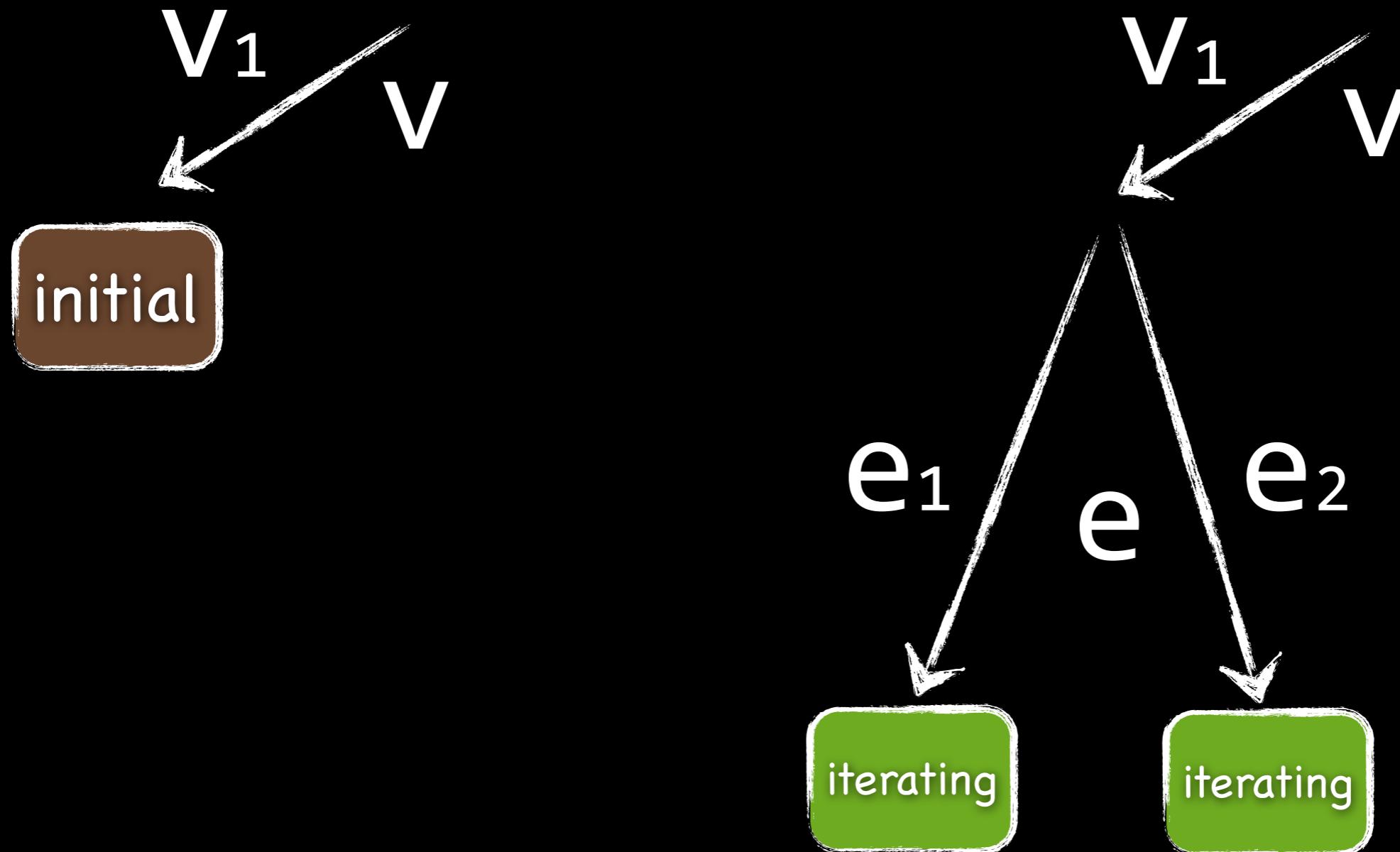
$e_1 = V_1.\text{elements}()$

JavaMOP - Indexing



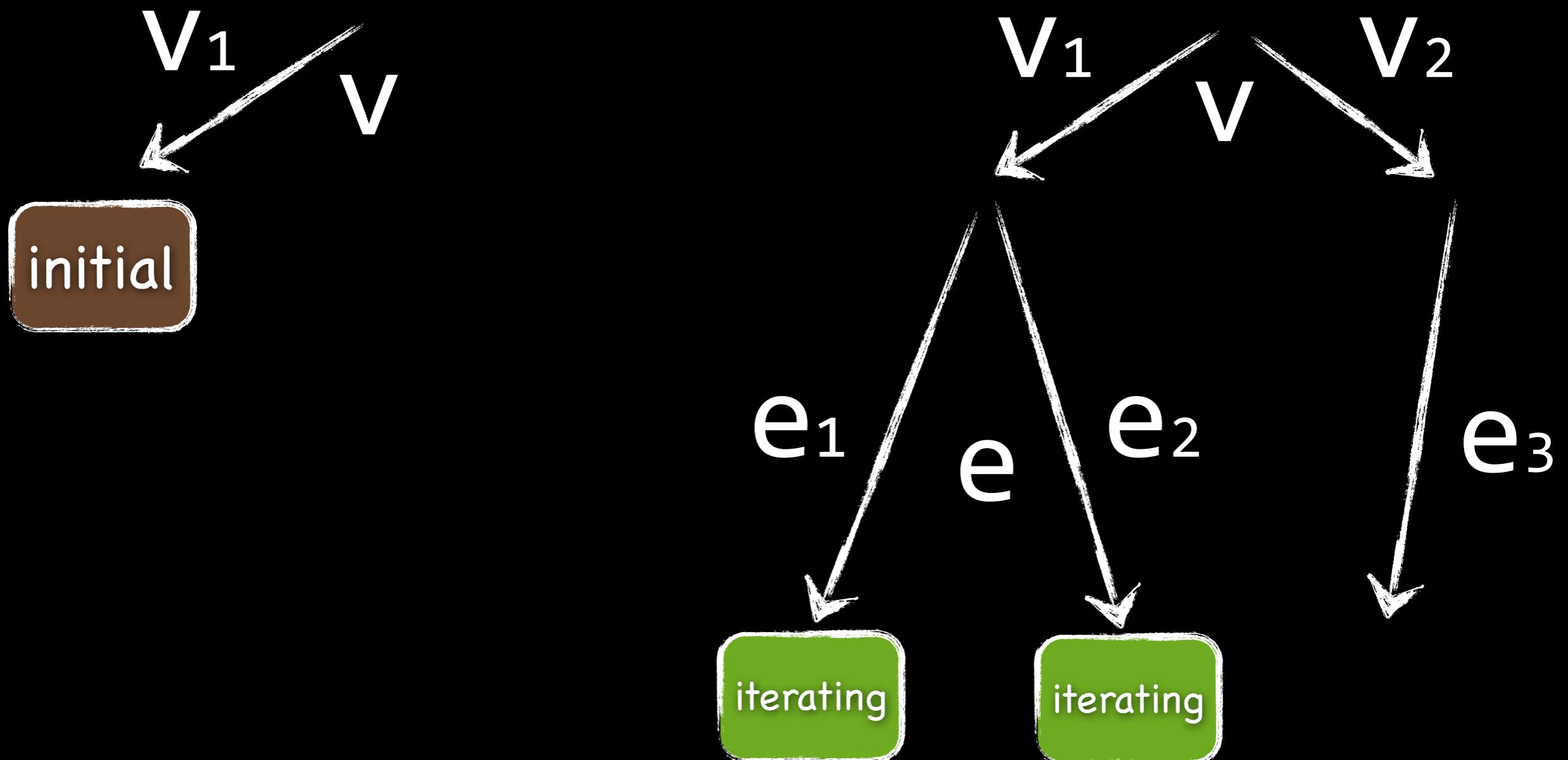
$e_2 = V_1.\text{elements}()$

JavaMOP - Indexing



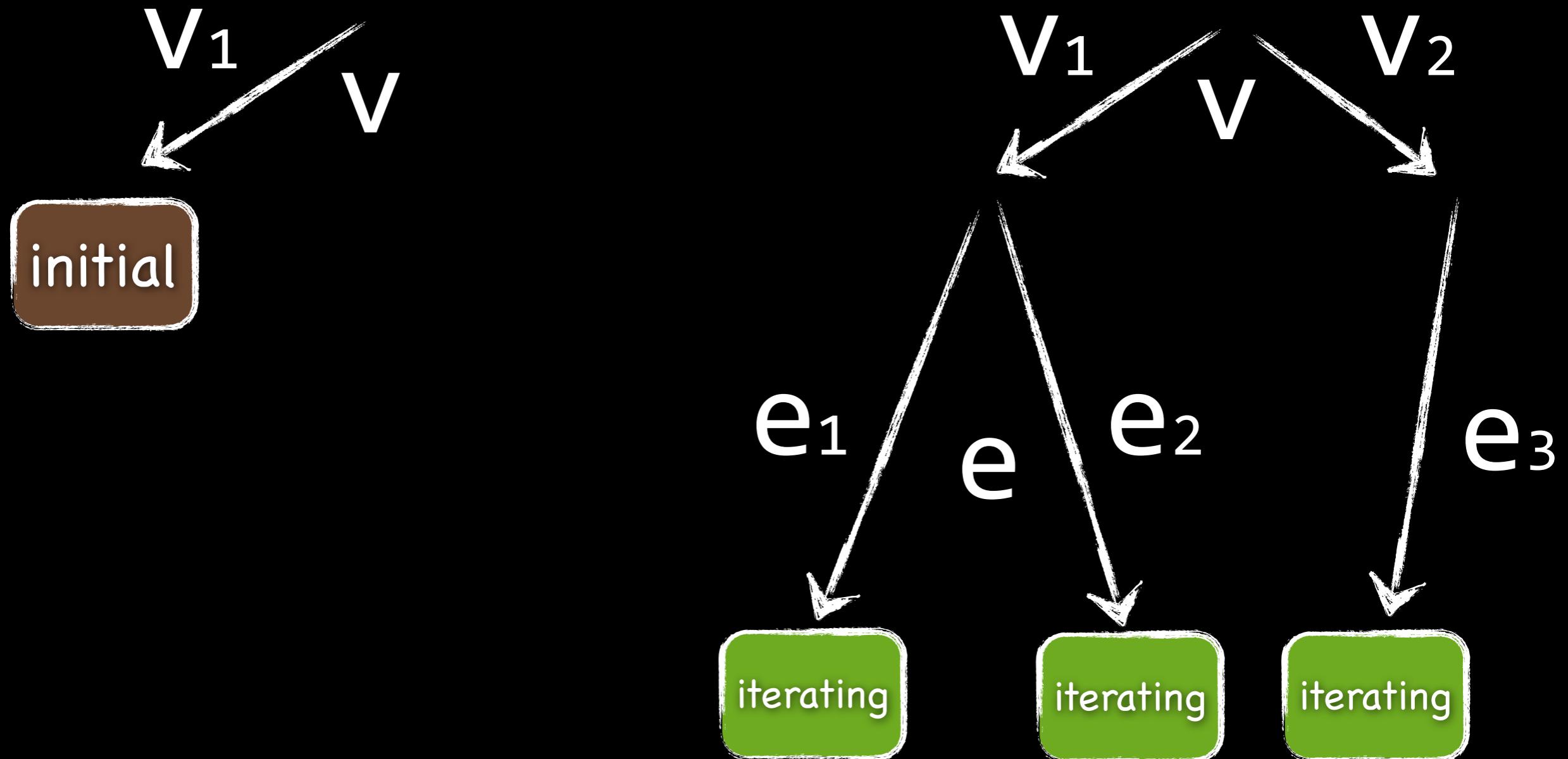
$e_2 = V_1.\text{elements}()$

JavaMOP - Indexing



$e_3 = V_2.\text{elements}()$

JavaMOP - Indexing



$e_3 = V_2.\text{elements}()$

JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){})  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e) {}  
  
    ere : create next* updatesource updatesource* next  
  
    @match {  
        //issue error message  
    }  
}
```

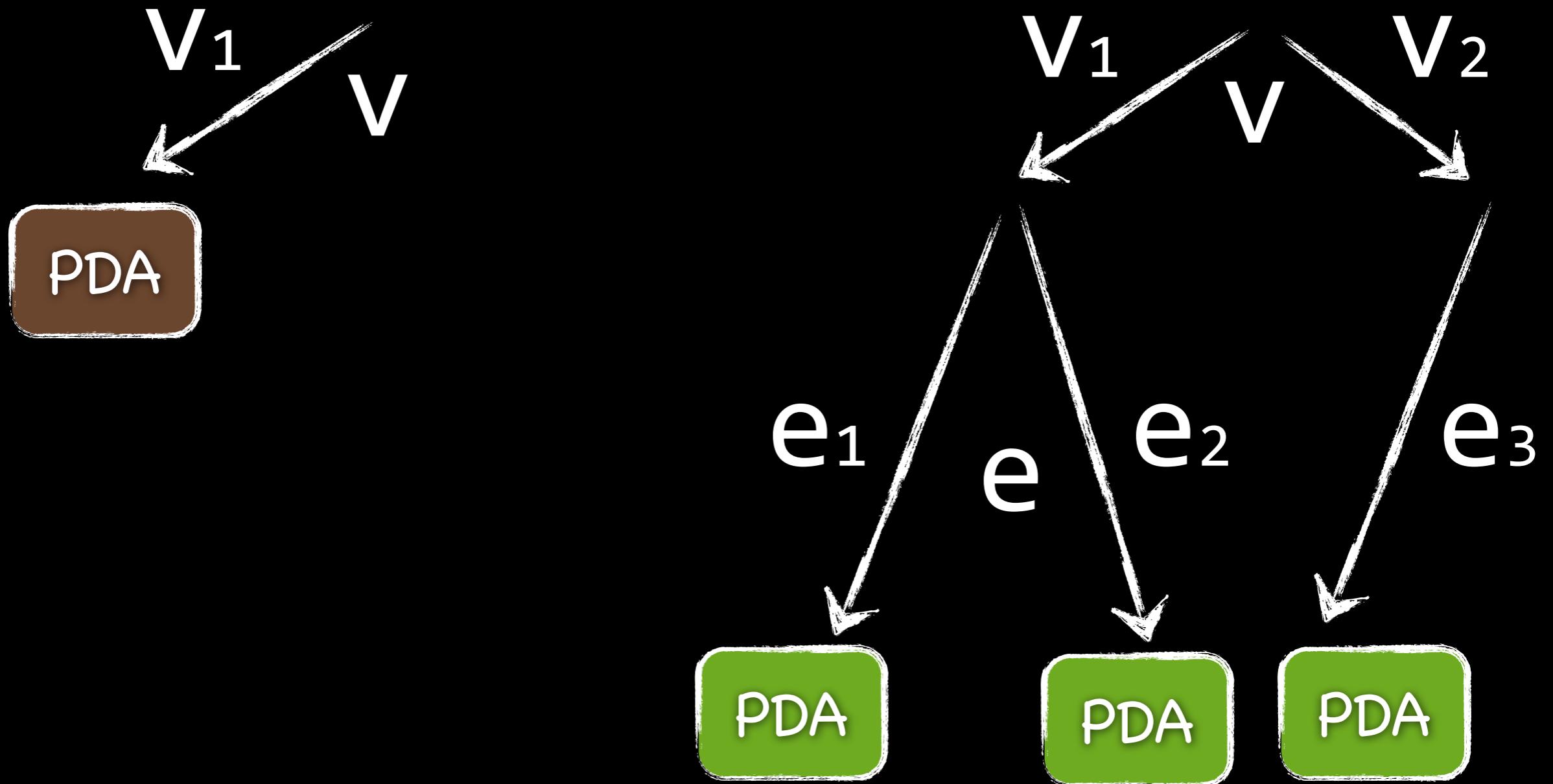
JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){}  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e){}  
  
    ere : create next* updatesource updatesource* next  
  
    @match {  
        //issue error message  
    }  
}
```

JavaMOP – Design

```
SafeEnum(Vector v, Enumeration e) {  
  
    event create after(Vector v)  
        returning(Enumeration e) :  
            call(Enumeration Vector+.elements())  
            && target(v) {}  
    event updatesource after(Vector v) :  
        (call(* Vector+.remove*(..))  
        || call(* Vector+.add*(..)) ...  
        && target(v){}  
    event next before(Enumeration e) :  
        call(* Enumeration+.nextElement())  
        && target(e){}  
  
    cfg : S -> create N U next, N -> next N | epsilon,  
        U -> updatesource U | updatesource  
  
    @match {  
        //issue error message  
    }  
}
```

JavaMOP - Indexing



JavaMOP – Code generation

- generates at least two classes:
 - aspect, intercepting events and implementing indexing
 - monitor class, depending on the specification formalism
- aspect always the same, irrespective of the formalism
- monitor class is very specialized, as it is highly optimized

Tracematches (abc)



abc: The AspectBench Compiler for AspectJ

[Introduction] [Download] [Bugs] [Documentation] [FAQ] [Mailing lists] [People] [Publications] [Projects & Extensions] [Benchmarks]

Introduction

News:

- New paper [Dependent Advice: A General Approach to Optimizing History-based Aspects](#) available

Latest version: 1.3.0, released on April 2nd, 2008

abc is a complete implementation of [AspectJ](#) (with some, mostly minor, [differences](#) to the original ajc compiler) that aims to make it easy to implement both extensions and optimisations of the core language. It provides two frontends: one built on the [Polyglot](#) framework for extensible Java compilation, the other an extension of the [JastAddJ](#) extensible Java compiler. Its backend is built on the [Soot](#) framework for Java optimisation.

For developers, abc offers:

- Fast code
- Alternative error messages
- Support through email lists and public bugzilla

For researchers, abc offers:

- An extensible frontend based on Polyglot
- An extensible backend based on Soot
- An ideal platform for experiments with new features and optimisations

abc is a joint project between the [Programming Tools Group](#) at Oxford University and the [Sable research group](#), mainly based at McGill University in Montreal. It is free software, licensed under the [GNU LGPL](#). We encourage people both to use it as an alternative [AspectJ](#) implementation and to extend and improve it.

<http://aspectbench.org/>

Tracematches

- meant as AspectJ language construct:
pointcuts over the execution history
- runtime verification is just one of many
possible applications
- looks similar to ERE plugin of JavaMOP
- but implementation is very different

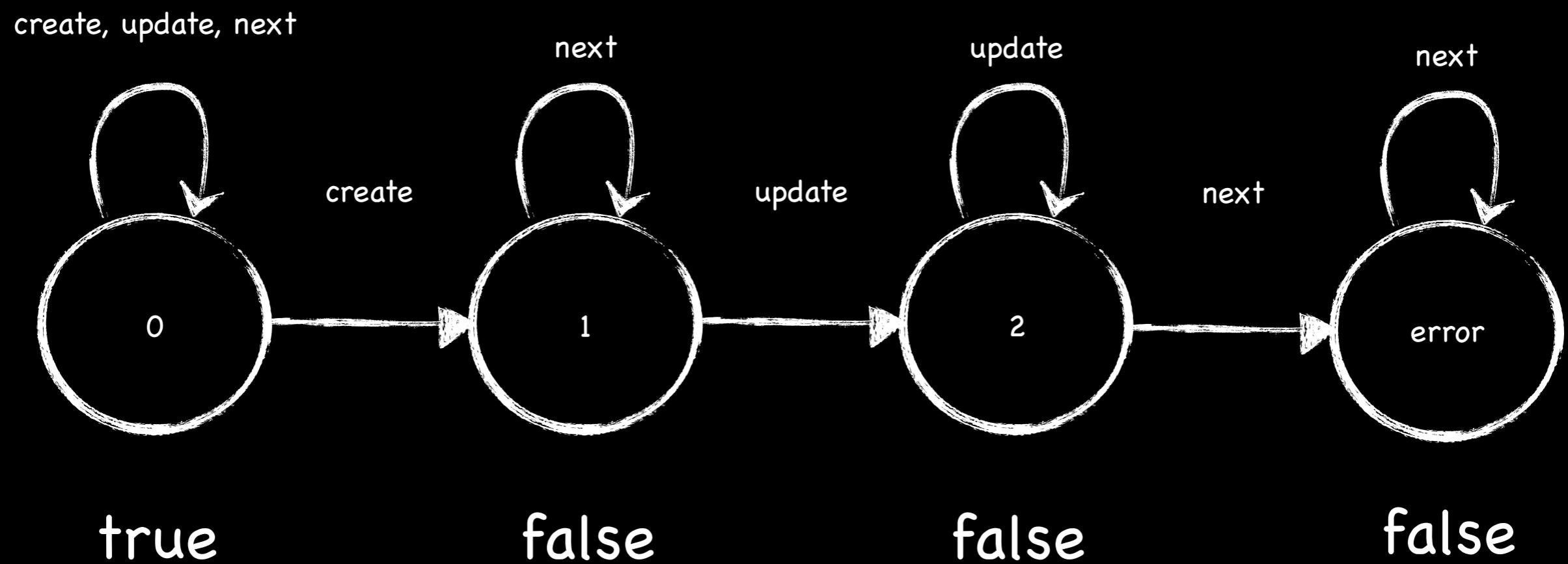
Tracematches - Syntax

```
pointcut vector_update() :  
    call(* Vector.add*(..)) || .. call(* Vector.clear()) ;  
  
tracematch(Vector v, Enumeration e) {  
    sym create after returning(e) :  
        call(* Vector+.elements()) && target(v);  
    sym next before :  
        call(Object Enumeration.nextElement()) && target(e);  
    sym update_source before :  
        vector_update() && target(v);  
  
    create next* update_source+ next  
    {  
        error("error with " + v + " and " + e);  
    }  
}
```

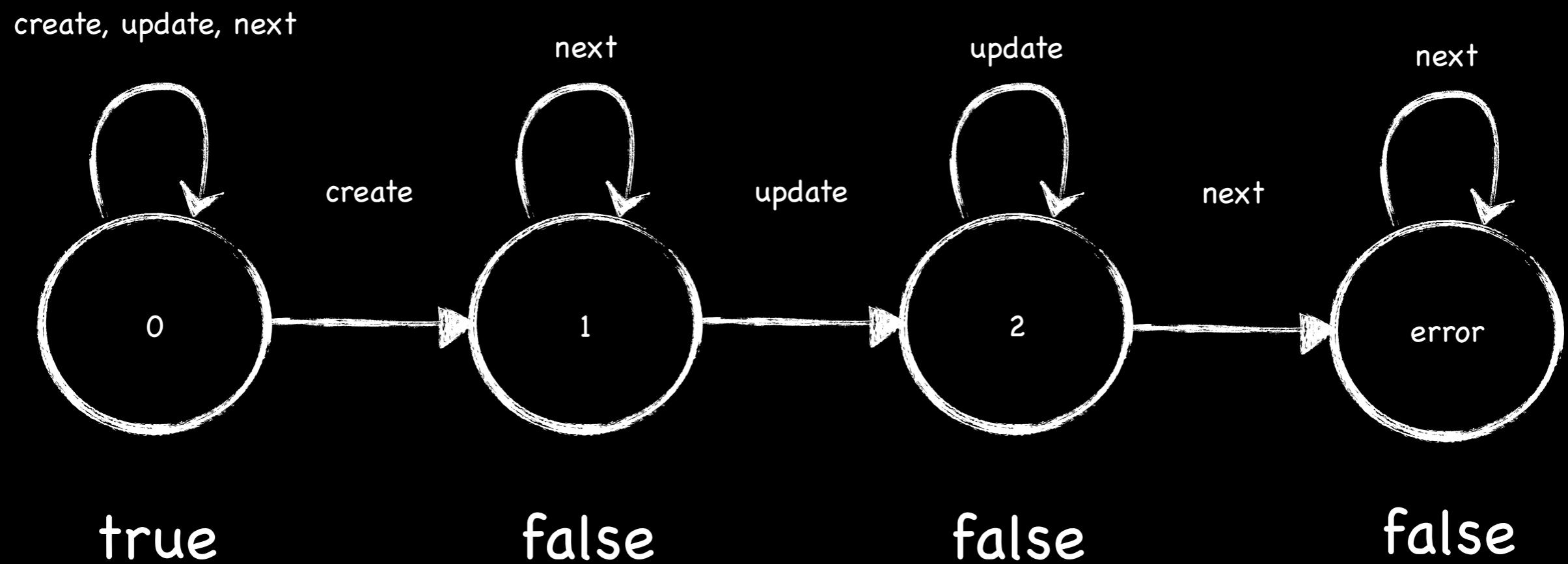
Tracematches - Syntax

```
pointcut vector_update() :  
    call(* Vector.add*(..)) || .. call(* Vector.clear()) ;  
  
tracematch(Vector v, Enumeration e) {  
    sym create after returning(e) :  
        call(* Vector+.elements()) && target(v);  
    sym next before :  
        call(Object Enumeration.nextElement()) && target(e);  
    sym update_source before :  
        vector_update() && target(v);  
  
    create next* update_source+ next  
    {  
        error("error with " + v + " and " + e);  
    }  
}
```

Tracematches - Indexing

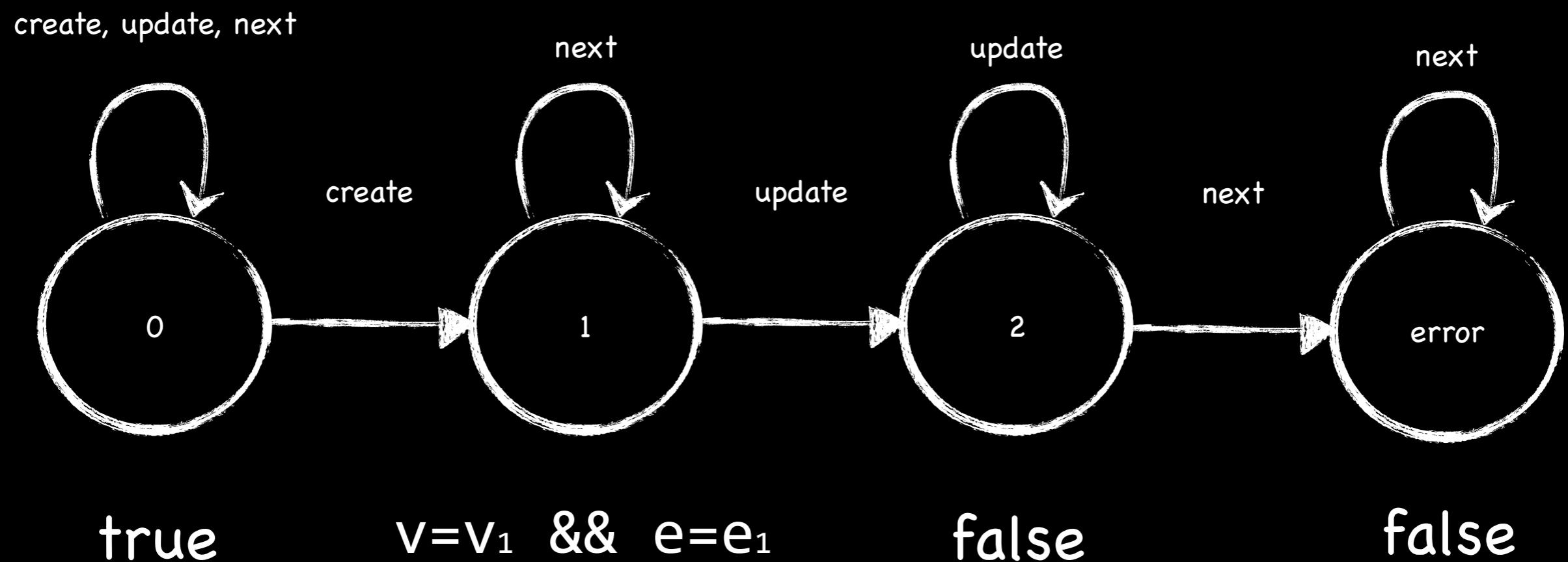


Tracematches - Indexing



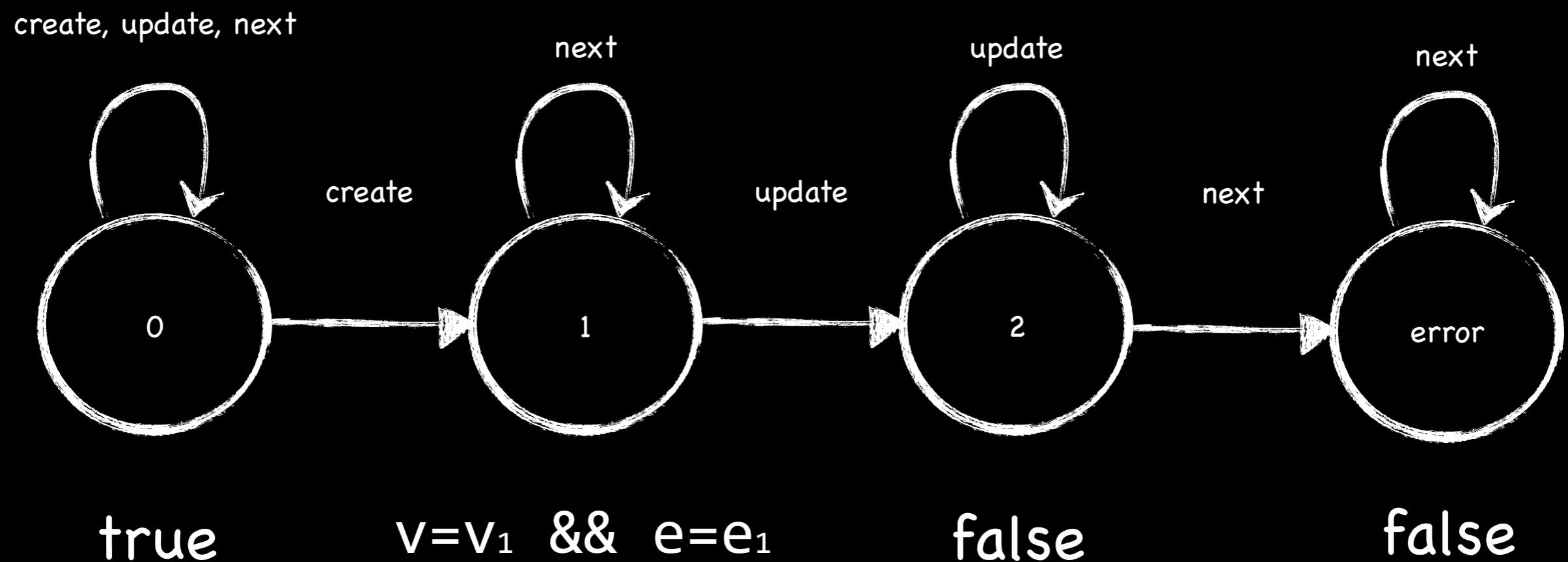
$e_1 = v_1.\text{elements}()$

Tracematches - Indexing



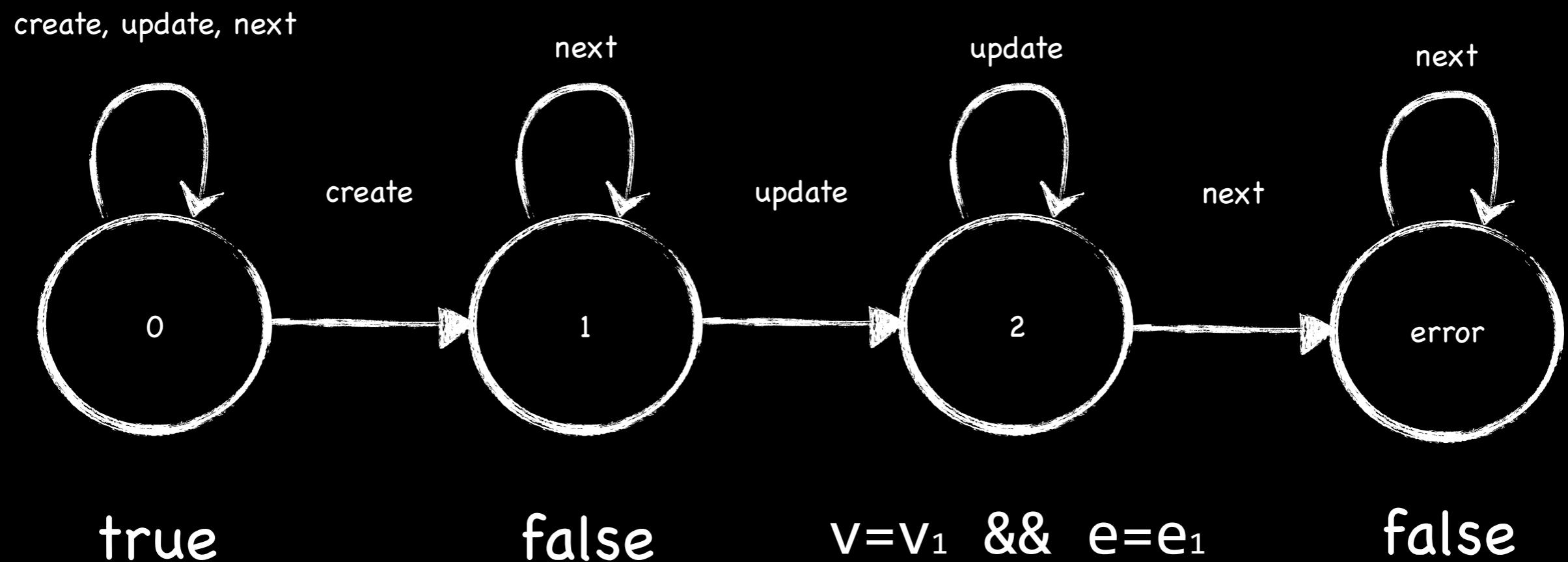
$e_1 = v_1.\text{elements}()$

Tracematches - Indexing



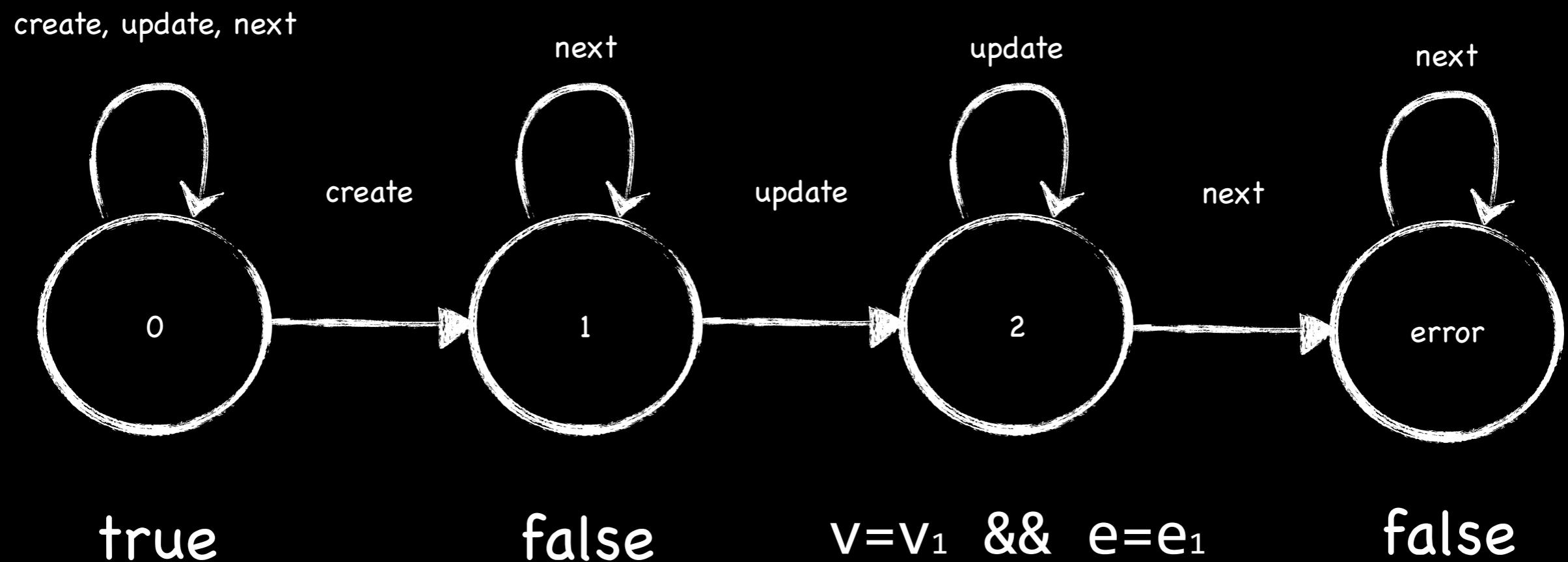
$v_1.add(\dots);$

Tracematches - Indexing



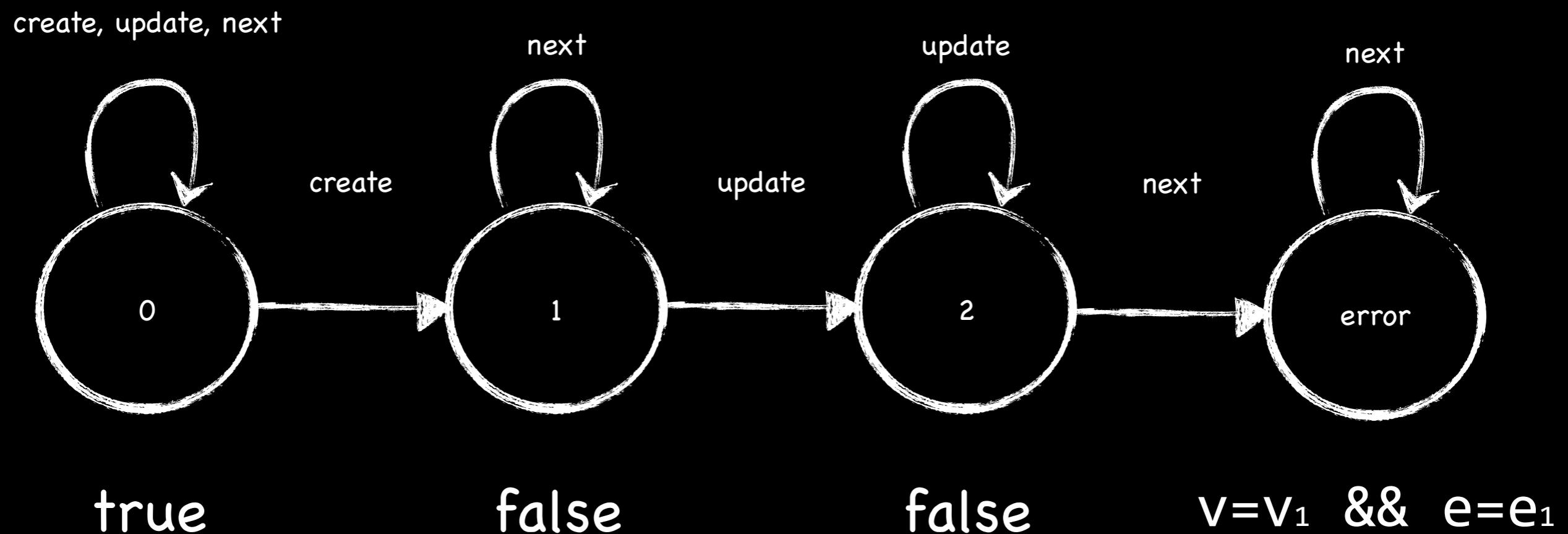
`v1.add(. . .);`

Tracematches - Indexing



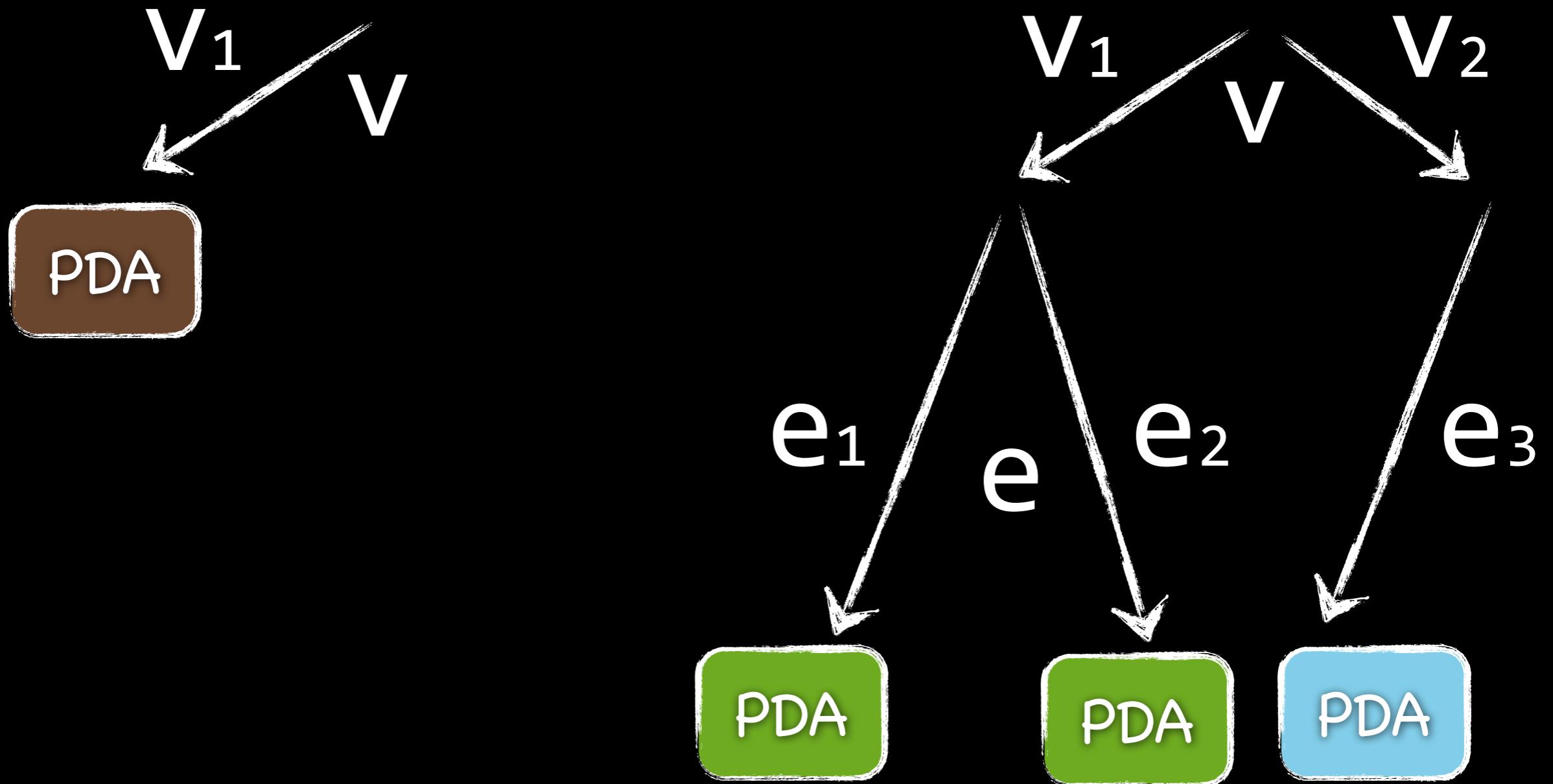
`e1.nextElement()`

Tracematches - Indexing

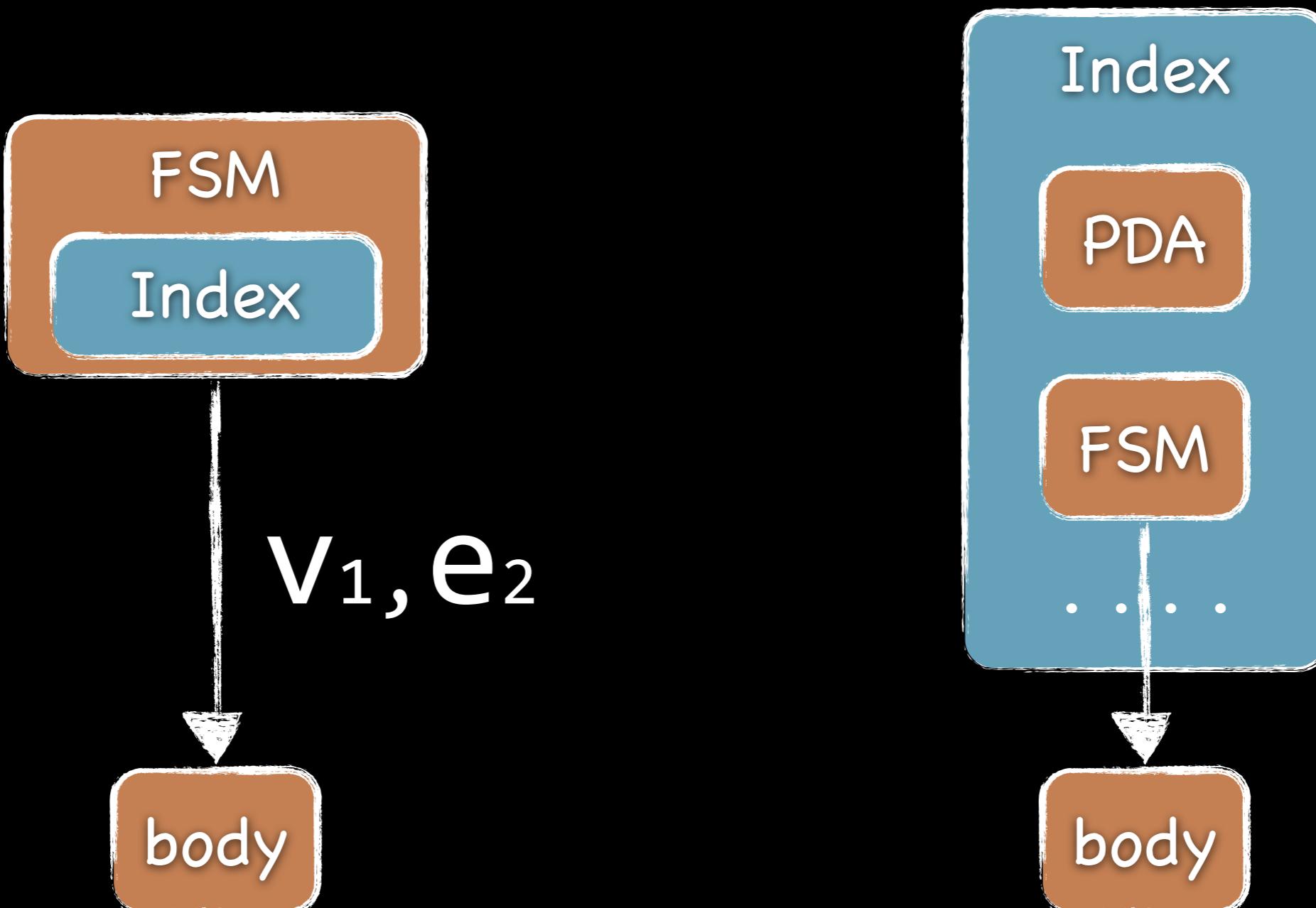


`e1.nextElement()`

JavaMOP - Indexing



Tracematches vs. JavaMOP



Tracematches – Code Gen.

- generates one advice per tracematch symbol
(in the aspect that declares the tracematch)
- generates one indexing tree per state (in the same aspect)

J-LO

This work was done at the [Chair of Computer Science 2, at RWTH Aachen University.](#)

J-LO, the Java Logical Observer A tool for runtime-checking temporal assertions

- [Introduction](#)
- [Requirements](#)
- [System overview](#)
- [Specifying temporal assertions](#)
- [Running J-LO \(with examples\)](#)
- [FAQ](#)
- [Related publications](#)
- [License](#)
- [Download](#)
- [Changelog](#)
- [Documentation](#)
- [Mailing lists](#)

The J-LO thesis is now [available](#).

News

June 5th, 2006

We decided to make J-LO available free via the BSD license. Source code is now available for download.

March 3rd, 2005

Version 0.9.2 is available with a bugfix, an updated version of abc and Soot as well as some smaller new features. See the changelog for details.

November 5th, 2005

Regarding mailing lists, please see [here](#).

November 2nd, 2005

Made [thesis available online](#).

October 28th, 2005

Started implementing the **Tracecheck syntax**:

This syntax is meant not for annotation-based deployment but rather for using J-LO functionality from within AspectJ. Therefore it is similar to the one of [tracematches](#). Formulae can then be written in the following form:

J-LO

- first RV tool to use aspects and at the same time support parameterized specifications
- restricted to LTL
- generates very slow monitors, and has very naive indexing
- monitors too slow in practice
- as JavaMOP/tracematches, generates one advice per event type, plus some other code

S2A

The S2A compiler and the related material below are property of the Weizmann Institute of Science, Rehovot, Israel.

Scenarios in Action

[Home of the S2A compiler](#)

S2A is a compiler that translates modal scenarios, given in a UML2-compliant variant of live sequence charts (LSC) into AspectJ code, and thus provides full code generation of reactive behavior from visual inter-object scenario-based specifications. S2A is based on a compilation scheme presented in S. Maoz and D. Harel, "From Multi-modal Scenarios to Code: Compiling LSCs into AspectJ", Proc. 14th ACM SIGSOFT Symp. on Foundations of Software Engineering (FSE'06), Portland, Oregon, Nov. 2006.

S2A was designed and programmed by Asaf Kleinbort and [Shahar Maoz](#). [Get the latest version below \(April 2009\)](#). Email [Shahar Maoz](#) for more recent updates.
S2A is developed in Prof. [David Harel](#)'s lab, Dept. of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel.

Related resources are available below.

Publications

S. Maoz and D. Harel. "From Multi-Modal Scenarios to Code: Compiling LSCs into AspectJ", In Proc. 14th ACM SIGSOFT Symp. on Foundations of Software Engineering (FSE'06), Portland, Oregon, Nov. 2006, pp. 219-230. [\[PDF\]](#)

D. Harel, A. Kleinbort, and S. Maoz. "S2A: A Compiler for Multi-Modal UML Sequence Diagrams", In Proc. 10th Int. Conf. on Fundamental Approaches to Software Engineering (FASE'07; tools track), LNCS, vol. 4422, Springer-Verlag, 2007, pp. 121-124. [\[PDF\]](#)

Y. Atir, D. Harel, A. Kleinbort, and S. Maoz. "Object Composition in Scenario-Based Programming", In Proc. 11th Int. Conf. on Fundamental Approaches to Software Engineering (FASE'08), LNCS, vol. 4961, Springer-Verlag, 2008, pp. 301-316.

S. Maoz, D. Harel, and A. Kleinbort, "A Compiler for Multi-Modal Scenarios: Transforming LSCs into AspectJ", ACM TOSEM. Extended version of FSE'06 and FASE'07 papers. To appear.

Documentation

[Getting started with S2A \[draft version available\]](#)

Explains how to start working with S2A. In preparation.

[S2A 0.5 beta Installation Guide](#)

System requirements, installation, configuration etc.

Downloads

Build name (date)

0.2 beta (Tue, 3 Oct 2006)

0.4 beta (Mon, 19 Mar 2007)

0.5 beta (Wed, 10 Oct 2007)

0.6 beta (April 2009)

Downloads

[musdc.zip](#); [musdrt.zip](#); [profiles.zip](#)

[musdc.zip](#); [musdrt.zip](#); [profiles.zip](#)

[musdc.zip](#); [musdrt.zip](#); [profiles.zip](#)

[s2a_packaging_01042009.zip](#)

Notes

[release notes](#)

[release notes](#)

[release notes](#)

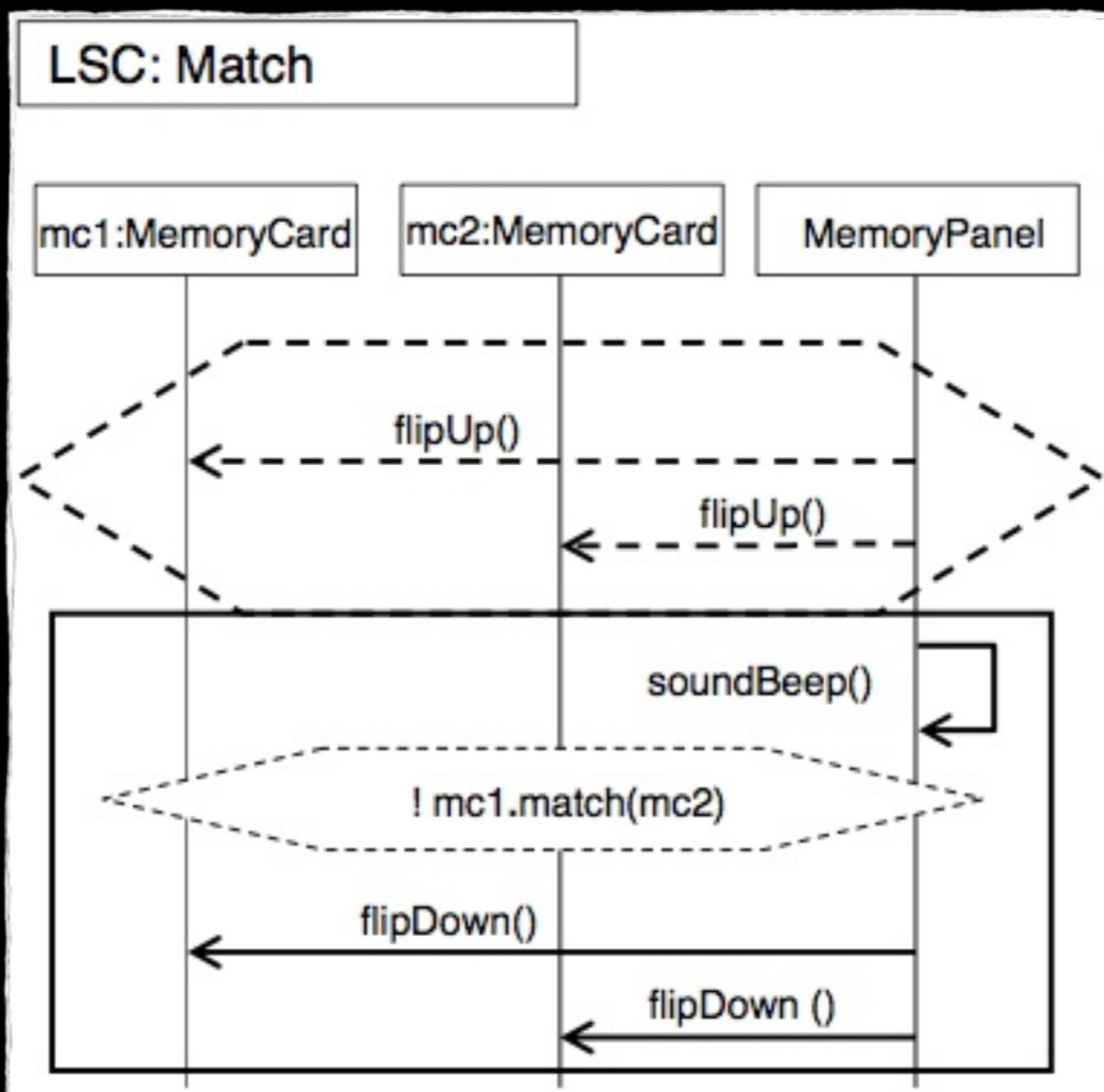
[release notes](#)

Gallery: Scenarios in Action

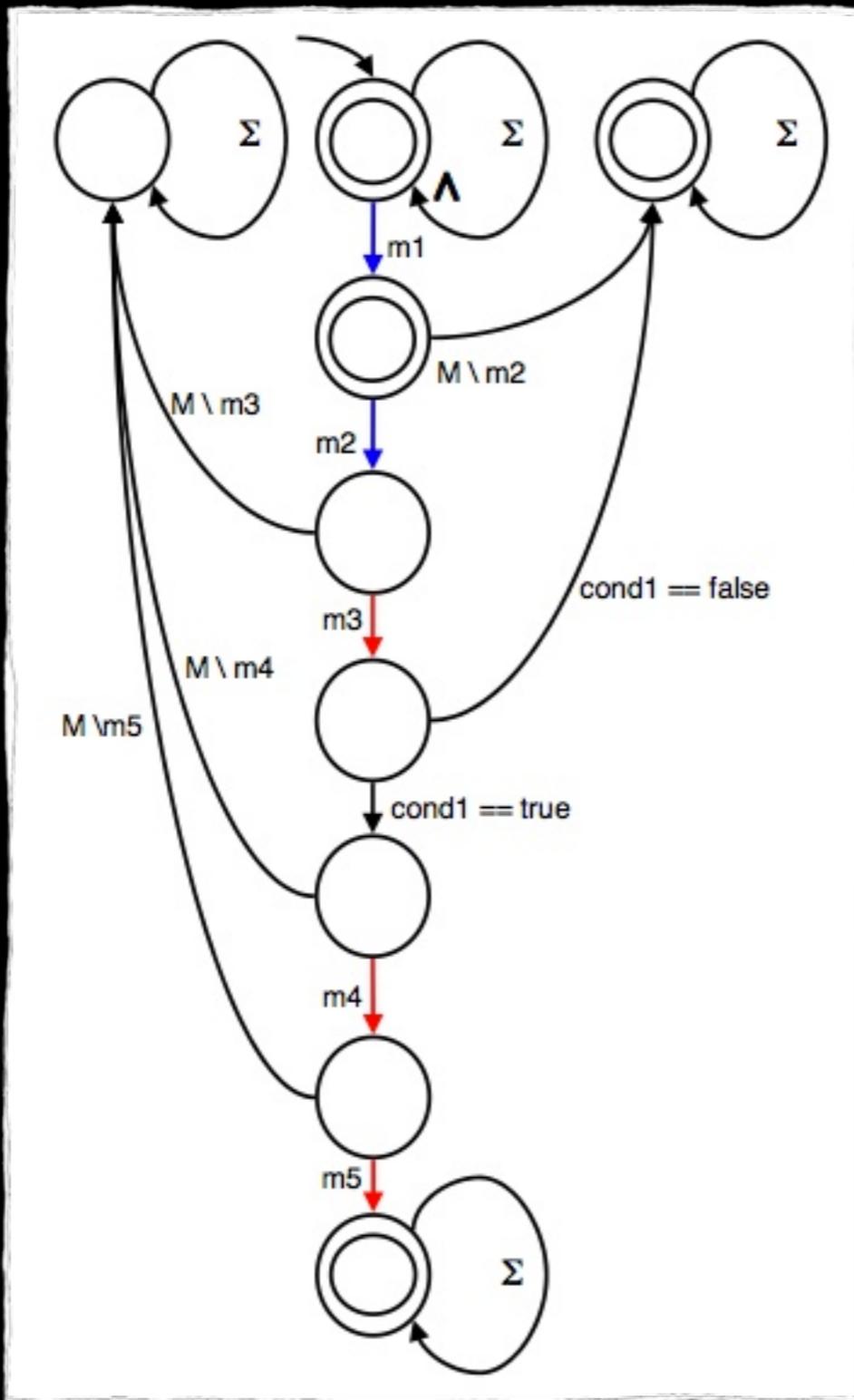
Case studies and applications constructed using S2A.

[Memory Game](#)

S2A - Input: LSCs



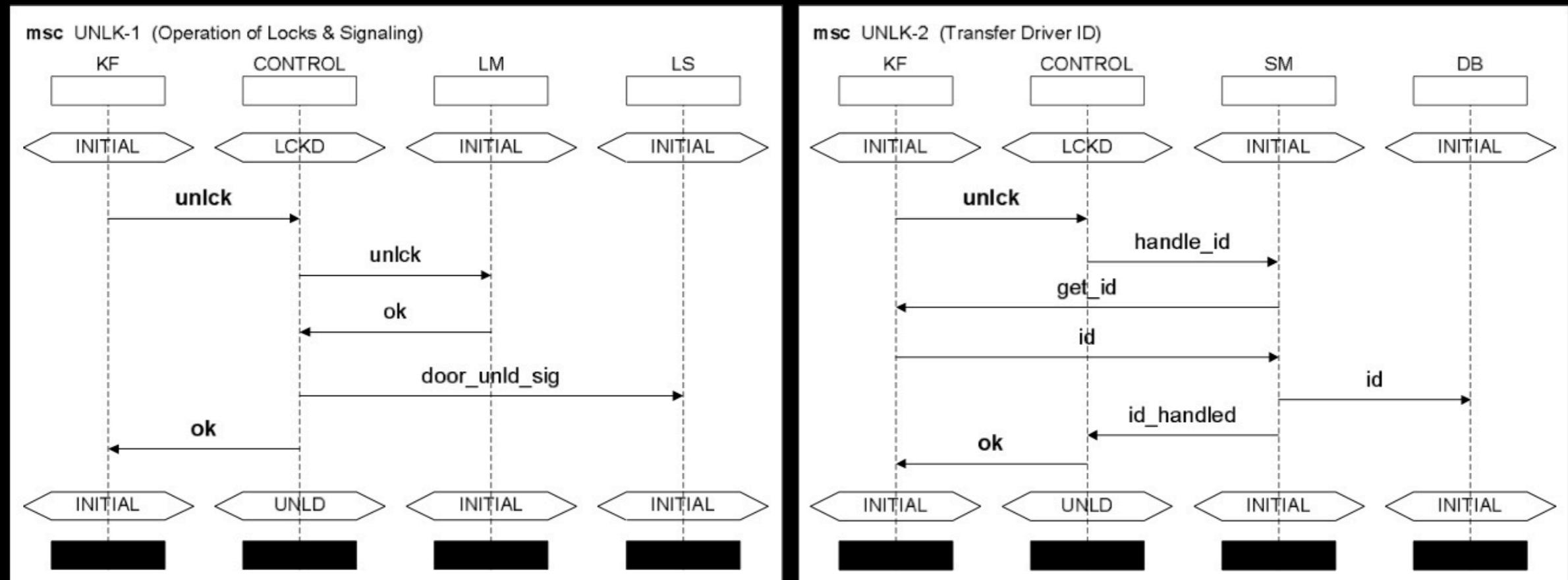
S2A - Internals



S2A – Code Generation

- Single aspect associating “cut states” with objects referenced by life lines
- No monitor class: states are just stored in maps, as integers

M2Aspects



Runtime Verification of Interactions: From MSCs to Aspects
Ingolf H. Krüger, Michael Meisinger and Massimiliano Menarini

RV 2007

M2Aspects – Code Gen.

- uses “M2Code” to generate state-machine classes from each MSC
- generates aspects that dispatch events to instances of these classes

MOFScript

eclipse

Visit other Eclipse Sites

mp microphone magnifying glass speech bubble bucket notepad clipboard

Home Downloads Users Members Committers Resources Projects About Us Google Custom Search Search

MOFScript Home page

Welcome

MOFScript is a tool for model to text transformation, e.g., to support generation of implementation code or documentation from models. It provides a metamodel-independent language that allows to use any kind of metamodel and its instances for text generation. The MOFScript tool is based on EMF and Ecore as metamodel framework.

[more about MOFScript »](#)

Installing MOFScript

You can install MOFScript from Eclipse using the update manager: <http://download.eclipse.org/modeling/gmt/mofscript/update/>

Getting MOFScript Source

You can download mofscript source code using subversion. See the [SVN](#) repository. Use a subversion client (e.g. [Subversive](#) for Eclipse, [TortoiseSVN](#) for Windows, or similar), use repository <http://dev.eclipse.org/svnroot/modelling/org.eclipse.gmt.mofscript> and check out the MOFScript projects.

Getting Started

- [Project description](#)
- [Documentation](#)

News

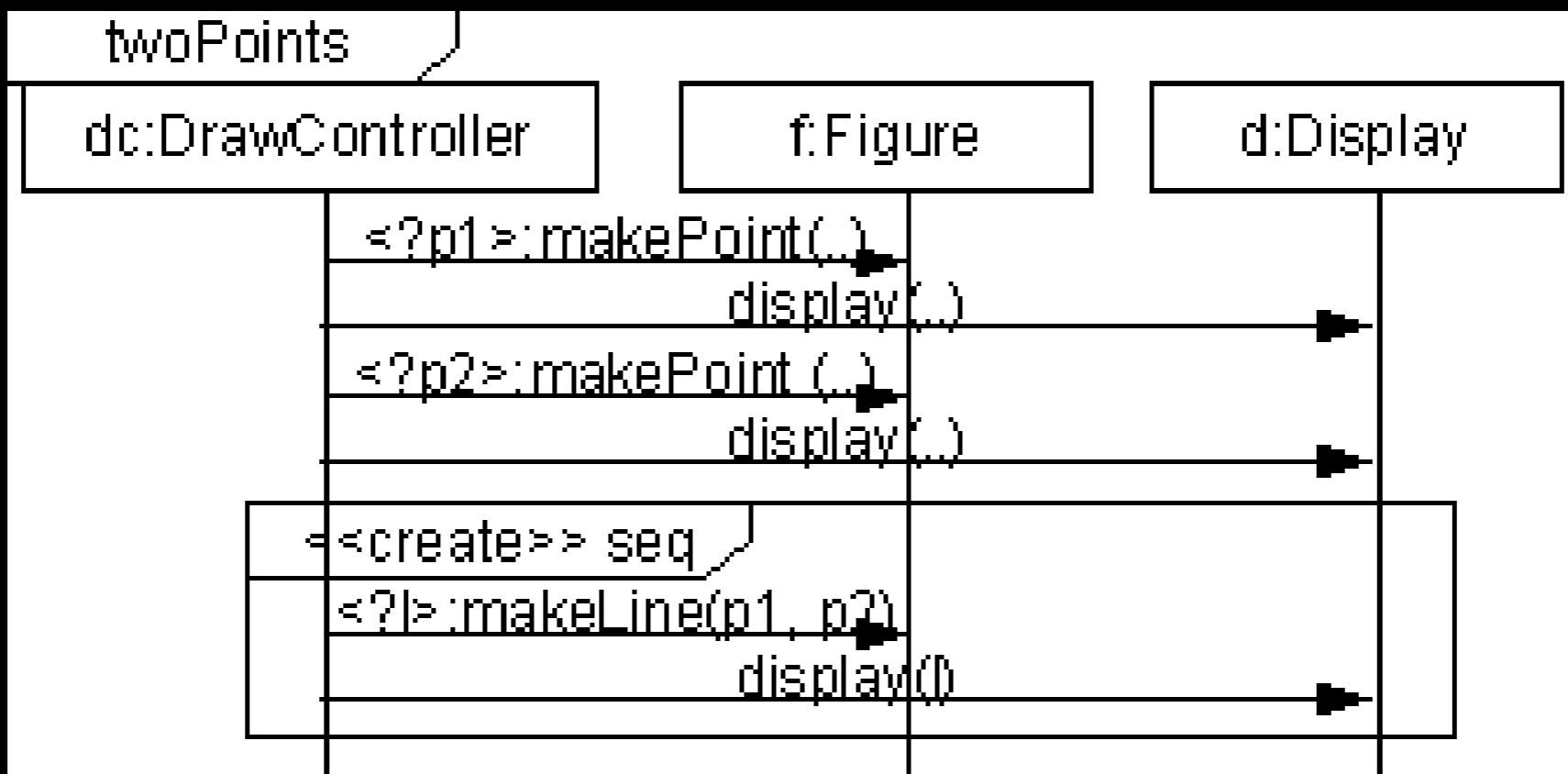
MOFScript News [RSS 2.0](#)

- [MOFScript 1.3.8 is available for download on update site](#) posted 19-04-2010
- [MOFScript 1.3.6 is available for download on update site](#) posted 16-12-2009
- [MOFScript 1.3.5 is available for download on update site](#) posted 20-10-2009
- [MOFScript 1.3.4 is available for download on update site](#) posted 09-03-2009
- [MOFScript sources available on SVN](#) posted 12-01-2009

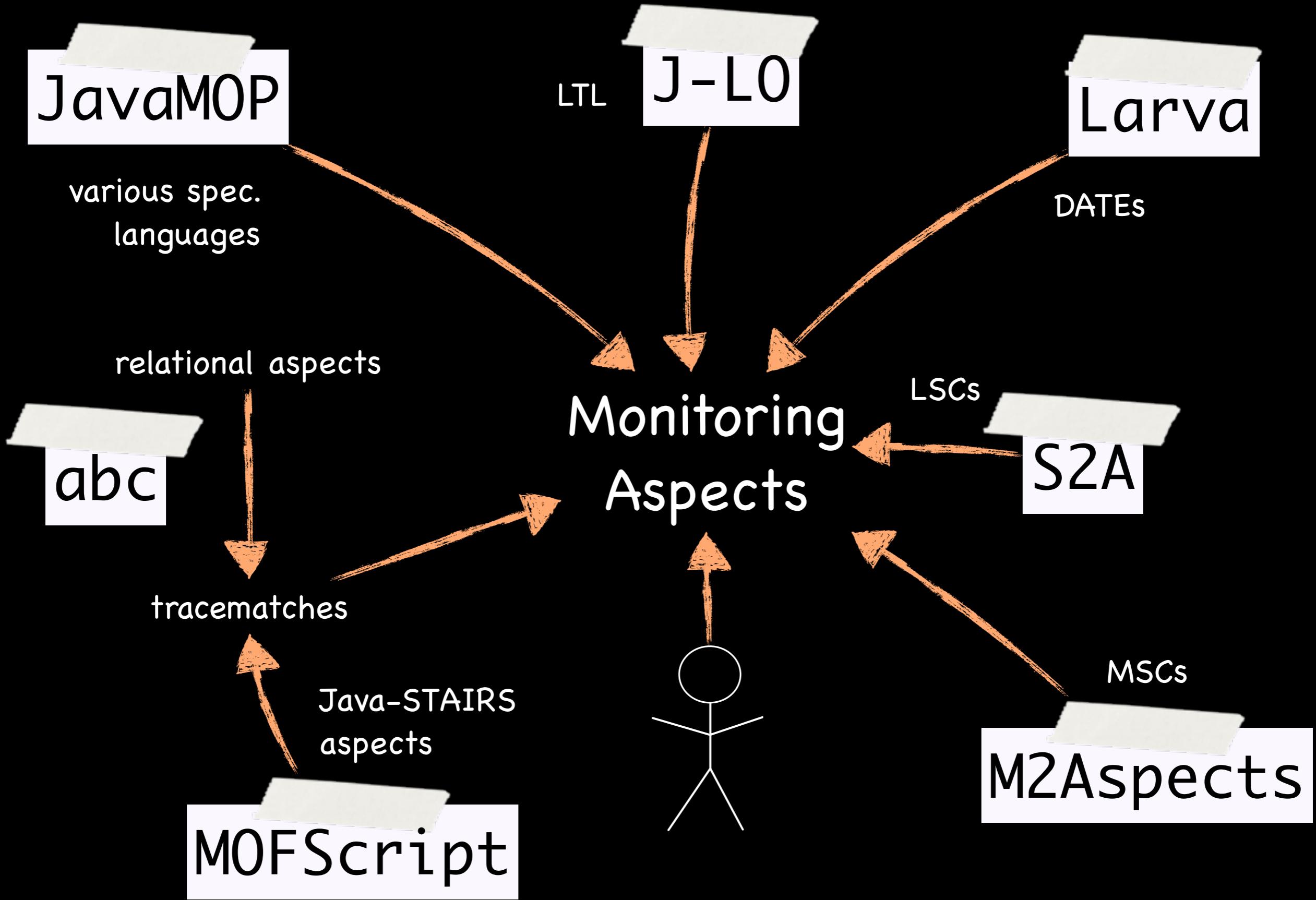
Home Privacy Policy Terms of Use Copyright Agent Legal Contact Us Copyright © 2010 The Eclipse Foundation. All Rights Reserved.

<http://www.eclipse.org/gmt/mofscript/>

JavaSTAIRS aspects



converted into tracematch



Clara's design and
architecture

RV tools - Differences

- support different input formalisms
 - LTL, ERE, CFG, LSC, MSC, ...
- use different monitor implementations
 - FSM, PDA, you name it...
- use different indexing implementations

RV tools - Commonalities

- generate AspectJ aspects
- generate advice for events
- pointcut of piece of advice is given through input specification

Example: ConnectionClosed

```
Set closed = new IdentityHashSet();
```

```
after(Connection c) returning:  
    call(* Connection.close()) && target(c) {  
        closed.add(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.reconnect()) && target(c) {  
        closed.remove(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.write(..)) && target(c) {  
        if(closed.contains(c))  
            error("May not write to "+c+", as it is closed!");  
    }
```

Example: ConnectionClosed

```
Set closed = new IdentityHashSet();
```

```
after(Connection c) returning:  
    call(* Connection.close()) && target(c) {  
        closed.add(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.reconnect()) && target(c) {  
        closed.remove(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.write(..)) && target(c) {  
        if(closed.contains(c))  
            error("May not write to "+c+", as it is closed!");  
    }
```

Indexing/monitor code different for different formalisms/tools

Example: ConnectionClosed

```
Set closed = new IdentityHashSet();
```

```
after(Connection c) returning:  
    call(* Connection.close()) && target(c) {  
        closed.add(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.reconnect()) && target(c) {  
        closed.remove(c);  
    }
```

```
after(Connection c) returning:  
    call(* Connection.write(..)) && target(c) {  
        if(closed.contains(c))  
            error("May not write to "+c+", as it is closed!");  
    }
```

Advice headers always the same

Dependency State Machines (DSMs)

```
Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }
```

Dependency State Machines (DSMs)

```
Set closed = new WeakIdentityHashSet();
```

```
dependent after disconnect(Connection c) returning:  
    call(* Connection.close()) && target(c) {  
        closed.add(c);  
    }
```

```
dependent after reconnect(Connection c) returning:  
    call(* Connection.reconnect()) && target(c) {  
        closed.remove(c);  
    }
```

```
dependent after write(Connection c) returning:  
    call(* Connection.write(..)) && target(c) {  
        if(closed.contains(c))  
            error("May not write to "+c+", as it is closed!");  
    }
```

abstract

concrete

```

Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

dependent after write(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }

dependency{
    disconnect, write, reconnect;
initial   connected: disconnect -> connected,
          write -> connected,
          reconnect -> connected,
          disconnect -> disconnected;
disconnect: disconnect -> disconnected,
          write -> error;
final     error: write -> error;
}

```

finite_state_property

```
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

dependent after write(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }

dependency{
    disconnect, write, reconnect;
    initial   connected: disconnect -> connected,
                write -> connected,
                reconnect -> connected,
                disconnect -> disconnected;
    disconnected: disconnect -> disconnected,
                  write -> error;
    final      error: write -> error;
}
```

Clara only uses
this information
during static
analysis

no advice bodies!

for all tools
the same!

```
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
call(* Connection.close()) && target(c) {
closed.add(c);
}

dependent after reconnect(Connection c) returning:
call(* Connection.reconnect()) && target(c) {
closed.remove(c);
}

dependent after write(Connection c) returning:
call(* Connection.write(..)) && target(c) {
if(closed.contains(c))
    error("May not write to "+c+", as it is closed!");
}

dependency{
    disconnect, write, reconnect;
initial    connected: disconnect -> connected,
            write -> connected,
            reconnect -> connected,
            disconnect -> disconnected;
disconnected: disconnect -> disconnected,
            write -> error;
final      error: write -> error;
}
```

Recap

- Want to partially evaluate monitoring aspects at compile time
- General monitoring aspects can have any shape
- This makes it impossible for the static analysis to determine the aspect's internal transition structure
- But we need to know this structure to conduct an effective partial evaluation
- Solution: Use explicit annotations with Dependency State Machines (tools that generate aspects can generate these annotations too)

Semantics of Dependency State Machines

CLARA also supports Collaborative Runtime Verification, which distributes instrumentation overhead among multiple users; and ranking heuristics, which aid programmers in inspecting remaining instrumentation manually [13] [2, Ch. 6 & 7]. Space limitations preclude us from discussing ranking and Collaborative Runtime Verification here.

CLARA is freely available as free software at <http://bodden.de/clara/>, along with extensive documentation, the first author's dissertation [2], which describes CLARA in detail, and benchmarks and benchmark results.

We next describe the syntax and semantics of Dependency State Machines, the key abstraction of CLARA. This abstraction allows CLARA to decouple runtime monitor implementations from static analyses.

3 Syntax and Semantics of Dependency State Machines

Dependency State Machines extend the AspectJ language to include semantic information about relationships between different pieces of advice. Runtime verification tools which generate AspectJ aspects can use this extension to produce augmented aspects. CLARA can reason about the augmented aspects to prove that programs never violate monitored properties or to generate optimized code.

3.1 Syntax

Our extensions modify the AspectJ grammar in two ways: they add syntax for defining Dependent Advice [14] and Dependency State Machines. The idea of Dependent Advice is that pieces of monitoring advice are often inter-dependent in the sense that the execution of one piece of advice only has an effect when executing before or after another piece of advice, on the same objects. Dependency State Machines allow programmers to make these dependencies explicit so that static analyses can exploit them. Our explanations below refer to the ConnectionClosed example in Figure 2.

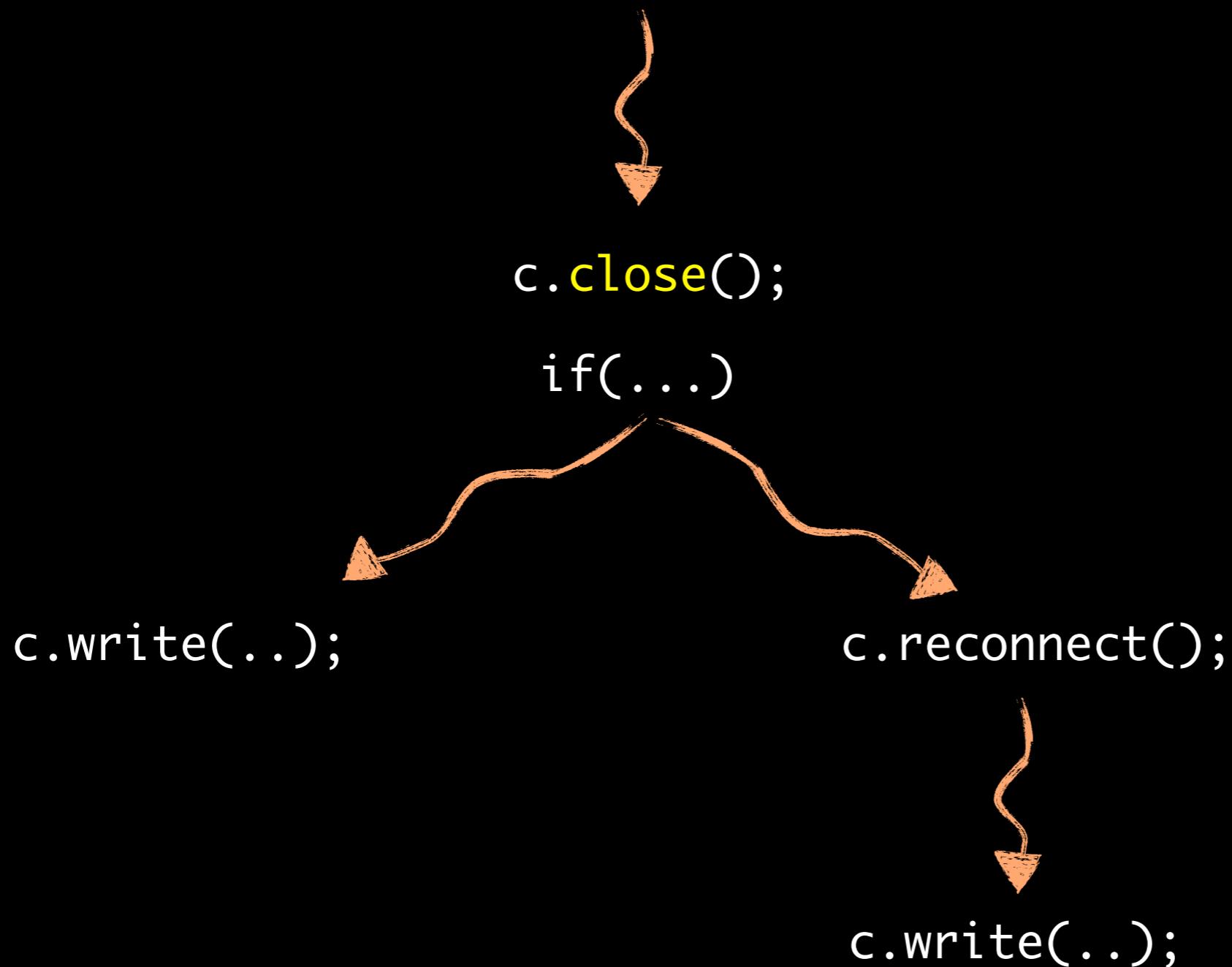
The `dependent` modifier flags advice to CLARA for potential optimization; such advice may be omitted from program locations at which it provably has no effect on the state of the runtime monitor. Dependent advice must be named. Lines 4, 7 and 10 all define dependent advice.

The Dependency State Machines extension enables users to specify state machines which relate different pieces of dependent advice. Dependency State Machine declarations define state machines by including a list of edges between states and an alphabet; each edge is labelled with a member of the alphabet. CLARA infers the set of states from the declared edges. Line 16 declares the state machine's alphabet: `{disconn, write, reconn}`. Every symbol in the alphabet references dependent advice from the same aspect. Lines 17–19 enumerate, for each state, a (potentially empty) list of outgoing transitions. An entry "`s1: t -> s2`" means "there exists a `t`-transition from `s1` to `s2`". Users can also mark states as `initial` or `final` (error states). Final states denote states

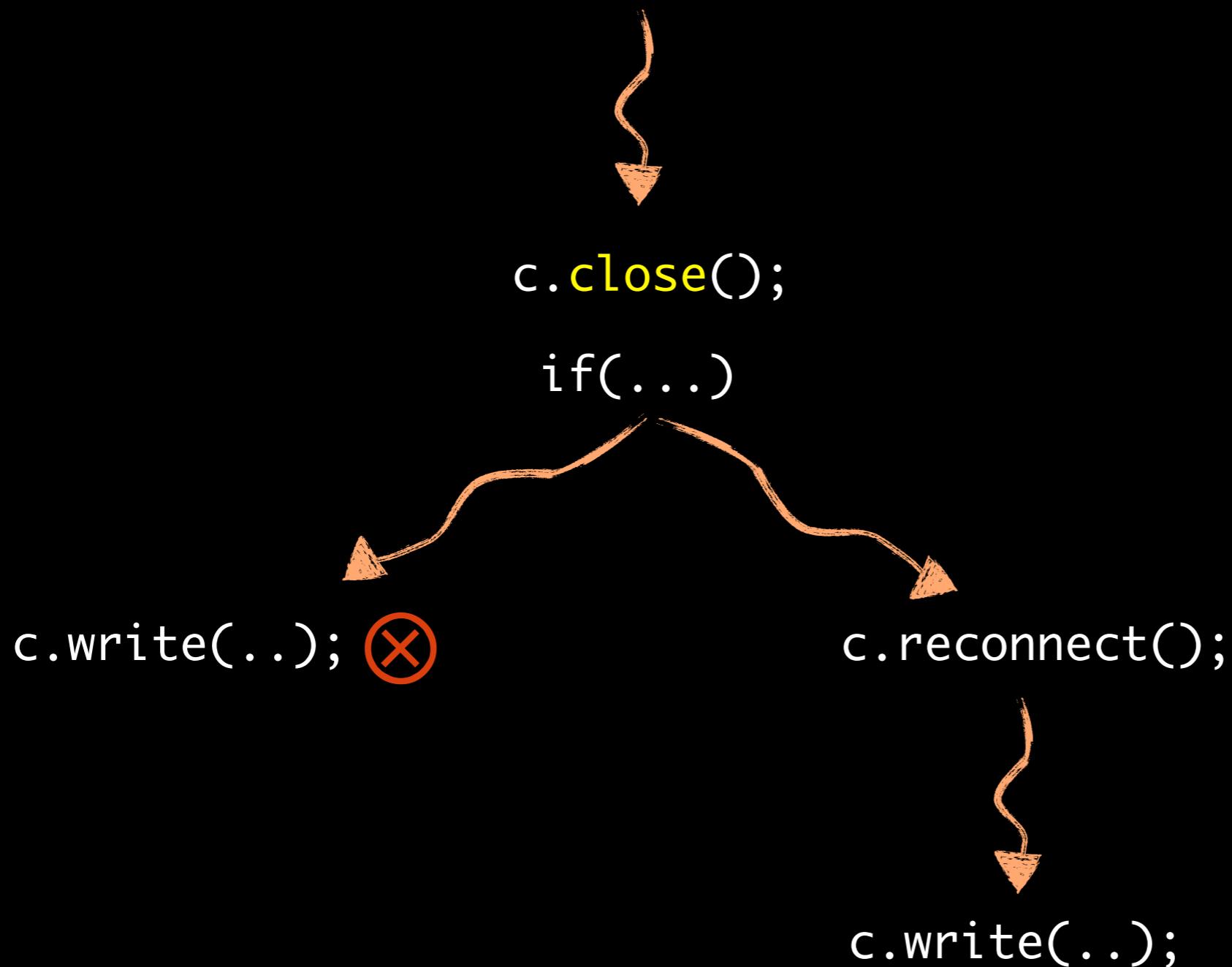
Research-track
Paper

Pages 74–88
of proceedings

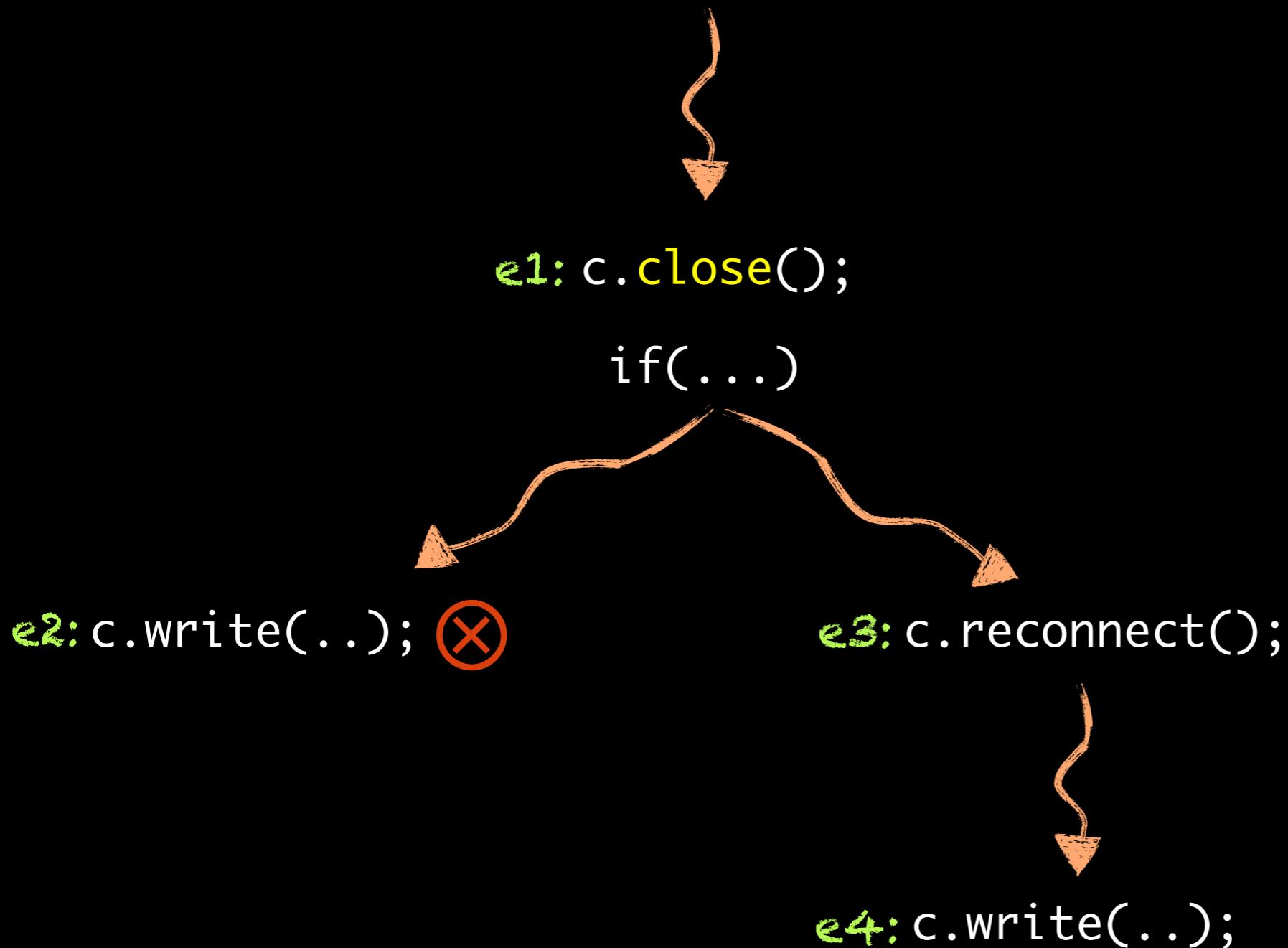
Semantics of Dependency State Machines



Semantics of Dependency State Machines

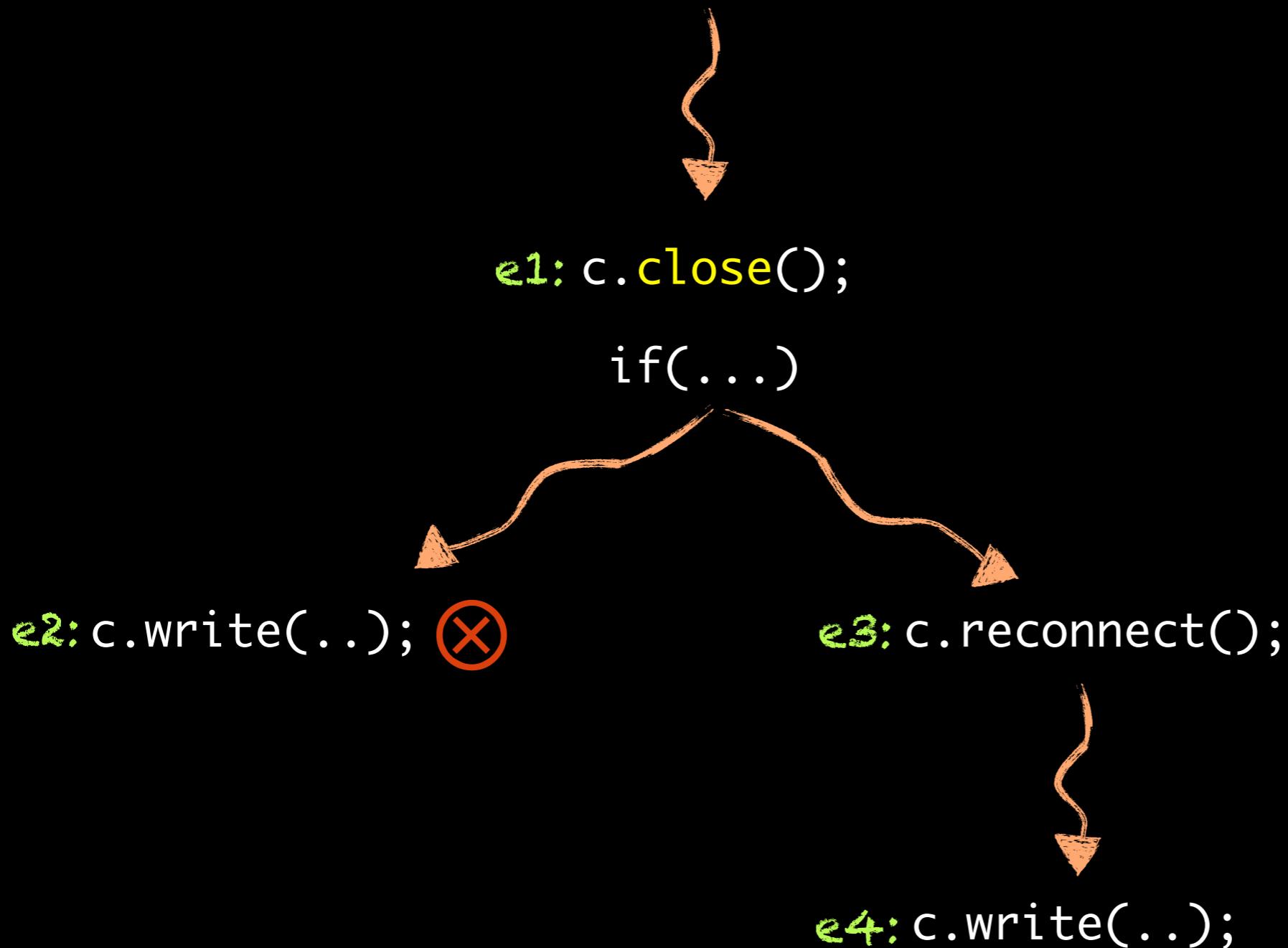


Semantics of Dependency State Machines



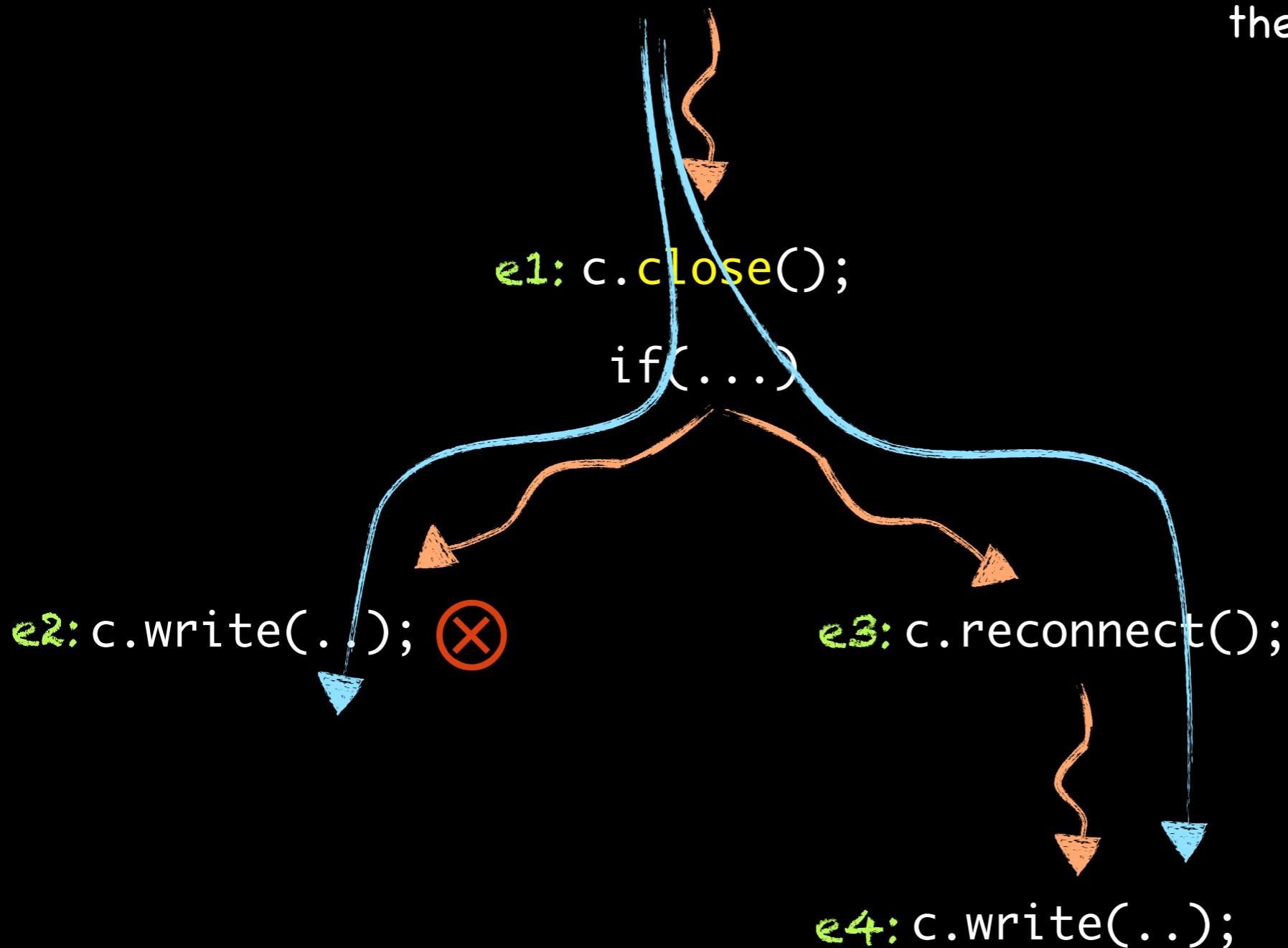
Semantics of Dependency State Machines

advice "close" must
execute at `e1` if



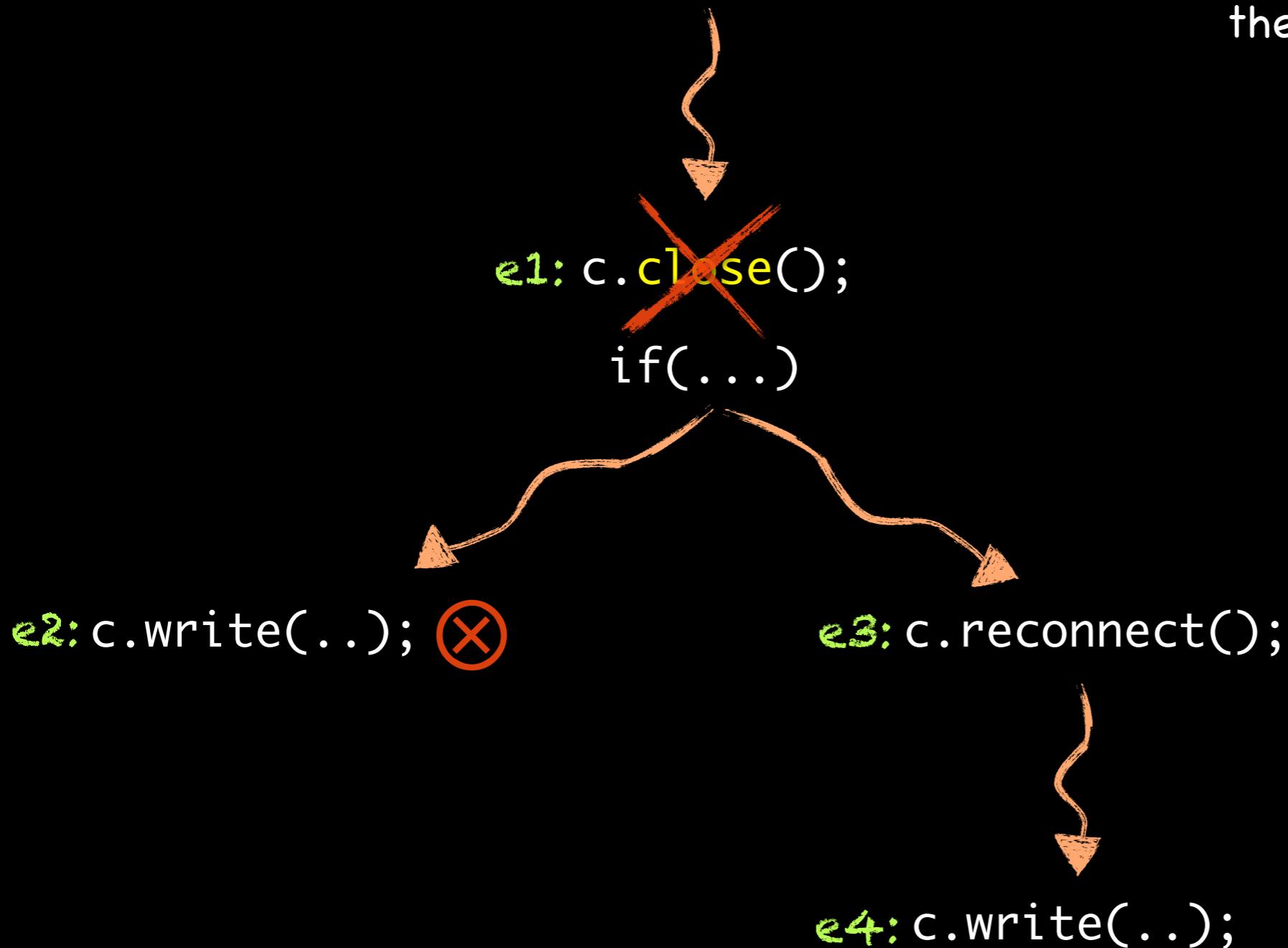
Semantics of Dependency State Machines

advice "close" must
execute at `e1` if
there exists an execution

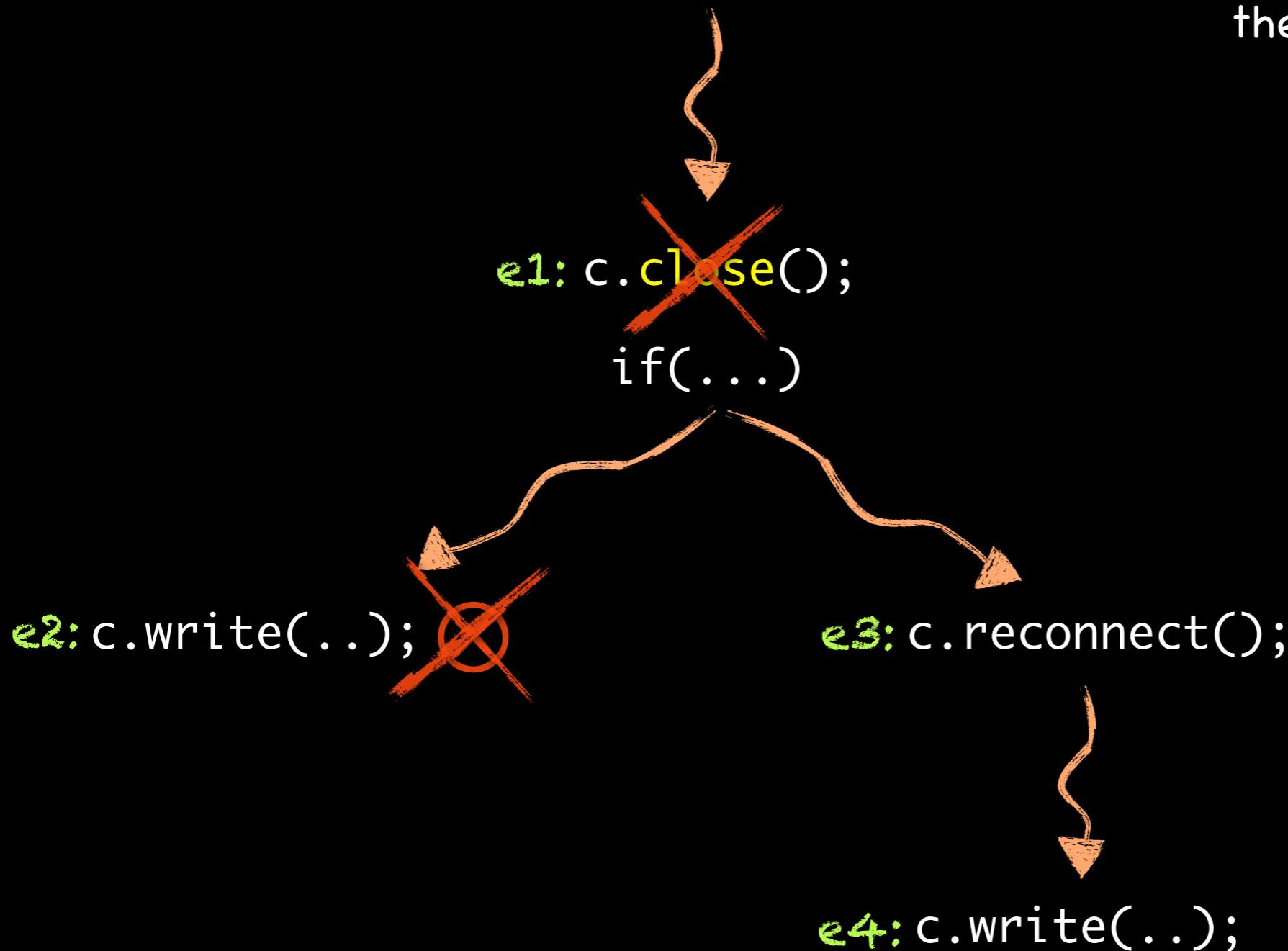


Semantics of Dependency State Machines

advice “close” must
execute at **e1** if
there exists an execution
such that omitting
“close” at **e1**

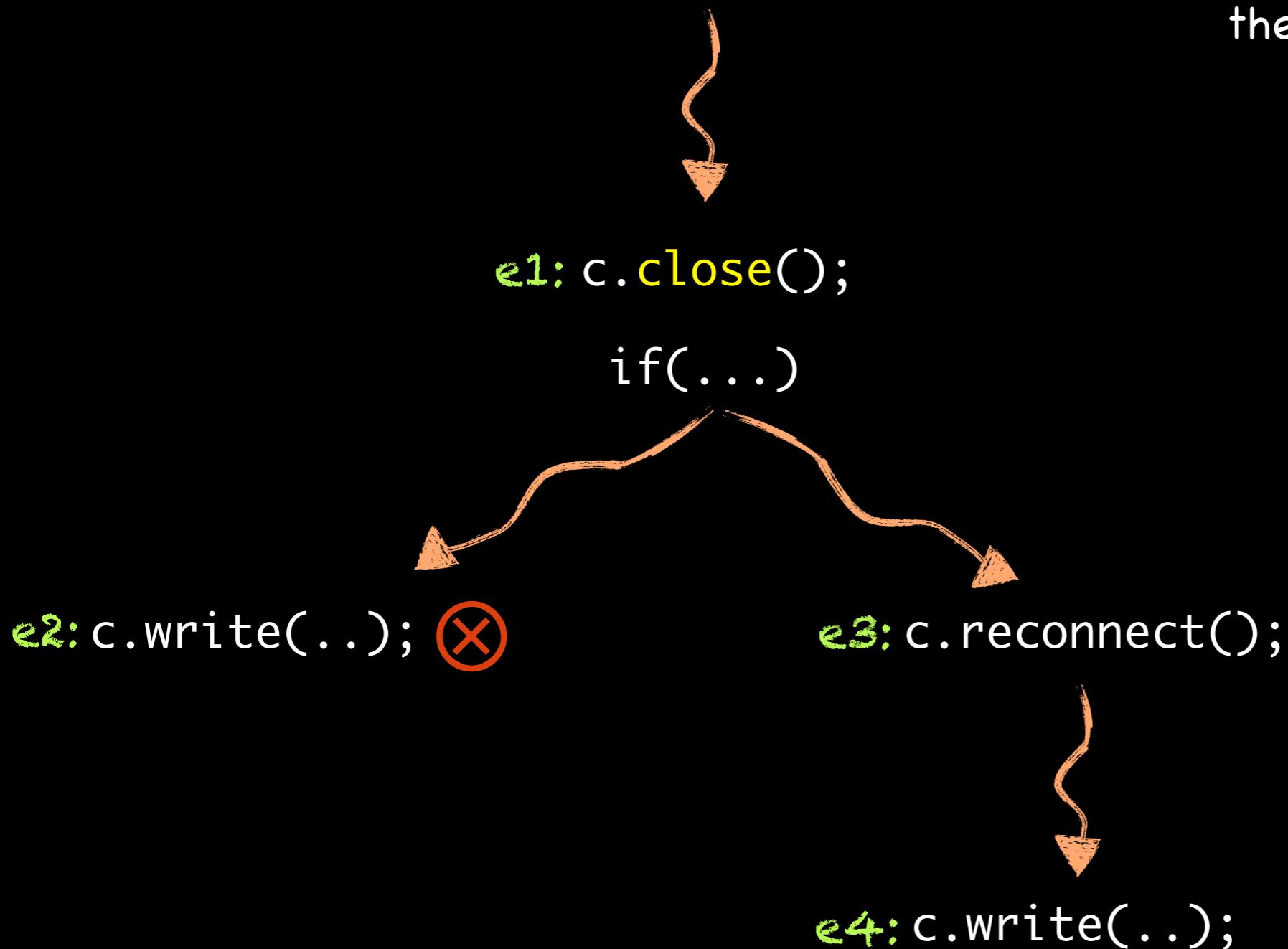


Semantics of Dependency State Machines



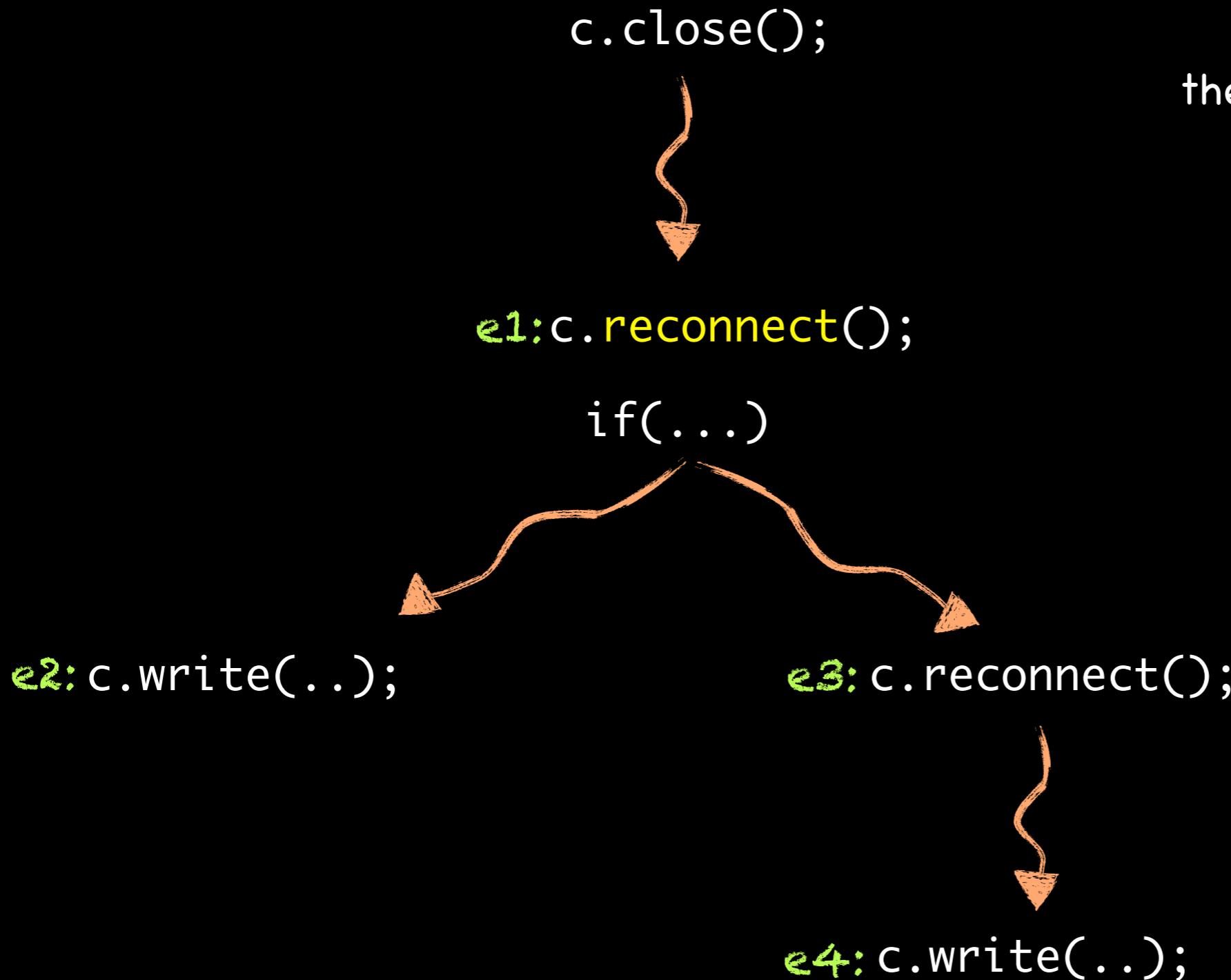
advice "close" must
execute at **e1** if
there exists an execution
such that omitting
"close" at **e1**
would change the
events at which
a DSM reaches an
accepting state

Semantics of Dependency State Machines



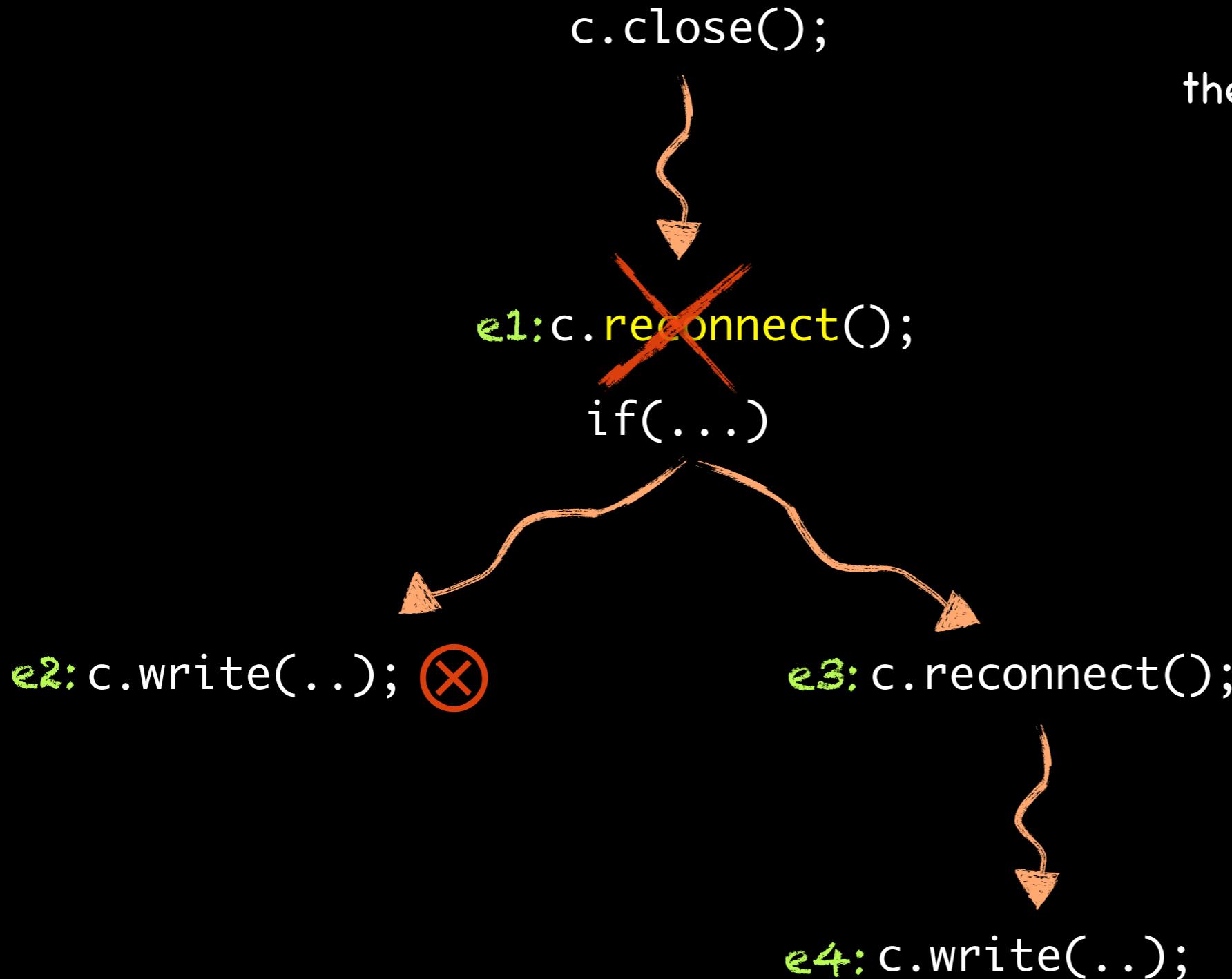
advice “close” must execute at `e1` if there exists an execution such that omitting “close” at `e1` would change the events at which a DSM reaches an accepting state

Inverse case: match-preventing events



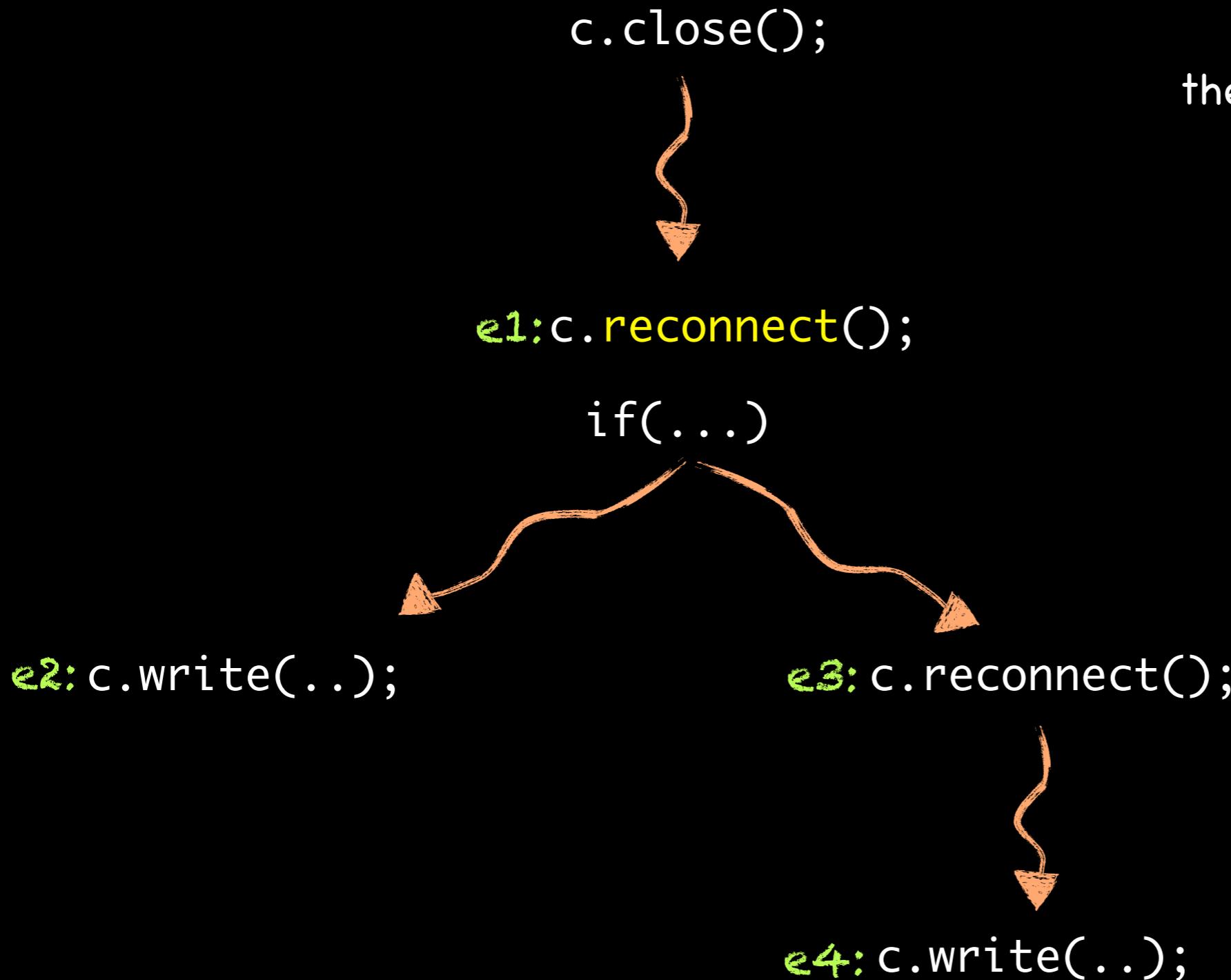
advice “reconnect” must execute at `e1` if there exists an execution such that omitting “reconnect” at `e1` would change the events at which a DSM reaches an accepting state

Inverse case: match-preventing events



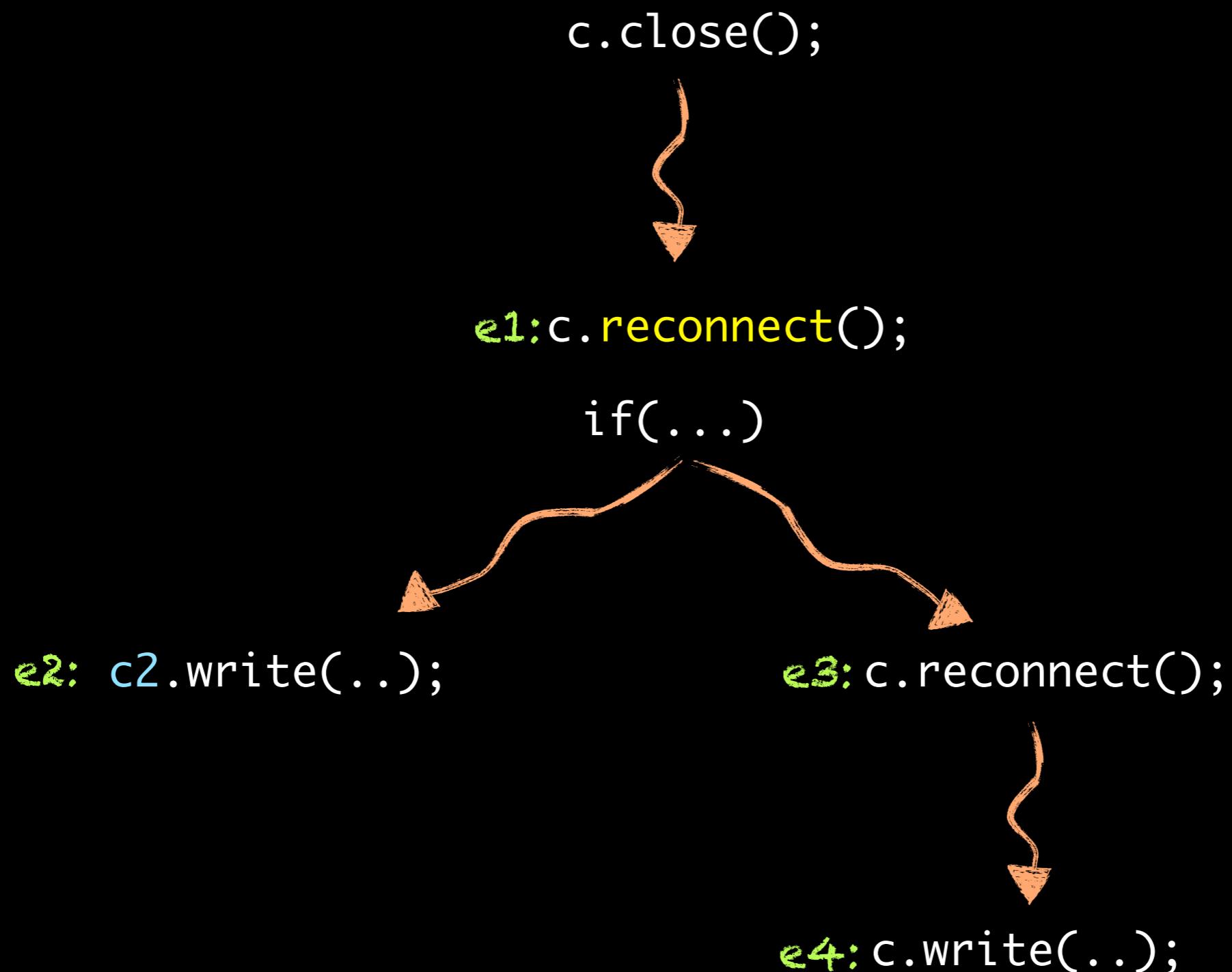
advice “reconnect” must execute at **e1** if there exists an execution such that omitting “reconnect” at **e1** would change the events at which a DSM reaches an accepting state

Inverse case: match-preventing events



advice “reconnect” must execute at `e1` if there exists an execution such that omitting “reconnect” at `e1` would change the events at which a DSM reaches an accepting state

Variable bindings matter



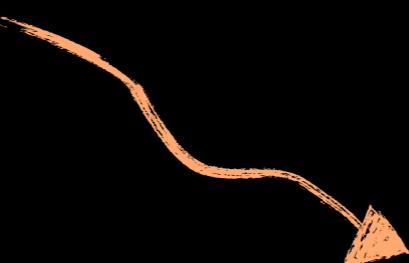
Variable bindings matter

```
c.close();
```



```
e1:c.reconnect();
```

```
if(...)
```

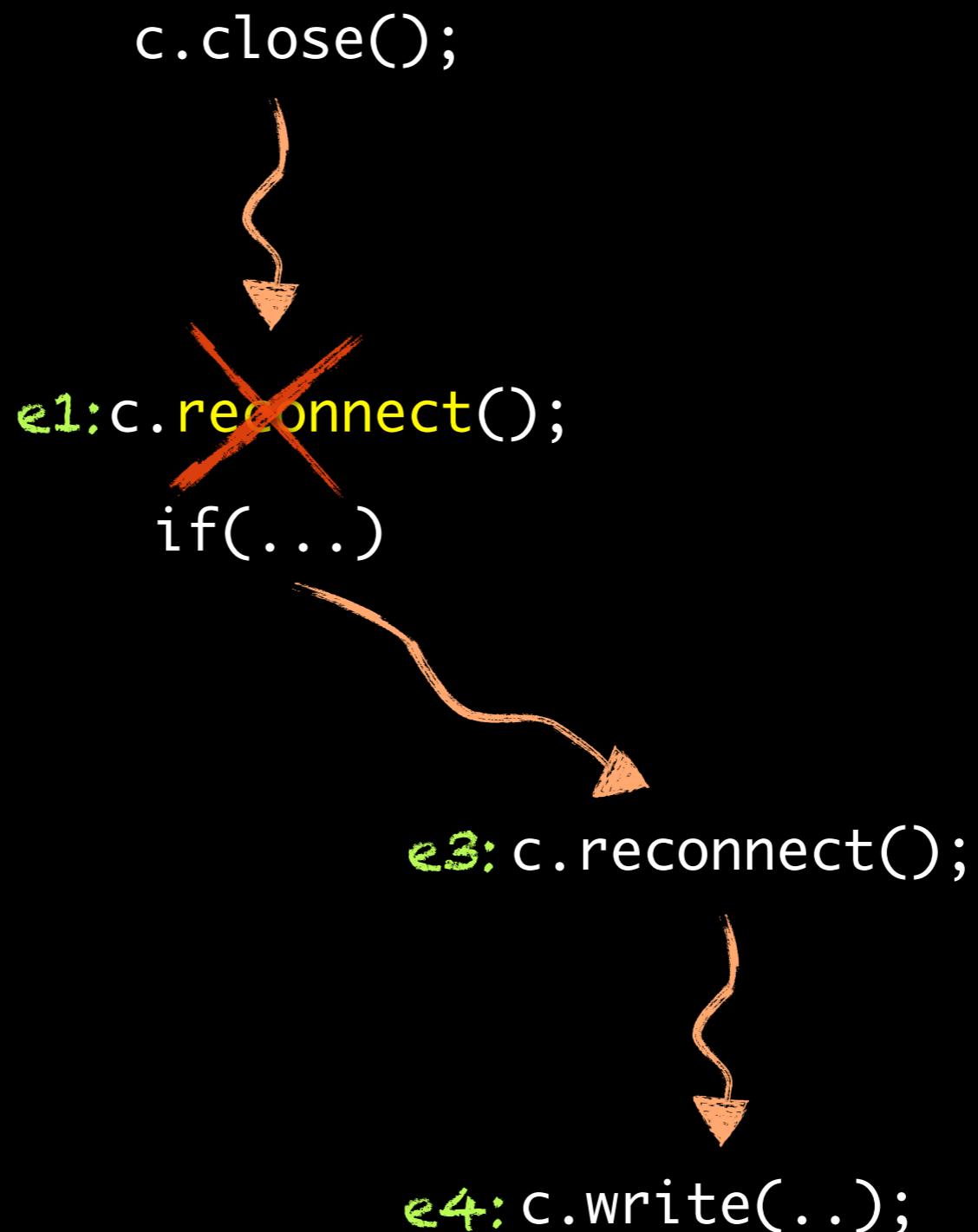


```
e3:c.reconnect();
```



```
e4:c.write(..);
```

Variable bindings matter

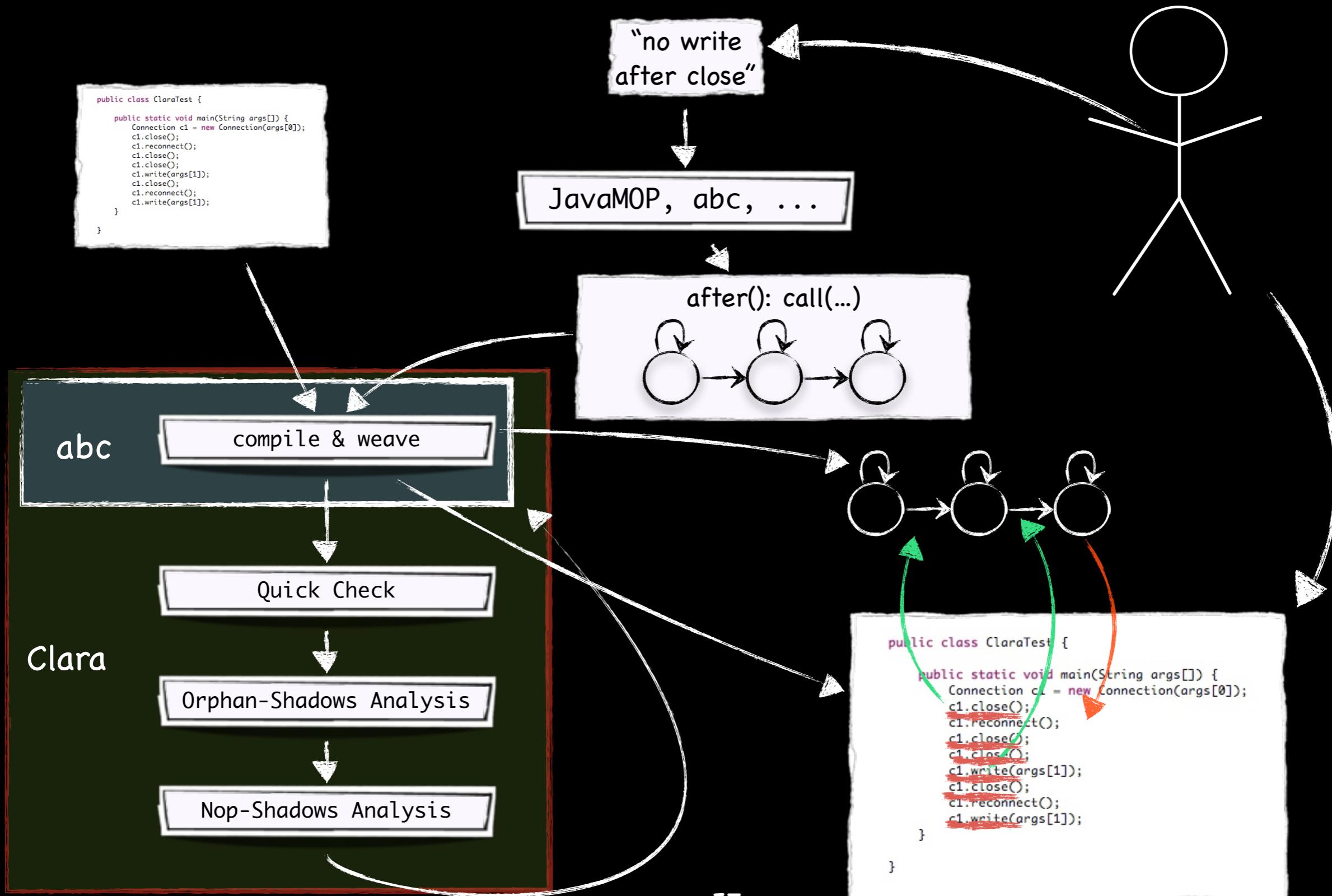


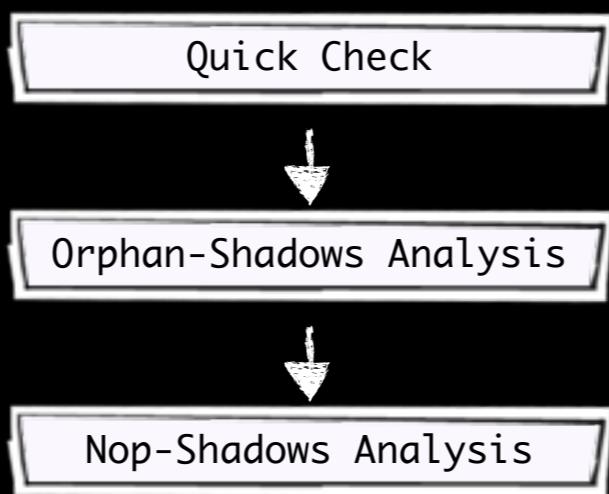
Note

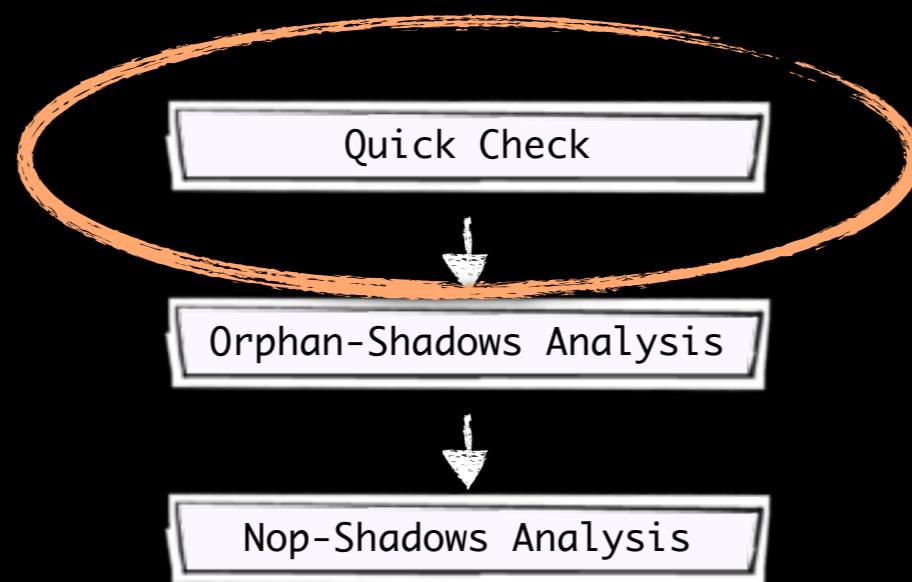
abc generates DSMs automatically from
the tracematch's regular expression
(internally, i.e., you never see the
annotations in the code)

Clara's three
pre-defined
static analyses

The Clara Framework



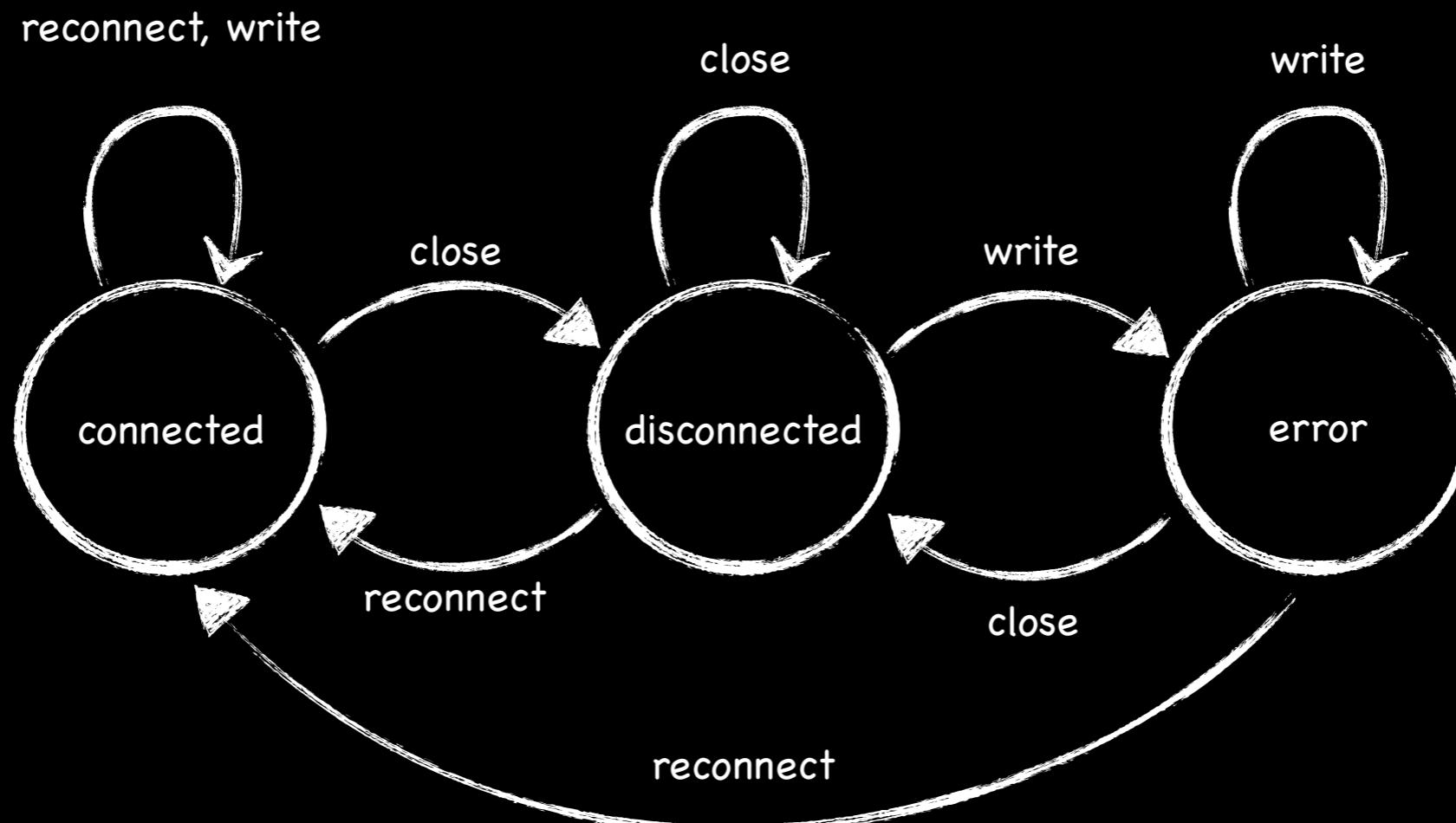




Quick Check

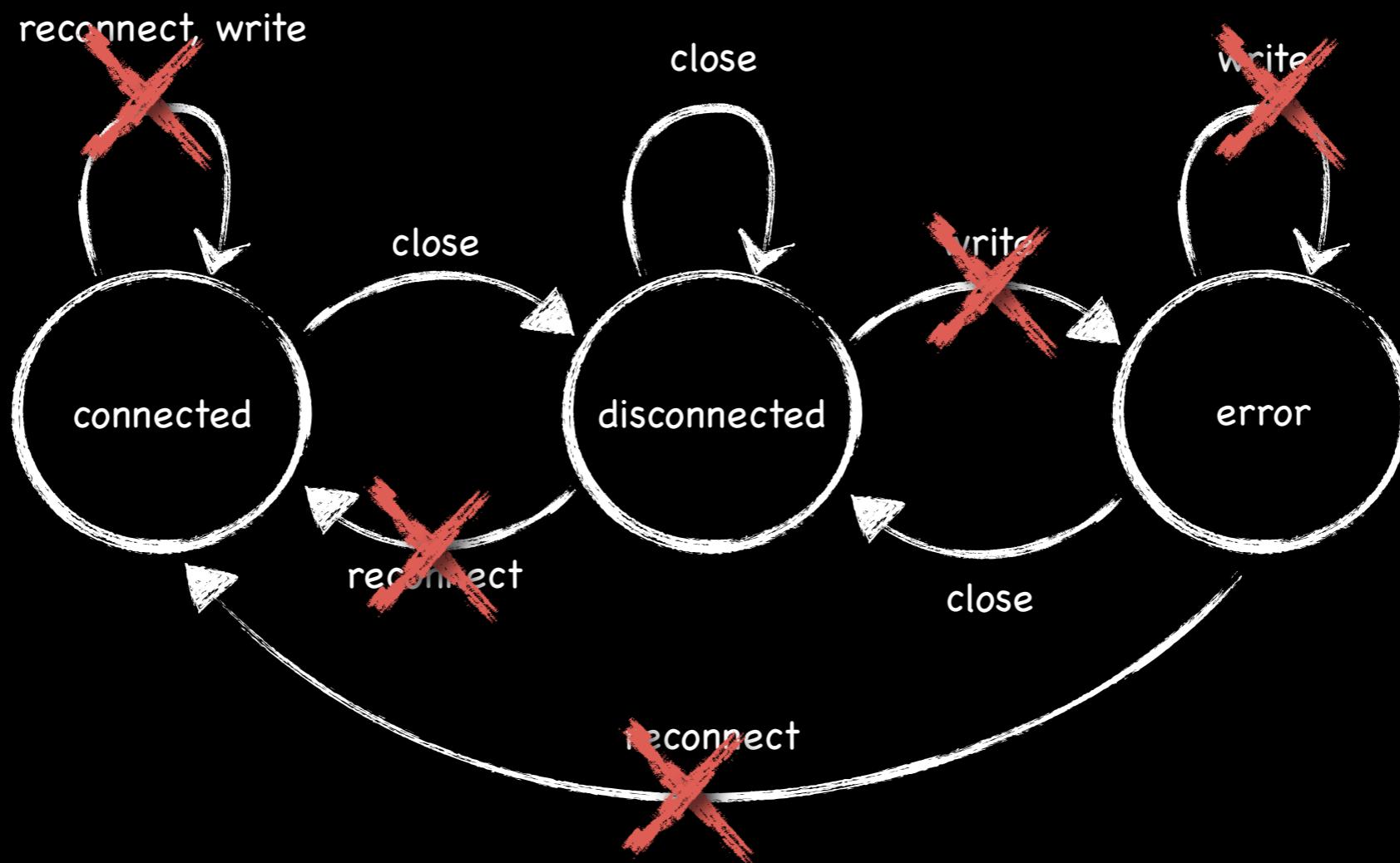
```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
}
```

Quick Check



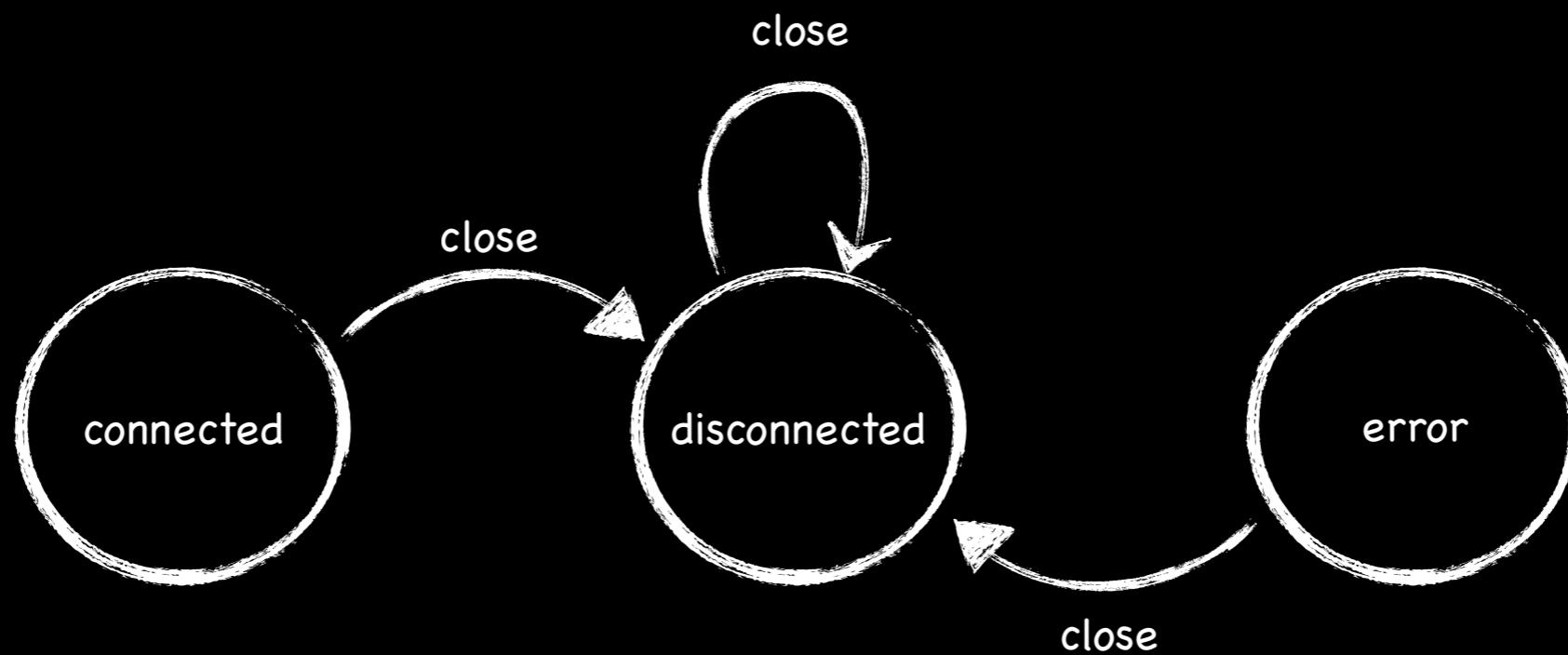
```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
}
```

Quick Check



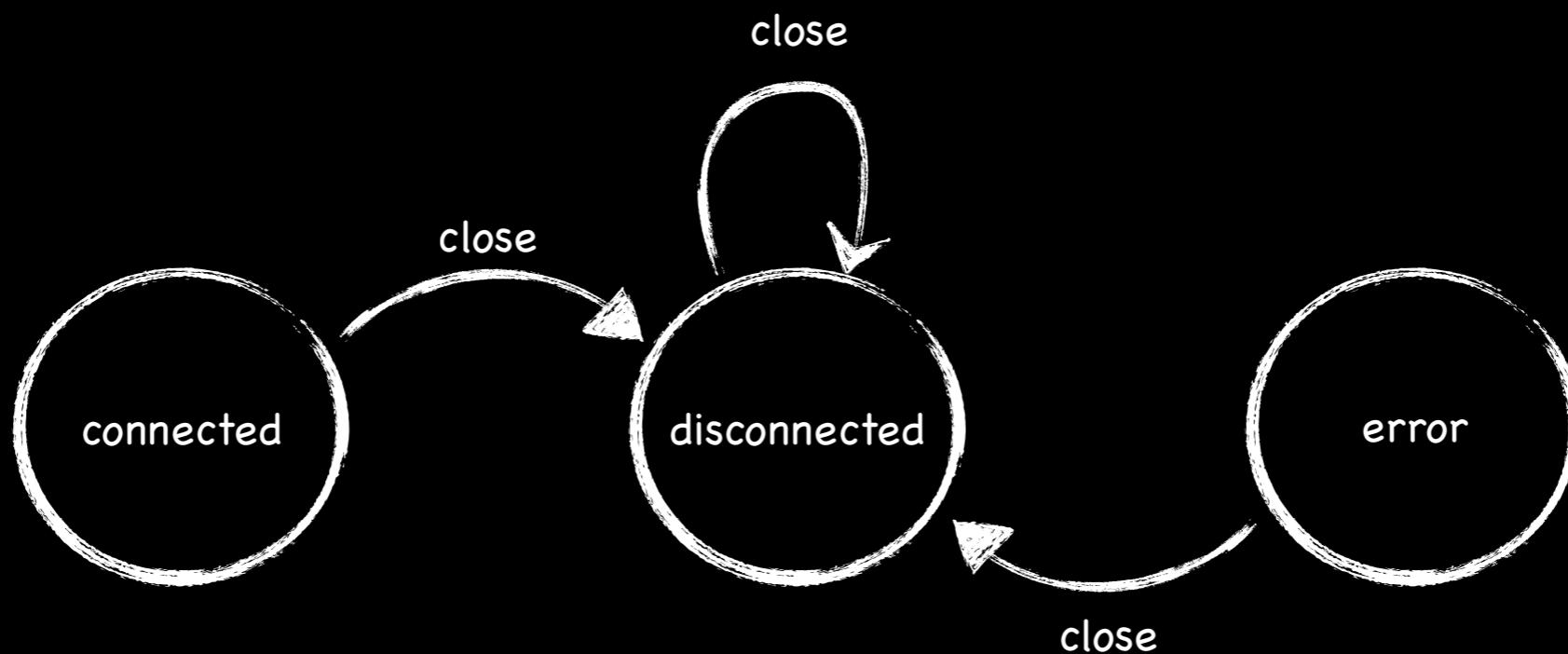
```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
}
```

Quick Check



```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
}
```

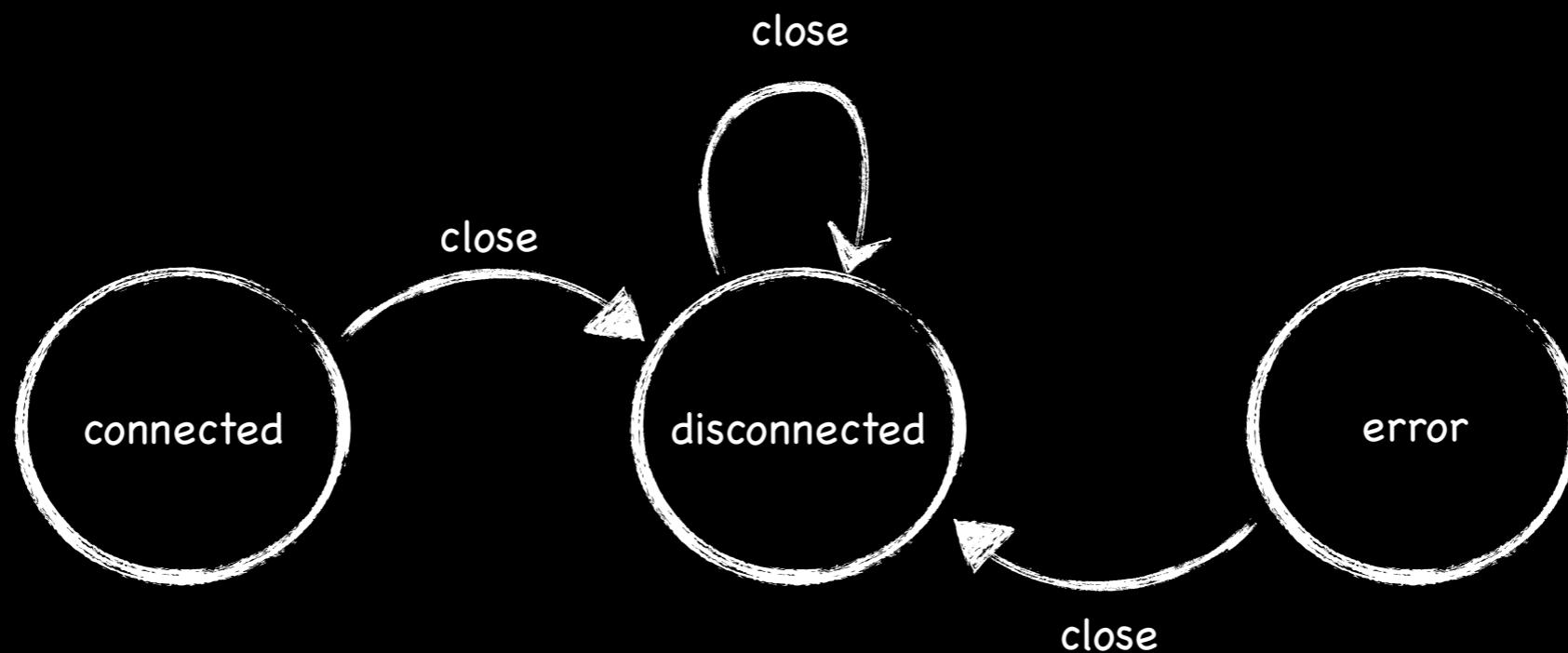
Quick Check



```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
}
```

A red 'X' is drawn over the `c.close();` line, indicating it is incorrect or unnecessary.

Quick Check



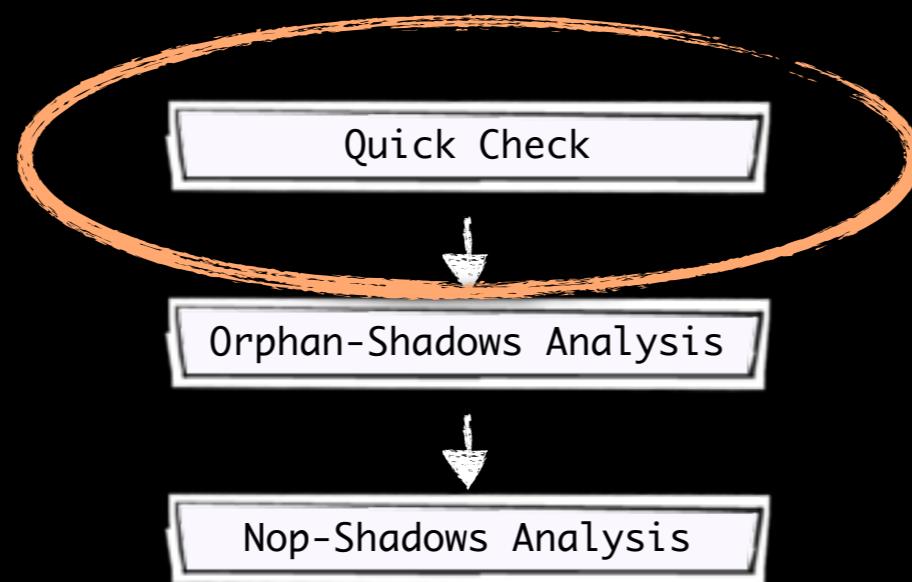
```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
}
```

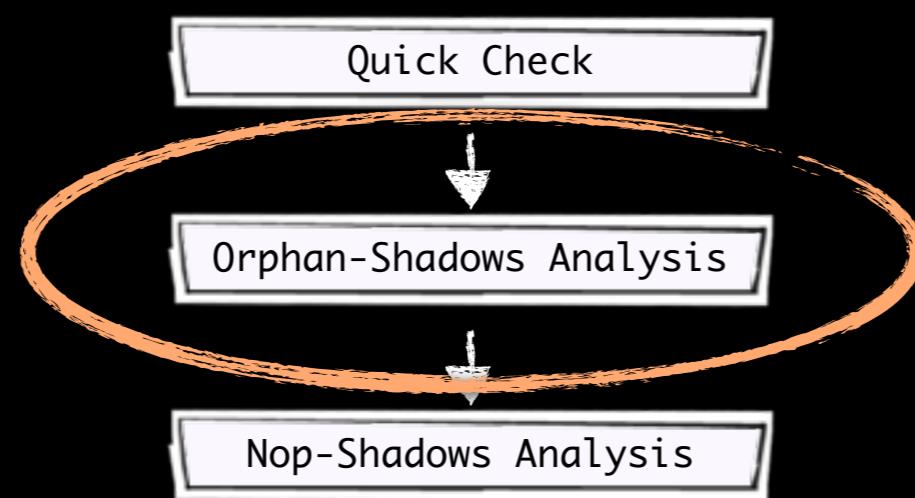
About Quick Check

- Strength: can rule out **very quickly** programs that **obviously** cannot violate the monitored property
- works well on surprisingly many programs
- Weakness: **not precise enough** to handle interesting cases, in which all necessary symbols match somewhere in the program

Quick Check is not perfect

```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
    Connection c2 = new Connection();  
    c2.write("foo");  
}
```





Orphan-Shadows Analysis

Like Quick Check but one object at a time.

```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.read());  
    }  
    c.close();  
    Connection c2 = new Connection();  
    c2.write("foo");  
}
```

Differentiate between c and c2.

Variables can be aliased

```
public static void main(String[] args) {  
    Connection good    = new Connection();  
    Connection bad     = new Connection();  
    Connection better  = new Connection();  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

Variables can be aliased

```
public static void main(String[] args) {  
    Connection good    = new Connection();  
    Connection bad     = new Connection();  
    Connection better  = new Connection();  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

good and bad
are now
"aliased"

Variables can be aliased

```
public static void main(String[] args) {  
    Connection good    = new Connection();  
    Connection bad     = new Connection();  
    Connection better  = new Connection();  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

good and bad
are now
"aliased"

better is not
"aliased"

Variables can be aliased

```
public static void main(String[] args) {  
    Connection good    = new Connection();  
    Connection bad     = new Connection();  
    Connection better  = new Connection();  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

good and bad
are now
"aliased"

better is not
"aliased"

how can we compute this?

Points-to
analysis

```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

new Connection(); //1

new Connection(); //2

new Connection(); //3

```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

new Connection(); //1

new Connection(); //2

new Connection(); //3

good

bad

better

```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better  = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

new Connection(); //1

new Connection(); //2

new Connection(); //3

good

bad

better

```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better  = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

new Connection(); //1

new Connection(); //2

new Connection(); //3



good



bad



better

```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better  = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

new Connection(); //1

new Connection(); //2

new Connection(); //3

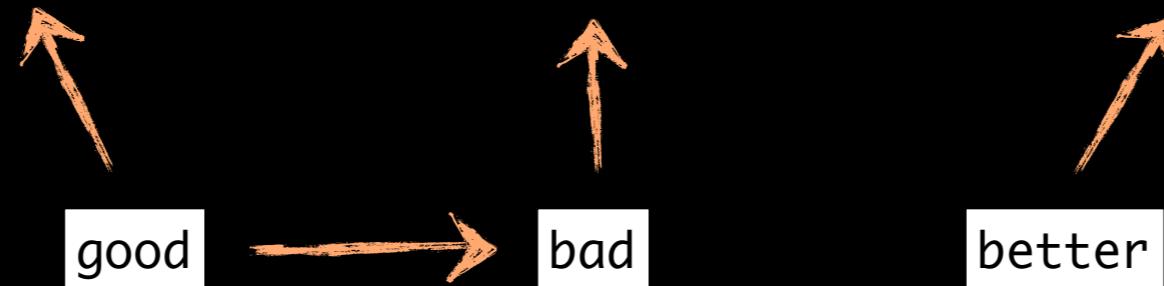


```
public static void main(String[] args) {  
    Connection good    = new Connection(); //1  
    Connection bad     = new Connection(); //2  
    Connection better = new Connection(); //3  
  
    bad.close();  
  
    if(args.length>0)  
        good = bad;  
  
    good.write("let's hope we gave no arguments");  
    better.write("I don't really care");  
}
```

new Connection(); //1

new Connection(); //2

new Connection(); //3



points-to(good) = { new Connection(); //1 , new Connection(); //2 }

```

public static void main(String[] args) {
    Connection good    = new Connection(); //1
    Connection bad     = new Connection(); //2
    Connection better = new Connection(); //3

    bad.close();

    if(args.length>0)
        good = bad;

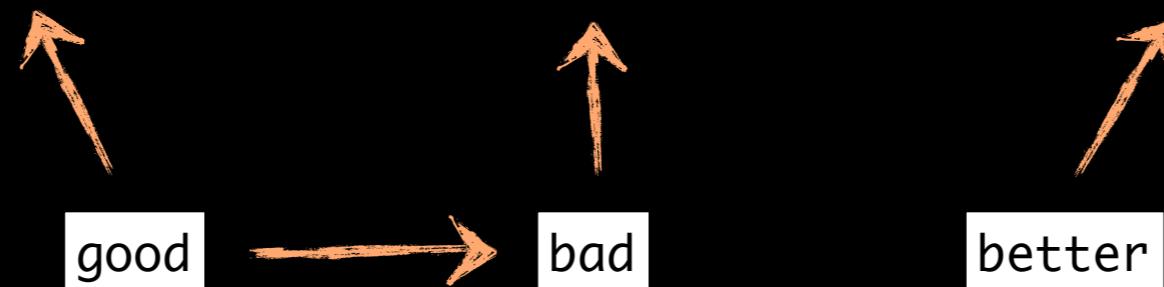
    good.write("let's hope we gave no arguments");
    better.write("I don't really care");
}

```

`new Connection();//1`

`new Connection();//2`

`new Connection();//3`



`points-to(good) = { new Connection();//1 , new Connection();//2 }`
`points-to(bad) = { new Connection();//2 }`

```

public static void main(String[] args) {
    Connection good    = new Connection(); //1
    Connection bad     = new Connection(); //2
    Connection better = new Connection(); //3

    bad.close();

    if(args.length>0)
        good = bad;

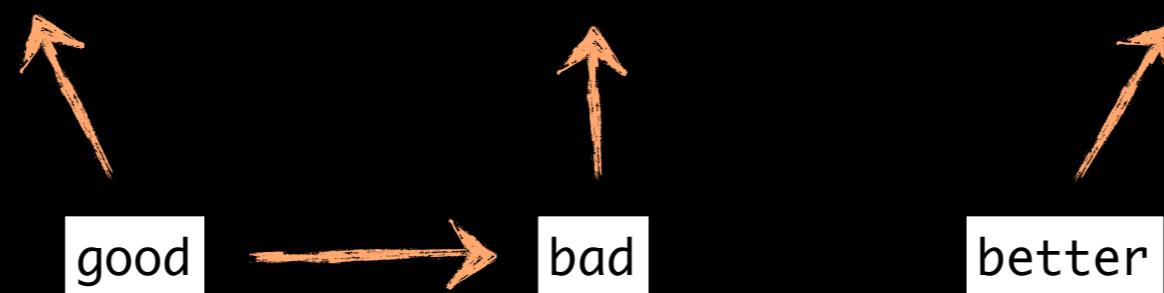
    good.write("let's hope we gave no arguments");
    better.write("I don't really care");
}

```

`new Connection();//1`

`new Connection();//2`

`new Connection();//3`



`points-to(good) = { new Connection();//1 , new Connection();//2 }`

`points-to(bad) = { new Connection();//2 }`

`points-to(better) = { new Connection();//3 }`

```

public static void main(String[] args) {
    Connection good    = new Connection(); //1
    Connection bad     = new Connection(); //2
    Connection better = new Connection(); //3

    bad.close();

    if(args.length>0)
        good = bad;

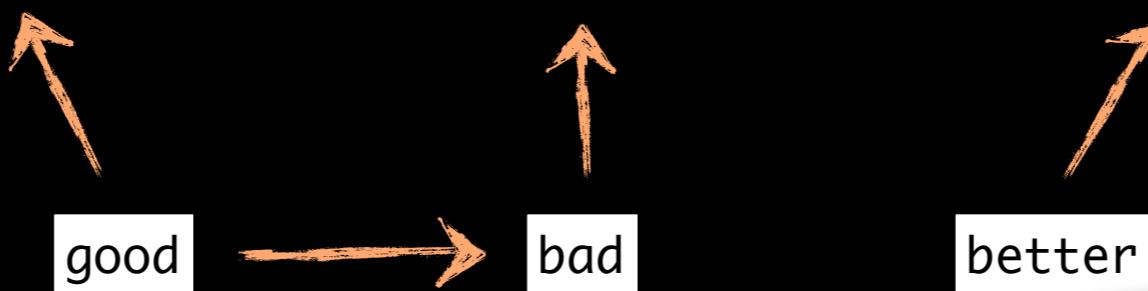
    good.write("let's hope we gave no arguments");
    better.write("I don't really care");
}

```

`new Connection();//1`

`new Connection();//2`

`new Connection();//3`



`points-to(good) = { new Connection();//1 , new Connection();//2 }`
`points-to(bad) = { new Connection();//2 }`
`points-to(better) = { new Connection();//3 }`

```

public static void main(String[] args) {
    Connection good    = new Connection(); //1
    Connection bad     = new Connection(); //2
    Connection better = new Connection(); //3

    bad.close();

    if(args.length>0)
        good = bad;

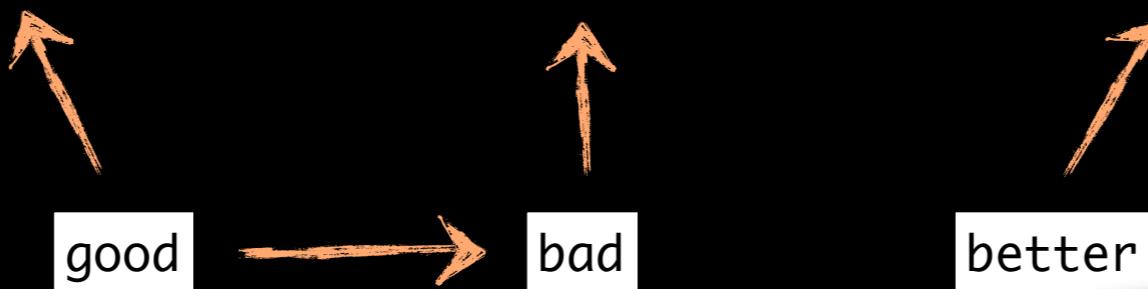
    good.write("let's hope we gave no arguments");
    better.write("I don't really care");
}

```

`new Connection();//1`

`new Connection();//2`

`new Connection();//3`



`points-to(good) = { new Connection();//1 }`

`points-to(bad) = { new Connection();//2 }`

`points-to(better) = { new Connection();//3 }`

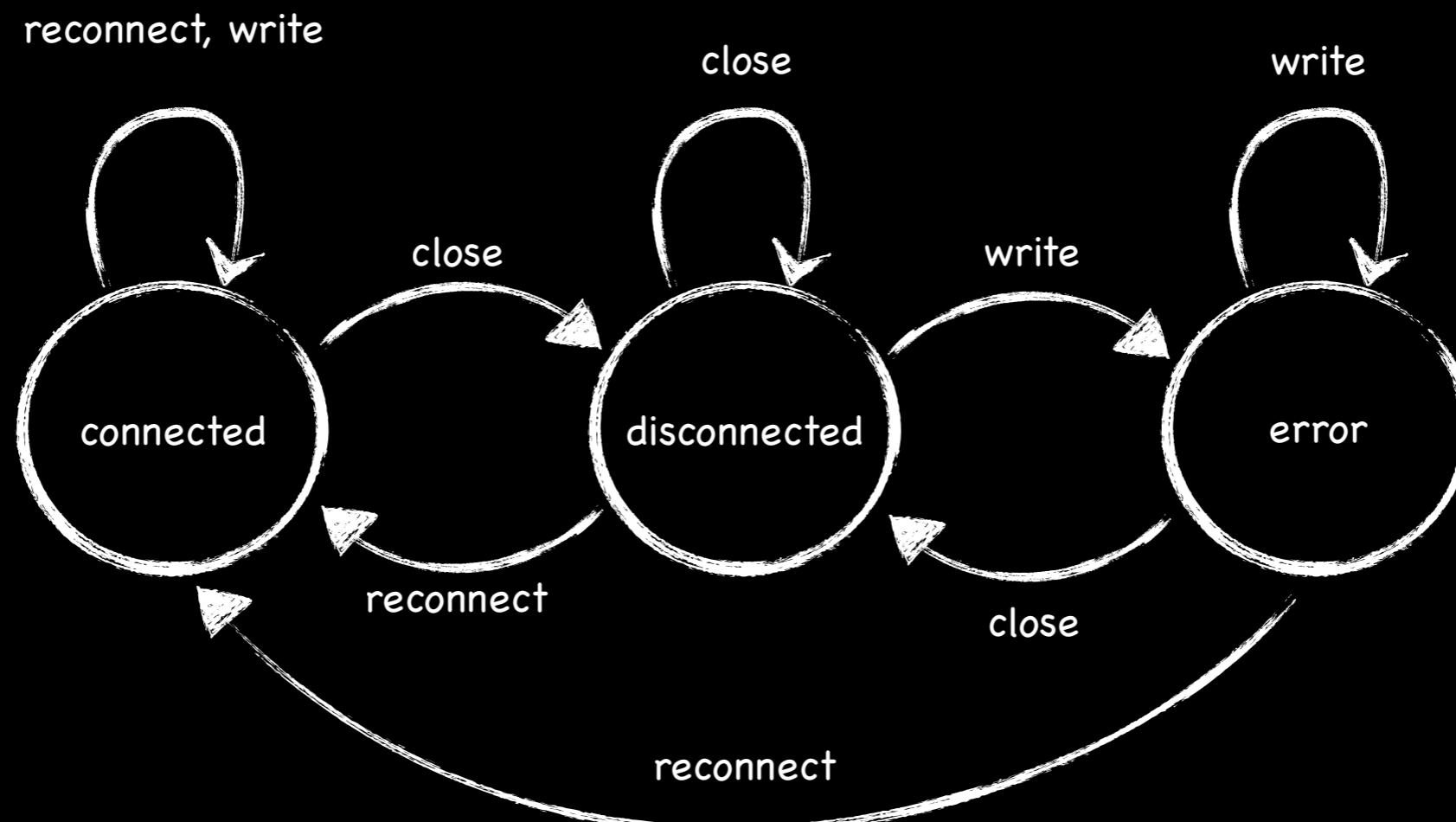
`new Connection();//2 }`

`{ }`

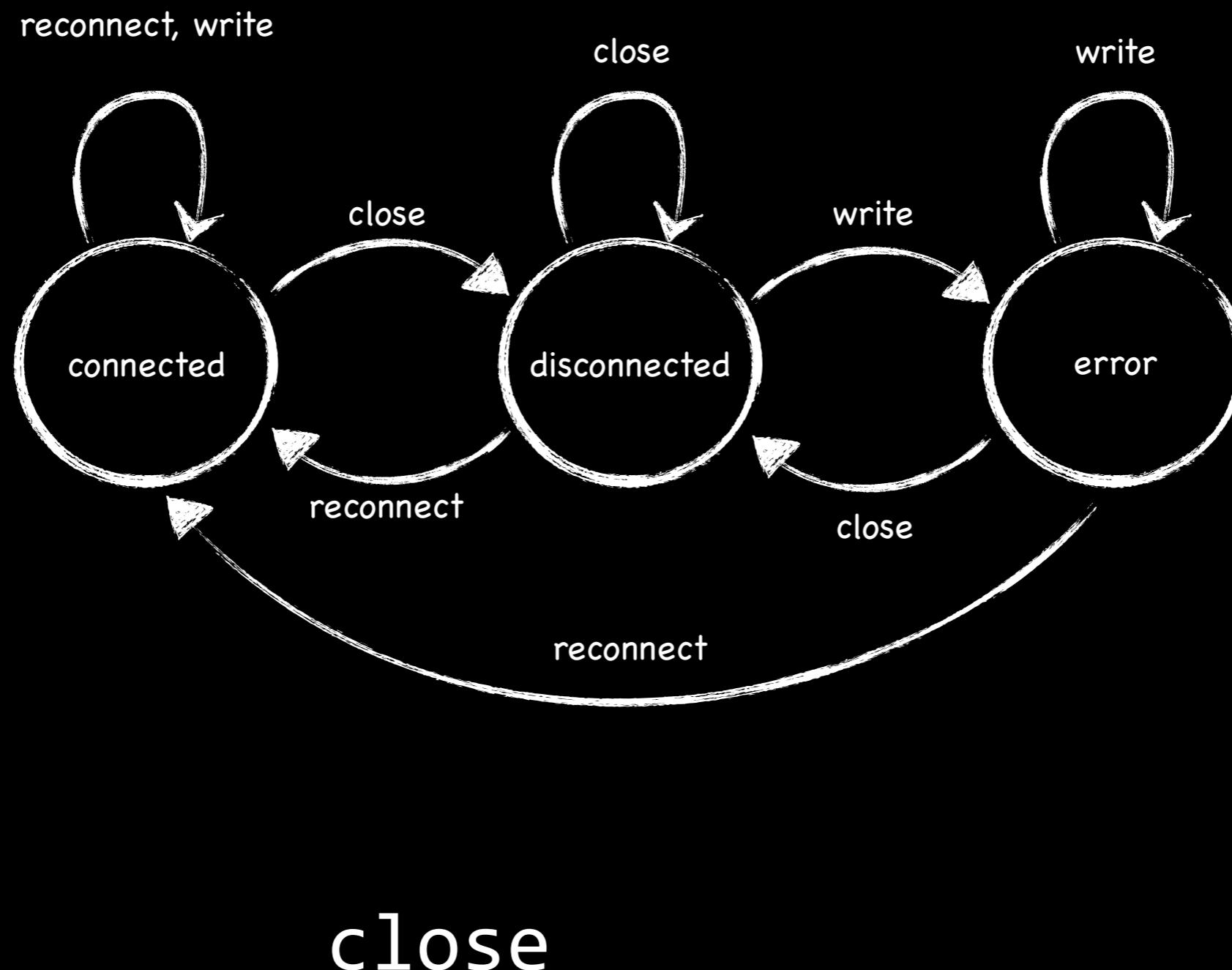
`{ }`

*good and bad
"may-alias"*

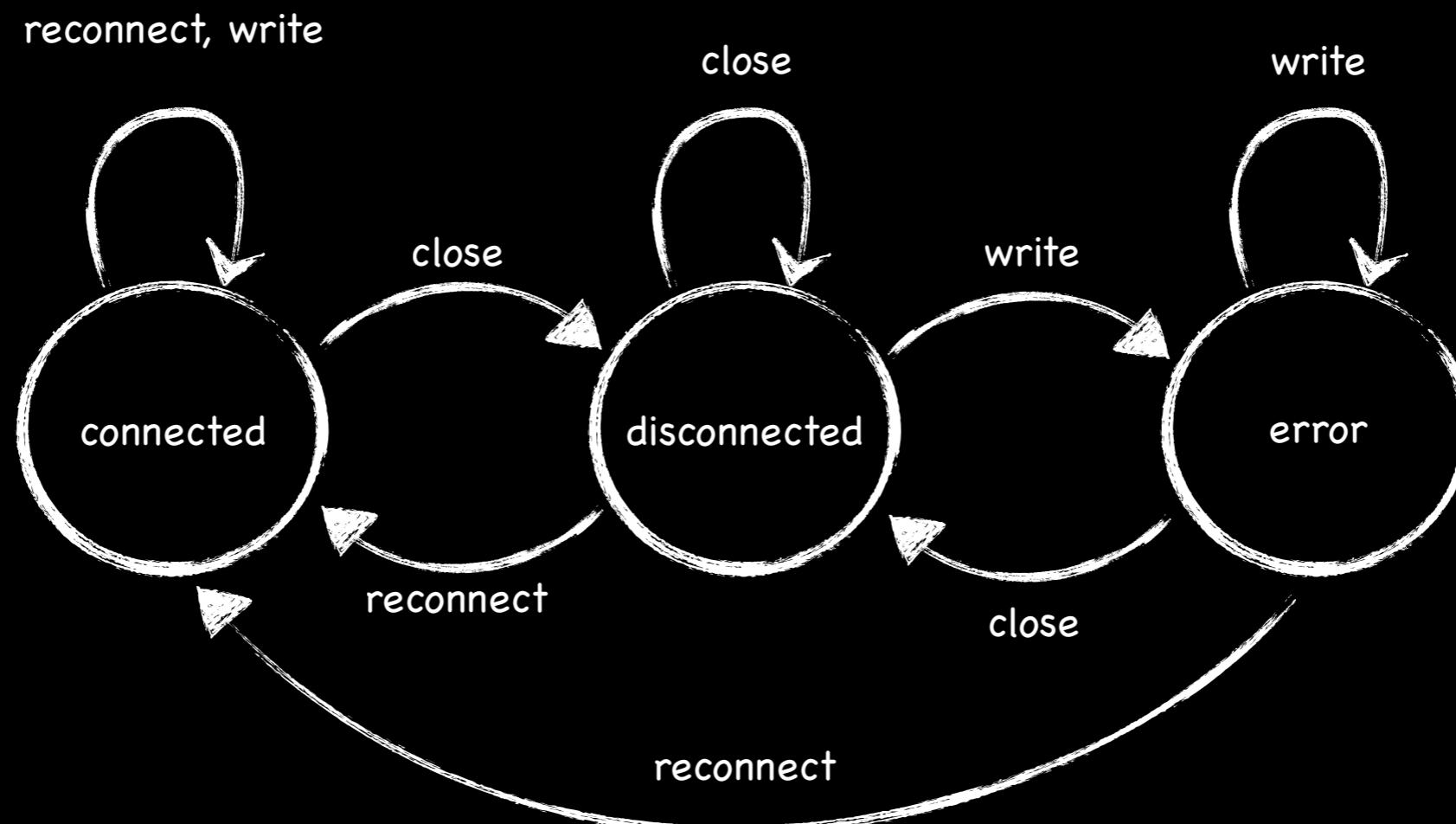
Orphan-Shadows Analysis



Orphan-Shadows Analysis



Orphan-Shadows Analysis



{ close, write }

Orphan-Shadows Analysis

{close, write}

Orphan-Shadows Analysis

{close, write}

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public static void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

W₁ C₁ W₂

Orphan-Shadows Analysis

{close, write}

{c₁} {w₁, w₂}

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

W₁ C₁ W₂

Orphan-Shadows Analysis

{close, write}

{c₁} × {w₁, w₂}

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public static void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

W₁ C₁ W₂

Orphan-Shadows Analysis

{close, write}

{c₁} × {w₁, w₂}

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public static void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

W₁ W₂
C₁

{c₁, w₁}

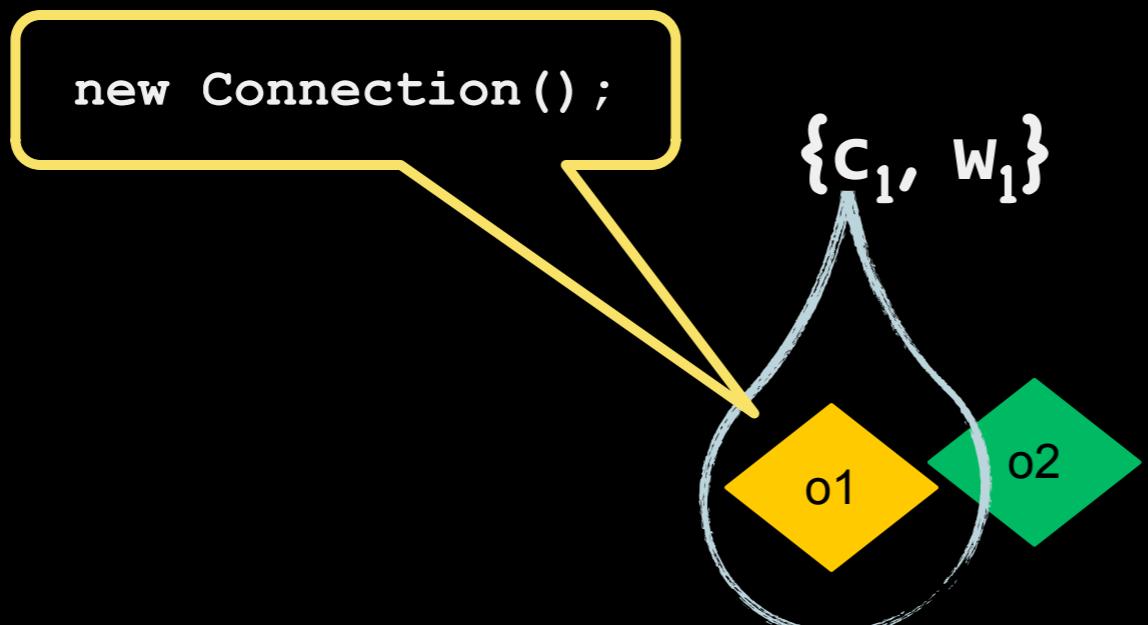
Orphan-Shadows Analysis

{close, write}

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

W₁ W₂
C₁

{C₁} × {W₁, W₂}



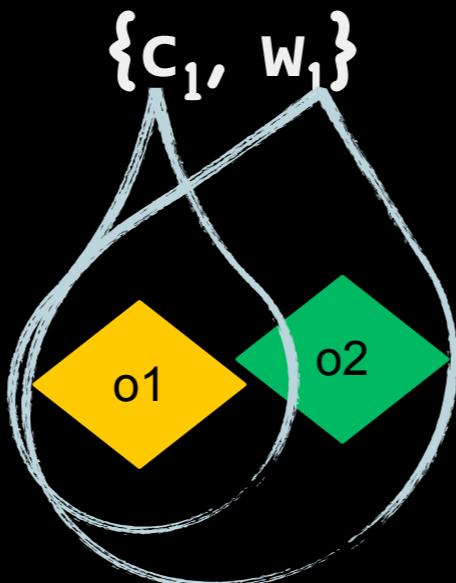
Orphan-Shadows Analysis

{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public static void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

C_1 w_1 w_2



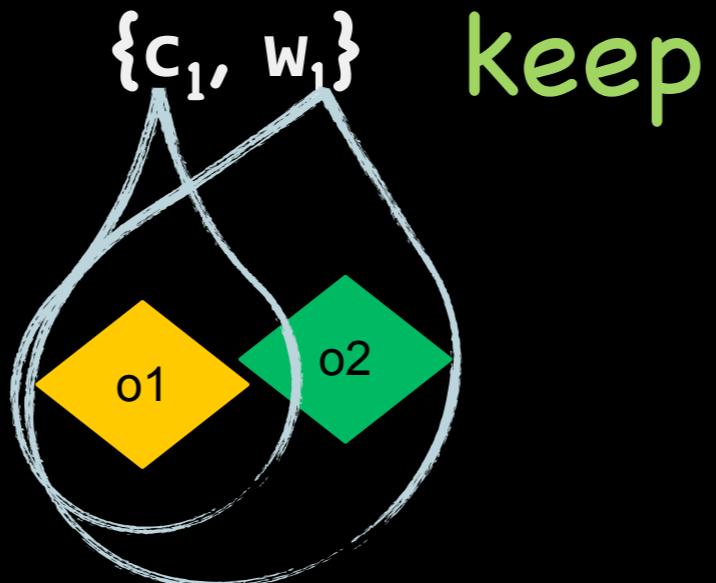
Orphan-Shadows Analysis

{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public static void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

C_1 w_1 w_2



Orphan-Shadows Analysis

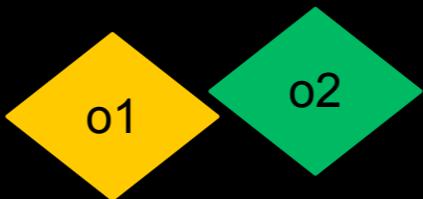
{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public static void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

C₁ **W₁** **W₂**

$\{c_1, w_1\}$ keep



$\{c_1, w_2\}$

Orphan-Shadows Analysis

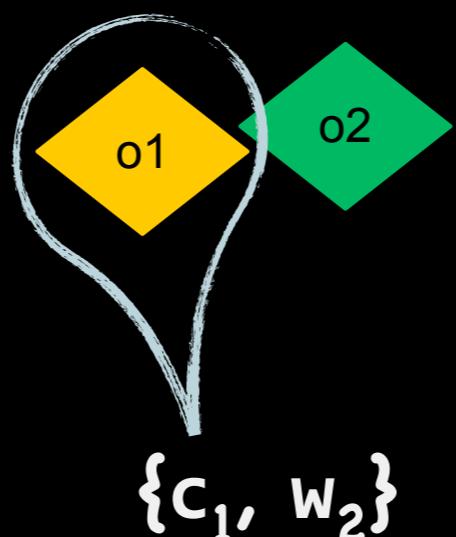
{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public static void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

C_1 W_1 W_2

$\{c_1, w_1\}$ keep



Orphan-Shadows Analysis

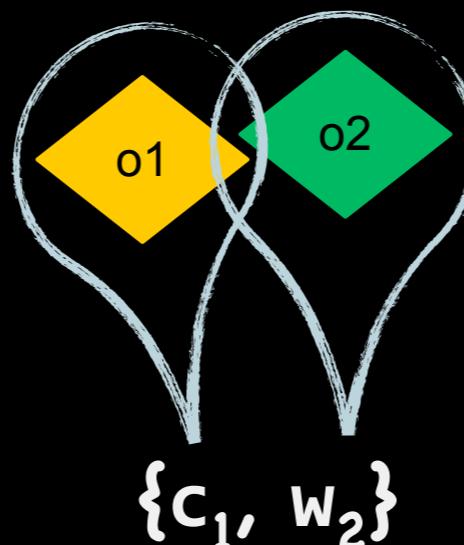
{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public static void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

C_1 W_1 W_2

$\{c_1, w_1\}$ keep



Orphan-Shadows Analysis

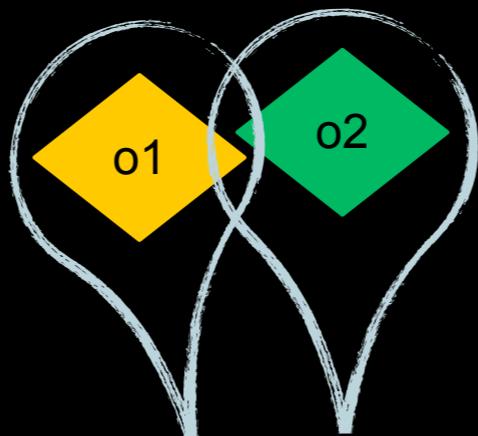
{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public static void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

C_1 w_1 w_2

$\{c_1, w_1\}$ keep



$\{c_1, w_2\}$ no need to keep

Orphan-Shadows Analysis

{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public static void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

W₁ C₁ W₂

$\{c_1, w_1\}$ keep

Orphan-Shadows Analysis

{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {
    private static Writer w;
    public static void init(OutputStream os) {
        w = new PrintWriter(os);
    }
    public static void log(String s) throws IOException {
        w.write(s);
    }
    public void logAll(Reader r) throws IOException {
        StringUtil.copyAll(r,w);
    }
}
```

The diagram shows a code snippet for a `Logging` class. The class has a static field `w` of type `Writer`. It contains three methods: `init`, `log`, and `logAll`. The `init` method initializes `w` to a `PrintWriter` using the provided `OutputStream`. The `log` method writes the provided `String` `s` to `w`. The `logAll` method reads from a `Reader` `r` and copies its contents to `w` using the `StringUtil.copyAll` helper method. Handwritten annotations are present: a blue circle labeled `C1` is placed over the `StringUtil.copyAll` call; a blue circle labeled `W1` is placed over the `w` variable; and a red circle with a large red X is placed over the `logAll` method.

$\{c_1, w_1\}$ keep

Orphan-Shadows Analysis

{close, write}

$\{c_1\} \times \{w_1, w_2\}$

```
public class Logging {  
    private static Writer w;  
    public static void init(OutputStream os) {  
        w = new PrintWriter(os);  
    }  
    public static void log(String s) throws IOException {  
        w.write(s);  
    }  
    public void logAll(Reader r) throws IOException {  
        StringUtil.copyAll(r,w);  
    }  
}
```

W₁
C₁

$\{c_1, w_1\}$ keep

Orphan-Shadows Analysis

```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.write("foo"));  
    }  
c.close();  
    Connection c2 = new Connection();  
c2.write("foo");  
}
```

Orphan-Shadows Analysis

```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.write("foo"));  
    }  
c.close();  
    Connection c2 = new Connection();  
c2.write("foo");  
}
```

Orphan-Shadows Analysis

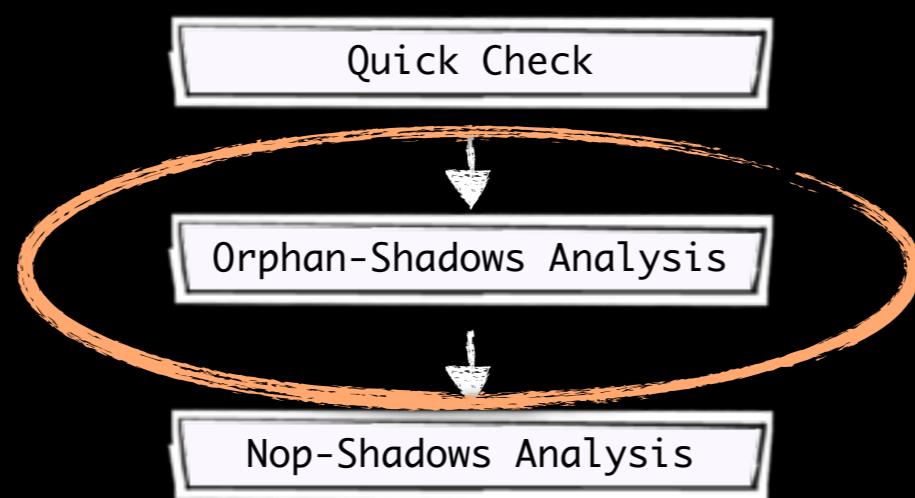
```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.write("foo"));  
    }  
c.close();  
    Connection c2 = new Connection();  
    c2.write("foo");  
}
```

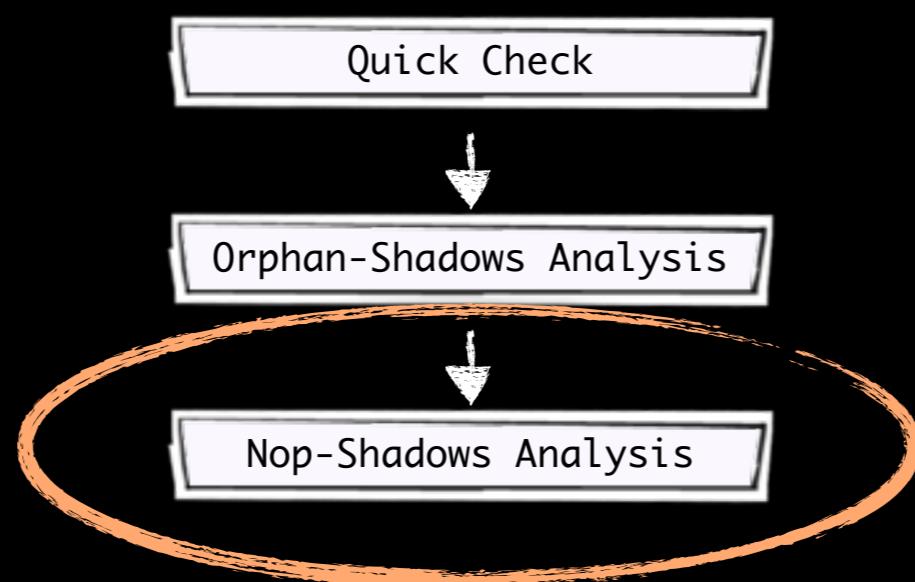
Effect of OSA

- Strength: can disable **80% shadows or more** for many programs
 - greatly reduces number of shadows to consider in third analysis stage
- Weakness: still just a better Quick Check; **only captures rather trivial cases**, in which an object is lacking some symbol to reach a final state

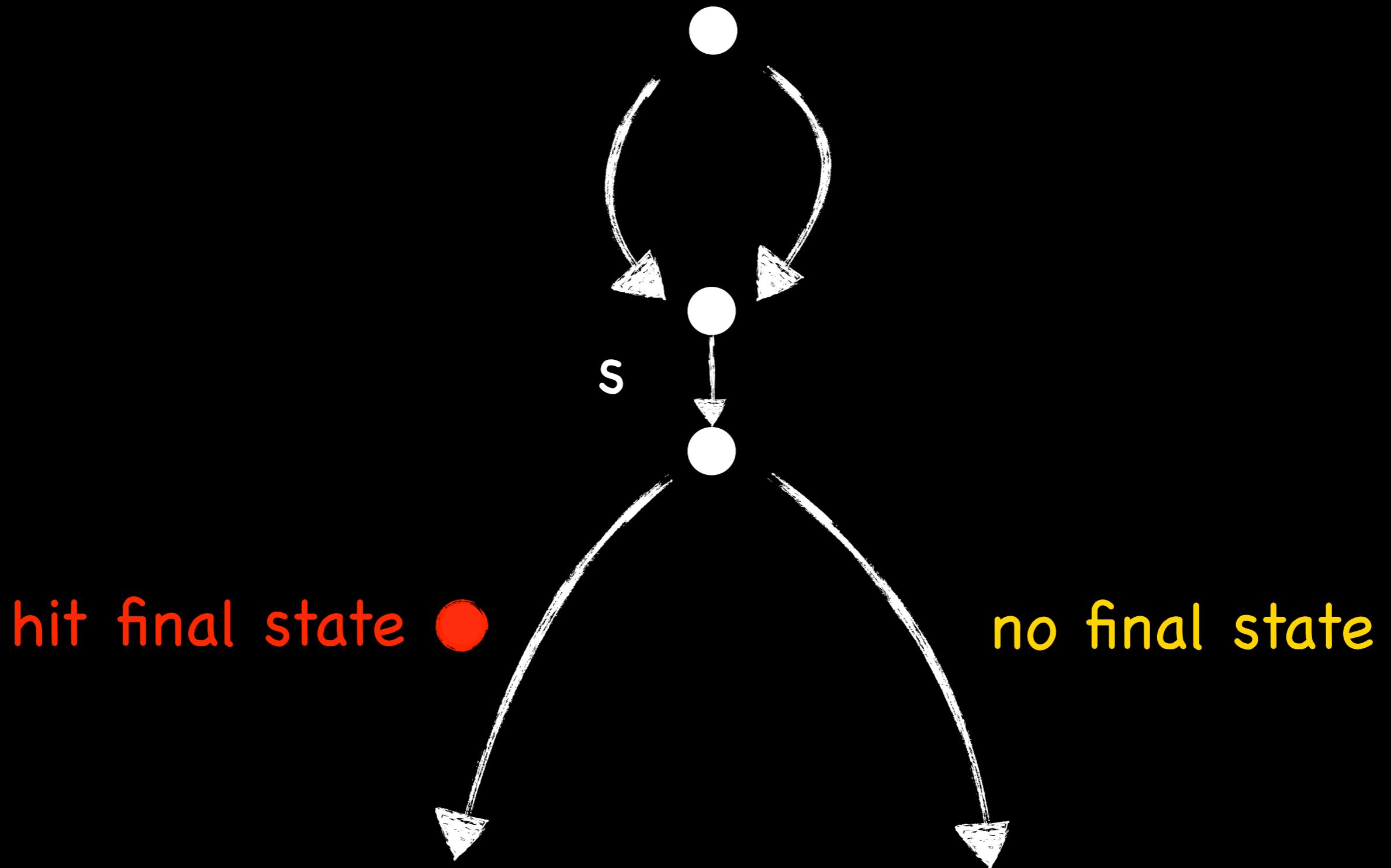
Actually no violation...

```
public static void main(String[] args) {  
    Connection c = new Connection();  
    while(c.hasMoreData()) {  
        System.err.println(c.write("foo"));  
    }  
    c.close();  happens before!  
    Connection c2 = new Connection();  
    c2.write("foo");  
}
```

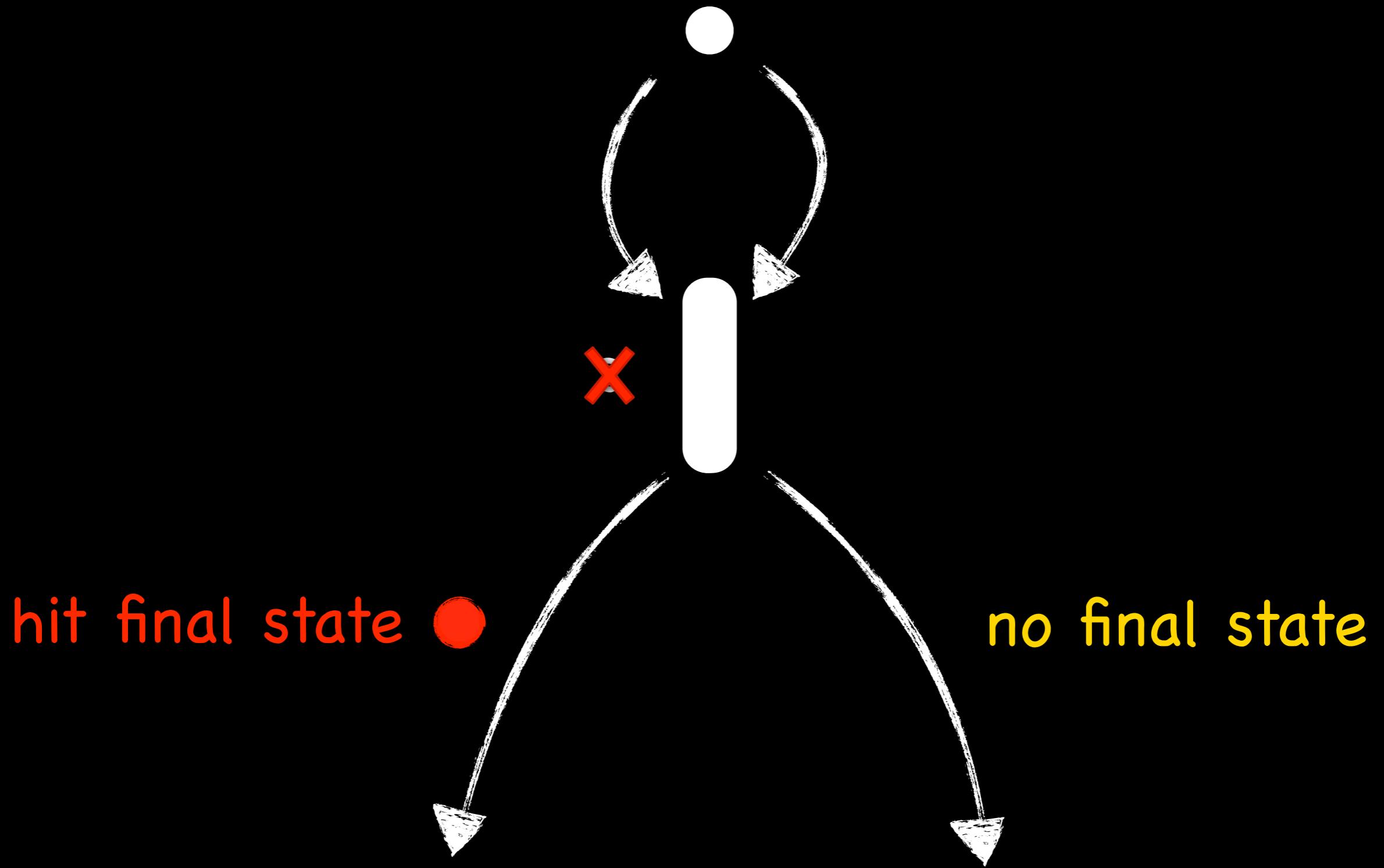




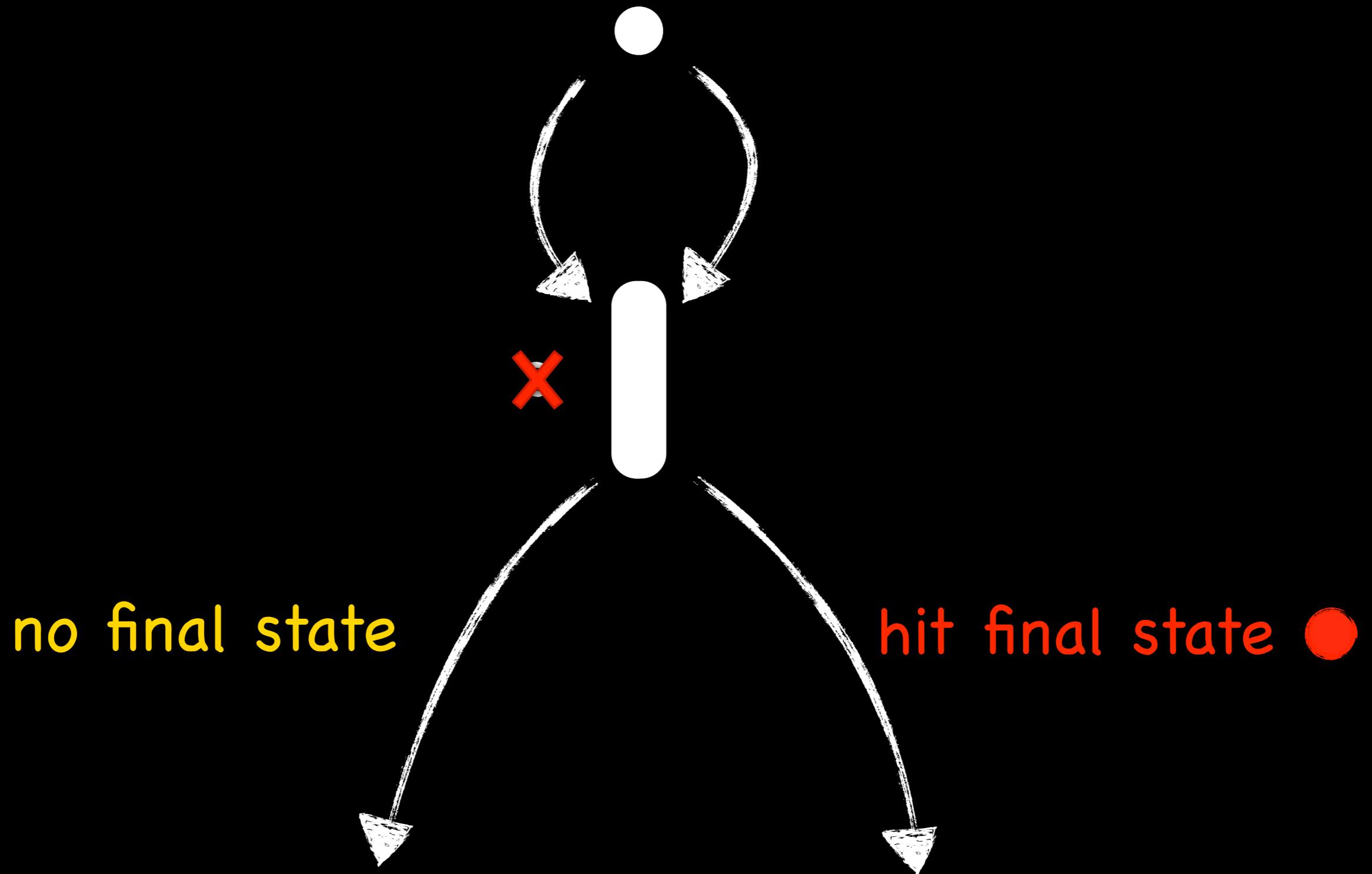
Nop-Shadows Analysis



Nop-Shadows Analysis



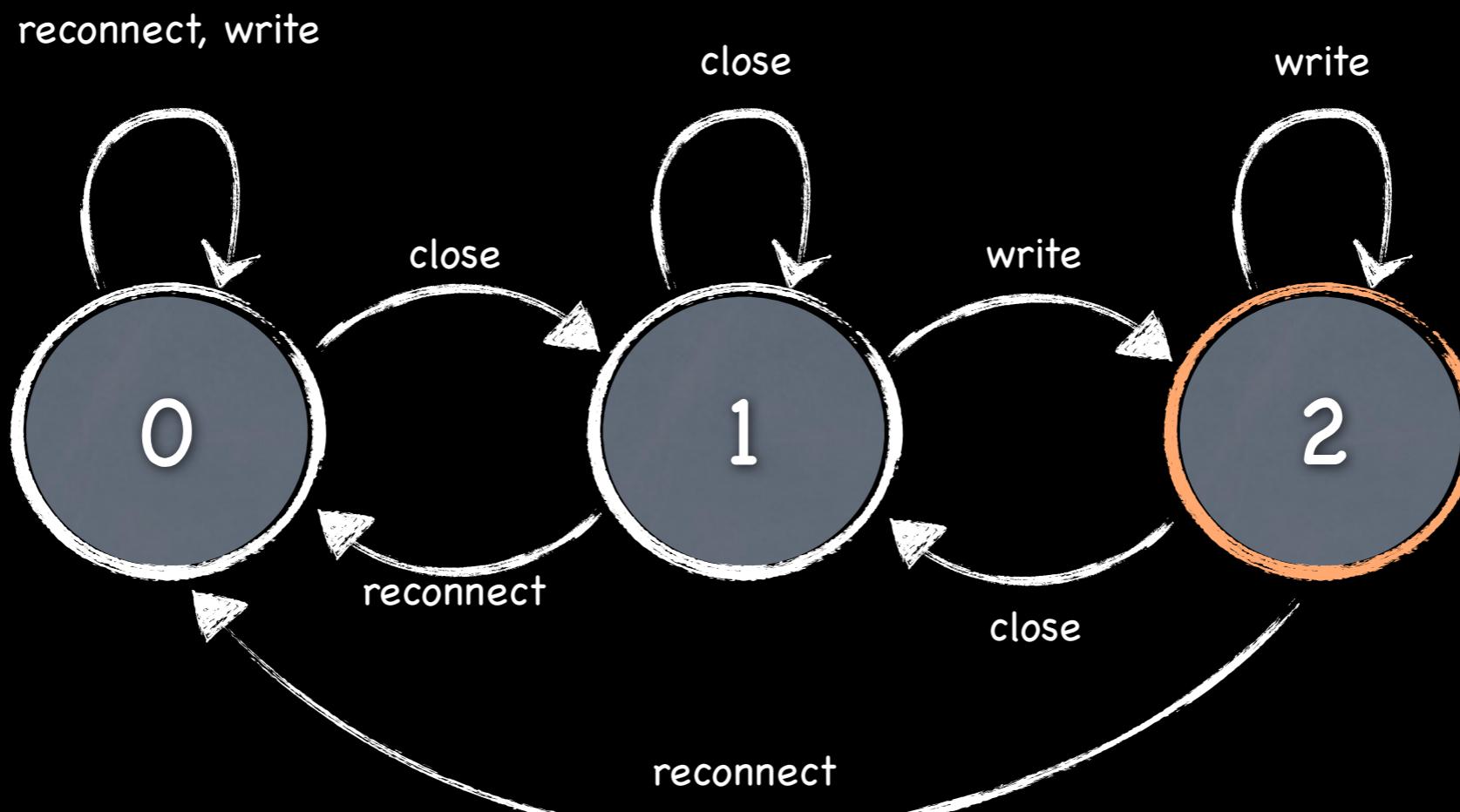
Nop-Shadows Analysis

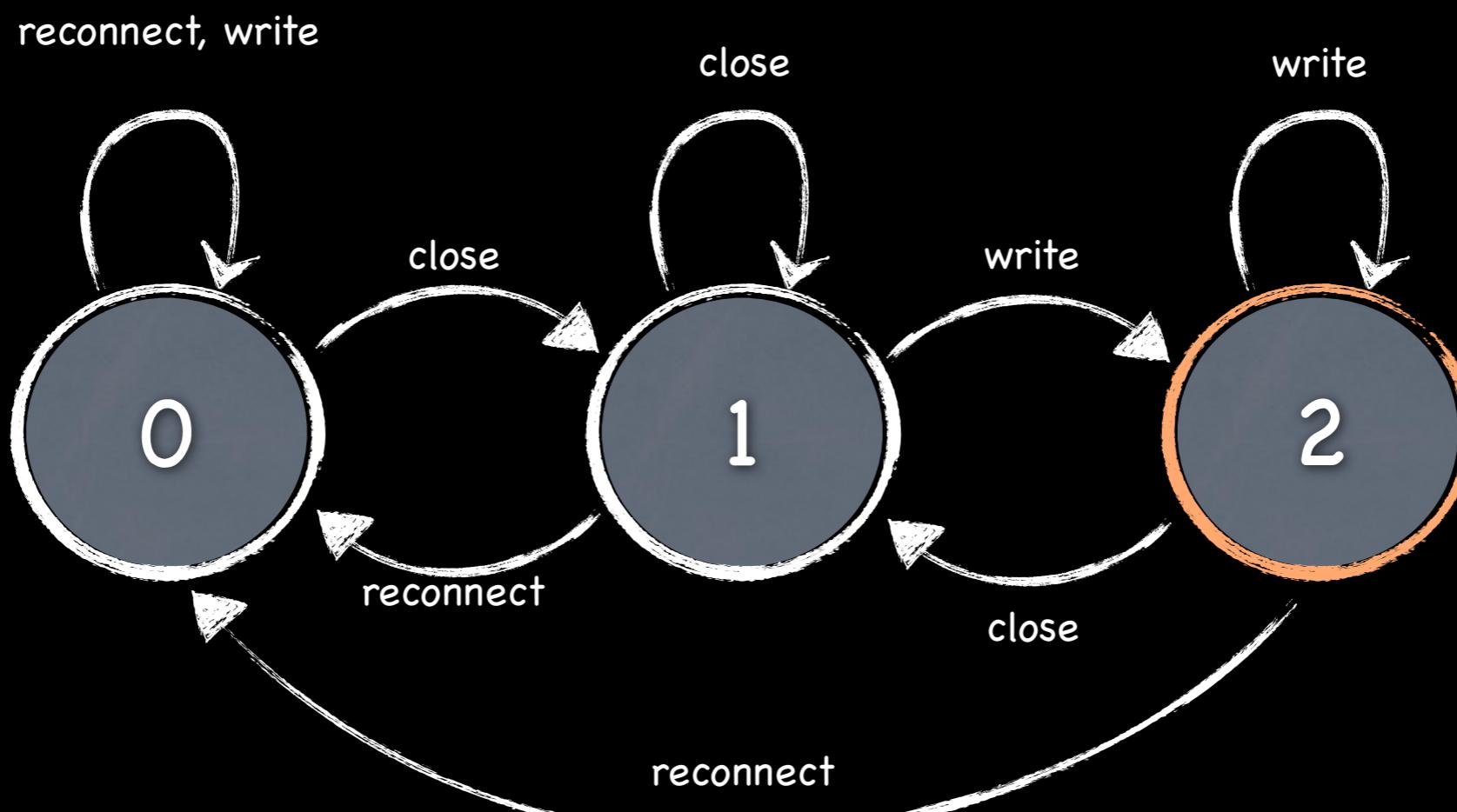
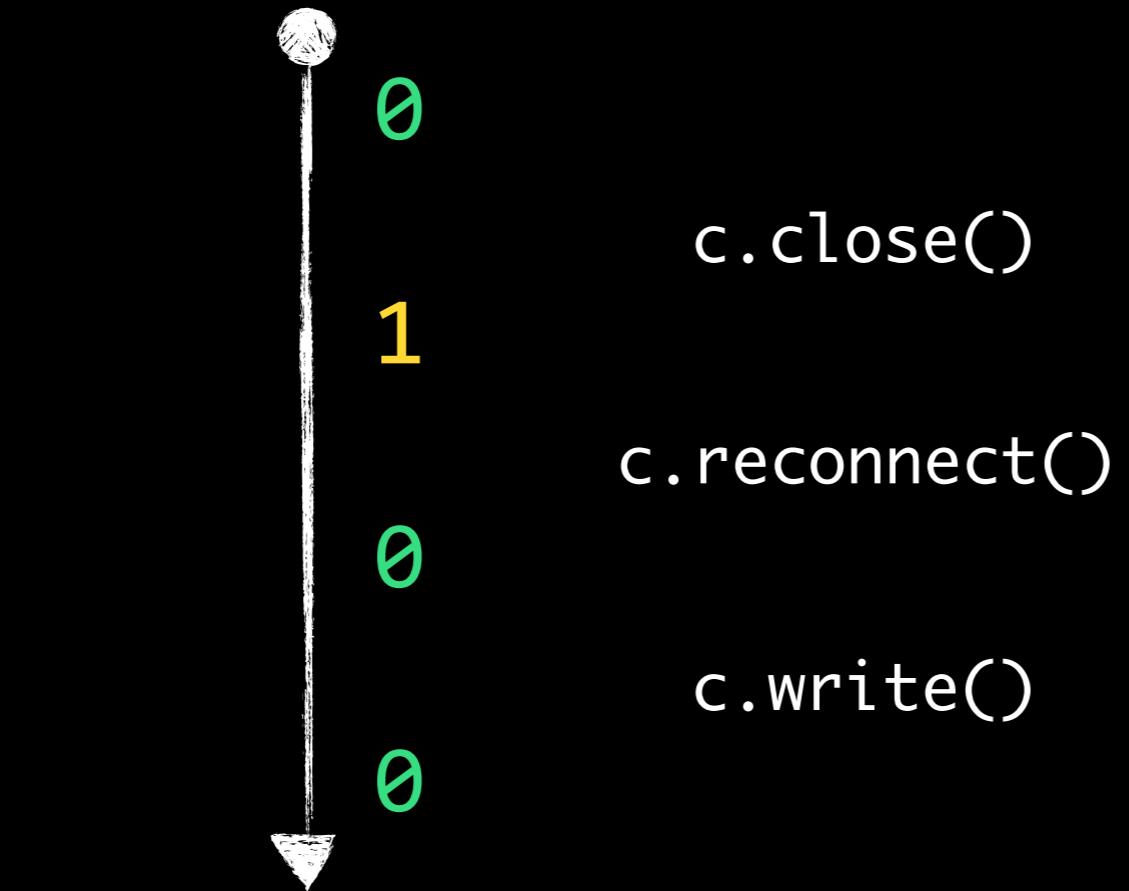


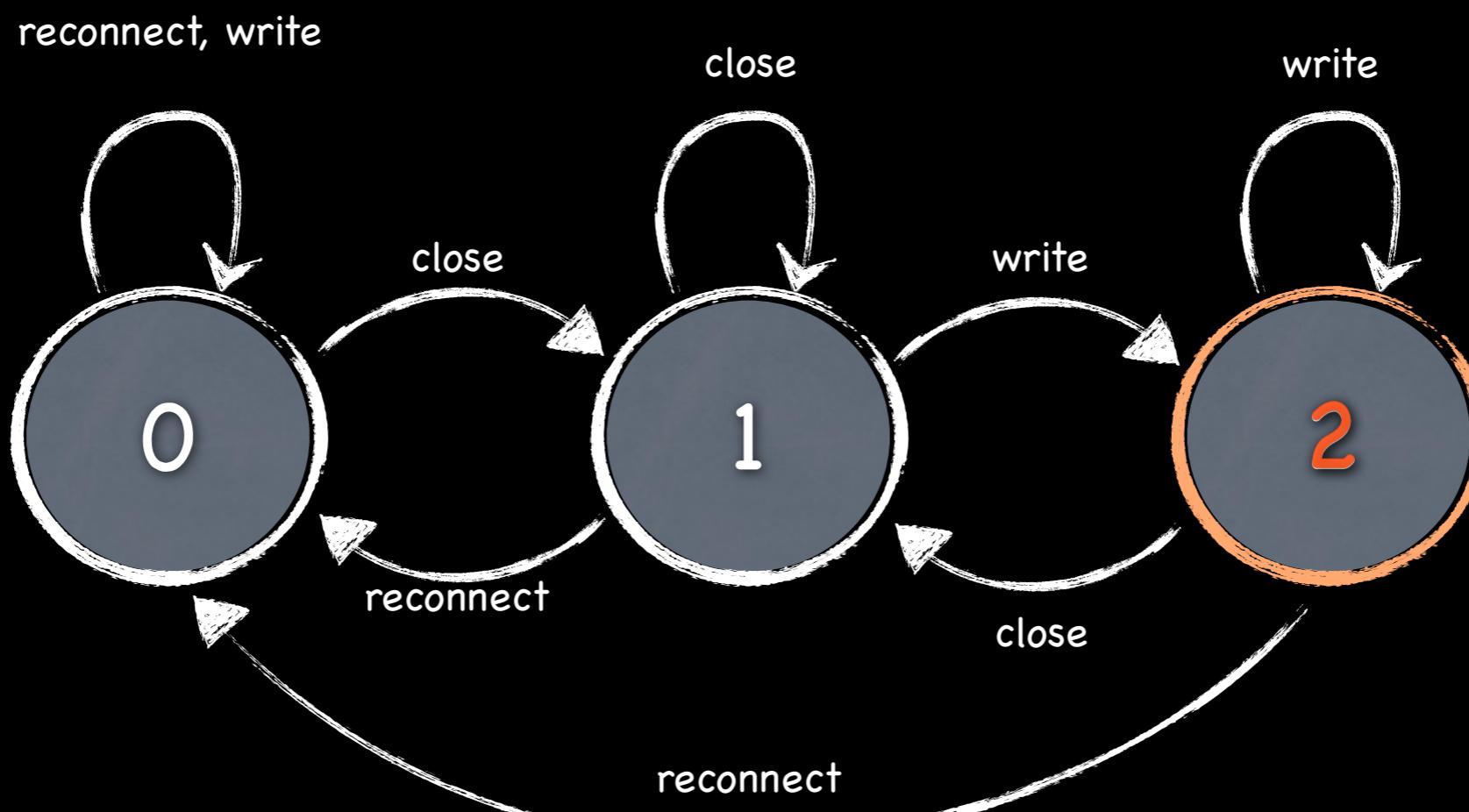
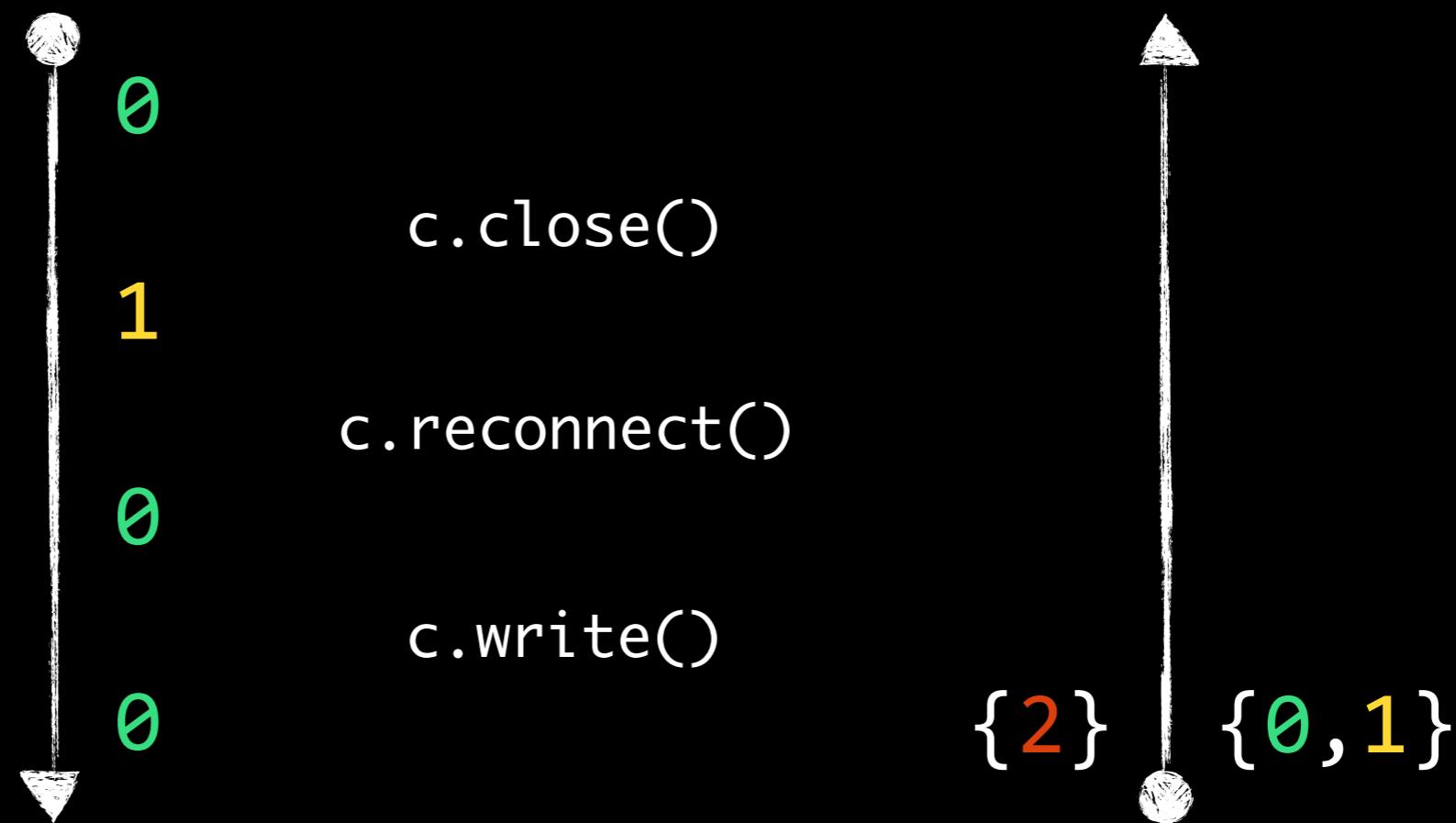
c.close()

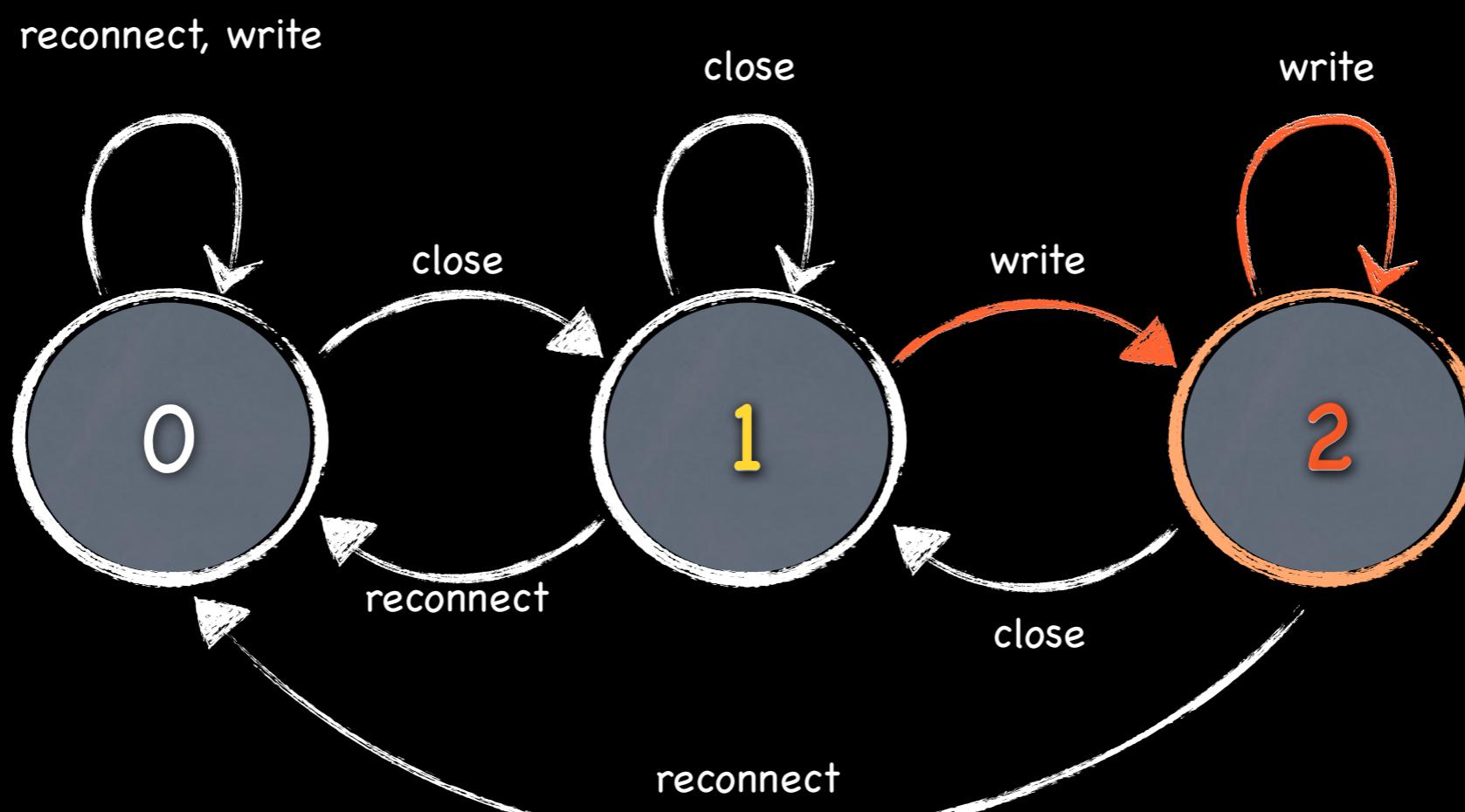
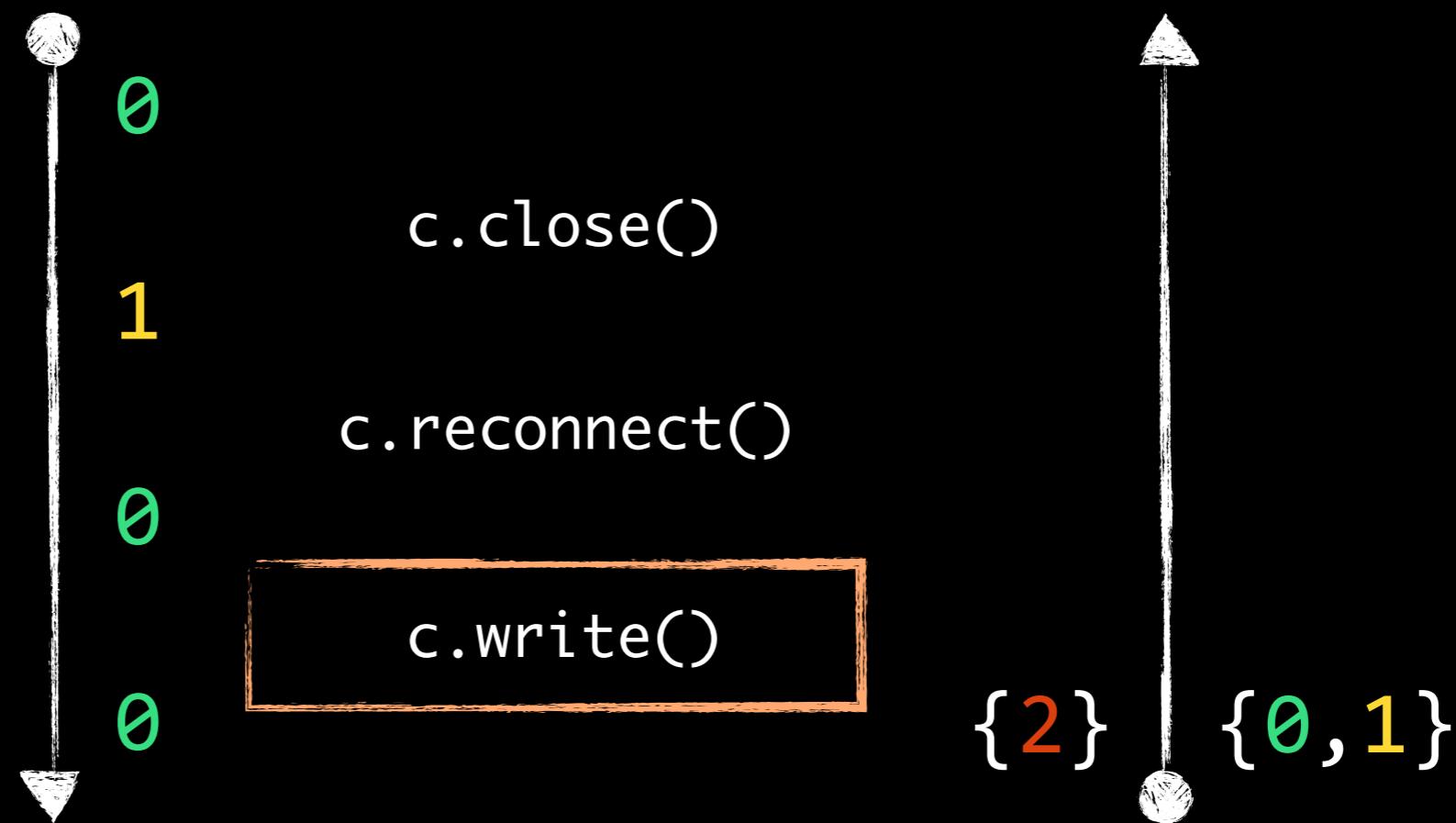
c.reconnect()

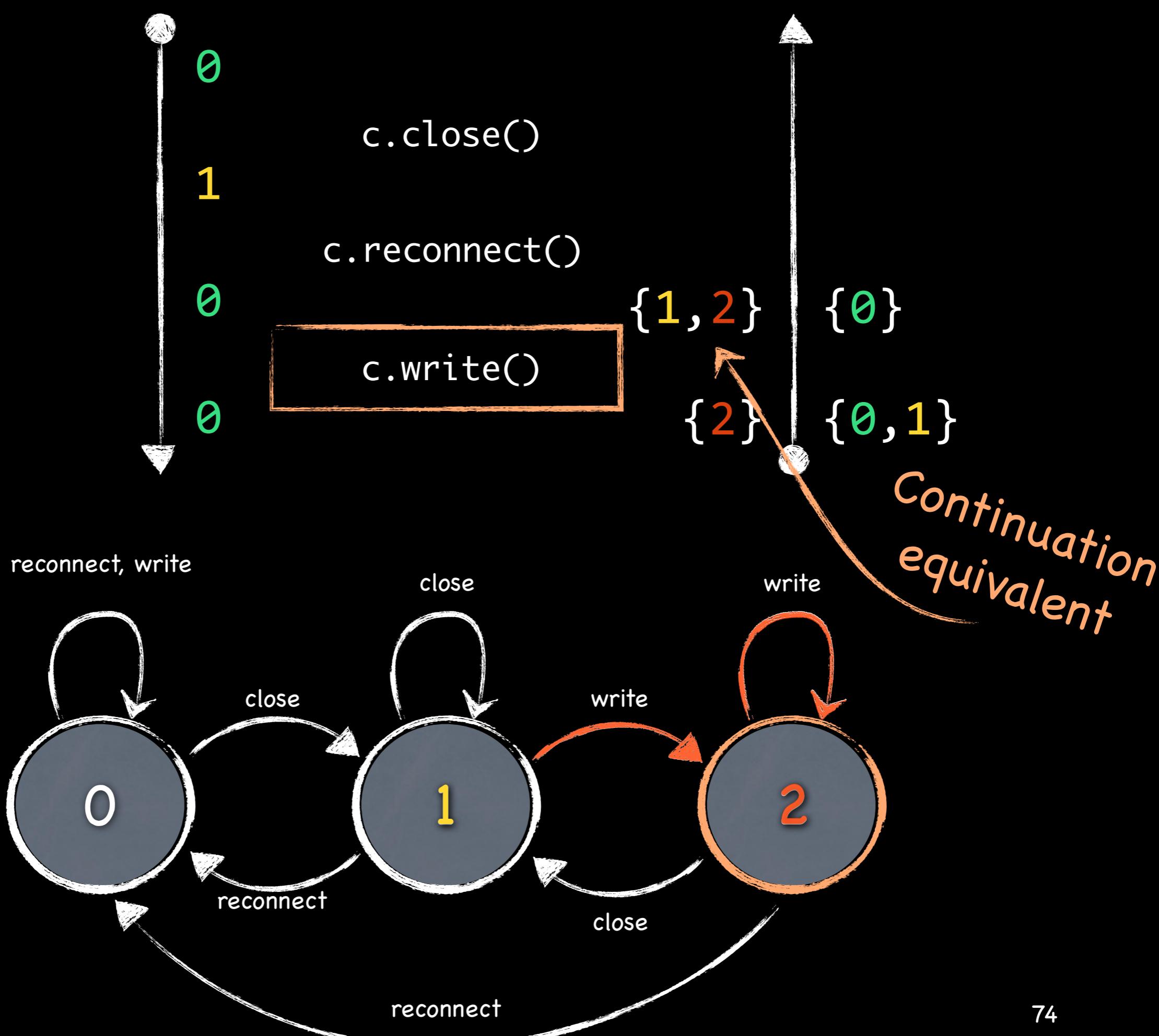
c.write()

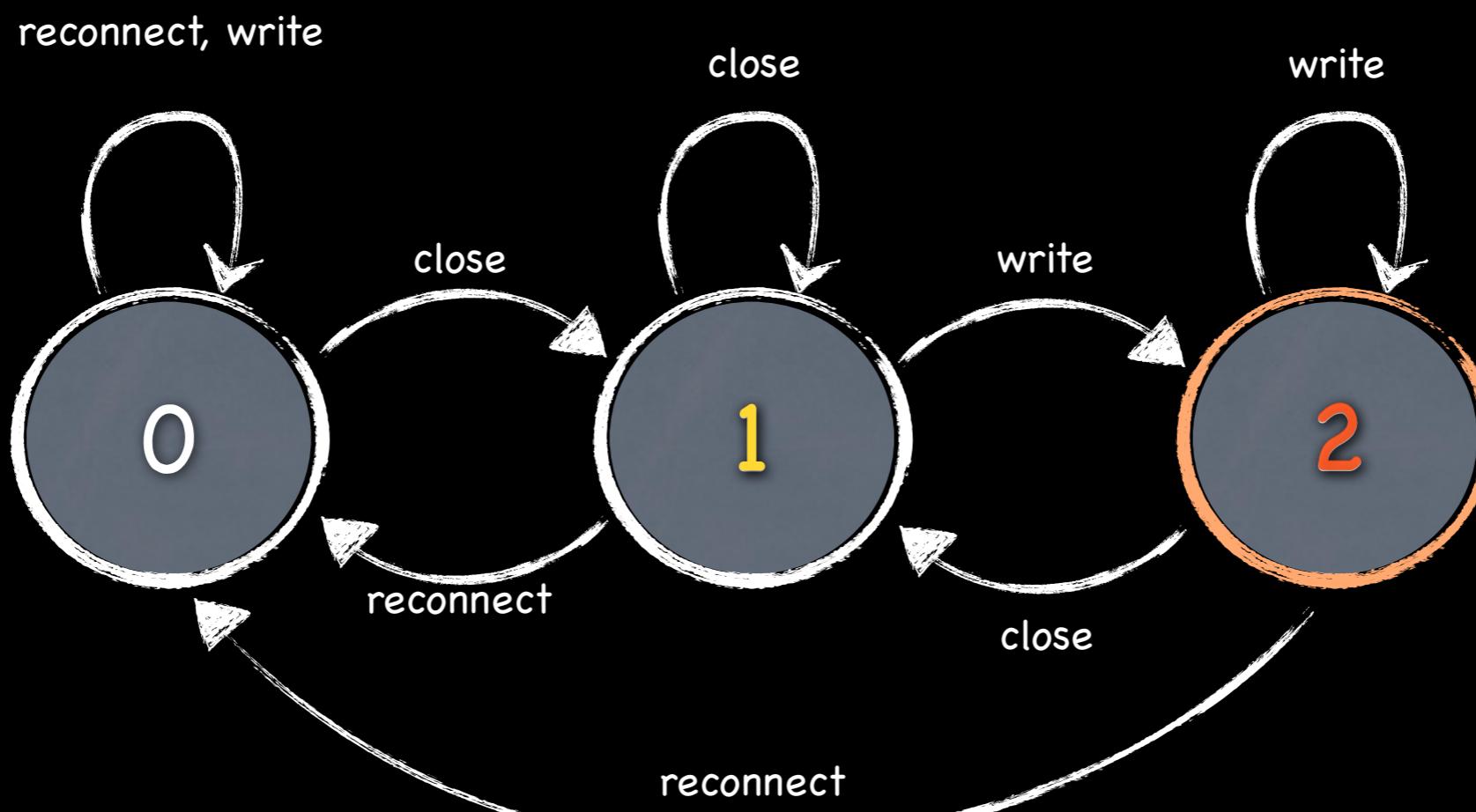
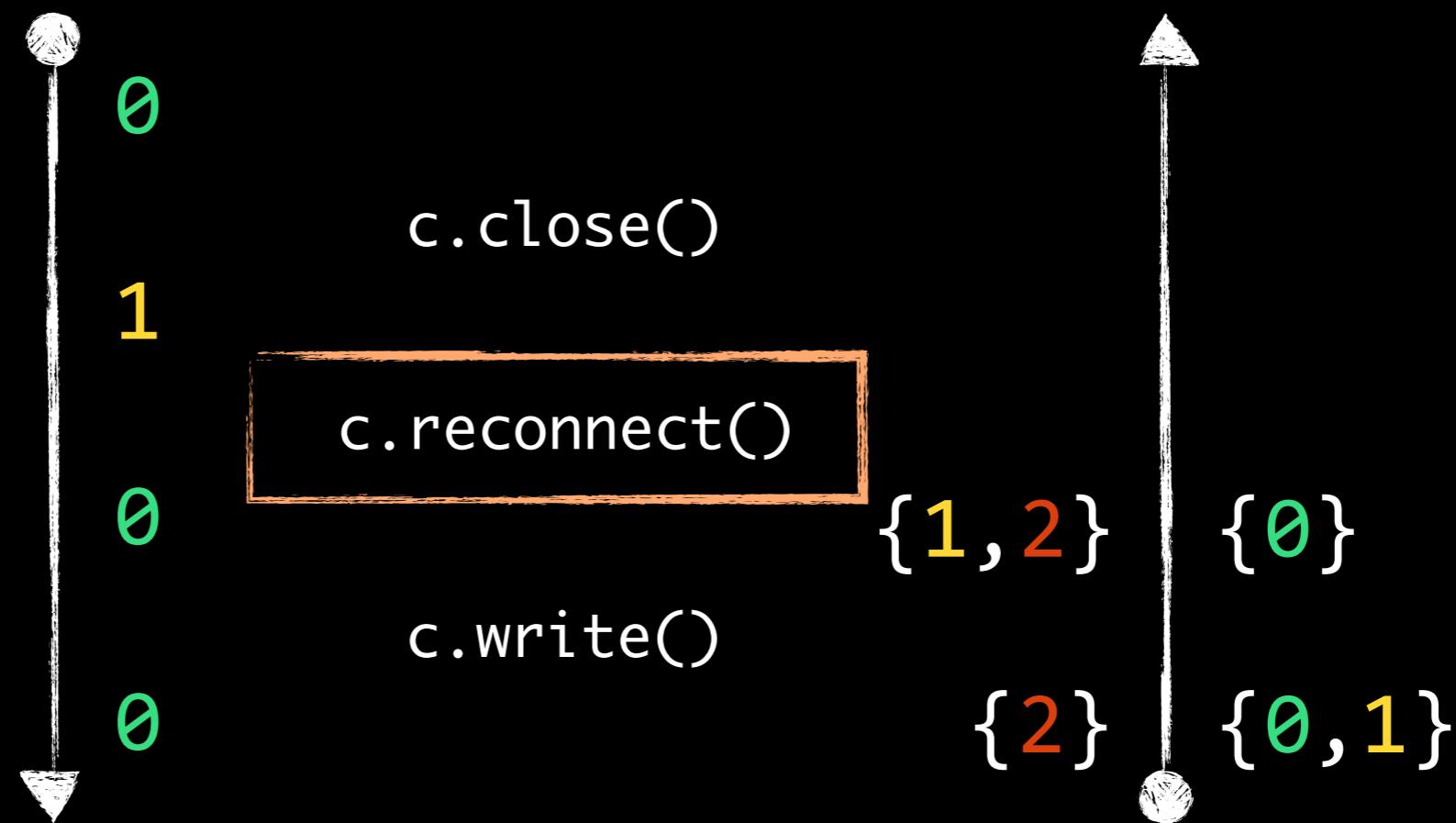


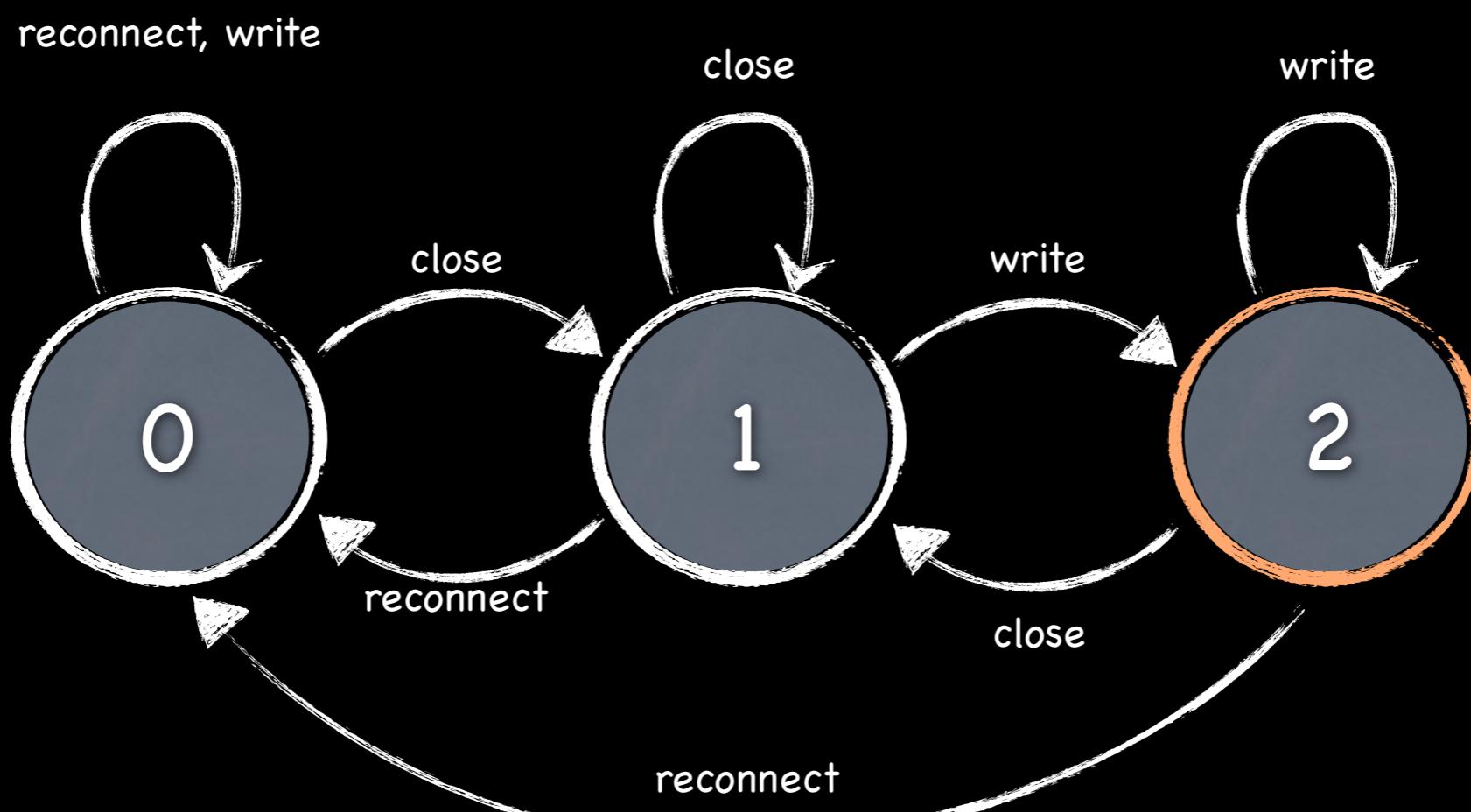
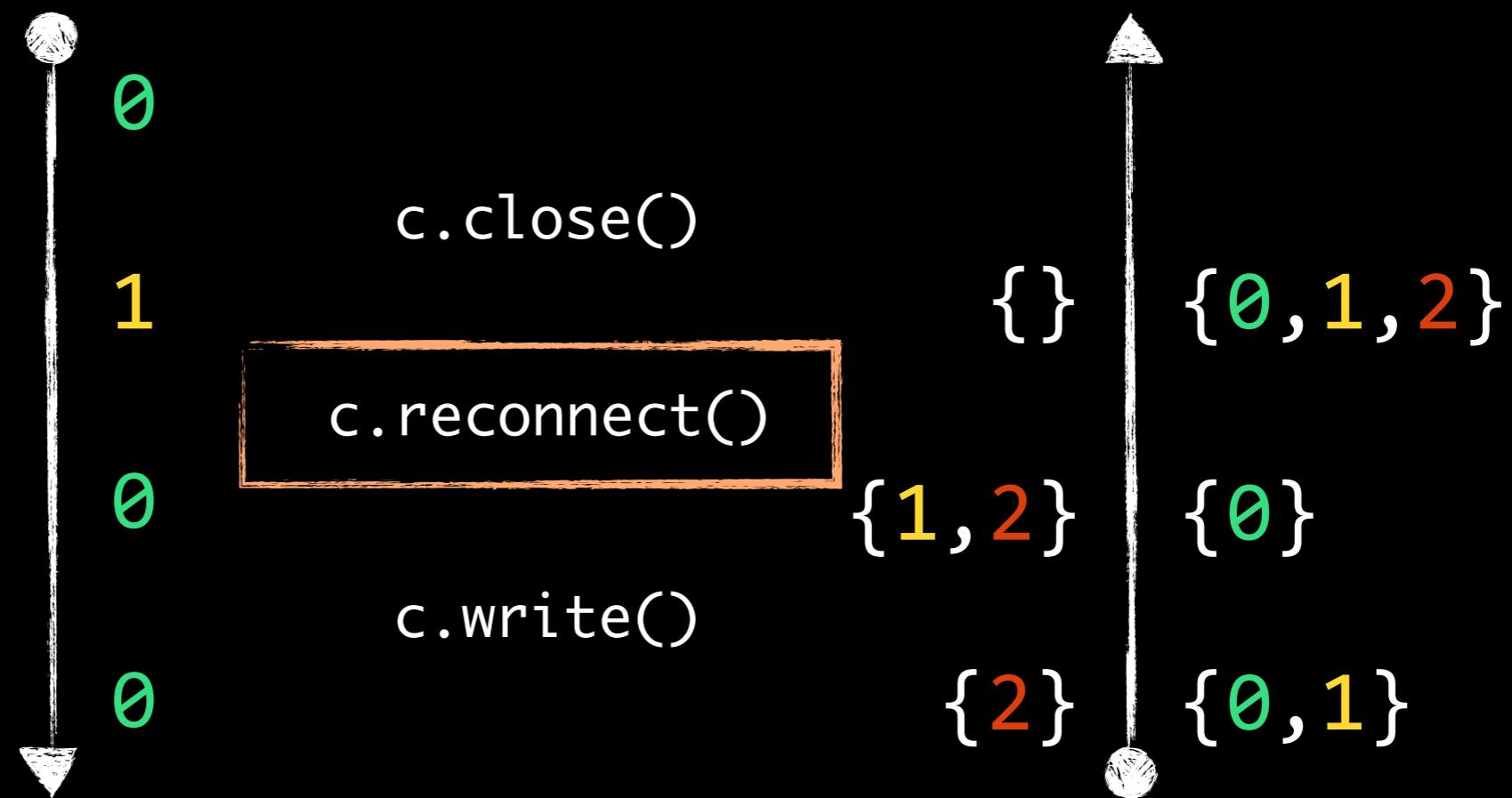


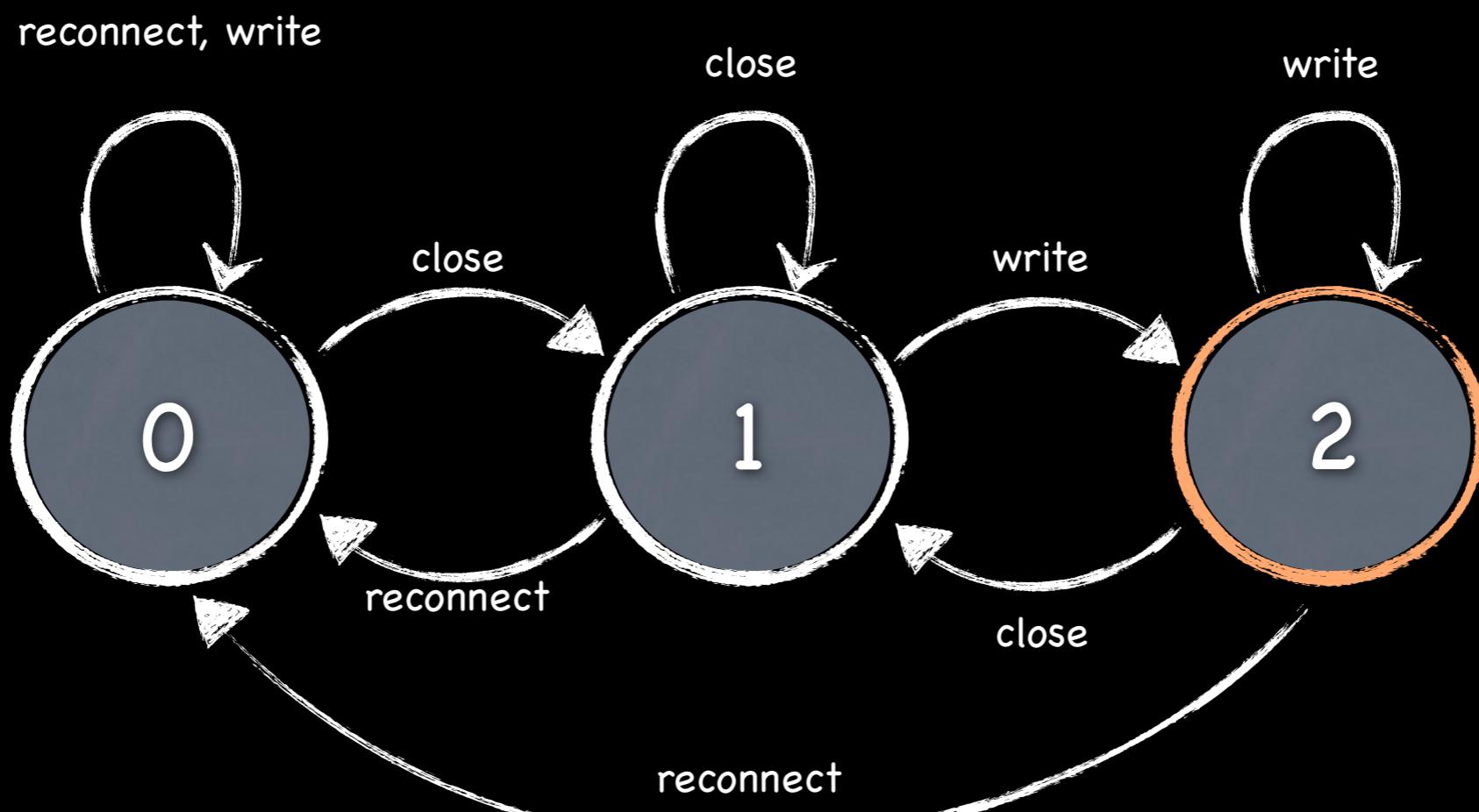
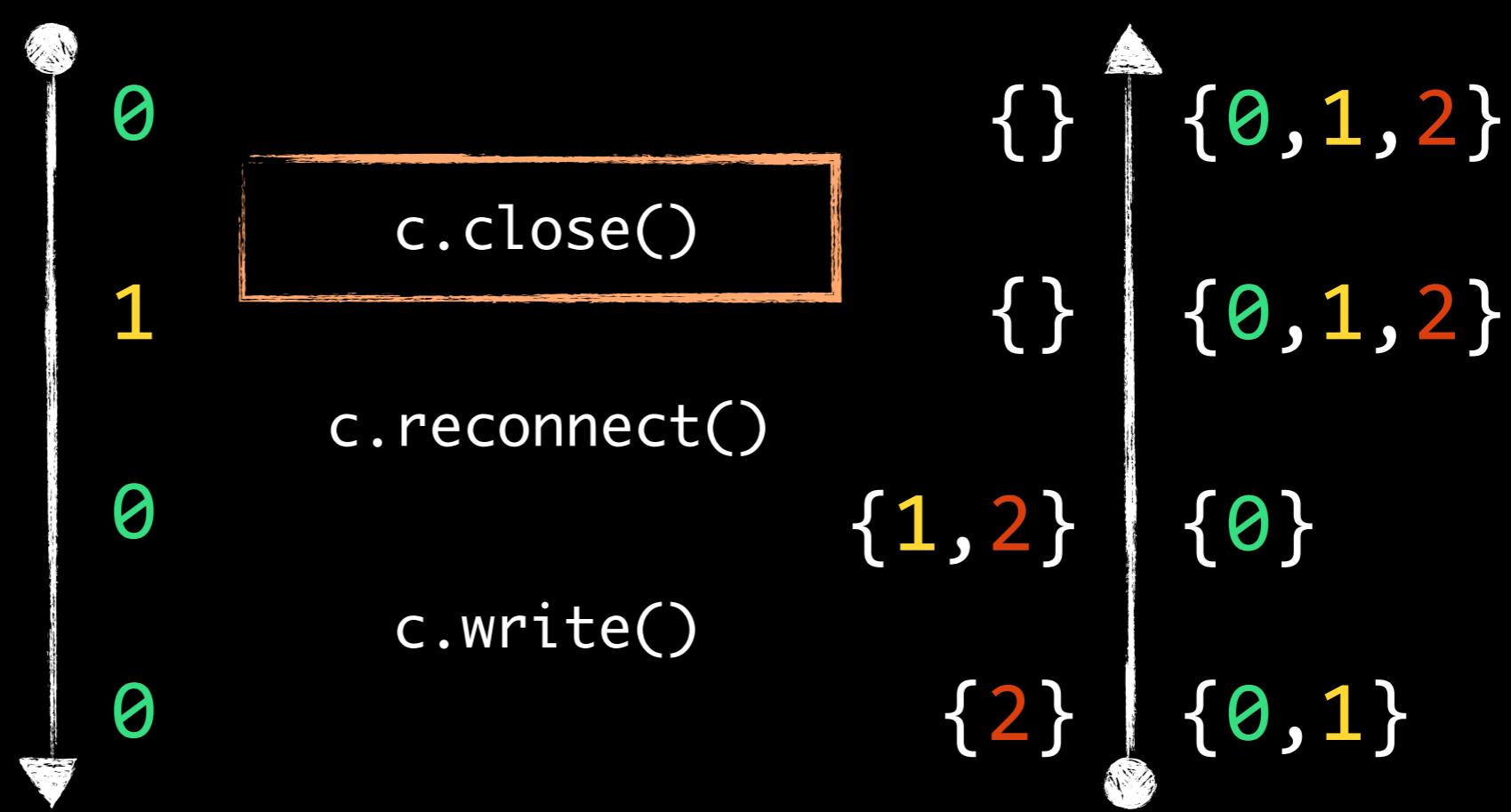


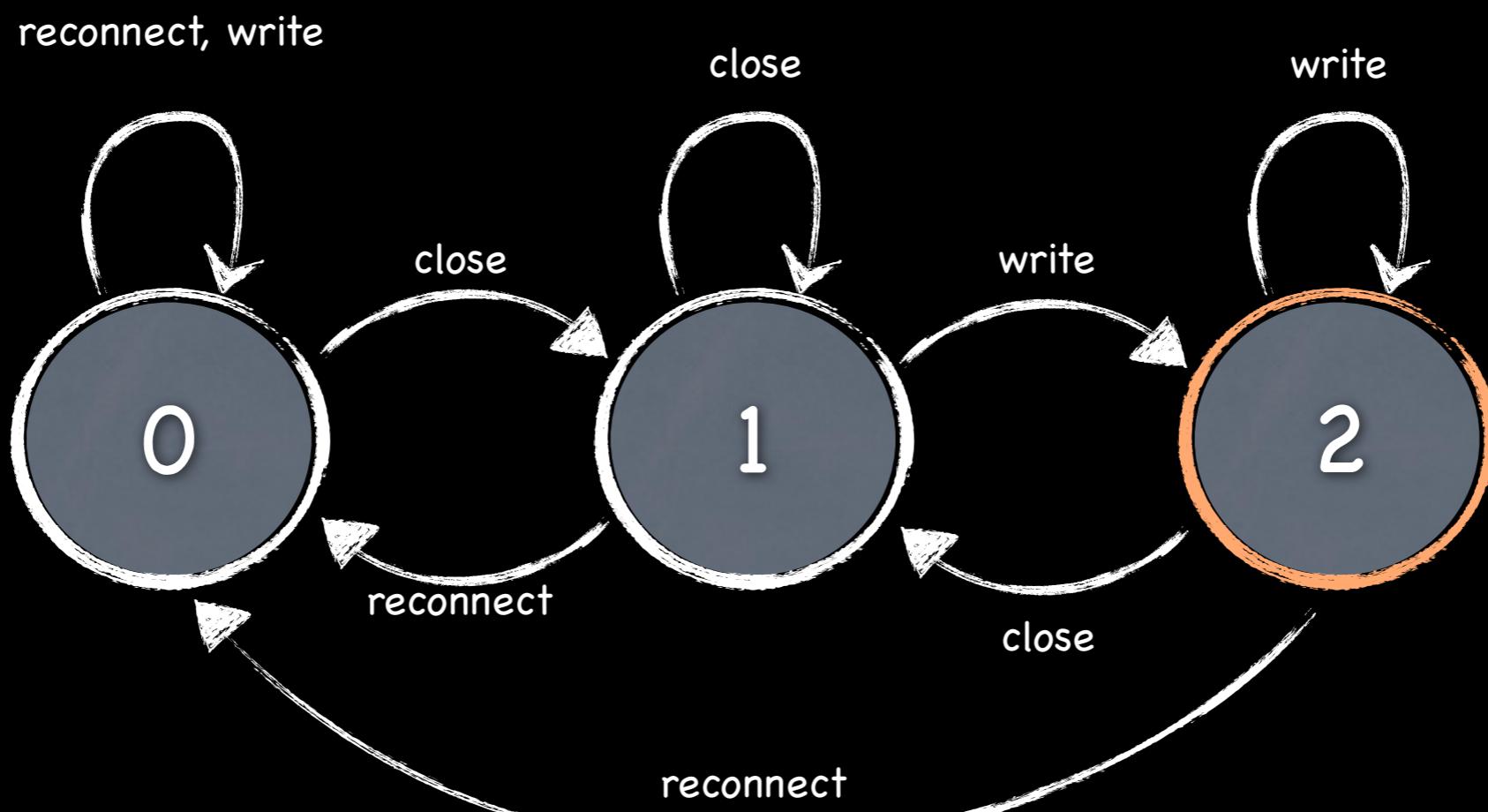
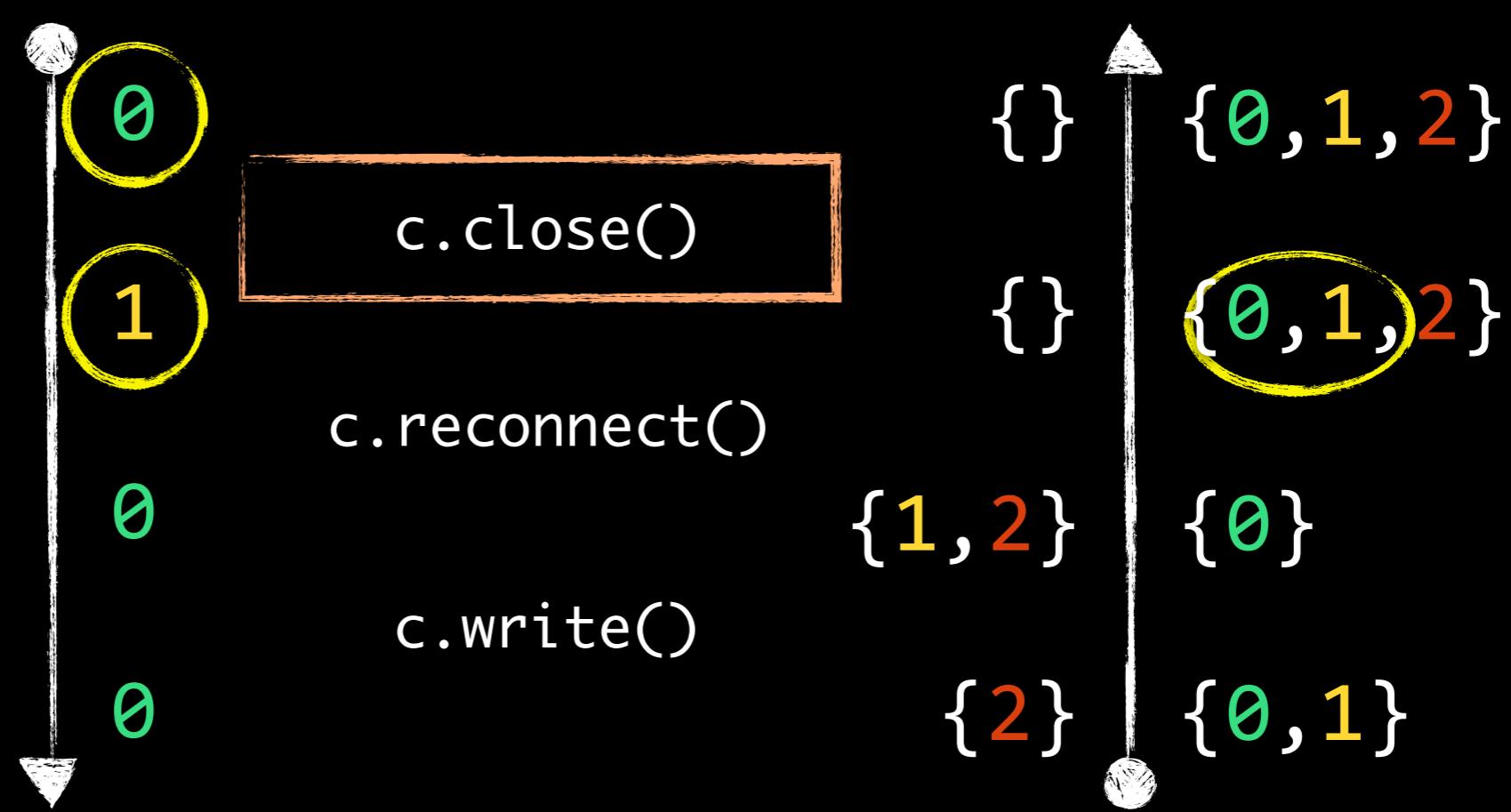


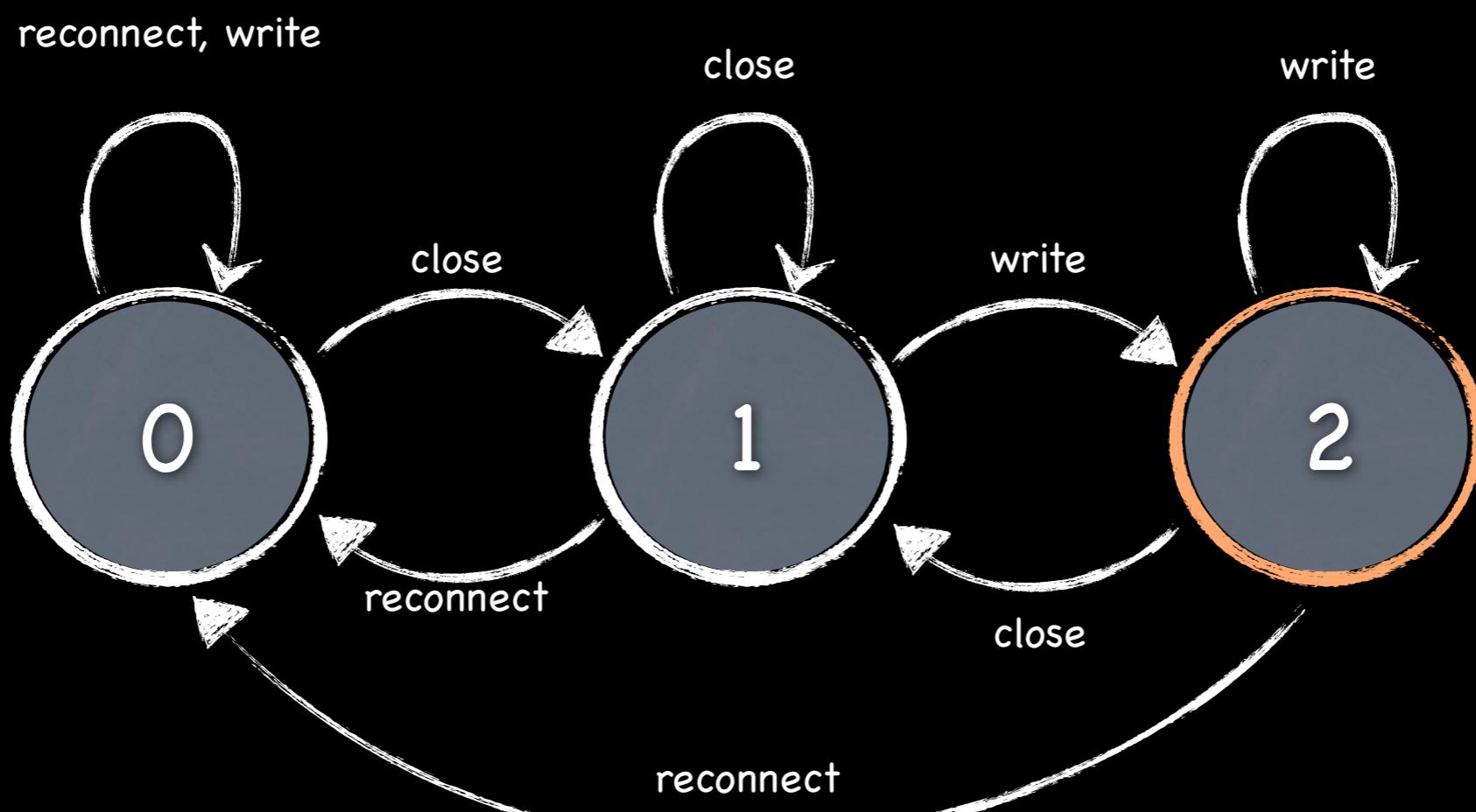
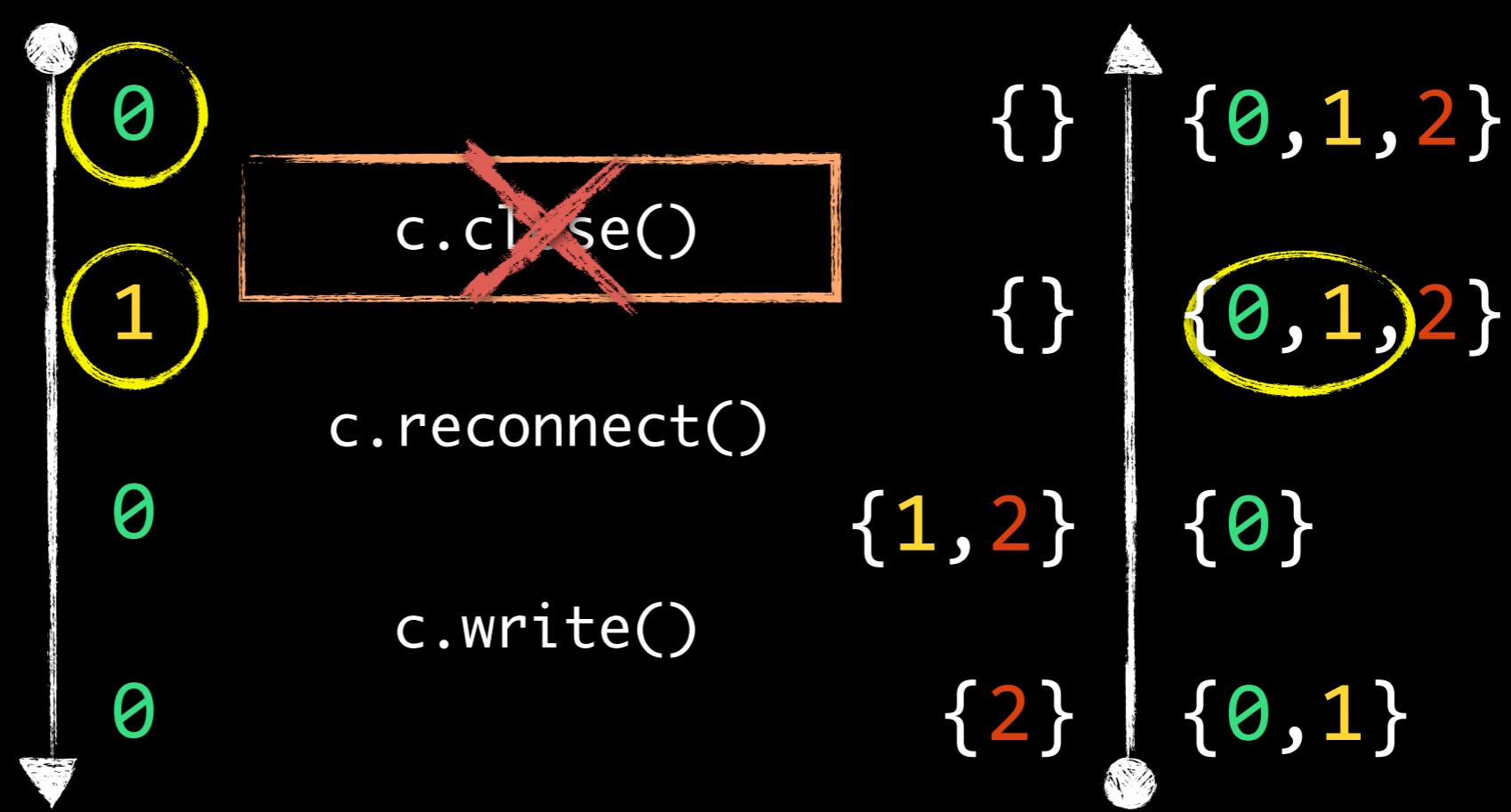


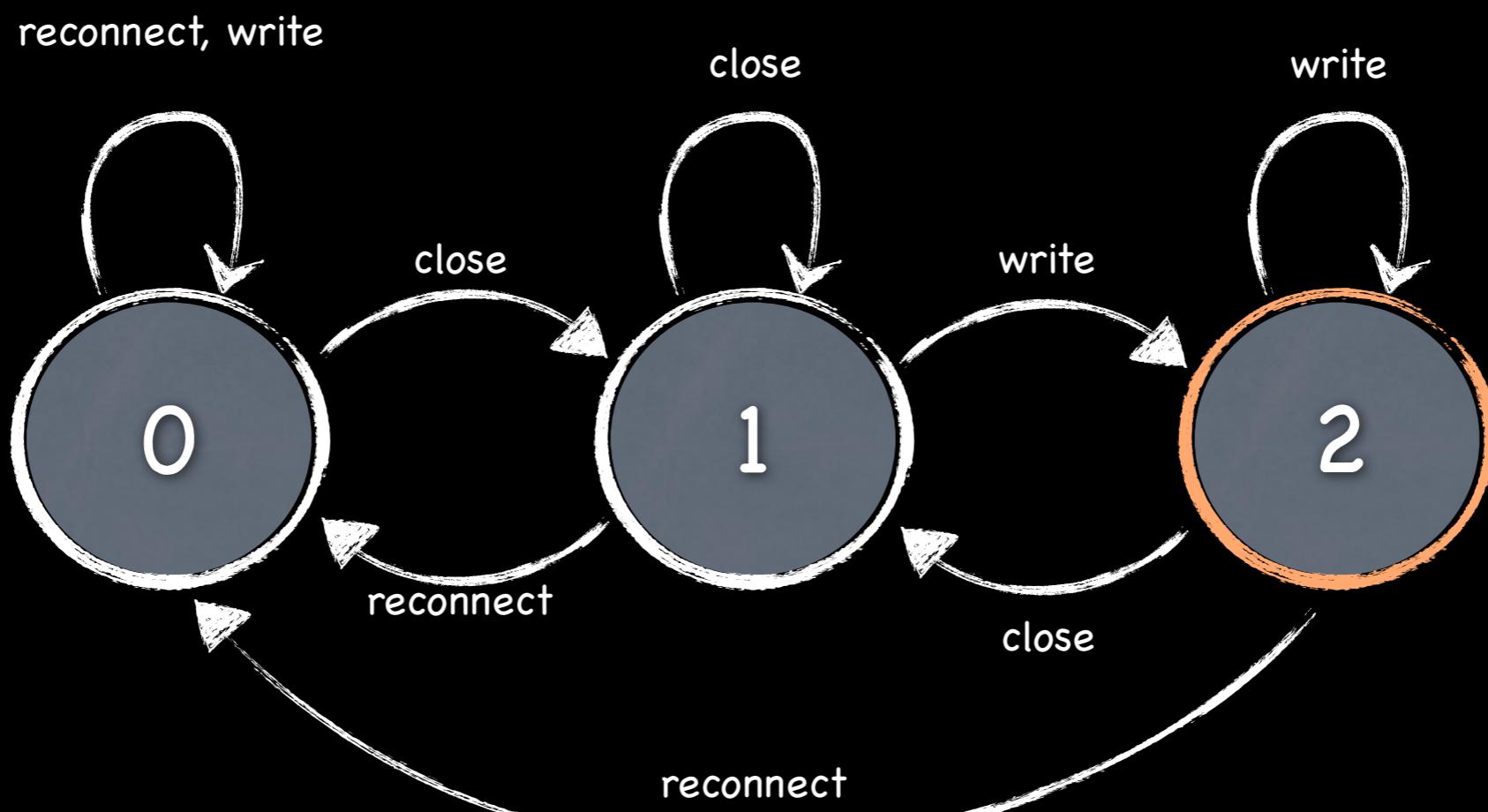
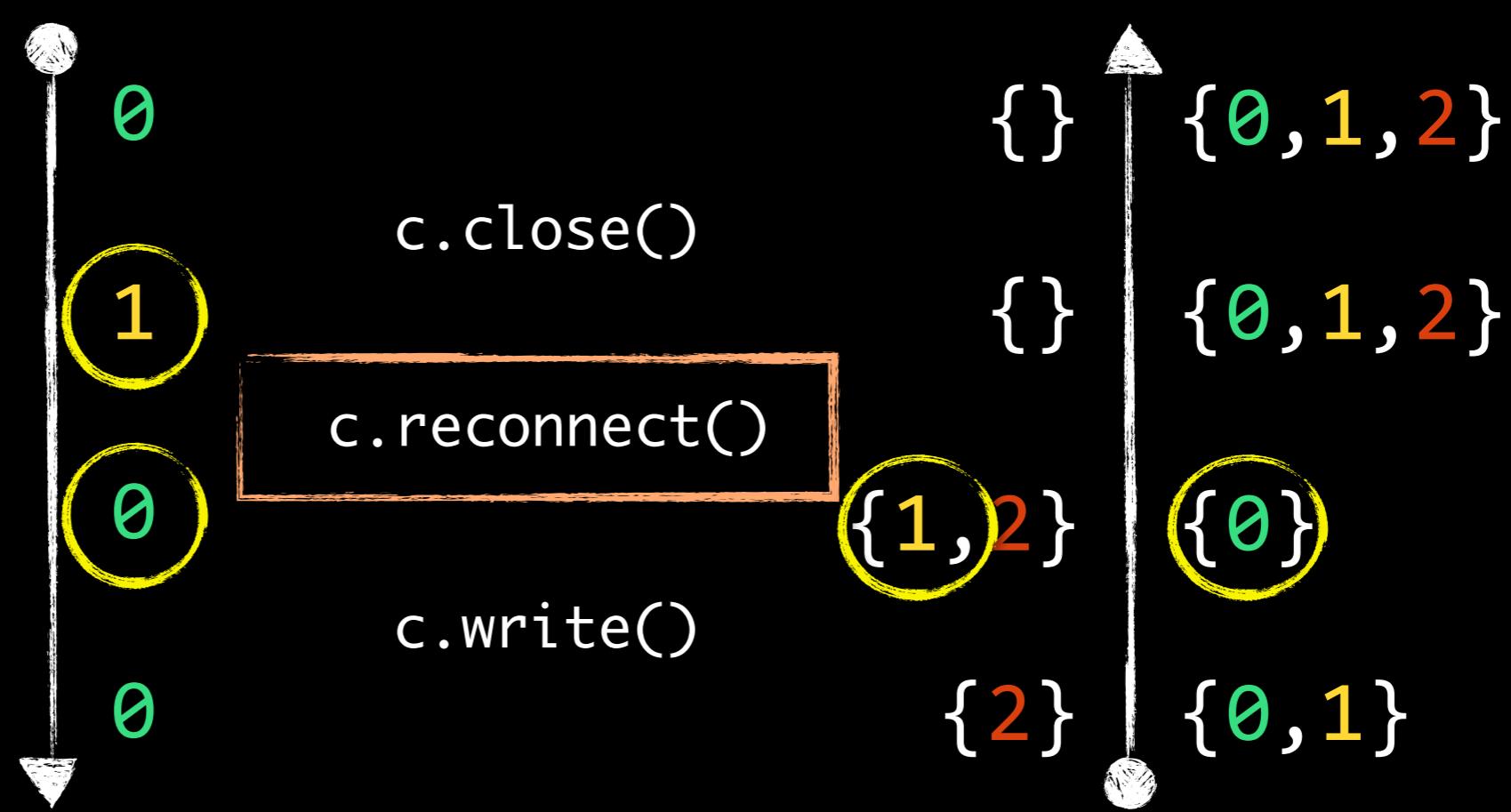


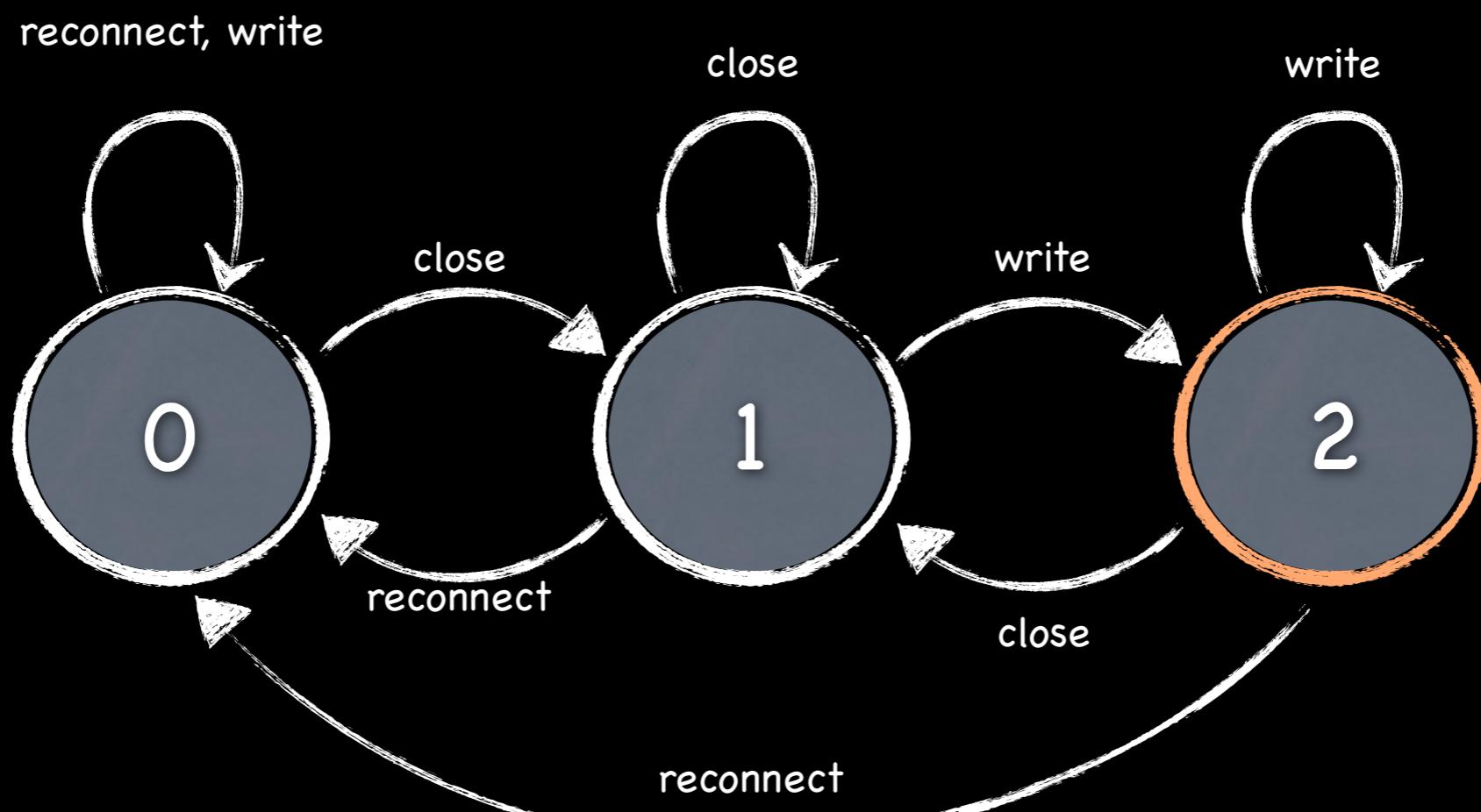
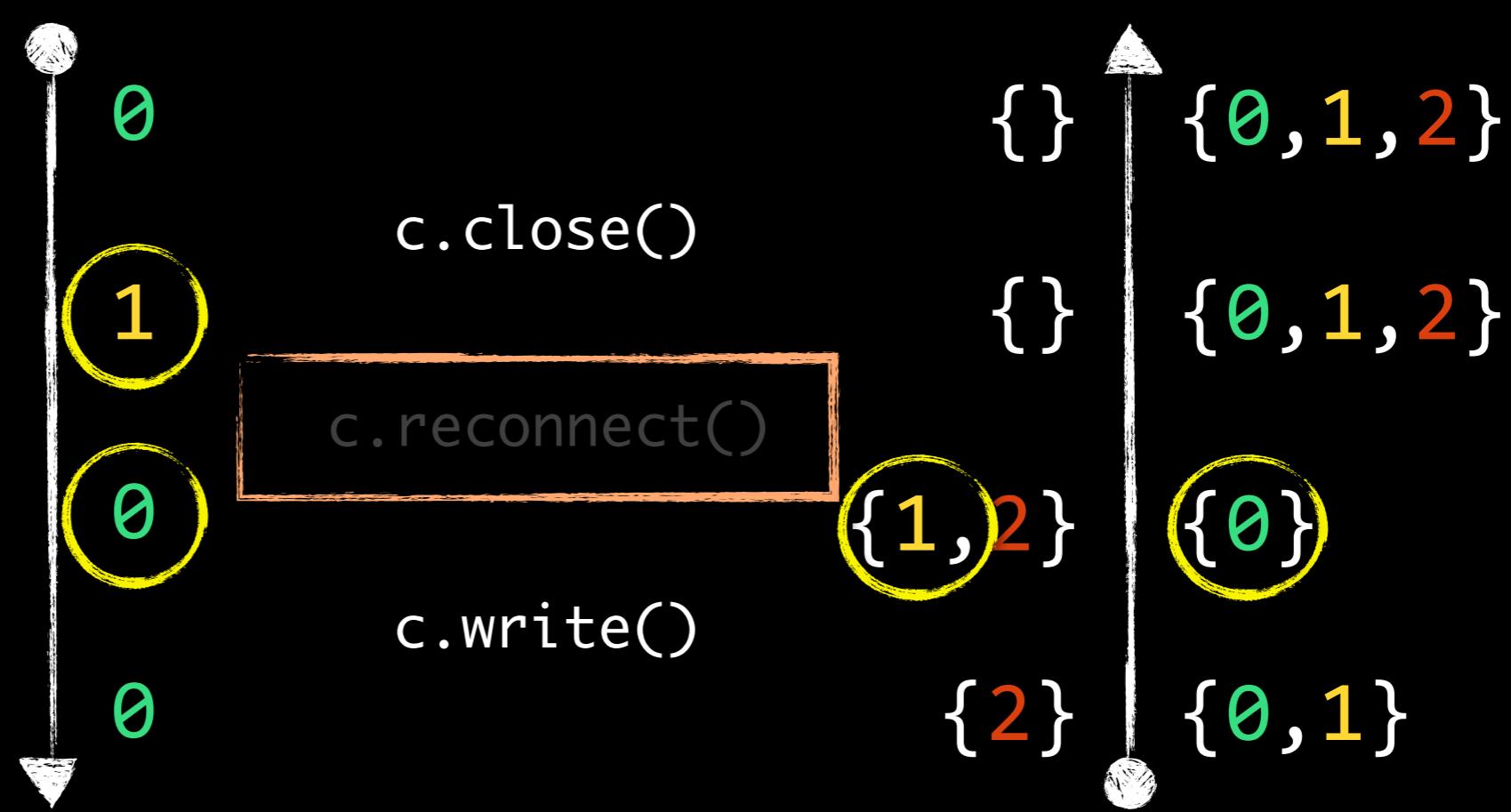


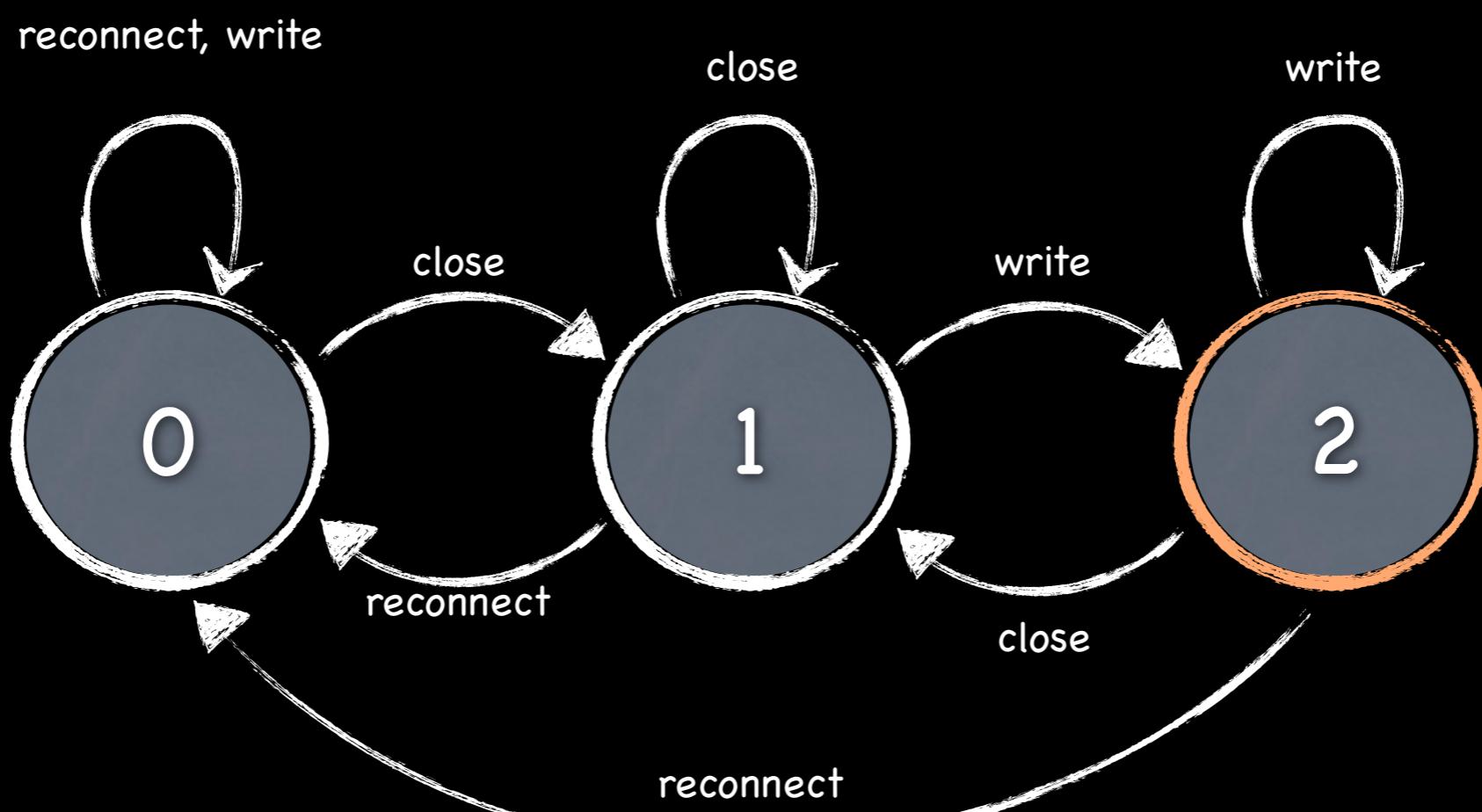
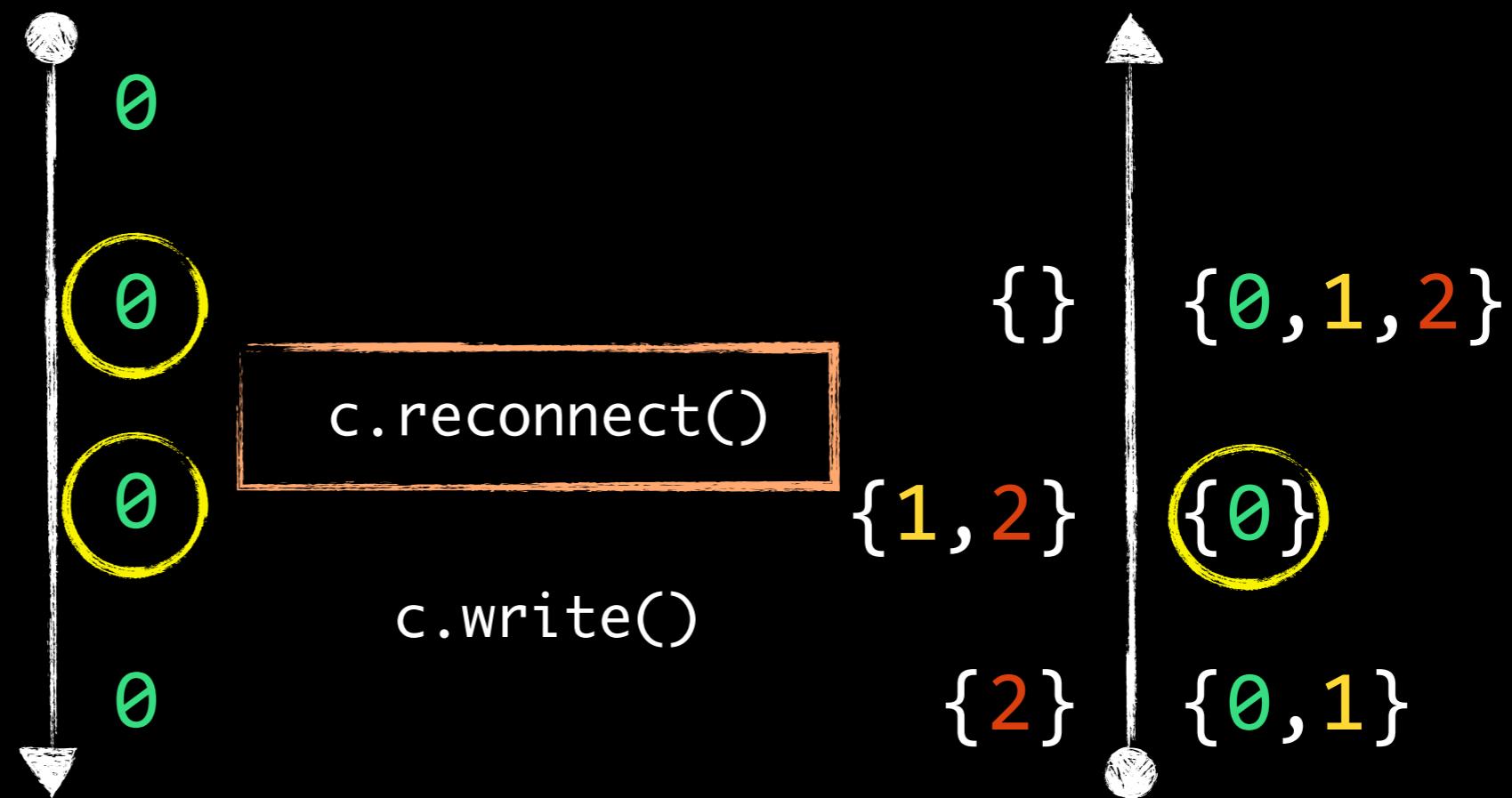


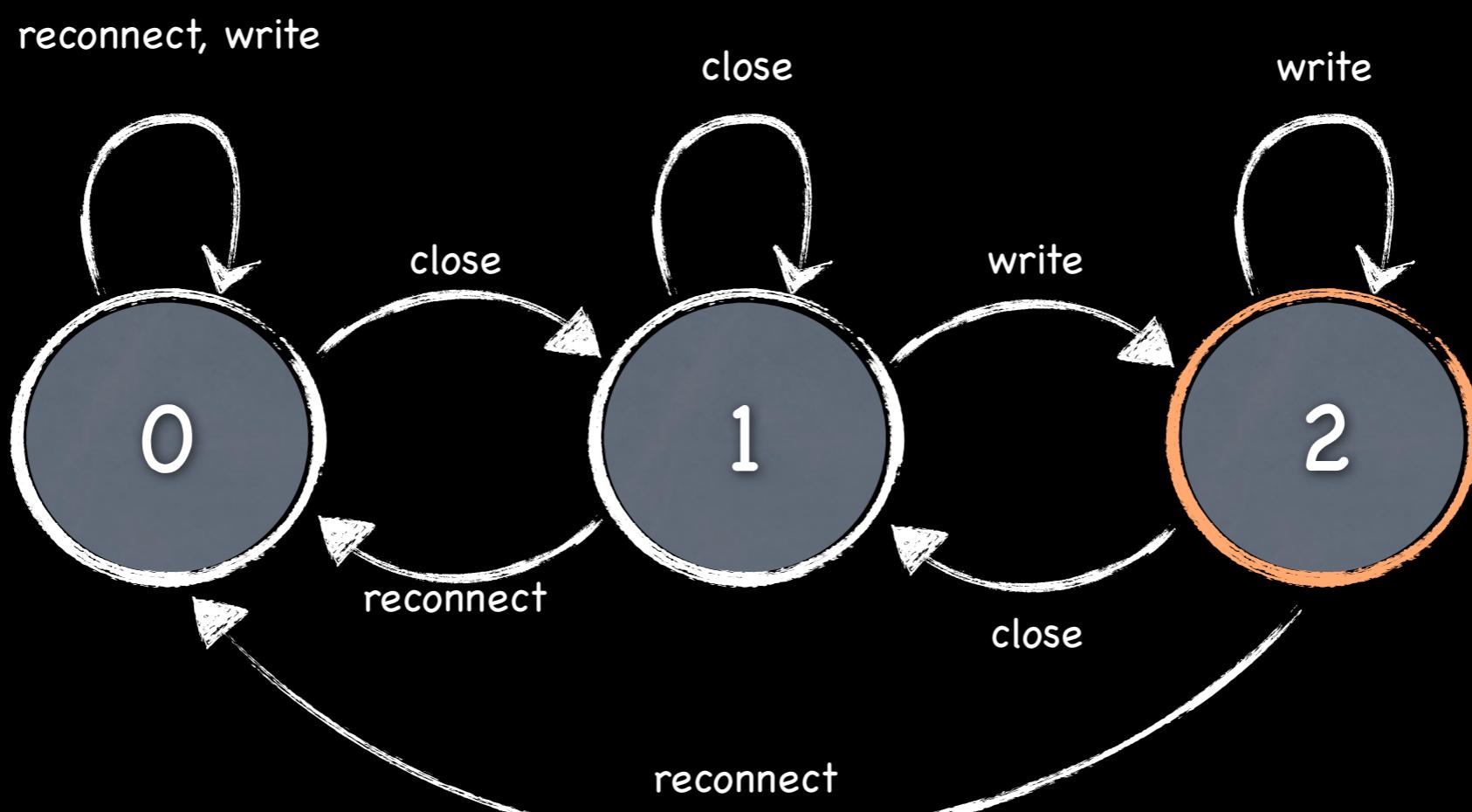
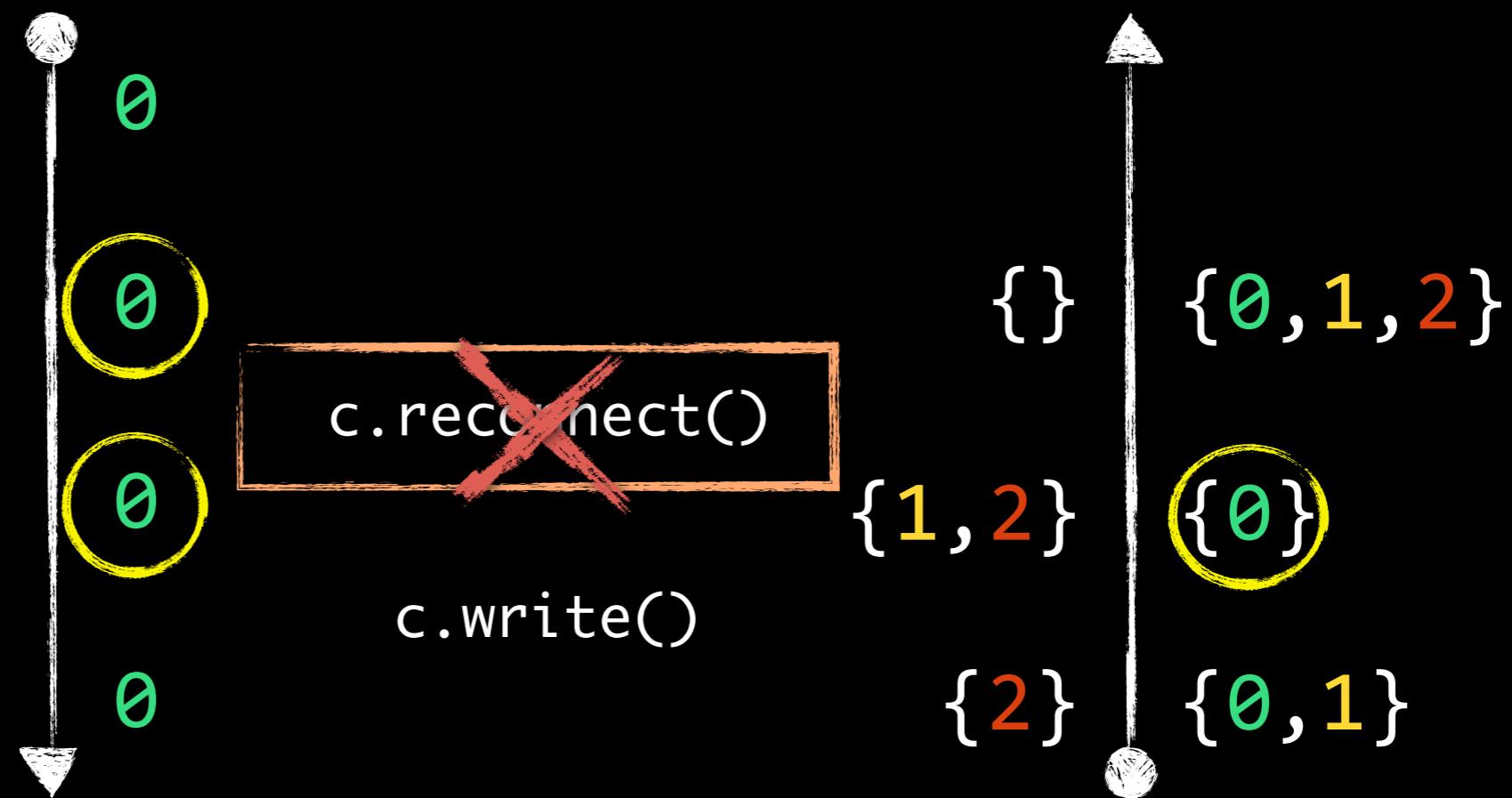


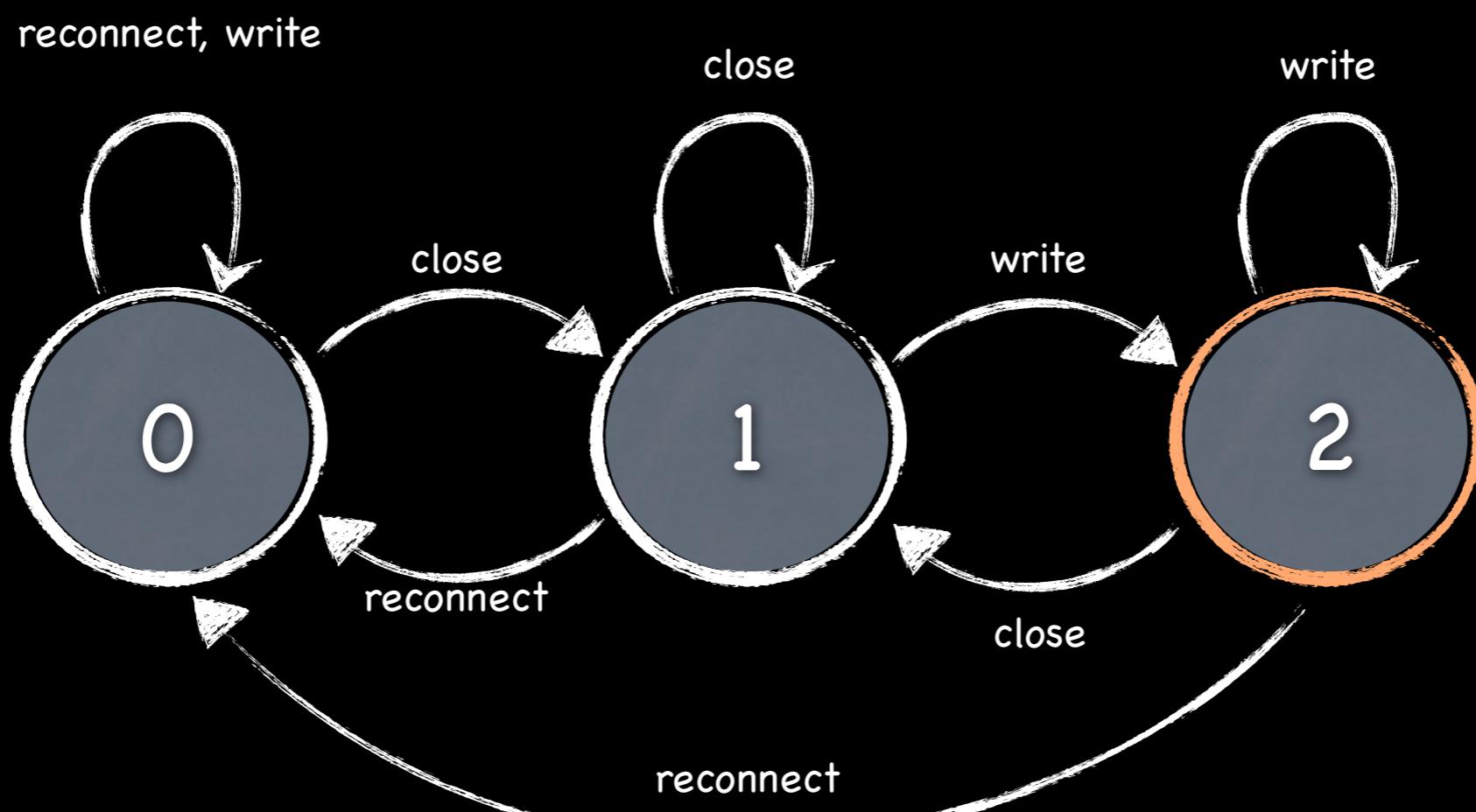
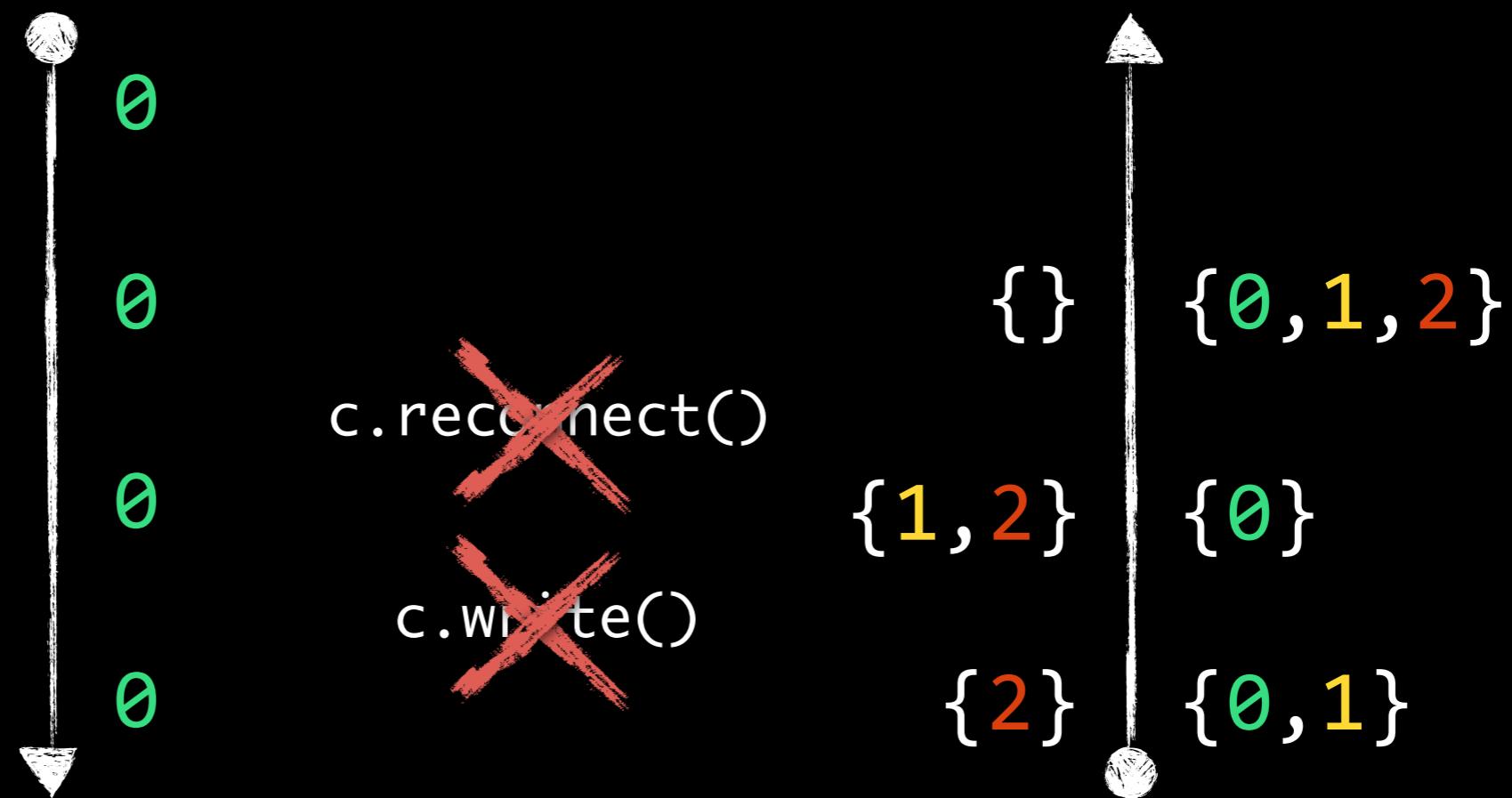


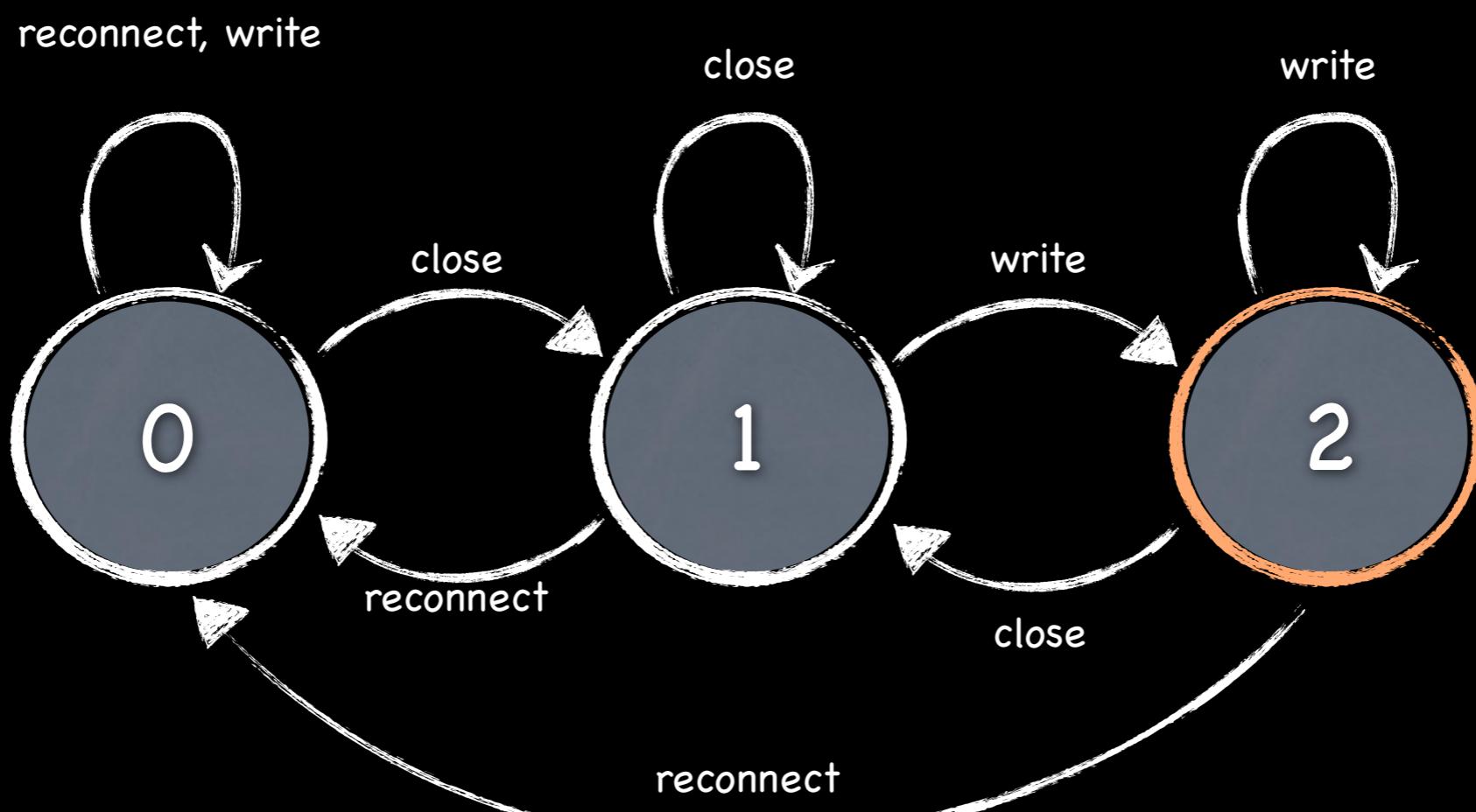
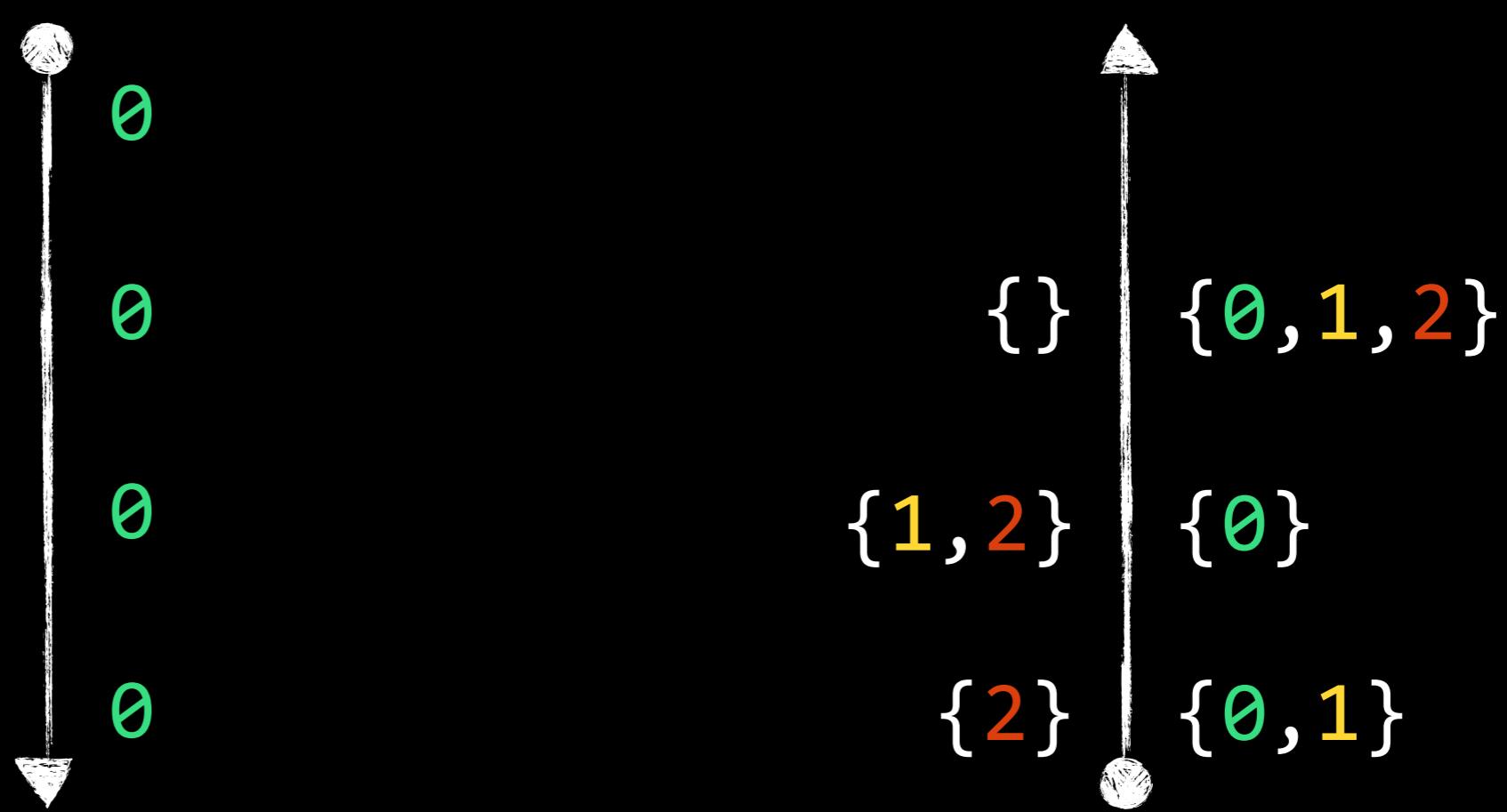




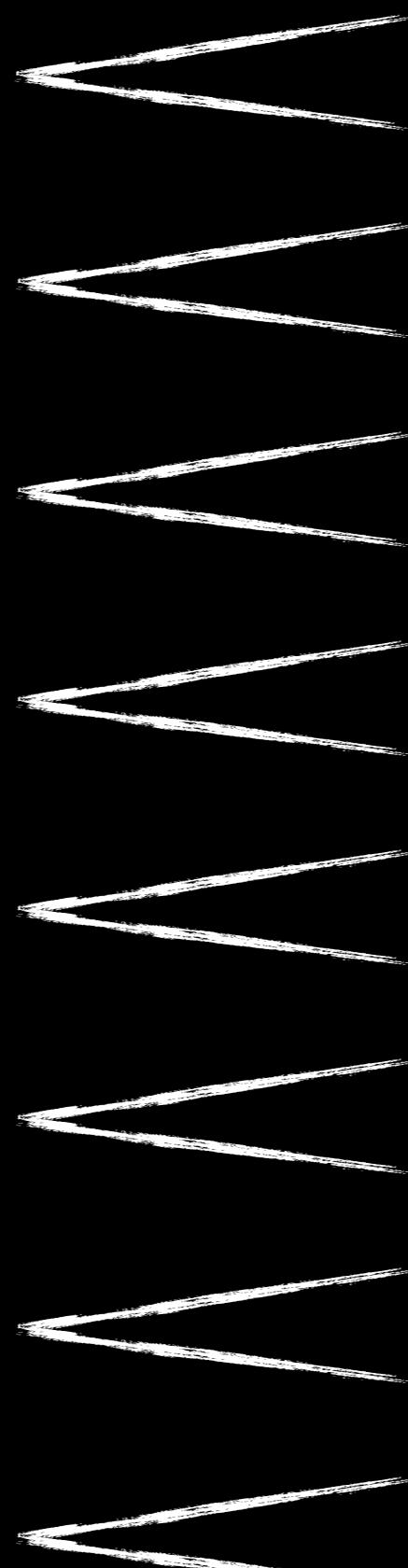








```
c1.close();
```



{ } {0,1,2}

{ } {0,1,2}

{ } {0,1,2}

{ } {0,1,2}

{ } {1,2}

{ } {2}

{ }

{1,2}

{2}

```
c1.reconnect();
```

```
c1.close();
```



0

{}

{0,1,2}

1

{}

{0,1,2}

```
c1.reconnect();
```



0

{}

{0,1,2}

```
c1.close();
```

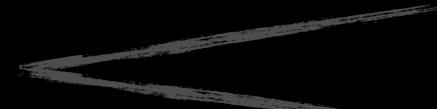


1

{}

{0,1,2}

```
c1.close();
```



1

{}

{1,2}

```
c1.write(..);
```



2

{}

{2}

```
c1.close();
```



1

{}

```
c1.reconnect();
```



0

{}

{1,2}

```
c1.write(..);
```

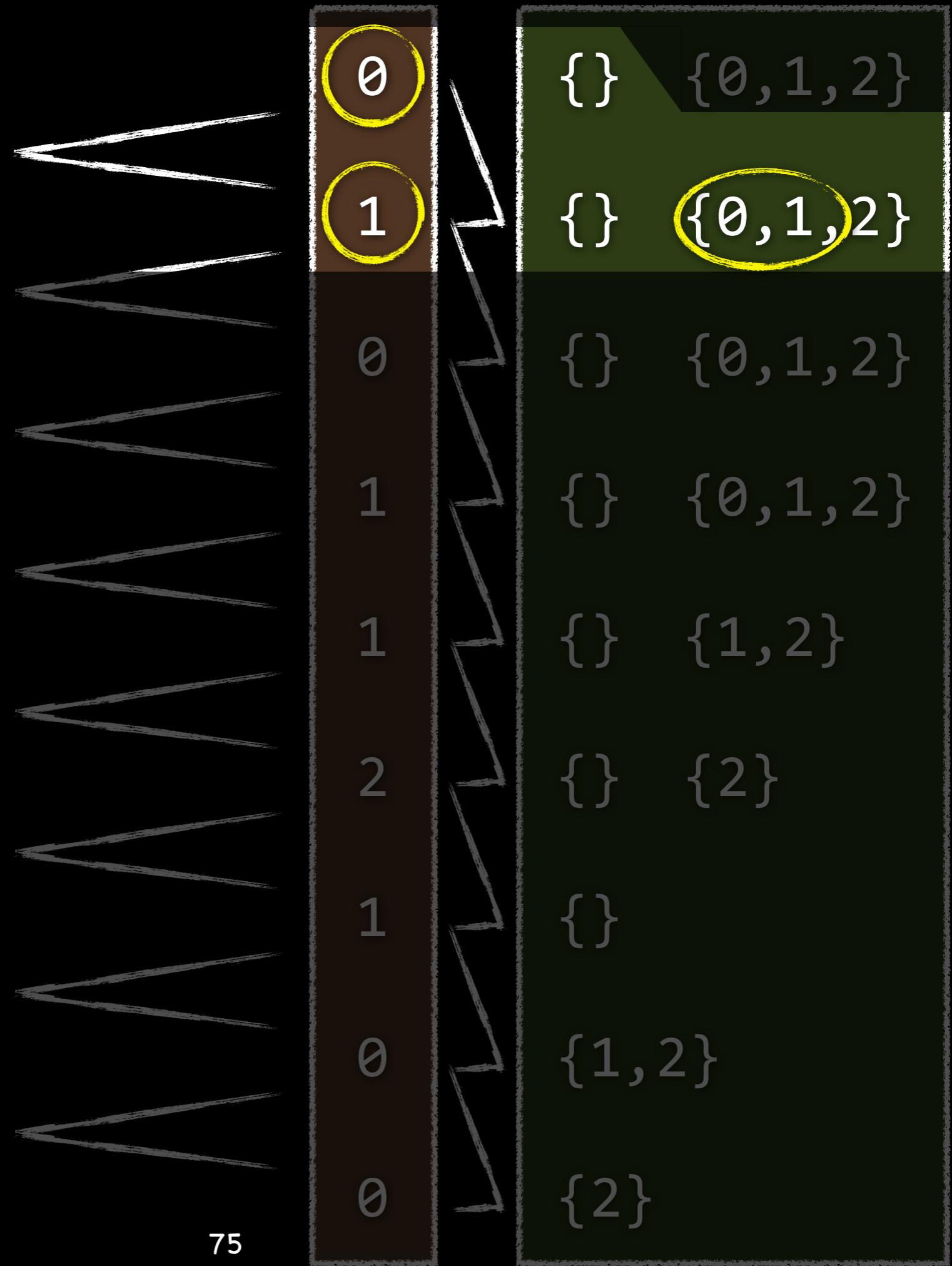


0

{}

{2}

```
c1.close();
```



```
c1.close();
```

```
c1.write(..);
```

```
c1.close();
```

```
c1.reconnect();
```

```
c1.write(..);
```

~~c1.close();~~

c1.reconnect();

c1.close();

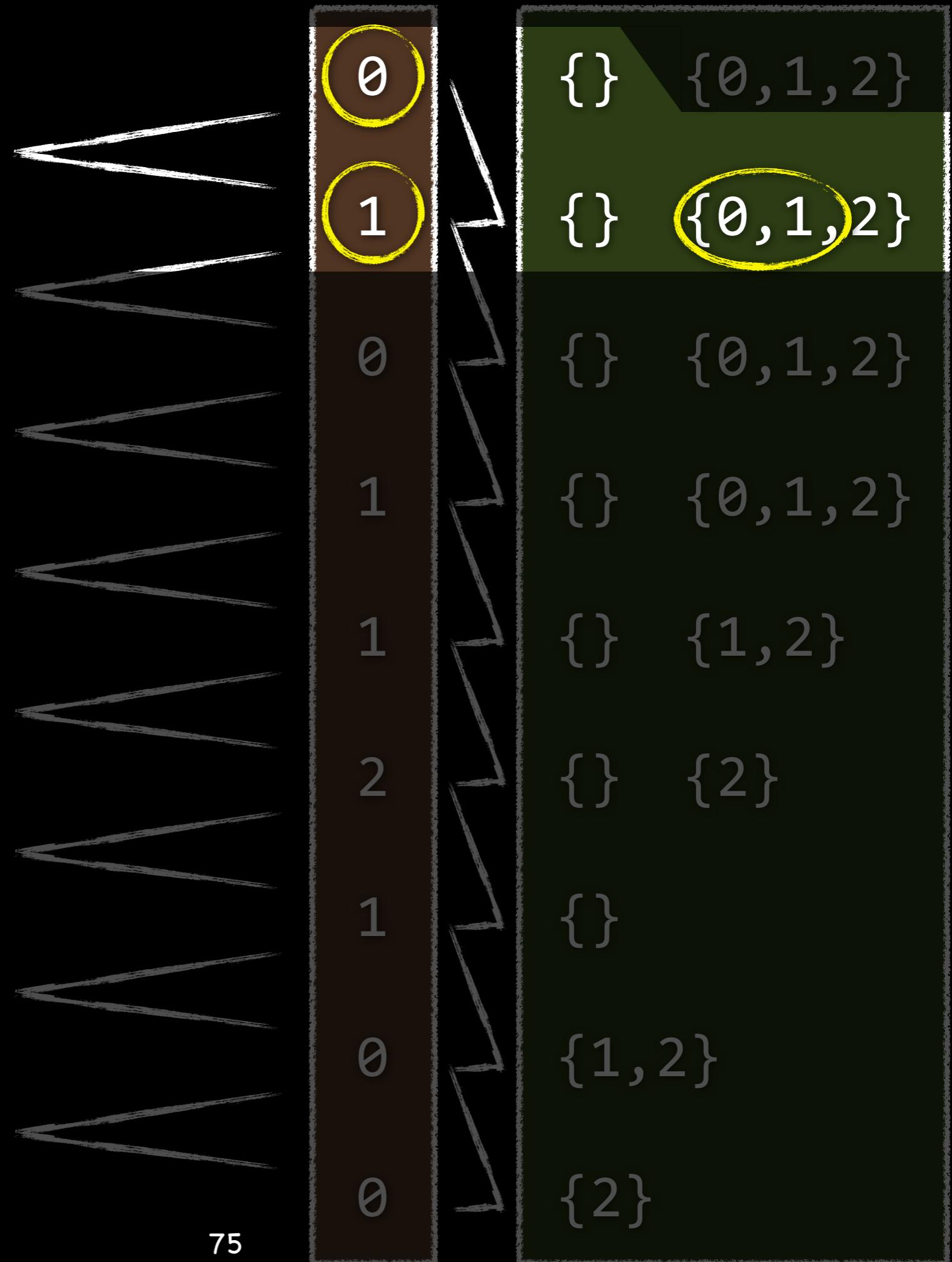
c1.close();

c1.write(..);

c1.close();

c1.reconnect();

c1.write(..);



~~c1.close();~~

c1.reconnect();

c1.close();

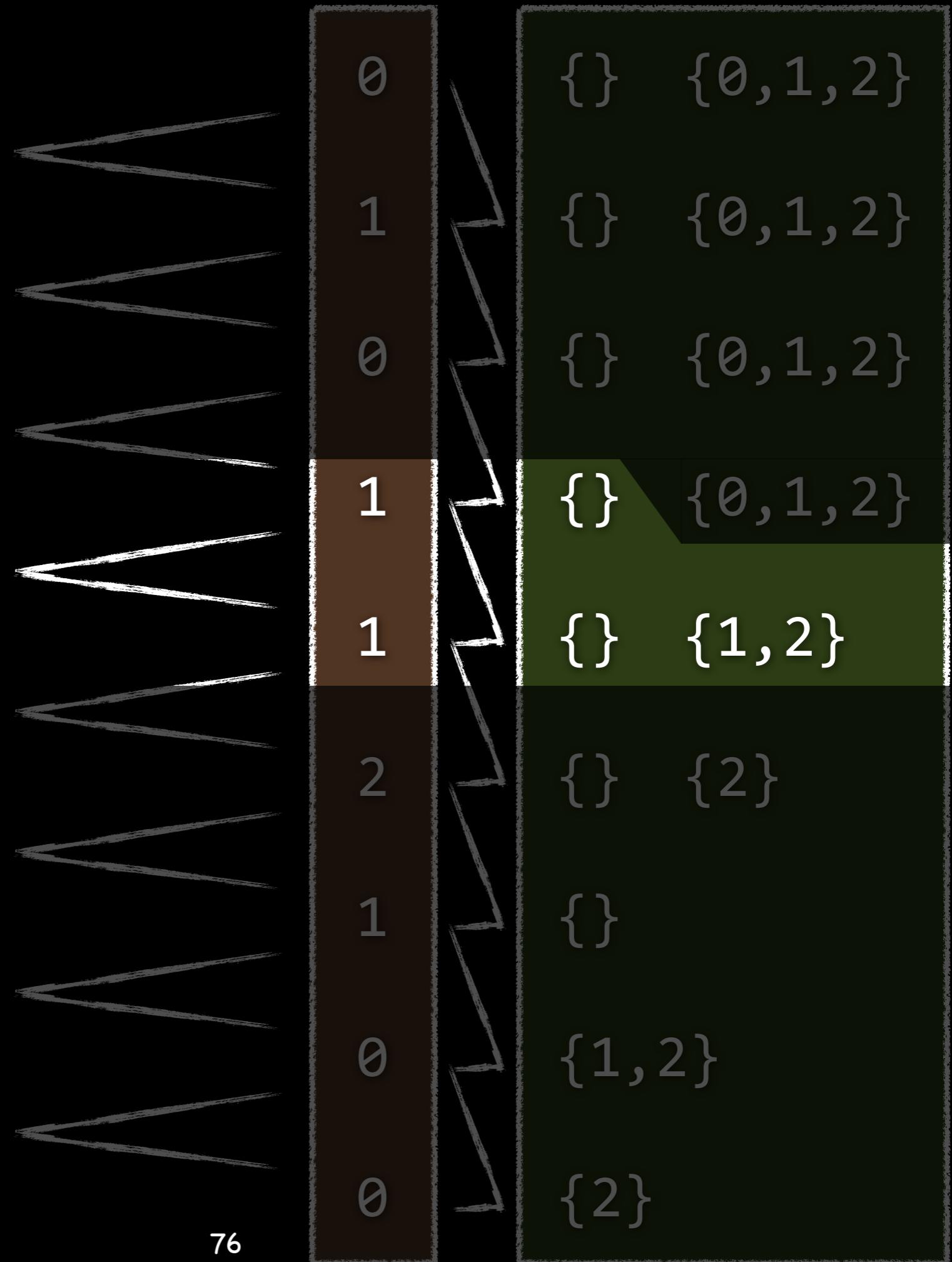
c1.close();

c1.write(..);

c1.close();

c1.reconnect();

c1.write(..);



~~c1.close();~~

c1.reconnect();

c1.close();

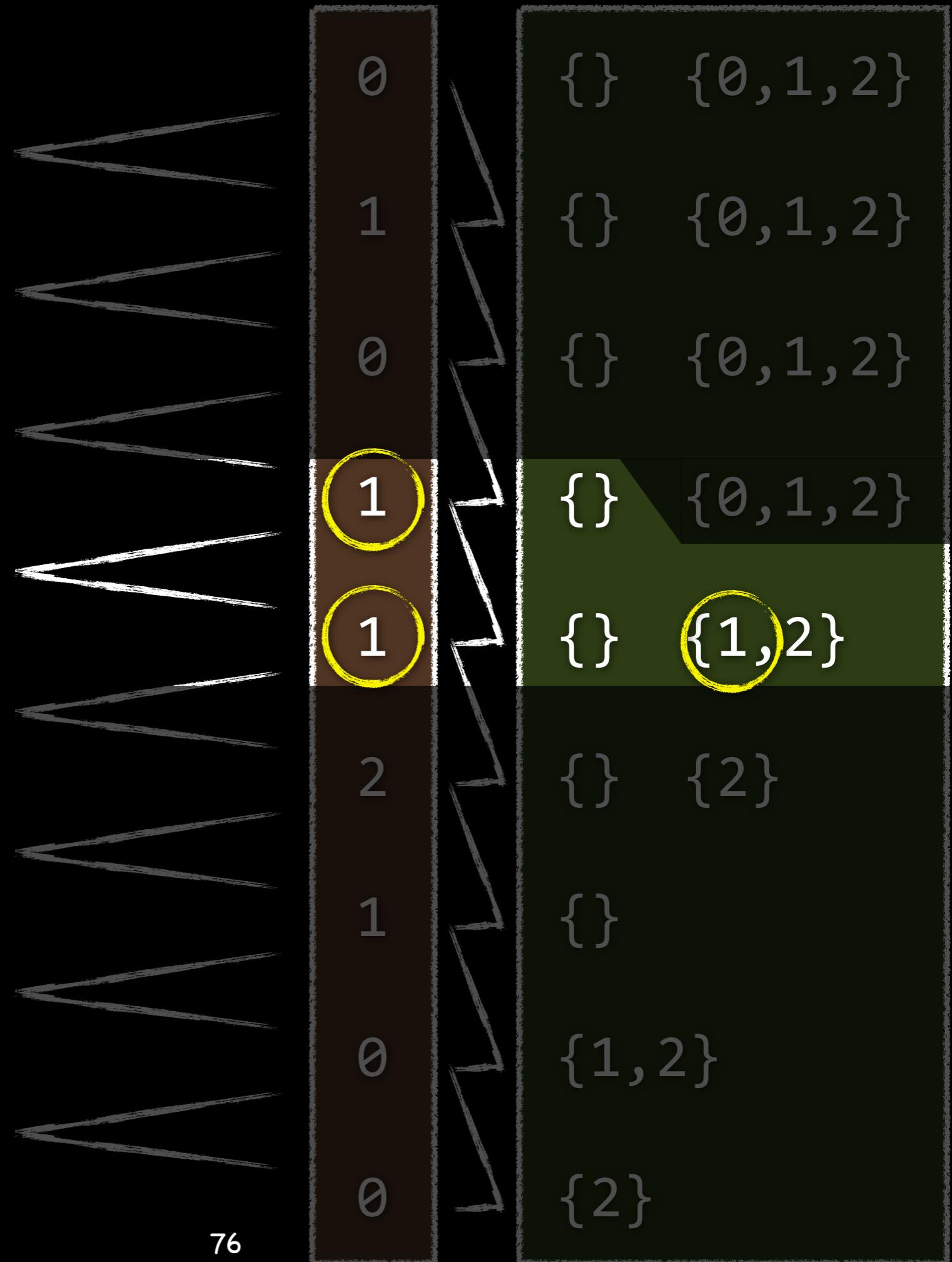
c1.close();

c1.write(...);

c1.close();

c1.reconnect();

c1.write(...);



~~c1.close();~~

c1.reconnect();

c1.close();

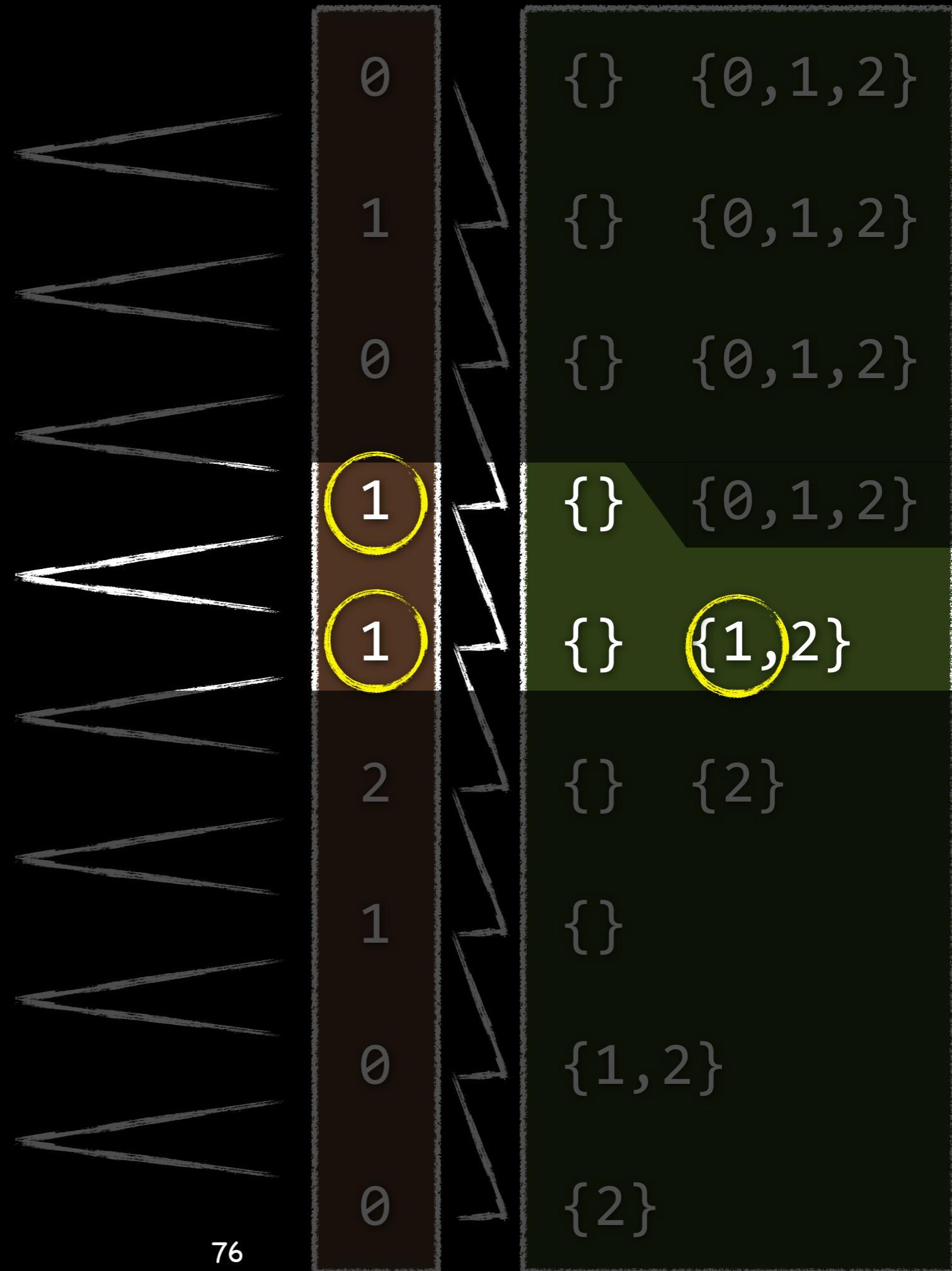
~~c1.close();~~

c1.write(...);

c1.close();

c1.reconnect();

c1.write(...);



~~c1.close();~~

c1.reconnect();

c1.close();

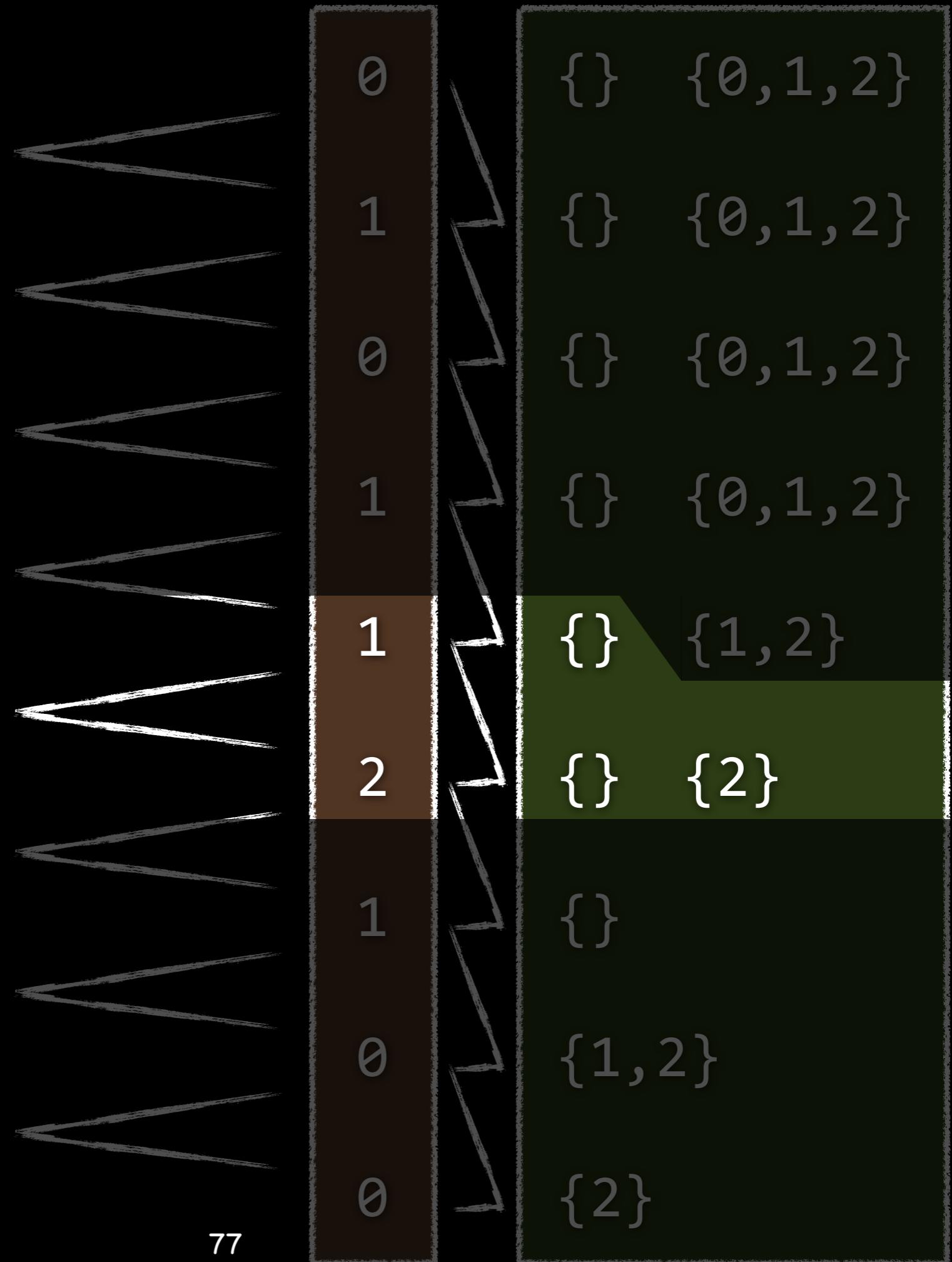
~~c1.close();~~

c1.write(..);

c1.close();

c1.reconnect();

c1.write(..);



~~c1.close();~~

c1.reconnect();

c1.close();

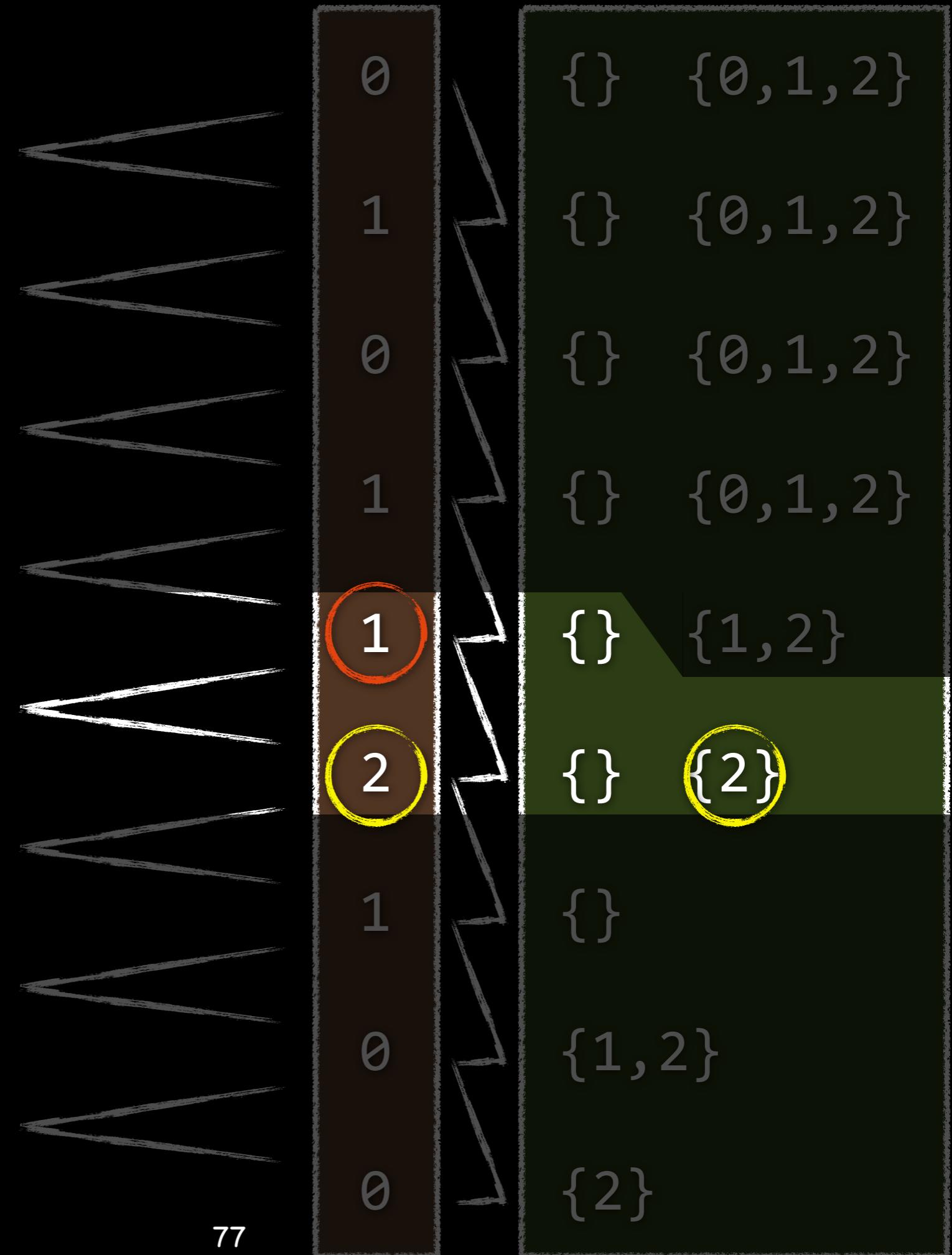
~~c1.close();~~

c1.write(..);

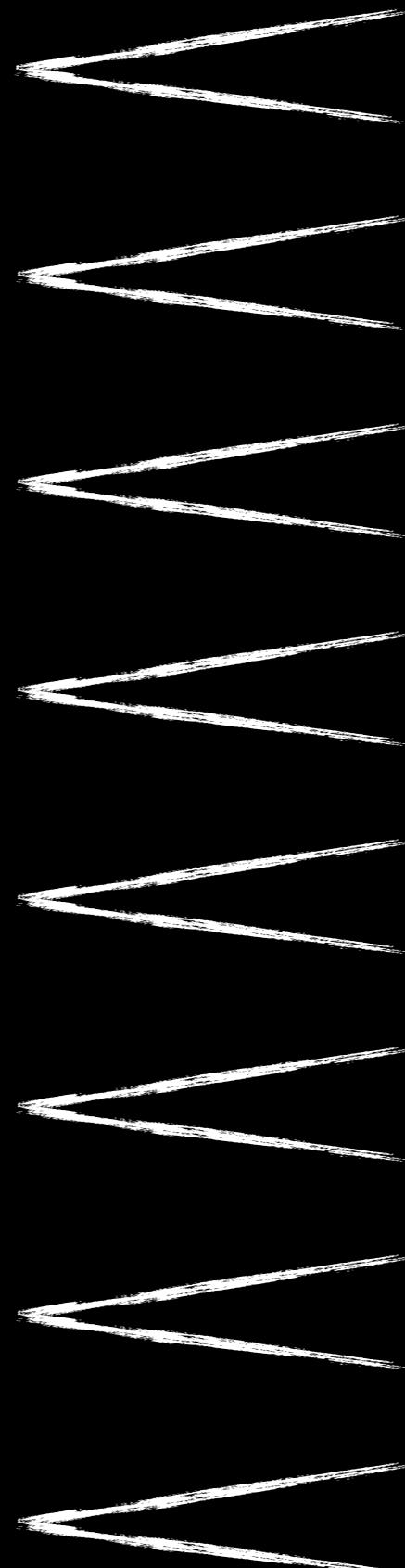
c1.close();

c1.reconnect();

c1.write(..);



```
c1.close();
```



0	{}	{0,1,2}
1	{}	{0,1,2}
0	{}	{0,1,2}
1	{}	{0,1,2}
1	{}	{1,2}
2	{}	{2}
1	{}	
0	{1,2}	
0	{2}	

```
c1.reconnect();
```

```
c1.close();
```

```
c1.close();
```

```
c1.write(..);
```

```
c1.close();
```

```
c1.reconnect();
```

```
c1.write(..);
```

~~c1.close();~~

~~c1.reconnect();~~

~~c1.close();~~

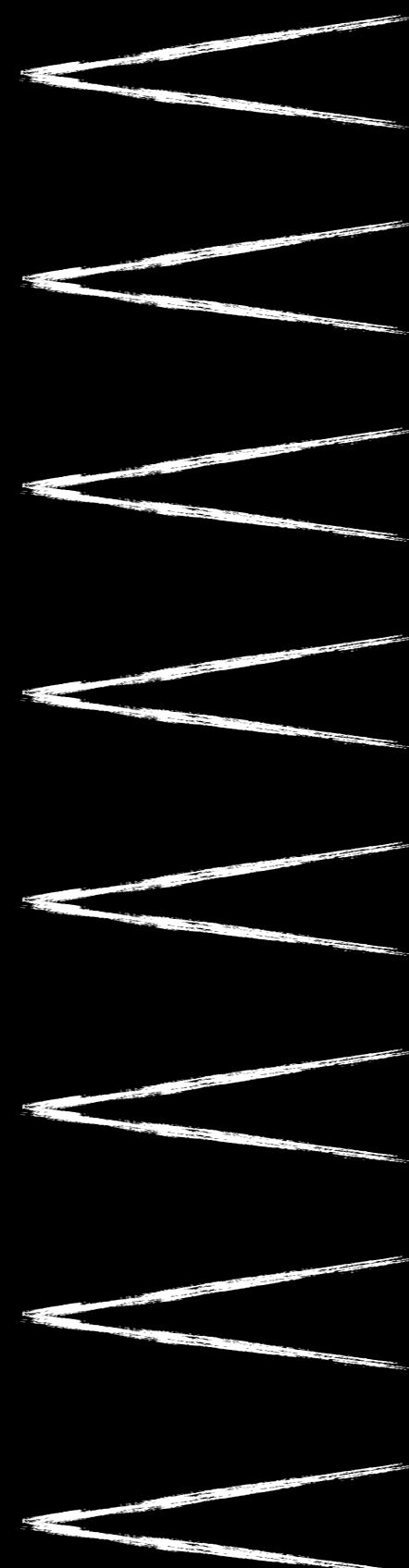
~~c1.close();~~

c1.write(..);

~~c1.close();~~

c1.reconnect();

~~c1.write(..);~~



```
c1.close();
```

```
c1.reconnect();
```

```
c1.close();
```

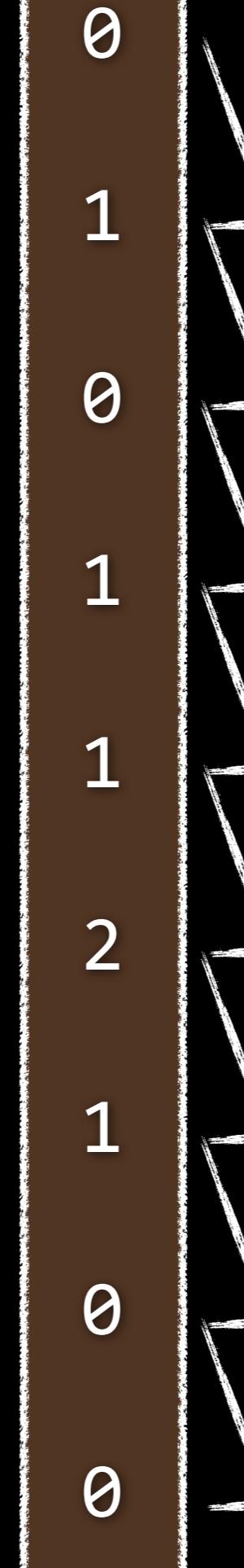
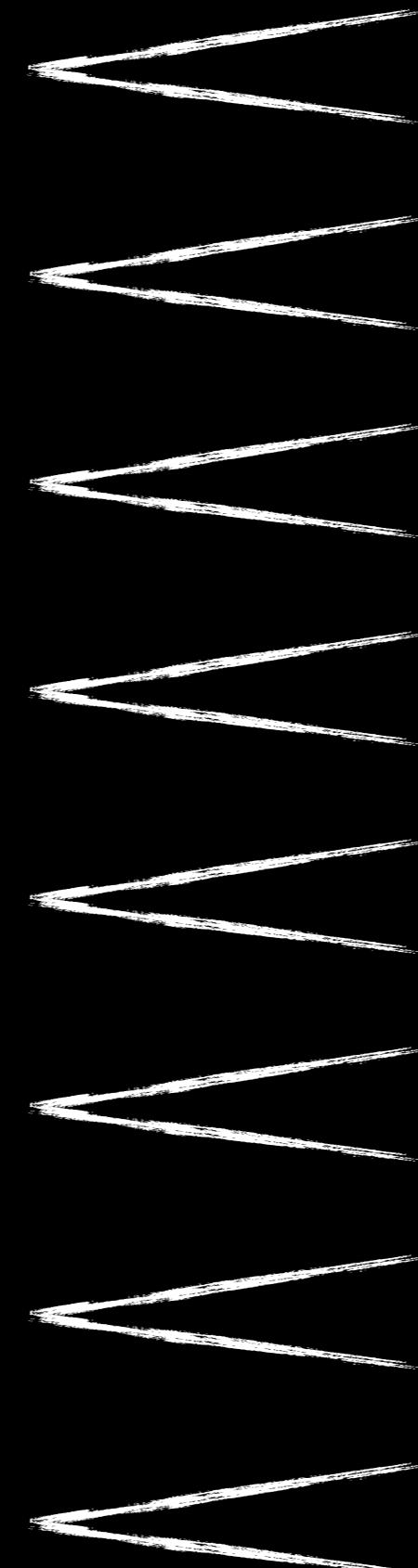
~~```
c1.close();
```~~

```
c1.write(..);
```

```
c1.close();
```

```
c1.reconnect();
```

```
c1.write(..);
```



```
c1.close();
```



0

{ } {0,1,2}

```
c1.reconnect();
```



1

{ } {0,1,2}

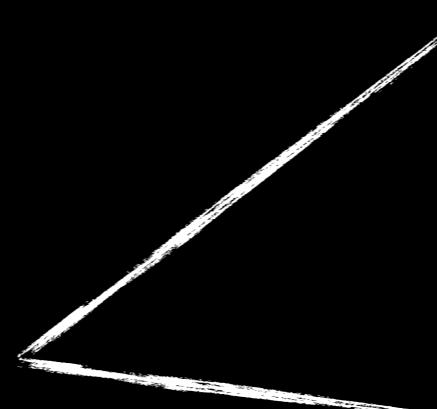
```
c1.close();
```



0

{ } {0,1,2}

```
c1.write(..);
```



1

{ } {1,2}

```
c1.close();
```



2

{ } {2}

```
c1.reconnect();
```



1

{ }

```
c1.write(..);
```



0

{1,2}  
{2}

~~c1.close();~~

~~c1.reconnect();~~

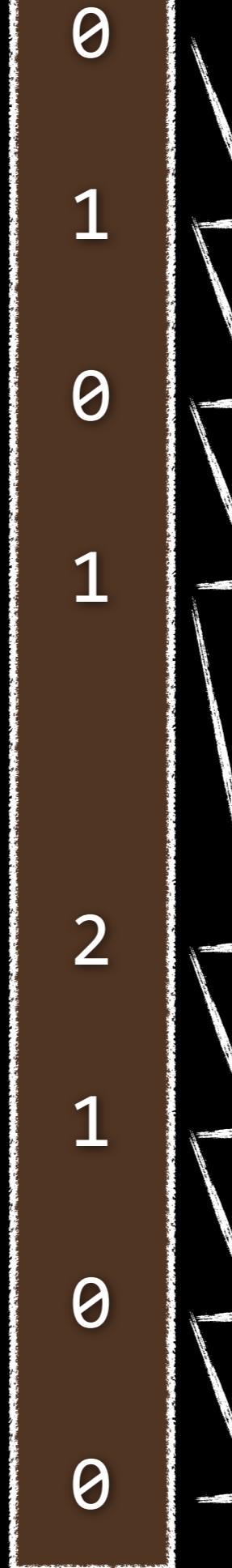
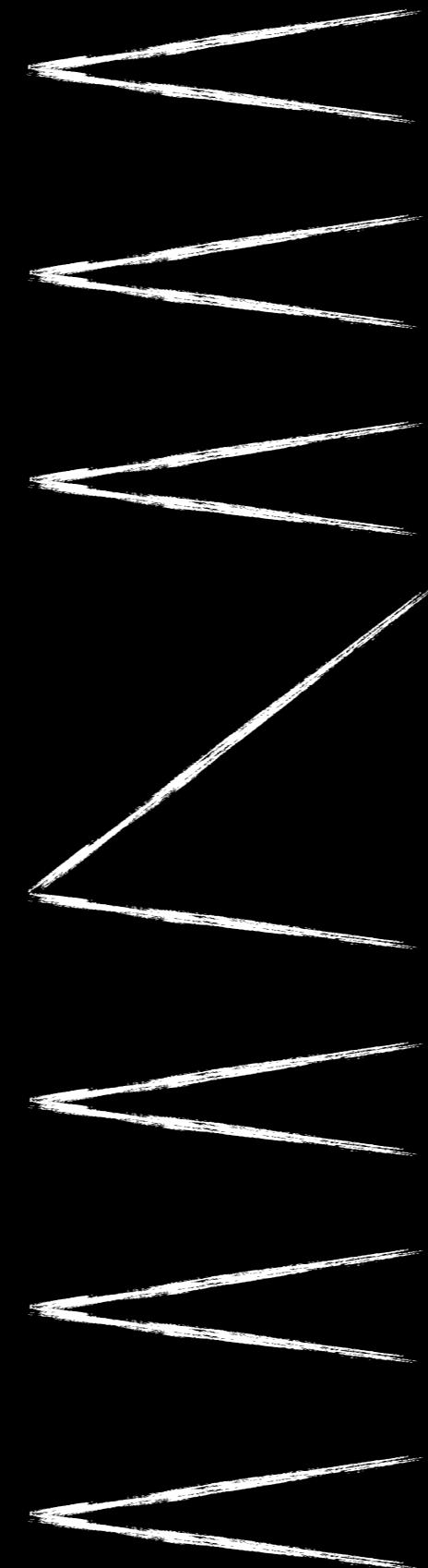
~~c1.close();~~

~~c1.write(..);~~

~~c1.close();~~

~~c1.reconnect();~~

~~c1.write(..);~~



|       |         |
|-------|---------|
| {}    | {0,1,2} |
| {}    | {0,1,2} |
| {}    | {0,1,2} |
| {}    | {1,2}   |
| {}    | {2}     |
|       |         |
| {1,2} |         |
| {2}   |         |

```
c1.close();
```



0

{ } {0,1,2}

```
c1.reconnect();
```



1

{ } {0,1,2}

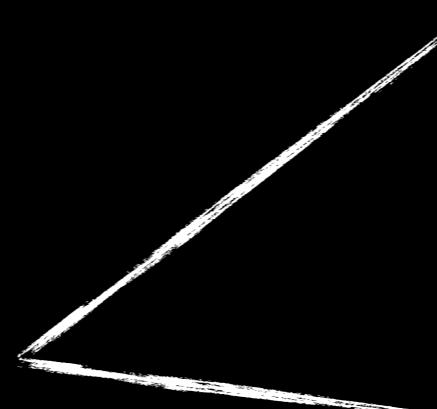
```
c1.close();
```



0

{ } {0,1,2}

```
c1.write(..);
```



1

{ } {1,2}

```
c1.close();
```



2

{ } {2}

```
c1.reconnect();
```



1

{ }

~~```
c1.write(..);
```~~

0

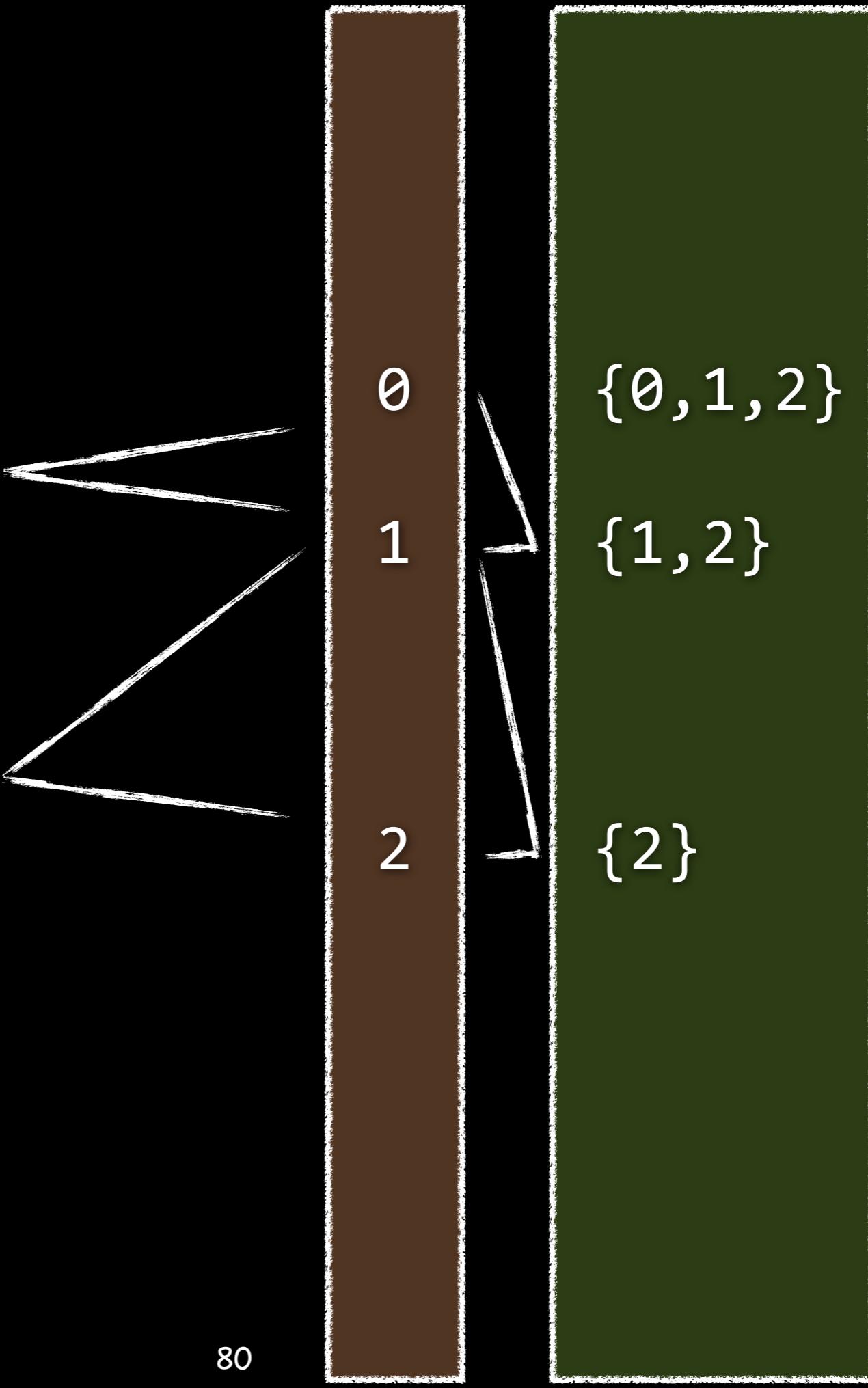
{1,2}

0

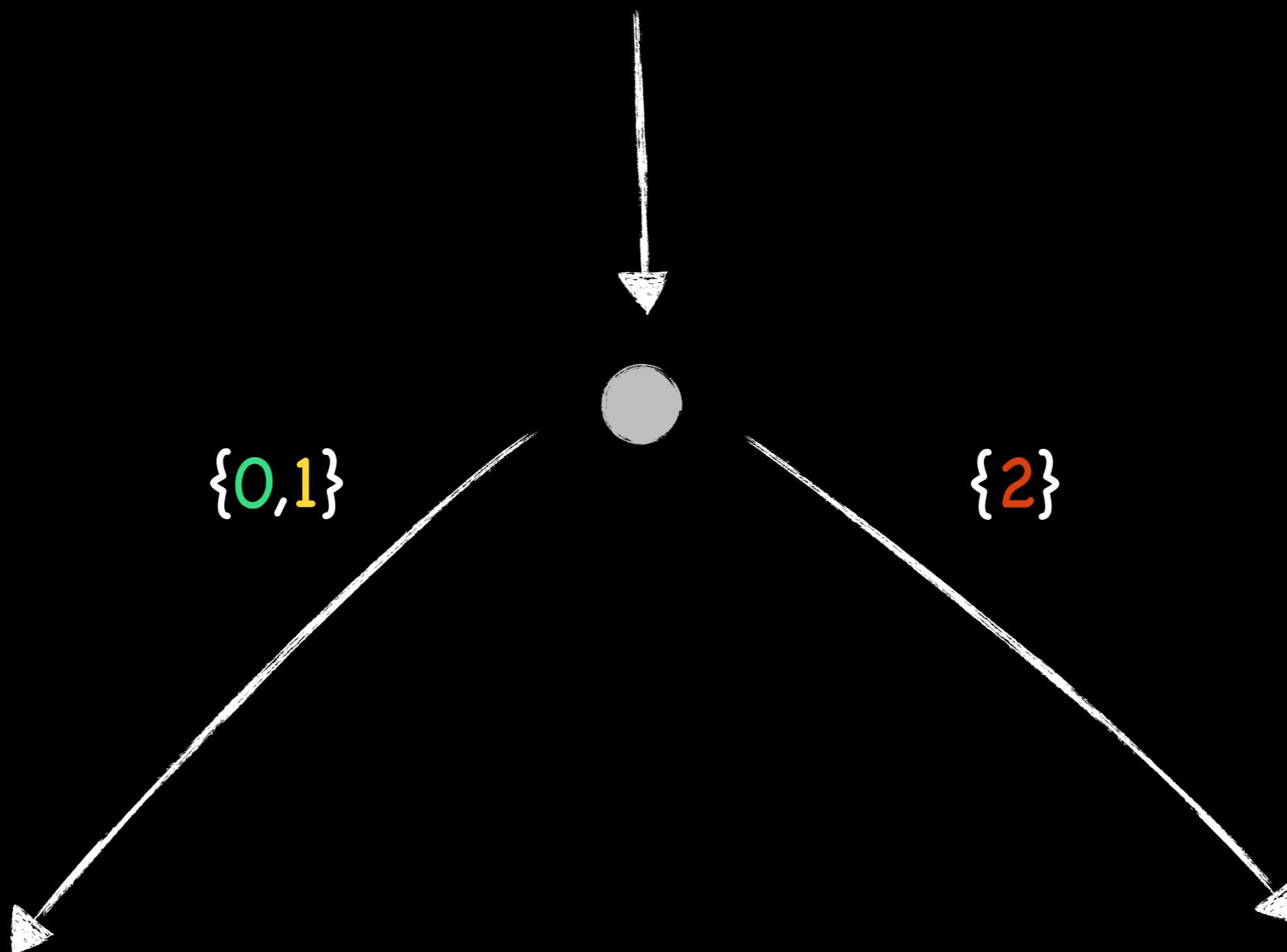
{2}

```
c1.close();
```

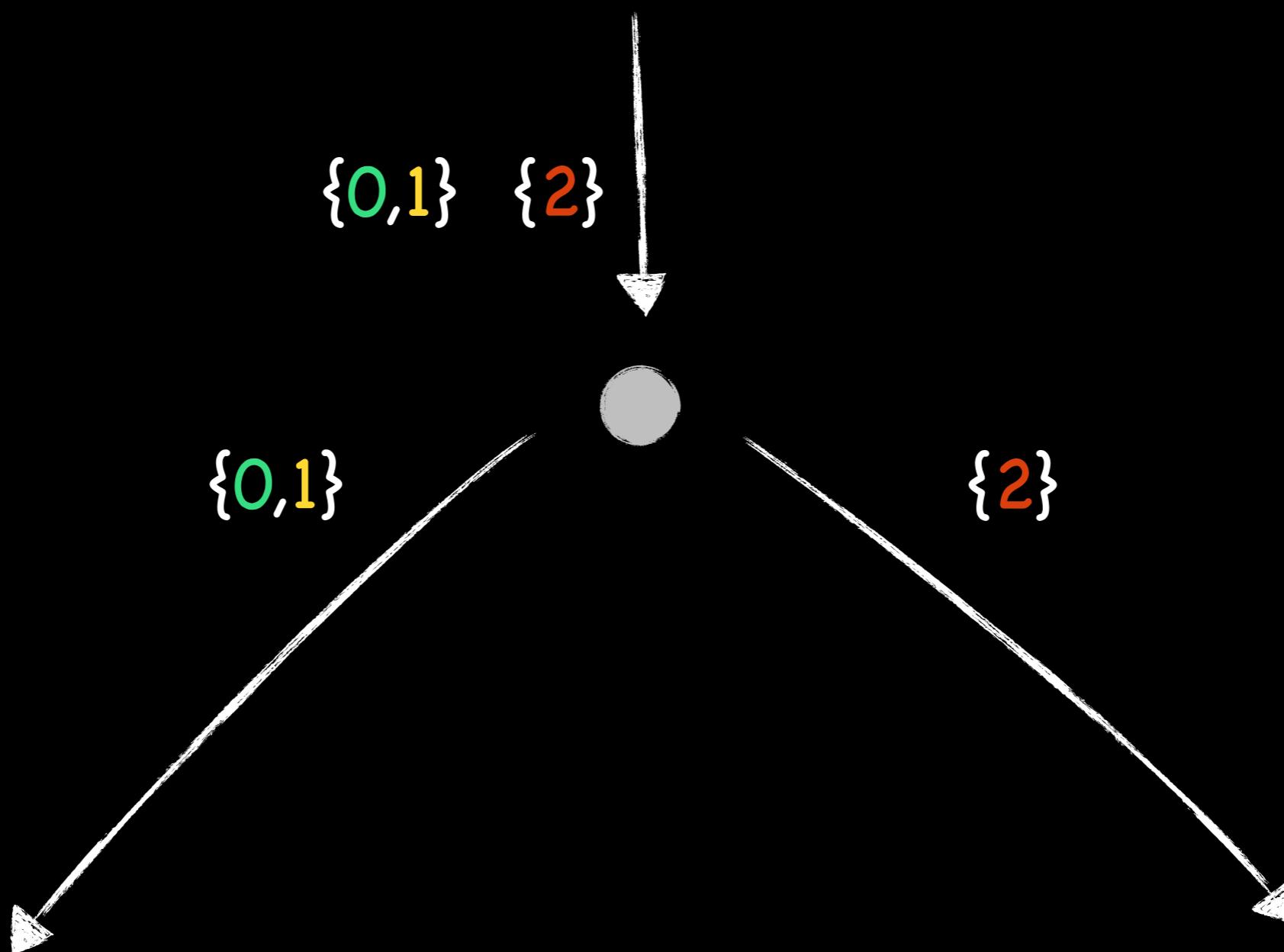
```
c1.write(...);
```



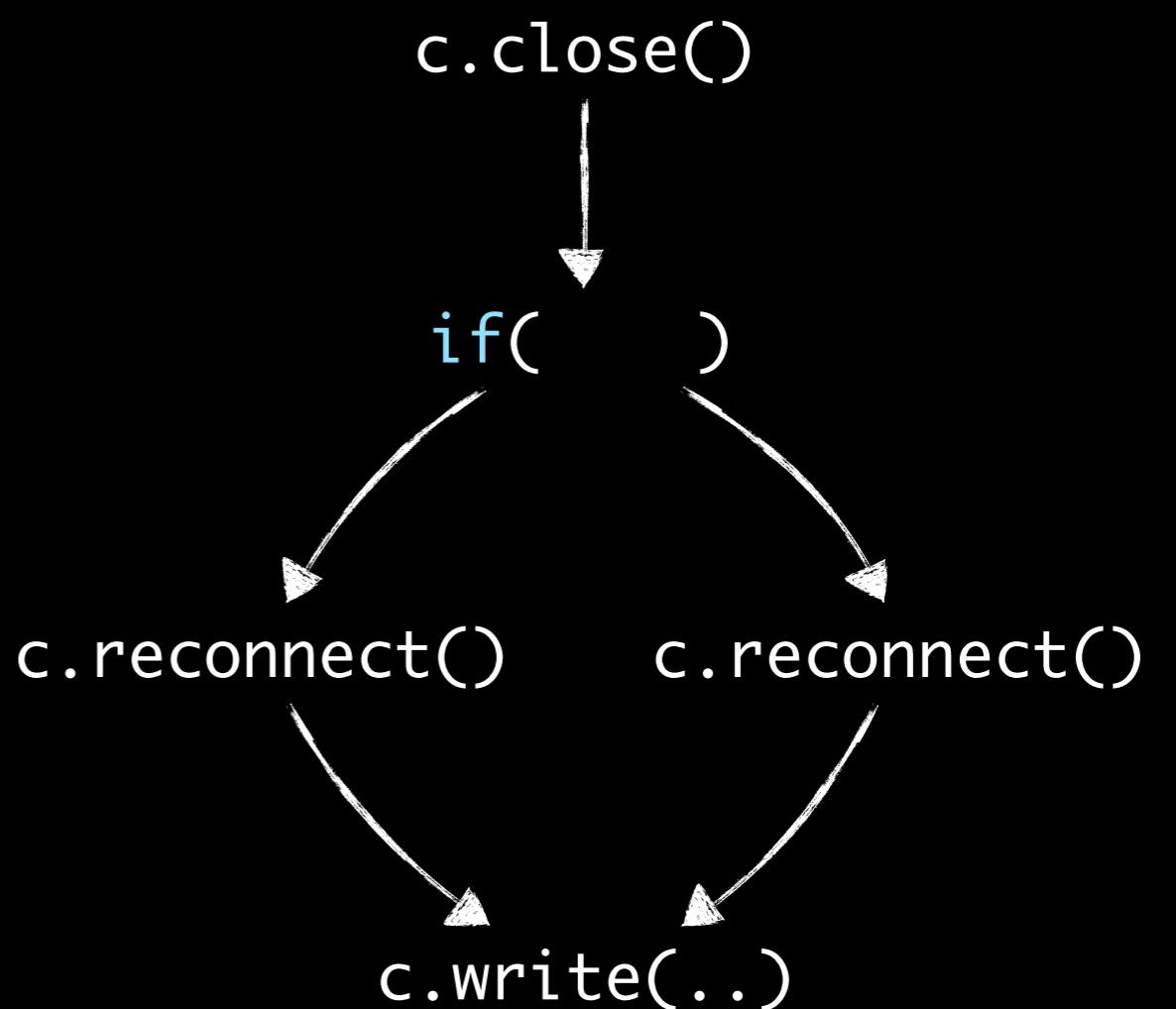
Analysis is path sensitive



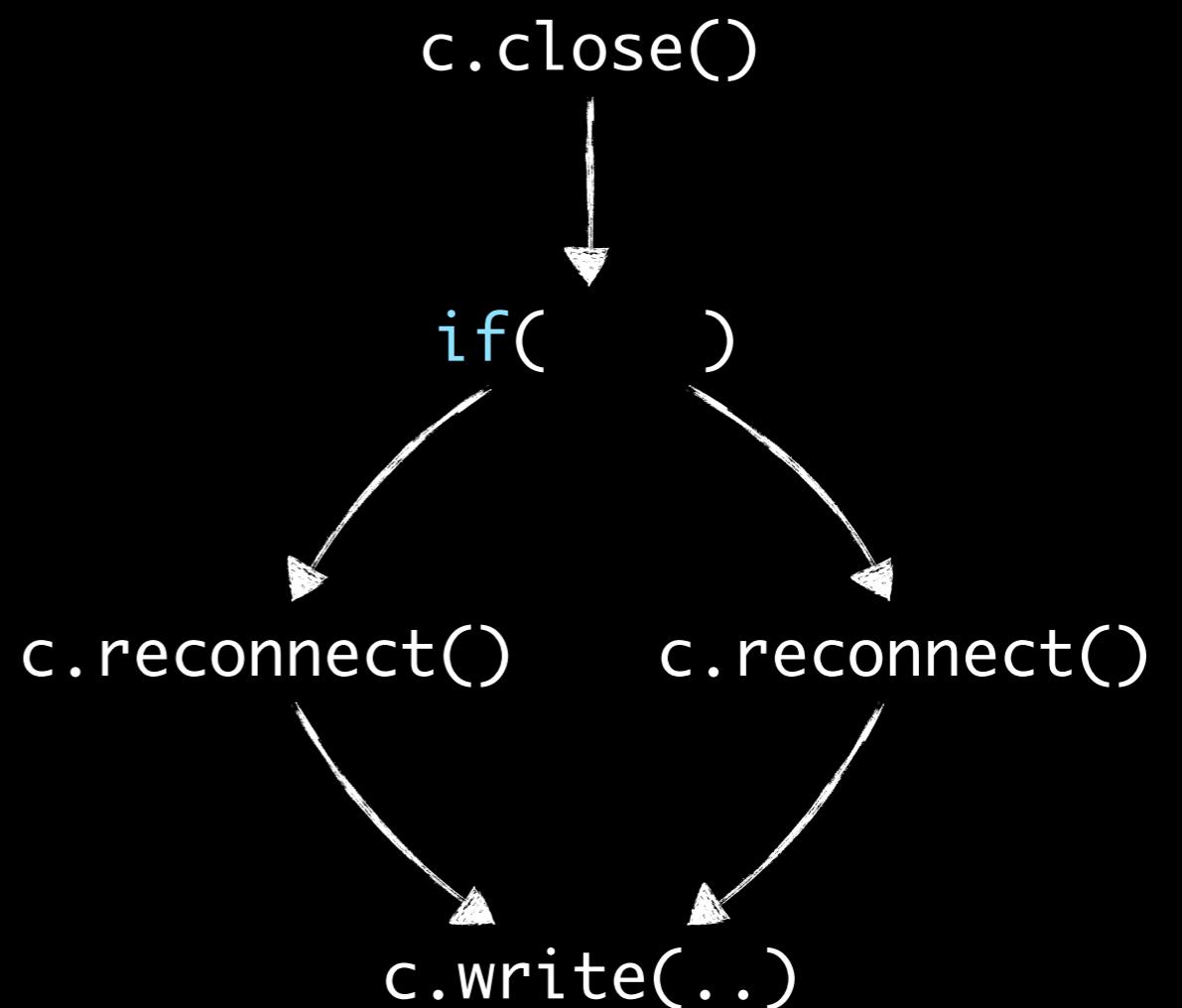
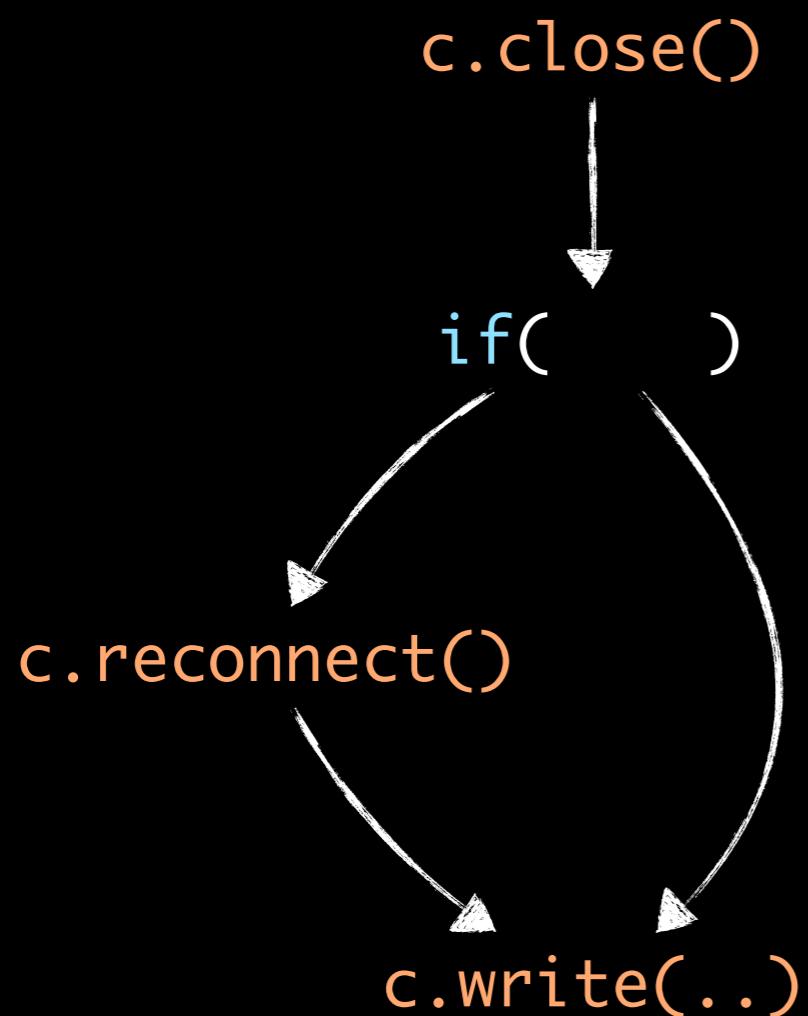
Analysis is path sensitive



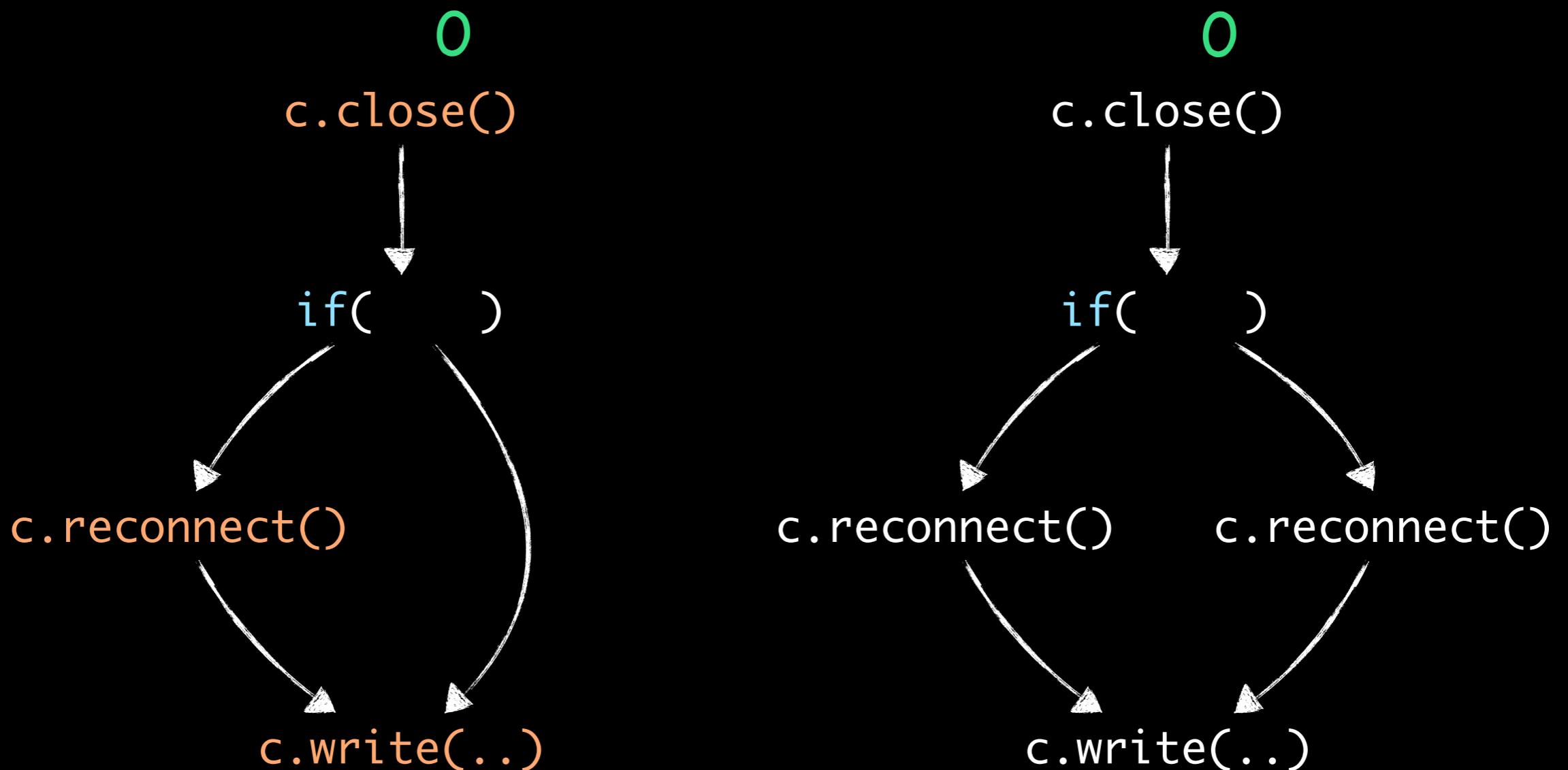
Handling conditionals



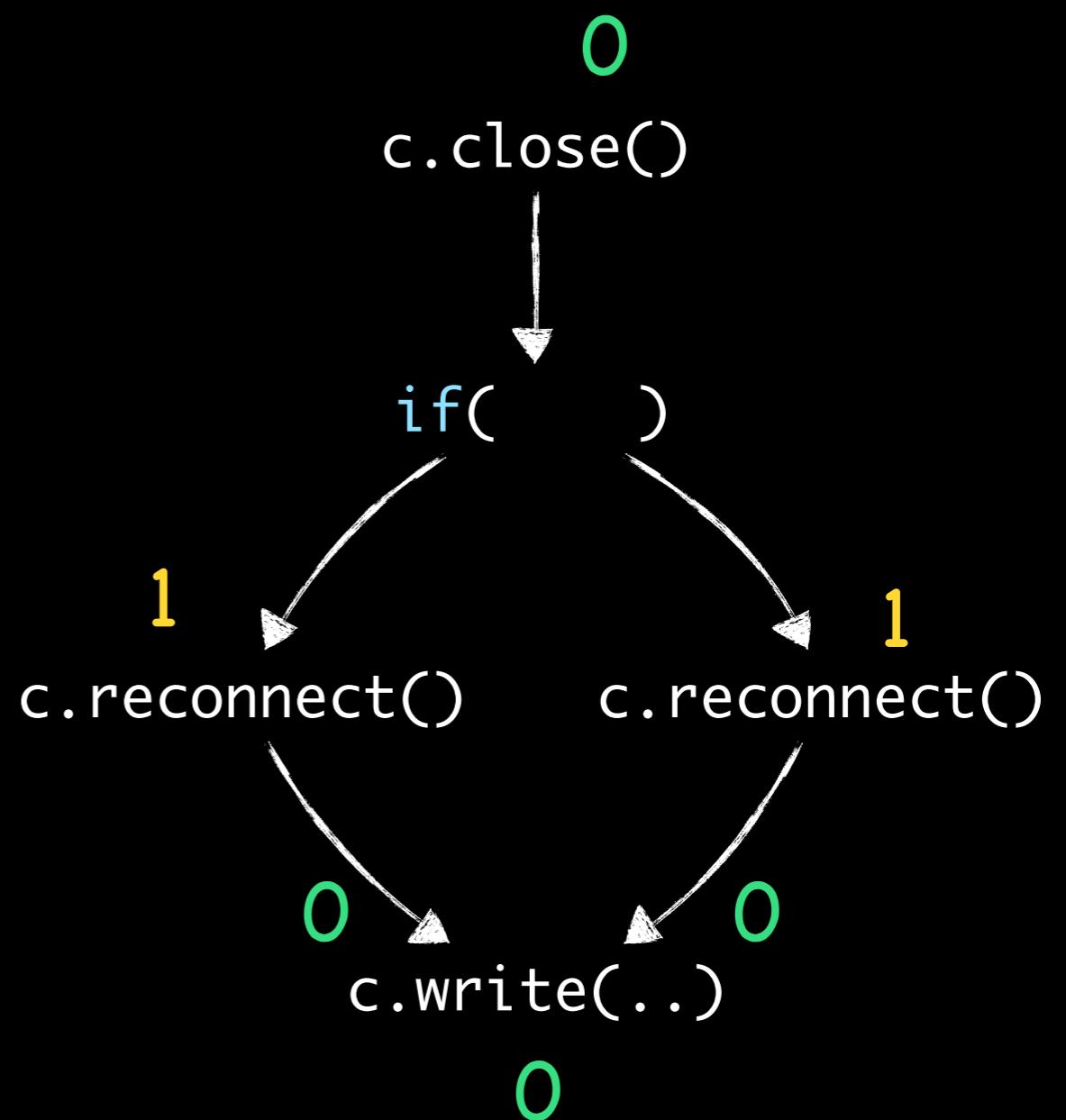
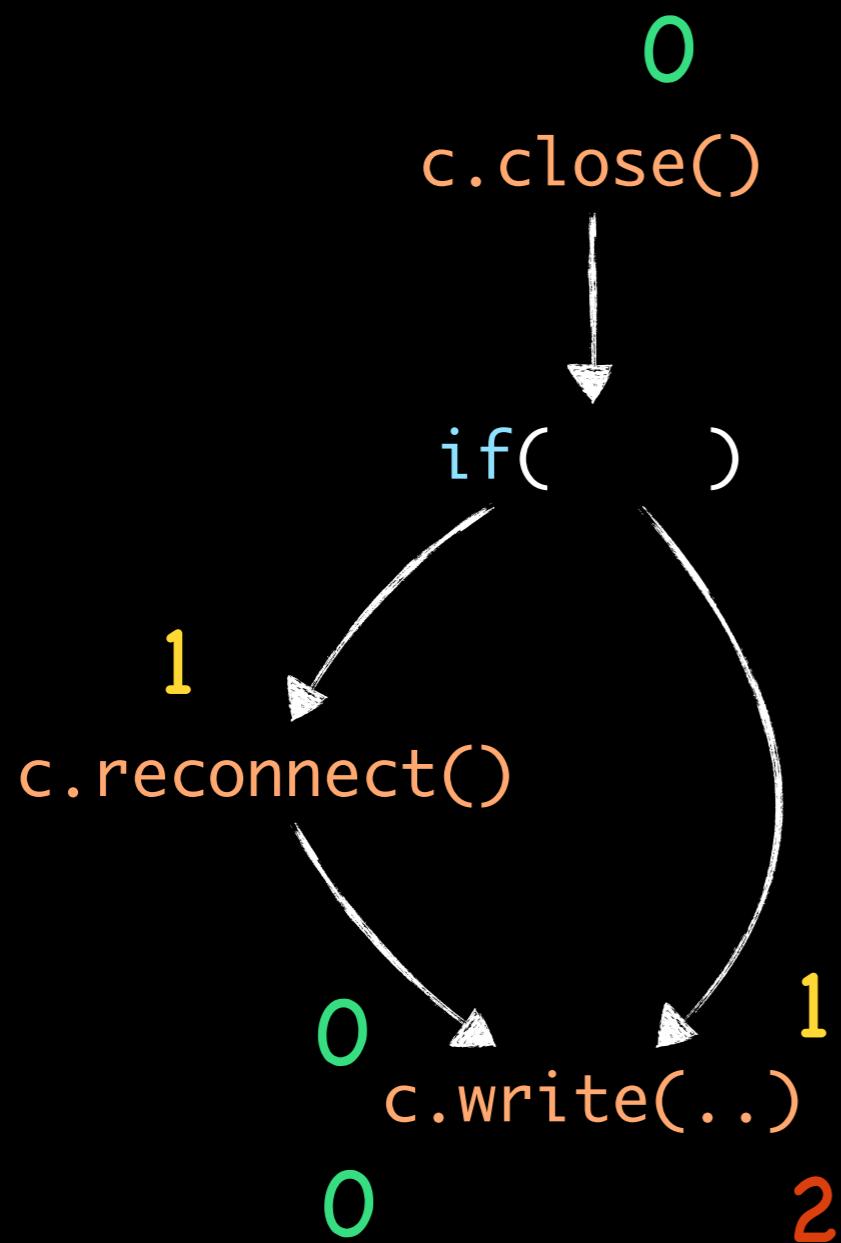
Handling conditionals



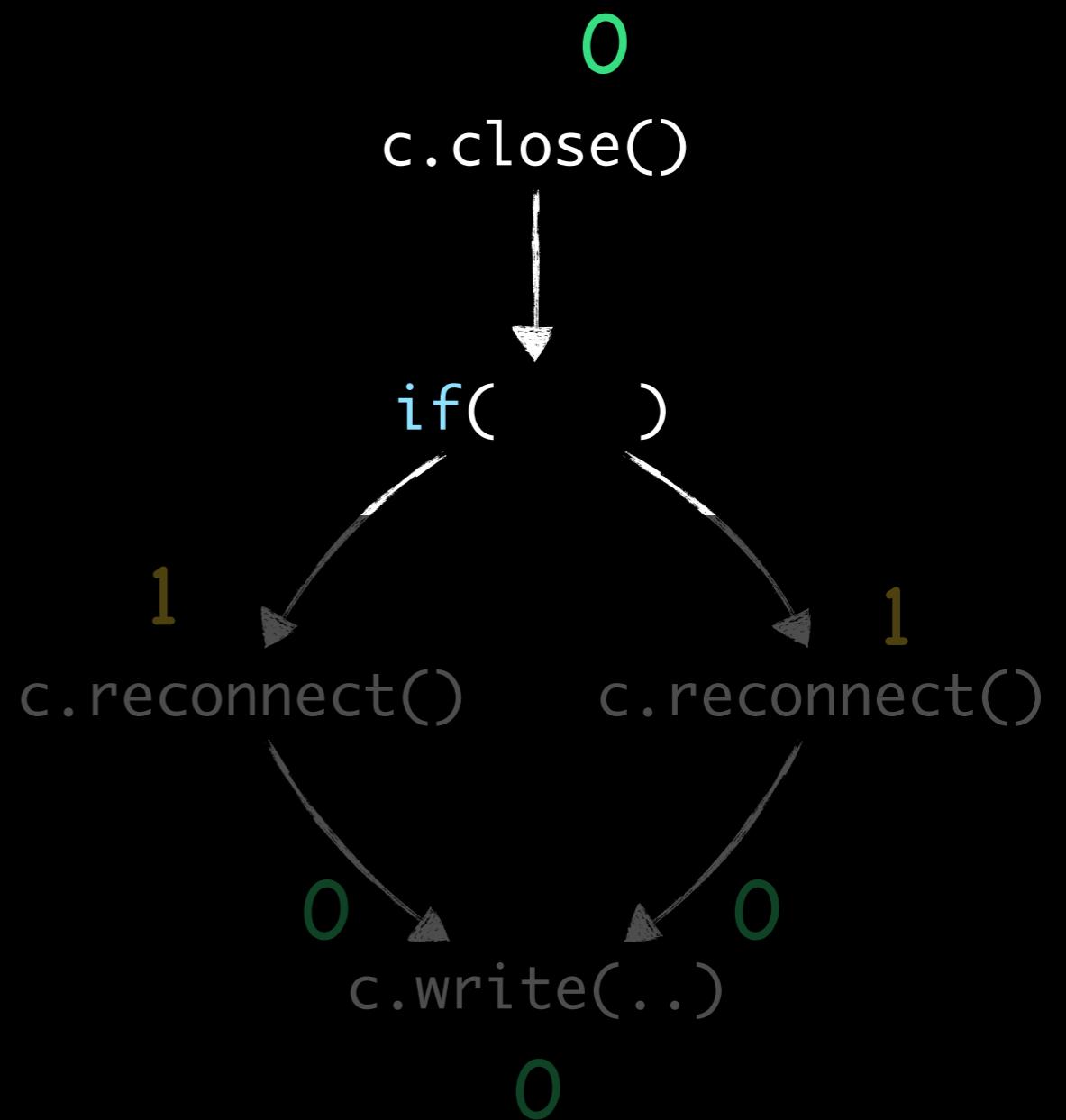
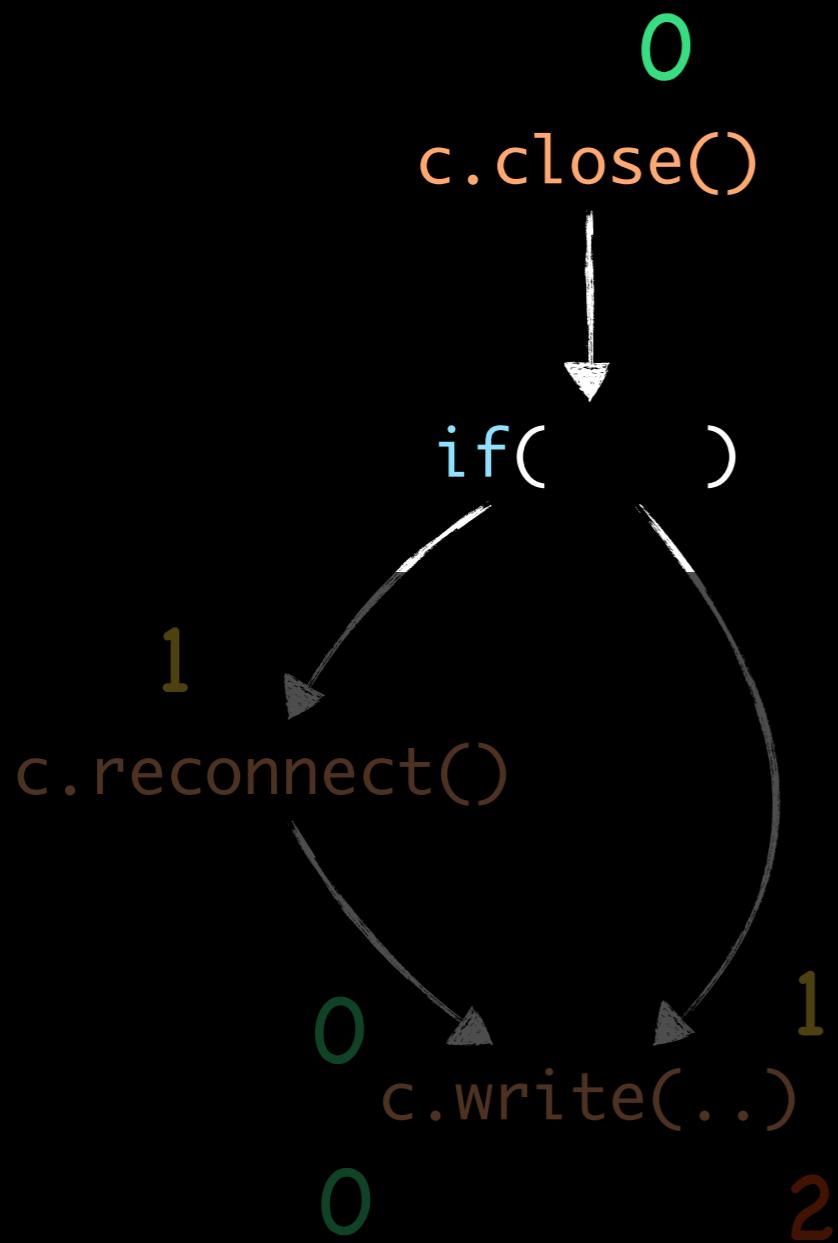
Handling conditionals

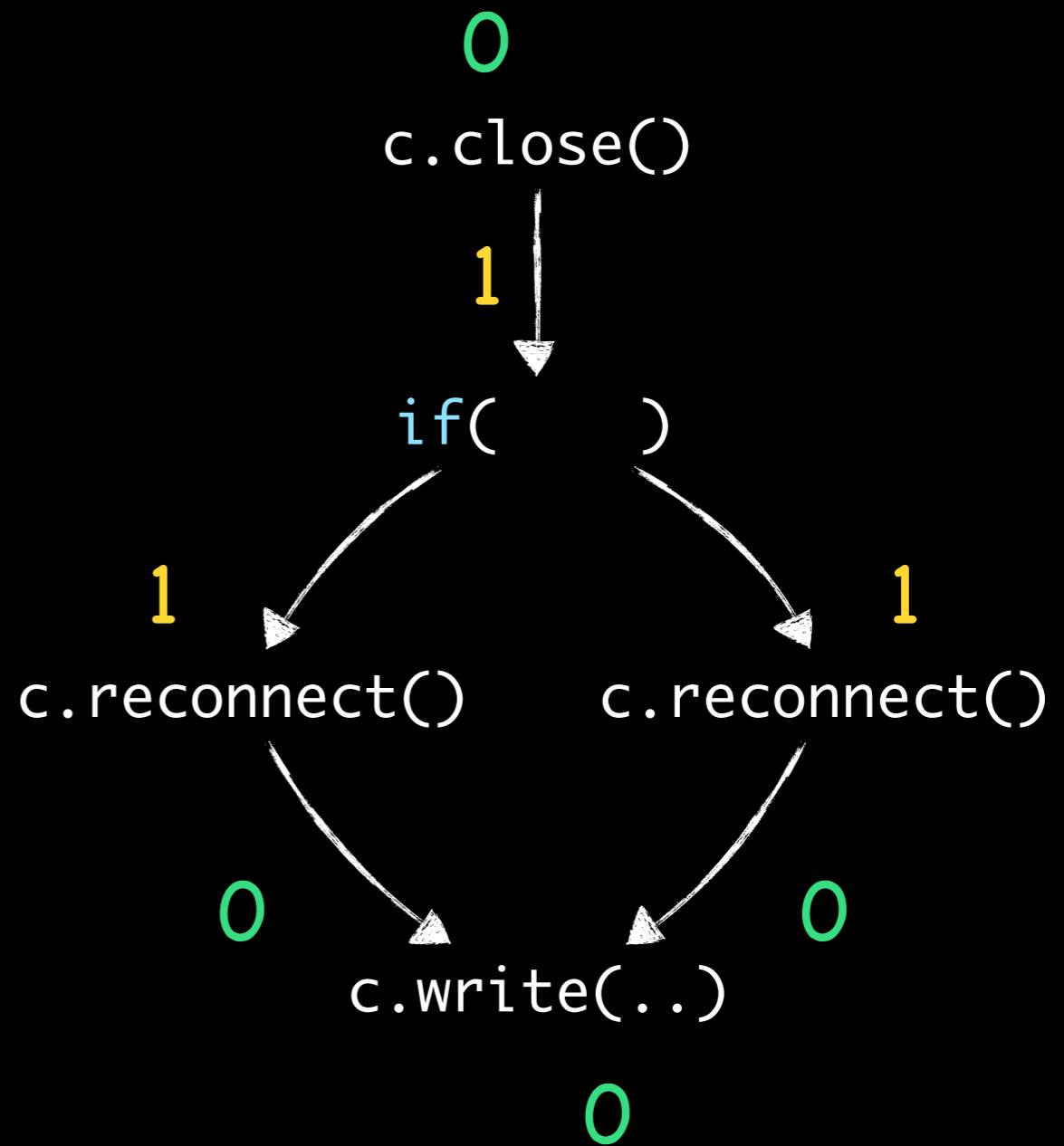
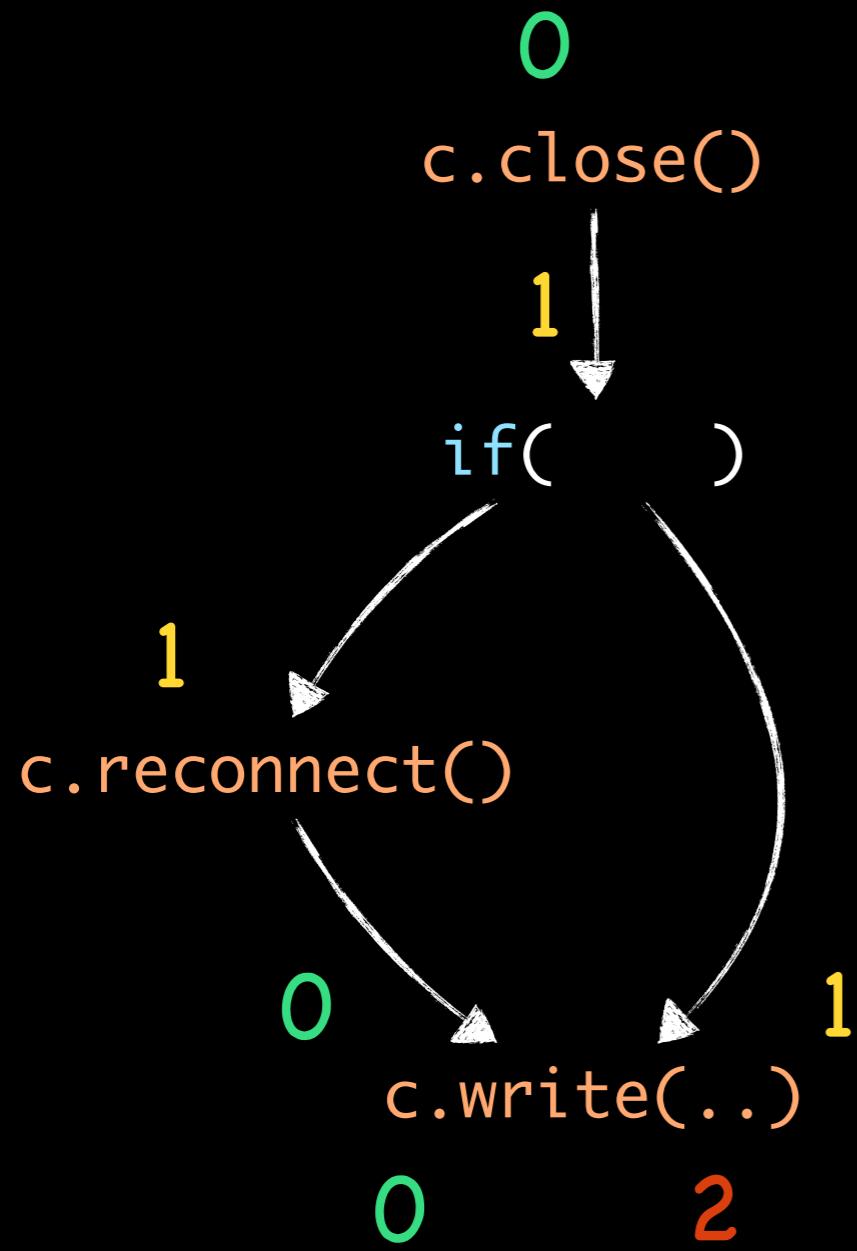


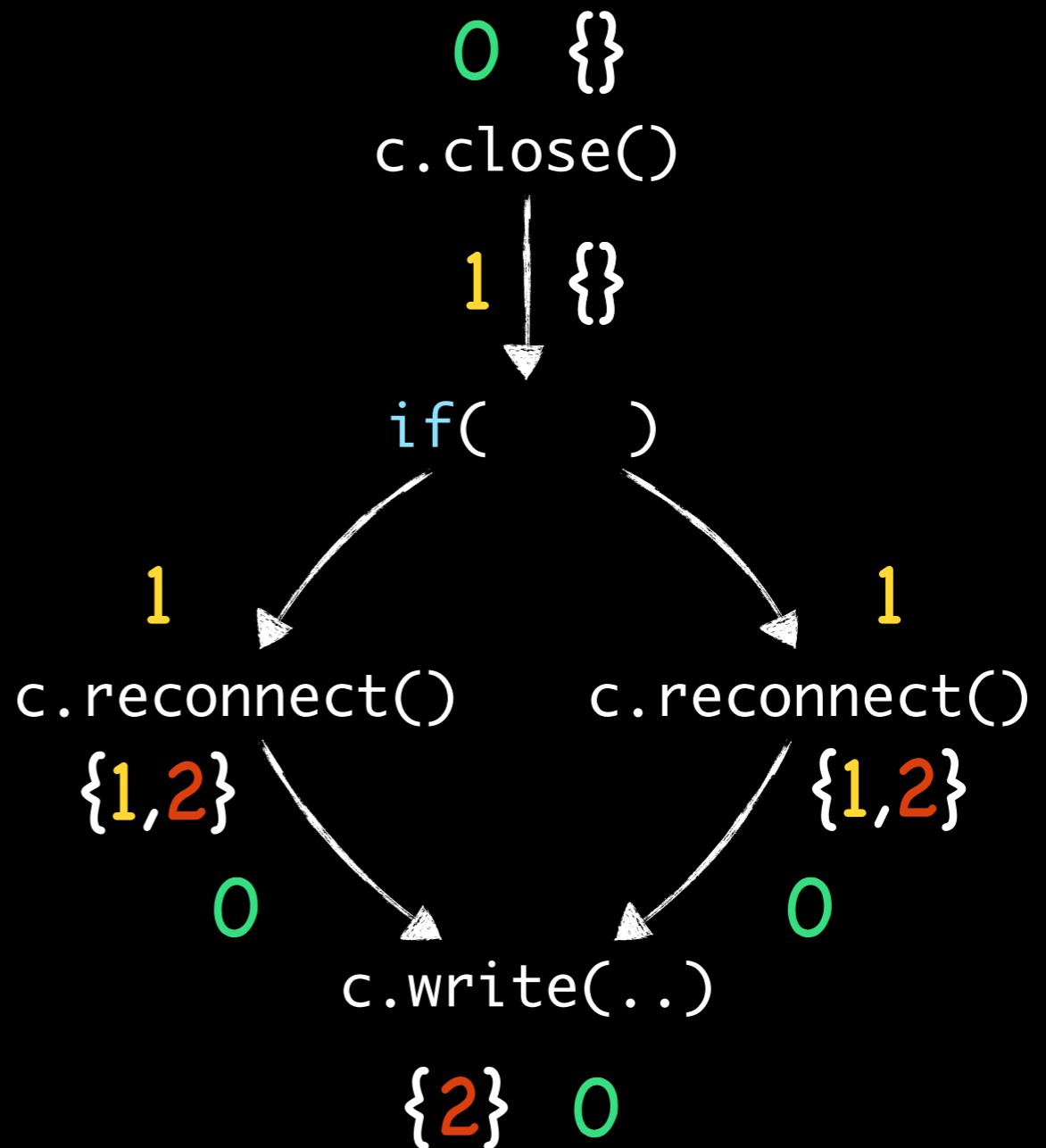
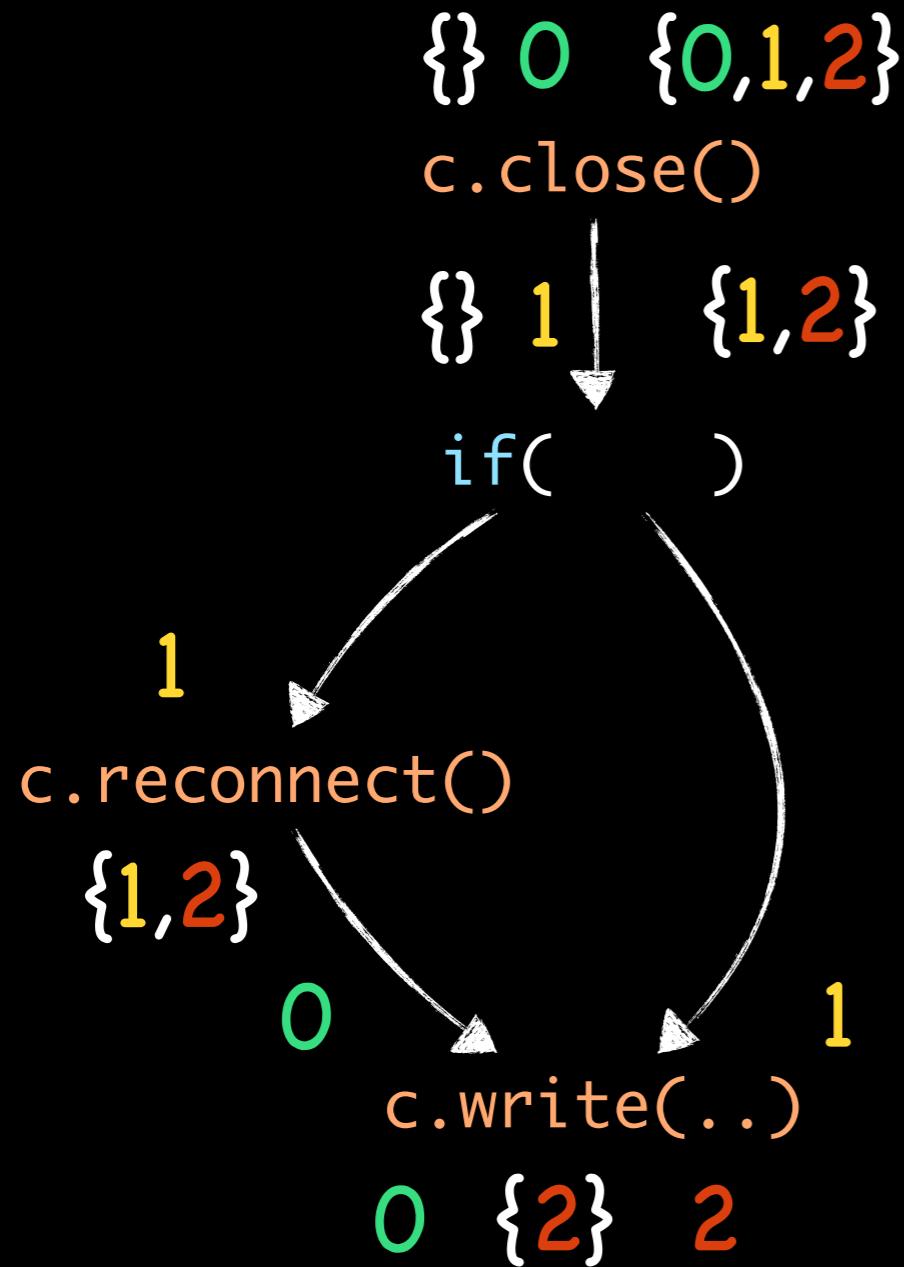
Handling conditionals

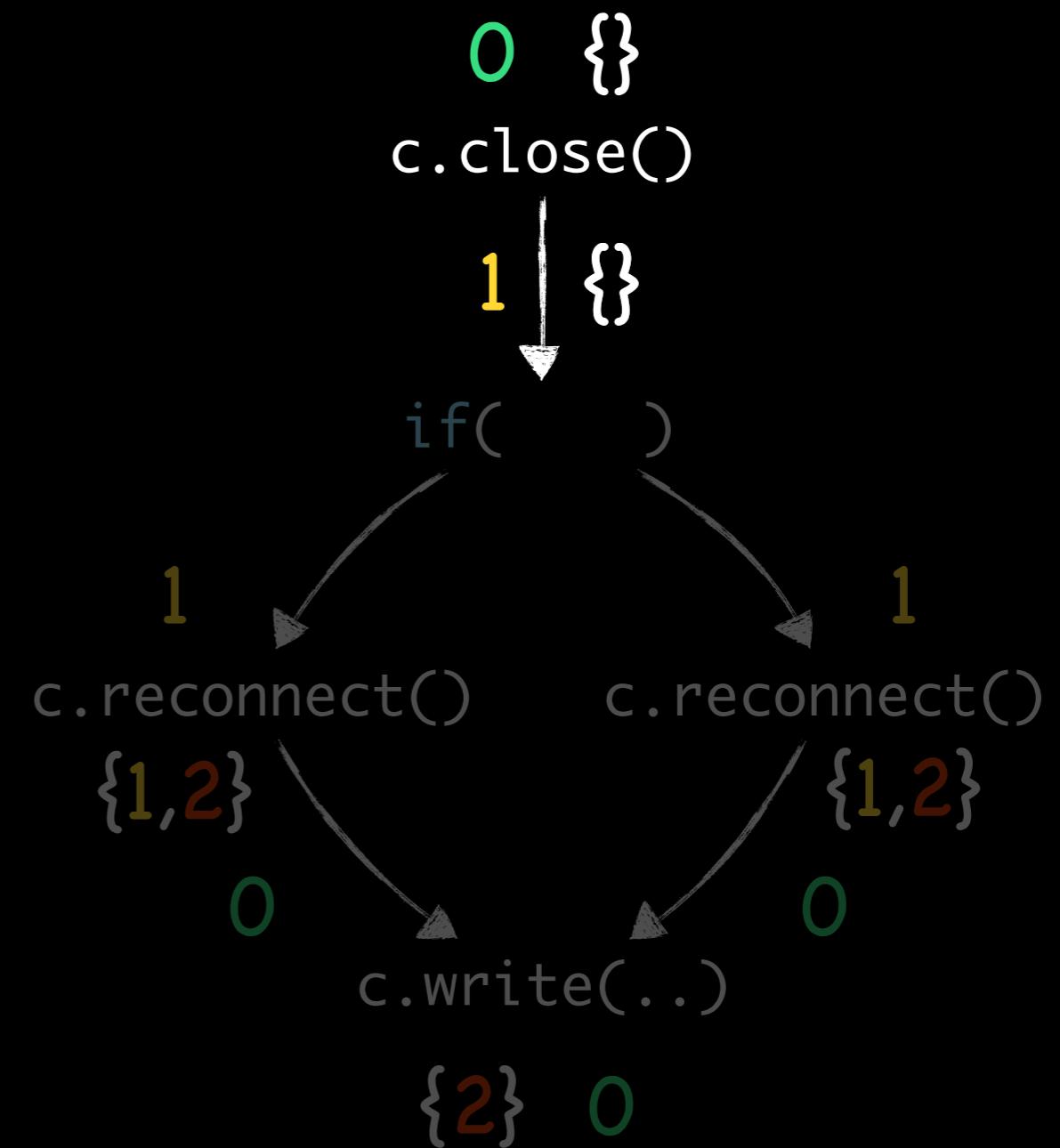
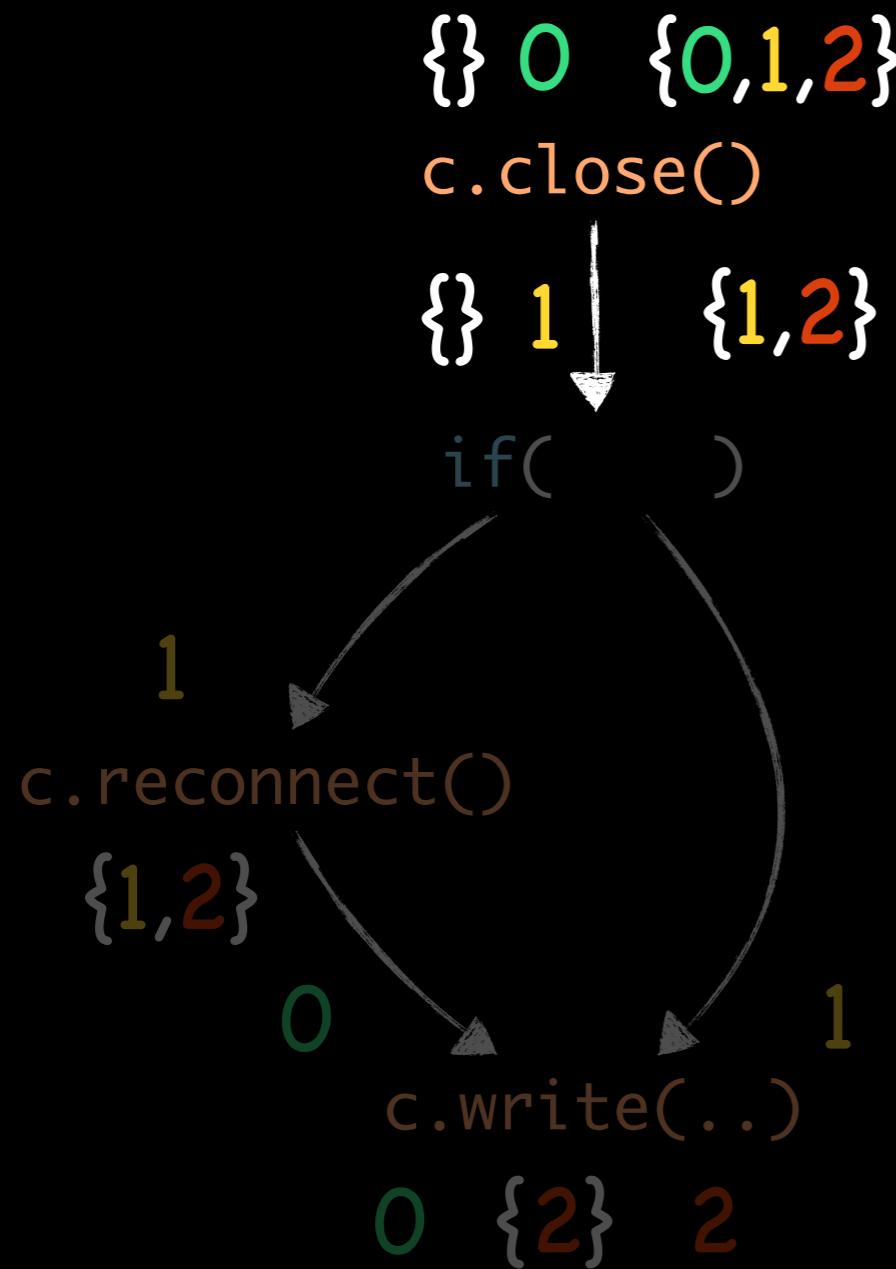


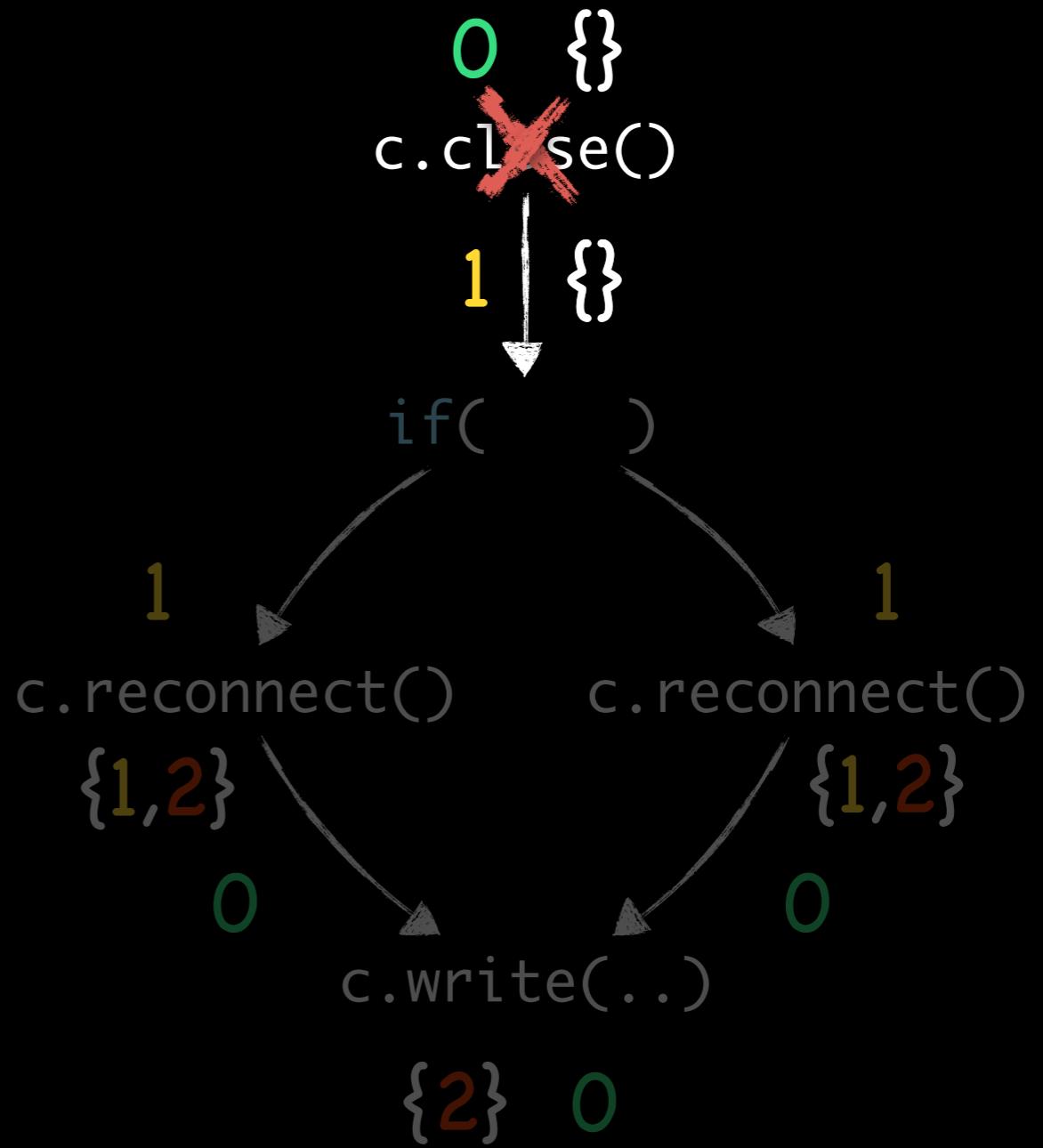
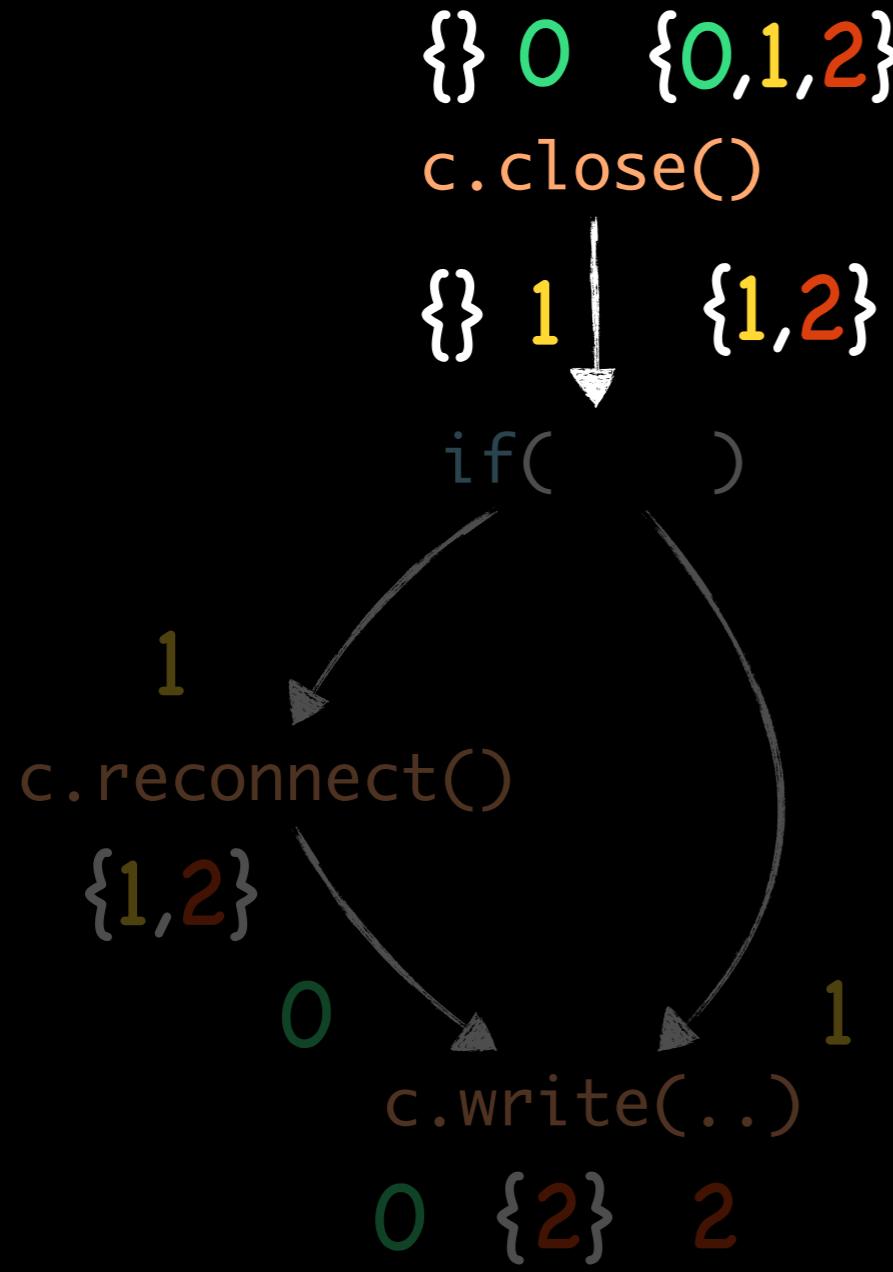
Handling conditionals

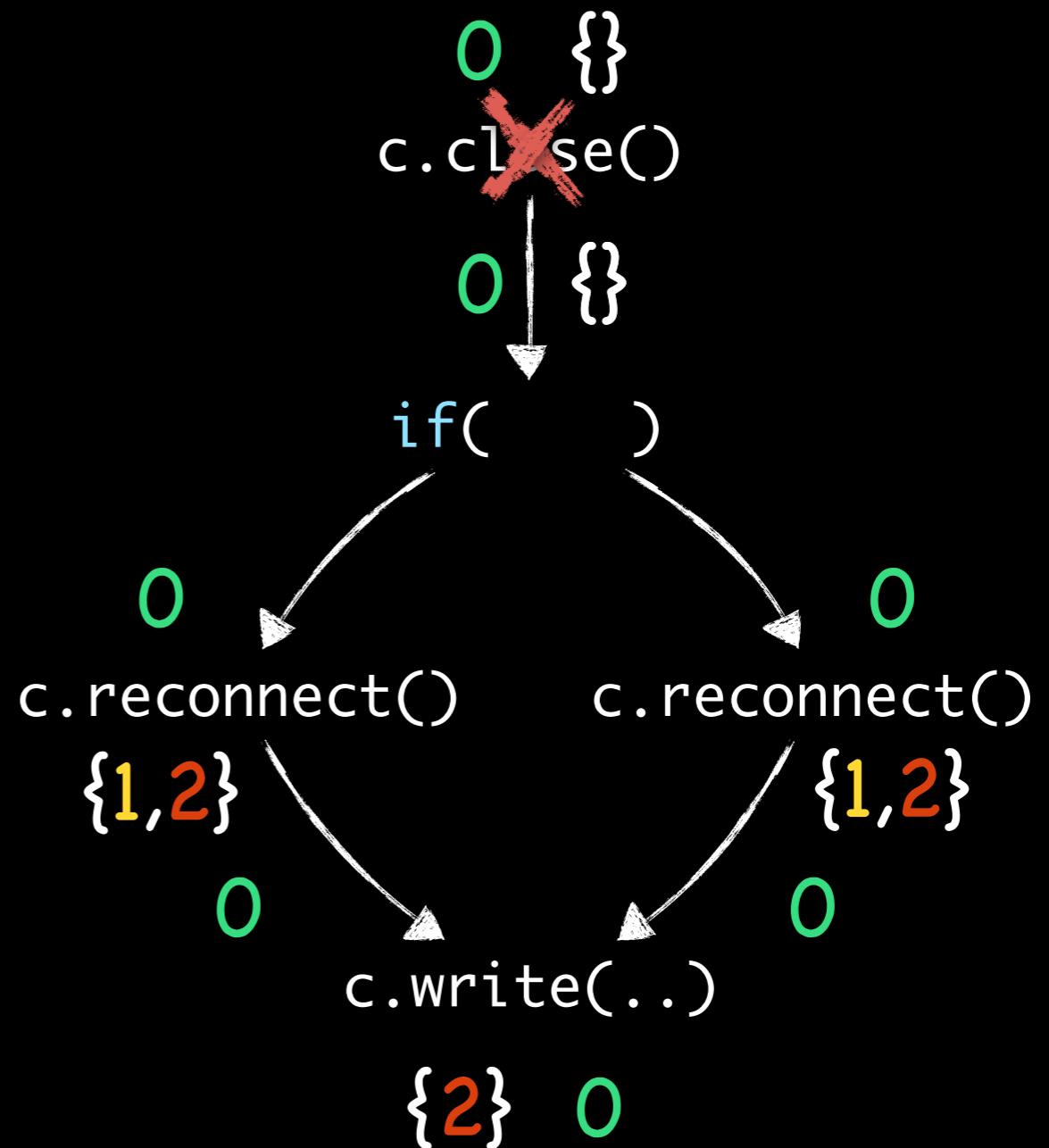
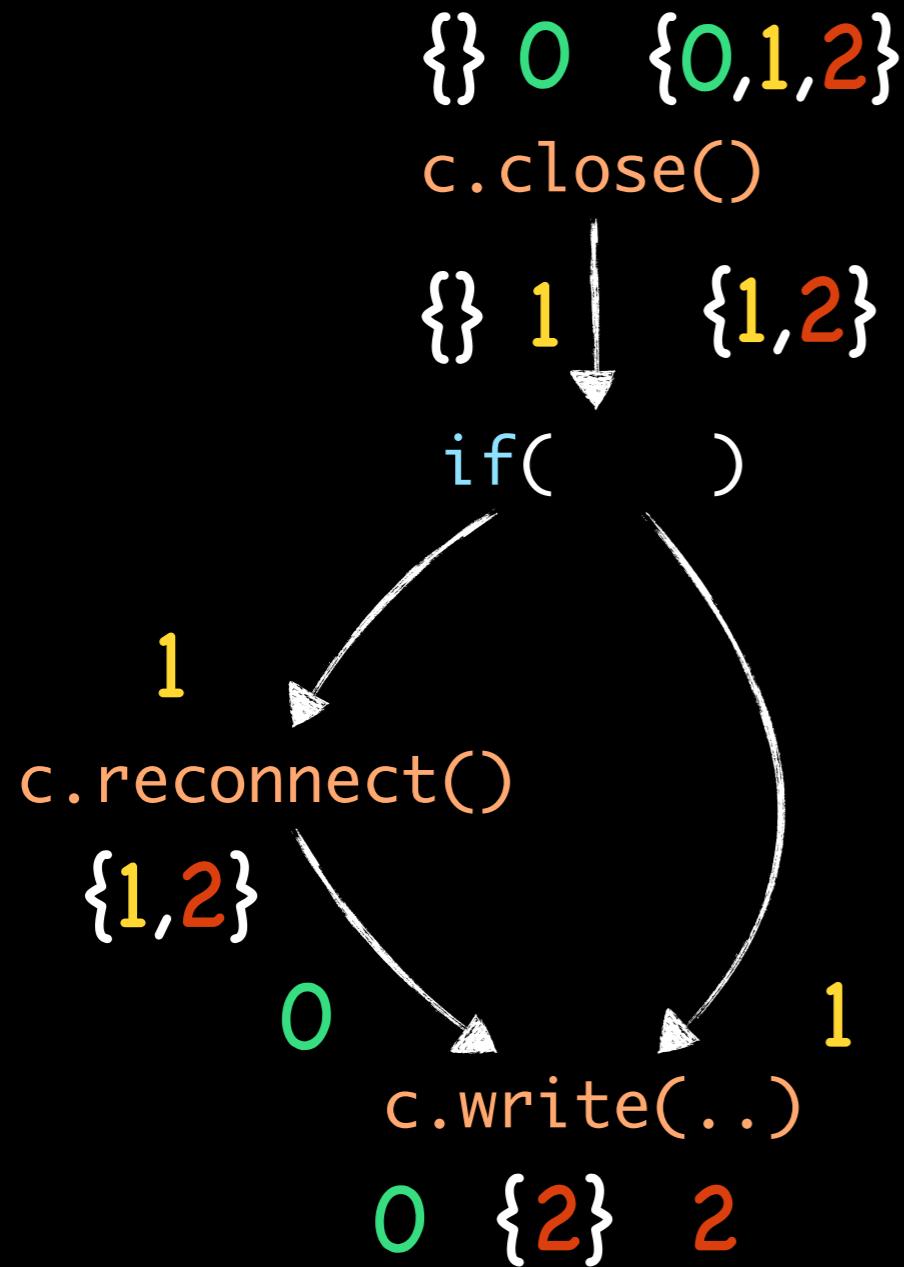


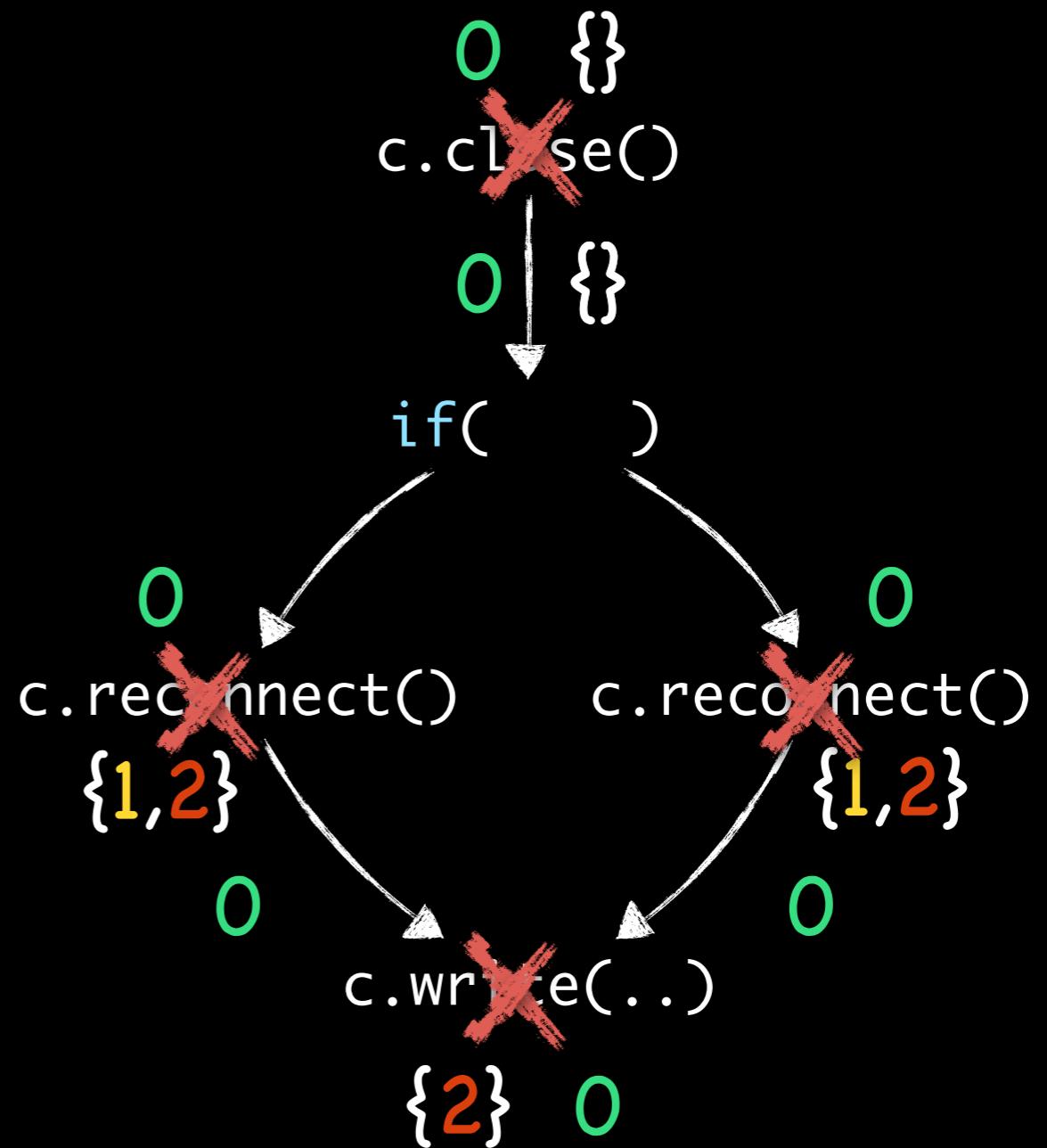
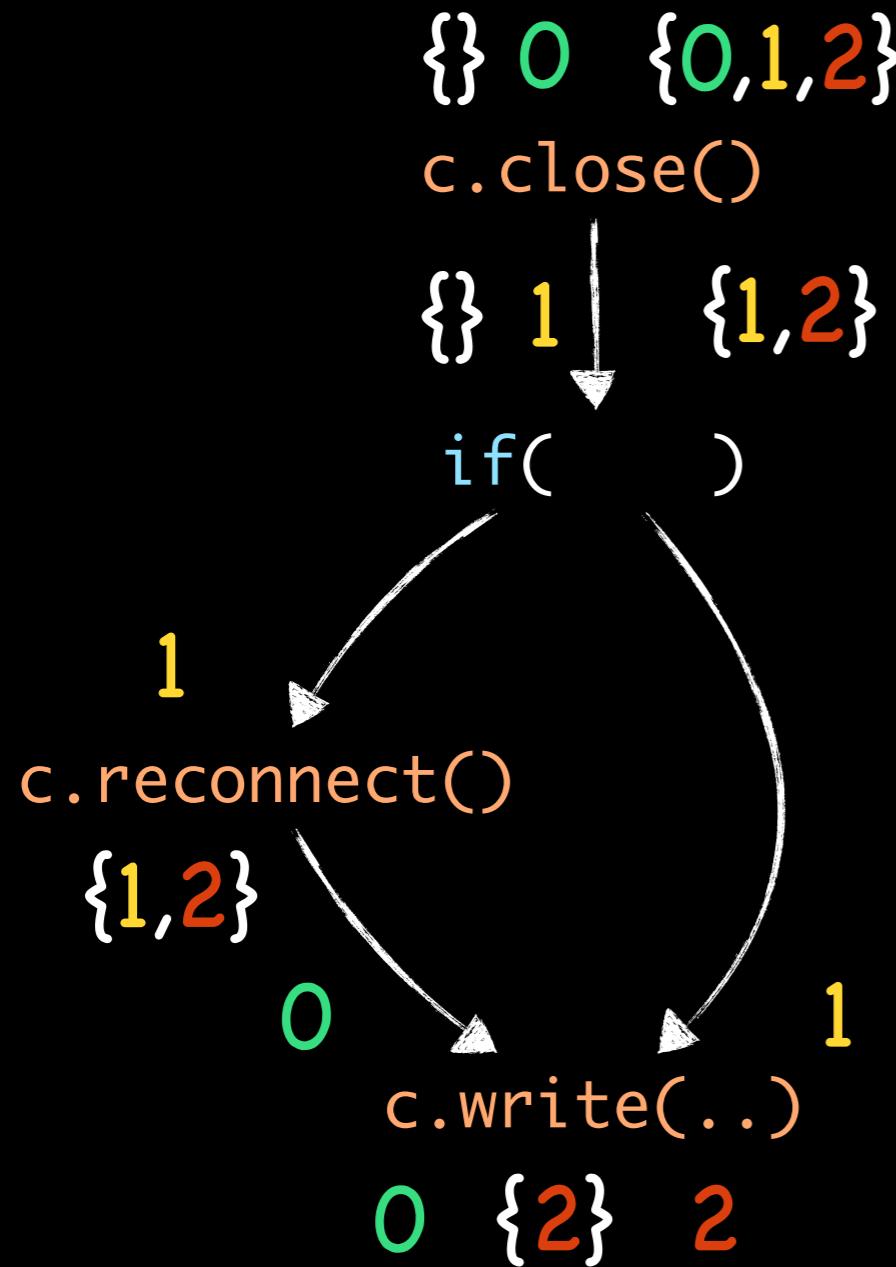




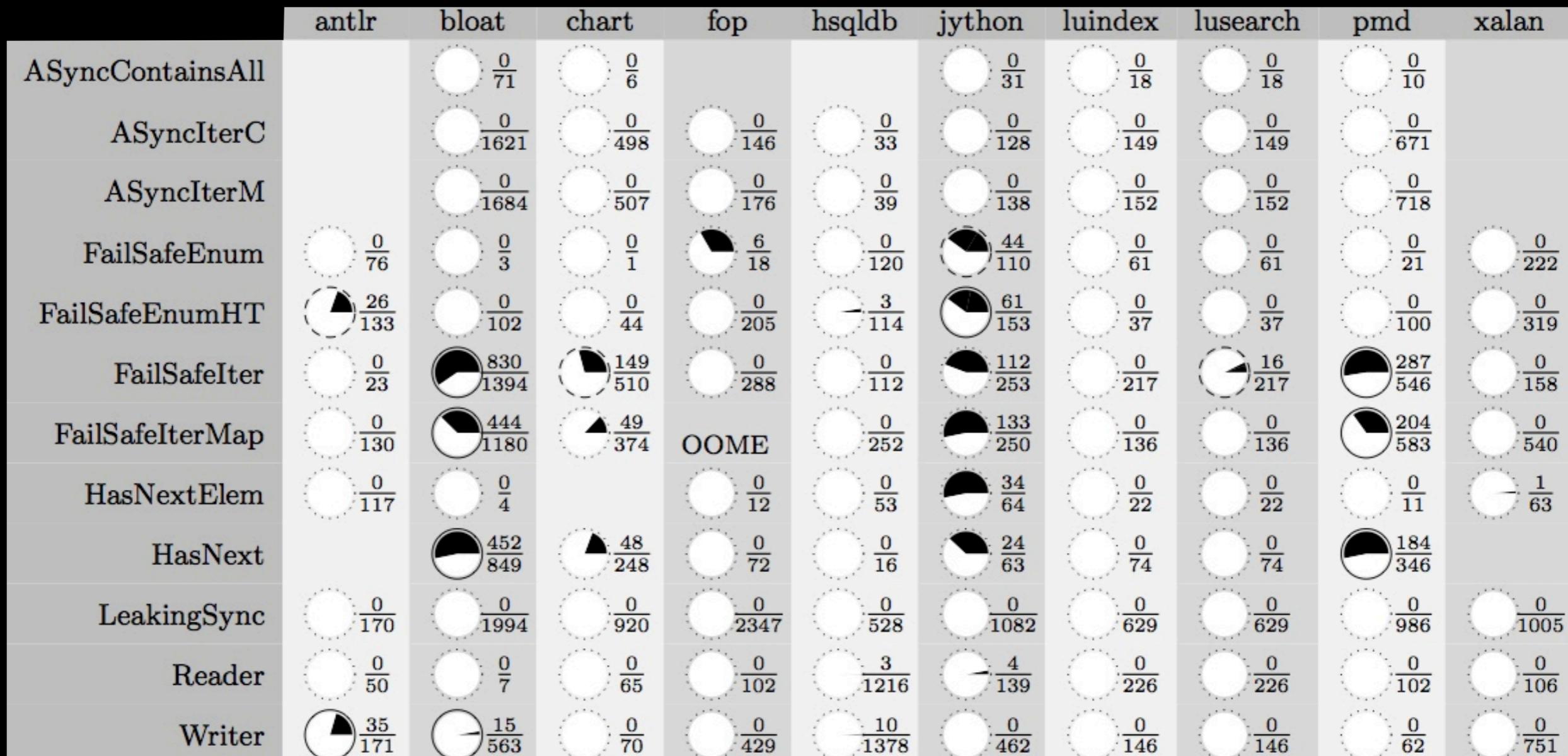








Overall success



Aiding inspection by the programmer

The screenshot shows a Java code editor with several methods annotated with comments and inspection results:

```
void canMatch1() {
    a(); //can match -> must stay
    b(); //can match -> must stay
}

void canMatch2() {
    a(); //can match -> must stay
    if(System.currentTimeMillis()>1021231)
        b(); //can match -> must stay
}

void canMatch3() {
    a(); //can match -> must stay
    if(System.currentTimeMillis()>1021231)
        x();
    b(); //can match -> must stay
}

void canMatch4() {
    b(); //can lead to a match in combination
}

void cannotMatch() { //cannot match its
    a(); //is on a dead path -> can be
    x(); //needs to stay because it can
    b(); //can be removed (dead)
}

void cannotMatch2() {
    x(); //needs to stay because it can
    b(); //can be removed (dead)
}

void unnecessary() {
    a(); //one of those
    a(); //is unnecessary
    b(); //can match -> must stay
}
```

Annotations and inspection results:

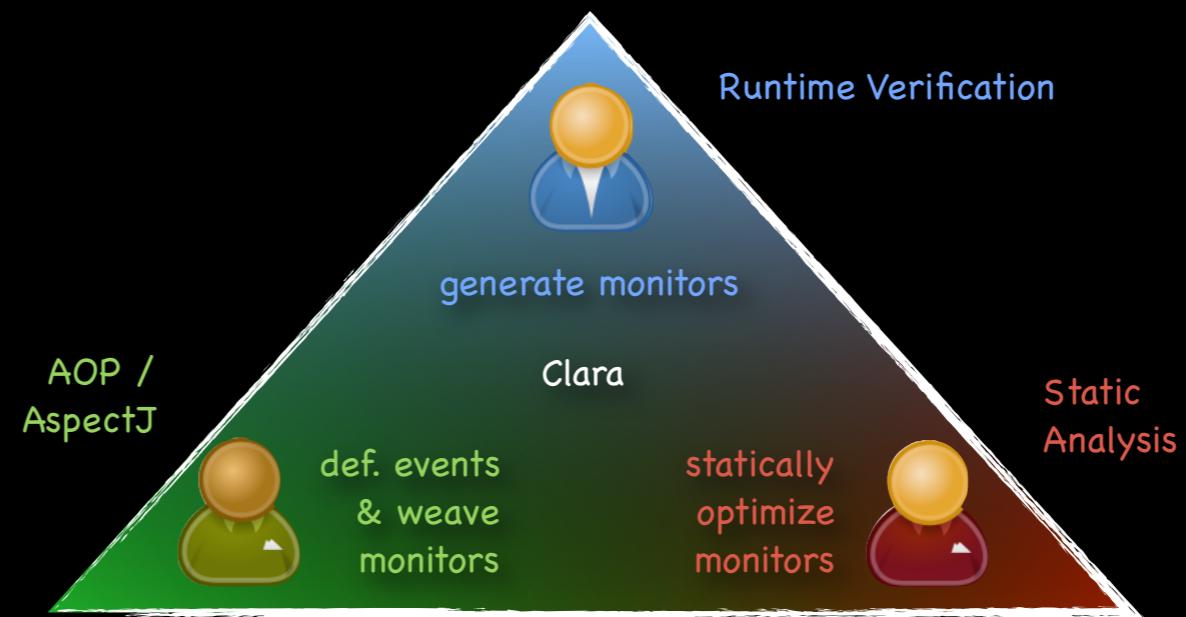
- Method canMatch1: a(), b(). Symbol: a, Transitions: {0=[1], 1=[1], 2=[1]}. Symbol: b, Transitions: {1=[2]}
- Method canMatch2: a(), b(). Symbol: a, Transitions: {0=[1], 1=[1], 2=[1]}. Symbol: b, Transitions: {1=[2]}
- Method canMatch3: a(), b(). Symbol: a, Transitions: {0=[1], 1=[1], 2=[1]}. Symbol: b, Transitions: {1=[2]}
- Method canMatch4: b(). Symbol: b, Transitions: {0=[0], 1=[2], 2=[0]}
- Method cannotMatch: a(), x(), b(). Symbol: x, Transitions: {1=[0]}. Symbol: a, Transitions: {0=[0], 1=[2], 2=[0]}. Symbol: b, Transitions: {0=[0], 1=[2], 2=[0]}
- Method cannotMatch2: x(), b(). Symbol: x, Transitions: {1=[0]}. Symbol: a, Transitions: {0=[0], 1=[2], 2=[0]}
- Method unnecessary: a(), a(), b(). Symbol: a, Transitions: {0=[1], 1=[1], 2=[1]}. Symbol: b, Transitions: {1=[2]}

A context menu is open over the code at the bottom of the canMatch3() method, showing options "Within this file" and "Within OutraFlow.java".

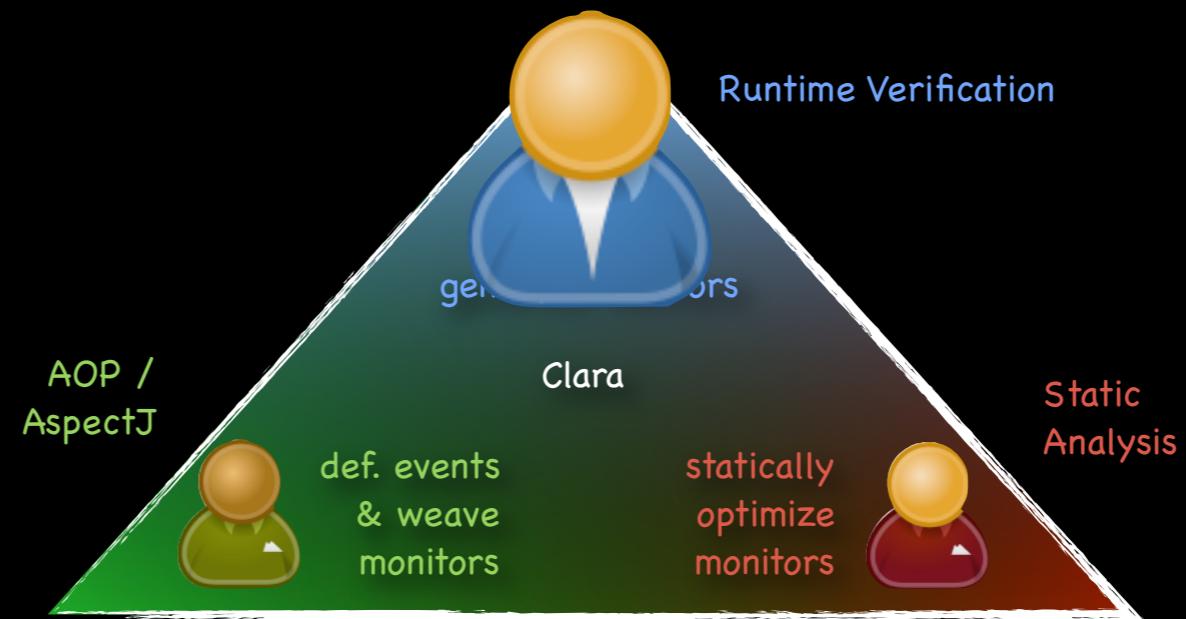
Inspection results (list 19-86):

- 19: sym a in method canMatch1
- 20: sym b in method canMatch1
- 24: sym a in method canMatch2
- 26: sym b in method canMatch2
- 37: sym b in method canMatch4
- 42: sym x in method cannotMatch
- 47: sym x in method cannotMatch2
- 53: sym a in method unnecessary
- 54: sym b in method unnecessary
- 58: sym a in method canMatchLoop
- 61: sym b in method canMatchLoop
- 65: sym x in method cannotMatchLoop
- 75: sym x in method cannotMatchFinally
- 82: sym a in method canMatchTryCatch
- 84: sym x in method canMatchTryCatch
- 86: sym b in method canMatchTryCatch

Using Clara to evaluate your monitoring aspects



Using Clara to evaluate your monitoring aspects



Case 1: Hand-written aspects

```
Set closed = new IdentityHashSet();

after(Connection c) returning:
call(* Connection.close()) && target(c) {
    closed.add(c);
}

after(Connection c) returning:
call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
}

after(Connection c) returning:
call(* Connection.write(..)) && target(c) {
    if(closed.contains(c))
        error("May not write to "+c+", as it is closed!");
}
```

Start with your normal monitoring aspect

Case 1: Hand-written aspects

```
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

dependent after write(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }
```

Name all pieces of advice

Case 1: Hand-written aspects

```
dependency{
    disconnect, write, reconnect;
initial   connected: disconnect -> connected,
           write -> connected,
           reconnect -> connected,
           disconnect -> disconnected;
disconnected: disconnect -> disconnected,
              write -> error;
final      error: write -> error;
}
```

Come up with the correct DSM annotation

Case 2: Extending an RV tool

- Assume: tool normally generates ordinary AspectJ aspects
- Idea: extend tool to automatically annotate these aspects with DSM annotations

Case 2: Extending an RV tool

- Simple case: RV tool already uses a FSM representation of the monitored property
 - simply name the pieces of advice correctly
 - simply print the FSM in DSM syntax
- Important: annotation must describe language of rejected traces, not language of accepted traces!

Case 2: Extending an RV tool

But what if the monitored language
is not regular?

Example: CFG that accepts $L = a^n b^n, n > 0$

$S \rightarrow a S b \mid a b$

Want to signal an error, whenever a sequence
of the form $L = a^n b^n$ is matched.

Case 2: Extending an RV tool

Want to signal an error, whenever a sequence of the form $L = a^n b^n$ is matched.

- DSM has to describe a regular language L_R
- two sensible options:
 $L_R \subseteq L$ or $L_R \supseteq L$

Case 2: Extending an RV tool

- $L_R \subseteq L$ guarantees no false error messages
- on the other hand, optimized monitor may miss actual violations
- assume $L_R = a\ b \mid a\ a\ b\ b$
- optimized monitor will correctly signal error for a program with event sequence “a a b b”
- Clara will incorrectly optimize away monitoring code for a program with event sequence “a a a b b b” -> monitor will miss this violation

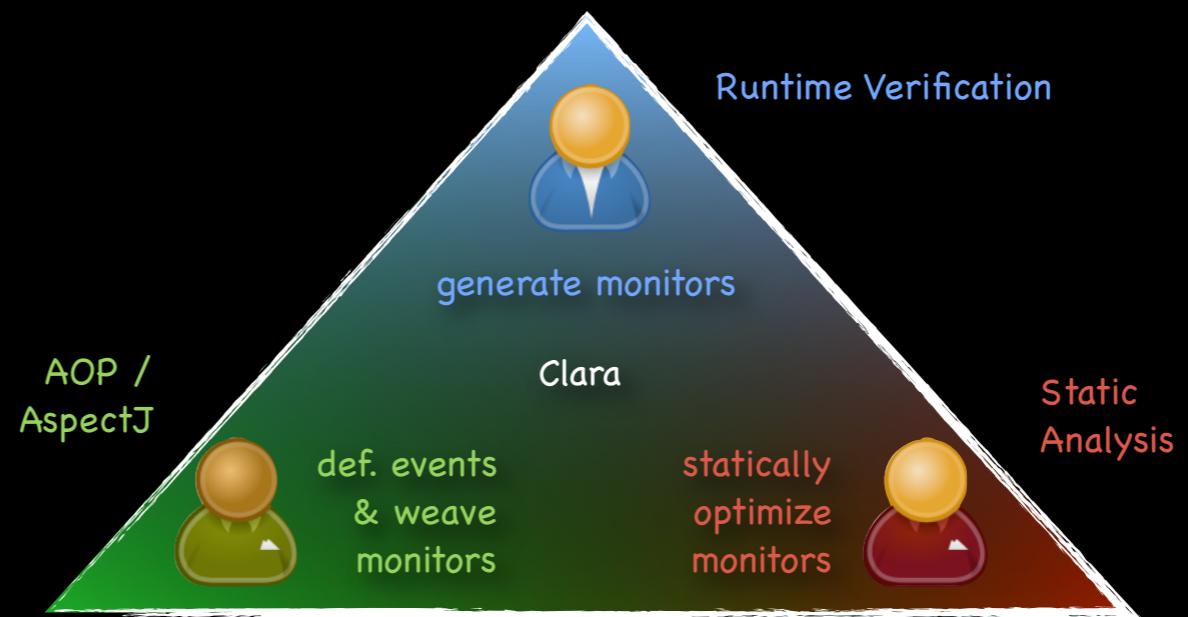
inverse analogous: $L_R \supseteq L$

Case 2: Extending an RV tool

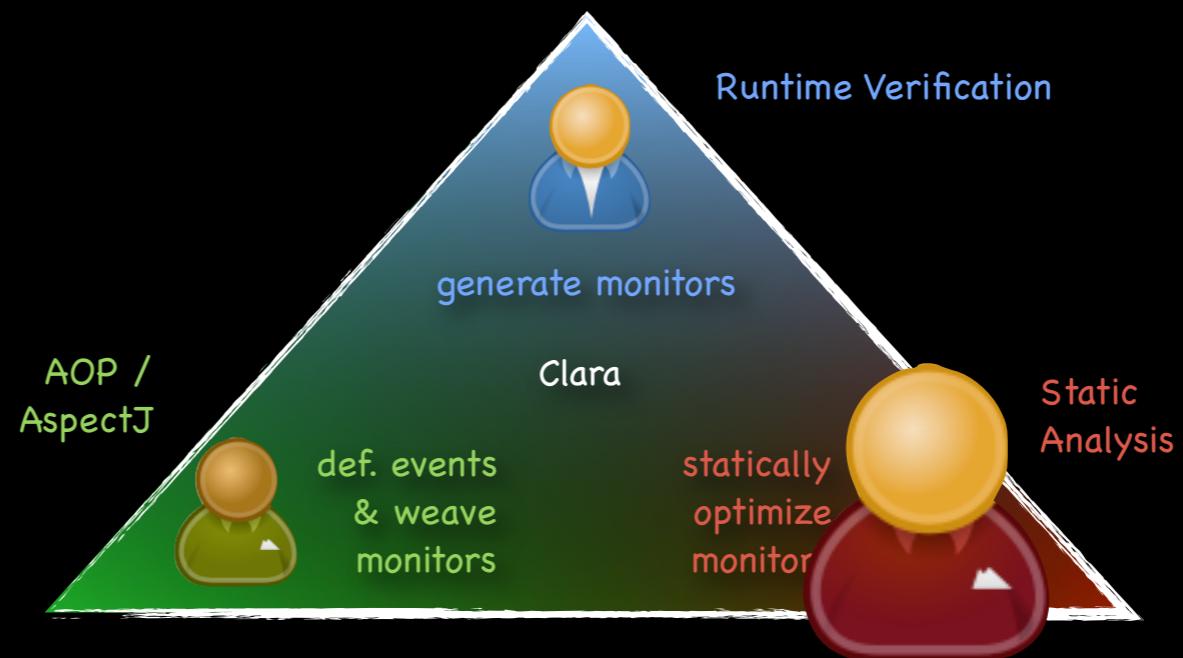
- Summary:

- if language of DSM is not exactly the monitored language, may lose some guarantees
- can avoid false positives/negatives by choosing a sub-/super-language
- false positives/negatives less likely if only Quick Check and Orphan-Shadows Analysis are used

Implementing your own static analyses



Implementing your own static analyses



Using Clara

```
$ java -jar clara-1.0.0-complete.jar -help
```

General Options:

| | |
|---------------------|---------------------------------|
| -h -help | Print the usage screen for abc. |
| -v -version | Print the abc version number. |
| -verbose | Verbose output. |
| @<filename> | |
| -argfile <filename> | Read arguments from a file. |

Input Options:

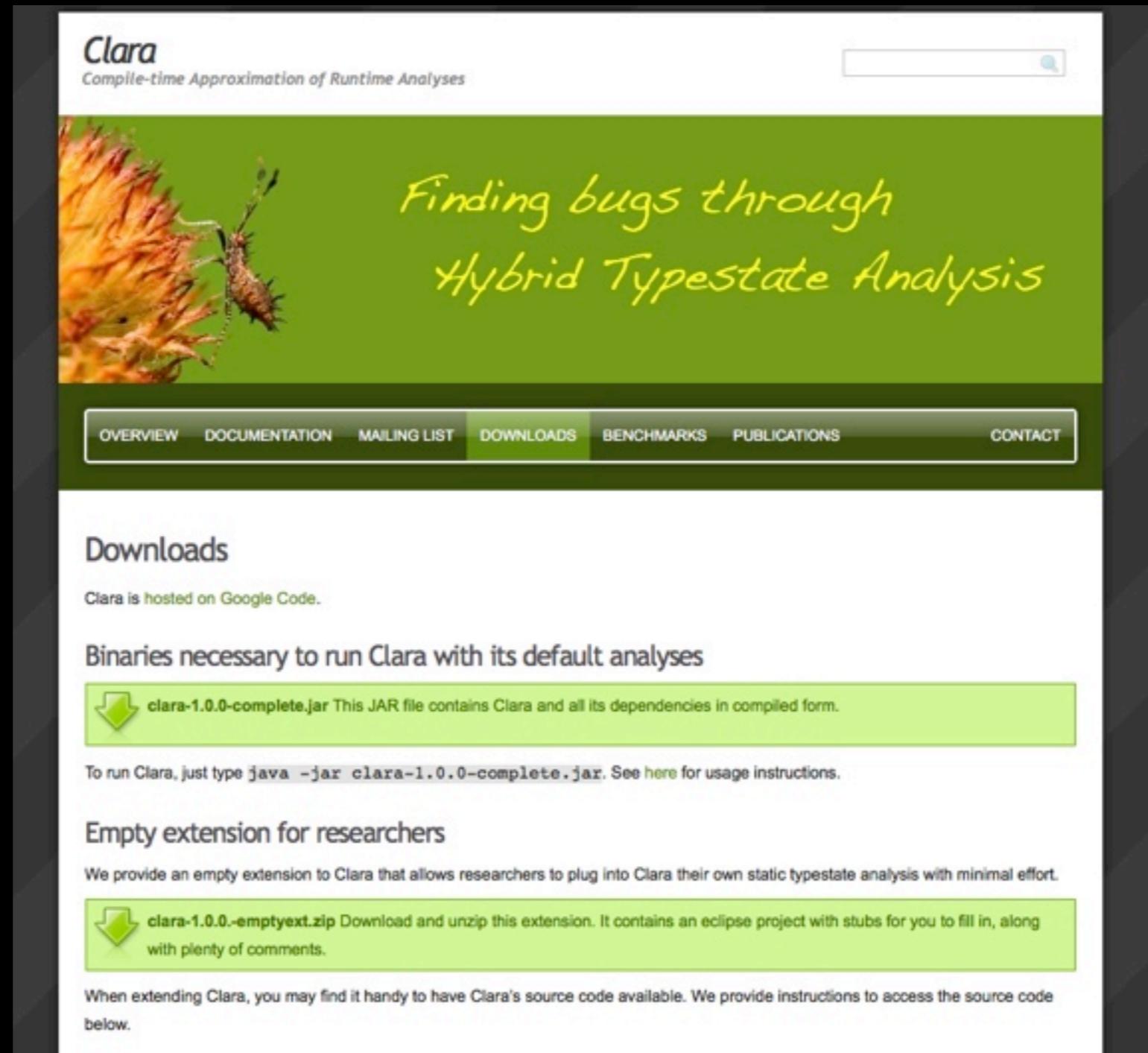
| | |
|------------------------|---|
| -sourceroots <path> | Use .java files in dirs in <path> as source. |
| -injars <jar list> | Use class files from the jars in <jar list> as source. |
| -inpath <dir list> | Use class files found in the directories in <dir list> as source. |
| -cp <classpath> | |
| -classpath <classpath> | Specify the class path to be used when searching for libraries. |
| -main-class <class> | Sets the main class for interprocedural analysis. |

...

Using Clara

```
$ java -Xmx1G \
-jar clara-1.0.0-complete.jar:your.jar \
-static-analyses quick-osa-nsa-yourAnalysis \
-injars yourTestProgram.jar \
-outjar instrumentedAndOptimized.jar \
monitors/MyMonitor.aj
```

Step 1: Download Clara & empty extension



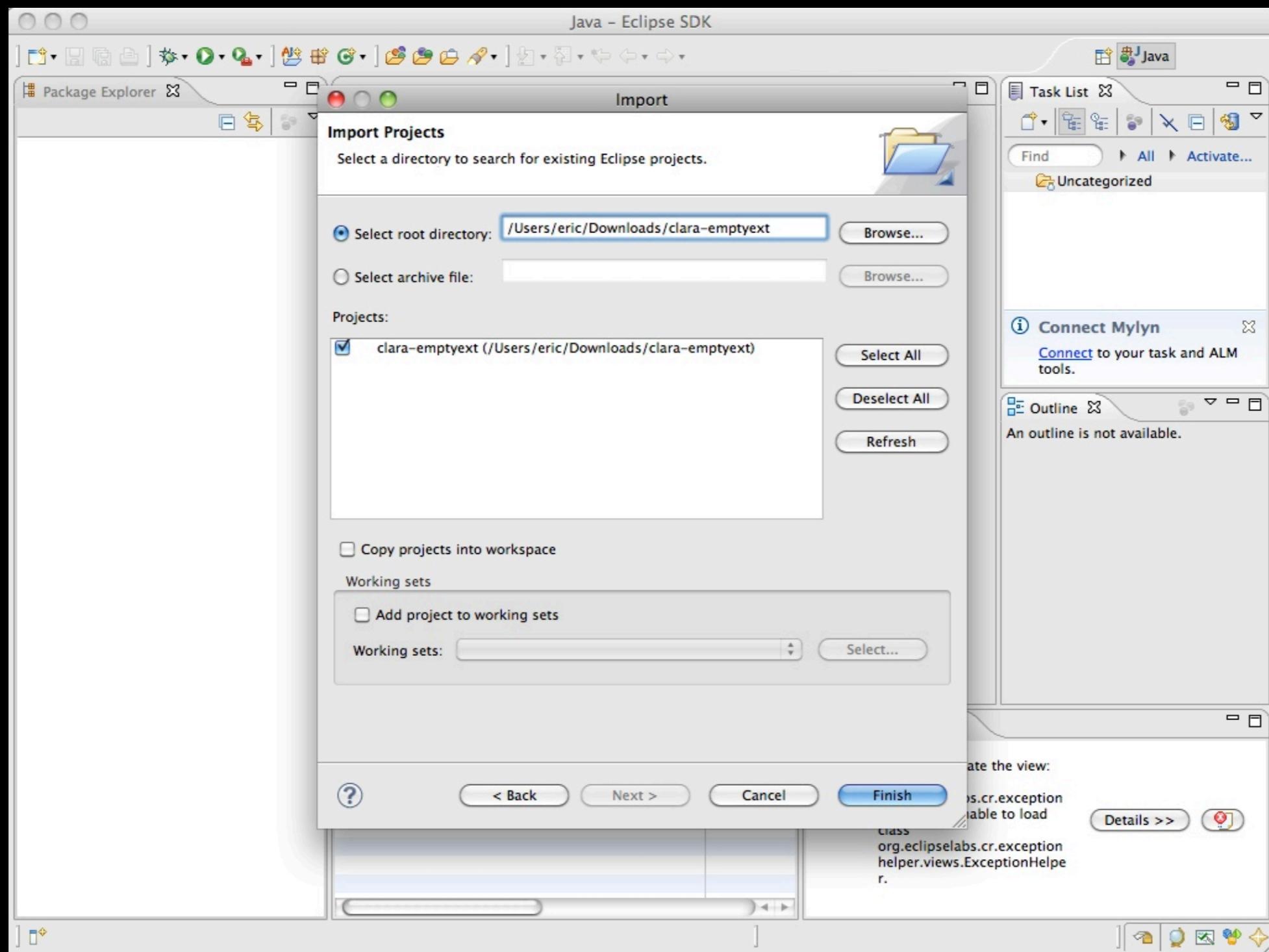
The screenshot shows the Clara project website's 'Downloads' page. The page features a green header with the text 'Finding bugs through Hybrid Typestate Analysis'. Below the header, there is a navigation bar with links: OVERVIEW, DOCUMENTATION, MAILING LIST, DOWNLOADS (which is highlighted), BENCHMARKS, PUBLICATIONS, and CONTACT. The main content area is titled 'Downloads' and contains two download links:

- Binaries necessary to run Clara with its default analyses**
clara-1.0.0-complete.jar This JAR file contains Clara and all its dependencies in compiled form.
- Empty extension for researchers**
clara-1.0.0-emptyext.zip Download and unzip this extension. It contains an eclipse project with stubs for you to fill in, along with plenty of comments.

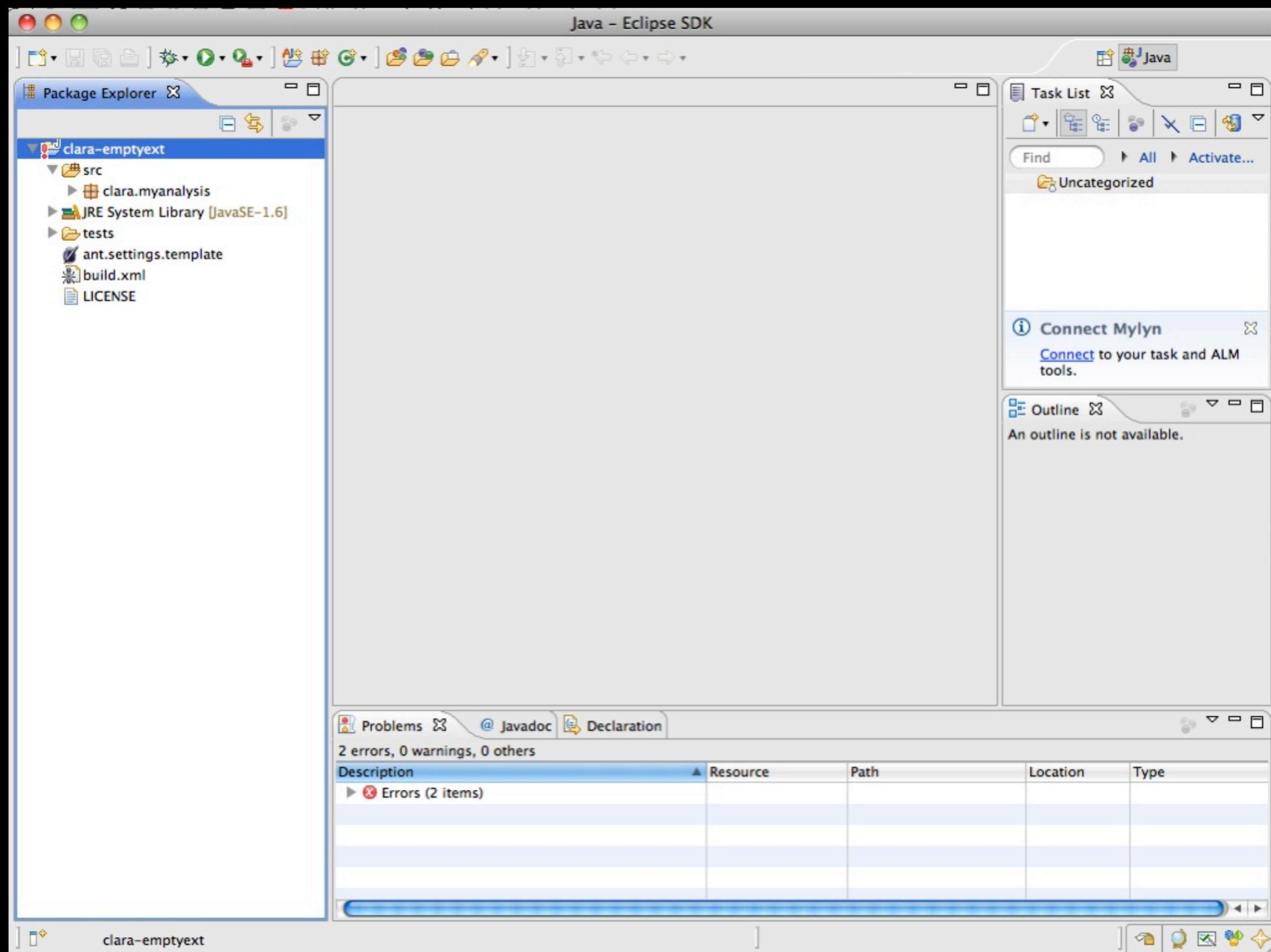
Below the download links, there is a note: 'To run Clara, just type `java -jar clara-1.0.0-complete.jar`. See [here](#) for usage instructions.'

<http://www.bodden.de/clara/downloads/>

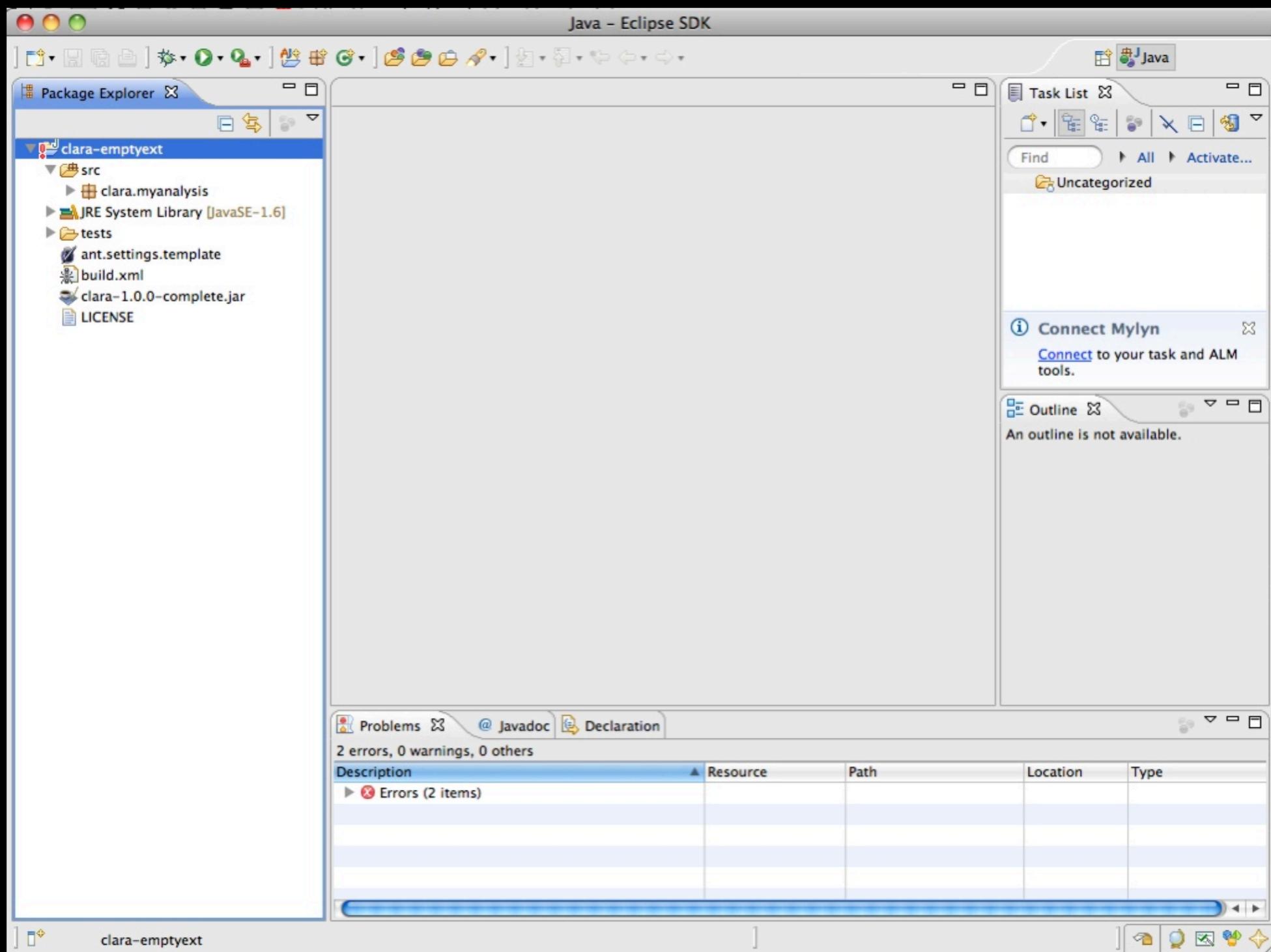
Step 2: Unzip & Import



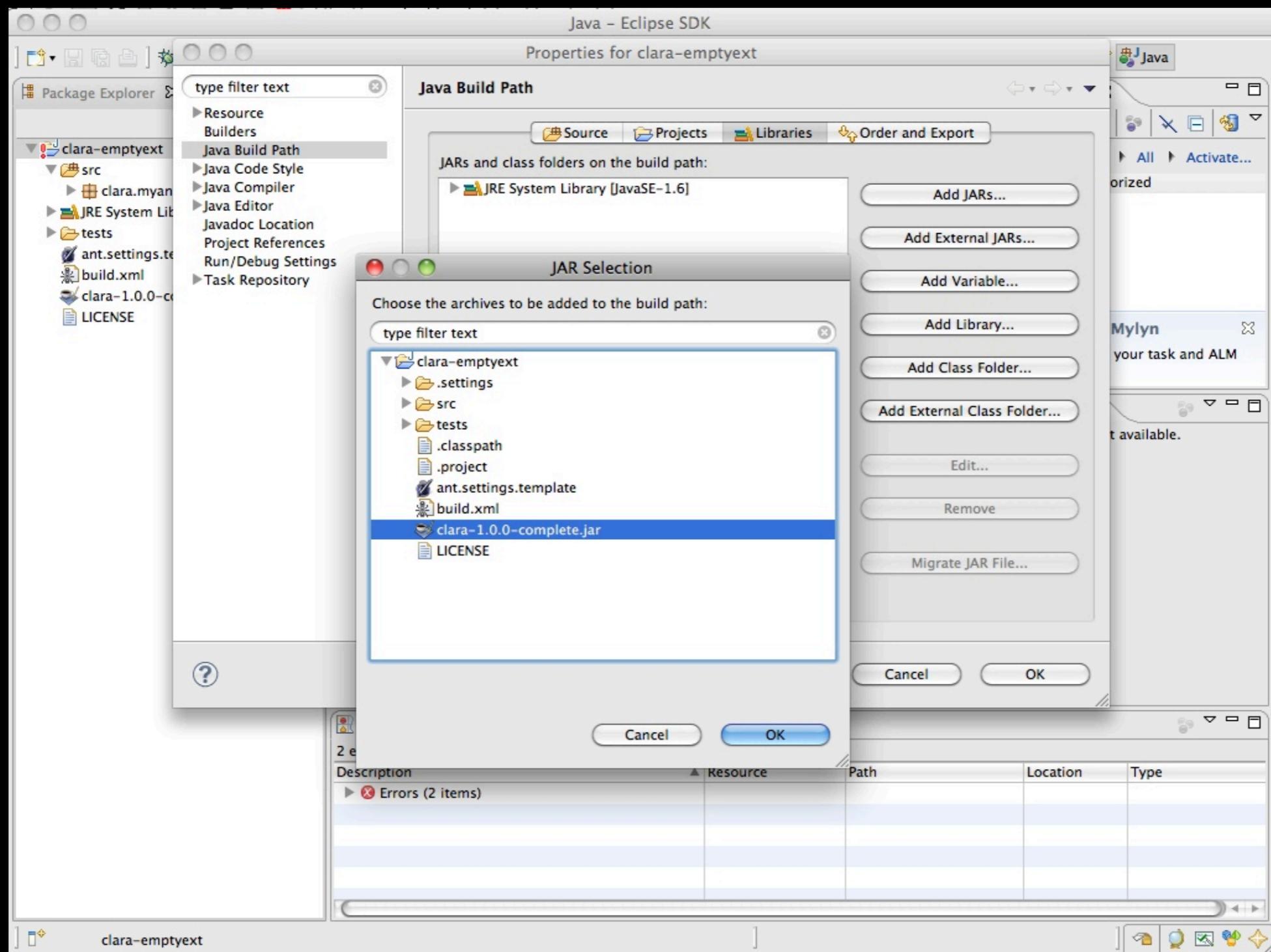
Step 2: Unzip & Import



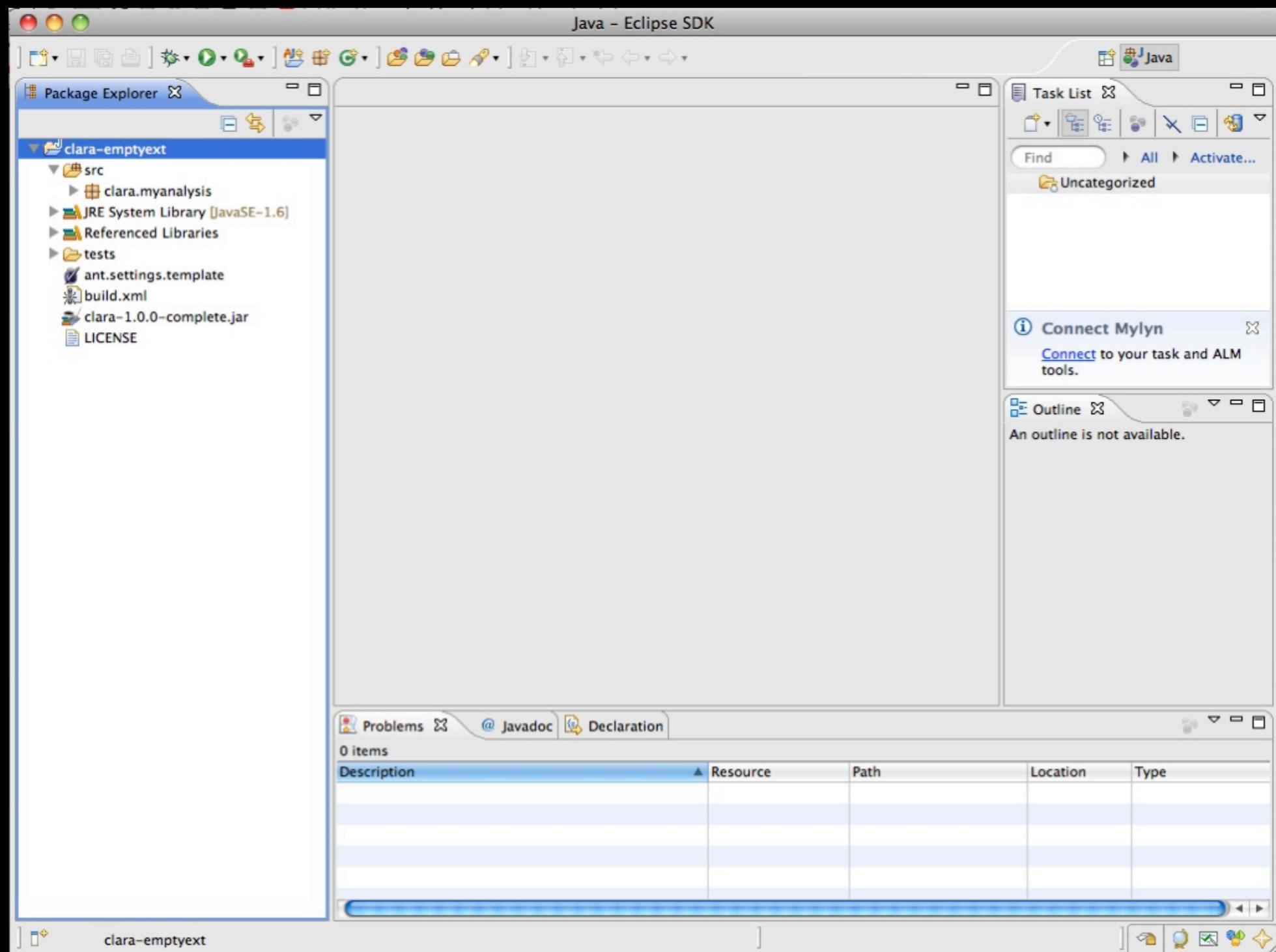
Step 3: Copy JAR file, adjust build path



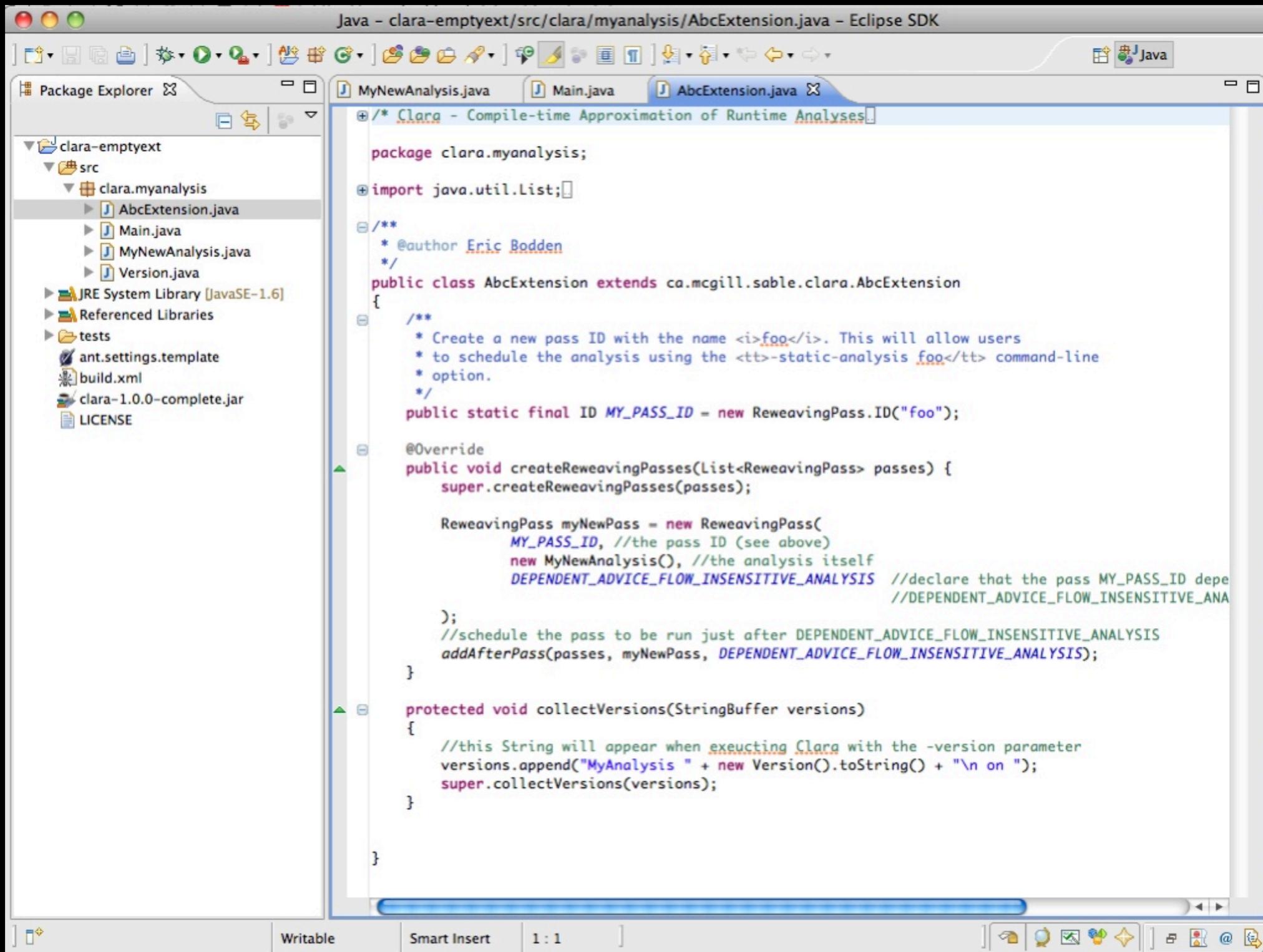
Step 3: Copy JAR file, adjust build path



Step 3: Copy JAR file, adjust build path



Hooking up the analysis



The screenshot shows the Eclipse IDE interface with the title "Java - clara-emptyext/src/clara/myanalysis/AbcExtension.java – Eclipse SDK". The left pane displays the "Package Explorer" with the project structure:

- clara-emptyext
- src
- clara.myanalysis
- AbcExtension.java (selected)
- Main.java
- MyNewAnalysis.java
- Version.java

The right pane shows the Java code for `AbcExtension.java`:/* Clara - Compile-time Approximation of Runtime Analyses */
package clara.myanalysis;

import java.util.List;

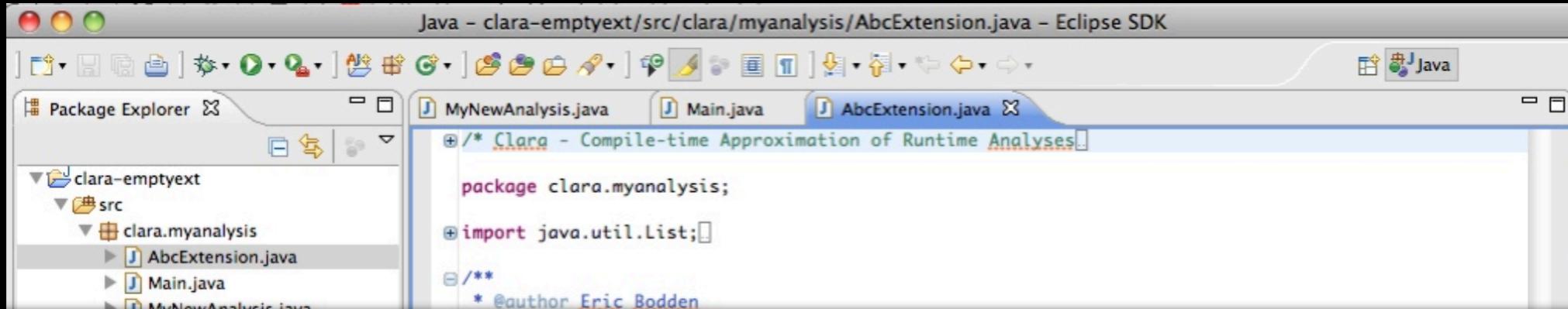
/**
 * @author Eric Bodden
 */
public class AbcExtension extends ca.mcgill.sable.clara.AbcExtension
{
 /**
 * Create a new pass ID with the name <i>foo</i>. This will allow users
 * to schedule the analysis using the <tt>-static-analysis foo</tt> command-line
 * option.
 */
 public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

 @Override
 public void createReweavingPasses(List<ReweavingPass> passes) {
 super.createReweavingPasses(passes);

 ReweavingPass myNewPass = new ReweavingPass(
 MY_PASS_ID, //the pass ID (see above)
 new MyNewAnalysis(), //the analysis itself
 DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on this analysis
);
 //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
 addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
 }

 protected void collectVersions(StringBuffer versions) {
 //this String will appear when executing Clara with the -version parameter
 versions.append("MyAnalysis " + new Version().toString() + "\n on ");
 super.collectVersions(versions);
 }
}

Hooking up the analysis



```
Java - clara-emptyext/src/clara/myanalysis/AbcExtension.java - Eclipse SDK
[...] Package Explorer [...] MyNewAnalysis.java Main.java AbcExtension.java [...]
+/* Clara - Compile-time Approximation of Runtime Analyses */
package clara.myanalysis;
import java.util.List;
/**
 * @author Eric Bodden
public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");
@Override
public void createReweavingPasses(List<ReweavingPass> passes) {
    super.createReweavingPasses(passes);

    ReweavingPass myNewPass = new ReweavingPass(
        MY_PASS_ID, //the pass ID (see above)
        new MyNewAnalysis(), //the analysis itself
        DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on the pass
                                                    //DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS being enabled,
    );
    //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
    addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
}

protected void collectVersions(StringBuffer versions)
{
    //this String will appear when executing Clara with the -version parameter
    versions.append("MyAnalysis " + new Version().toString() + "\n on ");
    super.collectVersions(versions);
}
}

Writable Smart Insert 1:1 [...]
```

Hooking up the analysis

The screenshot shows the Eclipse IDE interface with the title "Java - clara-emptyext/src/clara/myanalysis/AbcExtension.java – Eclipse SDK". The Package Explorer view on the left shows a project structure with files like AbcExtension.java, Main.java, and MyNewAnalysis.java. The main editor window displays Java code:

```
public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

@Override
public void createReweavingPasses(List<ReweavingPass> passes) {
    super.createReweavingPasses(passes);

    ReweavingPass myNewPass = new ReweavingPass(
        MY_PASS_ID, //the pass ID (see above)
        new MyNewAnalysis(),
        DEPENDENT_ADVICE_FLOW
    );
    //schedule the pass to be run
    addAfterPass(passes, myNewPass);
}
```

A callout box from the code highlights the `new MyNewAnalysis()` line, pointing to a tooltip for the `MyNewAnalysis` class. The tooltip content is:

MyNewAnalysis

- analyze(): boolean
- new ShadowDisabledListener() ...
- defaultSootArgs(List<String>): void
- enforceSootArgs(List<String>): void
- setupWeaving(): void
- tearDownWeaving(): void
- cleanup(): void

pass
g enabled,

analyze() method

```
public boolean analyze(){

/*
 * The most important information that we need to deal with is contained in the Dependent-Advice Info...
 */

DAInfo dai = ((HasDAInfo)Main.v()).getAbcExtension().getDependentAdviceInfo();

/* a tracepattern is essentially a Dependency State Machine;
 * it combines a state machine with information about the machine's
 * alphabet
 */
Set<TracePattern> tracePatterns = dai.getTracePatterns();
for (TracePattern tracePattern : tracePatterns) {

    tracePattern.getContainer();                                //returns the aspect that declares the pattern
    tracePattern.getContainerClass();                           //returns the SootClass that implements this aspect
    tracePattern.getFinalSymbols();                            //returns all symbols in the alphabet that lead into a final state
    tracePattern.getFormals();                                 //returns all variable names that are declared for this trace pattern
    tracePattern.getInitialSymbols();                         //returns all symbols in the alphabet that lead out of an initial state
    tracePattern.getName();                                   //returns an informal, unqualified name for this trace pattern;
    tracePattern.getStateMachine();                          //returns the state machine for this trace pattern
    tracePattern.getSymbolAdviceMethod("foo"); //returns the SootMethod that implements the monitoring advice for
    tracePattern.getSymbols();                             //returns all declared symbols of this trace pattern
    tracePattern.getVariableOrder("foo"); //returns the ordered list of variable names (see getFormals()) that

}
```

Shadows

```
//Let's have a look at all reachable shadows in the program... it's as easy as this!
EnabledShadowSet reachableActiveShadows = dai.getReachableActiveShadows();
/* The above call requires a call graph to already have been constructed.
 * Because we scheduled our pass to run after the Orphan-Shadows Analysis, which
 * instructs Soot to construct a call graph, this is no problem.
 */
for (Shadow shadow : reachableActiveShadows) {
    System.err.println(shadow);

    shadow.getAdviceBodyInvokeStmt();           //returns the statement that invokes the appropriate advice at the
    shadow.getAdviceDecl();                    //returns the advice that implements the transition for this shadow
    shadow.getAdviceFormalNames();              //returns the formal-parameter names of the variables that the shadow
    shadow.getAdviceFormalToSootLocal();         //returns a mapping from formal-parameter names to Soot Locals that
    shadow.getBoundSootLocals();                //returns the set of all Soot Locals that the shadow's advice takes
    shadow.getContainer();                     //returns the SootMethod that contains the shadow
    shadow.getID();                           //returns the unique ID of this shadow
    shadow.getPosition();                     //returns the source code position at which this shadow resides (if
    shadow.getSootLocalForAdviceFormalName("x"); //returns the local for formal parameter "x" (see getAdviceFormalName)
    //shadow.conjoinResidueWith(rhsResidue);   //conjoins this shadow's residue with another residue; useful e.g.
    //shadow.disable();                      //disables this shadow, setting its residue to NeverMatch and notifying
    shadow.registerListener(new ShadowDisabledListener() { //registers a new listener that is notified when the shadow
        public void shadowDisabled(Shadow shadow) {      //EnabledShadowSet is an example of a class that uses
            }
    });
}
```

EnabledShadowSets

```
int sizeBefore = reachableActiveShadows.size();

//nothing to do
if(sizeBefore==0) return false;

//An EnabledShadowsSet is a volatile thing. When we disable a shadow...
reachableActiveShadows.iterator().next().disable();

//then this automatically removes the shadow from the set!
assert reachableActiveShadows.size() == sizeBefore-1;

/* This allows analyses to automatically take into account disabled shadows when being
 * re-iterated. To get a copy of the set that stays as it is, produce a snapshot...
 */

Set<Shadow> snapshot = reachableActiveShadows.snapshot();

sizeBefore = snapshot.size();

//when we disable a shadow here...
reachableActiveShadows.iterator().next().disable();

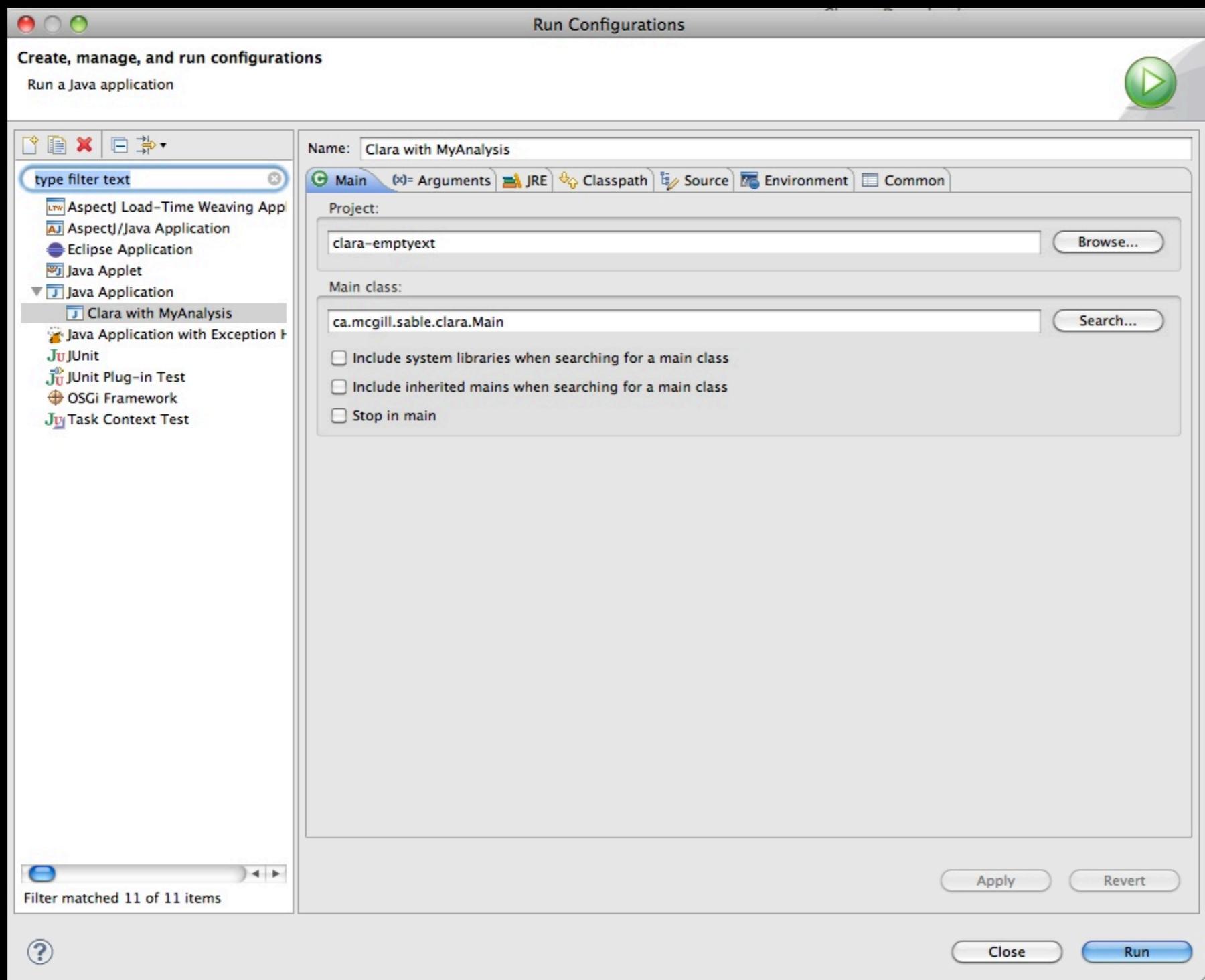
//then the snapshot remains as is
assert snapshot.size() == sizeBefore;

//do not reweave immediately
return false;
```

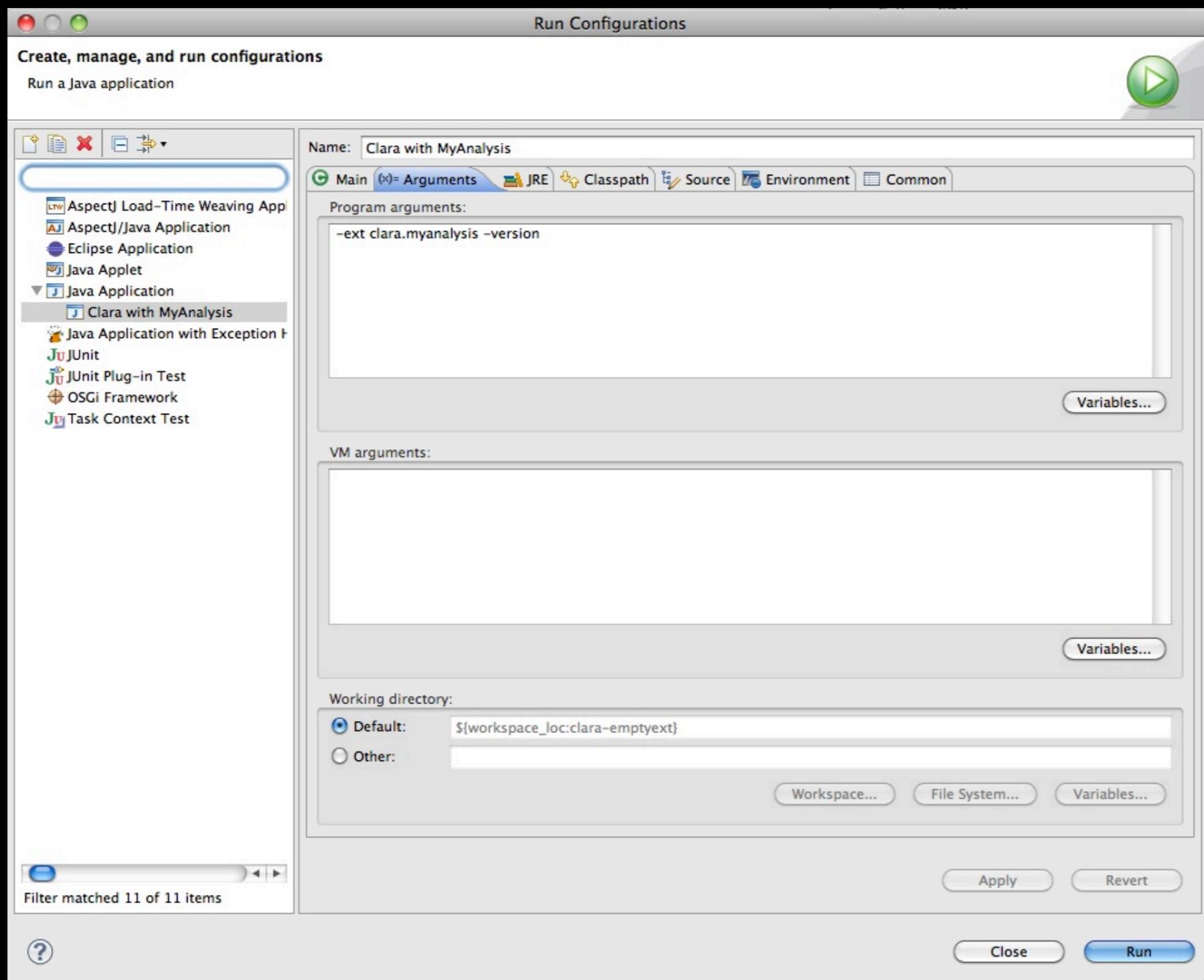
Example analyses

- ⦿ in package ca.mcgill.sable.clara.weaving.weaver.depadvceopt
- ⦿ Quick Check:
DependentAdviceQuickCheck
- ⦿ Orphan-Shadows Analysis:
DependentAdviceFlowInsensitiveAnalysis
- ⦿ Nop-Shadows Analysis:
DependentAdviceIntraproceduralAnalysis

Using the extension



Using the extension

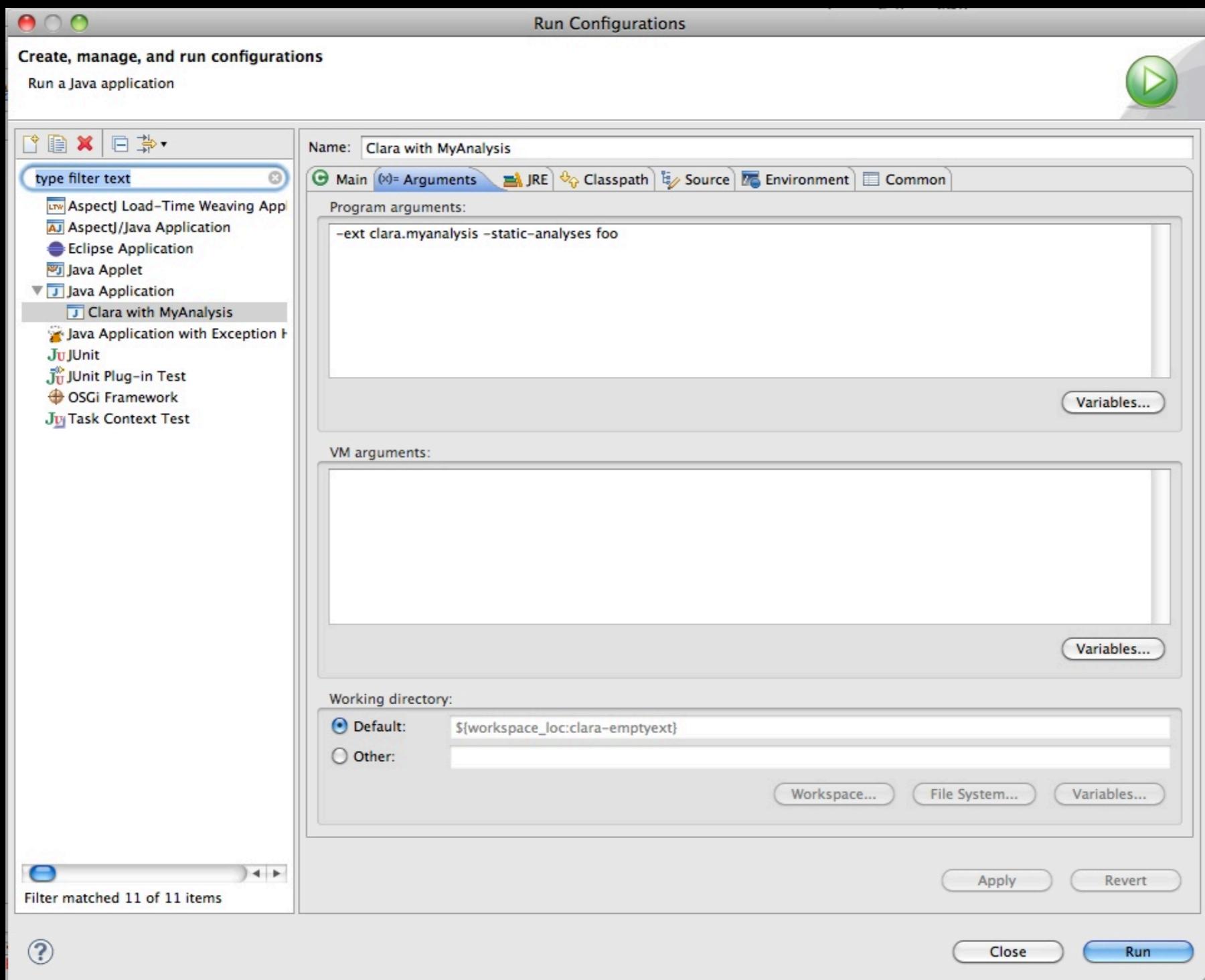


Using the extension

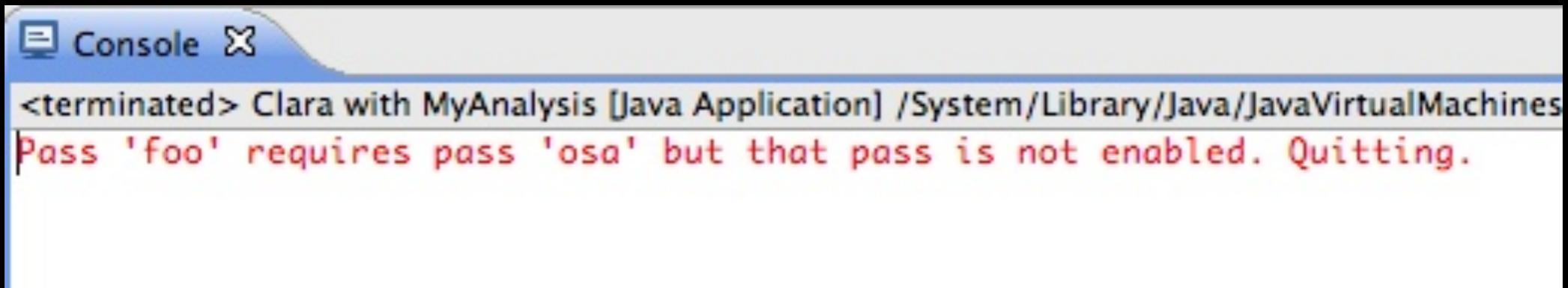
```
Console X
<terminated> Clara with MyAnalysis [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Content/Jav
MyAnalysis 0.0.0
on Clara 1.0.0
with abc version 1.3.0.DEV
... with JastAdd frontend 0.2.0.DEV
with EAJ (JastAdd version) 1.3.0.DEV
... using Soot toolkit version 2.3.0
... using Polyglot compiler toolkit version 1.3.0
For usage, abc -help
-----
Copyright (C) 2004-2006 The abc development team. All rights reserved.
See the file CREDITS for a list of contributors.
See individual source files for details.

Soot is Copyright (C) 1997-2006 Raja Vallee-Rai and others.
```

Using the extension

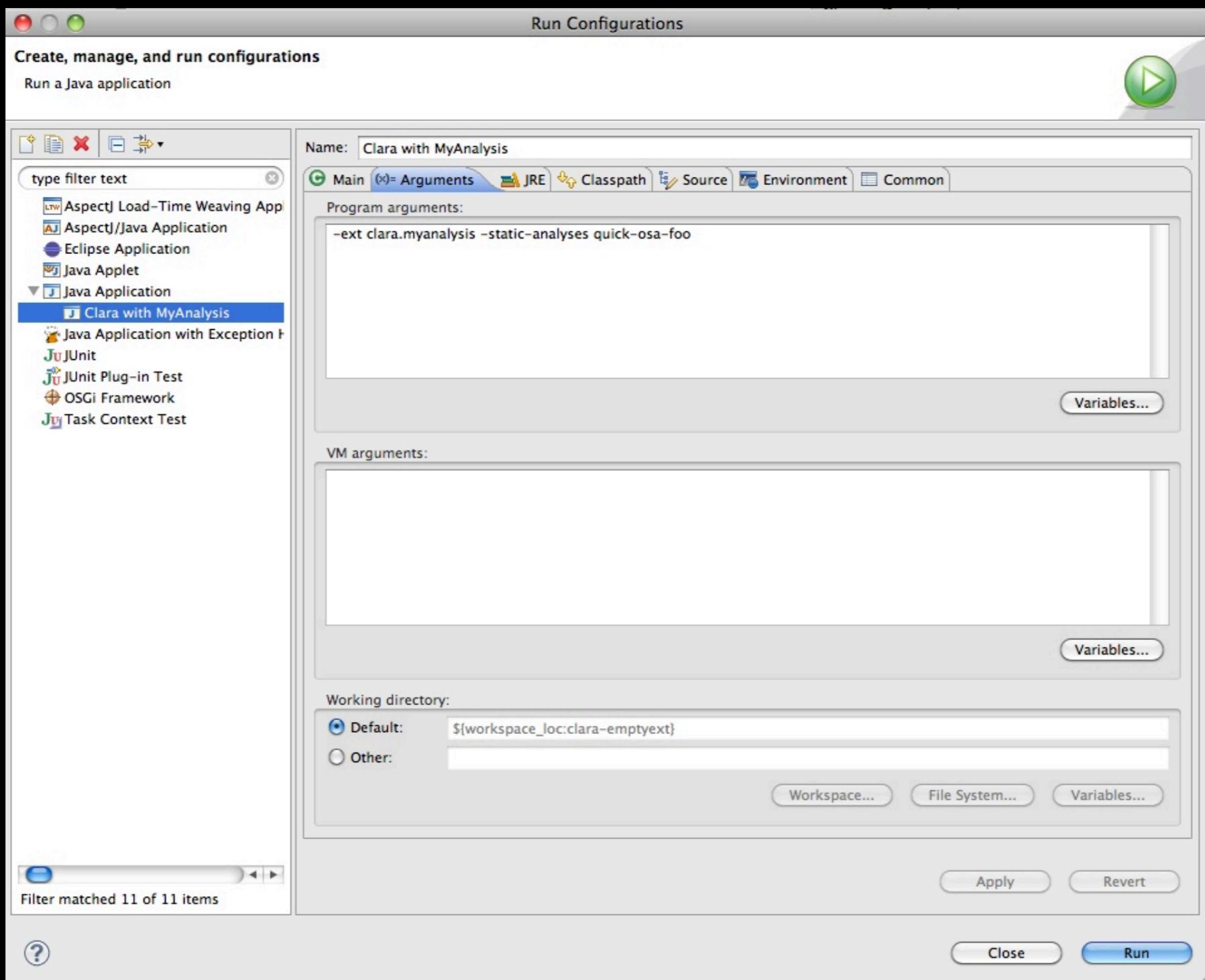


Using the extension



The screenshot shows a Java application window titled "Console". The title bar includes standard window controls (minimize, maximize, close) and the title "Console". The main area of the window displays a command-line interface. The output starts with "<terminated> Clara with MyAnalysis [Java Application] /System/Library/Java/JavaVirtualMachines", followed by a red error message: "Pass 'foo' requires pass 'osa' but that pass is not enabled. Quitting." This indicates that the application failed to start due to a dependency issue with the "osa" pass.

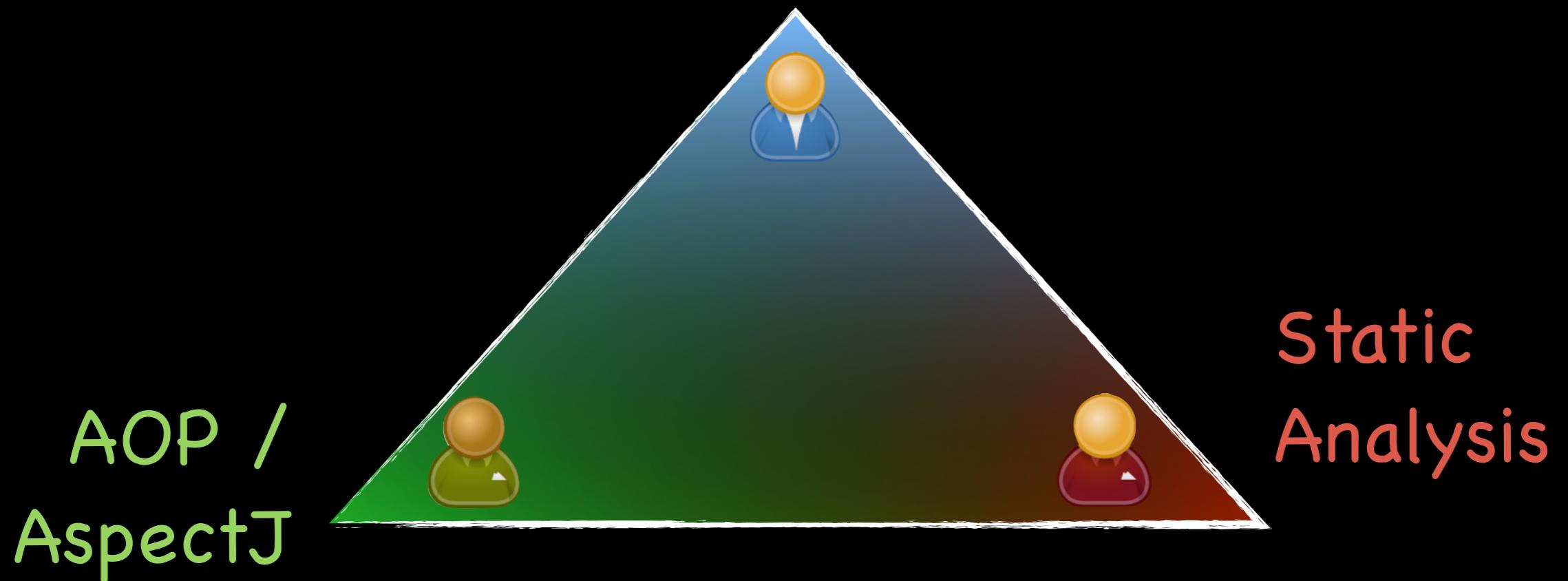
Using the extension



Related work

Related and previous work

Runtime Verification



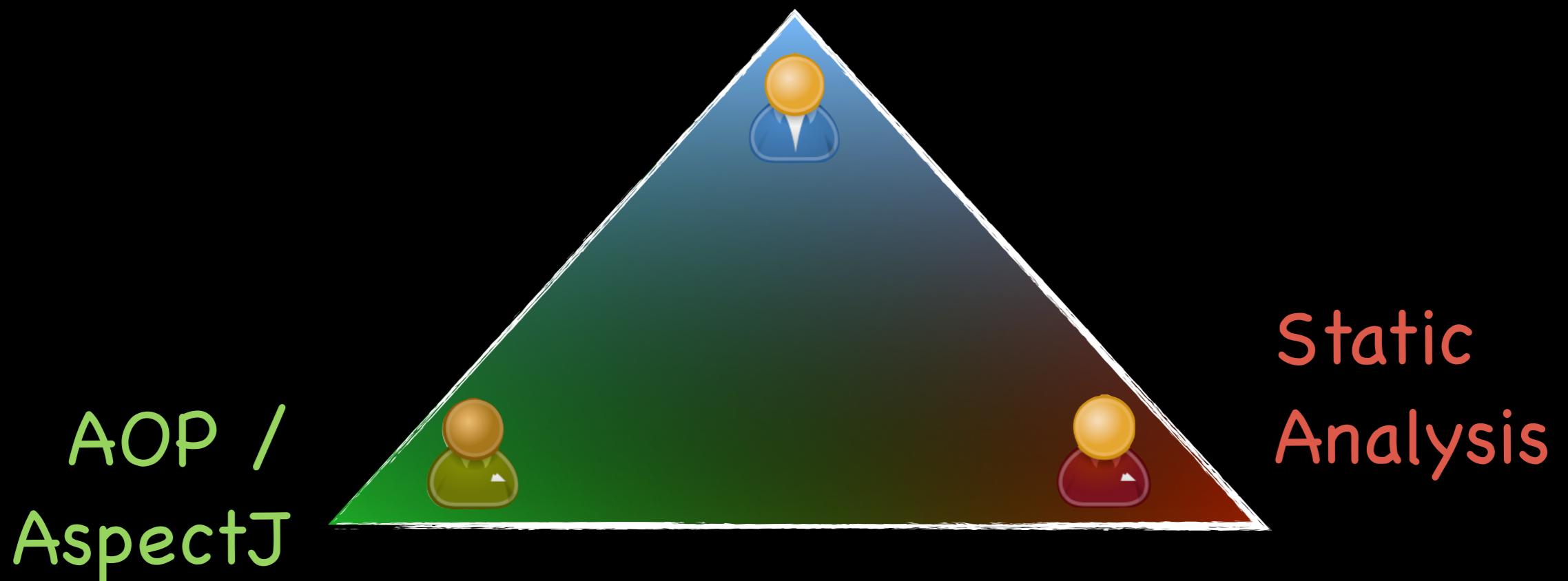
Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)



Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

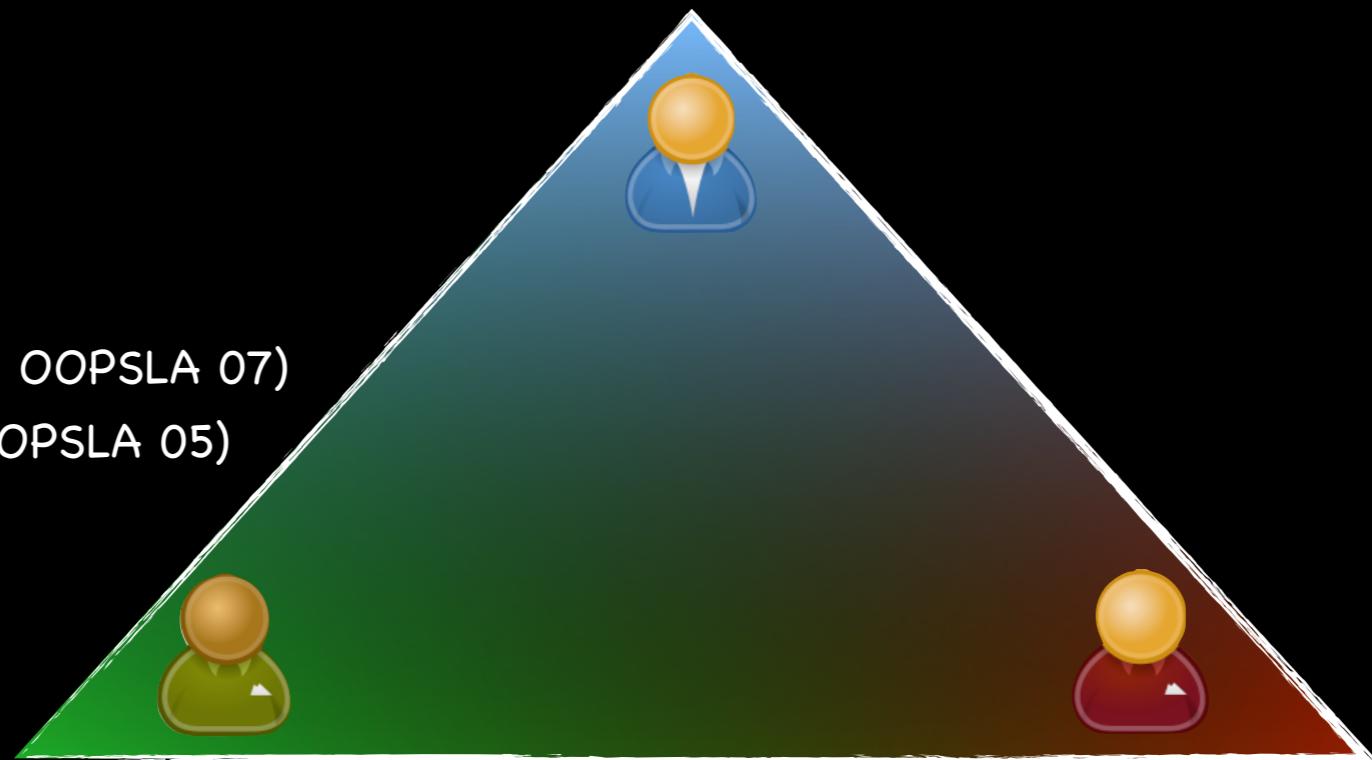
PQL (Martin et al., OOPSLA 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ

Static
Analysis



Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

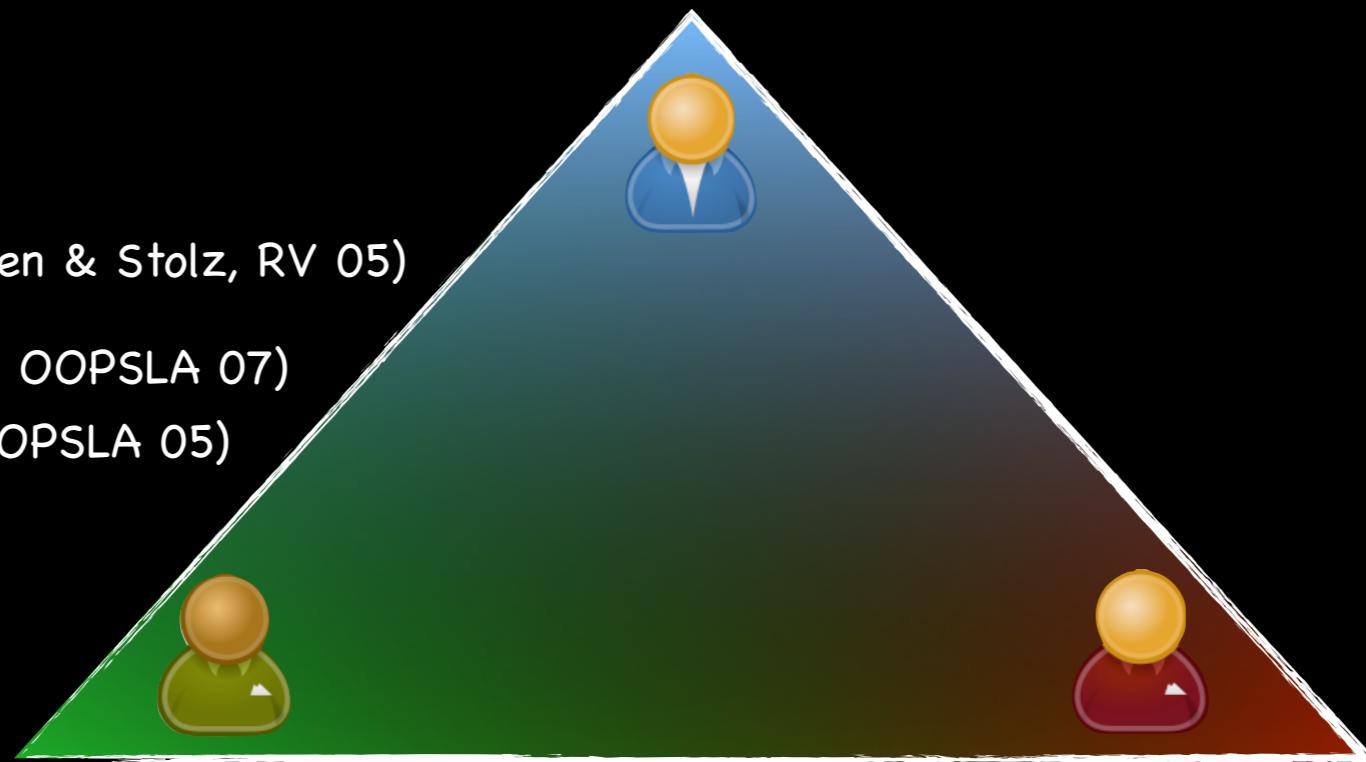
PQL (Martin et al., OOPSLA 05)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ



Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)

M2Aspects (Krüger et al., SCESM 06)

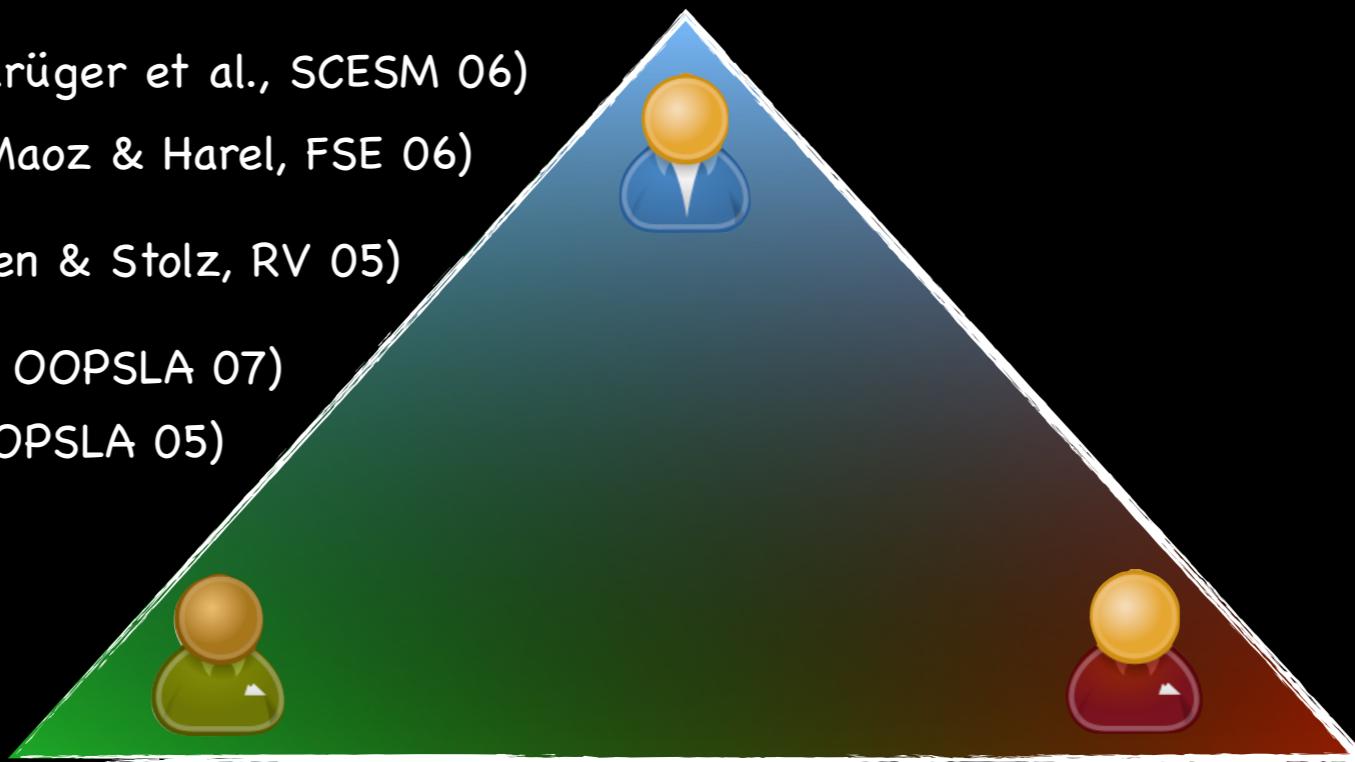
S2A (Maoz & Harel, FSE 06)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ



Static
Analysis

Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)

M2Aspects (Krüger et al., SCESM 06)

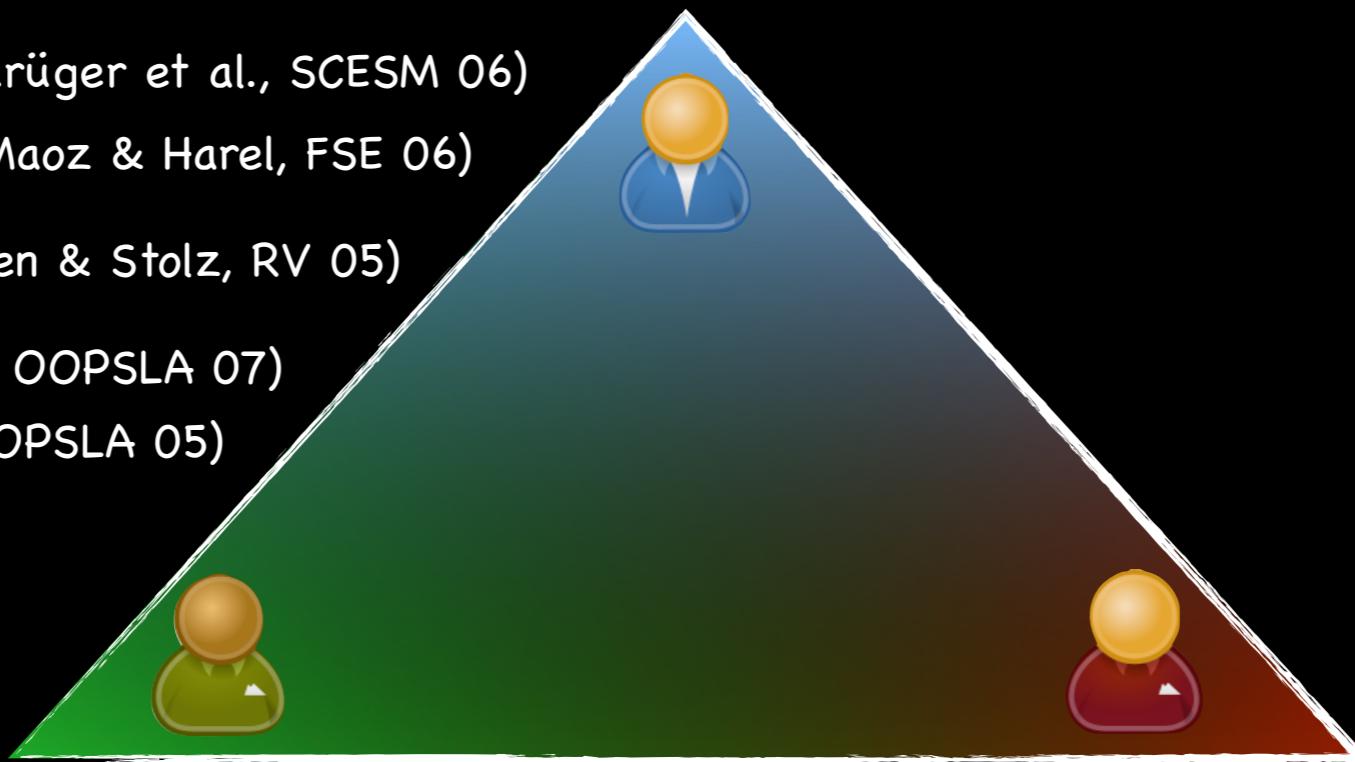
S2A (Maoz & Harel, FSE 06)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ



Static Typestate Analysis (Fink et al., ISSTA 06)

Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)

M2Aspects (Krüger et al., SCESM 06)

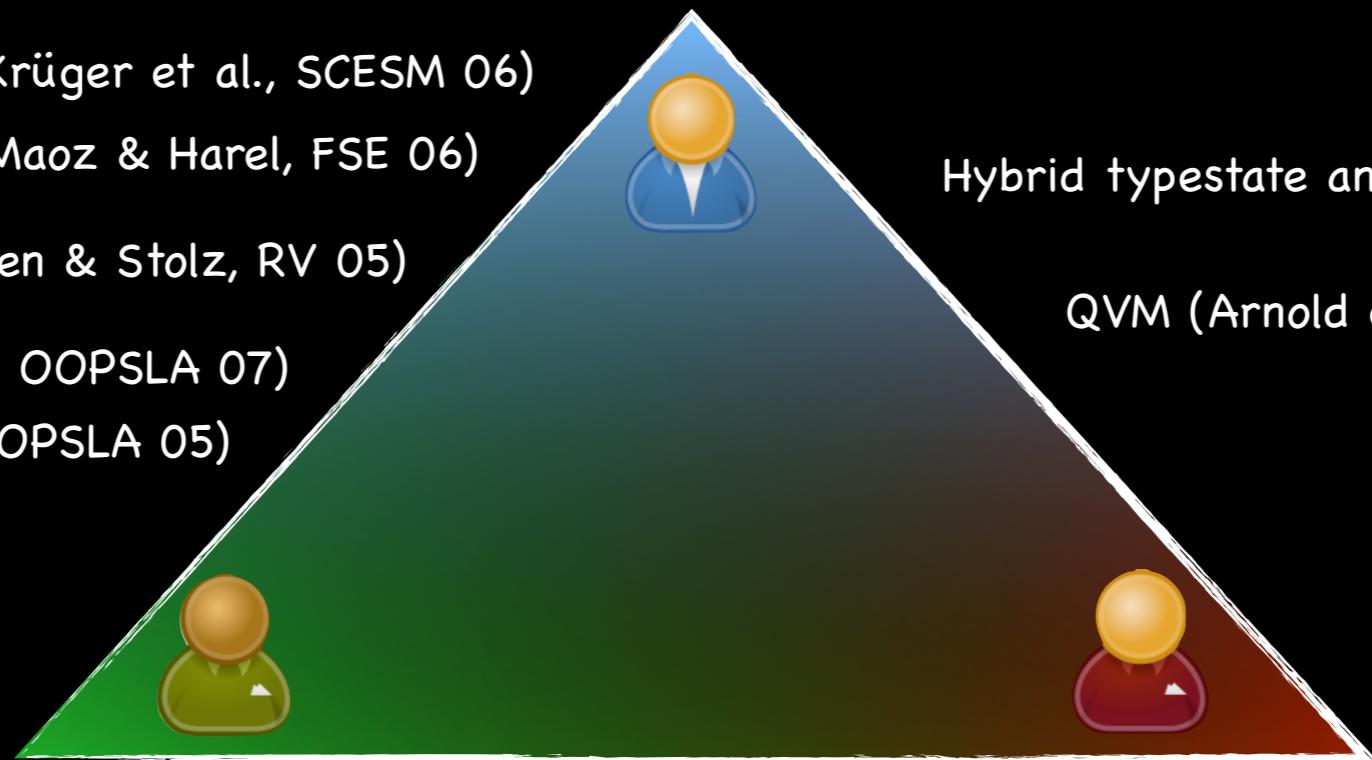
S2A (Maoz & Harel, FSE 06)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ



Static
Analysis

Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)

M2Aspects (Krüger et al., SCESM 06)

S2A (Maoz & Harel, FSE 06)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ

SCoPE (Aotani & Masuhara, AOSD 07)

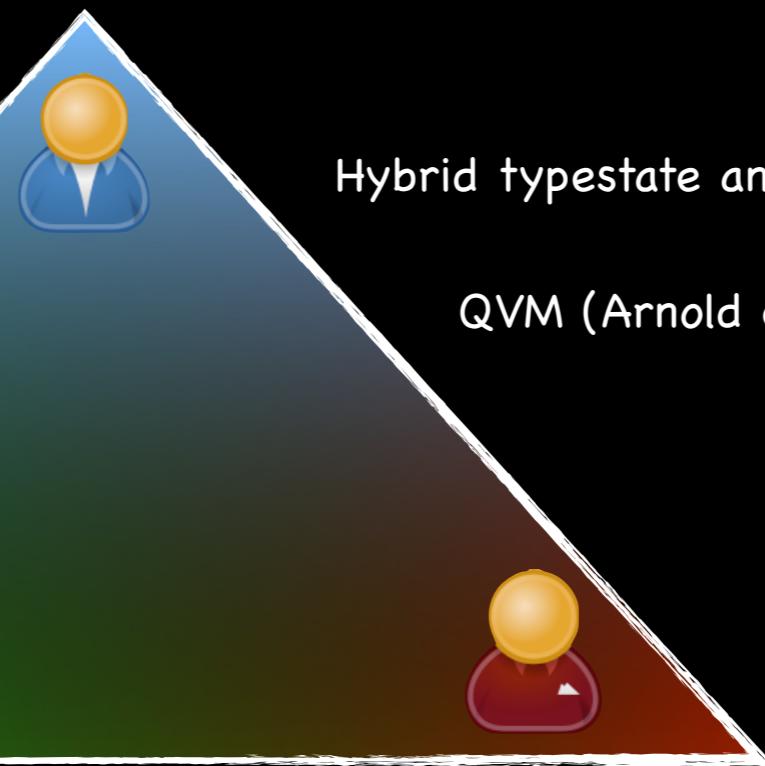
Optimizing cflow (Avgustinov et al., PLDI 05)

Statically optimizing tracematches
(Naeem & Lhotak, OOPSLA 08; Bodden et al. ECOOP 07, FSE 08)

Hybrid typestate analysis (Dwyer et al., ASE 07)

QVM (Arnold et al., OOPSLA 08)

Static
Analysis



Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)

M2Aspects (Krüger et al., SCESM 06)

S2A (Maoz & Harel, FSE 06)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

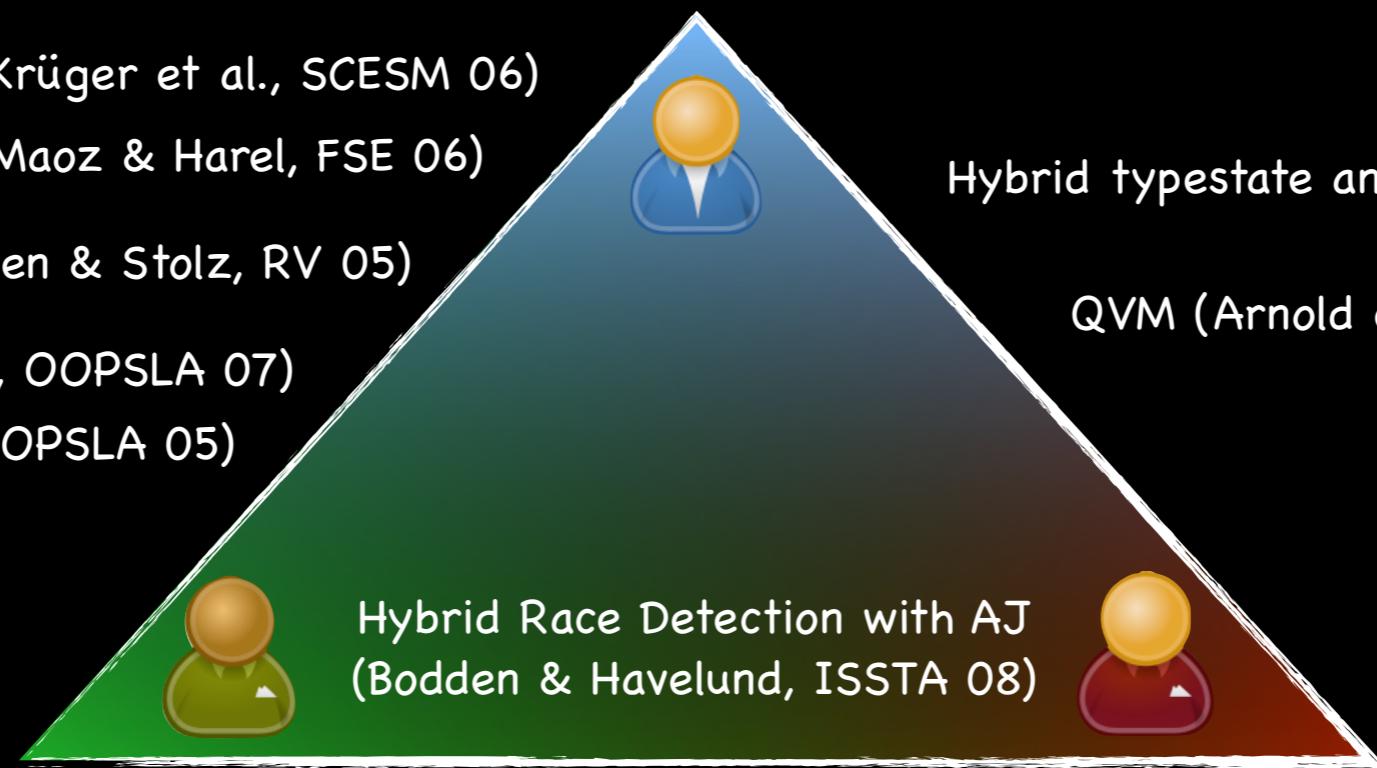
Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ

SCoPE (Aotani & Masuhara, AOSD 07)

Optimizing cflow (Avgustinov et al., PLDI 05)

Statically optimizing tracematches
(Naeem & Lhotak, OOPSLA 08; Bodden et al. ECOOP 07, FSE 08)



Related and previous work

Runtime Verification

Tracecuts (Walker & Viggers, FSE 04)

PTQL (Goldsmith et al., OOPSLA 05)

PQL (Martin et al., OOPSLA 05)

M2Aspects (Krüger et al., SCESM 06)

S2A (Maoz & Harel, FSE 06)

J-LO (Bodden & Stolz, RV 05)

JavaMOP (Chen et al., OOPSLA 07)

Tracematches (Allan et al., OOPSLA 05)

AOP /
AspectJ

SCoPE (Aotani & Masuhara, AOSD 07)

Optimizing cflow (Avgustinov et al., PLDI 05)

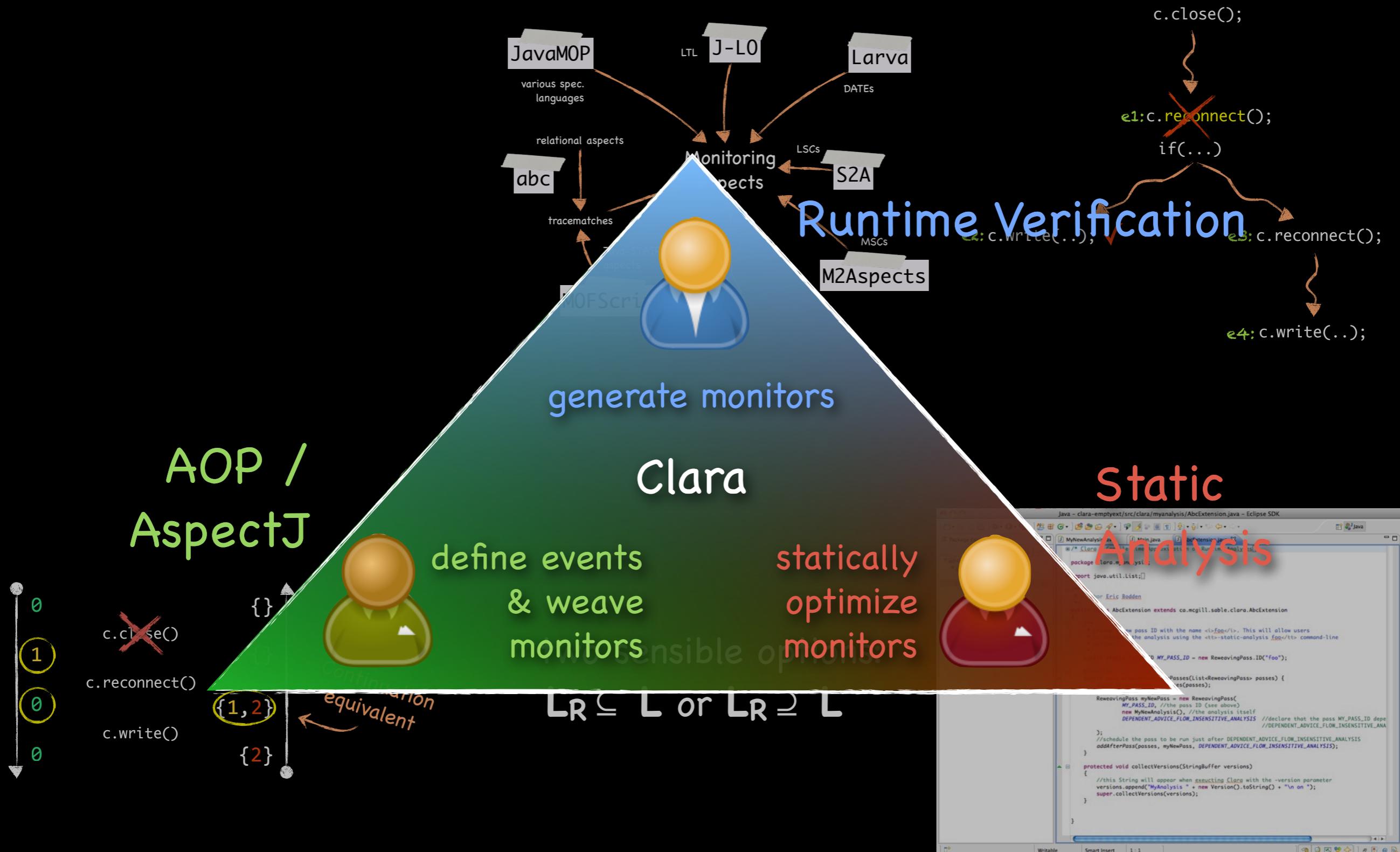
Statically optimizing tracematches
(Naeem & Lhotak, OOPSLA 08; Bodden et al. ECOOP 07, FSE 08)

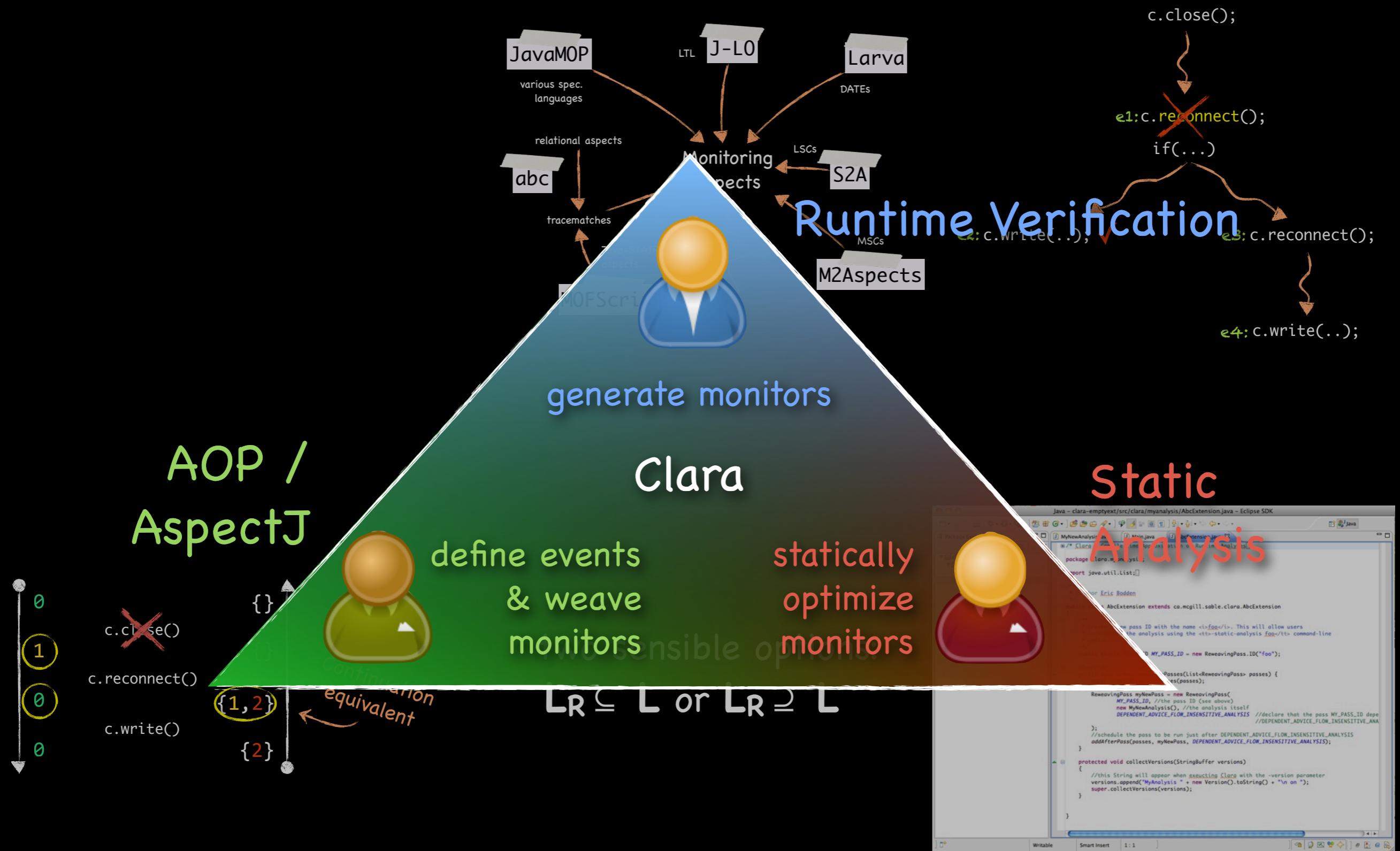
Clara

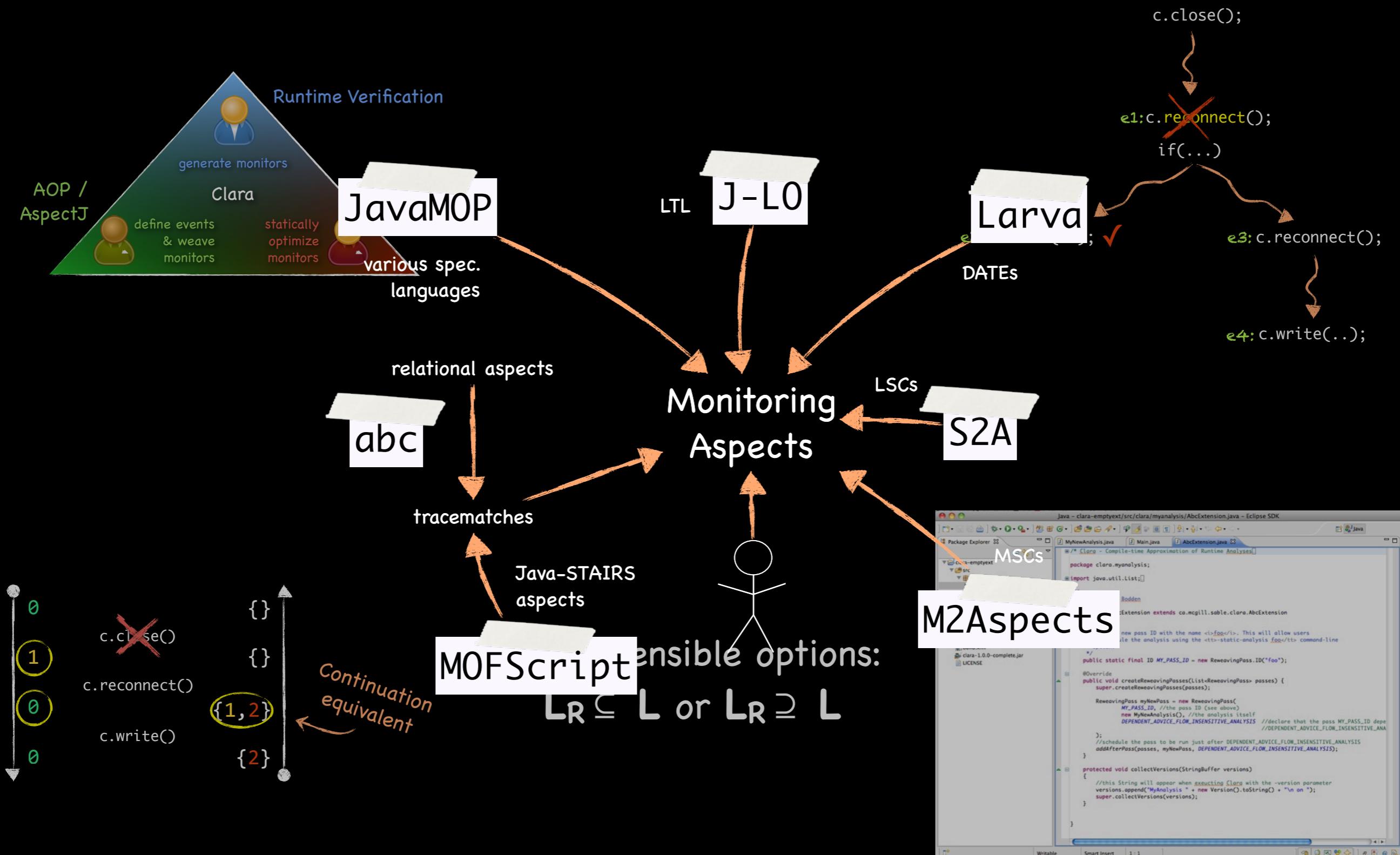
Hybrid Race Detection with AJ
(Bodden & Havelund, ISSTA 08)

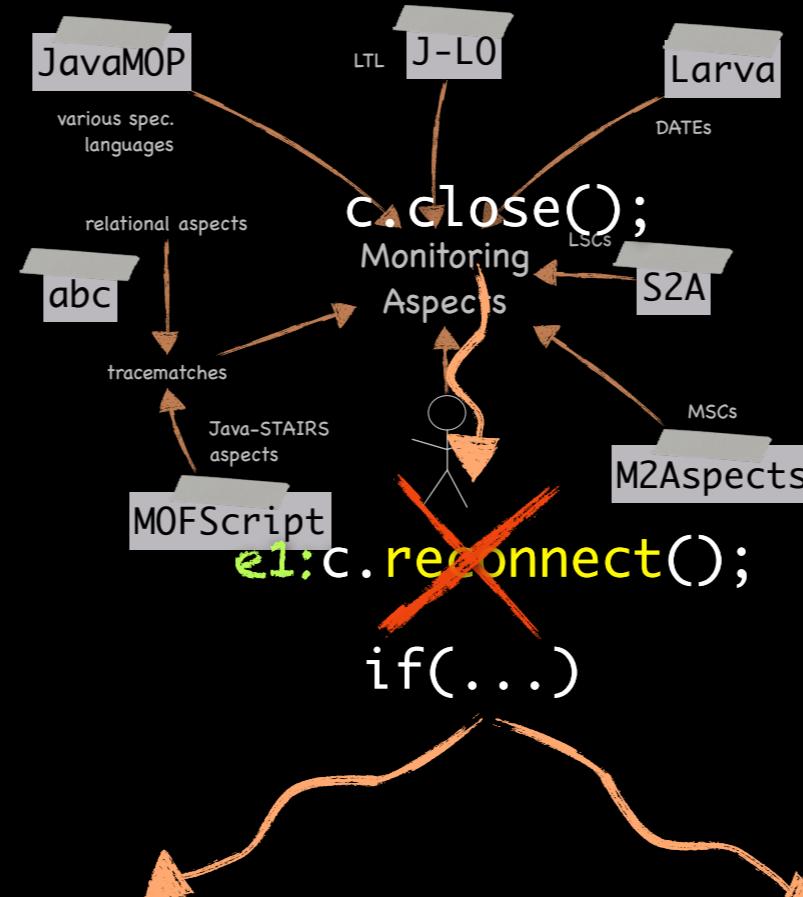
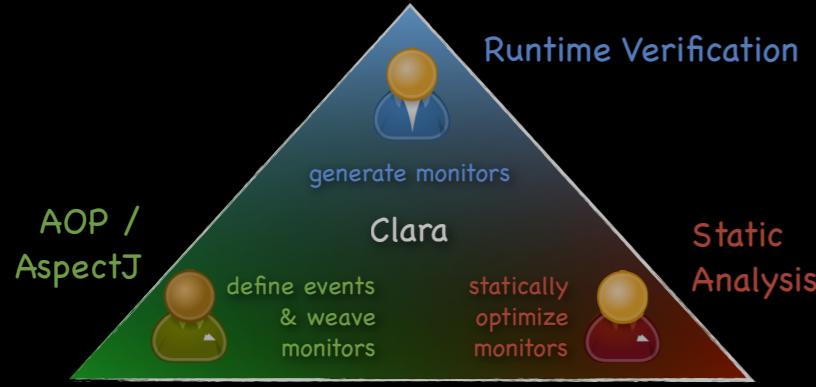
Static
Analysis

Static Typestate Analysis (Fink et al., ISSTA 06)



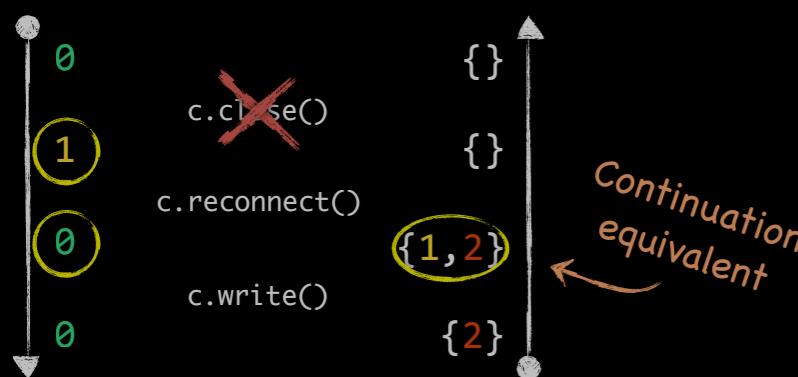






~~e2: c.write(..); ✓~~

~~e3: c.reconnect();~~



two sensible options:

$$L_R \subseteq L \text{ or } L_R \supseteq L$$

~~e4: c.write(..);~~

```

package clara.myanalysis;

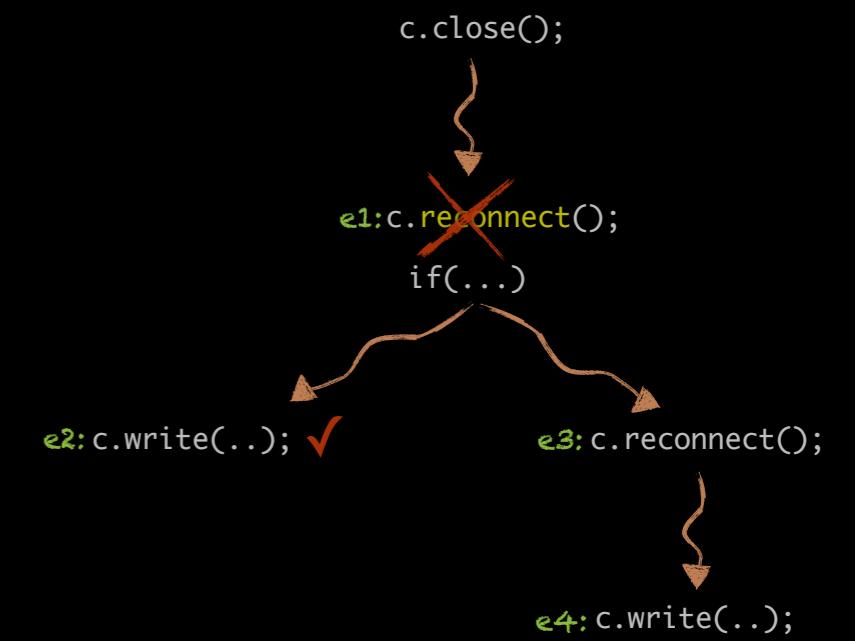
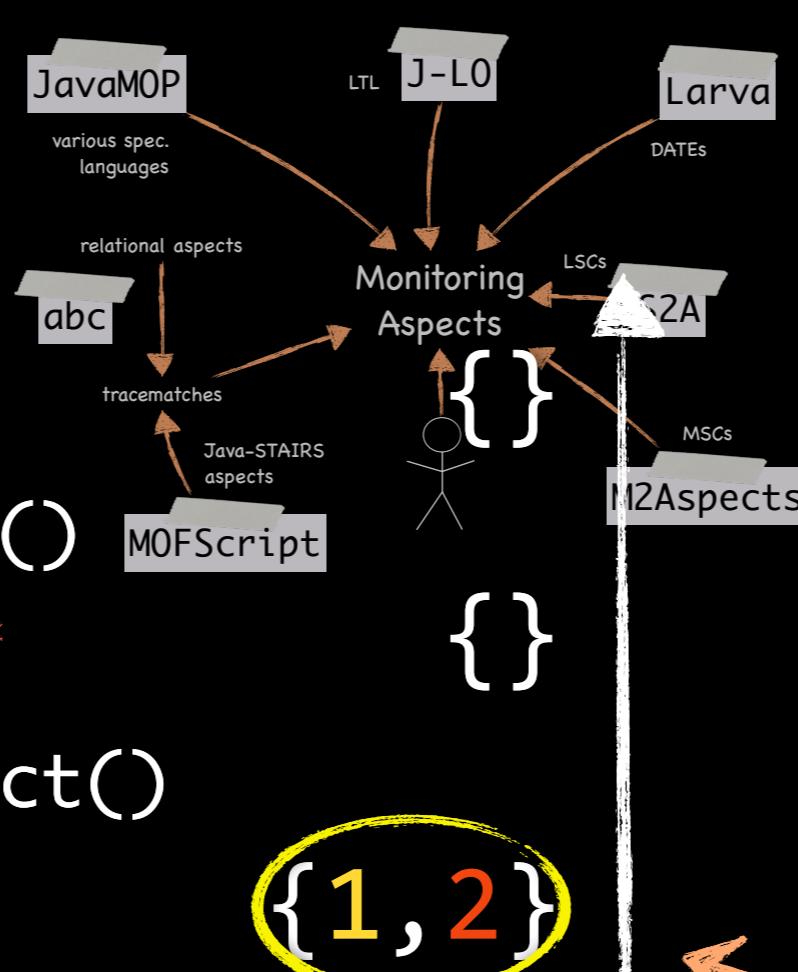
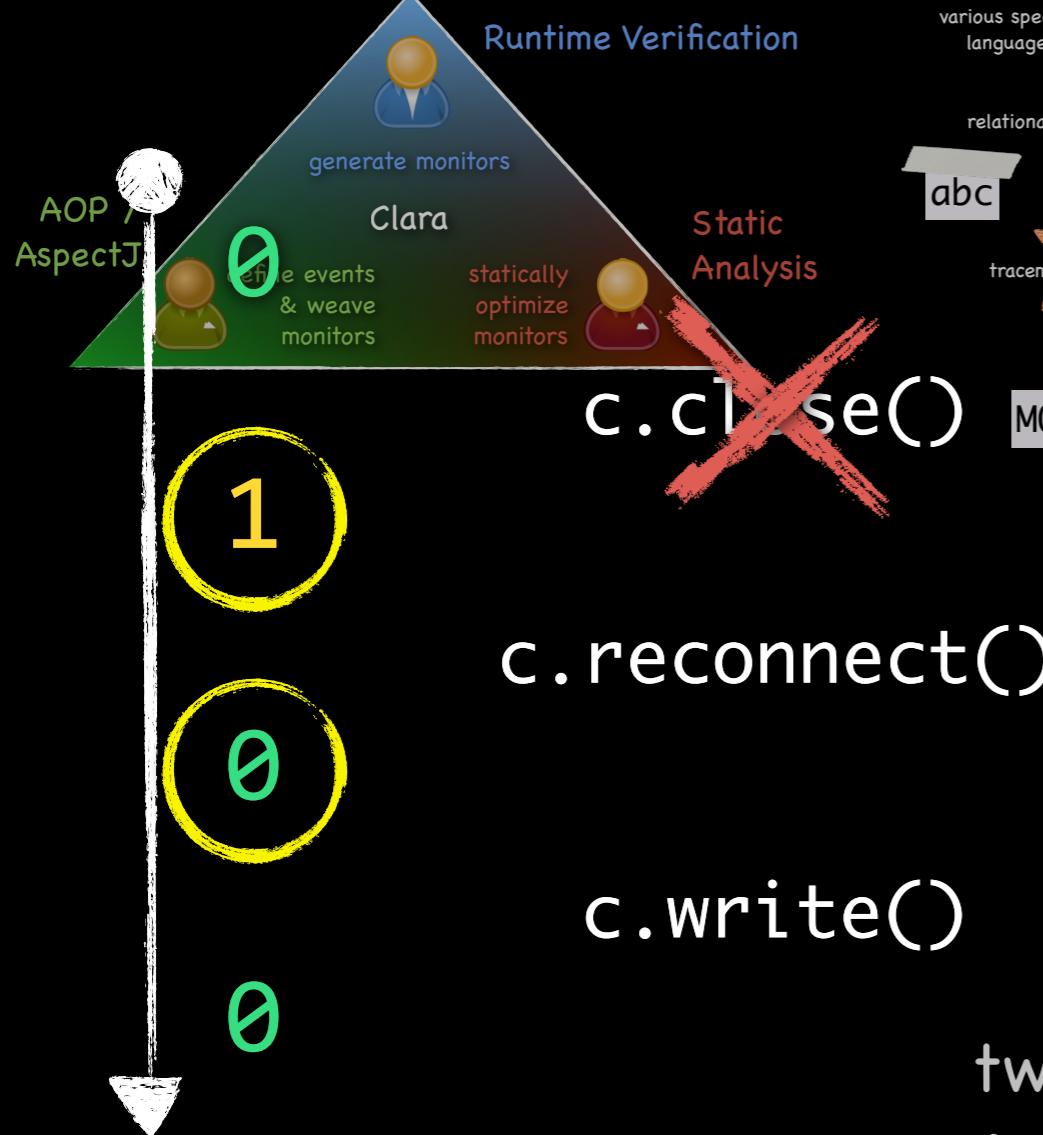
import java.util.List;

/*
 * Author: Eric Bodden
 */
public class AbcExtension extends ca.mcgill.sable.clara.AbcExtension {
    /**
     * Create a new pass ID with the name <i>foo</i>. This will allow users
     * to schedule the analysis using the <tt>-static-analysis foo</tt> command-line
     * option.
     */
    public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

    @Override
    public void createReweavingPasses(List<ReweavingPass> passes) {
        super.createReweavingPasses(passes);
        ReweavingPass myNewPass = new ReweavingPass(
            MY_PASS_ID, //the pass ID (see above)
            new MyNewAnalysis(), //the analysis itself
            DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on it
        );
        //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
        addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
    }

    protected void collectVersions(StringBuffer versions) {
        //this String will appear when executing Clang with the -version parameter
        versions.append("MyAnalysis " + new Version().toString() + "\n on ");
        super.collectVersions(versions);
    }
}

```



Continuation
equivalence

```

package clara.myanalysis;
import java.util.List;

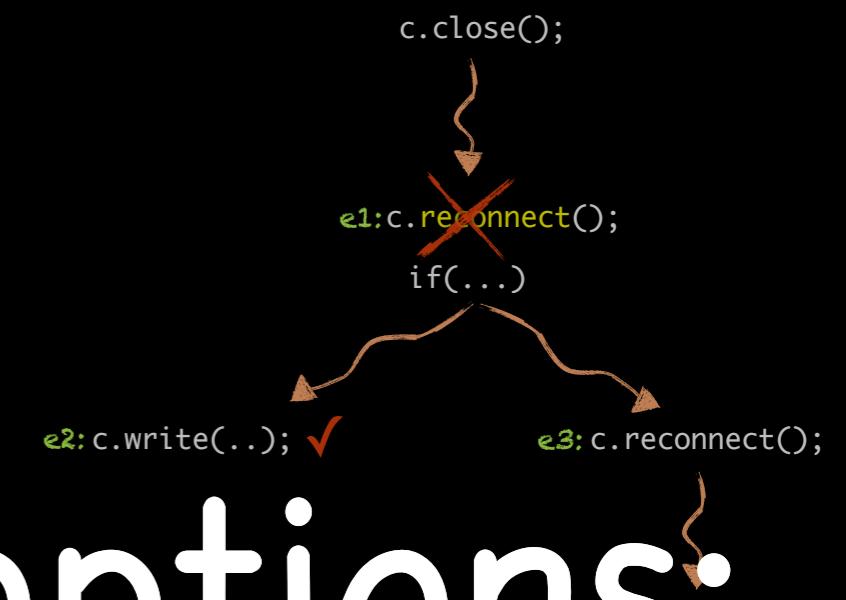
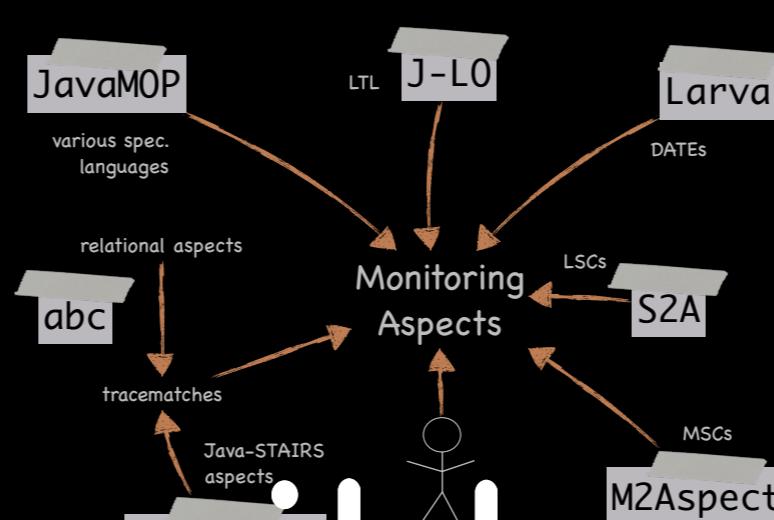
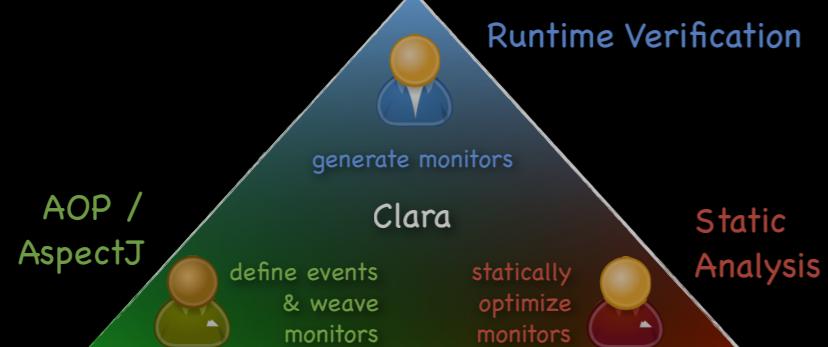
/*
 * Author Eric Bodden
 */
public class AbcExtension extends ca.mcgill.sable.clara.AbcExtension
{
    /**
     * Create a new pass ID with the name <i>foo</i>. This will allow users
     * to schedule the analysis using the <tt>-static-analysis foo</tt> command-line
     * option.
     */
    public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

    @Override
    public void createReweavingPasses(List<ReweavingPass> passes) {
        super.createReweavingPasses(passes);
    }

    ReweavingPass myNewPass = new ReweavingPass(
        MY_PASS_ID, //the pass ID (see above)
        new MyNewAnalysis(), //the analysis itself
        DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on it
    );
    //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
    addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
}

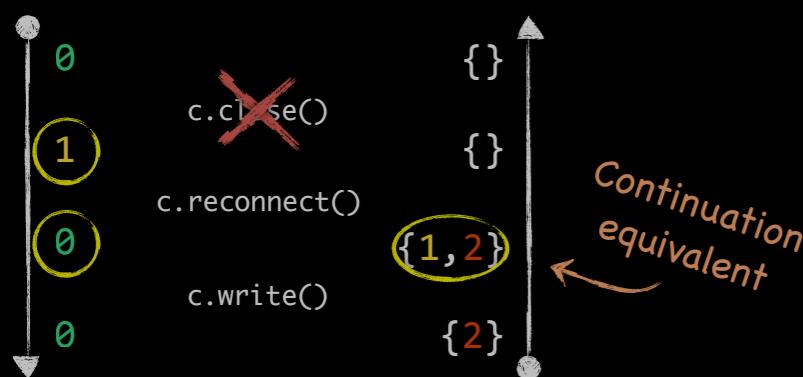
protected void collectVersions(StringBuffer versions) {
    //this String will appear when executing Clang with the -version parameter
    versions.append("MyAnalysis " + new Version().toString() + "\n on ");
    super.collectVersions(versions);
}

```



two sensible options:

$LR \subseteq L$ or $LR \supseteq L$



```

package clara.myanalysis;
import java.util.List;

/**
 * Create a new pass ID with the name <i>foo</i>. This will allow users
 * to schedule the analysis using the <tt>--static-analysis foo</tt> command-line
 * option.
 */
public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

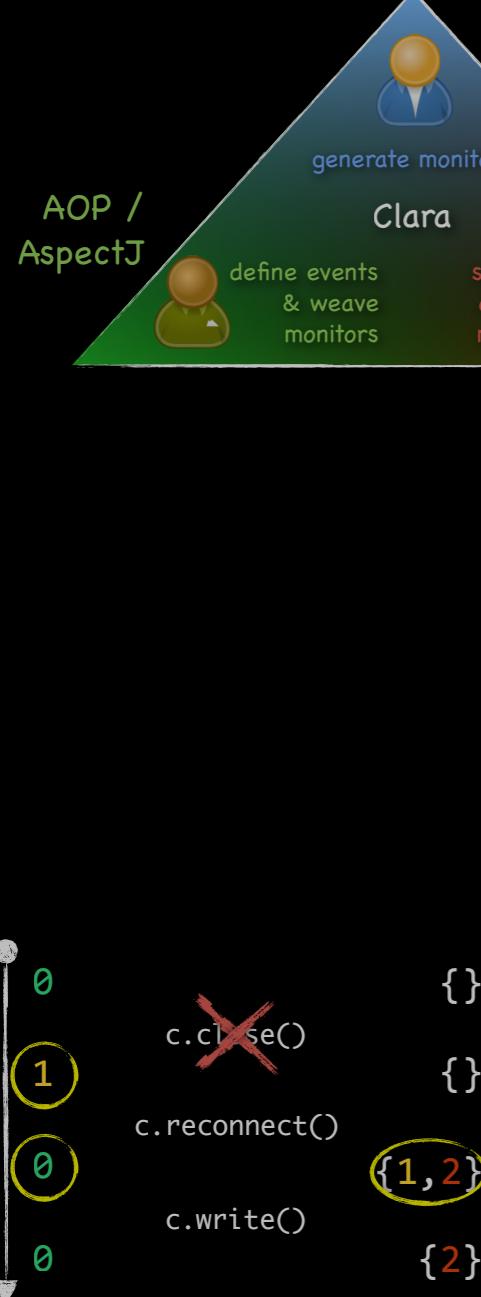
@Override
public void createReweavingPasses(List<ReweavingPass> passes) {
    super.createReweavingPasses(passes);
}

ReweavingPass myNewPass = new ReweavingPass(
    MY_PASS_ID, //the pass ID (see above)
    new MyNewAnalysis(), //the analysis itself
    DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on this analysis
);

//schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);

protected void collectVersions(StringBuffer versions) {
    //this String will appear when executing Clang with the -version parameter
    versions.append("MyAnalysis " + new Version().toString() + "\n on ");
    super.collectVersions(versions);
}

```



Java - clara-emptyext/src/clara/myanalysis/AbcExtension.java - Eclipse SDK

```

package clara.myanalysis;

import java.util.List;

/**
 * @author Eric Bodden
 */
public class AbcExtension extends ca.mcgill.sable.clara.AbcExtension
{
    /**
     * Create a new pass ID with the name <i>foo</i>. This will allow users
     * to schedule the analysis using the <tt>-static-analysis foo</tt> command-line
     * option.
     */
    public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

    @Override
    public void createReweavingPasses(List<ReweavingPass> passes) {
        super.createReweavingPasses(passes);

        ReweavingPass myNewPass = new ReweavingPass(
            MY_PASS_ID, //the pass ID (see above)
            new MyNewAnalysis(), //the analysis itself
            DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on this analysis
        );
        //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
        addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
    }

    protected void collectVersions(StringBuffer versions) {
        //this String will appear when executing Clara with the -version parameter
        versions.append("MyAnalysis " + new Version().toString() + "\n on ");
        super.collectVersions(versions);
    }
}

```

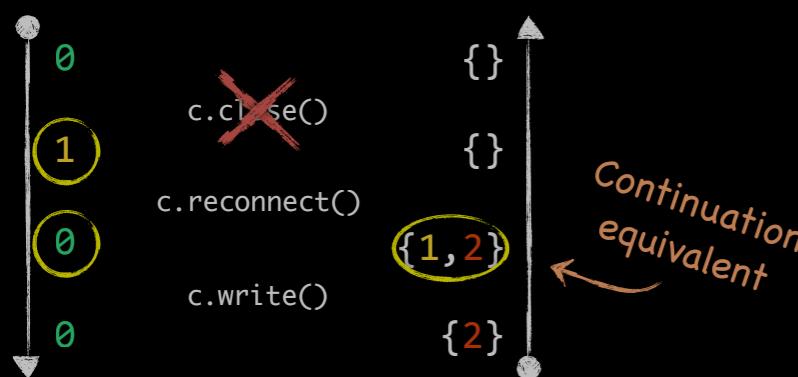
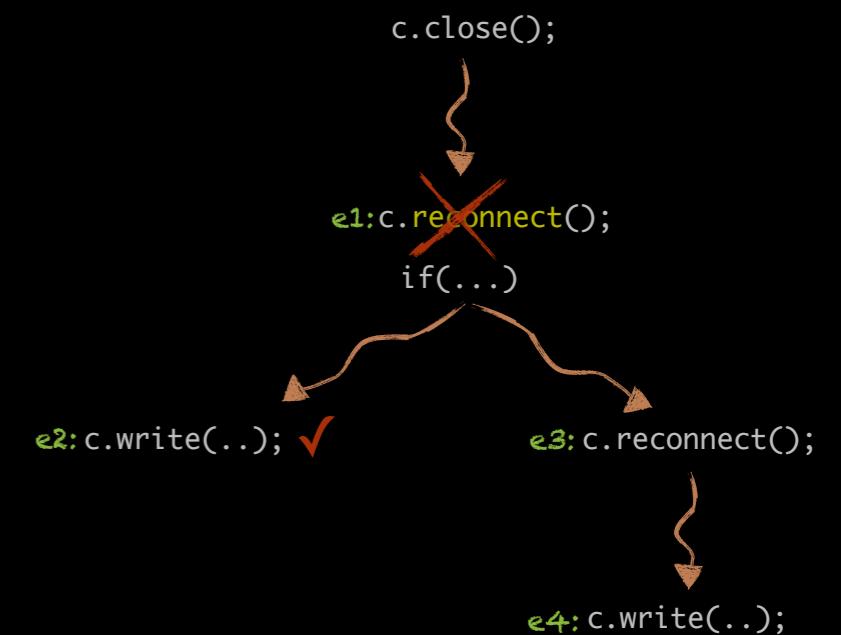
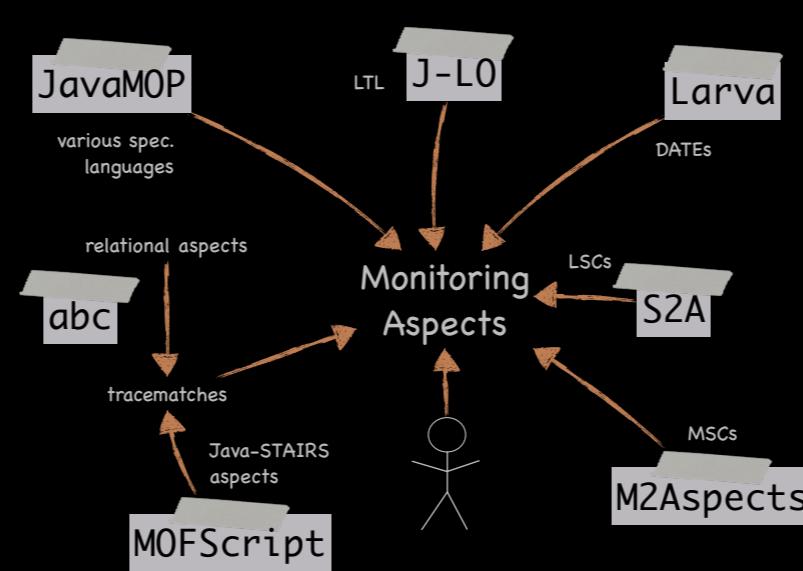
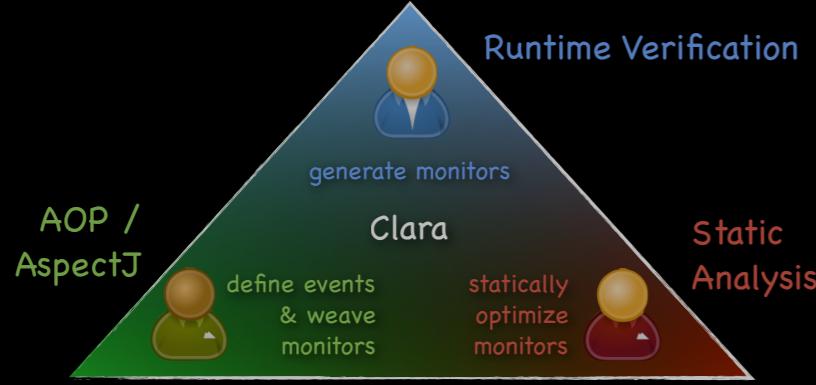
c.close();

~~c.reconnect();~~

if(...)

e3: c.reconnect();

e4: c.write(..);



two sensible options:
 $L_R \subseteq L$ or $L_R \supseteq L$

```

package clara.myanalysis;
import java.util.List;

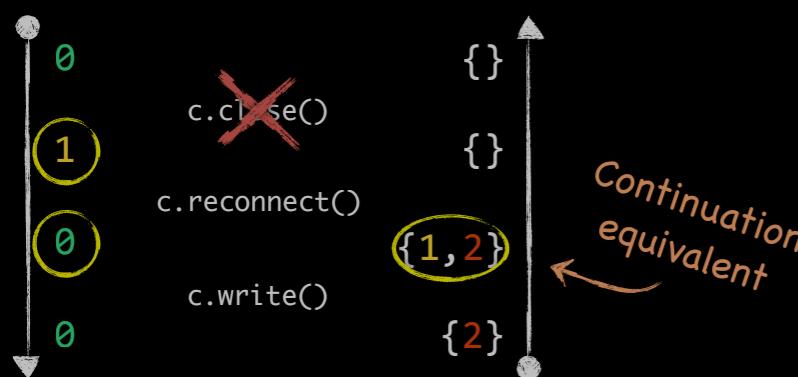
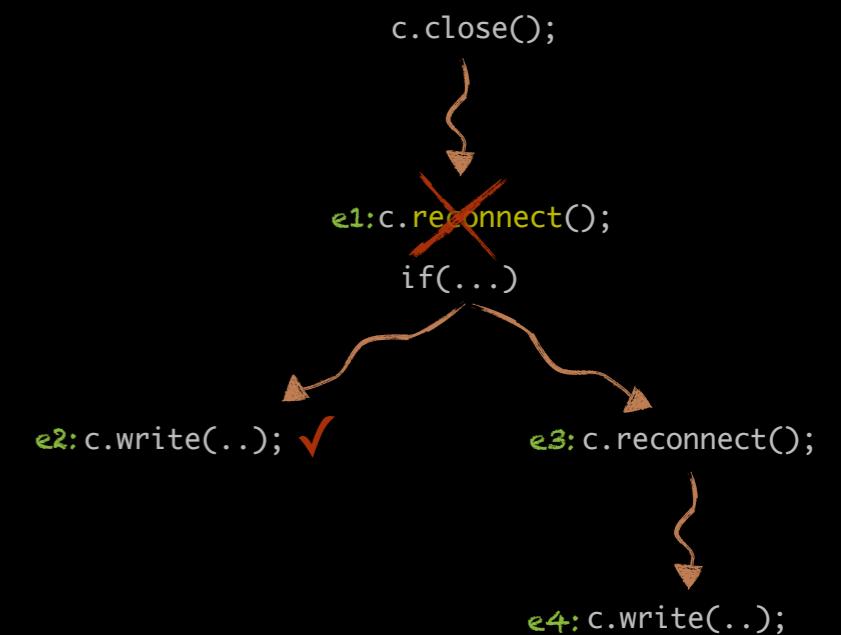
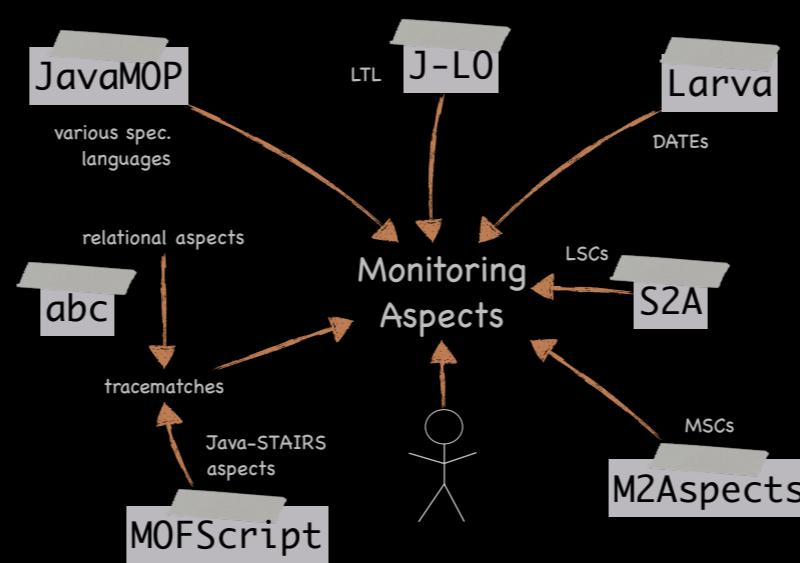
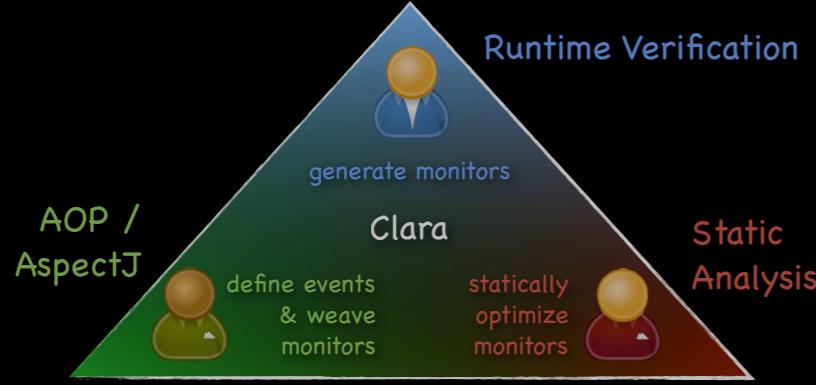
/**
 * Author: Eric Bodden
 */
public class AbcExtension extends ca.mcgill.sable.clara.AbcExtension
{
    /**
     * Create a new pass ID with the name <i>foo</i>. This will allow users
     * to schedule the analysis using the <tt>-static-analysis foo</tt> command-line
     * option.
     */
    public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

    @Override
    public void createReweavingPasses(List<ReweavingPass> passes)
    {
        super.createReweavingPasses(passes);

        ReweavingPass myNewPass = new ReweavingPass(
            MY_PASS_ID, //the pass ID (see above)
            new MyNewAnalysis(), //the analysis itself
            DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on this analysis
        );
        //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
        addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
    }

    protected void collectVersions(StringBuffer versions)
    {
        //this String will appear when executing Clang with the -version parameter
        versions.append("MyAnalysis " + new Version().toString() + "\n on ");
        super.collectVersions(versions);
    }
}

```



two sensible options:
 $L_R \subseteq L$ or $L_R \supseteq L$

```

package clara.myanalysis;
import java.util.List;

/**
 * Author: Eric Bodden
 */
public class AbcExtension extends ca.mcgill.sable.clara.AbcExtension
{
    /**
     * Create a new pass ID with the name <i>foo</i>. This will allow users
     * to schedule the analysis using the <tt>-static-analysis foo</tt> command-line
     * option.
     */
    public static final ID MY_PASS_ID = new ReweavingPass.ID("foo");

    @Override
    public void createReweavingPasses(List<ReweavingPass> passes)
    {
        super.createReweavingPasses(passes);
        ReweavingPass myNewPass = new ReweavingPass(
            MY_PASS_ID, //the pass ID (see above)
            new MyNewAnalysis(), //the analysis itself
            DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS //declare that the pass MY_PASS_ID depends on it
        );
        //schedule the pass to be run just after DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS
        addAfterPass(passes, myNewPass, DEPENDENT_ADVICE_FLOW_INSENSITIVE_ANALYSIS);
    }

    protected void collectVersions(StringBuffer versions)
    {
        //this String will appear when executing Clang with the -version parameter
        versions.append("MyAnalysis " + new Version().toString() + "\n on ");
        super.collectVersions(versions);
    }
}

```

www.bodden.de/clara/

