Continuation Equivalence

A Correctness Criterion for Static Optimizations of Dynamic Analyses

Eric Bodden

CASED

TECHNISCHE UNIVERSITÄT DARMSTADT

Dynamic Program Analysis

```
public class ClaraTest {

    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }

}
```

Analysis Configuration

Dynamic Program Analysis

```
public class ClaraTest {

    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }

}
```

Analysis Configuration

Instrumentor

weave

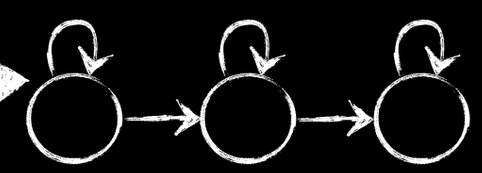
Dynamic Program Analysis

```
public class ClaraTest {

    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }

}
```

Analysis Configuration

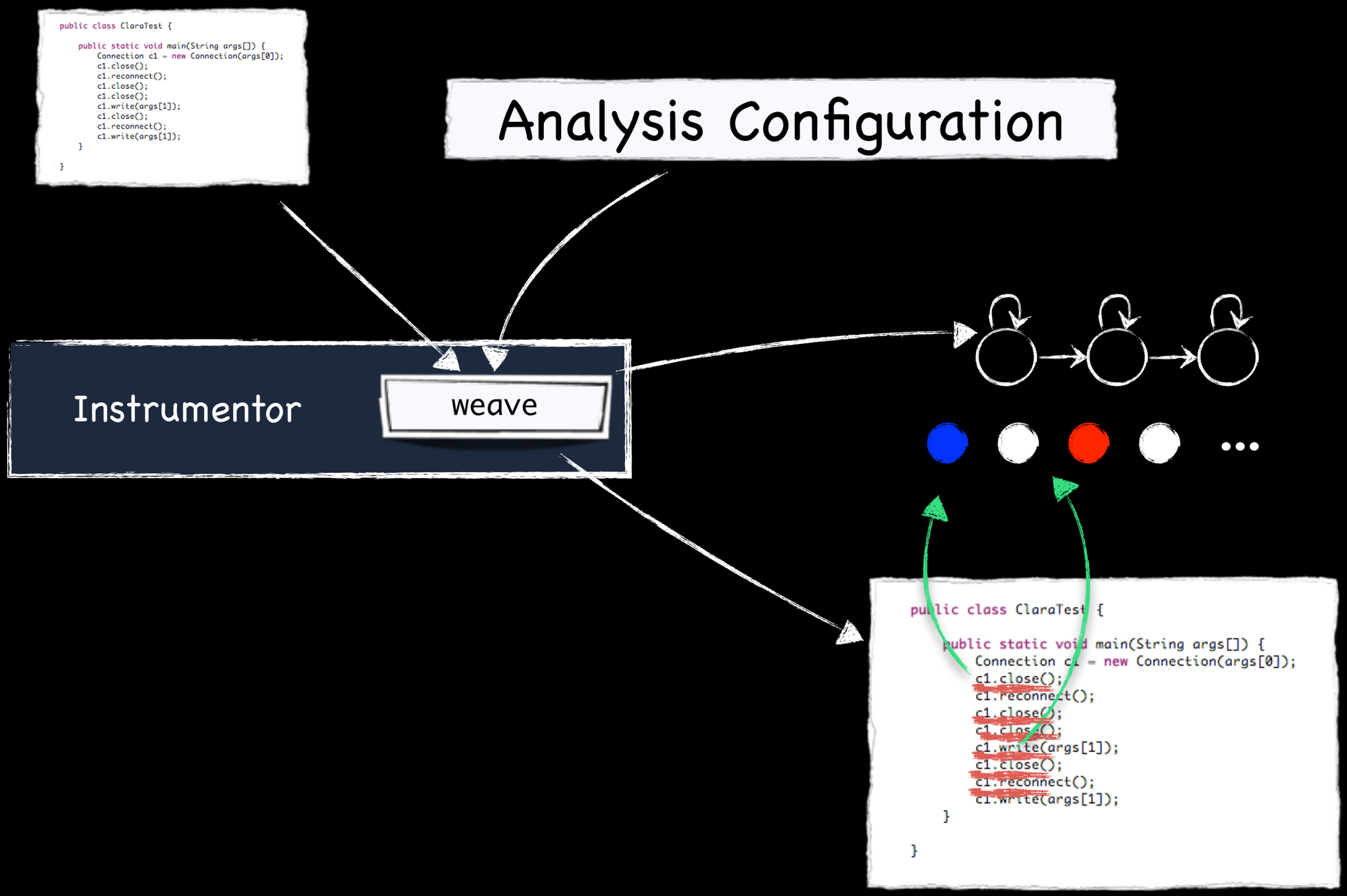
Instrumentor

weave



```
public class ClaraTest {

    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }

}
```

Dynamic Program Analysis



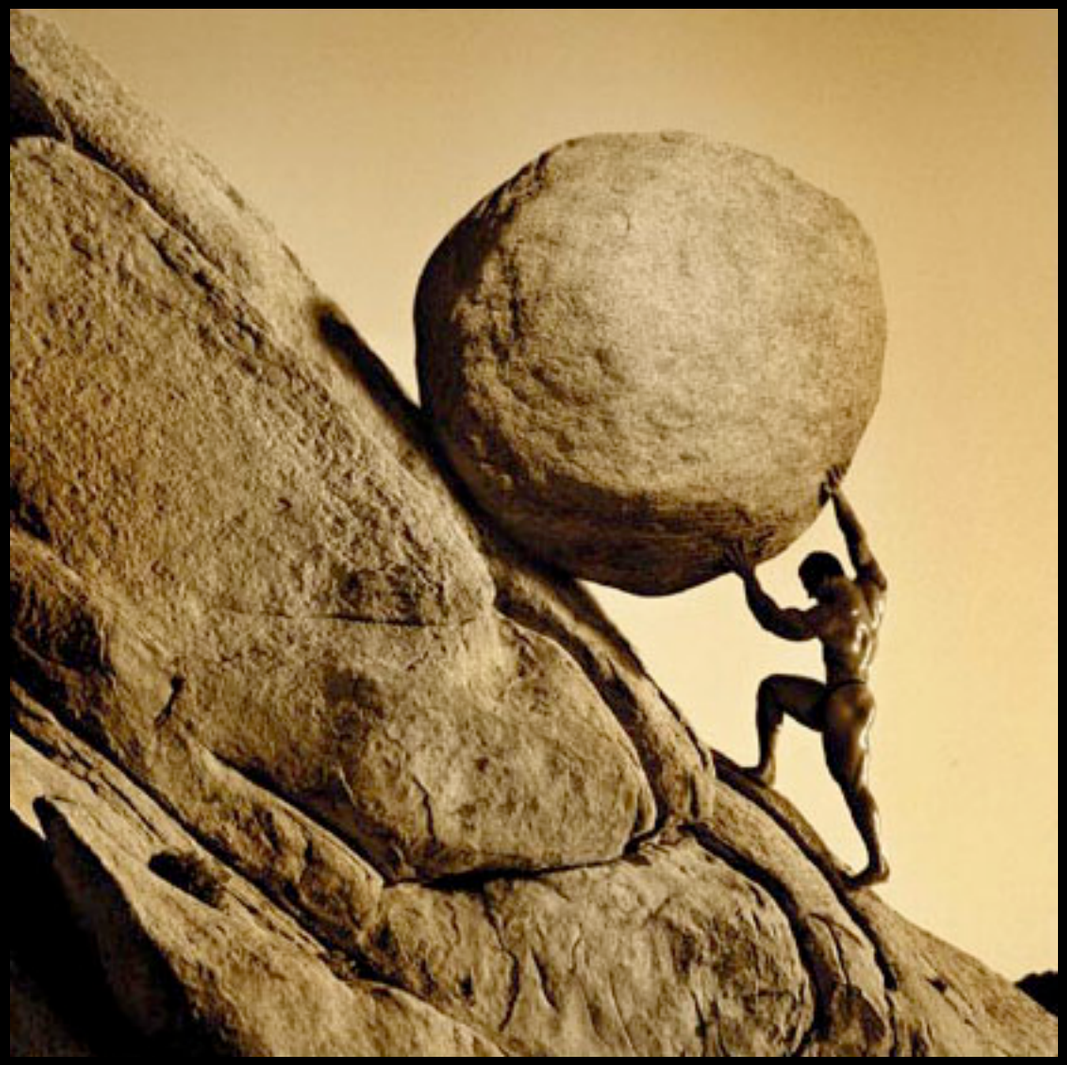
Problems of Runtime Analysis

No static guarantees

Problems of Runtime Analysis

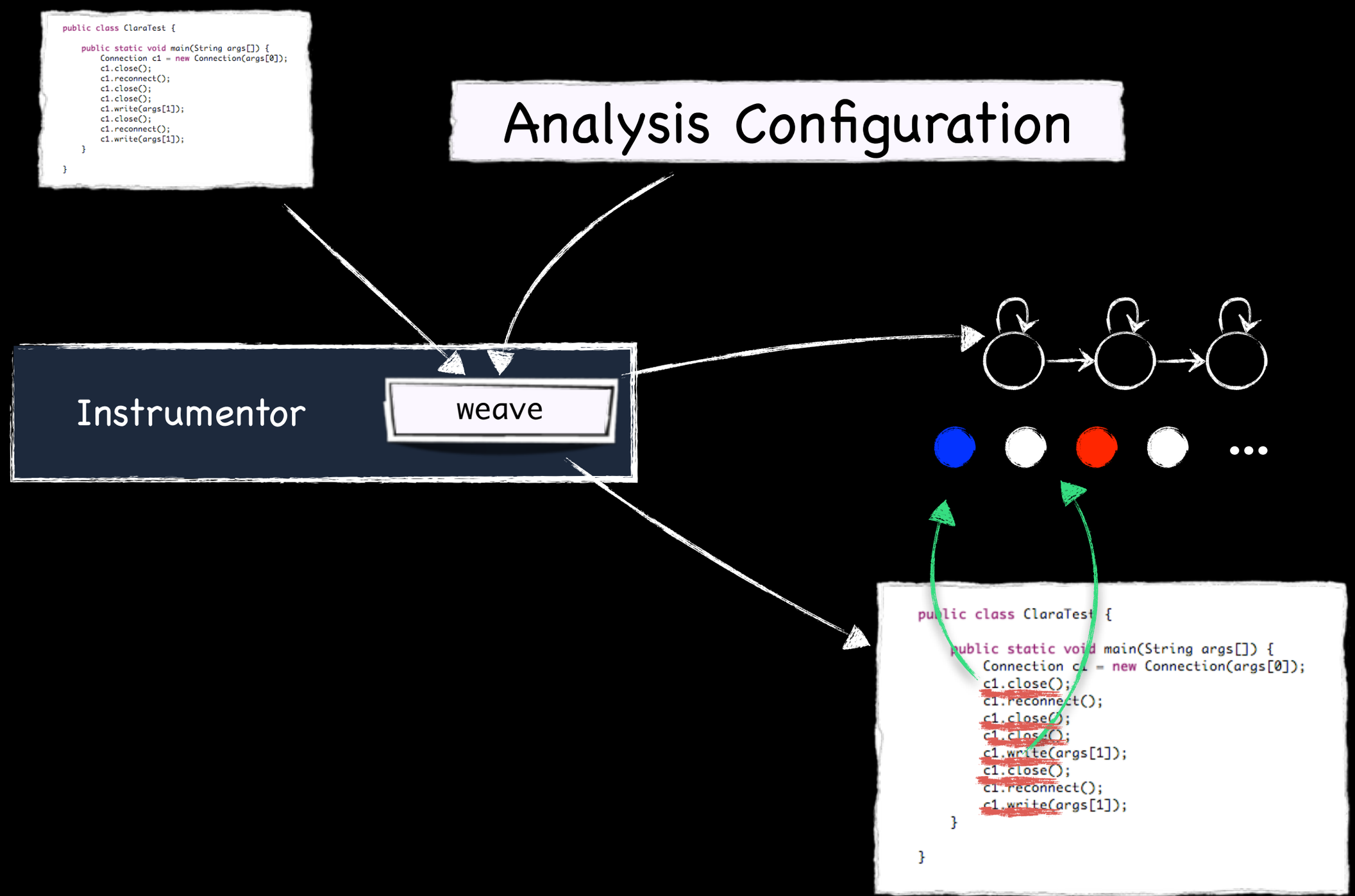
Potentially large runtime overhead

Problems of Runtime Analysis

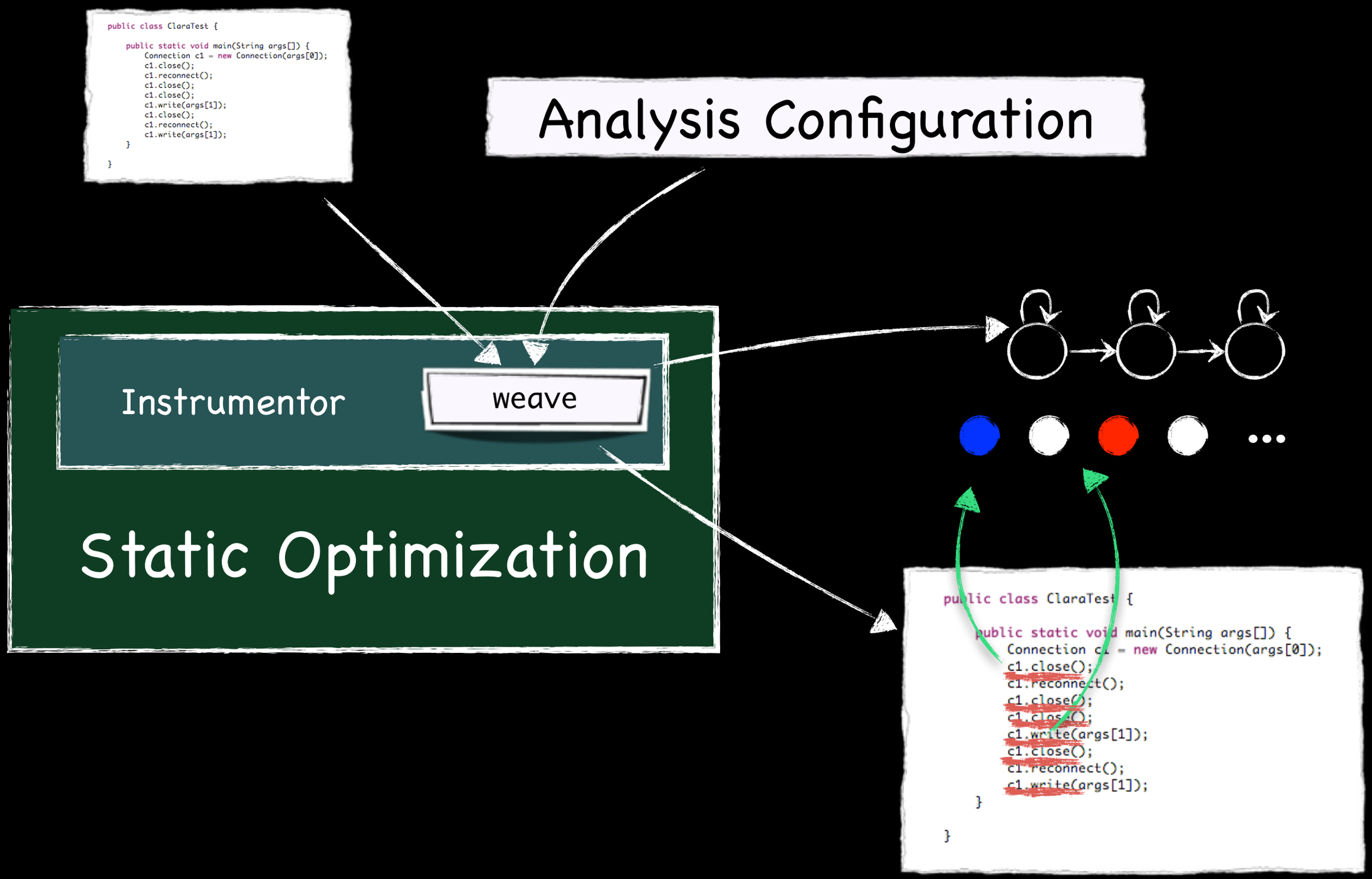


When analyzed enough?

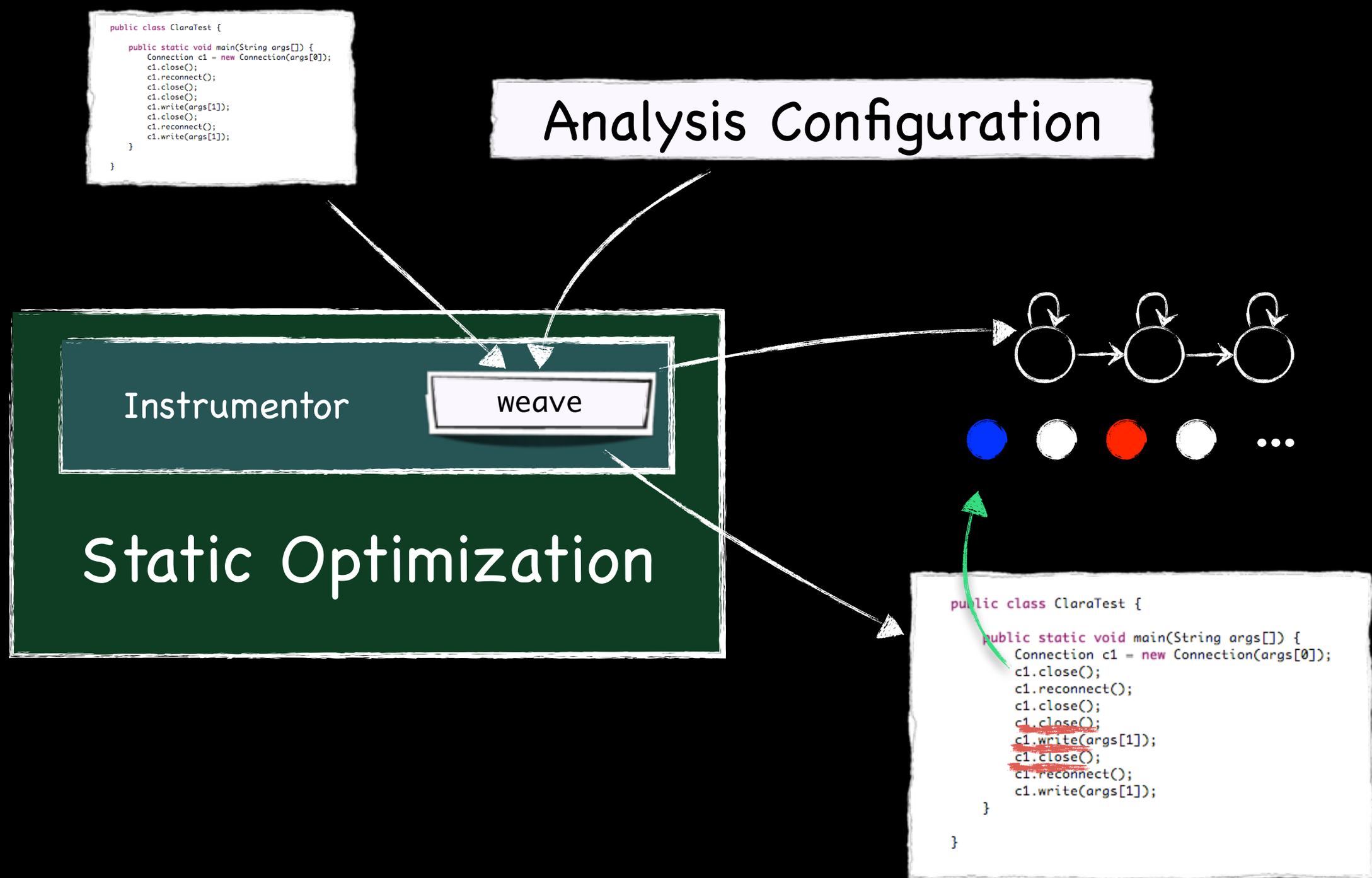
Ahead-of-time evaluation



Ahead-of-time evaluation



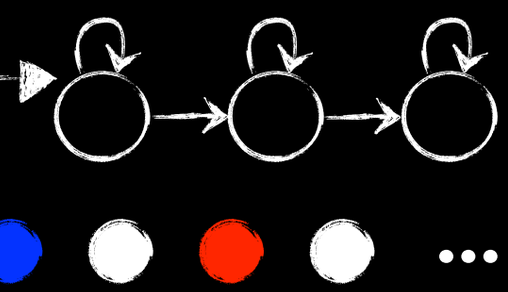
Ahead-of-time evaluation



Ahead-of-time evaluation

```
public class ClaraTest {

    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }

}
```

Analysis Configuration



```
public class ClaraTest {

    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }

}
```

- **Type checking of C Programs**
  Yong & Horwitz, FMSD `05
- **Points-to Analysis**
  Gutzmann & Löwe, WODA `11
- **Data-Race Detection**
  Bodden & Havelund, ISSTA 2008

Finite-state runtime monitoring / Typestate

Fink et al., ISSTA 2006

Bodden, Hendren & Lhotak, ECOOP 2007

Naeem & Lhotak, OOPSLA 2008

Bodden, Lam & Hendren, FSE 2008

Finite-state runtime monitoring / Typestate

Fink et al., ISSTA 2006

Bodden, Hendren & Lhotak, ECOOP 2007

Naeem & Lhotak, OOPSLA 2008

Bodden, Lam & Hendren, FSE 2008

Finite-state runtime monitoring / Typestate

Fink et al., ISSTA 2006

Bodden, Hendren & Lhotak, ECOOP 2007

Naeem & Lhotak, OOPSLA 2008

Bodden, Lam & Hendren, FSE 2008

Bodden, ICSE 2010

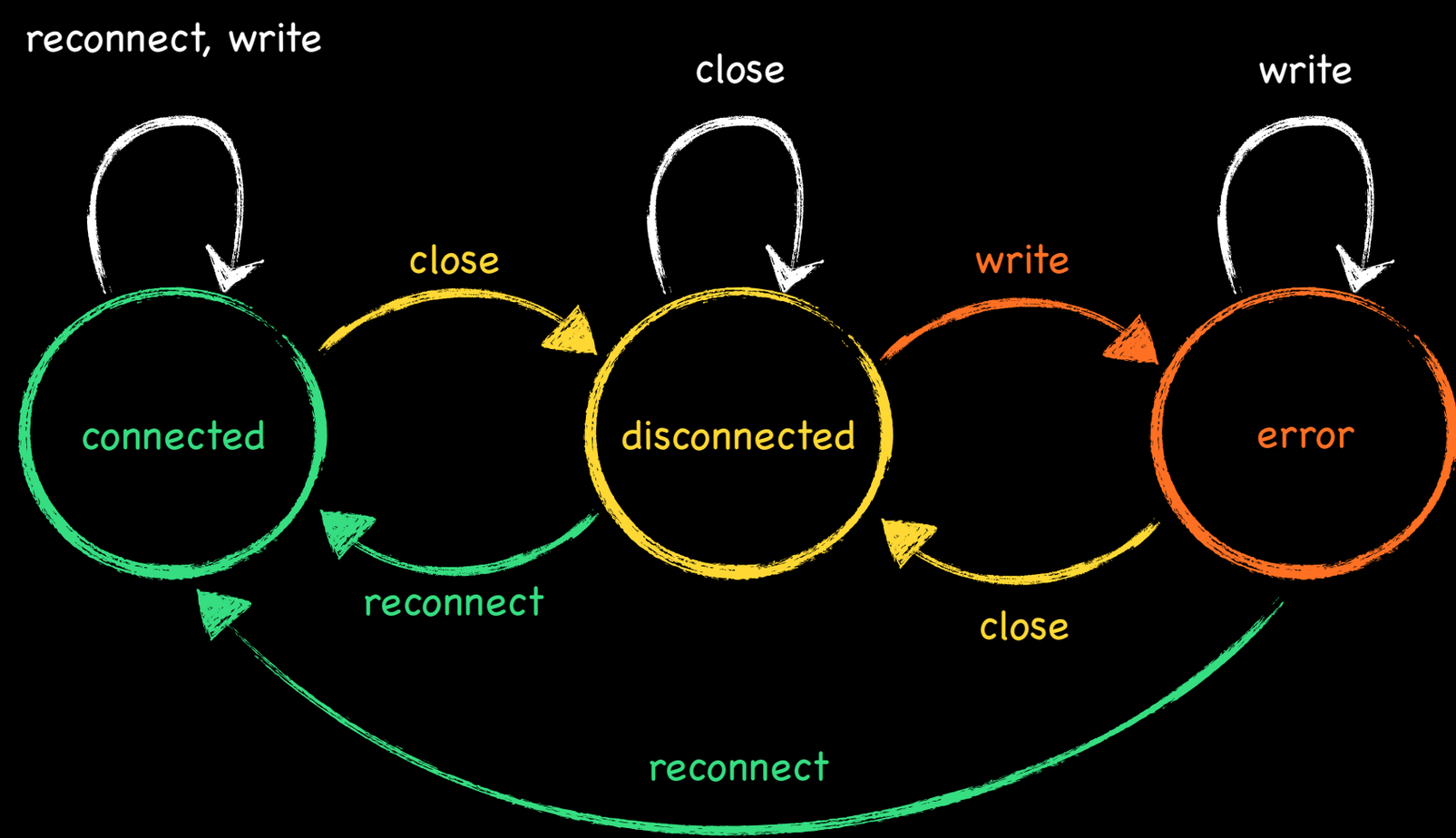
CLARA

http://bodden.de/clara/

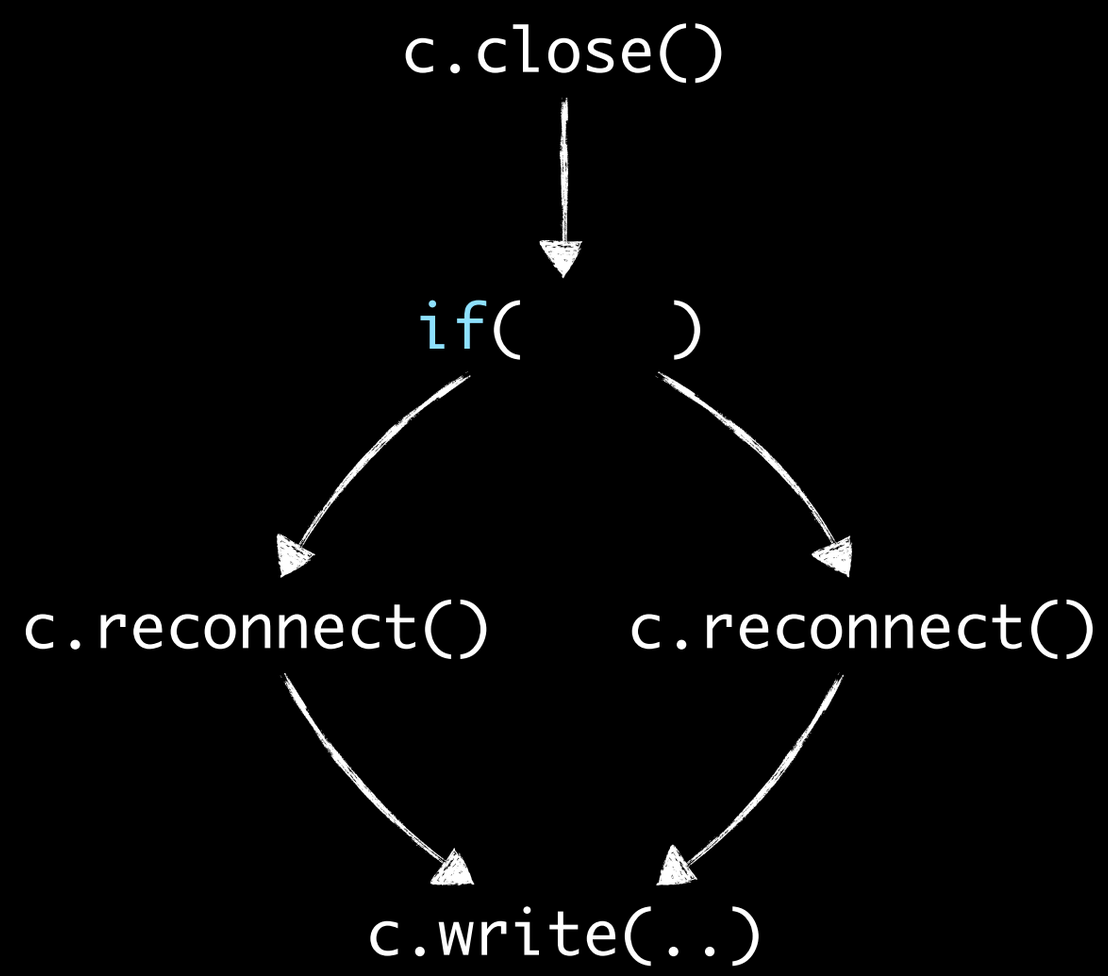
Finite-state property

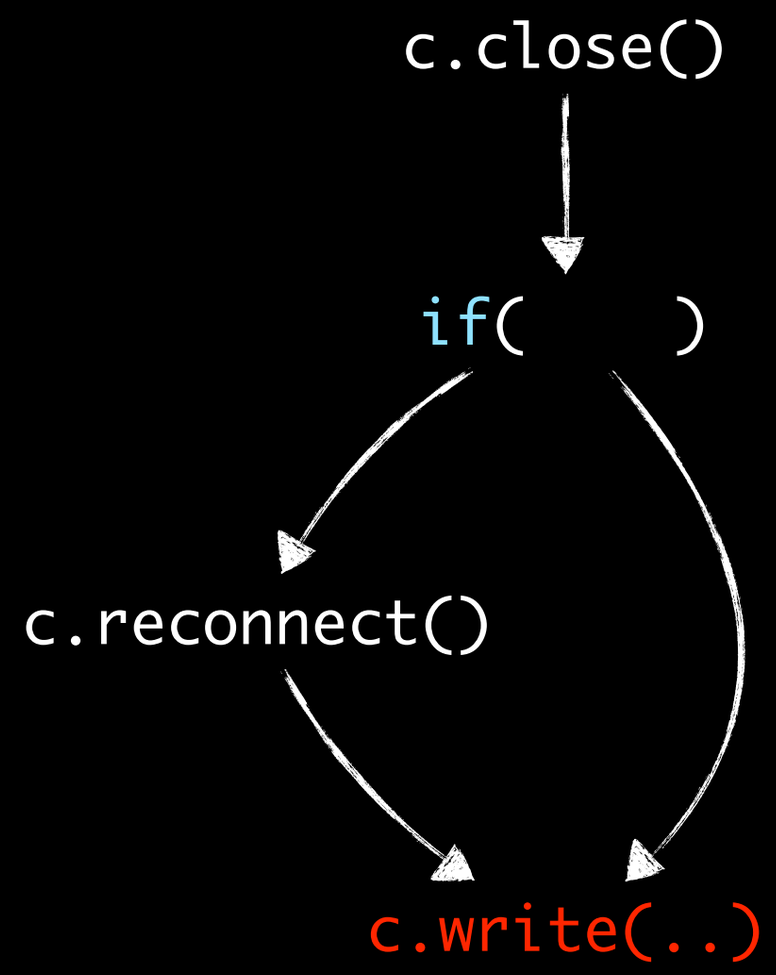
"After closing a connection c,
don't write to c until c is reconnected."

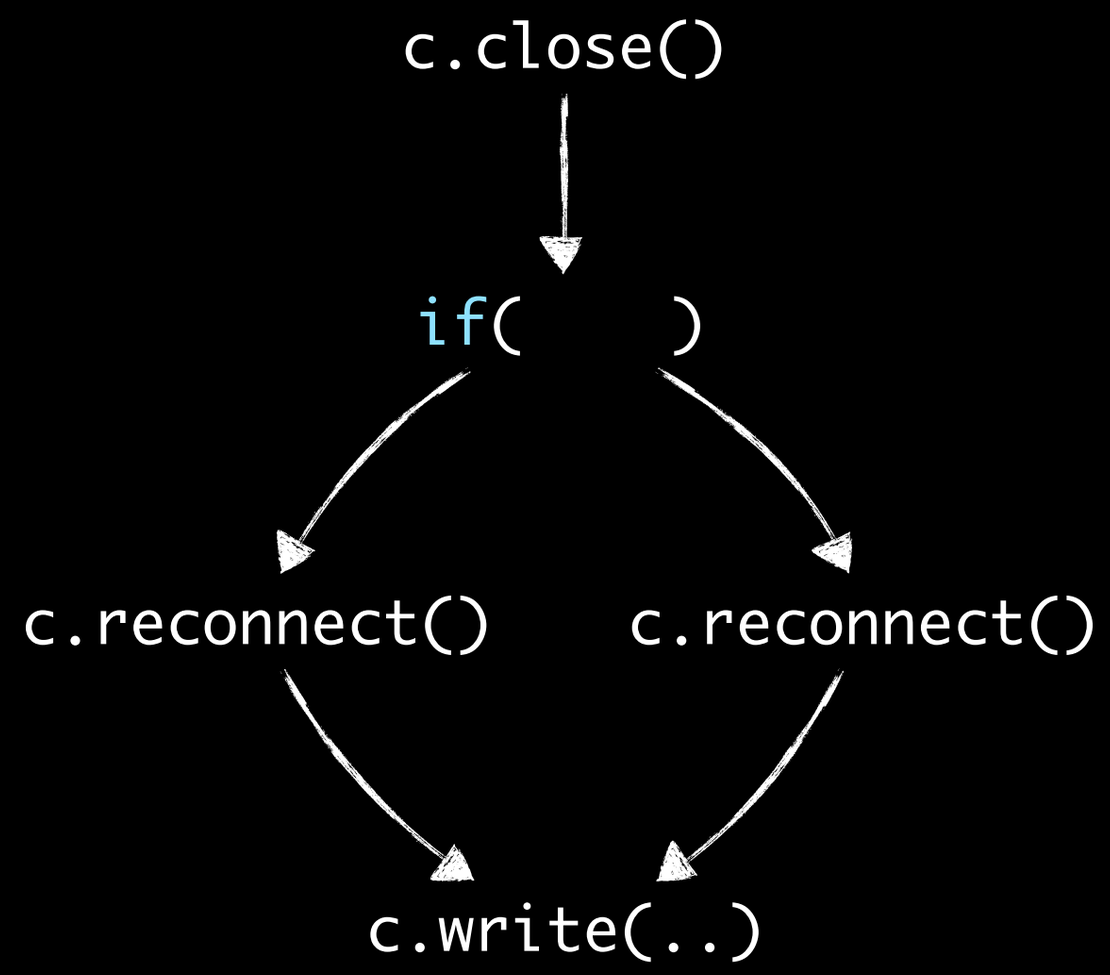
Finite-state property

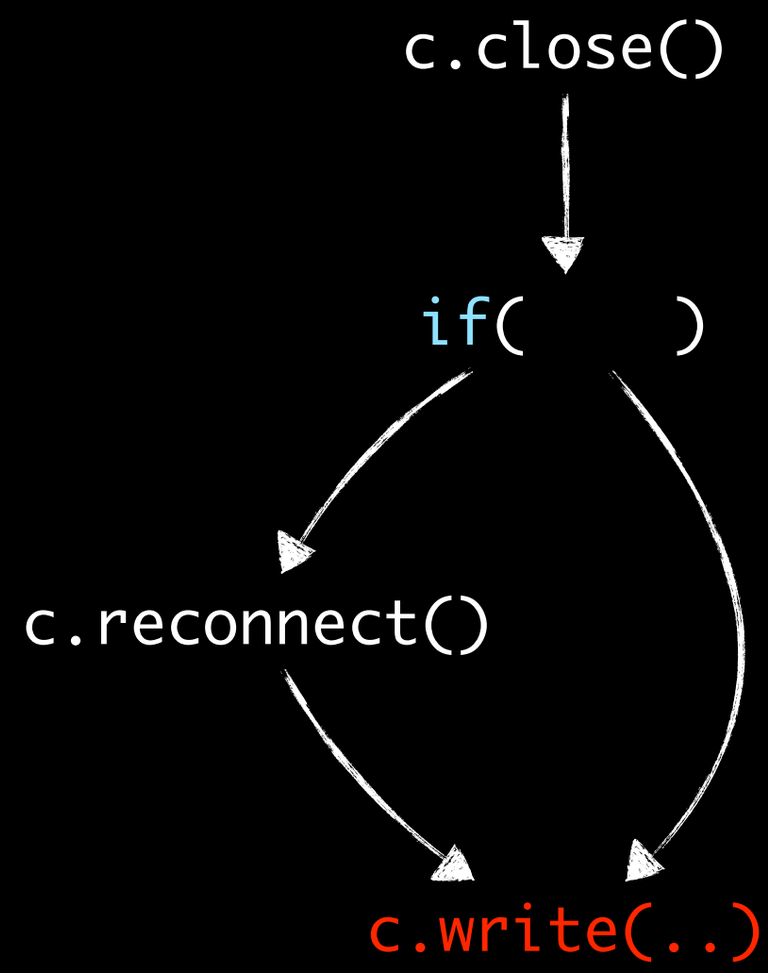


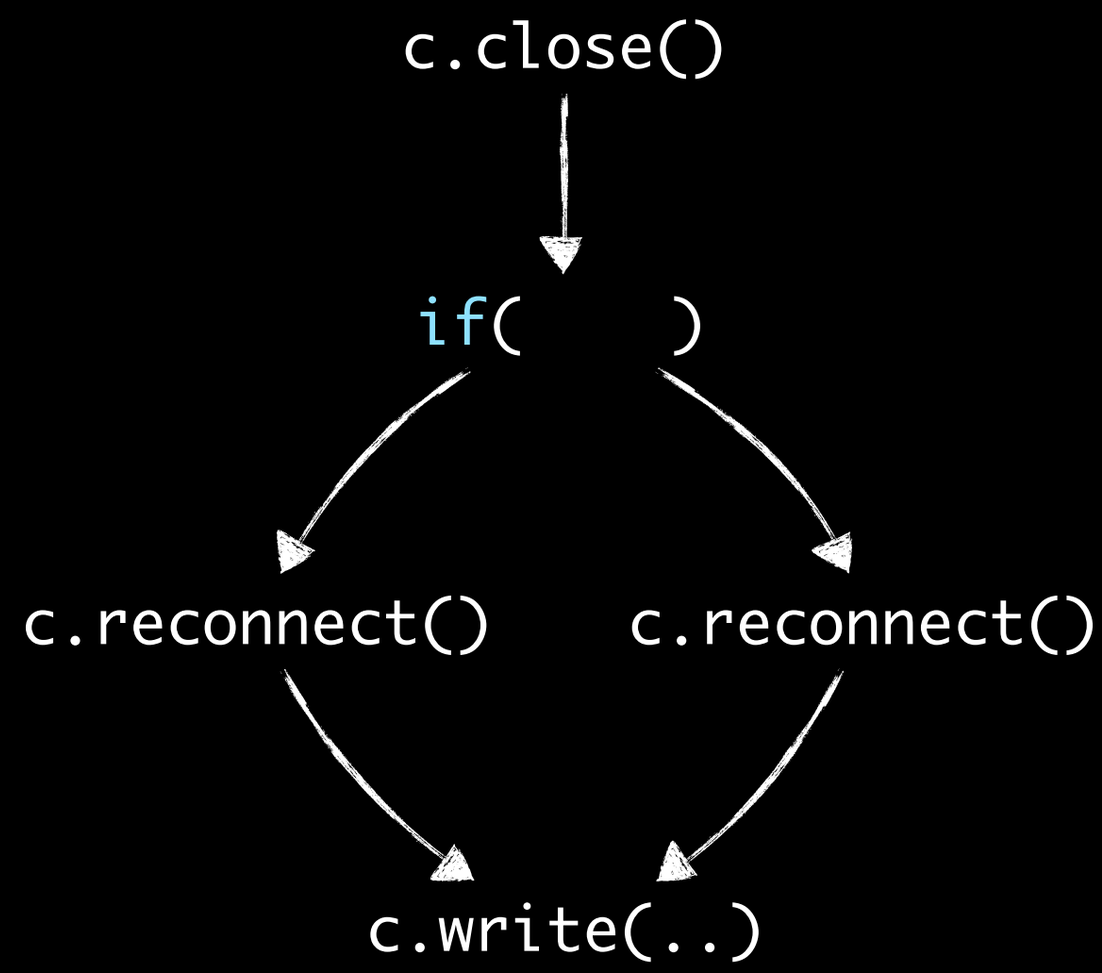
"After closing a connection c,
don't write to c until c is reconnected."


c.close()
if( )
c.reconnect()
c.reconnect()
c.write(..)

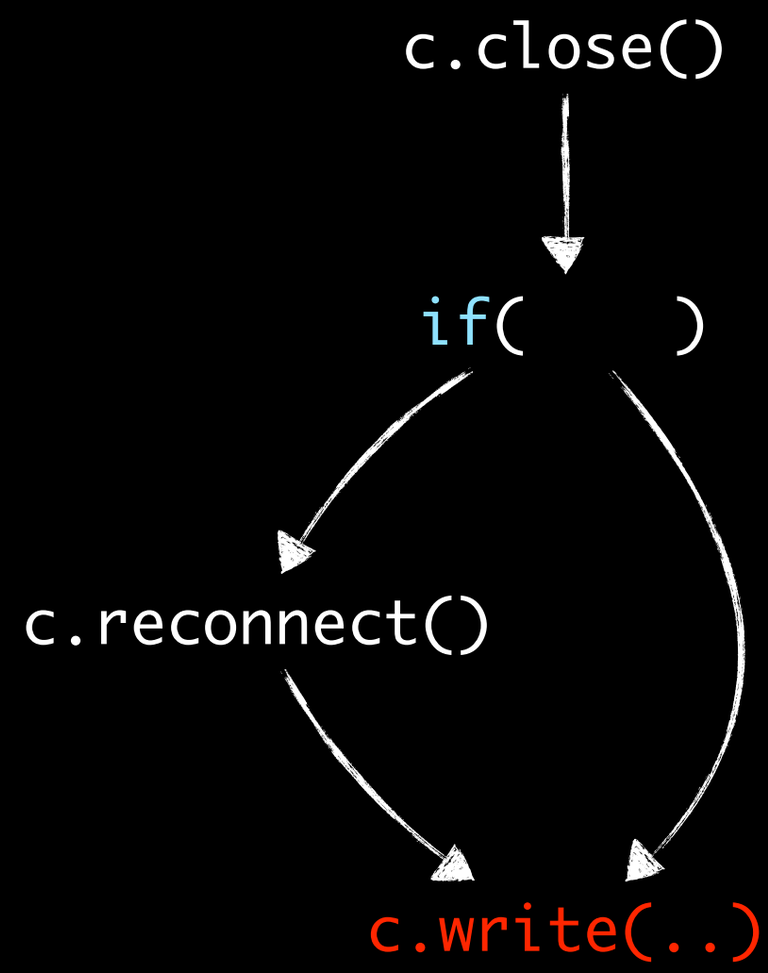

c.close()
if( )
c.reconnect()
c.write(..)

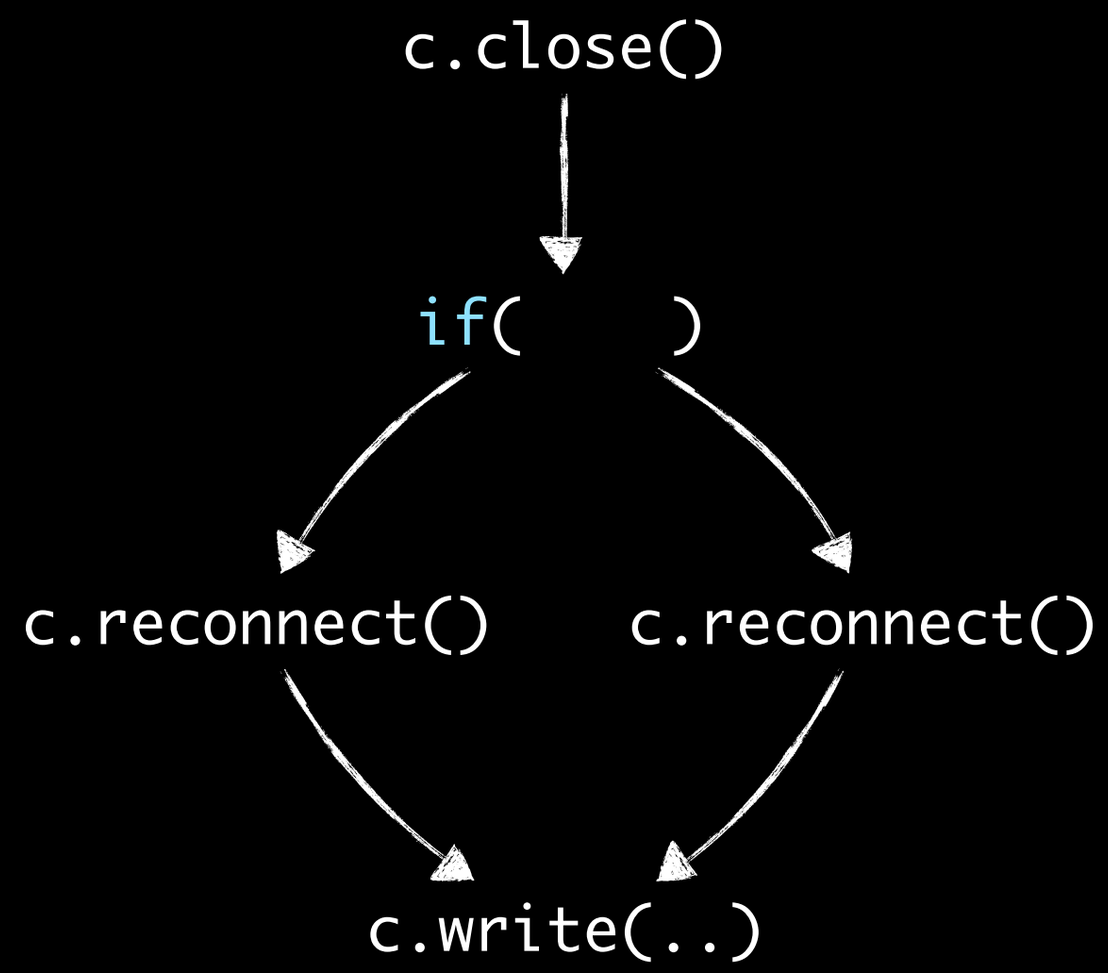

c.close()
if( )
c.reconnect()
c.reconnect()
c.write(..)



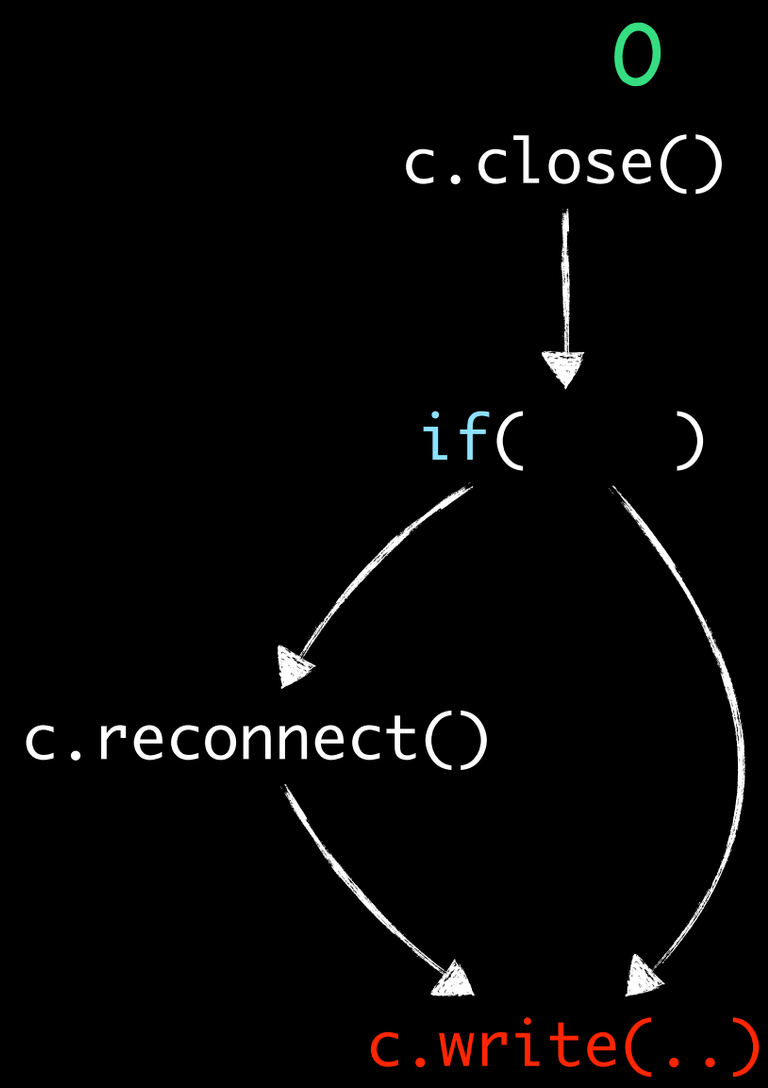


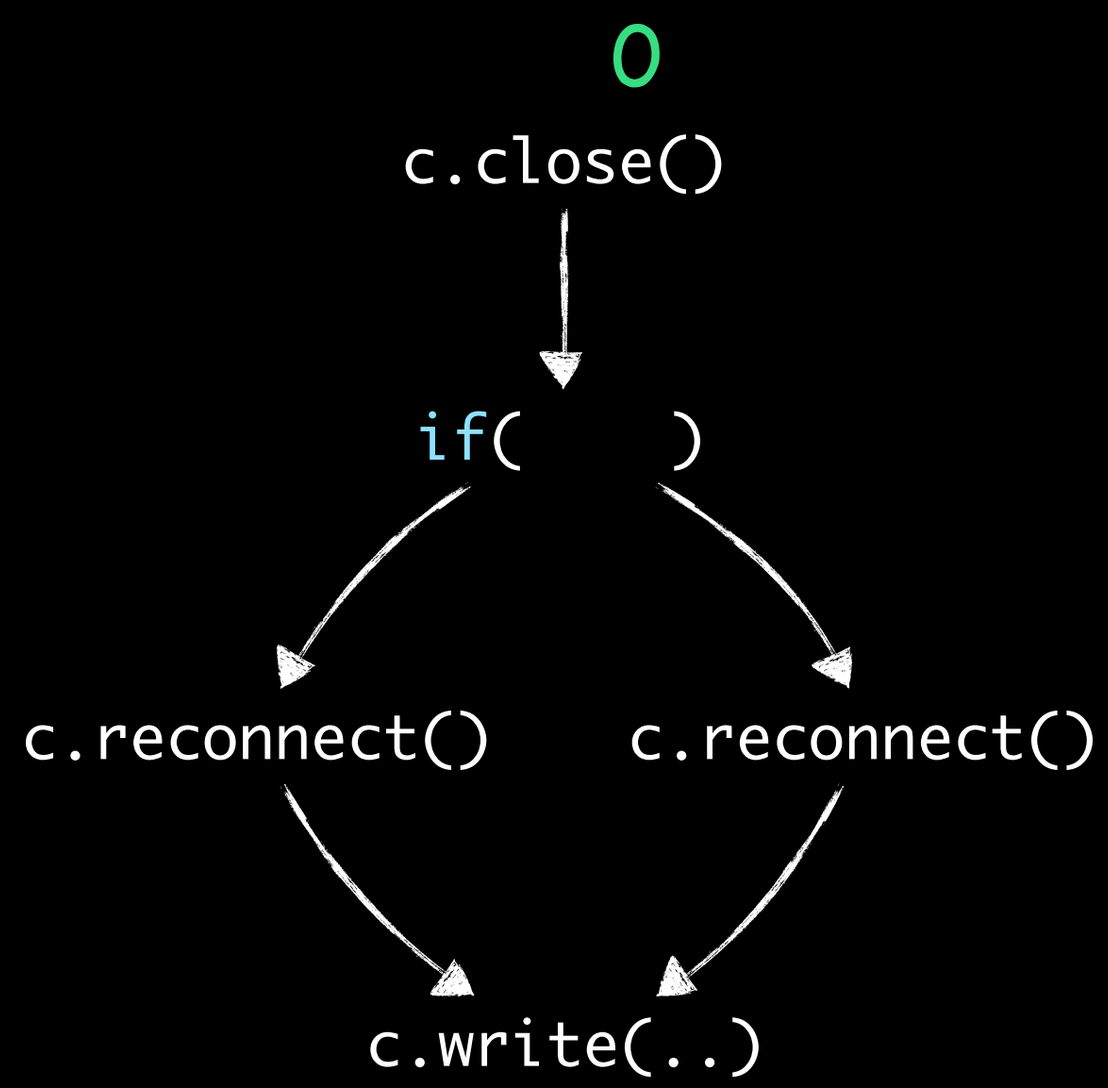
Forward or Backward?



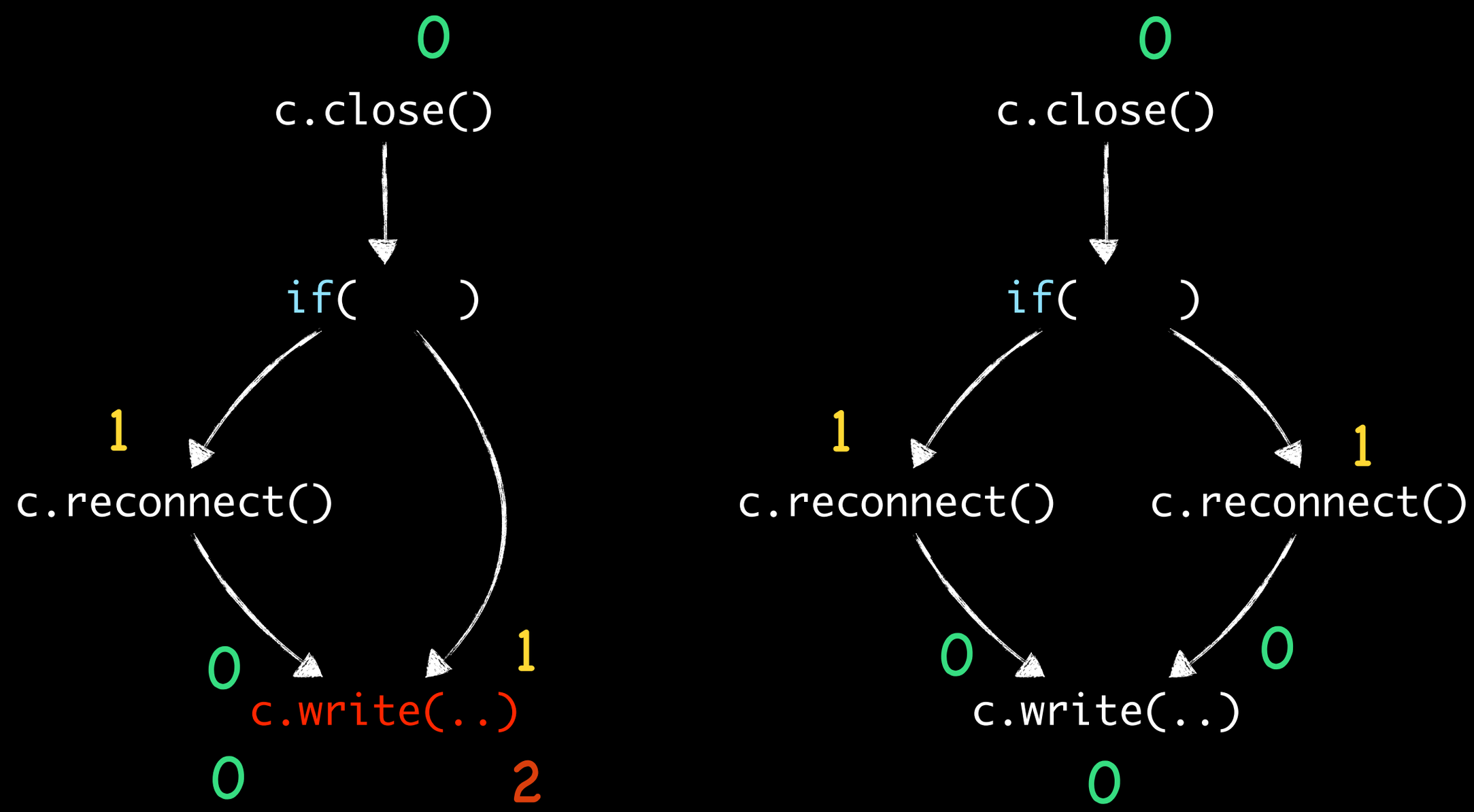


Forward ~~or Backward?~~



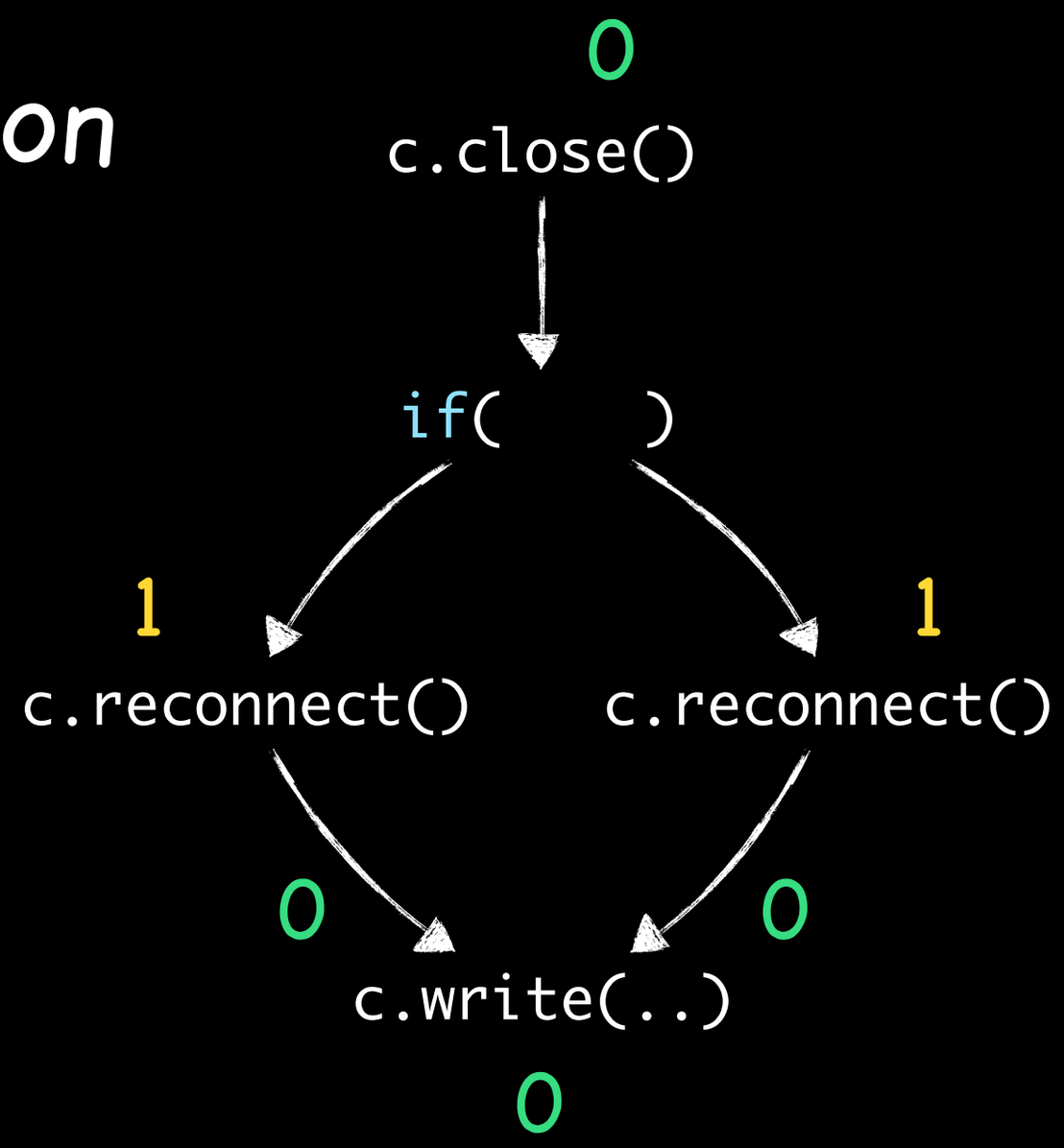


Forward ~~or Backward?~~

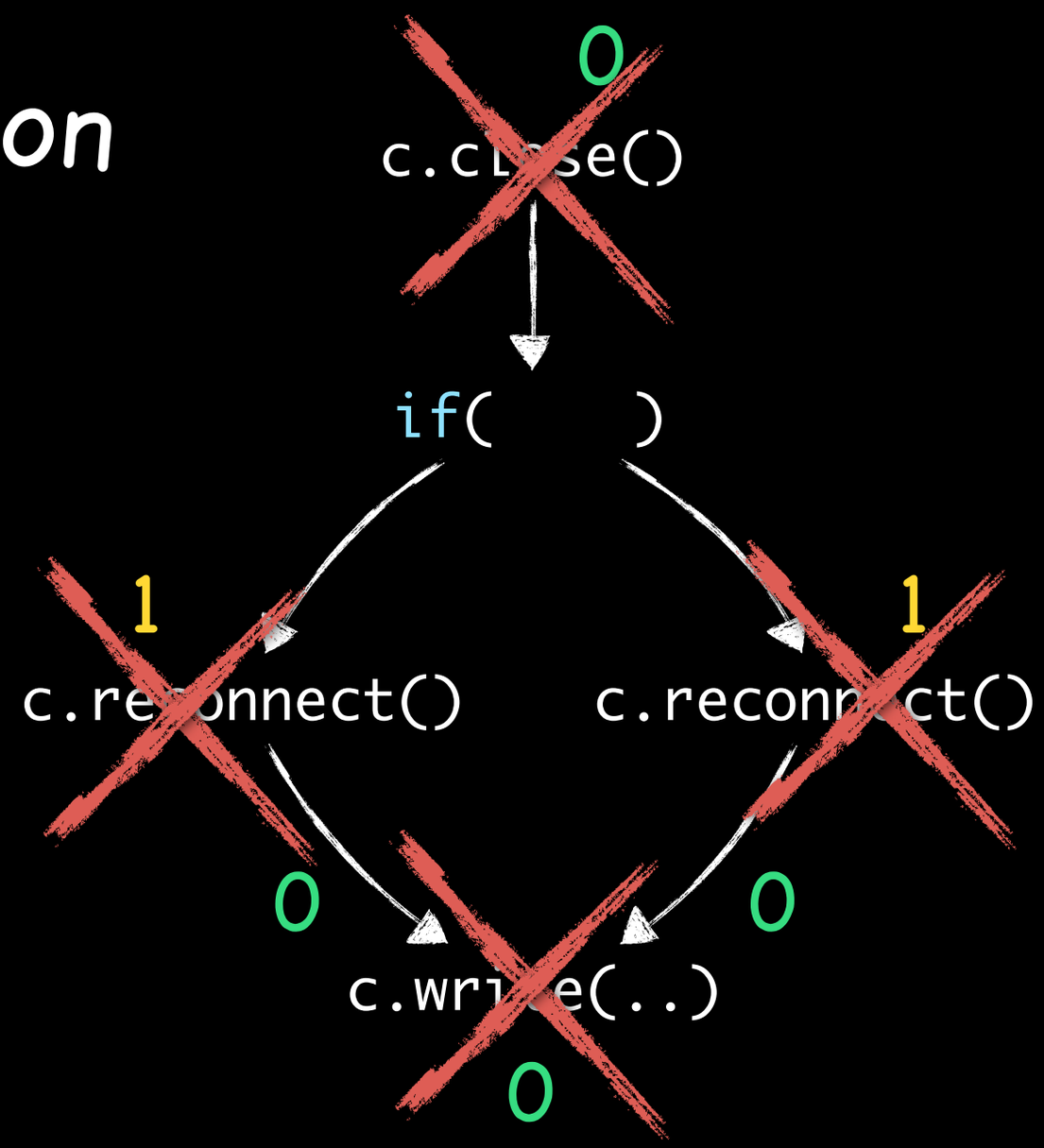


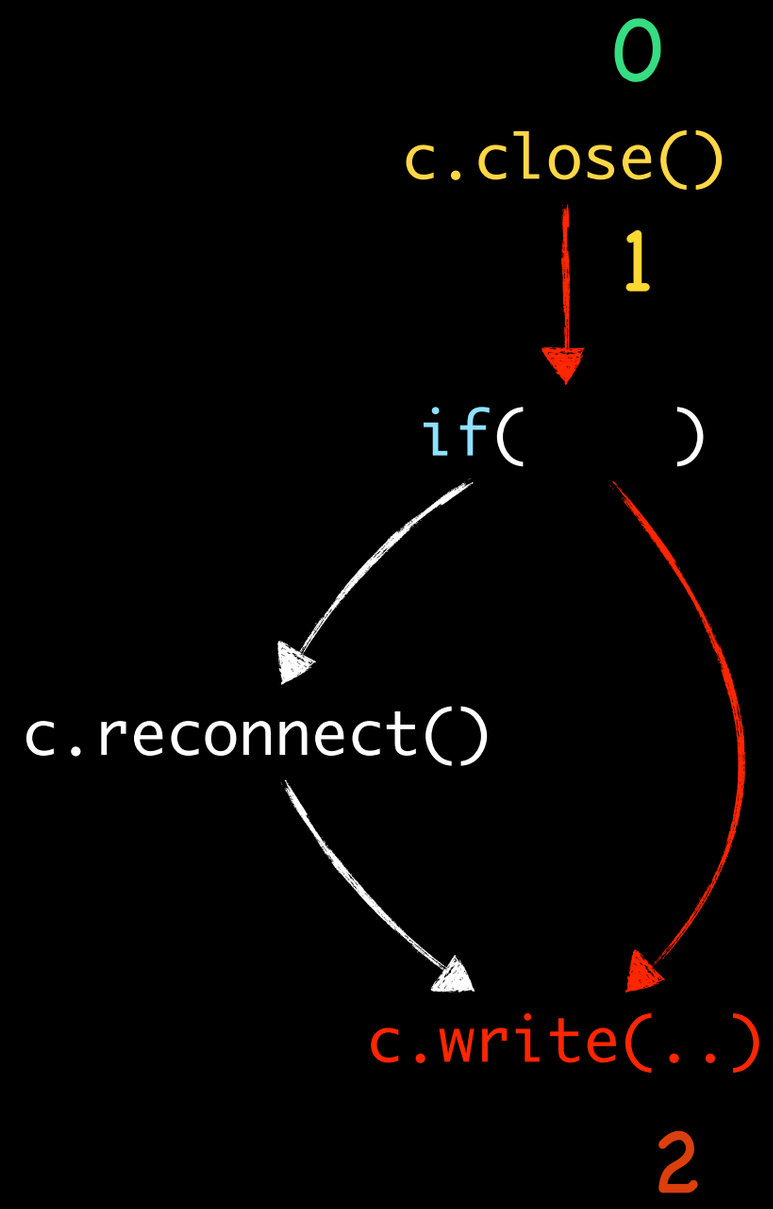
Forward ~~or Backward?~~

Which instrumentation to remove?

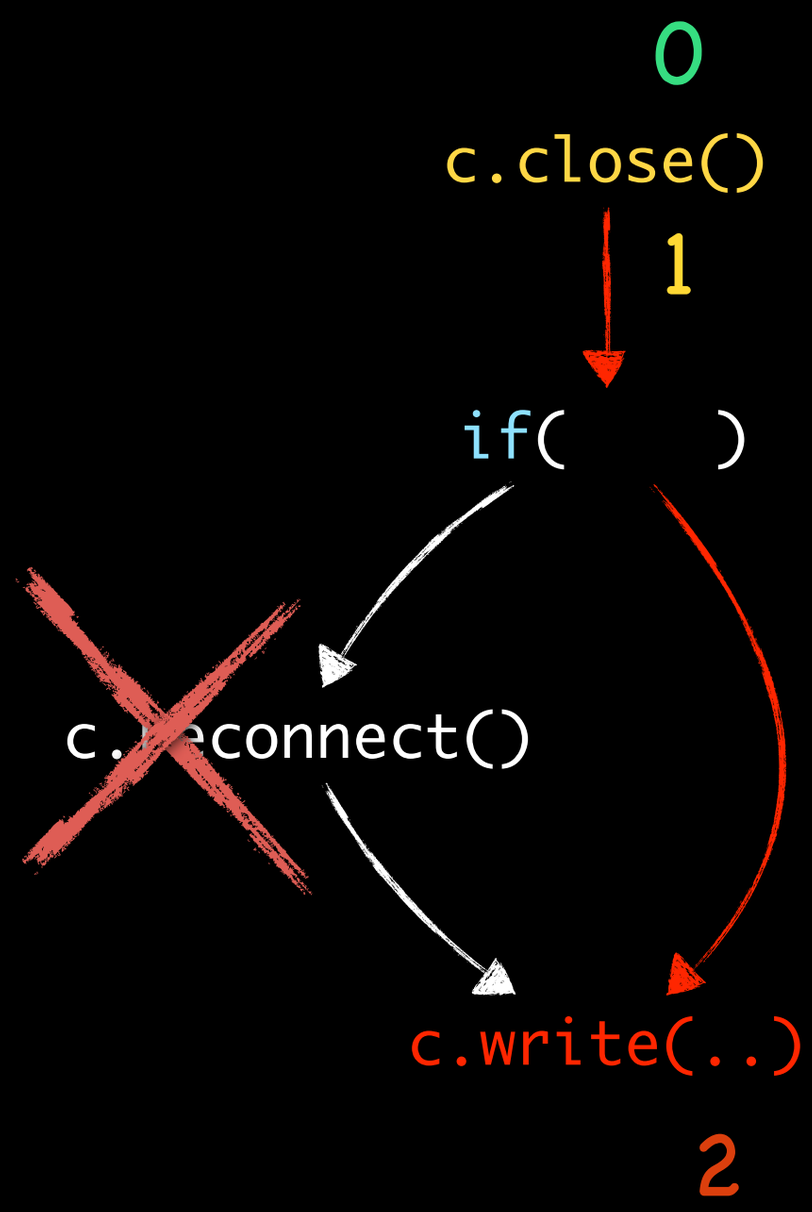


Which instrumentation to remove?

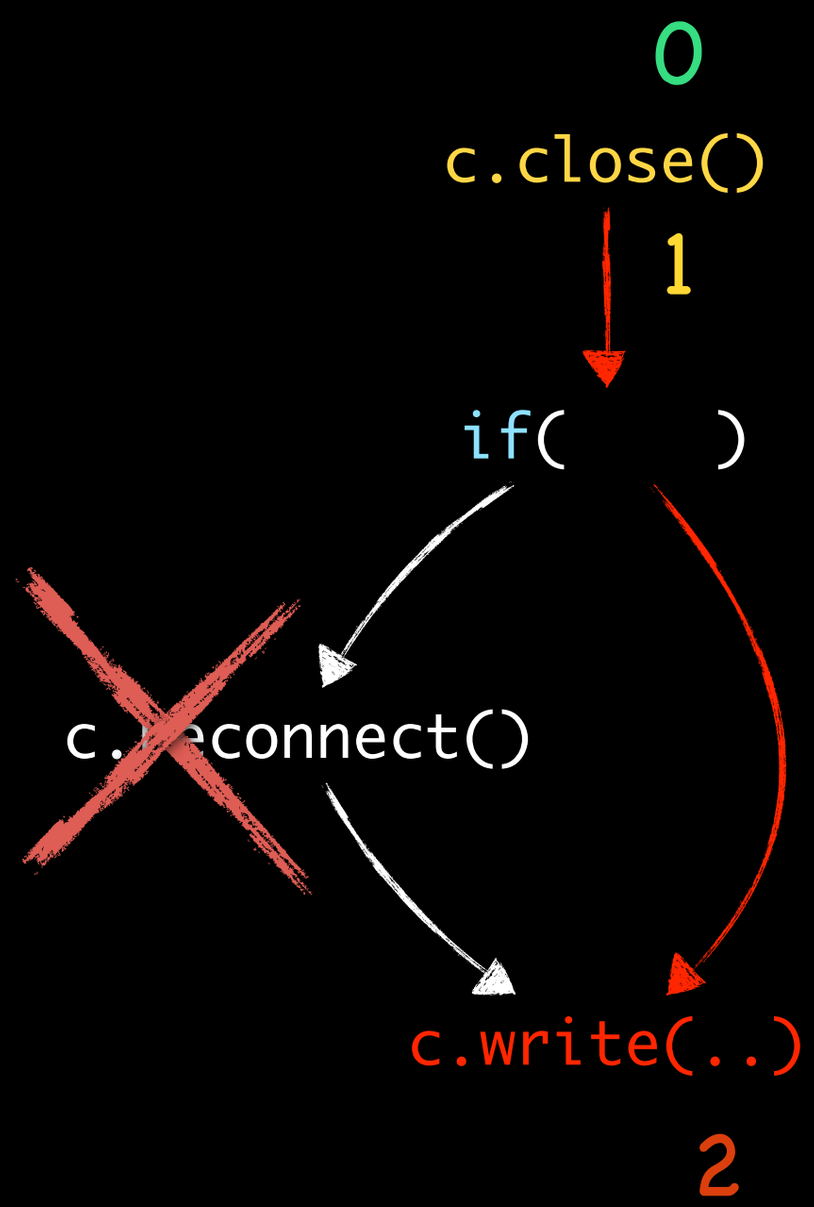




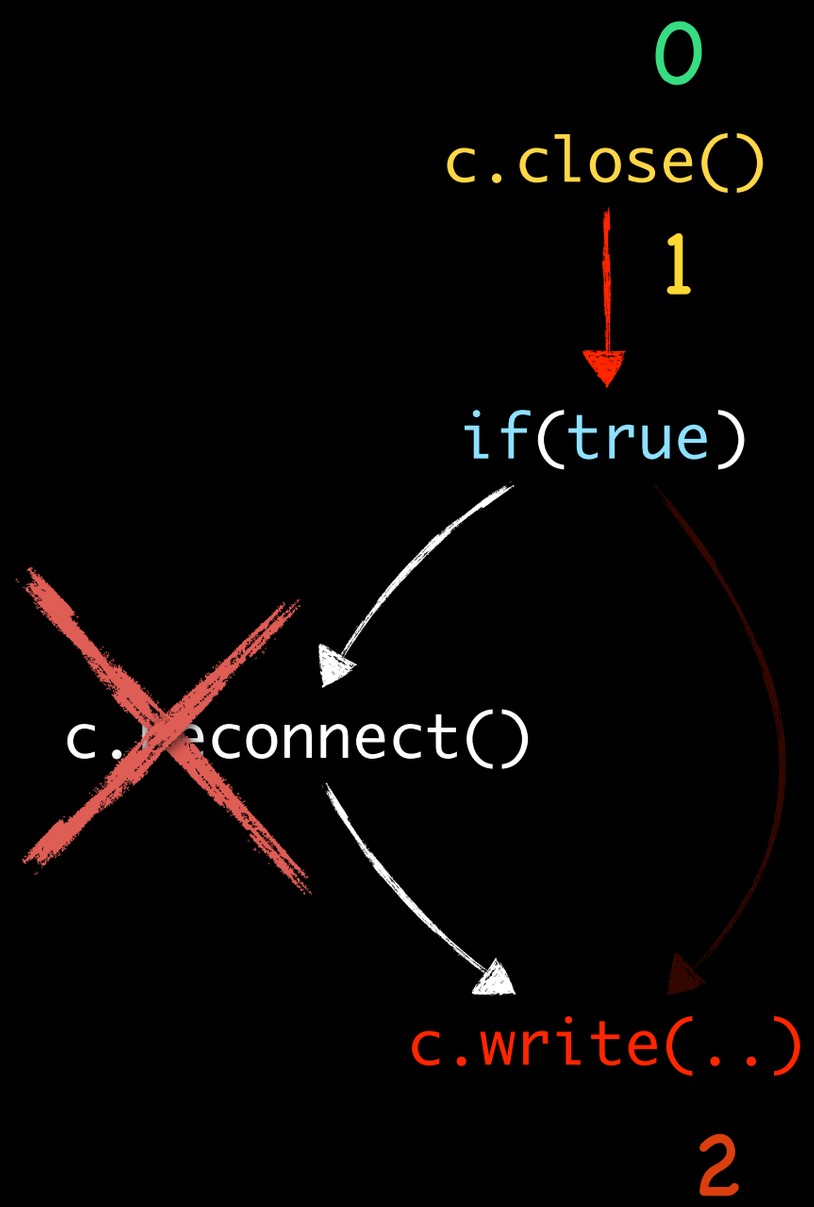
Which instrumentation to remove?



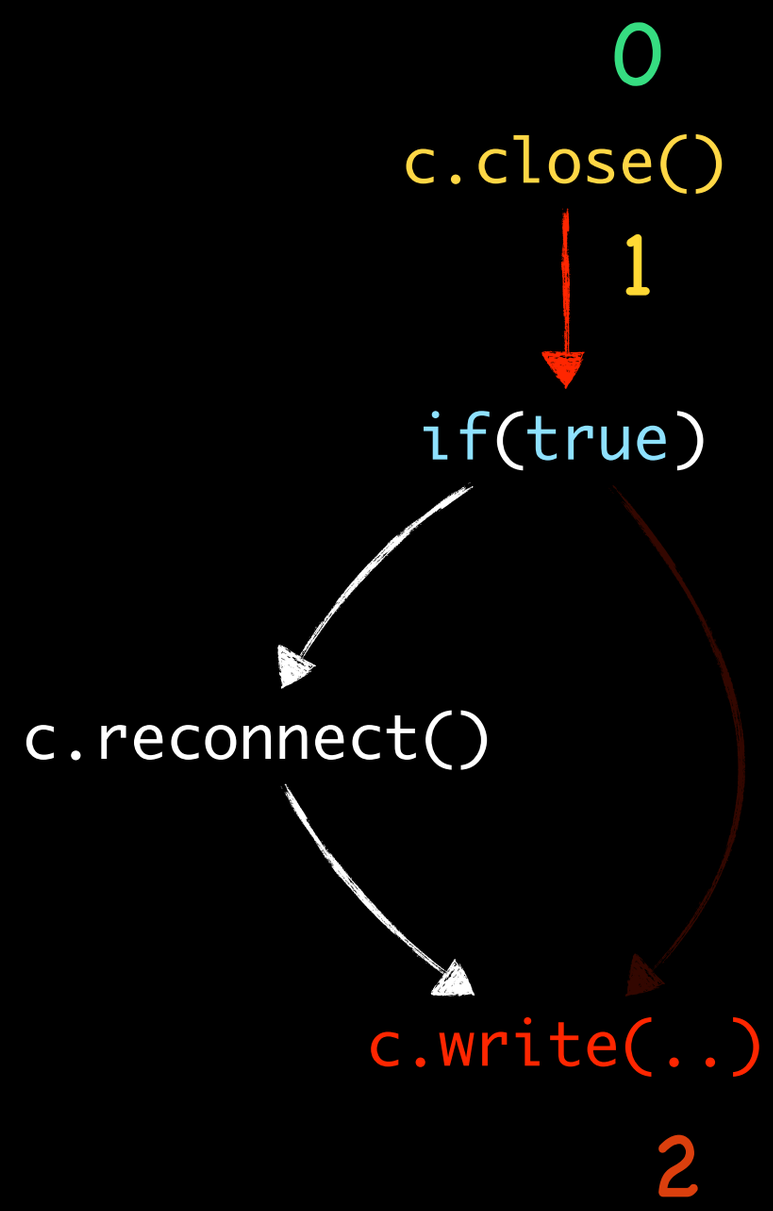
Which instrumentation to remove?



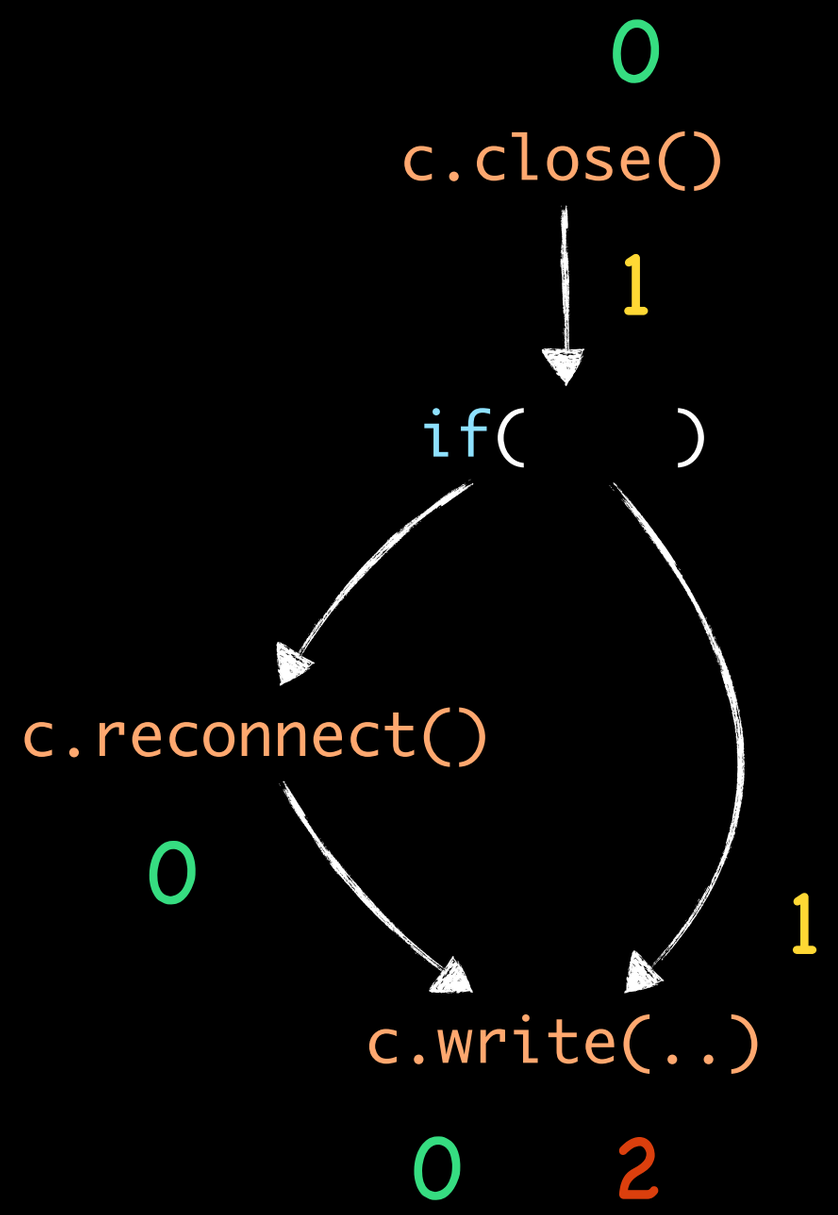
Which instrumentation to remove?

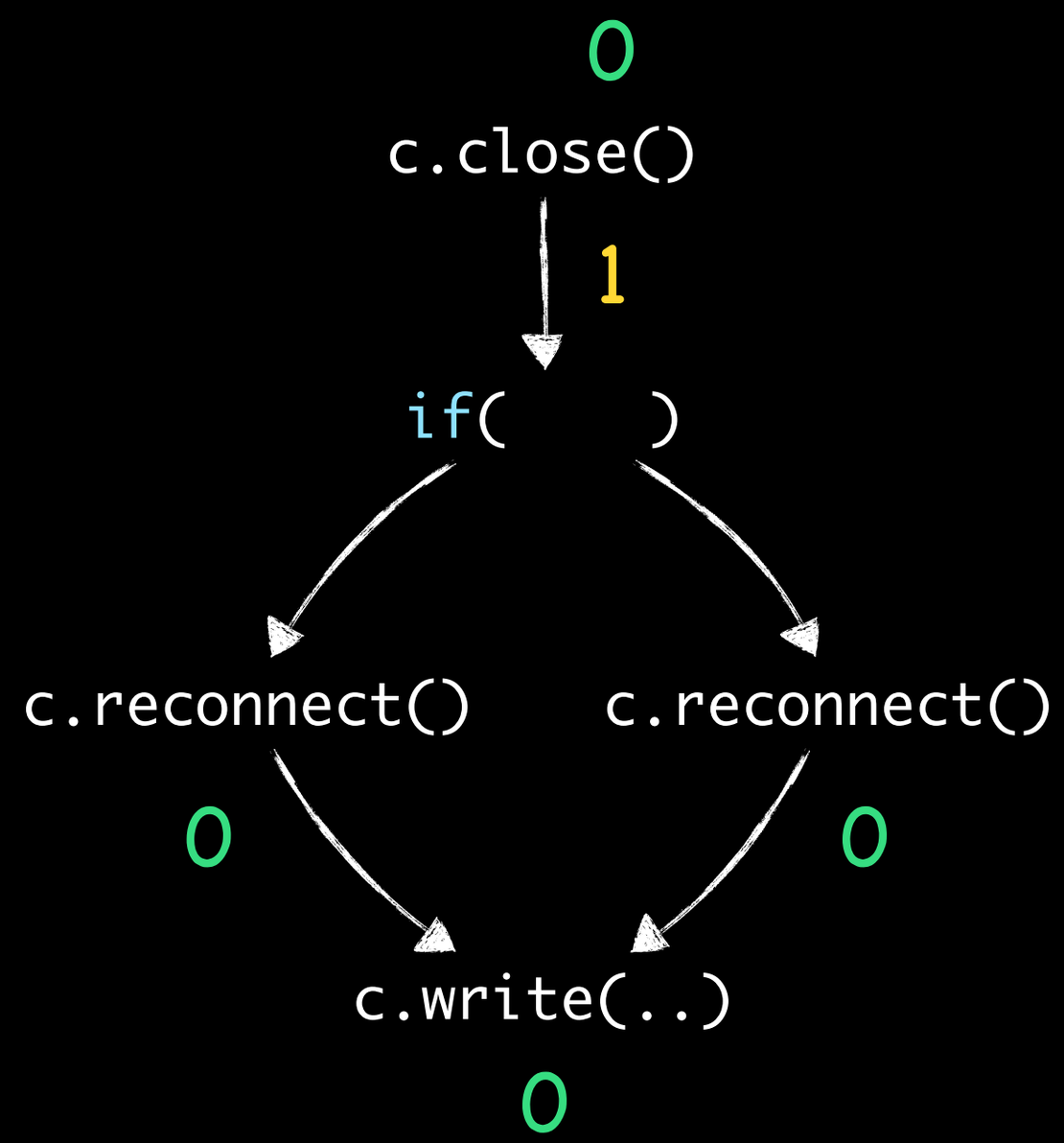


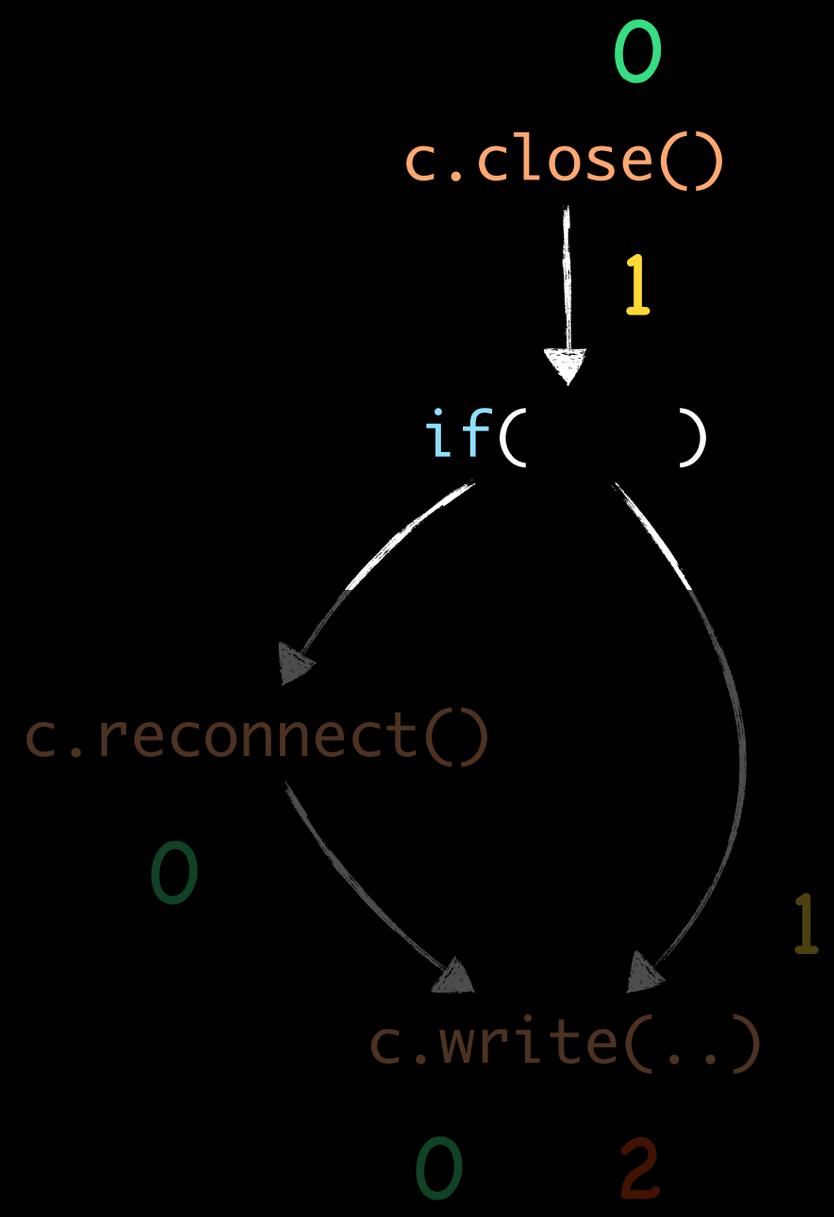
Which instrumentation to remove?

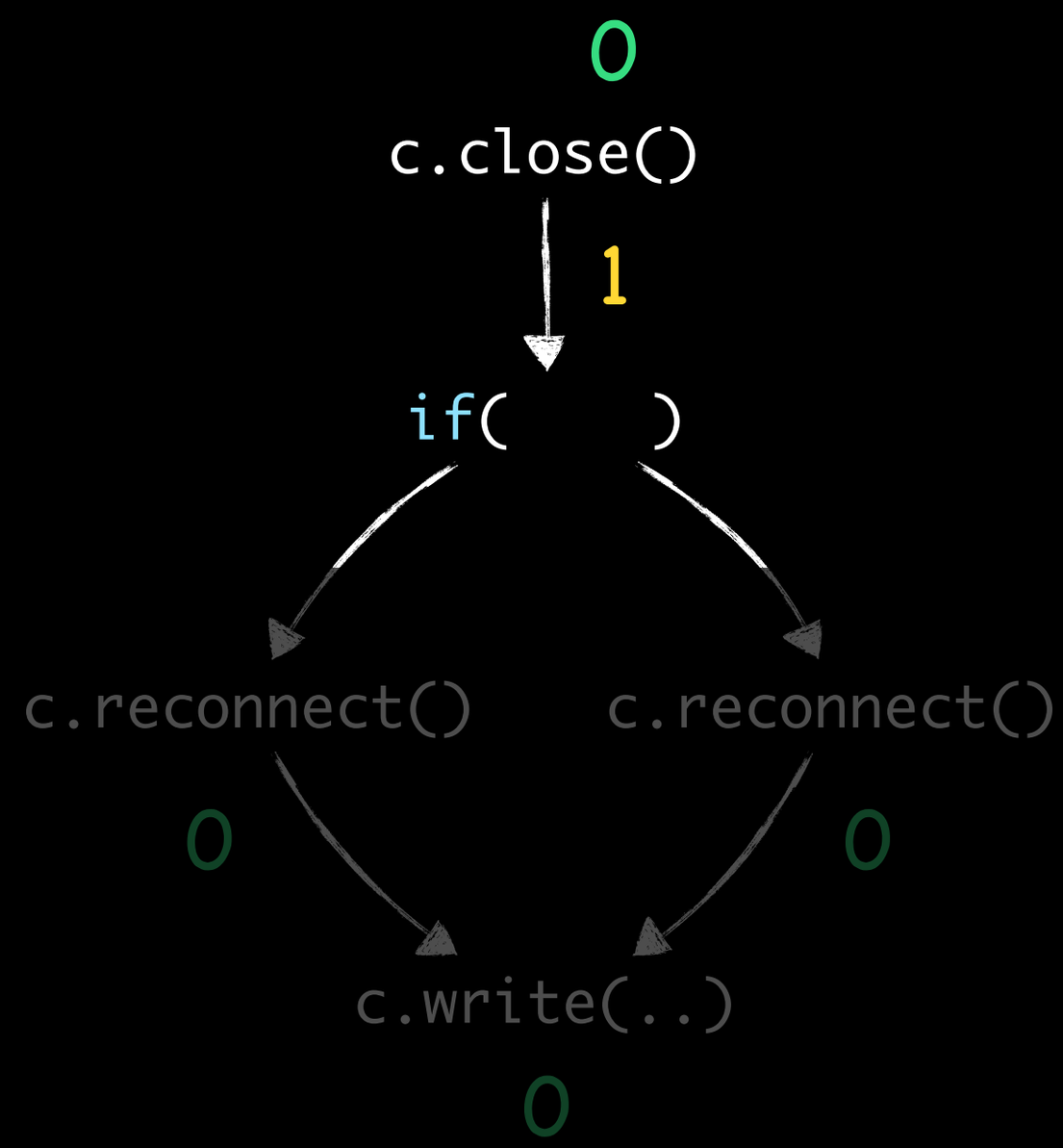


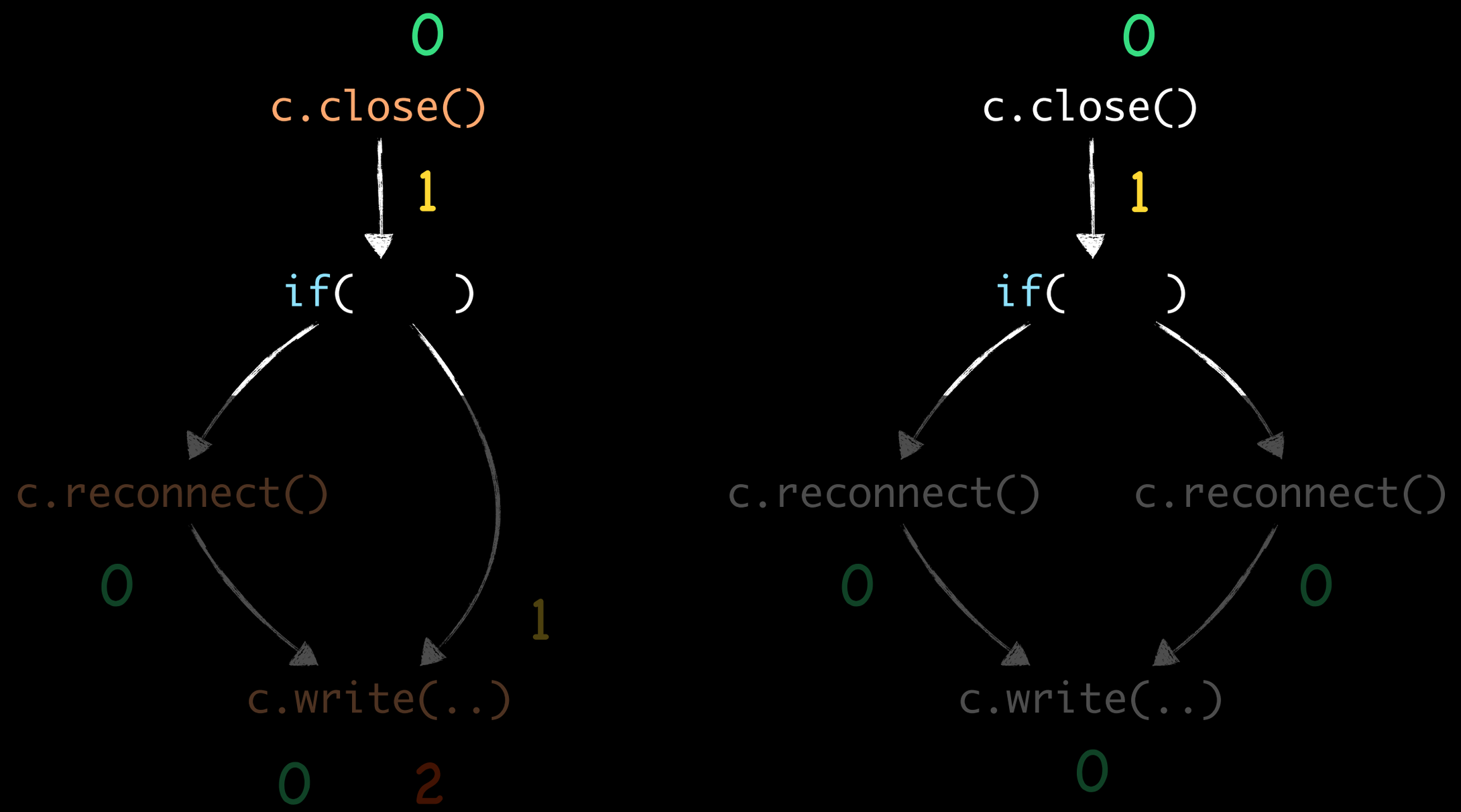
Which instrumentation to remove?


0
c.close()
1
if( )
c.reconnect()
0
1
c.write(..)
0
2


0
c.close()
1
if( )
c.reconnect()
c.reconnect()
0
0
c.write(..)
0


0
c.close()
1
if( )
c.reconnect()
0
1
c.write(..)
0
2


0
c.close()
1
if( )
c.reconnect()
c.reconnect()
0
0
c.write(..)
0



Forward analysis can never suffice!

Different Example: Dynamic Call Graphs

```
void visitNode(Visitor v) {
  left.accept(v);
  insertEdge("Node.accept(Visitor)");
  right.accept(v);
  insertEdge("Node.accept(Visitor)");
}
```

Different Example: Dynamic Call Graphs

```
void visitNode(Visitor v) {
  left.accept(v);
  insertEdge("Node.accept(Visitor)");
  right.accept(v);
  insertEdge("Node.accept(Visitor)");
}
```

Different Example: Dynamic Call Graphs

```
void visitNode(Visitor v) {
  left.accept(v);
  insertEdge("Node.accept(Visitor)");
  right.accept(v);
  insertEdge("Node.accept(Visitor)");
}
```

Again: Forward analysis cannot suffice!

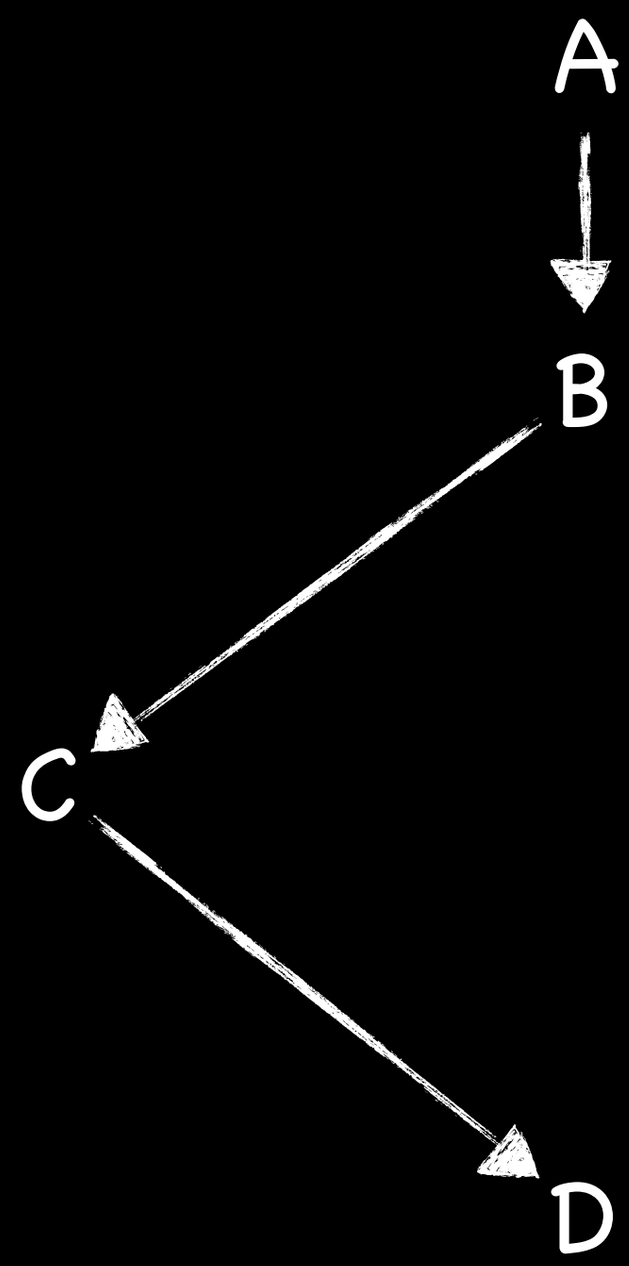
Continuation Equivalence

May disable instrumentation
at a statement s

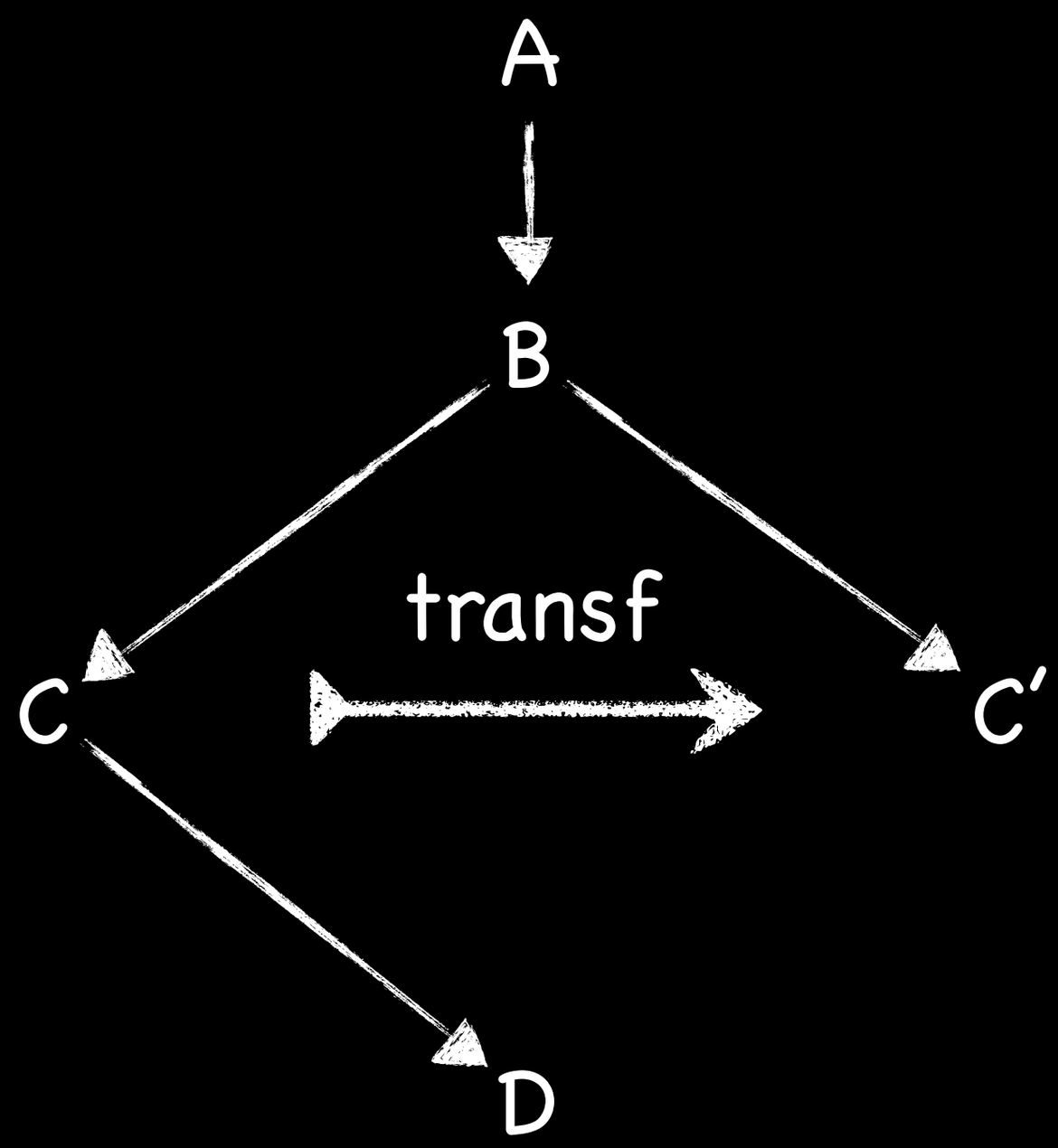
if and only if

dynamic-analysis configuration
will eventually be the same
on all possible continuations.

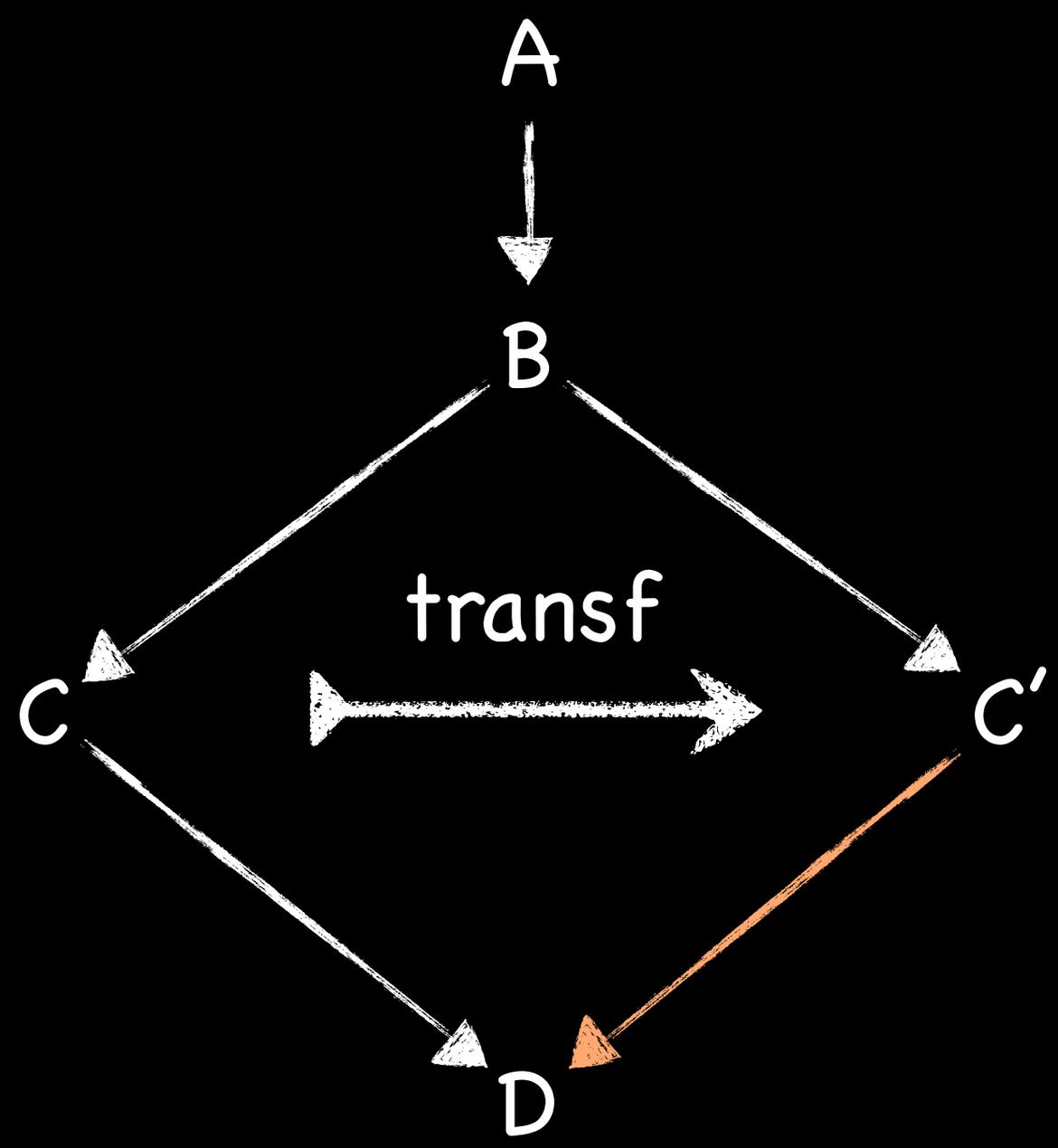
Continuation Equivalence



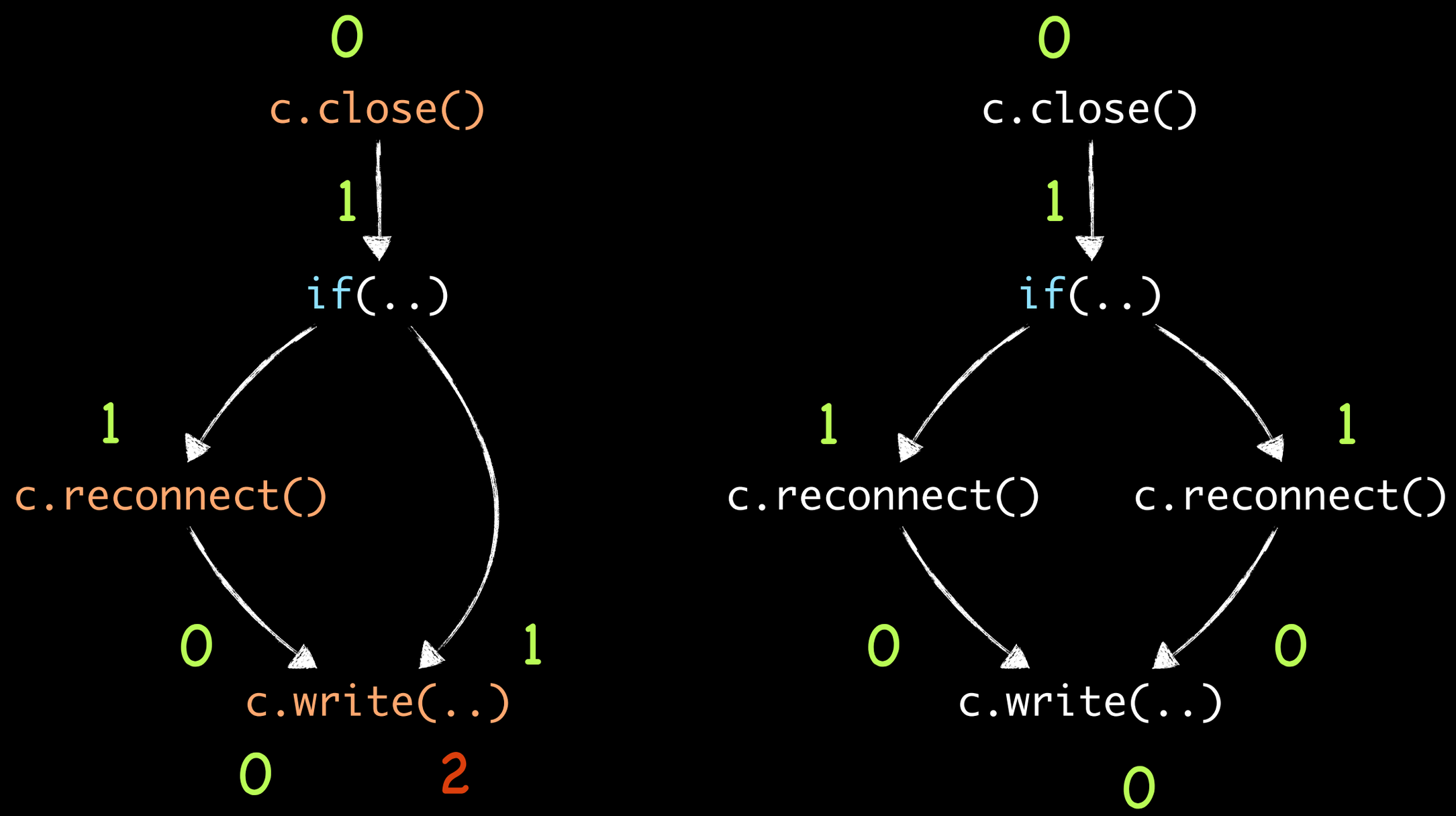
Continuation Equivalence



Continuation Equivalence

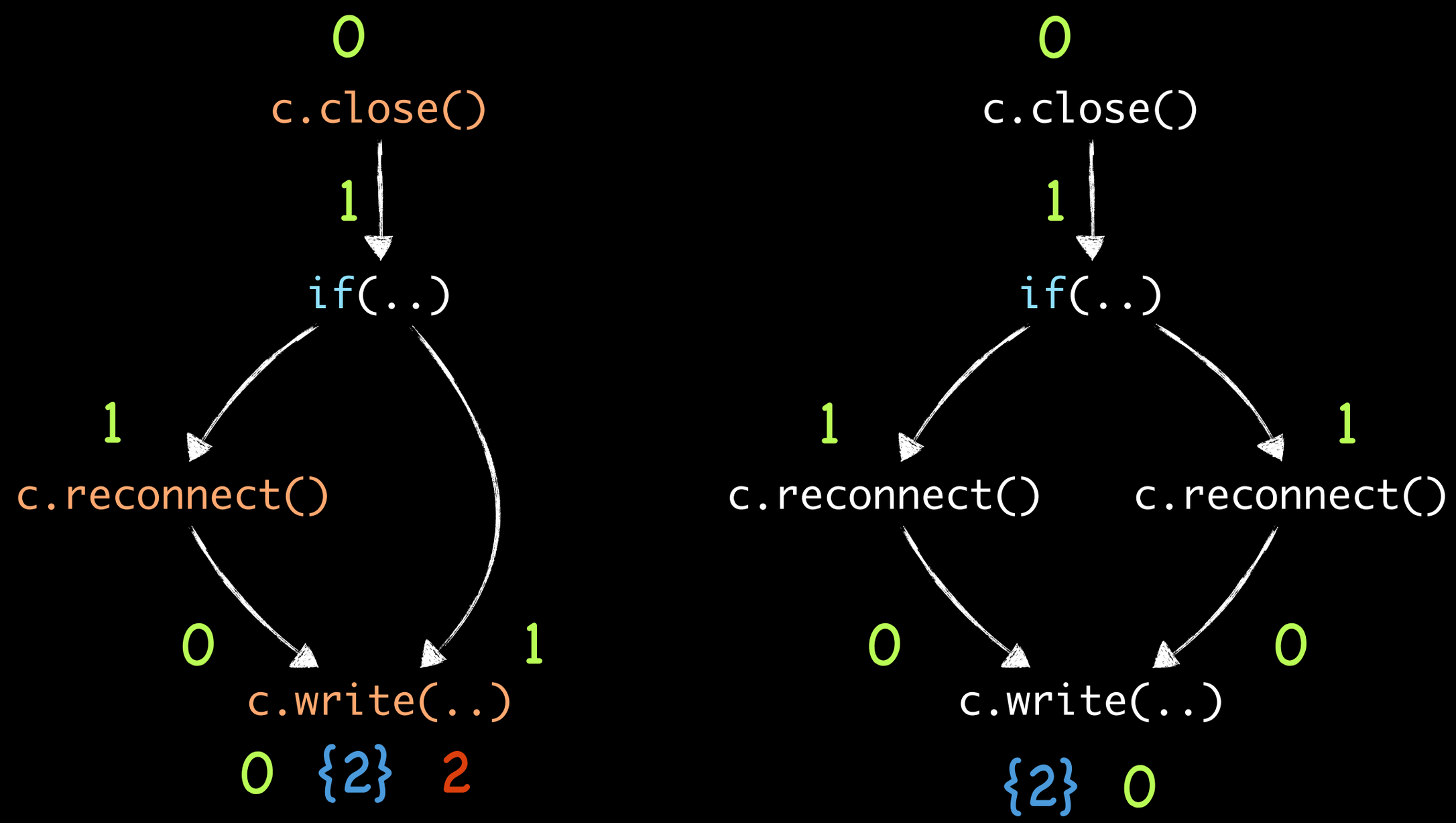


Continuation Equivalent States



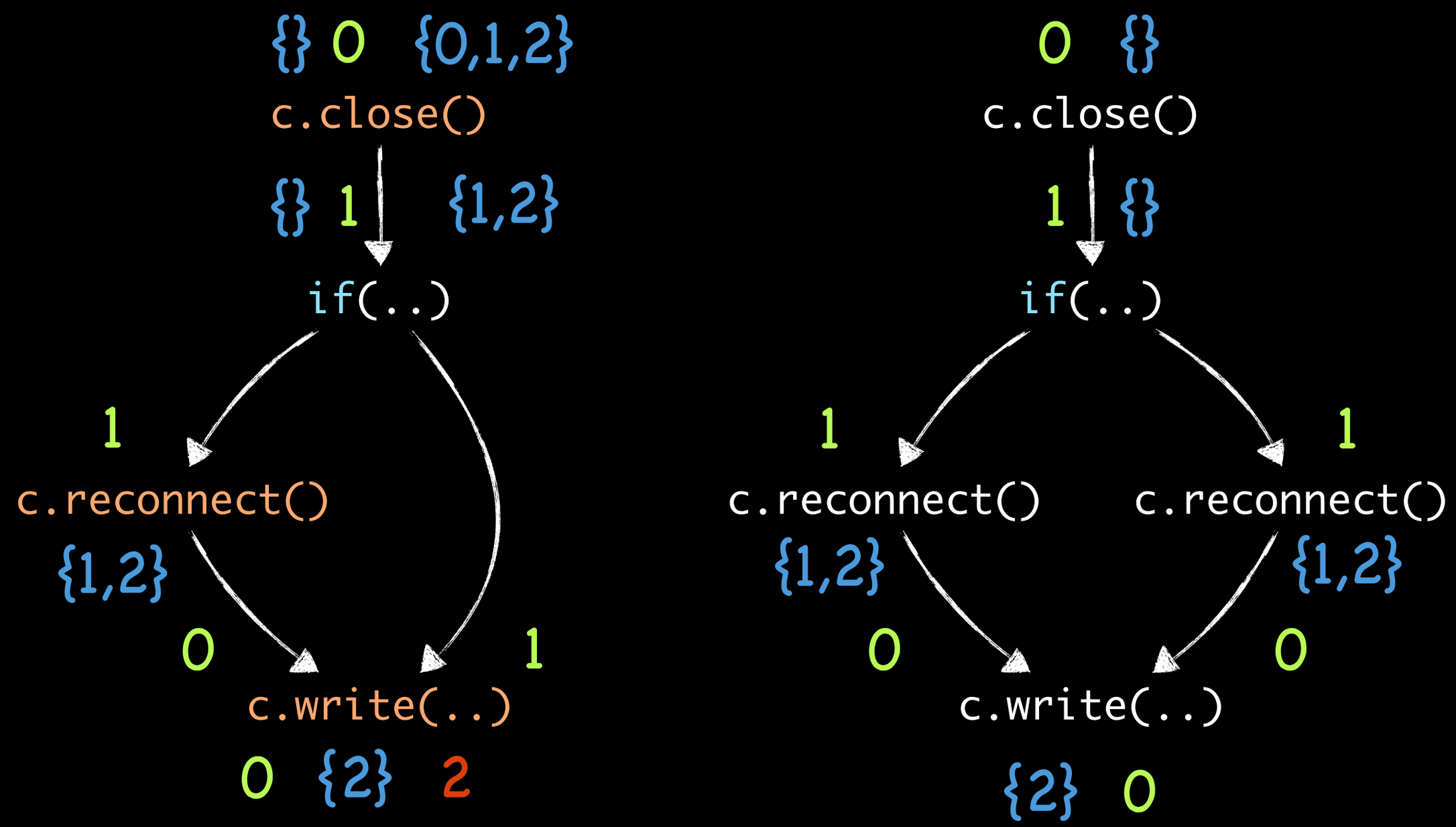
[ICSE 2010]

Continuation Equivalent States



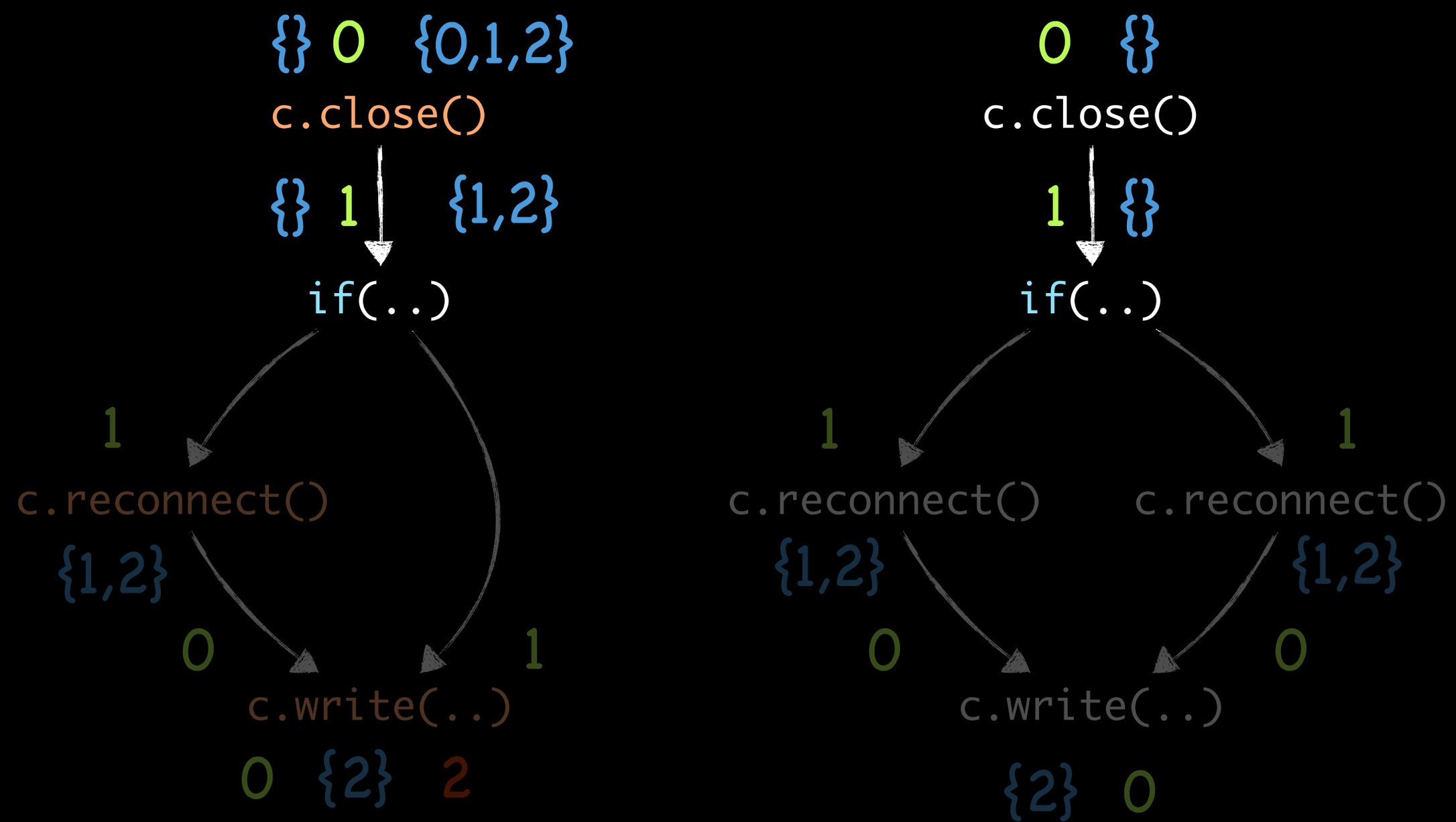
[ICSE 2010]

Continuation Equivalent States



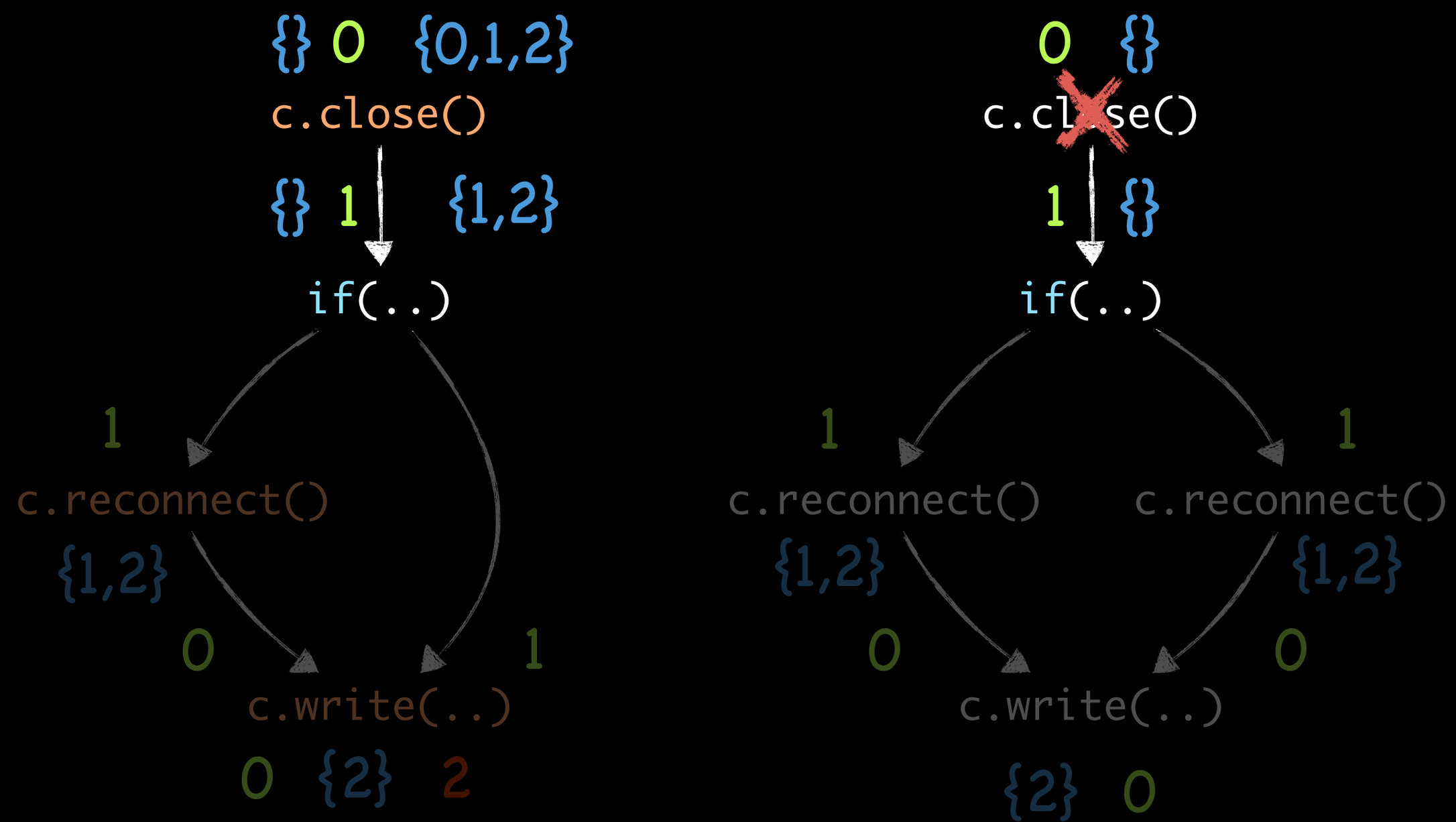
[ICSE 2010]

Continuation Equivalent States



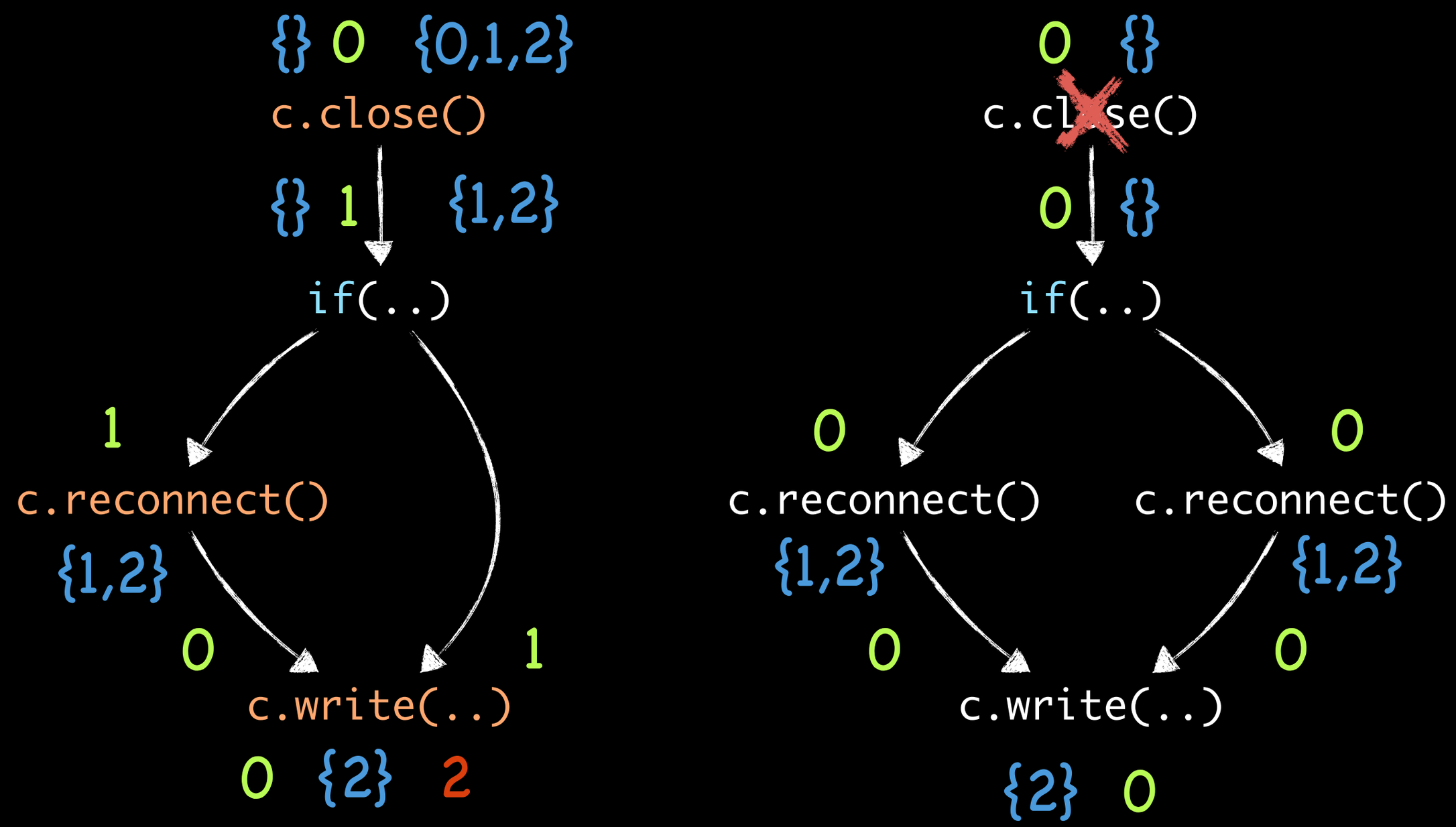
[ICSE 2010]

Continuation Equivalent States



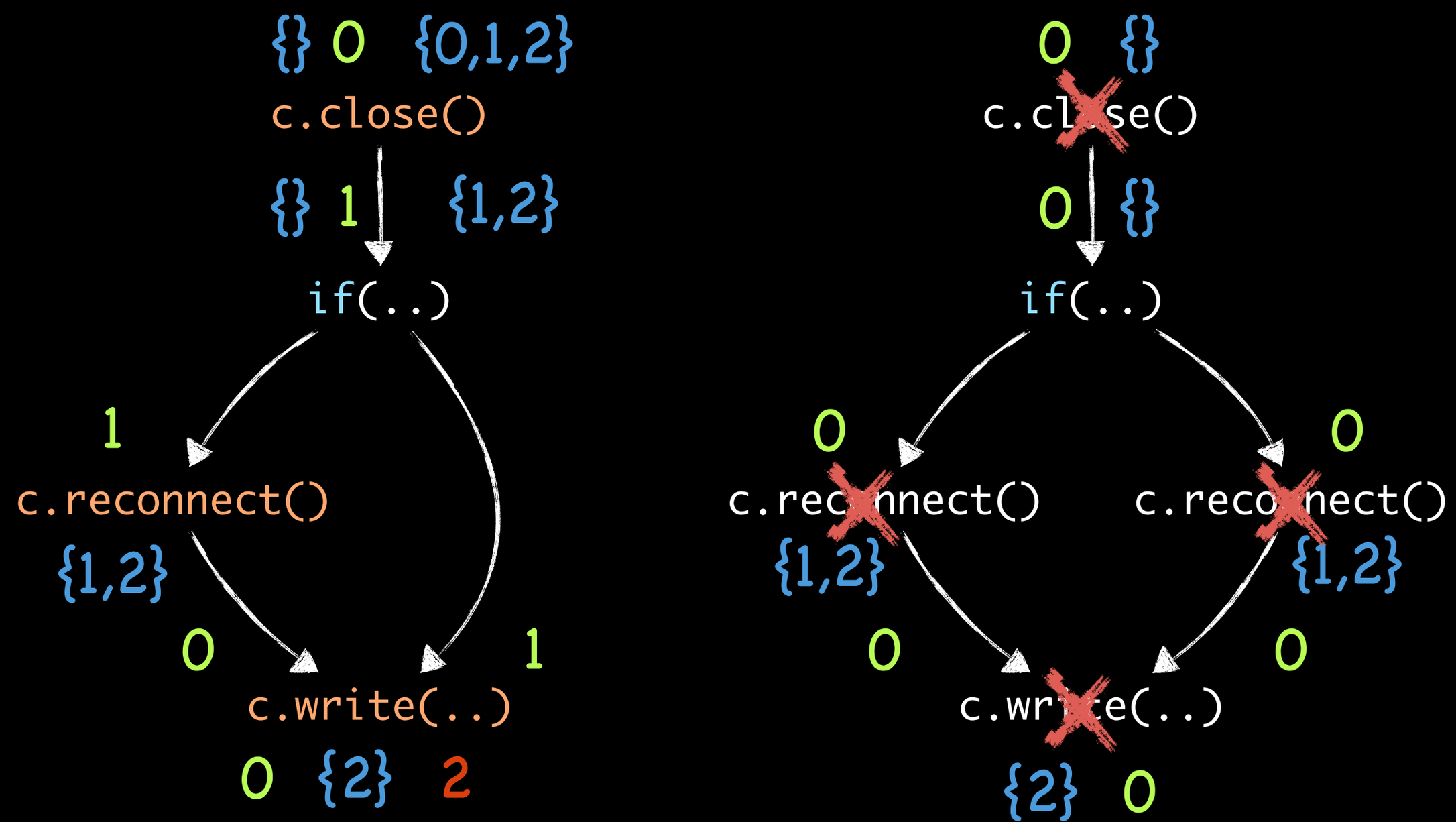
[ICSE 2010]

Continuation Equivalent States



[ICSE 2010]

Continuation Equivalent States



[ICSE 2010]

Lessons learned

Lesson 1:

Correct static approximations usually require both backward and forward analysis

Lessons learned

Lesson 2:

Forward analysis needs to approximate all possible histories

Backward analysis needs to approximate all possible continuations

Lessons learned

Lesson 3:

Optimizations may void previous static-analysis results

=> Re-iteration may be required

Lessons learned

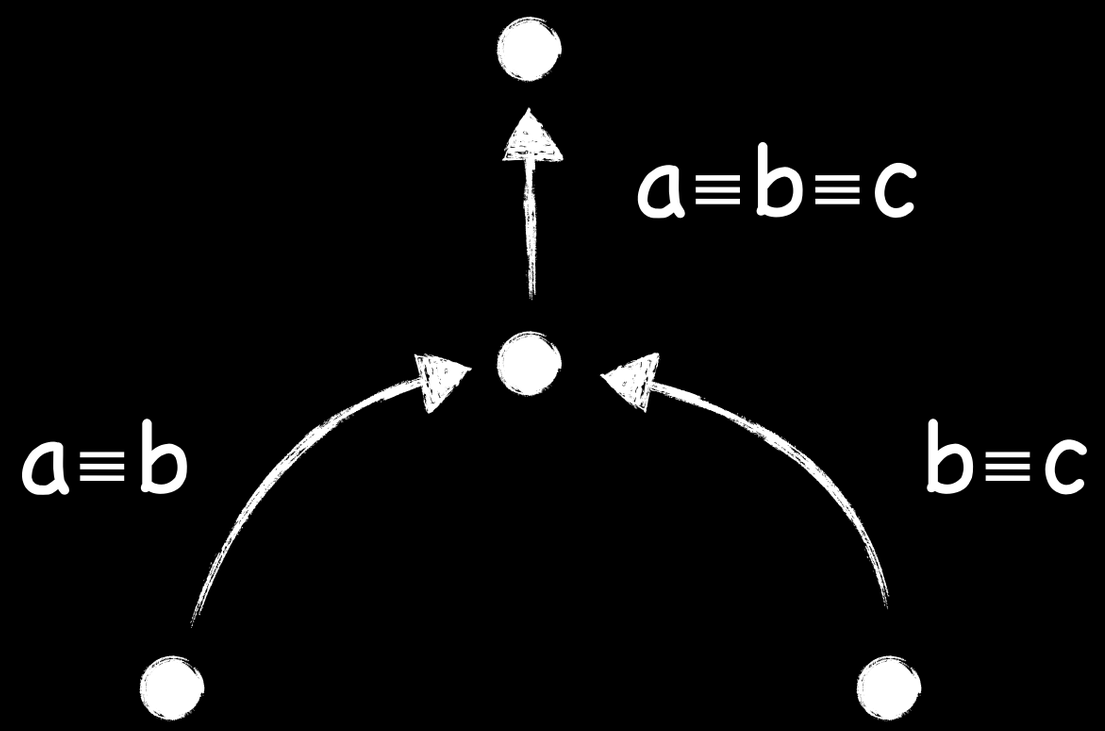
Lesson 4:

Analysis must be path sensitive

Lessons learned

Lesson 4:

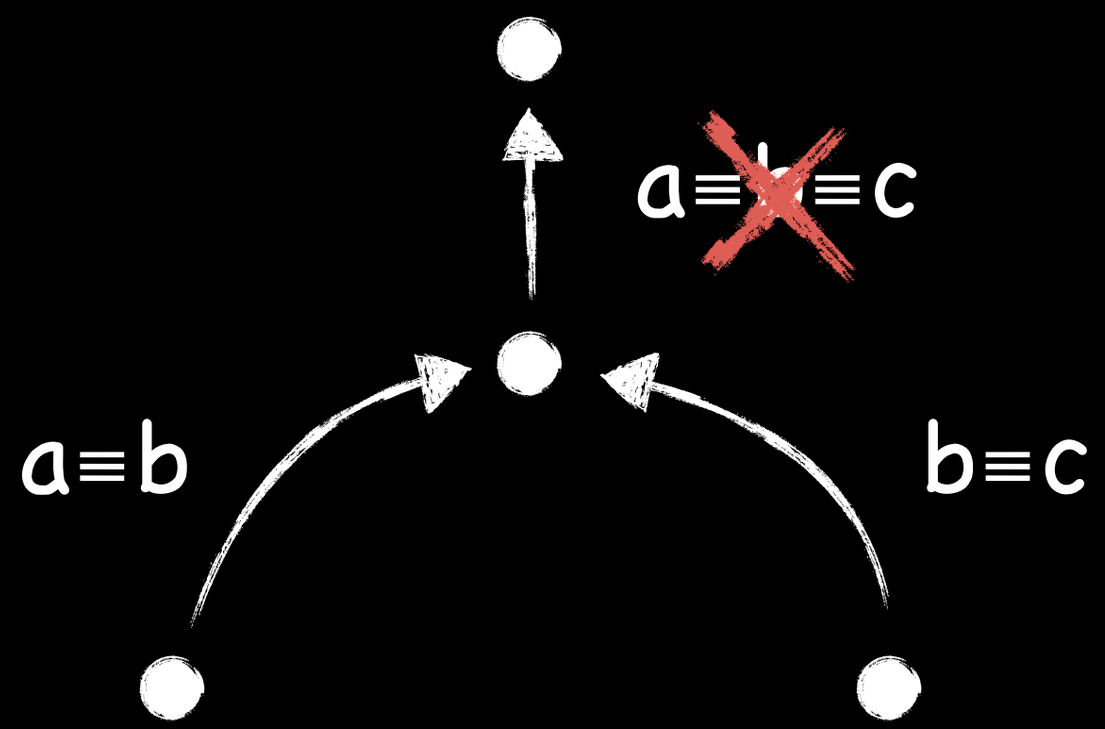
Analysis must be path sensitive



Lessons learned

Lesson 4:

Analysis must be path sensitive



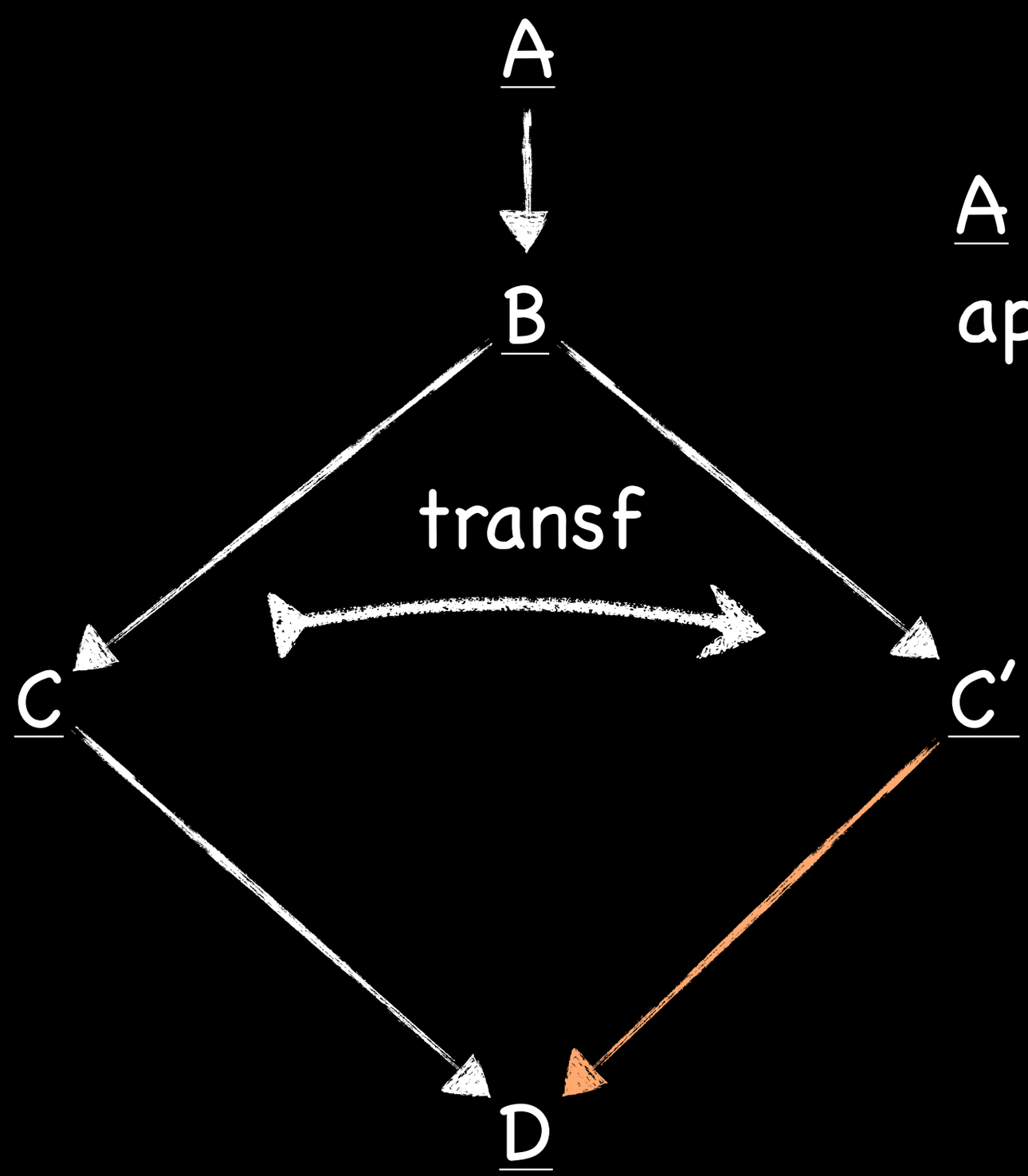
Idea: Generic Framework

- Static-analysis Framework
  - Automated propagation along all possible continuations
  - Reusable static approximations of runtime entities (Objects, events, etc.)
  - Support for different notions of equivalence
  - Pre-defined code transformations for optimizing instrumentation

Idea: Generic Framework

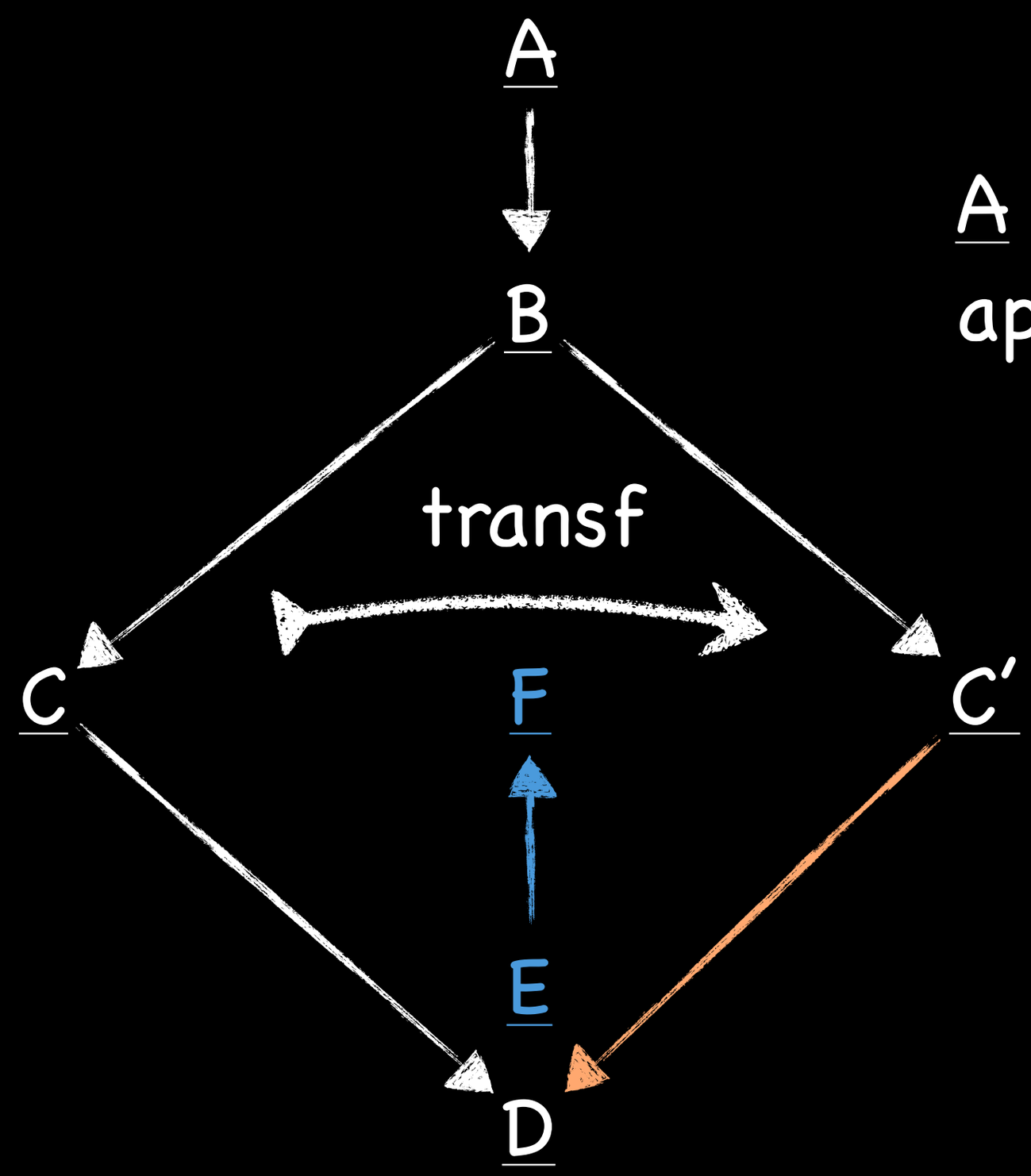
- Formal Proof Framework
  - Generic framework defining possible control flows and continuations
  - Generic static approximations of runtime entities (Objects, events, etc.)
  - Simply "plug in" your analysis' special notion of continuation-equivalence

Proving Continuation Equivalence



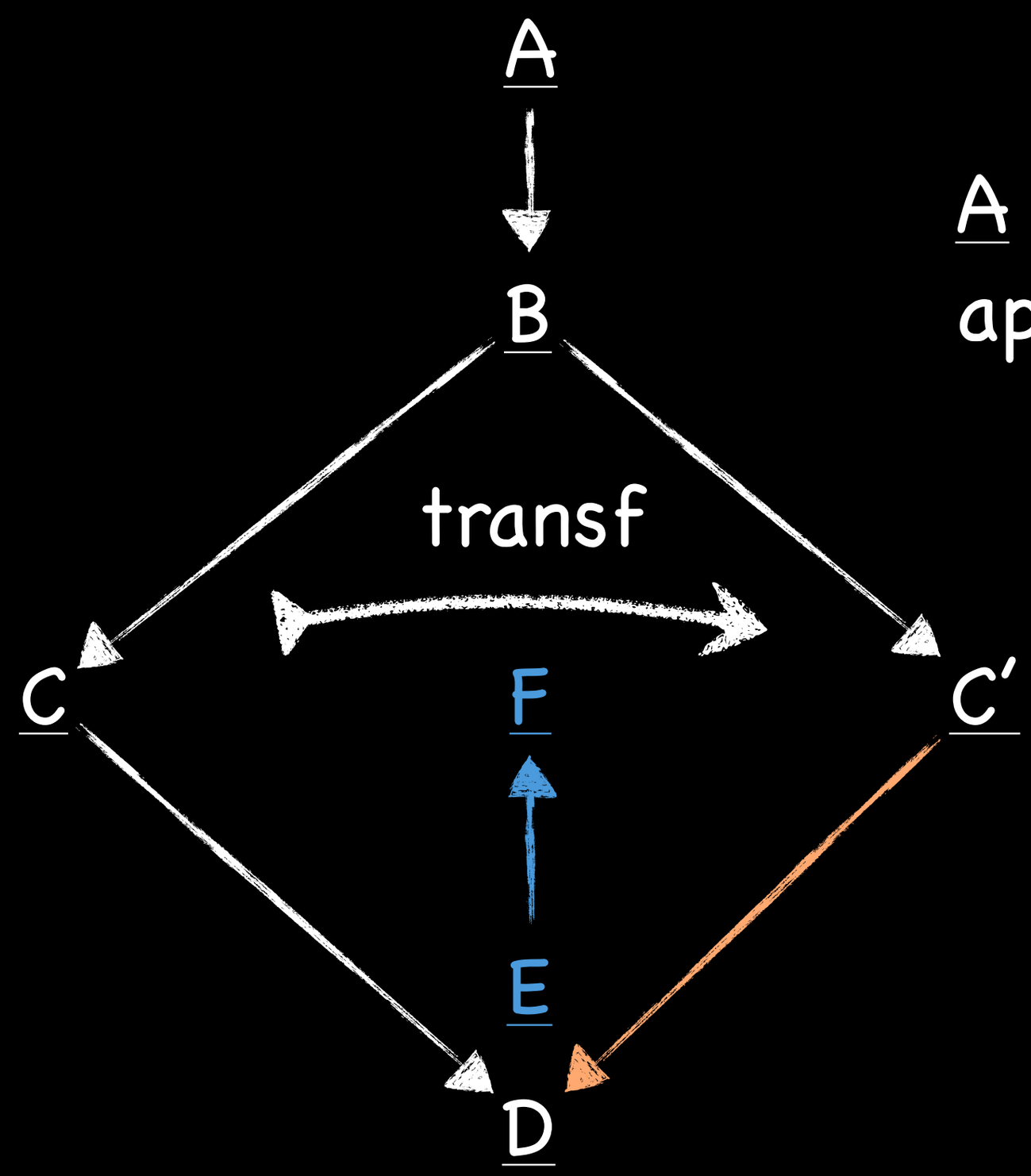
A static approximation of A

Proving Continuation Equivalence



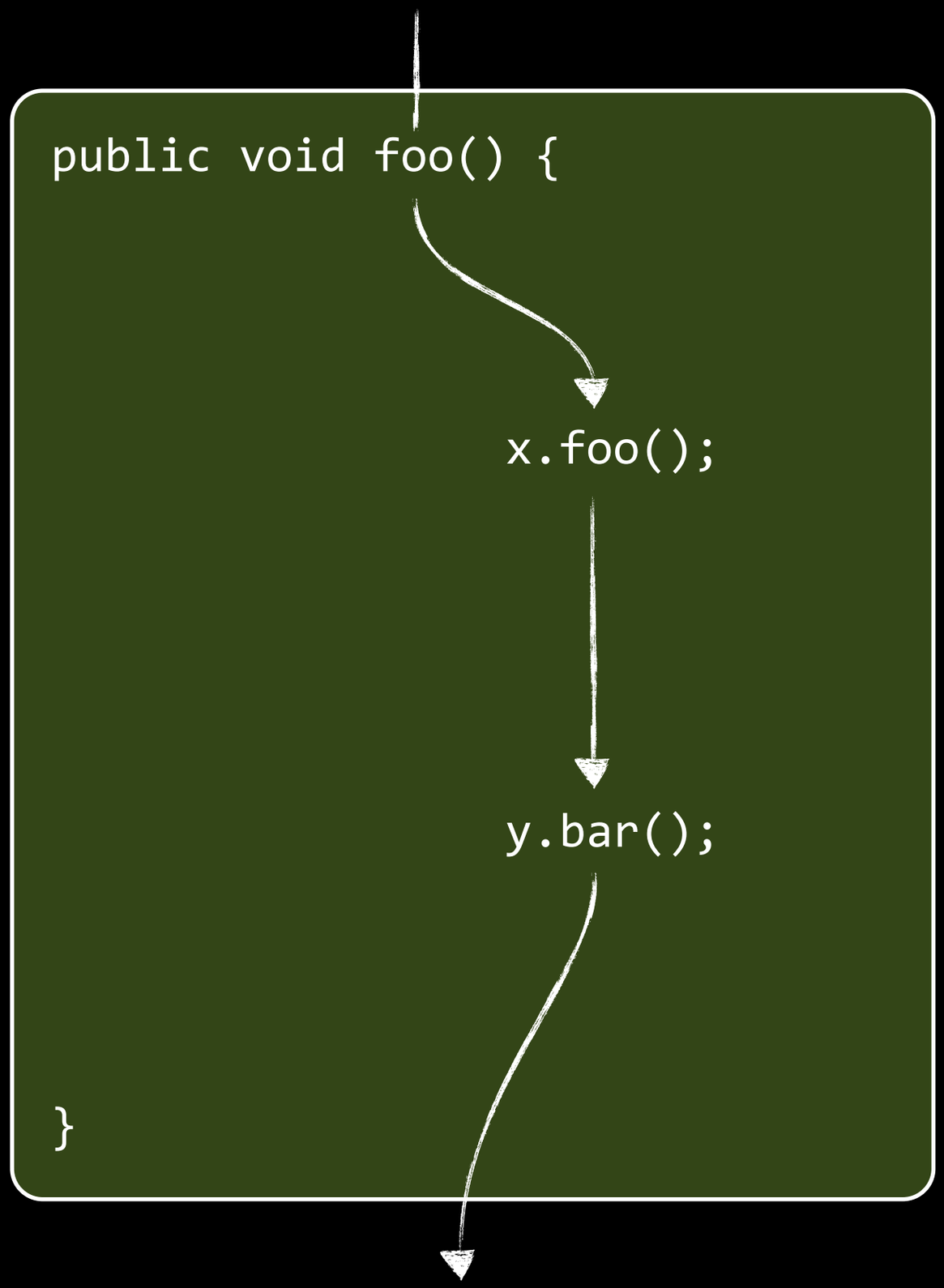
A static approximation of A

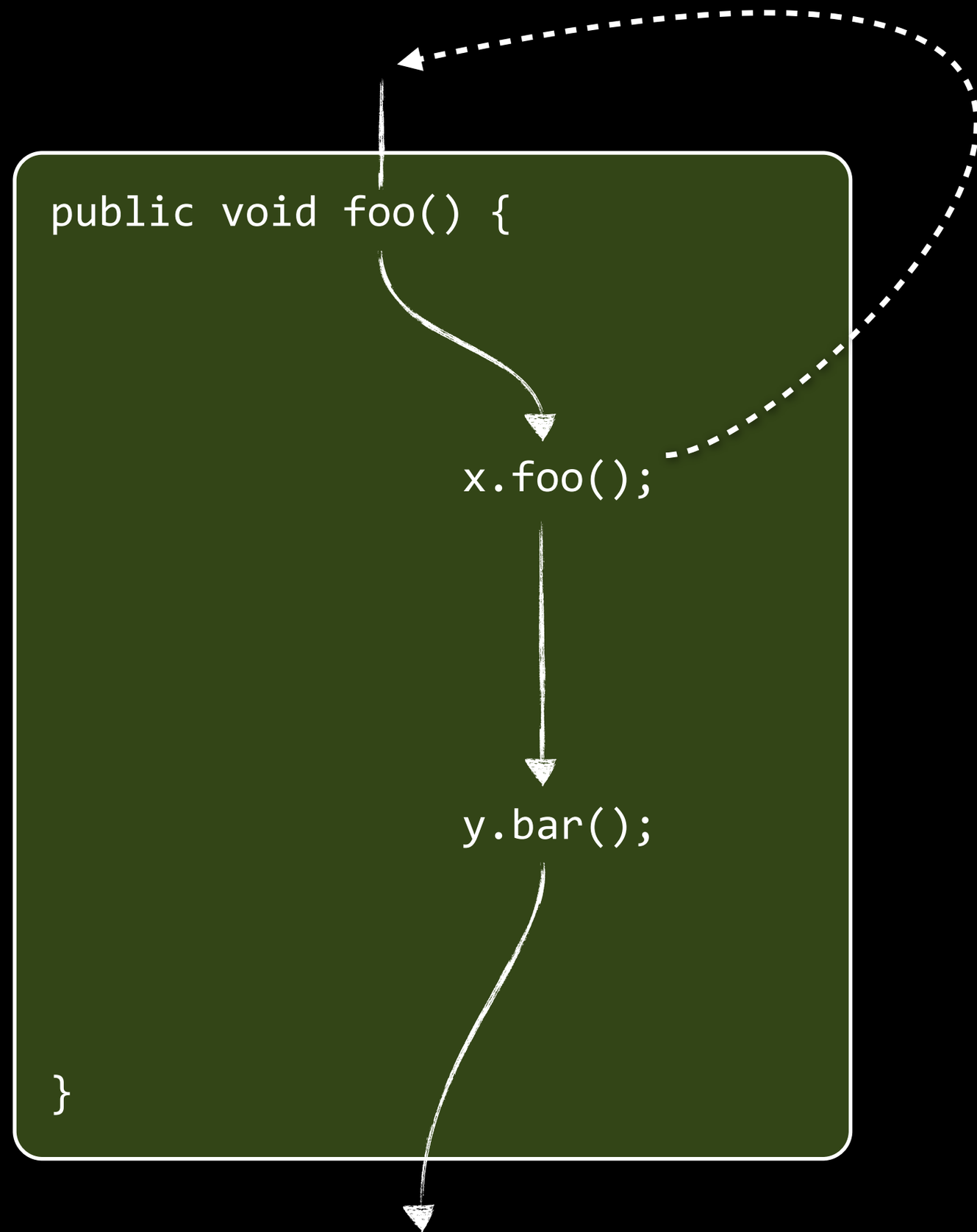
Proving Continuation Equivalence

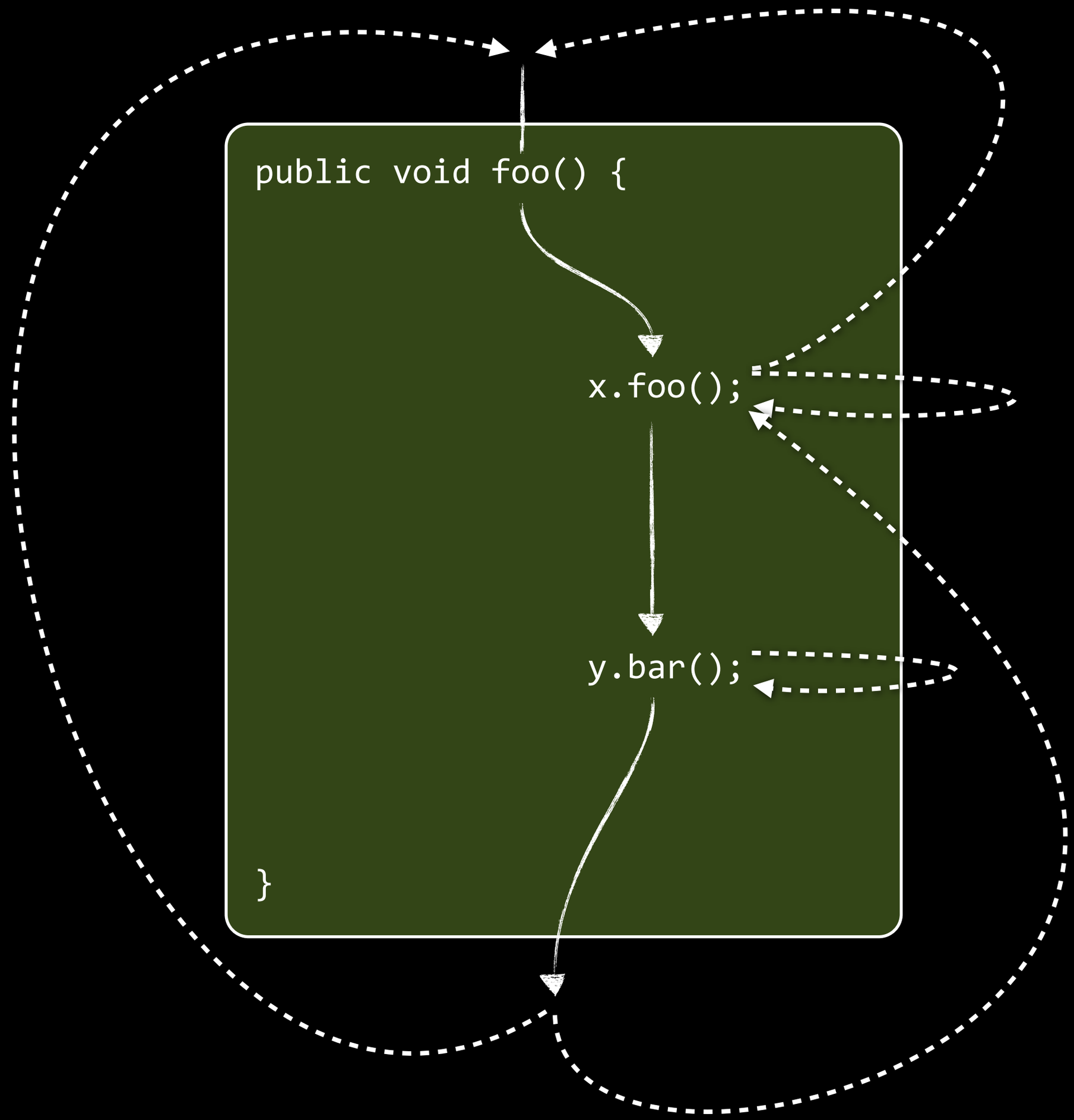


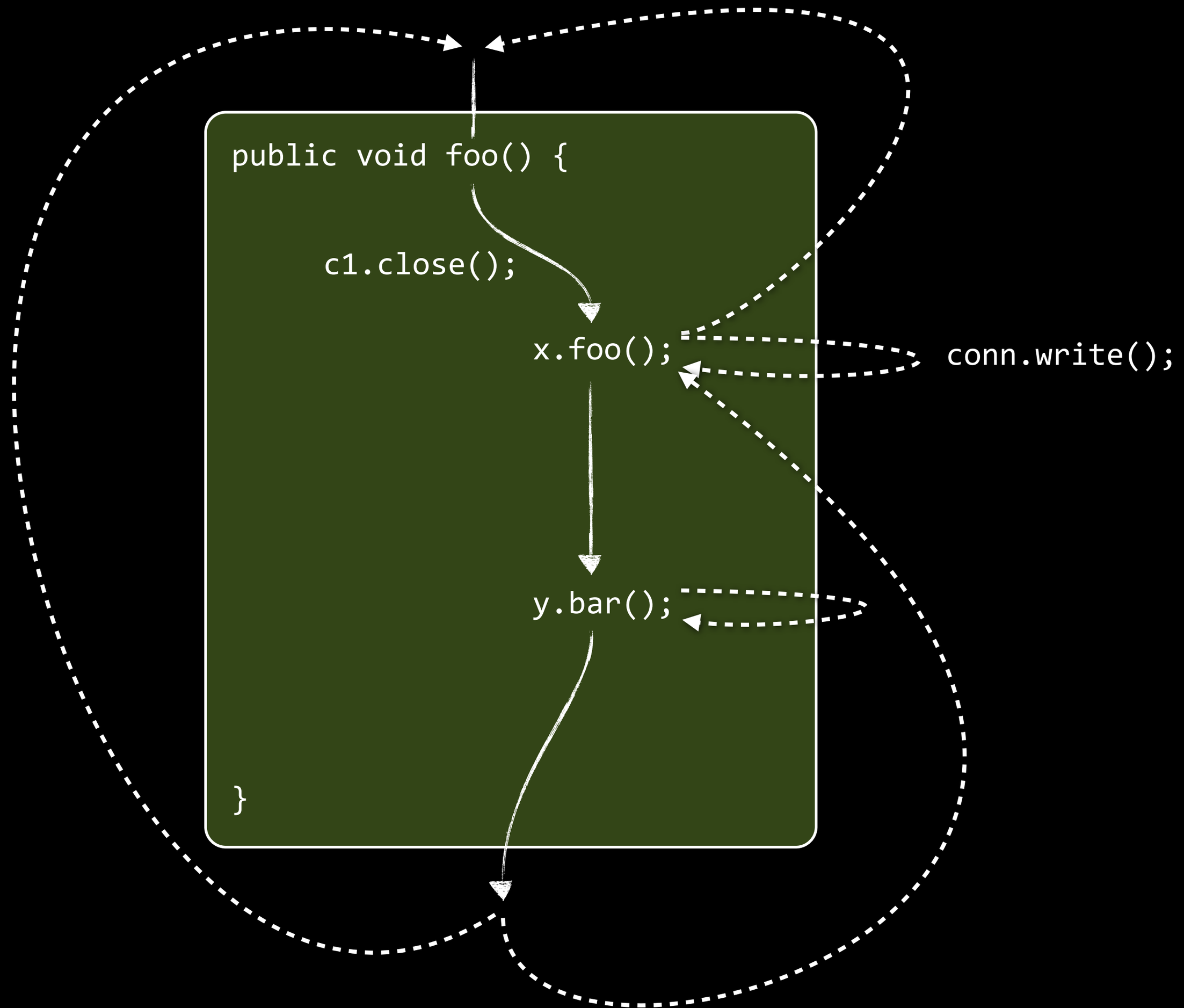
A static approximation of A

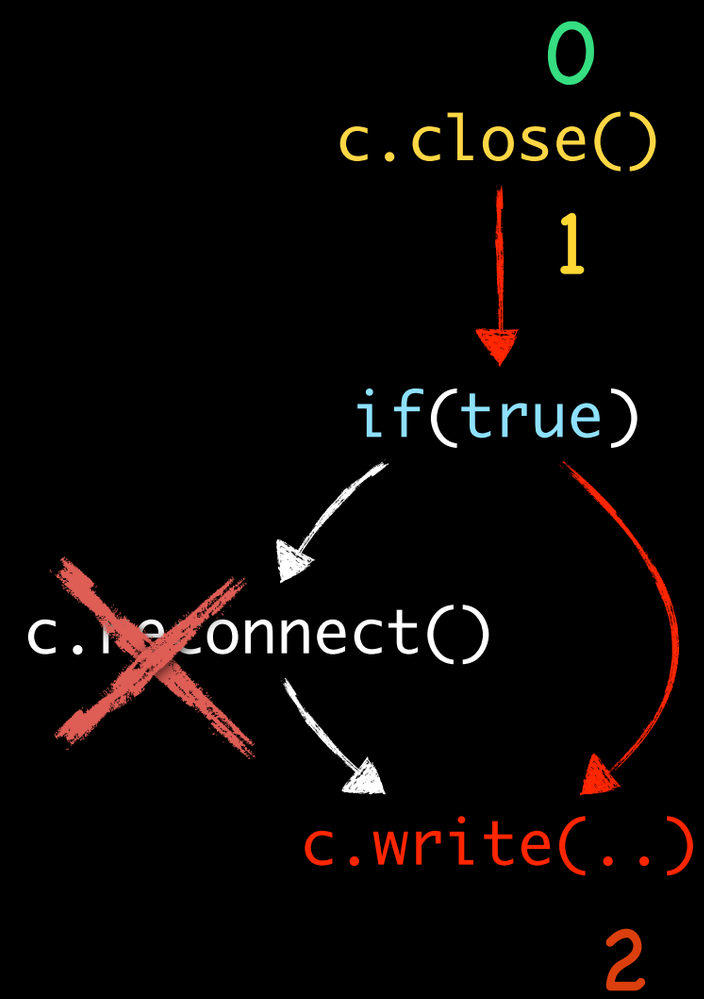
must hold:
C≡C′, given F

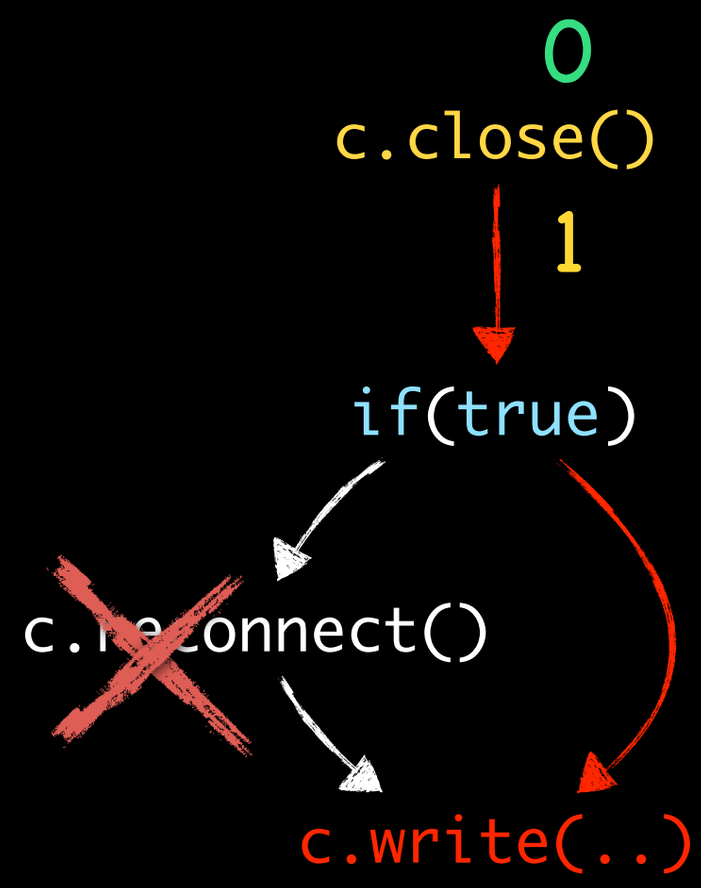

public void foo() {
x.foo();
y.bar();
}

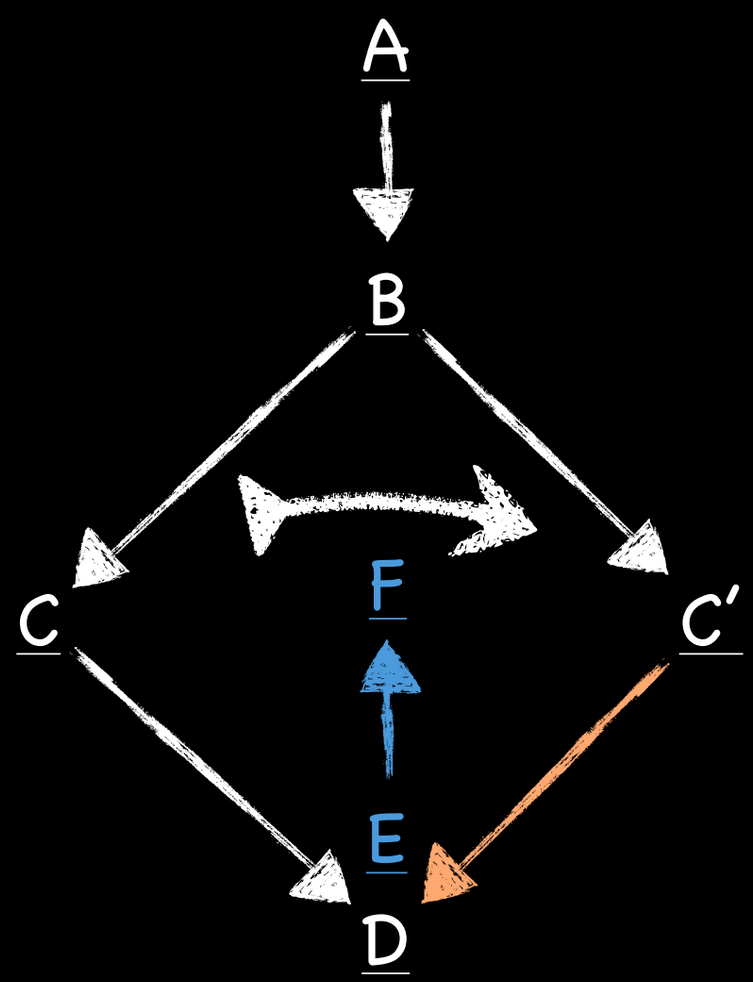

public void foo() {
x.foo();
y.bar();
}

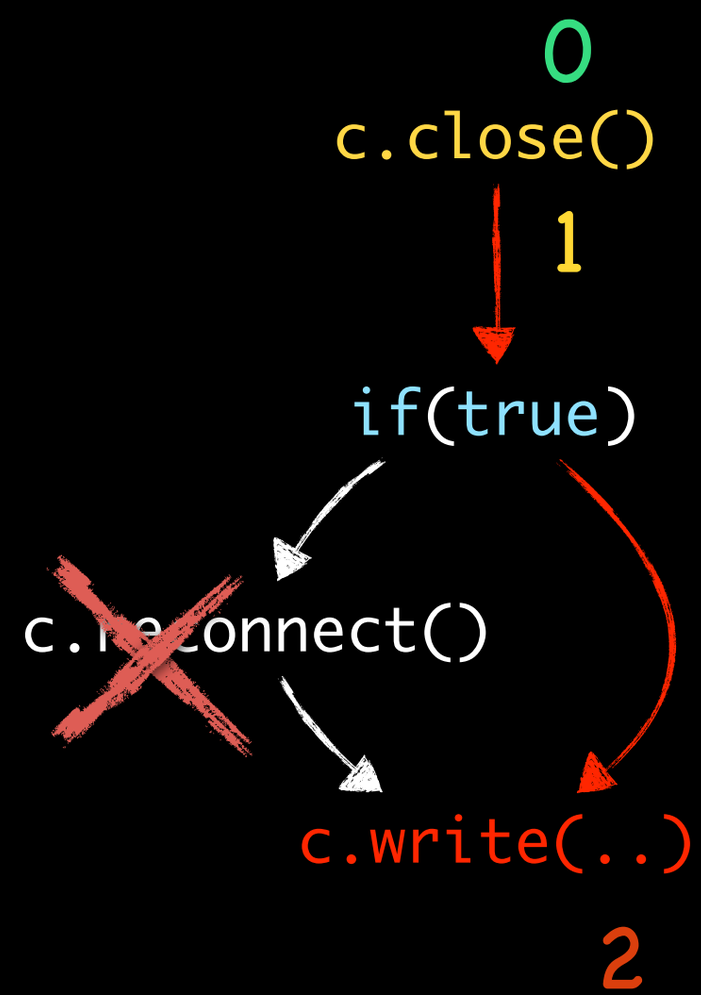

public void foo() {
x.foo();
y.bar();
}

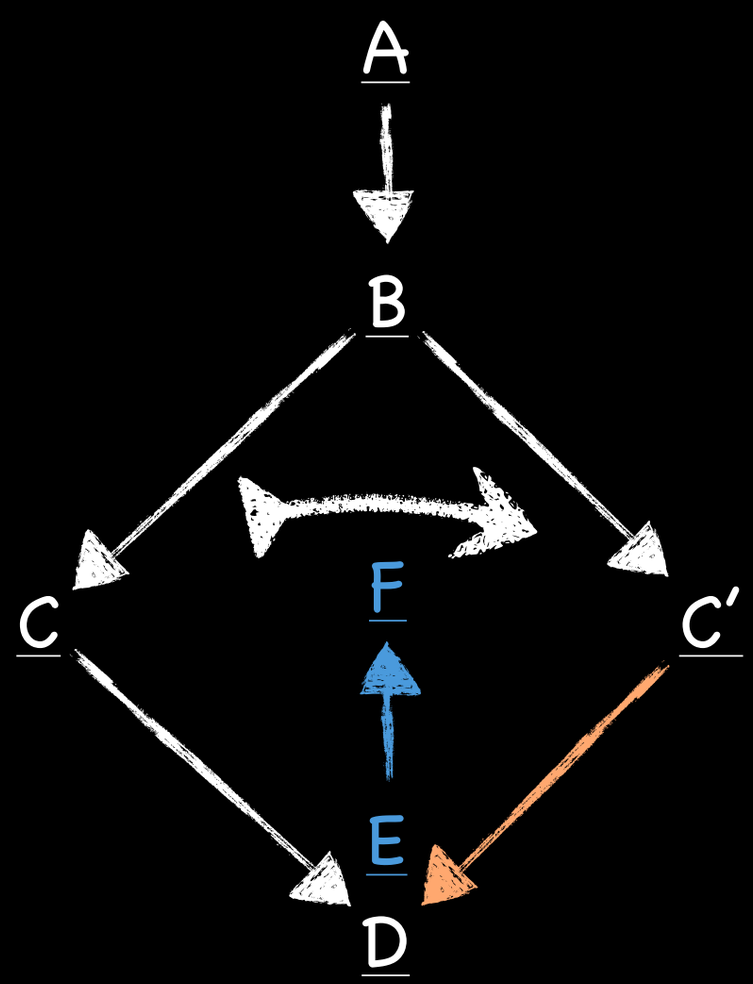

public void foo() {
c1.close();
x.foo();
conn.write();
y.bar();
}


0
c.close()
1
if(true)
c.reconnect()
c.write(..)
2


0
c.close()
1
if(true)
c.reconnect()
c.write(..)
2


A
B
C
C′
F
E
D



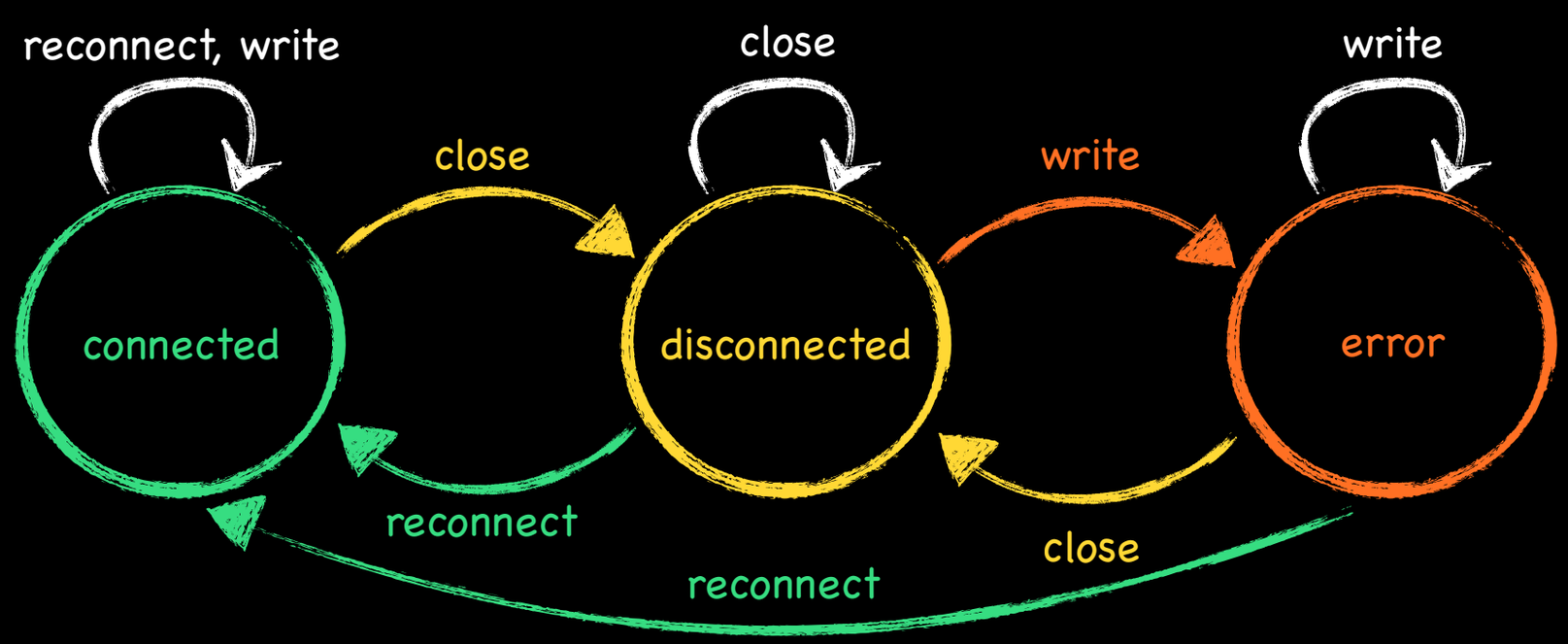


Grant application:

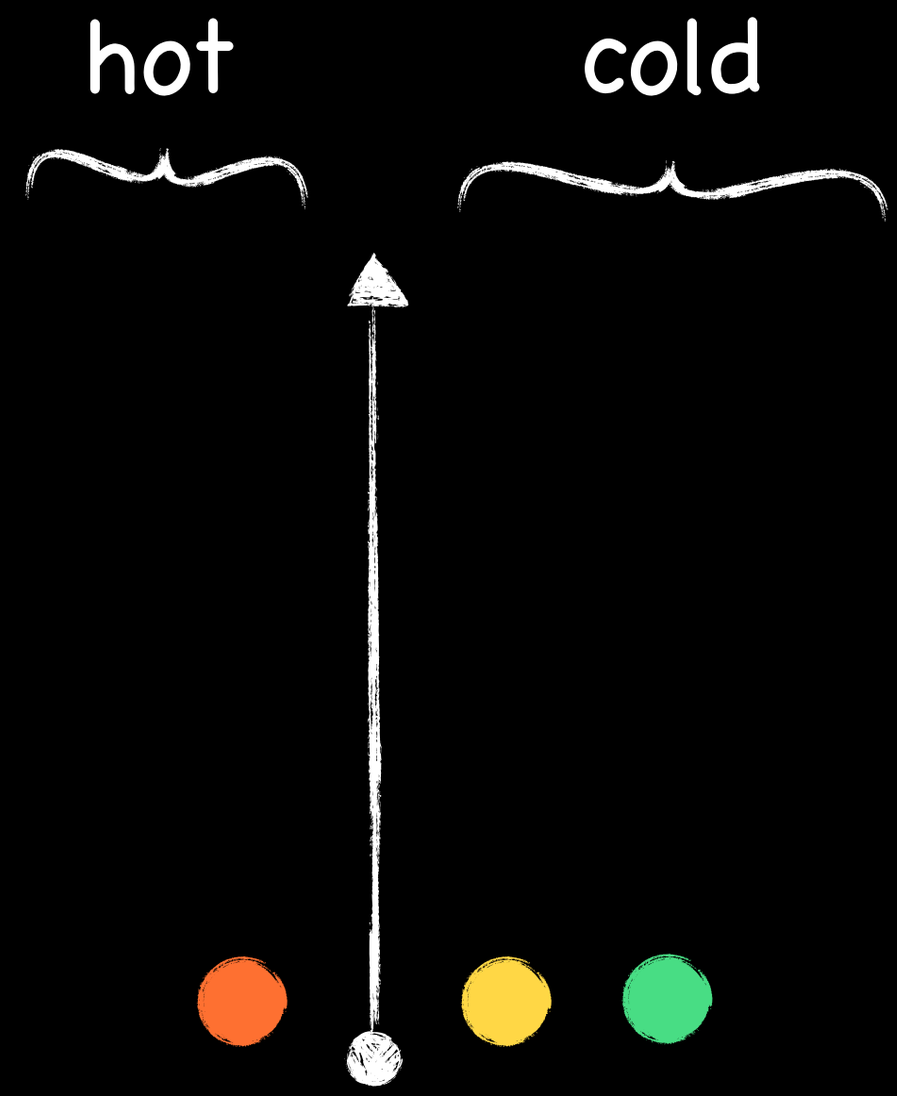
DFG Deutsche Forschungsgemeinschaft

1st Iteration

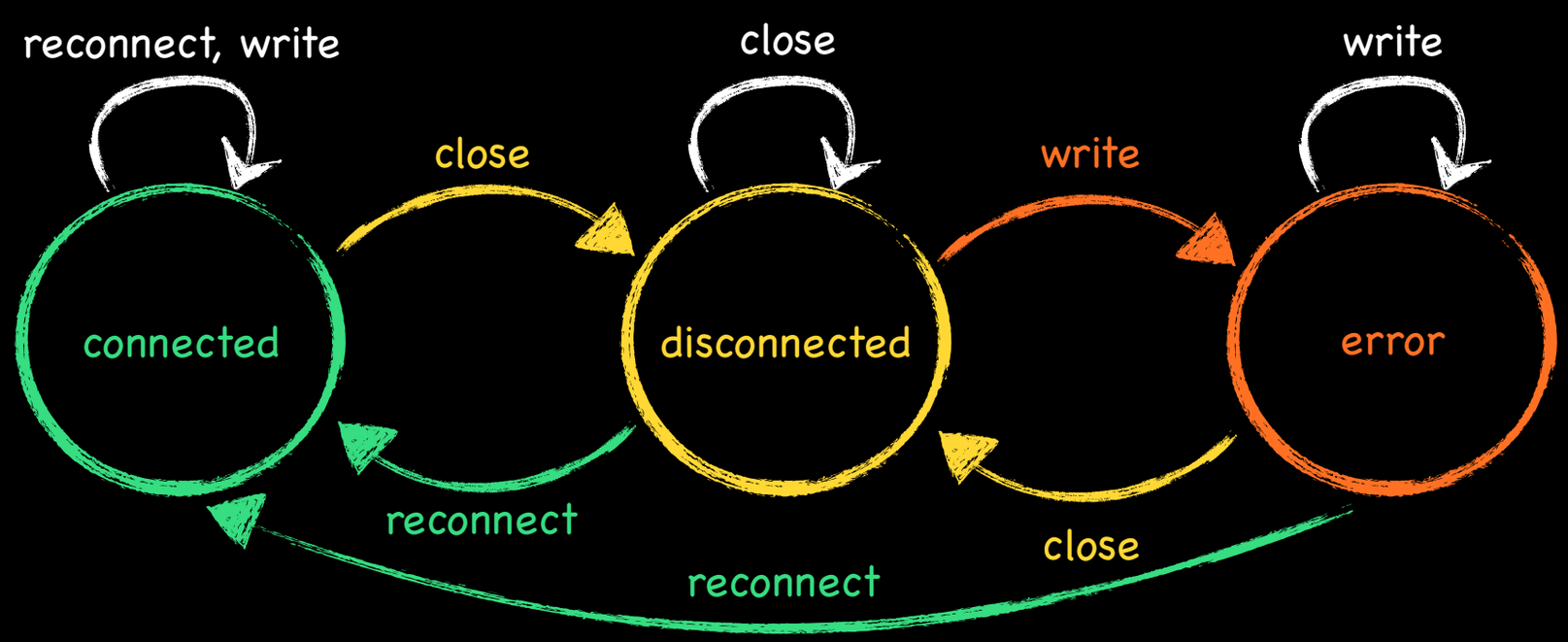
```
c.close()

c.reconnect()

c.write()
```



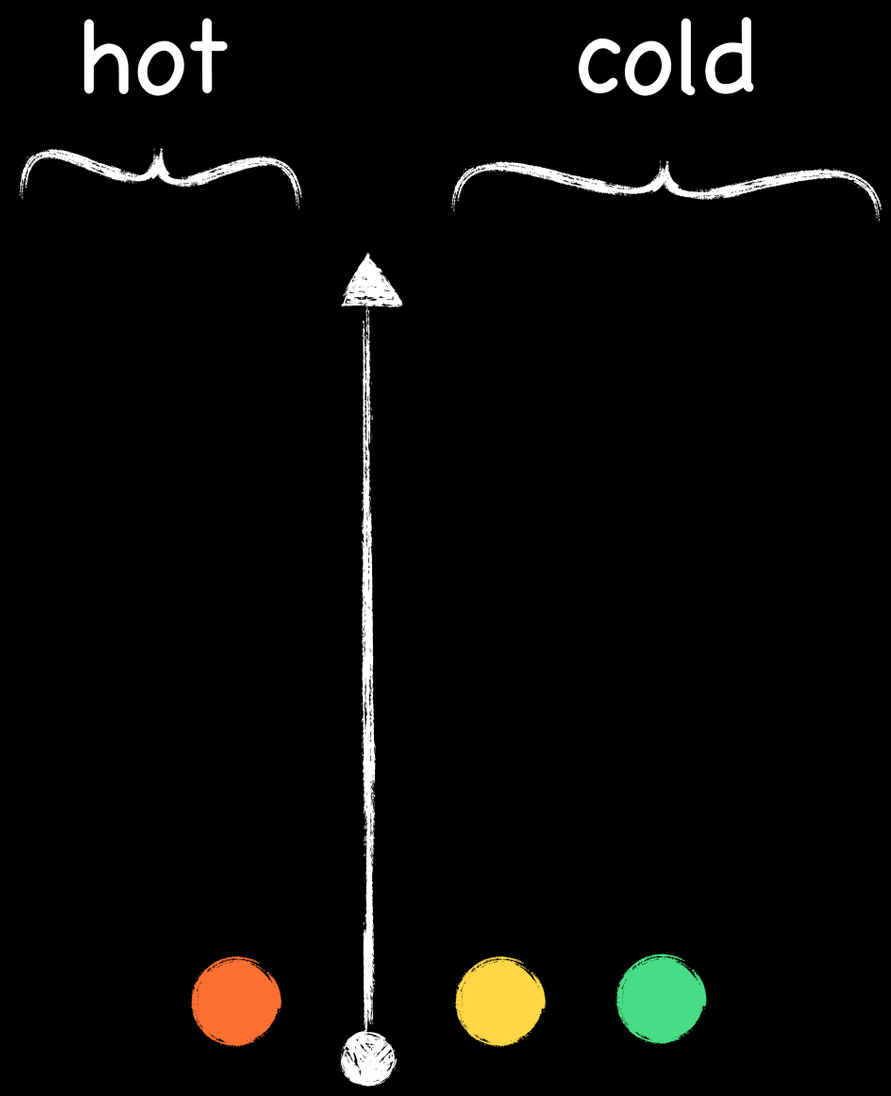
1st Iteration



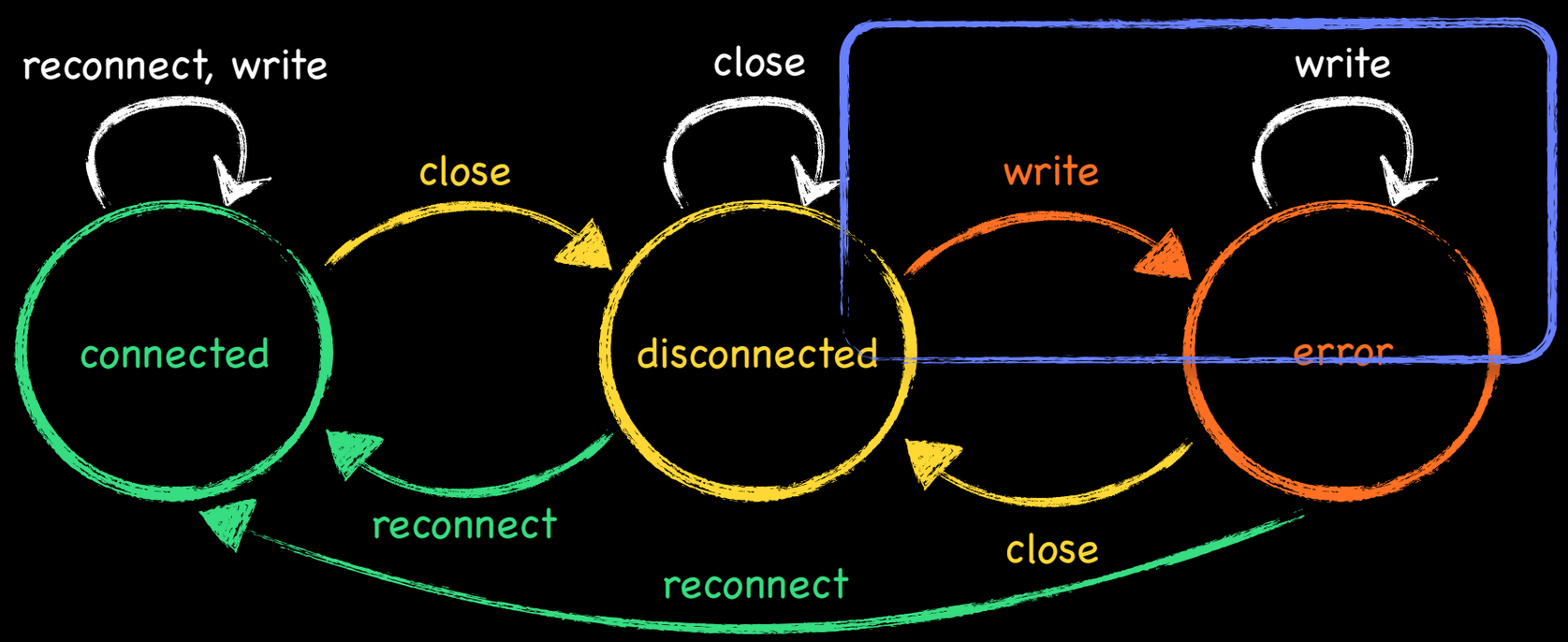

```
c.close()

c.reconnect()

c.write()
```



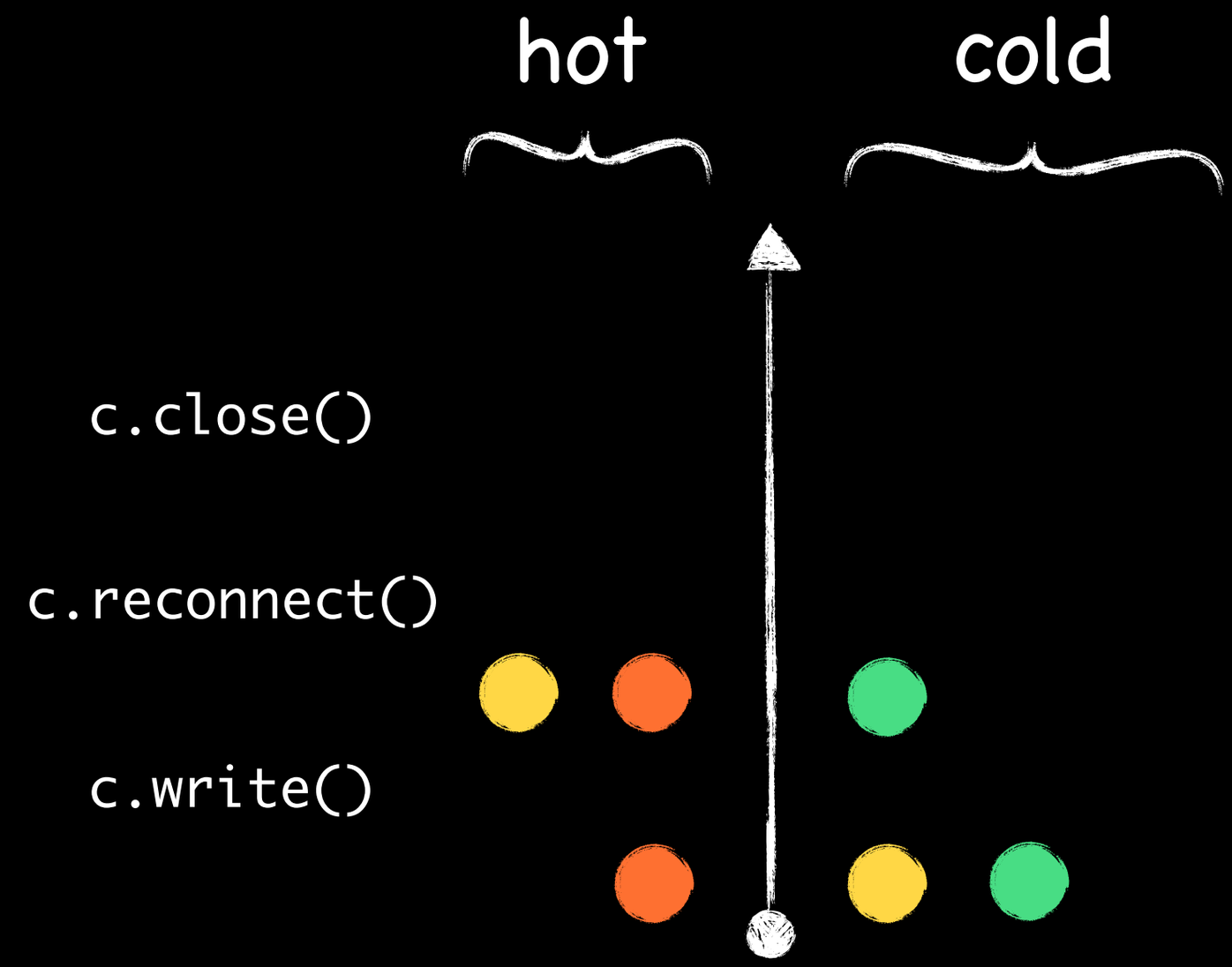
1st Iteration

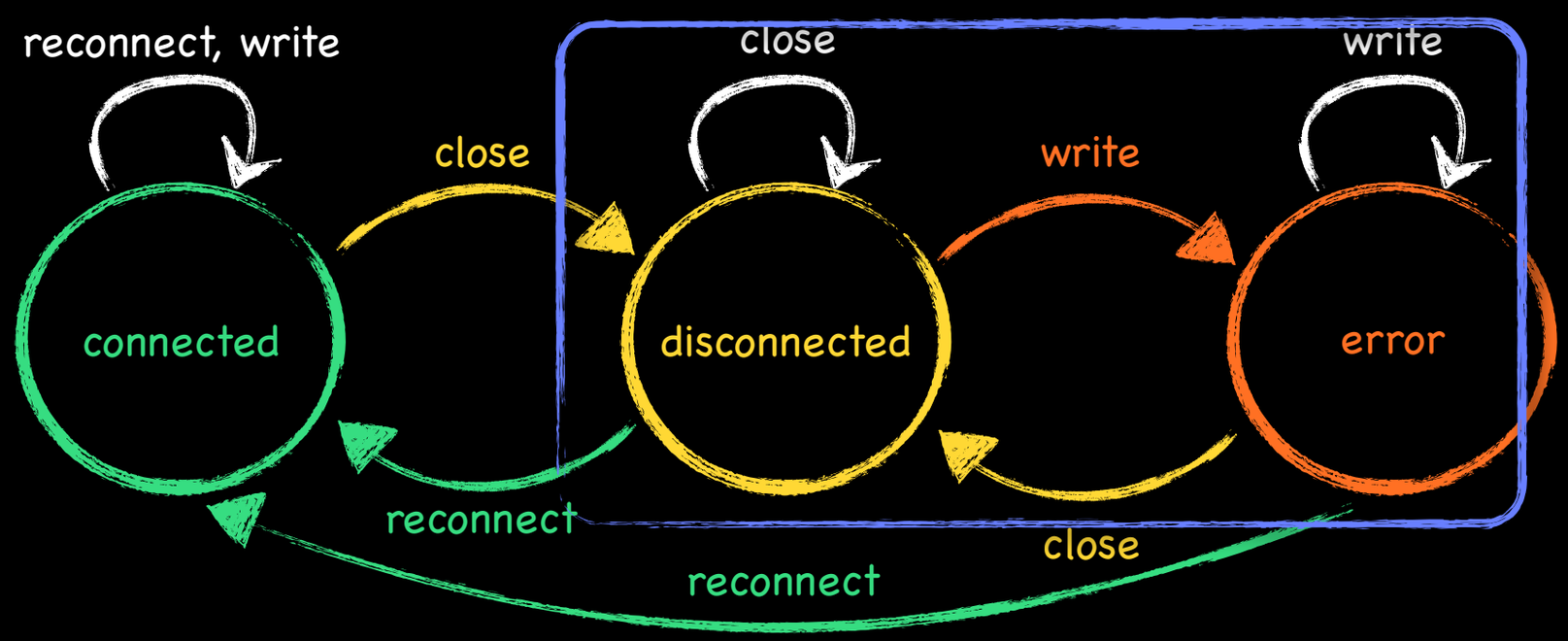



```
c.close()

c.reconnect()

c.write()
```

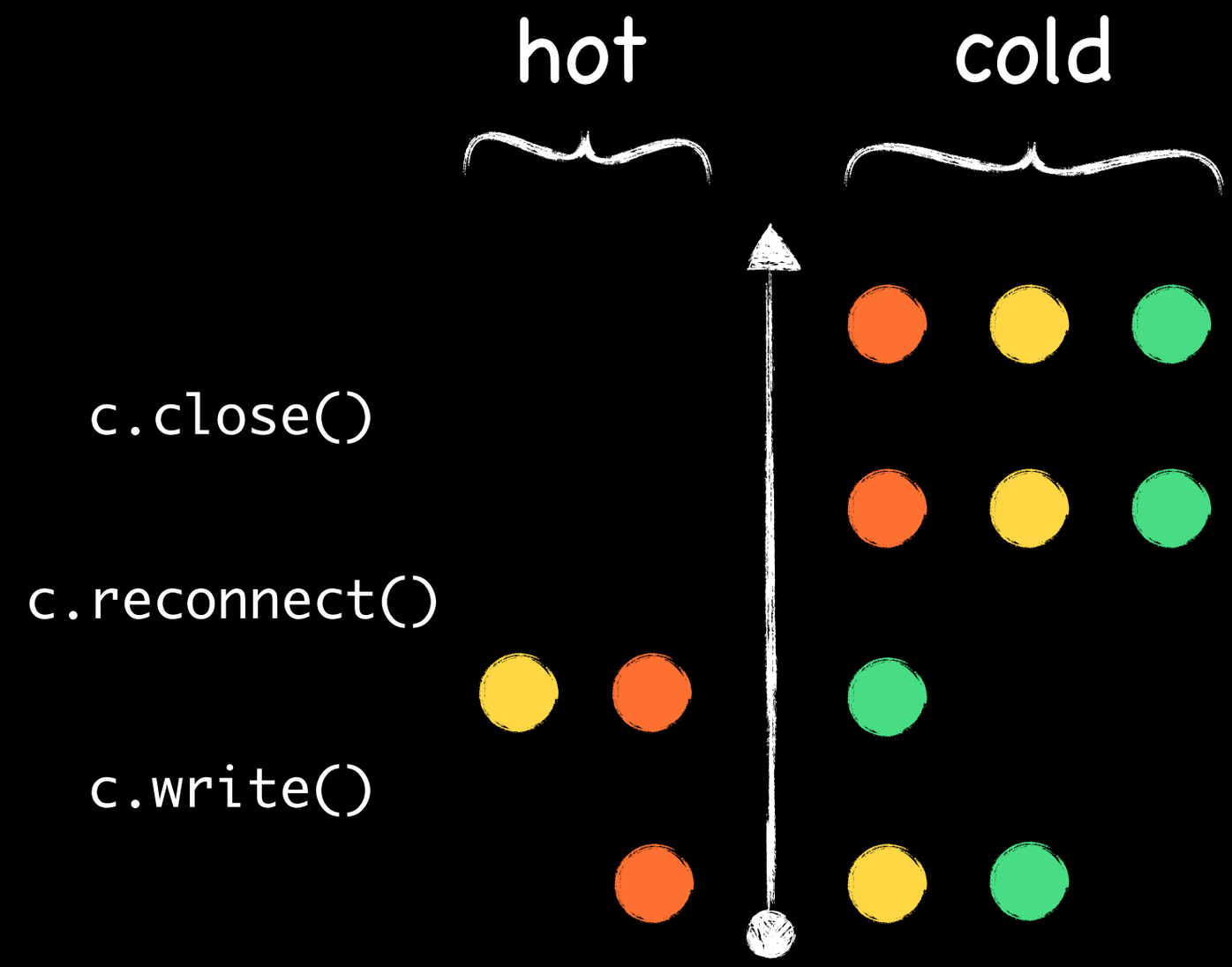


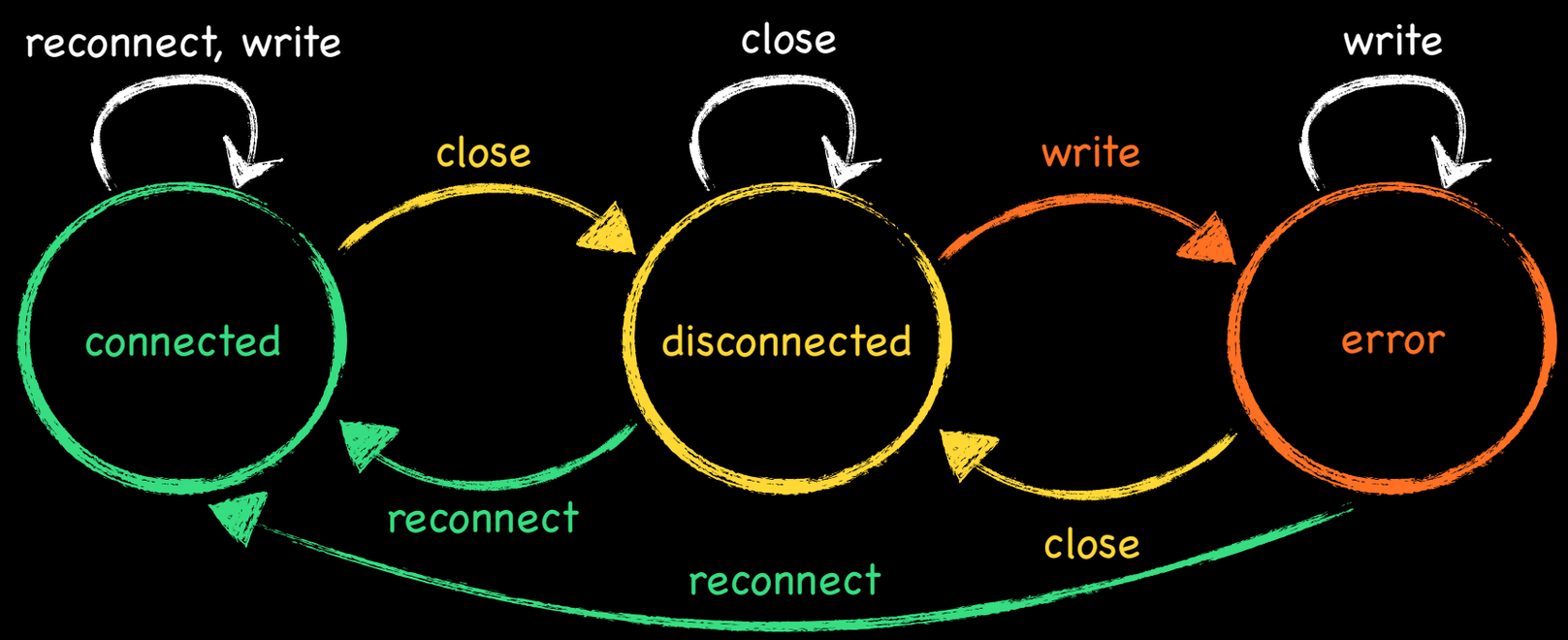
1st Iteration



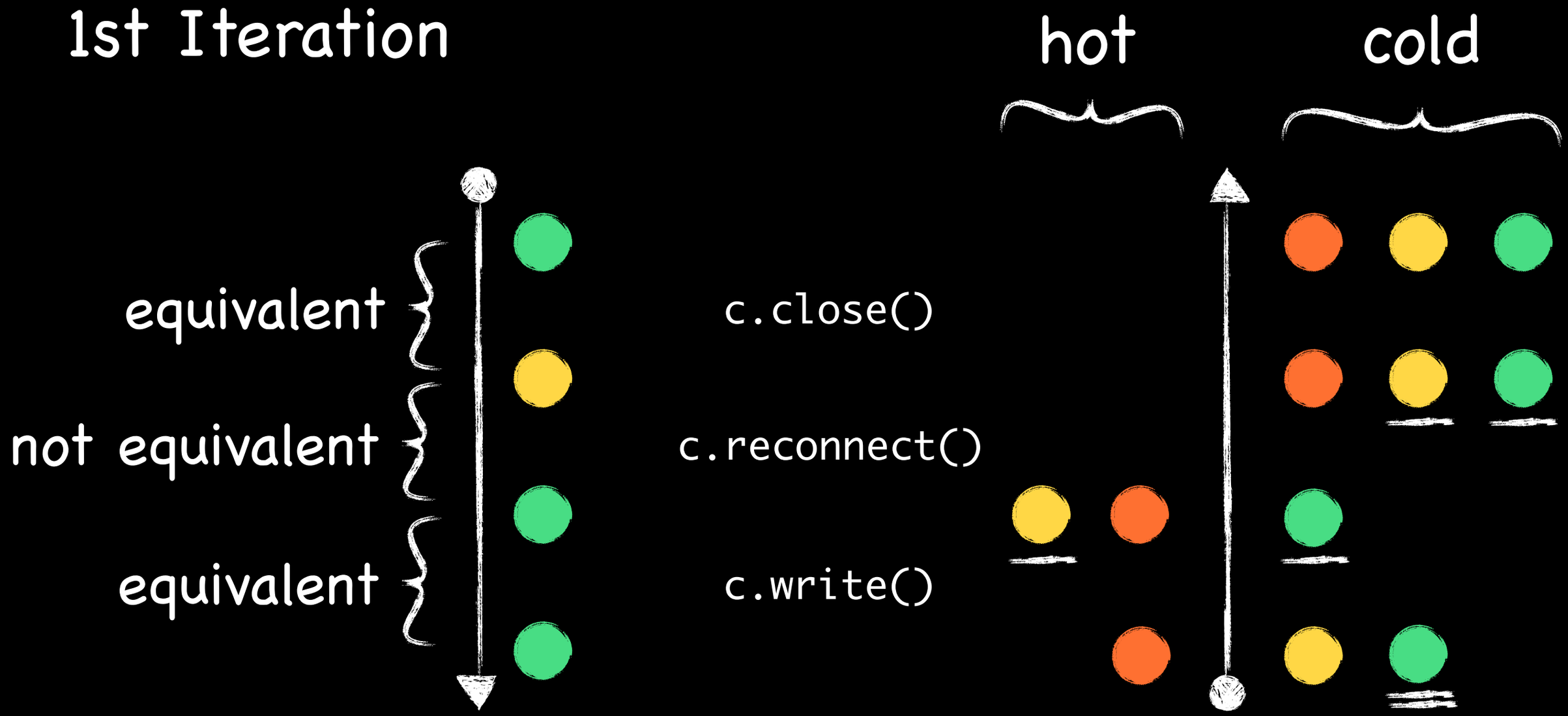


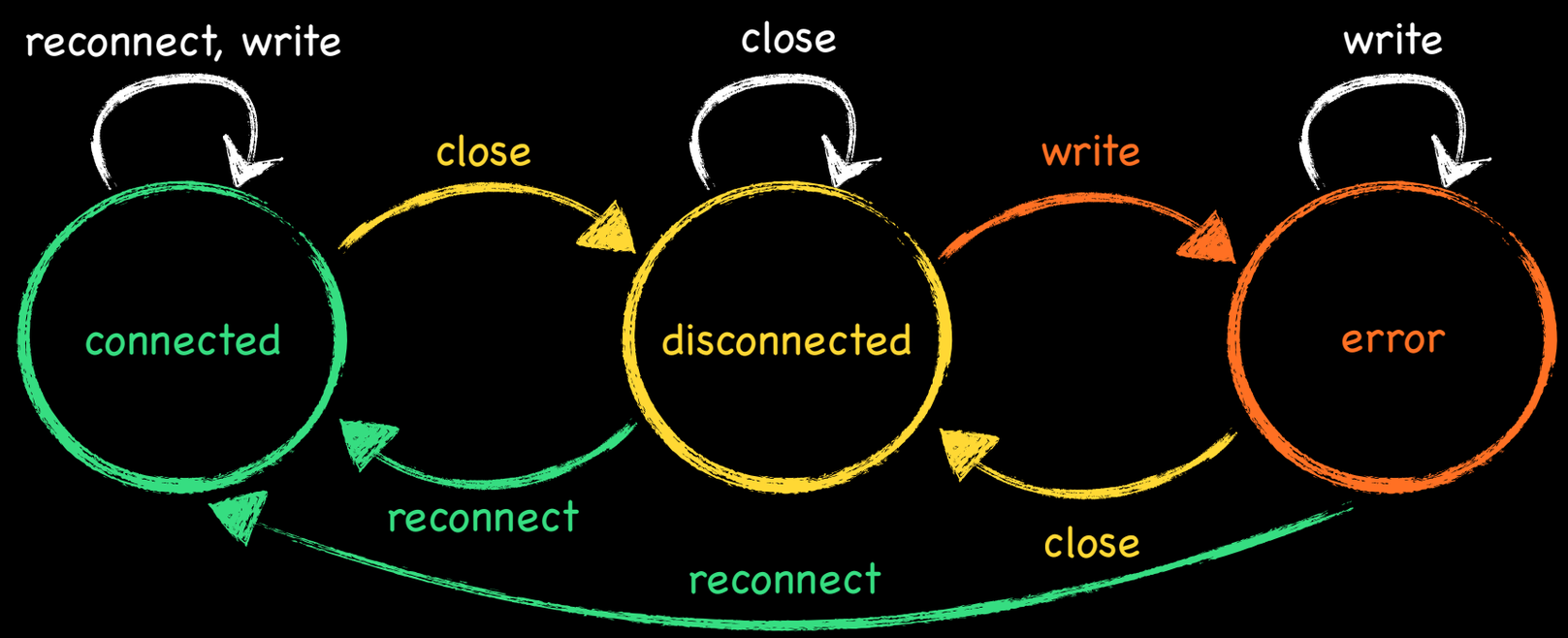
1st Iteration



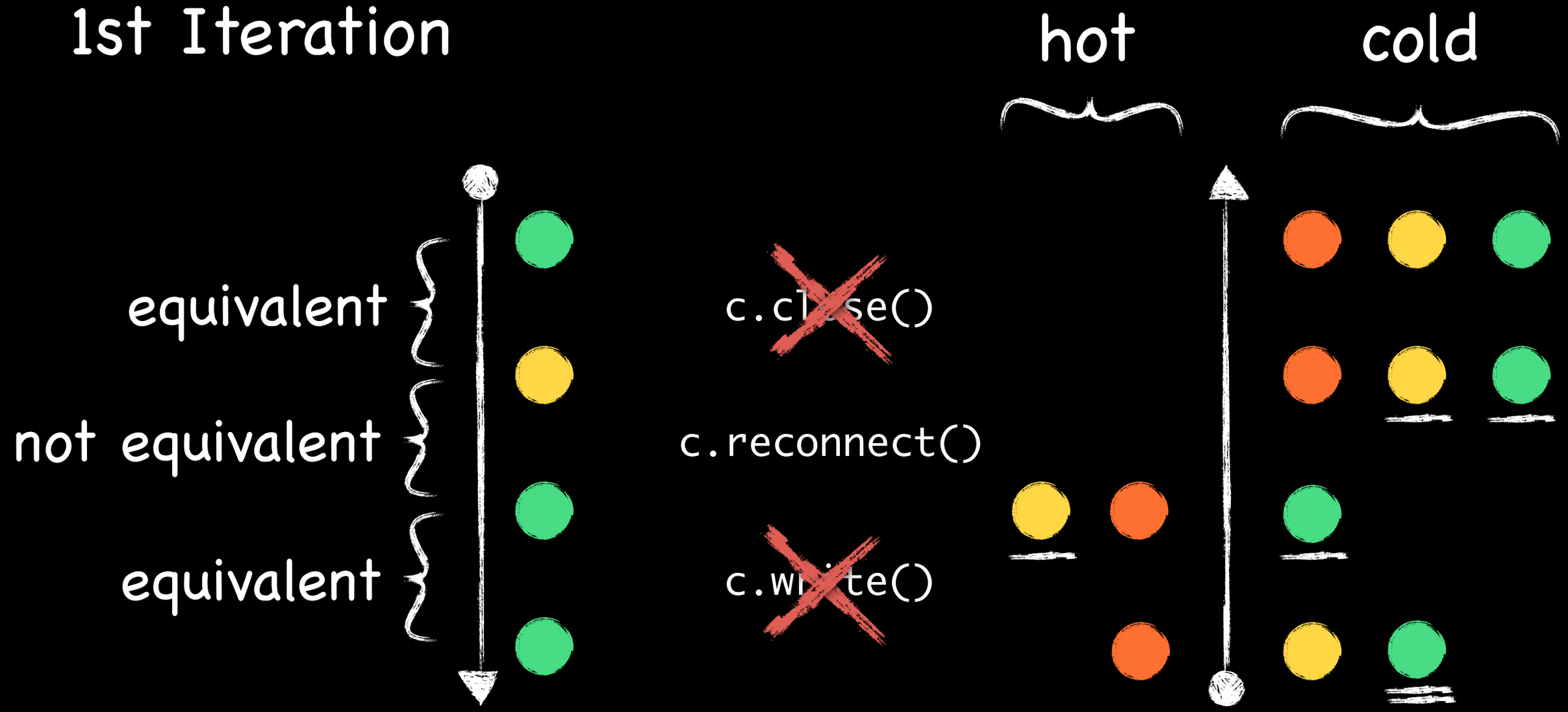


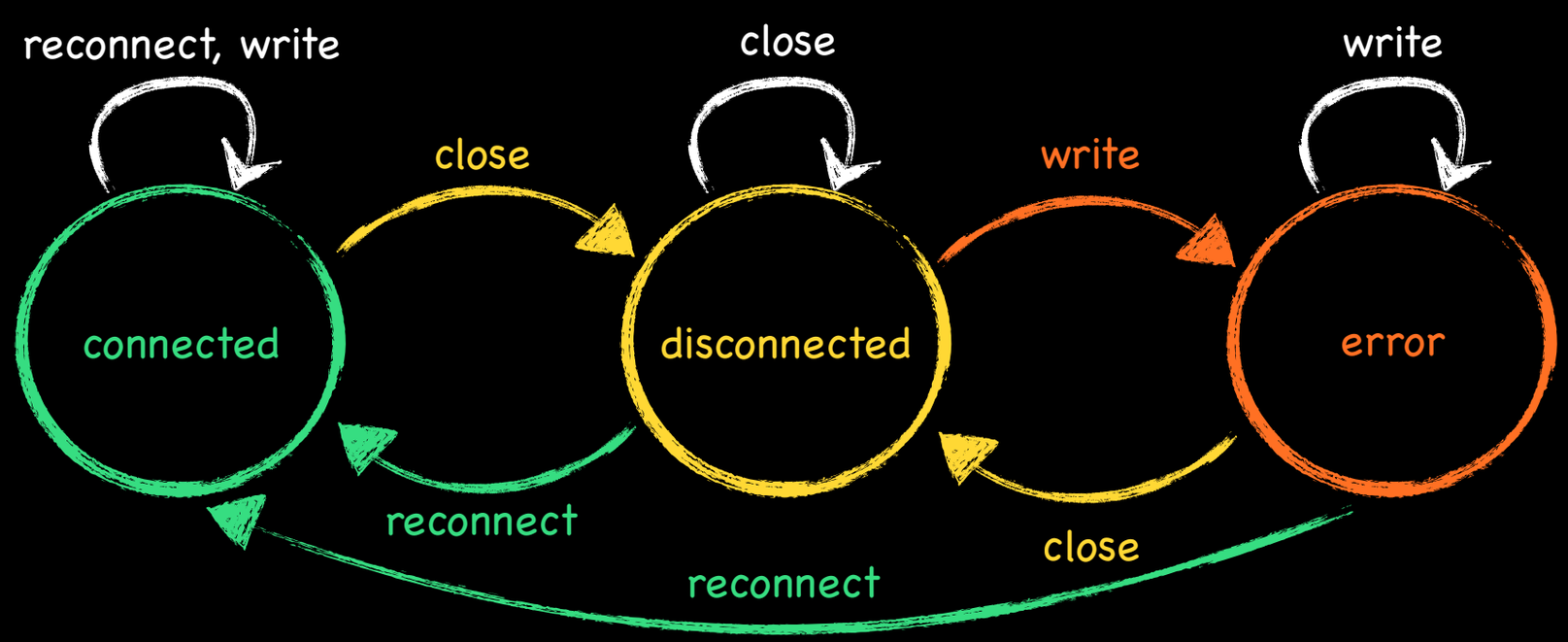
1st Iteration



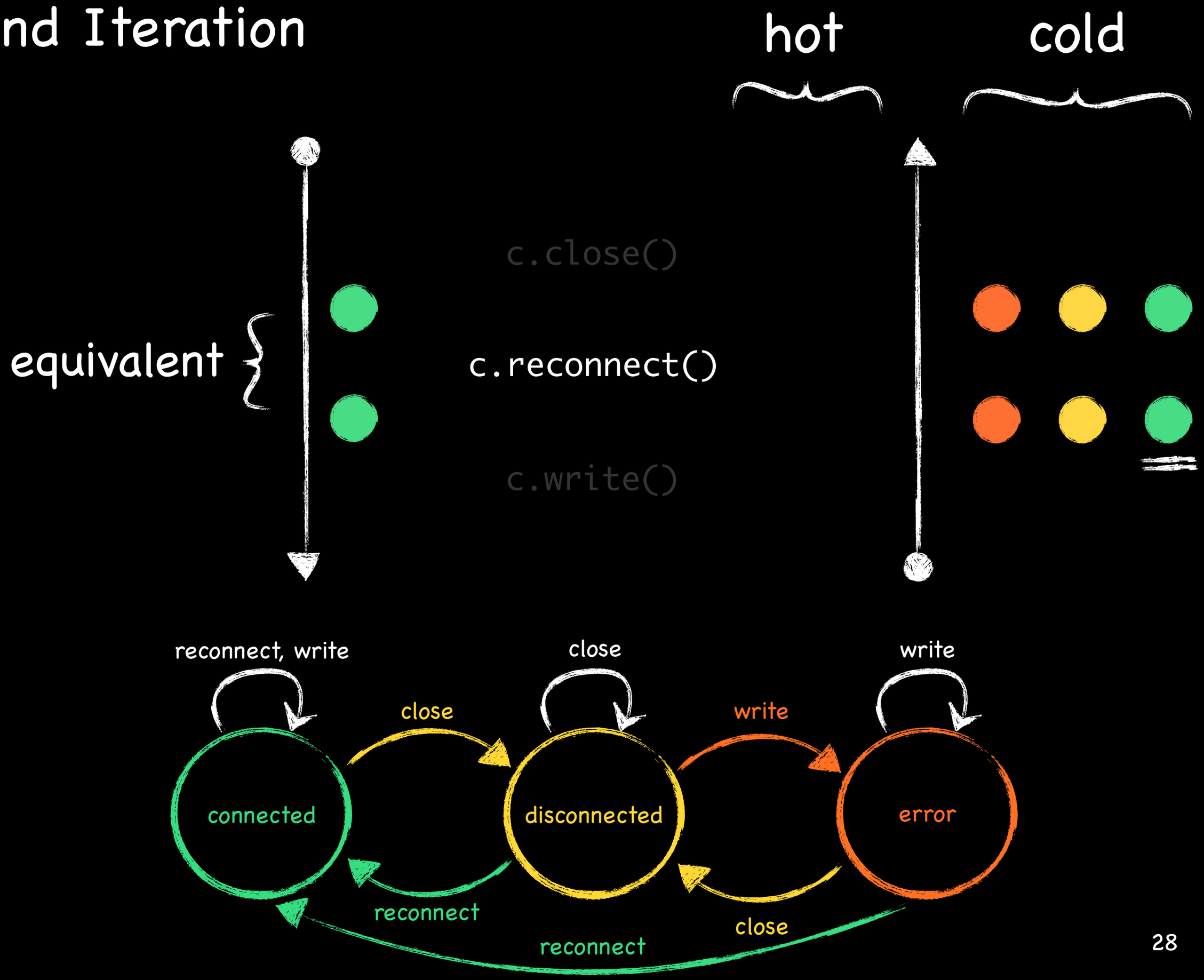


1st Iteration





2nd Iteration



2nd Iteration