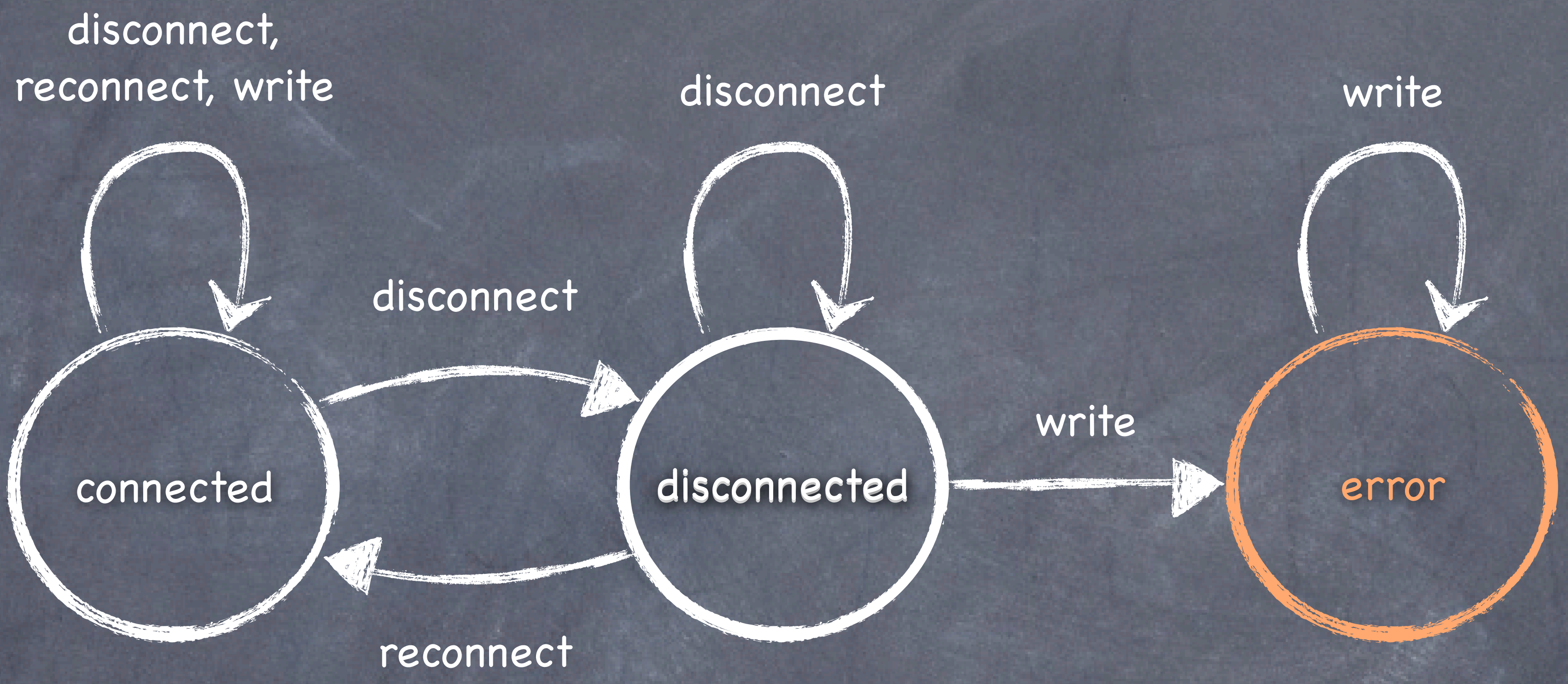
Verifying finite-state properties of large-scale programs

Eric Bodden

Finite-state properties

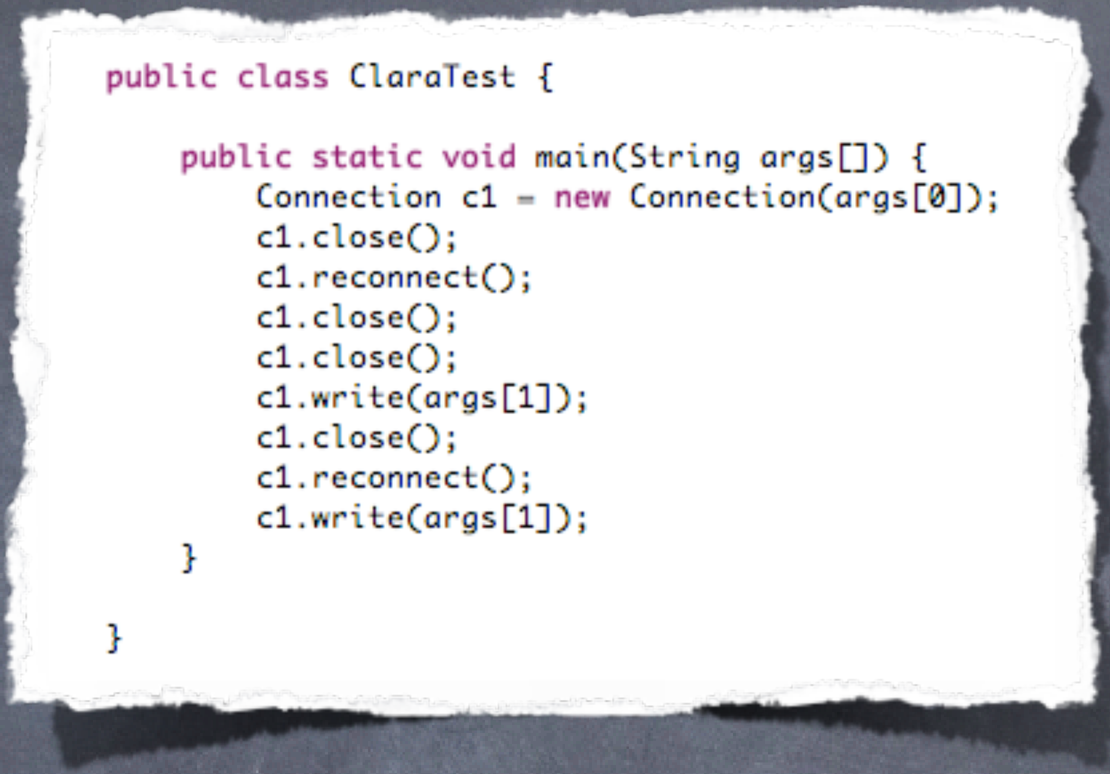
“When disconnecting a connection c, don’t write to c until c is reconnected.”

Finite-state properties



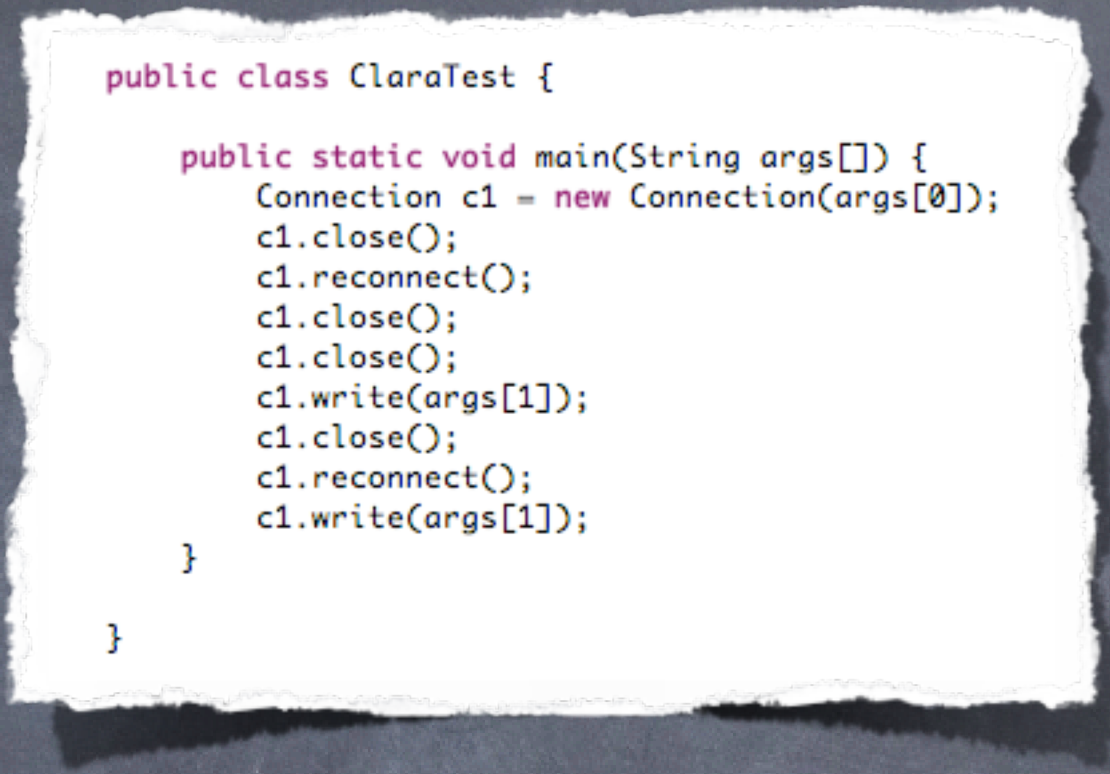
"When disconnecting a connection c, don't write to c until c is reconnected."

Runtime-verifying finite-state properties

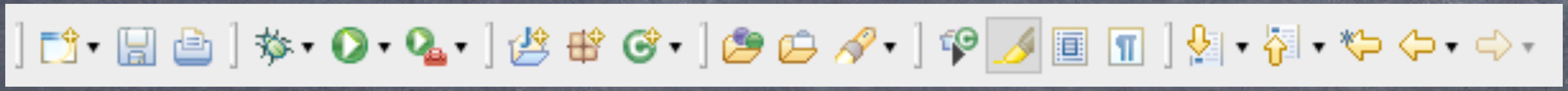


```
public class ClaraTest {

    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]);
    }

}
```

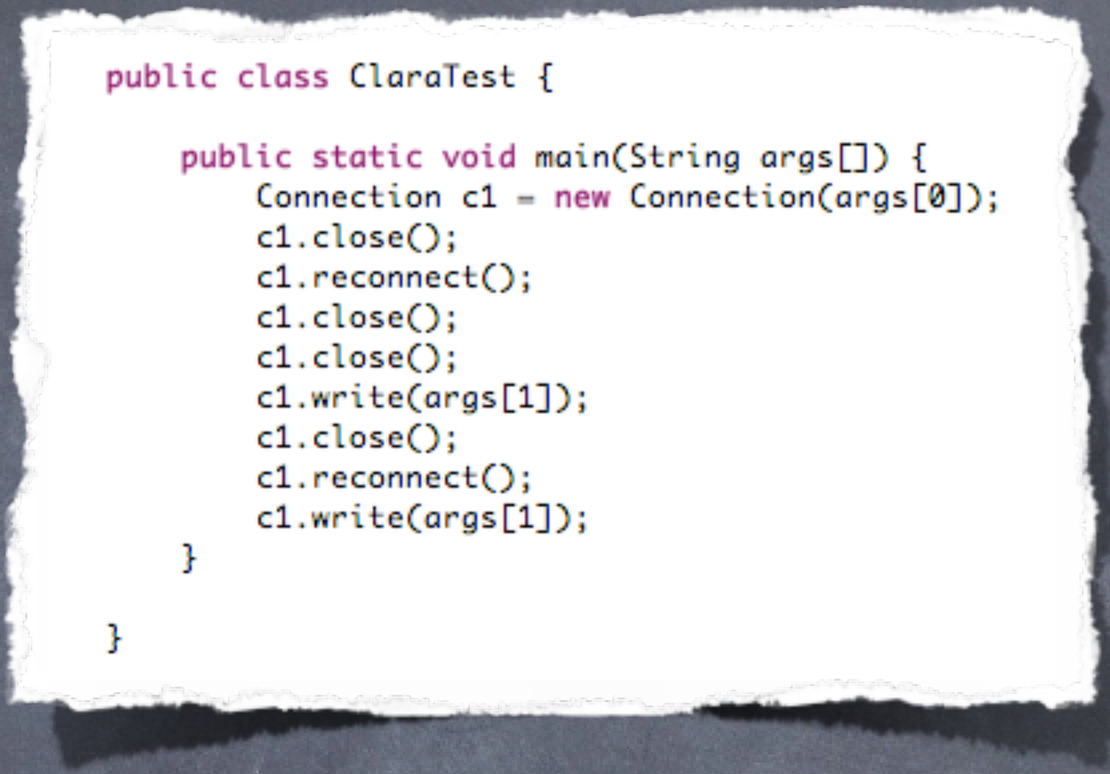
Runtime-verifying finite-state properties



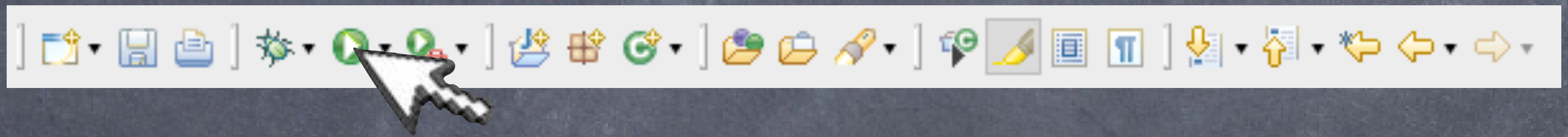
```
public class ClaraTest {

    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]);
    }

}
```



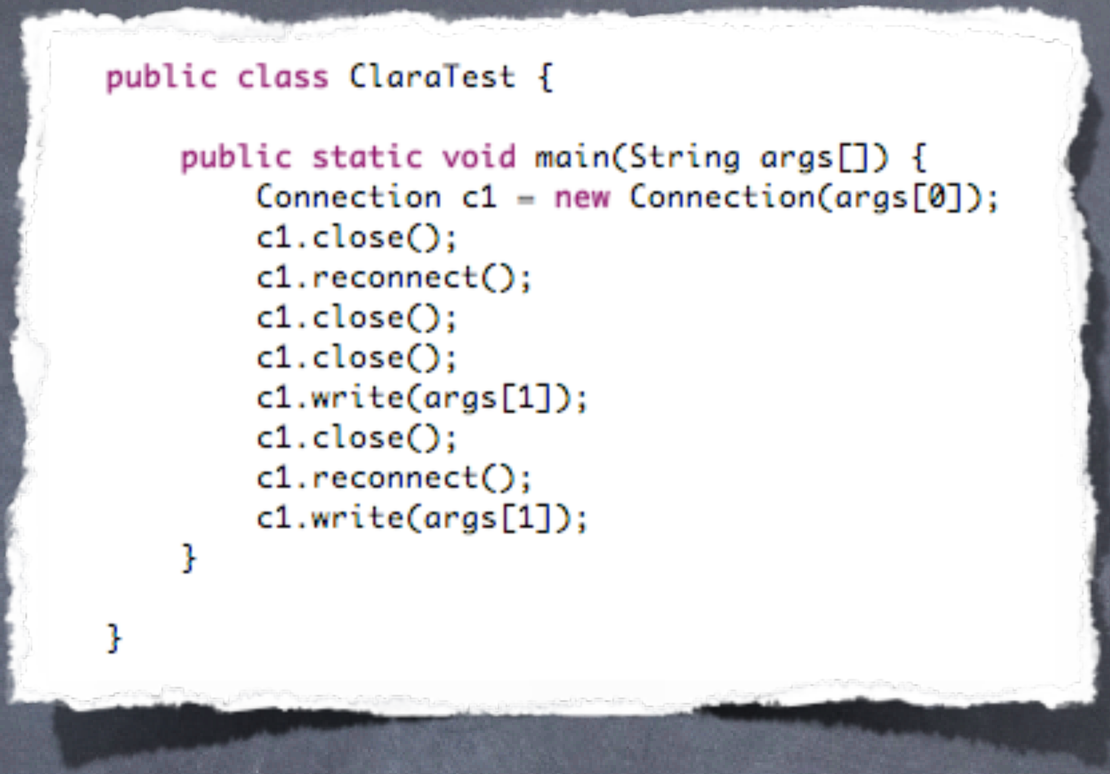
Runtime-verifying finite-state properties



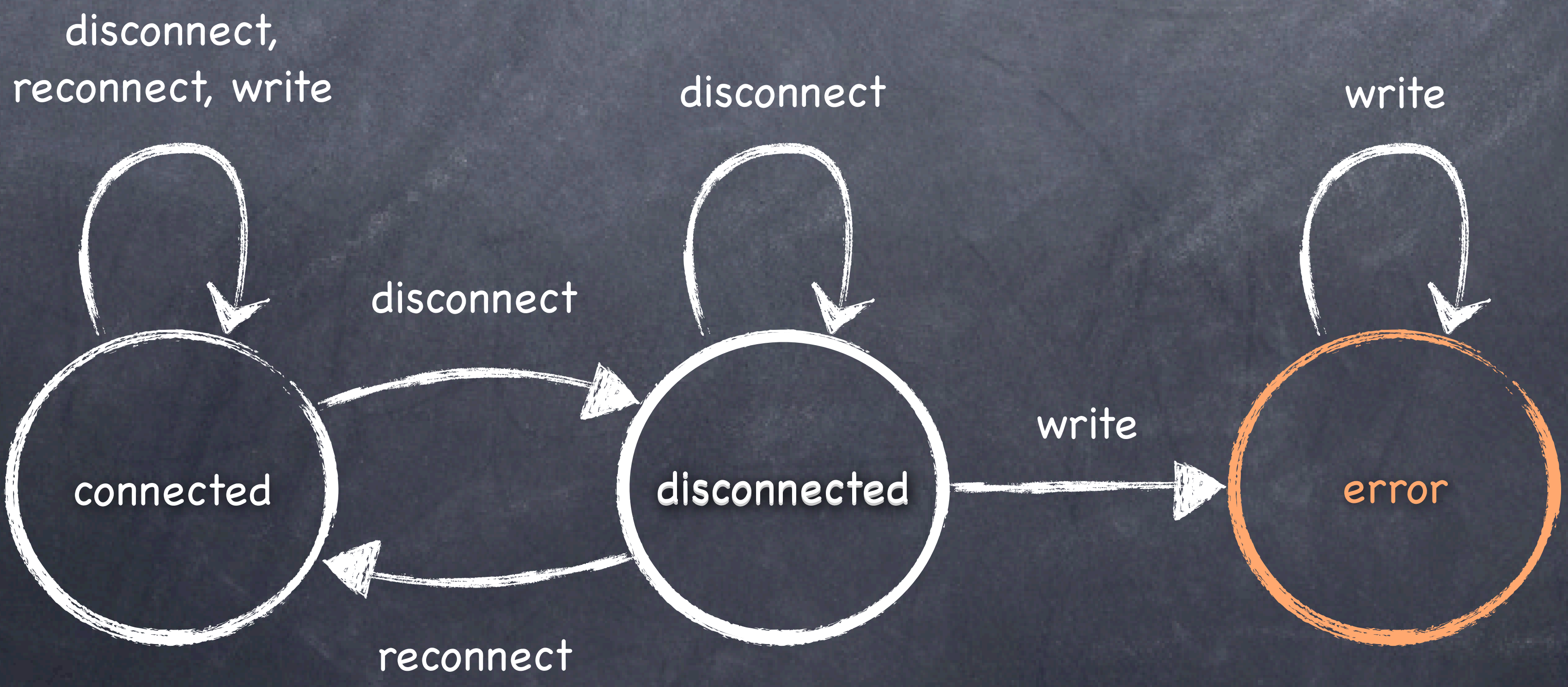
```
public class ClaraTest {

    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]);
    }

}
```



Runtime-verifying finite-state properties

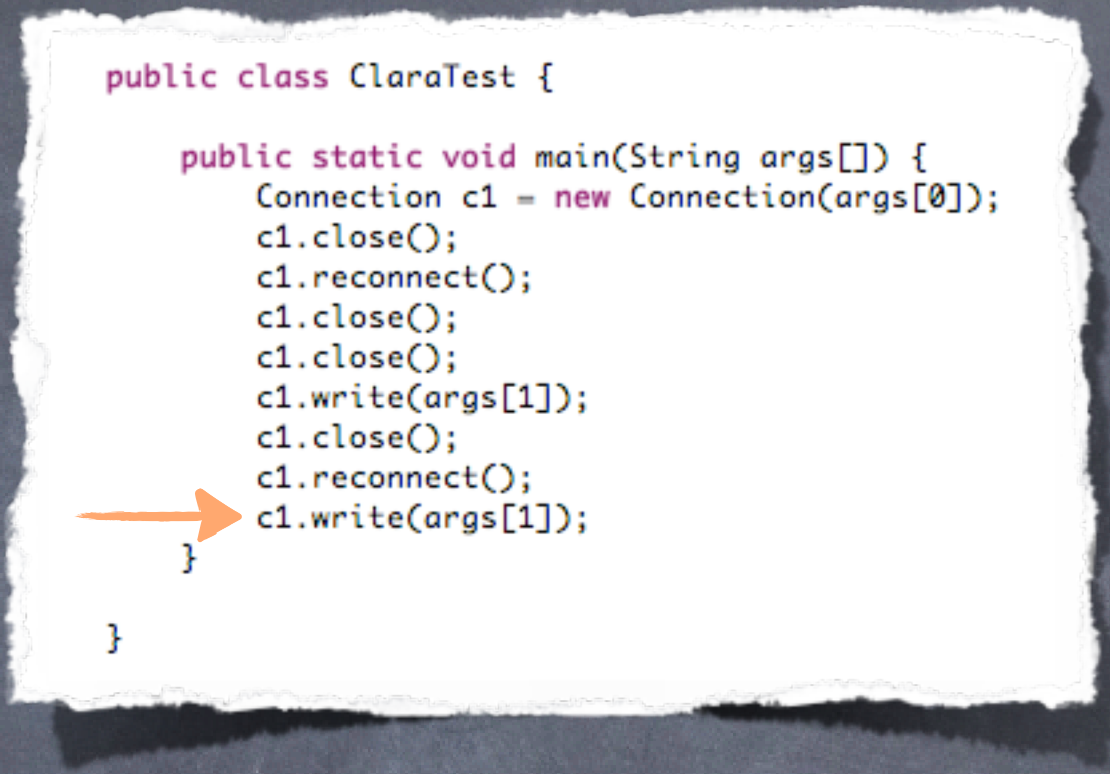


```
public class ClaraTest {

    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]);
    }

}
```

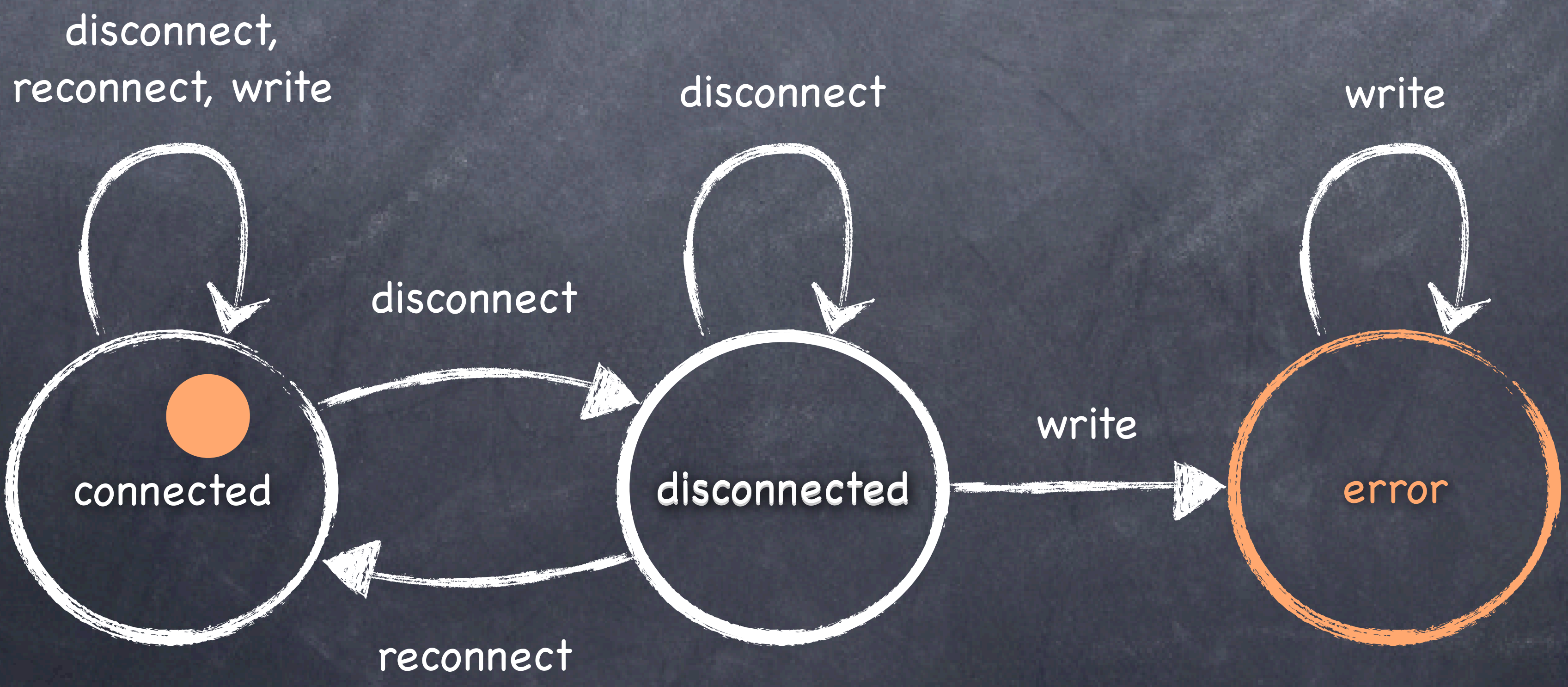


$$\begin{tikzcd}
\text{connected} \arrow[loop above, "\text{disconnect, reconnect, write}"] \arrow[r, bend left, "\text{disconnect}"] & \text{disconnected} \arrow[loop above, "\text{disconnect}"] \arrow[l, bend left, "\text{reconnect}"] \arrow[r, "\text{write}"] & \text{error} \arrow[loop above, "\text{write}"]
\end{tikzcd}$$

Runtime-verifying finite-state properties



```
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]);
    }

}
```



$$\begin{tikzcd}
\text{connected} \arrow[loop above, "\text{disconnect, reconnect, write}"] \arrow[r, bend left, "\text{disconnect}"] & \text{disconnected} \arrow[loop above, "\text{disconnect}"] \arrow[l, bend left, "\text{reconnect}"] \arrow[r, "\text{write}"] & \text{error} \arrow[loop above, "\text{write}"]
\end{tikzcd}$$

Runtime-verifying finite-state properties

JavaMOP

J-LO

abc

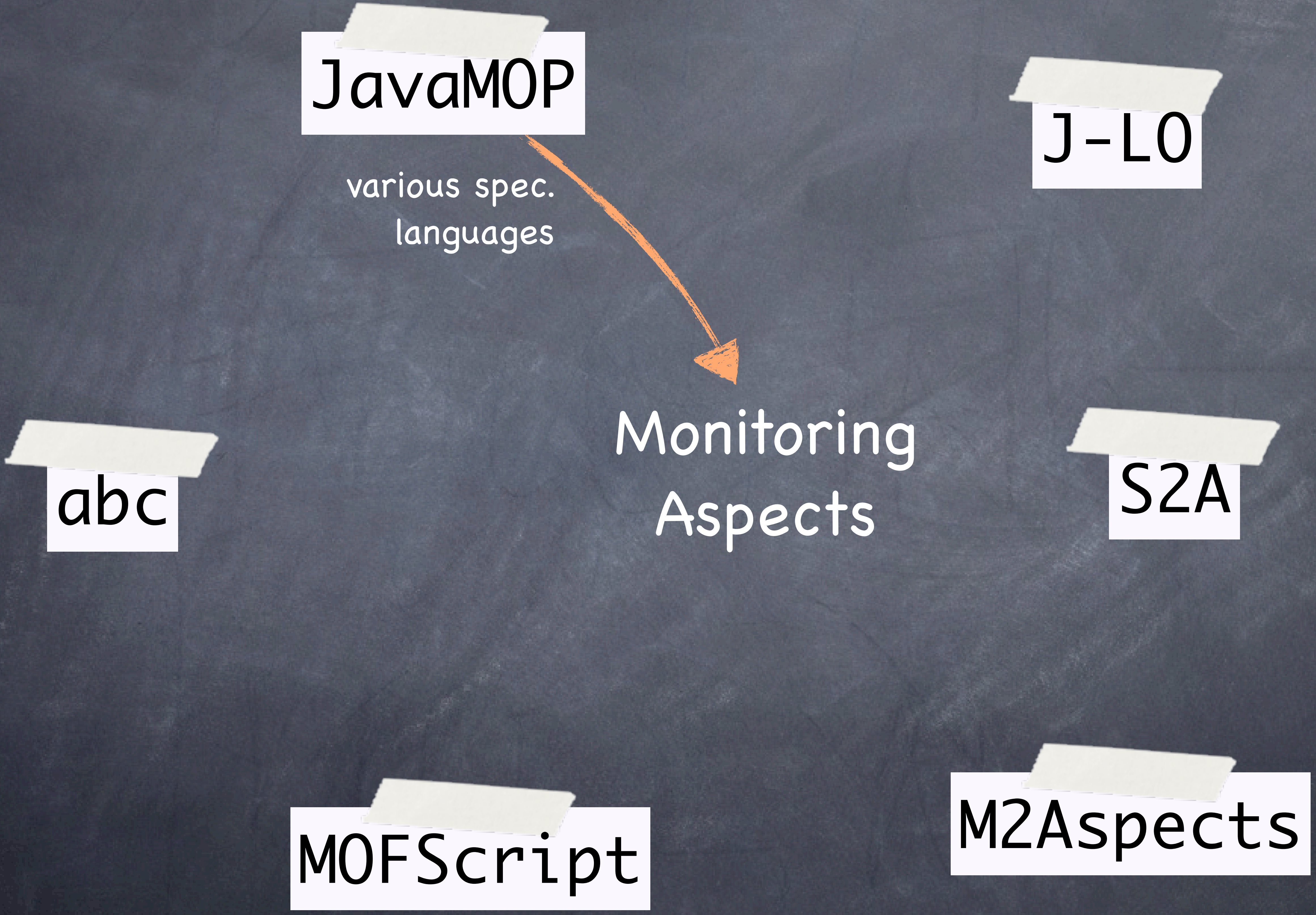
Monitoring Aspects

S2A

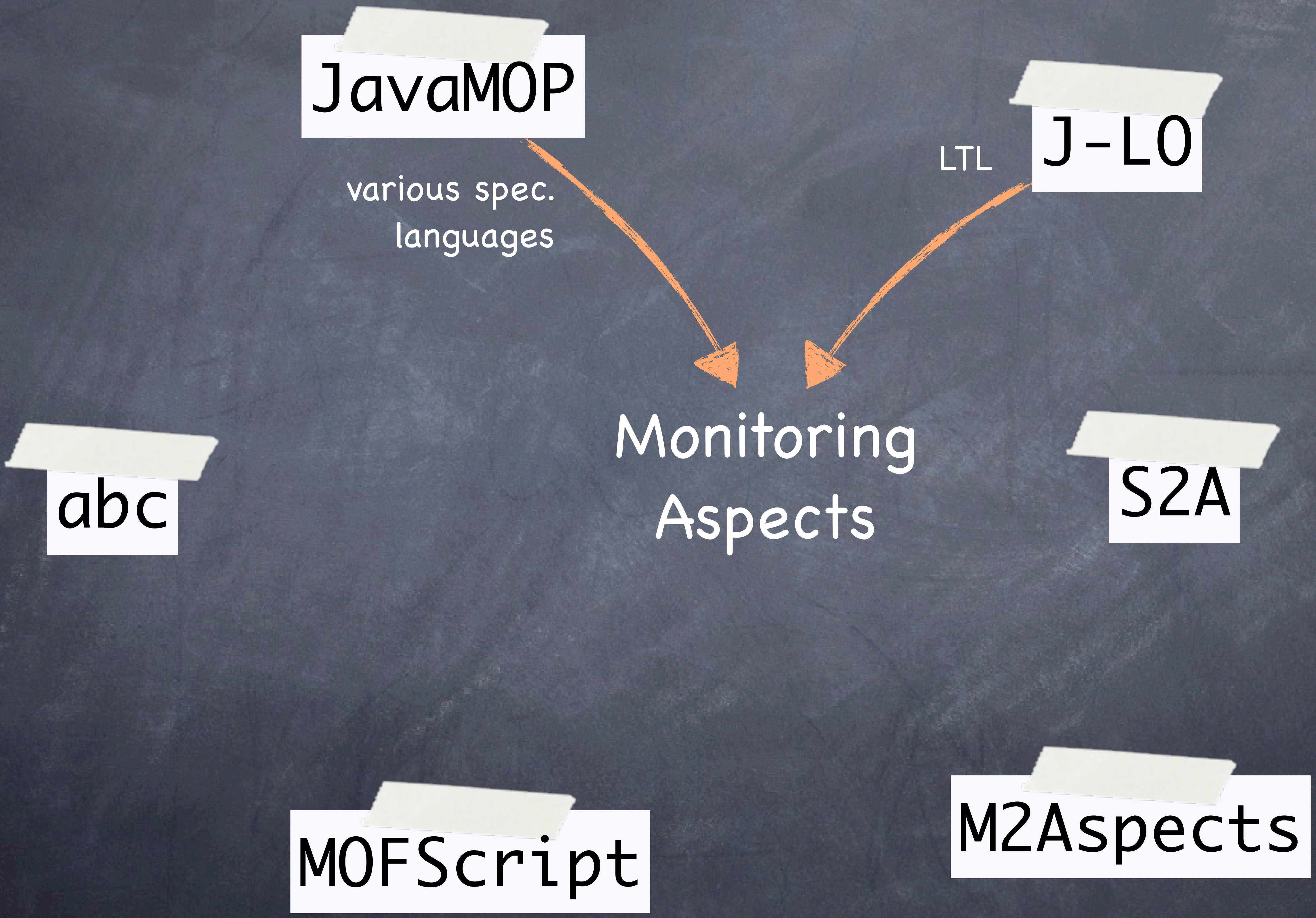
MOFScript

M2Aspects

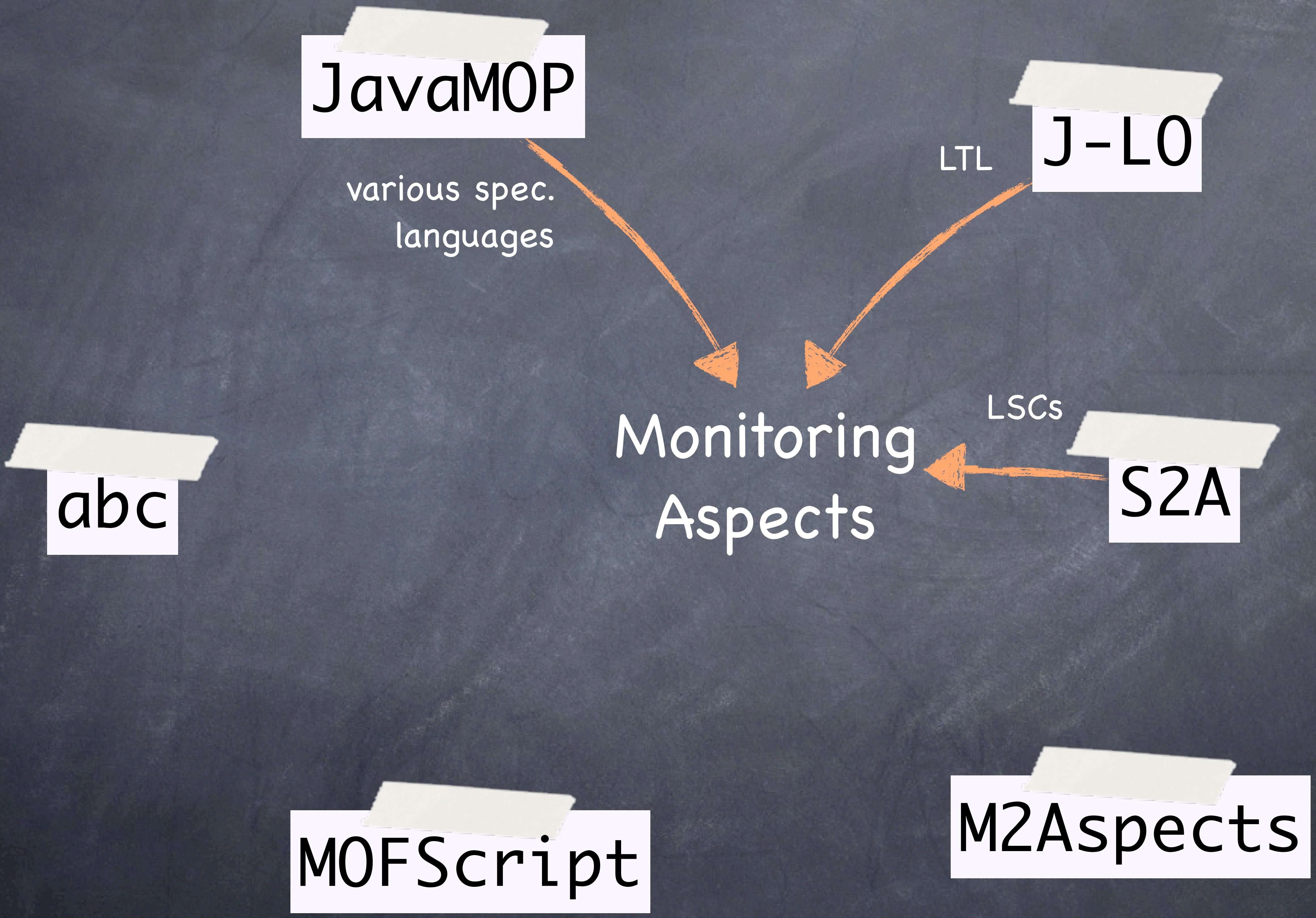
Runtime-verifying finite-state properties



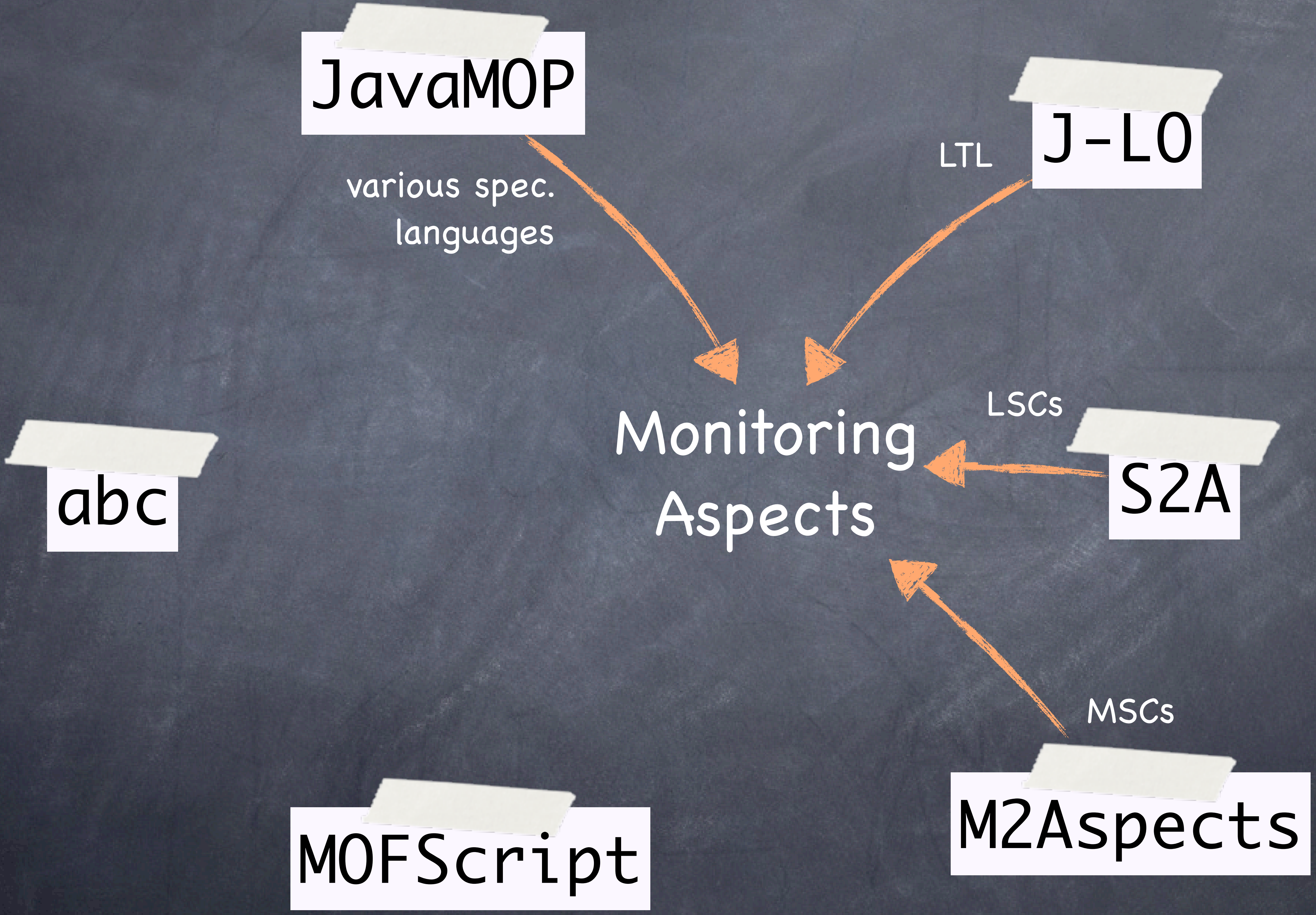
Runtime-verifying finite-state properties



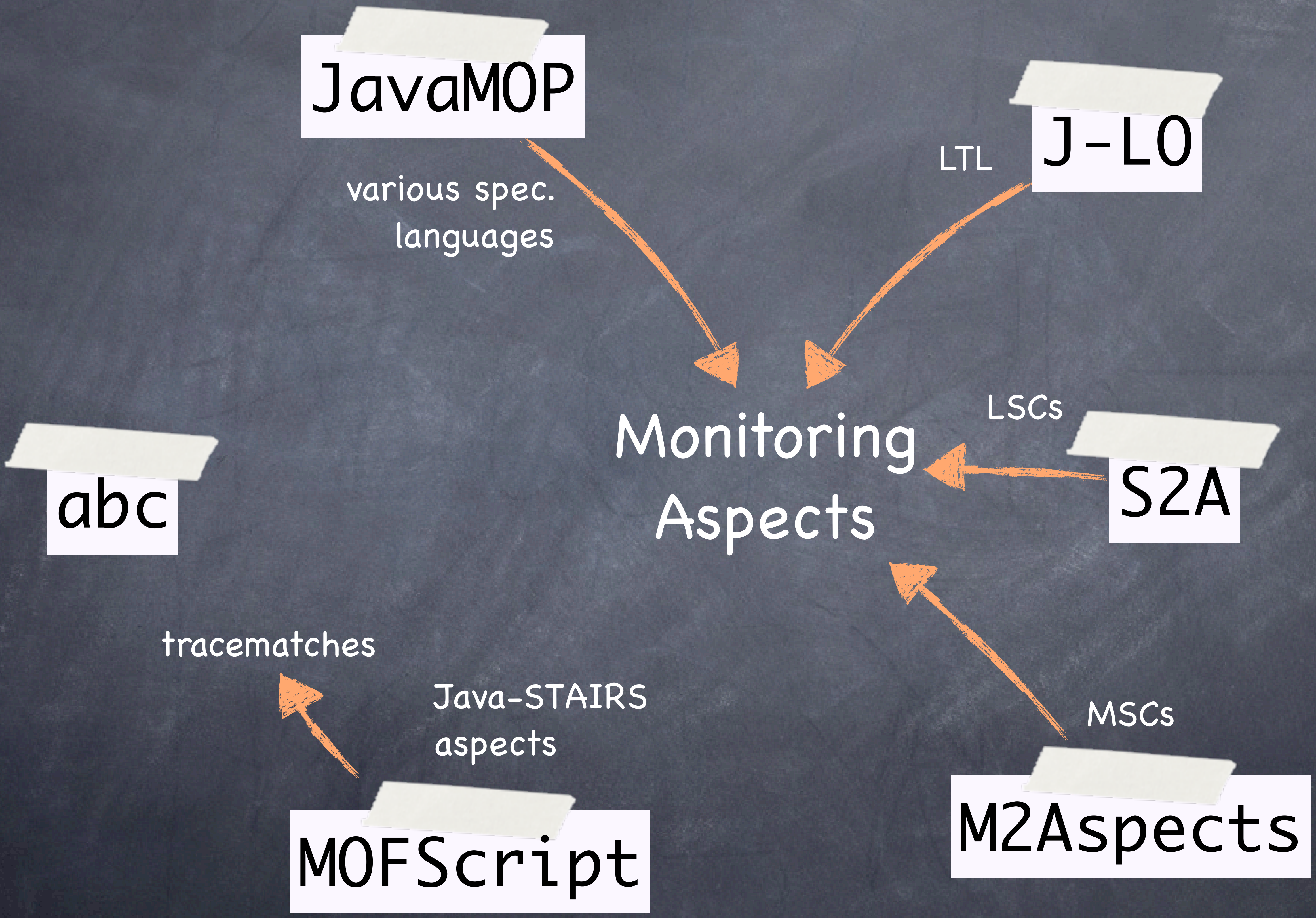
Runtime-verifying finite-state properties



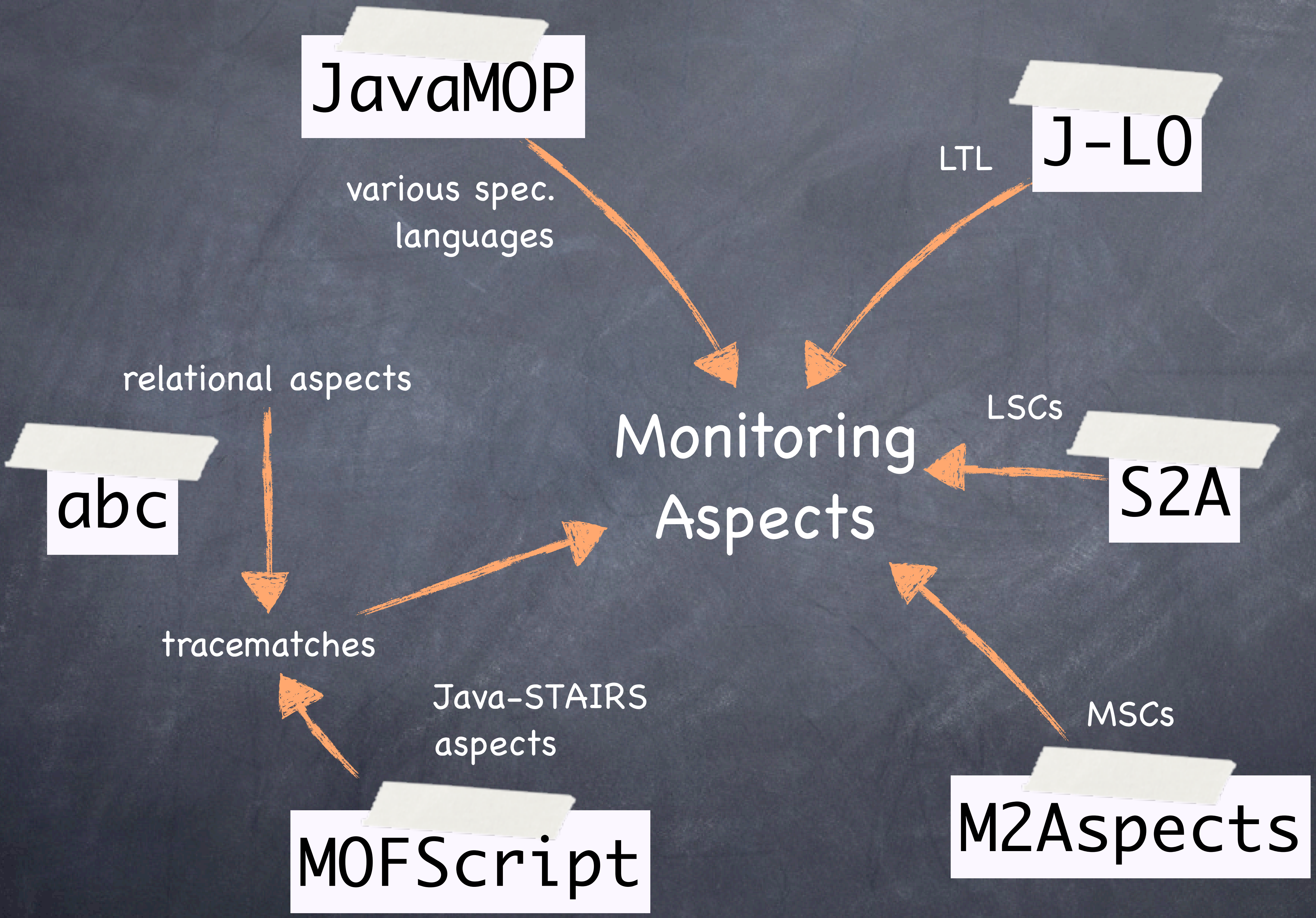
Runtime-verifying finite-state properties



Runtime-verifying finite-state properties



Runtime-verifying finite-state properties



Runtime-verifying finite-state properties

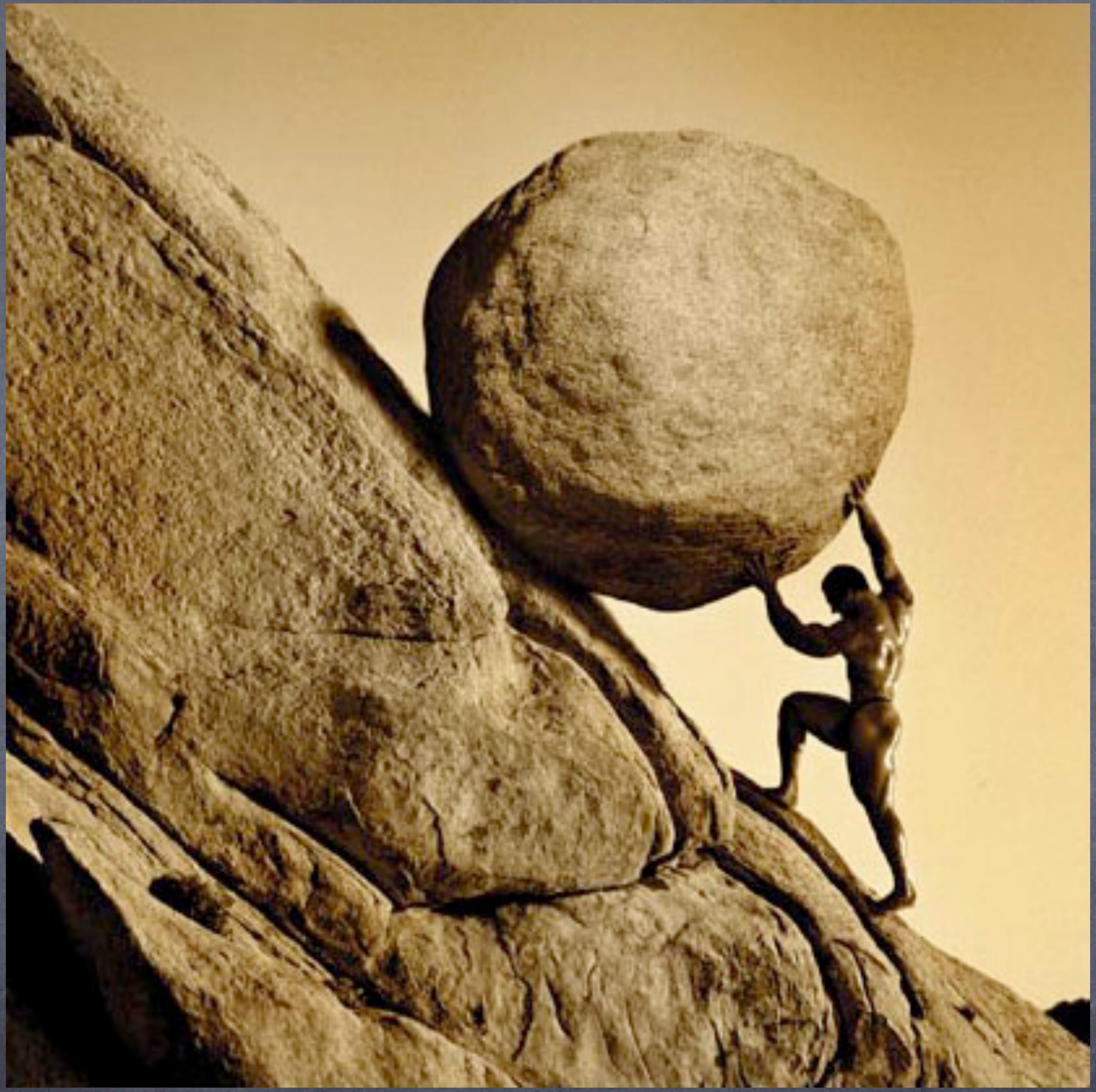
Runtime-verifying finite-state properties

No static guarantees

Runtime-verifying finite-state properties

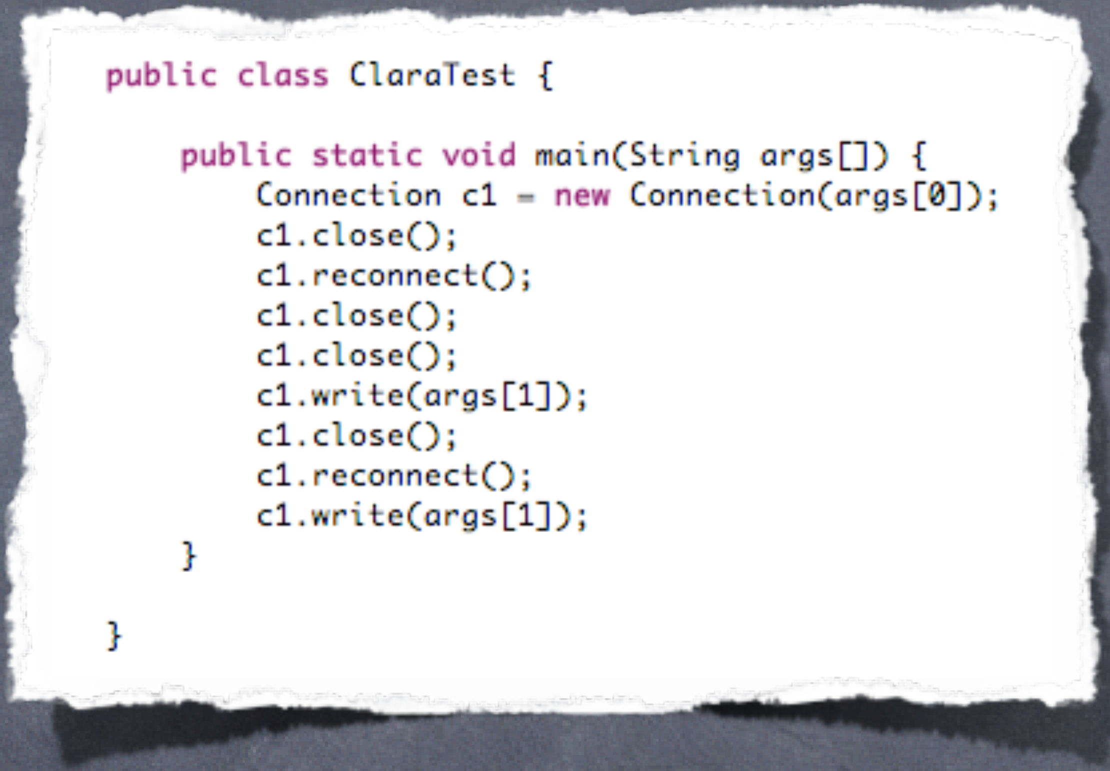
Potentially large runtime overhead

Runtime-verifying finite-state properties



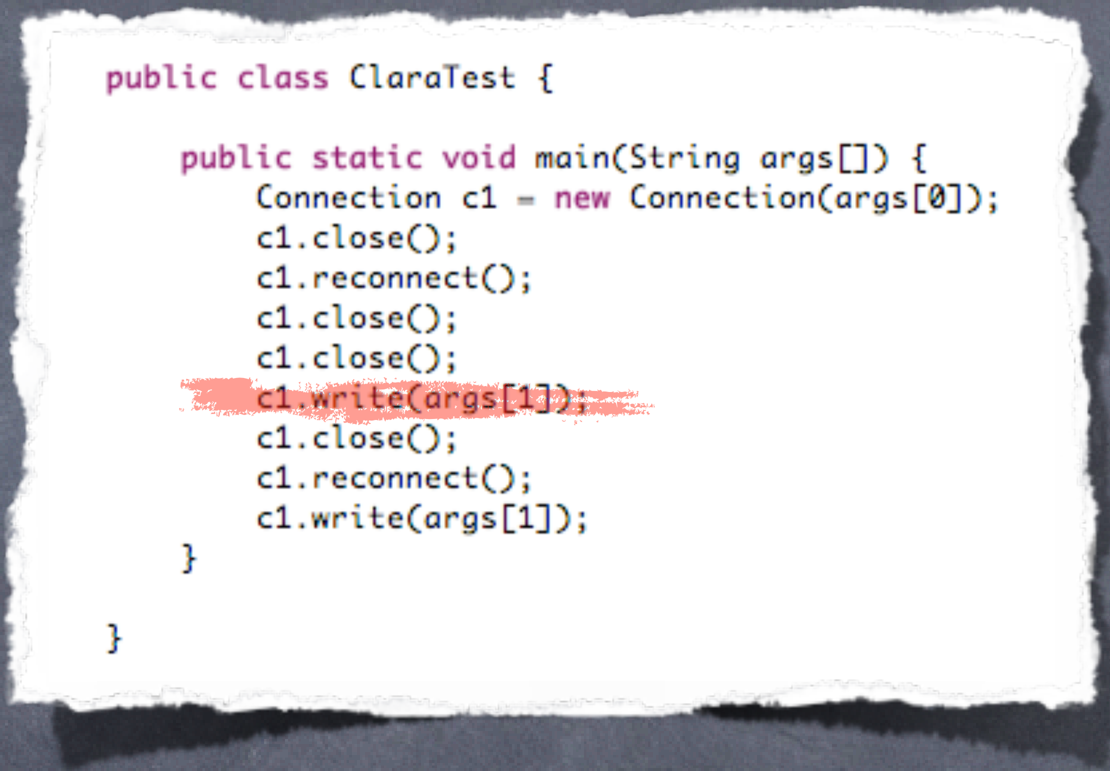
When to finish testing?

Static analysis: the solution?



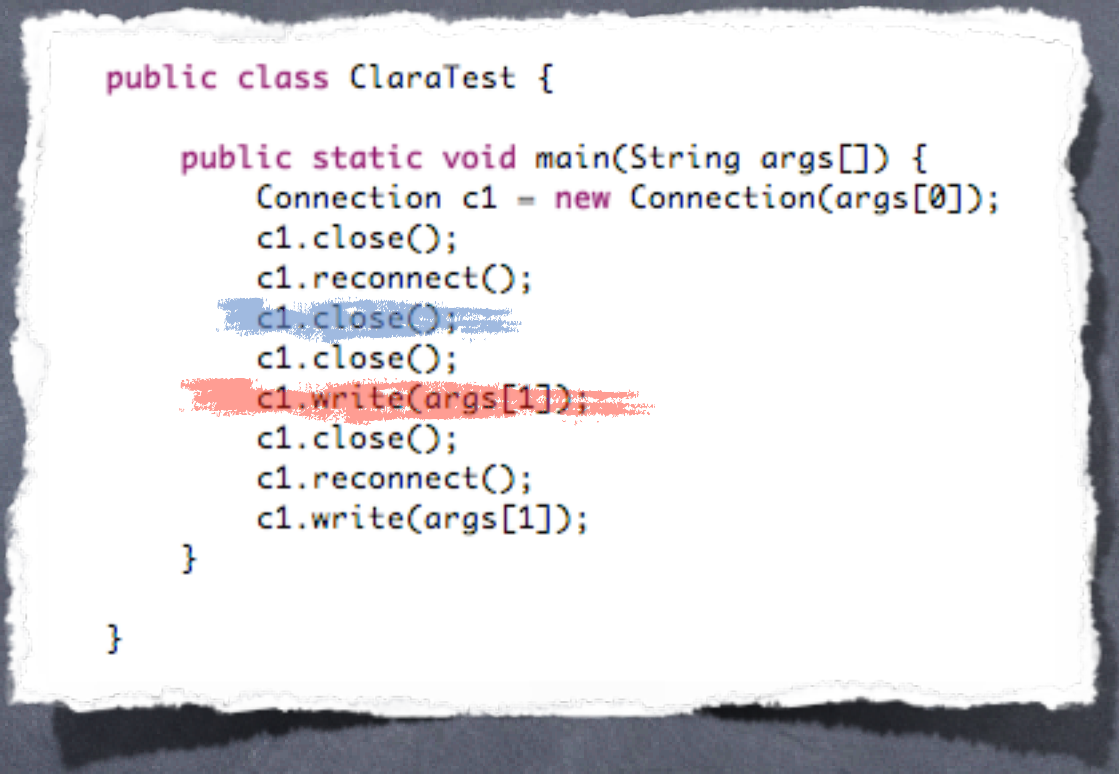
```
public class ClaraTest {

    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]);
    }

}
```

Static analysis: the solution?



```
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]);
    }

}
```

Static analysis: the solution?



```
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]);
    }

}
```

The Clara Framework

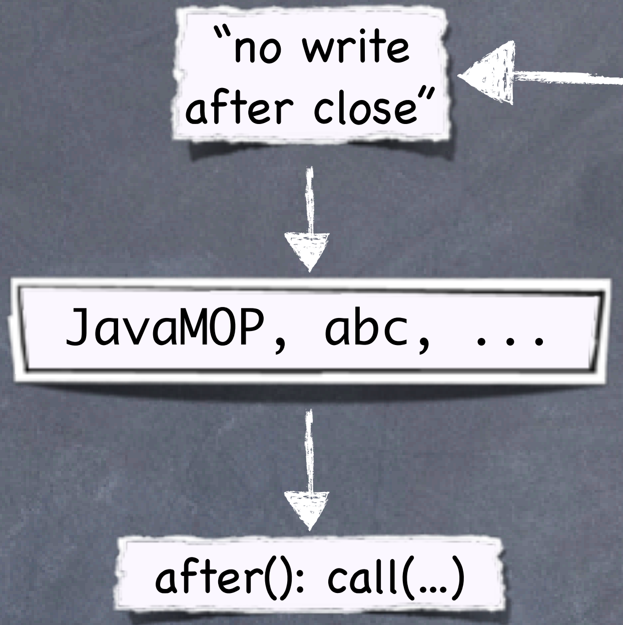
The Clara Framework

“no write after close”

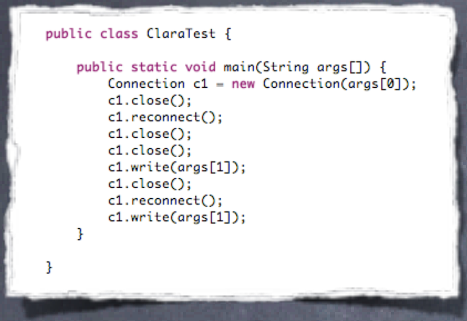
The Clara Framework

“no write after close”

The Clara Framework



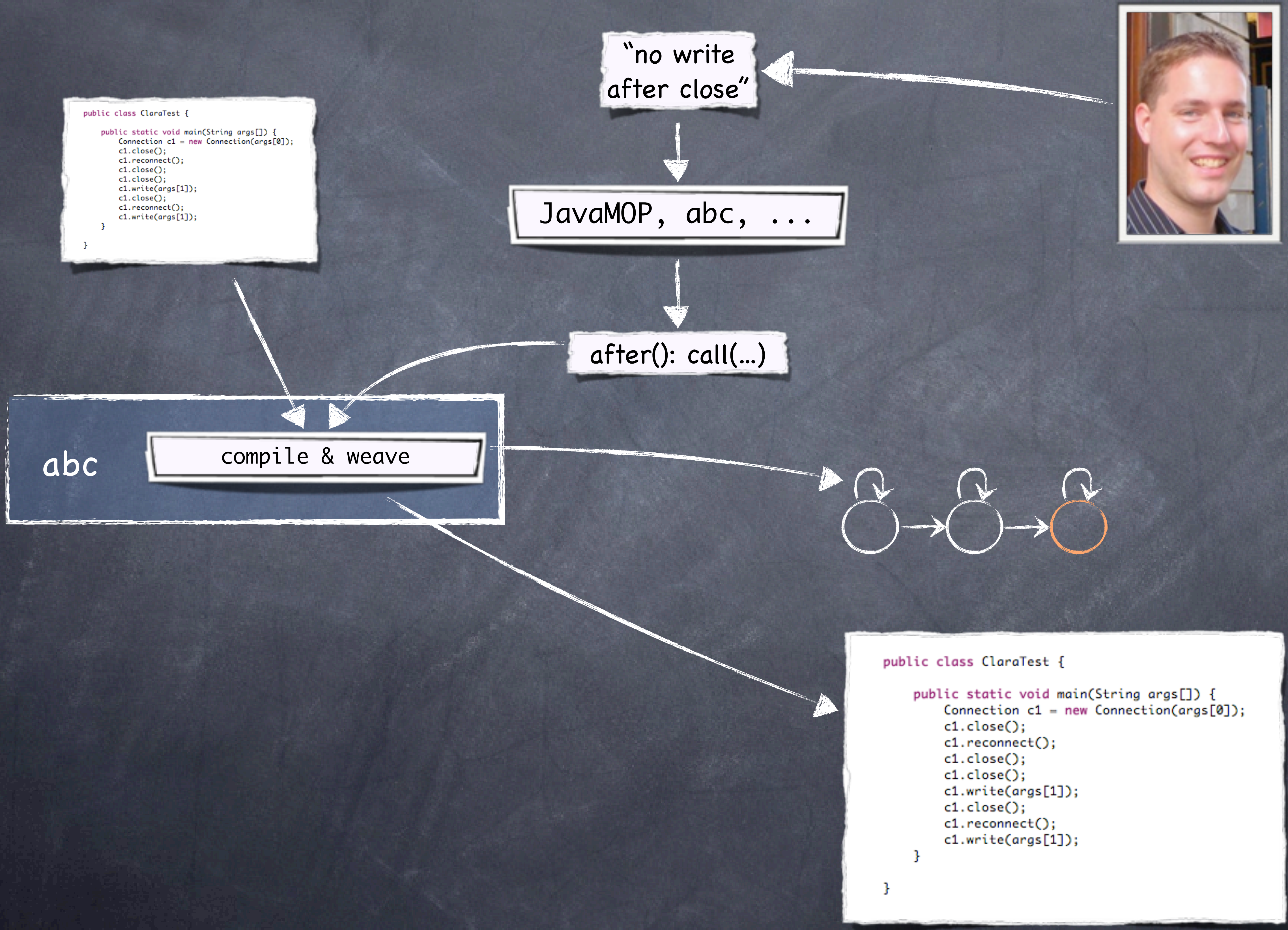
The Clara Framework



```java
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]);
    }

}
```

$$\begin{tikzcd}
\text{“no write after close”} \arrow[d] \\
\text{JavaMOP, abc, ...} \arrow[d] \\
\text{after(): call(...)}
\end{tikzcd}$$

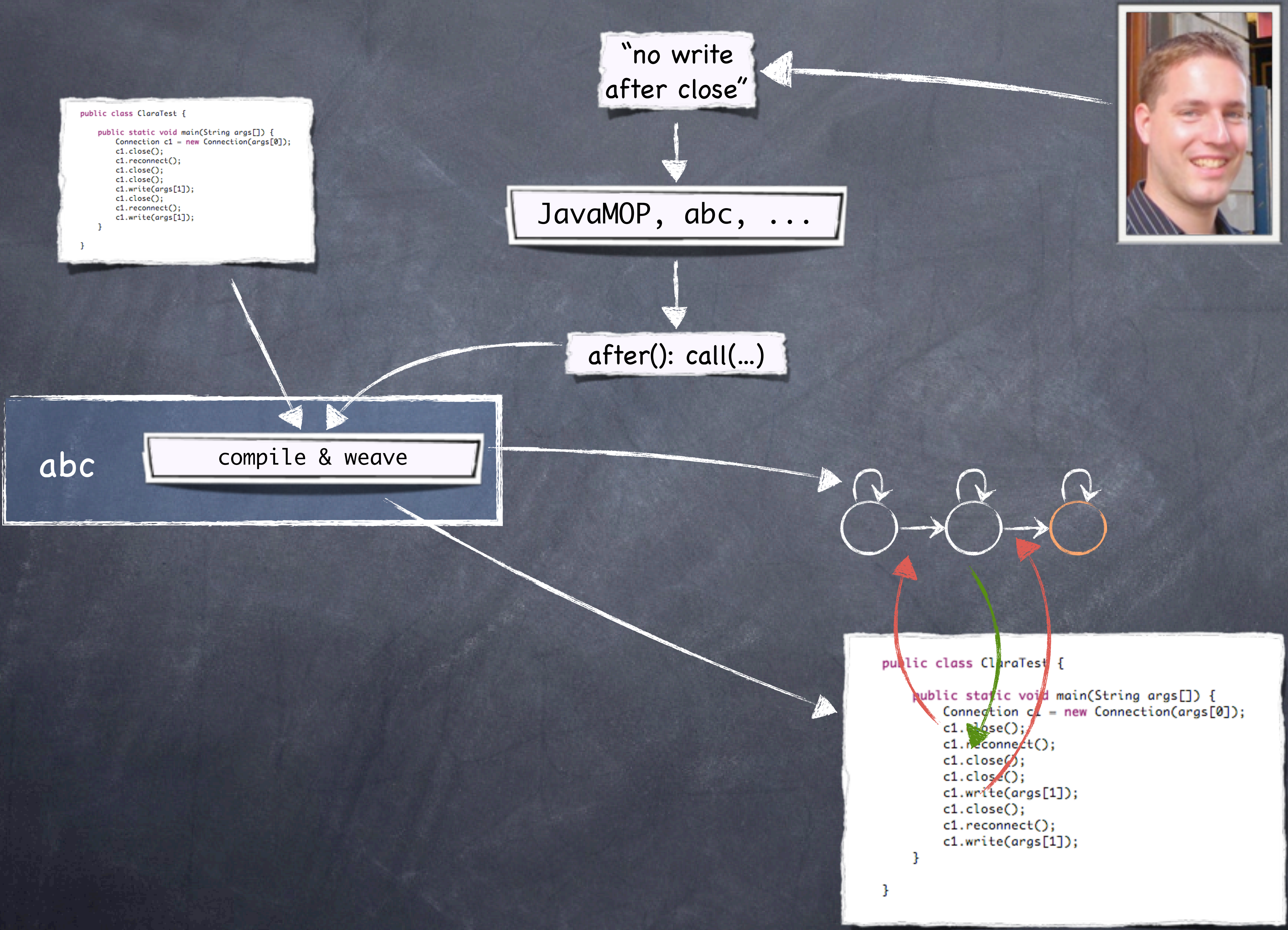
The Clara Framework



```
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]);
    }
}
```

```
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]);
    }
}
```

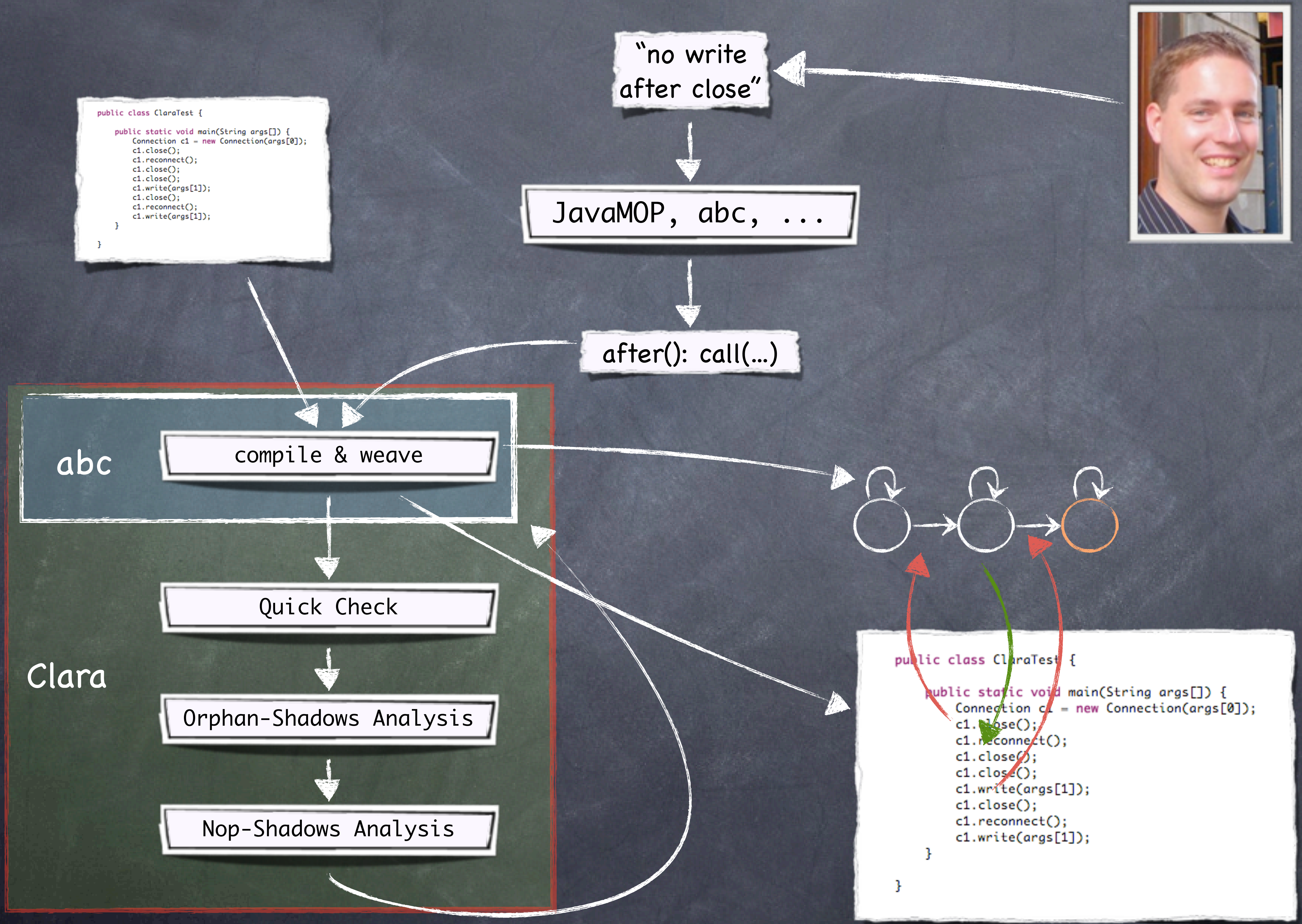
The Clara Framework



```
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]);
    }
}
```

```
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]);
    }
}
```

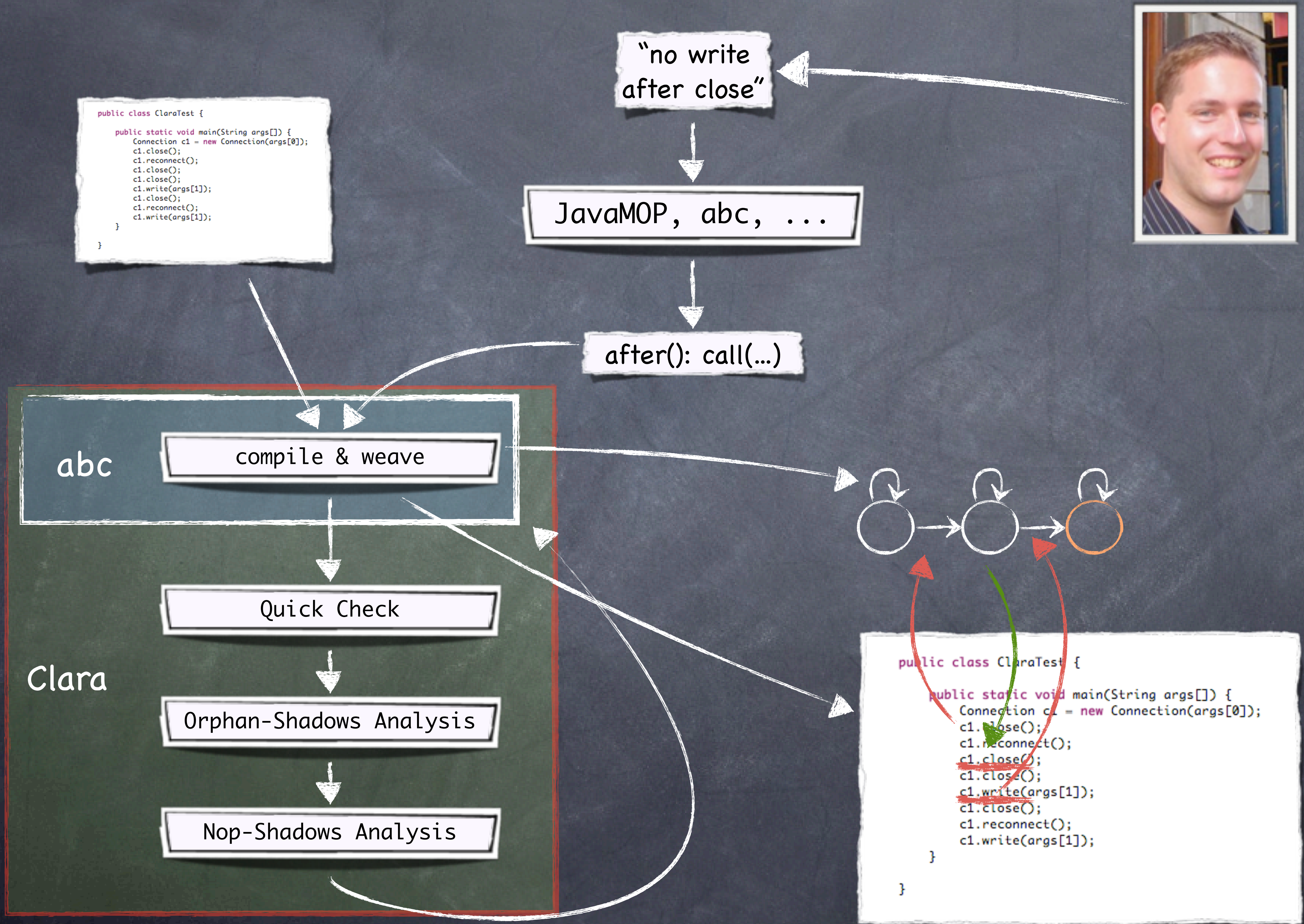
The Clara Framework



```
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]);
    }

}
```

```
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]);
    }

}
```

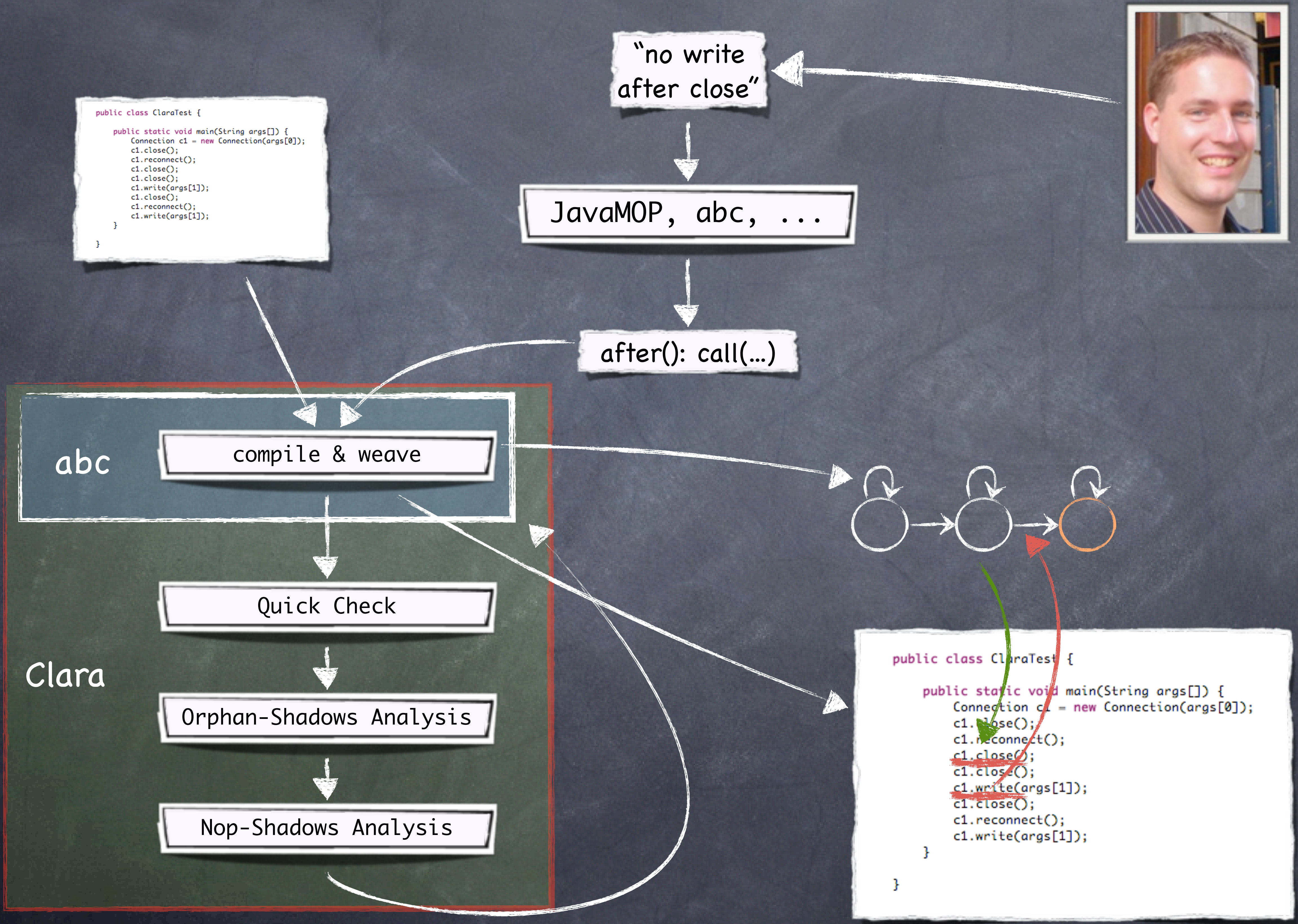
The Clara Framework



```java
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]);
    }
}
```

```java
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]);
    }
}
```

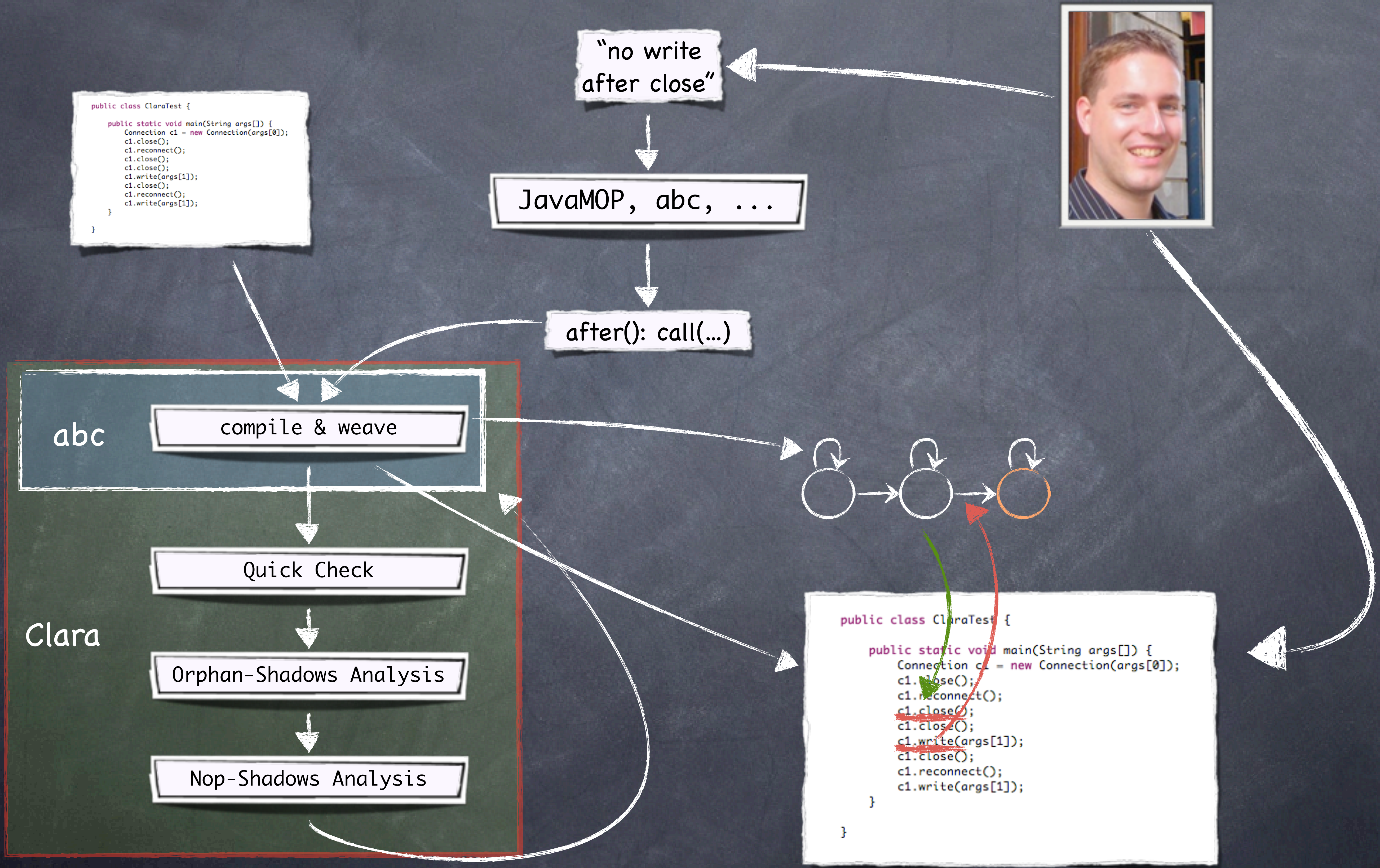
The Clara Framework



```
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]);
    }

}
```

```
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]);
    }

}
```

The Clara Framework

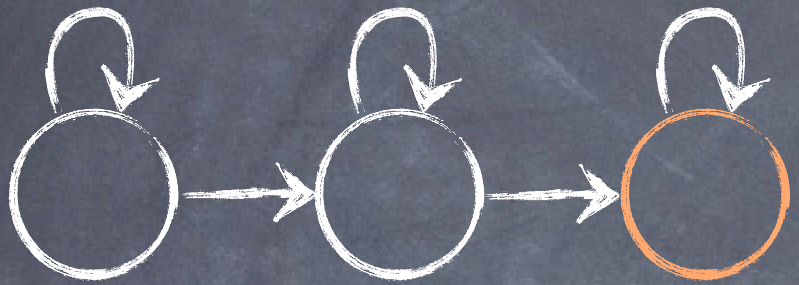


```
public class ClaraTest {

    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]);
    }
}
```

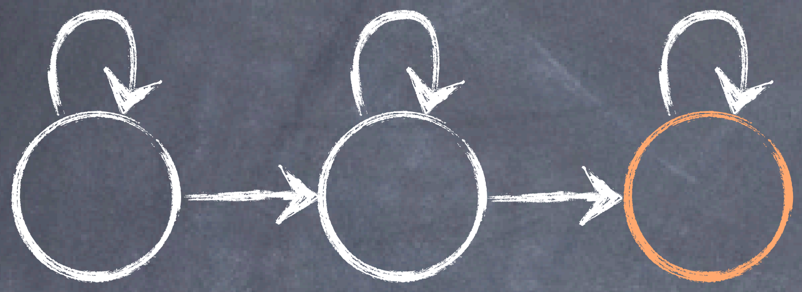
```
public class ClaraTest {

    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]);
    }
}
```

Houston, we have a problem...

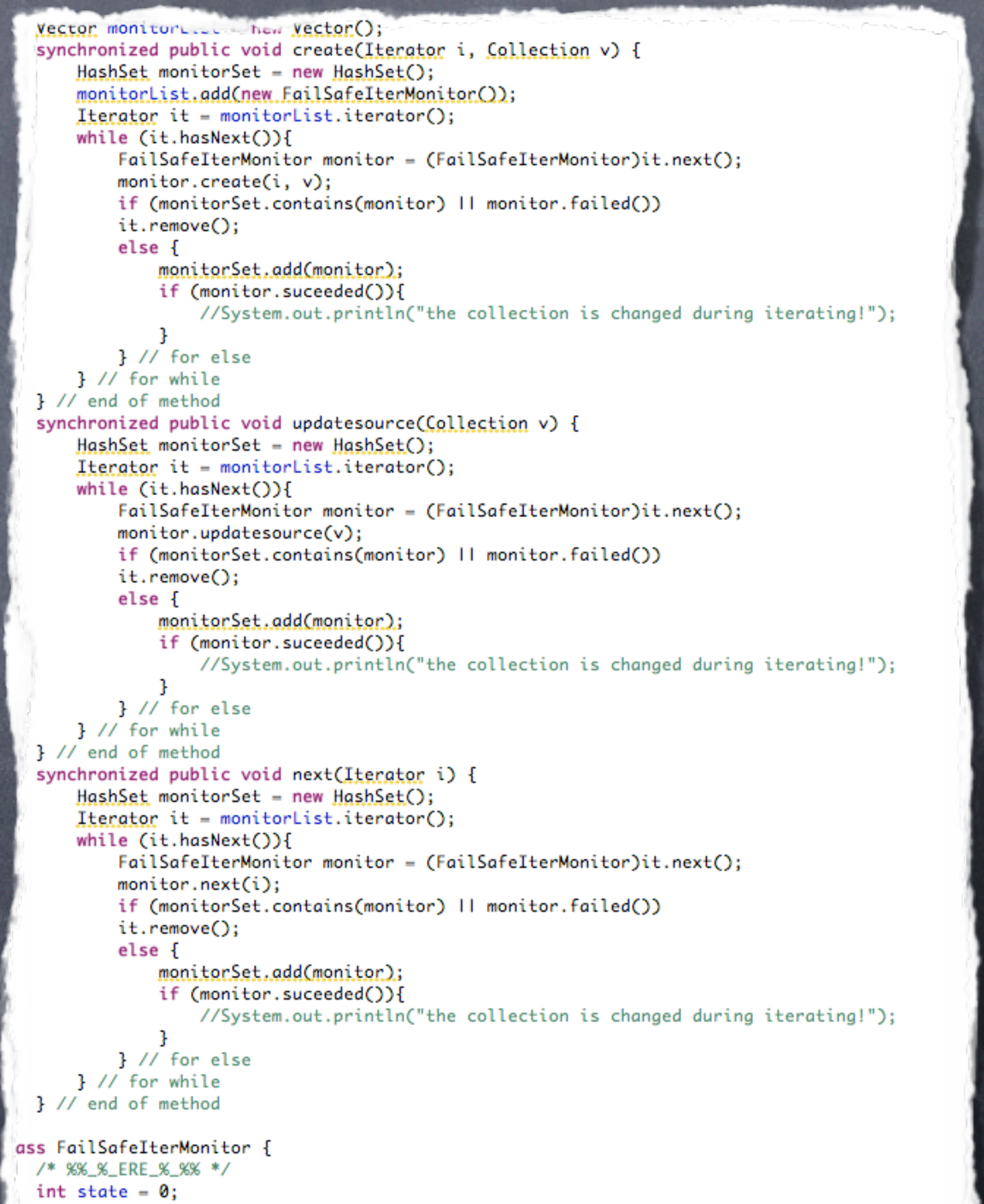
Need:



Need:

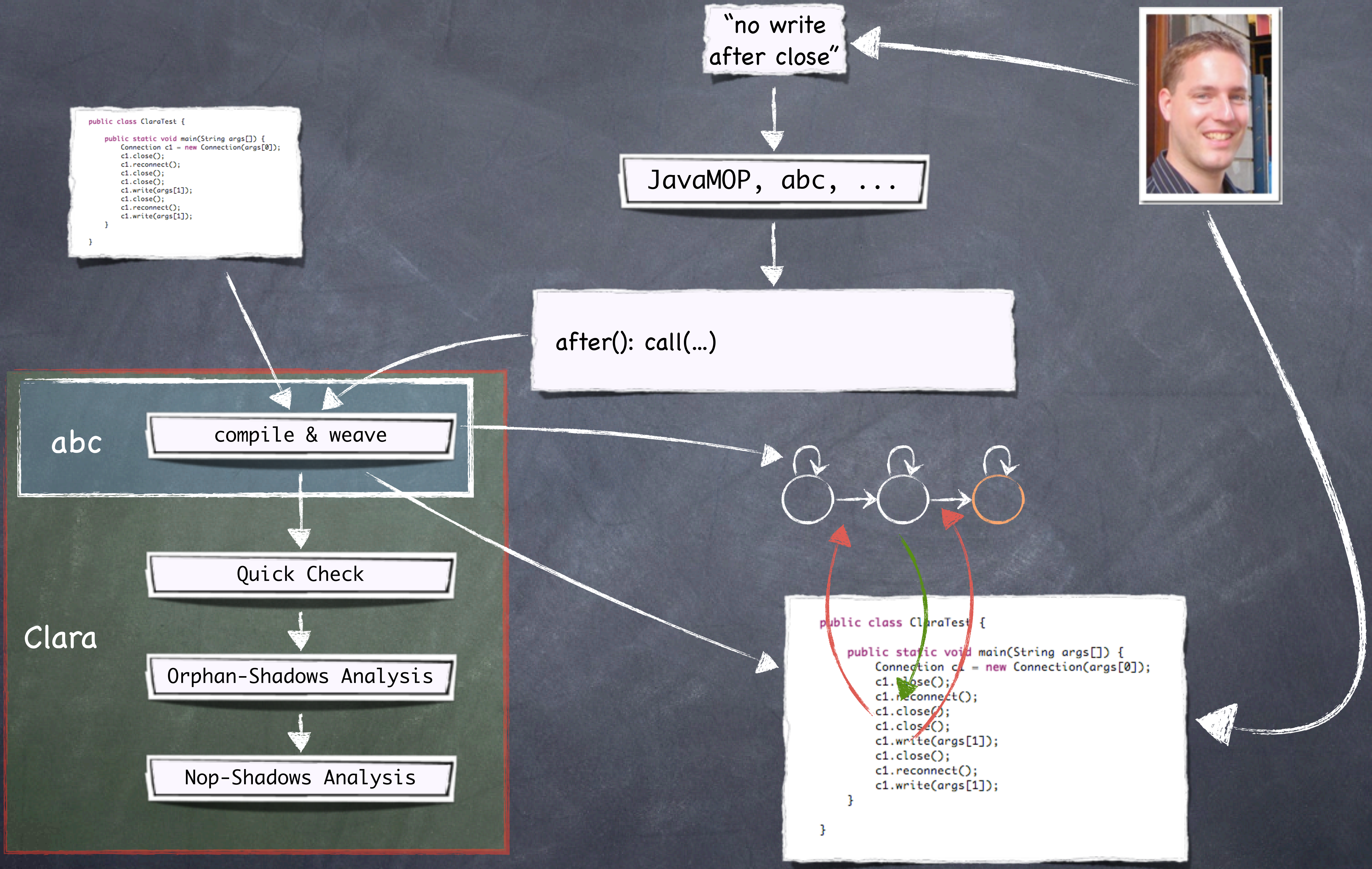


Have:



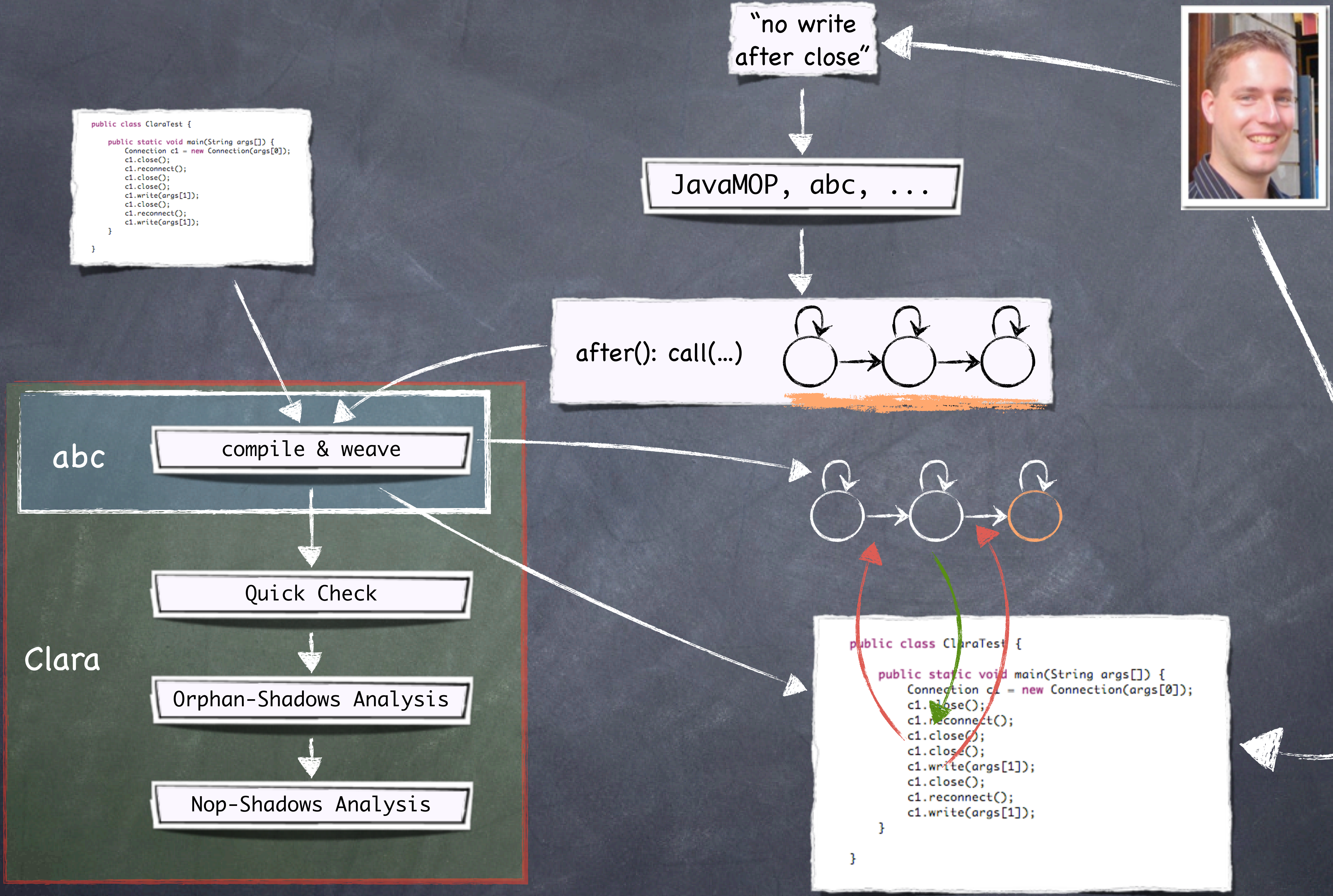
```
Vector monitor[illegible] new Vector();
synchronized public void create(Iterator i, Collection v) {
    HashSet monitorSet = new HashSet();
    monitorList.add(new FailSafeIterMonitor());
    Iterator it = monitorList.iterator();
    while (it.hasNext()){
        FailSafeIterMonitor monitor = (FailSafeIterMonitor)it.next();
        monitor.create(i, v);
        if (monitorSet.contains(monitor) || monitor.failed())
        it.remove();
        else {
            monitorSet.add(monitor);
            if (monitor.suceeded()){
                //System.out.println("the collection is changed during iterating!");
            }
        } // for else
    } // for while
} // end of method
synchronized public void updatesource(Collection v) {
    HashSet monitorSet = new HashSet();
    Iterator it = monitorList.iterator();
    while (it.hasNext()){
        FailSafeIterMonitor monitor = (FailSafeIterMonitor)it.next();
        monitor.updatesource(v);
        if (monitorSet.contains(monitor) || monitor.failed())
        it.remove();
        else {
            monitorSet.add(monitor);
            if (monitor.suceeded()){
                //System.out.println("the collection is changed during iterating!");
            }
        } // for else
    } // for while
} // end of method
synchronized public void next(Iterator i) {
    HashSet monitorSet = new HashSet();
    Iterator it = monitorList.iterator();
    while (it.hasNext()){
        FailSafeIterMonitor monitor = (FailSafeIterMonitor)it.next();
        monitor.next(i);
        if (monitorSet.contains(monitor) || monitor.failed())
        it.remove();
        else {
            monitorSet.add(monitor);
            if (monitor.suceeded()){
                //System.out.println("the collection is changed during iterating!");
            }
        } // for else
    } // for while
} // end of method

ass FailSafeIterMonitor {
 /* %%_%_ERE_%_%% */
 int state = 0;
```

The Solution…



```
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]);
    }

}
```

```
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]);
    }

}
```



```
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]);
    }

}
```

```
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]);
    }

}
```

Dependency State Machines

```
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

```
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 State Machines

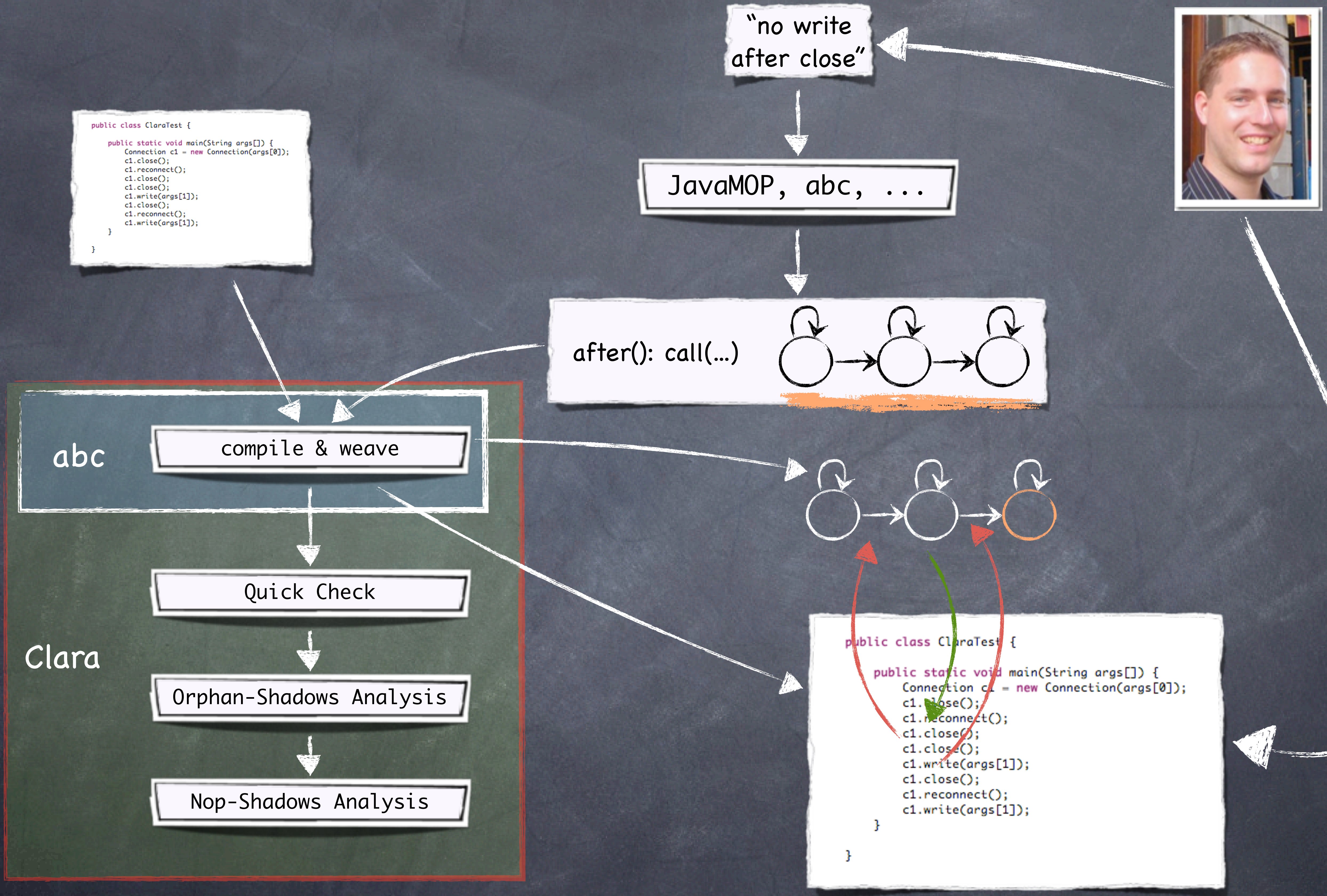
abstract

```
Set closed = new WeakIdentityHashSet();

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

concrete

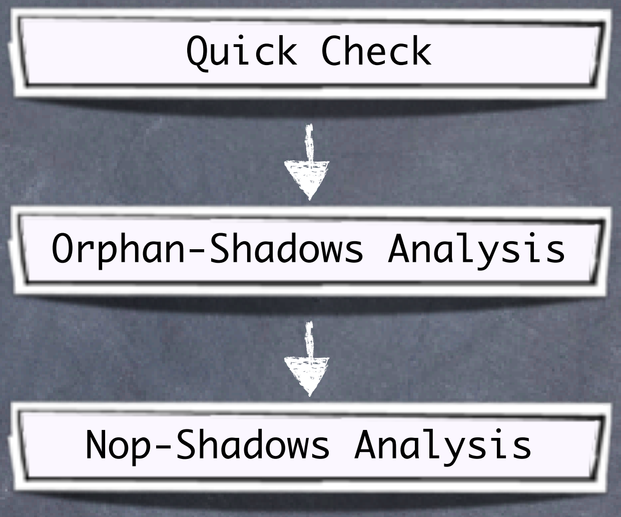
```
dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
}

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!");
}
```

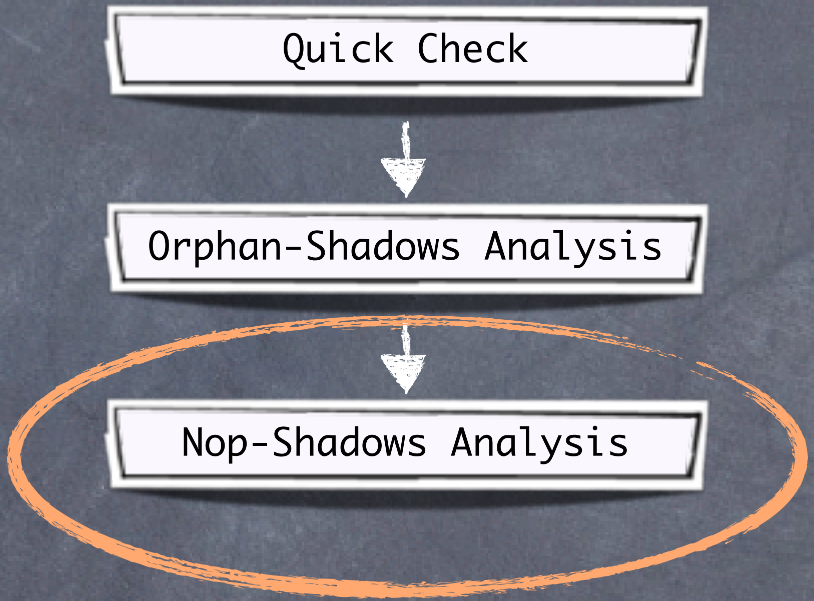
```
Set closed = new WeakIdentityHashSet();

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;
}
```



```
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]);
    }

}
```

```
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]);
    }

}
```


Quick Check
Orphan-Shadows Analysis
Nop-Shadows Analysis

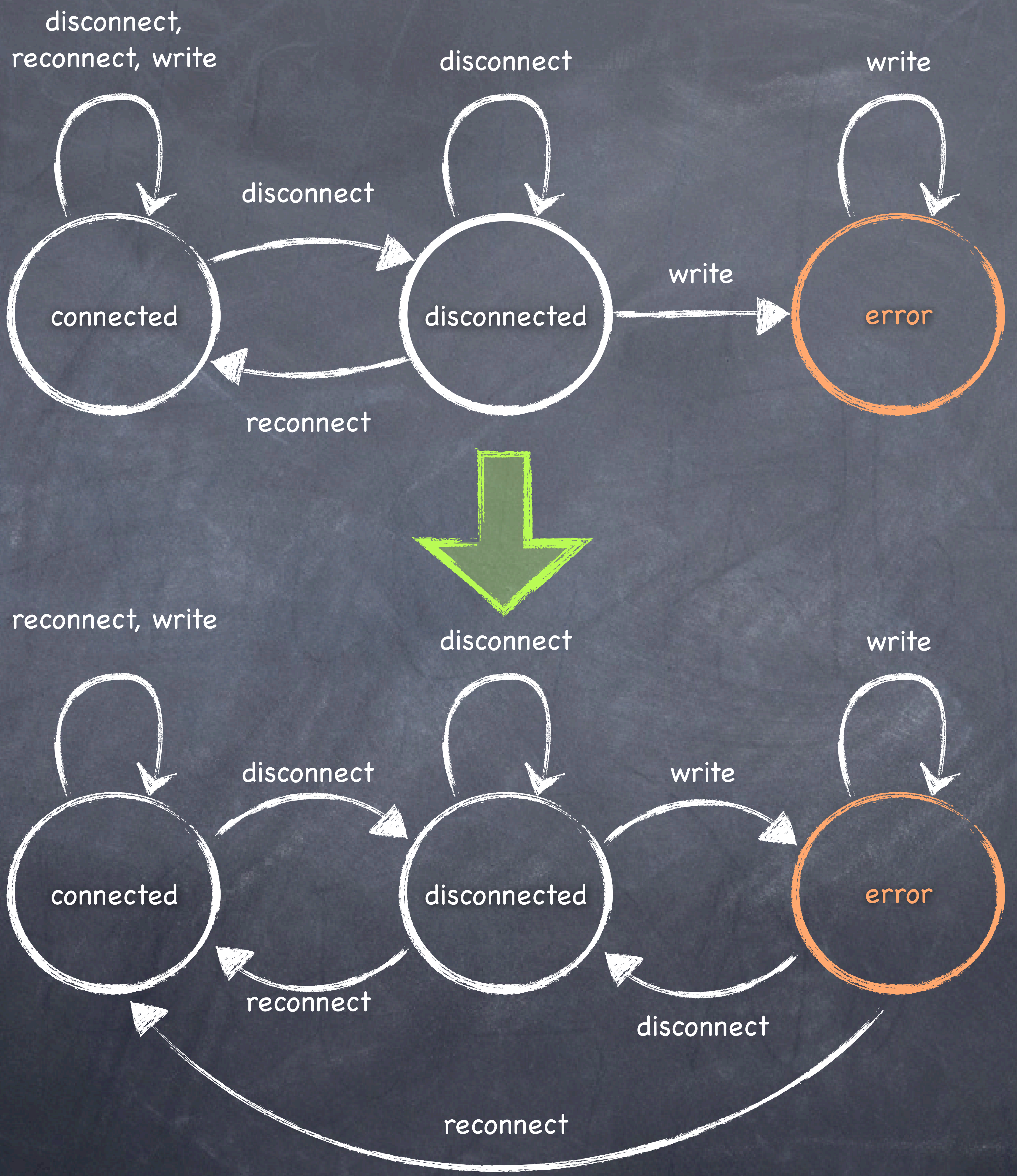

Quick Check
Orphan-Shadows Analysis
Nop-Shadows Analysis

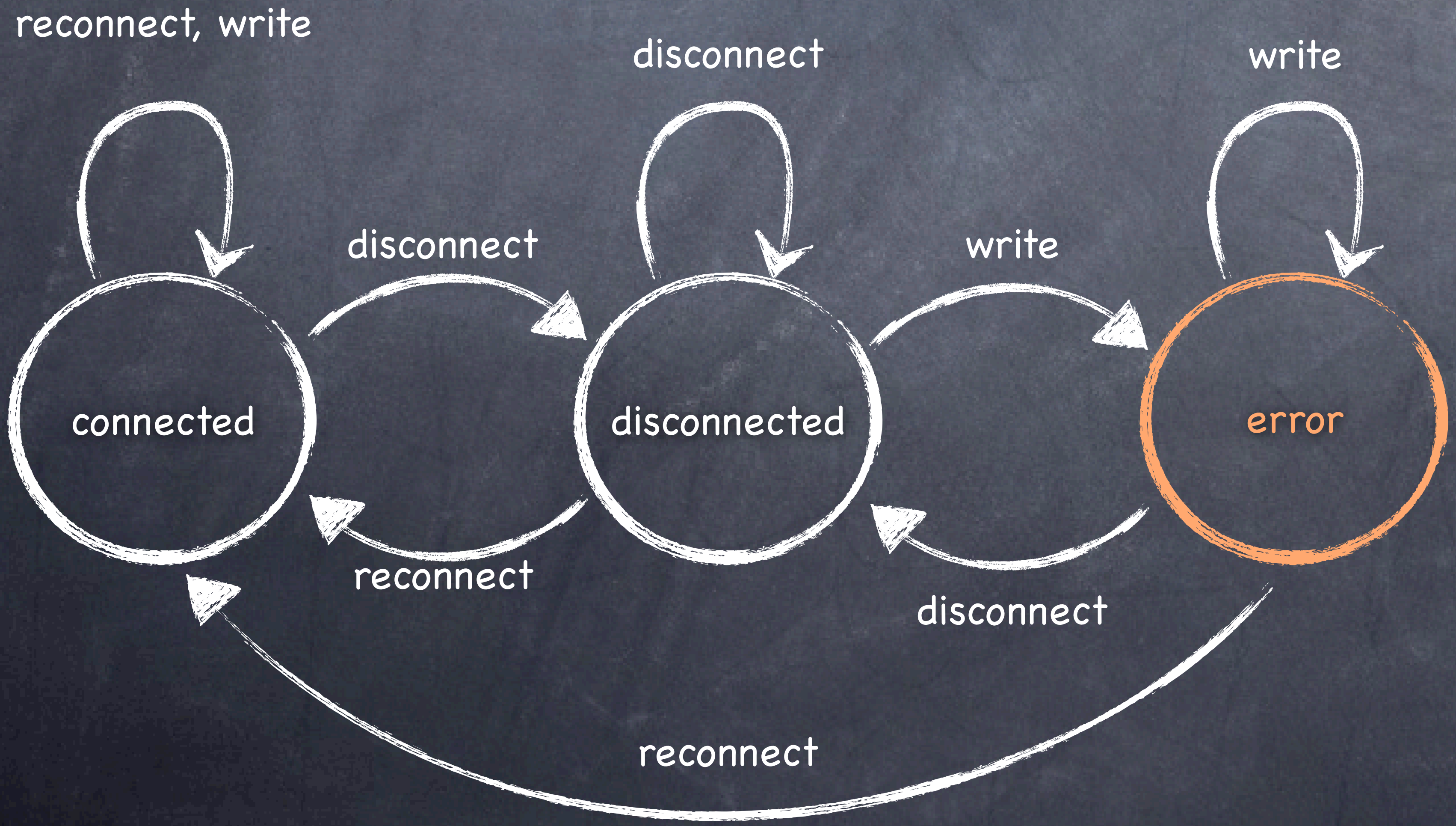
Nop-Shadows Analysis

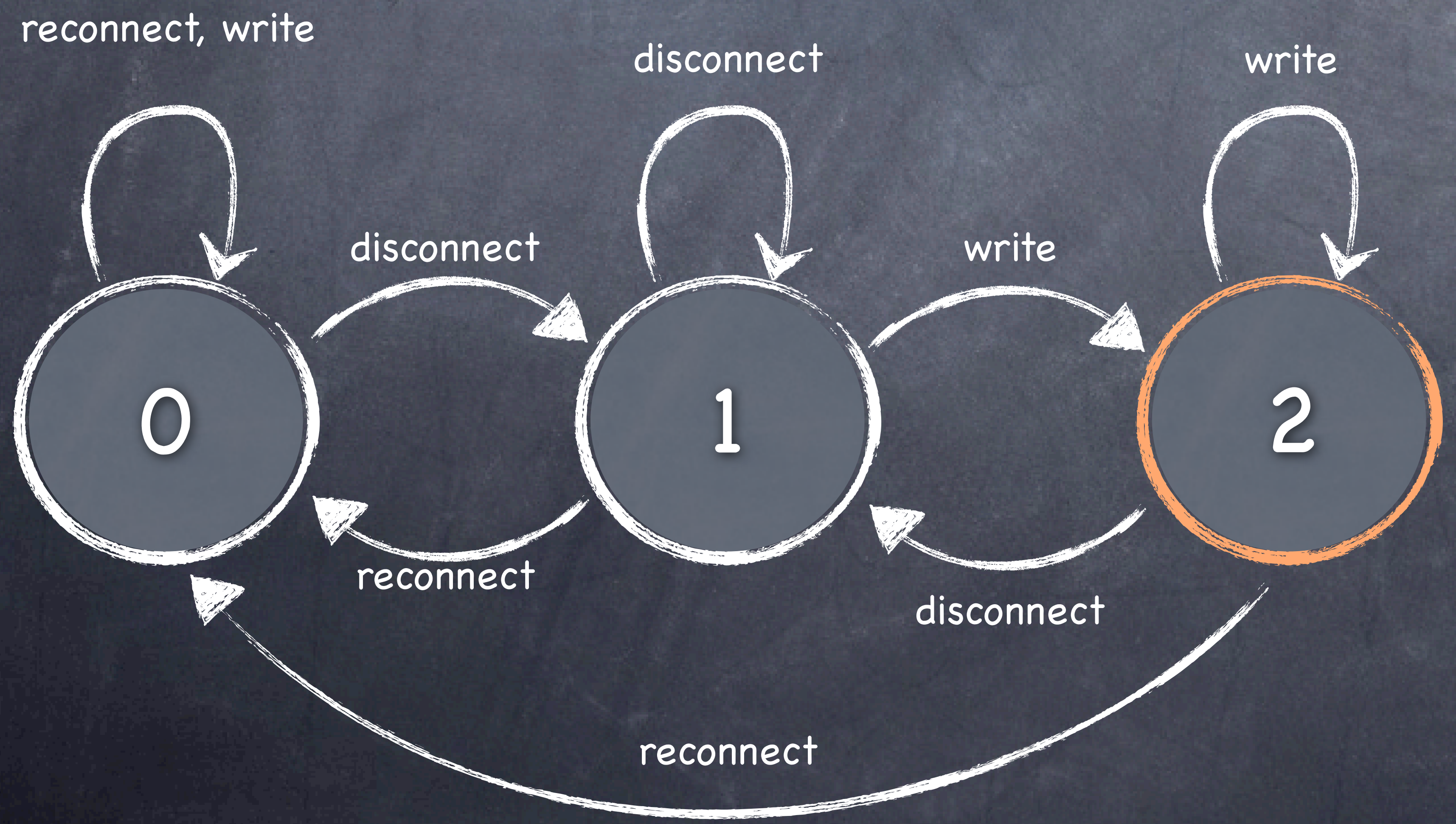
Idea:

For every joinpoint shadow s:

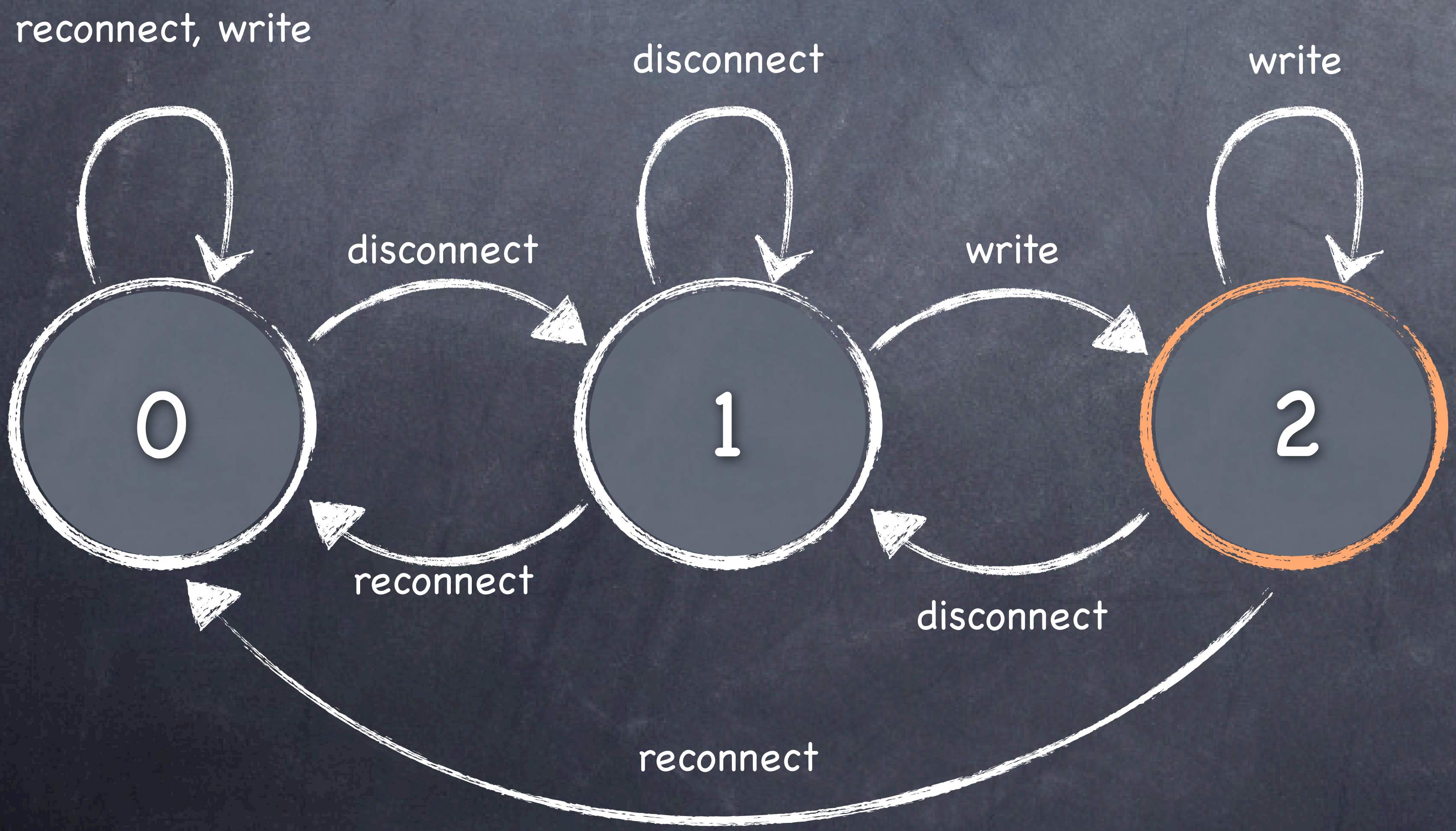
- Identify states that are equivalent at s.
- If s may transition only between equivalent states then disable s.


disconnect, reconnect, write
disconnect
write
disconnect
connected
disconnected
write
error
reconnect
reconnect, write
disconnect
write
disconnect
write
connected
disconnected
error
reconnect
disconnect
reconnect

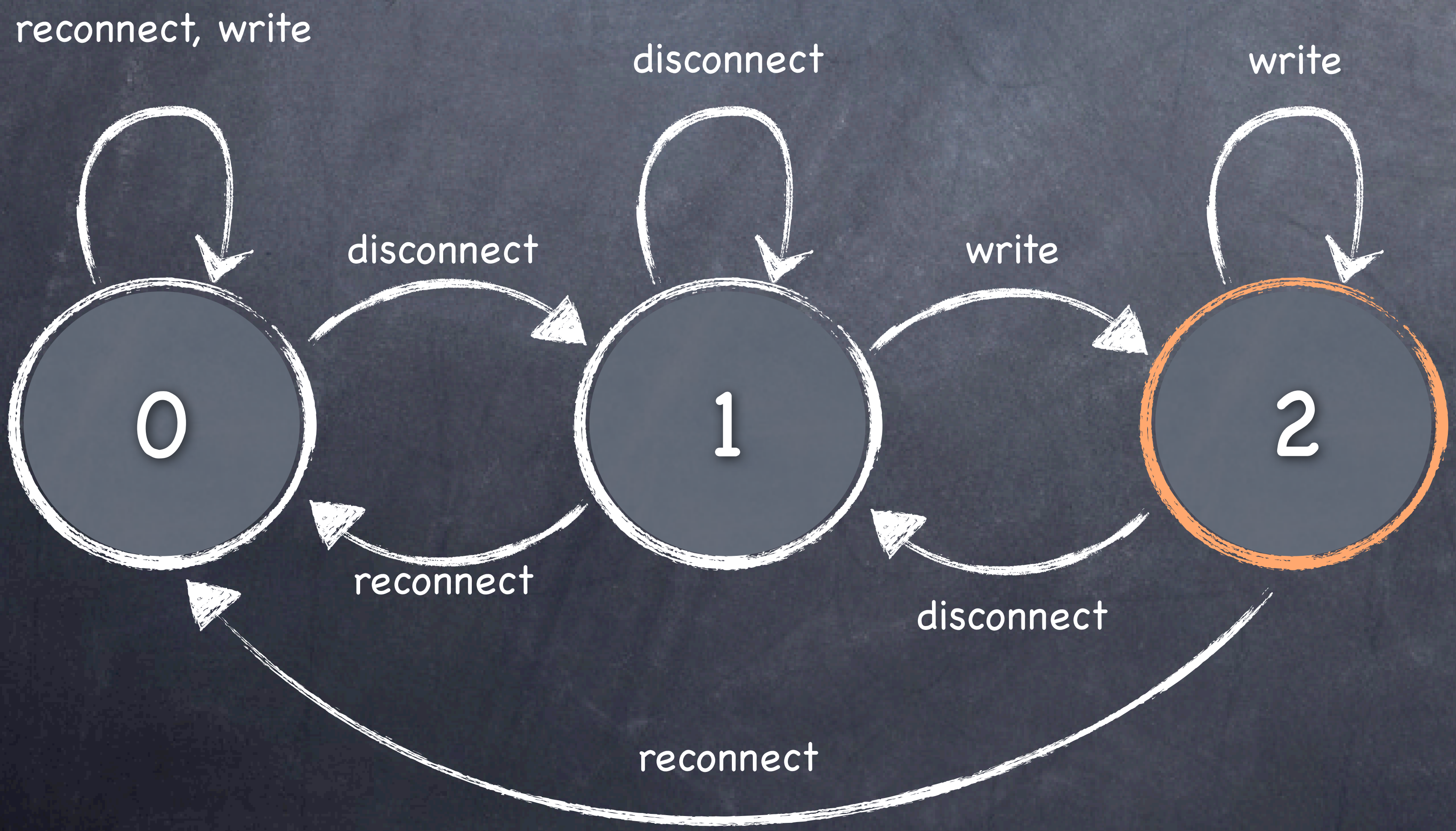

reconnect, write
disconnect
write
disconnect
write
connected
disconnected
error
reconnect
disconnect
reconnect


reconnect, write
disconnect
write
disconnect
write
0
1
2
reconnect
disconnect
reconnect

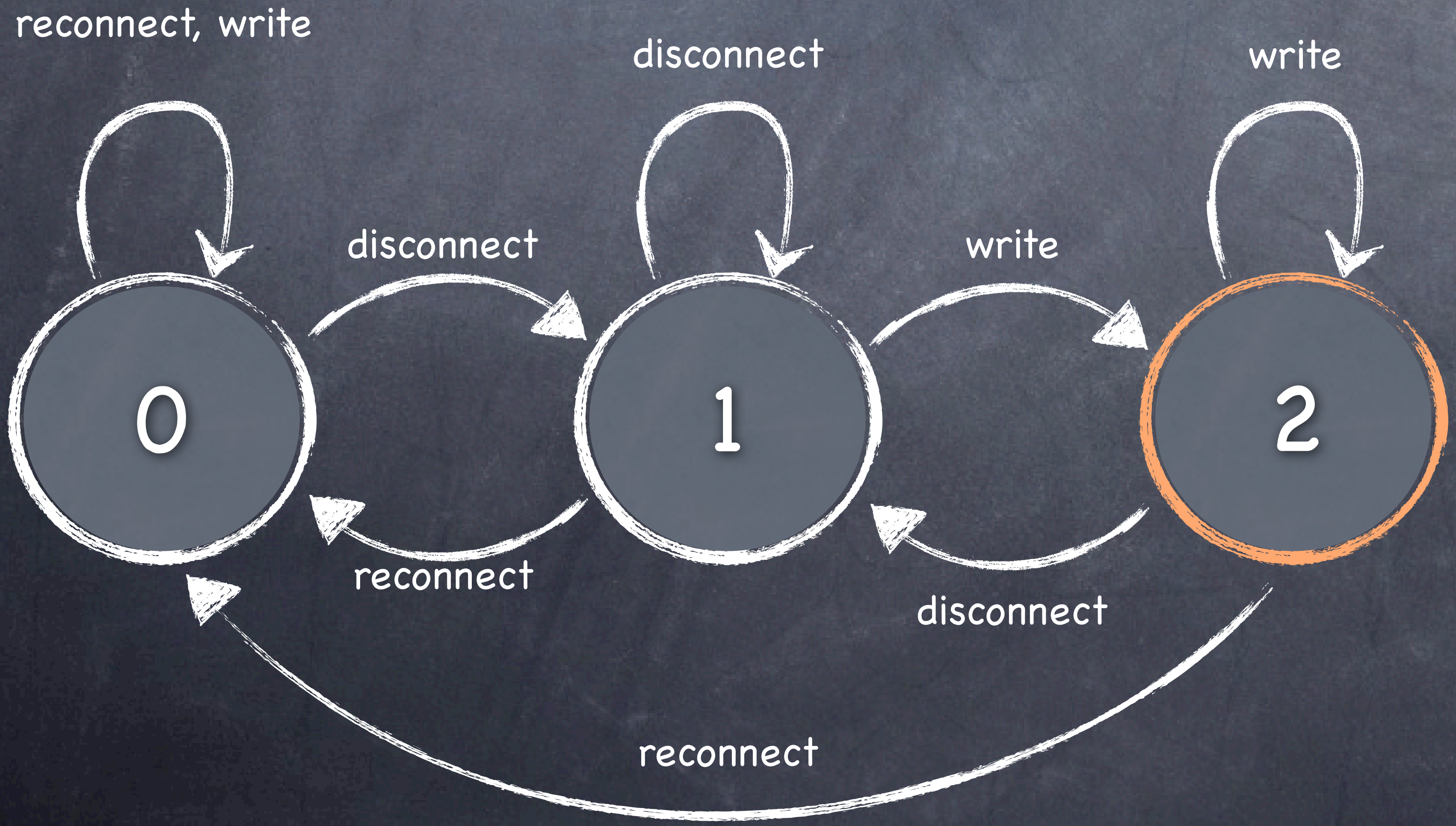
```
                                   0
c1.close();
                                   1
c1.reconnect();
                                   0
c1.close();
                                   1
c1.close();
                                   1
c1.write(..);
                                   2
c1.close();
                                   1
c1.reconnect();
                                   0
c1.write(..);
                                   0
```

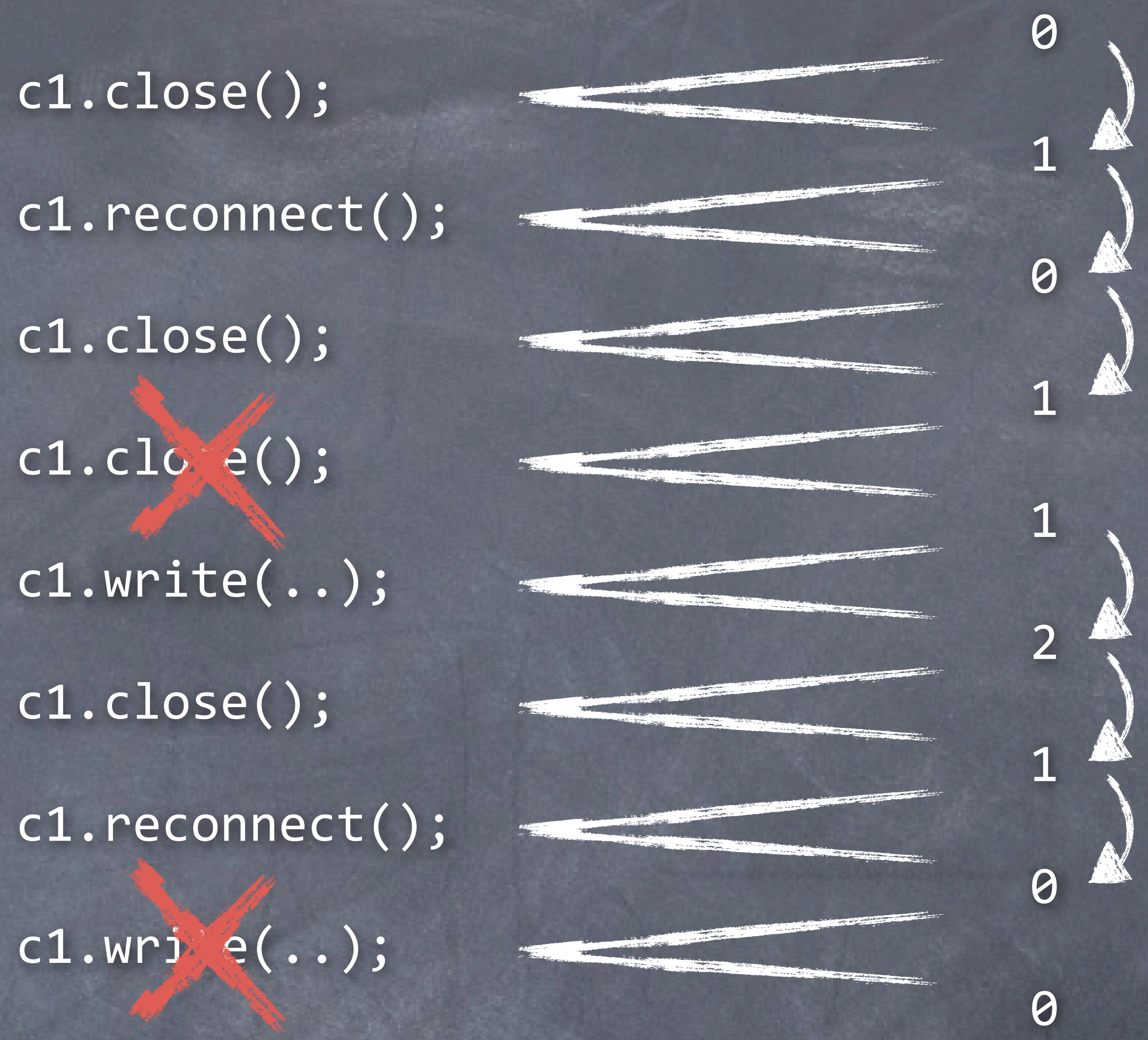


```
                          0
c1.close();
                          1
c1.reconnect();
                          0
c1.close();
                          1
c1.close();
                          1
c1.write(..);
                          2
c1.close();
                          1
c1.reconnect();
                          0
c1.write(..);
                          0
```

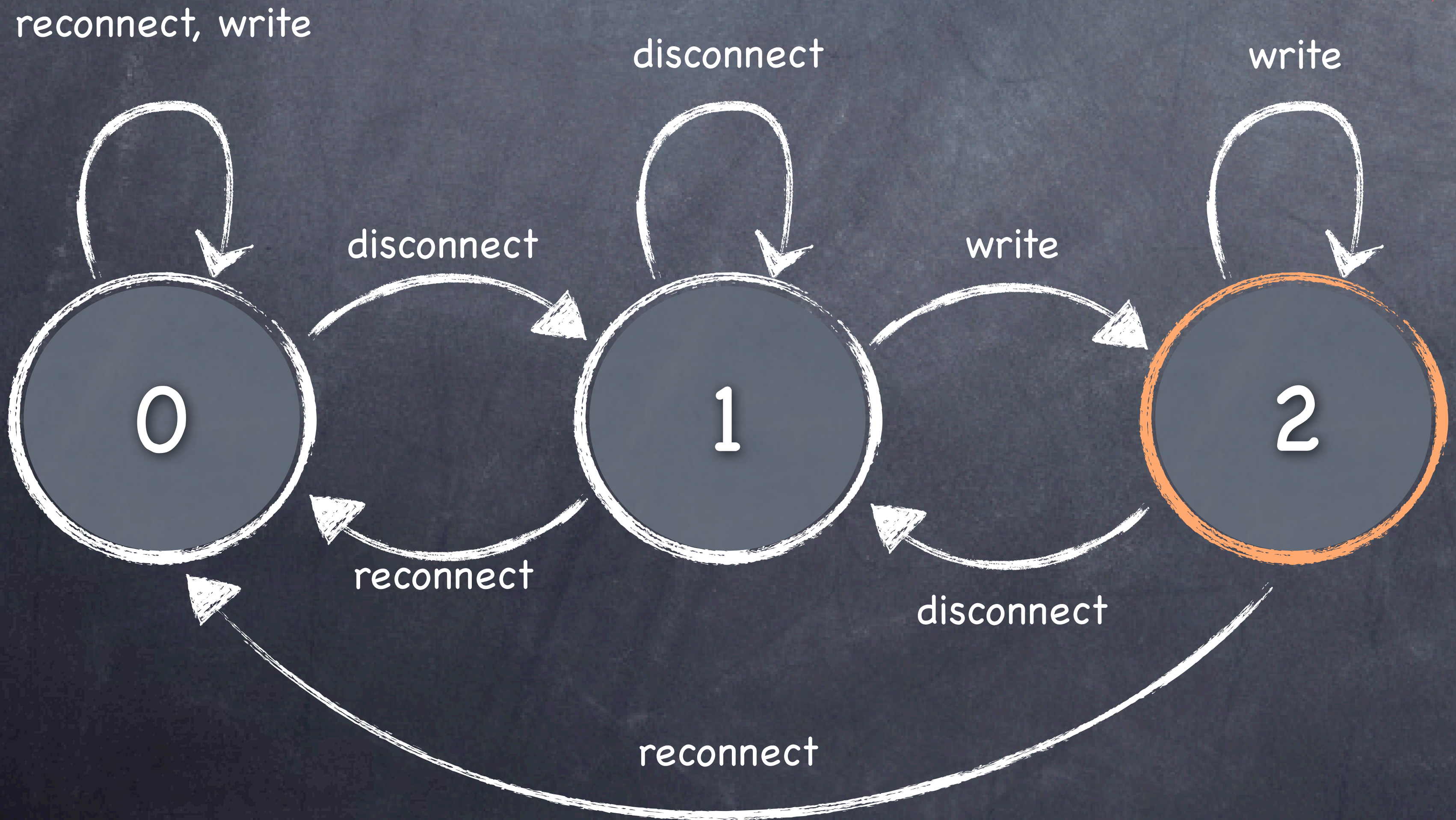


```
                        0
c1.close();
                        1
c1.reconnect();
                        0
c1.close();
                        1
c1.close();
                        1
c1.write(..);
                        2
c1.close();
                        1
c1.reconnect();
                        0
c1.write(..);
                        0
```

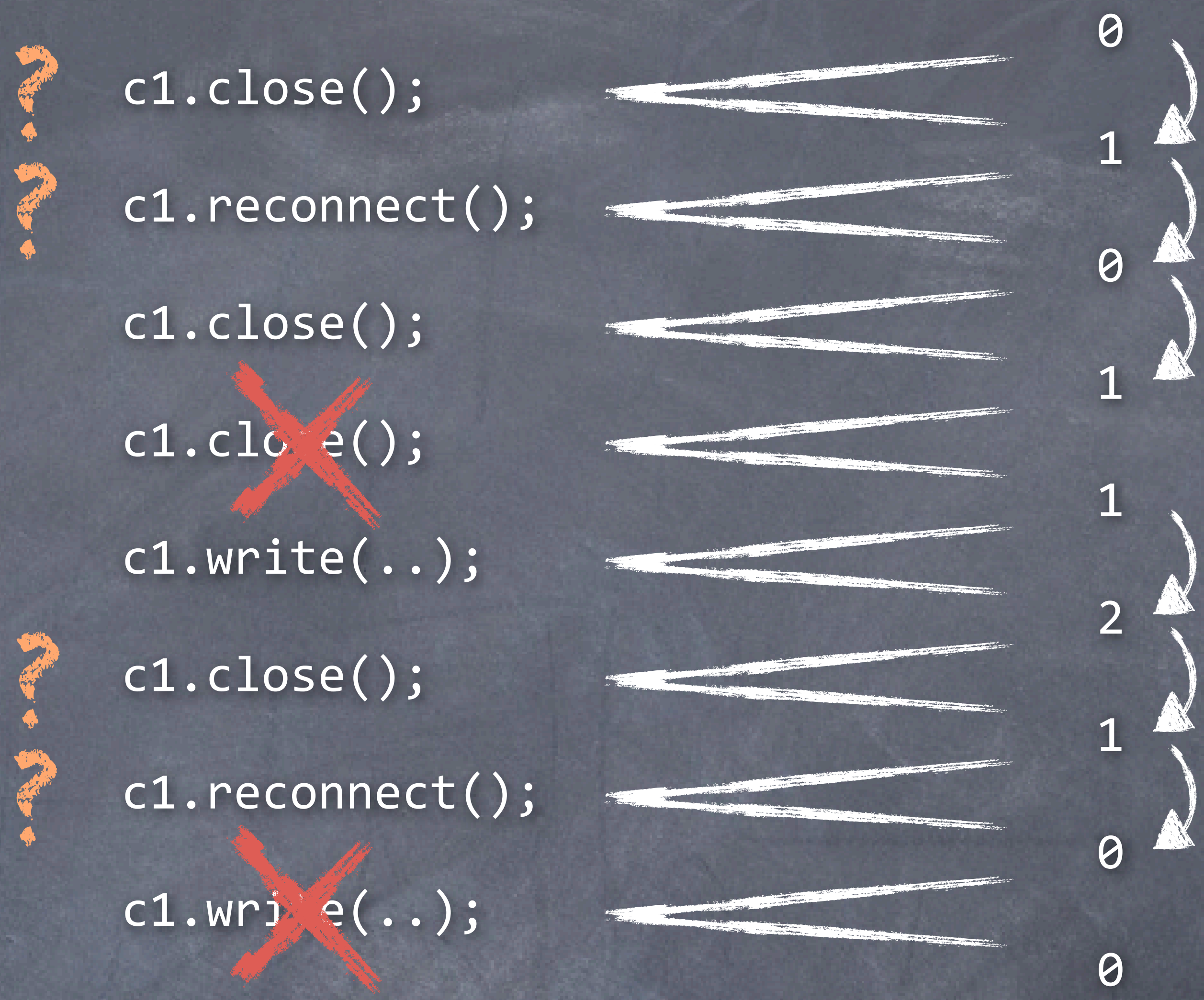




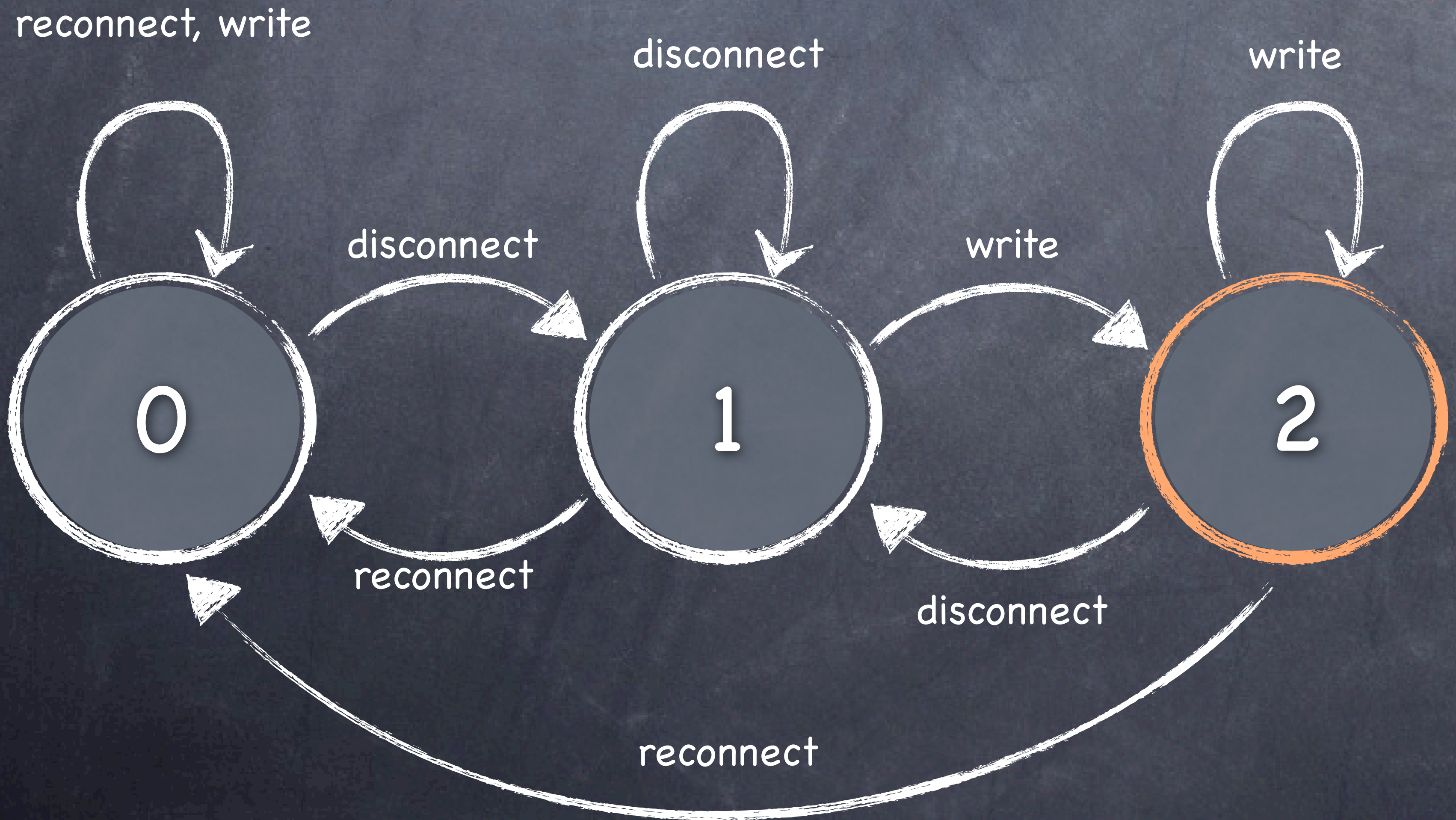
c1.close();
c1.reconnect();
c1.close();
~~c1.close();~~
c1.write(..);
c1.close();
c1.reconnect();
~~c1.write(..);~~



$$\begin{tikzcd}
0 \arrow[loop above, "\text{reconnect, write}"] \arrow[r, bend left, "\text{disconnect}"] & 1 \arrow[loop above, "\text{disconnect}"] \arrow[l, bend left, "\text{reconnect}"] \arrow[r, bend left, "\text{write}"] & 2 \arrow[loop above, "\text{write}"] \arrow[l, bend left, "\text{disconnect}"] \arrow[ll, bend left=60, "\text{reconnect}"]
\end{tikzcd}$$



```
? c1.close();
? c1.reconnect();
  c1.close();
  c1.close();
  c1.write(..);
? c1.close();
? c1.reconnect();
  c1.write(..);
```



$$\begin{tikzcd}
0 \arrow[loop above, "\text{reconnect, write}"] \arrow[r, bend left, "\text{disconnect}"] & 1 \arrow[loop above, "\text{disconnect}"] \arrow[l, bend left, "\text{reconnect}"] \arrow[r, bend left, "\text{write}"] & 2 \arrow[loop above, "\text{write}"] \arrow[l, bend left, "\text{disconnect}"] \arrow[ll, bend left=50, "\text{reconnect}"]
\end{tikzcd}$$

$$\begin{tikzcd}
0 \arrow[loop above, "{\text{reconnect, write}}"] \arrow[r, bend left, "\text{disconnect}"] & 1 \arrow[loop above, "\text{disconnect}"] \arrow[l, bend left, "\text{reconnect}"] \arrow[r, bend left, "\text{write}"] & 2 \arrow[loop above, "\text{write}"] \arrow[l, bend left, "\text{disconnect}"] \arrow[ll, bend left=60, "\text{reconnect}"]
\end{tikzcd}$$

$$\begin{tikzcd}
\{2\} \arrow[r, "\text{write}"] & \{1,2\} \arrow[loop above, "\text{write}"] \arrow[r, "\text{disconnect}"] & \{0,1,2\} \arrow[loop above, "{\text{disconnect reconnect, write}}"]
\end{tikzcd}$$

```
                      0    {}  {0,1,2}
c1.close();
                      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.reconnect();`

`c1.close();`

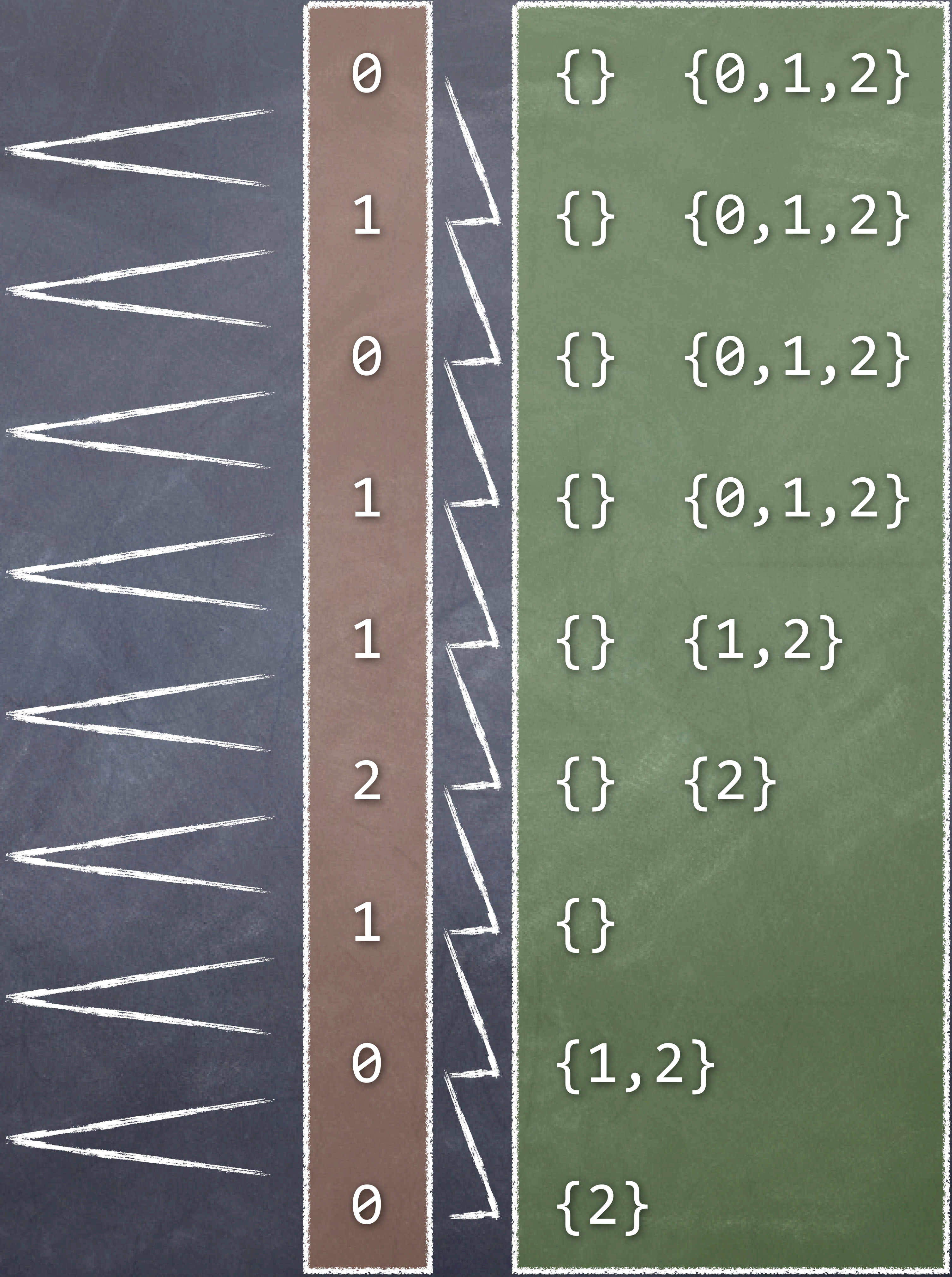
~~`c1.close();`~~

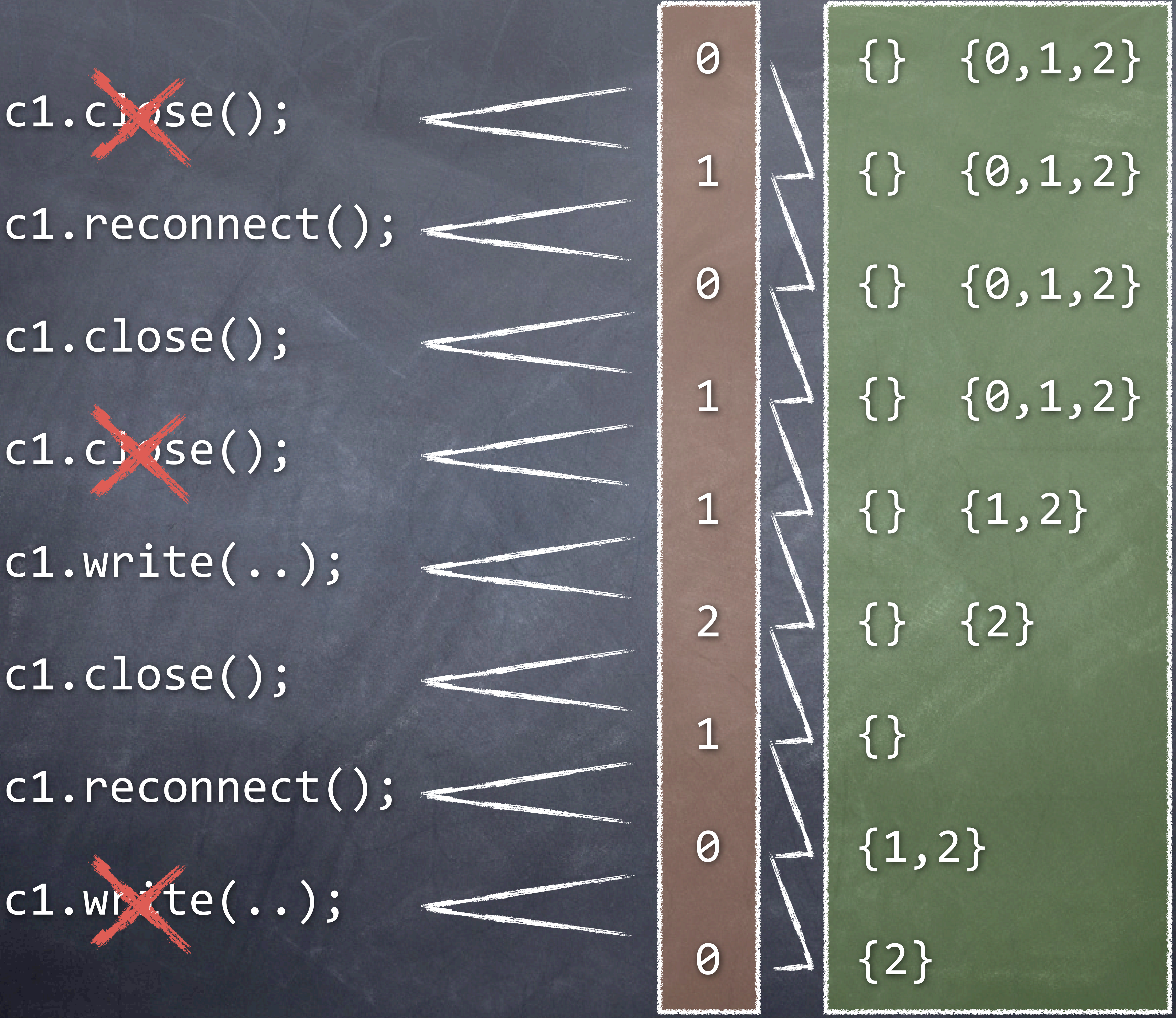
`c1.write(..);`

`c1.close();`

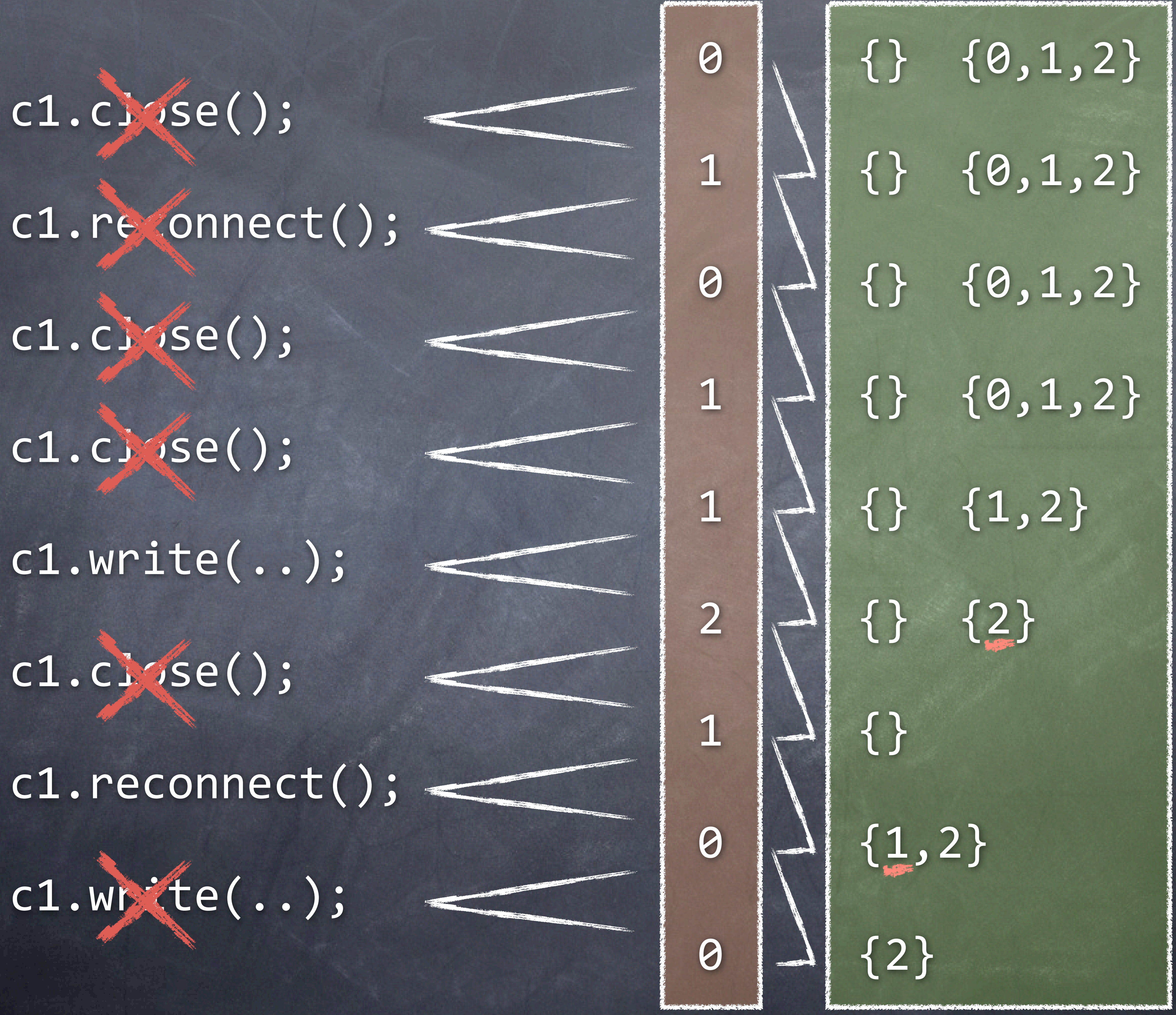
`c1.reconnect();`

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





	0	{} {0,1,2}
~~c1.close();~~		
	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.reconnect();~~

~~c1.close();~~

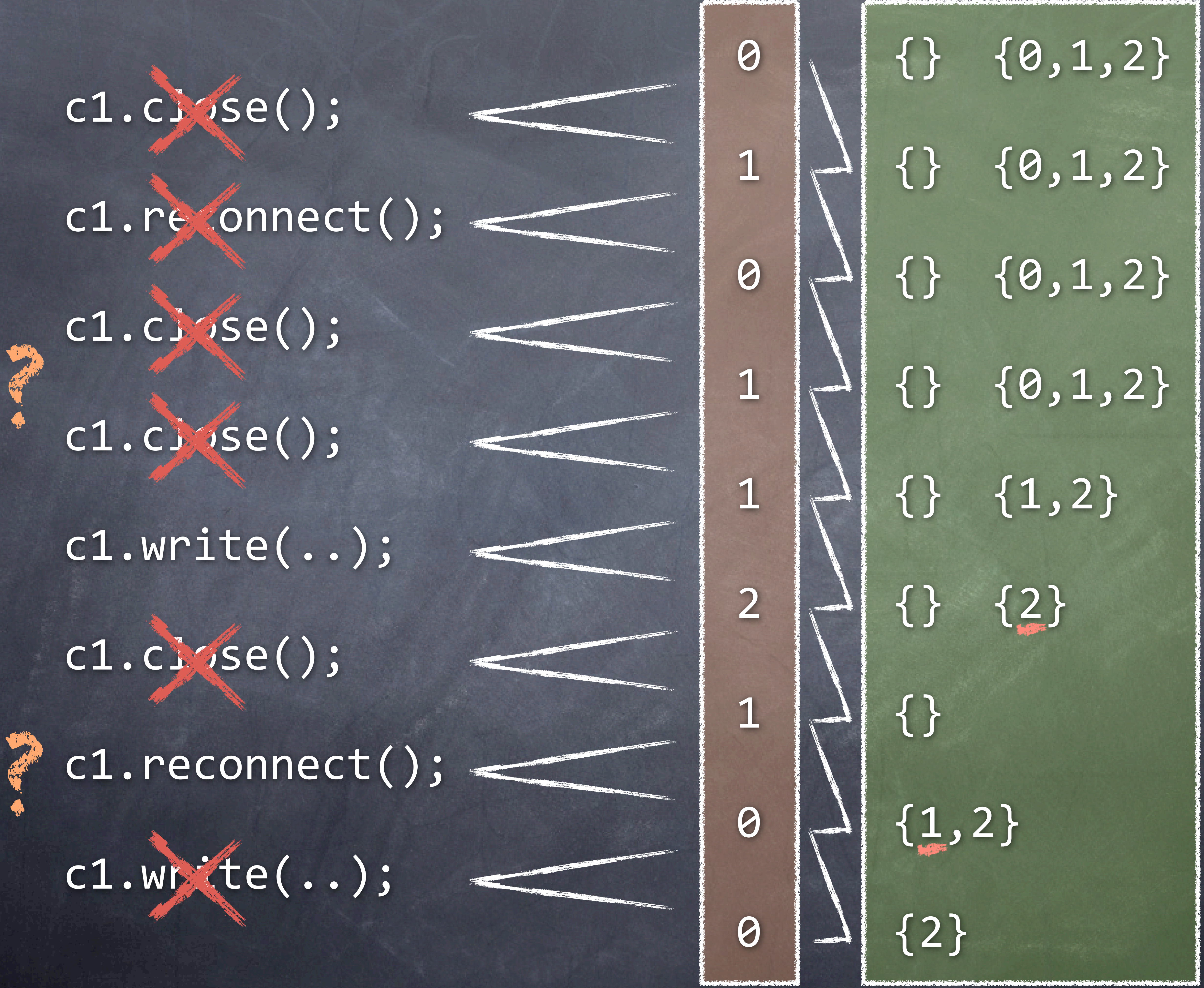
~~c1.close();~~

c1.write(..);

~~c1.close();~~

c1.reconnect();

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



0 {} {0,1,2}

~~c1.close();~~

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();~~

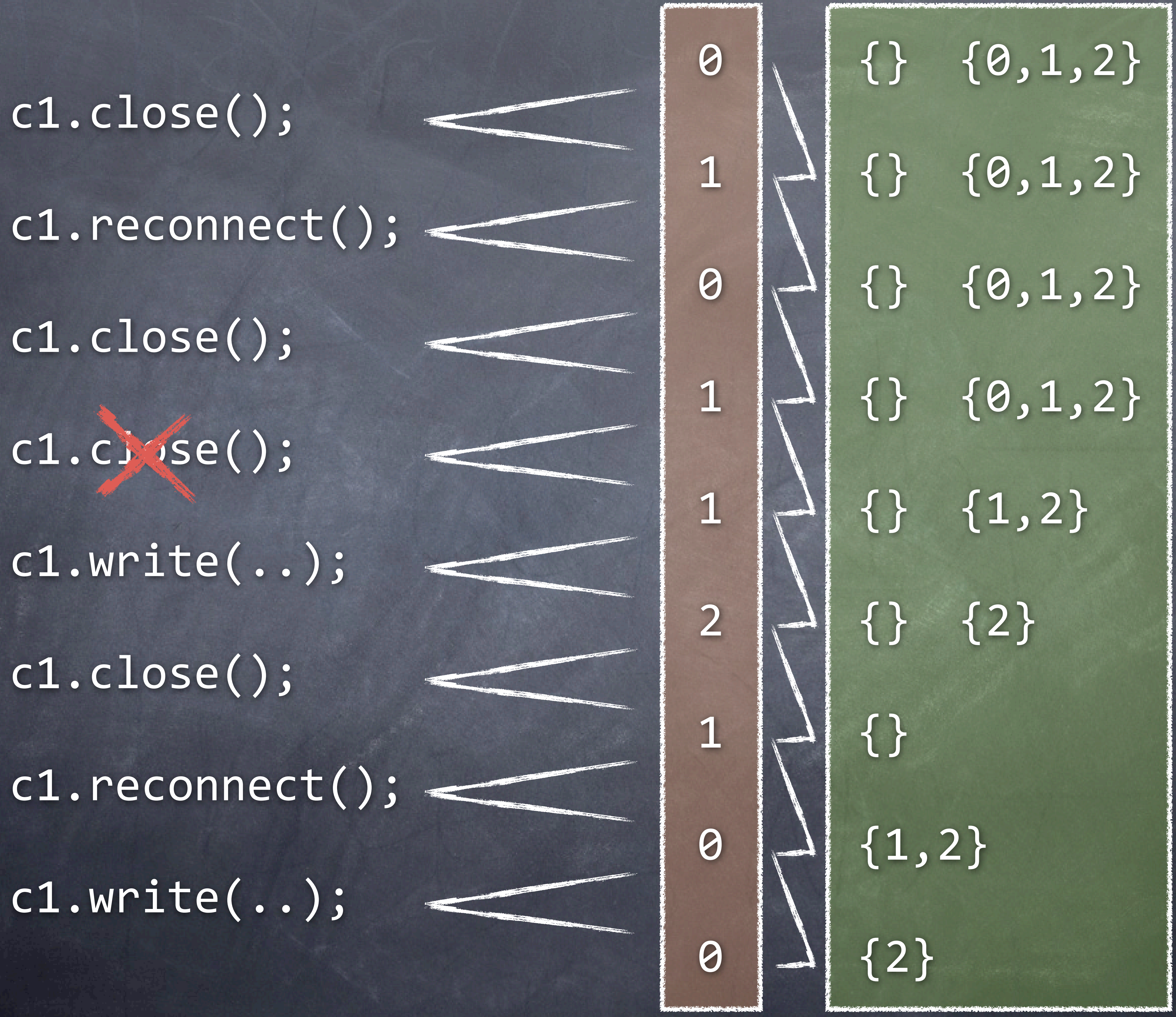
1 {}

? c1.reconnect();

0 {1,2}

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

0 {2}



c1.close();

c1.reconnect();

c1.close();

~~c1.close();~~

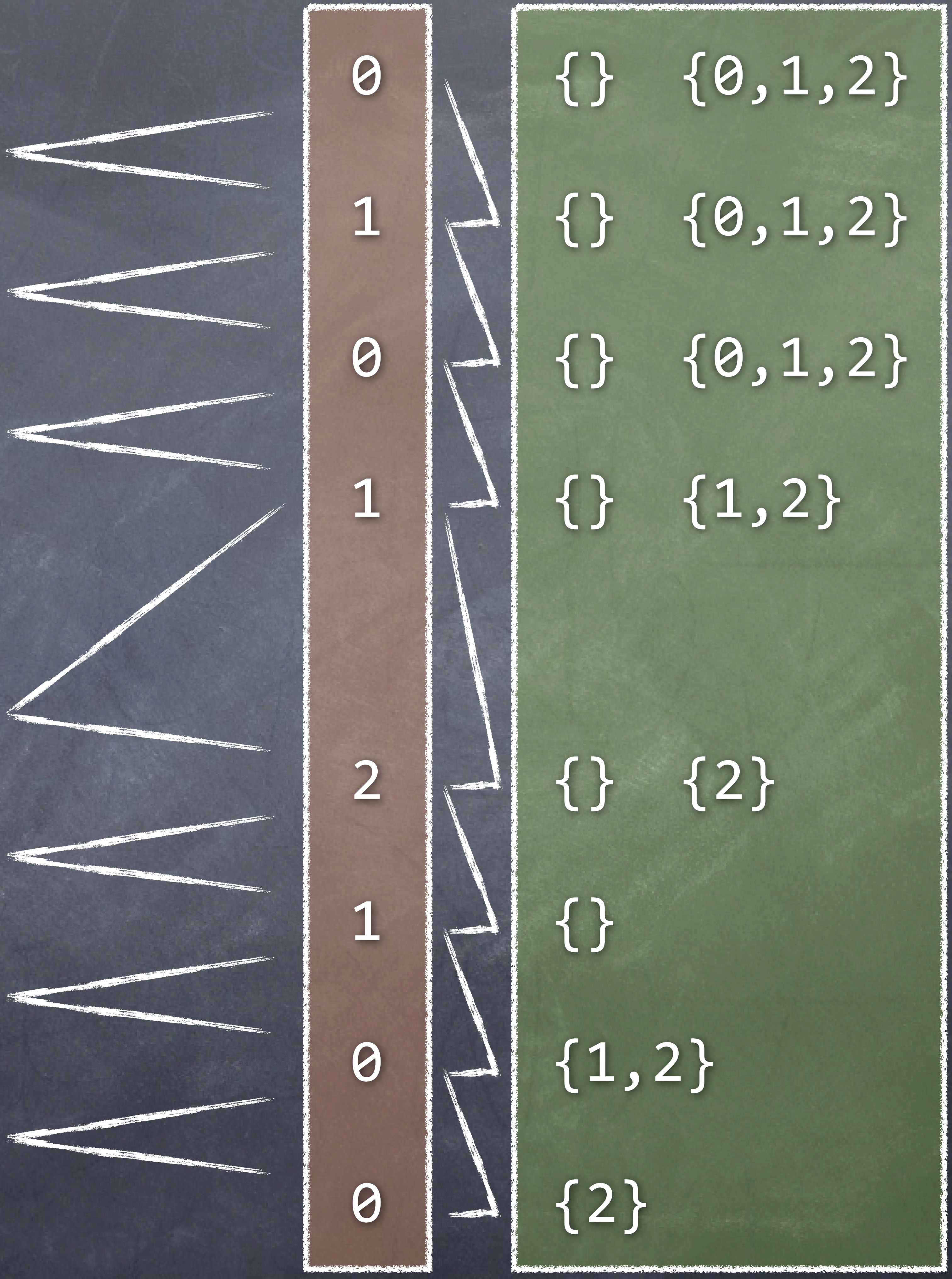
c1.write(..);

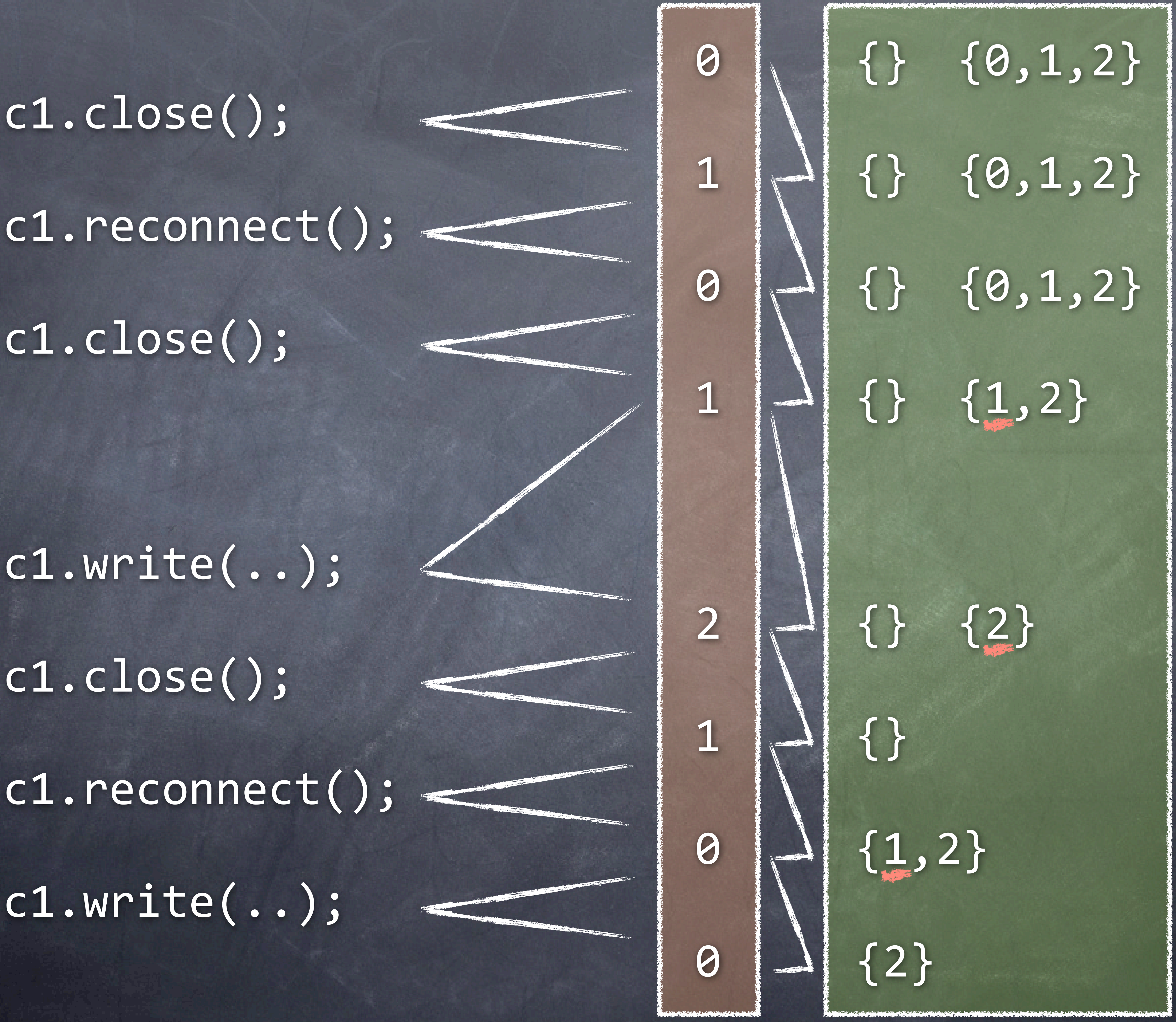
c1.close();

c1.reconnect();

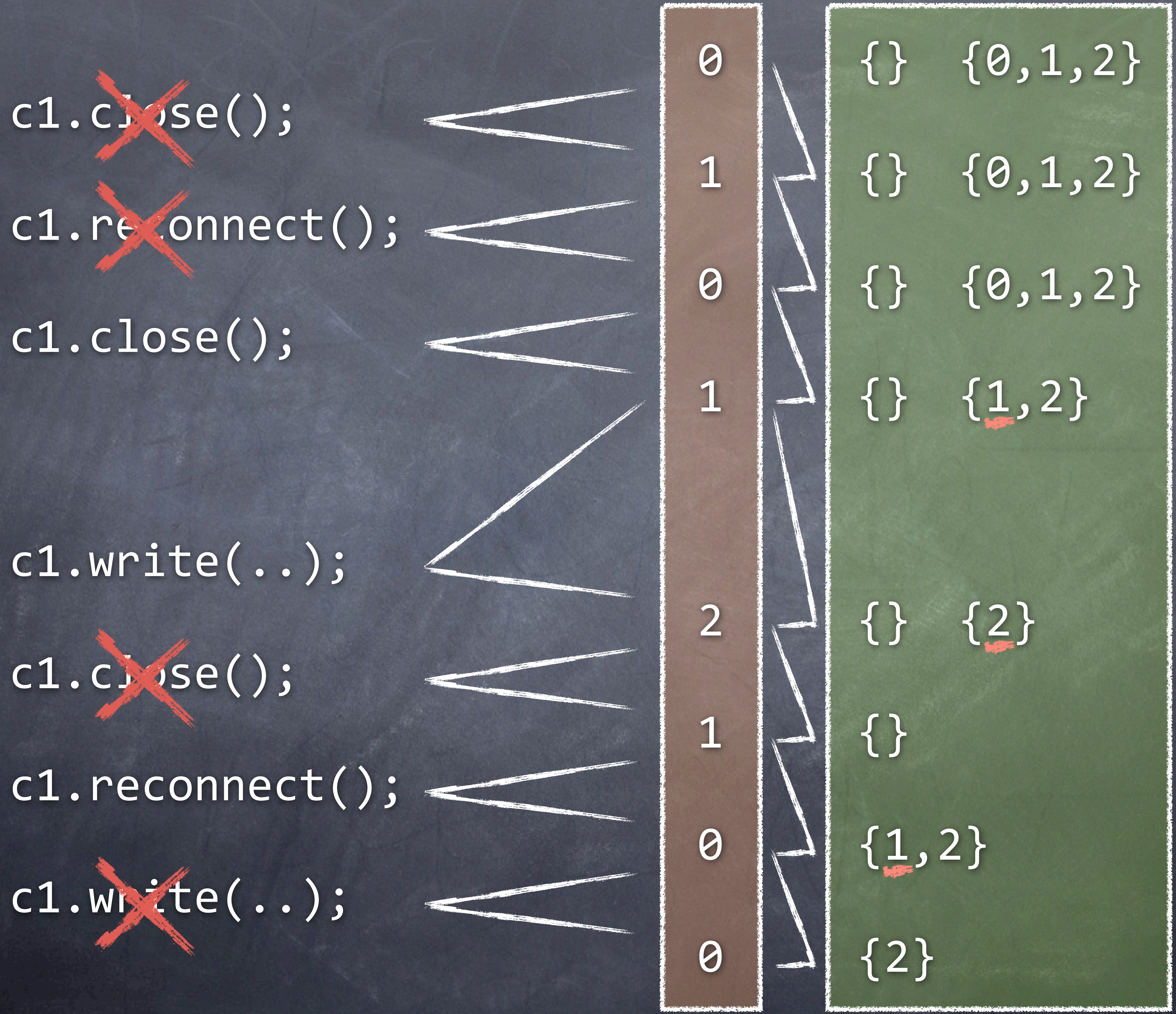
c1.write(..);

```
c1.close();

c1.reconnect();

c1.close();


c1.write(..);

c1.close();

c1.reconnect();

c1.write(..);
```





```
c1.close();
c1.reconnect();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
```



~~c1.close();~~

~~c1.reconnect();~~

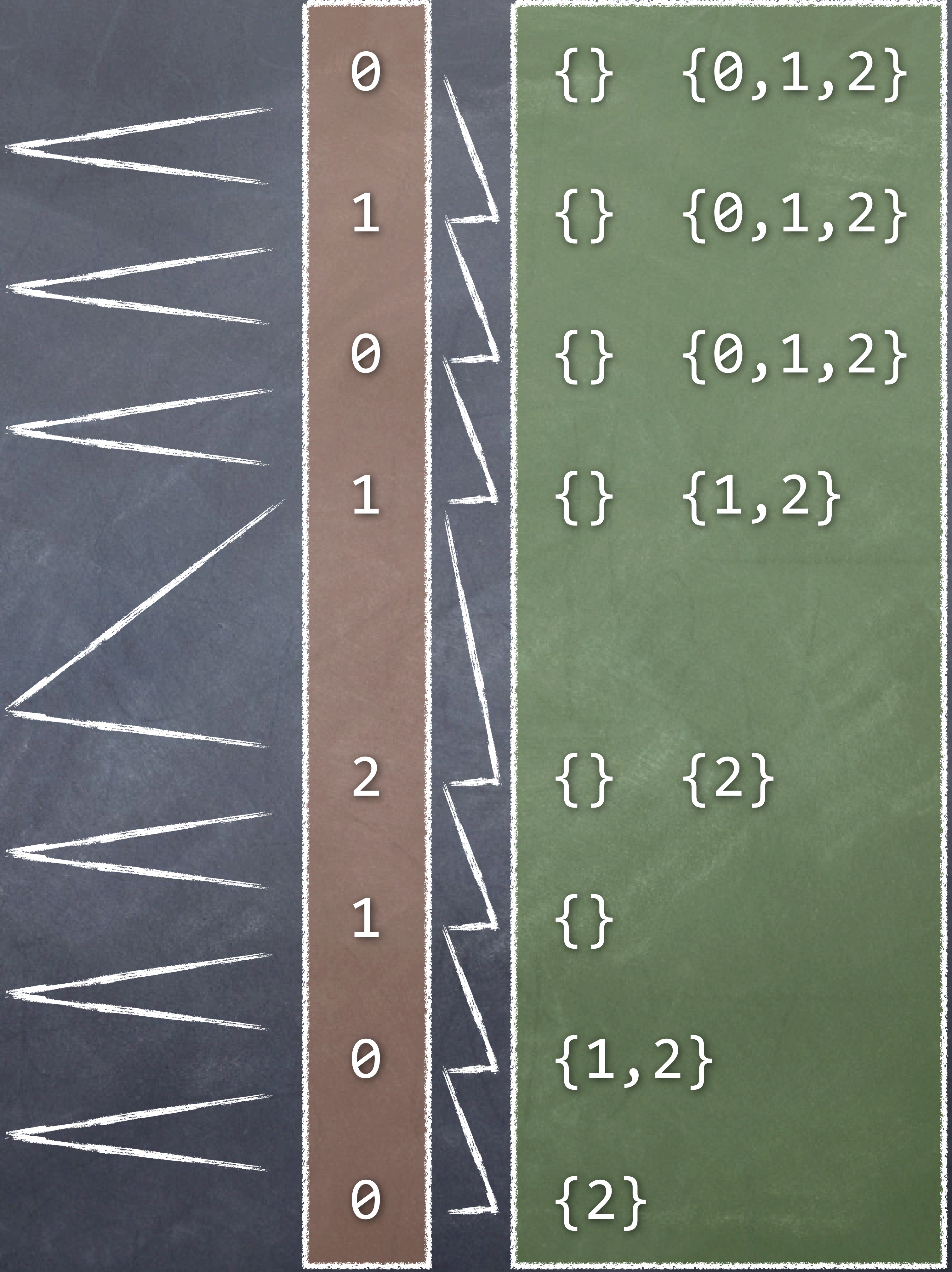
c1.close();

c1.write(..);

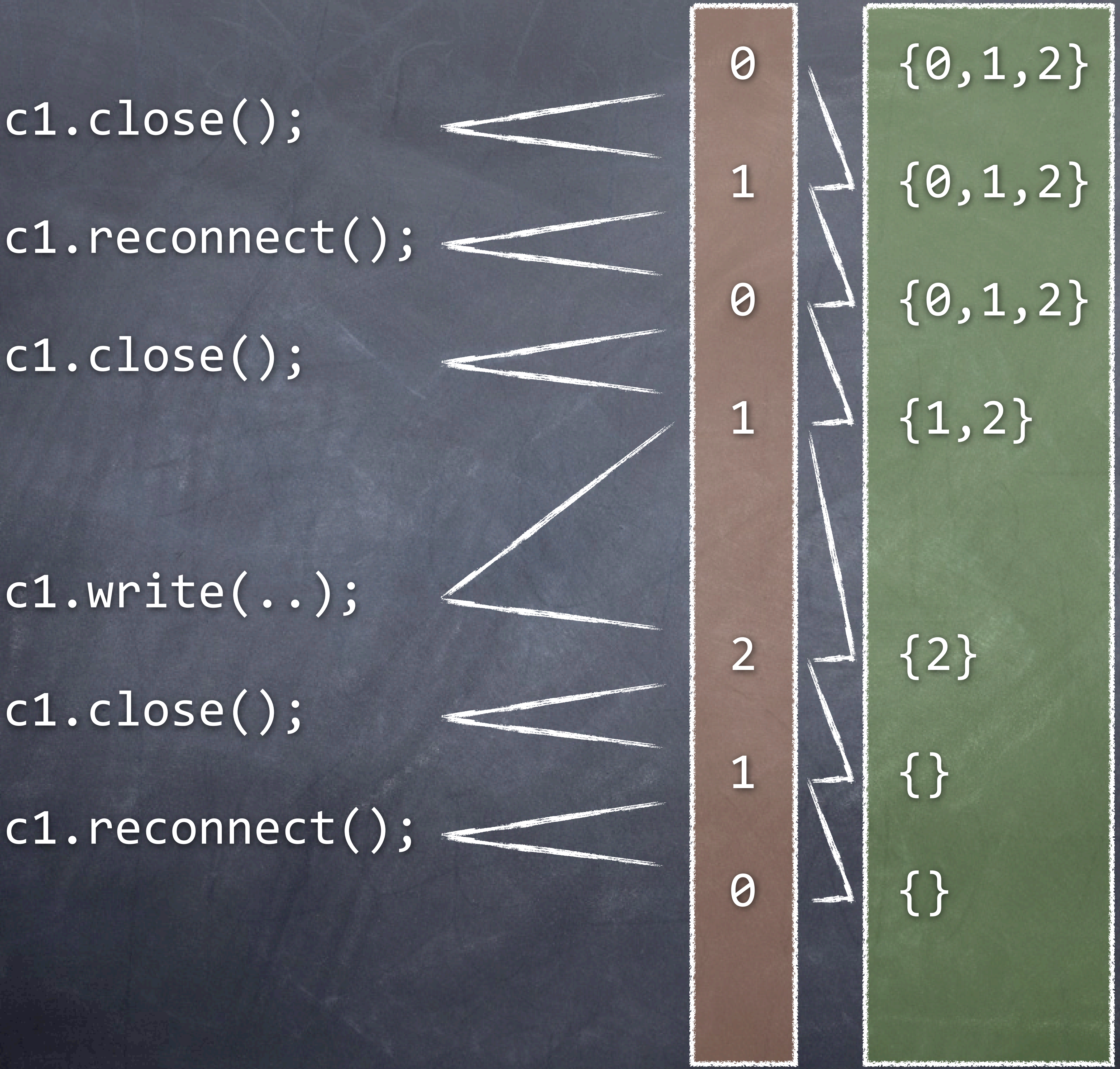
~~c1.close();~~

c1.reconnect();

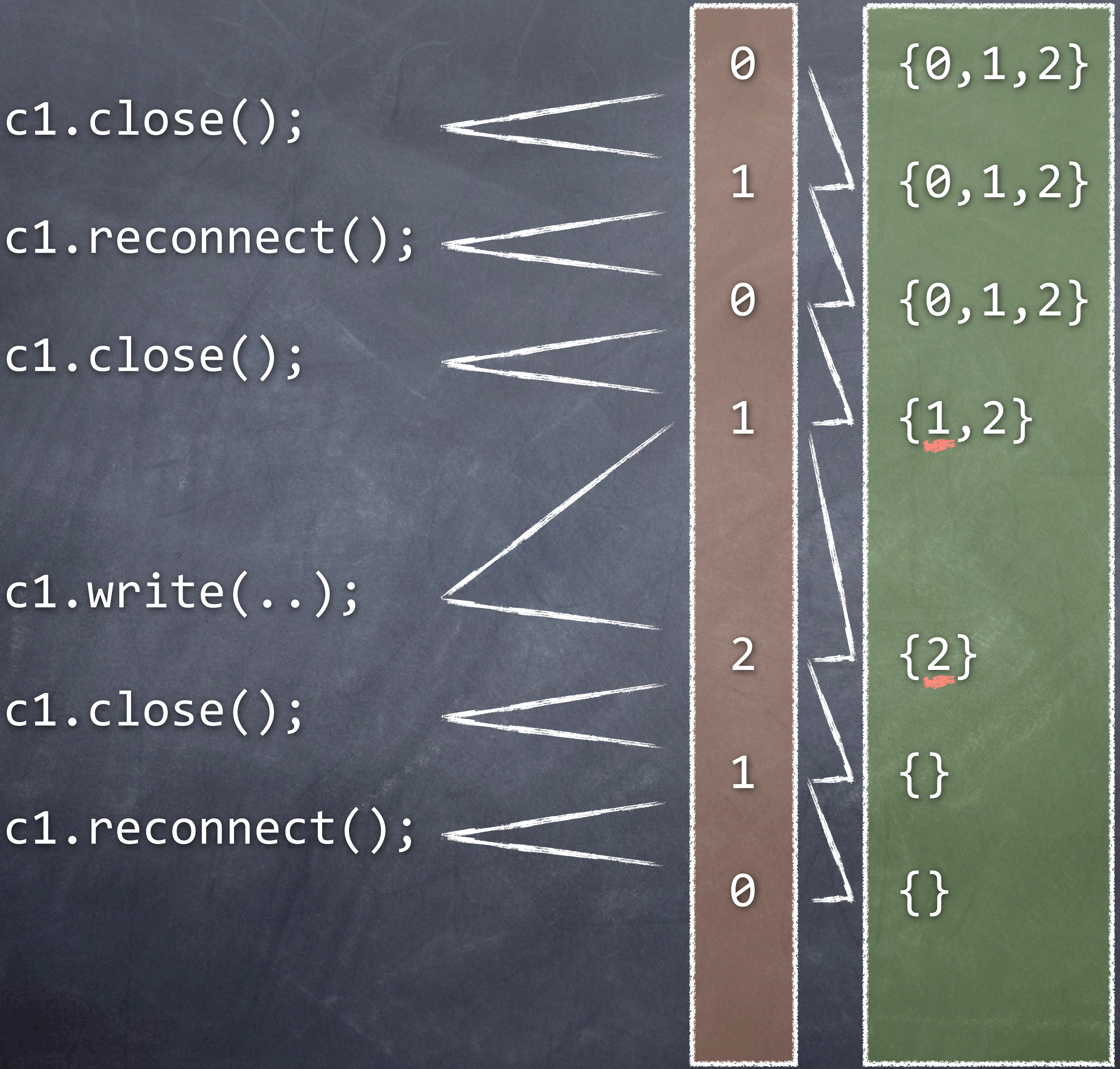
~~c1.write(..);~~



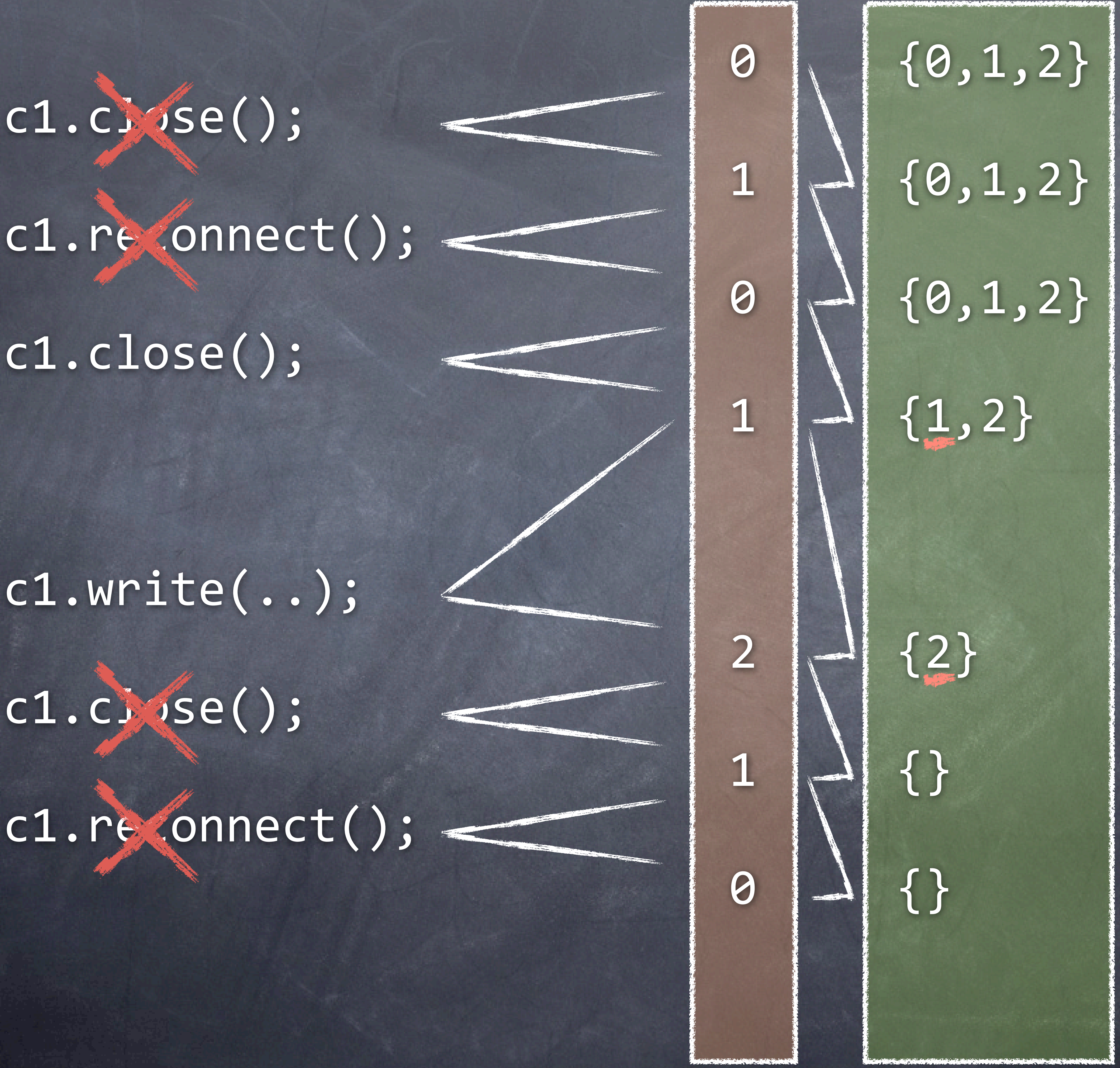
```
c1.close();

c1.reconnect();

c1.close();


c1.write(..);

c1.close();

c1.reconnect();

c1.write(..);
```



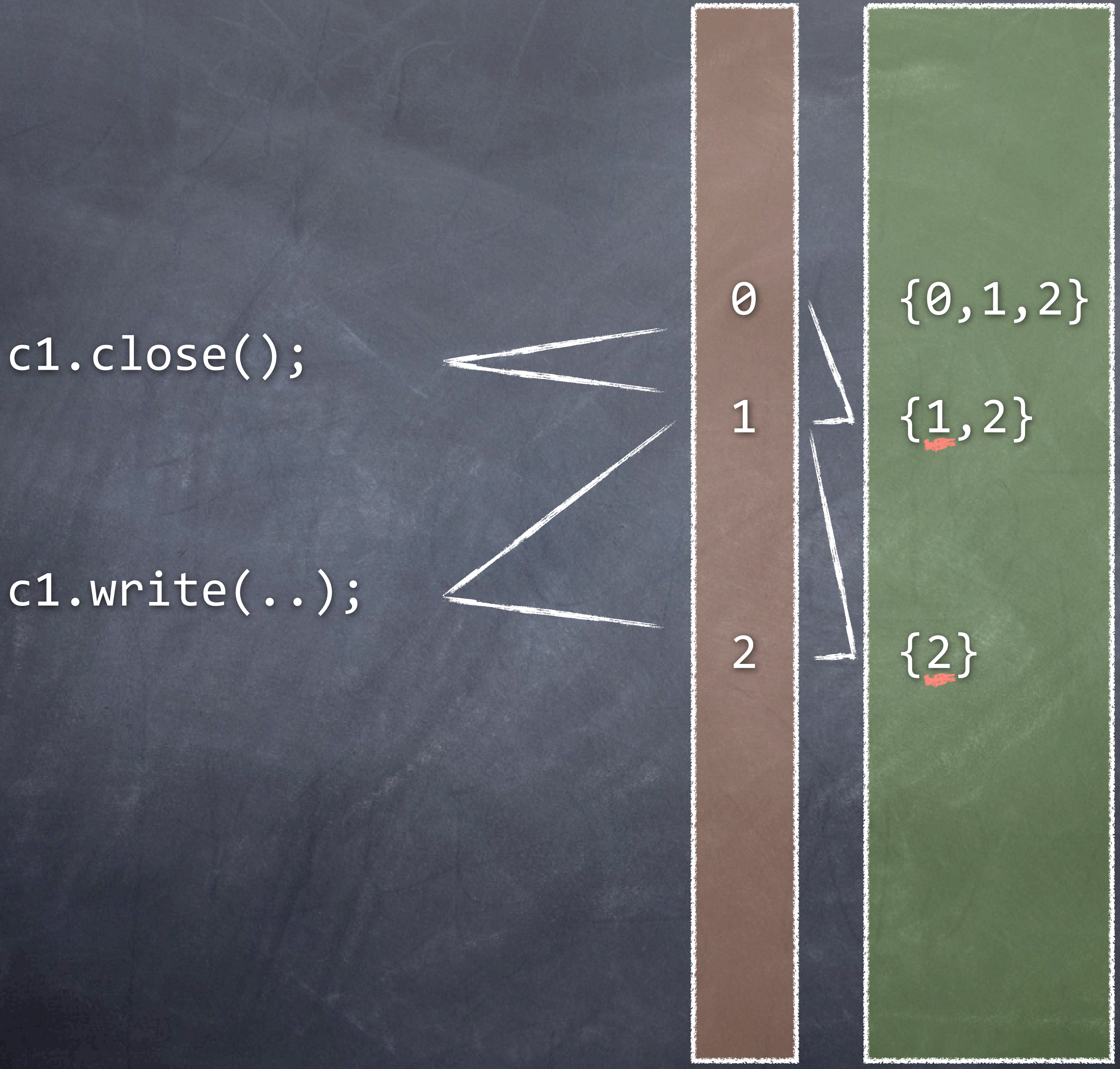
```
c1.close();

c1.reconnect();

c1.close();


c1.write(..);

c1.close();

c1.reconnect();
```



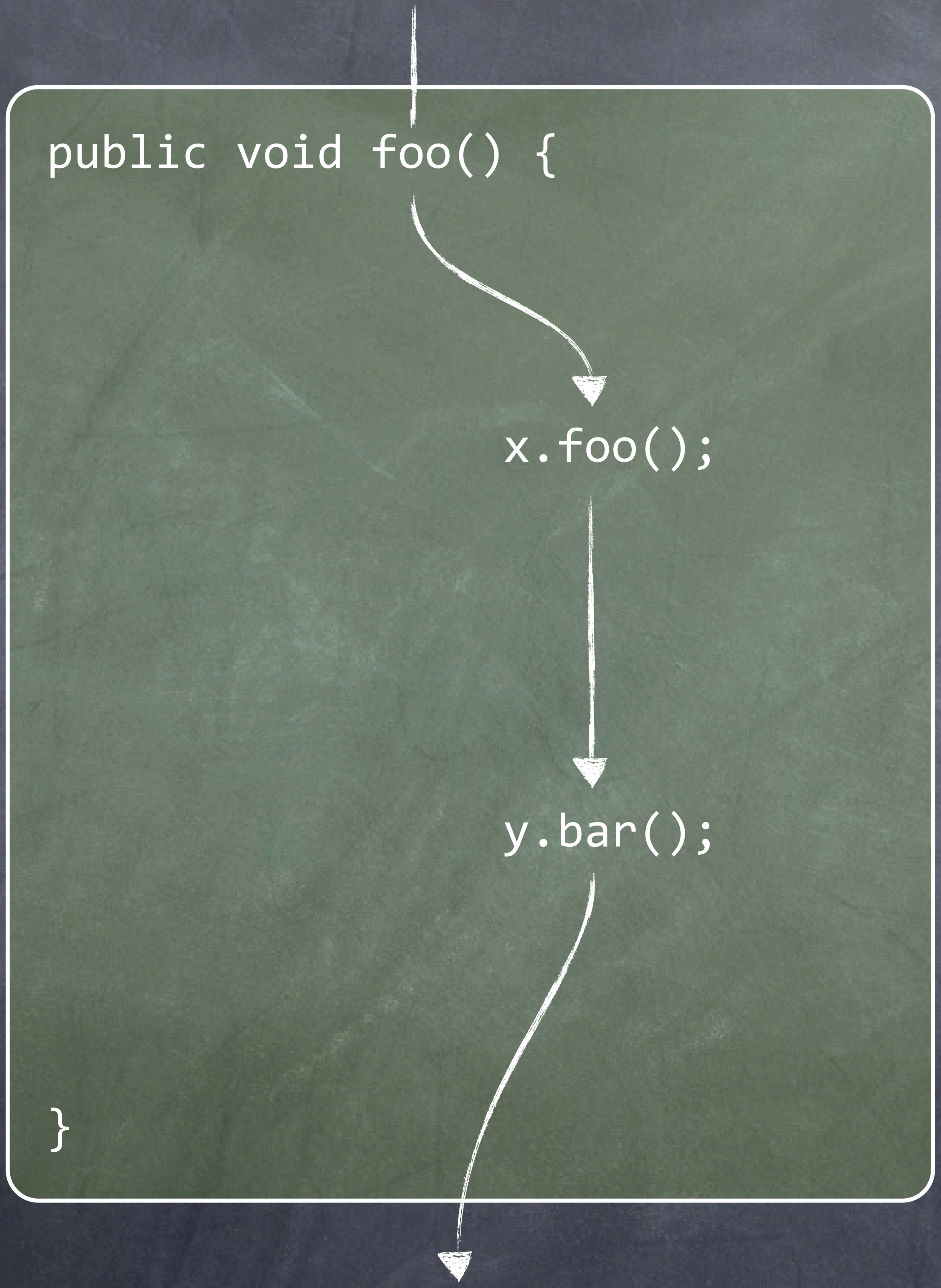
```
c1.close();

c1.reconnect();

c1.close();


c1.write(..);

c1.close();

c1.reconnect();
```



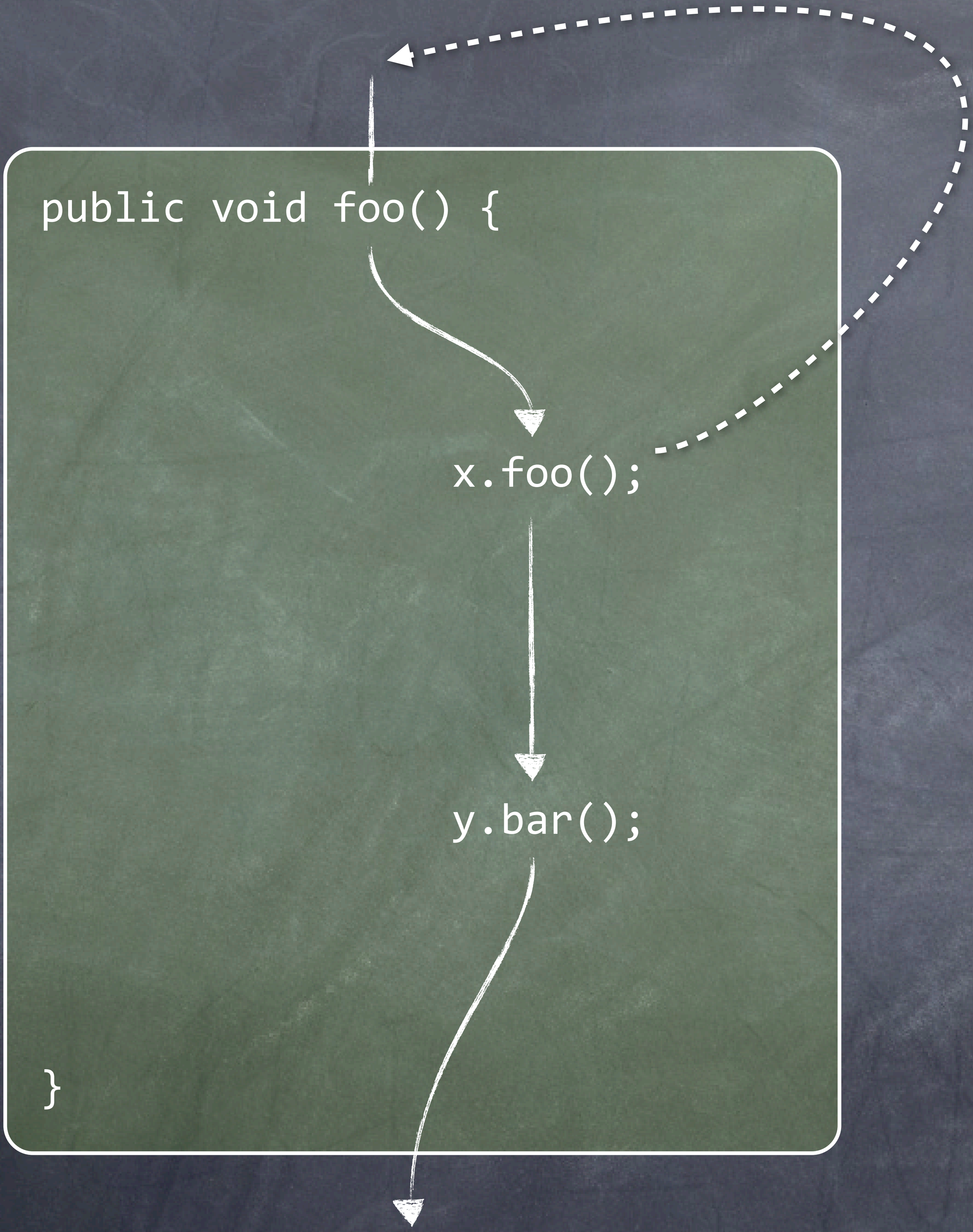
```
~~c1.close();~~
~~c1.reconnect();~~
c1.close();
c1.write(..);
~~c1.close();~~
~~c1.reconnect();~~
```



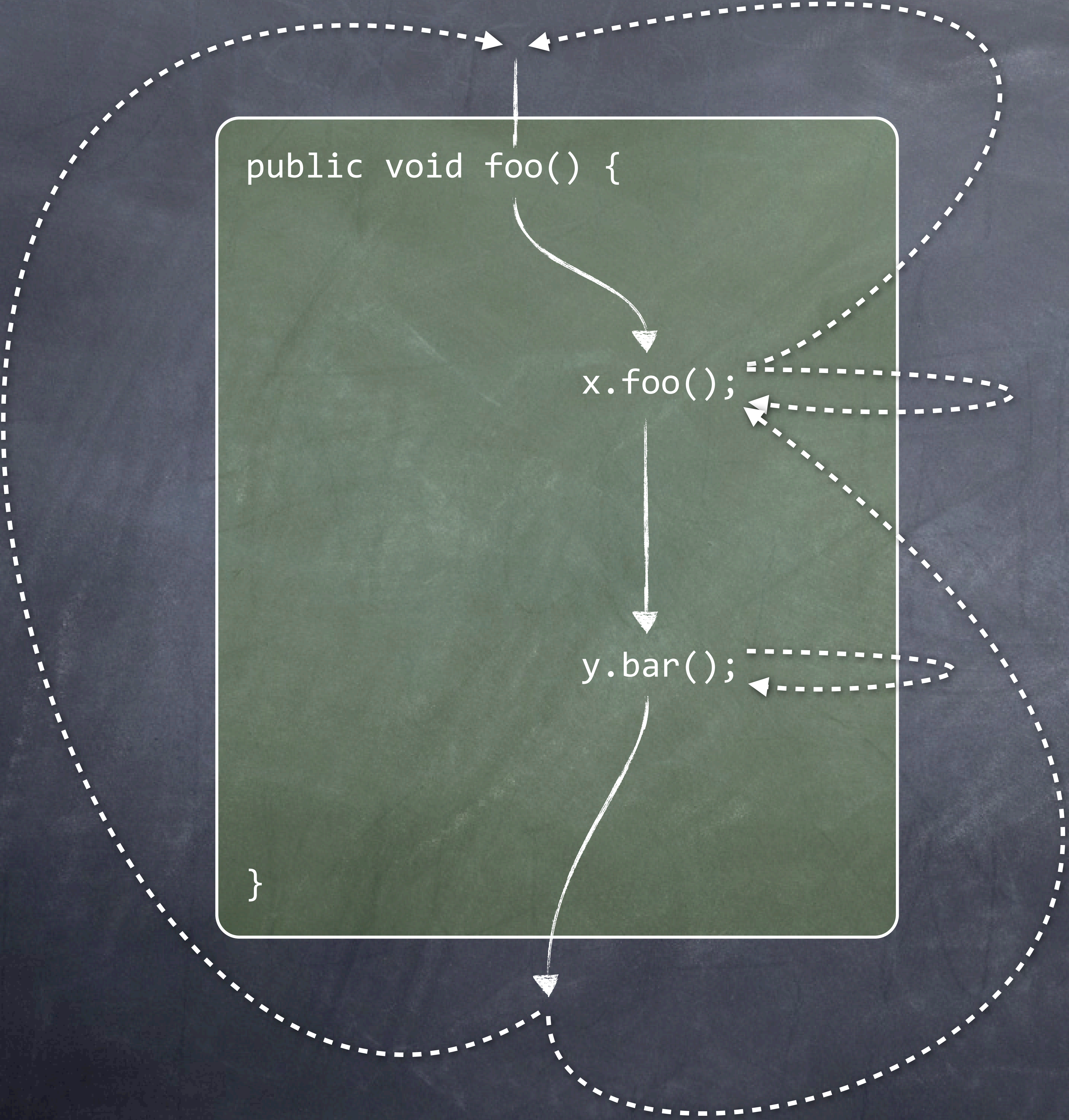
```
c1.close();
c1.write(..);
```



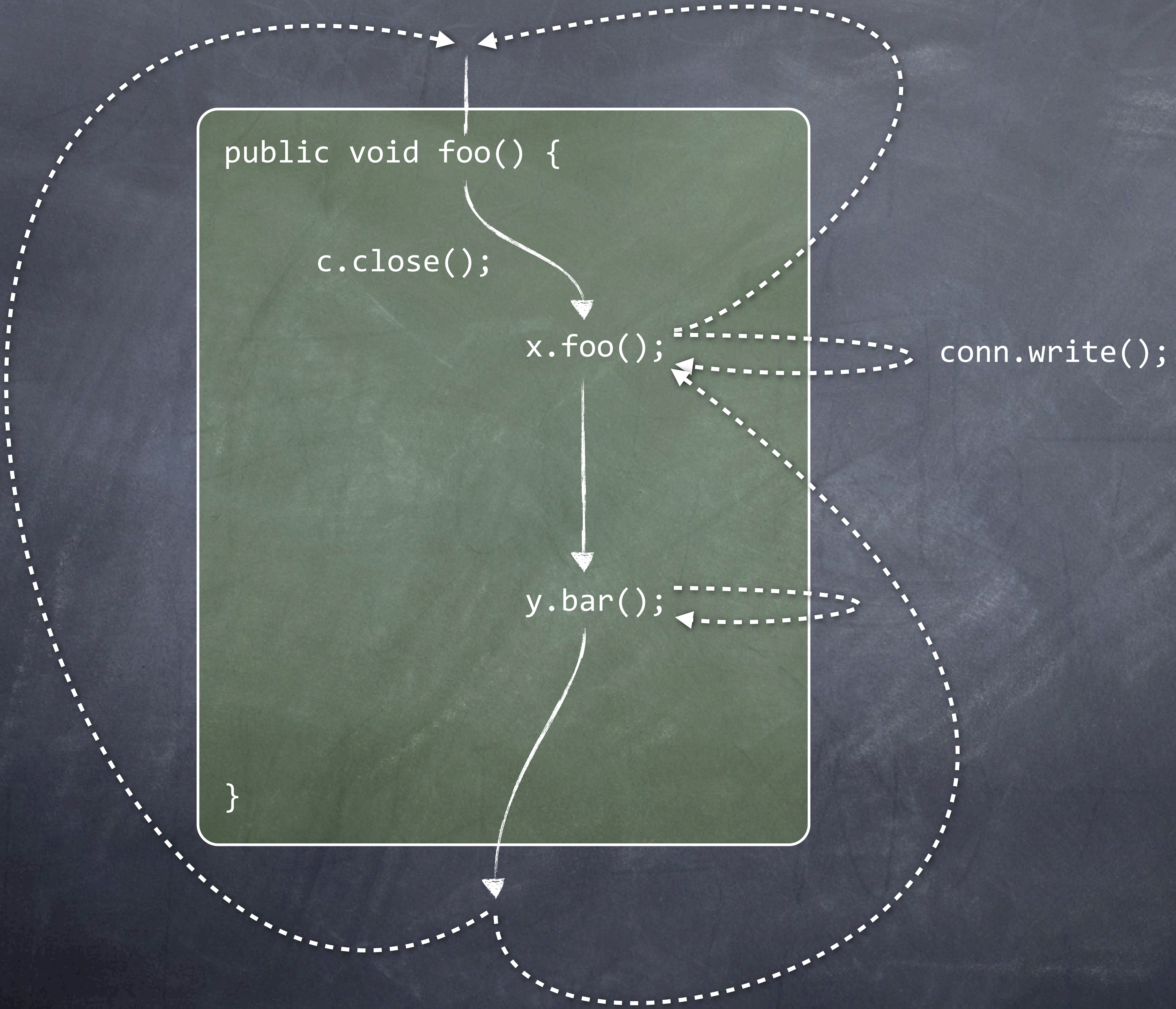
```
public void foo() {

                x.foo();

                y.bar();

}
```



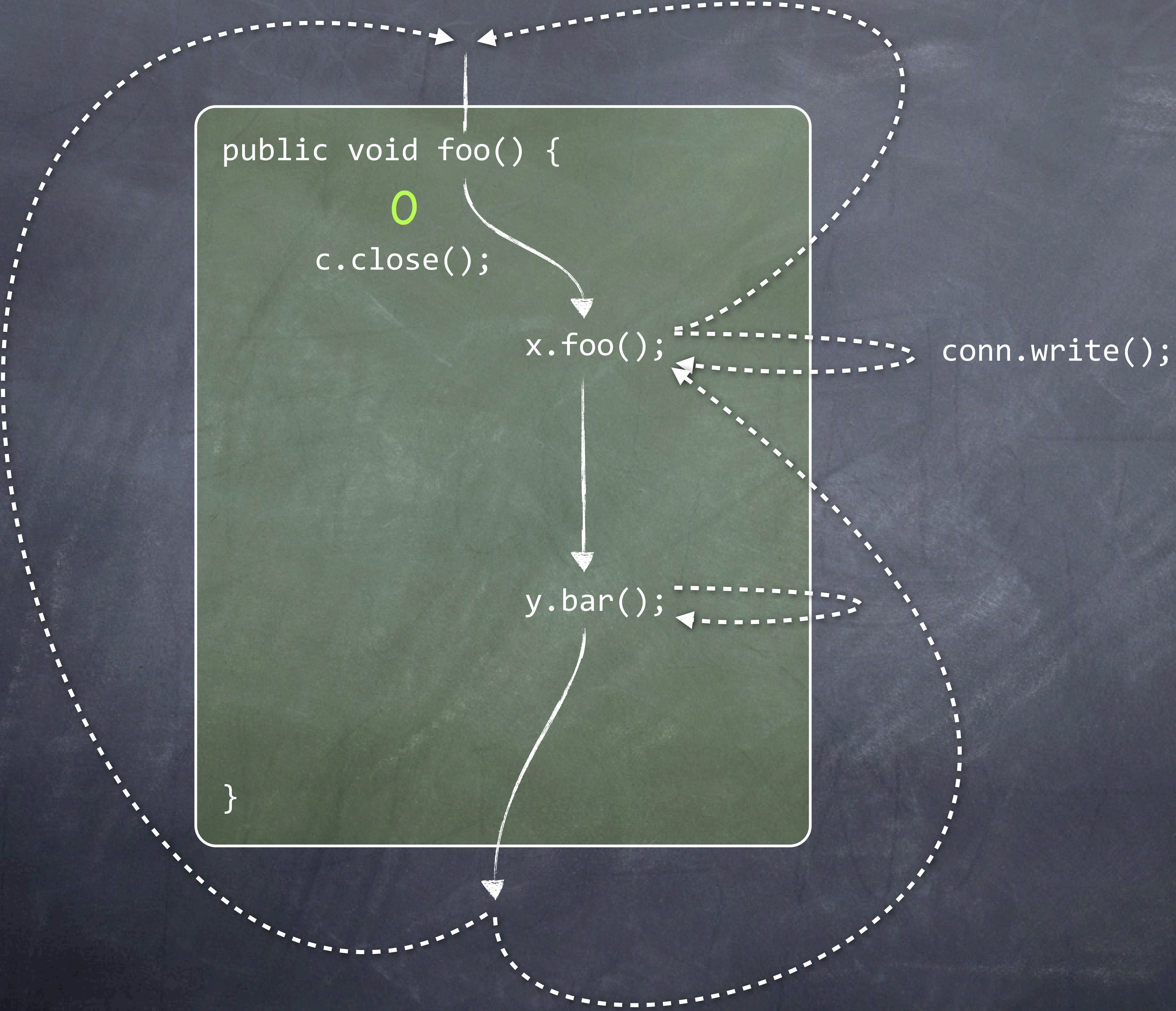
```
public void foo() {

          x.foo();

          y.bar();

}
```



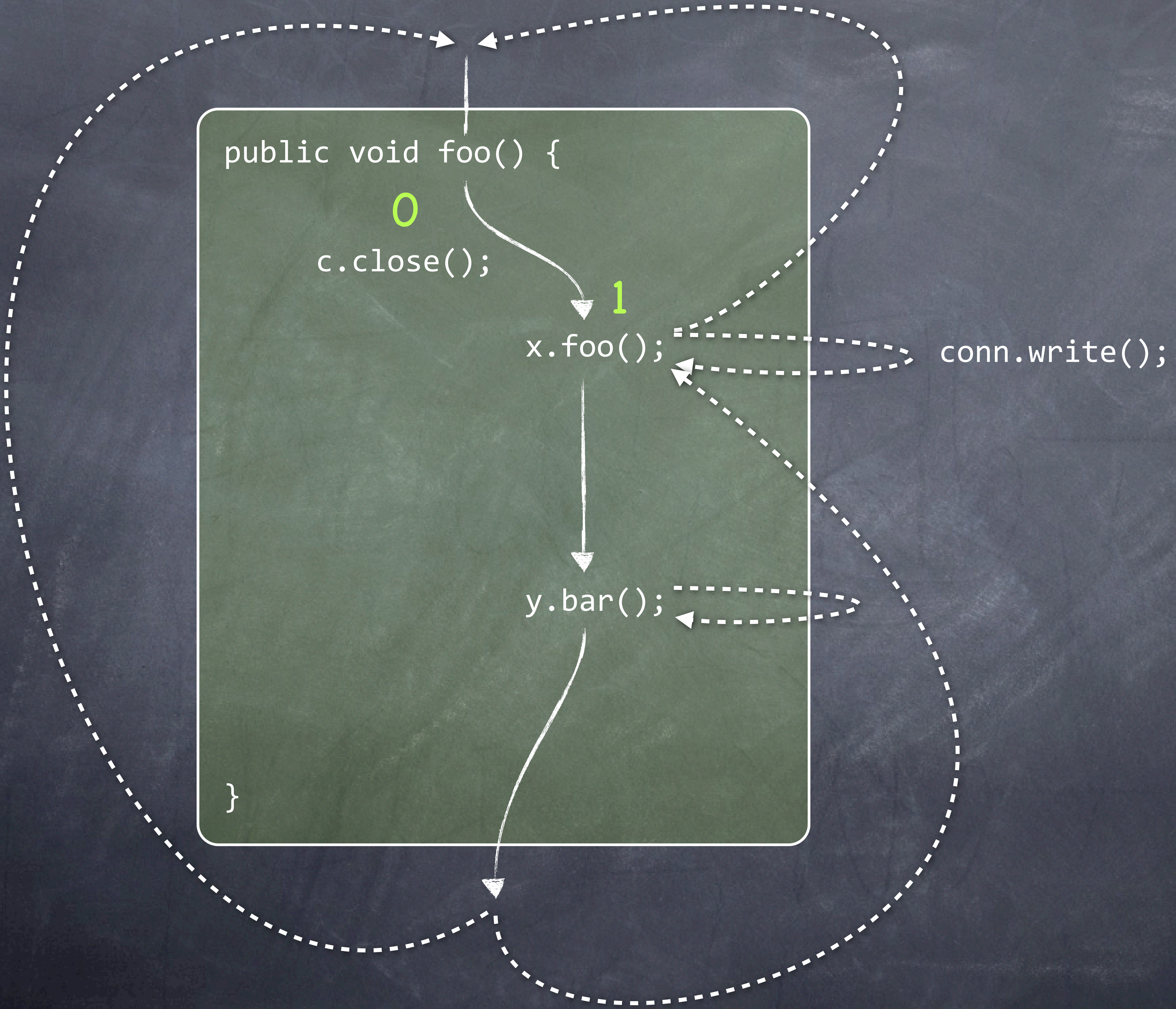
```
public void foo() {

                x.foo();

                y.bar();

}
```



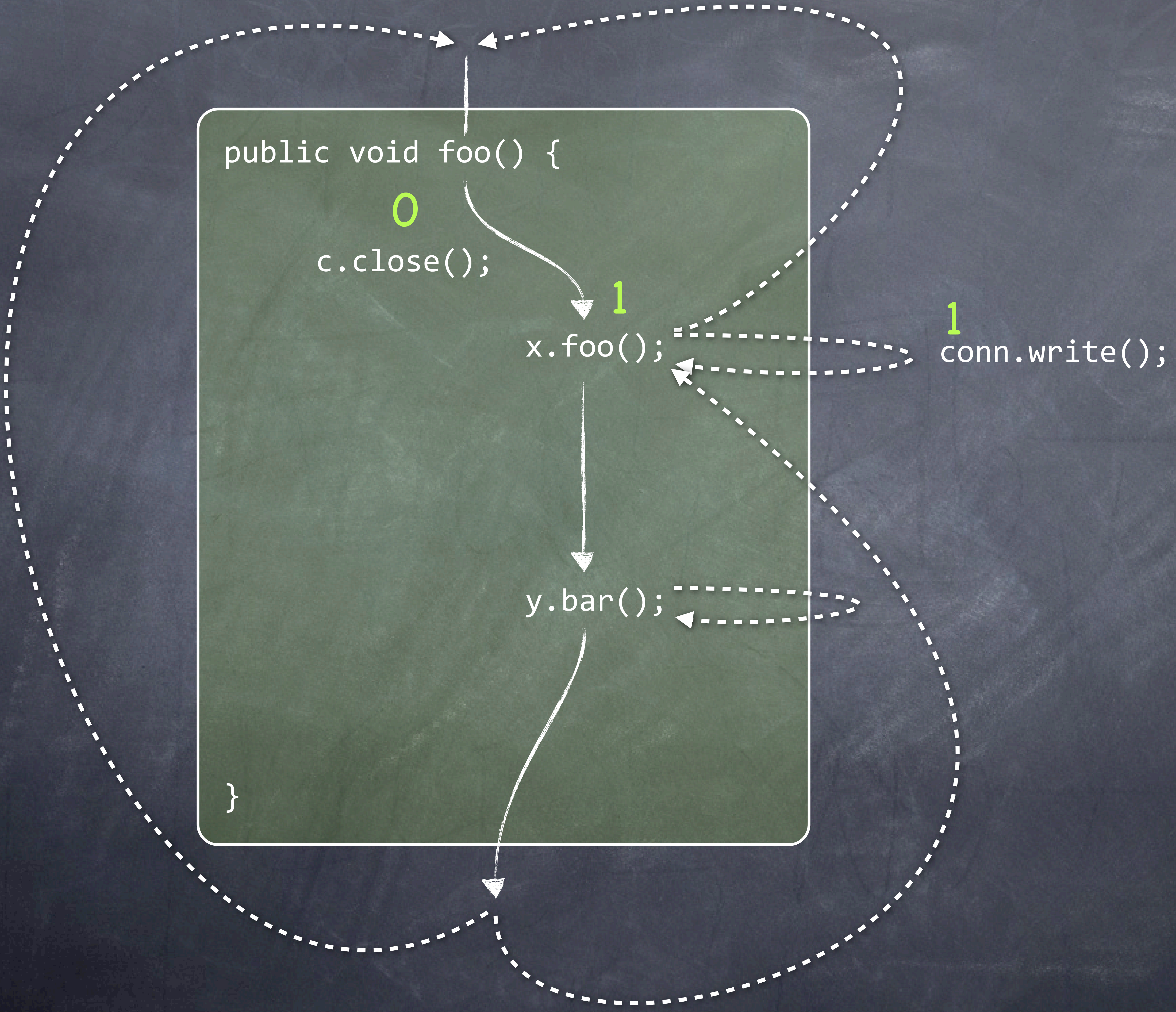
```
public void foo() {

     c.close();

                x.foo();                  conn.write();


                y.bar();


}
```



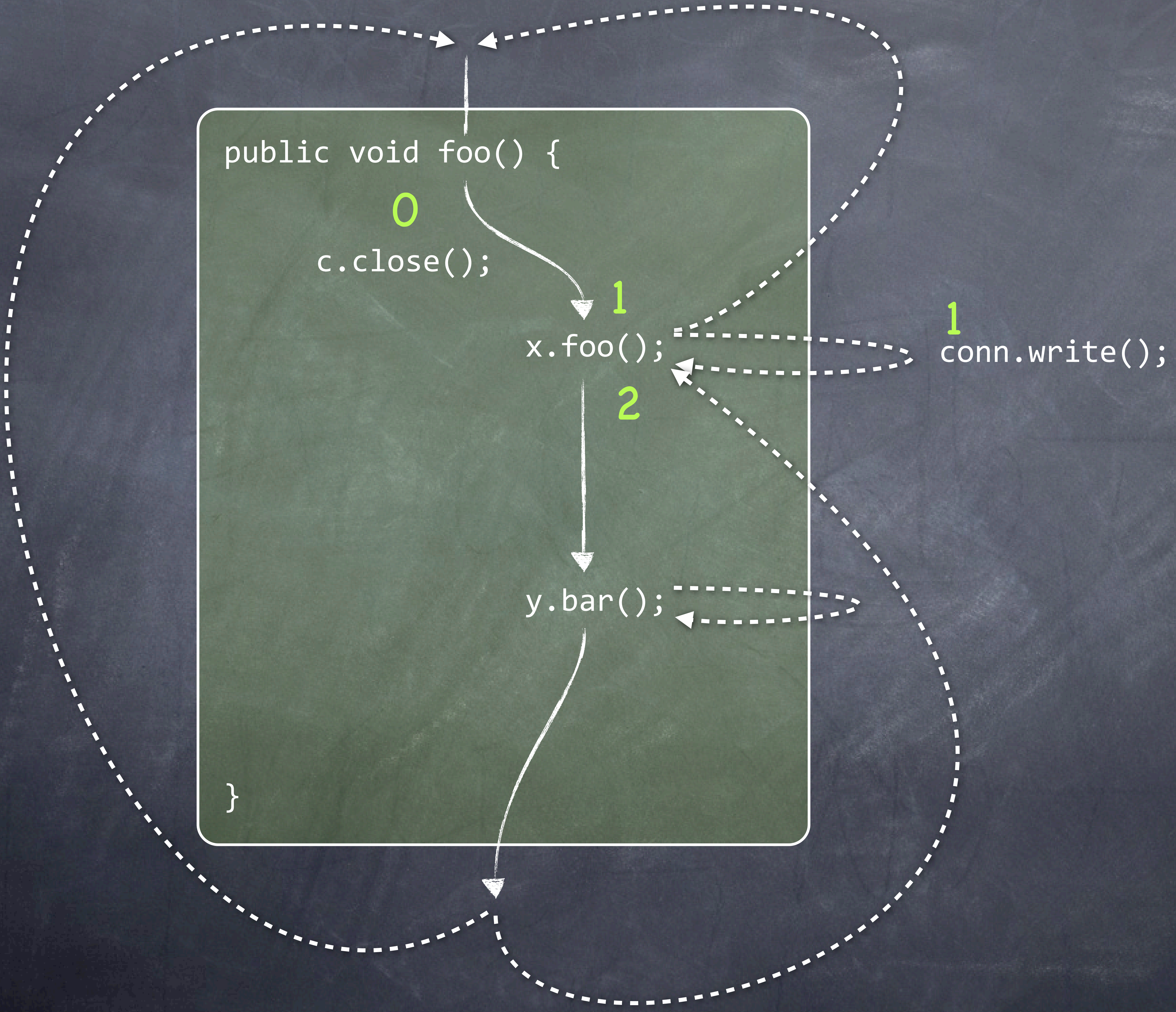
```
public void foo() {
        O
    c.close();

              x.foo();        conn.write();

              y.bar();

}
```



```
public void foo() {

    c.close();

                 x.foo();          conn.write();



                 y.bar();



}
```



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



```
public void foo() {

      c.close();

                 x.foo();                         conn.write();



                 y.bar();



}
```

Tested Properties

ASyncContainsAll	FailSafeIterMap
ASyncIterC	HasNextElem
ASyncIterM	HasNext
FailSafeEnum	LeakingSync
FailSafeEnumHT	Reader
FailSafeIter	Writer

Benchmark programs

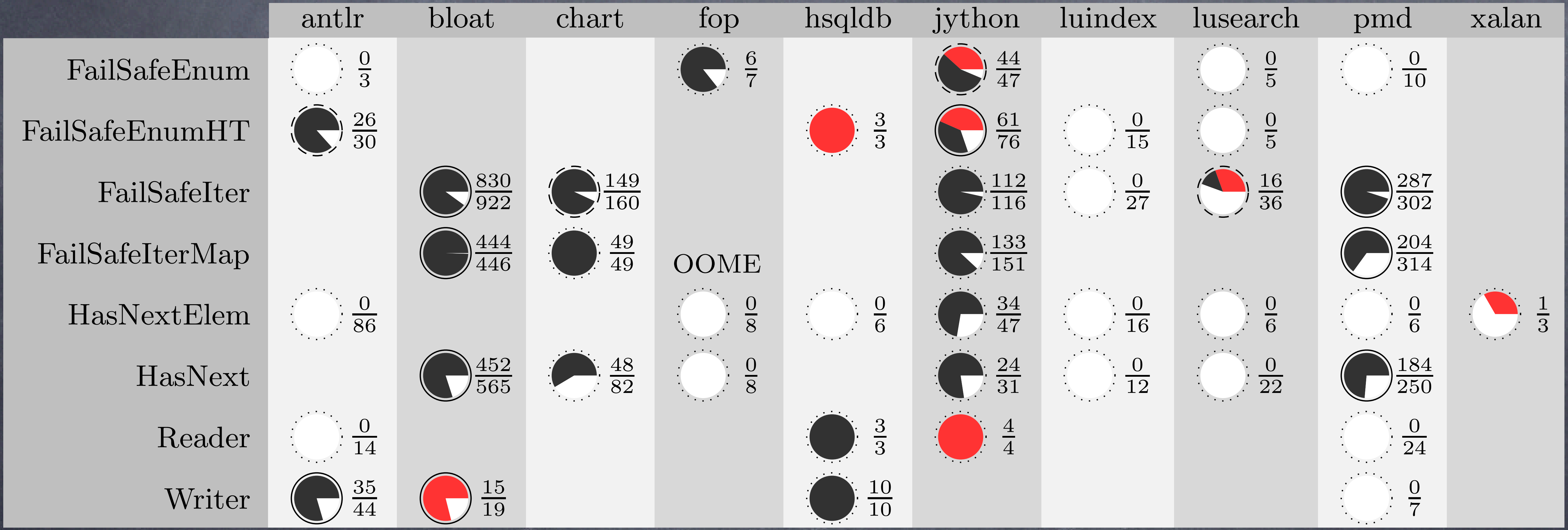
antlr	jython
bloat	luindex
chart	lusearch
fop	pmd
hsqldb	xalan

(whole DaCapo benchmark suite, except eclipse)

Overall success

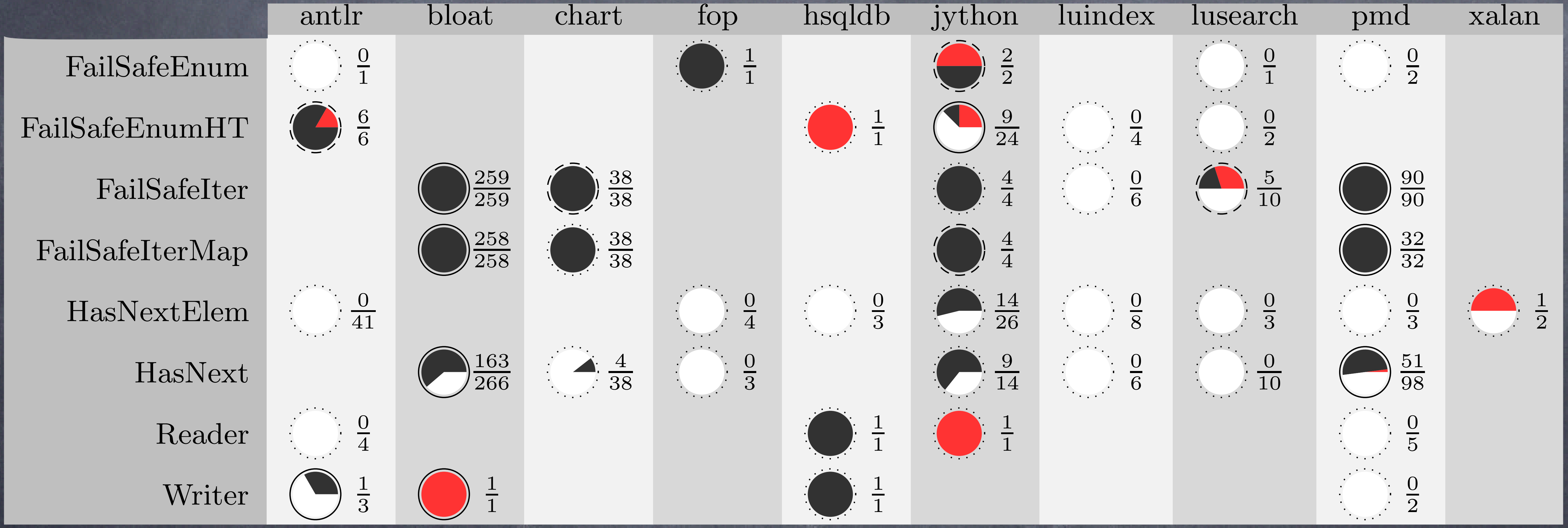
	antlr	bloat	chart	fop	hsqldb	jython	luindex	lusearch	pmd	xalan
ASyncContainsAll		$\frac{0}{71}$	$\frac{0}{6}$			$\frac{0}{31}$	$\frac{0}{18}$	$\frac{0}{18}$	$\frac{0}{10}$	
ASyncIterC		$\frac{0}{1621}$	$\frac{0}{498}$	$\frac{0}{146}$	$\frac{0}{33}$	$\frac{0}{128}$	$\frac{0}{149}$	$\frac{0}{149}$	$\frac{0}{671}$	
ASyncIterM		$\frac{0}{1684}$	$\frac{0}{507}$	$\frac{0}{176}$	$\frac{0}{39}$	$\frac{0}{138}$	$\frac{0}{152}$	$\frac{0}{152}$	$\frac{0}{718}$	
FailSafeEnum	$\frac{0}{76}$	$\frac{0}{3}$	$\frac{0}{1}$	$\frac{6}{18}$	$\frac{0}{120}$	$\frac{44}{110}$	$\frac{0}{61}$	$\frac{0}{61}$	$\frac{0}{21}$	$\frac{0}{222}$
FailSafeEnumHT	$\frac{26}{133}$	$\frac{0}{102}$	$\frac{0}{44}$	$\frac{0}{205}$	$\frac{3}{114}$	$\frac{61}{153}$	$\frac{0}{37}$	$\frac{0}{37}$	$\frac{0}{100}$	$\frac{0}{319}$
FailSafeIter	$\frac{0}{23}$	$\frac{830}{1394}$	$\frac{149}{510}$	$\frac{0}{288}$	$\frac{0}{112}$	$\frac{112}{253}$	$\frac{0}{217}$	$\frac{16}{217}$	$\frac{287}{546}$	$\frac{0}{158}$
FailSafeIterMap	$\frac{0}{130}$	$\frac{444}{1180}$	$\frac{49}{374}$	OOME	$\frac{0}{252}$	$\frac{133}{250}$	$\frac{0}{136}$	$\frac{0}{136}$	$\frac{204}{583}$	$\frac{0}{540}$
HasNextElem	$\frac{0}{117}$	$\frac{0}{4}$		$\frac{0}{12}$	$\frac{0}{53}$	$\frac{34}{64}$	$\frac{0}{22}$	$\frac{0}{22}$	$\frac{0}{11}$	$\frac{1}{63}$
HasNext		$\frac{452}{849}$	$\frac{48}{248}$	$\frac{0}{72}$	$\frac{0}{16}$	$\frac{24}{63}$	$\frac{0}{74}$	$\frac{0}{74}$	$\frac{184}{346}$	
LeakingSync	$\frac{0}{170}$	$\frac{0}{1994}$	$\frac{0}{920}$	$\frac{0}{2347}$	$\frac{0}{528}$	$\frac{0}{1082}$	$\frac{0}{629}$	$\frac{0}{629}$	$\frac{0}{986}$	$\frac{0}{1005}$
Reader	$\frac{0}{50}$	$\frac{0}{7}$	$\frac{0}{65}$	$\frac{0}{102}$	$\frac{3}{1216}$	$\frac{4}{139}$	$\frac{0}{226}$	$\frac{0}{226}$	$\frac{0}{102}$	$\frac{0}{106}$
Writer	$\frac{35}{171}$	$\frac{15}{563}$	$\frac{0}{70}$	$\frac{0}{429}$	$\frac{10}{1378}$	$\frac{0}{462}$	$\frac{0}{146}$	$\frac{0}{146}$	$\frac{0}{62}$	$\frac{0}{751}$

NSA over QC and OSA



	antlr	bloat	chart	fop	hsqldb	jython	luindex	lusearch	pmd	xalan
FailSafeEnum	$\frac{0}{3}$			$\frac{6}{7}$		$\frac{44}{47}$		$\frac{0}{5}$	$\frac{0}{10}$	
FailSafeEnumHT	$\frac{26}{30}$				$\frac{3}{3}$	$\frac{61}{76}$	$\frac{0}{15}$	$\frac{0}{5}$		
FailSafeIter		$\frac{830}{922}$	$\frac{149}{160}$			$\frac{112}{116}$	$\frac{0}{27}$	$\frac{16}{36}$	$\frac{287}{302}$	
FailSafeIterMap		$\frac{444}{446}$	$\frac{49}{49}$	OOME		$\frac{133}{151}$			$\frac{204}{314}$	
HasNextElem	$\frac{0}{86}$			$\frac{0}{8}$	$\frac{0}{6}$	$\frac{34}{47}$	$\frac{0}{16}$	$\frac{0}{6}$	$\frac{0}{6}$	$\frac{1}{3}$
HasNext		$\frac{452}{565}$	$\frac{48}{82}$	$\frac{0}{8}$		$\frac{24}{31}$	$\frac{0}{12}$	$\frac{0}{22}$	$\frac{184}{250}$	
Reader	$\frac{0}{14}$				$\frac{3}{3}$	$\frac{4}{4}$			$\frac{0}{24}$	
Writer	$\frac{35}{44}$	$\frac{15}{19}$			$\frac{10}{10}$				$\frac{0}{7}$	

“Final” shadows only



	antlr	bloat	chart	fop	hsqldb	jython	luindex	lusearch	pmd	xalan
FailSafeEnum	$\frac{0}{1}$			$\frac{1}{1}$		$\frac{2}{2}$		$\frac{0}{1}$	$\frac{0}{2}$	
FailSafeEnumHT	$\frac{6}{6}$				$\frac{1}{1}$	$\frac{9}{24}$	$\frac{0}{4}$	$\frac{0}{2}$		
FailSafeIter		$\frac{259}{259}$	$\frac{38}{38}$			$\frac{4}{4}$	$\frac{0}{6}$	$\frac{5}{10}$	$\frac{90}{90}$	
FailSafeIterMap		$\frac{258}{258}$	$\frac{38}{38}$			$\frac{4}{4}$			$\frac{32}{32}$	
HasNextElem	$\frac{0}{41}$			$\frac{0}{4}$	$\frac{0}{3}$	$\frac{14}{26}$	$\frac{0}{8}$	$\frac{0}{3}$	$\frac{0}{3}$	$\frac{1}{2}$
HasNext		$\frac{163}{266}$	$\frac{4}{38}$	$\frac{0}{3}$		$\frac{9}{14}$	$\frac{0}{6}$	$\frac{0}{10}$	$\frac{51}{98}$	
Reader	$\frac{0}{4}$				$\frac{1}{1}$	$\frac{1}{1}$			$\frac{0}{5}$	
Writer	$\frac{1}{3}$	$\frac{1}{1}$			$\frac{1}{1}$				$\frac{0}{2}$	

Related Work

Kevin Bierhoff and Jonathan Aldrich. Modular typestate checking of aliased objects. In Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications (Montreal, Quebec, Canada, October 21 - 25, 2007). OOPSLA '07. ACM, New York, NY, 301-320.

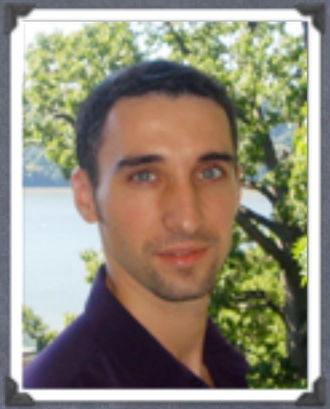
Typestates: Static typing

Related Work

Robert DeLine and Manuel Fähndrich. Typestates for objects. In ECOOP 2004, volume 3086 of Lecture Notes in Computer Science (LNCS), pages 465–490. Springer, June 2004.

Typestates: Static typing

Related Work



Stephen Fink, Eran Yahav, Nurit Dor, G. Ramalingam, and Emmanual Geay. Effective typestate verification in the presence of aliasing. In ISSTA 2006, pages 133–144. ACM Press, July 2006.

Typestate analysis

(just single objects)

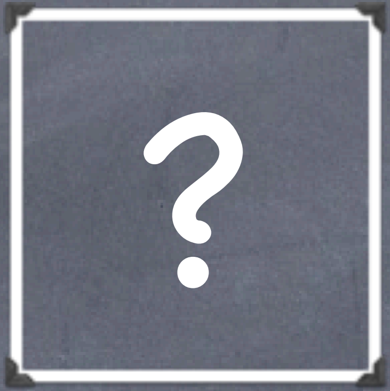
Related Work

Nomair A. Naeem and Ondrej Lhotak. Typestate-like analysis of multiple interacting objects. In OOPSLA 2008, pages 347–366. ACM Press, October 2008.

Typestate analysis

based on tracematches, supports multiple objects, unfortunately unsound (sort of my fault)

Related Work



Matthew B. Dwyer and Rahul Purandare. Residual dynamic typestate analysis: Exploiting static analysis results to reformulate and reduce the cost of dynamic analysis. In ASE 2007, pages 124–133. ACM Press, May 2007.

Residual Typestate analysis

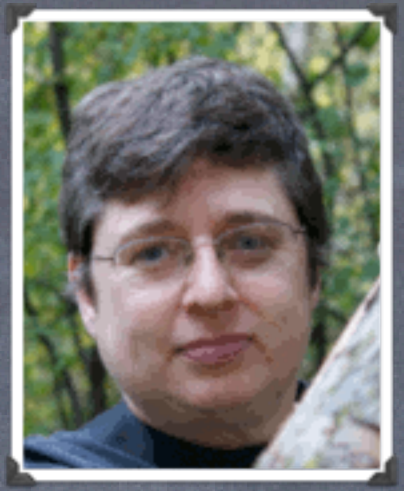
(just single objects)

Related Work: Conclusion

- Clara: First open framework for typestate analysis
- Novel spec. language: Dependency State Machines
- One of few approaches to support combinations of multiple objects
- Apart from Dwyer/Purandare only approach to hybrid typestate analysis
- NSA: New notion of continuation-equivalent states

Thanks!

McGill

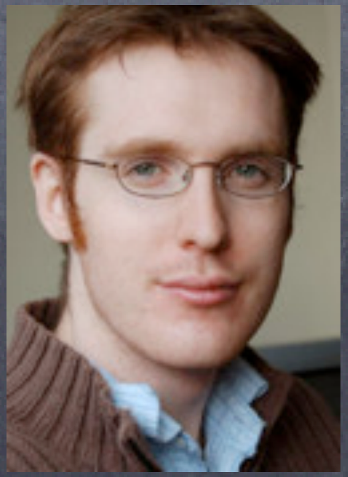


Laurie Hendren

Patrick Lam

Oxford

Pavel Avgustinov



Julian Tibble

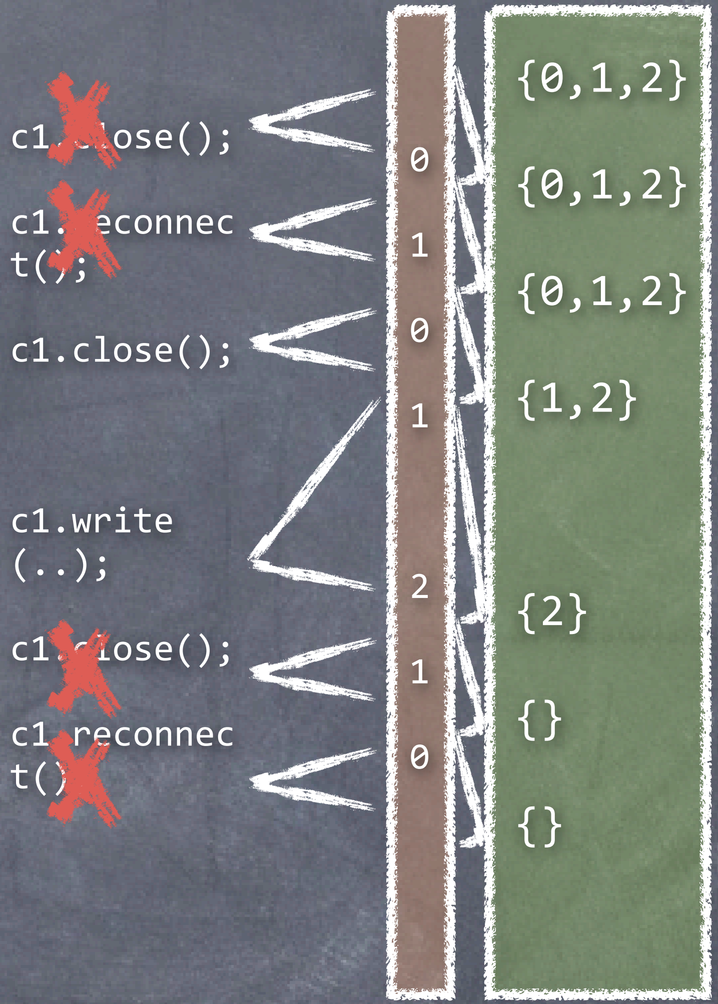
Conclusion

Conclusion

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

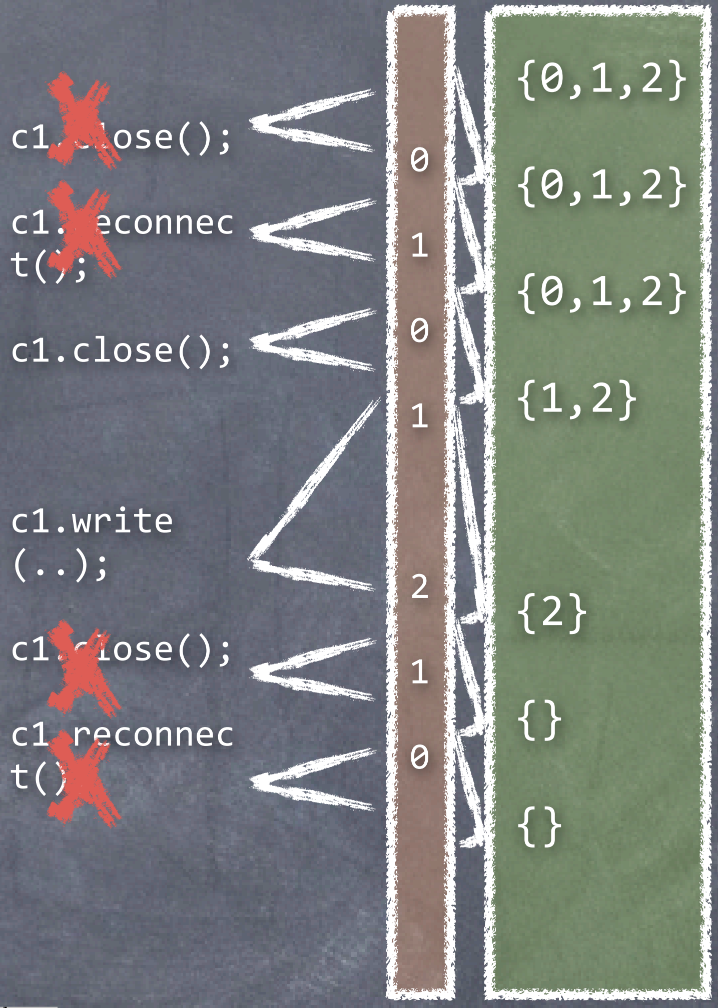
Conclusion

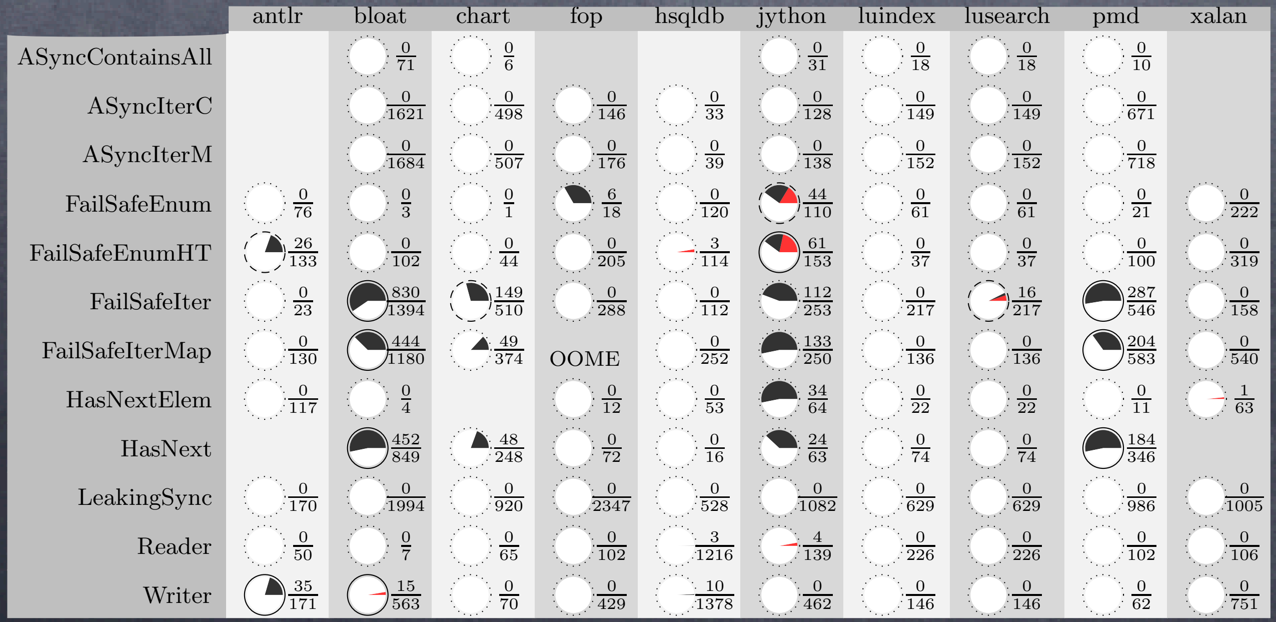
```
dependency{
    disconnect, write, reconnect;
    initial  connected: disconnect -> connected,
                  write -> connected,
                  reconnect -> connected,
                  disconnect -> disconnected;
               disconnect: disconnect -> disconnected,
                  write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



Conclusion

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

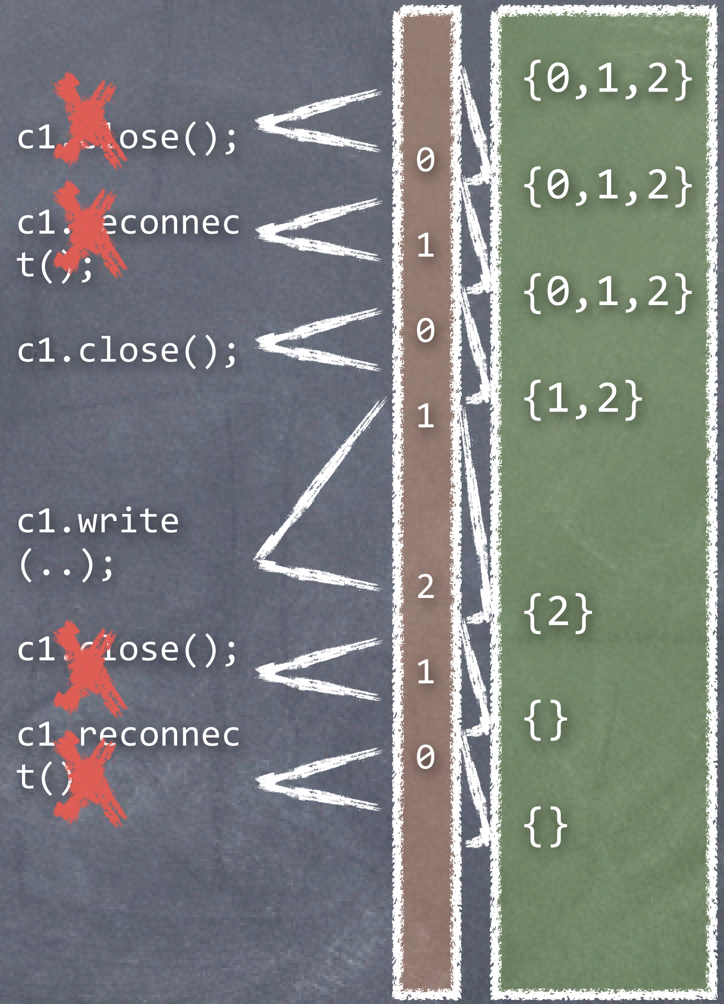




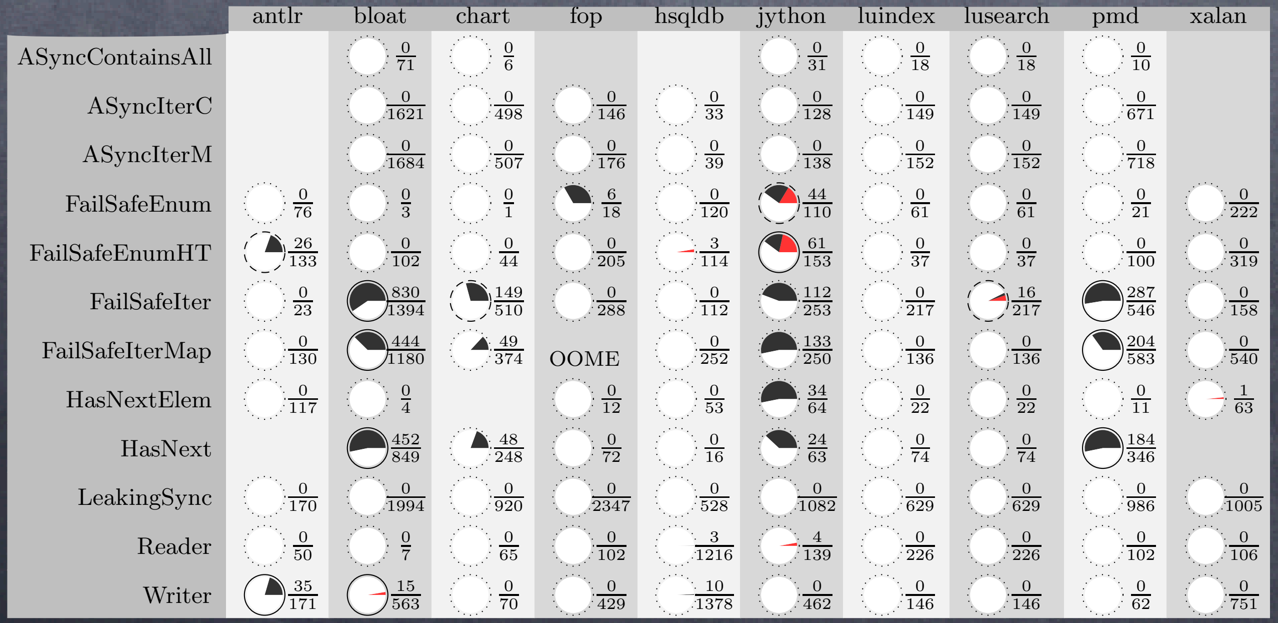
	antlr	bloat	chart	fop	hsqldb	jython	luindex	lusearch	pmd	xalan
ASyncContainsAll		0/71	0/6			0/31	0/18	0/18	0/10	
ASyncIterC		0/1621	0/498	0/146	0/33	0/128	0/149	0/149	0/671	
ASyncIterM		0/1684	0/507	0/176	0/39	0/138	0/152	0/152	0/718	
FailSafeEnum	0/76	0/3	0/1	6/18	0/120	44/110	0/61	0/61	0/21	0/222
FailSafeEnumHT	26/133	0/102	0/44	0/205	3/114	61/153	0/37	0/37	0/100	0/319
FailSafeIter	0/23	830/1394	149/510	0/288	0/112	112/253	0/217	16/217	287/546	0/158
FailSafeIterMap	0/130	444/1180	49/374	OOME	0/252	133/250	0/136	0/136	204/583	0/540
HasNextElem	0/117	0/4		0/12	0/53	34/64	0/22	0/22	0/11	1/63
HasNext		452/849	48/248	0/72	0/16	24/63	0/74	0/74	184/346	
LeakingSync	0/170	0/1994	0/920	0/2347	0/528	0/1082	0/629	0/629	0/986	0/1005
Reader	0/50	0/7	0/65	0/102	3/1216	4/139	0/226	0/226	0/102	0/106
Writer	35/171	15/563	0/70	0/429	10/1378	0/462	0/146	0/146	0/62	0/751

Conclusion

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



www.bodden.de/clara/



	antlr	bloat	chart	fop	hsqldb	jython	luindex	lusearch	pmd	xalan
ASyncContainsAll		0/71	0/6			0/31	0/18	0/18	0/10	
ASyncIterC		0/1621	0/498	0/146	0/33	0/128	0/149	0/149	0/671	
ASyncIterM		0/1684	0/507	0/176	0/39	0/138	0/152	0/152	0/718	
FailSafeEnum	0/76	0/3	0/1	6/18	0/120	44/110	0/61	0/61	0/21	0/222
FailSafeEnumHT	26/133	0/102	0/44	0/205	3/114	61/153	0/37	0/37	0/100	0/319
FailSafeIter	0/23	830/1394	149/510	0/288	0/112	112/253	0/217	16/217	287/546	0/158
FailSafeIterMap	0/130	444/1180	49/374	OOME	0/252	133/250	0/136	0/136	204/583	0/540
HasNextElem	0/117	0/4		0/12	0/53	34/64	0/22	0/22	0/11	1/63
HasNext		452/849	48/248	0/72	0/16	24/63	0/74	0/74	184/346	
LeakingSync	0/170	0/1994	0/920	0/2347	0/528	0/1082	0/629	0/629	0/986	0/1005
Reader	0/50	0/7	0/65	0/102	3/1216	4/139	0/226	0/226	0/102	0/106
Writer	35/171	15/563	0/70	0/429	10/1378	0/462	0/146	0/146	0/62	0/751