The Clara framework for partially evaluating runtime monitors ahead of time

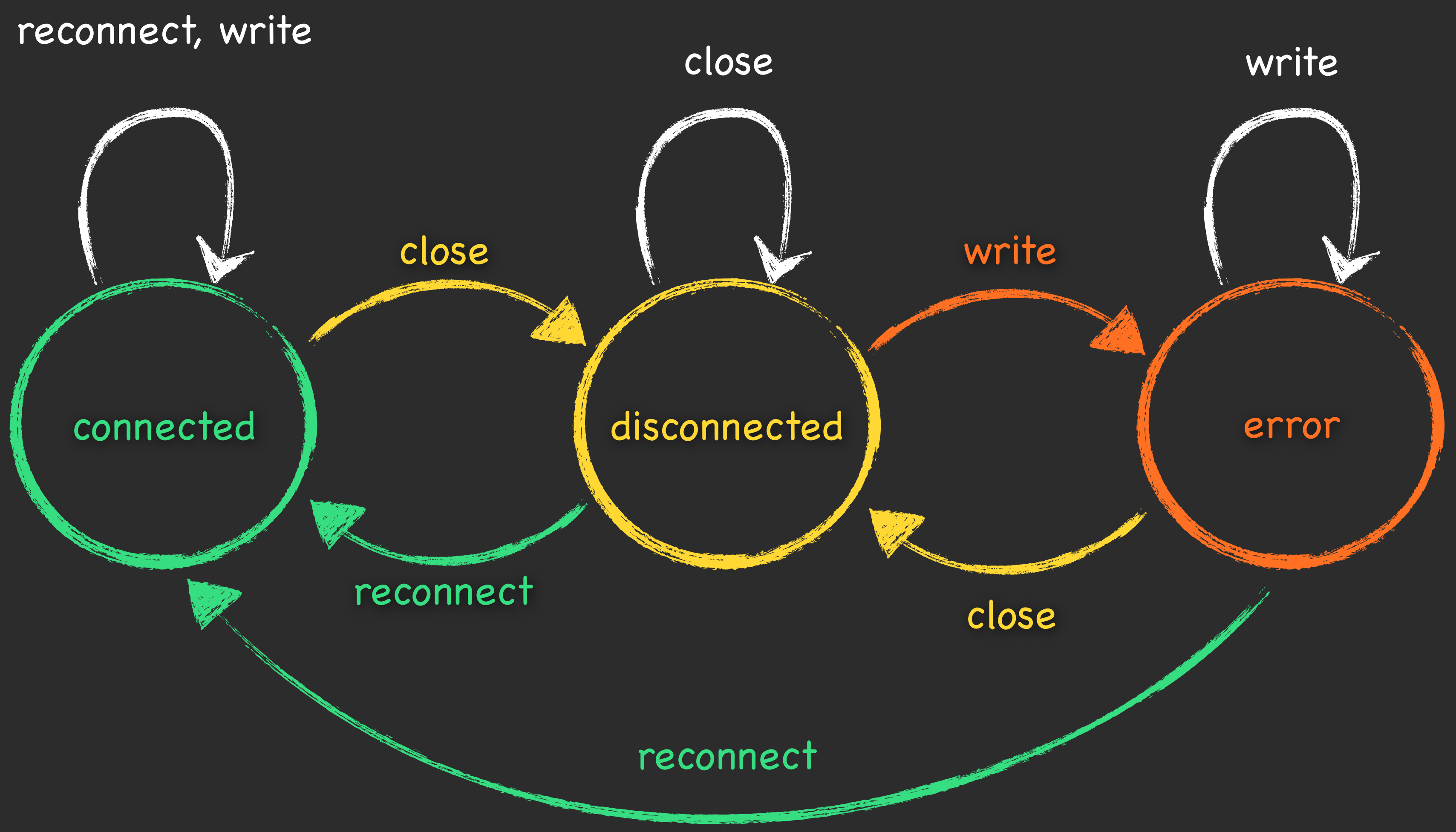
Eric Bodden

with Patrick Lam, Laurie Hendren

CASED

TECHNISCHE UNIVERSITÄT DARMSTADT

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



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

Runtime Monitoring with AspectJ

```
Set closed = new HashSet();

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!");
}
```

Runtime Monitoring with AspectJ

```
Set closed = new HashSet();

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!");
}
```

Runtime Monitoring with AspectJ

```
Set closed = new HashSet();

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!");
}
```

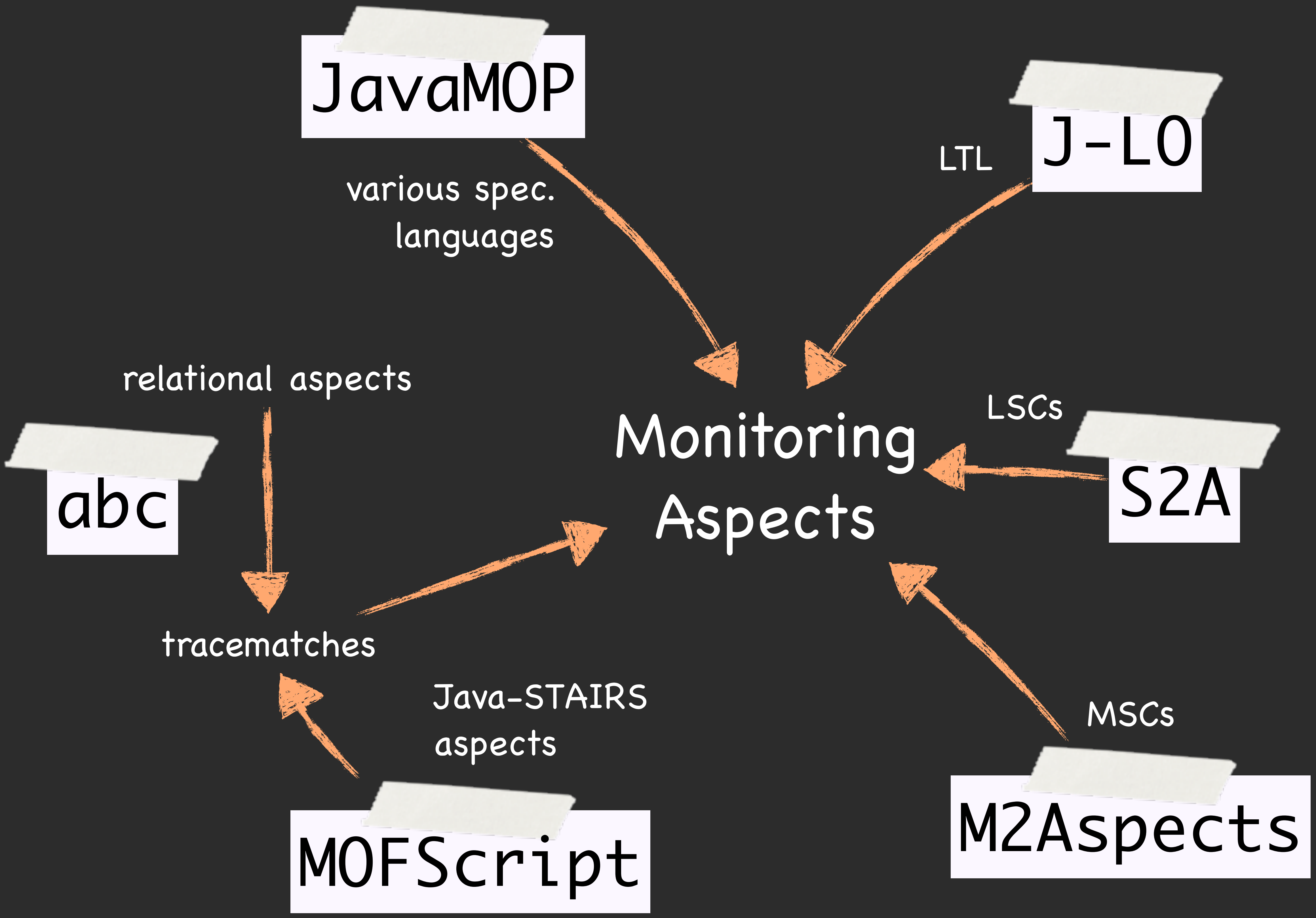
Runtime Monitoring with AspectJ

```
Set closed = new HashSet();

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!");
}
```

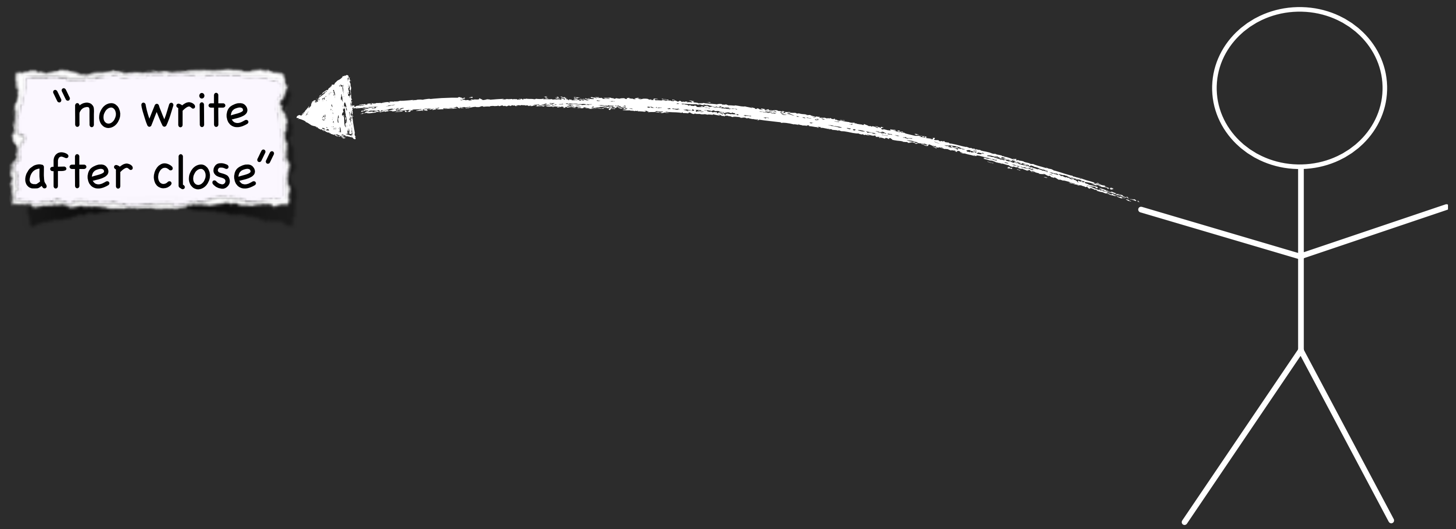
Runtime Monitoring with AspectJ

```
Set closed = new HashSet();

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!");
}
```

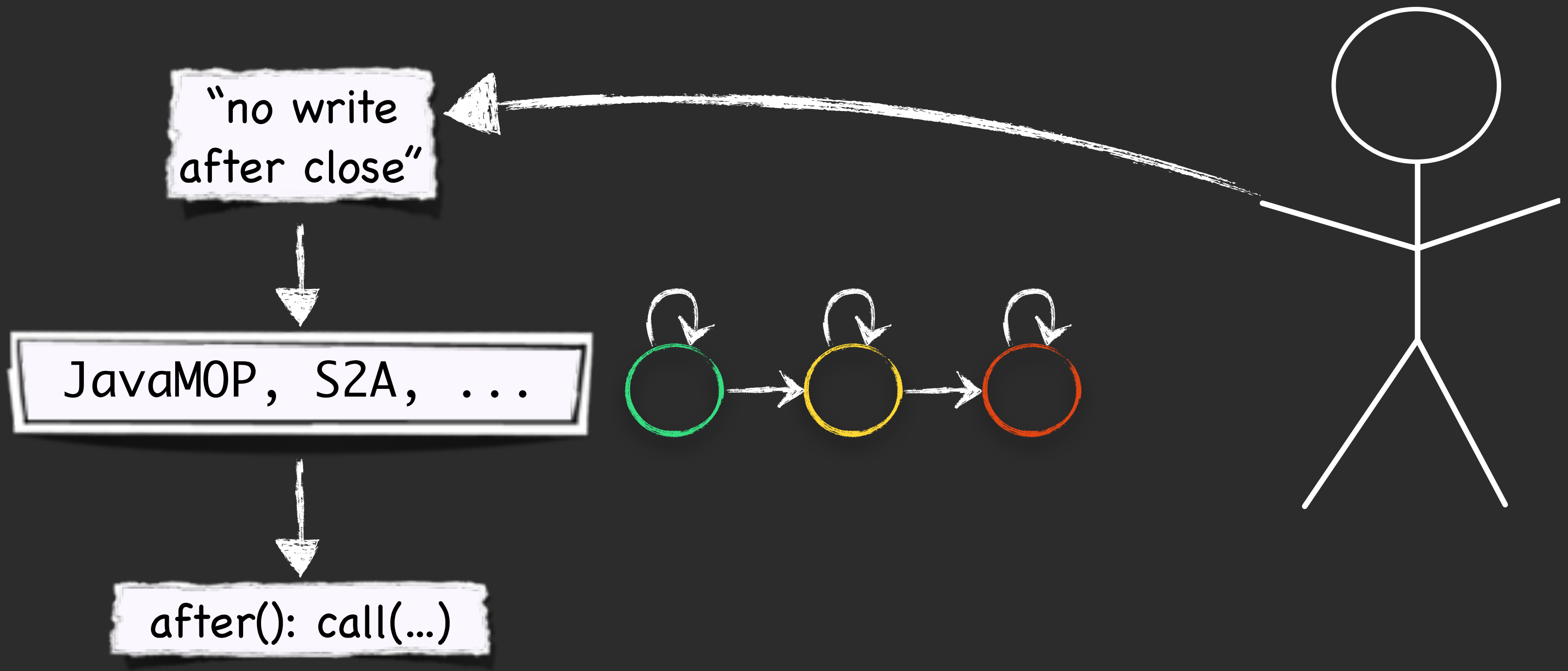
Existing Runtime Monitoring Tools



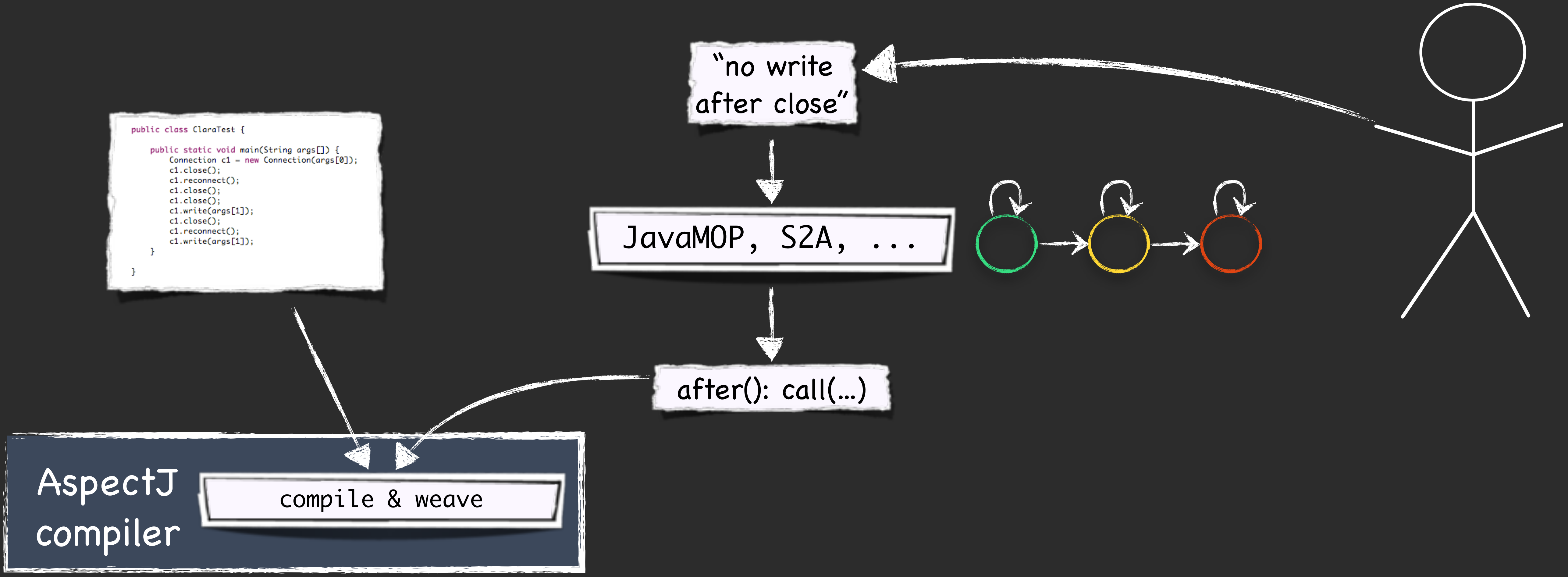
Runtime-verifying finite-state properties



Runtime-verifying finite-state properties

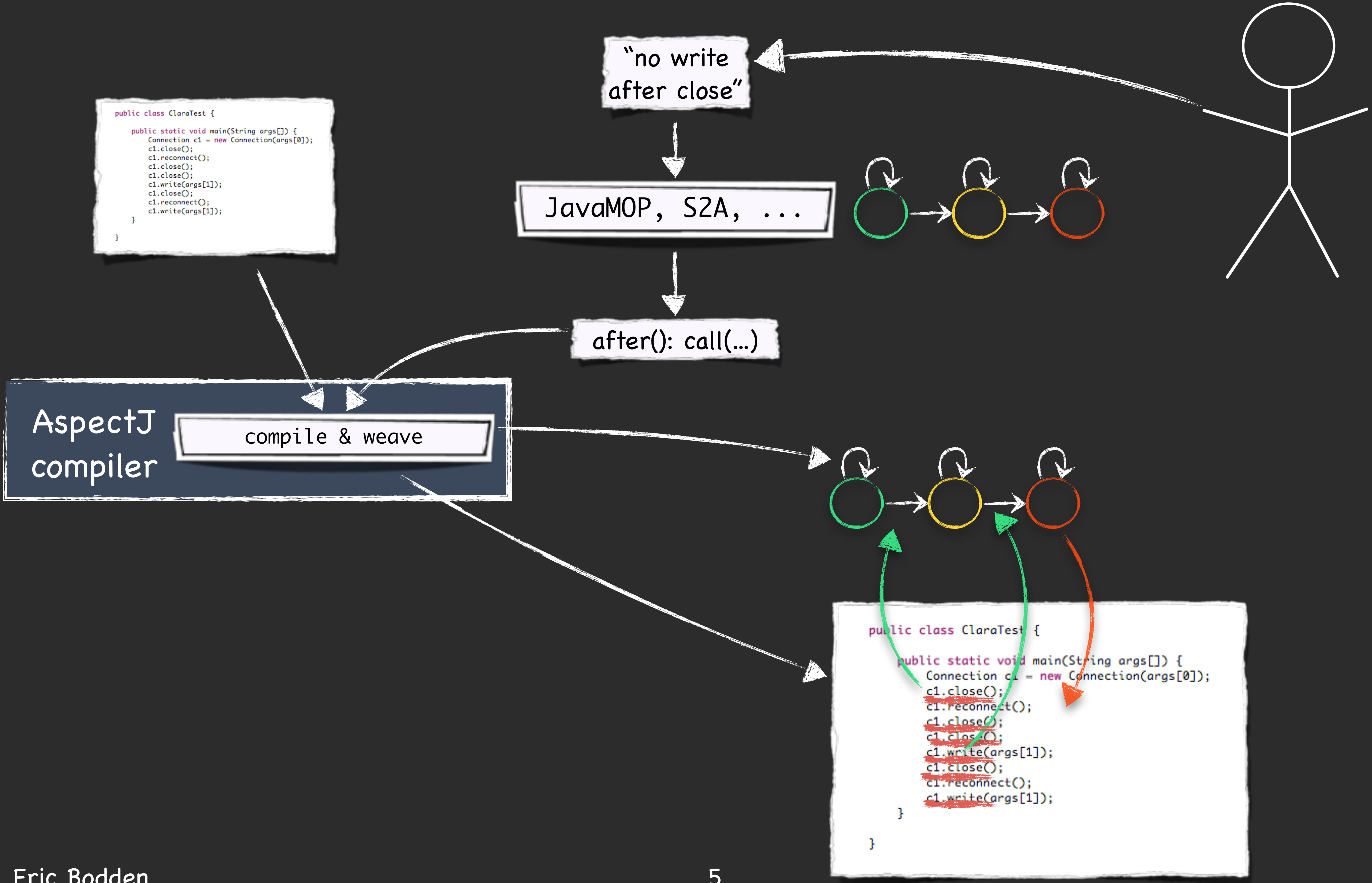


Runtime-verifying finite-state properties



```
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]);
    }

}
```

```
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

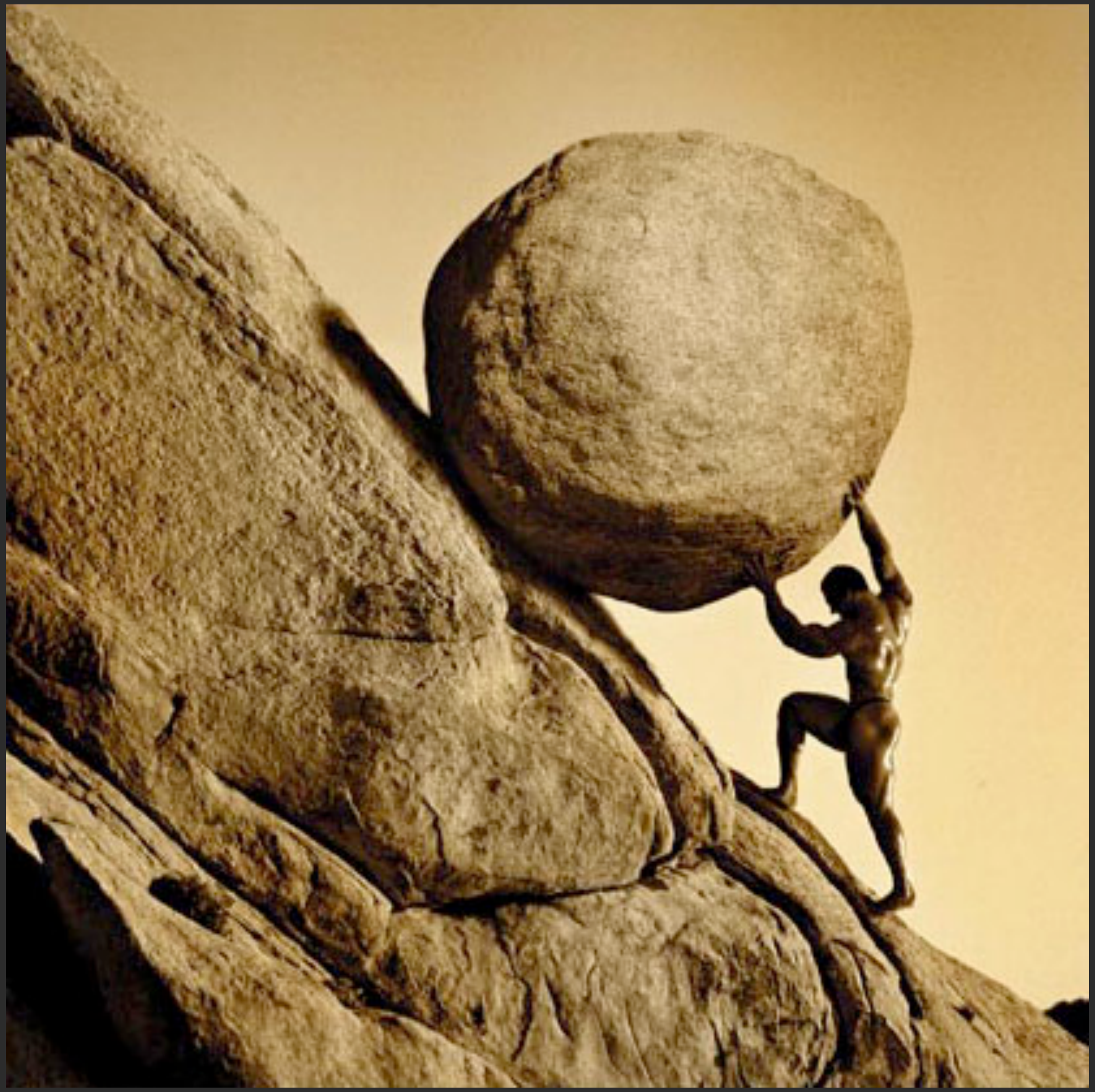
Runtime-verifying finite-state properties

No static guarantees

Runtime-verifying finite-state properties

Potentially large runtime overhead

Runtime-verifying finite-state properties



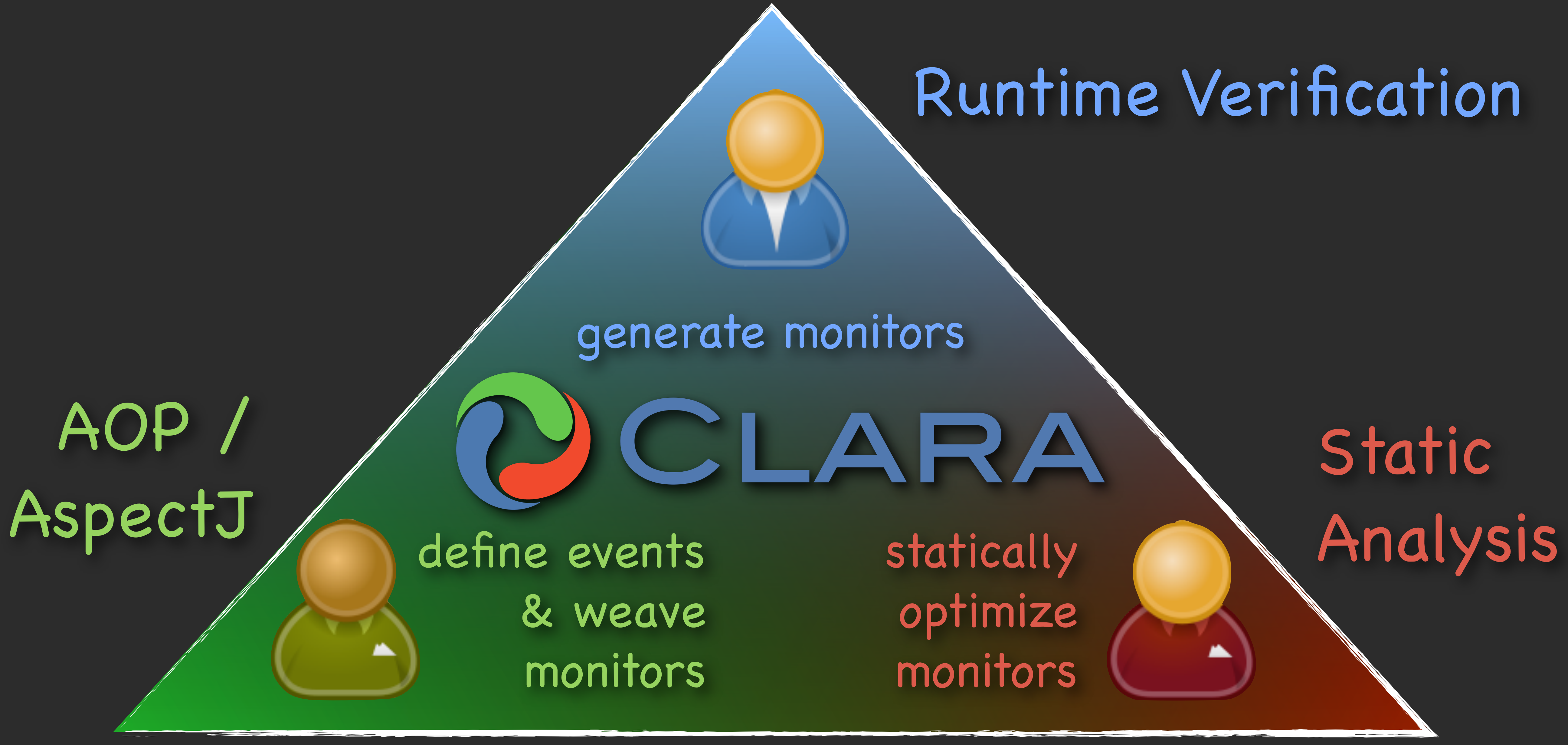
When to finish testing?

Integrate results of three communities

CLARA

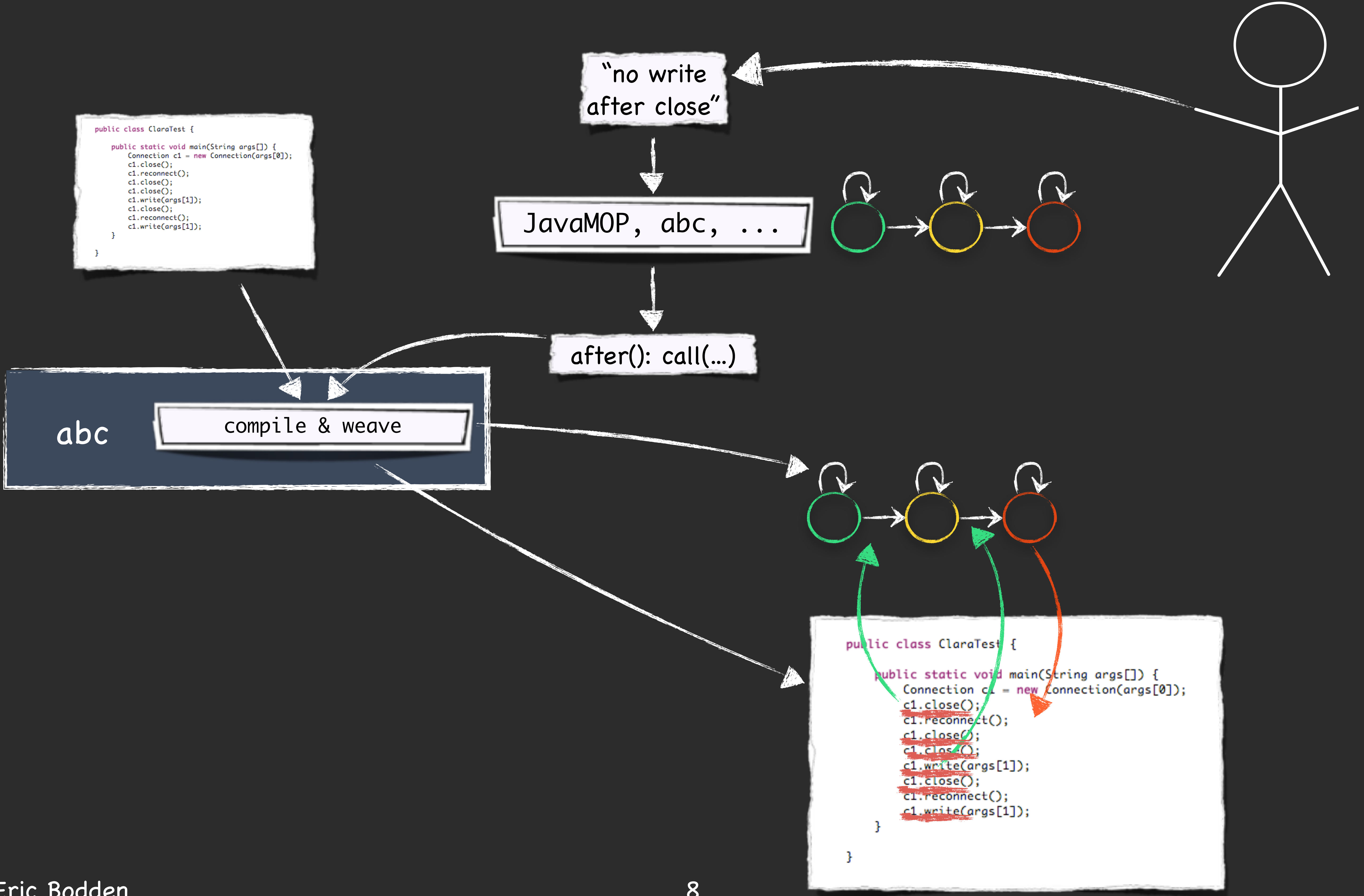
[RV2010] http://bodden.de/clara/

Integrate results of three communities



[RV2010] http://bodden.de/clara/

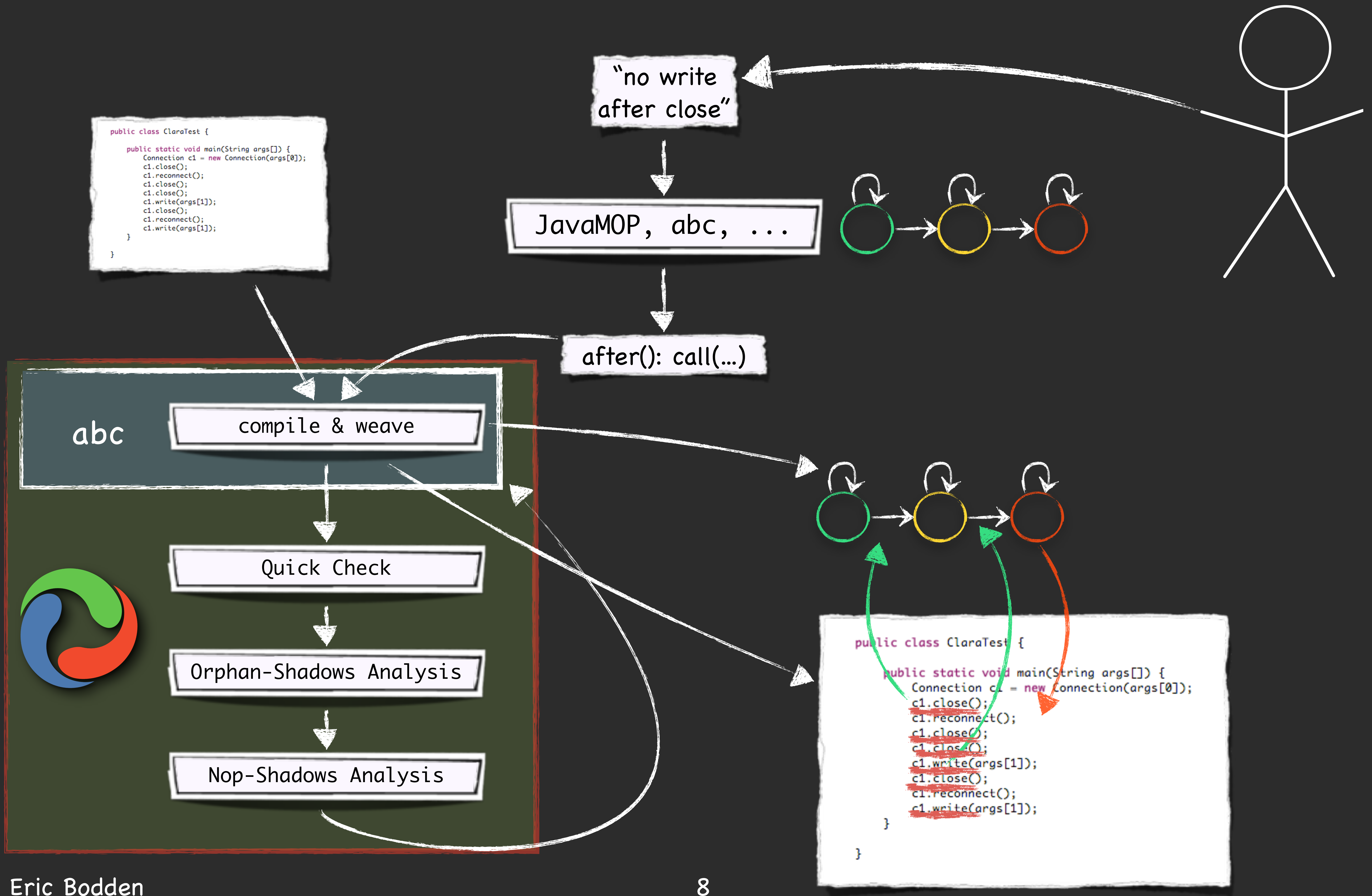
CLARA



```
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]);
    }

}
```

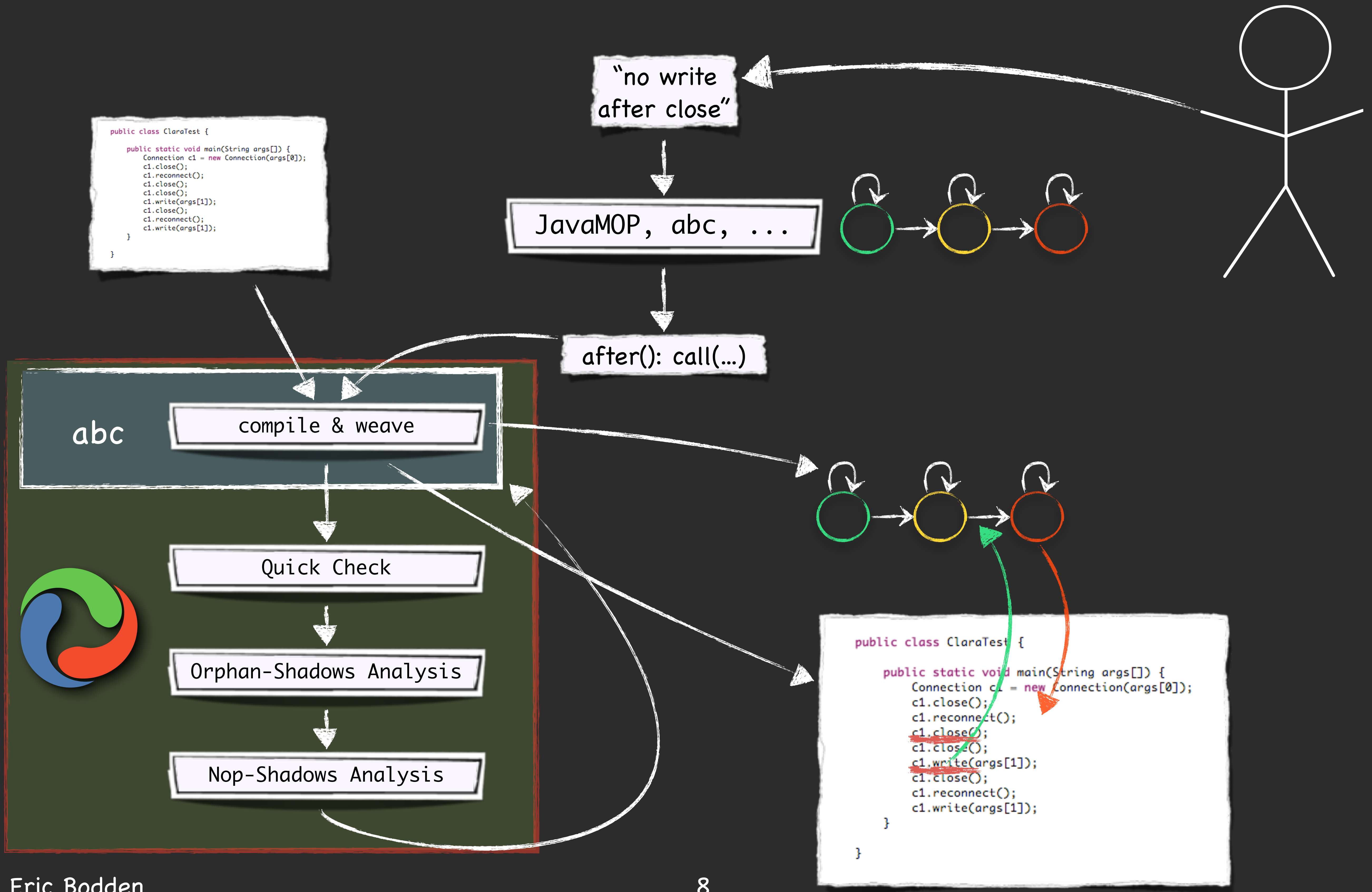
CLARA



```
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]);
    }

}
```

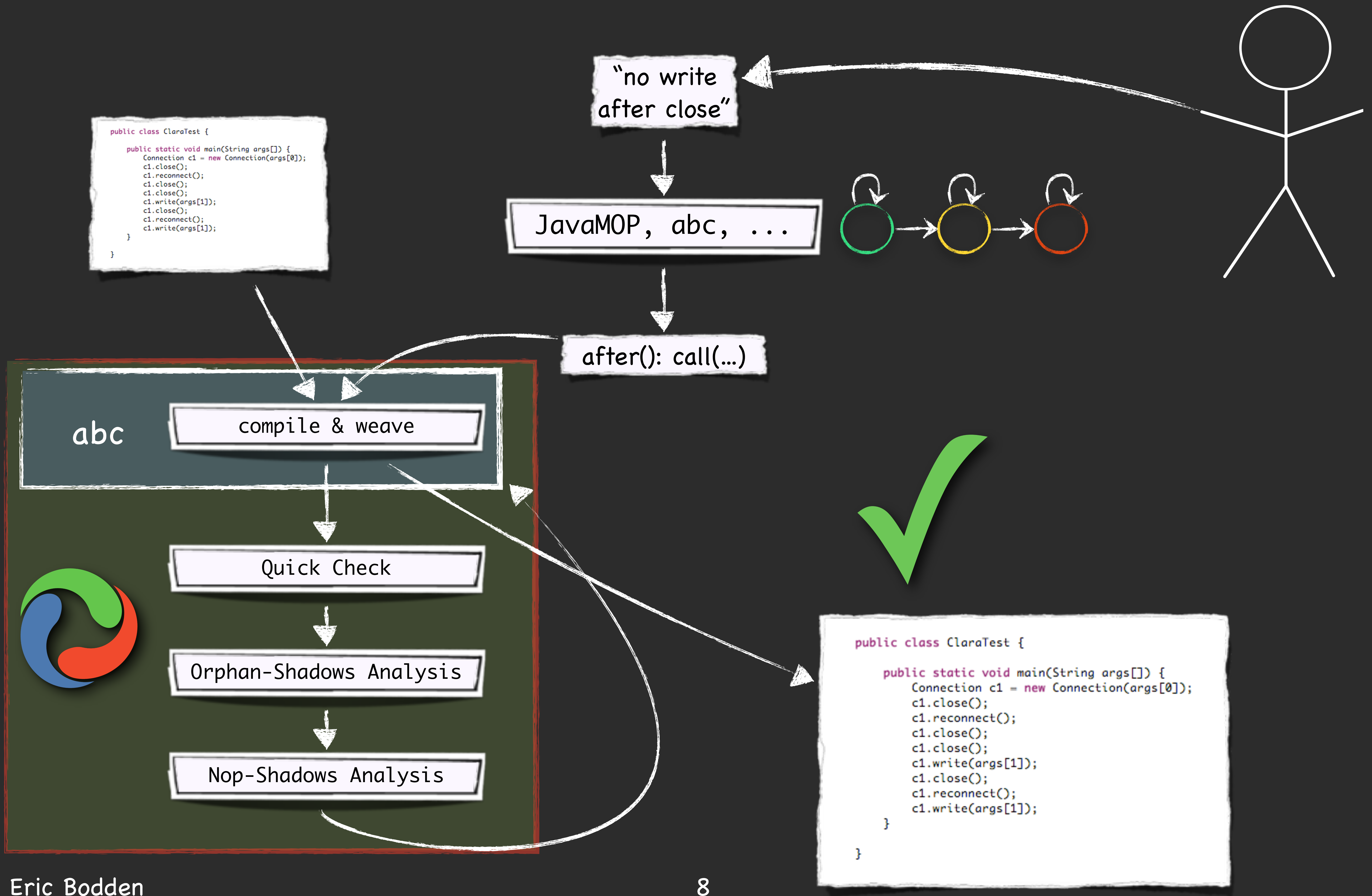
CLARA



```
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]);
    }

}
```

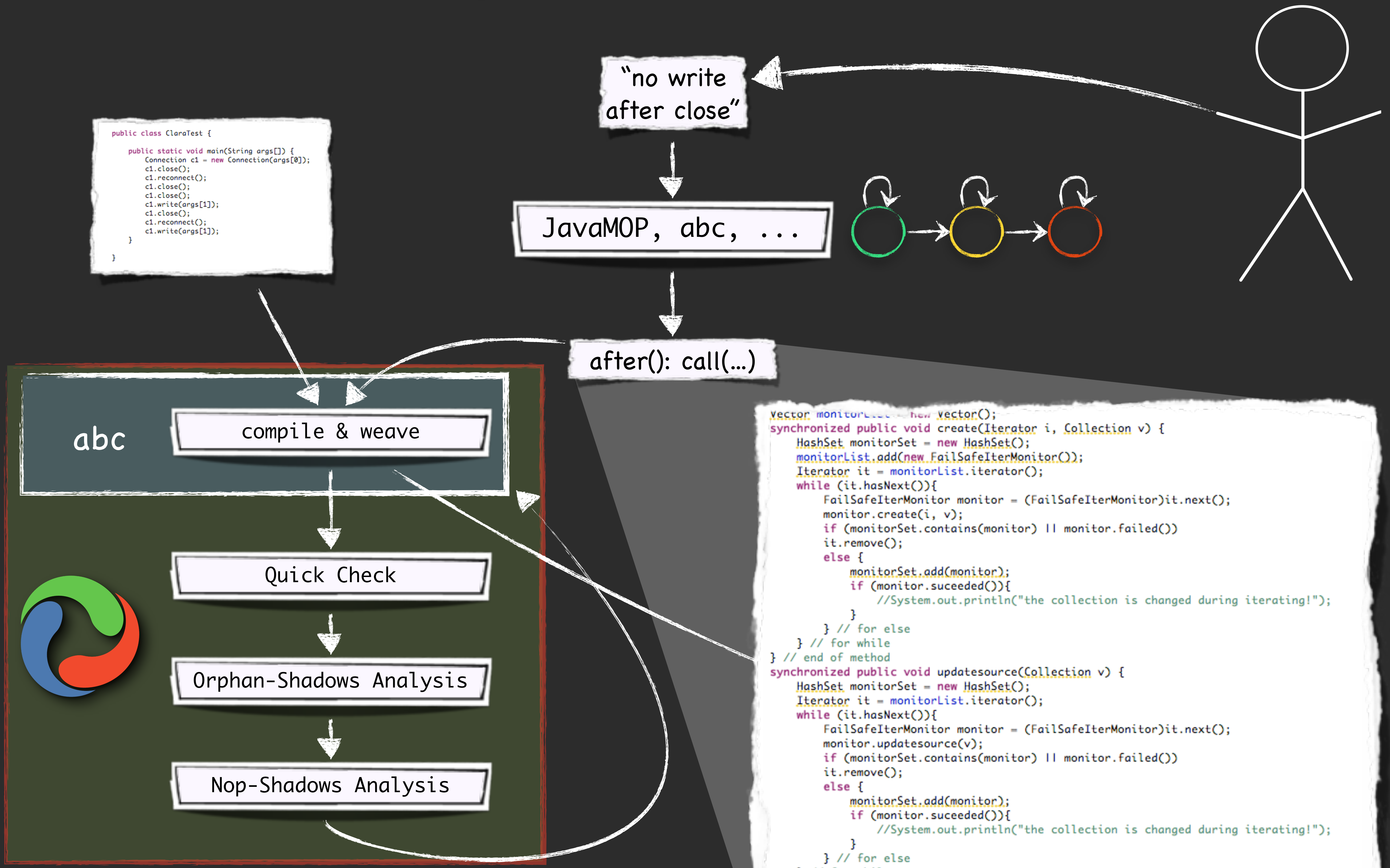
CLARA



```
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]);
    }
}
```

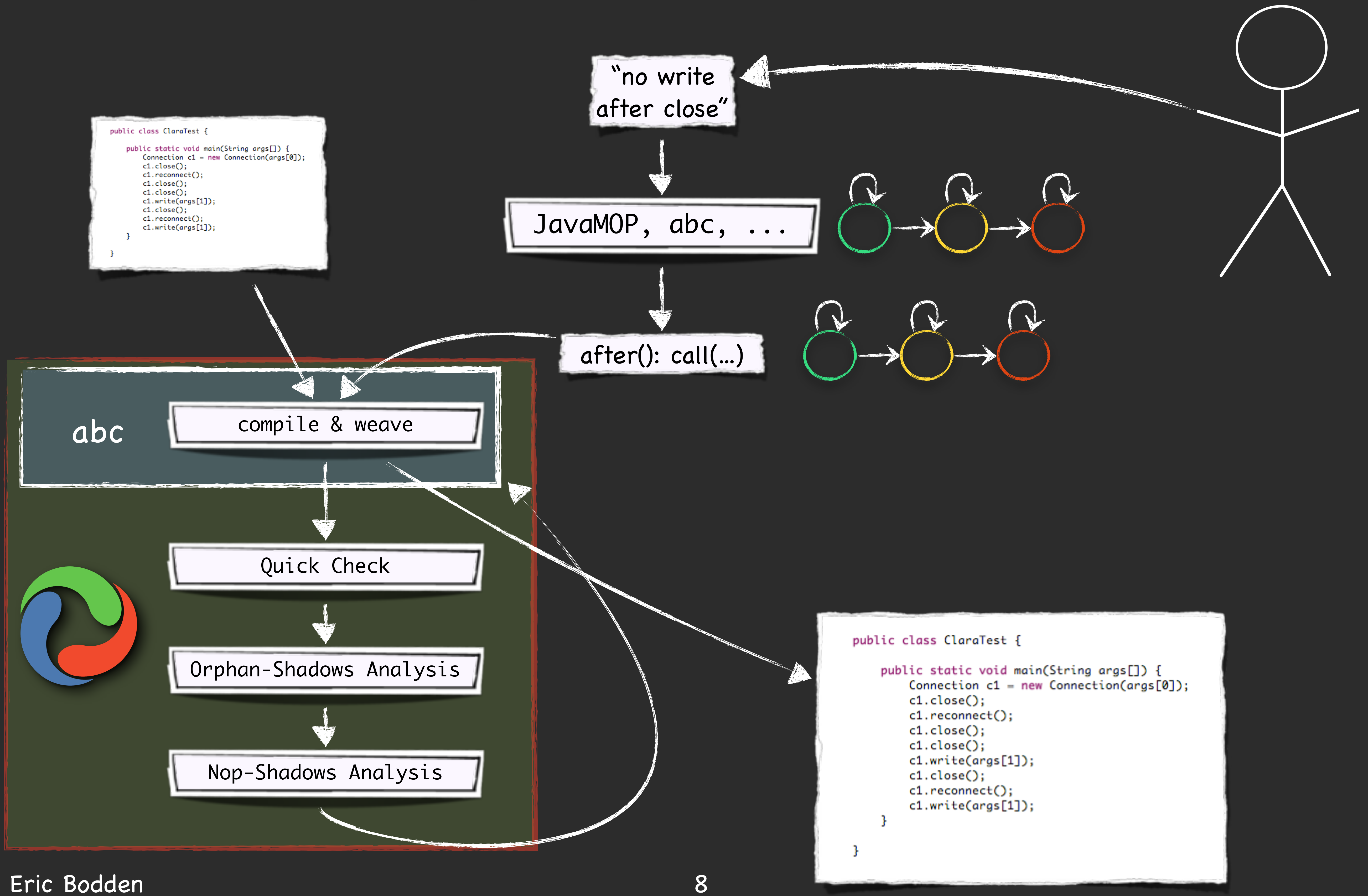
CLARA



```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
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();
```

CLARA



```
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]);
    }

}
```

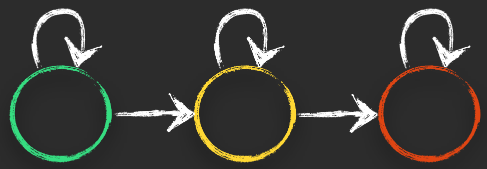
```
Set closed = new HashSet();

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

finite-state property

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



Annotation language comes with formal semantics

Interface definition through annotated AspectJ aspects

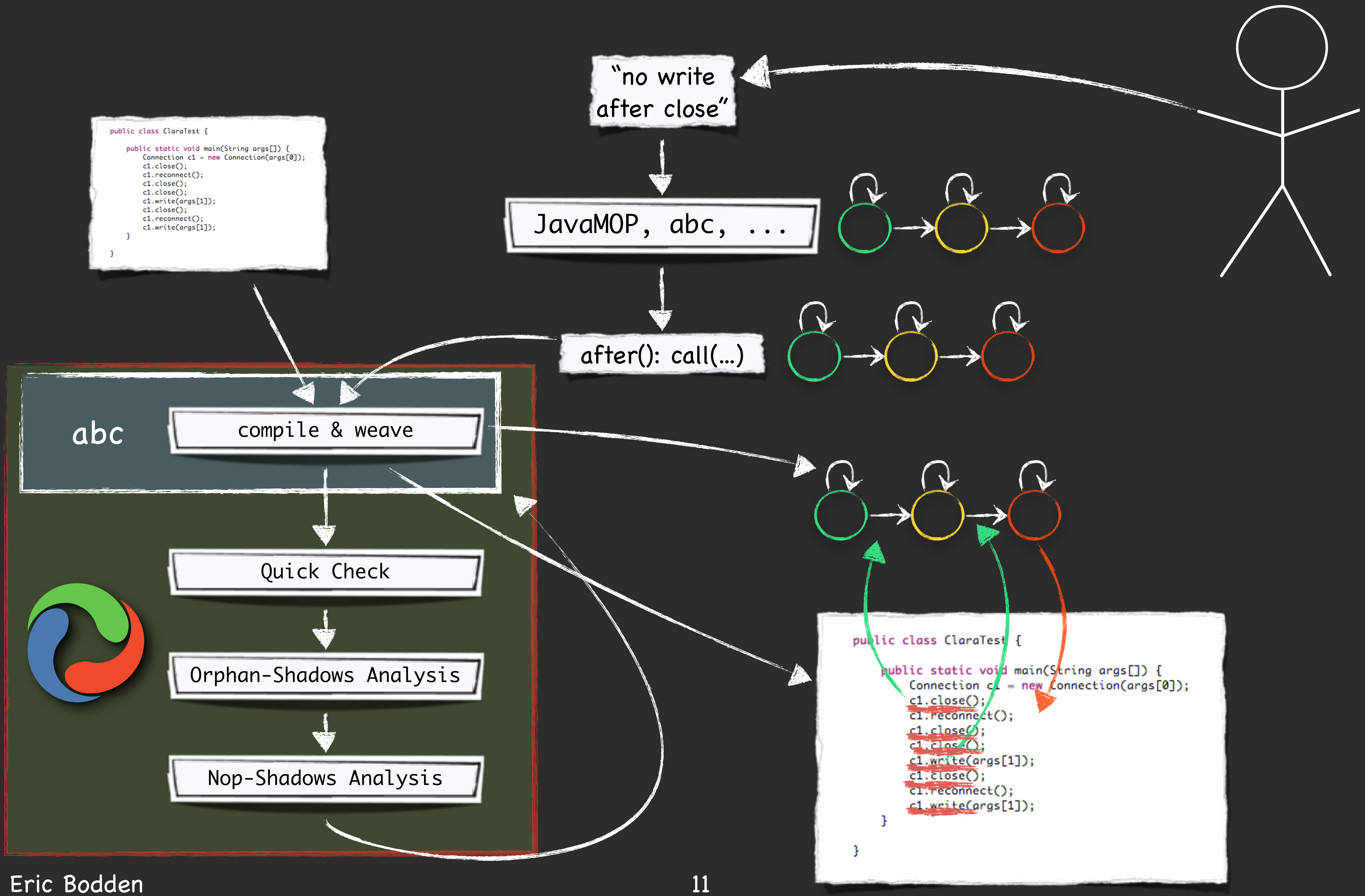
Annotation language comes with formal semantics

Are the annotations
I generated correct?

Interface definition through
annotated AspectJ aspects

Is my partial ahead-of-time
evaluation correct?
(no false warnings, no missed violations)

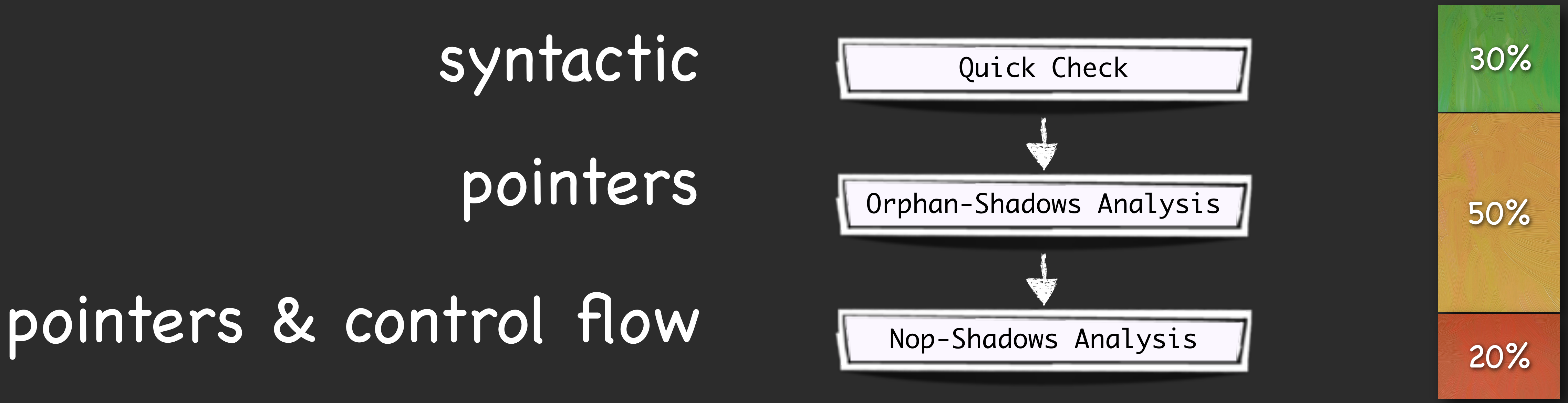
CLARA



```
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]);
    }

}
```

relative
effectiveness

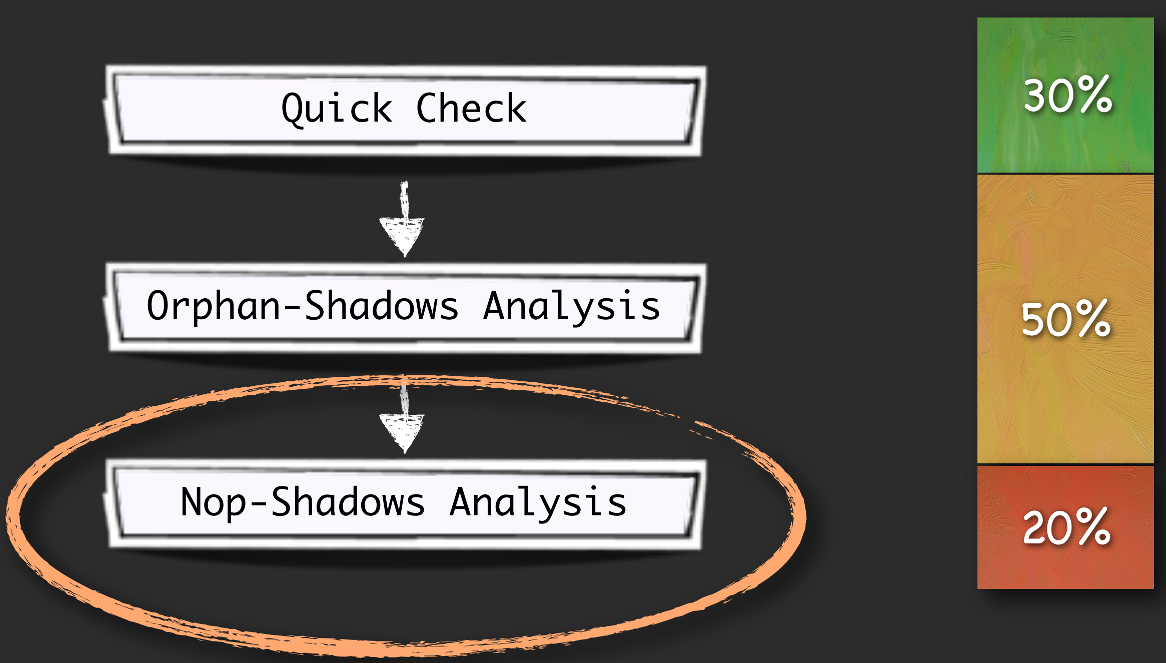


relative effectiveness

syntactic

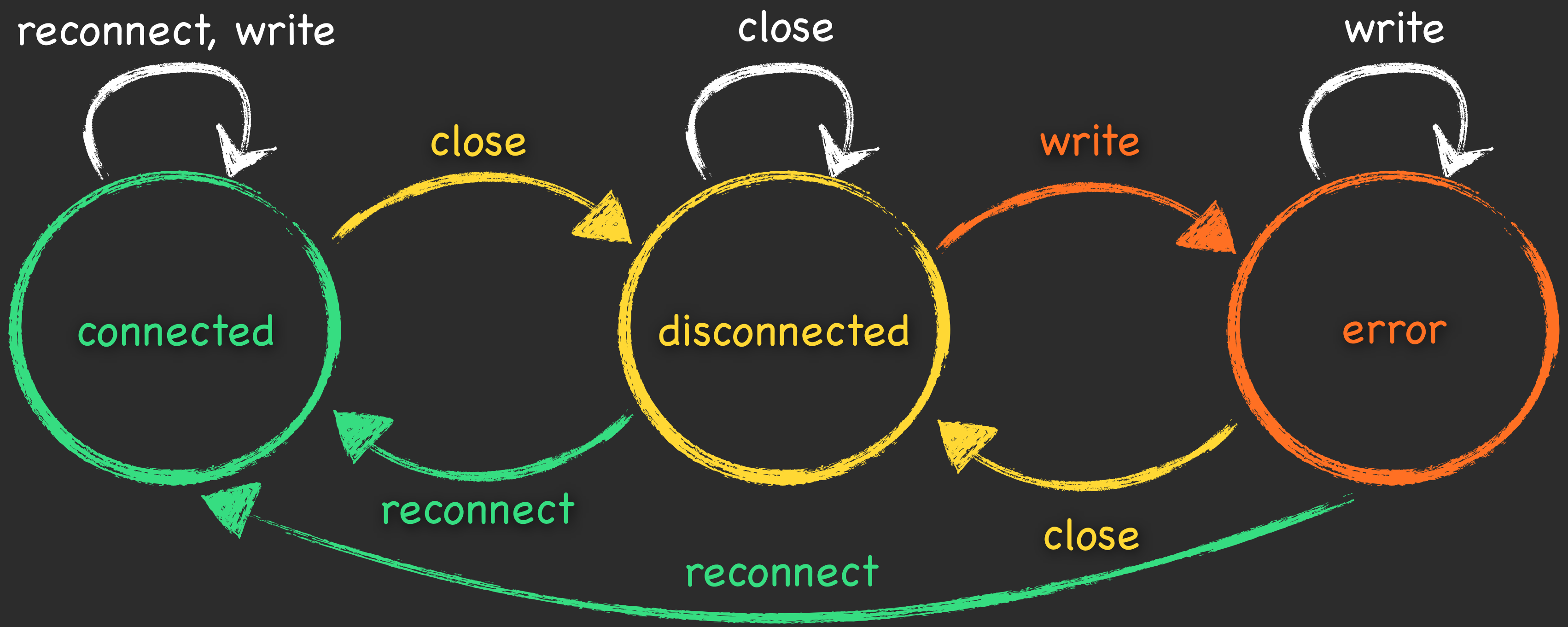
pointers

pointers & control flow



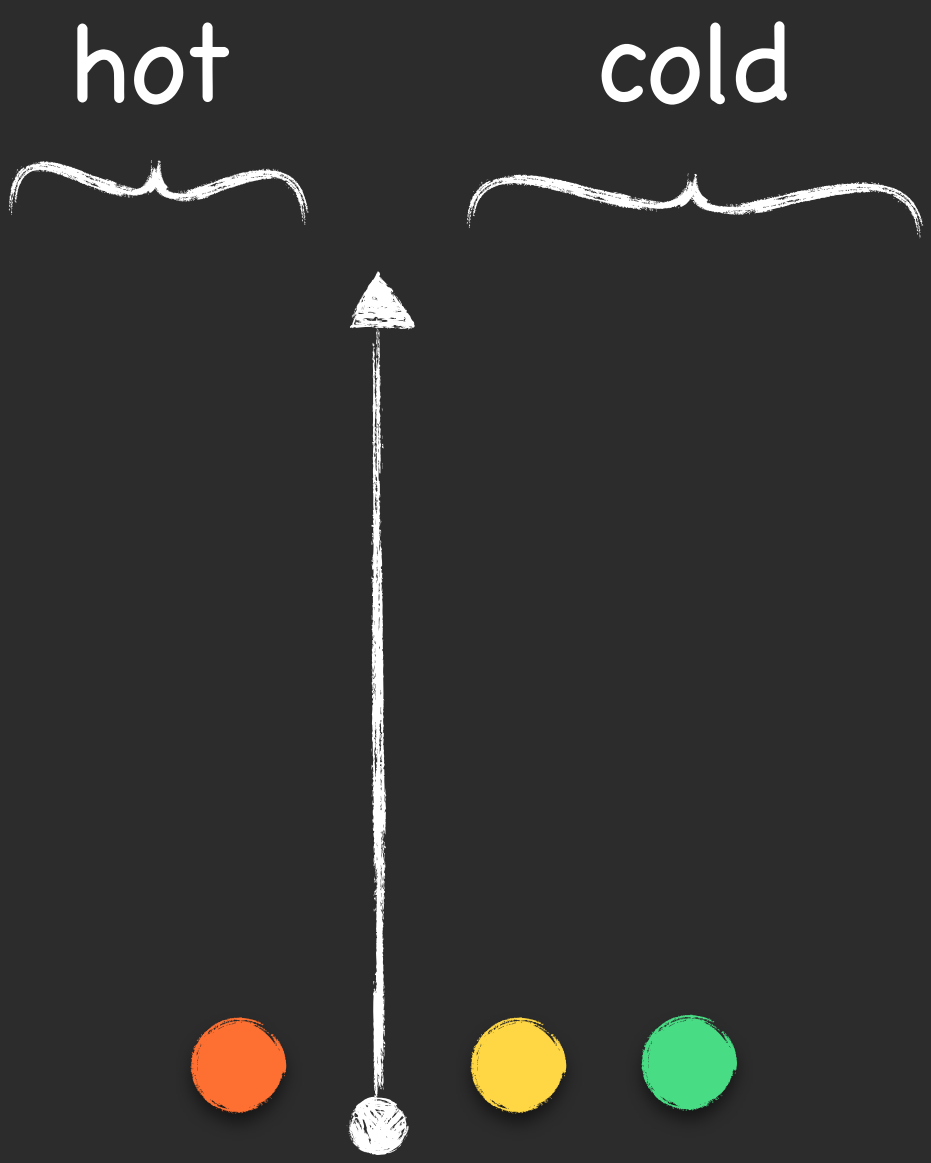
1st Iteration [ICSE2010]

```
c.close()

c.reconnect()

c.write()
```

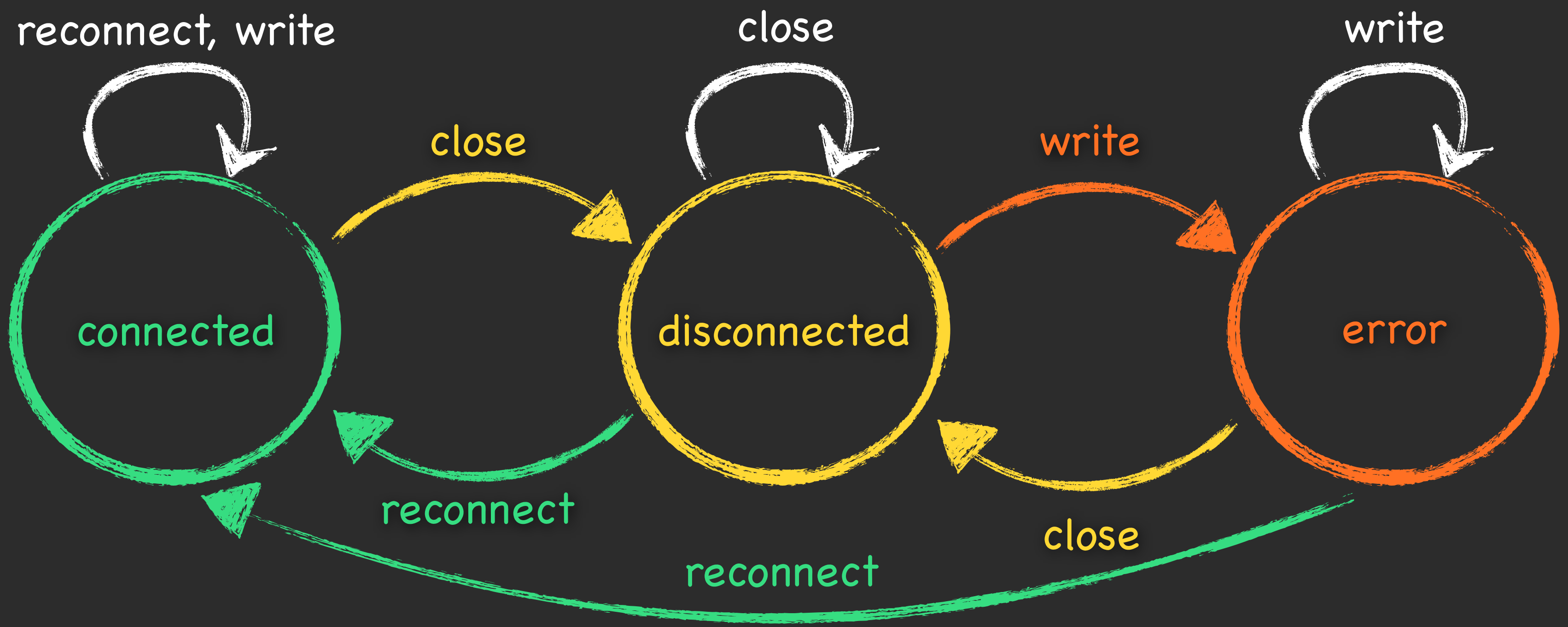


1st Iteration

[ICSE2010]

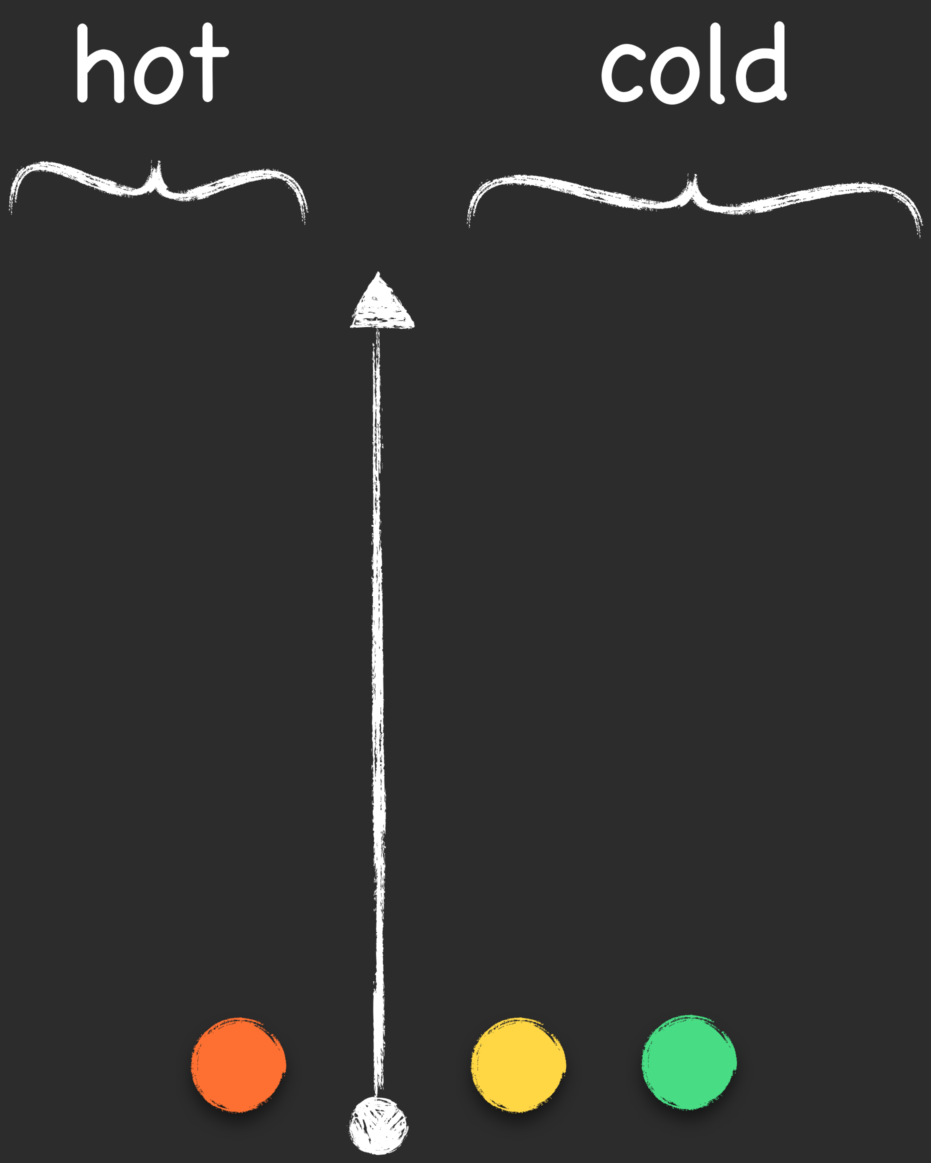


```
c.close()

c.reconnect()

c.write()
```

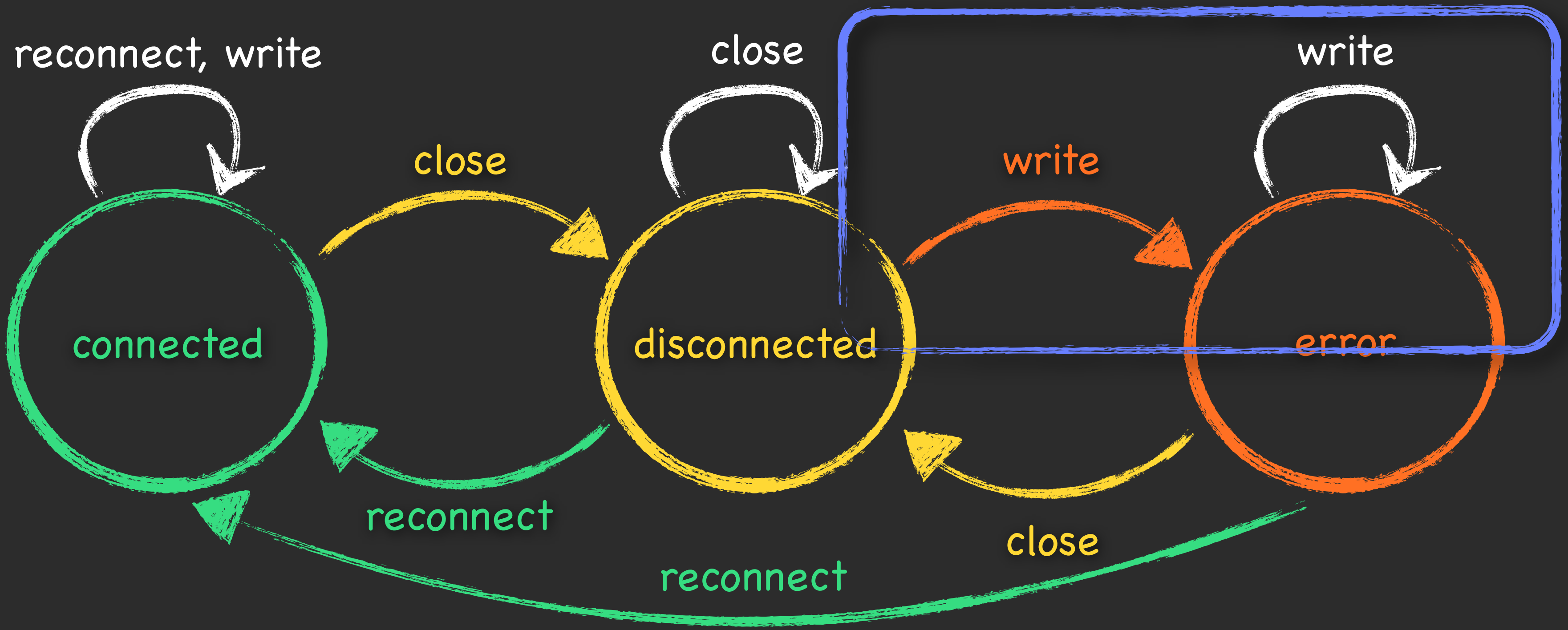


1st Iteration

[ICSE2010]

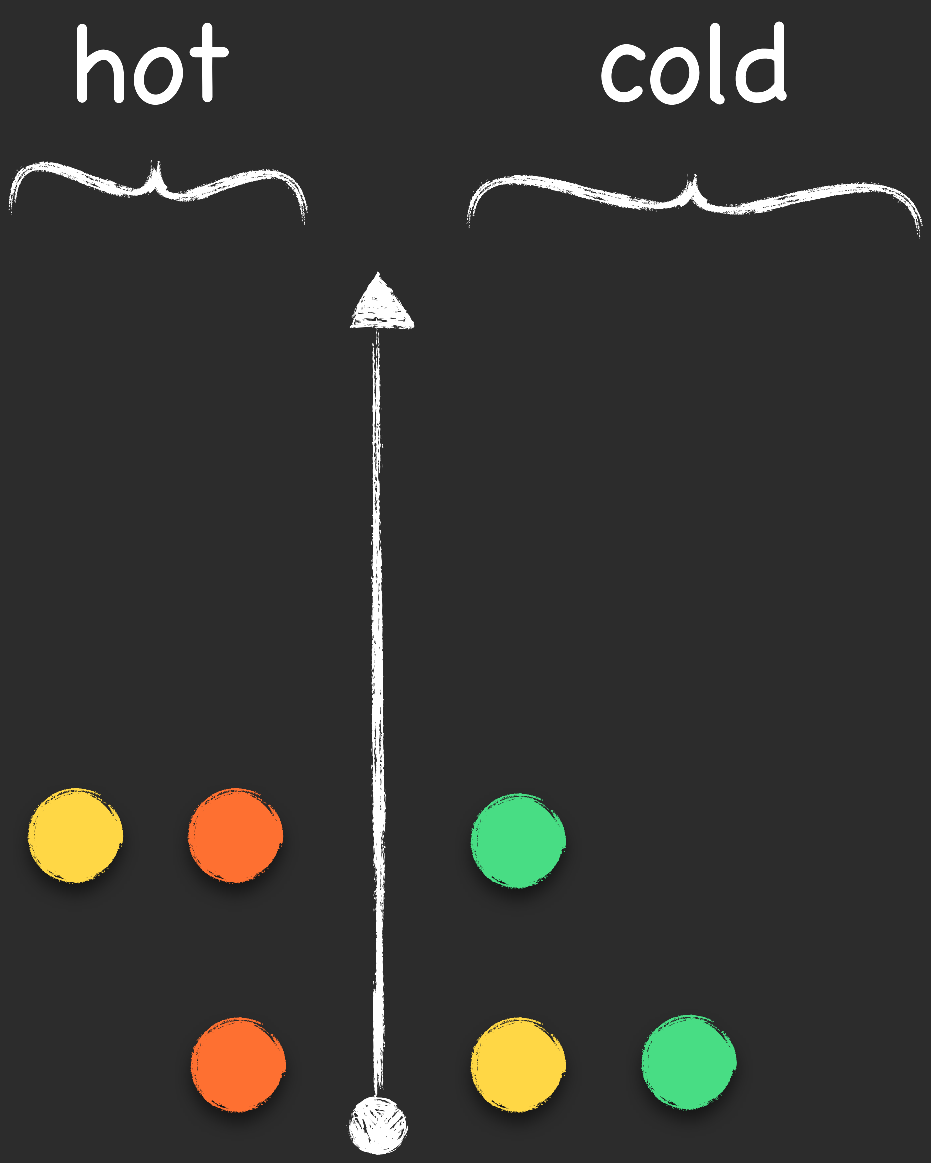


```
c.close()

c.reconnect()

c.write()
```



1st Iteration

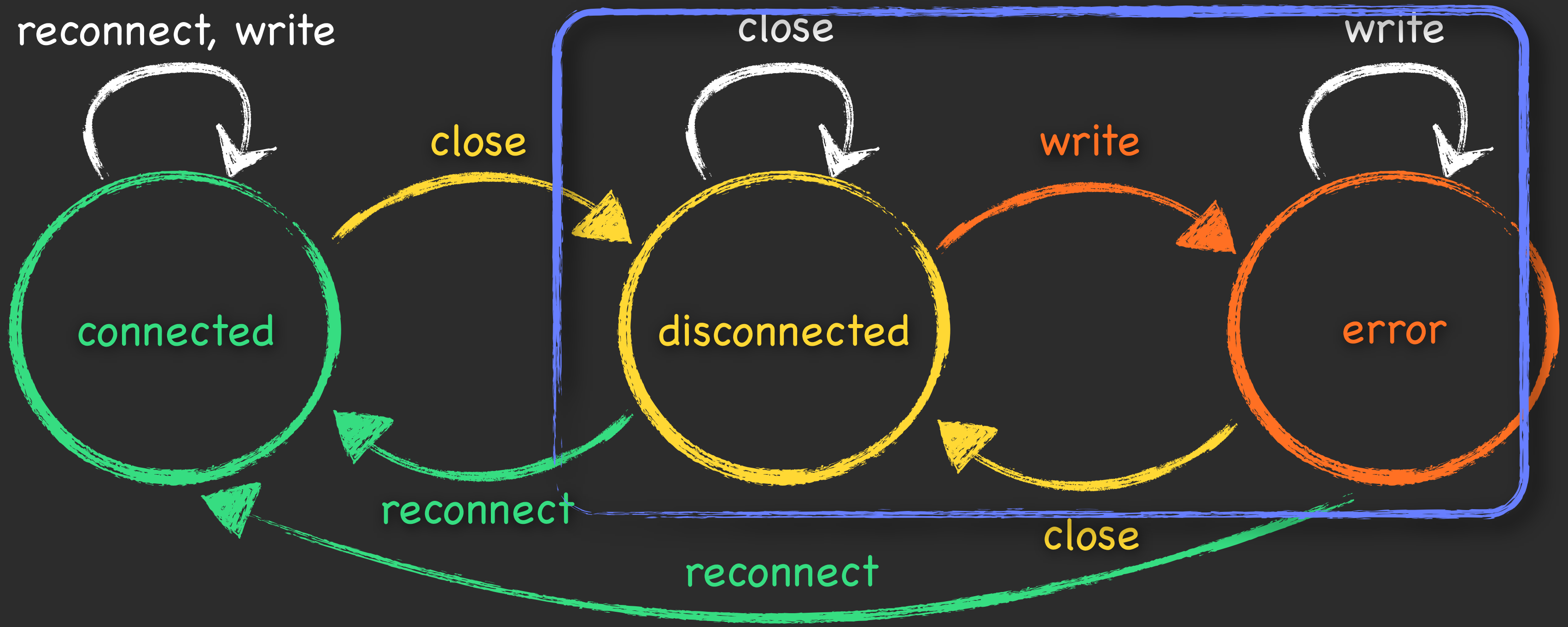
[ICSE2010]



c.close()

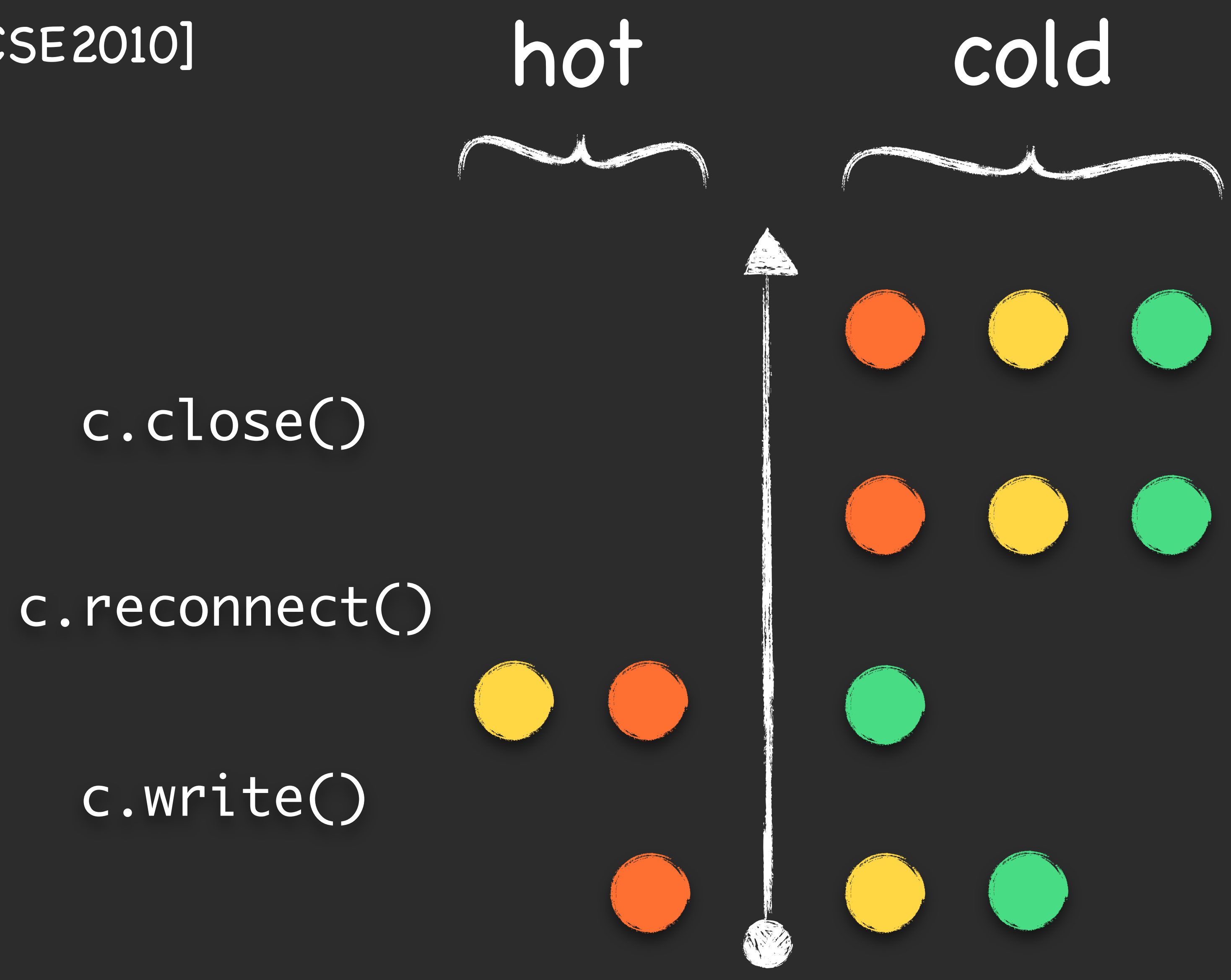
c.reconnect() ● ● ●

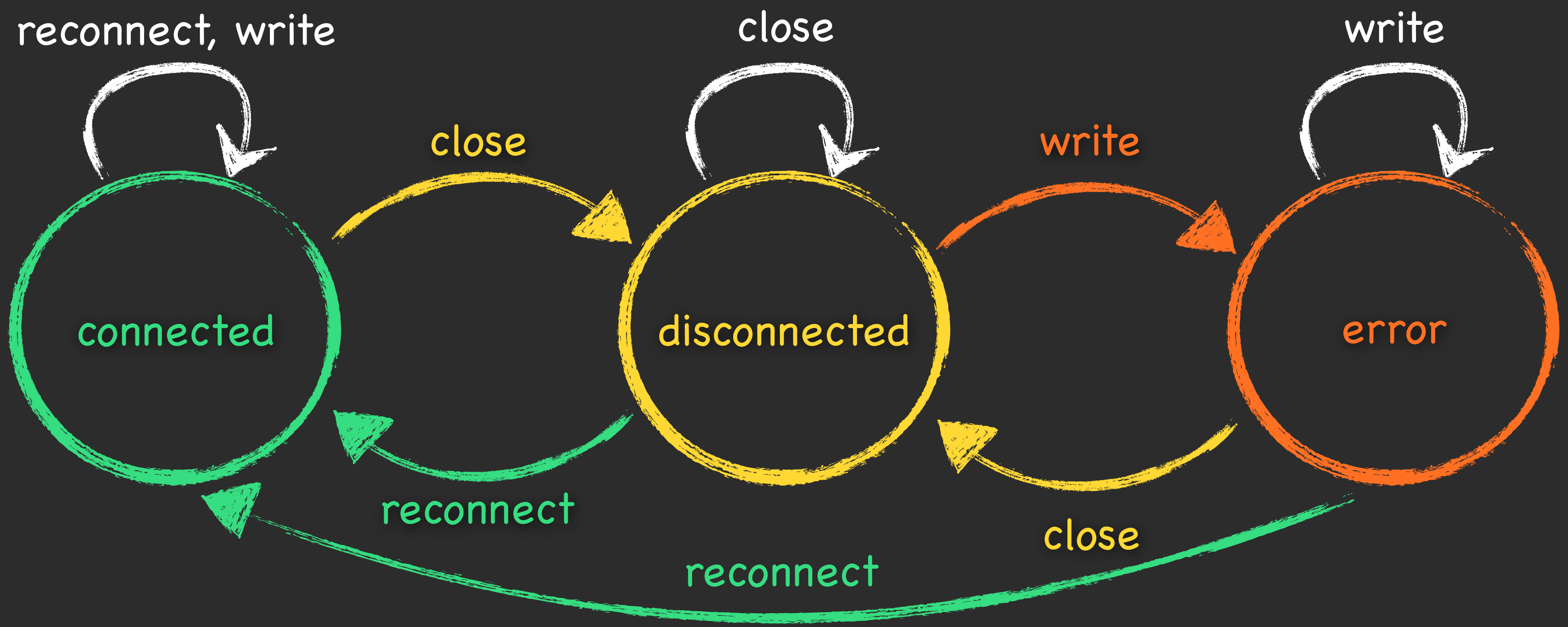
c.write() ● ● ●



1st Iteration

[ICSE2010]

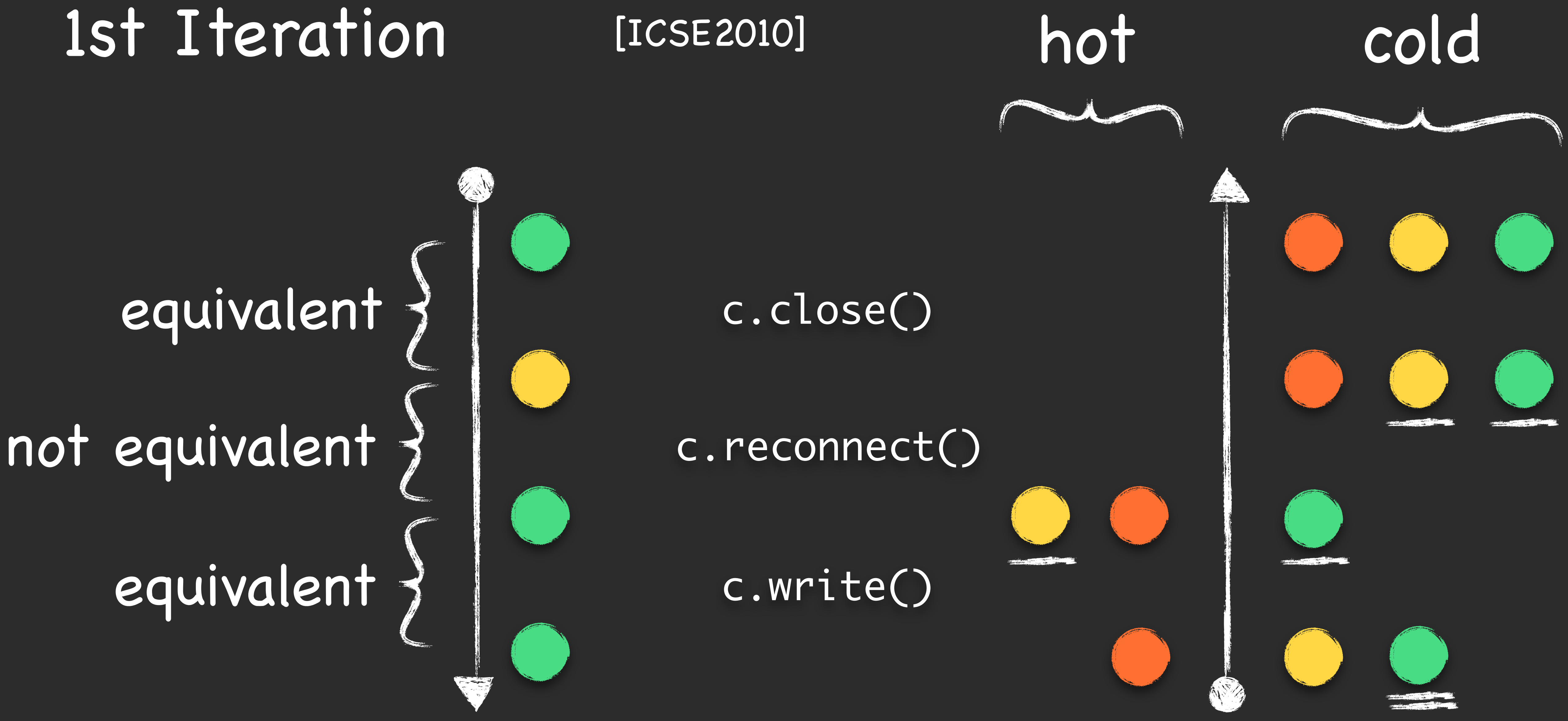


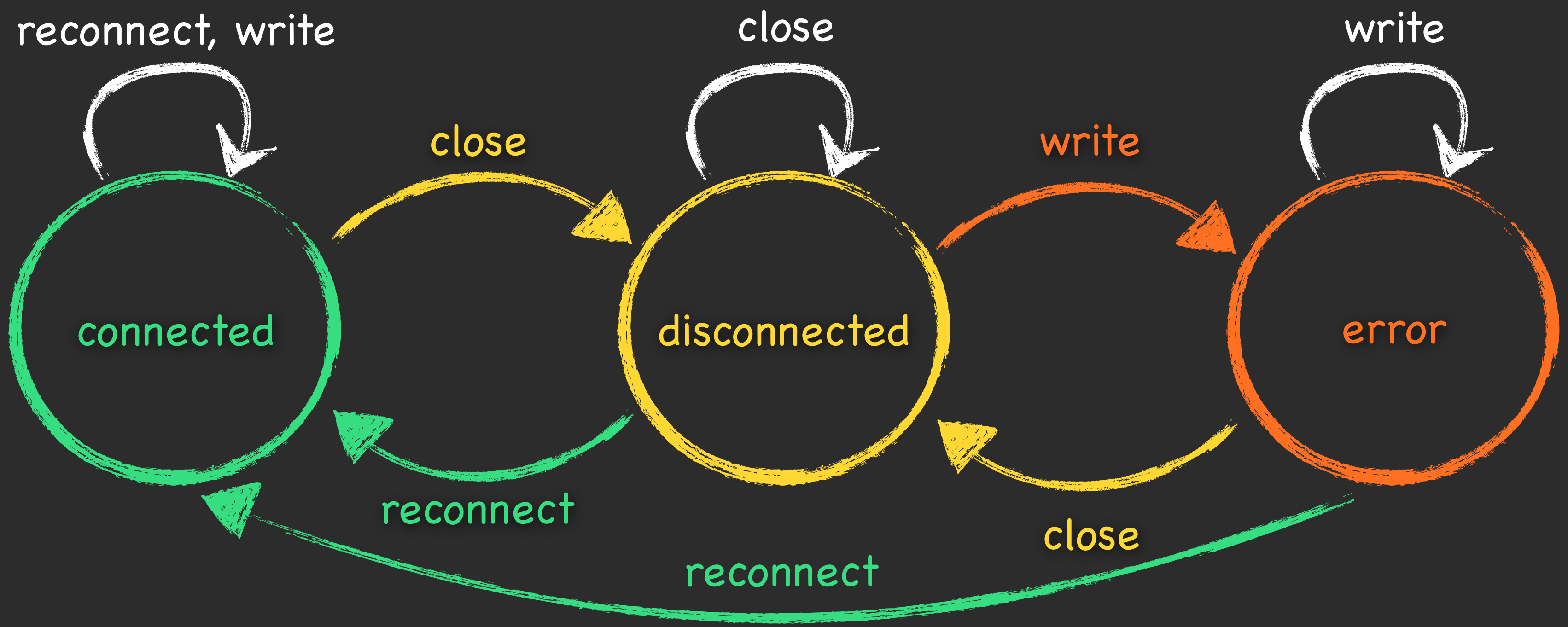


Eric Bodden

1st Iteration

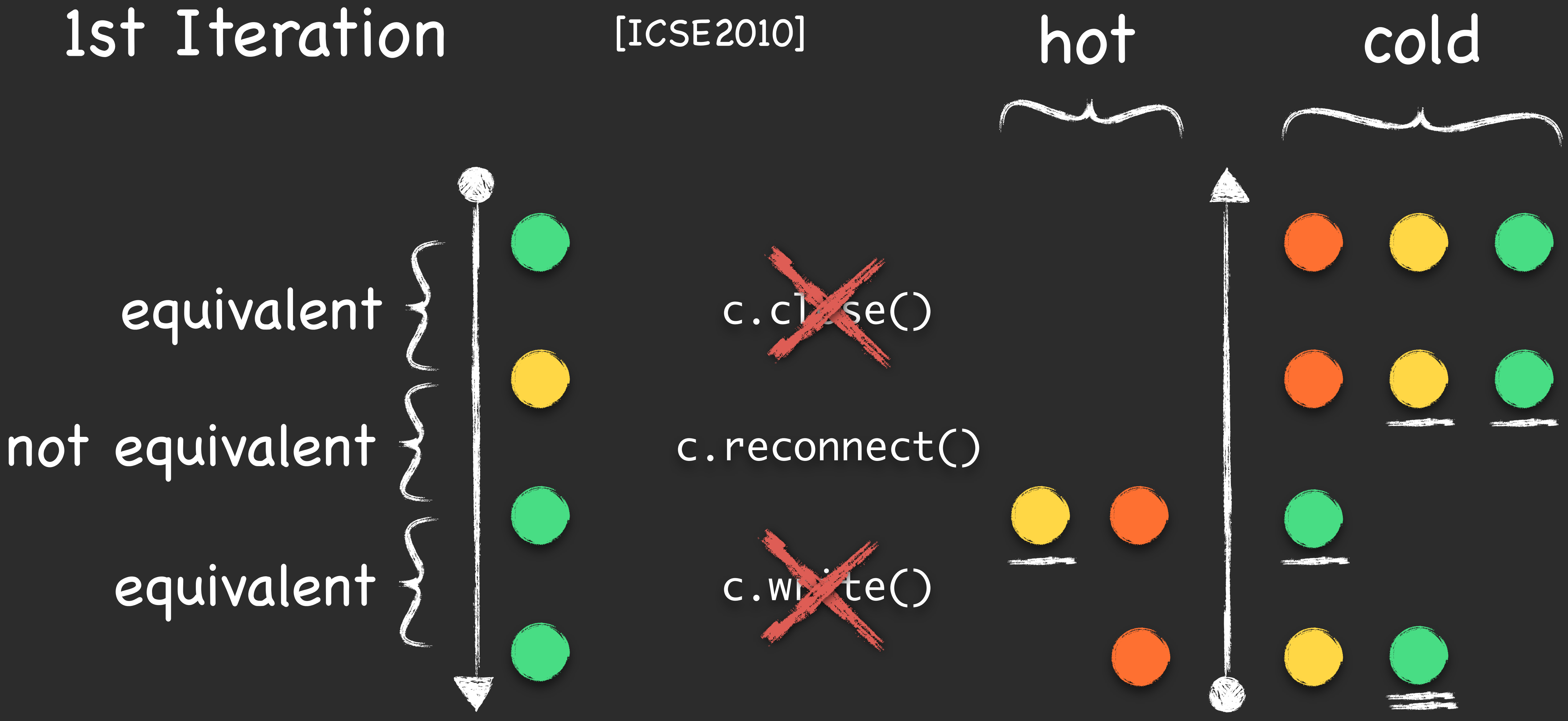
[ICSE2010]

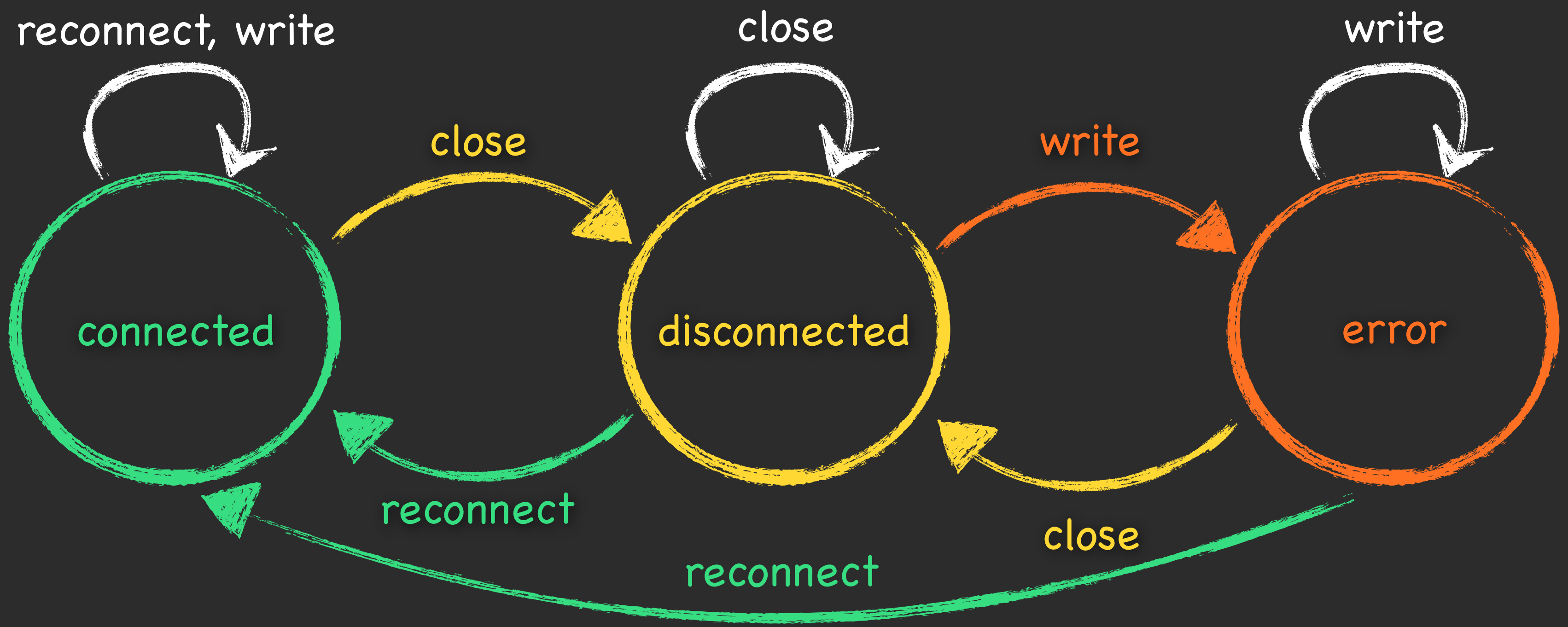




1st Iteration

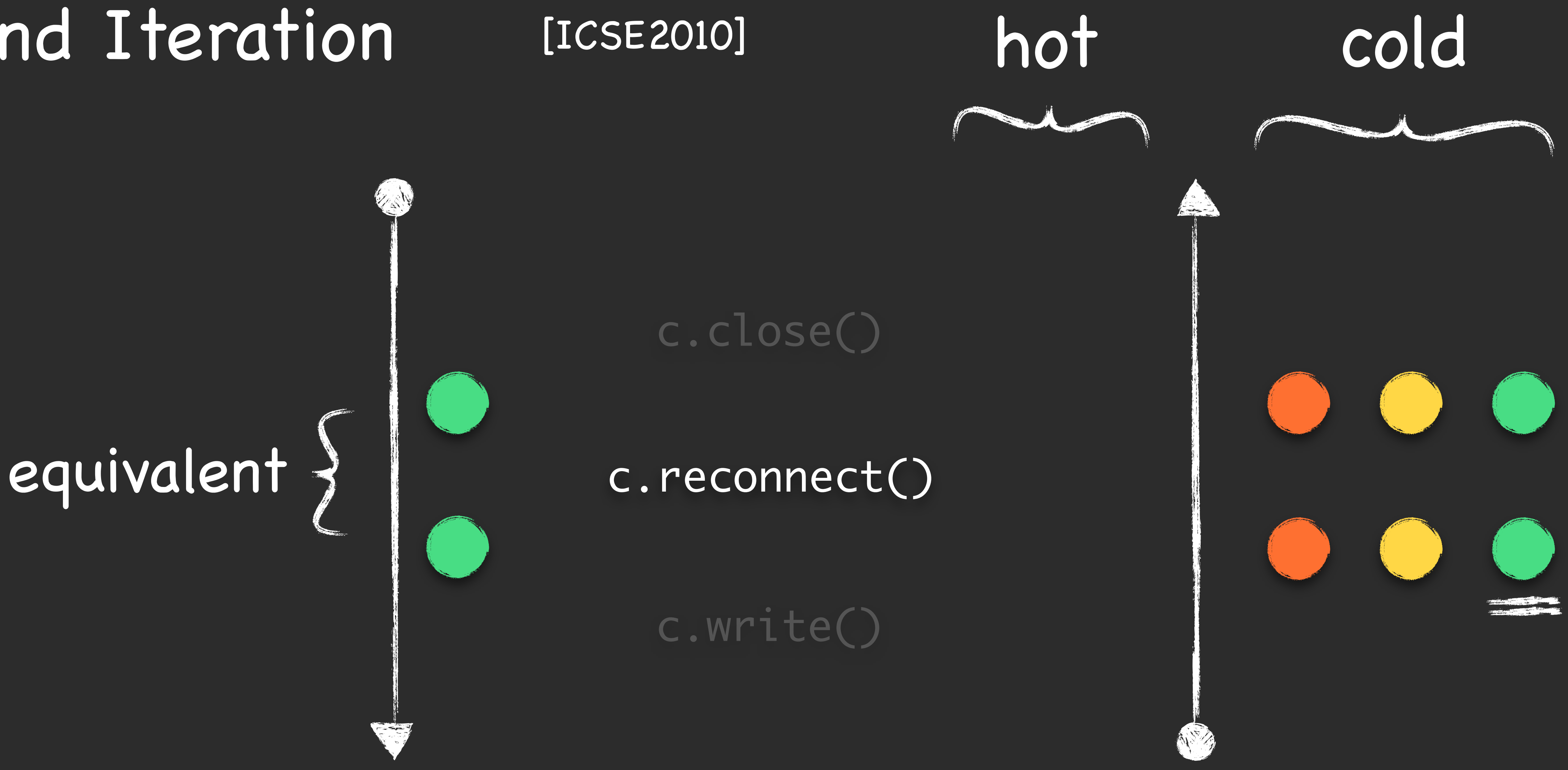
[ICSE2010]

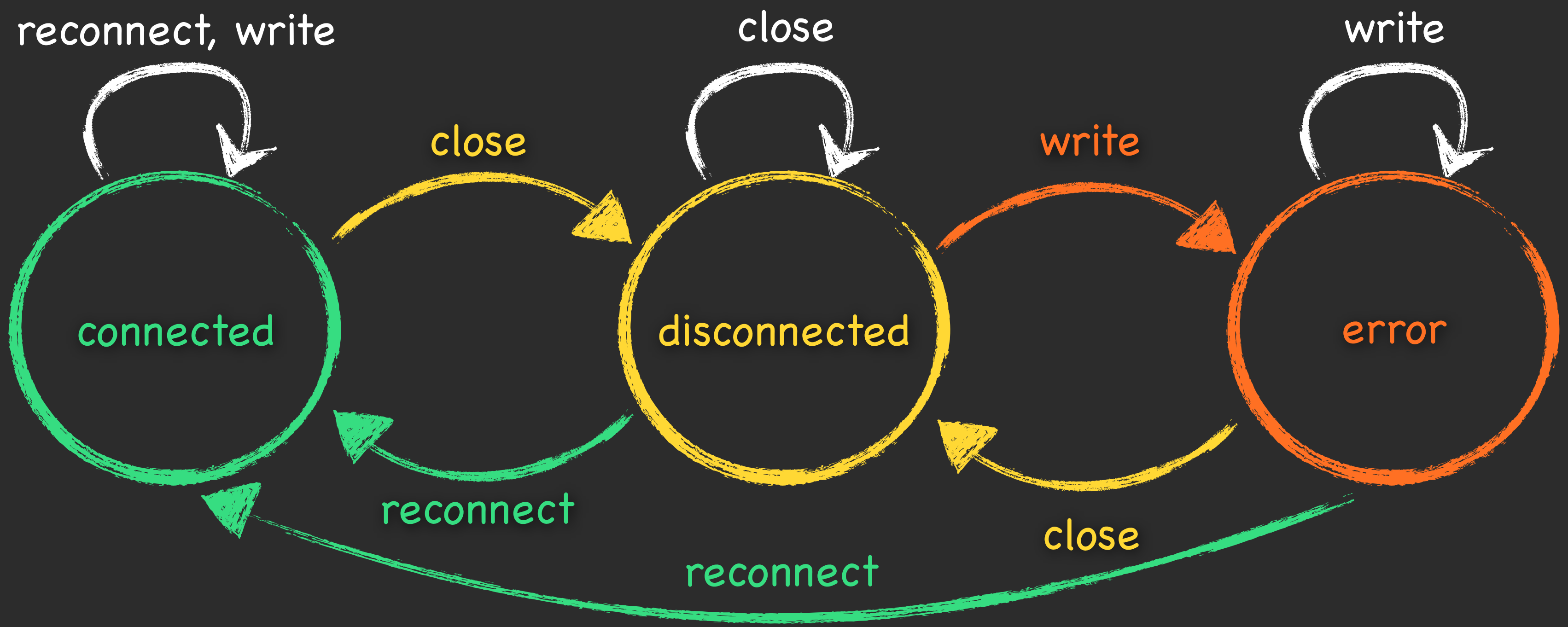




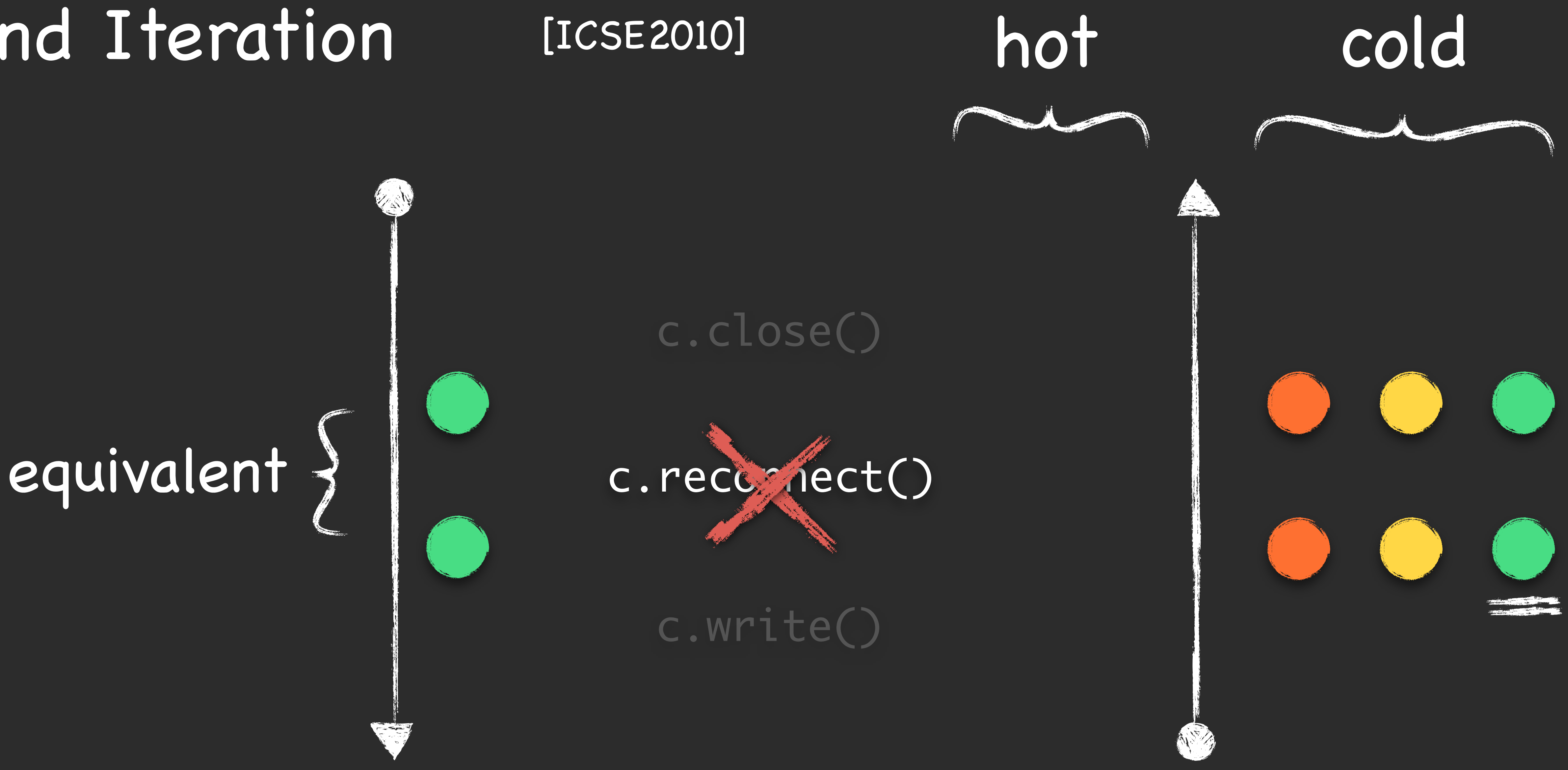
2nd Iteration

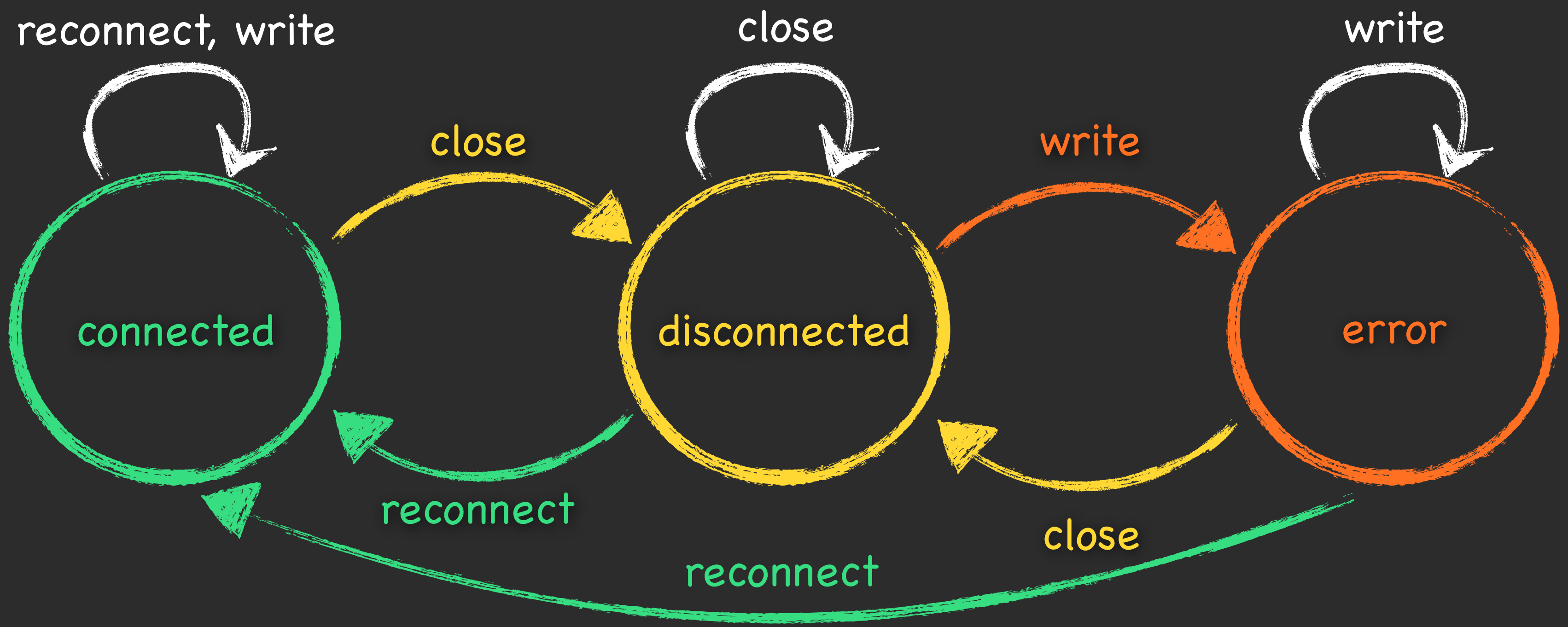
[ICSE2010]





2nd Iteration [ICSE2010]





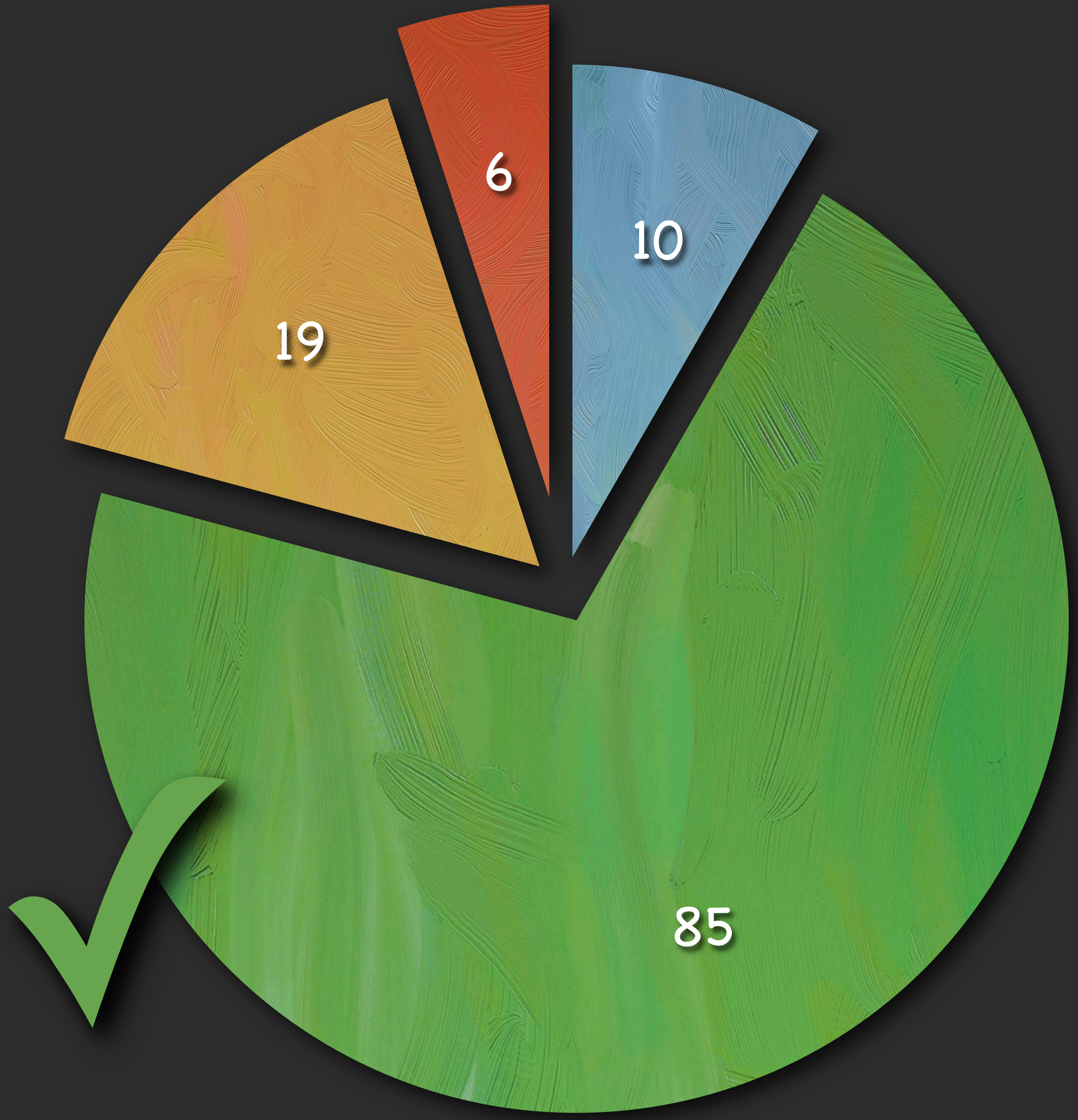
General solution

- On top of AspectBench Compiler / Soot
- Full Java support
    - recursion
    - exceptions
    - multi-object properties
    - reflection*

*[Program Surfing I, tomorrow 4pm]

10 Programs (DaCapo suite, 1.5MLOC)
x 12 Properties
= 120 Test cases

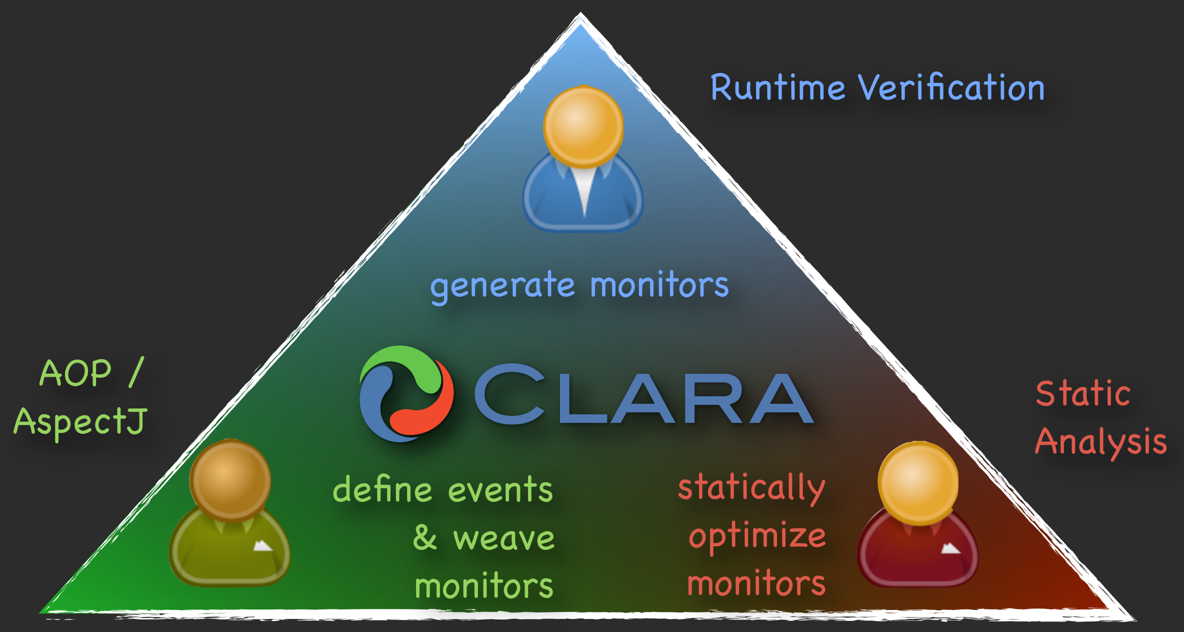
10 Programs (DaCapo suite, 1.5MLOC)
x 12 Properties
= 120 Test cases



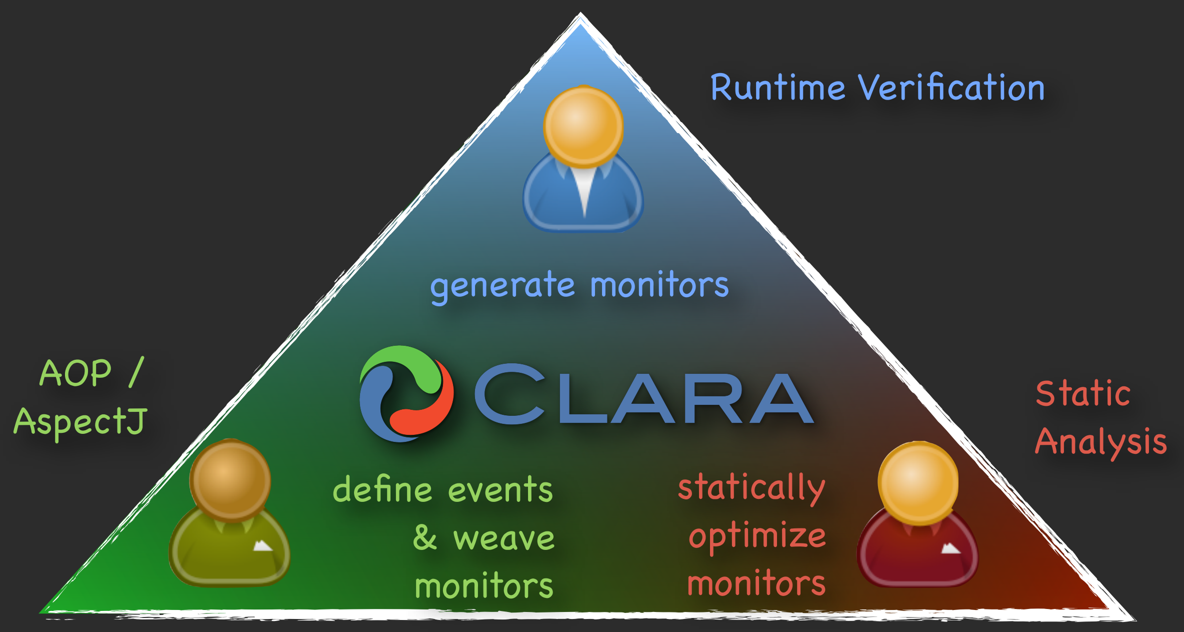
- ● trivially safe
- ● proven safe
- ● "just" optimized
- ● violations found

[ICSE2010]

http://bodden.de/clara/

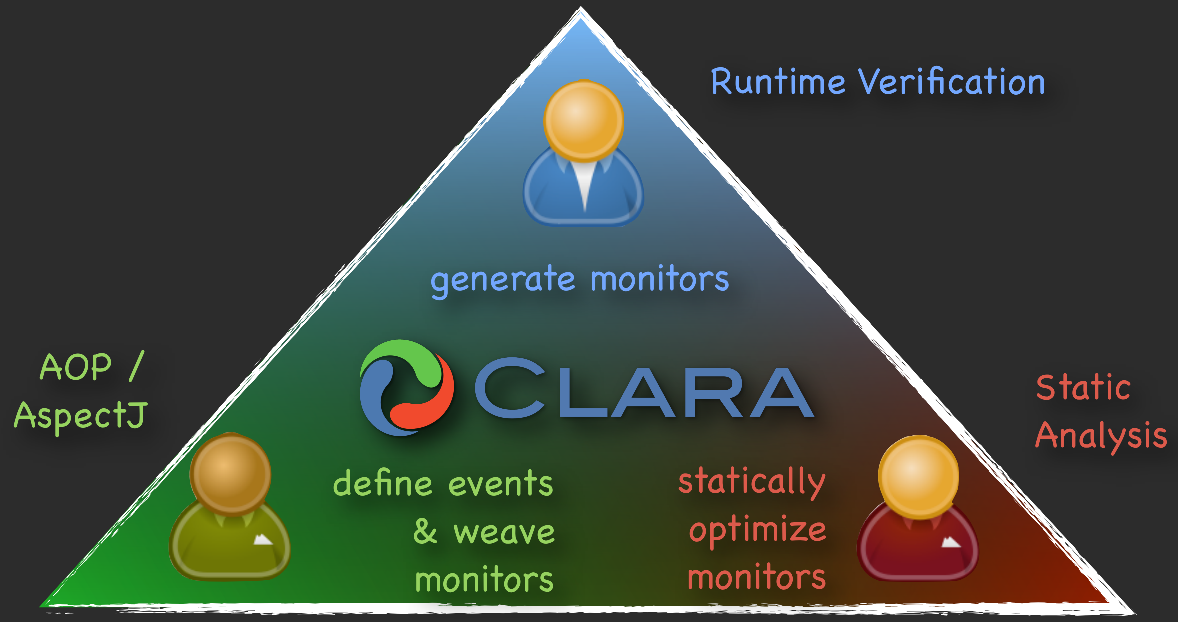


http://bodden.de/clara/

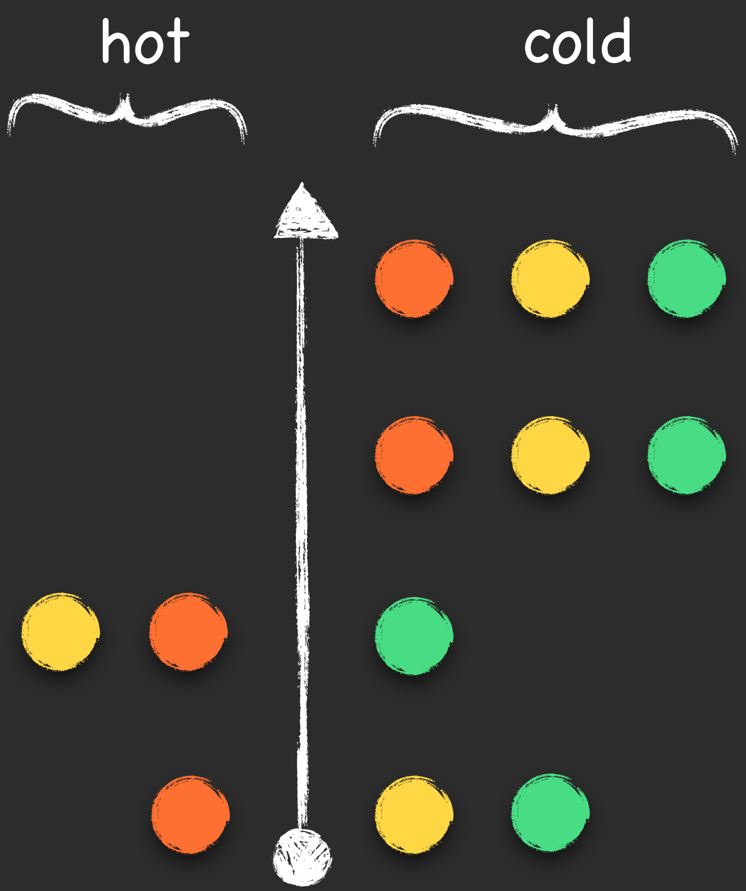


```
dependency{
    disconnect, write, reconnect;
    initial  connected: disconnect -> connected,
                         write -> connected,
                         reconnect -> connected,
                         disconnect -> disconnected;
                      disconnect: disconnect -> disconnected,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

http://bodden.de/clara/

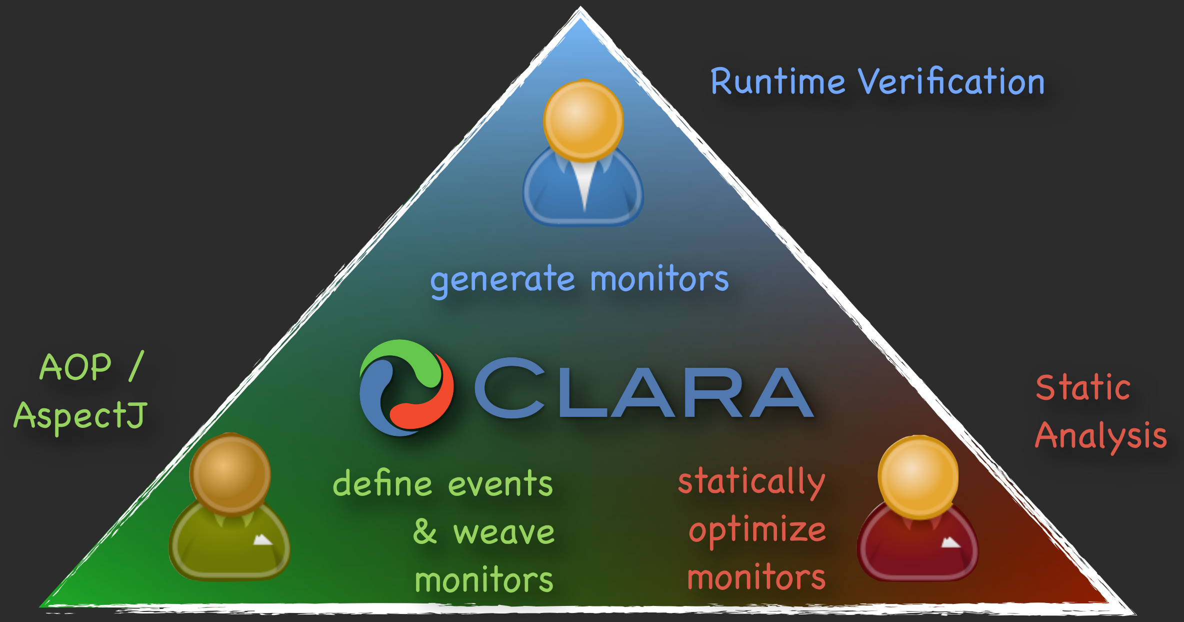


```
dependency{
    disconnect, write, reconnect;
    initial  connected: disconnect -> connected,
                        write -> connected,
                        reconnect -> connected,
                        disconnect -> disconnected;
                    disconnect: disconnect -> disconnected,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

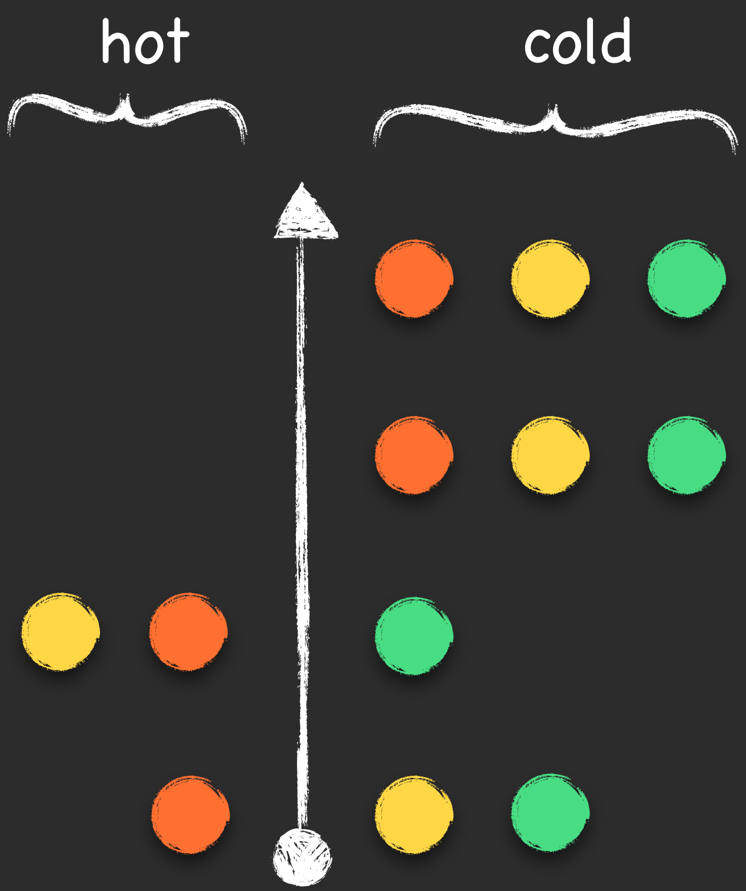


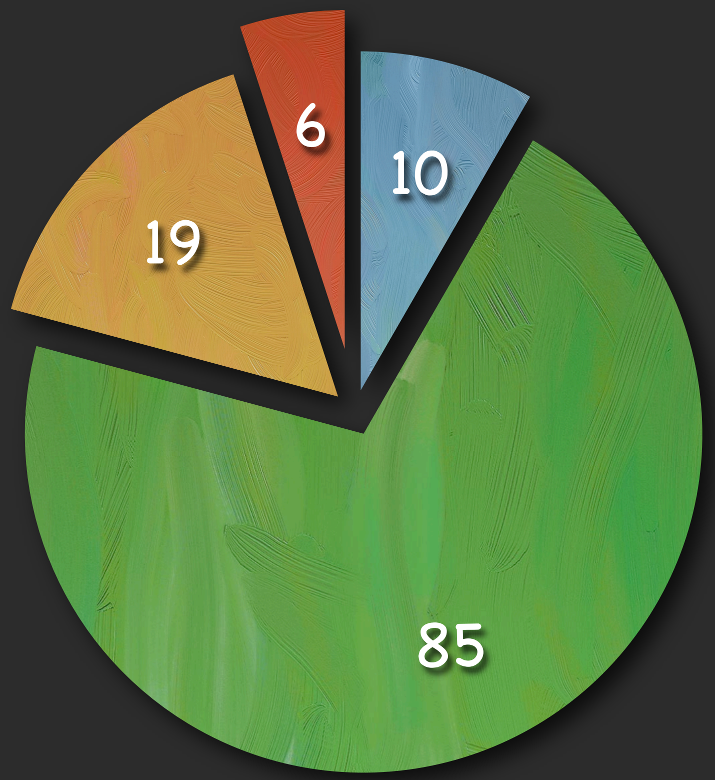
Eric Bodden

http://bodden.de/clara/



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





Eric Bodden