Mining Operational Preconditions

Andrzej Wasylkowski • Andreas Zeller
Saarland University

aspectj

crosscutting objects for better modularity

bug.aj

```
@interface A {}

aspect Test {
   declare @field : @A int var* : @A;
   declare @field : int var* : @A;

   interface Subject {}

   public int Subject.vara;
   public int Subject.varb;
}

class X implements Test.Subject {}
```

```
  private static void ajc$postClinit()    org.aspectj.weaver.AjAttribute$AjSynth
etic@abfbc6
:
                    NEW Test   (line 1)
                    DUP
                    INVOKESPECIAL Test.<init> ()V
                    PUTSTATIC Test.ajc$perSingletonInstance LTest;
                    RETURN
  end private static void ajc$postClinit()
end public class Test


Exception thrown from AspectJ 1.5.3

This might be logged as a bug already -- find current bugs at
  http://bugs.eclipse.org/bugs/buglist.cgi?product=AspectJ&component=Compiler

Bugs for exceptions thrown have titles File:line from the top stack,
e.g., "SomeFile.java:243"

If you don't find the exception below in a bug, please add a new bug
at http://bugs.eclipse.org/bugs/enter_bug.cgi?product=AspectJ
To make the bug a priority, please include a test program
that can reproduce this exception.
```

ajc Stack Trace

```
java.util.NoSuchElementException
   at java.util.AbstractList$Itr
      .next(AbstractList.java:427)
  at org.aspectj.weaver.bcel.BcelClassWeaver
      .weaveAtFieldRepeatedly
      (BcelClassWeaver.java:1016)
```

ajc Stack Trace

```
java.util.NoSuchElementException
   at java.util.AbstractList$Itr
      .next(AbstractList.java:427)
  at org.aspectj.weaver.bcel.BcelClassWeaver
      .weaveAtFieldRepeatedly
      (BcelClassWeaver.java:1016)
```

weaveAtFieldRepeatedly

```
for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
    }
}
```

weaveAtFieldRepeatedly

```
for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
    }
}
```

should be iter2

weaveAtFieldRepeatedly

```
for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
    }
}
```

should be iter2

- Invalid iterator usage:
  hasNext() should precede next()

Preconditions

Preconditions

- Invoking next() with no next element violates a *precondition*

Preconditions

- Invoking next() with no next element violates a *precondition*
- Traditional preconditions are axiomatic – describing the state of the system

Preconditions

- Invoking next() with no next element violates a *precondition*
- Traditional preconditions are axiomatic – describing the state of the system
- How do we reach this state?

Preconditions

Preconditions

```
close(int fildes)
```

Preconditions

`close(int fildes)`

- Axiomatic: *fildes* is a valid file descriptor

Preconditions

`close(int fildes)`

- Axiomatic: *fildes* is a valid file descriptor
- Operational: *fildes* stems from a call to *open*()
  with read() and write() calls in between

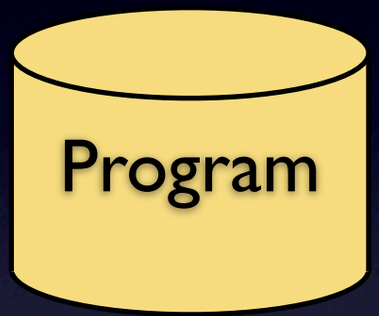
Preconditions

`close(int fildes)`

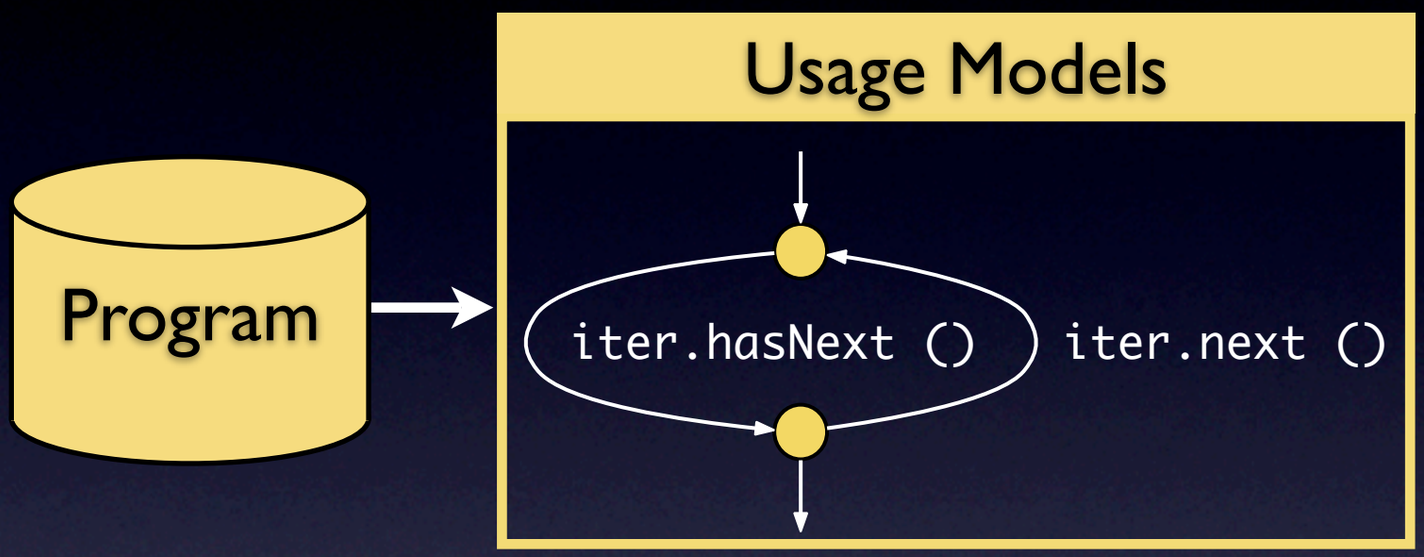
- Axiomatic: *fildes* is a valid file descriptor
- Operational: *fildes* stems from a call to *open*()
  with read() and write() calls in between
- Can we check operational preconditions?

OP-Miner

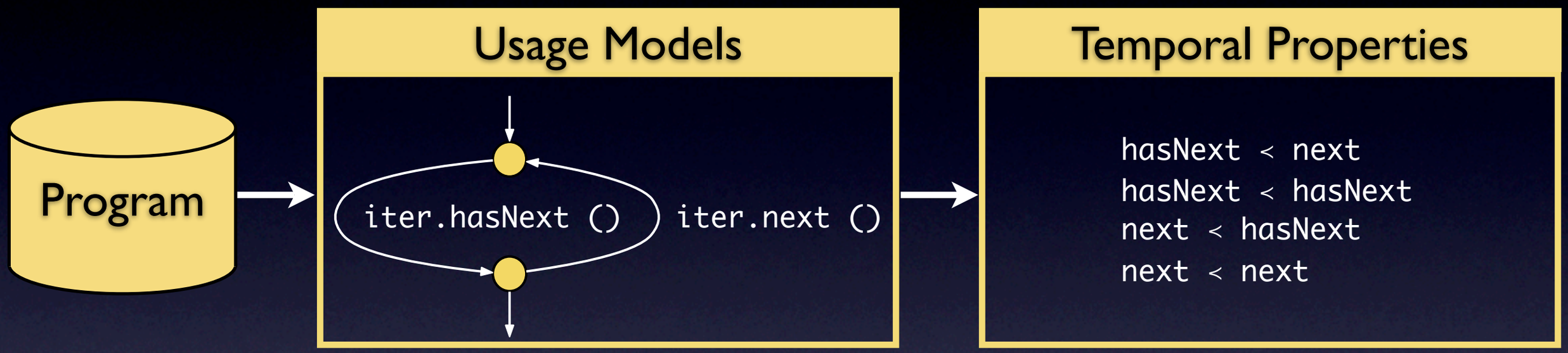
OP-Miner



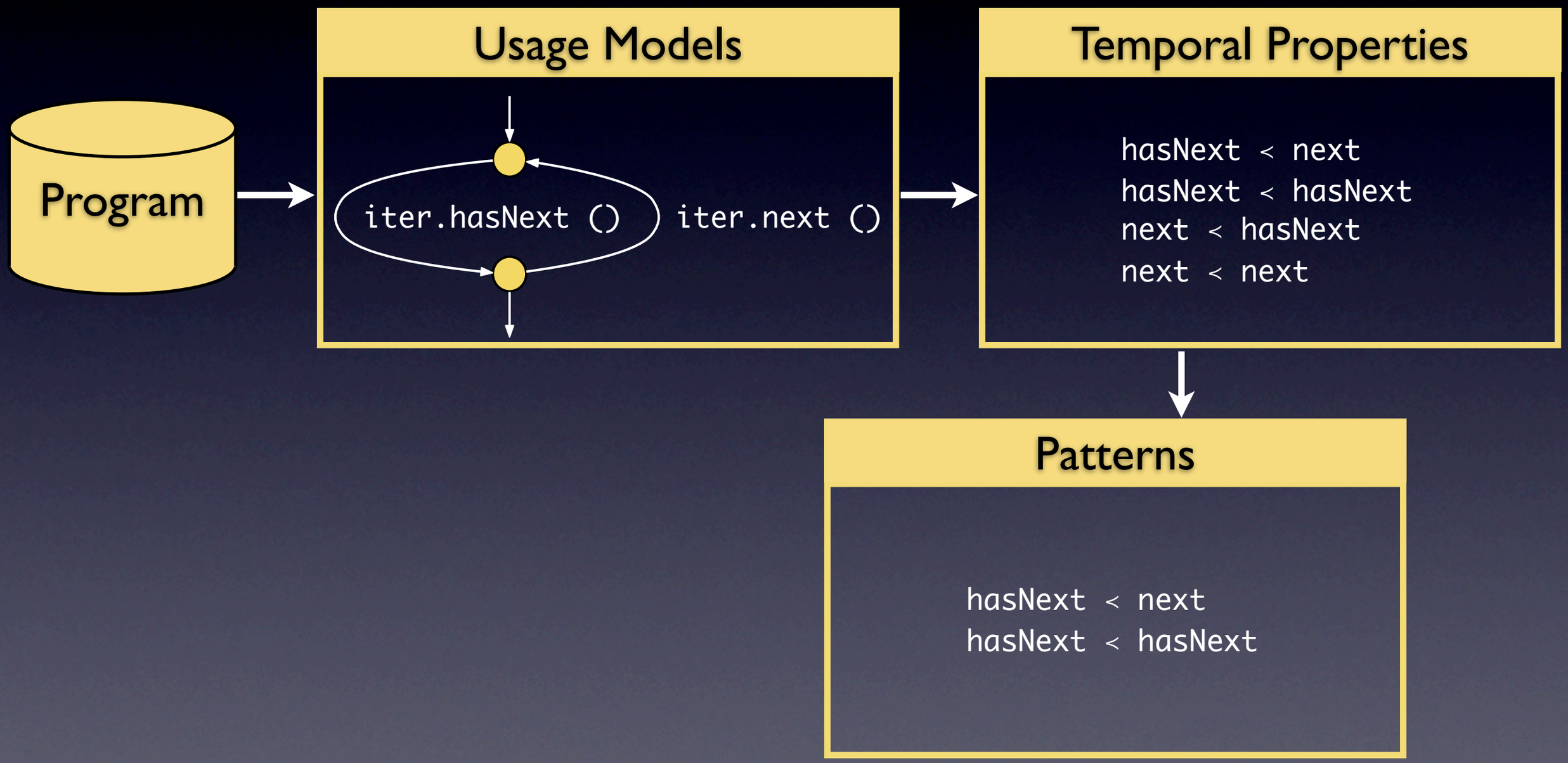
OP-Miner



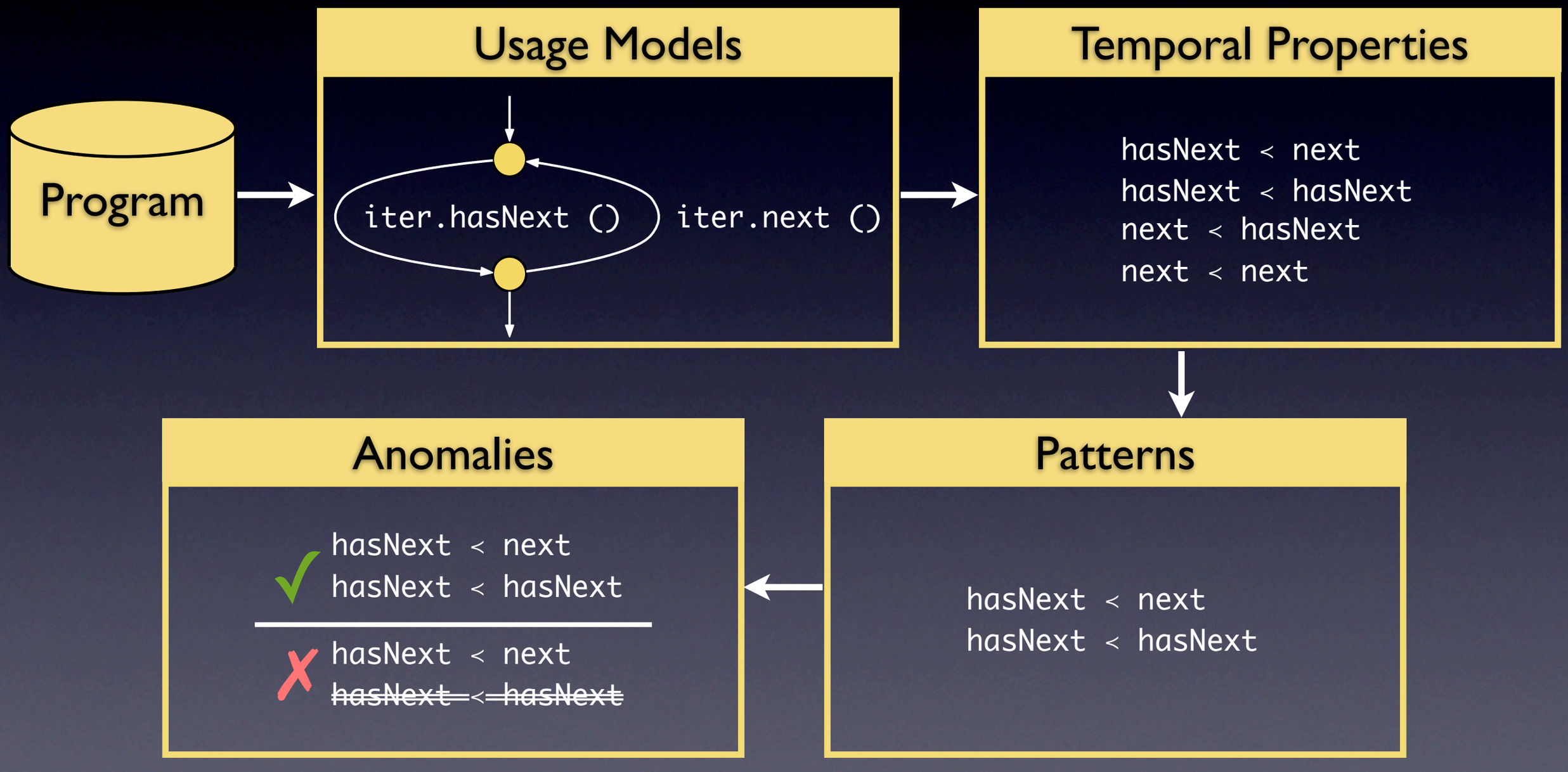
OP-Miner


Program
Usage Models
iter.hasNext ()
iter.next ()
Temporal Properties
hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next

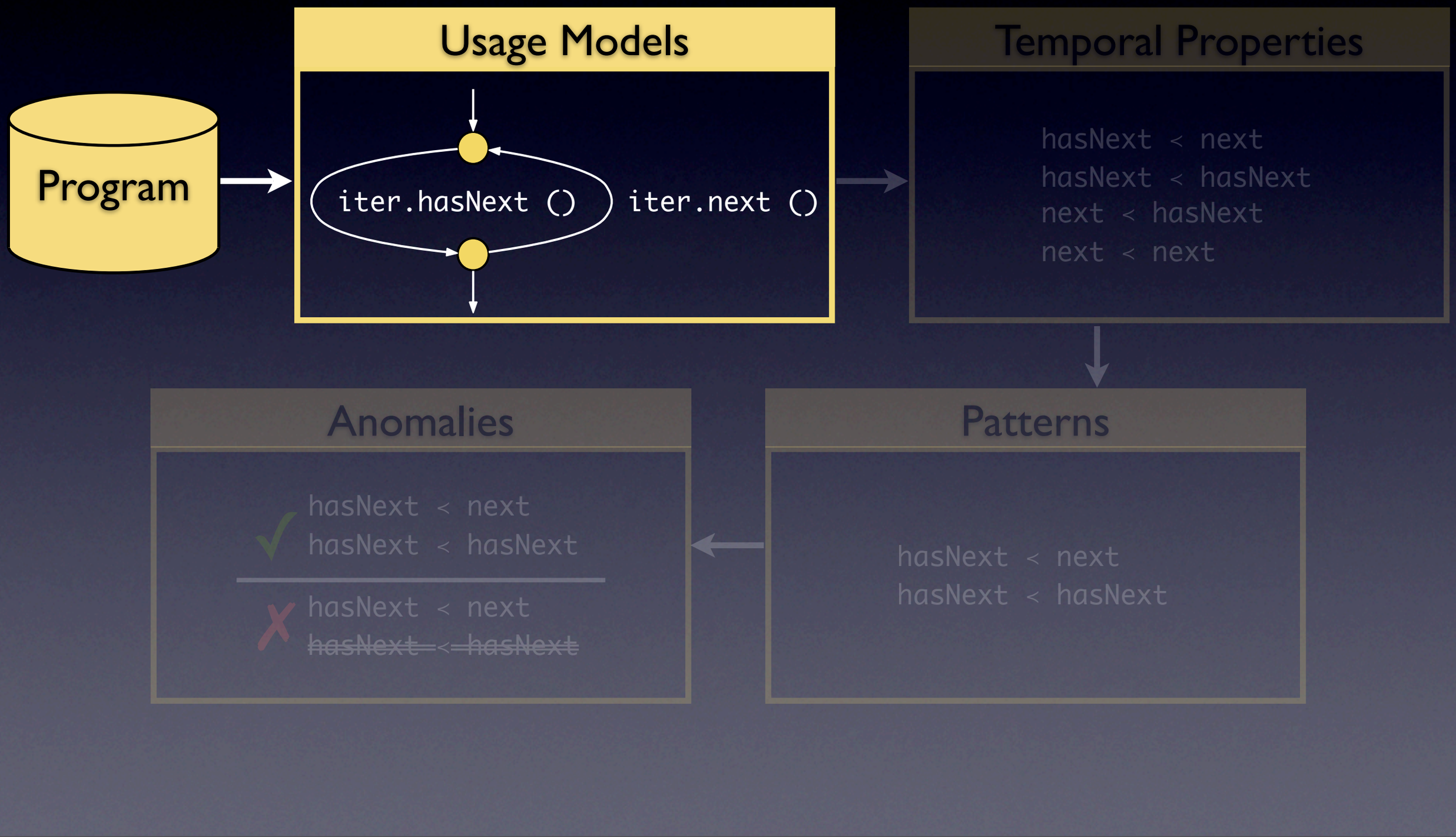
OP-Miner


Program
Usage Models
iter.hasNext ()
iter.next ()
Temporal Properties
hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next
Patterns
hasNext < next
hasNext < hasNext

OP-Miner


Usage Models
Program
iter.hasNext ()
iter.next ()
Temporal Properties
hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next
Anomalies
hasNext < next
hasNext < hasNext
hasNext < next
hasNext < hasNext
Patterns
hasNext < next
hasNext < hasNext

OP-Miner


Program
Usage Models
iter.hasNext ()
iter.next ()
Temporal Properties
hasNext < next
hasNext < hasNext
next < hasNext
next < next
Patterns
hasNext < next
hasNext < hasNext
Anomalies
hasNext < next
hasNext < hasNext
hasNext < next
hasNext < hasNext

Method Models

```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```

Method Models

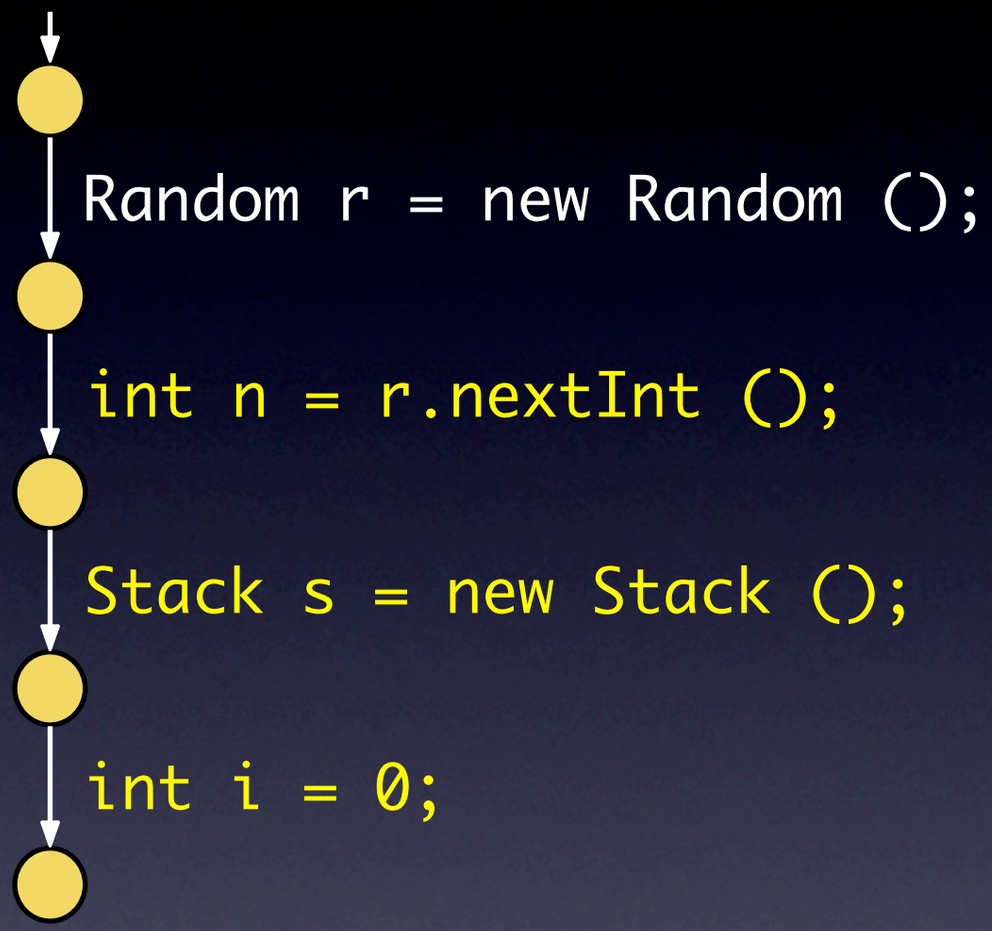
```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```

Method Models


```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```

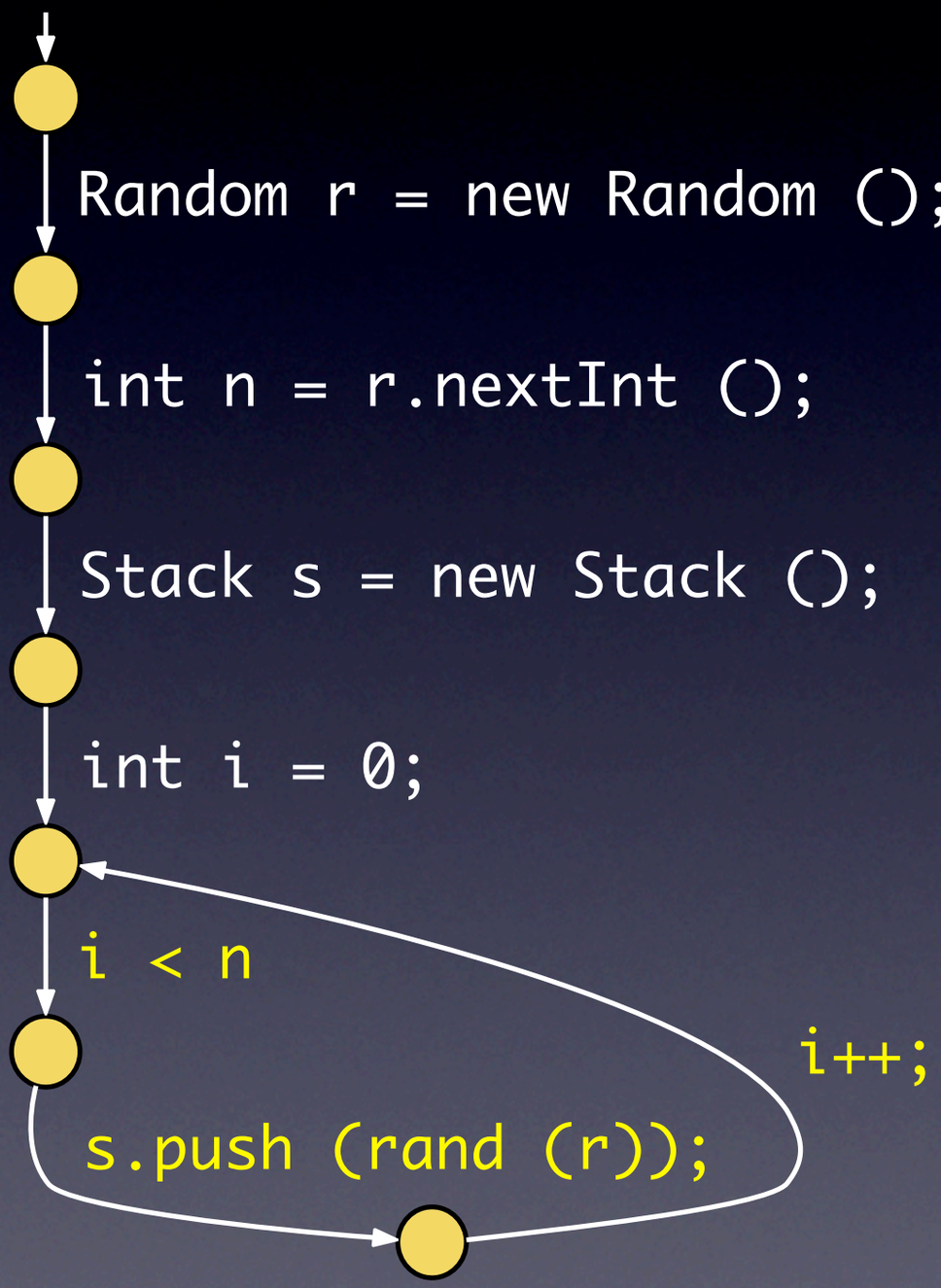
Method Models

```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```



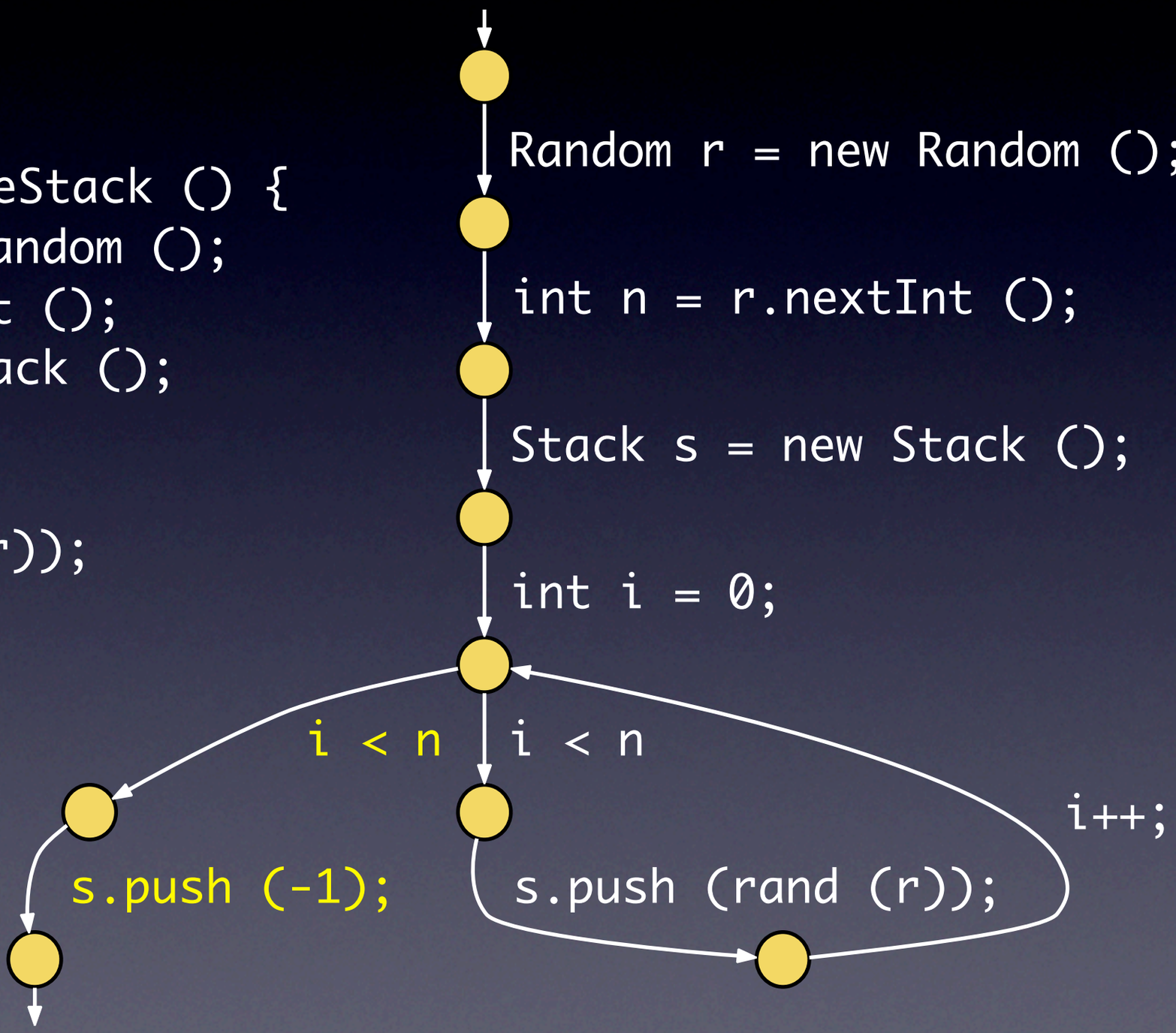
Method Models

```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```



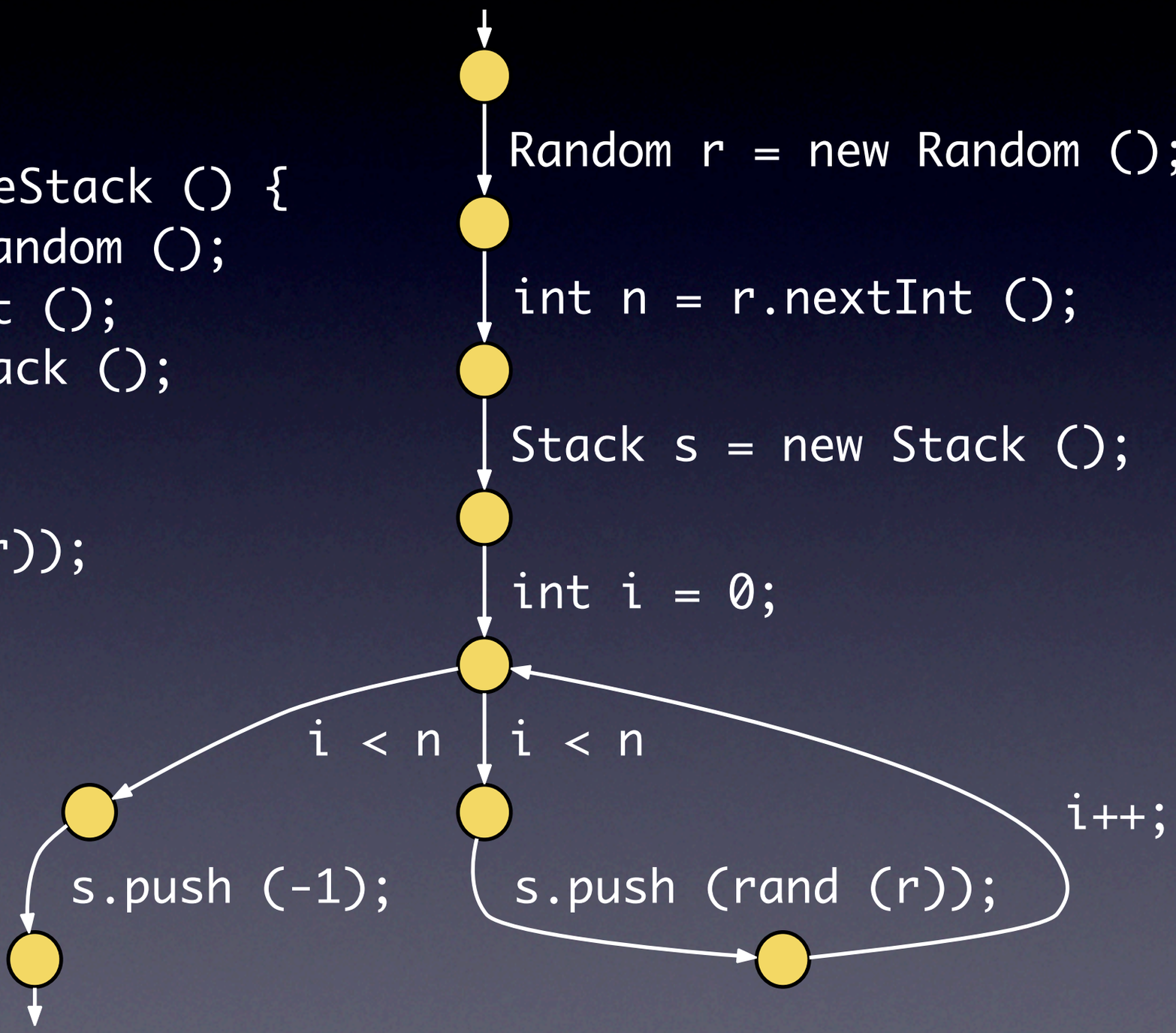
Method Models

```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```

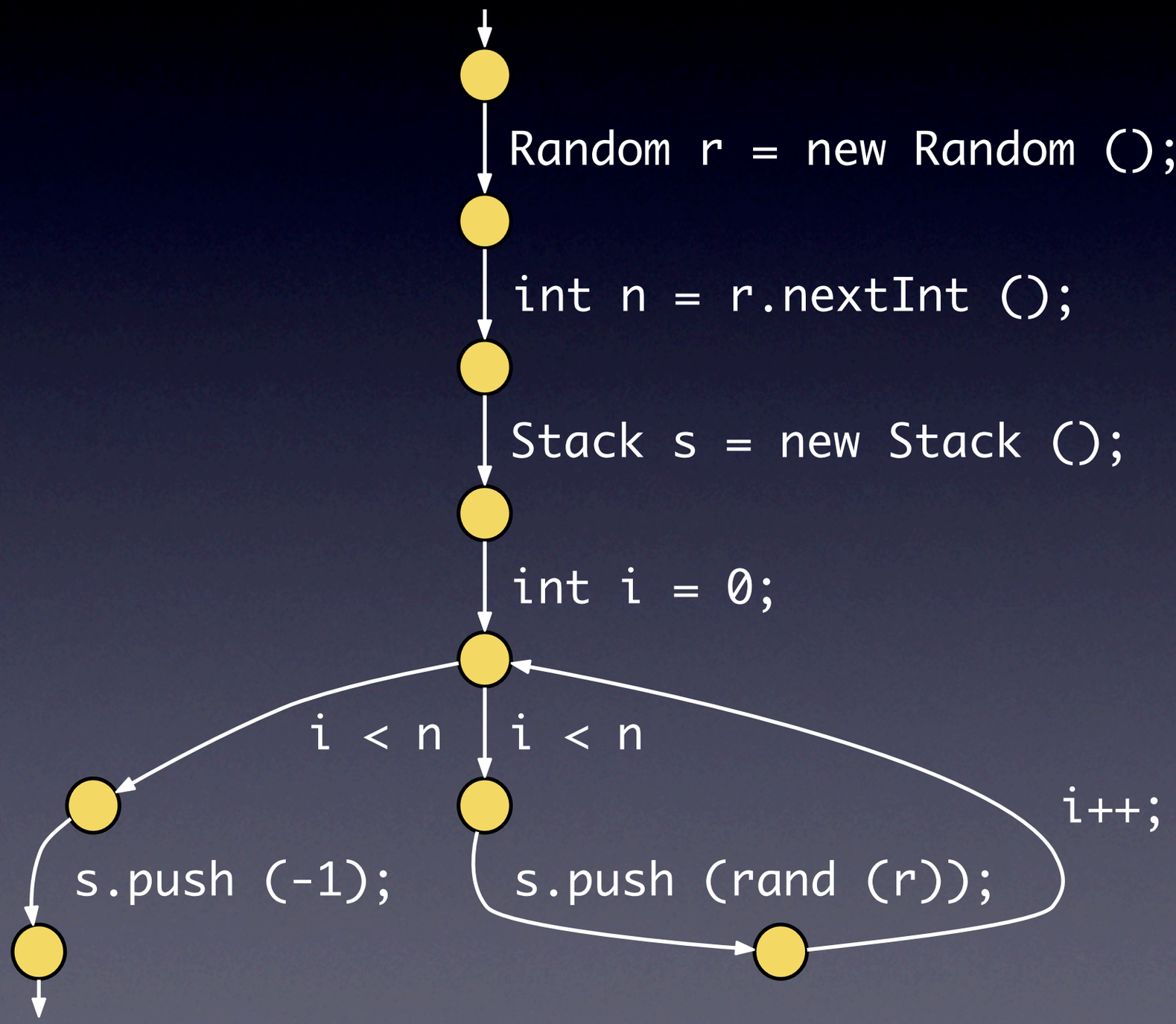


Method Models

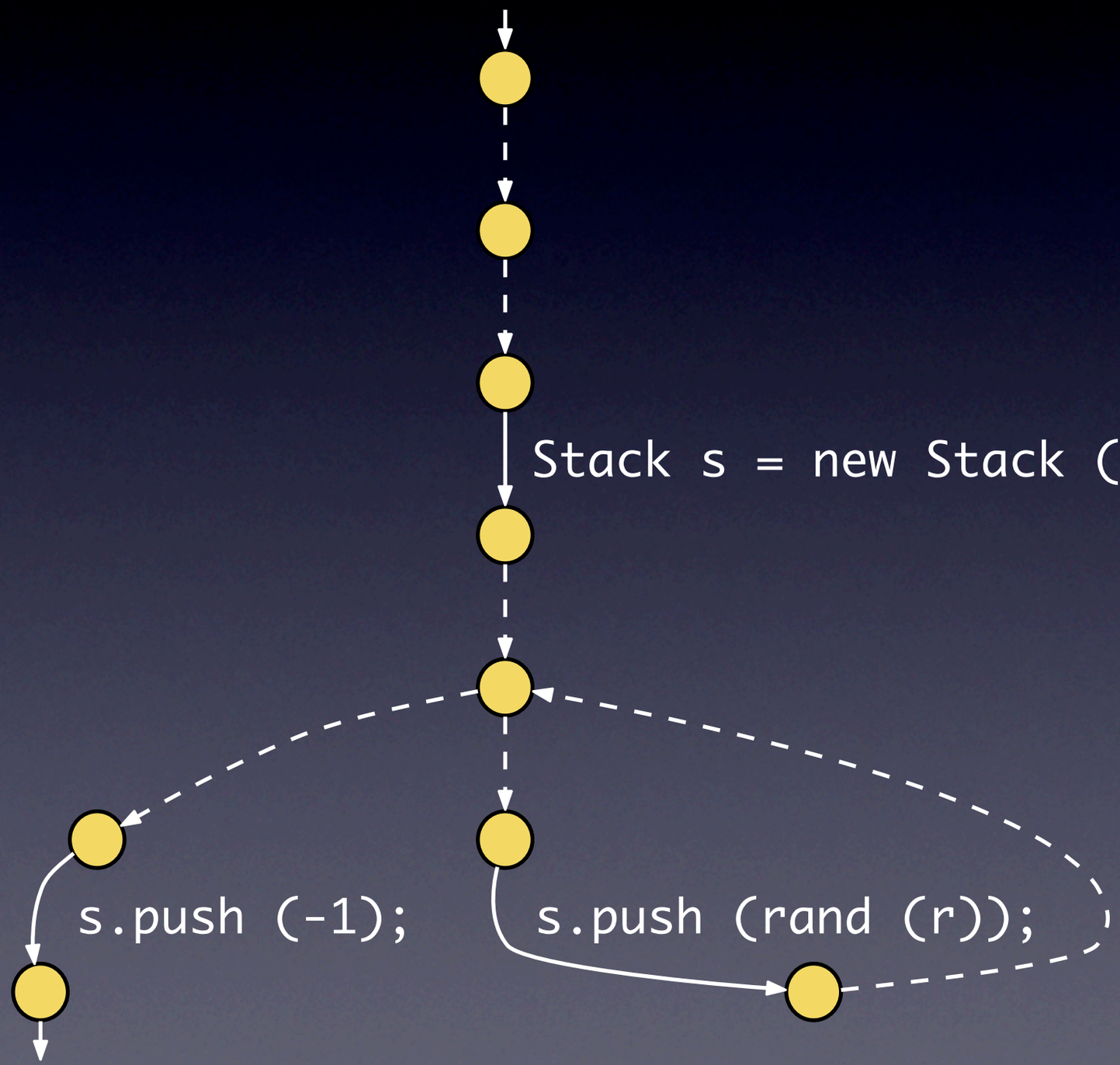
```
public Stack createStack () {
  Random r = new Random ();
  int n = r.nextInt ();
  Stack s = new Stack ();
  int i = 0;
  while (i < n) {
    s.push (rand (r));
    i++;
  }
  s.push (-1);
  return s;
}
```



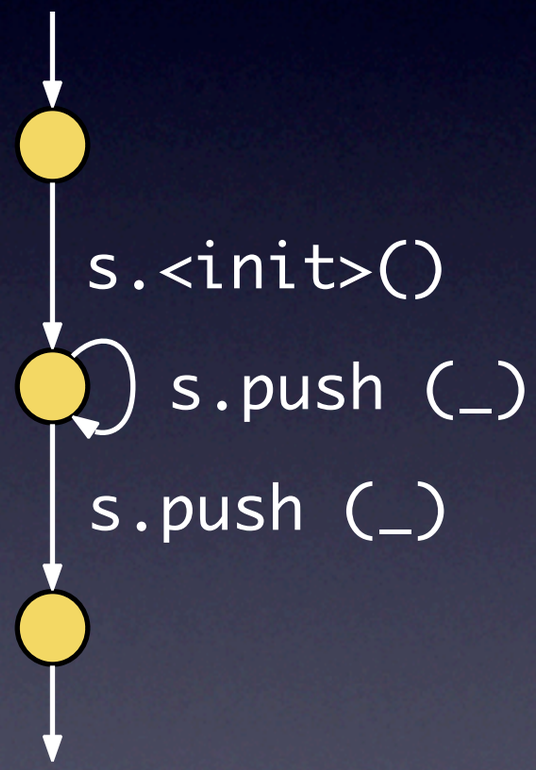
Usage Models



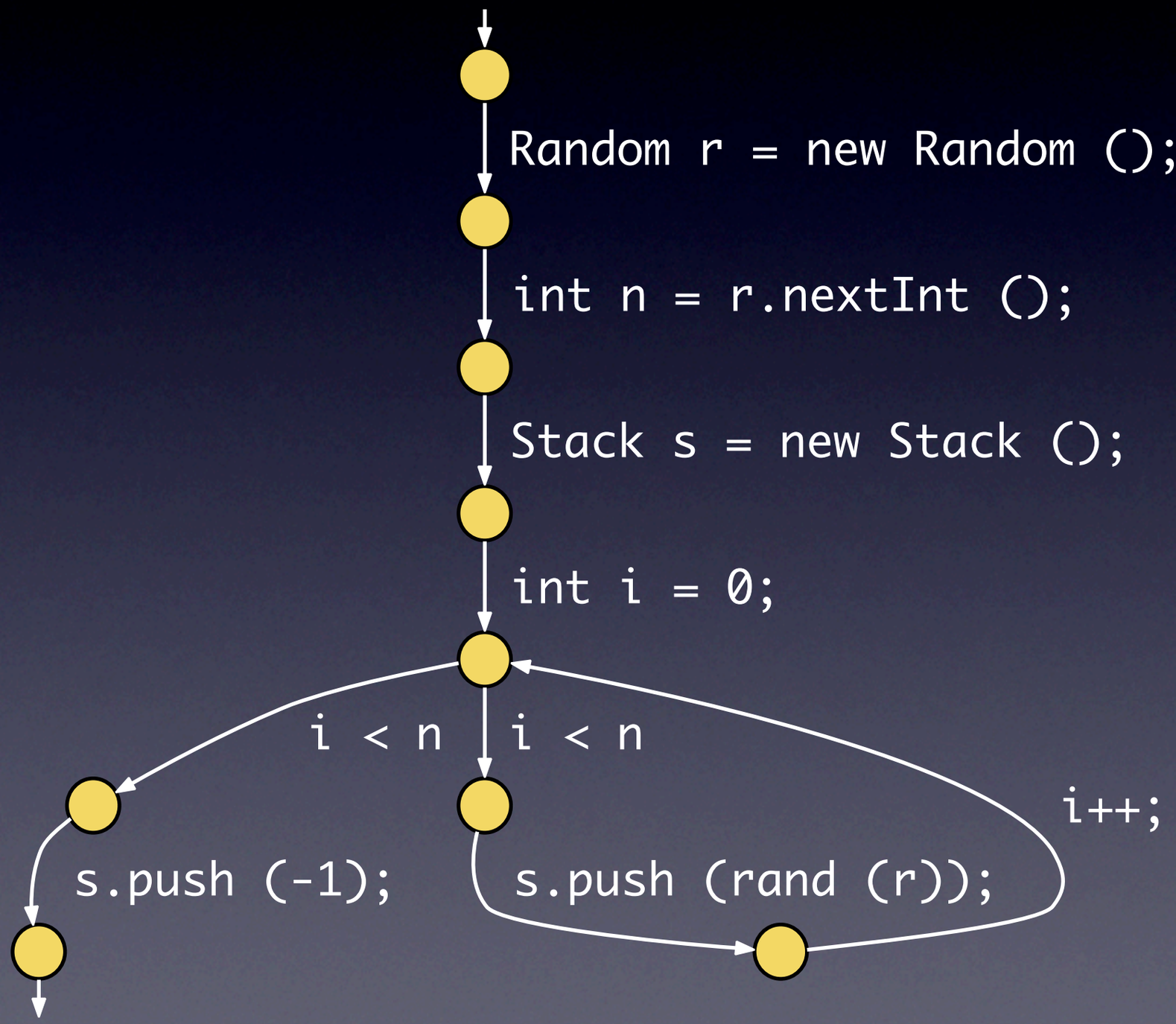
Usage Models



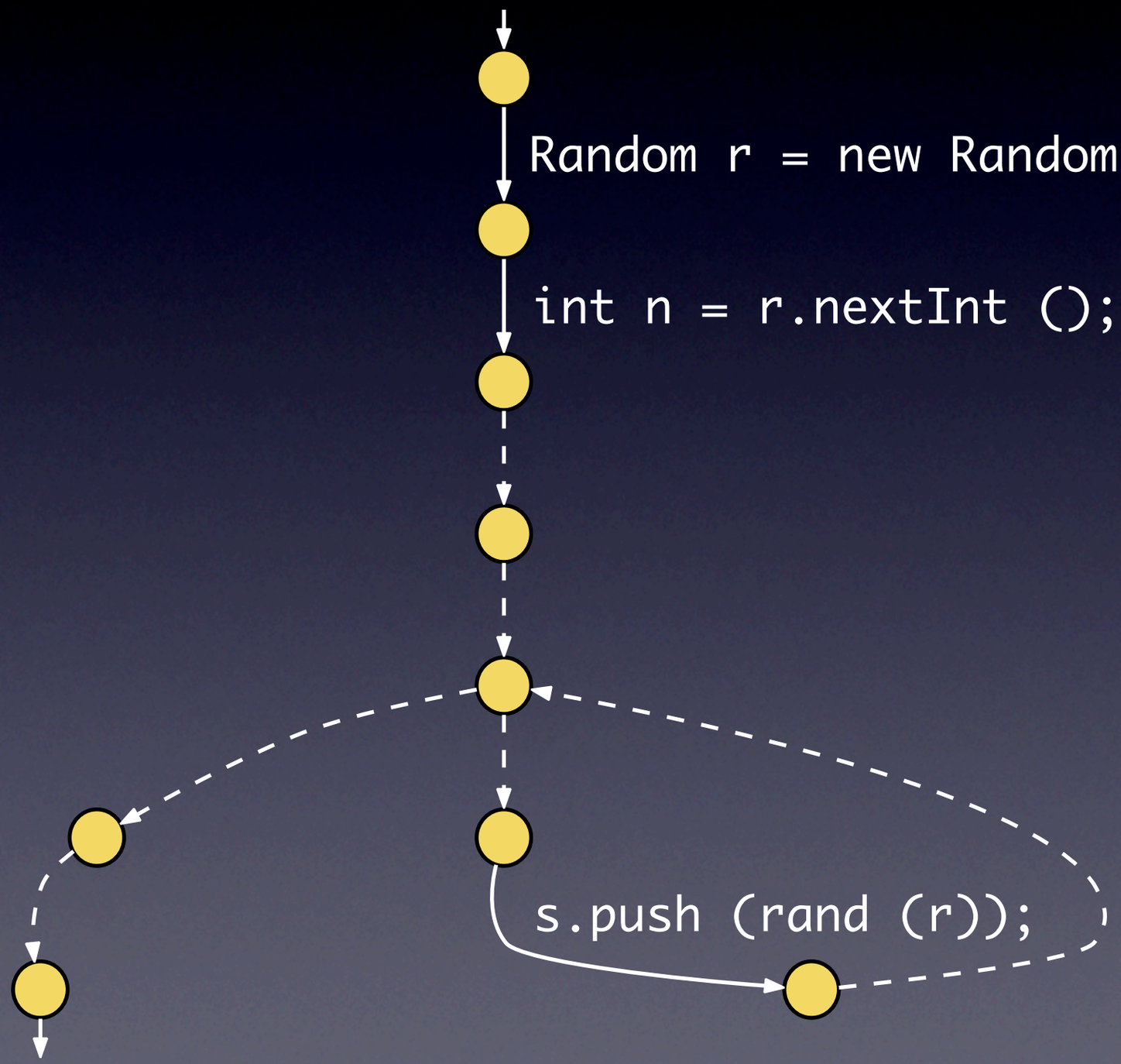
Usage Models



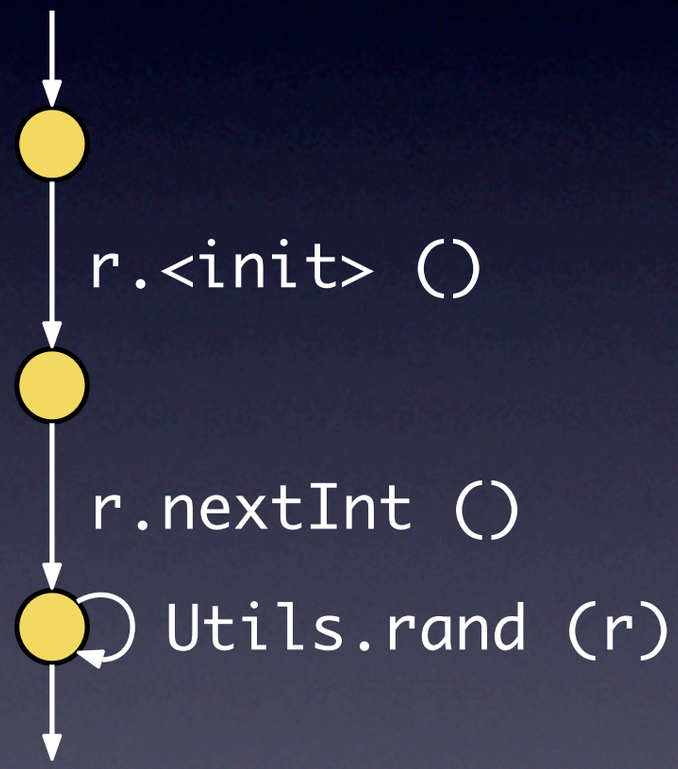
Usage Models



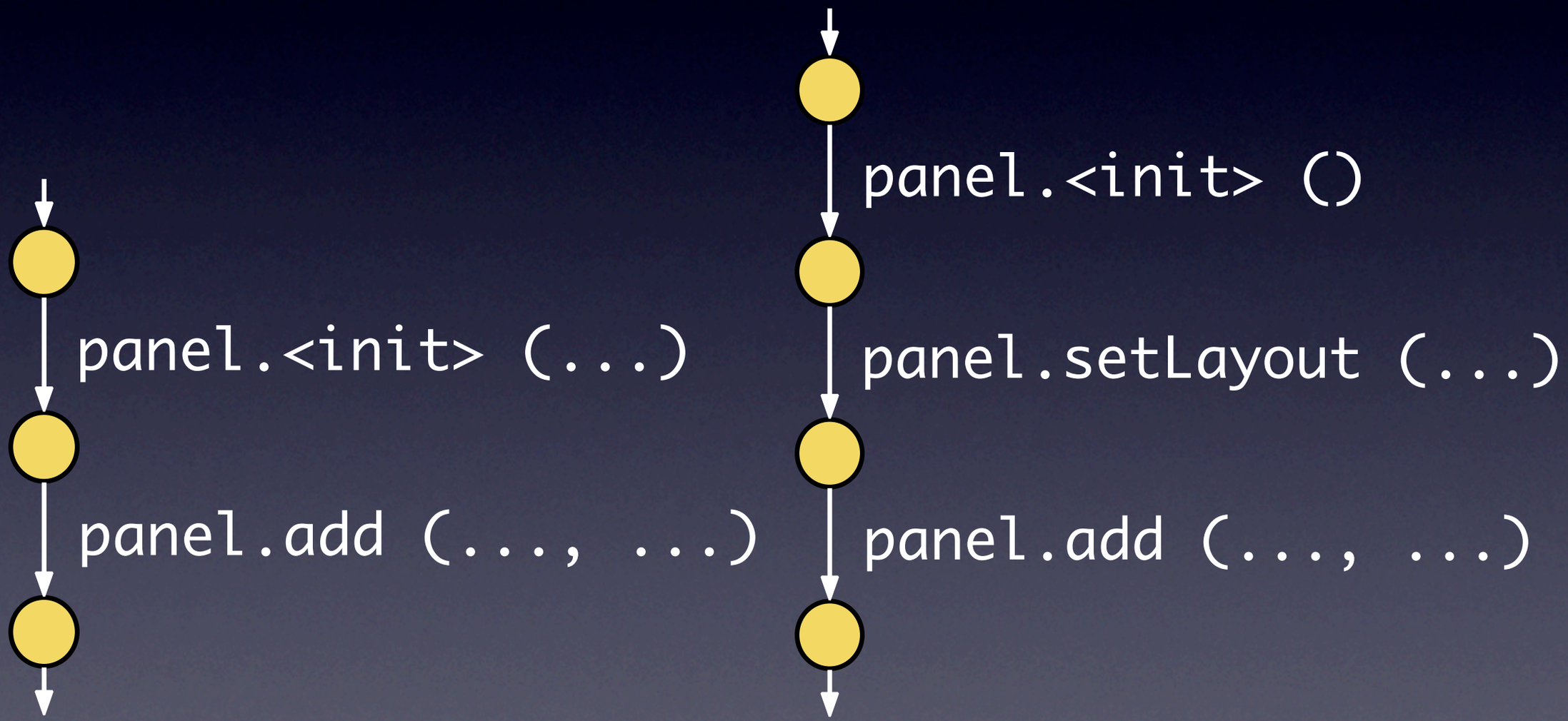
Usage Models



Usage Models



JPanel.add()



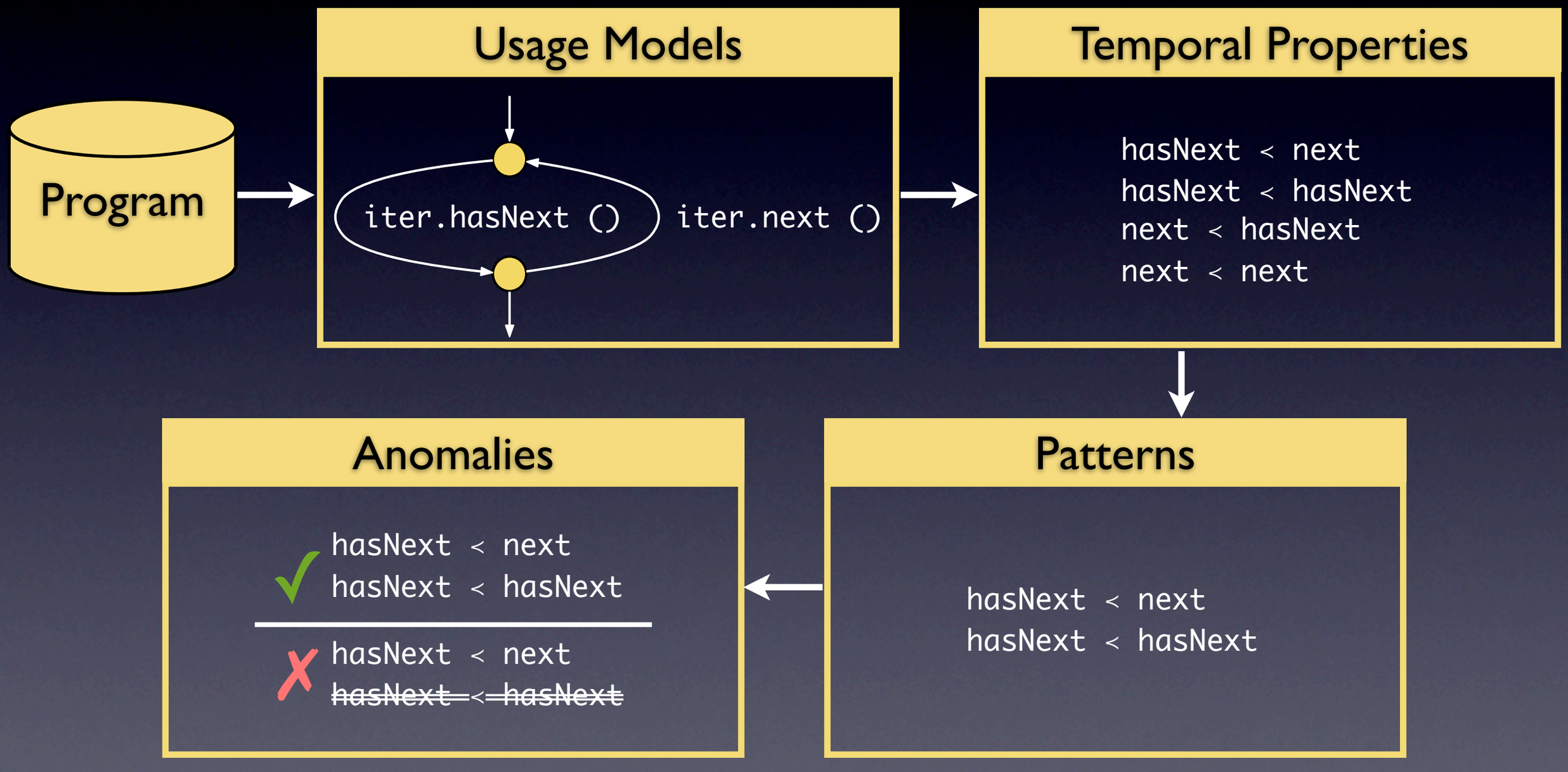
ASTNode.reapPropertyList()

Resource.getFlags()

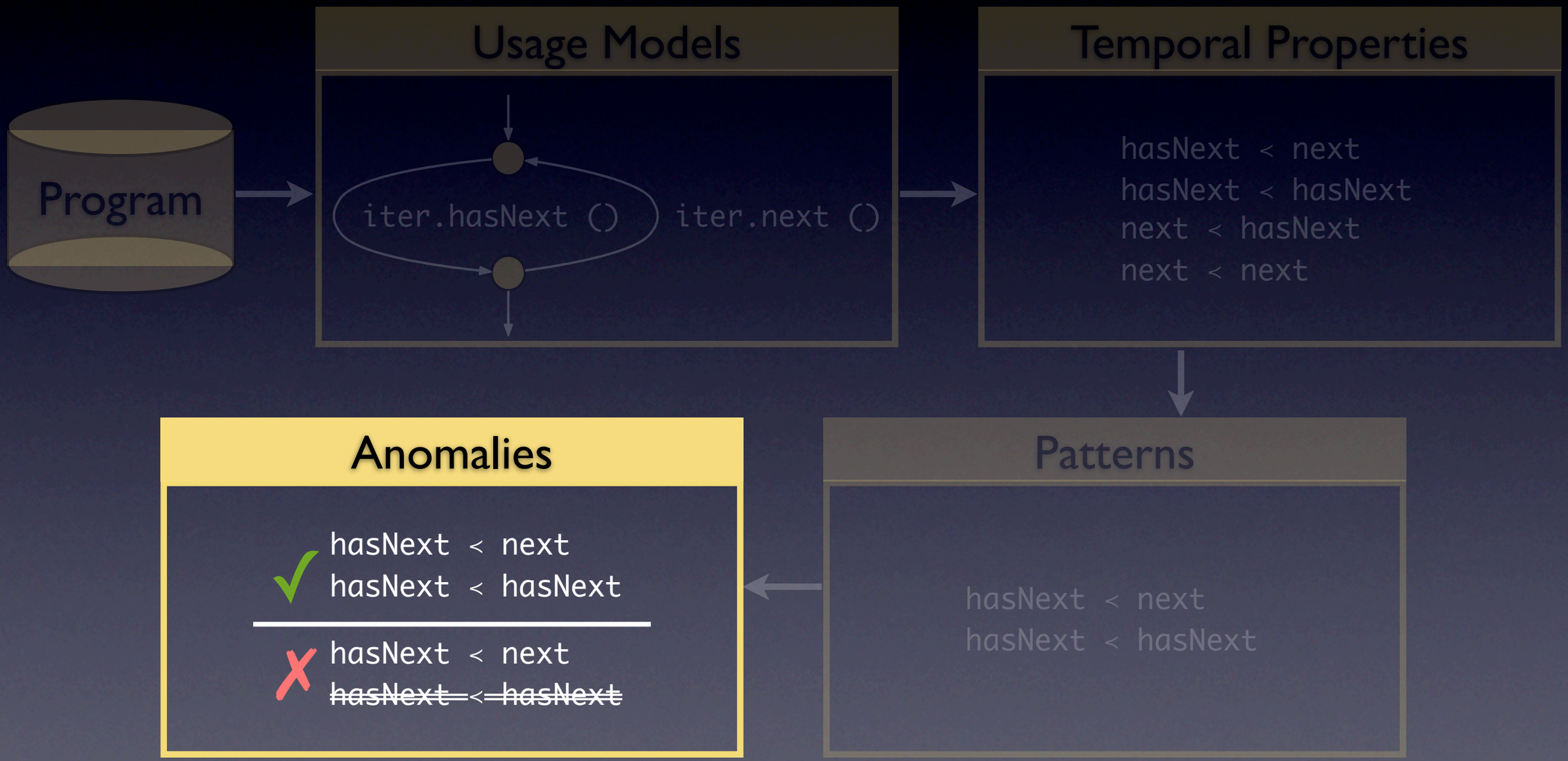
```
resource.getResourceInfo (..., ...)

resource.getFlags (...)
```

OP-Miner


Usage Models
Program
iter.hasNext ()
iter.next ()
Temporal Properties
hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next
Anomalies
hasNext < next
hasNext < hasNext
hasNext < next
hasNext < hasNext
Patterns
hasNext < next
hasNext < hasNext

OP-Miner

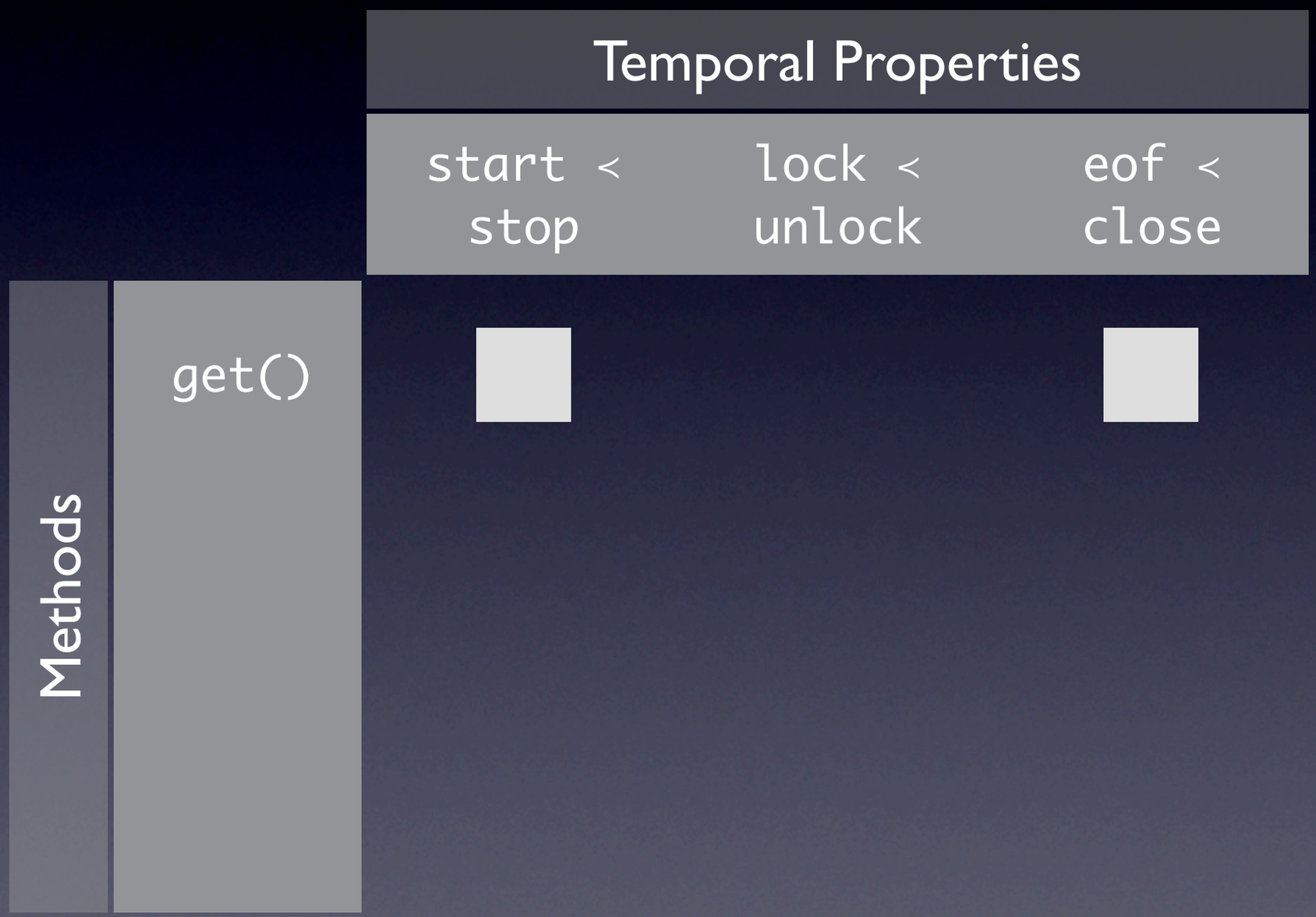

Program
Usage Models
iter.hasNext ()
iter.next ()
Temporal Properties
hasNext < next
hasNext < hasNext
next < hasNext
next < next
Patterns
hasNext < next
hasNext < hasNext
Anomalies
hasNext < next
hasNext < hasNext
hasNext < next
hasNext < hasNext

Methods vs. Properties

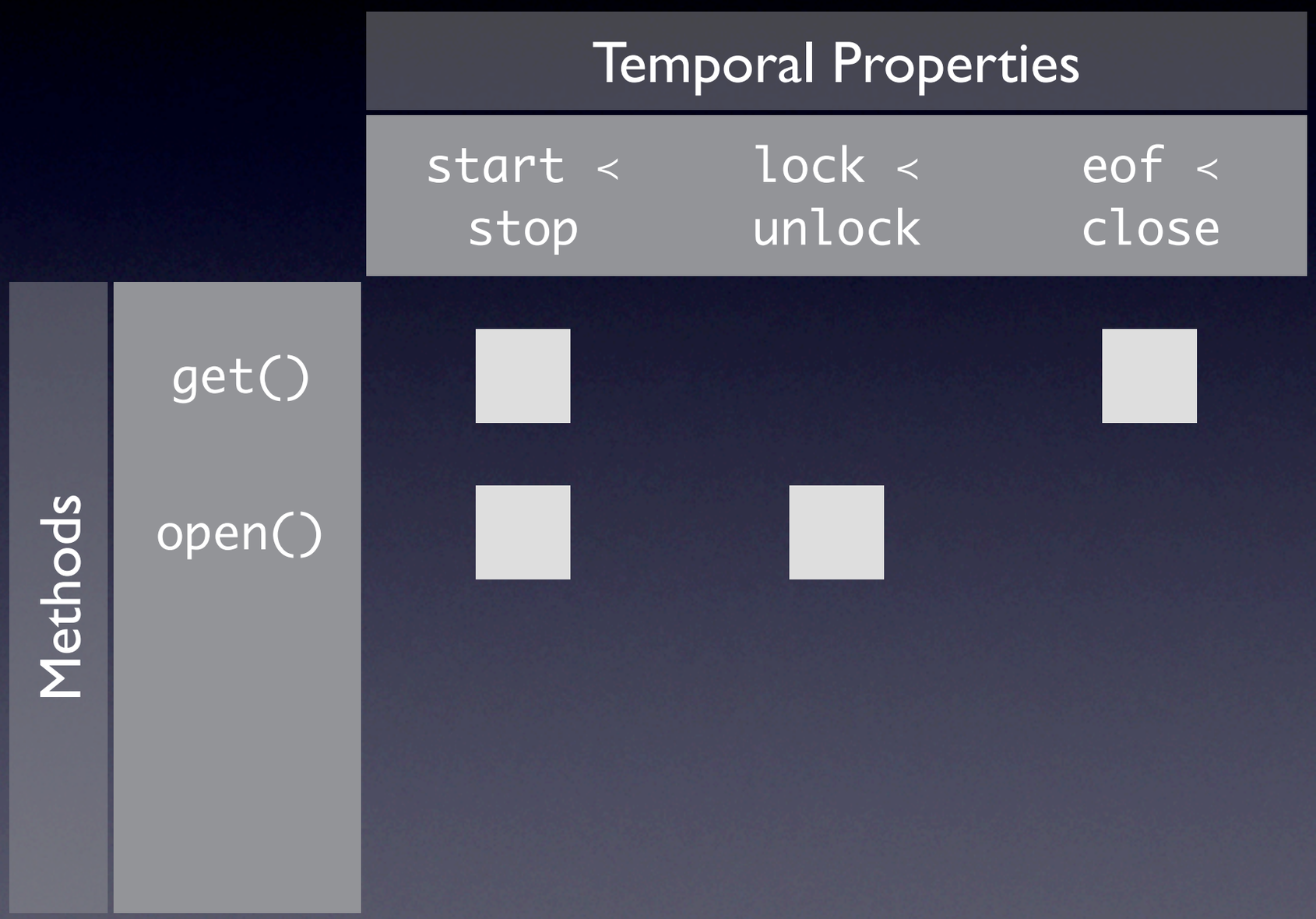
Temporal Properties		
`start <` `stop`	`lock <` `unlock`	`eof <` `close`

Methods

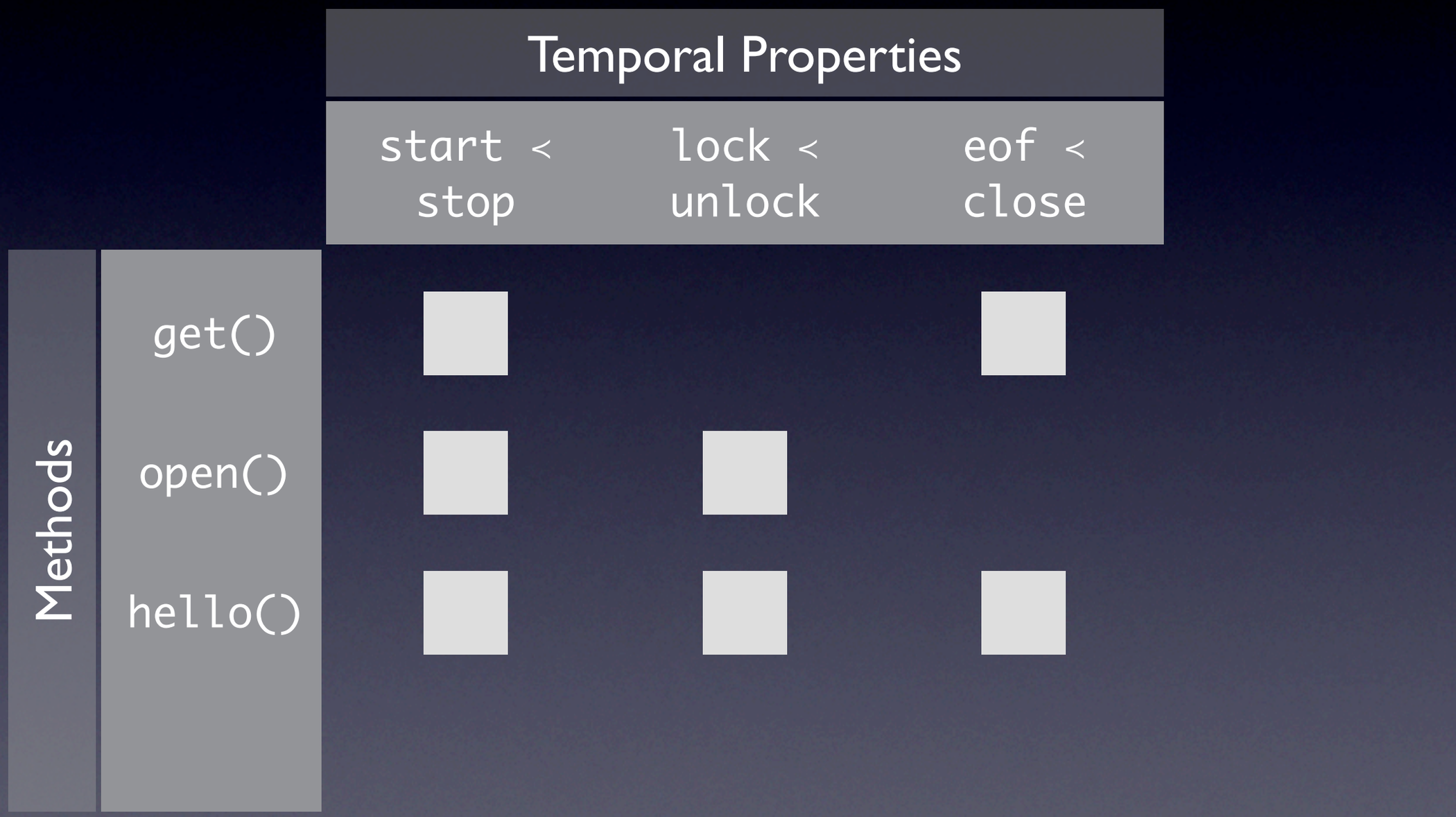
Methods vs. Properties



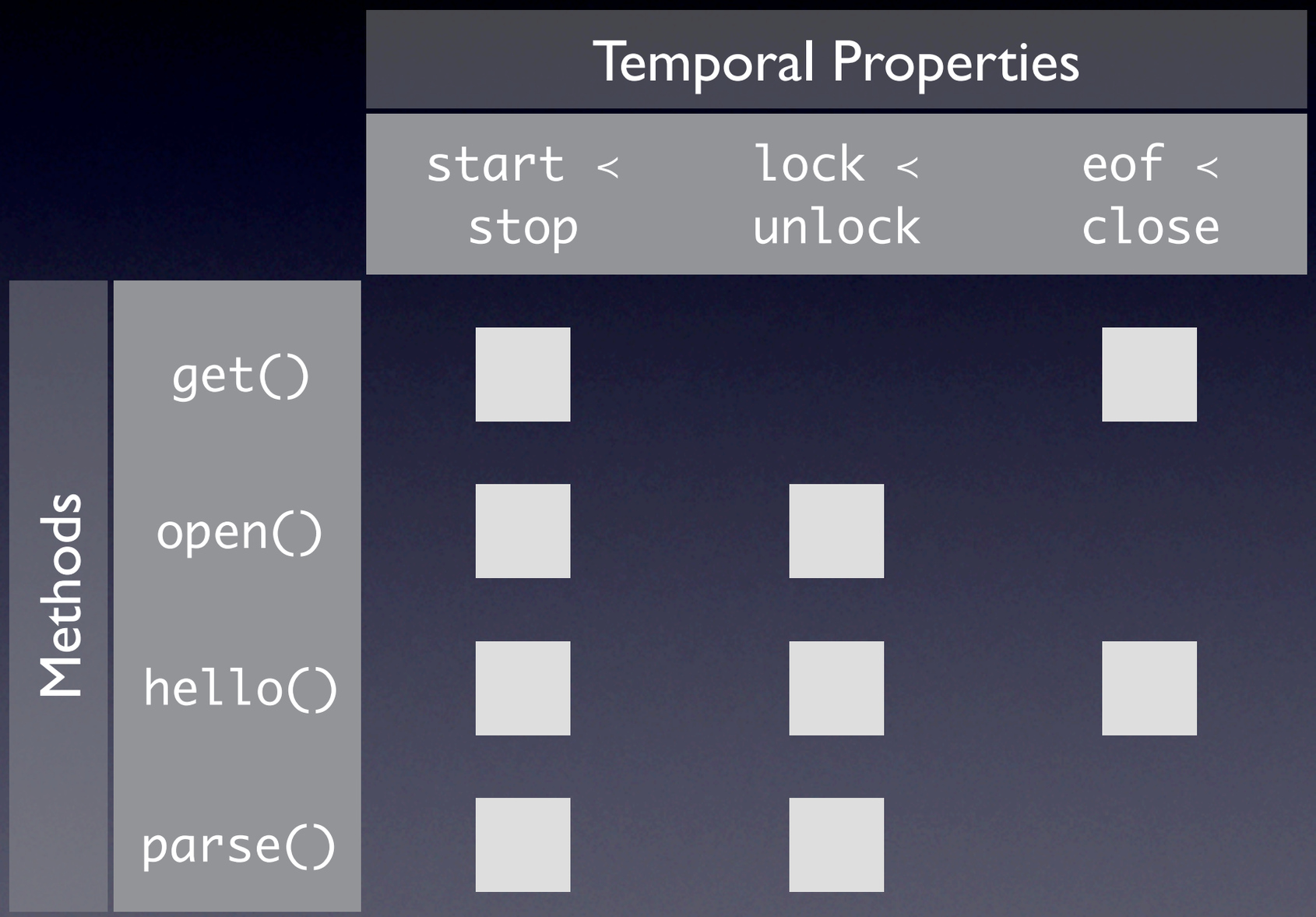
Methods vs. Properties



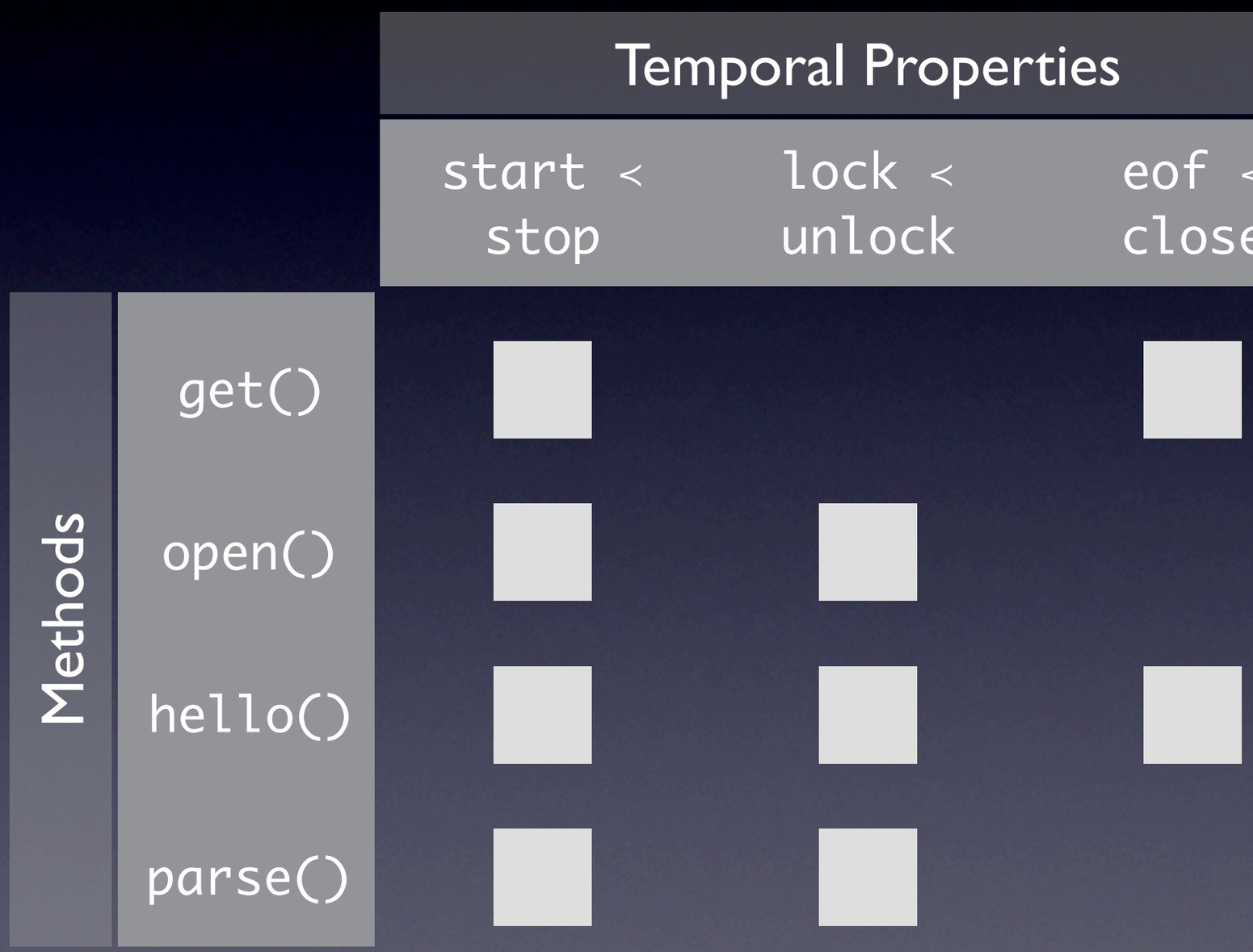
Methods vs. Properties



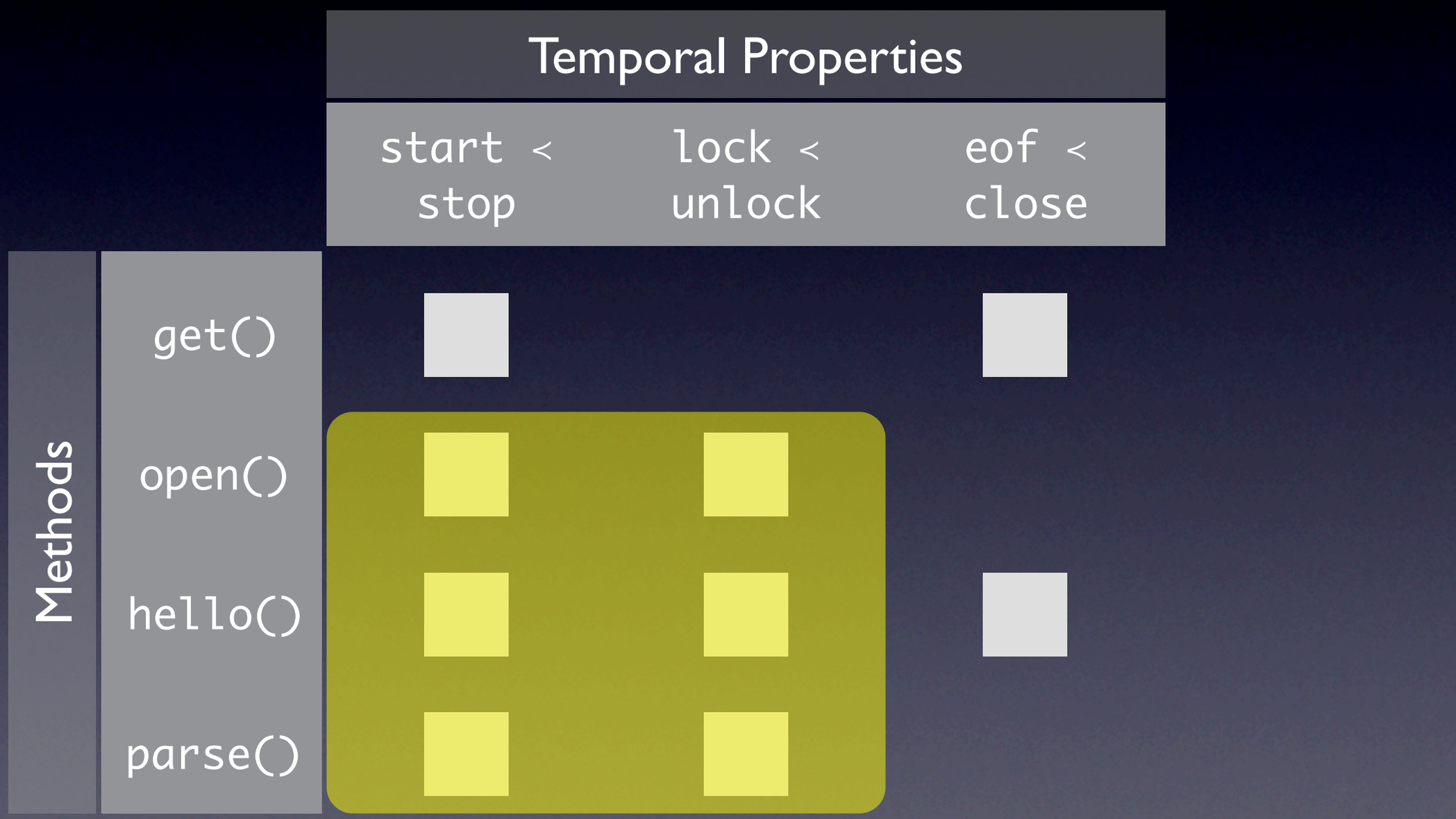
Methods vs. Properties



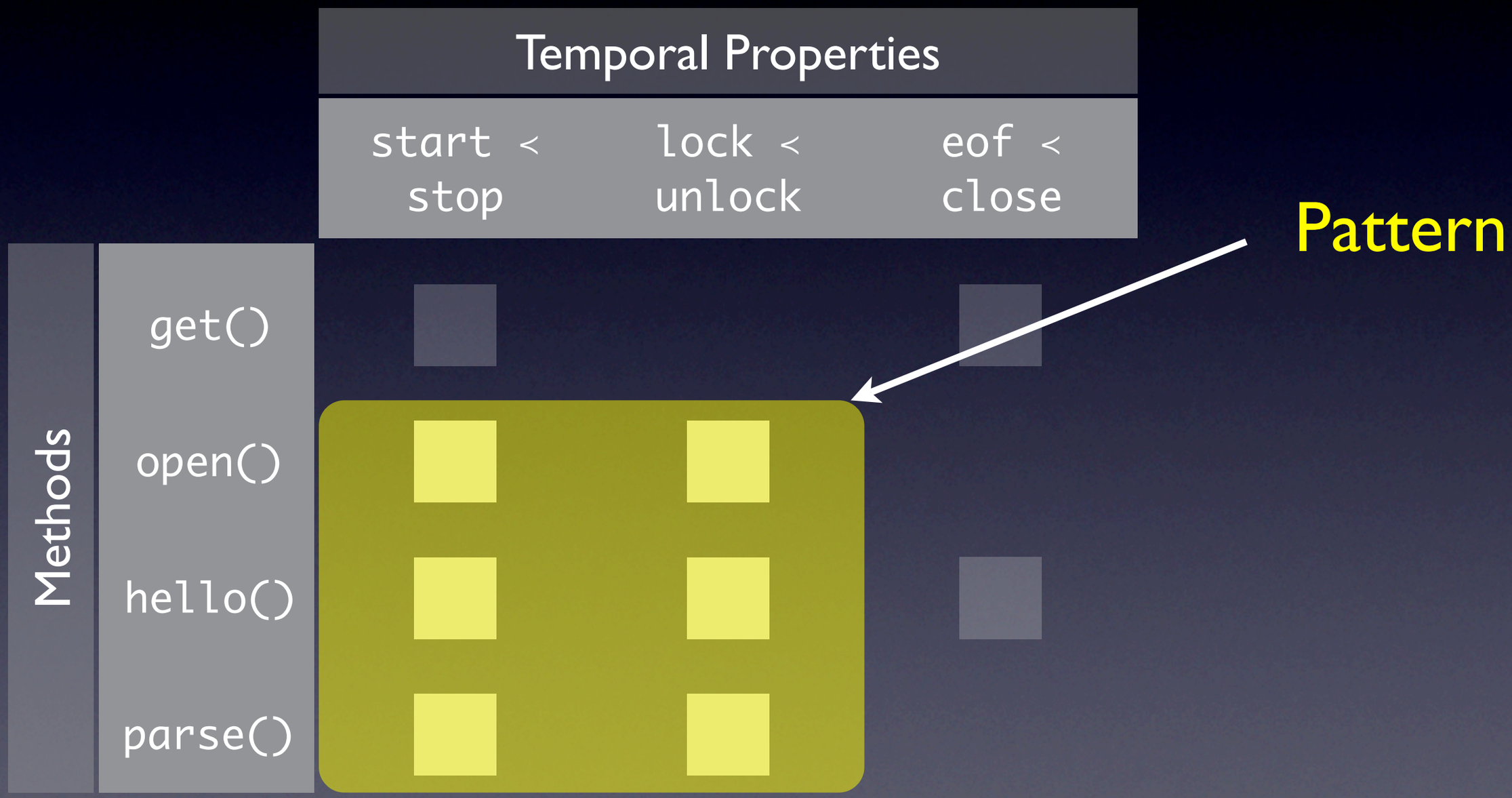
Methods vs. Properties



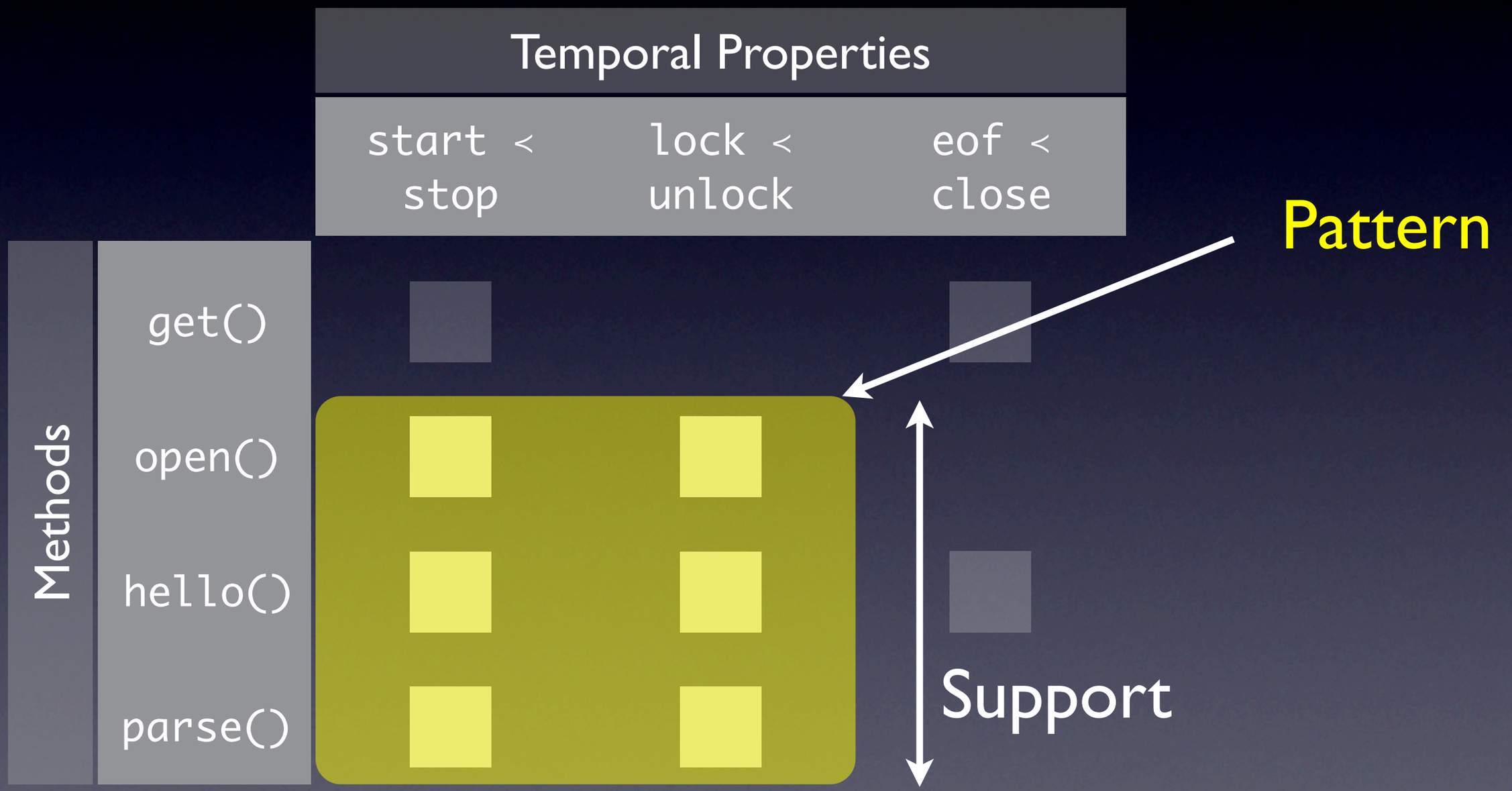
Methods vs. Properties



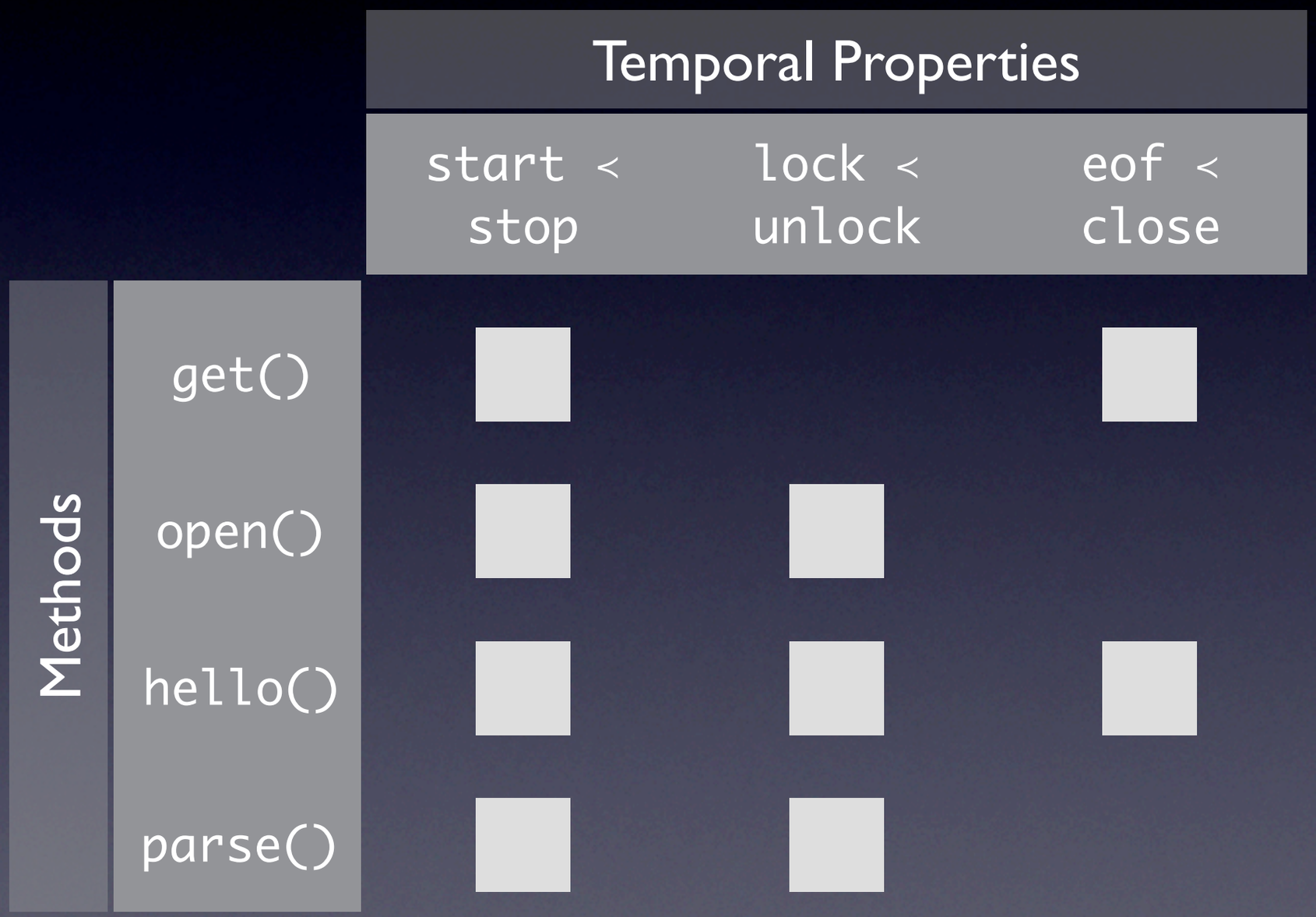
Methods vs. Properties



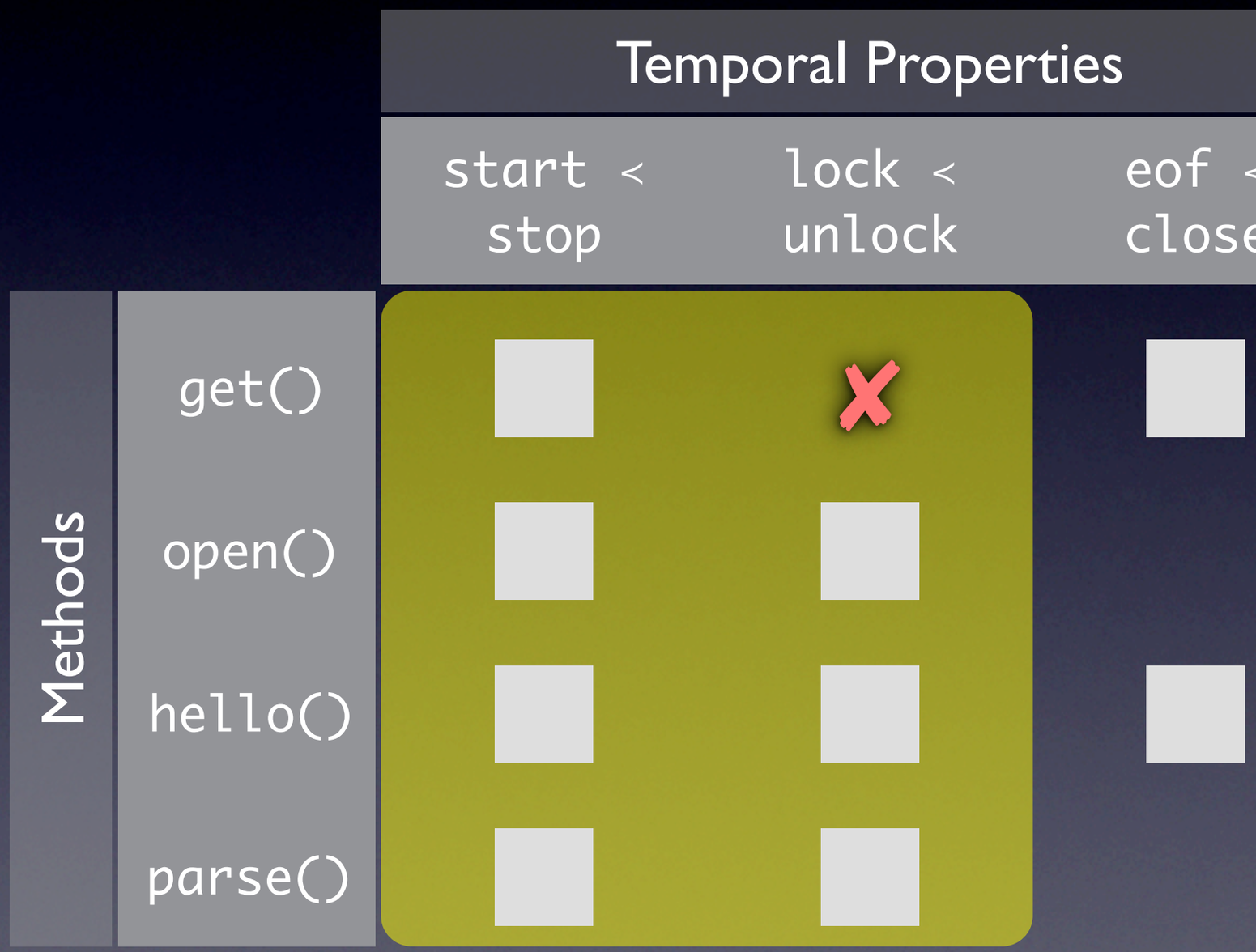
Methods vs. Properties



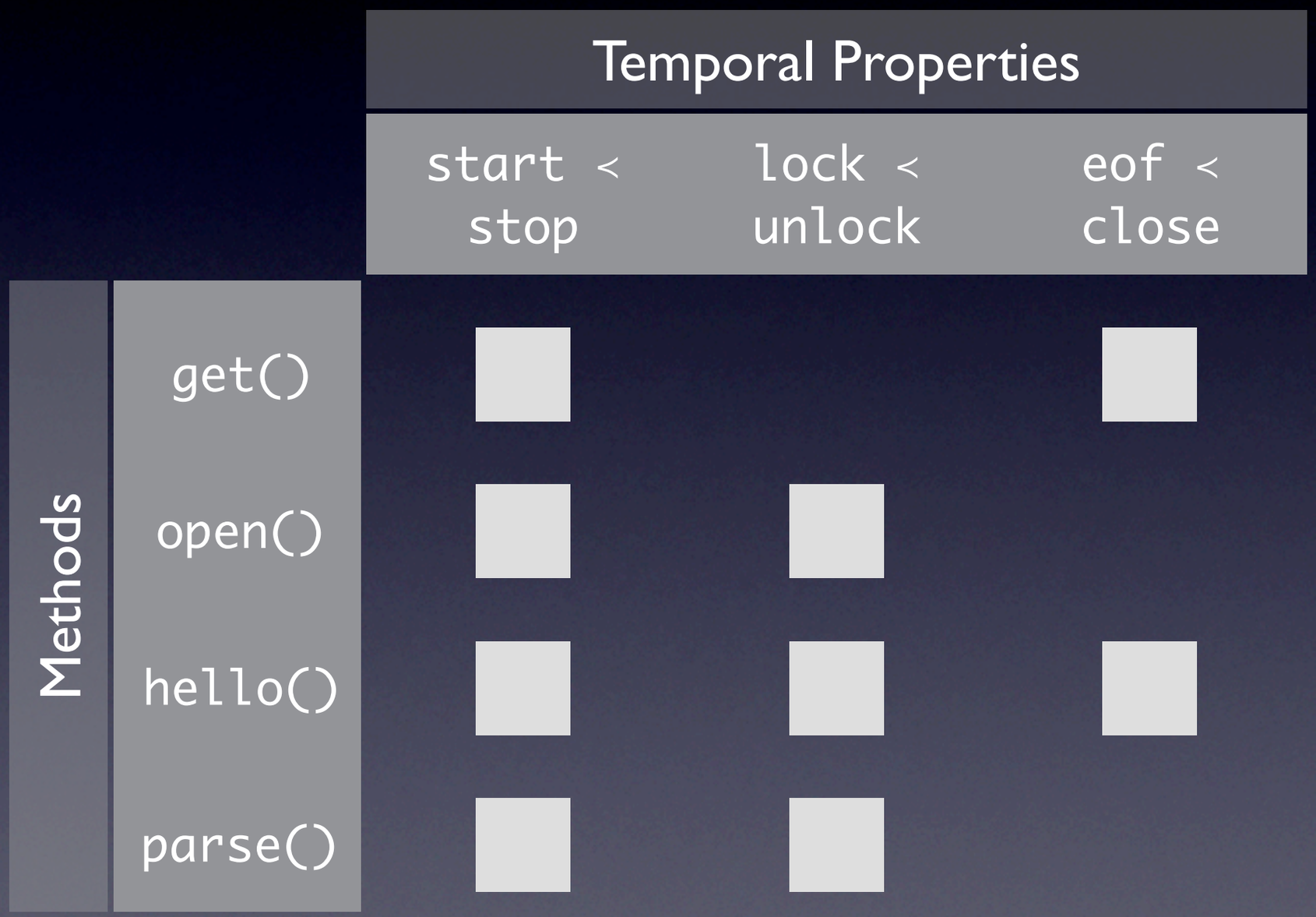
Discovering Anomalies



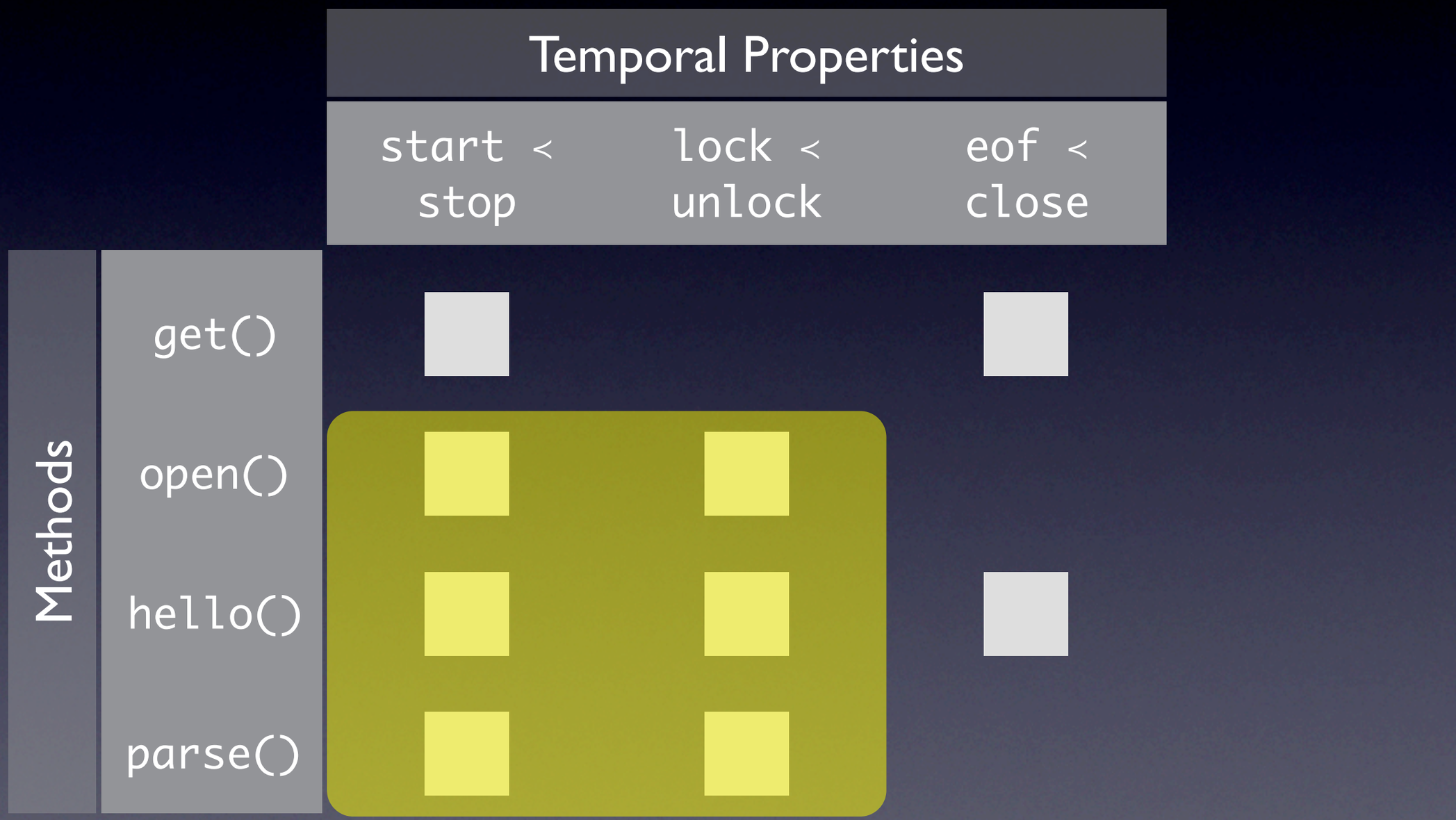
Discovering Anomalies



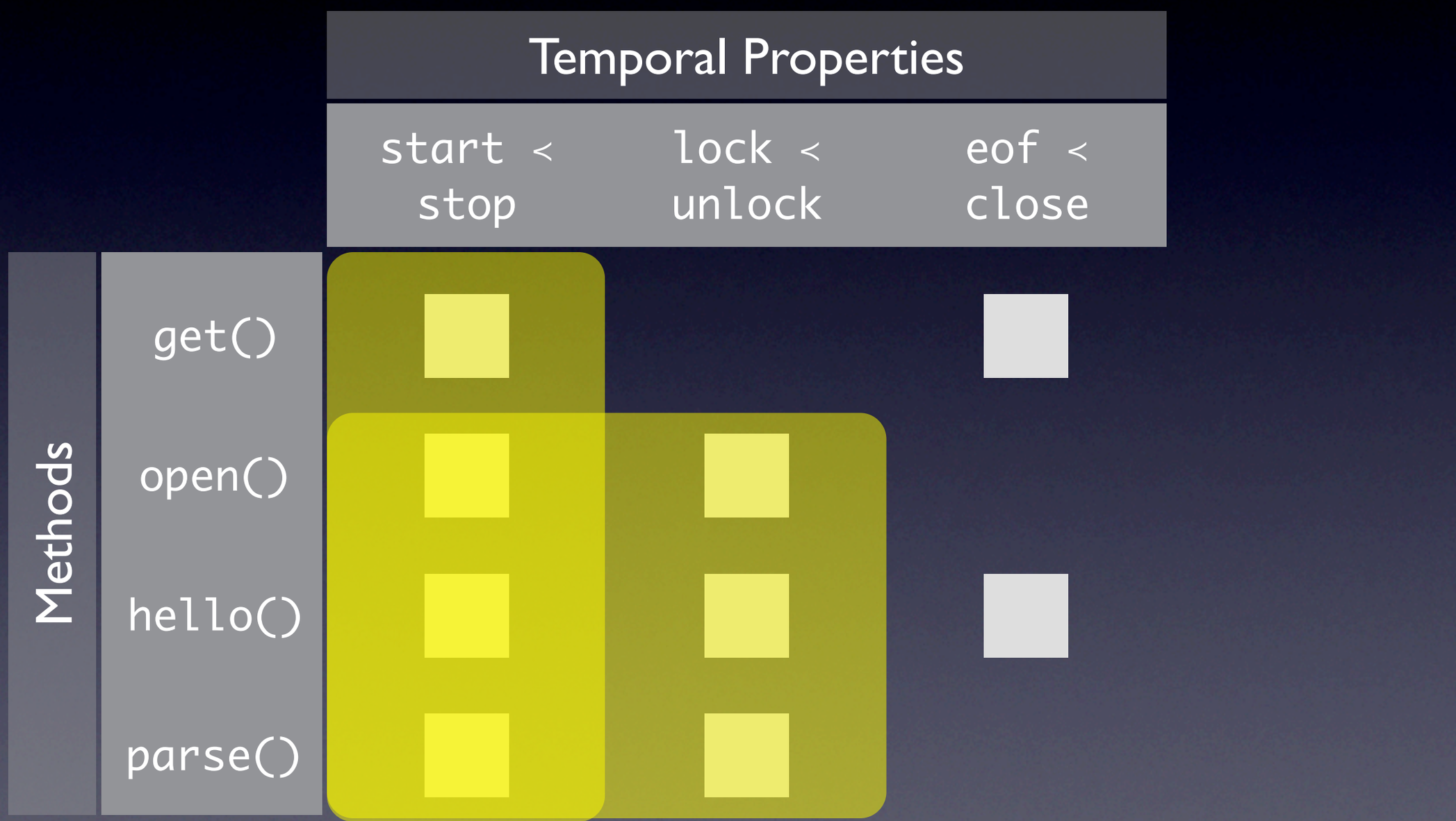
Discovering Anomalies



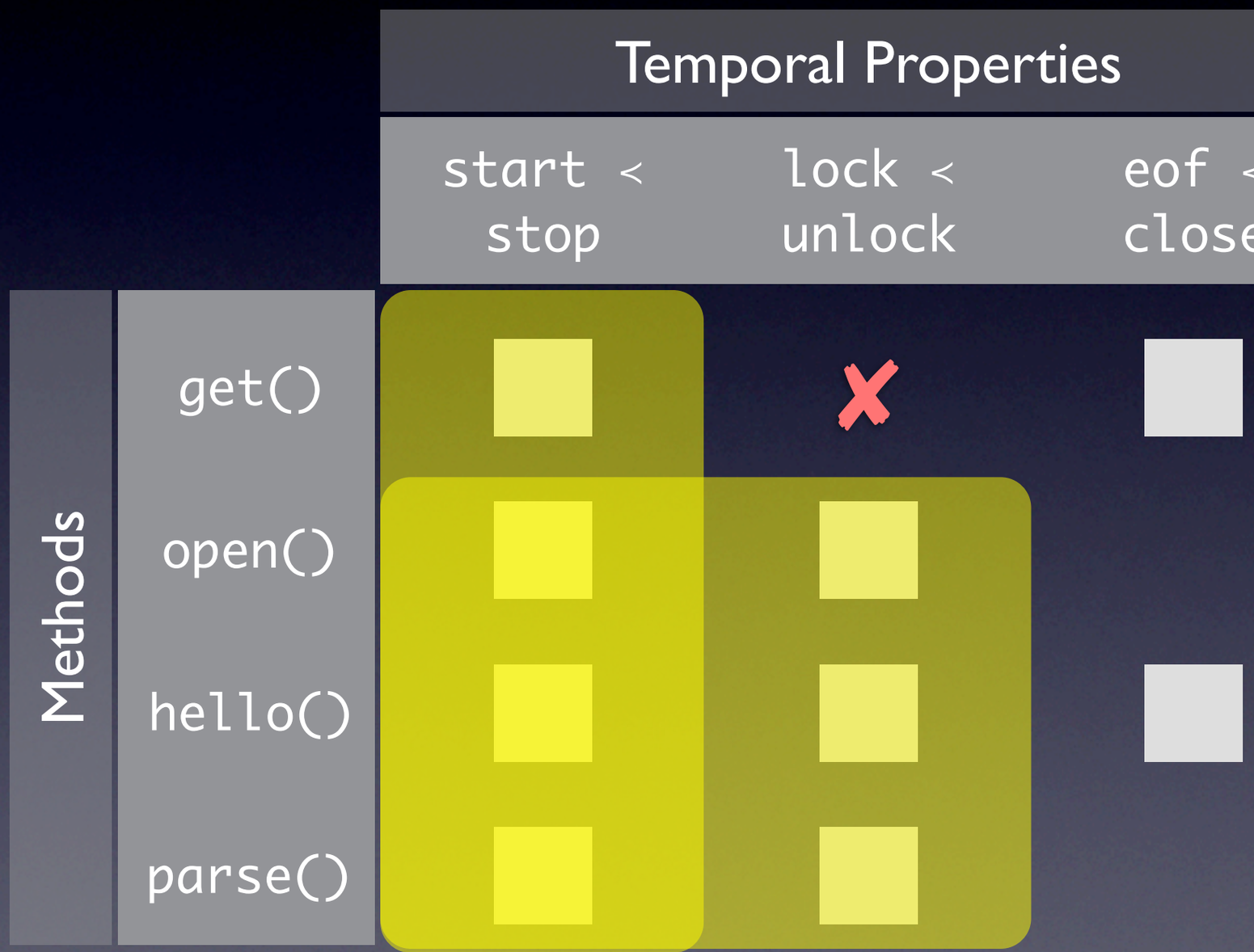
Discovering Anomalies



Discovering Anomalies



Discovering Anomalies



Case Study: AspectJ

Case Study: AspectJ

- 2,954 classes

Case Study: AspectJ

- 2,954 classes
- 36,045 methods

Case Study: AspectJ

- 2,954 classes
- 36,045 methods
- 1,154 methods with OP support ≥ 20

Case Study: AspectJ

- 2,954 classes
- 36,045 methods
- 1,154 methods with OP support ≥ 20
- 300 violations found in 8 minutes

Case Study: AspectJ

- 2,954 classes
- 36,045 methods
- 1,154 methods with OP support ≥ 20
- 300 violations found in 8 minutes
- Examined every single one

A Defect

```
for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
    }
}
```

A Defect

```
for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
    }
}
```

should be iter2

A Defect

```
for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
    }
}
```

should be iter2

bug.aj

```
@interface A {}

aspect Test {
  declare @field : @A int var* : @A;
  declare @field : int var* : @A;

  interface Subject {}

  public int Subject.vara;
  public int Subject.varb;
}

class X implements Test.Subject {}
```

Another Defect

```
public void visitNEWARRAY (NEWARRAY o) {
    byte t = o.getTypecode ();
    if (!((t == Constants.T_BOOLEAN) ||
          (t == Constants.T_CHAR) ||
           ...
          (t == Constants.T_LONG))) {
        constraintViolated (o, "(...) '+t+' (...)");
    }
}
```

Another Defect

```
public void visitNEWARRAY (NEWARRAY o) {
    byte t = o.getTypecode ();
    if (!((t == Constants.T_BOOLEAN) ||
          (t == Constants.T_CHAR) ||
           ...
          (t == Constants.T_LONG))) {
        constraintViolated (o, "(...)'+t+'(...)");
    }
}
```

should be double quotes

A False Positive

```
Name internalNewName (String[] identifiers)
    ...
    for (int i = 1; i < count; i++) {
        SimpleName name = new SimpleName(this);
        name.internalSetIdentifier(identifiers[i]);
        ...
    }
    ...
}
```

A False Positive

```
Name internalNewName (String[] identifiers)
    ...
    for (int i = 1; i < count; i++) {
        SimpleName name = new SimpleName(this);
        name.internalSetIdentifier(identifiers[i]);
        ...
    }
    ...
}
```

should stay as is

A Code Smell

```
public String getRetentionPolicy ()
{
   ...
   for (Iterator it = ...; it.hasNext();)
   {
        ... = it.next();
        ...
        return retentionPolicy;
   }
   ...
}
```

A Code Smell

```
public String getRetentionPolicy ()
{
   ...
   for (Iterator it = ...; it.hasNext();)
   {
      ... = it.next();
      ...
      return retentionPolicy;
   }
   ...
}
```

should be fixed

AspectJ

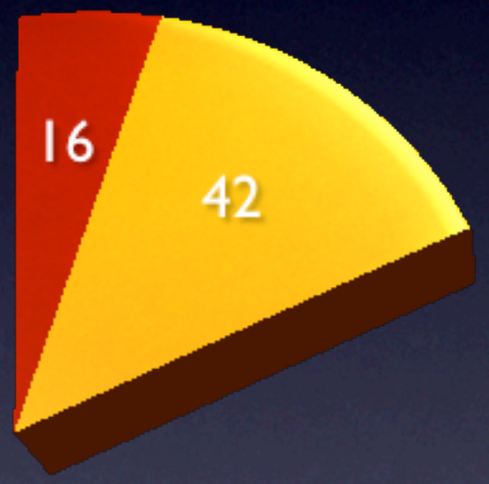
Defects Code smells False positives

AspectJ

Defects Code smells False positives

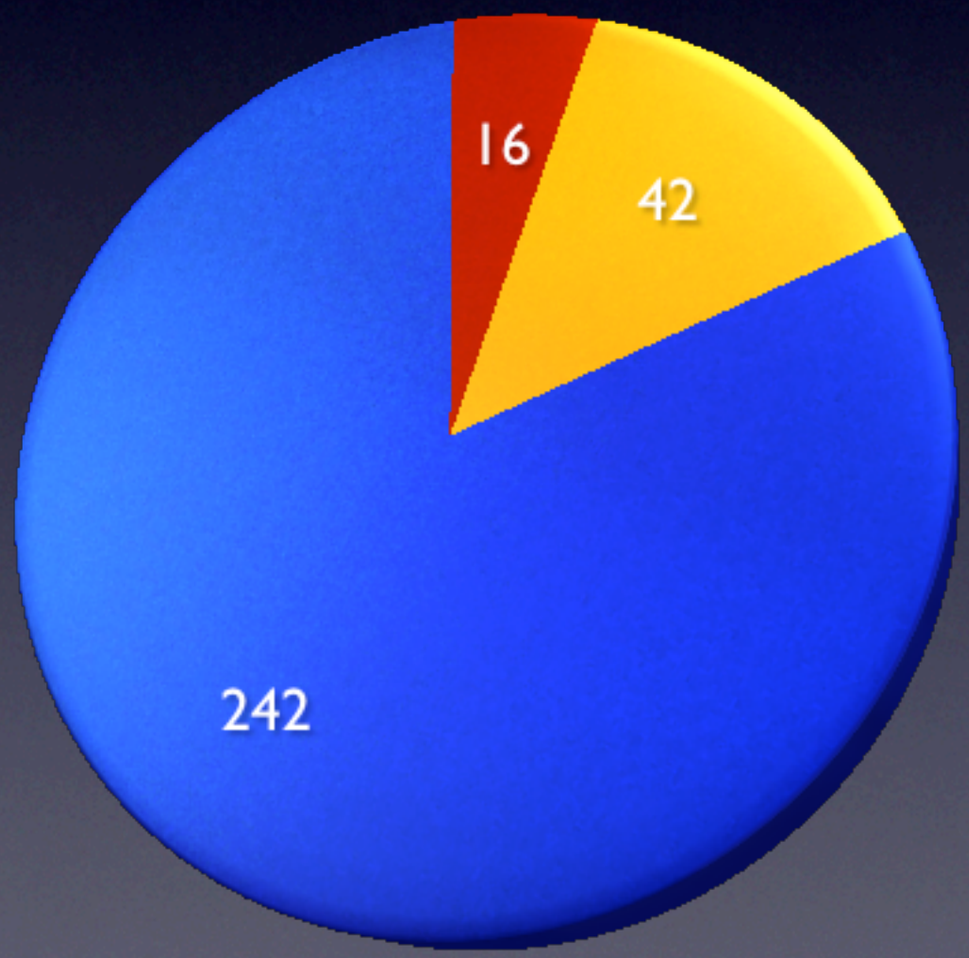
AspectJ

Defects Code smells False positives



AspectJ

Defects Code smells False positives



More Results

Program	# Violations		# Defects	# Code smells	# False positives	Efficiency
	Total	Investigated				
Act-Rbot 0.8.2	25	25	2	13	10	**60%**
Apache Tomcat 6.0.16	55	55	0	9	46	**16%**
ArgoUML 0.24	305	28	0	12	16	**43%**
AspectJ 1.5.3	300	300	16	42	242	**19%**
Azureus 2.5.0.0	315	85	1	26	58	**32%**
Columba 1.2	57	57	4	15	38	**33%**
jEdit 4.2	11	11	0	4	7	**36%**
	1,068	**562**	**23**	**121**	**417**	**26%**

Future Work

Future Work

- Procedural languages
  leveraging appropriate static analysis frameworks

Future Work

- Procedural languages
  leveraging appropriate static analysis frameworks
- Interprocedural analysis
  but does this make sense for learning usage?

Future Work

- Procedural languages
  leveraging appropriate static analysis frameworks
- Interprocedural analysis
  but does this make sense for learning usage?
- Ranking violations
  in particular in presence of low support

Future Work

- Procedural languages
  leveraging appropriate static analysis frameworks
- Interprocedural analysis
  but does this make sense for learning usage?
- Ranking violations
  in particular in presence of low support
- Early programmer support
  in terms of recommendations and documentation

OP-Miner

OP-Miner

- ★ OP-Miner learns *operational preconditions*
  i.e., how to typically construct arguments

OP-Miner

- ★ OP-Miner learns *operational preconditions*
  i.e., how to typically construct arguments
- ★ Learns from normal argument usage
  for specific projects or across projects

OP-Miner

- ★ OP-Miner learns *operational preconditions*
  i.e., how to typically construct arguments
- ★ Learns from normal argument usage
  for specific projects or across projects
- ★ Fully automatic

OP-Miner

- ★ OP-Miner learns *operational preconditions*
  i.e., how to typically construct arguments
- ★ Learns from normal argument usage
  for specific projects or across projects
- ★ Fully automatic
- ★ Found dozens of verified defects