Object Usage Models

Andrzej Wasylkowski
Saarland University

A bug in AspectJ

Bugzilla Bug 165631 It is possible to mark a class as implementing multiple parametrizations of a generic type

Bug List: (209 of 212) First Last Prev Next Show last search results Search page Enter new bug

[Tools] Bug#: 165631
Product: AspectJ
Component: Compiler
Status: NEW
Resolution:
Assigned To: aspectj inbox <aspectj-inbox@eclipse.org>

Hardware: Macintosh
OS: MacOS X
Version: DEVELOPMENT
Priority: P3
Severity: normal
Target Milestone: ---

Reporter: Andrzej Wasylkowski
Add CC:
CC:

QA Contact:
URL:
Summary: It is possible to mark a class as implementing multiple parametrizations of a g
Status Whiteboard:
Keywords:

A bug in AspectJ

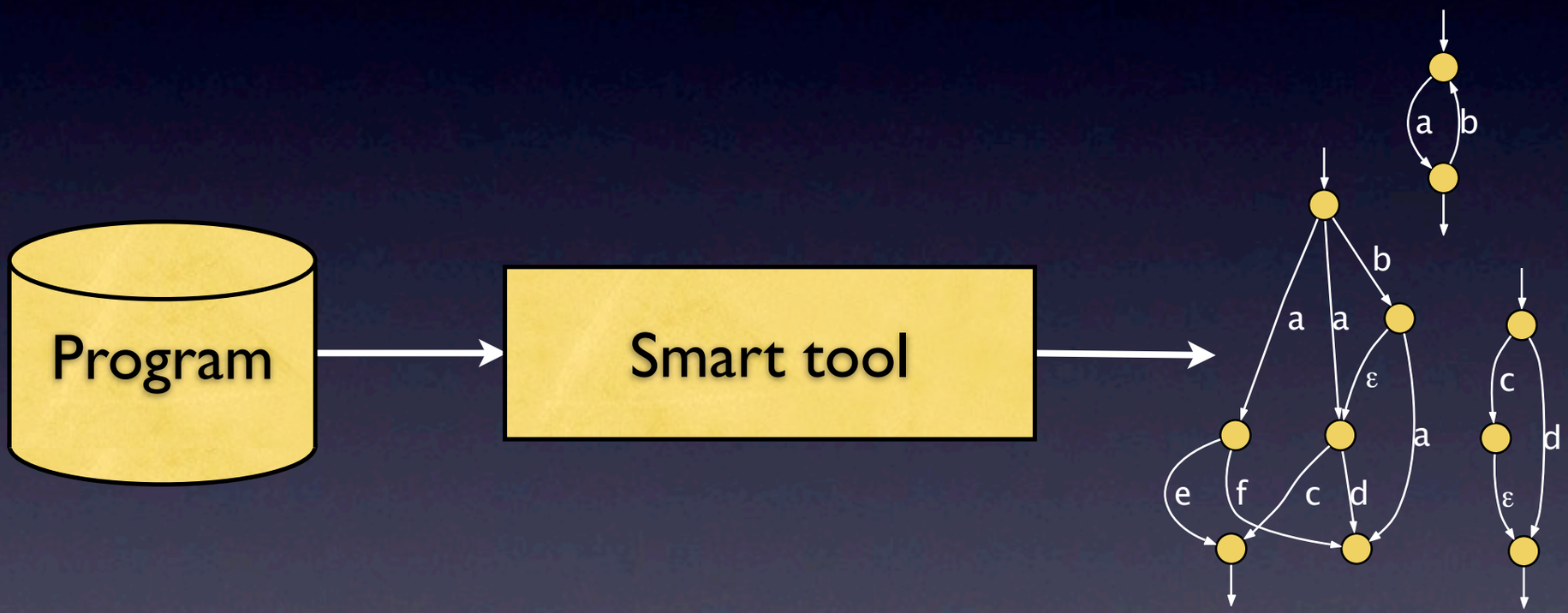
```
/**
 * This method looks through the type hierarchy for some target type - it is attempting to
 * find an existing parameterization that clashes with the new parent that the user
 * wants to apply to the type.  If it finds an existing parameterization that matches the
 * new one, it silently completes, if it finds one that clashes (e.g. a type already has
 * A<String> when the user wants to add A<Number>) then it will produce an error.
 *
 * It uses recursion and exits recursion on hitting 'jlObject'
  *
 * Related bugzilla entries: pr110788
 */
private boolean verifyNoInheritedAlternateParameterization(ResolvedType typeToVerify,ResolvedType newParent,World world) {


        if (typeToVerify.equals(ResolvedType.OBJECT)) return true;

        ResolvedType newParentGenericType = newParent.getGenericType();
        Iterator iter = typeToVerify.getDirectSupertypes();
        while (iter.hasNext()) {
                ResolvedType supertype = (ResolvedType)iter.next();
                if ( ((supertype.isRawType() && newParent.isParameterizedType()) ||
                      (supertype.isParameterizedType() && newParent.isRawType())) && newParentGenericType.equals(supertype.getGenericType())) {
                        // new parent is a parameterized type, but this is a raw type
                        world.getMessageHandler().handleMessage(new Message(
                                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTIPLE_PARAMETERIZATIONS,newParent.getName(),typeToVerify.getName(),supertype.getName()),
                                getSourceLocation(), true, new ISourceLocation[]{typeToVerify.getSourceLocation()}));
                        return false;
                }
                if (supertype.isParameterizedType()) {
                        ResolvedType generictype = supertype.getGenericType();

                        // If the generic types are compatible but the parameterizations aren't then we have a problem
                        if (generictype.isAssignableFrom(newParentGenericType) &&
                                        !supertype.isAssignableFrom(newParent)) {
                                world.getMessageHandler().handleMessage(new Message(
                                                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTIPLE_PARAMETERIZATIONS,newParent.getName(),typeToVerify.getName(),supertype.getName()),
                                                getSourceLocation(), true, new ISourceLocation[]{typeToVerify.getSourceLocation()}));
                                return false;
                        }
                }
                return verifyNoInheritedAlternateParameterization(supertype,newParent,world);
        }
        return true;
}
```

A bug in AspectJ

```
private boolean verify… (…) {
    …
    Iterator iter = …;
    while (iter.hasNext()) {
        … = iter.next();
        …
        return verify… (…);
    }
    return true;
}
```

How can we find such a bug automatically?

The approach



Modeling objects' behavior using finite state automata

How to create models?

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

- Use method calls as transitions
- How to define states?

How to define states?

1. Anonymous states via grammar inference

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

How to define states?

1. Anonymous states via grammar inference

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

Execution

```
q = new
q.offer
q.poll
q.offer
q.poll
q.offer
…
q.poll
q.offer
```

How to define states?

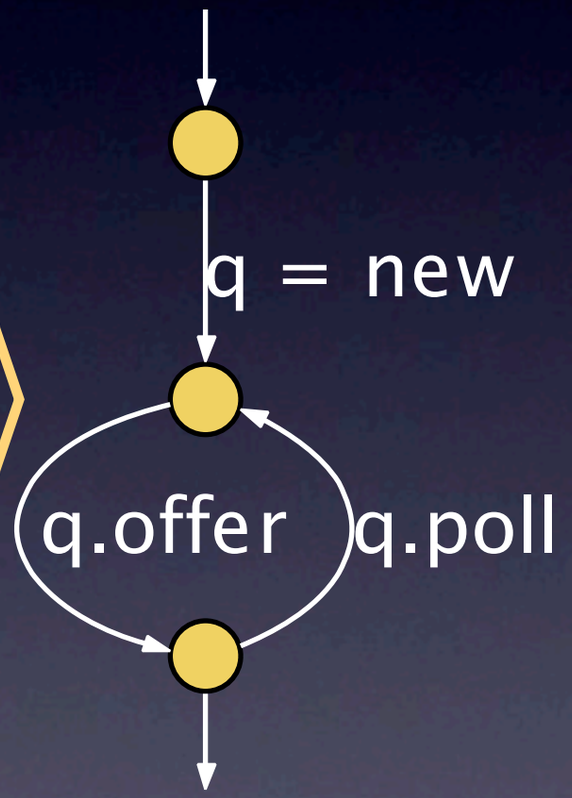
1. Anonymous states via grammar inference

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

Execution

```
q = new
q.offer
q.poll
q.offer
q.poll
q.offer
…
q.poll
q.offer
```

Grammar inference

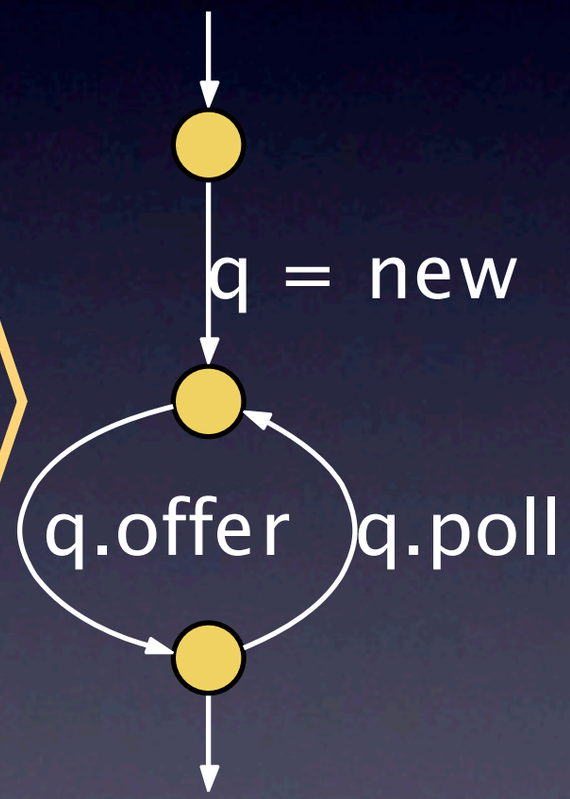


How to define states?

1. Anonymous states via grammar inference

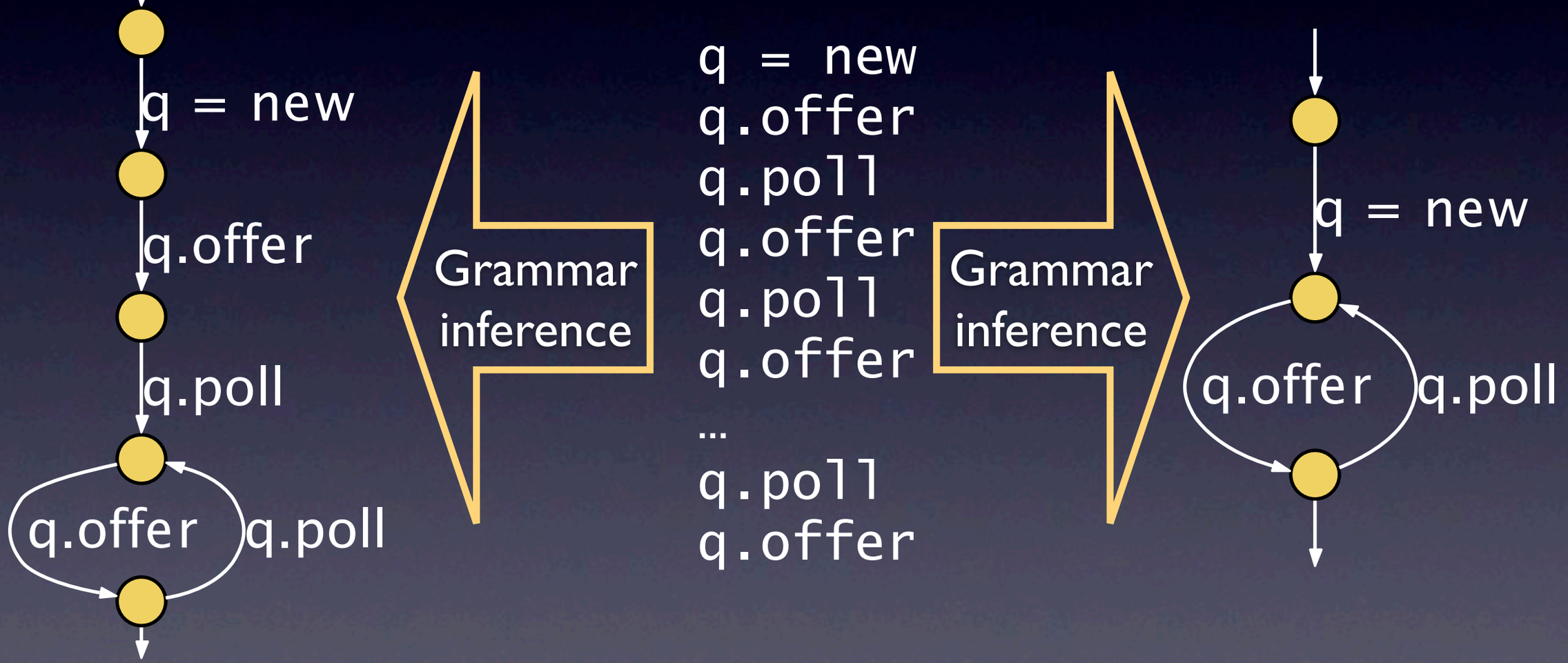
```
q = new
q.offer
q.poll
q.offer
q.poll
q.offer
…
q.poll
q.offer
```

Grammar inference



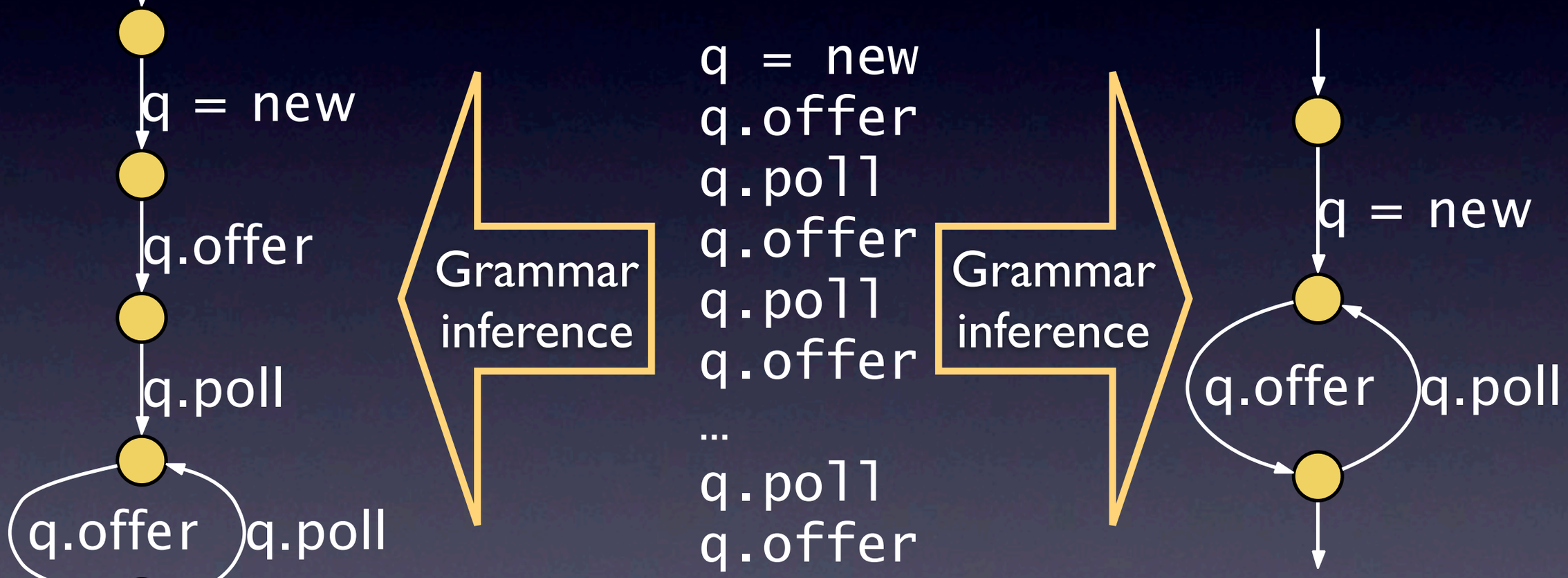
How to define states?

1. Anonymous states via grammar inference



How to define states?

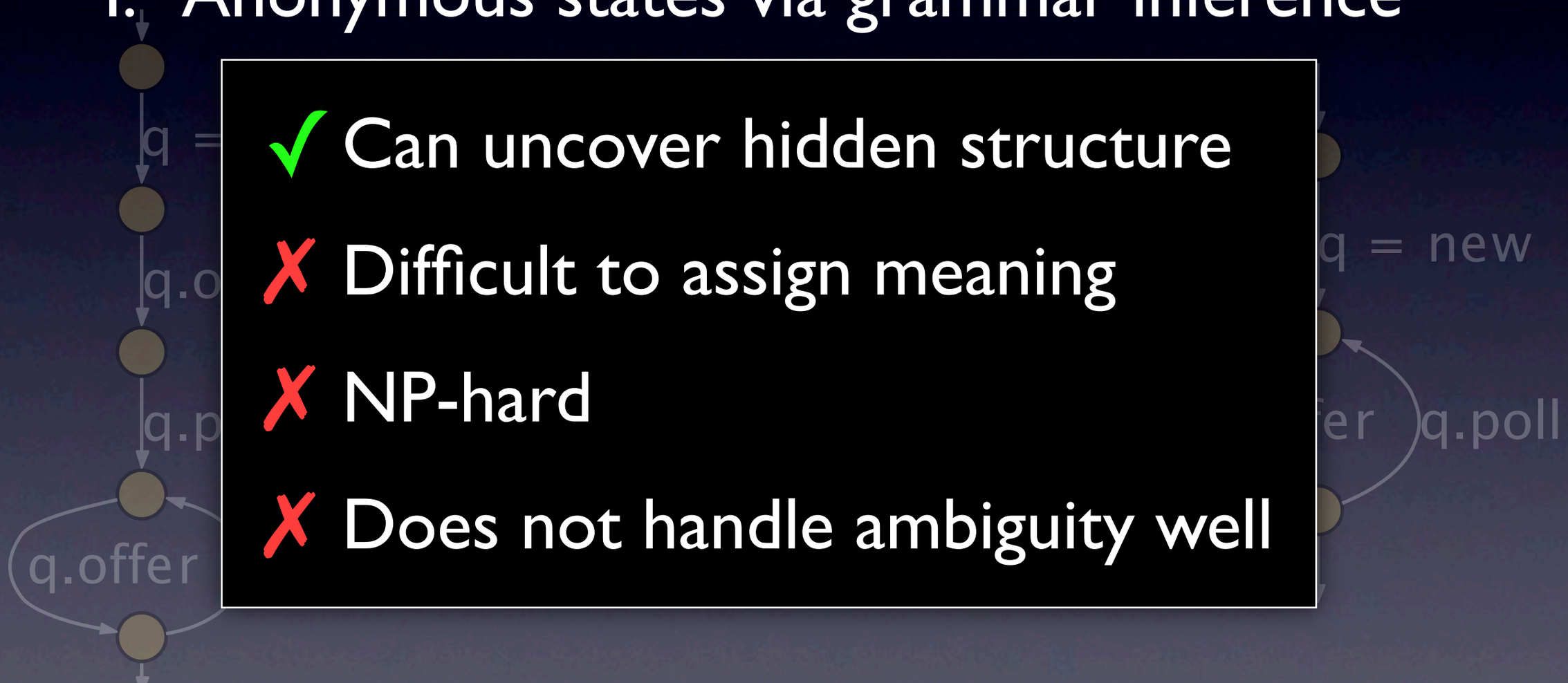
1. Anonymous states via grammar inference



2. Which model represents actual behavior?

How to define states?

1. Anonymous states via grammar inference



✓ Can uncover hidden structure

✗ Difficult to assign meaning

✗ NP-hard

✗ Does not handle ambiguity well

Which model represents actual behavior?

How to define states?

II. Based on object's state

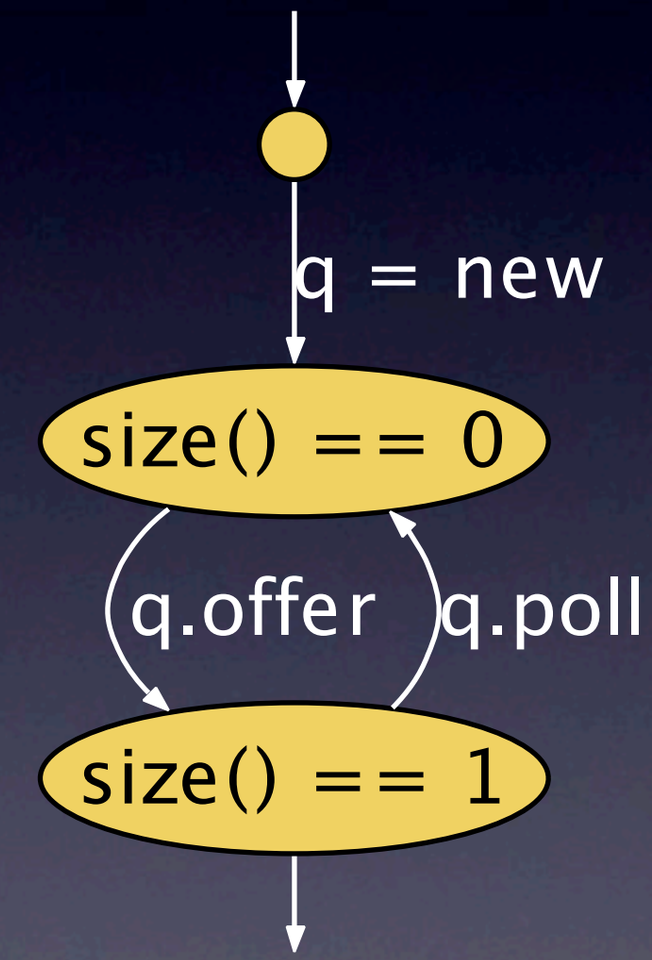
```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

How to define states?

II. Based on object's state

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

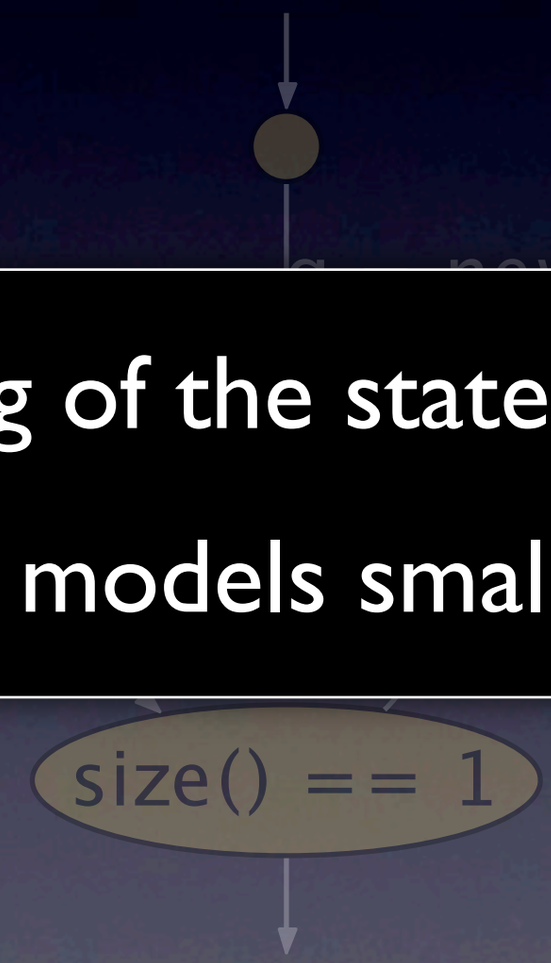
Execution



How to define states?

II. Based on object's state

```
q = new Queue ();
q.offer (...);
}
```



✓ Expresses the true meaning of the state

✗ Needs abstraction to keep models small

How to define states?

III. Based on the way the object is used

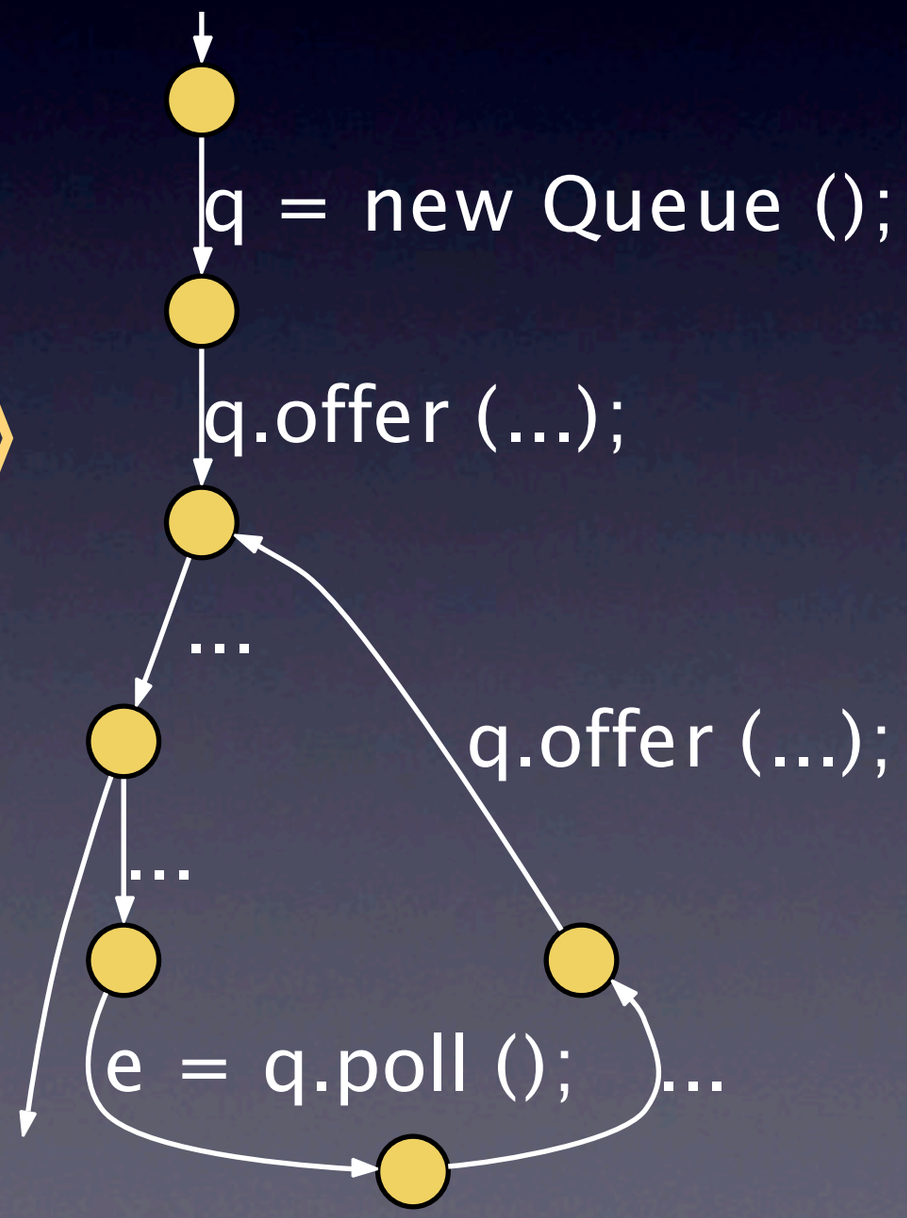
```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

How to define states?

III. Based on the way the object is used

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

Static analysis

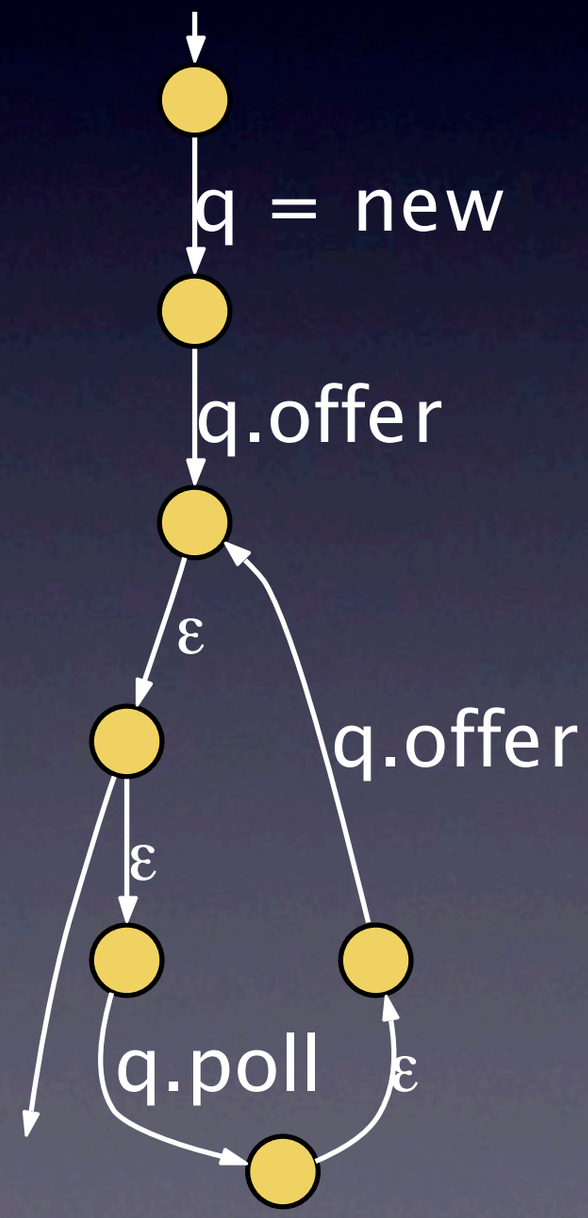


How to define states?

III. Based on the way the object is used

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```

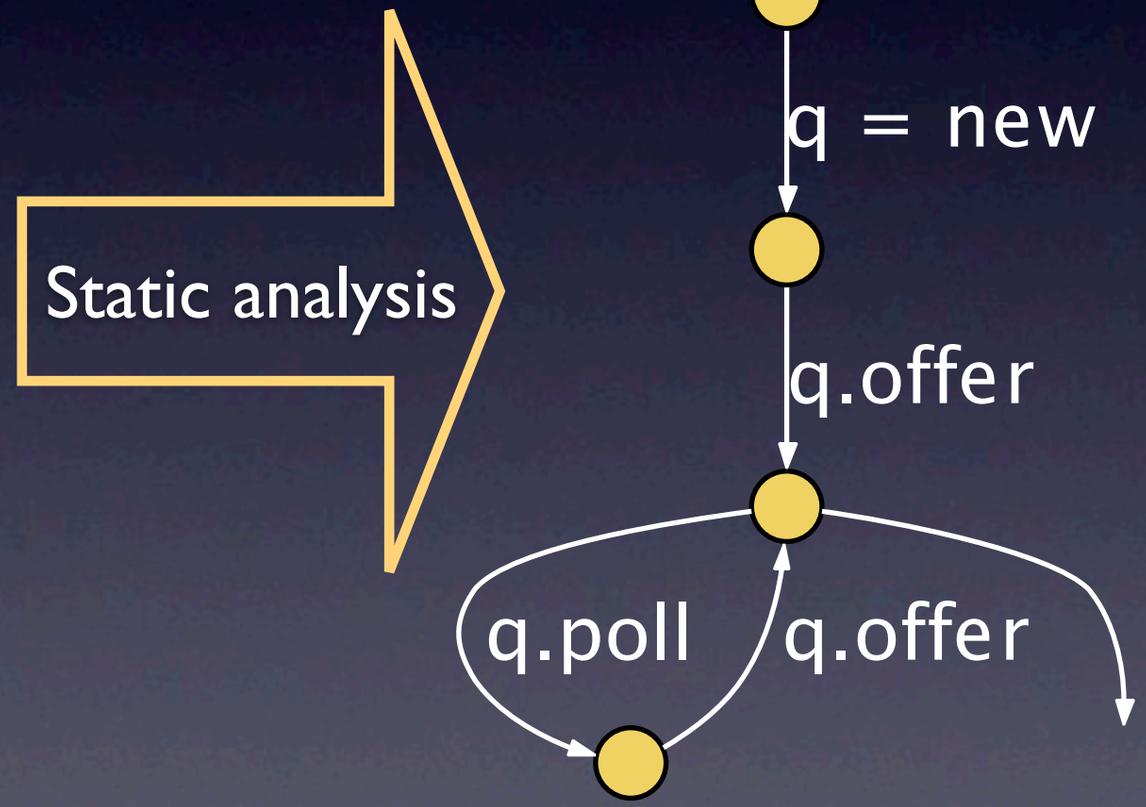
Static analysis



How to define states?

III. Based on the way the object is used

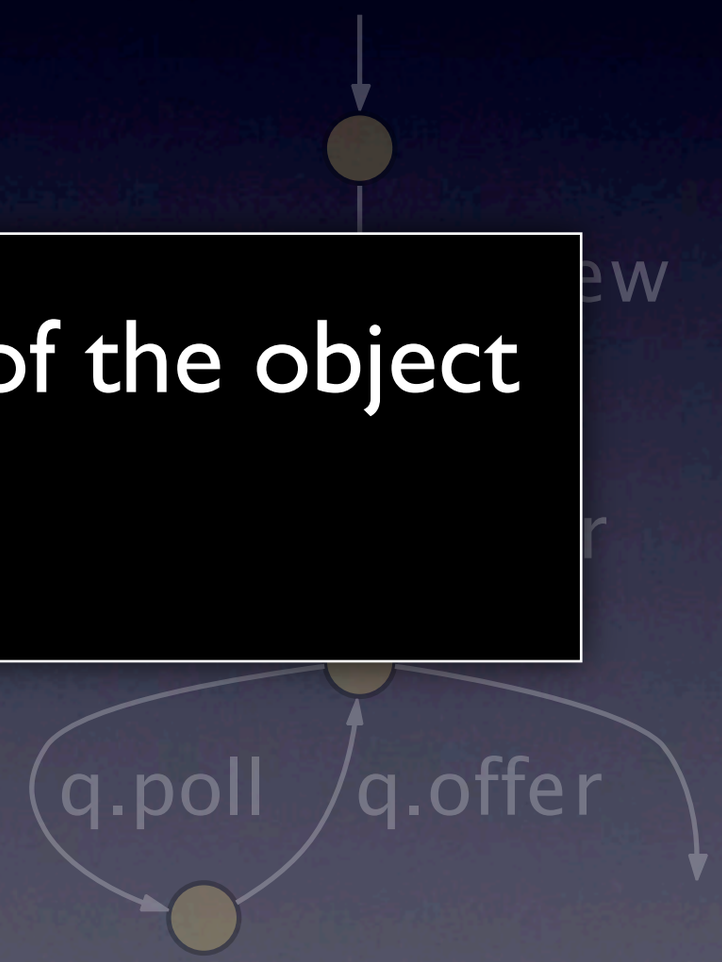
```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  q.offer (…);
}
```



How to define states?

III. Based on the way the object is used

```
q = new Queue ();
q.offer (…);
whi
  …
  e
  …
  q.offer (…);
}
```



✓ User-centered view of the object

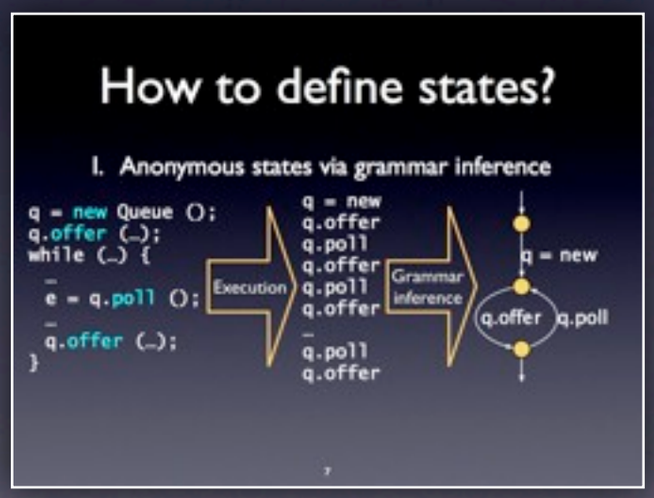
✗ May be too specific

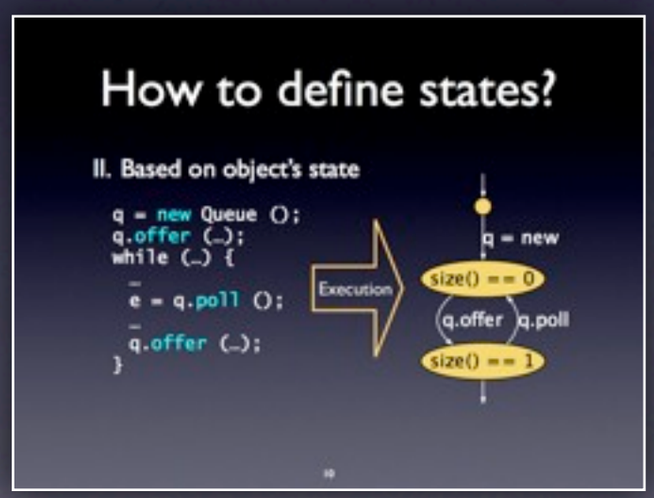
How to define states?

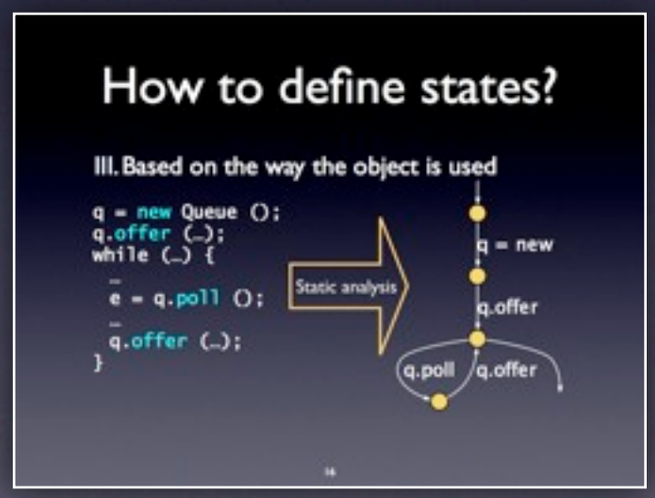
I. Anonymous states via grammar inference

II. Based on object's state

III. Based on the way the object is used





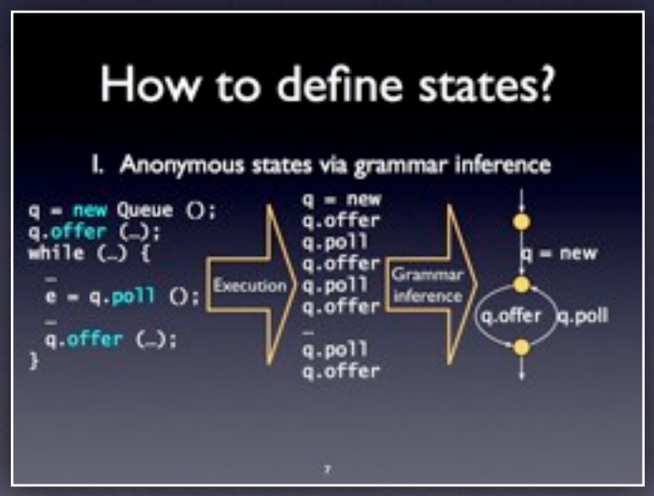


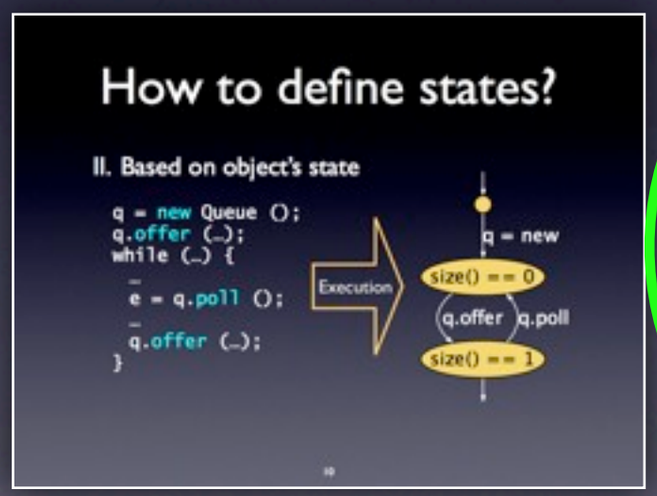
How to define states?

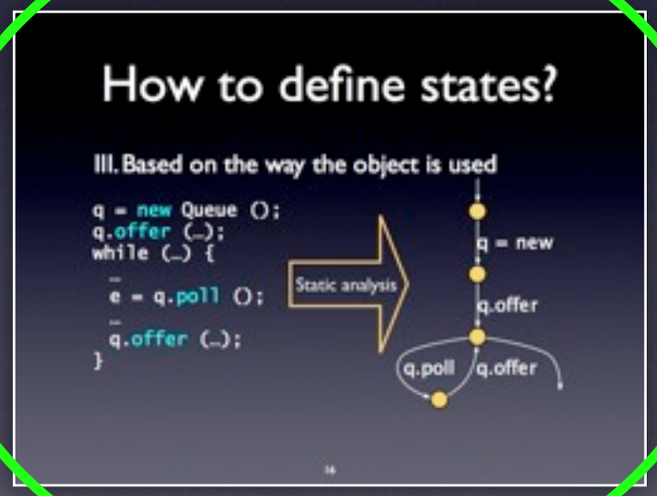
I. Anonymous states via grammar inference

II. Based on object's state

III. Based on the way the object is used







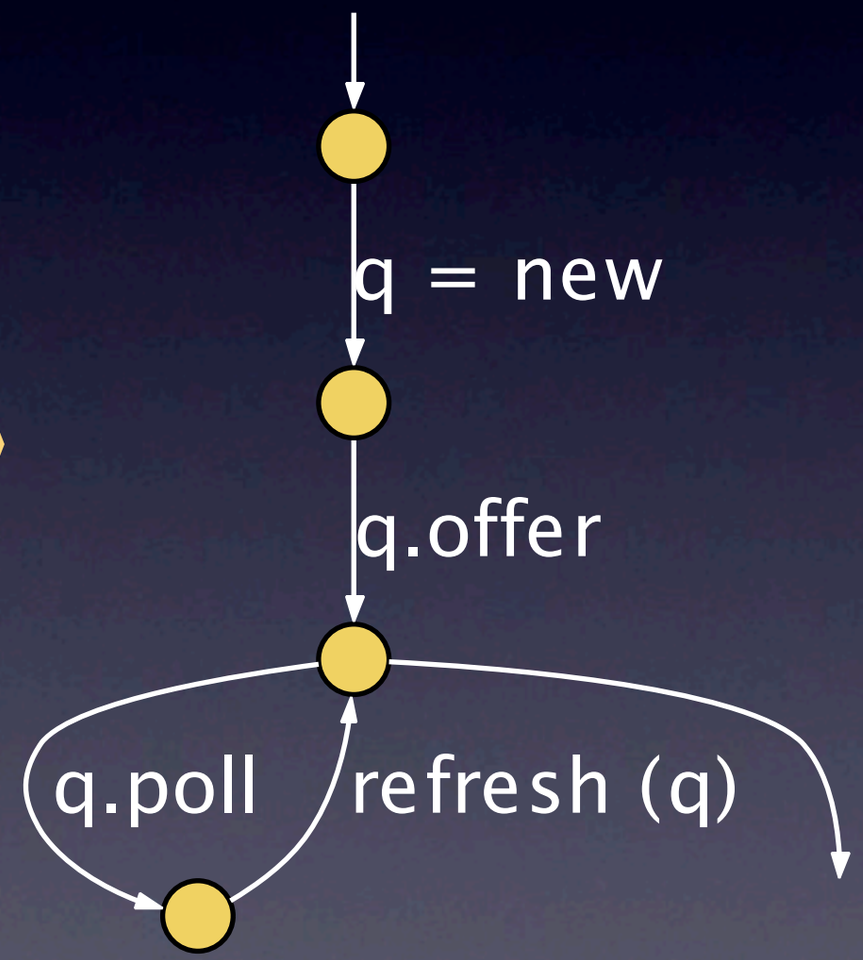
External method calls

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  refresh (q);
}
```

External method calls

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  refresh (q);
}
```

Static analysis

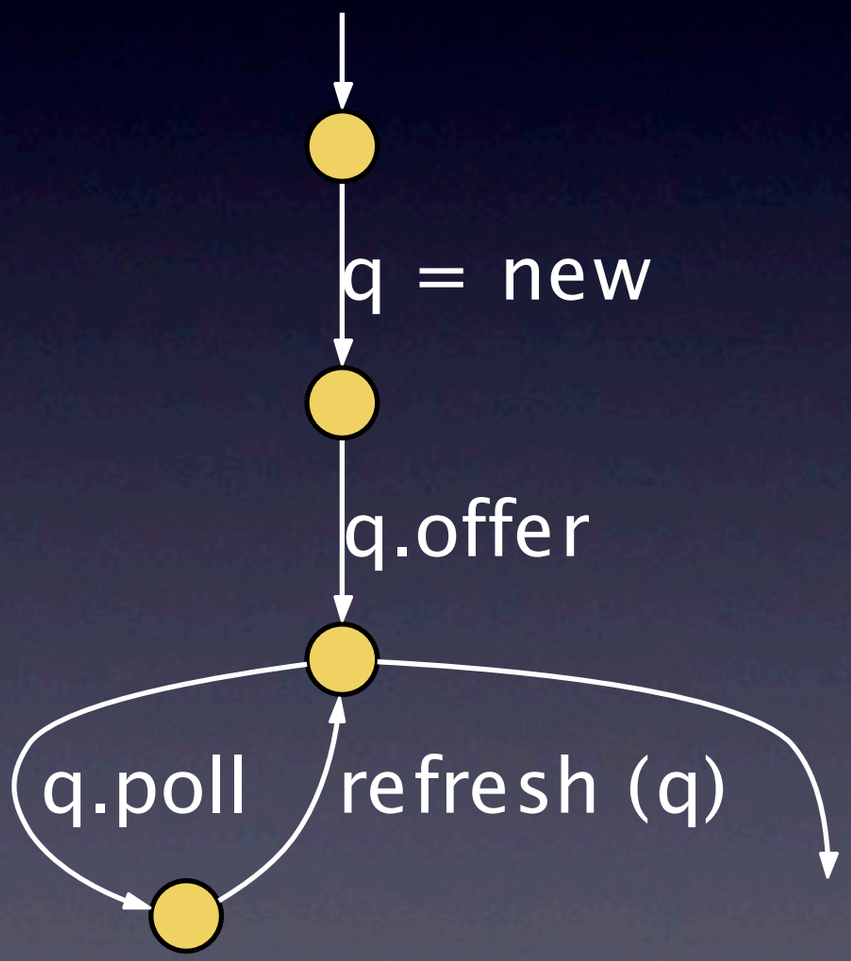


Inlining models

```
q = new Queue ();
q.offer (…);
while (…) {
  …
  e = q.poll ();
  …
  refresh (q);
}
```

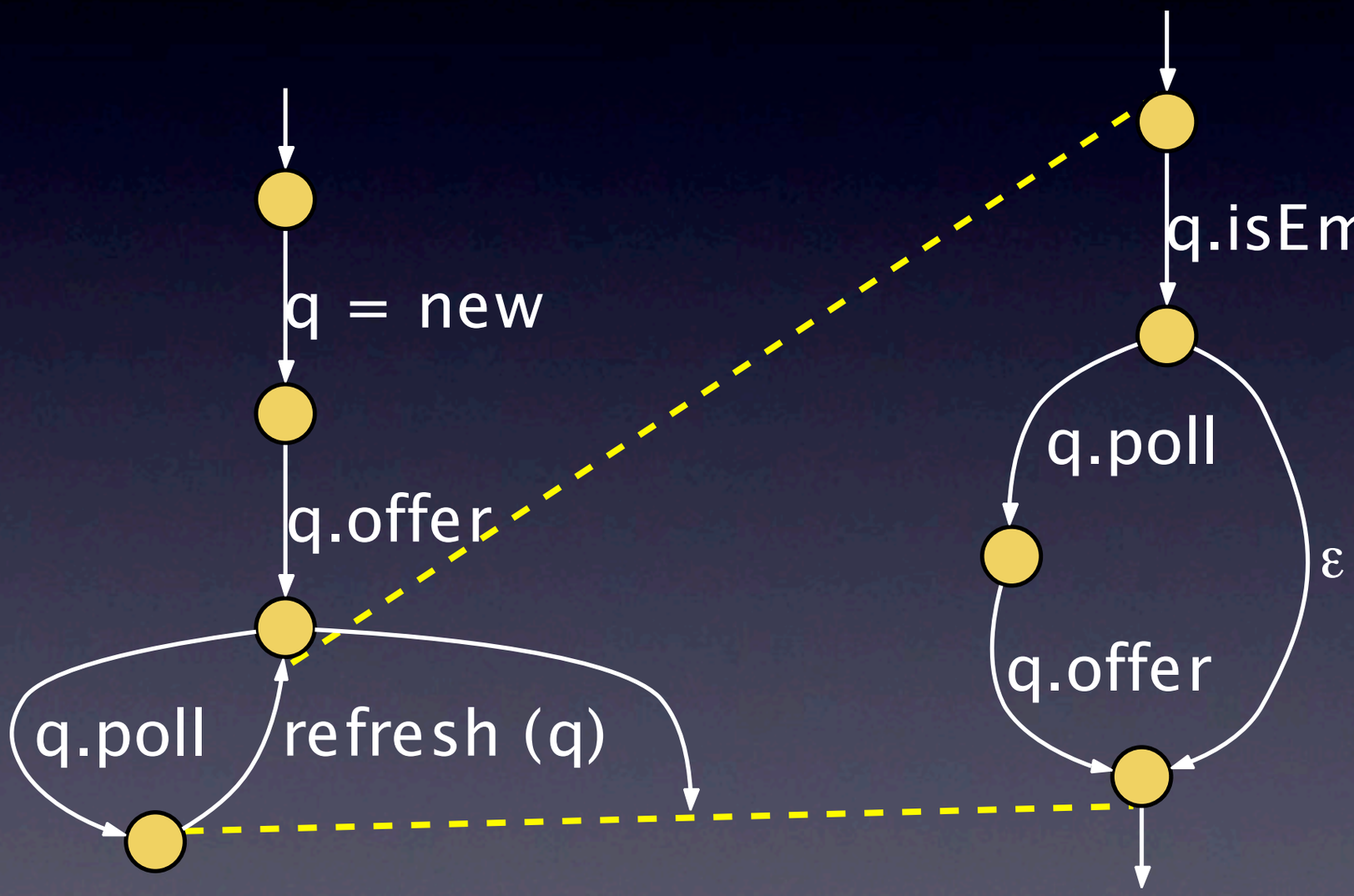
```
void refresh (Queue q) {
  if (!q.isEmpty ()) {
    e = q.poll ();
    …
    q.offer (e);
  }
}
```

Inlining models

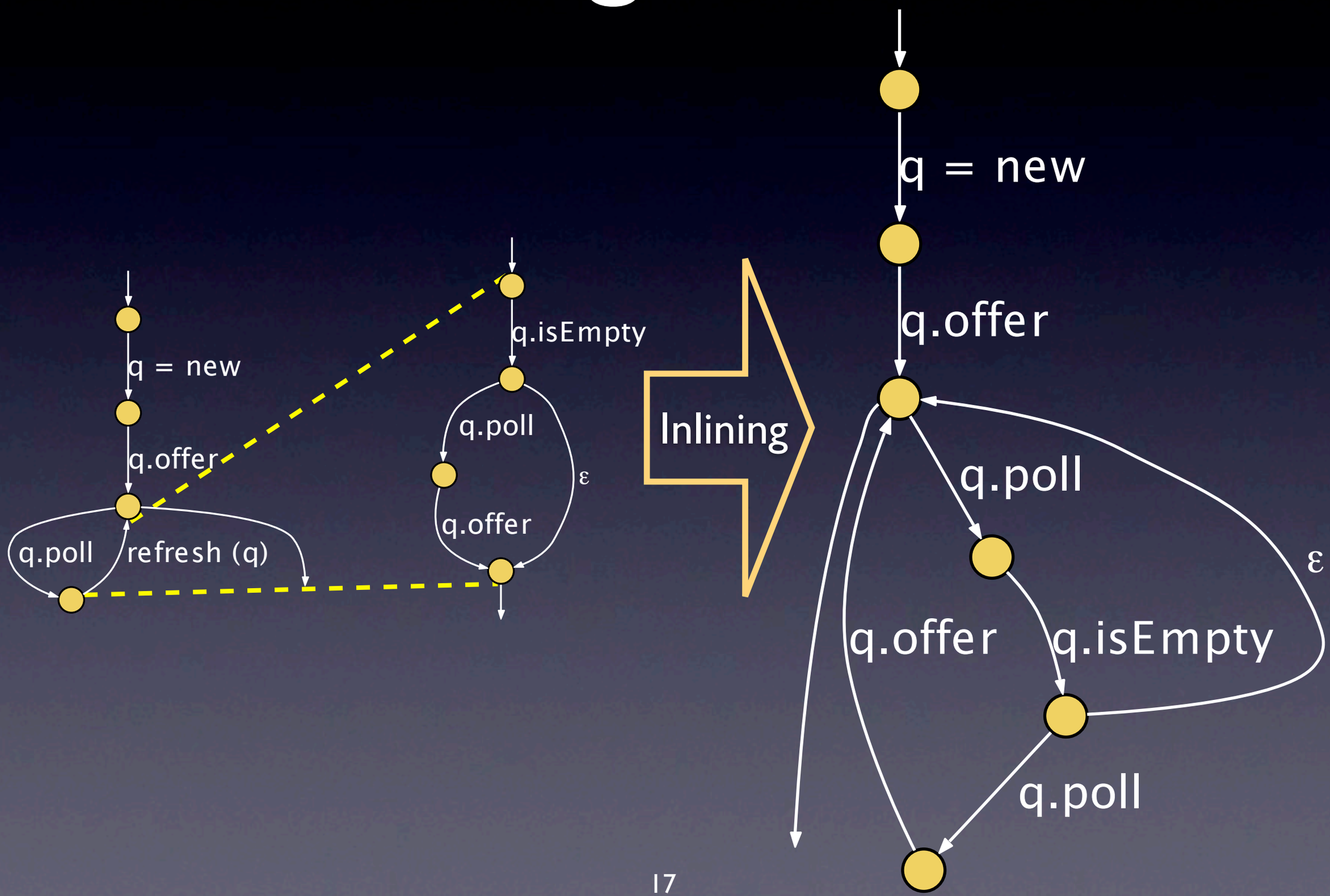



```
void refresh (Queue q) {
  if (!q.isEmpty ()) {
    e = q.poll ();
    …
    q.offer (e);
  }
}
```

Inlining models



Inlining models



Subject: AspectJ

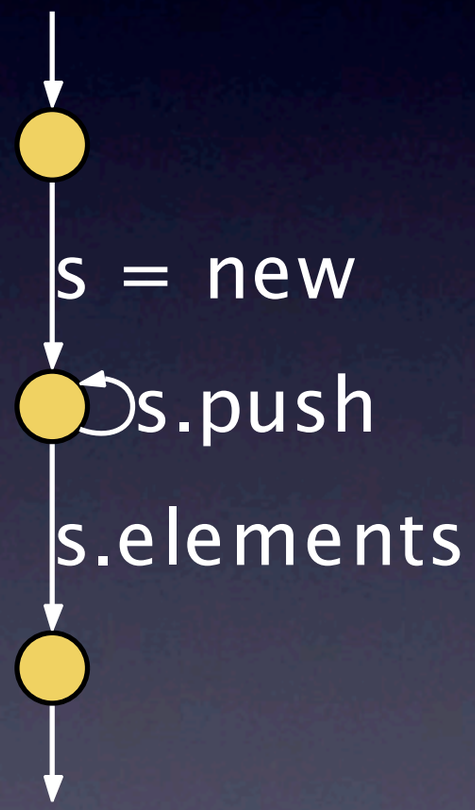
jar file	Size (kB)	Classes	Methods
aspectjlib.jar	8	3	21
aspectjrt.jar	110	68	496
aspectjweaver.jar	1800	956	9999
aspectjtools.jar	8100	2980	36372

Time & Models

jar file	Classes	Models	Time
aspectjlib.jar	3	116	0:01
aspectjrt.jar	68	2865	0:04
aspectjweaver.jar	956	74236	1:23
aspectjtools.jar	2980	243804	7:33

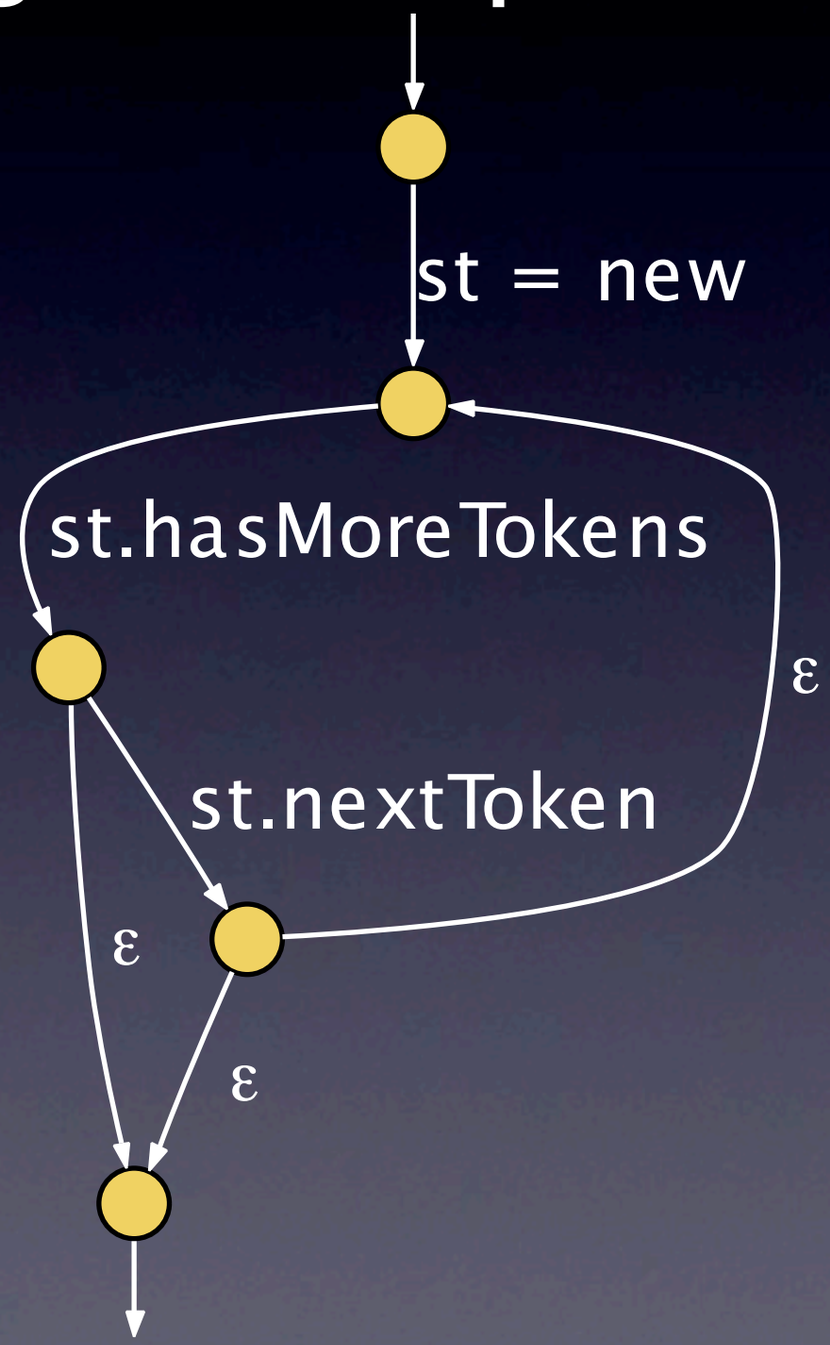
Stack s

in ThreadStackImpl1 1.getThreadStack()

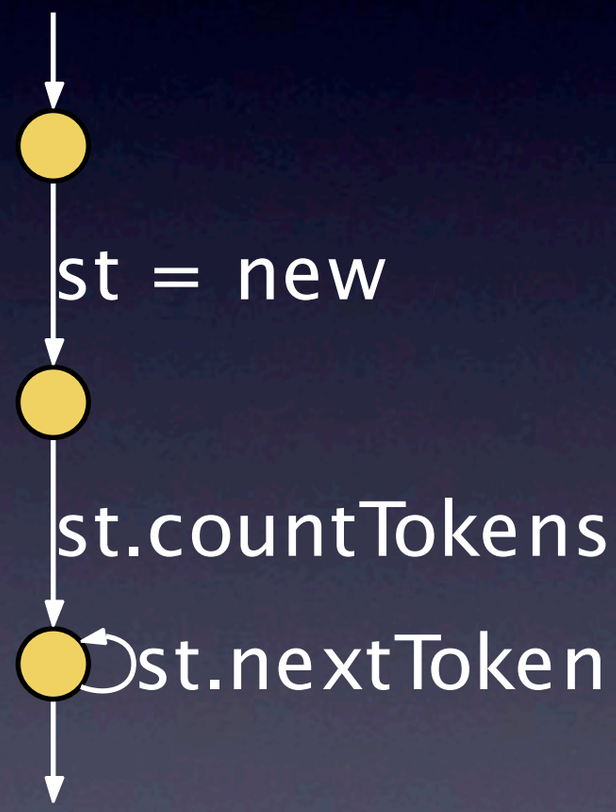


StringTokenizer st

in AdviceSignatureImpl.toAdviceName()

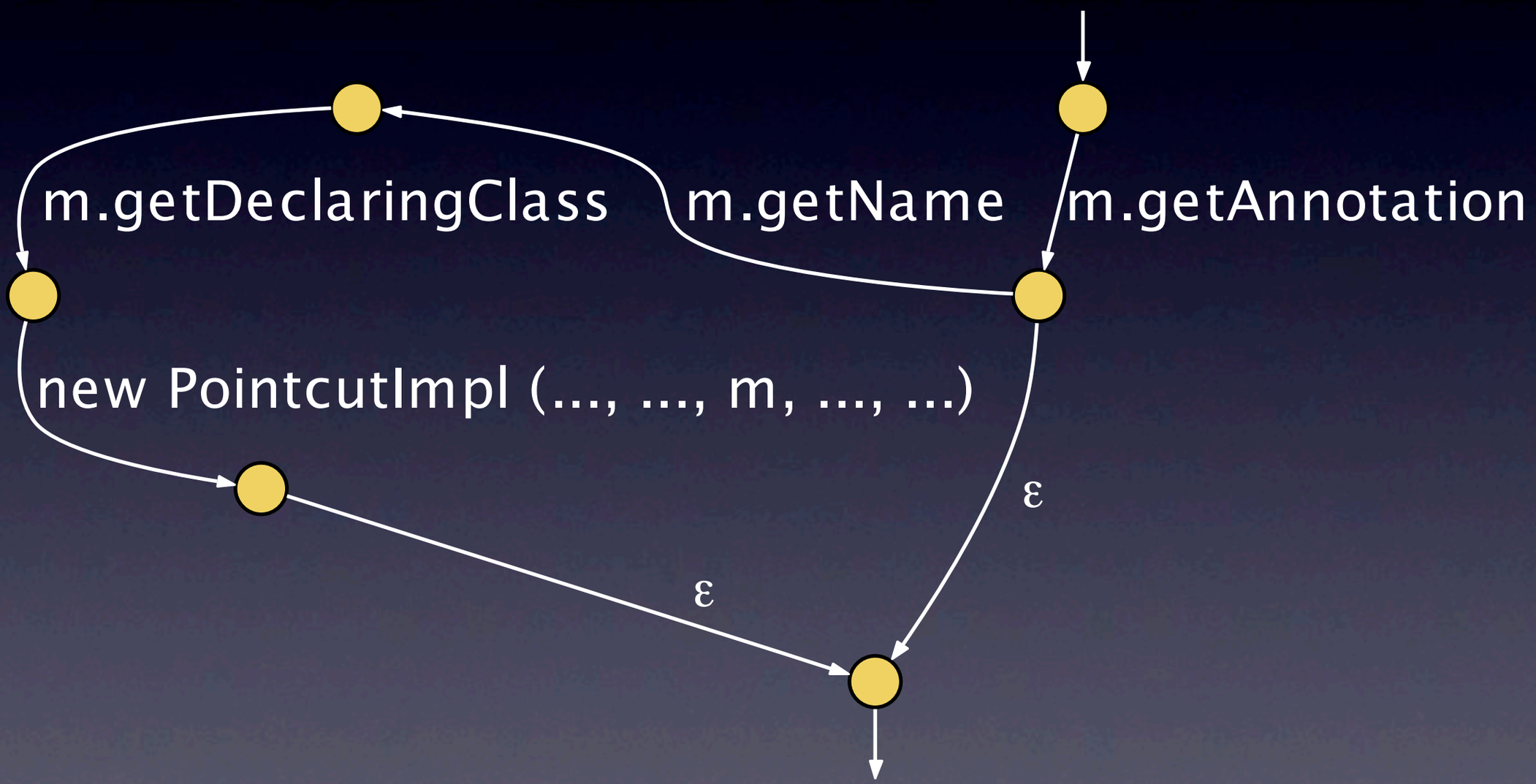


StringTokenizer st
in Factory.makeConstructorSig()



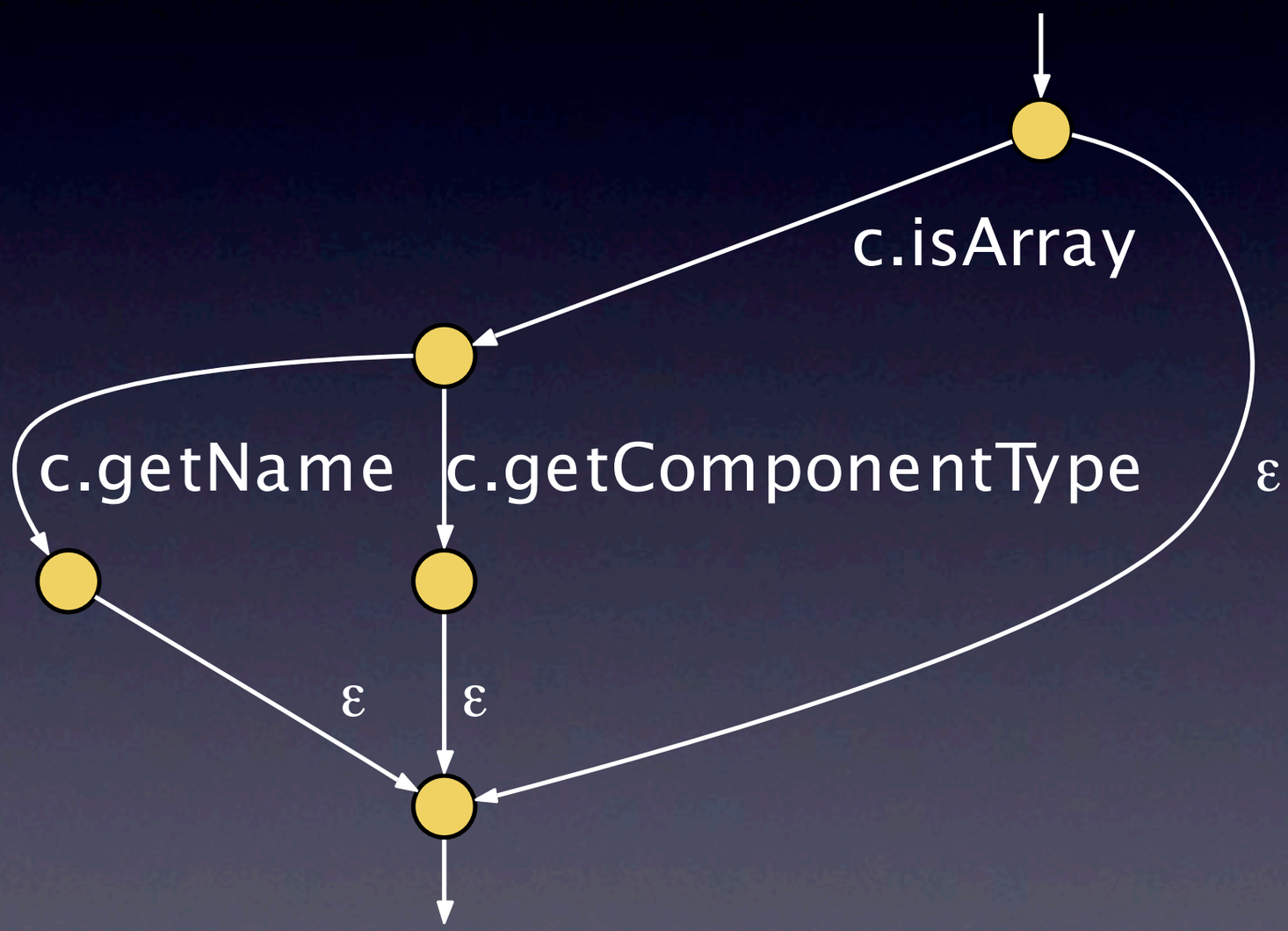
Method m

in AjTypeImpl.asPointcut()



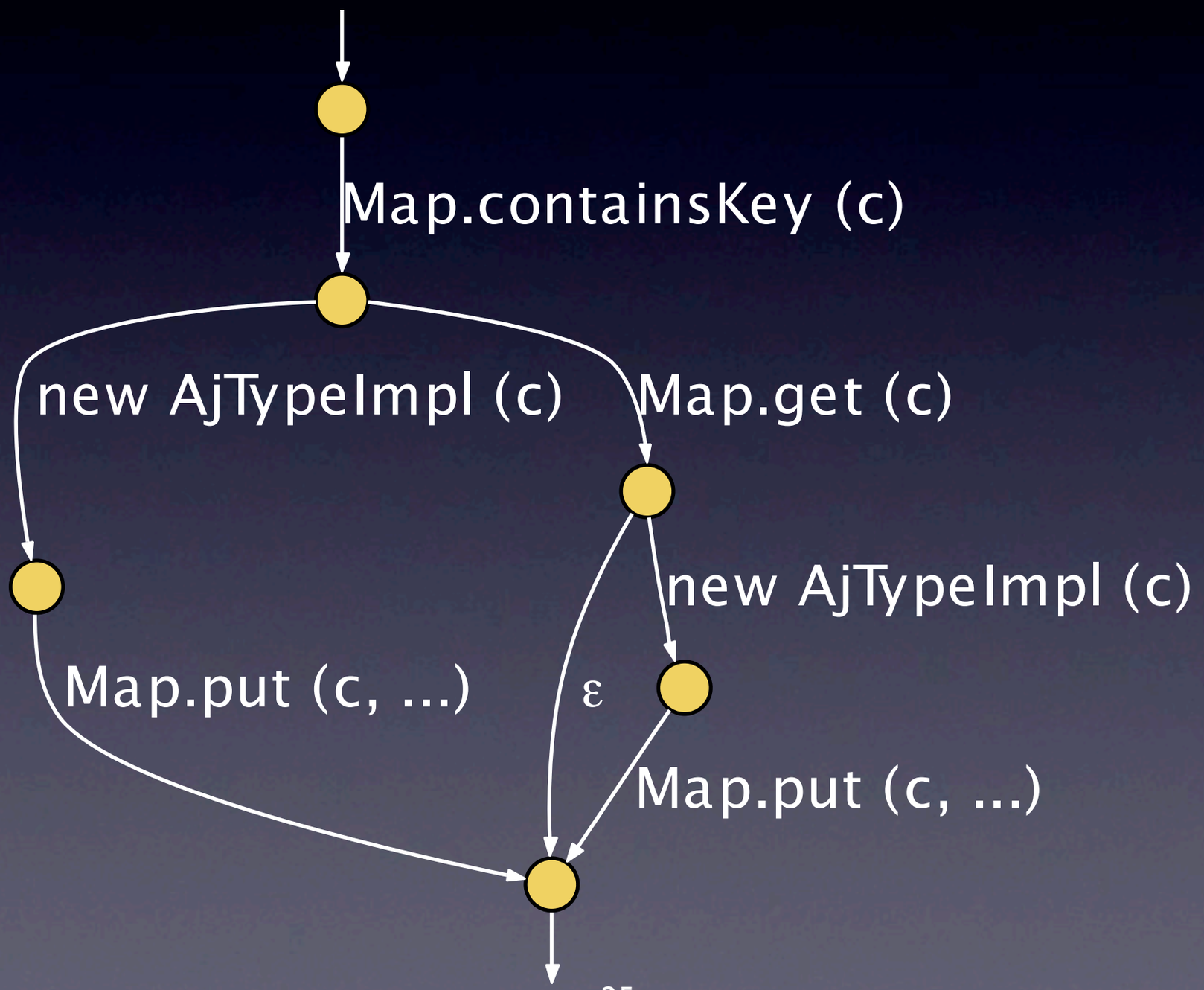
Class c

in SignatureImpl.shortTypeName()

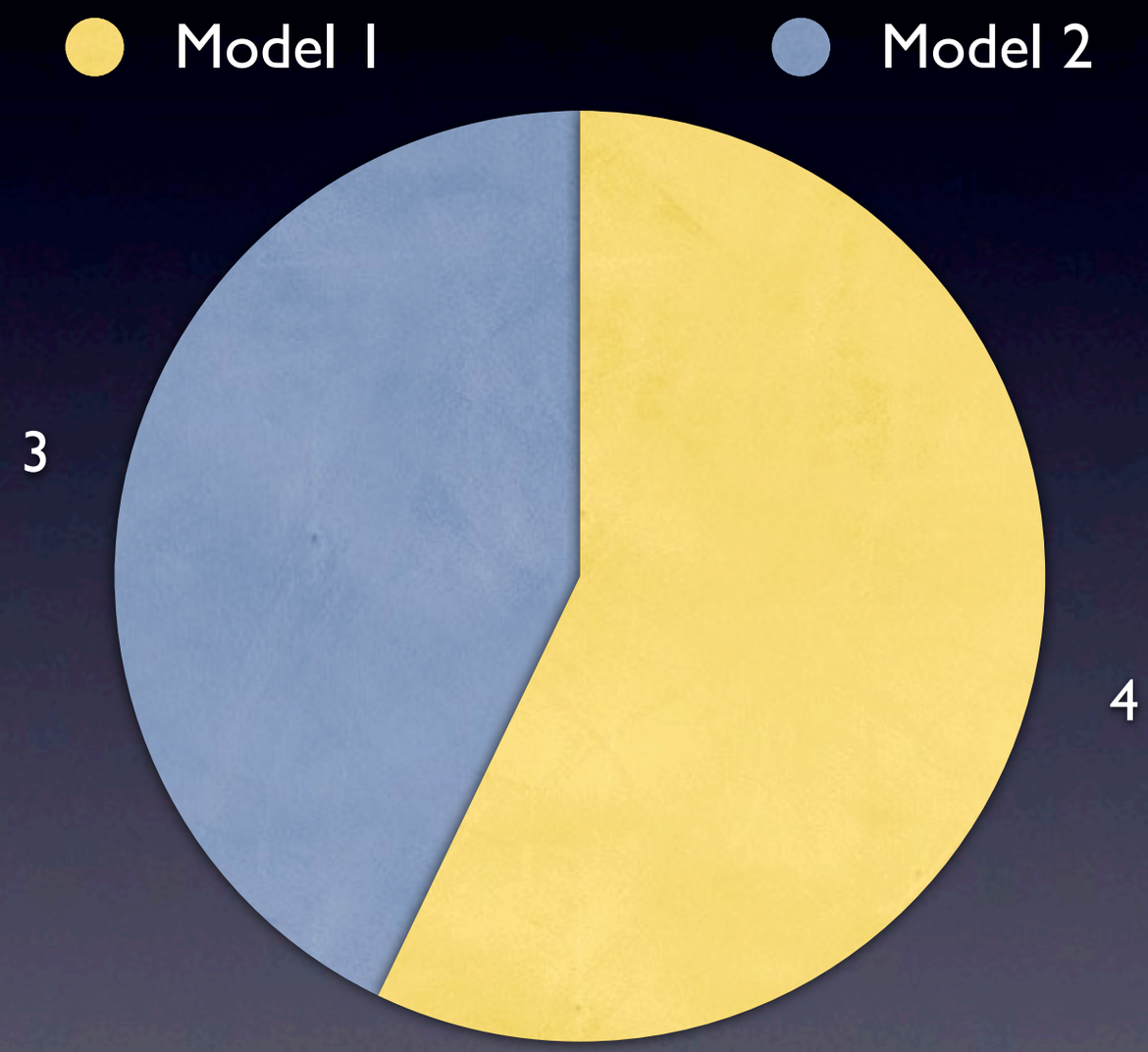


Class c

in AjTypeSystem.getAjType()

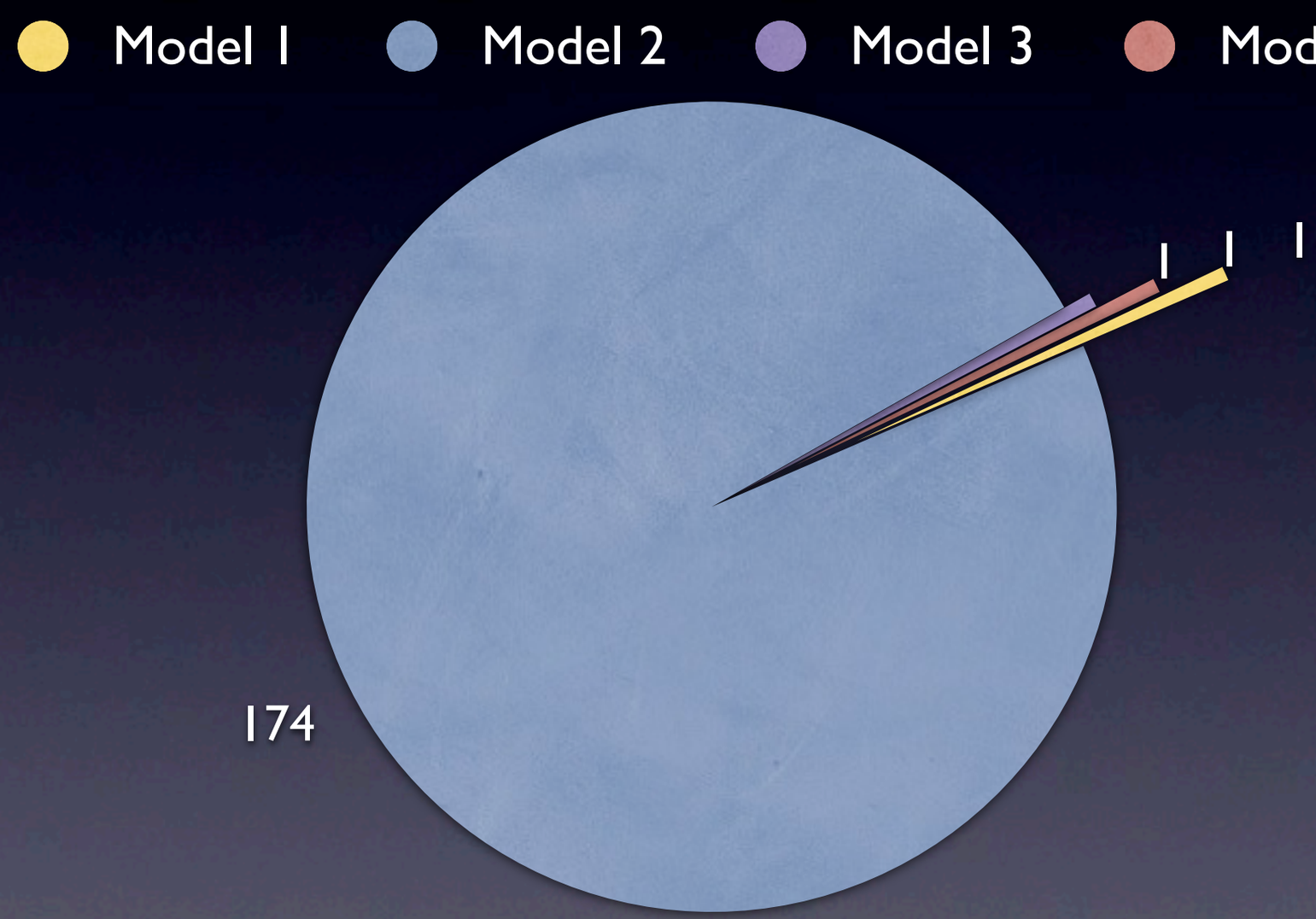


Infrequent models (1)



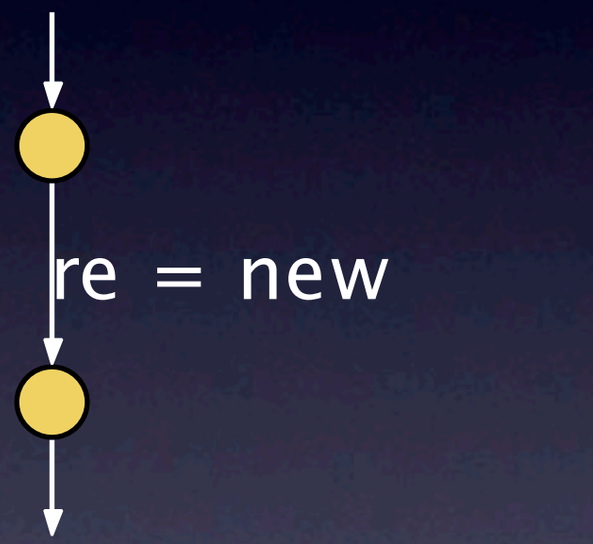
Occurrences of models: SystemColor

Infrequent models (2)



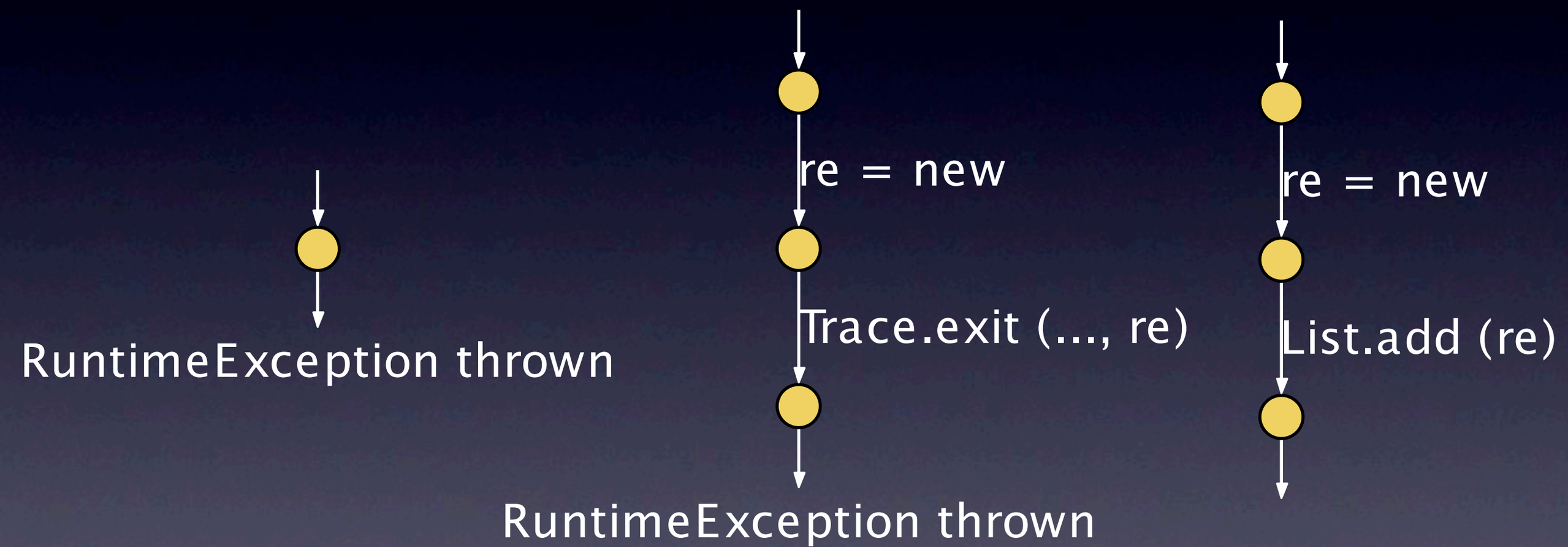
Occurrences of models: RuntimeException

Infrequent models (3)



The most frequently occurring RuntimeException model

Infrequent models (3)

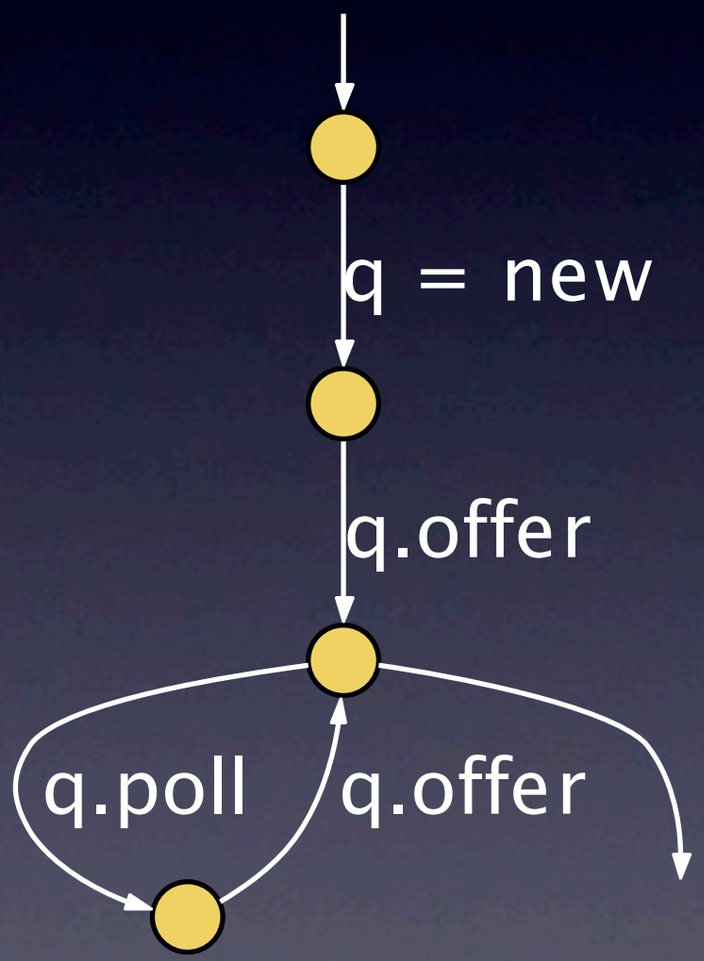


Anomalous RuntimeException models

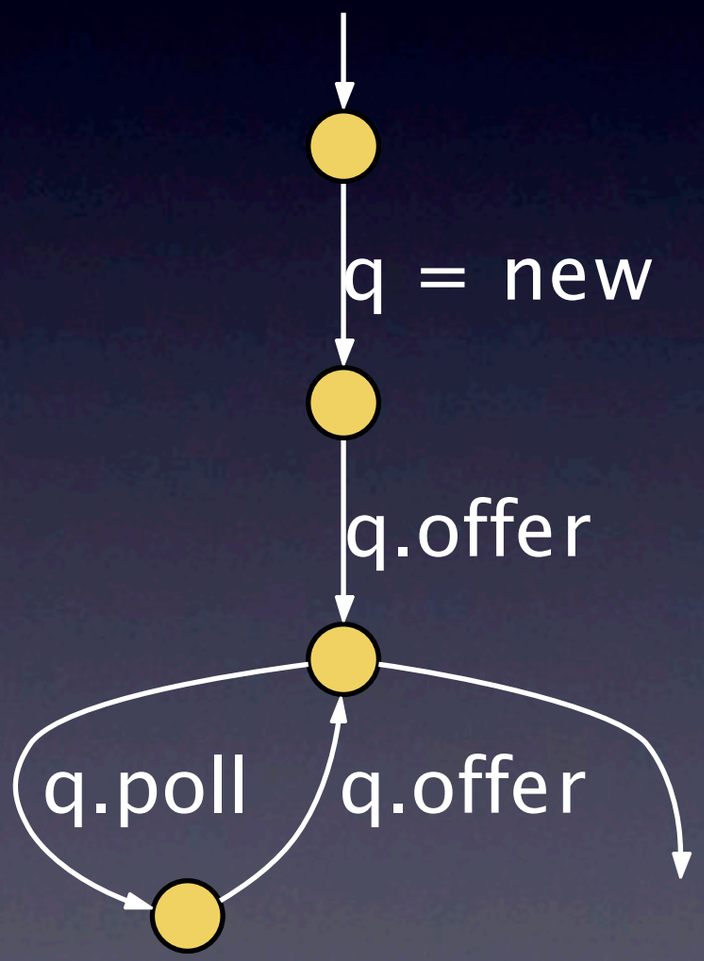
Infrequent models statistics

jar file	Total Classes	Anomalies Classes	Models	Time
aspectjlib.jar	3	0	0	0:02
aspectjrt.jar	68	0	0	0:05
aspectjweaver.jar	956	8	75	1:16
aspectjtools.jar	2980	20	249	9:06

Subsequences



Subsequences



Subsequences extraction

```
(new, offer)
(new, poll)
(offer, offer)
(offer, poll)
(poll, offer)
(poll, poll)
```

Frequent subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Frequent subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Subsequence 48
forms an itemset with support 4

Frequent subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Frequent subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Subsequences 45 and 48
form an itemset with support 3

Missing subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Missing subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Missing subsequences point to anomalies

Missing subsequences

Method	Ids of subsequences
Dump.println	28, 45, 46, 47, 48, 4386
ClassPathManager.<init>	45, 46, 47, 48
DeclareParents.verify…	4, 48, 10537
BcelObjectType.get…	45, 48, 12028

Subsequence 45: `(hasNext, hasNext)`

Subsequence 48: `(hasNext, next)`

Missing subsequences point to anomalies

The anomalous method

```
/**
 * This method looks through the type hierarchy for some target type - it is attempting to
 * find an existing parameterization that clashes with the new parent that the user
 * wants to apply to the type.  If it finds an existing parameterization that matches the
 * new one, it silently completes, if it finds one that clashes (e.g. a type already has
 * A<String> when the user wants to add A<Number>) then it will produce an error.
 *
 * It uses recursion and exits recursion on hitting 'jlObject'
  *
 * Related bugzilla entries: pr110788
 */
private boolean verifyNoInheritedAlternateParameterization(ResolvedType typeToVerify,ResolvedType newParent,World world) {


        if (typeToVerify.equals(ResolvedType.OBJECT)) return true;

        ResolvedType newParentGenericType = newParent.getGenericType();
        Iterator iter = typeToVerify.getDirectSupertypes();
        while (iter.hasNext()) {
                ResolvedType supertype = (ResolvedType)iter.next();
                if ( ((supertype.isRawType() && newParent.isParameterizedType()) ||
                      (supertype.isParameterizedType() && newParent.isRawType())) && newParentGenericType.equals(supertype.getGenericType())) {
                        // new parent is a parameterized type, but this is a raw type
                        world.getMessageHandler().handleMessage(new Message(
                                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTIPLE_PARAMETERIZATIONS,newParent.getName(),typeToVerify.getName(),supertype.getName()),
                                getSourceLocation(), true, new ISourceLocation[]{typeToVerify.getSourceLocation()}));
                        return false;
                }
                if (supertype.isParameterizedType()) {
                        ResolvedType generictype = supertype.getGenericType();

                        // If the generic types are compatible but the parameterizations aren't then we have a problem
                        if (generictype.isAssignableFrom(newParentGenericType) &&
                                        !supertype.isAssignableFrom(newParent)) {
                                world.getMessageHandler().handleMessage(new Message(
                                                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTIPLE_PARAMETERIZATIONS,newParent.getName(),typeToVerify.getName(),supertype.getName()),
                                                getSourceLocation(), true, new ISourceLocation[]{typeToVerify.getSourceLocation()}));
                                return false;
                        }
                }
                return verifyNoInheritedAlternateParameterization(supertype,newParent,world);
        }
        return true;
}
```

The anomalous method

```
/**
 * This method looks through the type hierarchy for some target type - it is attempting to
 * find an existing parameterization that clashes with the new parent that the user
 * wants to apply to the type.  If it finds an existing parameterization that matches the
 * new one, it silently completes, if it finds one that clashes (e.g. a type already has
 * A<String> when the user wants to add A<Number>) then it will produce an error.
 *
 * It uses recursion and exits recursion on hitting 'jlObject'
  *
 * Related bugzilla entries: pr110788
 */
private boolean verifyNoInheritedAlternateParameterization(ResolvedType typeToVerify,ResolvedType newParent,World world) {


        if (typeToVerify.equals(ResolvedType.OBJECT)) return true;

        ResolvedType newParentGenericType = newParent.getGenericType();
        Iterator iter = typeToVerify.getDirectSupertypes();
        while (iter.hasNext()) {
                ResolvedType supertype = (ResolvedType)iter.next();
                if ( ((supertype.isRawType() && newParent.isParameterizedType()) ||
                      (supertype.isParameterizedType() && newParent.isRawType())) && newParentGenericType.equals(supertype.getGenericType())) {
                        // new parent is a parameterized type, but this is a raw type
                        world.getMessageHandler().handleMessage(new Message(
                                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTIPLE_PARAMETERIZATIONS,newParent.getName(),typeToVerify.getName(),supertype.getName()),
                                getSourceLocation(), true, new ISourceLocation[]{typeToVerify.getSourceLocation()}));
                        return false;
                }
                if (supertype.isParameterizedType()) {
                        ResolvedType generictype = supertype.getGenericType();

                        // If the generic types are compatible but the parameterizations aren't then we have a problem
                        if (generictype.isAssignableFrom(newParentGenericType) &&
                                        !supertype.isAssignableFrom(newParent)) {
                                world.getMessageHandler().handleMessage(new Message(
                                                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTIPLE_PARAMETERIZATIONS,newParent.getName(),typeToVerify.getName(),supertype.getName()),
                                                getSourceLocation(), true, new ISourceLocation[]{typeToVerify.getSourceLocation()}));
                                return false;
                        }
                }
                return verifyNoInheritedAlternateParameterization(supertype,newParent,world);
        }
        return true;
}
```

This method is in fact buggy

Another anomaly

```
…
List values = …;
for (Iterator it = values.iterator();
     it.hasNext();) {
    …
    return …;
}
…
```

The list holds in fact at most one element

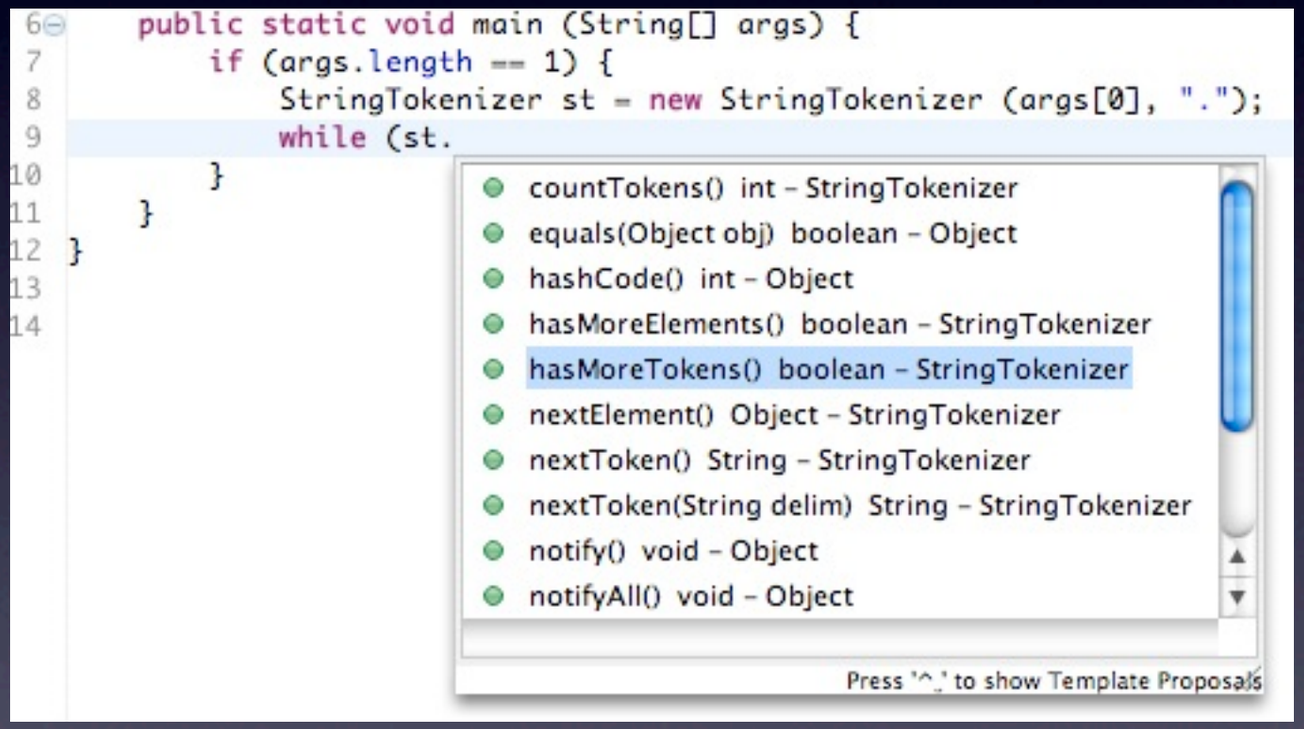
Frequent subsequences statistics

jar file	Classes	Rules	Flaws	Time
aspectjlib.jar	3	0	0	0:03
aspectjrt.jar	68	2	0	0:04
aspectjweaver.jar	956	38	26	1:06
aspectjtools.jar	2980	161	71	6:50

Frequent subsequences statistics

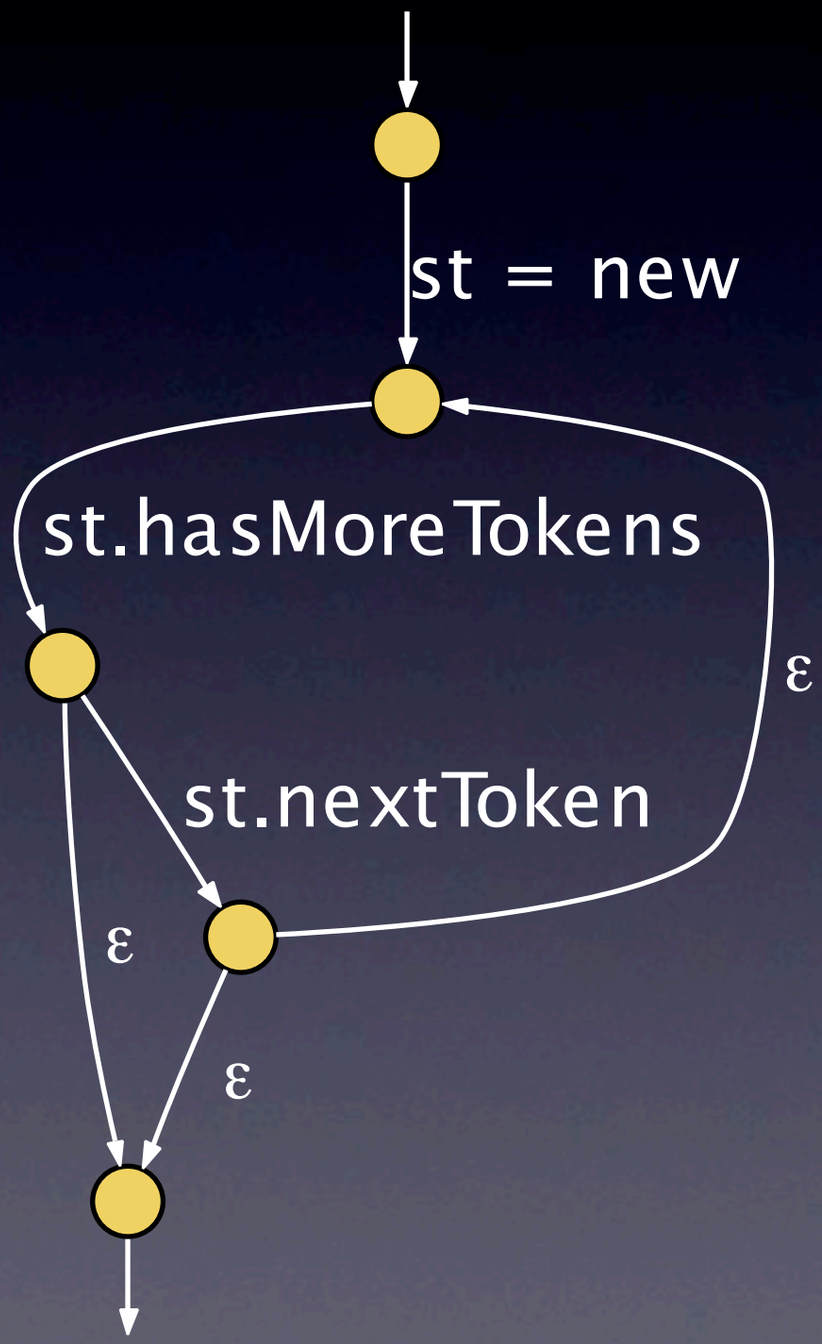
- ✓ Fast enough to be practically useful
- ✓ Found previously unknown bug in AspectJ
- ✓ Found violations of convention in AspectJ
- ✗ Many false positives

Suggesting method calls



In Eclipse

Suggesting method calls




```
6    public static void main (String[] args) {
7        if (args.length == 1) {
8            StringTokenizer st = new StringTokenizer (args[0], ".");
9            while (st.
10       }
11   }
12 }
13
14
```

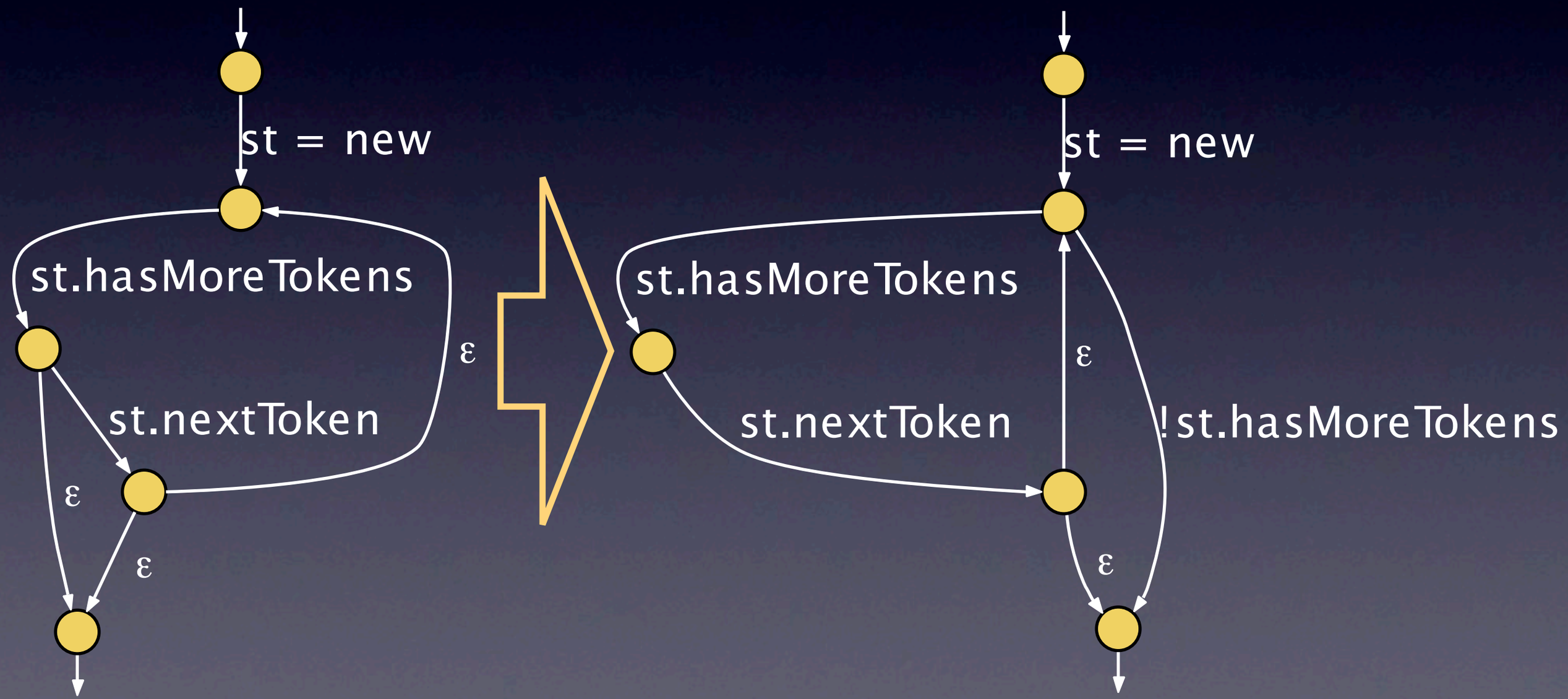
hasMoreTokens() boolean – StringTokenizer
countTokens() int – StringTokenizer
equals(Object obj) boolean – Object
hashCode() int – Object
hasMoreElements() boolean – StringTokenizer
nextElement() Object – StringTokenizer
nextToken() String – StringTokenizer
nextToken(String delim) String – StringTokenizer
notify() void – Object
notifyAll() void – Object

Press '^␣' to show Template Proposals

Using models

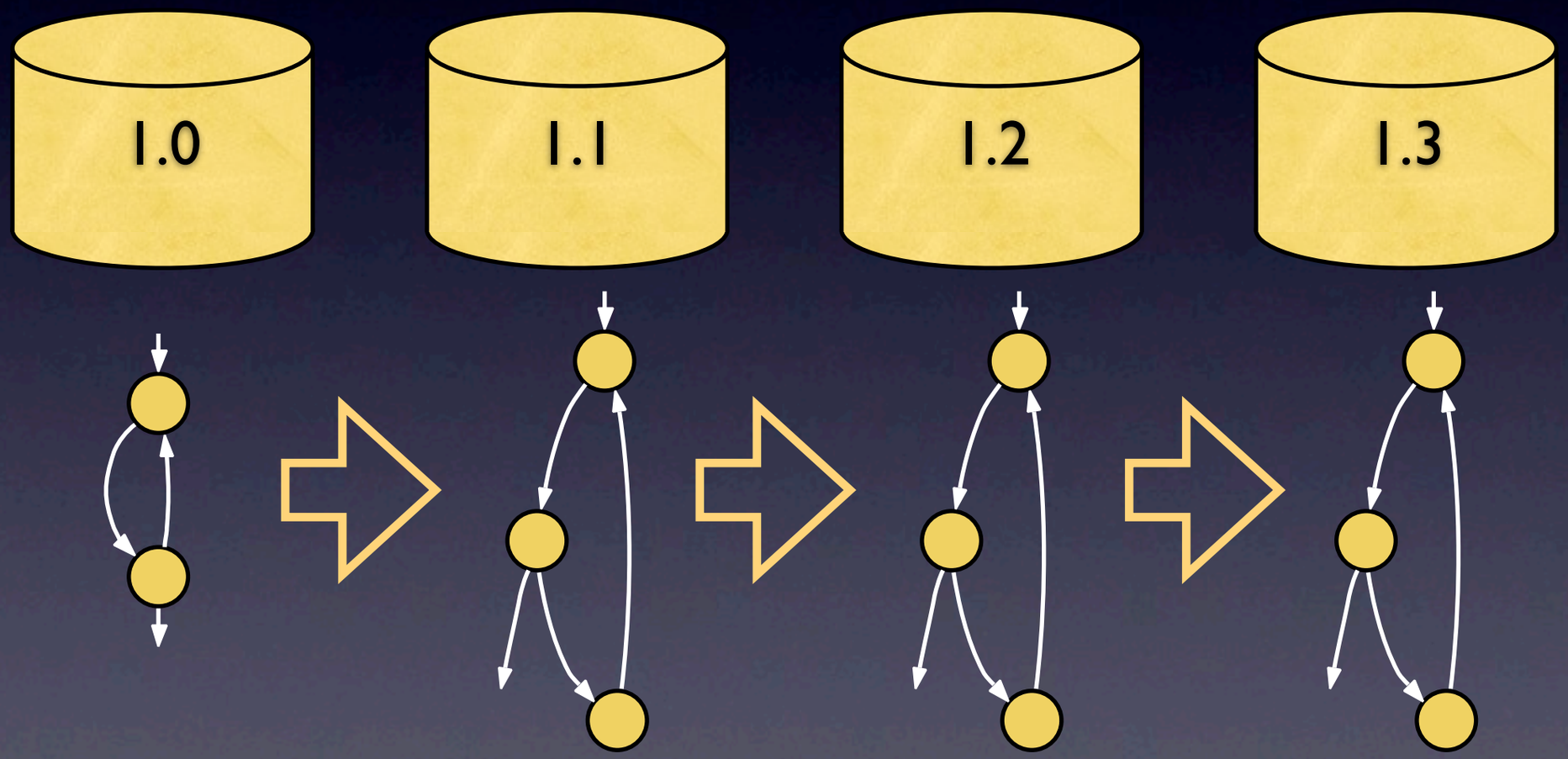
Other ideas (1)

- Use conditions in code to enhance models



Other ideas (2)

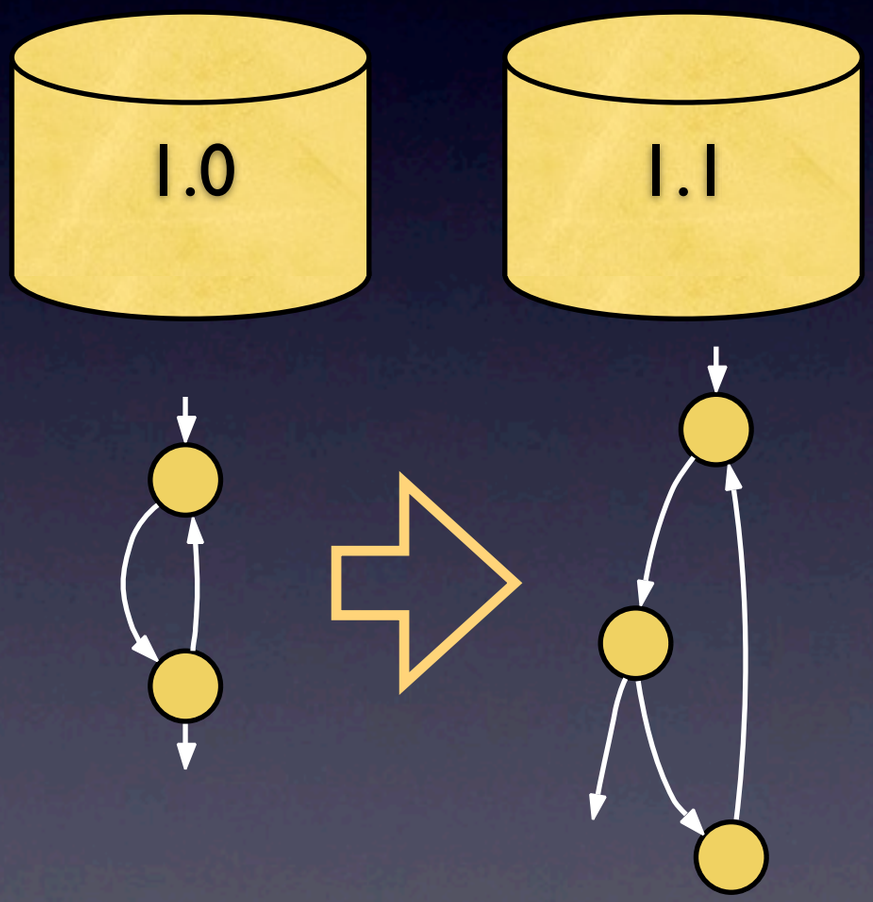
- Suggest changes based on model's evolution



Evolution of a model of class A originating from method B

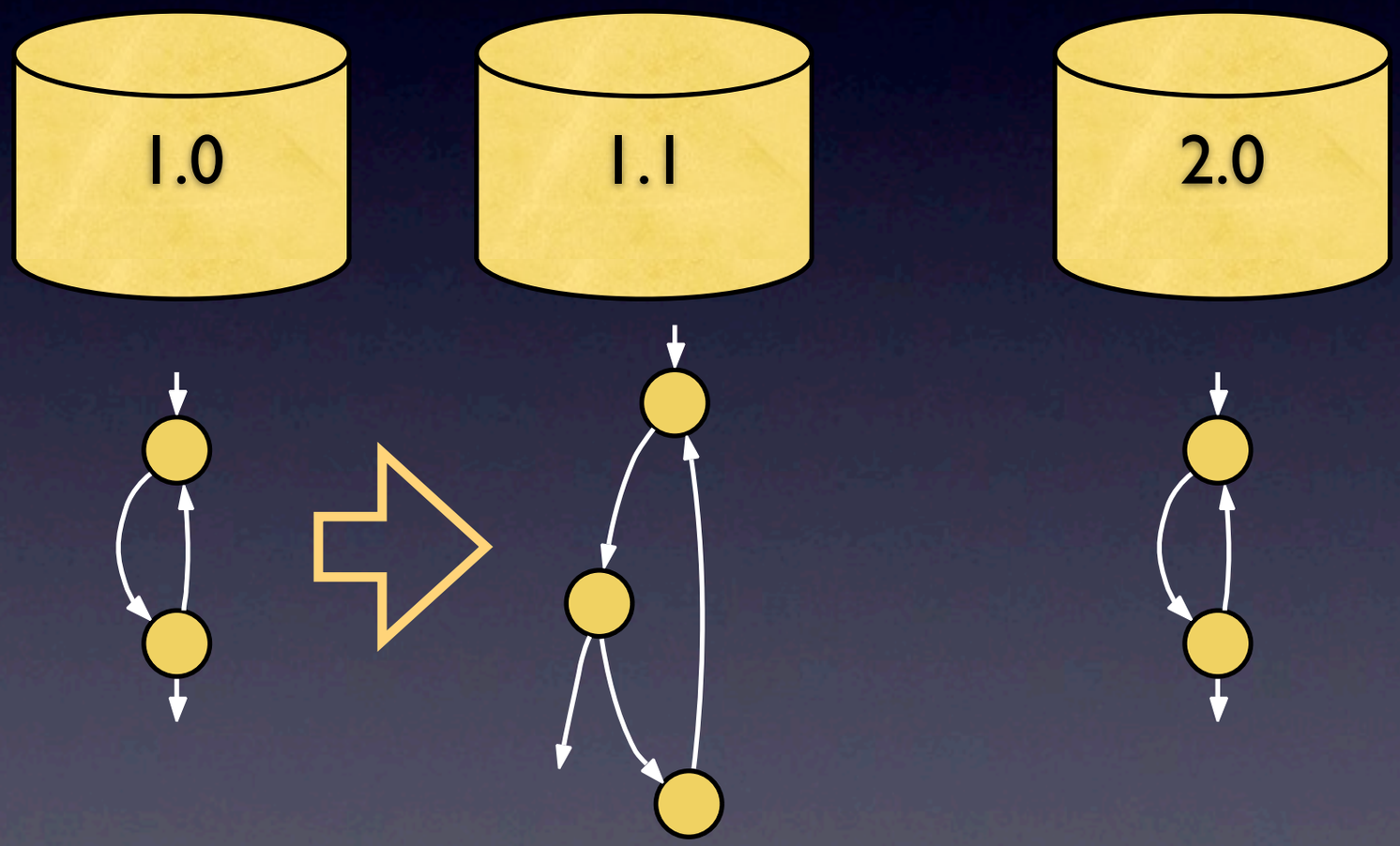
Other ideas (2)

- Suggest changes based on model's evolution



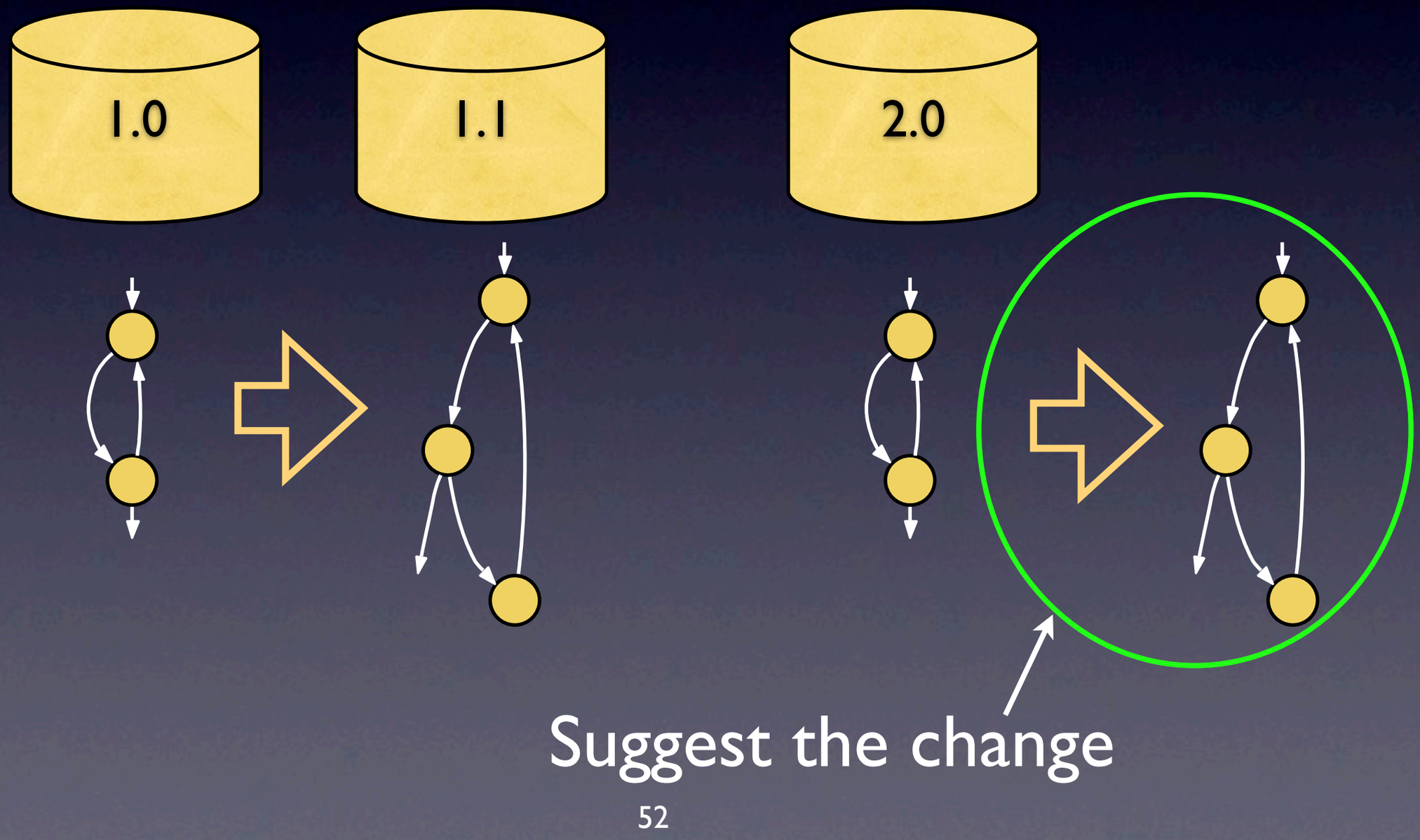
Other ideas (2)

- Suggest changes based on model's evolution

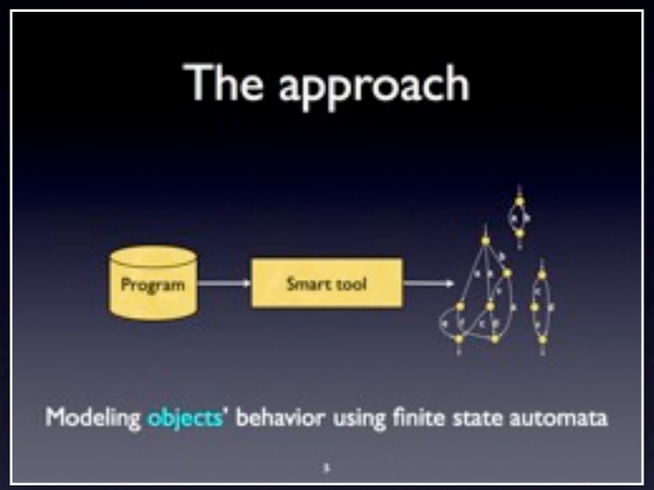


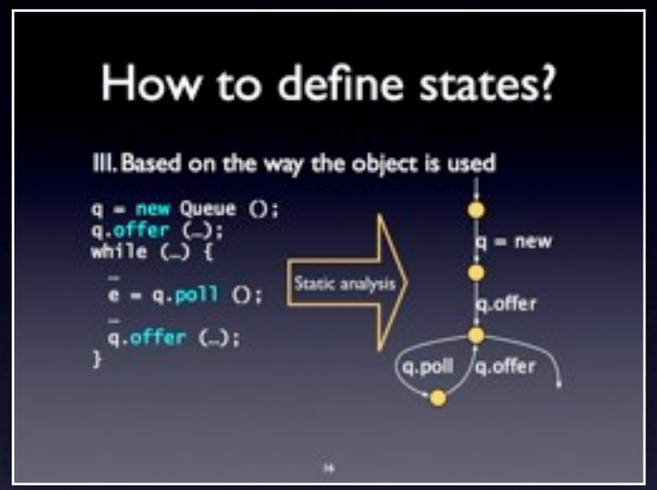
Other ideas (2)

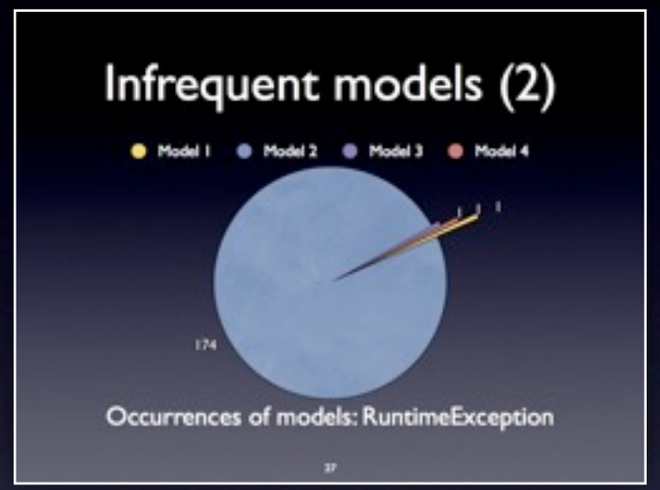
- Suggest changes based on model's evolution

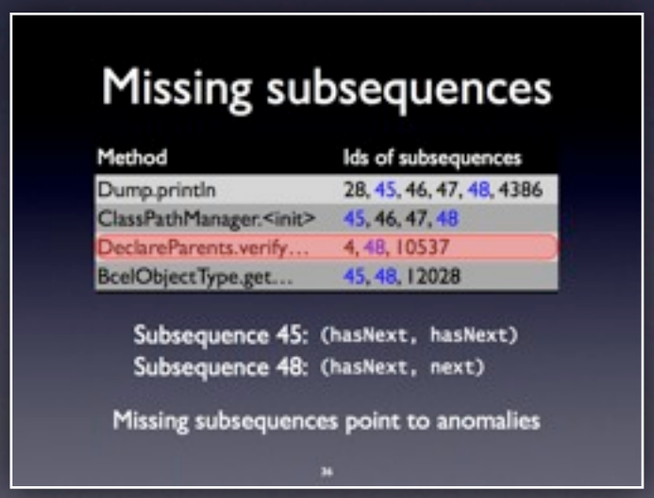


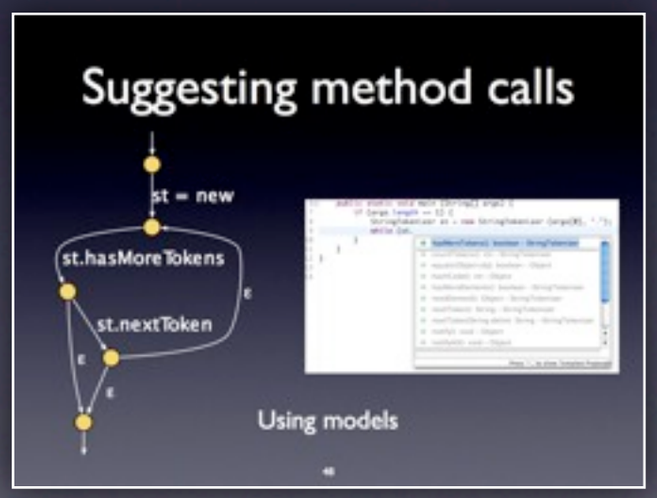
Conclusion

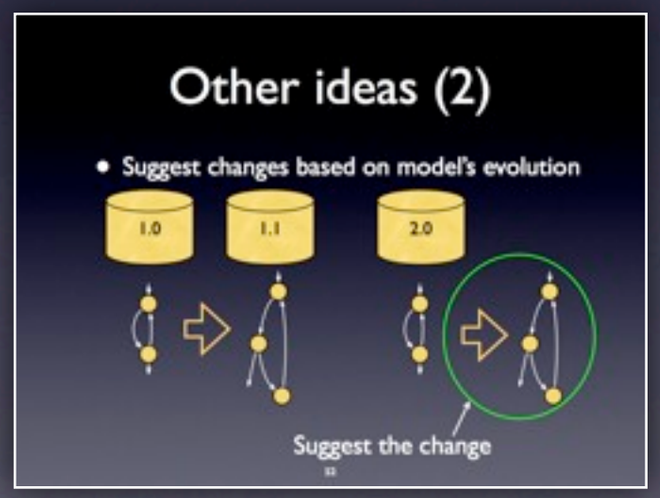

The approach
Program
Smart tool
Modeling objects' behavior using finite state automata


How to define states?
III. Based on the way the object is used
q = new Queue ();
q.offer (…);
while (…) {
e = q.poll ();
q.offer (…);
}
Static analysis
q = new
q.offer
q.poll
q.offer


Infrequent models (2)
Model 1
Model 2
Model 3
Model 4
174
1
Occurrences of models: RuntimeException


Missing subsequences
Method
Ids of subsequences
Dump.println
28, 45, 46, 47, 48, 4386
ClassPathManager.<init>
45, 46, 47, 48
DeclareParents.verify…
4, 48, 10537
BcelObjectType.get…
45, 48, 12028
Subsequence 45: (hasNext, hasNext)
Subsequence 48: (hasNext, next)
Missing subsequences point to anomalies


Suggesting method calls
st = new
st.hasMoreTokens
st.nextToken
ε
ε
ε
Using models


Other ideas (2)
Suggest changes based on model's evolution
1.0
1.1
2.0
Suggest the change