PROGRAMMING WITH MILLIONS OF EXAMPLES

Alon Mishne Eran Yahav

Technion, Israel

Components are Prevalent

Component APIs are Complicated

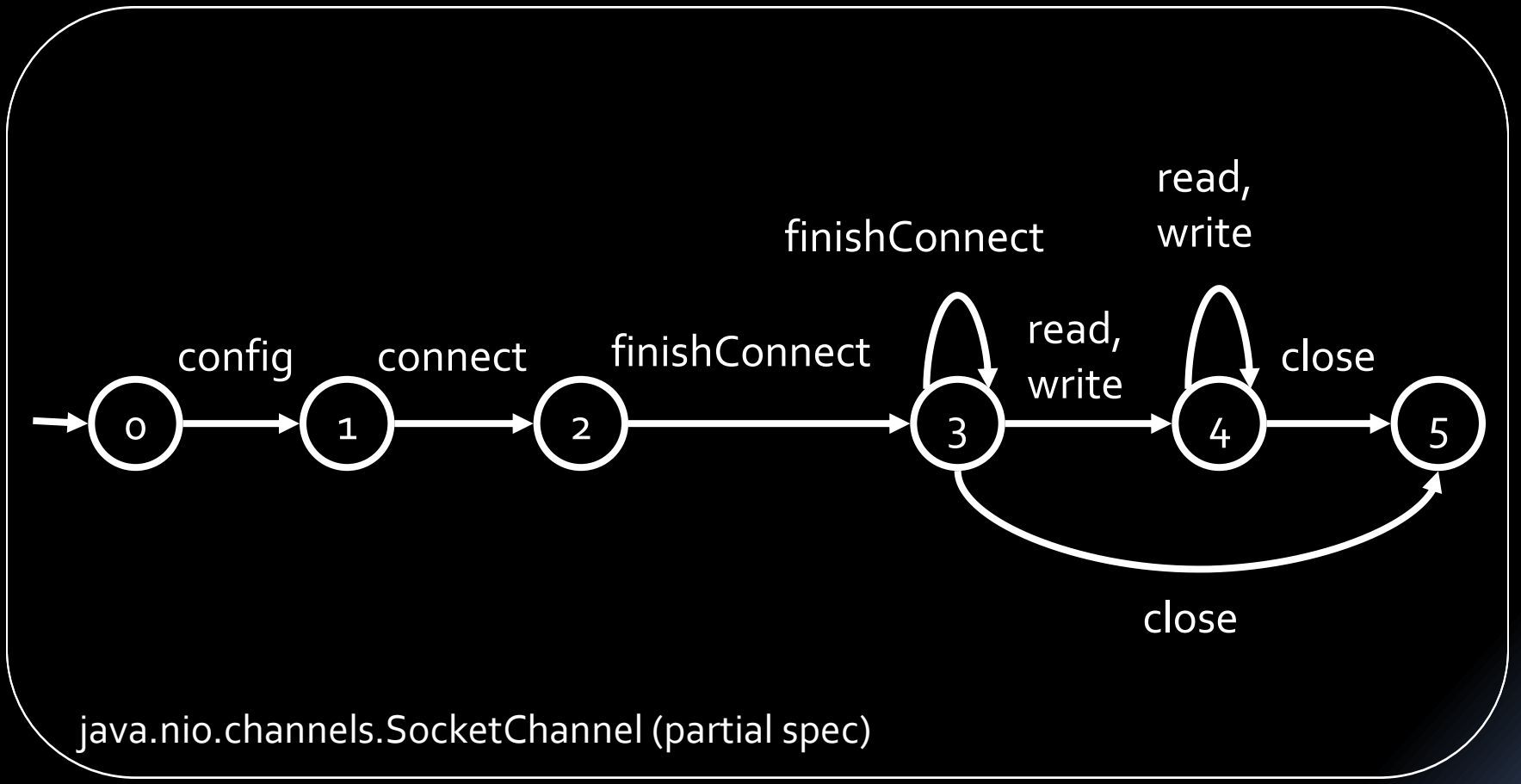
There is only one thing more painful than learning from experience and that is not learning from experience.
– Archibald MacLeish

Temporal API Specifications

```
java.nio.channels.SocketChannel {

  boolean connect(SocketAddress remote)

  int read(ByteBuffer dst)

  int write(ByteBuffer src)

  SelectableChannel configureBlocking(boolean block)

  boolean finishConnect()

  boolean isBlocking()

  void close()
  …
}
```

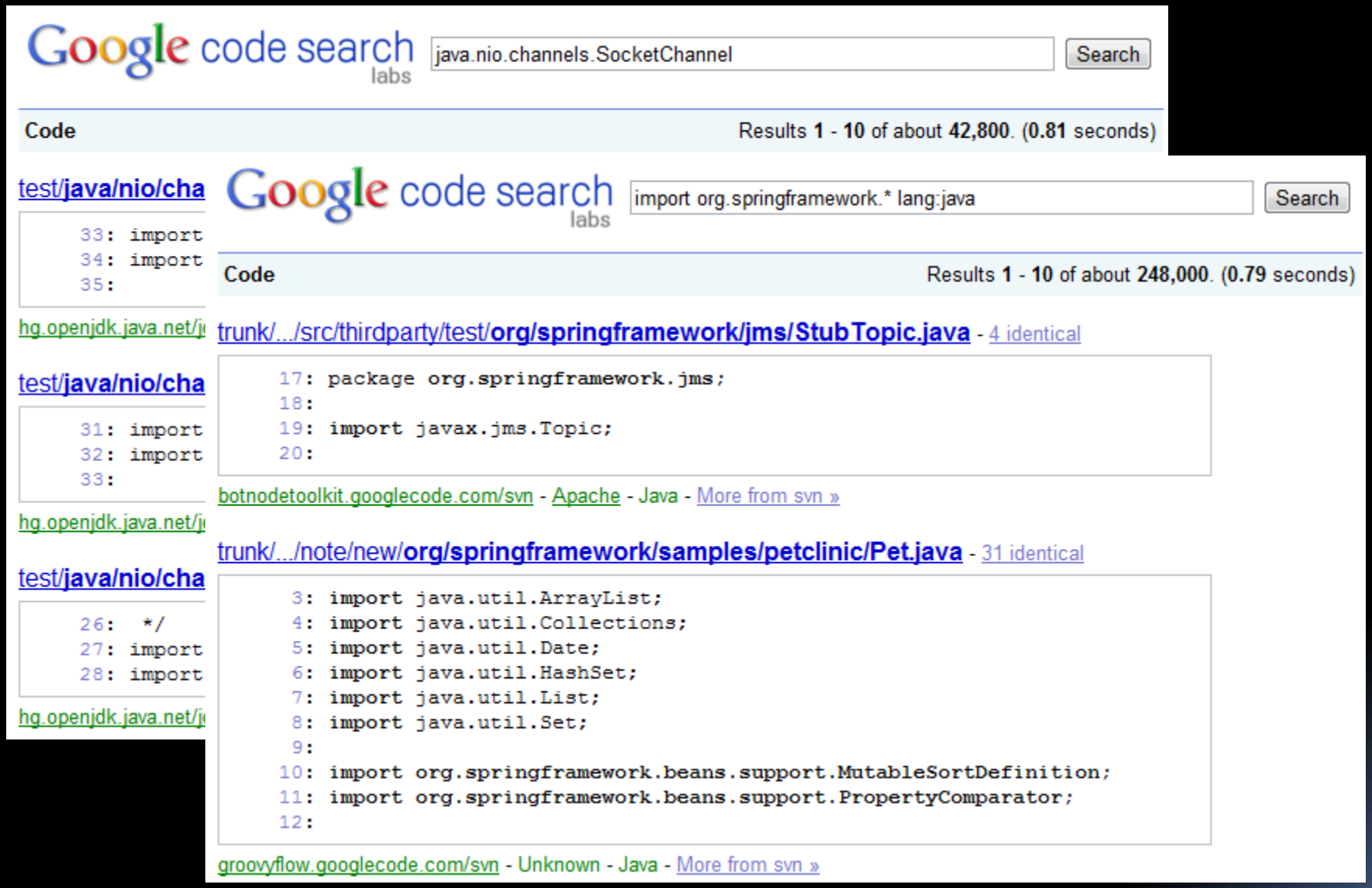
- Legal interactions with a component
- What methods could be called at every internal state

Temporal API Specifications



- Legal interactions with a component
- What methods could be called at every internal state

Examples are Prevalent



Google code search labs

java.nio.channels.SocketChannel

Search

Code

Results 1 - 10 of about 42,800. (0.81 seconds)

test/java/nio/cha

```
33: import
34: import
35:
```

hg.openjdk.java.net/j

test/java/nio/cha

```
31: import
32: import
33:
```

hg.openjdk.java.net/j

test/java/nio/cha

```
26:  */
27: import
28: import
```

hg.openjdk.java.net/j

Google code search labs

import org.springframework.* lang:java

Search

Code

Results 1 - 10 of about 248,000. (0.79 seconds)

trunk/.../src/thirdparty/test/**org/springframework/jms/StubTopic.java** - 4 identical

```
17: package org.springframework.jms;
18:
19: import javax.jms.Topic;
20:
```

botnodetoolkit.googlecode.com/svn - Apache - Java - More from svn »

trunk/.../note/new/**org/springframework/samples/petclinic/Pet.java** - 31 identical

```
 3: import java.util.ArrayList;
 4: import java.util.Collections;
 5: import java.util.Date;
 6: import java.util.HashSet;
 7: import java.util.List;
 8: import java.util.Set;
 9:
10: import org.springframework.beans.support.MutableSortDefinition;
11: import org.springframework.beans.support.PropertyComparator;
12:
```

groovyflow.googlecode.com/svn - Unknown - Java - More from svn »

Examples are Prevalent

stackoverflow Questions Tags Users Badges Unanswered

Retrieve column names from java.sql.ResultSet

10

With `java.sql.ResultSet` is there a way to get a column's name as a `String` by using the column's index? I had a look through the API doc but I can't find anything.

java jdbc

1

link | edit | retag | flag

edited Apr 19 '10 at 14:28
BalusC
168k • 14 • 119 • 224

asked Mar 30 '09 at 11:10
Ben

add comment

start a bounty

3 Answers

active oldest votes

19

See ResultSetMetaData

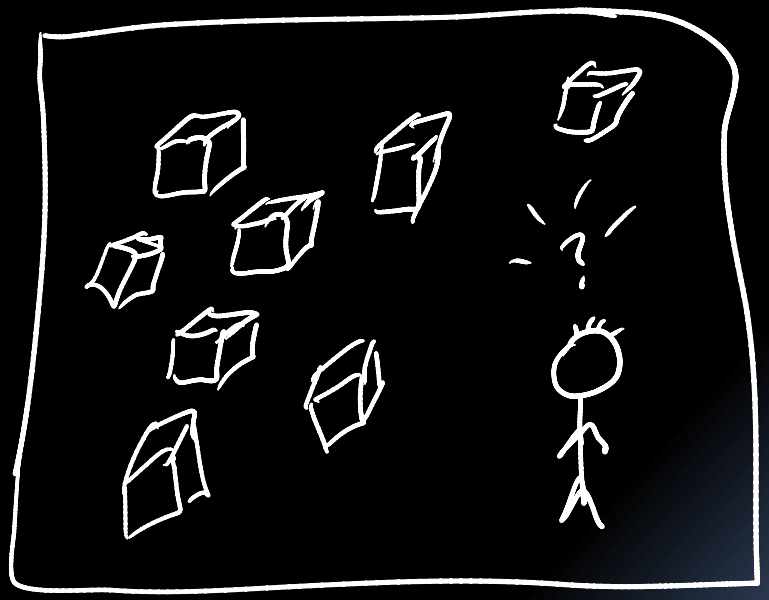
e.g.

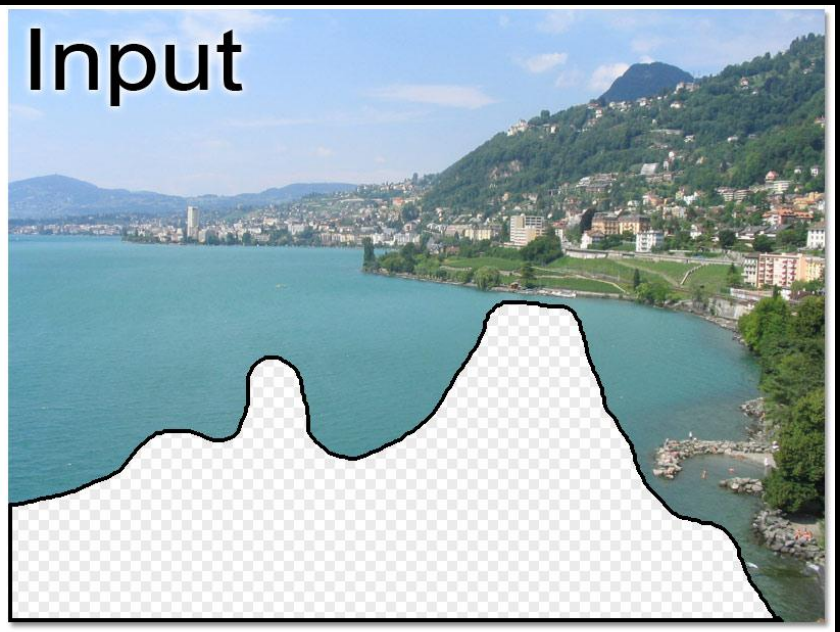
```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM TABLE2");
ResultSetMetaData rsmd = rs.getMetaData();
String name = rsmd.getColumnName(1);
```

and you can get the column name from there.

Challenge

Can we leverage the vast number of component usage examples to make it easier for programmers to write code using the component?





From http://graphics.cs.cmu.edu/projects/scene-completion/

Scene Completion

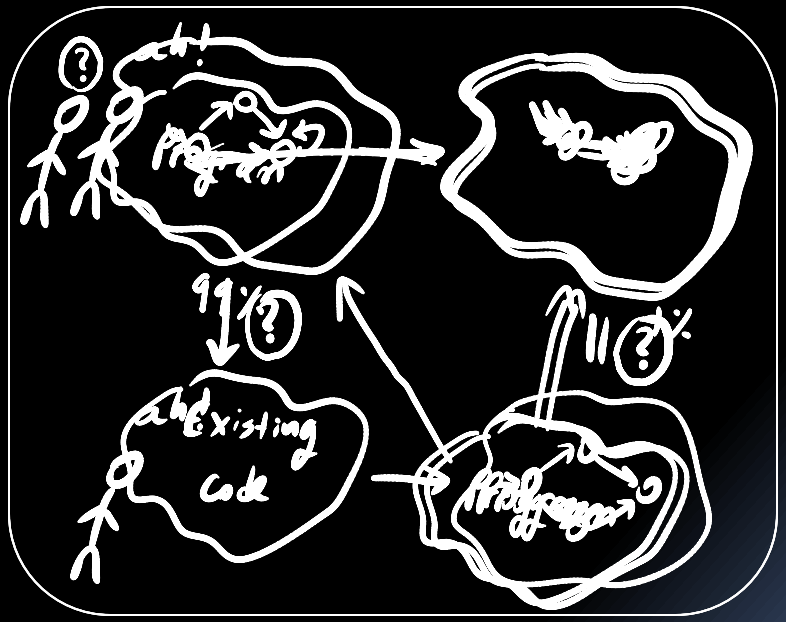
```
Connection c = new Conn();
???
ResultSet r = ???
while (?) {
  …
}
```

PRIME

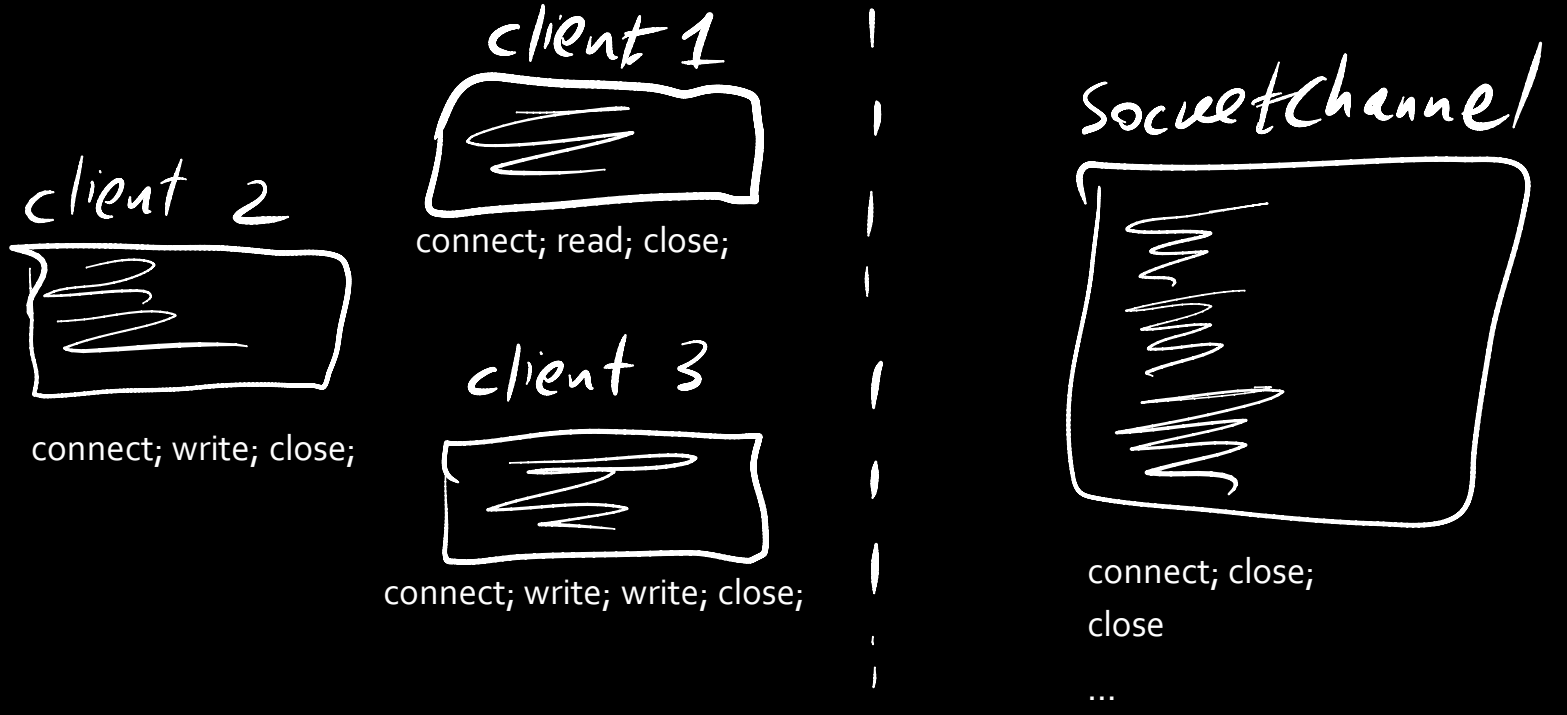
```
Connection c = new Conn();
Statement s = c.createStatement();
ResultSet r = s.executeQuery(…);
while (r.next()) {
  …
}
```

Mining Temporal Specifications

- Extract temporal specification from the program
- Applications
  - Program understanding
  - Regression
  - Deviant behaviors
  - Specs for verification
  - …



Approaches for Mining Temporal Specifications



Real usage scenarios << Permitted scenarios

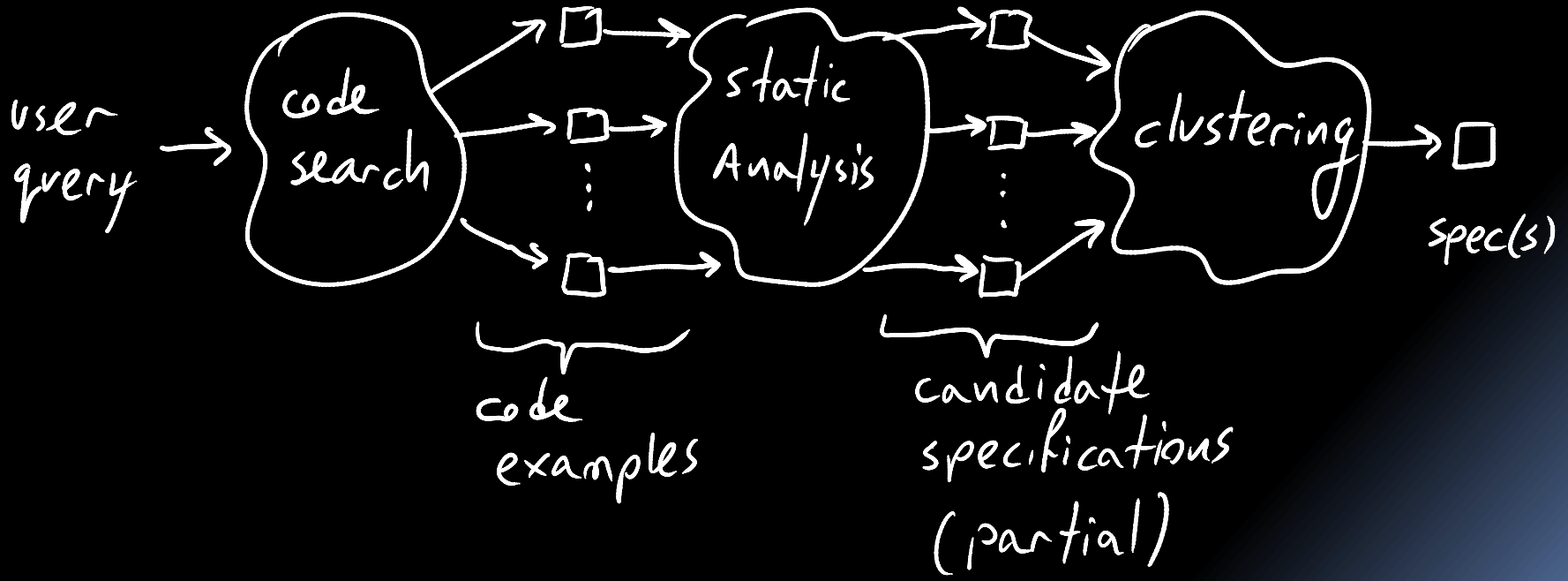
- Client-side mining
  - Infer usage from existing clients using the component
- Component-side mining
  - Infer usage from component implementation
  - Relies on error conditions in component implementation

Dynamic vs. Static Specification Mining

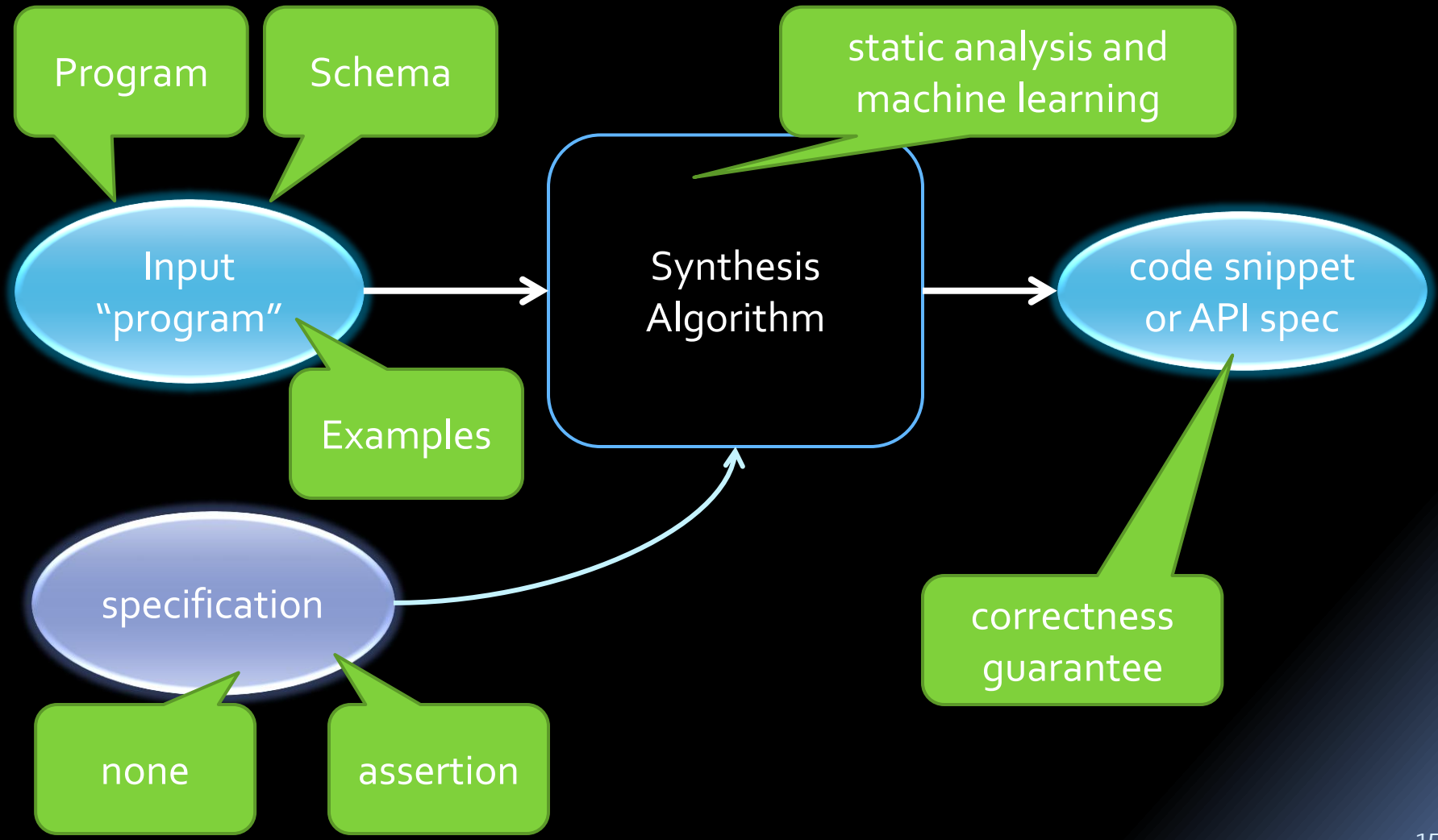
- Dynamic
  - Mine specification from representative executions
  - Requires running the program (with varying inputs)
  - Incomplete coverage of behaviors
- Static
  - Analyze the program without running it
  - Covers all client behaviors
- Reality: the amount of code available for inspection vastly exceeds the amount of code amenable to automated dynamic analysis

PRIME Approach

- Static client-side specification mining
- Bad news: this is hard
- Good news: we can still make it work

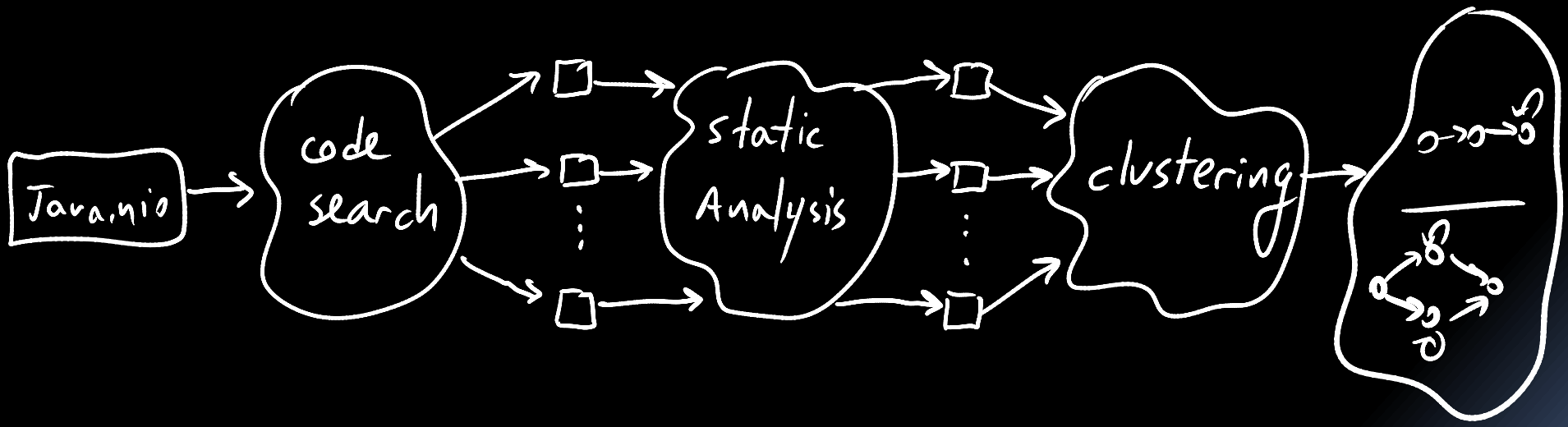


Dimensions of Synthesis



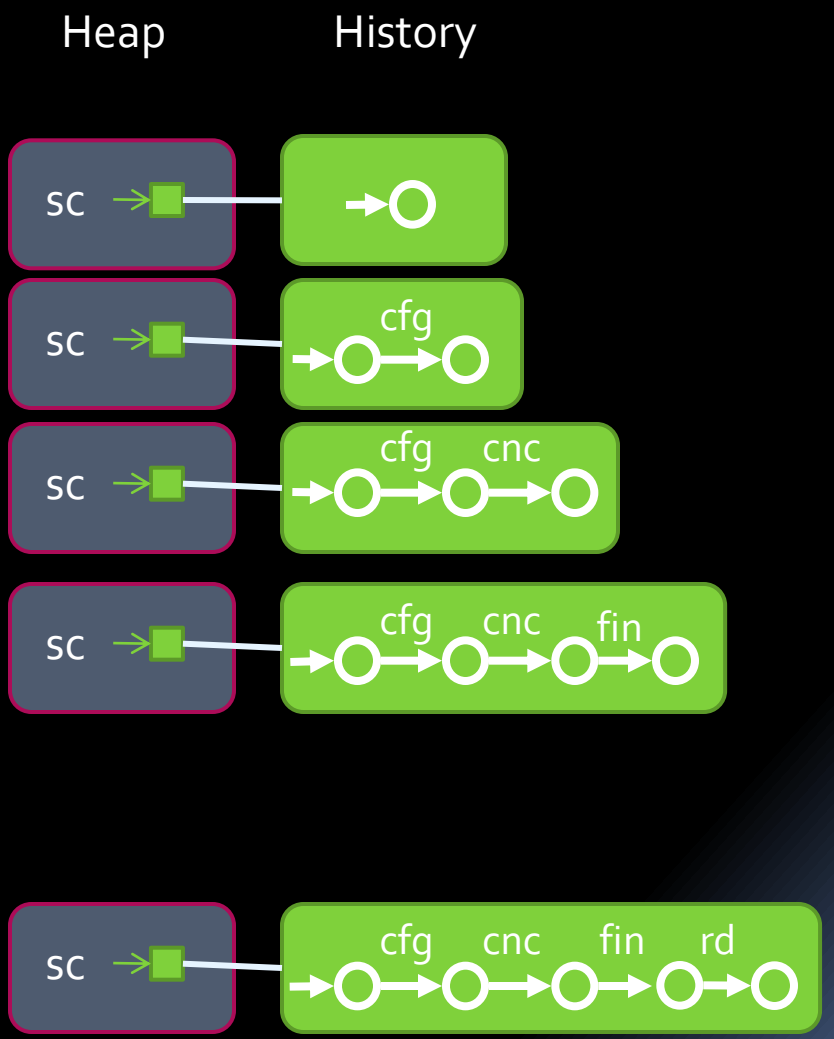
Example

How should I use a java.nio.channels.SocketChannel?



Analyzing a Single Code Sample

```
example1() {
  SocketChannel sc = SocketChannel.open();
  sc.configureBlocking(false);
 sc.connect();
 sc.finishConnect();
  ByteBuffer dst = …;
  sc.read(dst);
}
```



```
void example2() {
  Collection<SocketChannel> chnls = createChannels();
  for (SocketChannel sc : chnls){
    sc.connect(new ...);
    while (!sc.finishConnect()) { /* ... wait for connection ... */ }
    if (?) { receive(sc); } else { send(sc); }
  }
  closeAll(channels);
 }

Collection<SocketChannel> createChannels() {
  List<SocketChannel> list = new LinkedList<SocketChannel>();
  list.add(createChannel(" ", 80));
  //... more channels added to list ...
  return list;
 }

SocketChannel createChannel (String hostName, int port) {
  SocketChannel sc = SocketChannel.open();
  sc.configureBlocking(false);
  return sc;
 }
```

```
void example2() {
  Collection<SocketChannel> chnls = createChannels();
  for (SocketChannel sc : chnls){
    sc.connect(new ...);
    while (!sc.finishConnect()) { /* ... wait ... */ }
    if (?) { receive(sc); } else { send(sc); }
  }
  closeAll(channels);
}

void receive(SocketChannel x) {
  //...
  FileOutputStream fos = new ...;
  ByteBuffer dst = ...;
  int numBytesRead = 0;
  while (numBytesRead >= 0) {
    numBytesRead = x.read(dst);
    fos.write(dst.array());
  }
  fos.close();
}

void send(SocketChannel x) {
  for (?) {
    int numWritten = x.write(buf);
  }
}

void closeAll(Collection<SocketChannel> chnls) {
  for (SocketChannel sc : chnls) {
    sc.close();
  }
}
```

Bad News

Partial Programs

Unbounded Number of Objects

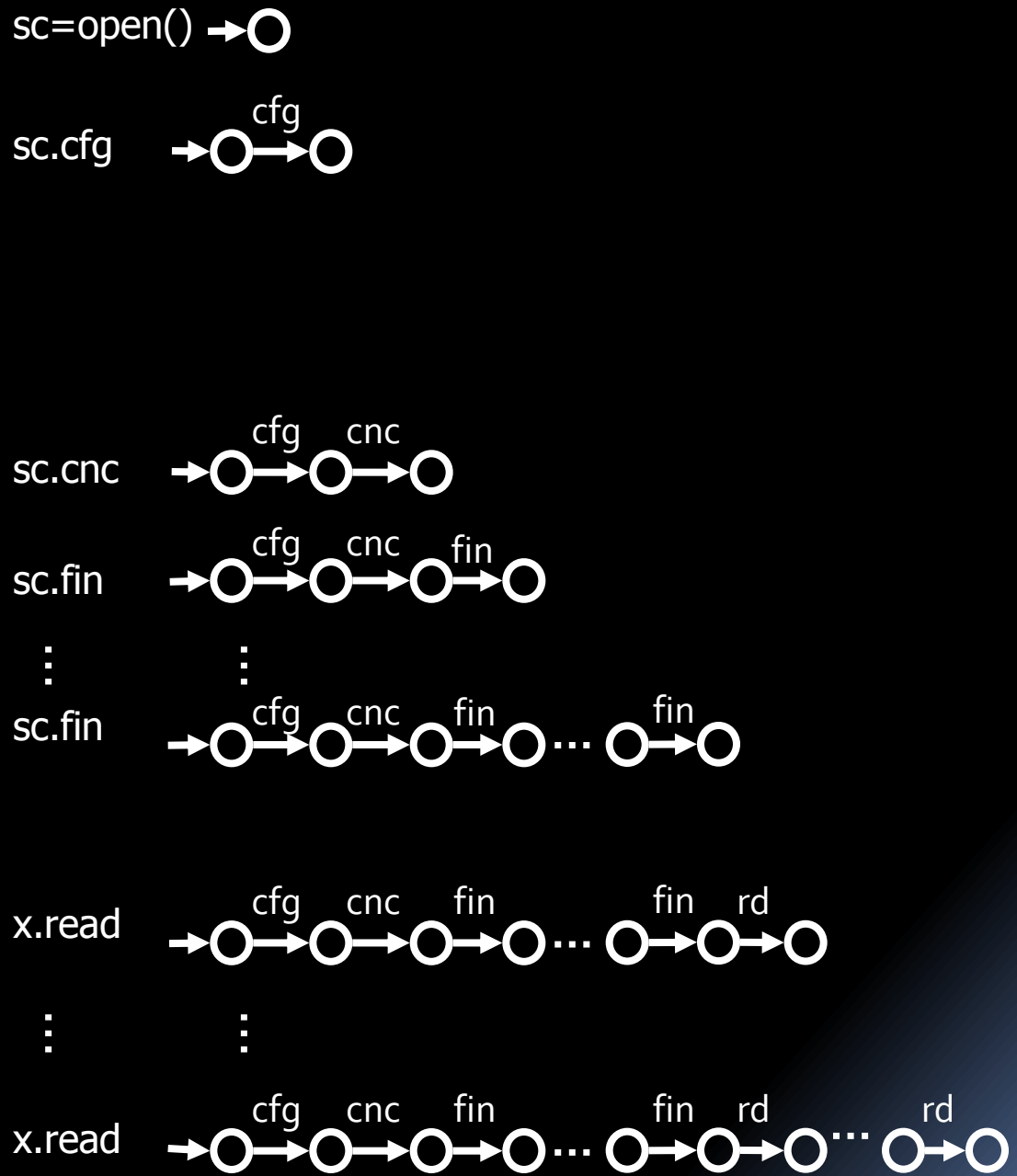
Non-trivial aliasing

Interprocedural Flow

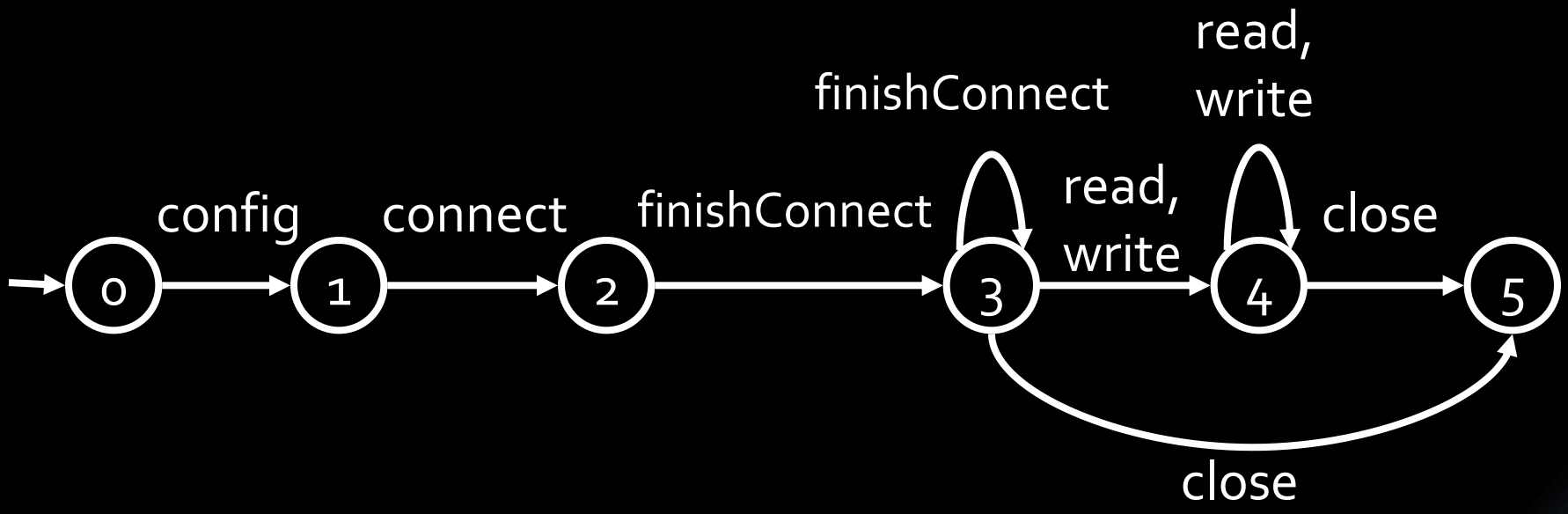
Flow Sensitivity

Context Sensitivity

```
SocketChannel createChannel (...)
{
 SocketChannel sc =
     SocketChannel.open();
 sc.configureBlocking(false);
 return sc;
}

void example() {
 Collection<SocketChannel> chnls =
     createChannels();
 for (SocketChannel sc : chnls){
   sc.connect(new ...);
   while (!sc.finishConnect()) { ... }
   if (?) { receive(sc); }
     else { send(sc); }
 }
 closeAll(channels);
}


void receive(SocketChannel x) {
 ...
 while (numBytesRead >= 0) {
   numBytesRead = x.read(dst);
   fos.write(dst.array());
 } ...
}}
```



SocketChannel Specification



(Partial specification)

Challenges

- Partial programs
  - Program fragments
  - Missing information

  ➔ Support mining specifications with partial information

- Dynamically allocated objects
  - unbounded number of objects
  - aliasing
  - objects flow through complex heap-allocated data structures

  ➔ heap abstraction

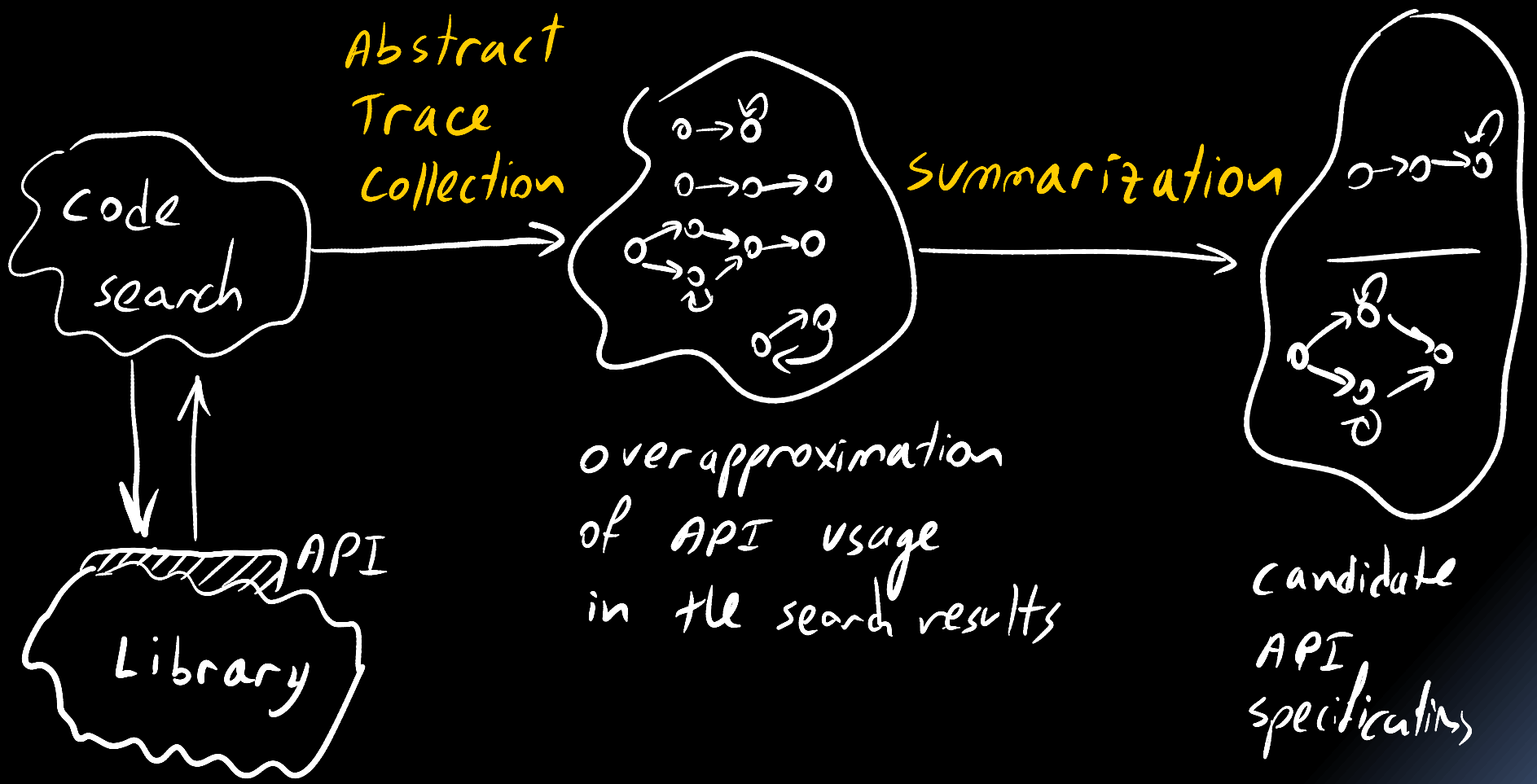
- Unbounded length of histories
  - History (event sequence) observed for an object might be unbounded

  ➔ history abstraction

- Noise
  - analysis imprecision and/or incorrect client programs

  ➔ Noise reduction

Overview

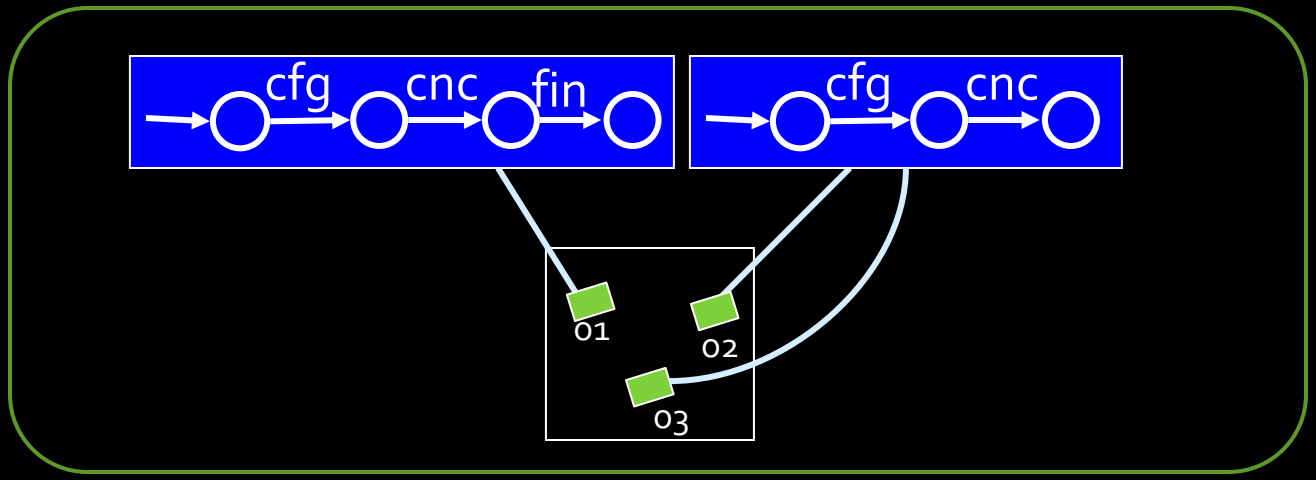

Abstract
Trace
Collection
Code
search
API
Library
overapproximation
of API usage
in the search results
summarization
candidate
API
specifications

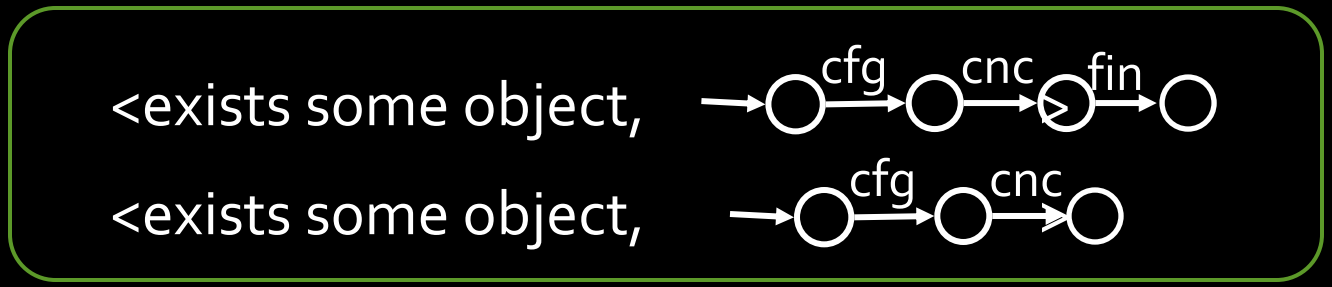
Abstract Trace Collection

- The French Recipe for Abstract Interpretation [Cousot&Cousot77]

- Abstraction
  - Abstract state provides a bounded description of possible program states at a program point
- Abstract Transformers
  - Conservatively represent the effect of statements on abstract states
- Exploration
  - Compute the possible abstract states at each program point by fixed-point iteration

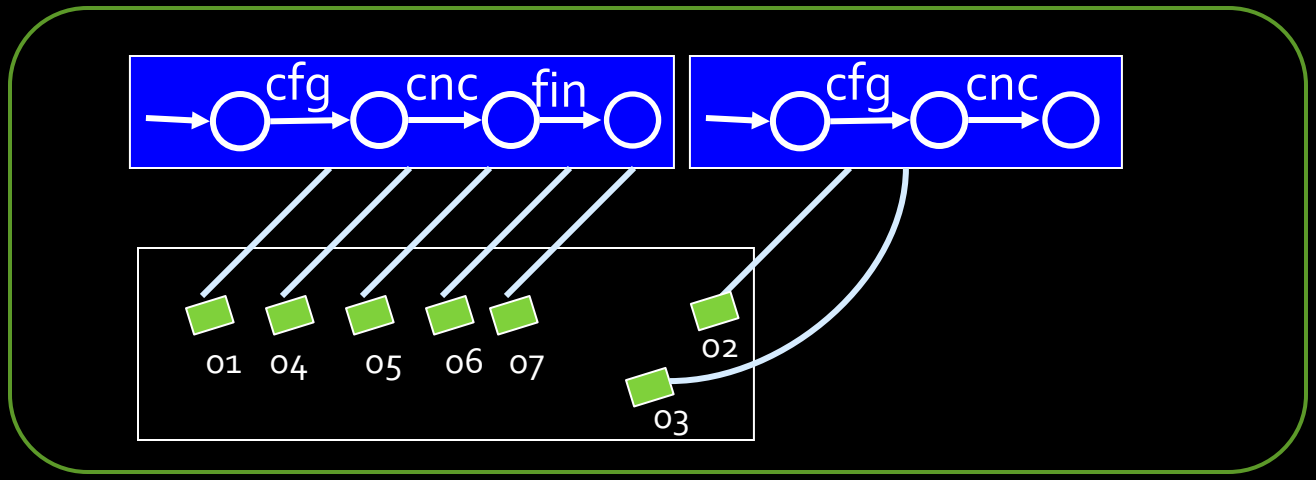
Abstraction

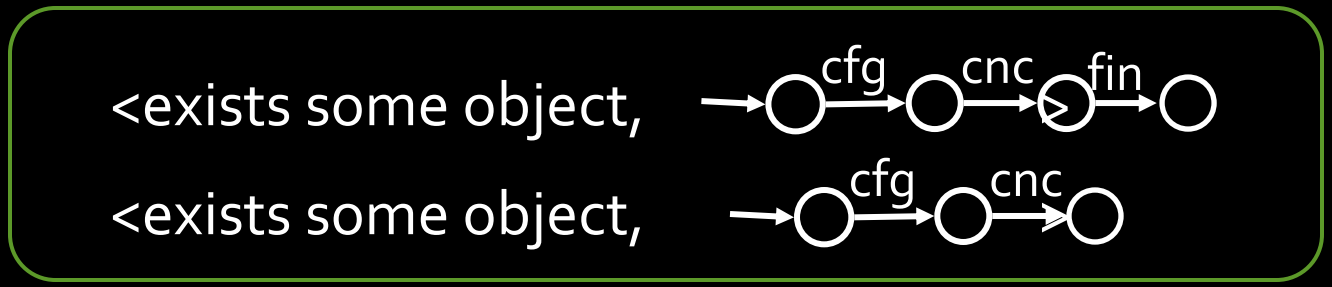




- Abstract state is a set of abstract values (disjunction)
- Abstract value is a pair (conjunction)
  - Heap abstraction: abstracts unbounded heap
  - History abstraction: abstracts unbounded sequences of operations

Abstraction

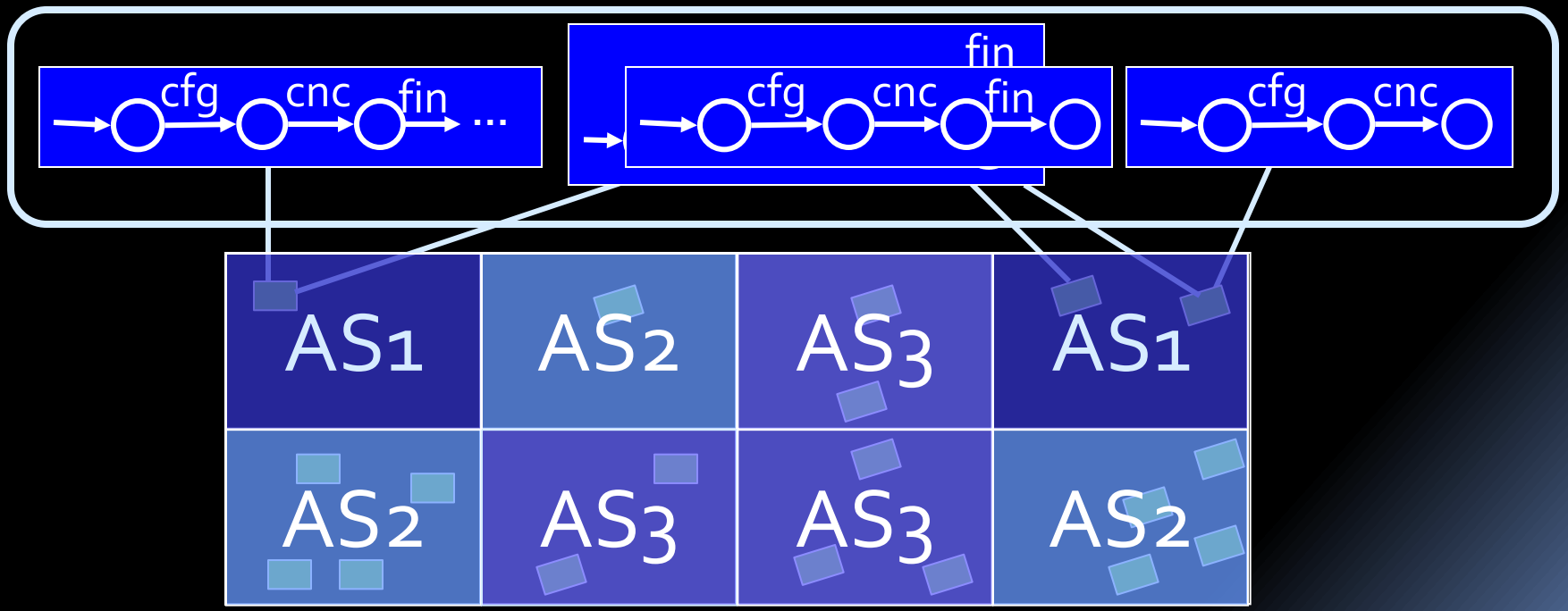




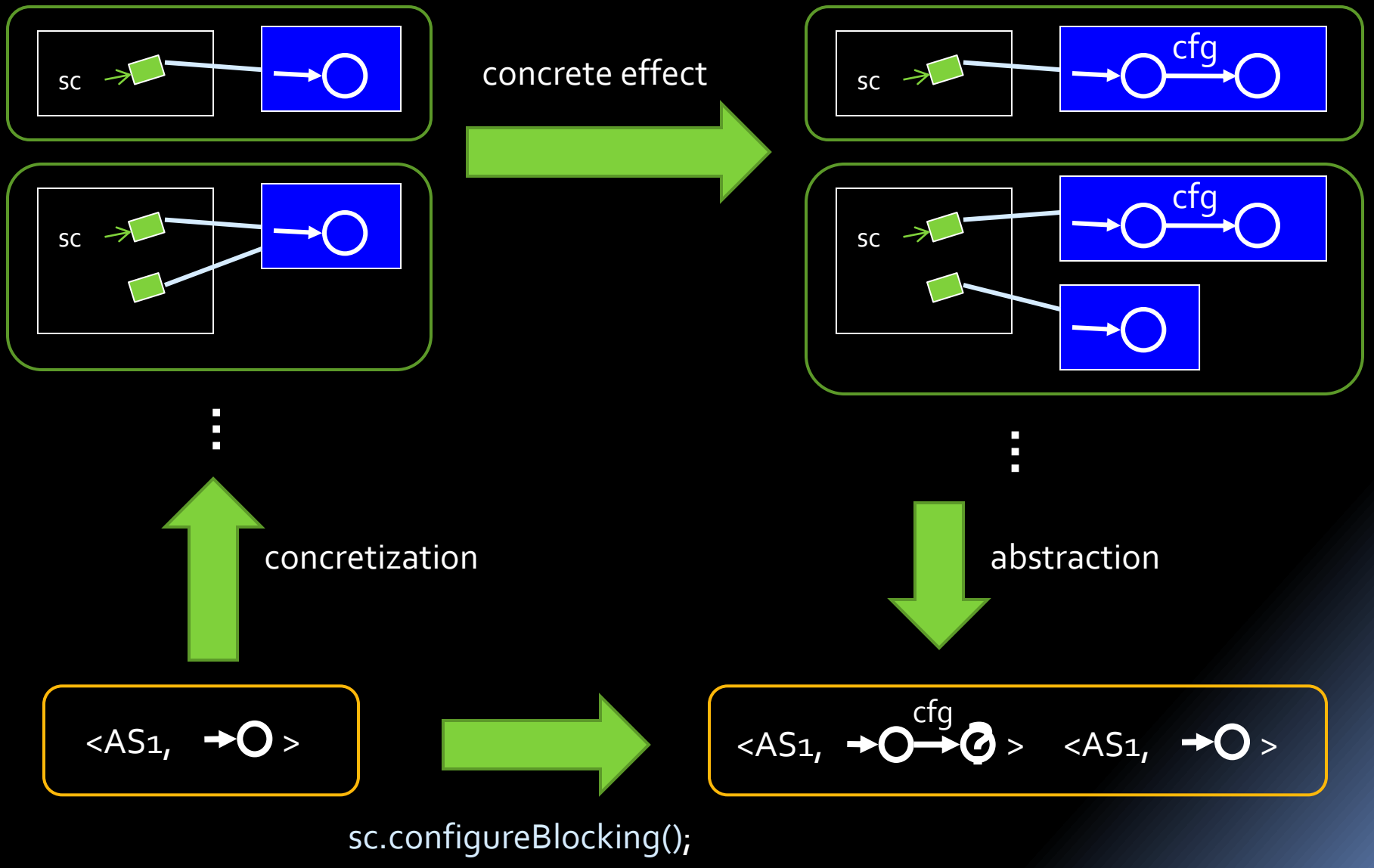
- Abstract state is a set of abstract values (disjunction)
- Abstract value is a pair (conjunction)
  - Heap abstraction: abstracts unbounded heap
  - History abstraction: abstracts unbounded sequences of operations

Heap Abstraction – Take I

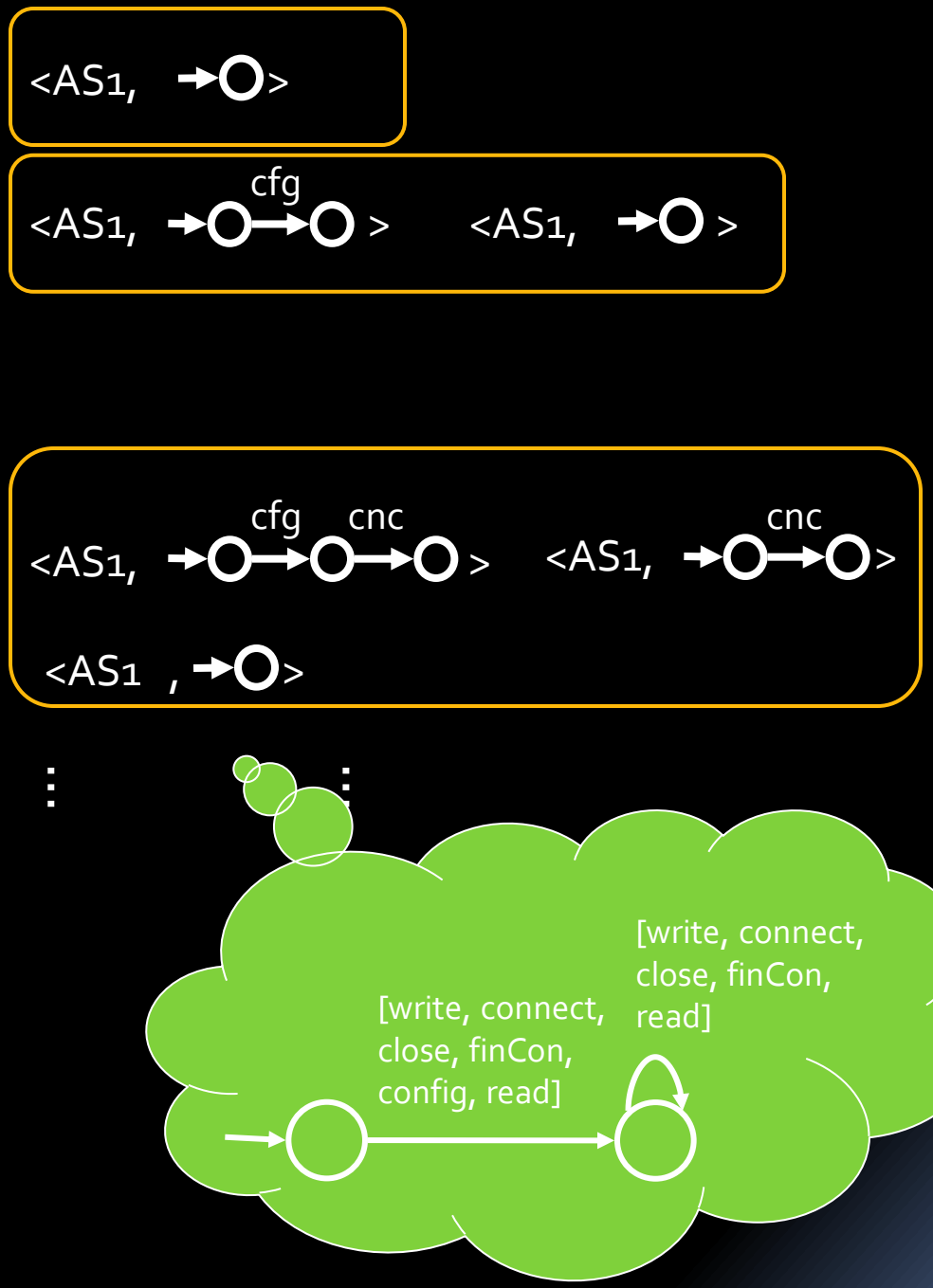
- Divide the heap into a fixed partition based on allocation site
- All objects allocated at the same program point represented by a single "abstract object"



Abstract Transformers – Take I

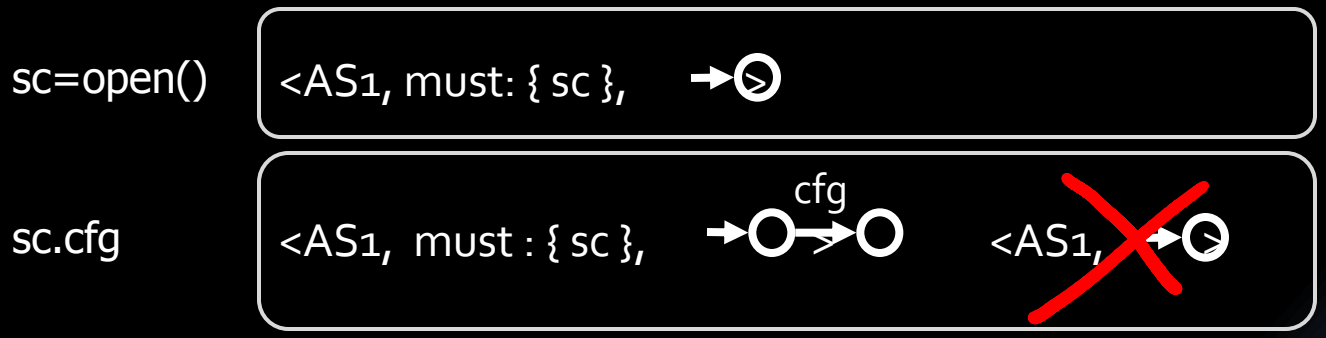


```
SocketChannel createChannel (…)
{
 SocketChannel sc =
      SocketChannel.open(); // AS1
 sc.configureBlocking(false);
 return sc;
}

void example() {
 Collection<SocketChannel> chnls =
      createChannels();
 for (SocketChannel sc : chnls){
   sc.connect(new …);
   while (!sc.finishConnect()) { … }
   if (?) { receive(sc); }
      else { send(sc); }
 }
 closeAll(channels);
}


void receive(SocketChannel x) {
 …
 while (numBytesRead >= 0) {
   numBytesRead = x.read(dst);
   fos.write(dst.array());
 } …
}}
```



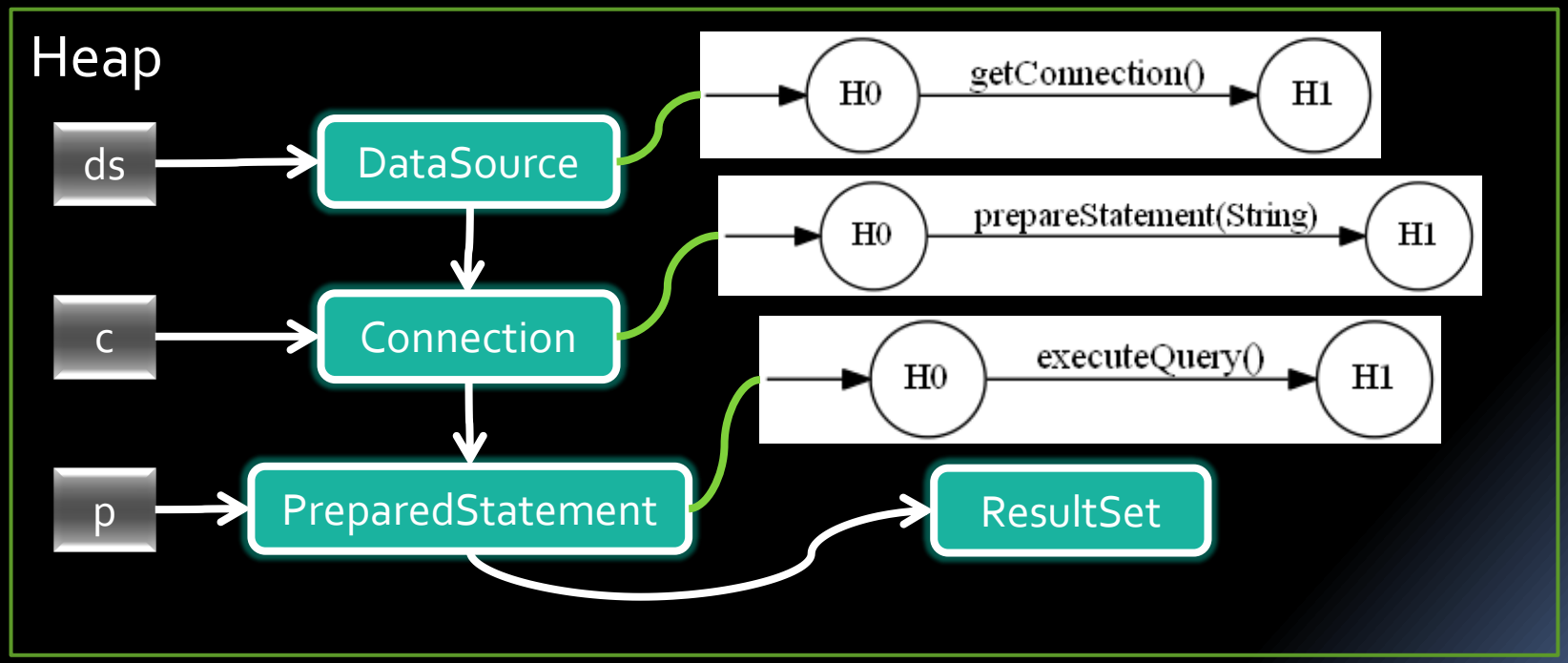
Refined Heap Abstraction

- Heap data for an "abstract object" o
  - unique = true
    - abstract value represents a single object
  - must = {x.f}
    - the access path x.f must point to o
  - mustNot = {y.g}
    - the access path y.g must not to point to o
  - …
- Dynamic partition
- Must points-to information allows strong updates



Objects Are Related

```
public ResultSet realLifeCreateResultSet(name) {
  DataSource ds = ConnectionFactory.createConnectionFactory();
  Connection c = ds.getConnection();
  PreparedStatement p = c.prepareStatement(
    "select * from " + name);
  return p.executeQuery();
}
```



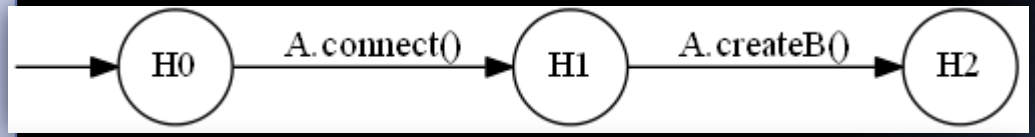
Maintaining (some) Object Relations

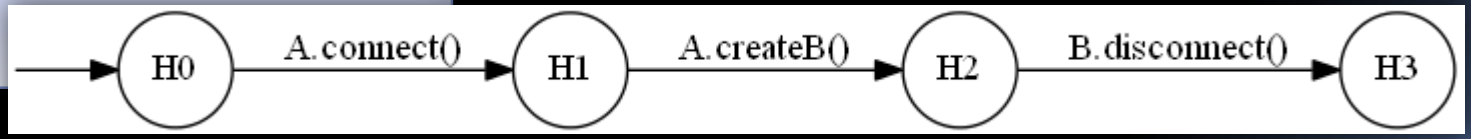
Ignore relations

Maintain creation context

Full heap analysis

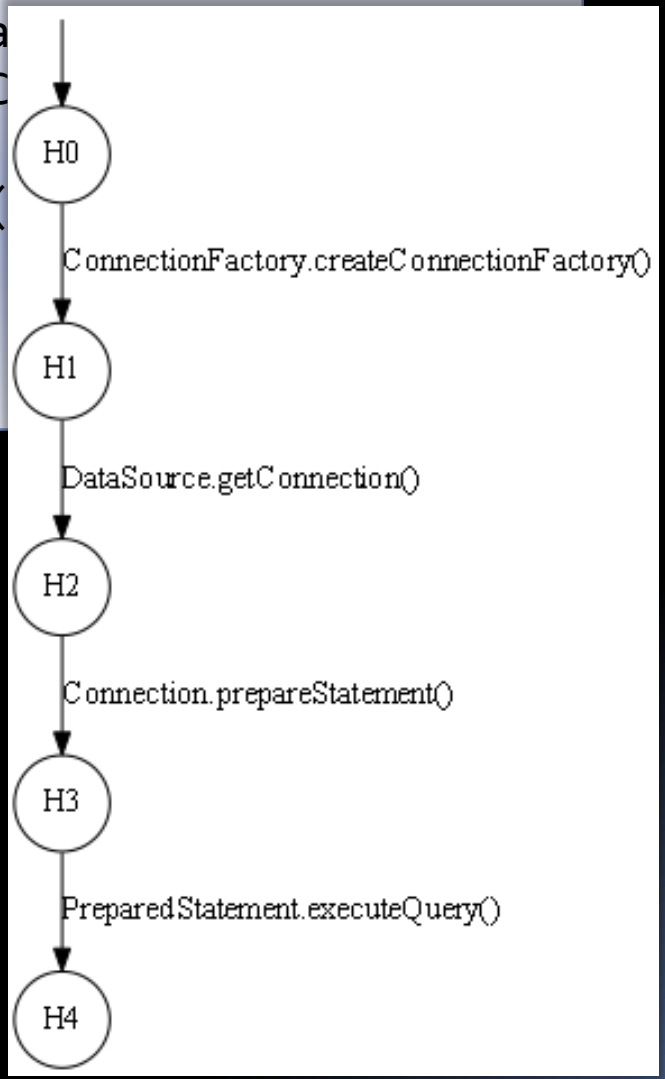
```
public void method1(A a) {
  a.connect();
  B b = a.createB();
  b.disconnect();
}
```



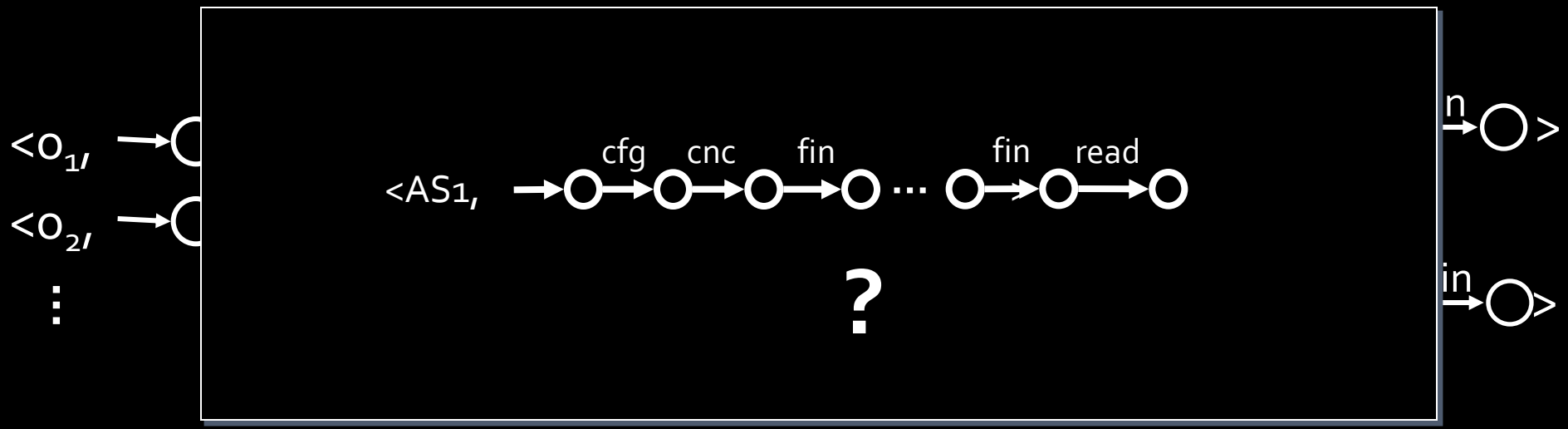


Maintaining (some) Object Relations

```
public ResultSet realLifeCreateResultSet(na
  DataSource ds = ConnectionFactory.createC
  Connection c = ds.getConnection();
  PreparedStatement p = c.prepareStatement(
    "select * from " + name);
  return p.executeQuery();
}
```



History Abstraction

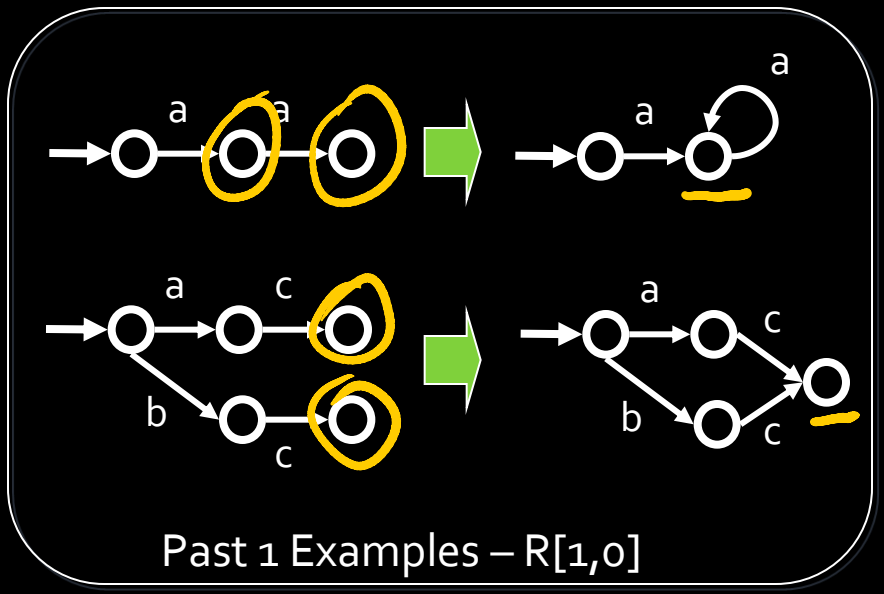


- Abstract history
  - Automaton over-approximating unbounded event sequences
- Quotient-based abstractions for history
  - Automata states which are equivalent w.r.t. a given equivalence relation R are merged

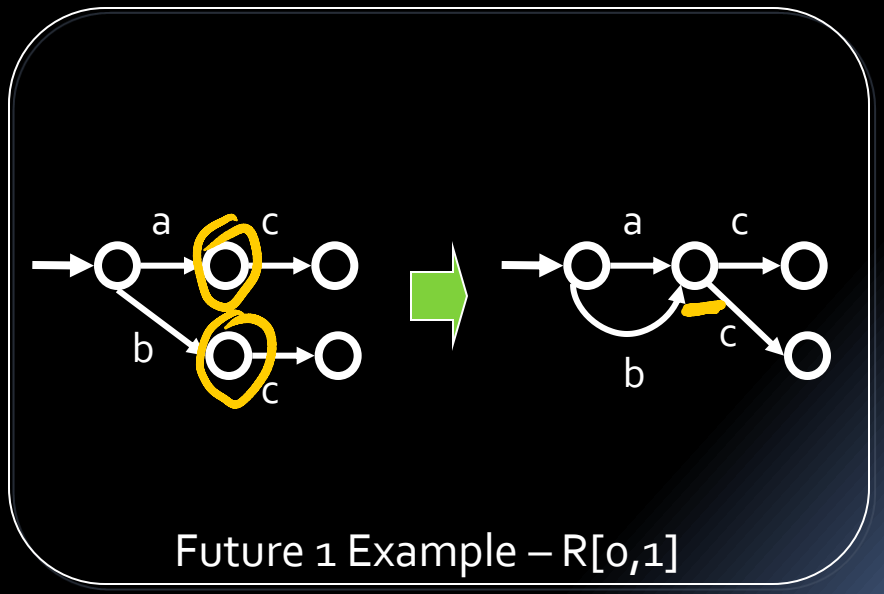
History Abstraction

- Past-Future Abstraction

  $(q_1, q_2) \in R[k_{in}, k_{out}]$ if q_1 and q_2 share both an incoming sequence of length k_{in} and an outgoing sequence of length k_{out}



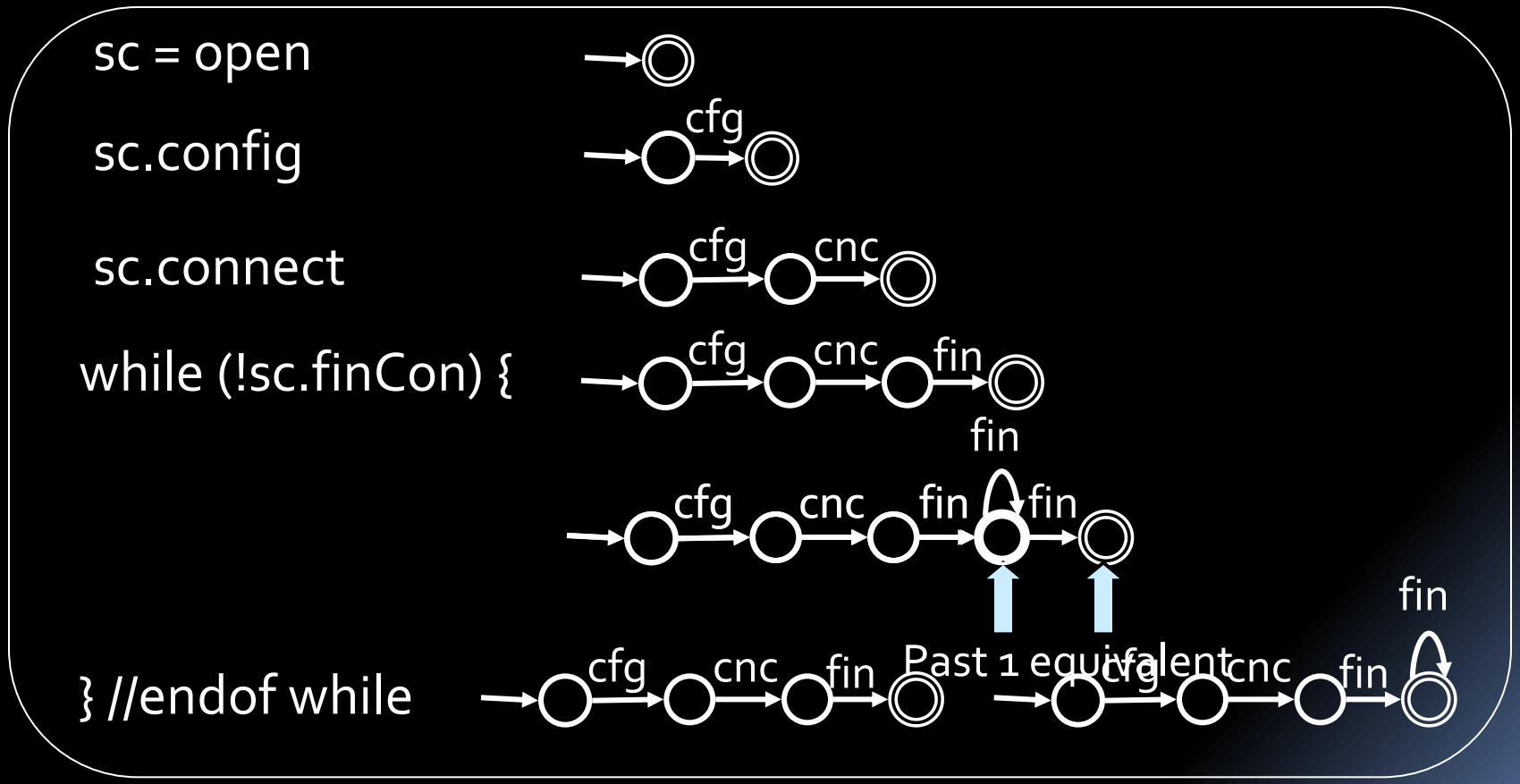
Past 1 Examples – R[1,0]



Future 1 Example – R[0,1]

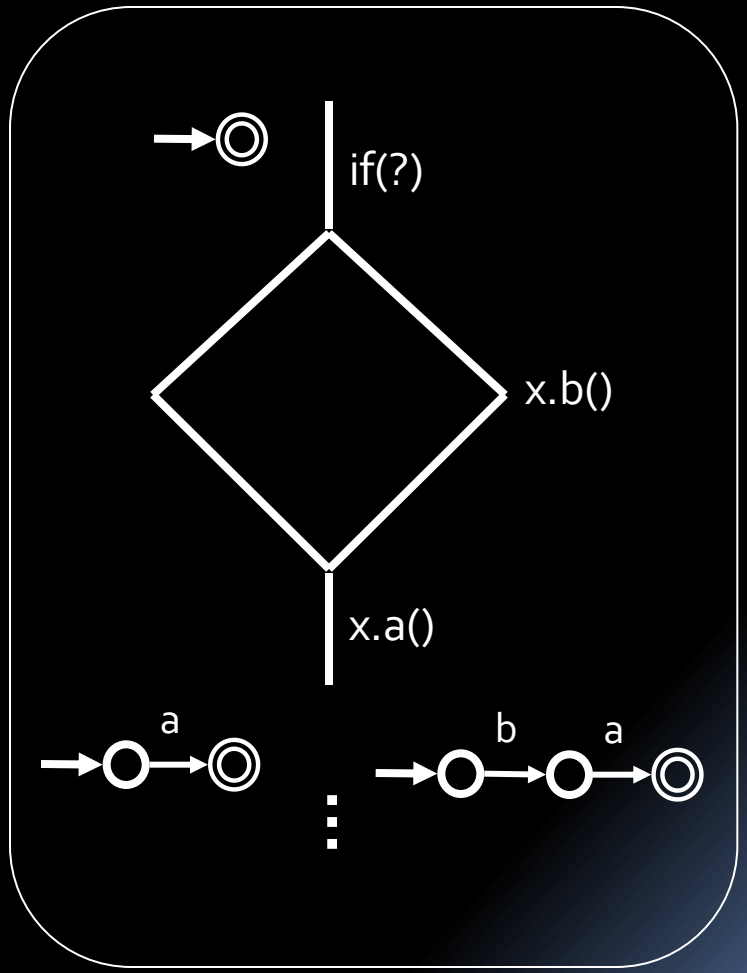
Abstract Semantics

- Initial abstract history
  - empty sequence automaton
- When an API method is invoked
  - history extended: append event and construct quotient



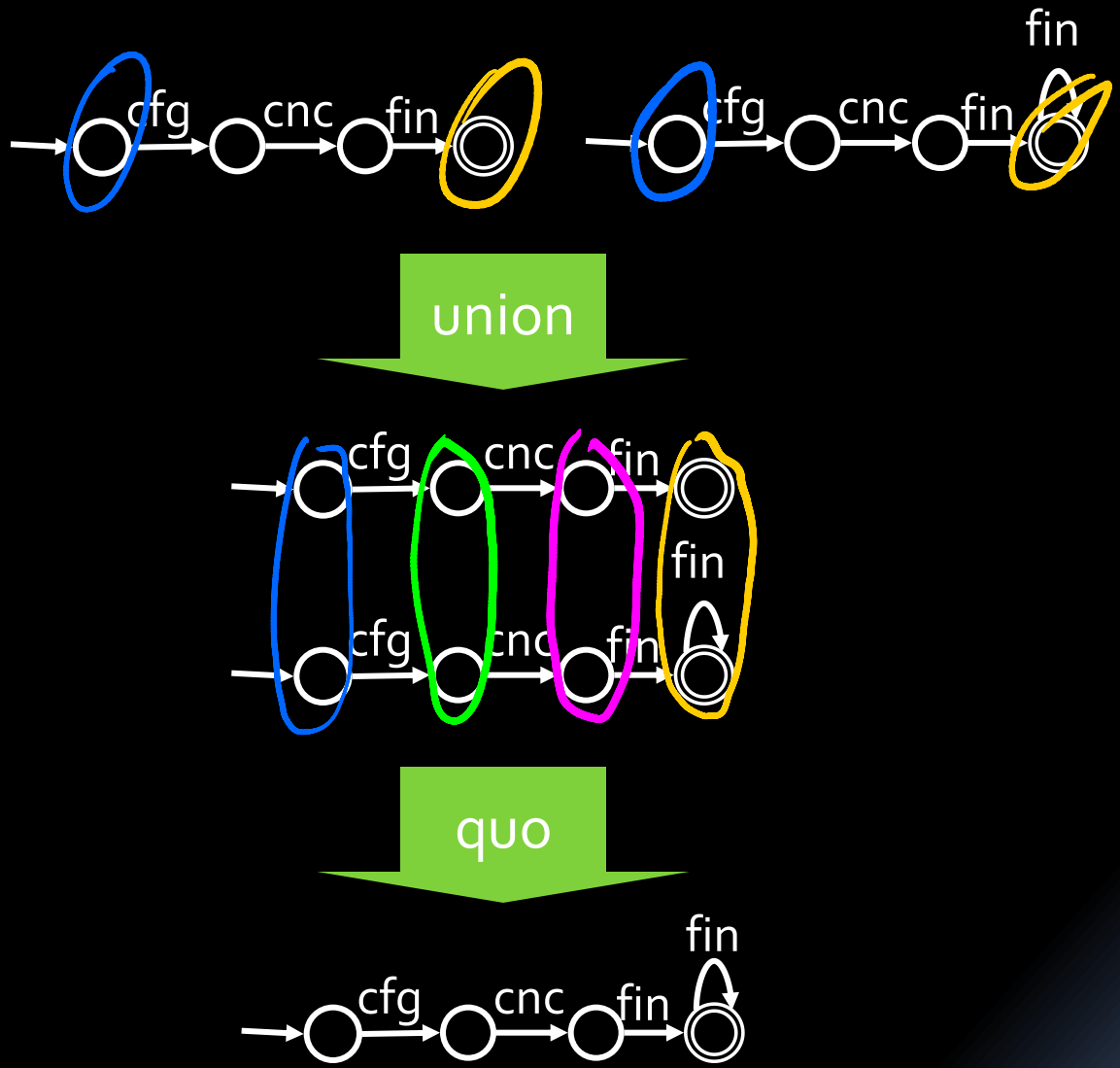
Are We Done?

- Bounded is great, but not enough
- Merge histories at control flow join points
  - Speed up convergence
- Merge all histories that
  - have identical heap-data, and
  - satisfy a given merge criterion
- Merge: union construction followed by quotient construction



Example: Past Abstraction with Exterior Merge

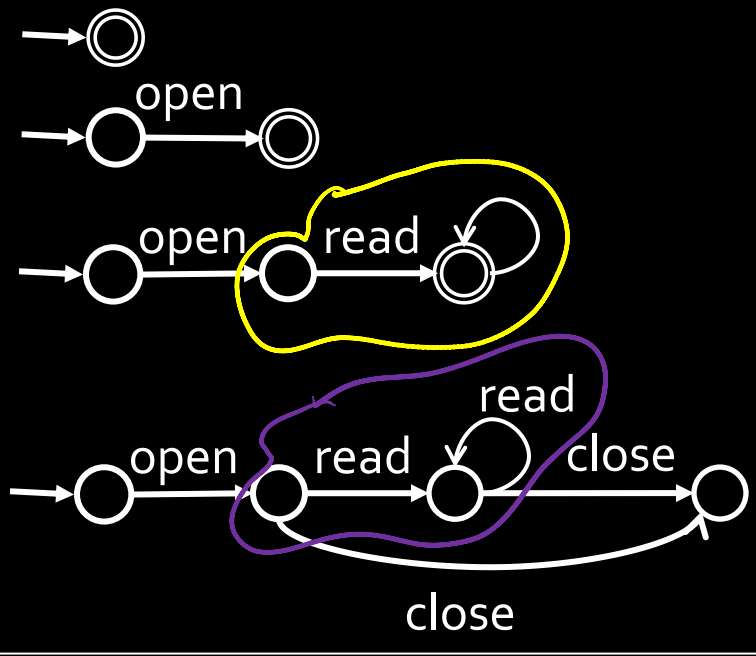
endof while



Dealing with the Unknown

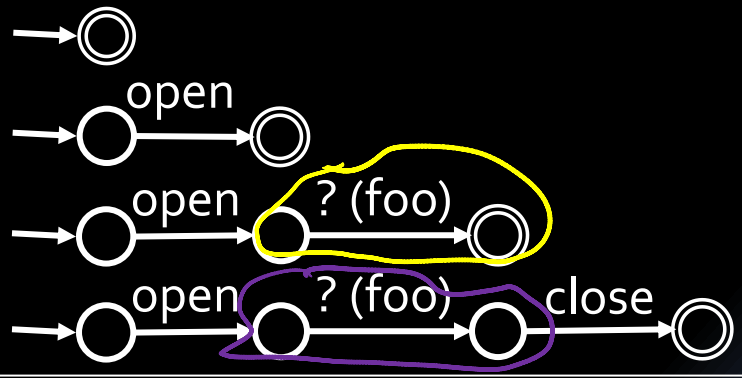
example1

```
FileComponent fc
    = new FileComponent();
fc.open();
while(?) {
 fc.read();
}
fc.close();
```

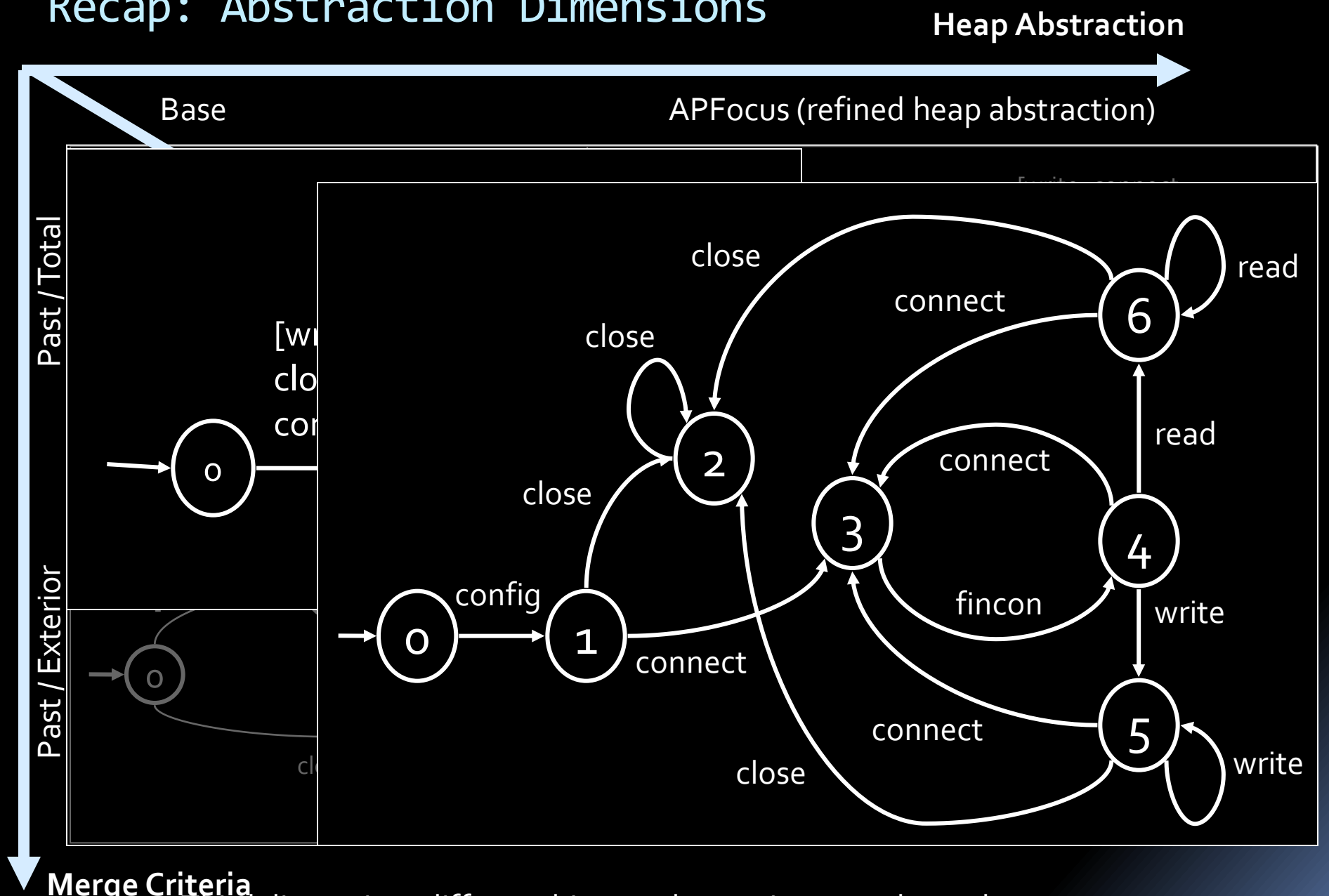


example2

```
FileComponent fc
    = new FileComponent();
fc.open();
foo(fc);
fc.close();
```

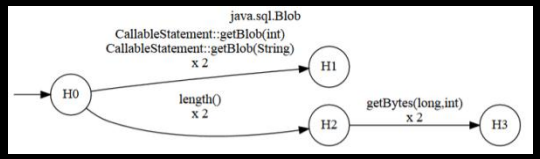


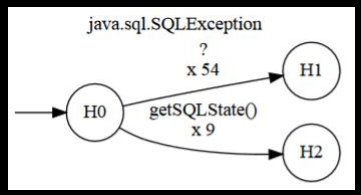
Recap: Abstraction Dimensions

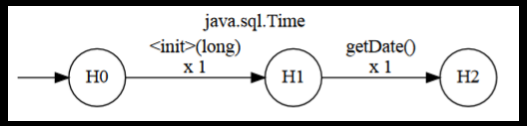


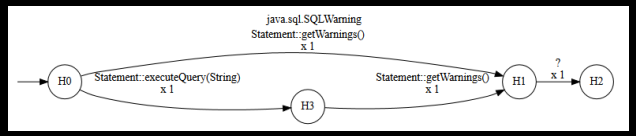
Third dimension: different history abstraction, not shown here

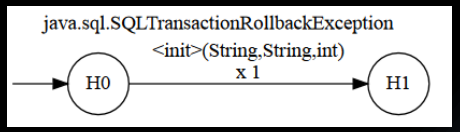
Using the Analyzed Samples

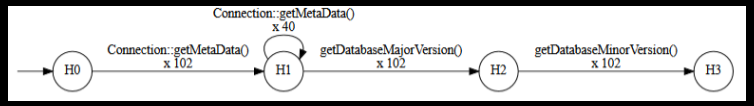




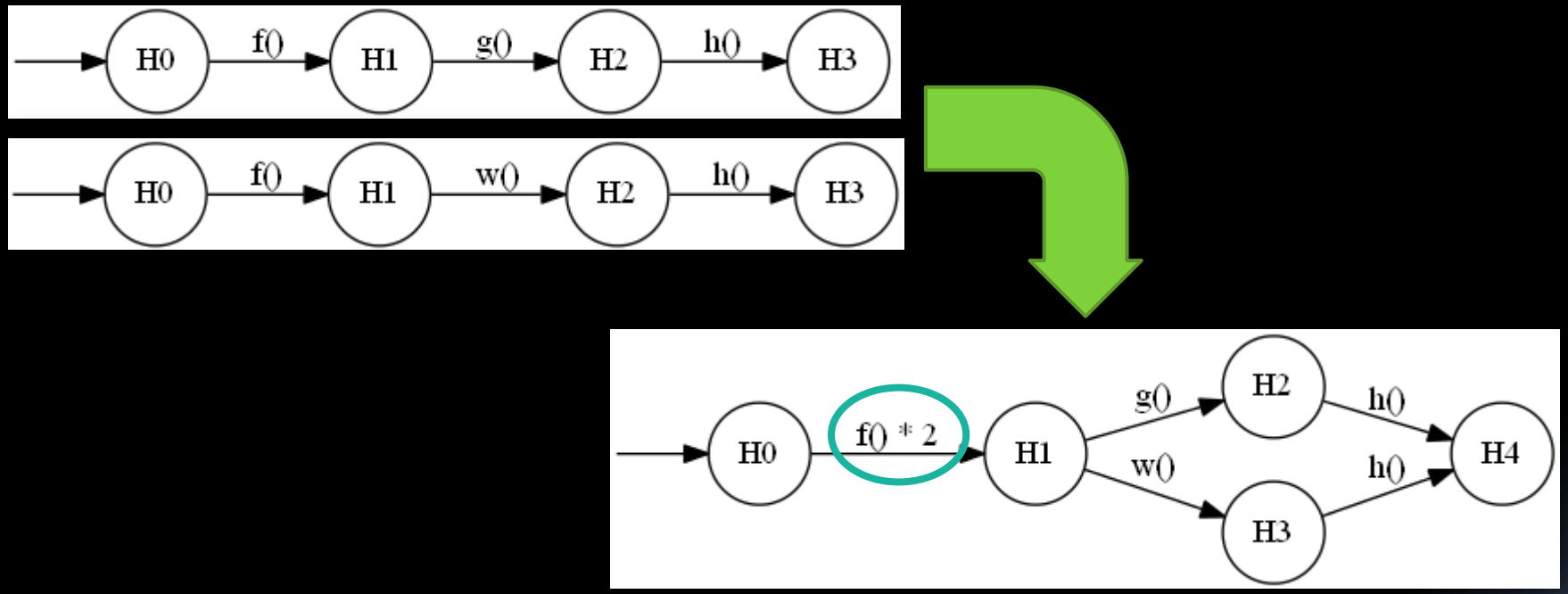




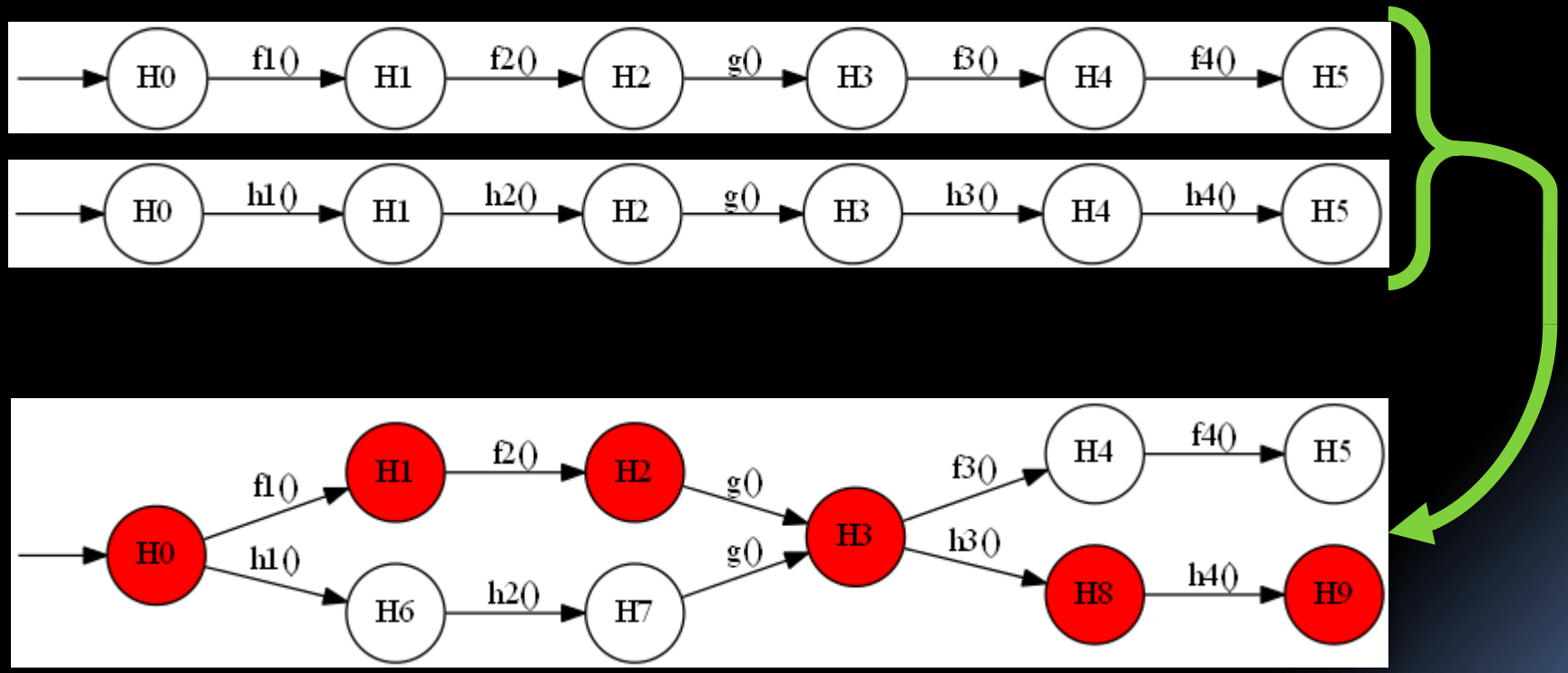




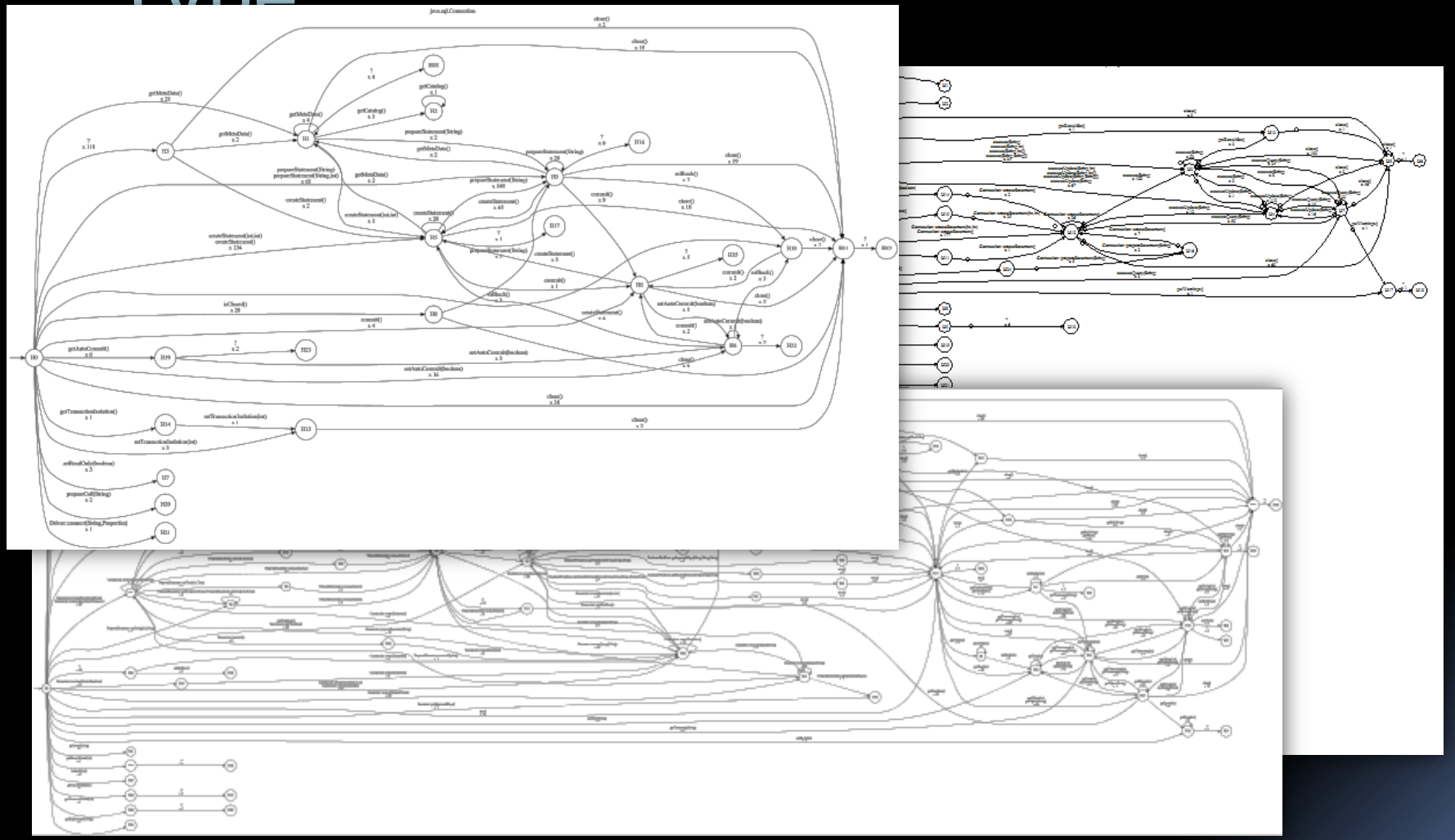
Merge Same Type Together



Merge All Samples of Same Type

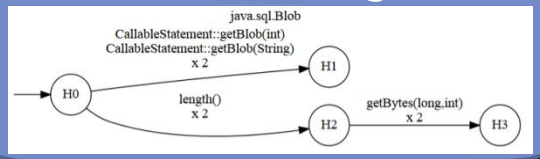


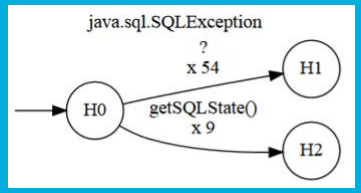
Merging all Samples of Same Type



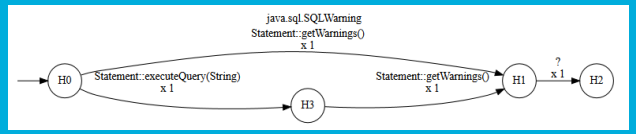
Merge by Use Case

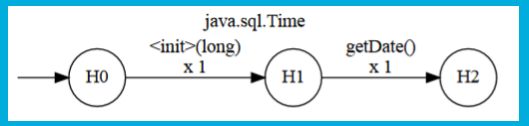
Use case 3



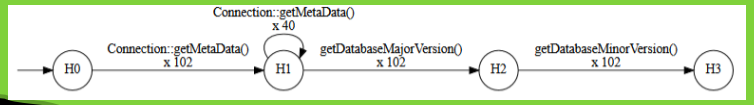


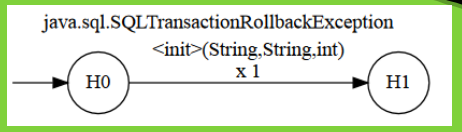
Use case 1





Use case 2

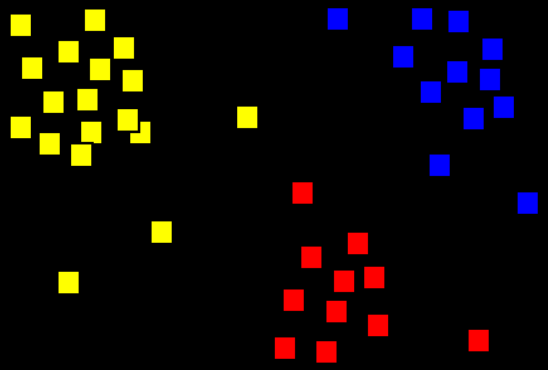




But how?

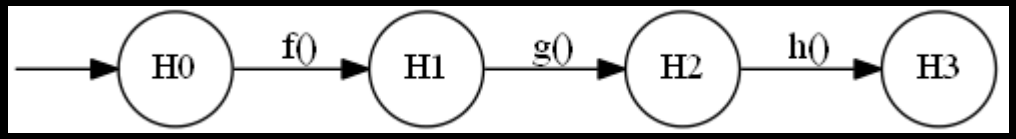
Clustering

We define a distance function between samples, then use classic clustering techniques from data mining.

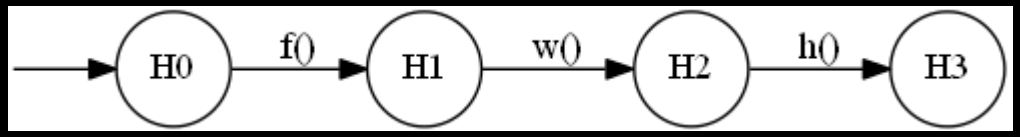


Clustering: Distance Function

Different use-cases typically use different methods



f()	g()	w()	h()
1	1	0	1



f()	g()	w()	h()
1	0	1	1

Distance = $\sqrt{(1-1)^2+(1-0)^2+(0-1)^2+(1-1)^2}$ = ~1.41

Clustering Results

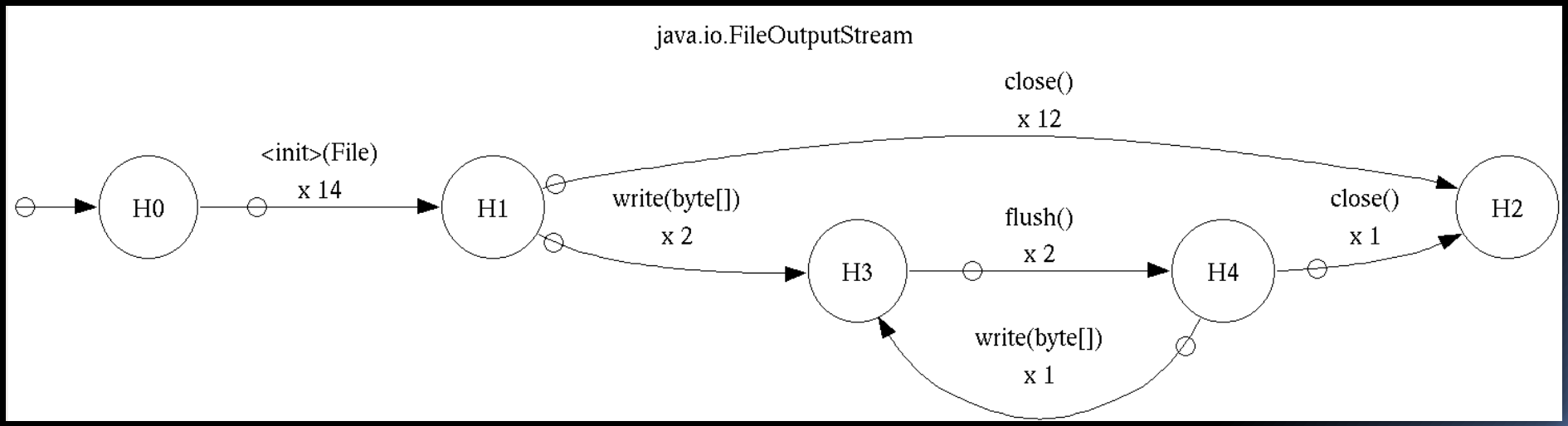
	# samples	% samples	size
java.sql.Statement	396	18	40
Statement cluster 4	239	60	5
Statement cluster 6	77	19	7
Statement cluster 1	32	8	28
Statement cluster 5	27	7	5
Statement cluster 0	9	2	8
Statement cluster 3	6	2	6
Statement cluster 2	6	2	7
java.sql.Timestamp	62	3	22
Timestamp cluster 1	29	47	3
Timestamp cluster 2	13	21	9
Timestamp cluster 0	7	11	7
Timestamp cluster 4	7	11	10
Timestamp cluster 3	6	10	12

Implementation Details

- Sample collection is multithreaded
- Specialized partial compiler for analyzing fragments
- Weka for clustering
- Inter-procedural analysis:
  - Dealing with parameters and return values
  - Dealing with recursion (both direct and indirect)
- Optimizing history representation for scalability
- Integrating with Eclipse to provide GUI

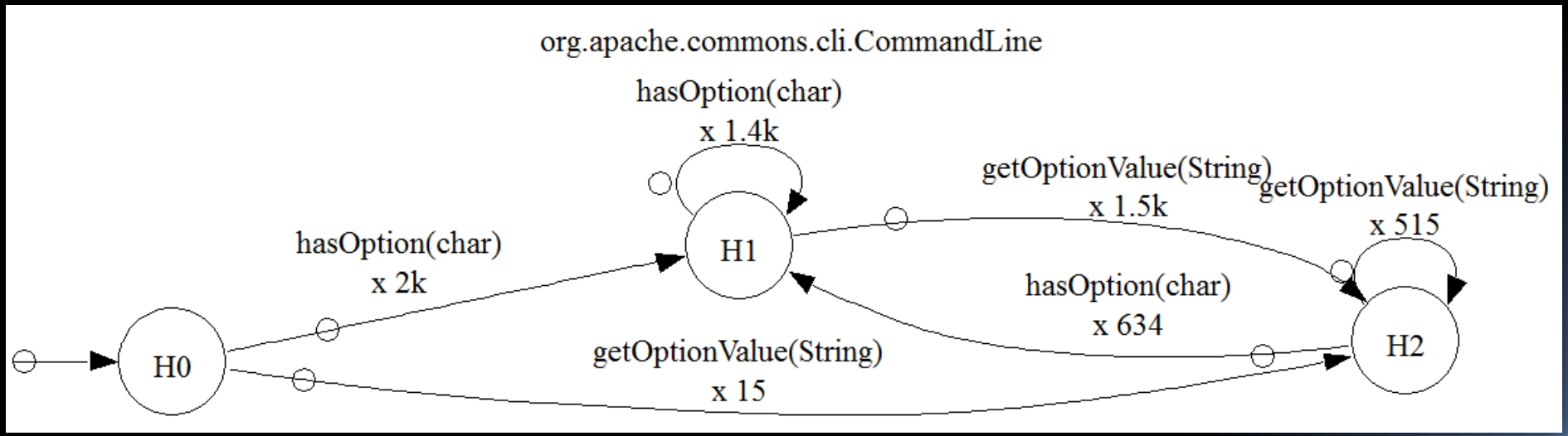
Results

- FileOutputStream is the common way to write to a file in Java.
- Even a few samples generate this cluster:



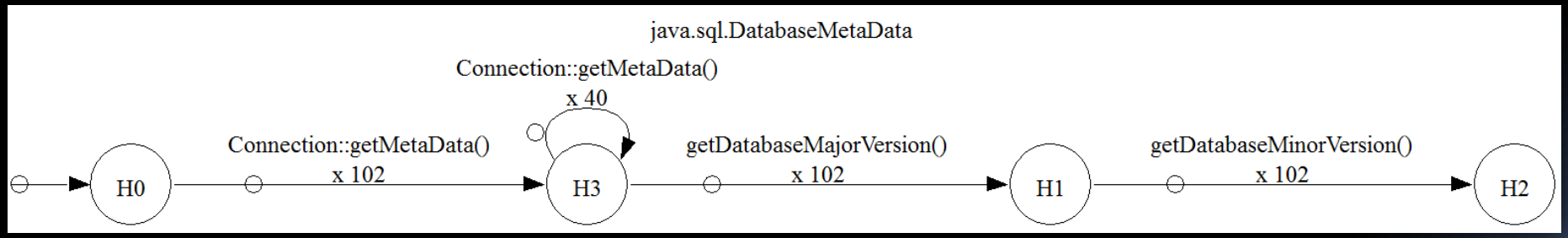
Results

- "Apache Commons CLI" is a library used for command-line parsing.
- 80% of samples belong to the CommandLine class, and 74% of CommandLine's samples got clustered together into:



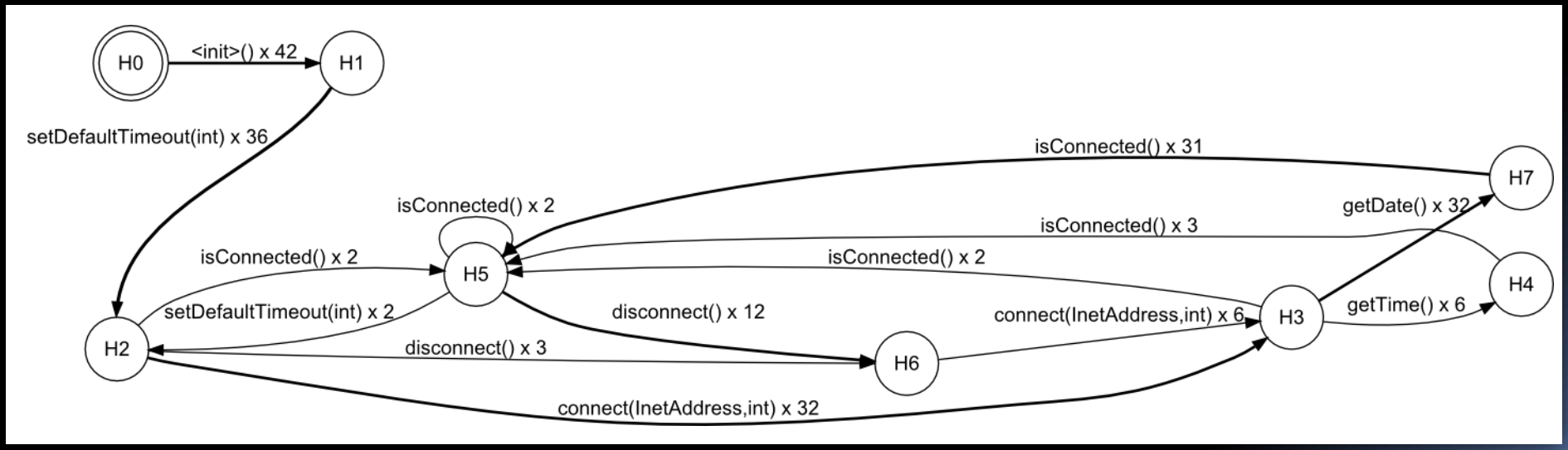
Results

- "JDBC", our pet example, is used for accessing SQL databases.
- 88% of samples from java.sql.DatabaseMetaData got clustered together into:



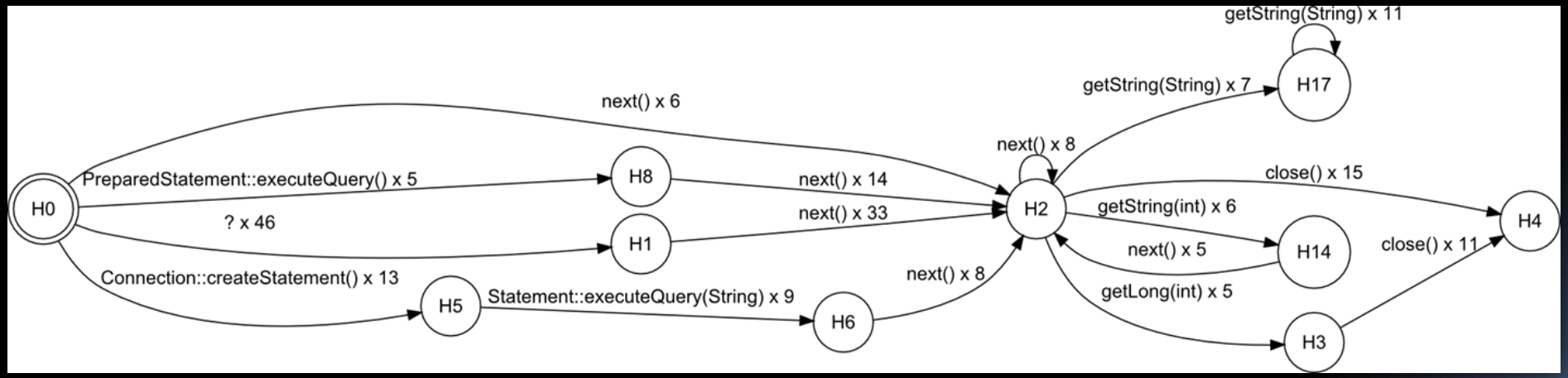
Results

- "Apache Commons Net", client implementations for many net protocols
- Analysis results for org.apache.commons.net.time.TimeTCPClient class:



Results

- "JDBC" once again
- Analysis results for java.sql.ResultSet class:



Future Applications

```
public void method1() {
  File f = new File();
  f.?;
  f.write(buffer);
  f.?;
}
```


```
public void method1() {
  File f = new File();
  f.open();
  f.write(buffer);
  f.flush();
  f.close();
}
```

```
public void method1() {
  File f = new File();
  f.open();
  f.close();
  f.write(buffer);
}
```

WARNING: java.io.File is only used this way in 0.5% of samples

Future Applications

- Stack-overflow for API usage
  - Questions in English
  - Representative code samples mined automatically
  - Programmers can add results, and rank them
- "Scene Completion"

Opportunities

- Improving the analysis
  - Dealing with pure methods
  - Removing spurious ordering constraints
- Adding other sources for examples
  - Krugle
  - Coders
  - Stackoverflow
  - …
- Applying to other programming languages
  - e.g., javascript
- Many more…

Summary

- Client-side static specification mining
  - Partial programs
  - Based on flow-sensitive, context-sensitive abstract interpretation
  - Combined domain abstracting both aliasing and event sequences
- family of abstractions to represent unbounded event sequences
- Summarization algorithms
  - Stitching of partial specifications with unknowns
- Preliminary experimental results

The End