Large-Scale API Protocol Mining for Automated Bug Detection

Michael Pradel

Department of Computer Science
ETH Zurich

Motivation

```
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while ( i.hasNext() ) {
  PinLink curr = (PinLink) i.next();
  if ( ... ) {
    pinConnections.remove(curr);
  }
}
```

(from DaCapo benchmarks)

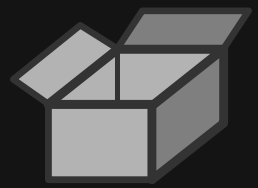
Motivation

```
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while ( i.hasNext() ) {
  PinLink curr = (PinLink) i.next();
  if ( ... ) {
    pinConnections.remove(curr);
  }
}
```

(from DaCapo benchmarks)

Don’t modify a collection while iterating over it!

API Usage Constraints



Program

Constraints

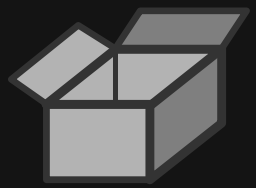
API

call x before y

eventually call x

don’t call x while calling y and z

API Usage Constraints



Program

Constraints

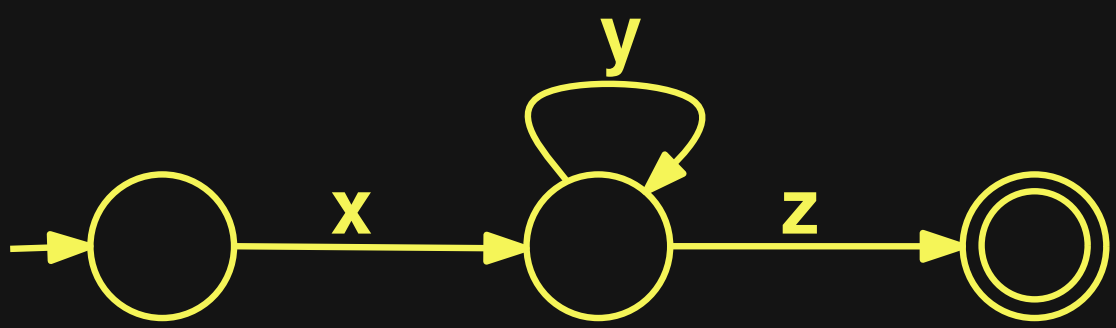
API

call x before y

eventually call x

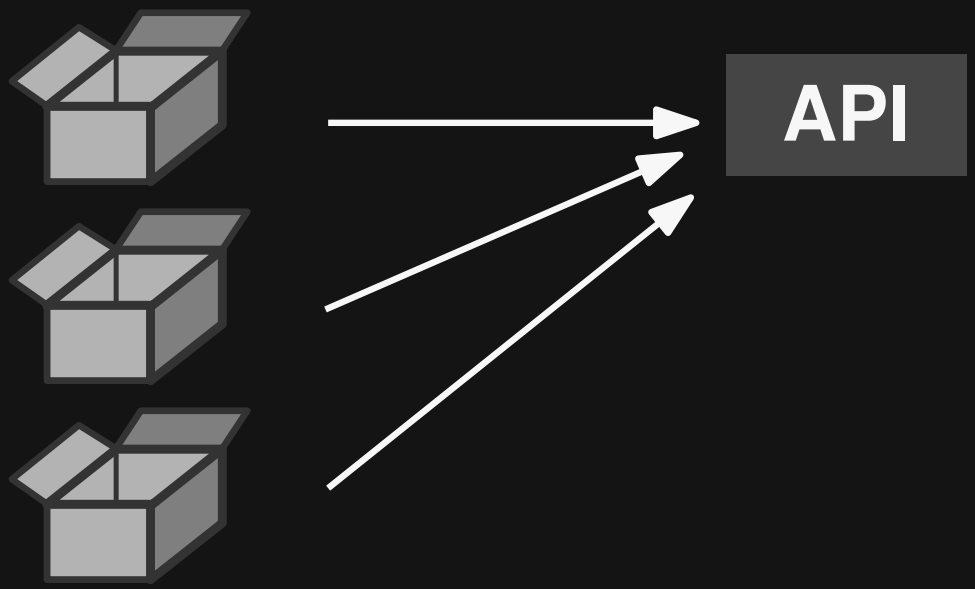
don’t call x while calling y and z

Protocols!



Protocol Mining

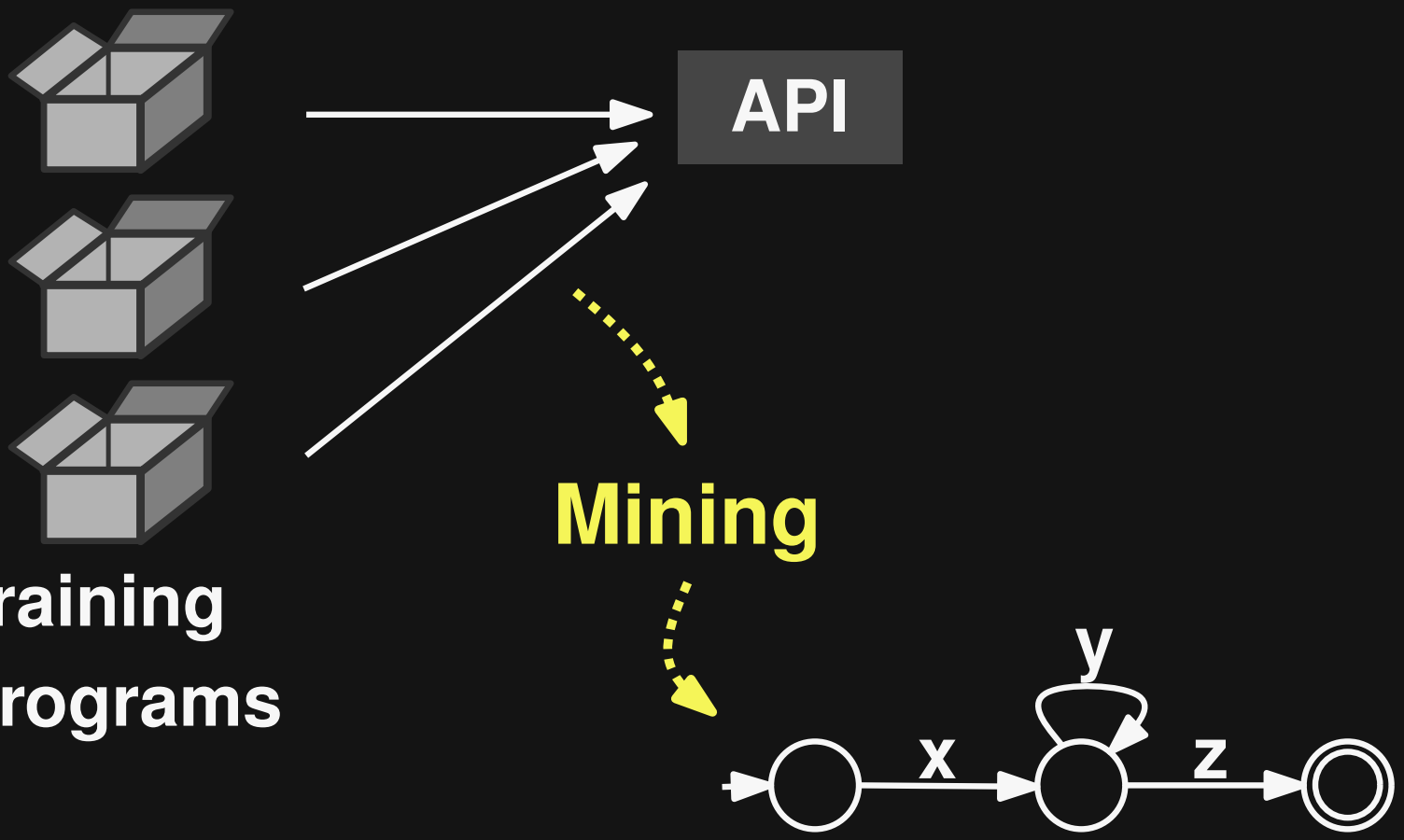
Problem: No protocols specified



Training Programs

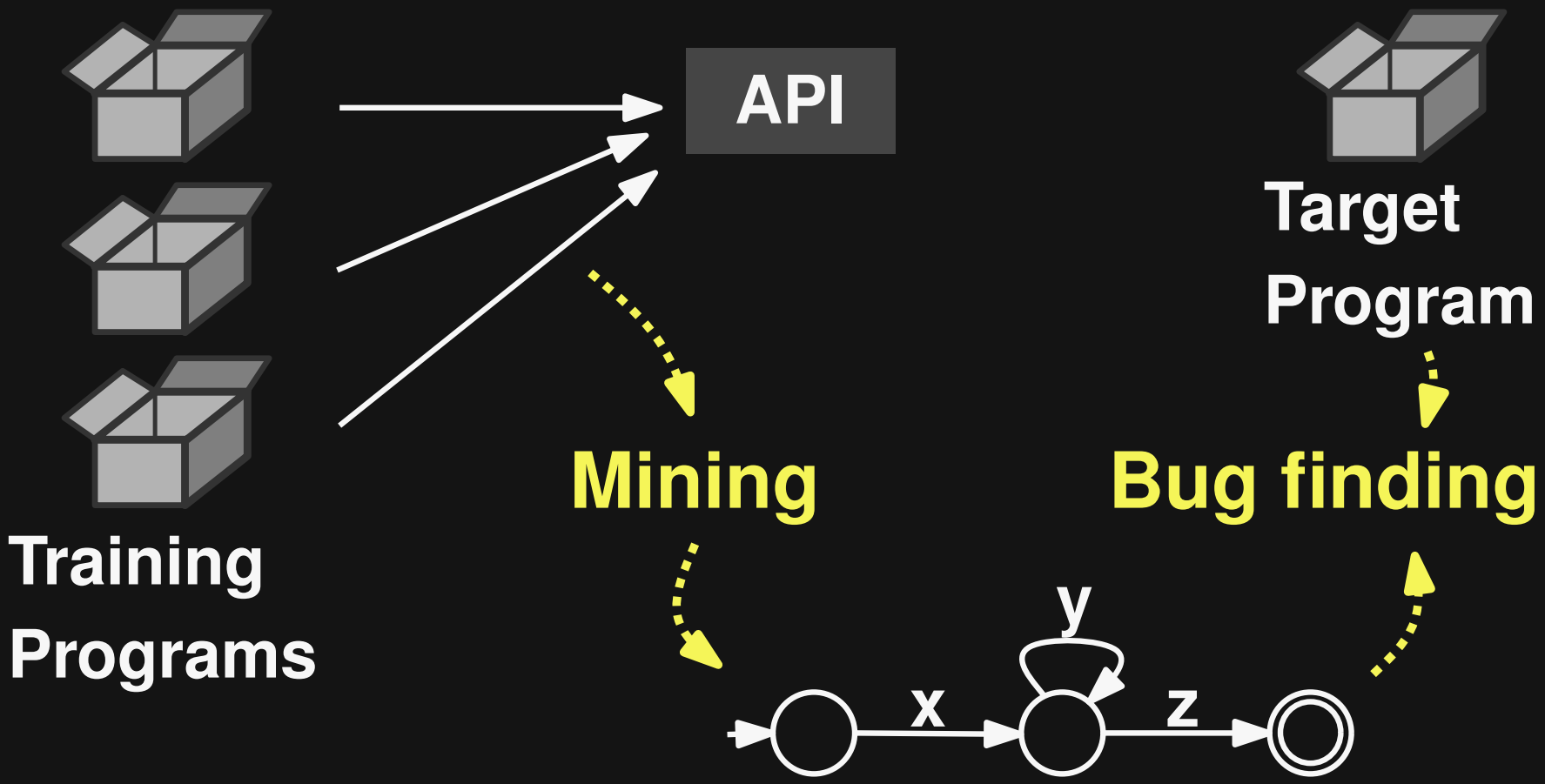
Protocol Mining

Problem: No protocols specified



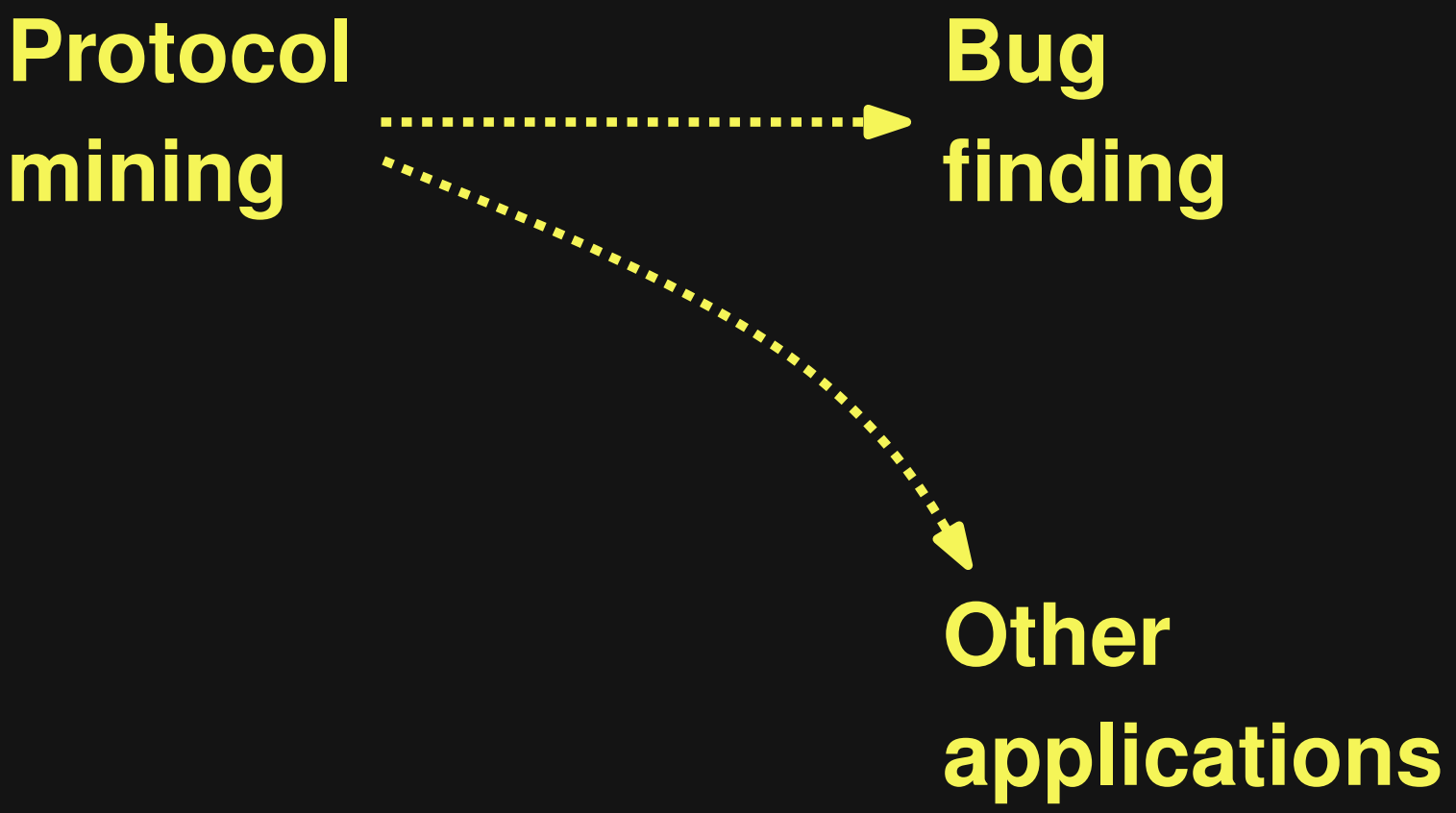
Protocol Mining

Problem: No protocols specified

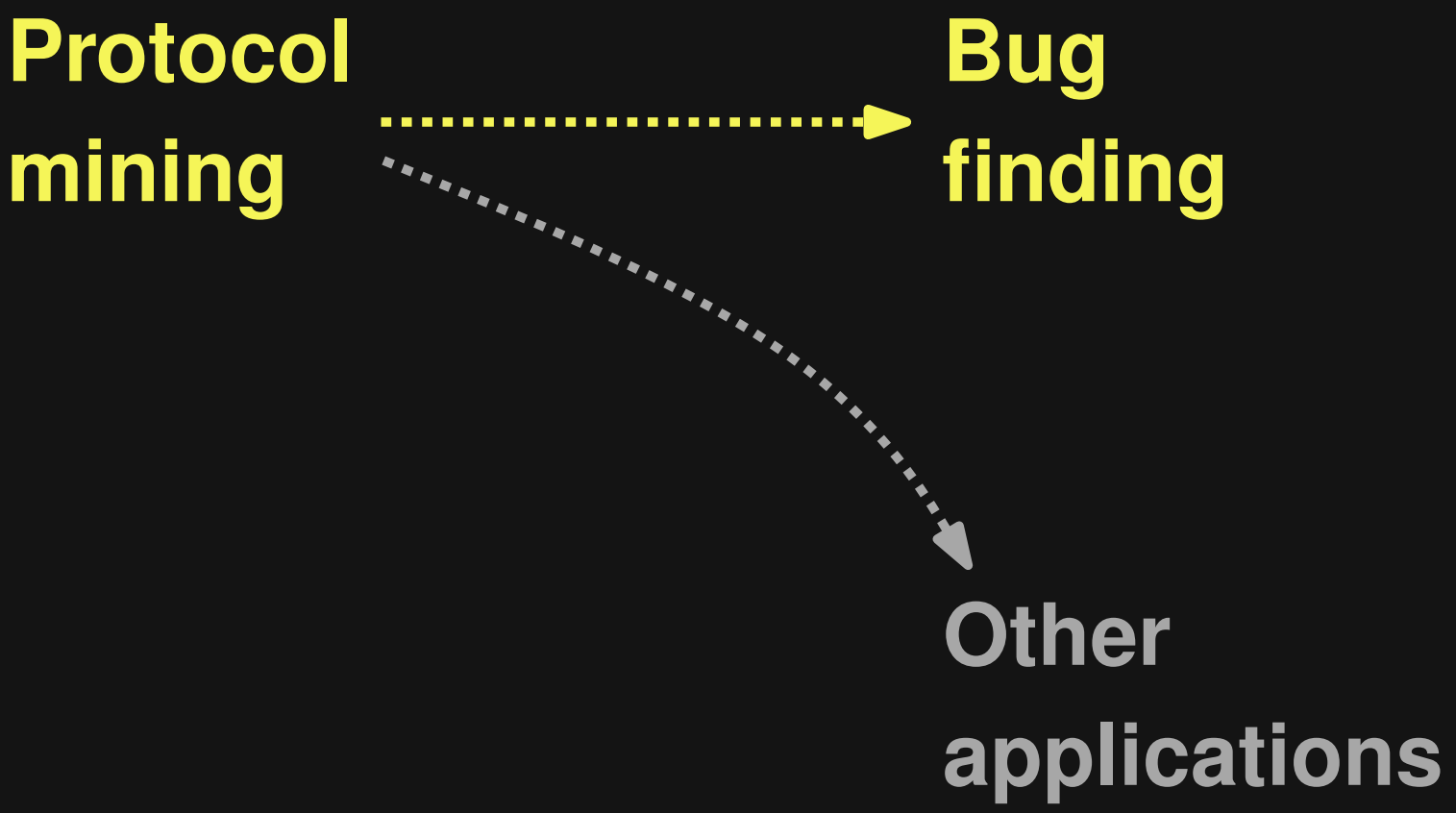


Big Picture

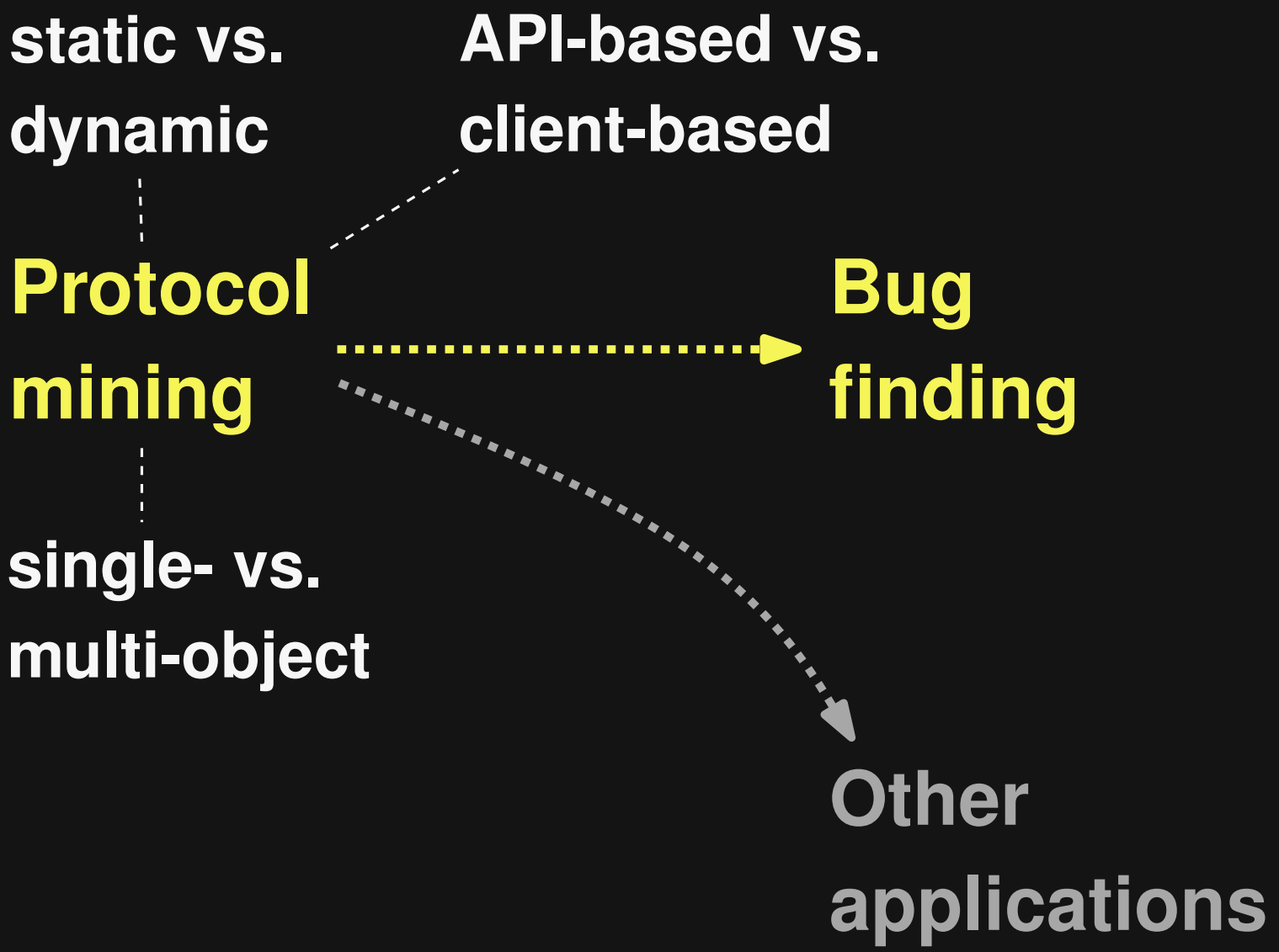
Big Picture



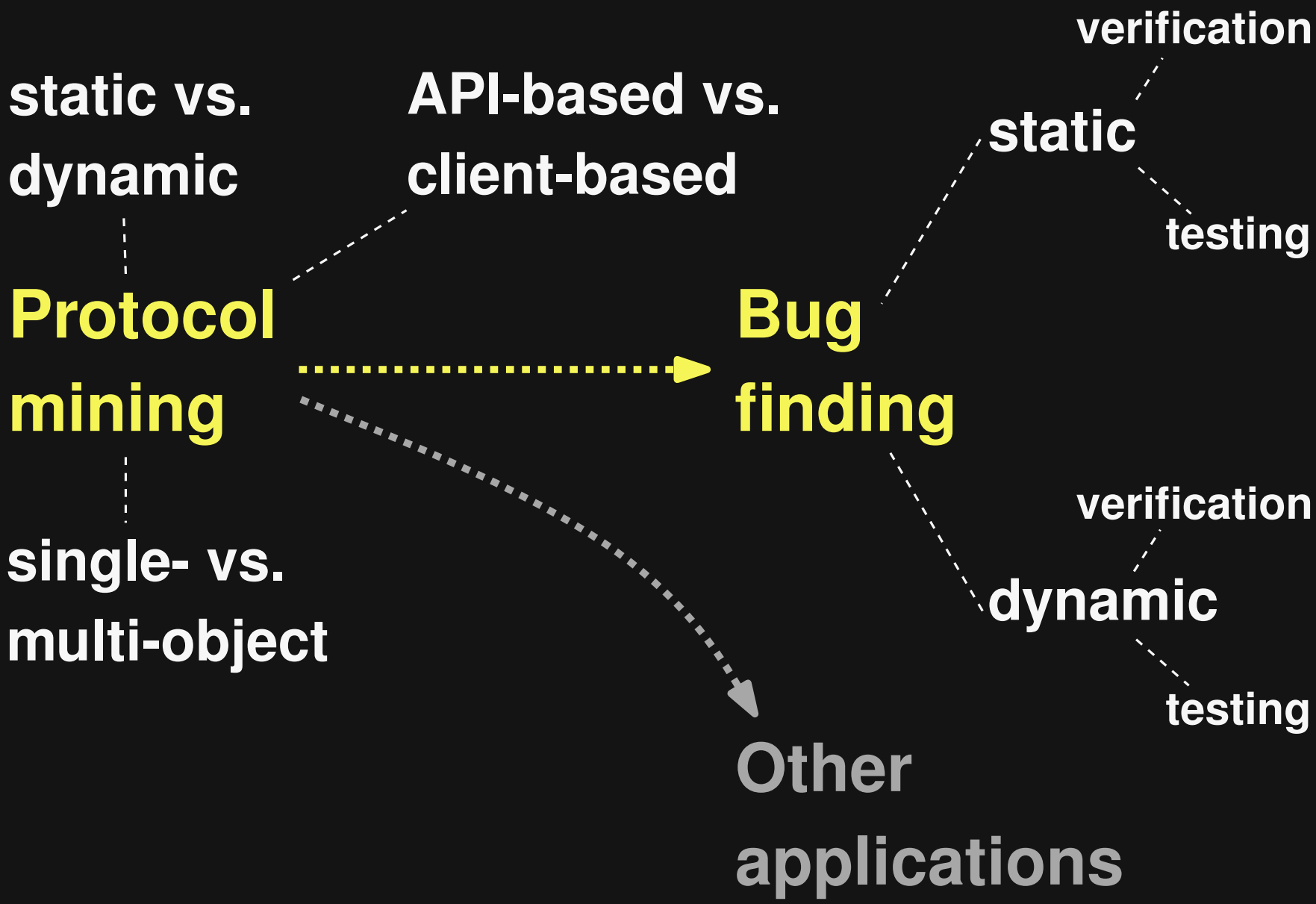
Big Picture



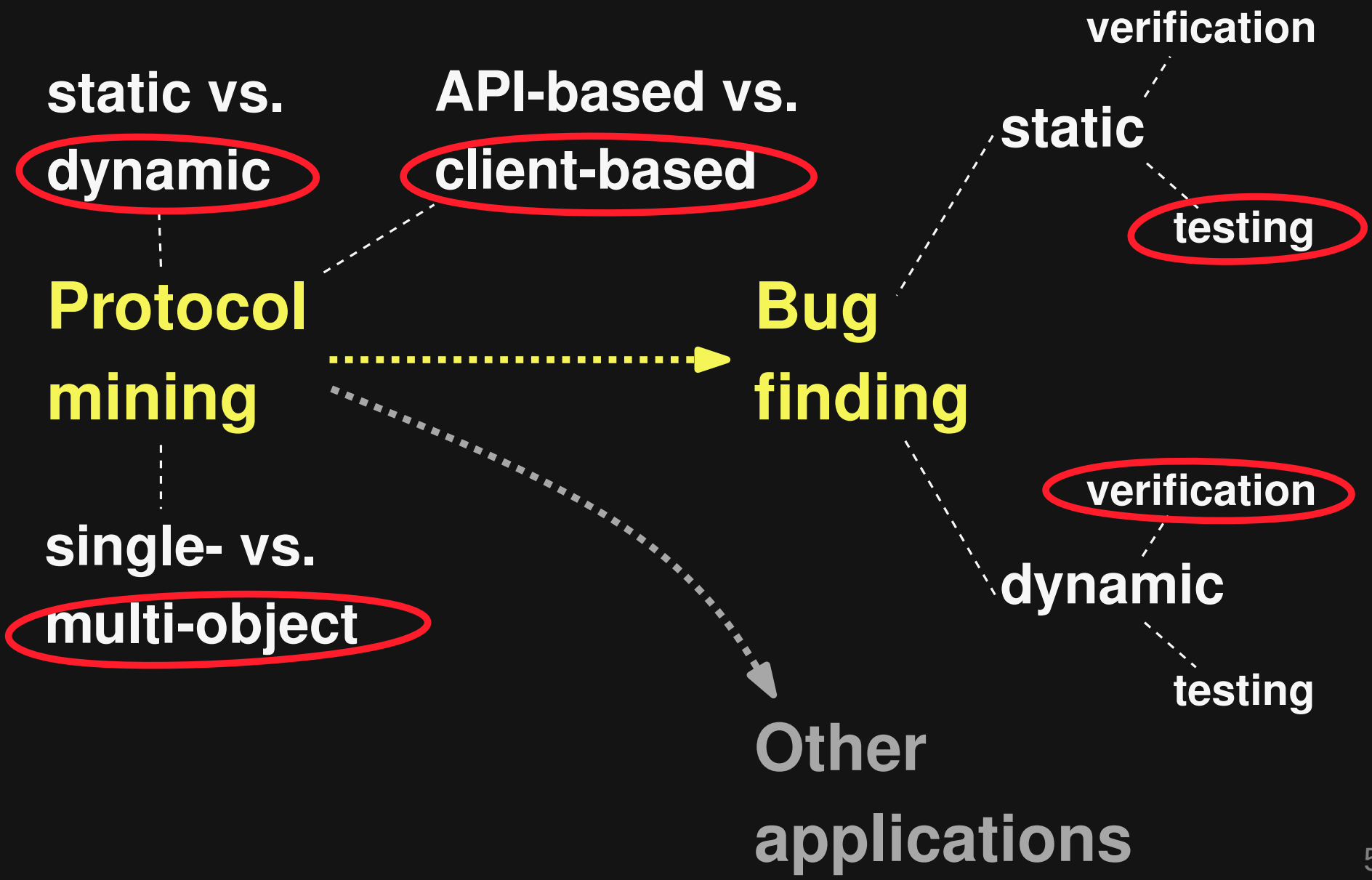
Big Picture


static vs.
dynamic
API-based vs.
client-based
Protocol
mining
Bug
finding
single- vs.
multi-object
Other
applications

Big Picture

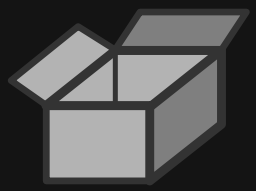

static vs.
dynamic
API-based vs.
client-based
Protocol
mining
single- vs.
multi-object
Bug
finding
static
verification
testing
dynamic
verification
testing
Other
applications

Big Picture


static vs.
dynamic
API-based vs.
client-based
Protocol
mining
single- vs.
multi-object
Bug
finding
static
verification
testing
dynamic
verification
testing
Other
applications

Part 1:
Protocol Mining

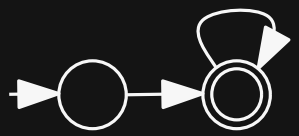
Protocol Mining - Overview



Program
& Input

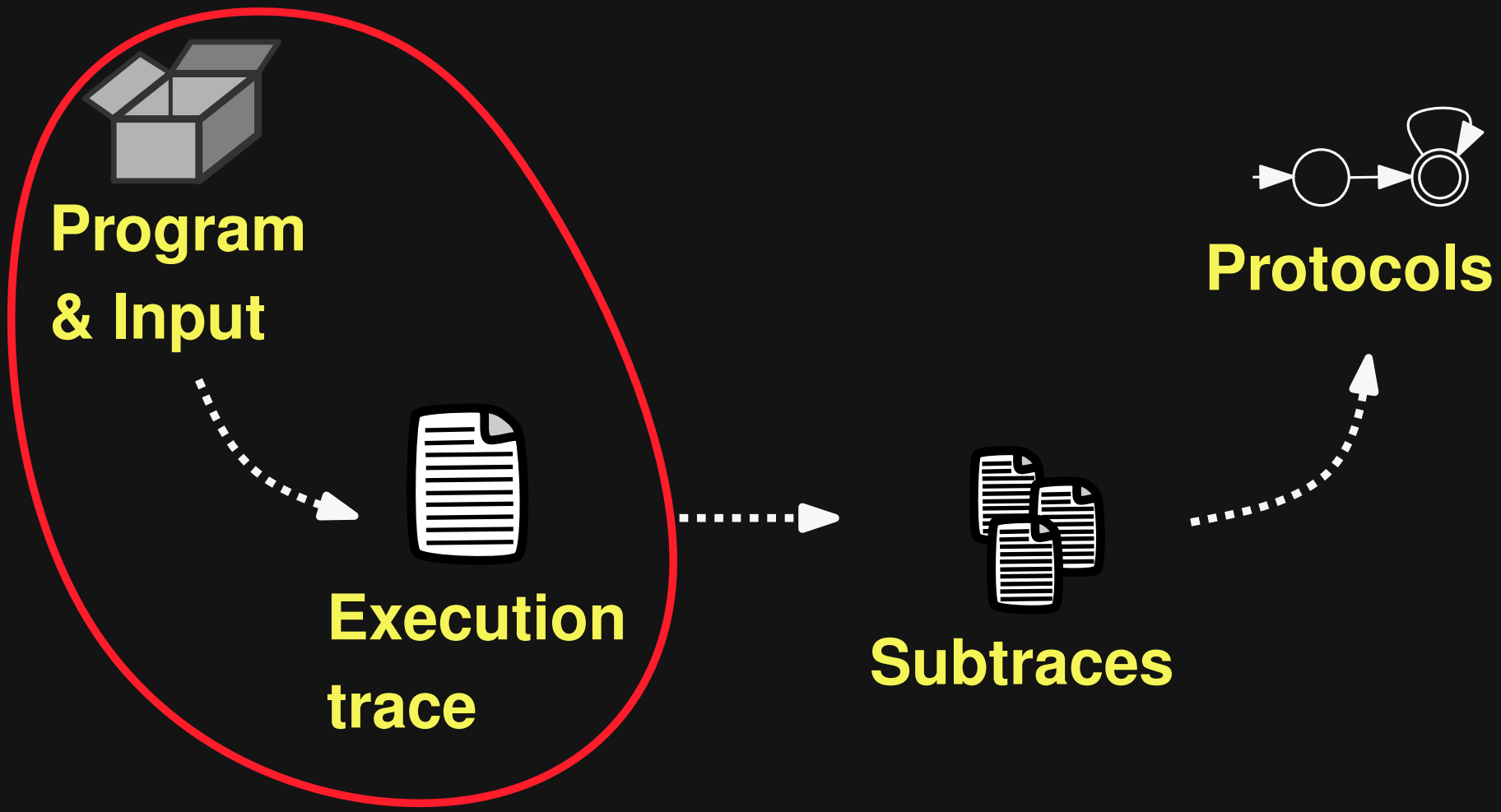
Execution
trace

Subtraces



Protocols

Protocol Mining - Overview



Execution Traces

```
List l = new LinkedList();
l.add(new Foo());
Iterator i = l.iterator();
OutputStream s
    = new FileOutputStream("f");
while (i.hasNext()) {
  Foo f = i.next();
  if (f.isOK())
    s.write(f.getData());
}
s.close();
```

Execution Traces

```
List l = new LinkedList();
l.add(new Foo());
Iterator i = l.iterator();
OutputStream s
    = new FileOutputStream("f");
while (i.hasNext()) {
  Foo f = i.next();
  if (f.isOK())
    s.write(f.getData());
}
s.close();
```

AspectJ instrumentation

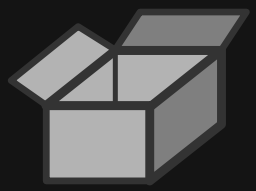
Execution Traces

```
List l = new LinkedList();
l.add(new Foo());
Iterator i = l.iterator();
OutputStream s
    = new FileOutputStream("f");
while (i.hasNext()) {
  Foo f = i.next();
  if (f.isOK())
    s.write(f.getData());
}
s.close();
```

AspectJ instrumentation

- new LinkedList $\rightarrow$ 1
- 1.add(2) $\rightarrow$ 3
- 1.iterator $\rightarrow$ 4
- new FileOS(6) $\rightarrow$ 5
- 4.hasNext
- 4.next
- 5.write(7)
- 4.hasNext
- 5.close

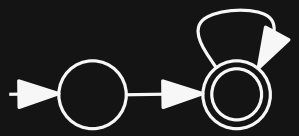
Protocol Mining - Overview



Program
& Input

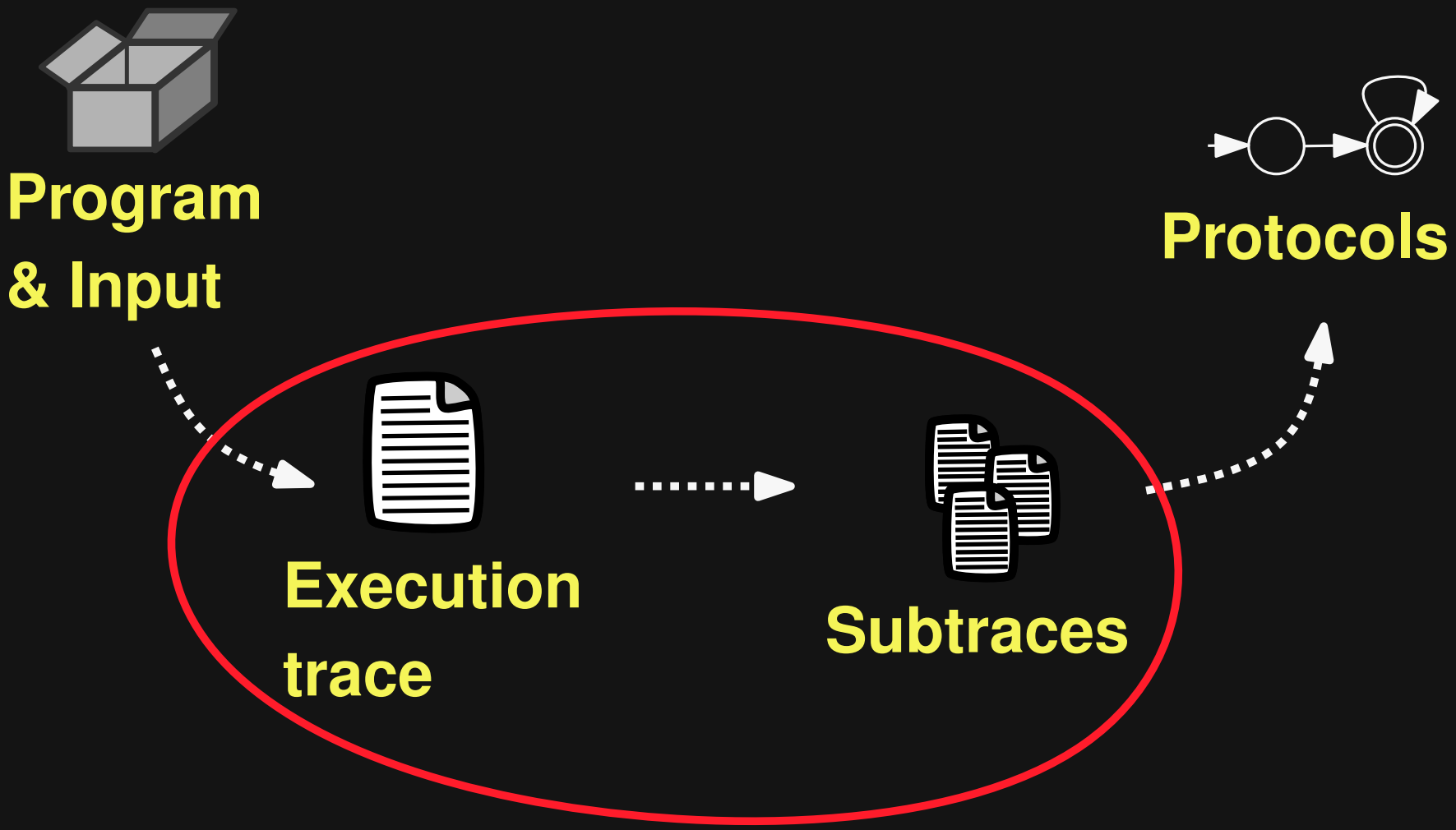
Execution
trace

Subtraces



Protocols

Protocol Mining - Overview



Subtraces

new LinkedList $\rightarrow$ 1
1.add(2) $\rightarrow$ 3
1.iterator $\rightarrow$ 4
new FileOS(6) $\rightarrow$ 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close

Subtraces

new LinkedList $\rightarrow$ 1
1.add(2) $\rightarrow$ 3
1.iterator $\rightarrow$ 4
new FileOS(6) $\rightarrow$ 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close

Different API usages intermingled!

Subtraces

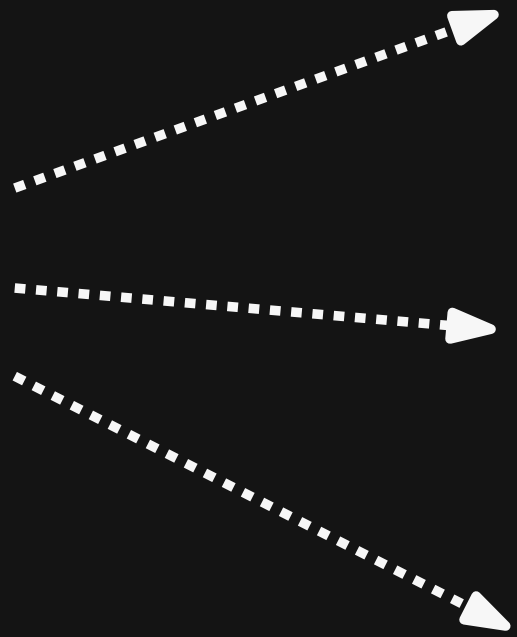
new LinkedList $\rightarrow$ 1
1.add(2) $\rightarrow$ 3
1.iterator $\rightarrow$ 4
new FileOS(6) $\rightarrow$ 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close

Core object x:

- Calls to x
- Calls to parameters passed to x
- Calls to objects returned by x

Subtraces

new LinkedList → 1
1.add(2) → 3
1.iterator → 4
new FileOS(6) → 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close

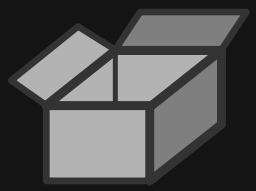


new LinkedList → 1
1.add(2)→3
1.iterator→4
4.hasNext
4.next
4.hasNext

4.hasNext
4.next
4.hasNext

new FileOS(6) → 5
5.write(7)
5.close

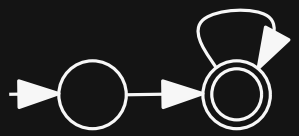
Protocol Mining - Overview



Program
& Input

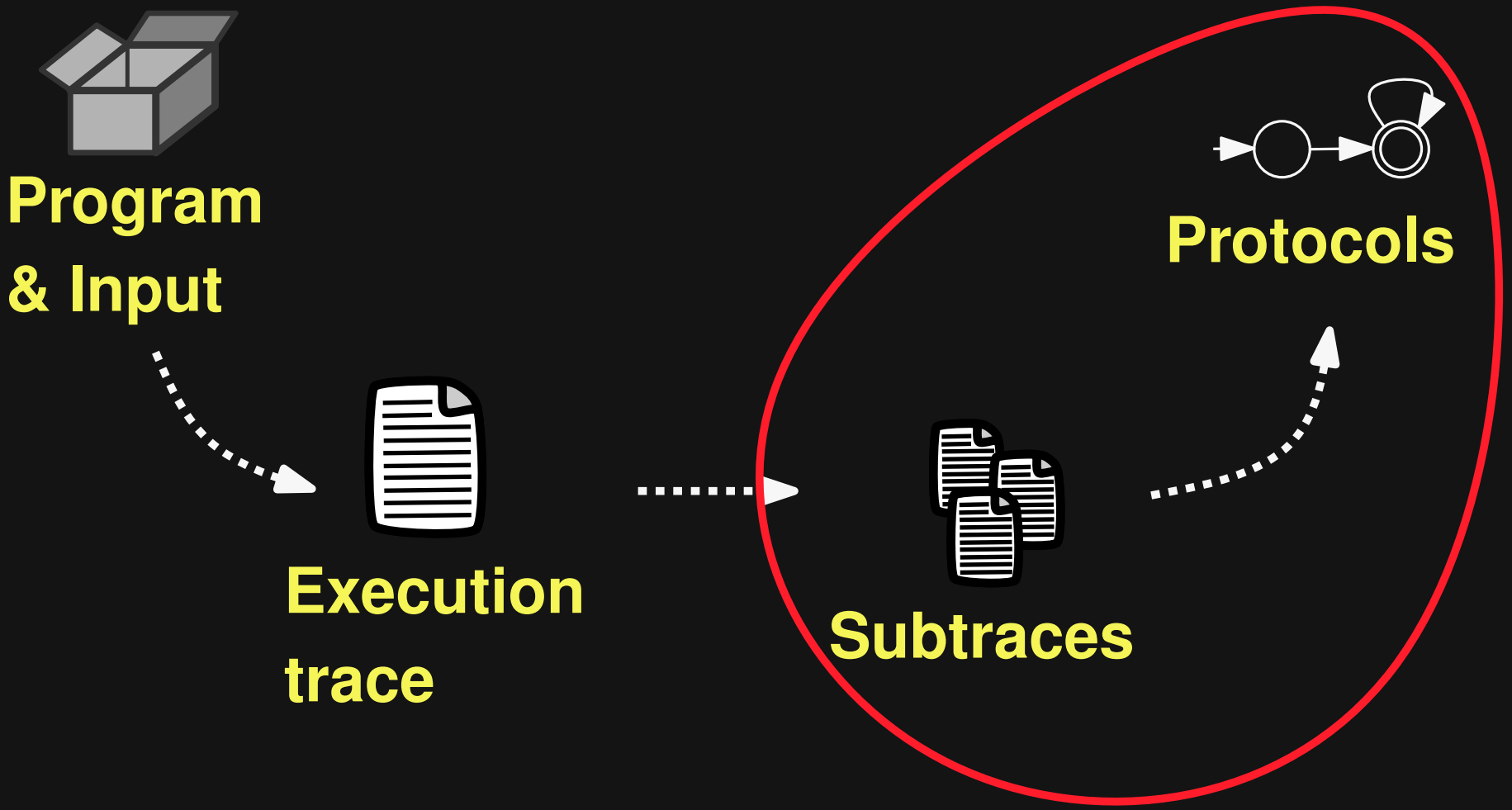
Execution
trace

Subtraces



Protocols

Protocol Mining - Overview



Group Subtraces

Group by set of involved types

LinkedList, Iterator

Iterator

FileOS

Generate Protocols

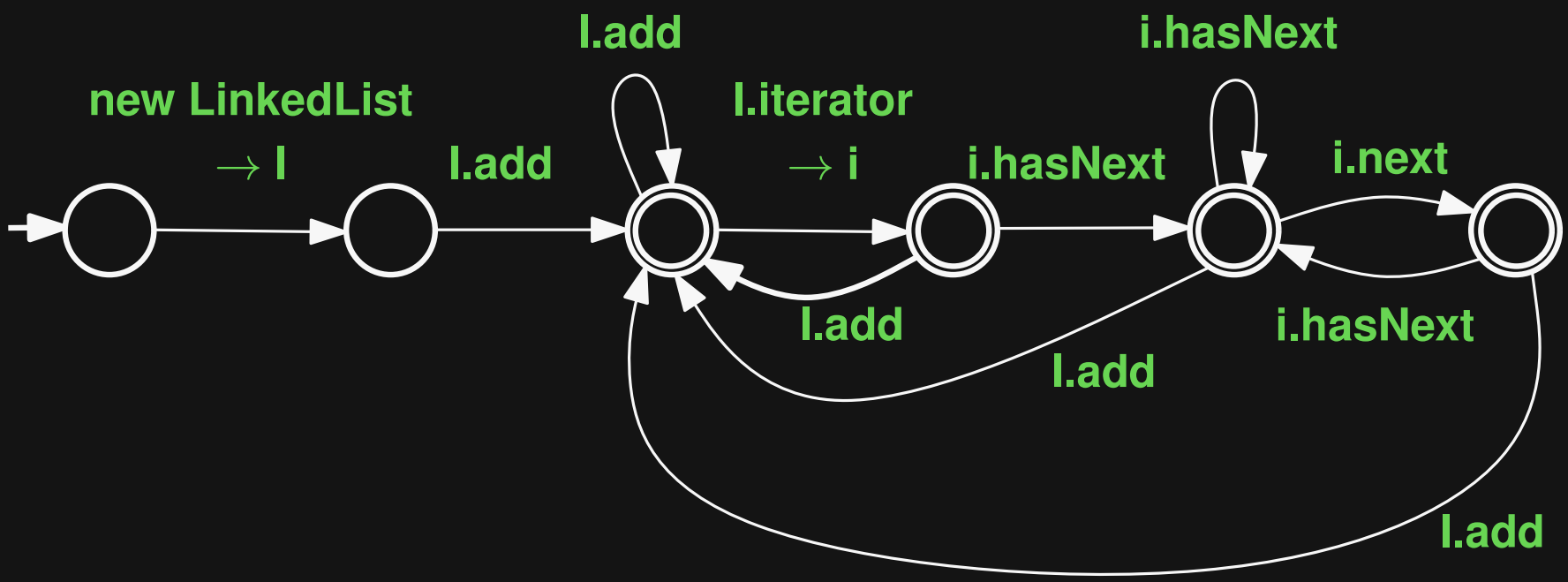
Finite state machine

- **Method $\rightarrow$ state**
- **Consecutive call $\rightarrow$ transition**

Generate Protocols

Finite state machine

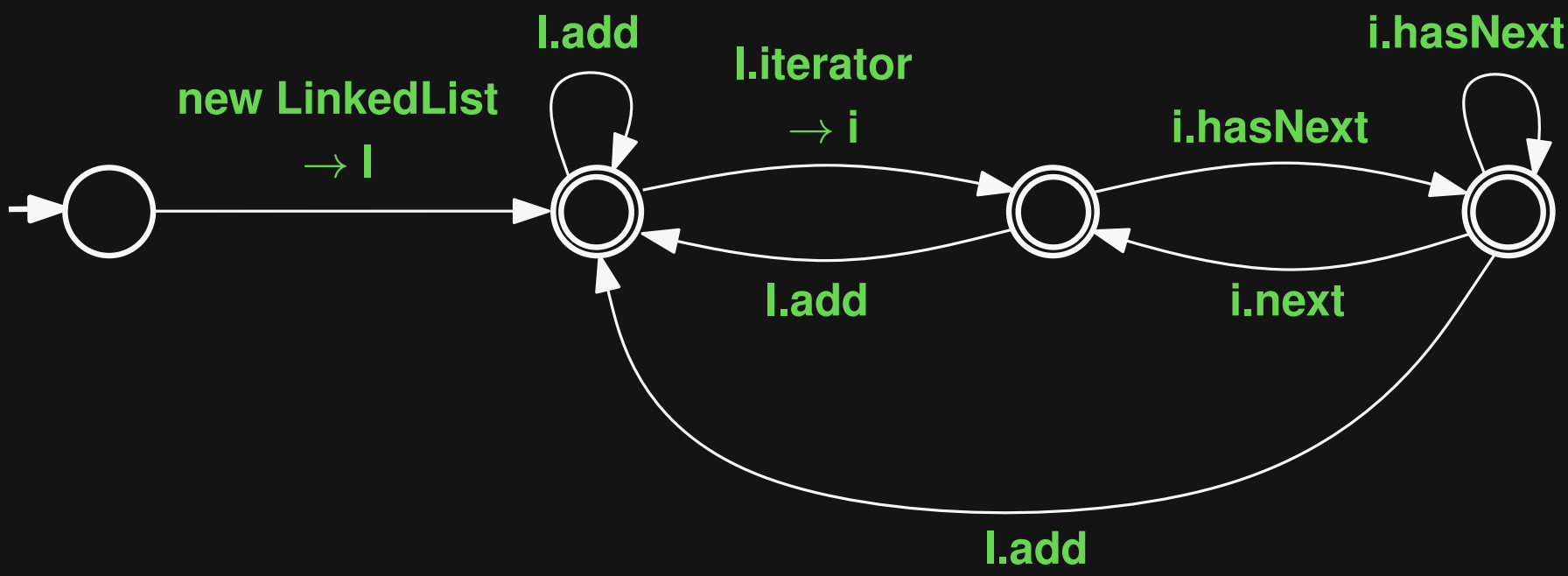
- Method $\rightarrow$ state
- Consecutive call $\rightarrow$ transition



Generate Protocols

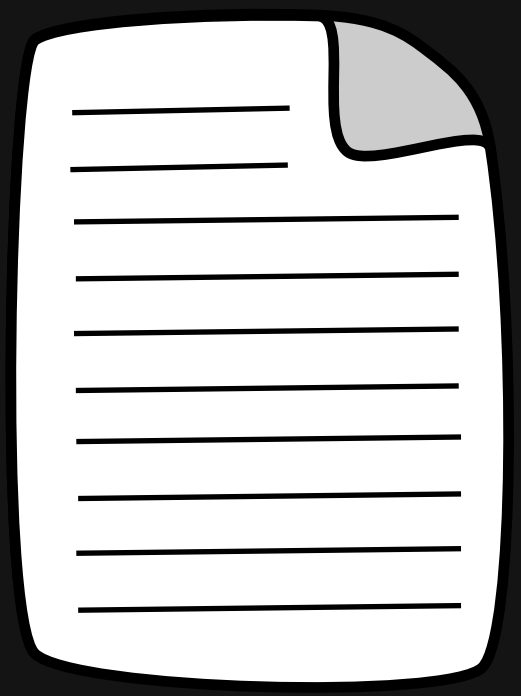
Finite state machine

- **Method $\rightarrow$ state**
- **Consecutive call $\rightarrow$ transition**



Scalability

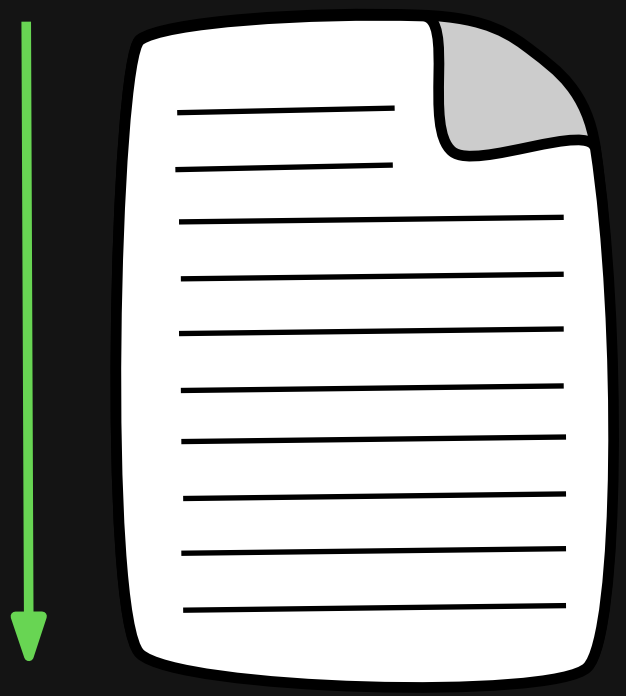
Bottleneck: Large execution traces



Scalability

Bottleneck: Large execution traces

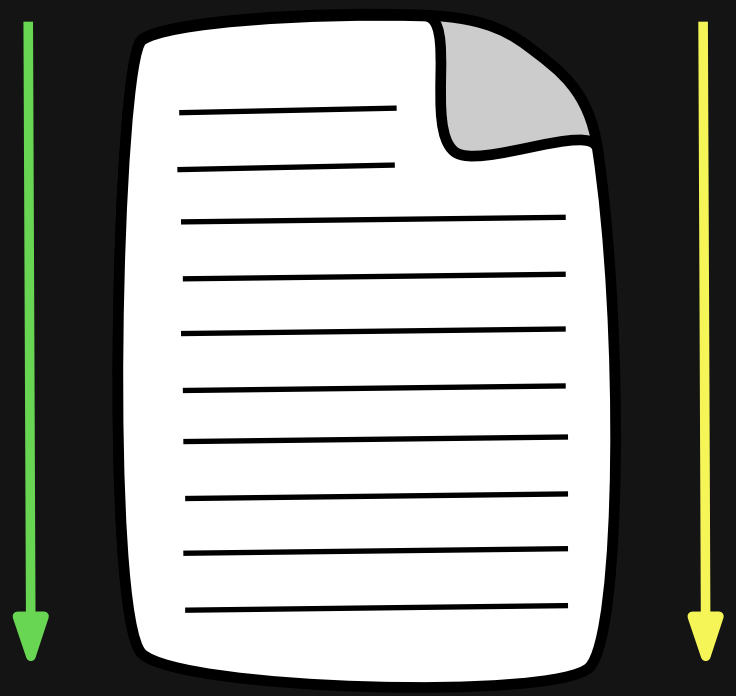
Pass 1: Find core objects and associated objects



Scalability

Bottleneck: Large execution traces

Pass 1: Find core objects and associated objects

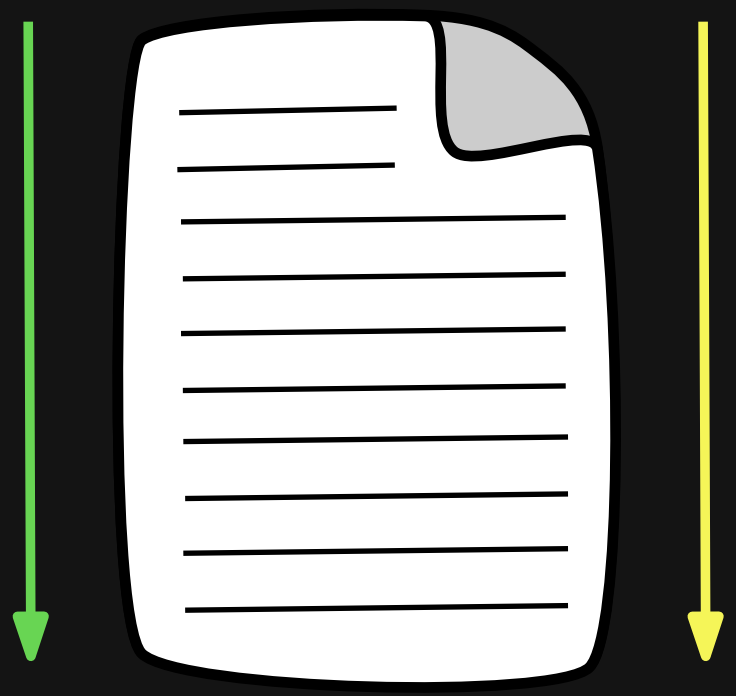


Pass 2: Extract calls for each subtrace

Scalability

Bottleneck: Large execution traces

Pass 1: Find core objects and associated objects

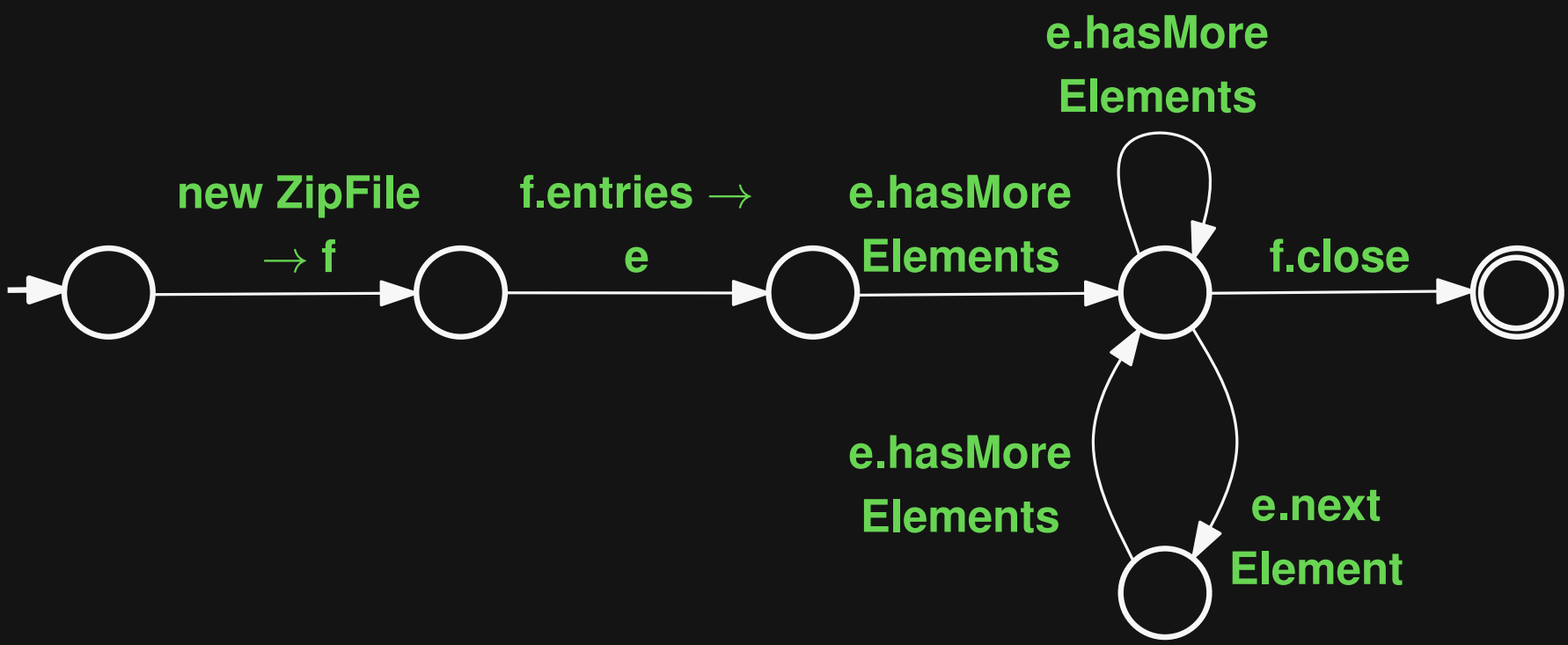


Pass 2: Extract calls for each subtrace

Mine millions of events in a few minutes

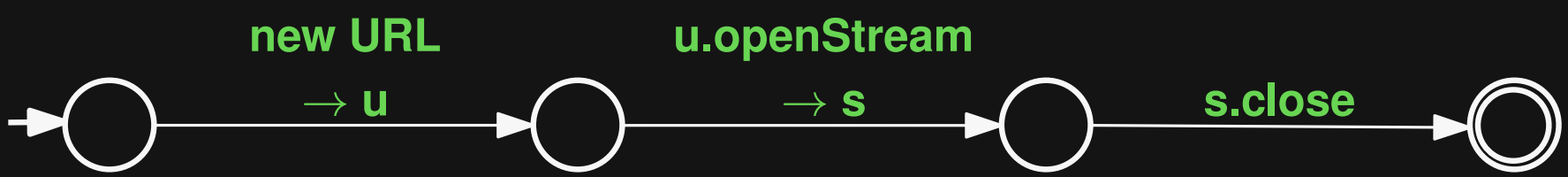
Examples

ZipFile and Enumeration



Examples

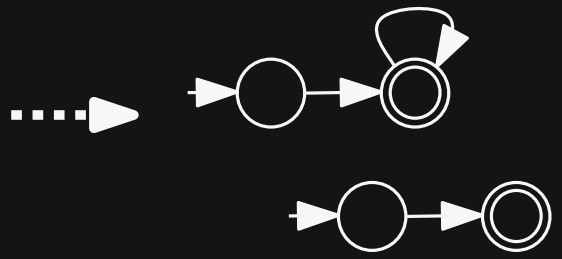
URL and InputStream



Evaluation

Are examples convincing enough?

20+ mining approaches

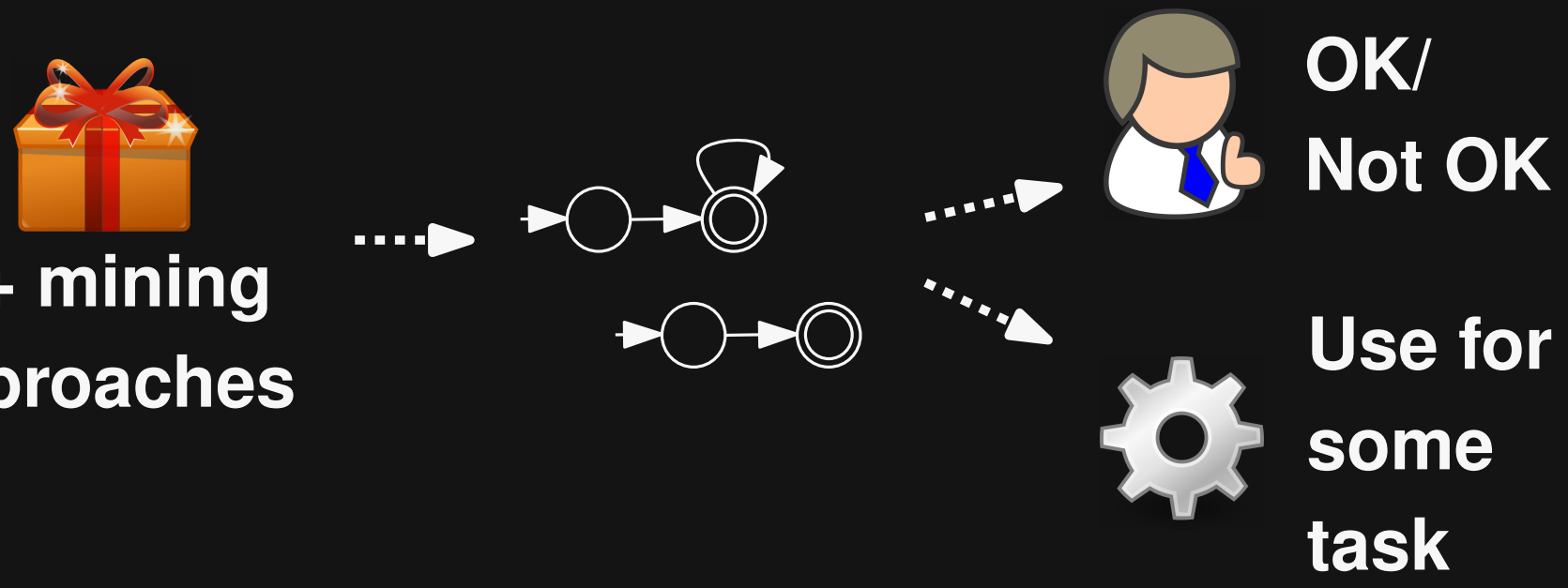


OK/
Not OK

Use for some task

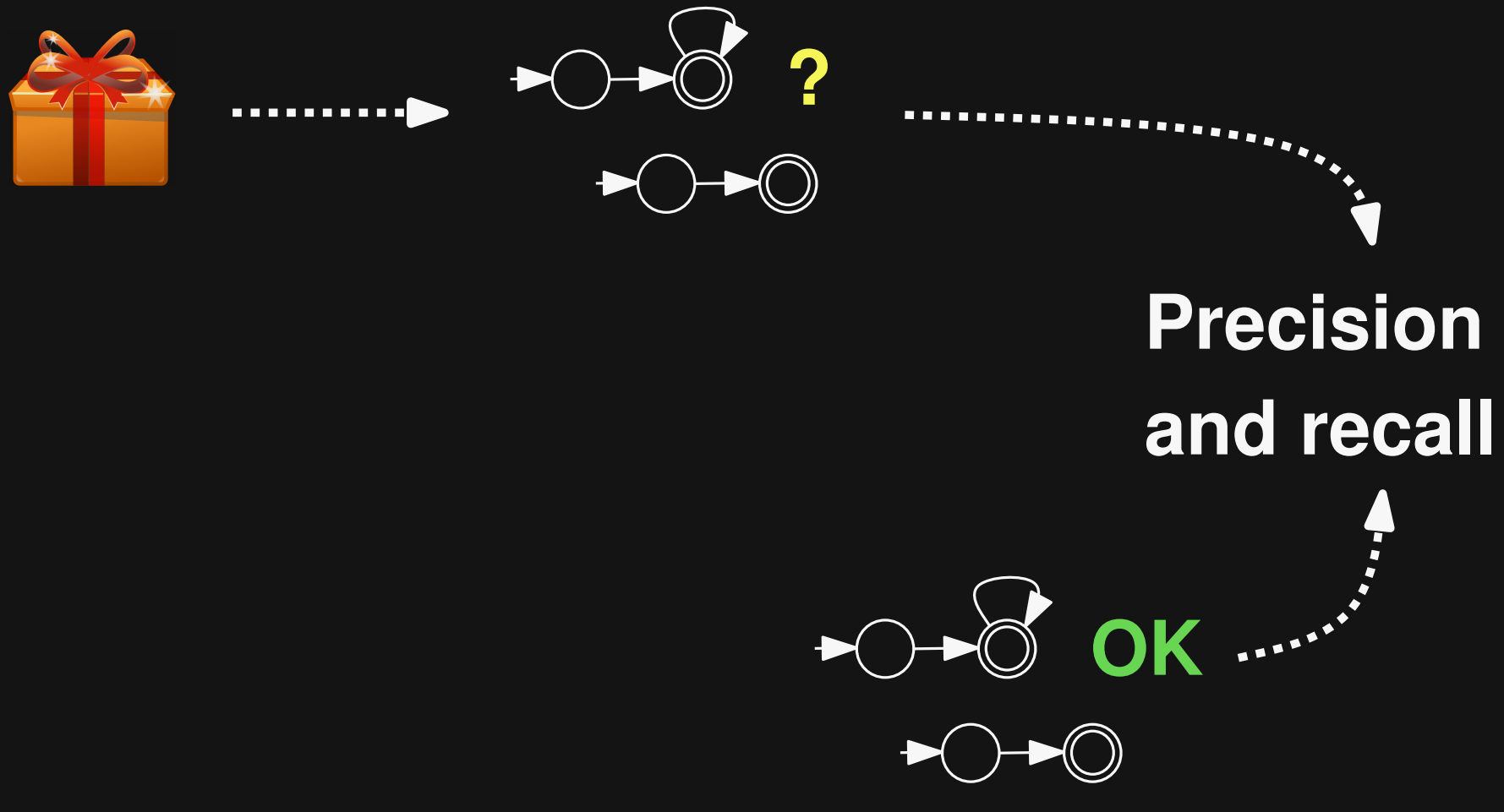
Evaluation

Are examples convincing enough?

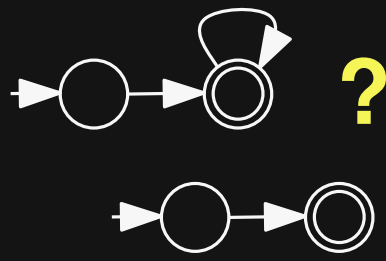


Neither reproducible nor comparable!

Evaluation Framework

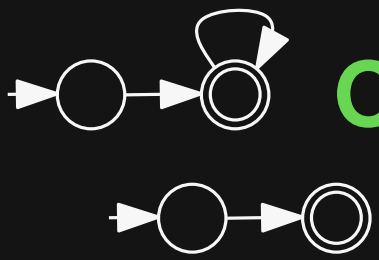


Evaluation Framework



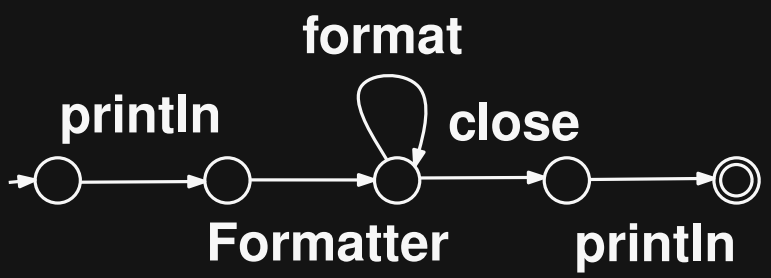
Precision and recall

Method constraint groups

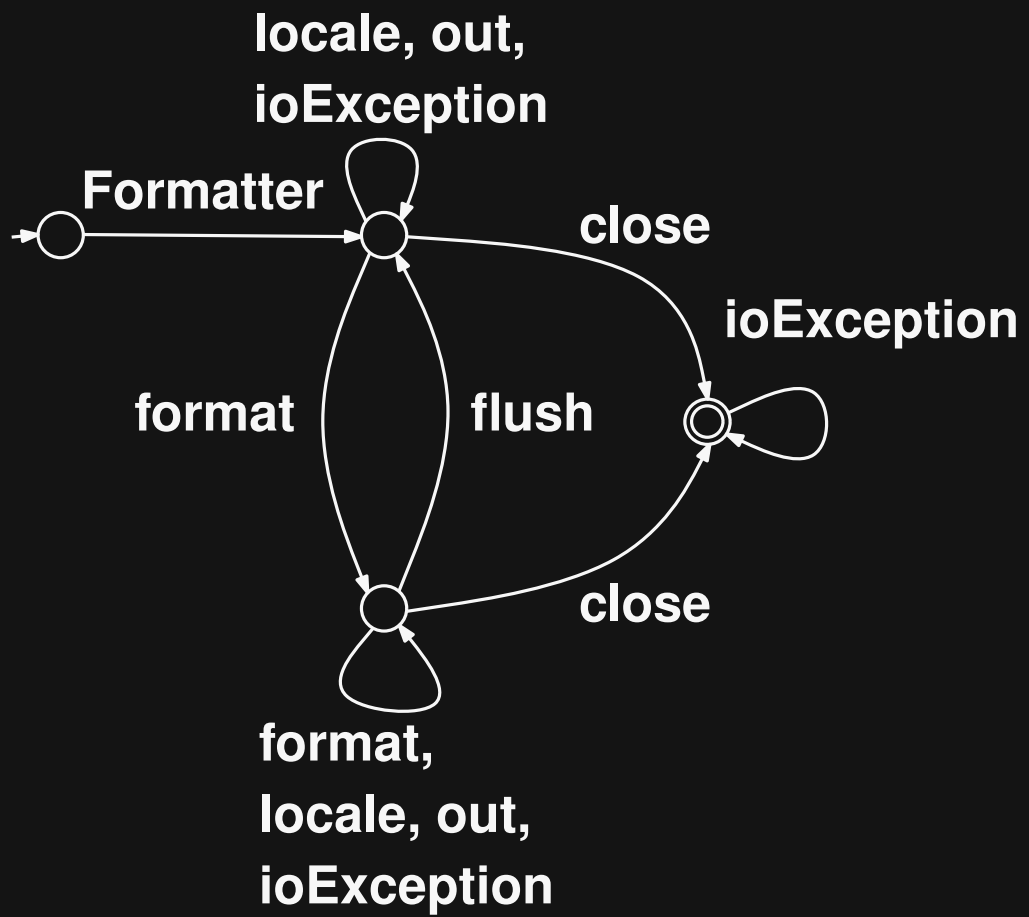


Example

Mined protocol M

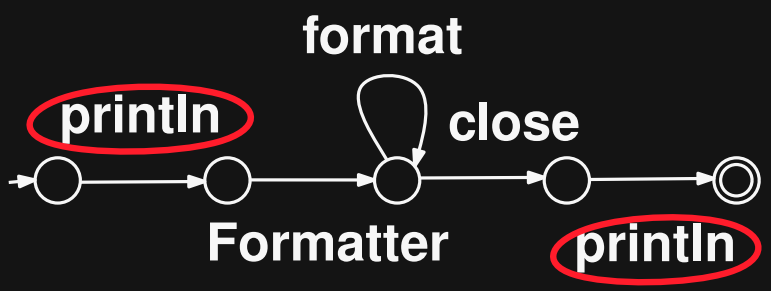


Reference protocol

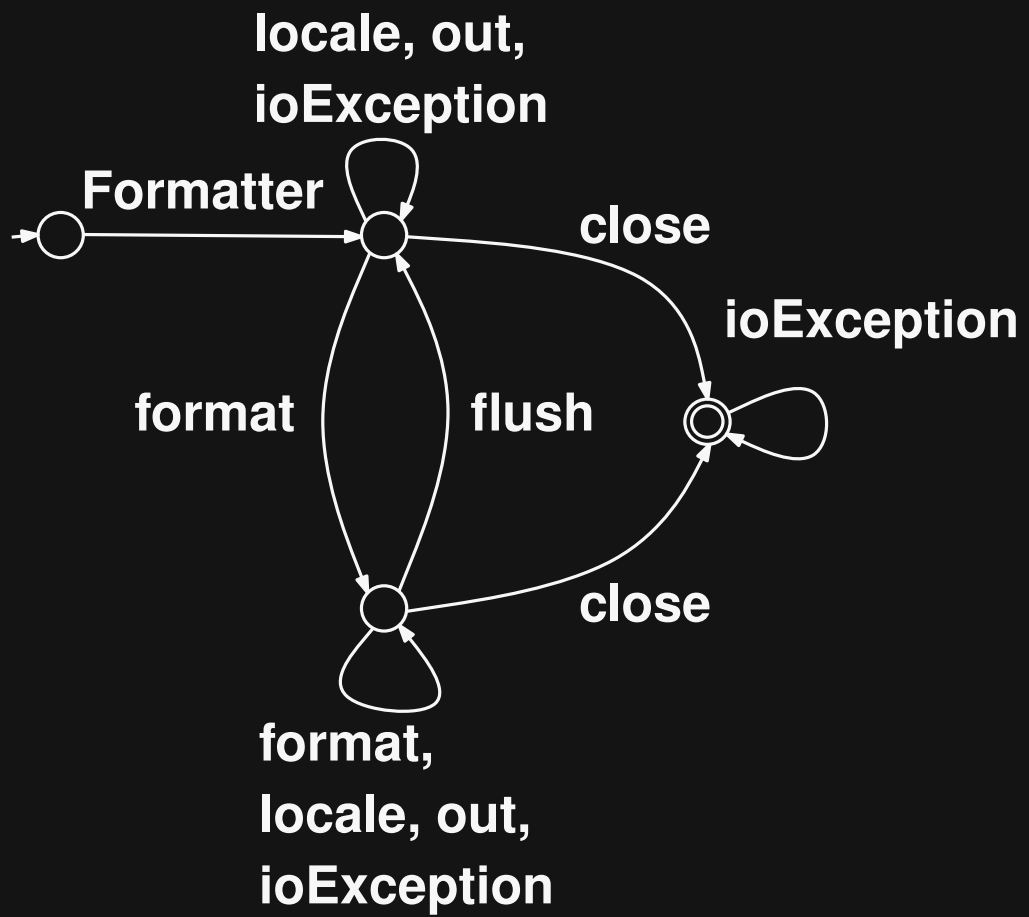


Example

Mined protocol M



Reference protocol

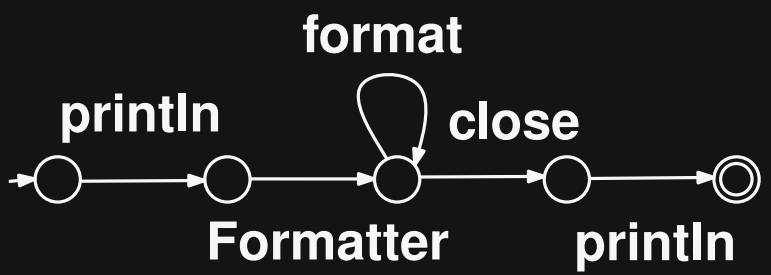


Precision:

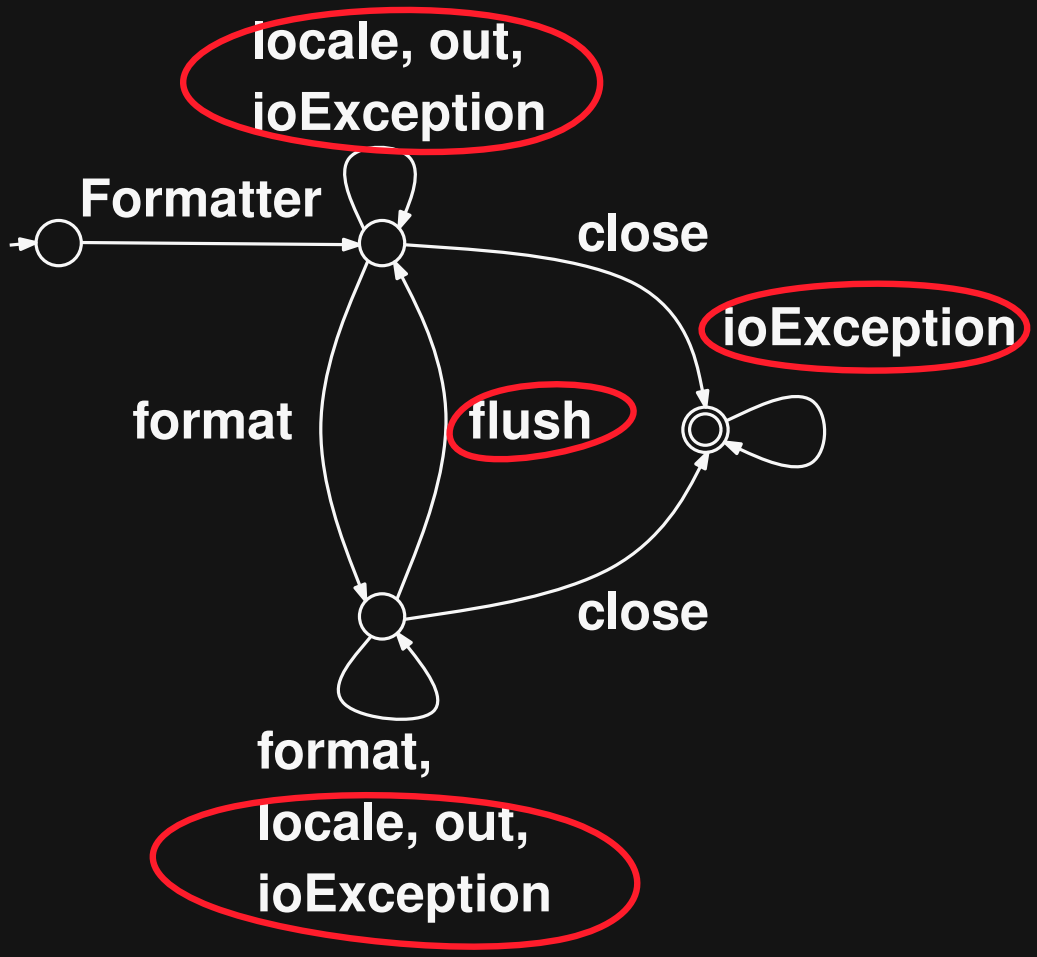
How much of M is correct? $\rightarrow$ 23%

Example

Mined protocol M



Reference protocol

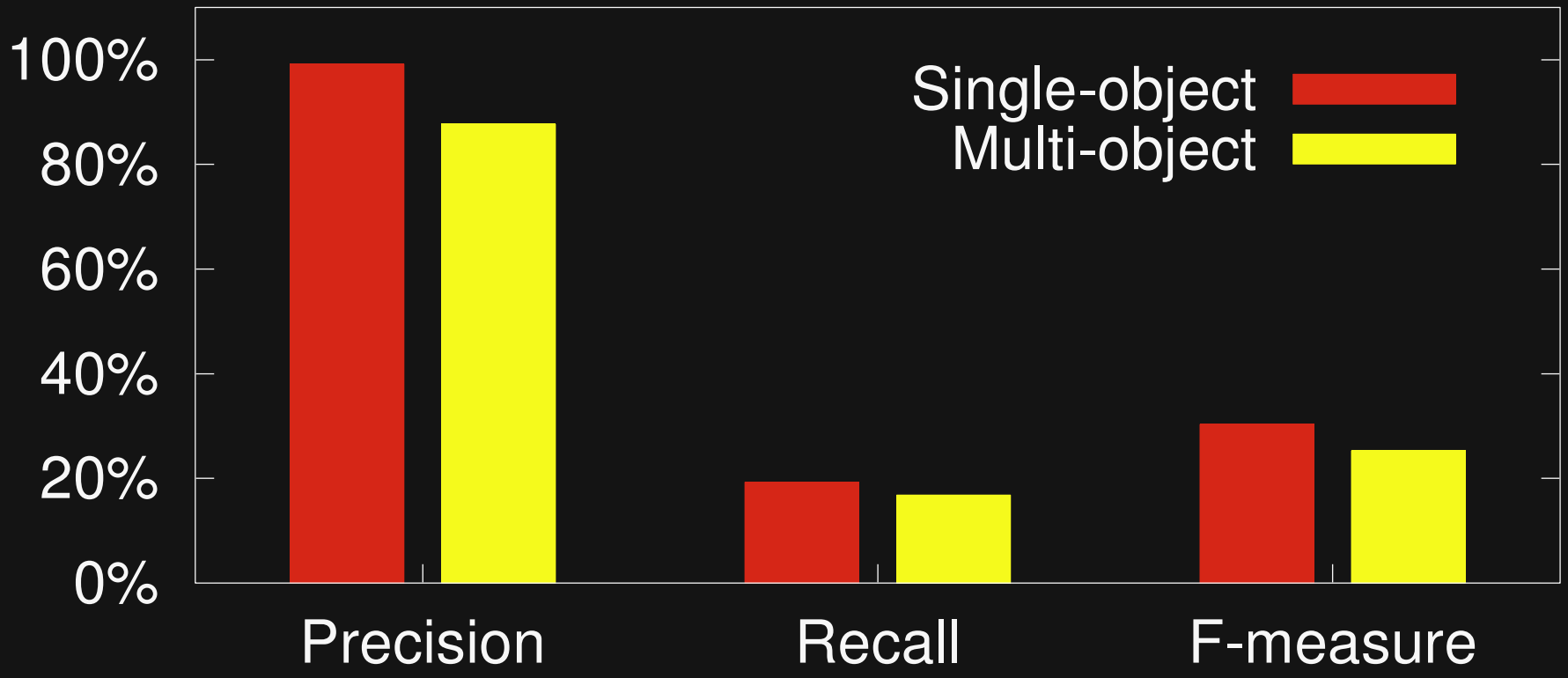


Recall:

How complete is M?

$\rightarrow$ **9%**

Results



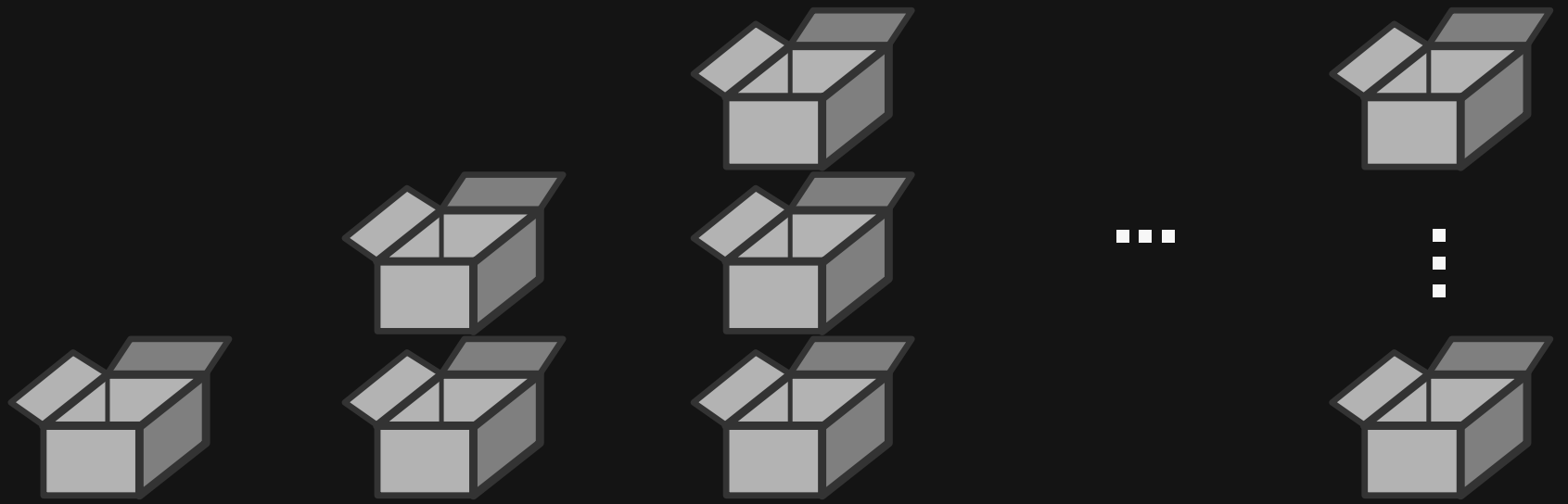
12 training programs

32 reference protocols

Learn More from Many Teachers?

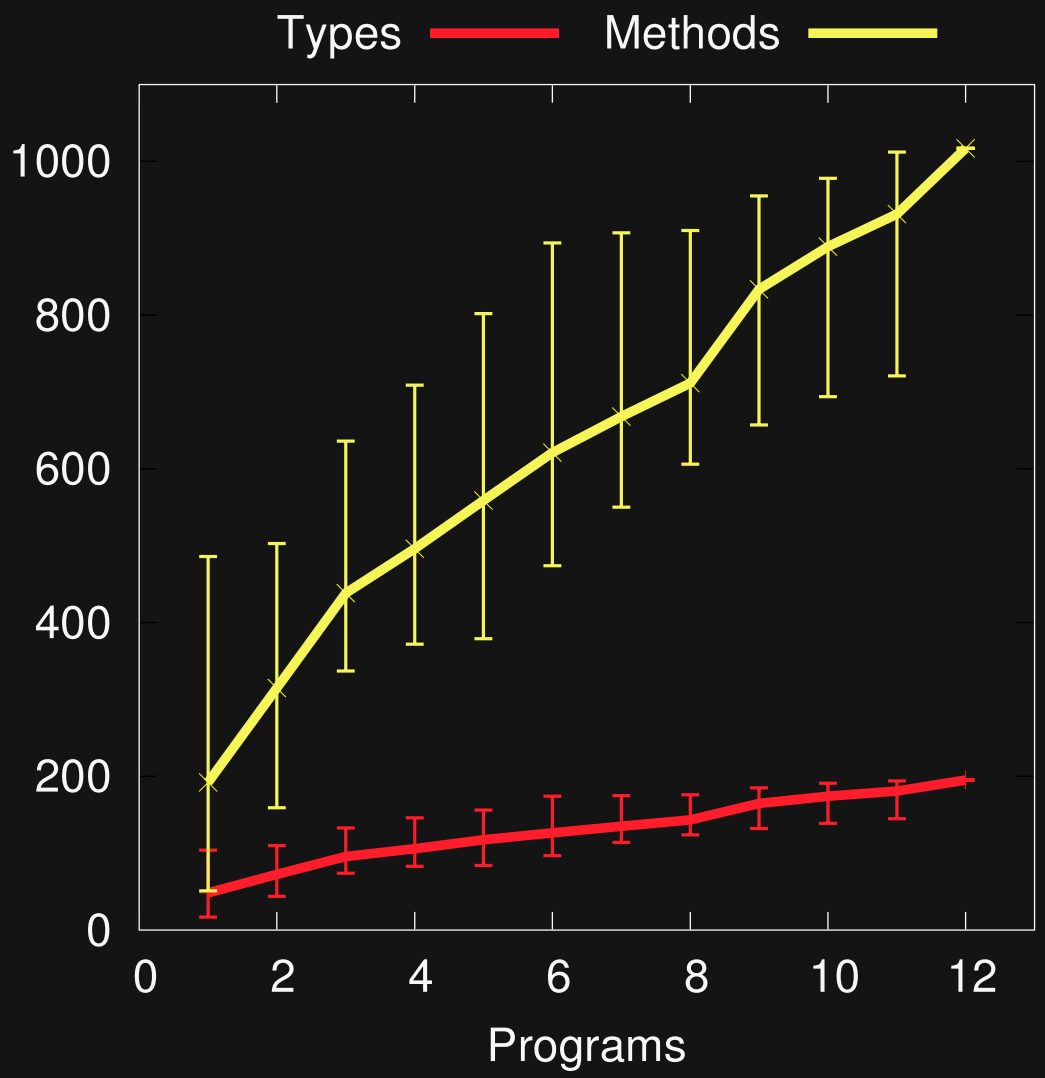
Empirical study:

How does mining more programs influence the results?



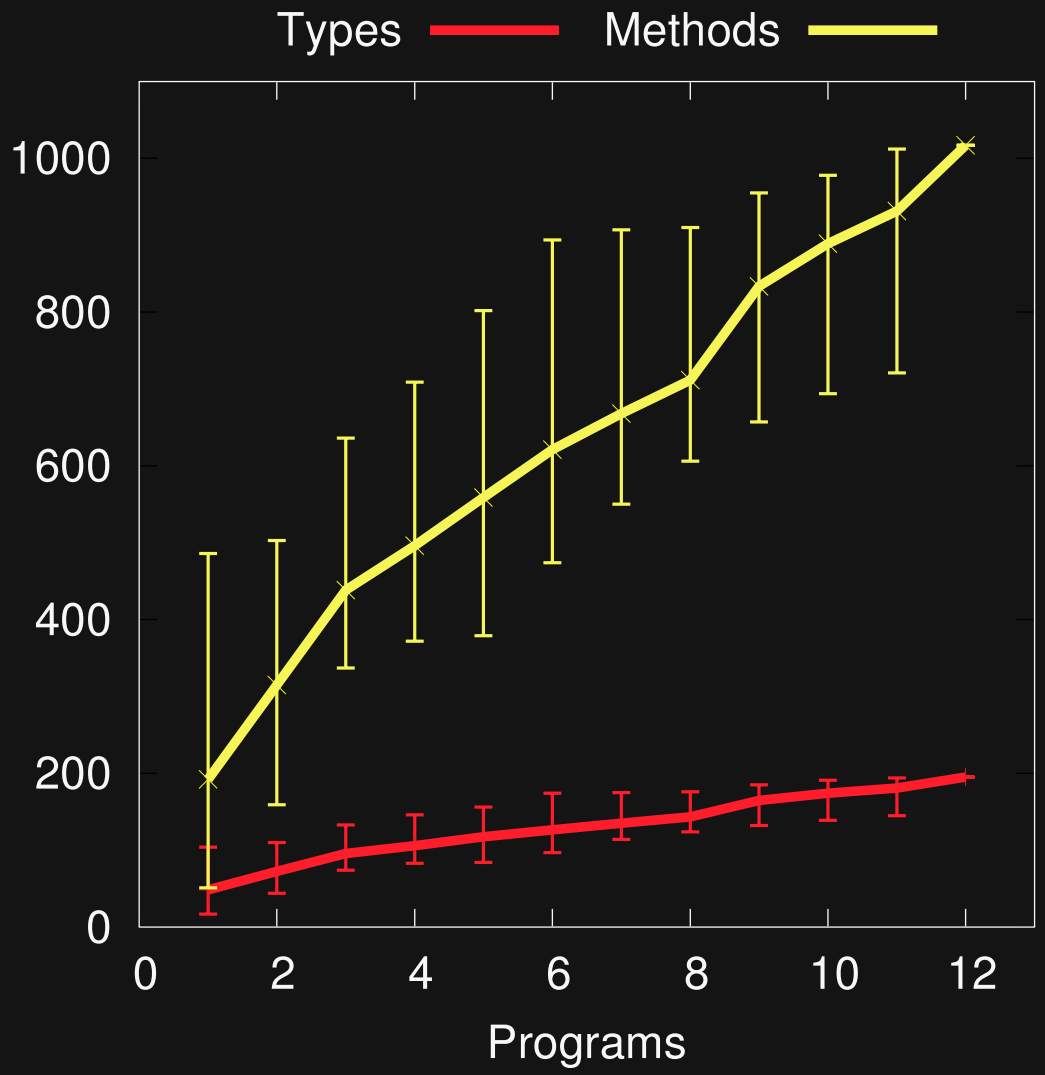
API Coverage

Types and methods covered in mined protocols



API Coverage

Types and methods covered in mined protocols



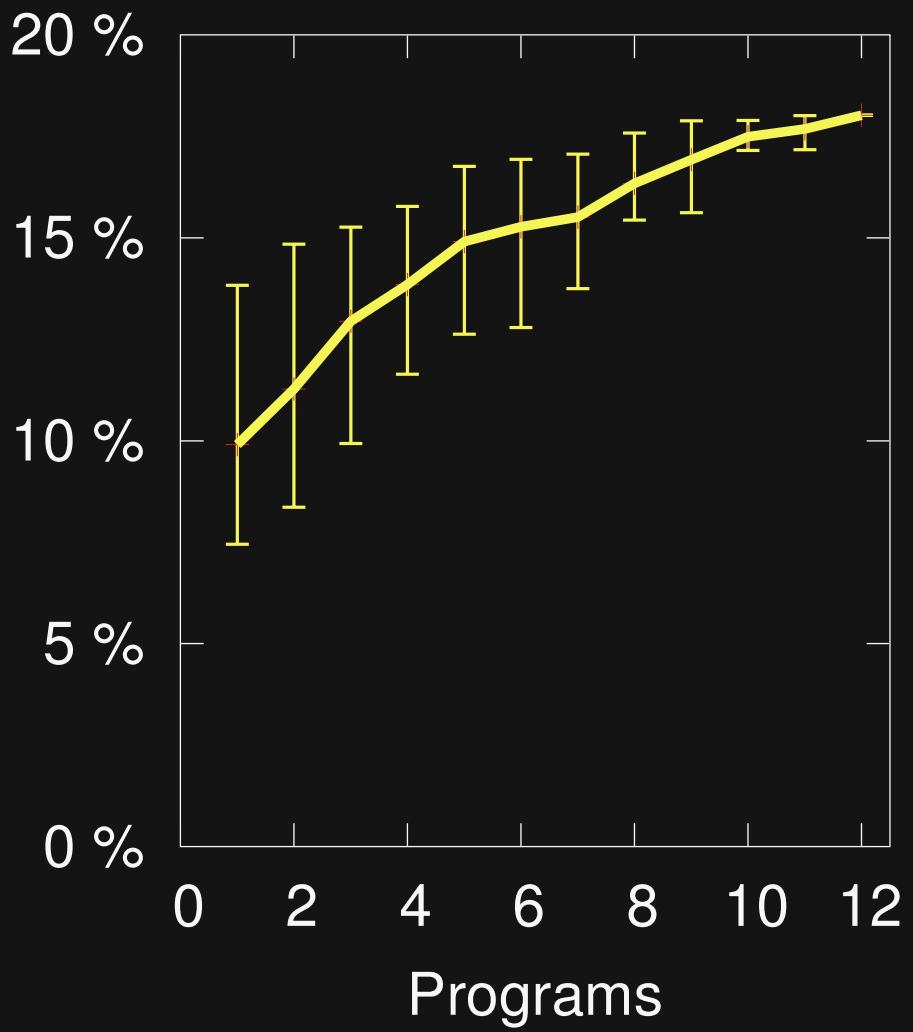
More programs

↓

Higher coverage

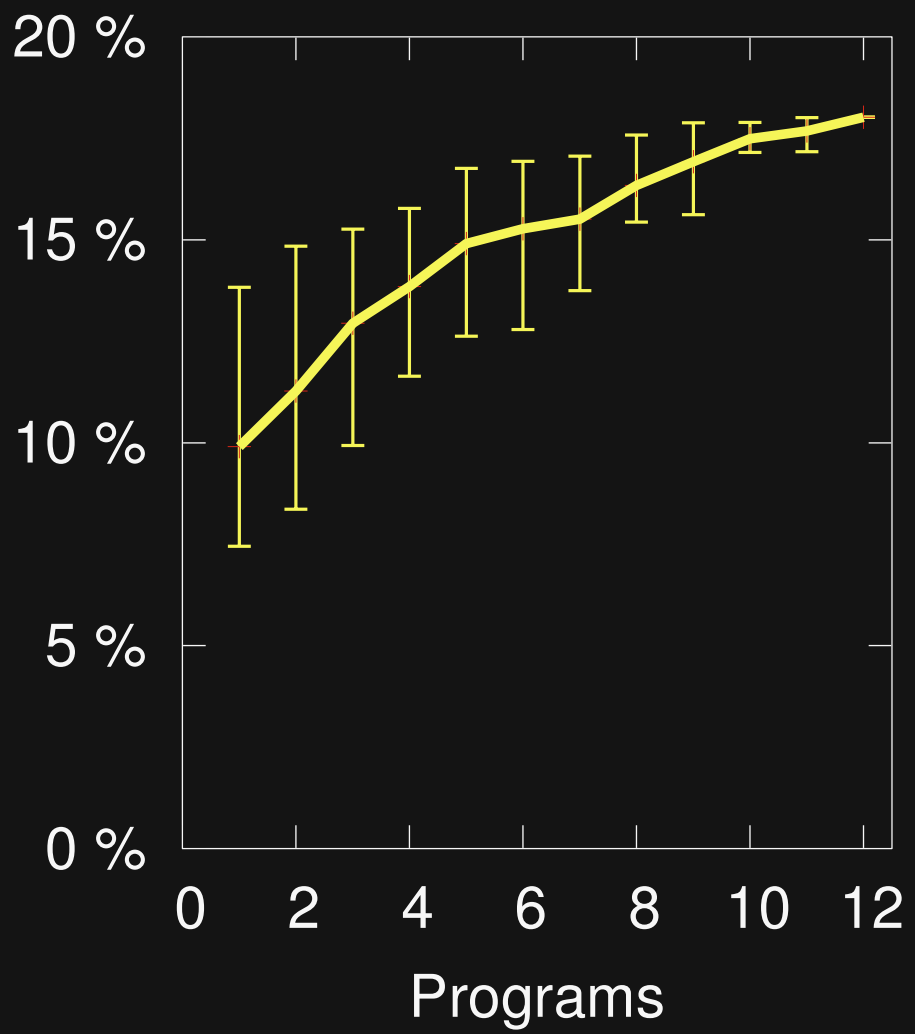
Recall

Recall of mined protocols



Recall

Recall of mined protocols



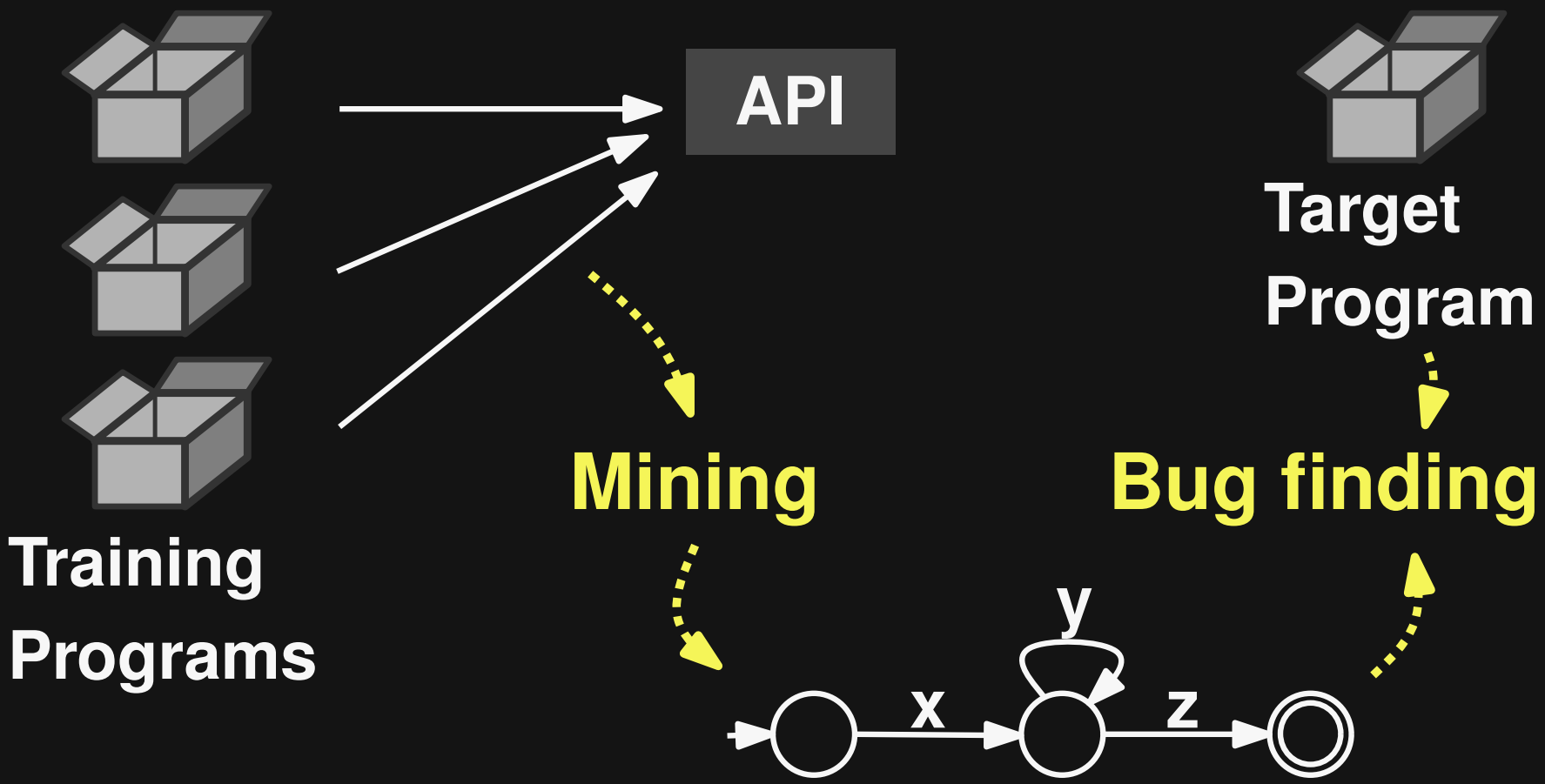
More programs

↓

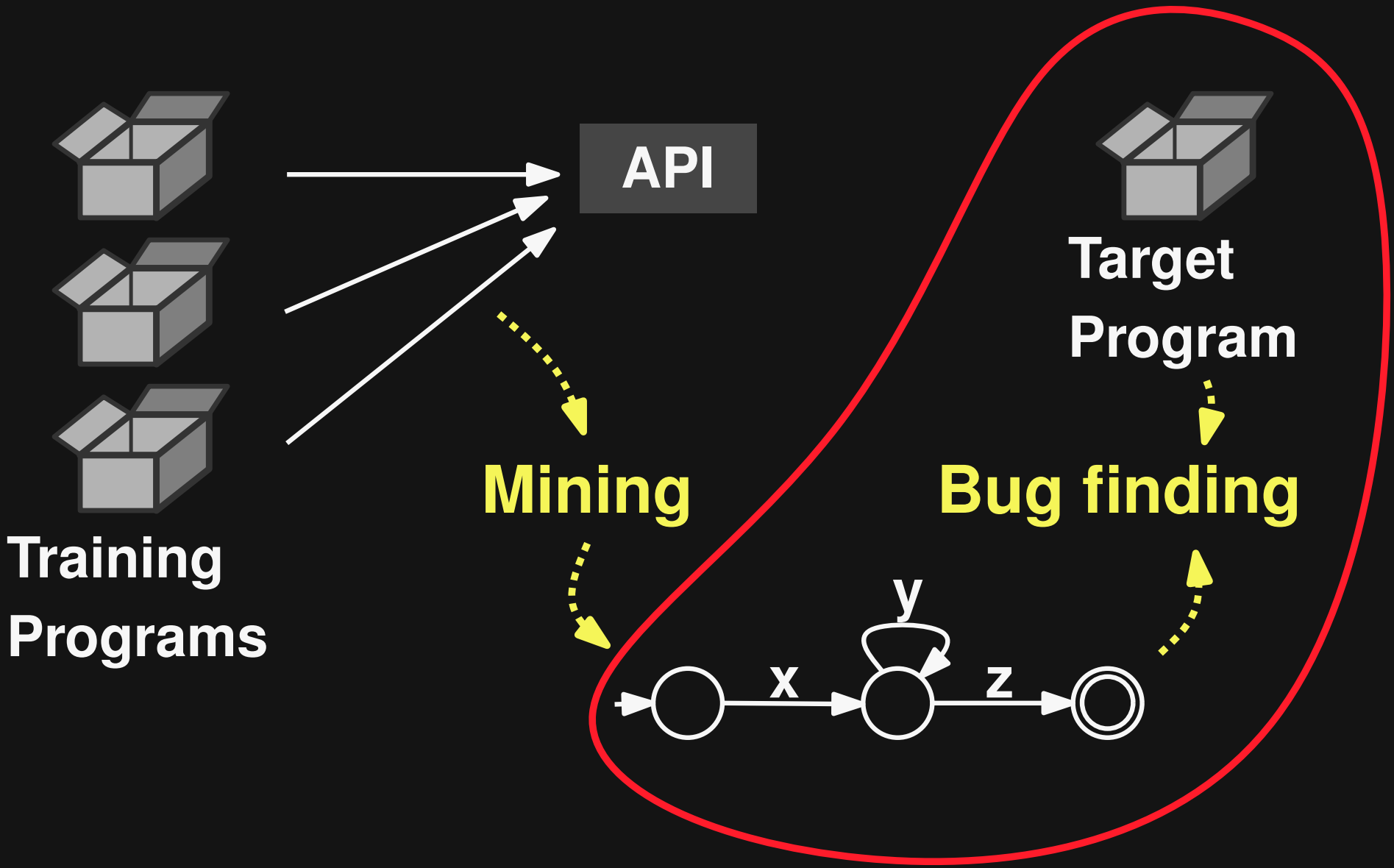
Higher recall

Part 2:
Bug Finding

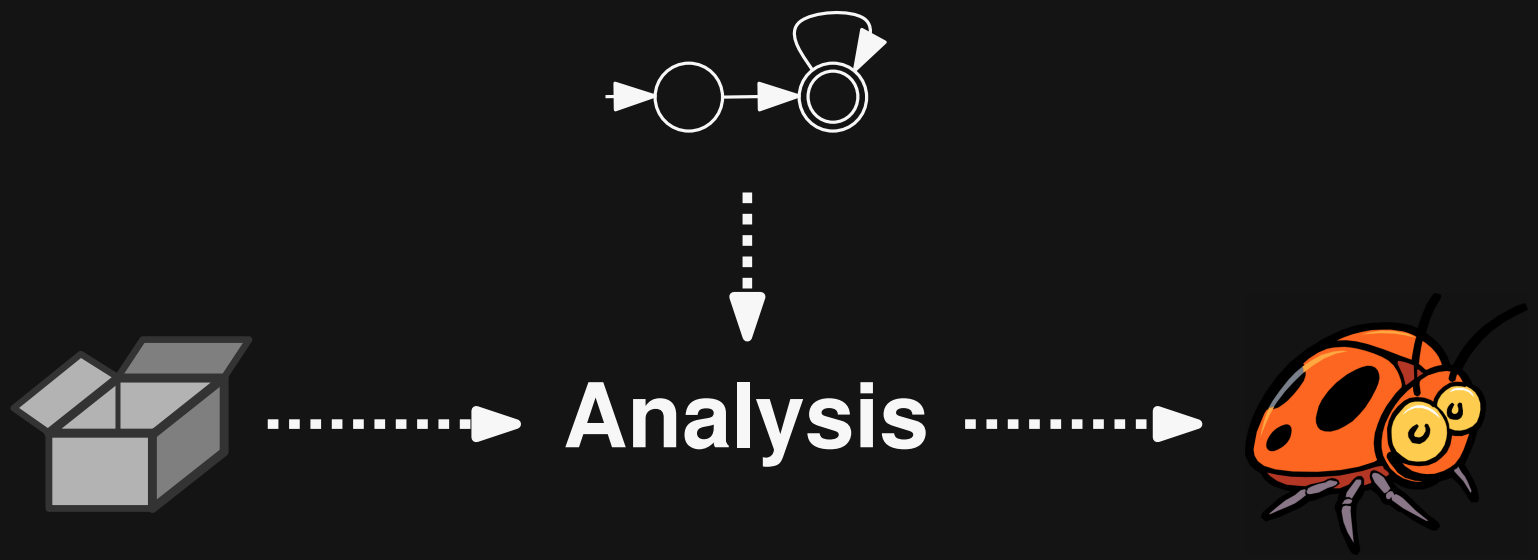
Bug Finding


API
Target Program
Training Programs
Mining
Bug finding
y
x
z

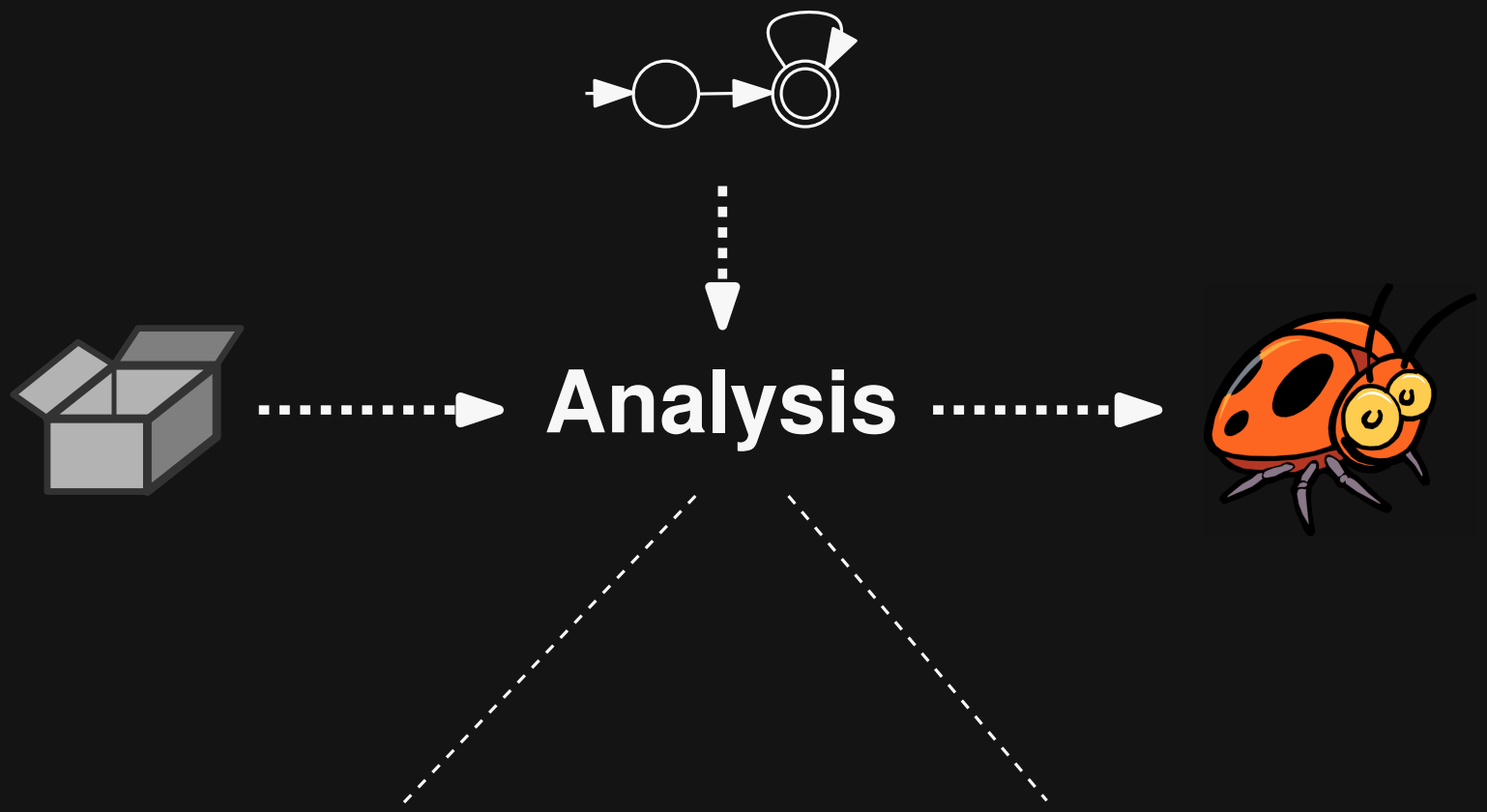
Bug Finding


API
Target
Program
Training
Programs
Mining
Bug finding
y
x
z

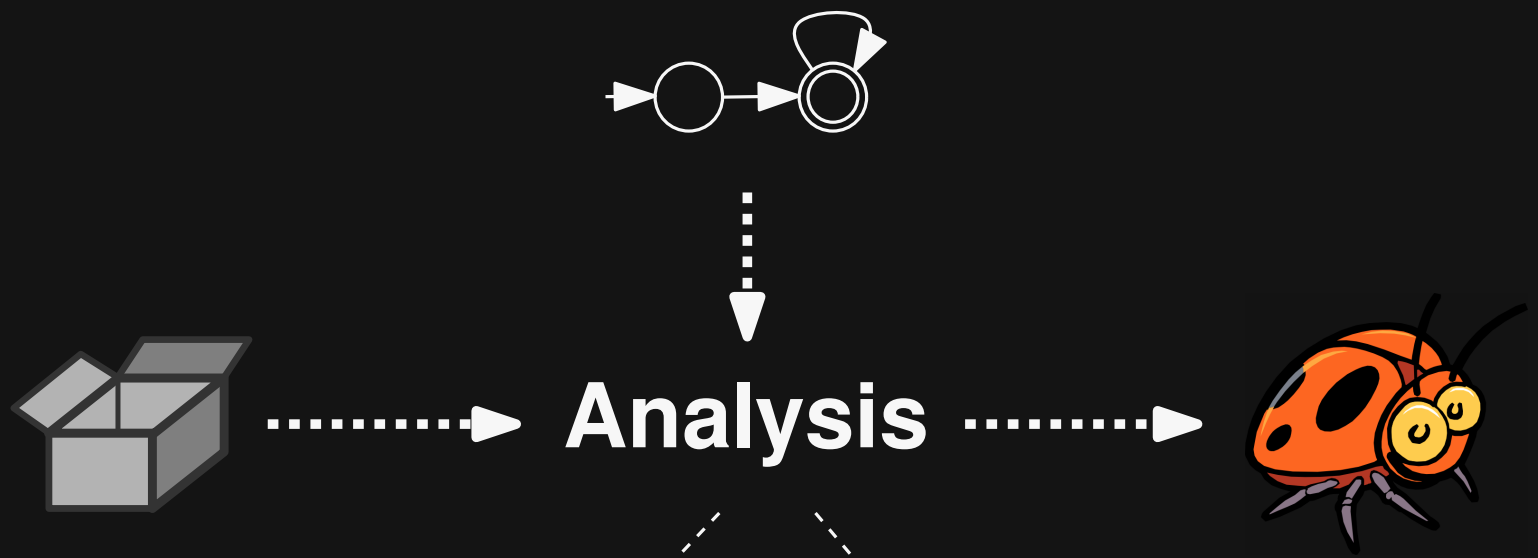
Overview



Overview



Overview

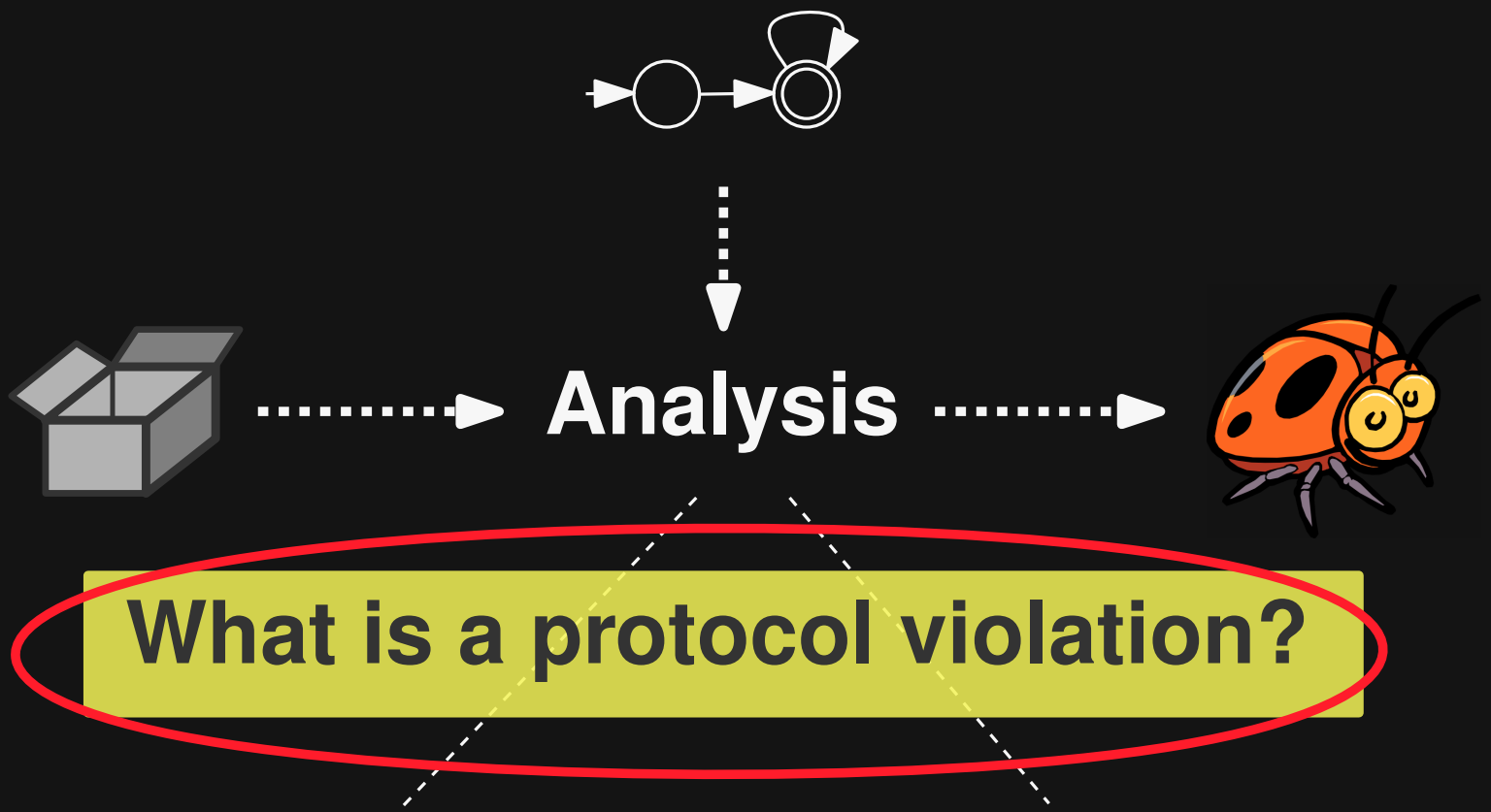


What is a protocol violation?

Runtime verification

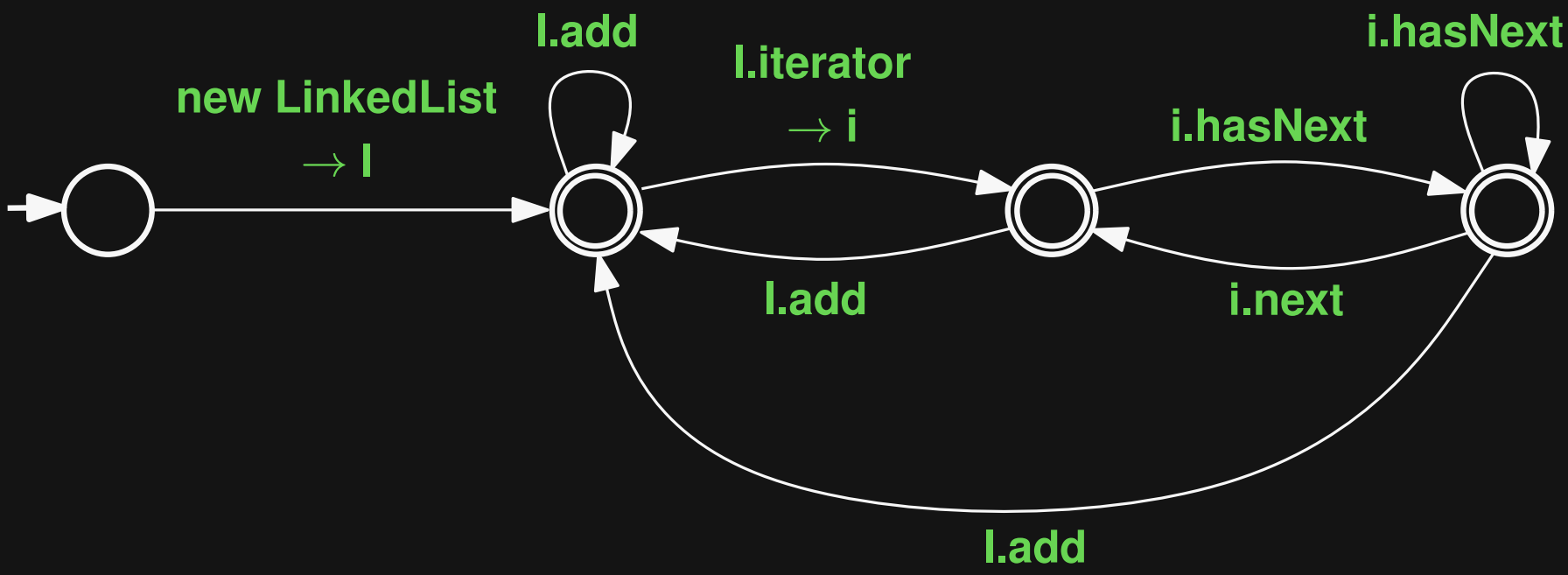
Static checking

Overview


Analysis
What is a protocol violation?
Runtime verification
Static checking

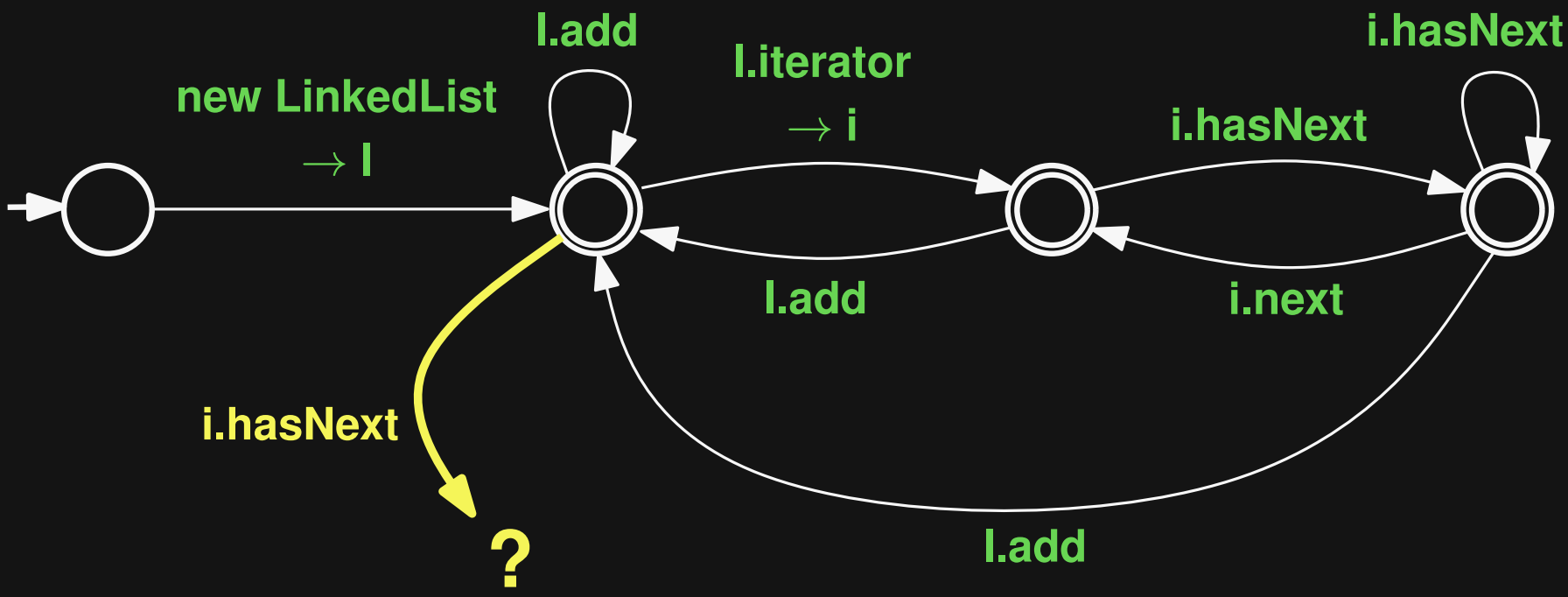
Incomplete Specifications

Protocols = Incomplete specifications

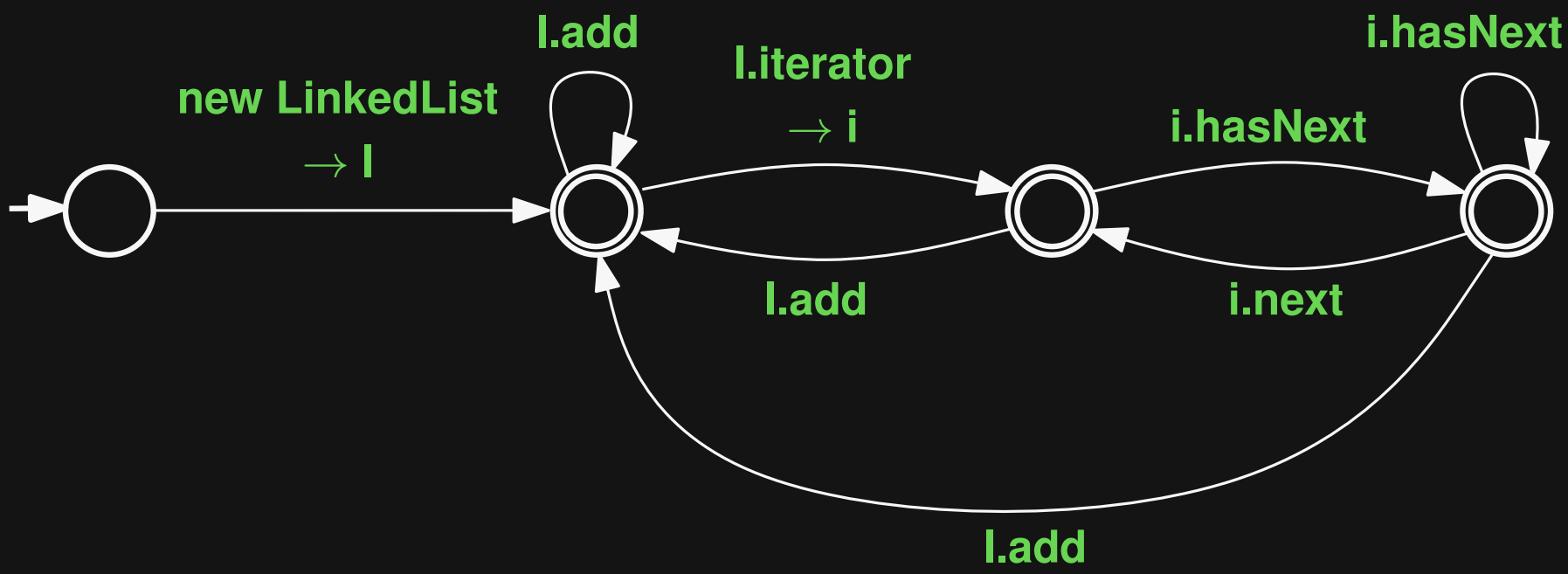


Incomplete Specifications

Protocols = Incomplete specifications



What is a Protocol Violation?

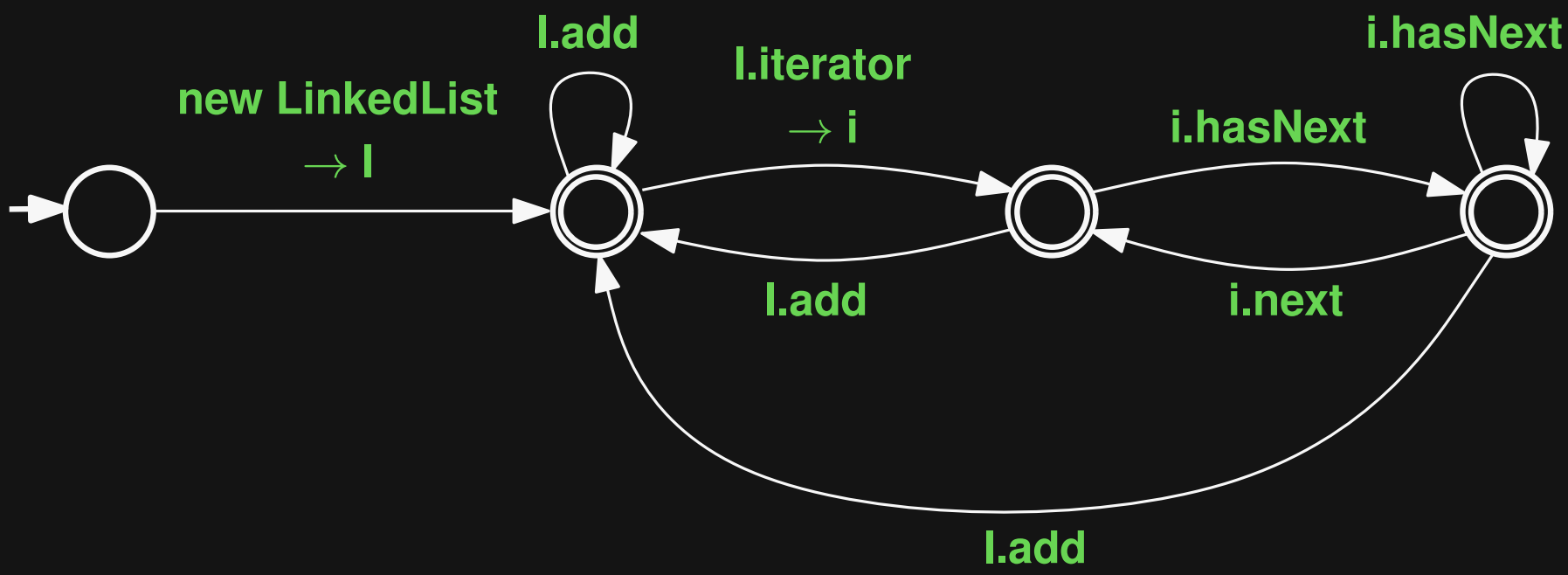


Setup phase:
- bind parameters

Liable phase:
- all parameters bound
- violation:
  - take non-existing transition
  - end in non-final state

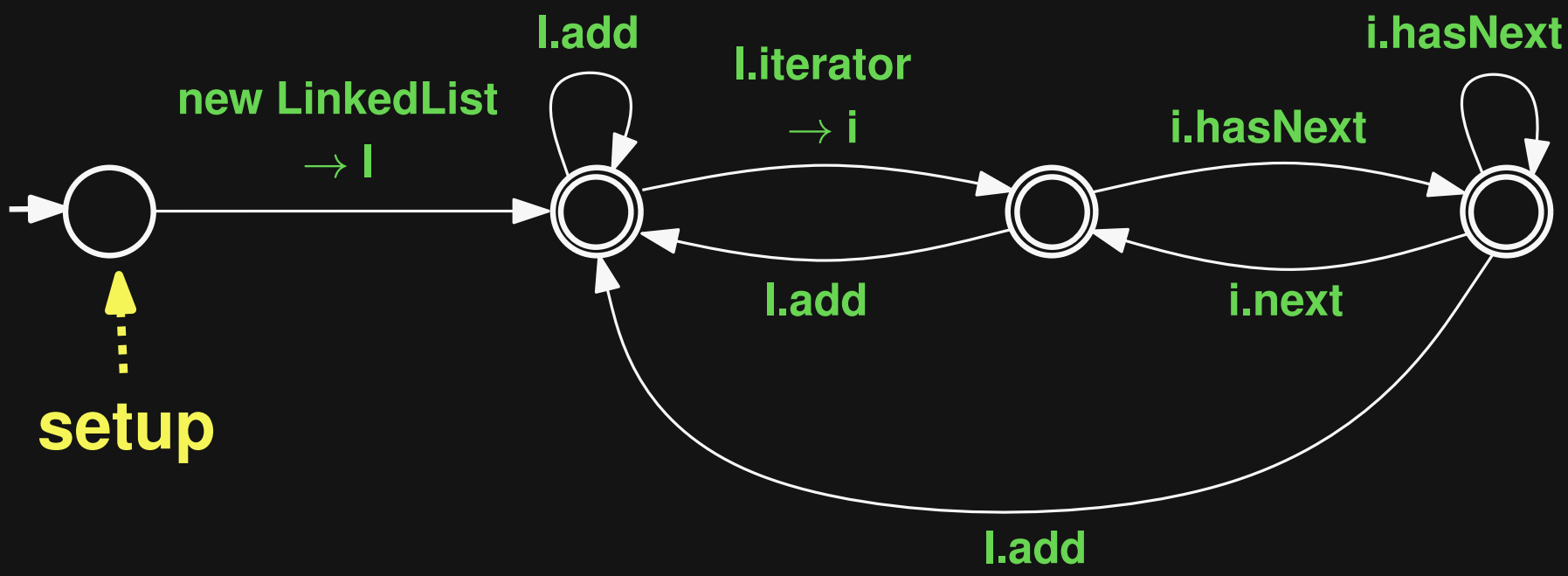
State Partitioning

Protocol transformation:
Setup states vs. liable states



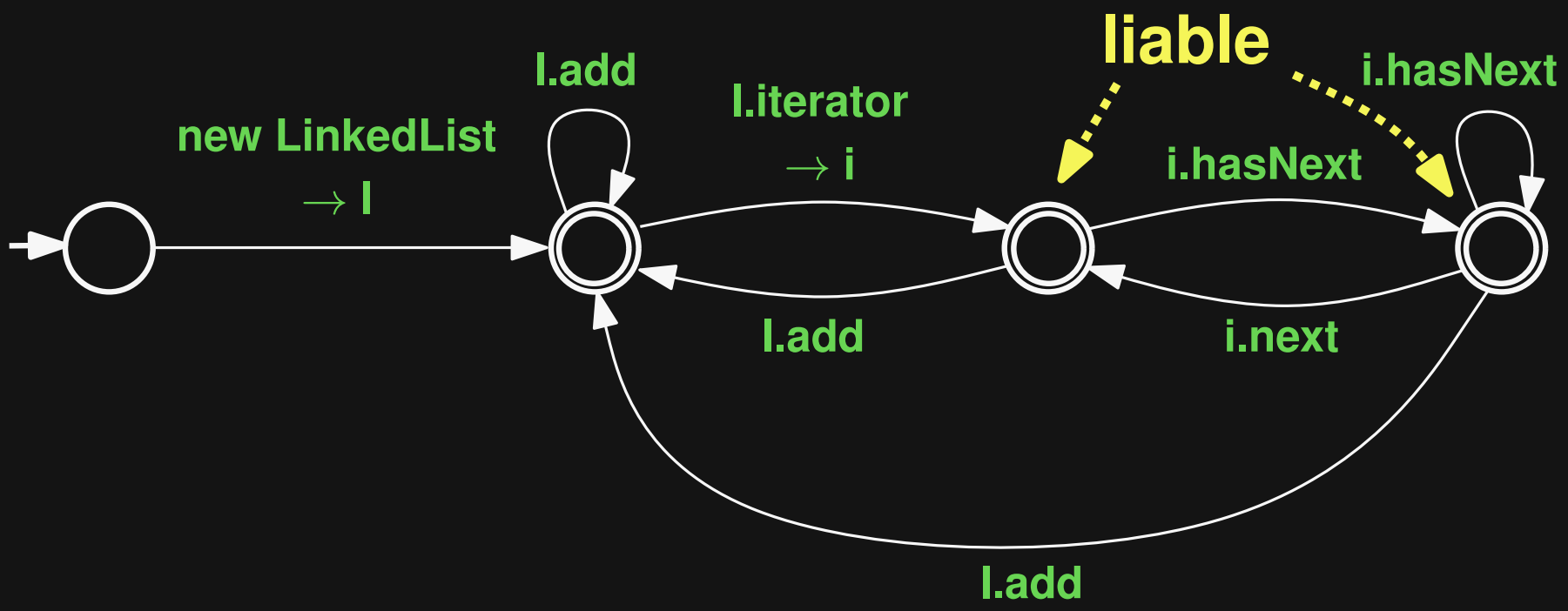
State Partitioning

Protocol transformation:
Setup states vs. liable states



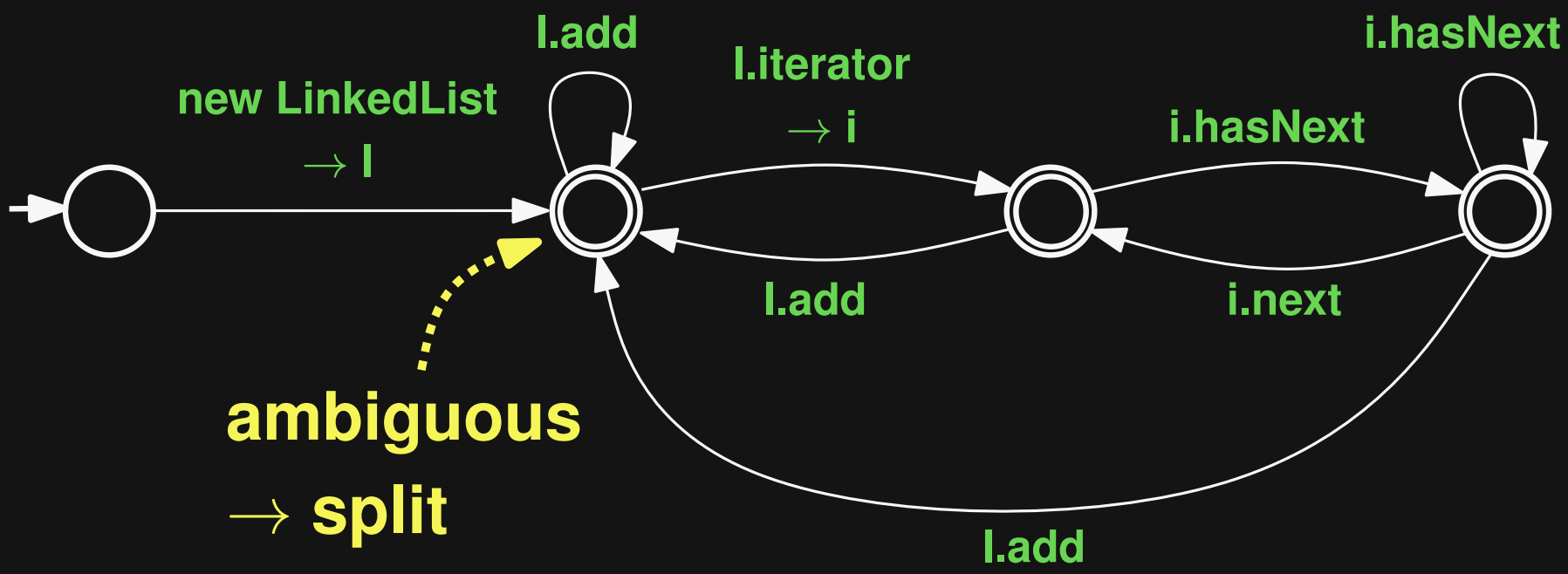
State Partitioning

Protocol transformation:
Setup states vs. liable states



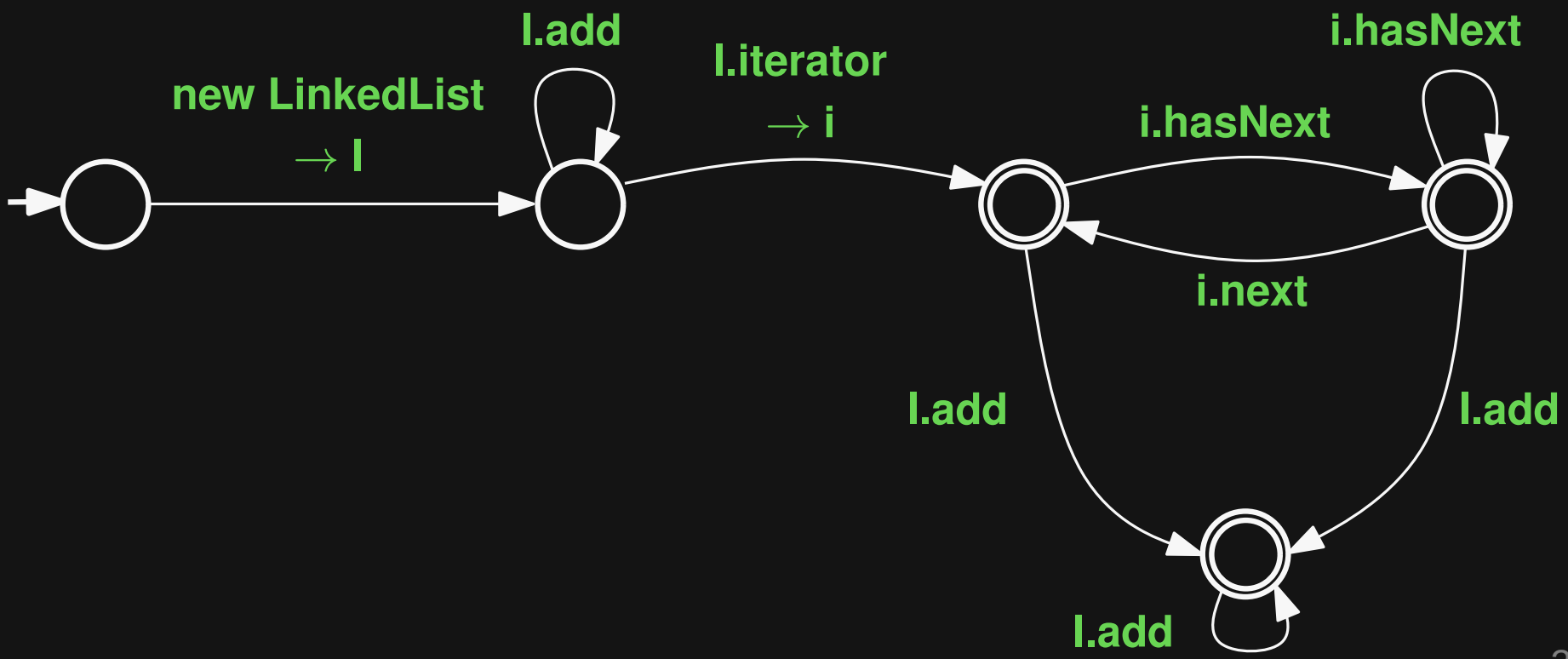
State Partitioning

Protocol transformation:
Setup states vs. liable states



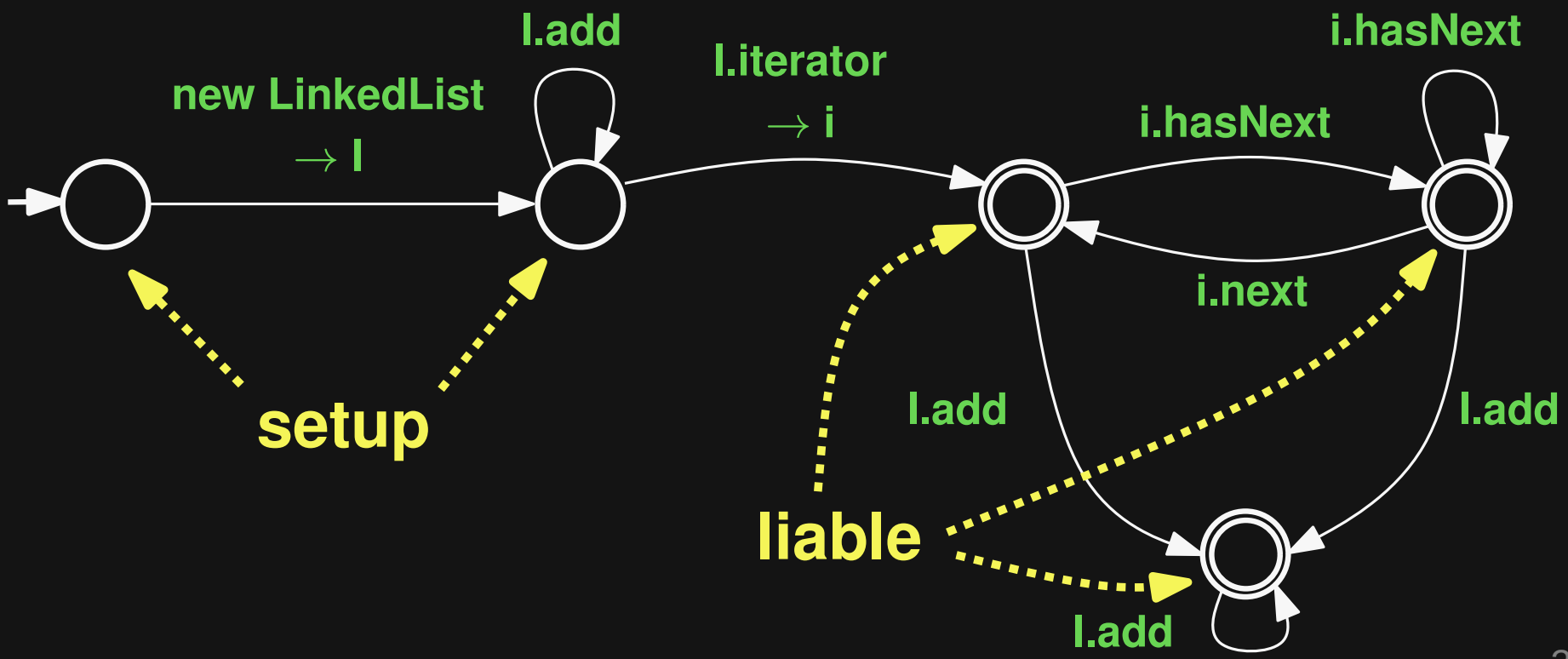
State Partitioning

Protocol transformation:
Setup states vs. liable states

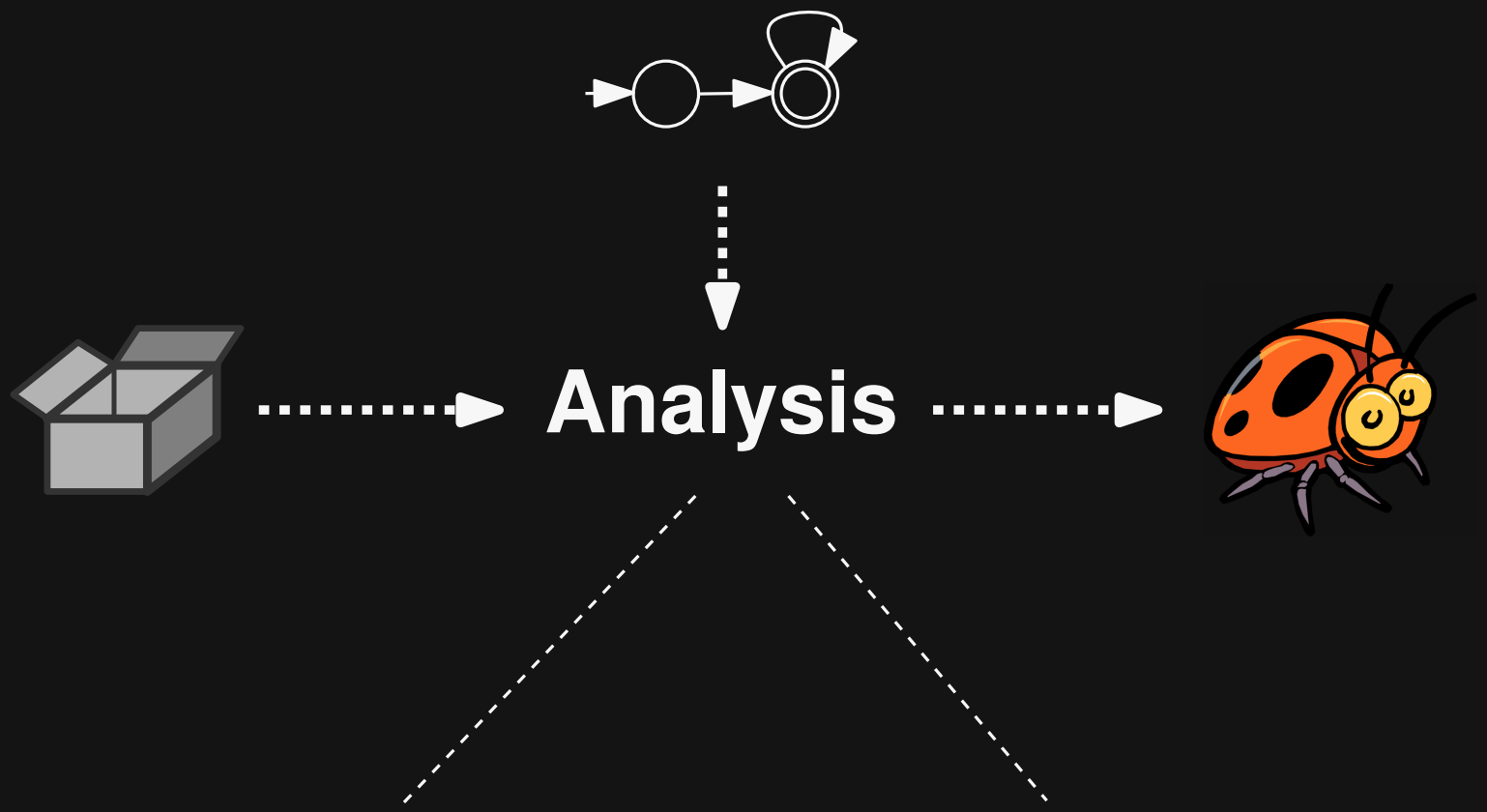


State Partitioning

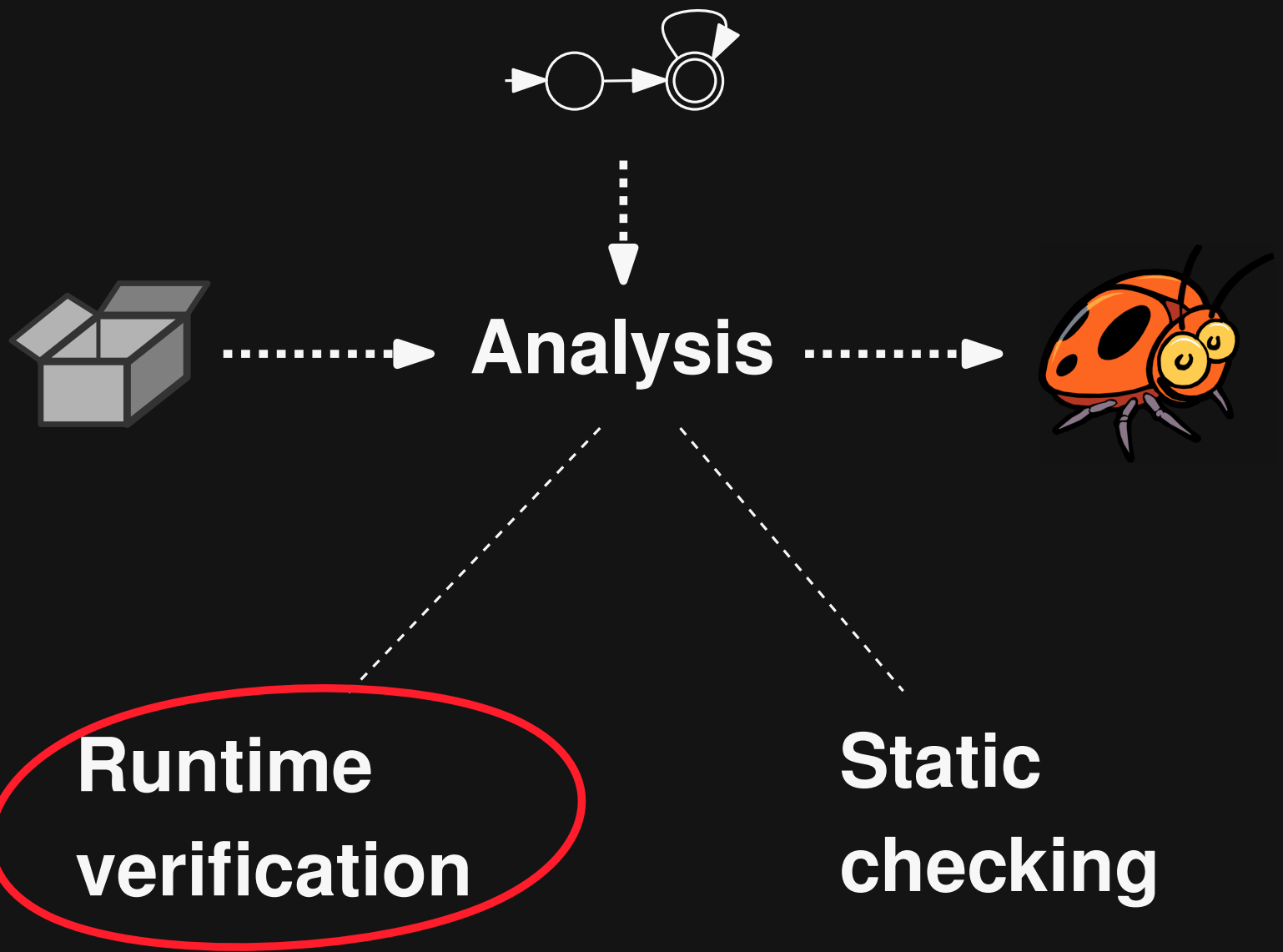
Protocol transformation:
Setup states vs. liable states



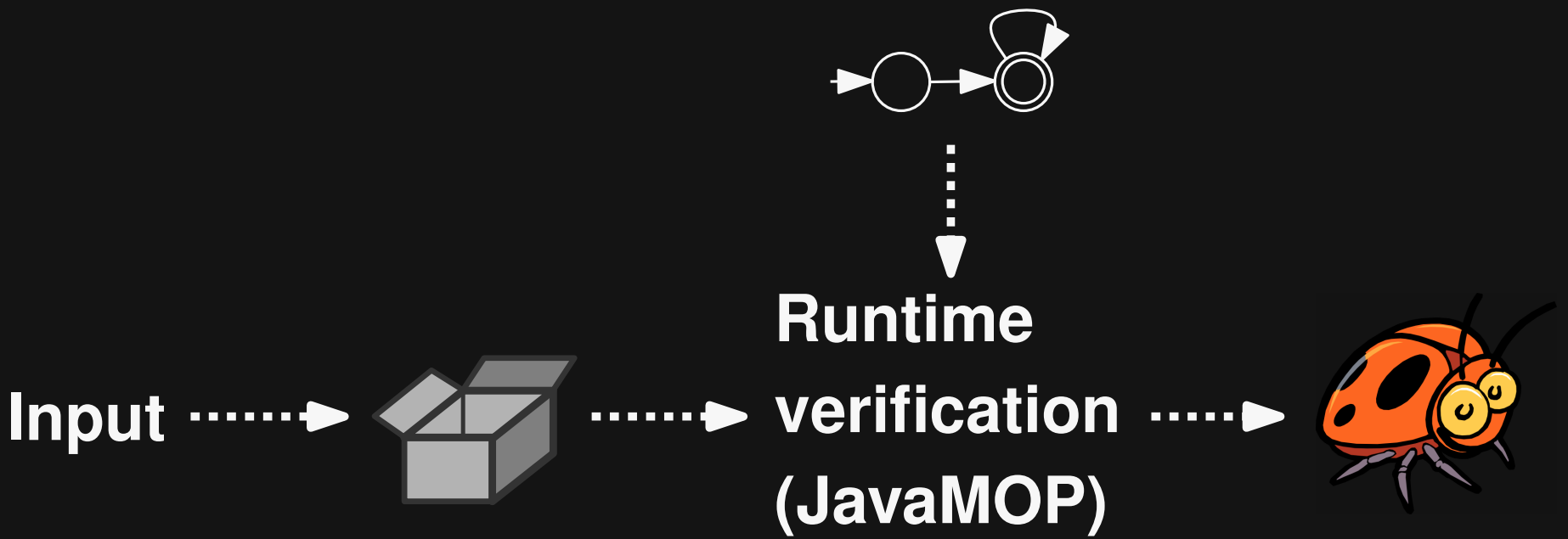
Overview



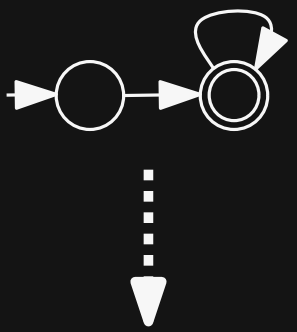
Overview



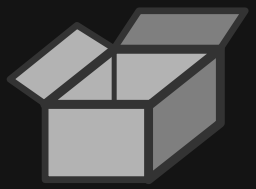
Dynamic Protocol Checking



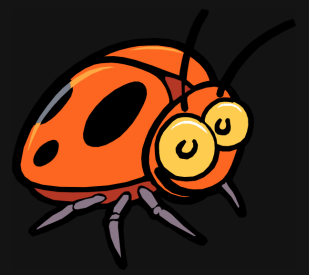
Dynamic Protocol Checking



Input

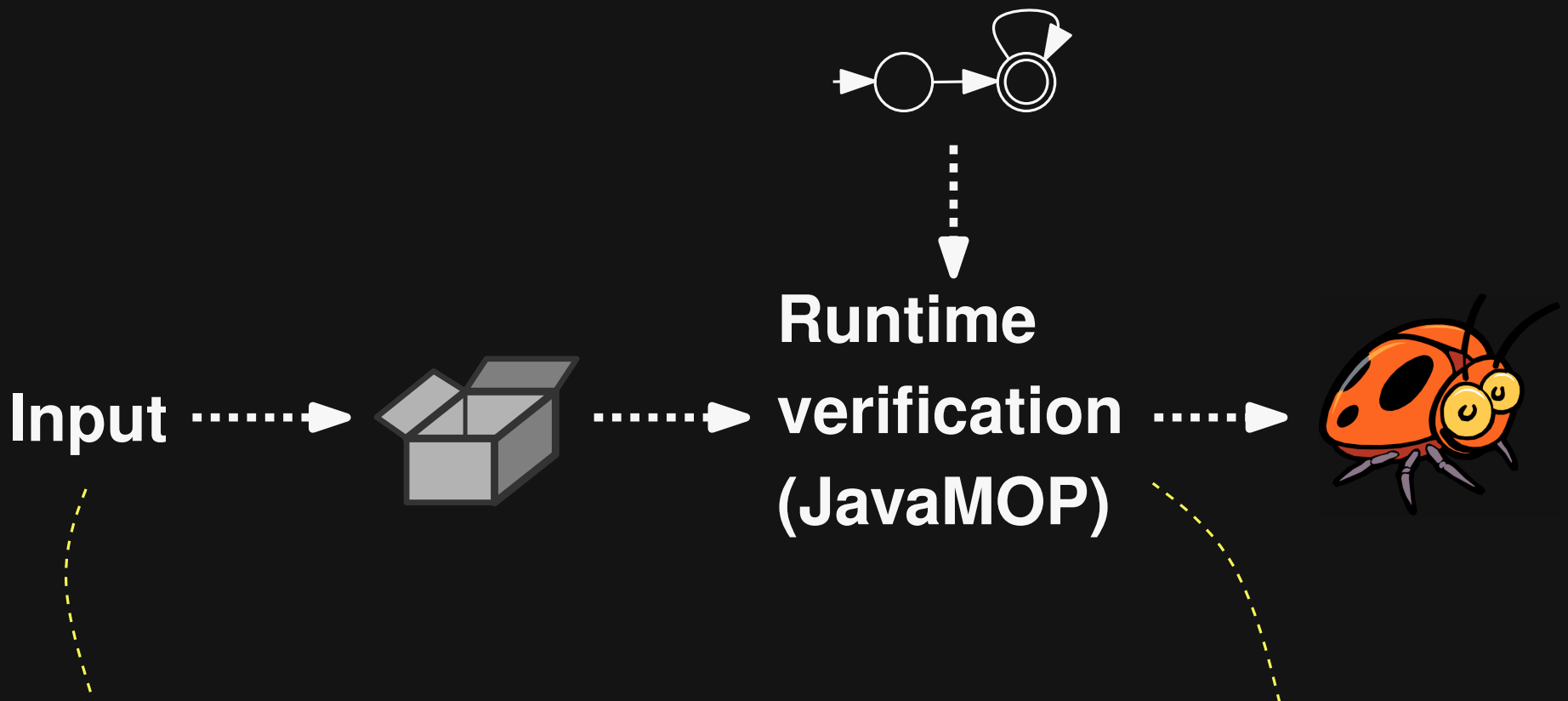


Runtime verification (JavaMOP)



Challenge 1:
Check many different execution paths

Dynamic Protocol Checking

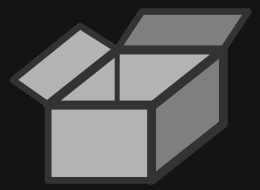


Challenge 1:
Check many different execution paths

Challenge 2:
Monitoring mined protocols

Randomly Generated Input

Challenge 1:
Check many different execution paths

Input 

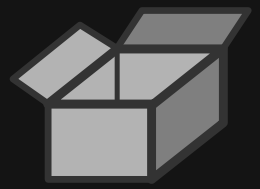
Randomly Generated Input

Challenge 1:
Check many different execution paths

Randomly Generated Input

Challenge 1:
Check many different execution paths

Input

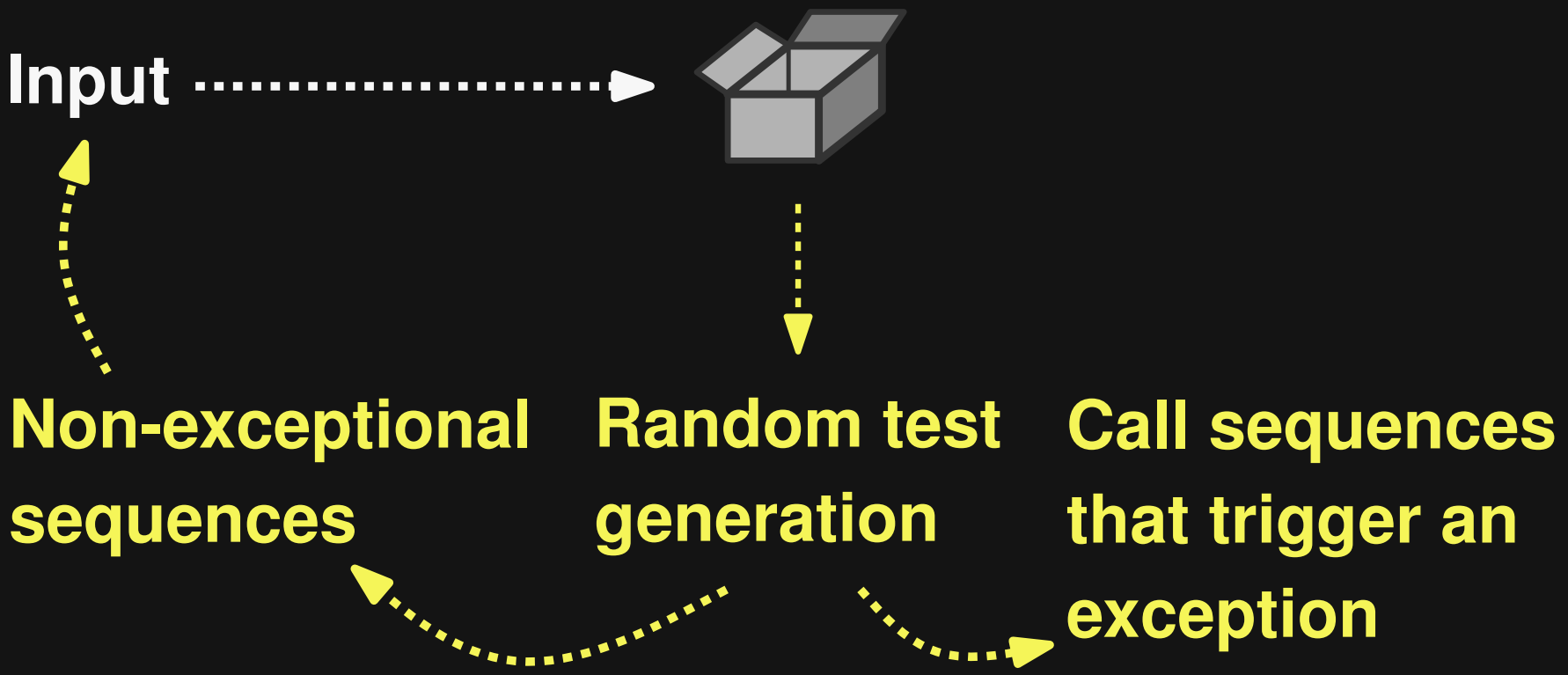


Random test generation

Call sequences that trigger an exception

Randomly Generated Input

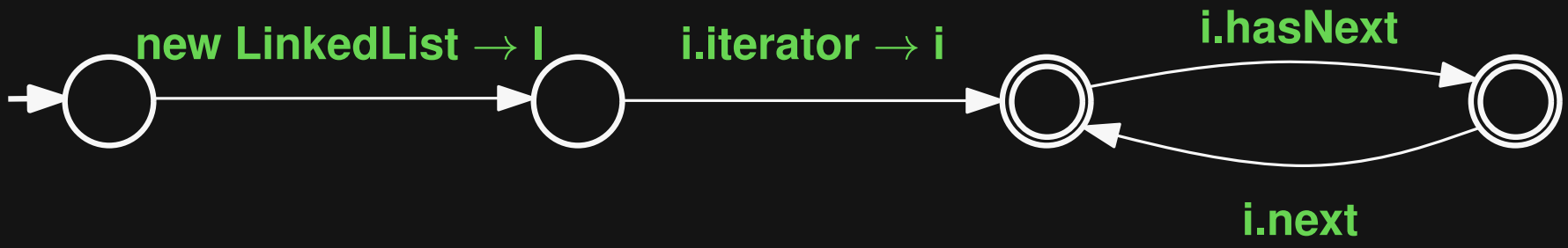
Challenge 1:
Check many different execution paths



Protocol Monitoring

Challenge 2:
Monitoring mined protocols

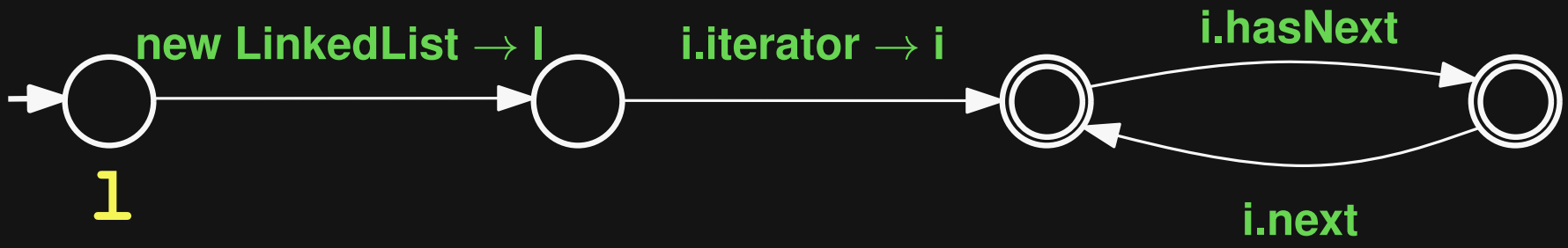
```
l = new LinkedList()
```



Protocol Monitoring

Challenge 2:
Monitoring mined protocols

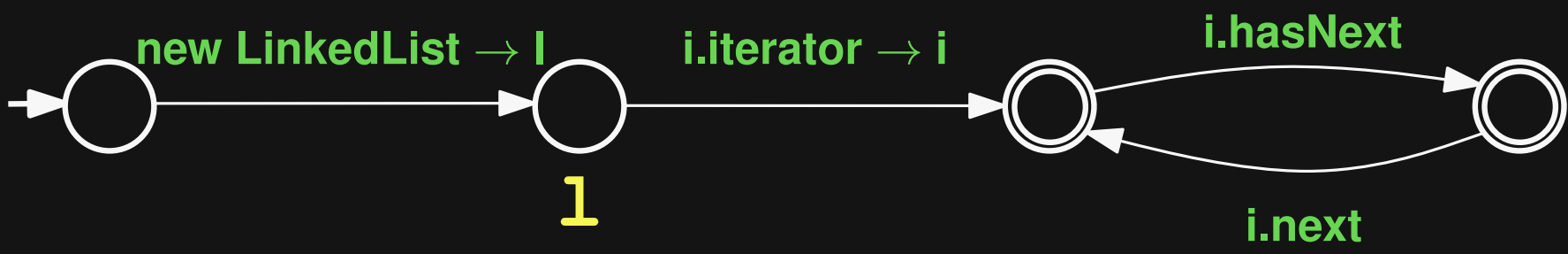
```
l = new LinkedList()
```



Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```
l = new LinkedList()
```

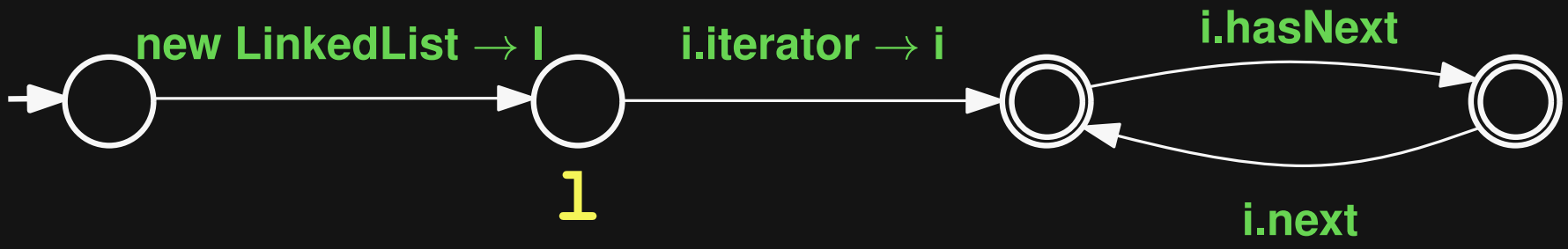


Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```
l = new LinkedList()
```

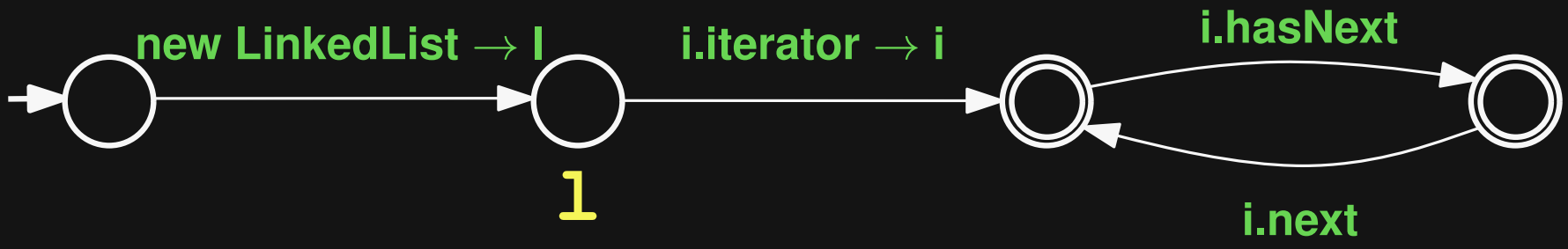
Violation!?



Protocol Monitoring

Challenge 2:
Monitoring mined protocols

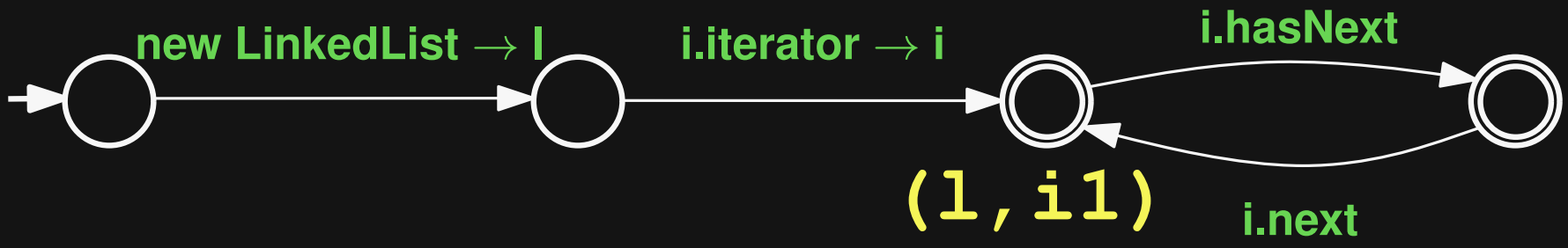
```
l = new LinkedList()
i1 = l.iterator()
i2.next()
```



Protocol Monitoring

Challenge 2:
Monitoring mined protocols

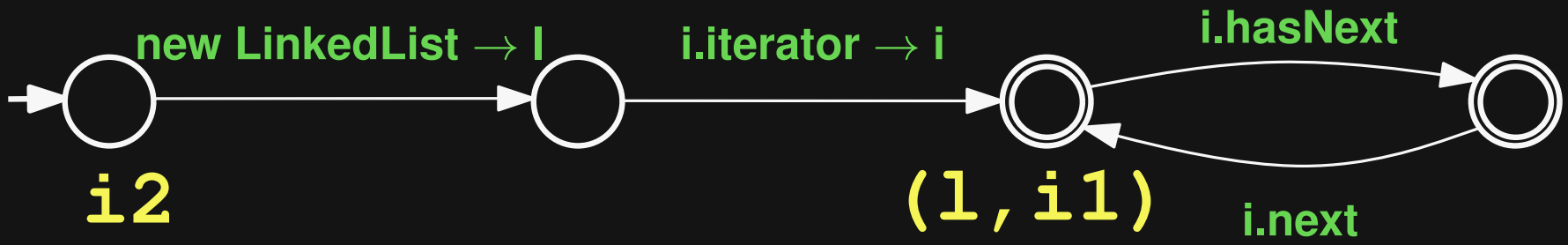
```
l = new LinkedList()
i1 = l.iterator()
i2.next()
```



Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```
l = new LinkedList()
i1 = l.iterator()
i2.next()
```

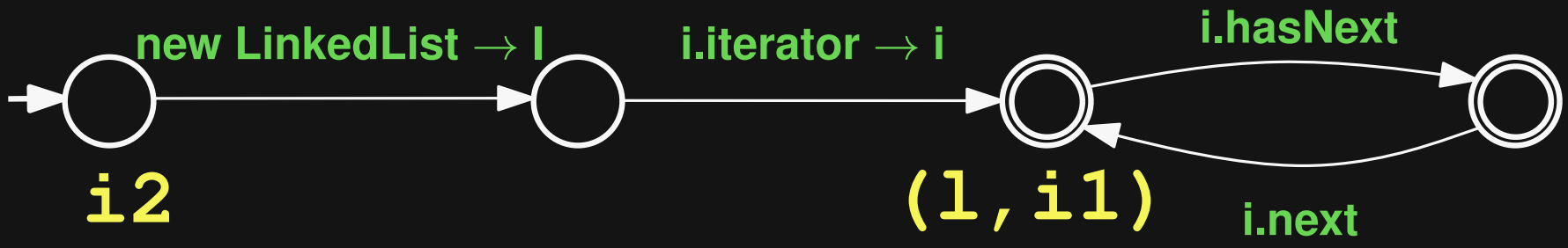


Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```
l = new LinkedList()
i1 = l.iterator()
i2.next()
```

Violation!?



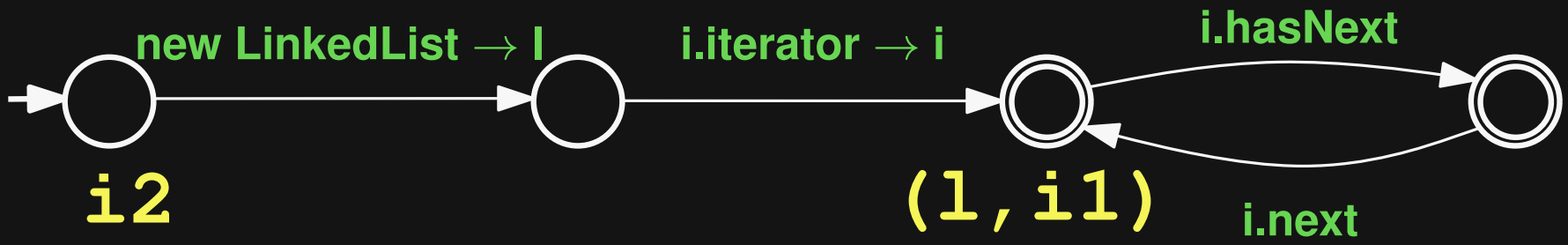
Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```
l = new LinkedList()
i1 = l.iterator()
i2.next()
```

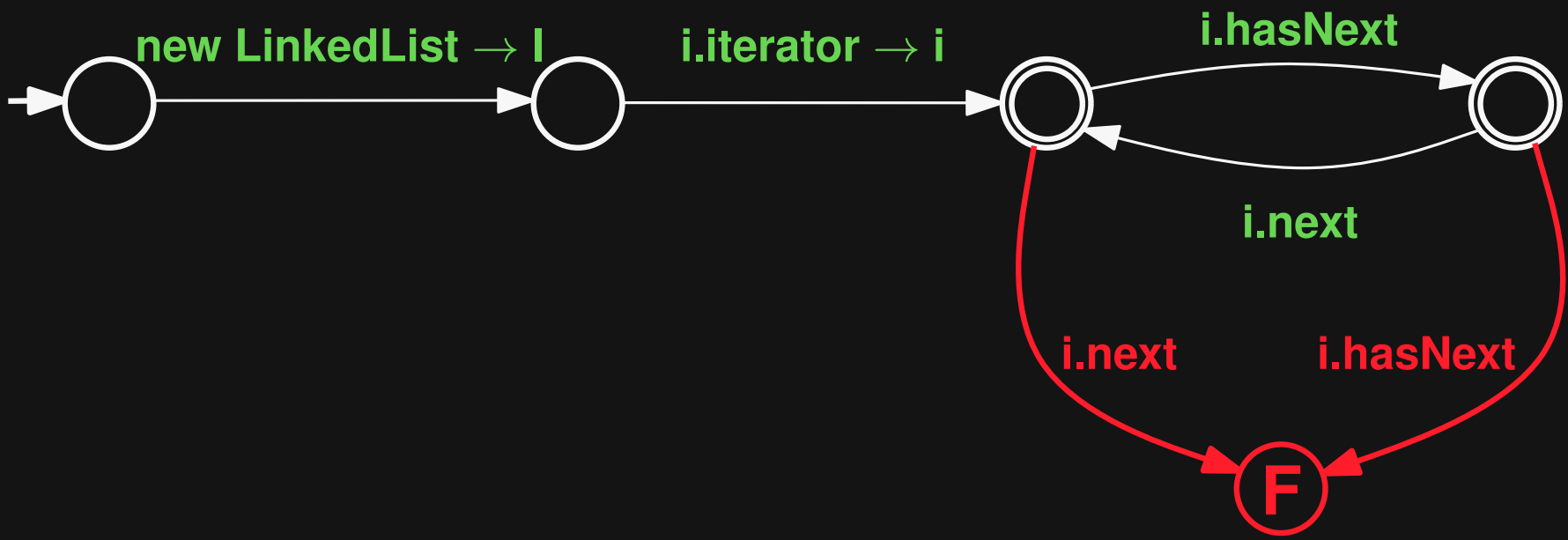
Violation!?

Naive approach gives too many violations



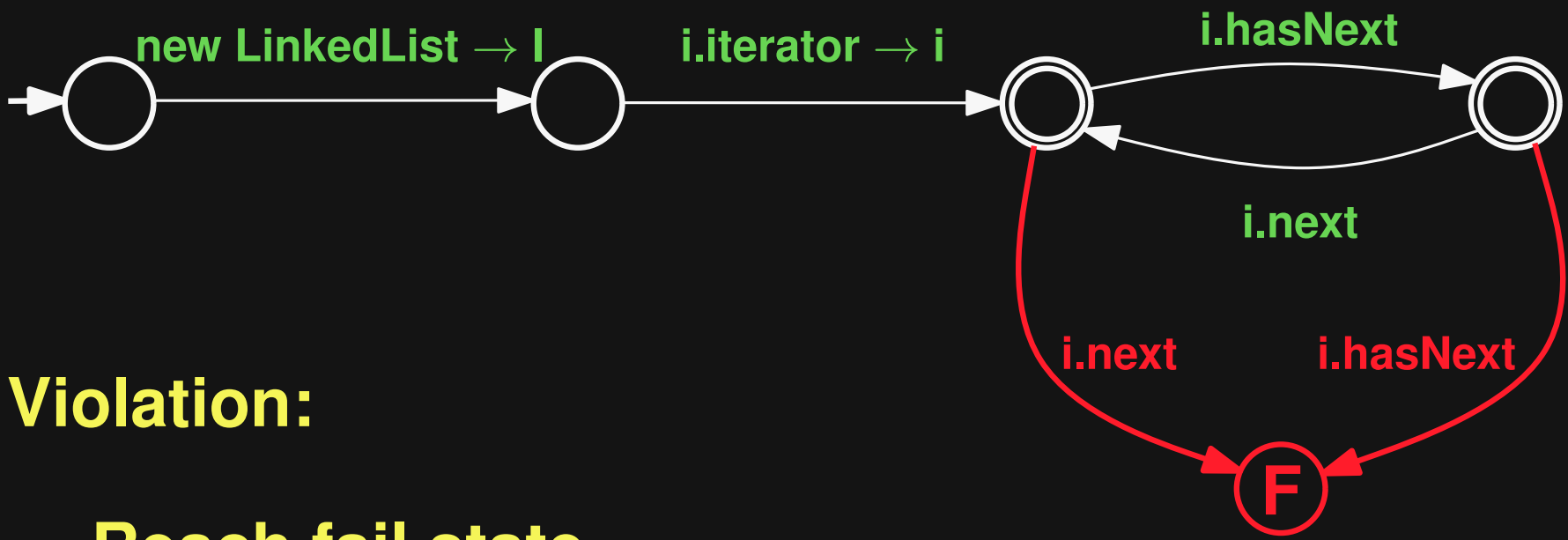
Explicit Fail Transitions

Fail only in liable states



Explicit Fail Transitions

Fail only in liable states



Violation:

- Reach fail state
- End in non-final, liable state

Evaluation

Questions

- Find relevant issues by monitoring mined protocols?
- How useful is generated input?

Setup:

DaCapo benchmarks, 1.6 MLOC Java

Results

	Randomly generated		
		Protocol violations	
Program	Test cases	Total	Relevant
avrora	15,753	5	4
batik	3,477	0	0
daytrader	32,446	0	0
eclipse	816	0	0
fop	6,536	52	50
h2	7,584	14	7
lucene	1,985	0	0
pmd	1,286	0	0
sunflow	4,300	1	0
tomcat	14,627	1	1
xalan	21,083	1	1
Sum	160,857	74	63

Bug (exception, unexpected behavior) or code smell (performance/maintainability problem)

Results

	Randomly generated		
		Protocol violations	
Program	Test cases	Total	Relevant
avrora	15,753	5	4
batik	3,477	0	0
daytrader	32,446	0	0
eclipse	816	0	0
fop	6,536	52	50
h2	7,584	14	7
lucene	1,985	0	0
pmd	1,286	0	0
sunflow	4,300	1	0
tomcat	14,627	1	1
xalan	21,083	1	1
Sum	160,857	74	63

Bug (exception, unexpected behavior) or code smell (performance/maintainability problem)

85% true positives

Examples

```
try {
  is = u.openStream();
  r = new InputStreamReader(is, "UTF-8");
  br = new BufferedReader(r);
} finally {
  if ( is != null ){
    try { is.close(); } catch ( IOException ignored ){}
    is = null;
  }
  if ( r != null ){
    try{ r.close(); } catch ( IOException ignored ){}
    r = null;
  }
  if ( br == null ){
    try{ br.close(); } catch ( IOException ignored ){}
    br = null;
  }
}
```

Examples

```
try {
  is = u.openStream();
  r = new InputStreamReader(is, "UTF-8");
  br = new BufferedReader(r);
} finally {
  if ( is != null ){
    try { is.close(); } catch ( IOException ignored ){}
    is = null;
  }
  if ( r != null ){
    try{ r.close(); } catch ( IOException ignored ){}
    r = null;
  }
  if ( br == null ){
    try{ br.close(); } catch ( IOException ignored ){}
    br = null;
  }
}
```

Reader never closed

Examples

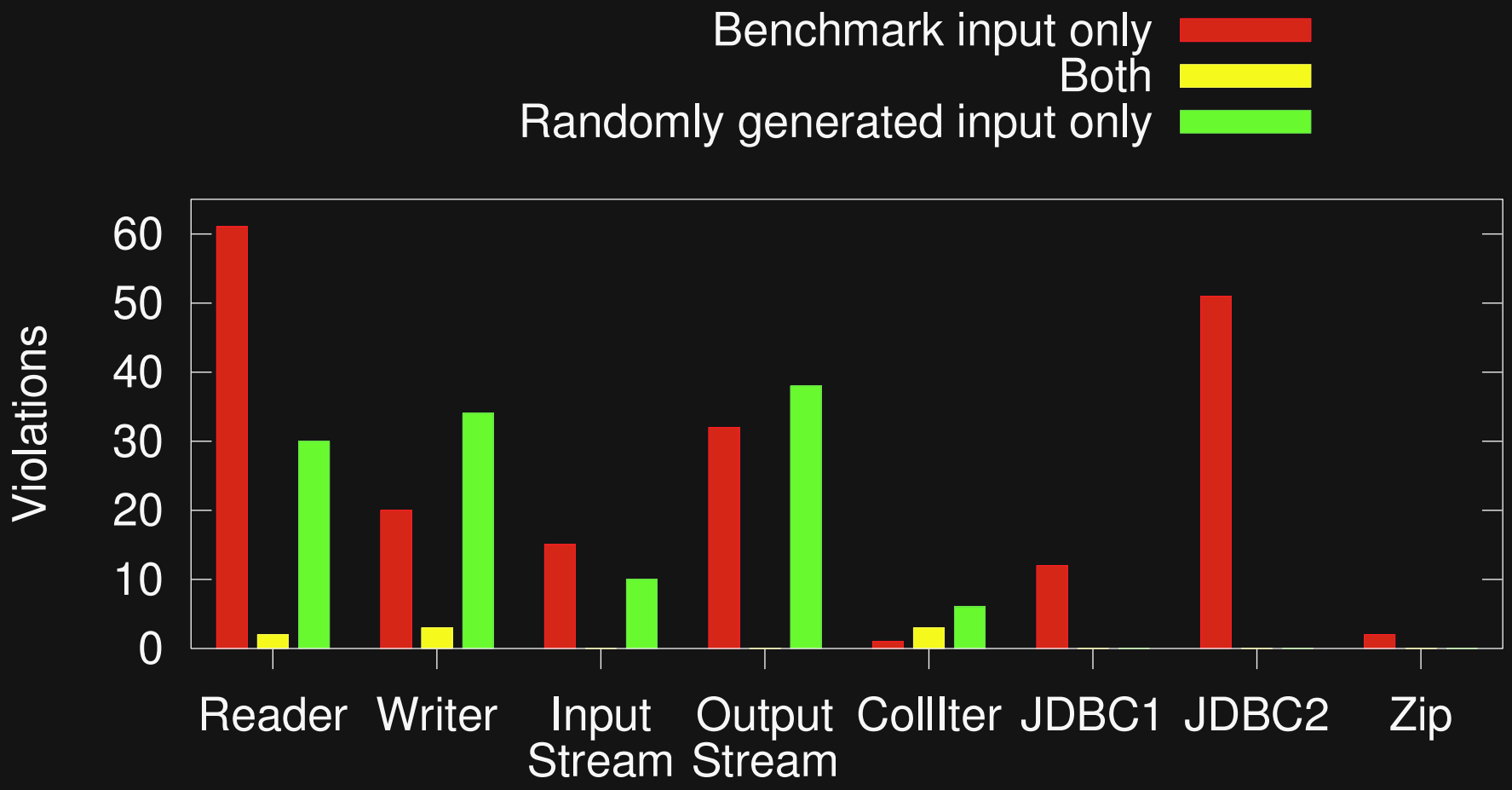
```
Iterator i = pinConnections.iterator();

PinLink currLink = (PinConnect.PinLink) i.next();
currLink.propagateSignals();

while (i.hasNext()) {
  currLink = (PinConnect.PinLink) i.next();
  currLink.propagateSignals();
}
```

Examples

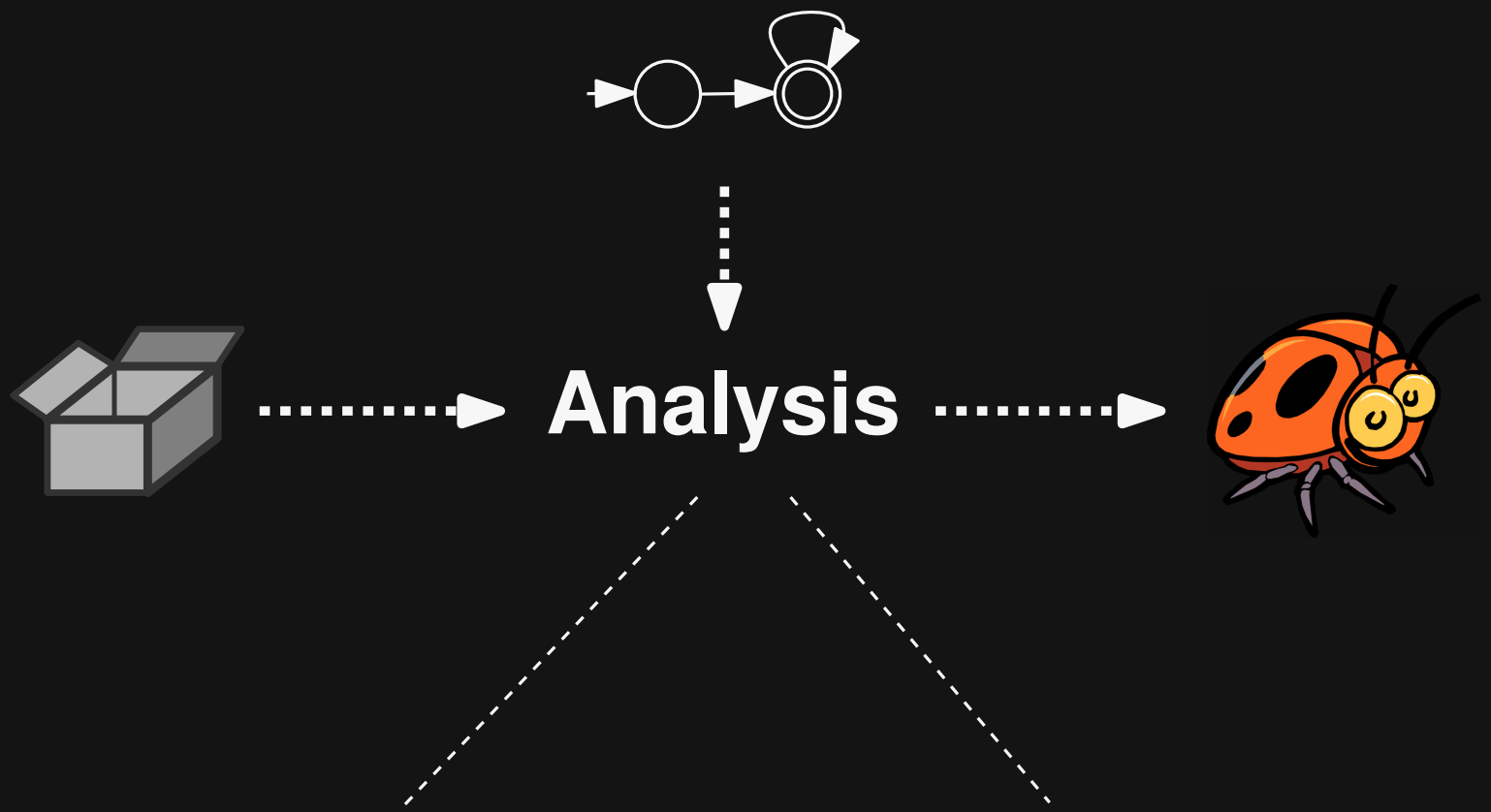
```
Iterator i = pinConnections.iterator();

PinLink currLink = (PinConnect.PinLink) i.next();
currLink.propagateSignals();

while (i.hasNext()) {
  currLink = (PinConnect.PinLink) i.next();
  currLink.propagateSignals();
}
```

Incorrect iterator usage

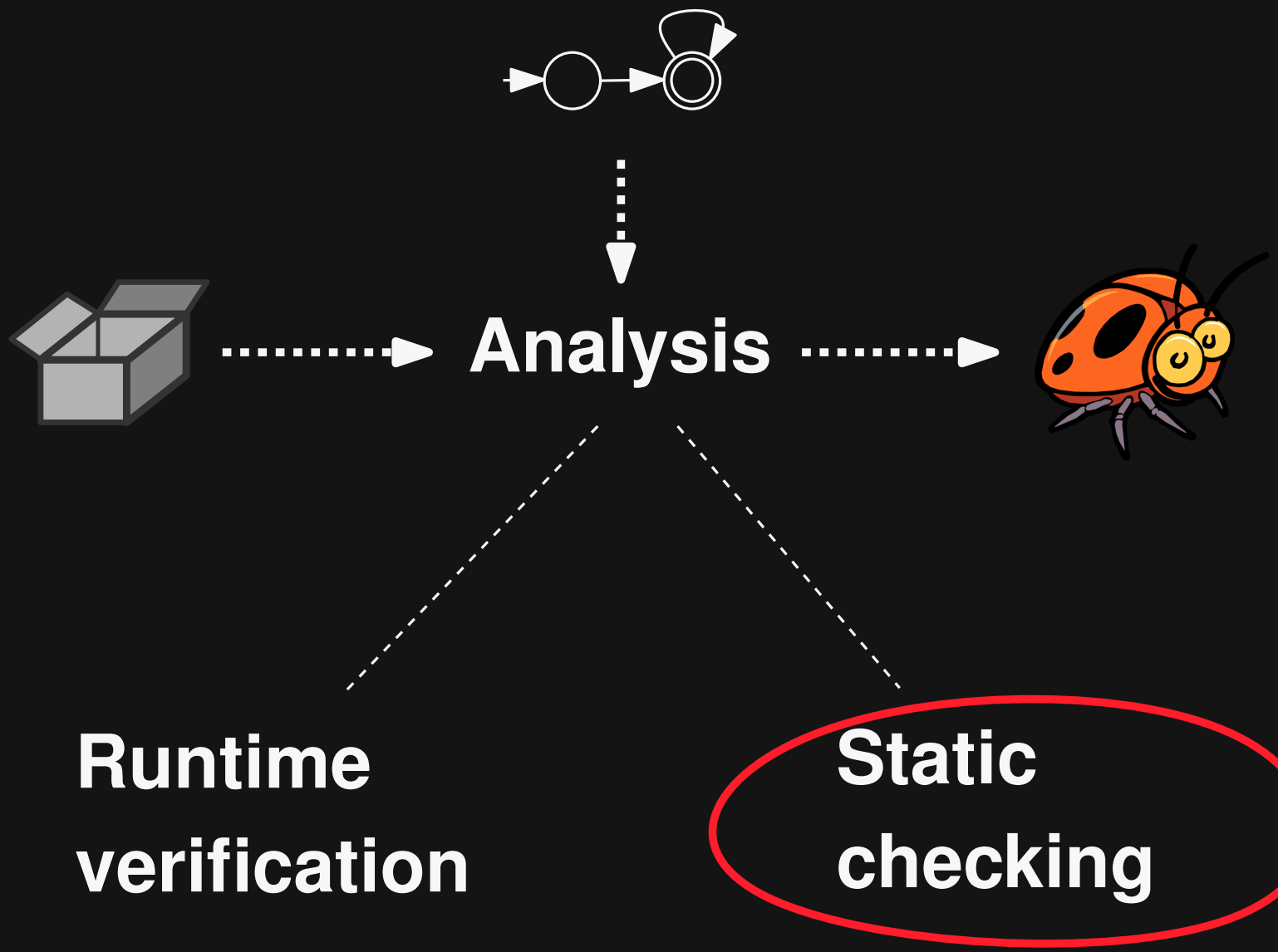
Normal vs. Generated Input



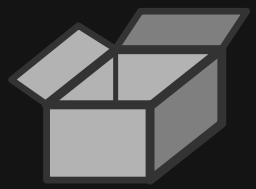
Overview



Overview

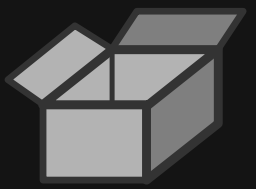


State of the Art



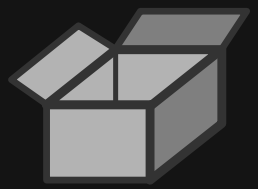
+ specification

Typestate checking



Anomaly detection

State of the Art

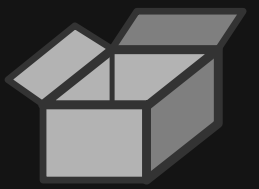


+ specification

Typestate checking

+ Precise

- Needs specification

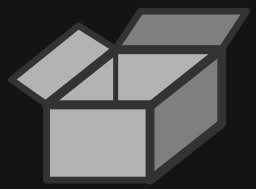


Anomaly detection

+ Automatic

- Imprecise

State of the Art

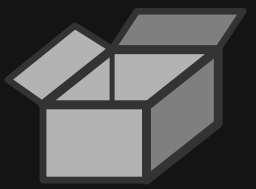


+ specification

Typestate checking

+ Precise

- Needs specification



Anomaly detection

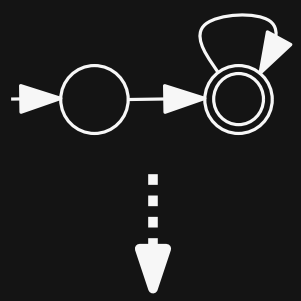
+ Automatic

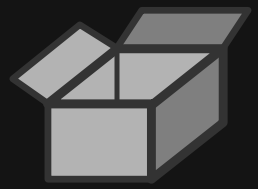
- Imprecise

Combine both!

Precise checker for mined multi-object protocols

Static Protocol Checking





```
@Constraint(
  requires="..."
)
```

Relationship constraints

Fusion analysis

f() ✓
g() ✗
h() ✗

Pruning

f()
g()
h()

Joint work with Ciera Jaspan and Jonathan Aldrich (ISR, CMU)

Fusion

Static analysis to check API usages

Good match with mined protocols:

- Reasons about **interacting objects**
- Distinguishes setup from checking

C. Jaspan and J. Aldrich
Checking Framework Interactions with Relationships
ECOOP ’09

Relationship-based Analysis

Relationships = Tuples of objects

```
void m() {
  ..
}
```

Effects: Add/remove objects

Requirements: Check before call

Relationship-based Analysis

Relationships = Tuples of objects

```
void m() {
   ..
}
```

Keep track of protocol execution (e.g., current state)

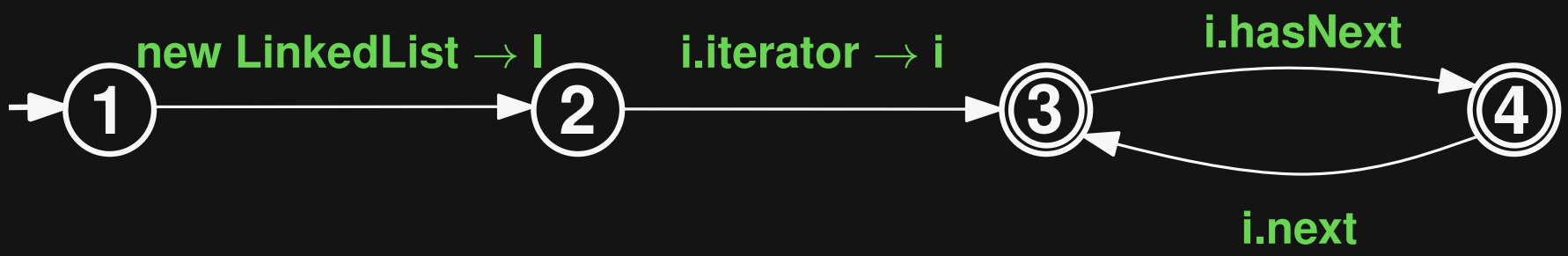
Effects: Add/remove objects

Requirements: Check before call

Check protocol constraints if in liable state

Example

```
LinkedList l = new LinkedList();

Iterator i = l.iterator();

i.next();
```



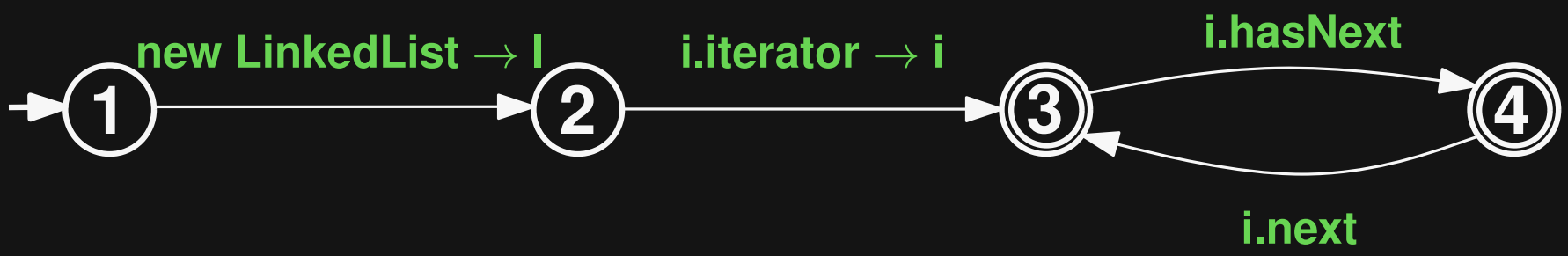
Example

```
LinkedList l = new LinkedList();
```

$$l \in r_{state2}, l \in r_{iterator}$$

```
Iterator i = l.iterator();
```

```
i.next();
```



Example

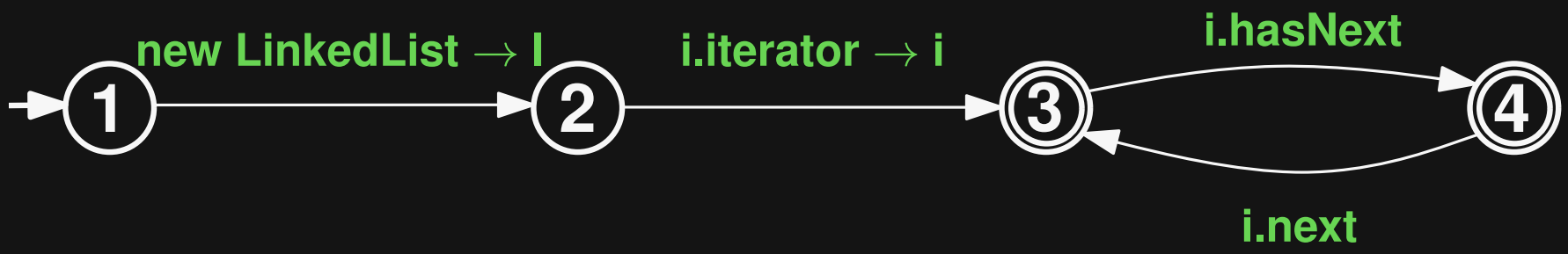
```
LinkedList l = new LinkedList();
```

$$l \in r_{state2}, l \in r_{iterator}$$

```
Iterator i = l.iterator();
```

$$l \in r_{state3}, i \in r_{state3}, i \in r_{hasNext}, (l, i) \in r_{protocol}$$

```
i.next();
```



Example

```
LinkedList l = new LinkedList();
```

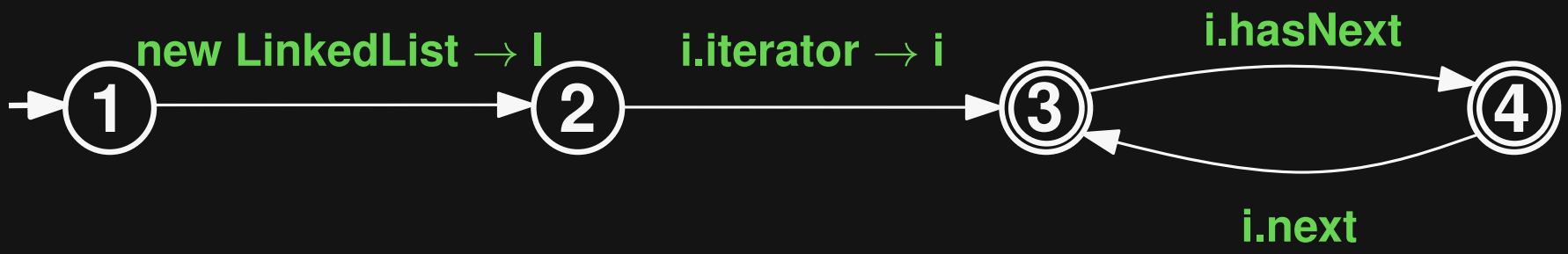
$$l \in r_{state2}, l \in r_{iterator}$$

```
Iterator i = l.iterator();
```

$$l \in r_{state3}, i \in r_{state3}, i \in r_{hasNext}, (l, i) \in r_{protocol}$$

```
i.next();
```

$$i \notin r_{next}$$



Results

Program	Warnings			
	Total	Bugs	Code smells	True pos.
avrora	13	9	0	69%
batik	1	0	0	0%
daytrader	0	0	0	—
eclipse	15	2	1	20%
fop	13	8	1	69%
h2	1	0	0	0%
jython	7	2	1	43%
lucene	13	3	3	46%
pmd	15	2	8	67%
sunflow	0	0	0	—
tomcat	2	0	0	0%
xalan	1	0	1	100%
Total	81	26	15	51%

exception or unexpected behavior

maintainability or performance issue

Results

Program	Warnings			
	Total	Bugs	Code smells	True pos.
avrora	13	9	0	69%
batik	1	0	0	0%
daytrader	0	0	0	—
eclipse	15	2	1	20%
fop	13	8	1	69%
h2	1	0	0	0%
jython	7	2	1	43%
lucene	13	3	3	46%
pmd	15	2	8	67%
sunflow	0	0	0	—
tomcat	2	0	0	0%
xalan	1	0	1	100%
Total	81	26	15	51%

exception or unexpected behavior

maintainability or performance issue

Few false positives!

Examples

```
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while ( i.hasNext() ) {
  PinLink curr = (PinLink) i.next();
  if ( ... ) {
    pinConnections.remove(curr);
  }
}
```

Examples

```
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while ( i.hasNext() ) {
  PinLink curr = (PinLink) i.next();
  if ( ... ) {
    pinConnections.remove(curr);
  }
}
```

Concurrent modification

Examples

```
BufferedReader in = null;
try {
  in = new BufferedReader(...);
  ...
  in.close();
} finally {
  if (in != null) {
    try { in.close(); }
    catch (IOException e) { ... }
  }
}
```

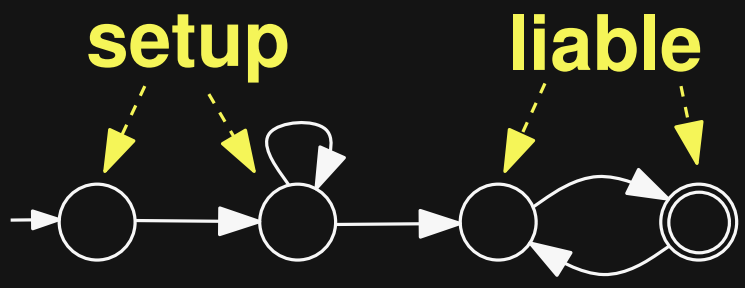
Examples

```
BufferedReader in = null;
try {
  in = new BufferedReader(...);
  ...
  in.close();
} finally {
  if (in != null) {
    try { in.close(); }
    catch (IOException e) { ... }
  }
}
```

Duplicate close

Summary: Bug Finding

Clear definition of protocol violation



Static and dynamic checking:

- Both are practical
- Complement each other

Conclusion

An attractive tool to help programmers

- understand large systems
- pinpoint problem areas

Supplements other approaches:

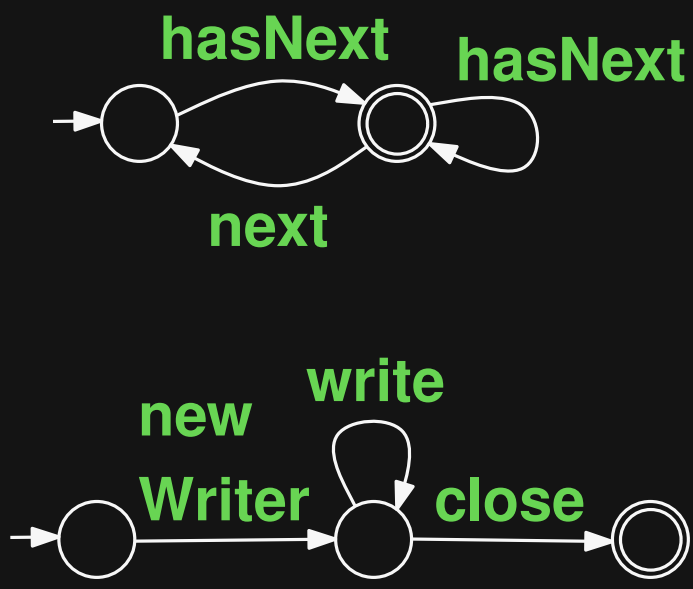
- Reveals multi-object bugs
- Easy to use

Thank you!

michael@binaervarianz.de

What is a Violation?

vs.

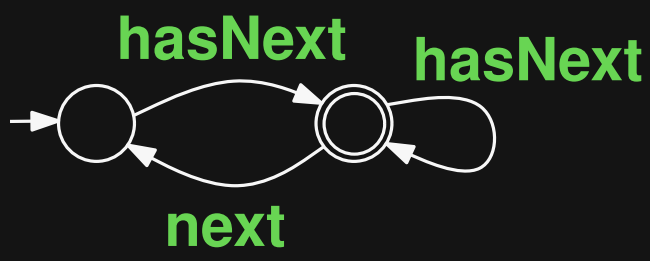


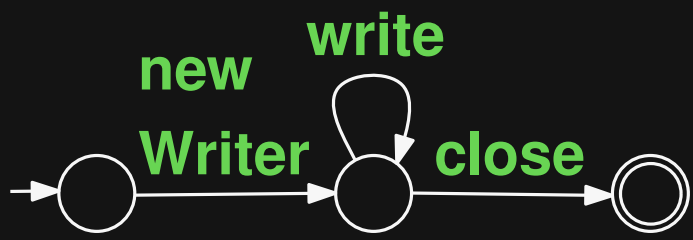
Negative protocols

Positive protocols

What is a Violation?

vs.





Negative protocols

- Reaching final state

Positive protocols

- Taking non-existing transition
- Not reaching final state

Related Work (Mining)

- Gabel & Su, FSE 2008
  - Language learning algorithm with pre-defined micro-patterns
  - Don’t consider dataflow
- Lee, Chen & Rosu, ICSE 2011
  - First, learn related events; then, mine sliced trace with PFSA learner
  - Require unit tests for first step

Template

Template

Template

Template

Template