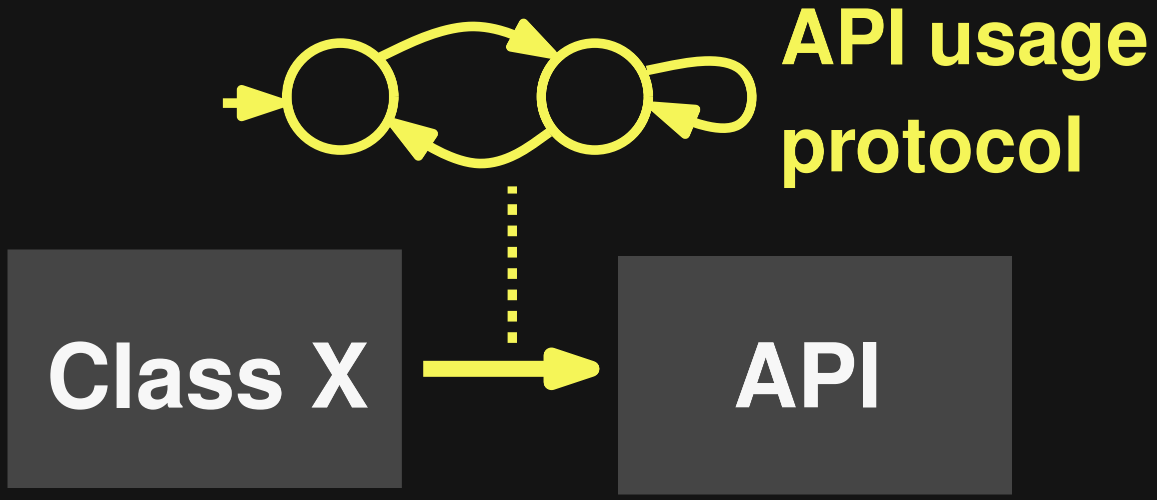
Leveraging Test Generation and Specification Mining for Automated Bug Detection without False Positives

Michael Pradel and Thomas R. Gross

Department of Computer Science
ETH Zurich

Motivation

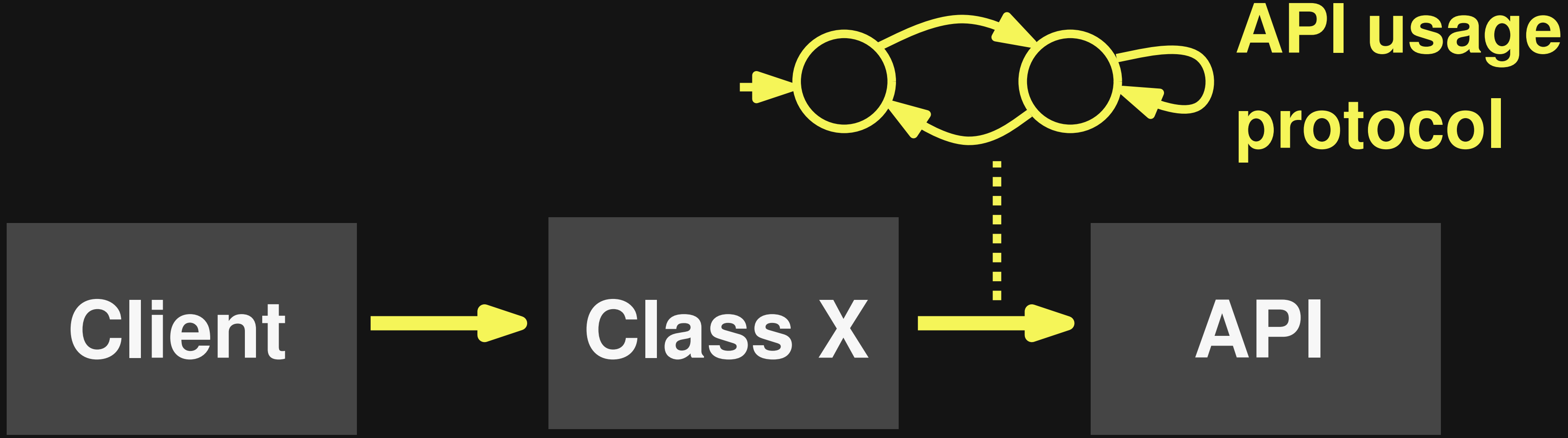


API usage protocol

Class X

API

Motivation



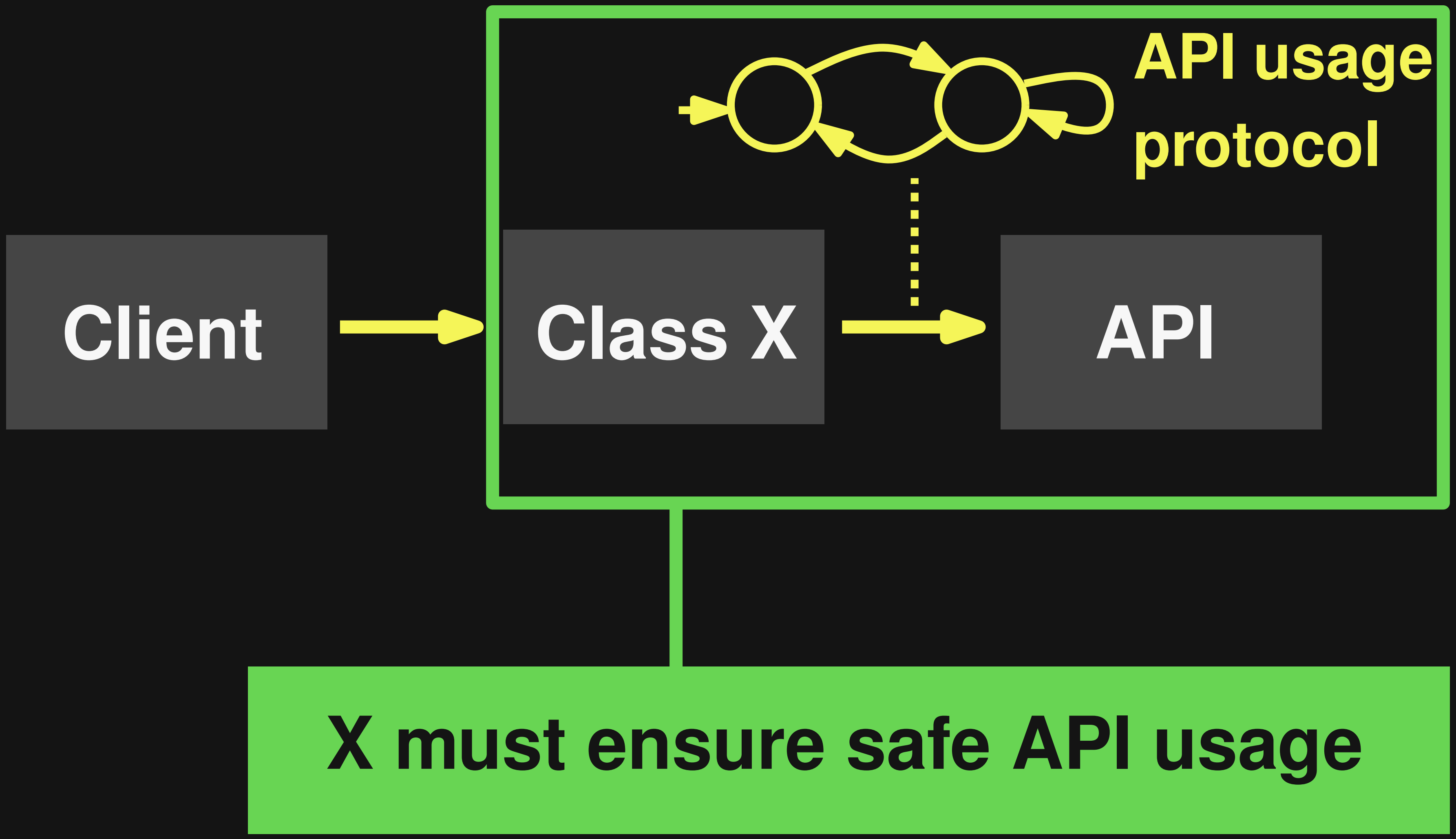
Client

Class X

API

API usage protocol

Motivation



Client

Class X

API

API usage protocol

X must ensure safe API usage

Example from Apache Xalan

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```

Example from Apache Xalan

Stack has a protocol ...

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```

Example from Apache Xalan

Stack has a protocol ...

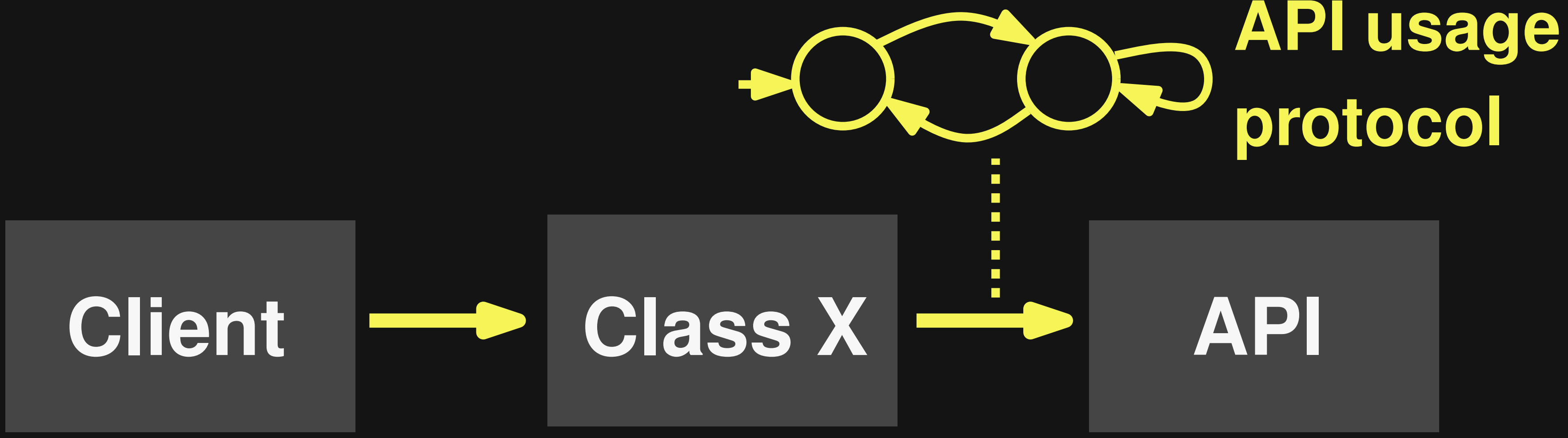
... but X fails to ensure it:

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```

```
X x = new X();
x.report();
```

EmptyStackException

Unsafe API Usage



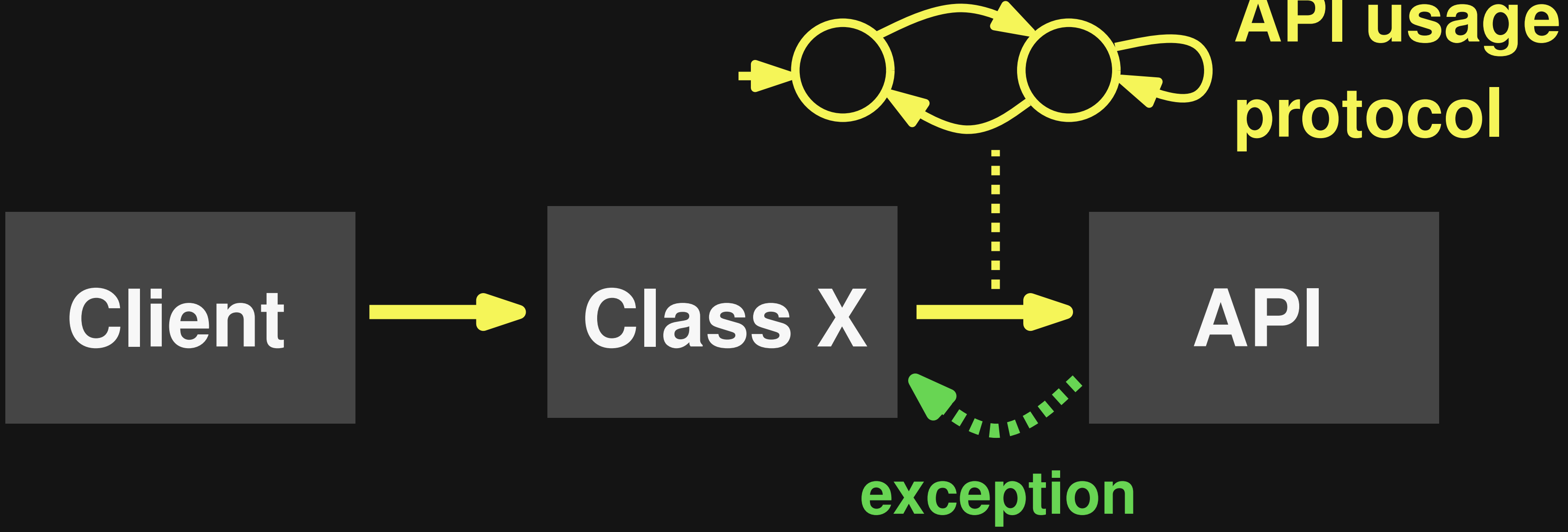
Client

Class X

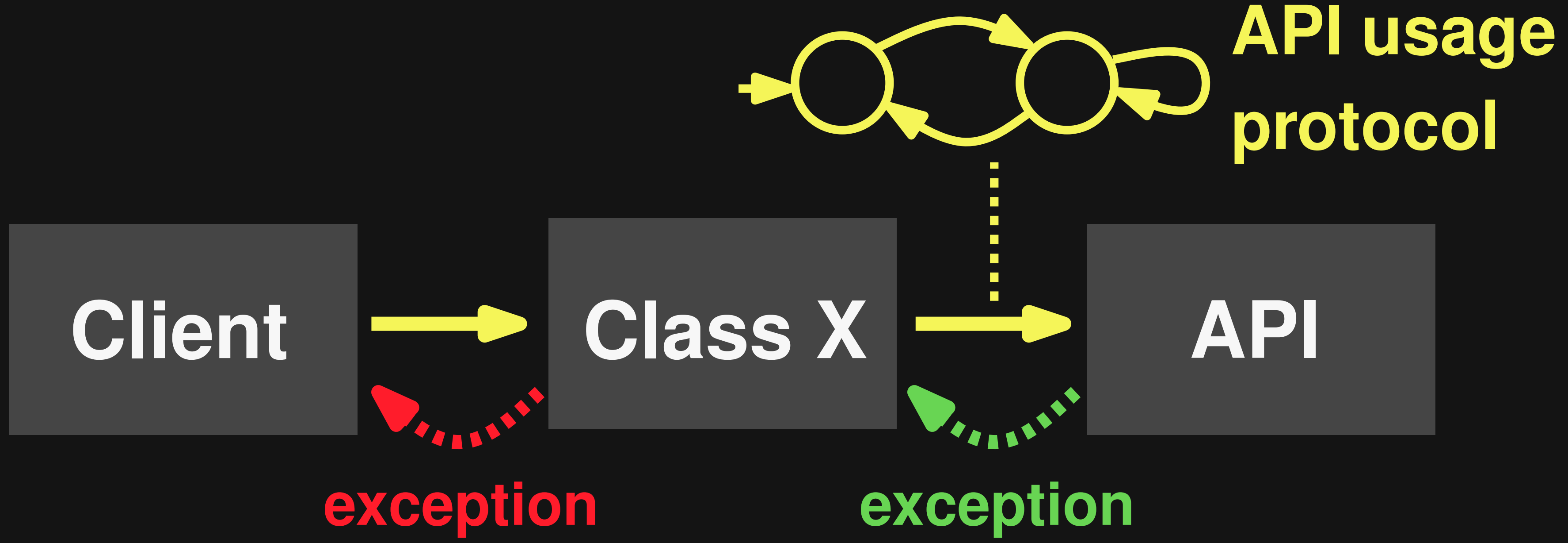
API

API usage protocol

Unsafe API Usage



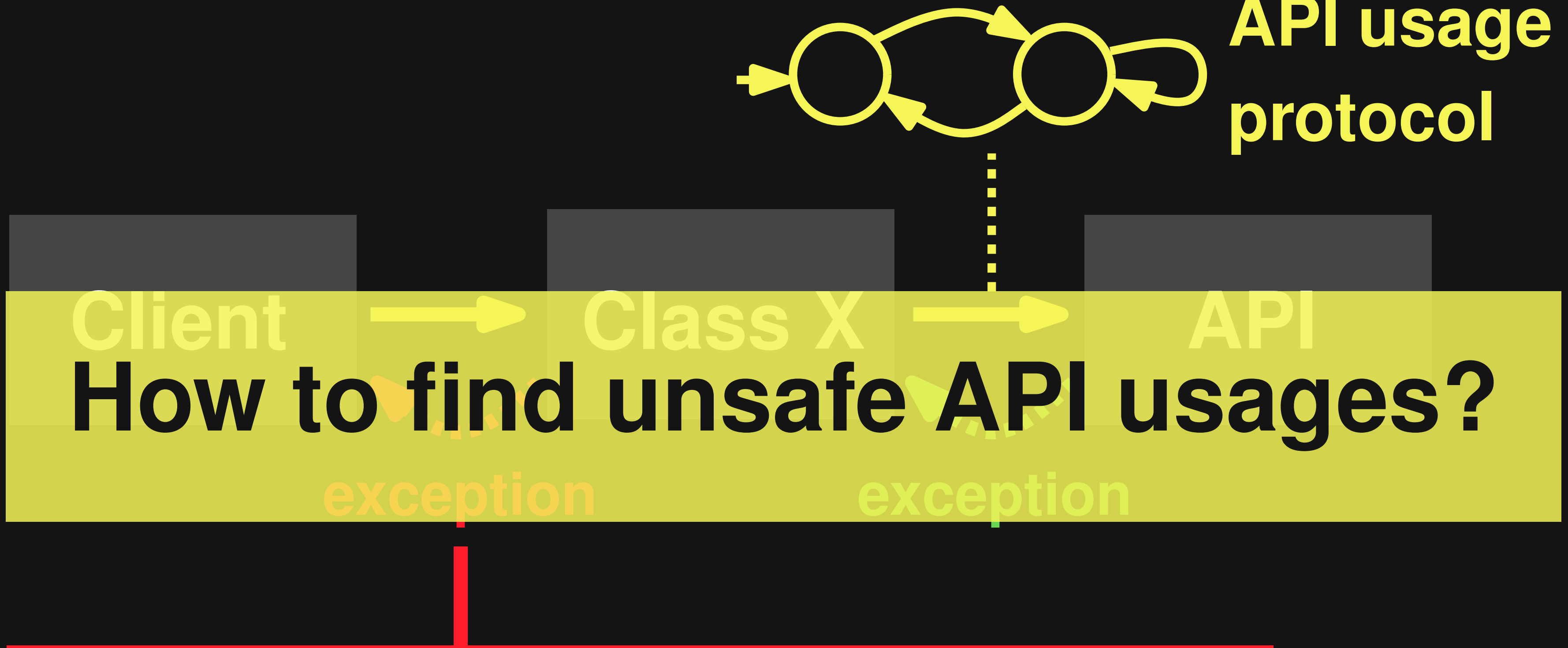
Unsafe API Usage



Unsafe API usage:

X exposes client to unexpected protocol exception

Unsafe API Usage

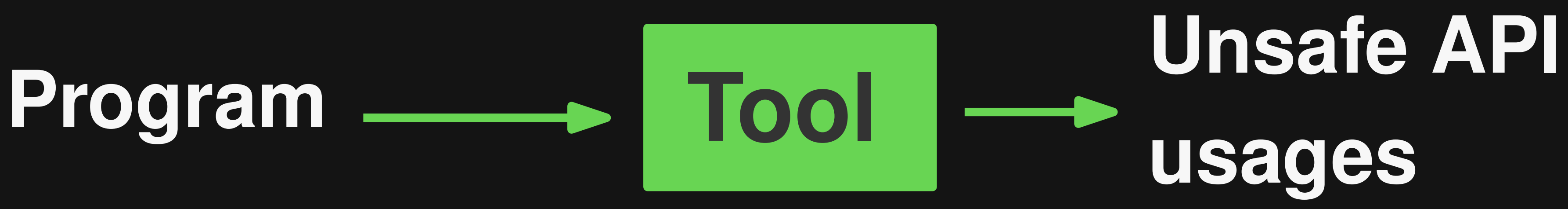


How to find unsafe API usages?

Unsafe API usage:

X exposes client to unexpected protocol exception

Goal

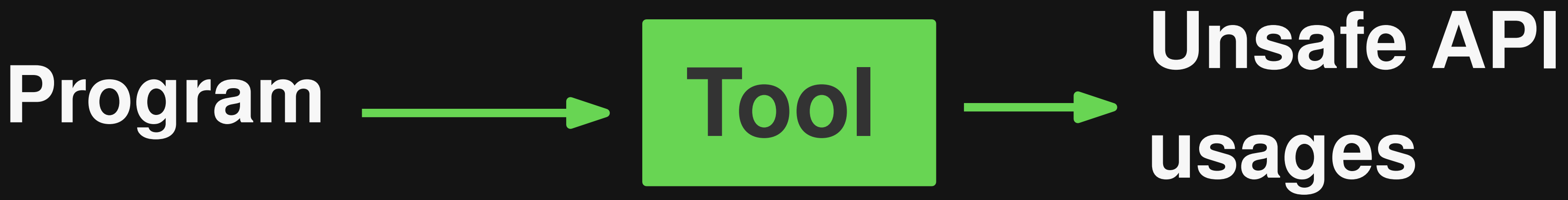


Program

Tool

Unsafe API usages

Goal



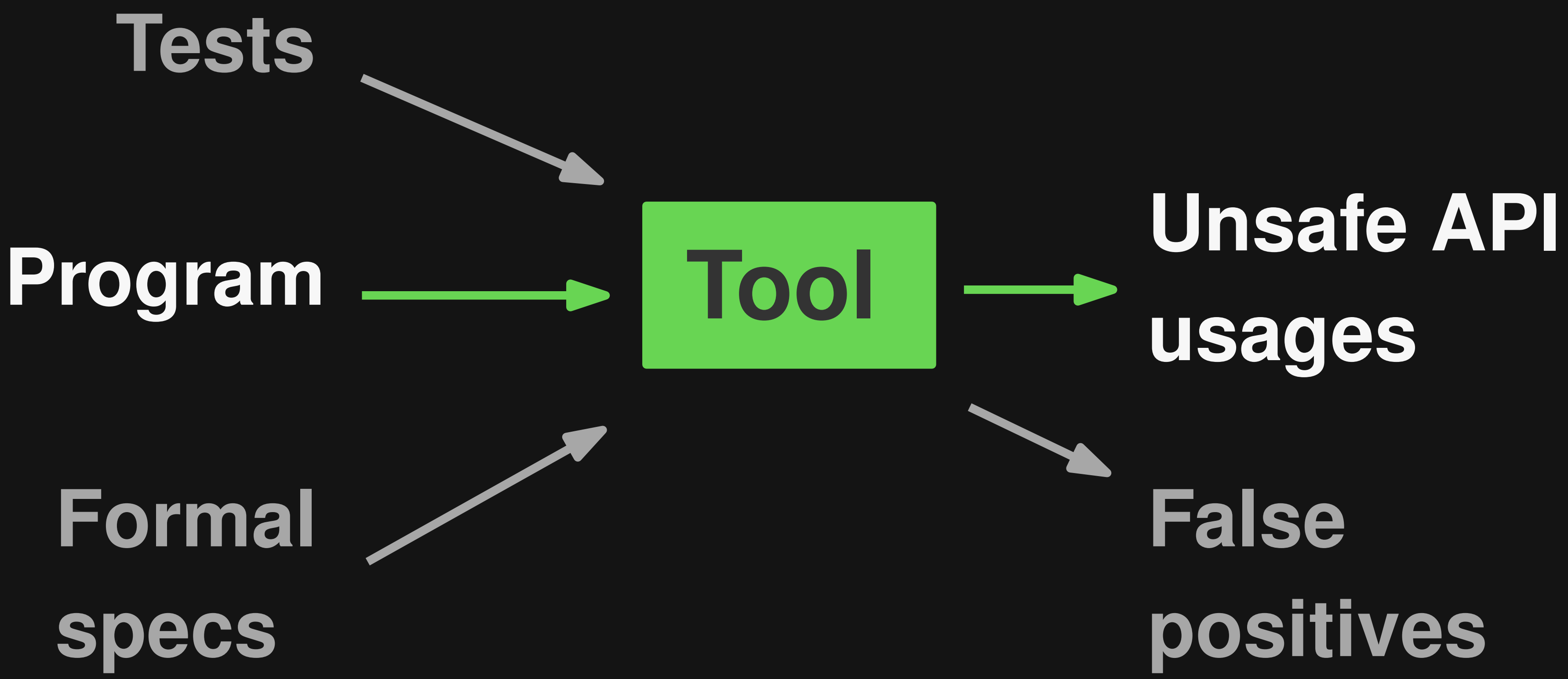
Program

Tool

Unsafe API usages

Automatic and precise bug detection

Goal



Tests

Program

Formal specs

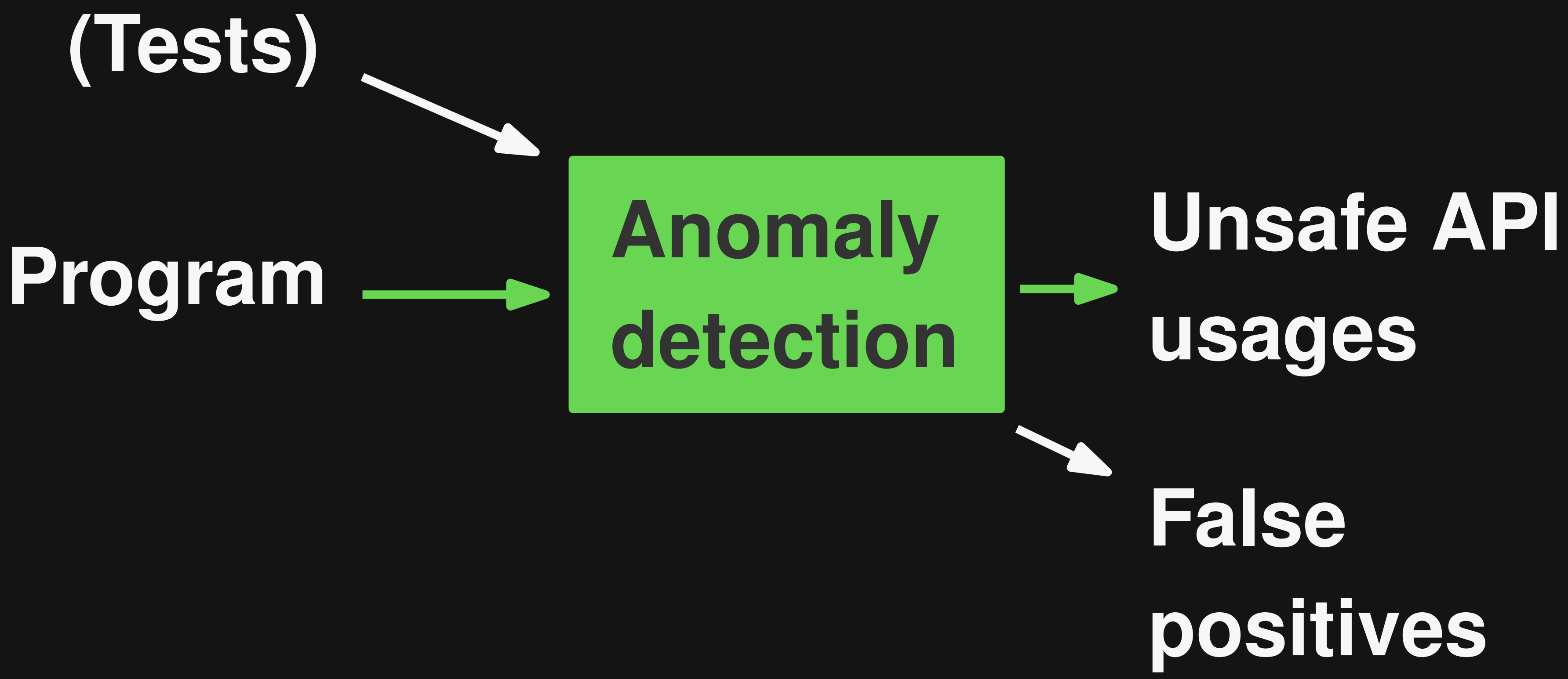
Tool

Unsafe API usages

False positives

~~Automatic~~ and ~~precise~~ bug detection

State of the Art (1)



(Tests)

Program

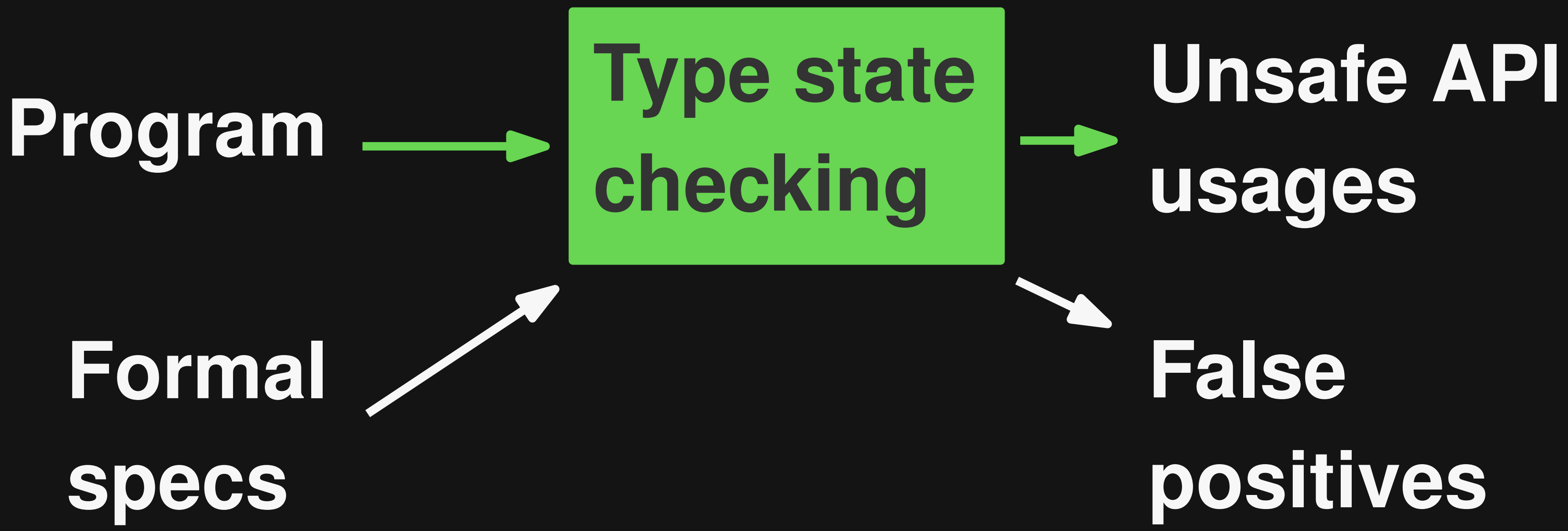
Anomaly detection

Unsafe API usages

False positives

Nguyen et al. ’09
Wasylkowski + Zeller ’09
Thummalapenta + Xie ’09
Monperrus et al. ’10
Gabel + Su ’10

State of the Art (2)



Program

Formal specs

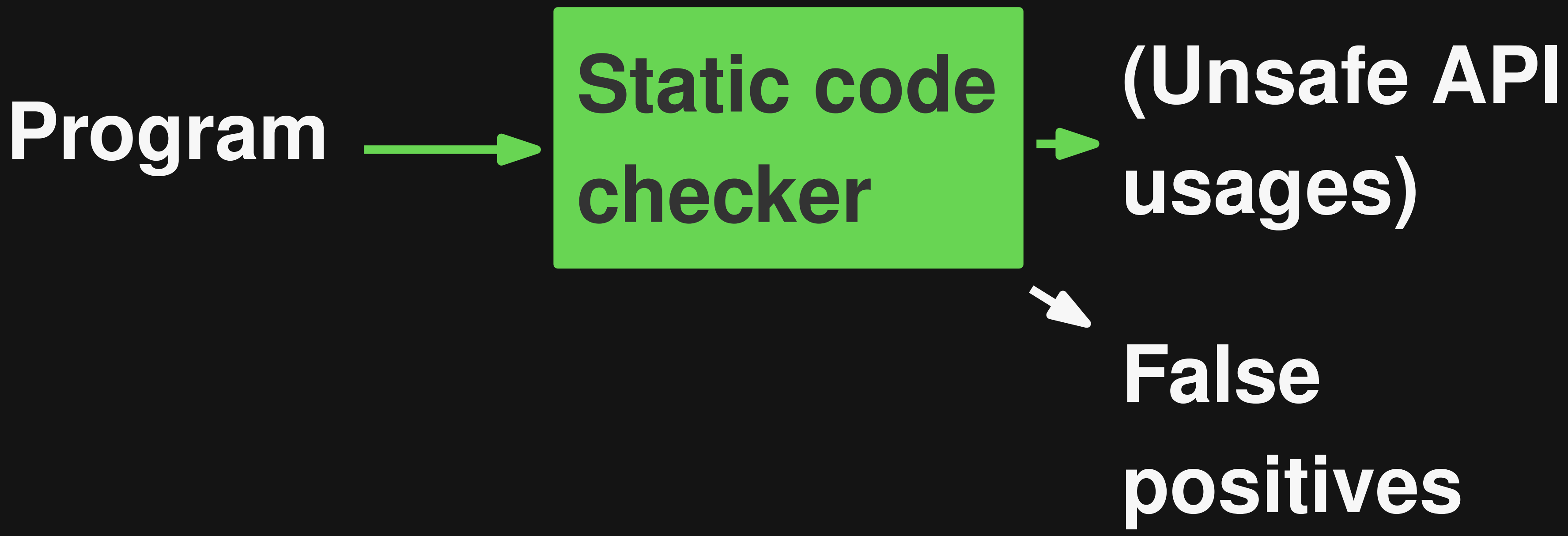
Type state checking

Unsafe API usages

False positives

DeLine + Fähndrich ’04
Bierhoff + Aldrich ’07
Fink et al. ’08
Naeem + Lhotak ’08
Bodden ’10

State of the Art (3)



Program

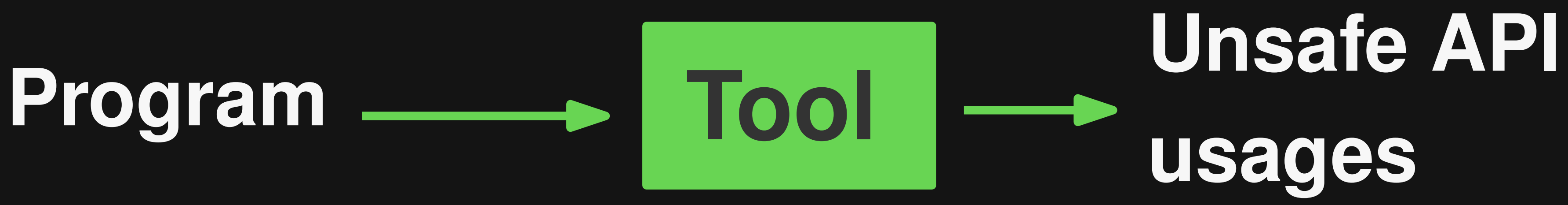
Static code checker

(Unsafe API usages)

False positives

FindBugs
PMD

Goal



Program

Tool

Unsafe API usages

Automatic and precise bug detection

Approach

Dynamic protocol mining

Test generation

Runtime protocol verification

Approach

- ✓ provides protocols
- ✗ requires input to run program

Dynamic protocol mining

Test generation

Runtime protocol verification

Approach

Test generation

Dynamic protocol mining

- ✓ provides protocols
- ✗ requires input to run program

Runtime protocol verification

- ✓ finds protocol violations
- ✗ requires protocols and input

Approach

- ✓ provides protocols
- ✗ requires input to run program

Dynamic protocol mining

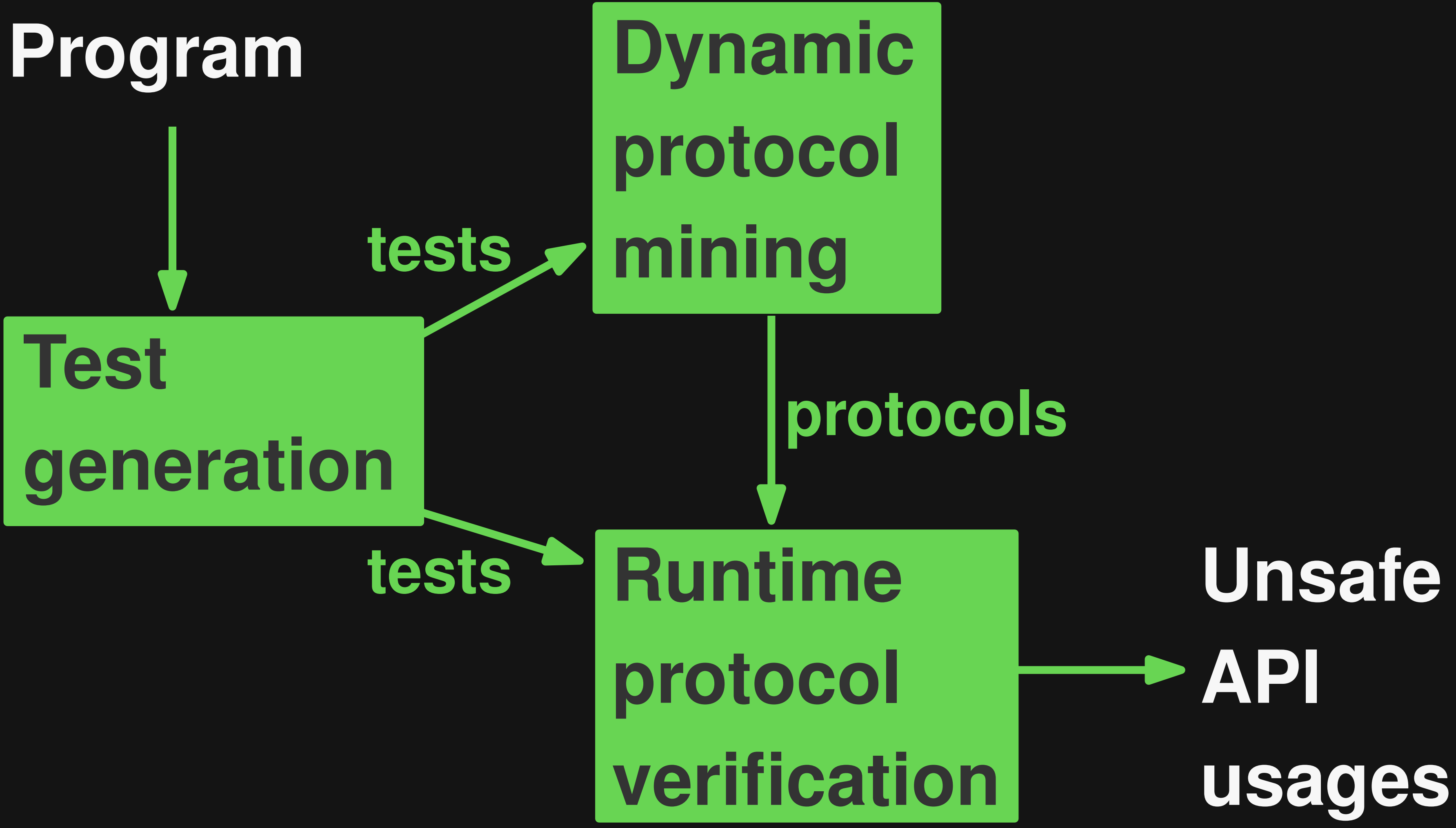
Test generation

- ✓ provides input
- ✗ requires test oracle to find bugs

Runtime protocol verification

- ✓ finds protocol violations
- ✗ requires protocols and input

Approach



Program

Test generation

tests

tests

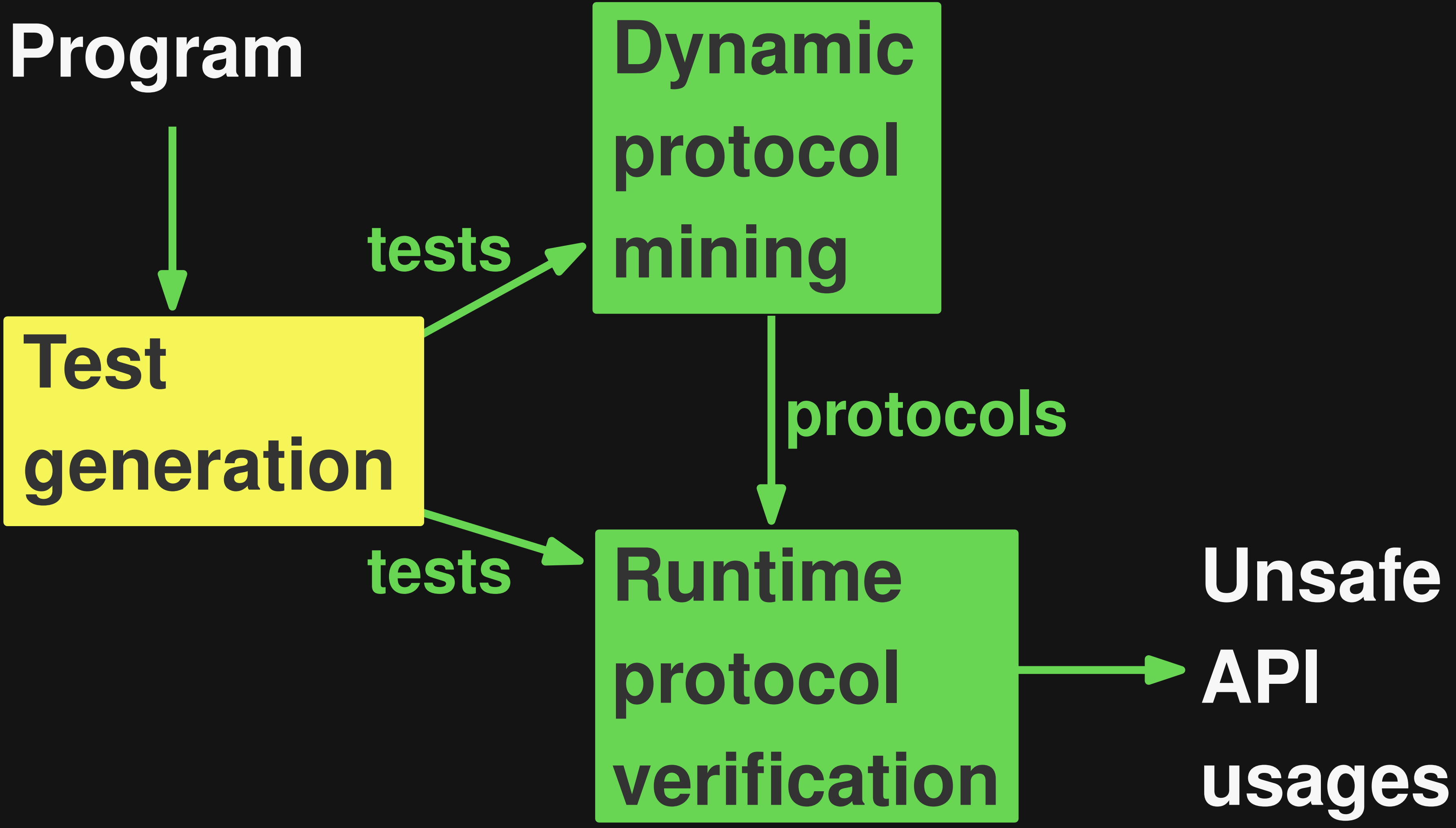
Dynamic protocol mining

protocols

Runtime protocol verification

Unsafe API usages

Approach



Program

Test generation

tests

Dynamic protocol mining

tests

protocols

Runtime protocol verification

Unsafe API usages

Test Generation

Feedback-directed, random test generation [Randoop, Pacheco2007]

Two kinds of tests:

```
class Test {
   ...
}
```

✗

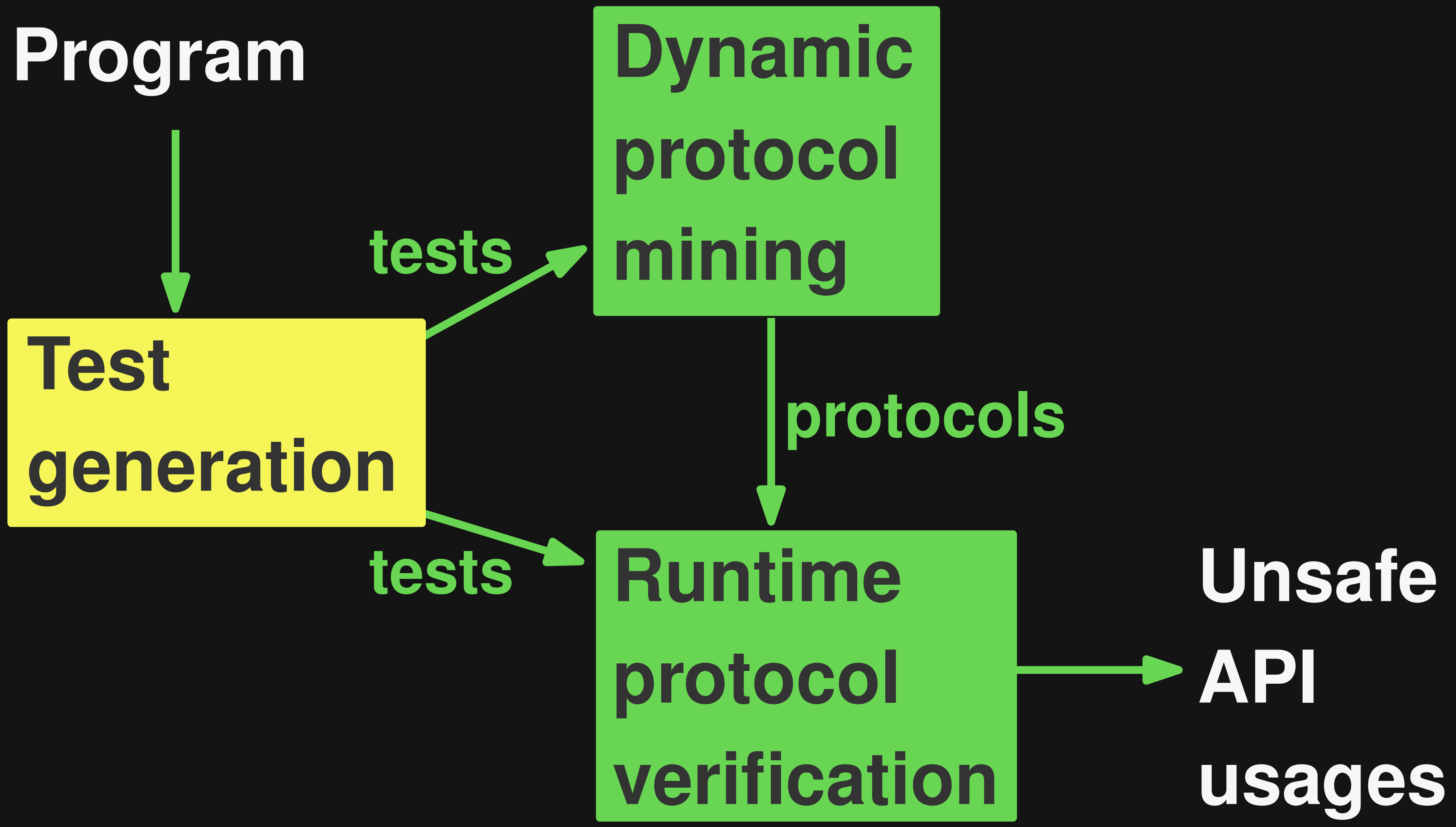
Failing (exception or assertion violation)

```
class Test {
   ...
}
```

✓

Passing

Approach



Program

Test generation

tests

Dynamic protocol mining

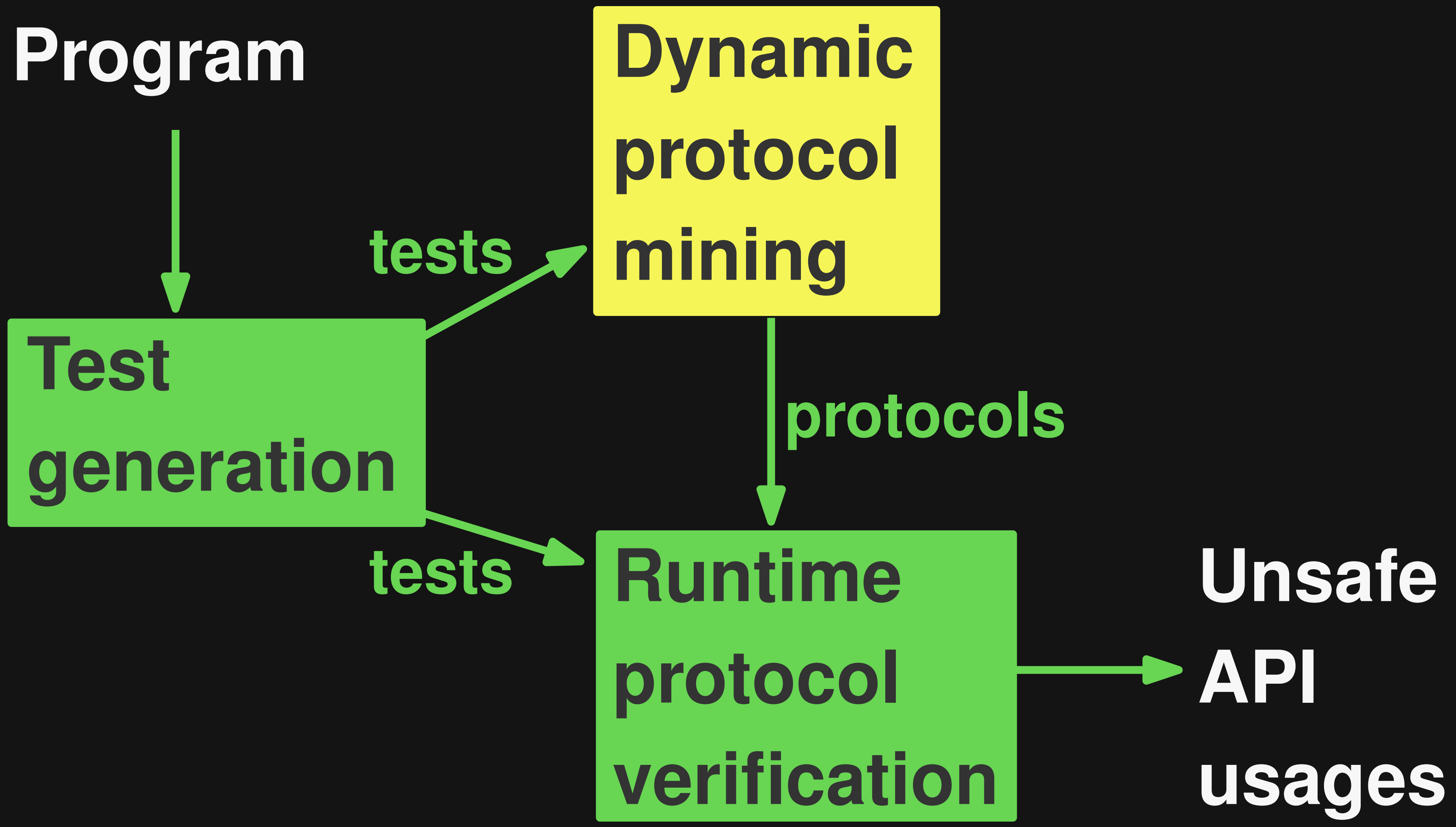
tests

protocols

Runtime protocol verification

Unsafe API usages

Approach


Program

Test generation

tests

Dynamic protocol mining

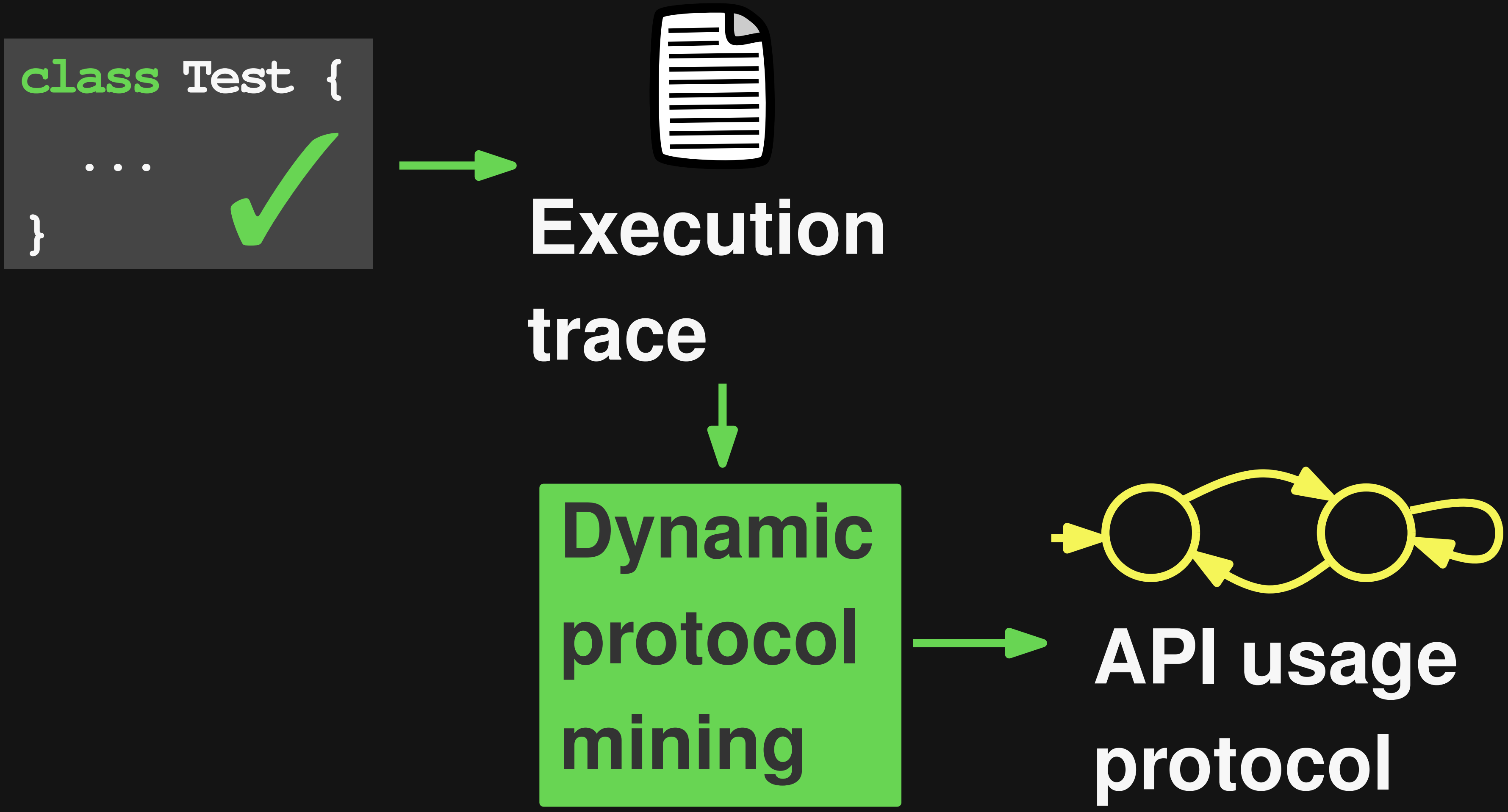
protocols

tests

Runtime protocol verification

Unsafe API usages

Protocol Mining



```
class Test {
   ...
}
```

✓

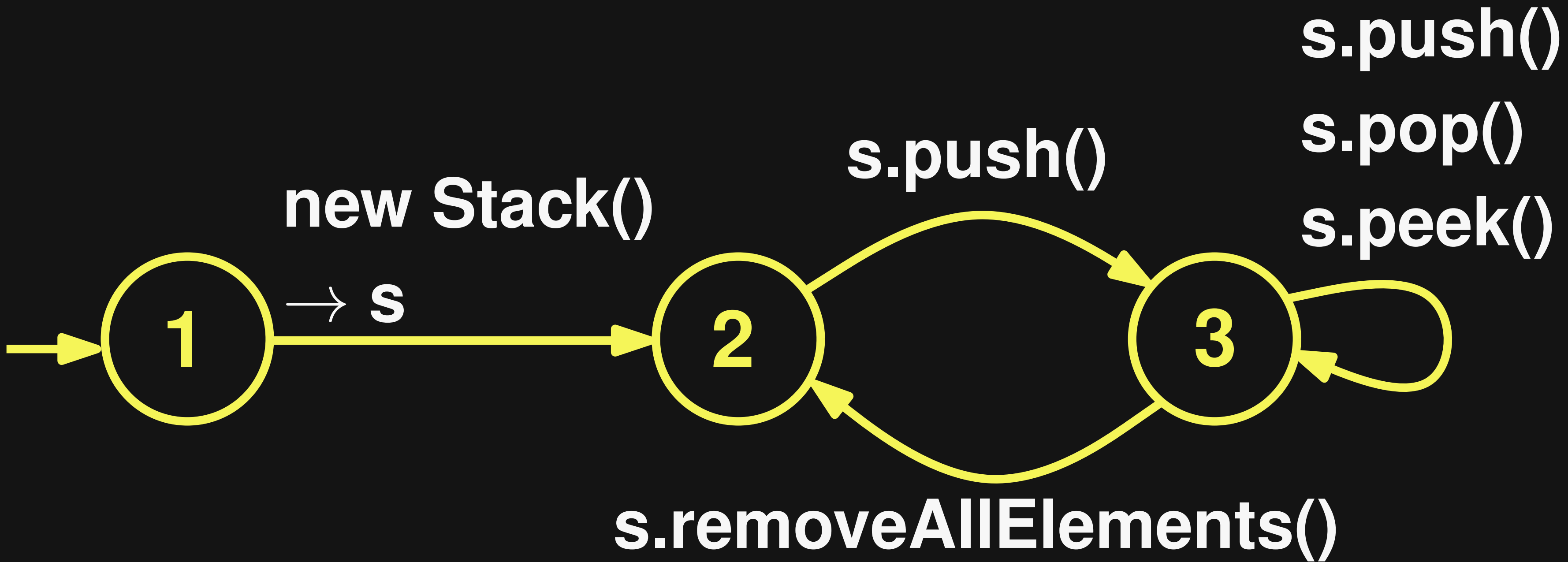
Execution trace

Dynamic protocol mining

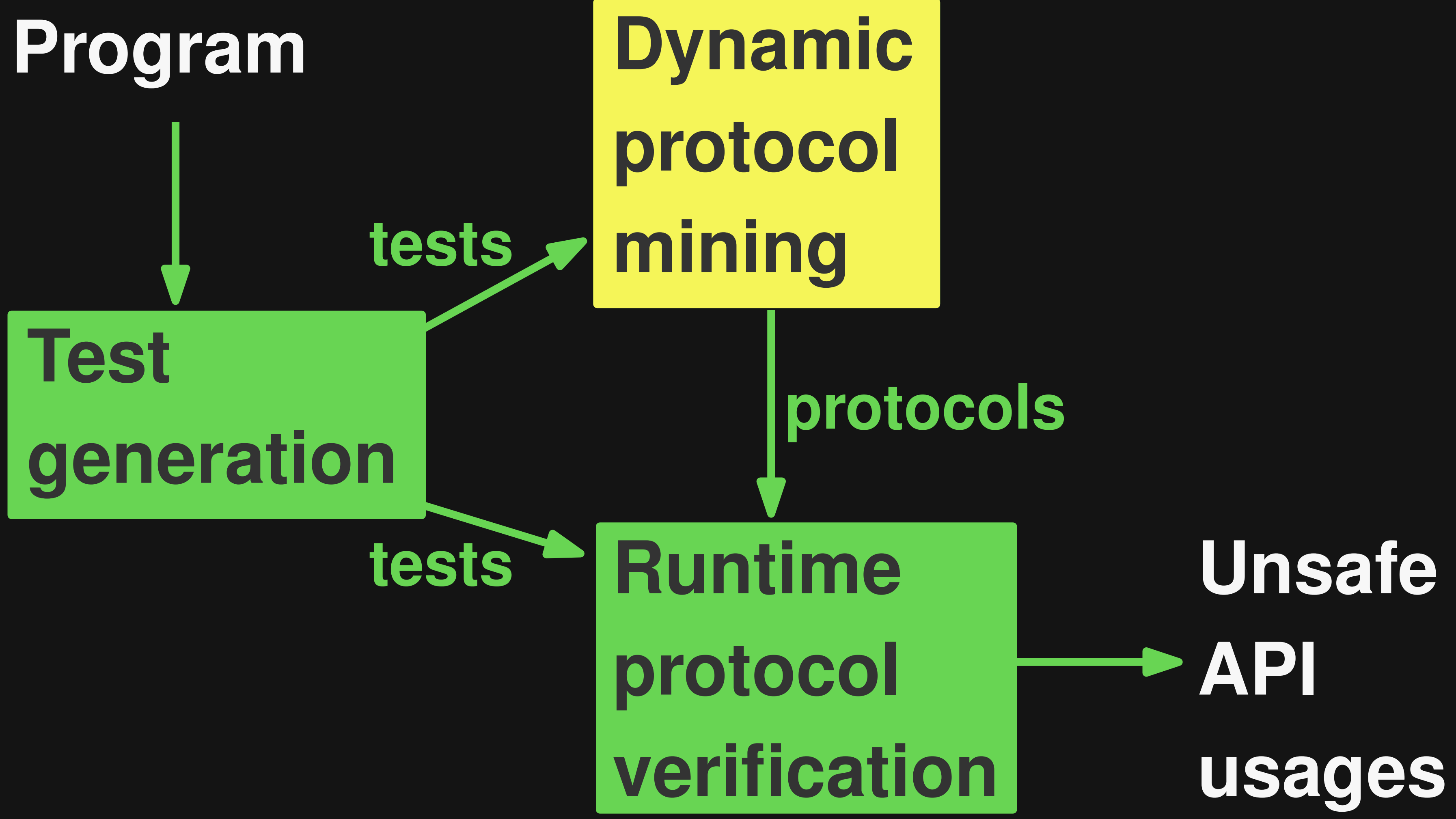
API usage protocol

[ASE'09 and ICSM'10, Pradel et al.]

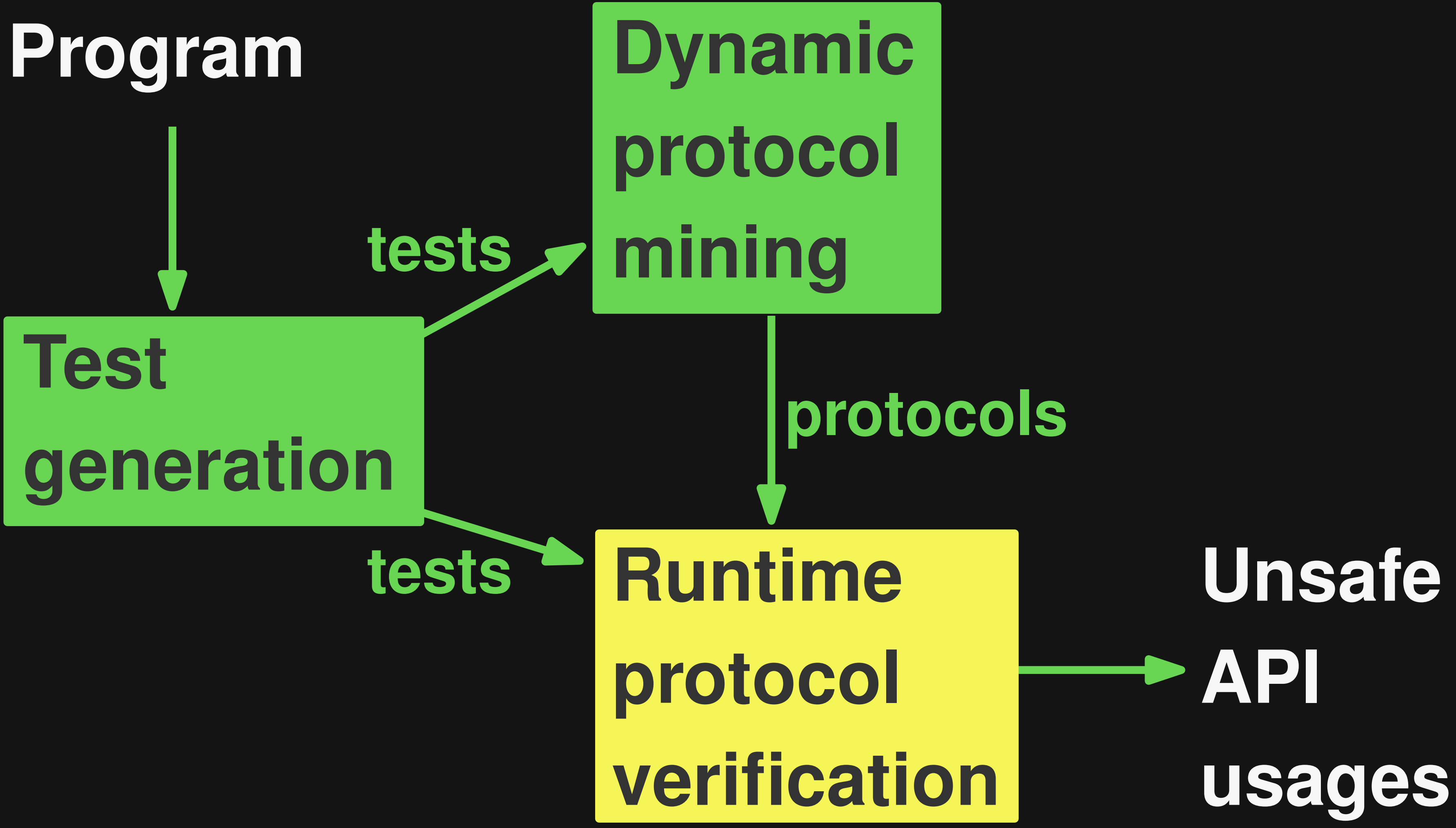
Example



Approach


Program
Dynamic protocol mining
tests
Test generation
protocols
tests
Runtime protocol verification
Unsafe API usages

Approach



Program

Test generation

tests

Dynamic protocol mining

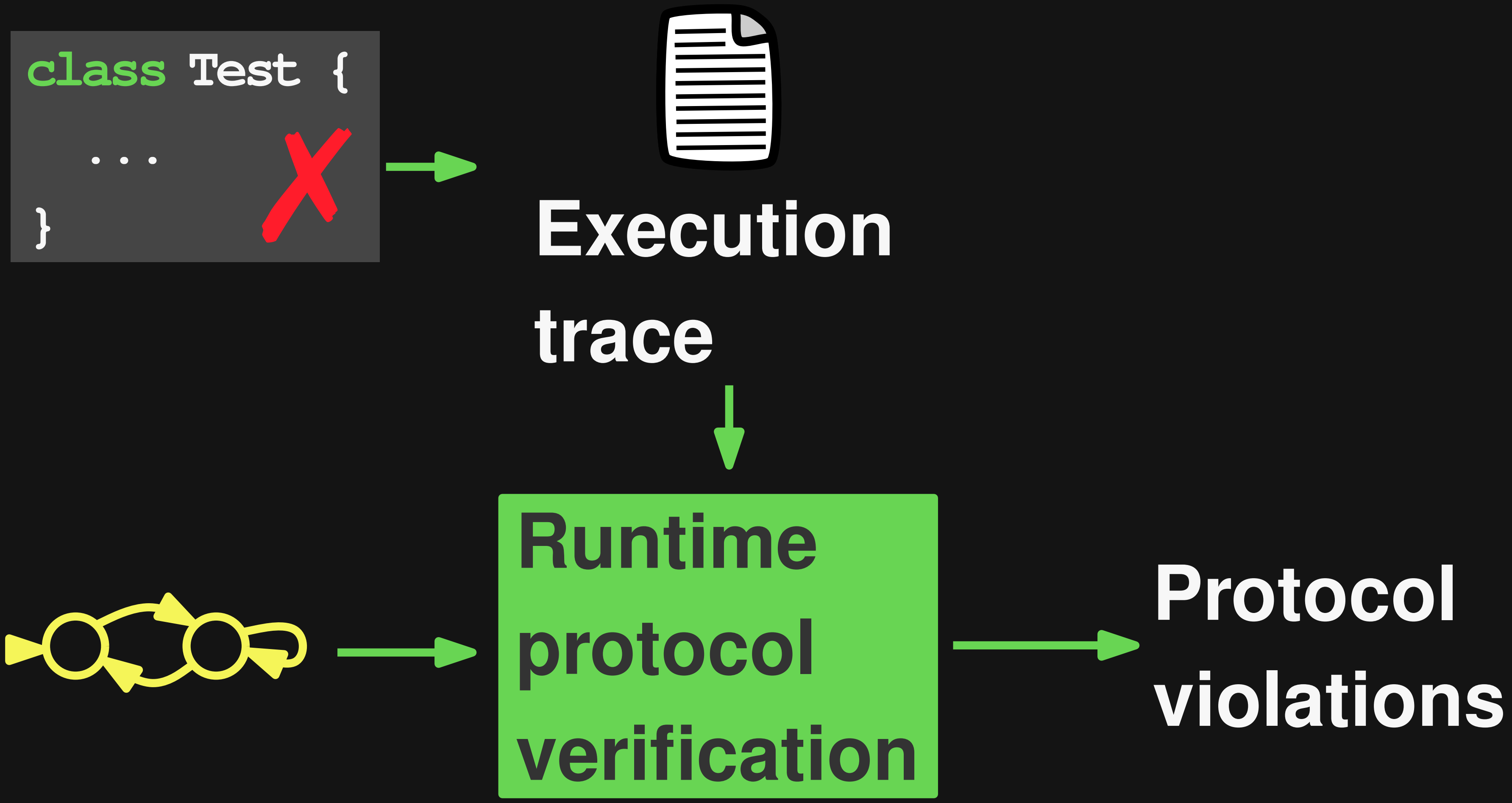
protocols

tests

Runtime protocol verification

Unsafe API usages

Runtime Protocol Verification



```
class Test {
    ...
}
```

Execution trace

Runtime protocol verification

Protocol violations

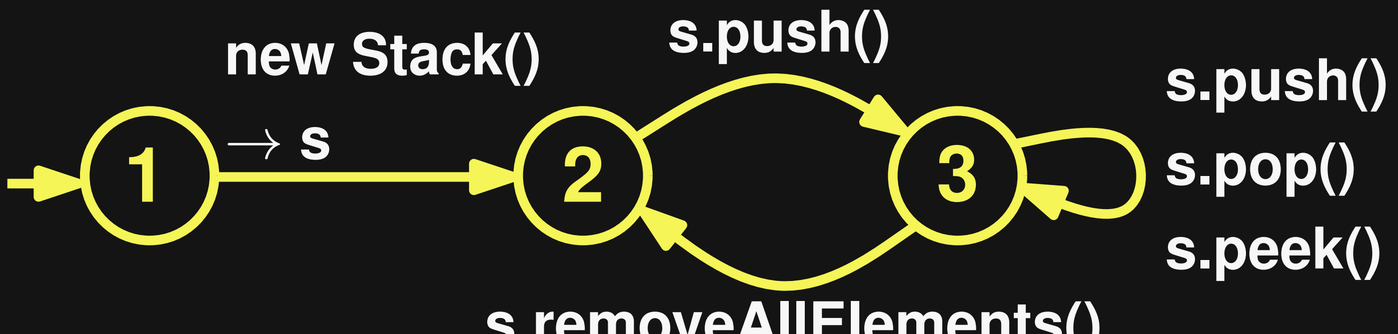
- Check all instances of protocol
- Warn if non-existing transition is taken

Example

Test:

```
X x = new X();
x.report();
```

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```

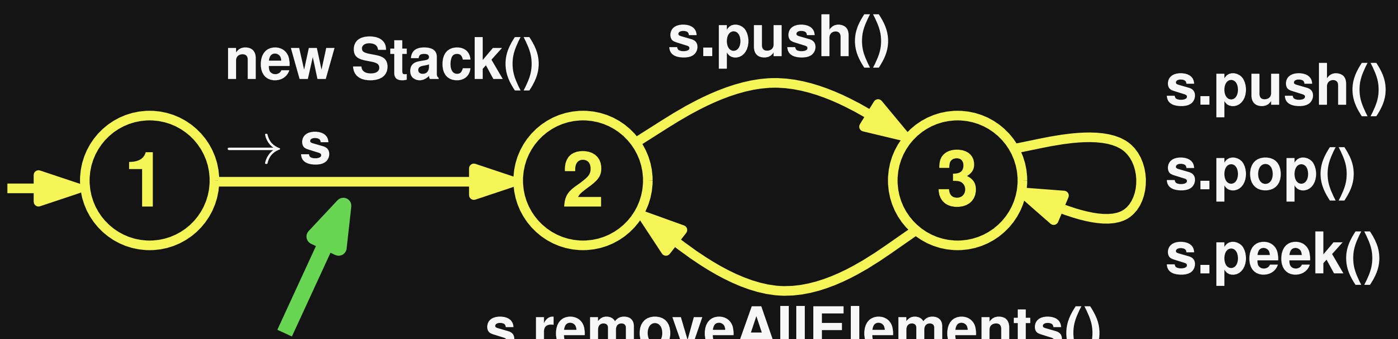


Example

Test:

```
X x = new X();
x.report();
```

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```

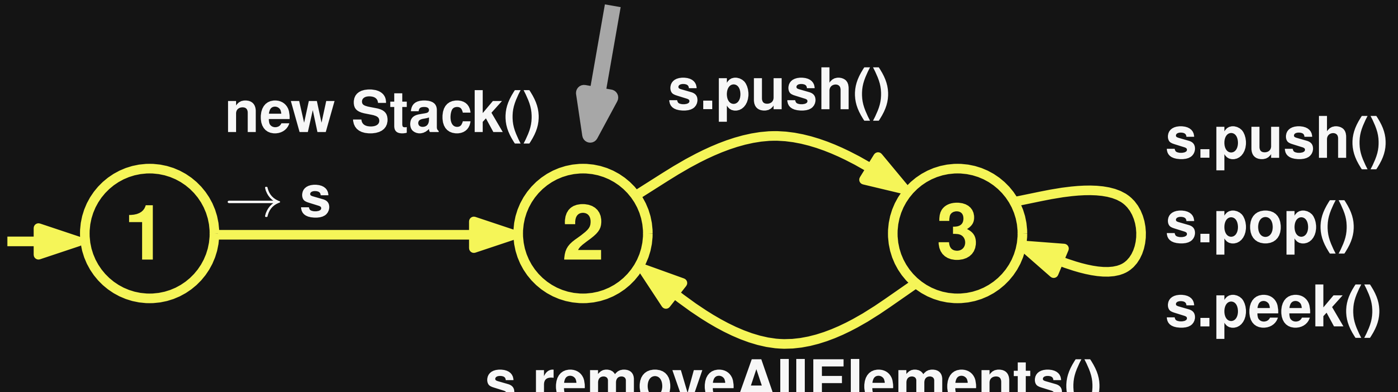


Example

Test:

```
X x = new X();
x.report();
```

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```



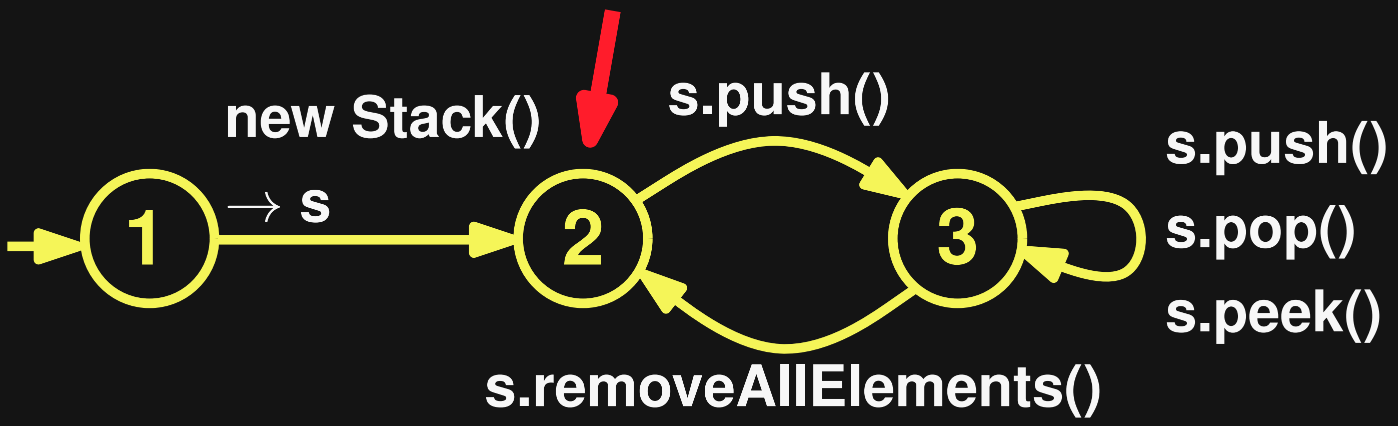
Example

Test:

```
X x = new X();
x.report();
```

```
class X {
  private Stack s = new Stack();
  public String report() {
    return get().toString();
  }
  private Object get() {
    s.peek();
  }
}
```

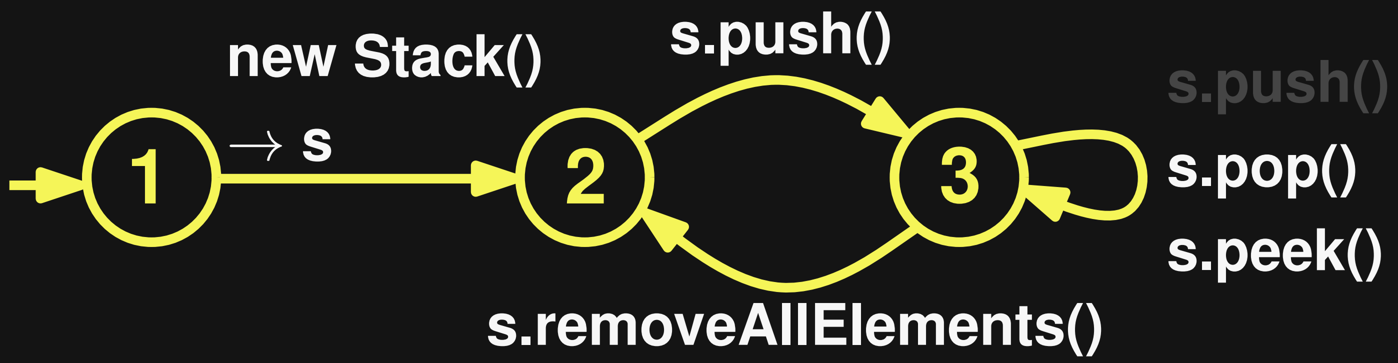
Protocol violation



False Positives

Challenge:

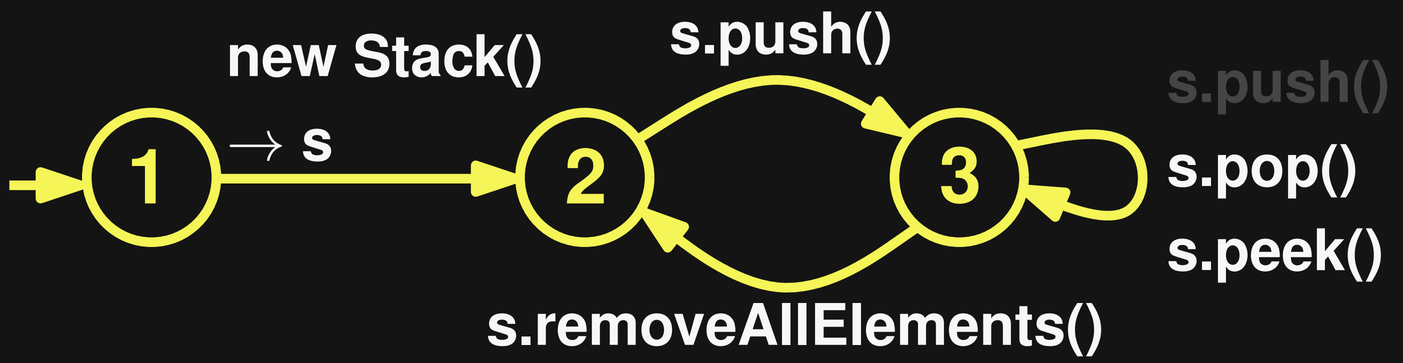
Incomplete protocols (depend on mining)



False Positives

Challenge:

Incomplete protocols (depend on mining)



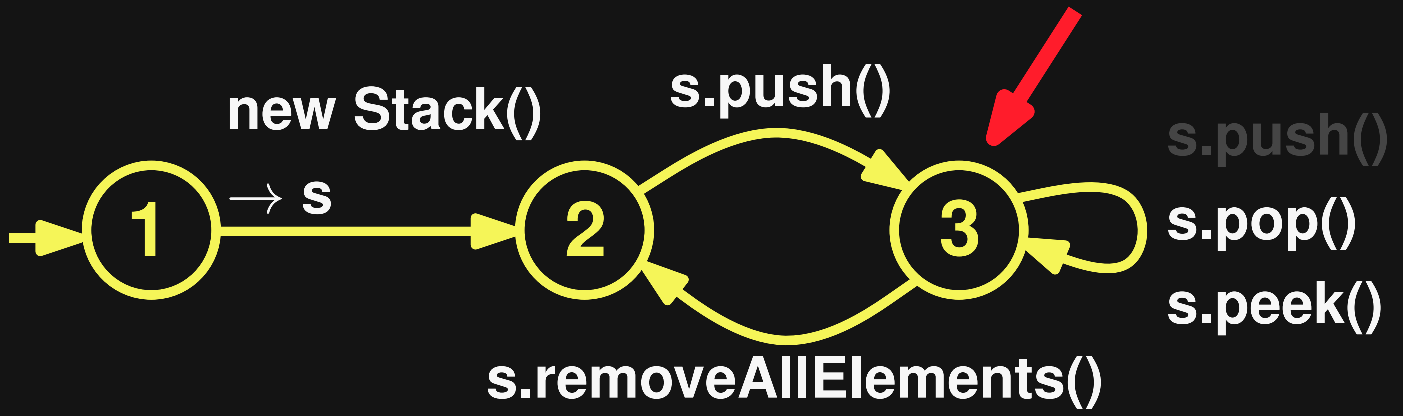
Program:

```
Stack s = new Stack();
s.push(..);
s.push(..);
```

False Positives

Challenge:

Incomplete protocols (depend on mining)



new Stack()
→ s
s.push()
s.push()
s.pop()
s.peek()
s.removeAllElements()

Program:

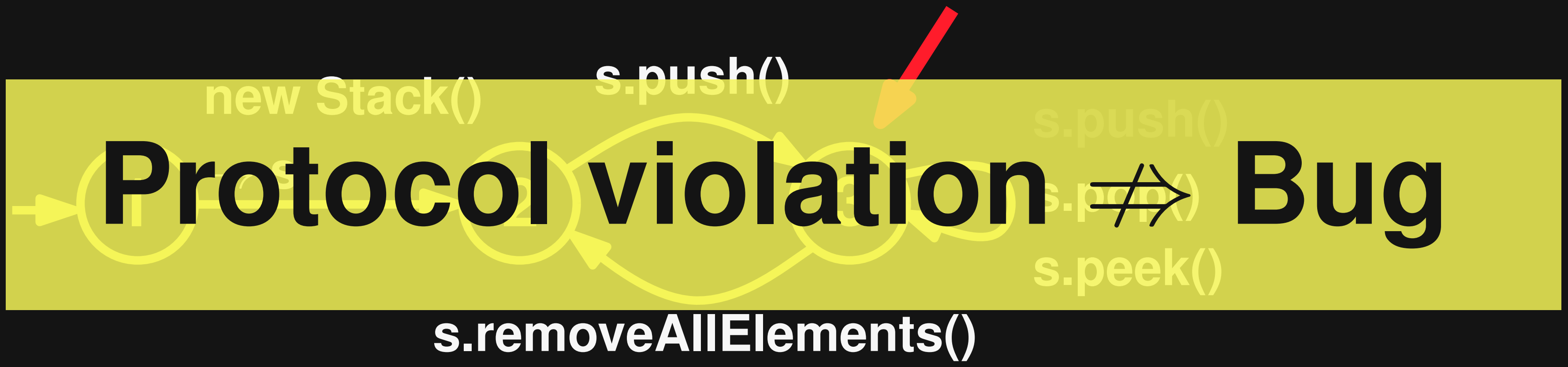
```
Stack s = new Stack();
s.push(..);
s.push(..);
```

False positive protocol violation

False Positives

Challenge:

Incomplete protocols (depend on mining)



Protocol violation ⇏ Bug

Program:

```
Stack s = new Stack();
s.push(..);
s.push(..);
```

False positive protocol violation

Warnings without False Positives

Protocol violation

Program crash

Undeclared exception

Warnings without False Positives

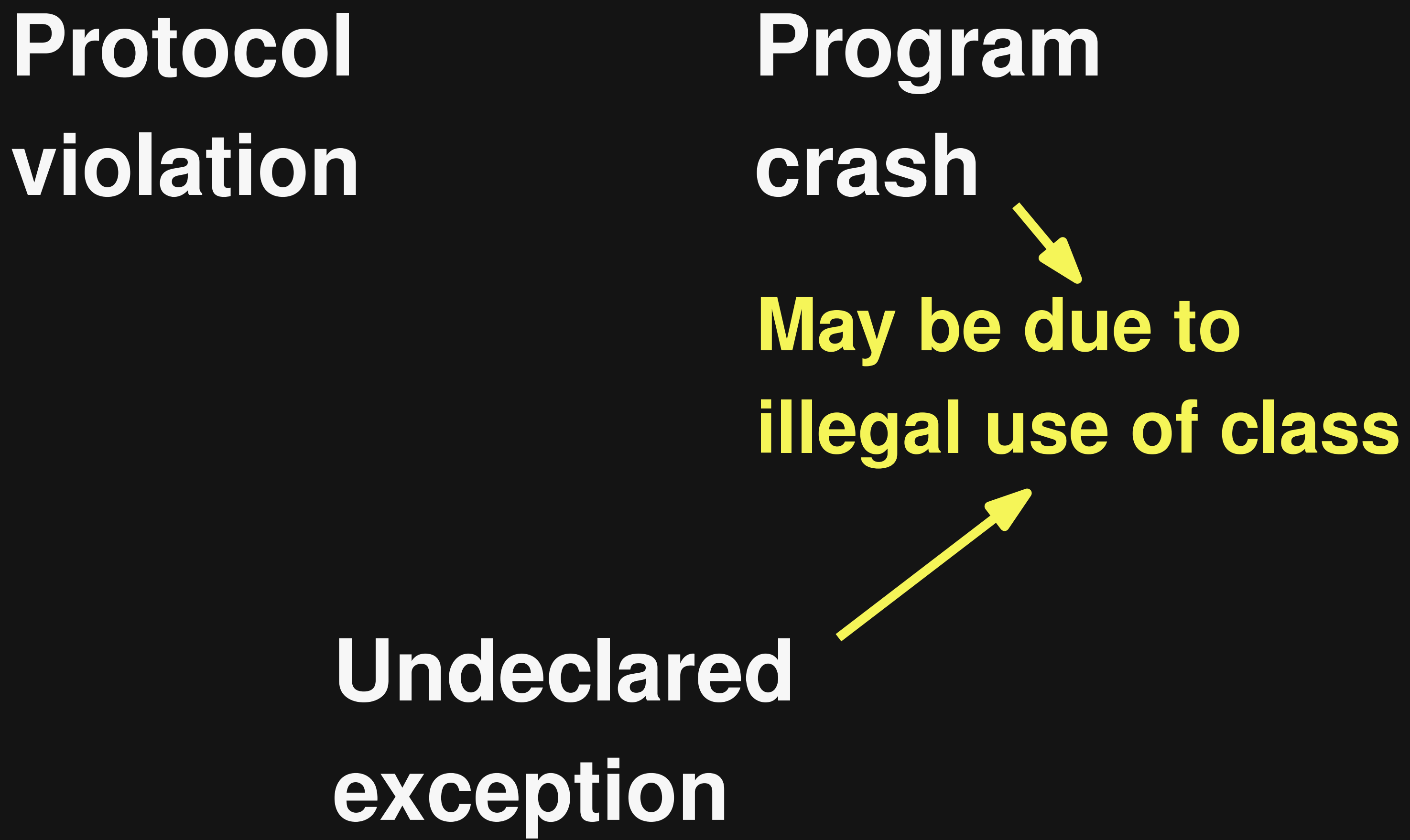
Protocol violation

Program crash

May be due to incomplete protocol

Undeclared exception

Warnings without False Positives



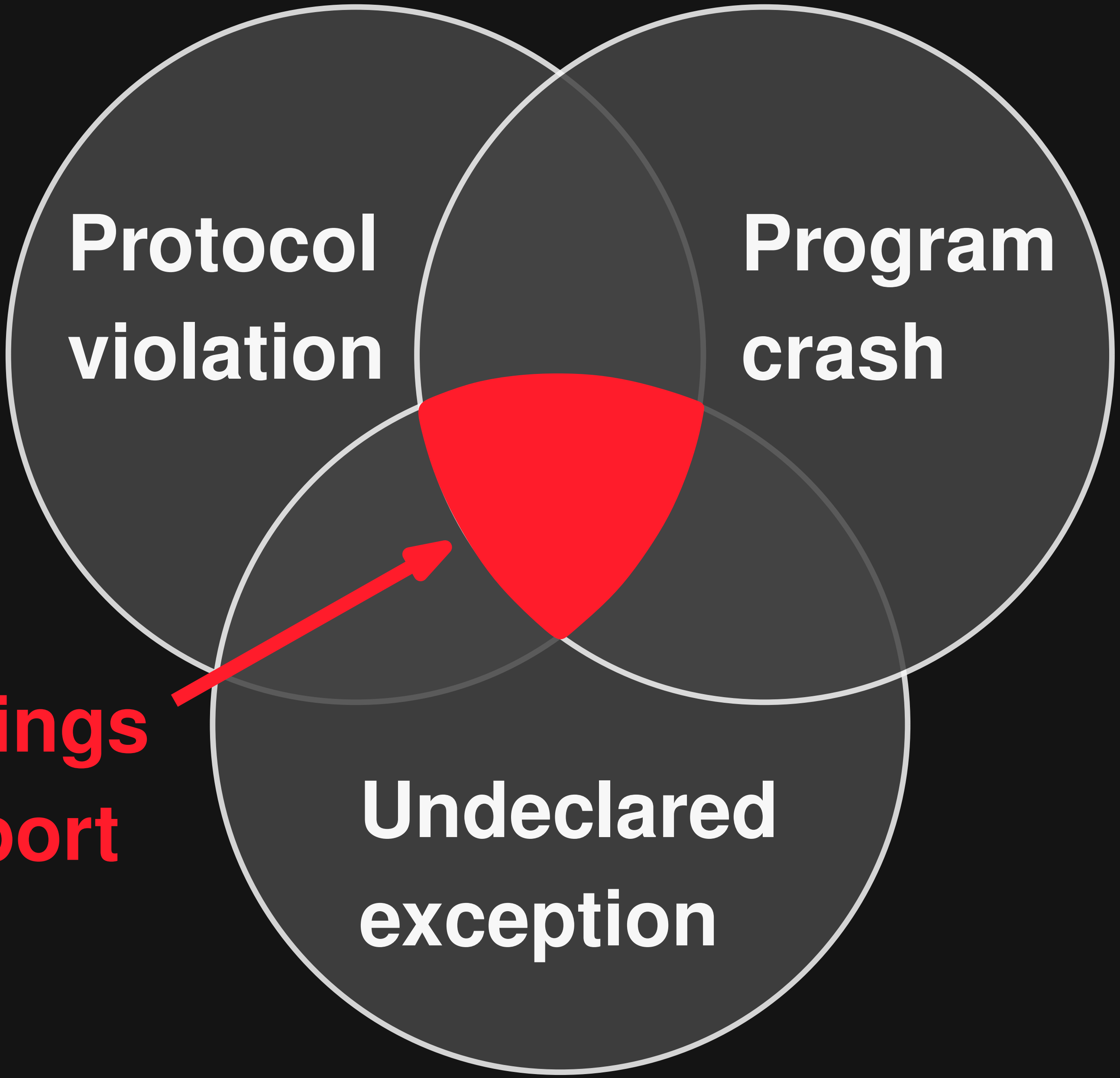
Protocol violation

Program crash

May be due to illegal use of class

Undeclared exception

Warnings without False Positives



Protocol violation

Program crash

Undeclared exception

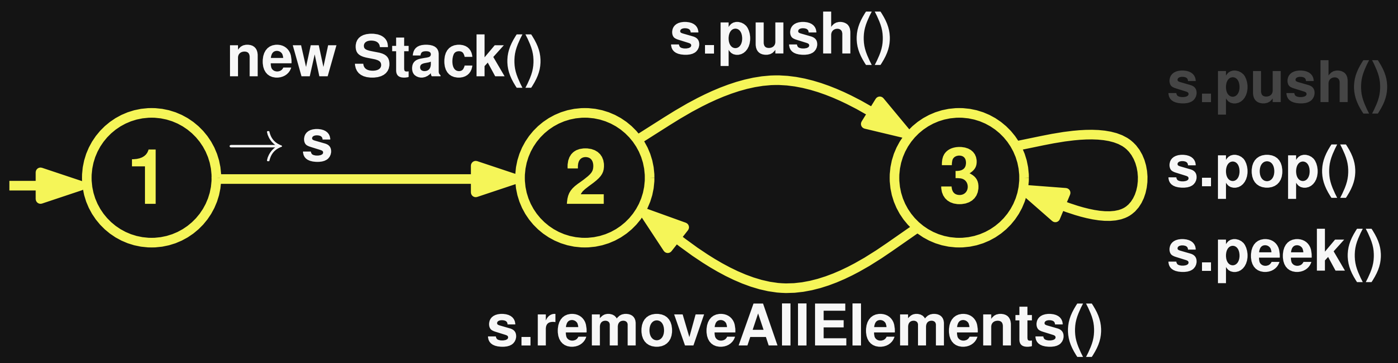
Warnings to report

Warnings without False Positives (2)

Only report problem if:

- protocol violated
  and
- protocol-violating call fails the test
  and
- protocol-violating method does not declare the exception

Example (again)

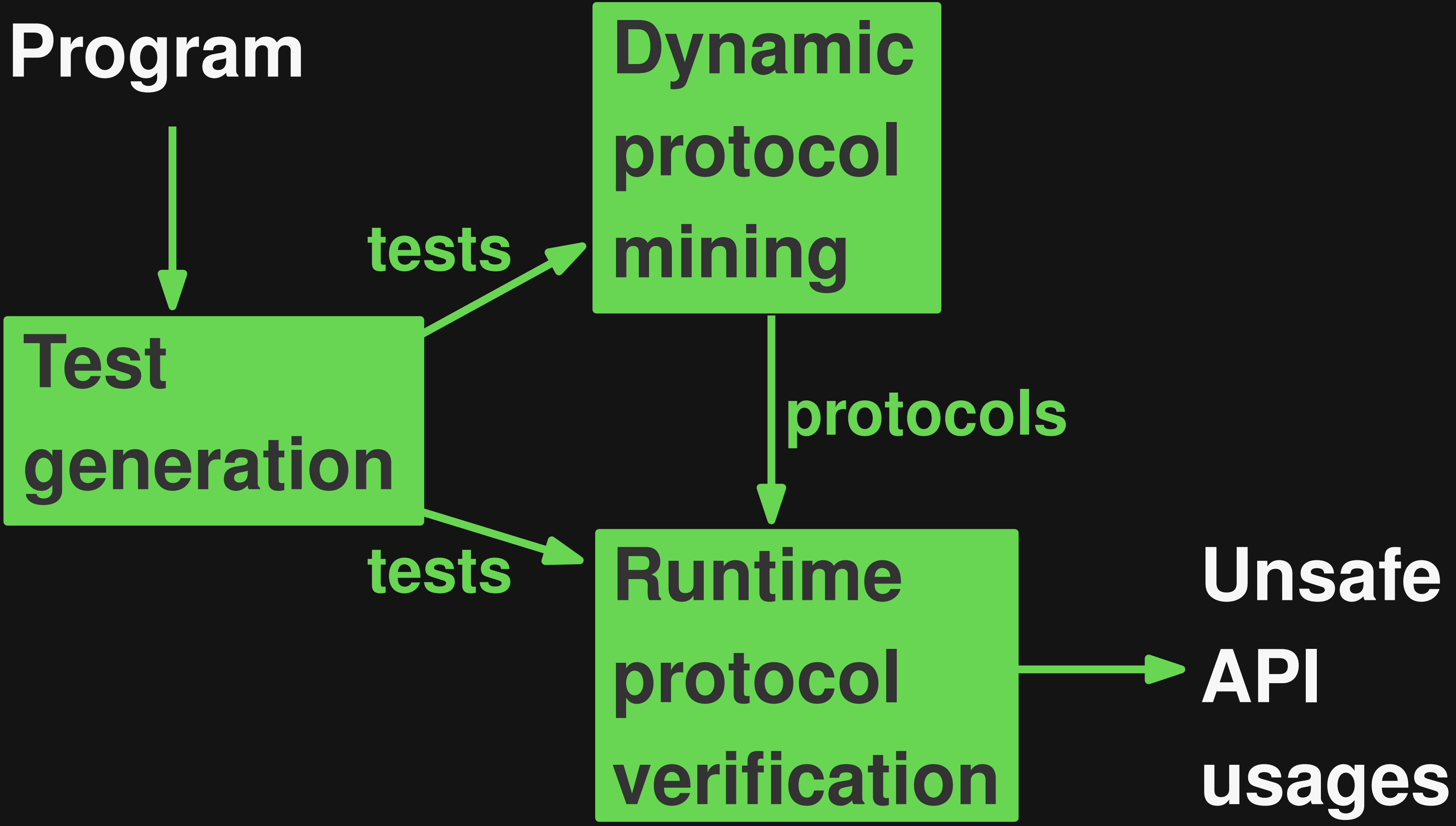


Program:

```
Stack s = new Stack();
s.push(..);
s.push(..);
```

No warning, since protocol violation doesn’t raise exception

Approach



Program

Test generation

tests

tests

Dynamic protocol mining

protocols

Runtime protocol verification

Unsafe API usages

Evaluation

Implemented into fully automatic tool

Main questions:

1. Effectiveness in finding unsafe API usages
2. Comparison with existing work
3. Performance

Setup

Programs:

- DaCapo benchmarks (5,012 classes)

APIs:

- Collection+Iterator
- Vector+Enumeration

} including subclasses

Stopping criterion:

- Generate 10,000 tests per program

Unsafe API Usages

54 unsafe API usages

0 false positives

Example from Jython

```
public class X {
  protected Iterator iter;
  public void _beginCanonical() {
    iter = classes.values().iterator();
  }
  public Object _next() {
    if (iter.hasNext()) return iter.next();
    else return null;
  }
  public void _flushCurrent() {
    iter.remove();
  }
}
```

Example from Jython

```
public class X {
  protected Iterator iter;
  public void _beginCanonical() {
    iter = classes.values().iterator();
  }
  public Object _next() {
    if (iter.hasNext()) return iter.next();
    else return null;
  }
  public void _flushCurrent() {
    iter.remove();
  }
}
```

Example from Jython

```
public class X {
  protected Iterator iter;
  public void _beginCanonical() {
    iter = classes.values().iterator();
  }
  public Object _next() {
    if (iter.hasNext()) return iter.next();
    else return null;
  }
  public void _flushCurrent() {
    iter.remove();
  }
}
```

✓ Safe API usage

Example from Jython

```
public class X {
  protected Iterator iter;
  public void _beginCanonical() {
    iter = classes.values().iterator();
  }
  public Object _next() {
    if (iter.hasNext()) return iter.next();
    else return null;
  }
  public void _flushCurrent() {
    iter.remove();
  }
}
```

✗ Protocol violation

✗ Crash through exception

✗ Not declared

Kinds of Bugs

Diverse kinds of unsafe API usages

- Invalid indexing of lists and vectors
- Iterators: Illegal next() and remove()
- Accessing non-existing elements: E.g., pop()

All unsafe API usages for download:

http://mp.binaervarianz.de/icse2012-leveraging/

Comparison with Prior Work

JDK-API usage in DaCapo:

OCD [Gabel+Su, ICSE'10]	**Our approach**
1 potential bug	54 crashing bugs
2 false positives	0 false positives

Comparison with Prior Work

JDK-API usage in DaCapo:

OCD [Gabel+Su, ICSE'10]	Our approach
1 potential bug	54 crashing bugs
2 false positives	0 false positives

DaCapo input vs. generated input

Comparison with Prior Work

JDK-API usage in DaCapo:

OCD [Gabel+Su, ICSE'10]	**Our approach**
1 potential bug	54 crashing bugs
2 false positives	0 false positives

Avoid false positives by construction

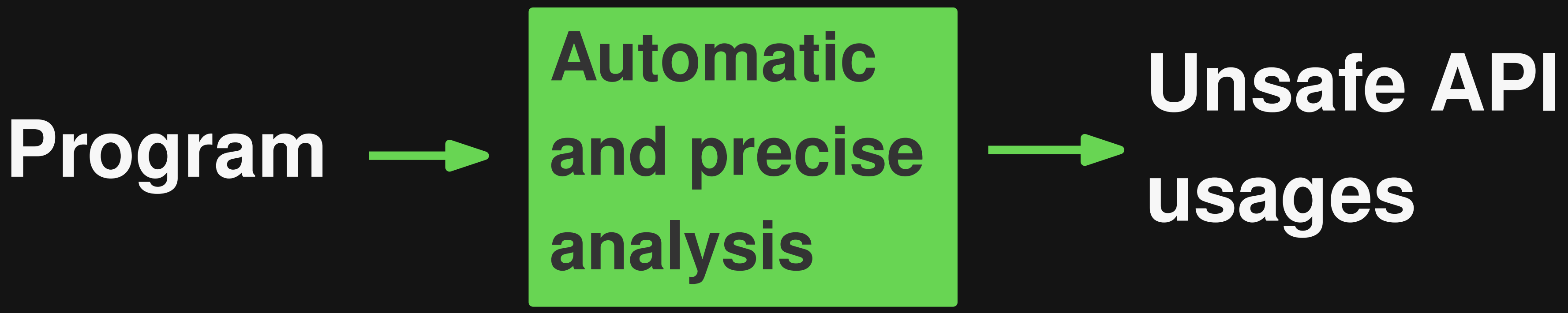
Performance

Between less then a minute and several minutes per program-API pair

Optimization: Find bugs with 5x less tests

- Static analysis: Prioritize methods
- Guide random test generator towards API-relevant parts of program

Summary


Program → Automatic and precise analysis → Unsafe API usages

- Benefits of dynamic analysis without providing input
- Find bugs with mined specifications without false positives
- Guide test generator towards API

Conclusion

Don’t waste precious developer time

Lots of testing with little effort

Photo: Alfonso Silóniz

Leveraging Test Generation and Specification Mining for Automated Bug Detection without False Positives

Michael Pradel, Thomas R. Gross

Department of Computer Science
ETH Zurich

Thank you!