Università della Svizzera italiana

Faculty of Informatics

Search-based Synthesis of Equivalent Method Sequences

Alberto Goffi, **Andrea Mattavelli**, Mauro Pezzè
University of Lugano - Switzerland

Alessandra Gorla
Saarland University - Germany

Paolo Tonella
Fondazione Bruno Kessler - Italy


Endeavour
United States
NASA
N905NA

Software is Redundant

> A system is redundant when it is able to perform **equivalent functionalities** by executing **different code**.

Software is Redundant

> A system is redundant when it is able to perform **equivalent functionalities** by executing **different code**.

- compute **identical results**
- lead to **identical states**

Equivalence in Software

Java Stack

```
Stack s = new Stack();
//...
//remove the element on top of the stack
s.pop();
```

Equivalence in Software

Java Stack

```
Stack s = new Stack();
//...
//remove the element on top of the stack
s.pop();
 s.remove(size() - 1);
 s.removeElementAt(size() - 1);
```

Equivalence in Software

Java Stack

```
Stack s = new Stack();
//...
//remove the element on top of the stack
s.pop();
  s.remove(size() - 1);
  s.removeElementAt(size() - 1);
```

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
```

Equivalence in Software

Java Stack

```
Stack s = new Stack();
//...
//remove the element on top of the stack
s.pop();
  s.remove(size() - 1);
  s.removeElementAt(size() - 1);
```

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
                  = t.toDateMidnight().toDateTime();
                  = t.withTimeAtStartOfDay();
```

Exploiting Equivalence

Automatic Recc

Antonio Carzaniga* Alessandra G

*University
Faculty of I
Lugano, Sw

***Abstract*—We present a technique to make ap**
silient to failures. This technique is intended t
faulty application functional in the field while t
work on permanent and radical fixes. We target
in applications built on reusable components. In
technique exploits the intrinsic redundancy of thos
by identifying *workarounds* consisting of alterna
the faulty components that avoid the failure. The
currently implemented for Java applications but
no assumptions about the nature of the applicatio
without interrupting the execution flow of the ap
without restarting its components. We demonstrate
this technique on four mid-size applications and
libraries of reusable components affected by rea
faults. In these cases the technique is effective
the application fully functional with between 19
of the failure-causing faults, depending on the ap
experiments also show that the technique incurs
runtime overhead in all cases.

I. Introduction

Software systems are sometimes released and t
with faults, and those faults may cause field fail
happens despite the best effort and the rigorou
developers and testers. Furthermore, even when
reported to developers, field failures may take a
diagnose and eliminate. As a perhaps extreme but
unique example, consider fault n. 3655 in the Fir
which was reported first in March 1999 and oth
the following ten years, and is yet to be cor
time of writing of this paper (summer 2012).[1] T
and longevity of faults in deployed applications
to the difficulty of reproducing failures in the
environment or more generally to the difficulty
and eliminating faults at a cost and with a schedu
with the objectives of developers and users.

At any rate, dealing with faults that escap
environment seems to be a necessity for modern
in fact, several lines of research have been devote
or at least mitigating the effects of faults in deplo
A primary example is software fault tolerance
hardware fault-tolerance techniques such as RA
ware fault tolerance is based on the idea of p
executing different versions of an application (or
as to obtain a correct behavior from the majority
even just one) of the versions [2], [3].

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=3655

Automatic Worka

Antonio Carzaniga, Aless

{antonio.carzaniga|alessa

ABSTRACT

We present a technique that finds and executes w
for faulty Web applications automatically and
Automatic workarounds exploit the inherent red
Web applications, whereby a functionality of the
can be obtained through different sequences of
of Web APIs. In general, runtime workarounds
in response to a failure, and require that the app
main in a consistent state before and after the e
a workaround. Therefore, they are ideally suite
active Web applications, since those allow the
as a failure detector with minimal effort, and
use read-only state or manage their state throu
actional data store. In this paper we focus on f
in the access libraries of widely used Web applic
as Google Maps. We start by classifying a nu
ported faults of the Google Maps and YouTube
have known workarounds. From those we derive
general and API-specific program-rewriting rules
then apply to other faults for which no workaroun
Our experiments show that workarounds can be
ployed within Web applications, through a simpl
plug-in, and that program-rewriting rules derive
mentary properties of a common library can be
finding valid and previously unknown workarou

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and D
Error handling and recovery

General Terms

Reliability, Design

Keywords

Automatic Workarounds, Web Applications, We

*Mauro Pezzè is also with the University of Milan

Permission to make digital or hard copies of all or part of
personal or classroom use is granted without fee provided t
not made or distributed for profit or commercial advantage a
bear this notice and the full citation on the first page. To copy
republish, to post on servers or to redistribute to lists, requires
permission and/or a fee.
FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA
Copyright 2010 ACM 978-1-60558-791-2/10/11 ...$10.00.

Internation

On the Enhancement of BPE

Sattanathan Subram
Ghita Kou

[1]INRIA Saclay-Île-
[2]Precise, Univ
[3]IBM India R
[4]Comlab,
[5]College of Information Te

Abstract

The dynamic nature of the Internet poses var
to the successful execution of composite Web
ures are samples of these challenges. It
dressed for the smooth progress of Web servi
Unfortunately, the de facto *standard for mo*
tion namely BPEL is not equipped with mec
Web services "heal" themselves in case of fa
tion, current BPEL engines lack appropria
permit satisfying self-healing requirements
a transparent way functional failures of com
vices during runtime. This paper presents
enhancing BPEL engines with such facilities
ment happens through the following steps: i
categories of failures, develop solutions t
recover from these failures, and suggest a
tensions to BPEL engines (with focus on
support these solutions. We also present a p
prototype that illustrates our ideas.
Keywords. *BPEL, Composition, Self-healin*

1. Introduction

Modern B2B applications requirements
to compose Web services, which results in d
posite Web services. Simply put, compo
making independent Web services interac
other according to a specific business log
ent specification languages of composition
the Business Process Execution Language
the Web Service Choreography Interface (
though BPEL is the *de facto* standard fo
composition, current engines that impleme
fications suffer from a major limitation. I
gines do not allow changes to take place i
a composition specification execution-strea
designers envisage alternatives to address a

978-0-7695-3297-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SAINT.2008.12

Cross
from Intrins

Antonio Ca
University of
Switzerla
antonio.carzani

Alessandra Gorla
Saarland University
Germany
gorla@st.cs.uni-saarland.de

ABSTRACT

Despite the recent advances in automatic tes
testers must still write test oracles manually. If
fications are available, it might be possible to
procedures derived from those specifications.
technique that is based on a form of specificat
leverages more information from the system un
assume that the system under test is somewha
in the sense that some operations are designe
like others but their executions are different. O
in this and previous work indicates that this
exists and is easily documented. We then genera
cross-checking the execution of a test with the
which we replace some operations with redunda
develop this notion of cross-checking oracles i
technique to automatically insert oracles into u
experimental evaluation shows that cross-chec
used in combination with automatic test gen
niques, can be very effective in revealing faul
they can even improve good hand-written test

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Pro
cation; D.2.5 [**Software Engineering**]: Testing
ging

General Terms

Verification

Keywords

Redundancy, test oracles, oracle generation

Permission to make digital or hard copies of all or part
personal or classroom use is granted without fee provided
not made or distributed for profit or commercial advantage
bear this notice and the full citation on the first page. To co
republish, to post on servers or to redistribute to lists, requir
permission and/or a fee.
*ICSE'*14, May 31 – June 7, 2014, Hyderabad, India
Copyright 14 ACM 978-1-4503-2756-5/14/05 ...$15.00.

Metamorphic Testing and Beyond *

T. Y. Chen †, F.-C. Kuo †, T. H. Tse ‡ §, Zhi Quan Zhou †

†*School of Information Technology*
Swinburne University of Technology
Hawthorn, Victoria 3122, Australia
Email: {tchen, dkuo, zhzhou}@it.swin.edu.au

‡*Department of Computer Science and Information Systems*
The University of Hong Kong
Pokfulam Road, Hong Kong
Email: tse@csis.hku.hk

Abstract

When testing a program, correctly executed test cases are seldom explored further, even though they may carry useful information. Metamorphic testing proposes to generate follow-up test cases to check important properties of the target function. It does not need a human oracle for output prediction and comparison. In this paper, we highlight the basic concepts of metamorphic testing and some interesting extensions in the areas of program testing, proving, and debugging. Future research directions are also proposed.

Keywords: *Follow-up test cases, metamorphic testing, semi-proving, successful test case, test case selection strategy, testing oracle*

1. Introduction

It is impractical, if not impossible, to test a program with all conceivable inputs [1]. Instead, we should aim at selecting test cases with higher probabilities of revealing program failures. Hence, a lot of research has been done on developing *test case selection strategies*.

A *successful test case* is one on which the program computes correctly. Since successful test cases do not reveal any failure, they are conventionally considered useless [22] and thus discarded by testers or merely retained for reuse in regression testing later. We note, however, that successful test cases *do* carry useful information, albeit seldom explored. Fault-based testing [21], for example, is a significant attempt to make use of such information. In fault-based testing, if a program has successfully passed all the test cases, then it can be guaranteed to be free from certain types of faults. Unfortunately, most testing methods are not fault-based, and most test cases are executed successfully. Thus, some valuable information that results from program testing will remain buried and unused.

Another limitation of software testing is the *oracle problem* [23]. An *oracle* is a mechanism against which people can decide whether the outcome of the program on test cases is correct. In some situations, the oracle is not available or is too expensive to be applied [23]. In cryptography systems, for example, large number arithmetic is usually involved. It is very expensive to verify the correctness of a computed result. Other examples include deciding the equivalence between the source and object codes when testing a compiler; and deciding the correctness of an output when testing a program that performs numerical integration. Furthermore, even when manual prediction and comparison of testing results are possible, they are often time consuming and error prone [18, 20]. The oracle problem is "one of the most difficult tasks in software testing" [20] but is often ignored in the testing theory [18].

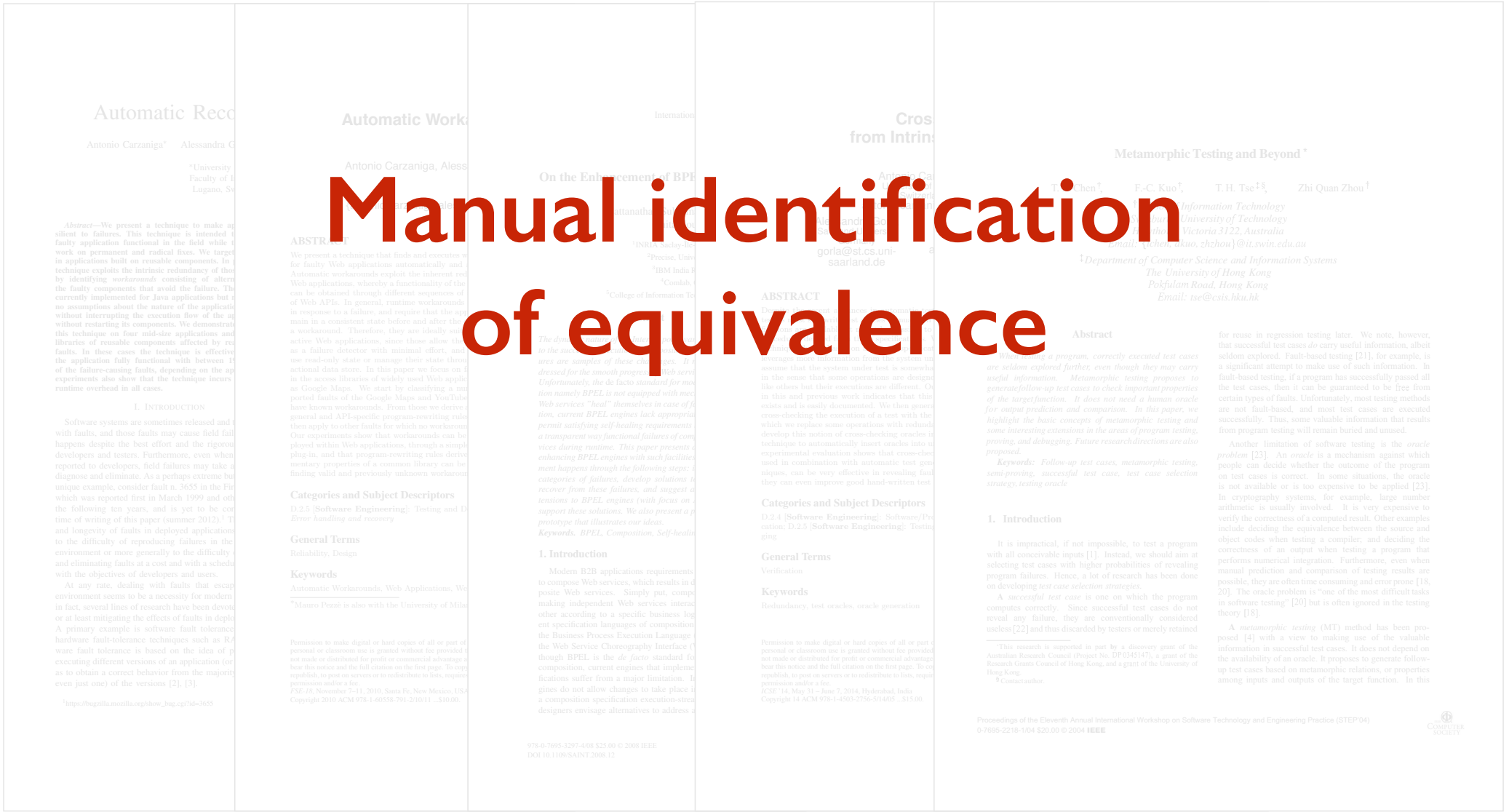
A *metamorphic testing* (MT) method has been proposed [4] with a view to making use of the valuable information in successful test cases. It does not depend on the availability of an oracle. It proposes to generate follow-up test cases based on metamorphic relations, or properties among inputs and outputs of the target function. In this

*This research is supported in part **by** a discovery grant of the Australian Research Council (Project No. DP 0345147), a grant of the Research Grants Council of Hong Kong, and a grant of the University of Hong Kong.

§ Contact author.

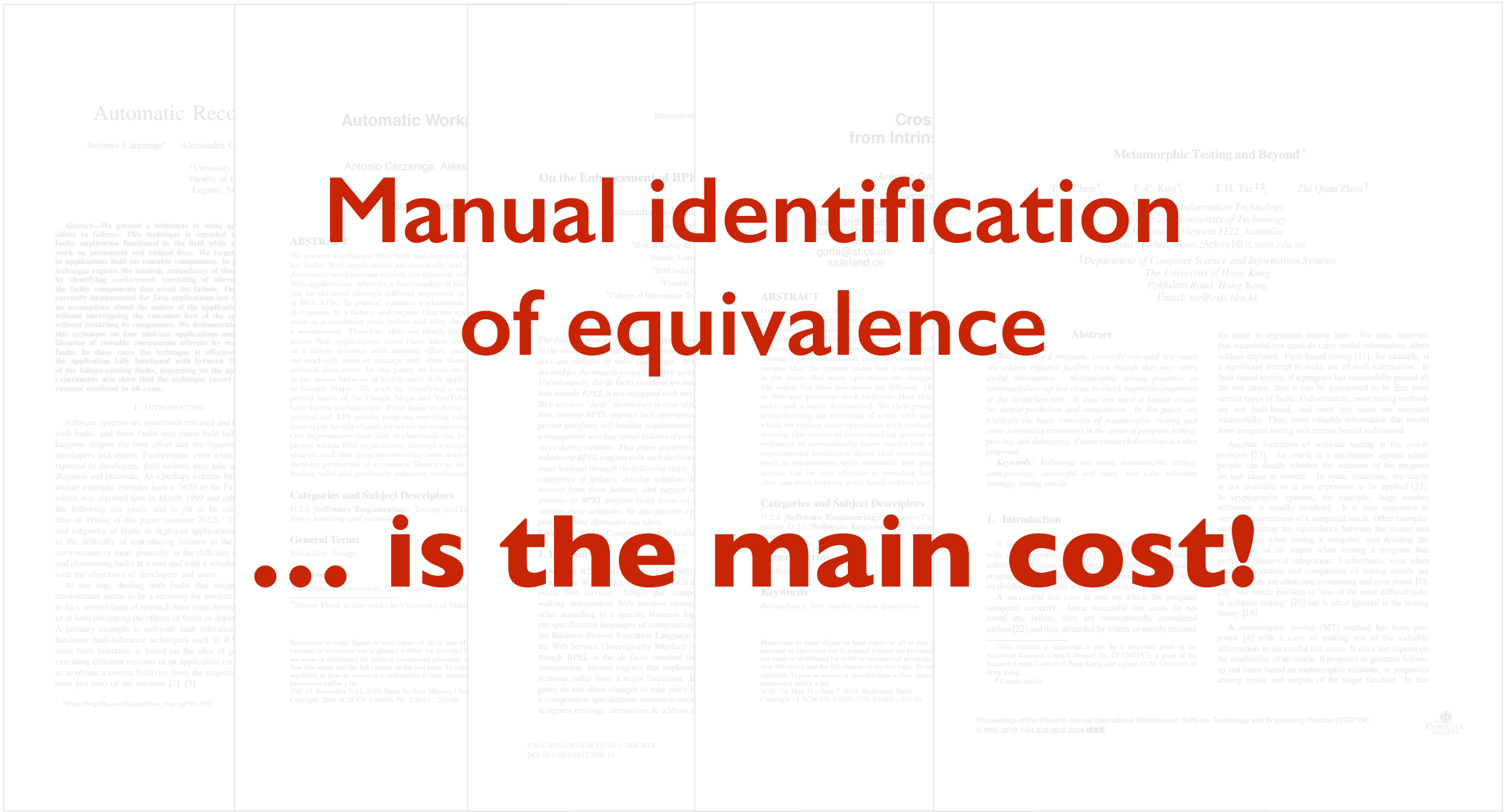
Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'04)
0-7695-2218-1/04 $20.00 © 2004 **IEEE**

Exploiting Equivalence



Manual identification of equivalence

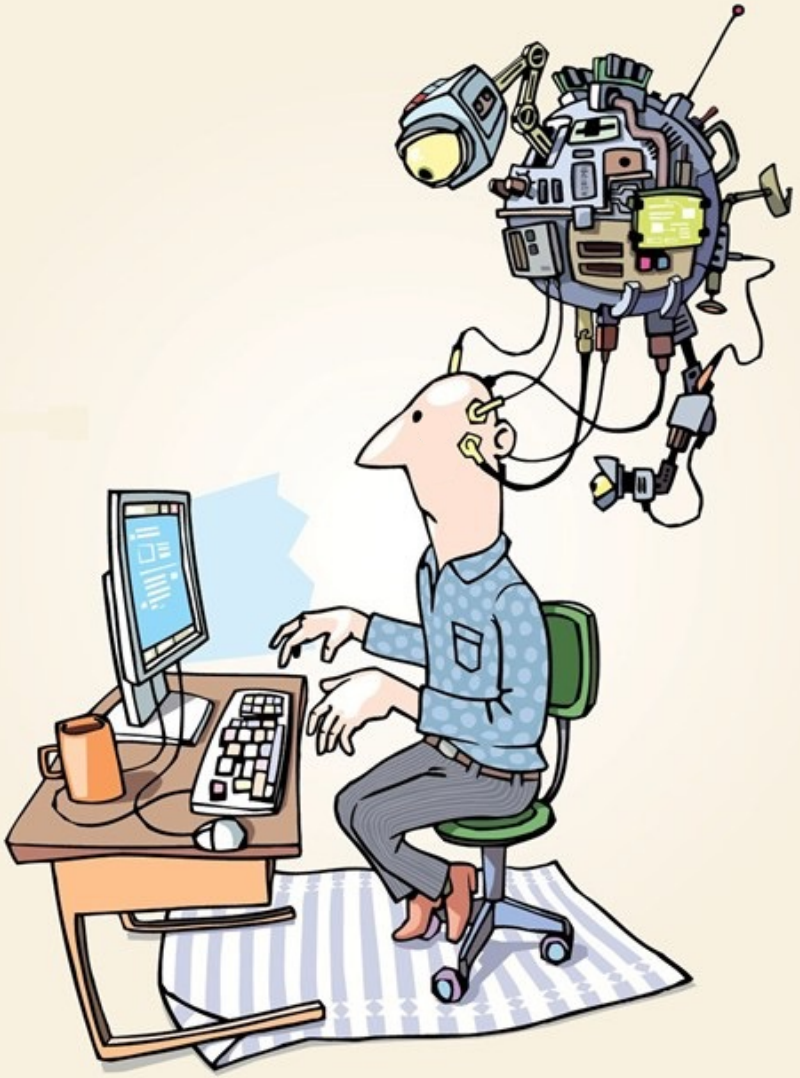
Exploiting Equivalence



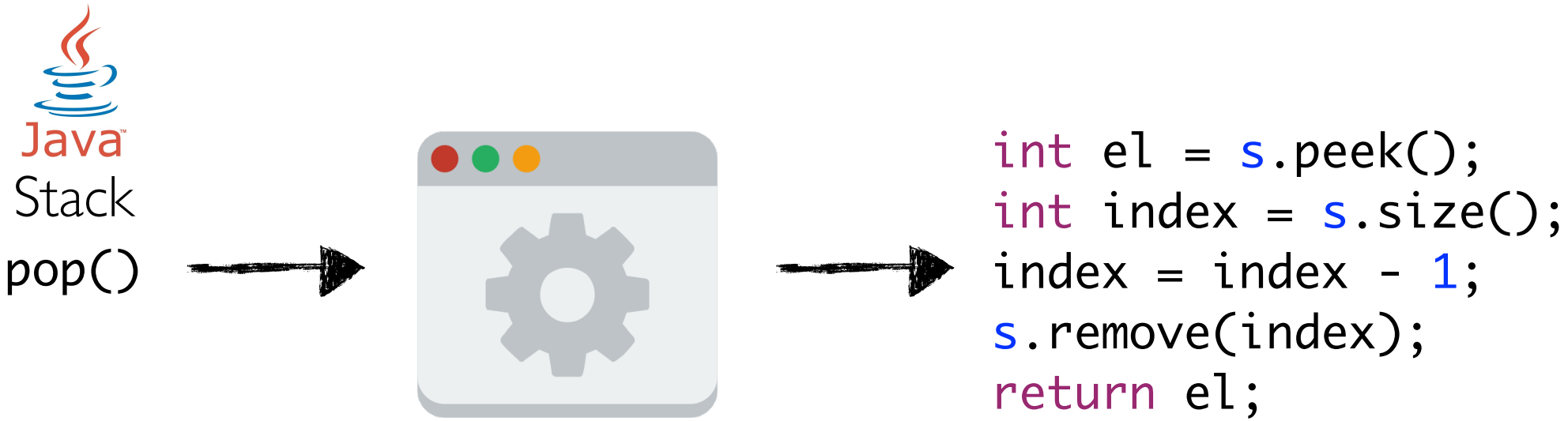
Manual identification of equivalence

… is the main cost!

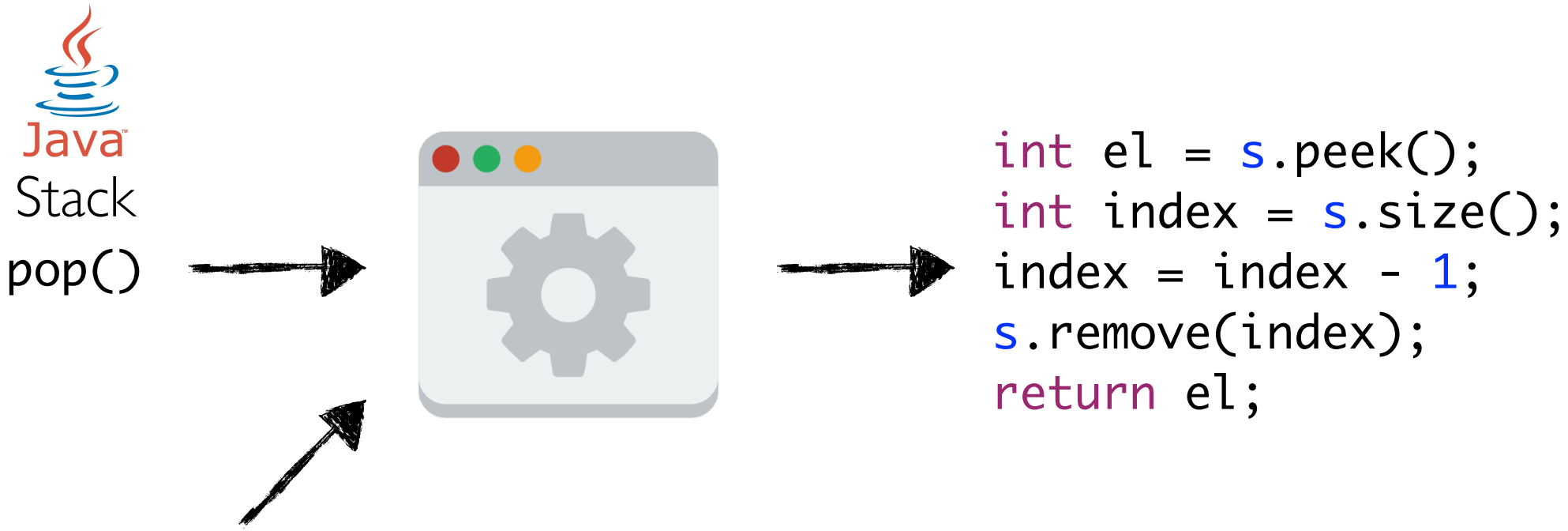
Automatic Synthesis of Equivalent Method Sequences



Automatic Synthesis of Equivalences



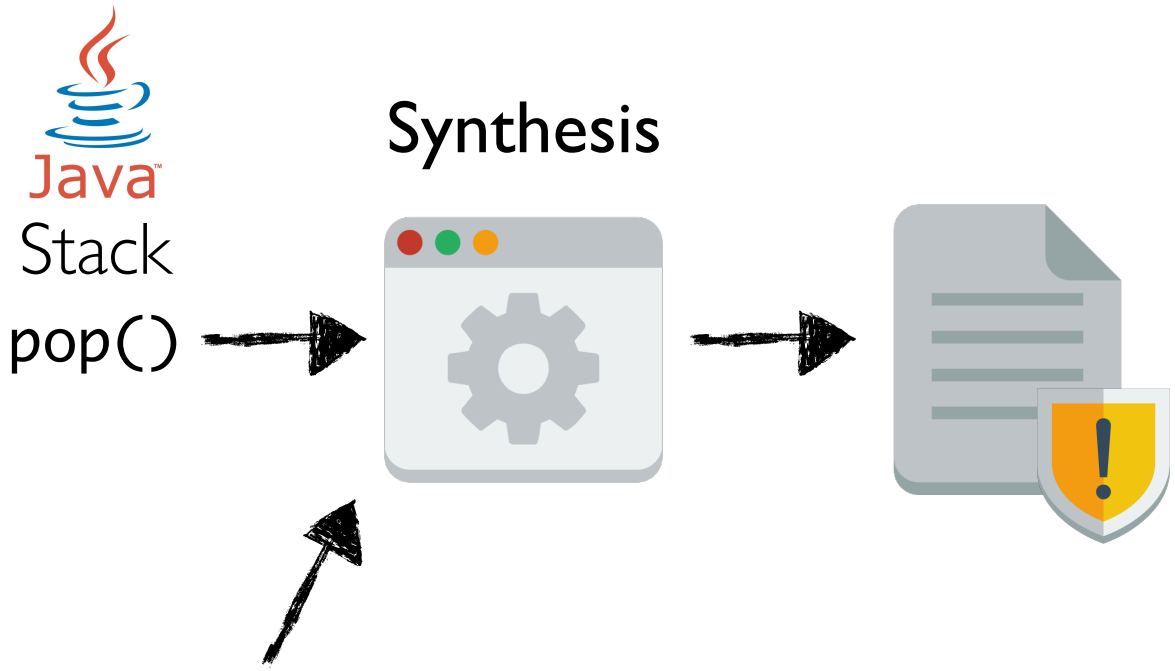
Automatic Synthesis of Equivalences




```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

Automatic Synthesis of Equivalences

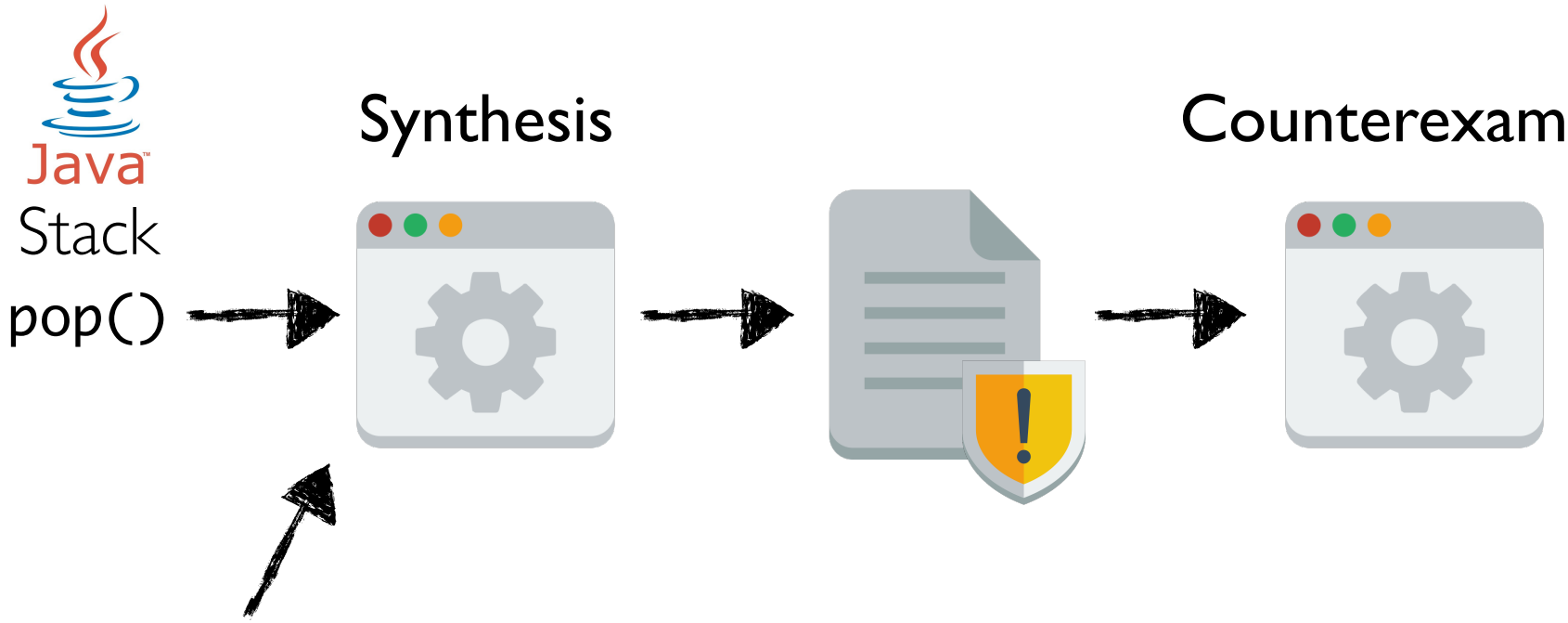


Execution scenarios

```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

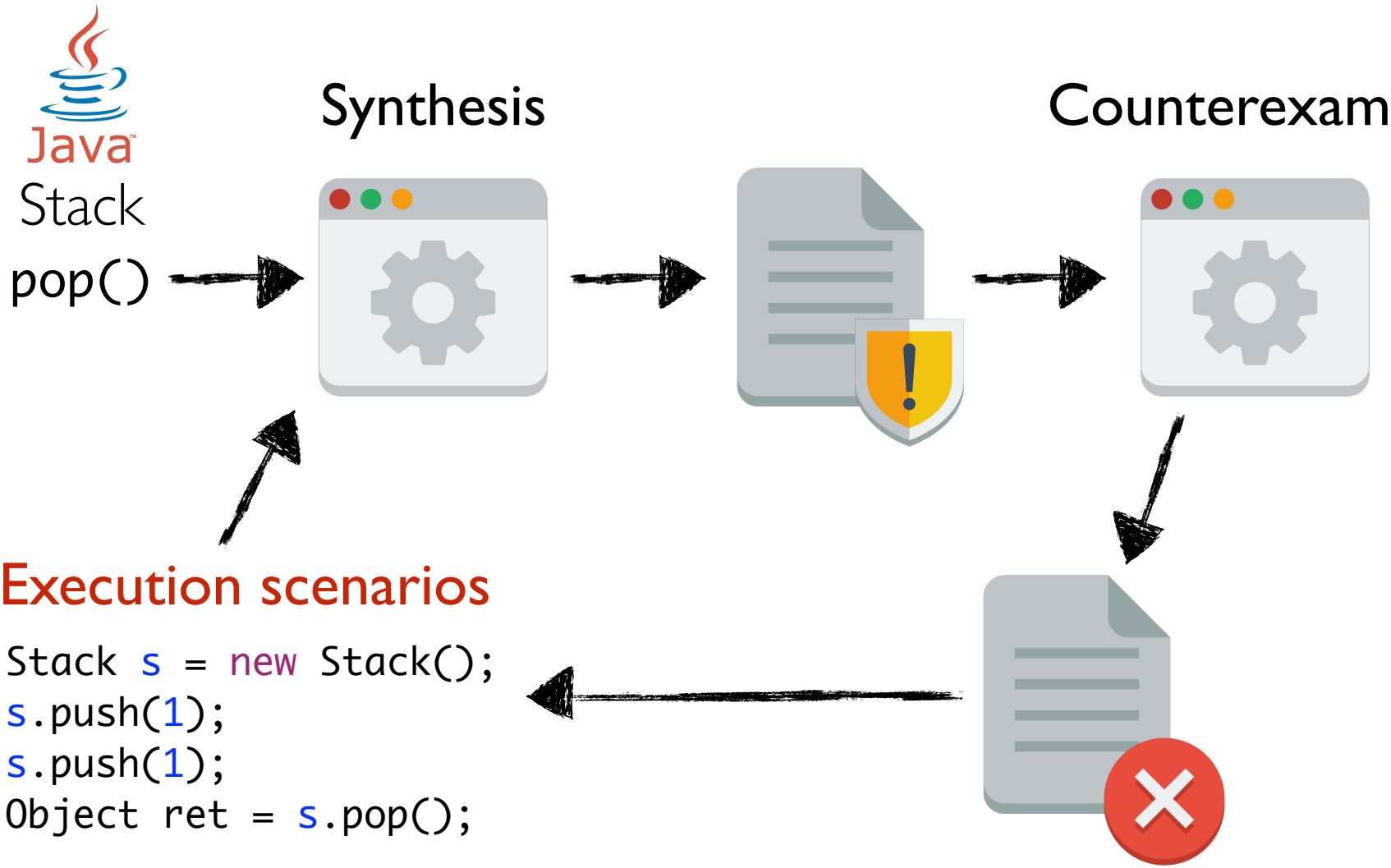
Automatic Synthesis of Equivalences




```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();

Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

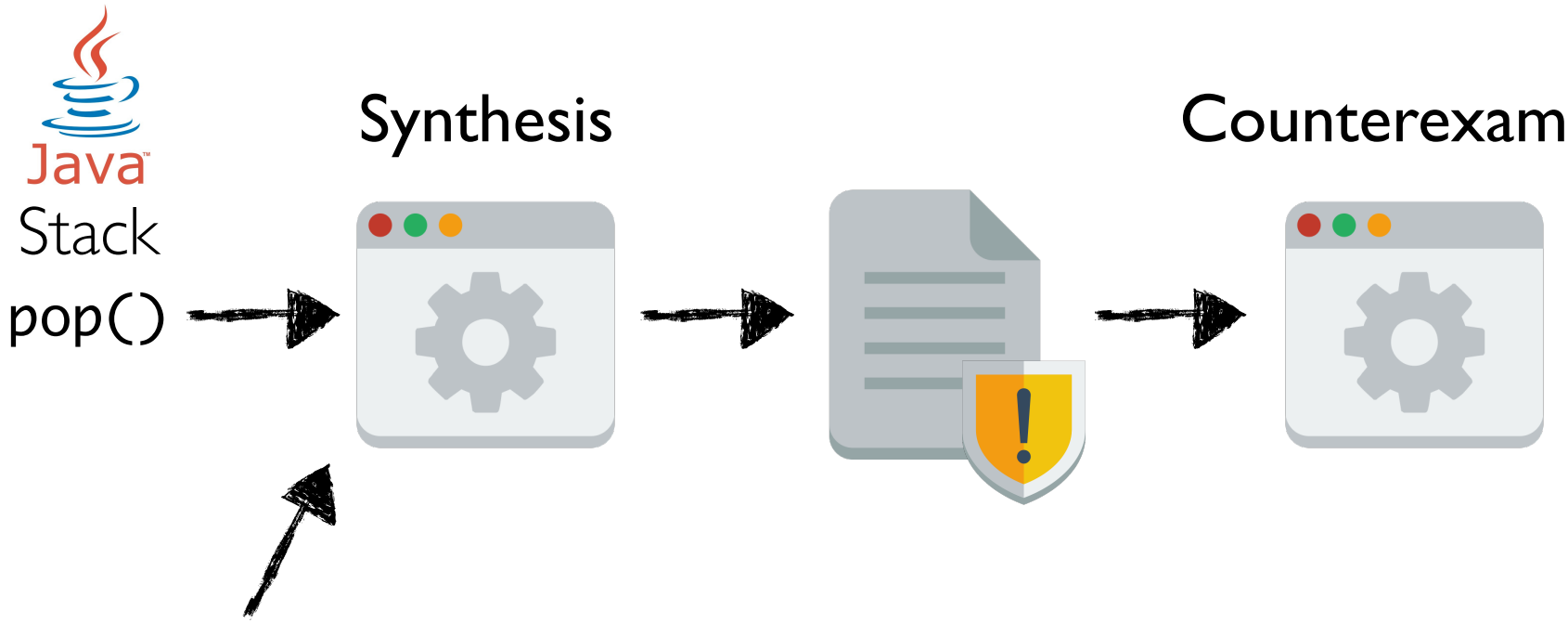
Automatic Synthesis of Equivalences




```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

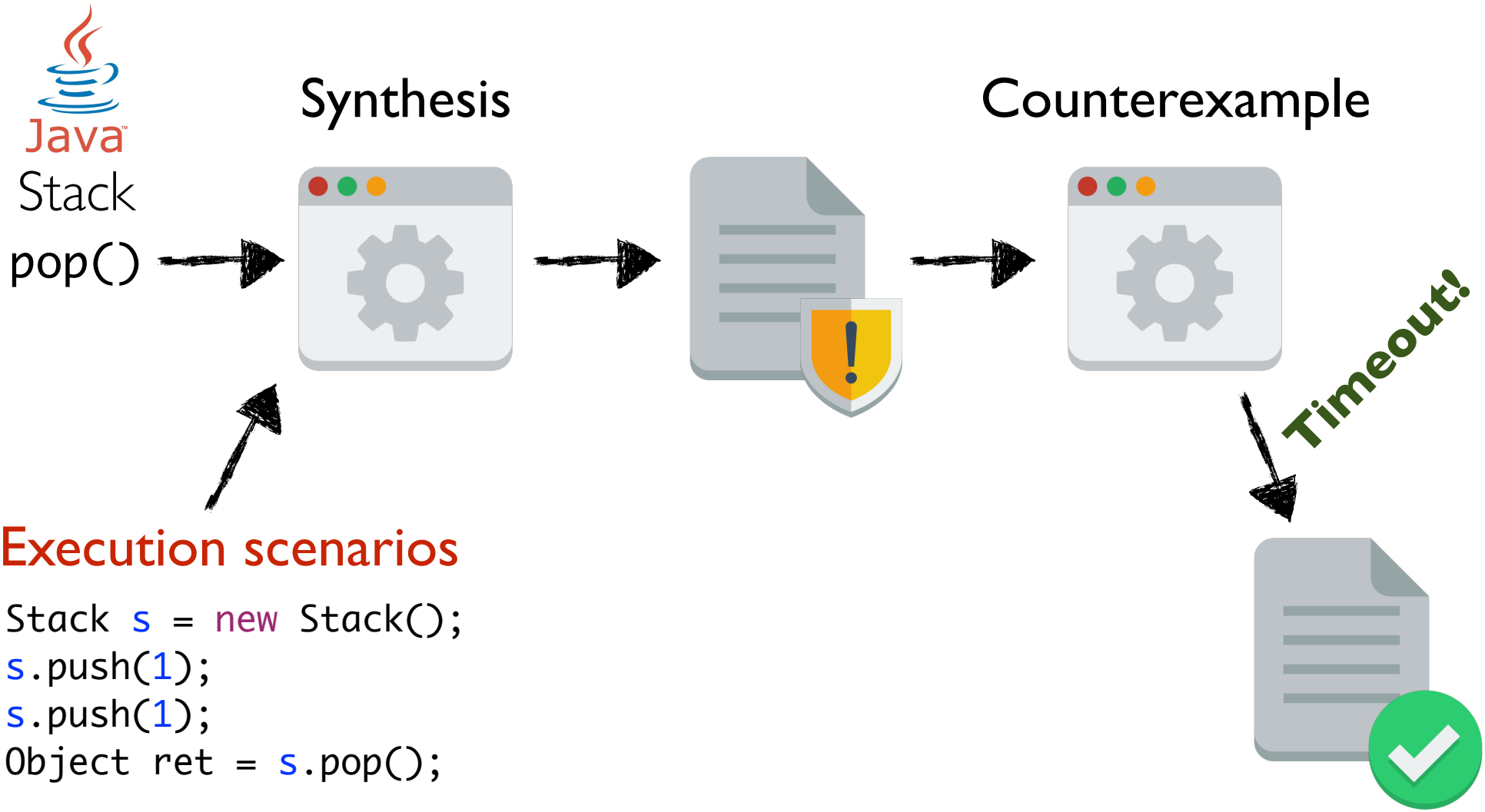
Automatic Synthesis of Equivalences




```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();

Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

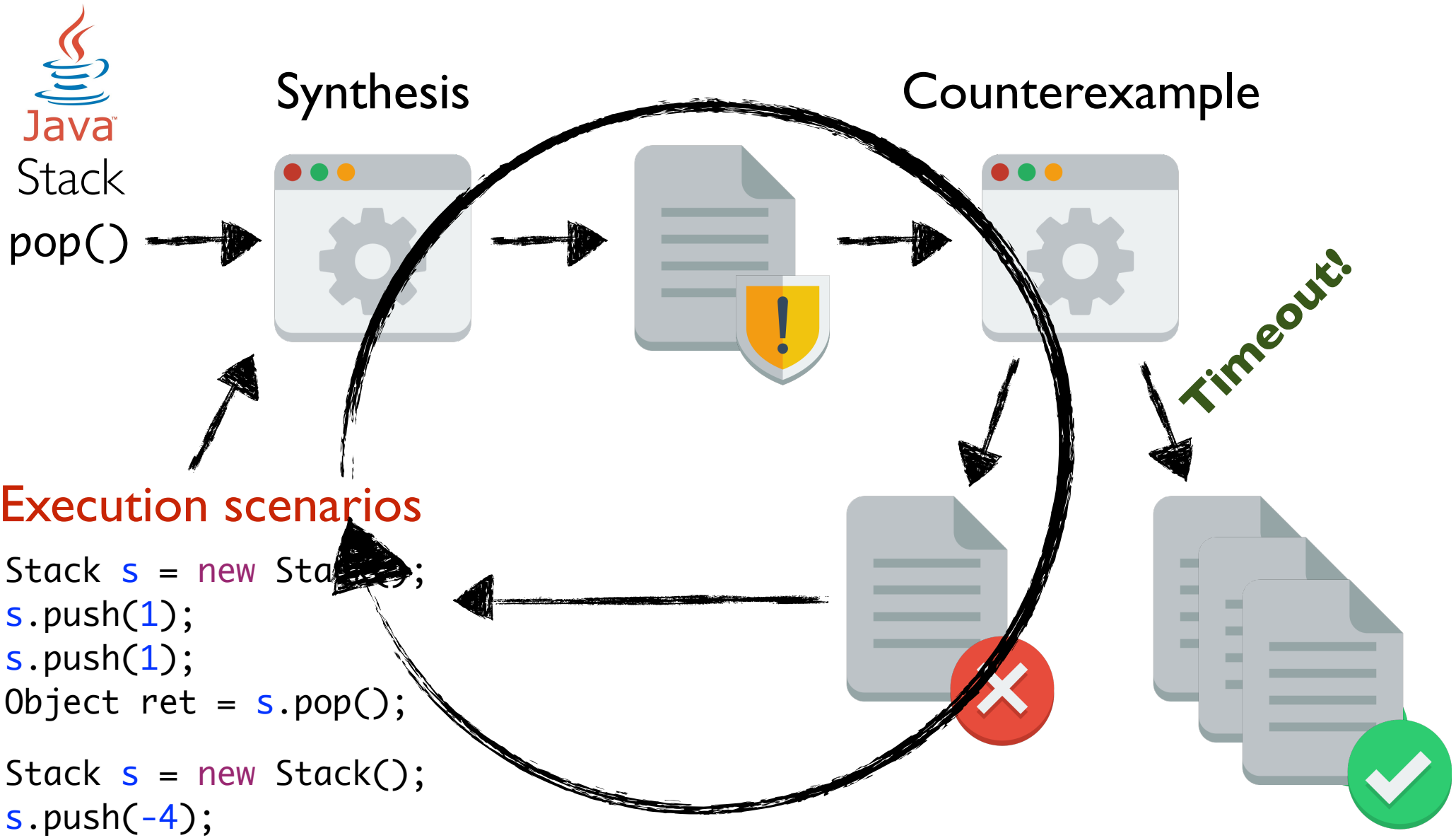
Automatic Synthesis of Equivalences



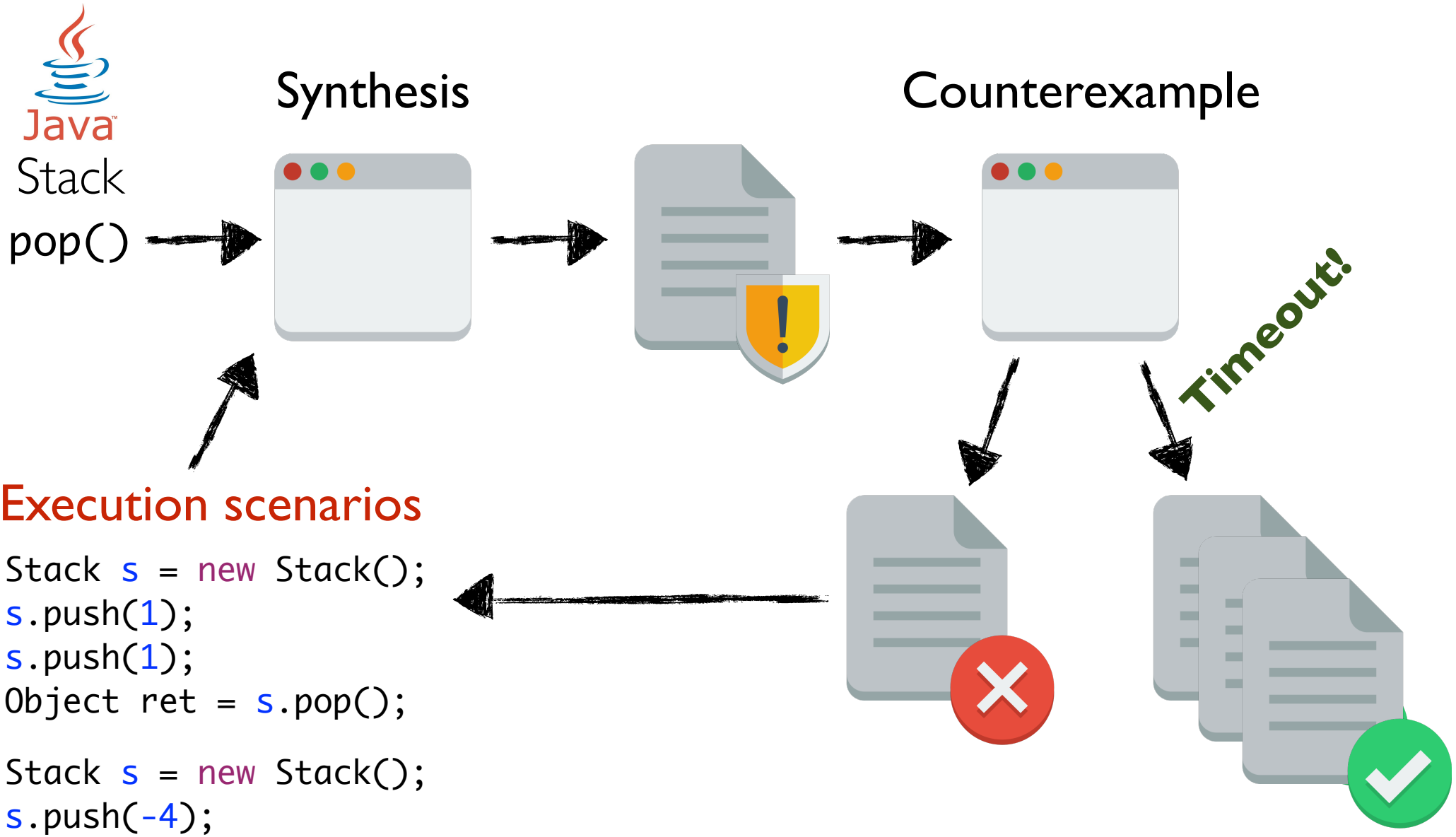

```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

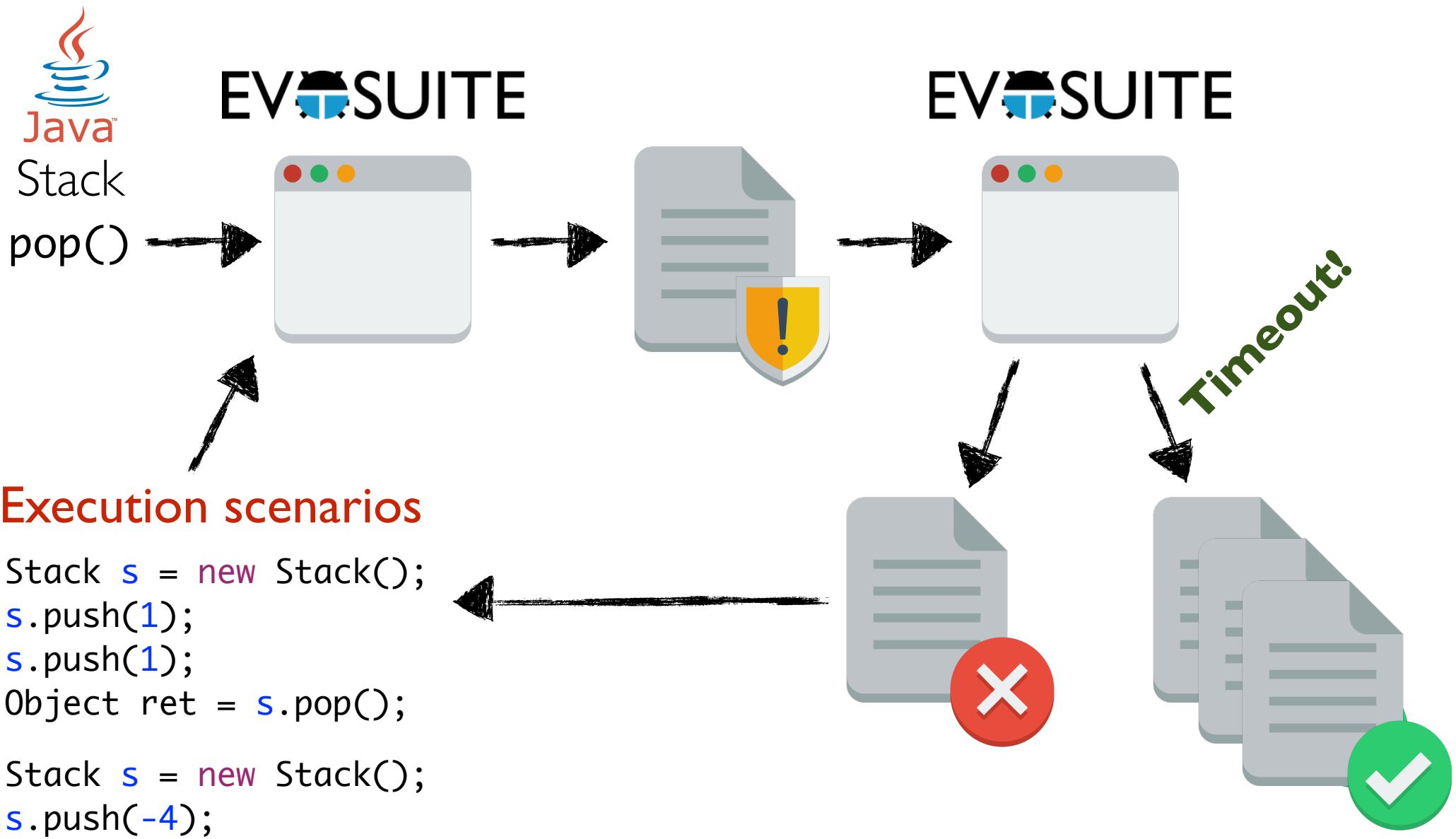
Automatic Synthesis of Equivalences



Search-based Synthesis of Equivalences



Search-based Synthesis of Equivalences



Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
   if ( condition ) {
      // equivalent!
   }
}
```

Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( condition ) {
     // equivalent!
  }
}
```

On **all** execution scenarios:

- compute **identical** results
- lead to **identical** object states

Equivalence Synthesis as TCG Problem

EVOSUITE

pop()

Java
Stack

```
public void method_under_test() {
  if ( condition ) {
    // equivalent!
  }
}
```

Execution scenarios

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( condition ) {
    // equivalent!
  }
}
```

pop()

Java
Stack

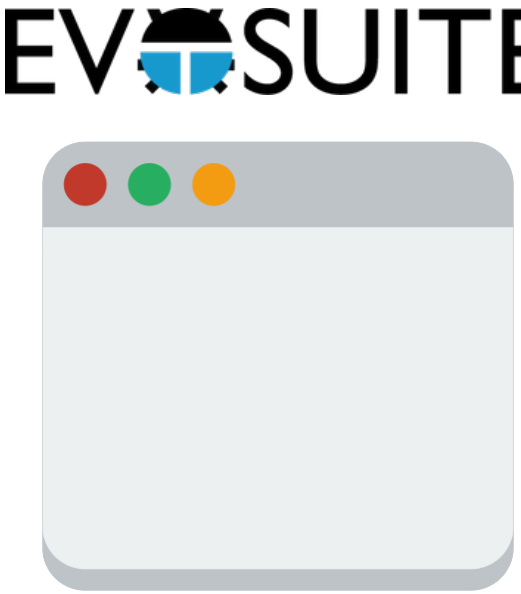
```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = ? ? ? ? ? ;
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret = ? ? ? ? ? ;
```

Equivalence Synthesis as TCG Problem



pop()

Java
Stack

```
public void method_under_test() {
  if (      ==      &&      ==      &&
                                  ){
    // equivalent!
  }
}
```

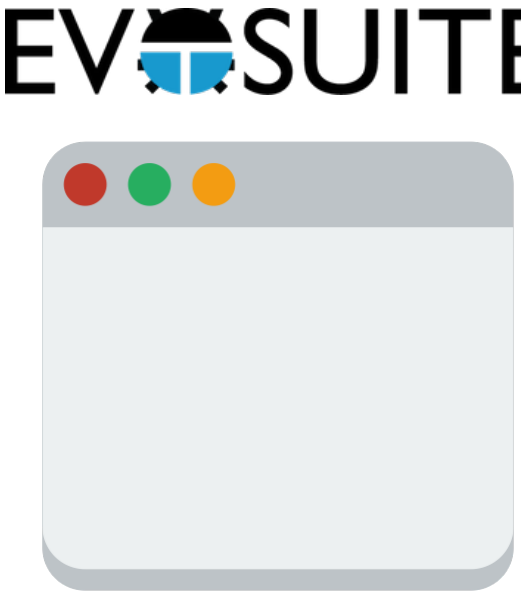
```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = ? ? ? ? ? ;
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret = ? ? ? ? ? ;
```

Equivalence Synthesis as TCG Problem



pop()

Java
Stack

```
public void method_under_test() {
  if (      ==      &&      ==      &&
      ==      &&      ==      ){
    // equivalent!
  }
}
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret =  ? ? ? ? ? ;
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret =  ? ? ? ? ? ;
```

Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if (      ==      &&      ==      &&
            ==      &&      ==      ){
    // equivalent!
  }
}
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret =  ? ? ? ? ? ;
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret =  ? ? ? ? ? ;
```

Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( [ ] == [ ] && [ ] == [ ] &&
       [ ] == [ ] && [ ] == [ ]){
    // equivalent!
  }
}
```

`remove(0)`

Java Stack

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = ? ? ? ? ? ;
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret = ? ? ? ? ? ;
```

Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( ____ == ____ && ____ == ____ &&
       ____ == ____ && ____ == ____ ){
    // equivalent!
  }
}
```

remove(0)

Java
Stack

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.remove(0);
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.remove(0);
```

Equivalence Synthesis as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( [1] == [1] && [] == [] &&

       1 == 1 && -4 == -4 ){
    // equivalent!
  }
}
```

`remove(0)`

Java
Stack

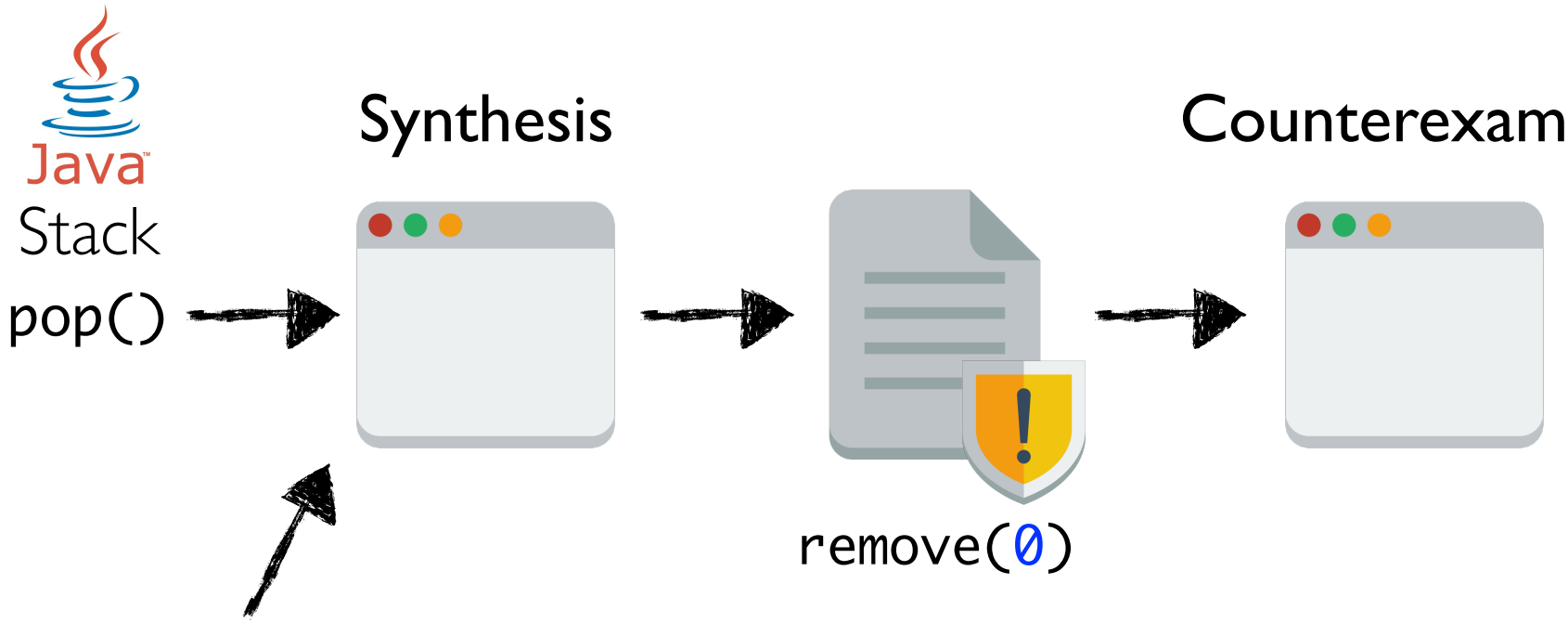
```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(1);
s.push(1);
int ret = s.remove(0);
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(-4);
int ret = s.remove(0);
```

Search-based Synthesis of Equivalences



Execution scenarios

```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();
```

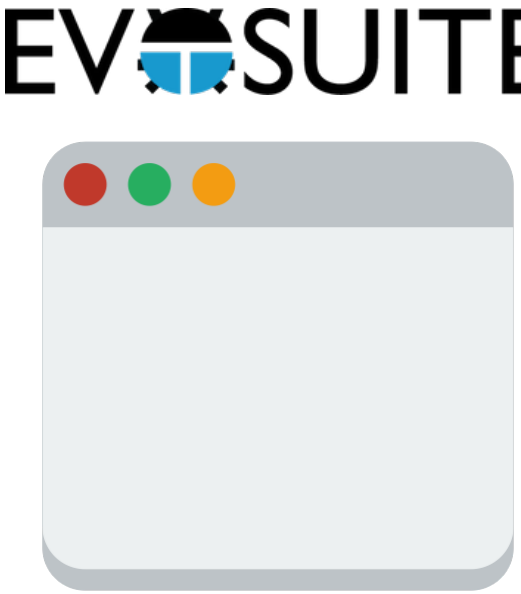
```
Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

Counterexample as TCG Problem

EVOSUITE

```
public void method_under_test() {
   if ( condition ) {
      // counterexample
   }
}
```

Counterexample as TCG Problem




```
public void method_under_test() {
   if ( condition ) {
      // counterexample
   }
}
```

On **one** execution scenario:

- compute **different** results, or
- lead to **different** object states

Counterexample as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( condition ) {
    // counterexample
  }
}
```

remove(0)

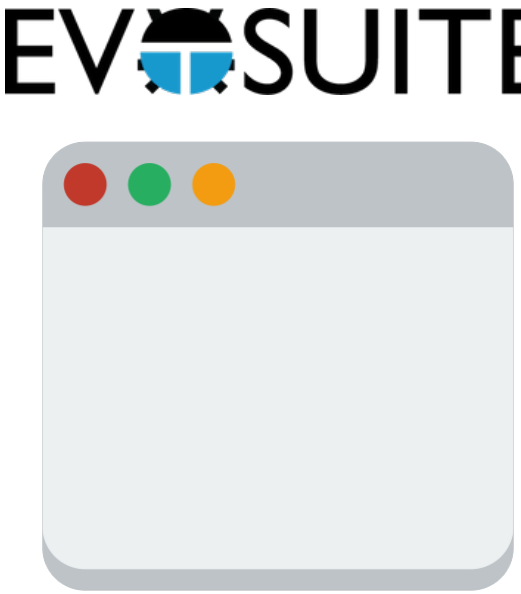
Java

Stack

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.pop();
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.remove(0);
```

Counterexample as TCG Problem



remove(0)

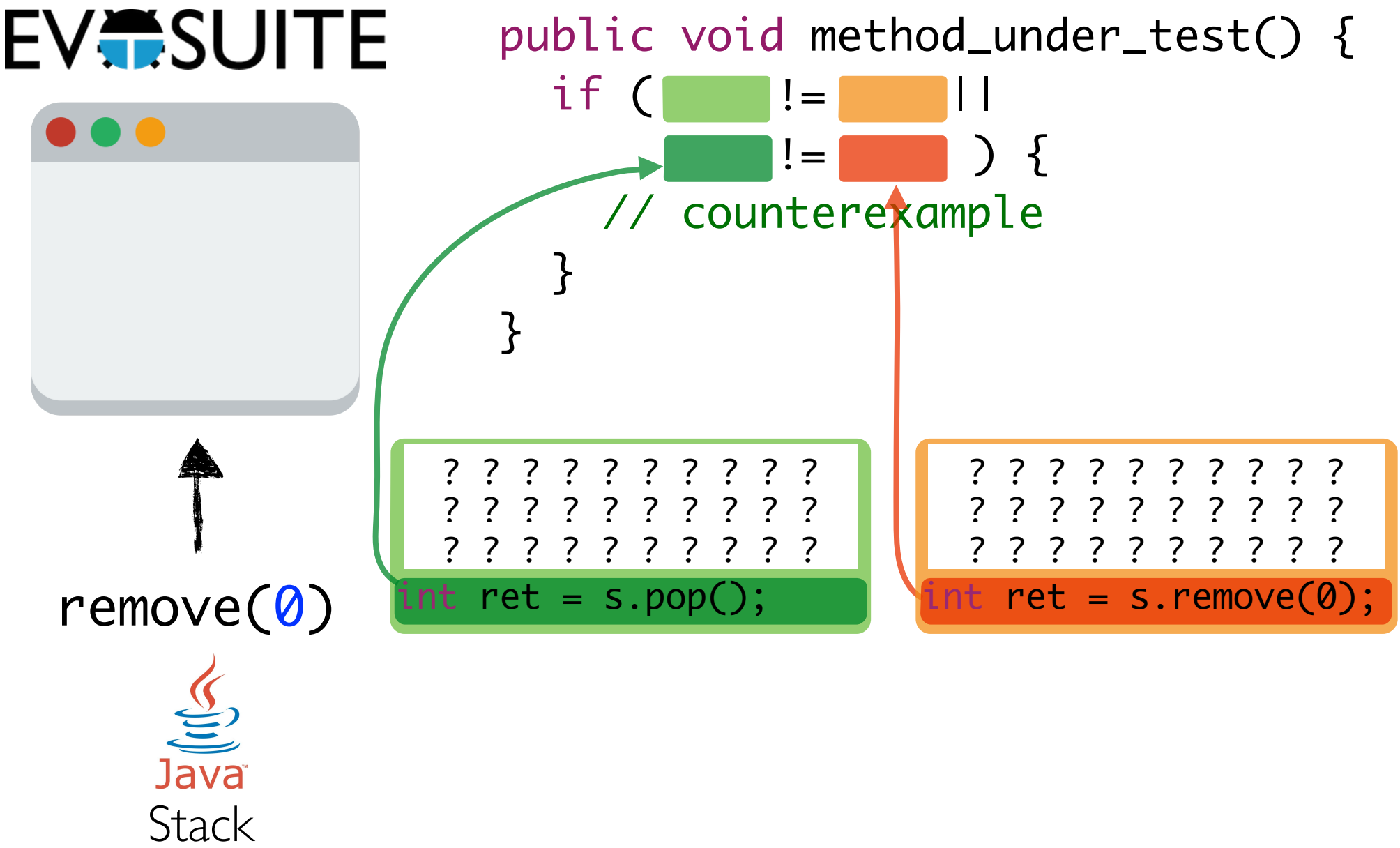
Java
Stack

```
public void method_under_test() {
  if (          !=          ||
                !=          ) {
    // counterexample
  }
}
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.pop();
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.remove(0);
```

Counterexample as TCG Problem



Counterexample as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( [ ] != [ ] ||
       [ ] != [ ] ) {
    // counterexample
  }
}
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.pop();
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.remove(0);
```

Counterexample as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( [ ] != [ ] ||
       [ ] != [ ] ) {
    // counterexample
  }
}
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.pop();
```

```
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
int ret = s.remove(0);
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
```

Counterexample as TCG Problem

EVOSUITE

```
public void method_under_test() {
  if ( ____ != ____ ||
       ____ != ____ ) {
    // counterexample
  }
}
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
int ret = s.remove(0);
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
```

Counterexample as TCG Problem

EVOSUITE

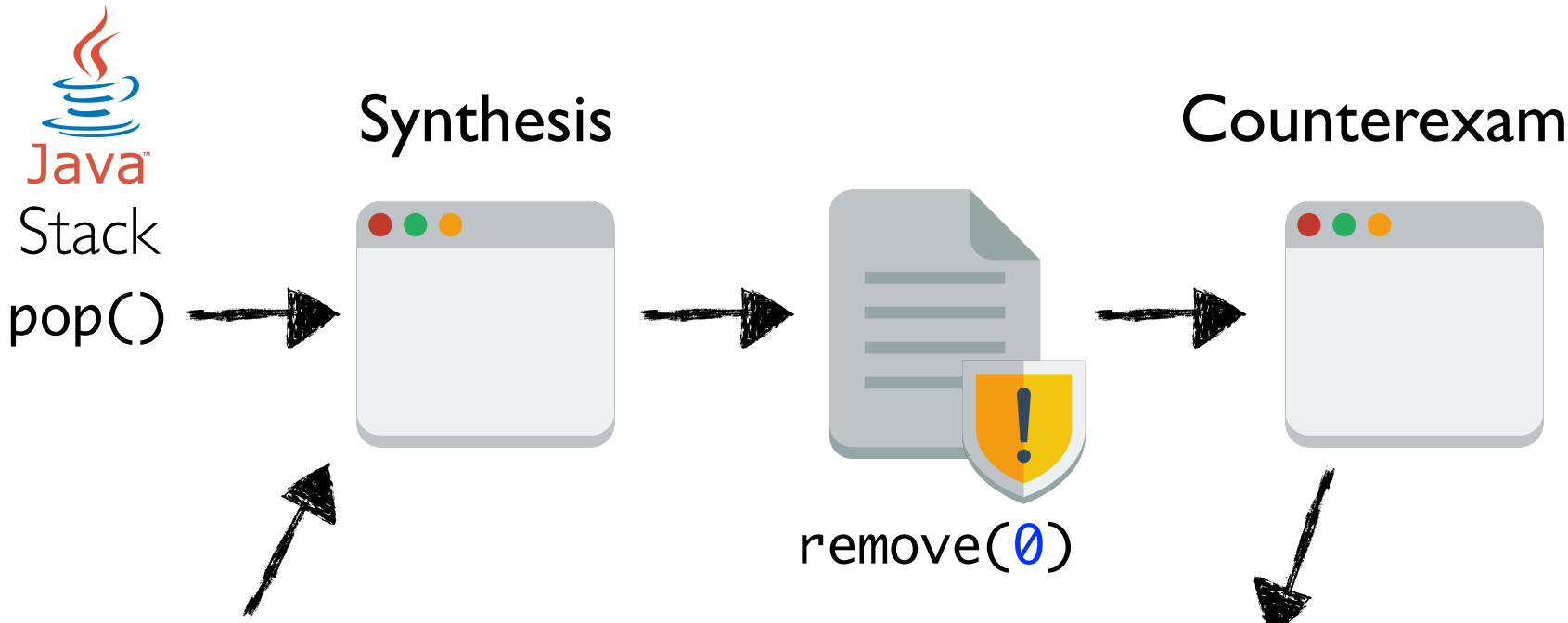
```
public void method_under_test() {
  if ( [0] != [1] ||
       1  !=  0   ) {
    // counterexample
  }
}
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
int ret = s.pop();
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
int ret = s.remove(0);
```

```
Stack s = new Stack();
s.push(0);
s.push(1);
```

Search-based Synthesis of Equivalences



Execution scenarios

```
Stack s = new Stack();
s.push(1);
s.push(1);
Object ret = s.pop();
```

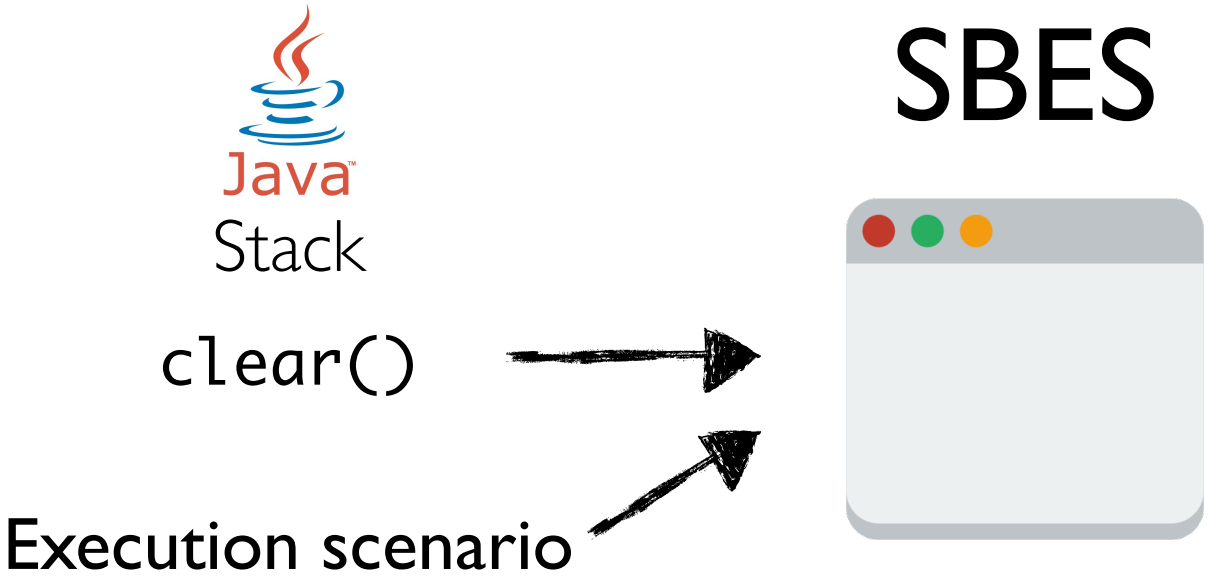
```
Stack s = new Stack();
s.push(-4);
Object ret = s.pop();
```

```
Stack s = new Stack();
s.push(0); s.push(1);
```

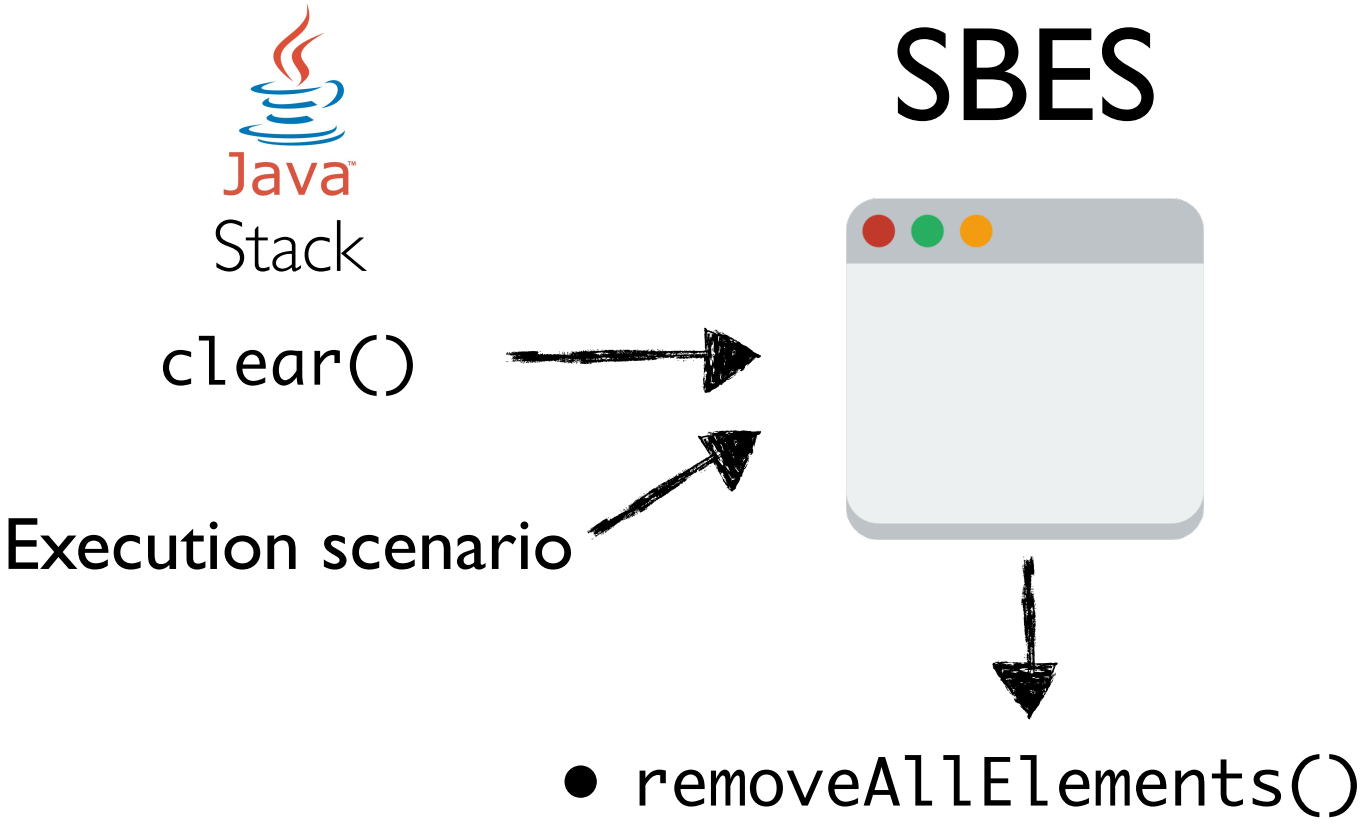
Search-based Synthesis of Equivalences

SBES

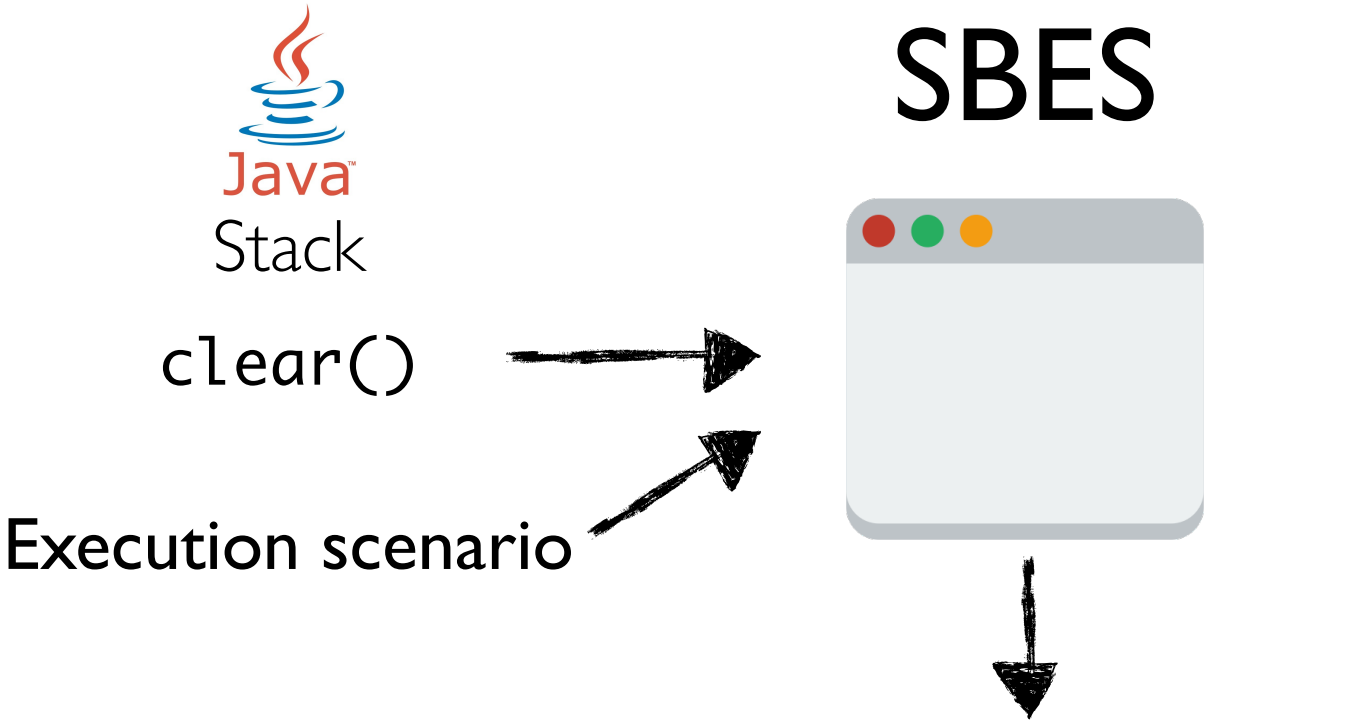
Search-based Synthesis of Equivalences



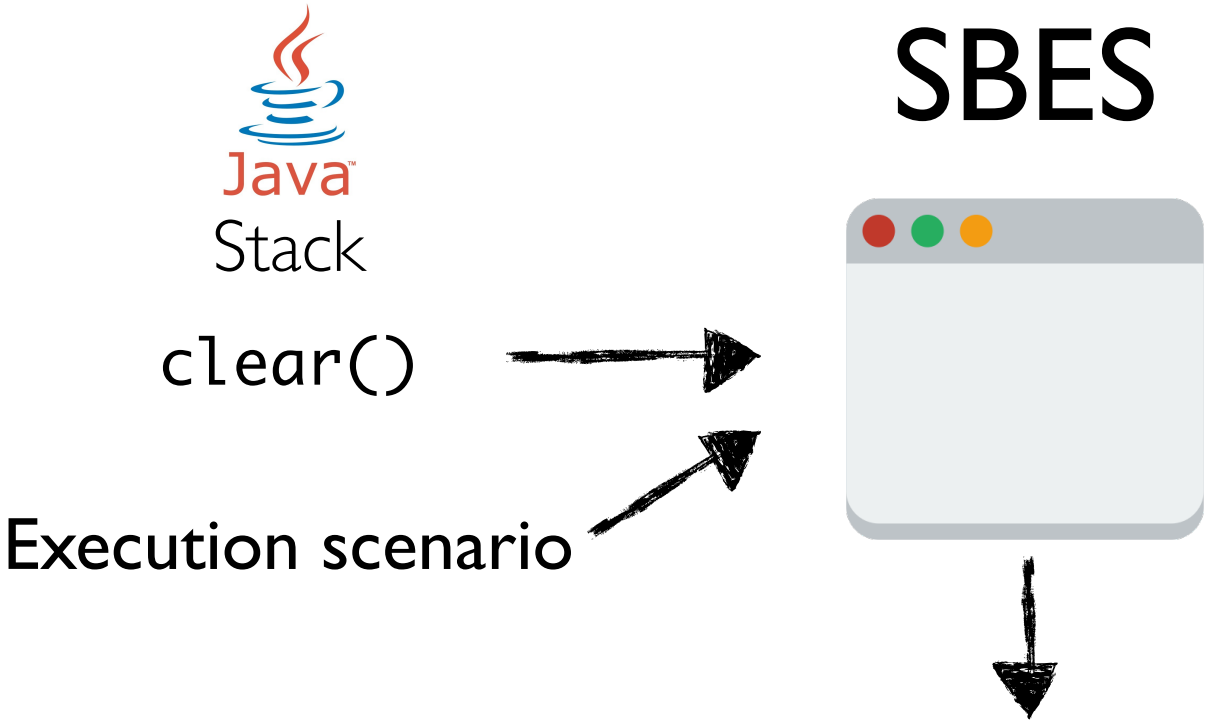
Search-based Synthesis of Equivalences



Search-based Synthesis of Equivalences



Search-based Synthesis of Equivalences

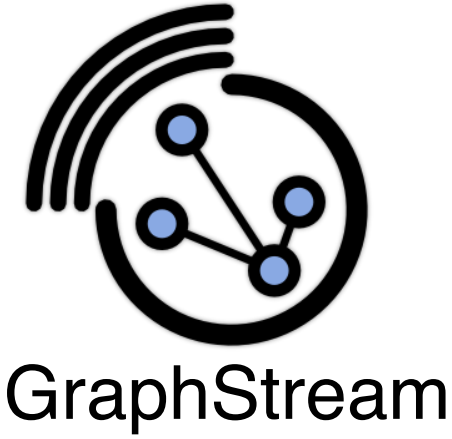


- `removeAllElements()`
- `setSize(0)`
- `Collection c = new Collection();`
  `retainAll(c)`

Evaluation

Evaluation

java.util.Stack



graphstream.Path

graphstream.Edge

graphstream.Node

graphstream.MultiNode

graphstream.Vector2

graphstream.Vector3

Evaluation

java.util.Stack

```
pop()          remove(size() - 1)
```

Evaluation

java.util.Stack

```
pop()          remove(size() - 1)

               push(0);pop();
               remove(size() - 1)
```

Evaluation

java.util.Stack

```
pop()          remove(size() - 1)

               push(0);pop();
               remove(size() - 1)

               push(0);pop(); push(0);pop();
               remove(size() - 1)

               …
```

Evaluation

java.util.Stack

`pop()`

```
remove(size() - 1)
```

```
push(0);pop();
remove(size() - 1)

push(0);pop(); push(0);pop();
remove(size() - 1)

…
```

Evaluation

	# methods	# eqs
java.util.Stack	15	45
graphstream.Path	2	5
graphstream.Edge	9	20
graphstream.Node	5	12
graphstream.MultiNode	5	12
graphstream.Vector2	5	21
graphstream.Vector3	6	22
	47	**137**

Evaluation

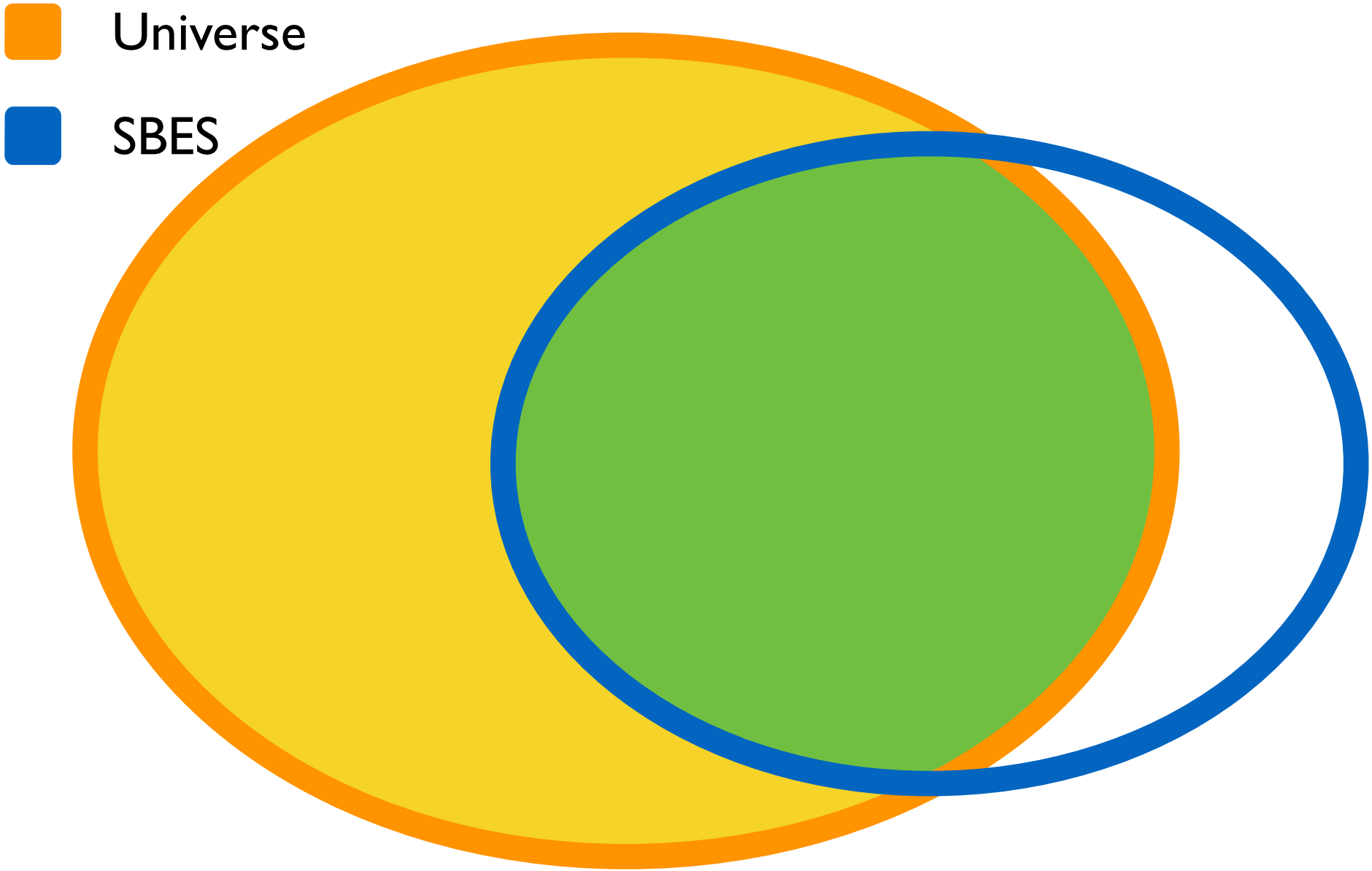
How **effective** is the search-based synthesis?

How **efficient** is the search-based synthesis?

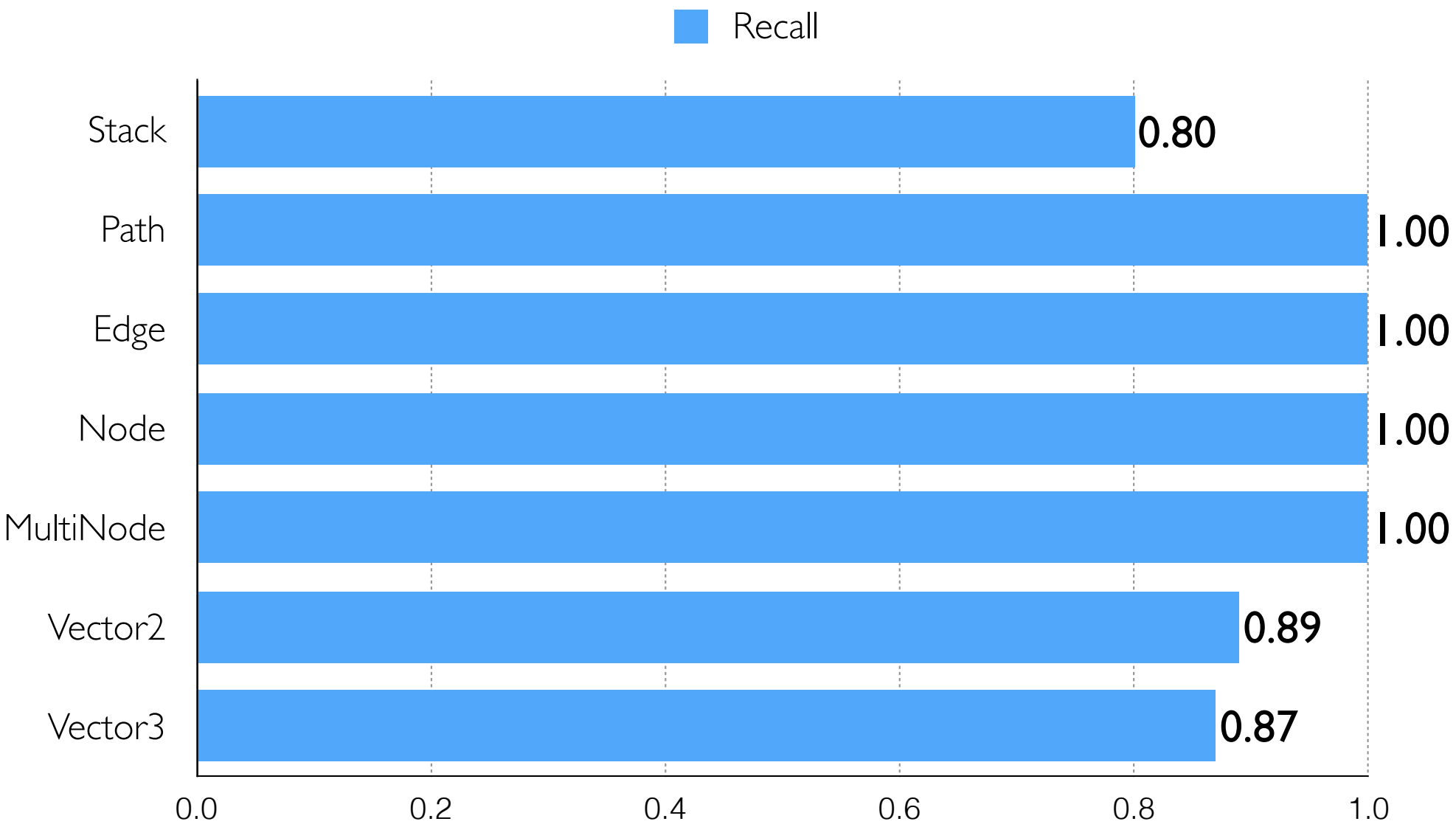
Evaluation

How **effective** is the search-based synthesis?

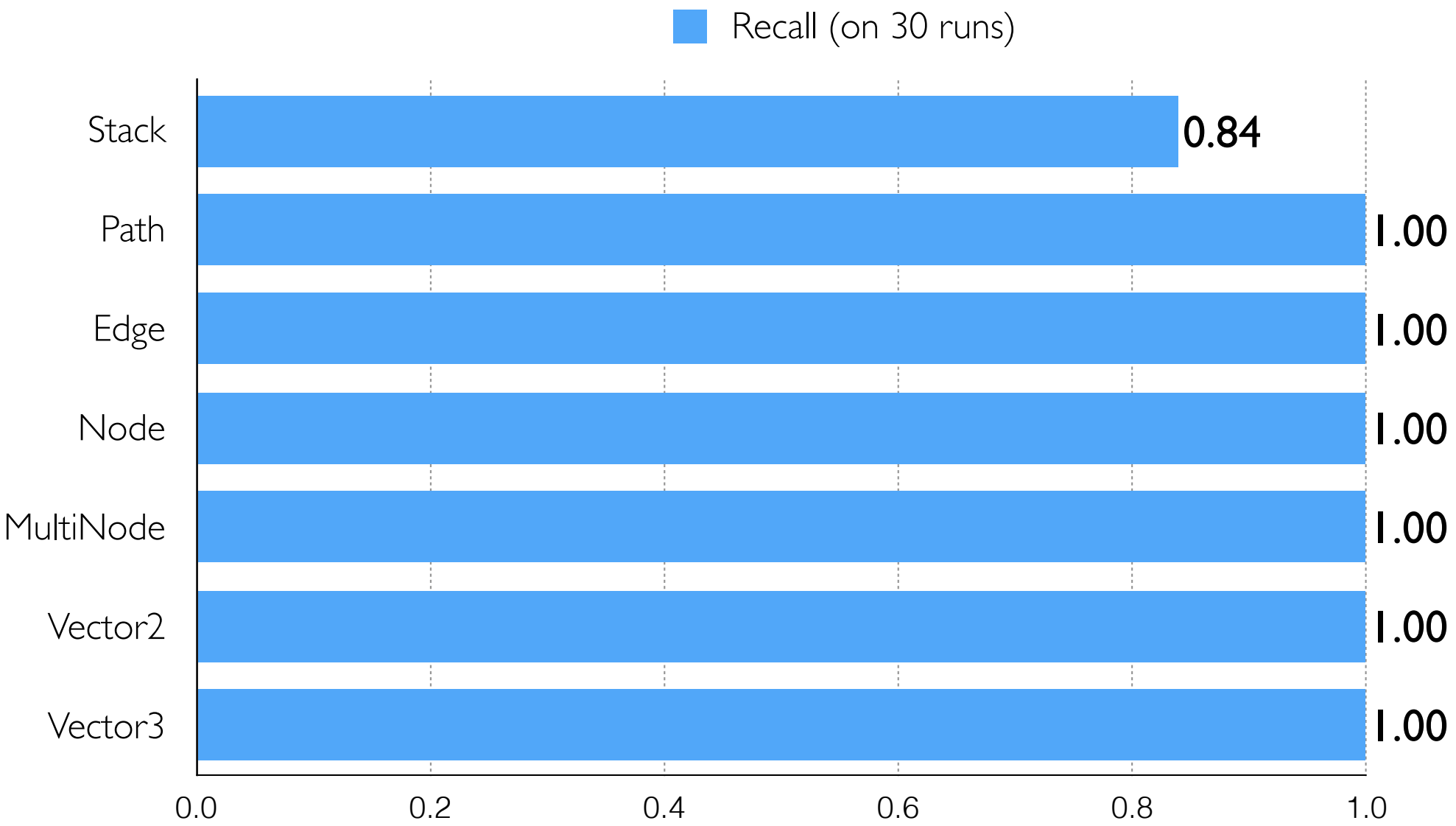
Recall



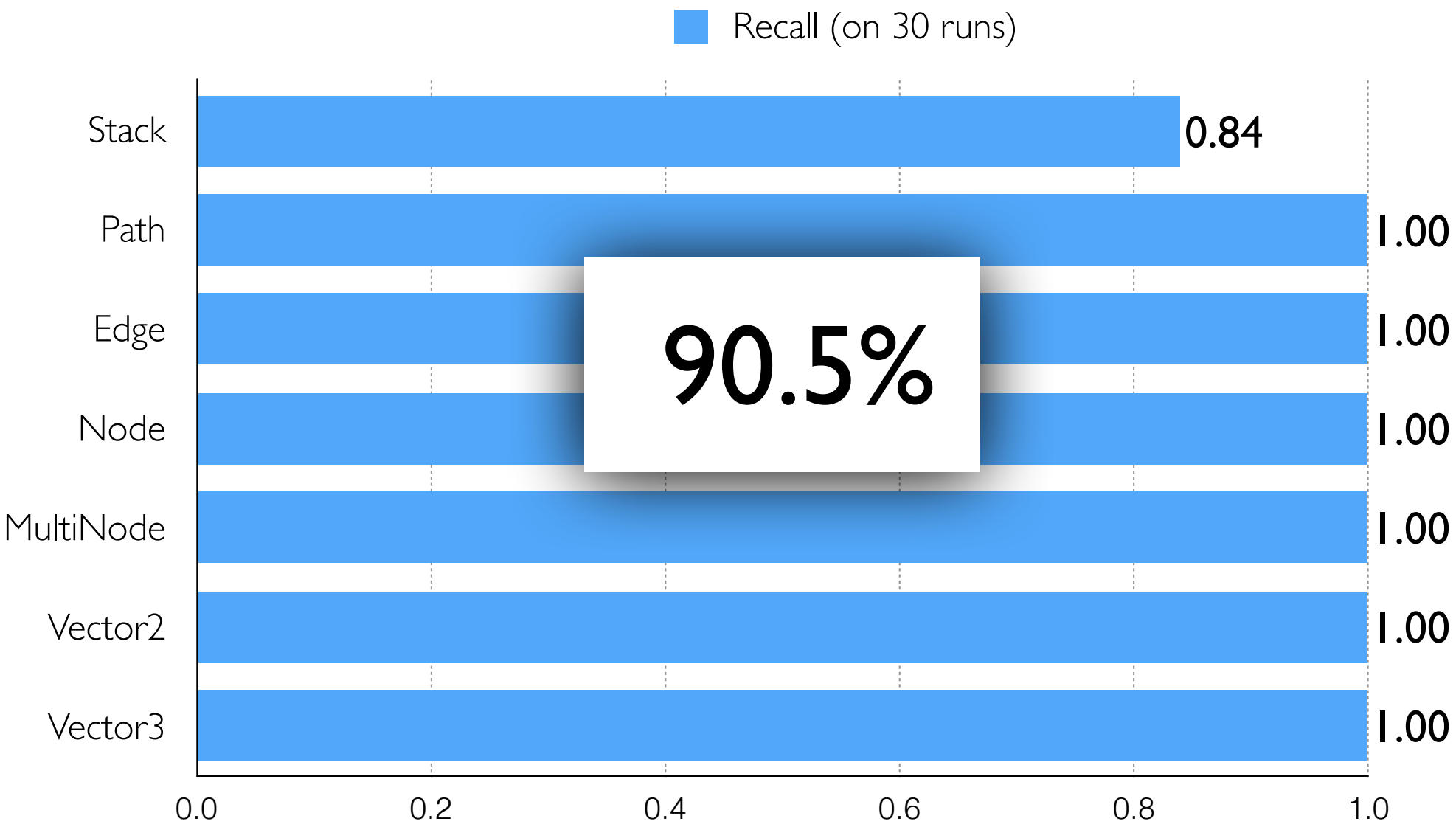
Effectiveness of Search-based Synthesis



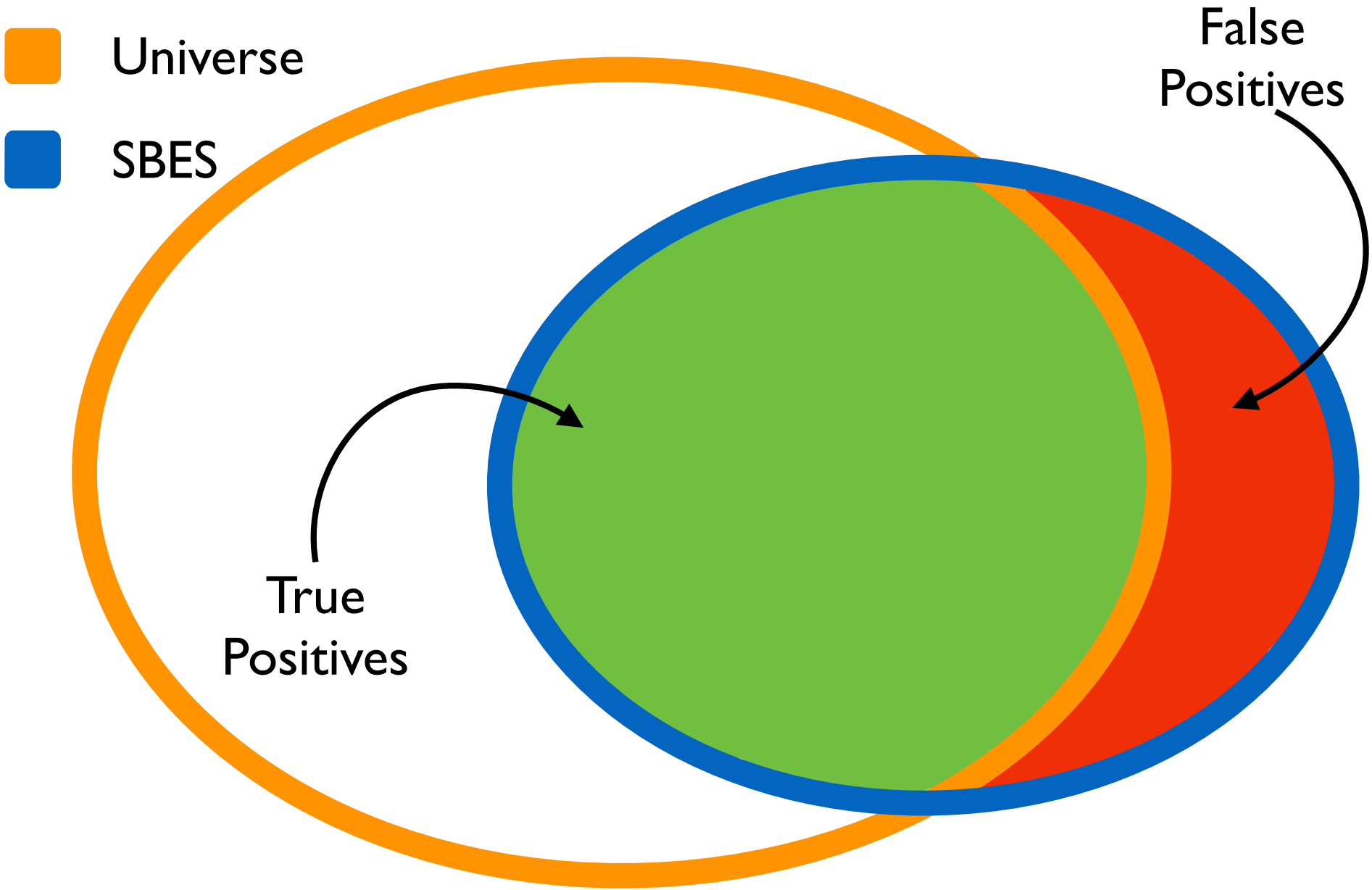
Effectiveness of Search-based Synthesis



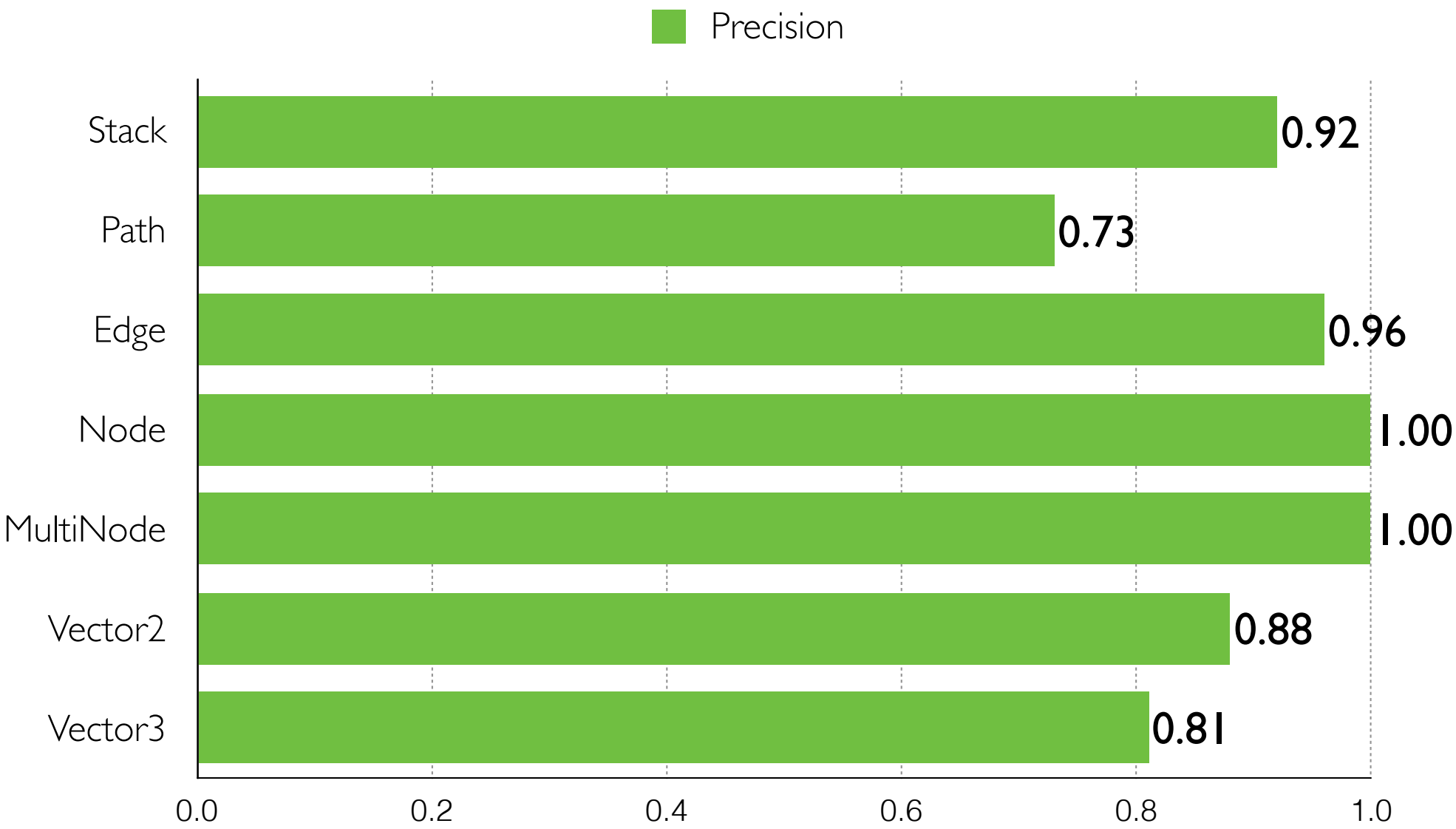
Effectiveness of Search-based Synthesis



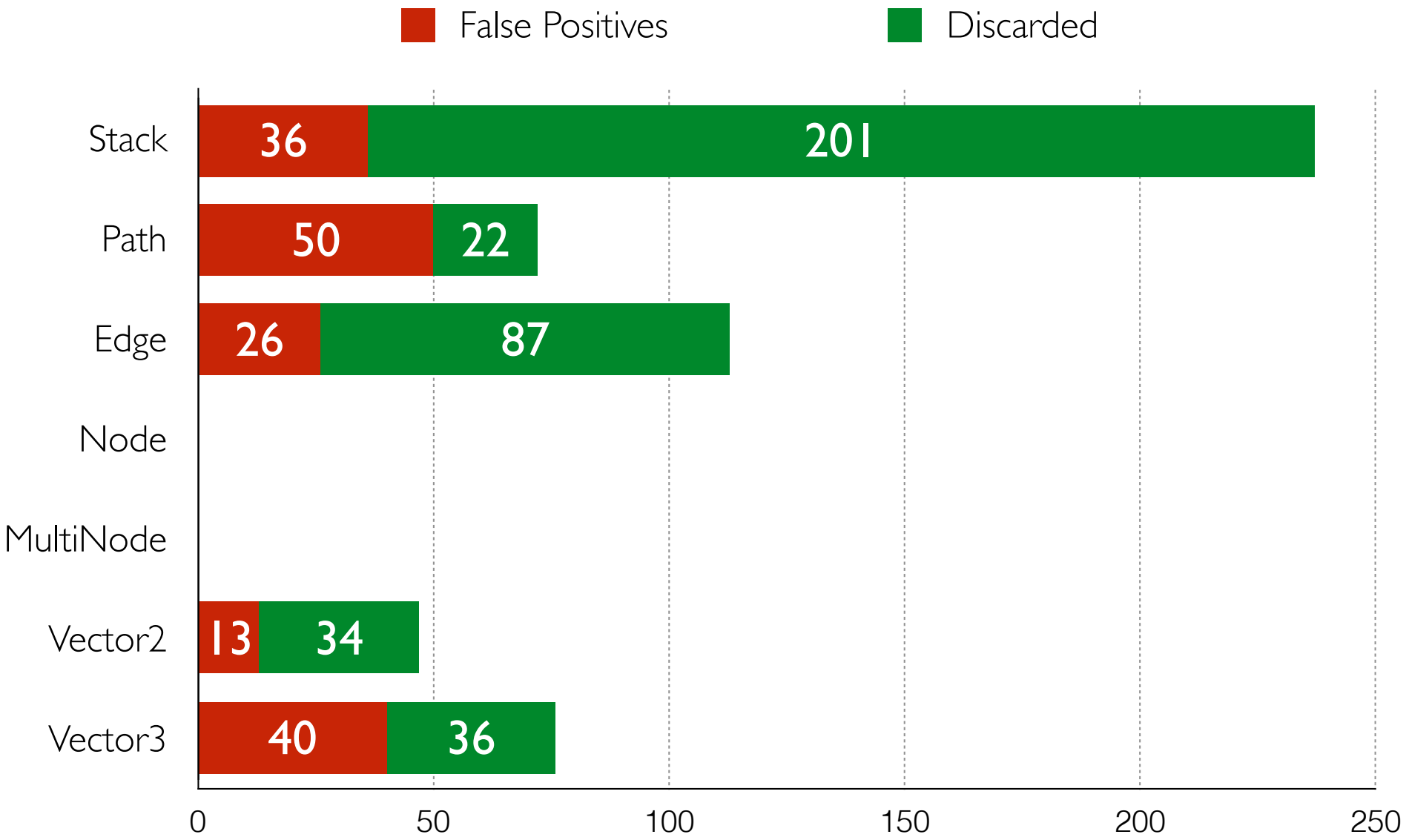
Precision



Effectiveness of Search-based Synthesis



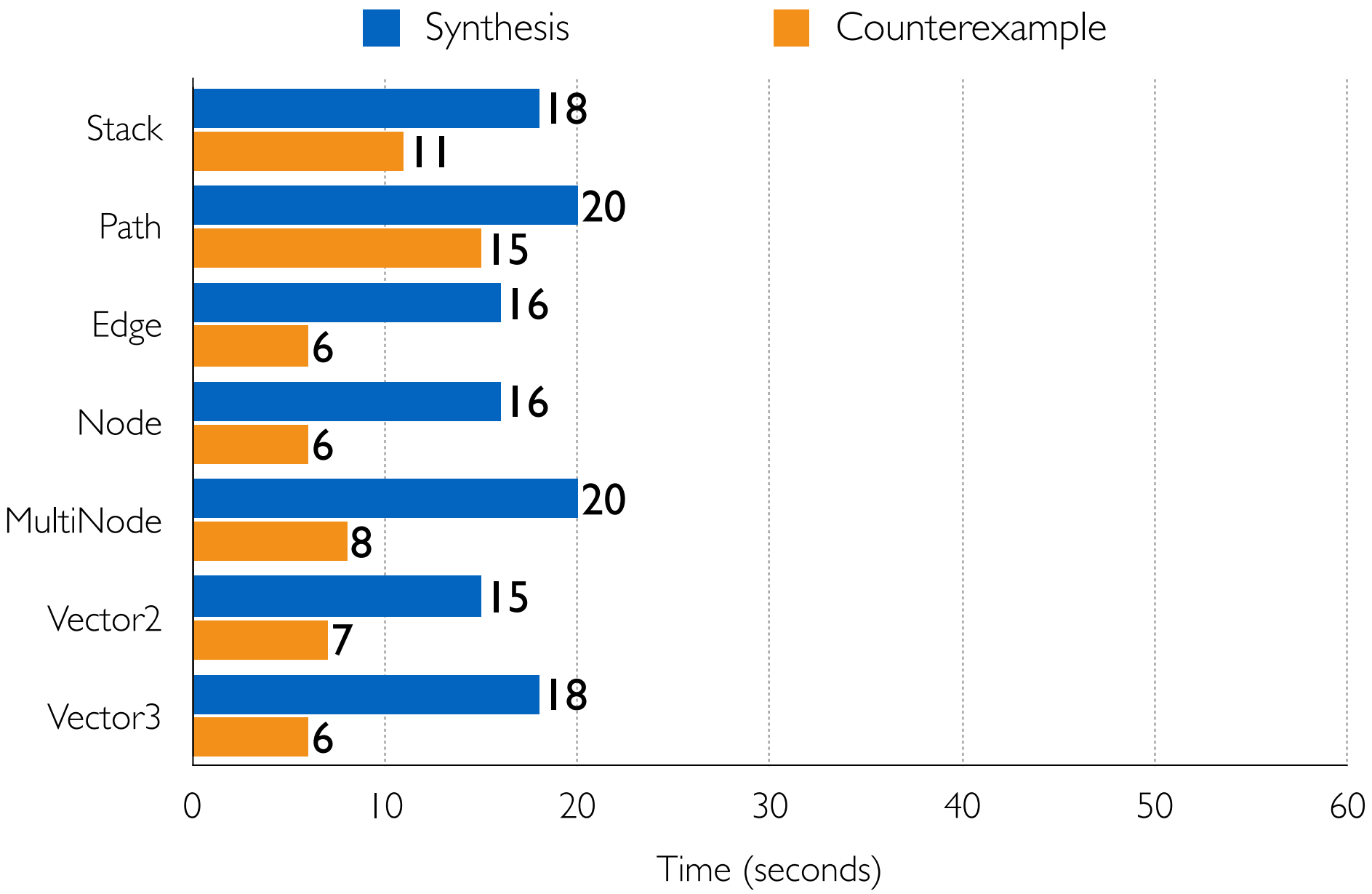
Effectiveness of Counterexamples



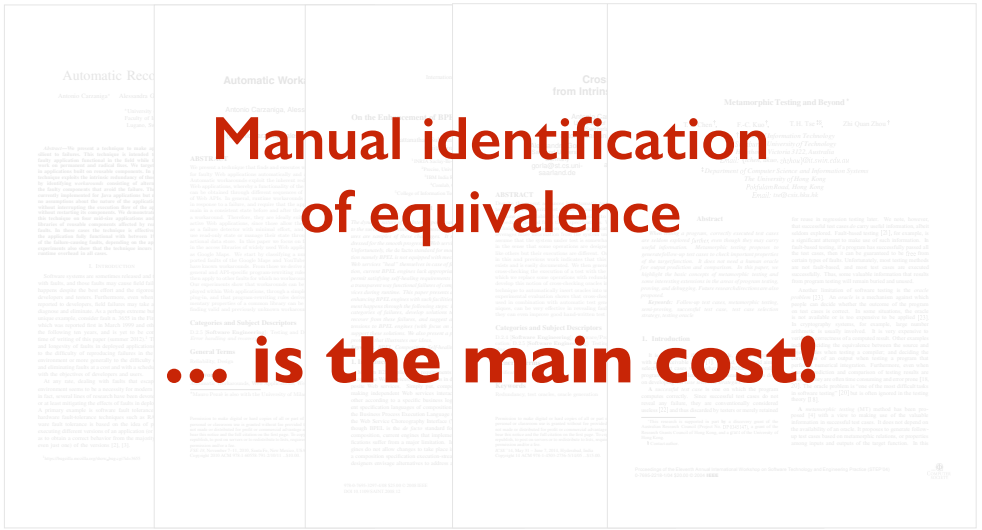
Evaluation

How **efficient** is the search-based synthesis?

Efficiency of Search-based Synthesis



Exploiting Equivalence



Manual identification of equivalence

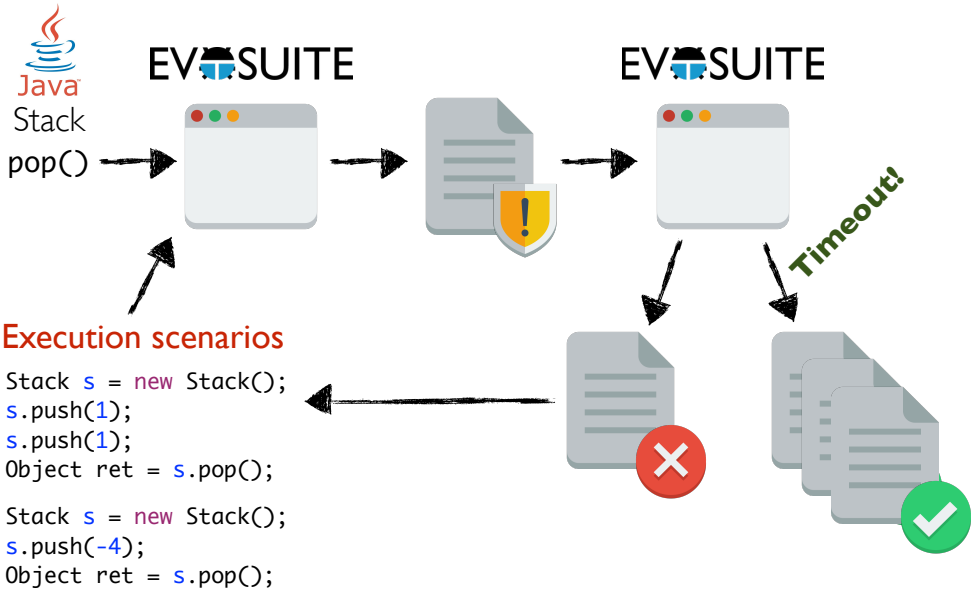
… is the main cost!

Exploiting Equivalence

Manual identification of equivalence

… is the main cost!

Search-based Synthesis of Equivalences

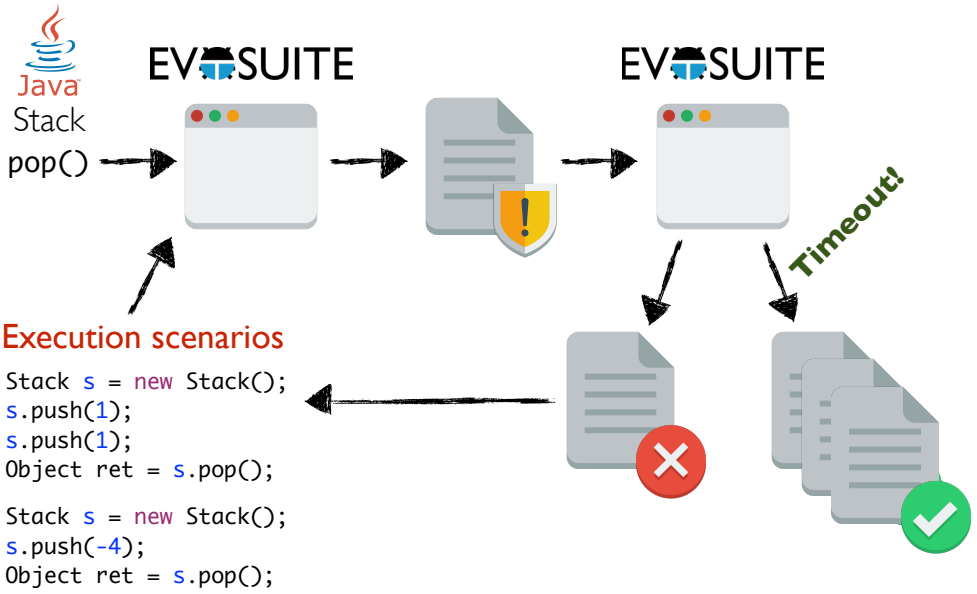


Exploiting Equivalence

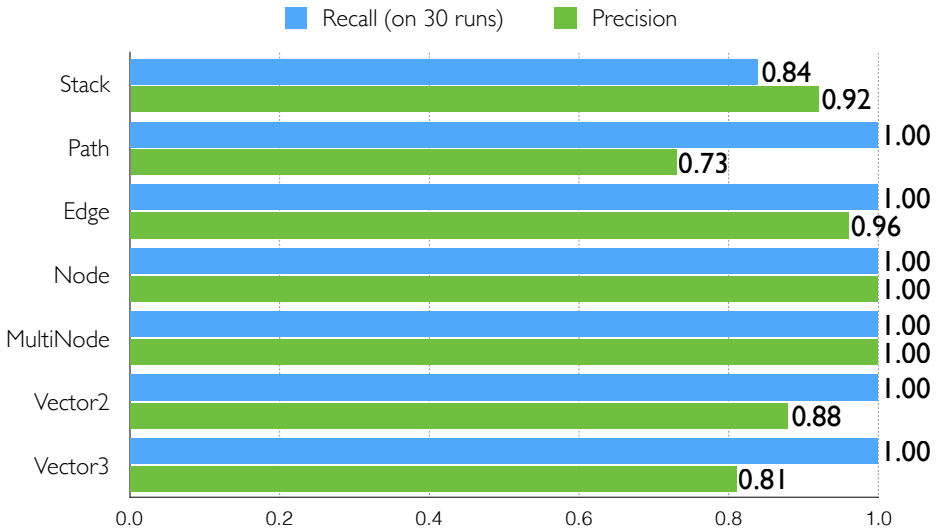
Manual identification of equivalence

... is the main cost!

Search-based Synthesis of Equivalences



Effectiveness of Search-based Synthesis

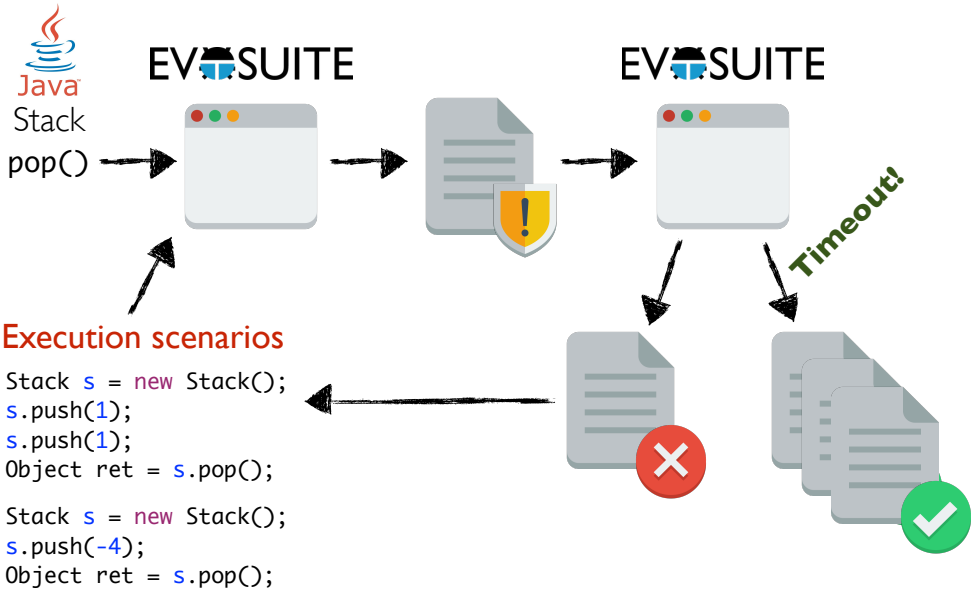


Exploiting Equivalence

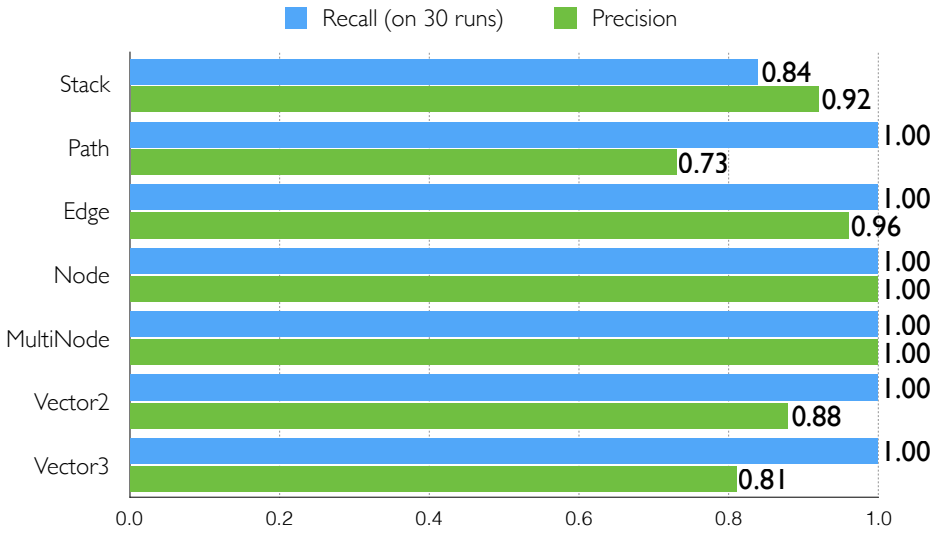
Manual identification of equivalence

... is the main cost!

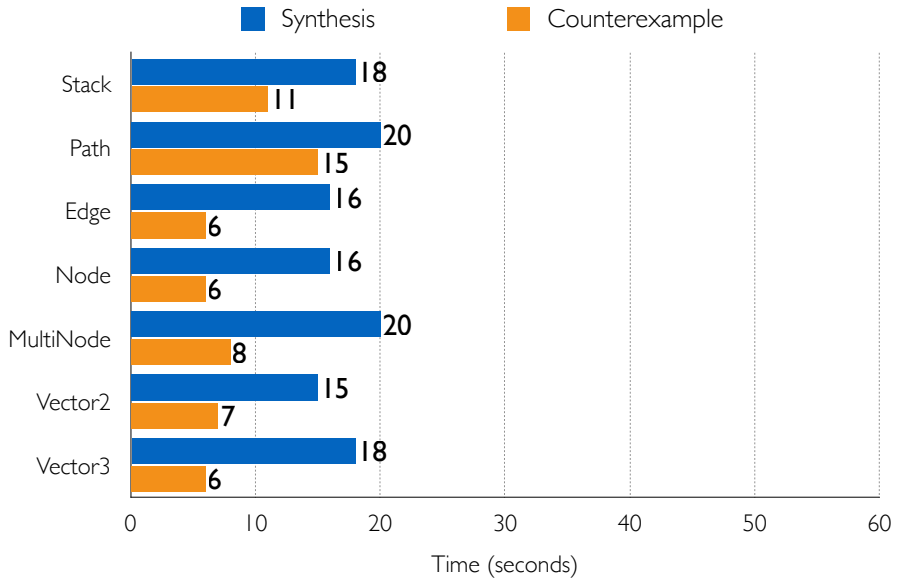
Search-based Synthesis of Equivalences



Effectiveness of Search-based Synthesis



Efficiency of Search-based Synthesis

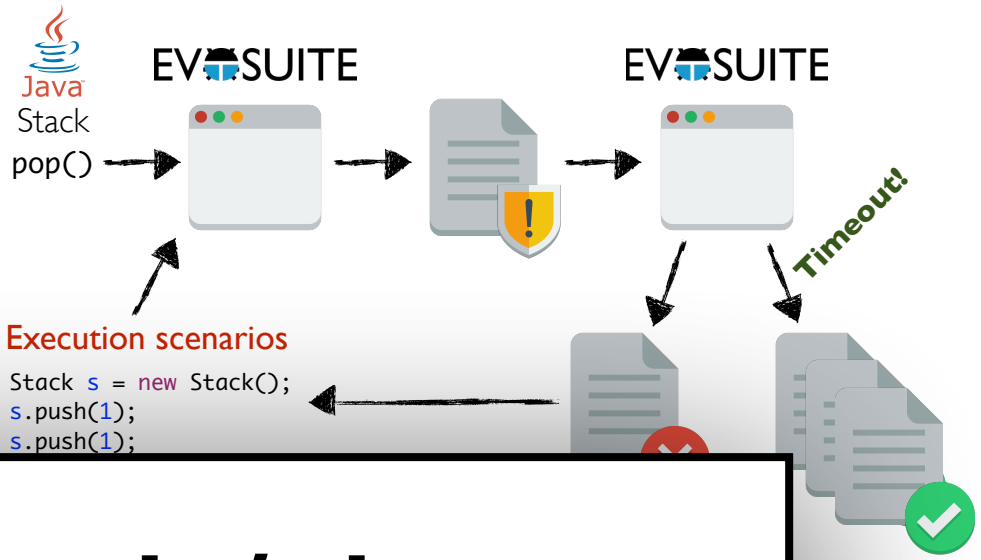


Exploiting Equivalence

Manual identification of equivalence

... is the main cost!

Search-based Synthesis of Equivalences



star.inf.usi.ch/sbes

Effecti … hesis