Improving Patch Correctness Analysis via Random Testing and Large Language Models

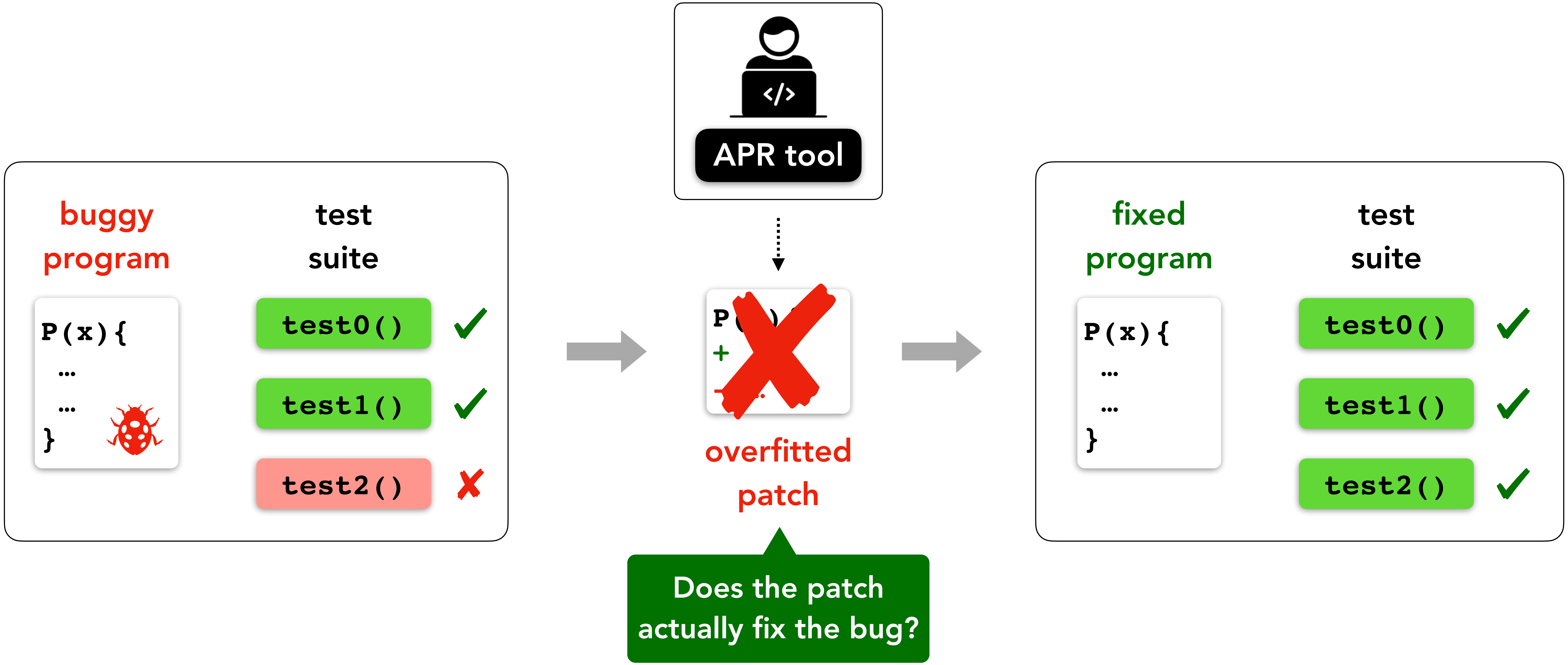
Facundo Molina
IMDEA Software Institute,
Spain

Juan Manuel Copia
IMDEA Software Institute,
Spain

Alessandra Gorla
IMDEA Software Institute,
Spain

ICST 2024

A Program Repair Scenario



buggy program

test suite

P(x){
…
…
}

test0() ✓

test1() ✓

test2() ✗

APR tool

P(x){
+
-…

overfitted patch

Does the patch actually fix the bug?

fixed program

test suite

P(x){
…
…
}

test0() ✓

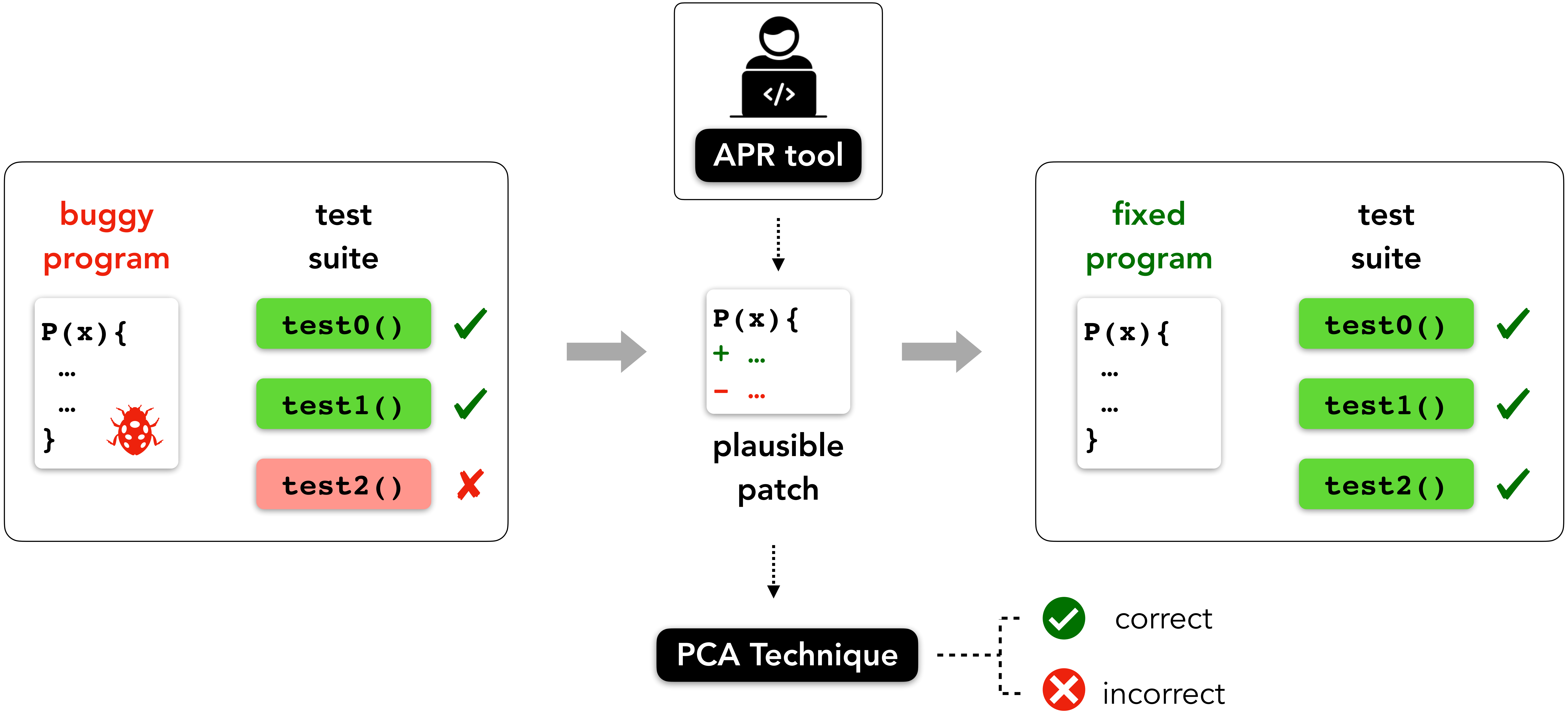
test1() ✓

test2() ✓

Plausible patches are prone to overfitting, leading to the creation of incorrect patches

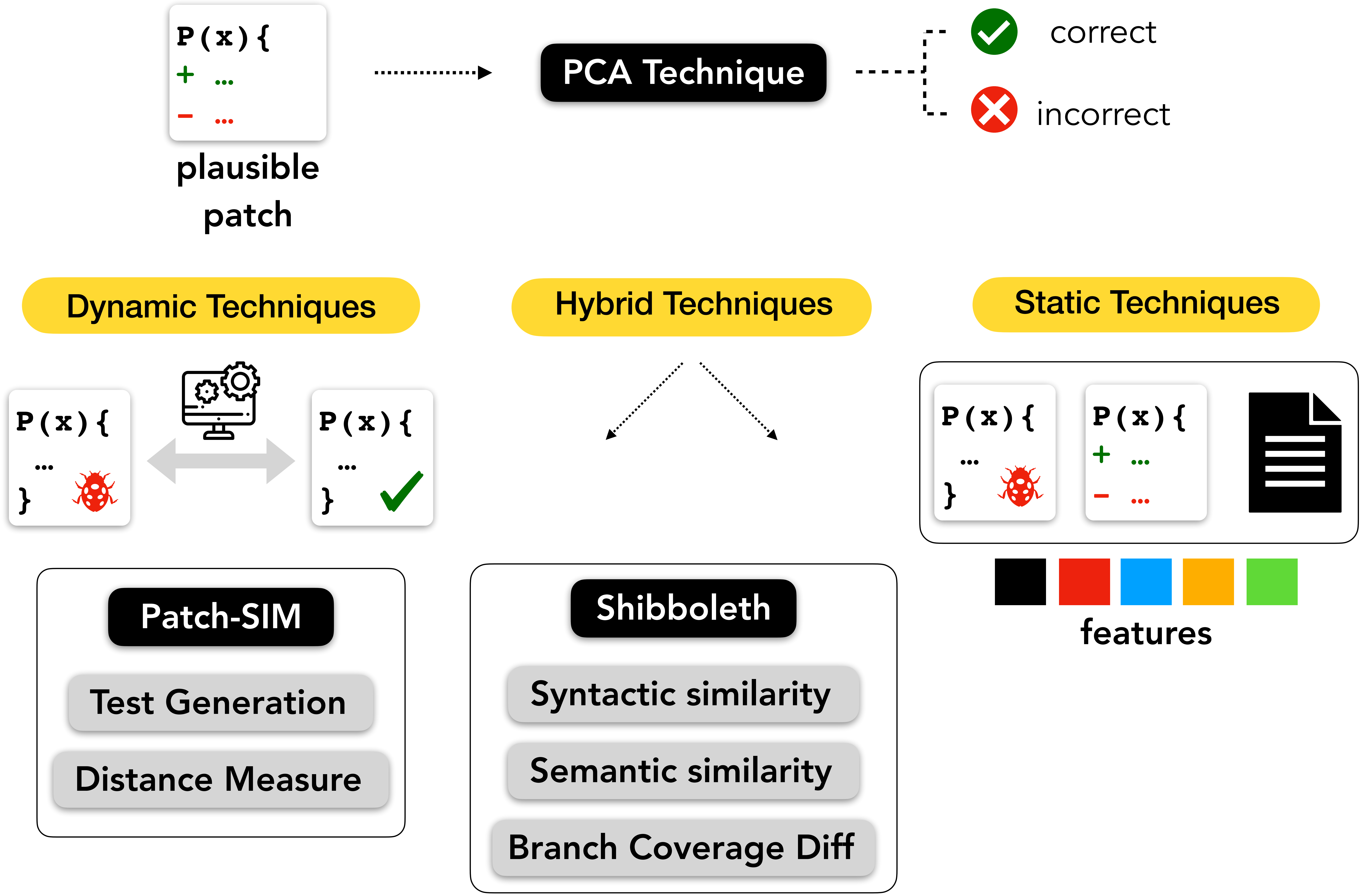
Claire Le Goues, Michael Pardal, and Abhik Roychoudhury. **Automated Program Repair.** Commun. ACM 2019

Patch Correctness Assessment



Shangwen Wang et al. **Automated Patch Correctness Assessment: How far are we?.** ASE 2020

Patch Correctness Assessment



plausible patch

PCA Technique

correct

incorrect

Dynamic Techniques

Hybrid Techniques

Static Techniques

features

Patch-SIM

Test Generation

Distance Measure

Shibboleth

Syntactic similarity

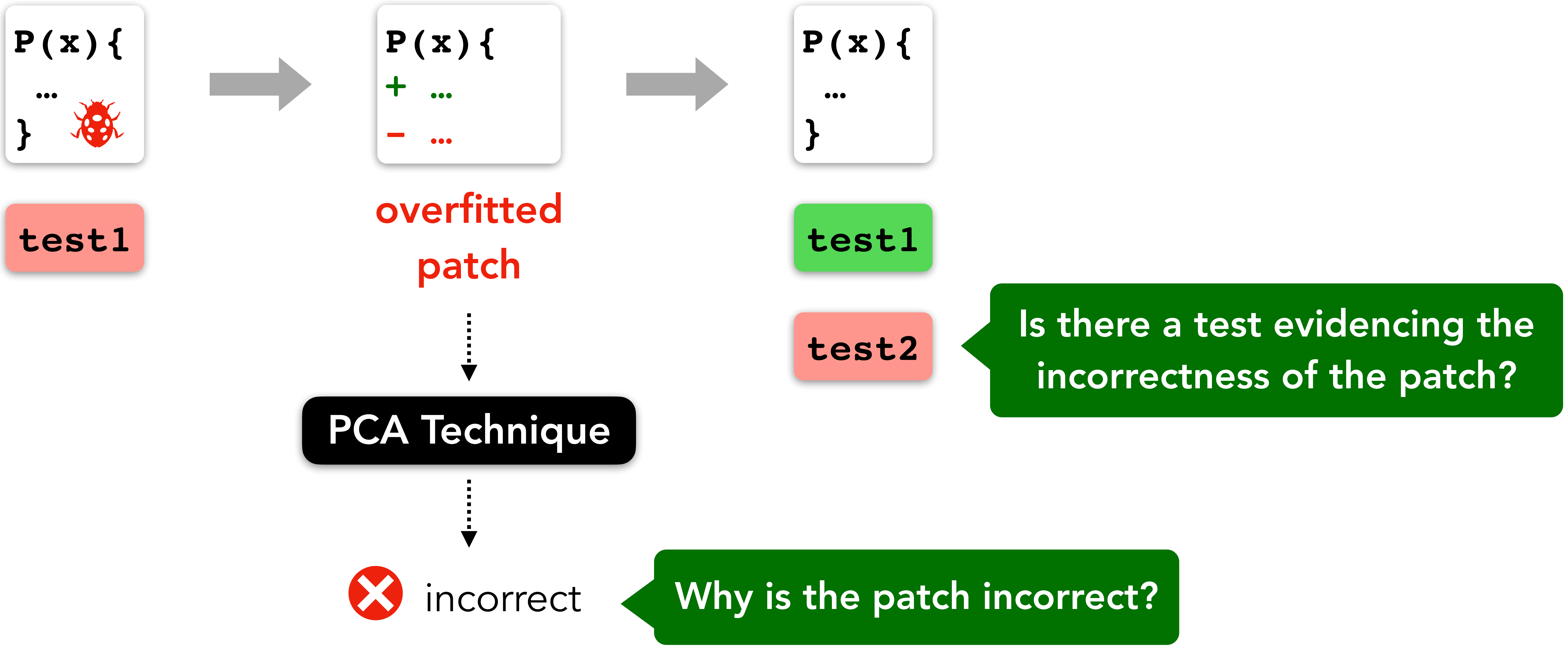
Semantic similarity

Branch Coverage Diff

Yingfei Xiong et al. **Identifying patch correctness in test-based program repair.** ICSE 2018

Ali Ghanbari and Andrian Marcus. **Patch correctness assessment in automated program repair based on the impact of patches on production and test code.** ISSTA 2022

Patch Correctness Assessment


P(x){
…
}
test1
P(x){
+ …
- …
overfitted patch
PCA Technique
incorrect
P(x){
…
}
test1
test2
Is there a test evidencing the incorrectness of the patch?
Why is the patch incorrect?

Goal: improve patch correctness analysis by generating tests highlighting and explaining the incorrectness of a patch

Shangwen Wang et al. **Automated Patch Correctness Assessment: How far are we?.** ASE 2020

Hypothesis

Jackson-databind example

```
P(x){
+ …
- …
```

plausible patch

We found that 70% of subsequent tests only differs from the initial fault-revealing test in the test input and in the corresponding assertions

```
public void testWithScalar118() {
  ObjectMapper mapper = new ObjectMapper();
  ExternalTypeWithNonPOJO input =
        new ExternalTypeWithNonPOJO(new Date(123L));
  String json = mapper.writeValueAsString(input);
  assertNotNull(json);
  // and back just to be sure:
  ExternalTypeWithNonPOJO result = mapper.readValue(json,
        ExternalTypeWithNonPOJO.class);
  assertNotNull(result.value);
  assertTrue(result.value instanceof java.util.Date);
}
```

✓

initial fault-revealing test

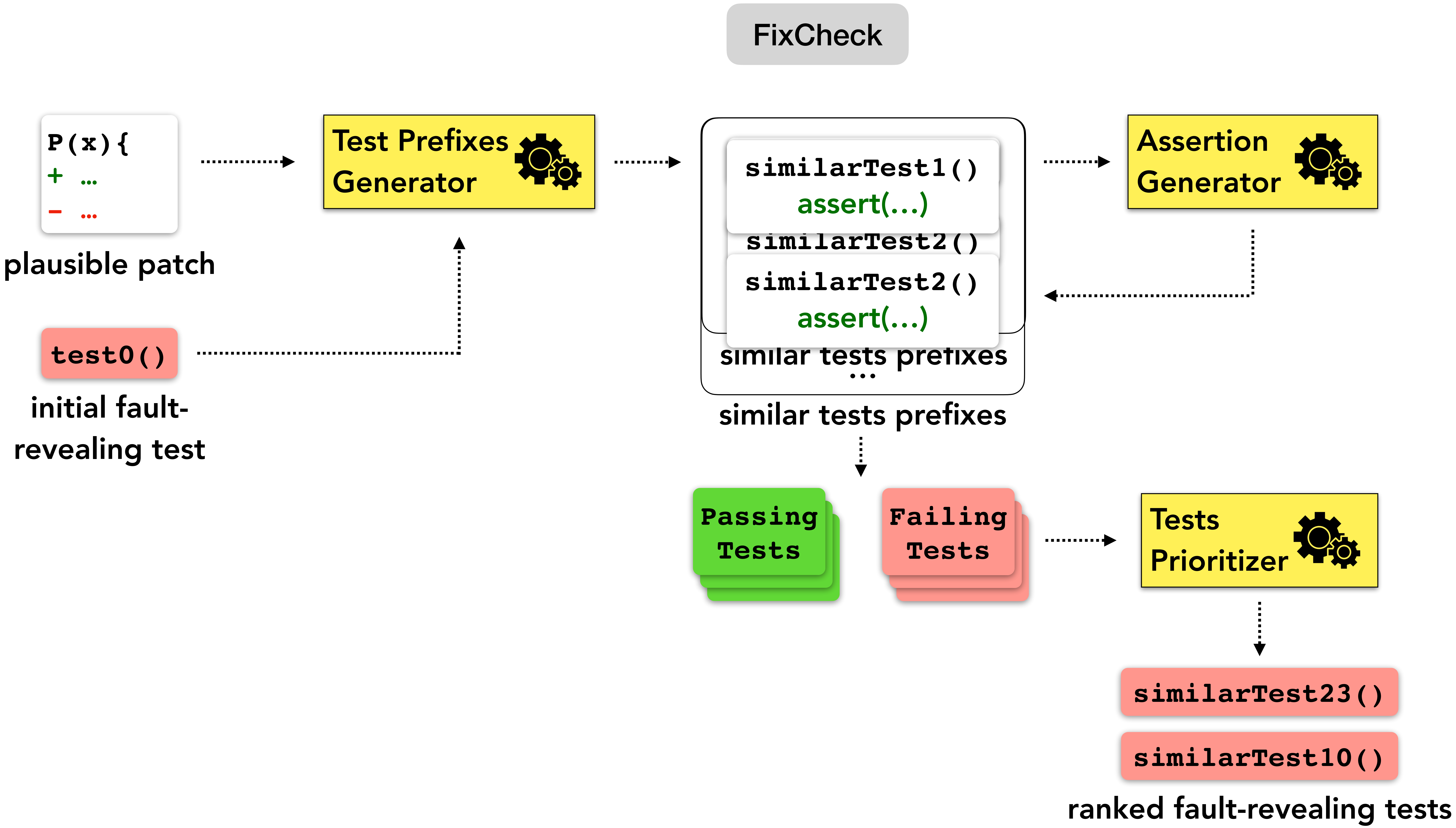
```
public void testWithNaturalScalar118() {
  ObjectMapper mapper = new ObjectMapper();
  ExternalTypeWithNonPOJO input =
        new ExternalTypeWithNonPOJO(Integer.valueOf(13));
  String json = mapper.writeValueAsString(input);
  assertNotNull(json);
  // and back just to be sure:
  ExternalTypeWithNonPOJO result = mapper.readValue(json,
        ExternalTypeWithNonPOJO.class);
  assertNotNull(result.value);
  assertTrue(result.value instanceof Integer);
}
```

✗

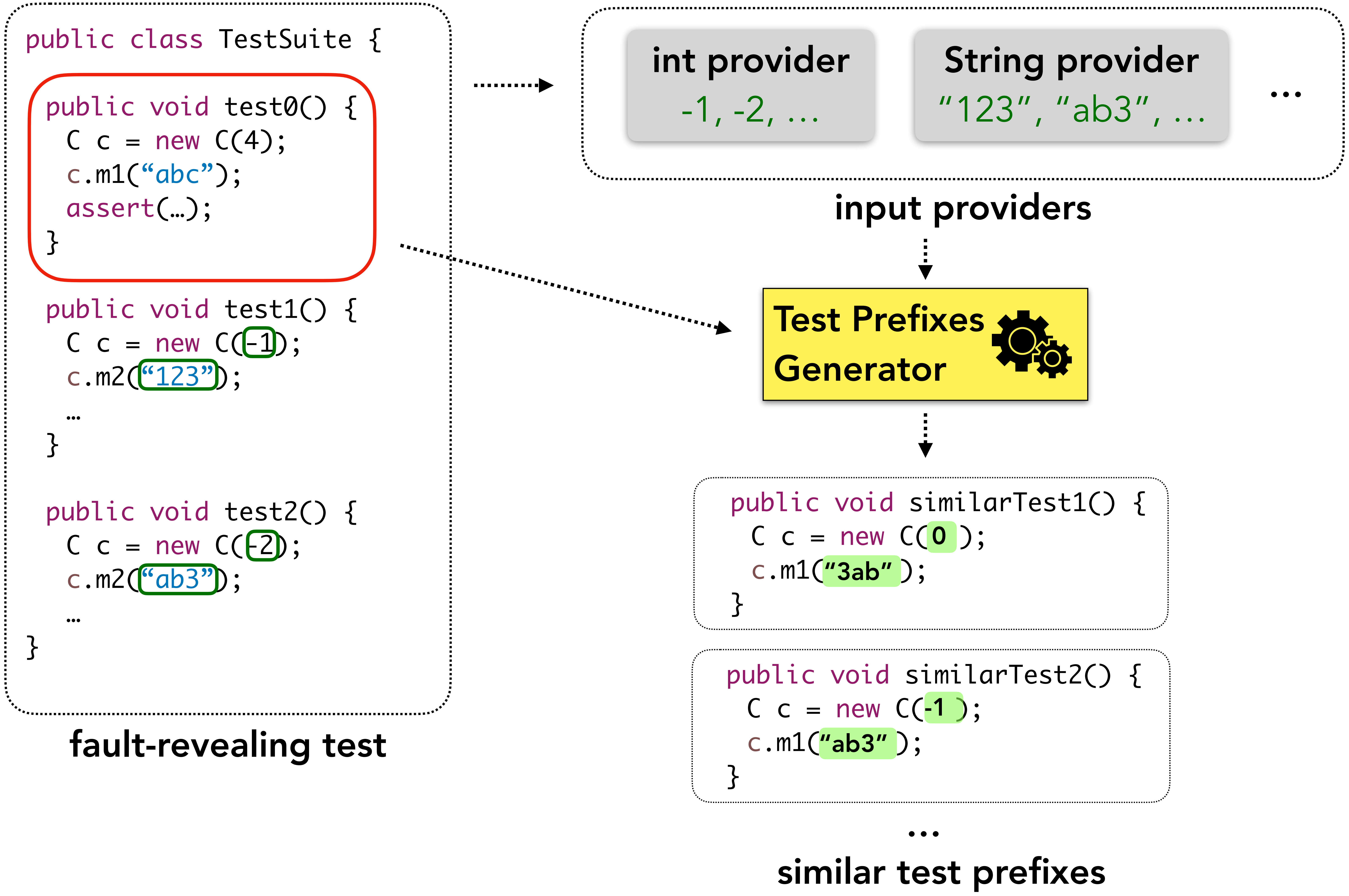
subsequent developer-written test

A test revealing the incorrectness of a patch is similar to the initial fault-revealing test case

Improving Patch Correctness Assessment



Test Prefixes Generation



```
public class TestSuite {

  public void test0() {
    C c = new C(4);
    c.m1(“abc”);
    assert(…);
  }

  public void test1() {
    C c = new C(-1);
    c.m2(“123”);
    …
  }

  public void test2() {
    C c = new C(-2);
    c.m2(“ab3”);
    …
}
```

fault-revealing test

input providers

```
public void similarTest1() {
  C c = new C(0);
  c.m1(”3ab”);
}
```

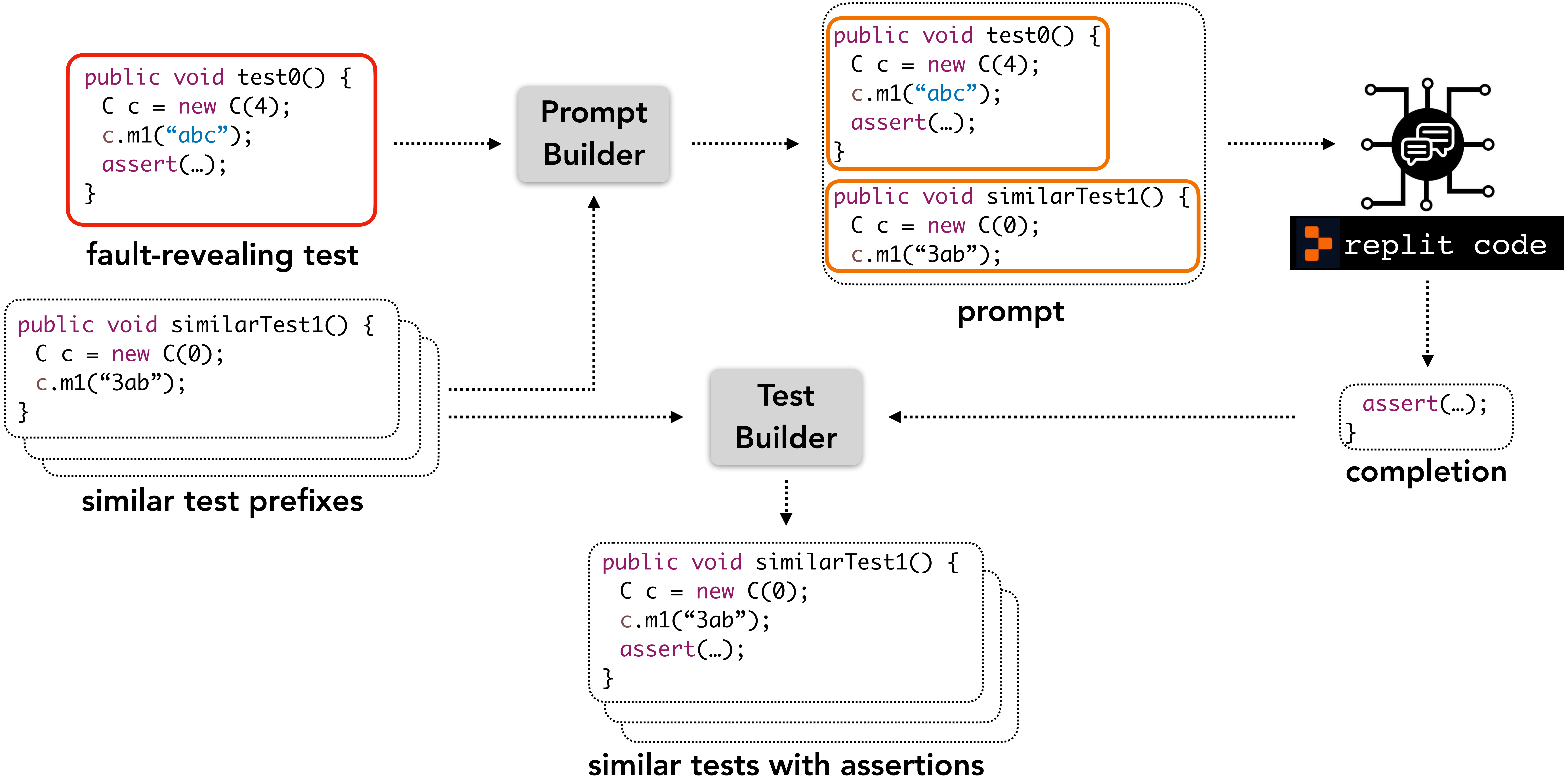
```
public void similarTest2() {
  C c = new C(-1);
  c.m1(”ab3”);
}
```

…

similar test prefixes

Generate tests prefixes that are *similar* to the initial fault-revealing test

Assertion Generation



```
public void test0() {
  C c = new C(4);
  c.m1("abc");
  assert(…);
}
```

```
public void test0() {
  C c = new C(4);
  c.m1("abc");
  assert(…);
}
public void similarTest1() {
  C c = new C(0);
  c.m1("3ab");
```

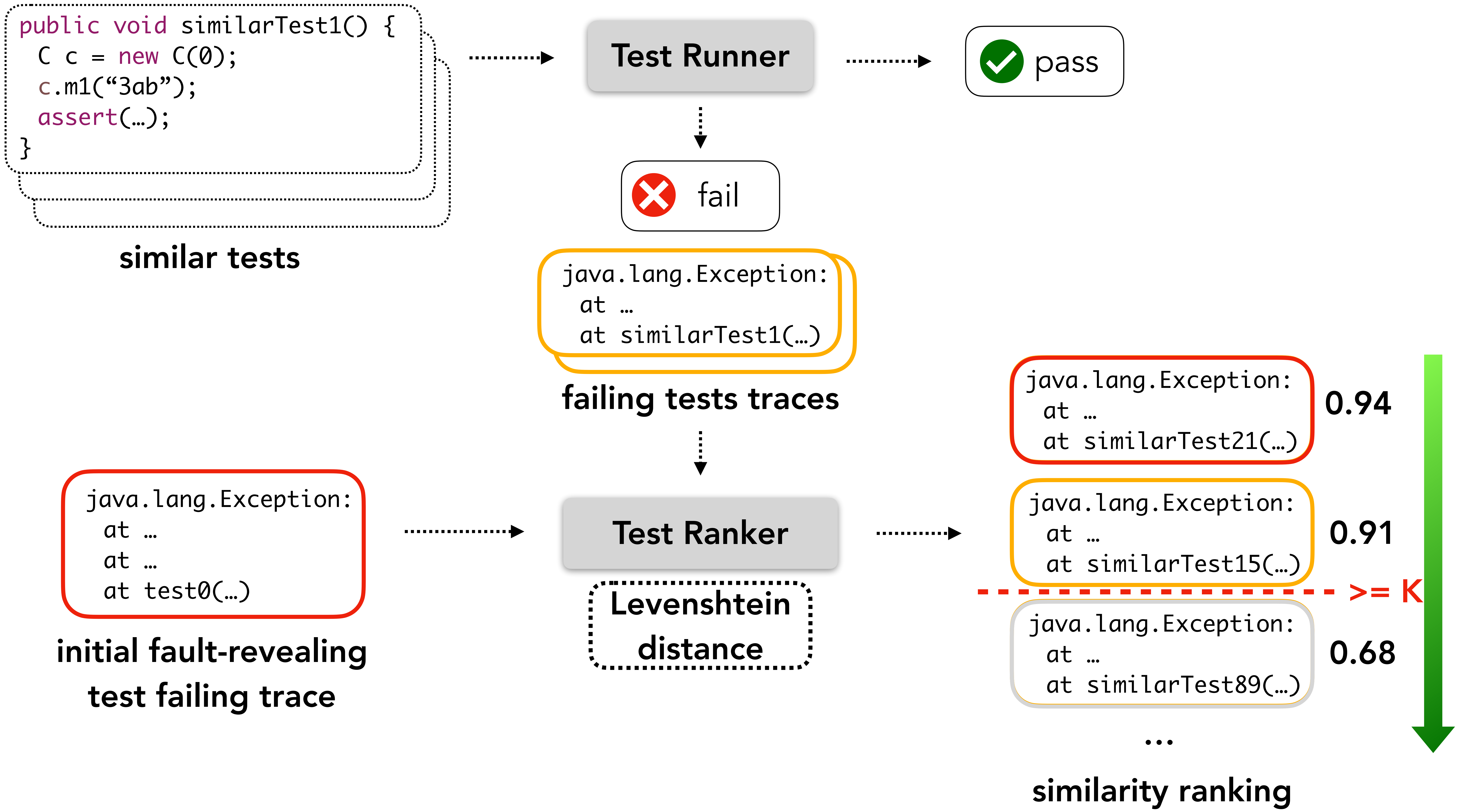
```
  assert(…);
}
```

```
public void similarTest1() {
  C c = new C(0);
  c.m1("3ab");
}
```

```
public void similarTest1() {
  C c = new C(0);
  c.m1("3ab");
  assert(…);
}
```

Equip the generated test prefixes with *meaningful* test assertions

Test Selection and Prioritization



```
public void similarTest1() {
  C c = new C(0);
  c.m1("3ab");
  assert(…);
}
```

similar tests

```
java.lang.Exception:
  at …
  at similarTest1(…)
```

failing tests traces

```
java.lang.Exception:
  at …
  at …
  at test0(…)
```

initial fault-revealing test failing trace

```
java.lang.Exception:
  at …
  at similarTest21(…)
```

0.94

```
java.lang.Exception:
  at …
  at similarTest15(…)
```

0.91

```
java.lang.Exception:
  at …
  at similarTest89(…)
```

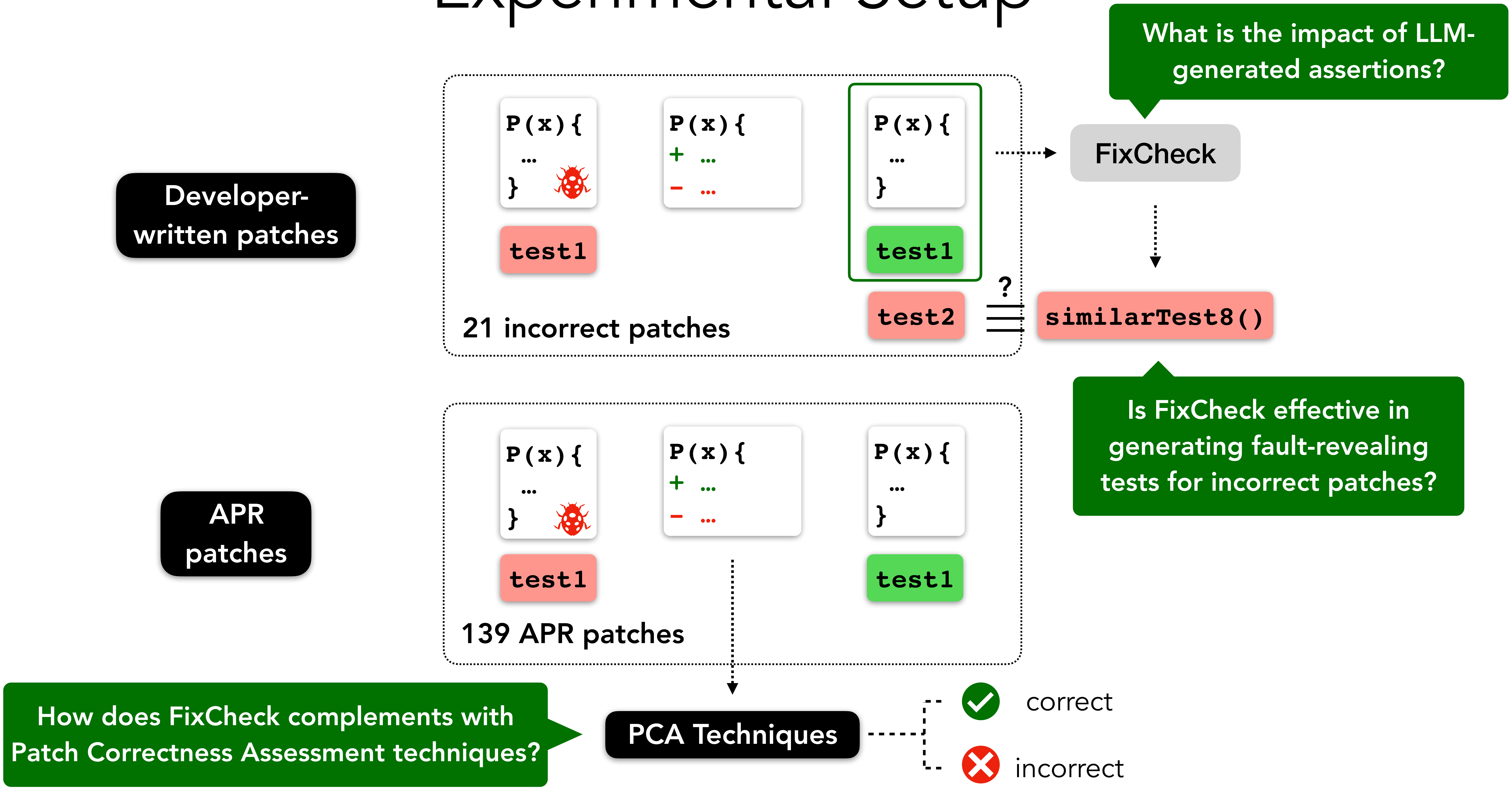
0.68

…

similarity ranking

Select and prioritise the failing tests based on their likelihood of actually revealing a defect in the patch

Experimental Setup



What is the impact of LLM-generated assertions?

Developer-written patches

```
P(x){
  ...
}
```

test1

```
P(x){
+ ...
- ...
```

```
P(x){
  ...
}
```

test1

test2 ≡? similarTest8()

FixCheck

21 incorrect patches

Is FixCheck effective in generating fault-revealing tests for incorrect patches?

APR patches

```
P(x){
  ...
}
```

test1

```
P(x){
+ ...
- ...
```

```
P(x){
  ...
}
```

test1

139 APR patches

How does FixCheck complements with Patch Correctness Assessment techniques?

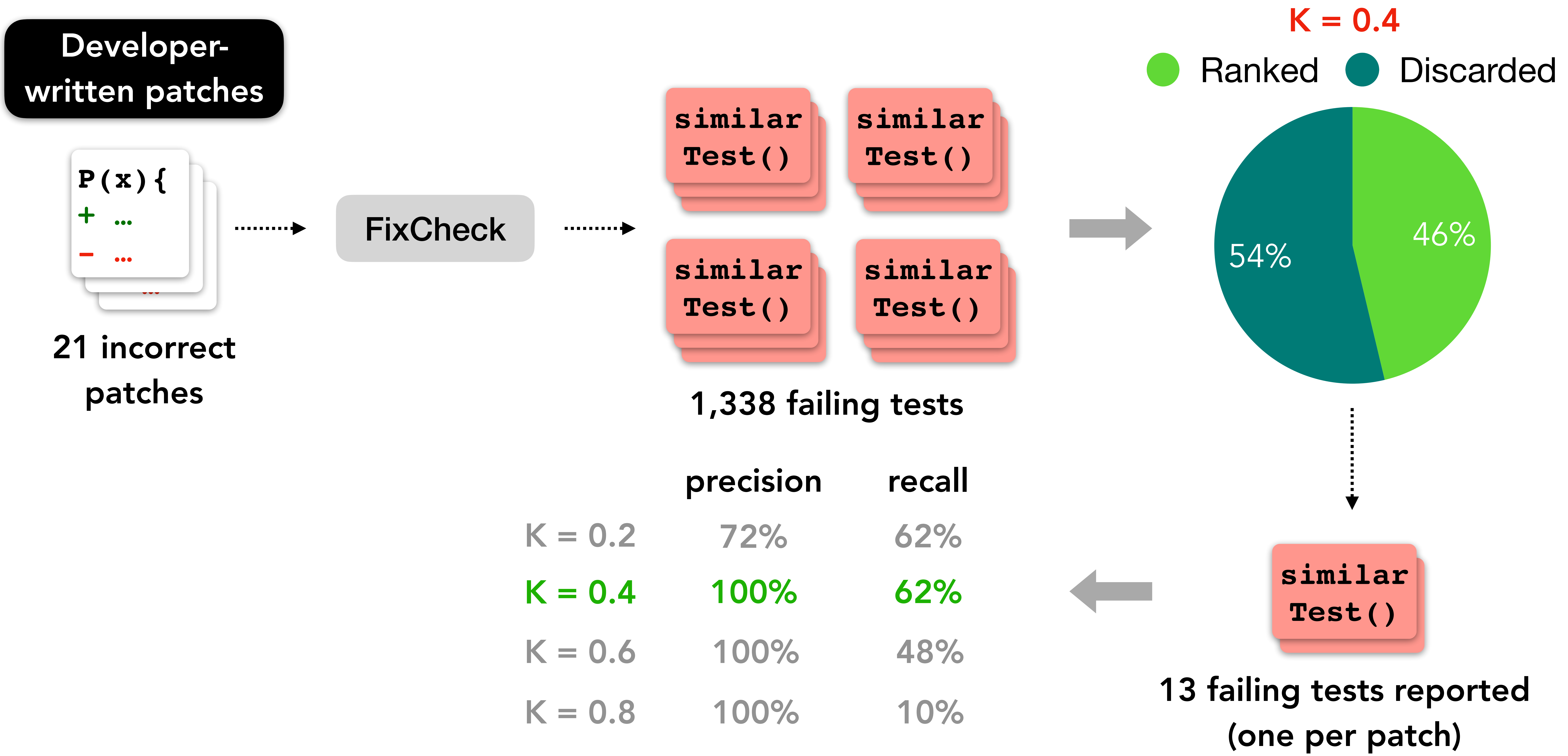
PCA Techniques

✓ correct

✗ incorrect

Yingfei Xiong et al. **Identifying patch correctness in test-based program repair.** ICSE 2018

Effectiveness of FixCheck

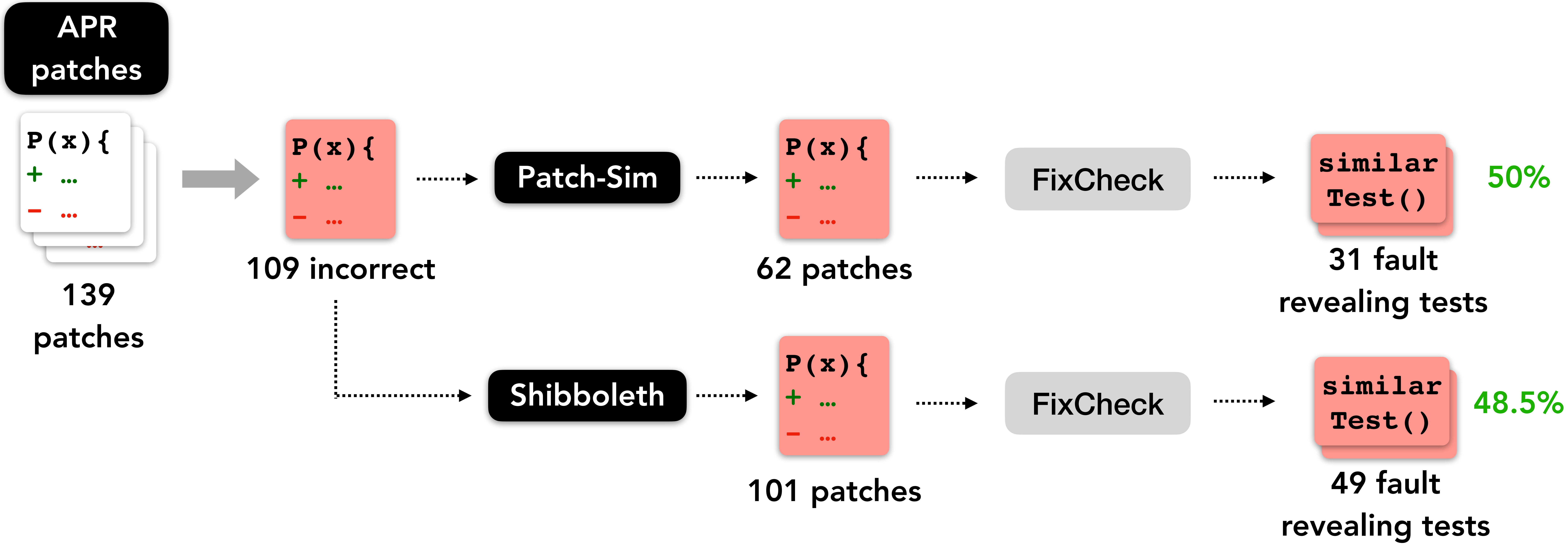


	precision	recall
K = 0.2	72%	62%
K = 0.4	100%	62%
K = 0.6	100%	48%
K = 0.8	100%	10%

FixCheck reports failing tests for up to 62% of incorrect patches

FixCheck reports failing tests for incorrect patches with a precision of 70-100%

Complementing Patch Correctness Assessment Techniques



APR patches

P(x){
+ ...
− ...

139 patches

P(x){
+ ...
− ...

109 incorrect

Patch-Sim

P(x){
+ ...
− ...

62 patches

FixCheck

similar Test()

50%

31 fault revealing tests

Shibboleth

P(x){
+ ...
− ...

101 patches

FixCheck

similar Test()

48.5%

49 fault revealing tests

FixCheck can generate fault-revealing tests for up to 50% of incorrect patches detected by patch correctness assessment tools

Remarks

✦ Automated Patch Correctness Assessment is a relevant problem for APR.

✦ FixCheck is a technique that combines static analysis, random testing, and large language models to effectively generate fault-revealing tests for incorrect patches.

✦ FixCheck can complement patch correctness analyses by providing fault revealing tests.

✦ Fault-revealing test generation for incorrect patches can still be improved:

Test Input Generation **Assertion Generation** **Applicability**