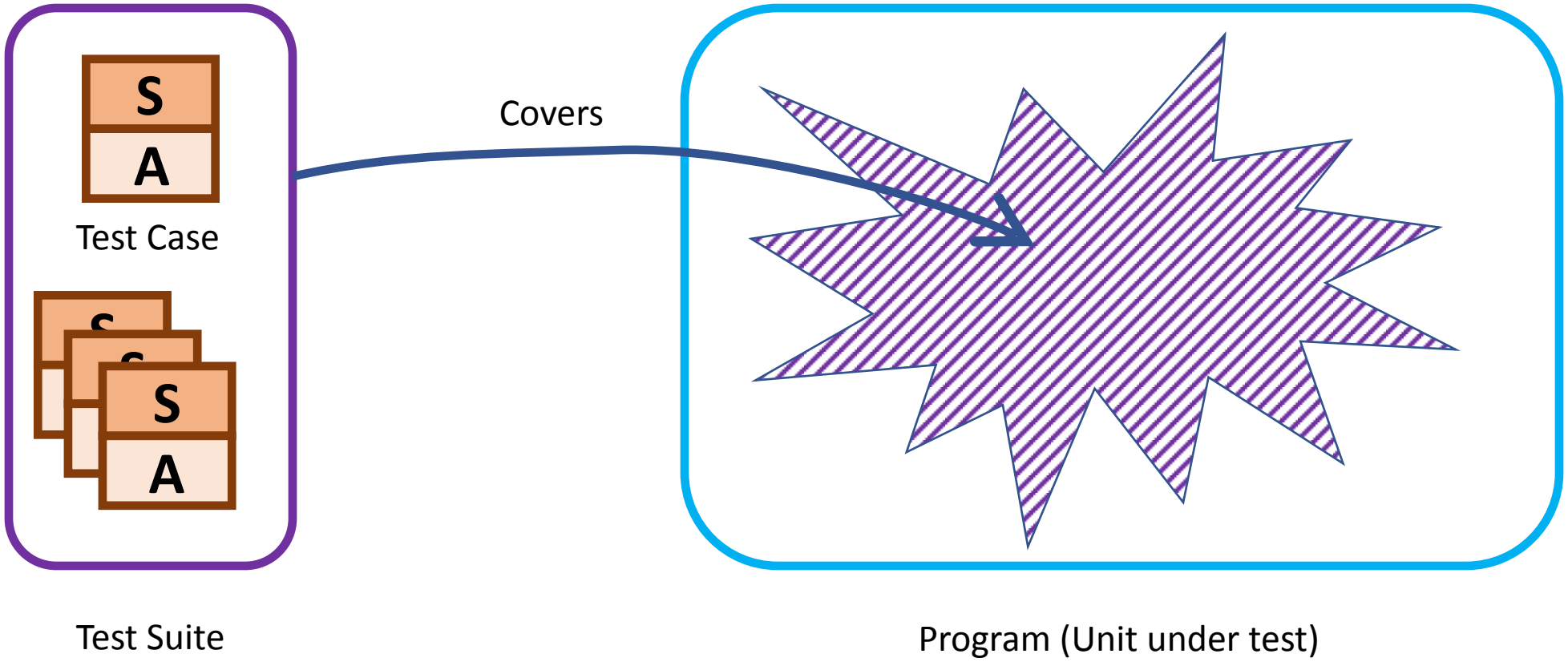
Test Amplification in the Pharo Smalltalk Ecosystem

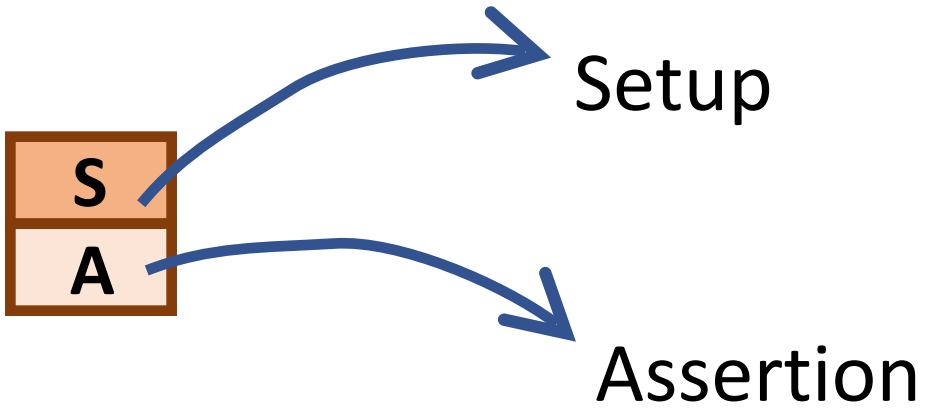
Mehrdad Abdi, Henrique Rocha, Serge Demeyer

Universiteit Antwerpen

Unit Testing

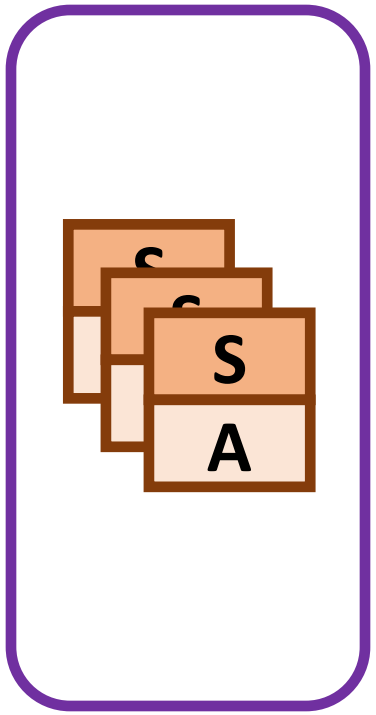


Test Case

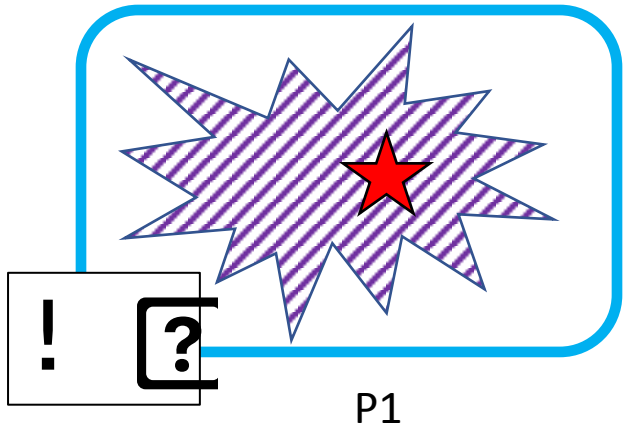



```
b := SmallBank new.
b deposit: 10.
b deposit: 90.

self assert: b balance equals: 100.

b withdraw: 99.

self assert: b balance equals: 0
```

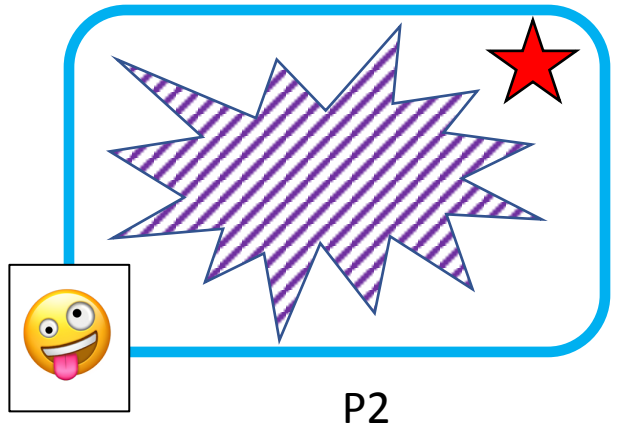
Mutation Testing



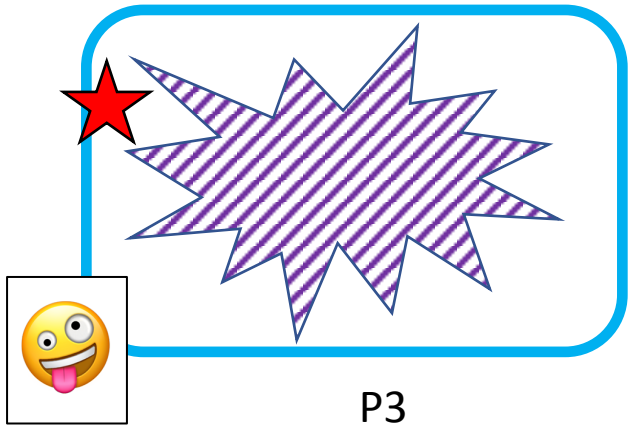
Test Suite



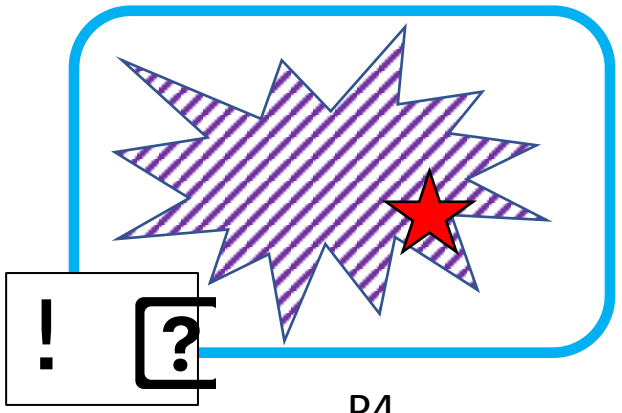
P1



P2

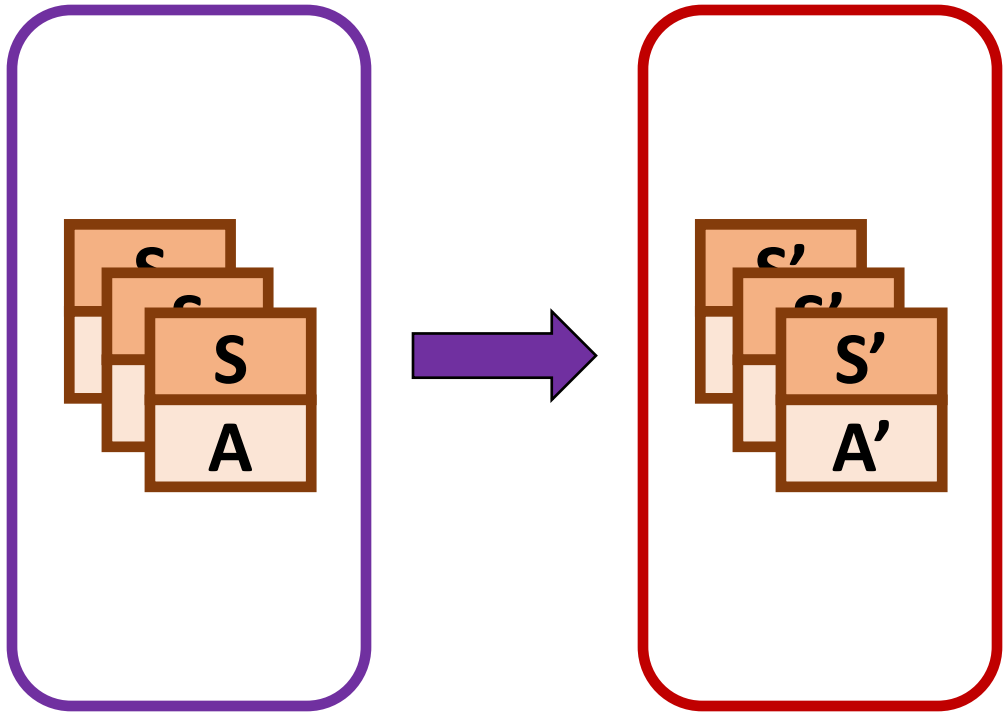


P3



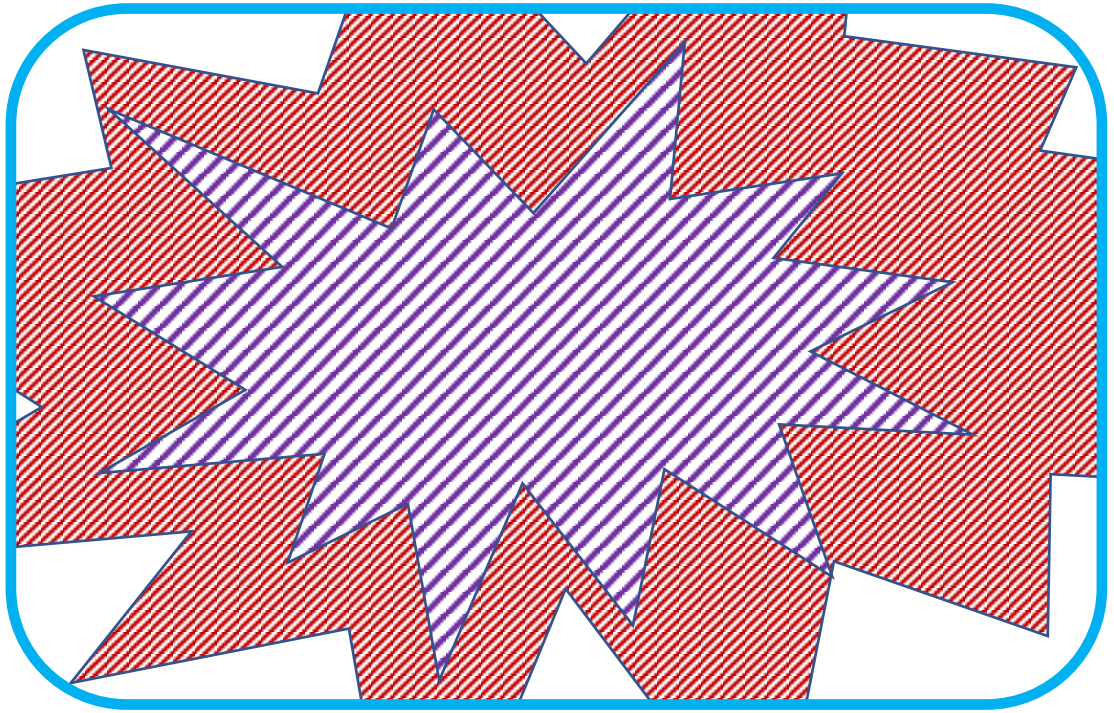
P4

Test Amplification



Test Suite

Amplified Test Suite



Program (Unit under test)

Definition:

Test amplification consists of the automatic transformation of an existing manually written test suite, to enhance a specific, measurable property.

[Danglot 2018] Benjamin Danglot, Oscar Vera-Perez, Zhongxing Yu, Andy Zaidman, Martin Monperrus, and Benoit Baudry. 2018. A Snowballing Literature Study on Test Amplification. arXiv paper 1705.10692v2 (2018).

SMALL-AMP

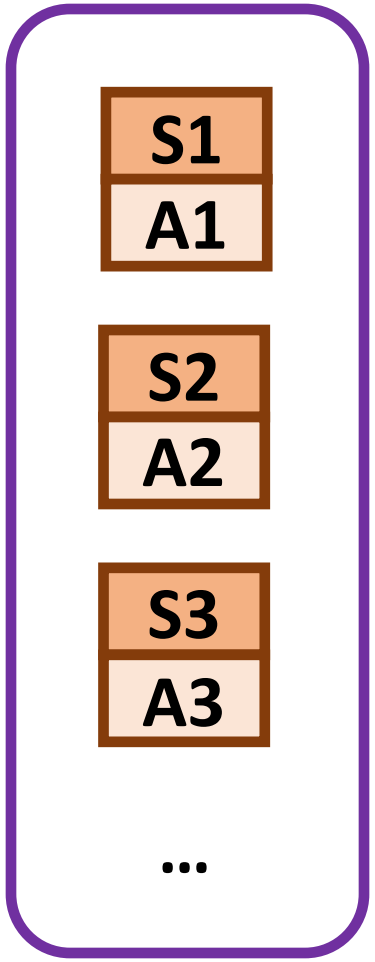
- SMALL-AMP is a replication of Dspot [Danglot 2019] in Pharo
  - And is yet under development
- DSpot is based on 2 techniques:
  - Input amplification
    - Evolutionary test generation [Tonella, 2004]
  - Assert amplification
    - Test oracle generation [Xie, 2006]

[Danglot 2019] Benjamin Danglot, Oscar Luis Vera-Pérez, Benoit Baudry, and Martin Monperrus. 2019. Automatic Test Improvement with DSpot: a Study with Ten Mature Open-Source Projects. Empirical Software Engineering, Springer Verlag (2019).

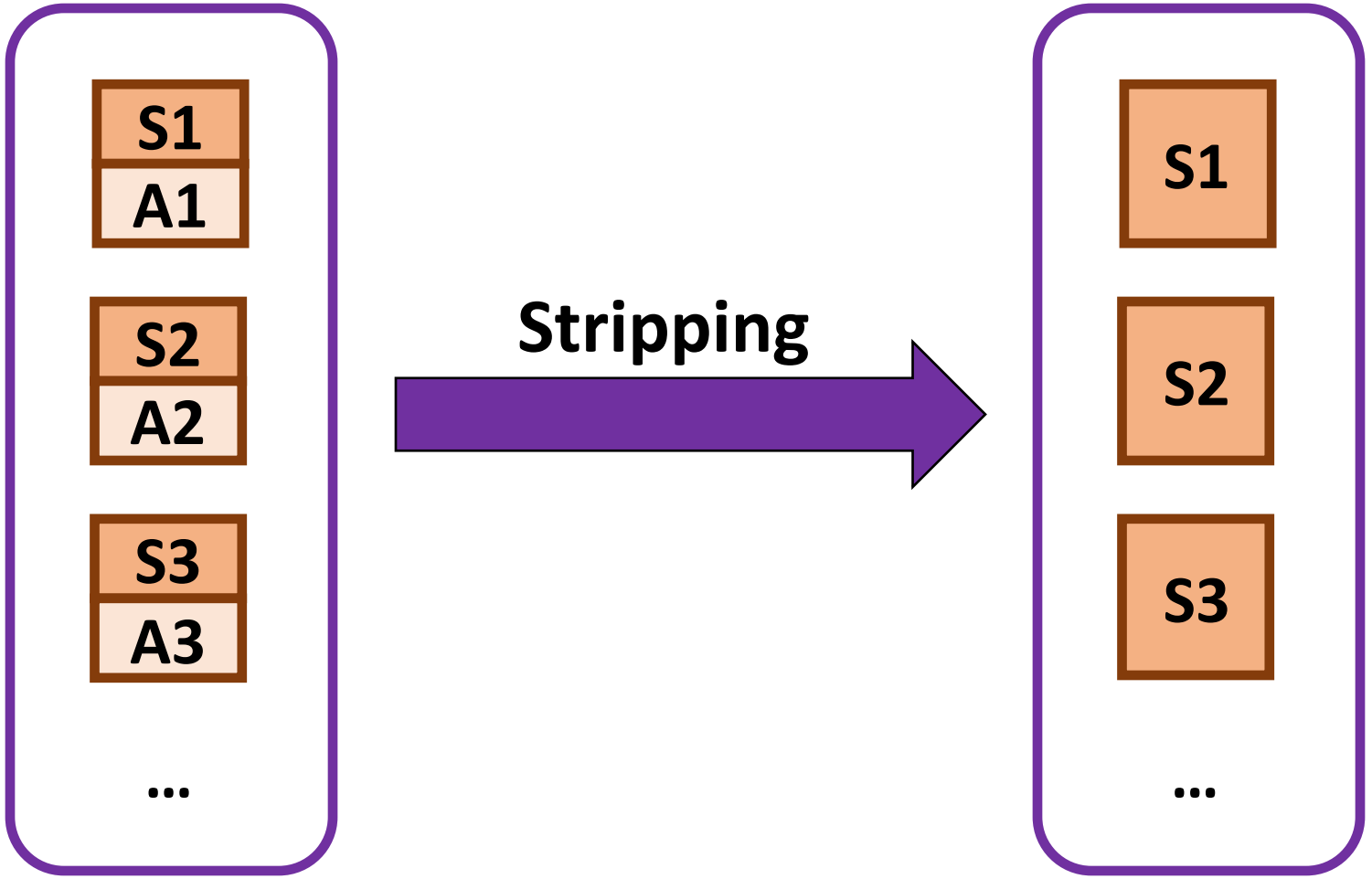
[Tonella, 2004] Paolo Tonella. 2004. Evolutionary testing of classes. Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis - ISSTA '04 (2004).

[Xie, 2006] Tao Xie. 2006. Augmenting Automatically Generated Unit-Test Suites with Regression Oracle Checking. Lecture Notes in Computer Science (2006), 380–403.

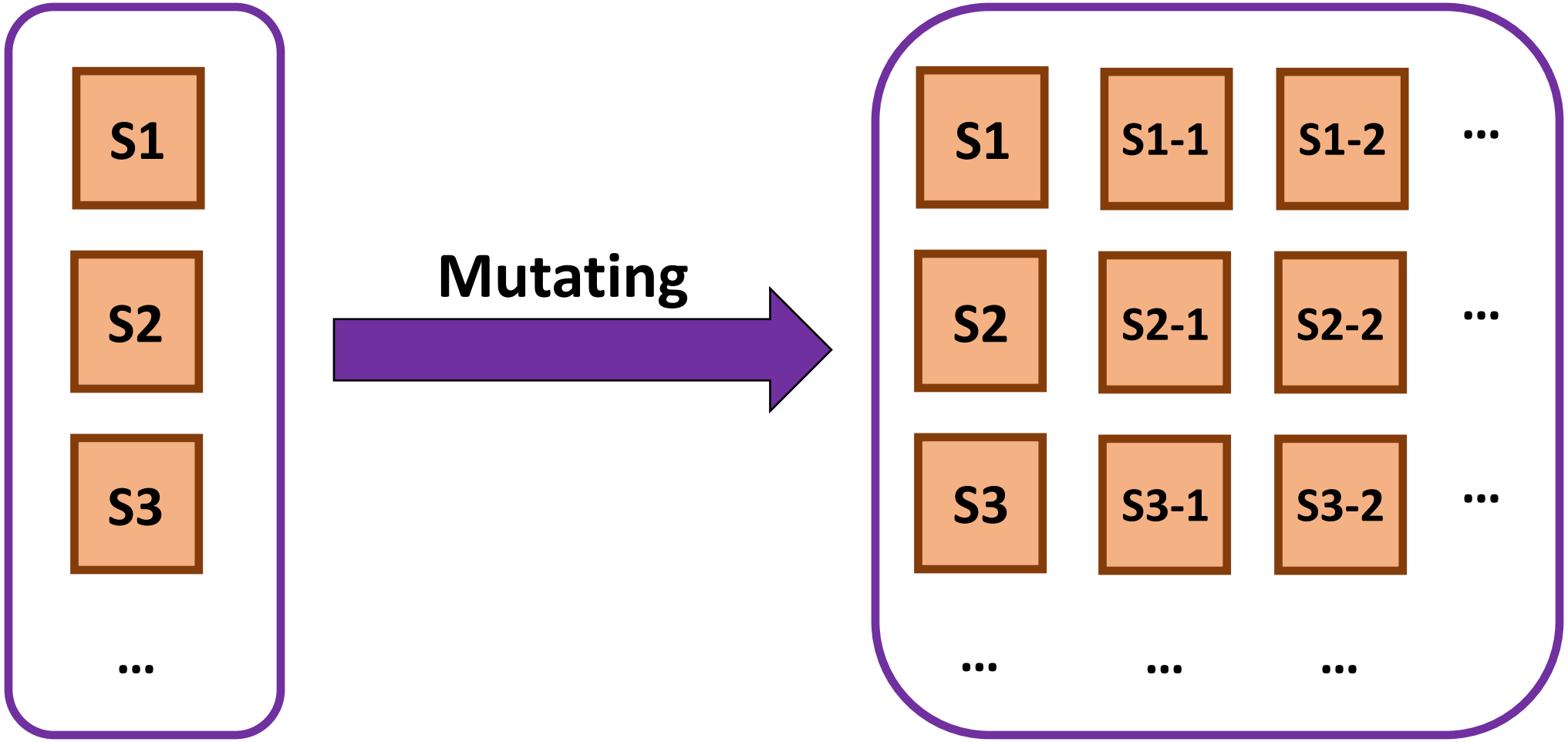
Initial population



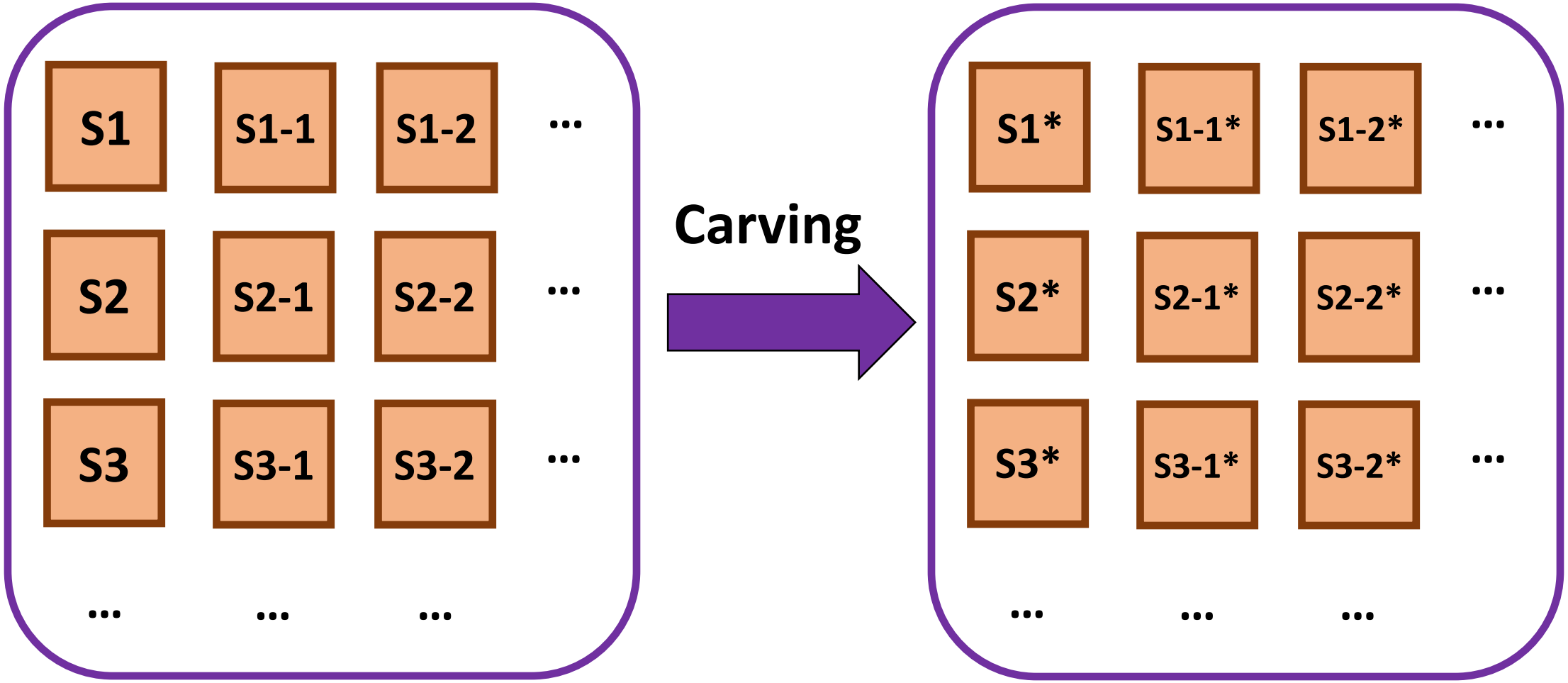
Stripping: Removing assertions



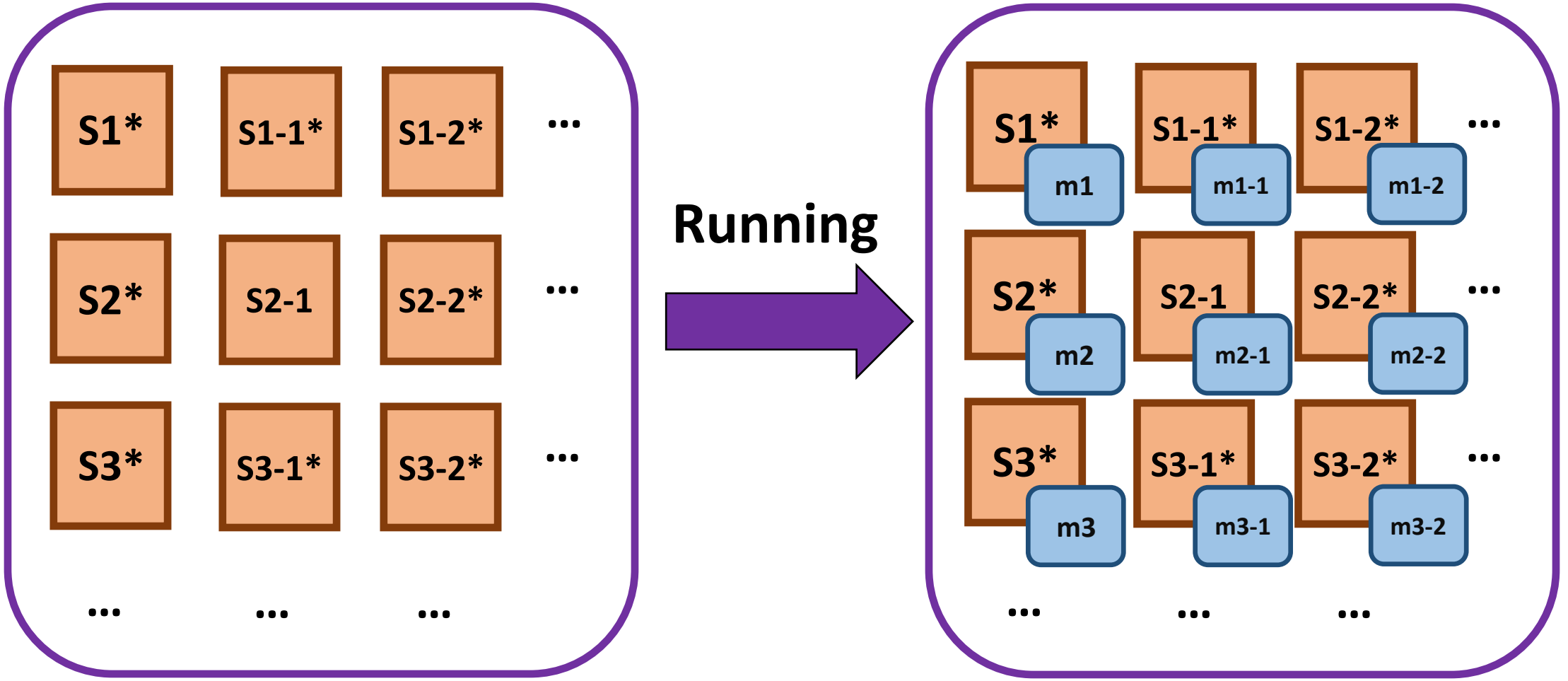
Mutating: New versions of test-cases



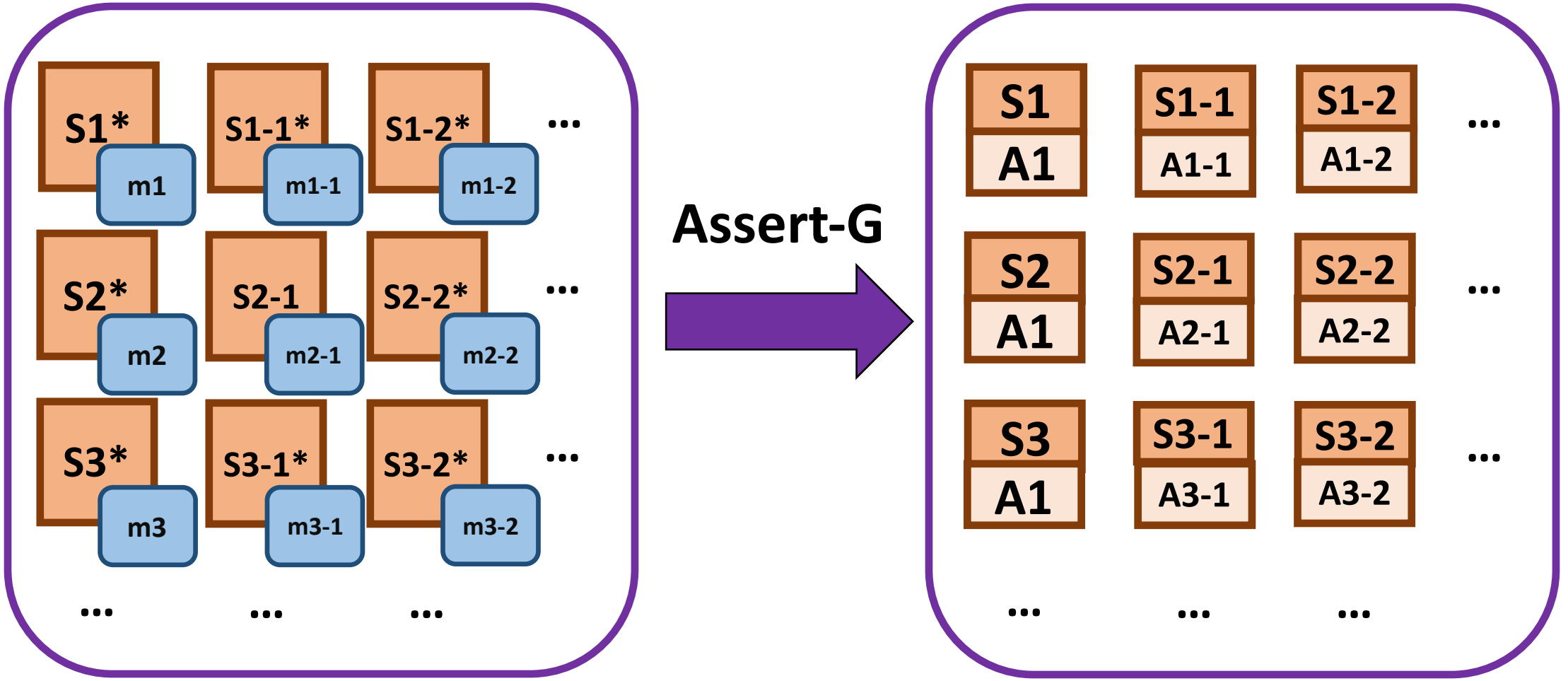
Carving: Inserting observation points



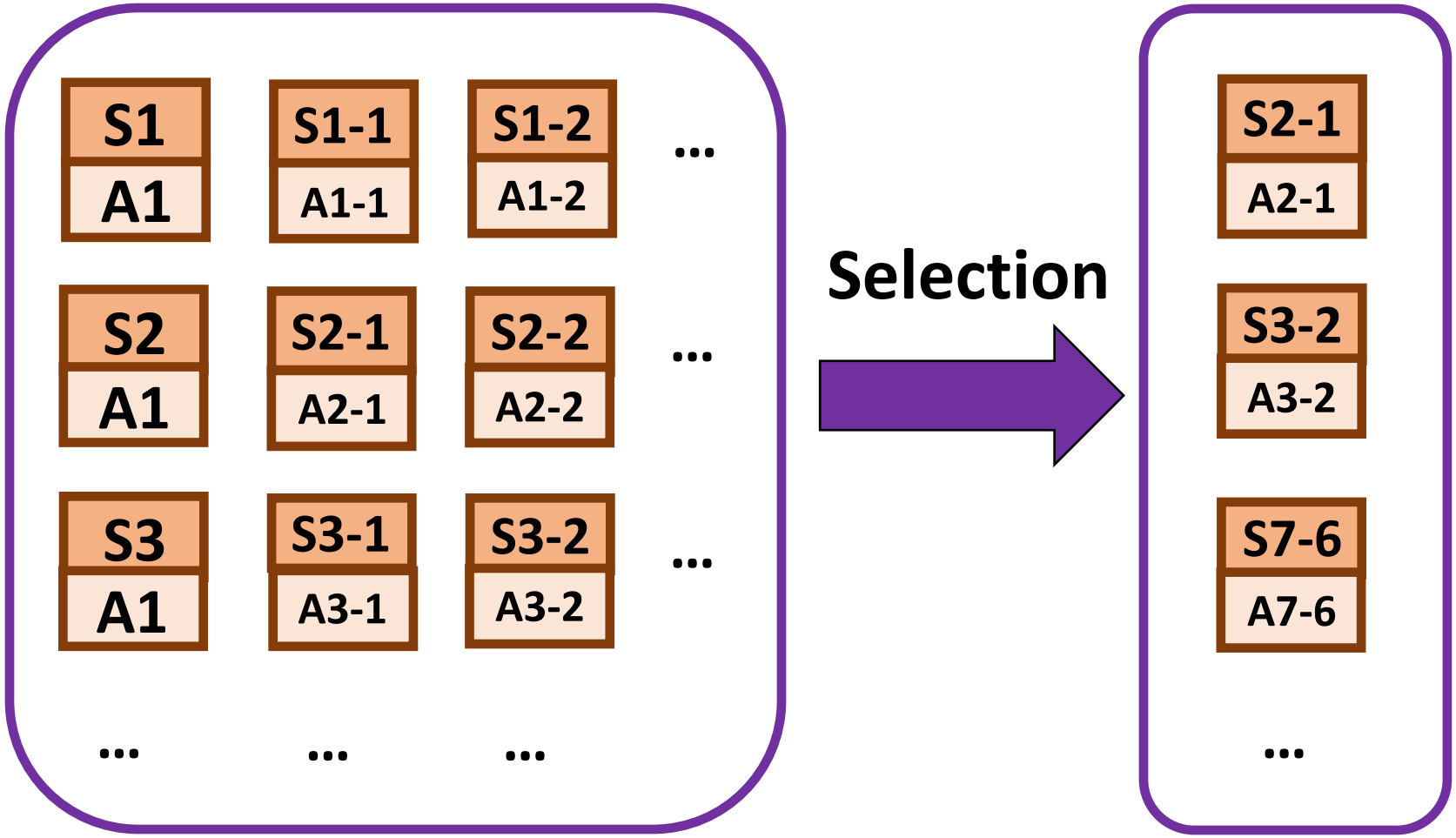
Running: Executing test-cases



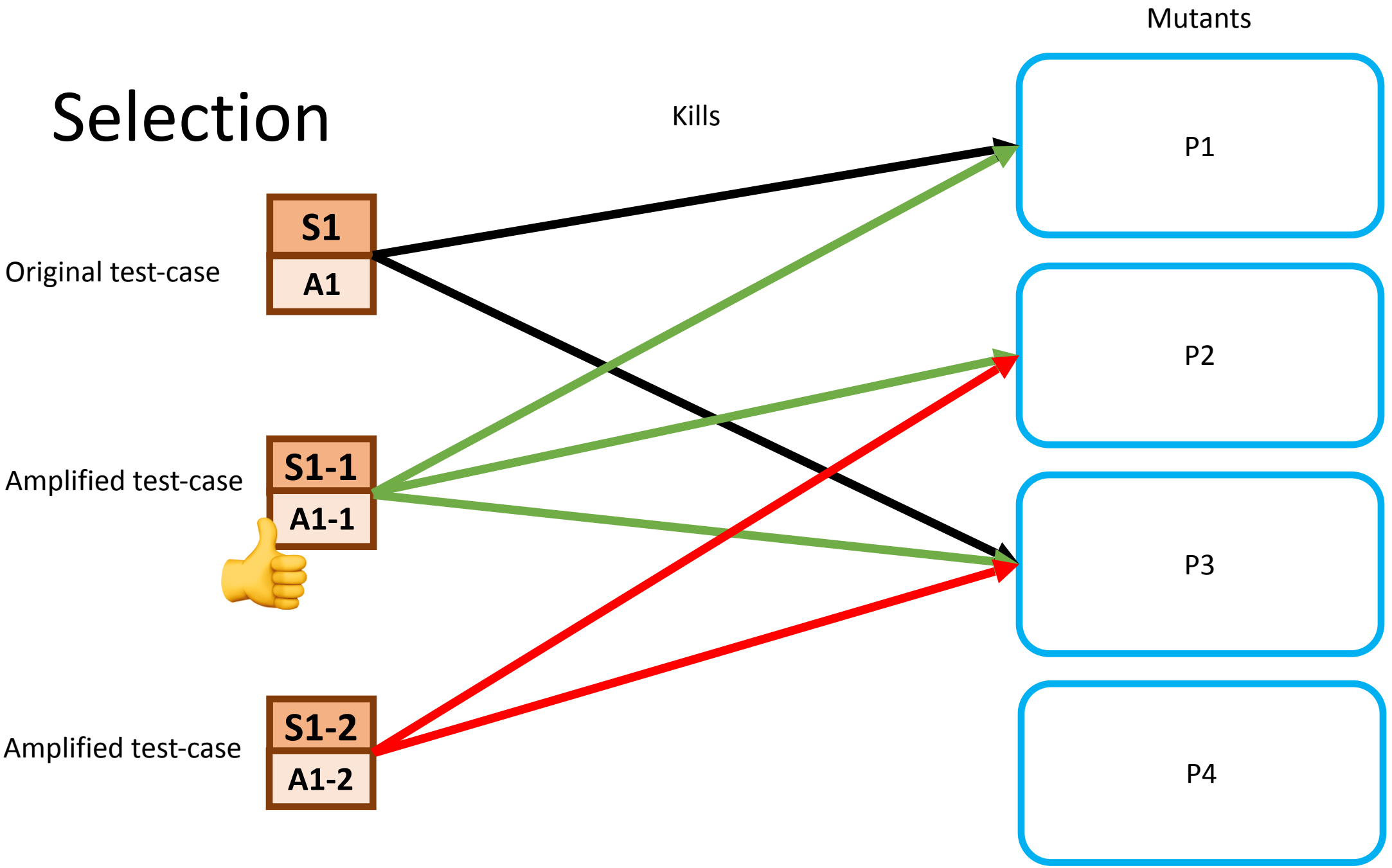
Assert Generation



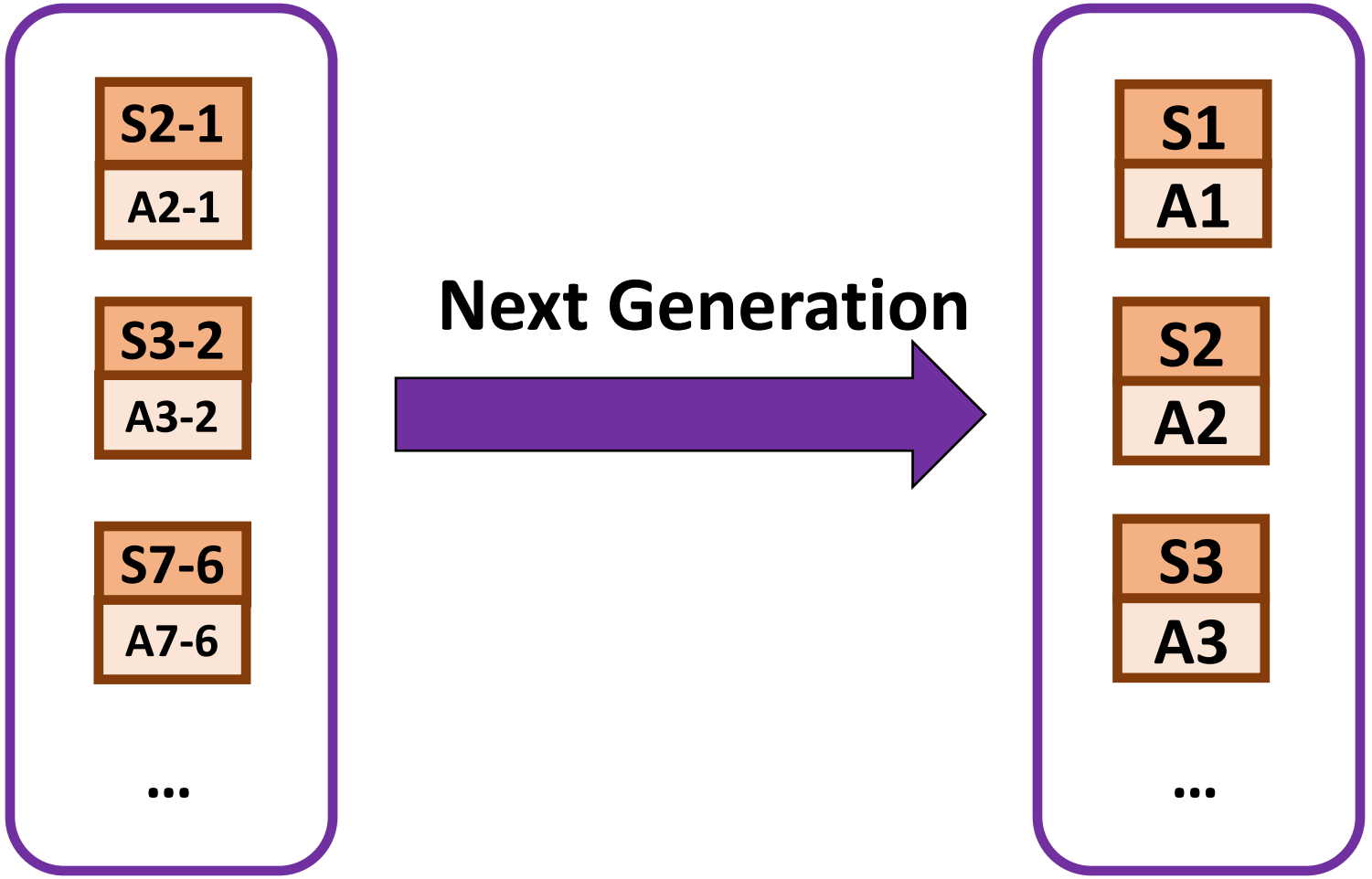
Selection

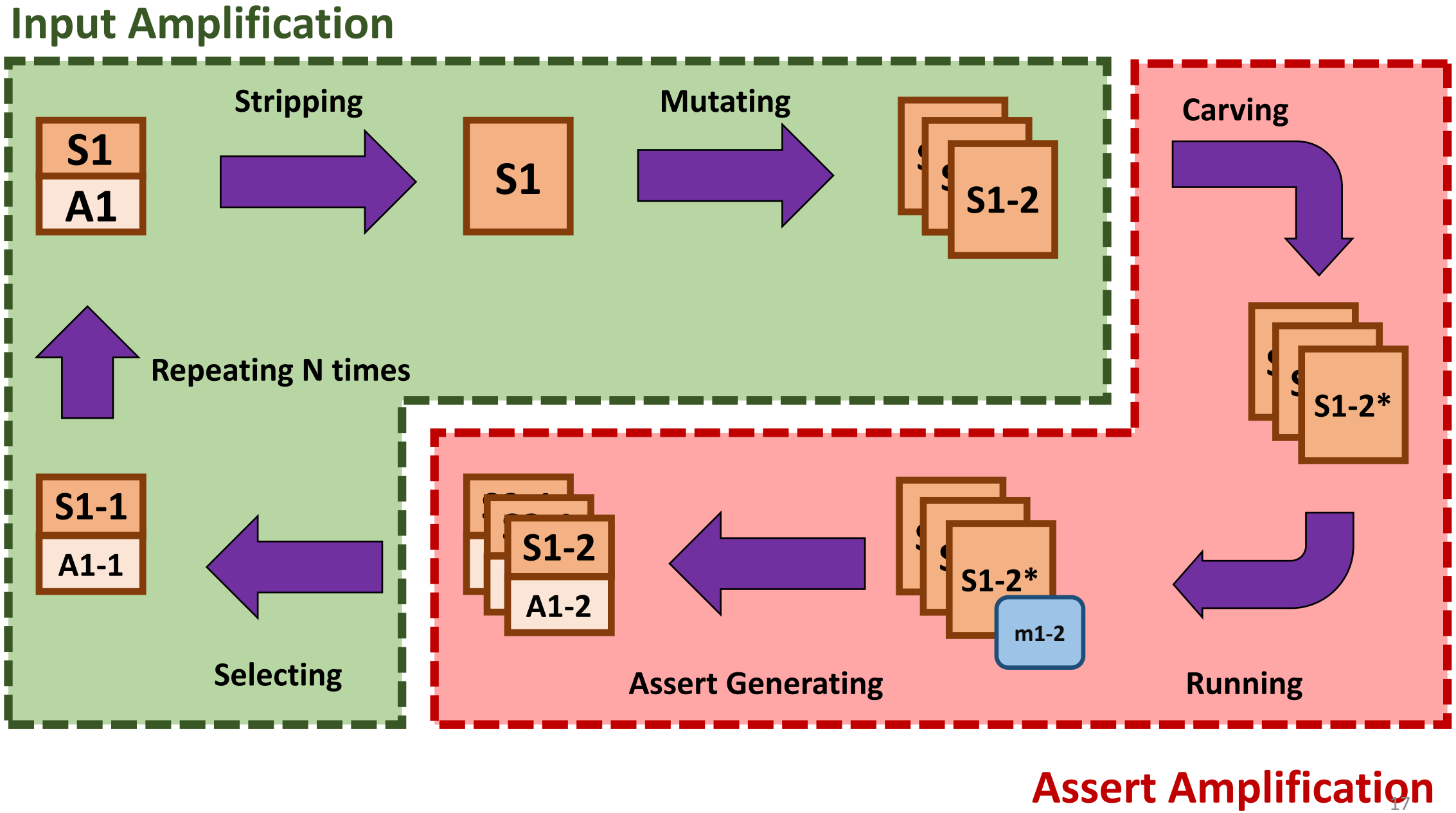

S1
A1
S1-1
A1-1
S1-2
A1-2
...
S2
A1
S2-1
A2-1
S2-2
A2-2
...
S3
A1
S3-1
A3-1
S3-2
A3-2
...
...
...
...
Selection
S2-1
A2-1
S3-2
A3-2
S7-6
A7-6
...

Selection

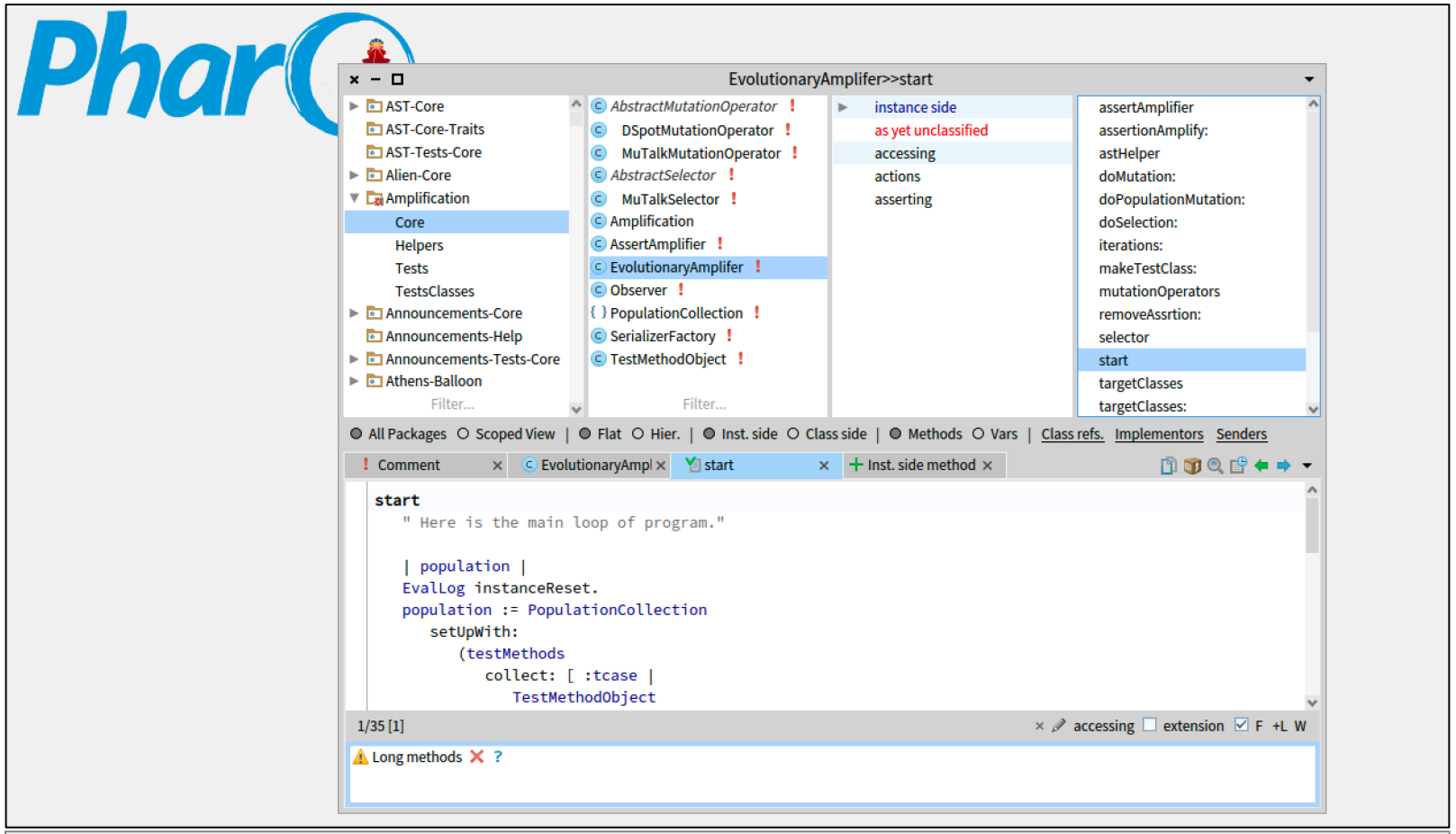

Mutants
Kills
P1
P2
P3
P4
S1
A1
Original test-case
S1-1
A1-1
Amplified test-case
S1-2
A1-2
Amplified test-case

Iteration N time




Input Amplification
Stripping
Mutating
Carving
S1
A1
S1
S1-2
S1-2*
Repeating N times
S1-1
A1-1
S1-2
A1-2
S1-2*
m1-2
Selecting
Assert Generating
Running
Assert Amplification

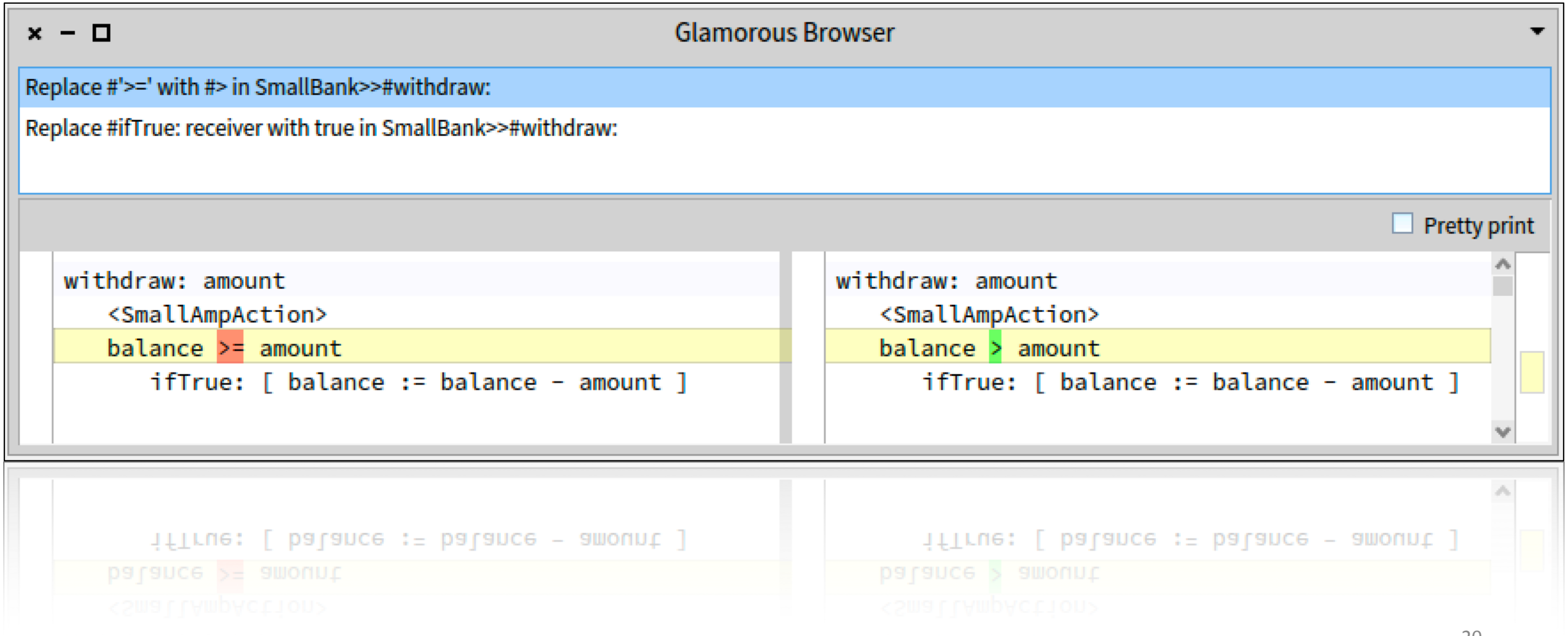
Implementation



https://github.com/mabdi/small-amp

```
testDeposit
    | b |
    b := SmallBank new.
    b deposit: 10.
    self assert: b balance equals: 10.
    b deposit: 100.
    self assert: b balance equals: 110

testWithdraw
    | b |
    b := SmallBank new.
    b deposit: 100.
    self assert: b balance equals: 100.
    b withdraw: 30.
    self assert: b balance equals: 70
```

Mutation coverage

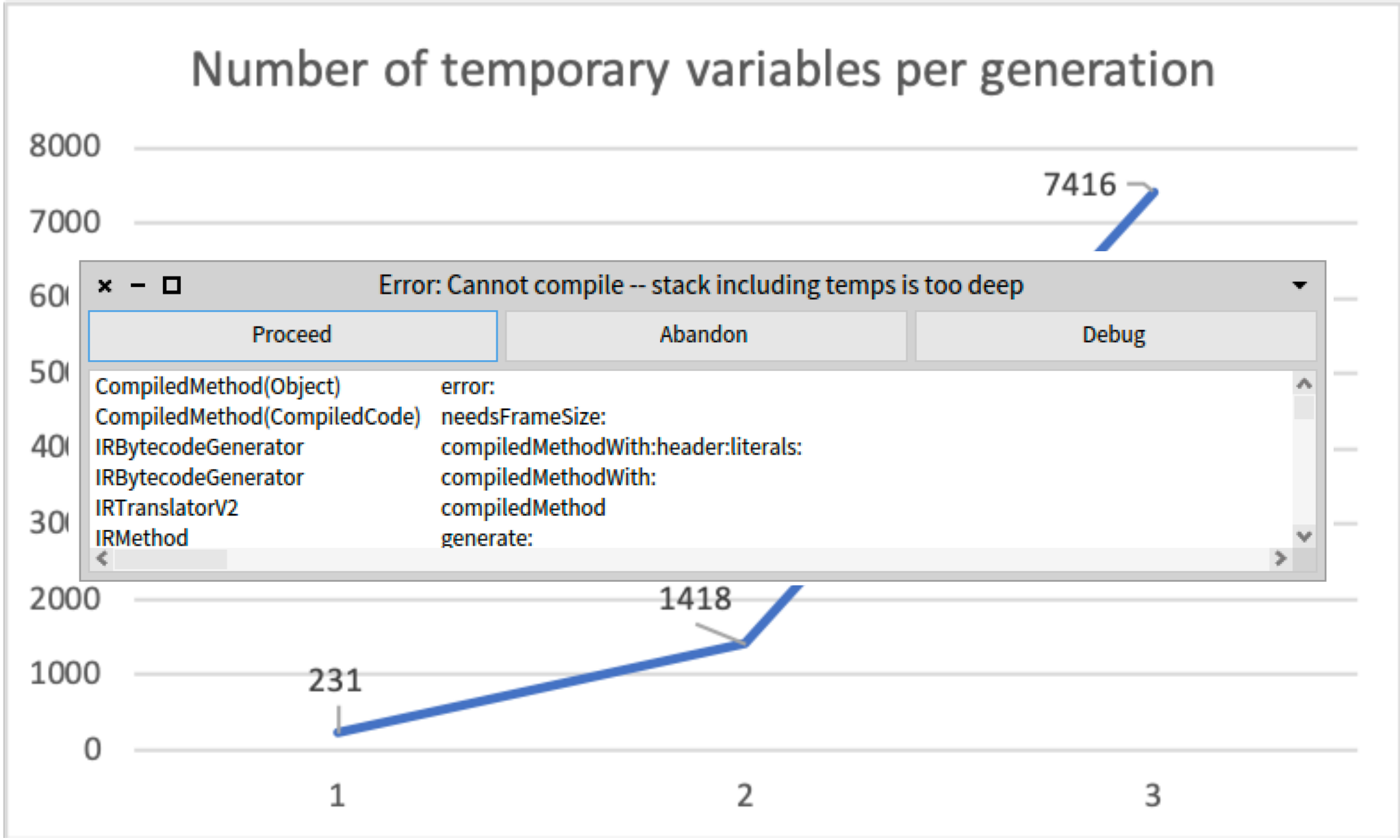


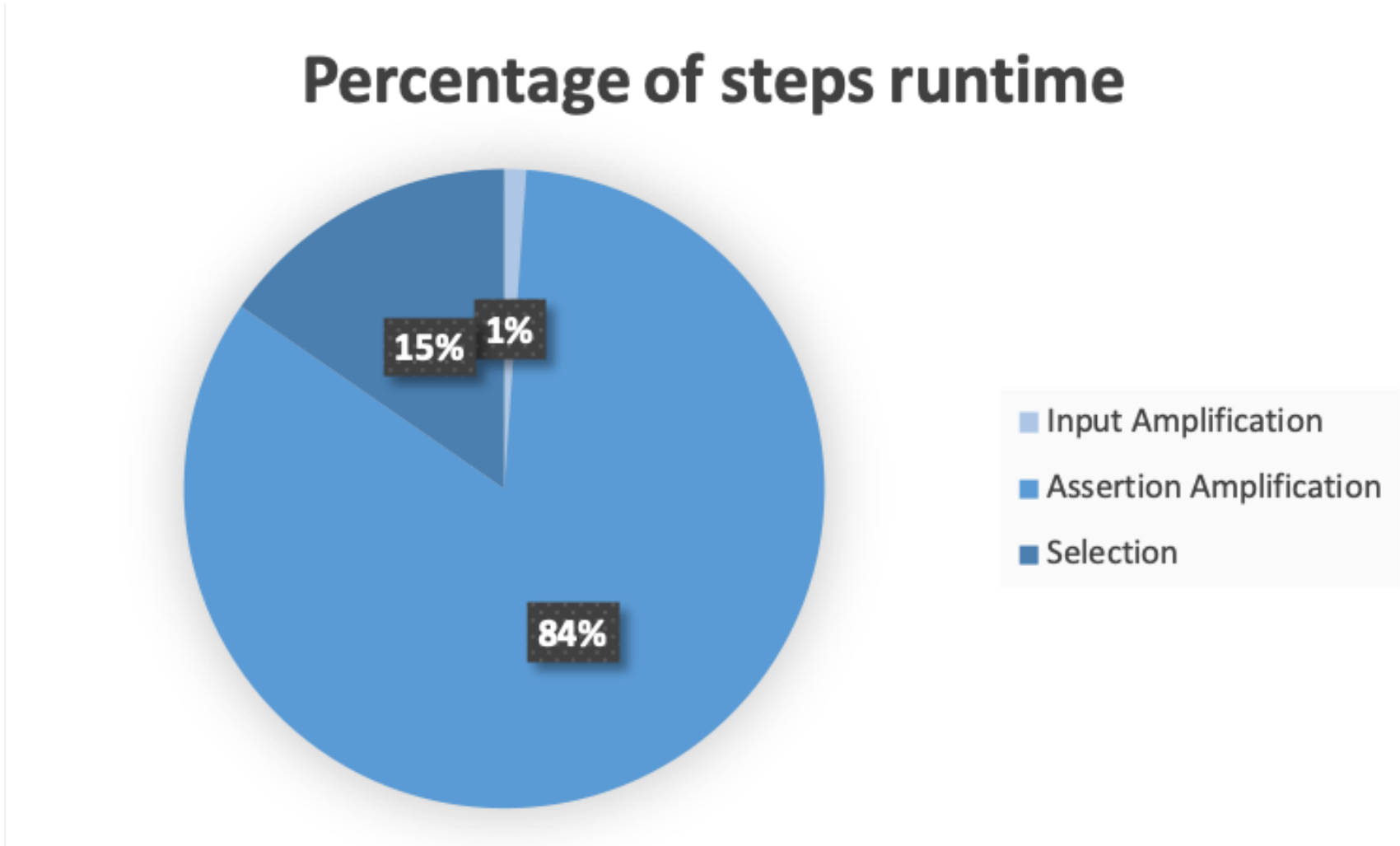
Lesson learned 1: Dynamic Language

- Lack of static type system
  - Test body mutating
    - Literal mutation
  - Mutation analysis (Selection)
    - Former works: Smutant and MuTalk
  - Assert generation
    - It’s a dynamic process
- Different structure
  - Cascades and nested message sends
    - Easy to normalize
  - Blocks (it’s ignored currently!)

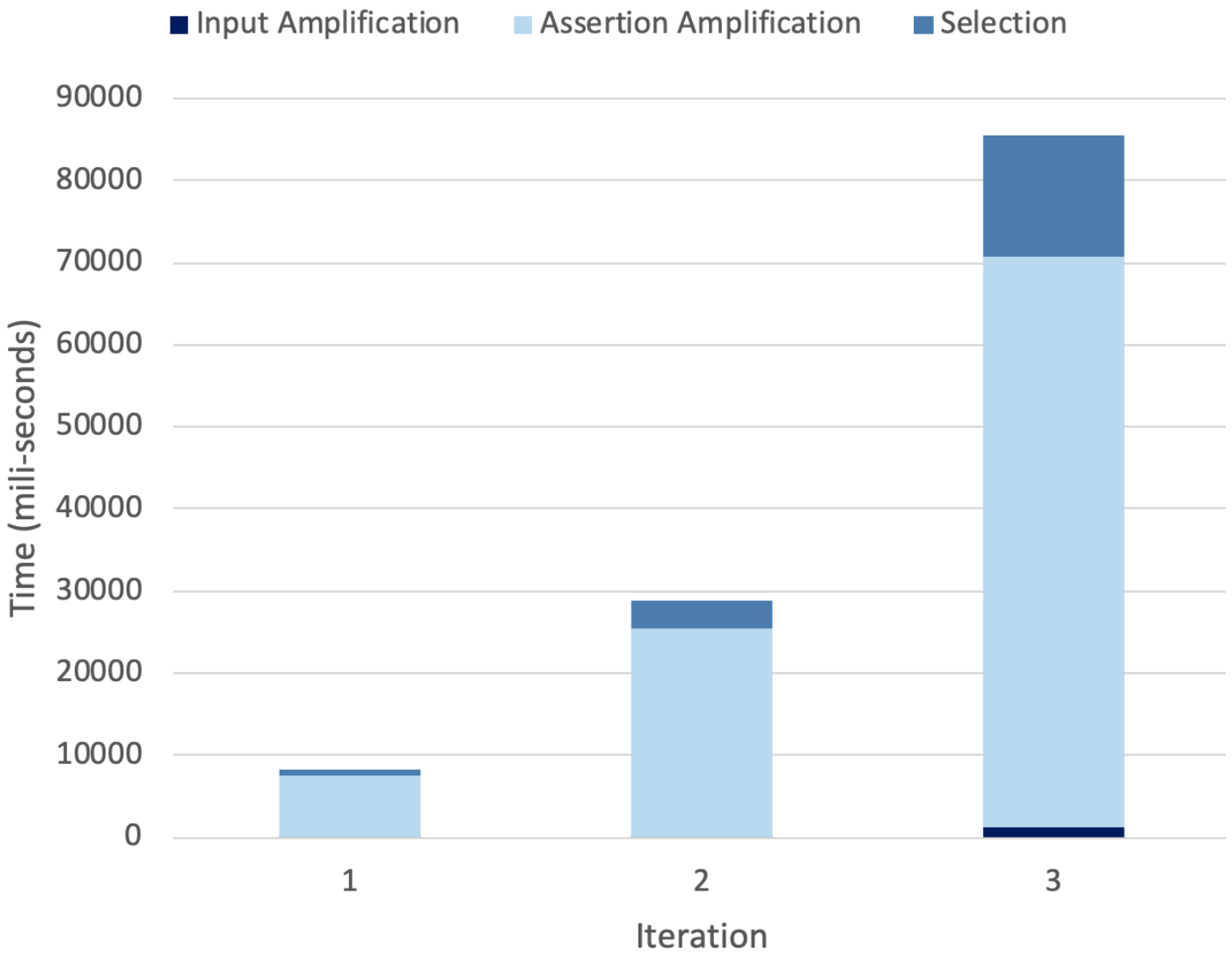
Lesson learned 2: The costs

- Number of temp variables
- Assert amplify costs
- Small number of iterations
  - Small N -> No evolution!


Number of temporary variables per generation
8000
7000
60
50
40
30
2000
1000
0
1
2
3
7416
1418
231
Error: Cannot compile -- stack including temps is too deep
Proceed
Abandon
Debug
CompiledMethod(Object) error:
CompiledMethod(CompiledCode) needsFrameSize:
IRBytecodeGenerator compiledMethodWith:header:literals:
IRBytecodeGenerator compiledMethodWith:
IRTranslatorV2 compiledMethod
IRMethod generate:


Percentage of steps runtime
15%
1%
84%
Input Amplification
Assertion Amplification
Selection

Run time performance per iteration



Lesson learned 3: Ugly result

- Hard to understand
- Hard to maintain

```
testWithdraw_13_1
    | b tmp_EZeM6tpa6L1 tmp_lSKWzgvgyA2 tmp_uZHTCNlbrV3 tmp_QjRCv7cBVF4 tmp_NLB2aqKzKK1 tmp_kXMDtgPEgR2
tmp_L1bQ4mmOjF3 tmp_UrU63KPiDy4 tmp_wiTLxkr4GF5 tmp_EGSYJCbXqK6 |
    b := SmallBank new.
    self assert: b balance equals: 0.
    tmp_NLB2aqKzKK1 := b balance = 1.
    self assert: tmp_NLB2aqKzKK1 equals: false.
    tmp_EZeM6tpa6L1 := b deposit: -1152921504606846976.
    self assert: tmp_EZeM6tpa6L1 balance equals: -1152921504606846976.
    self assert: b balance equals: -1152921504606846976.
    tmp_kXMDtgPEgR2 := tmp_EZeM6tpa6L1 balance = -1152921504606846976.
    self assert: tmp_kXMDtgPEgR2 equals: true.
    tmp_L1bQ4mmOjF3 := b balance = -1152921504606846976.
    self assert: tmp_L1bQ4mmOjF3 equals: true.
    tmp_lSKWzgvgyA2 := b balance = -1152921504606846976.
    self assert: tmp_lSKWzgvgyA2 equals: true.
    tmp_UrU63KPiDy4 := tmp_lSKWzgvgyA2 = true.
    self assert: tmp_UrU63KPiDy4 equals: true.
    tmp_uZHTCNlbrV3 := b withdraw: 30.
    self assert: tmp_uZHTCNlbrV3 balance equals: -1152921504606846976.
    tmp_wiTLxkr4GF5 := tmp_uZHTCNlbrV3 balance = -1152921504606846976.
    self assert: tmp_wiTLxkr4GF5 equals: true.
    tmp_QjRCv7cBVF4 := b balance = 70.
    self assert: tmp_QjRCv7cBVF4 equals: false.
    tmp_EGSYJCbXqK6 := tmp_QjRCv7cBVF4 = false.
    self assert: tmp_EGSYJCbXqK6 equals: true
```

Too many temp variables

Some checks make no sense! Or are redundant

Too many assert statements.

Strange random values

We have implemented

- Clean-up extra code after each generation
  - Identify assertion statements that are redundant
  - Discard extra temp variables

```
testWithdraw_12_5
    | b |
    b := SmallBank new.
    b deposit: SmallInteger maxVal.
    self assert: b balance equals: 11529215046068469 75.
    b withdraw: SmallInteger maxVal.
    self assert: b balance equals: 0
```

Next directions

- Using test amplification in **real application**
  - With a mature test suite
- Make generated tests more understandable
  - Modeling good tests
  - Building good tests

We welcome

- Suggestions
- Collaborations
  - Real applications
- And else …