CodaMOSA: Escaping Coverage Plateaus in Test Generation with Large Language Models

Caroline Lemieux
University of British Columbia

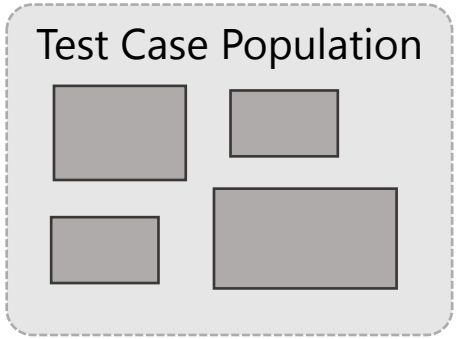
Jeevana Priya Inala
Microsoft Research

Shuvendu K. Lahiri
Microsoft Research

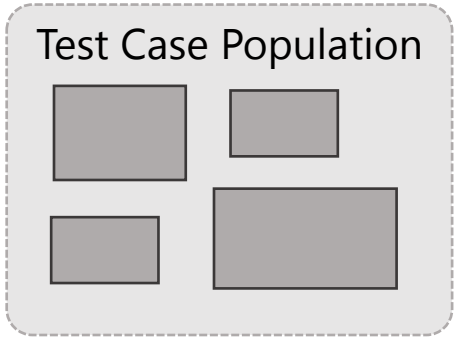
Siddhartha Sen
Microsoft Research

In Proceedings of the *45th IEEE/ACM International Conference on Software Engineering (ICSE'23)*, Melbourne, Australia

Search-Based Test Suite Generation
(High-level view)

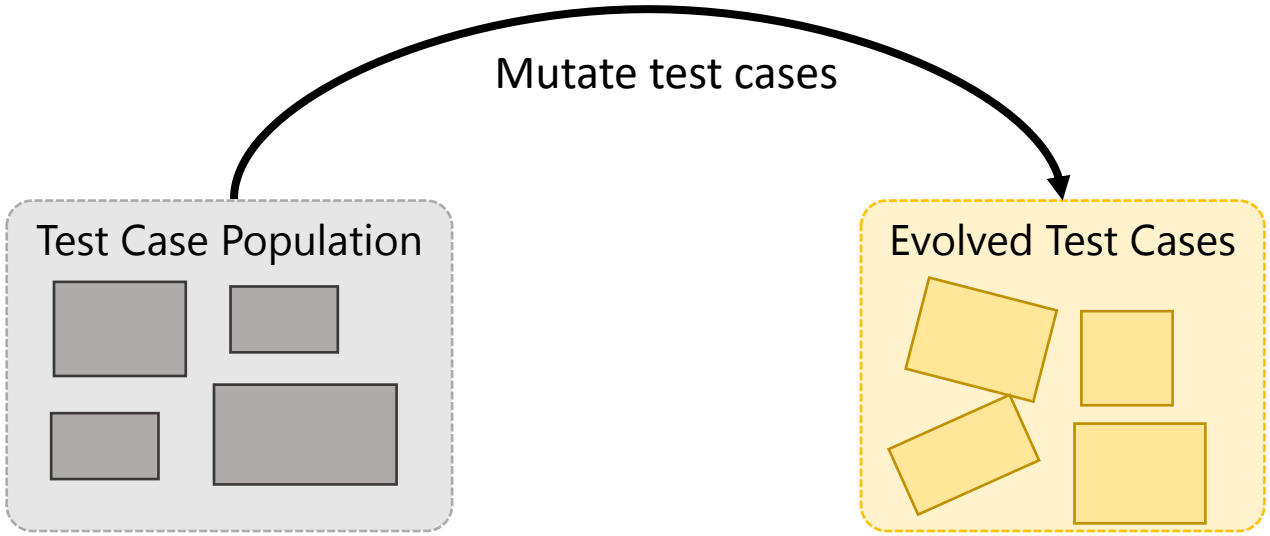


Search-Based Test Suite Generation
(High-level view)

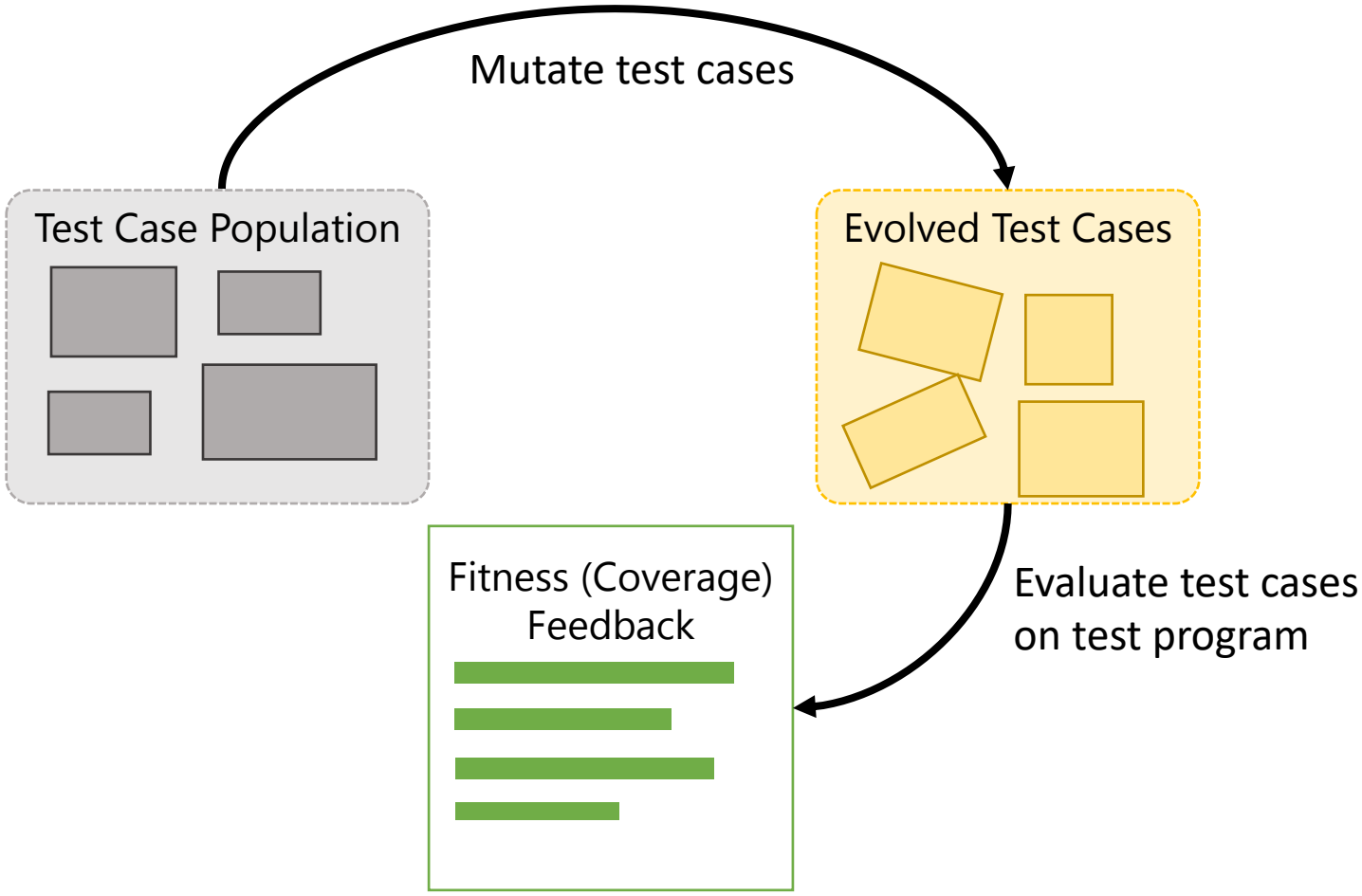


(usually, generate randomly)

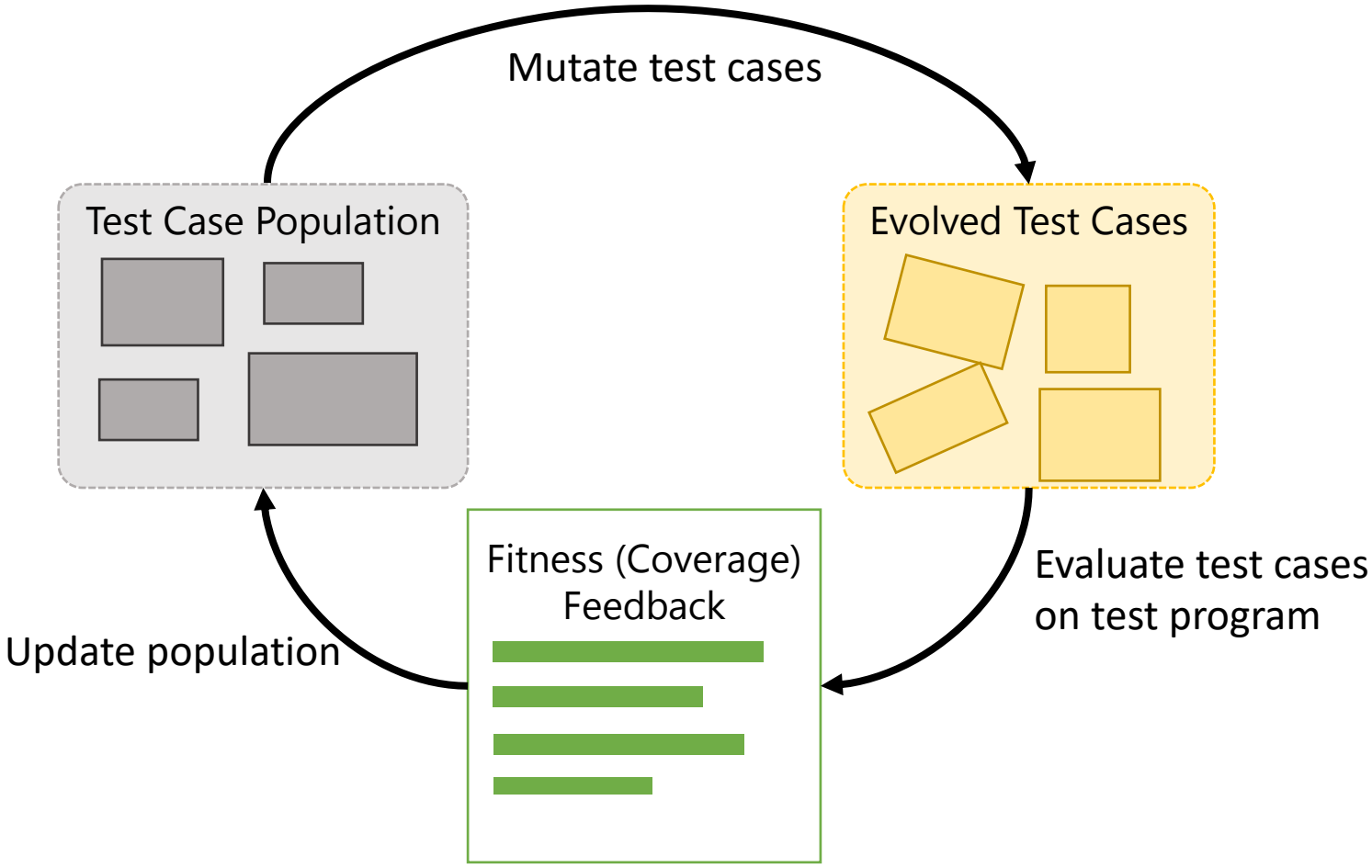
Search-Based Test Suite Generation
(High-level view)



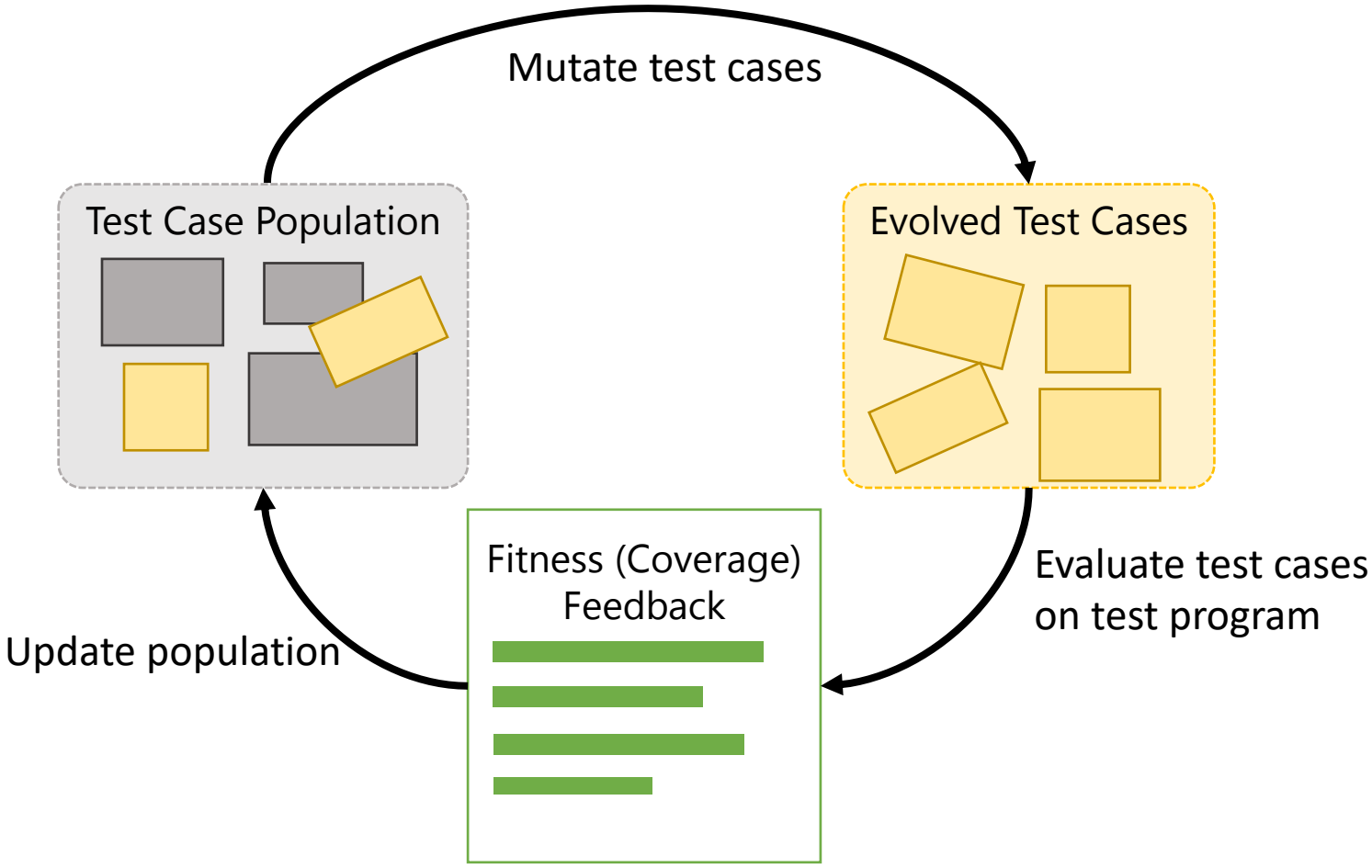
Search-Based Test Suite Generation
(High-level view)



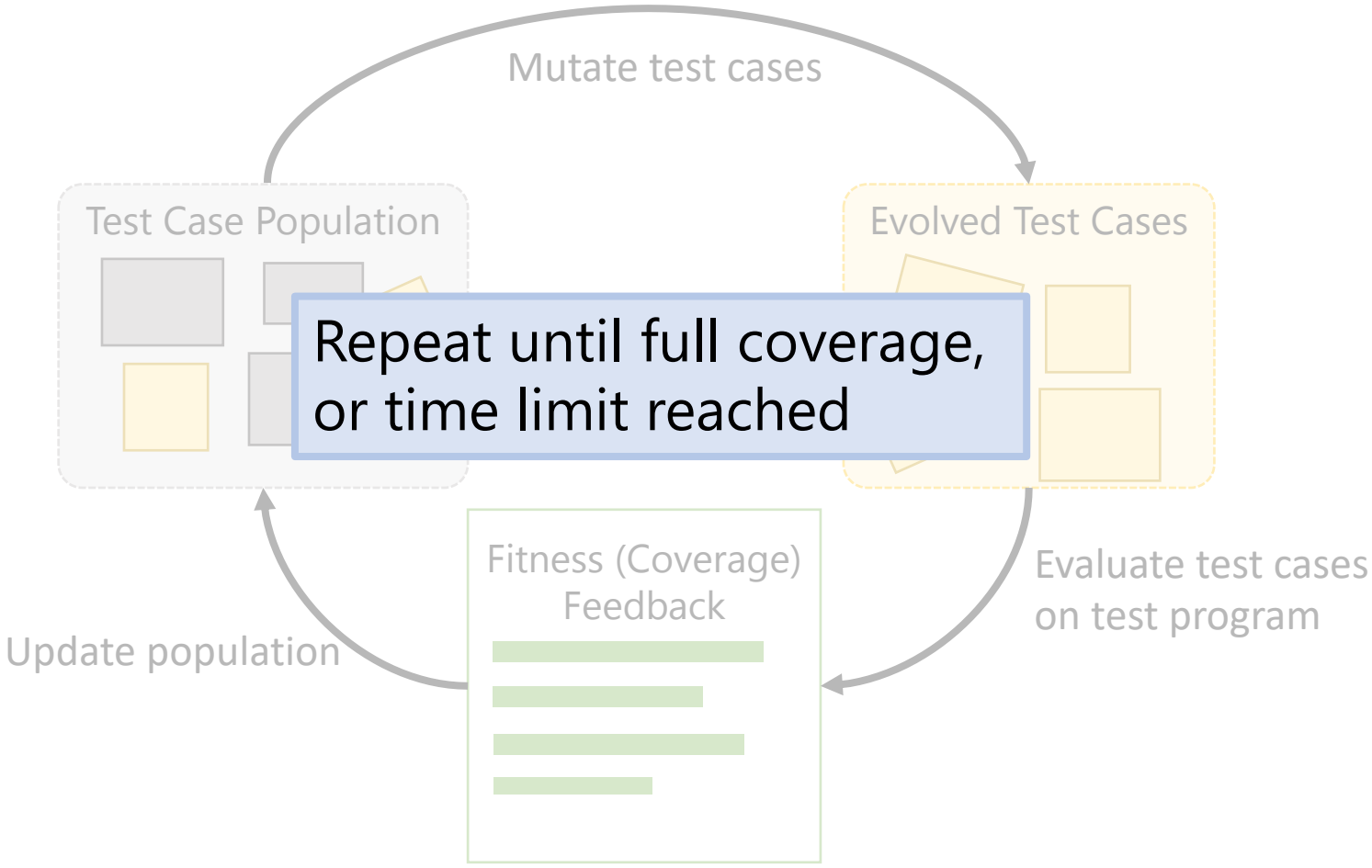
Search-Based Test Suite Generation
(High-level view)



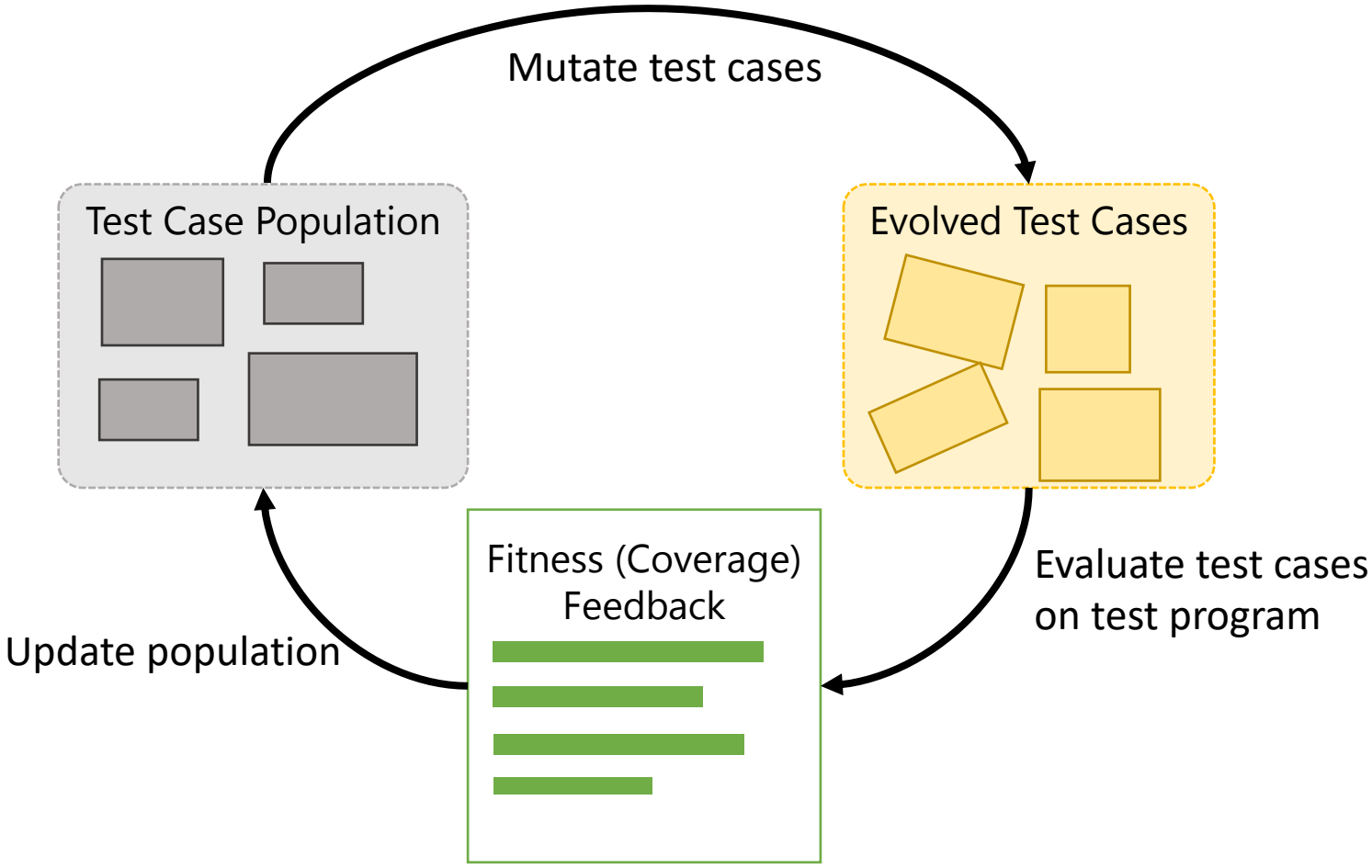
Search-Based Test Suite Generation
(High-level view)



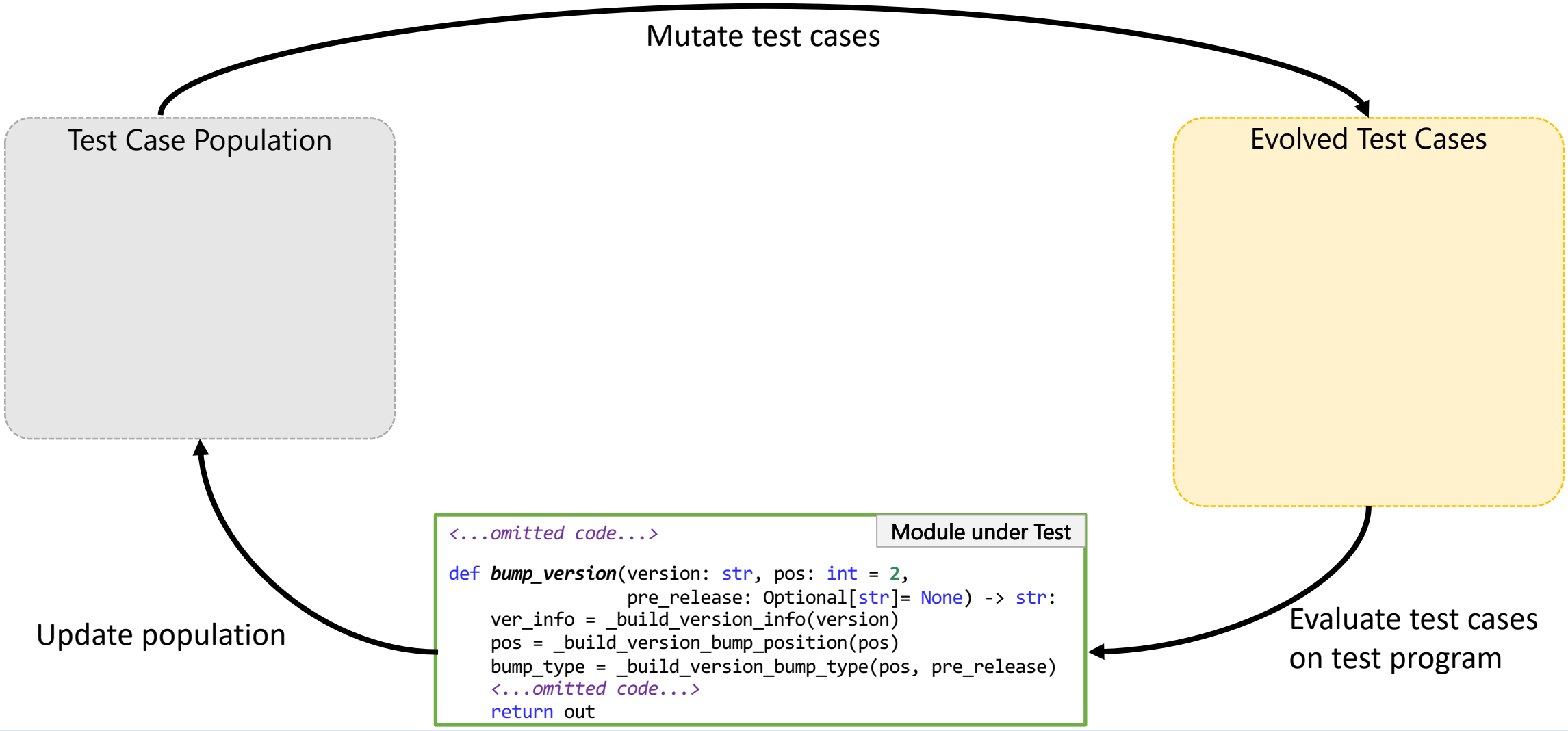
Search-Based Test Suite Generation
(High-level view)



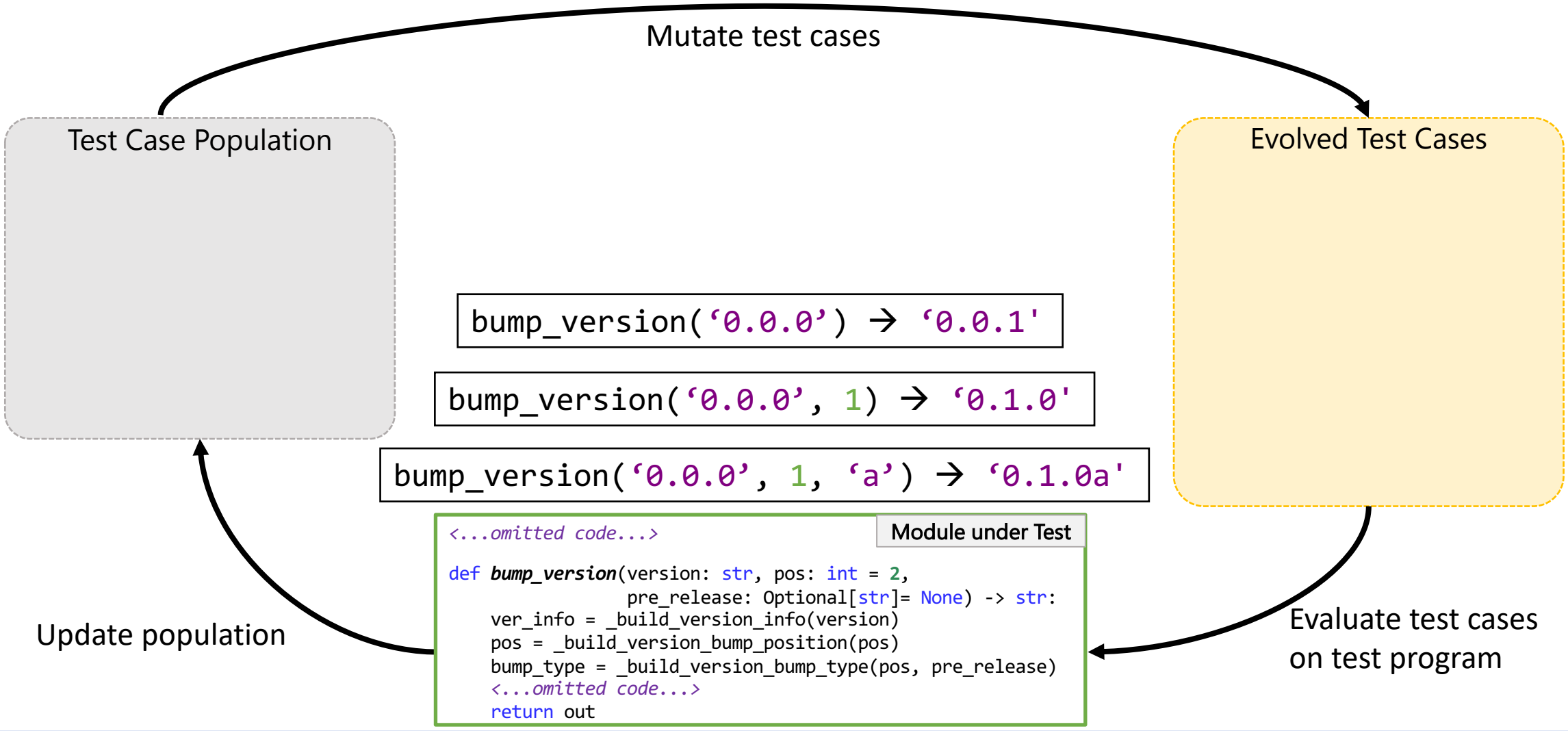
Search-Based Test Suite Generation
(High-level view)



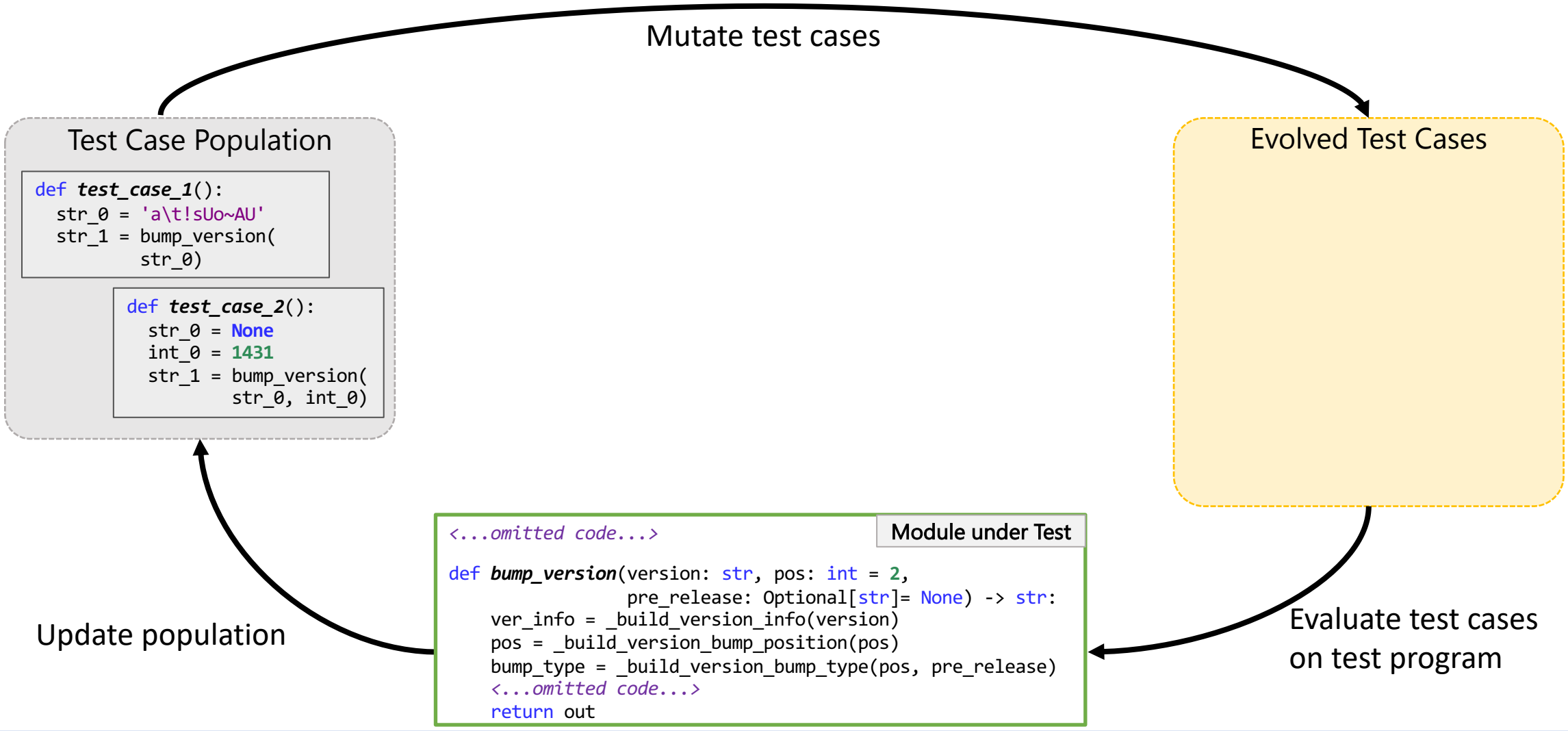
Concrete Example



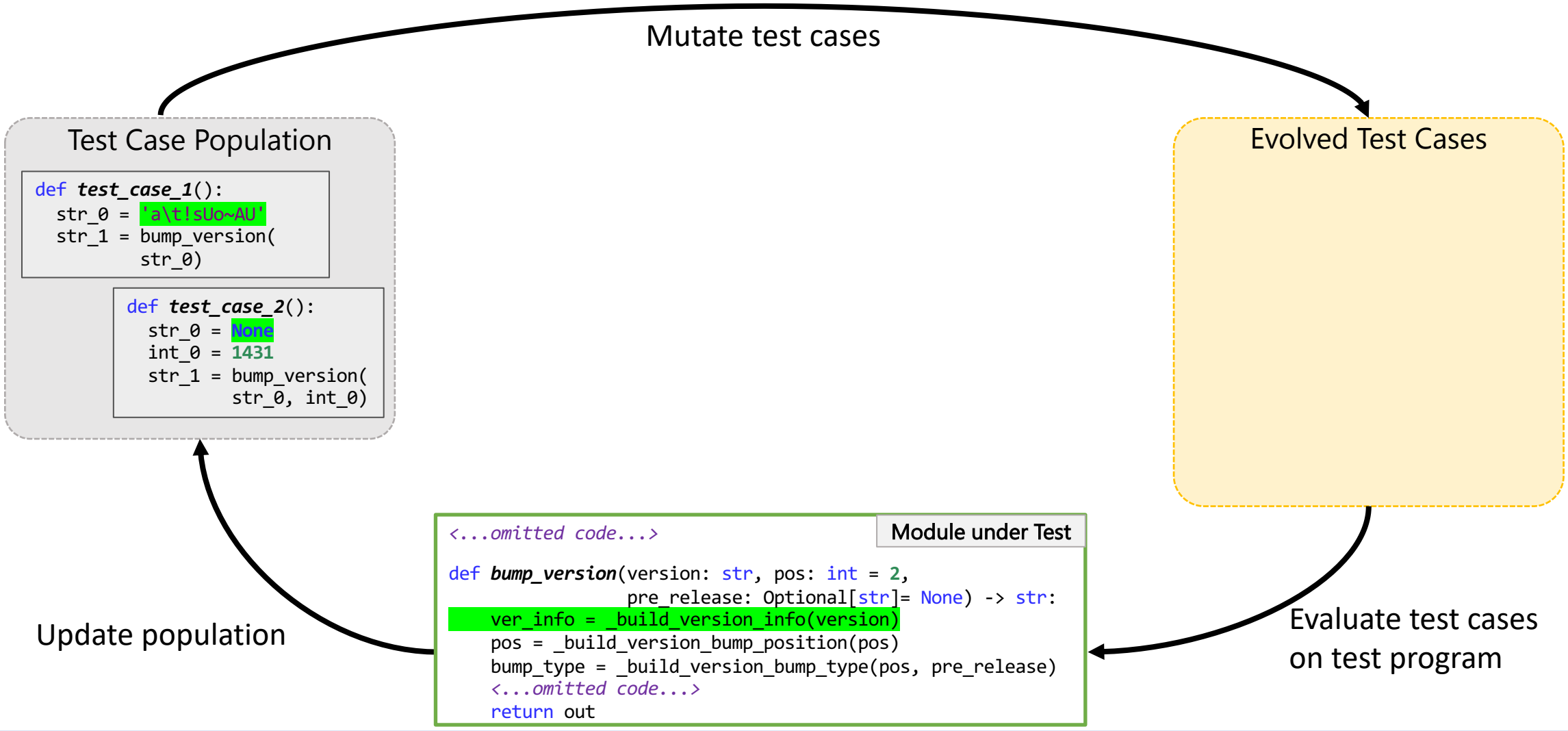
Example: Expected Behavior of Function



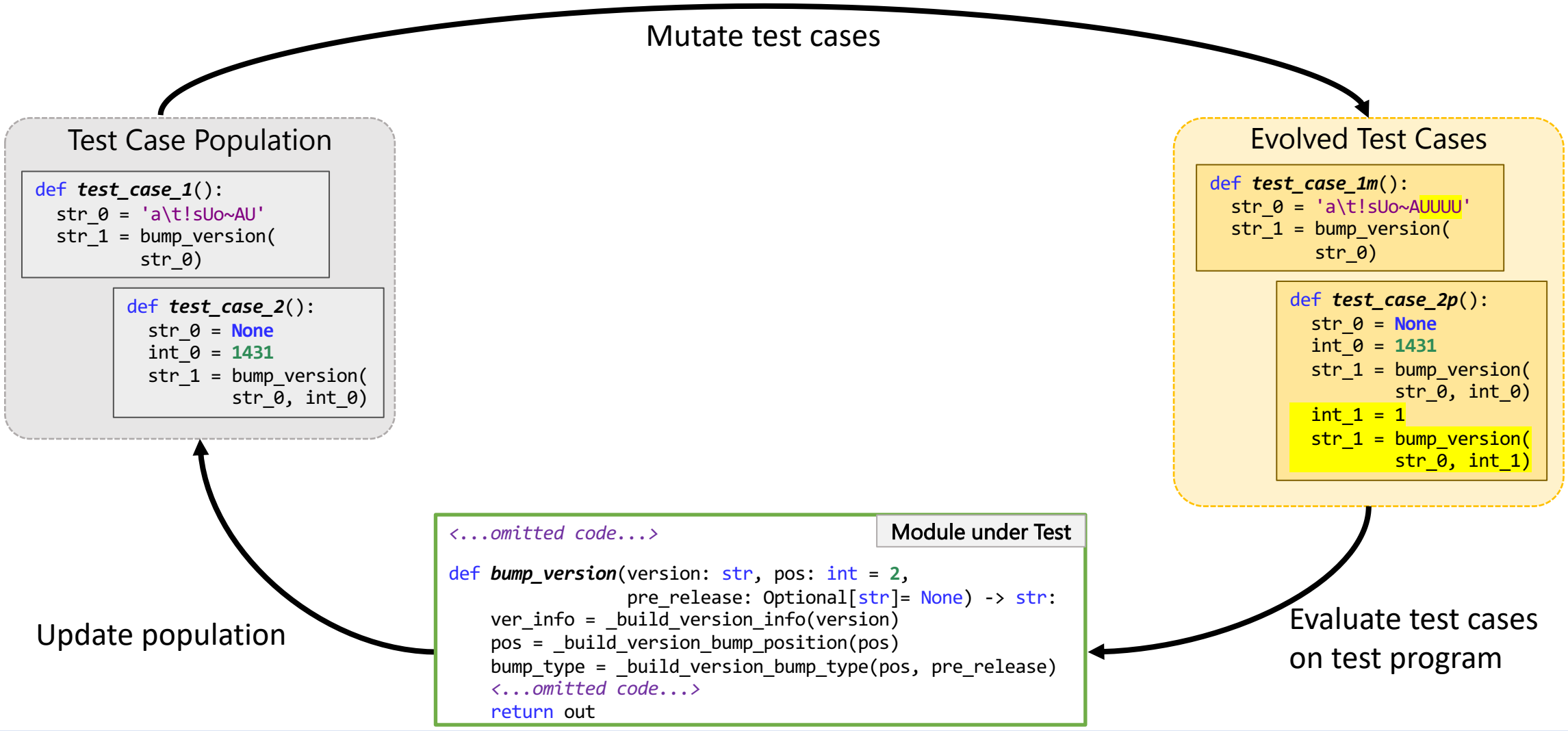
Example: Search-Based Test Suite Generation



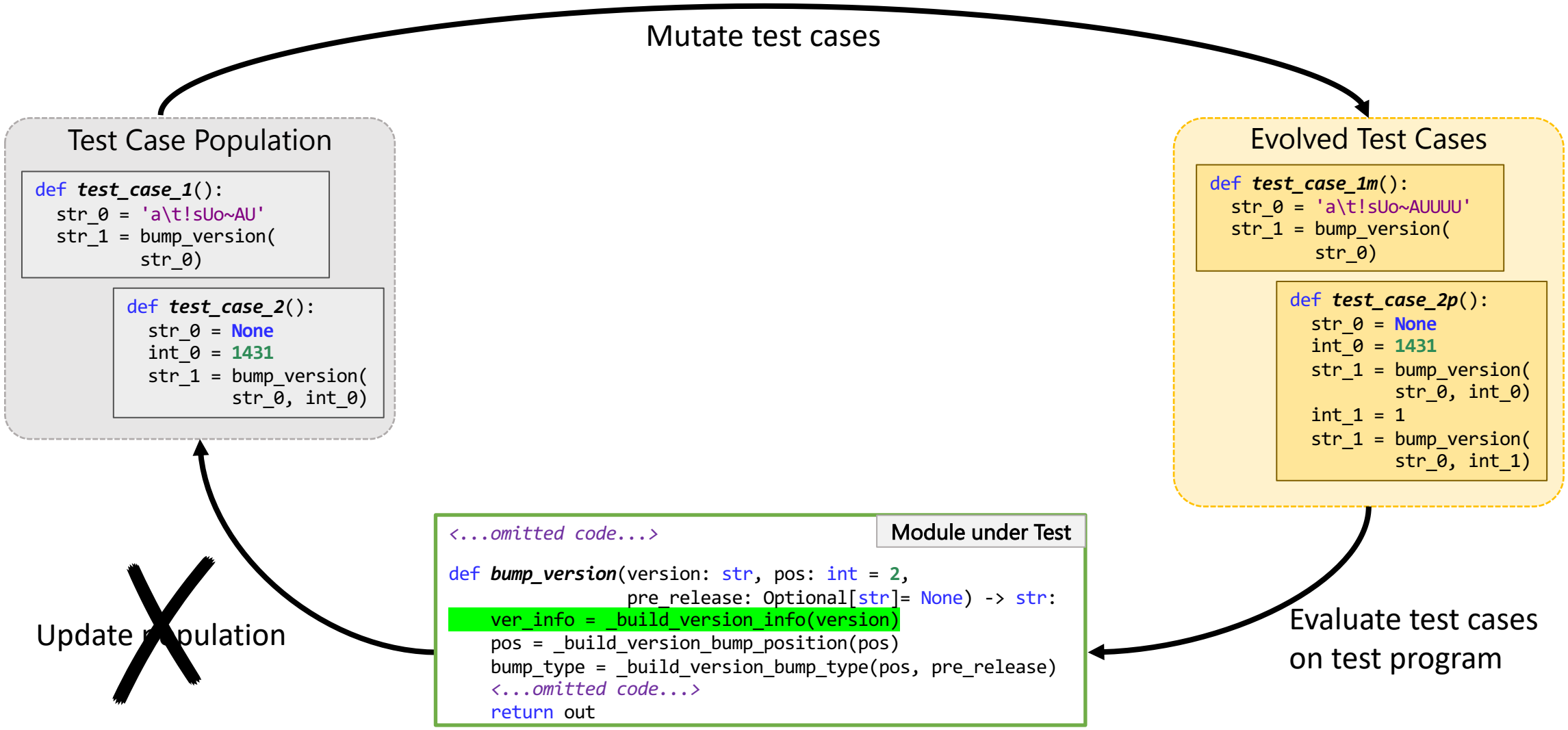
Current Tests have Low Coverage



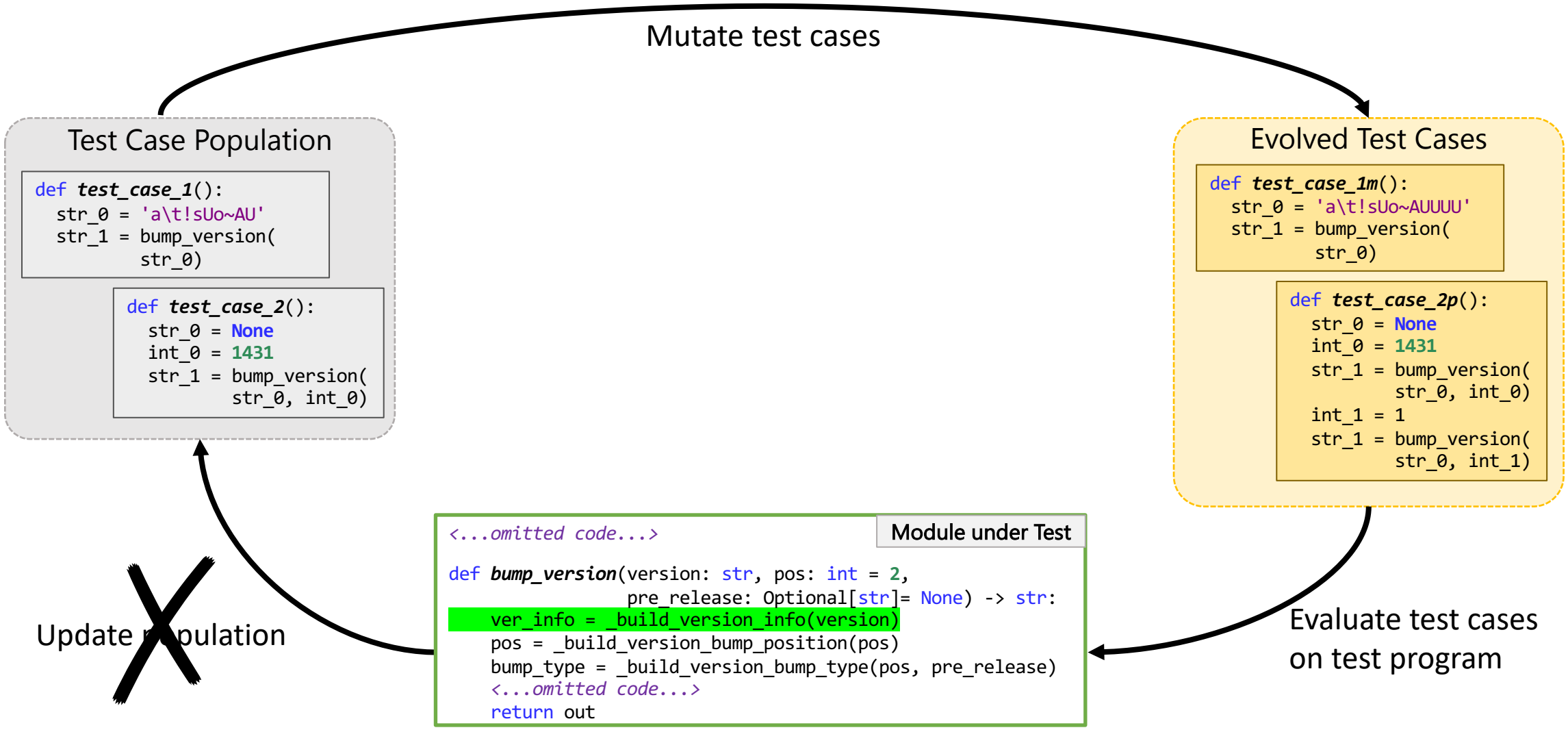
Create New Test Cases via Mutation



Mutation Unable to Increase Coverage



Search stalled. What to do now?



Core Approach

Evaluation Highlights

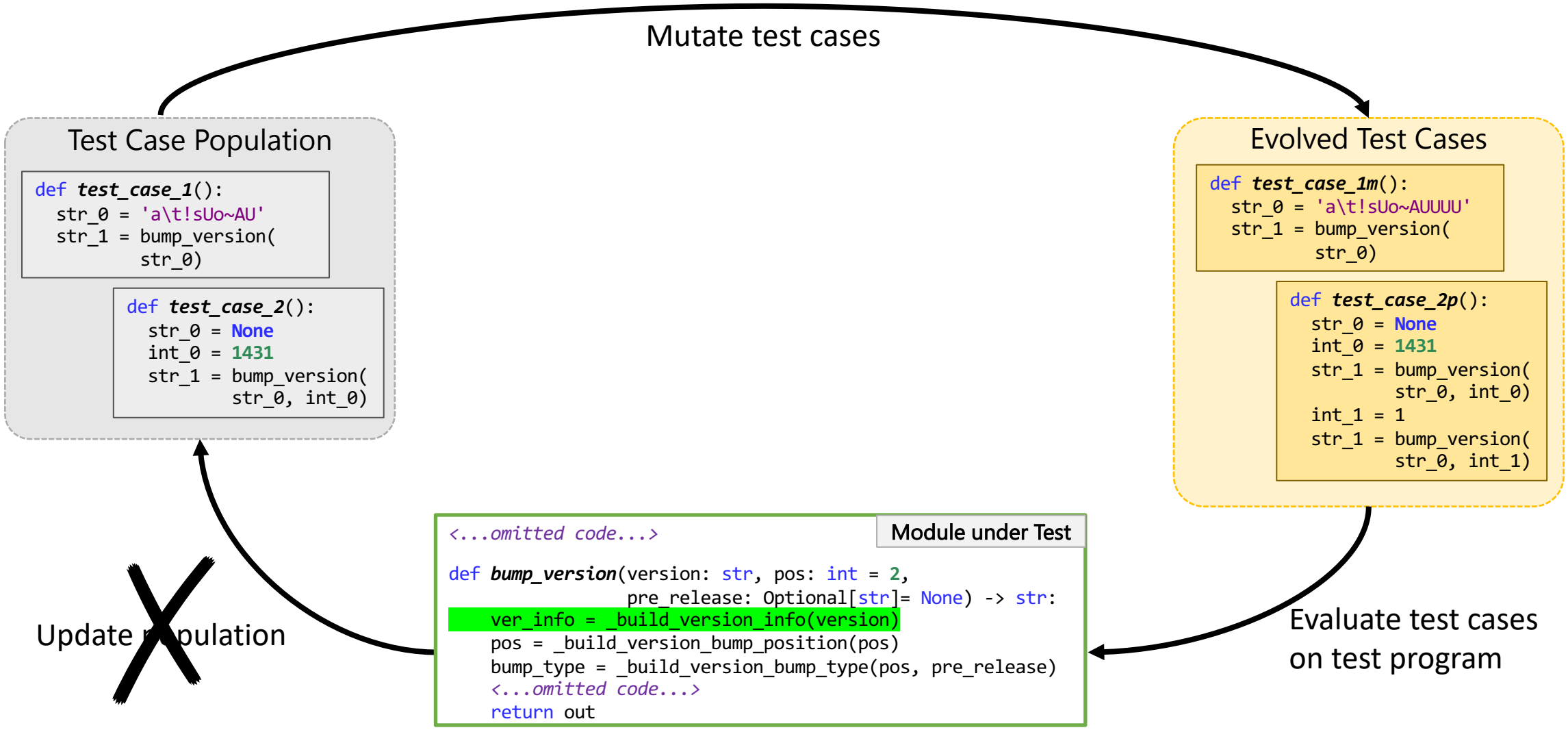
Remaining Challenges

Core Approach

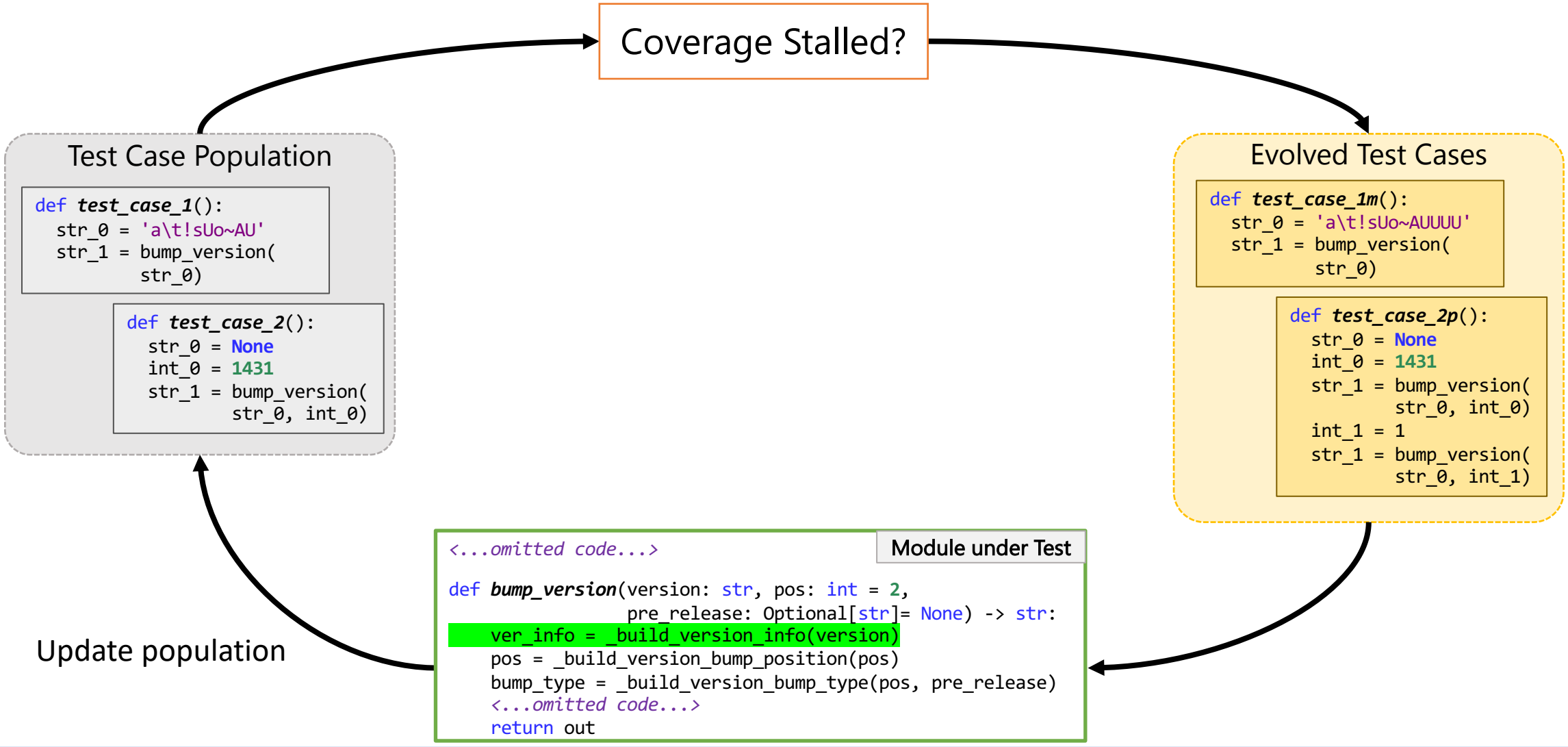
Evaluation Highlights

Remaining Challenges

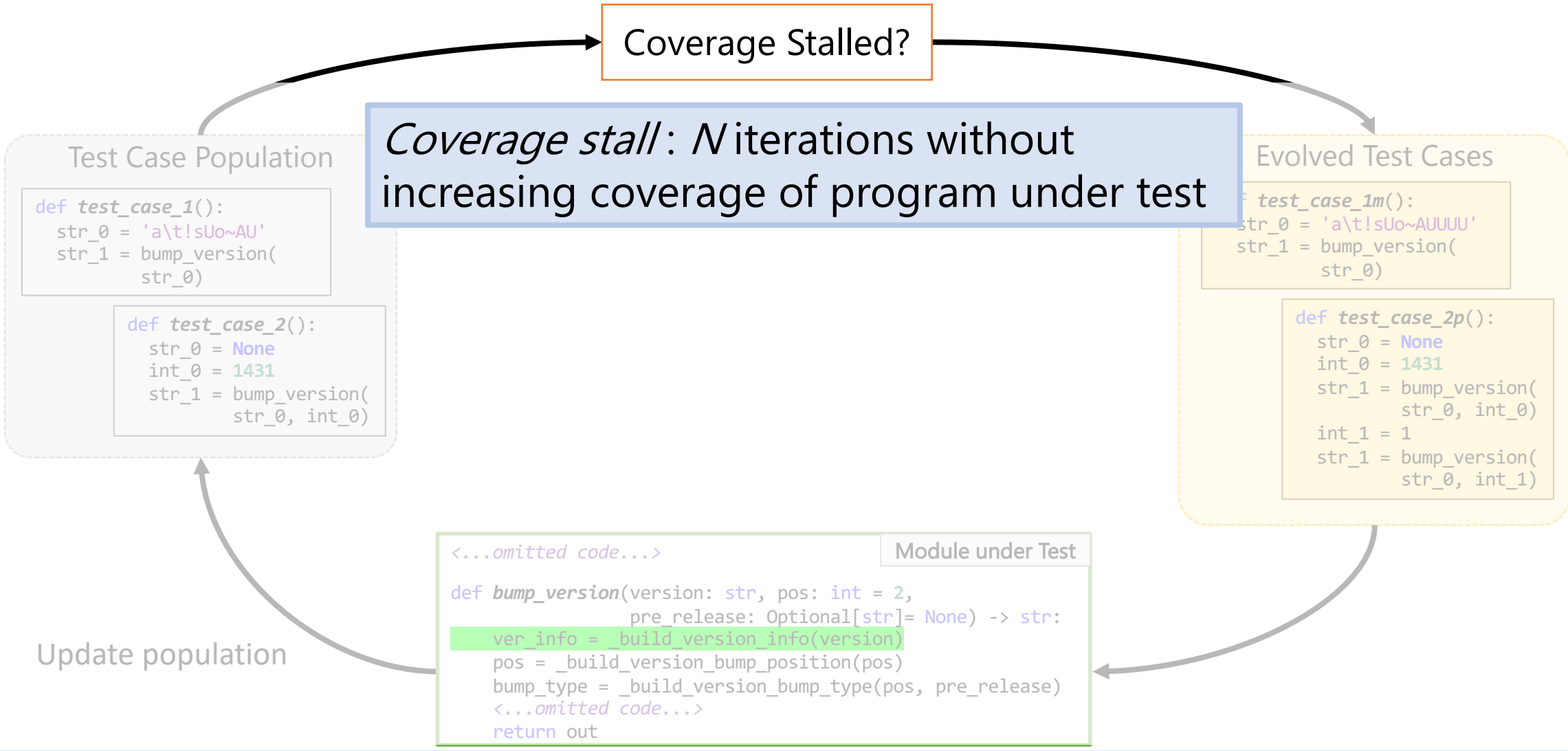
Search stalled. What to do now?



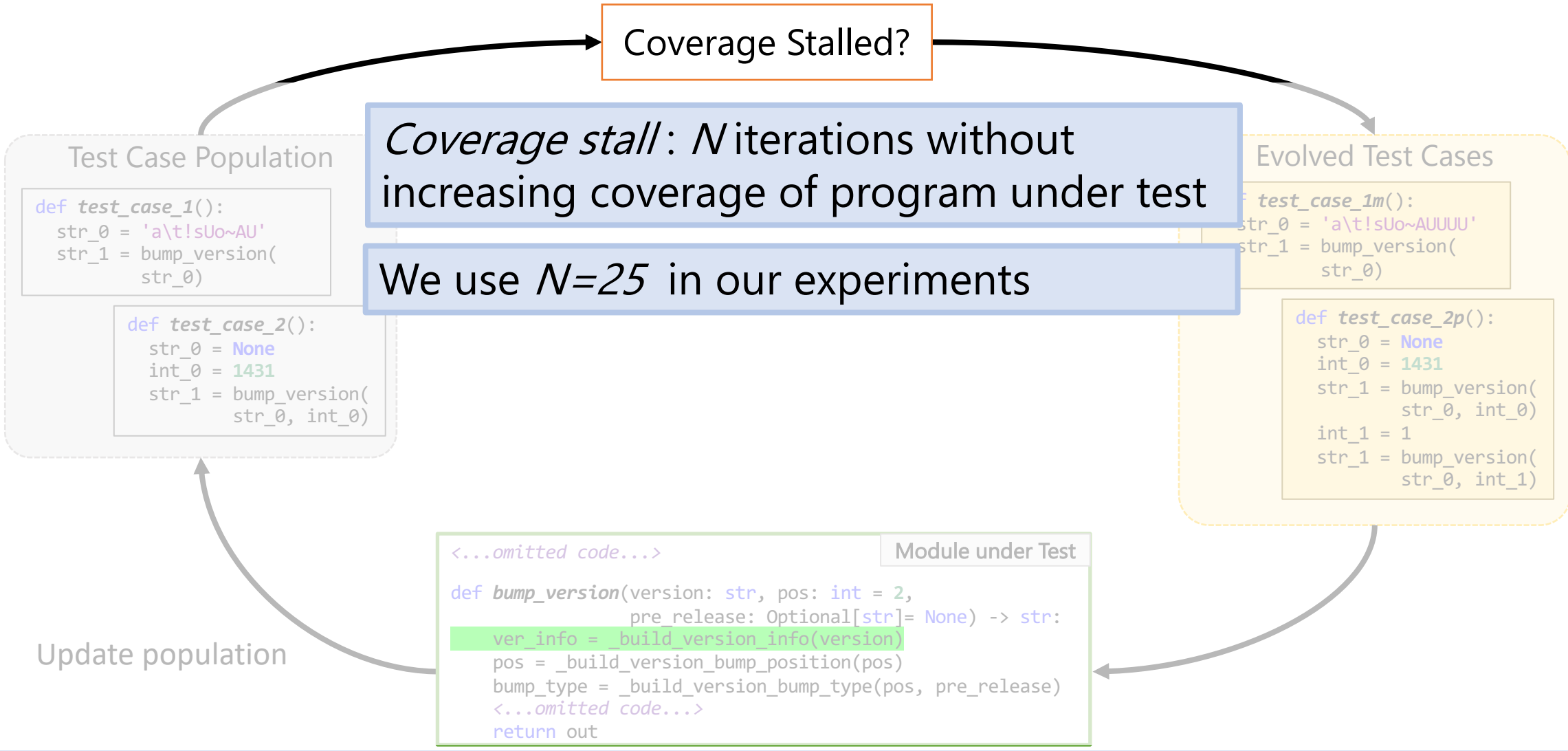
CodaMOSA: Asks for hints when stuck



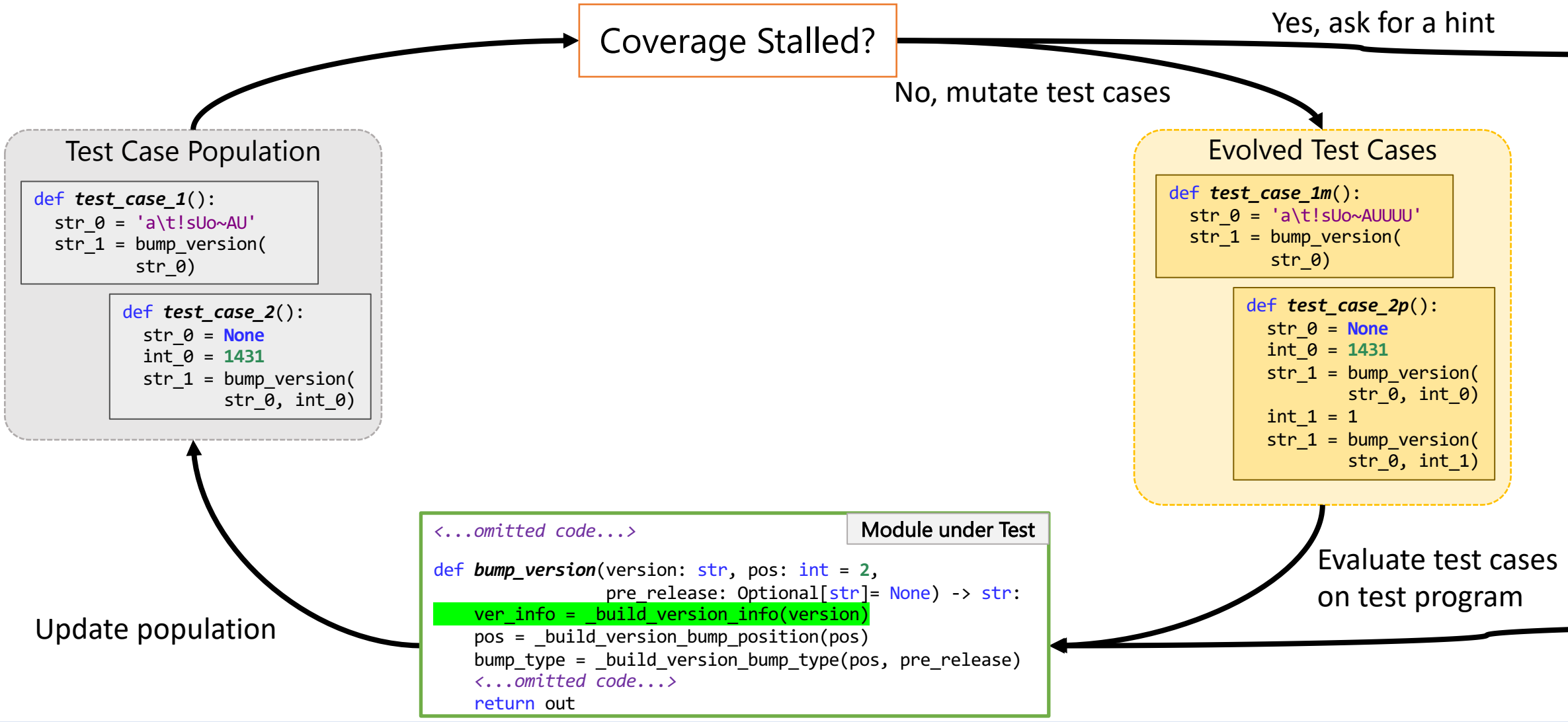
CodaMOSA: Asks for hints when stuck



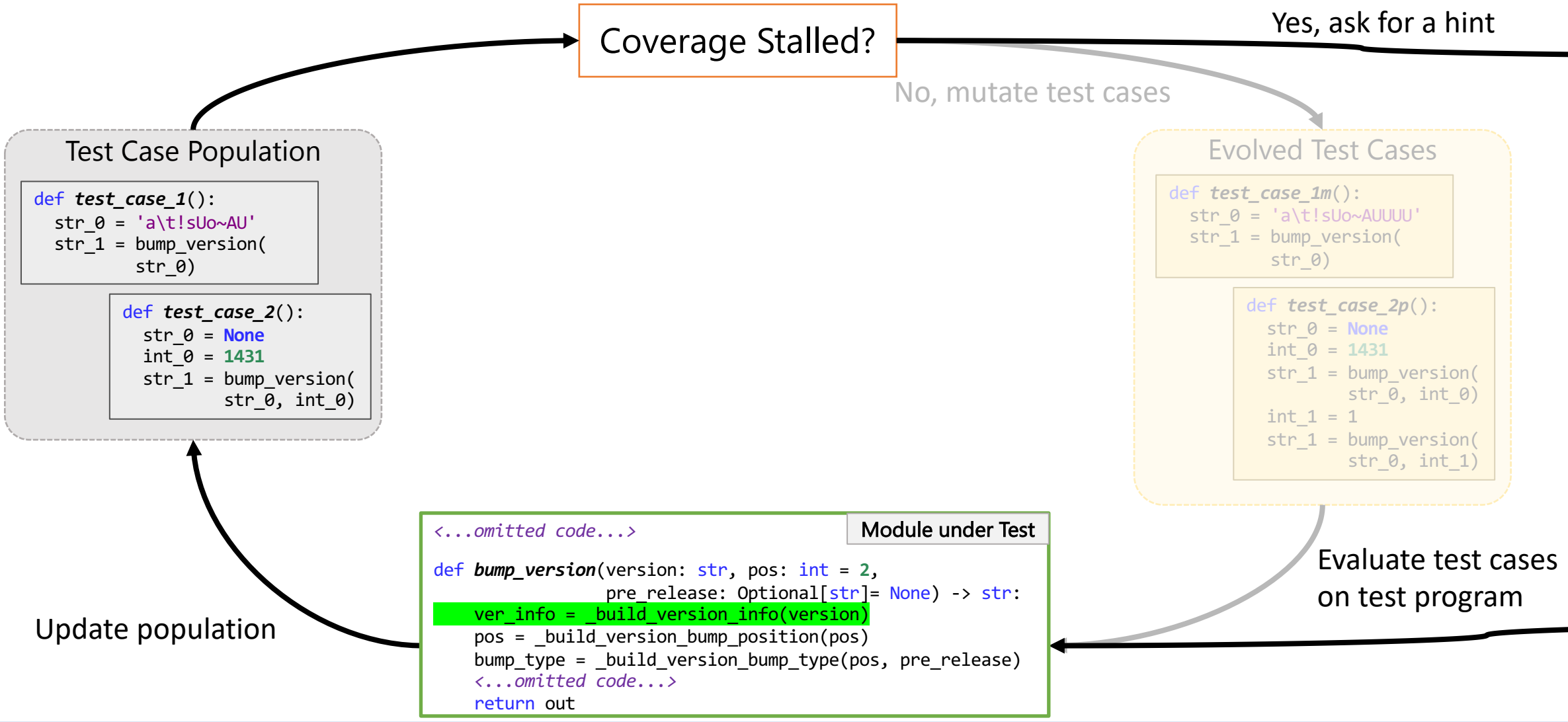
CodaMOSA: Asks for hints when stuck



CodaMOSA: Asks for hints when stuck



Search Stalled



Time to Ask for a Hint

Coverage Stalled?

Yes, ask for a hint

Prompt to create low-coverage functions tests

Ask Codex

No, mutate test cases

Test Case Population

test_case_1

test_case_2

aw Model Output ("natural" code)

Deserialize to mutatable format

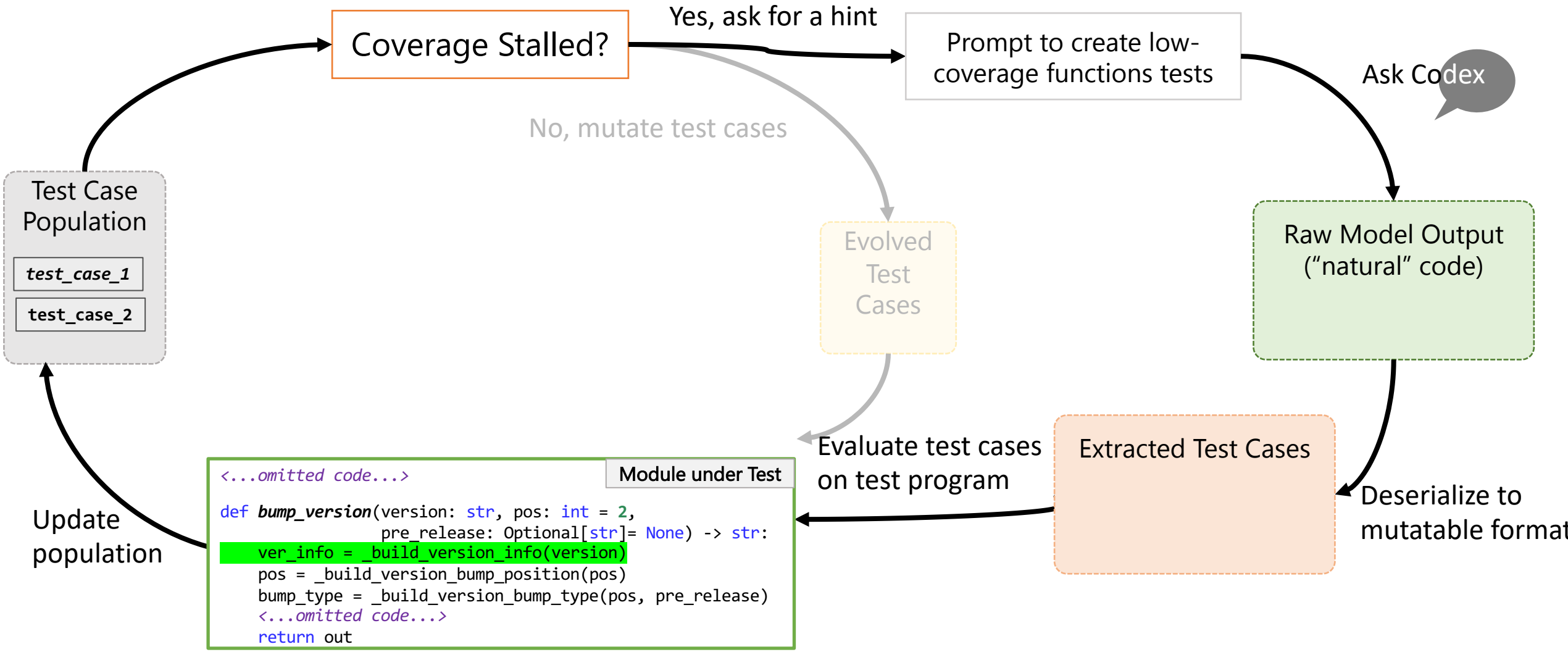
Update population

Codex, LLM trained on code
Released Aug 2021

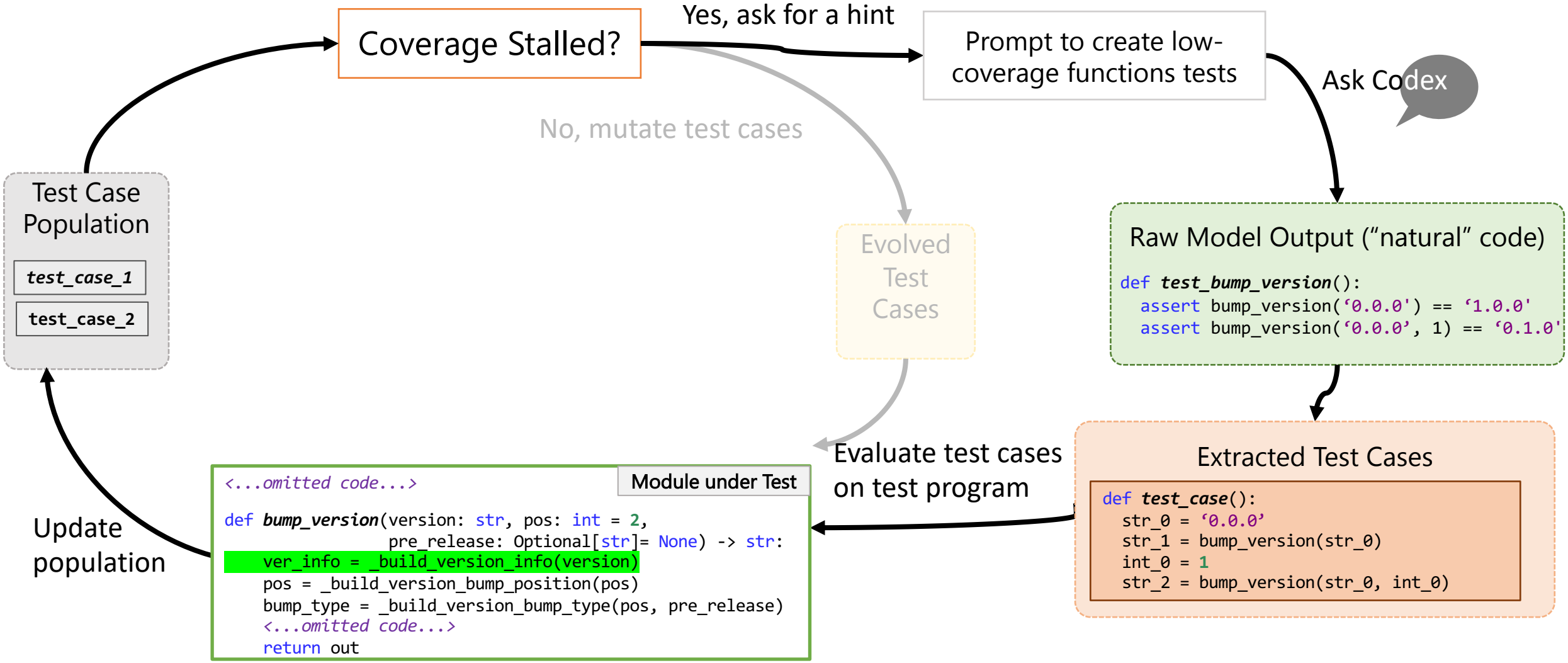

```
<...omitted code...>

def bump_version(version: str, pos: int = 2,
                 pre_release: Optional[str]= None) -> s
    ver_info = _build_version_info(version)
    pos = _build_version_bump_position(pos)
    bump_type = _build_version_bump_type(pos, pre_release)
    <...omitted code...>
    return out
```

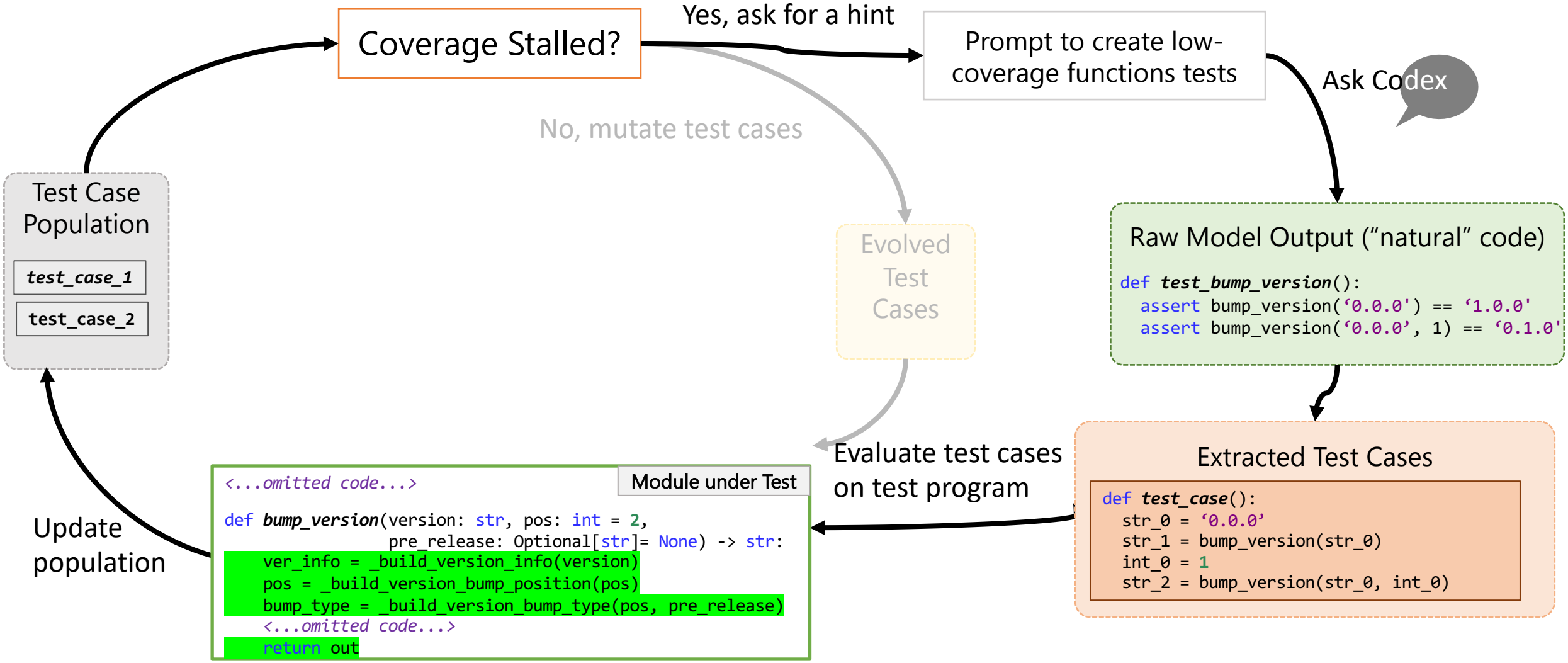
Time to Ask for a Hint



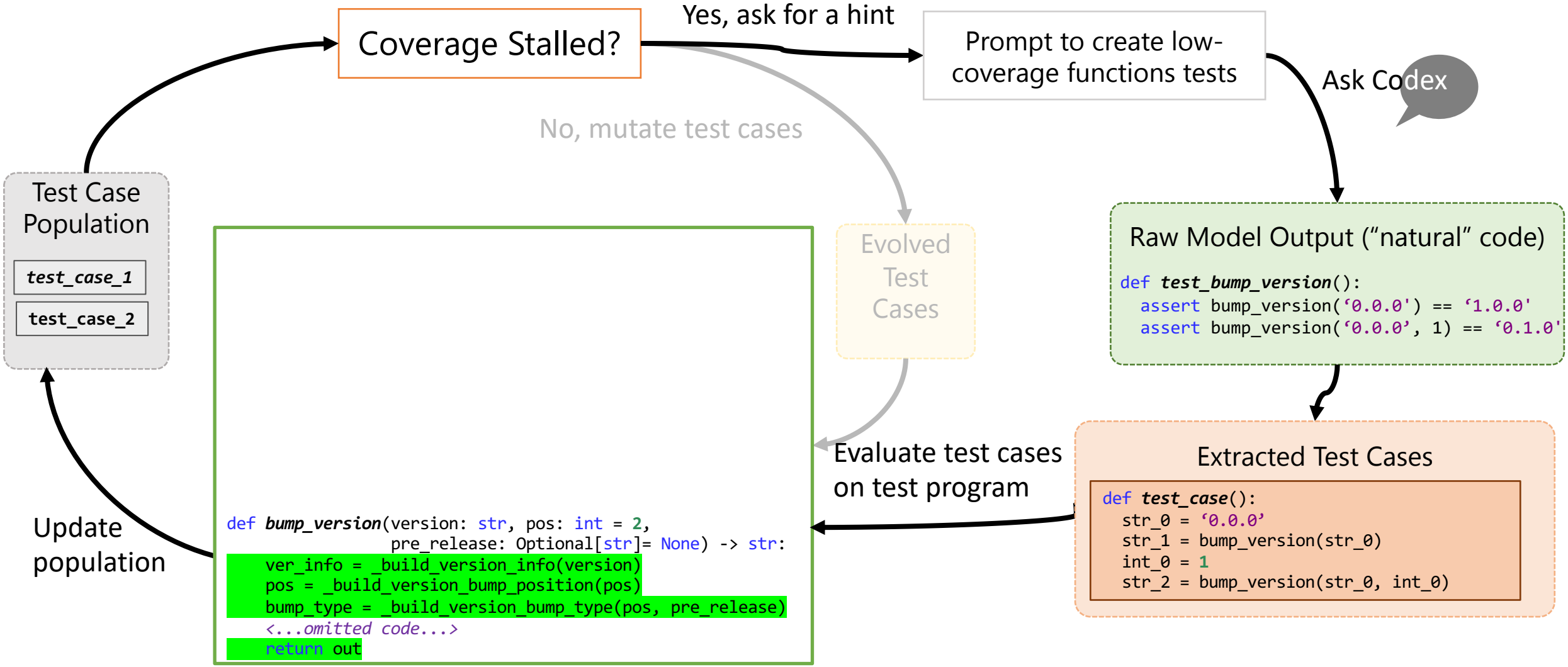
Time to Ask for a Hint



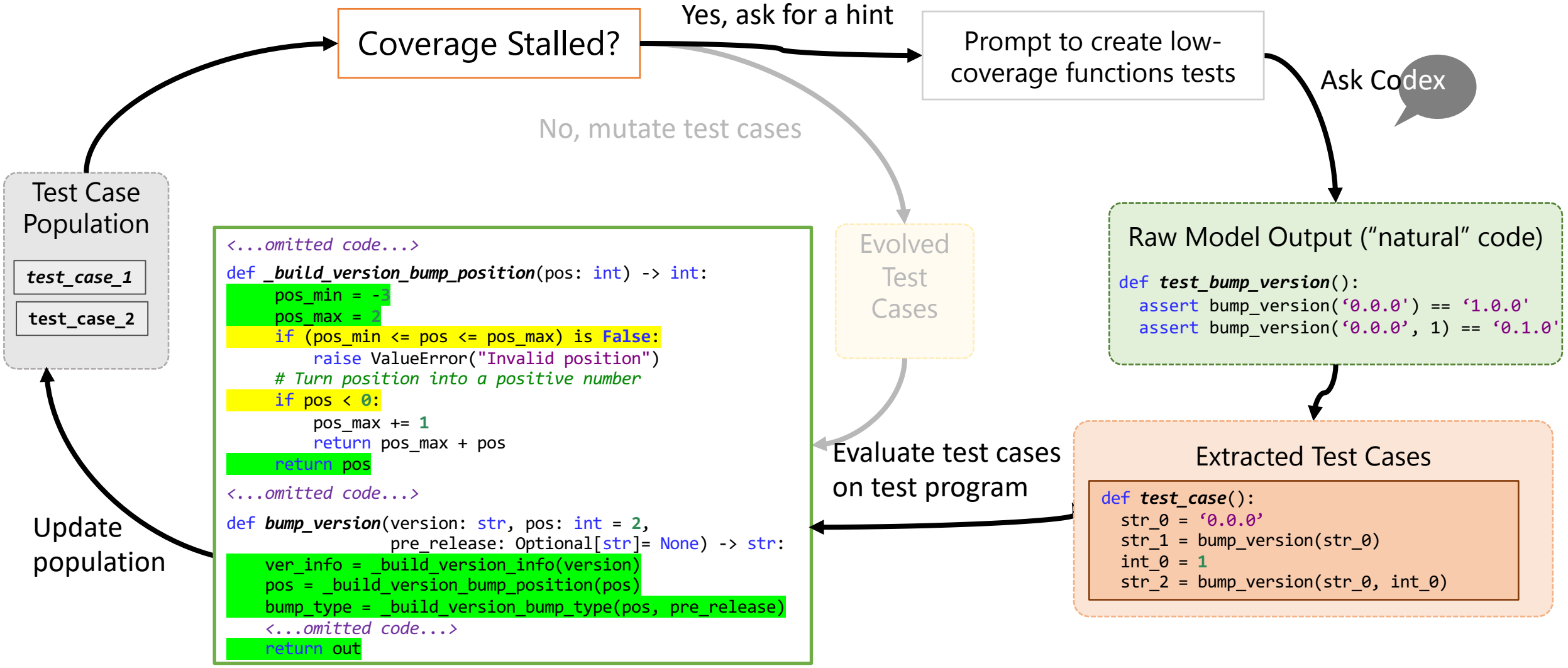
Suggested Test Case Increases Coverage



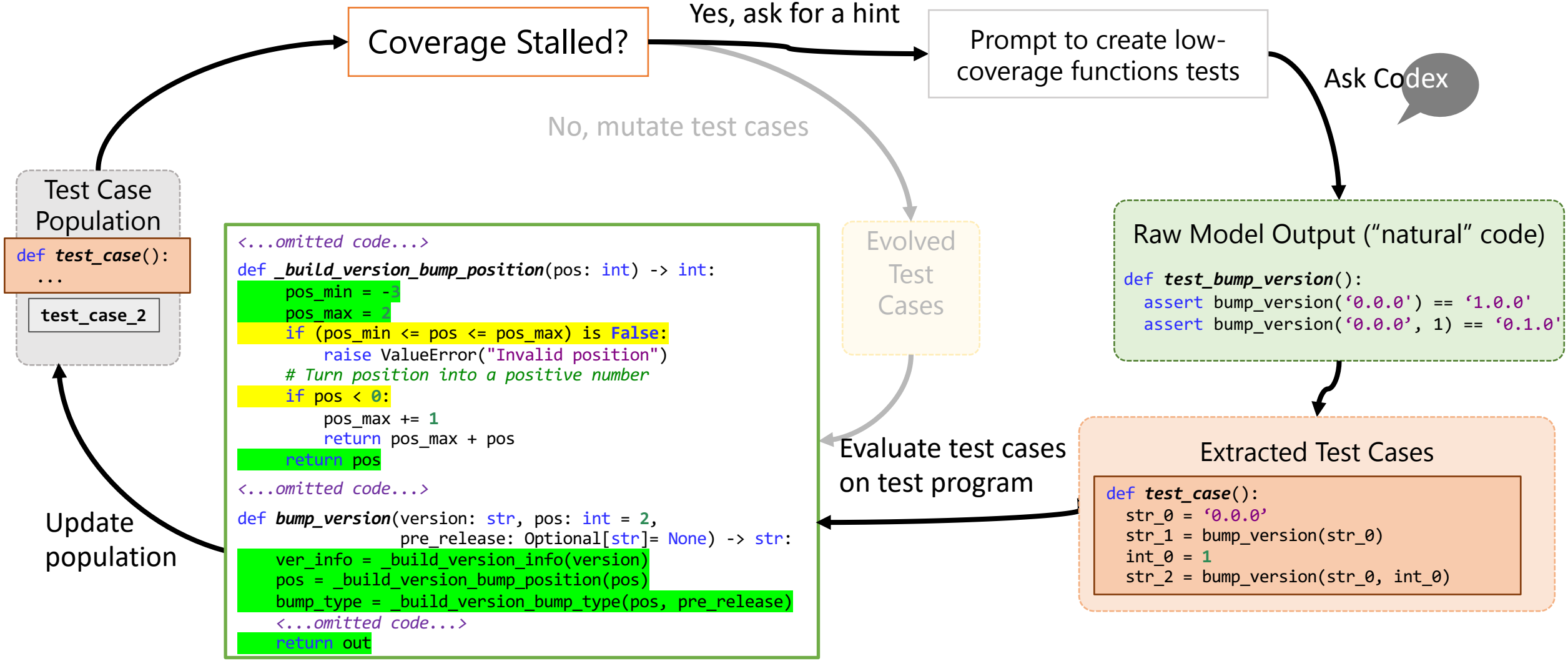
Suggested Test Case Increases Coverage



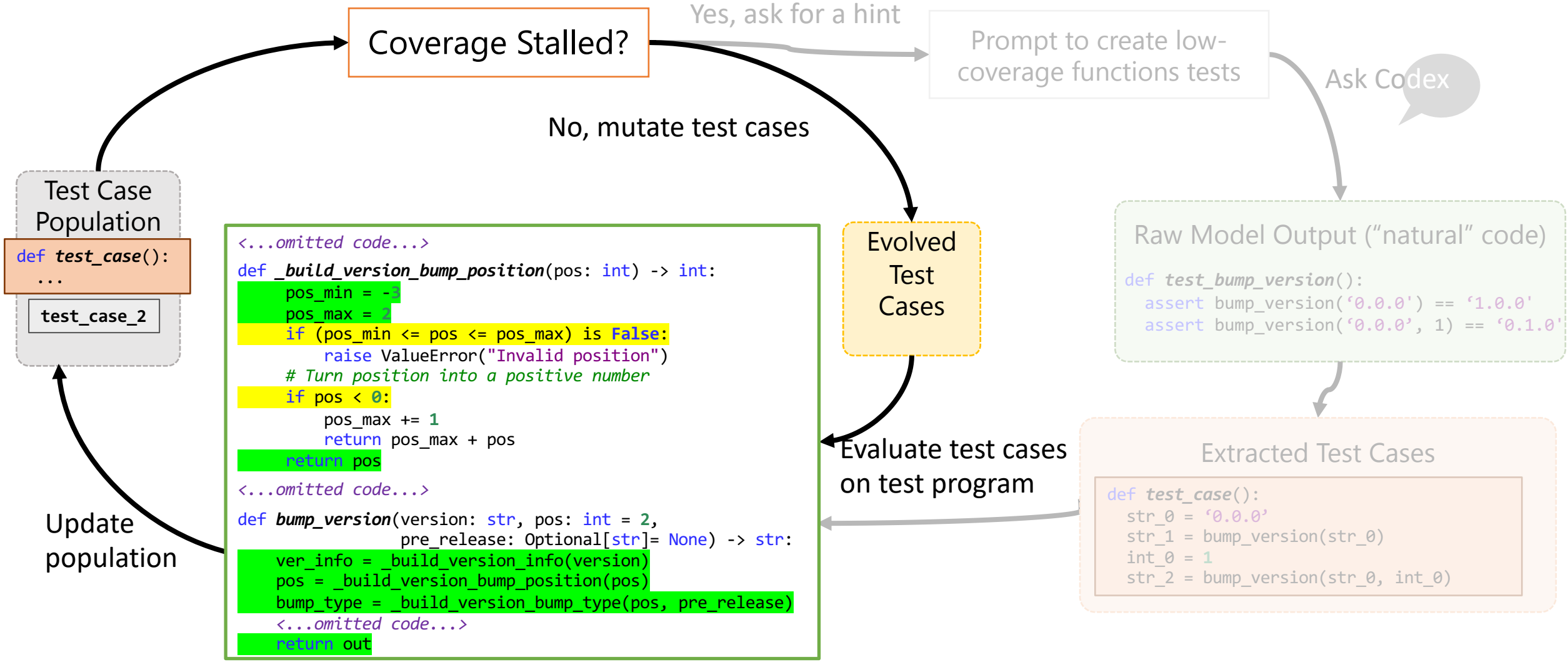
Suggested Test Case Increases Coverage



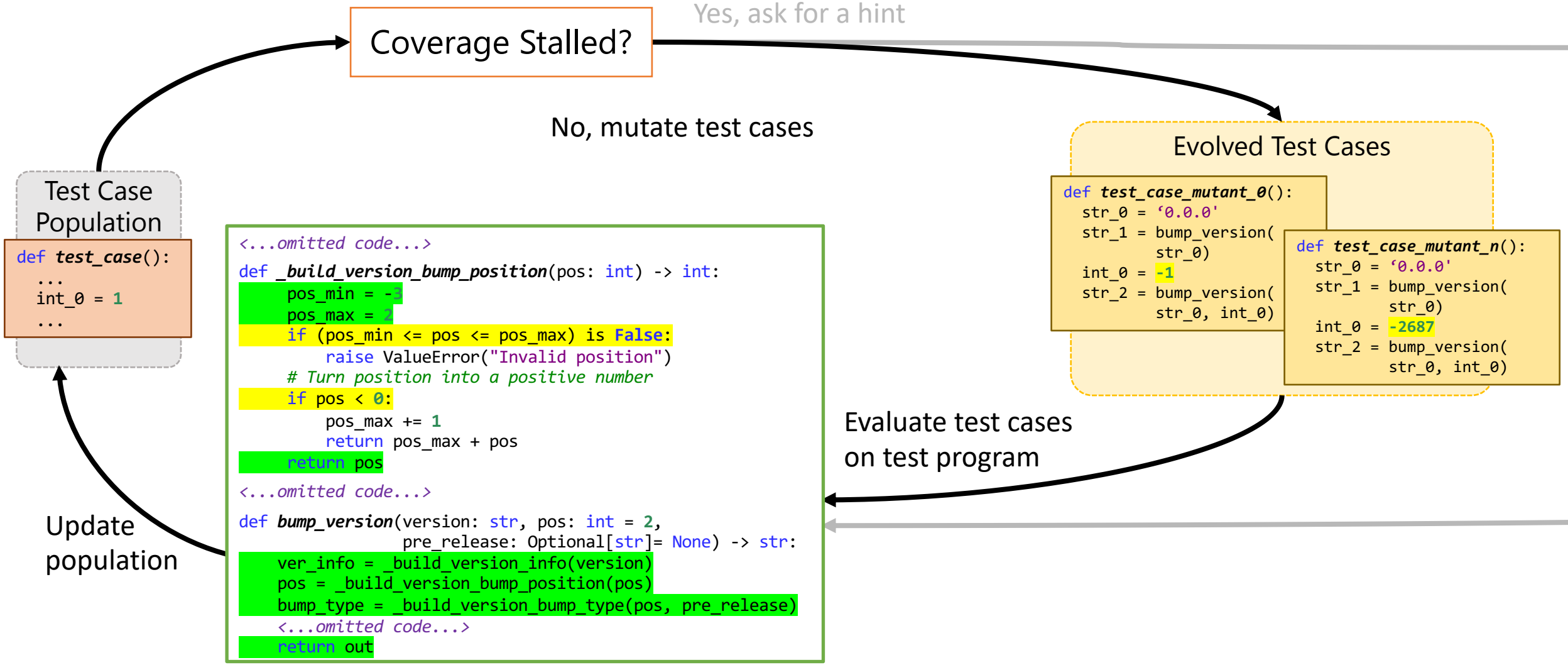
Update Population



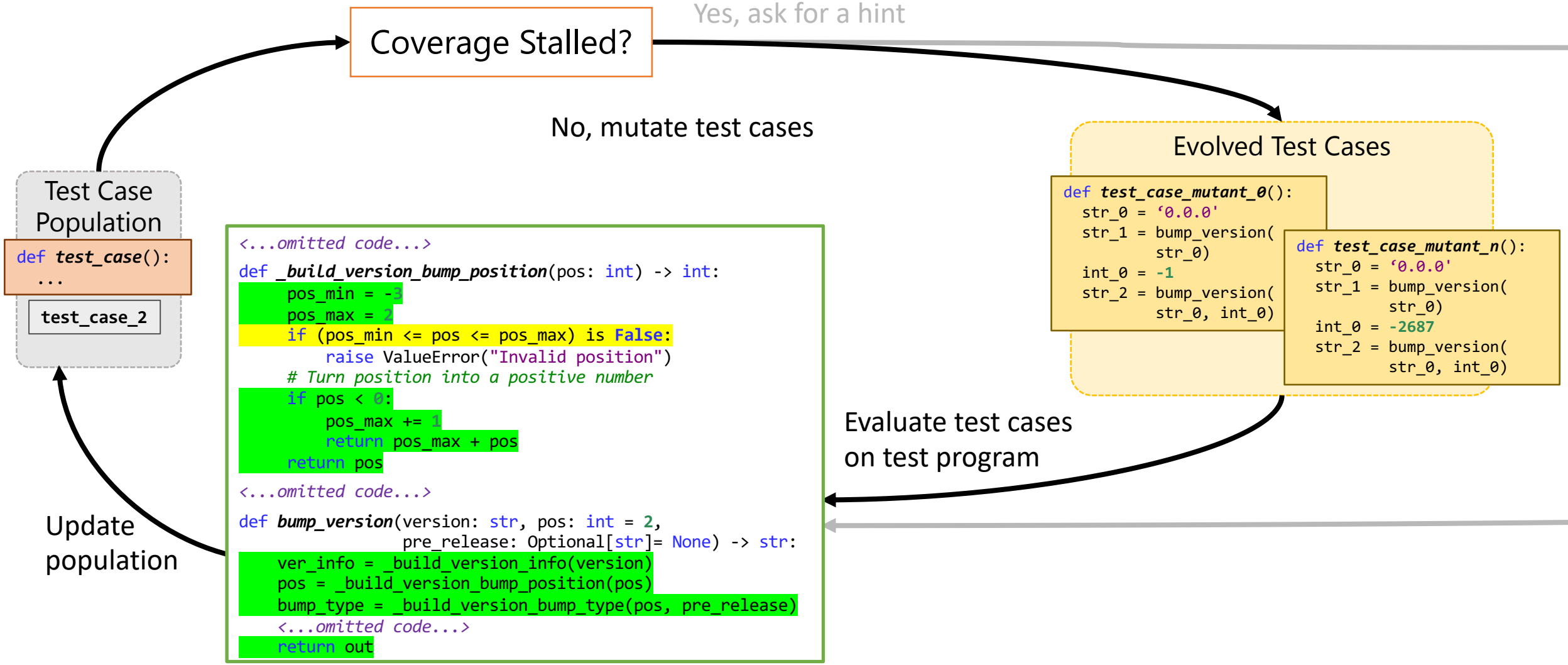
Search No Longer Stalled



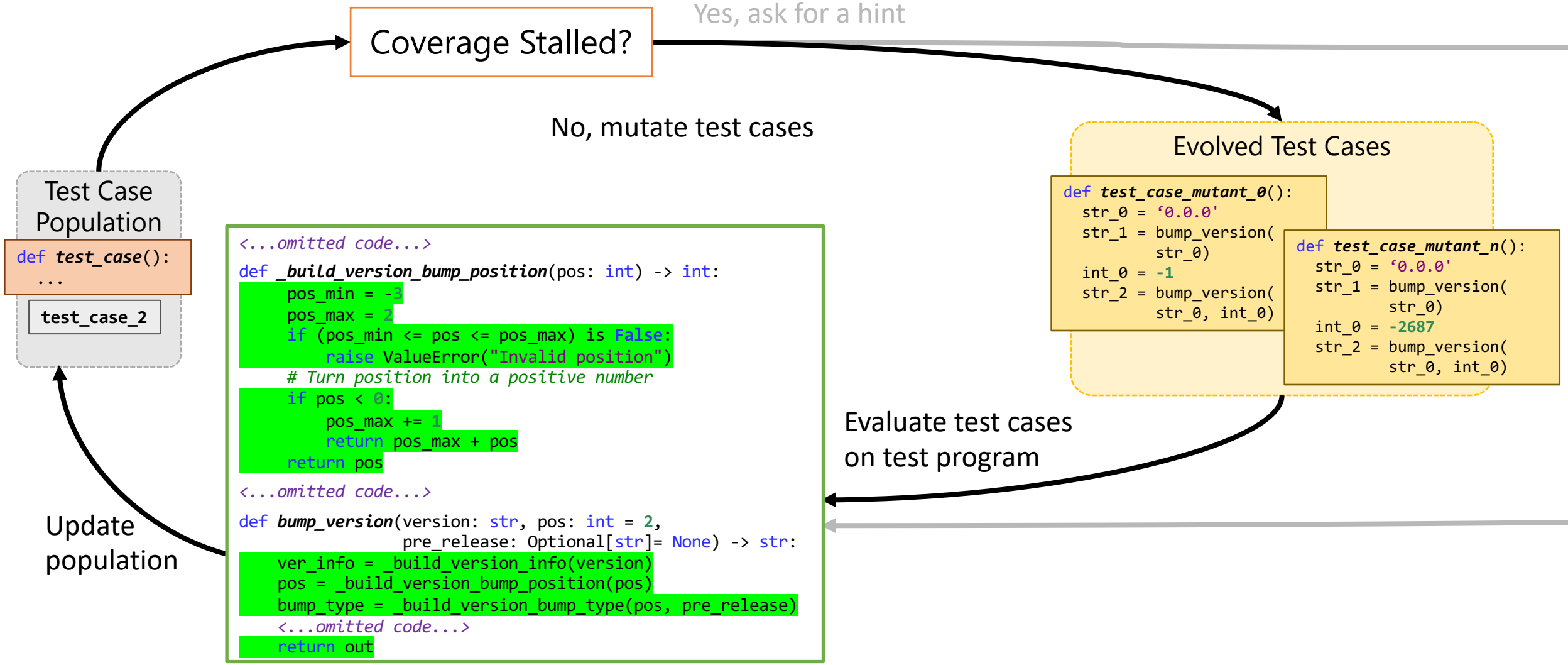
Search No Longer Stalled



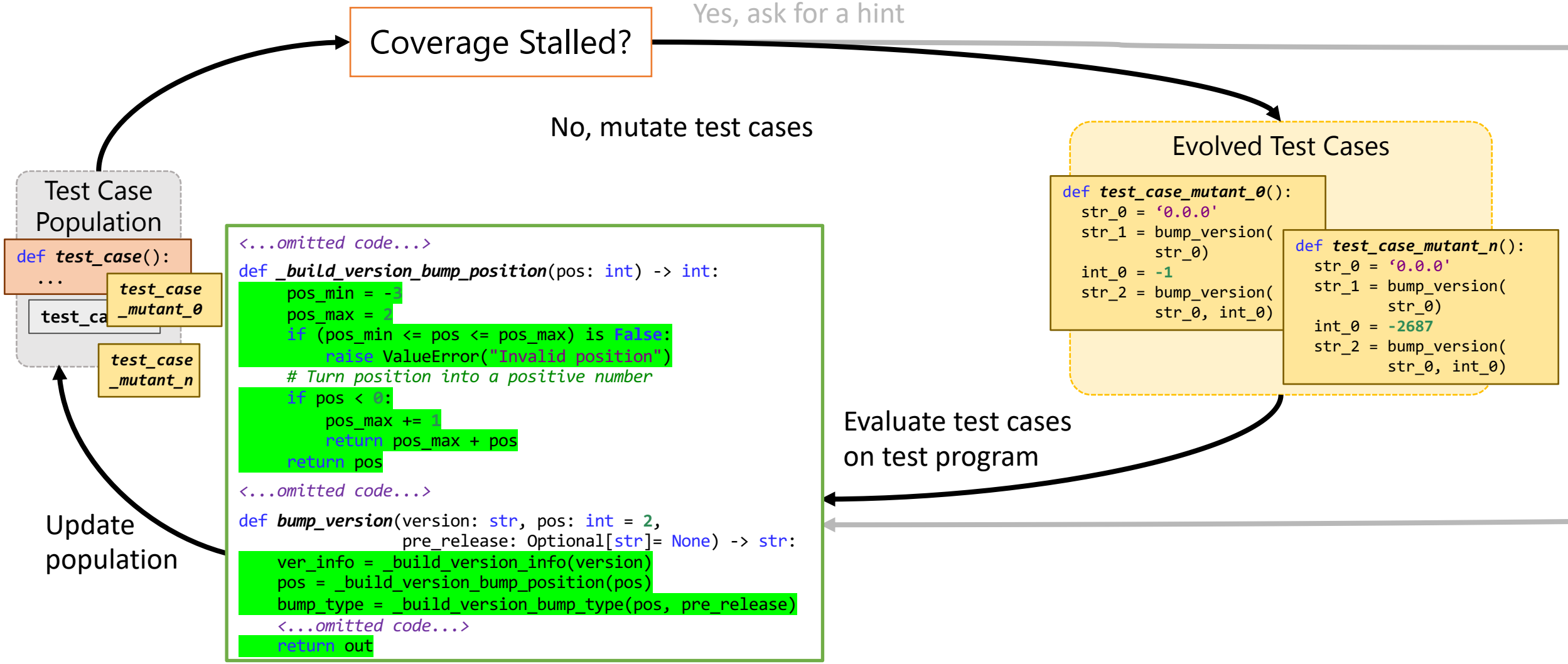
Search No Longer Stalled



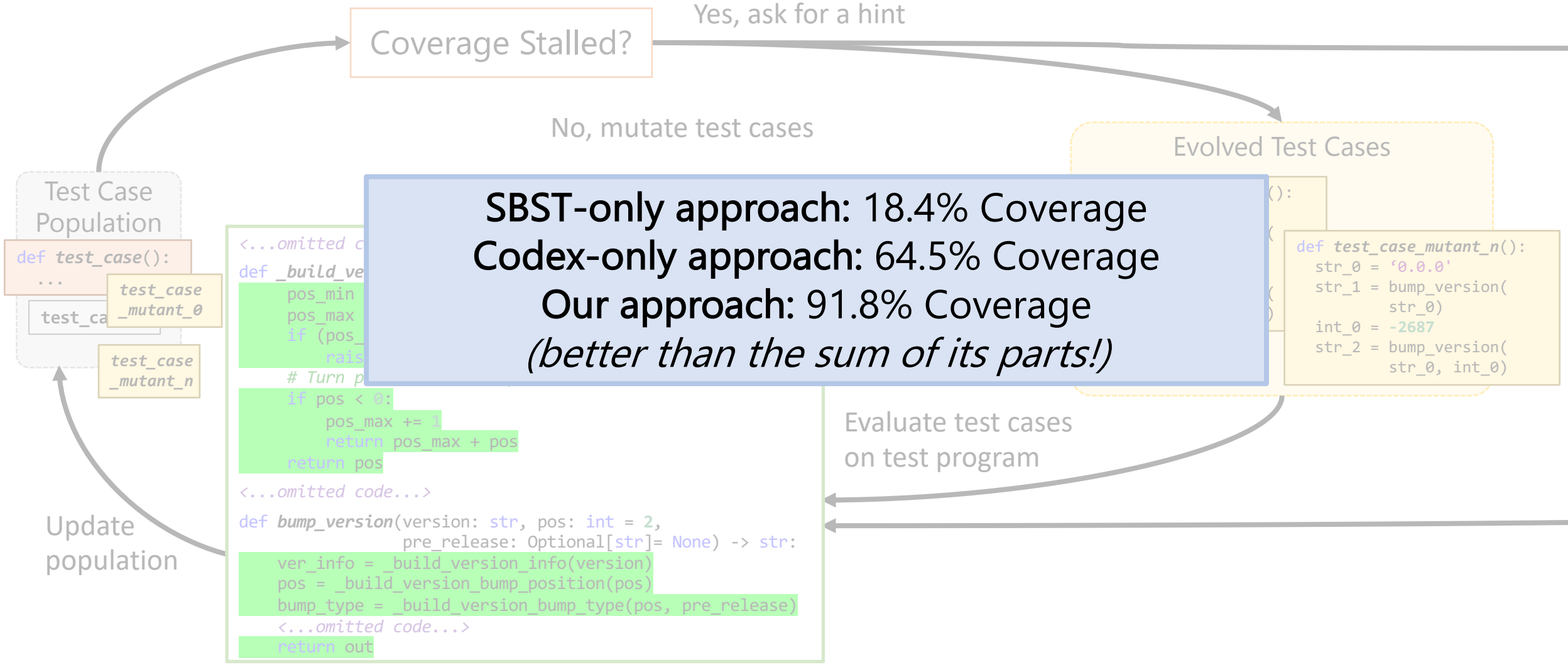
Search No Longer Stalled



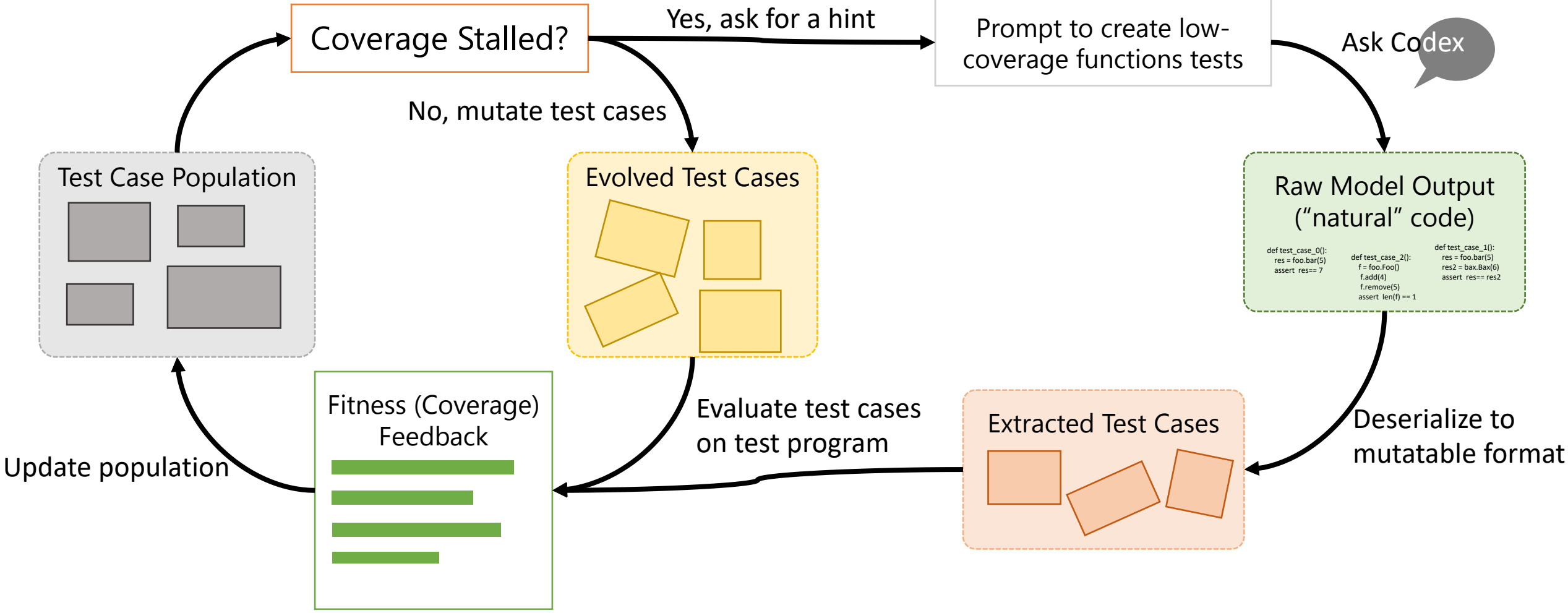
Search No Longer Stalled



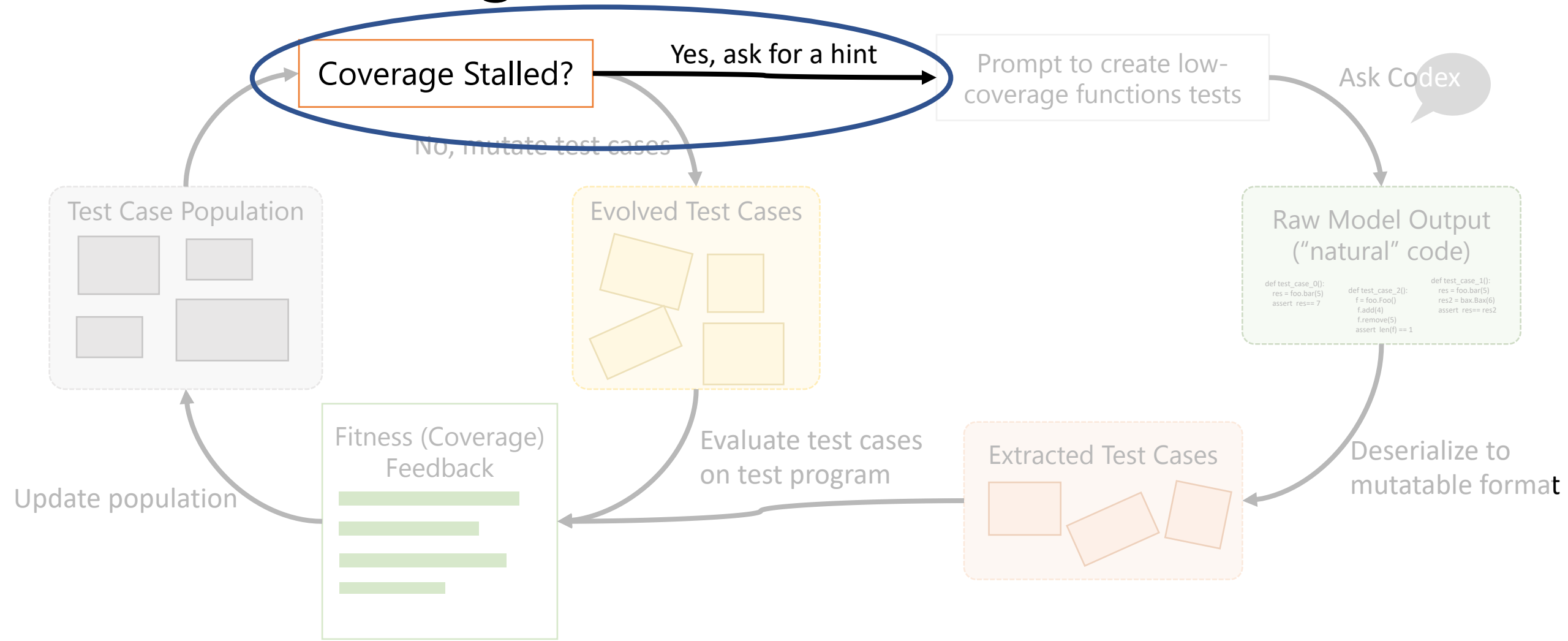
Spoiler: Results on this Benchmark



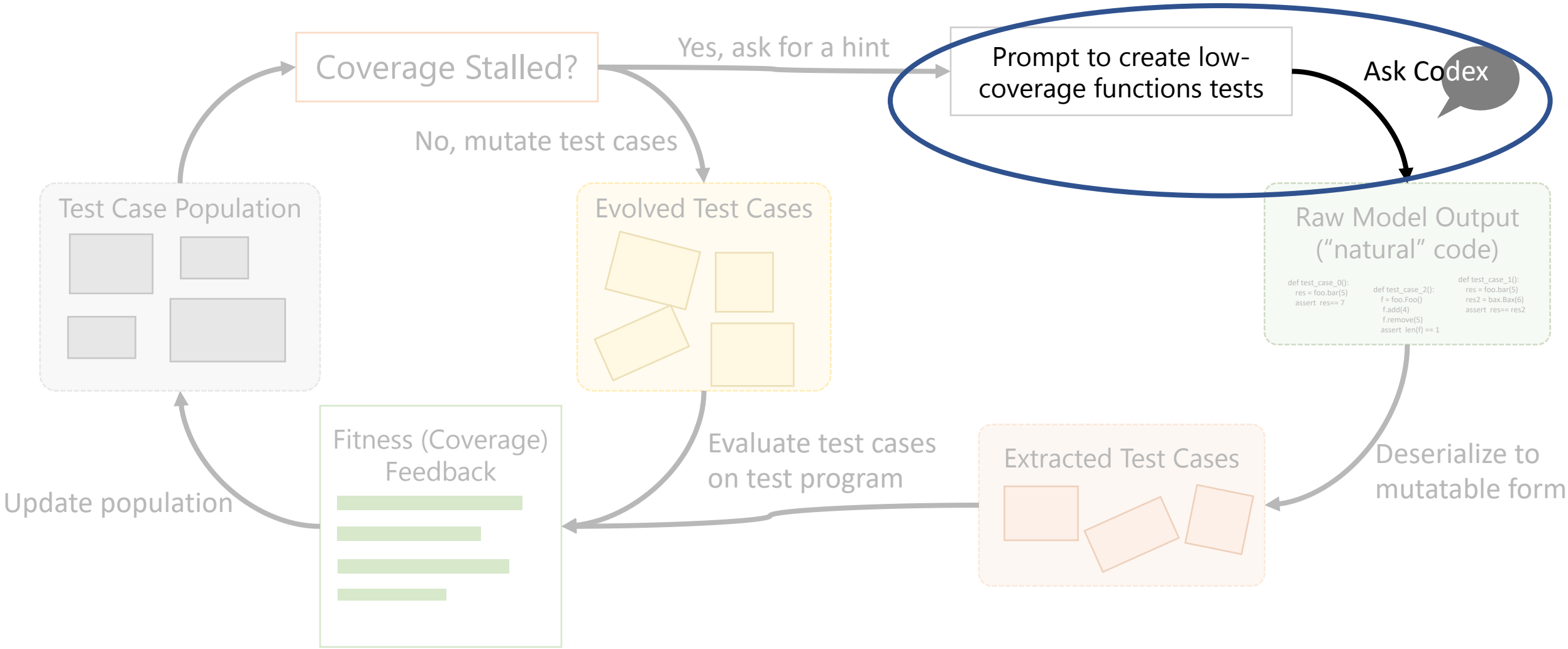
CodaMOSA Approach



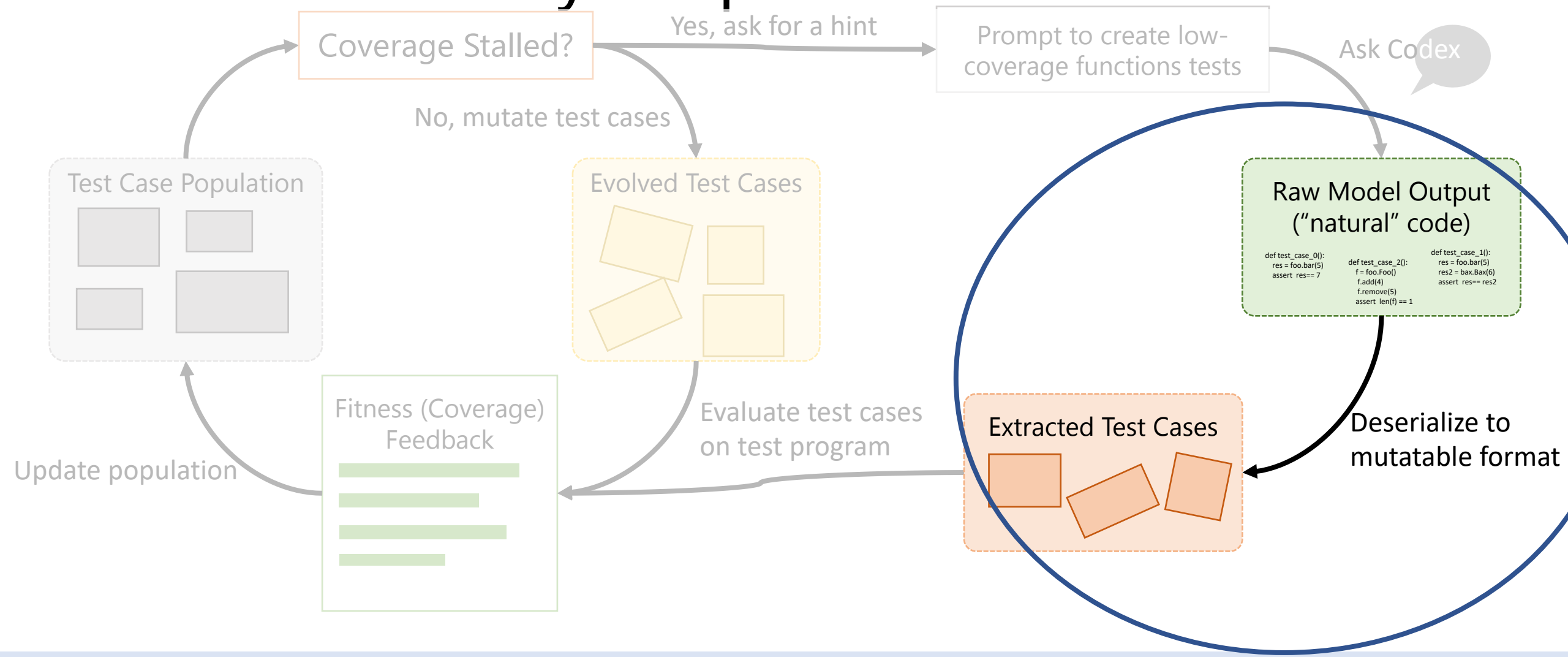
Challenge: When to ask for a hint?



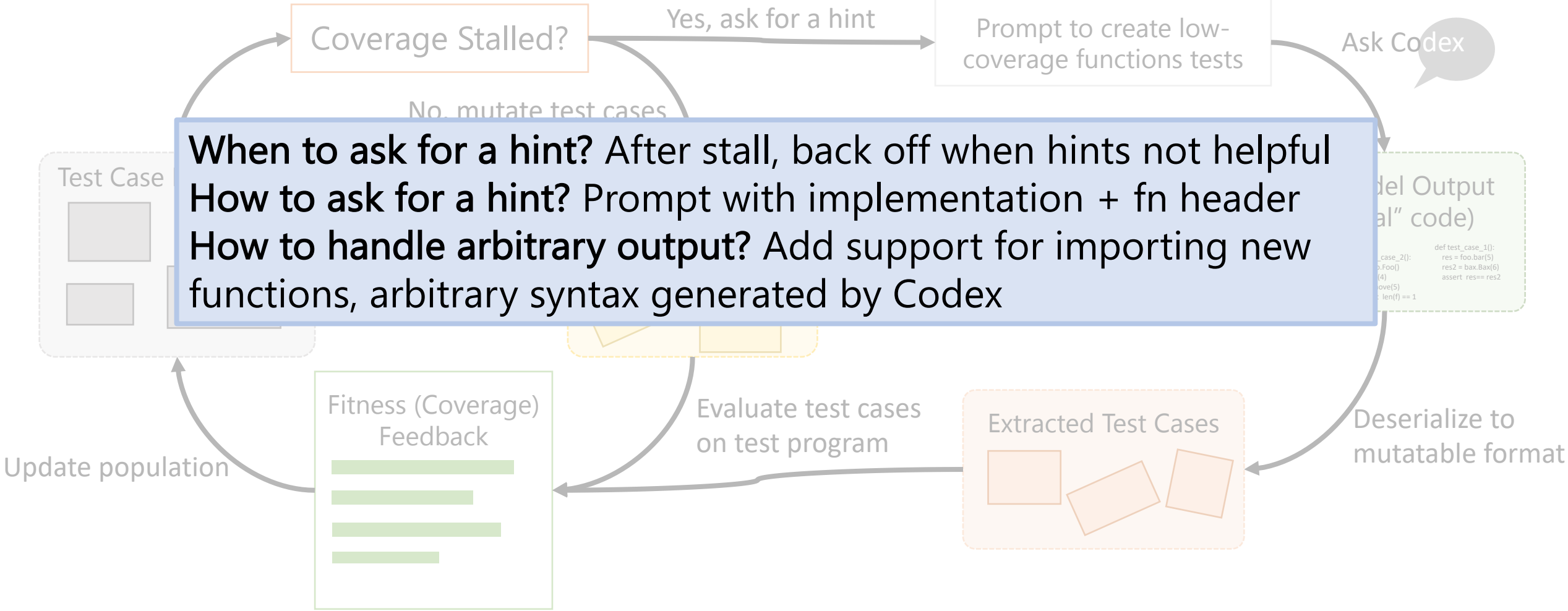
Challenge: How to ask for a hint?



Challenge: How to handle (potentially) arbitrary output from Codex?



Solutions Discussed Further in Paper



When to ask for a hint? After stall, back off when hints not helpful
How to ask for a hint? Prompt with implementation + fn header
How to handle arbitrary output? Add support for importing new functions, arbitrary syntax generated by Codex

Core Approach

Evaluation Highlights

Remaining Challenges

Core Approach

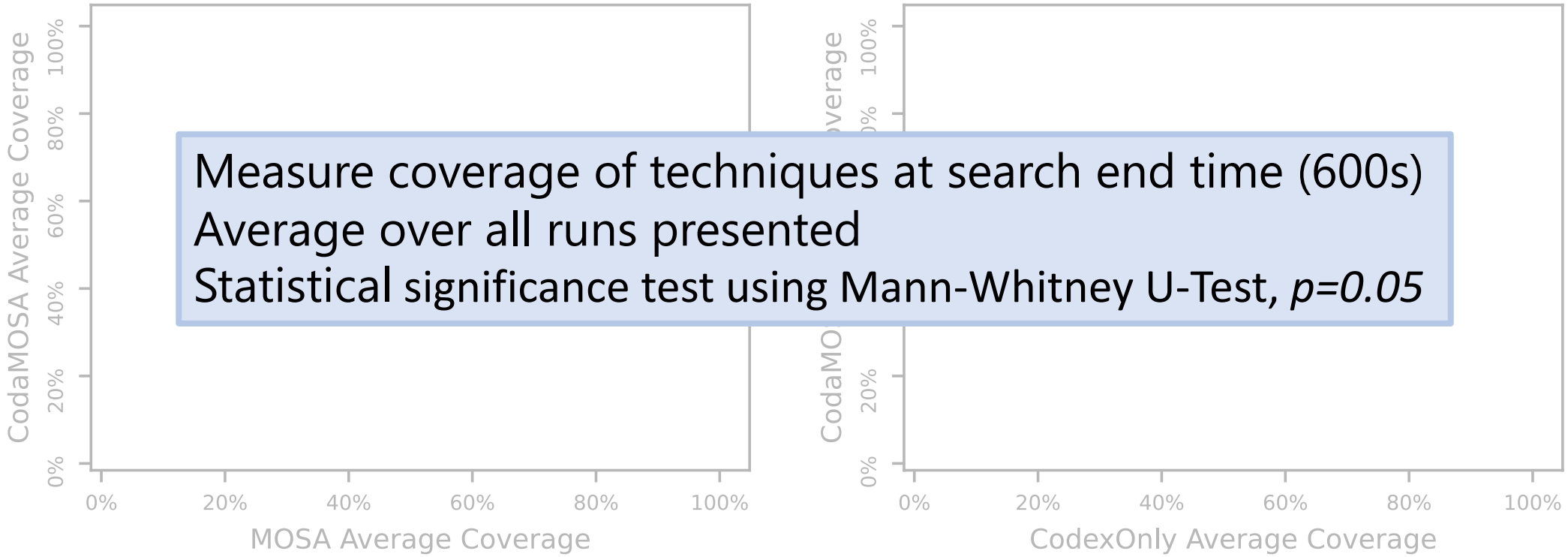
Evaluation Highlights

Remaining Challenges

Evaluation Setup

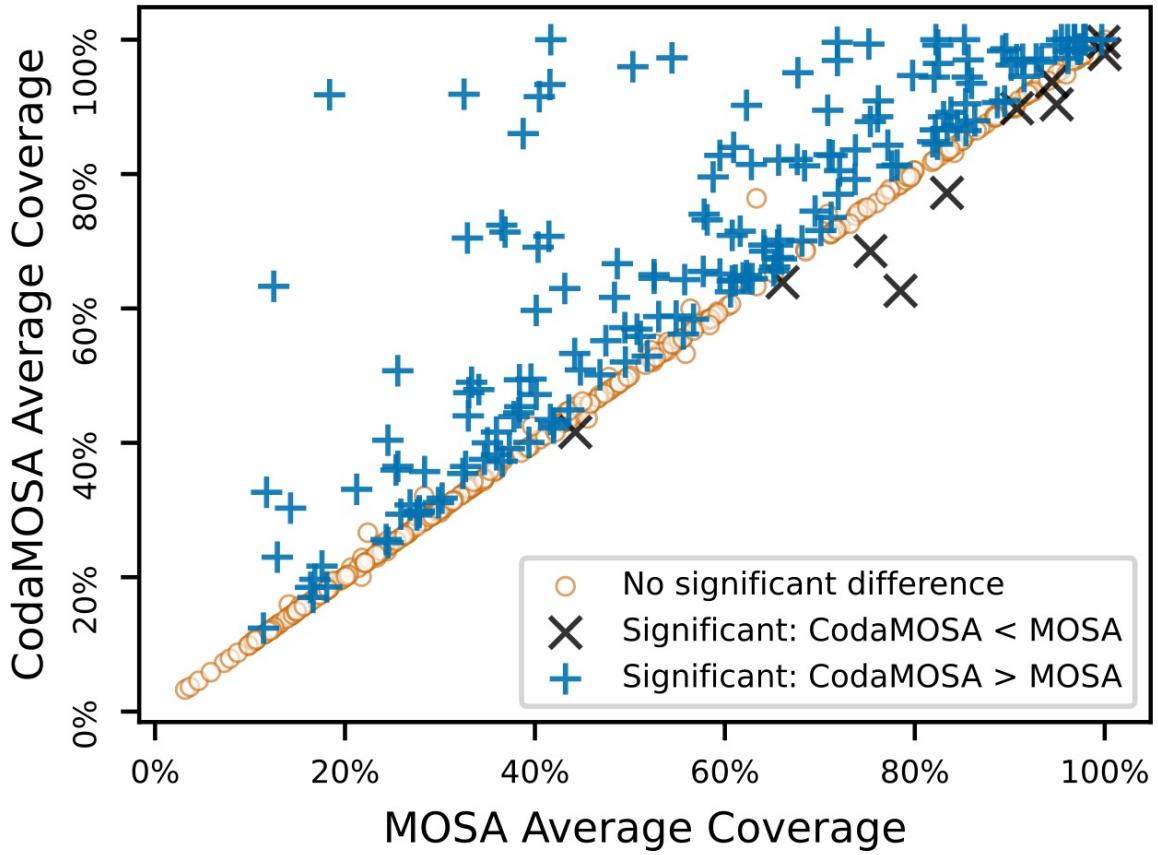
- 486 Modules from 27 Python Projects
- Run each technique 16 times, 600s each
- Compare to baselines:
  - MOSA (no Codex hints, Pynguin Implementation)
  - CodexOnly (no mutative search)
- Paper: evaluate CodaMOSA design decisions

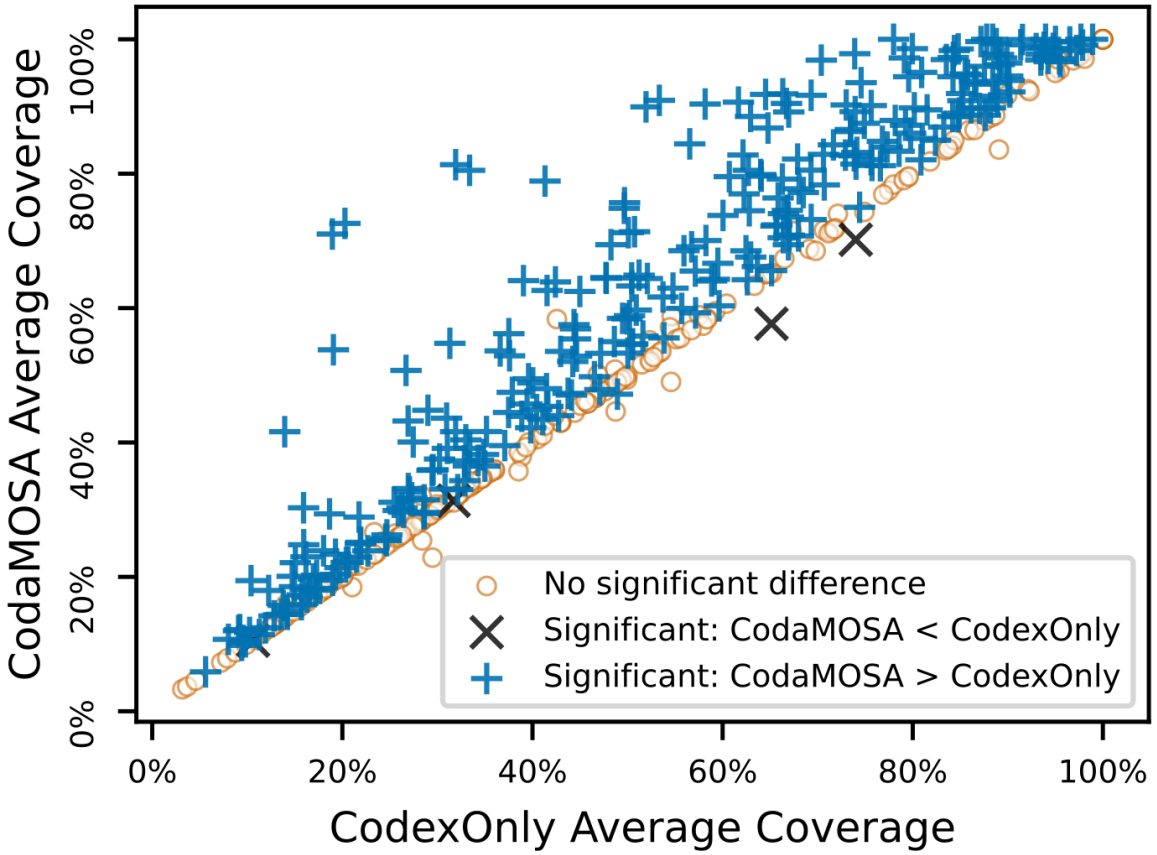
Final Coverage Comparison



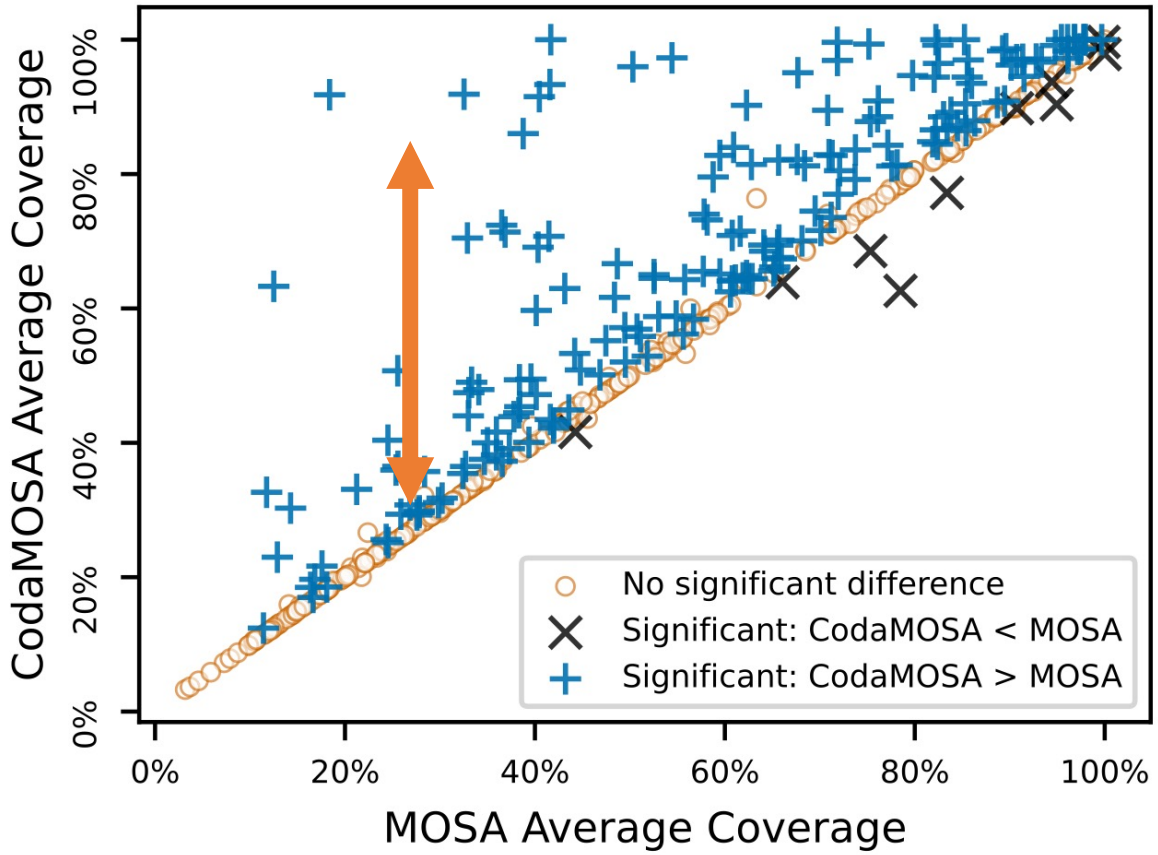
Measure coverage of techniques at search end time (600s)
Average over all runs presented
Statistical significance test using Mann-Whitney U-Test, *p=0.05*

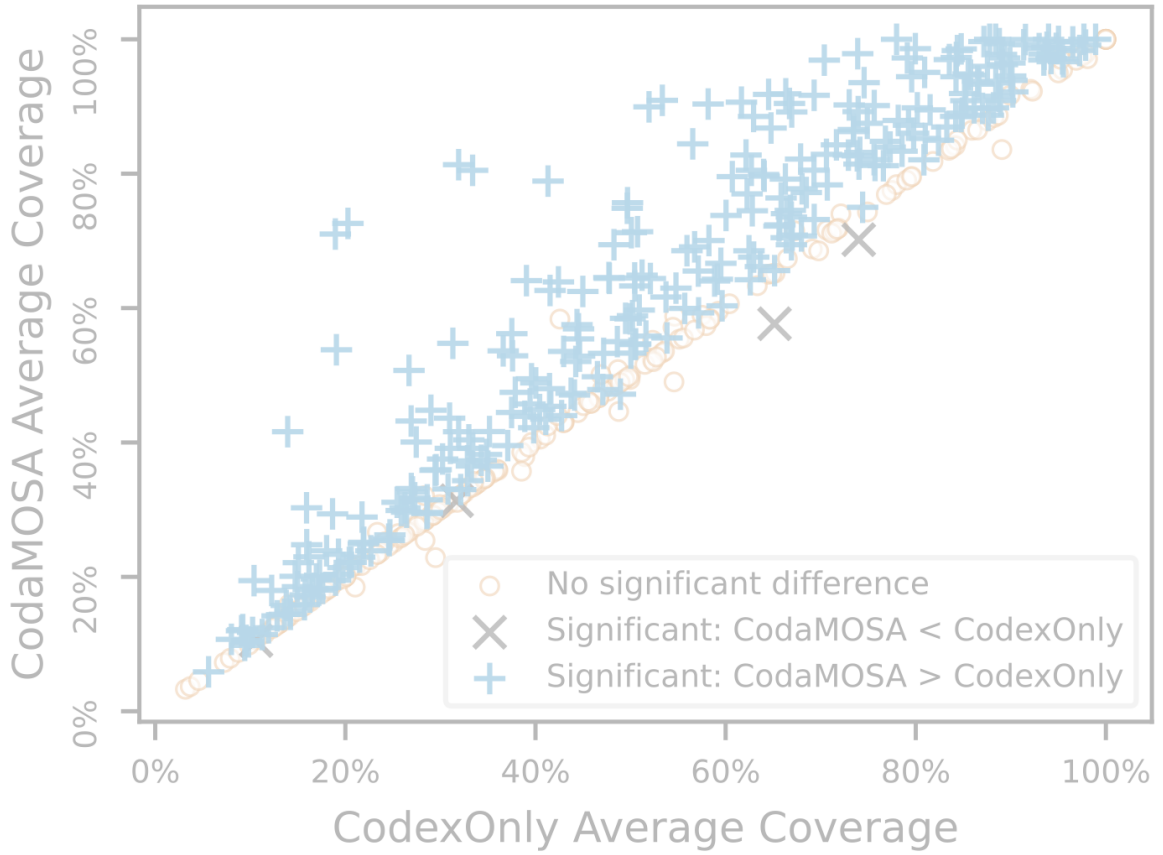
CodaMOSA Outperforms Baselines



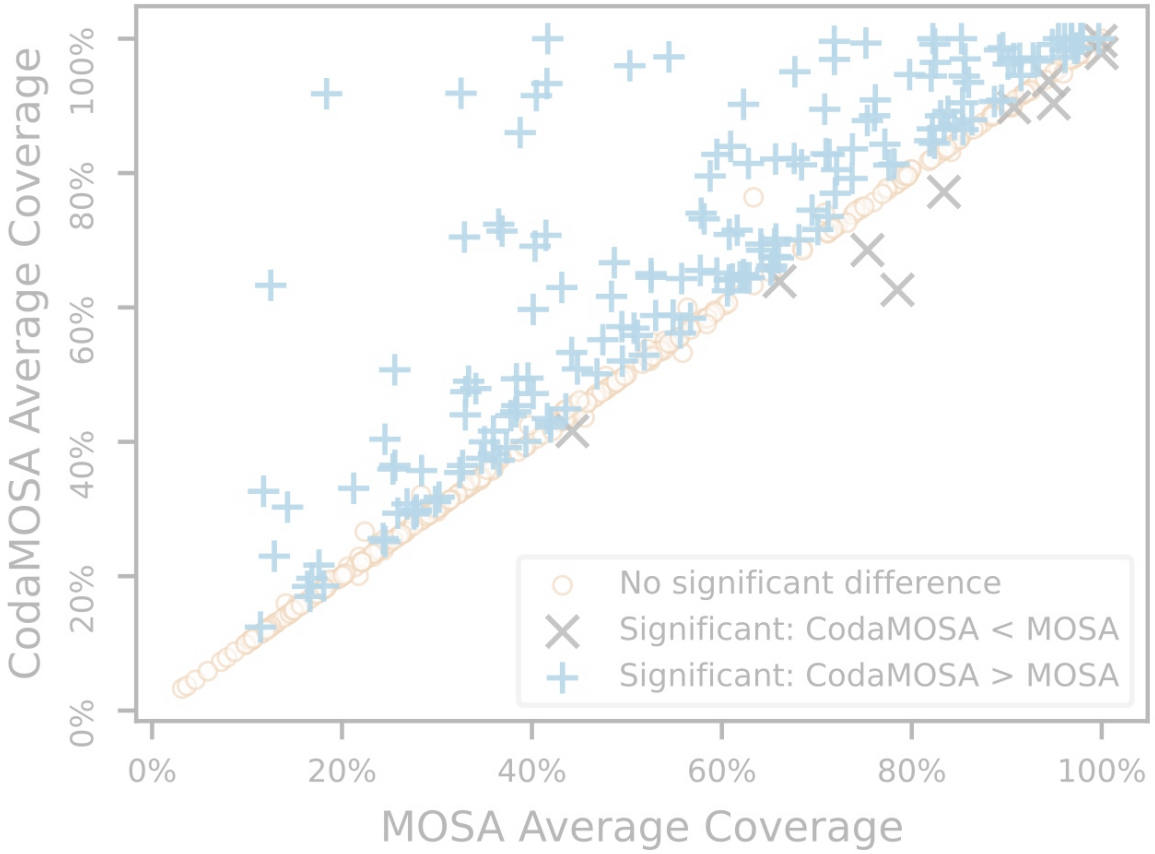


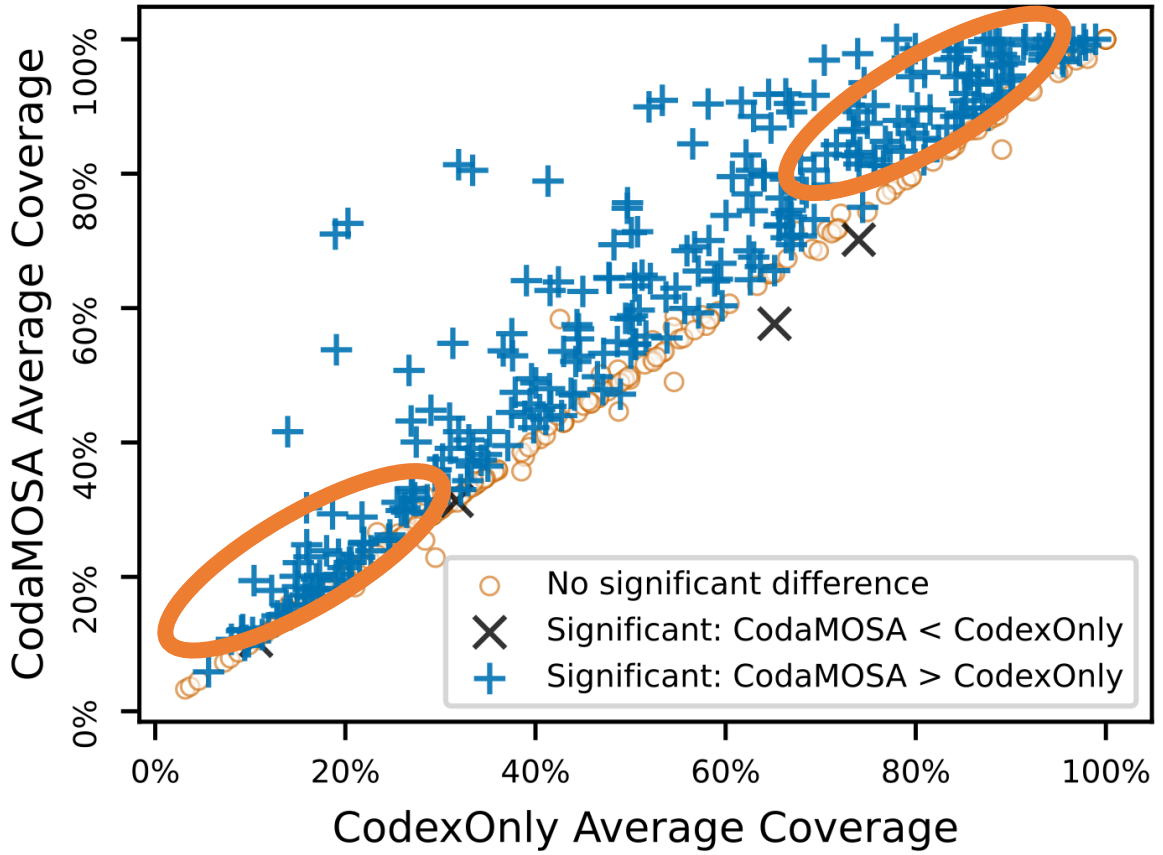
CodaMOSA Outperforms Baselines





CodaMOSA Outperforms Baselines





Common Causes for Improvements

- Manually analyze 20 benchmarks w/ biggest coverage increases
  - 15/20 benchmarks: “special strings”

```
bump_version('0.0.0')
```

  - 7/20 benchmarks: construct data correctly w/o Type Hints

```
str_0 = 'devbox01'
host_0 = module_2.Host(str_0)
group_0 = module_0.Group(str_0)
var_0 = [host_0, host_0, host_0, group_0, group_0, group_0]
vars_module_0 = module_1.VarsModule()
var_1 = vars_module_0.get_vars(str_0, str_0, var_0)
```

  - 5/20 benchmarks: introduce new syntactical constructs

```
str_0 = 'c'
var_0 = module_0.join_each(str_0, str_0)
var_1 = list(var_0)
```

Is Codex Just Copying Existing Tests?

Example with high similarity (0.713)

Codex generation:

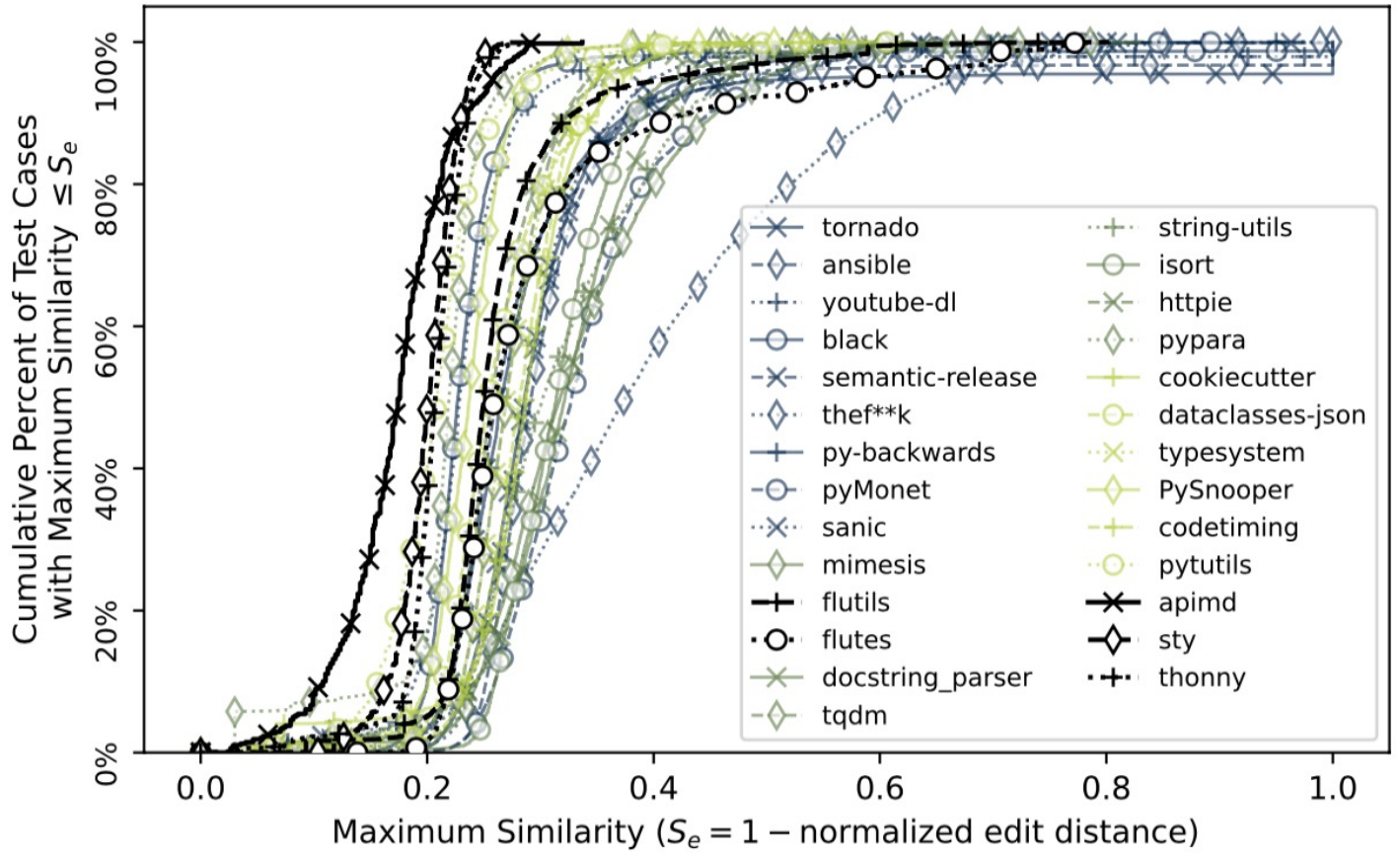
```
assert list(scanl(operator.add, [1,2,3,4], 0)) == [0,1,3,6,10]
assert list(scanl(lambda acc, x: x + acc, ['a', 'b', 'c', 'd']))
                                == ['a', 'ba', 'cba', 'dcba']}
```

Function in other part of code base (out of prompt):

```
check_iterator(flutes.scanl(operator.add, [1,2,3,4], 0), [0,1,3,6,10])}
check_iterator(flutes.scanl(lambda s, x: x + s, ['a', 'b', 'c', 'd']),
                              ['a','ba', 'cba', 'dcba'])}
```

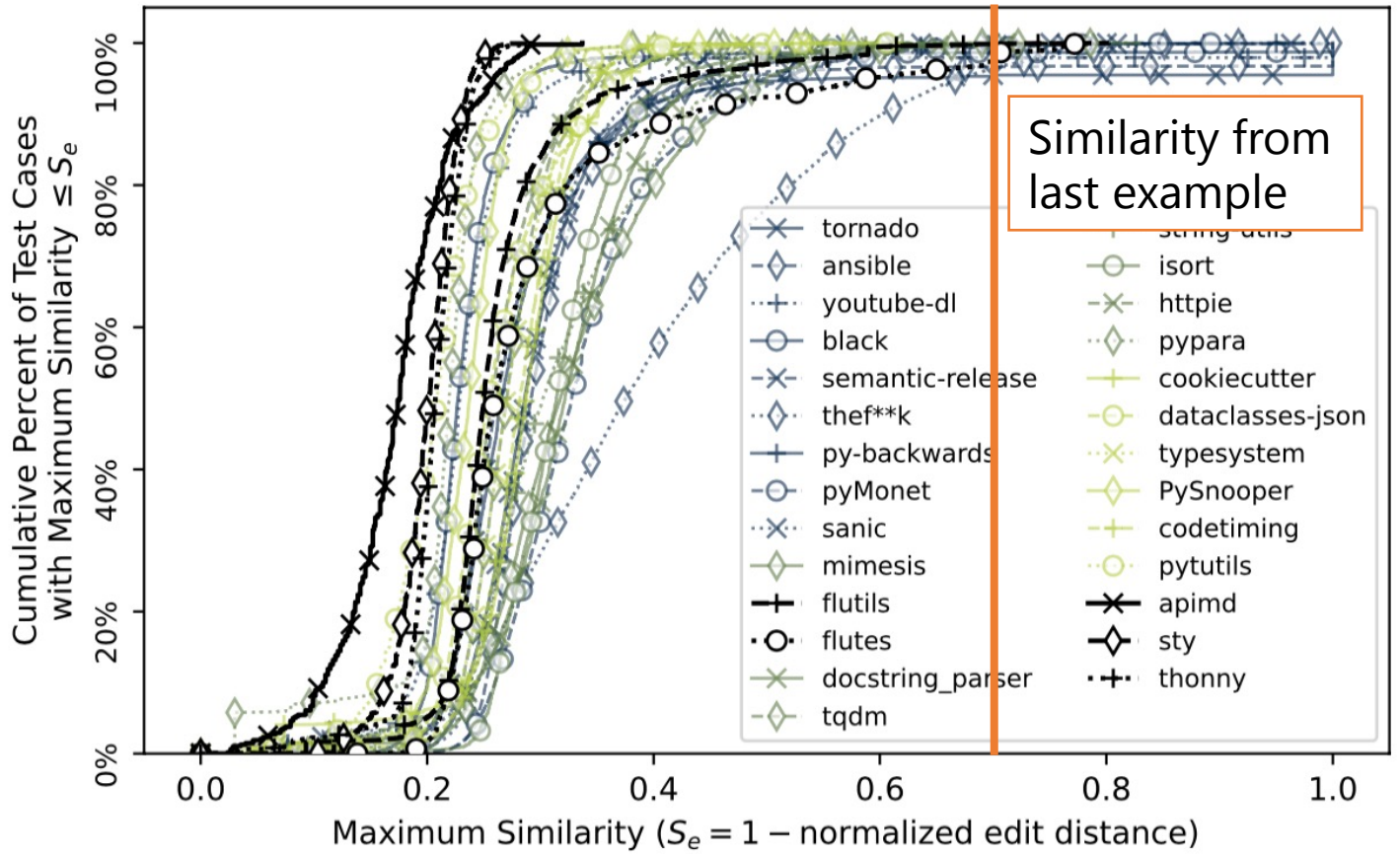
Most Generated Tests Not Too Similar

y-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis



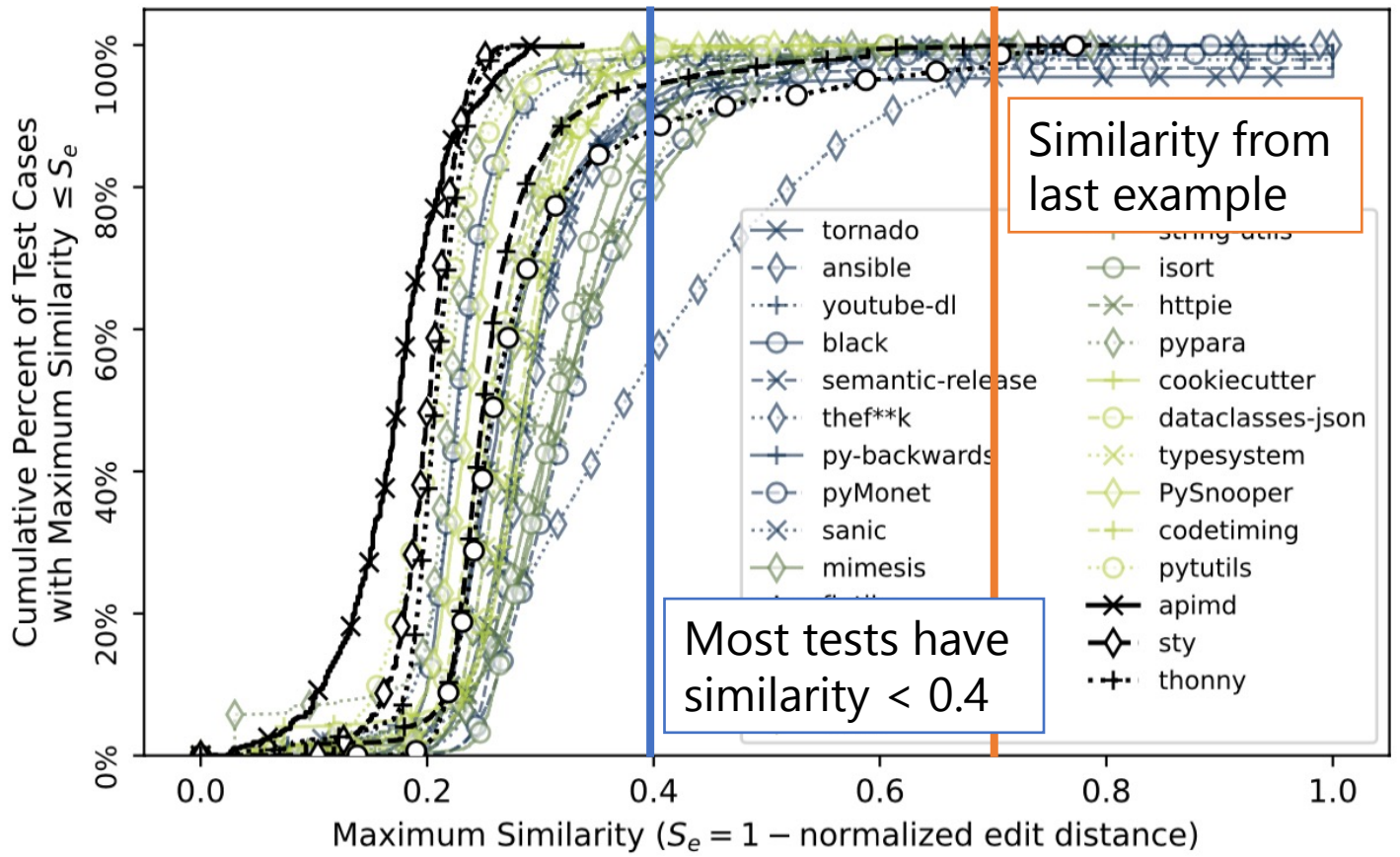
Most Generated Tests Not Too Similar

y-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis



Most Generated Tests Not Too Similar

y-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis



Core Approach

Evaluation Highlights

Remaining Challenges

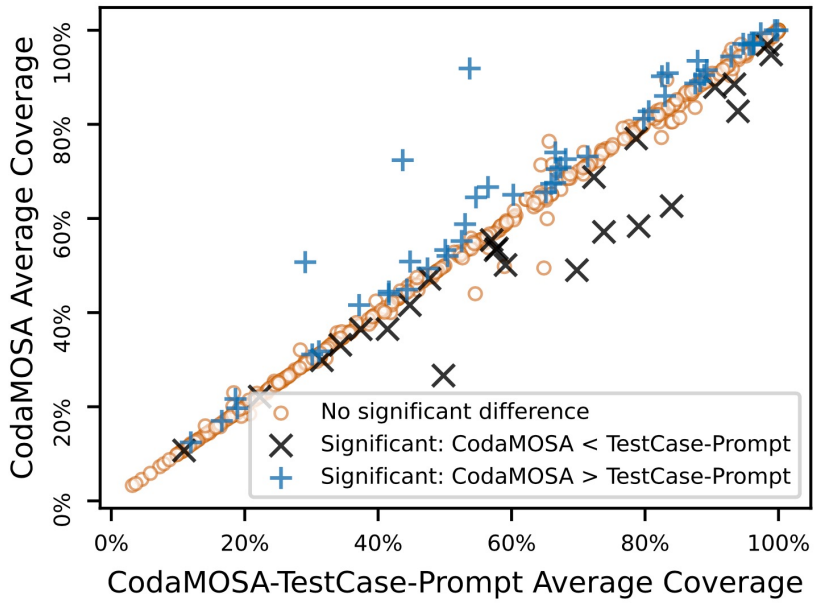
Core Approach

Evaluation Highlights

Remaining Challenges

Prompt Engineering

- Our prompts are very simple.
  - Implementation + “`# Unit test for function X\ndef test_X():\n`”
- Our evaluation showed prompts that included example tests sometimes improved performance:
- Could do substantially more here.



Finding Bugs

- We only evaluate coverage achieved.
  - coverage =/= bug-finding ability
- Codex generations often contain asserts, e.g.:

```
def test_bump_version():
  assert bump_version('0.0.0') == '1.0.0'
  assert bump_version('0.0.0', 1) == '0.1.0'
```

- Can these assertions help us find bugs with the test suites?
  - In example above, assertions capture wrong semantics

Naturalness of Tests

- Codex-generated tests are more natural

```
def test_bump_version():
  assert bump_version('0.0.0') == '1.0.0'
  assert bump_version('0.0.0', 1) == '0.1.0'
```

- Than SBST-generated ones

```
def test_case_2():
  str_0 = None
  int_0 = 1431
  str_1 = bump_version(str_0, int_0)
```

- Can we better preserve this naturalness?

CodaMOSA exploits synergy between

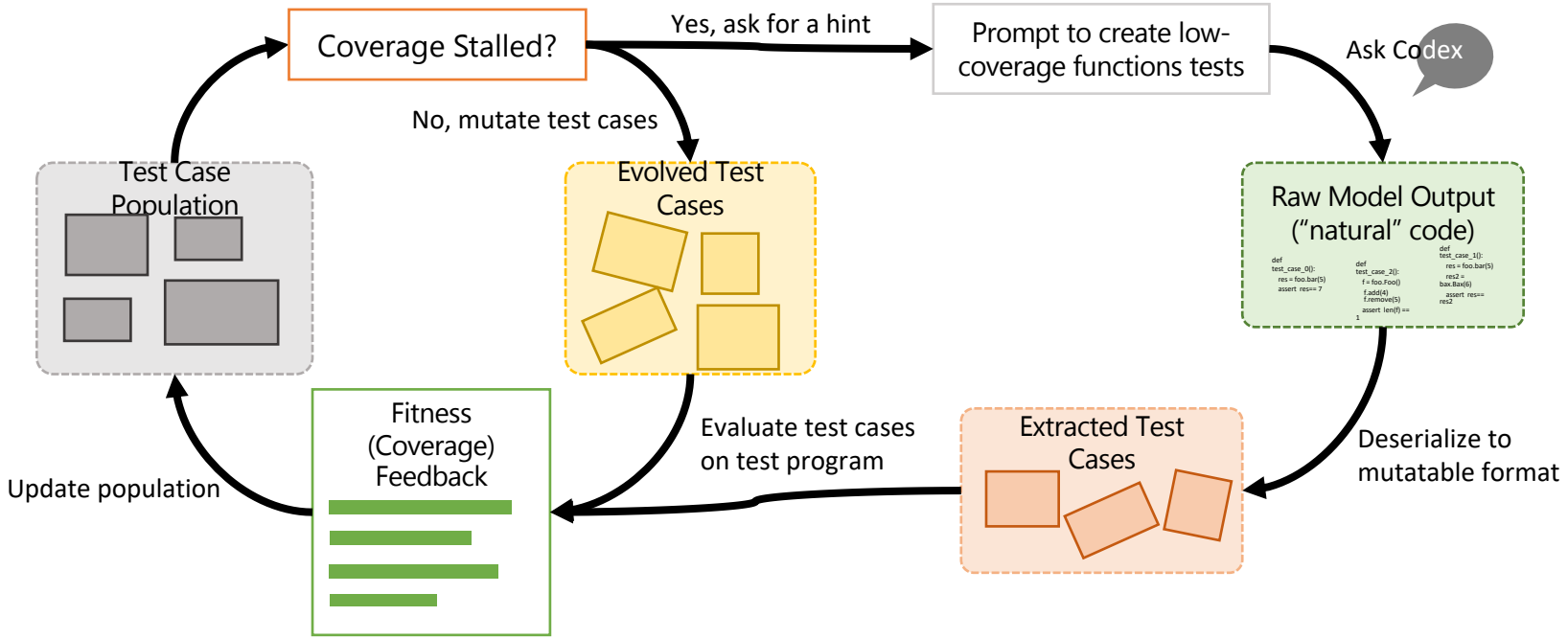
Large Language Models

"what is most expected"

Mutational Search

"something close, but unexpected"

(+) Big coverage increases
(?) Bug-finding ability
(?) More complex prompting
(?) Performance on "unseen" code hard to evaluate
(-) Relies on closed-access API



CodaMOSA: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models.
C Lemieux, J P Inala, S K Lahiri, S Sen.