Taller #5: EvoSuite

Generación Automática de Tests - 2018

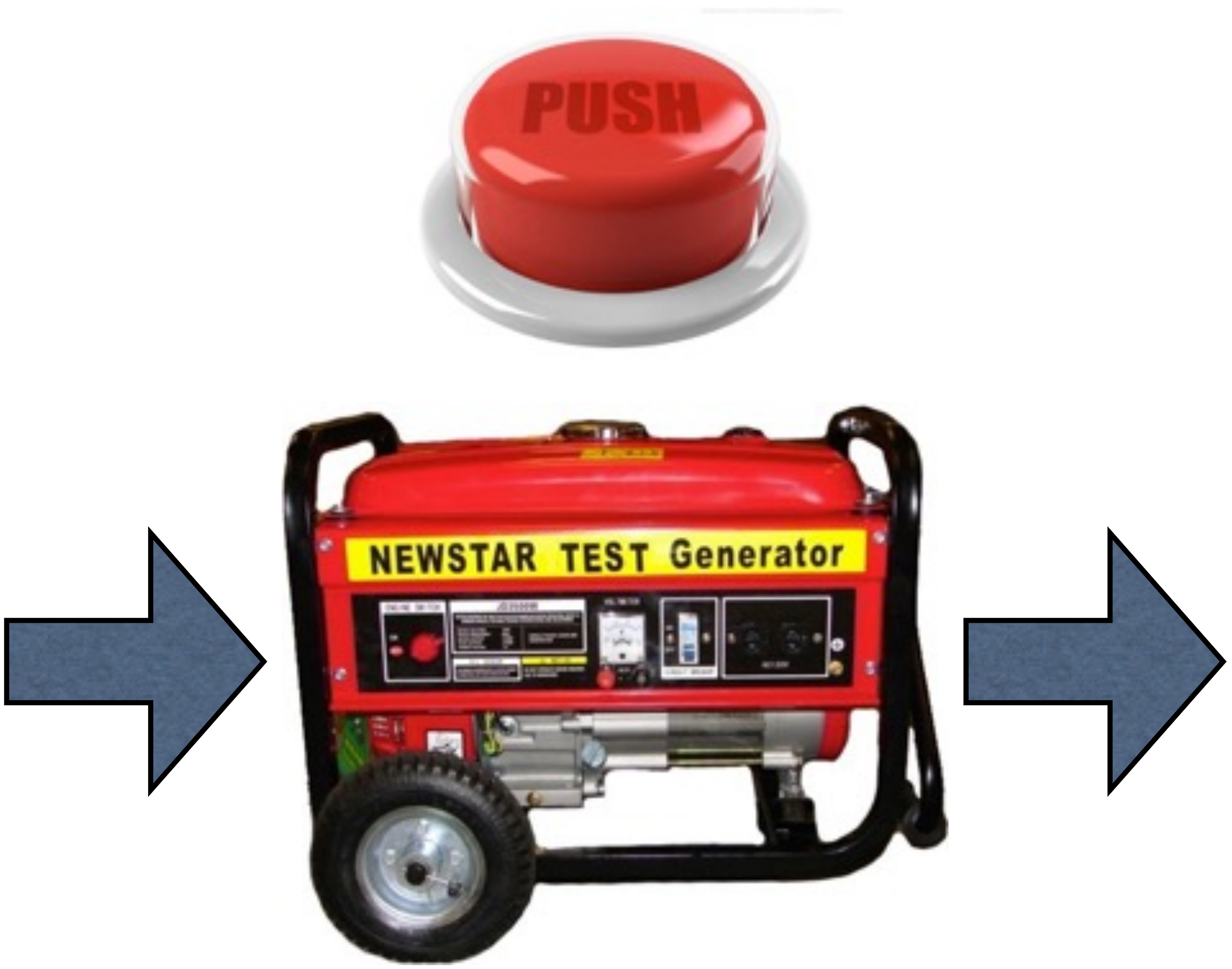
Test Suite Generation

User Code

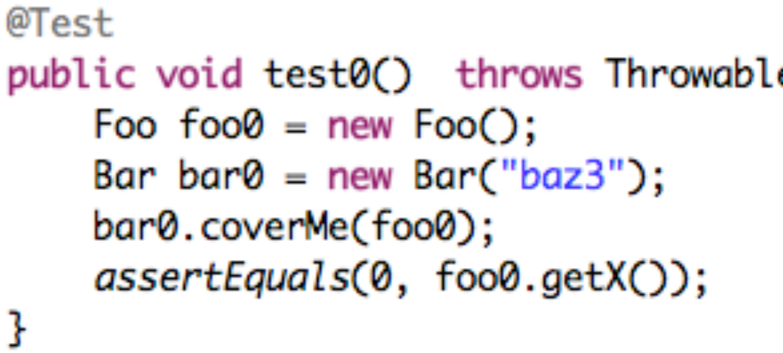
```
Image
Thread timer;
public void init() {  count = 0;
    lastcount = 10; Image[10];
    pictures = new           = new Media
    MediaTracker tracker  a < lastcount;
    for (int a = 0; a < lastcount;
        pictures[a] = getImage(
            getCodeBase(), new In
        tracker.addImage(pictures
    }
    tr   er.checkAll(true);
}
public void start() {
    if (timer == null) {
        timer = new Thread(
        timer.start();

    }
}
public void paint(Graph
    g.drawImage(pictur
    if (count == last

}
   id run() {
```



Test Suite



```
@Test
public void test0()  throws Throwabl
    Foo foo0 = new Foo();
    Bar bar0 = new Bar("baz3");
    bar0.coverMe(foo0);
    assertEquals(0, foo0.getX());
}
```

EVOSUITE

- Automatic Test Suite Generation for Java
- http://www.evosuite.org (Release: 1.0.6)
- https://github.com/EvoSuite/evosuite
    - GNU Lesser General Public License (LGPL)
- Plugins: Eclipse, IntelliJ, Maven, Jenkins

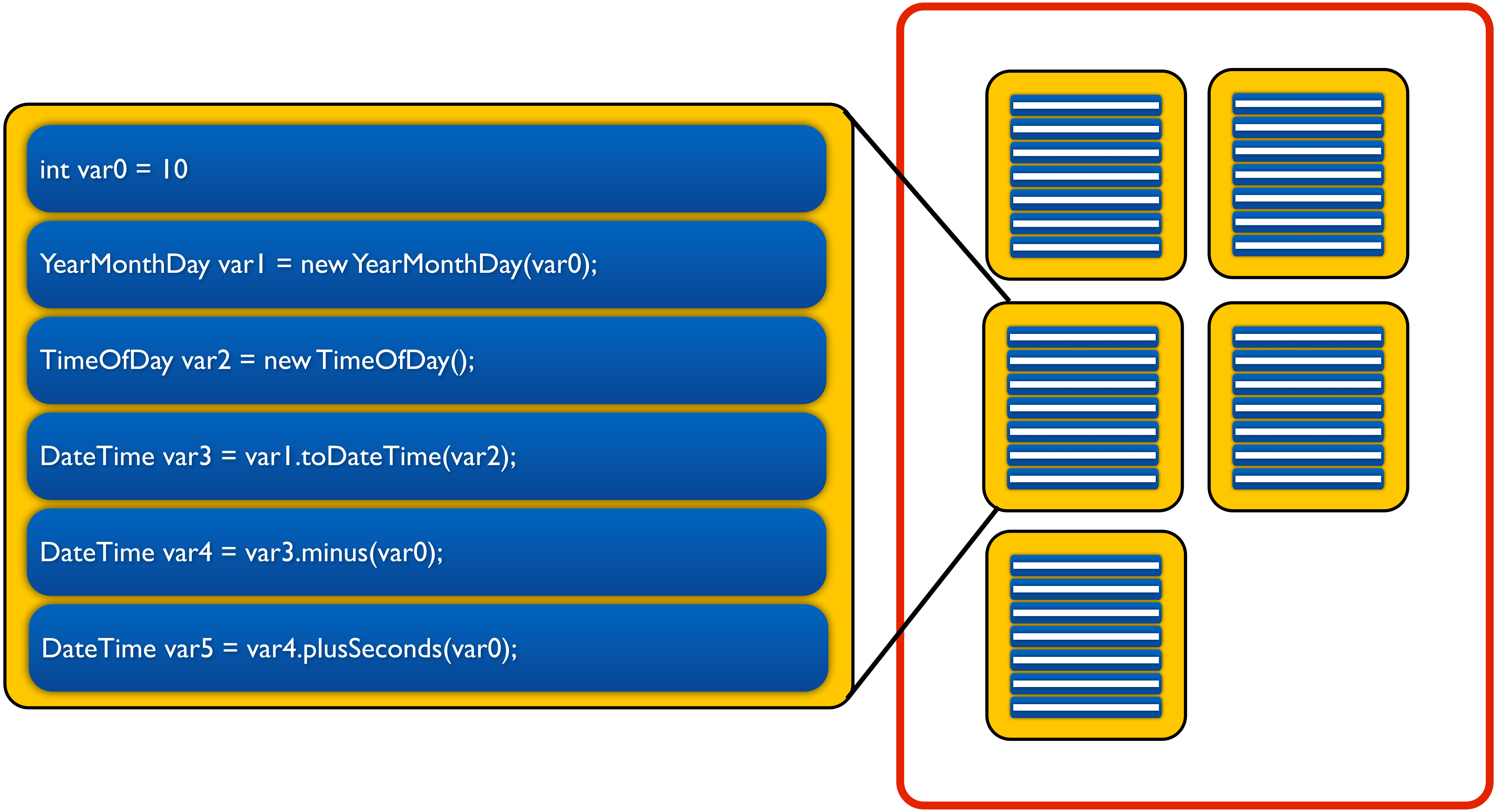
Search-Based Test Generation

- Test Generation as an *optimization* problem
- Genetic Algorithms:
  - Mimics natural process of evolution
- Traditional approach: optimize test case for each objective goal in isolation

Single Goal Strategy

- Population: set of test cases
- Let G1, G2, G3 be objective goals:
  - How to distribute the search budget?
    - What happens if G2 is unsatisfiable?
    - What happens if G1 is more complex than G3?

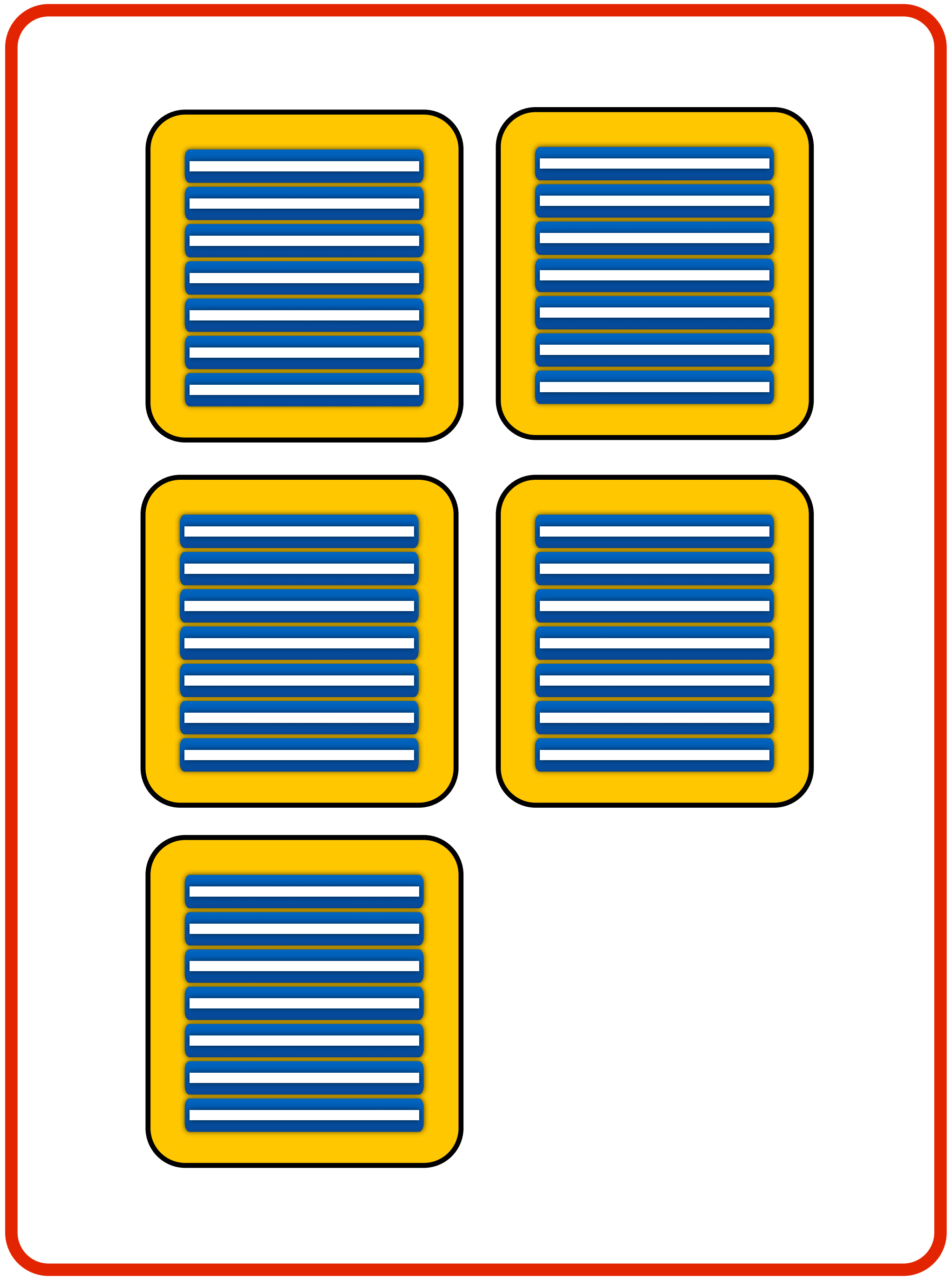
EvoSuite: Whole-Test Suite Generation



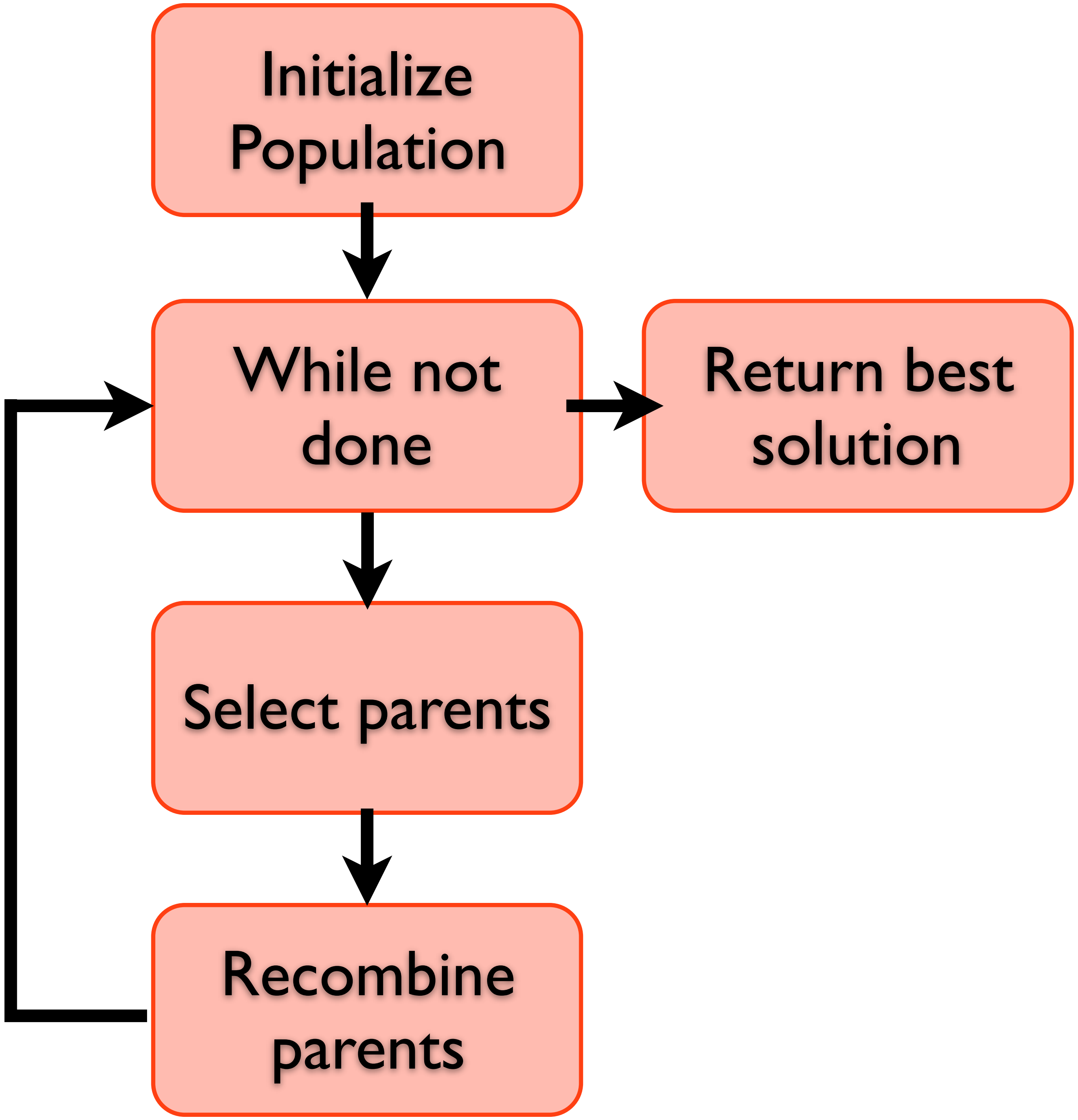
```
int var0 = 10
YearMonthDay var1 = new YearMonthDay(var0);
TimeOfDay var2 = new TimeOfDay();
DateTime var3 = var1.toDateTime(var2);
DateTime var4 = var3.minus(var0);
DateTime var5 = var4.plusSeconds(var0);
```

EvoSuite: Whole-Test Suite Generation

- Optimize entire test suite at once towards
- Ordering of coverage goals no longer an issue
- Infeasibility of individual coverage goals does not affect search.



Genetic Algorithm (GA)



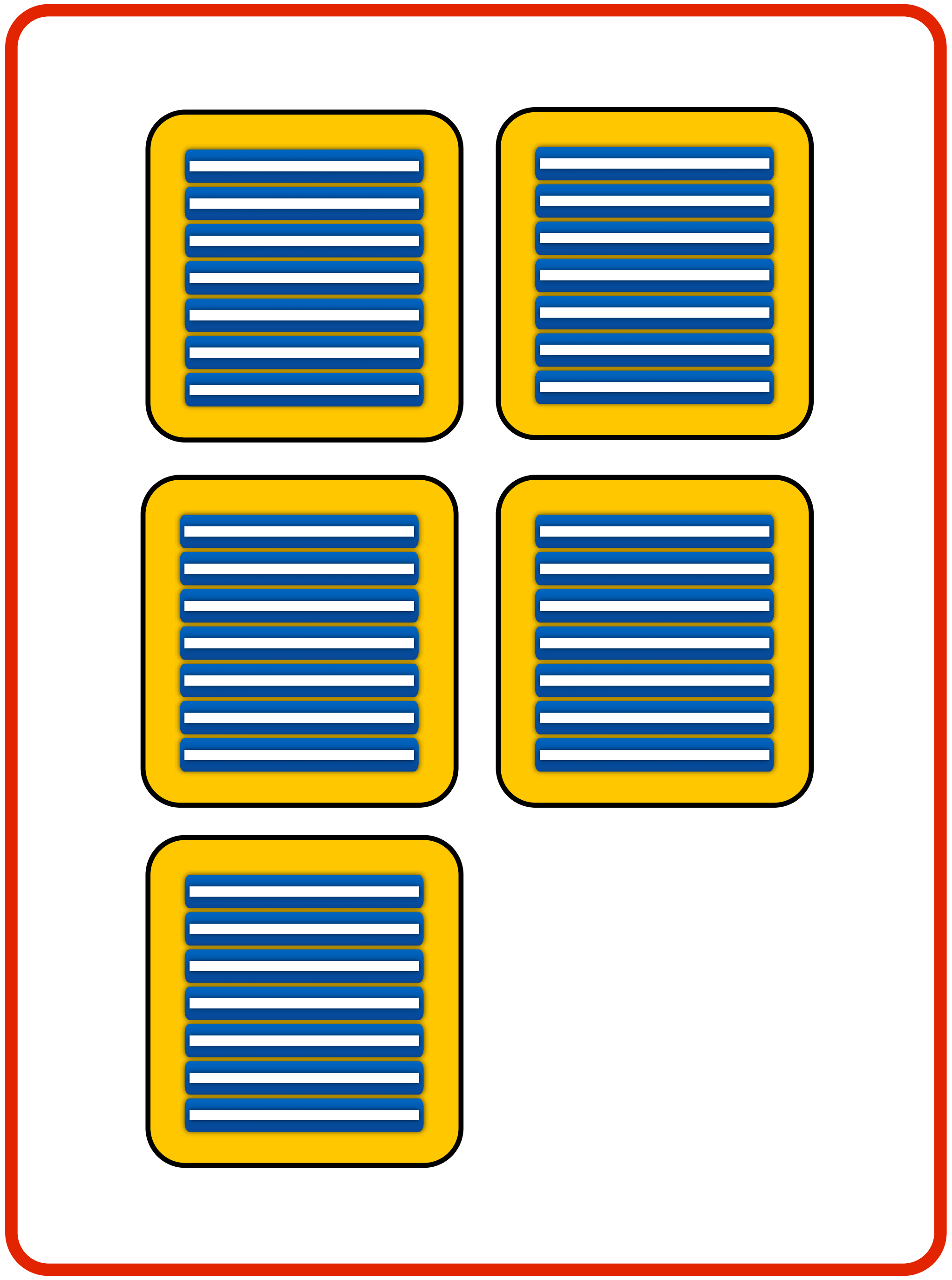
Initialize Population

While not done

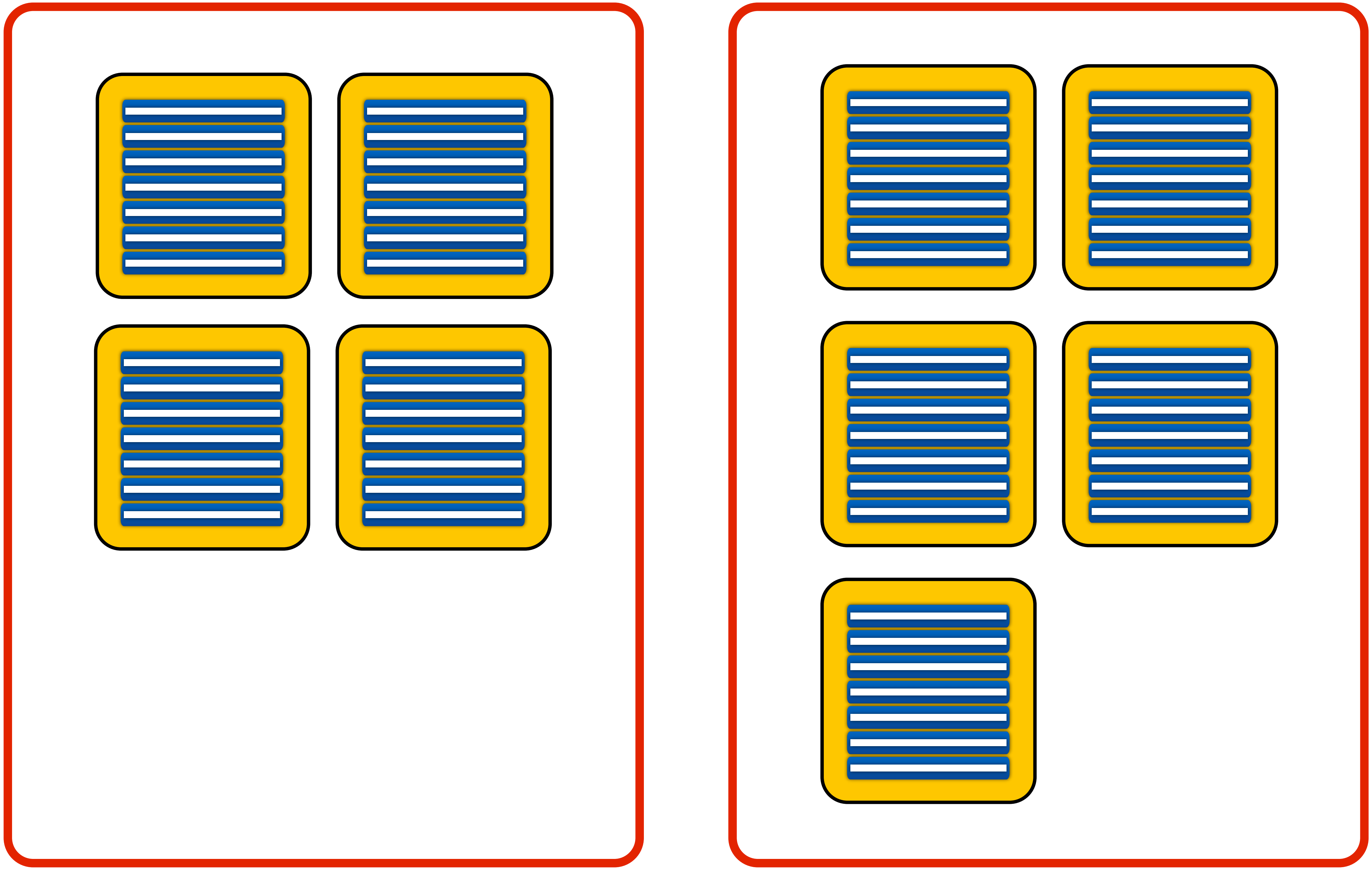
Return best solution

Select parents

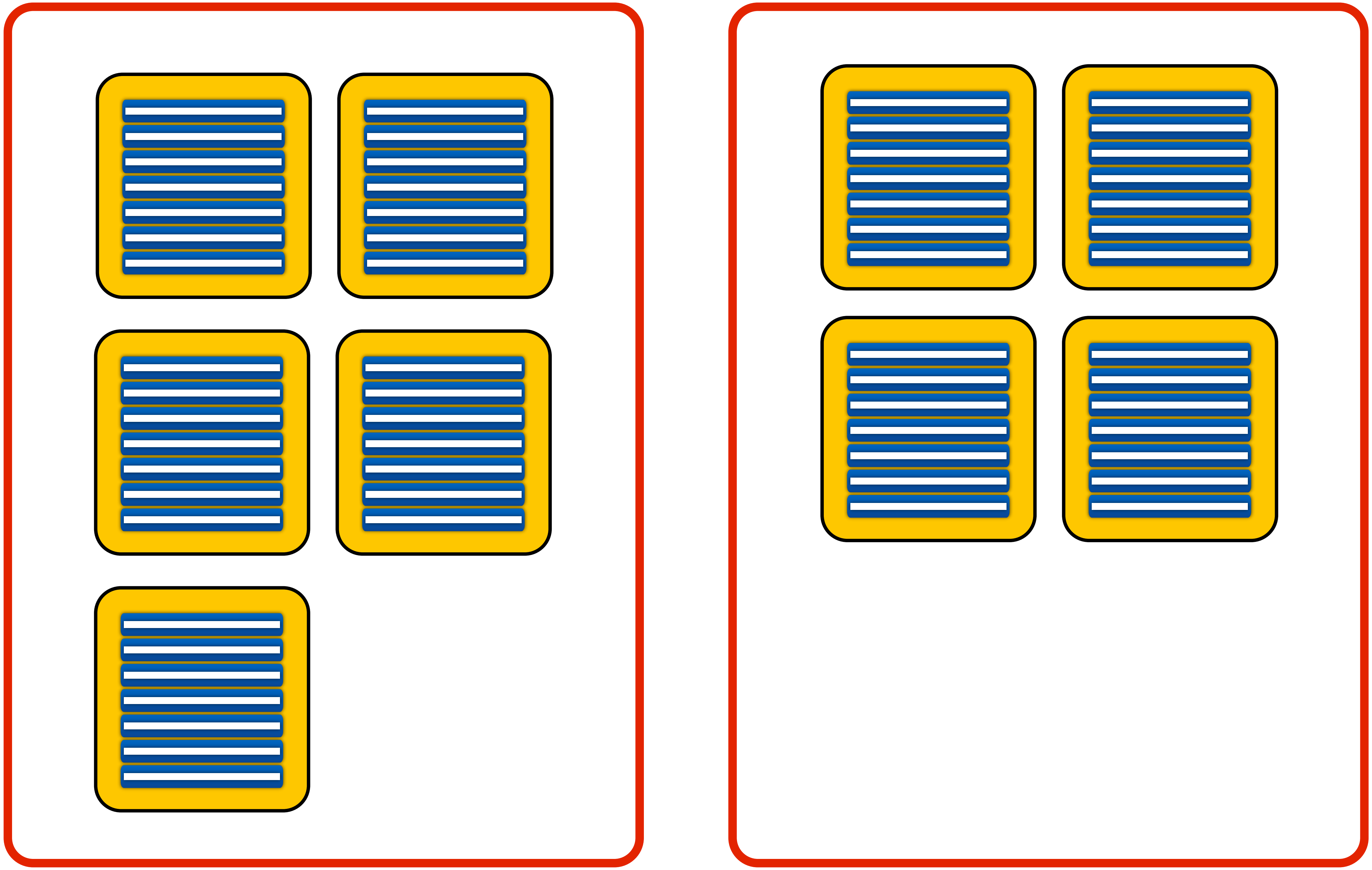
Recombine parents



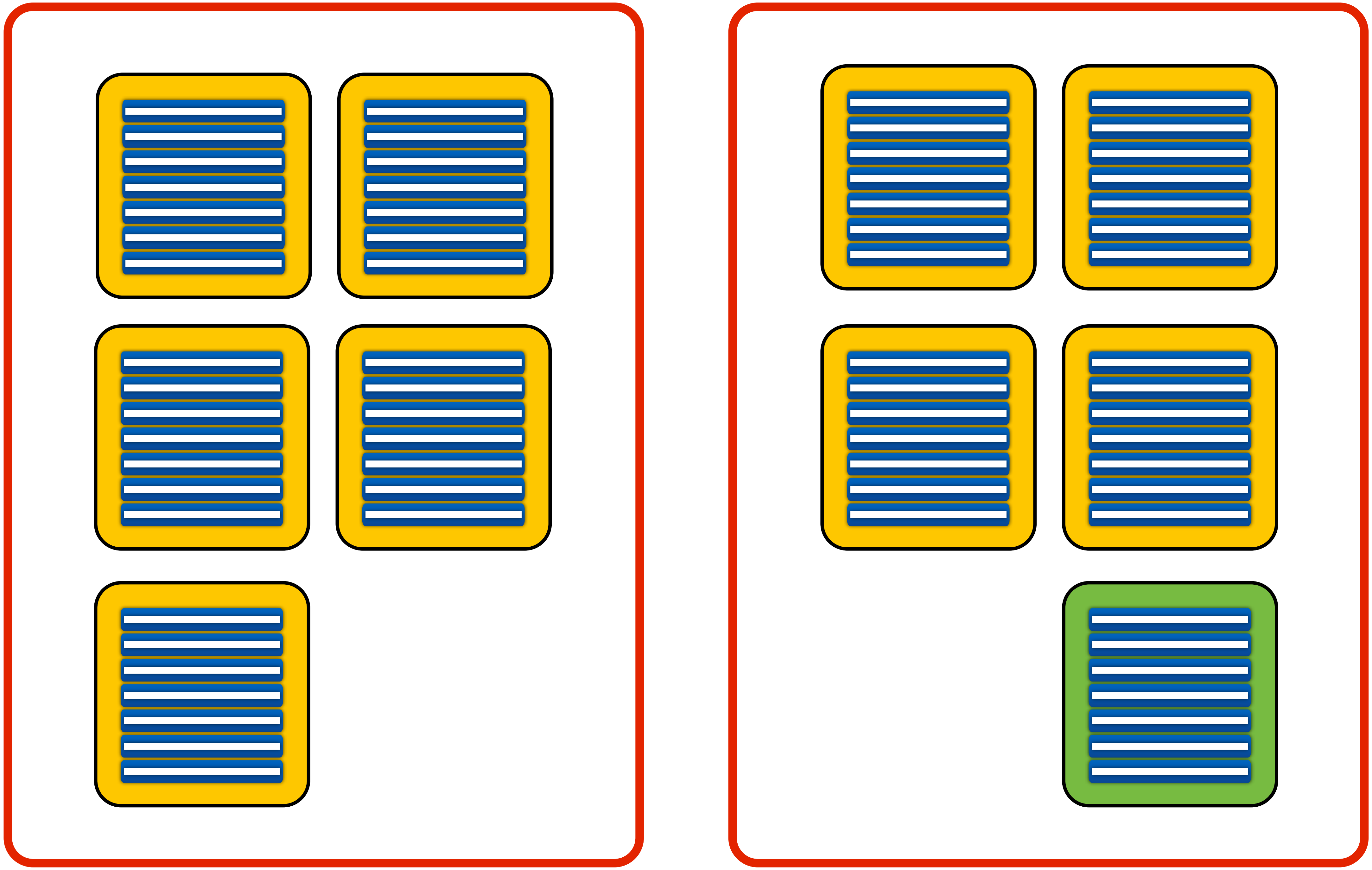
Whole-Test Suite Generation



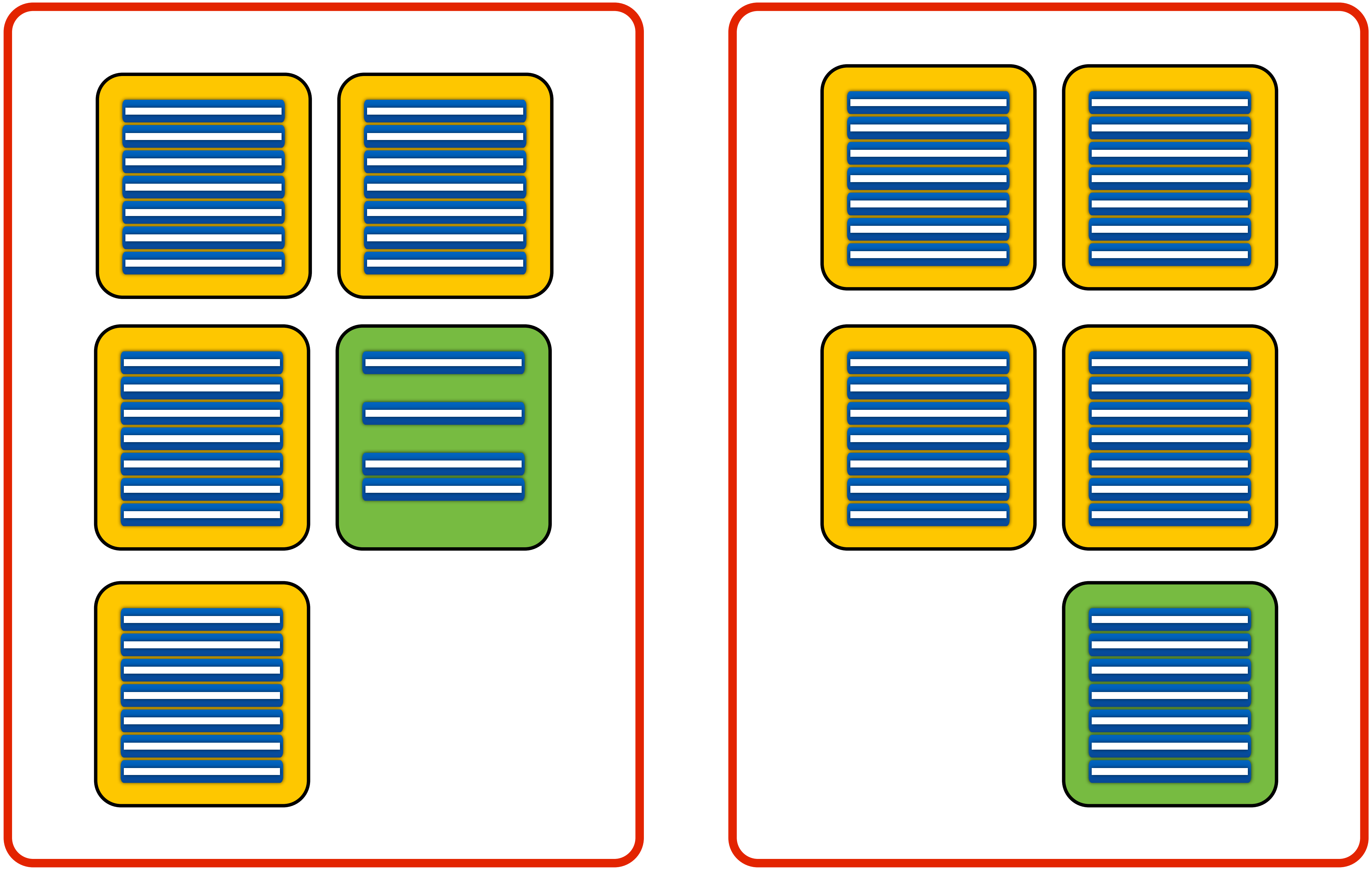
Crossover

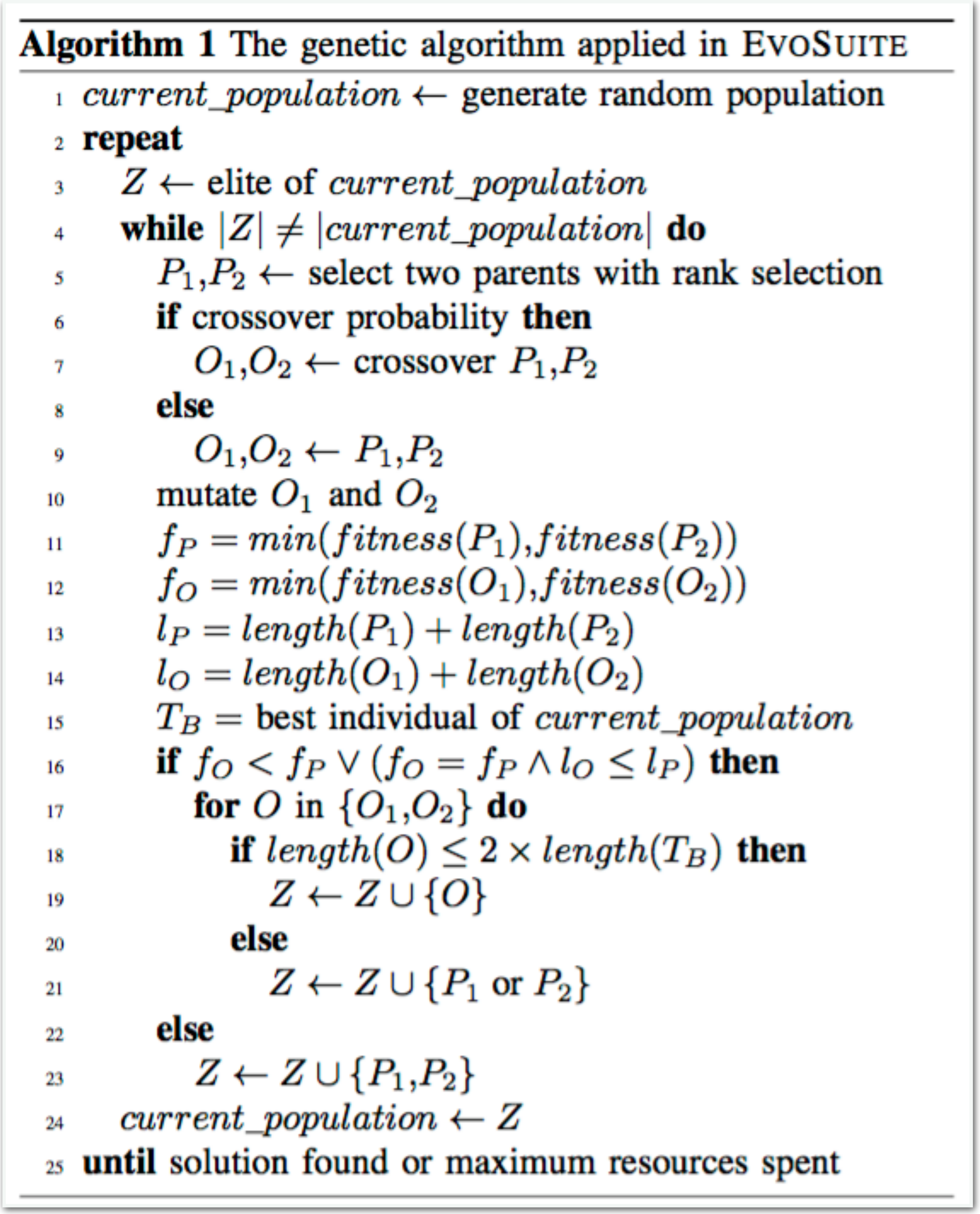


Mutation



Mutation





Algorithm 1 The genetic algorithm applied in EvoSuite

```
$current\_population \leftarrow$ generate random population
repeat
  $Z \leftarrow$ elite of $current\_population$
  while $|Z| \neq |current\_population|$ do
    $P_1,P_2 \leftarrow$ select two parents with rank selection
    if crossover probability then
      $O_1,O_2 \leftarrow$ crossover $P_1,P_2$
    else
      $O_1,O_2 \leftarrow P_1,P_2$
    mutate $O_1$ and $O_2$
    $f_P = min(fitness(P_1),fitness(P_2))$
    $f_O = min(fitness(O_1),fitness(O_2))$
    $l_P = length(P_1) + length(P_2)$
    $l_O = length(O_1) + length(O_2)$
    $T_B =$ best individual of $current\_population$
    if $f_O < f_P \vee (f_O = f_P \wedge l_O \leq l_P)$ then
      for $O$ in $\{O_1,O_2\}$ do
        if $length(O) \leq 2 \times length(T_B)$ then
          $Z \leftarrow Z \cup \{O\}$
        else
          $Z \leftarrow Z \cup \{P_1$ or $P_2\}$
    else
      $Z \leftarrow Z \cup \{P_1,P_2\}$
  $current\_population \leftarrow Z$
until solution found or maximum resources spent
```

Coverage Criteria

- Method Coverage: all methods in the CUT are executed
- Top-Level Method Coverage: each method is also invoked **directly** (from the test case)
- No-Exception Top-Level Method Coverage: all methods are covered via direct invocations from the tests and considering **only** normal-terminating executions (i.e., no exception)

Coverage Criteria

- Line Coverage: all statements to be executed
- Branch Coverage: all branch predicates evaluated to true and false.
- Direct Branch Coverage: each branch in a public method of the CUT to be covered by a direct call from a unit test

Coverage Criteria

- Output Coverage: output covers different domains (positive, negative, digits, alphabetical, etc.)
- Weak Mutation: each mutant for the CUT at least one its tests reaches state infection (no propagation)
- Exception Coverage: all possible exceptions in each method of the CUT (cannot be defined with a percentage)

Compound Criteria

- Non-Conflicting criteria (no need of multi-objective)
- We can easily combine all criteria to define our ad-hoc fitness function. For example:
    - LINE:BRANCH
    - METHOD:EXCEPTION
    - All together
- By Default: same weight (not normalized)

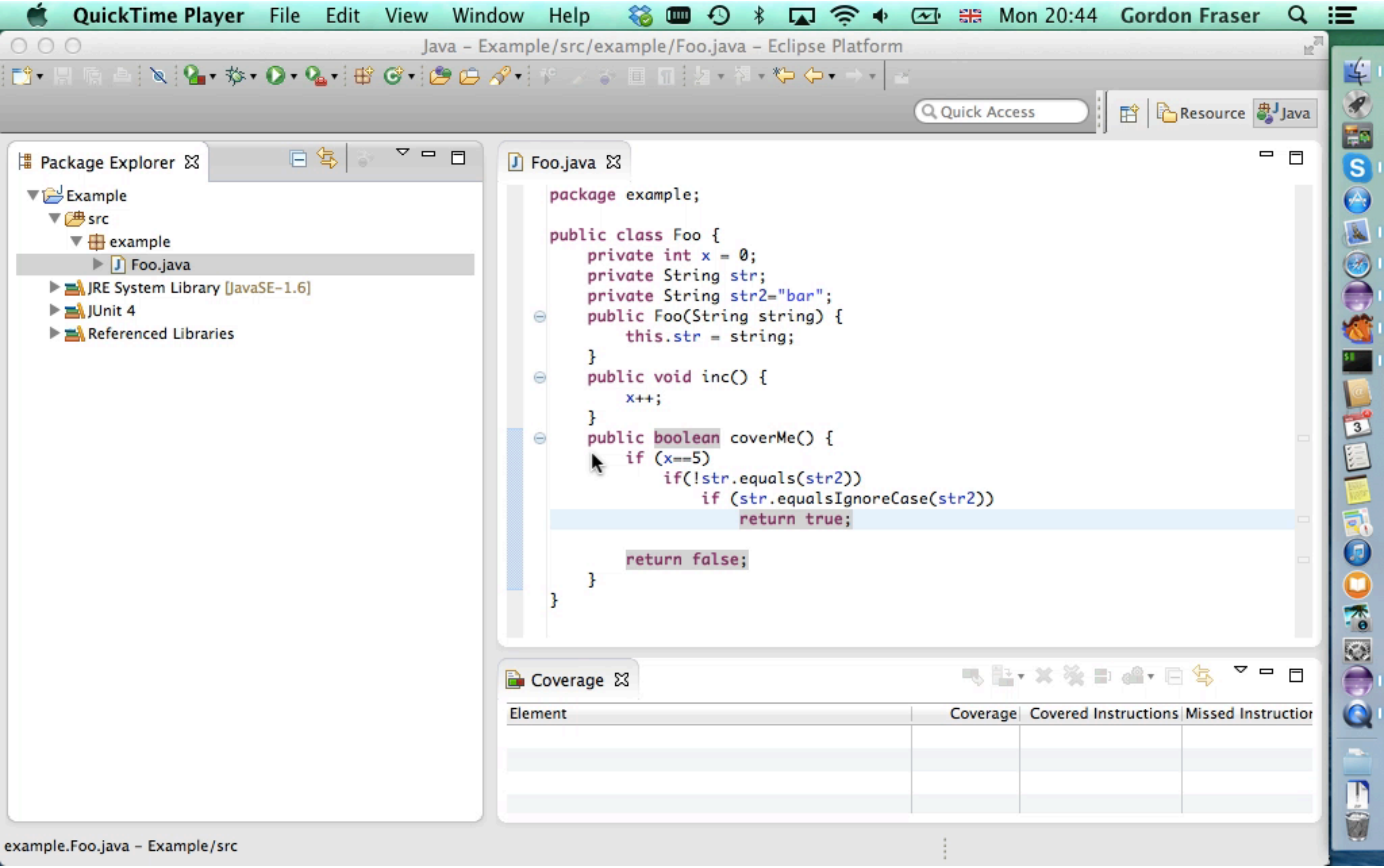
Test Suite Post-Process

- Once the GA is over, EvoSuite post-processes the test suite

1. **Minimization**: Removes statements/tests that do not contribute to goal coverage

2. **Assertion Generation**: Adds Assertions (finite set of patterns) such that at least one mutant is killed by the assertion

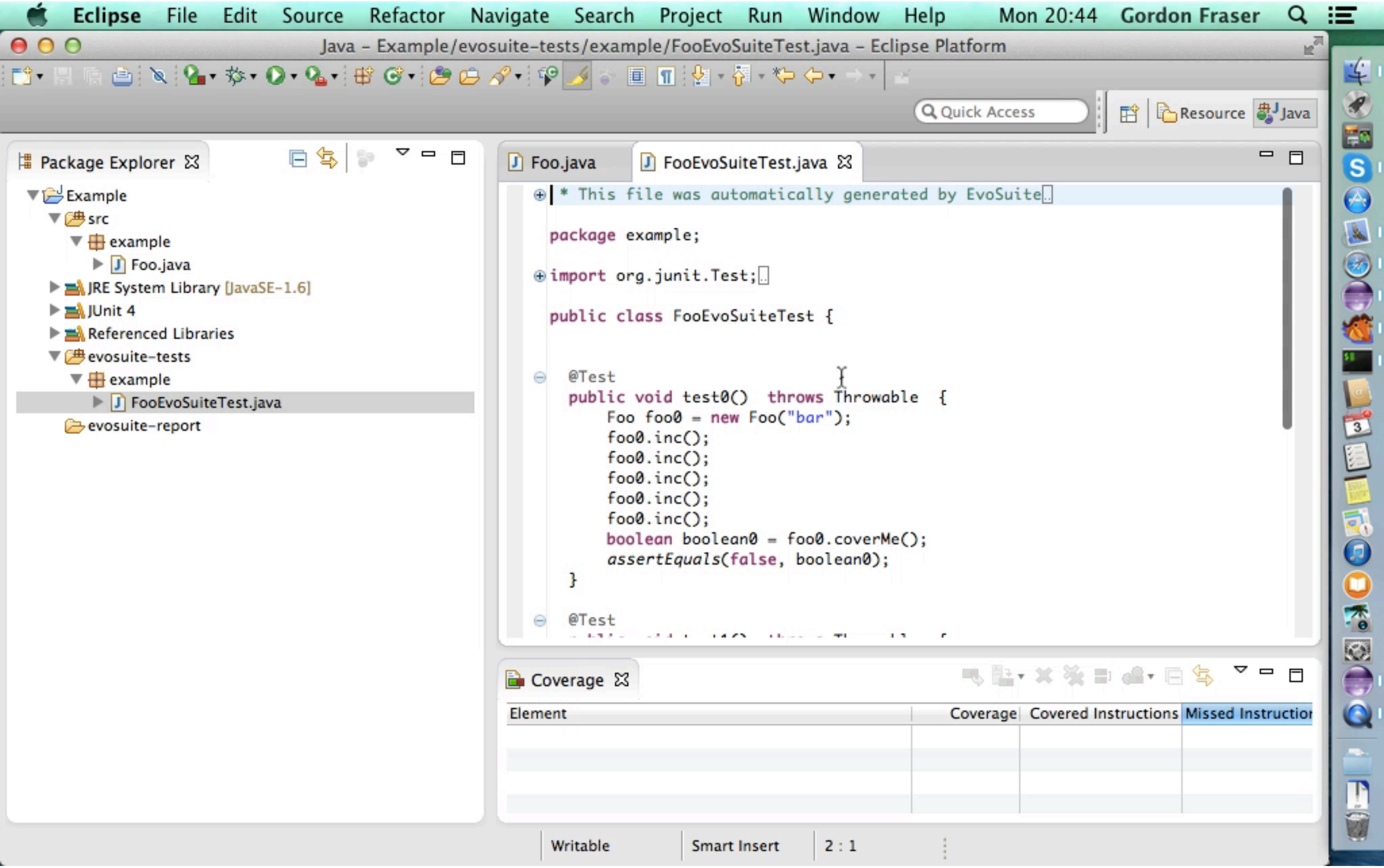
3. **JUnit Write**: Checks if the resulting JUnit is stable



```
package example;

public class Foo {
    private int x = 0;
    private String str;
    private String str2="bar";
    public Foo(String string) {
        this.str = string;
    }
    public void inc() {
        x++;
    }
    public boolean coverMe() {
        if (x==5)
            if(!str.equals(str2))
                if (str.equalsIgnoreCase(str2))
                    return true;

        return false;
    }
}
```



```
 * This file was automatically generated by EvoSuite

package example;

import org.junit.Test;

public class FooEvoSuiteTest {


  @Test
  public void test0()  throws Throwable  {
      Foo foo0 = new Foo("bar");
      foo0.inc();
      foo0.inc();
      foo0.inc();
      foo0.inc();
      foo0.inc();
      boolean boolean0 = foo0.coverMe();
      assertEquals(false, boolean0);
  }

  @Test
```

Element	Coverage	Covered Instructions	Missed Instructior

Enunciado - Ejercicio #1

- Generar automáticamente un test suite para StackAr usando EvoSuite

Enunciado - Ejercicio #2

- Extender StackAr con una nueva operación increaseCapacity(int) que permita aumentar la capacidad del StackAr si el valor es mayor a 0 (sino IllegalArgEx..)
- Extender el test suite con 100% de line/branch coverage del ejercicio anterior para la nueva versión extendida de StackAr.

Enunciado - Ejercicio #2

- La extensión debe cumplir los siguientes requisitos de complejidad
- Sea *k* es la cantidad de veces que se efectuó la operación increaseCapacity(int)
    - push(), get(int) y pop(): se ejecute en O(*k*)
    - top(): se ejecute en O(1)
    - increaseCapacity(int): se ejecute en O(*k*)
    - aclaración: asumir **new int[int]** se ejecuta en “O(1)”