FONDAZIONE BRUNO KESSLER

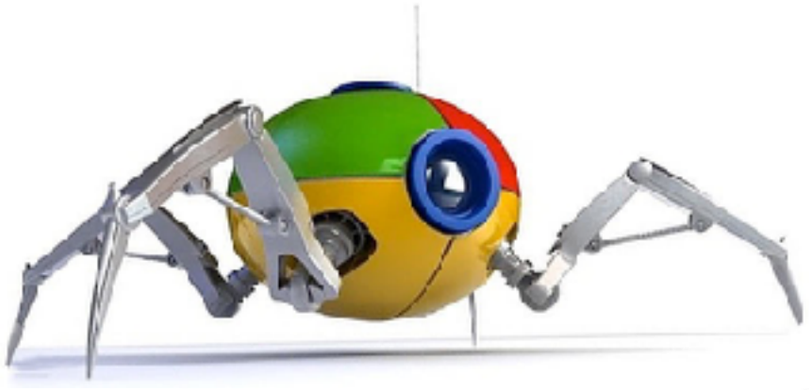
SE SOFTWARE ENGINEERING

Search Based Testing of Web Applications

Paolo Tonella

tonella@fbk.eu

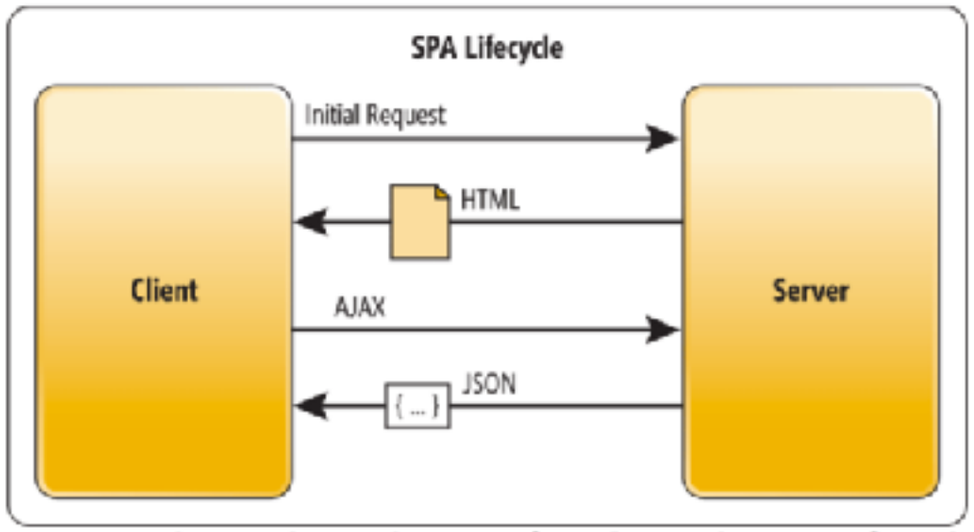
Web testing

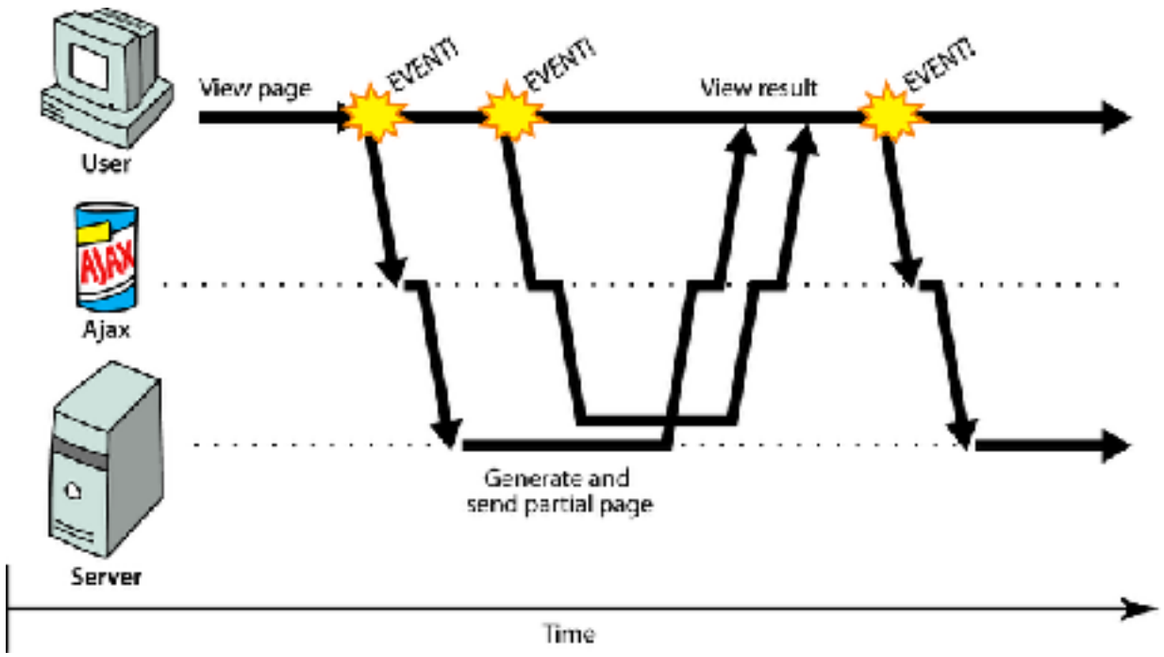


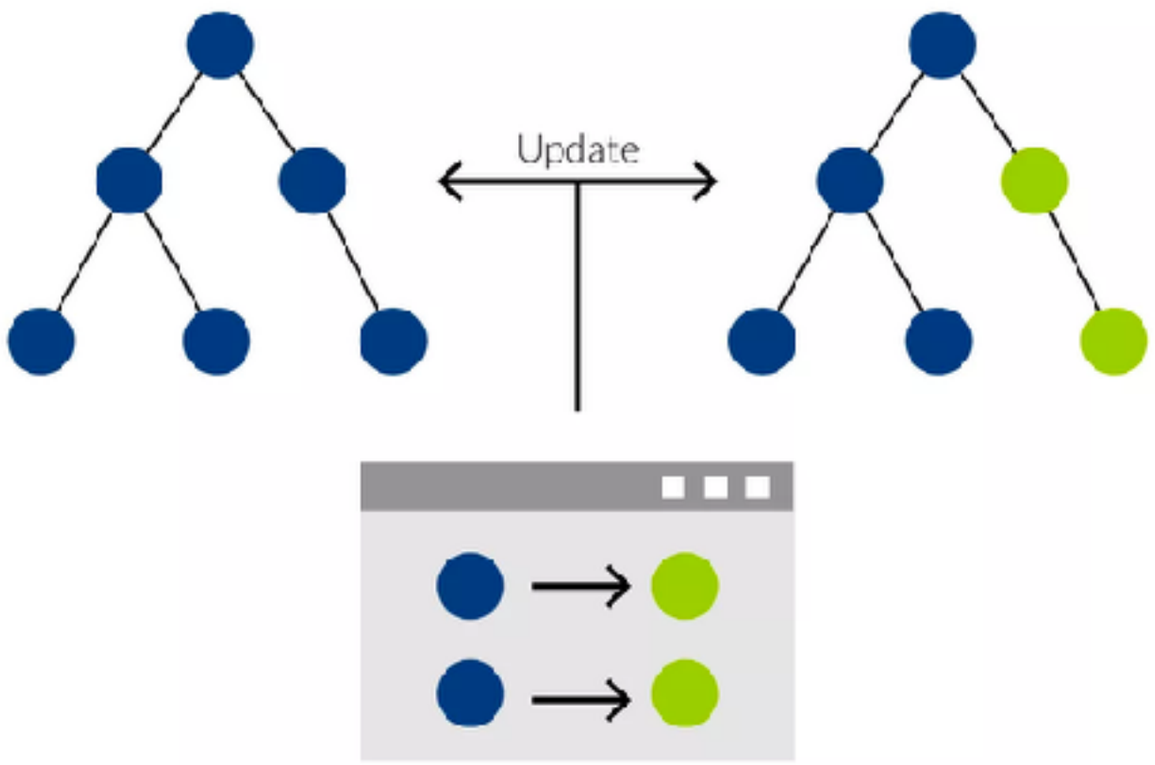
Crawler

Matteo Biagiola, Filippo Ricca, Paolo Tonella: ***Search Based Path and Input Data Generation for Web Application Testing***. 9th Int. Symposium on Search Based Software Engineering (SSBSE), pp. 18-32, 2017

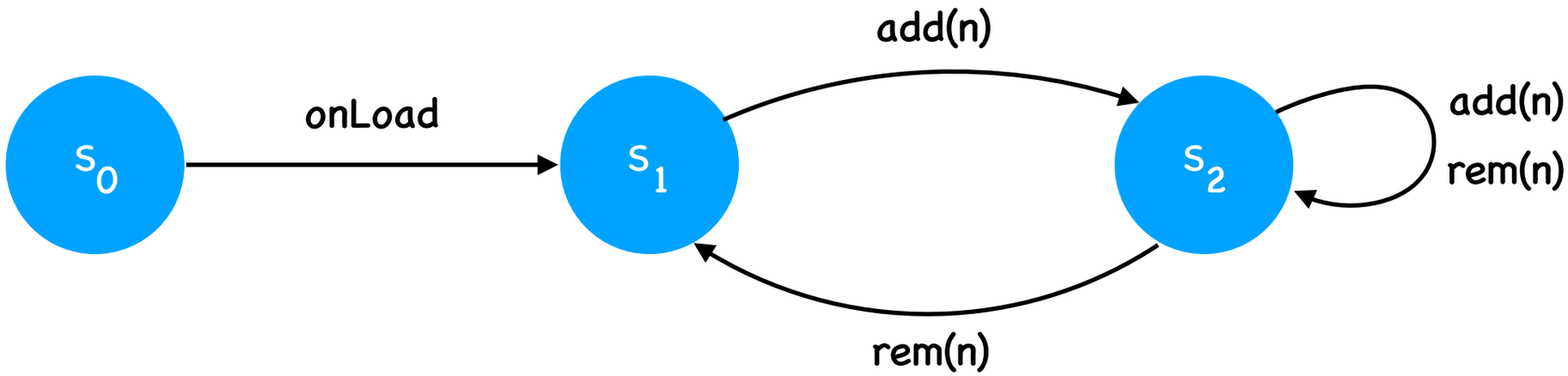
Single Page Applications







Model based testing

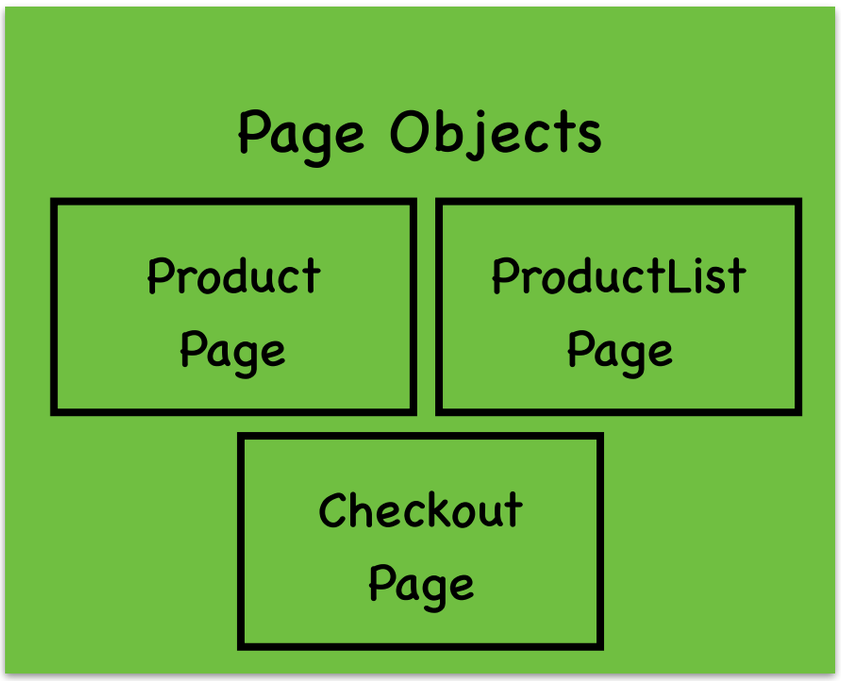


<add(1), rem(1)>

<add(1), rem(1), rem(1)> **Divergence**

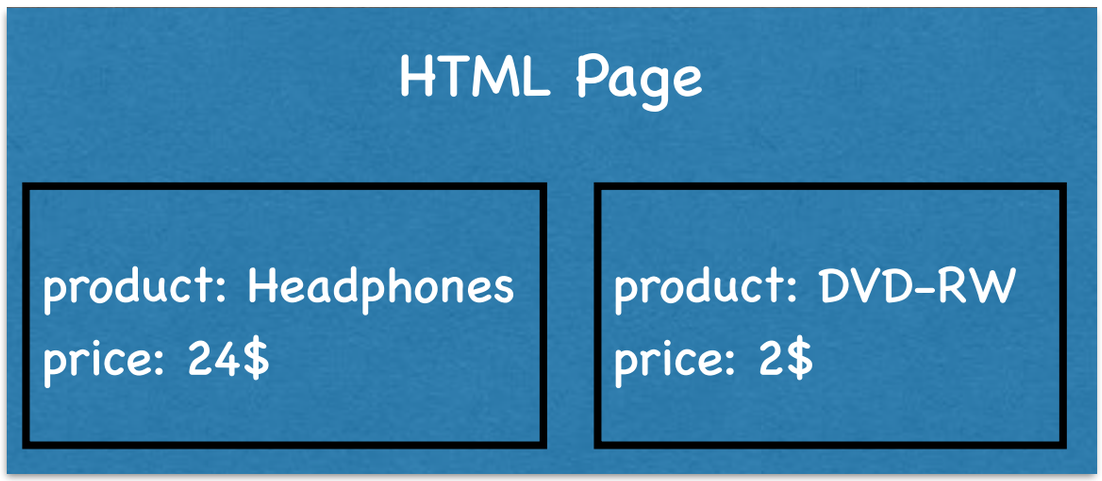
1) how do we get the model?
2) how do we avoid divergences?
3) how do we reduce execution time?

Model Construction



Page Object API

```
class ProductPage extends PageObject {
  void selectProductByName(String name){…}
  double getPrice(){…}
  void updatePrice(double price){…}
  CheckoutPage checkout() {…}
}
```

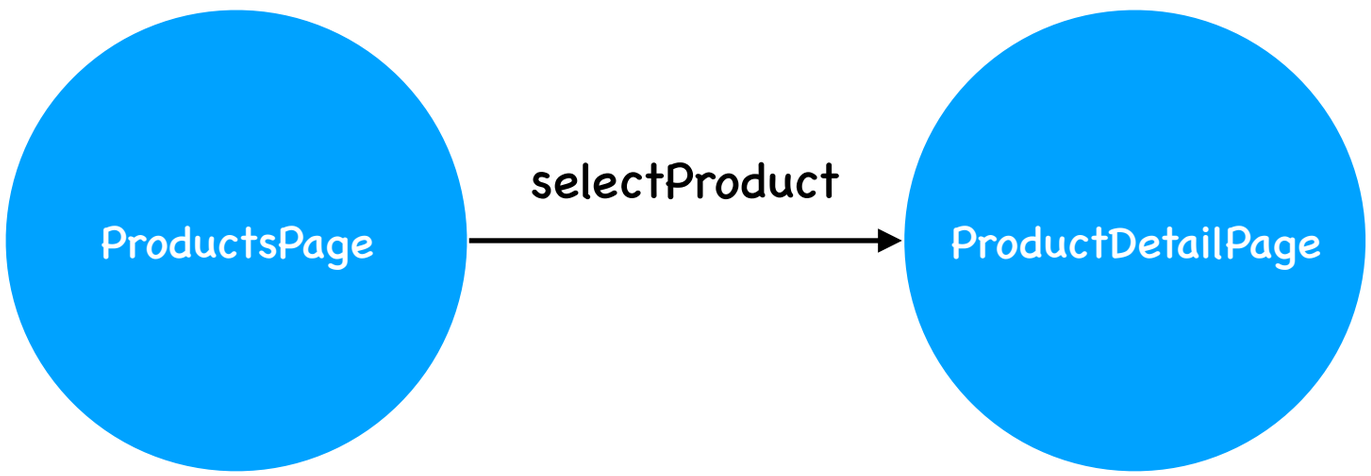


DOM API

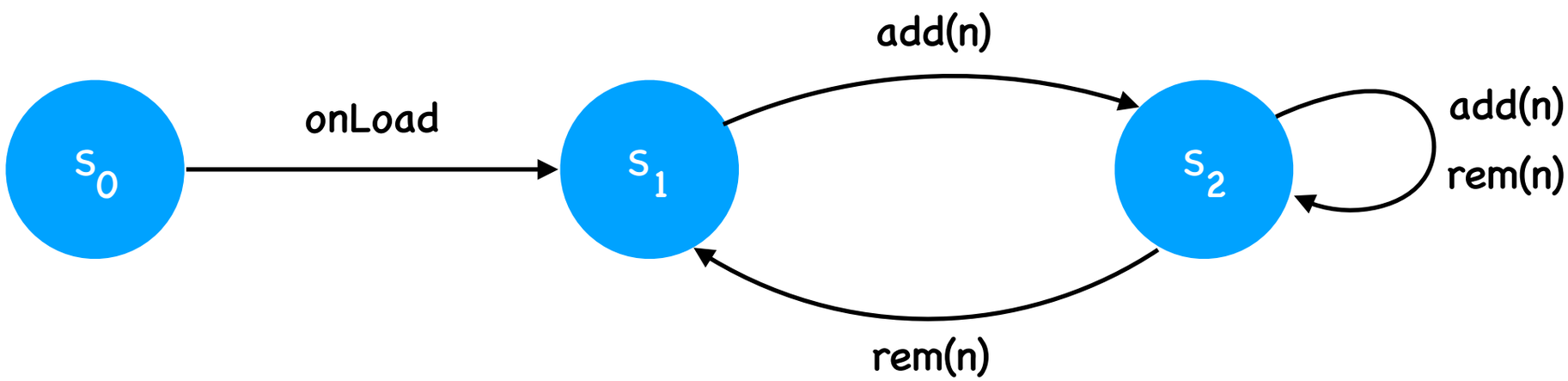
```
el = find_element(By.XPATH('/html/div/h2/table/div[3]'));
el.getText();
el.setText('49.99');
el.click();
```

From Page Objects to Navigation Model

```
1  public class ProductsPage implements PageObject {
2     public WebDriver driver;
3     public ProductsPage(WebDriver driver) {...}
4     public int getActiveCategory() {...}
5
6     public ProductDetailPage selectProduct(int id, int category) {
7        if((id >= 1 && id <= 6) &&
8           (category >= 1 && category <= 3) &&
9           (this.getActiveCategory() == category)) {
10          this.driver.findElement(By.id("product-" + id + "-" +
                category)).click();
11          return new ProductDetailPage(this.driver);
12       } else {
13          throw new IllegalArgumentException("Invalid parameter values");
14       }
15    }
16 }
```

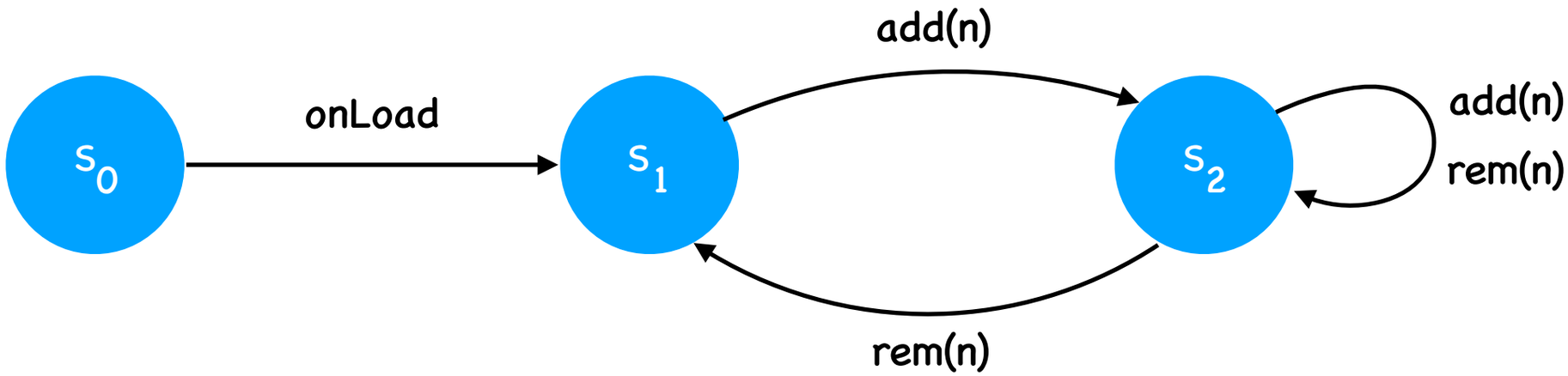


Test derivation



1) by graph visit algorithms
2) by model checking
3) by search based algorithms
4) by diversity based algorithms

Depth-first visit

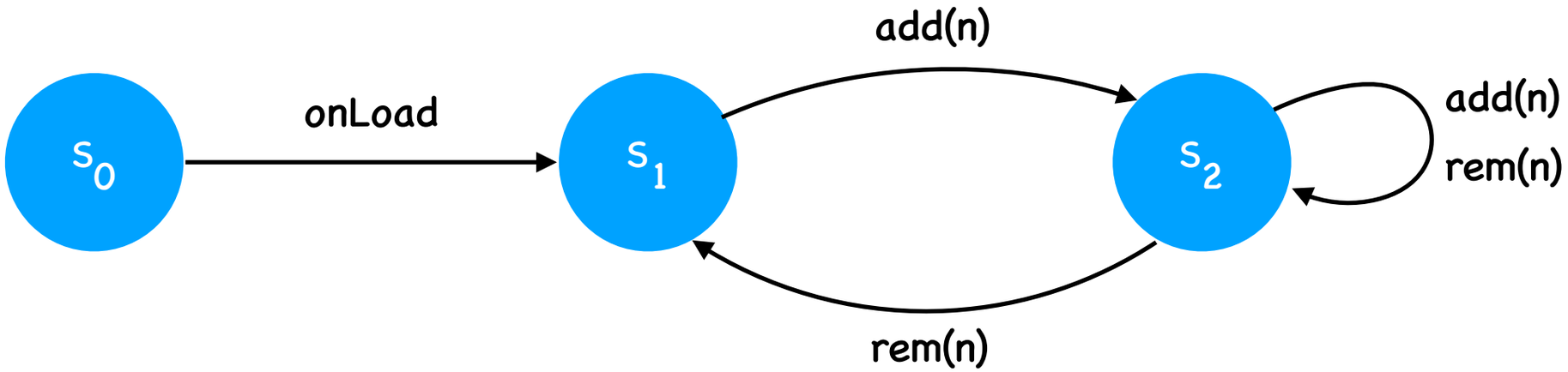


<add(57), rem(4)>

<add(1), add(2), rem(102)>

Actual parameter values are filled with random numbers

Model checking

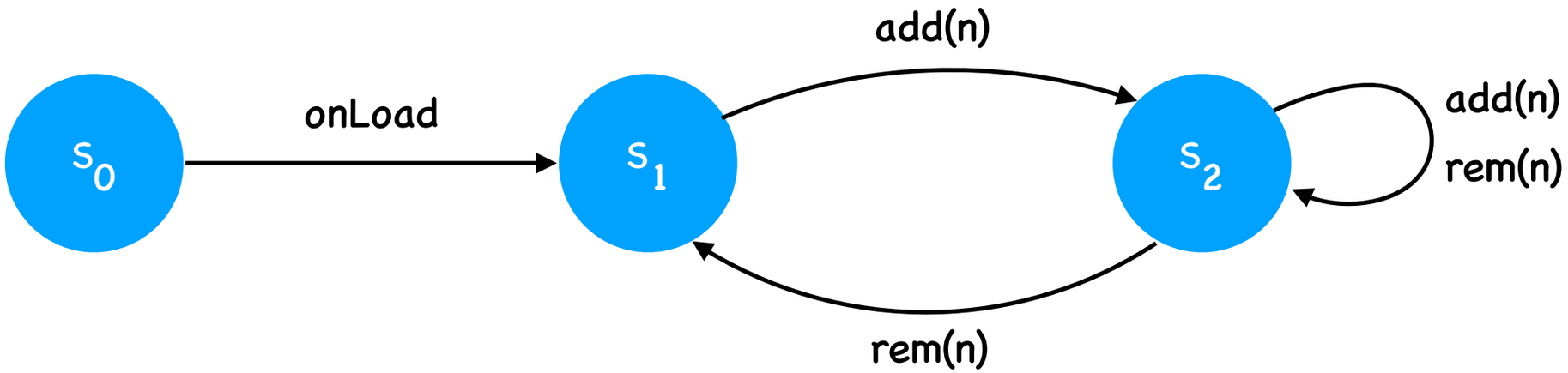



```
TRANS
  case
    action = add:
      next(state) = S2;
      next(items) = items + n; -- server side
    action = rem & state = S2 & items - n > 0 :
      next(state) = S2;
      next(items) = items - n; -- server side
    action = rem & state = S2 & items - n = 0 :
      next(state) = S1;
      next(items) = 0; -- server side
    action = rem & state = S2 & items - n < 0 :
      next(state) = ERROR;
  esac

LTLSPEC G ( state = S2 & action = rem -> X state != S2 )
LTLSPEC G ( state = S2 & action = rem -> X state != S1 )
```

✓ <add(2), rem(1)>

✓ <add(1), rem(1)>

Model checking



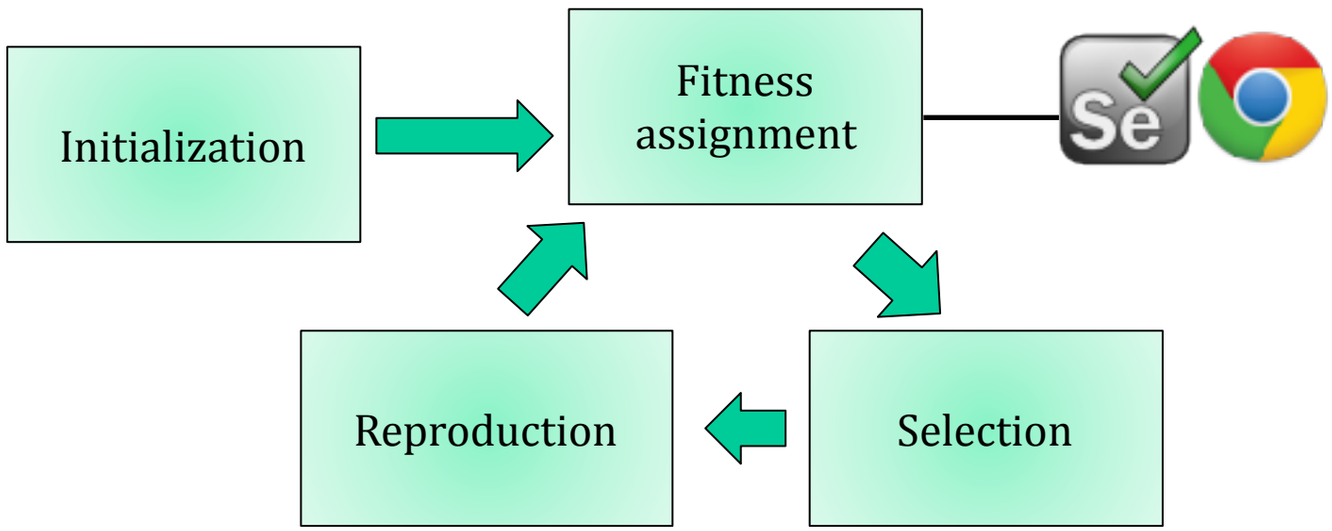

```
TRANS
  case
    action = add:
      next(state) = S2;
      -- next(items) = items + n;
    action = rem & state = S2 & items - n > 0 :
      next(state) = S2;
      -- next(items) = items - n;
    action = rem & state = S2 & items - n = 0 :
      next(state) = S1;
      -- next(items) = 0;
    action = rem & state = S2 & items - n < 0 :
      next(state) = ERROR;
  esac

LTLSPEC G ( state = S2 & action = rem -> X state != S2 )
LTLSPEC G ( state = S2 & action = rem -> X state != S1 )
```

✓ <add(2), rem(1)>

✗ <add(1), rem(2)>

Model checker sets `items` to 2

Search based generator



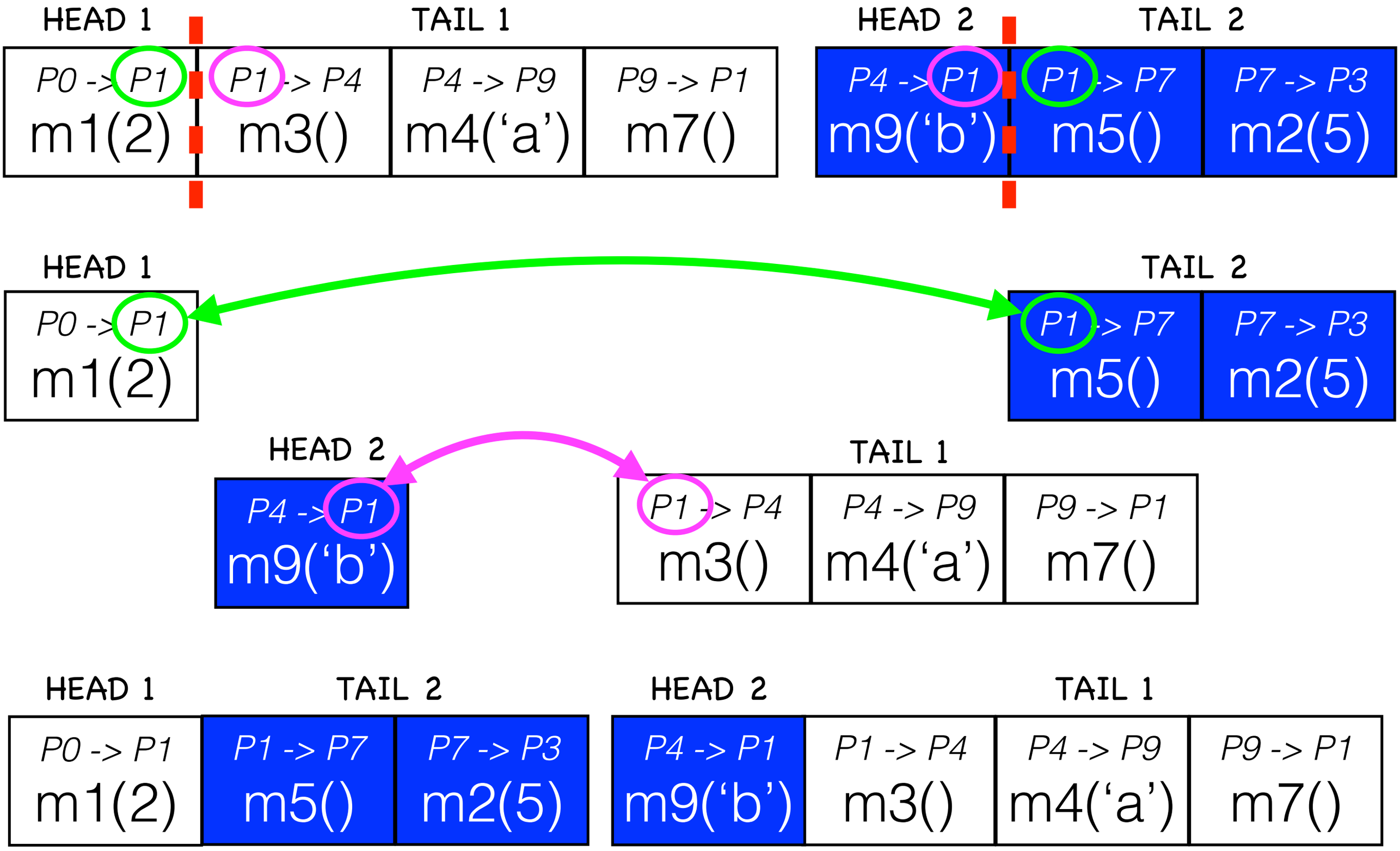
Chromosome:

P0 -> P1	*P1 -> P4*	*P4 -> P9*	*P9 -> P1*
m1(2)	m3()	m4(‘a’)	m7()

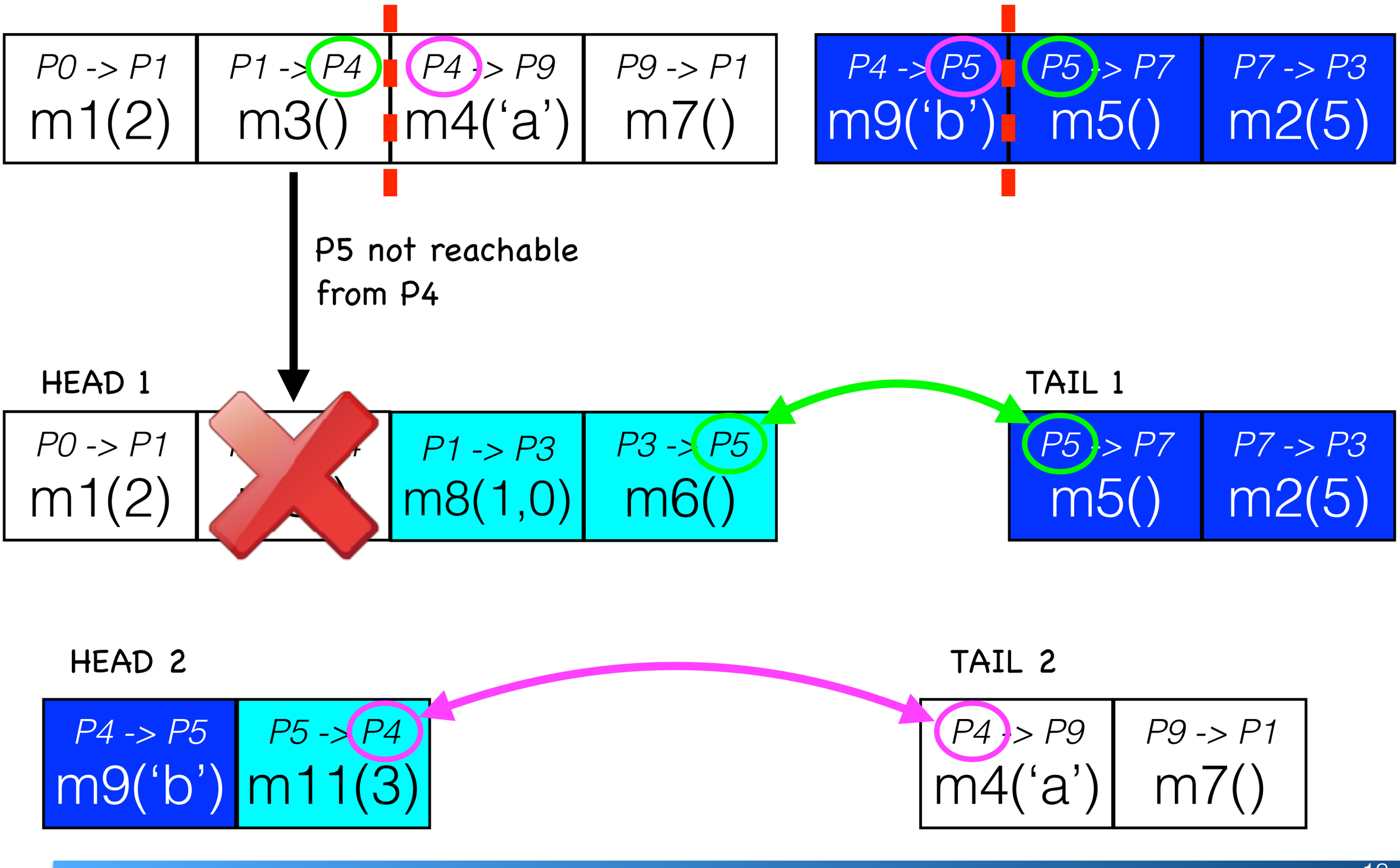
Fitness function: distance from a yet uncovered navigation method. Requires test case execution by Selenium on browser.

Genetic operators: crossover and mutation, applied only to feasible path prefixes

Crossover: matching tail



Crossover: non-matching tail



Mutation: delete and insert

Starting chromosome

P0 -> P1	*P1 -> P4*	*P4 -> P9*	*P9 -> P1*	*P4 -> P11*
m1(2)	m3()	m4('a')	m7()	m8(1,0)

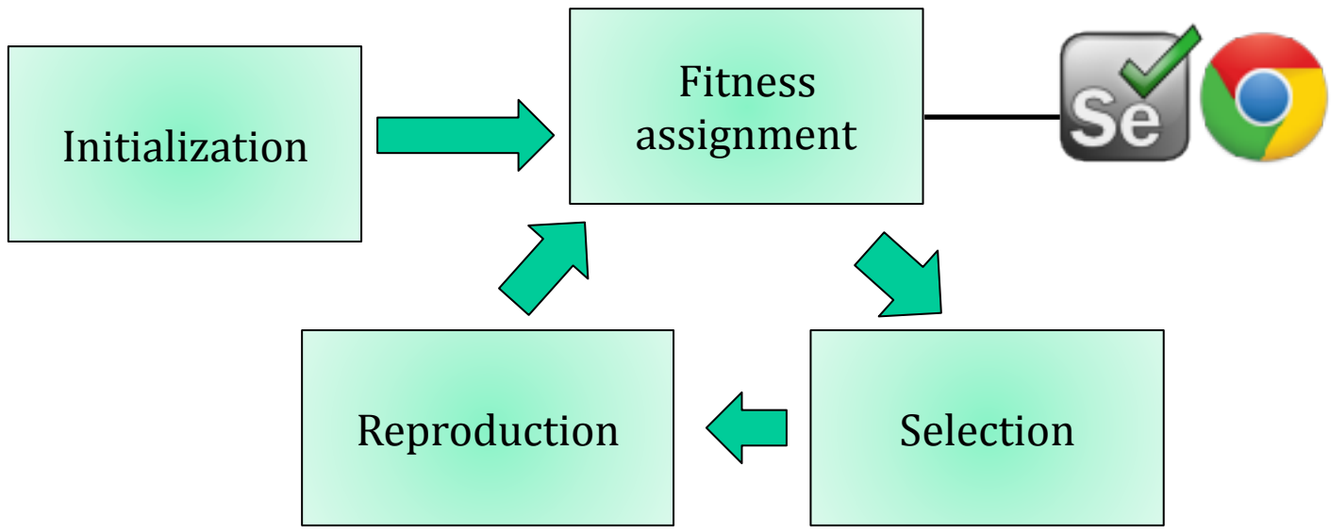
Delete

P0 -> P1	*P1 -> P4*			*P4 -> P11*
m1(2)	m3()			m8(1,0)

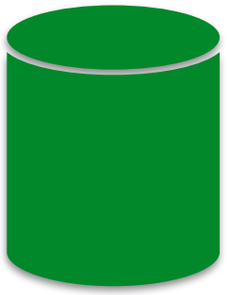
Insert

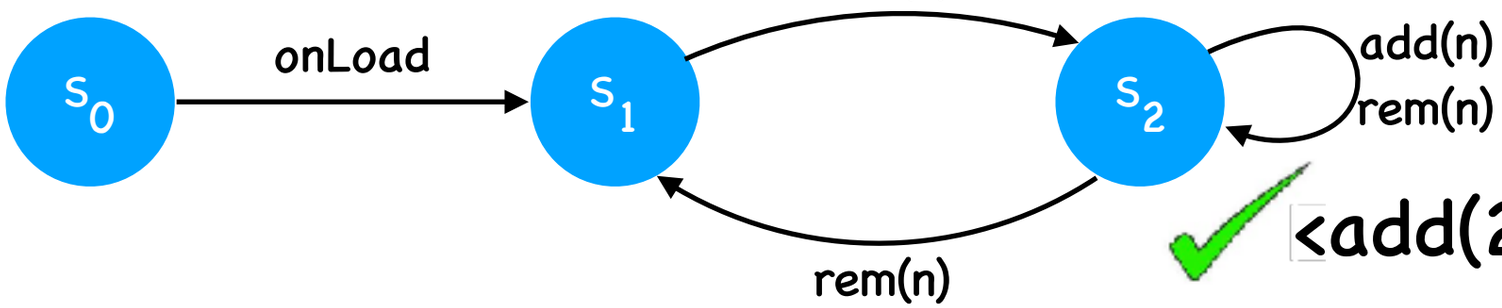
P0 -> P1	*P1 -> P4*	*P4 -> P11*	*P11 -> P8*	*P8 -> P7*	*P7 -> P12*
m1(2)	m3()	m8(1,0)	m5()	m6()	m9('b')

Search based generator



keep non diverging prefixes that increase coverage





✓ <add(2), add(1), rem(1)>

✓ <add(2), rem(2)>

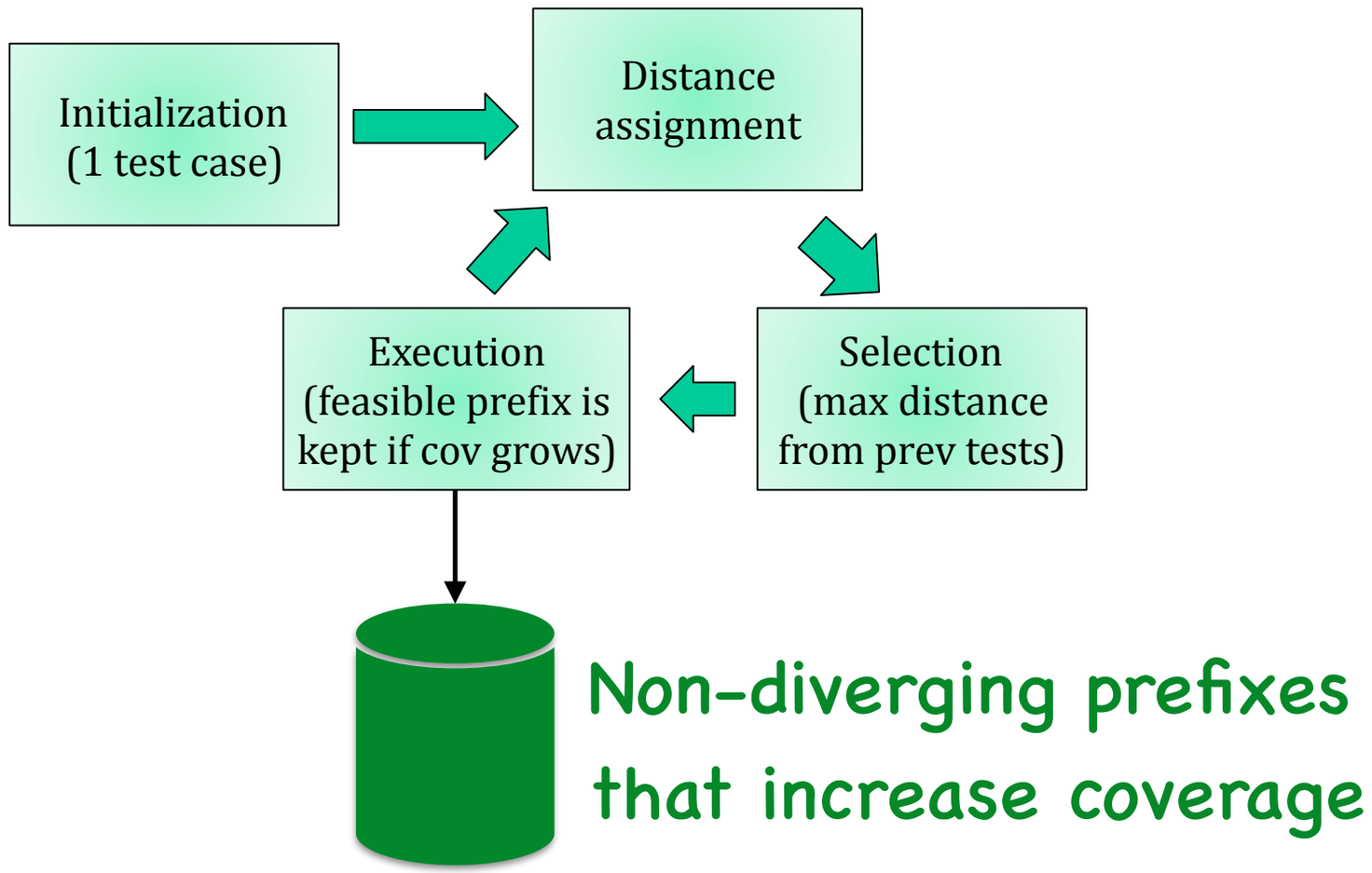
By construction, only non diverging navigation paths are kept

Evolving a population of navigation paths requires many test case executions

Diversity based generator

$$d(T_i, T_j) = D + \sum_k \frac{id_k}{1 + id_k}$$

D: sequence edit distance
id: input distance



Diversity based generator

$$d(T_i, T_j) = D + \sum_k \frac{id_k}{1 + id_k}$$

Randomly generated candidates

P0 -> P1	*P1 -> P4*	*P4 -> P9*	
m1(1)	m5()	m4('a')	)

min d = 2 + 1/2

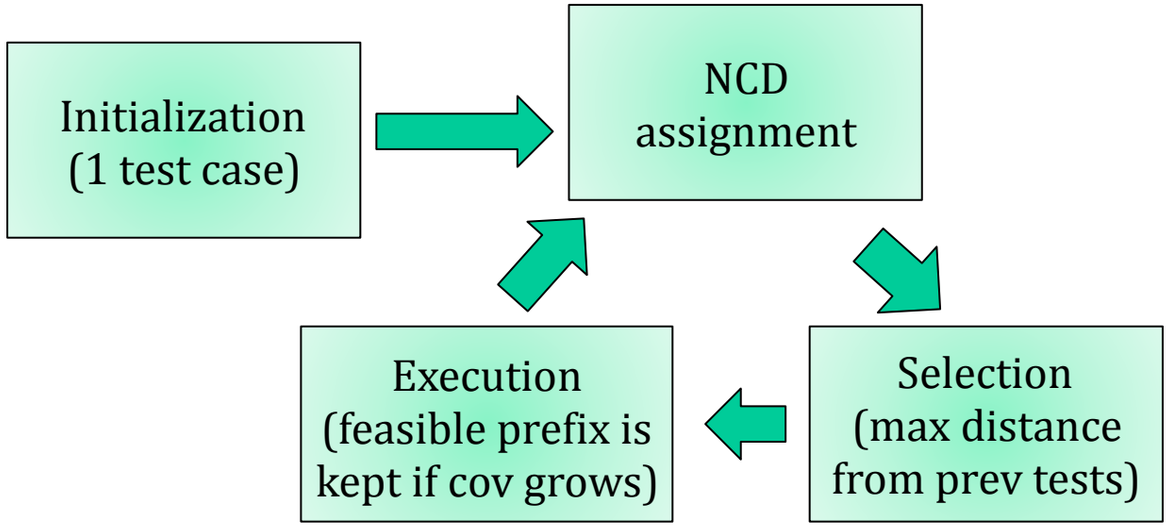
Select candidate at **max min distance** from previously generated tests and execute it

Previously generated tests

P0 -> P1	*P1 -> P4*	*P4 -> P9*	
m1(6)	m8()	m9('z')	
P0 -> P1	*P1 -> P4*	*P4 -> P9*	*P9 -> P1*
m1(2)	m3()	m4('a')	m7()

Several candidates are evaluated without being executed; only the selected candidate is executed

Kolmogorov complexity



Conditional Kolmogorov Complexity $K(t_1|t_2)$: for a string (test case) t_1, number of bits of the shortest program $P(t_2)$ that generates t_1.

Normalized Information Distance $ID(t_1, t_2)$: given two strings (test cases) t_1, t_2:

$$NID(t_1, t_2) = max(K(t_1|t_2), K(t_2|t_1)) / max(K(t_1), K(t_2))$$

Normalized Compression Distance $NCD(t_1, t_2)$: given two strings (test cases) t_1, t_2:

$$NCD(t_1, t_2) = (C(t_1 \cdot t_2) - min(C(t_1), C(t_2))) / max(C(t_1), C(t_2))$$

Preliminary results using SB

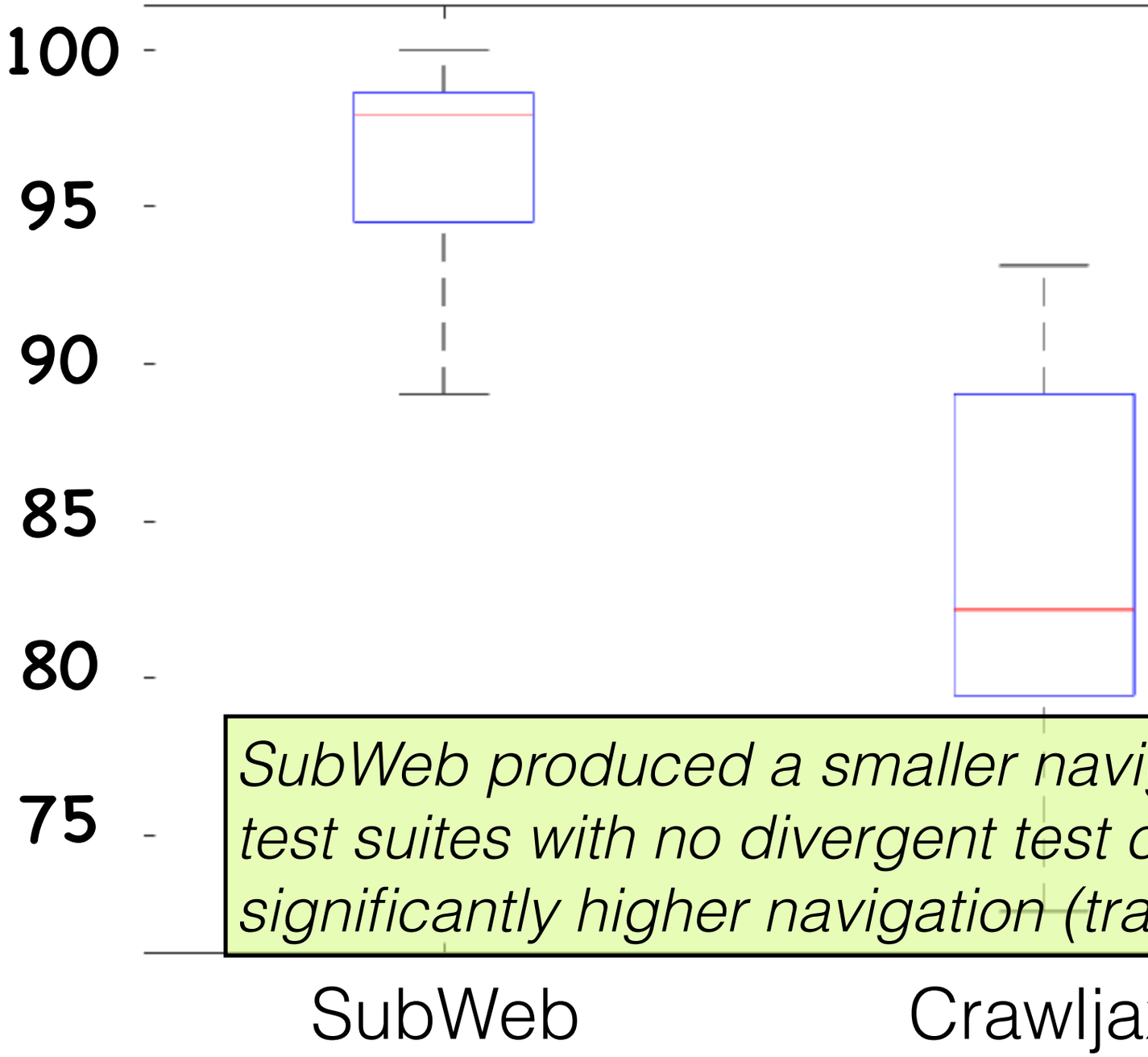
AddressBook

App	PHP LOC	30223
	JavaScript LOC	1288
POs	LOC	764
	Total number	13
	Navig. methods	73
Preconds	Method LOC	75
	Total number	16
	Logic operators	54

PO graph	States	12
	Transitions	73
Crawled graph	States	329
	Transitions	927
	Missing states	0
	Missing trans	5
	Split state ratio	27
	Split trans ratio	13

Coverage

SUBWEB	Test cases	54
Ext-Crawljax	Test cases Divergent test cases	598 104 (17%)



SUBWEB	
Mean	96%
Var	12%

Ext-Crawljax	
Mean	83%
Var	39%

Mann-Whitney U test	
p-value	$4.70 \cdot 10^{-4}$

SubWeb produced a smaller navigation graph and smaller test suites with no divergent test case, and it achieved significantly higher navigation (transition) coverage

Conclusion and future work

- **Search based generation** of web test cases outperforms crawling by reducing divergence and increasing coverage, but it requires execution of all candidates within a browser to select the fittest candidate.
- **Diversity based on edit/input distance** does not require test execution to evaluate the fitness of candidates:
  - it is more efficient than search based generation;
  - but it does not address divergence directly;
  - preliminary results are very encouraging.
- **Diversity based on information distance**, approximated by NCD (Normalized Compression Distance), is promising (e.g., it is potentially less sensitive to repetitions than edit distance), but we do not have empirical results yet: it will be investigated in our future work.