Achieving Scalability in Software Testing with Machine Learning and Metaheuristic Search

Lionel Briand

RAISE @ ICSE 2018

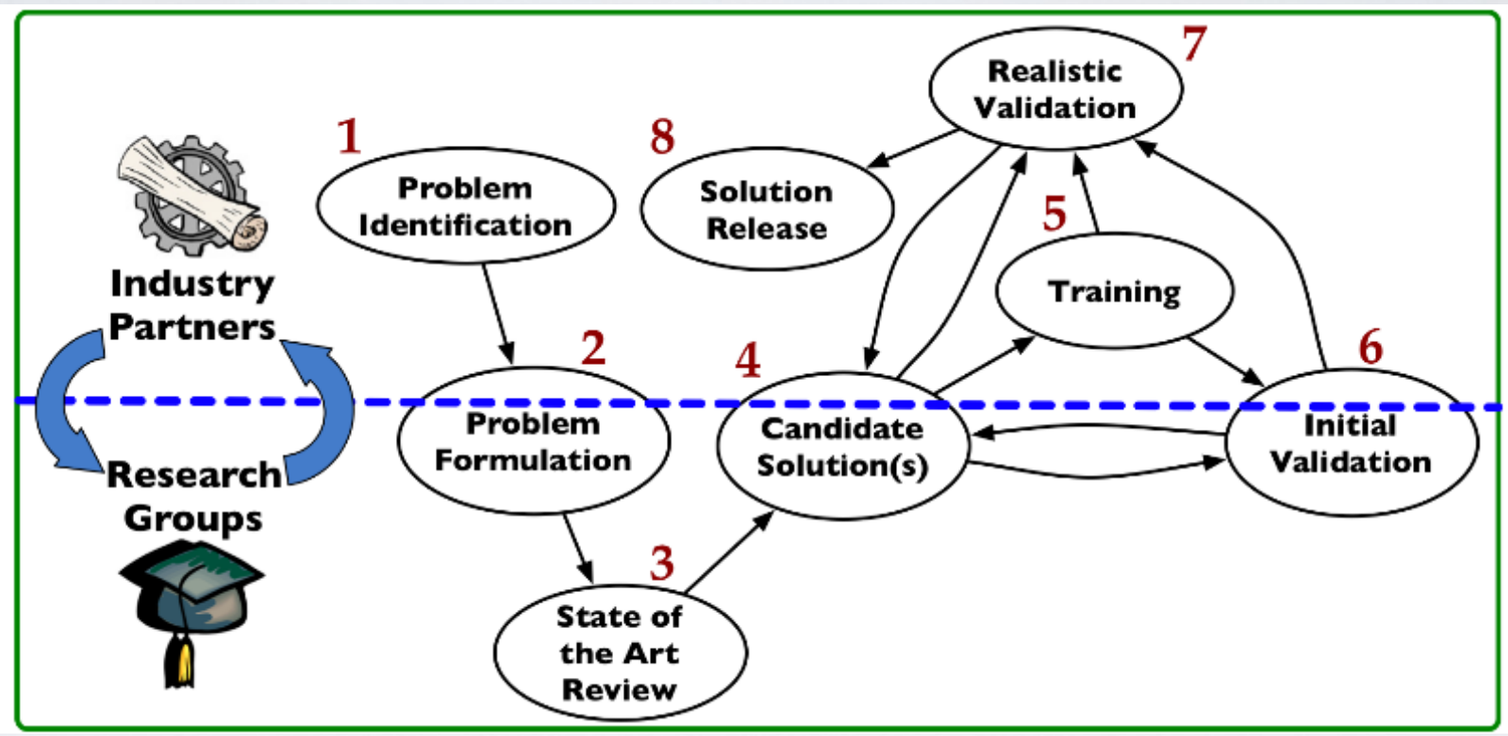
Scope

- The main challenge in testing software systems is **scalability**
- Addressing scalability entails effective **automation**
- **Lessons learned** from industrial research collaborations: satellite, automotive, finance, energy …
- **Experiences** from combining metaheuristic search, machine learning, and other AI techniques, in addressing testing scalability

Scalability

- **The extent to which a technique can be applied on large or complex artifacts (e.g., input spaces, code, models) and still provide useful, automated support with acceptable effort, CPU, and memory?**

Collaborative Research @ SnT



- Research in context
- Addresses actual needs
- Well-defined problem
- Long-term collaborations
- Our lab is the industry

SVV Dept.

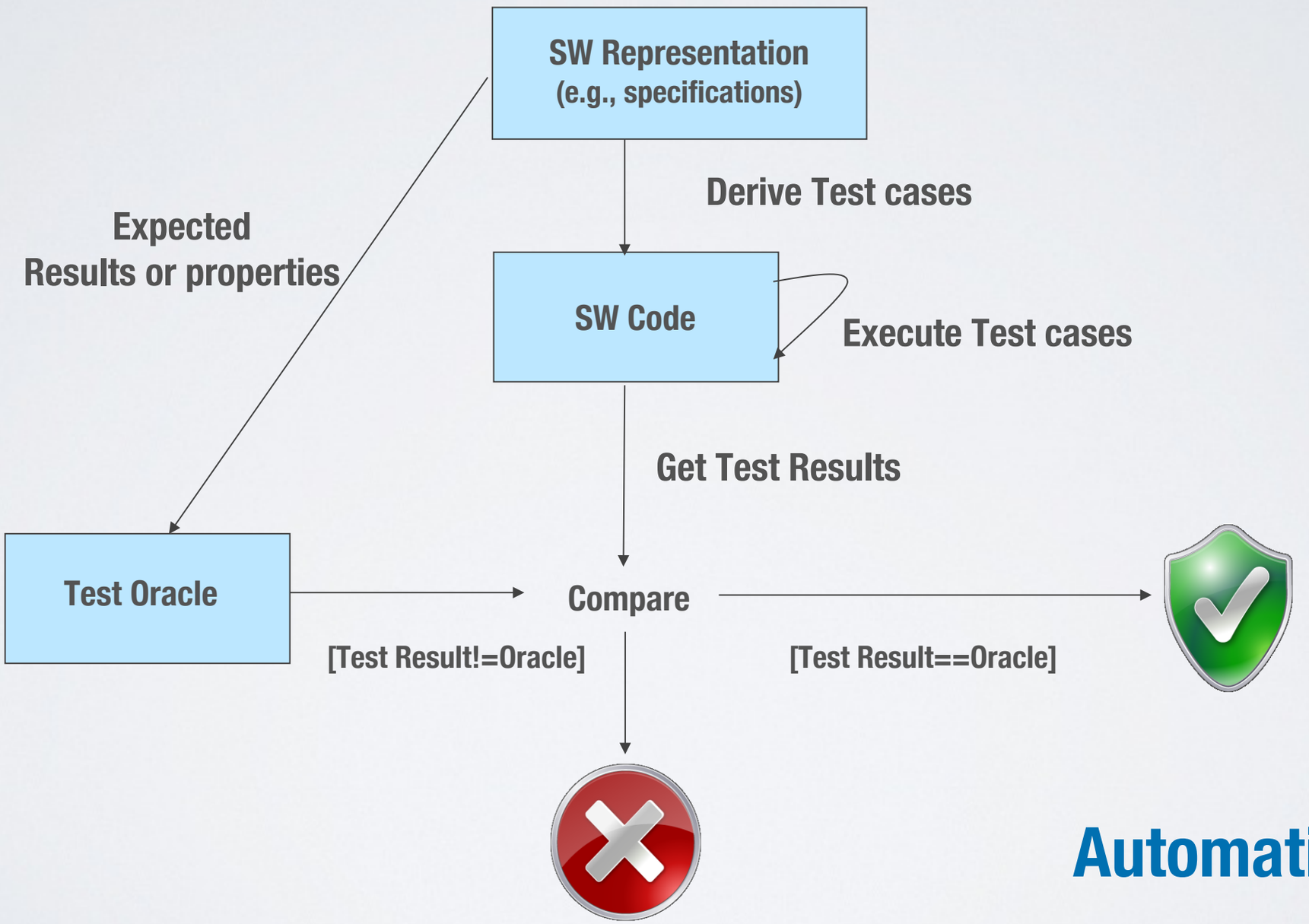
- Established in 2012, part of the SnT centre
- Requirements Engineering, Security Analysis, Design Verification, Automated Testing, Runtime Monitoring
- ~ 25 lab members
- Partnerships
- ERC Advanced grant

Outline

- **Overview, problem definition**
- **Example research projects with industry partners:**
  - **Testing advanced driver assistance systems**
  - **Testing controllers (automotive)**
  - **Stress testing critical task deadlines (Energy)**
  - **Vulnerability testing (Banking)**
- **Reflections and lessons learned**

Introduction

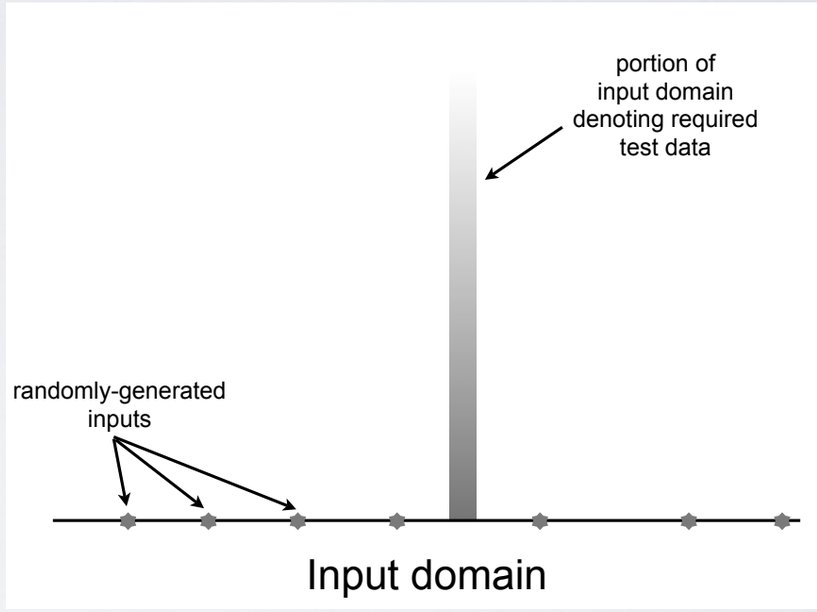
Software Testing

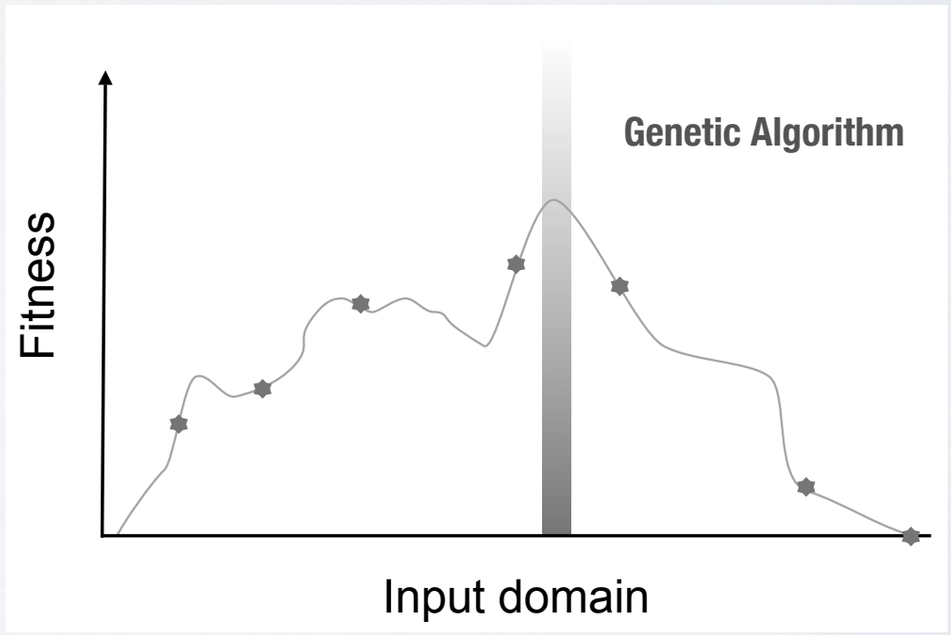


Automation!

Search-Based Software Testing

- Express test generation problem as a **search problem**
- Search for **test input data** with certain properties, i.e., constraints
- **Non-linearity** of software (if, loops, …): complex, discontinuous, non-linear search spaces (Baresel)
- Many search algorithms **(metaheuristics)**, from local search to global search, e.g., Hill Climbing, Simulated Annealing and Genetic Algorithms

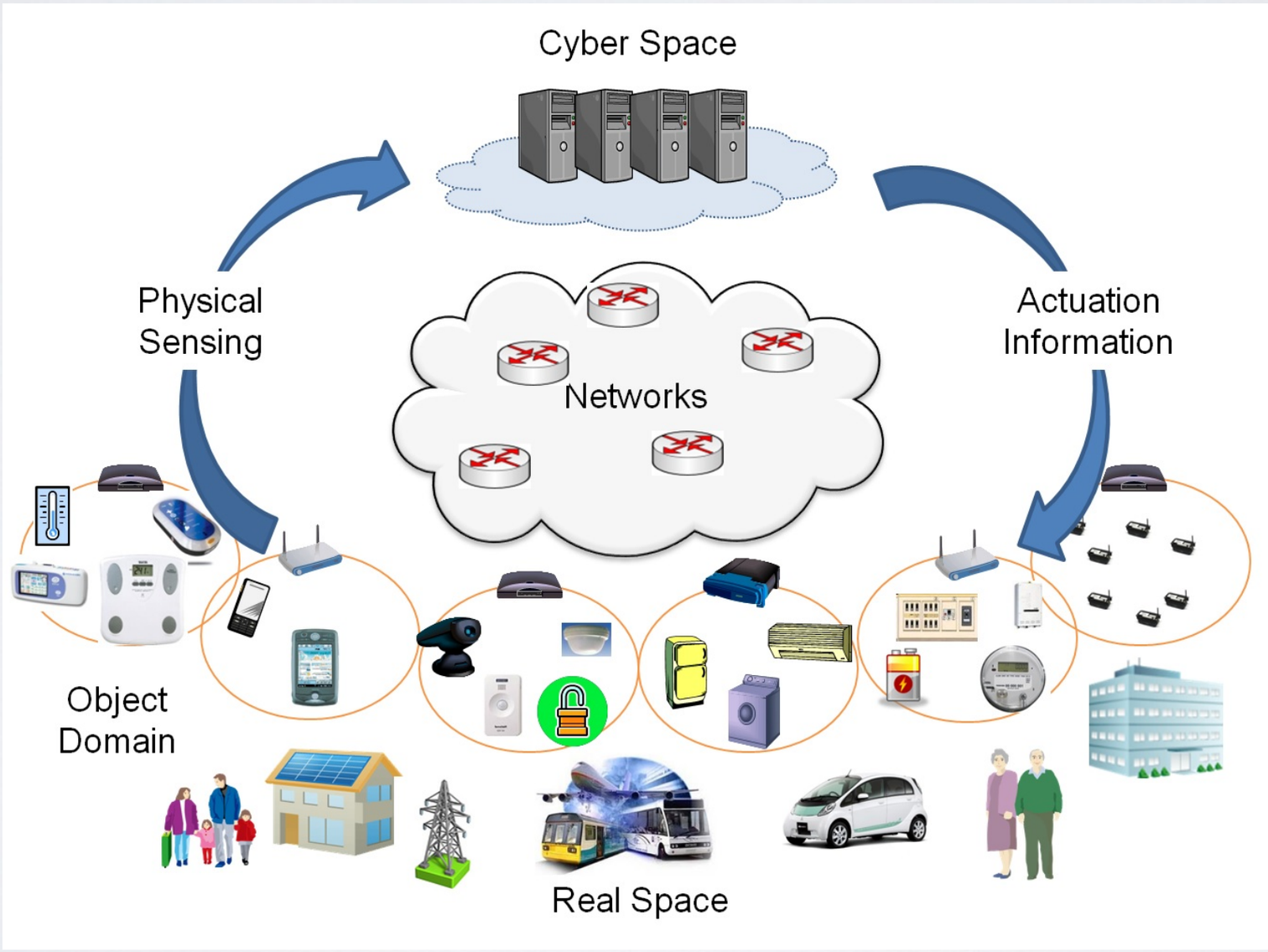




Testing Advanced Driving Assistance Systems

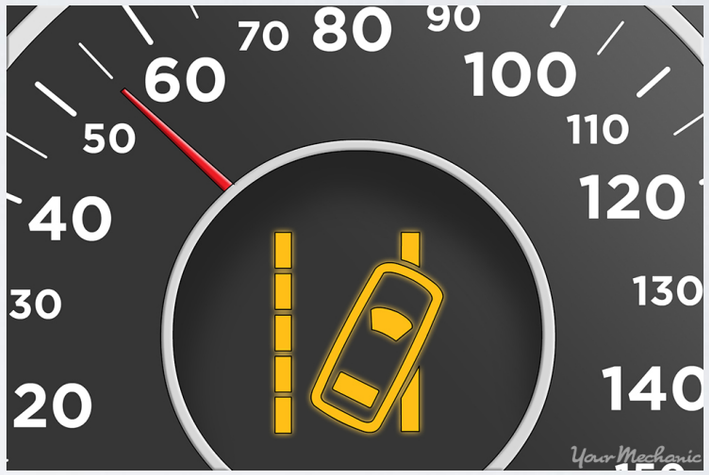
Cyber-Physical Systems

- **A system of collaborating computational elements controlling physical entities**



Advanced Driver Assistance Systems (ADAS)

Automated Emergency Braking (AEB)



Lane Departure Warning (LDW)

Pedestrian Protection (PP)

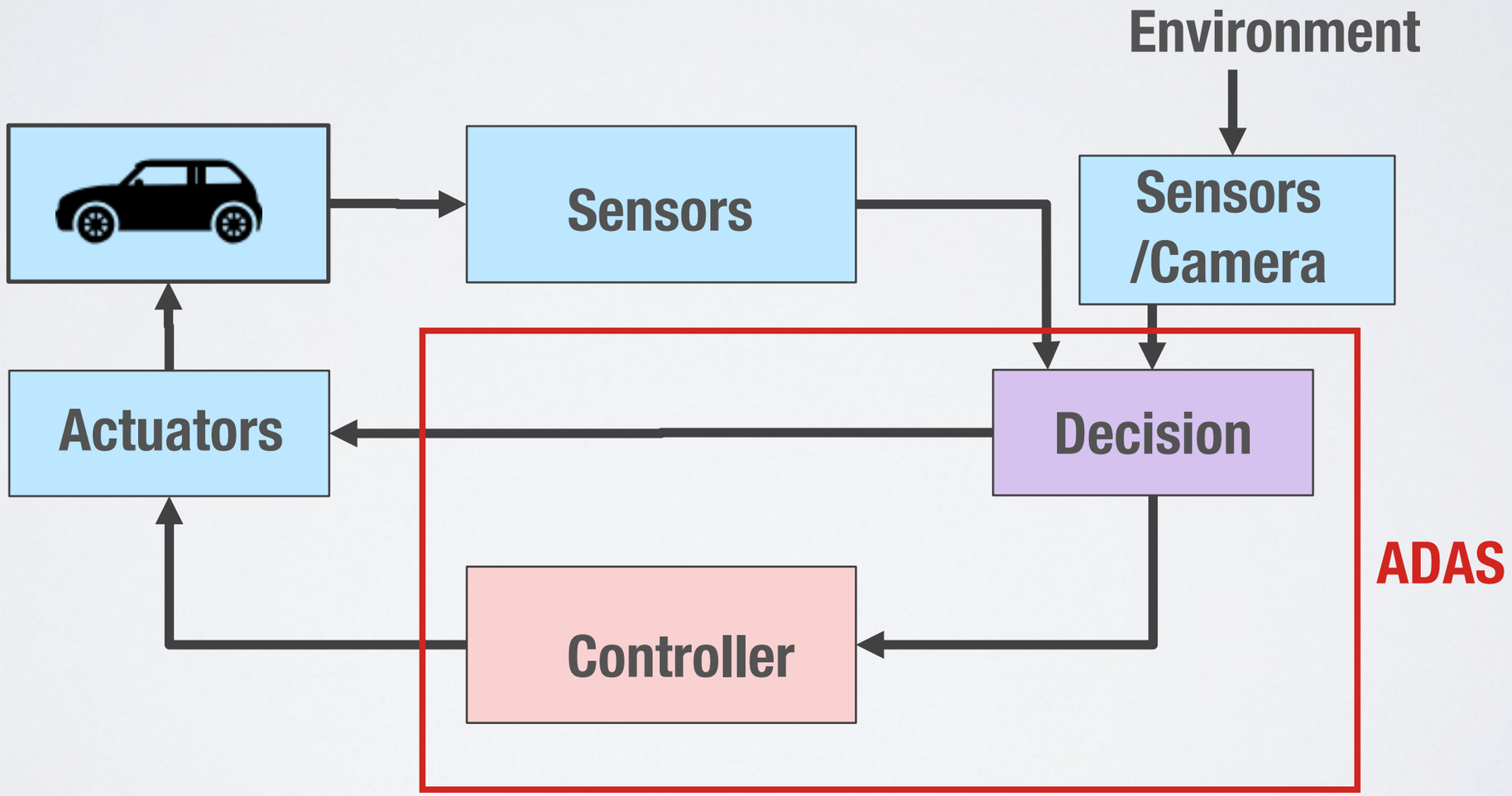
Traffic Sign Recognition (TSR)

Automotive Environment

- **Highly varied environments, e.g., road topology, weather, building and pedestrians …**
- **Huge number of possible scenarios, e.g., determined by trajectories of pedestrians and cars**
- **ADAS play an increasingly critical role**
- **A challenge for testing**

Advanced Driver Assistance Systems (ADAS)

Decisions are made over time based on sensor data



A General and Fundamental Shift

- Increasingly so, it is easier to learn behavior from data using machine learning, rather than specify and code

- Deep learning, reinforcement learning …

- Example: Neural networks (deep learning)

- Millions of weights learned

- No explicit code, no specifications

- Verification, testing?

CPS Development Process

Model-in-the-Loop Stage

Functional modeling:
- **Controllers**
- **Plant**
- **Decision**

Continuous and discrete Simulink models

Model simulation and testing

MATLAB SIMULINK

Software-in-the-Loop Stage

Architecture modelling
- **Structure**
- **Behavior**
- **Traceability**

System engineering modeling (SysML)

Analysis:
- **Model execution and testing**
- **Model-based testing**
- **Traceability and change impact analysis**
- **...**

(partial) Code generation

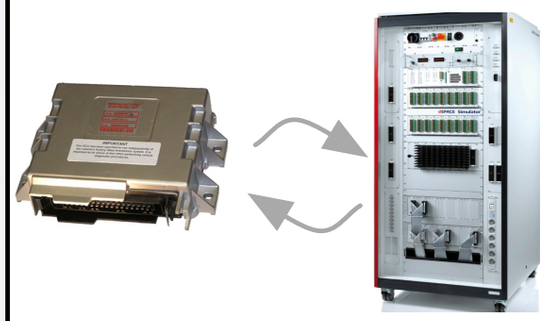
C SysML

Hardware-in-the-Loop Stage

Deployed executables on target platform

Hardware (Sensors ...)
Analog simulators

Testing (expensive)



Automotive Environment

- **Highly varied environments, e.g., road topology, weather, building and pedestrians …**
- **Huge number of possible scenarios, e.g., determined by trajectories of pedestrians and cars**
- **ADAS play an increasingly critical role**
- **A challenge for testing**

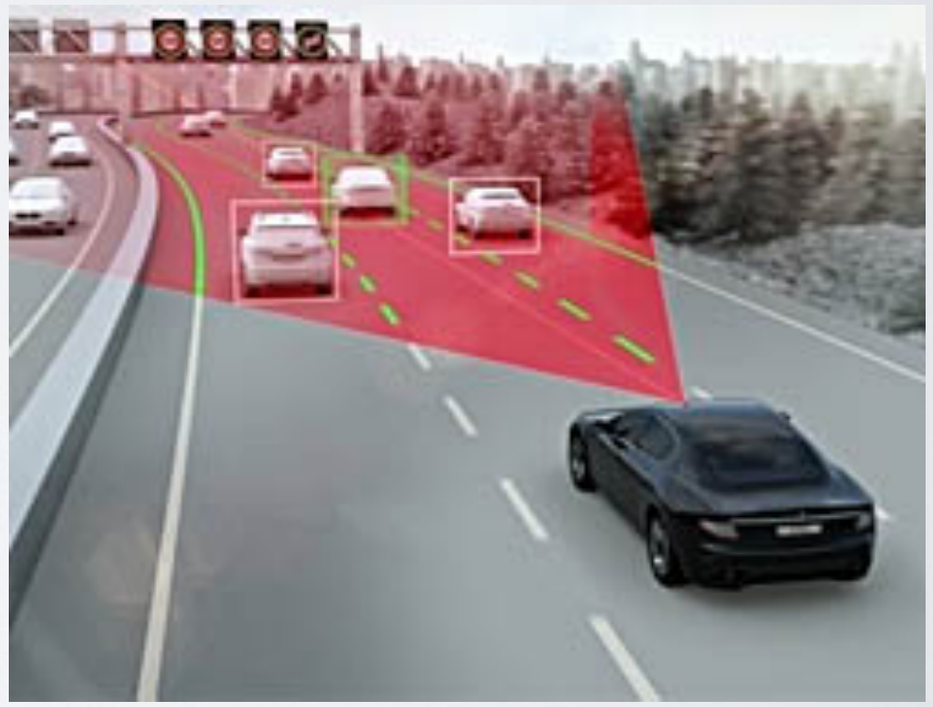
Our Goal

- **Developing an automated testing technique for ADAS**

- **To help engineers efficiently and effectively explore the complex test input space of ADAS**

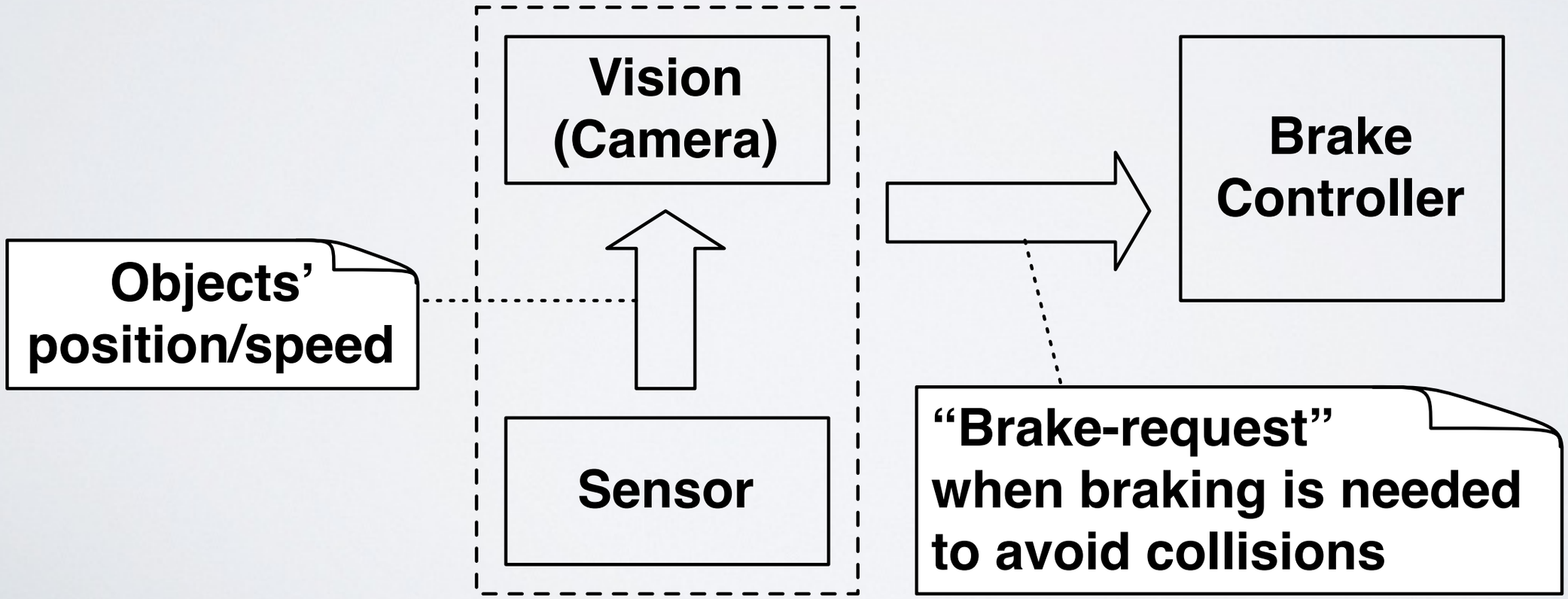
- **To identify critical (failure-revealing) test scenarios**

- **Characterization of input conditions that lead to most critical situations, e.g., safety violations**



Automated Emergency Braking System (AEB)

Decision making



Example Critical Situation

- **“AEB properly detects a pedestrian in front of the car with a high degree of certainty and applies braking, but an accident still happens where the car hits the pedestrian with a relatively high speed”**

Testing ADAS

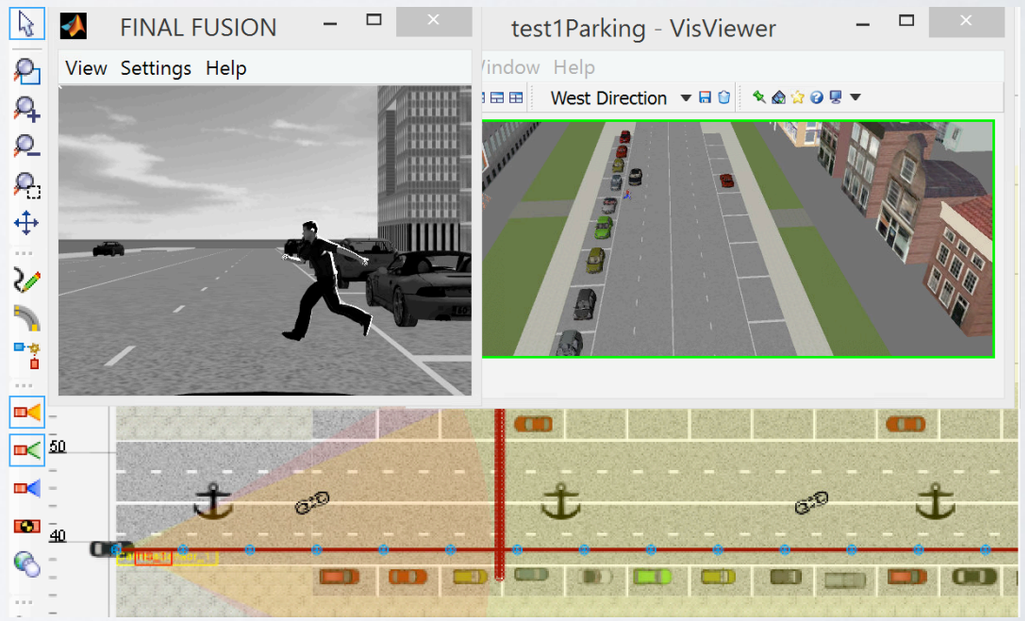
On-road testing

Time-consuming

Expensive

Unsafe

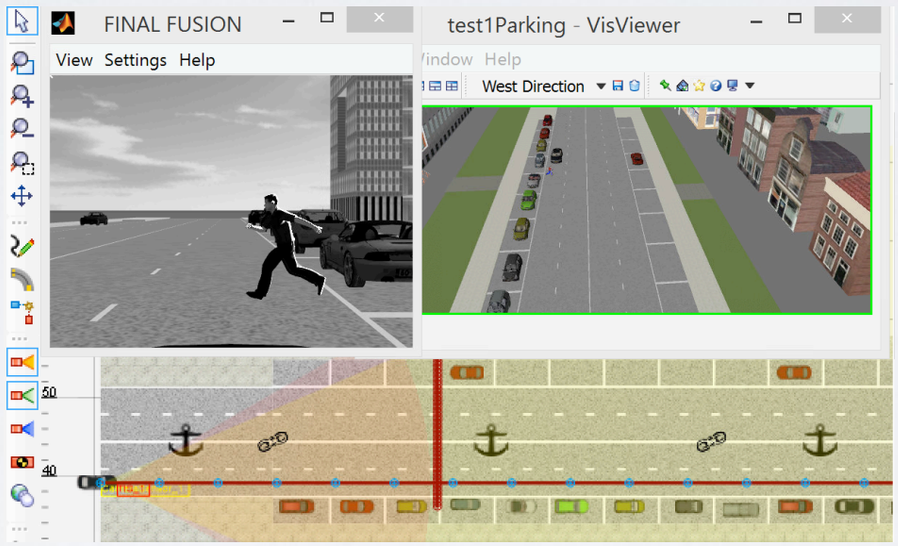
Simulation-based (model) testing



A simulator based on Physical/Mathematical models

Testing via Physics-based Simulation

Simulator (Matlab/Simulink)



Model (Matlab/Simulink)

ADAS (SUT)

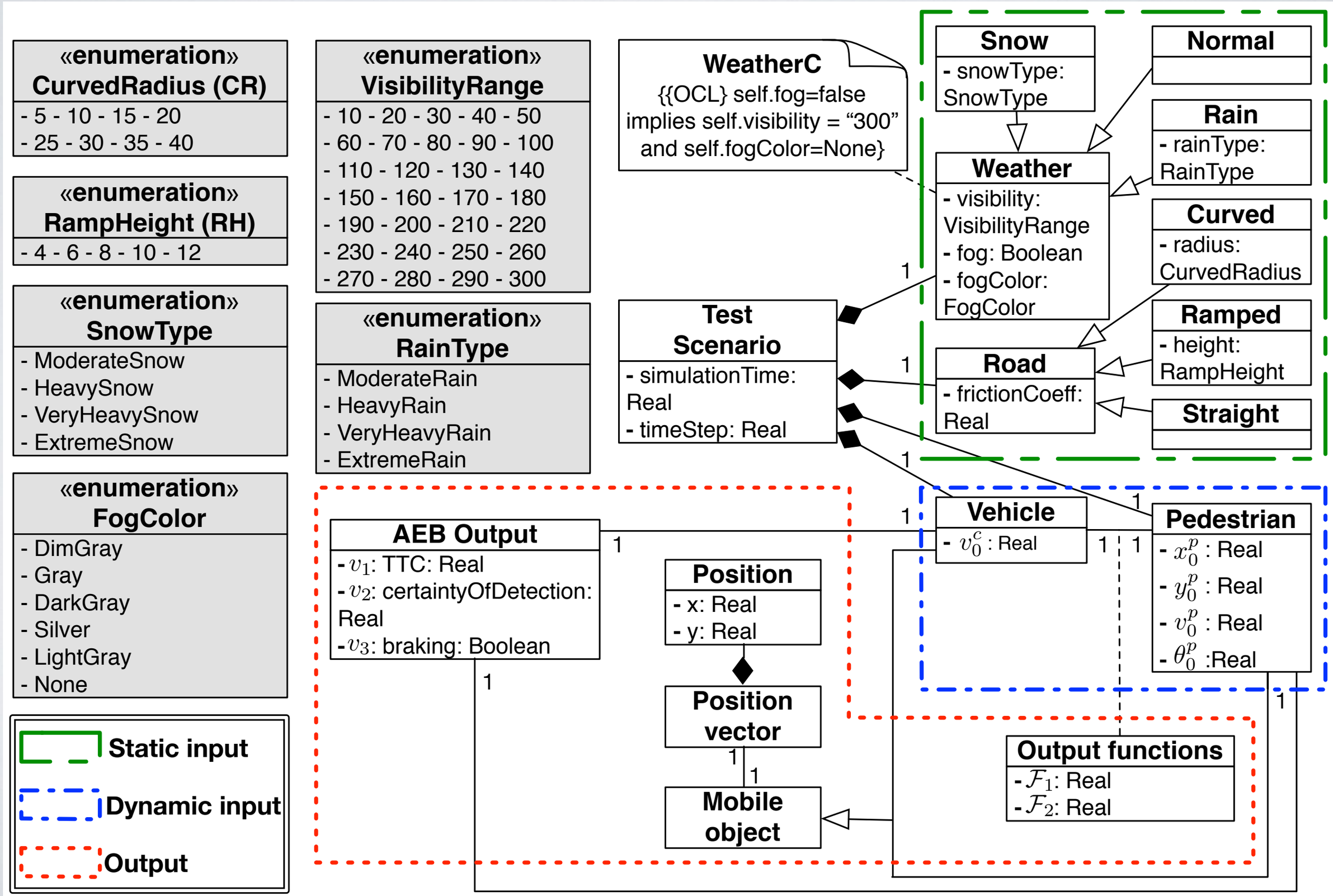
- Physical plant (vehicle / sensors / actuators)
- Other cars
- Pedestrians
- Environment (weather / roads / traffic signs)

Test input

Test output

time-stamped output

AEB Domain Model



ADAS Testing Challenges

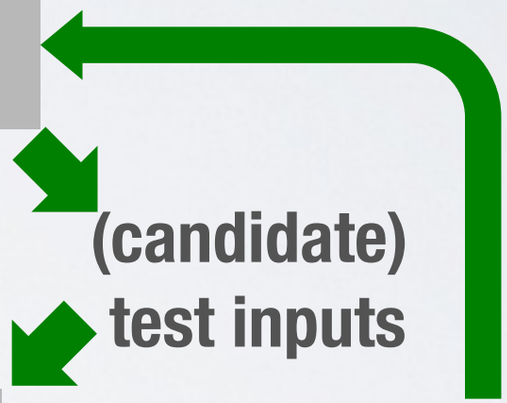
- Test input space is large, complex and multidimensional
- Explaining failures and fault localization are difficult
- Execution of physics-based simulation models is computationally expensive

Black-Box Search-based Testing

Input data ranges/dependencies + Simulator + Fitness functions defined based on Oracles

Test input generation (NSGA II)

- Select best tests
- Generate new tests

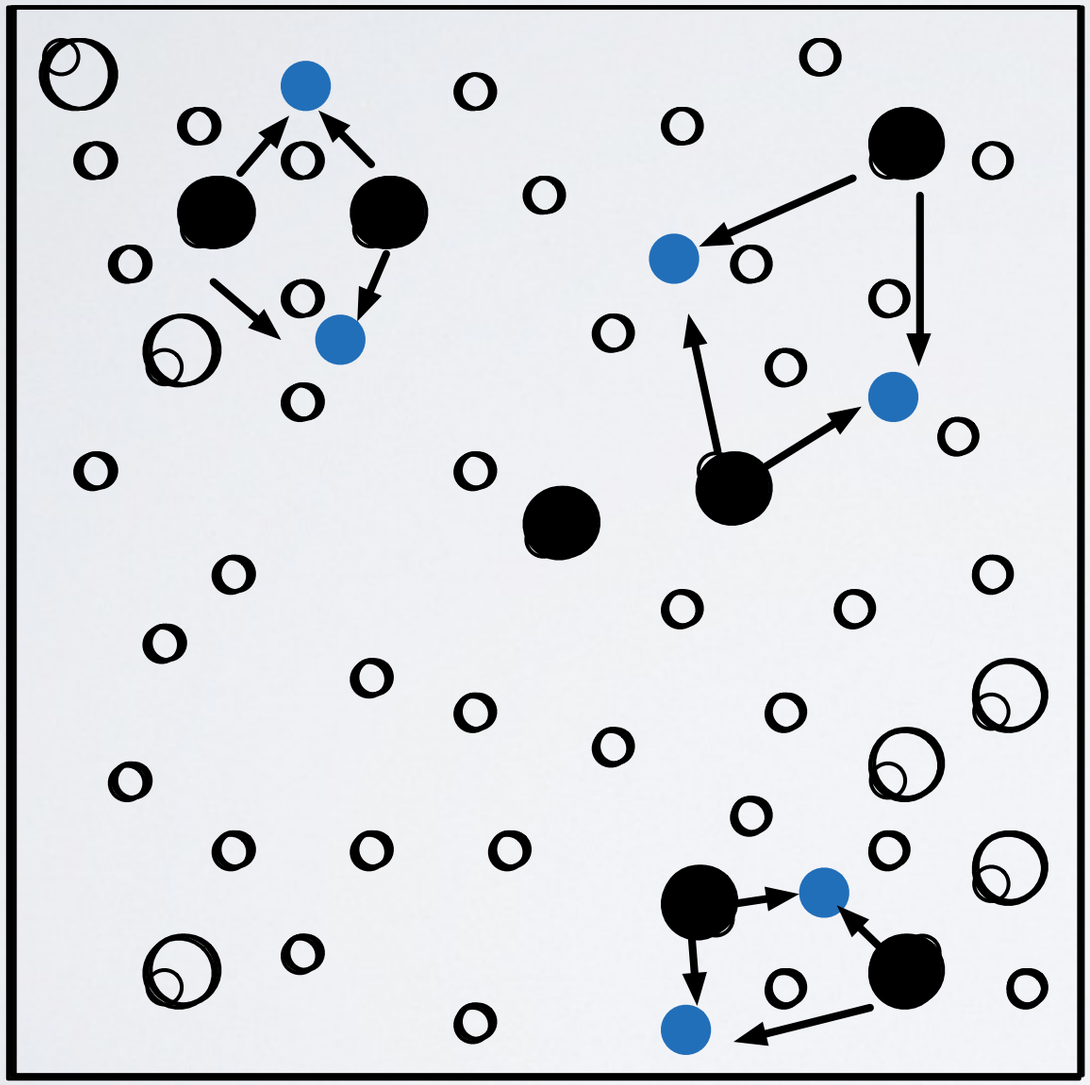


Evaluating test inputs

- Simulate every (candidate) test
- Compute fitness functions

Test cases revealing worst case system behaviors

Search: Genetic Evolution



~~Initial input~~

~~Fitness computation~~

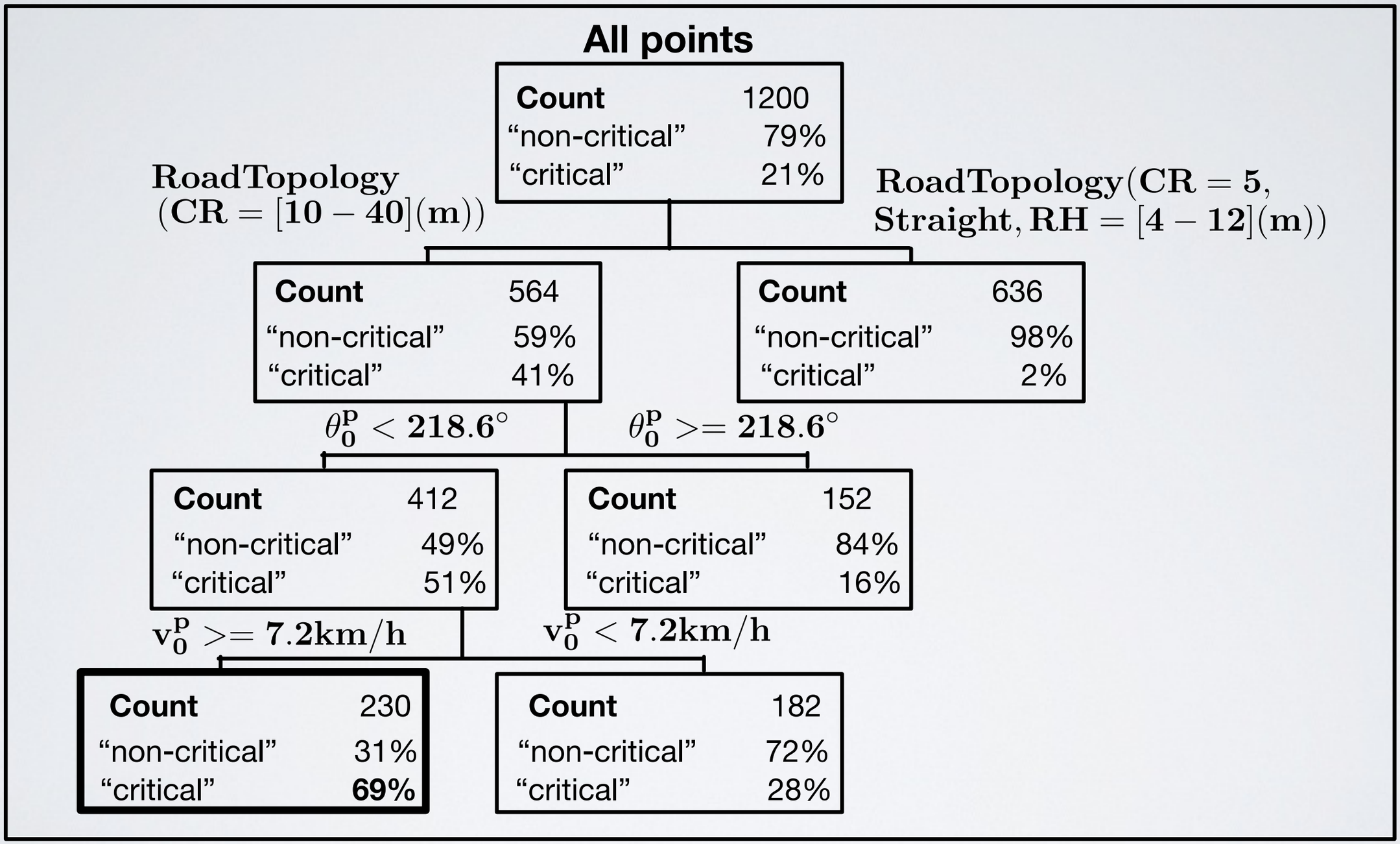
~~Selection~~

Breeding

Better Guidance

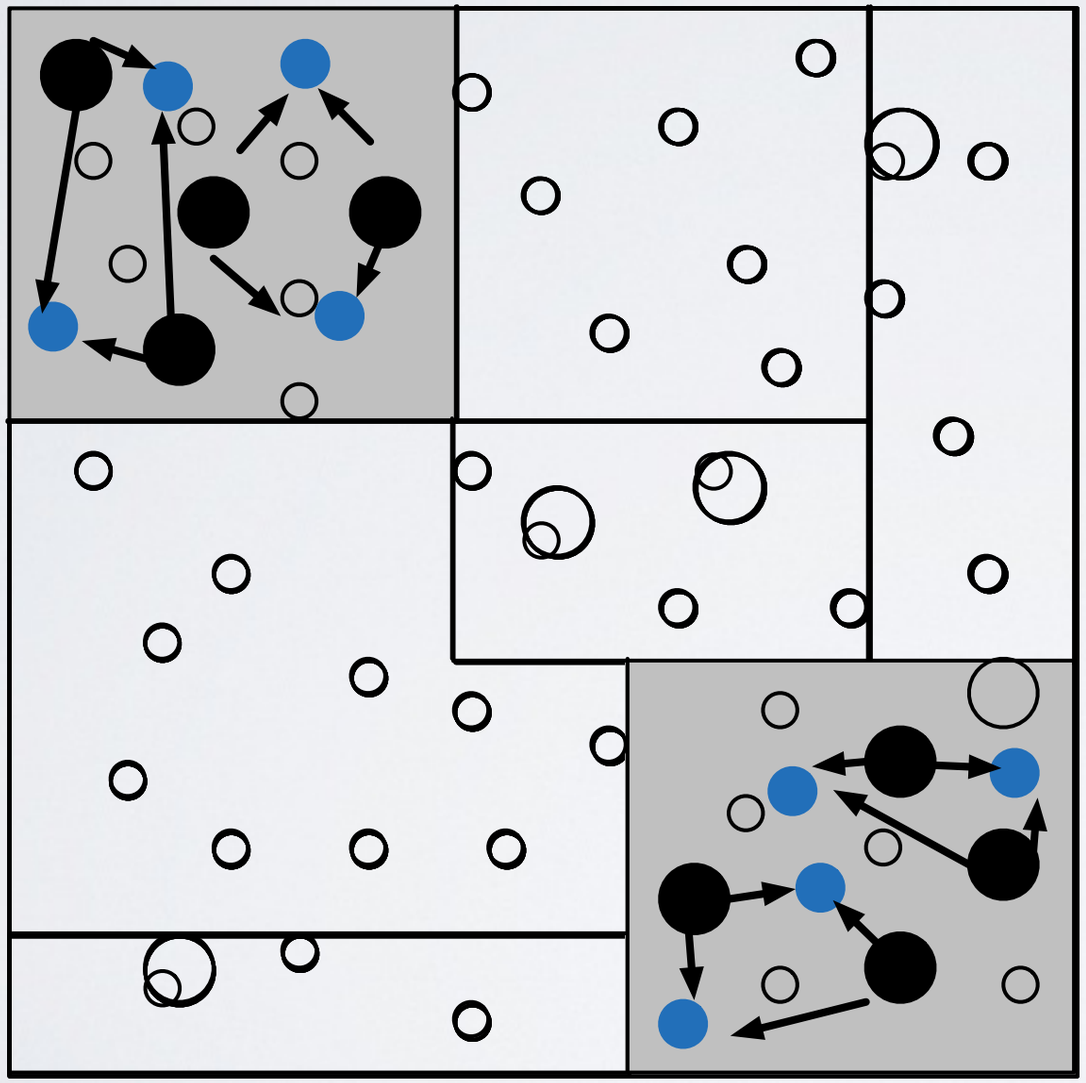
- Fitness computations rely on simulations and are very expensive
- Search needs better guidance

Decision Trees



Partition the input space into homogeneous regions

Genetic Evolution Guided by Classification



~~Initial input~~

~~Fitness computation~~

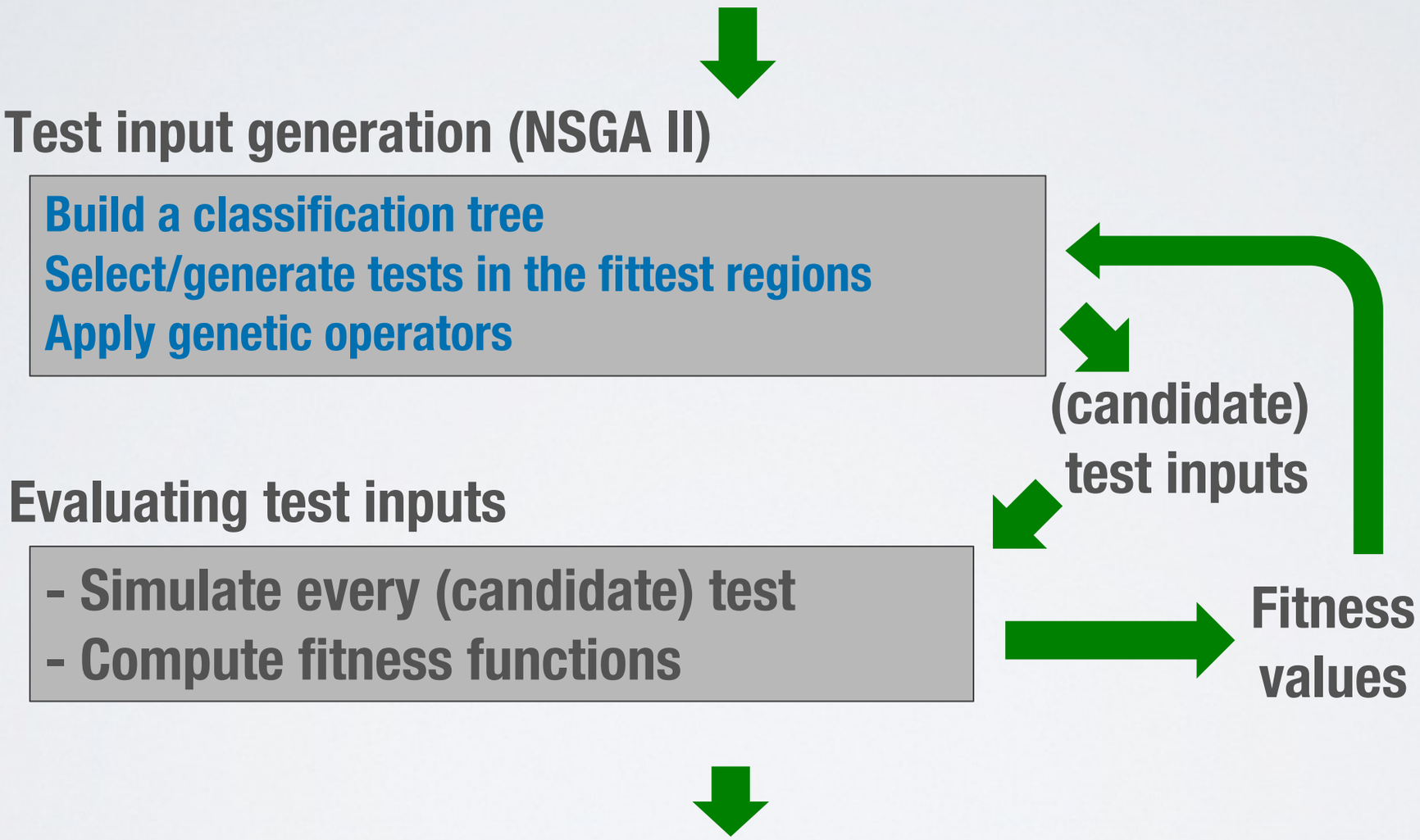
~~Classification~~

~~Selection~~

Breeding

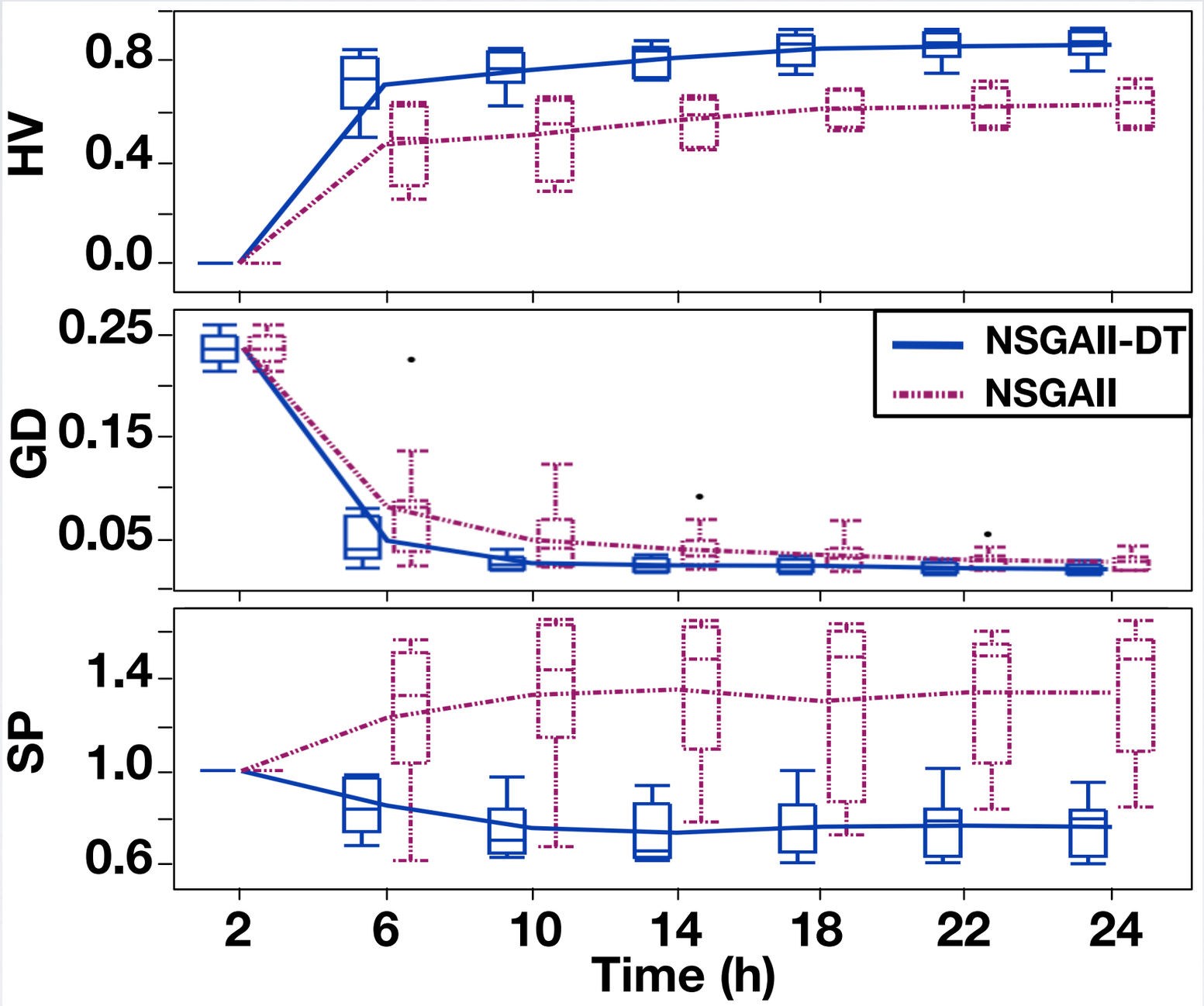
Search Guided by Classification

Input data ranges/dependencies + Simulator + Fitness functions defined based on Oracles



Test cases revealing worst case system behaviors +
A characterization of critical input regions

NSGAII-DT vs. NSGAII

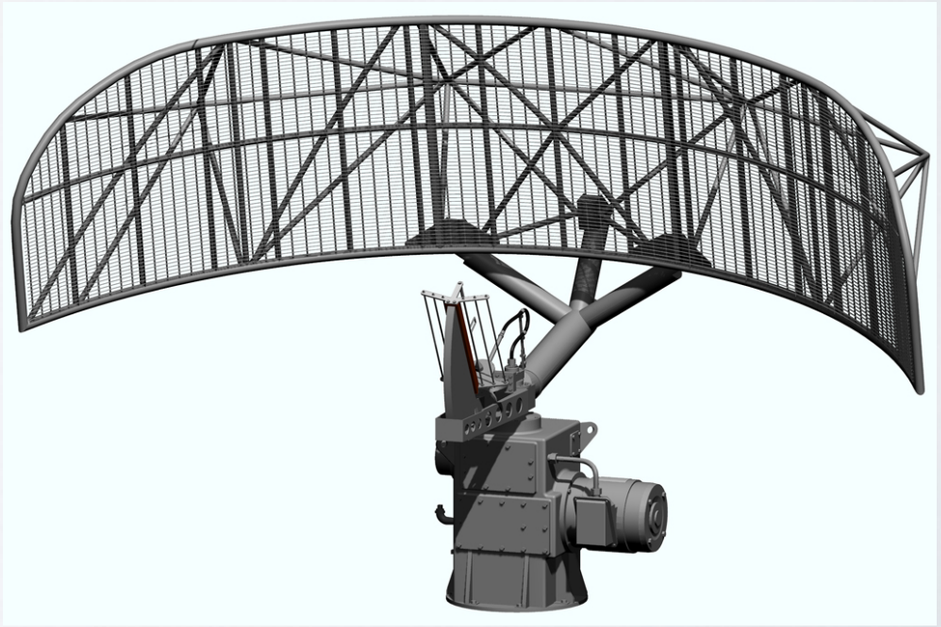


NSGAII-DT outperforms NSGAII

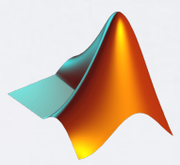
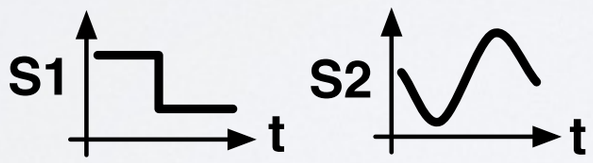
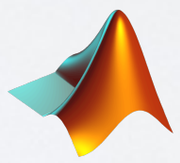
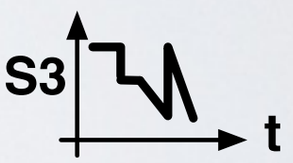
Testing Controllers

DELPHI

Dynamic Continuous Controllers



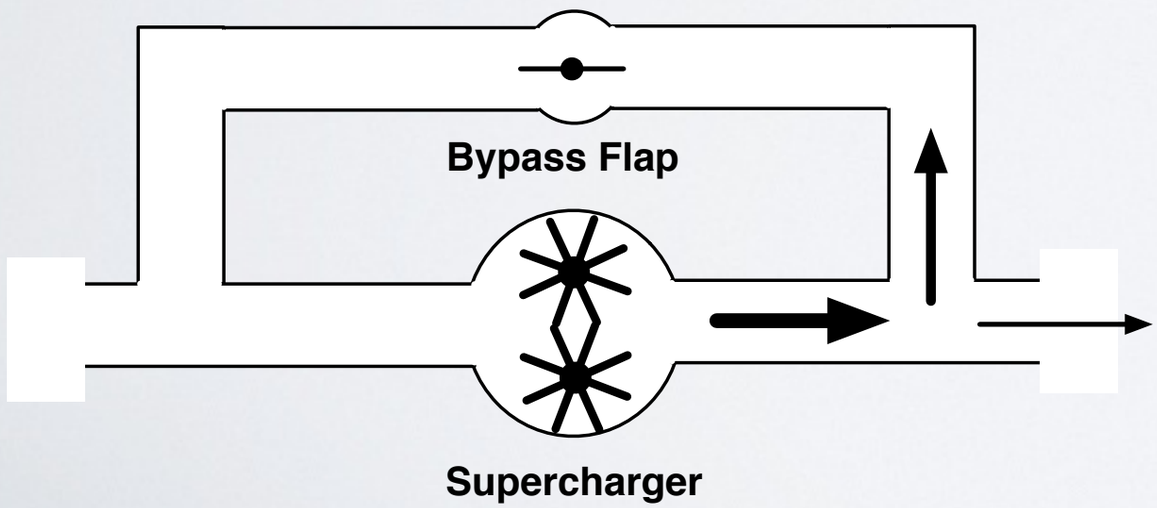
MiL Test Cases

	Input Signals	Model Simulation	Output Signal(s)
Test Case 1			
Test Case 2			
⋮	⋮	⋮	⋮

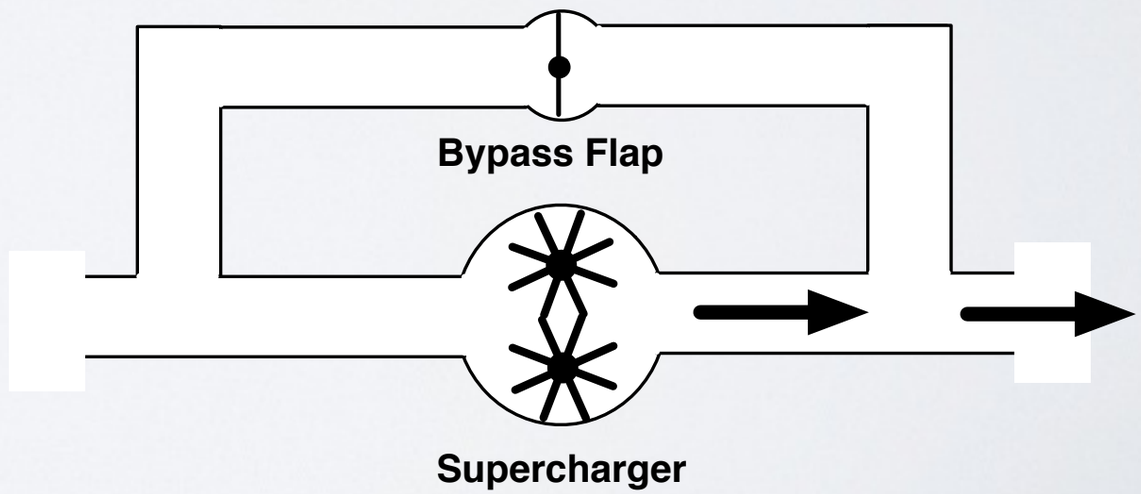
Simple Example

- Supercharger bypass flap controller
  - ✓Flap position is bounded within [0..1]
  - ✓Implemented in MATLAB/Simulink
  - ✓34 (sub-)blocks decomposed into 6 abstraction levels

DELPHI

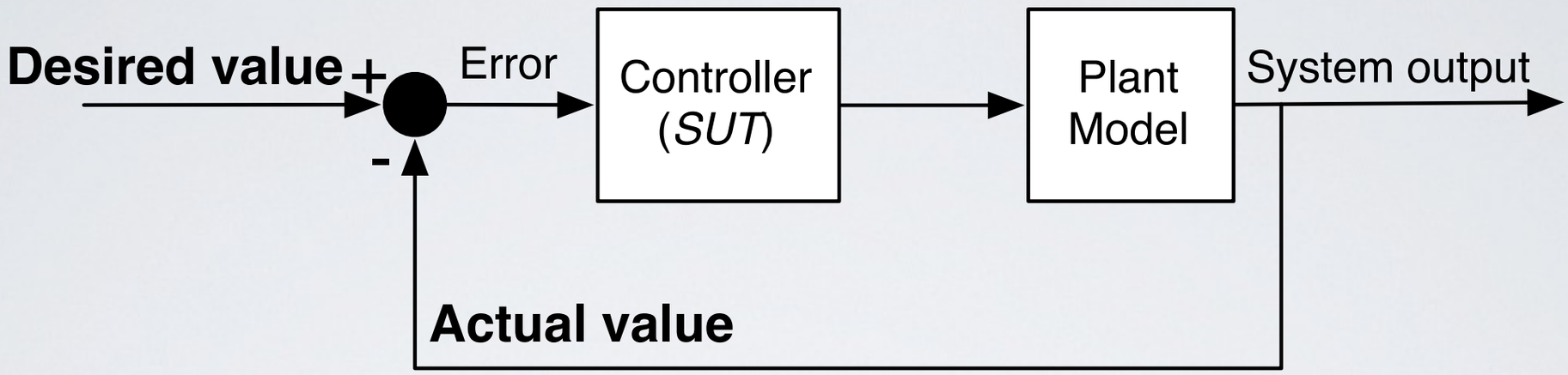


Flap position = 0 (open)

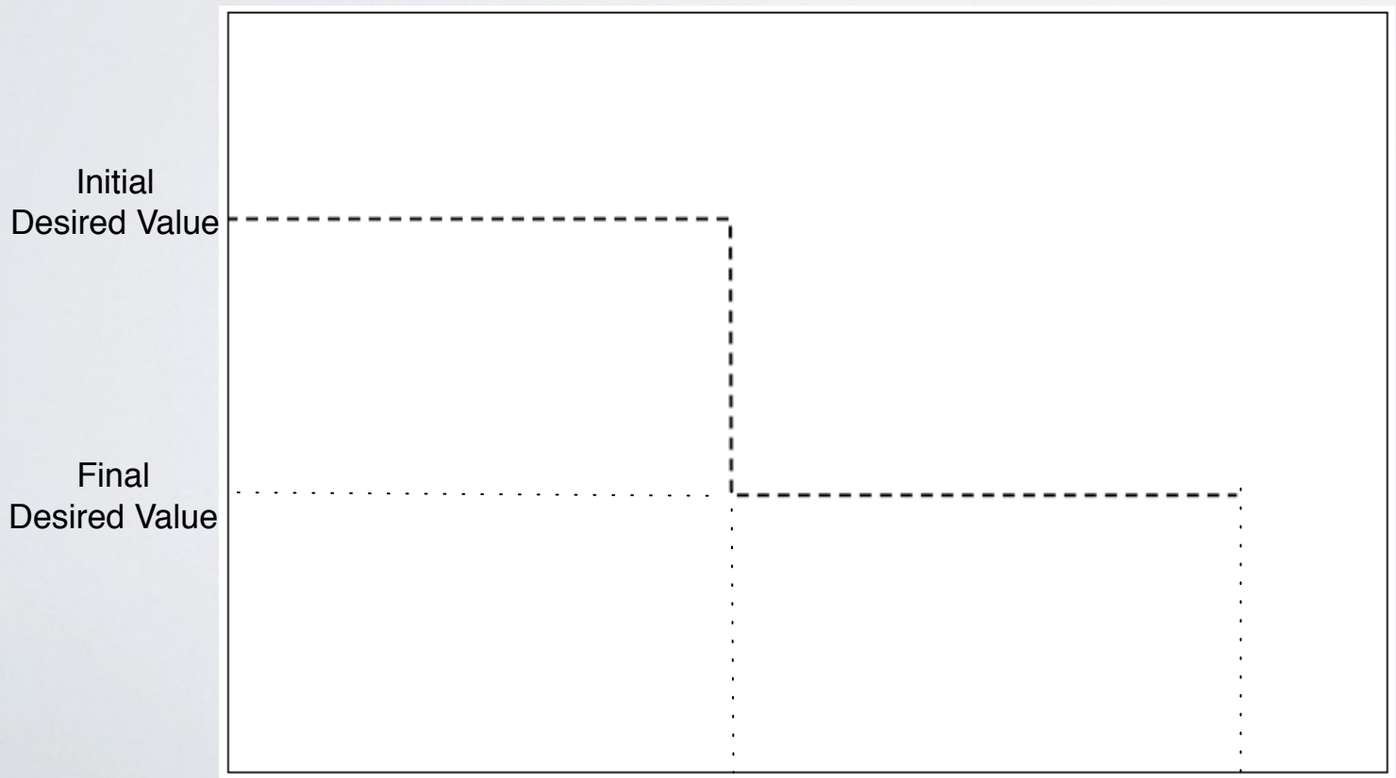


Flap position = 1 (closed)

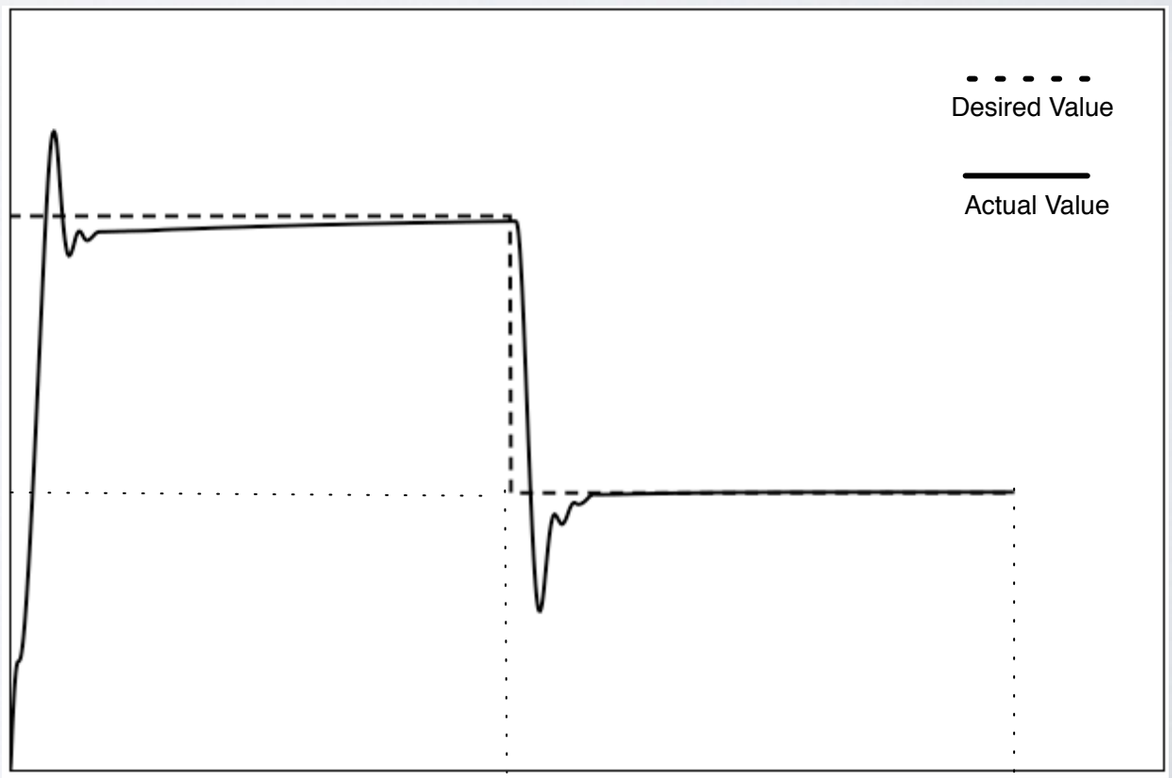
MiL Testing of Controllers



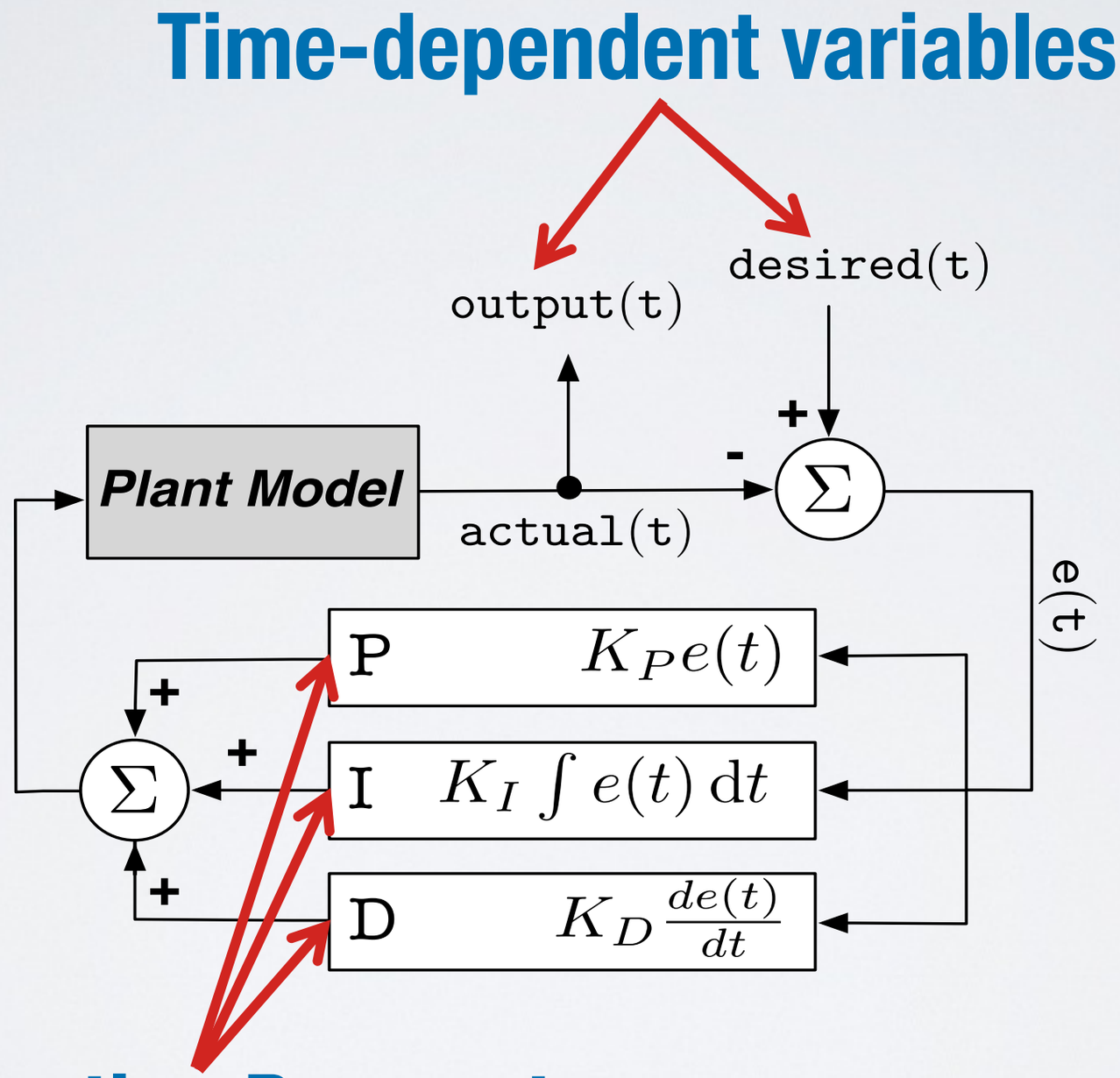
Test Input



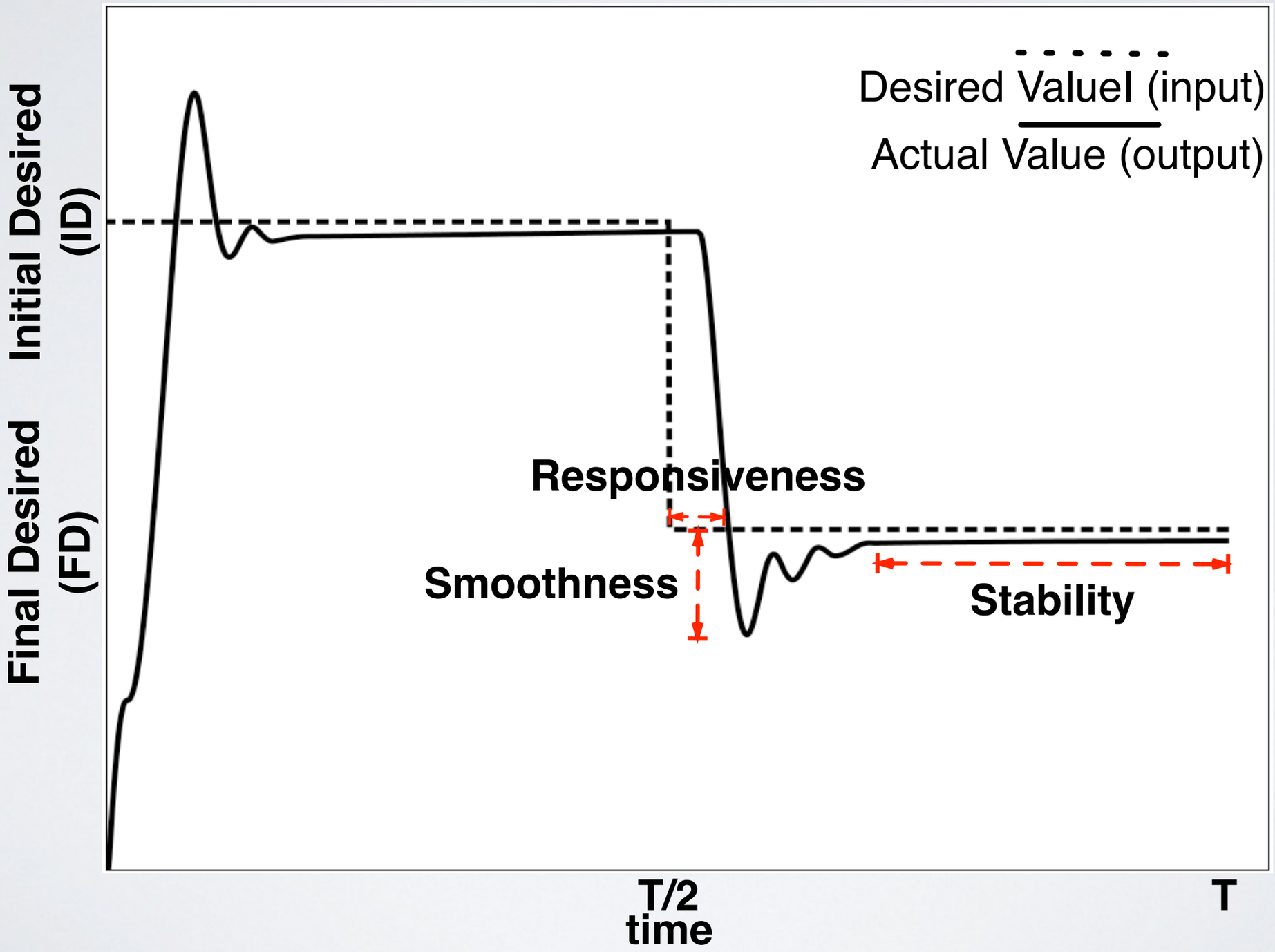
Test Output



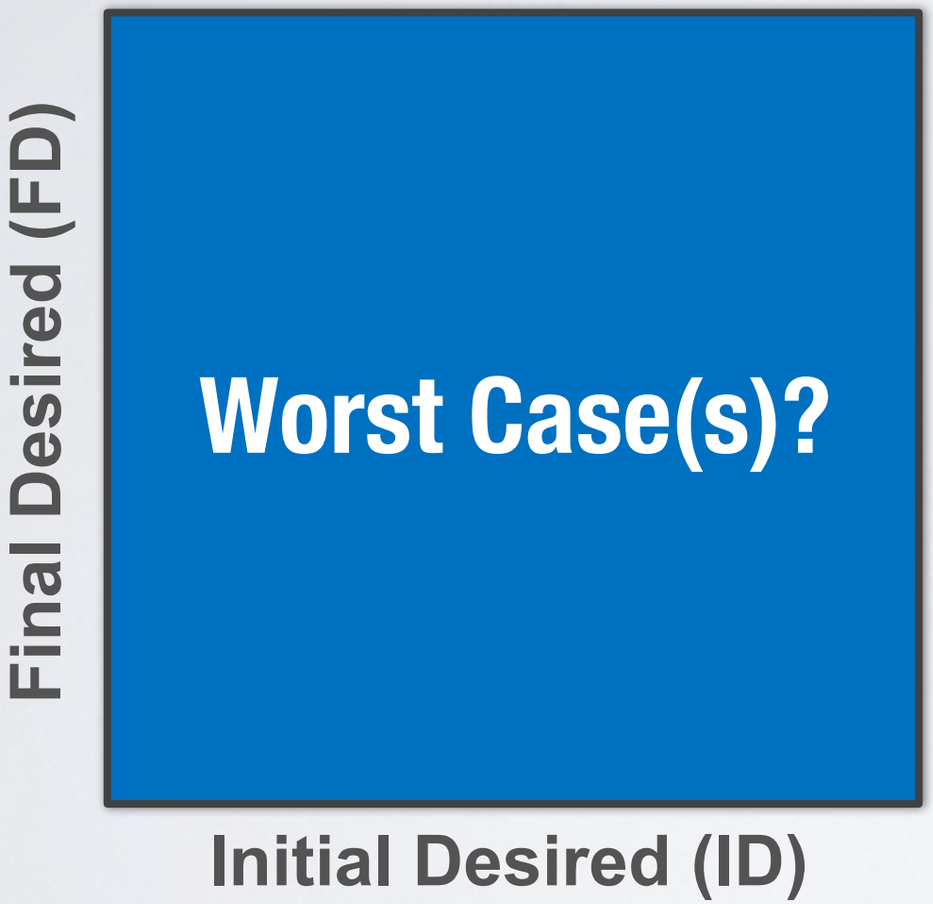
Configurable Controllers at MIL



Requirements and Test Objectives

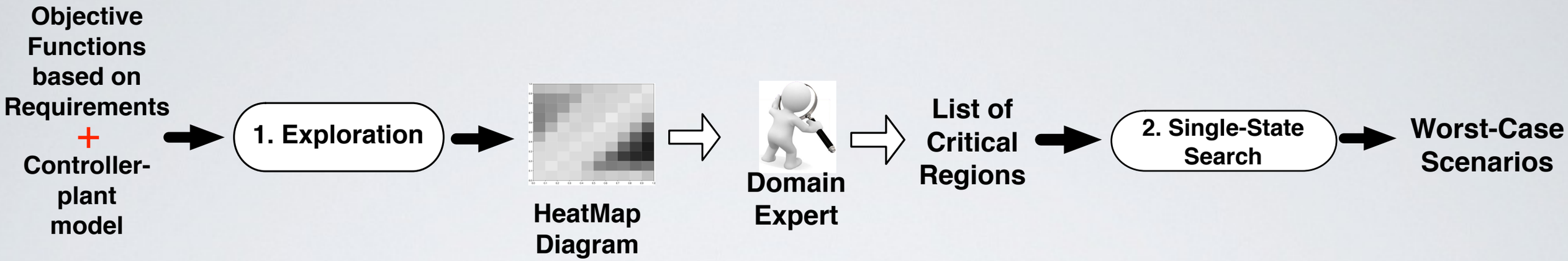


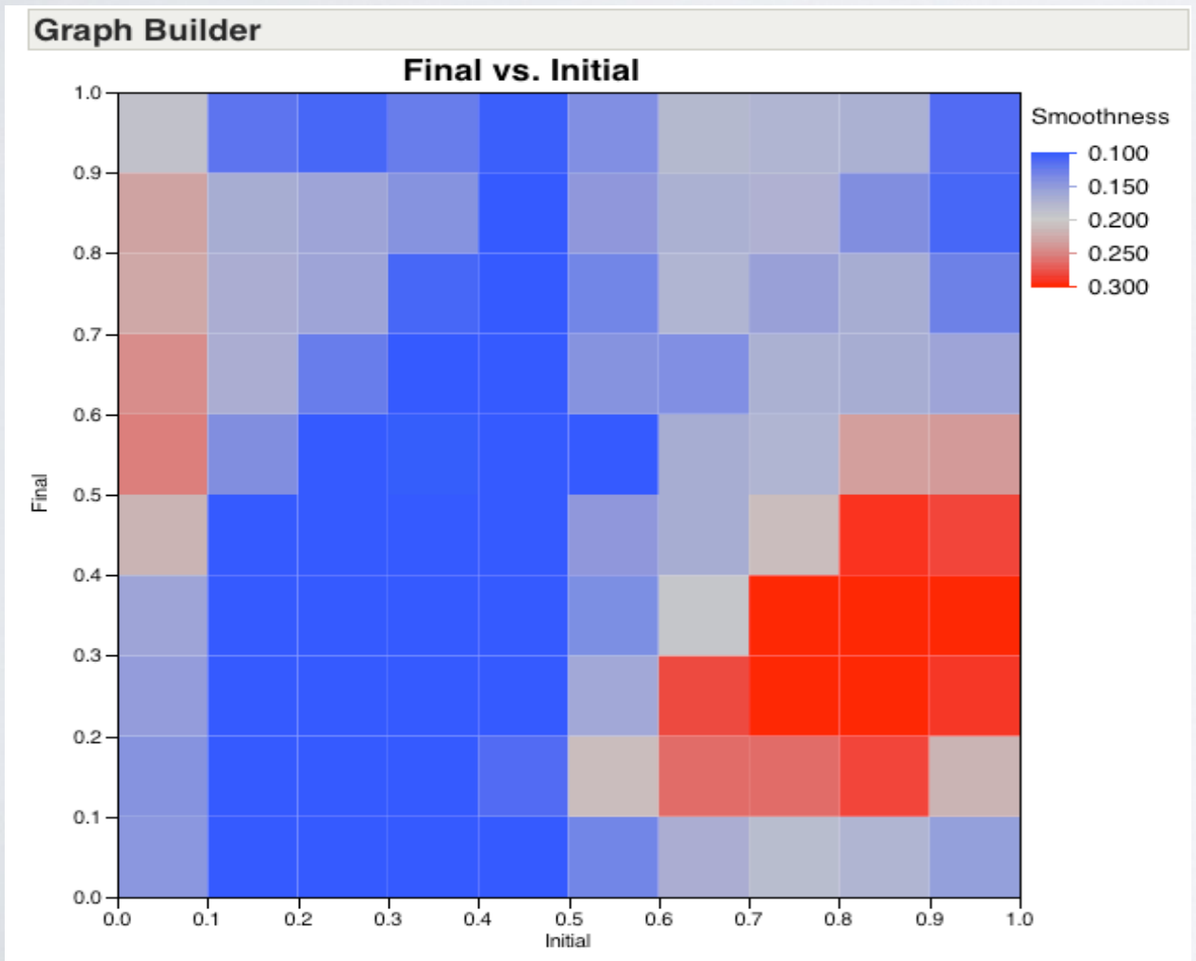
A Search-Based Test Approach

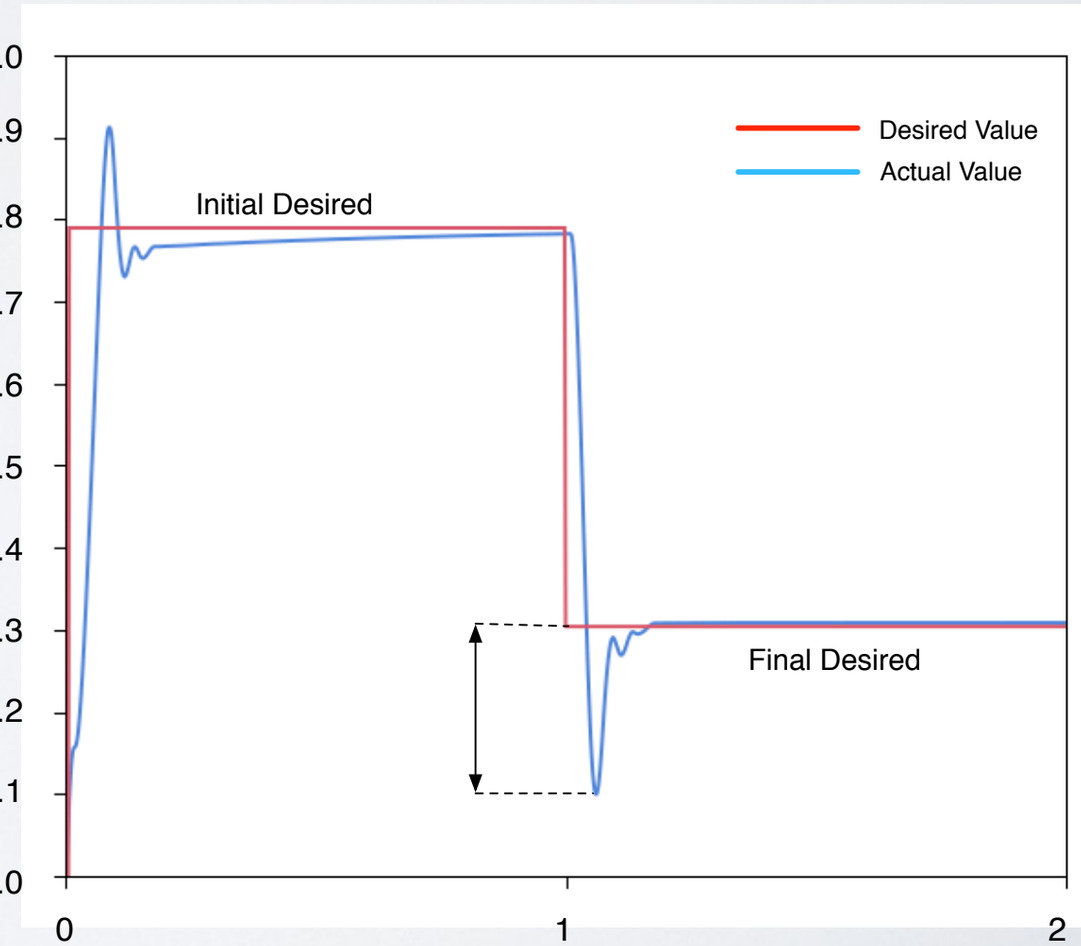


- Search directed by model execution feedback
- Controller's dynamic behavior can be complex
- Meta-heuristic search in (large) input space: Finding worst case inputs
- Possible because of automated oracle (feedback loop)
- Different worst cases for different requirements

Initial Solution







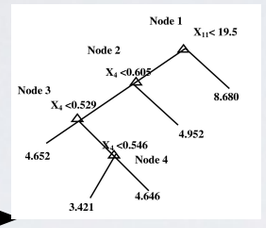
Results

- We found much worse scenarios during MiL testing than our partner had found so far
- These scenarios are also run at the HiL level, where testing is much more expensive: MiL results => test selection for HiL
- But further research was needed:
  - Simulations are expensive
  - Configuration parameters

Final Solution

Objective Functions + Controller Model (Simulink)

1.Exploration with Dimensionality Reduction



Regression Tree

Domain Expert

List of Critical Partitions

2.Search with Surrogate Modeling

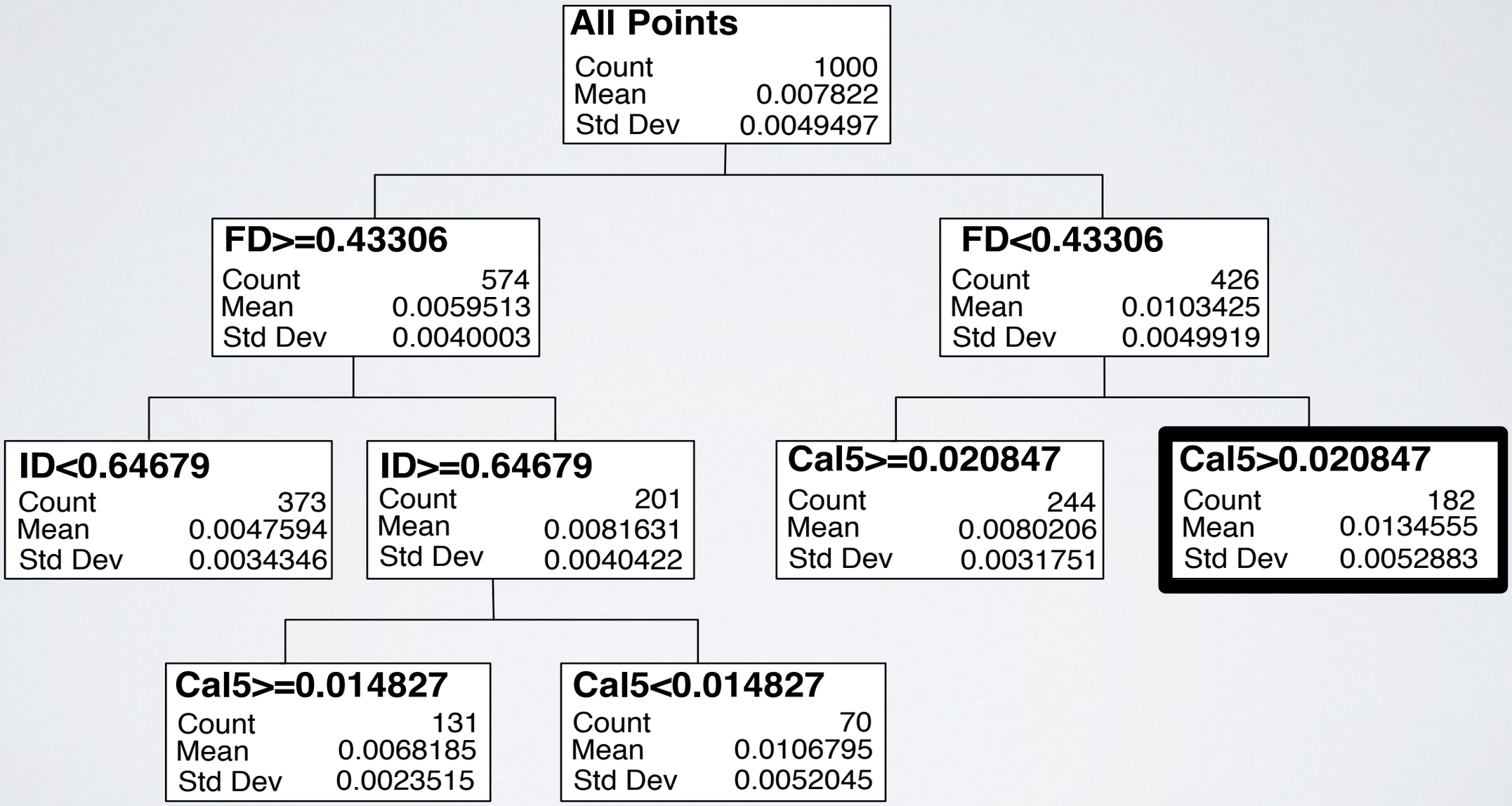
Worst-Case Scenarios

Dimensionality reduction to identify the significant variables (Elementary Effect Analysis)

Visualization of the 8-dimension space using regression trees

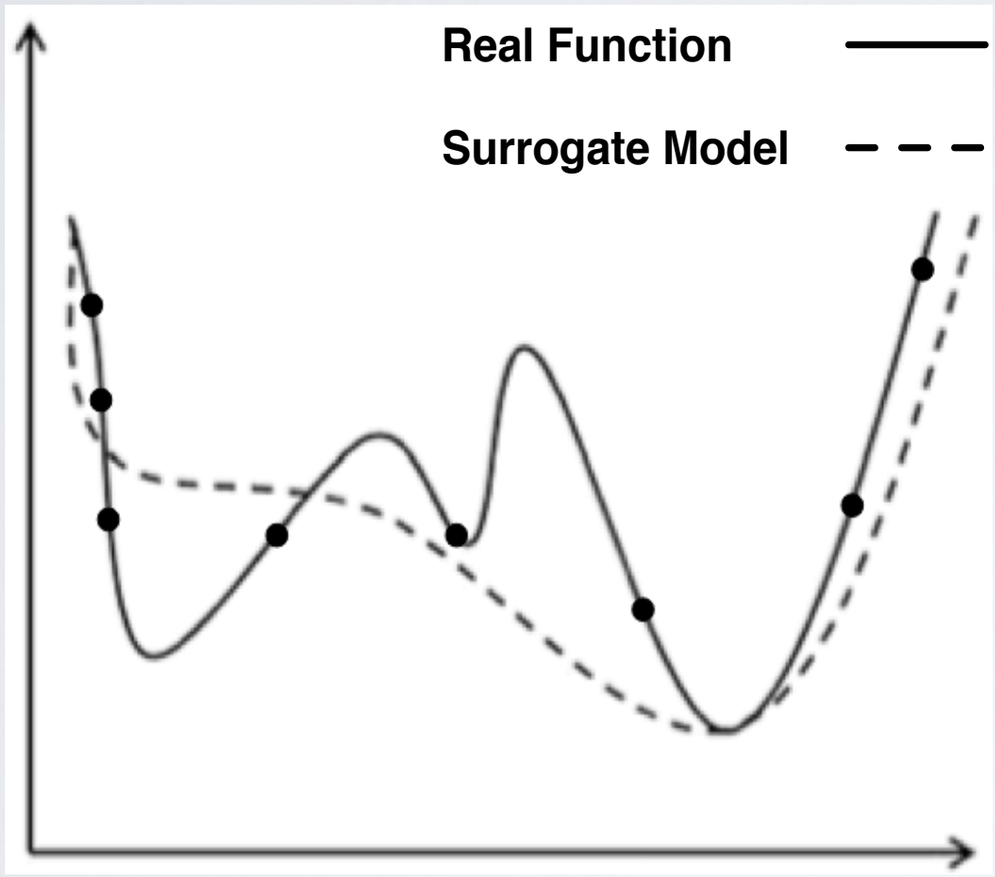
Surrogate modeling to predict the fitness function and speed up the search (Neural network)

Regression Tree



Surrogate Modeling

Fitness



x

Any supervised learning or statistical technique providing fitness predictions with confidence intervals

1. **Predict higher fitness with high confidence: Move to new position, no simulation**
2. **Predict lower fitness with high confidence: Do not move to new position, no simulation**
3. **Low confidence in prediction: Simulation**

Results

- ✓ **Our approach is able to identify more critical violations of the controller requirements that had neither been found with default/fixed configurations nor by manual testing.**

	MiL-Testing different configurations	MiL-Testing fixed configurations	Manual MiL-Testing
Stability	2.2% deviation	-	-
Smoothness	24% over/undershoot	20% over/undershoot	5% over/undershoot
Responsiveness	170 ms response time	80 ms response time	50 ms response time

Schedulability Analysis and Testing

Problem and Context

- **Schedulability analysis** encompasses techniques that try to predict whether (critical) tasks are schedulable, i.e., meet their deadlines

- **Stress testing** runs carefully selected test cases that have a high probability of leading to deadline misses

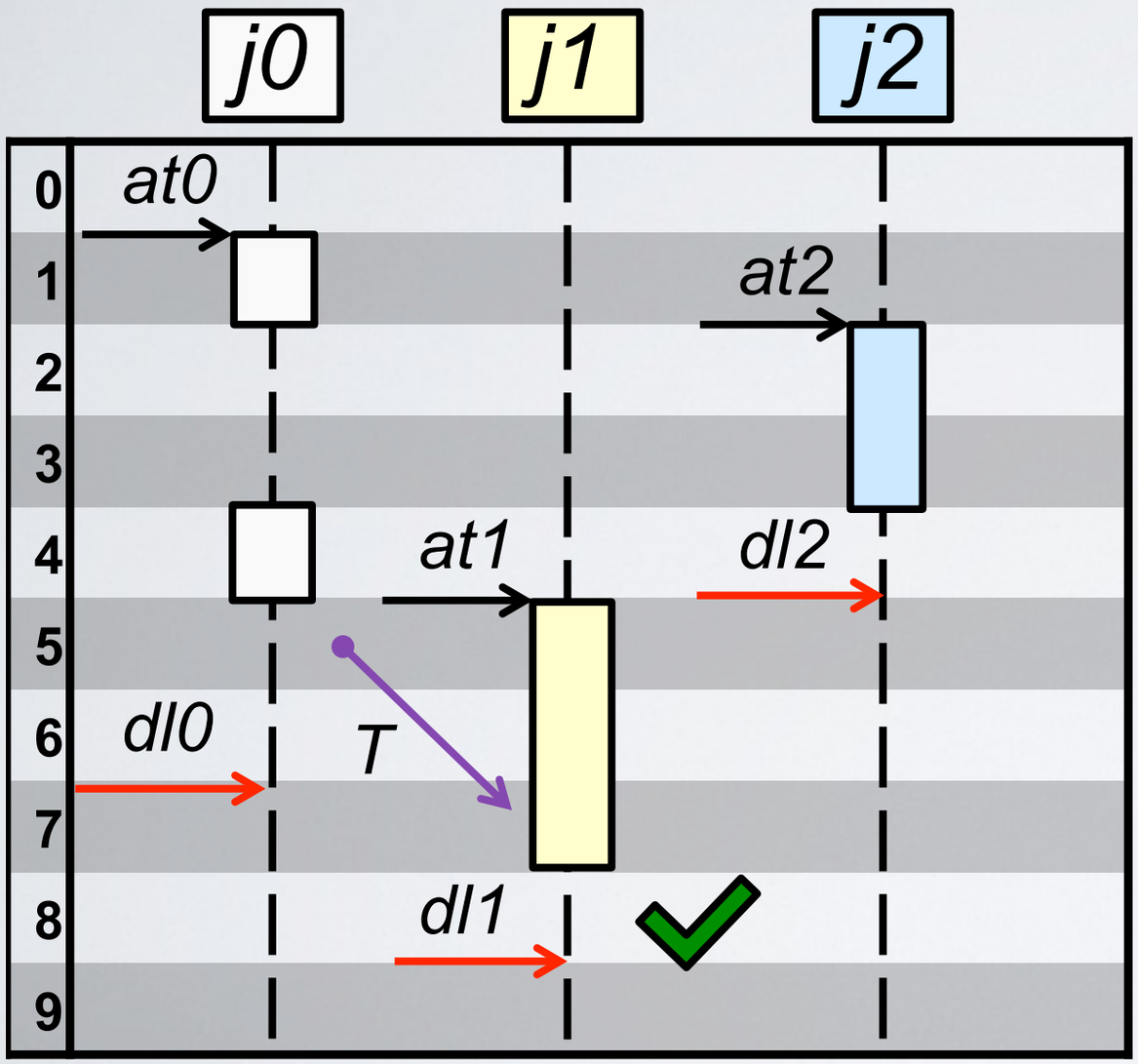
- **Stress testing** is **complementary** to schedulability analysis

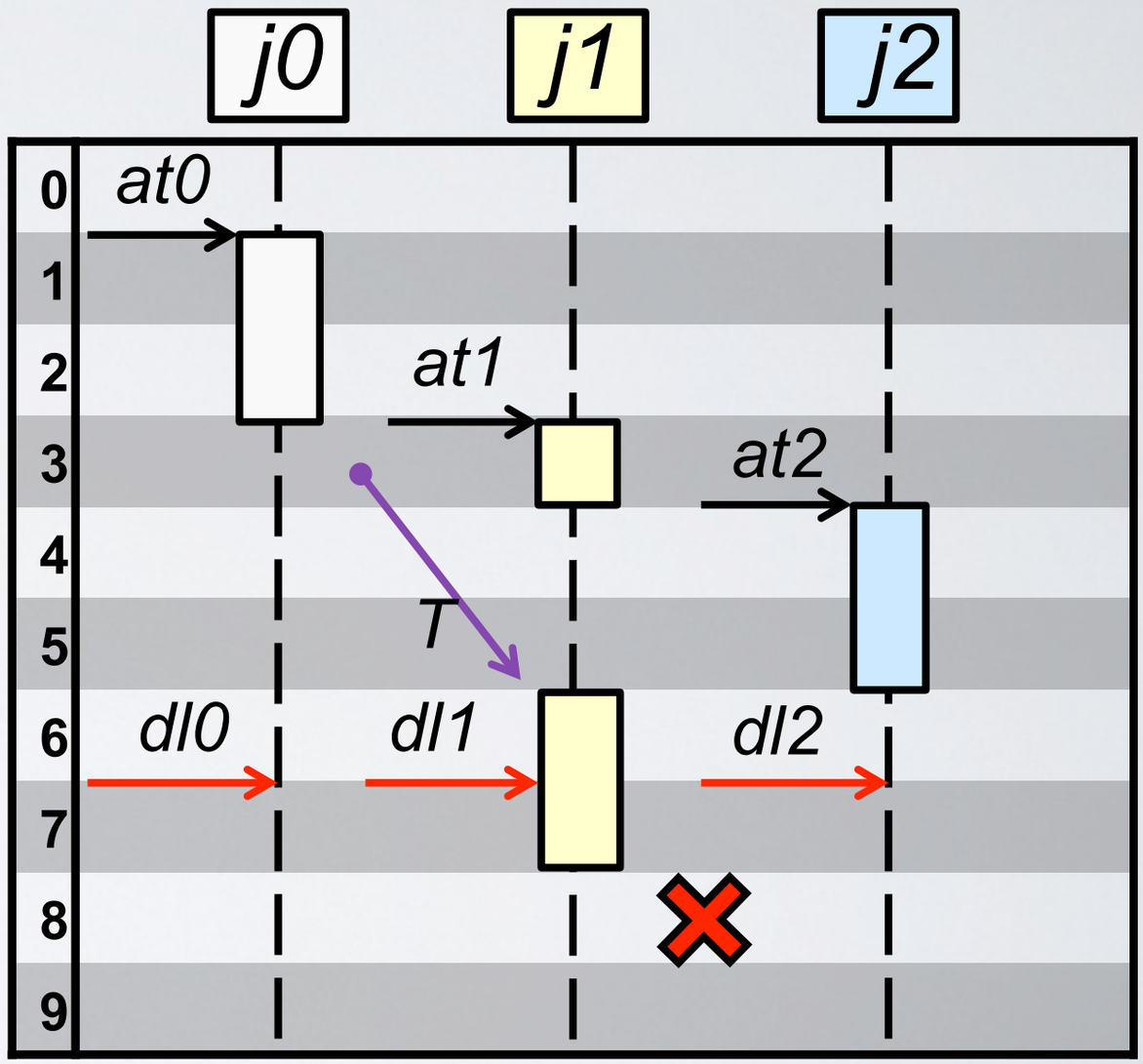
- Testing is typically expensive, e.g., hardware in the loop

- **Finding stress test cases is difficult**

Finding Stress Test Cases is Hard

j0, j1 , j2 arrive at at0 , at1 , at2 and must finish before dl0 , dl1 , dl2





J1 can miss its deadline dl1 depending on when at2 occurs!

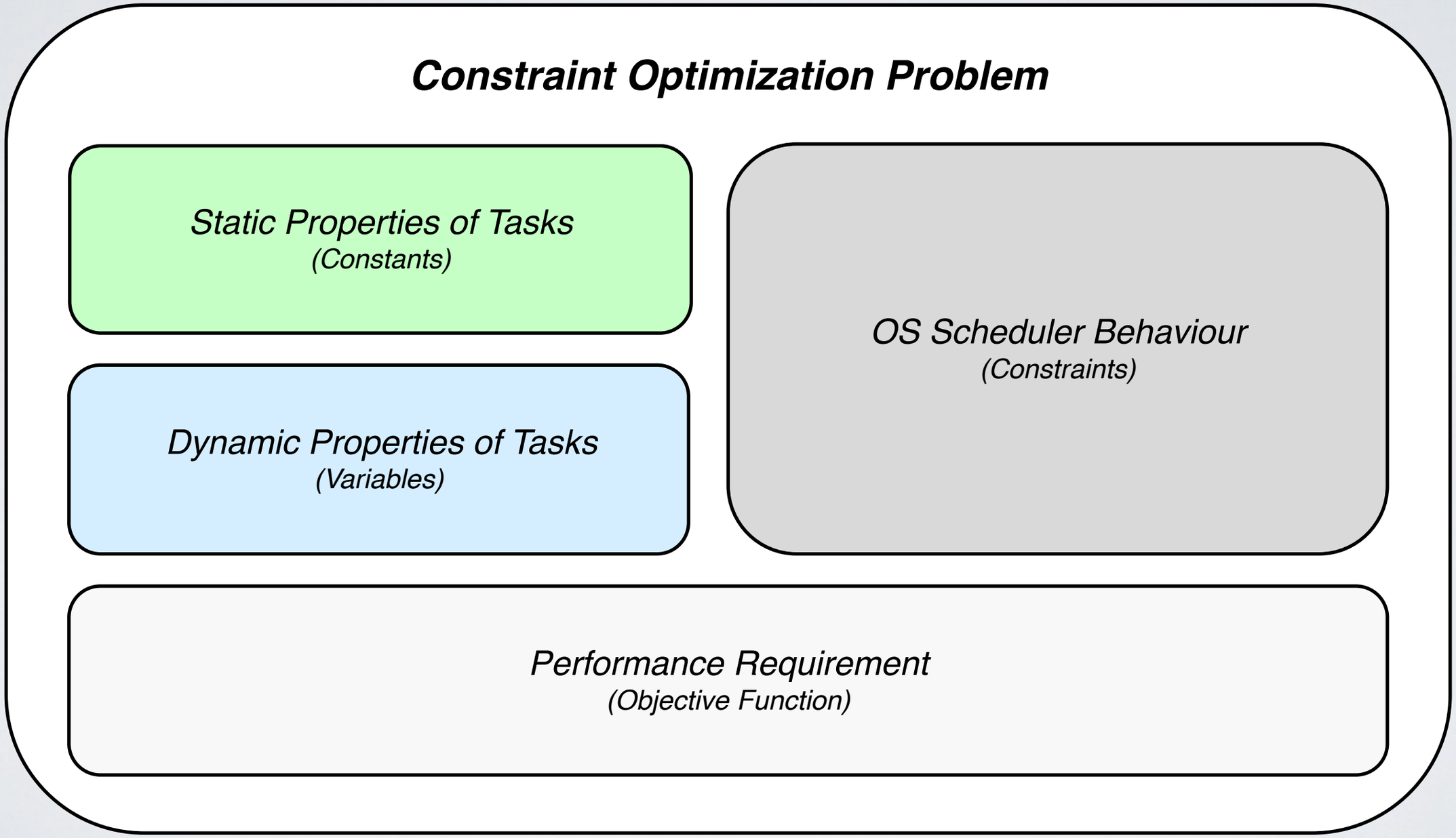
Challenges and Solutions

- Ranges for arrival times form a very large input space

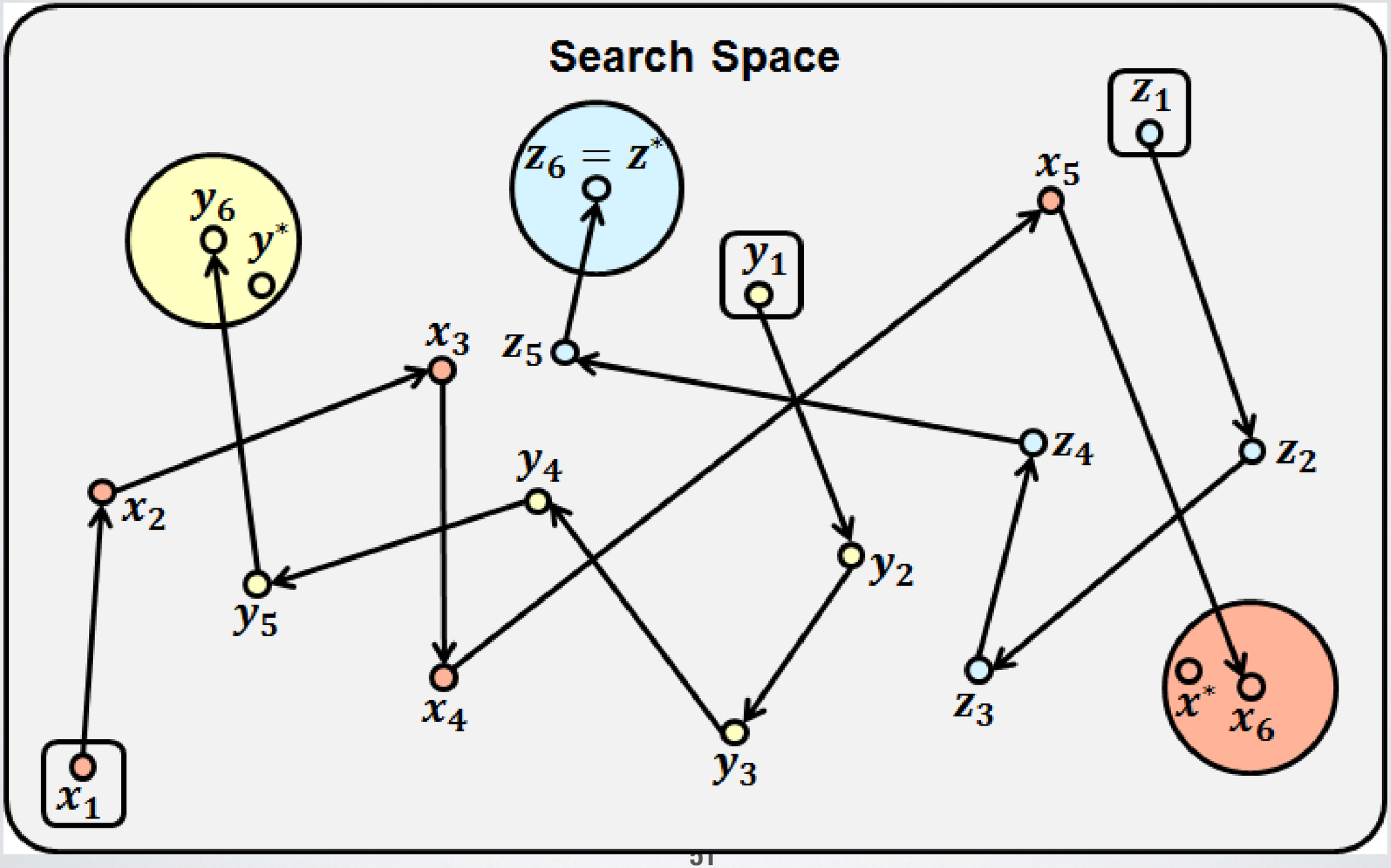
- Task interdependencies and properties constrain what parts of the space are feasible

- **Solution:** We re-expressed the problem as a constraint optimization problem and used **a combination of constraint programming (IBM CPLEX) and meta-heuristic search (GA)**

Constraint Optimization



Combining CP and GA



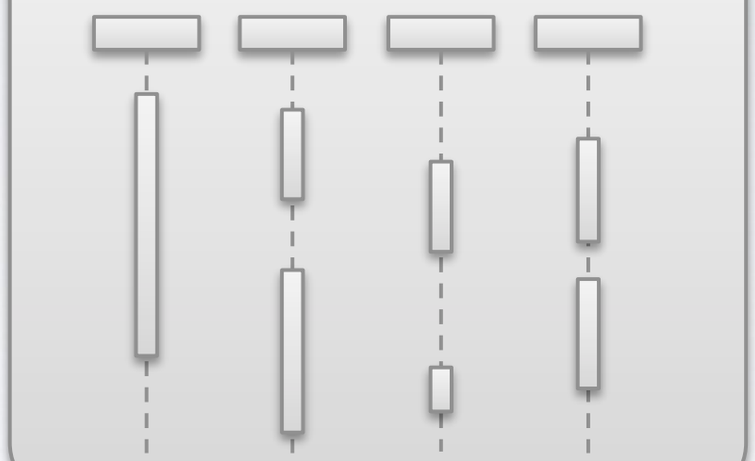
Case Study

System monitors gas leaks and fire in oil extraction platforms

KONGSBERG

Control Modules

Drivers
(Software-Hardware Interface)



Alarm Devices
(Hardware)

Real-Time Operating System

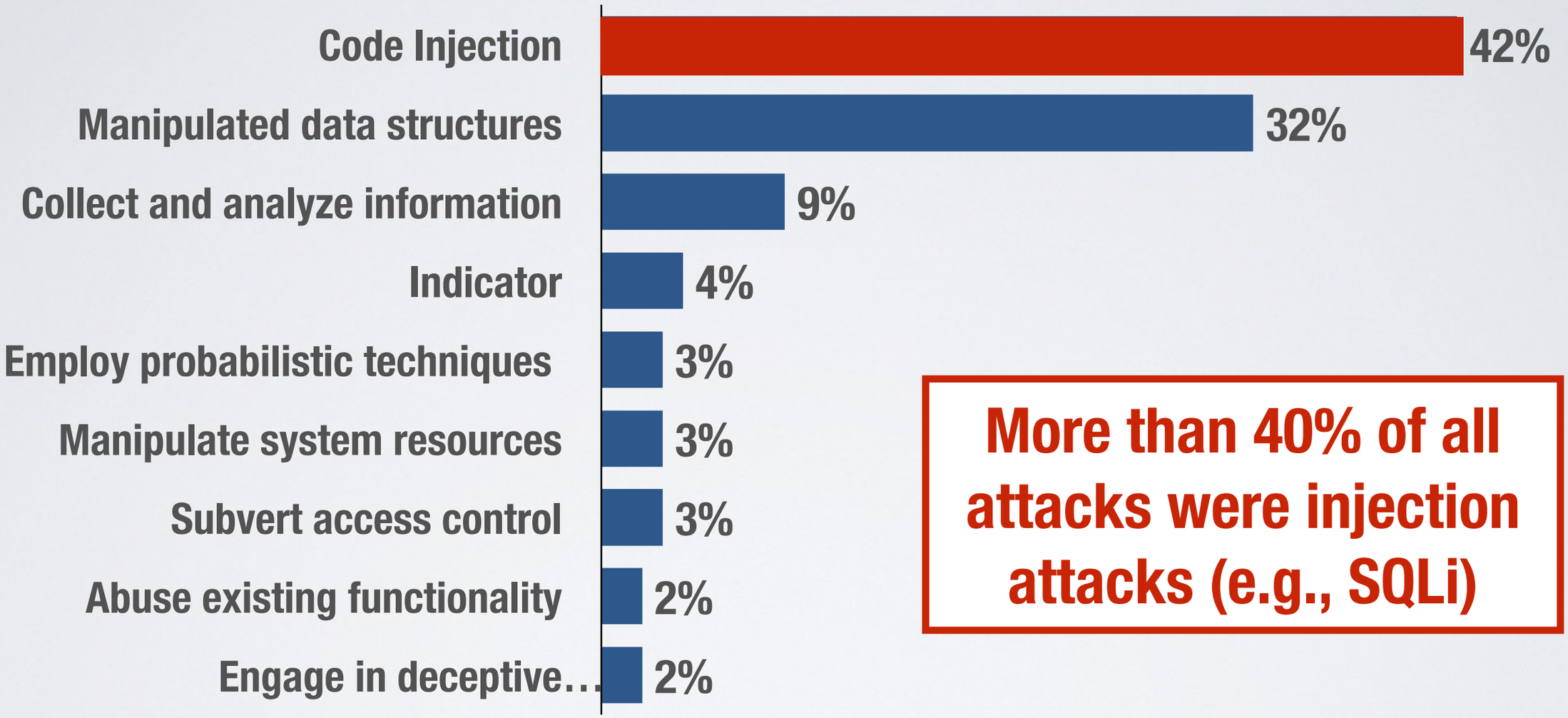
Multicore Architecture

Summary

- We provided a solution for generating stress test cases by combining meta-heuristic search and constraint programming
  - Meta-heuristic search (GA) identifies high risk regions in the input space
  - Constraint programming (CP) finds provably worst-case schedules within these (limited) regions
  - Achieve (nearly) GA efficiency and CP effectiveness
- Our approach can be used both for stress testing and schedulability analysis (assumption free)

Vulnerability Testing

X-Force Threat Intelligence Index 2017



More than 40% of all attacks were injection attacks (e.g., SQLi)

IBM

https://www.ibm.com/security/xforce/

Web Applications

Web Applications

Client

Server

SQL Database

Username str1

Password str2

OK

Web form

```
SELECT *
FROM Users WHERE
(usr = 'str1' AND psw = 'str2')
```

SQL query

Name	Surname	…
John	Smith	…

Result

Injection Attacks

Client

Server

SQL Database

Username

Password ‘) OR 1=1 --

OK

Web form

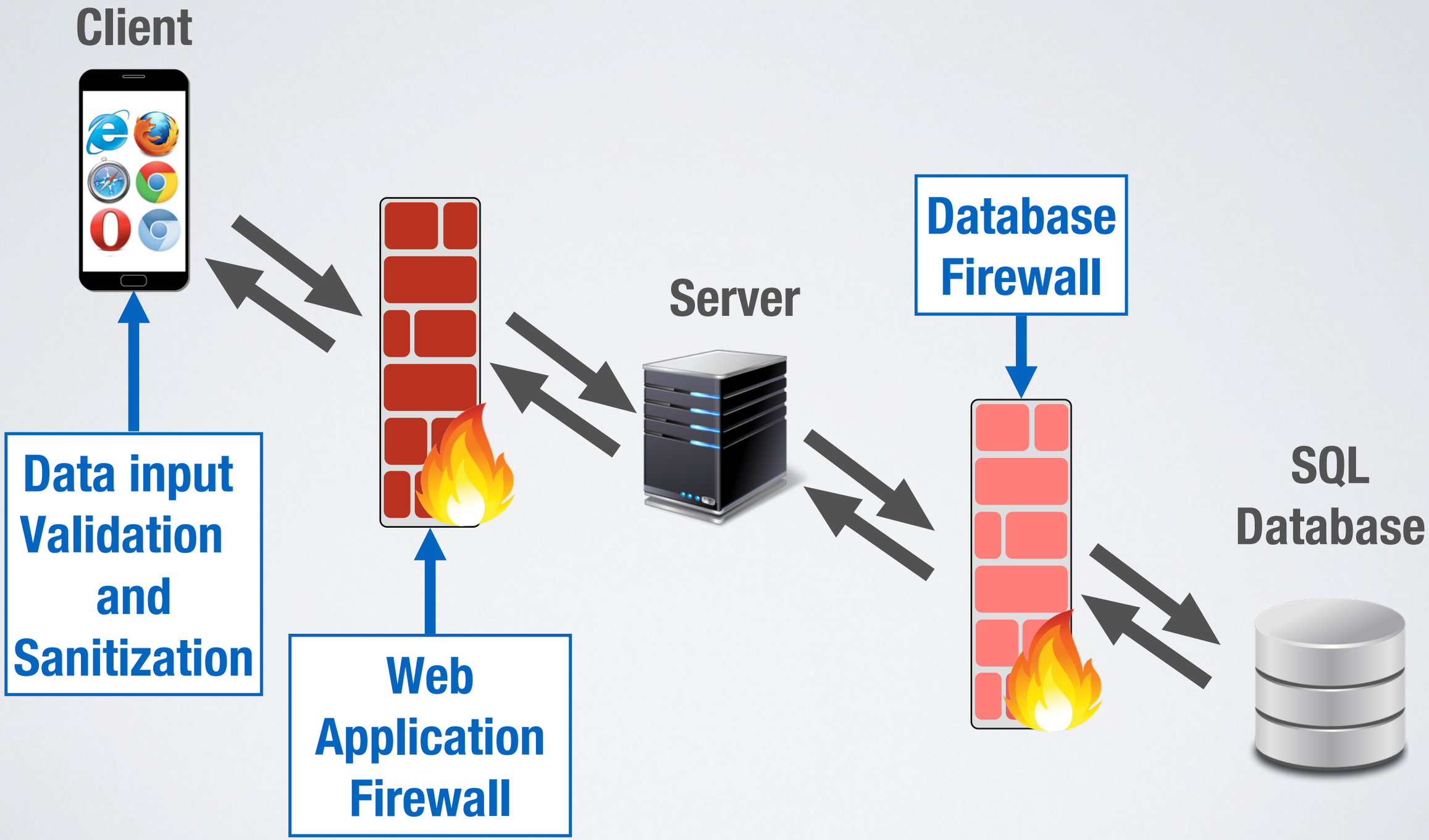
```
SELECT *
FROM Users
WHERE (usr = ‘’ AND
psw = ‘’) OR  1=1 --
```

SQL query

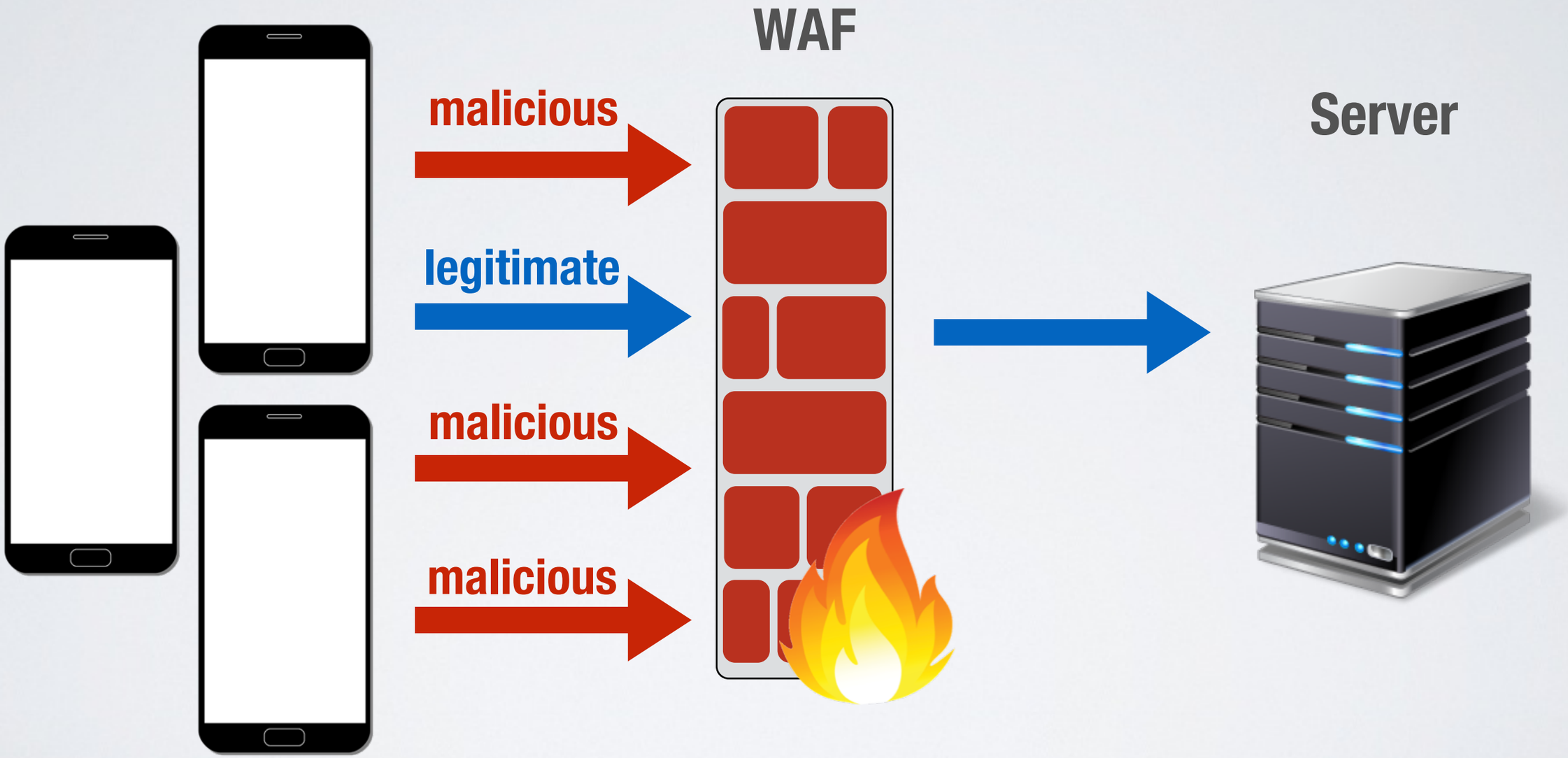
Name	Surname	…
Aria	Stark	…
John	Snow	…
…	…	…

Query result

Protection Layers



Web Application Firewalls (WAFs)



WAF Rule Set

Rule set of Apache ModSecurity

```
#
# SQL Keyword Anomaly Scoring
#
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "@pm select show top
distinct from dual where group by order having limit offset union rownum as (case"
"phase:2,id:'981300',t:none,t:urlDecodeUni,t:lowercase,nolog,pass,nolog,setvar:'tx.sqli_select_statement=%
{tx.sqli_select_statement} %{matched_var}'"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord select"
"phase:2,id:'981301',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord show"
"phase:2,id:'981302',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord top"
"phase:2,id:'981303',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord distinct"
"phase:2,id:'981304',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord from"
"phase:2,id:'981305',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord dual"
"phase:2,id:'981306',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord where"
"phase:2,id:'981307',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@contains group by"
"phase:2,id:'981308',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@contains order by"
"phase:2,id:'981309',t:none,pass,nolog,setvar:tx.sqli_select_statement_count=+1,setvar:tx.sql_injection_score=+1"
SecRule TX:SQLI_SELECT_STATEMENT "@containsWord having"
```

https://github.com/SpiderLabs/ModSecurity

Misconfigured WAFs

False Positive

False Negative

Grammar-based Attack Generation

- BNF grammar for SQLi attacks
- Random strategy: randomly selected production rules are applied recursively until only terminals are left
- Random strategy not efficient for bypassing attacks that are difficult to find
- Machine learning? Search?
- How to guide the search? How can ML help?

Anatomy of SQLi attacks

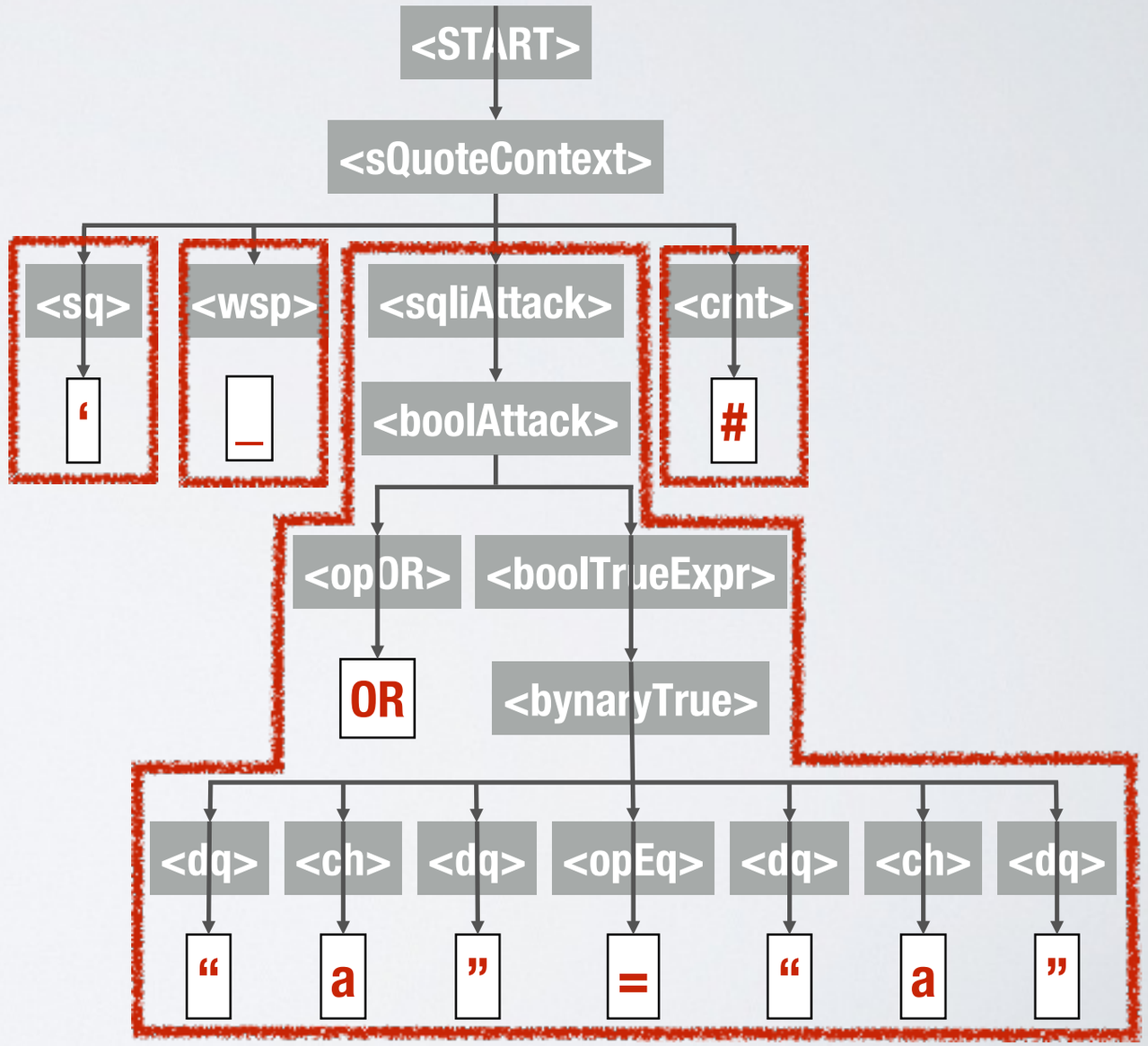
Bypassing Attack

‘ OR“a”=“a”#

Attack Slices

$$S = \begin{cases} ‘ \\ \_ \\ OR“a”=“a” \\ # \end{cases}$$

Derivation Tree



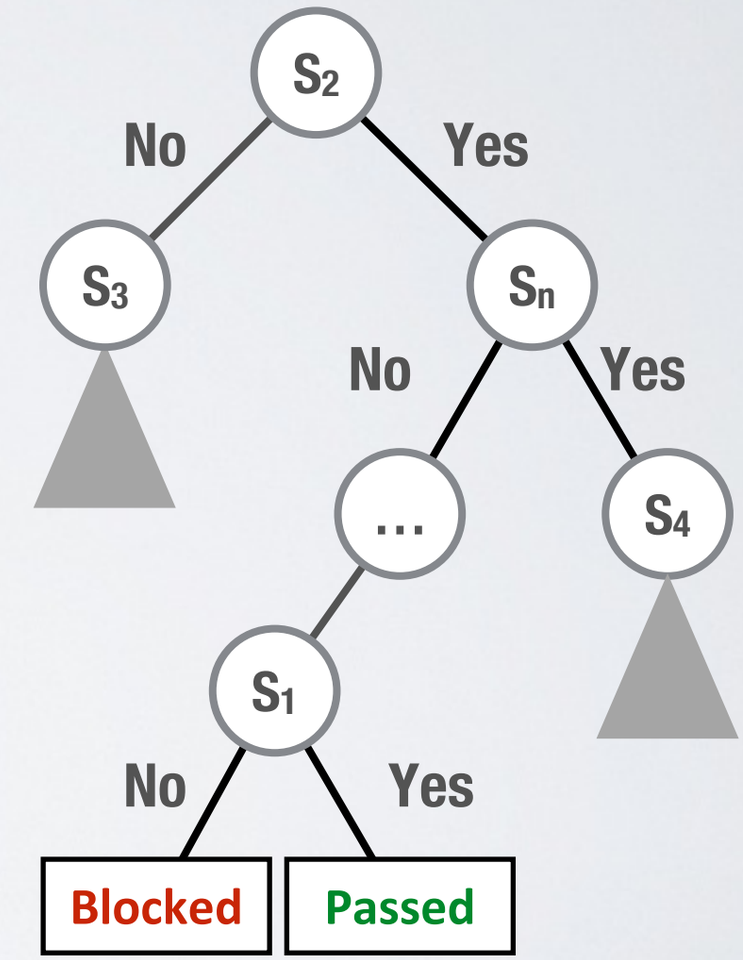
Learning Attack Patterns

Training Set

	S_1	S_2	S_3	S_4	…	S_n	Outcome
A_1	1	1	0	0	…	0	Passed
A_2	0	1	0	0	…	0	Blocked
…	…	…	…	…	…	…	…
A_m	1	1	1	1	…	1	Blocked

- Random trees
- Random forest

Decision Tree

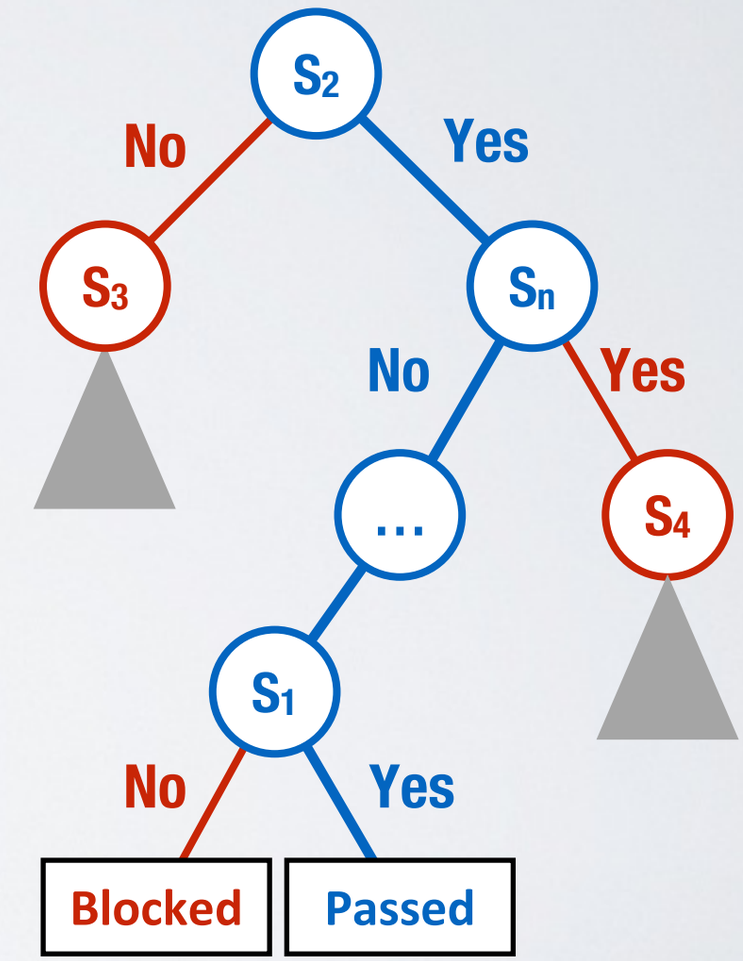


Learning Attack Patterns

Training Set

	S_1	S_2	S_3	S_4	…	S_n	Outcome
A_1	1	1	0	0	…	0	Passed
A_2	0	1	0	0	…	0	Blocked
…	…	…	…	…	…	…	…
A_m	1	1	1	1	…	1	Blocked

Decision Tree

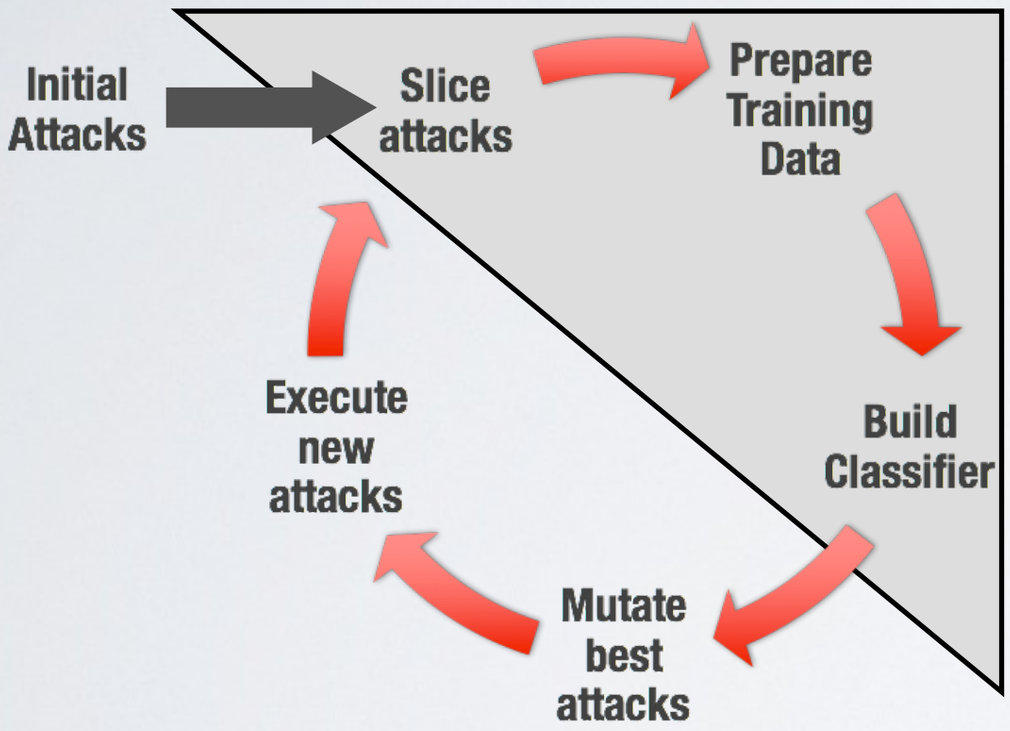


Attack Pattern

$$S_2 \wedge \neg S_n \wedge S_1$$

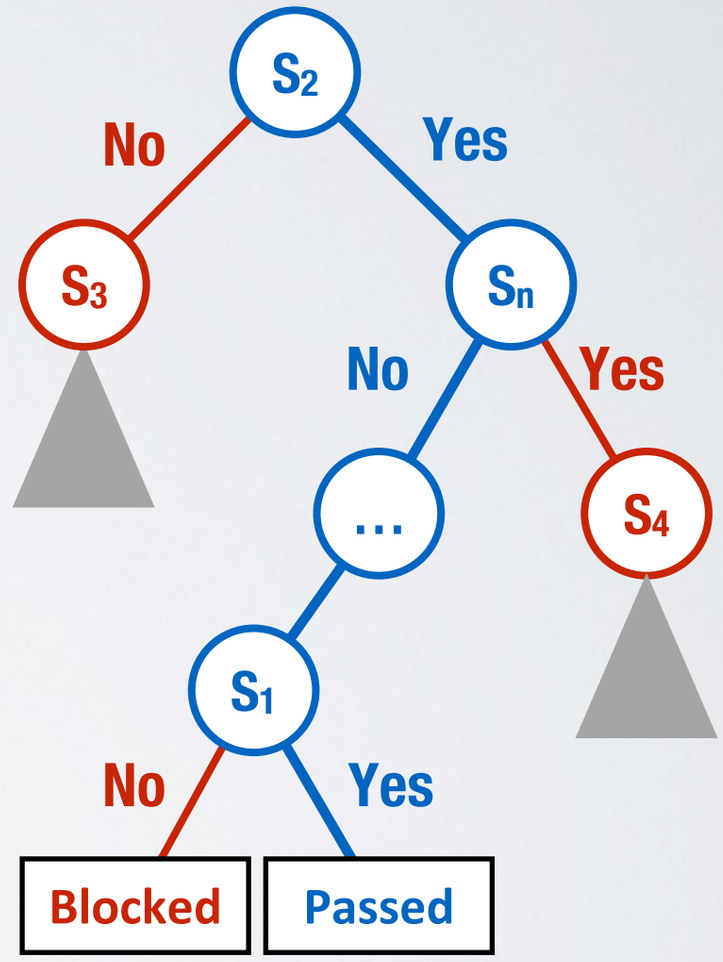
Generating Attacks via ML and EAs

Evolutionary Algorithm (EA)



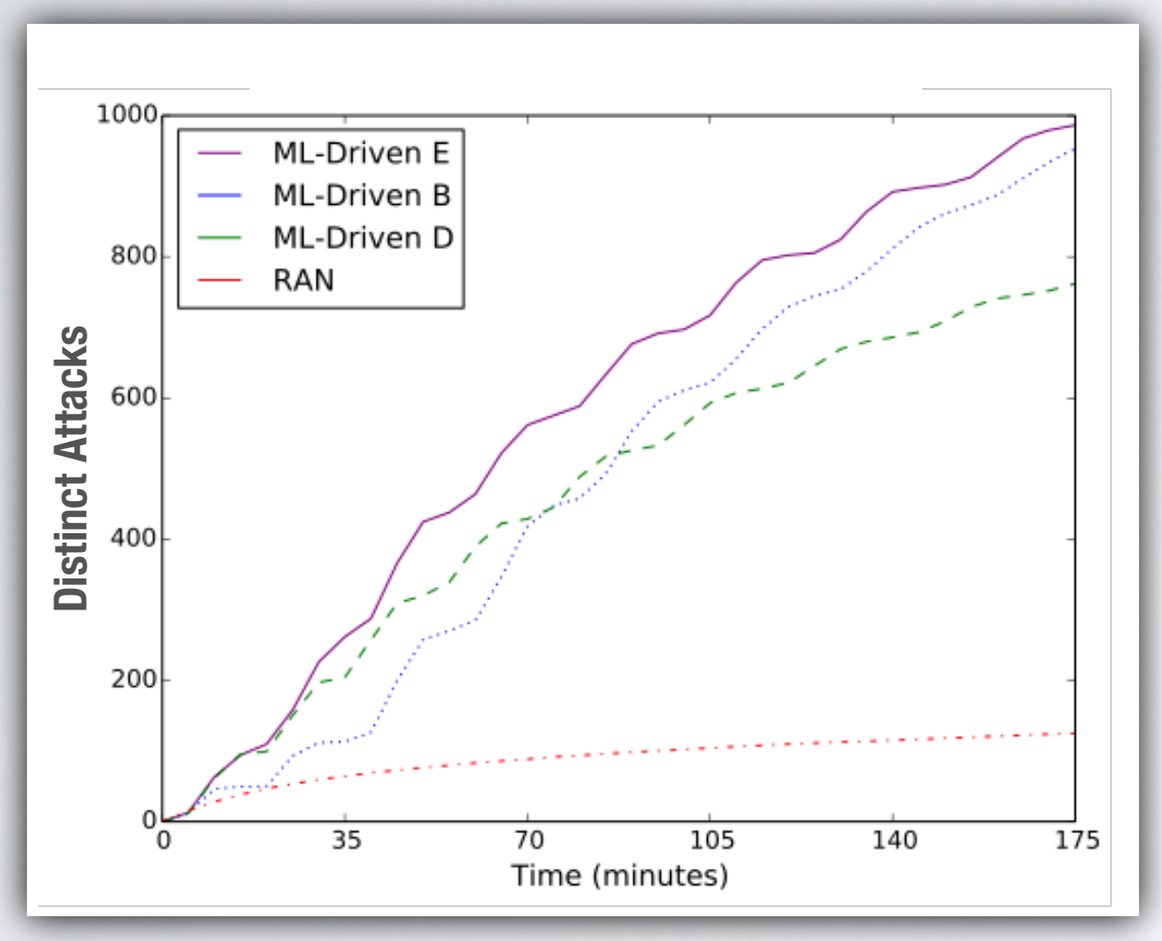
Iteratively refine successful attack conditions

Machine Learning

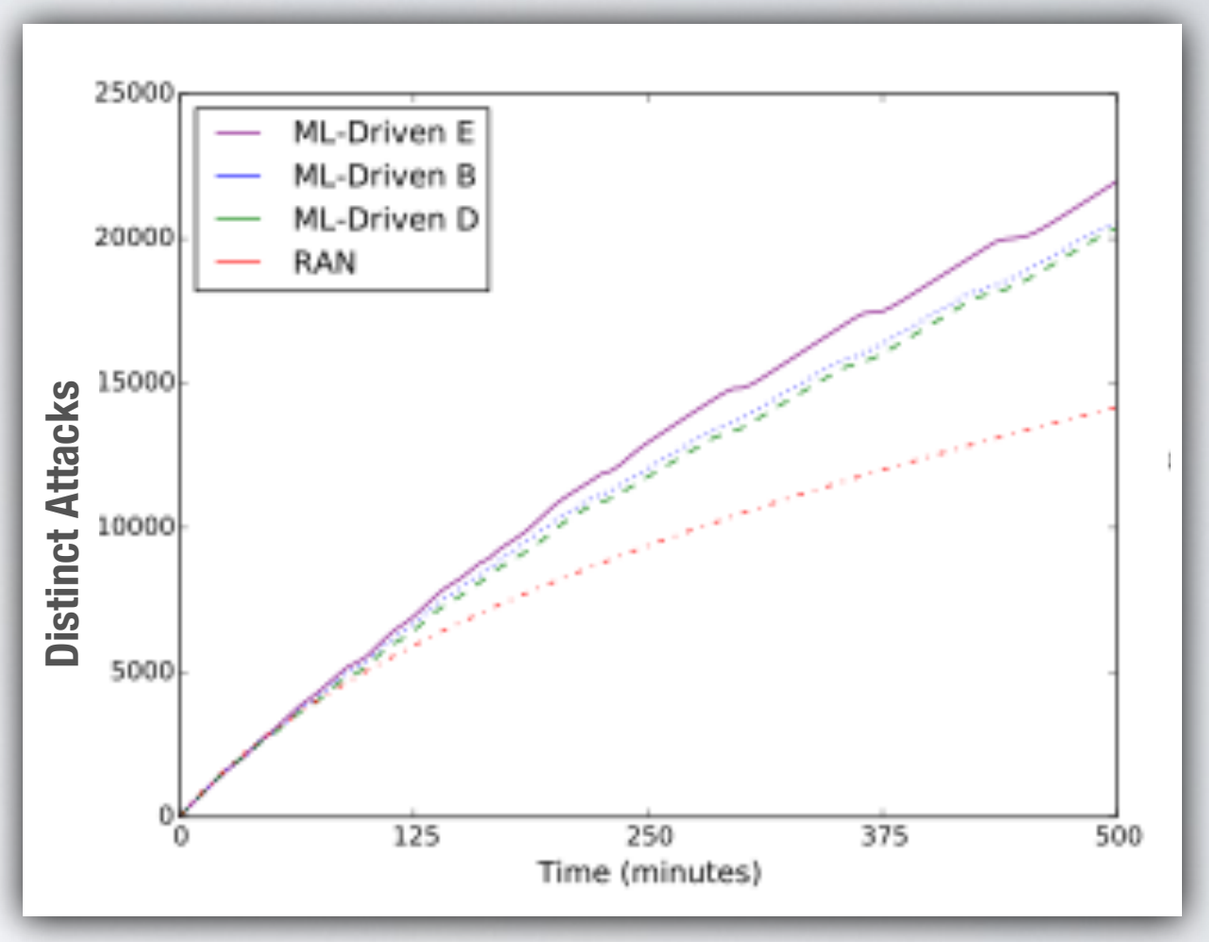


Some Results

Apache ModSecurity



Industrial WAFs



Machine Learning-driven attack generation led to more distinct, successful attacks being discovered faster

Related Work

- **Automated repair of WAFs**
- **Automated testing targeting XML and SQL injections in web applications**

Reflecting

Search-Based Solutions

- Versatile
- Helps relax assumptions compared to exact approaches
- Helps decrease modeling requirements
- Scalability, e.g., easy to parallelize
- Requires massive empirical studies
- Search is rarely sufficient by itself

Multidisciplinary Approach

- Single-technology approaches rarely work in practice
- Combined search with:
  - Machine learning
  - Solvers, e.g., CP, SMT
  - Statistical approaches, e.g., sensitivity analysis
  - System and environment modeling and simulation

Objectives

- Reduce search space
- Better guide and focus search
- Compute fitness and provide guidance
- Avoid expensive and useless fitness computations
- Explain failures (e.g., decision trees)
- Get more guarantees (e.g., constraint programming)

Acknowledgements

- Shiva Nejati
- Reza Matinnejad
- Raja Ben Abdessalem
- Stefano Di Alesio
- Dennis Appelt
- Annibale Panichella

Selected References

- L. Briand et al. “Testing the untestable: Model testing of complex software-intensive systems”, IEEE/ACM ICSE 2016, V2025

- R. Matinnejad et al., “MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models”, IEEE/ACM ASE 2014 (Distinguished paper award)

- S. Di Alesio et al. “Combining genetic algorithms and constraint programming to support stress testing of task deadlines”, ACM Transactions on Software Engineering and Methodology (TOSEM), 25(1):4, 2015

- R. Ben Abdessalem et al., "Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms”, IEEE/ACM ICSE 2018

- D. Appelt et al., “A Machine Learning-Driven Evolutionary Approach for Testing Web Application Firewalls”, To appear in IEEE Transaction on Reliability

- More on: https://wwwen.uni.lu/snt/people/lionel_briand?page=Publications

Achieving Scalability in Software Testing with Machine Learning and Metaheuristic Search

Lionel Briand

RAISE @ ICSE 2018