Testing Cyber Physical Systems via Evolutionary Algorithms and Machine Learning

Shiva Nejati
SnT, University of Luxembourg

SBST @ ICSE 2019
May 27, 2019

About SnT

securityandtrust.lu

- ICT research centre to fuel the national innovation system
- Part of the University of Luxembourg

40+ industry partners

20 MEUR turnover (70% external funding)

>100 M€

Acquired competitive funding since launch

60% of PhDs and RAs work on industry projects

>300 employees

51 nationalities

Software Verification and Validation Group (http://svv.lu)

- Established in 2012
- Requirements Engineering, Security Analysis, Design Verification, Automated Testing, Runtime Monitoring
- 5 faculty members (head: Lionel Briand)
- 11 research associates
- 13 PhD candidates
- 3 research fellows
- 10 current industry partnerships
- Budget 2018: ~2 M€

SVV Industry Partners

- SES and LuxSpace (Satellites)
- Delphi and IEE (Automotive)
- Government of Luxembourg
- HITEC (Emergency systems)
- BGL – BNP Paribas, Clearstream (Banking)
- Escent (MDE Coaching)
- QRA (Quality Assurance)

SES your satellite company

LUXSPACE

DELPHI

a sense for innovation IEE

LE GOUVERNEMENT DU GRAND-DUCHÉ DE LUXEMBOURG

HITEC LUXEMBOURG

BGL BNP PARIBAS

Fondation Alphonse Weicker

clearstream Deutsche Börse Group

'ESCENT

QRA critical thinking

SVV Industry Partners

SES and LuxSpace (Satellites)

Delphi and IEE (Automotive)

Government of Luxembourg

HITEC (Emergency systems)

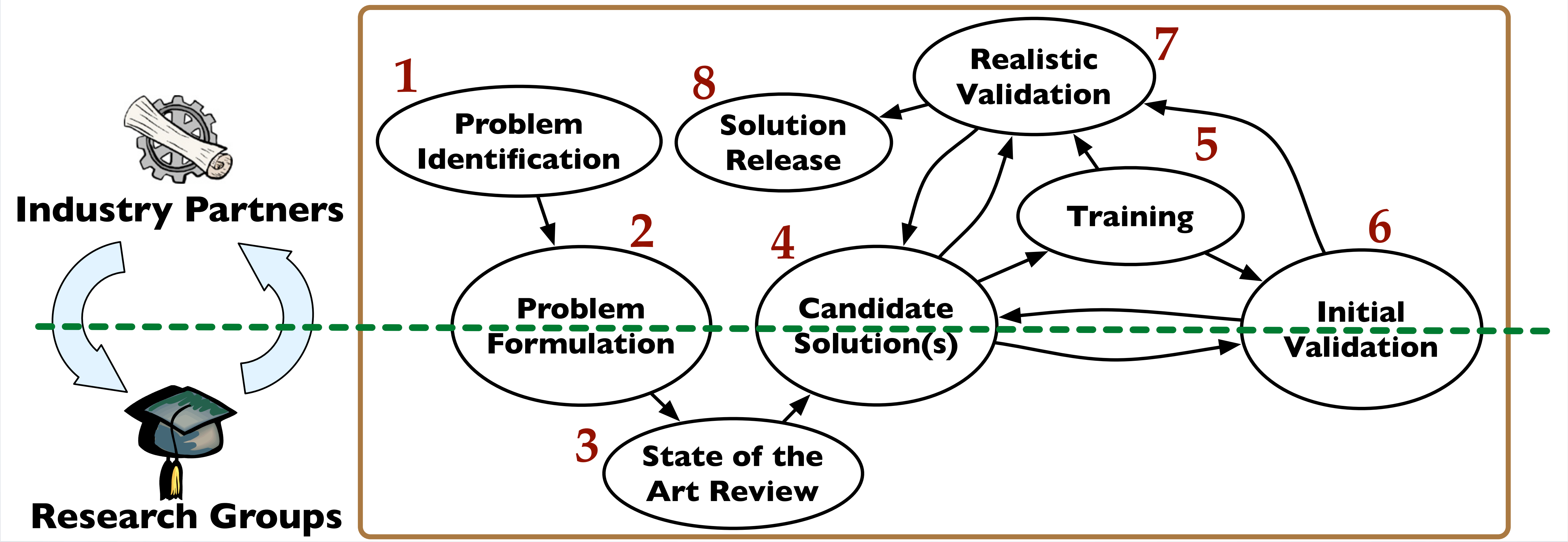
BGL – BNP Paribas, Clearstream (Banking)

Escent (MDE Coaching)

QRA (Quality Assurance)

Mode of Collaboration

- Research driven by industry needs
- Realistic evaluations
- Combining research with innovation and technology transfer



Adapted from [Gorschek et al. 2006]

Acknowledgements

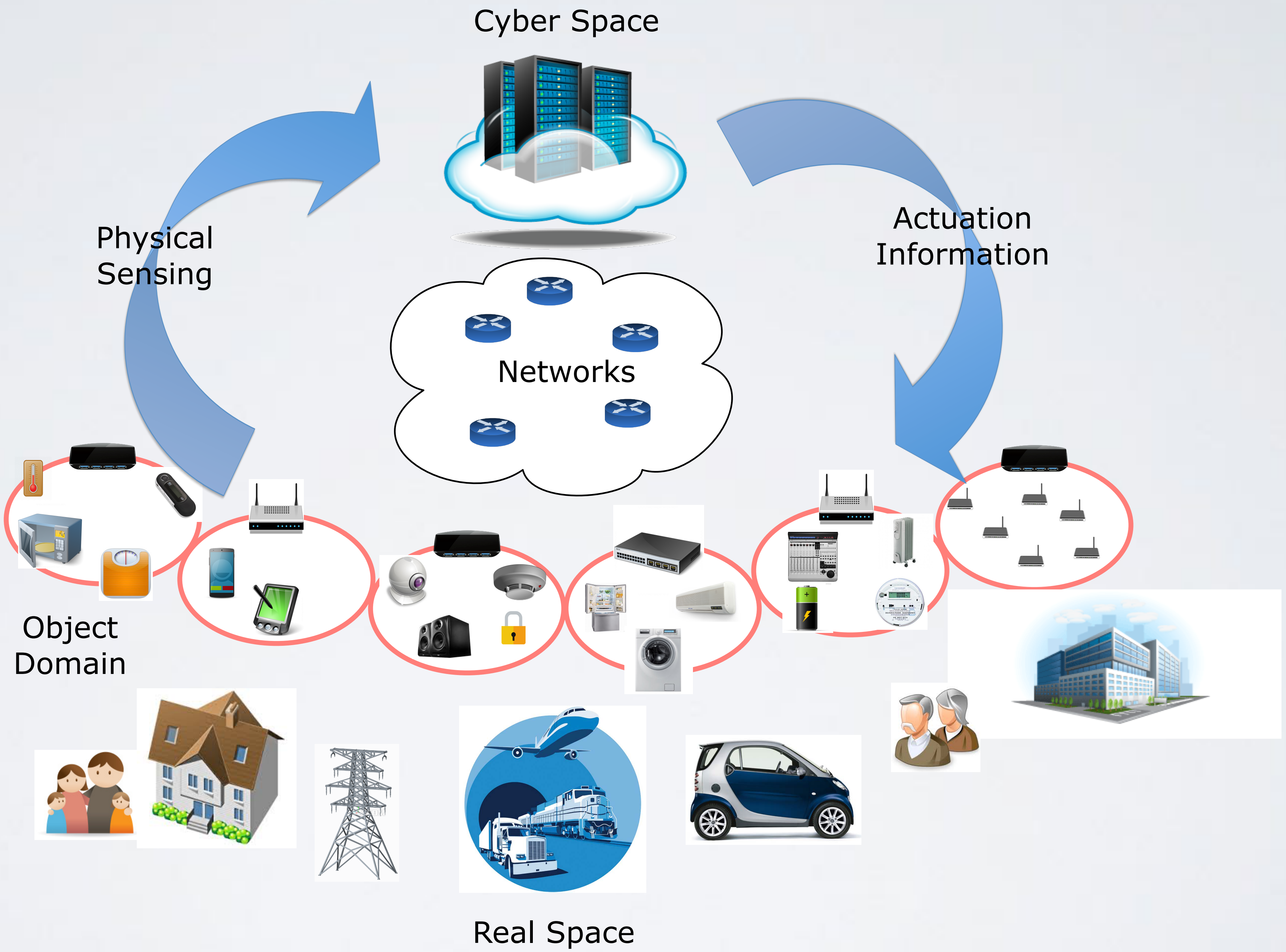
Raja
Ben Abdessalem

Reza
Matinnejad

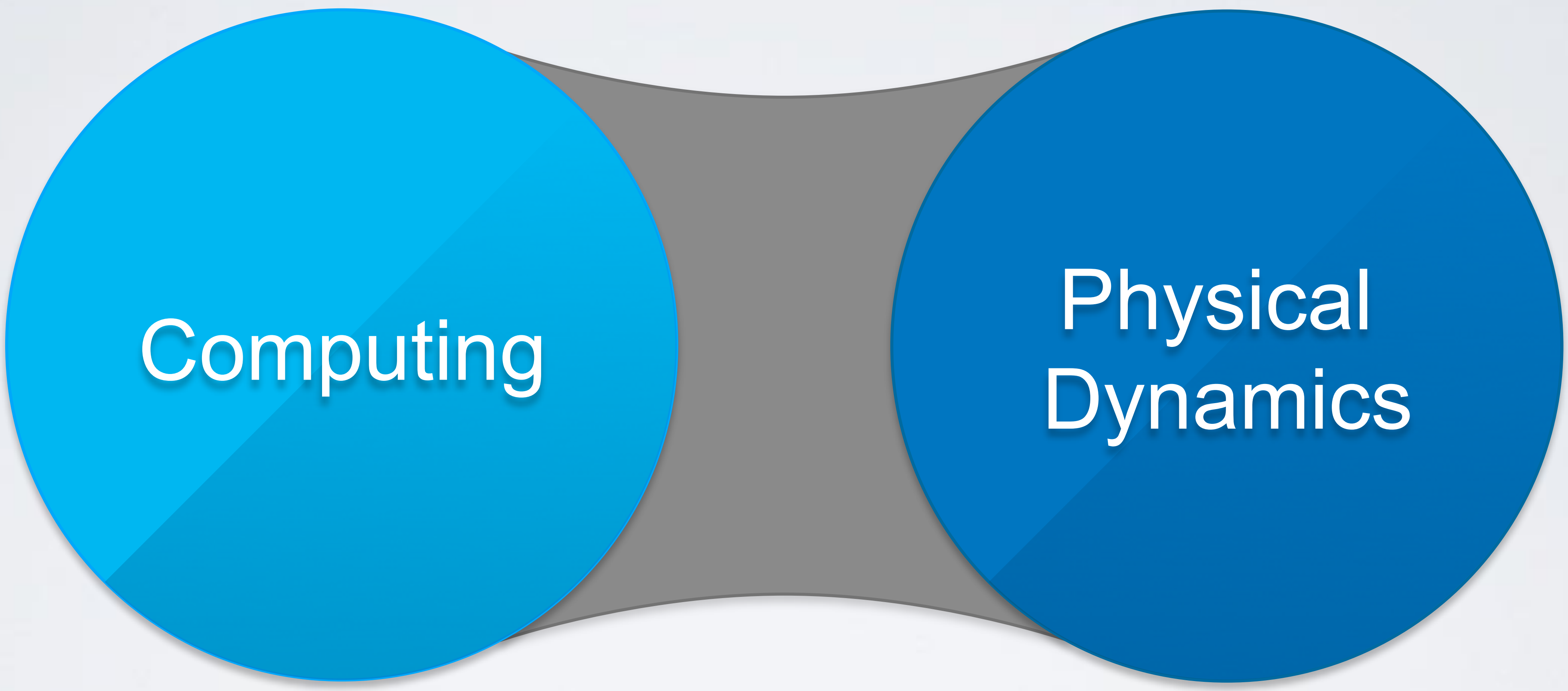
Annibale
Panichella

Lionel
Briand

Cyber Physical Systems (CPS)

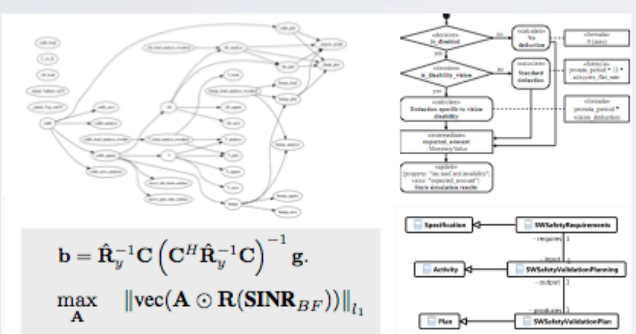


CPS Challenge



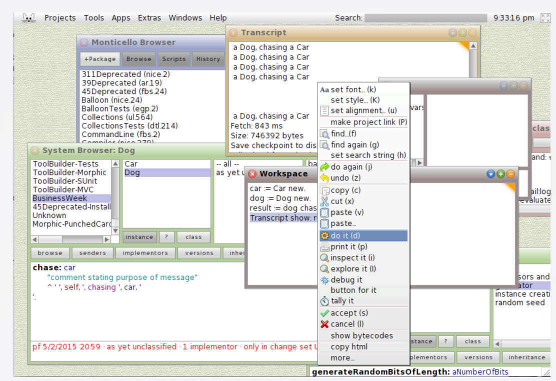
Model-based Development of CPS

Function Modeling



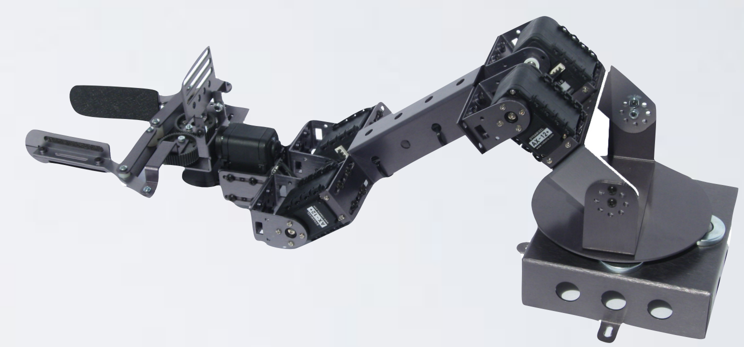
Model in the Loop (MiL)

Software Modeling/ Development



Software in the Loop (SiL)

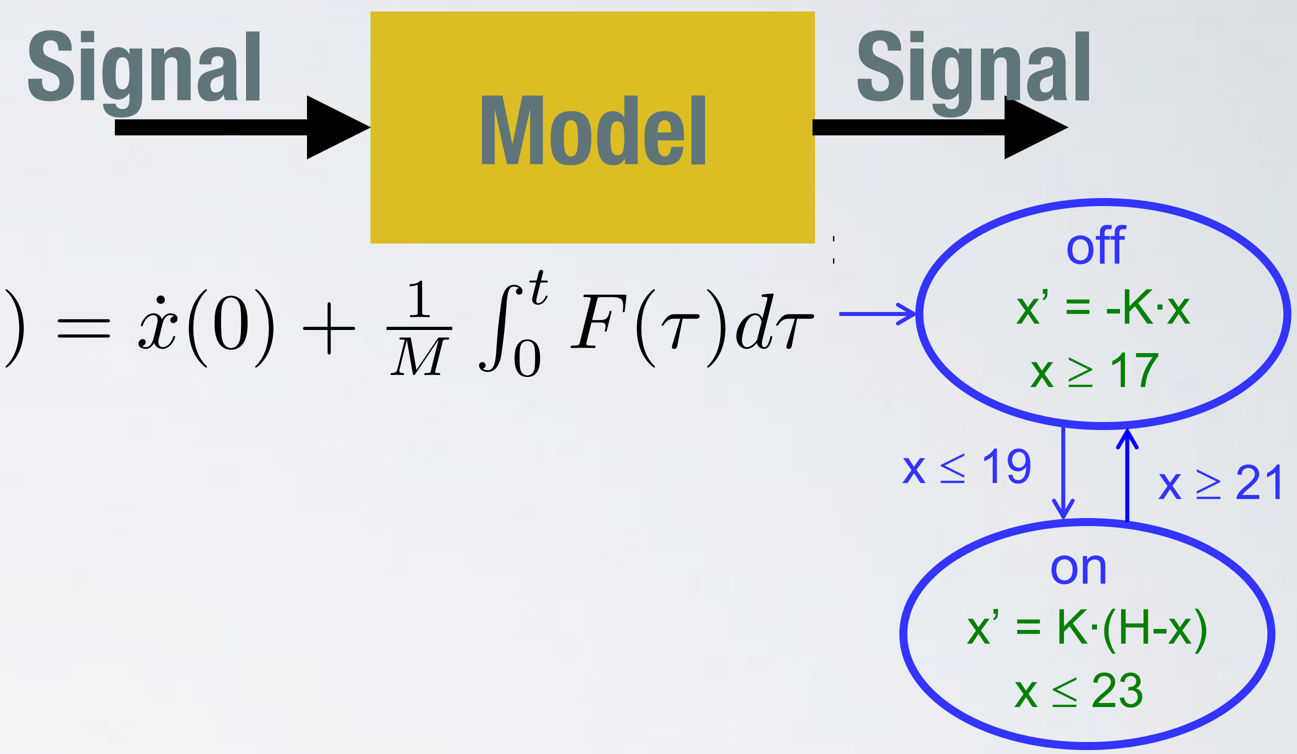
Integration of SW and HW



Hardware in the Loop (HiL)

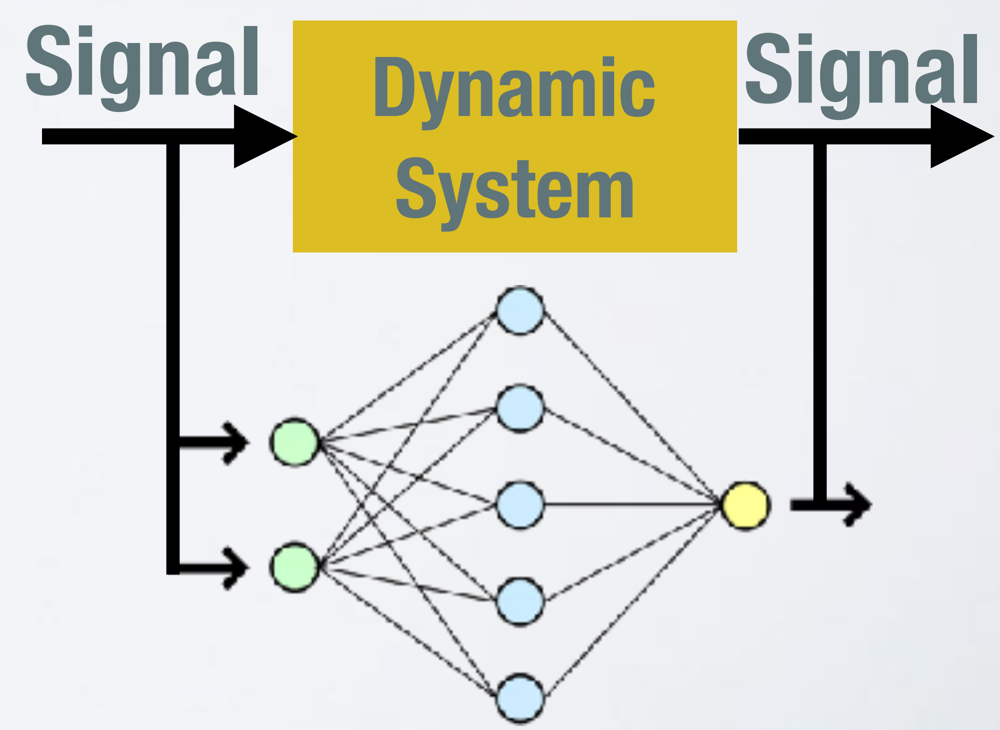
Function Models

- are **hybrid** – capture both **discrete** (algorithms) and **continuous** (physical dynamics) computations



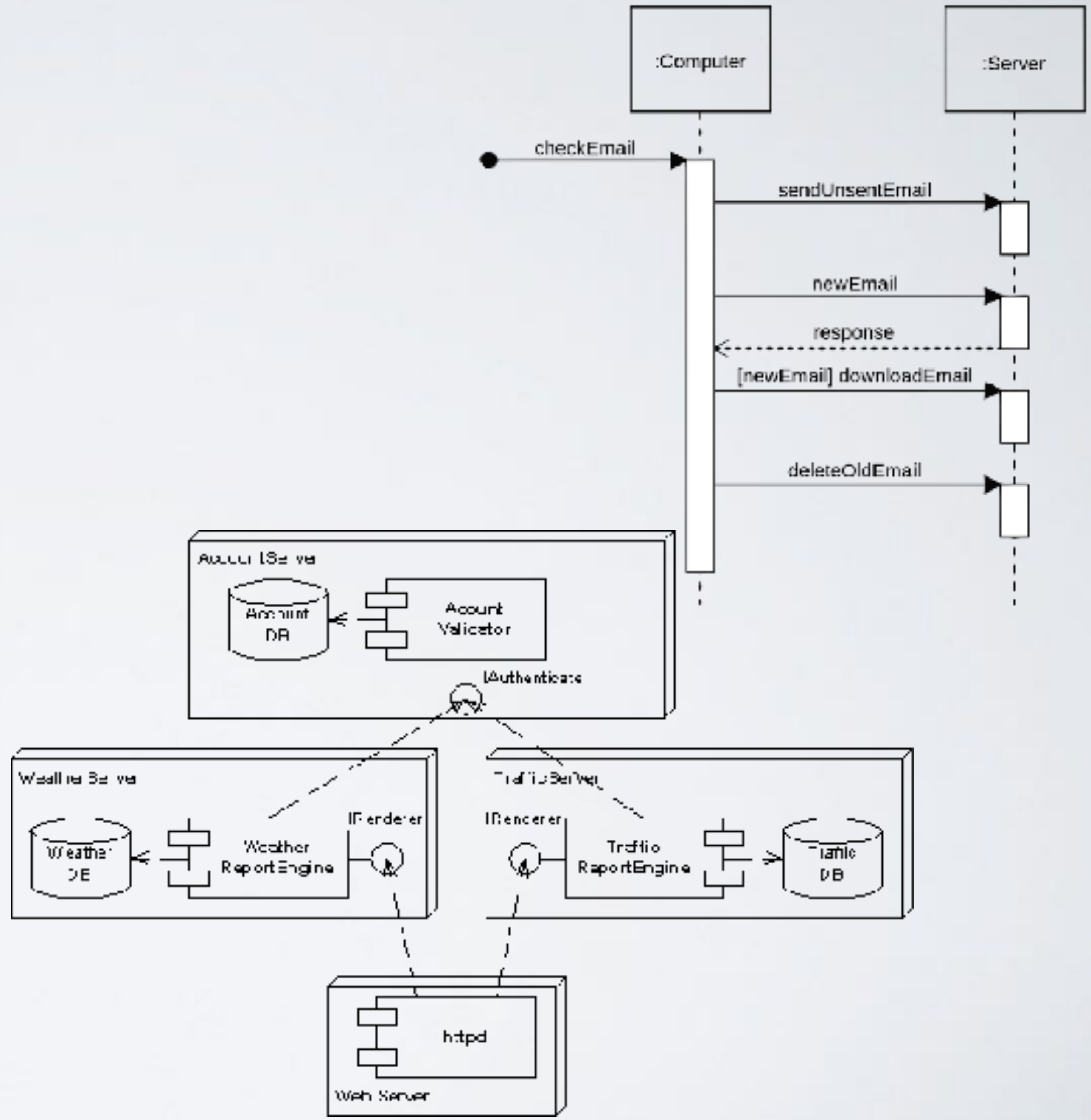
- are **executable**

- capture **uncertainty** e.g., about the environment



Software Models

- capture software **architecture** and **real-time** constraints
- specify **performance**, **security** and **timing** requirements
- are in charge of **integrating** different components
- are **heterogeneous**



Benefits of CPS Modelling

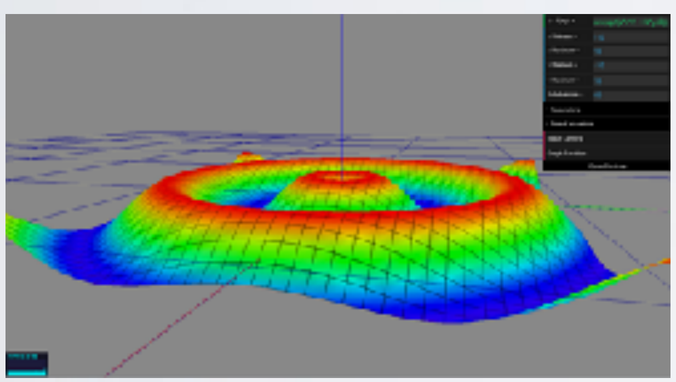
Automated Code Generation

Early Testing Verification

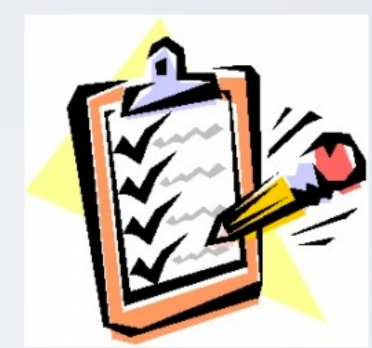
CPS Models

Software Model

Function Model



Simulation/ Prediction

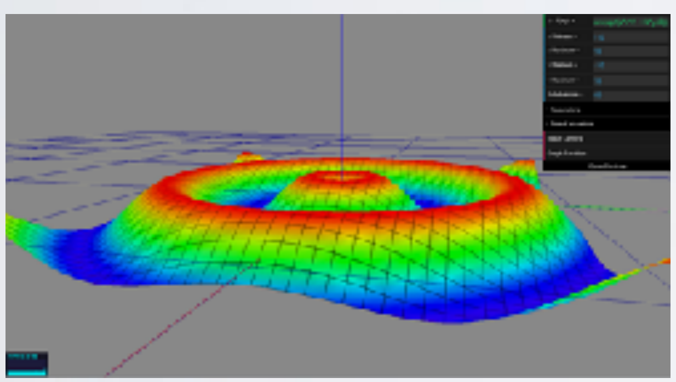


Certification

Benefits of CPS Modelling

Automated Code Generation

Early Testing Verification

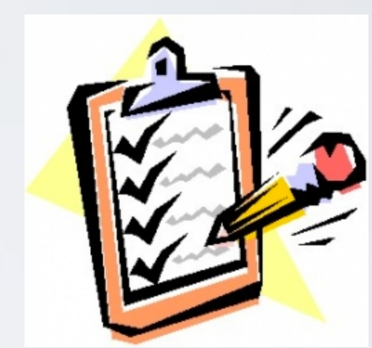


Simulation/ Prediction

CPS Models

Software Model

Function Model



Certification

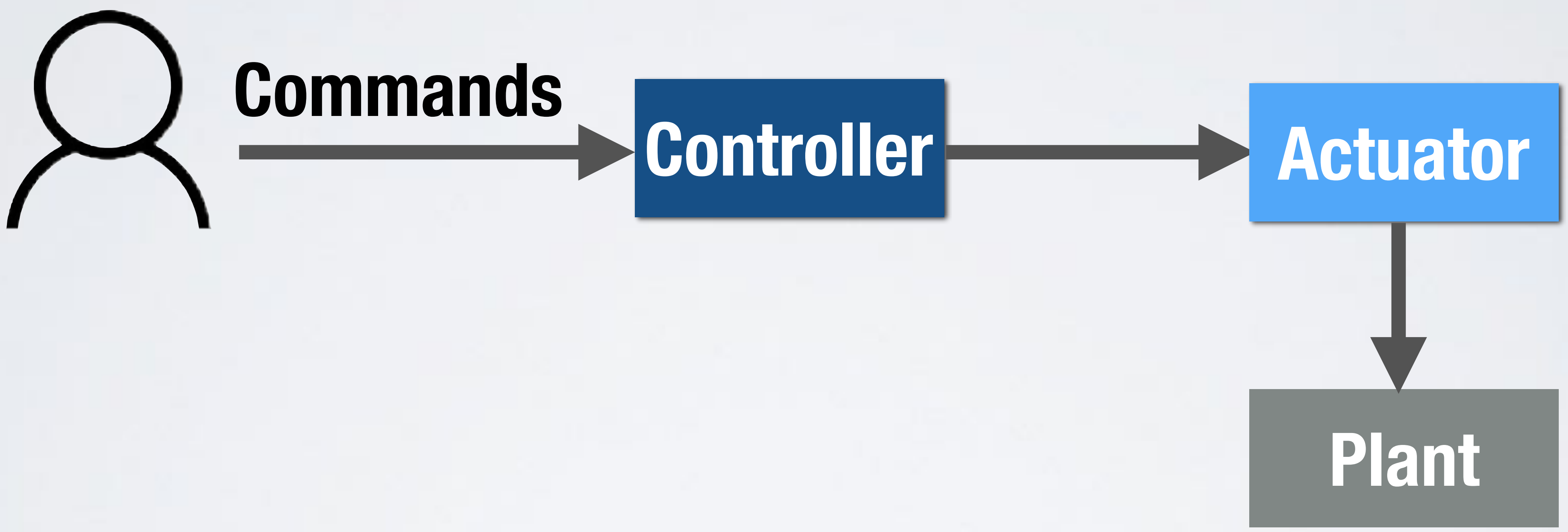
Fundamental Questions

- What are the **useful and realistic models** of CPS?

- What **requirements** should CPS satisfy to meet their **safety standards**?

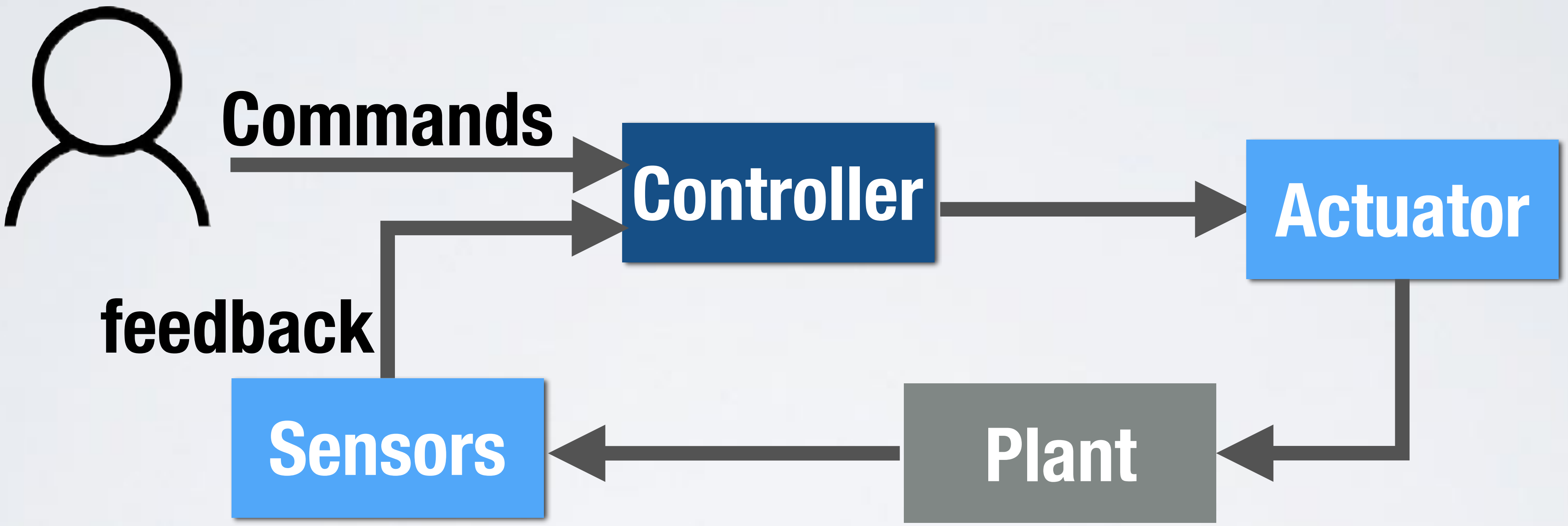
- What are the main challenges in developing **scalable and effective testing** techniques for CPS?

Simple Controller



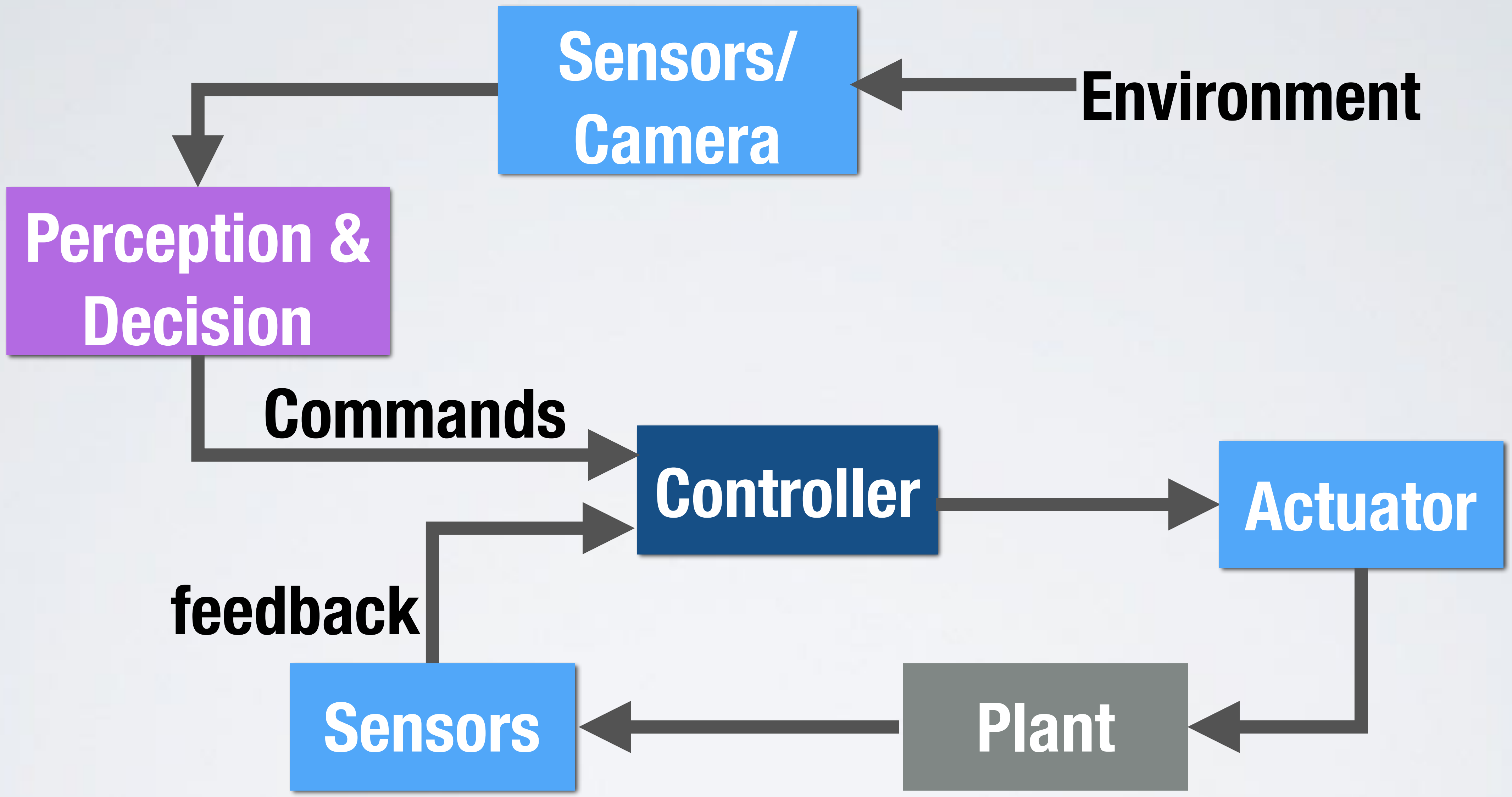
Electronic dryer controller

Adaptive Controller



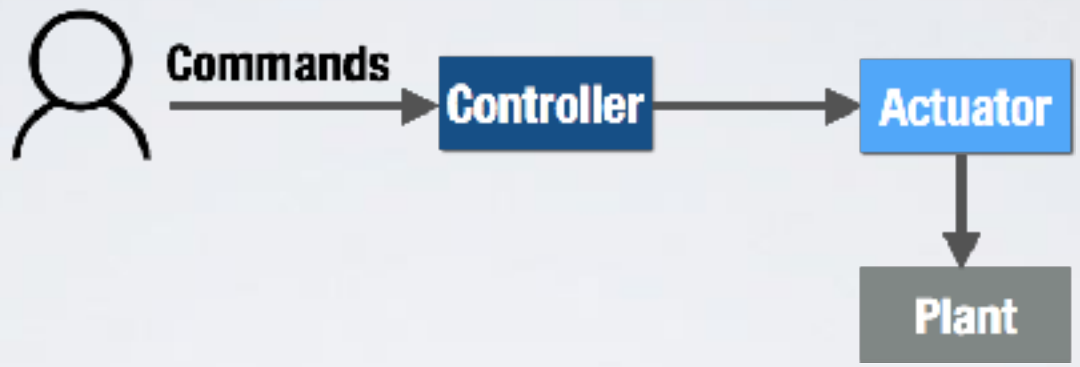
Cruise control system, Satellite controller

Autonomous Controller



Automated Driving, Unmanned Aerial Vehicle, Smart IoT

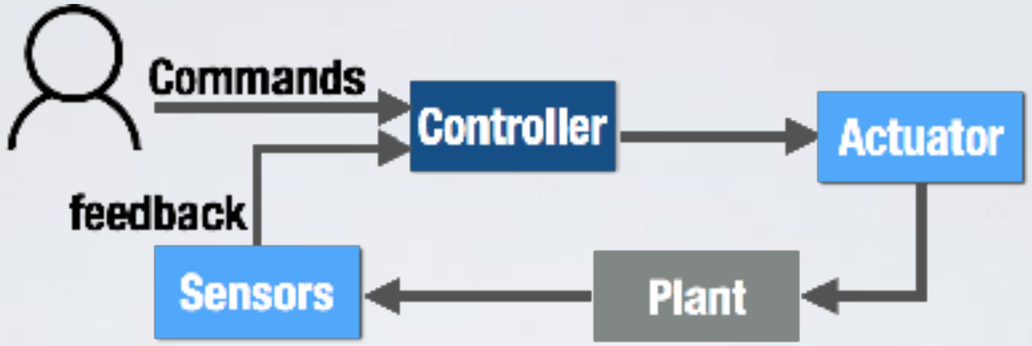
Temporal/Real Time Requirements



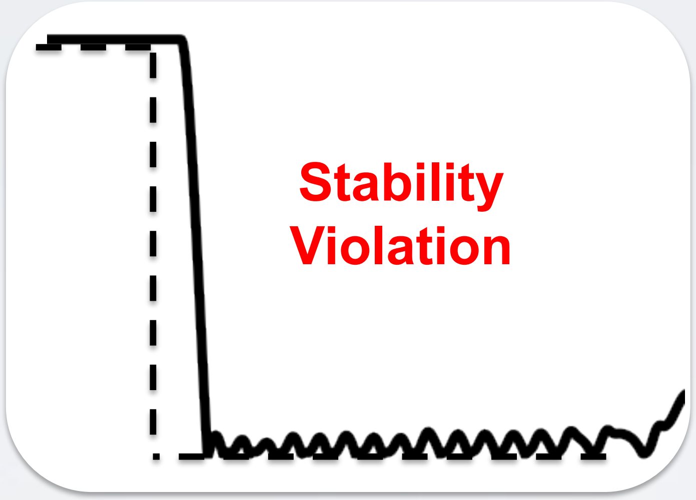
- ``As soon as braking is requested, the contact between Caliper and Disk shall occur within 20ms"

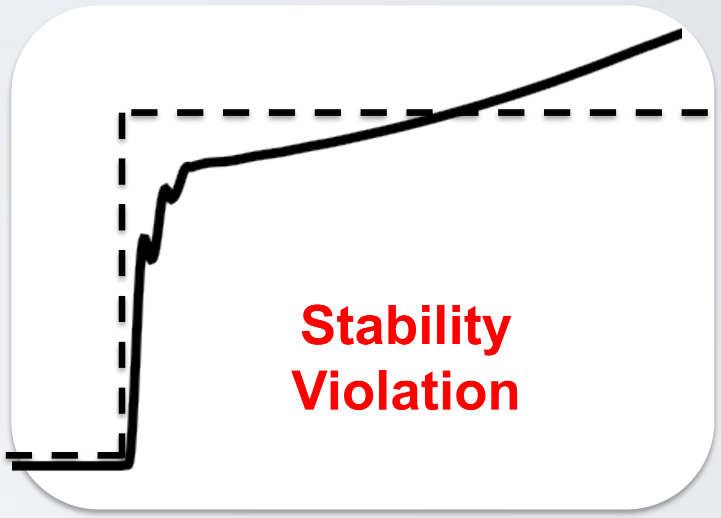
- ``The system shall respond within 32ms"

Controller Requirements

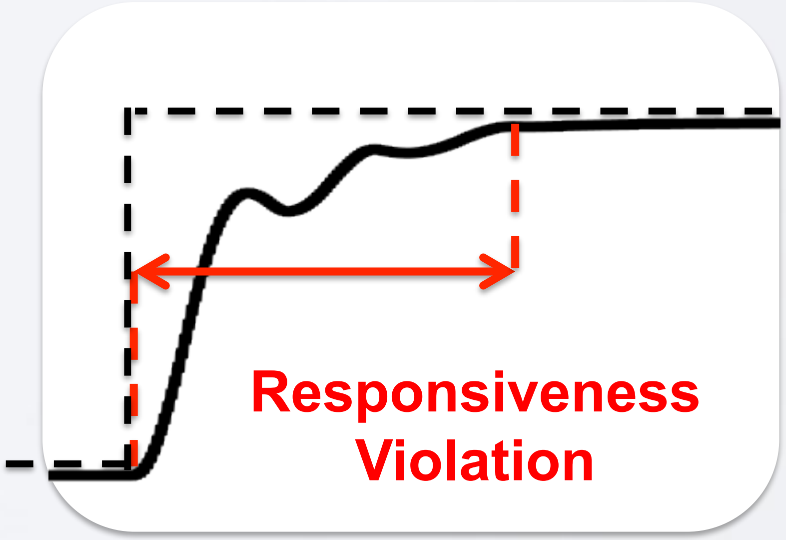


- Stability



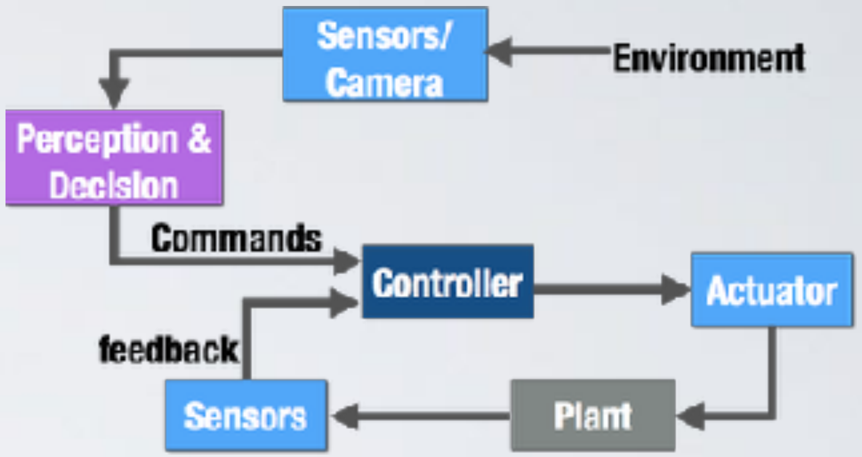


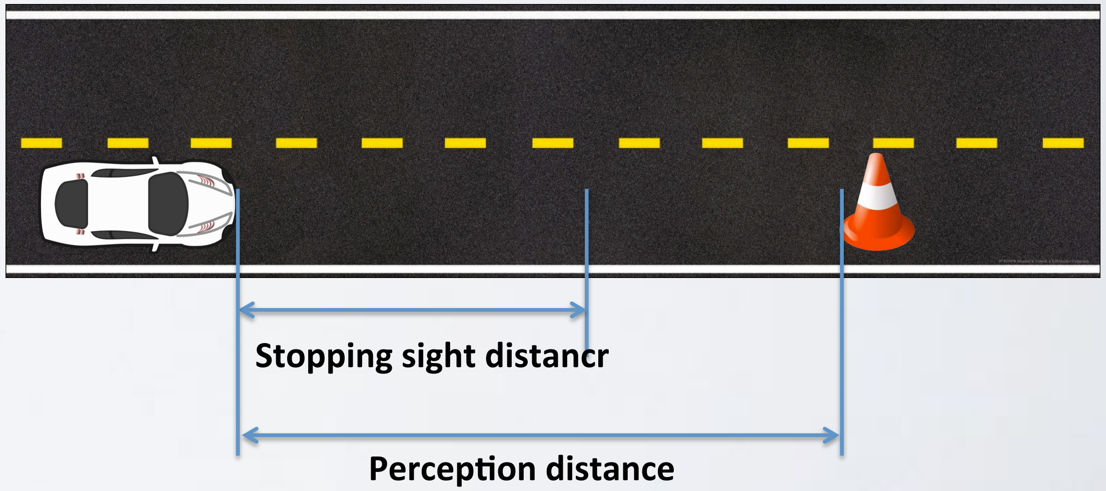
- Smoothness



Autonomous Systems

- Perception and decision requirements
  - ``The car shall detect all obstacles ahead of the vehicle within 100m distance."
  - ``An unintended braking manouvre by the Automated Emergency Braking shall be prevented."
- Behavioral Safety
- Driving Behavior Comfort
- Energy Efficiency
- ….

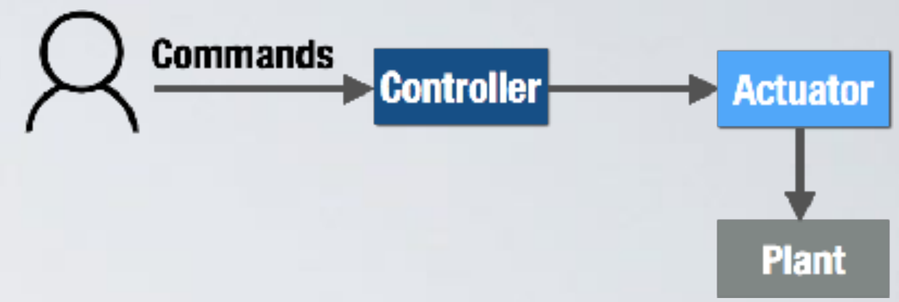




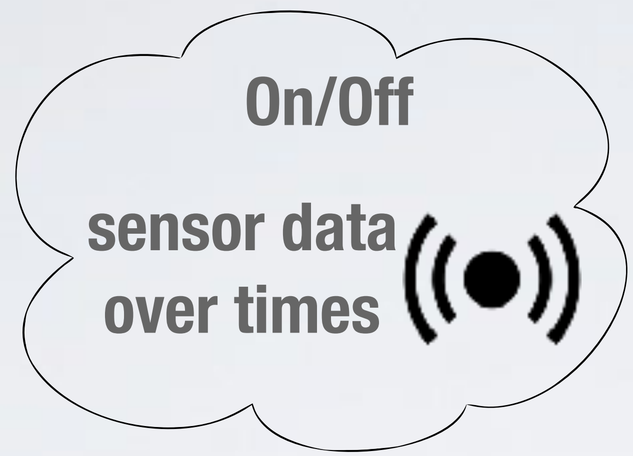
CPS Verification Challenge

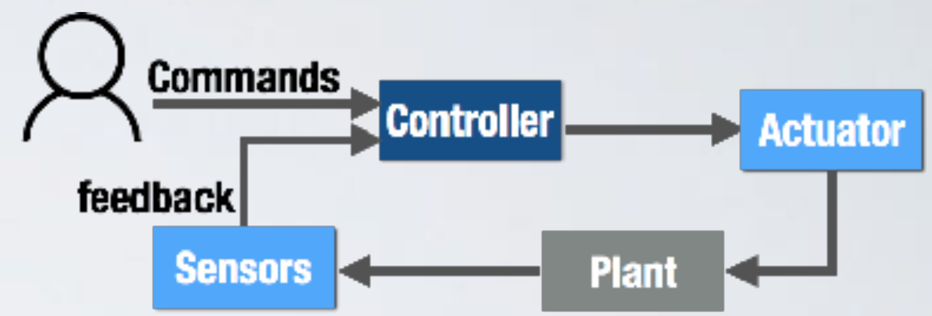
- Analytical techniques and exact solvers cannot be applied to CPS models due to
  - **non-linear, non-algebraic** computations
  - **continuous dynamic** behaviours
  - **heterogeneity**

commands

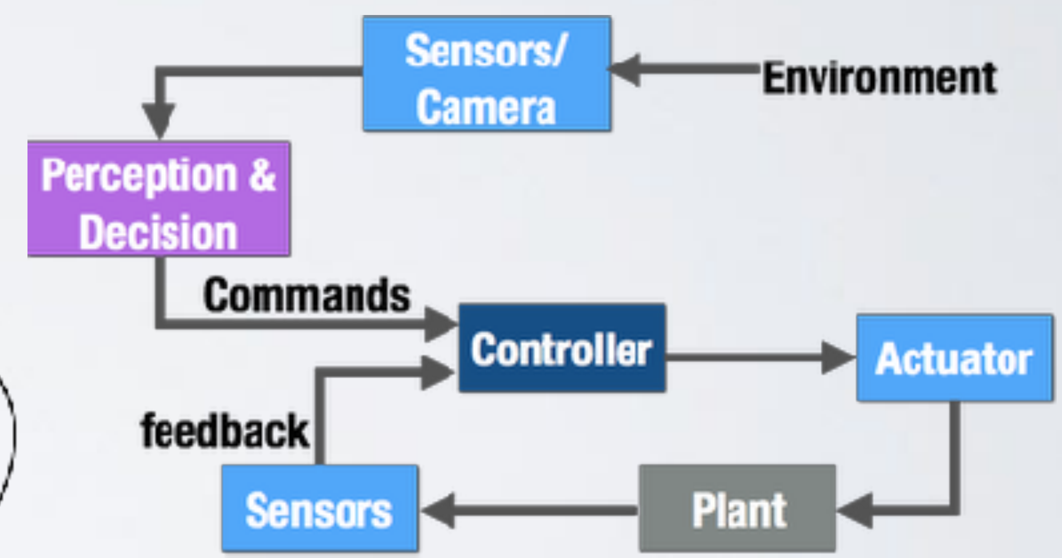


commands +
plant states





plant states +
environment

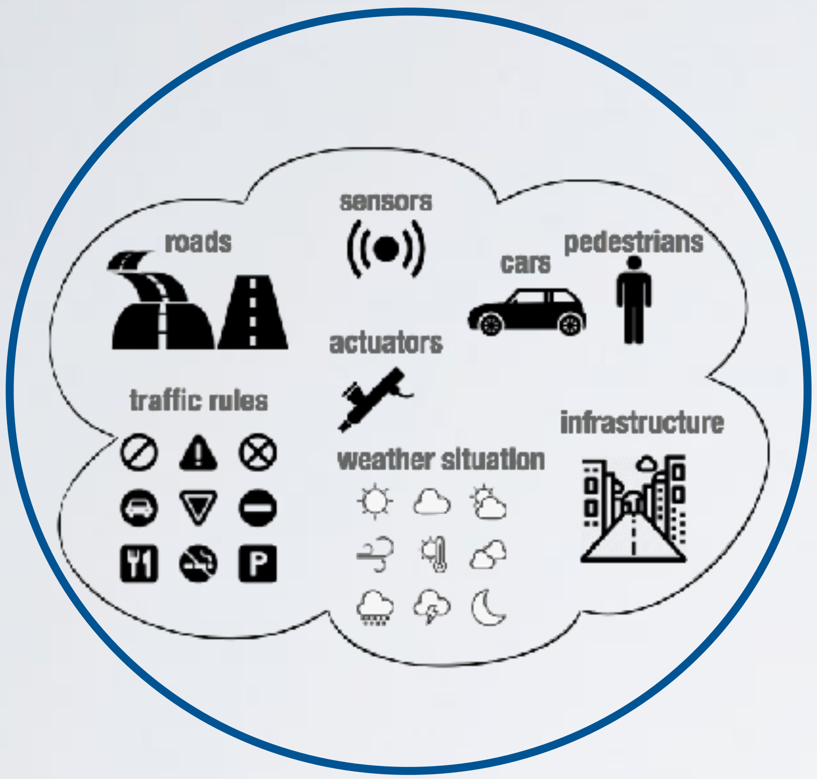


CPS test input spaces are large and multi-dimensional

Metaheuristic Search

- **Stochastic optimisation**, e.g., evolutionary computing
- Efficiently explore the search space in order to find **good (near optimal)** feasible solutions
- Applicable to any **search space** irrespective of the **size**
- **Flexible** and can be combined with different **optimisation methods**
- Amenable to analysis of **heterogeneous** models
- Applicable to many **practical situations**, including **SW testing**

Our Approach in a Nutshell

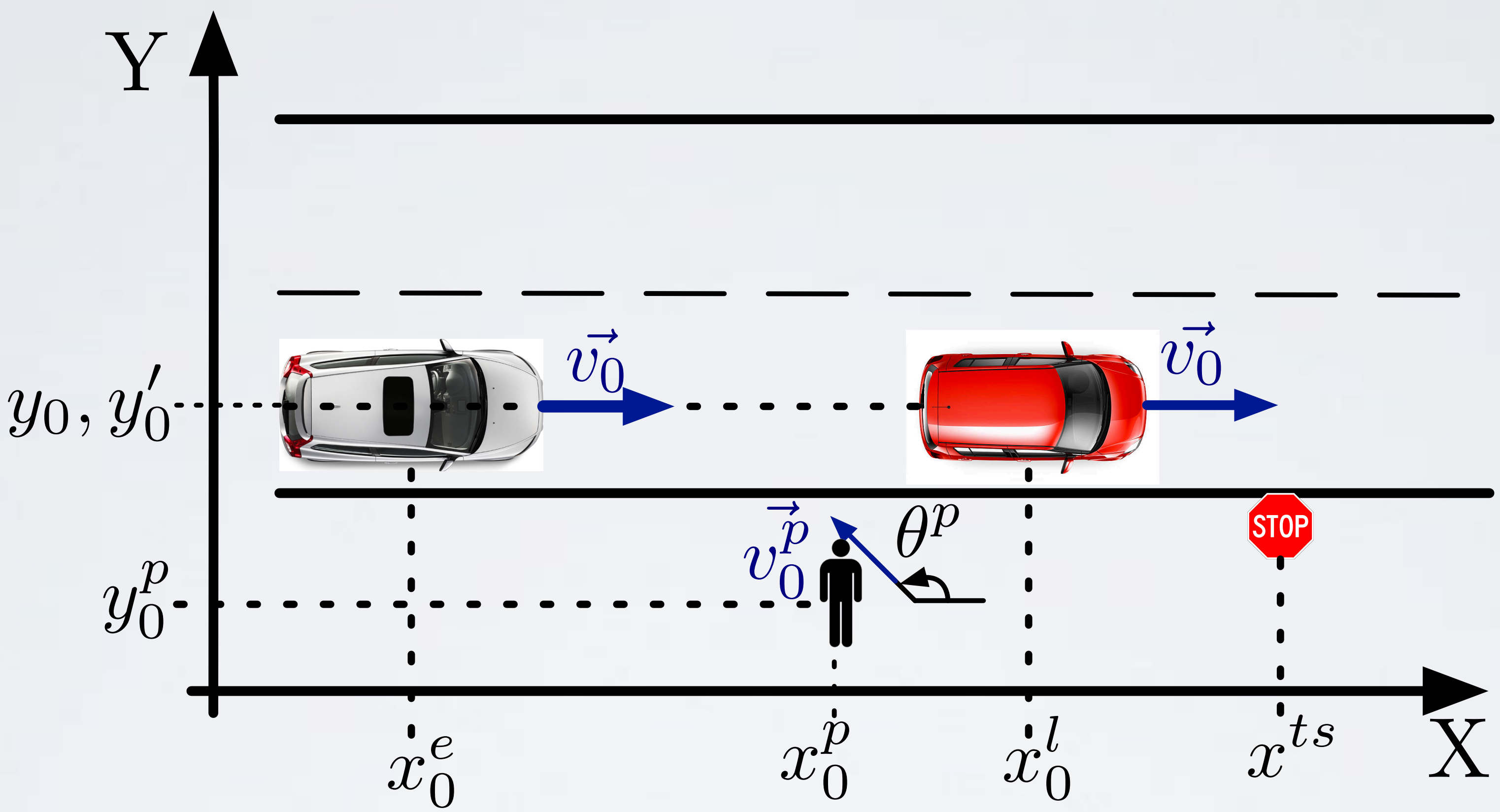


Test Input Generation

Guided Search

Optimisations via Machine Learning

Structured Test Inputs



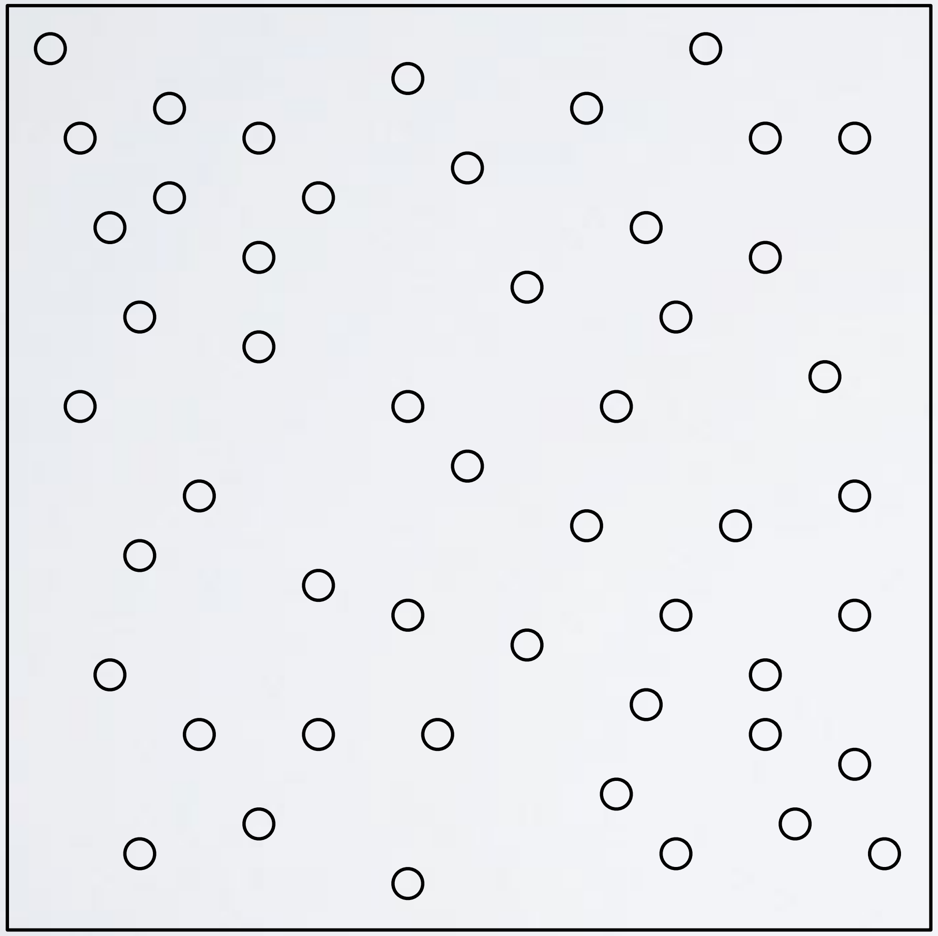
- Domain models
- Vectors and constraints

Genetic Algorithms

Search algorithms inspired by the theory of evolution

Genetic Algorithms

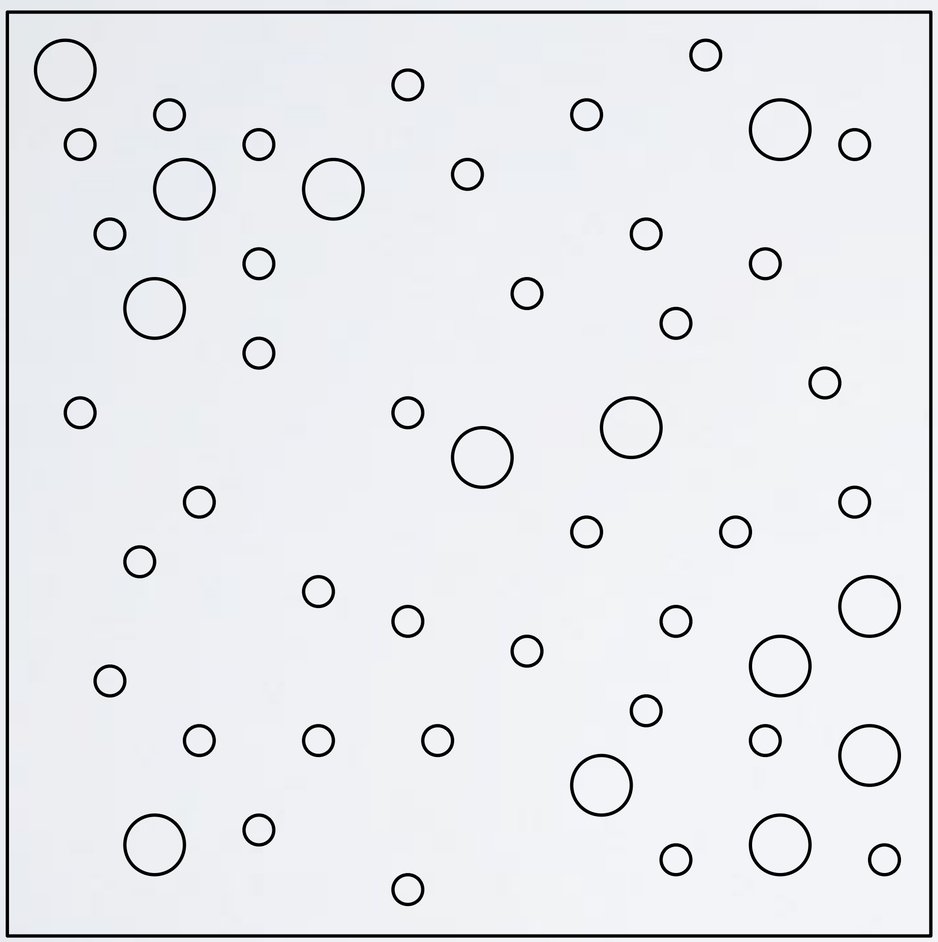
Search algorithms inspired by the theory of evolution



Initial test inputs

Genetic Algorithms

Search algorithms inspired by the theory of evolution

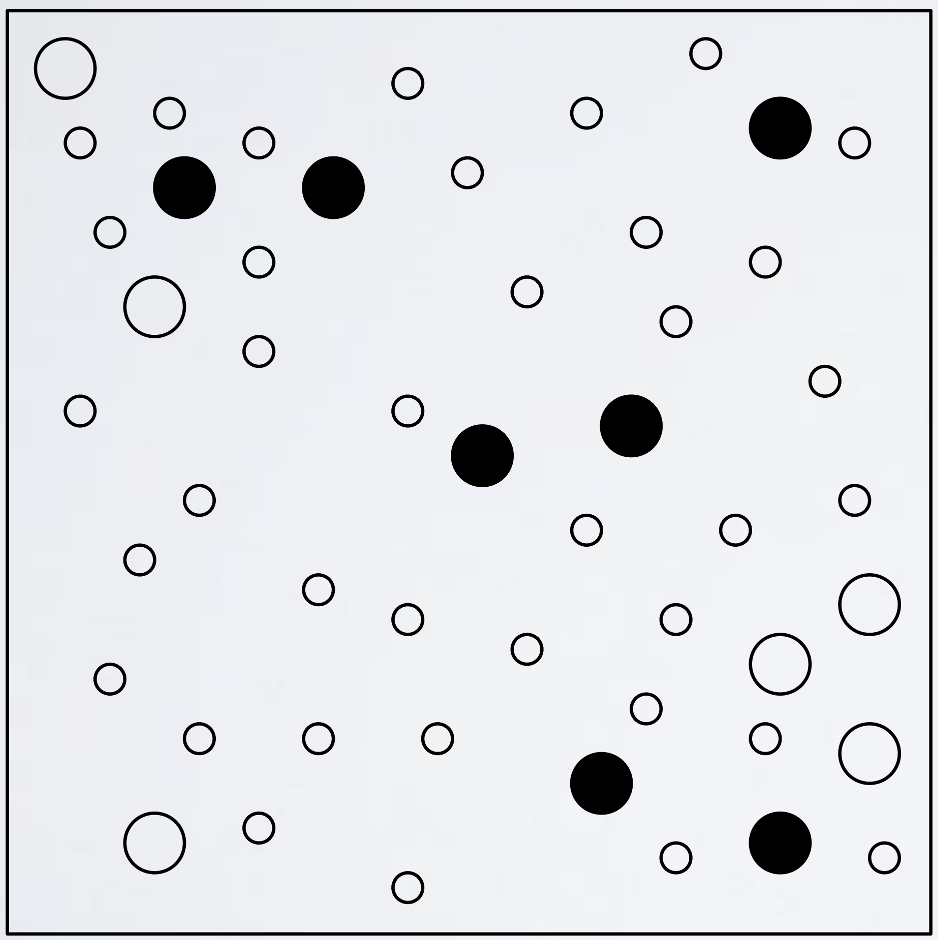


Initial test inputs

Fitness computation (which test is more likely to reveal faults?)

Genetic Algorithms

Search algorithms inspired by the theory of evolution



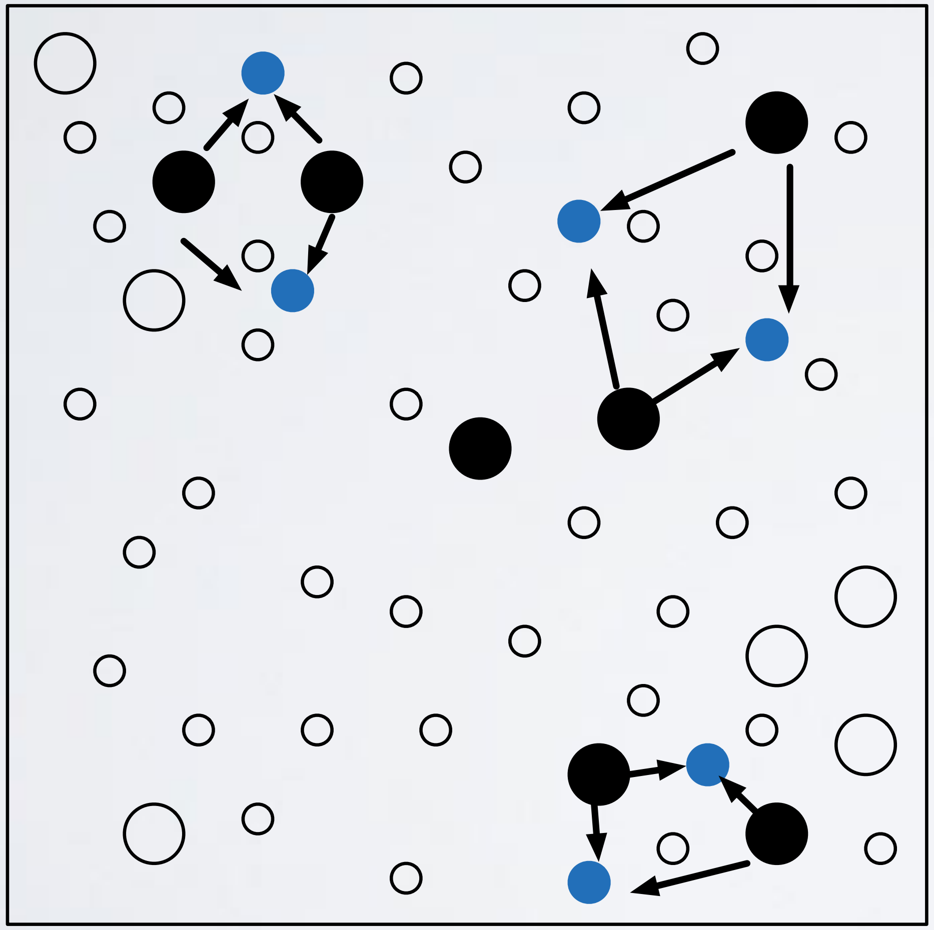
Initial test inputs

Fitness computation (which test is more likely to reveal faults?)

Select the most critical tests (the ones more likely to reveal faults)

Genetic Algorithms

Search algorithms inspired by the theory of evolution



Initial test inputs

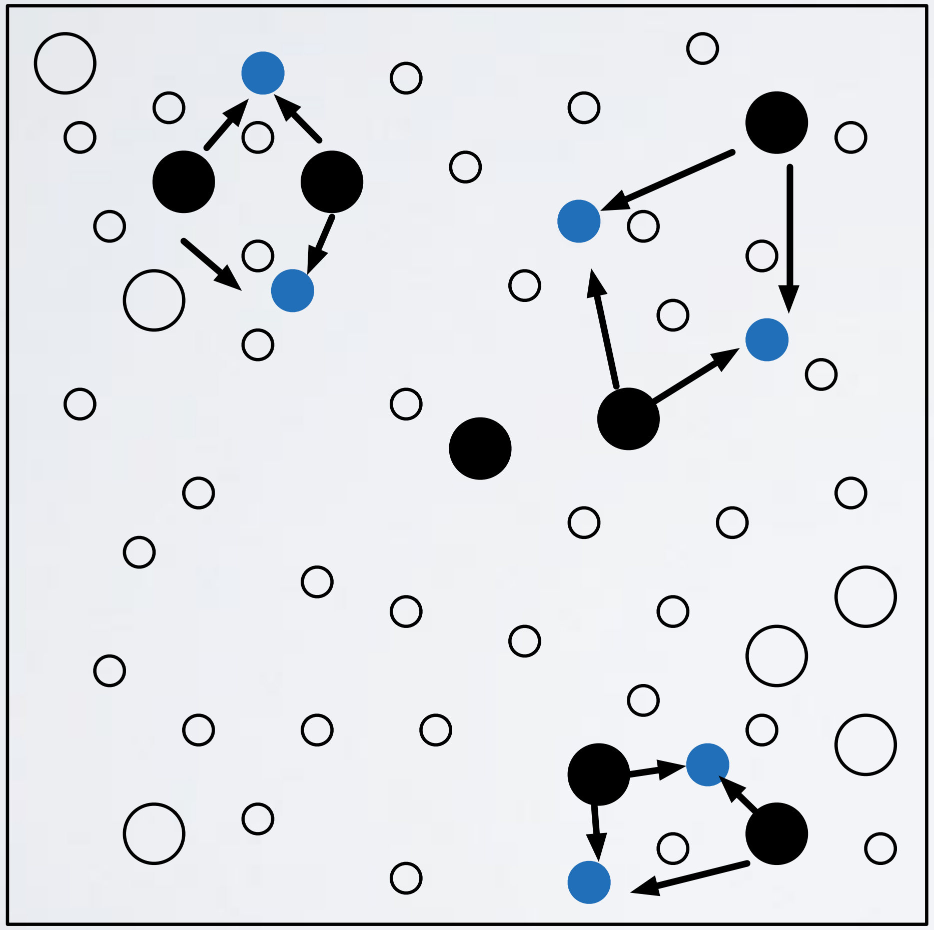
Fitness computation (which test is more likely to reveal faults?)

Select the most critical tests (the ones more likely to reveal faults)

Bread (generate new tests using Genetic operators)

Genetic Algorithms

Search algorithms inspired by the theory of evolution



Initial test inputs

Fitness computation (which test is more likely to reveal faults?)

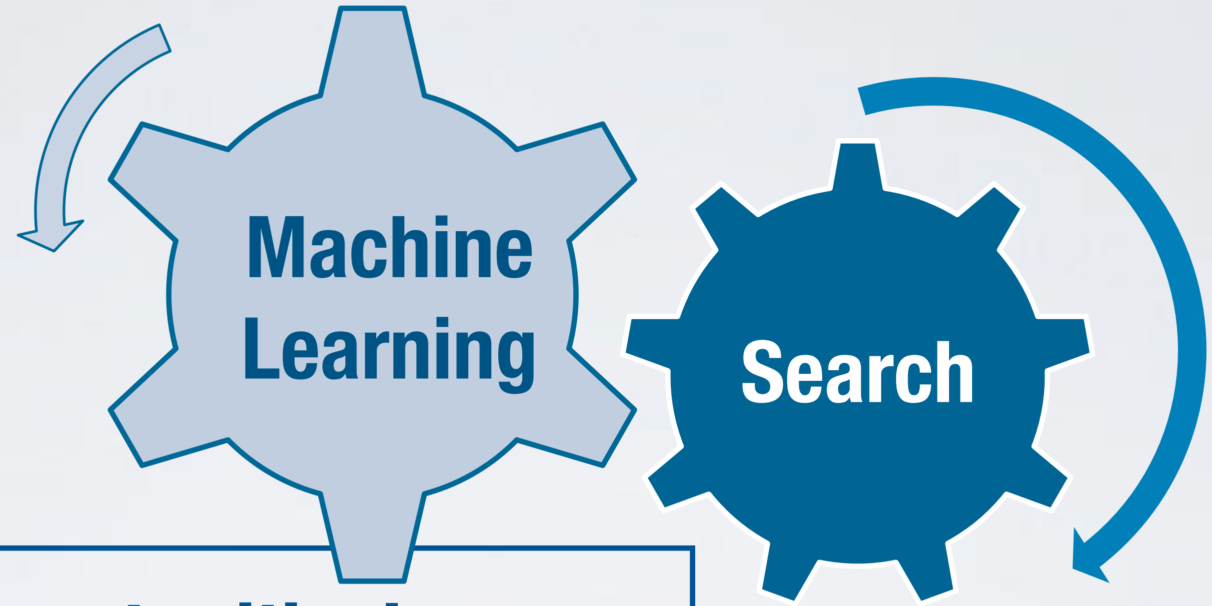
Select the most critical tests (the ones more likely to reveal faults)

Bread (generate new tests using Genetic operators)

Why Do We Need Additional Optimizations?

- **Few objective function evaluations** are possible because executing/simulating CPS function models is **expensive**
  - They should be executed for a long enough time duration
  - They capture, in addition to software/controllers, models of **hardware and environment**
- **Several local-optima**
- **Large** and **multi-dimensional** search input spaces

Machine Learning and Search

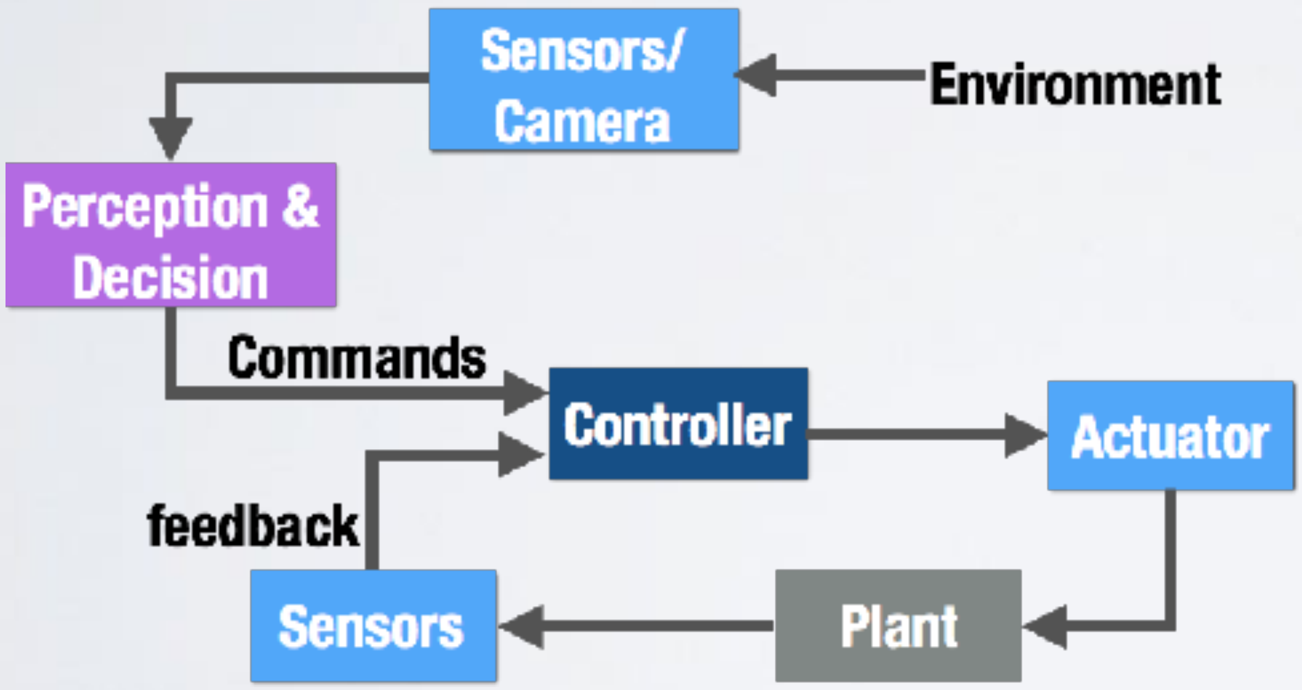


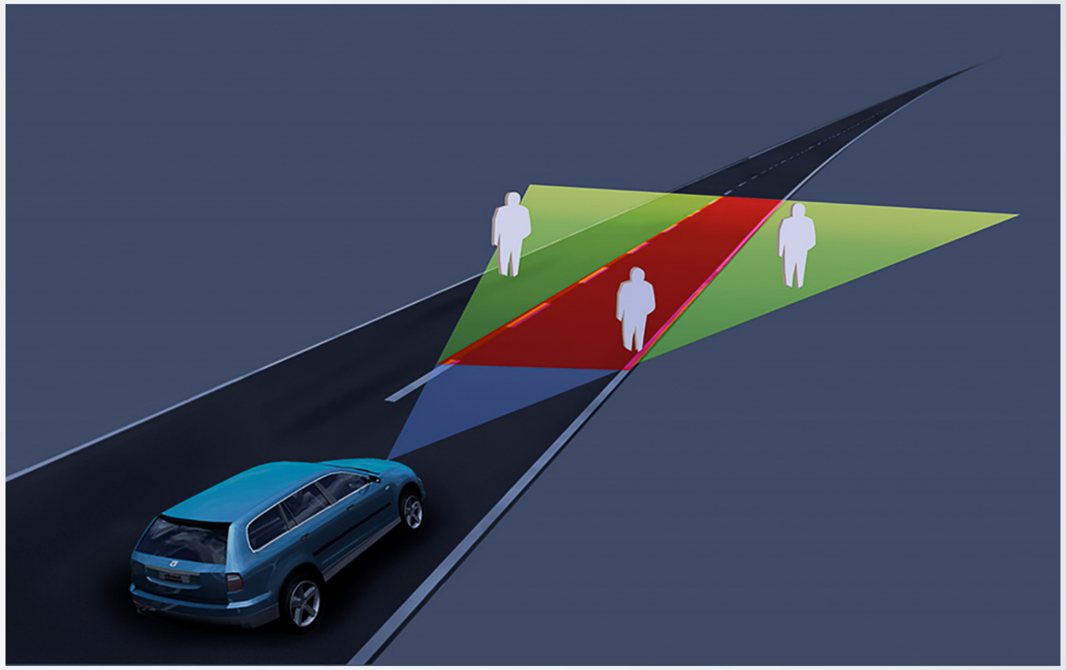
- **Learning** where the **most critical regions** are
- **Learning fitter solutions** instead of breading them
- **Predicting fitness values** instead of computing them
- **Selecting** effective search algorithms and **tuning** their parameters
- …

Find **critical test inputs** in the entire search space

Industrial Research Projects

Testing Automated Driving Systems

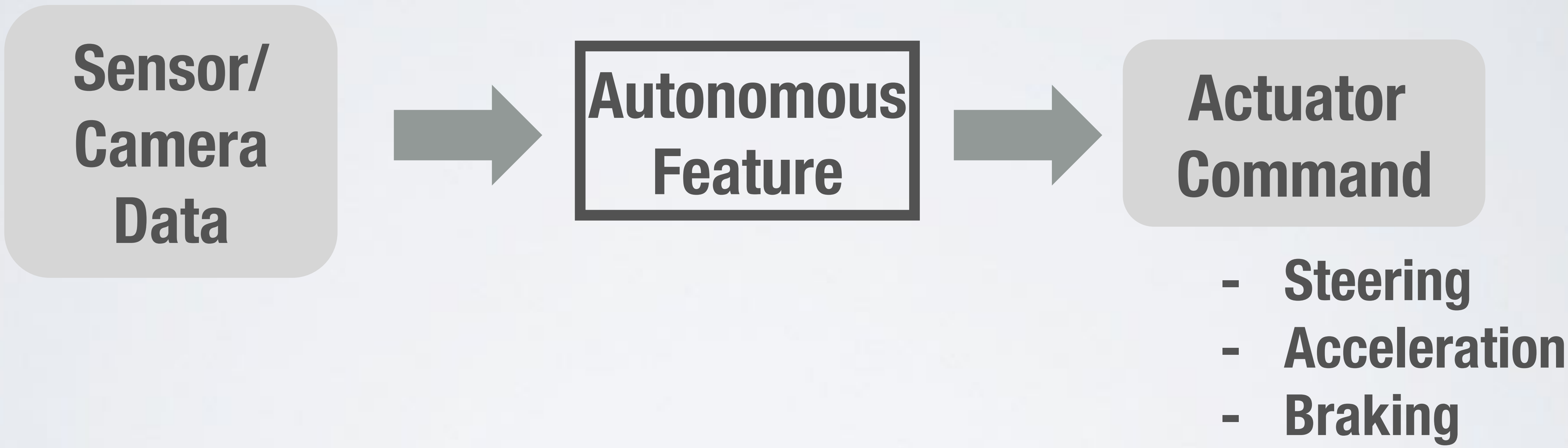




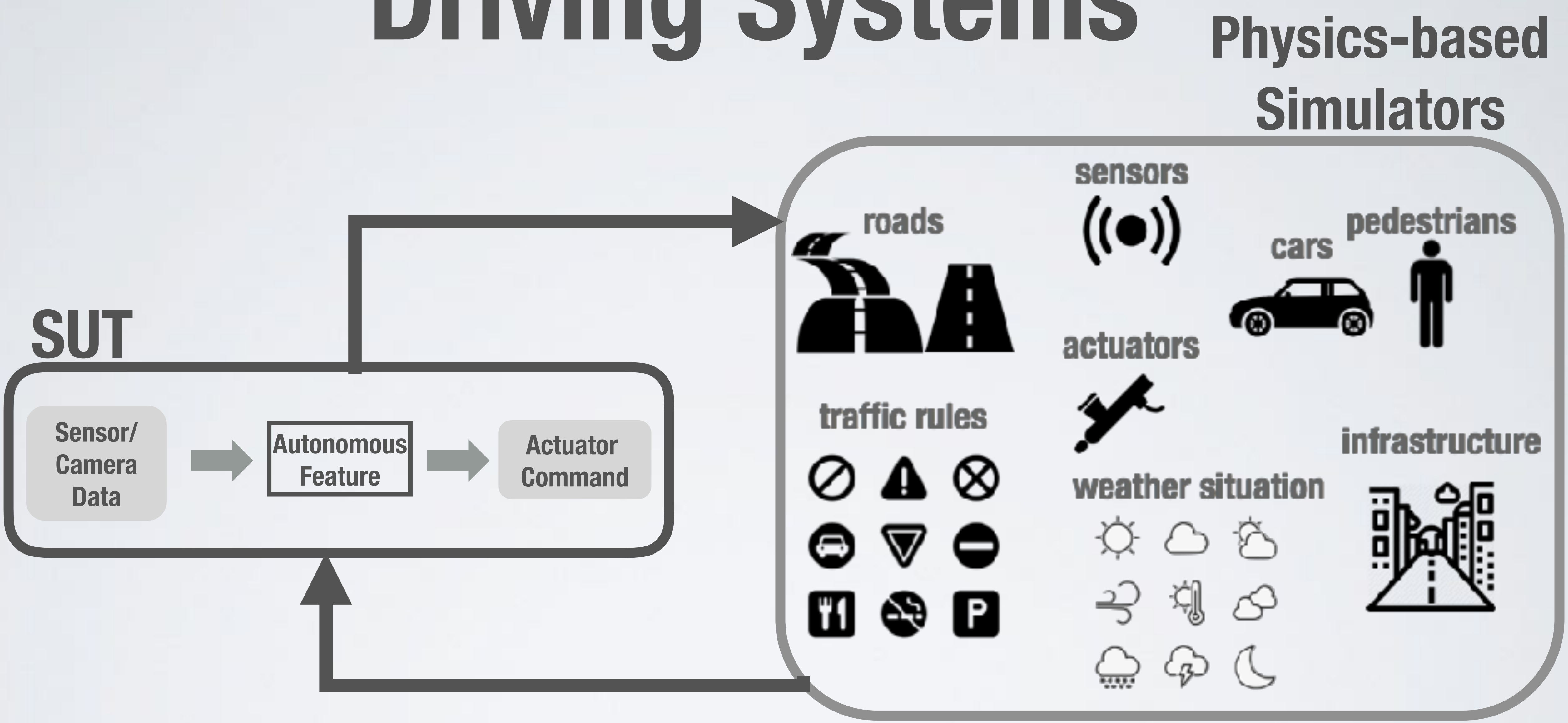
Autonomous Car Features

Automated Emergency Breaking
(AEB)

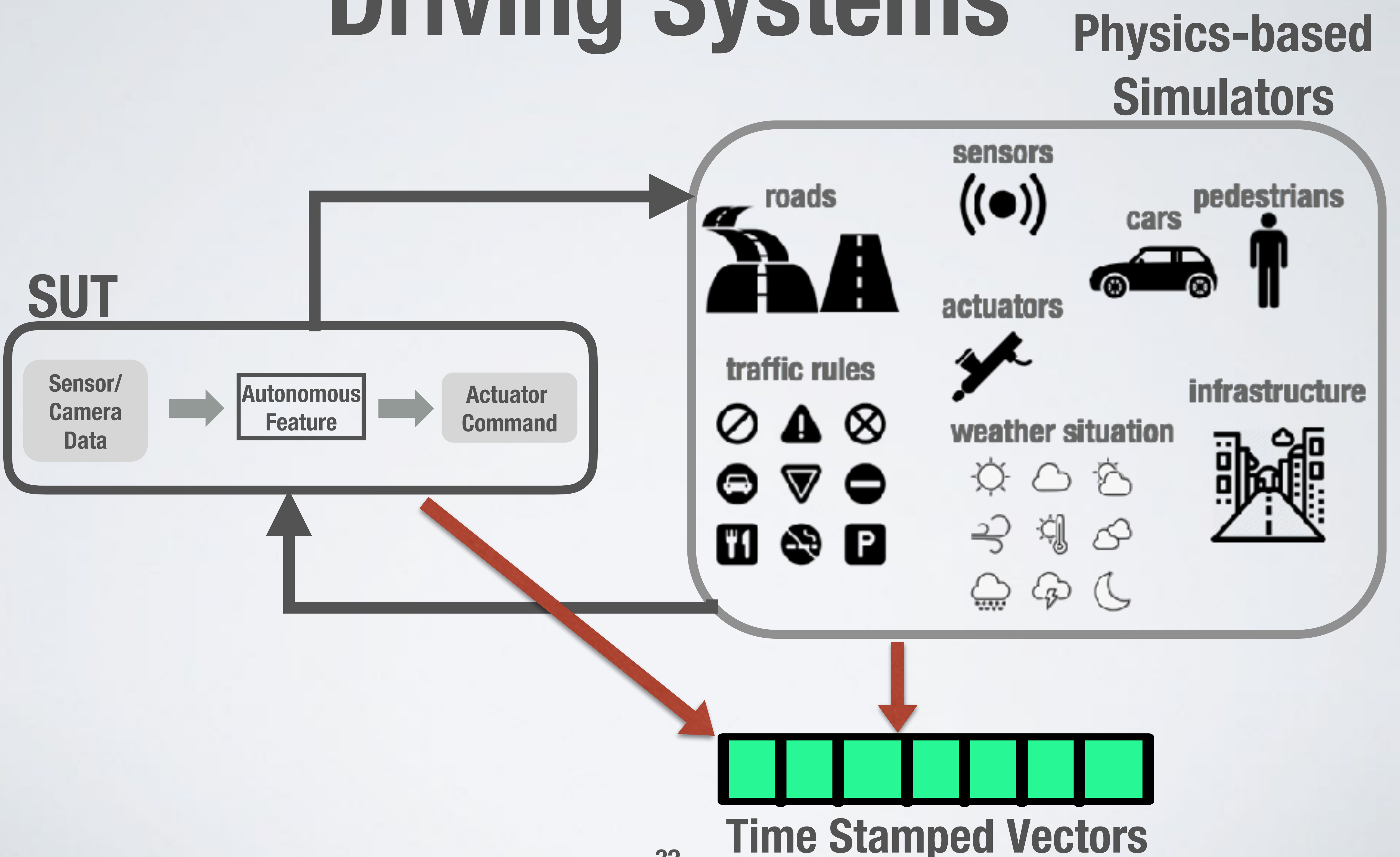
Traffic Sign Recognition
(TSR)


Sensor/
Camera
Data
Autonomous
Feature
Actuator
Command
- Steering
- Acceleration
- Braking

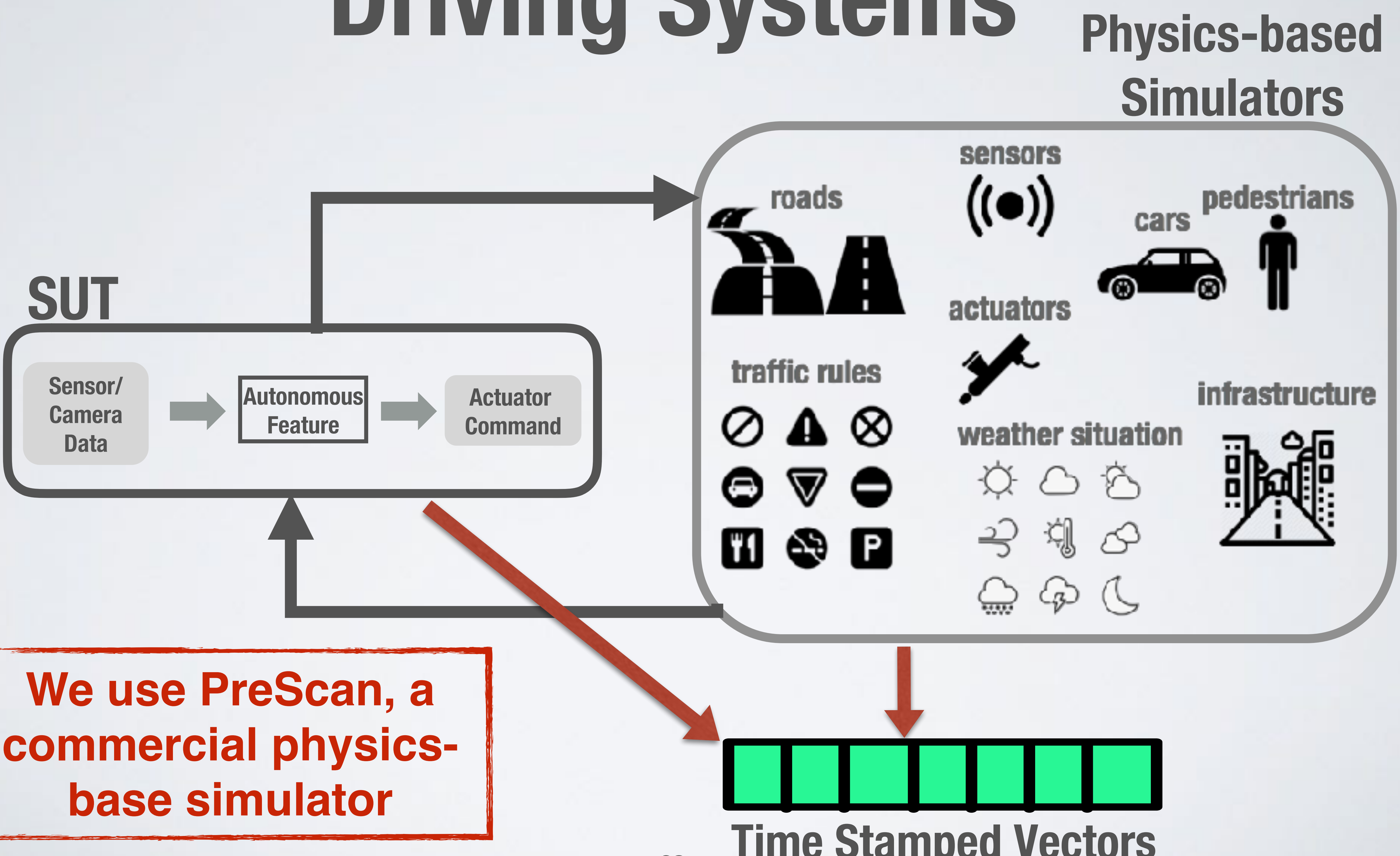
Testing Models of Automated Driving Systems



Testing Models of Automated Driving Systems

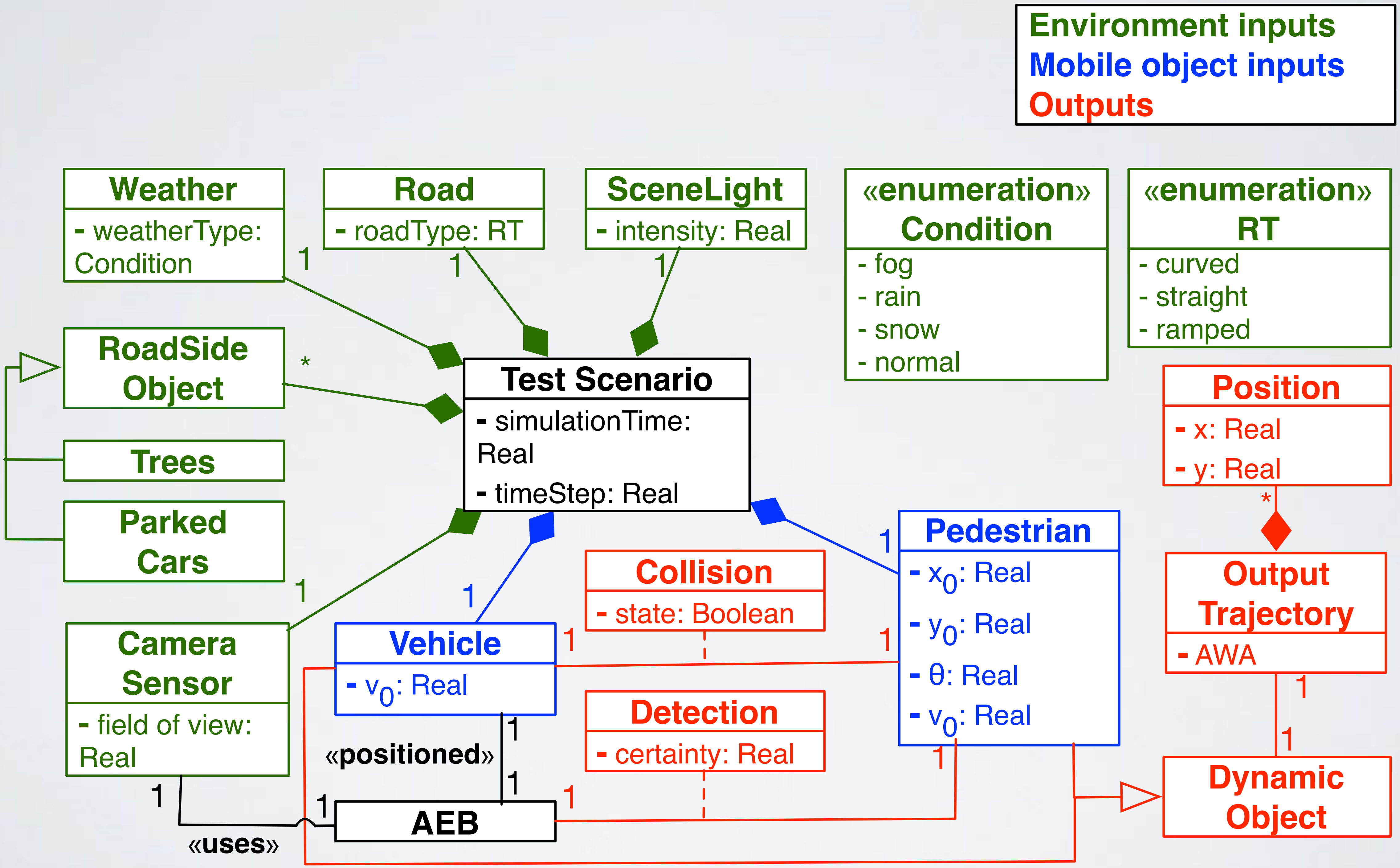


Testing Models of Automated Driving Systems



We use PreScan, a commercial physics-base simulator

Test Inputs/Outputs

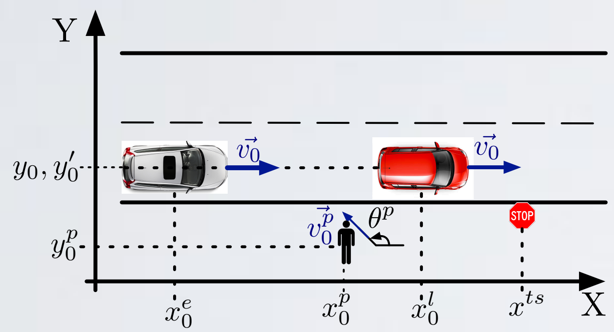


System Safety Requirements

- **Req1**: "Automated Emergency Braking (AEB) shall detect pedestrians in front of the car and stop the car when there is a risk of collision"
- **Req2**: "An unintended manoeuvre by AEB shall be prevented"
- **Fitness functions** estimate how close AEB is into violating its requirements (e.g., by having a collision)

Guided Test Generation

Test Input Characterisation



Test input generation

- Select best tests
- Generate new tests (Genetic Operators)

Evaluating test inputs

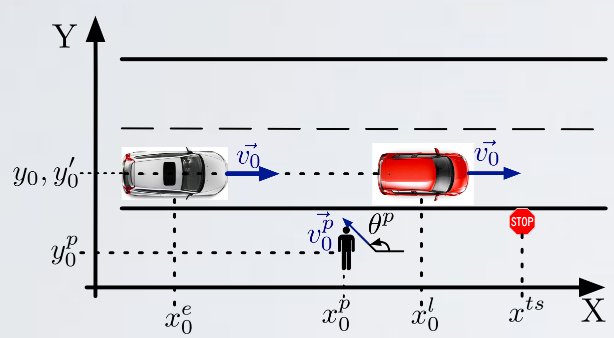
- Simulate every (candidate) test
- Compute fitness functions

Fitness values

Tests revealing requirements violations

Guided Test Generation

Test Input Characterisation



Test input generation

- Select best tests
- Generate new tests (Genetic Operators)

Evaluating test inputs

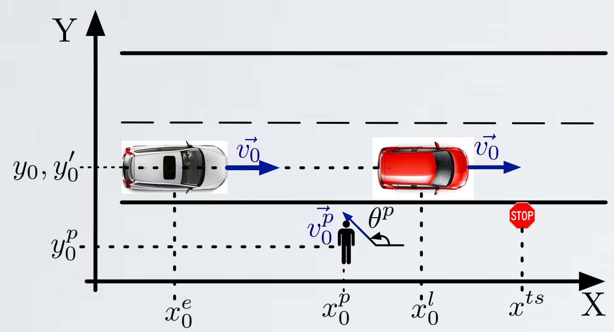
- Simulate every (candidate) test
- Compute fitness functions

Fitness values

Tests revealing requirements violations

Guided Test Generation

Test Input Characterisation



Test input generation

- Select best tests
- **Generate new tests (Genetic Operators)**

Evaluating test inputs

- **Simulate** every (candidate) test
- Compute **fitness functions**

Fitness values

Tests revealing requirements violations

But, simulations are expensive to run!

Surrogate Models

- It takes 8 hours to run our search-based test generation (≈500 simulations)

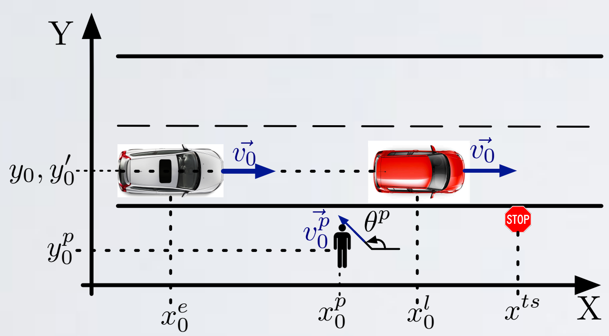
➡We use **surrogate models** developed based on machine learning to reduce the number of fitness computations

  - We first train a model based on a large number of simulations

  - We use this model during the search to predict fitnesses instead of actually computing them, but …

Guided Test Generation

Test Input Characterisation



Test input generation

- Select best tests
- Generate new tests (Genetic Operators)

Evaluating test inputs

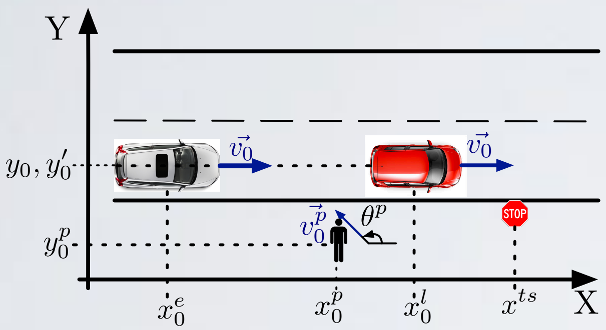
- Simulate every (candidate) test
- Compute fitness values

Fitnesses

Tests revealing requirements violations

Test Generation with Surrogates

Test Input Characterisation



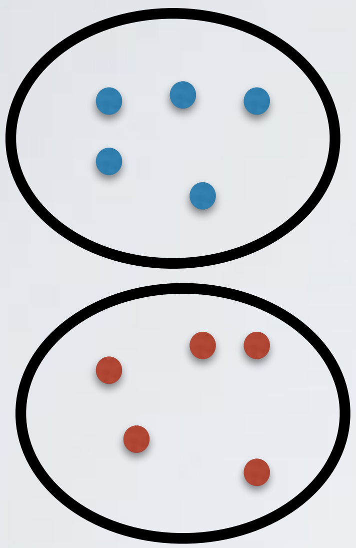
- Select best tests
- Generate new tests (Genetic Operators)

- Predict the fitness and the error (surrogate)
- If the test is likely to be selected
  - Simulate the test
  - Compute the fitness

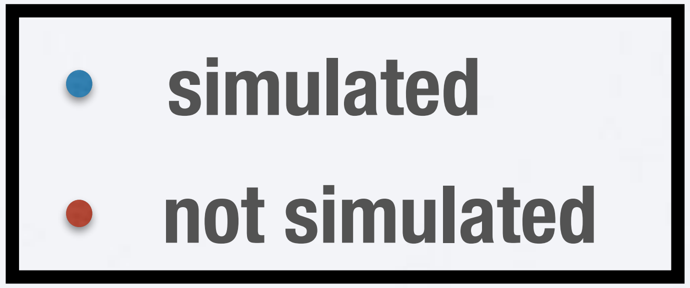
Fitness values

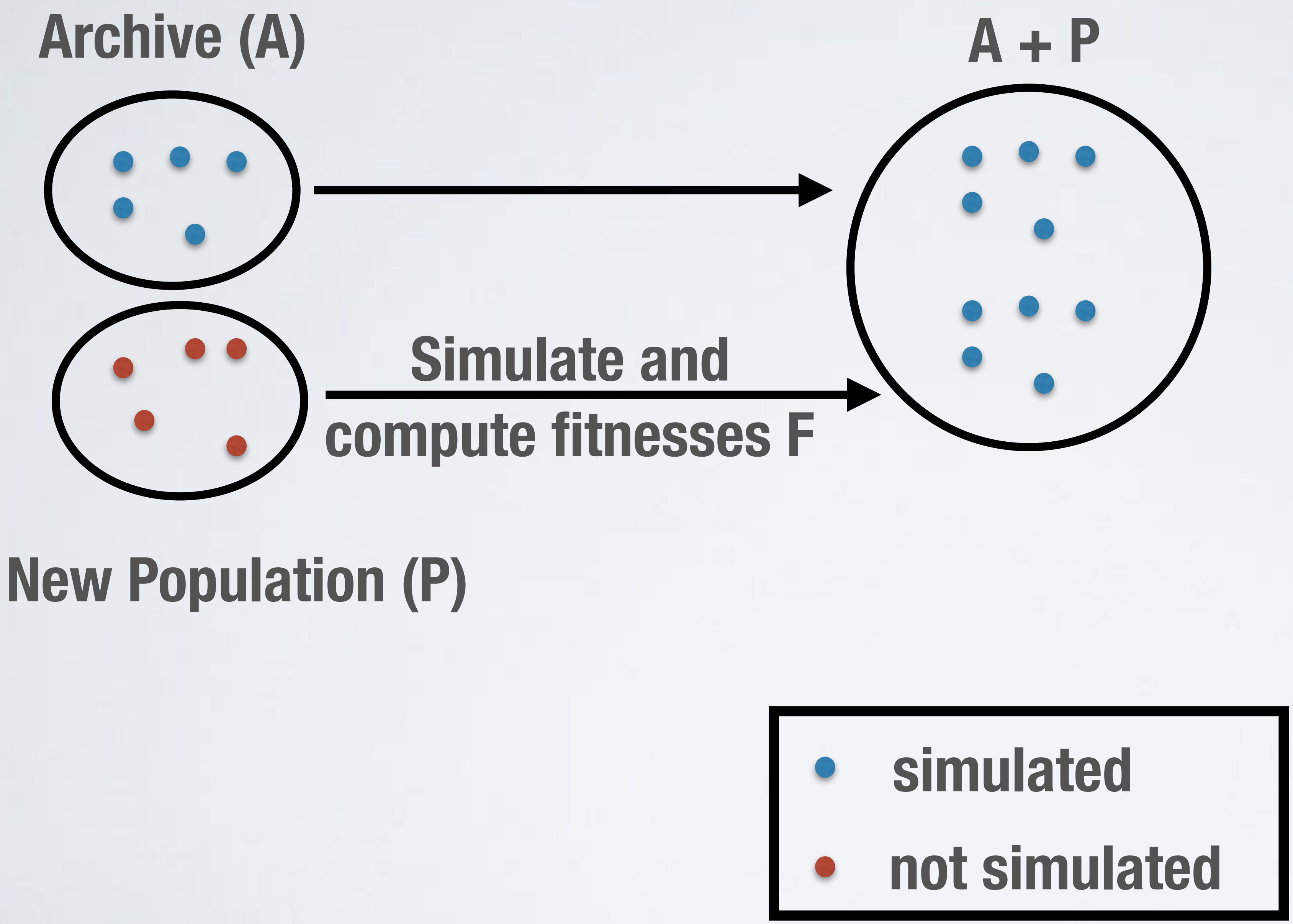
Tests revealing requirements violations

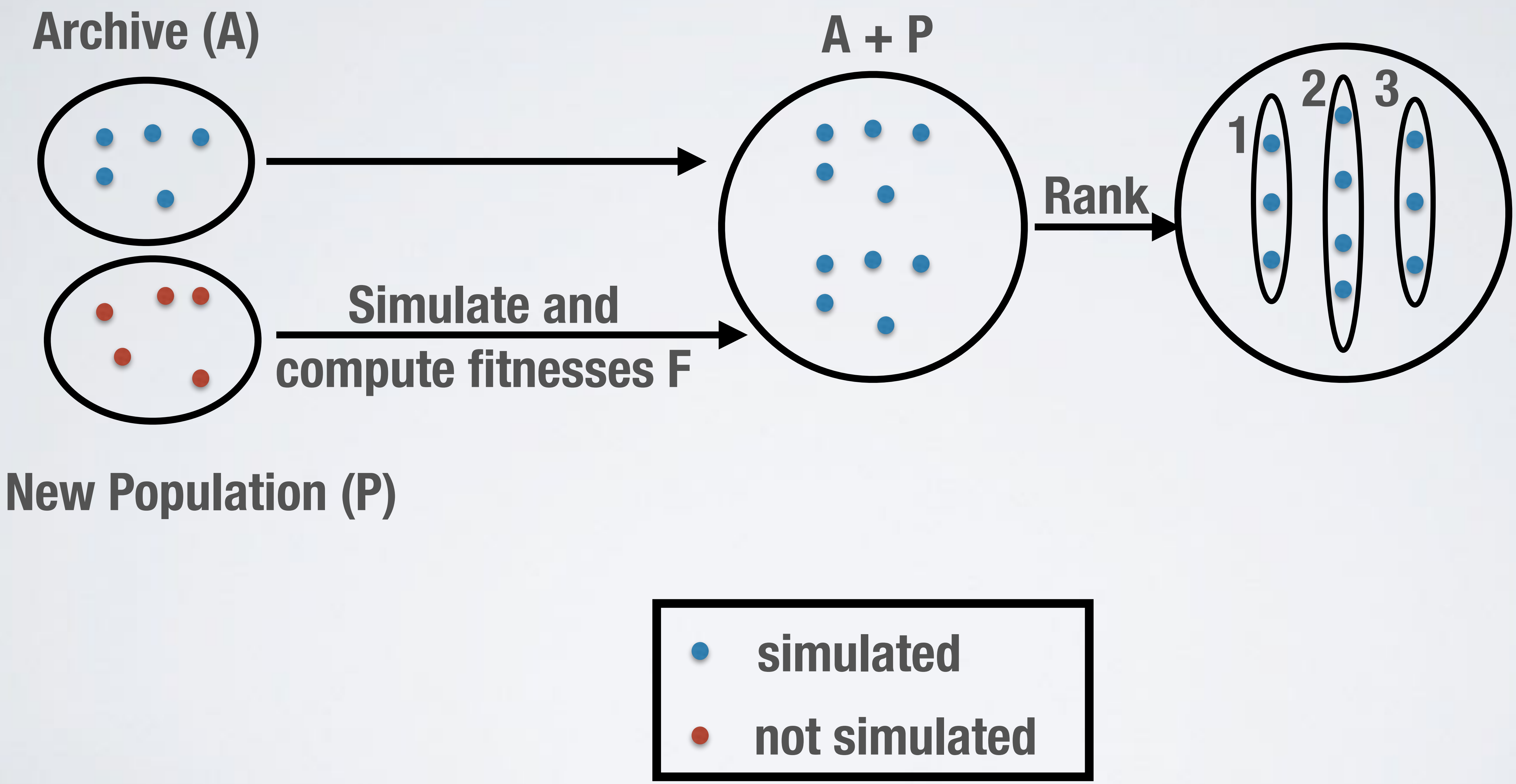
Archive (A)

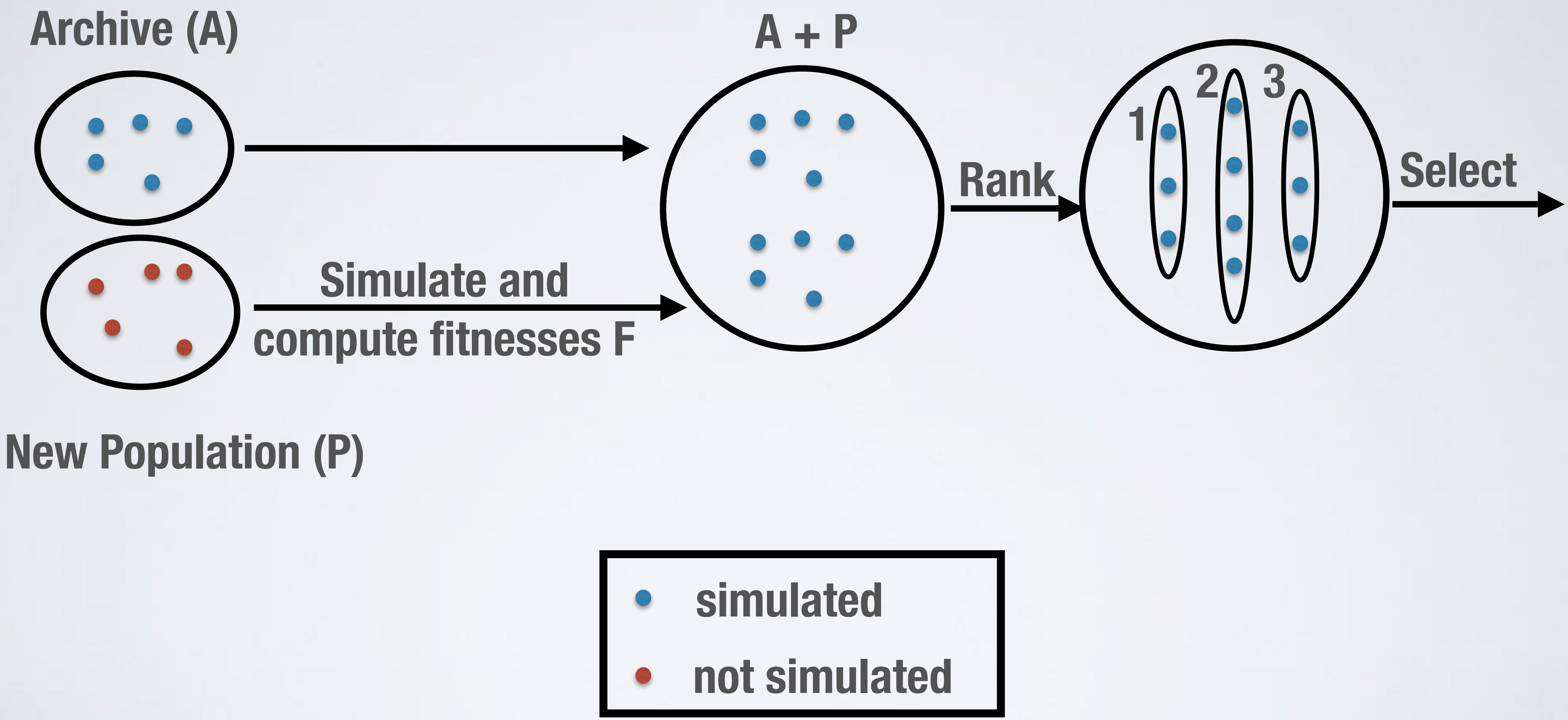


New Population (P)

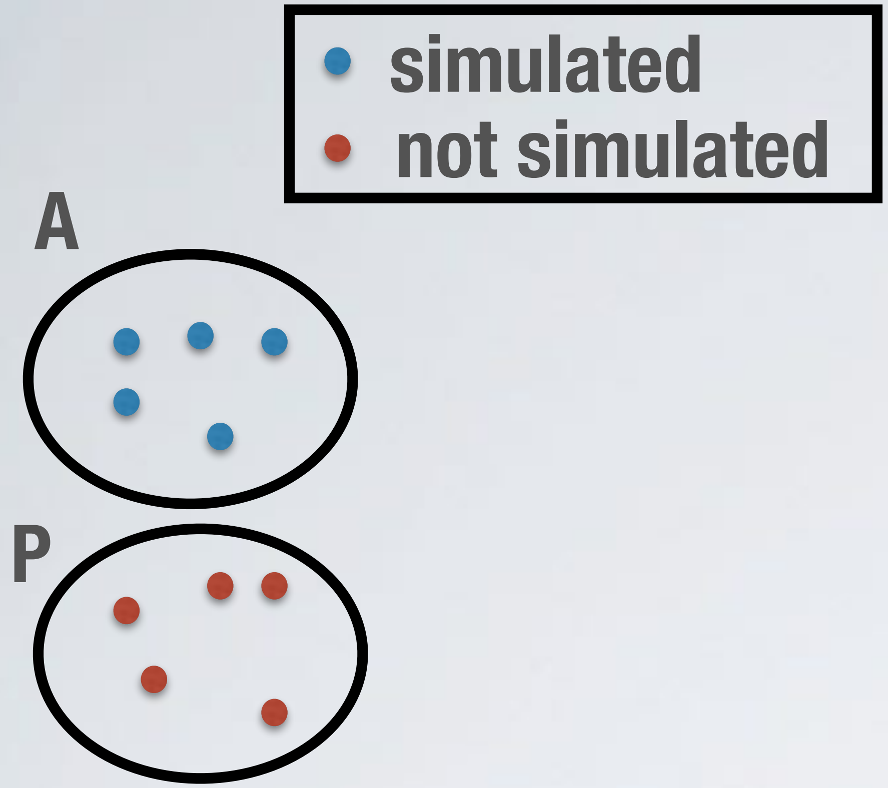


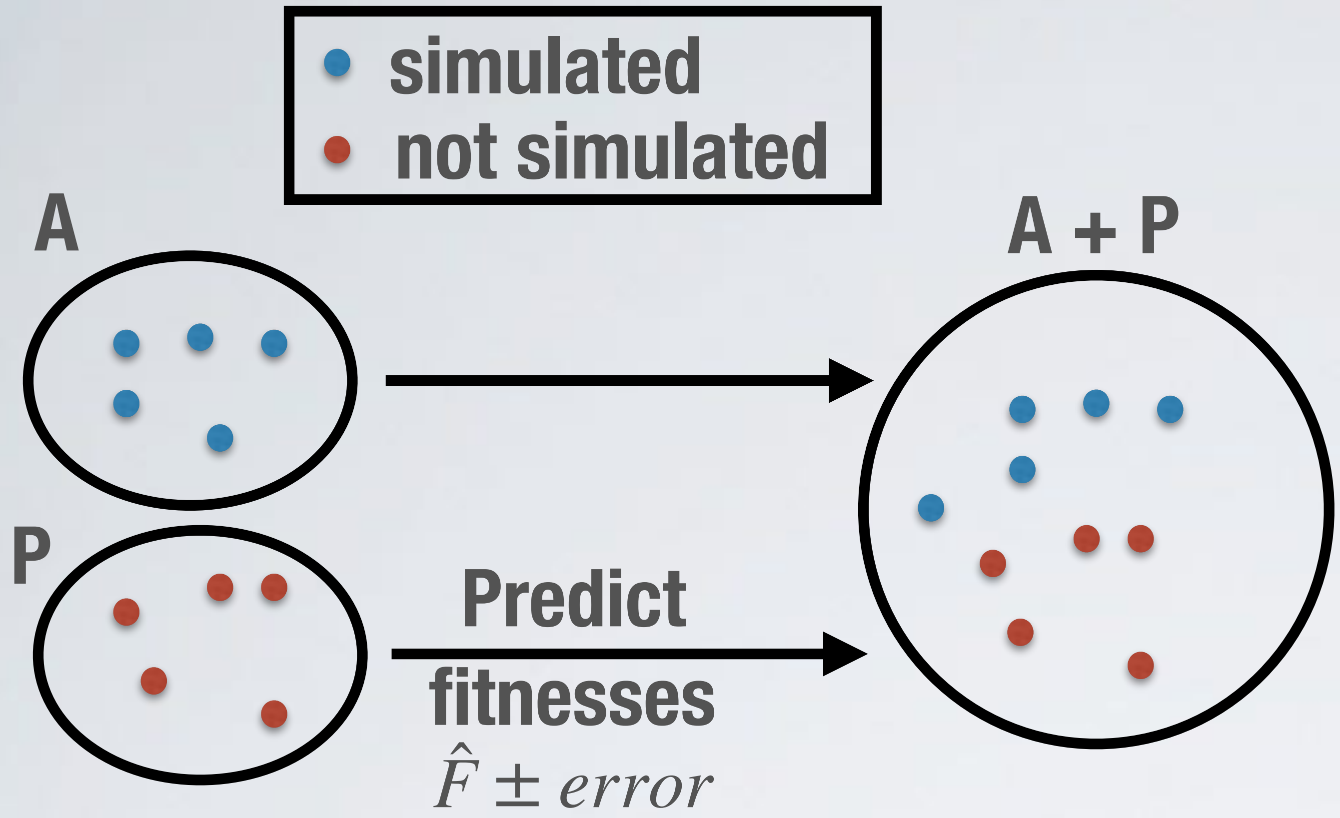

Archive (A)
A + P
Simulate and
compute fitnesses F
New Population (P)
simulated
not simulated

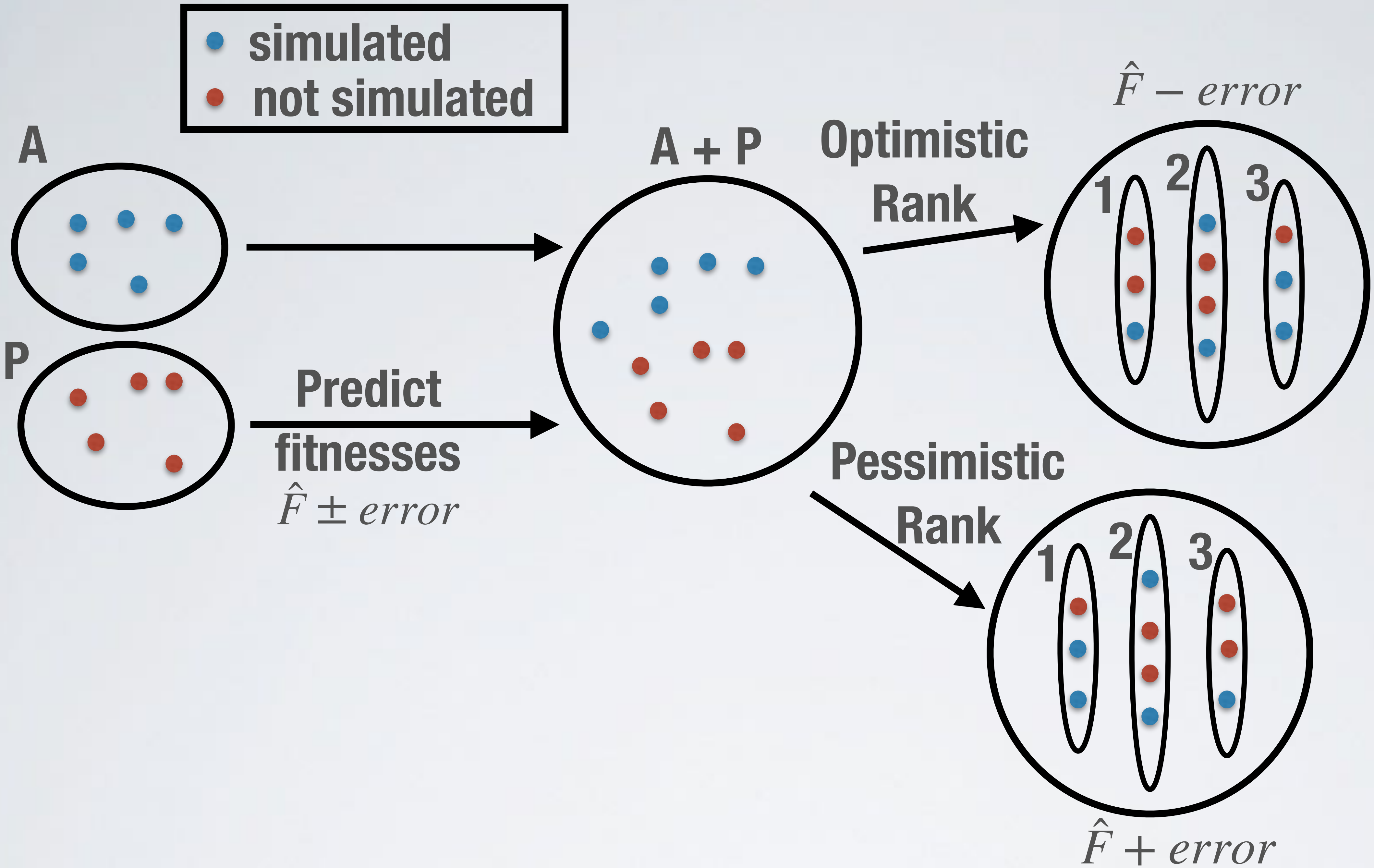

Archive (A)
A + P
1
2
3
Rank
Simulate and
compute fitnesses F
New Population (P)
simulated
not simulated



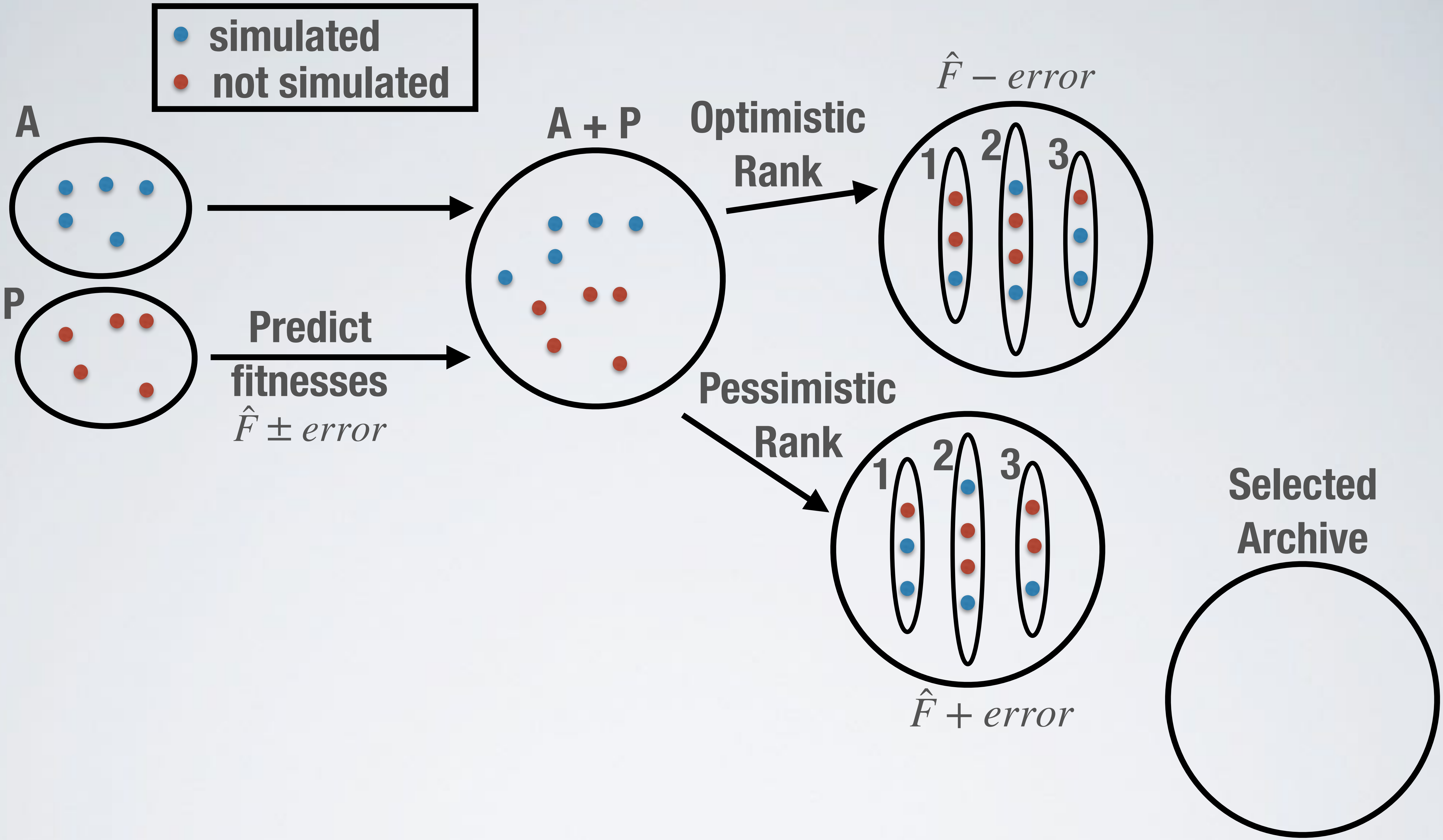
Archive (A)
A + P
1
2
3
Rank
Select
Simulate and
compute fitnesses F
New Population (P)
simulated
not simulated

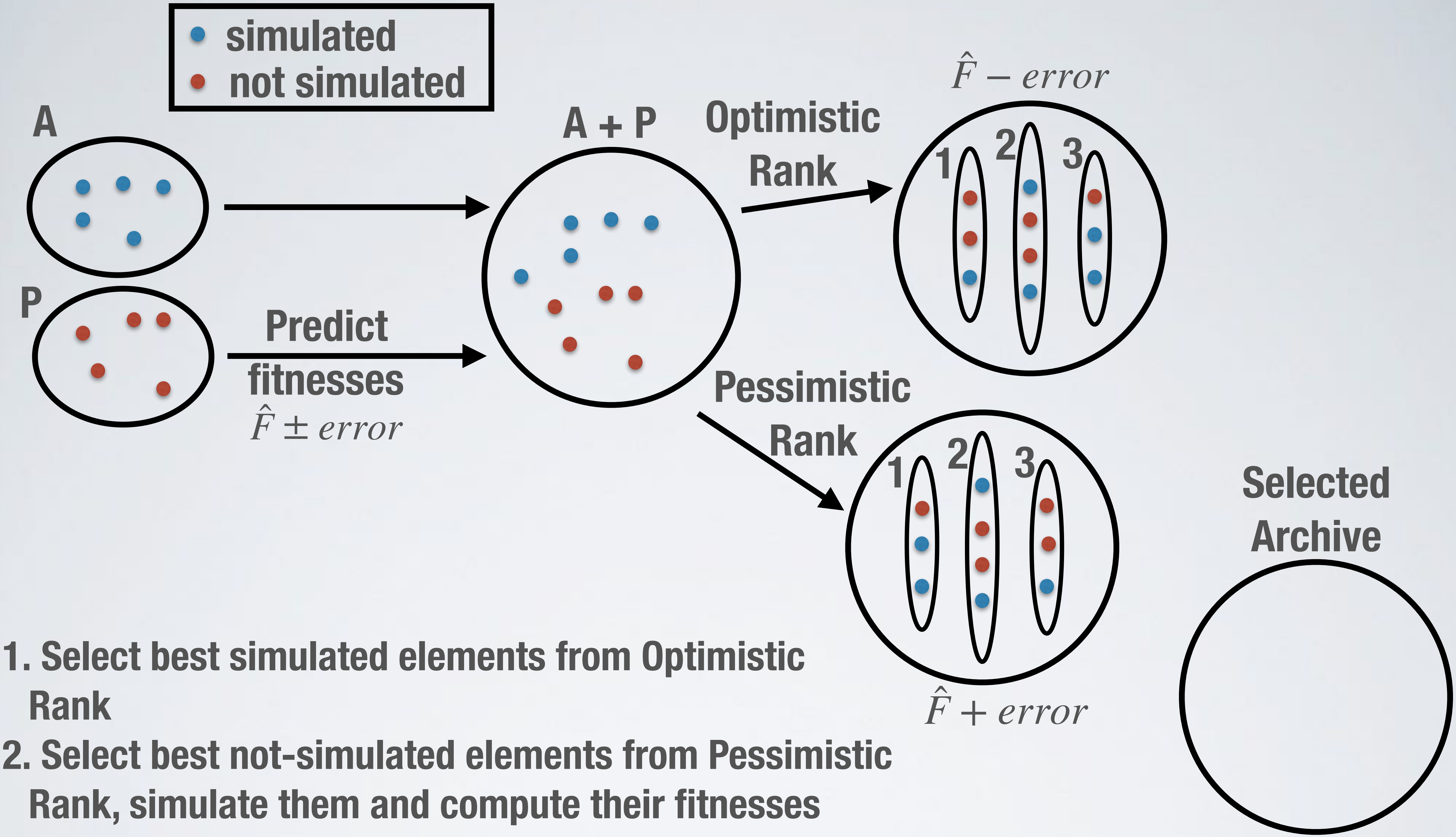

simulated
not simulated
A
P


simulated
not simulated
A
P
A + P
Predict
fitnesses
$\hat{F} \pm error$

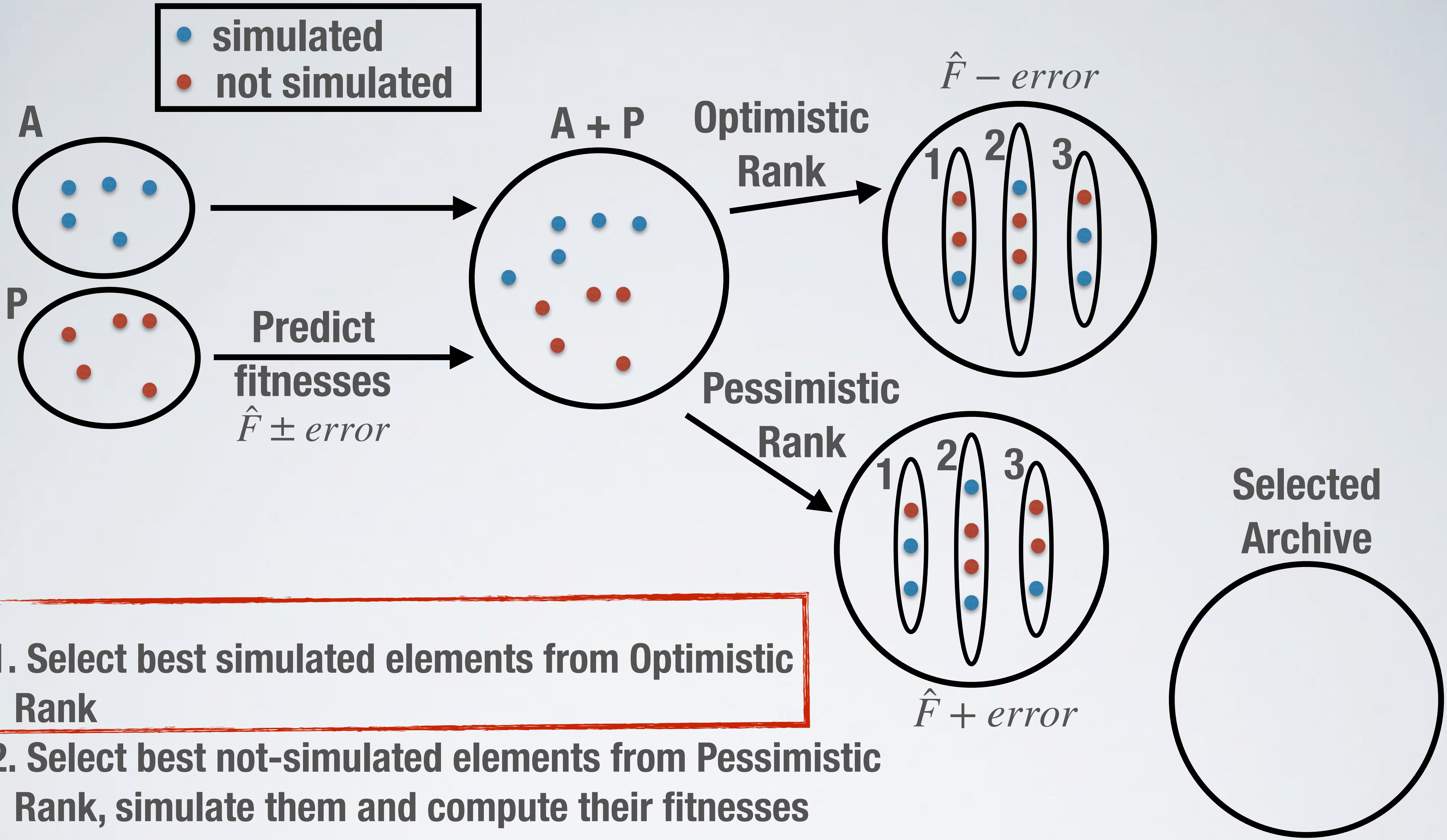


simulated
not simulated
A
P
A + P
Predict
fitnesses
$\hat{F} \pm error$
Optimistic
Rank
$\hat{F} - error$
1
2
3
Pessimistic
Rank
1
2
3
$\hat{F} + error$

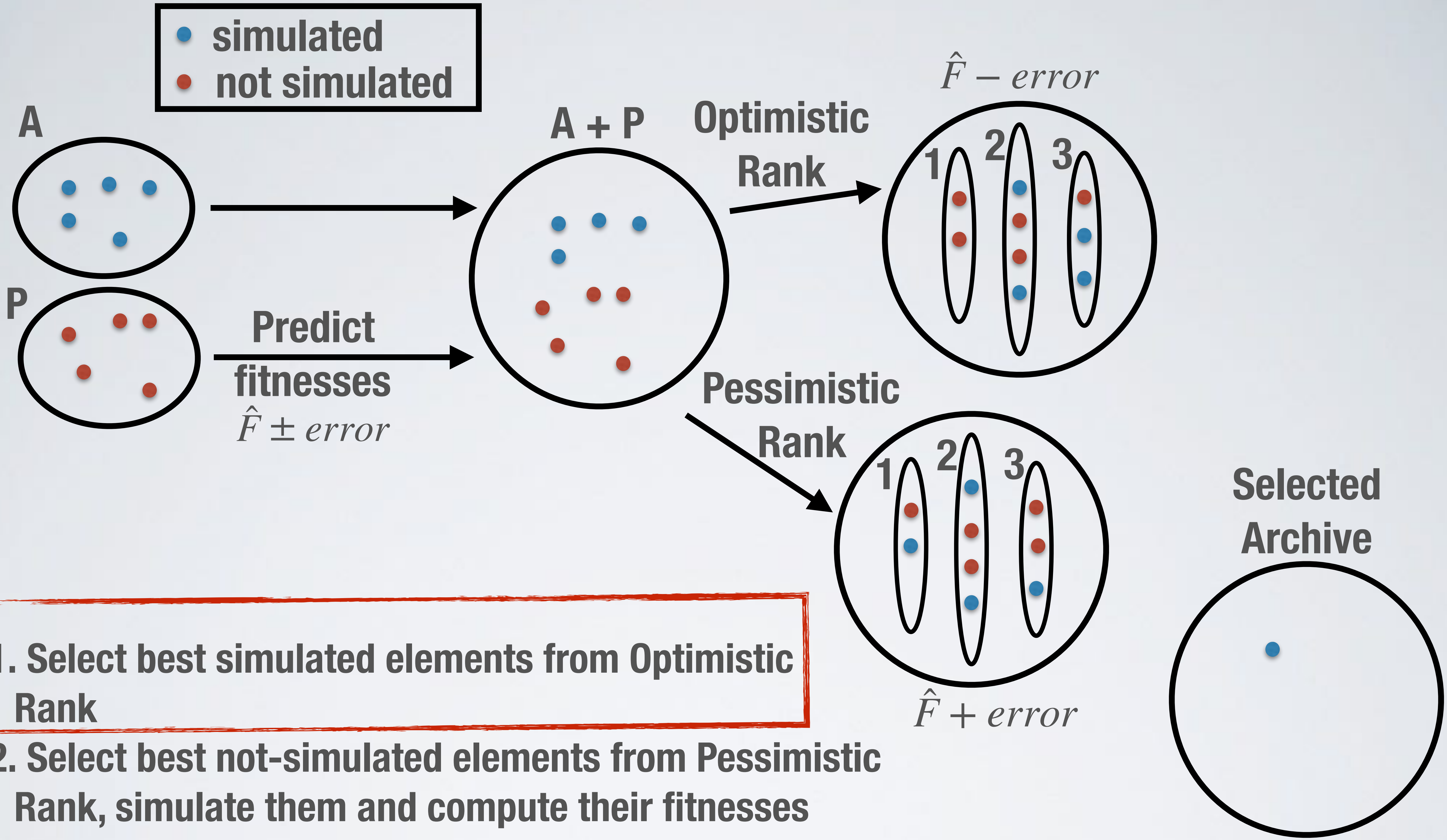

simulated
not simulated
A
P
A + P
Predict
fitnesses
$\hat{F} \pm error$
Optimistic
Rank
$\hat{F} - error$
1
2
3
Pessimistic
Rank
1
2
3
$\hat{F} + error$
Selected
Archive



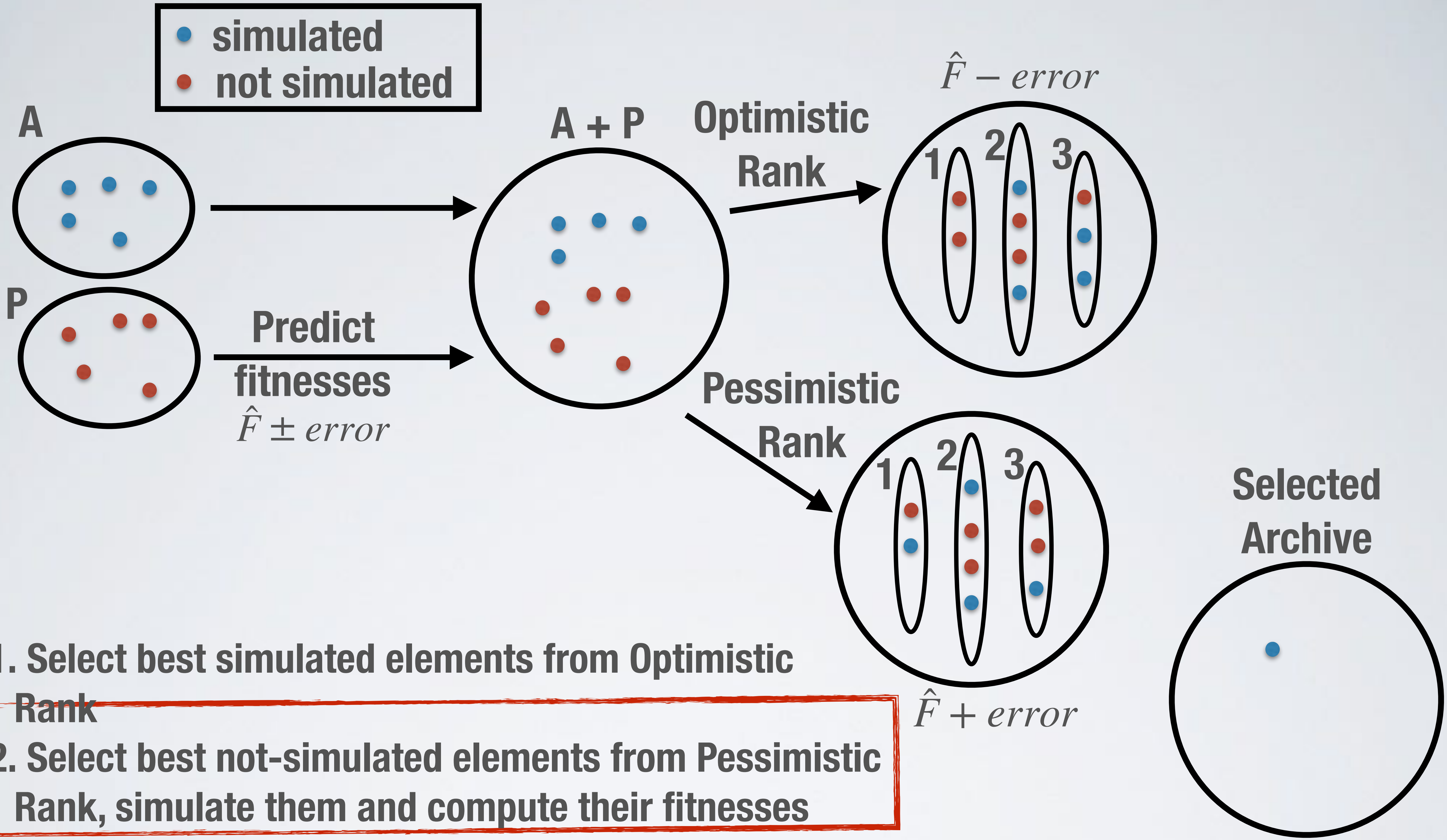
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



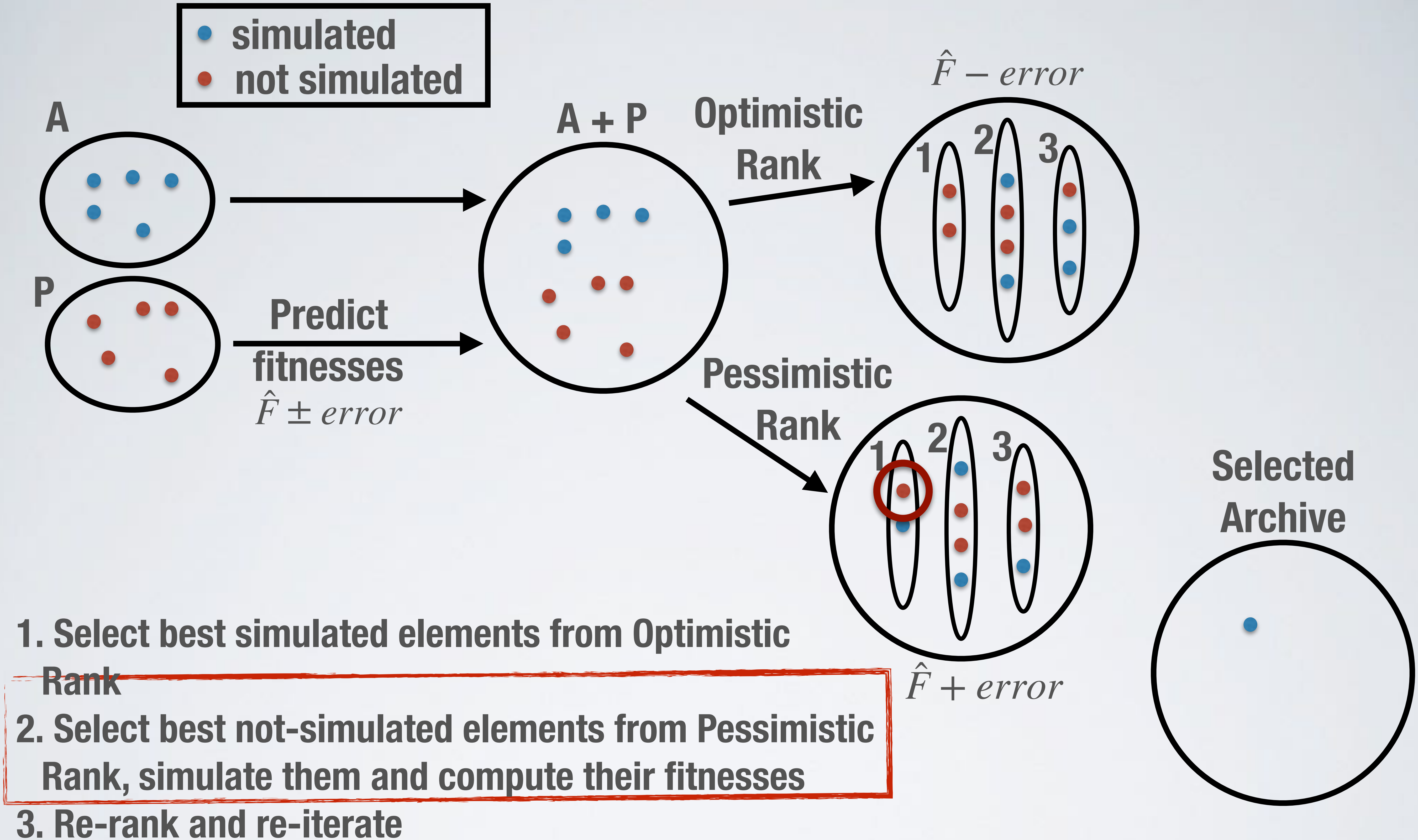
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



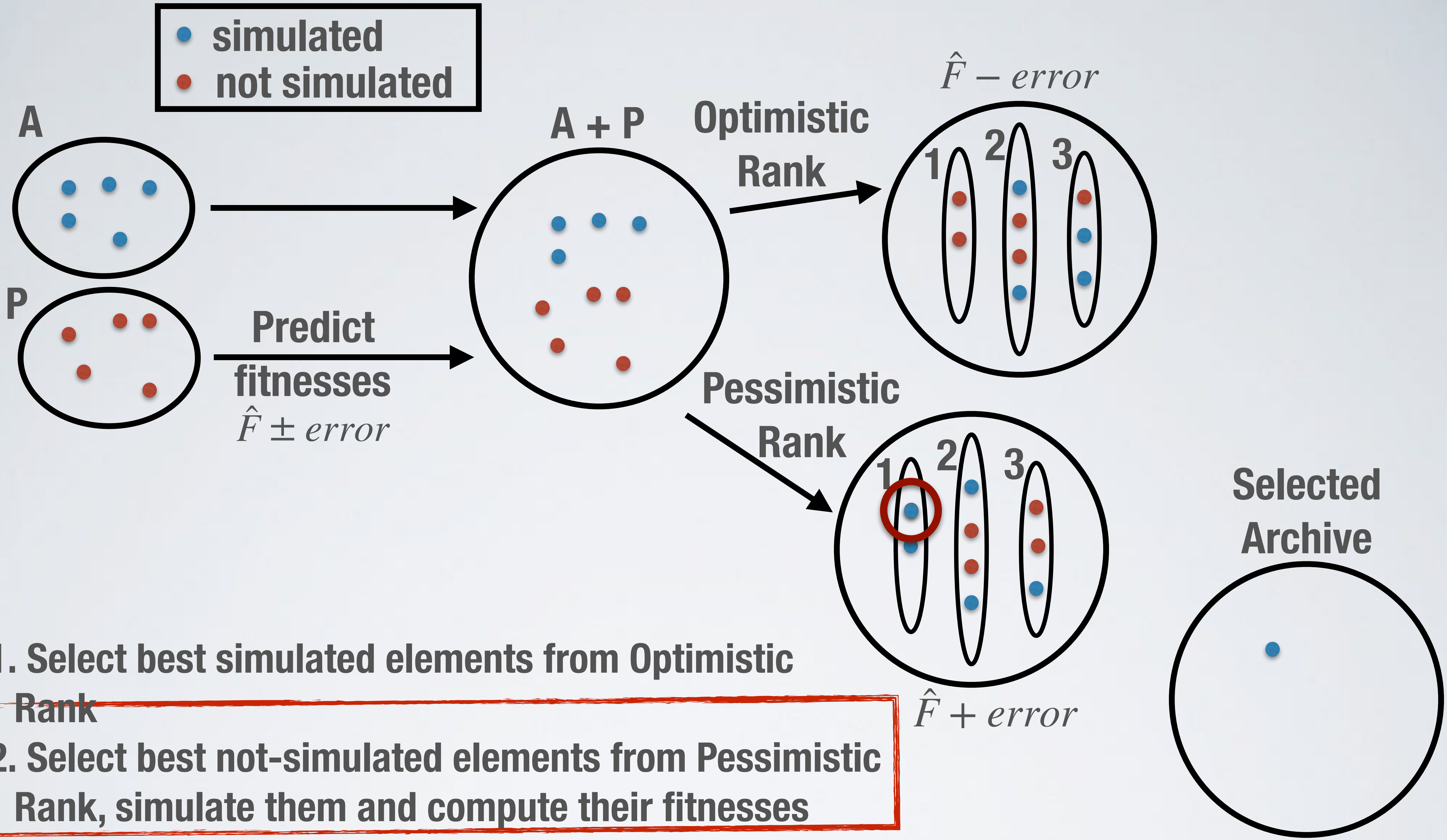
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



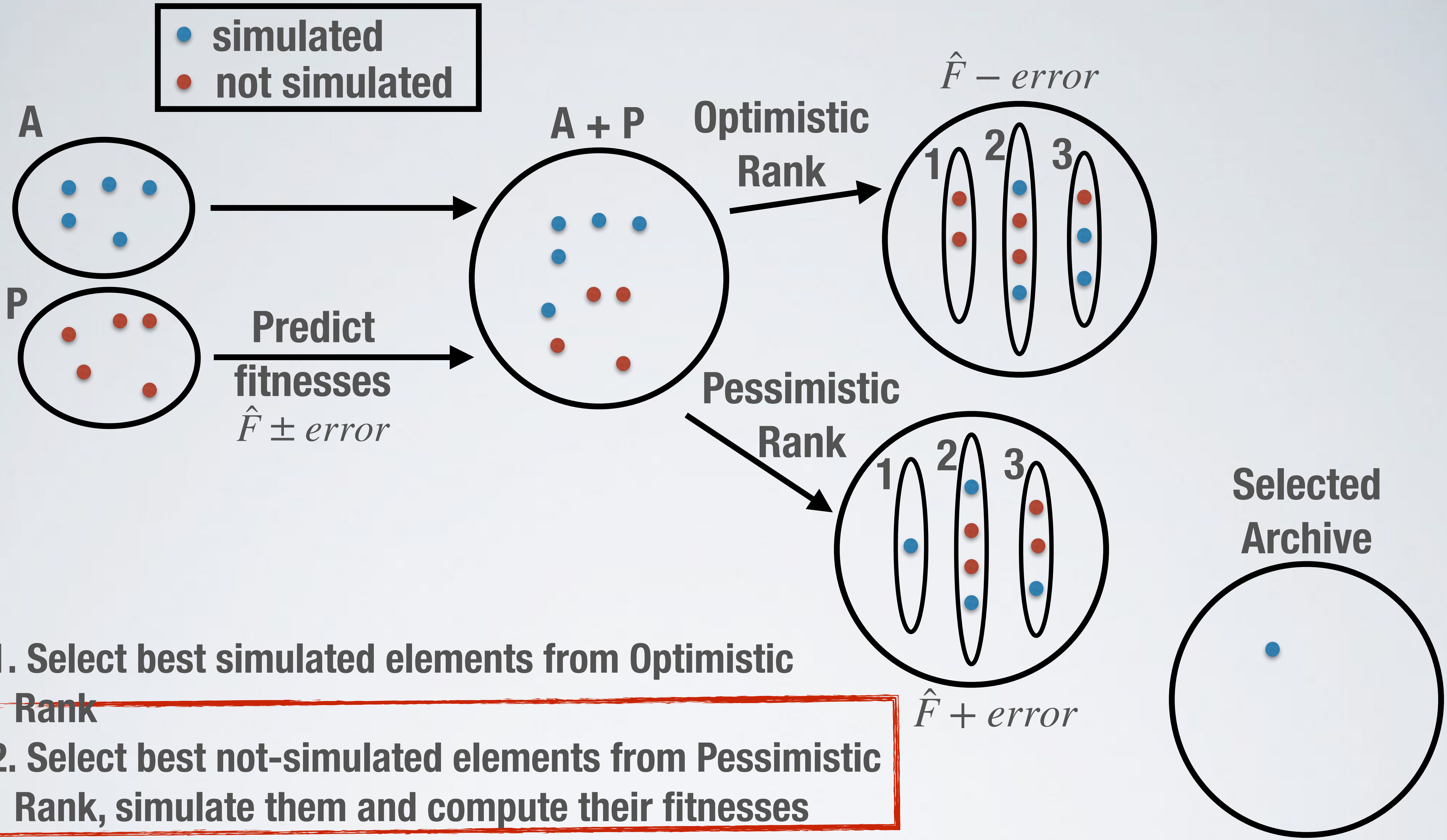
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



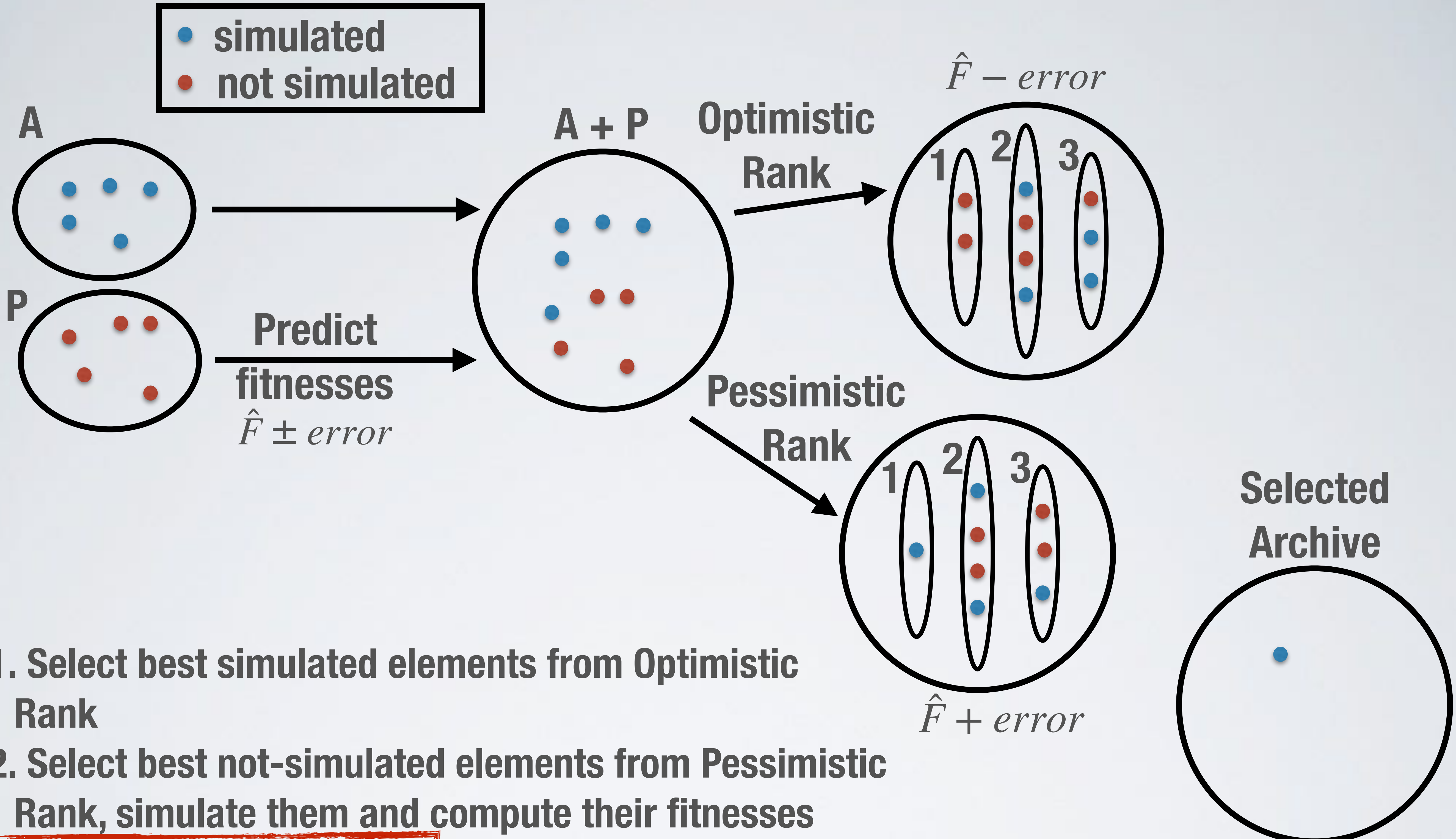
simulated
not simulated
A
P
A + P
Predict
fitnesses
$\hat{F} \pm error$
Optimistic
Rank
$\hat{F} - error$
1
2
3
Pessimistic
Rank
1
2
3
$\hat{F} + error$
Selected
Archive
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



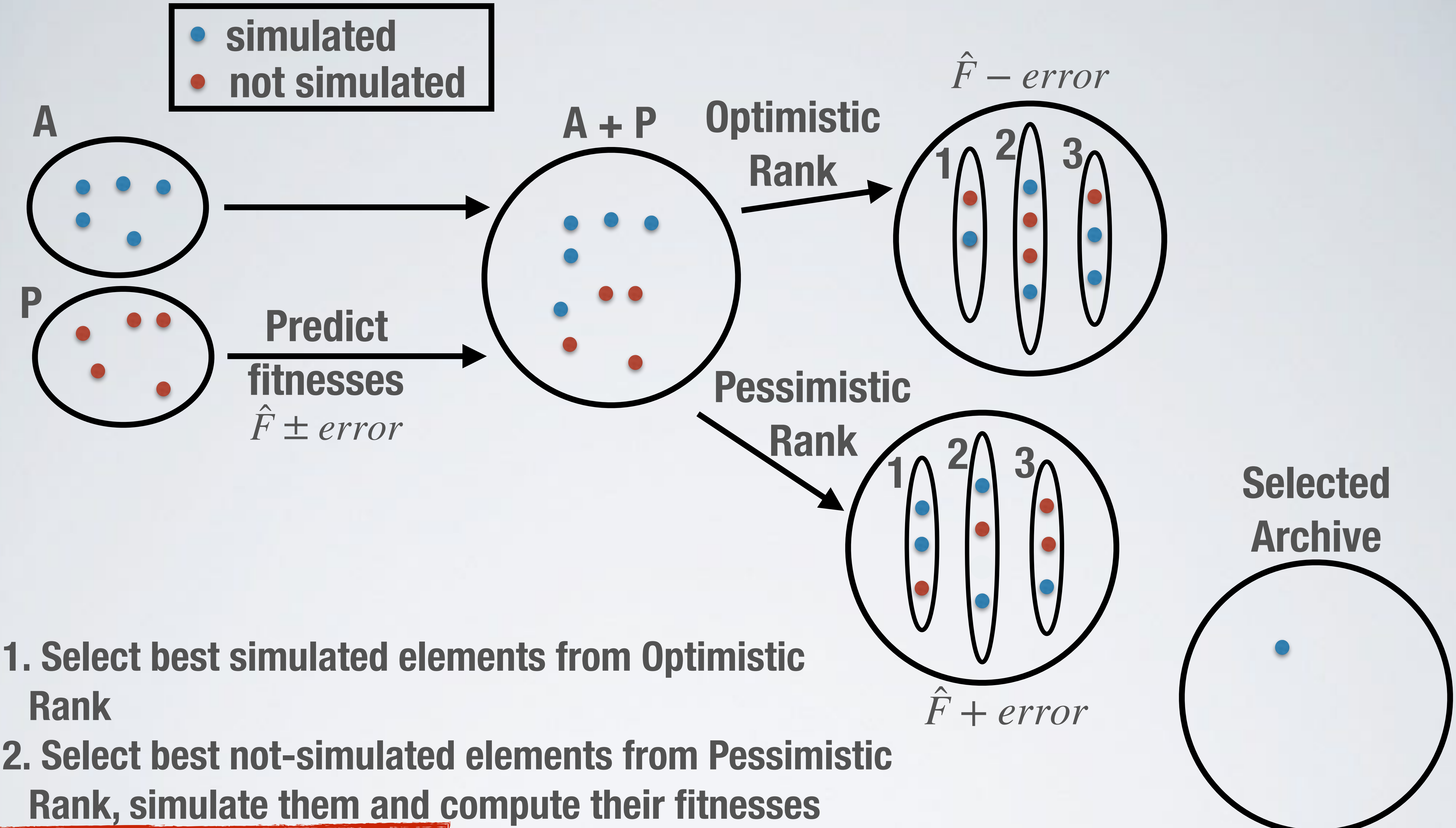
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



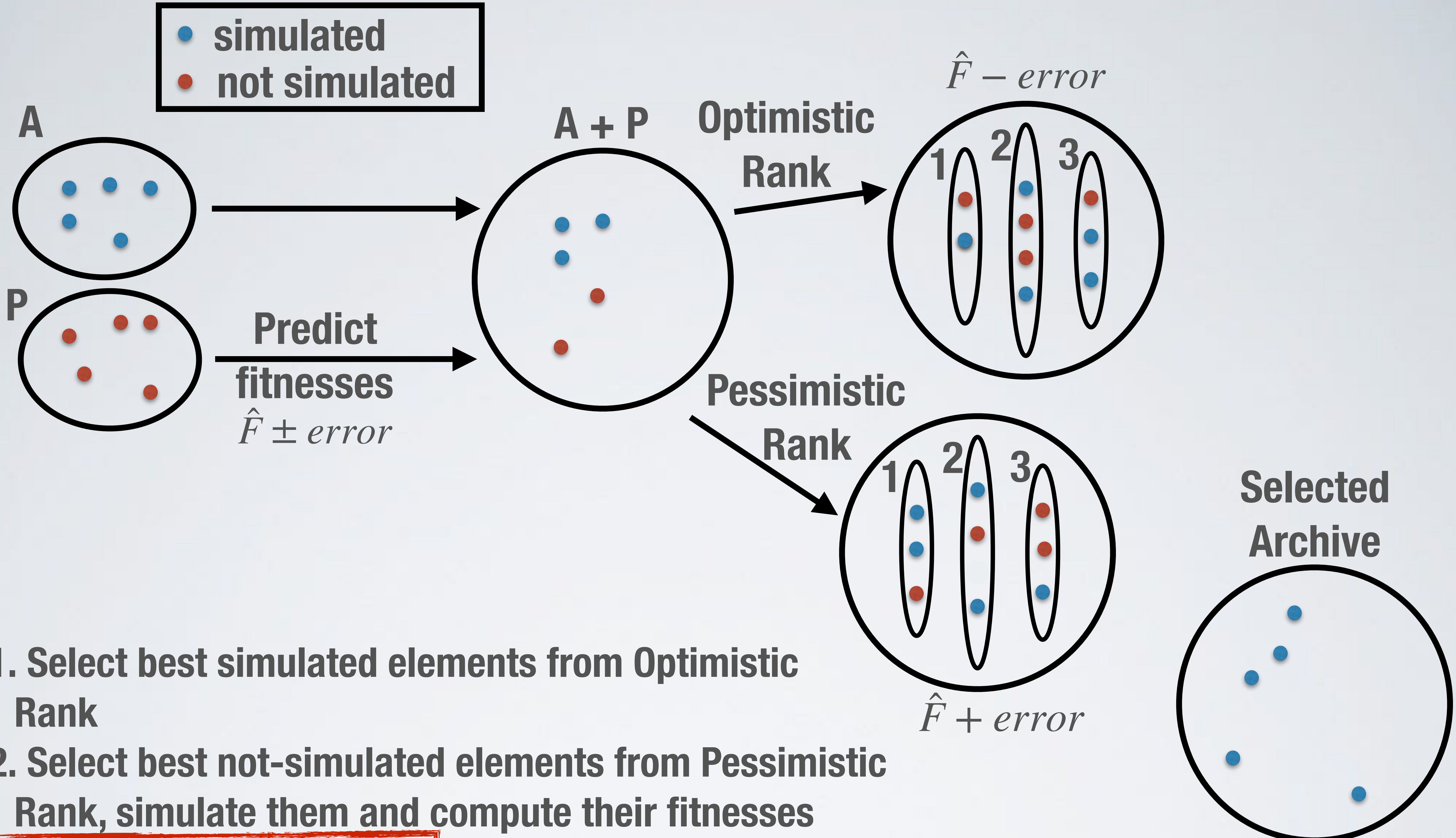
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



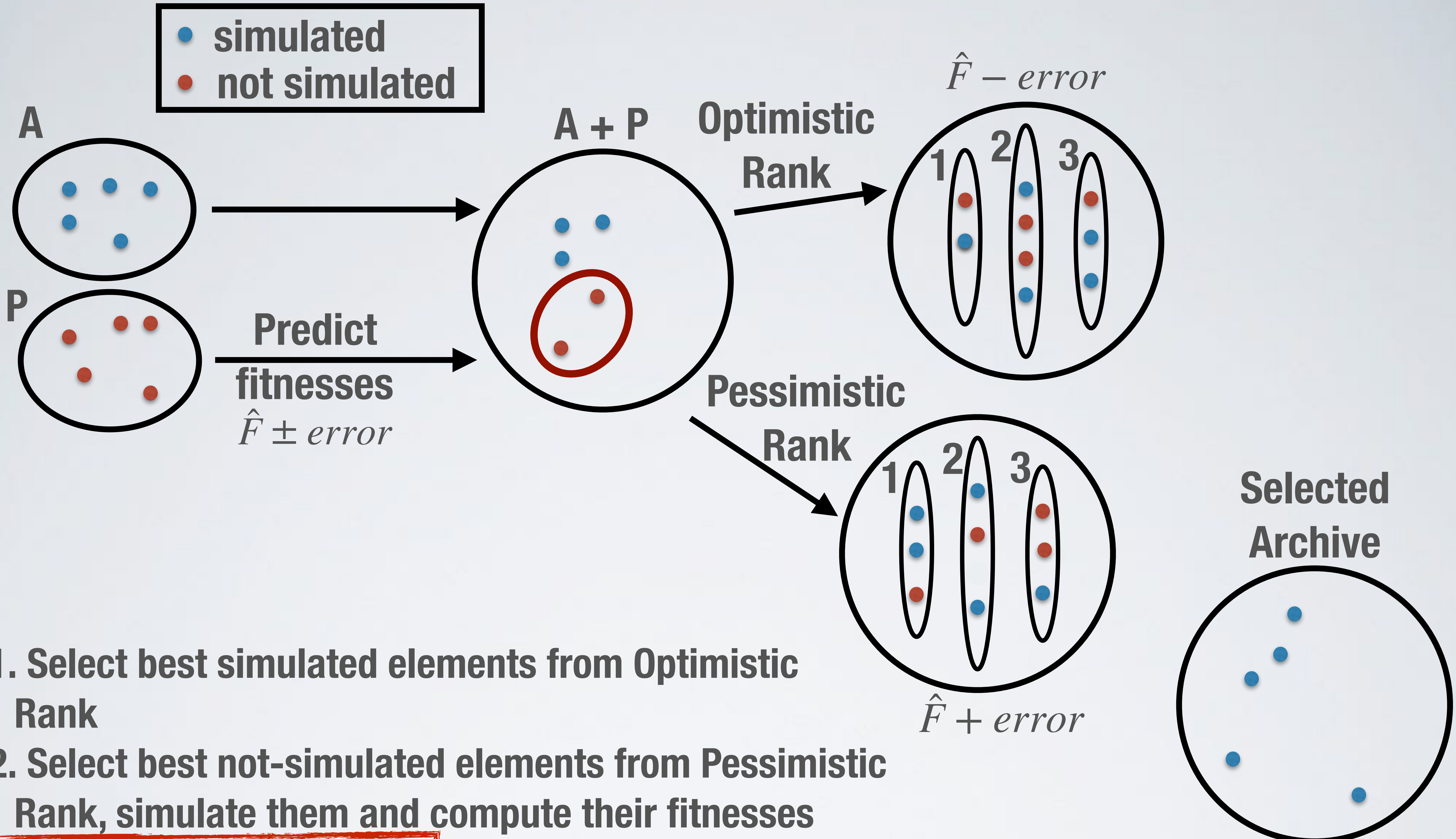
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



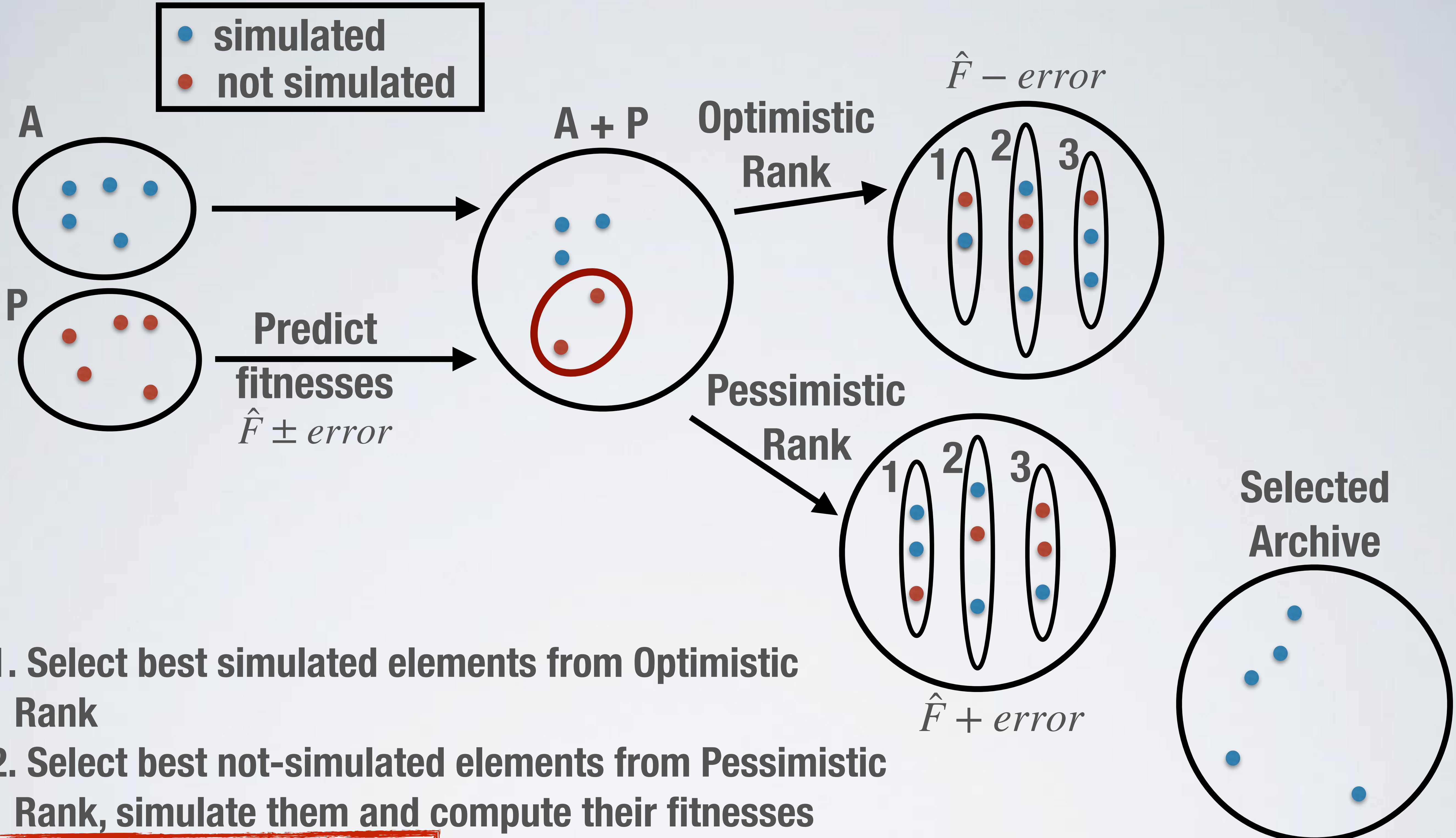
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



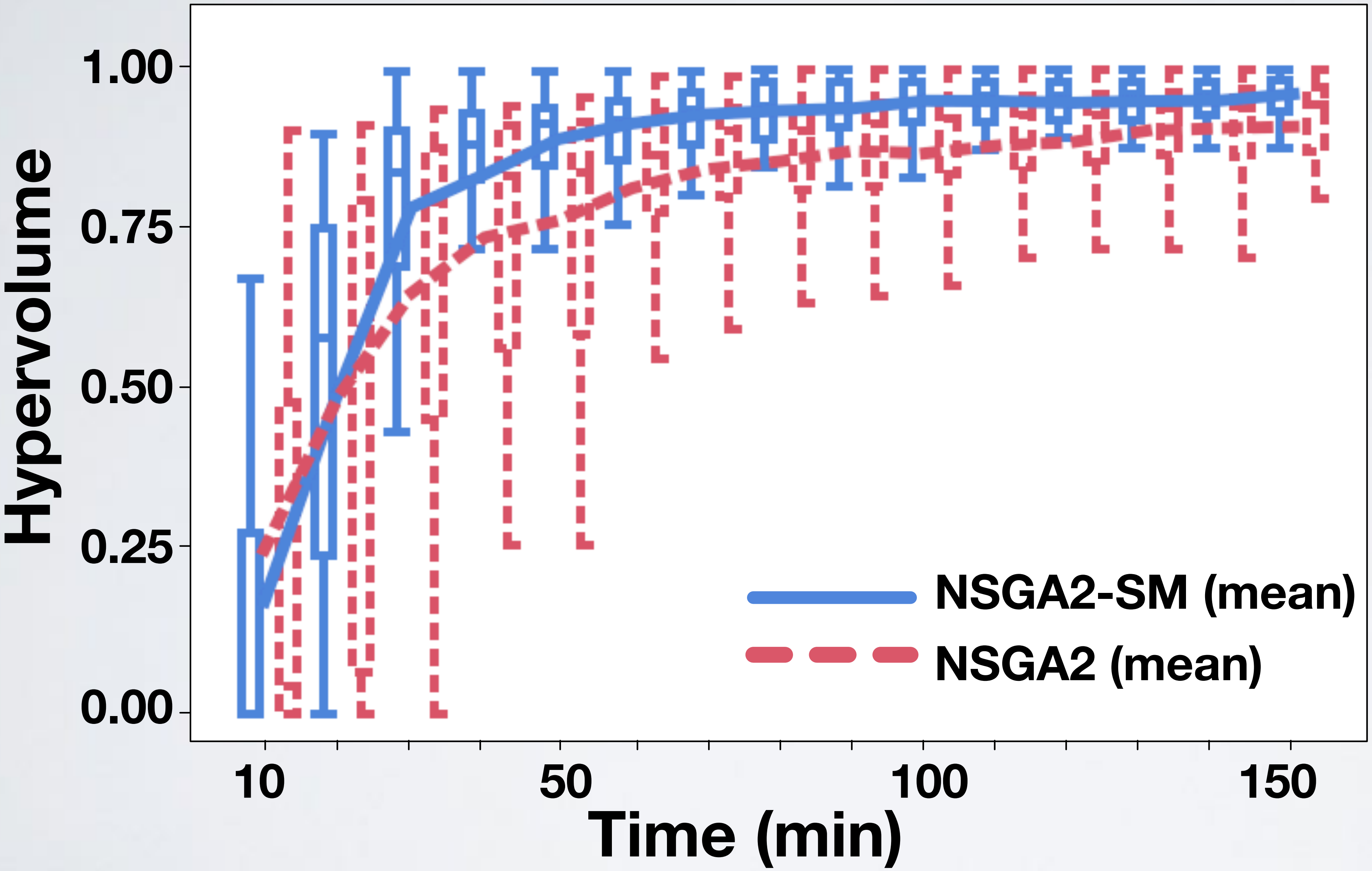
1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate



1. Select best simulated elements from Optimistic Rank
2. Select best not-simulated elements from Pessimistic Rank, simulate them and compute their fitnesses
3. Re-rank and re-iterate

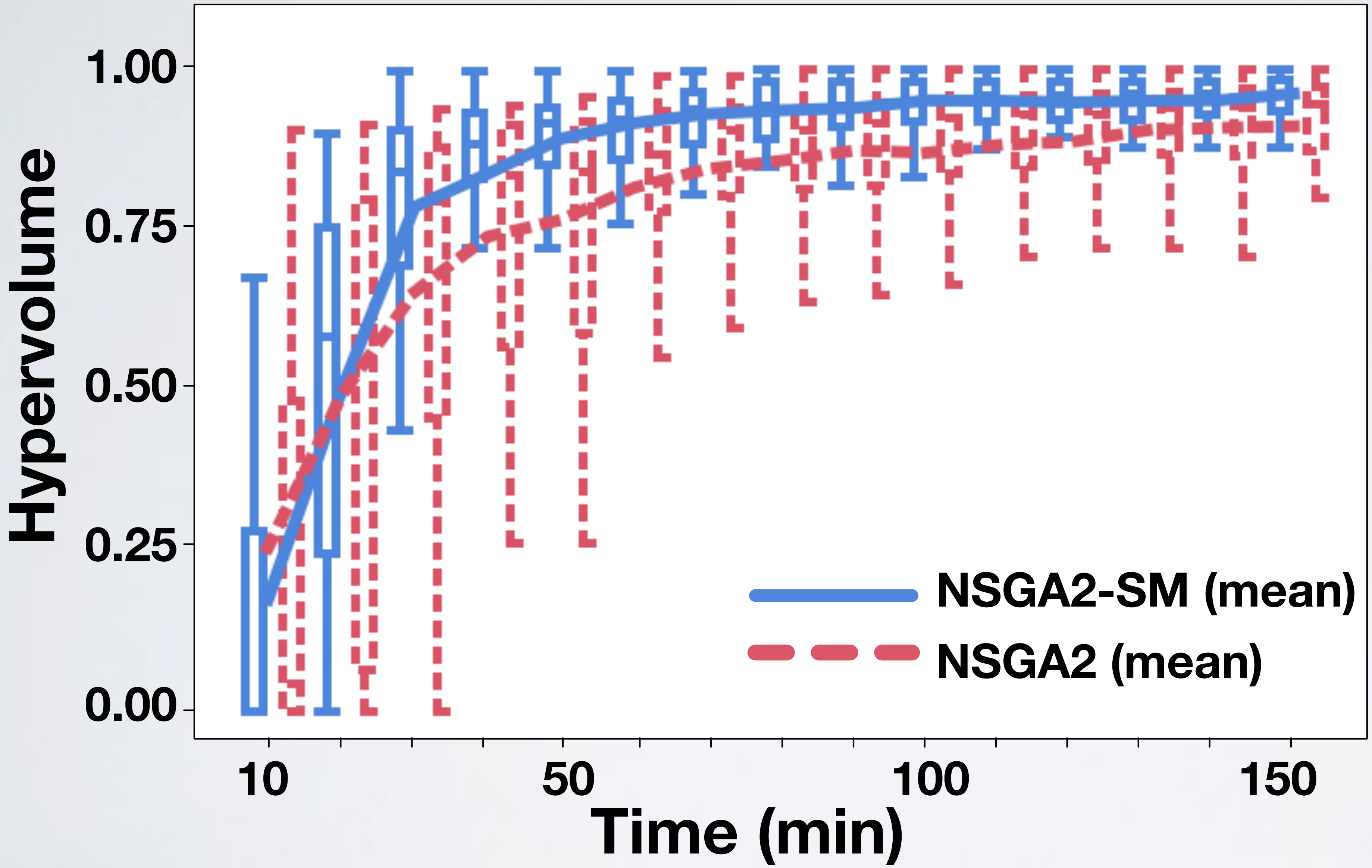
Predicted values are only used to bypass simulations for unfit individuals

Comparing Search w/ and w/o Surrogate



Search with surrogate models generates higher quality solutions than search without surrogate models

Comparing Search w/ and w/o Surrogate

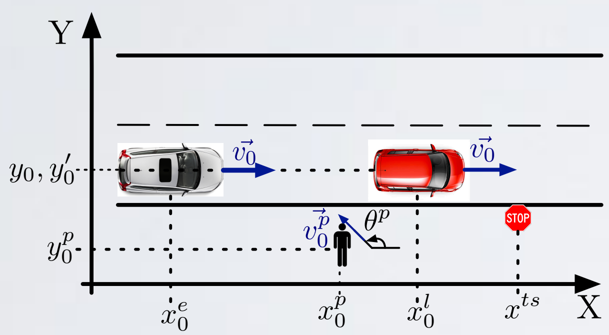


Search with surrogate models generates higher quality solutions than search without surrogate models

A worst case scenario example

Guided Test Generation

Test Input Characterisation



Test input generation

- Select best tests
- Generate new tests (Genetic Operators)

Evaluating test inputs

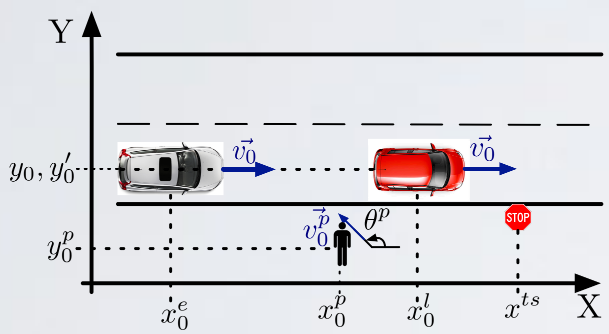
- Simulate every (candidate) test
- Compute fitness values

Fitnesses

Tests revealing requirements violations

Test Generation Guided by Classification

Test Input Characterisation



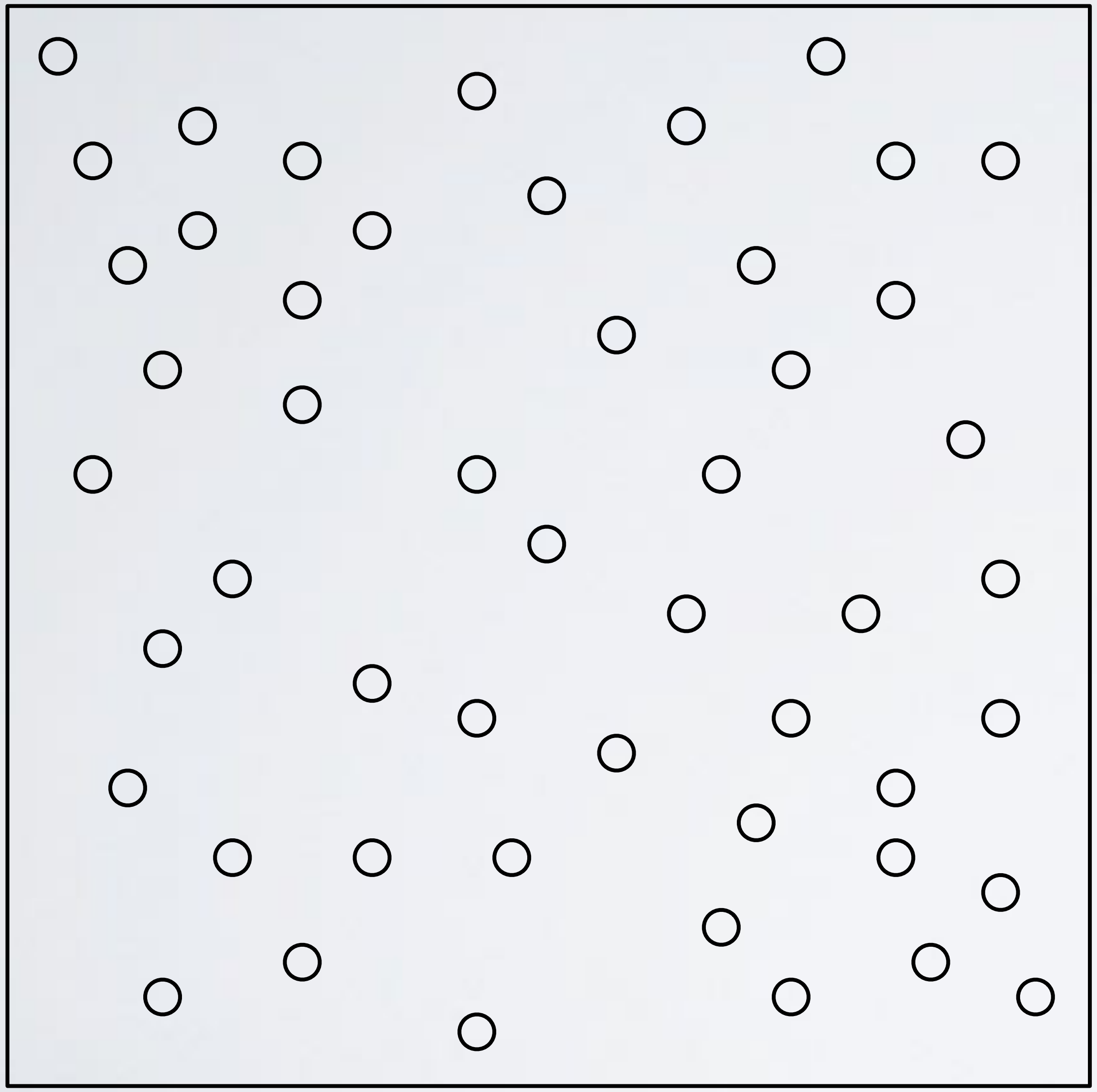
- **Build a classification tree**
- **Select/generate tests in the fittest regions**

- Simulate every (candidate) test
- Compute **fitness values**

Fitnesses

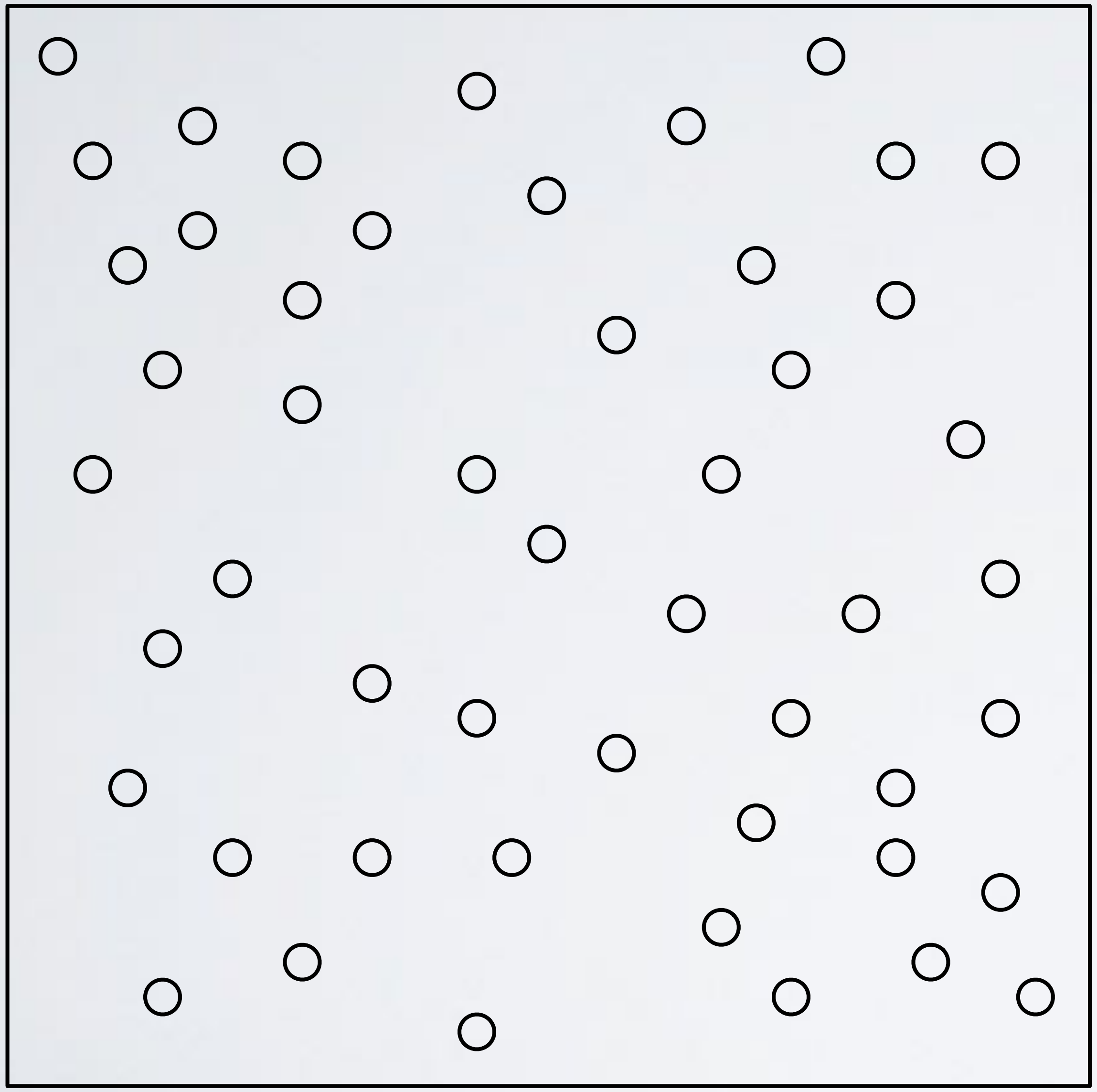
Tests revealing requirements violations **+**
Failure Explanations

Genetic Evolution Guided by Classification



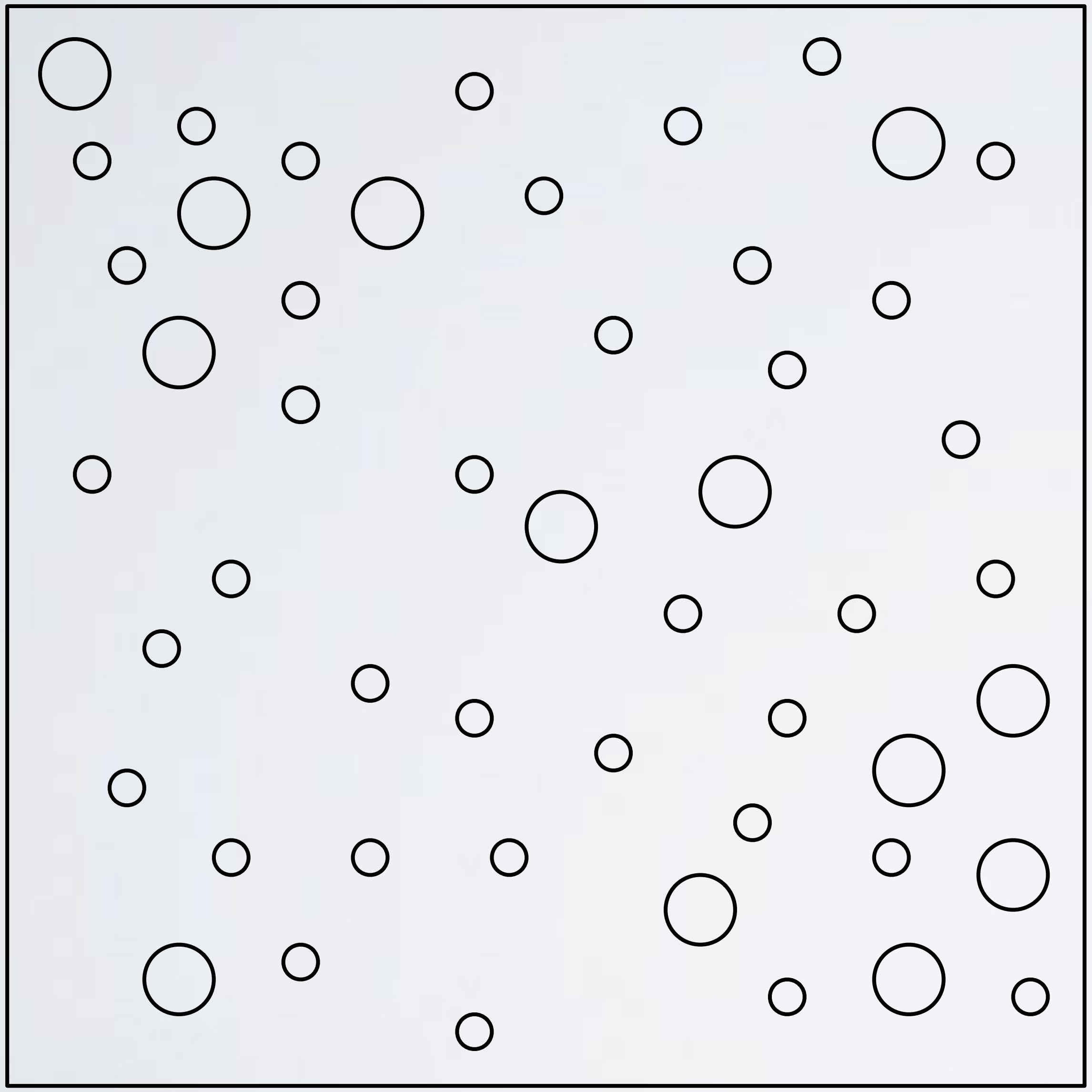
1. Initial Inputs
2. Fitness Computation
3. **Classification**
4. Selection
5. Breeding

Genetic Evolution Guided by Classification



1. Initial Inputs ✓
2. Fitness Computation
3. **Classification**
4. Selection
5. Breeding

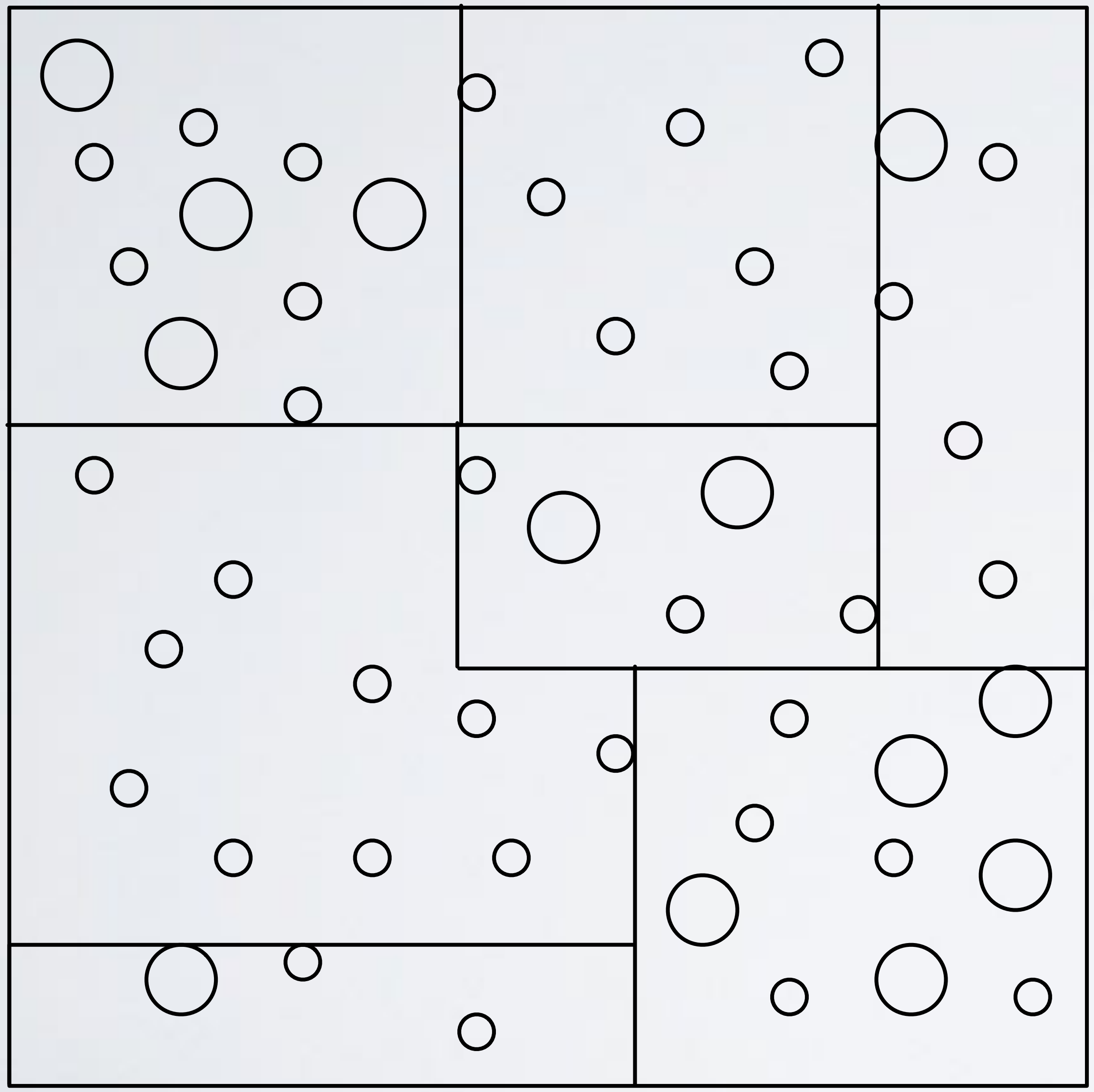
Genetic Evolution Guided by Classification



1. Initial Inputs ✓
2. Fitness Computation ✓
3. **Classification**
4. Selection
5. Breeding

Fitnesses:
F1. Min distance between pedestrian and the car
F2. Speed of the car at the time of collision

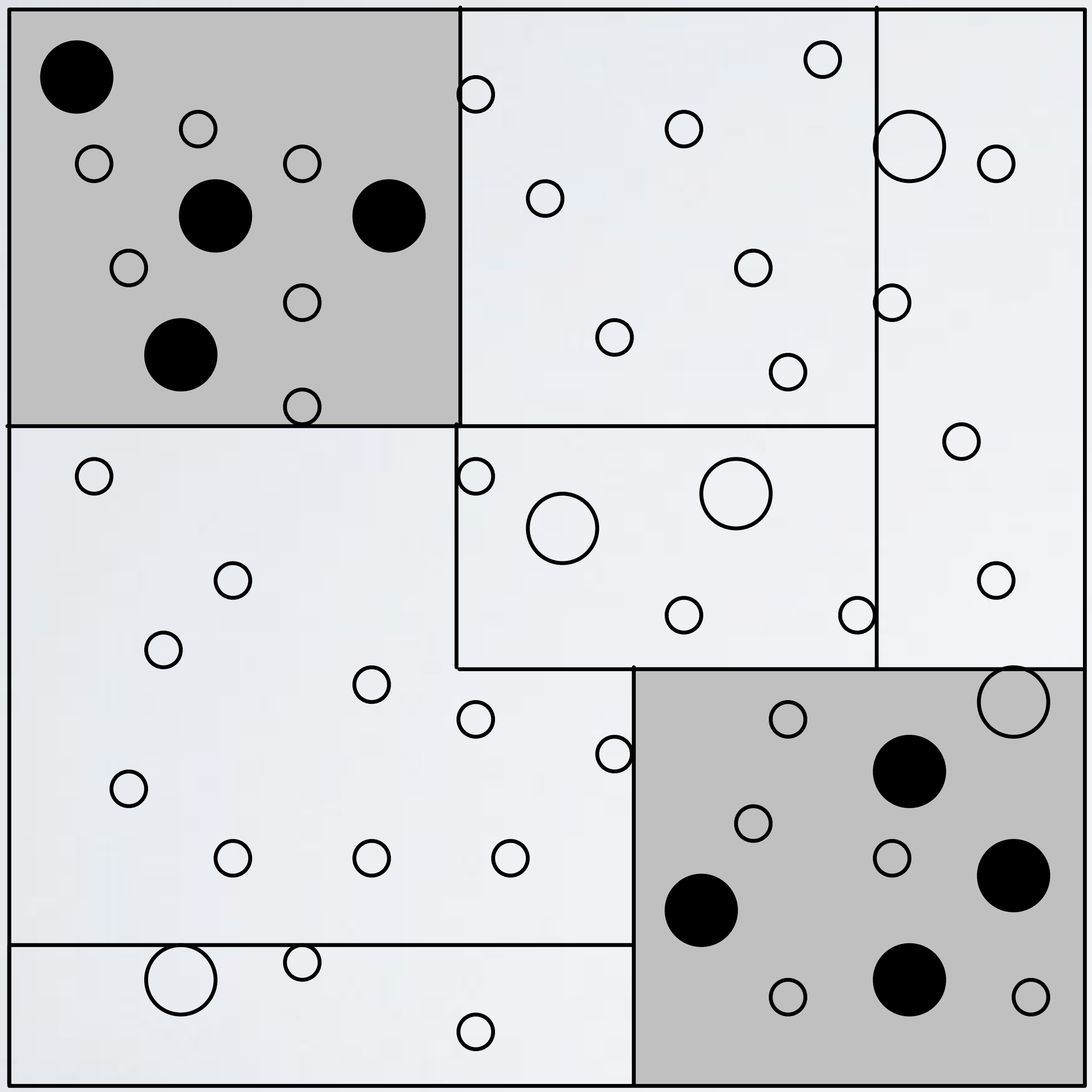
Genetic Evolution Guided by Classification



1. Initial Inputs ✓
2. Fitness Computation ✓
3. **Classification** ✓
4. Selection
5. Breeding

Label:
(F1 < threshold1) $\wedge$ (F2 > threshold2)

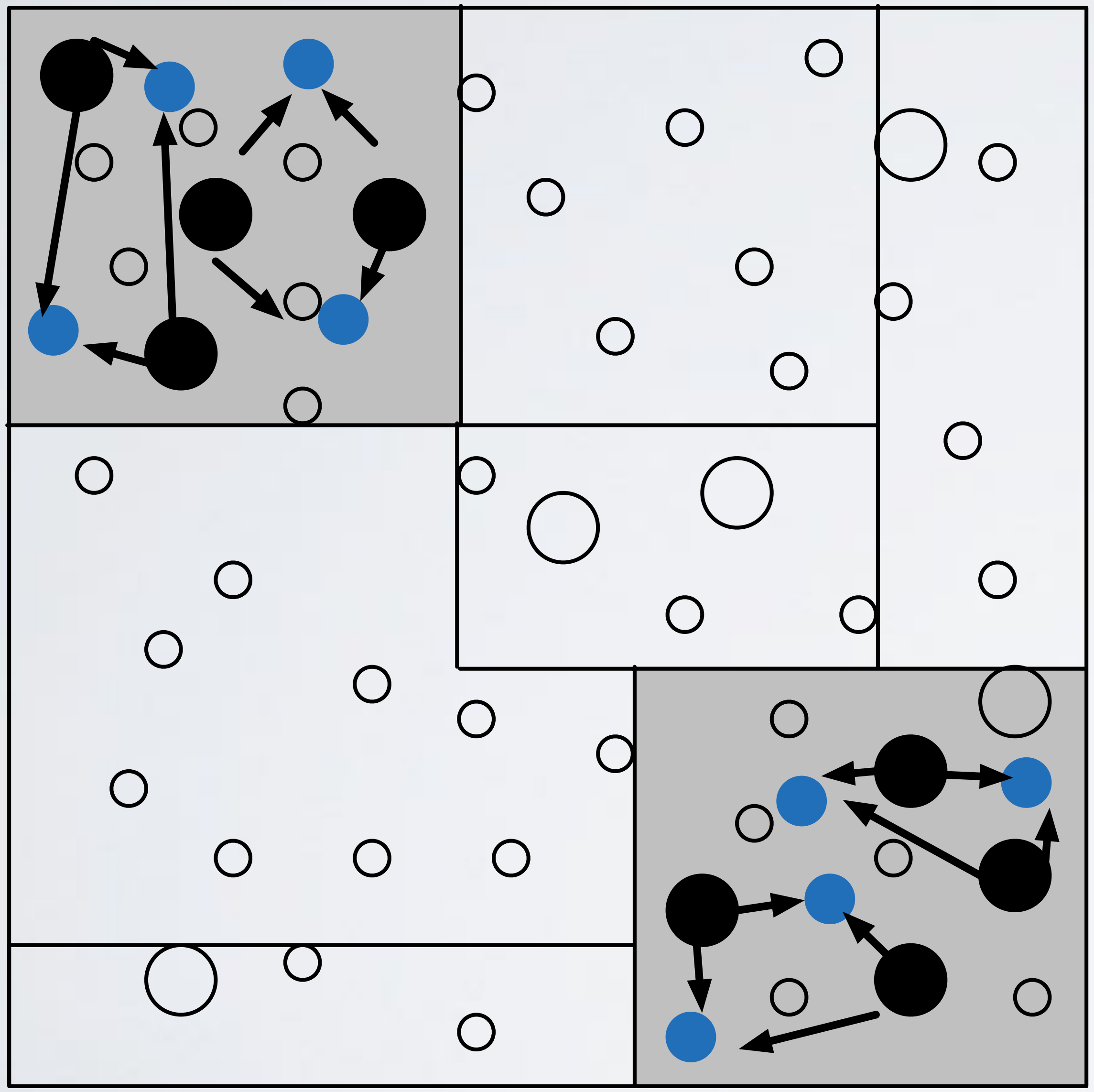
Genetic Evolution Guided by Classification



1. Initial Inputs ✓
2. Fitness Computation ✓
3. **Classification** ✓
4. Selection ✓
5. Breeding

Label:
(F1 < threshold1) $\wedge$ (F2 > threshold2)

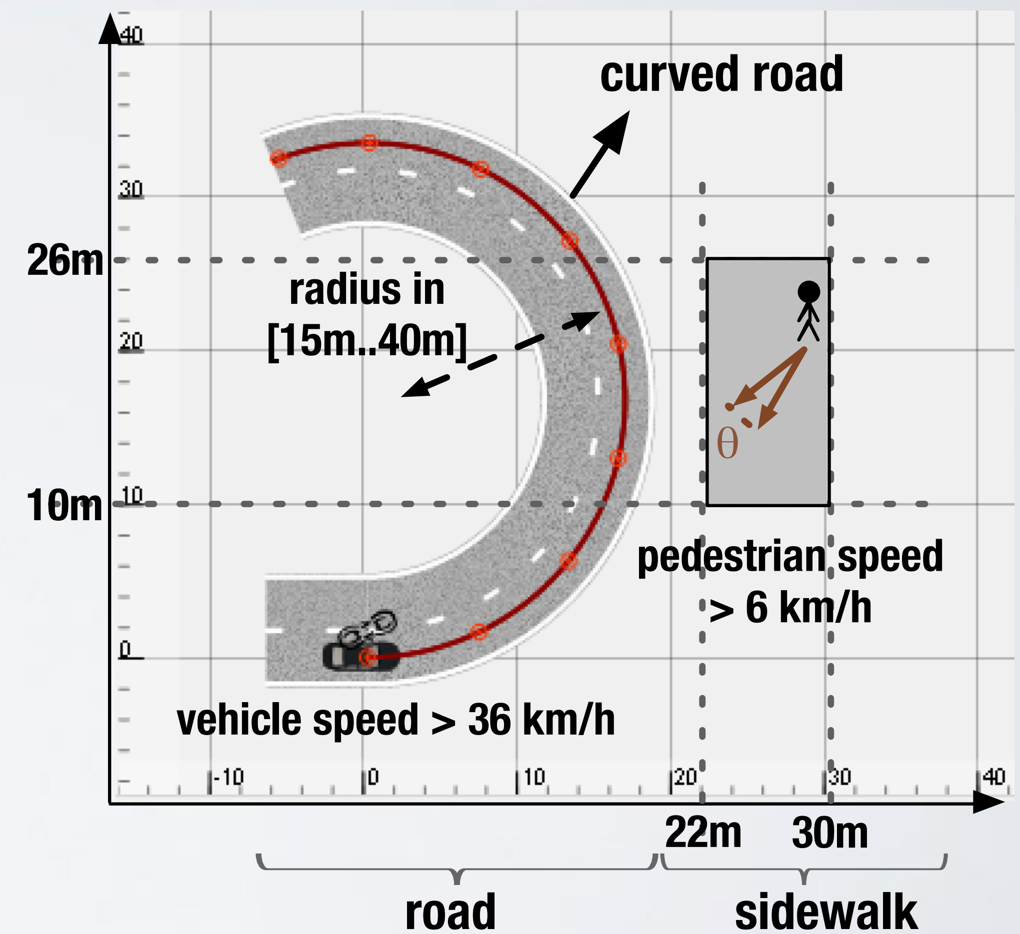
Genetic Evolution Guided by Classification



1. Initial Inputs ✓
2. Fitness Computation ✓
3. **Classification** ✓
4. Selection ✓
5. Breeding

Failure Explanation

- A characterisation of the input space showing **under what conditions the system is likely to fail**



- Path conditions in the decision tree
- Visualized by decision trees or dedicated diagrams

Results

- Does the decision tree technique help guide the evolutionary search and make it more effective?
  - Search with decision tree classifications can find 78% more distinct, critical test scenarios compared to a baseline search algorithm
- Does our approach help characterize and converge towards homogeneous critical regions?
  - The generated critical regions consistently become smaller, more homogeneous and more precise over successive tree generations

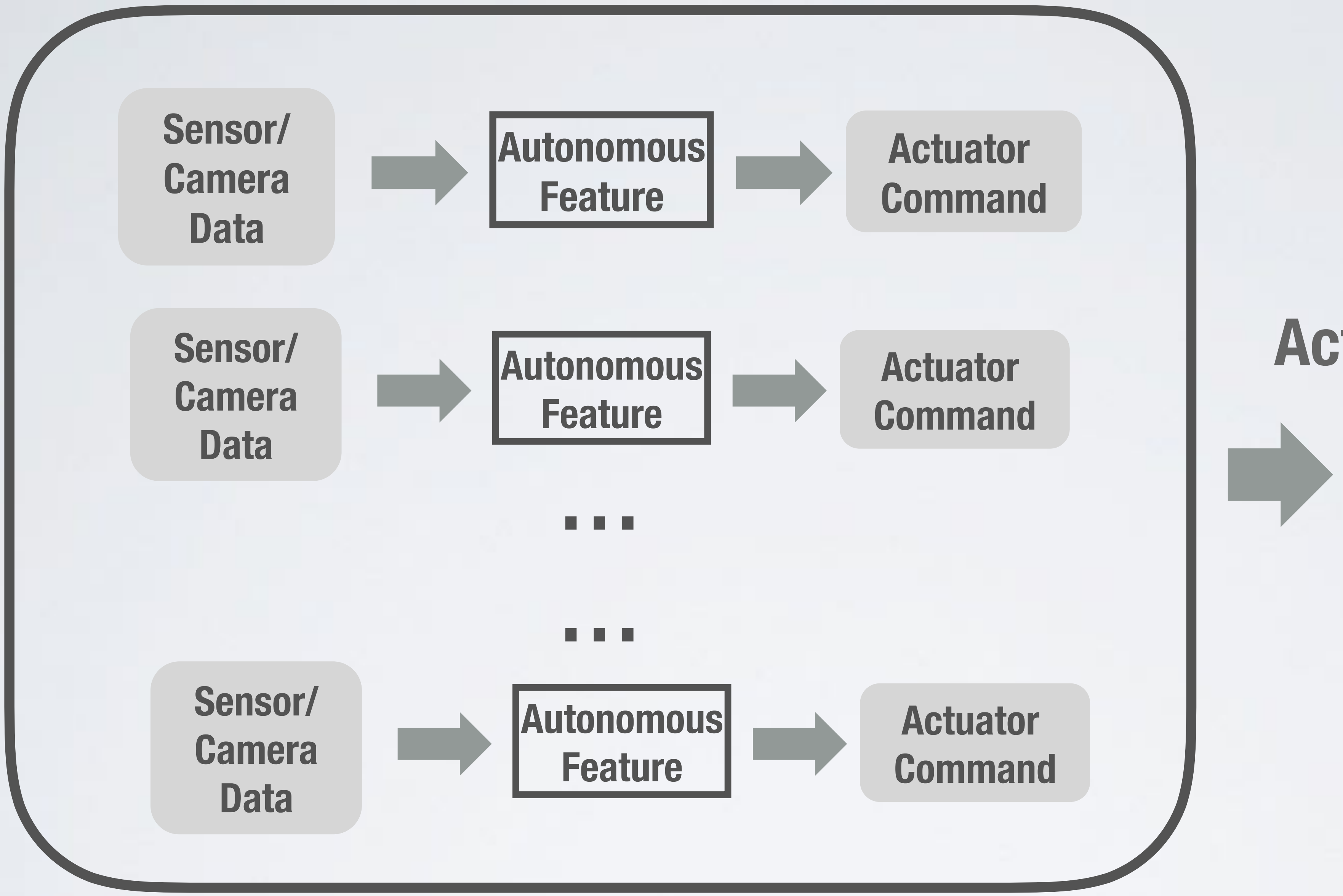
Usefulness

- The characterisations of the different critical regions can help with:

  (1) Debugging the system or the simulator

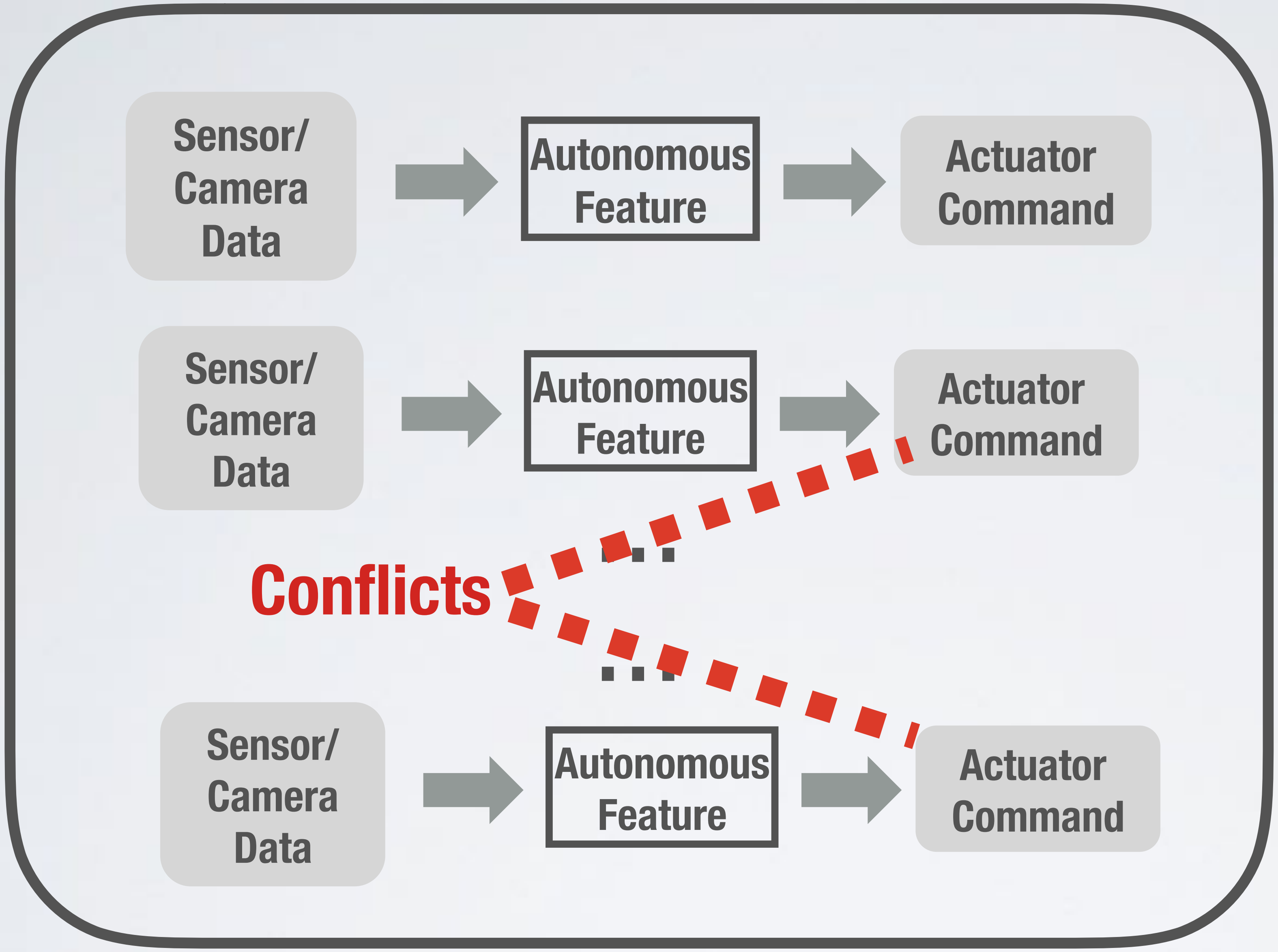
  (2) Identifying hardware changes to increase ADAS safety

  (3) Identifying proper warnings to drivers



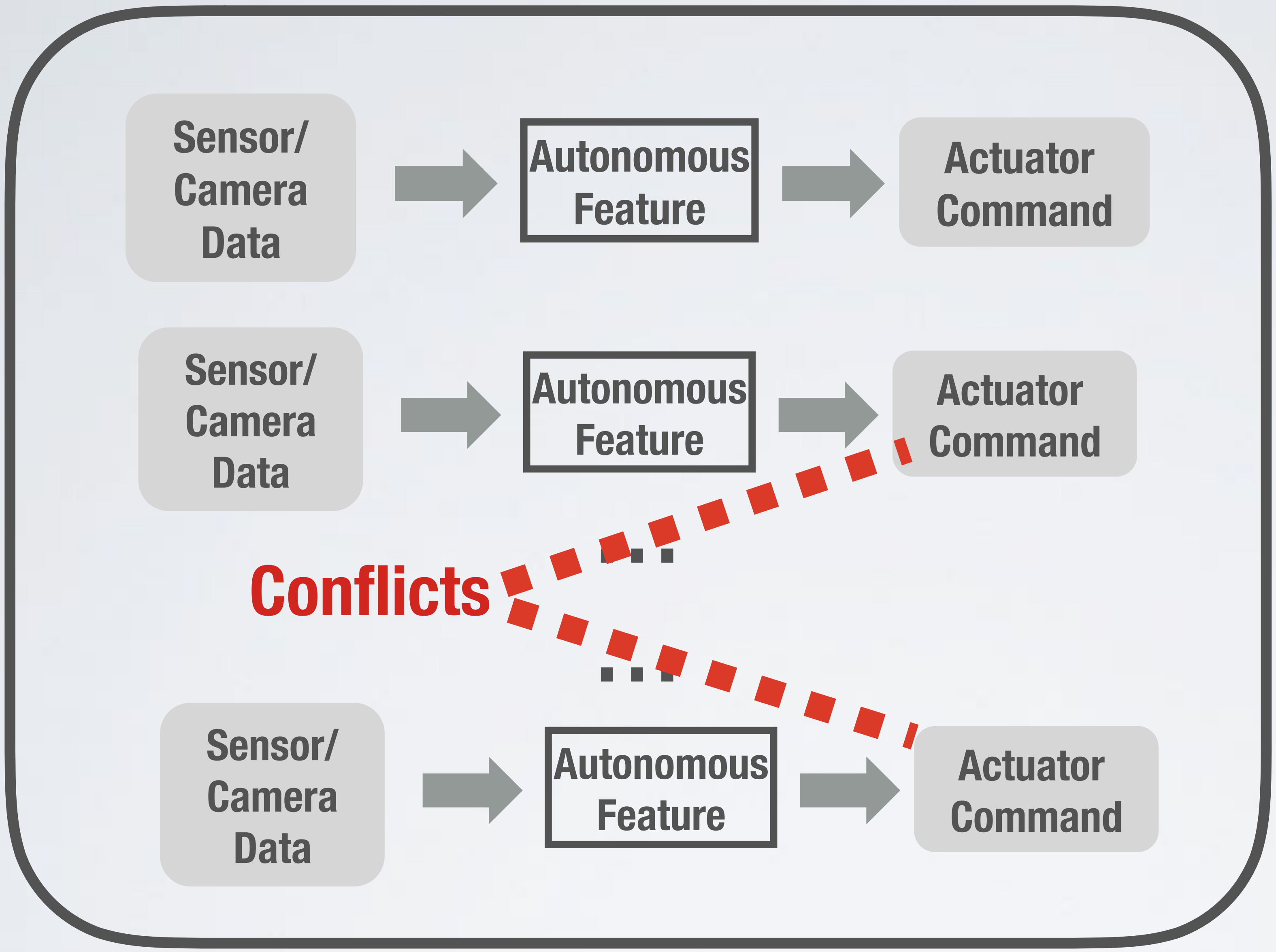
Actuator Commands:

- Steering
- Acceleration
- Braking



Actuator Commands:

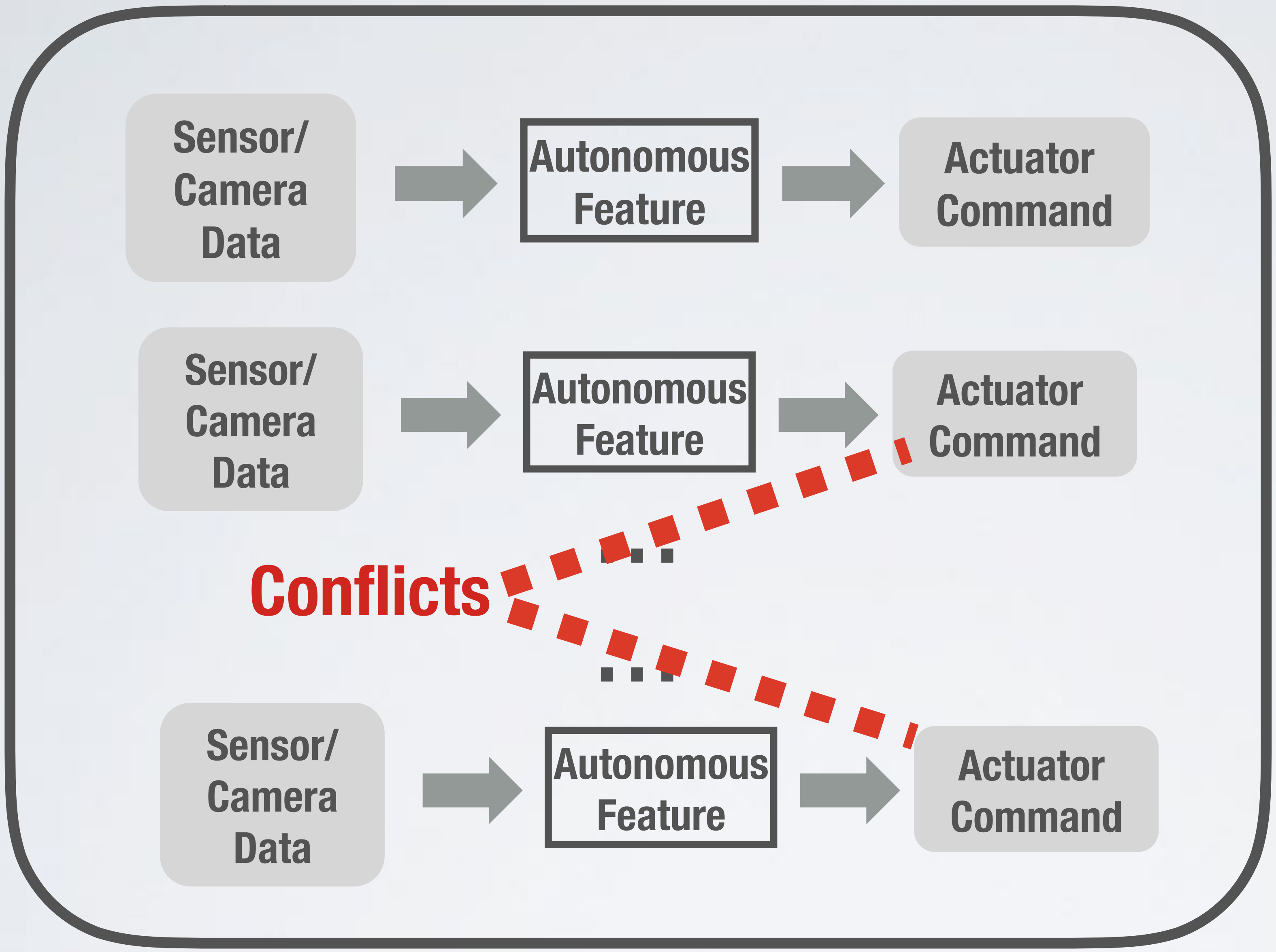
- Steering
- Acceleration
- Braking



Actuator Commands:

- Steering
- Acceleration
- Braking

Feature Interaction Problem



Actuator Commands:

- Steering
- Acceleration
- Braking

Undesired Feature Interactions

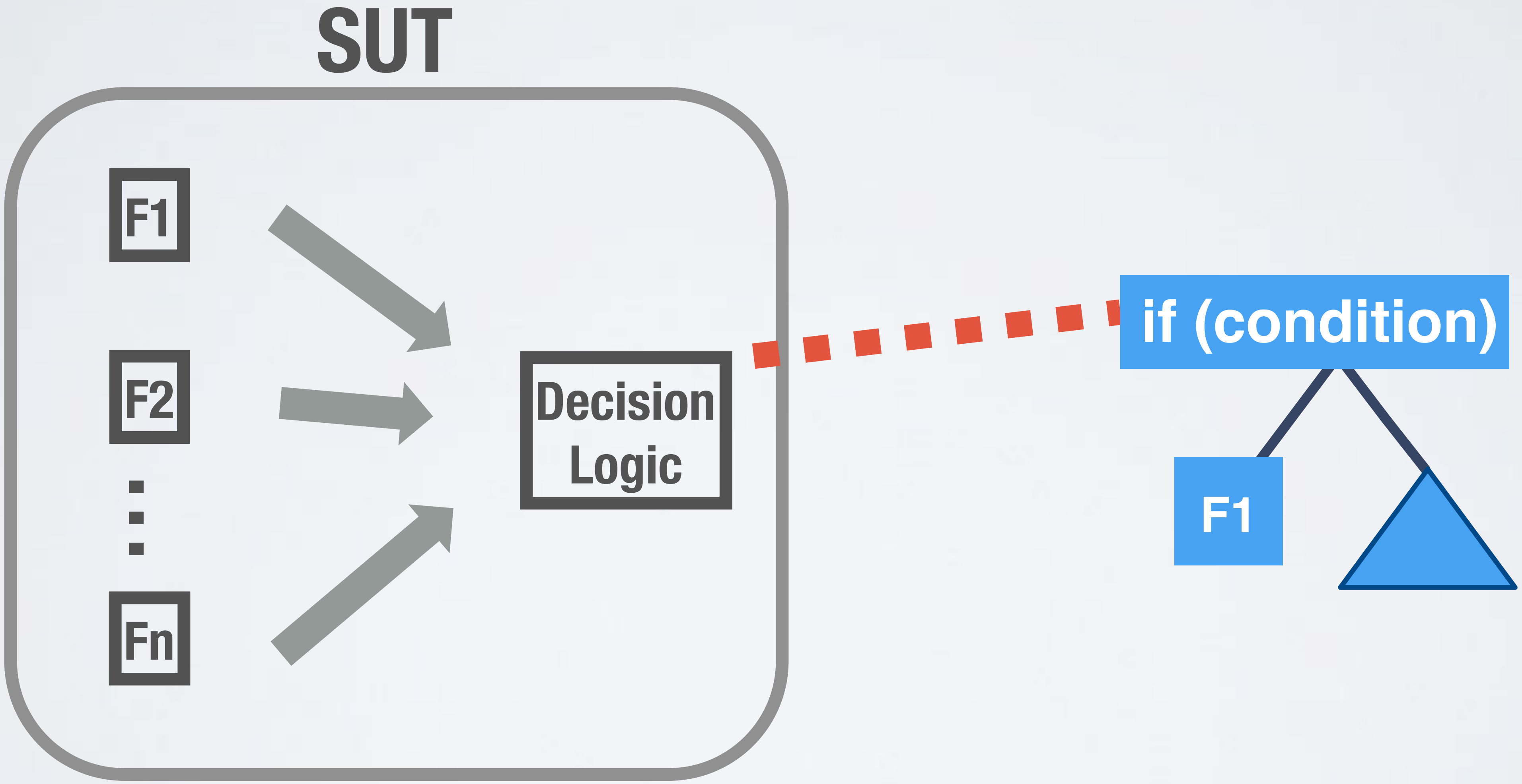
Using search-based testing to detect undesired feature interactions among function models of self-driving systems

Our Fitness Function

- A combination of three heuristics
  - Coverage-based
  - Failure-based
  - Unsafe overriding

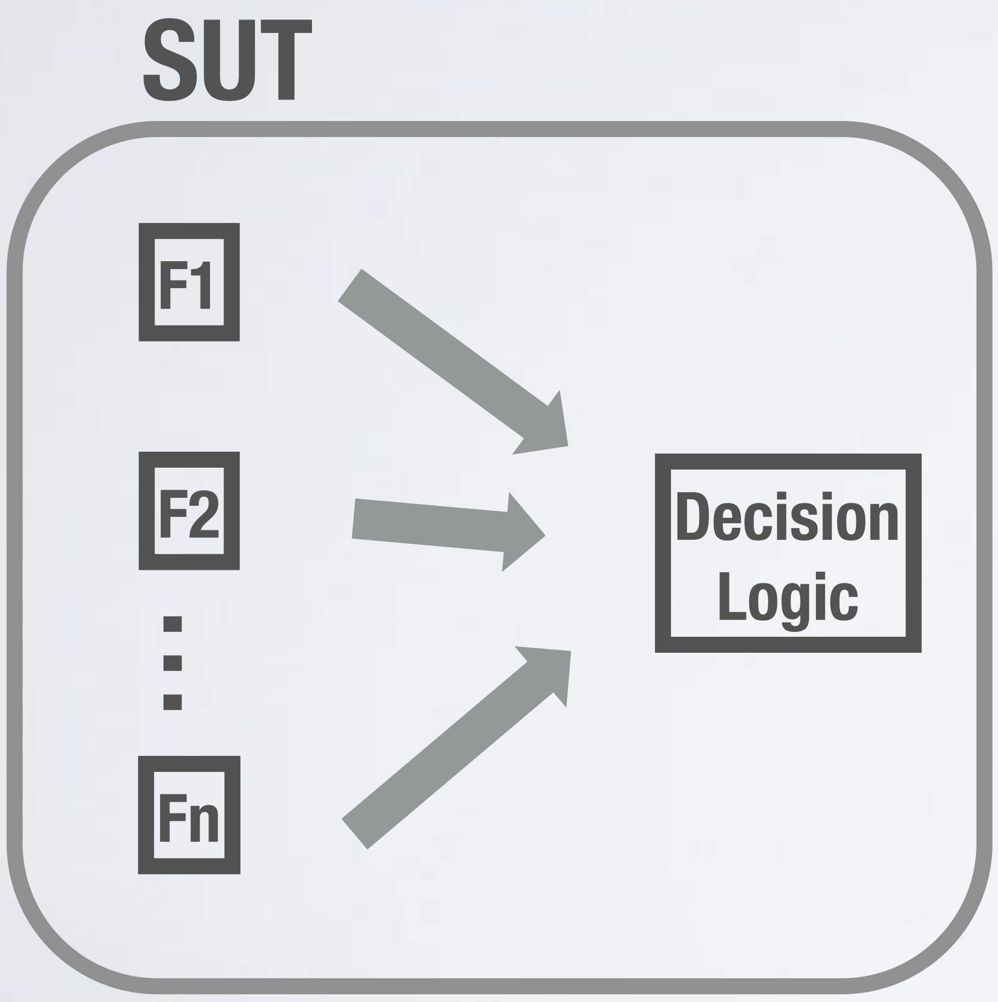
Coverage-based Objective

Goal: Exercising as many decision rules as possible



Failure-based Test Objective

Goal: Revealing violations of system-level requirements

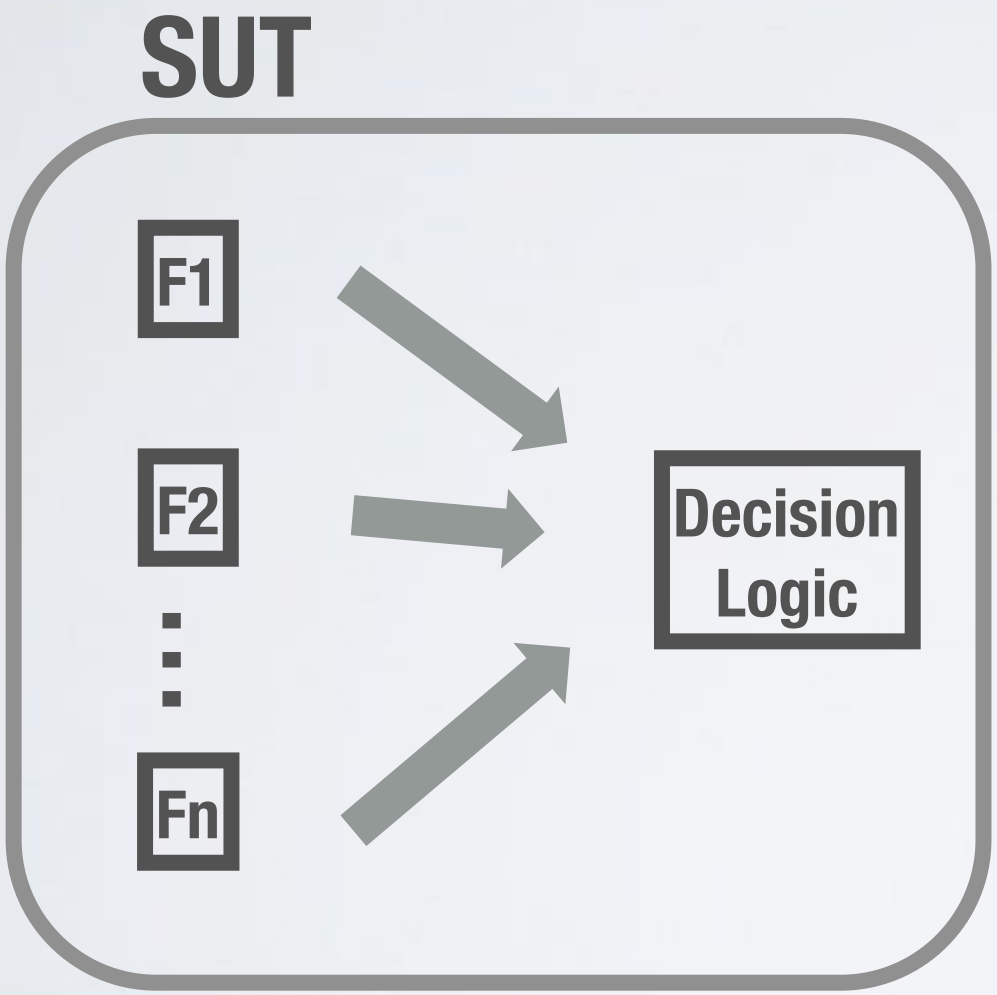


Example:

- Req: No collision between pedestrians and cars
- Generating test cases that minimize the distance between the car and the pedestrian

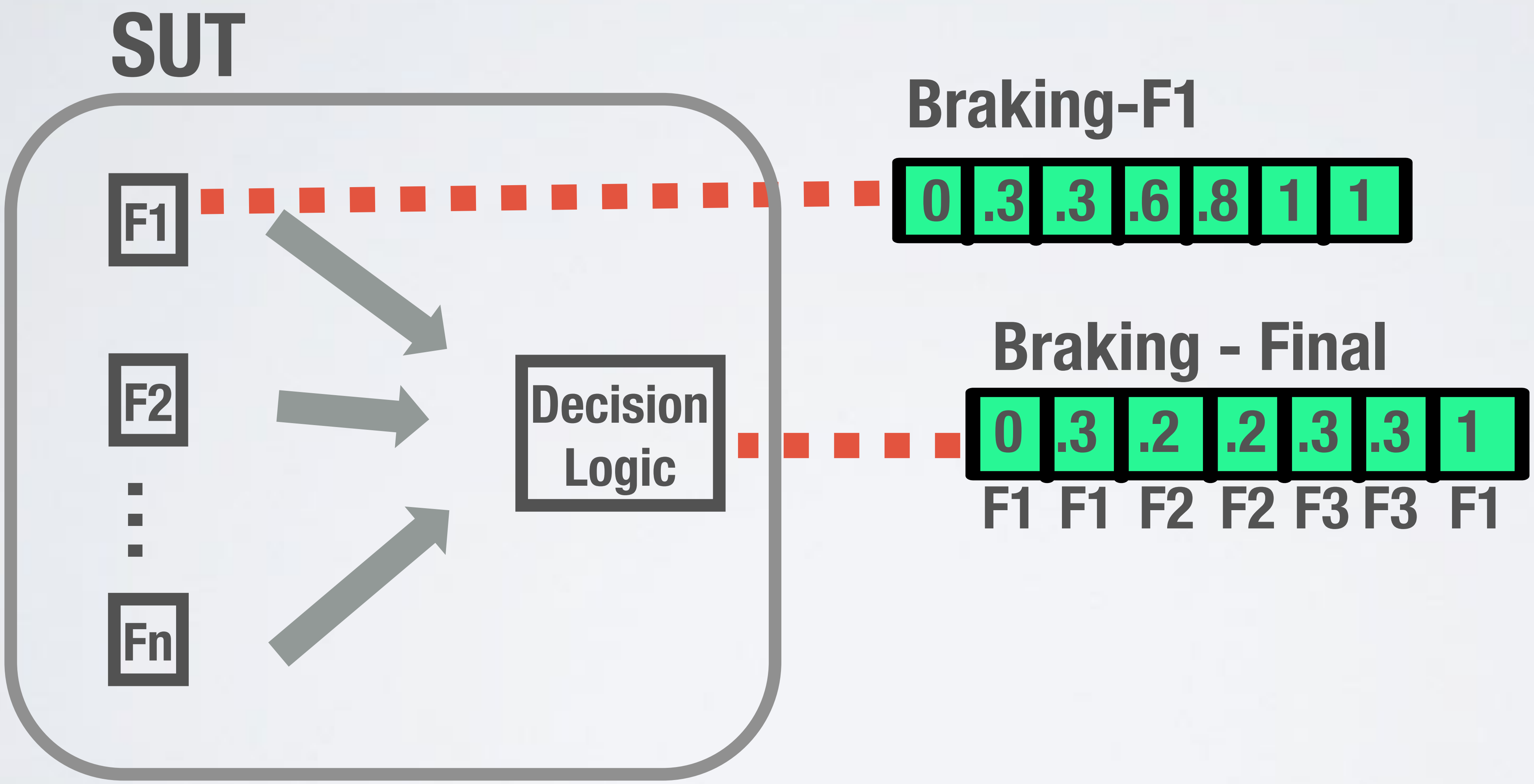
Feature Interaction Test Objective

Goal: Finding failures that are more likely to be due to faults in the integration component rather than faults in the features



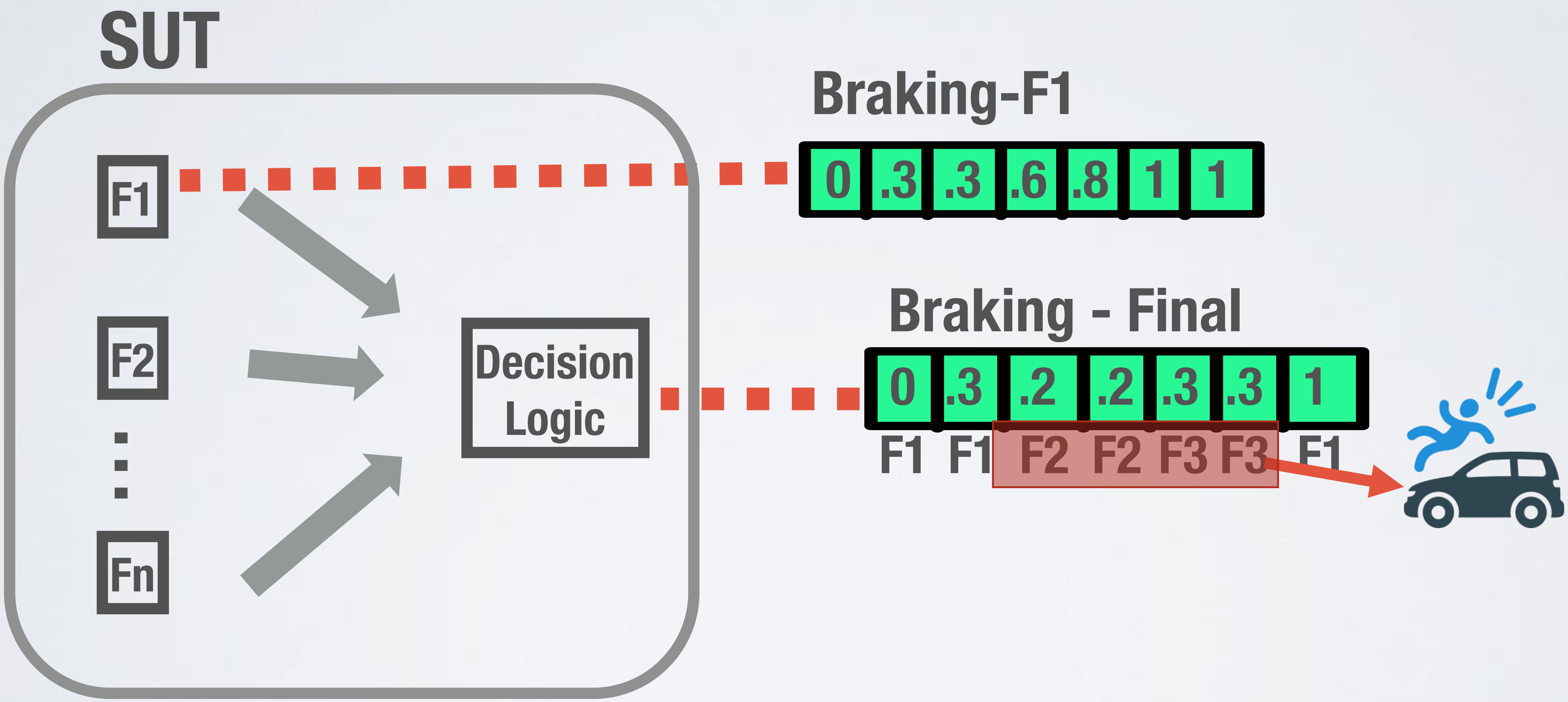
Feature Interaction Test Objective

Goal: Finding failures that are more likely to be due to faults in the integration component rather than faults in the features



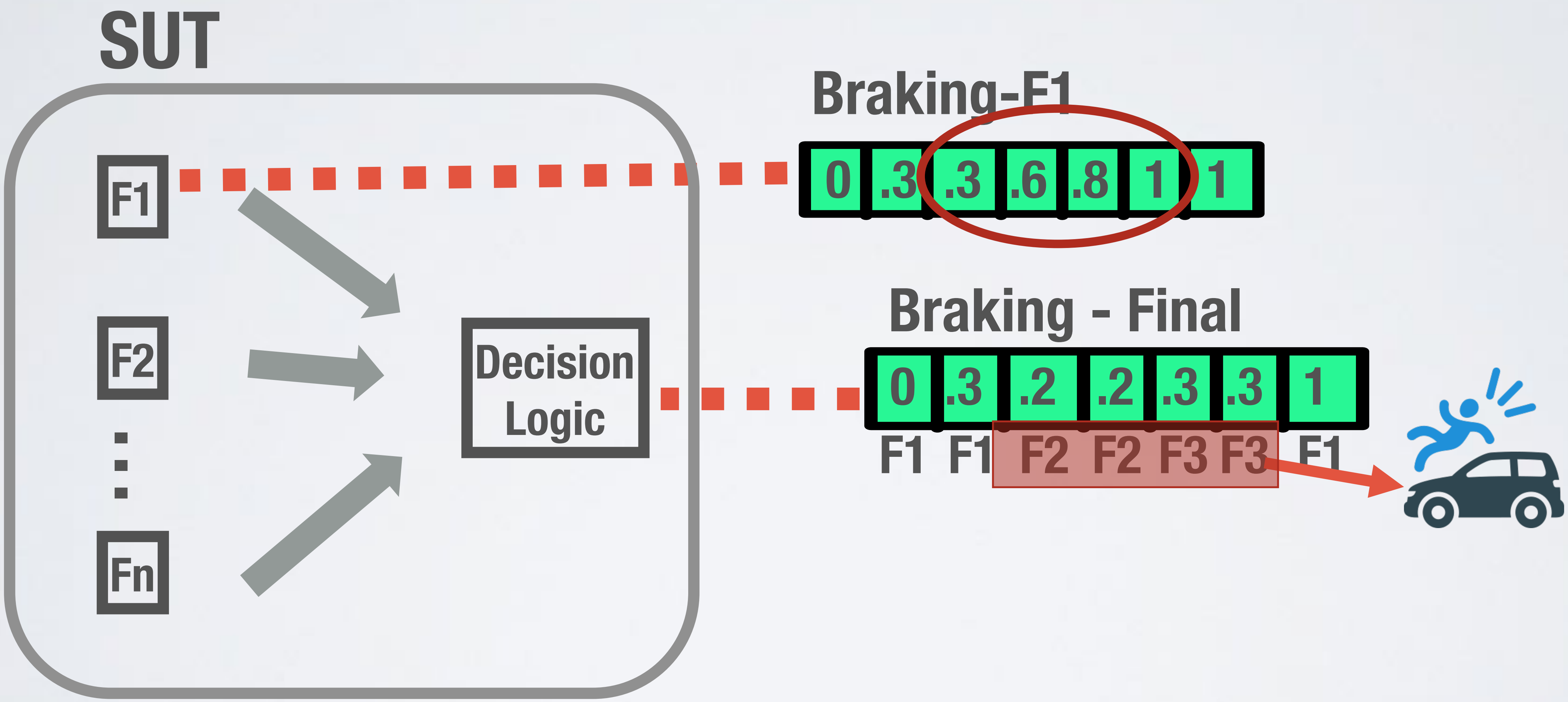
Feature Interaction Test Objective

Goal: Finding failures that are more likely to be due to faults in the integration component rather than faults in the features



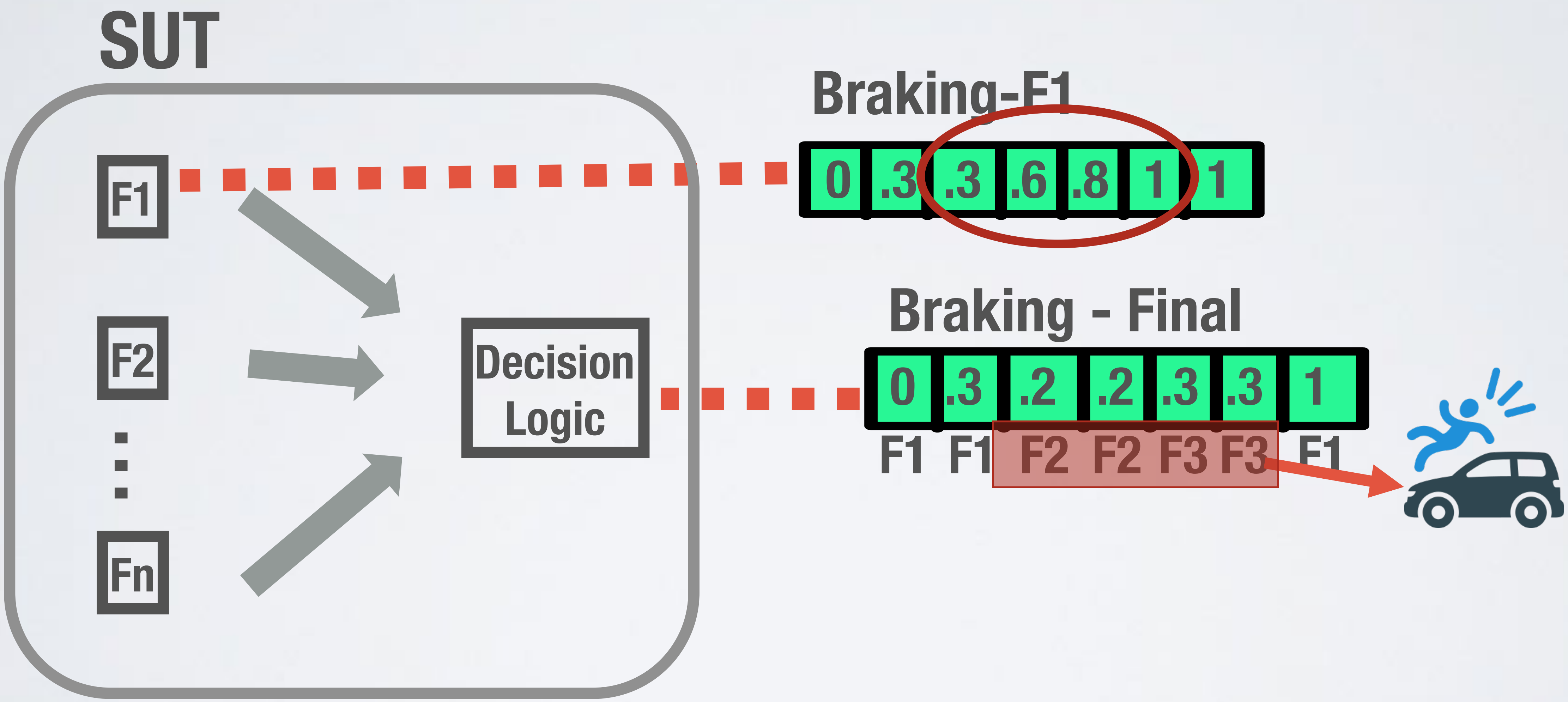
Feature Interaction Test Objective

Goal: Finding failures that are more likely to be due to faults in the integration component rather than faults in the features



Feature Interaction Test Objective

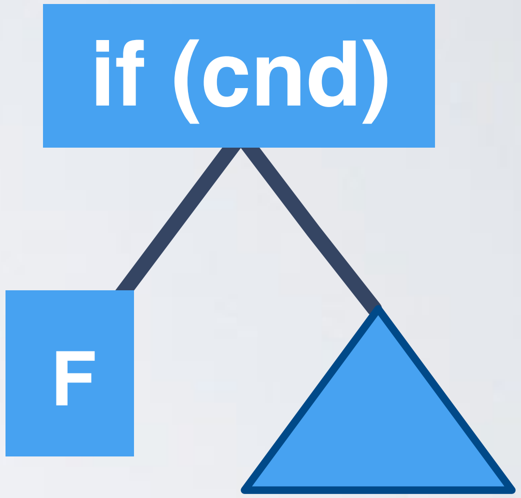
Goal: Finding failures that are more likely to be due to faults in the integration component rather than faults in the features



Reward failures that could have been avoided if another feature had been prioritised by the decision rules

On Hybrid Fitness Function

One hybrid test objective $\Omega_{j,l}$ for every rule *j* and every requirement *l*

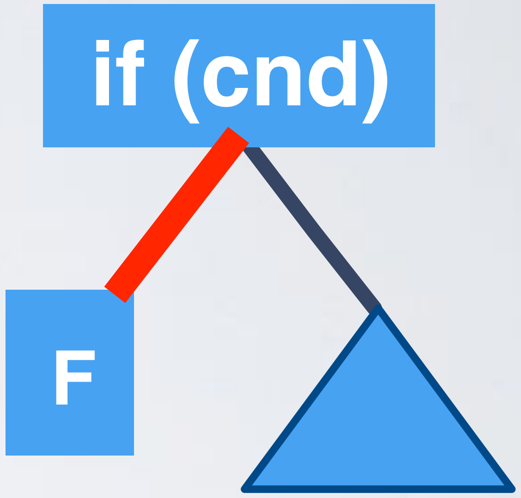


On Hybrid Fitness Function

One hybrid test objective $\Omega_{j,l}$ for every rule *j* and every requirement *l*

$\Omega_{j,l}(tc) > 2$

tc does not cover Branch *j*

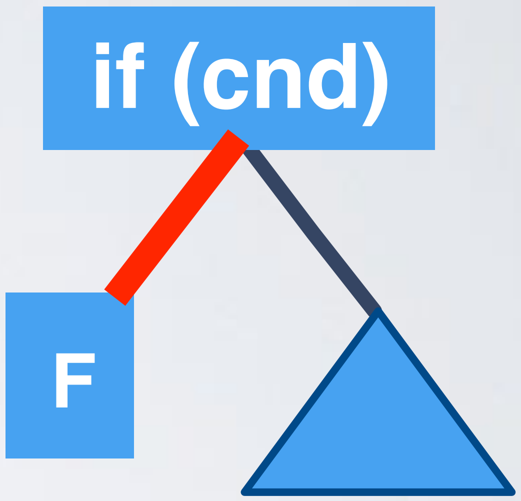


On Hybrid Fitness Function

One hybrid test objective $\Omega_{j,l}$ for every rule *j* and every requirement *l*

$\Omega_{j,l}(tc) > 2$ — *tc* does not cover Branch *j*

$2 \geq \Omega_{j,l}(tc) > 1$ — *tc* covers branch *j* but F is not unsafely overriden



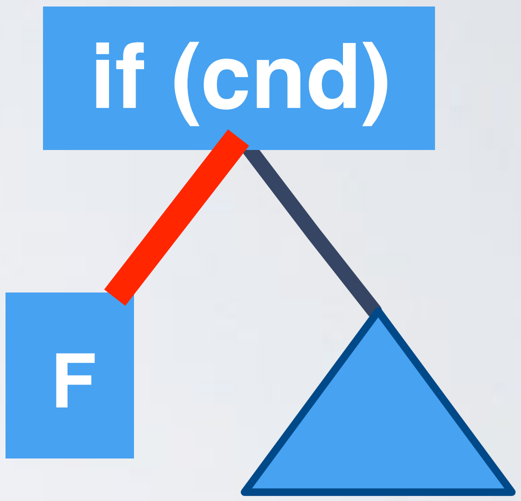
On Hybrid Fitness Function

One hybrid test objective $\Omega_{j,l}$ for every rule *j* and every requirement *l*

$\Omega_{j,l}(tc) > 2$ — *tc* does not cover Branch *j*

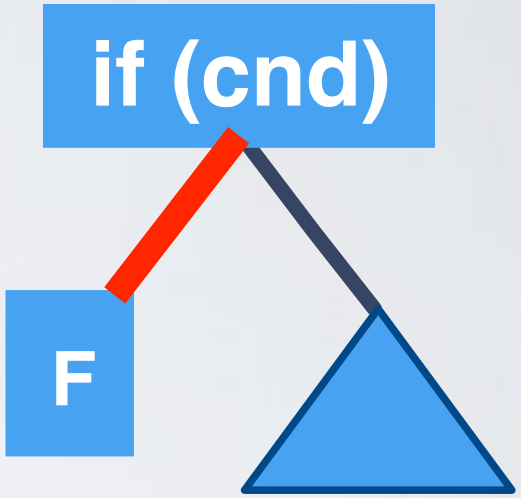
$2 \geq \Omega_{j,l}(tc) > 1$ — *tc* covers branch *j* **but** F is not unsafely overriden

$1 \geq \Omega_{j,l}(tc) > 0$ — *tc* covers branch *j* and F is unsafely overriden **but** req *l* is not violated



On Hybrid Fitness Function

One hybrid test objective $\Omega_{j,l}$ for every rule *j* and every requirement *l*



$\Omega_{j,l}(tc) > 2$ — *tc* does not cover Branch *j*

$2 \geq \Omega_{j,l}(tc) > 1$ — *tc* covers branch *j* **but** F is not unsafely overriden

$1 \geq \Omega_{j,l}(tc) > 0$ — *tc* covers branch *j* and F is unsafely overriden **but** req *l* is not violated

$\Omega_{j,l}(tc) = 0$ — **A feature interaction failure is likely detected**

Search Algorithm

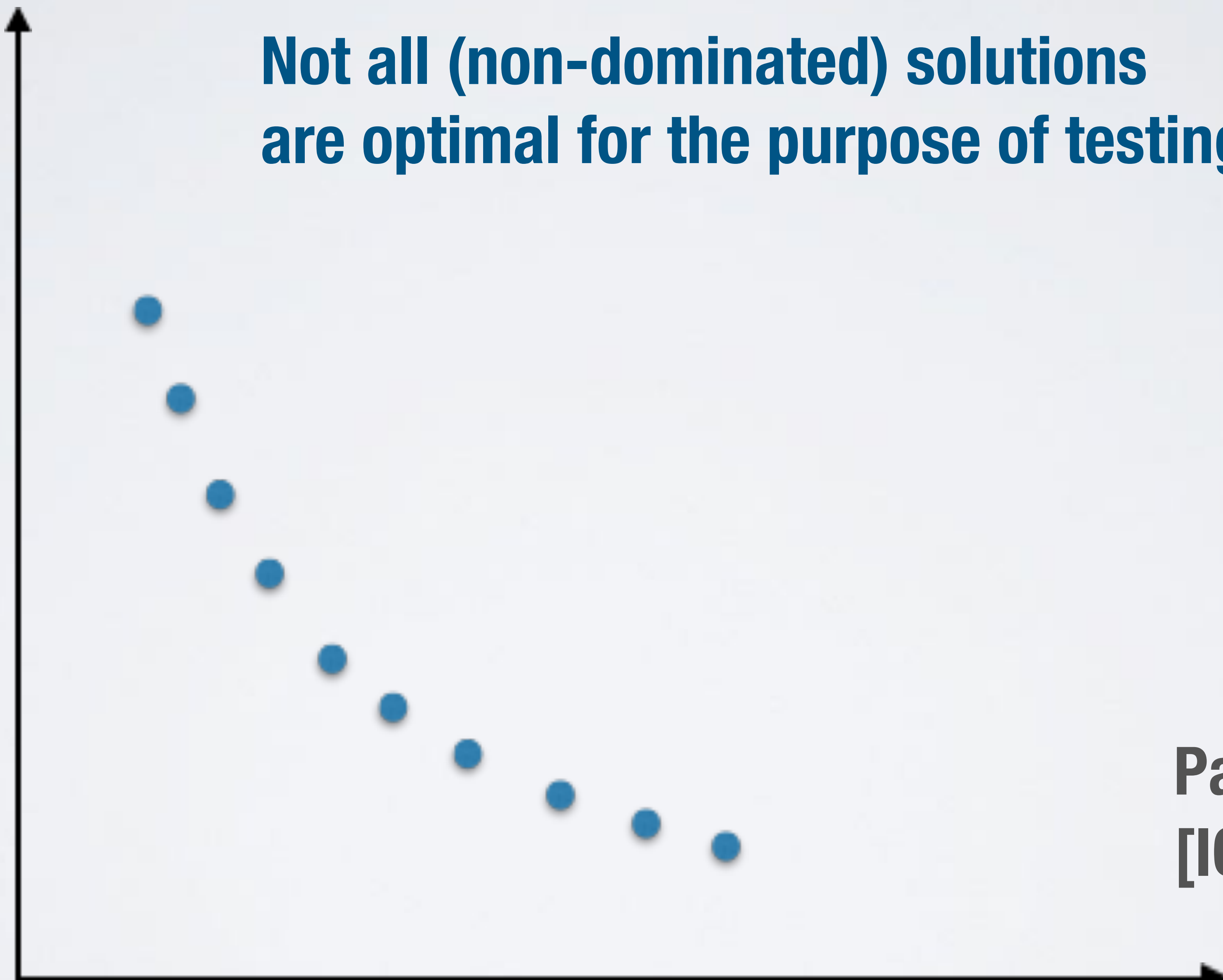
- **Goal: Computing a test suite that covers all the test objectives**
- Challenges:
  - The number of test objectives is **large**:

    $$\# \text{ of requirements} \times \# \text{ of rules}$$

  - Computing test objectives is computationally **expensive**
  - Not a Pareto front optimization problem
    - Objectives **compete** with each others, e.g., cannot have, in a single test scenario, a car that violates the speed limit after hitting the leading car

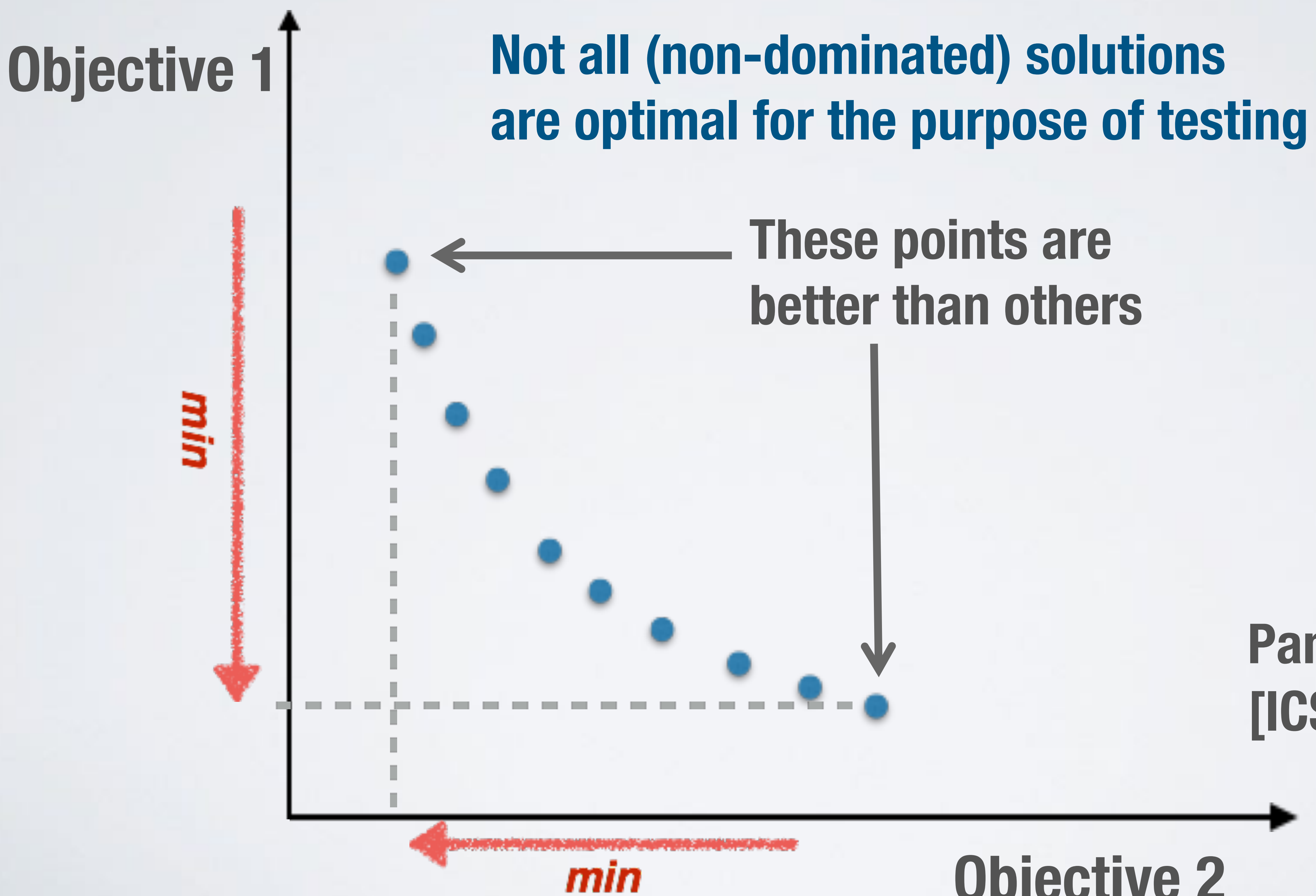
MOSA: Many-Objective Search-based Test Generation

Not all (non-dominated) solutions are optimal for the purpose of testing

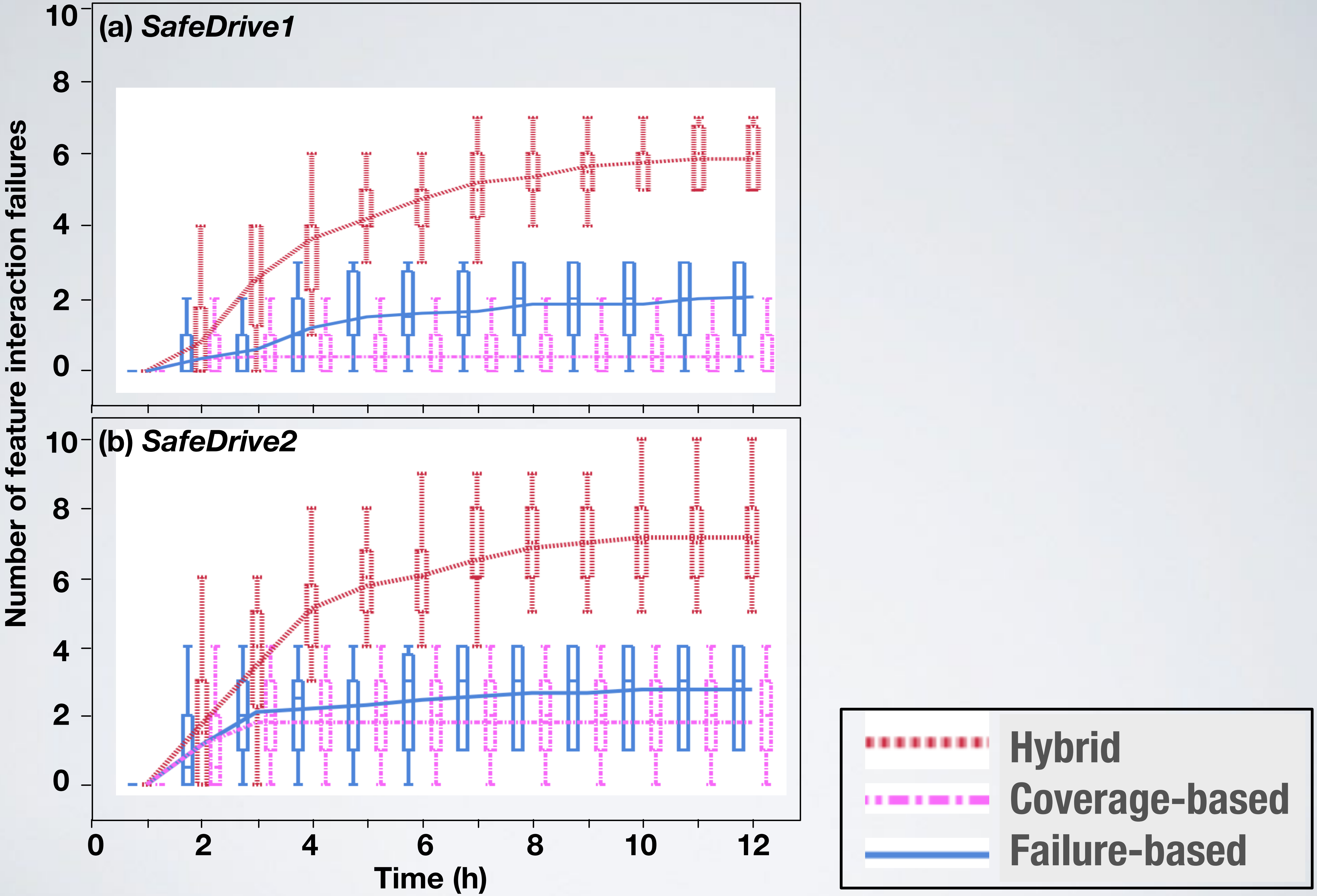


Panichella et. al.
[ICST 2015]

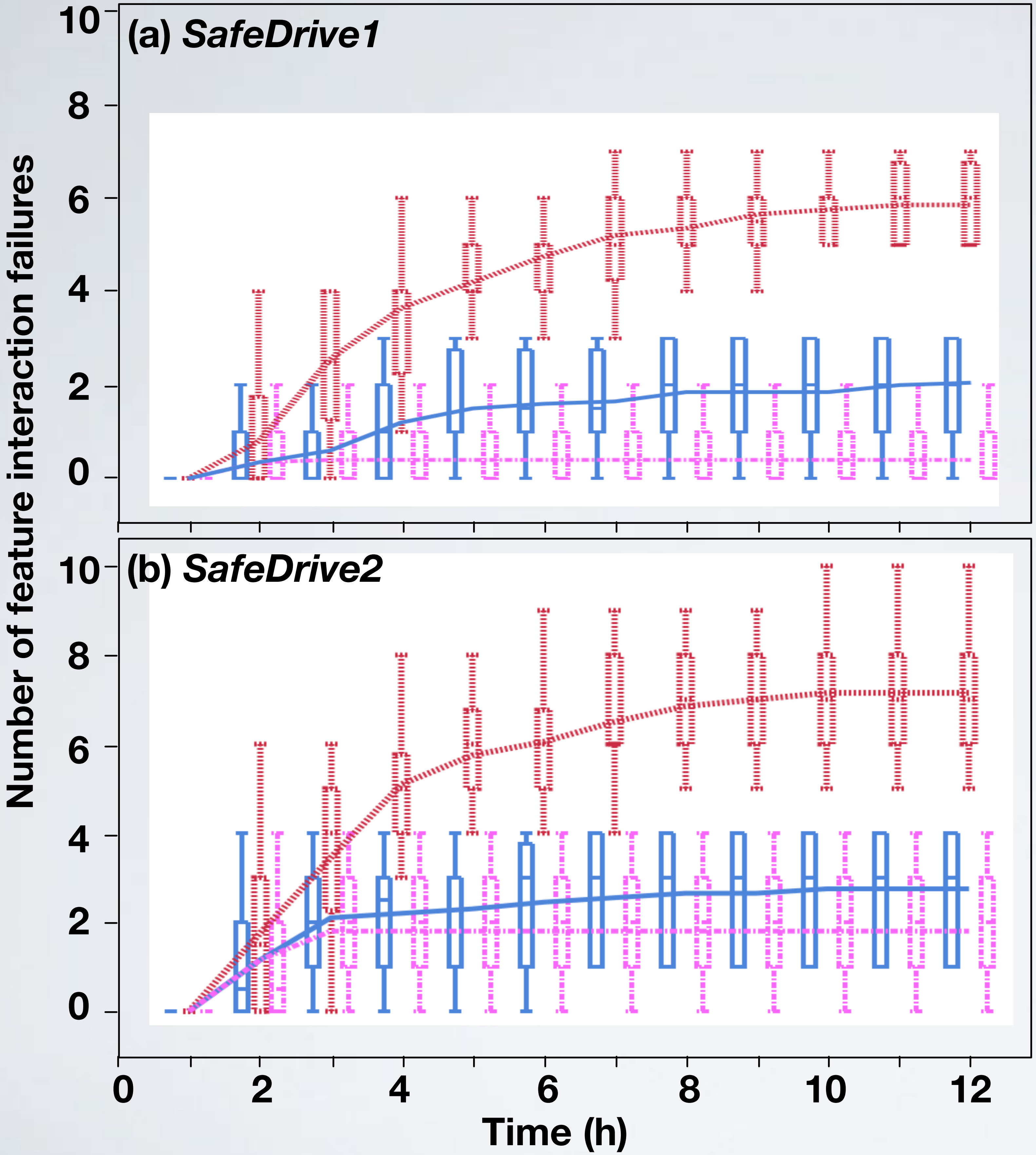
MOSA: Many-Objective Search-based Test Generation



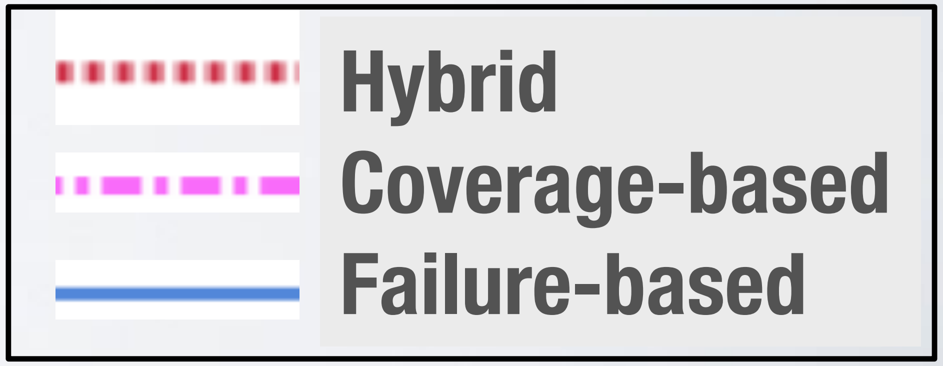
Panichella et. al.
[ICST 2015]



(a) SafeDrive1
(b) SafeDrive2
Number of feature interaction failures
Time (h)
0
2
4
6
8
10
12
Hybrid
Coverage-based
Failure-based



Hybrid test objectives reveal significantly more feature interaction failures (more than twice) compared to baseline alternatives



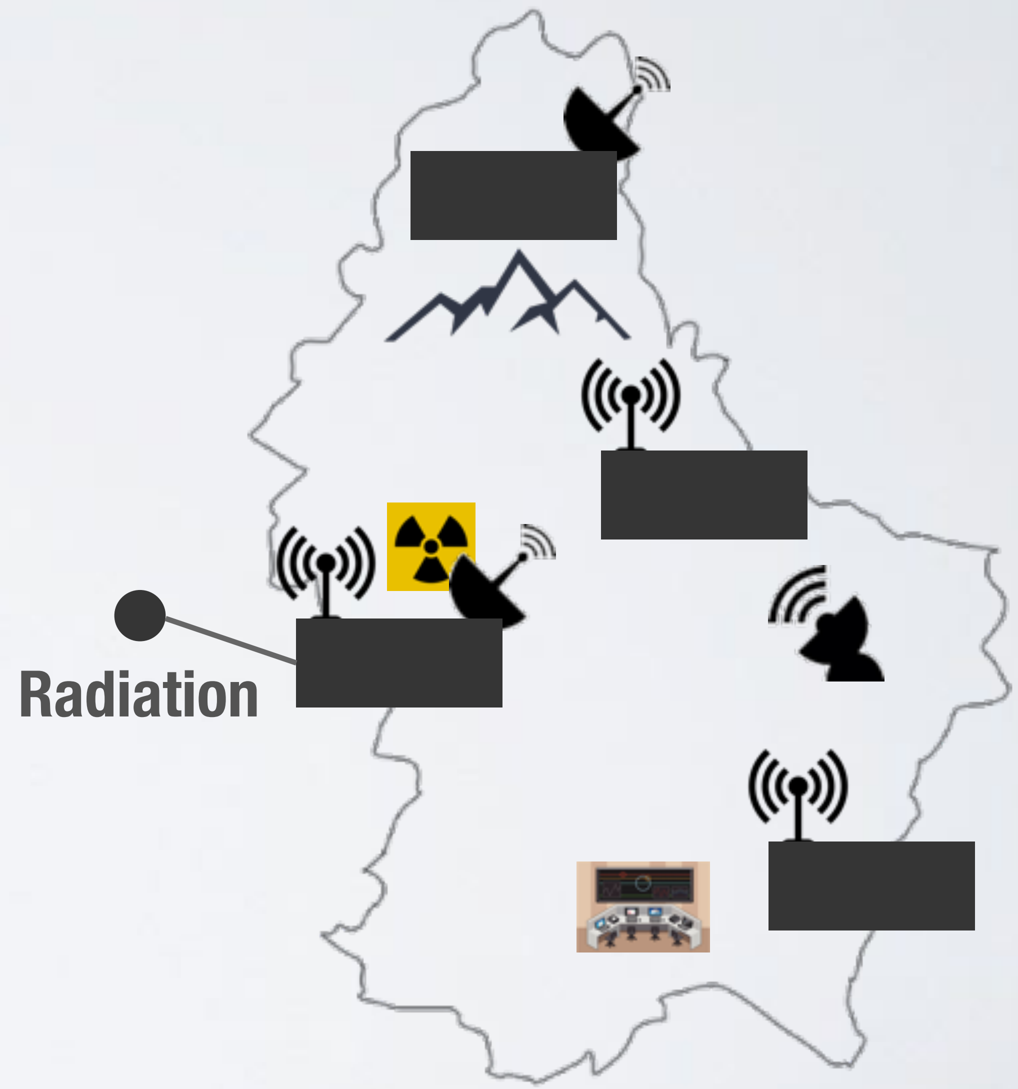
Feedback from Domain Experts

- The failures we found were due to undesired feature interactions
- The failures were not previously known to them
- We identified ways to improve the decision logic (integration component) to avoid failures

Example Feature Interaction Failure

Luxembourg Emergency Management System

- Goal: **Monitoring** emergency situations and providing a robust **communication** platform for disaster situations
- Requirements
  - **Resilience**
  - Maintaining an acceptable level of **quality of service** in the face of emergency situations



Concluding Remarks

Search-Based Testing

- Versatile
  - Can be applied to complex systems (non-linear, non-algebraic, continuous, heterogeneous)
  - Can be used when systems have black box components or rely on computer simulations
- Scalable, easy to parallelize
- Can be combined with: Machine learning, Statistics, Solvers, e.g., SMT and CP

Conclusions

- Contextual factors influence both the significance of a problem and the shape of the solution
  - Our context: function models capturing CPS continuous dynamics, functional requirements and simulators capturing environment and hardware
- Focus on system-level testing
  - Not just on the perception layer (DNN) or the decision layer or the control layer
- We have to deal with computational complexity, heterogeneity and very large input spaces

- Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, Thomas Stifter,``Testing vision-based control systems using learnable evolutionary algorithms”, ICSE 2018: 1016-1026

- Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, Thomas Stifter, ``Testing autonomous cars for feature interaction failures using many-objective search”, ASE 2018: 143-154

- Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, Thomas Stifter, ``Testing advanced driver assistance systems using multi-objective search and neural networks”, ASE 2016: 63-74

- Annibale Panichella, Fitsum Meshesha Kifetew, Paolo Tonella, ``Reformulating Branch Coverage as a Many-Objective Optimization Problem”, ICST 2015: 1-10

- Nejati et al., “Evaluating Model Testing and Model Checking for Finding Requirements Violations in Simulink Models”, arXiv:1905.03490, 2019

We are hiring!

Talk to me if you are interested in research positions in any of the following areas: Applied Machine Learning, Applied Natural Language Processing, Automated Verification and Validation, Information Retrieval, Model-driven Engineering, Program Analysis, Requirements Engineering, Software Security, Software Testing