The Research Challenges of the ERC Project PRECRIME

Paolo Tonella

Fondazione Bruno Kessler

Trento - Italy

FONDAZIONE BRUNO KESSLER

SE SOFTWARE ENGINEERING

Outline

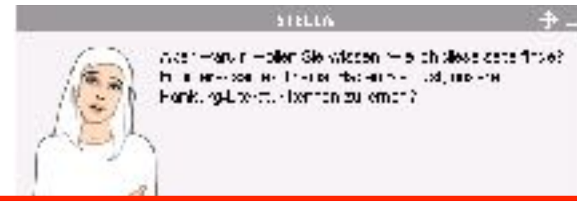
1. **Motivating scenarios**

2. **PRECRIME**
   - Goals
   - Challenges
   - Approach

3. **Building blocks**
   - Oracle quality
   - Evolutionary testing

4. **Conclusion**

Land of AI-based systems

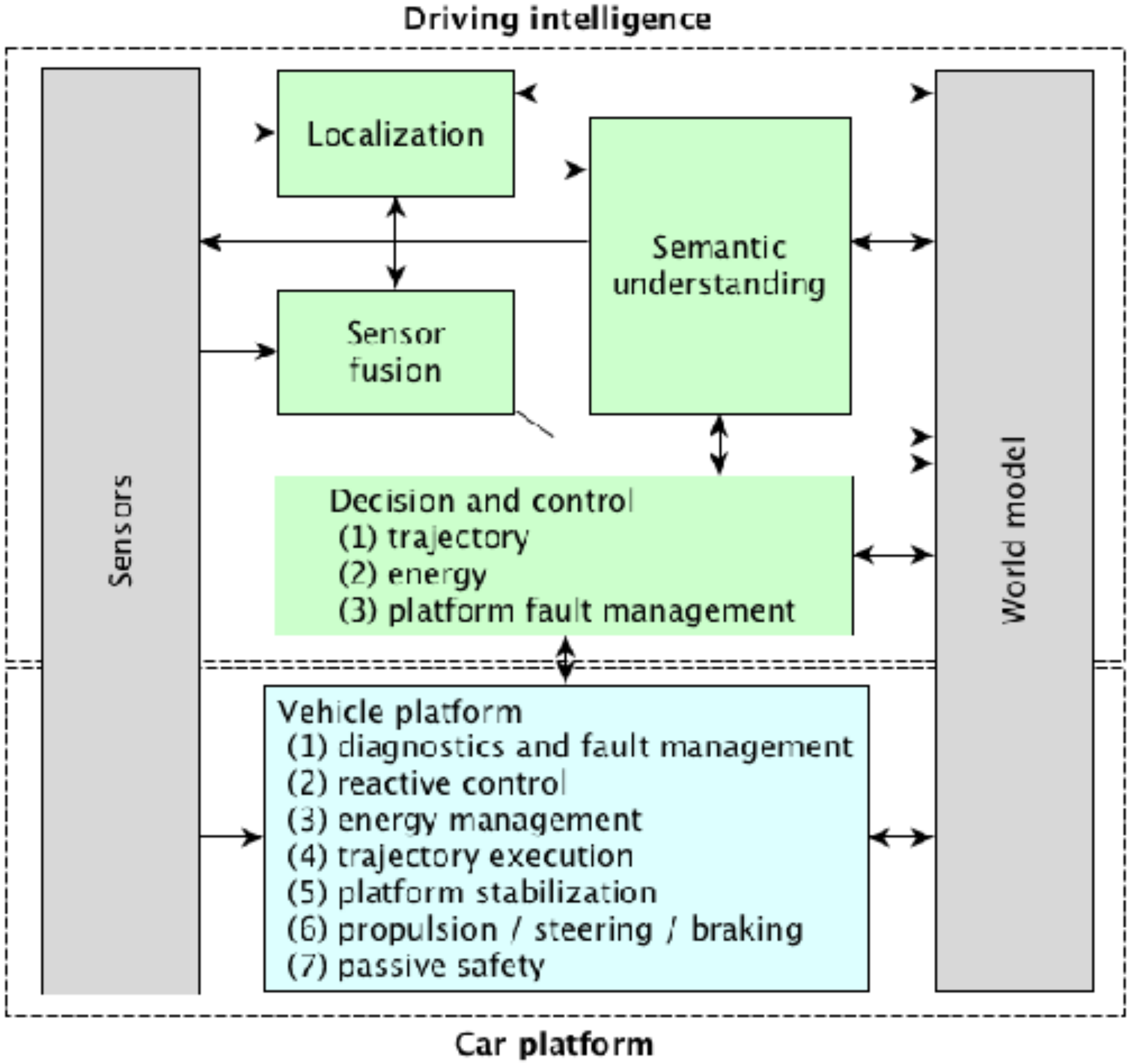


How can we test effectively and efficiently AI based systems?

Impact on all sectors of our society:

- **Mobility**
- **Industry**
- **Finance**
- **Services**
- **Entertainment**

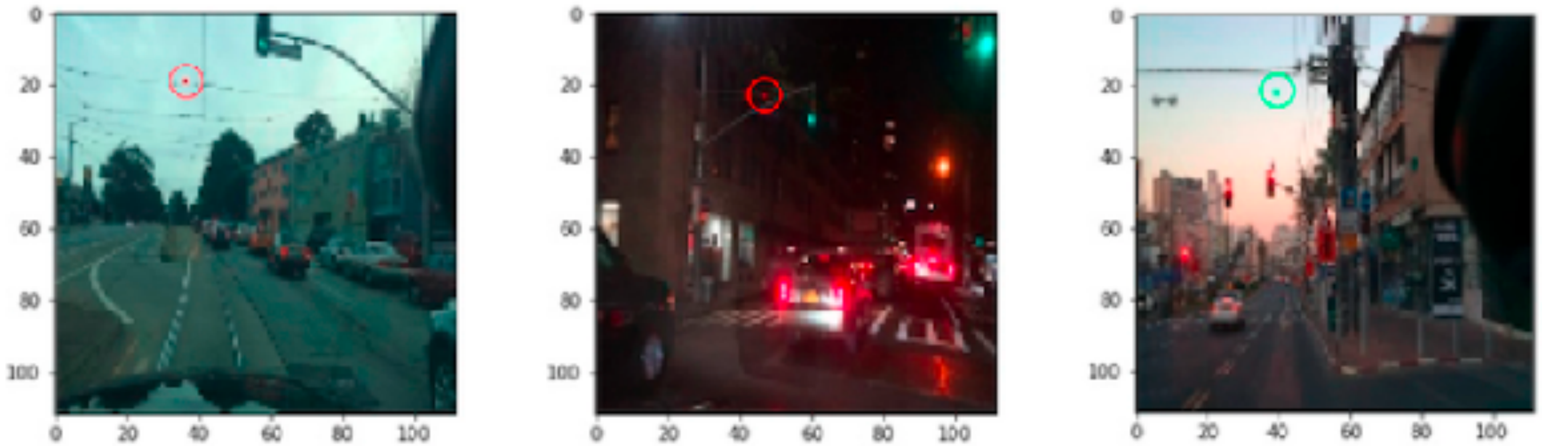
Autonomous driving



Testing issues:

- Simulation is slow
- Field testing is extremely expensive
- World model instances grow combinatorially
- Code coverage is not always an appropriate adequacy criterion
- Test scenarios should be at the same time realistic and extreme

Adversarial testing

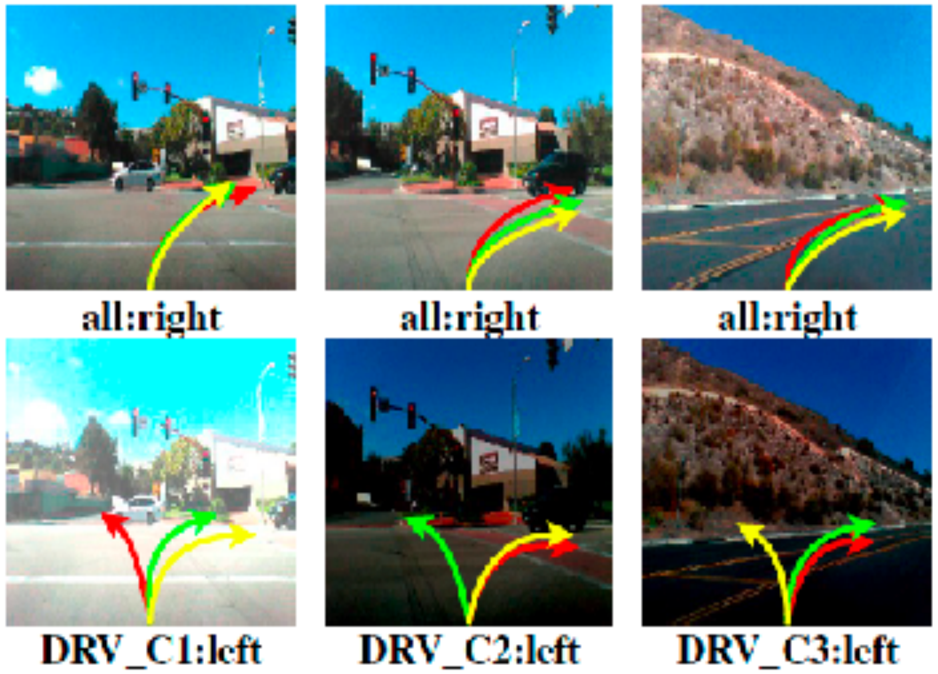


One-pixel change

- two player game
- Monte Carlo tree search

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, Min Wu. ***Safety Verification of Deep Neural Networks***. CAV 2017: 3-29

Adversarial testing

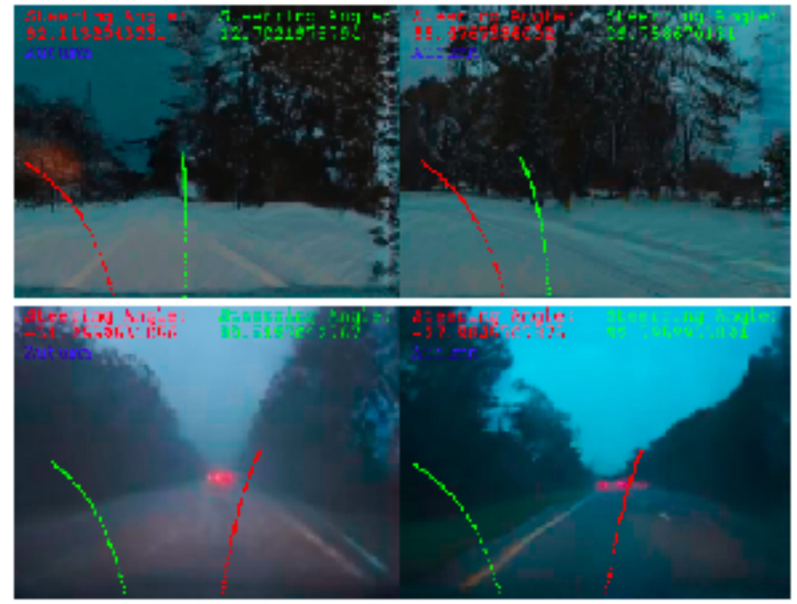


DeepXplore:

- max neuron coverage
- max differential behaviours
- gradient ascent optimization

K. Pei, Y. Cao, J. Yang, and S. Jana. ***DeepXplore: Automated whitebox testing of deep learning systems***. SOSP 2017

Adversarial testing



DeepRoad

- map image from source domain to latent domain
- generate image in the new domain from latent domain

Given such adversarial examples, how do we know if they may affect any real execution scenario?

Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, Sarfraz Khurshid. ***DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing***. arXiv:1802.02295

Financial bots

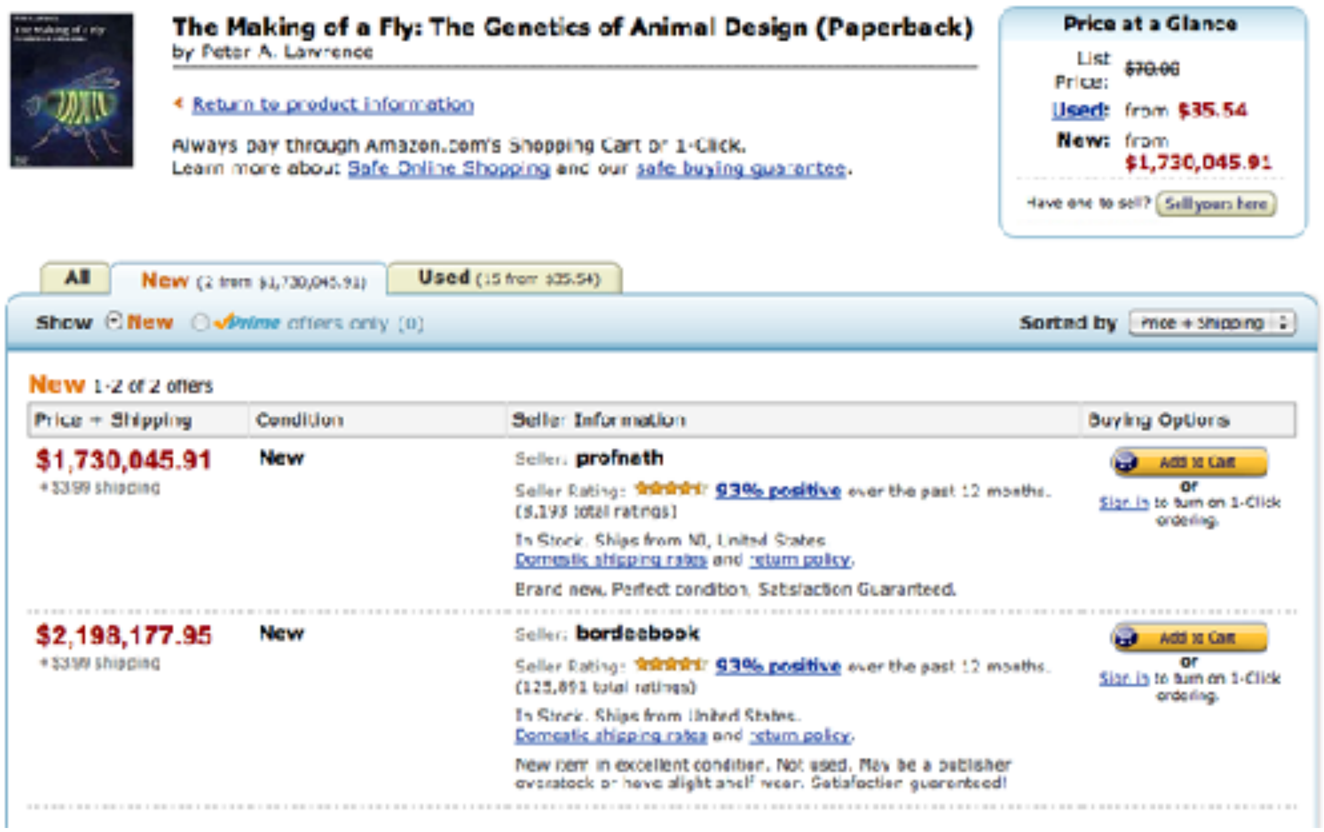
```
contract SendBalance {
 mapping (address => uint) userBalances;
 bool withdrawn = false;
 function getBalance(address u) constant returns(uint){
  return userBalances[u];
 }
 function addToBalance() {
  userBalances[msg.sender] += msg.value;
 }
 function withdrawBalance(){
  if (!(msg.sender.call.value(
    userBalances[msg.sender])())) { throw; }
  userBalances[msg.sender] = 0;
  }}
```

`withdrawBalance` is not reentrant:

- The default function `value` of the sender may call `withdrawBalance` again, causing a double transfer of money
- The recent `TheDao` hack exploited a reentrancy vulnerability to steal around 60 M$ from Ethereum
- A malicious bot may easily discover and trigger such reentrant calls

Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, Aquinas Hobor. ***Making Smart Contracts Smarter***. CCS 2016: 254-269

Financial bots



Algorithmic pricing

Algo 1:

`1% discount over min price`

Algo 2:

`27% extra cost for higher reliability and better services`

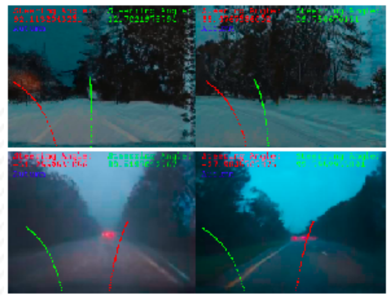
How do we test a society of intelligent bots so as to rule out undesirable emergent behaviours?

Test challenges

- **Latent bugs** and in-field misbehaviours are unavoidable
- Existing **adequacy criteria** are not sufficient
- Runtime monitoring for **fail-safe execution** becomes essential
- **Realistic**, yet extreme, **scenarios** should be generated for testing
- **Anomalies** and unexpected execution contexts should be detected at run time

Many activities traditionally conducted during offline/pre-release testing must be moved online/post-release

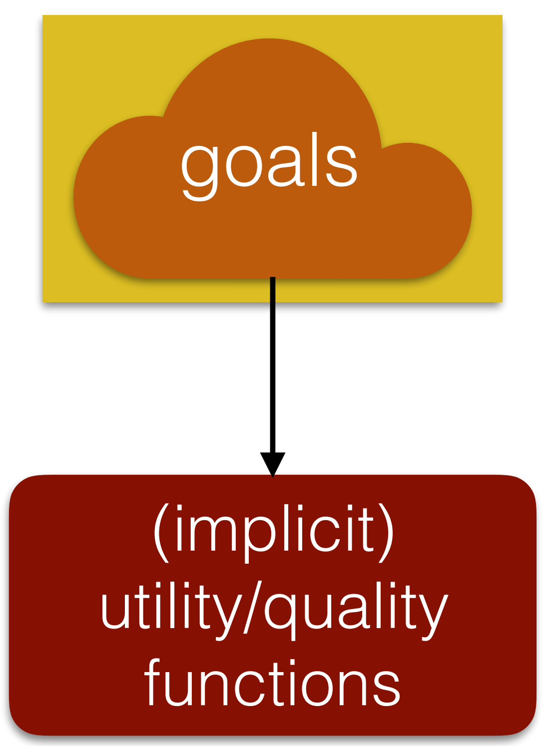
Is it still software testing?



```
contract SendBalance {
mapping (address => uint) userBalances;
bool withdrawn = false;
function getBalance(address u) constant returns(uint){
 return userBalances[u];
}
function addToBalance() {
 userBalances[msg.sender] += msg.value;
}
function withdrawBalance(){
 if (!(msg.sender.call.value(
   userBalances[msg.sender])())) { throw; }
 userBalances[msg.sender] = 0;
 }}
```

NO: no bug in the code

- Code implementing DNN is correct
- Learnt behaviours might be incorrect even if the learning algorithm is implemented correctly



TEST OBJECTIVES

YES: implementation deviates from intended behaviour

- Issue fixing might involve DNN retraining
- Training data and learning algorithms might *be* the fault, rather than just contain the fault

PRECRIME

Project

Self Assessment Oracles for Anticipatory Testing

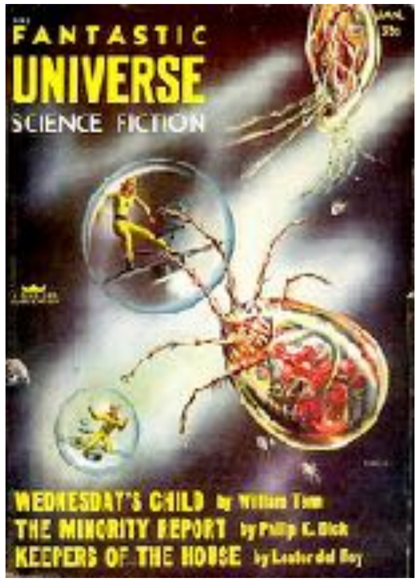
Facts:
- Funded by the European Research Council under the Advanced Grant programme
- Will start in Jan 2019; last for 5 years
- Team composed of 10 people (PI, 4 Postdocs, 4 PhD Students, 1 Technologist)
- Website: pre-crime.eu

Inspiration

MINORITY REPORT

- **Precrime**, police agency that blocks and imprison murderers before they commit crime
- **Precogs**, mutants endowed with the ability to see future events before they happen

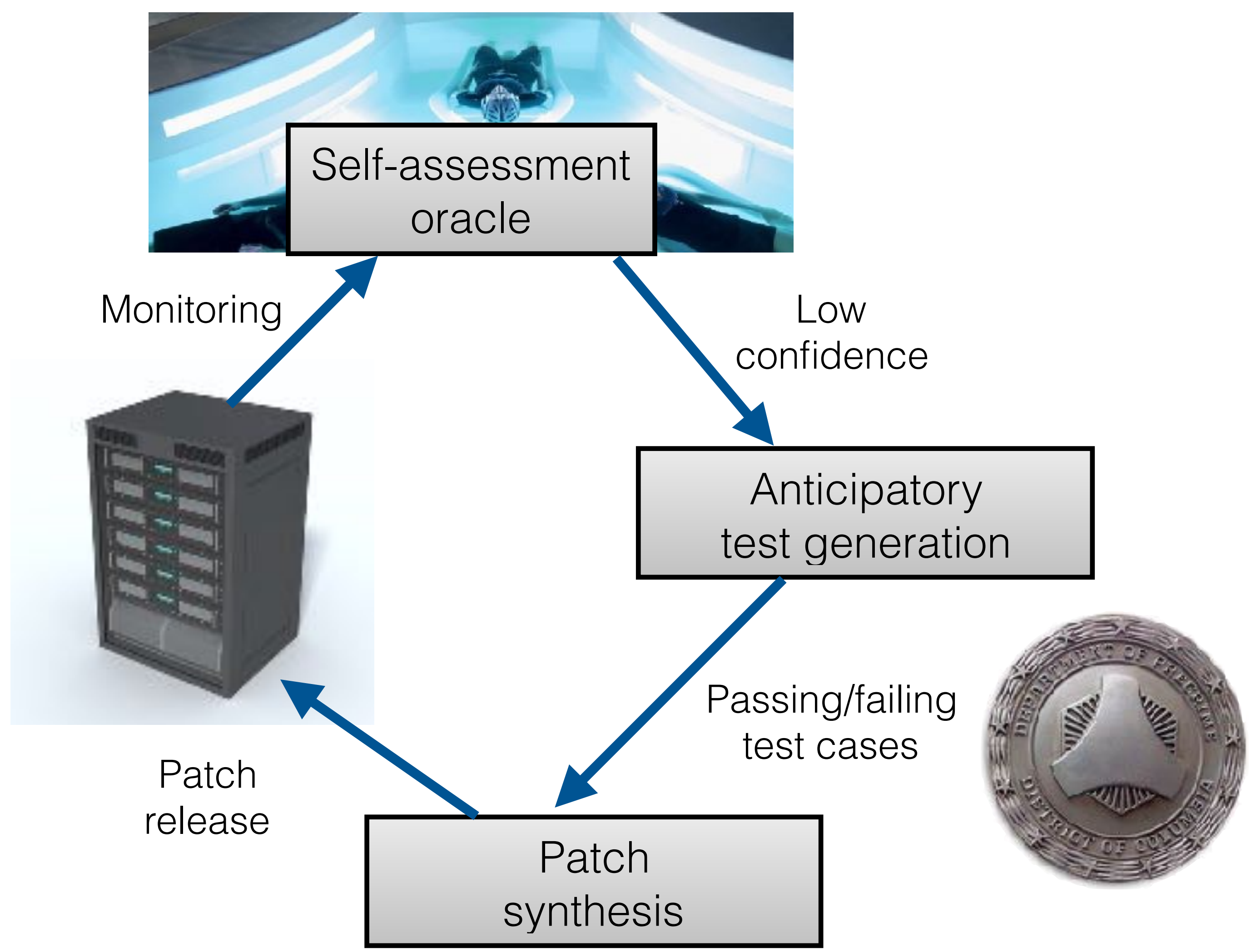
Precogs = <u>self-assessment oracle</u>
Precrime agency = <u>anticipatory testing & patch synthesis</u>



Philip K. Dick. *The Minority Report*. Fantastic Universe, 1956
Minority Report. Directed by Steven Spielberg; featuring Tom Cruise, Colin Farrell, Samantha Morton, Max von Sydow, 2002

Goal

PRECRIME aims at preventing the occurrence of failures in unexpected execution contexts by identifying new, possibly error prone, contexts



Challenges

Challenge 1 (Self-assessment oracle): How *to estimate the system's confidence even before correctness of execution can be evaluated?*

Challenge 2 (Context aware test case generation): How *to define a novel, context aware test adequacy criterion, based on a model of the evolving execution context, so that test cases derived from such a model achieve high execution confidence?*

Challenge 3 (Property adaptation): *How to assess the fault detection capability of currently available properties when used in a newly identified context (or state) and how to adapt them so that they can effectively detect misbehaviours of the system in a new context?*

Challenge 4 (Patch synthesis): *How to synthesize a dependable patch that can bring the system to a high confidence?*

Ideas

PRECIME will make use of:

- **Self-assessment oracle.** *A self-assessment oracle is an estimator of the system's confidence in being able to handle a given execution context correctly.*

  confidence = (1 – novelty) • (1 – failure probability)

- **Anticipatory test generator.** *Anticipatory testing aims at the creation of test cases that target a new execution context for which the self-assessment oracle reports a low confidence level.*

Objectives

Objective 1 (self-assessment oracle): *Define a confidence metric, to measure the confidence of the system in handling a new execution context, and create a self–assessment oracle that can measure confidence.*

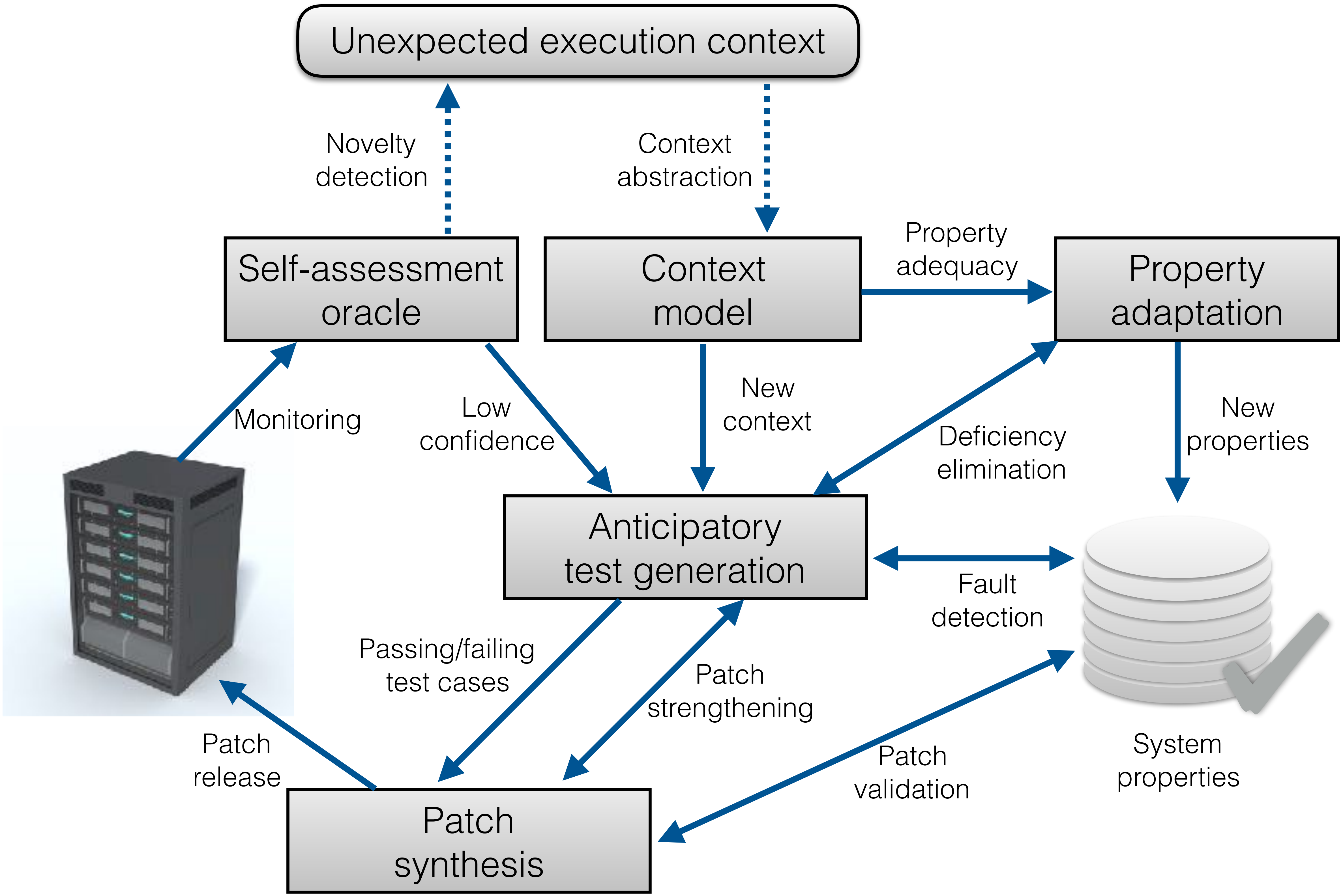
Objective 2 (context model): *Abstract a new execution context and system state into a data and behavioural model that can be compared against previously executed/tested context models.*

Objective 3 (context-aware test generator): *Generate new test cases focused on the inadequately tested aspects of a new execution context.*

Objective 4 (property adaptation): *Identify deficiencies in available system properties when applied to a new execution context and determine candidate adaptations that make such system properties suitable to act as functional correctness assertions in the new context.*

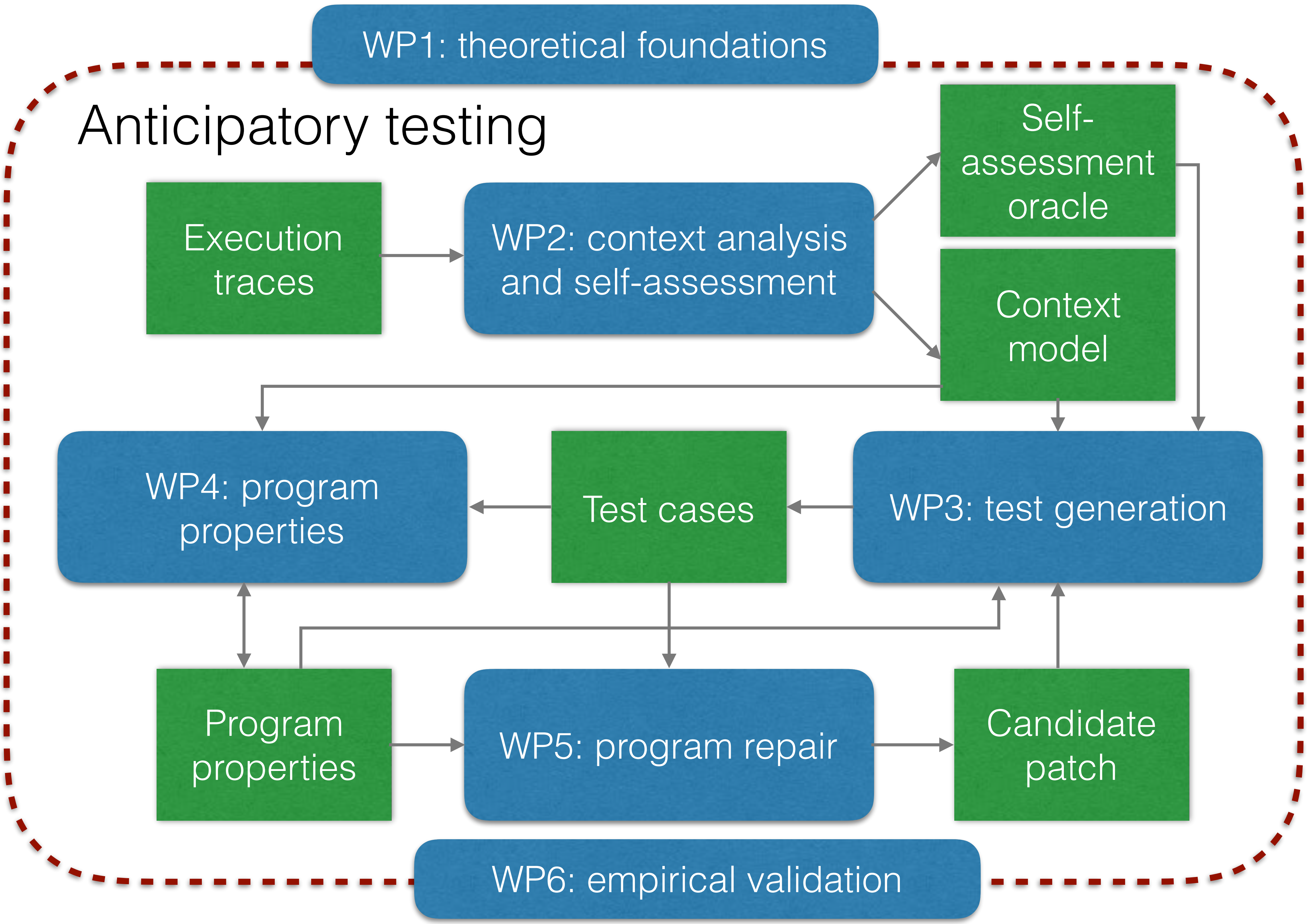
Objective 5 (patch synthesis): *Synthesize a candidate patch for a fault exposed by anticipatory testing.*

Approach



Unexpected execution context
Novelty detection
Context abstraction
Self-assessment oracle
Context model
Property adequacy
Property adaptation
Monitoring
Low confidence
New context
Deficiency elimination
New properties
Anticipatory test generation
Fault detection
Passing/failing test cases
Patch strengthening
Patch release
Patch validation
System properties
Patch synthesis

Workpackages



Building blocks

- **Statistics and information theory** (theoretical framework of self-assessment oracles)
- **Model inference** (context modeling)
- **Machine learning** (confidence estimation)
- **Evolutionary testing** (test generation)
- **Constraint solving** (test generation)
- **Oracle quality** (property adaptation)
- **Genetic programming, program transformation** (patch synthesis)
- **Fault localization, symbolic execution** (patch constraints)

Oracle quality

Gunel Jahangirova, David Clark, Mark Harman, Paolo Tonella. ***Test oracle assessment and improvement***. ISSTA 2016: 247-258

Oracle deficiencies

```java
public class Subtract {
    public double value(double x, double y) {
        double result = x-y;
         assert (result != x);
         assert (result == x-y);
         return result;
    }
}
```

False alarm

```java
public class FastMath {
    public int max(int a, int b) {
        int max;
        if (a >= b) {
            max = a;
        } else {
            max = b; // max = a;
        }
        assert (max >= a);
        return max;
```

Missed fault

Oracles may be too strong (false alarms) or too weak (missed faults)

False positives and false negatives

```
public class Subtract {
    public double value(double x, double y) {
        double result = x-y;
         assert (result != x);
         assert (result == x-y);
         return result;
   }
}
```

TC=(0, 0)

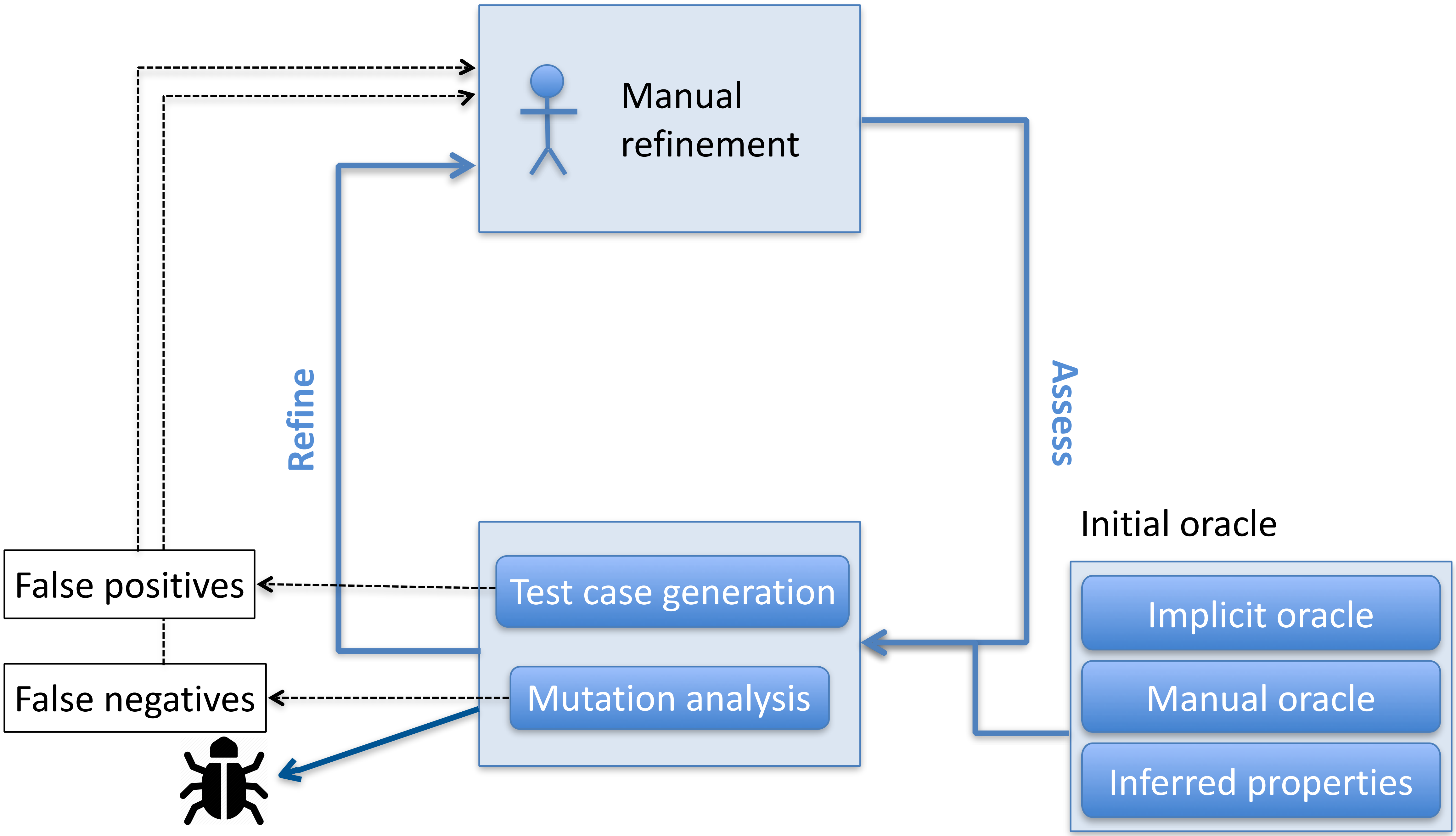
False positive: *program state where the assertion fails, although such state respects the intended program behaviour*

```
public class FastMath {
    public int max(int a, int b) {
        int max;
        if (a >= b) {
            max = a;
        } else {
            max = b; // max = a;
        }
        assert (max >= a);
        return max;
    }
}
```

TC=(0, 1)

False negative: *program state where the assertion passes, although such state violates the intended program behaviour*

Oracle improvement process



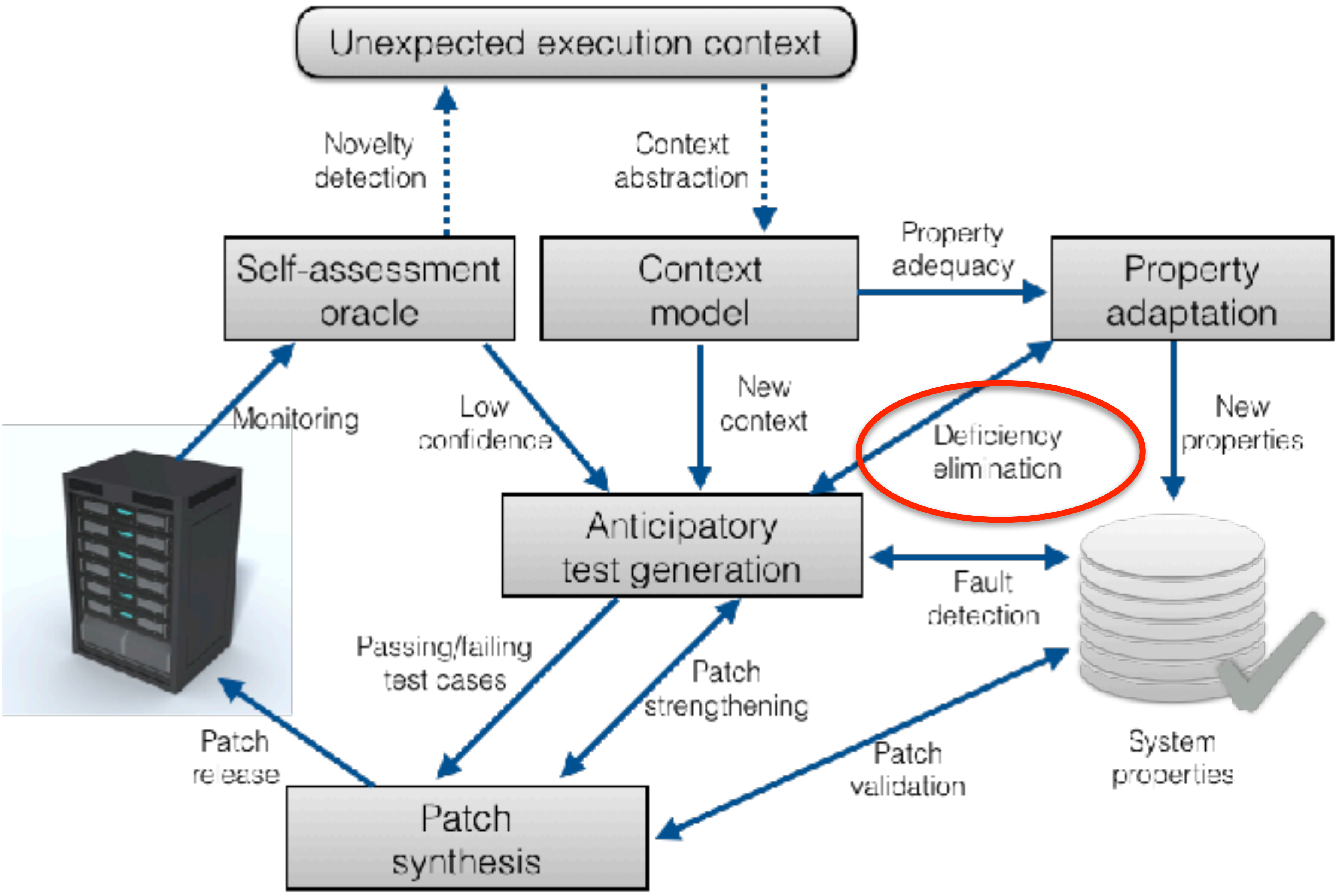
Oracle deficiency detection

```java
public class Subtract {
    public double value(double x, double y) {
        double result = x - y;
        assert (result != x);
        assert (result == x - y);
        return result;
    }
}

@Test
public void test1() throws Throwable {
    Subtract subtract0 = new Subtract();
    try {
        subtract0.value(0.0, 0.0);
        fail();
    } catch (Exception e) {
    }
}
```

```java
public class FastMath {
    public int max(int a, int b) {
        int max;
        if (a >= b) {
            max = a;
        } else {
            max = b; // max = a;
        }
        assert (max >= a); assert (max >= b);
        return max;
    }
}

@Test
/* Strong mutation L:5: // max = a; */
public void test2() throws Throwable {
    FastMath fastmath0 = new FastMath();
    int int0 = fastmath0.max(0, 1);
    int orig0 = 1;
    assertEquals(int0, orig0);
}
```

Use in Precrime

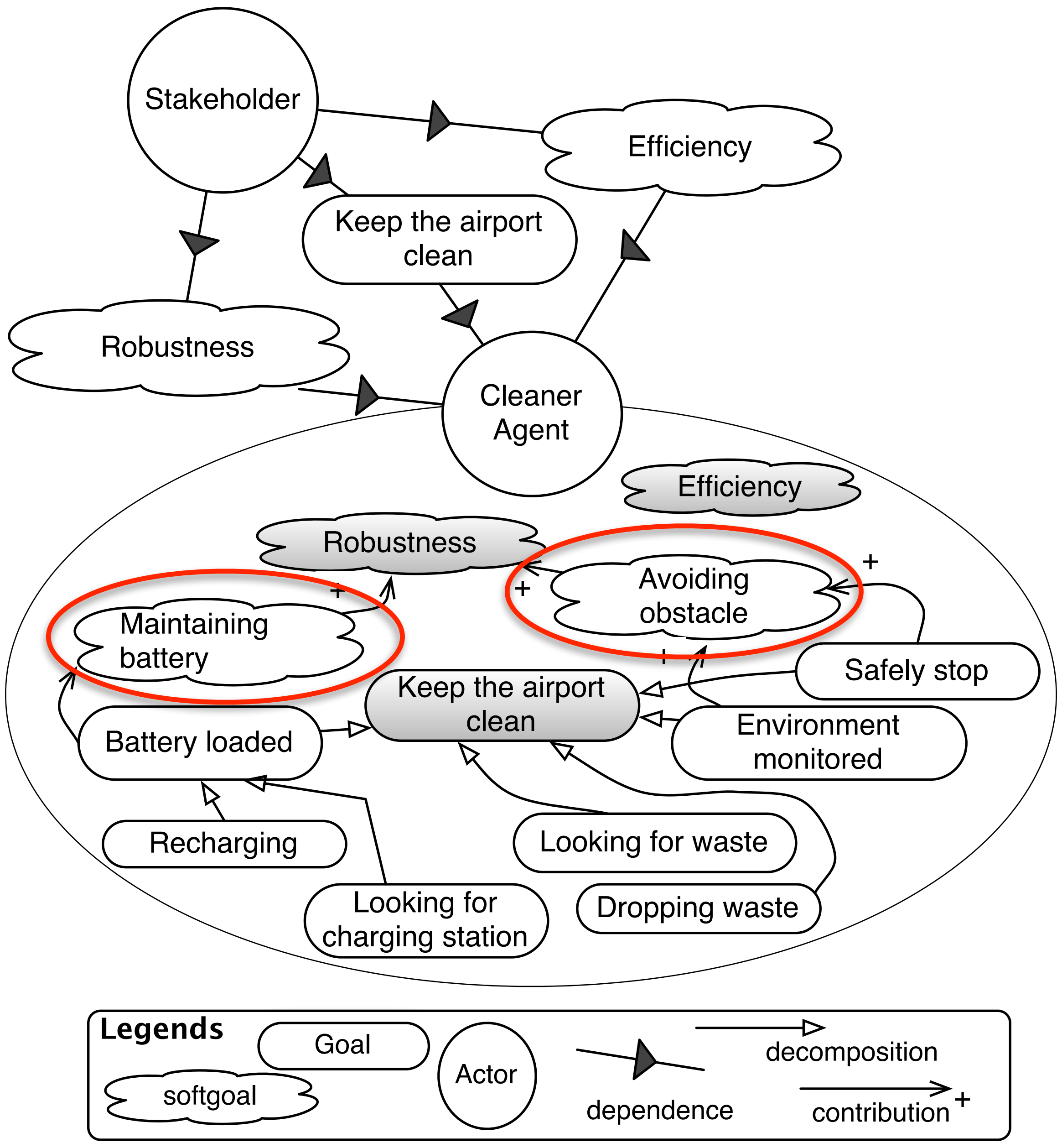
Objective 4 (property adaptation): *Identify **deficiencies** in available system properties when applied to a new execution context and determine candidate adaptations that make such system properties suitable to act as functional correctness assertions in the new context.*



Evolutionary testing

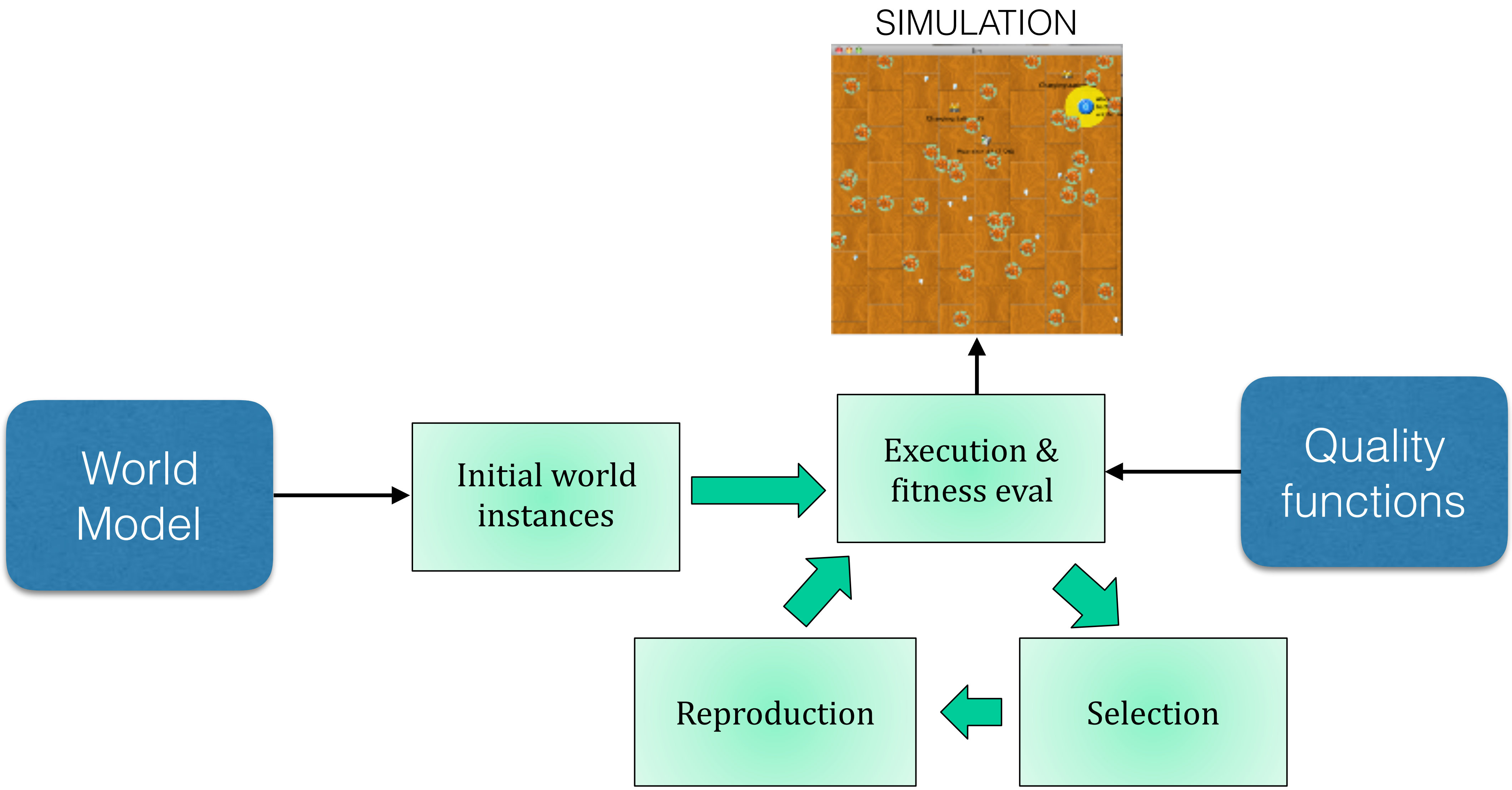
Cu D. Nguyen, Simon Miles, Anna Perini, Paolo Tonella, Mark Harman, Michael Luck: ***Evolutionary testing of autonomous software agents***. Journal of Autonomous Agents and Multi-Agent Systems, vol. 25, n. 2, pp. 260-283 2012.

Autonomous cleaner robot



- Quality functions are derived from the agent goals
  - Maintaining battery
  - Avoiding obstacle
- Evolutionary testing generates test scenarios that exhibit poor quality levels

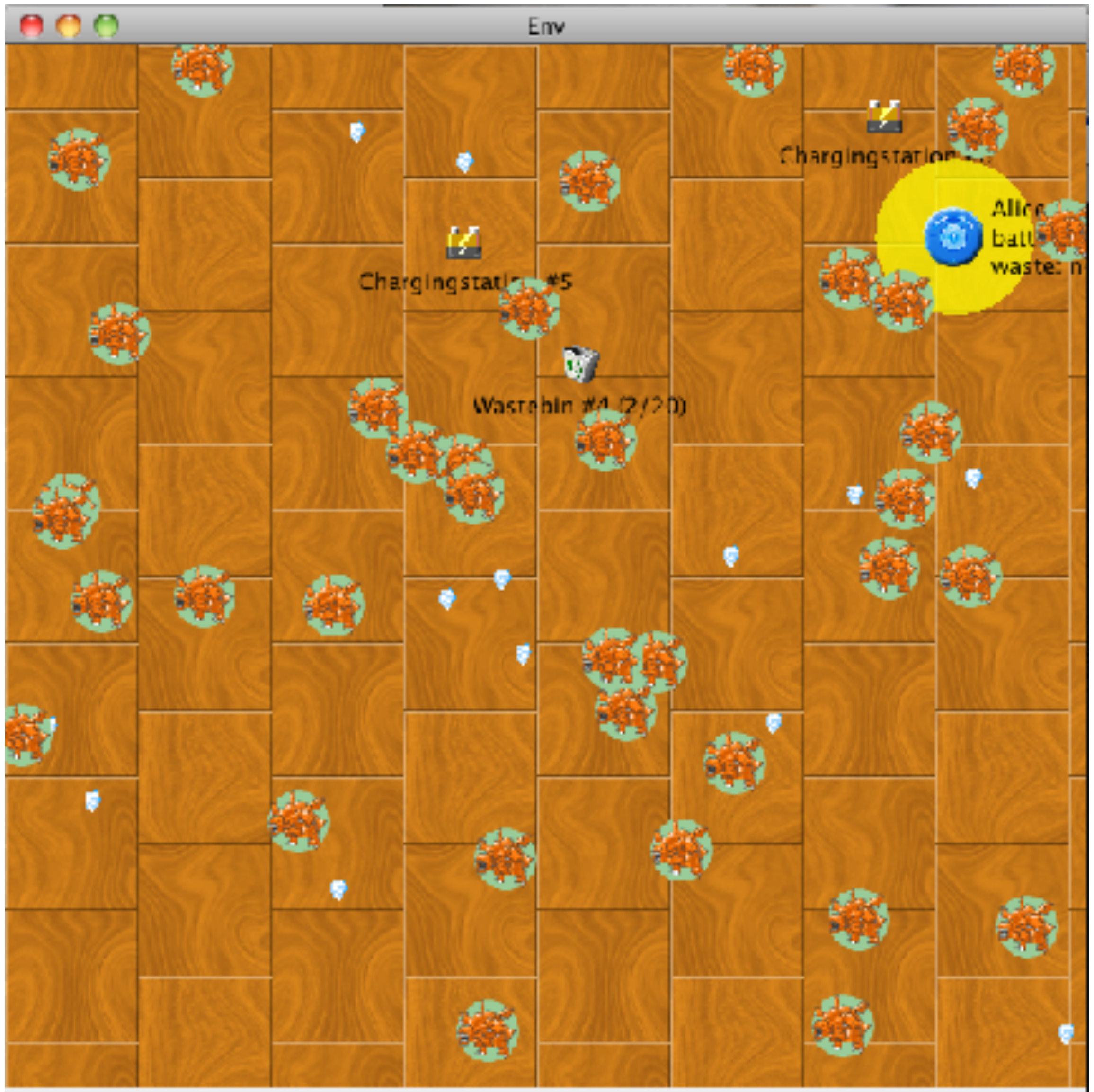
Evolutionary world creation



Single vs. multiple fitness functions:

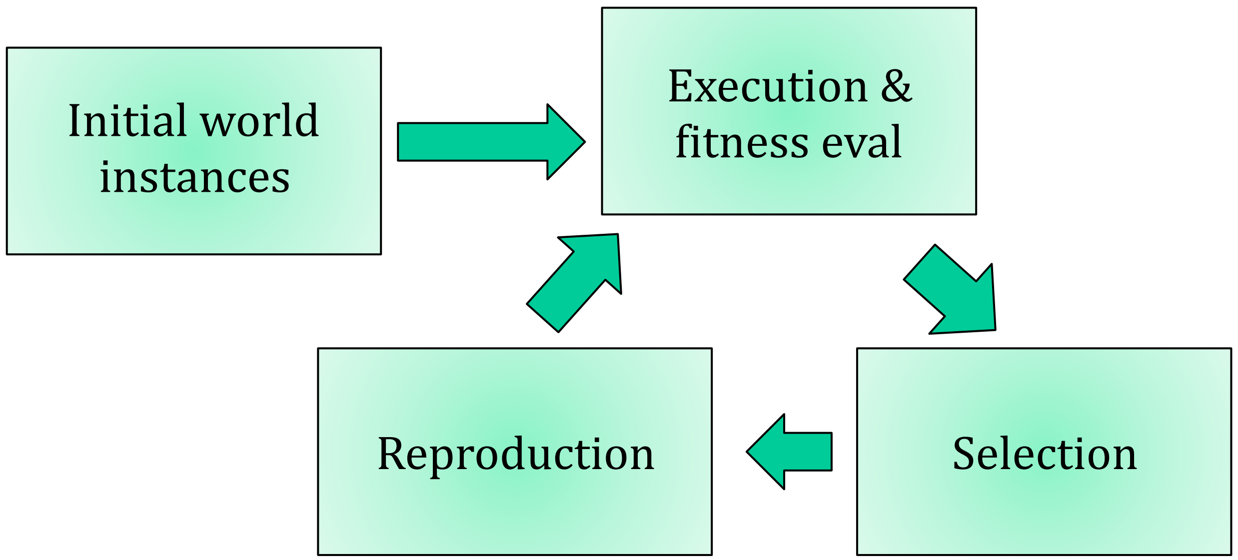
- *Maintain battery*: world instances where recharging is difficult
- *Avoid obstacles*: world instances where paths through obstacles are narrow and difficult to take
- *Maintain battery* and *Avoid obstacles*: world instances where low power consumption and obstacle avoidance conflict with each other

World model



Wastebin	$<x_1, y_1, x_2, y_2>$
Charging station	$<x_1, y_1, x_2, y_2>$
Obstacle	$<x_1, y_1, \ldots, x_{No}, y_{No}>$
Waste	$<x_1, y_1, \ldots, x_{Nw}, y_{Nw}>$

Evolutionary world creator



Population size	30
Max generations	100
Mutation probability	3%
Crossover probability	90%

Chromosome:

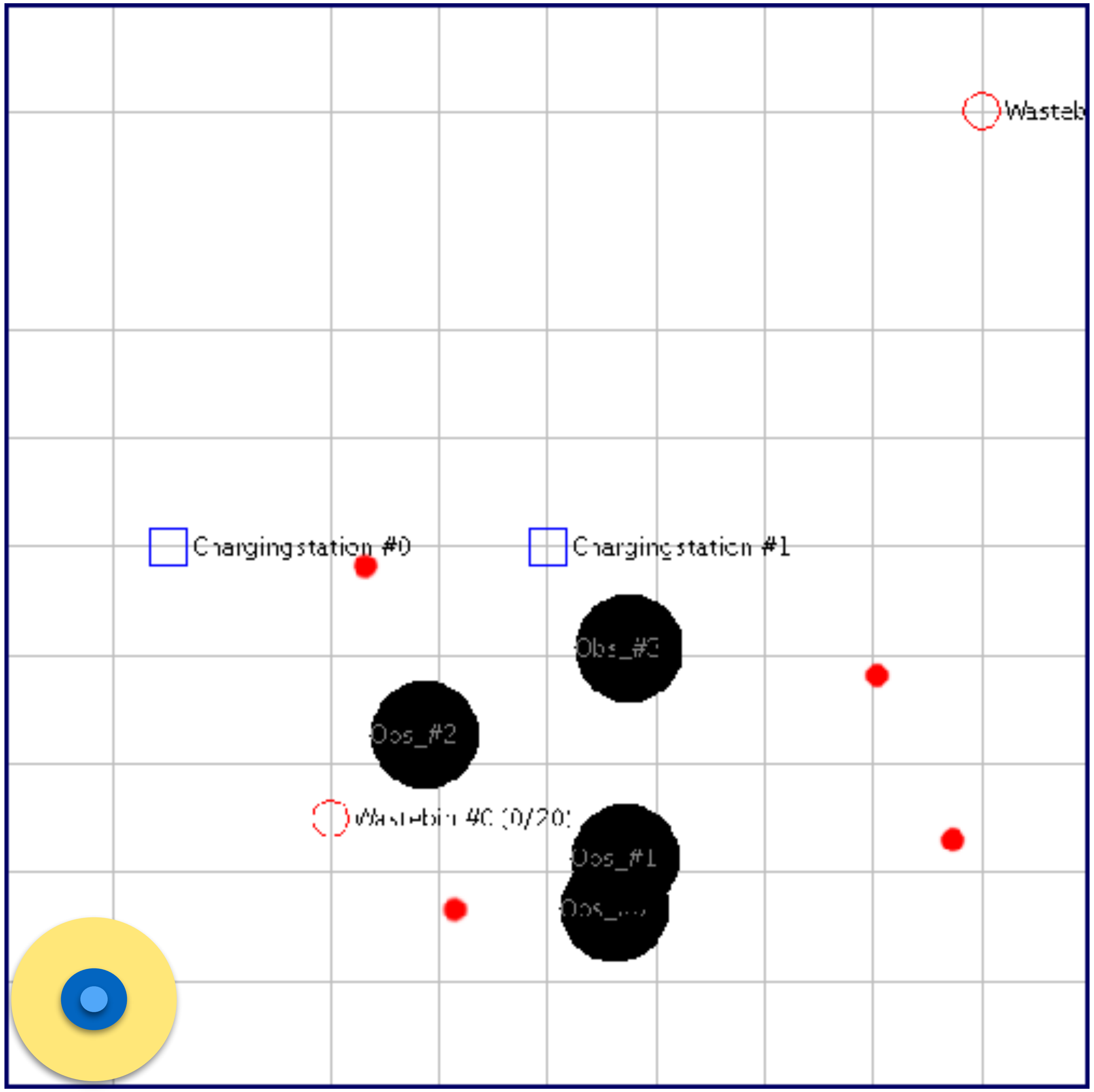
$\langle\langle x_1, y_1, x_2, y_2\rangle, \langle x_1, y_1, x_2, y_2\rangle, \langle x_1, y_1, \ldots, x_{No}, y_{No}\rangle, \langle x_1, y_1, \ldots, x_{Nw}, y_{Nw}\rangle\rangle$

Fitness functions:

f_{power} = 1 / Total power consumption

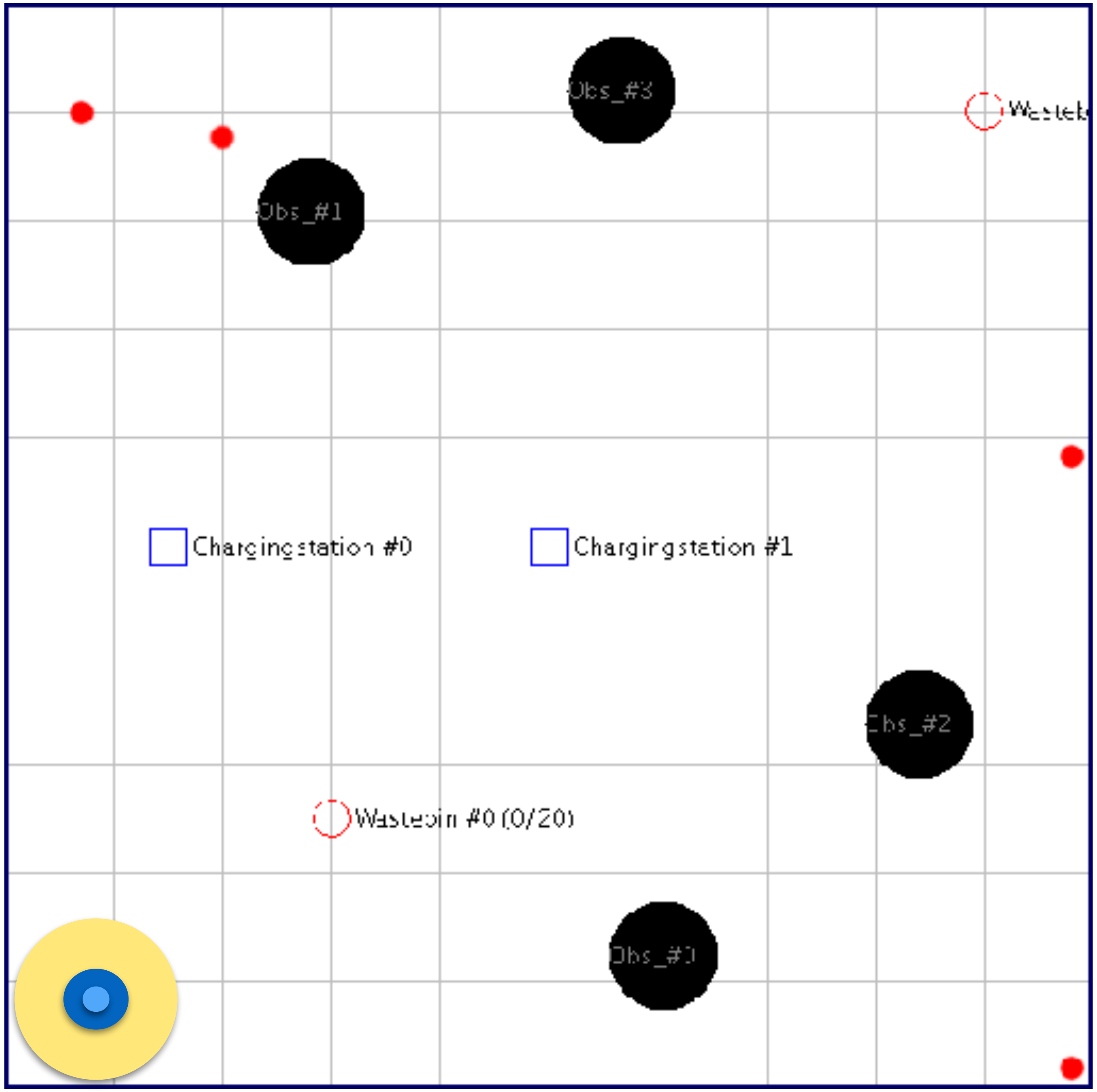
f_{obs} = 1 / Number of obstacles encountered

A world with min f_{obs}



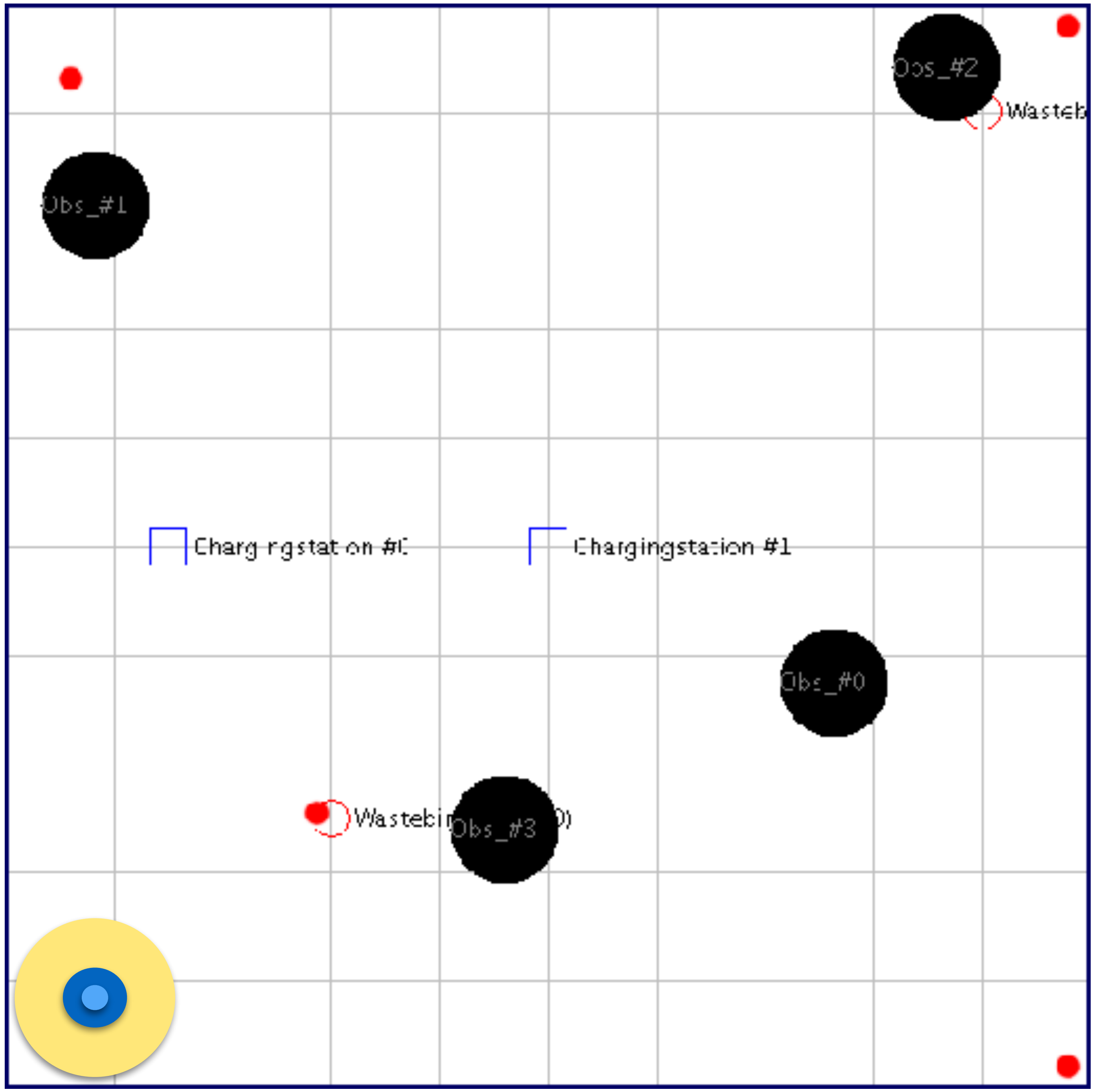
High probability of crashing into an obstacle after reaching wastebin #0.

A world with min f_{power}



Waste items near the corners increase battery consumption.

A world with min f_{obs}, f_{power}



Waste items near the corners increase battery consumption and obstacles along the paths to the waste items increase the chances of collisions

Real fault

```
<maintaingoal name="maintainbattery" retry="true" recur="true" retrydelay="0">
        <deliberation cardinality="-1">
                <inhibits ref="performlookforwaste" inhibit="when_in_process"/>
                <inhibits ref="achievecleanup" inhibit="when_in_process"/>
                <inhibits ref="achievepickupwaste" inhibit="when_in_process"/>
                <inhibits ref="achievedropwaste" inhibit="when_in_process"/>
                <!-- disable also the avoiding obstacle goal when battery is too low -->
                 <inhibits ref="avoidobstacles" inhibit="when_in_process">
                        $beliefbase.my_chargestate &lt; 0.03
                 </inhibits>
        </deliberation>
        <!-- engage in actions when the state is below MINIMUM_BATTERY_CHARGE. -->
        <maintaincondition>
                $beliefbase.my_chargestate > MyConstants.MINIMUM_BATTERY_CHARGE
        </maintaincondition>
        <!-- The goal is satisfied when the charge state is 1.0. -->
        <targetcondition>
                $beliefbase.my_chargestate == 1.0
        </targetcondition>
</maintaingoal>
```

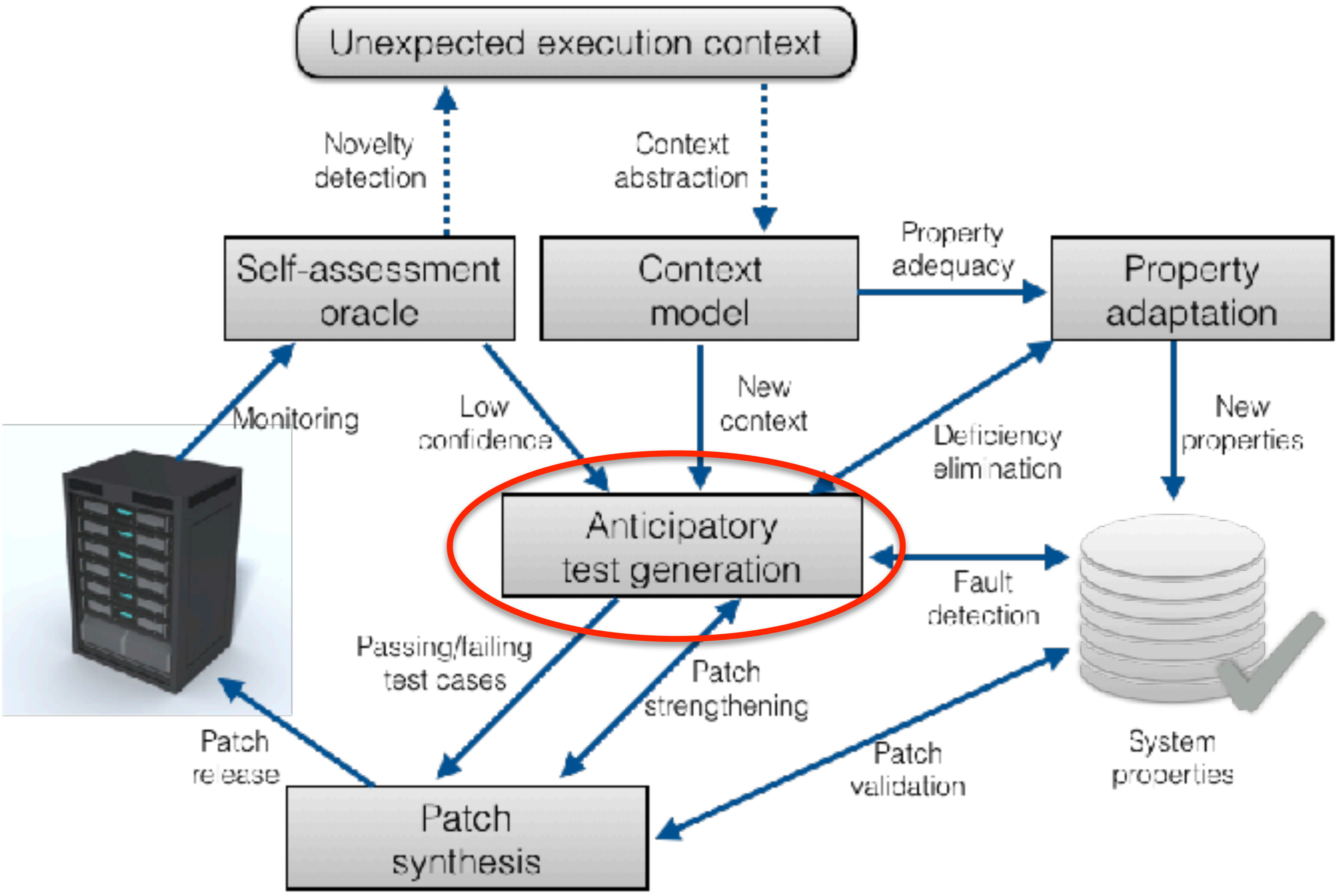
A **collision** happens when:

1. the battery level drops below 3%;
2. there are nearby obstacles in the path of the agent.

The chance of detecting this fault with a single objective fitness function is very low, while it is high when both quality functions are minimized.

Use in Precrime

Objective 3 (context-aware test generator): *Generate new test cases focused on the inadequately tested aspects of a new execution context.*



Conclusion

Self Assessment Oracles for Anticipatory Testing

Test the unexpected before it causes any failure…

http://pre-crime.eu

- Evolutionary and AI capabilities are granted to test generators, in order to make them capable of testing autonomous, AI based systems.
- The test oracle becomes an adaptive, live artifact, whose deficiencies are automatically detected and resolved.
- Self-assessment quantifies the likelihood of achievement of the AI system's goals in the given execution context.
- Online testing becomes a critical component of AI systems, due to the huge amount of execution contexts that cannot be exercised a priori.