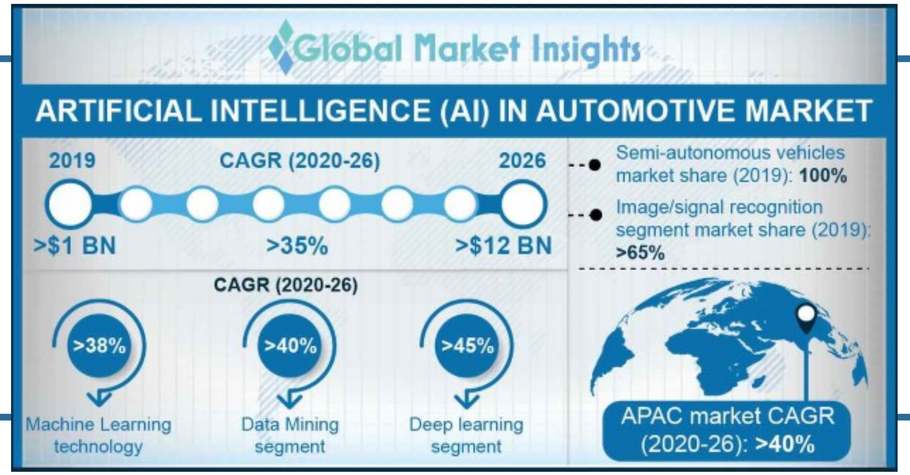
Learning Monitorable Operational Design Domains for Assured Autonomy

Hazem Torfah, Carol Xie, Sebastian Junges, Marcell Vazquez-Chanlatte, Sanjit A. Seshia

Berkeley
UNIVERSITY OF CALIFORNIA

Increasing AI in autonomy

- **Trend:**
  - Data-driven machine learning techniques are essential to perform perception tasks
  - The use of ML techniques is projected to grow



- **Growing concerns:**
  - Unpredictability hinders deployment and adoption in safety-critical applications
  - Neural models (e.g. DNNs) are brittle: unanticipated changes in the environment may cause faulty behavior

Tesla Crash 2020

The Tesla autopilot could not distinguish the side of the truck from the sky behind it!

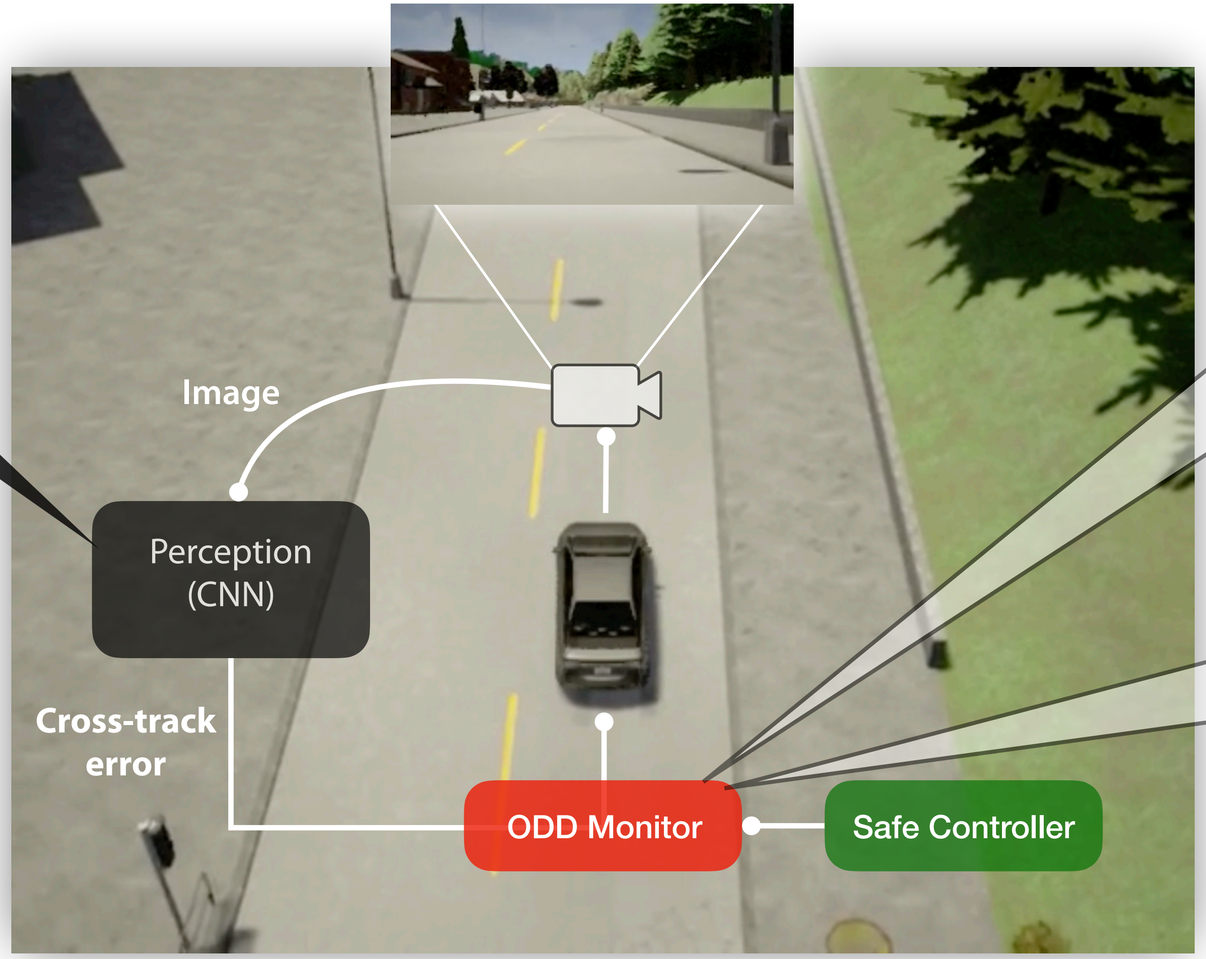
To raise the level of assurance it is crucial to

capture the conditions under which AI-based components maintain the safety of the system

Example: Image-based lane keeping

Black Box

- No access to implementation
- No access to the images used in training
- No knowledge about potential gaps in the training set



- Simplex-based architecture. Implements a switching logic between ML-based controller and safe controller

- Learn with respect to a system-level safety specification and over an observable set of features

Goal: Learn monitor that predicts violation of ODD and switches to a safe controller

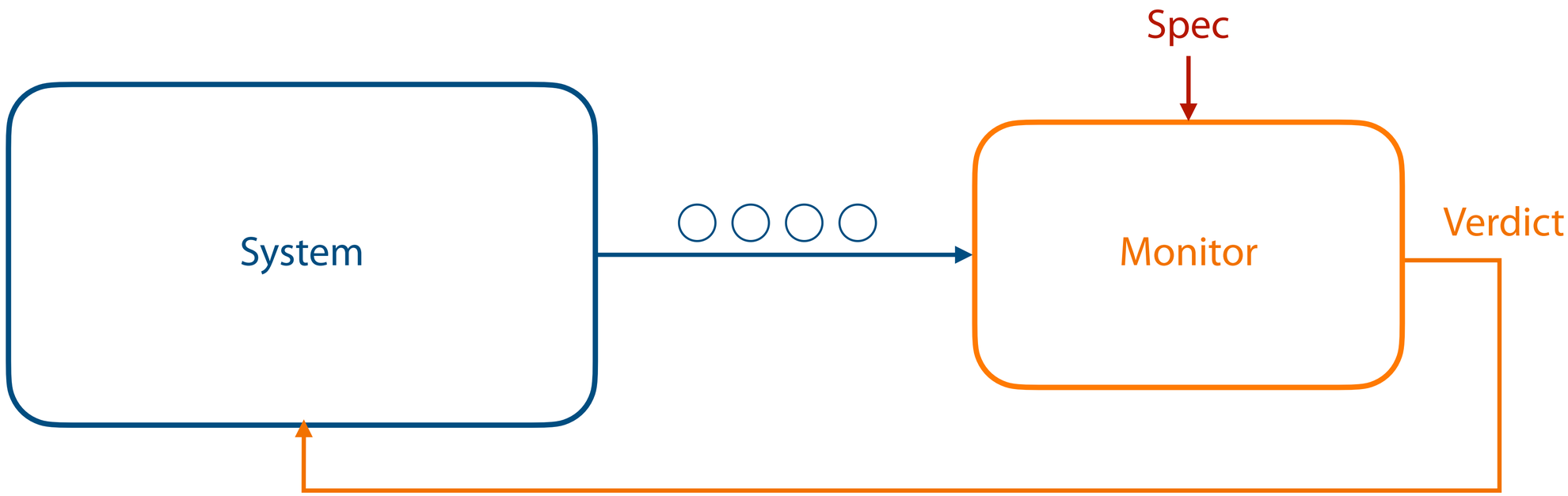
Example: Image-based lane keeping

No Monitor

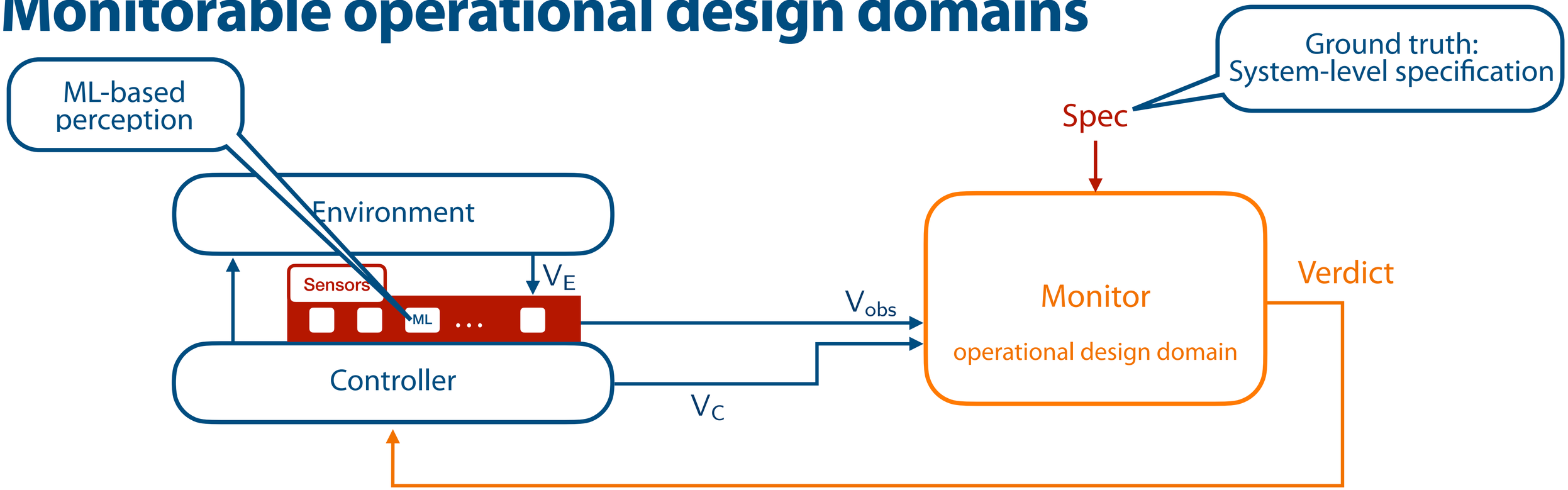
With Monitor

Monitor was able to mostly keep the vehicle in the lane by appropriately switching to the safe controller

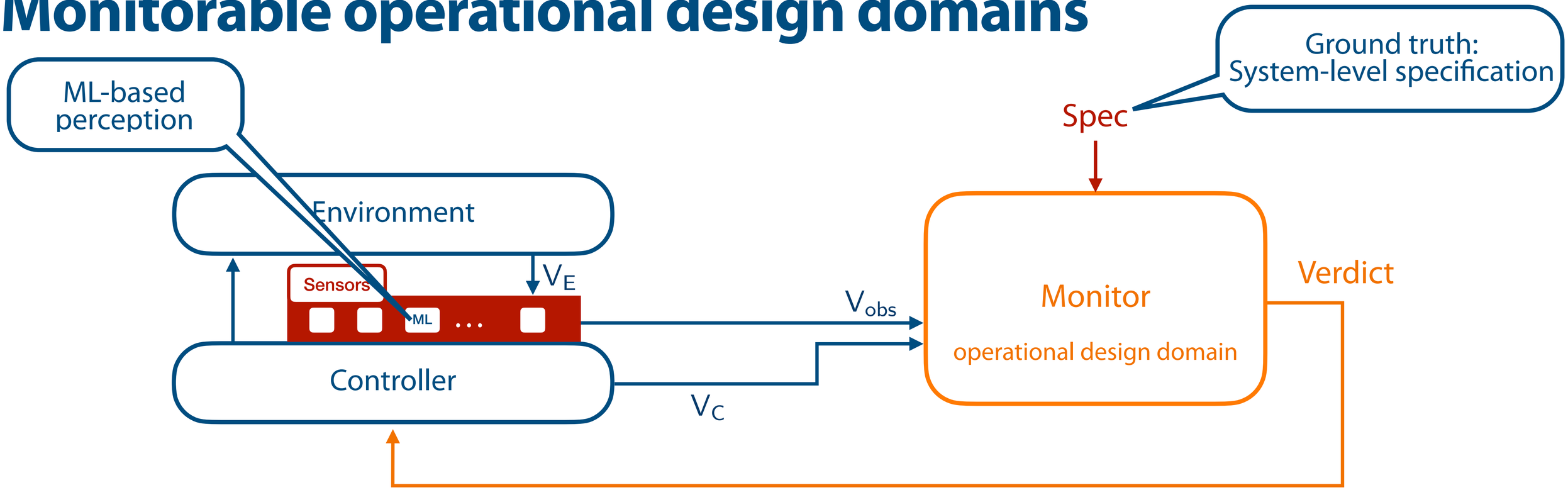
Monitorable operational design domains



Monitorable operational design domains

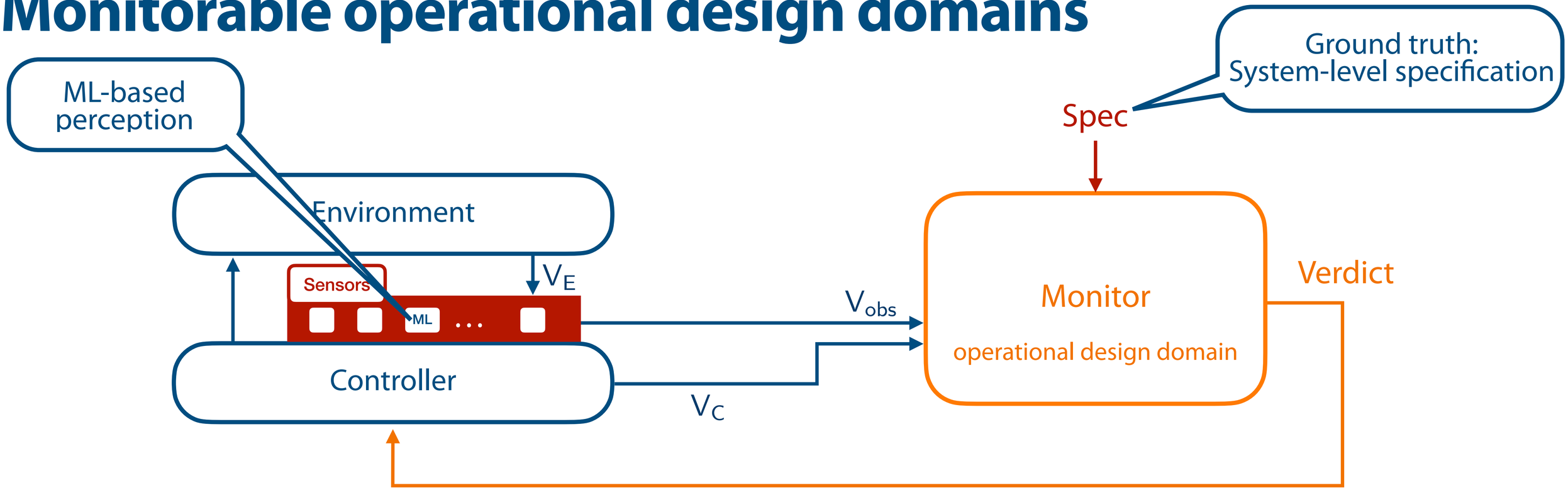


Monitorable operational design domains



$$D_{S,\varphi} = (V_{obs}, obs, d)$$

Monitorable operational design domains



$$D_{S,\varphi} = (V_{obs}, obs, d)$$

System under scrutiny
$S \subseteq V^*_{sys}$

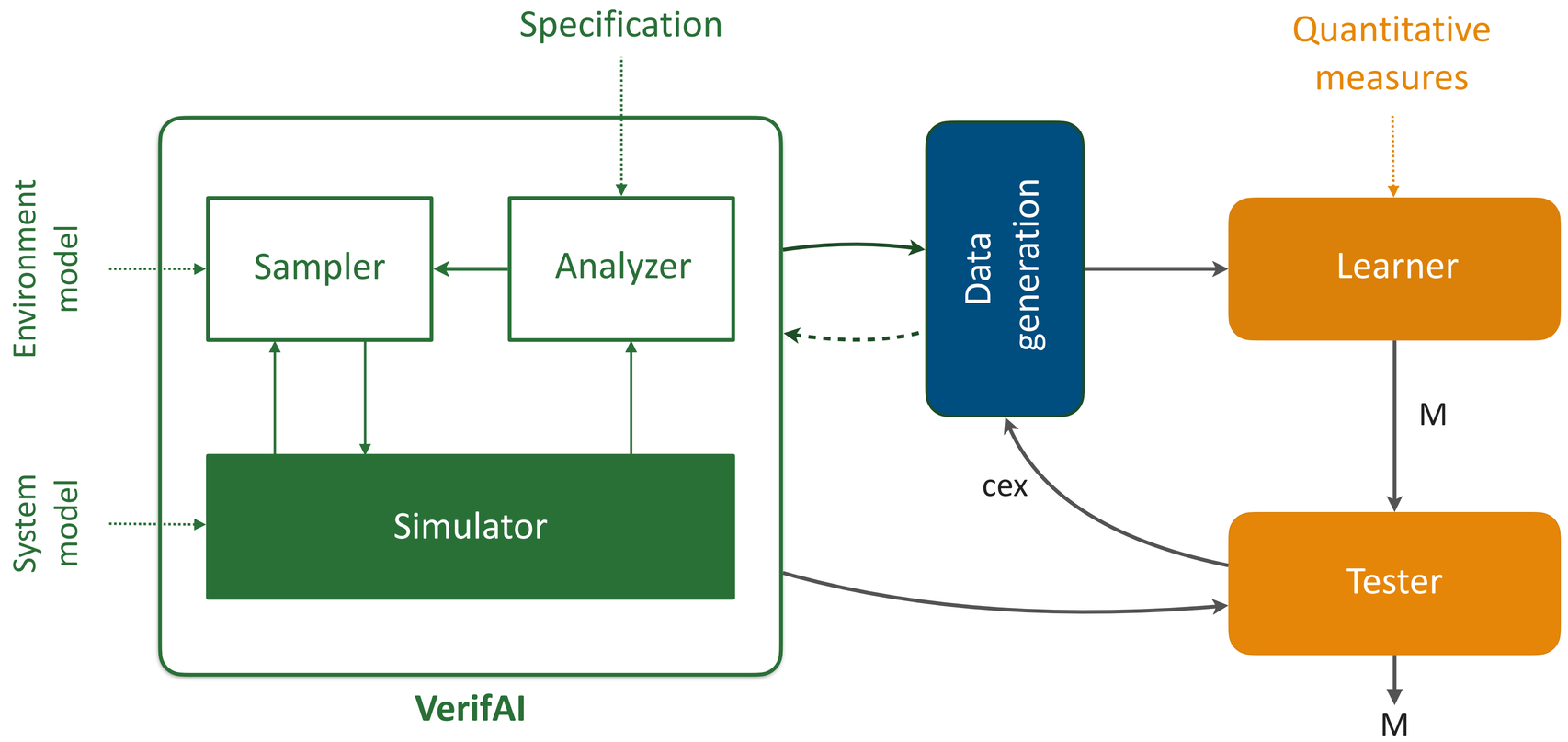
System-level specification
$\varphi \subseteq V^*_{sys}$

Observable monitor inputs

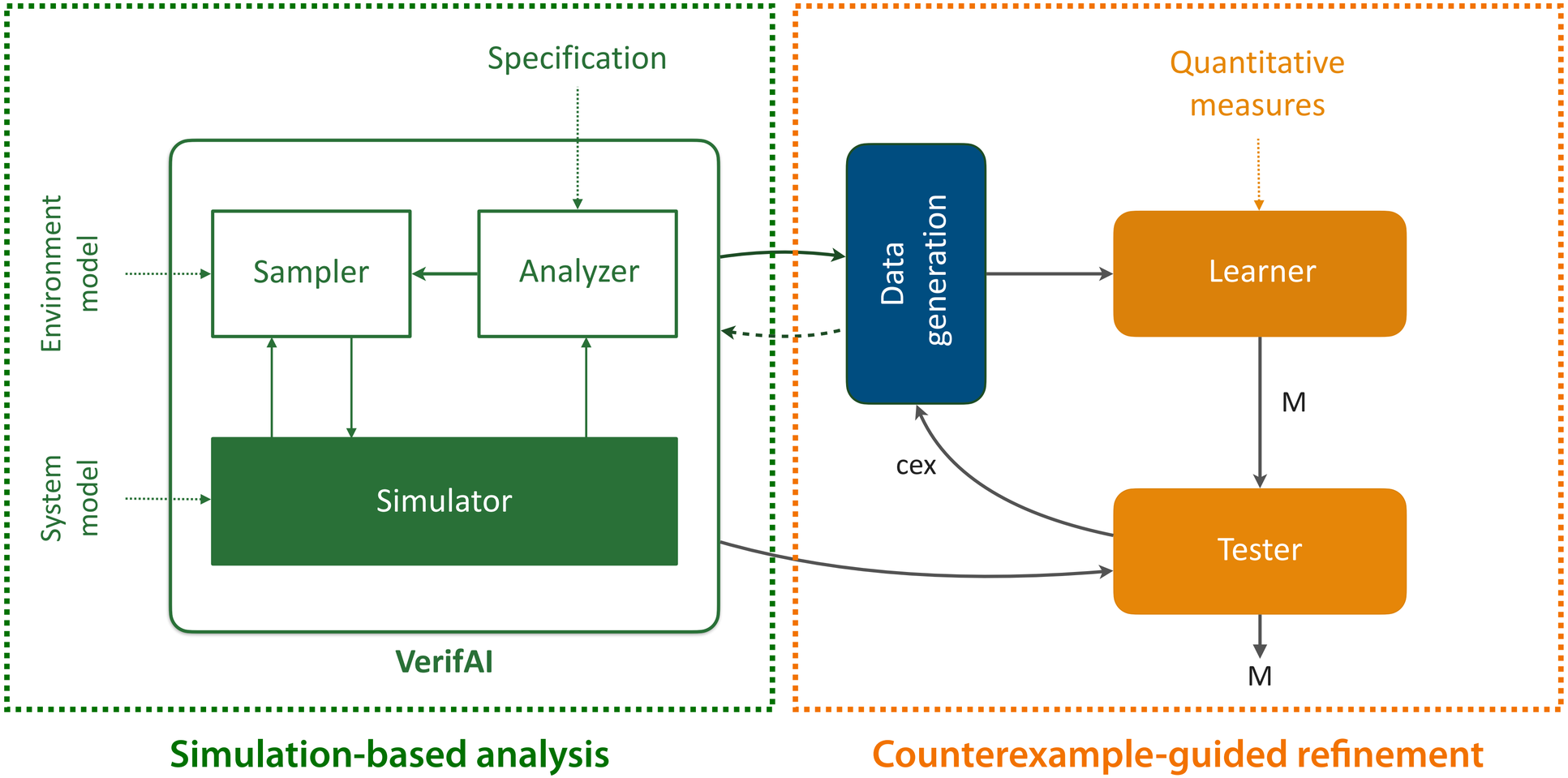
Mapping between system traces and observable traces
$obs\colon V^*_{sys} \to V^*_{obs}$

Prediction horizon

Counterexample-guided learning

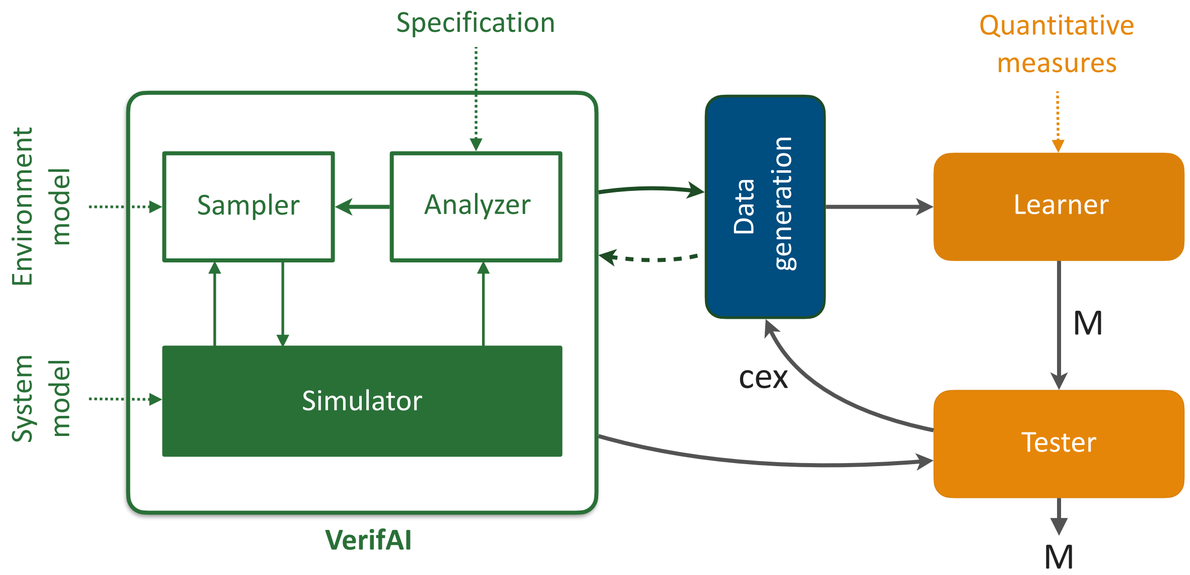


Counterexample-guided learning

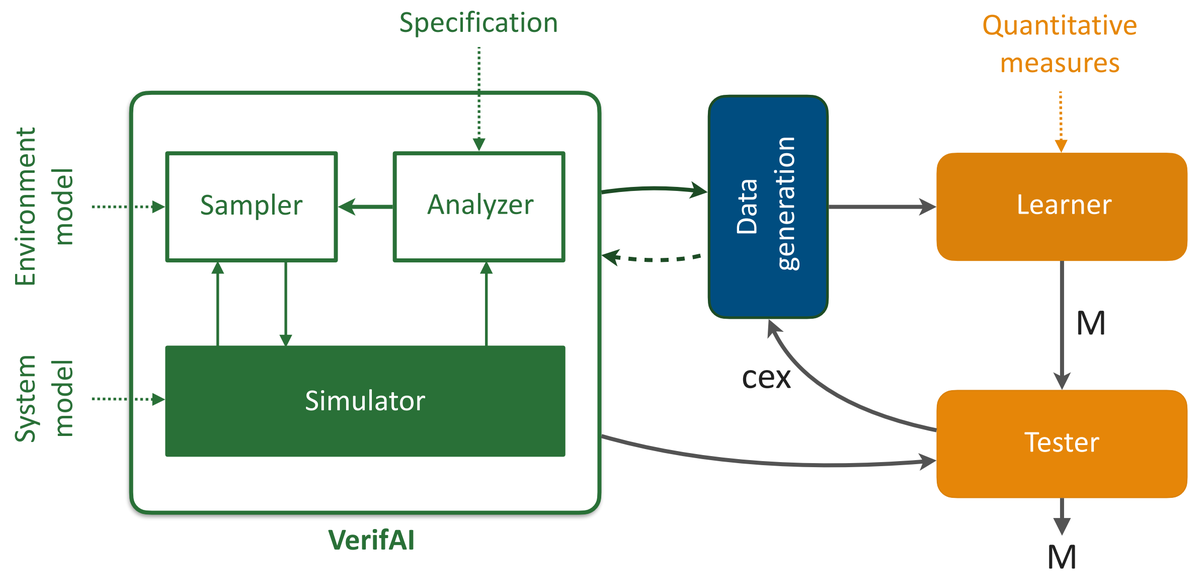


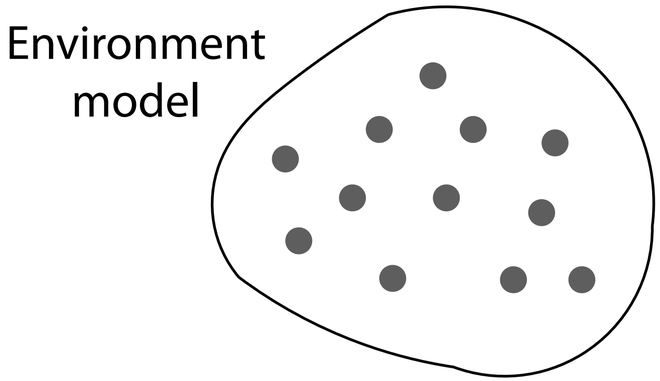
Dreossi et al. VerifAI: A Toolkit for the Formal Design and Analysis of AI-Based Systems. CAV 2019

Counterexample-guided learning

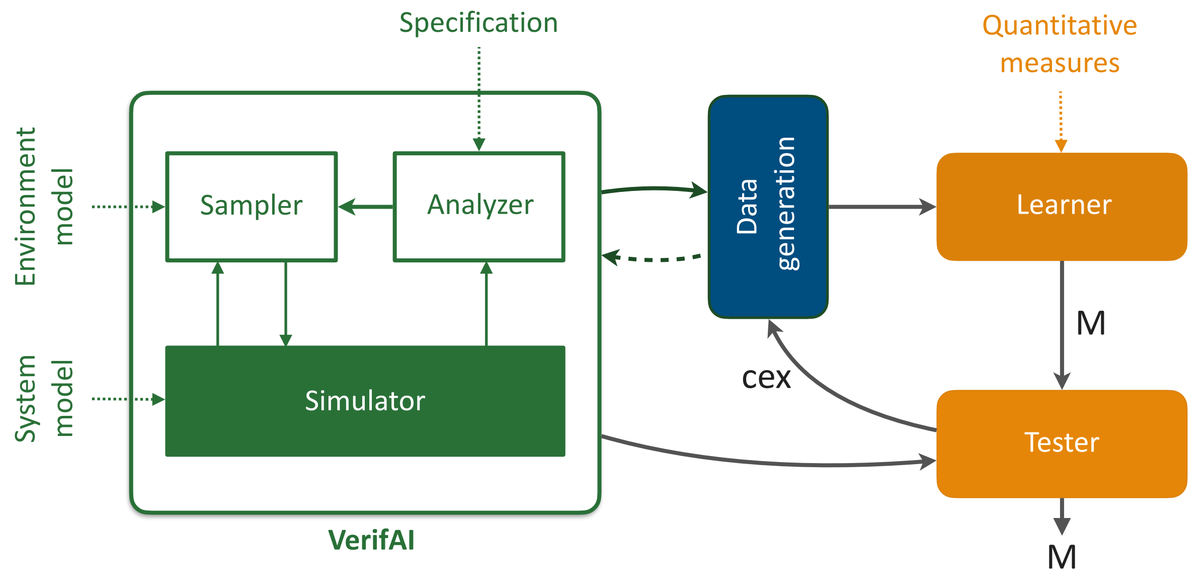


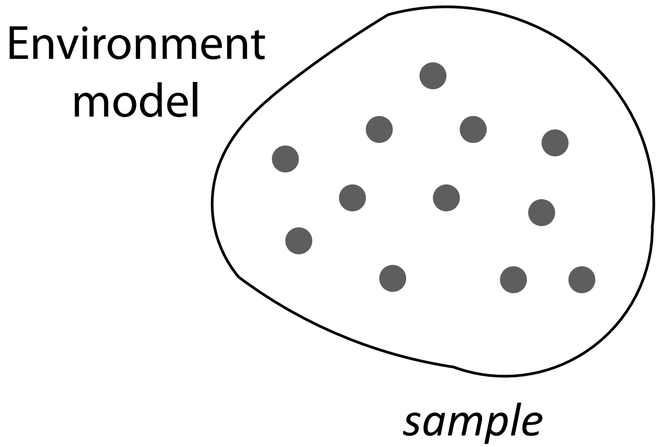
Counterexample-guided learning



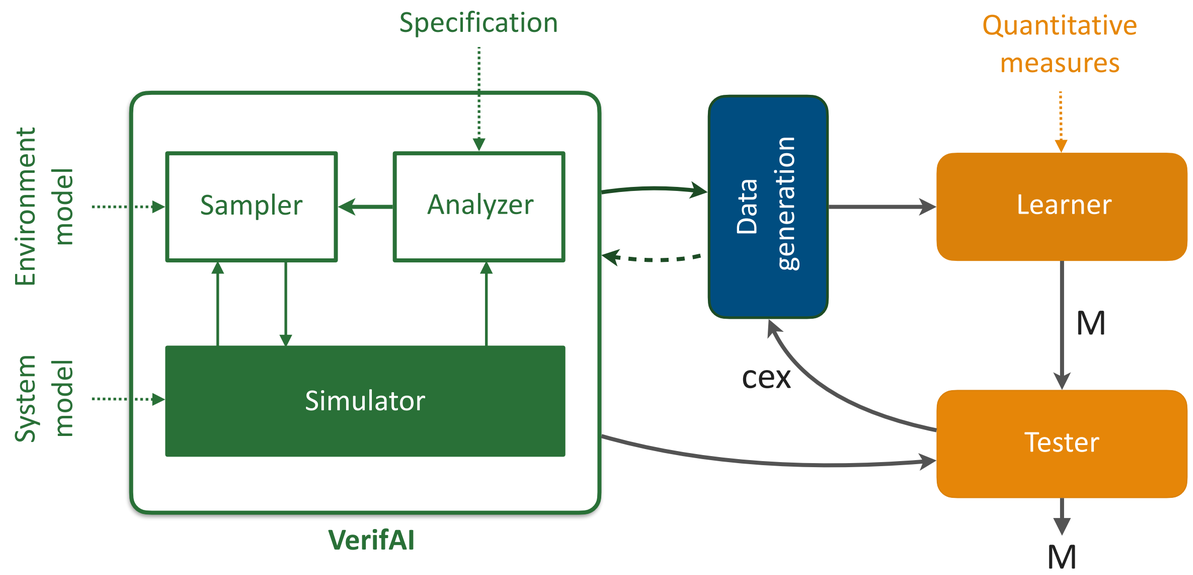


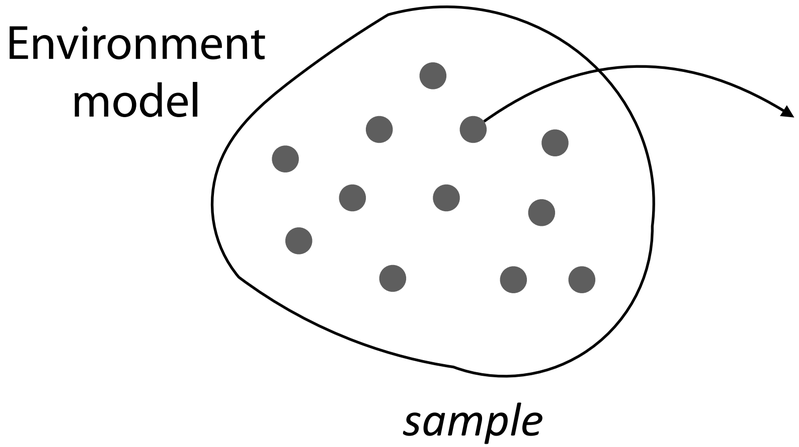
Counterexample-guided learning



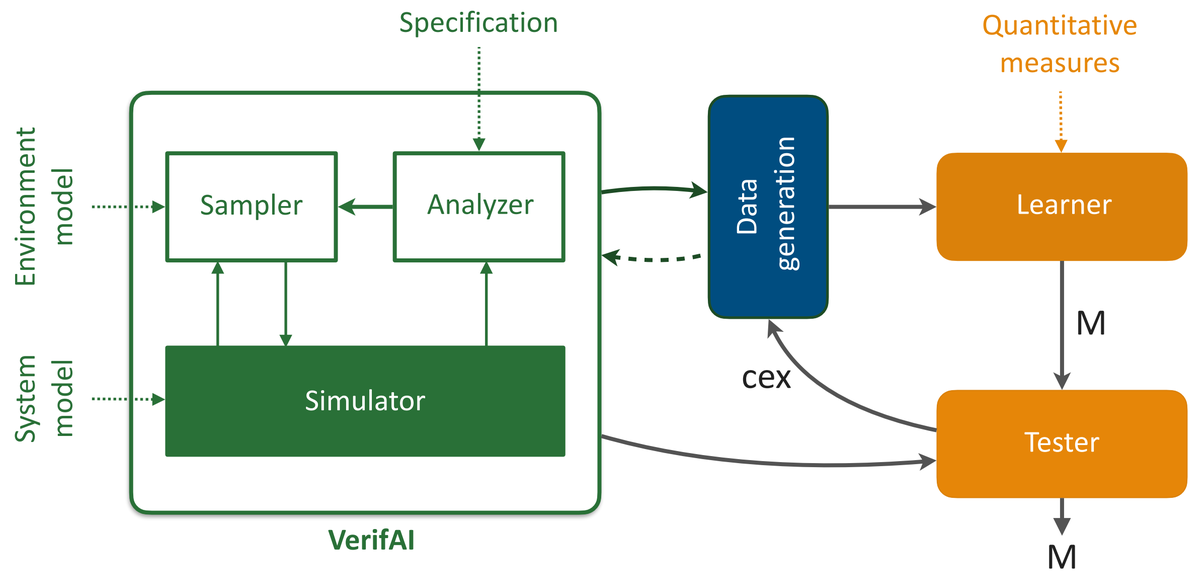


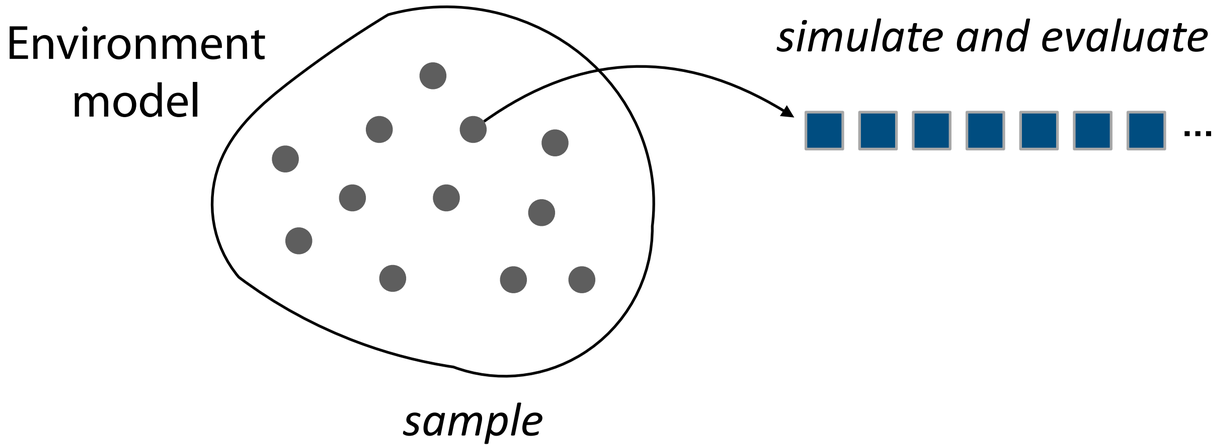
Counterexample-guided learning



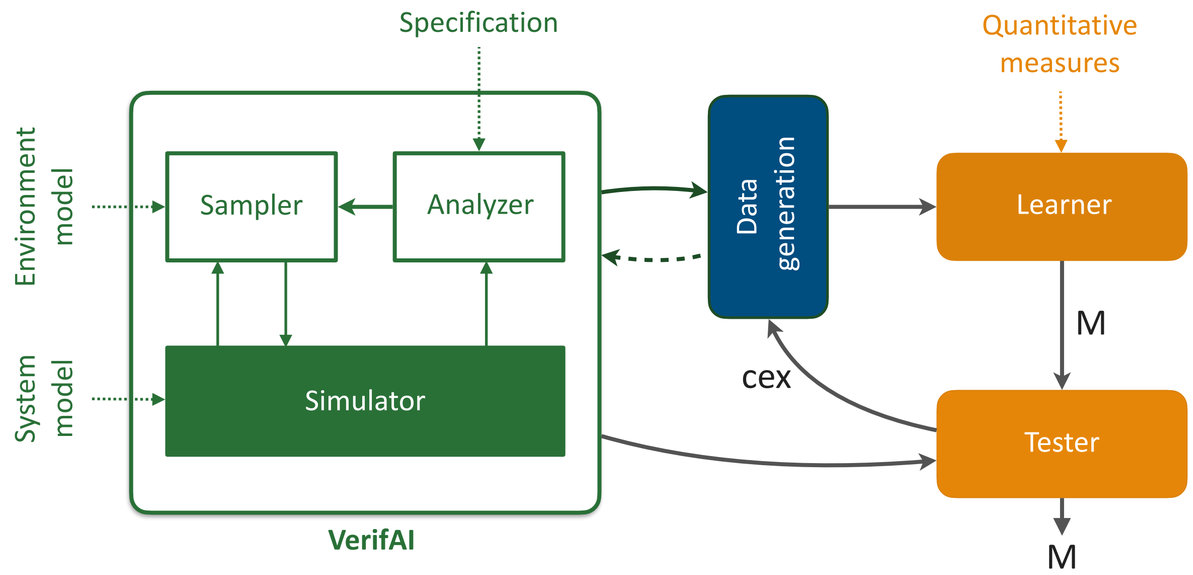


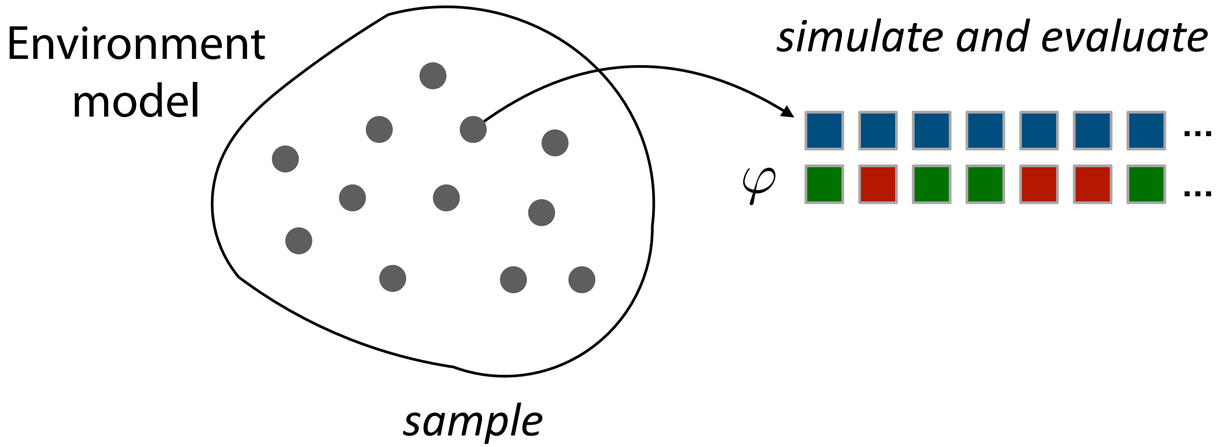
Counterexample-guided learning



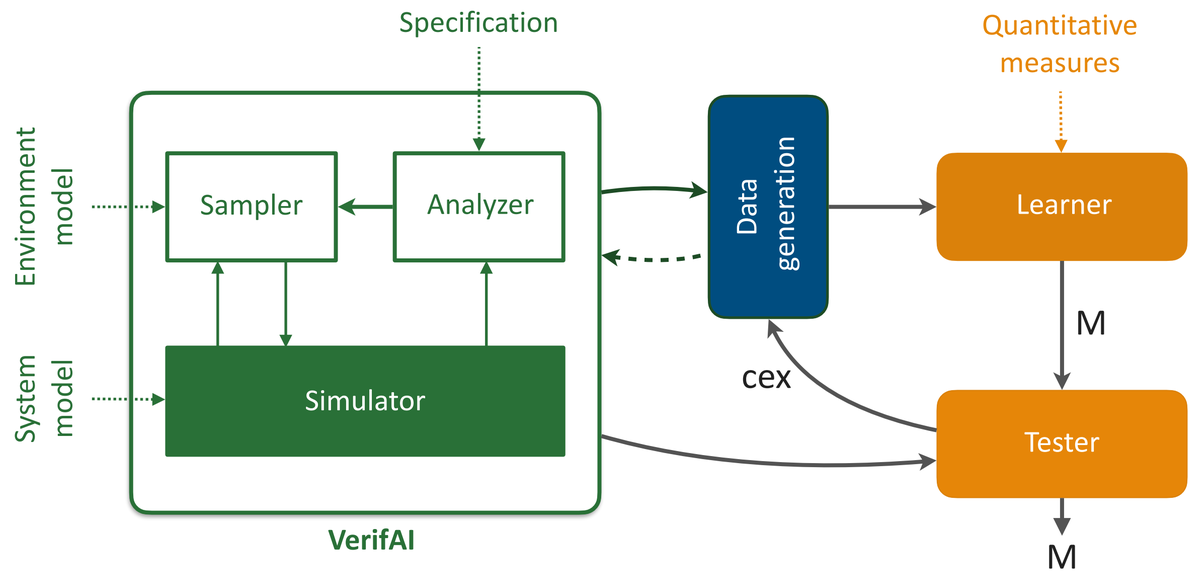


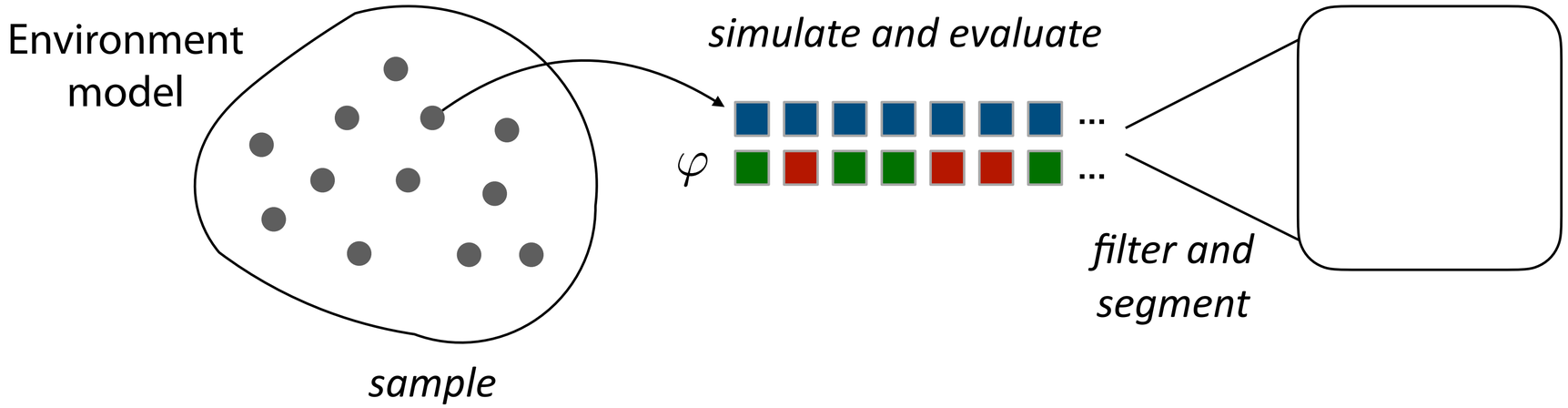
Counterexample-guided learning



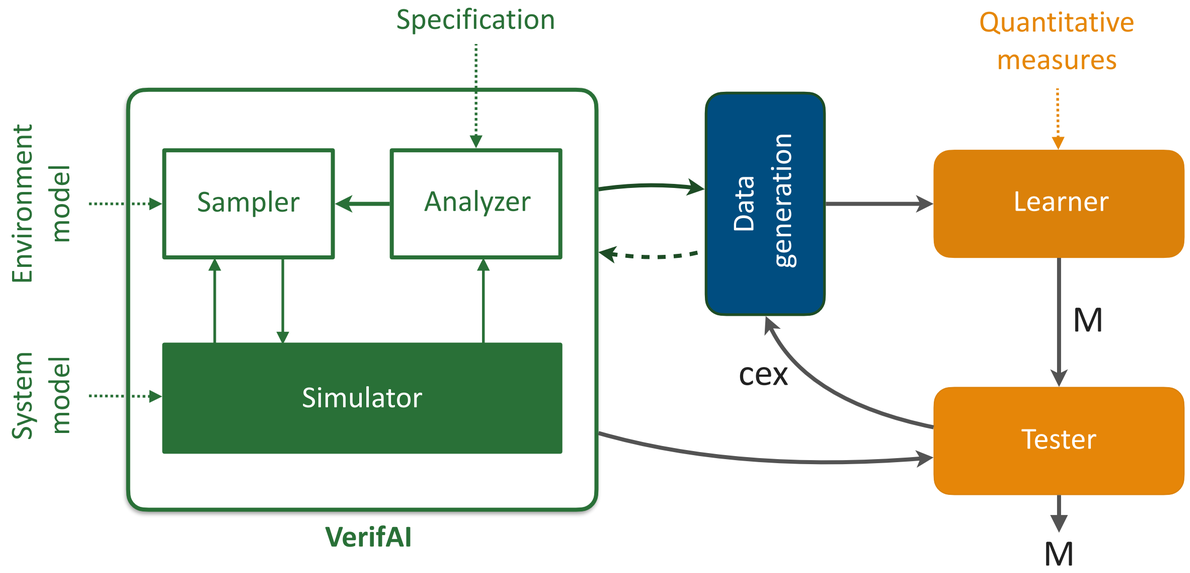


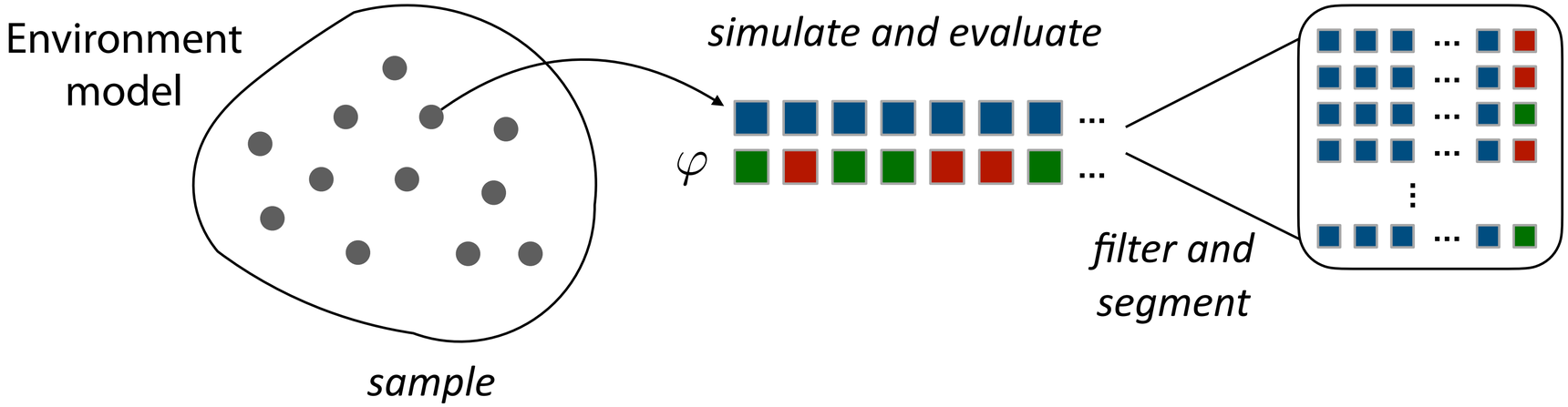
Counterexample-guided learning



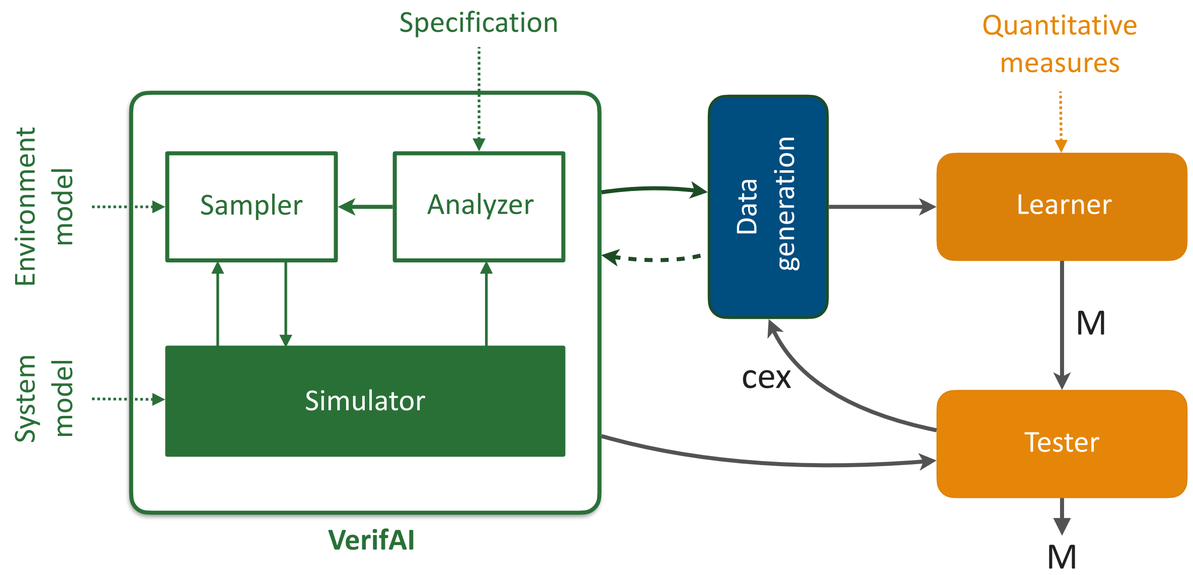


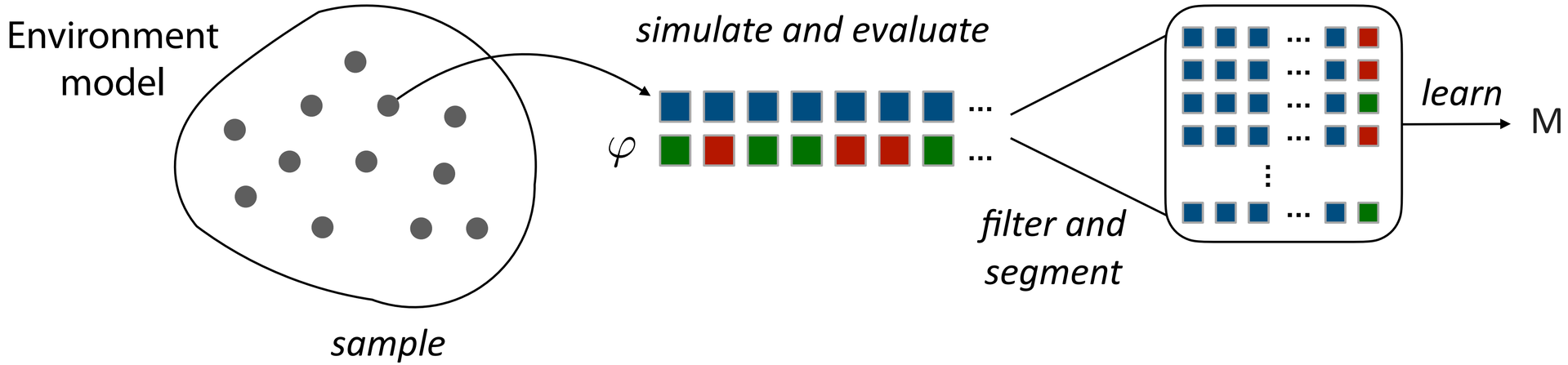
Counterexample-guided learning



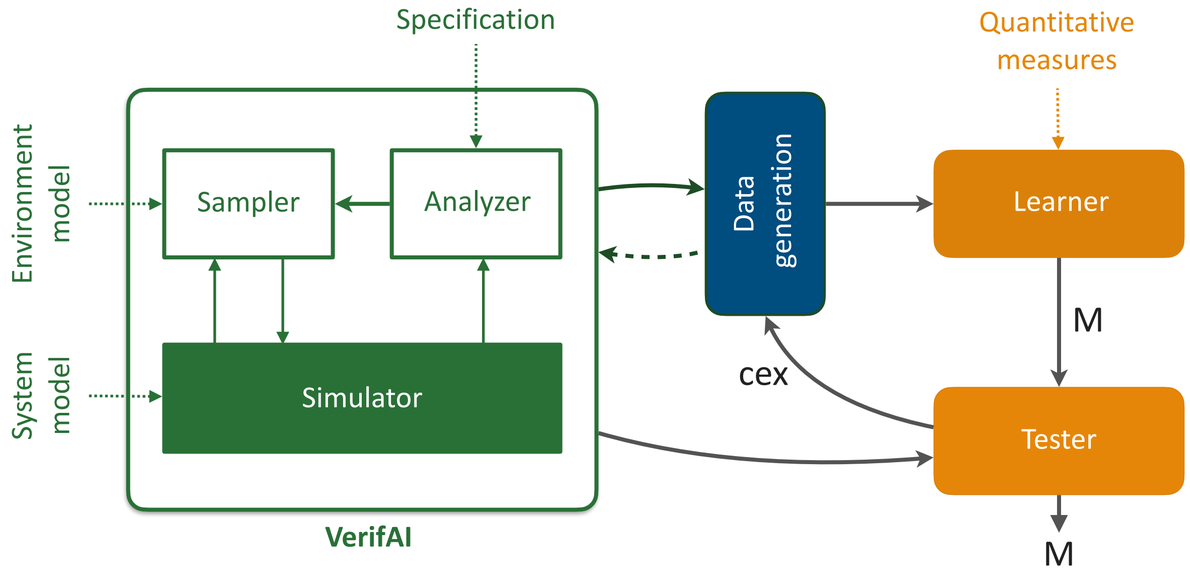


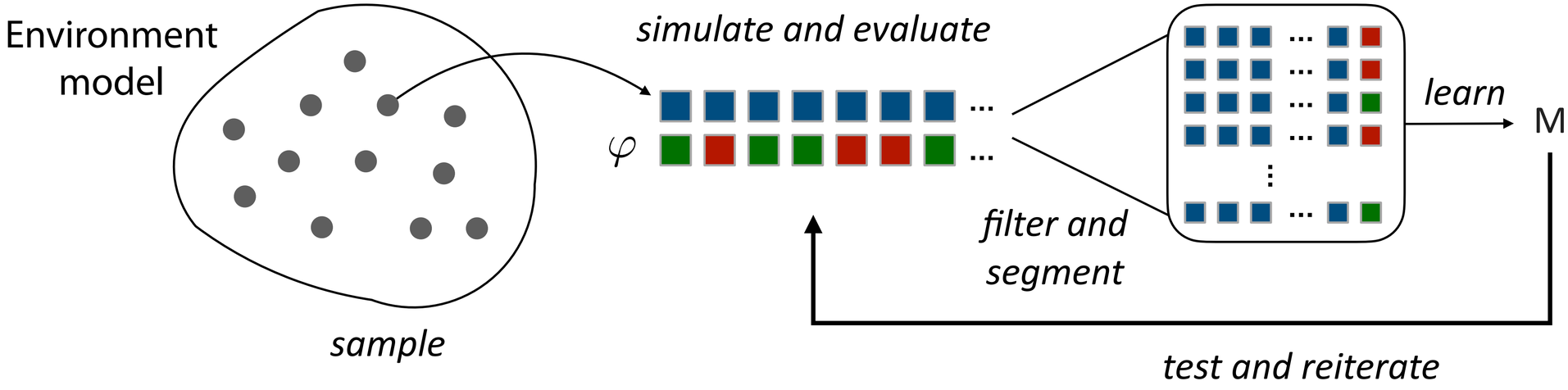
Counterexample-guided learning





Counterexample-guided learning



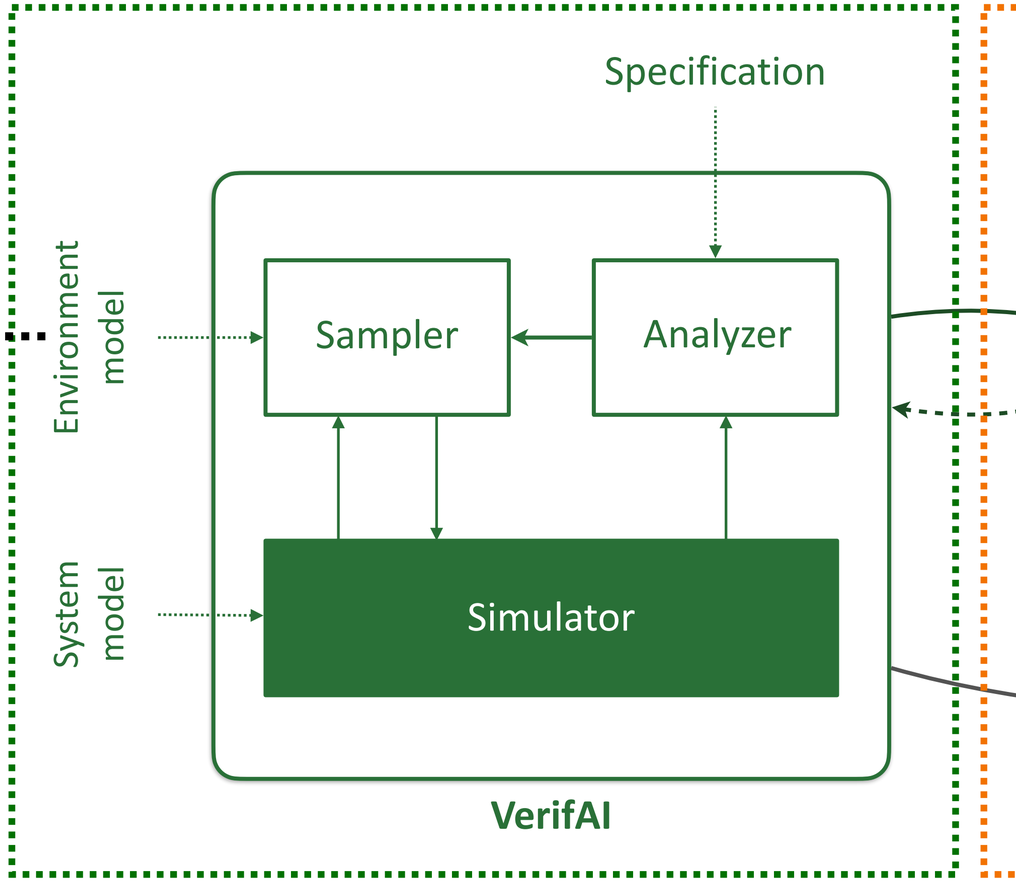


Environment modeling

Scenic: a probabilistic scenario-description programming language for modeling environments.
Fremont et al. Machine Learning Journal 2022

```
param weather =
        Uniform('ClearNoon', 'CloudyNoon',
                'WetNoon', 'MidRainyNoon',
                'ClearSunSet')

lane = Uniform(*network.lanes)
start = OrientedPoint on lane.centerline

ego = Car at start,
        with visibleDistance 60,
        with behavior EgoBehavior(10)
```

A Scenic program defines a distribution over scenes, i.e., configurations of physical objects and their behaviors over time

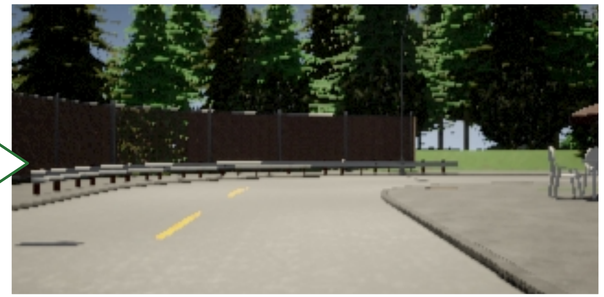


Simulation-based analysis

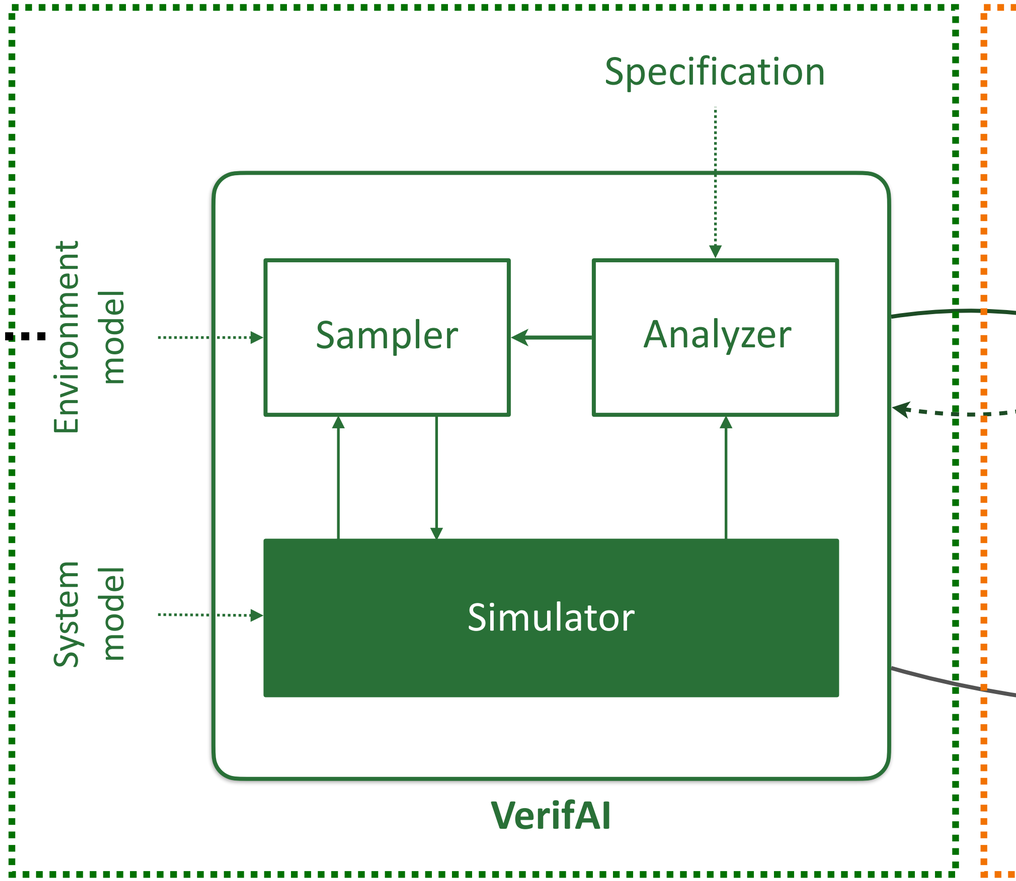
Environment modeling

Scenic: a probabilistic scenario-description programming language for modeling environments.
Fremont et al. Machine Learning Journal 2022

```
param weather =
        Uniform('ClearNoon', 'CloudyNoon',
                'WetNoon', 'MidRainyNoon',
                'ClearSunSet')

lane = Uniform(*network.lanes)
start = OrientedPoint on lane.centerline

ego = Car at start,
        with visibleDistance 60,
        with behavior EgoBehavior(10)
```



A Scenic program defines a distribution over scenes, i.e., configurations of physical objects and their behaviors over time



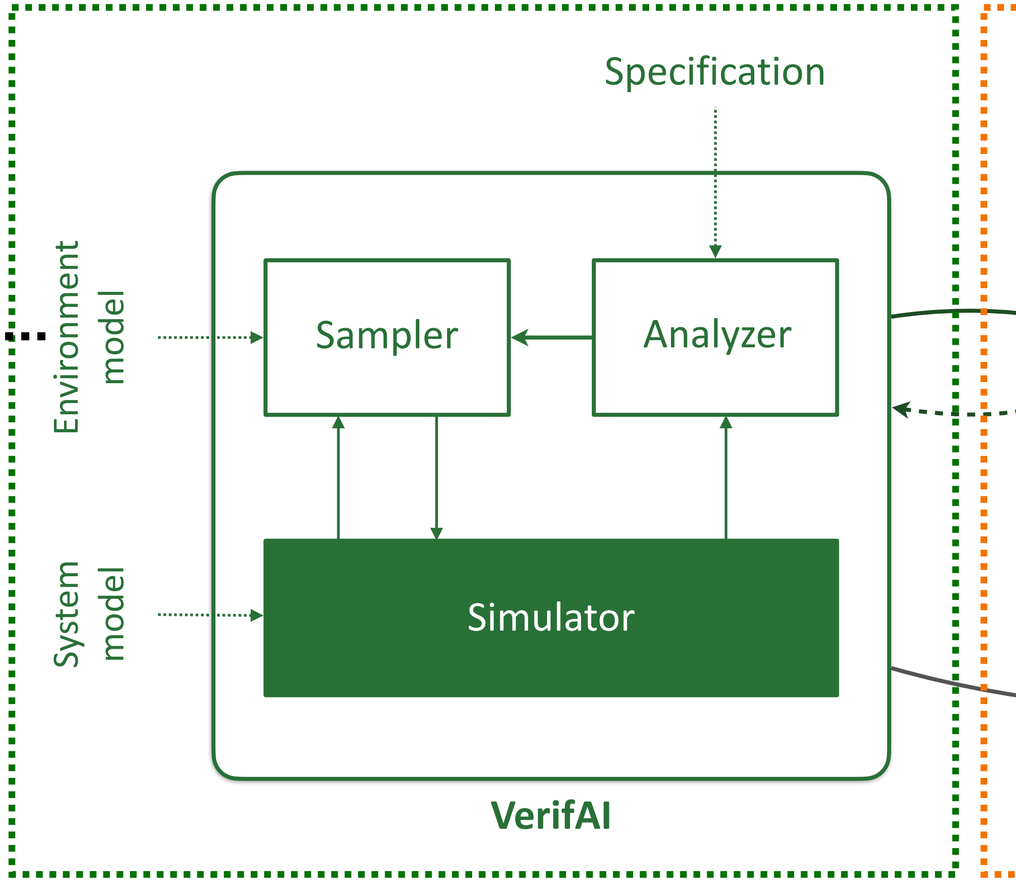
Simulation-based analysis

Environment modeling

Scenic: a probabilistic scenario-description programming language for modeling environments.
Fremont et al. Machine Learning Journal 2022

```
param weather =
        Uniform('ClearNoon', 'CloudyNoon',
                'WetNoon', 'MidRainyNoon',
                'ClearSunSet')

lane = Uniform(*network.lanes)
start = OrientedPoint on lane.centerline

ego = Car at start,
        with visibleDistance 60,
        with behavior EgoBehavior(10)
```

A Scenic program defines a distribution over scenes, i.e., configurations of physical objects and their behaviors over time



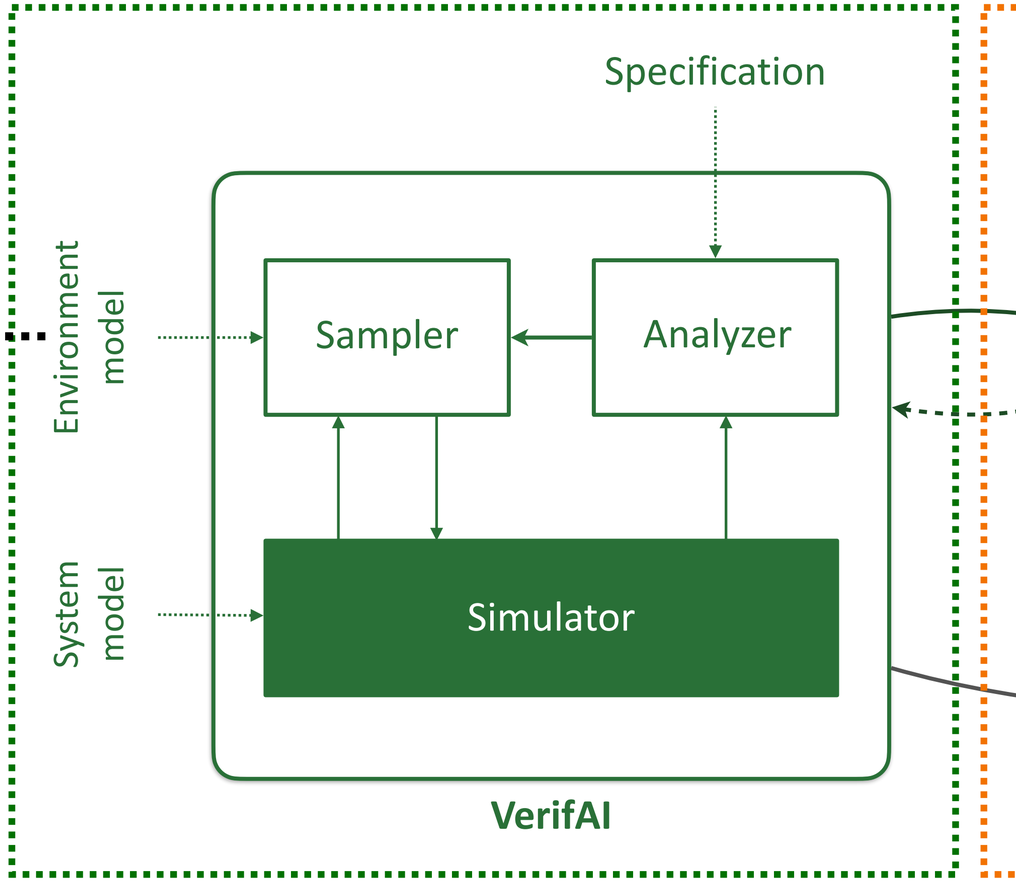
Simulation-based analysis

Environment modeling

Scenic: a probabilistic scenario-description programming language for modeling environments.
Fremont et al. Machine Learning Journal 2022

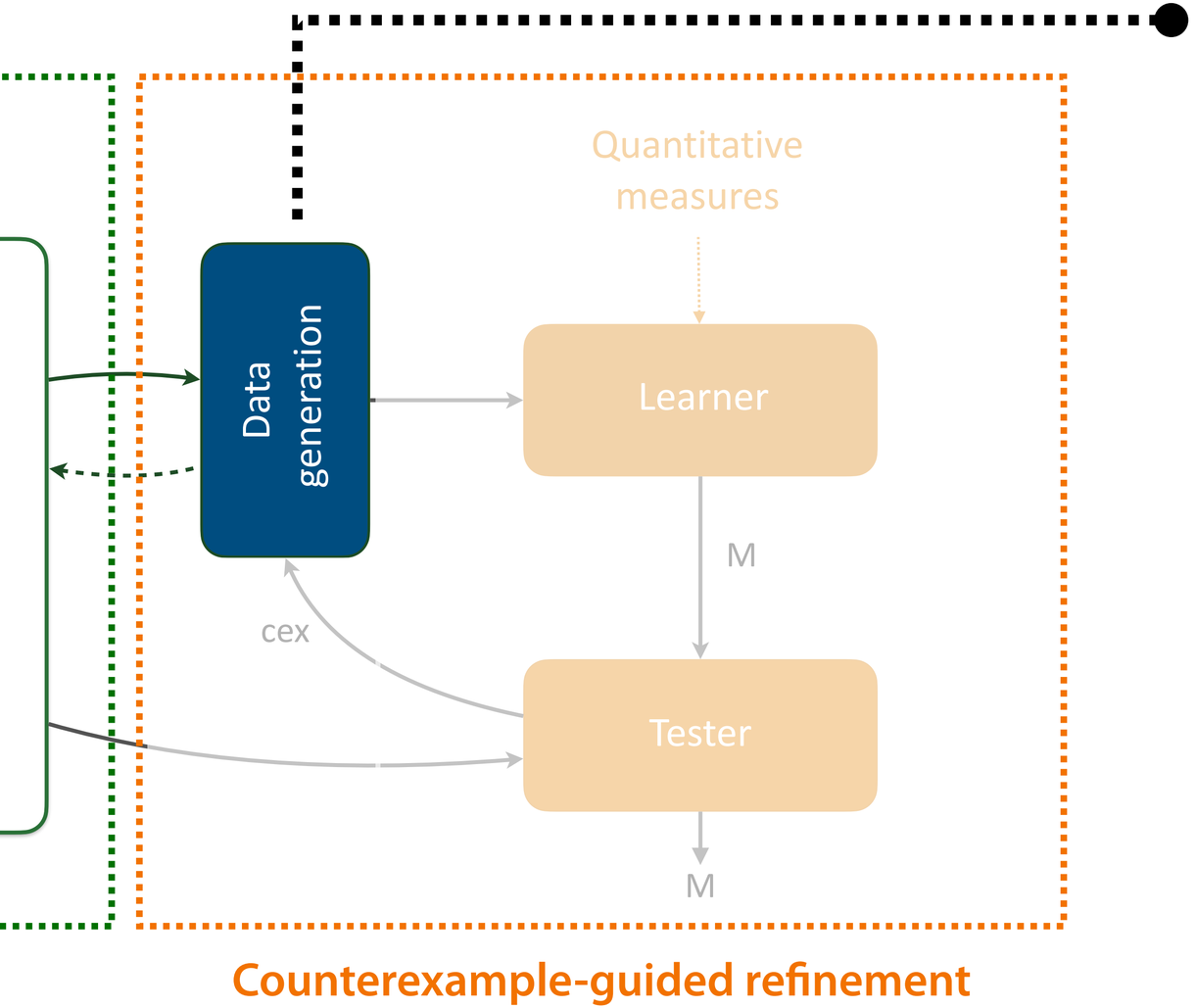
```
param weather =
        Uniform('ClearNoon', 'CloudyNoon',
                'WetNoon', 'MidRainyNoon',
                'ClearSunSet')

lane = Uniform(*network.lanes)
start = OrientedPoint on lane.centerline

ego = Car at start,
        with visibleDistance 60,
        with behavior EgoBehavior(10)
```

A Scenic program defines a distribution over scenes, i.e., configurations of physical objects and their behaviors over time



Simulation-based analysis

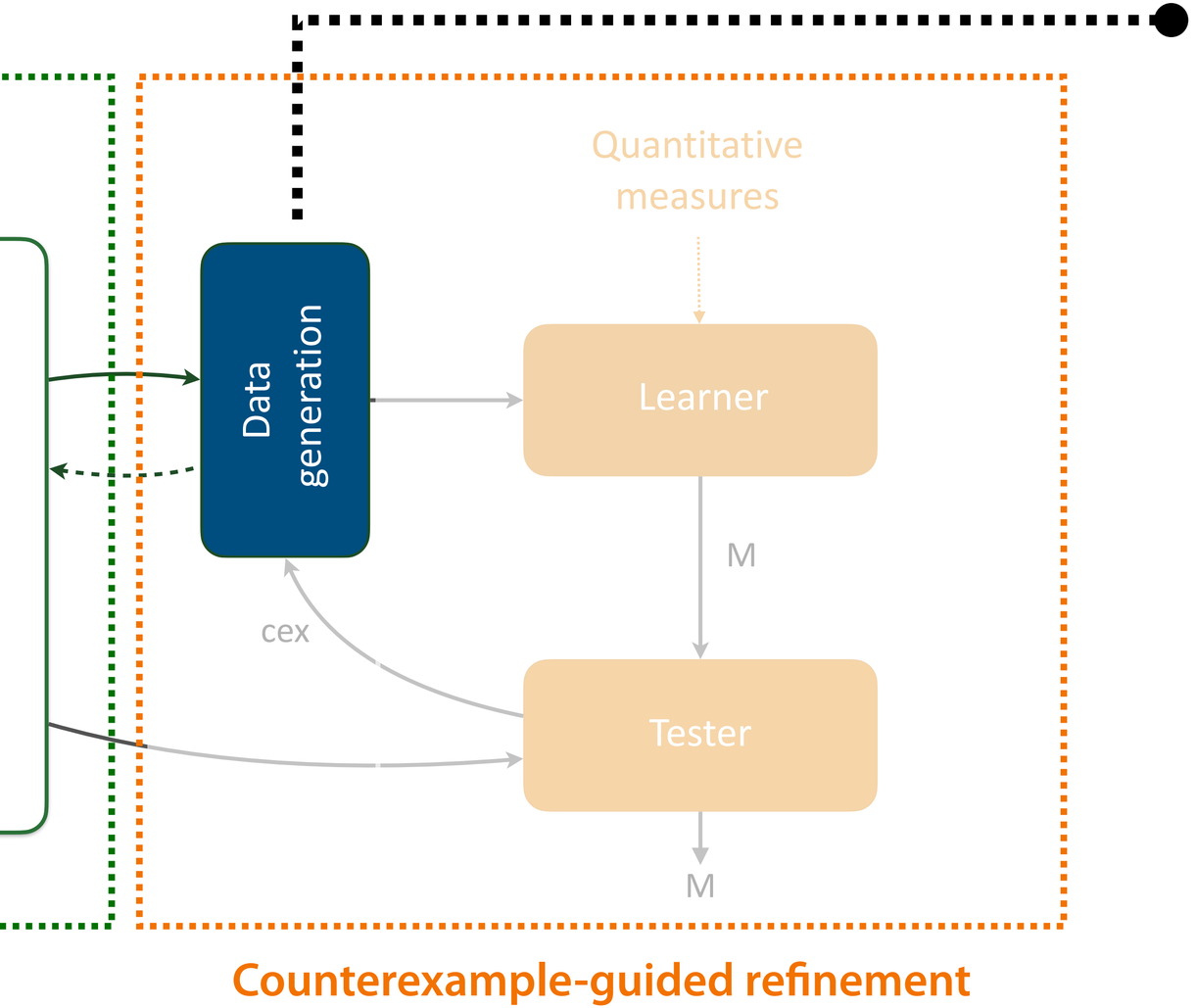
Data generation



Counterexample-guided refinement

Create monitor training set from simulation runs

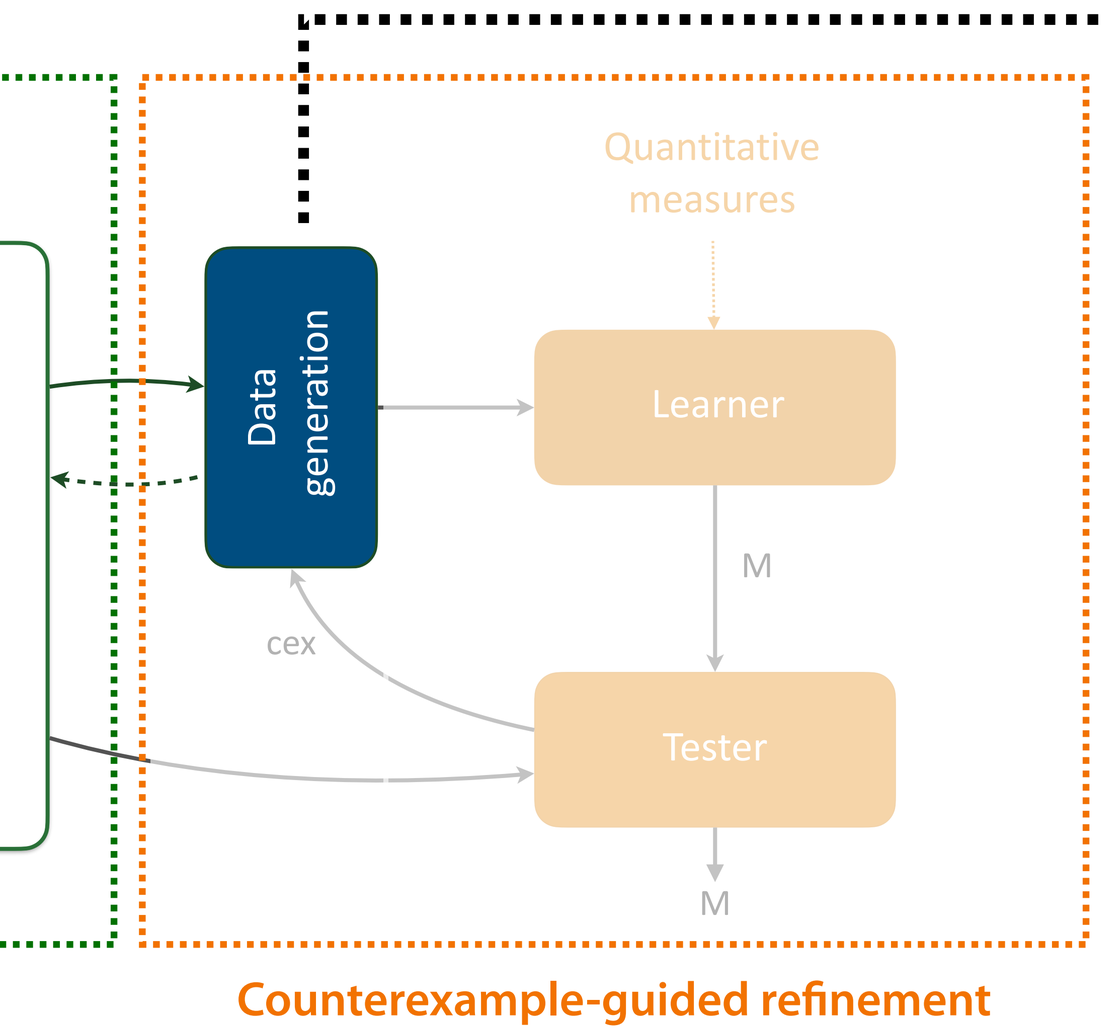
Data generation



Create monitor training set from simulation runs

- **Filtering:** mapping traces to other traces using transformation functions
- **Projection:** mapping a sequence of simulation events to a (sub)set of events that can be reliably observed at runtime

Data generation

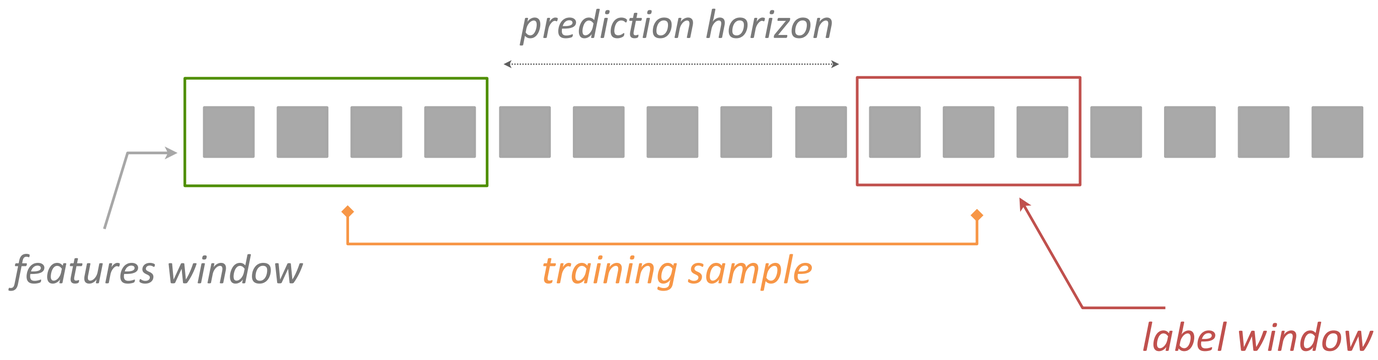


Counterexample-guided refinement

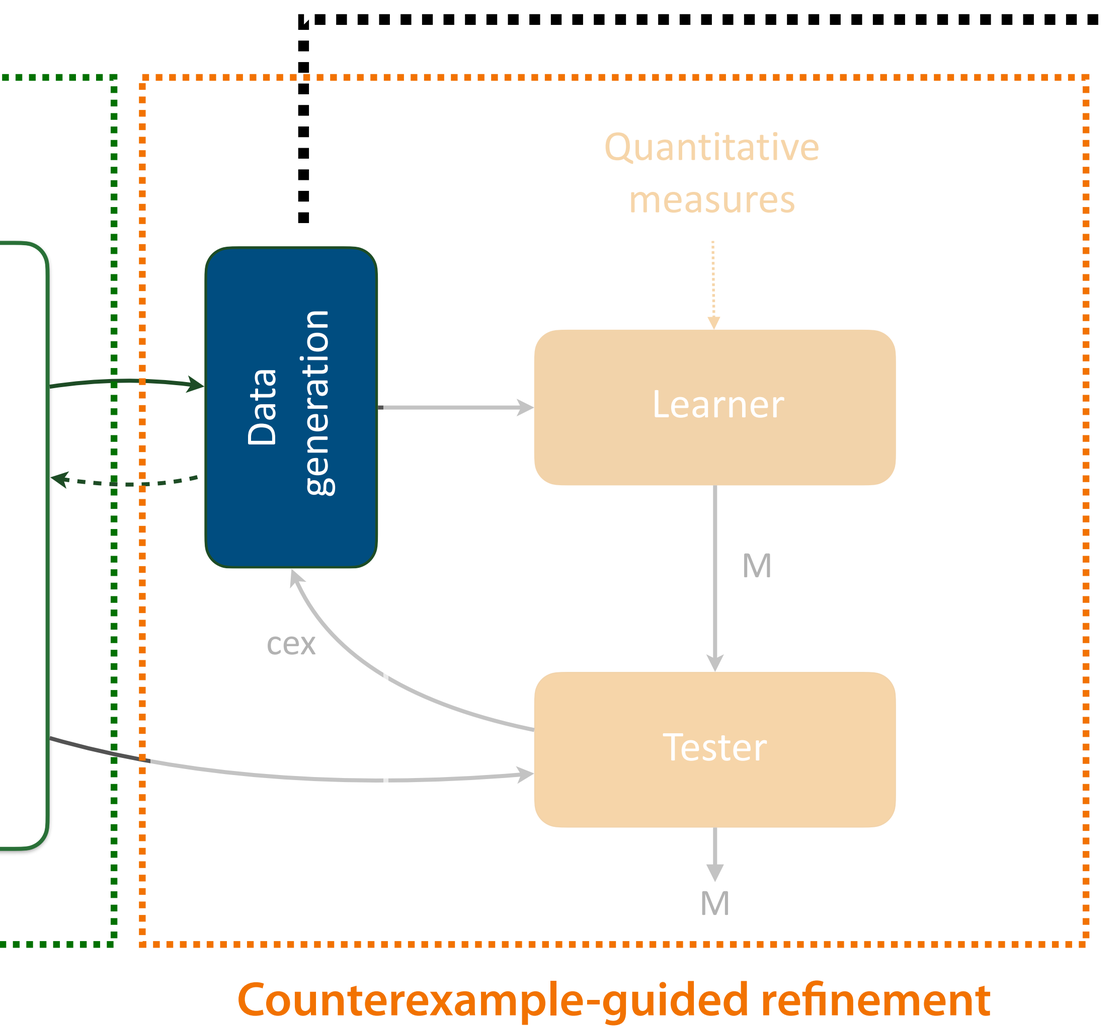
Create monitor training set from simulation runs

- **Filtering:** mapping traces to other traces using transformation functions
- **Projection:** mapping a sequence of simulation events to a (sub)set of events that can be reliably observed at runtime

- Segment every simulation run in sliding window fashion:



Data generation

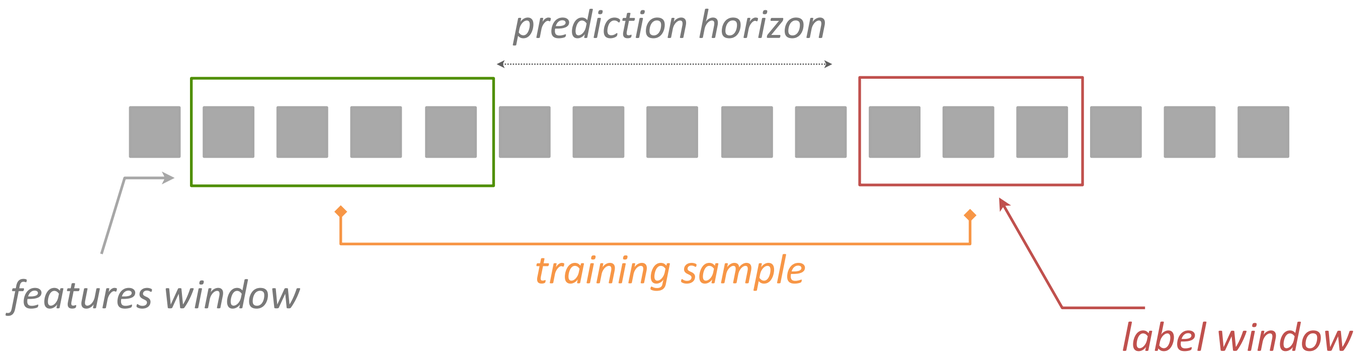


Counterexample-guided refinement

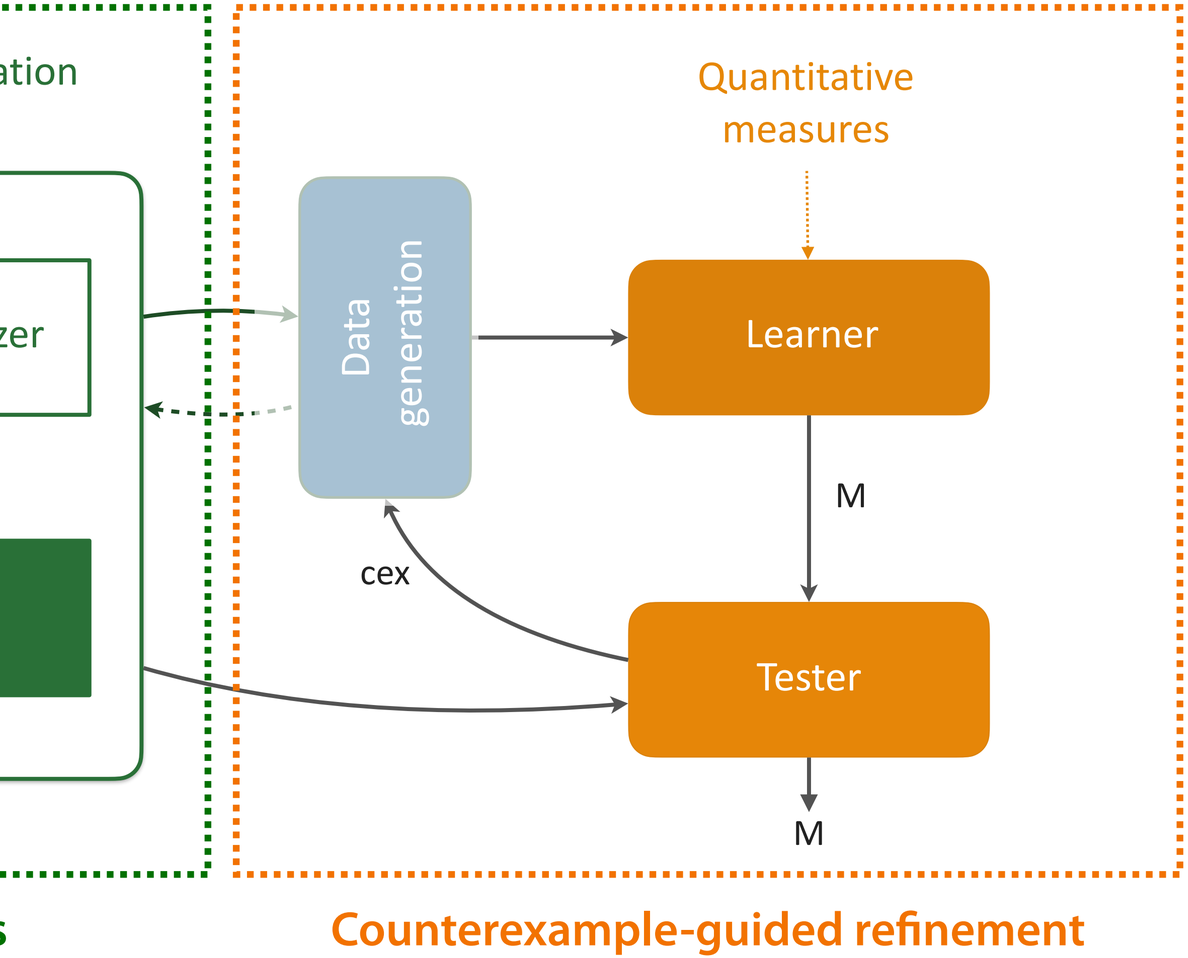
Create monitor training set from simulation runs

- **Filtering:** mapping traces to other traces using transformation functions
- **Projection:** mapping a sequence of simulation events to a (sub)set of events that can be reliably observed at runtime

- Segment every simulation run in sliding window fashion:



Learning and refinement



Learner:

- any learner for the chosen class of monitors
- quantitative measures: misclassification rate, weighted error probability, etc.

Tester:

- conformance testing using new simulation runs
- for high confidence rely on statistical theory to determine number of simulations, e.g., Hoeffding's inequality

Refinement:

- collect counterexamples during testing and forward to data generation
- termination: quantitive measure below threshold, observe stability in false positive and false negative rates, etc.

Example: Image-based lane keeping

No Monitor

With Monitor

Monitor was able to mostly keep the vehicle in the lane by appropriately switching to the safe controller

Example: Image-based lane keeping

No Monitor

With Monitor

Monitor was able to mostly keep the vehicle in the lane by appropriately switching to the safe controller

Summary

- We introduced the notion of monitorable ODDs
- We presented a counterexample-guided framework for learning ODDs
- We demonstrated the importance of defining monitors over histories of observations
- **Discussion:**
  - Automating the feature engineering process
  - A detailed investigation on the role of statistical learning methods
  - Methods for exploring the Pareto-optimal space of monitors, e.g., accuracy vs efficiency
  - Explainability of learned monitors

Thank you!