Formal Analysis of AI-Based Autonomy: From Modeling to Runtime Assurance

Hazem Torfah **Sanjit A. Seshia** **Daniel J. Fremont** **Sebastian Junges**




UC Berkeley UC Santa Cruz Radboud University

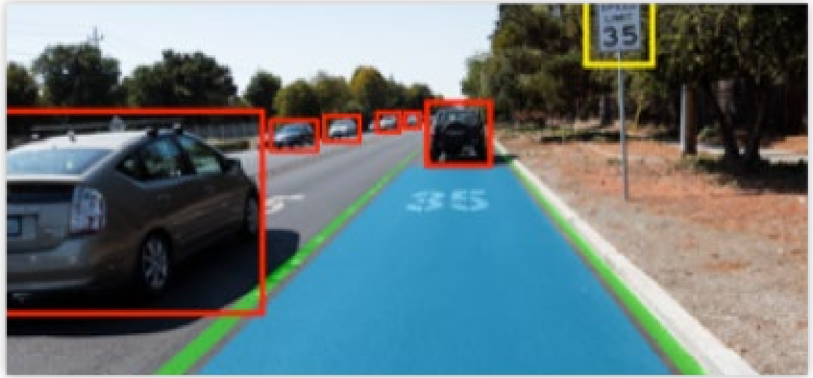
http://learnverify.org/VerifiedAI

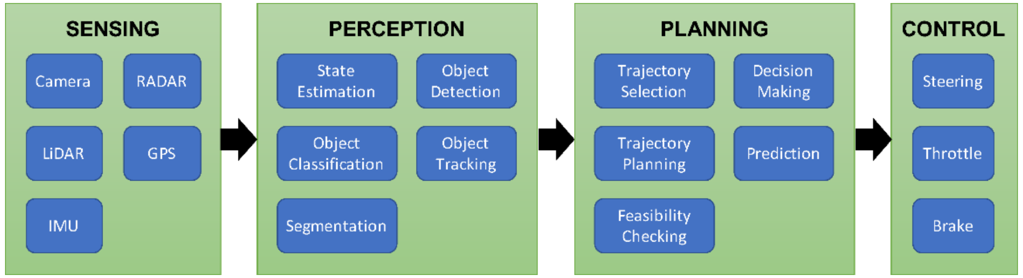
RV 2021 Tutorial
October 14, 2021

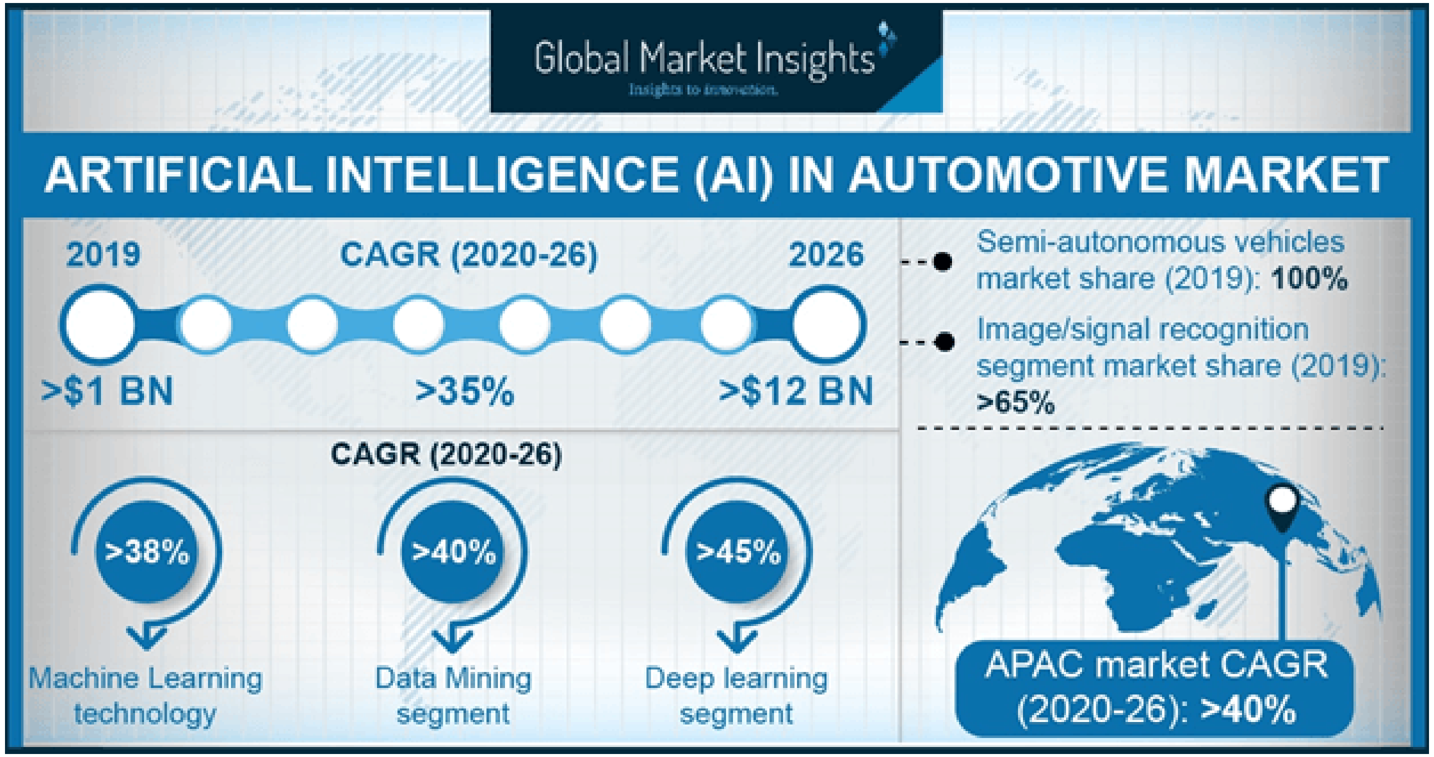
Artificial Intelligence (AI) and Autonomy

Computational Systems that attempt to mimic aspects of human intelligence, including especially the ability to learn from experience.

Growing Use of Machine Learning/Artificial Intelligence in Safety-Critical Autonomous & Semi-Autonomous Systems







Source: gminsights.com

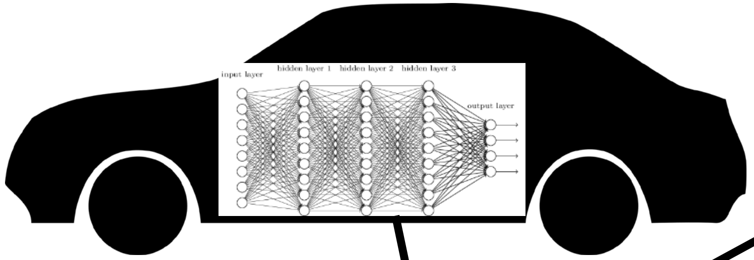
Growing Concerns about Safety:

- Numerous papers showing that *Deep Neural Networks can be easily fooled*
- *Accidents*, including some *fatal,* involving potential failure of AI/ML-based perception systems in self-driving cars

Can we address the Design & Verification Challenges of AI/ML-Based Autonomy with Formal Methods?

Challenges for Verified AI

S. A. Seshia, D. Sadigh, S. S. Sastry.
Towards Verified Artificial Intelligence. July 2016. https://arxiv.org/abs/1606.08514.



System **S**

Environment **E**

Specification **φ**

Does S || E satisfy φ?

YES [+ proof]

NO [+ counterexample]

Need to Search Very High-Dimensional Input and State Spaces

Design Correct-by-Construction?

Need Principles for Verified AI

Challenges | **Principles**

1. Environment (incl. Human) Modeling →
2. Formal Specification →
3. Learning Systems Representation →
4. Scalable Training, Testing, Verification →
5. Design for Correctness →

?

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514.

http://learnverify.org/VerifiedAI

Scenic

High-Level, Probabilistic Programming Language for Modeling Environment Scenarios

VerifAI

Requirements Specification + Algorithms for Design, Verification, Testing, Debugging

GitHub **Open-Source Tools**

https://github.com/BerkeleyLearnVerify/Scenic
https://github.com/BerkeleyLearnVerify/VerifAI

for

Industry

Improve assurance of the systems you build

Academia

Use these tools in your research

Government/ Regulators

Evaluate the safety of AI-based autonomous systems

Share Scenarios and Metrics **Community** Develop Corpus of Tools and Data

Outline

- Overview of Scenic and VerifAI
  - Basic syntax of the Scenic language
- Falsification
  - Case study in the Webots simulator
- Dynamic Scenarios in Scenic
  - Case study in autonomous driving simulators (e.g., CARLA)
- Falsification → Debugging → Retraining
  - Case study in the X-Plane simulator
- Data-Driven Run-Time Monitor Generation with Scenic & VerifAI
  - Case study in the X-Plane simulator
- Conclusion

SCENIC: Environment Modeling and Data Generation

- *Scenic* is a probabilistic programming language defining *distributions over scenes/scenarios*
- *Use cases:* data generation, test generation, verification, debugging, design exploration, etc.

```
model scenic.domains.driving.model

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5,
                facing badAngle relative to roadDirection
```

```
model scenic.domains.driving.model

behavior PullIntoRoad():
    while (distance from self to ego) > 15:
        wait
    FollowLaneBehavior(lane=ego.lane)

ego = Car with behavior DriveAvoidingCollisions

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad
```

Example: Badly-parked car

Image created with GTA-V

Video created with CARLA

[D. Fremont et al., “Scenic: A Language for Scenario Specification and Scene Generation”, TR 2018, PLDI 2019.]

SCENIC: Environment Scenario Modeling Language

Scenic makes it possible to specify broad scenarios with complex structure, then generate many concrete instances from them automatically:

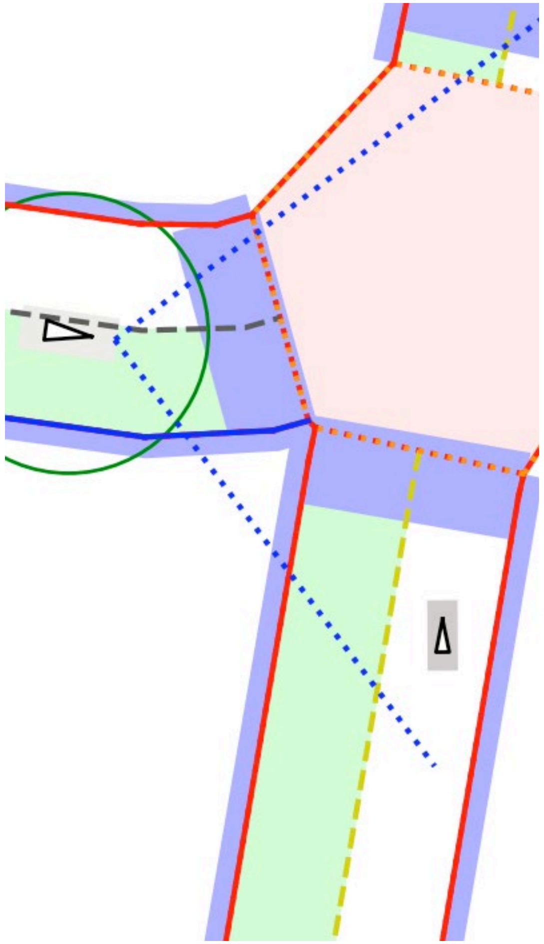
Platoons

Bumper-to-Bumper Traffic

(~20 lines of Scenic code)

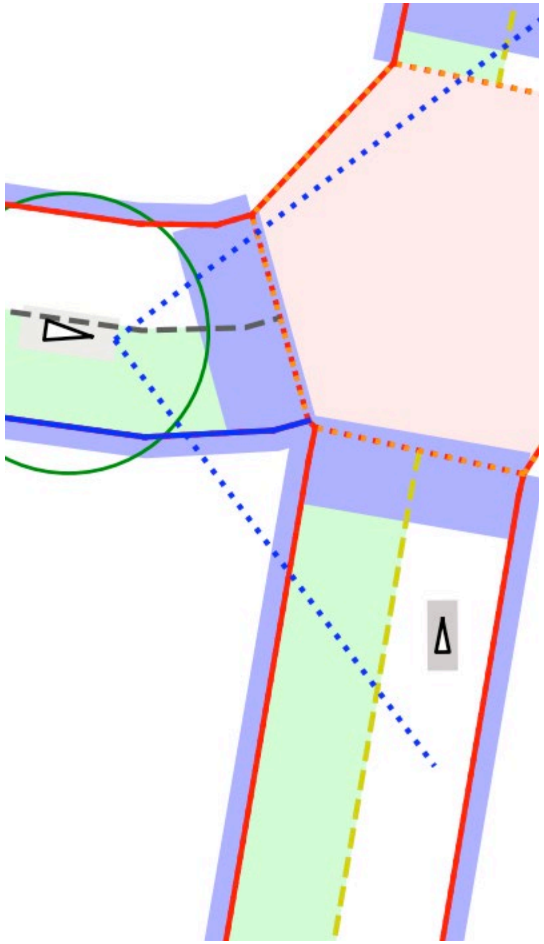
Example: a Badly-Parked Car

Example: a Badly-Parked Car



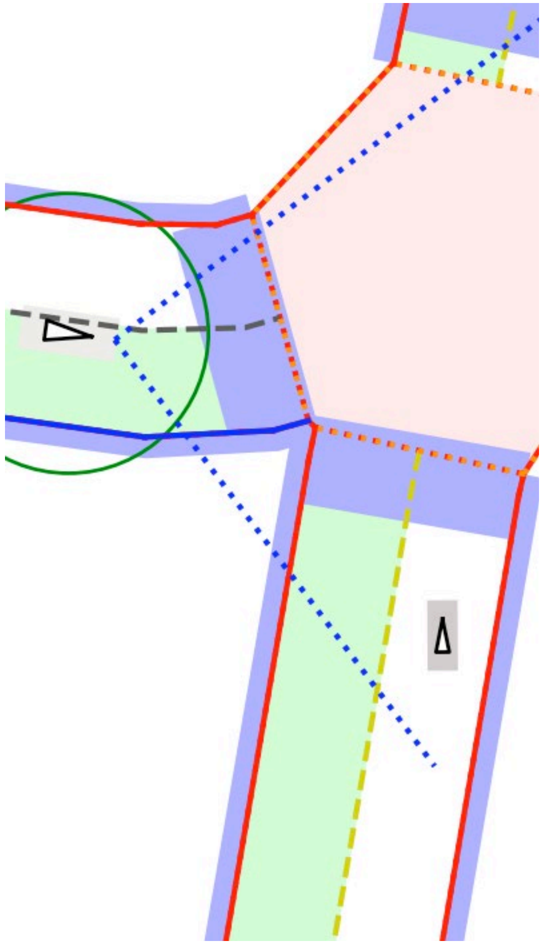
```
model scenic.simulators.gta.model # defines Car, etc.
```

Example: a Badly-Parked Car



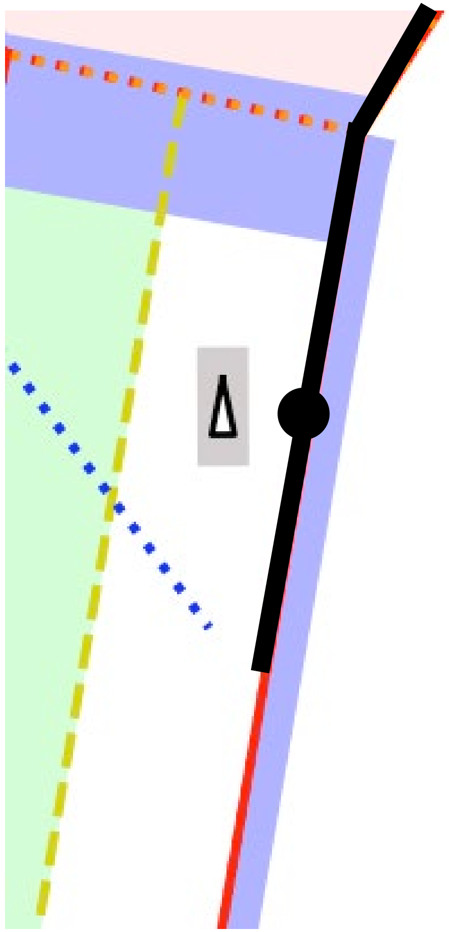
```
model scenic.simulators.gta.model # defines Car, etc.

ego = Car
```

Example: a Badly-Parked Car



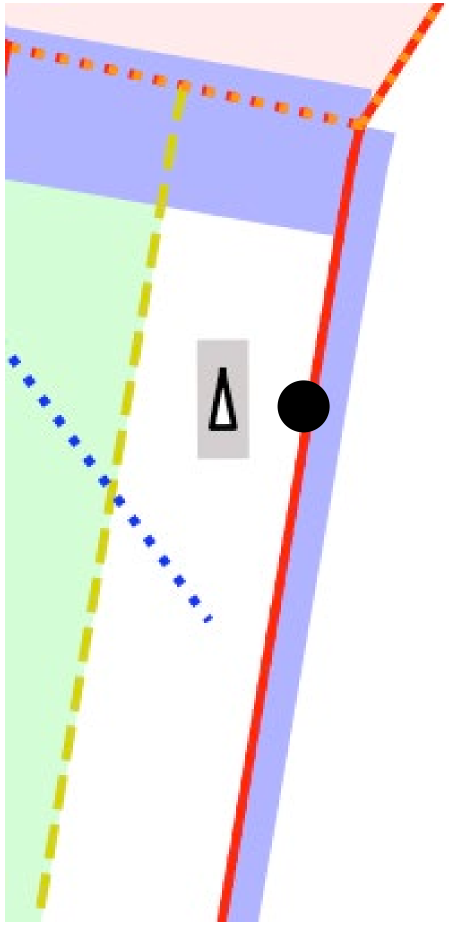
```
model scenic.simulators.gta.model # defines Car, etc.

ego = Car

spot = OrientedPoint on visible curb
```

Example: a Badly-Parked Car



```
model scenic.simulators.gta.model # defines Car, etc.

ego = Car

spot = OrientedPoint on visible curb
```

Example: a Badly-Parked Car

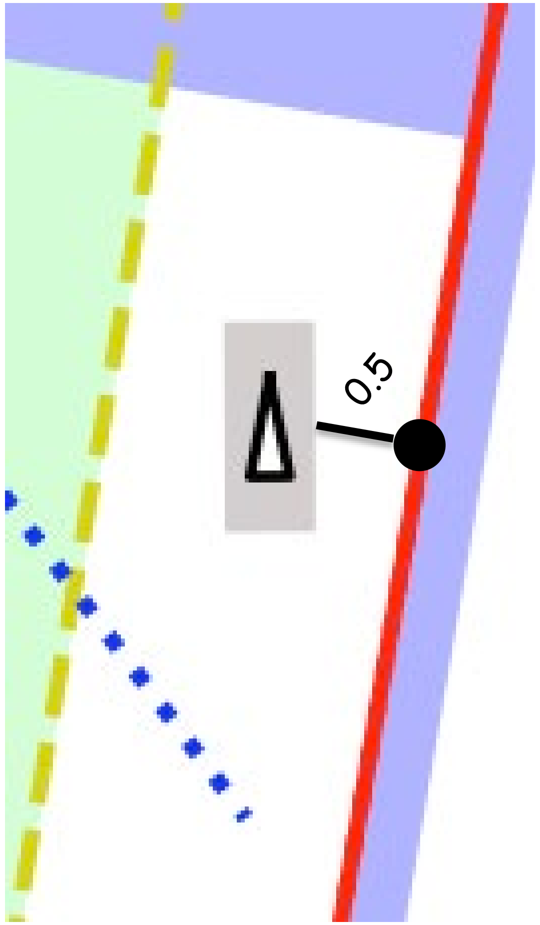


```
model scenic.simulators.gta.model # defines Car, etc.

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
```

angled left or right
uniformly at random

uniform
distribution over
this interval

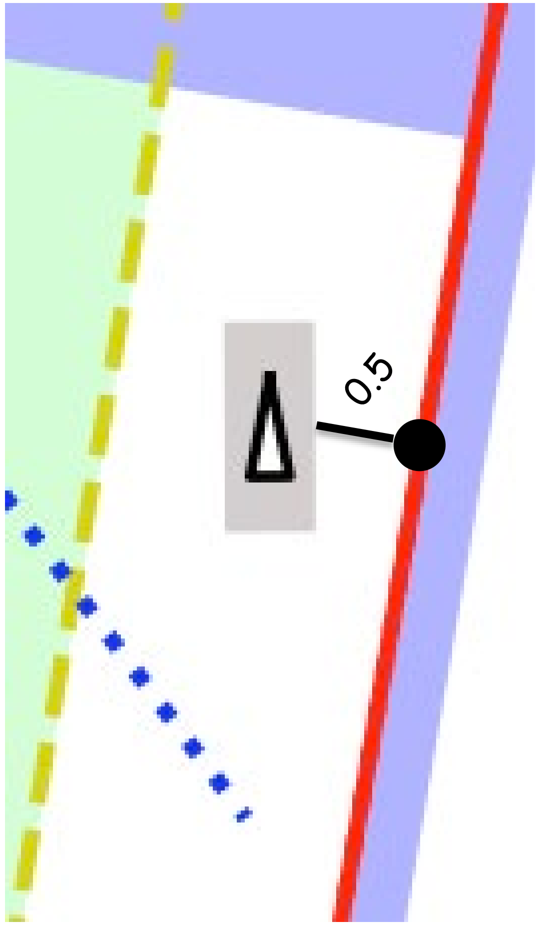
Example: a Badly-Parked Car




```
model scenic.simulators.gta.model # defines Car, etc.

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
Car left of spot by 0.5,
```

specify offset in meters

Example: a Badly-Parked Car



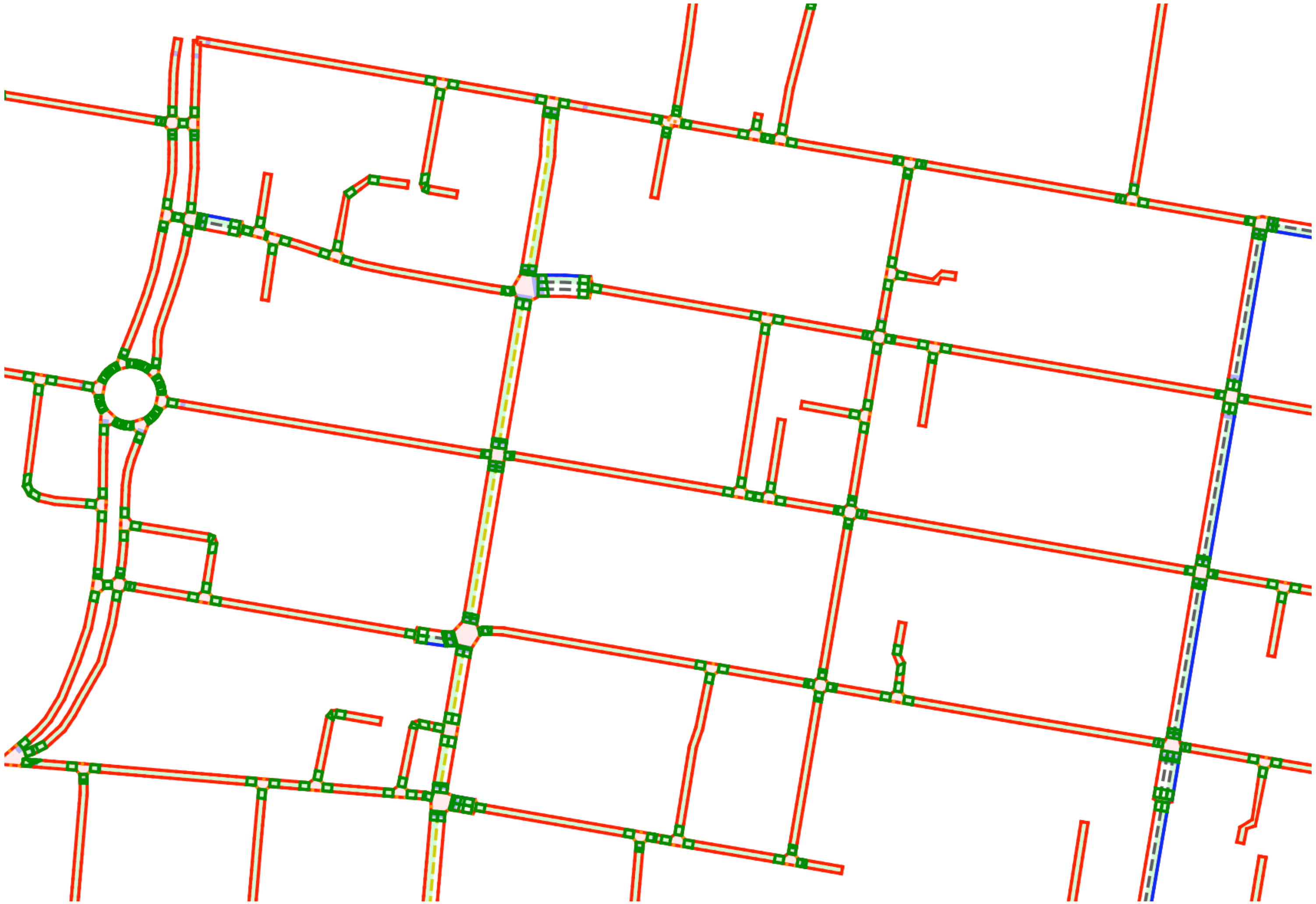

```
model scenic.simulators.gta.model # defines Car, etc.

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
Car left of spot by 0.5,
    facing badAngle relative to roadDirection
```

Example: a Badly-Parked Car, Rendered with GTA-V

Domain-Specific Sampling Techniques

- Prune infeasible parts of the space by dilating polygons

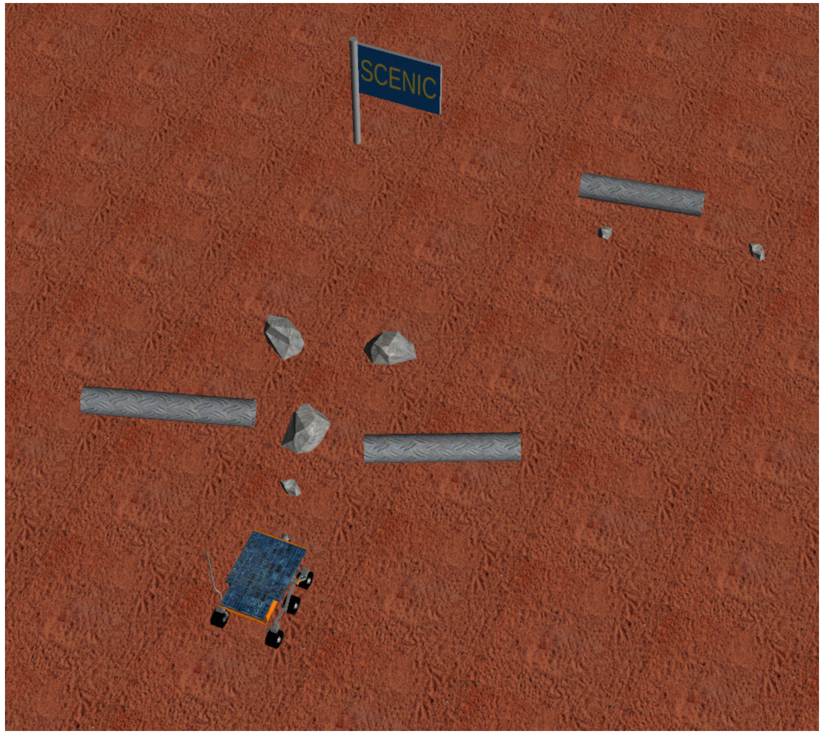
```
require distance to taxi <= 5
require 15 deg <= (relative
  heading of taxi) <= 45 deg
```



Early Applications of Scenic

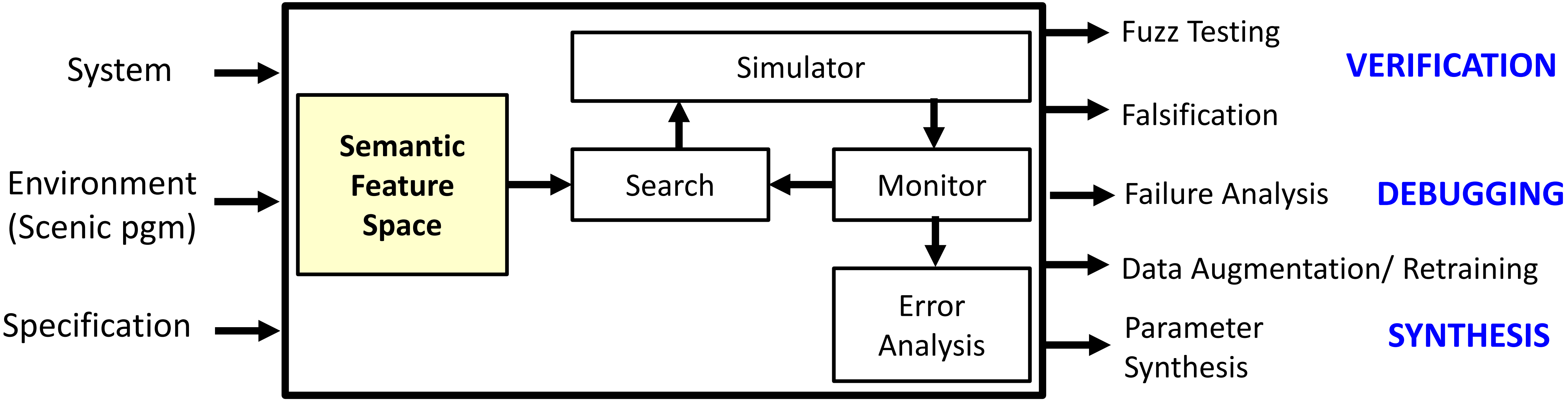
[see PLDI'19 paper]

- Exploring system performance
  - Generating specialized test sets
- Debugging a known failure
  - Generalizing in different directions
- Designing more effective training sets
  - Training on hard cases



VERIFAI: A Toolkit for the Design and Analysis of AI-Based Systems [CAV 2019]

https://github.com/BerkeleyLearnVerify/VerifAI



Outline

- Overview of Scenic and VerifAI
  - Basic syntax of the Scenic language
- Falsification
  - Case study in the Webots simulator
- Dynamic Scenarios in Scenic
  - Case study in autonomous driving simulators (e.g., CARLA)
- Falsification → Debugging → Retraining
  - Case study in the X-Plane simulator
- Data-Driven Run-Time Monitor Generation with Scenic & VerifAI
  - Case study in the X-Plane simulator
- Conclusion

Simulation-based Falsification: Logical Formulas to Objective Functions

- Use Temporal Logics with Quantitative Semantics (STL, MTL, etc.)

- Example:

  $\boldsymbol{G}_{[0,\tau]}(\text{dist(vehicle, obstacle)} > \delta)$

  $\Downarrow$

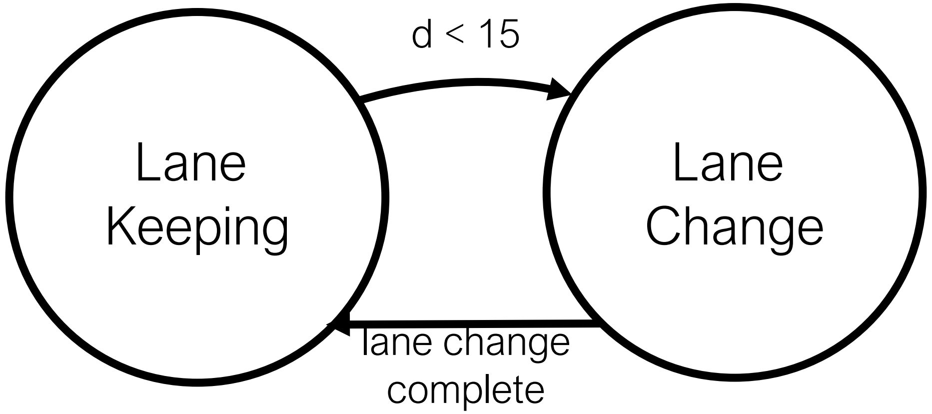
  $\inf_{[0,\tau]}\ [\ \text{dist(vehicle, obstacle)} - \delta\ ]$

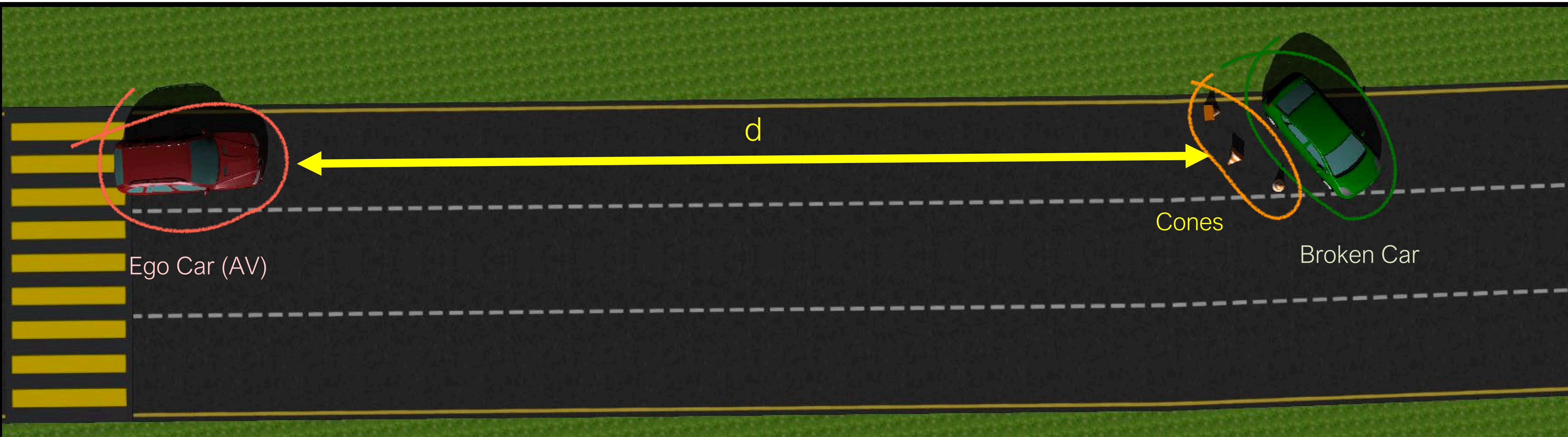
- Verification → Optimization

Falsification in VerifAI

- Input space is Semantic Feature Space
  - E.g. variables in Scenic program with their value domains / distributions
- Multi-Modal Specification
  - Metric/Signal Temporal Logic
  - Cost Function
  - Custom monitor function <Your formalism here>
- Several Sampling/Optimization Techniques
  - Passive Sampling: Uniform Random, Grid, Halton, Scenic, …
  - Active Sampling/Optimization: Bayesian Optimization, Cross Entropy, Simulated Annealing, Multi-Armed Bandit, …
  - <Your falsification method here>
- Parallelized and Multi-Objective Falsification (new feature @ RV'21)

Case Study: Falsifying a Collision-Avoidance System

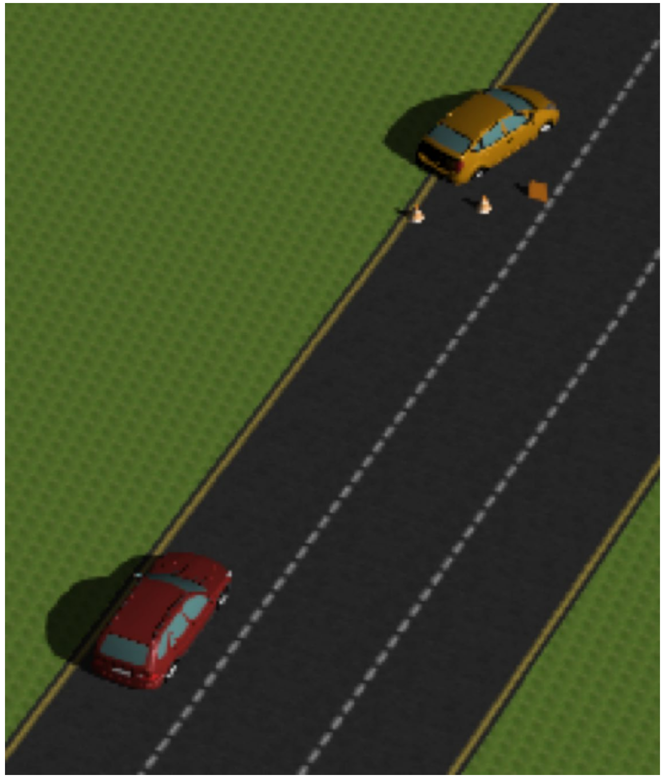




Using Scenic to Generate Initial Scenes

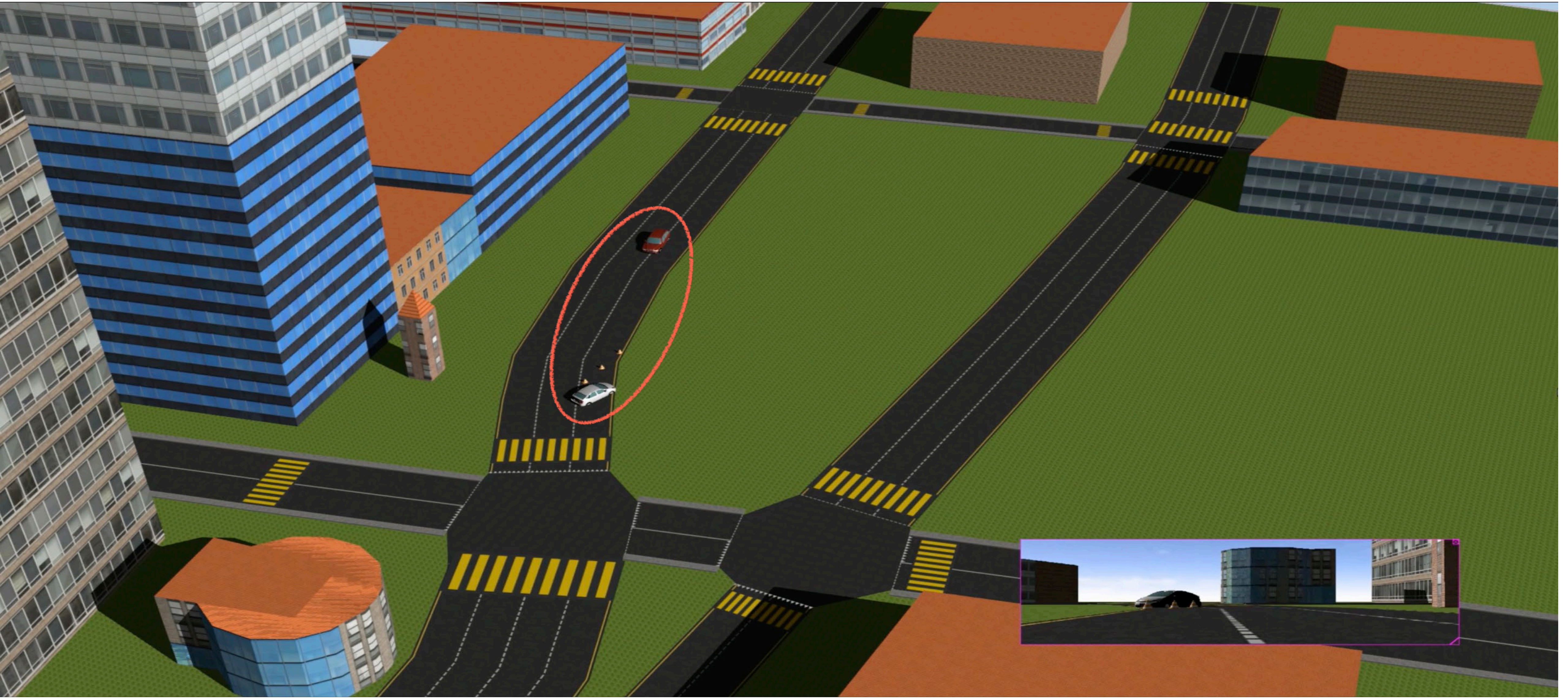
- A scene can be the initial condition for a simulation

```
# Pick location for blockage randomly along curb
blockageSite = OrientedPoint on curb

# Place traffic cones
spot1 = OrientedPoint left of blockageSite by (0.3, 1)
cone1 = TrafficCone at spot1,
                    facing (0, 360) deg

...

# Place disabled car ahead of cones
SmallCar ahead of spot2 by (-1, 0.5) @ (4, 10),
         facing (0, 360) deg
```

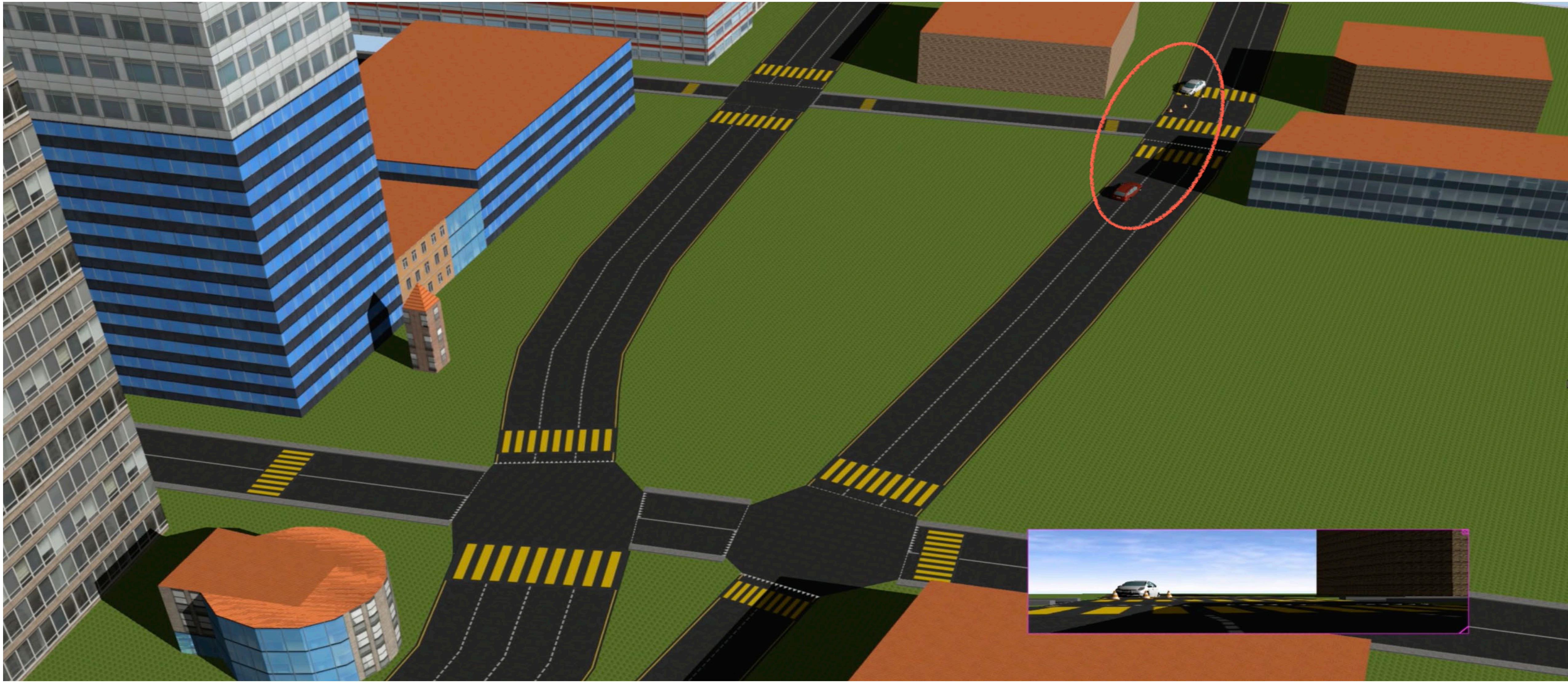


- Can also include parameters for controllers (e.g. reaction time, how quickly to swerve)

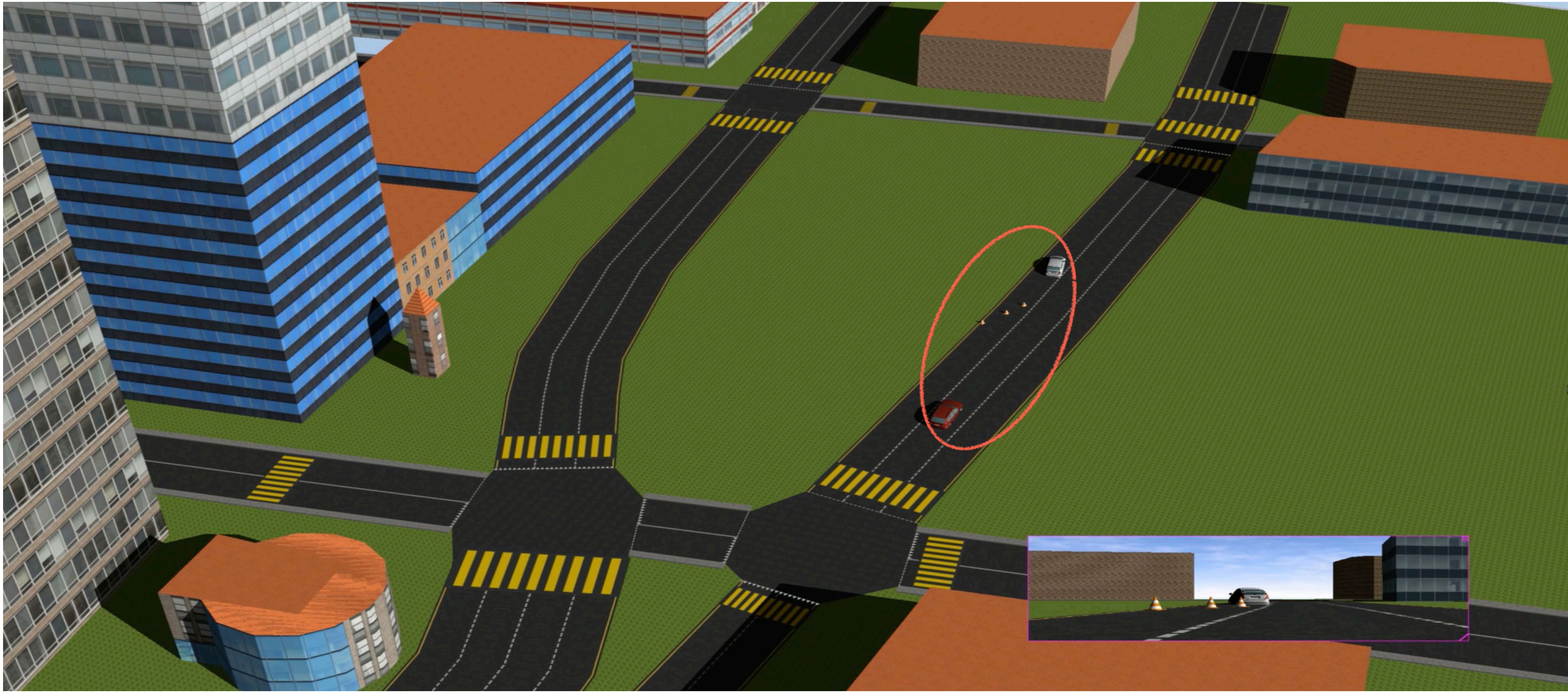
Using Scenic to Generate Initial Scenes



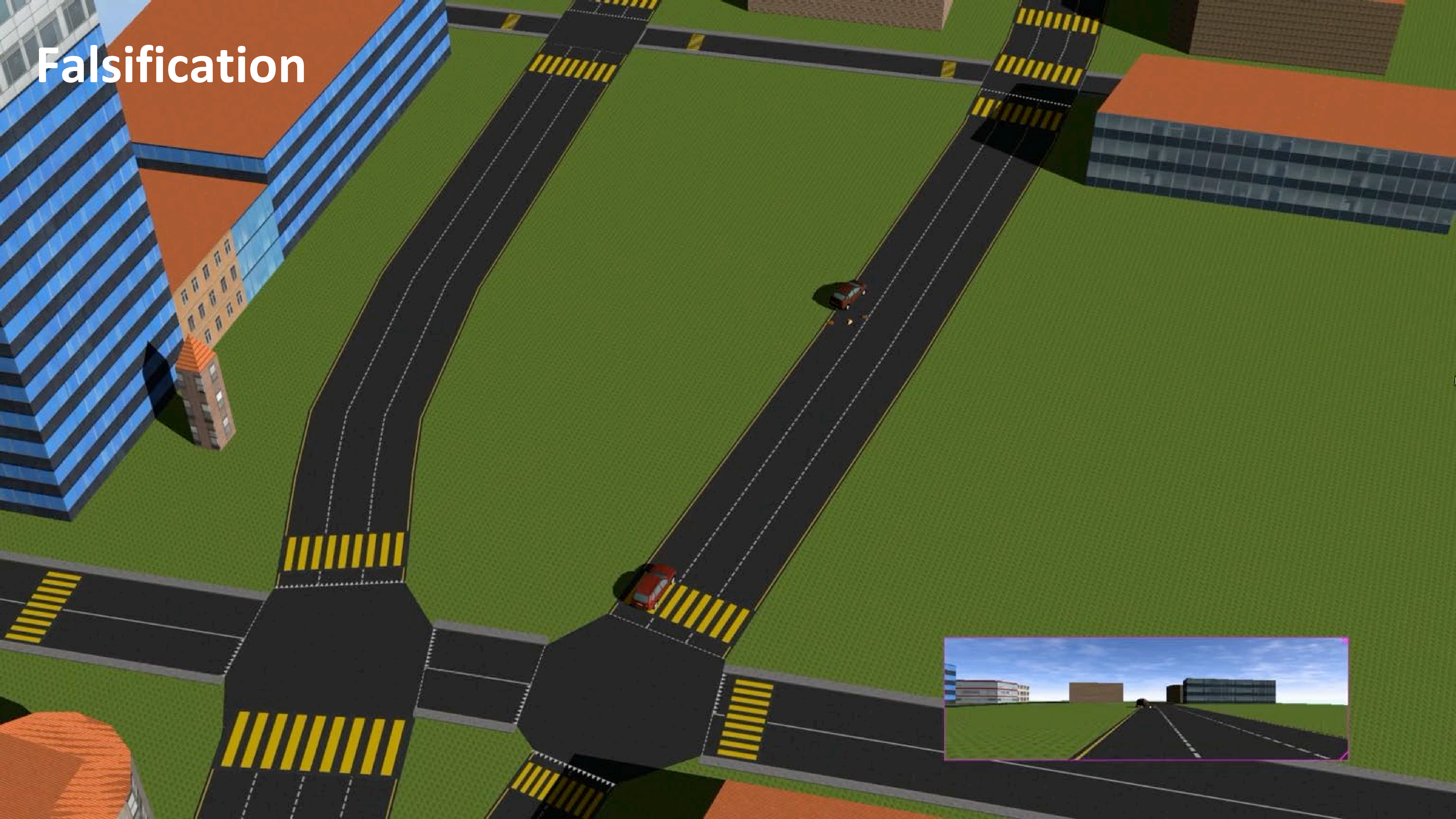
Using Scenic to Generate Initial Scenes



Using Scenic to Generate Initial Scenes



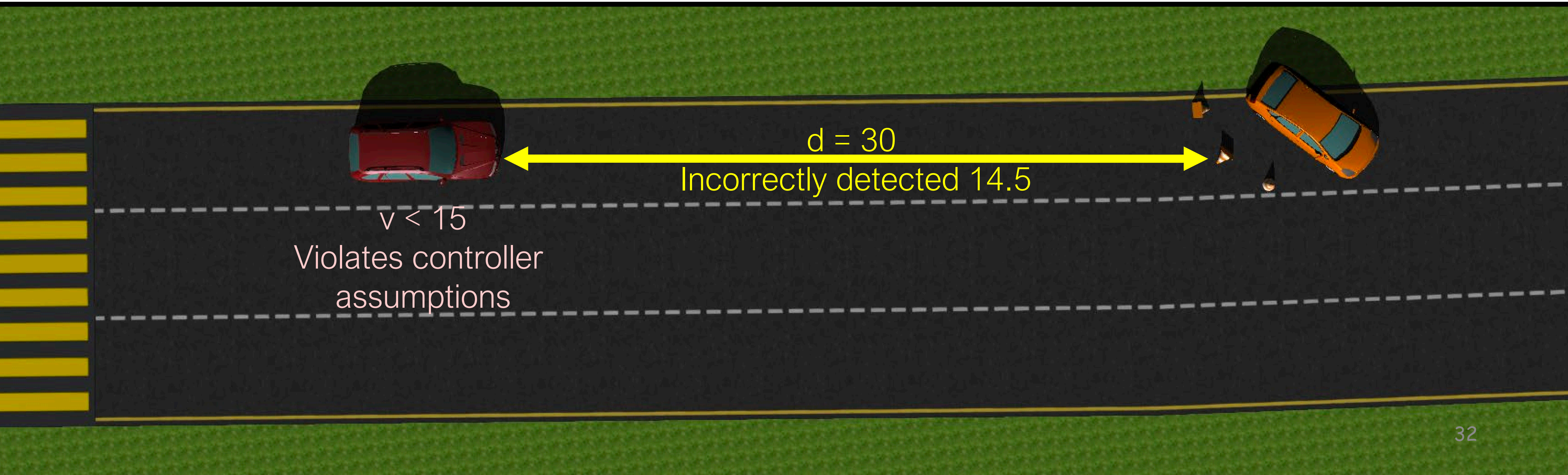
Falsification



Analyzing the Failure: Repair and Retraining

Fix the controller:
Update model assumptions
and re-design controller

Retrain the perception module:
Collect the counter-example images and
retrain the network [IJCAI'18]



Outline

- Overview of Scenic and VerifAI
  - Basic syntax of the Scenic language
- Falsification
  - Case study in the Webots simulator
- Dynamic Scenarios in Scenic
  - Case study in autonomous driving simulators (e.g., CARLA)
- Falsification → Debugging → Retraining
  - Case study in the X-Plane simulator
- Data-Driven Run-Time Monitor Generation with Scenic & VerifAI
  - Case study in the X-Plane simulator
- Conclusion

Going Beyond Initial Conditions

- Scenic can also describe *dynamic agents* which take actions over time, reacting to a changing environment

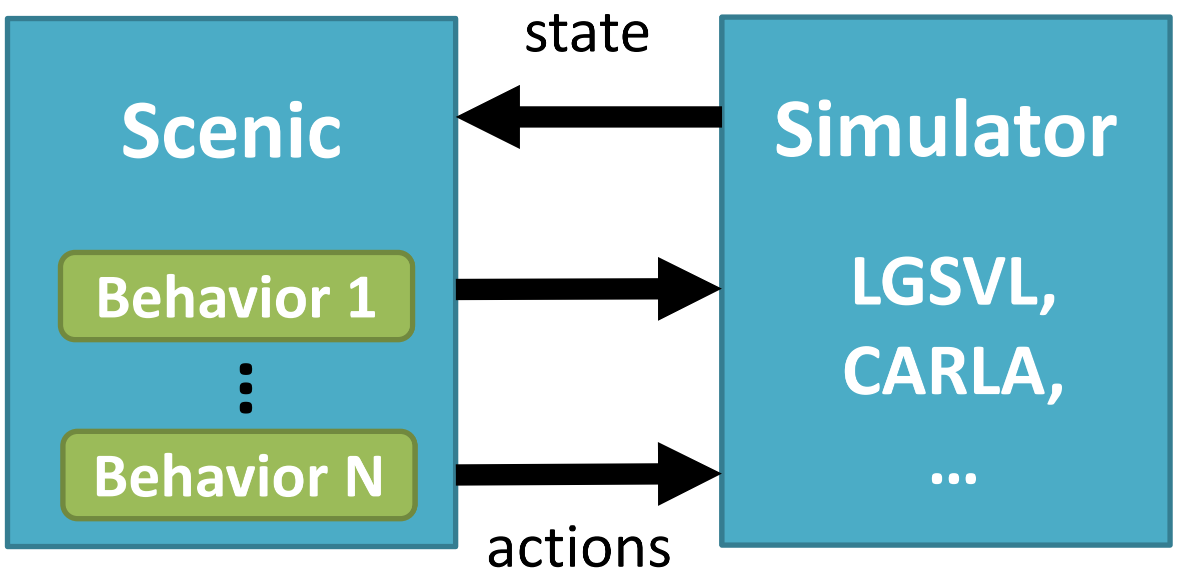
- Example: ”a badly-parked car, which suddenly pulls into the road as the ego car approaches”

- The dynamic actions of the car are specified by giving it a *behavior*

```
parkedCar = Car left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad
```

Behaviors and Actions

- Behaviors are functions running in parallel with the simulation, issuing *actions* at each time step
  - e.g. for AVs: set throttle, set steering angle, turn on turn signal
  - Provided by a Scenic library for the driving domain
  - Abstract away details of simulator interface
- Behaviors can access the state of the simulation and make choices accordingly




```
behavior FollowLaneBehavior(lane):
    while True:
        throttle, steering = ...
        take (SetThrottleAction(throttle),
              SetSteerAction(steering))
```

More Advanced Temporal Constructs

- *Interrupts* allow adding special cases to behaviors without modifying their code

```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when (distance to other) < safety_distance:
        do CollisionAvoidance()
```

- *Temporal requirements* and *monitors* allow enforcing constraints during simulation

```
require always taxi in lane
require eventually ego can see pedestrian
```

A Worked Example

- OAS Voyage Scenario 2-2-XX-CF-STR-CAR:02
- Lead car periodically stops and starts; ego car must brake to avoid collision
- Cross-platform scenario works in CARLA and LGSVL

```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when (distance to other) < safety_distance:
        do CollisionAvoidance()

behavior StopsAndStarts():
    stop_delay = Range(3, 6) seconds
    last_stop = 0
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when simulation().currentTime - last_stop > stop_delay:
        do FullBraking() for 5 seconds
        last_stop = simulation().currentTime

ego = Car with behavior FollowLeadCar(safety_distance=10)
other = Car ahead of ego by 10,
            with behavior StopsAndStarts

require (Point ahead of ego by 100) in road

terminate when ego._lane is None
```

A Worked Example: CARLA

A Worked Example: LGSVL

Composing Scenarios

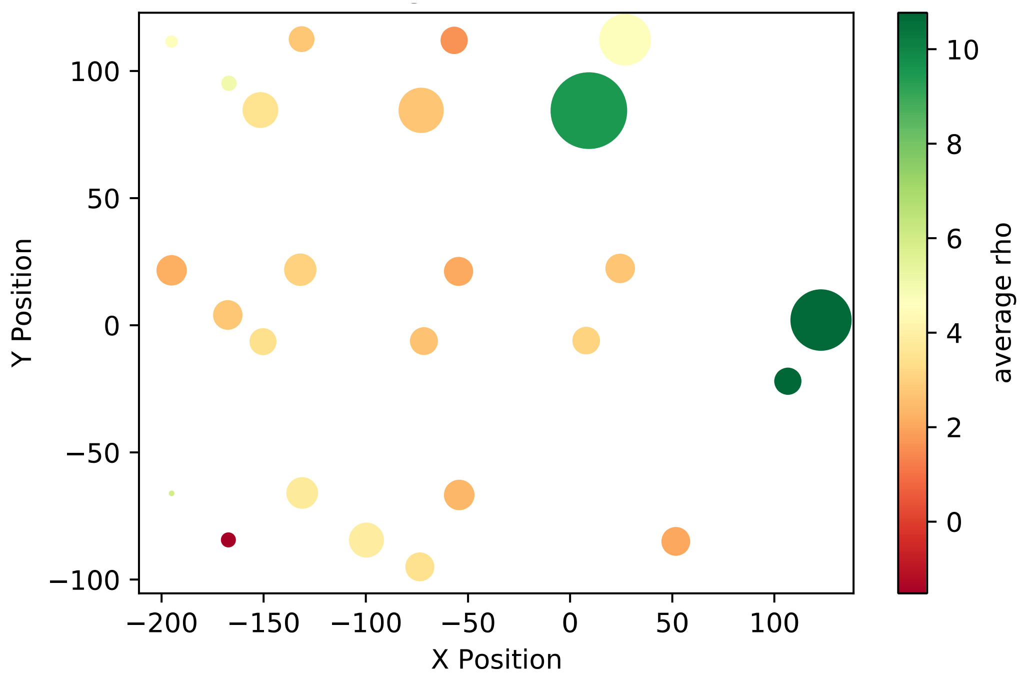
- Scenic allows scenarios to be defined modularly and combined into more complex scenarios

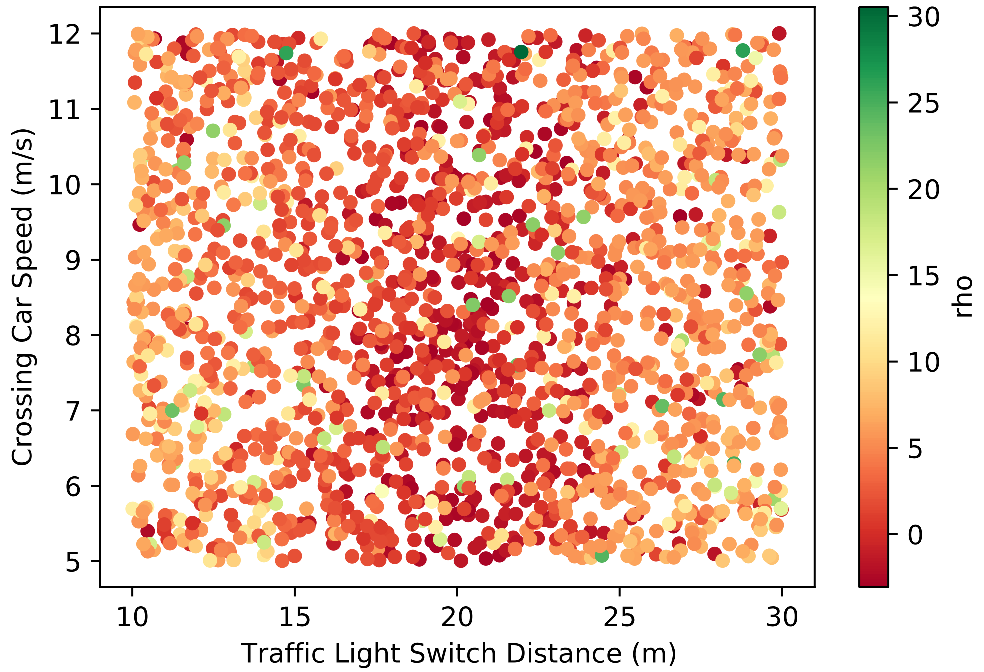
- Parallel, sequential, and more complex forms of composition

```
import StopAndStart, BadlyParkedCar

scenario StopStartWithParkedCar():
    compose:
        do StopAndStart(), BadlyParkedCar()


scenario StopStartThenParkedCar():
    compose:
        do StopAndStart()
        do BadlyParkedCar()


scenario StopStartThenParkedCar():
    compose:
        try:
            do StopAndStart()
        interrupt when ...:
            do BadlyParkedCar()
```

Falsification with a Dynamic Scenario

- Case study in CARLA [Fremont et al. 2021, arXiv:2010.06580]
- AV turns right at a 4-way intersection
  - Traffic light turns green as AV approaches, but another car runs the light
- Semantic features: intersection, traffic light timing, car speed



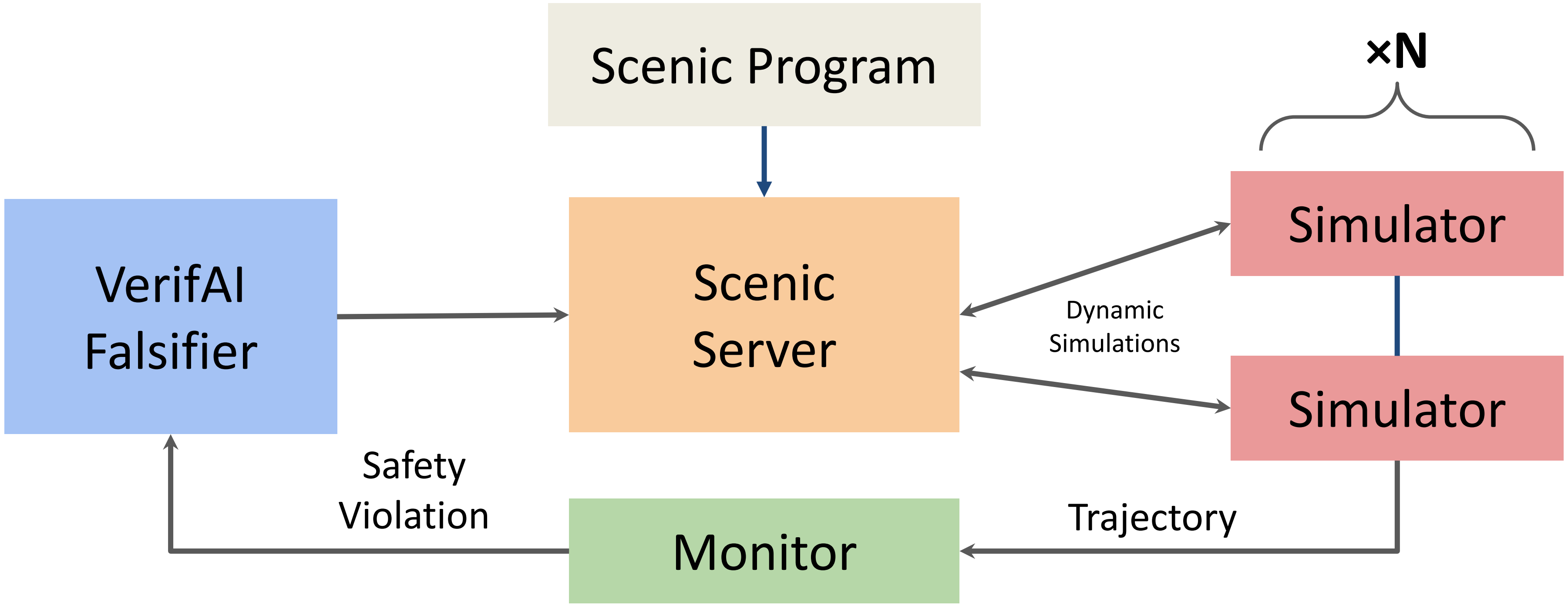


Falsification Demo

- Using simple Newtonian physics simulator built into Scenic
- To play with it yourself:
  - Install Python 3.8+ and Poetry (https://python-poetry.org/)
  - git clone https://github.com/BerkeleyLearnVerify/VerifAI
  - cd VerifAI; git checkout av-test-challenge; poetry install; poetry shell
  - cd ..; git clone https://github.com/BerkeleyLearnVerify/Scenic
  - cd Scenic; poetry install
  - cd ../VerifAI/experiments
  - python experiments.py --model newtonian --path intersection_01.scenic

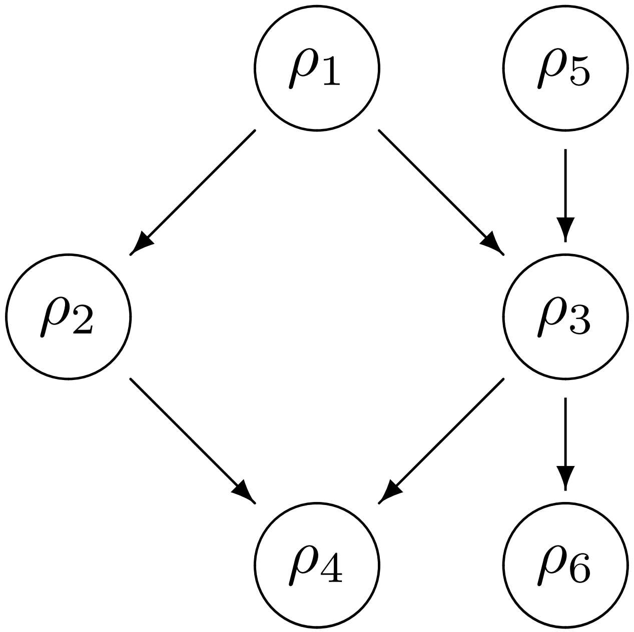
Parallel and Multi-Objective Falsification

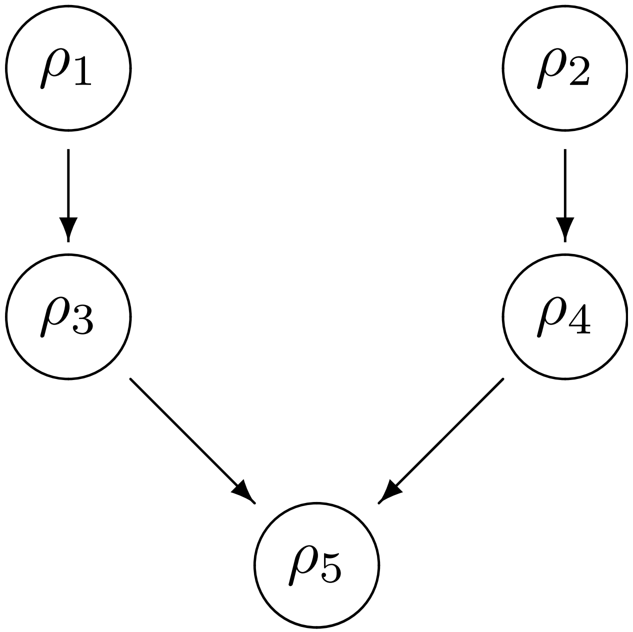
- New features in the VerifAI toolkit [Viswanadha et al., RV'21]
  - Run simulations in parallel for substantial speedups



Parallel and Multi-Objective Falsification

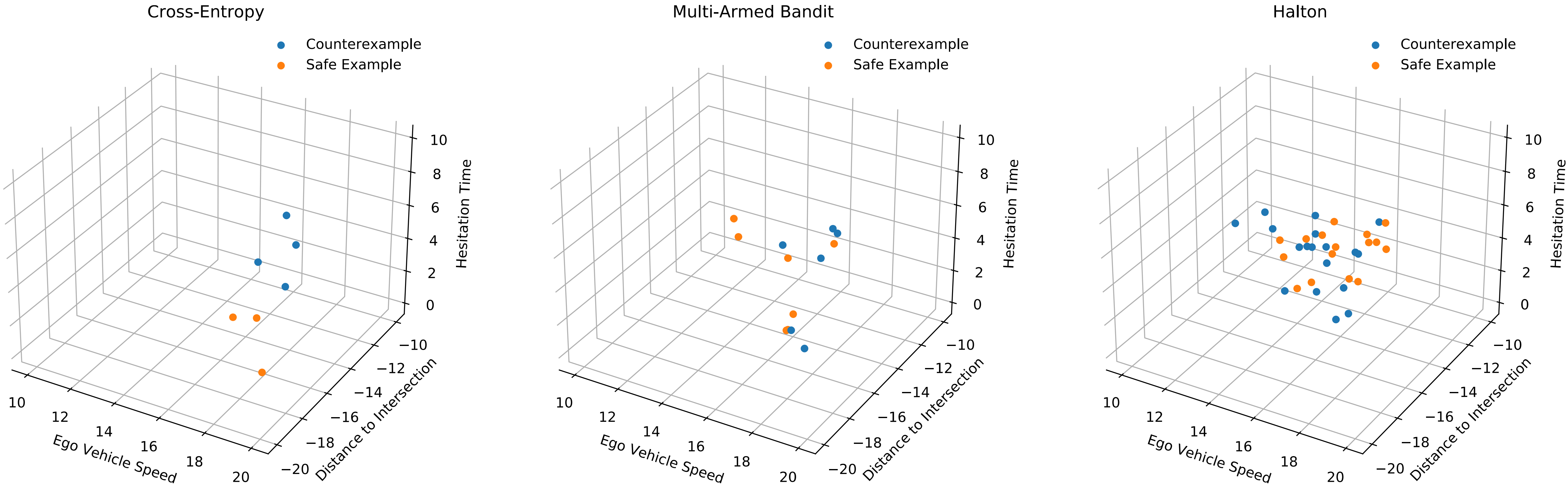
- New features in the VerifAI toolkit [Viswanadha et al., RV'21]
  - Falsify multiple specifications, with priorities





Parallel and Multi-Objective Falsification

- New features in the VerifAI toolkit [Viswanadha et al., RV'21]
  - Multi-armed bandit sampler trading off exploration and exploitation



Outline

- Overview of Scenic and VerifAI
  - Basic syntax of the Scenic language
- Falsification
  - Case study in the Webots simulator
- Dynamic Scenarios in Scenic
  - Case study in autonomous driving simulators (e.g., CARLA)
- Falsification → Debugging → Retraining
  - Case study in the X-Plane simulator
- Data-Driven Run-Time Monitor Generation with Scenic & VerifAI
  - Case study in the X-Plane simulator
- Conclusion

A Full Design Iteration using Scenic & VerifAI

- In addition to discovering failures, VerifAI can help debug and fix them

- Industrial case study on **TaxiNet**, a NN-based taxiing system [CAV 2020]
  - **Modeling** runway scenarios in SCENIC
  - **Falsifying** the system, finding scenarios when it violates its specification
  - **Debugging** to find distinct failures and their root causes
  - **Retraining** the system to eliminate failures and improve performance

TaxiNet

- Experimental autonomous aircraft taxiing system developed by Boeing
- Neural network uses camera image to estimate the *cross-track error*
  - CTE = distance from centerline
- System-level spec: plane must track centerline to within 1.5 meters

$$\varphi_{\text{eventually}} = \Diamond_{[0,10]} \Box (\text{CTE} \leq 1.5)$$

Neural Network

CTE estimate

Controller

Steering, throttle controls

Modeling and Falsification

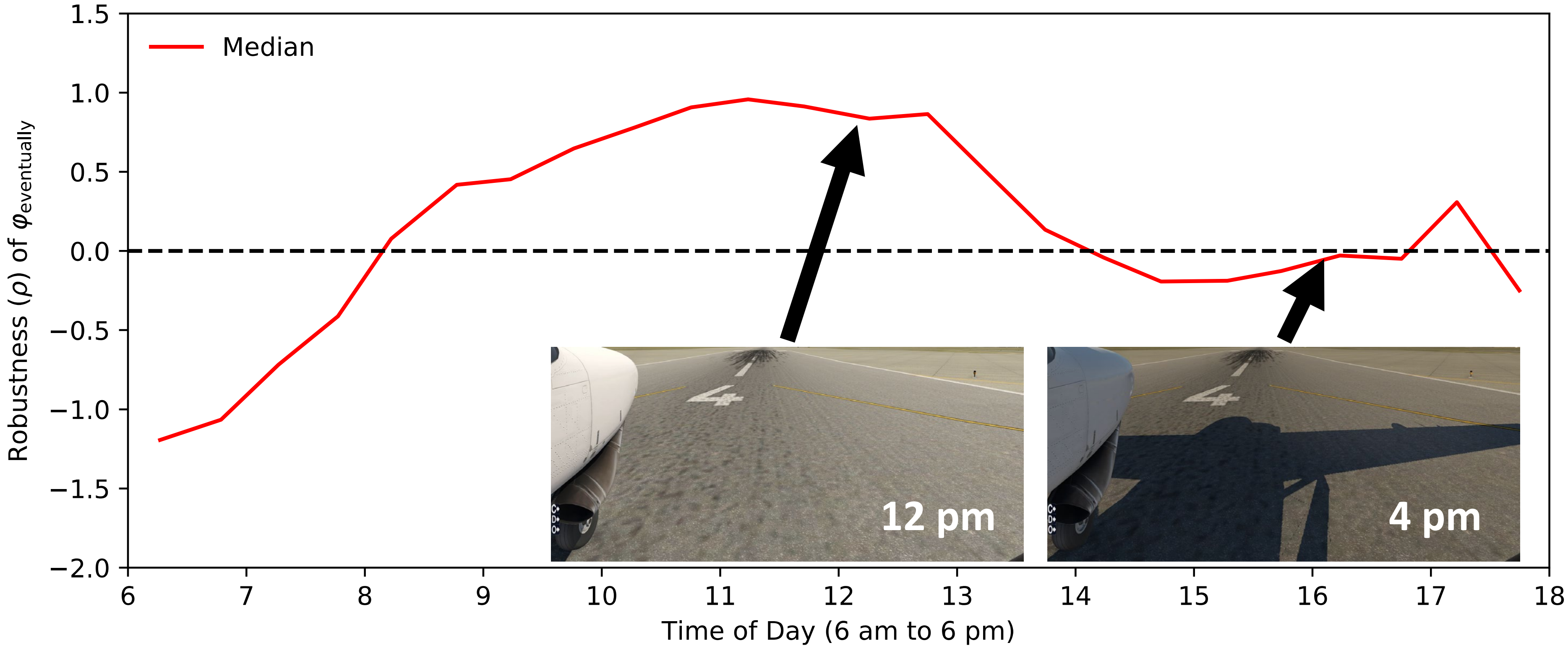
- Semantic features: time, clouds, rain, position/orientation of plane

```
# Time of day: from 6 am to 6 pm. (+8 to get GMT, as used by X-Plane)
param zulu_time = ((6, 18) + 8) * 60 * 60

# Rain: 1/3 of the time. Clouds: rain requires types 3-5; otherwise 0-5.
clouds_and_rain = Options({
    tuple([Uniform(0, 1, 2, 3, 4, 5), 0]): 2,  # no rain
    tuple([Uniform(3, 4, 5), (0.25, 1)]): 1    # 25% to 100% rain
})
param cloud_type = clouds_and_rain[0], rain_percent = clouds_and_rain[1]

# Plane: up to 8 m left/right, 2000 m down the runway, 30° left/right.
ego = Plane at (-8, 8) @ (0, 2000),
            facing (-30, 30) deg
```

- Falsification: out of ~4,000 simulations,
  - **45%** violated $\varphi_{\text{eventually}}$
  - **9%** left runway entirely

Counterexample Analysis

- Falsification found several types of failures, e.g. sensitivity to time



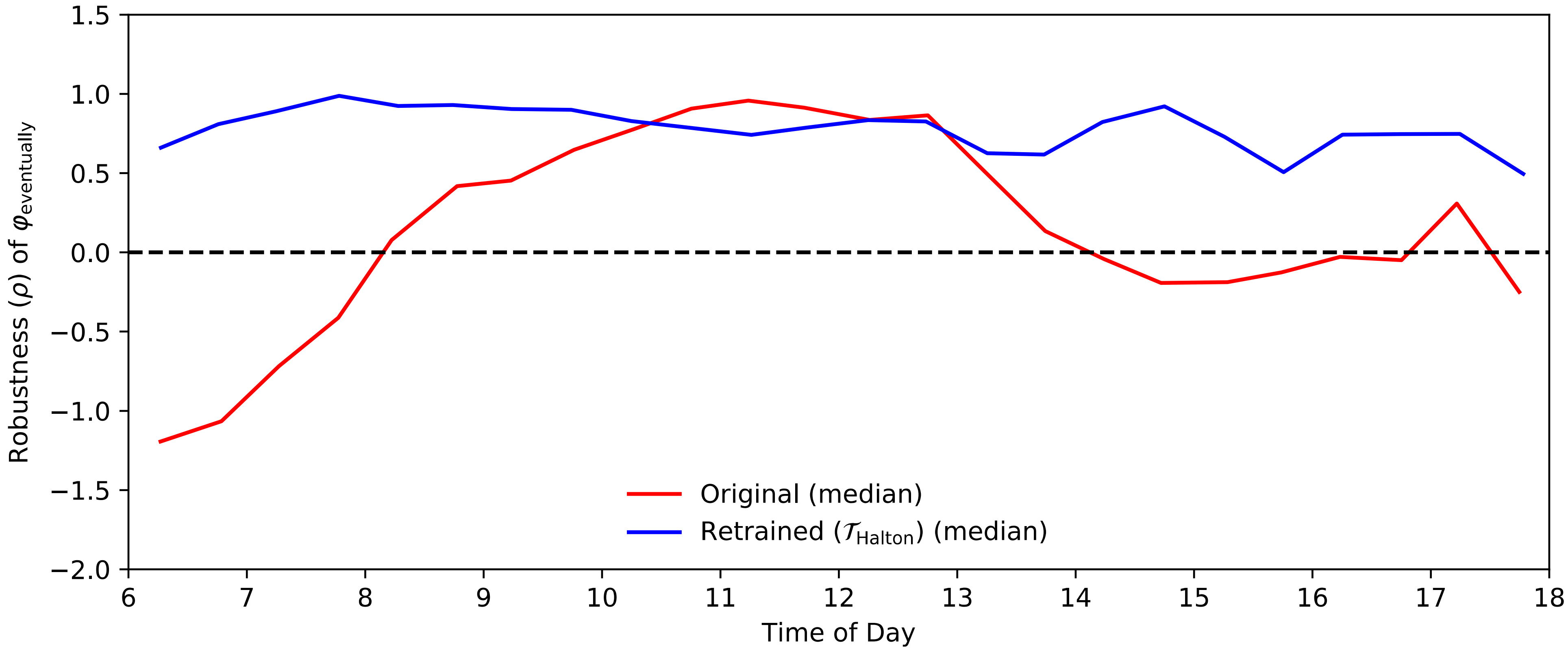
- Follow-up experiments confirmed root cause is the plane's shadow

Retraining

- Use VERIFAI to generate a new training set (same size as original)
- Obtained much better performance
  - **17%** violated $\varphi_{\text{eventually}}$ (vs. 45%)
  - **0.6%** left runway entirely (vs. 9%)

Retraining

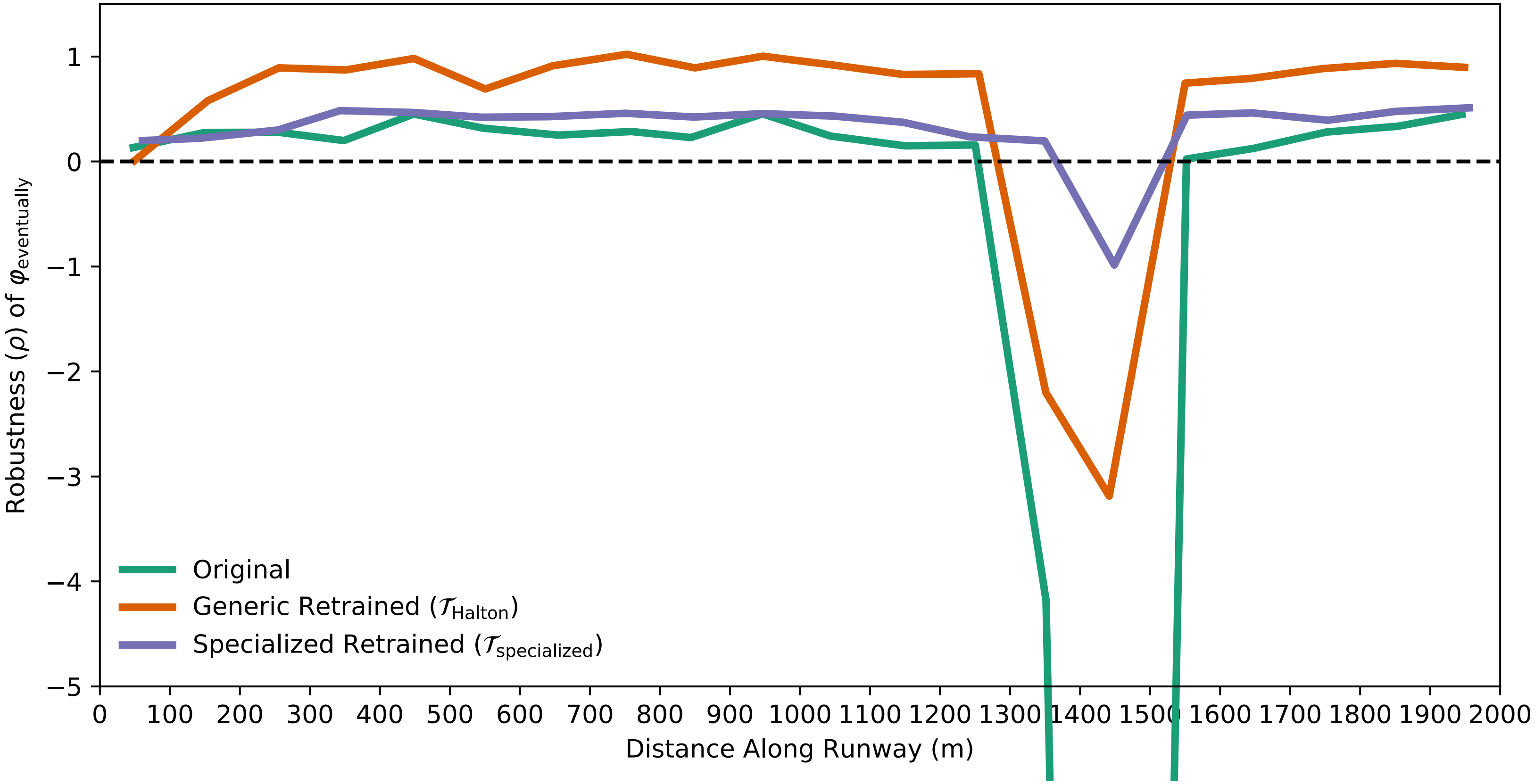
- Eliminated dependence on time of day



- Used cross-entropy method to *learn* good training distributions

Retraining

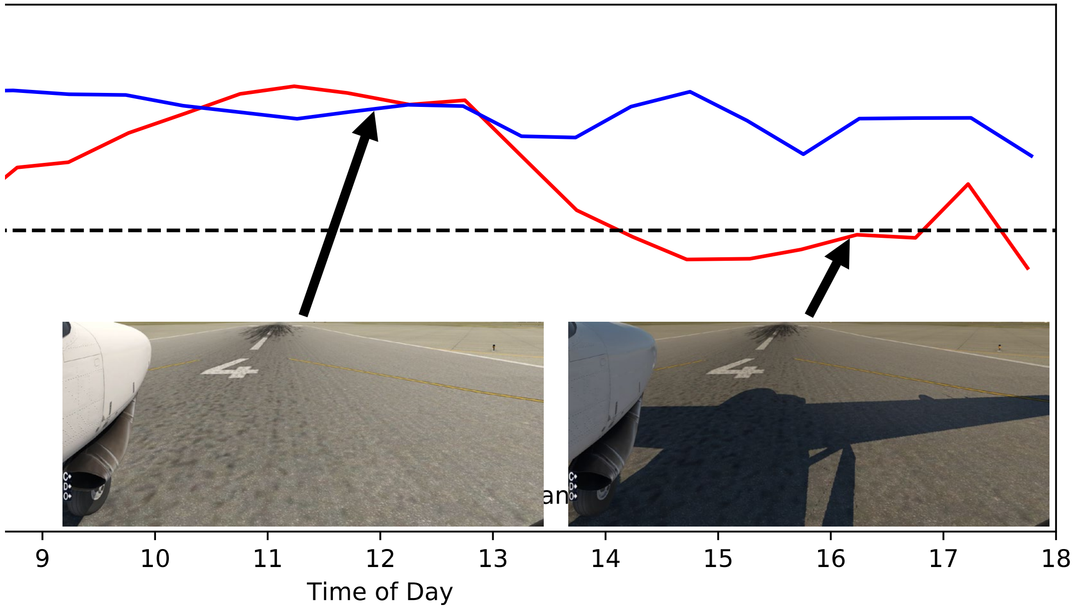
- Improved handling of runway intersections, but still problematic
- Can do better using specialized training
  - Concentrate training distribution around hardest points (using Scenic)
  - *Learn* a suitable distribution using cross-entropy optimization



Conclusion

- VERIFAI can be applied to realistic, industrial autonomous systems

- We used it to find bugs in TaxiNet, diagnose them, and eliminate some of them through more intelligent training set design



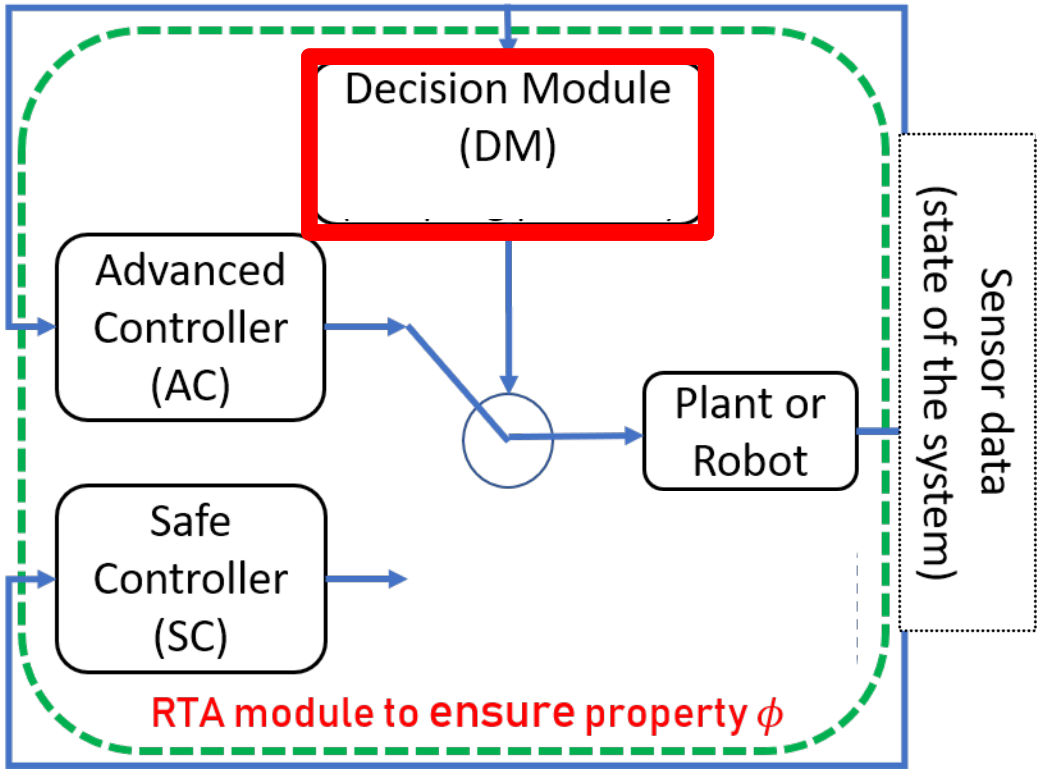
- But not all counterexamples can be eliminated through retraining
  - How can we use the results of falsification to generate runtime monitors?

Outline

- Overview of Scenic and VerifAI
  - Basic syntax of the Scenic language
- Falsification
  - Case study in the Webots simulator
- Dynamic Scenarios in Scenic
  - Case study in autonomous driving simulators (e.g., CARLA)
- Falsification → Debugging → Retraining
  - Case study in the X-Plane simulator
- Data-Driven Run-Time Monitor Generation with Scenic & VerifAI
  - Case study in the X-Plane simulator
- Conclusion

Simplex Architecture for Run-Time Assurance

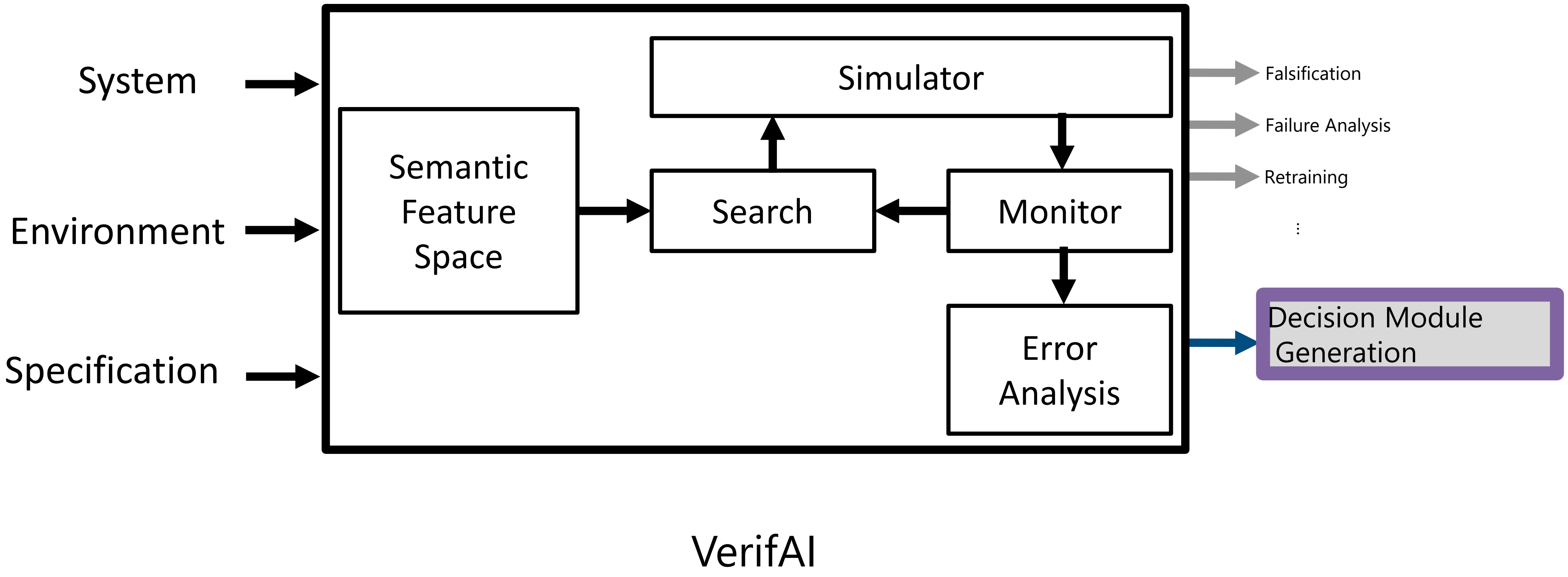
[Lui Sha, RTSS’98]



How do we generate the switching logic for the Decision Module as a Run-Time Monitor?

Already used in fault-tolerant CPS (e.g. avionics)

Extending VerifAI with a Generator for Decision Modules



Data-driven Monitor Generation On One Slide

- **Naive approach**: Generate positive and negative examples (negative = raise an alert).
- **Goal**: Generalise the negative examples to unseen traces, i.e. generate a decision module for raising an alert.

- Some Challenges:
  - High-dimensional alphabet/space
  - Relevant information may not be observable/reliable at runtime
  - Needs to be predictive

Data-driven Monitor Generation

Step 1: Data preparation

Step 2: Monitor generation

Step 3: Monitor implementation

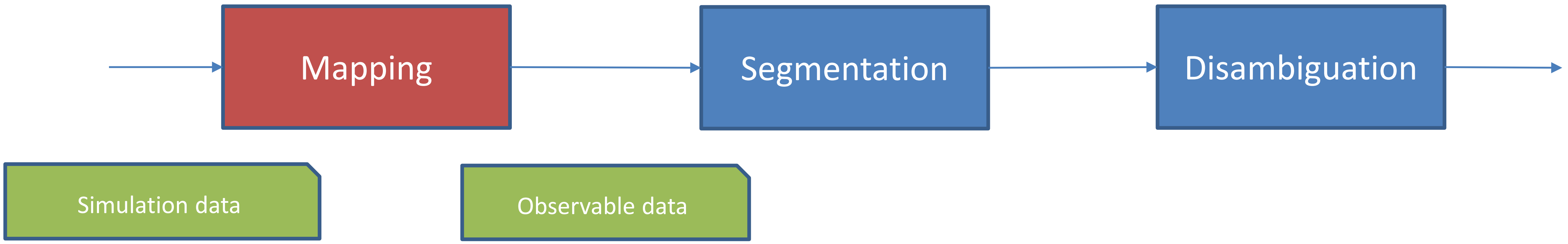
Data-driven Monitor Generation

Step 1: Data preparation

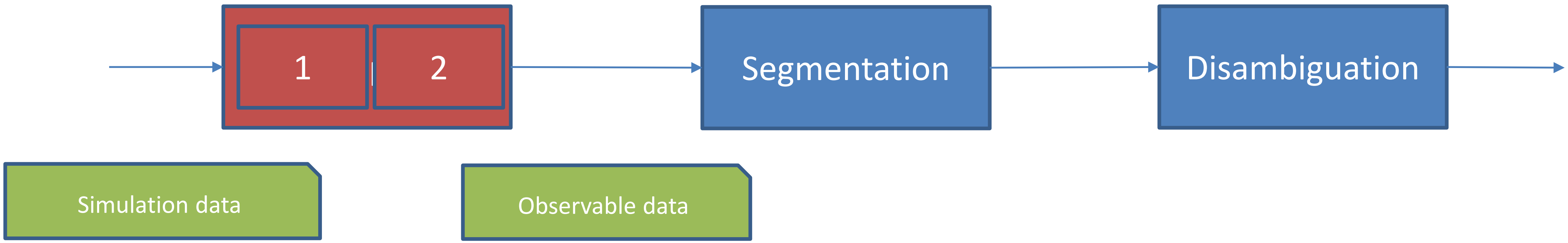
Step 2: Monitor generation

Step 3: Monitor implementation

Data-driven Monitor Generation: Obtaining Traces (Mapping)



Data-driven Monitor Generation: Obtaining Traces (Mapping)



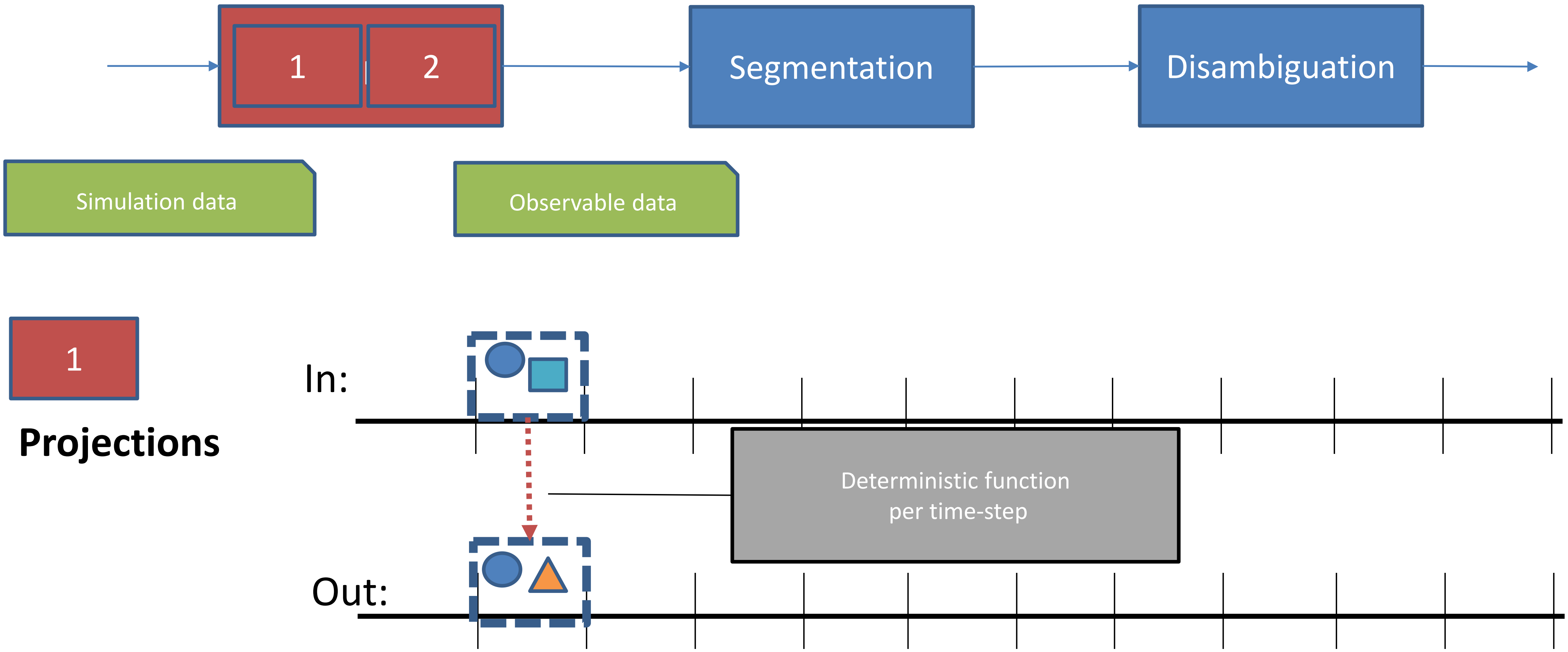
1

Projections

2

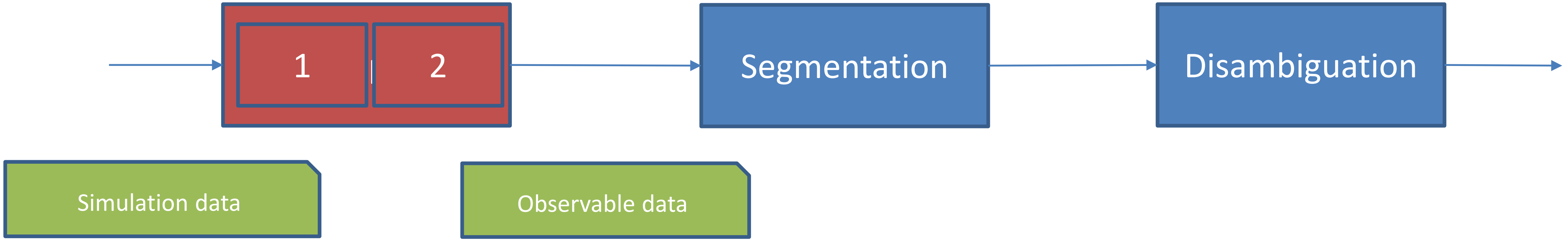
Filters

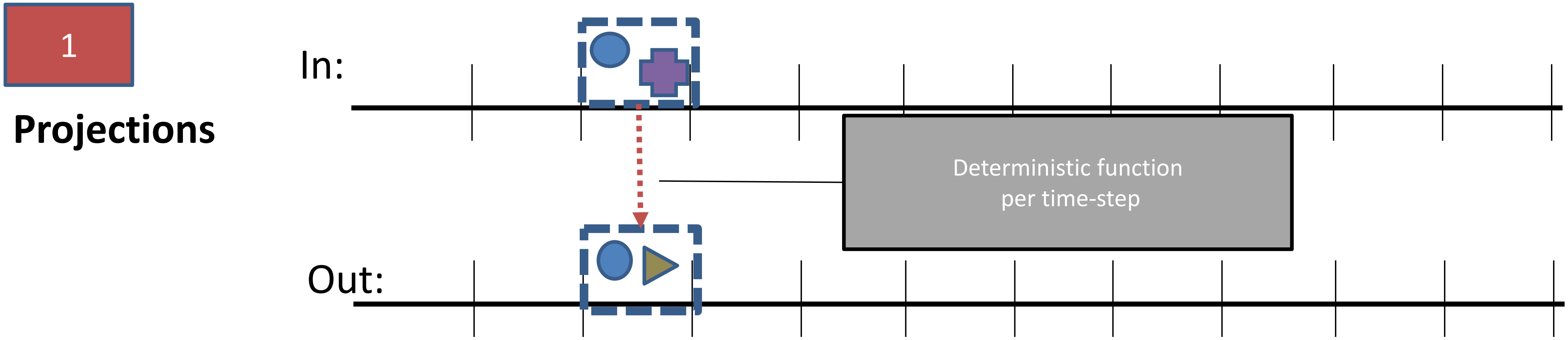
Data-driven Monitor Generation: Obtaining Traces (Mapping)



Rationale: Output alphabet consists of reliably observable data

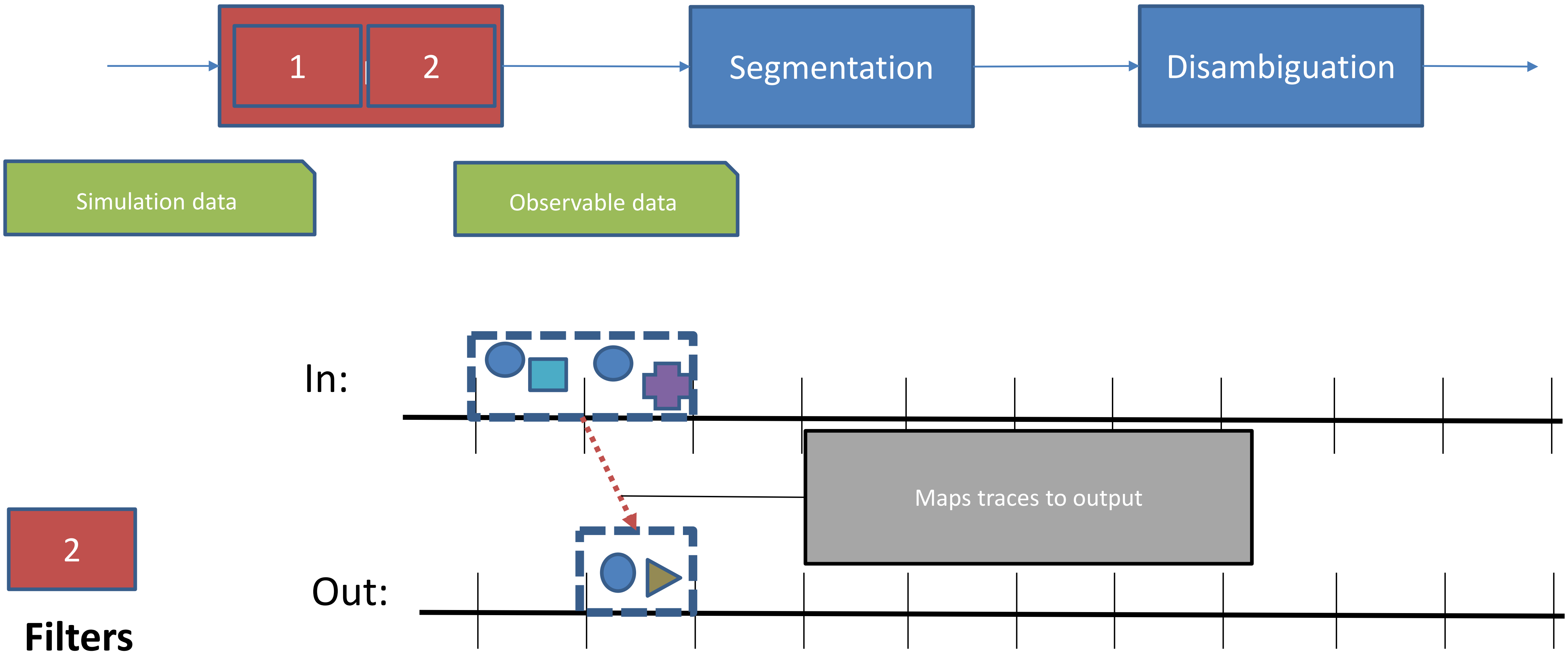
Data-driven Monitor Generation: Obtaining Traces (Mapping)



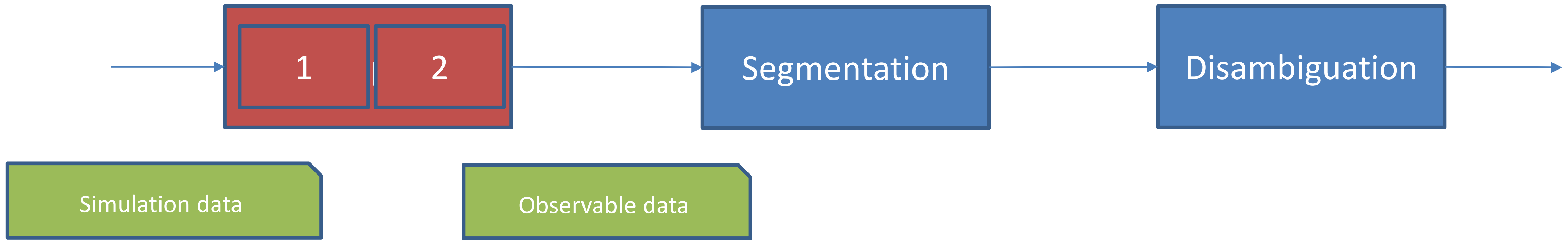


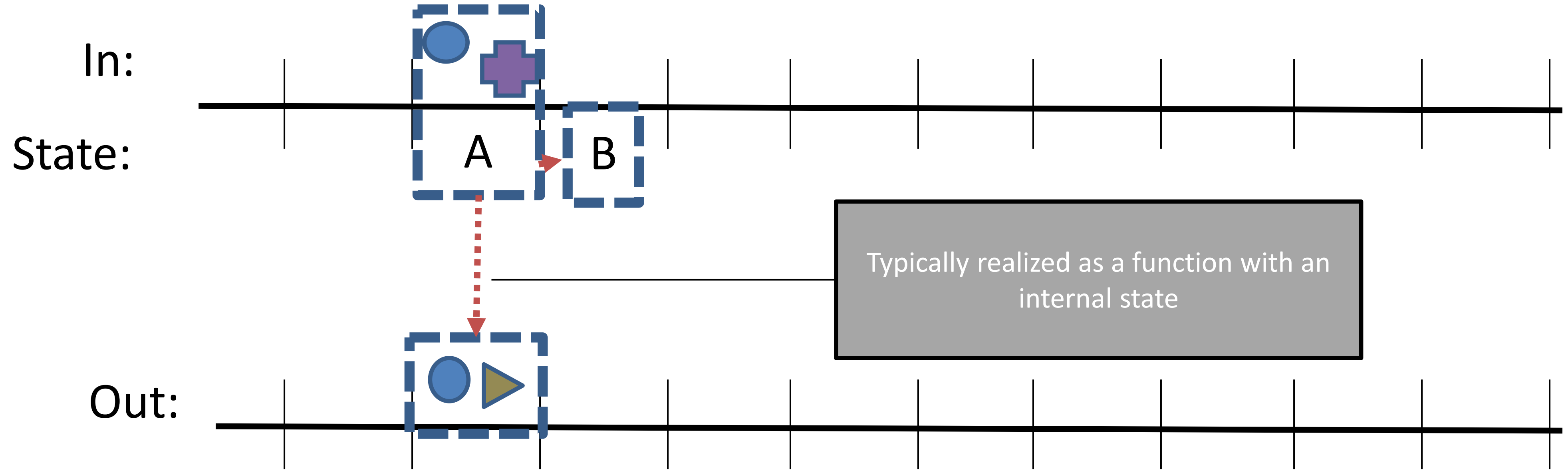
Rationale: Output alphabet consists of reliably observable data

Data-driven Monitor Generation: Obtaining Traces (Mapping)



Data-driven Monitor Generation: Obtaining Traces (Mapping)



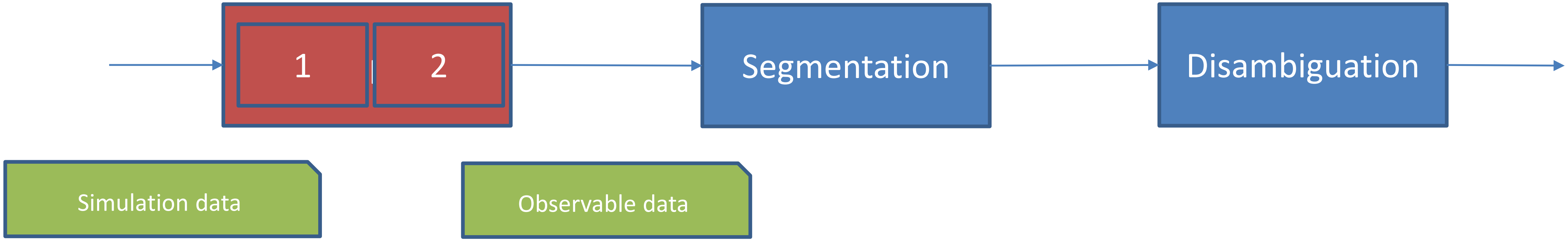


2

Filters

Rationale: explicitly add relevant filters

Data-driven Monitor Generation: Obtaining Traces (Mapping)



1

Projections

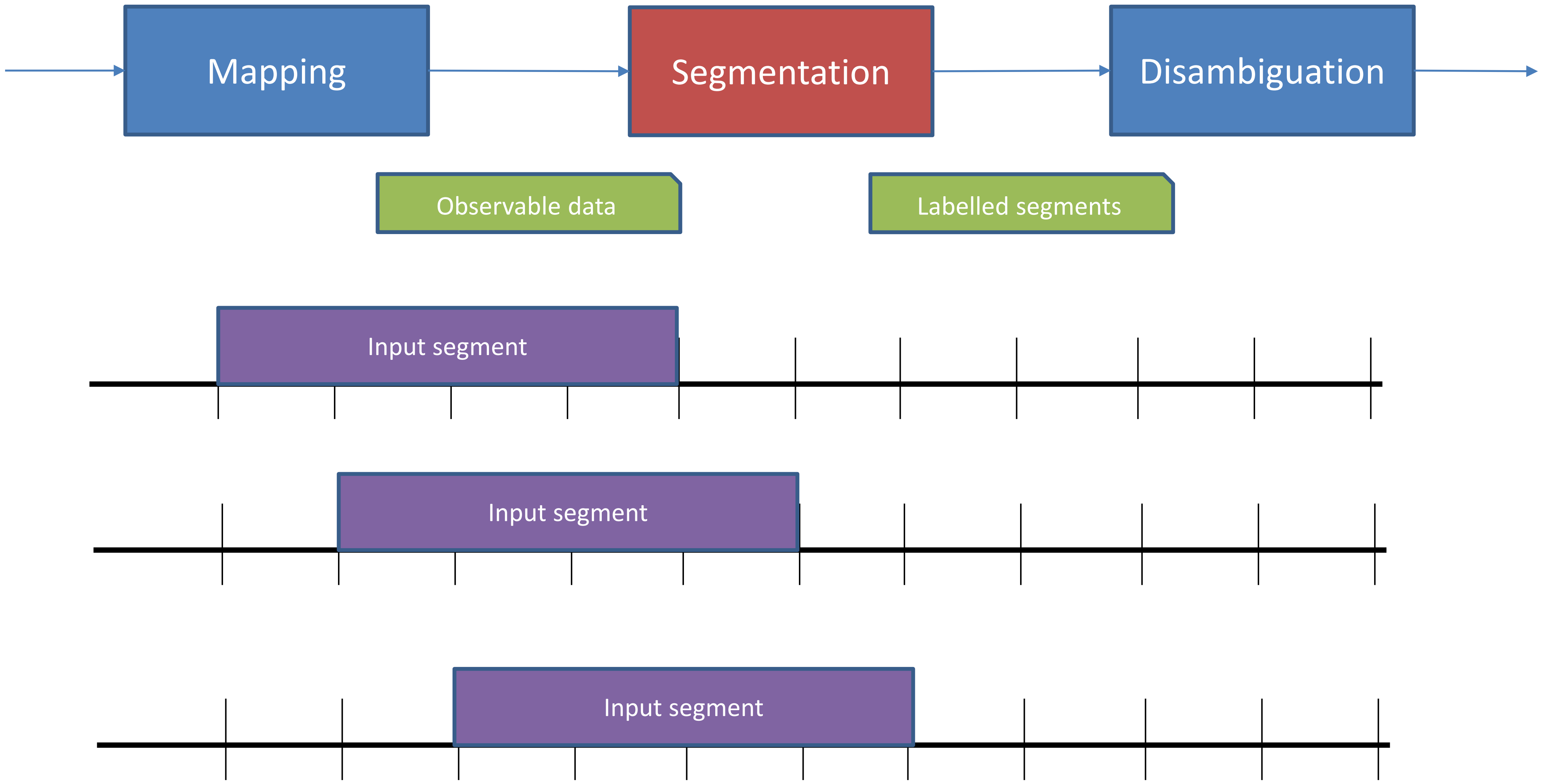
When rainy, do not include camera in observable data.
Never include temperature

2

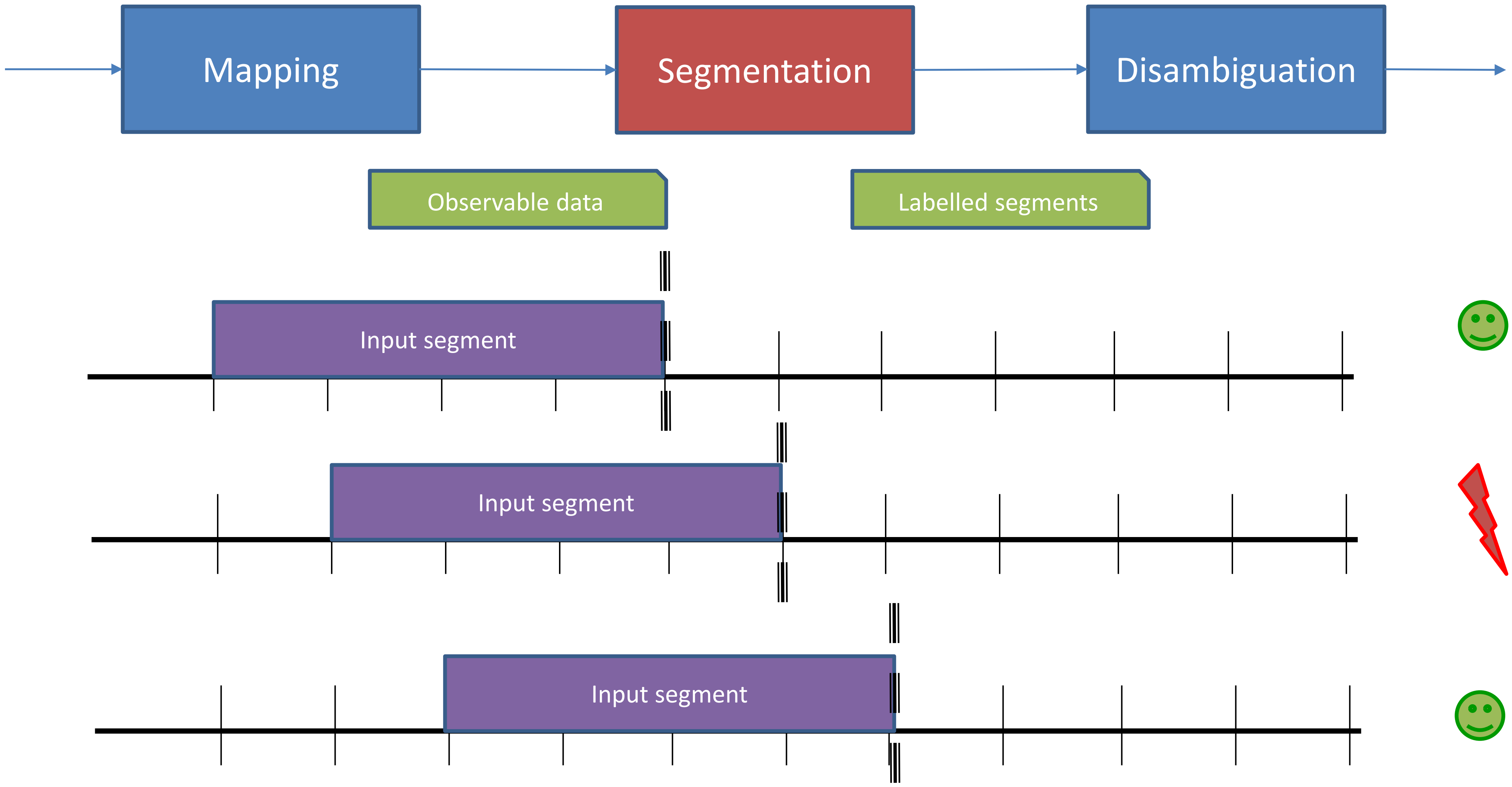
Filters

Aggregate deviations in the past
Smoothen GPS position

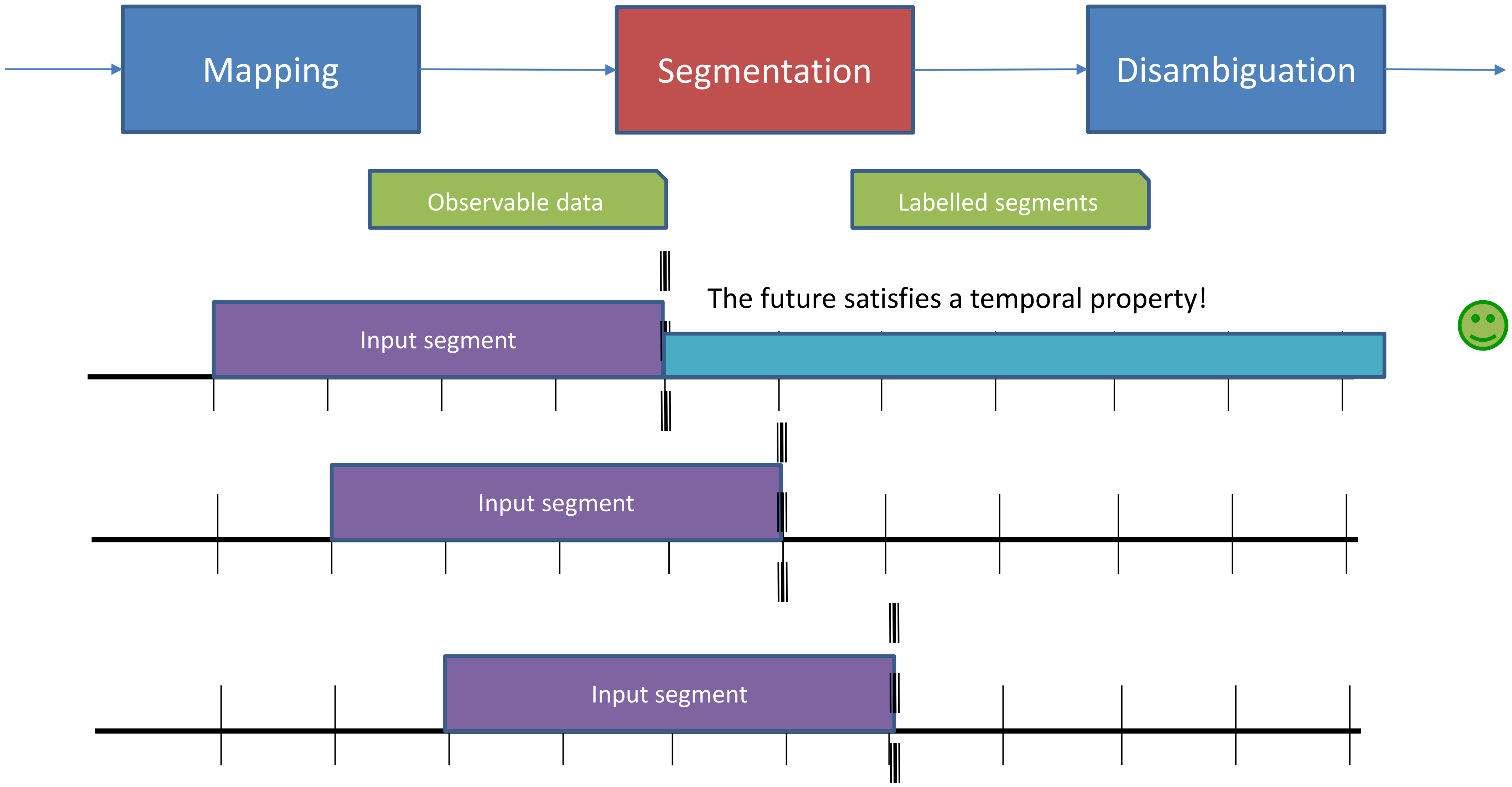
Data-driven Monitor Generation: Obtaining Traces (Segments)



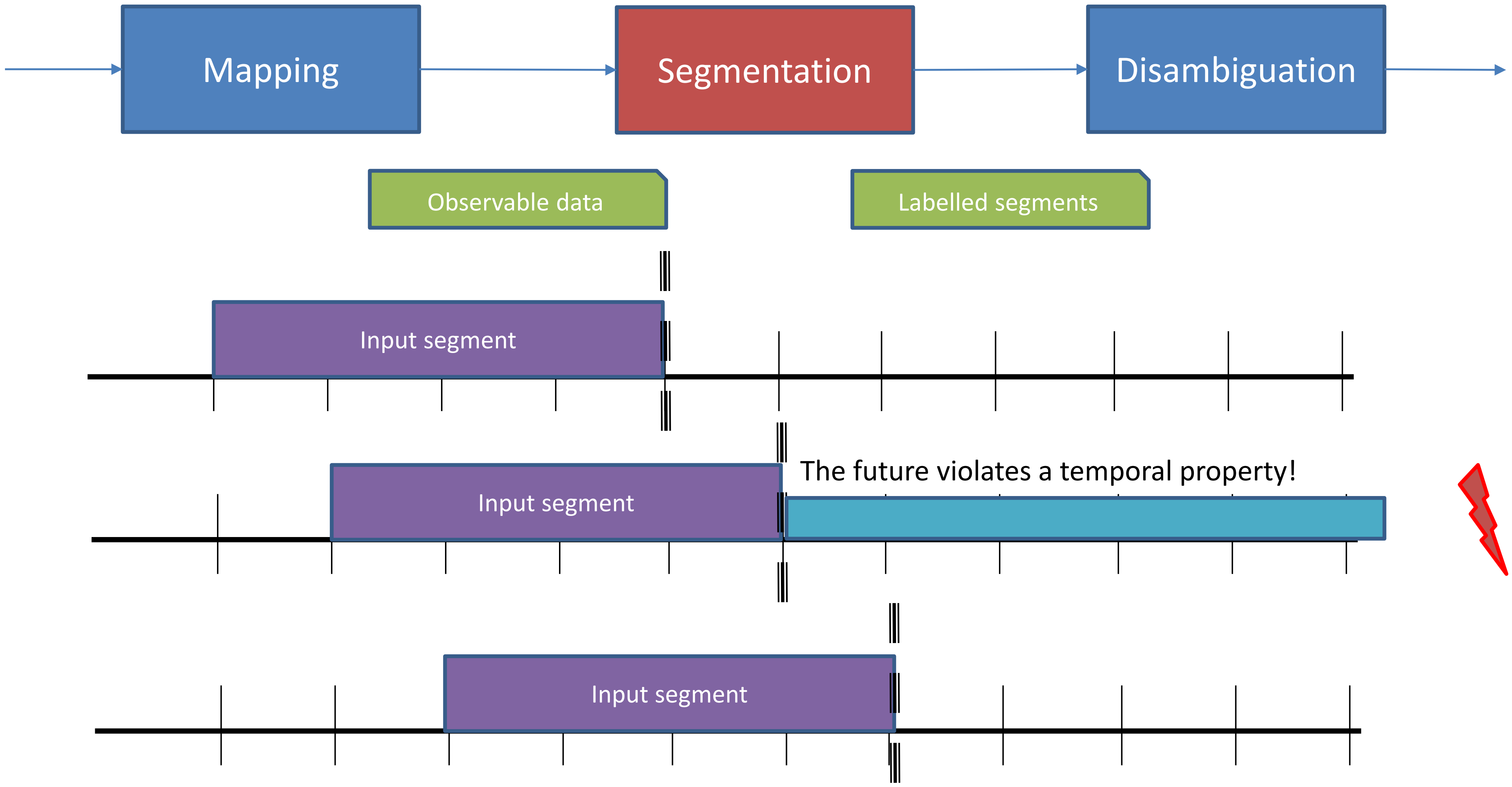
Data-driven Monitor Generation: Obtaining Traces (Segments)



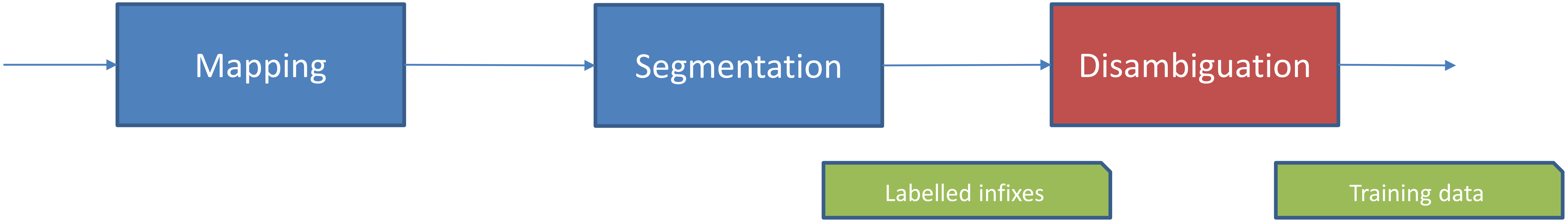
Data-driven Monitor Generation: Obtaining Traces (Segments)



Data-driven Monitor Generation: Obtaining Traces (Segments)



Data-driven Monitor Generation: Obtaining Traces



Handle duplicates: either conservatively or quantitatively

Data-driven Monitor Generation

Step 1: Data preparation

Result: Obtained finitely many positive and negative examples

Step 2: Monitor generation

Step 3: Monitor implementation

Data-driven Monitor Generation

Step 1: Data preparation

Result: Obtained finitely many positive and negative examples

Step 2: Monitor generation

3 Aspects: Implementability, Quantitative Correctness, Trustworthiness

Step 3: Monitor implementation

Data-driven Monitor Generation: Discussion and Desiderata

- Implementability

- Quantitative correctness

- Trustworthiness

Data-driven Monitor Generation: Discussion and Desiderata

- Implementability
  - Realizability
  - Performance
  - AI systems are often computationally heavy
- Quantitative correctness
- Trustworthiness

Data-driven Monitor Generation: Discussion and Desiderata

- Implementability
  - Realizability
  - Performance
  - AI systems are often computationally heavy
- Quantitative correctness
  - Overapproximation (e.g. only accepting seen traces) is typically too conservative
  - Quantify false positives and false negatives differently
- Trustworthiness

Data-driven Monitor Generation (Exact vs Approximate)

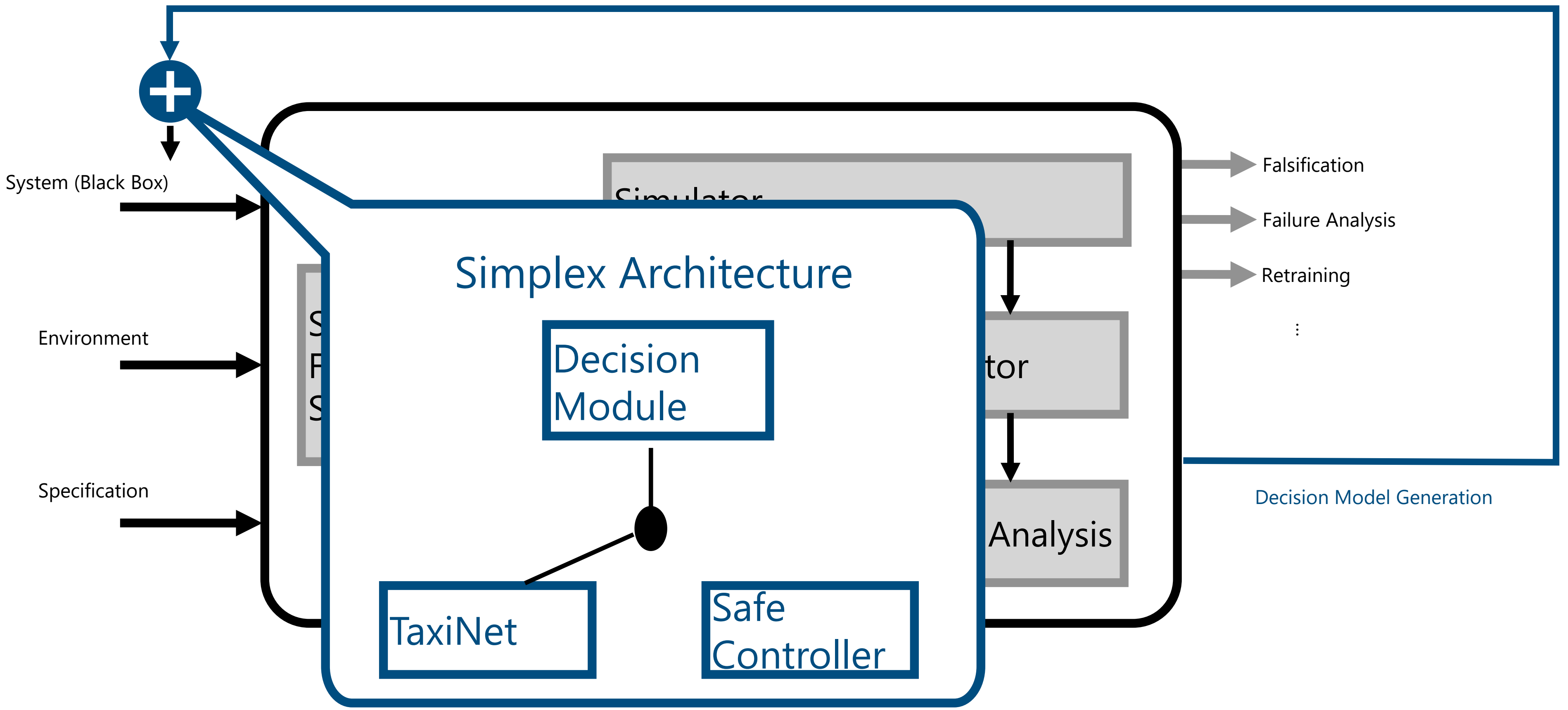
- Exact learning:
  - Guaranteed to be perfect on training set
  - May be overfitting, no guarantees outside training set
- PAC learning:
  - May be arbitrarily off (although this is unlikely)
  - Typically is correct in most cases

VerifAI-monitor generation currently allows using automata learning, decision tree learning and neural network classifiers

Data-driven Monitor Generation: Discussion and Desiderata

- Implementability
  - Realizability
  - Performance
  - AI systems are often computationally heavy
- Quantitative correctness
  - Overapproximation (e.g. only accepting seen traces) is typically too conservative
  - Quantify false positives and false negatives differently
- Trustworthiness
  - Quantitative correctness statistical -> Monitors should make the system more trustworthy
  - Monitor-in-the-loop testing

Monitor in the loop



Data-driven Monitor Generation

Step 1: Data preparation

Result: Obtained finitely many positive and negative examples

Step 2: Monitor generation

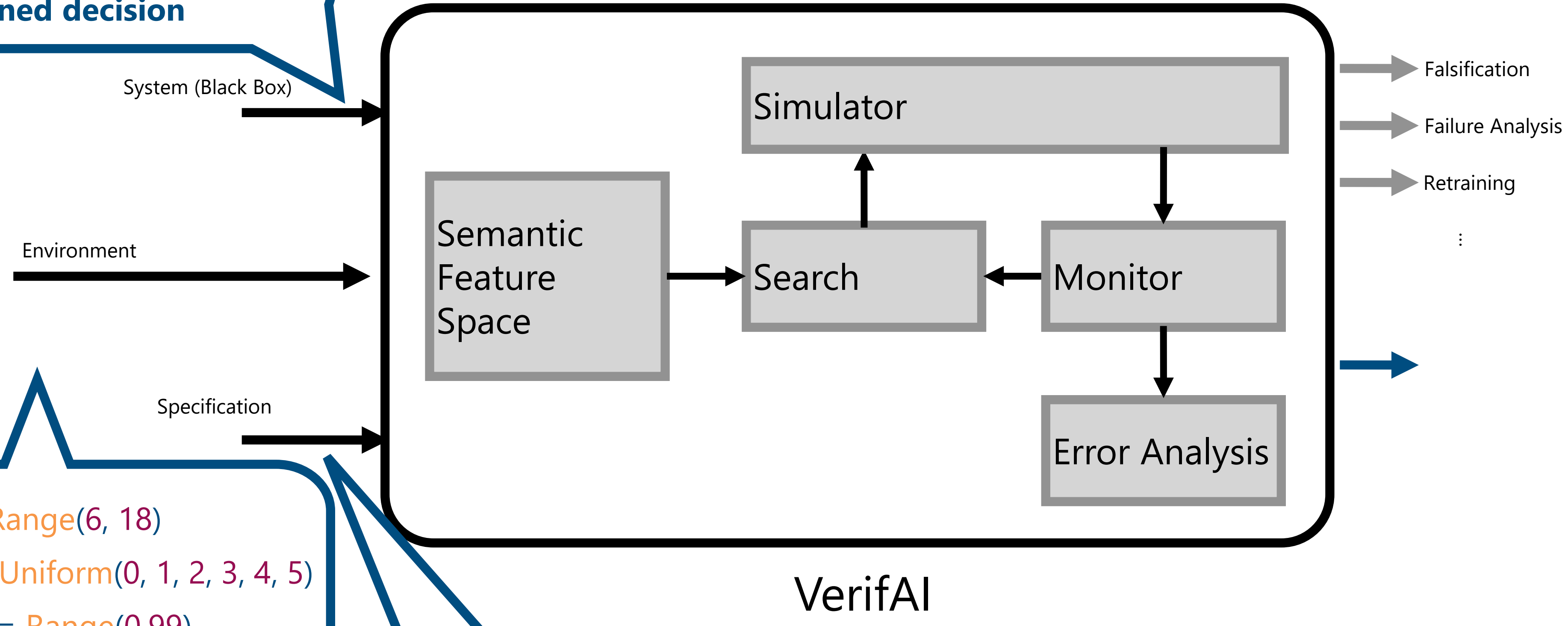
Result: Obtained logic that increases system trustworthiness
(based on formal & reproducible empirical evidence)

Step 3: Monitor implementation

Leverage work by the runtime verification community!

Application to TaxiNet

A controller that uses TaxiNet to predict distance to centerline **including a learned decision module**



local_time = Range(6, 18)

cloud_type = Uniform(0, 1, 2, 3, 4, 5)

rain_percent' = Range(0,99)

ego = Plane at Range(-8, 8) @ 0

Distance to centerline must be below a threshold

$$\Diamond_{[0,10]}\Box(\mathrm{CTE} \leq 2.5)$$

Learning Decision Modules over “Static” Features

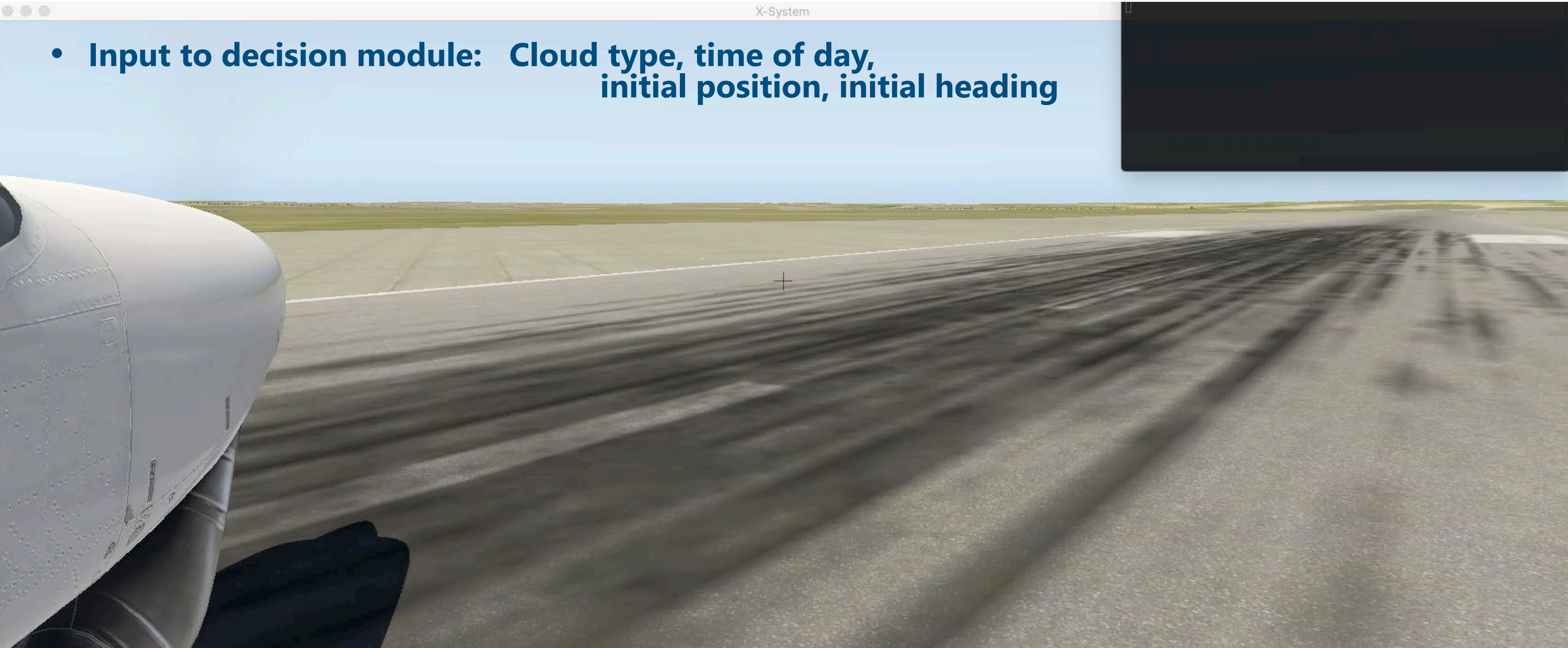
Make decision at beginning of simulation: decision made based on initial values

- **Input to decision module: Cloud type, time of day, initial position, initial heading**

Learning Decision Modules over "Static" Features

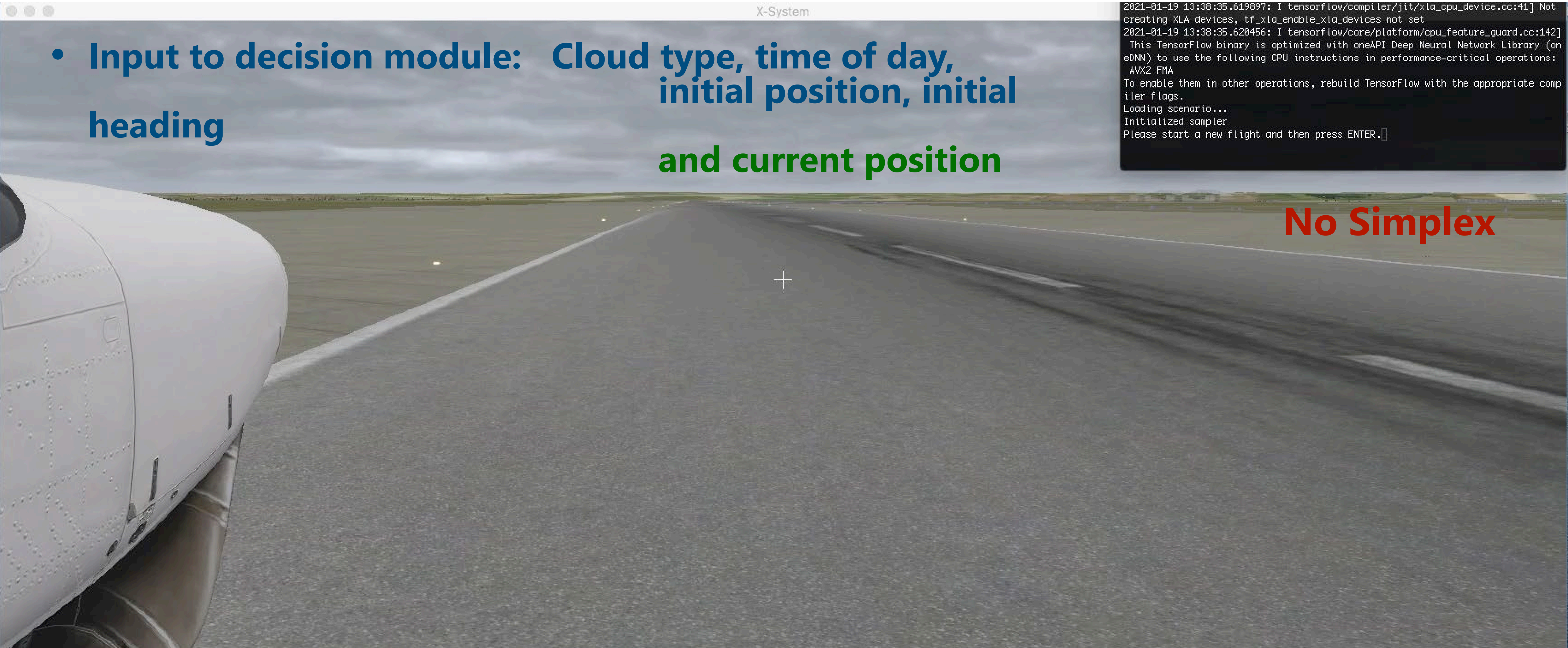
Make decision at beginning of simulation: decision made based on initial values

- **Input to decision module: Cloud type, time of day, initial position, initial heading**



Learning Decision Modules over "Dynamic" Features

Make decision at beginning of simulation: decision made based on recent history



- **Input to decision module: Cloud type, time of day, initial position, initial heading and current position**

No Simplex

Learning Decision Modules over "Dynamic" Features

Make decision at beginning of simulation: decision made based on recent history

- **Input to decision module: Cloud type, time of day, initial position, initial heading and current position**

Summary

- **Learn decision modules** from data sampled and processed with VerifAI
- **Evaluate system with the monitor** within VerifAI
- Plenty of **open and ongoing topics**:
  - Create efficient implementations from the logic described by our monitors
  - Feature selection for monitors
  - Use information from scenic-program or other models
  - Integrating learning

- Happy to cooperate!

Outline

- Overview of Scenic and VerifAI
  - Basic syntax of the Scenic language
- Falsification
  - Case study in the Webots simulator
- Dynamic Scenarios in Scenic
  - Case study in autonomous driving simulators (e.g., CARLA)
- Falsification → Debugging → Retraining
  - Case study in the X-Plane simulator
- Data-Driven Run-Time Monitor Generation with Scenic & VerifAI
  - Case study in the X-Plane simulator
- Conclusion

Scenic and VerifAI: Summary of Features and Use Cases

- Classes, Objects, Geometry, and Distributions
- Local Coordinate Systems
- Readable, Flexible Specifiers
- Declarative Hard & Soft Constraints
- Externally-Controllable Parameters
- Agent Actions and Behaviors, Interrupts, Termination
- Monitors, Temporal Constraints
- Scenario Composition

…

- Synthetic Data Generation
- Test Generation, Fuzz Testing
- Requirements Specification
- Falsification
- Debugging and Error Explanation
- Data Augmentation
- Goal-Directed Parameter Synthesis
- Run-Time Monitor Generation

…

Documentation on Scenic and VerifAI – linked from GitHub

Scenic
latest

Search docs

- Getting Started with Scenic
- Scenic Tutorial
- Guide to Scenic Syntax
- Scenic Syntax Reference
- Supported Simulators
- Interfacing to New Simulators
- Scenic Internals
- Publications Using Scenic
- Credits

Welcome to Scenic's documentation!

Scenic is a domain-specific probabilistic programming language for modeling the environments of cyber-physical systems like robots and autonomous cars. A Scenic program defines a distribution over *scenes*, configurations of physical objects and agents; sampling from this distribution yields concrete scenes which can be simulated to produce training or testing data.

Scenic was designed and implemented by Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. For a description of the language and some of its applications, see our PLDI 2019 paper; a more in-depth discussion is in Chapters 5 and 8 of this thesis. Our publications page lists additional papers using Scenic.

VerifAI
latest

Search docs

- Getting Started with VerifAI
- Basic Usage
- Tutorial / Case Studies
- Feature APIs in VerifAI
- Search Techniques
- Publications Using VerifAI

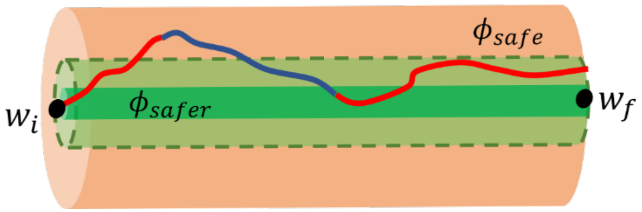
Docs » Welcome to VerifAI's documentation!   Edit on GitHub

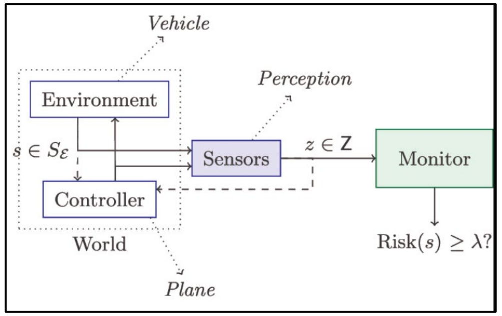
Welcome to VerifAI's documentation!

VerifAI is a software toolkit for the formal design and analysis of systems that include artificial intelligence (AI) and machine learning (ML) components. VerifAI particularly seeks to address challenges with applying formal methods to perception and ML components, including those based on neural networks, and to model and analyze system behavior in the presence of environment uncertainty. The current version of the toolkit performs intelligent simulation guided by formal models and specifications, enabling a variety of use cases including temporal-logic falsification (bug-finding), model-based systematic fuzz testing, parameter synthesis, counterexample analysis,

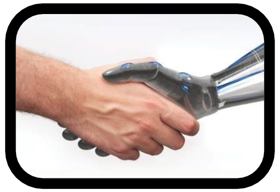
Ongoing/Future Directions

Run-Time Assurance
SOTER framework based on Simplex architecture [DSN 2019, RV 2020]



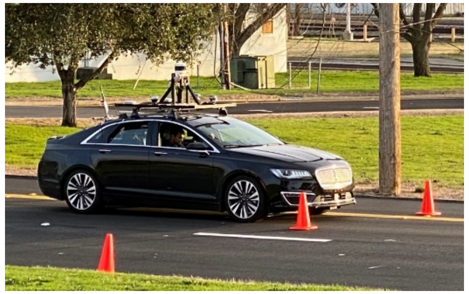


Run-Time Monitoring in MDPs
monitoring partially observable systems with nondeterministic and probabilistic dynamics [CAV 2021]



Verified Human-Robot Collaboration
Learning Specifications from Demonstrations, Interaction-Aware Control, etc. [IROS 2016, NeurIPS 2018, CAV 2020]

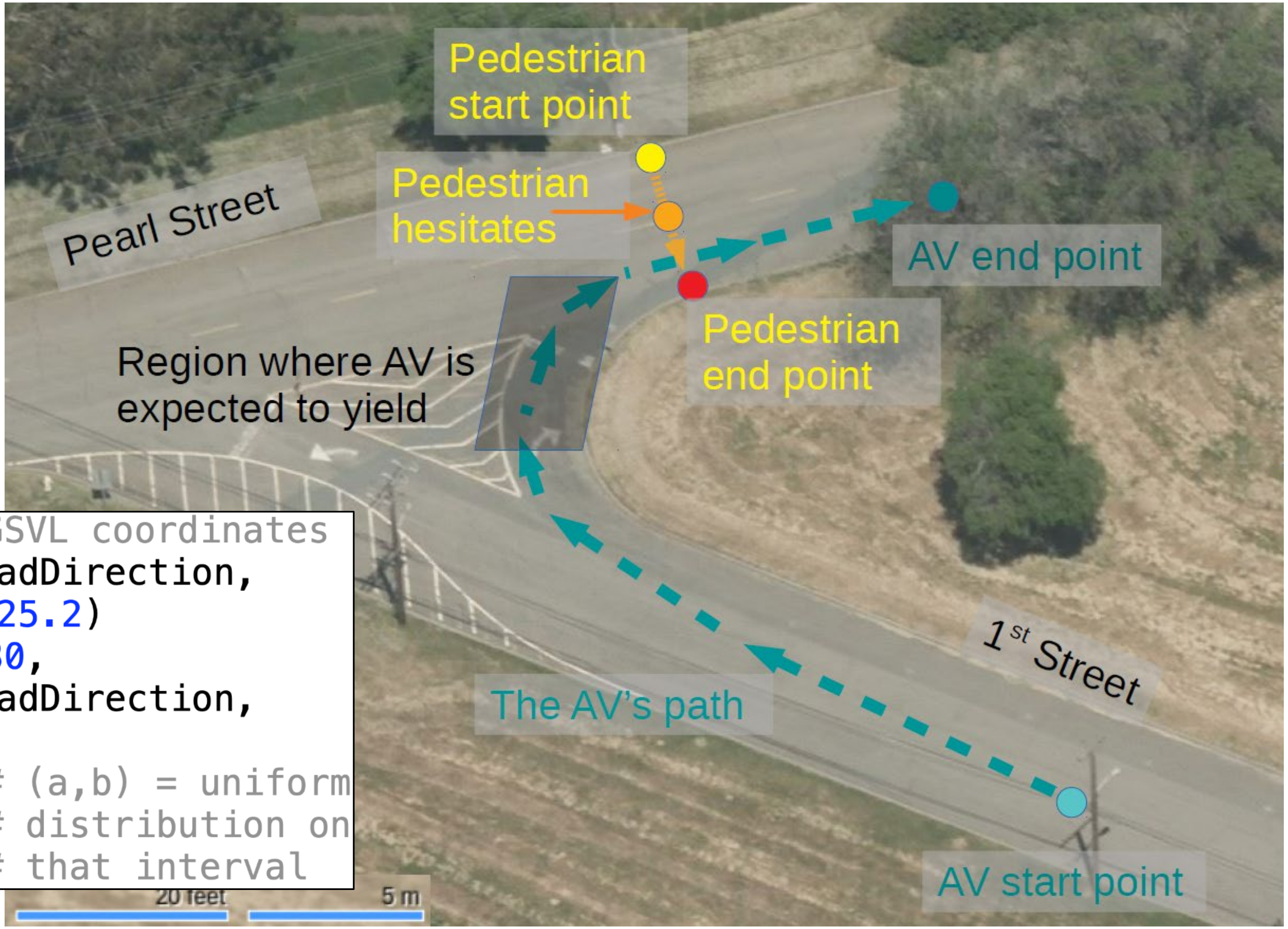
Explaining Success/Failures of Deep Learning
Automated approach using Scenic [CVPR 2020]



Bridging Simulation & Real World
Metrics to compare simulated vs real behaviors [HSCC 2019]
Using falsification to design real world tests [ITSC 2020]

Example Scenario: AV making right turn, pedestrian crossing

Lincoln MKZ running Apollo 3.5




```
ego = EgoCar at 38.6 @ 183.9,  # LGSVL coordinates
       facing 10 deg relative to roadDirection,
       with behavior DriveTo(40 @ 225.2)
ped = Pedestrian at 19.782 @ 225.680,
       facing 90 deg relative to roadDirection,
       with behavior Hesitate,
       with startDelay (7, 15),    # (a,b) = uniform
       with walkDistance (4, 7),   # distribution on
       with hesitateTime (1, 3)    # that interval
```

Snippet of Scenic program

Safety in Simulation → Safety on the Road? [Fremont et al., ITSC 2020]

Unsafe in simulation → unsafe on the road: **62.5% (incl. collision)**

Safe in simulation → safe on the road: **93.5% (no collisions)**

[joint work with American Automobile Association and LG Electronics]

Conclusion: Towards Verified AI/ML based Autonomy

Challenges		Core Principles
1. Environment (incl. Human) Modeling	→	Data-Driven, Introspective, Probabilistic Modeling
2. Specification	→	Start with System-Level Specification, then Component Spec (robustness, …)
3. Learning Systems Complexity	→	Abstraction, Semantic Representation, and Explanations
4. Efficient Training, Testing, Verification	→	Compositional Analysis and Semantics-directed Search/Training
5. Design for Correctness	→	Oracle-Guided Inductive Synthesis; Run-Time Assurance

Exciting Times Ahead!!! Thank you!

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514.

List of References

- Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards Verified Artificial Intelligence. ArXiv e-prints, July 2016.
- Hazem Torfah, Sebastian Junges, Daniel J. Fremont, Sanjit A. Seshia: Formal Analysis of AI-Based Autonomy: From Modeling to Runtime Assurance. RV 2021
- Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A Language for Scenario Specification and Scene Generation. In Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI), June 2019.
- Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In 31st International Conference on Computer Aided Verification (CAV), July 2019.
- Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems. In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2019.
- Daniel J. Fremont, Johnathan Chiu, Dragos D. Margineantu, Denis Osipychev, and Sanjit A. Seshia. Formal Analysis and Redesign of a Neural Network-Based Aircraft Taxiing System with VerifAI. In 32nd International Conference on Computer Aided Verification (CAV), July 2020.
- Daniel J. Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A. Seshia, Atul Acharya, Xantha Bruso, Paul Wells, Steve Lemke, Qiang Lu, and Shalin Mehta. Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World. In 23rd IEEE International Conference on Intelligent Transportation Systems (ITSC), September 2020.
- Sumukh Shivakumar, Hazem Torfah, Ankush Desai, and Sanjit A. Seshia. SOTER on ROS: A Run-Time Assurance Framework on the Robot Operating System. In 20th International Conference on Runtime Verification (RV), October 2020.
- Kesav Viswanadha, Edward Kim, Francis Indaheng, Daniel J. Fremont, Sanjit A. Seshia: Parallel and Multi-objective Falsification with Scenic and VerifAI. RV 2021