2022 ICSA
Honolulu, HI
March 12-15

# ROSDiscover: Statically Detecting Run-Time Architecture Misconfigurations in Robotics Systems*

Christopher S. Timperley, Tobias Dürschmid, Bradley Schmerl, David Garlan, Claire Le Goues

Carnegie Mellon University, Pittsburgh, PA, USA

* This work has been supported in part by NASA (Award 80NSSC20K1720), AFRL (Award 19-PAF00747), and the NSF (Award CCF-1750116)

# What is ROS?

ROS

- Robot Operating System
- Popular framework for **component-based** robot software
- Used by Amazon, Bosch, CAT, and many other companies
- 200,000+ software projects
- Library infrastructure of **reusable software packages**
- Uses **late binding** for architectural connectors
  - Flexible & extensible but **error-prone**

# Here is a Real Architecture Misconfiguration Bug from Autoware.AI (Existed for 2 Months)

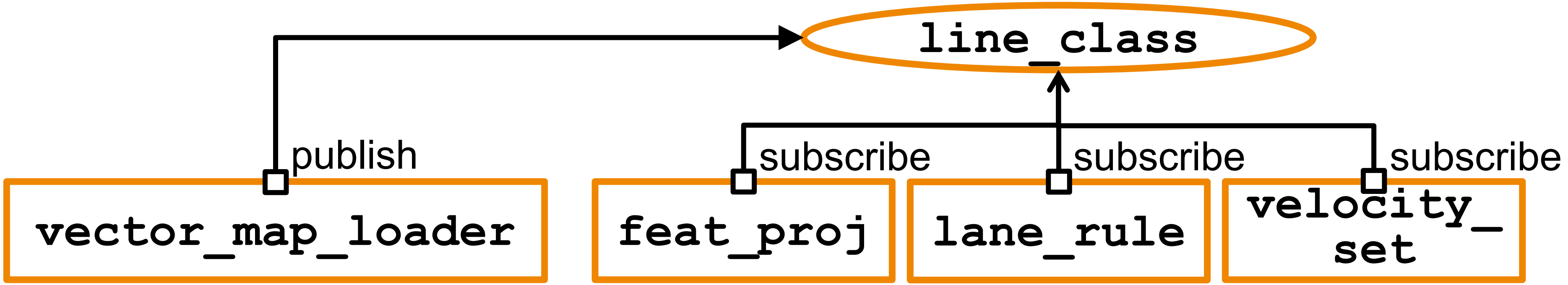
Legend

**Node** □Port **Topic** Role →

Bug-introducing commit (inconsistent topic-renaming):

```
- ros::Publisher pub = n.advertise<[…]>("/line_class",[…]);
+ ros::Publisher pub = n.advertise<[…]>("/line",[…]);
```

**Intended Architecture:**



line_class

publish

subscribe

subscribe

subscribe

vector_map_loader

feat_proj

lane_rule

velocity_set

# Here is a Real Architecture Misconfiguration Bug from Autoware.AI (Existed for 2 Months)

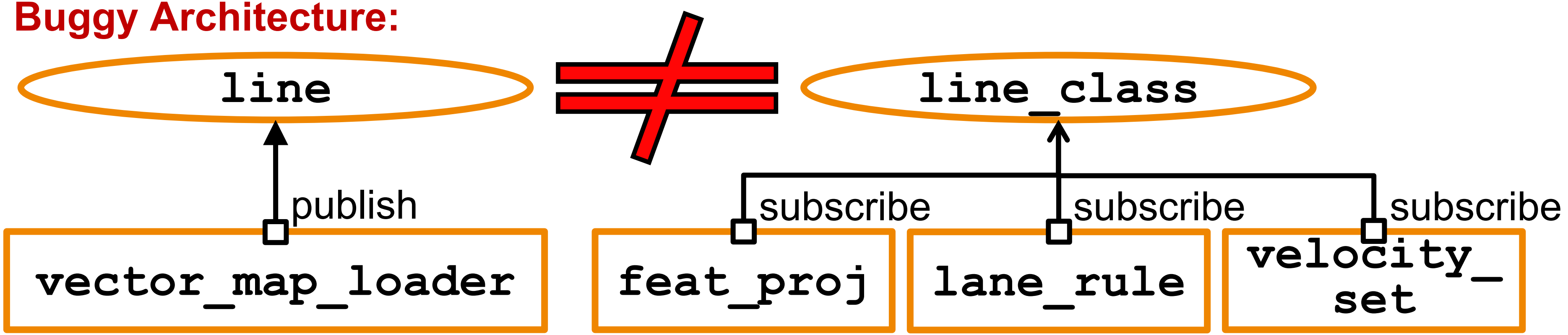
Legend

Node □Port Topic Role

Bug-introducing commit (inconsistent topic-renaming):

```
- ros::Publisher pub = n.advertise<[…]>("/line_class",[…]);
+ ros::Publisher pub = n.advertise<[…]>("/line",[…]);
```

**Buggy Architecture:**



line

≠

line_class

publish

subscribe

subscribe

subscribe

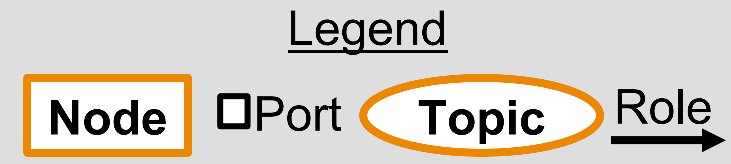
vector_map_loader

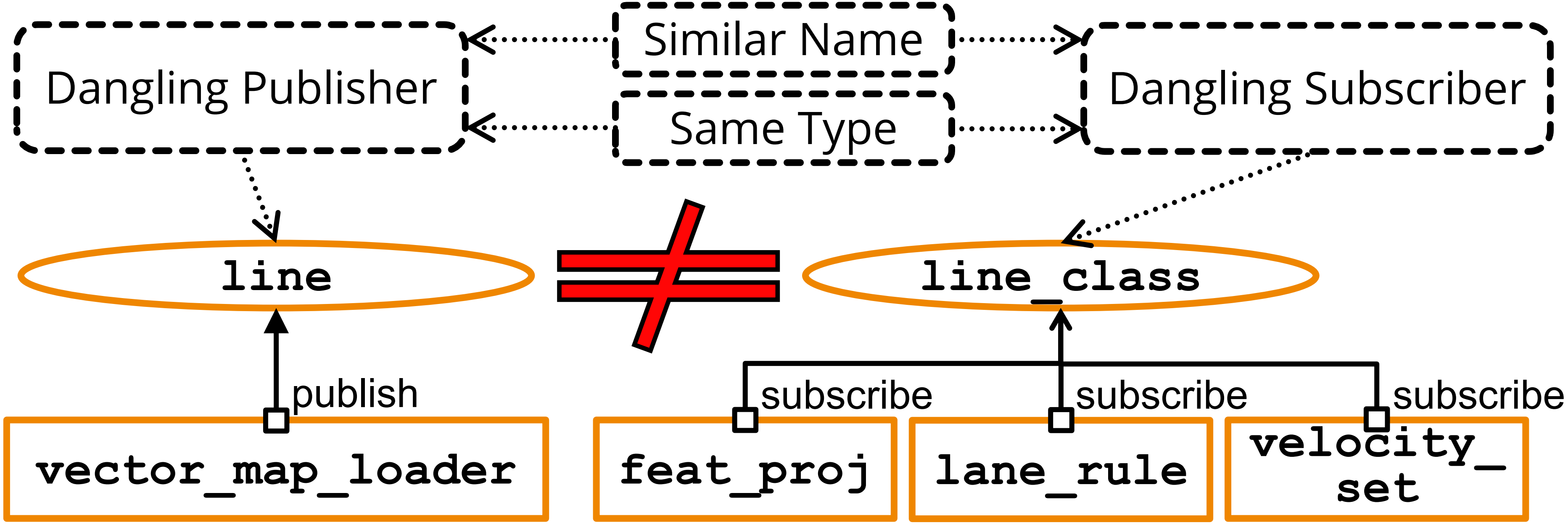
feat_proj

lane_rule

velocity_set

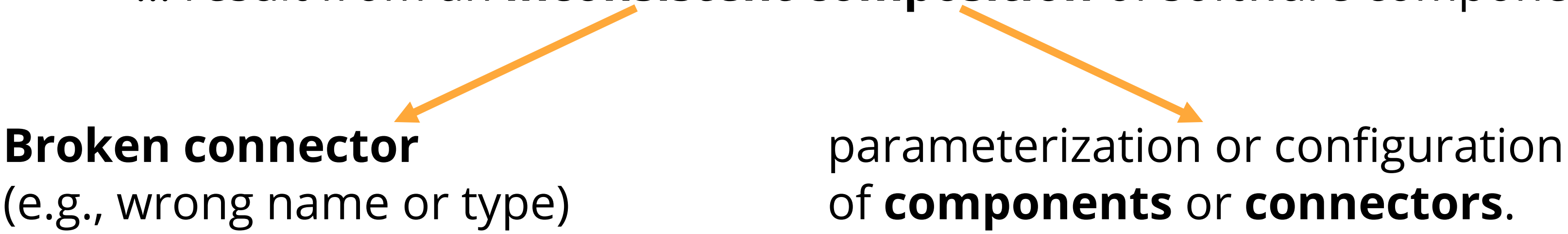
# Architecture Misconfiguration Bugs can be Detected in Run-Time Architecture Models





# Architecture Misconfiguration Bugs …

… result from an **inconsistent composition** of software components.



**Broken connector**
(e.g., wrong name or type)

parameterization or configuration
of **components** or **connectors**.

**Incorrect component parameterization**
(e.g., wrong name or type)

We manually collected a public **data set** of 29 architecture misconfiguration bugs in ROS on GitHub (see paper & artifact)

# Problem Definition

How can an **automatic analysis** technique find **architecture misconfiguration bugs** in ROS?

# Why is static architecture recovery hard?

Static recovery of a run-time architectures is **undecidable** in general.

Architecture-defining code is **scattered across the entire system**.

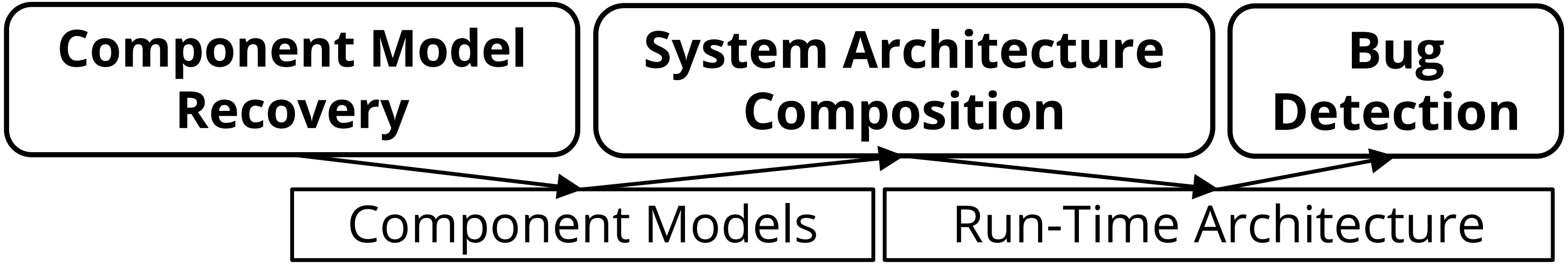
=> Exploiting 3 key observations about the ROS framework and ecosystem

# Problem Definition

How can an **automatic analysis** technique find **architecture configuration bugs** in ROS?

# Solution: `ROSDiscover`

Architectural recovery using static analysis + rule checking



**Component Model Recovery**

**System Architecture Composition**

**Bug Detection**

Component Models

Run-Time Architecture

# Static Architectural Recovery Approach

## Module View

vector_map_loader.cpp:

```
ros::Publisher line_pub = n.advertise<[…]>
("vector_map_info/line", […]);
```

velocity_set.cpp:

```
ros::Subscriber sub_line = nh.subscribe
("vector_map_info/line_class", […]);
```

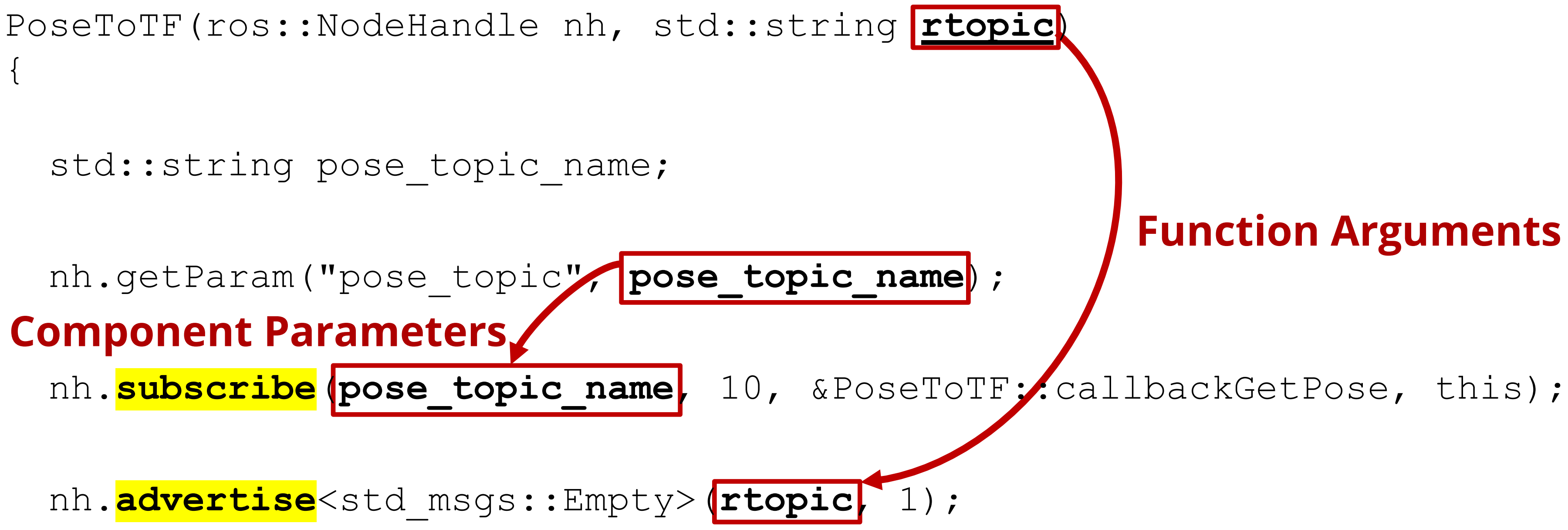
## Component Run-Time Model

```
publishes-to
"/vector_map_info/line"
LineArray

subscribes-to
"/vector_map_info/line_class"
LineArray
```

- Observation: ROS systems often have **quasi-static** architectures defined by a **small set of API calls** [1]
- Approach: Flow-sensitive static analysis of architecture-defining API calls

[1] A. Santos, A. Cunha, N. Macedo, R. Arrais, and F. N. dos Santos, "Mining the usage patterns of ROS primitives," in *International Conference on Intelligent Robots and Systems (IROS '17)*, IEEE, 2017, pp. 3855–3860

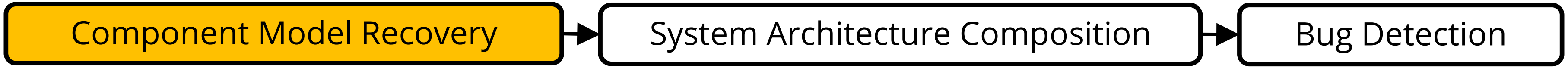
# Flow-sensitive Static Analysis is Needed to Resolve API Call Arguments



```
PoseToTF(ros::NodeHandle nh, std::string rtopic)
{

  std::string pose_topic_name;

  nh.getParam("pose_topic", pose_topic_name);

  nh.subscribe(pose_topic_name, 10, &PoseToTF::callbackGetPose, this);

  nh.advertise<std_msgs::Empty>(rtopic, 1);
```

Function Arguments

Component Parameters



Component Model Recovery

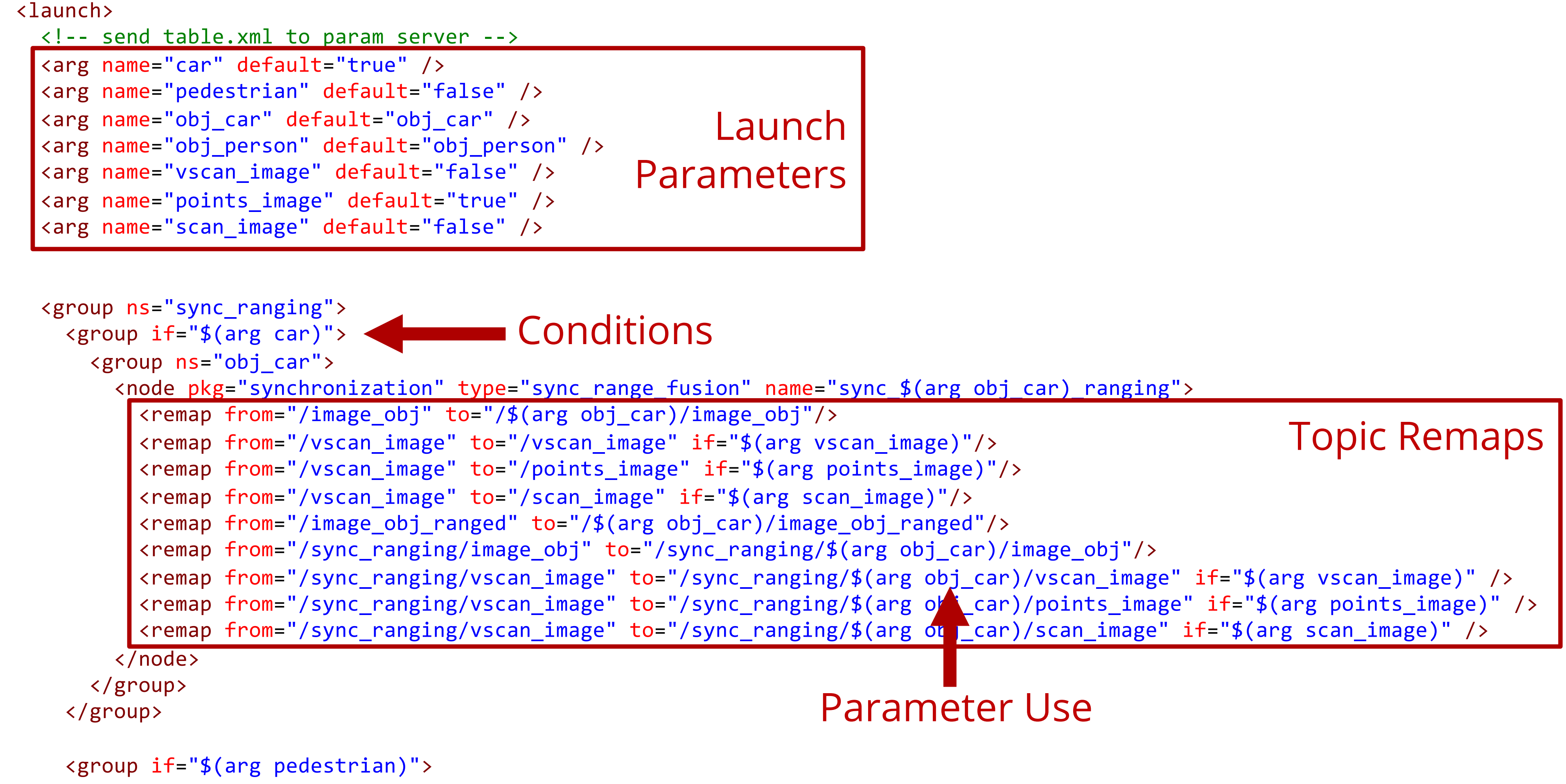
System Architecture Composition

Bug Detection

# RQ1: How accurately does ROSDiscover statically recover architecture-defining API calls?

- Metric: percentage of API calls for which architectural recovery can **resolve all arguments** (i.e., for which static analysis is complete)

| System | API Calls | Nodes | Fully recovered API Calls |
|---|---|---|---|
| AutoRally | 75 | 25 | **86.67%** |
| Autoware | 882 | 209 | **85.49%** |
| Fetch | 103 | 93 | **98.06%** |
| Husky | 223 | 105 | **97.31%** |
| TurtleBot | 130 | 104 | **85.38%** |
| All | 1306 | 507 | **87.37%** |



```
<launch>
  <!-- send table.xml to param server -->
  <arg name="car" default="true" />
  <arg name="pedestrian" default="false" />
  <arg name="obj_car" default="obj_car" />
  <arg name="obj_person" default="obj_person" />
  <arg name="vscan_image" default="false" />
  <arg name="points_image" default="true" />
  <arg name="scan_image" default="false" />


  <group ns="sync_ranging">
    <group if="$(arg car)">
      <group ns="obj_car">
        <node pkg="synchronization" type="sync_range_fusion" name="sync_$(arg obj_car)_ranging">
          <remap from="/image_obj" to="/$(arg obj_car)/image_obj"/>
          <remap from="/vscan_image" to="/vscan_image" if="$(arg vscan_image)"/>
          <remap from="/vscan_image" to="/points_image" if="$(arg points_image)"/>
          <remap from="/vscan_image" to="/scan_image" if="$(arg scan_image)"/>
          <remap from="/image_obj_ranged" to="/$(arg obj_car)/image_obj_ranged"/>
          <remap from="/sync_ranging/image_obj" to="/sync_ranging/$(arg obj_car)/image_obj"/>
          <remap from="/sync_ranging/vscan_image" to="/sync_ranging/$(arg obj_car)/vscan_image" if="$(arg vscan_image)" />
          <remap from="/sync_ranging/vscan_image" to="/sync_ranging/$(arg obj_car)/points_image" if="$(arg points_image)" />
          <remap from="/sync_ranging/vscan_image" to="/sync_ranging/$(arg obj_car)/scan_image" if="$(arg scan_image)" />
        </node>
      </group>
    </group>

    <group if="$(arg pedestrian)">
```

Launch Parameters

Conditions

Topic Remaps

Parameter Use

# Static Architecture Recovery Approach

- Compose component models using architecture configuration files (“launch files”)
- Challenge: For realistic systems, static recovery will **never be complete**
- Observation: ROS systems rely on a small de facto **core library** of components [2]
    - These components use very dynamic structures
- Solution: Let developers provide **hand-written** models

[2] S. Kolak, A. Afzal, C. Le Goues, M. Hilton, and C. S. Timperley, “It Takes a Village to Build a Robot: An Empirical Study of The ROS Ecosystem,” in *International Conference on Software Maintenance and Evolution (ICSME ’20),* IEEE, 2020, pp. 430–440

# RQ2: How accurately does ROSDiscover statically recover run-time architectures of real ROS systems?

- Baseline: Executing a configuration of the system and **dynamically observe** run-time architecture
- Compare with statically recovered run-time architecture

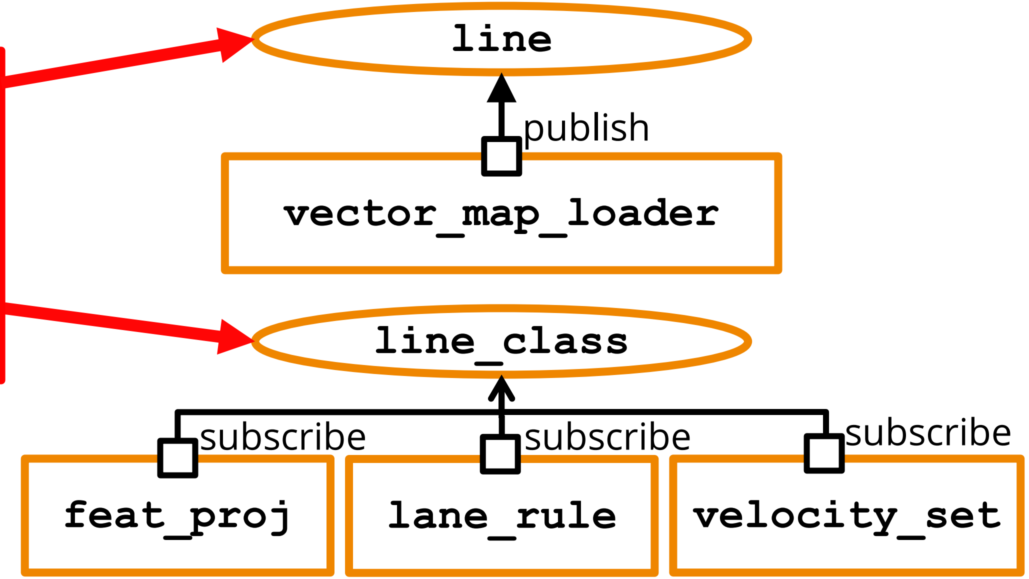
| System | Precision | Recall |
|---|---|---|
| AutoRally | 100.00% | **100.00%** |
| Husky | 94.74% | **84.21%** |
| TurtleBot | 83.33% | **91.67%** |
| All | 93.18% | **90.91%** |

# Bug Detection Approach

- Checking architectural well-formedness rules

**Rule Violation**

There should not be a dangling subscriber/publisher if there is a publisher/subscriber that has a compatible type and a similar topic name.



Component Model Recovery
System Architecture Composition
Bug Detection

# RQ3: How effectively does ROSDiscover find configuration bugs in real ROS systems?

- Goal: Minimize false positives while still finding enough bugs
- Method: manually collected data set from documented bugs (see artifact), rule checking on statically recovered architectures

| System | Bugs | Bugs Found | False Positives |
|---|---|---|---|
| Autoware | 8 | 2 (25%) | |
| AutoRally | 5 | 3 (60%) | 8 |
| Husky | 5 | 3 (60%) | 5 |
| TurtleBot | 1 | 0 (0%) | 2 |
| All | **19** | **8 (42%)** | |

# We have an Artifact!

Reusable Badge

Open Science Badge

- Available at Zenodo: https://doi.org/10.5281/zenodo.5834633
- Reusable **ROSDiscover tool** (code and docker image)
  - Also on GitHub: https://github.com/rosqual/rosdiscover
- Reusable **Data set** of 29 architecture misconfiguration bugs across 5 real-world ROS systems reproduceable in provided Docker images
- All **results** and **analysis scripts** of the evaluation for replication

# Summary

**Problem:** How can an automatic analysis find architecture misconfiguration bugs in real-world ROS systems?

**Solution:** Rule checking on statically recovered architectures enabled by key observations:

- Well-defined component framework API
- Quasi-static architectures
- Highly reused core library

**Results:** 90% recall of system recovery

Detecting 8 of 19 of real-world bugs

**Artifact:** Tool, bug data set, and results

**Conclusion:** Static recovery of run-time architectures in ROS is feasible and can be used for finding architecture misconfiguration bugs in real-world systems