Chain-Aware ROS Evaluation Tool (CARET)

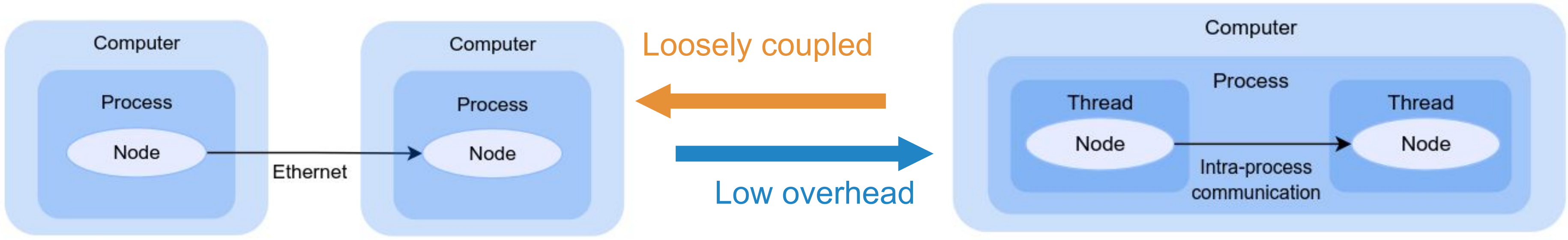
Atsushi Hasegawa (Research Institute of Systems Planning, Inc.)
Keita Miura (EMB IV, Inc.)

This project is being conducted in cooperation with the University of Tokyo and TIER IV.

Background | ROS 2 Features

- ROS 2 provides:
  - Pub/Sub communication
  - Callback scheduler

  ⇒ ROS 2 realizes loosely-coupled, highly interoperable systems
- Appropriate designing allow for a variety of systems.



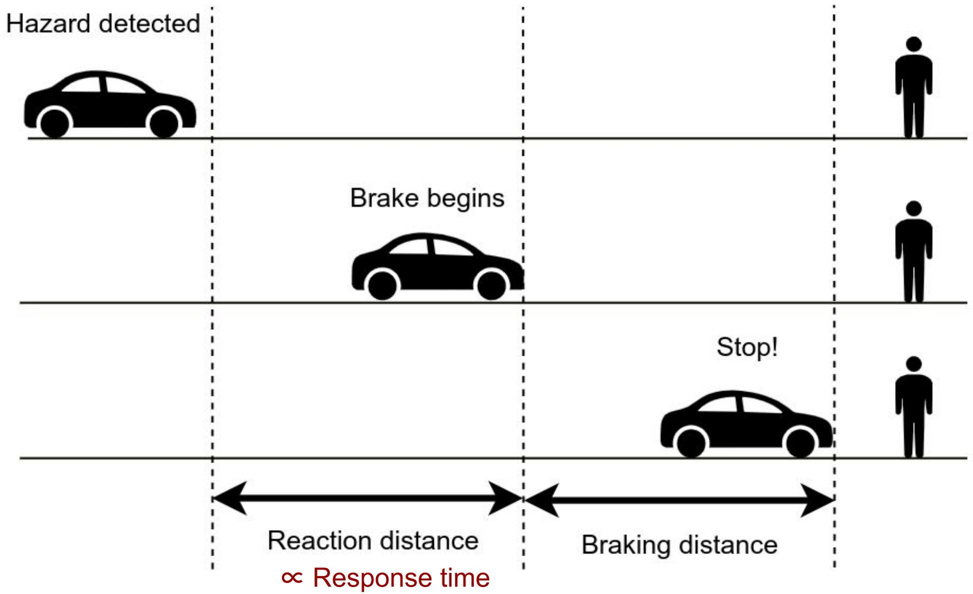
(a) Many hosts

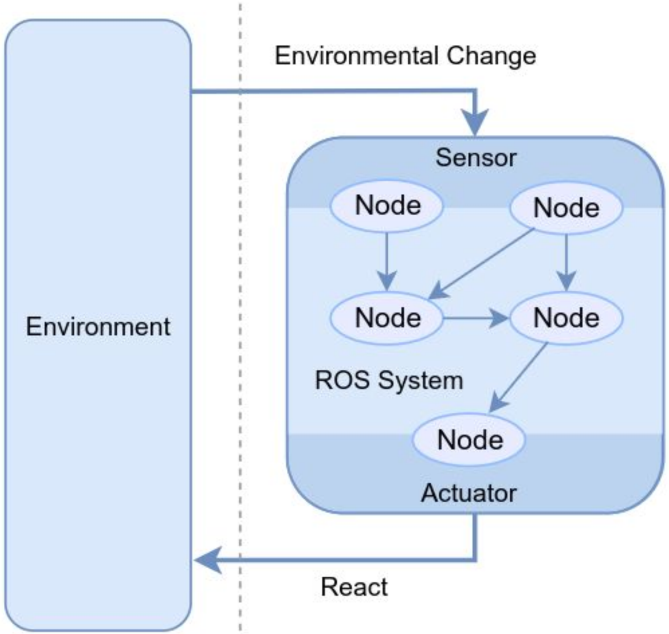
(b) Single process

Example of implementation for parallel processing of nodes

Background | Response Time

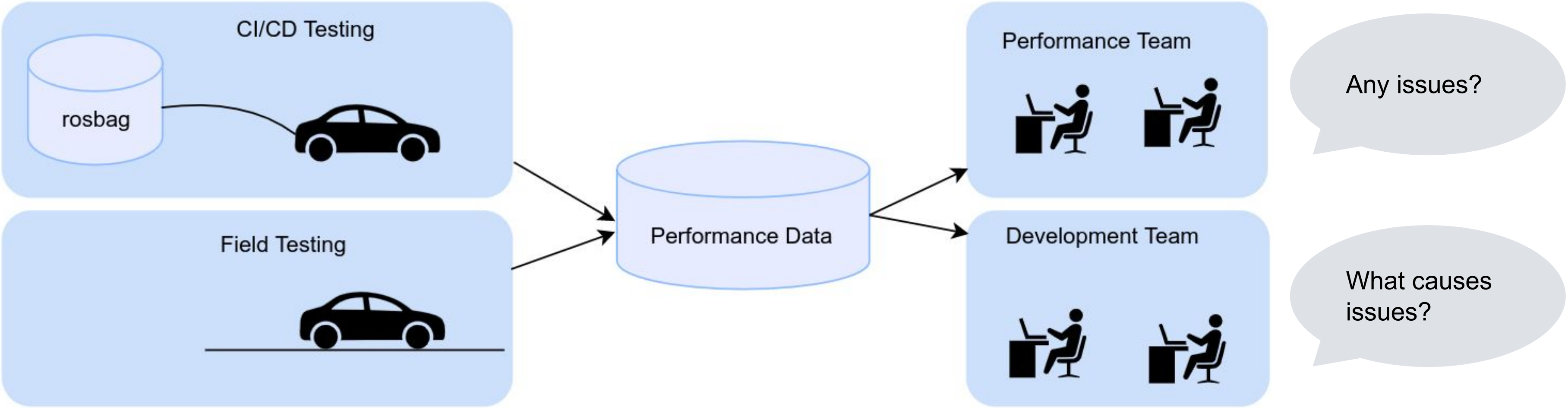
- Response time is important for safety-critical systems, but difficult to measure.
- Cooperation of many nodes is important for system to behave as desired.





Use cases for Evaluation

- Division of labor for performance evaluation and analysis.
  - Measure performance with simulation or field testing.
  - Analyze from multiple perspectives with various teams.



Introduction to CARET

- Our objective:
  - Evaluating response time
  - Involving various teams in tackling performance issues

even for a large-scale ROS 2 system like Autoware.

- To achieve these objectives,
  we developed Chain-Aware ROS Evaluation Tool (CARET).

For example, CARET answers:

- “Autoware response time was X ms.”
- “QoS history of /cmd_vel topic was X.”

https://github.com/tier4/CARET

Features

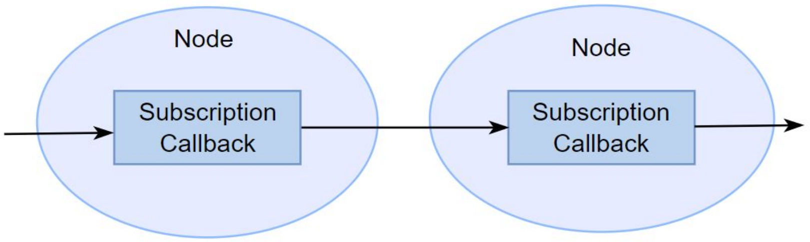
- Lightweight
  - Add extra LTTng trace points via hook
  - Record only necessary information

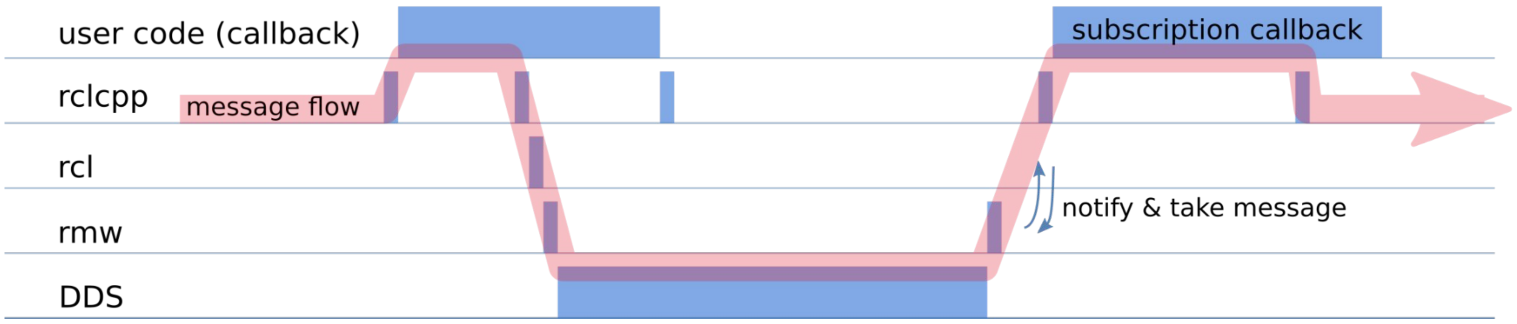
- Flexibility
  - Provide Python-API for developers to analyze issues

- Intuitively evaluable visualization
  - Support several types of intuitive figures

Measurement Targets

- CARET measures:
  - Callback latency
  - Communication latency
  - Node latency
  - Path latency

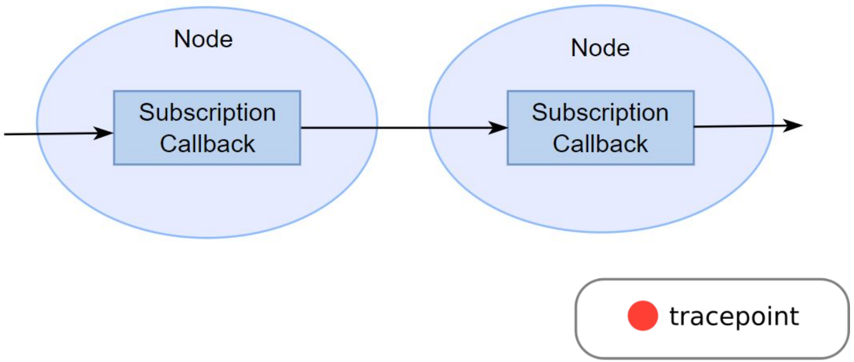


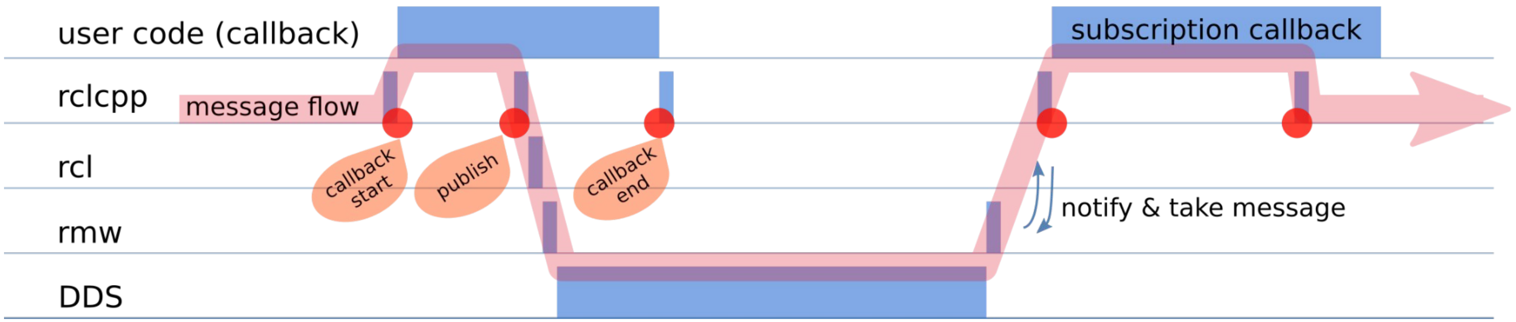


Major trace points and latency

Measurement Targets

- CARET measures:
  - Callback latency
  - Communication latency
  - Node latency
  - Path latency

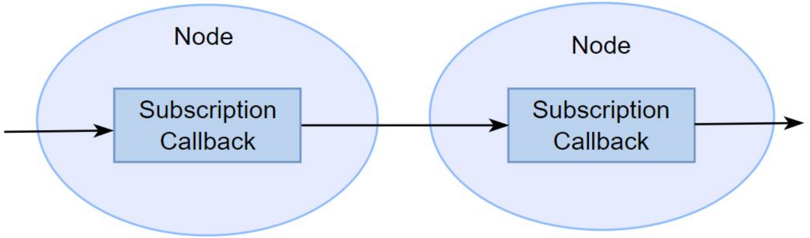


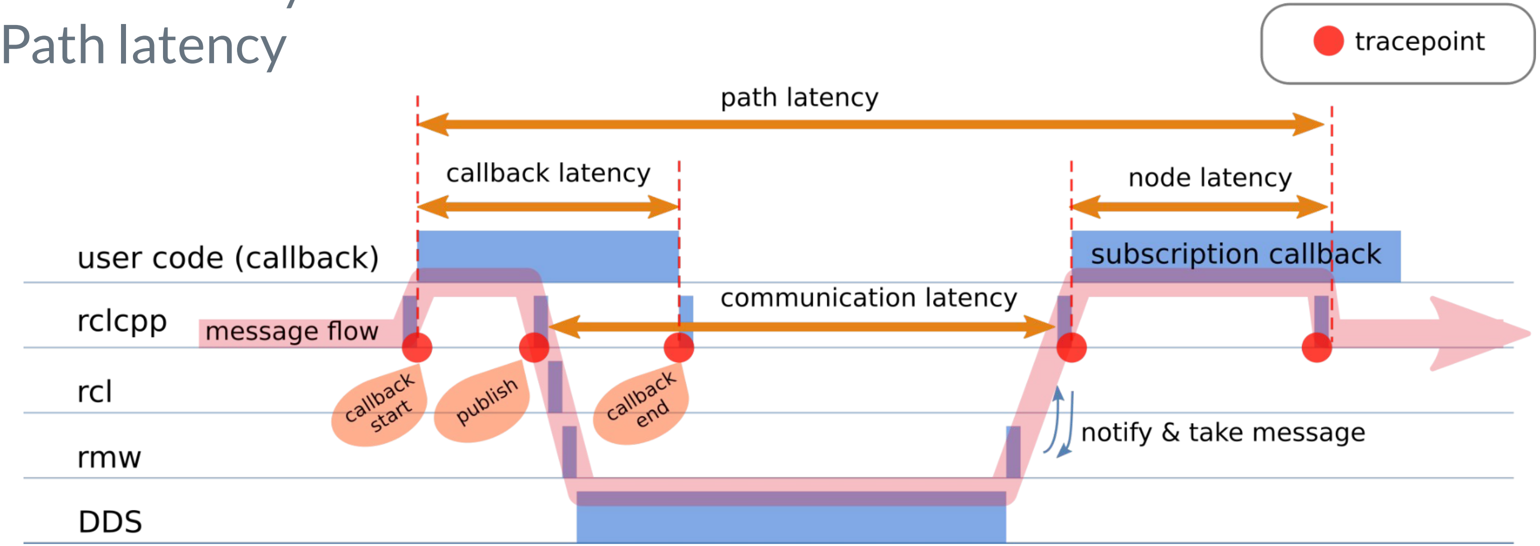


Major trace points and latency

Measurement Targets

- CARET measures:
  - Callback latency
  - Communication latency
  - Node latency
  - Path latency





Major trace points and latency

Visualization Concept

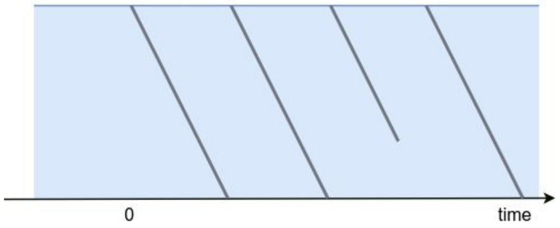
CARET supports coarse to fine granularity visualization APIs.

Any issues?

response time
- min: 100 ms
- ave: 150 ms
- max: 300 ms

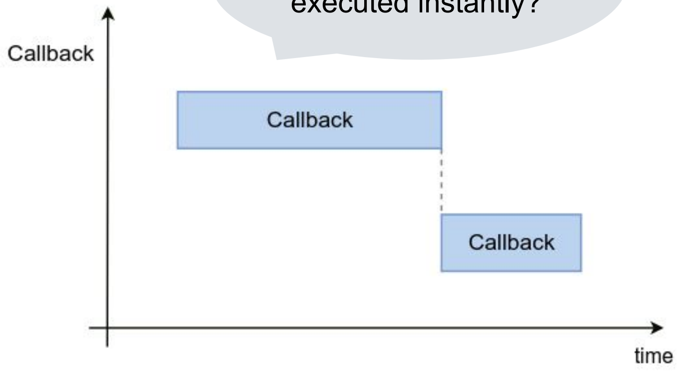
Response time

Is QoS history depth appropriate?



Message flow

Were callbacks executed instantly?



Callback scheduling

Coarse

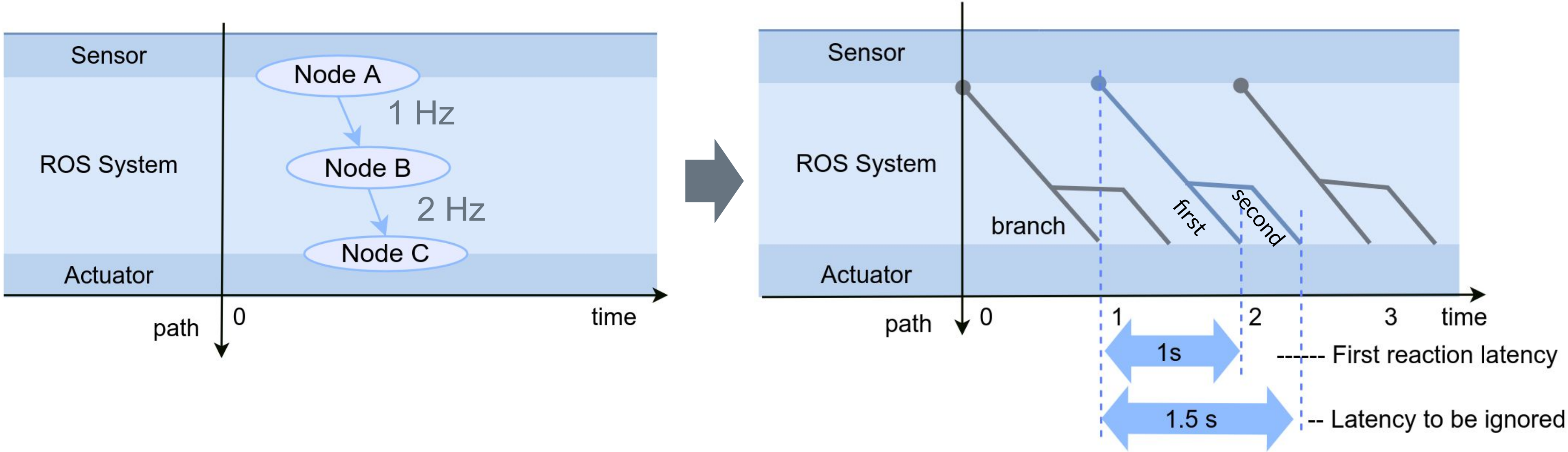
Granularity

Fine

Key Idea | Message flow

"Which input message did the output message use?"

1. Developers define node path to evaluate.
2. CARET draws message dependencies between I/O nodes.

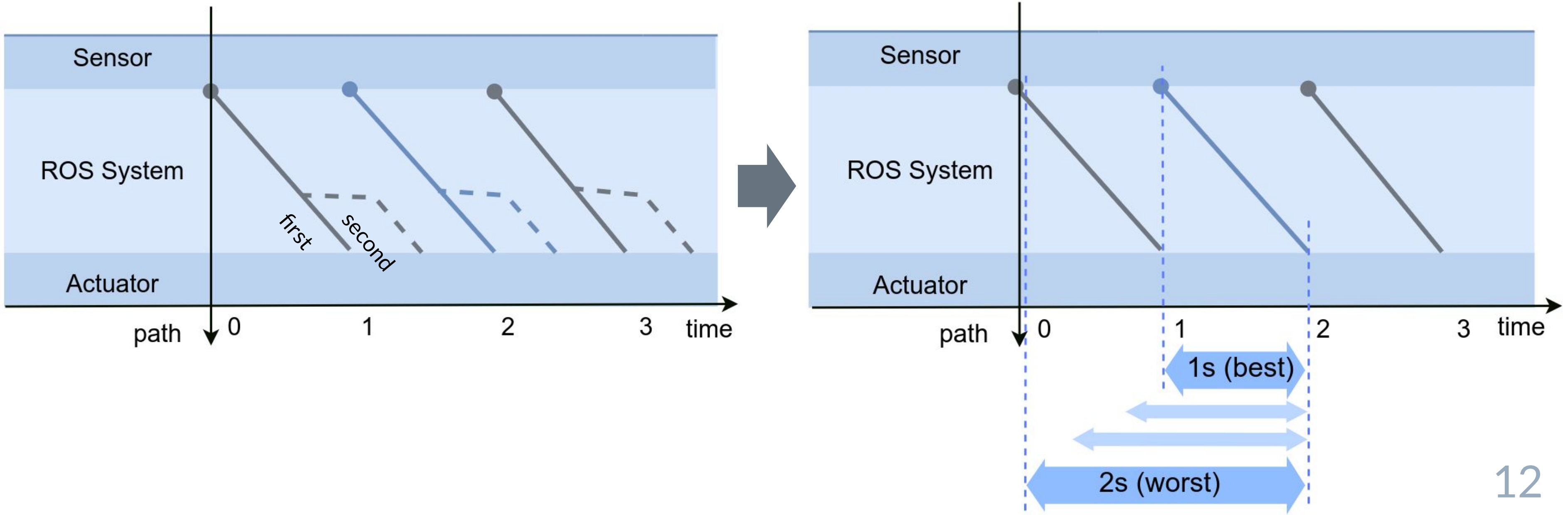


However, latency in message flow is not suitable for response time evaluation.

Key Idea | Response time

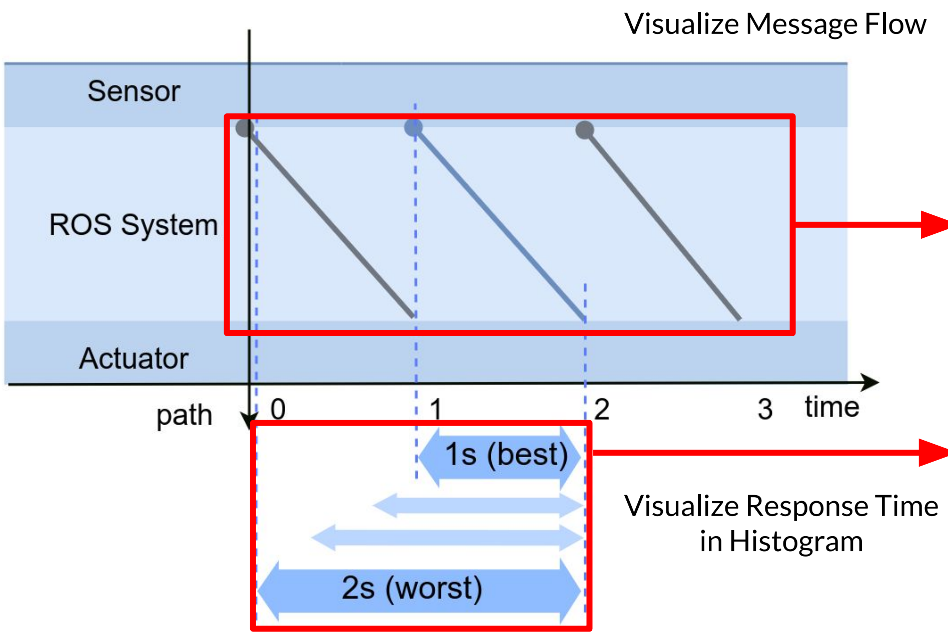
"When does the system respond to events?"

1. CARET uses each first flow that reflects input message.
2. CARET considers worst case as well.

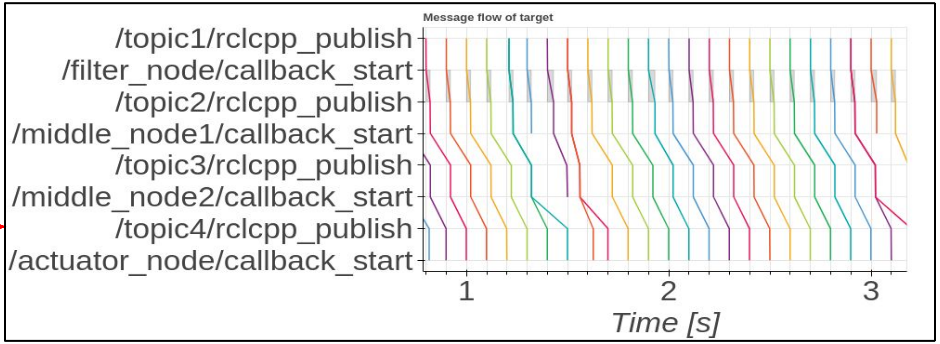


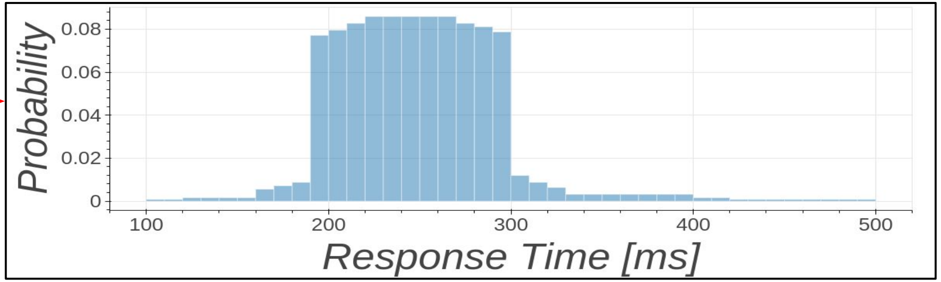
Diagrams and Figures

Diagram



Figures by CARET

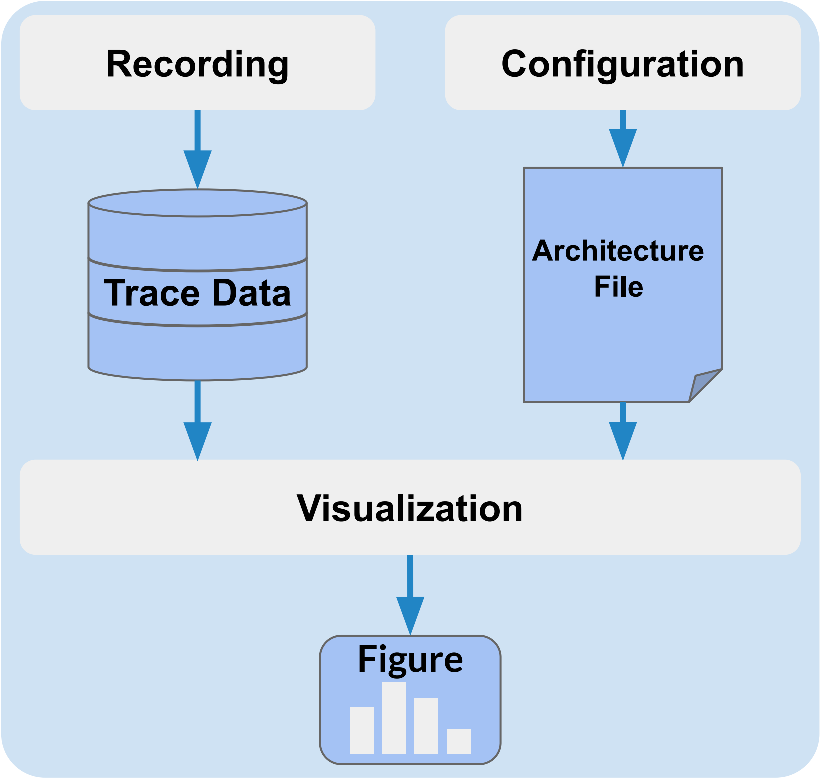




Evaluation Steps with CARET

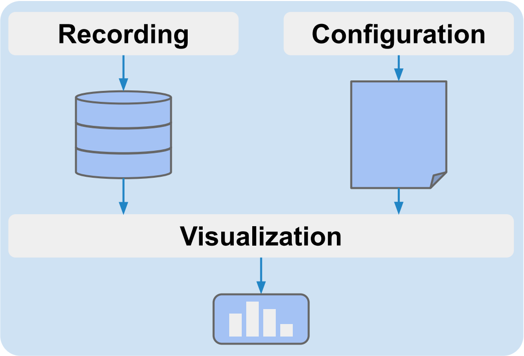
Evaluation steps:

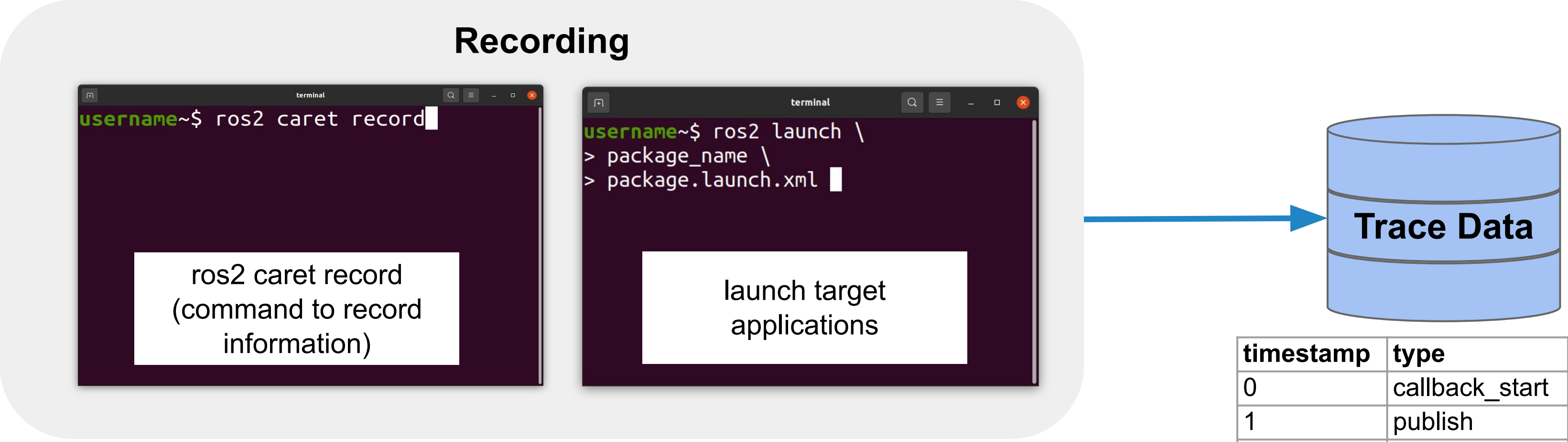
1. Recording
2. Configuration
3. Visualization



Evaluation Steps | Recording

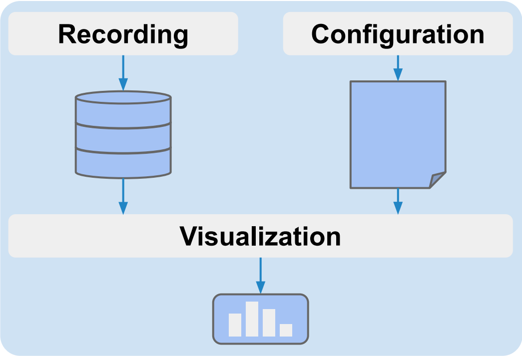
- Generate trace data
  - Trace data is historical data and consists of information at trace points.
    (timestamp, trace point’s type, message address, etc)





timestamp	type	args
0	callback_start	…
1	publish	…
2	callback_end	…
…	…	…

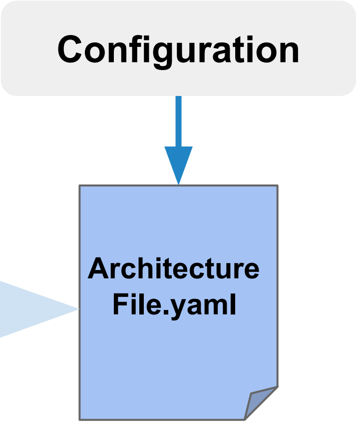
Evaluation Steps | Configuration



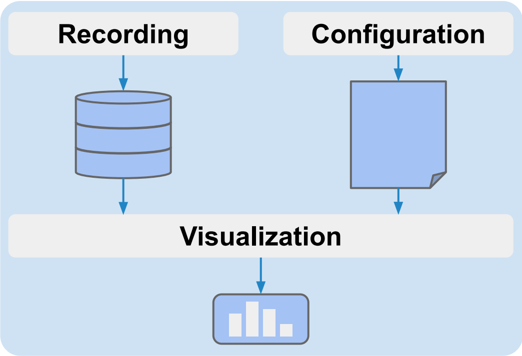
- Create an “Architecture file” which defines followings
  - Target paths
  - Target application structure

<u>Architecture File</u>

- **Target path**
  - [node_name, topic_name, node_name, …]
- **Executor Information**
  - Type
- **Node Information**
  - Callbacks Information
  - Node Latency Definition



Evaluation Steps | Visualization



- Visualize the measurement results
  - CARET provides visualization APIs.
  - Developers can check the results with Jupyter-notebook.

Trace Data

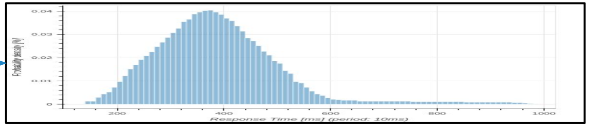
Architecture File

Visualization

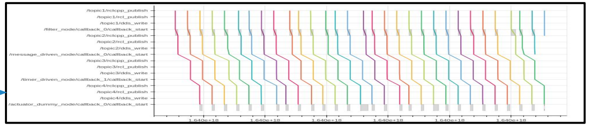
[1]:
```
#!/usr/bin/env python3
from caret_analyze.plot import Plot
… # Processing trace data and architecture
Plot.create_path_histogram(target_path)
```

[2]:
```
Plot.create_message_flow(target_path)
```

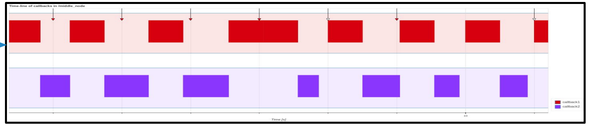
[3]:
```
Plot.create_callback_sched(target_path)
```



Histogram



Message Flow



Callback Schedule

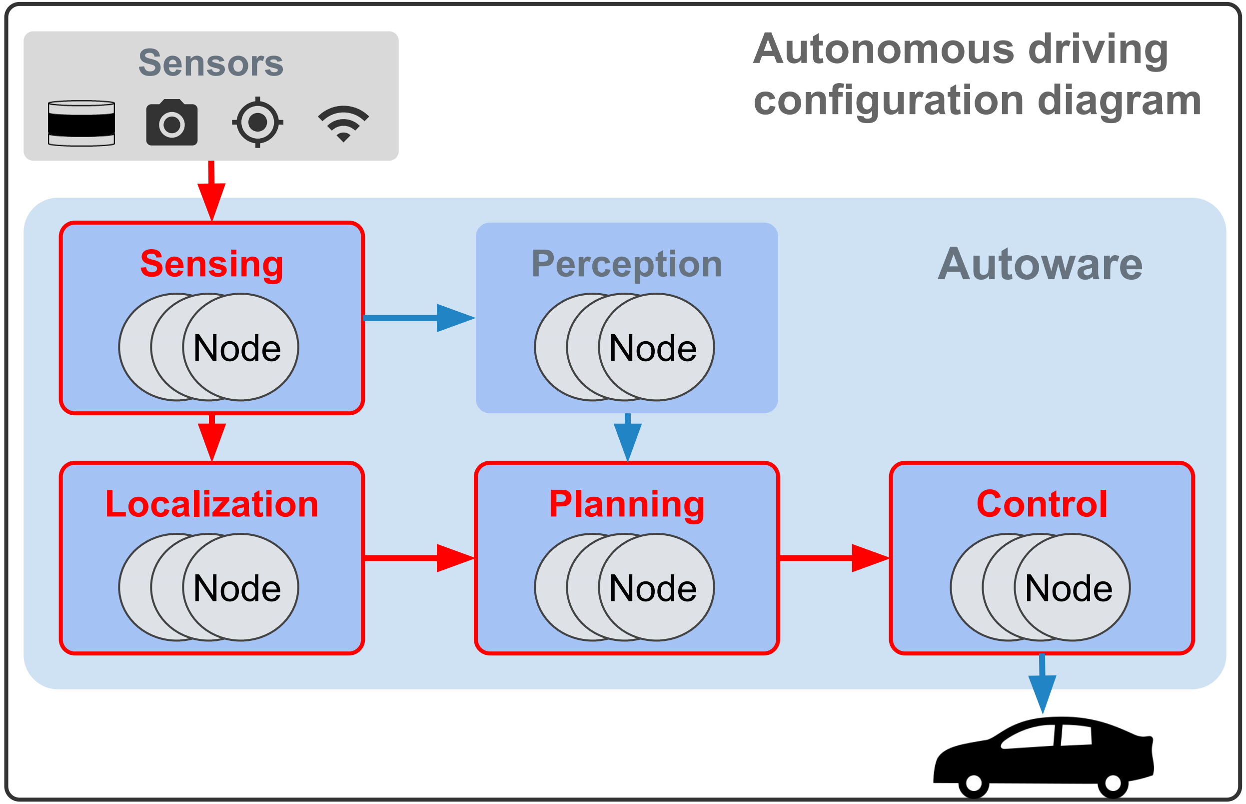
Measurement of an Actual System

Autoware

- An open source **ROS based** autonomous driving system

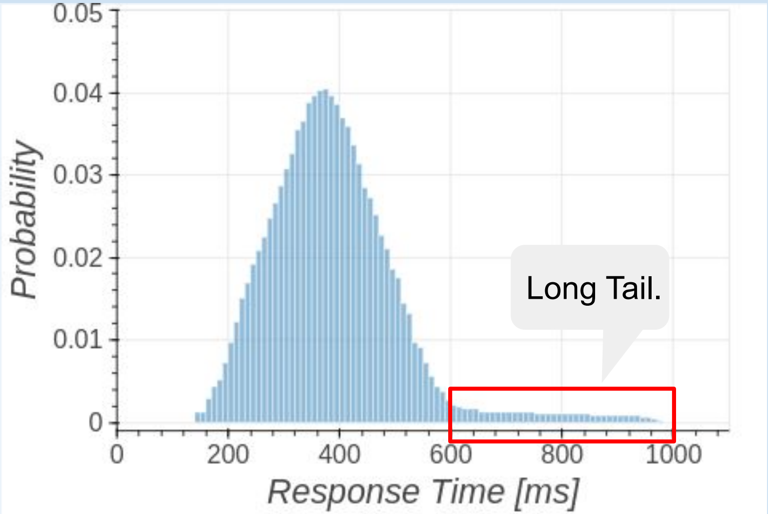
Target path :

- From Sensing to Control
- Including major modules / nodes for autonomous driving



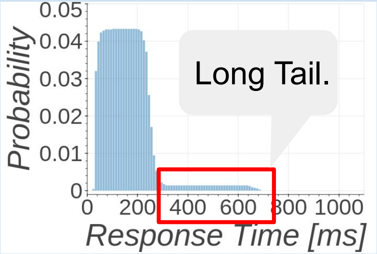
Measurement Results of Autoware

Target path Histogram

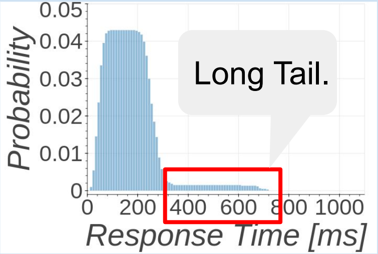


Sensing → Localization → Planning → Control

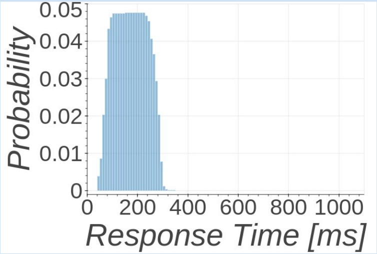
Each module Histogram



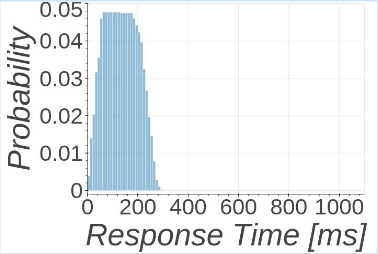
Sensing



Localization



Planning



Control

Node Histogram

...

Coarse Granularity Fine

Future Work

Provide feedback on the perspectives of the autoware evaluation.

1. Expand measurement coverage
   - Path containing /tf topic
   - DDS layer
   - System call
   - Multiple host
2. Mitigate limits and constraints
   - Support complex nodes
3. Propose to integrate CARET trace points into each official package

Discussion and comments are welcome !

CARET

Chain-Aware ROS Evaluation Tool

https://github.com/tier4/CARET

ISP EmbIV Intelligent Vehicle

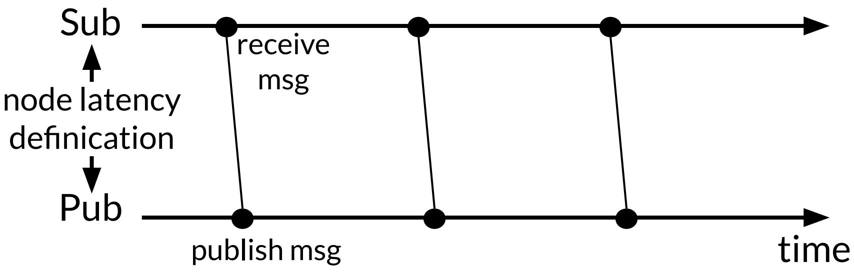
This presentation is based on results obtained from a project, JPNP16007, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

T. Kuboichi, A. Hasegawa, B. Peng, K. Miura, K. Funaoka, S. Kato and T. Azumi, “**CARET: Chain-Aware ROS 2 Evaluation Tool,**” in Proc. of IEEE/ECU, Nov 2022 (in press)

Appendix

Examples of Complex Nodes

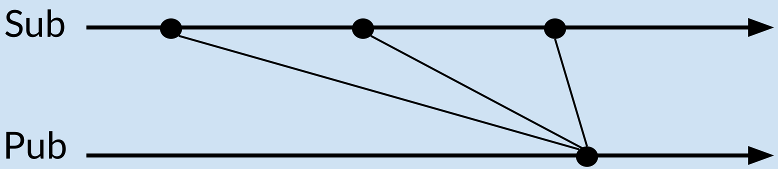
Possible to measure



The node receiving data and publishing the data.

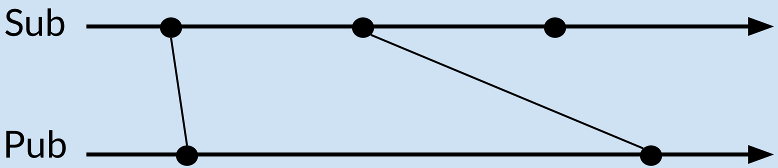
```
sub_cb = [](&msg){
  msg_ = func(msg);
 publish(msg_)
}
```

Impossible to measure



The node receiving some data and publishing one data.

```
sub_cb = [](&msg){
  queue.push(msg)
  msg_ = avg(queue);
 publish(msg_)
}
```



The node using the data at a specific time.

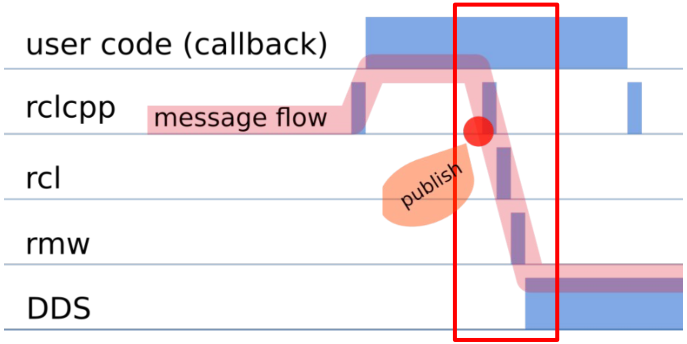
```
sub_cb = [](&msg){
 msg_ = lookup(msg.stamp);
 publish(msg_)
}
```

Overhead

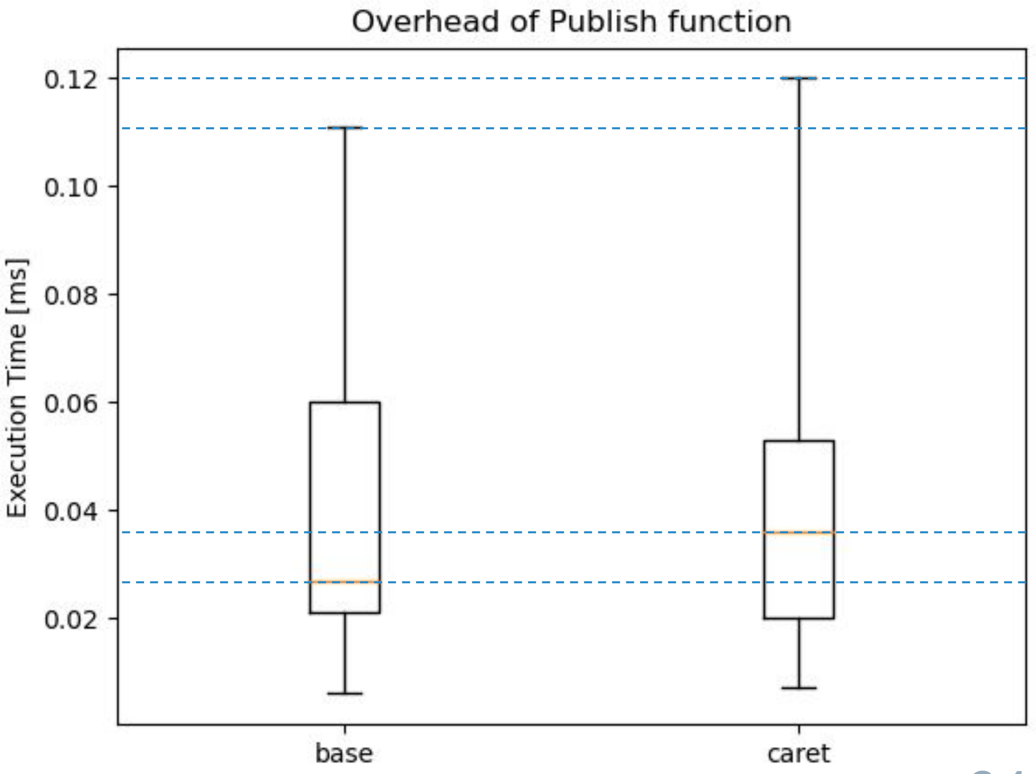
Measurement Method

```
msg_->data = "Hello World";
start_time = now();
pub_->publish(msg_);
end_time = now();

exe_time = end_time - start_time;
```



additional 4~7 tracepoints



Output Figures with CARET

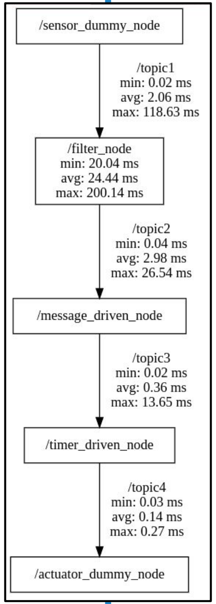
Coarse ← **Visualization Granularity** → Fine

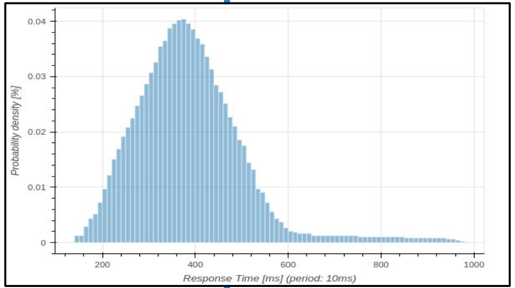
System Level

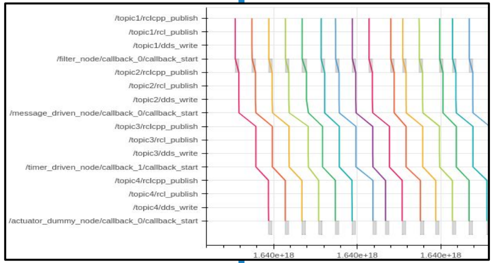
Target Granularity

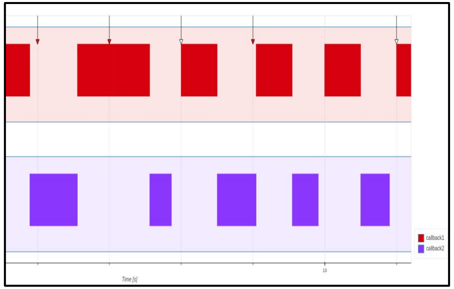
Callback Level

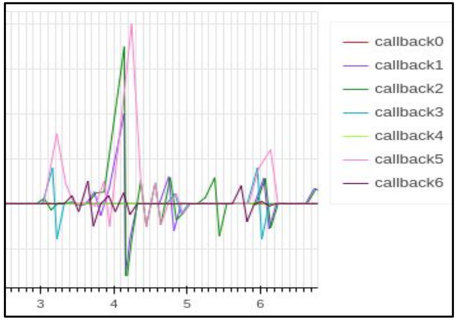
Scalable





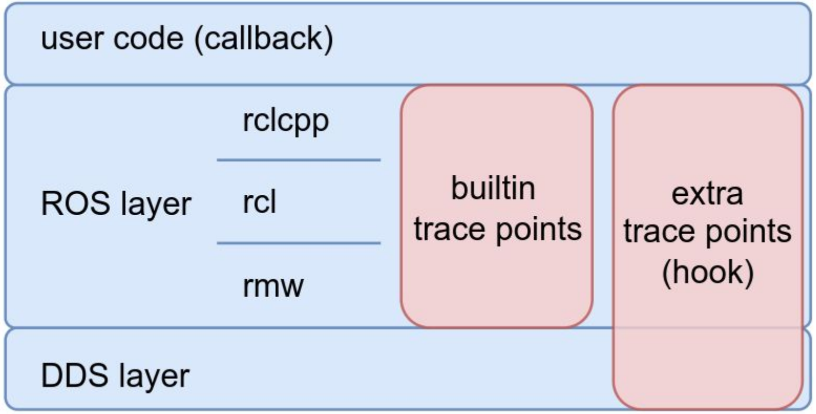




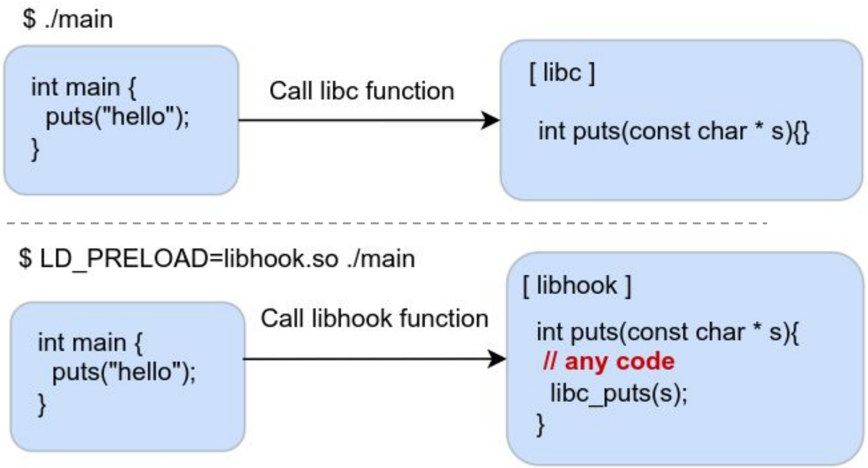


Trace Points

- Add trace points to record information.
    - Time (e.g. callback start)
    - Implementation (e.g. node name)
    - Configuration (e.g. QoS)
    - Message identifier (address, message stamp)
      ※ Skip recording by hook if nodes or topics are unnecessary.



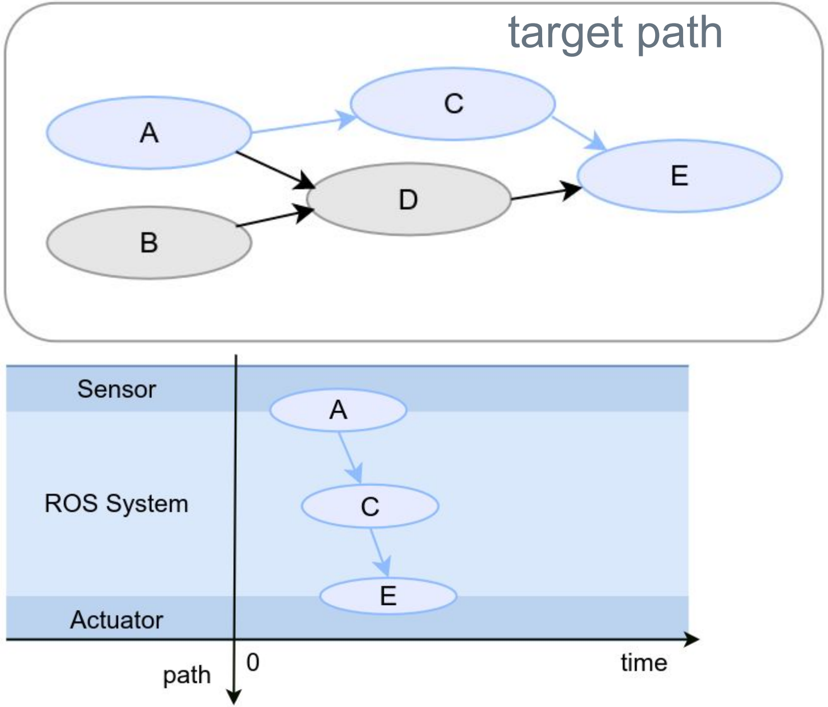
Tracepoints added layers



Hooking with LD_PRELOAD

Definition of target paths

- Define a path to evaluate from a node graph.



- Search to select a path
  - search(A, E) # CARET API
    Enumerate the paths to be written in one stroke from node A to node E.
    returns:
    - [A,C,E]
    - [A,D,E]
- Define manually
  - It is possible to specify the path manually.

Scheduling Visualization

- Callback scheduling affects node latencies.
- Scheduling visualization decompose node latencies into callback latencies and scheduling latencies.