TIER IV

ROS 2 Can Finally Evaluate Your DAG Scheduler: Autoware as a Real-Time Systems Research Platform

Takahiro Ishikawa-Aso

Department Manager, System Software Department

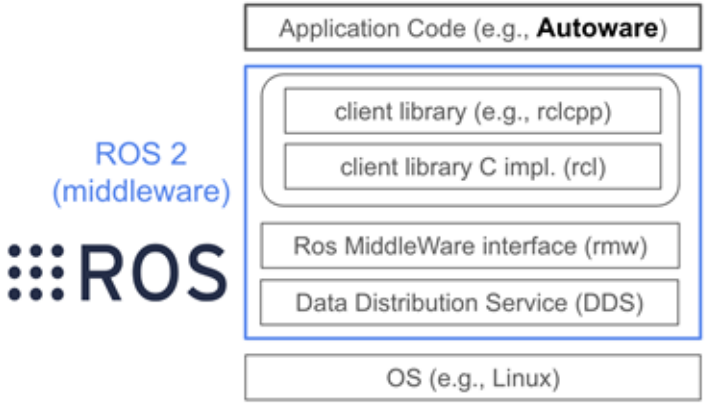
TIER IV, Inc. (Japan)

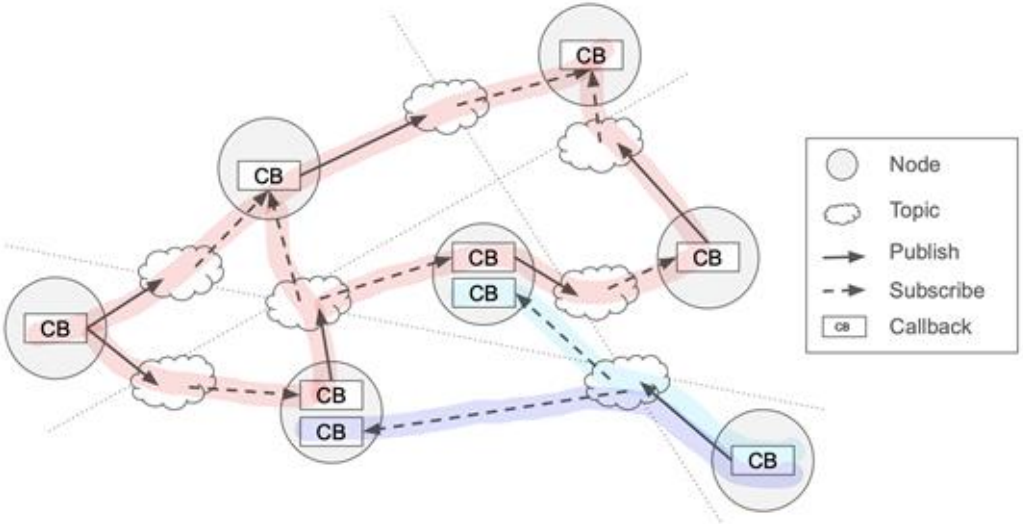
Talk Goal

- **Robot Operating System 2 (ROS 2)** has been an active target for real-time scheduling research over the last several years.
- **Autoware** — the largest open-source ROS 2 application, deployed on public roads in commercial autonomous driving — is officially adopting new middleware that closes long-standing structural gaps for real-time research.
  - **CallbackIsolatedExecutor** — new ROS 2 Executor
  - **Agnocast** — rclcpp (ROS 2 C++) compatible, new zero-copy middleware
- This talk reports what those gaps were, what closed them, and how to put the result into your own evaluation setup.

What is ROS 2

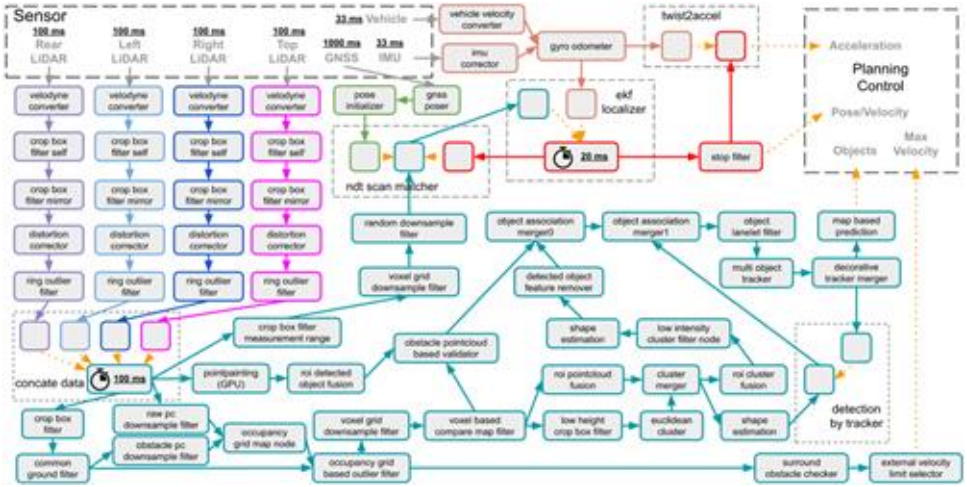
ROS 2 is the de facto open-source middleware for robotics — applications are graphs of callbacks (vertices) connected by topics (edges).





What is Autoware (and TIER IV)

Autoware: the largest open-source autonomous driving stack on ROS 2.
Governed by the Autoware Foundation — 100+ organizations across 20+ countries, co-founded by TIER IV in 2018. Already in Level 4 autonomous bus operation on Japanese public roads.



About this work

Industry

Head of System Software Department at TIER IV, Inc.

Research, develop and integrate system software tailored for Autoware

Academia

Industrial PhD candidate at the University of Tokyo

PhD thesis submission planned this year (Shinpei Kato's laboratory)

Maintainer

Lead Maintainer of System Software for Autoware

Decides system software strategy (which ones to introduce) for Autoware

Three aligned roles drive research end to end, by design —
Academic research → production-grade software development → ROS distribution platform → integration into Autoware → public-road autonomous driving all over the world

TIER IV

ROS 2 Can Finally Evaluate Your DAG Scheduler: Autoware as a Real-Time Systems Research Platform

01 / Research Background: Two Gaps, Three Papers

02 / Autoware Deployment:

Officially adopting CallbackIsolatedExecutor + Agnocast

03 / Hands-On (1): CallbackIsolatedExecutor

04 / Hands-On (2): Agnocast

05 / Closing

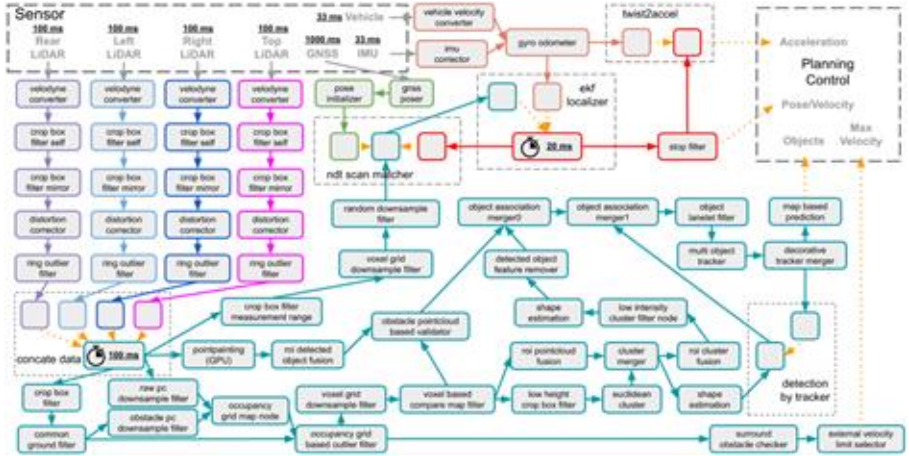
Research Background

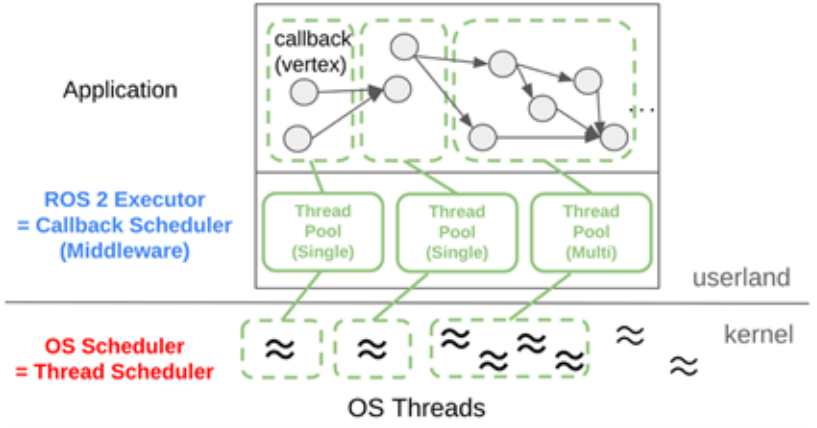
Two Gaps, Three Papers

01

ROS 2 as real-time workload

- ROS 2 apps form callback DAGs – callbacks as vertices, topic edges as dataflow
- Multi-source, multi-sink DAGs with independent **periods** and **deadlines**
  - Work-in-Progress: Multi-Deadline DAG Scheduling Model for Autonomous Driving Systems (RTSS '24 BP)
- Callbacks are subject to nested scheduling: callback scheduler and thread scheduler





How RT research has approached this workload

Two foundational idealizations underpin existing ROS 2 realtime analyses:

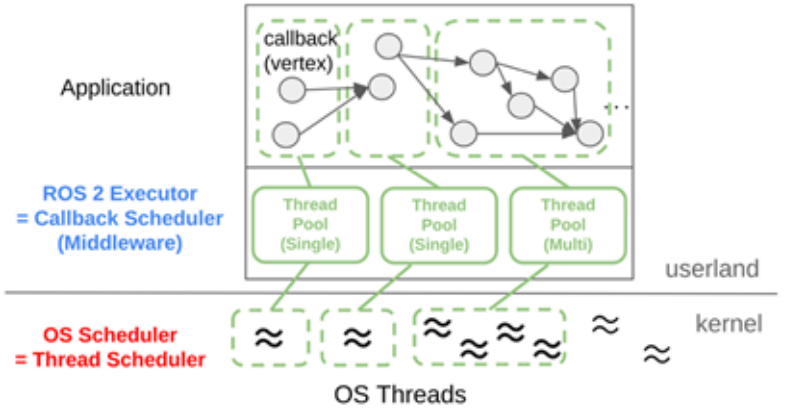
① Executor-aware scheduling algorithms

Research designs algorithms specific to the nested scheduling structure.

- Response-time analysis
- Priority assignment under round-robin (or Event) Executor

② Inter-node comm. cost idealized

DAG / Cause-Effect Chain models treat edge weight as 0 or a known constant, enabling analytic tractability.



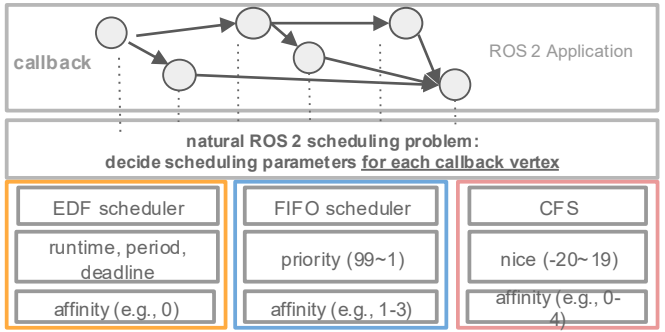
→ These two idealizations have made RT research outcomes hard to absorb into Autoware. 6+ years of papers across RTAS / RTSS / ECRTS / TCAD — foundational by Casini et al. (ECRTS '19), and many others.

Where the idealizations meet Autoware reality

Both idealizations diverge in Autoware specifically:

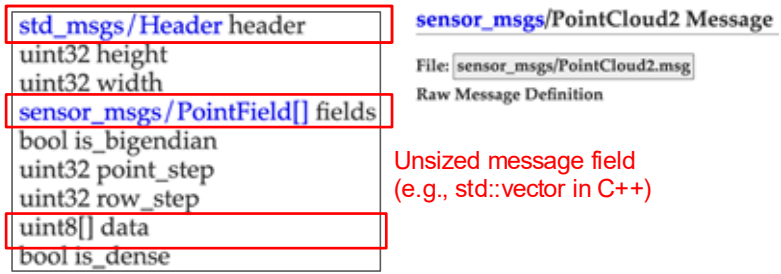
Gap1 **Executor-aware scheduling is too complex and artifical**

- Executor-aware algorithms are too complex for practitioners to adopt in Autoware.
- The callback-level DAG is the natural model of the workload, so scheduling attributes (e.g., priority, affinity) should be set directly on its vertices.



Gap2 **No zero-copy IPC middleware for arbitrary ROS message types**

Autoware relies heavily on unsized message types (e.g., those containing std::vector), requiring a middleware for zero-copy IPC. Current solutions cannot achieve both fault isolation and zero-copy for arbitrary message types.



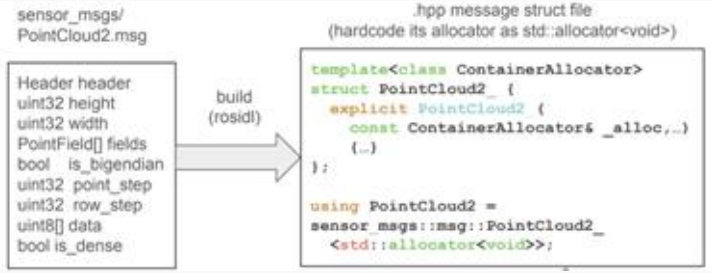
Gap2 Detail: Zero-Copy IPC for Unsized Types

The Problem: SHMEM Redirection

Fields like `std::vector` trigger reallocations at arbitrary times.For zero-copy, **custom allocators must redirect these to shared memory.**

The Barrier: Hardcoded rosidl

The `rosidl` tool (generating headers from `.msg` files) hardcodes the default allocator in message structs. This creates a **type mismatch** when an application attempts to use a custom allocator.




```
template<typename T>
class MyAllocator {...};

MyAllocator<void> my_allocator;

auto output_msg = std::make_unique
  <sensor_msgs::msg::PointCloud2_
  <MyAllocator<void>>(my_allocator);
```

Why existing (without Agnocast) attempts fail:

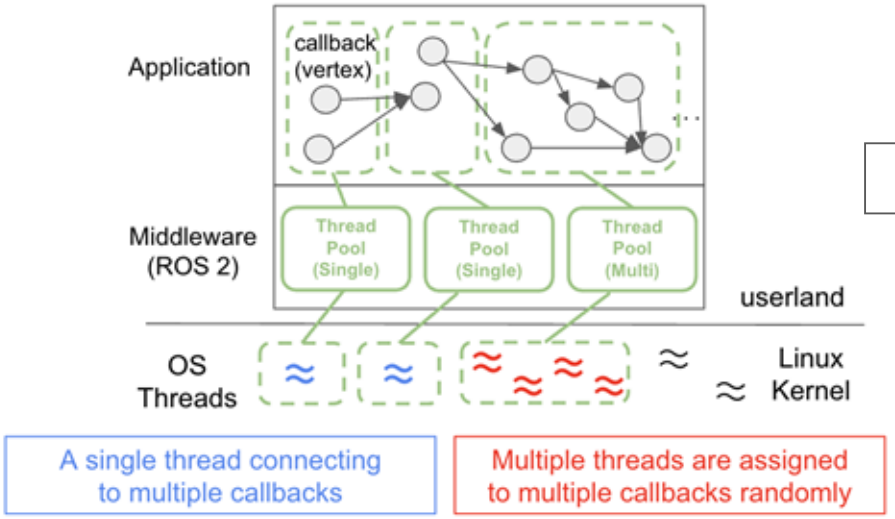
- **Custom Allocators:** Changes the type signature, causing system-wide incompatibility (as described in left).
- **Polymorphic Allocators:** Decouples allocator from the container type signature, yet introduces incompatibility between normal and polymorphic-aware container instances.
- **Implicitly Bounded Fields:** Making unsized message fields implicitly bounded in rosidl without changing API to app code (rclcpp #2201), yet same as above

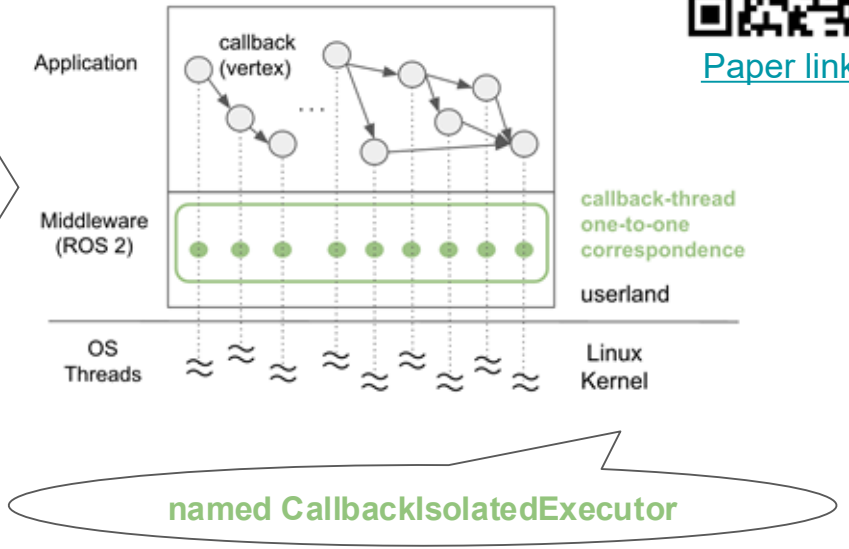
Despite various attempts to fix this, all current approaches lead to "type mismatch" errors due to ROS 2's "already rosidl-generated" message ecosystem.

Paper 1 — RTAS BP 2025: CallbackIsolatedExecutor (CIE)

T. Ishikawa-Aso, A. Yano, T. Azumi, and S. Kato, “Work in progress: Middleware-transparent callback enforcement in commoditized component-oriented real-time systems” (RTAS Brief Paper 2025)

Paper link





per-callback scheduler configuration is impossible

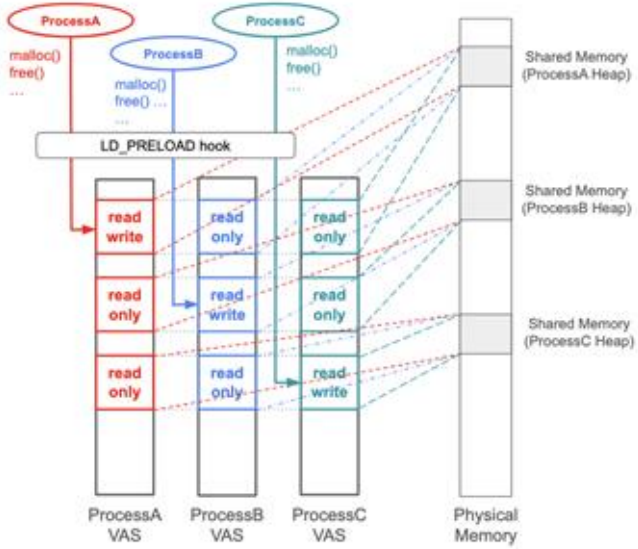
per-callback scheduler configuration is possible

Paper 2 — ISORC 2025: Agnocast prototype

T. Ishikawa-Aso and S. Kato, “ROS 2 Agnocast: Supporting unsized message types for true zero-copy publish/subscribe IPC,” (ISORC 2025)

Publishers expose their parts of heap directly to subscribers with the same offset → raw pointer is still valid across processes → All ROS message types can be IPC-ed

Paper link




```cpp
using T = sensor_msgs::msg::PointCloud2;

agnocast::Publisher<T>::SharedPtr pub;
pub = agnocast::create_publisher<T>("mytopic")
    ;

void callback(const T::ConstPtr input) {
  agnocast::ipc_shared_ptr<T> output =
    publisher->borrow_loaded_message();

  fill(input, output);

  pub->publish(std::move(output));
}
```

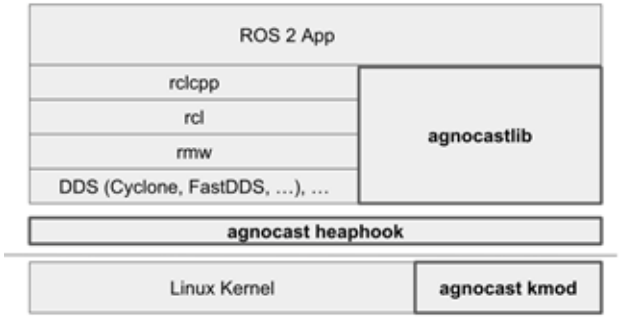
Publisher

```cpp
using T = sensor_msgs::msg::PointCloud2;

void callback(const
    agnocast::ipc_shared_ptr<T> input) {
    ...
}

agnocast::Subscription<T>::SharedPtr sub;
sub = agnocast::create_subscription<T>("
    mytopic", ...);
```

Subscriber

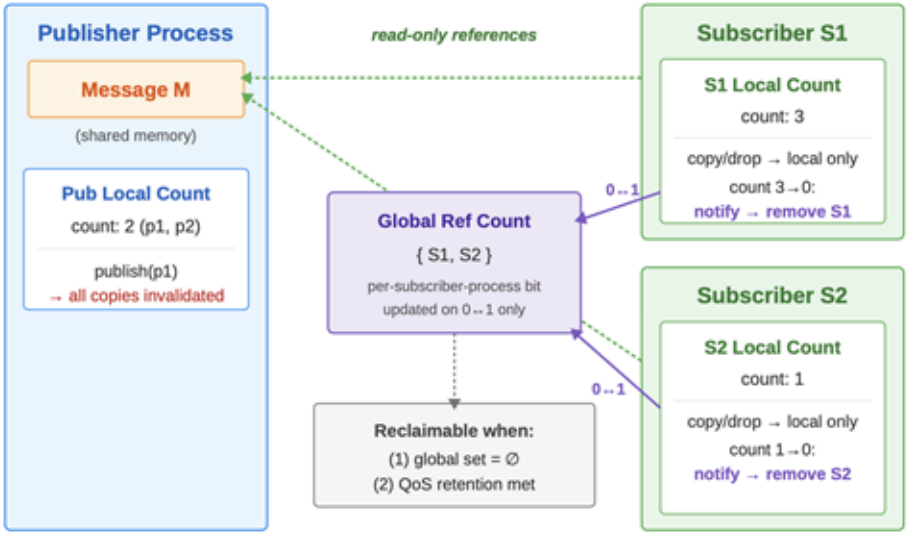


Paper 3 — ISORC 2026: Agnocast ipc_shared_ptr

T. Ishikawa-Aso, A. Yano, K. Imai, T. Azumi, and S. Kato, “ipc_shared_ptr: A Publish/Subscribe-Aware Smart Pointer for Cross-Process Object Lifetime Management.” (ISORC 2026)

Preprint link

Cross-process object lifetime — when can a message be reclaimed? → Pub/sub structural properties collapse distributed ref counting into **per-subscriber 0↔1 transitions**.



- **Transparent API**: std::shared_ptr-equivalent copy/move/destroy on both pub & sub sides
- **Publisher**: publish() invalidates all remaining copies → post-publish access = runtime error. Without publish(): local dealloc like std::shared_ptr (no IPC)
- **Subscriber**: copy/drop is local-only (no IPC); only local 1→0 triggers a kernel call, removing from global ref count

Where this leaves Autoware

- Both barriers between RT scheduling theory and Autoware deployment — **nested scheduling**, and **the lack of true zero-copy IPC** under Autoware constraints — are now removed.
- Three peer-reviewed papers in two years (RTAS '25, ISORC '25, ISORC '26)
- All three are also production-grade middleware, apt-installable, running on real autonomous vehicles. Section 2 shows that side.
- Years of RT scheduling theory can now flow into Autoware deployment. This is what 'Autoware as a real-time systems research platform' means in practice.

Autoware Deployment

Officially adopting CallbackIsolatedExecutor + Agnocast

02

Public roadmap: Autoware Middleware Strategy

Autoware officially adopting CallbackIsolatedExecutor + Agnocast

- **March 2026**: Replaced the middleware of the entire Autoware with Agnocast (from rclcpp) in an internal TIER IV project and conducted performance evaluation.
- **Now (May 2026)**: CallbackIsolatedExecutor has been introduced in most major nodes, and the majority of Perception nodes have migrated from rclcpp to Agnocast (behind a env variable, disabled by default).
- **Planned (September 2026)**: CallbackIsolatedExecutor + Agnocast will be adopted in almost all Autoware nodes.

Internal evidence: TIER IV bus project

Performance evaluation of full-stack Autoware middleware replacement with Agnocast in an internal TIER IV bus project

- Confirmed normal operation even after replacing the entire stack from rclcpp (ROS 2 C++) to Agnocast
- Launch time reduced from **several minutes to under 20 seconds**: Eliminated DDS Discovery Protocol bottlenecks
- **20% reduction in CPU consumption** (Average CPU usage: 88.1% → 72.3%): Wasteful underlying network activity from DDS, etc., completely vanished → Enabling more essential performance evaluation

Autoware, by Q3 2026, is a clean research platform

- By ~September 2026, your ROS 2-based evaluation can run on the largest open-source AD platform that has both **no executor-layer scheduling** and a **true zero-copy IPC layer** satisfying realistic constraints.
- Pre-validated at production scale on TIER IV's bus codebase.
- Plan your next paper around this configuration.

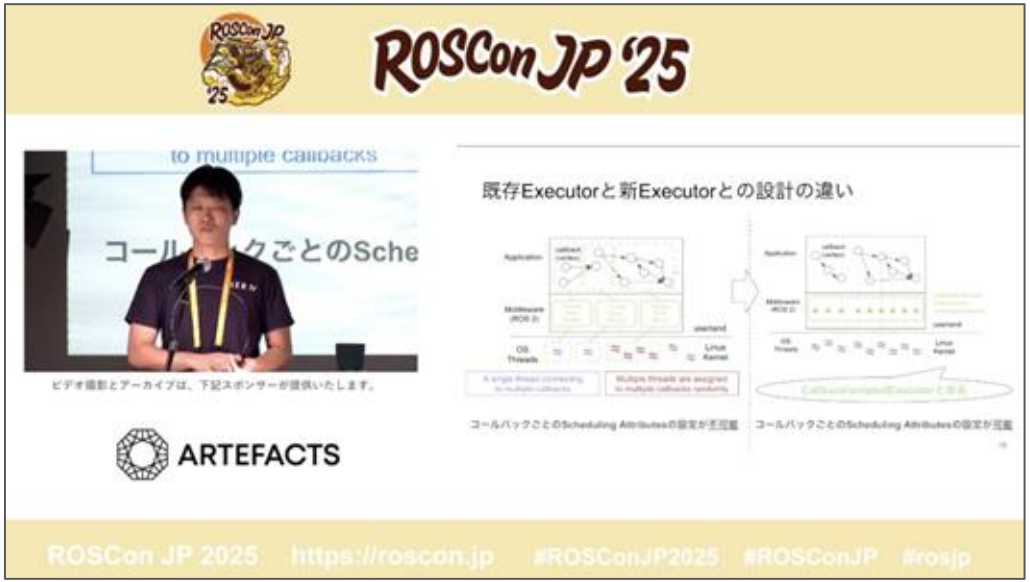
Hands-On (1)

How to use CallbackIsolatedExecutor

03

CallbackIsolatedExecutor in ROS Community

CallbackIsolatedExecutor was presented at ROSConJP '25, and its integration into Autoware is gaining recognition within the community.



ROSConJP video (japanese)

Article about video (english)

How to install CallbackIsolatedExecutor

Since CallbackIsolatedExecutor (CIE) is bundled with Agnocast, you can use CIE (independent of Agnocast) by installing agnocastlib as follows:

```
// After ROS 2 software stack installed (in ROS 2 Jazzy)
sudo apt update
sudo apt install ros-jazzy-agnocastlib
```

See Agnocast Doc for details

A standalone repository for CIE is also available (apt install support planned for the future)

→ https://github.com/autowarefoundation/callback_isolated_executor

Application Changes Required for CIE Integration

1) Replace the Executor in the code (right)

2) For ComposableNodes, replace the Component Container in the launch file with the CIE equivalent (below)

```cpp
#include "static_callback_isolated_executor.hpp"

int main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node = std::make_shared<SampleNode>();
  auto executor = std::make_shared<CallbackIsolatedExecutor>();

  executor->add_node(node);
  executor->spin();

  rclcpp::shutdown();
  return 0;
}
```

```xml
<launch>
  <node_container pkg="callback_isolated_executor" exec="component_container_callback_isolated"
    name="sample_container" namespace="">
    <composable_node pkg="cie_sample_application" plugin="SampleNode" name="sample_node" namespace="">
      ...
    </composable_node>
  </node_container>
</launch>
```

See Agnocast Doc for details

Configure System-wide Callbacks via YAML File

Configure each CallbackGroup across the system by filling properties in the auto-generated YAML format

See Agnocast Doc for details

```yaml
callback_groups:
  - id: xxxxx
    affinity:
      - 0
      - 1
    policy: SCHED_OTHER
    priority: -10

  - id: yyyyy
    affinity:
      - 2
      - 3
    policy: SCHED_FIFO
    priority: 50

...
```

CFS

```yaml
- id: xxxxx
  affinity:
    - 0
    - 1
  policy: SCHED_OTHER
  priority: -10
```

FIFO Scheduler

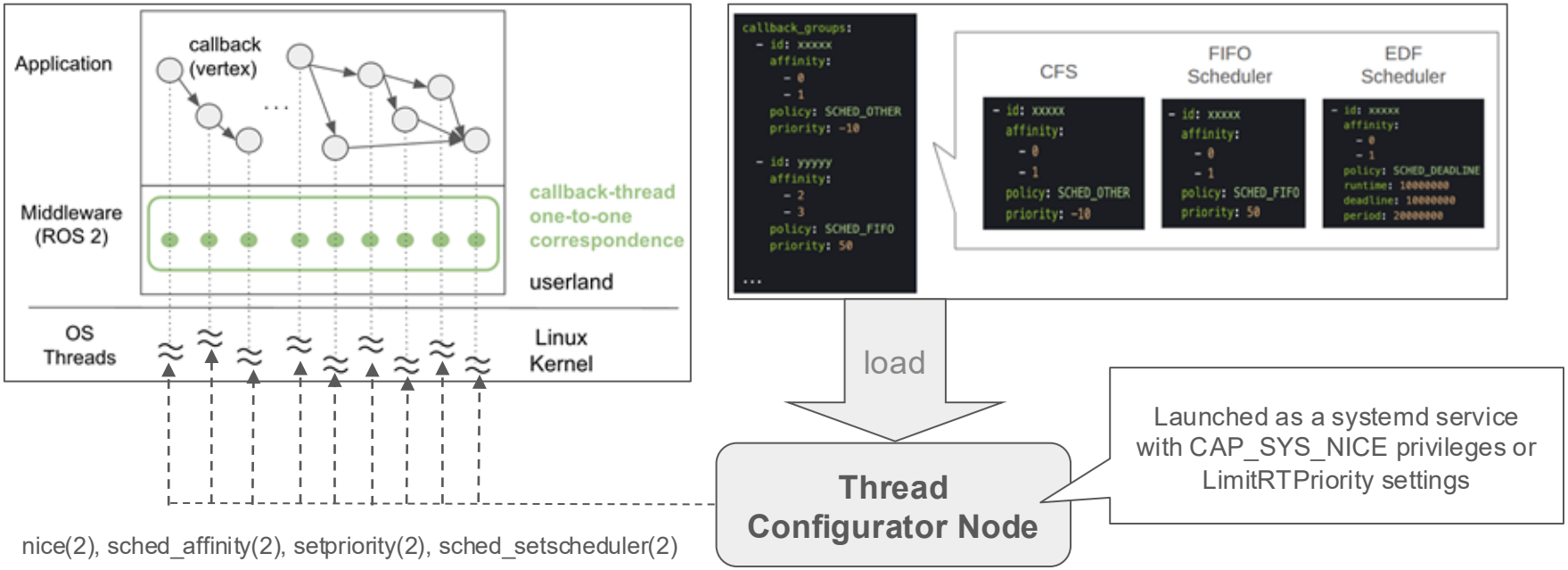
```yaml
- id: xxxxx
  affinity:
    - 0
    - 1
  policy: SCHED_FIFO
  priority: 50
```

EDF Scheduler

```yaml
- id: xxxxx
  affinity:
    - 0
    - 1
  policy: SCHED_DEADLINE
  runtime: 10000000
  deadline: 10000000
  period: 20000000
```

Mechanism for Thread Configuration

The Thread Configurator node configures the threads of the CallbackIsolatedExecutor



Hands-On (2)

How to use Agnocast

04

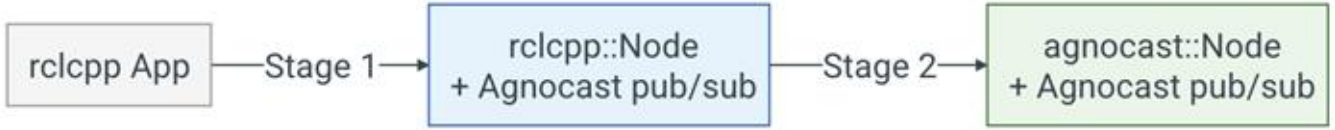
Agnocast in ROS Community

Agnocast was presented at ROSCon Global ‘25, and its integration into Autoware is gaining recognition within the community.

ROSCon Global video

ROS Discourse (english)

Migration from rclcpp to Agnocast (stage1)



See Agnocast Doc for details

```cpp
class MyPublisher : public rclcpp::Node
{
  agnocast::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;

  void timer_callback()
  {
    auto msg = pub_->borrow_loaned_message();
    msg->data = "Hello, world!";
    pub_->publish(std::move(msg));
  }

public:
  MyPublisher() : Node("my_publisher")
  {
    auto group = create_callback_group(rclcpp::CallbackGroupType::MutuallyExclusive);

    pub_ = agnocast::create_publisher<std_msgs::msg::String>(
      this, "/topic", 10);
    timer_ = create_wall_timer(100ms,
      std::bind(&MyPublisher::timer_callback, this), group);
  }
};
```

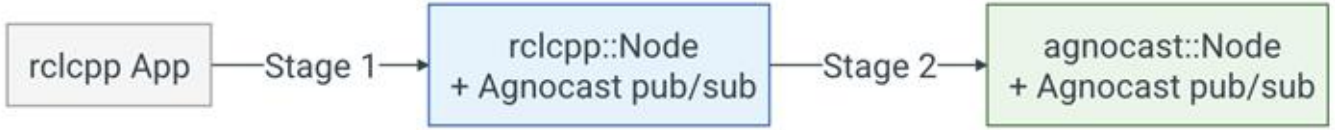
```cpp
class MySubscriber : public rclcpp::Node
{
  agnocast::Subscription<std_msgs::msg::String>::SharedPtr sub_;

  void callback(
    const agnocast::ipc_shared_ptr<std_msgs::msg::String> & msg)
  {
    RCLCPP_INFO(get_logger(), "Received: %s", msg->data.c_str());
  }

public:
  MySubscriber() : Node("my_subscriber")
  {
    auto group = create_callback_group(rclcpp::CallbackGroupType::MutuallyExclusive);
    agnocast::SubscriptionOptions options;
    options.callback_group = group;

    sub_ = agnocast::create_subscription<std_msgs::msg::String>(
      this, "/topic", 10,
      std::bind(&MySubscriber::callback, this, std::placeholders::_1),
      options);
  }
};
```

Migration from rclcpp to Agnocast (stage2)



See Agnocast Doc for details

```cpp
class MyPublisher : public agnocast::Node
{
  agnocast::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  agnocast::TimerBase::SharedPtr timer_;

  void timer_callback()
  {
    auto msg = pub_->borrow_loaned_message();
    msg->data = "Hello, world!";
    pub_->publish(std::move(msg));
  }

public:
  MyPublisher() : Node("my_publisher")
  {
    auto group = create_callback_group(rclcpp::CallbackGroupType::MutuallyExclusive);

    pub_ = this->create_publisher<std_msgs::msg::String>(
      "/topic", 10);
    timer_ = this->create_wall_timer(100ms,
      std::bind(&MyPublisher::timer_callback, this), group);
  }
};
```

```cpp
lass MySubscriber : public agnocast::Node

 agnocast::Subscription<std_msgs::msg::String>::SharedPtr sub_;

 void callback(const agnocast::ipc_shared_ptr<std_msgs::msg::String> & msg)
 {
   RCLCPP_INFO(get_logger(), "Received: %s", msg->data.c_str());
 }

ublic:
 MySubscriber() : Node("my_subscriber")
 {
   auto group = create_callback_group(rclcpp::CallbackGroupType::MutuallyExclusive
   agnocast::SubscriptionOptions options;
   options.callback_group = group;

   sub_ = this->create_subscription<std_msgs::msg::String>(
     "/topic", 10,
     std::bind(&MySubscriber::callback, this, std::placeholders::_1),
     options);
 }
;
```

Closing

05

Closing

- By utilizing **CallbackIsolatedExecutor** and **Agnocast**, real-time scheduling experiments for ROS 2 can be conducted in conditions much closer to ideal.
- In particular, static priority preemptive DAG scheduling on Linux can be performed in a nearly ideal state.
- The entirety of **Autoware** is scheduled to fully migrate to CallbackIsolatedExecutor and Agnocast by the end of September 2026.
- Future Work: Integration with Linux sched_ext is planned to support a wider range of scheduling algorithms.

awf/agnocast

awf/callback_isolated_executor

Acknowledgement

This research is based on results obtained from a project, JPNP21027, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).