

CallbackIsolatedExecutor: 二重スケジューリングを不要にする 新しいExecutorとスケジューリング理論

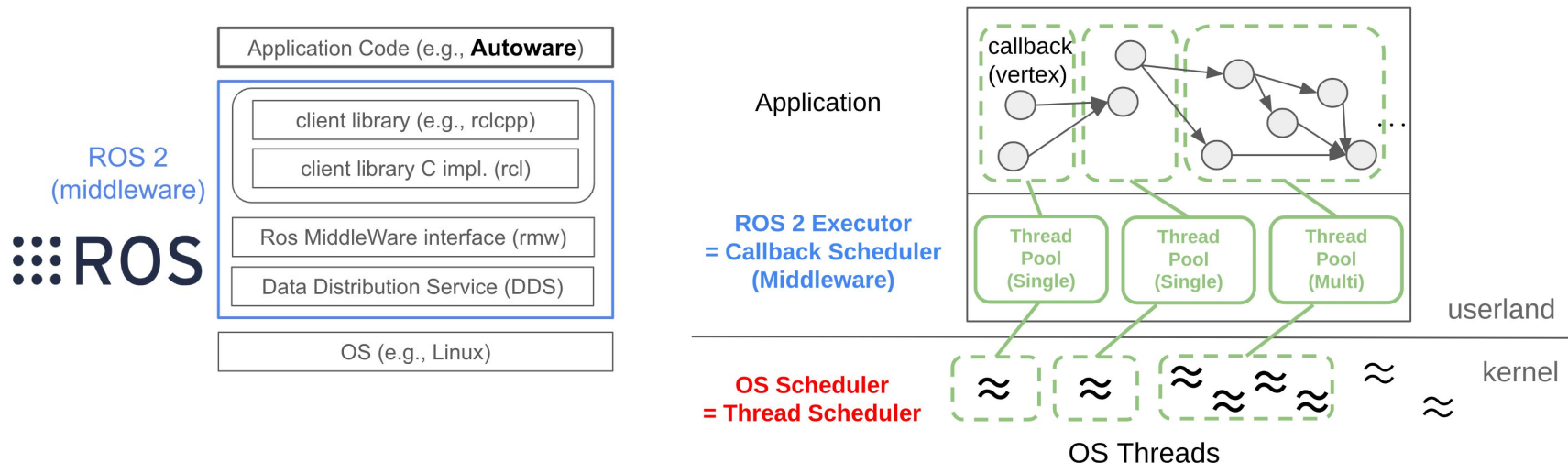
株式会社ティアフォー
システムソフトウェア部 部長

石川貴大 (Takahiro Ishikawa-Aso) @sywker

今日の話題: Executorとリアルタイムスケジューリング

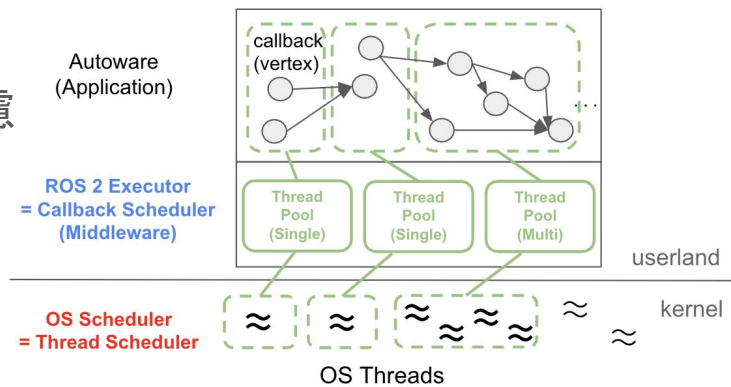
ROS 2アプリケーションはコールバックをvertexとしたDAG形状をしている

コールバックは二重スケジューリングの対象: **callback scheduler** と **thread scheduler**



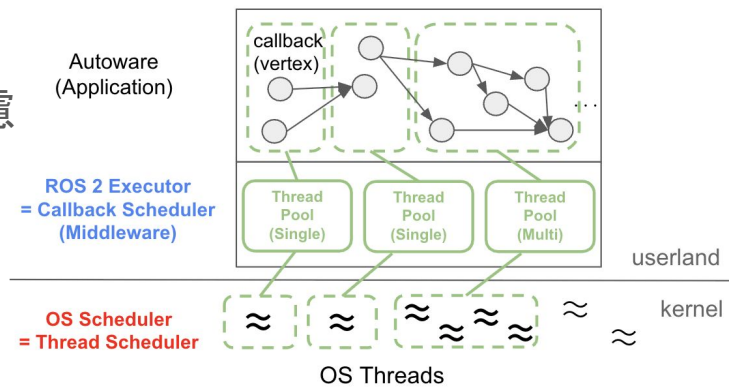
要約

- ROS 2のExecutor (ミドルウェアレイヤのタスクスケジューラ) の存在によって、OSレイヤとの間で二重スケジューリングが発生→リアルタイムスケジューリングの複雑性高
- 新しいExecutorであるCallbackIsolatedExecutor (CIE) を考案した
- CIEによって、コールバックに対して優先度やAffinityを直接指定可能に→Executorの存在を考慮不要になり、二重スケジューリングが解消
- CIEの導入によって、Autowareベースのロボタクシーで最悪Response Timeが5倍改善 (計測)



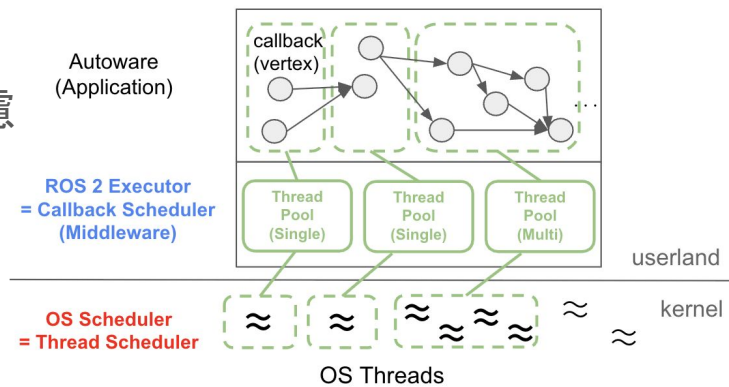
要約

- ROS 2の**Executor** (ミドルウェアレイヤのタスクスケジューラ) の存在によって、OSレイヤとの間で**二重スケジューリング** が発生→リアルタイムスケジューリングの複雑性高
- 新しいExecutorであるCallbackIsolatedExecutor (CIE) を考案した
- CIEによって、コールバックに対して優先度やAffinityを直接指定可能に→Executorの存在を考慮不要になり、二重スケジューリングが解消
- CIEの導入によって、Autowareベースのロボタクシーで最悪Response Timeが5倍改善 (計測)



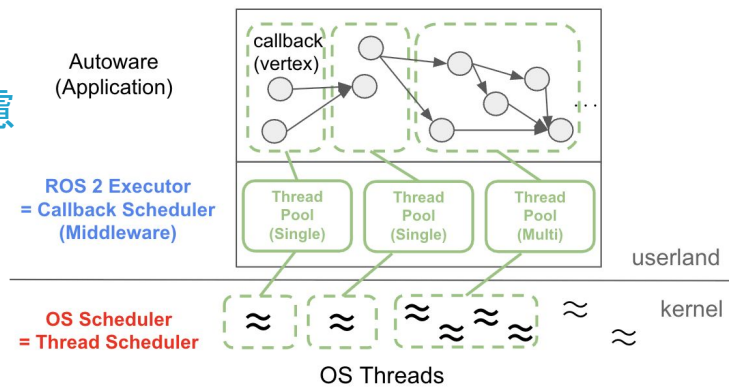
要約

- ROS 2のExecutor (ミドルウェアレイヤのタスクスケジューラ) の存在によって、OSレイヤとの間で二重スケジューリングが発生→リアルタイムスケジューリングの複雑性高
- 新しいExecutorである**CallbackIsolatedExecutor (CIE)** を考案した
- CIEによって、コールバックに対して優先度やAffinityを直接指定可能に→Executorの存在を考慮不要になり、二重スケジューリングが解消
- CIEの導入によって、Autowareベースのロボタクシーで最悪Response Timeが5倍改善 (計測)



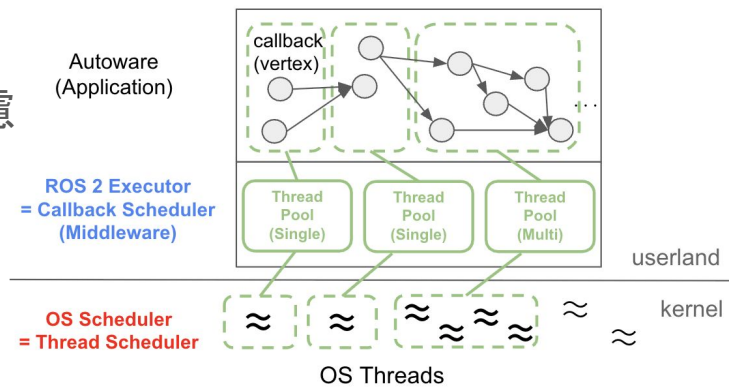
要約

- ROS 2のExecutor (ミドルウェアレイヤのタスクスケジューラ) の存在によって、OSレイヤとの間で二重スケジューリングが発生→リアルタイムスケジューリングの複雑性高
- 新しいExecutorであるCallbackIsolatedExecutor (CIE) を考案した
- CIEによって、コールバックに対して優先度やAffinityを直接指定可能に→Executorの存在を考慮不要になり、二重スケジューリングが解消
- CIEの導入によって、Autowareベースのロボタクシーで最悪Response Timeが5倍改善 (計測)



要約

- ROS 2のExecutor (ミドルウェアレイヤのタスクスケジューラ) の存在によって、OSレイヤとの間で二重スケジューリングが発生→リアルタイムスケジューリングの複雑性高
- 新しいExecutorであるCallbackIsolatedExecutor (CIE) を考案した
- CIEによって、コールバックに対して優先度やAffinityを直接指定可能に→Executorの存在を考慮不要になり、二重スケジューリングが解消
- CIEの導入によって、Autowareベースのロボタクシーで最悪Response Timeが5倍改善 (計測)



CallbackIsolatedExecutorの論文がRTAS '25 Brief Paperに採択

プレプリントがarXivで取得可能 (IEEEでも公開済)。本発表はこの論文の内容を含む

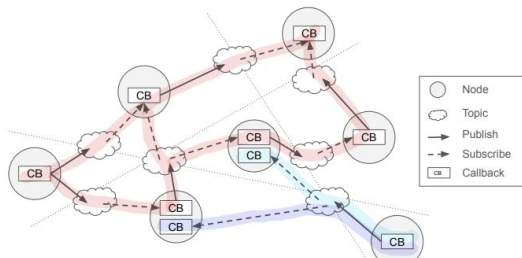
Work in Progress: Middleware-Transparent Callback Enforcement in Commoditized Component-Oriented Real-time Systems

Takahiro Ishikawa-Aso^{†‡}, Atsushi Yano^{*‡}, Takuya Azumi^{*}, Shinpei Kato[†]

[†]The University of Tokyo, Japan ^{*}Saitama University, Japan

[‡]TIER IV Incorporated, Japan

Abstract—Real-time scheduling in commoditized component-oriented real-time systems, such as ROS 2 systems on Linux, has been studied under nested scheduling: OS thread scheduling and middleware layer scheduling (e.g., ROS 2 Executor). However, by establishing a persistent one-to-one correspondence between callbacks and OS threads, we can ignore the middleware layer and directly apply OS scheduling parameters (e.g., scheduling policy, priority, and affinity) to individual callbacks. We propose a middleware model that enables this idea and implements *CallbackIsolatedExecutor* as a novel ROS 2 Executor. We demonstrate that the costs (user-kernel switches, context switches, and memory usage) of *CallbackIsolatedExecutor* remain lower than



<https://arxiv.org/pdf/2505.06546>

CallbackIsolatedExecutor:
二重スケジューリングを不要にする
新しいExecutorとスケジューリング理論

RTAS '25 Brief Paper の貢献を含む

01 / 背景

02 / CallbackIsolatedExecutorの
設計と実装

03 / CallbackIsolatedExecutorによる
スケジューリング問題の単純化

04 / Autowareへの適用と評価

05 / CallbackIsolatedExecutorの使い方

CallbackIsolatedExecutor:
二重スケジューリングを不要にする
新しいExecutorとスケジューリング理論

RTAS '25 Brief Paper の貢献を含む

01 / 背景

02 / CallbackIsolatedExecutorの
設計と実装

03 / CallbackIsolatedExecutorによる
スケジューリング問題の単純化

04 / Autowareへの適用と評価

05 / CallbackIsolatedExecutorの使い方

背景

01

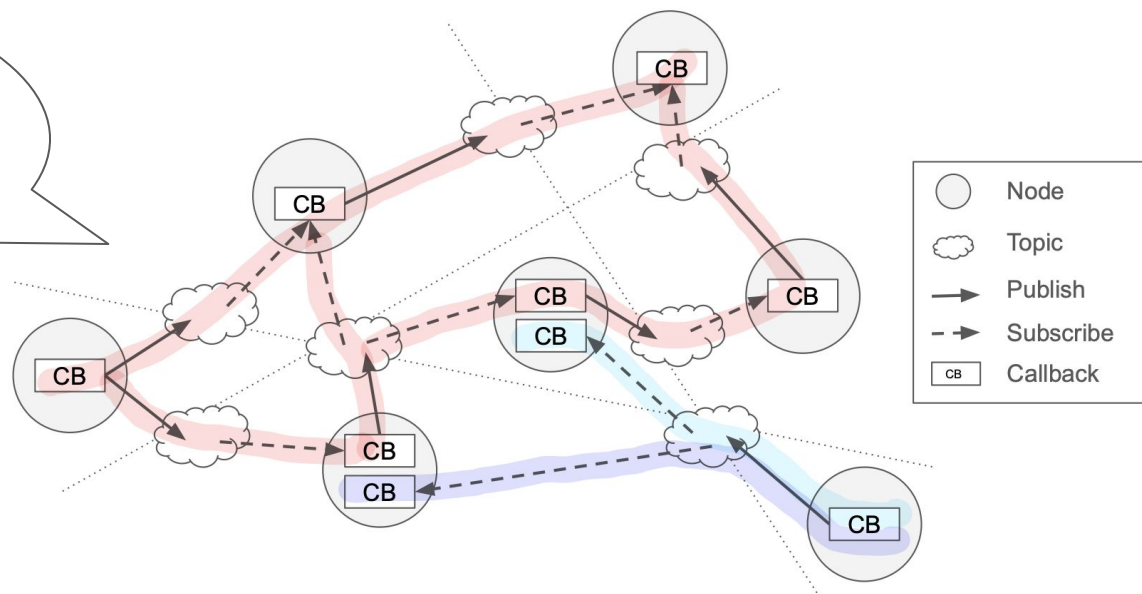


ROS 2における時間制約付きのデータフロー

Node内callback同士の実行順序の依存関係が、システム全体にデータフローを形成

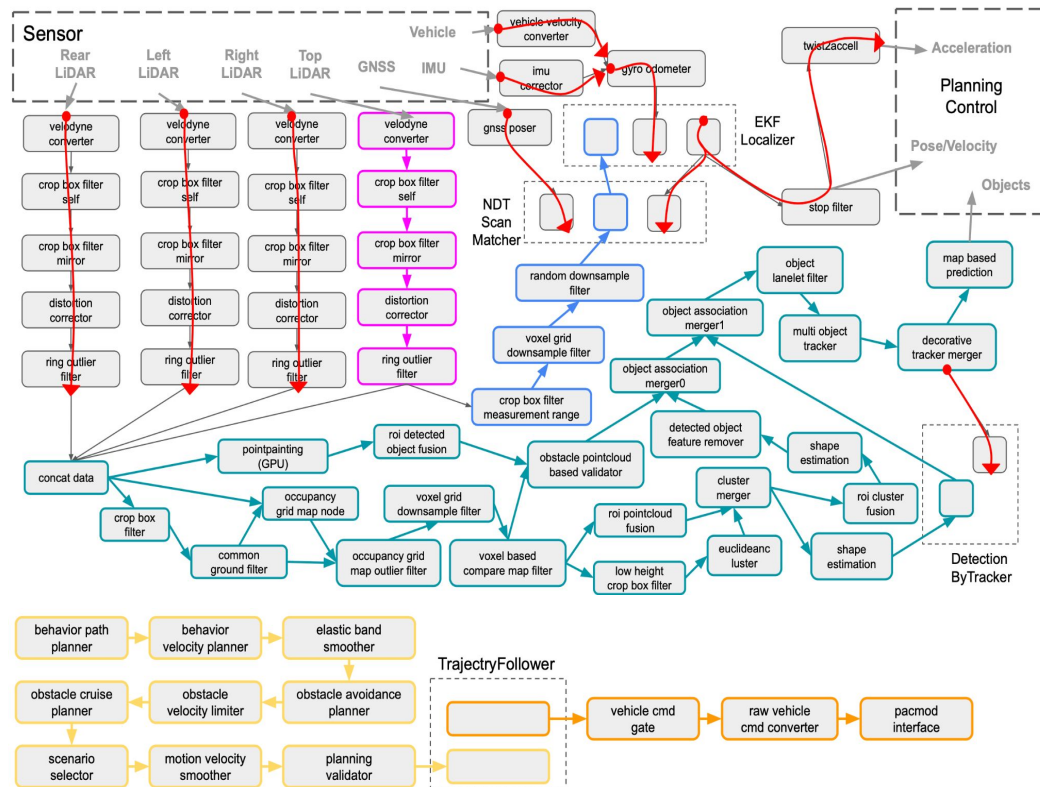
データフローはそれぞれ時間制約を持つ (**end-to-end response time**)

学術的には、**DAG**や**Cause Effect Chain**としてリアルタイムスケジューリングのためにモデル化される



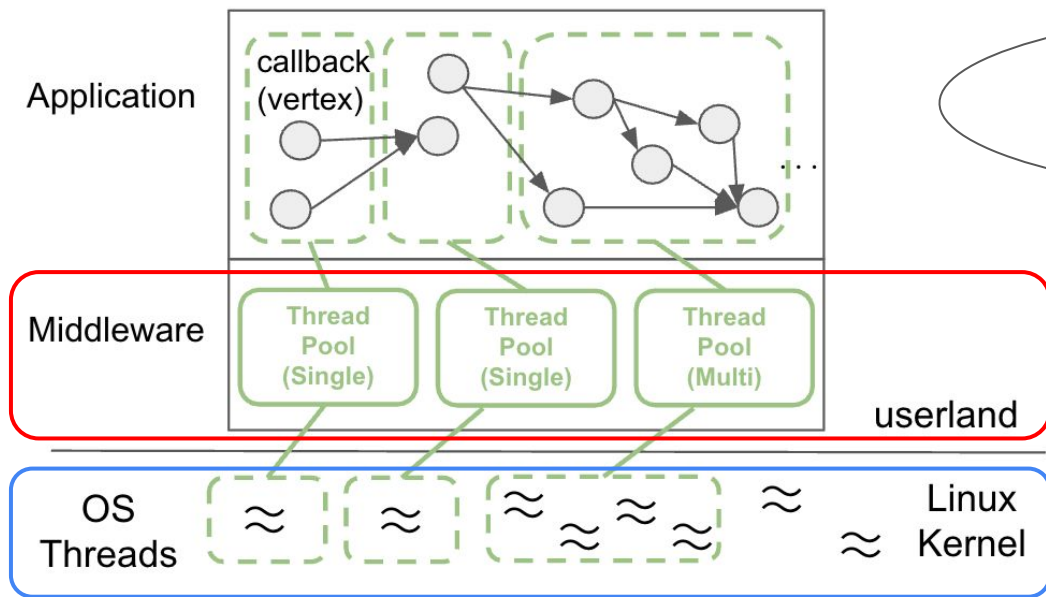
ティアフォーの事例: Autowareの時間制約付きDAG

- Autowareも例に漏れず、複数の時間制約付きDAGから構成される
- 現在は主に5つのDAGをボトルネックと認識している: **Top LiDAR**
Preprocessing, **Localization**,
Perception, **Planning**, **Control**
- 5つのResponse Timeを最適化したい
- 理想: その最適化のために、コールバック (DAGのvertex) に直接 Scheduling Attributes (優先度やコアアフィニティなど) を設定したい



ROS 2におけるリアルタイムスケジューリング問題

リアルタイムの学术界は、ROS 2に対してずっと**二重スケジューリング**を想定してきた



この複雑性が、伝統的な
スケジューリングアルゴリズムの
効果的な活用を阻んできた

Scheduler in Middleware: コールバックをスケジューリング。ROS 2 特有のラウンドロビンのアルゴリズム

Scheduler in OS: スレッドをスケジューリング。EDF/FIFO/ CFSなどのポリシー

※ EventExecutorにも同様に二重スケジューリングの問題が発生

ROS 2リアルタイムスケジューリングの先行研究

二重スケジューリングを前提として、大量の論文がこれまで発表されている

- ...
- (ECRTS '19) D. Casini, T. Blaß, I. Lutkebohle, and B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling,"
- (RTSS '20) Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, and W. Yi, "Response time analysis and priority assignment of processing chains on ROS2 executors,"
- (RTAS '21) J. T. Blass, A. Hamann, R. Lange, D. Ziegenbein, and B. B. Brandenburg, "Automatic latency management for ROS 2: Benefits, challenges, and open problems,"
- (RTAS '21) H. Choi, Y. Xiang, and H. Kim, "PiCAS: New design of priority-driven chain-aware scheduling for ROS2,"
- (RTSS '21) T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, "A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance,"
- (RTSS '22) X. Jiang, D. Ji, N. Guan, R. Li, Y. Tang, and Y. Wang, "Real-time scheduling and analysis of processing chains on multi-threaded executor in ROS 2,"
- (IROS '22) J. A. Partap, S. Grayson, M. Huzaifa, S. Adve, B. Godfrey, S. Gupta, K. Hauser, and R. Mittal, "On-device CPU scheduling for robot systems,"
- (RTSS '22) H. Teper, M. Gunzel, N. Ueter, G. von der Brüggen, and J.-J. Chen, "End-to-end timing analysis in ROS2,"
- (RTAS '23) Y. Tang, N. Guan, X. Jiang, X. Luo, and W. Yi, "Real-time performance analysis of processing systems on ROS 2 executors,"
- (RTAS '23) H. Sobhani, H. Choi, and H. Kim, "Timing analysis and priority-driven enhancements of ROS 2 multi-threaded executors,"
- (RTCSA '23) H. Teper, T. Betz, G. Von Der Brüggen, K.-H. Chen, J. Betz, and J.-J. Chen, "Timing-aware ROS 2 architecture and system optimization,"
- (TCAD '24) A. Al Arafat, K. Wilson, K. Yang, and Z. Guo, "Dynamic priority scheduling of multithreaded ROS 2 executor with shared resources,"
- (TCAD '24) H. Teper, D. Kuhse, M. Gunzel, G. von der Brüggen, F. Howar, and J.-J. Chen, "Thread carefully: Preventing starvation in the ROS 2 multithreaded executor,"
- (RTAS '24) H. Teper, T. Betz, M. Gunzel, D. Ebner, G. von der Brüggen, J. Betz, and J.-J. Chen, "End-to-end timing analysis and optimization of multi executor ROS 2 systems,"
- (RTAS '25) Harun Teper; Oren Bell; Mario Günzel; Chris Gill; Jian-Jia Chen, "Reconciling ROS 2 with Classical Real-Time Scheduling of Periodic Tasks"
- ...

CallbackIsolatedExecutor:
二重スケジューリングを不要にする
新しいExecutorとスケジューリング理論

RTAS '25 Brief Paper の貢献を含む

01 / 背景

**02 / CallbackIsolatedExecutorの
設計と実装**

03 / CallbackIsolatedExecutorによる
スケジューリング問題の単純化

04 / Autowareへの適用と評価

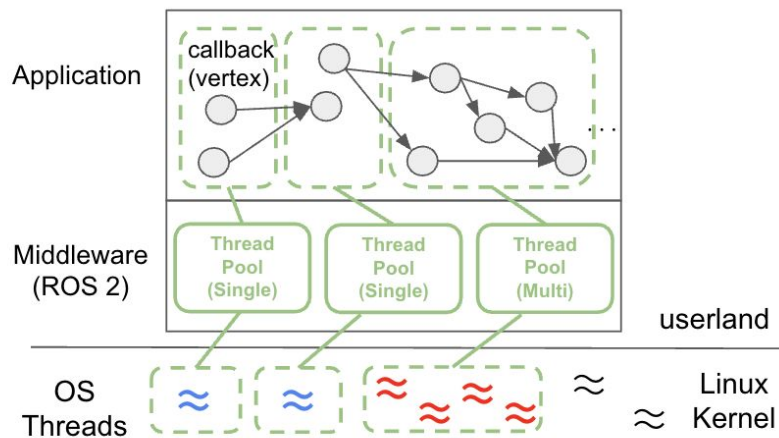
05 / CallbackIsolatedExecutorの使い方

CallbackIsolatedExecutorの 設計と実装

02



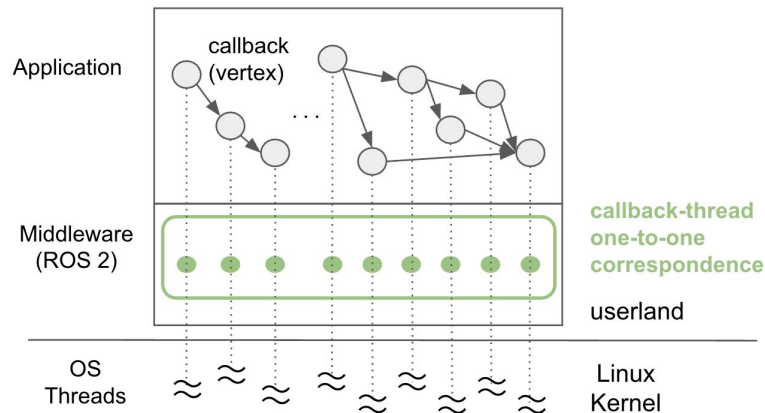
既存Executorと新Executorとの設計の違い



A single thread connecting to multiple callbacks

Multiple threads are assigned to multiple callbacks randomly

コールバックごとの Scheduling Attributes の設定が 不可能



CallbackIsolatedExecutor と命名

コールバックごとの Scheduling Attributes の設定が 可能

1つのYAMLファイルでシステム全体のコールバックを設定

自動生成されるYAMLフォーマットに対して、各プロパティを埋めるだけで、システム全体のCallbackGroupそれぞれを設定できる (使い方は後に紹介)

```
callback_groups:  
- id: xxxxx  
  affinity:  
    - 0  
    - 1  
  policy: SCHED_OTHER  
  priority: -10  
  
- id: yyyyy  
  affinity:  
    - 2  
    - 3  
  policy: SCHED_FIFO  
  priority: 50  
  
...
```

CFS

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

FIFO
Scheduler

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

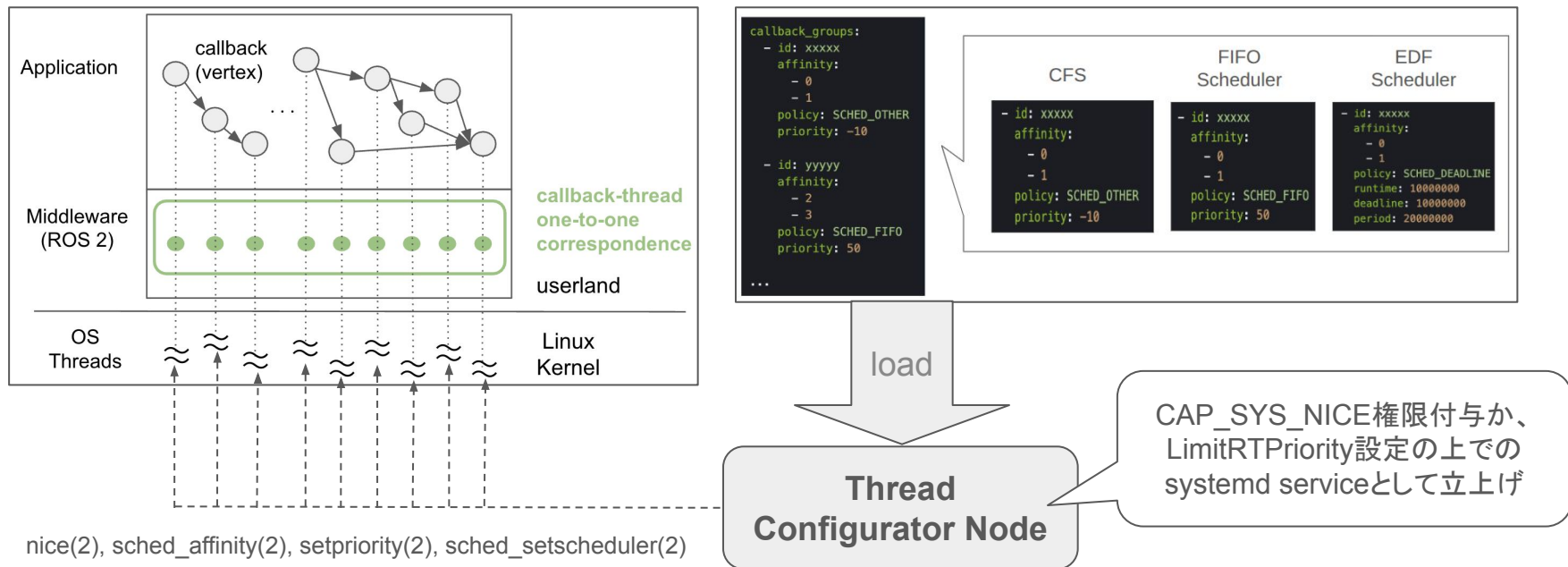
EDF
Scheduler

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

詳細はREADMEにて: https://github.com/tier4/callback_isolated_executor

スレッドの設定を行う仕組み

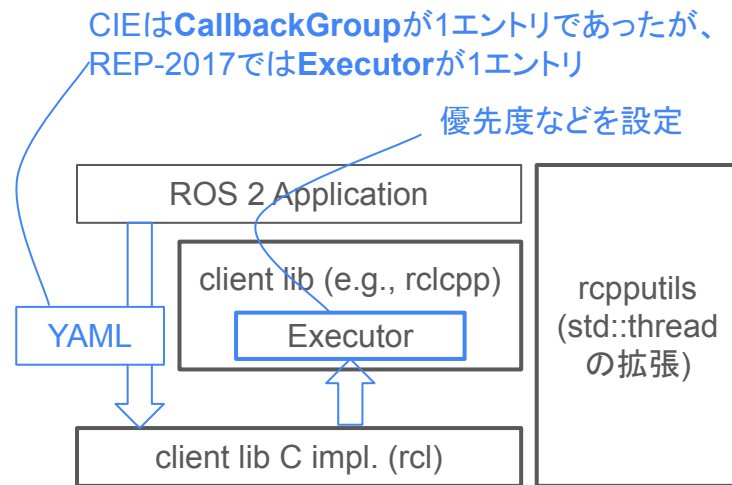
Thread ConfiguratorノードがCallbackIsolatedExecutorのスレッド群を実際に設定



REP-2017との比較

- eSOL様が2023年から取り組む、CIEと類似した取り組み。右上図のようにrclcpp, rclの変更とstd::threadの拡張を要する (PRが出ている)
- 右下図のように、CallbackGroupごとにSingleThreadedExecutorを作成しそれぞれ別のスレッドで実行させると、CIEと等価
- REP-2017に対するCIEのメリット
 - rclやrclcppに変更が一切不要であり、独立したパッケージをインストールするだけ
 - CIEのThread Configurator Nodeにのみ、CAP_SYS_NICE等の権限を付与すればよい (権限範囲が最小になるセキュリティ的利点)

※ CIEもサブ機能として、REP-2017と同様のExecutor単位の設定が可能



```
1 int main() {
2     ...
3     std::vector<std::thread> threads;
4     auto node = std::make_shared<rclcpp::Node>("example_node");
5     int i = 0;
6     node->for_each_callback_group(
7         [](rclcpp::CallbackGroup::SharedPtr group) {
8             ...
9             rclcpp::ExecutorOptions opts;
10            opts.name = "executor-" + std::to_string(i++); // tag
11            auto executor = std::make_shared<
12                rclcpp::executors::SingleThreadedExecutor>(ops);
13            threads.emplace_back([executor]() { executor->spin(); })
14        }
15    ...
16 }
```

CallbackIsolatedExecutor:
二重スケジューリングを不要にする
新しいExecutorとスケジューリング理論

RTAS '25 Brief Paper の貢献を含む

01 / 背景

02 / CallbackIsolatedExecutorの
設計と実装

**03 / CallbackIsolatedExecutorによる
スケジューリング問題の単純化**

04 / Autowareへの適用と評価

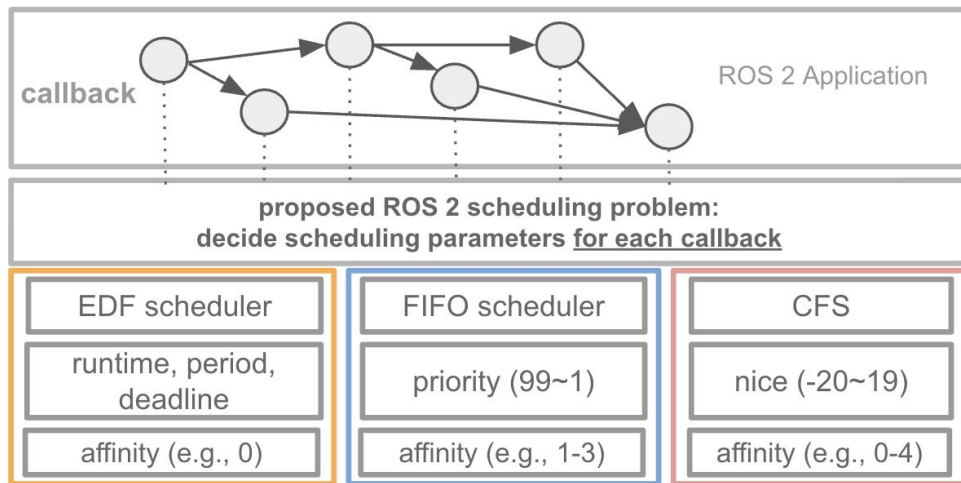
05 / CallbackIsolatedExecutorの使い方

CallbackIsolatedExecutorによる スケジューリング問題の単純化

03

Linux上のROS 2スケジューリング問題の単純化

CallbackIsolatedExecutorの導入により、Linux上のROS 2システムにおけるリアルタイムスケジューリング問題は、ミドルウェアレイヤを無視することが可能に



条件: CallbackGroup (並行実行させたくないコールバックをグループにする機能) 1つにつきコールバックを1つしか持たせない

CallbackIsolatedExecutor:
二重スケジューリングを不要にする
新しいExecutorとスケジューリング理論

RTAS '25 Brief Paper の貢献を含む

01 / 背景

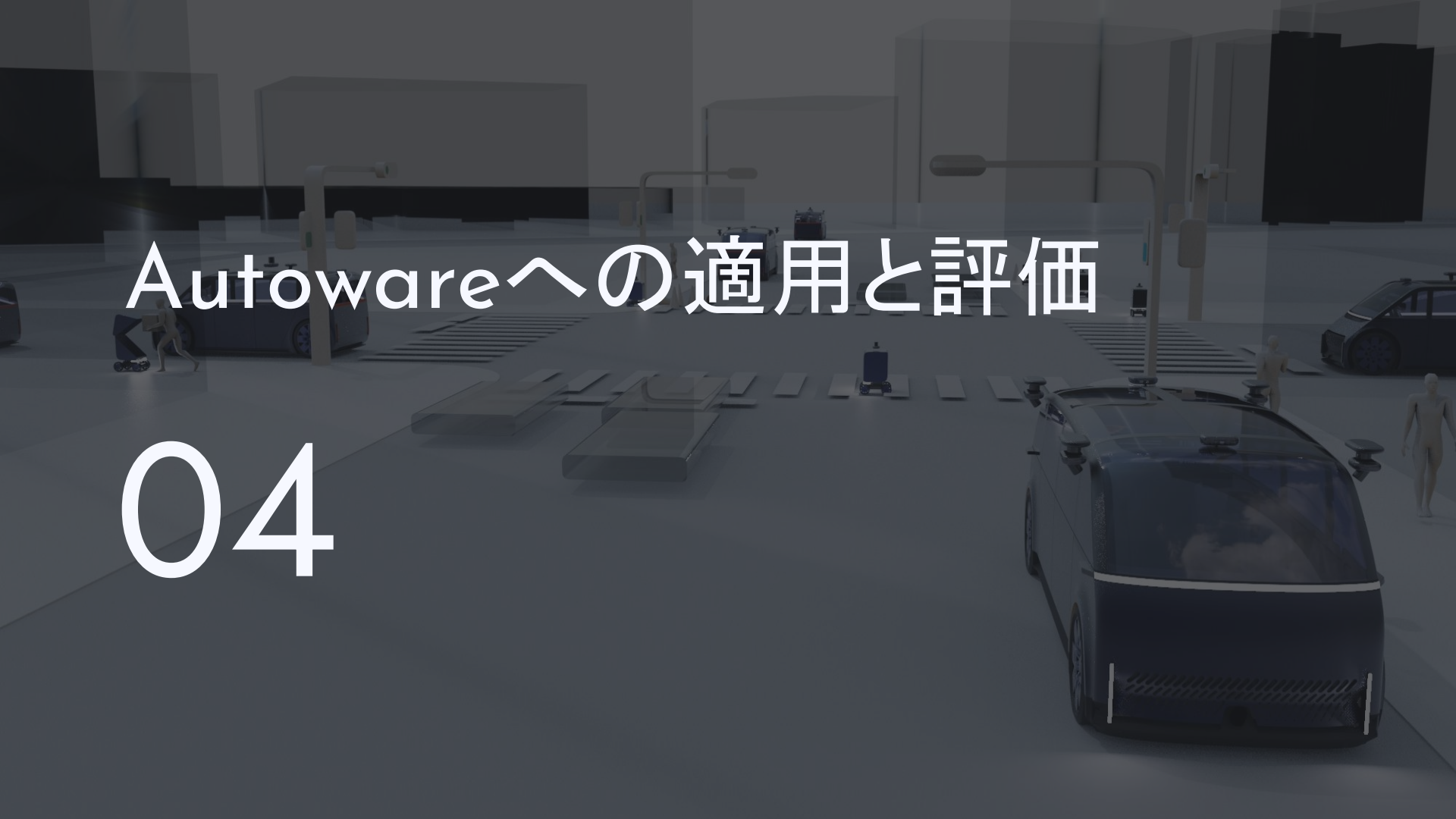
02 / CallbackIsolatedExecutorの
設計と実装

03 / CallbackIsolatedExecutorによる
スケジューリング問題の単純化

04 / Autowareへの適用と評価

05 / CallbackIsolatedExecutorの使い方

Autowareへの適用と評価

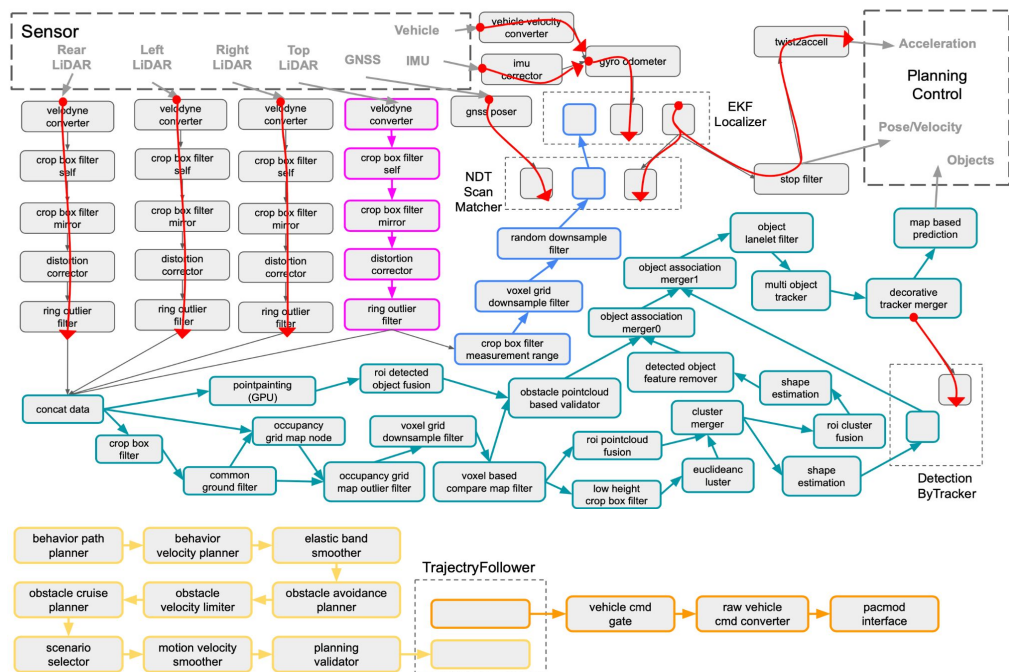


04

CallbackIsolatedExecutorをAutowareに適用

アルゴリズムは研究中であり現
状ヒューリスティック

ティアフォーのロボタクシーにて適用し、ボトルネックである5つのDAGを最適化



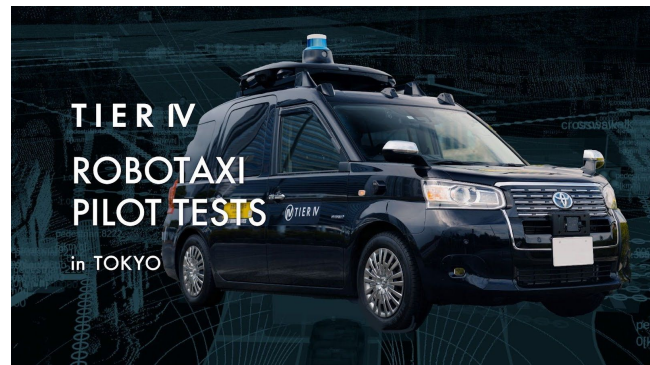
Top LIDAR

Localization

Perception

Planning

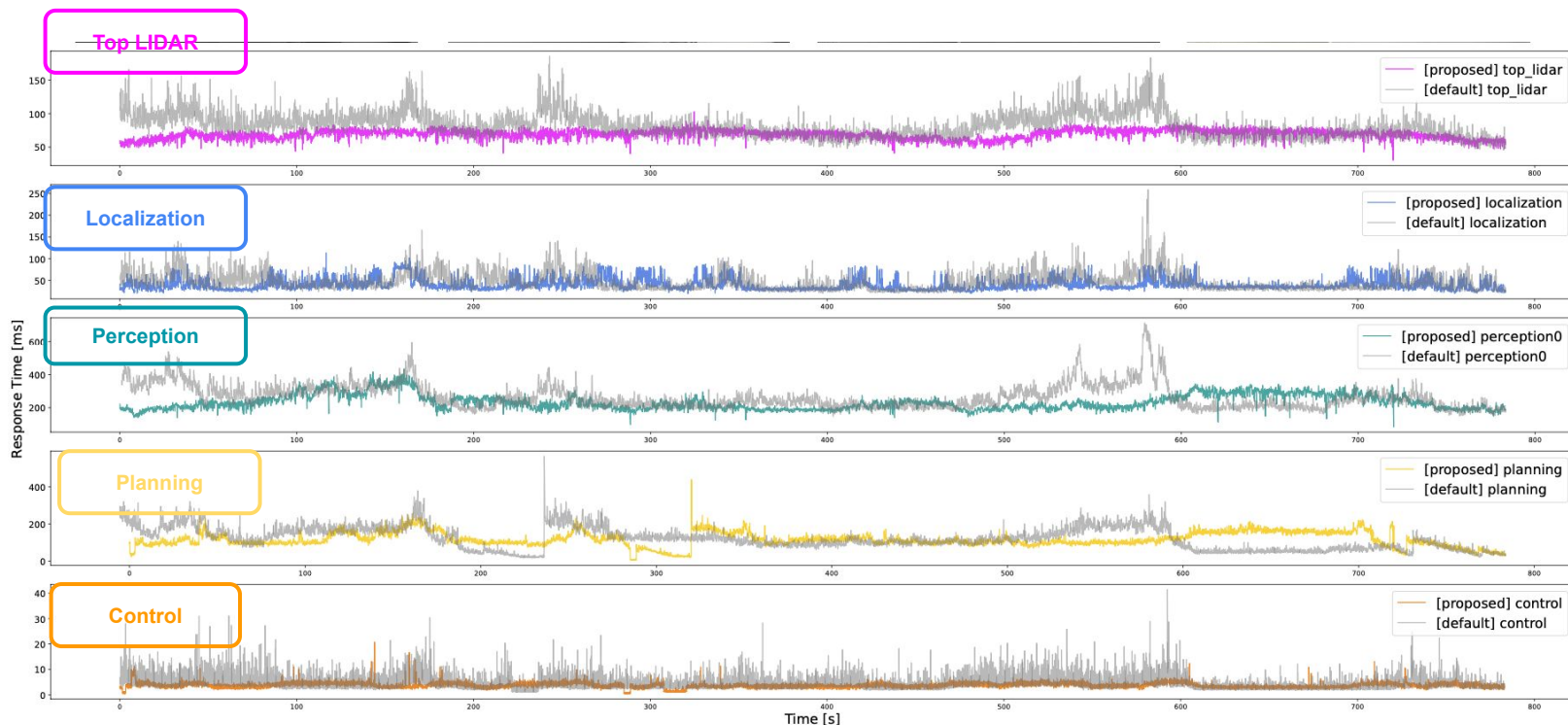
Control



https://tier4.jp/en/media/detail/?sys_id=1YIT156gL CvufxeUeumQ7y

ボトルネックである5つのDAGのResponse Time改善

CFS (灰色) でスケジューリングしたときとの比較。公道で計測。横軸は時間経過 (sec)



CallbackIsolatedExecutor:
二重スケジューリングを不要にする
新しいExecutorとスケジューリング理論

RTAS '25 Brief Paper の貢献を含む

01 / 背景

02 / CallbackIsolatedExecutorの
設計と実装

03 / CallbackIsolatedExecutorによる
スケジューリング問題の単純化

04 / Autowareへの適用と評価

05 / CallbackIsolatedExecutorの使い方

CallbackIsolatedExecutorの使い方

05



CallbackIsolatedExecutorのパッケージをインストール

CallbackIsolatedExecutorは独立したパッケージであり、普通にインストールするだけ
2025年9月現在まだソースビルドのみ提供だが、ROSビルドファームから配布準備中

```
$ git clone https://github.com/tier4/callback_isolated_executor.git
$ cd callback_isolated_executor
$ source /opt/ros/humble/setup.bash
$ colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release
$ source install/setup.bash
```

Thread ConfiguratorのバイナリにCAP_SYS_NICEを付与する必要があるが
詳細はREADMEにて: https://github.com/tier4/callback_isolated_executor

CIE導入に必要なアプリケーションへの変更

- 1) ComposableNodeではない場合は、コード中のExecutorを置き換える (右)
- 2) ComposableNodeである場合は、launchファイル中のComponent ContainerをCIE 相当のものに置き換える (下)

```
#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;
}
```

```
<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>
```

Thread Configuratorの使い方 ①

Thread Configuratorノードを --prerun
オプションで立ち上げた後、ROS 2
アプリを立ち上げると、右のような
設定ファイルのテンプレートを生成する

```
callback_groups:
- id: /sample_node@Subscription(/parameter_events)@Service(/sample_node/get_parameters)@Serv:
  affinity: ~
  policy: SCHED_OTHER
  priority: 0

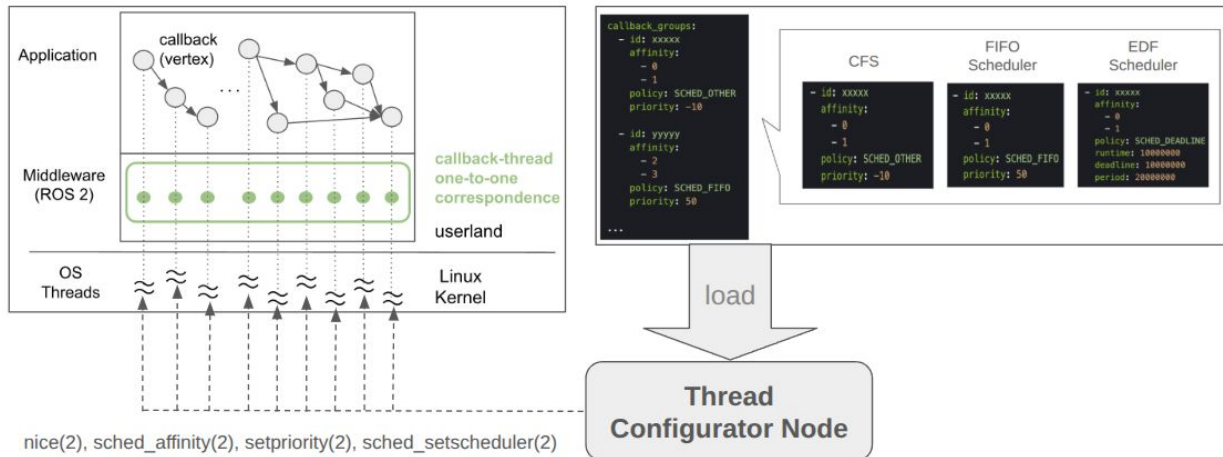
- id: /sample_node@Subscription(/topic_in)@Waitable
  affinity: ~
  policy: SCHED_OTHER
  priority: 0

- id: /sample_node@Timer(1333000000)
  affinity: ~
  policy: SCHED_OTHER
  priority: 0
```

```
autoware@logging:~/myrepo/callback_isolated_executor$ ros2 run ros2_thread_configurator thread_configurator_node --prerun
prerun mode
1715230539.402008 [1] thread_con: using network interface enp2s0f1 (udp/192.168.40.61) selected arbitrarily from: enp2s0f1, enp2s0f1
[INFO] [1715230967.918168913] [prerun_node]: Received CallbackGroupInfo: tid=546003 | /sample_node@Subscription(/parameter_events)@S
ervice(/sample_node/get_parameters)@Service(/sample_node/get_parameter_types)@Service(/sample_node/set_parameters)@Service(/sample_n
ode/set_parameters_atomically)@Service(/sample_node/describe_parameters)@Service(/sample_node/list_parameters)@Waitable@Waitable@Wai
table@Waitable
[INFO] [1715230967.918286899] [prerun_node]: Received CallbackGroupInfo: tid=546004 | /sample_node@Timer(3000000000)
[INFO] [1715230967.918325414] [prerun_node]: Received CallbackGroupInfo: tid=546005 | /sample_node@Timer(1333000000)
[INFO] [1715230967.918355465] [prerun_node]: Received CallbackGroupInfo: tid=546006 | /sample_node@Subscription(/topic_in)@Waitable
```

Thread Configuratorの使い方 ②

YAMLテンプレートに設定を記入し、以下のコマンドでthread configuratorノードを立ち上げた後、ROS 2アプリを動作させると、スケジューラ設定が反映されている



```
$ ros2 run cie_thread_configurator thread_configurator_node --config-file your_config.yaml
```

総括

06



総括

- CallbackIsolatedExecutor (CIE) の登場により、二重スケジューリング (Middleware & OS) をリアルタイムスケジューリングで考慮する必要がなくなり、DAG上のvertexであるコールバックに直接パラメータを設定可能になった
- CIEはAutowareに導入され、ティアフォーのロボタクシーにおいては、計測ベースで5倍のWorst-Case Response Time改善が達成されている
- CIEは、ティアフォーが開発するROS 2互換ゼロコピーミドルウェア Agnocastにも同梱されている

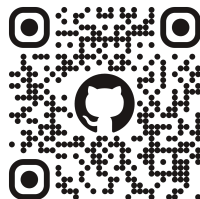
ROSCon 2025で発表予定



Paper [arXiv link](#)



Github
[tier4/callback_isolated_executor](#)



Github [tier4/agnocast](#)

謝辞

この成果は、国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO) の助成事業 (JPNP21027) の結果得られたものです。

Appendix

Overhead of New Executor

Appendix



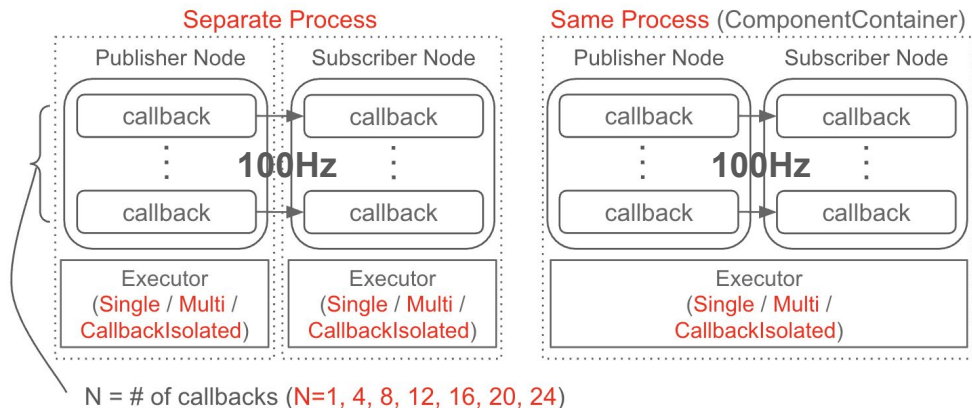
Evaluation Setup

In terms of theoretical real-time scheduling models, our Executor is a superior choice.

→ Overhead of **mapping each callback to a dedicated thread** must be discussed.

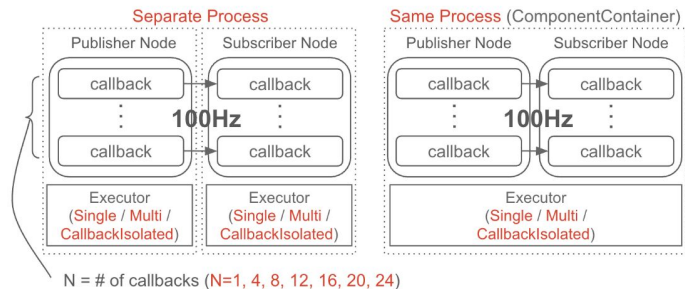
Changing **parameters**, following overhead factors are measured.

- **user-kernel switches**
- **context switches**
- **memory consumption**



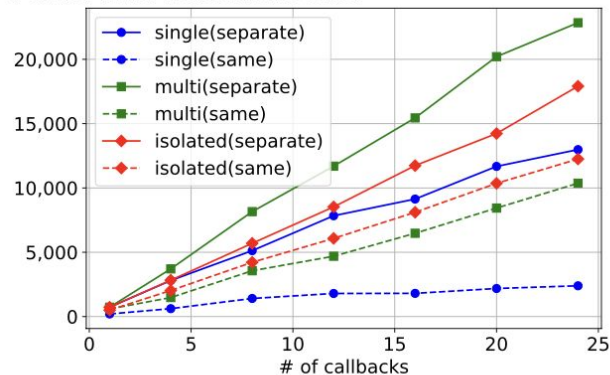
- # of threads = one per process (SingleThreadedExecutor)
- # of threads = min(16, # of callbacks) (MultiThreadedExecutor)
- # of threads = # of callbacks (CallbackIsolatedExecutor)

Evaluation Results



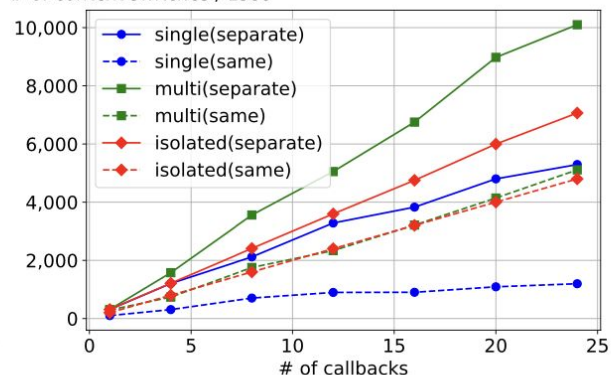
- # of threads = one per process (SingleThreadedExecutor)
- # of threads = $\min(16, \# \text{ of callbacks})$ (MultiThreadedExecutor)
- # of threads = # of callbacks (CallbackIsolatedExecutor)

of user-kernel mode switches / 1sec



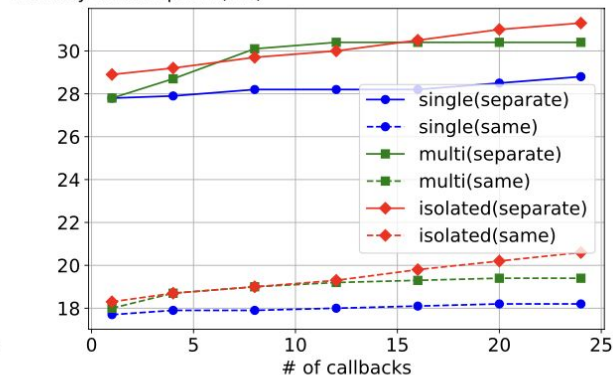
(a) # of user-kernel switches per sec.

of context switches / 1sec



(b) # of context switches per sec.

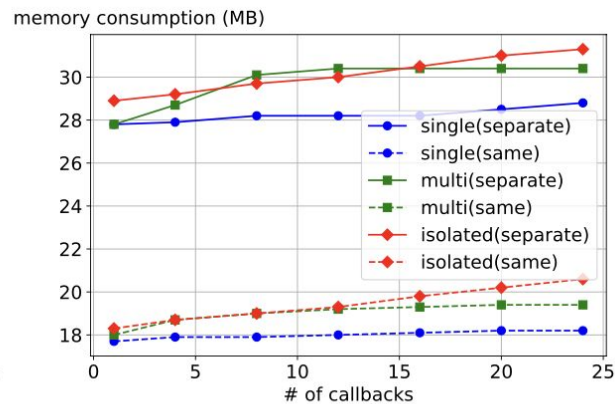
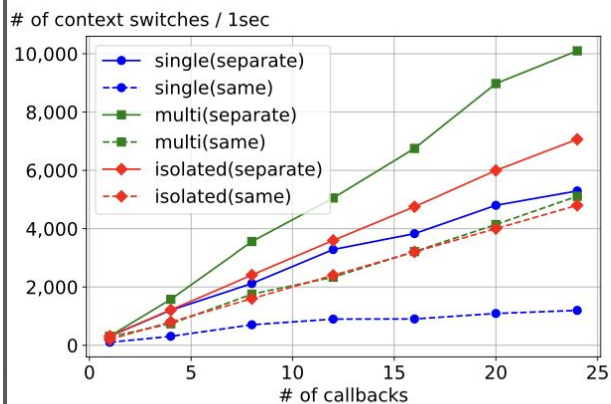
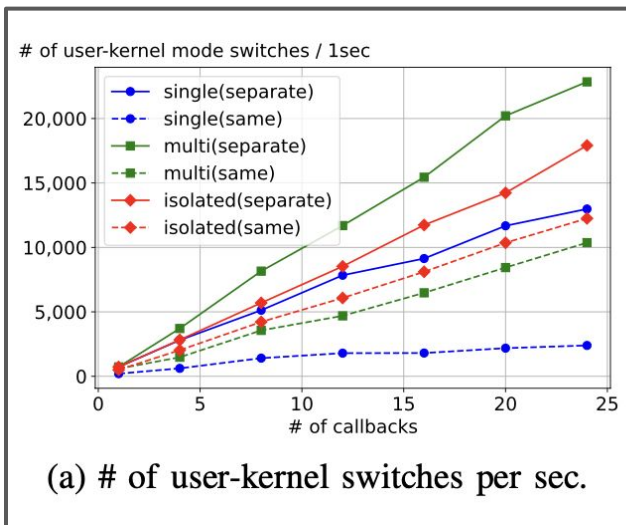
memory consumption (MB)



(c) Memory consumption

Evaluation Results (user-kernel switches)

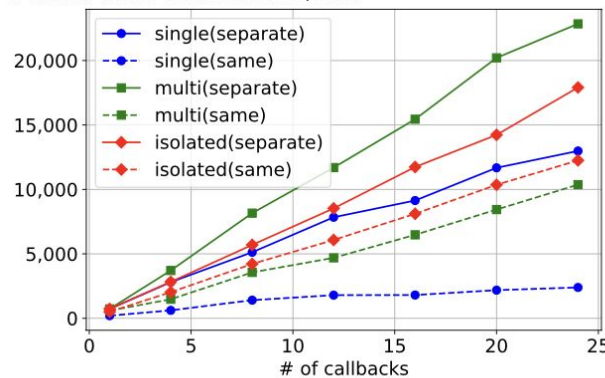
Compared to **MultiThreadedExecutor** (despite MultiThreadedExecutor being limited by hardware concurrency), **CallbackIsolatedExecutor** outperforms it when nodes in separate processes.



Evaluation Results (context switches)

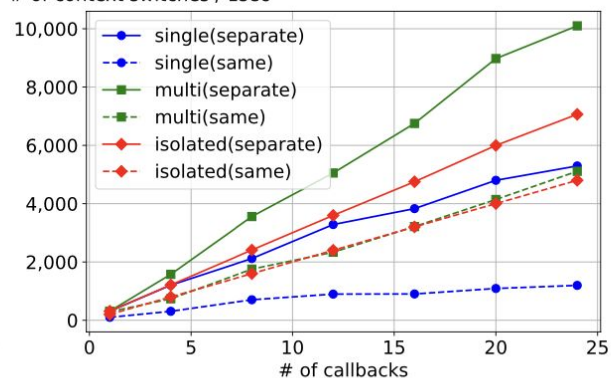
Compared to **MultiThreadedExecutor** (despite MultiThreadedExecutor being limited by hardware concurrency), **CallbackIsolatedExecutor** outperforms it.

of user-kernel mode switches / 1sec



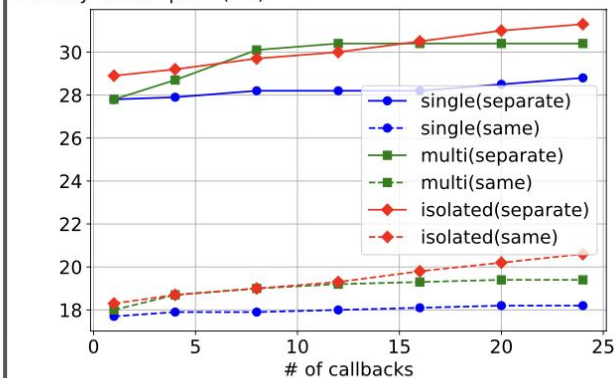
(a) # of user-kernel switches per sec.

of context switches / 1sec



(b) # of context switches per sec.

memory consumption (MB)

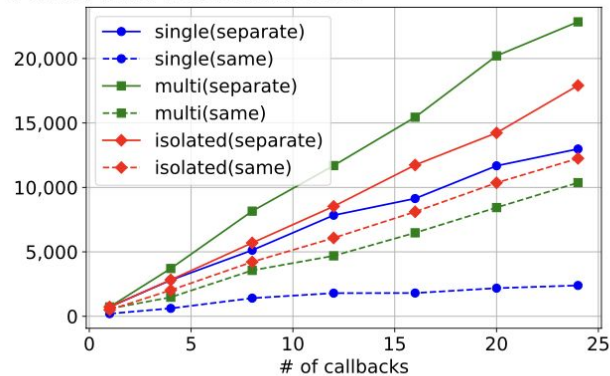


(c) Memory consumption

Evaluation Results (memory consumption)

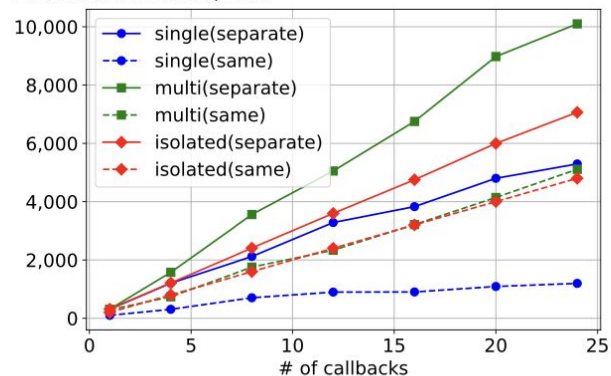
Memory consumption in **CallbackIsolatedExecutor** is slightly higher but negligible.

of user-kernel mode switches / 1sec



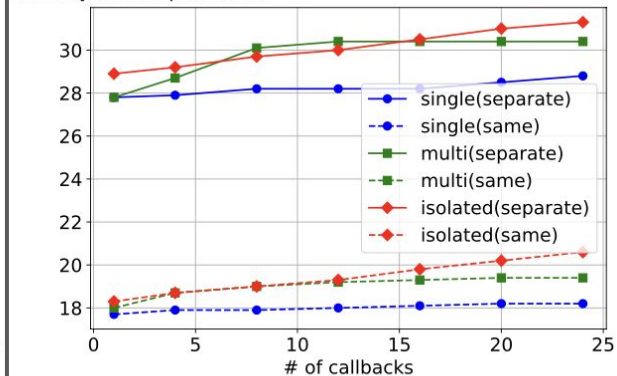
(a) # of user-kernel switches per sec.

of context switches / 1sec



(b) # of context switches per sec.

memory consumption (MB)



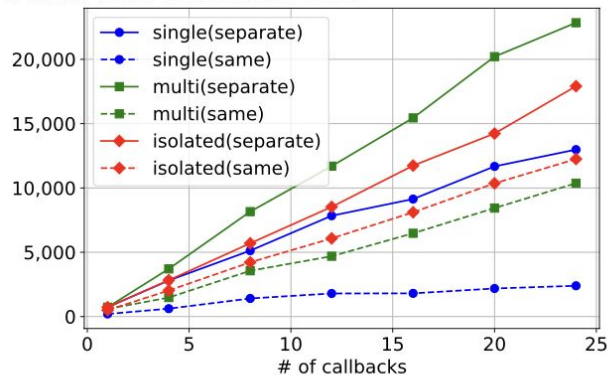
(c) Memory consumption

Conclusion in RTAS Brief Paper

With CallbackIsolatedExecutor, the real-time community no longer needs to account for the existence of the Executor in ROS 2 scheduling **in practical cases**.

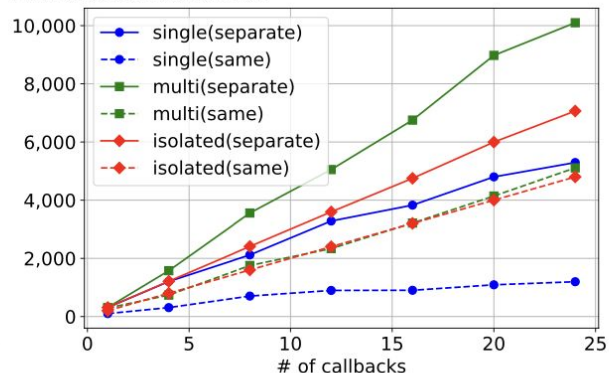
- Real-world ROS 2 systems rarely have dozens of callbacks per node
- In Autoware, each node typically has at most around 10 callbacks.

of user-kernel mode switches / 1sec



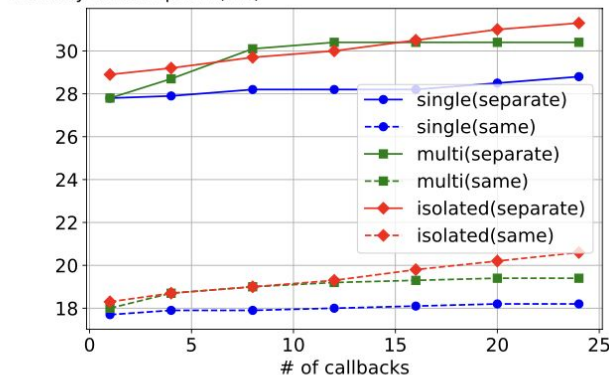
(a) # of user-kernel switches per sec.

of context switches / 1sec



(b) # of context switches per sec.

memory consumption (MB)



(c) Memory consumption