Apex.AI®

The vehicle OS company.

Executor based on wait-set and polling subscription

michael.poehnl@apex.ai

What ROS users are used to

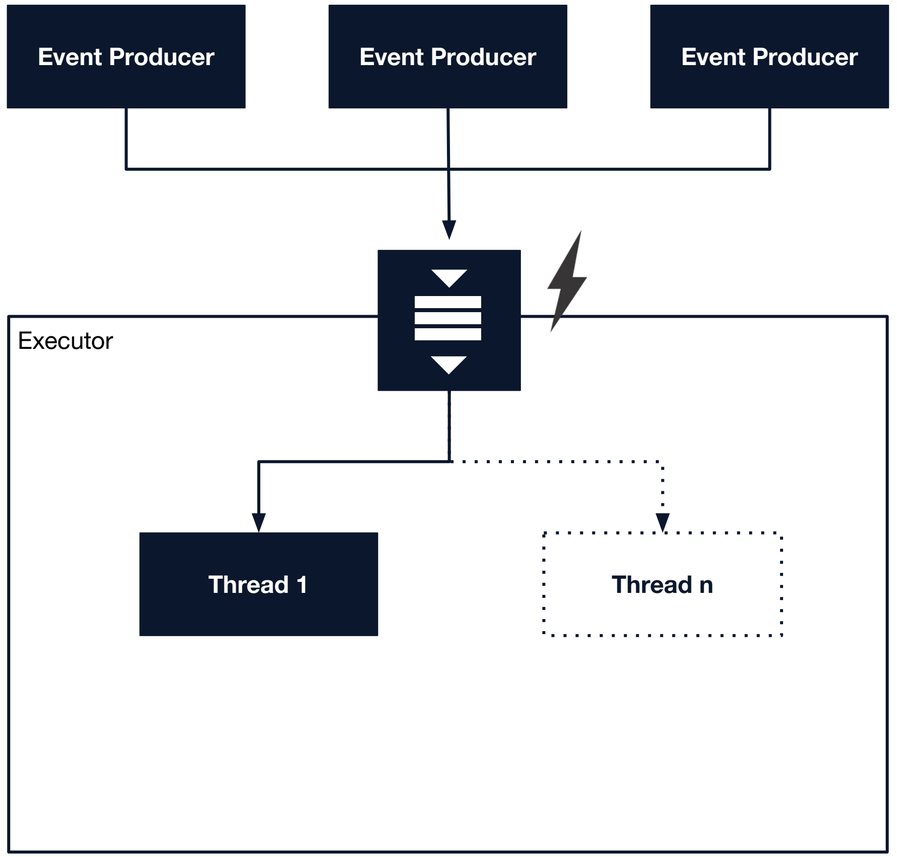
Default ROS 2 execution model

- Node
  - Publishers
  - Subscriptions
  - Clients
  - Services
  - Timers
  - …
- Executor reacts on events
  - Publisher sends a message
  - Client sends a request
  - Timer expires
  - ….
- Executor executes callbacks for the events
  - Subscription callbacks
  - Service callbacks
  - Timer callbacks
  - ...

How can such an executor
be implemented?

Active Object Pattern

- Executor has an event queue and an own thread of control for decoupling from producers
- E.g. a condition variable is used to do a non-busy wait on the queue
- Executor thread is woken up if a new event is pushed to the queue
- Events typically contain the data to be processed
- One or many threads are used to execute the tasks associated with the events
- That’s roughly how it was implemented in ROS 1



Perfect match when the events
are related to a specific task

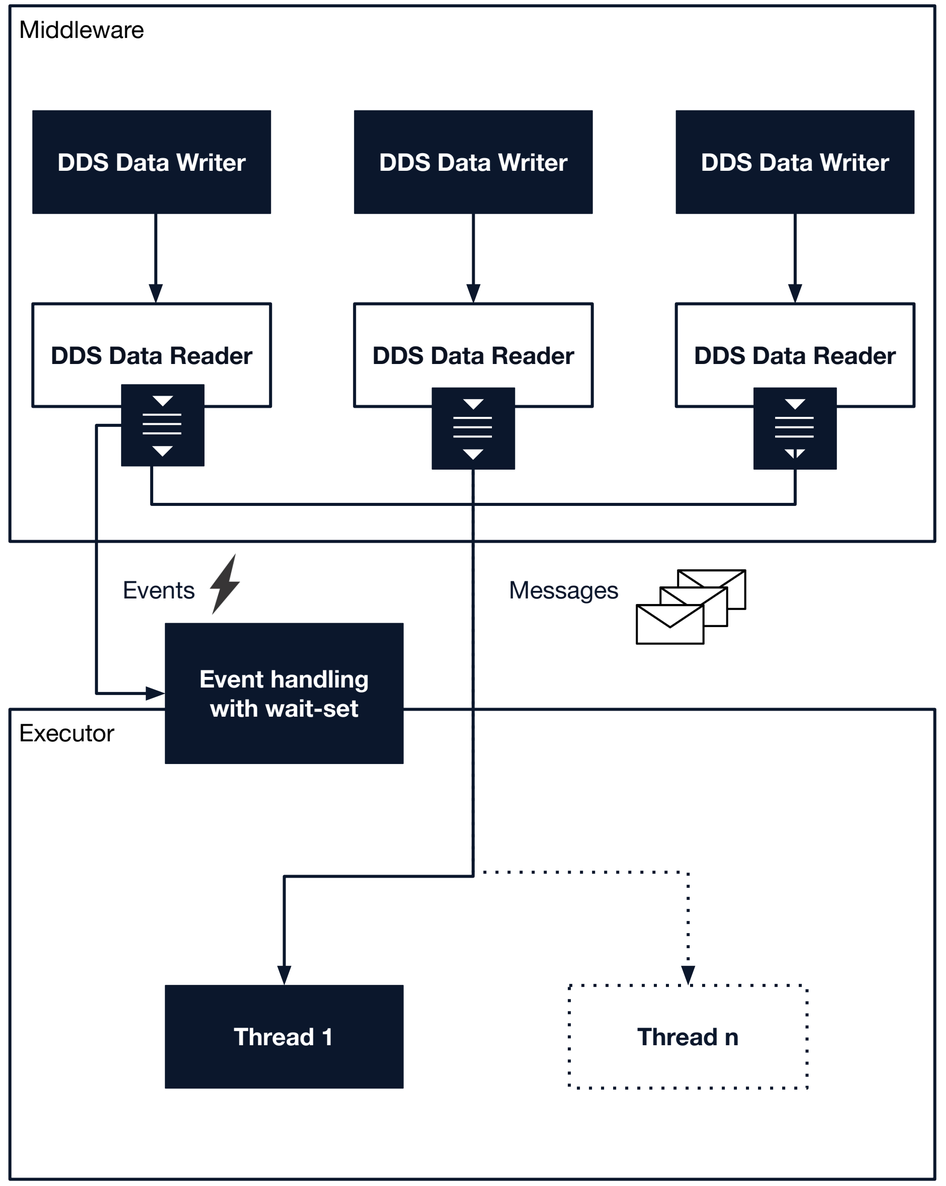
+ onPhoneButtonPressed()
+ onMusicButtonPressed()
+ onMapsButtonPressed()
+ onMessagesButtonPressed()
+ ...

Some things are different
with ROS 2, right?

DDS and the default ROS 2 executors

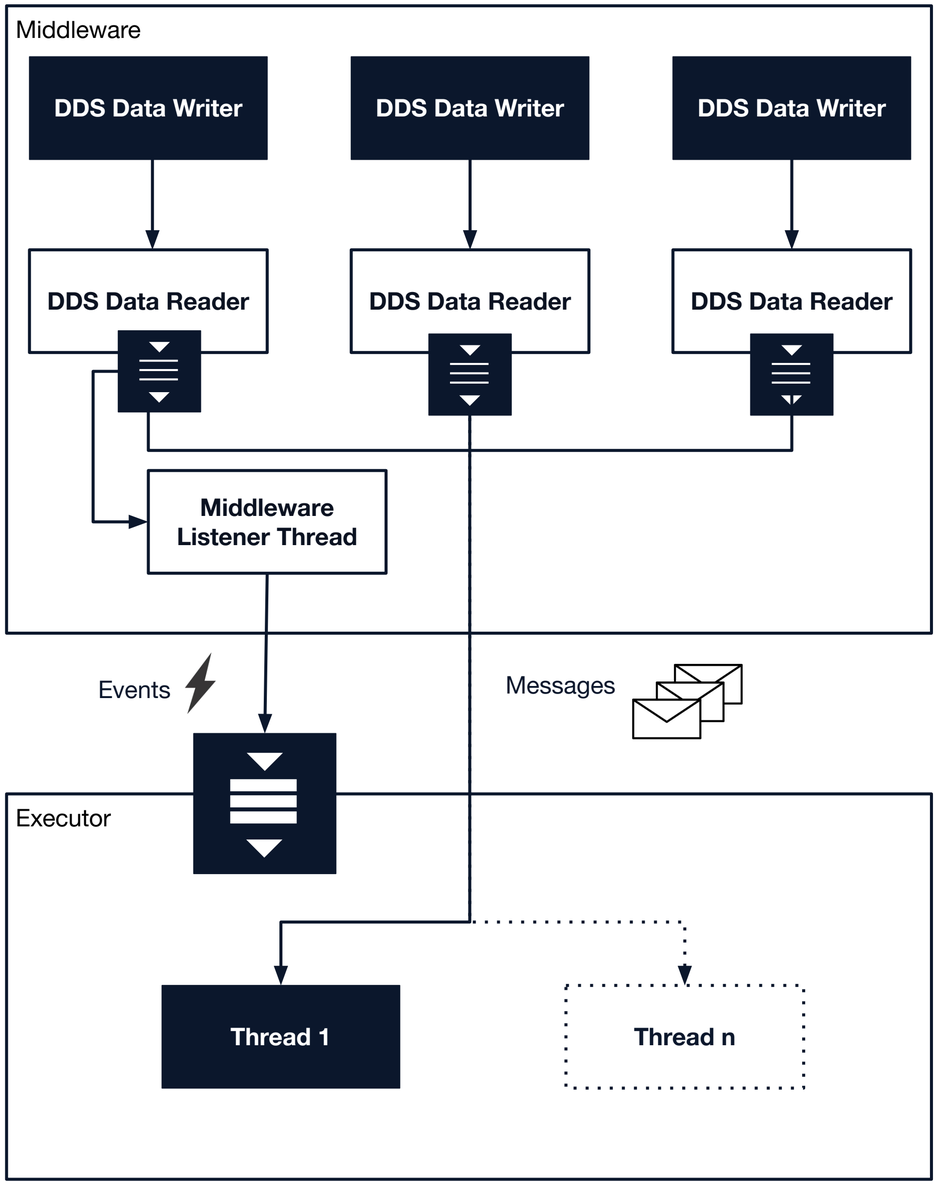
- DDS is used as middleware
  - Data readers already queue messages (in the reader cache)
  - Event-driven interaction via
    - Listener (Event callbacks are executed in a middleware thread)
    - Wait-set (User thread can wait for events that trigger the wait-set)

- Current ROS 2 default executors use a wait-set for DDS related events
  - Attach the event sources like subscriptions or services to a wait-set
  - Wait in an executor thread for the wait-set to get triggered
  - Execute callbacks for the entities that triggered the wait-set
  - By (DDS) design, handling of events and messages are separated



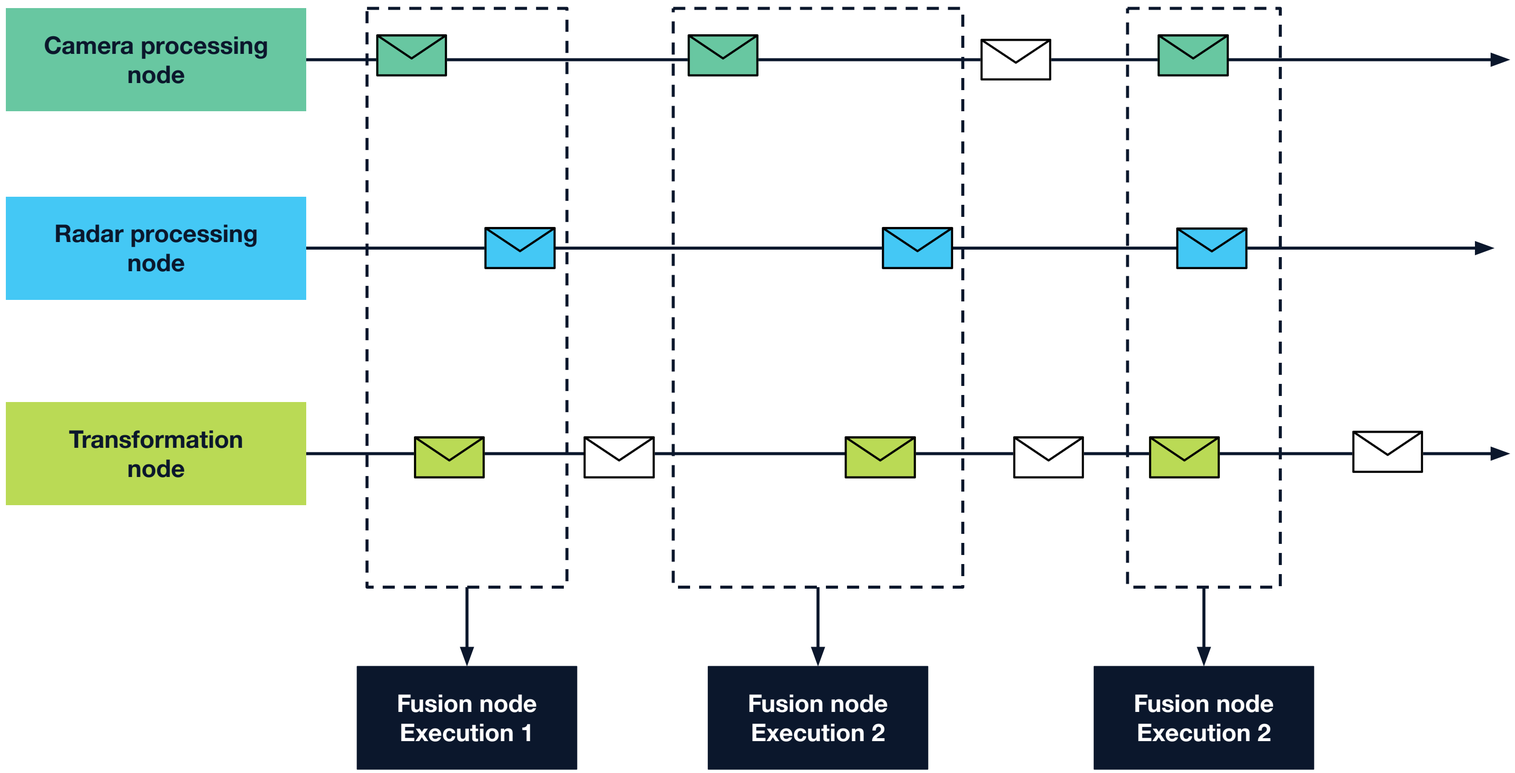
You can also have the ROS 1
event queue back

Roughly like this



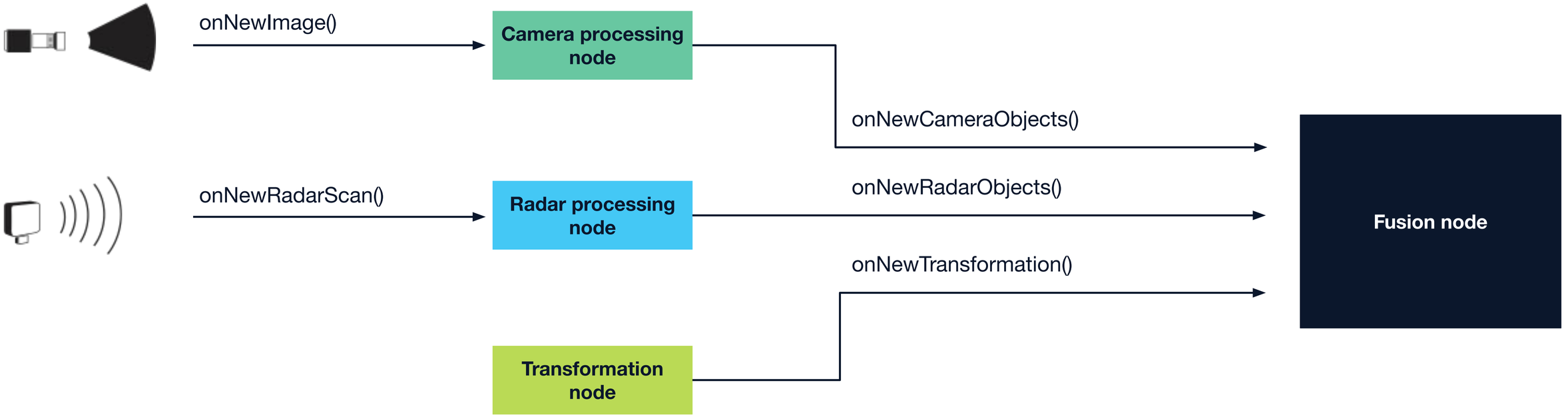
But my use case is another

A typical use case

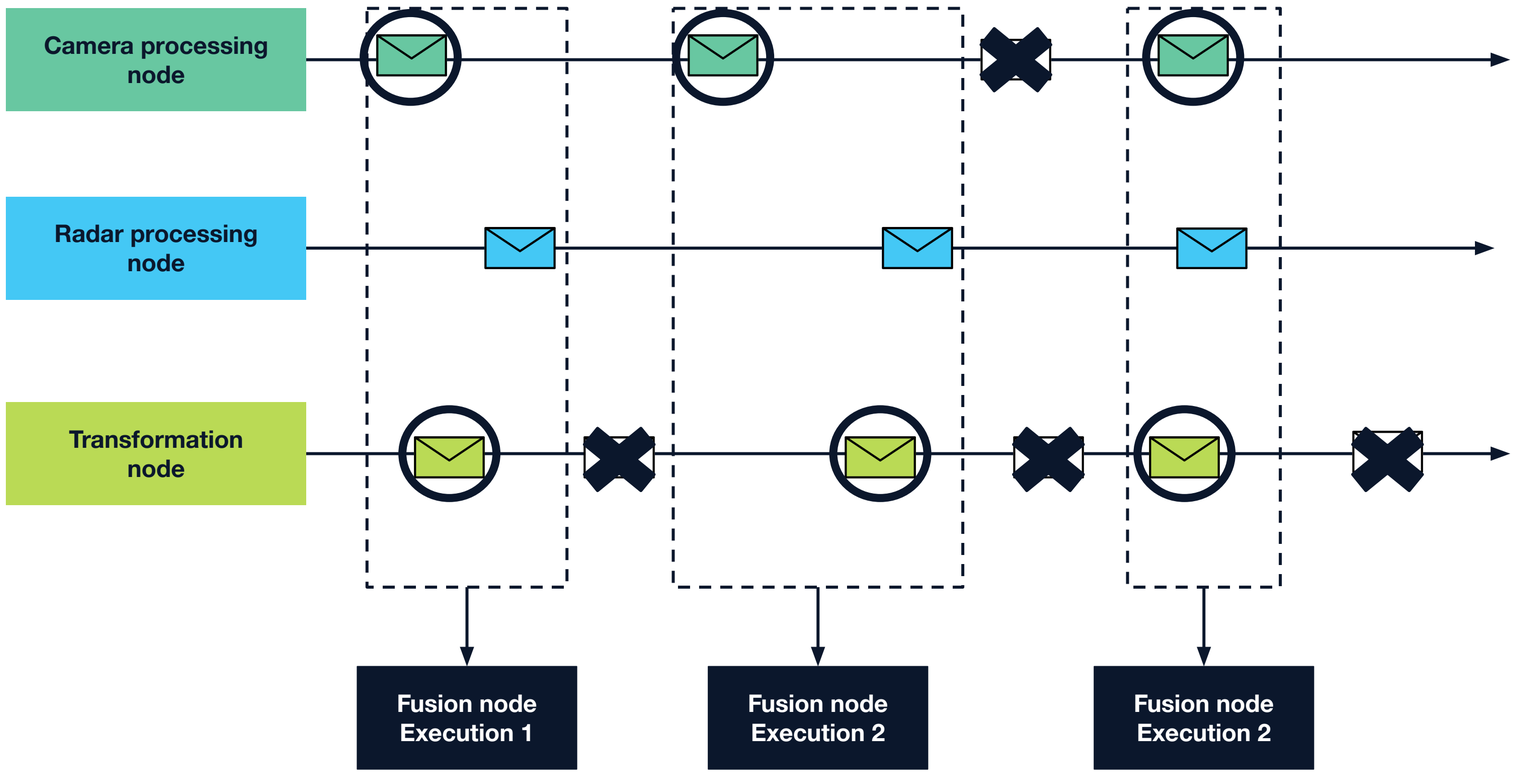


- A node has several subscriptions with different update frequencies (e.g. a fusion node)
- A node has a specific task (e.g. fuse the radar objects with the latest camera objects, use latest transformation for this)
- The node task shall be executed whenever a specific condition is met (e.g. new radar message is available)

Is this also a perfect match?



A typical use case



- Why should I handle callbacks for messages that are not needed for my task?
- Why should I take care of message caching when DDS can do this for me (History QoS)?
- Why should I bother with these unnecessary context switches when I could avoid them?
- Wouldn’t it be good if the node were called to do its task whenever the execution condition is met?

Executor based on wait-set and polling subscription

- Node base class
  - *void execute()* - called by the executor when the execution condition is met
  - *subscription_list get_triggering_subscriptions()* - get the subscriptions relevant for triggering the node execution

- Polling subscription
  - Sounds worse than it is - We only “poll” when the executor tells us it makes sense (execution condition is met)
  - Allows to *read()* and *take()* messages from the DDS reader cache
  - Allows to drop uninteresting messages already in the middleware (e.g. set History QoS=1 if latest is greatest)

- Efficient use of the DDS wait-set
  - Only attach to the wait-set the events that are relevant for the node execution

- Executor based on these building blocks (and some more …)
  - Calls the execute() method of a node when one of the triggering events occurs
  - Optionally, a callable provided by the user is used as an execution condition that is evaluated on triggering events

- Can you do this with ROS 2?
  - Not straightforward, but there is a starting point as with the Foxy release an rclcpp::WaitSet and a *take()* method for subscriptions were introduced (https://github.com/ros2/rclcpp/pull/1047)

Fusion example

- Only the radar subscription is a triggering one
- Executor calls execute() of the fusion node whenever a new radar message is received
- In execute_impl() the new messages from radar, camera and transformation are taken and processed

```
class my_node : public apex_node_base
{
public:
  ...

private:
  void execute_impl() override
  {
    auto radarMessages = m_radarSubscription->take();
    auto cameraMessages = m_cameraSubscription->take();
    auto transformationMessages = m_transformationSubscription->take();
    // do the fusion and publish the result
  }

  subscription_list get_triggering_subscriptions_impl() const override
  {
    return {m_radarSubscription};
  }

  rclcpp::PollingSubscription<Radar>::SharedPtr m_radarSubscription{...};
  rclcpp::PollingSubscription<Image>::SharedPtr m_cameraSubscription{...QoS(KeepLast(1))...};
  rclcpp::PollingSubscription<Transform>::SharedPtr m_transformSubscription{...};
};
```

```
int main()
{
    rclcpp::init(0, nullptr);

    // create a node
    auto node = std::make_shared<my_node>("my_node");
    // create an executor
    auto exec = executor_factory::create();
    // add node to executor
    exec->add(node);
    // run the executor
    exec->run();

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

Planner example

- The planner shall be executed every 100ms, no triggering subscriptions
- Executor calls execute() of the planner node whenever the cyclic timer expires
- In execute_impl() all new messages are taken and processed

```
class my_node : public apex_node_base
{
public:
  ...

private:
  void execute_impl() override
  {
    auto messages1 = m_sub1->take();
    auto messages2 = m_sub2->take();
    auto messages3 = m_sub3->take();
    // do the processing and publish the results
  }

  rclcpp::PollingSubscription<X>::SharedPtr m_sub1{...};
  rclcpp::PollingSubscription<Y>::SharedPtr m_sub2{...};
  rclcpp::PollingSubscription<Z>::SharedPtr m_sub3{...};
};
```

```
int main()
{
    rclcpp::init(0, nullptr);

    // create a node
    auto node = std::make_shared<my_node>("my_node");
    // create an executor
    auto exec = executor_factory::create();
    // execution of the node will happen every 100ms
    exec->add(node, 100ms);
    // run the executor
    exec->run();

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

Execution condition example

- The fusion node shall be executed if there is at least one message for radar and camera
- Executor calls execute() only if the provided execution condition returns true
- Execution condition is evaluated whenever the wait-set gets triggered (here on new radar and camera messages)

```
class my_node : public apex_node_base
{
public:
  bool ready()
  {
    auto radarMessages = m_radarSubscription->read();
    auto cameraMessages = m_cameraSubscription->read();
    return !radarMessages.empty() && !cameraMessages.empty();
  }

private:
  void execute_impl() override
  {
    auto radarMessages = m_radarSubscription->take();
    auto cameraMessages = m_cameraSubscription->take();
    auto transformationMessages = m_transformationSubscription->take();
    // do the fusion and publish the result
  }

  subscription_list get_triggering_subscriptions_impl() const override
  {
    return {m_radarSubscription, m_cameraSubscription};
  }
};
```

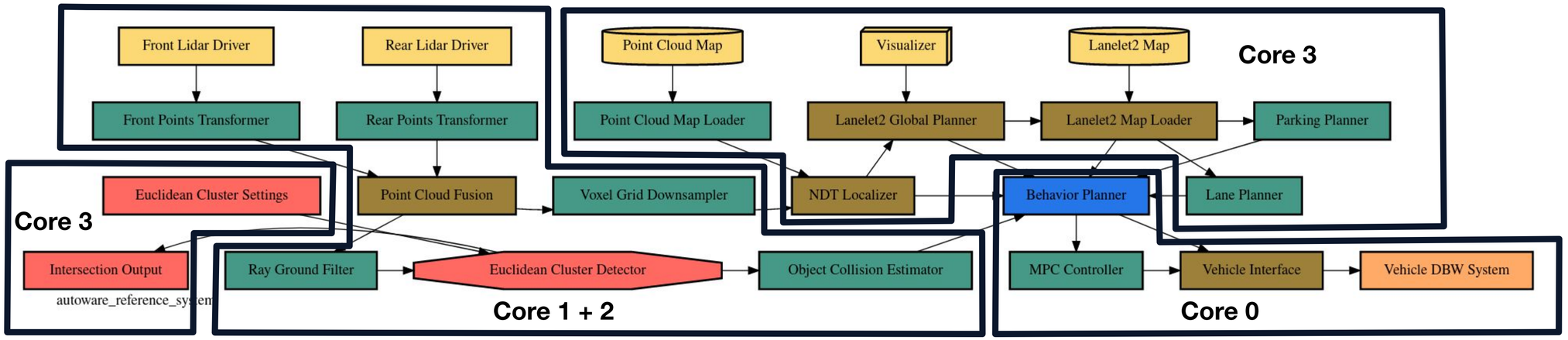
```
int main()
{
    rclcpp::init(0, nullptr);

    // create a node
    auto node = std::make_shared<my_node>("my_node");
    // create an executor
    auto exec = executor_factory::create();
    // The execution of the node will happen only if the condition
    // is true, meaning node->ready() returns true
    exec->add(node, [node] {return node->ready();});
    // run the executor
    exec->run();

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

Results for the reference system

- Reference system v0.1.1, Raspberry Pi with Ubuntu 20.04, using all 4 cores
  - No overload or dropped messages when using the ROS 2 multi-threaded executor
- Focusing on comparison of multi-threaded executor with Apex.OS executor
- Measurements were done with Apex.OS and Apex.Middleware
  - I.e. the multi-threaded executor runs in Apex.OS and not in ROS 2 Galactic
  - rmw_apex_middleware was also used for the multi-threaded executor
- Assignment of nodes to Apex.OS executors and core affinity for executors to best meet the target KPIs

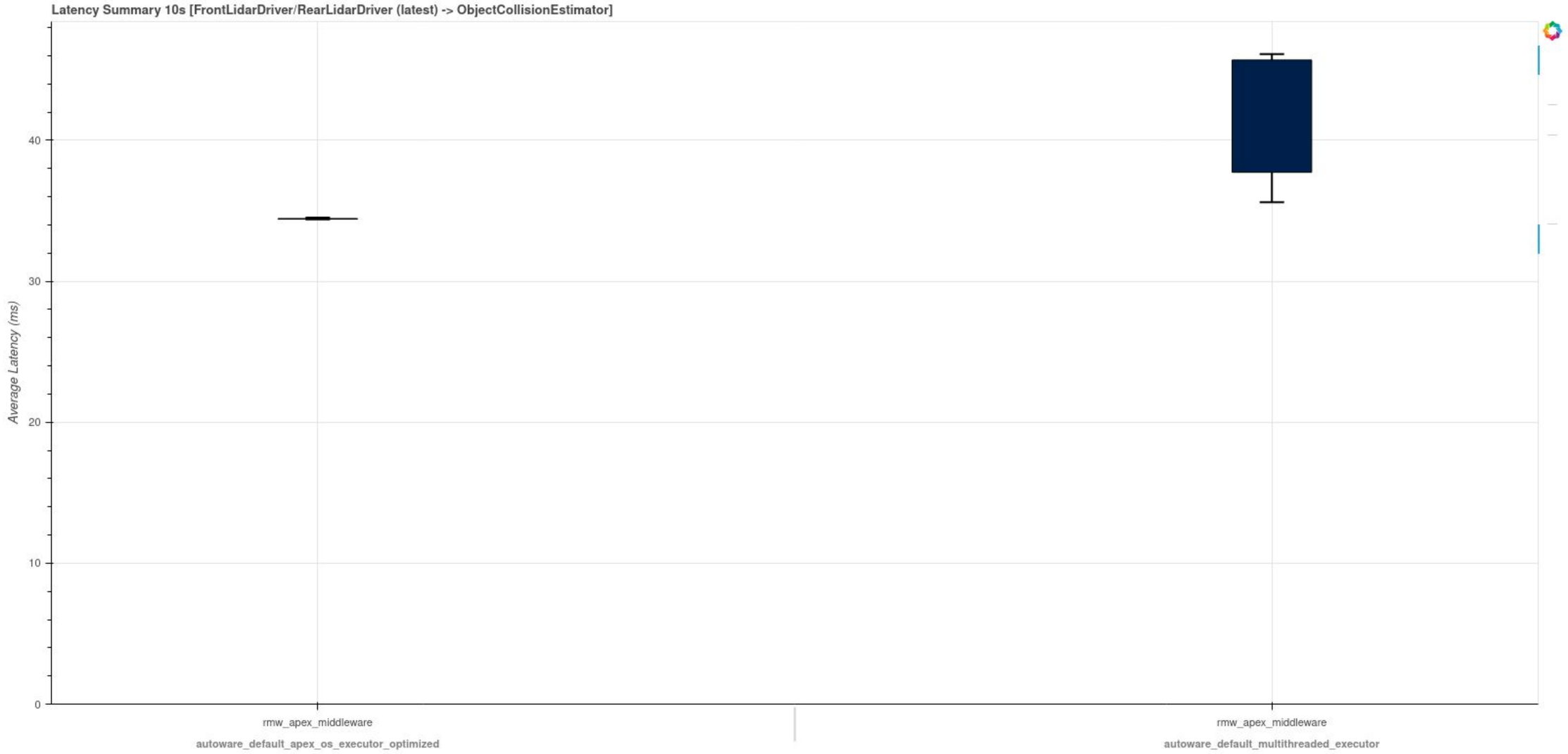


Results for the reference system

Latency Summary Table 10s [FrontLidarDriver/RearLidarDriver (latest) -> ObjectCollisionEstimator]

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_ap	rmw_apex_middlewa	latency	34.4016	34.4322	34.5152	34.45023780000000	34.4141622	0.0180378
1	autoware_default_m	rmw_apex_middlewa	latency	35.6035	41.7147	46.1216	45.69625	37.73315	3.98155

with default number crunching

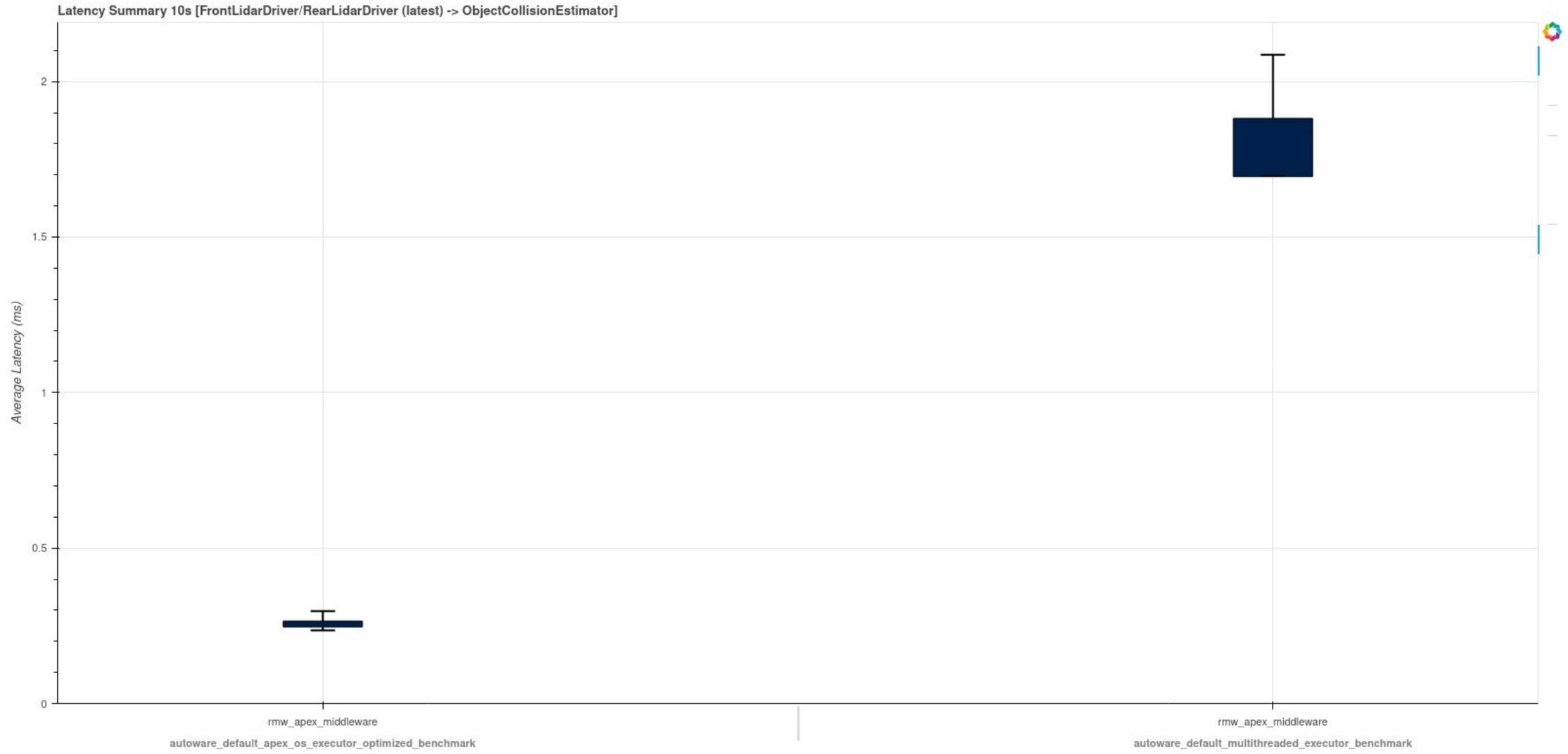


Results for the reference system

Latency Summary Table 10s [FrontLidarDriver/RearLidarDriver (latest) -> ObjectCollisionEstimator]

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_ap	rmw_apex_middlewa	latency	0.235445	0.25557	0.297055	0.26494148	0.246198520000000	0.00937148
1	autoware_default_m	rmw_apex_middlewa	latency	1.696	1.78783	2.08607	1.8814388	1.6942212	0.0936088

without number crunching



Results for the reference system

Behavior Planner Jitter Summary Table 10s

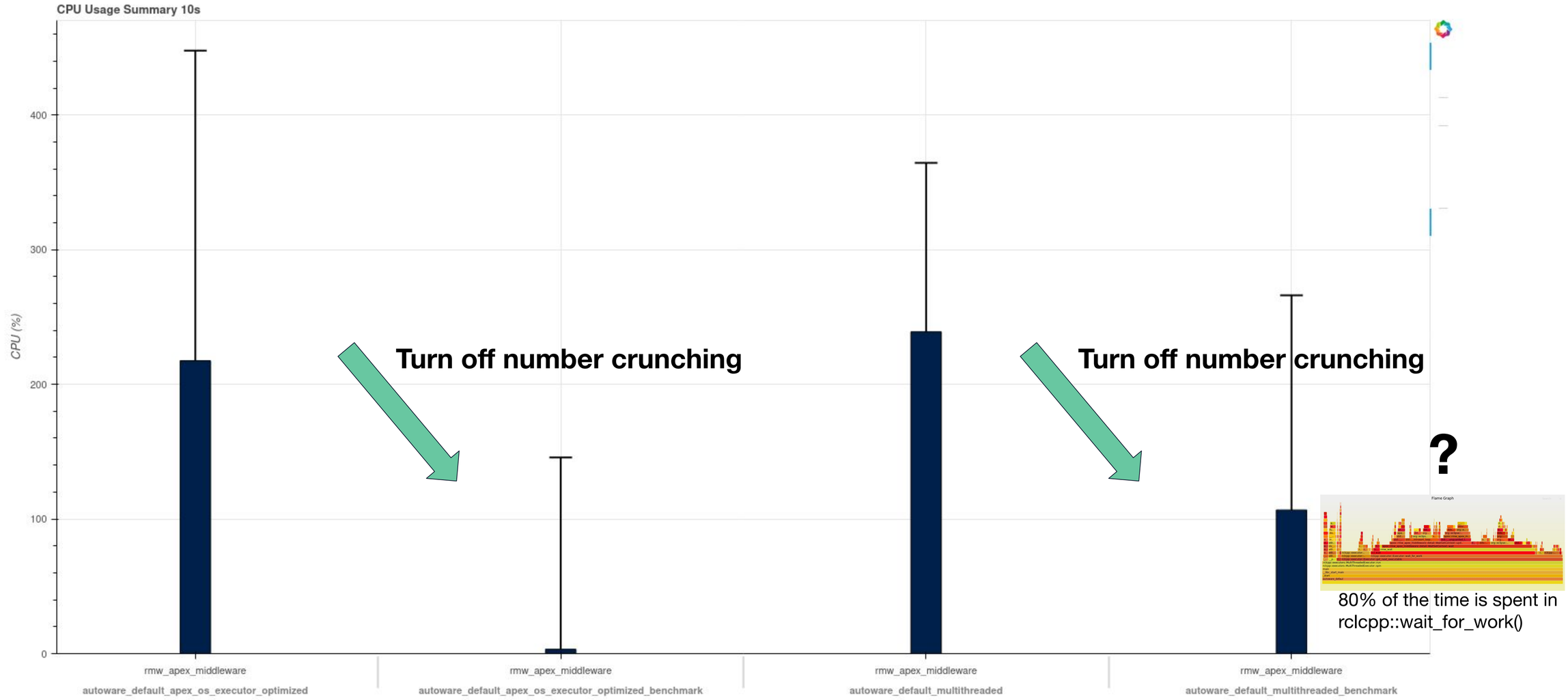
#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_ap	rmw_apex_middlewa	period	99.9829	100	100.017	100.00284866	99.99715134	0.00284866
1	autoware_default_m	rmw_apex_middlewa	period	99.8355	99.9984	100.181	100.0733053	99.9234947	0.0749053

with default number crunching

Results for the reference system

CPU Usage Statistics 10s

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_ap	rmw_apex_middlewa	cpu	0	217.5268676277850	447.8	275.9	181.4	80.04410660125261
1	autoware_default_ap	rmw_apex_middlewa	cpu	0	3.237541163556531	145.7	0	0	16.82830151766534
2	autoware_default_mu	rmw_apex_middlewa	cpu	0	238.9803716608594	364.5	265.7	175	66.59557197970123
3	autoware_default_mu	rmw_apex_middlewa	cpu	0	106.6778285714285	266	89.2	87.2	37.73353960732825

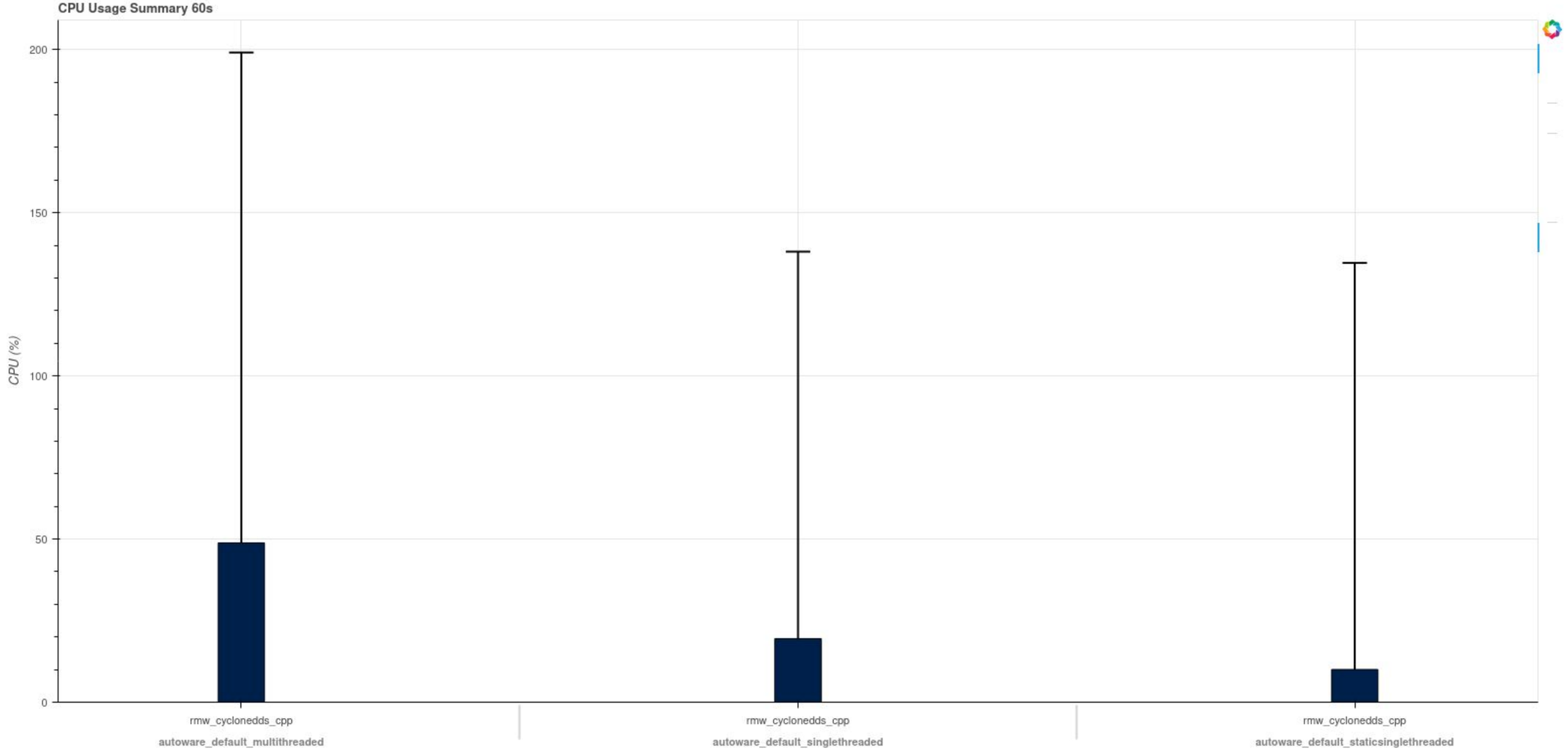


Results for the reference system

CPU Usage Statistics 60s

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_m	rmw_cyclonedds_cp	cpu	0	48.82882021093942	199.1	68.4	0	41.89926966822244
1	autoware_default_si	rmw_cyclonedds_cp	cpu	0	19.48428085519922	138.1	66.9	0	31.16869512840751
2	autoware_default_st	rmw_cyclonedds_cp	cpu	0	10.02866488975042	134.6	0	0	24.52763198350506

ROS 2 Galactic without number crunching



Apex.AI®

The vehicle OS company.

Thank you!

michael.poehnl@apex.ai