Reconciling ROS 2 with Classical Real-Time Scheduling of Periodic Tasks

Harun Teper[1], Oren Bell[2], Mario Günzel[1], Chris Gill[2], Jian-Jia Chen[1,3]

[1]TU Dortmund University, Germany
[2]Washington University at St. Louis, USA
[3]Lamarr Institute, Germany

May 8, 2025

Partially supported by the German Federal Ministry of Education and Research (BMBF) and by the European Research Council (ERC).

Introduction - Robot Operating System 2

Motivation

- ROS 2 as middleware for robotics systems
- Enables creation of modular systems
- Features real-time capabilities

Applications

- Autonomous vehicles
- Industrial robotics
- Safety-critical systems

Introduction: Scheduling Comparison

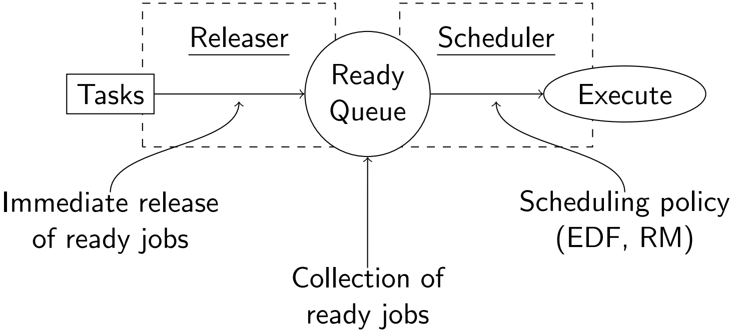
Classical Real-Time Scheduling	Robot Operating System 2 (ROS 2)
Scheduler	**Executor**
Established theory	Limited theory
Deadline-driven	Best effort
Fixed priority and dynamic priority	Fixed priority
Periodic, sporadic tasks	Sporadic tasks
Preemptive and non-preemptive	Non-preemptive

Introduction

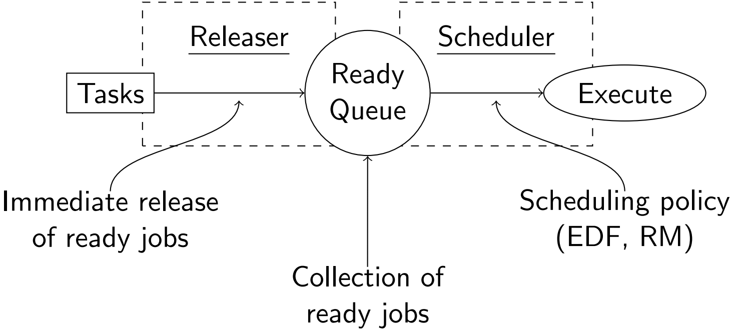
Contributions

- Examine incompatibilities between ROS 2 and classical scheduling theory
- Introduce modifications to the executor to enable compatibility
- Evaluate modified executor to determine its performance

Classical Real-Time Scheduling



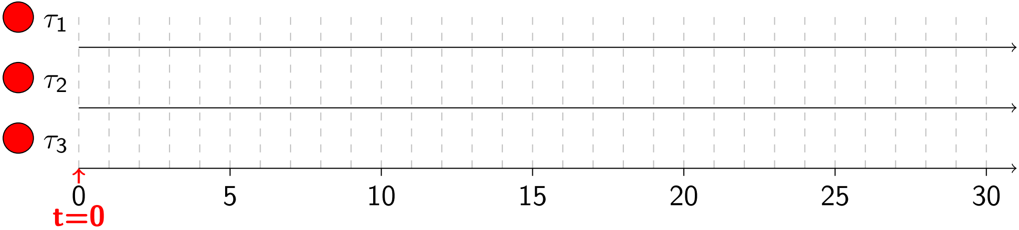
Classical Real-Time Scheduling



→ Well-established analytical frameworks exist

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

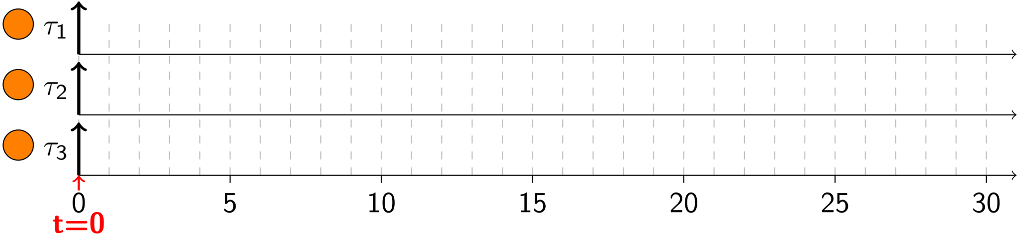


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

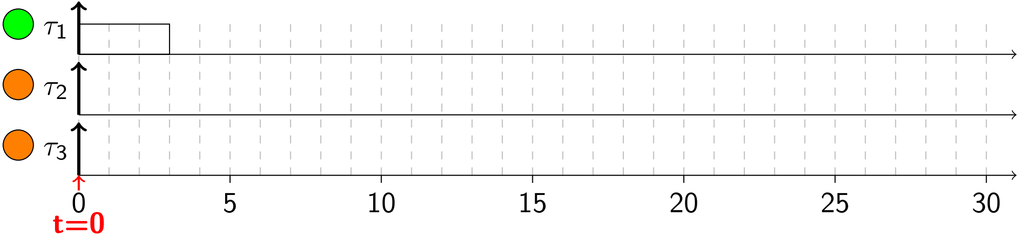


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

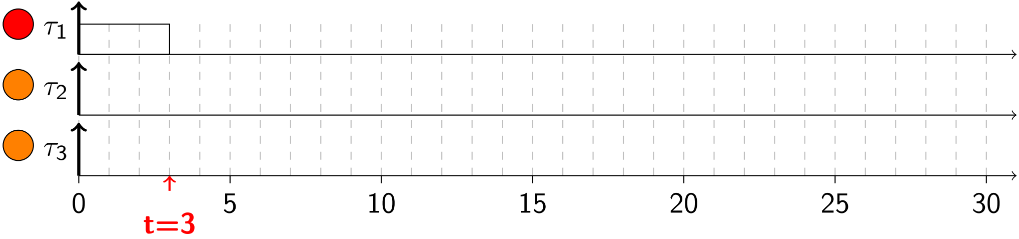


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

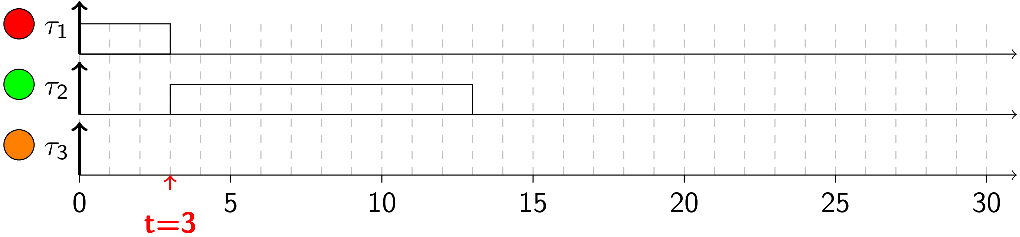


🟥 Idle 🟧 Ready 🟩 Executing

Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

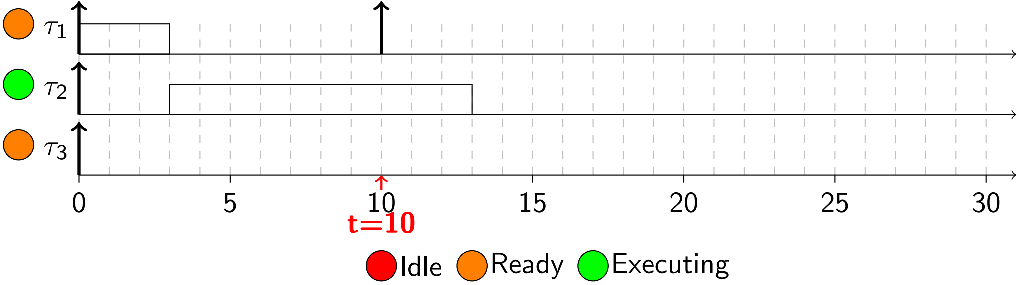


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

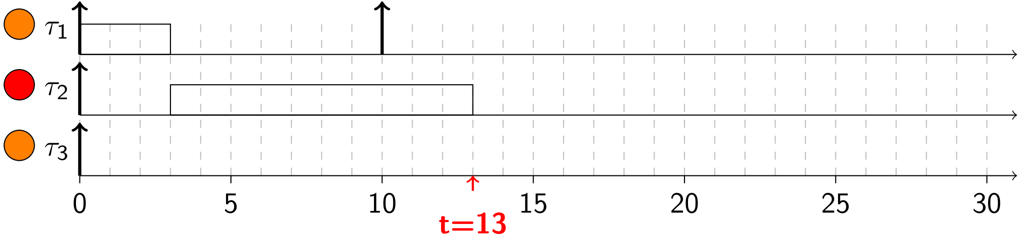
Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling



Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

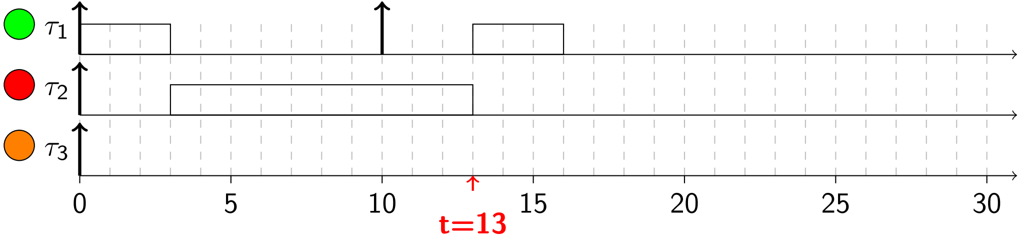


🟥 Idle 🟧 Ready 🟩 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

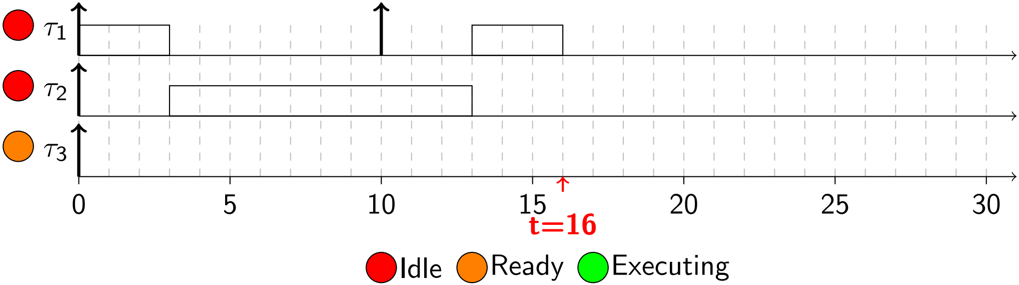


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

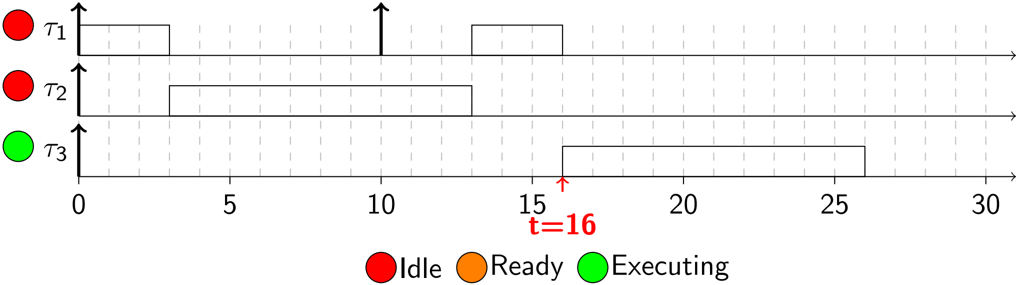
Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling



Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

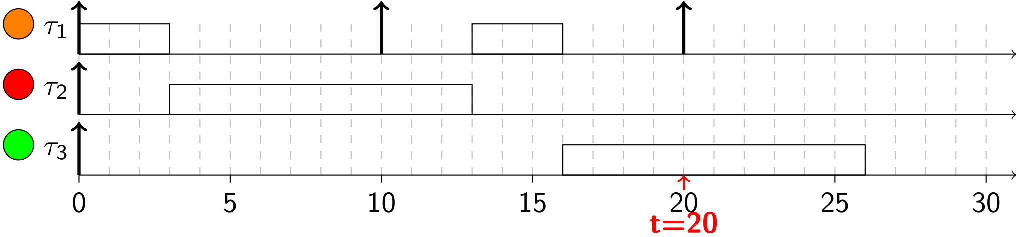
Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling



Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

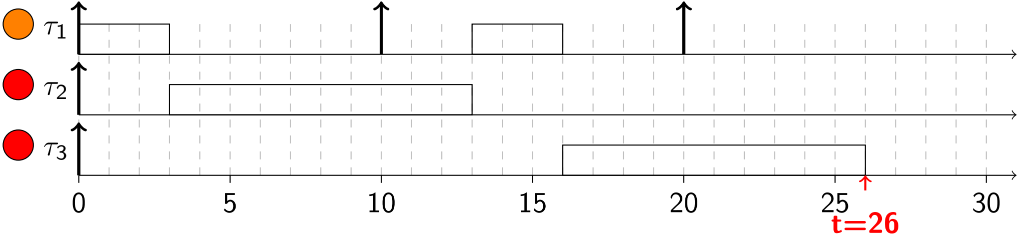


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

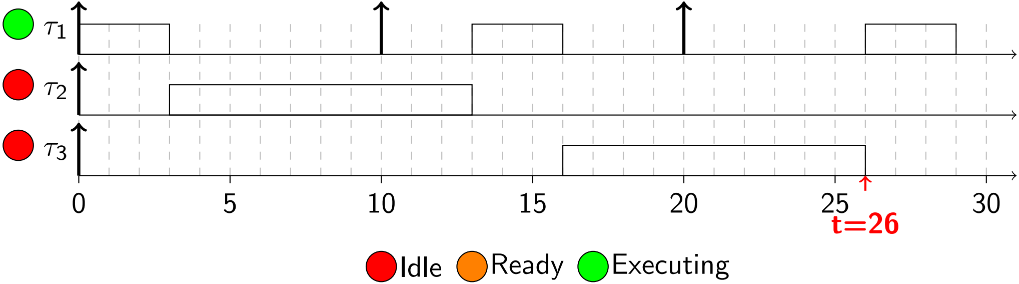


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

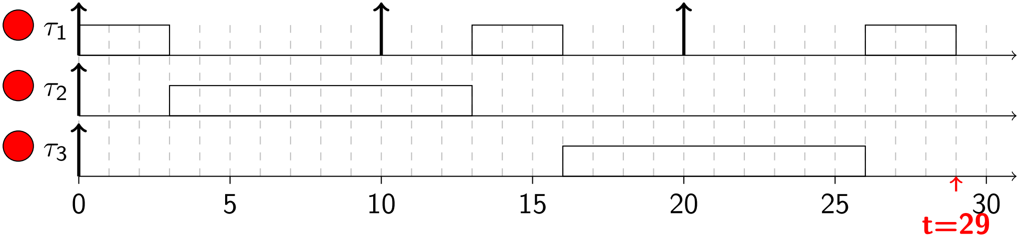
Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling



Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling

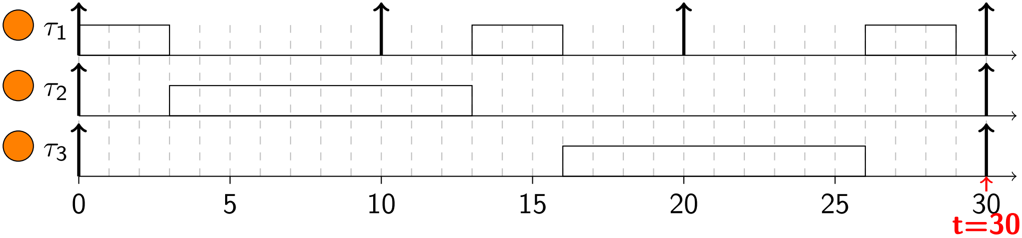


🔴 Idle 🟠 Ready 🟢 Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Classical Real-Time Scheduling

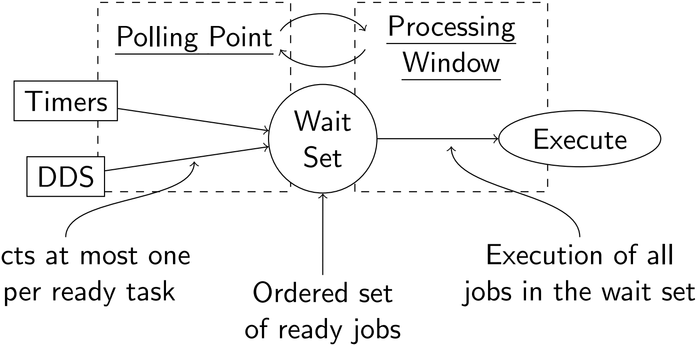
Setting: Fixed-priority (rate-monotonic) non-preemptive scheduling



🔴Idle 🟠Ready 🟢Executing

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

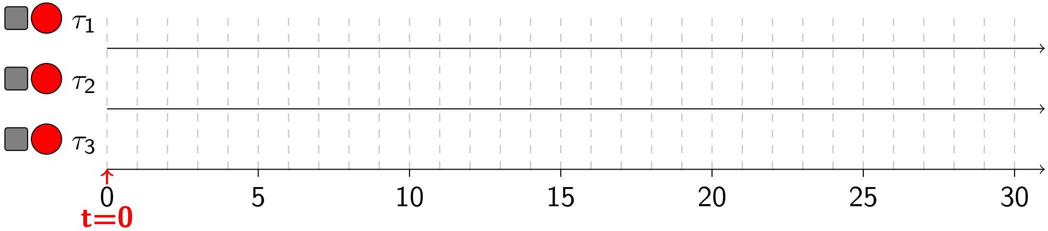
Background: ROS 2 Default Executor



Polling Point
Processing Window
Timers
DDS
Wait Set
Execute
Collects at most one job per ready task
Ordered set of ready jobs
Execution of all jobs in the wait set

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



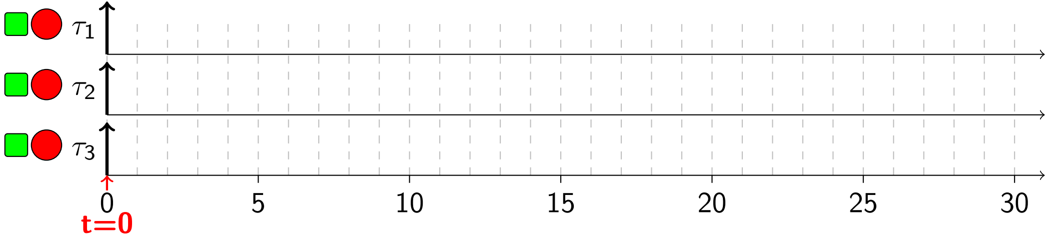
⬛ Not activated 🟩 Activated → Release ⇢ Dropped release

🔴 Not in wait set 🟠 In wait set 🟢 Executing - - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



■ Not activated 🟩 Activated

🔴 Not in wait set 🟠 In wait set 🟢 Executing

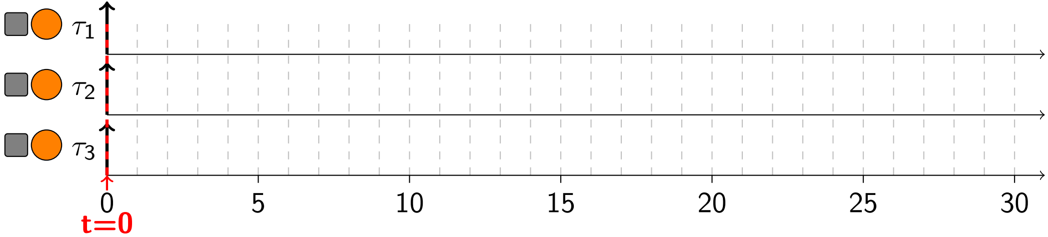
→ Release ⇢ Dropped release

-- Polling Point

Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



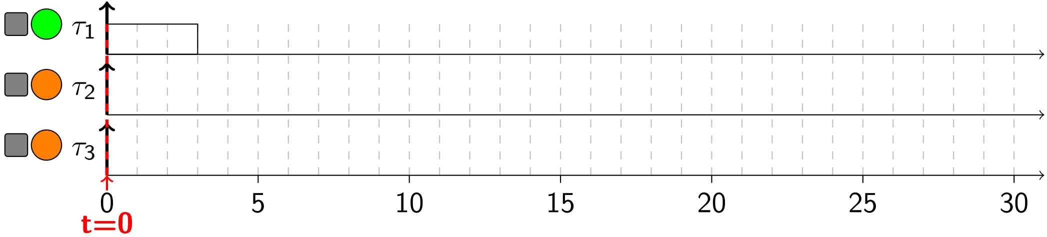
■ Not activated 🟩 Activated → Release ⋯→ Dropped release

🔴 Not in wait set 🟠 In wait set 🟢 Executing - - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



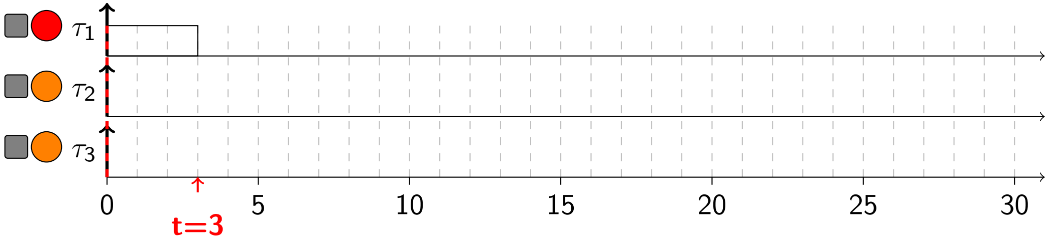
⬛ Not activated 🟩 Activated ⟶ Release ⇢ Dropped release

🔴 Not in wait set 🟠 In wait set 🟢 Executing - - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



◼ Not activated 🟩 Activated

🔴 Not in wait set 🟠 In wait set 🟢 Executing

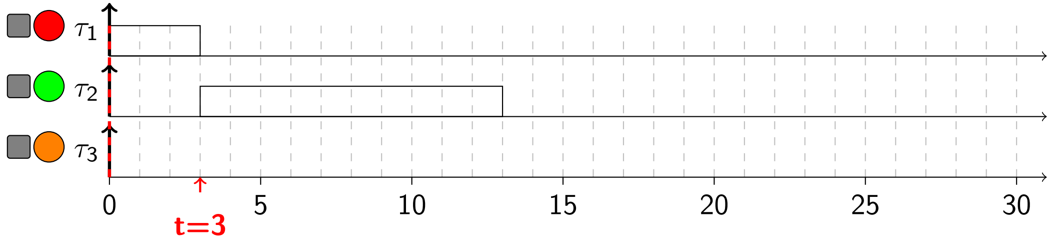
⟶ Release ⇢ Dropped release

- - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)

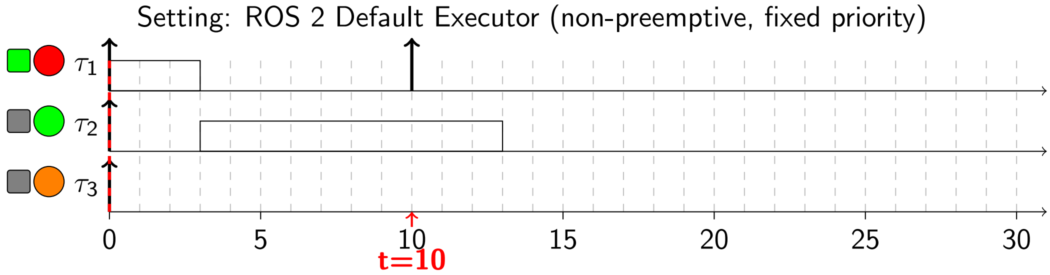


■ Not activated 🟩 Activated → Release ⇢ Dropped release

🔴 Not in wait set 🟠 In wait set 🟢 Executing - - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor



⬛ Not activated 🟩 Activated

🔴 Not in wait set 🟠 In wait set 🟢 Executing

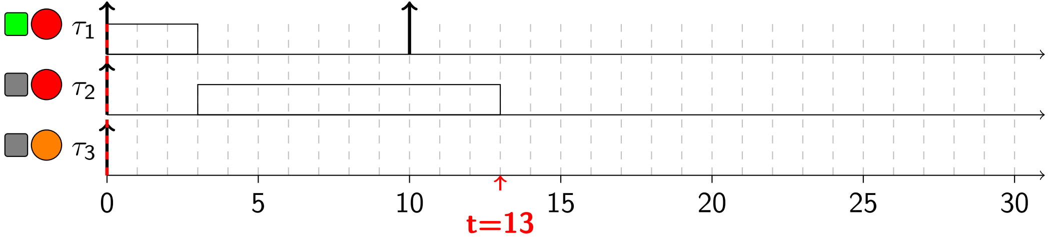
⟶ Release ⋯→ Dropped release

- - Polling Point

Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



■ Not activated 🟩 Activated

🔴 Not in wait set 🟠 In wait set 🟢 Executing

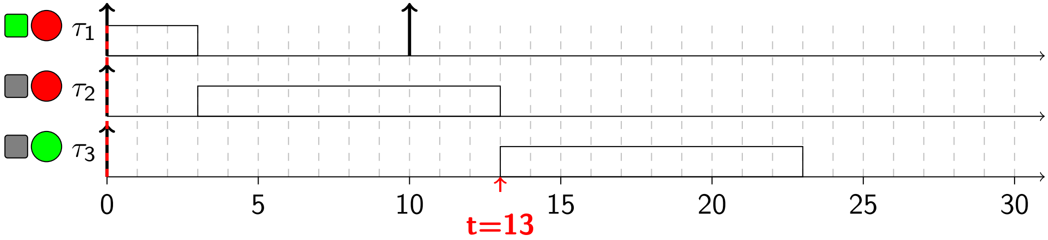
⟶ Release ⇢ Dropped release

- - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



▨ Not activated 🟩 Activated

→ Release ⇢ Dropped release

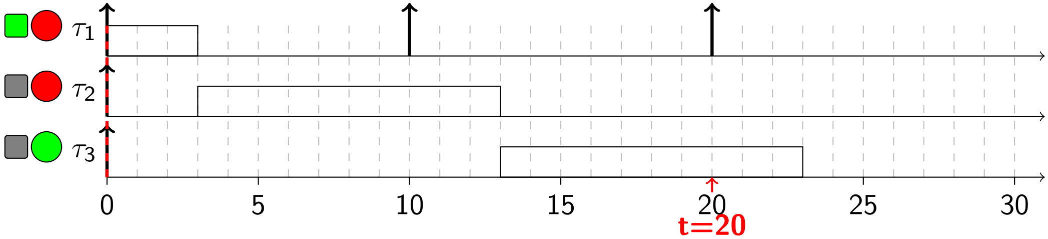
🔴 Not in wait set 🟠 In wait set 🟢 Executing

- - Polling Point

Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



⬛ Not activated 🟩 Activated

🔴 Not in wait set 🟠 In wait set 🟢 Executing

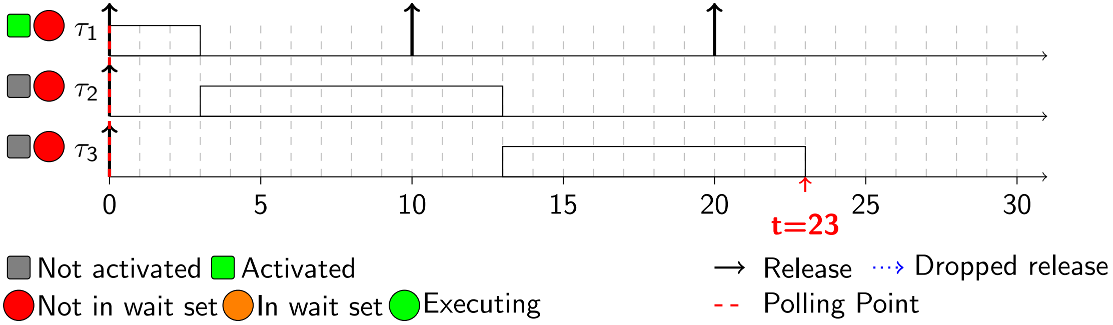
⟶ Release ⇢ Dropped release

-- Polling Point

Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

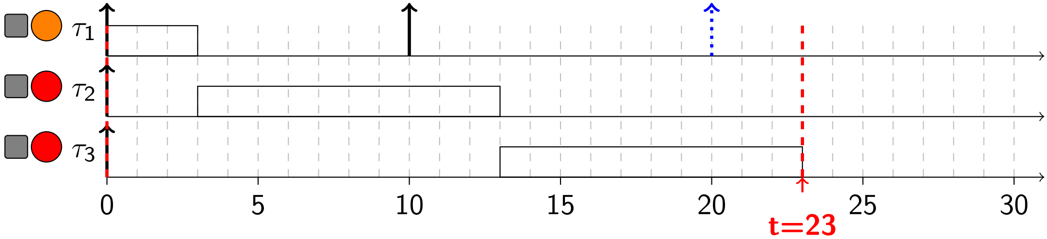
Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



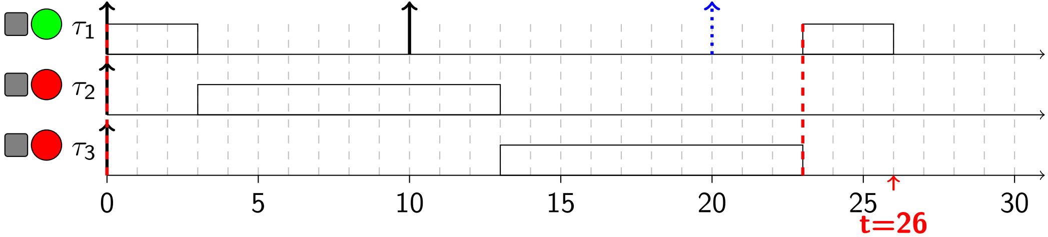
▨ Not activated 🟩 Activated ⟶ Release ⇢ Dropped release

🔴 Not in wait set 🟠 In wait set 🟢 Executing - - Polling Point

Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



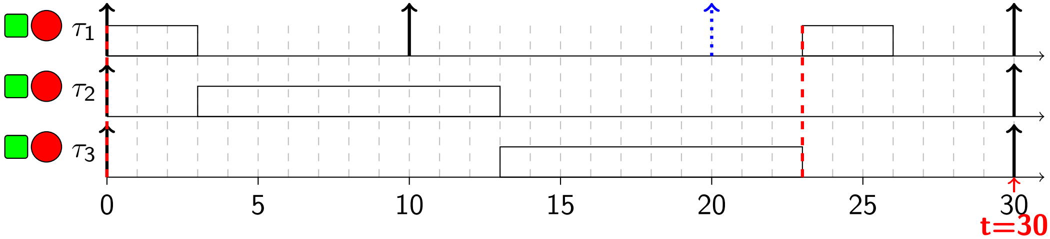
⬛ Not activated 🟩 Activated ⟶ Release ⋯→ Dropped release

🔴 Not in wait set 🟠 In wait set 🟢 Executing - - Polling Point

Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

ROS 2 Default Executor

Setting: ROS 2 Default Executor (non-preemptive, fixed priority)



▨ Not activated 🟩 Activated

🔴 Not in wait set 🟠 In wait set 🟢 Executing

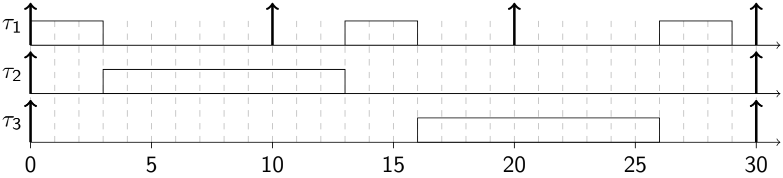
⟶ Release ⋯→ Dropped release

-- Polling Point

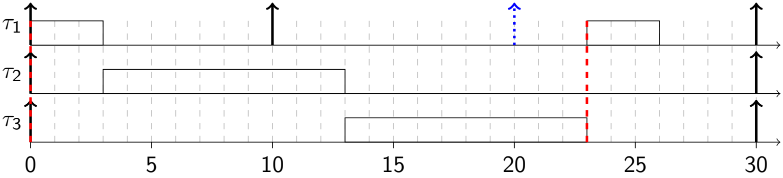
Task	Period (P)	WCET (C)	Priority
τ_1	10	3	1 (highest)
τ_2	30	10	2
τ_3	30	10	3 (lowest)

Schedule comparison

Classical Scheduling

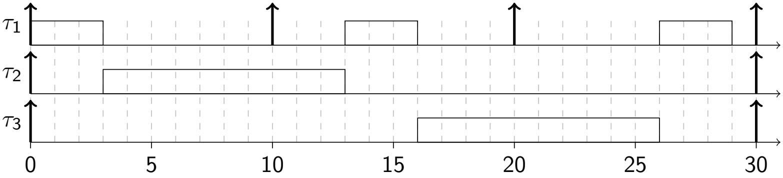


ROS 2 Default Executor

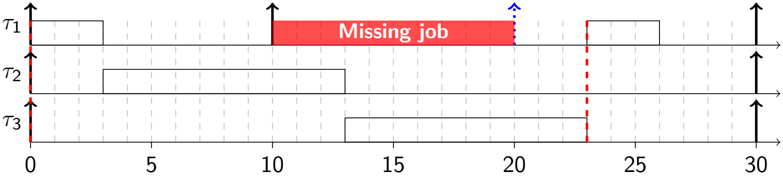


Schedule comparison

Classical Scheduling



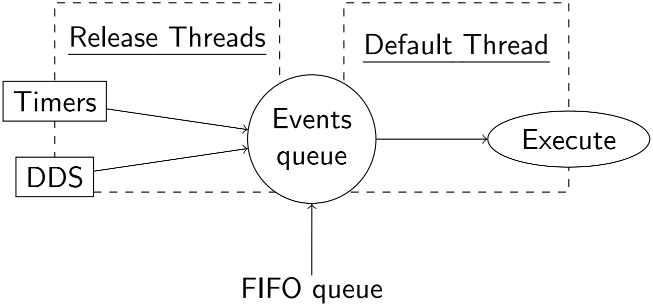
ROS 2 Default Executor



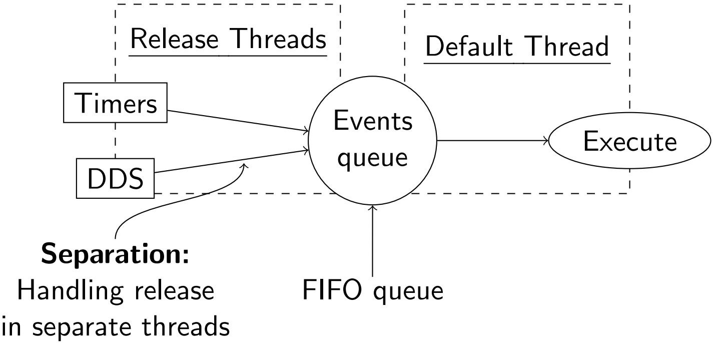
Problem Statement

Can we utilize the ROS 2 ecosystem to enable compatibility with classical real-time scheduling theory?

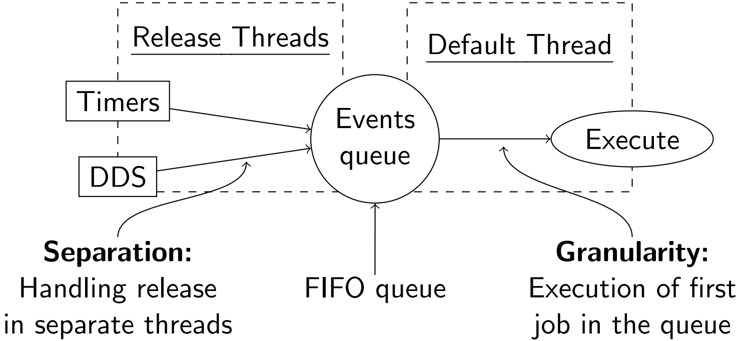
ROS 2 Events Executor



ROS 2 Events Executor



ROS 2 Events Executor



Release Threads

Default Thread

Timers

DDS

Events queue

Execute

Separation:
Handling release
in separate threads

FIFO queue

Granularity:
Execution of first
job in the queue

Enabling Compatibility with Classical Scheduling Theory

Classical Real-Time Scheduling	Robot Operating System 2 (ROS 2)	
Scheduler	**Events Executor**	**Default Executor**
Established theory	No theory	Limited theory
Deadline-driven	Best effort	Best effort
Fixed priority and dynamic priority	FIFO	Fixed priority
Periodic, sporadic tasks	Sporadic tasks	Sporadic tasks
Preemptive and non-preemptive	Non-preemptive	Non preemptive

Enabling Compatibility with Classical Scheduling Theory

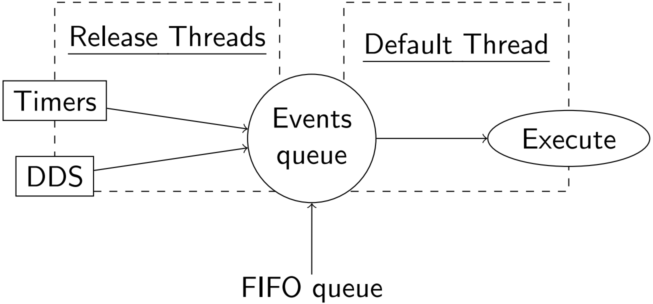
Classical Real-Time Scheduling	Robot Operating System 2 (ROS 2)	
Scheduler	**Events Executor**	**Default Executor**
Established theory	Established theory	Limited theory
Deadline-driven	Deadline-driven	Best effort
Fixed priority and dynamic priority	Fixed and dyn. pr.	Fixed priority
Periodic, sporadic tasks	Periodic tasks	Sporadic tasks
Preemptive and non-preemptive	Non-preemptive	Non preemptive

Can we utilize the ROS 2 Events Executor to enable compatibility with classical real-time scheduling theory?

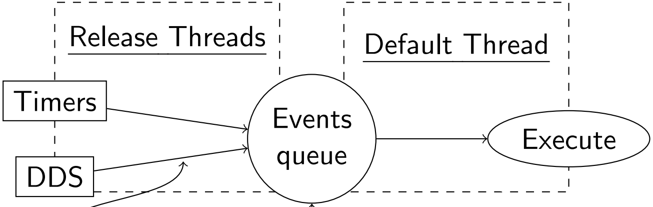
ROS 2 Events Executor - Subproblems

1. How to guarantee periodic release?
2. How to add priority-based scheduling?

ROS 2 Events Executor



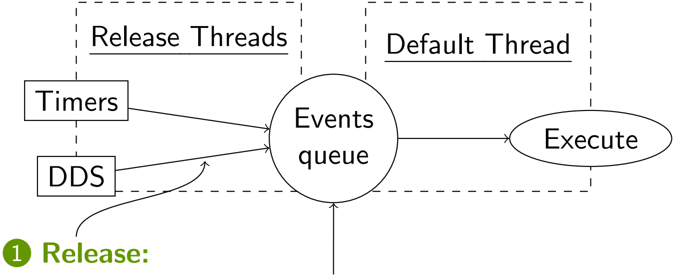
ROS 2 Events Executor



FIFO queue

❶ **Release:**
Guaranteeing periodic release

ROS 2 Events Executor



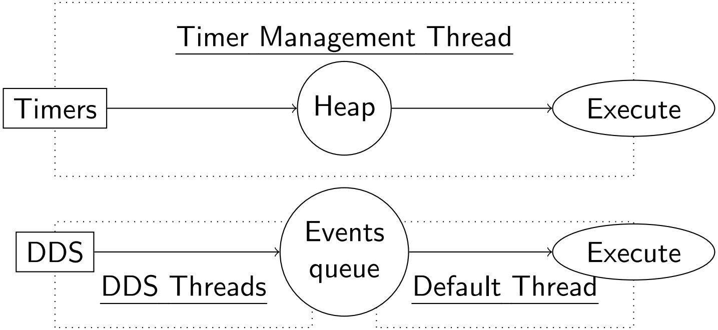
❶ **Release:**
Guaranteeing periodic release

FIFO queue

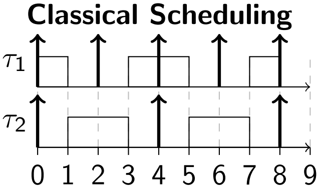
Configurations

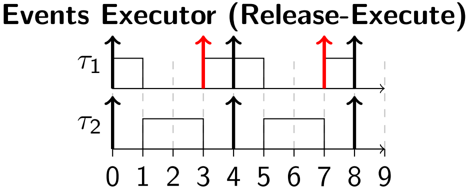
- **Release-Execute:** Separate thread for timer release **AND** execution
- **Release-Only:** Separate thread for timer release only

ROS 2 Events Executor - Release-Execute Configuration



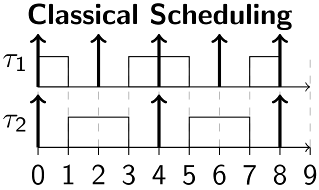
ROS 2 Events Executor - Release-Execute Schedule

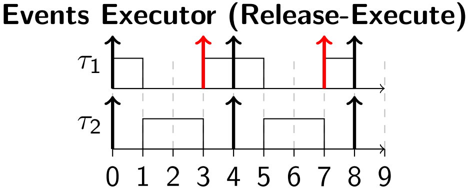




Task	Period (P)	WCET (C)	Priority
τ_1	2	1	1 (highest)
τ_2	4	2	2 (lowest)

ROS 2 Events Executor - Release-Execute Schedule



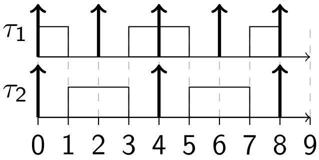


Task	Period (P)	WCET (C)	Priority
τ_1	2	1	1 (highest)
τ_2	4	2	2 (lowest)

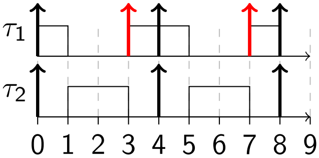
No separation of timer release and execution

ROS 2 Events Executor - Release-Execute Schedule

Classical Scheduling



Events Executor (Release-Execute)

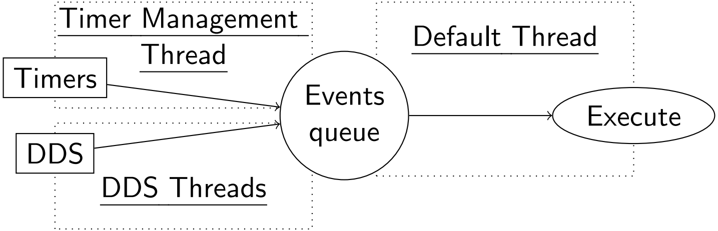


Task	**Period (P)**	**WCET (C)**	**Priority**
τ_1	2	1	1 (highest)
τ_2	4	2	2 (lowest)

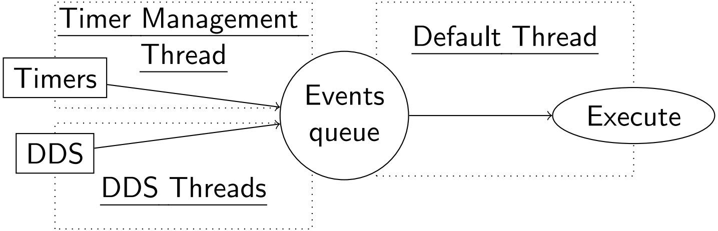
No separation of timer release and execution

→No guarantee of periodic release due to non-preemptive execution

ROS 2 Events Executor - Release-Only Configuration

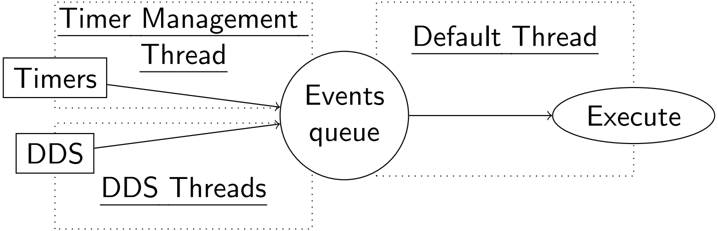


ROS 2 Events Executor - Release-Only Configuration



Separation of timer release and execution

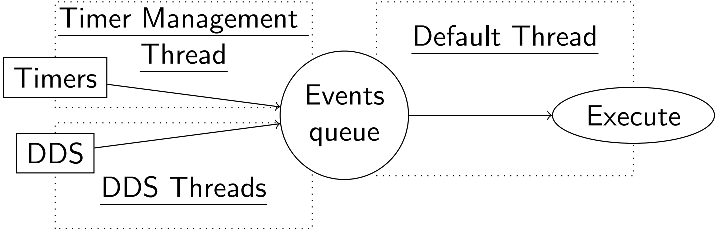
ROS 2 Events Executor - Release-Only Configuration



Separation of timer release and execution

→**Possibility of guaranteeing periodic release**

ROS 2 Events Executor - Release-Only Configuration



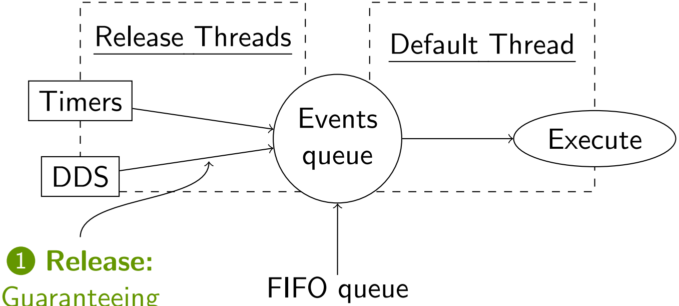
Separation of timer release and execution

→**Possibility of guaranteeing periodic release**

Requirements

- Preemptive thread scheduling
- Prioritization of release threads over default thread

ROS 2 Events Executor

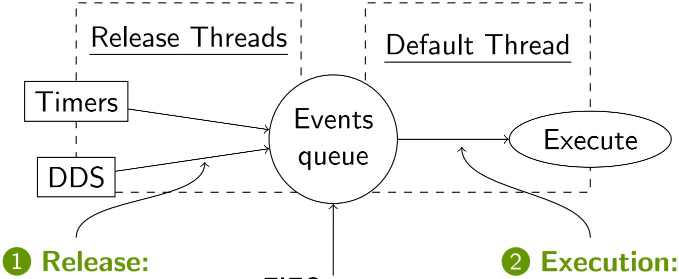


❶ **Release:**

Guaranteeing periodic release

✓

ROS 2 Events Executor

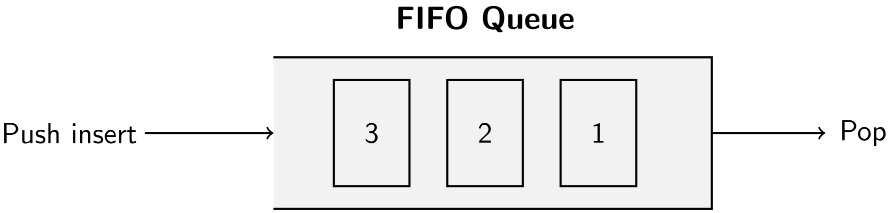


❶ Release:
Guaranteeing
periodic release
✓

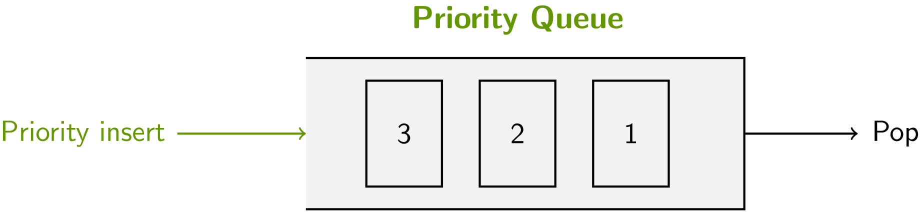
FIFO queue

❷ Execution:
Priority-based
scheduling

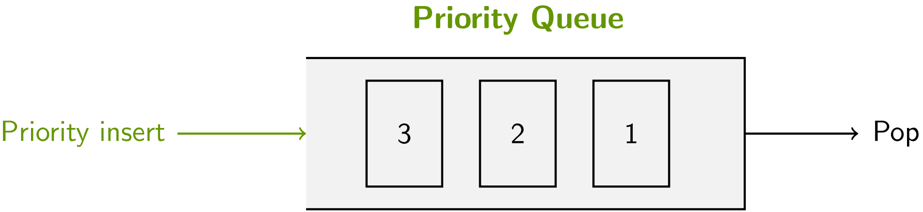
ROS 2 Events Executor - Priority-Based Scheduling



ROS 2 Events Executor - Priority-Based Scheduling



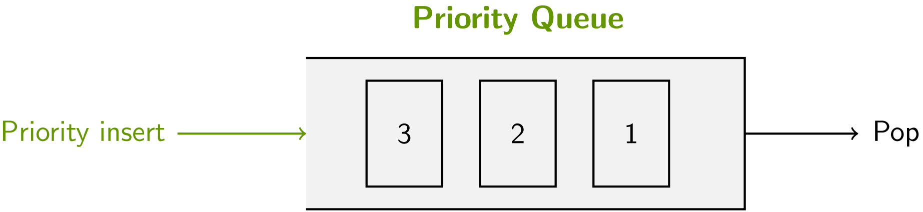
ROS 2 Events Executor - Priority-Based Scheduling



Prioritization

- Timers: Implicit priorities through periods

ROS 2 Events Executor - Priority-Based Scheduling



Prioritization

- Timers: Implicit priorities through periods
- Subscriptions: No prioritization interfaces provided by ROS 2

ROS 2 Events Executor - Priority-Based Scheduling

Proposal: Add universal priority field to ROS 2 tasks

ROS 2 Events Executor - Subscription Prioritization

Modeling as Sporadic Tasks

→ Each subscription gets a minimum inter-arrival time

🕒

> 0

ROS 2 Events Executor - Subscription Prioritization

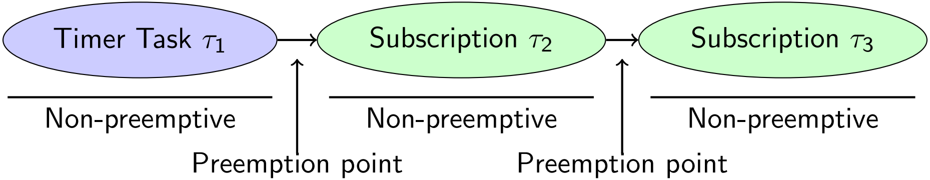
Modeling as Sporadic Tasks

→ Each subscription gets a minimum inter-arrival time

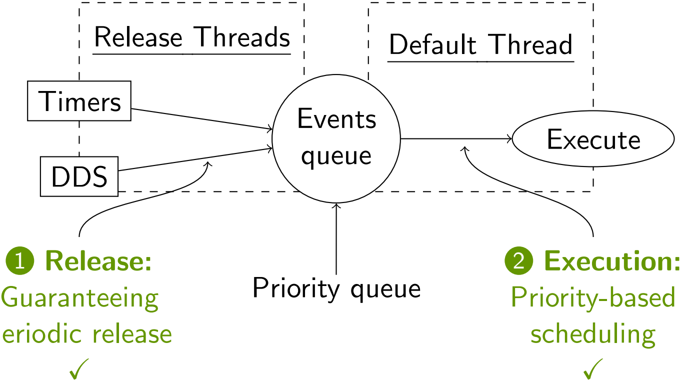
> 0

Modeling as Limited Preemptive Tasks

→ Subscriptions form processing chains



ROS 2 Events Executor - Compatibility



Release Threads

Default Thread

Timers

DDS

Events queue

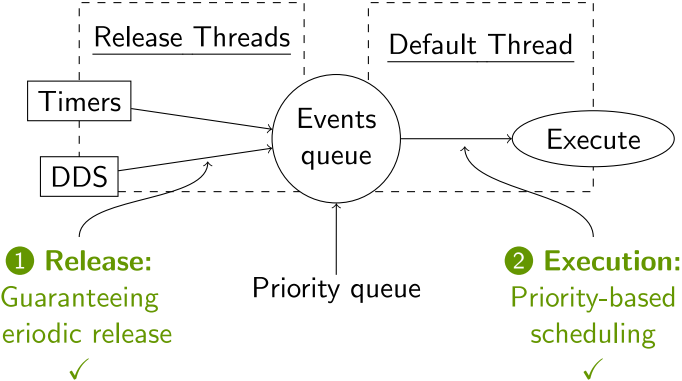
Execute

Priority queue

1 Release:
Guaranteeing periodic release
✓

2 Execution:
Priority-based scheduling
✓

ROS 2 Events Executor - Compatibility



Release Threads

Default Thread

Timers

DDS

Events queue

Execute

1 **Release:**
Guaranteeing
periodic release
✓

Priority queue

2 **Execution:**
Priority-based
scheduling
✓

→ **We can now apply classical scheduling theory to ROS 2!**

Evaluation

Experiments

1. Response time comparison (timer-only)
2. End-to-end latency comparison (timer-only)
3. Autoware reference system performance (timer + subscription tasks)

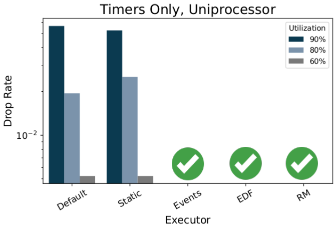
Executors

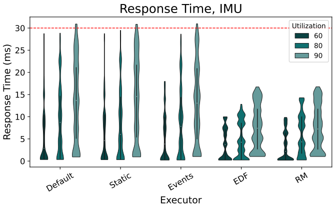
(Static) Default Executor, Events Executor, Modified Events Executor (RM, EDF)

Evaluation: Response Time

Experimental Setup:

- 10 periodic timer tasks (camera, LIDAR, IMU)
- Varying loads (30%, 60%, 90%)
- Metrics: Dropped jobs, response time, deadline misses





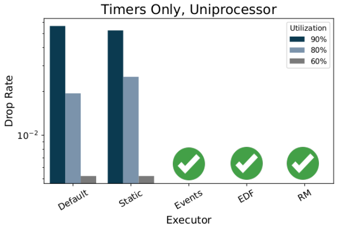
Evaluation: Response Time

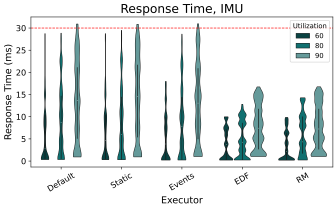
Experimental Setup:

- 10 periodic timer tasks (camera, LIDAR, IMU)
- Varying loads (30%, 60%, 90%)
- Metrics: Dropped jobs, response time, deadline misses

Results

- No more dropped jobs
- No more deadline misses

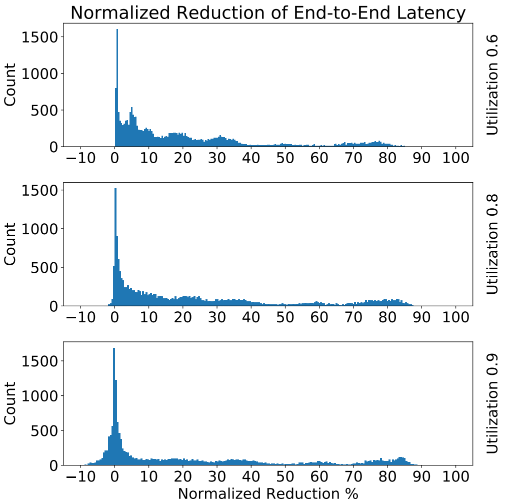




Evaluation: End-to-End Latency

Experimental Setup:

- WATERS benchmark
- Varying loads (30%, 60%, 90%)
- Metric: End-to-end latency reduction between default and our executor



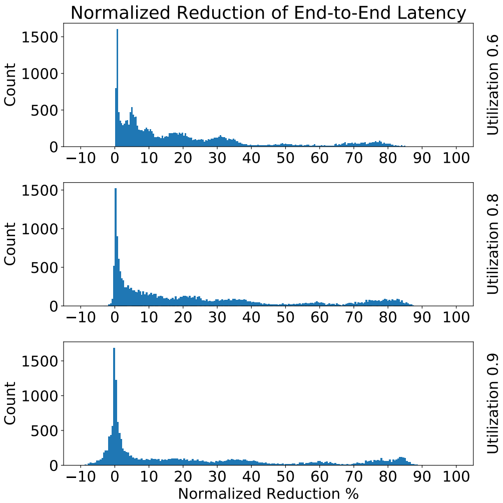
Evaluation: End-to-End Latency

Experimental Setup:

- WATERS benchmark
- Varying loads (30%, 60%, 90%)
- Metric: End-to-end latency reduction between default and our executor

Results

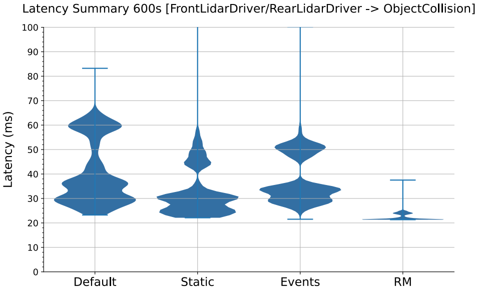
- Latencies greatly reduced
- Reductions up to 90%



Evaluation: Autoware Reference System

Experimental Setup:

- Autoware reference system
- Measurement of hot path
- Metric: End-to-end latency



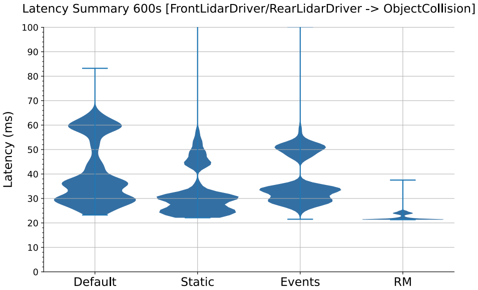
Evaluation: Autoware Reference System

Experimental Setup:

- Autoware reference system
- Measurement of hot path
- Metric: End-to-end latency

Results

- Lower mean and variance
- Much lower maximum latencies



Conclusion

- Bridged the gap between ROS 2 and classical scheduling theory
- Proposed modifications to enable timing guarantees in ROS 2
- Enabled application of established analytical methods for ROS 2 systems
- Provided tighter bounds on response times and end-to-end latencies