RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

cnes

Towards Multi-Policy Hierarchical Scheduling in Linux for Containerized Space Applications

Merlin Kooshmanian, Jérôme Ermont, Lucas Miné, Simon Corbin, Frédéric Boniol

RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

1. Introduction

cnes

1. Introduction

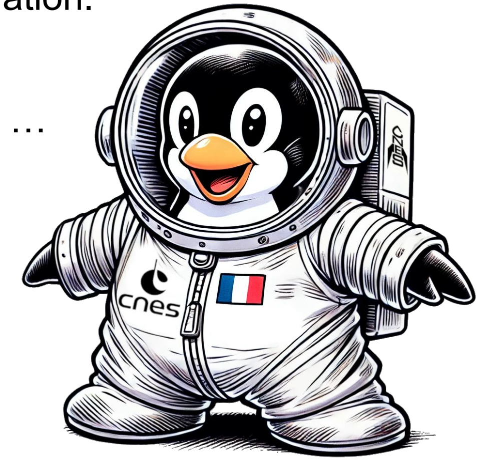
Why Using Linux for Space Applications ?

Current Solutions : RTOS and Hypervisors

- **Pros:** Deterministic, strong isolation, certifiable (DO-178C).
- **Cons:** Limited flexibility, restricted language/library support, static configuration.

Linux

- **Pros:** Open-source, rich ecosystem (languages, libraries, tools), scalable, …
- **Cons:** Limited isolation, unpredictable behavior for critical applications.



Need

- **Complement existing solutions** by **improving** Linux **guarantees**:
  - Enhanced isolation.
  - Performance for heterogeneous workloads.

2. Context

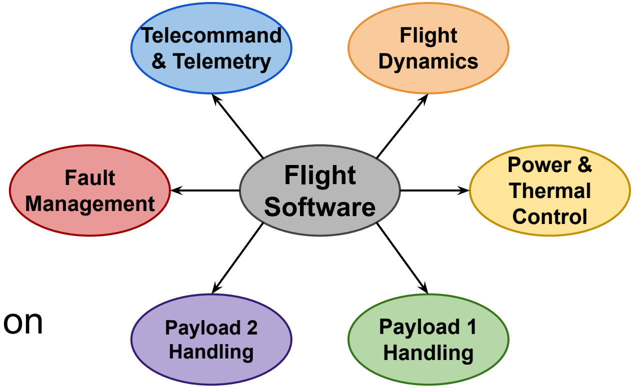
2.1. Spacecraft Flight Software

Diverse Functions

- Attitude and orbit control, power management, payload operations
- Monitoring, diagnostics, and fault management

Heterogeneous Requirements

- Varying **criticality** levels
- Mixed **timing constraints**



Needs

- **Isolation**
  - **Prevent fault propagation** by **separating** functions based on **criticality levels**.
  - **Simplify development**, validation, and integration by **reducing interdependencies**.
- **Optimized Scheduling**
  - Handle **heterogeneous workloads** with **mixed timing** constraints

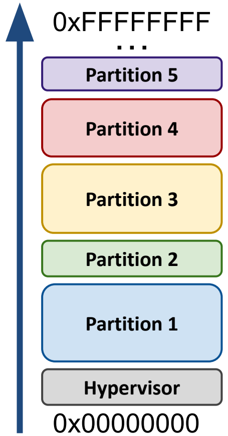
2.2. Isolation Concepts

Spatial Isolation

Ensures that each component can **access only** its **explicitly assigned memory** and **hardware** resources, and **cannot read** or **modify** the **state** of **other components**.

Examples :

- Memory partitioning (static or dynamic)
- Access control & permissions



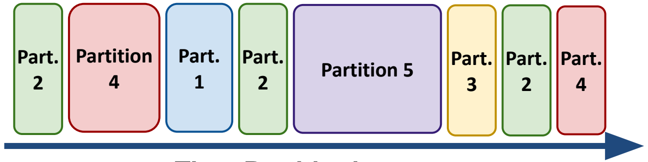
Memory Partitioning

Temporal Isolation

Ensures that the **timing behavior** of a component (e.g., execution time, response time, deadline satisfaction) **is independent** of the execution behavior **of other components**, within defined resource budgets.

Examples :

- Temporal partitioning
- Budget enforcement



Time Partitioning

⚠ *For our work, we adopt a **relaxed view** of **temporal isolation**, focusing on maintaining **bounded response-time guarantees** for tasks to **ensure predictable timing** behavior.*

2.3. Isolating Linux Applications

Hypervisor-Based Isolation

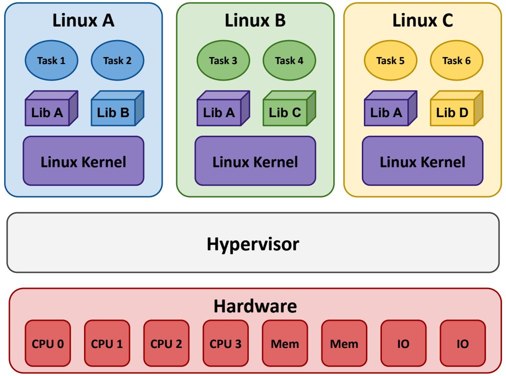
(Linux runs as a Guest OS in a virtualized partition)

Pros

- **Strong** spatial and temporal **isolation**

Cons

- **Limited flexibility**
- **High** memory **footprint**
- Hypervisor **overhead**



Hypervisor + Linux Partitions

Native Linux Isolation

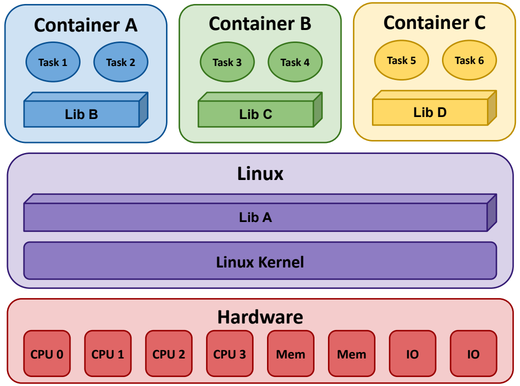
(Linux is the Host OS; isolation via processes, containers, etc.)

Pros

- **Low** memory **footprint**
- **Less overhead**

Cons

- Mostly **software-based** isolation
- **Limited guarantees**
- **Weak temporal** isolation



Linux + Containers

2.4. Linux Scheduling

POSIX Scheduling Policies

- **SCHED_FIFO**, **SCHED_RR**, **SCHED_OTHER** defined by **POSIX**
- **Linux** extension: **SCHED_DEADLINE**

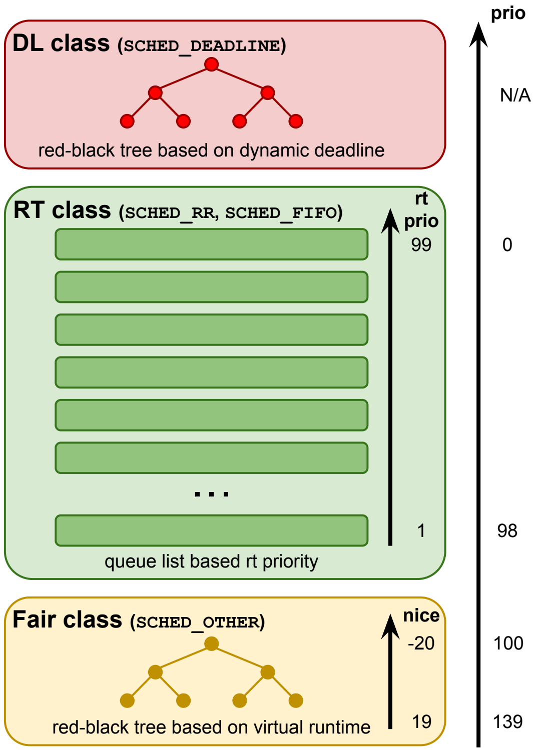
Linux Scheduling Classes vs Policies

- **DL class** ⇔ SCHED_DEADLINE : EDF + Constant Bandwidth Server (CBS) for bandwidth enforcement
- **RT class** ⇔ SCHED_FIFO, SCHED_RR : fixed-priority preemptive scheduling (FIFO vs RR when same priority)
- **FAIR class** ⇔ SCHED_OTHER : fair CPU sharing

For convenience, policies will be grouped by this 3 classes

Design

- Linux is a **multi-policy** OS : FAIR, RT, DEADLINE **coexist** at runtime
- Strict **class priority**: DEADLINE > RT > FAIR



2.5. Multi-Policy Scheduling

Limits of Single-Policy Scheduling

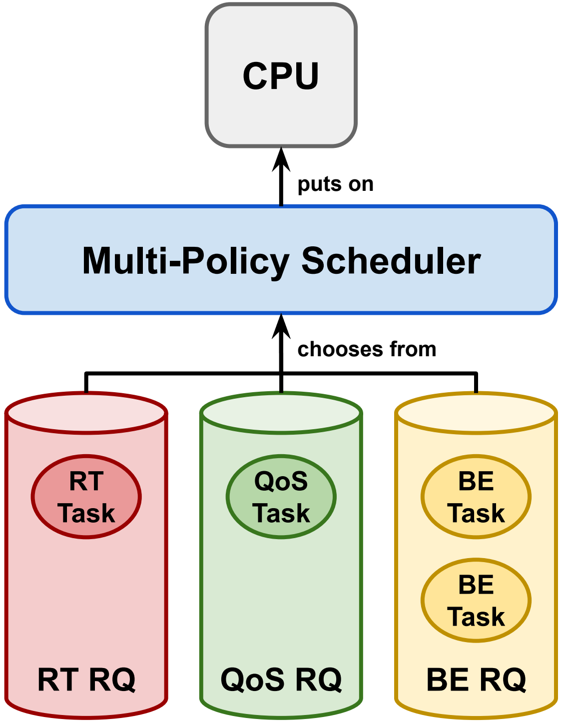
- **Optimized** for a **single objective** (e.g., reactivity, fairness, etc.)
- **Inefficient** for **heterogeneous software** needs

Origins of Multi-Policy Scheduling

- Emerged in **general-purpose operating systems**
- Designed to meet **diverse user requirements**

Key Benefits

- Provides **all** scheduling **policies at runtime**
- Enables **better fit** for **heterogeneous** workload **requirements**



2.6. Temporal isolation within Linux : HCBS patch

Overview

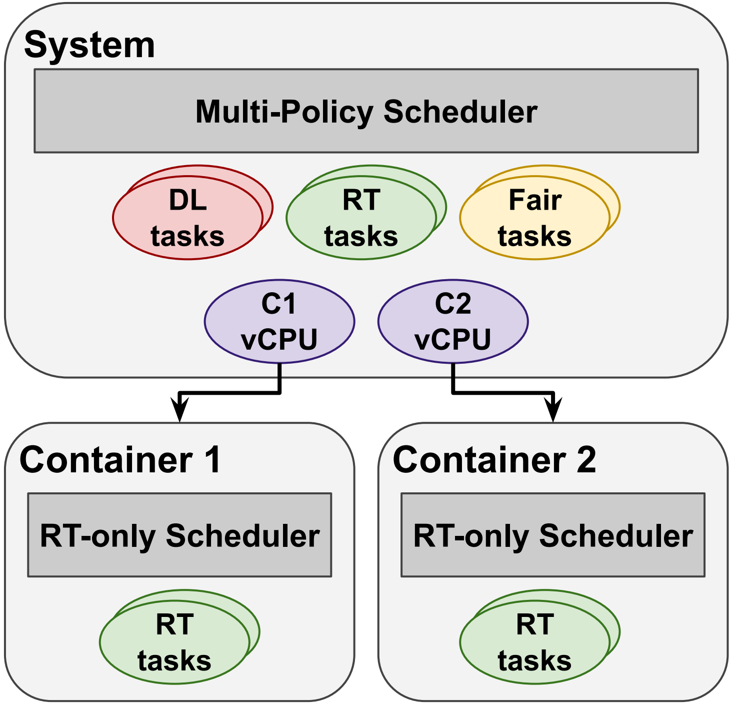
- **Hierarchical Constant Bandwidth Server** (HCBS) enables hierarchical scheduling for the RT class
- Developed by **Abeni et al**. in 2019 [1] and improved by **Andriaccio et al**. in 2025 [2]

Key Features

- Based on **deadline-servers** (CBS) to enable a **guaranteed bandwidth enforcement** between containers
- Associated **RT runqueues** with **deadline servers**, making it only suitable for RT tasks

Limitation

- **Mono-policy**, relies solely on RT class
- **Not suited** for task sets with **heterogeneous objectives**



➡ ***HCBS creates a hierarchical scheduling in Linux***

HCBS (original): https://github.com/lucabe72/LinuxPatches/tree/HCBS
HCBS (updated): https://github.com/Yurand2000/HCBS-patch/tree/rt-cgroups

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

3. Problem Statement

cnes

3.1. Current Capabilities and Limitations with Linux

Linux-based Flight Software Needs

- **Spatial isolation**
  Strict resource separation (memory, CPU, devices))
- **Temporal isolation**
  Guaranteed CPU bandwidth and bounded latency
- **Scheduler for heterogeneous workloads**
  Ability to handle tasks with varying constraints

Native Linux Current Capabilities

✓ **Spatial isolation**
Achieved via processes, cgroups, and namespaces

✗ **Temporal isolation**
No guaranteed temporal isolation

✓ **Scheduler for heterogeneous workloads**
Supports multiple scheduling policies

HCBS

✓ **Spatial isolation**
Achieved via processes, cgroups, and namespaces

✓ **Temporal isolation**
Hierarchical Scheduling including BW enforcement

✗ **Scheduler for heterogeneous workloads**
Supports only RT tasks in containers

3.2. Objective: Hierarchical Multi-Policy Scheduling

Problem Summary

Native Linux supports **multi-policy** scheduling but **lacks temporal isolation** guarantees. **HCBS** provides **hierarchical scheduling** with **bandwidth enforcement** but restricts scheduling to **real-time** tasks **only**.

Objective

Extend hierarchical scheduling to support all Linux scheduling class (RT, DEADLINE, FAIR) within containers while **preserving**:

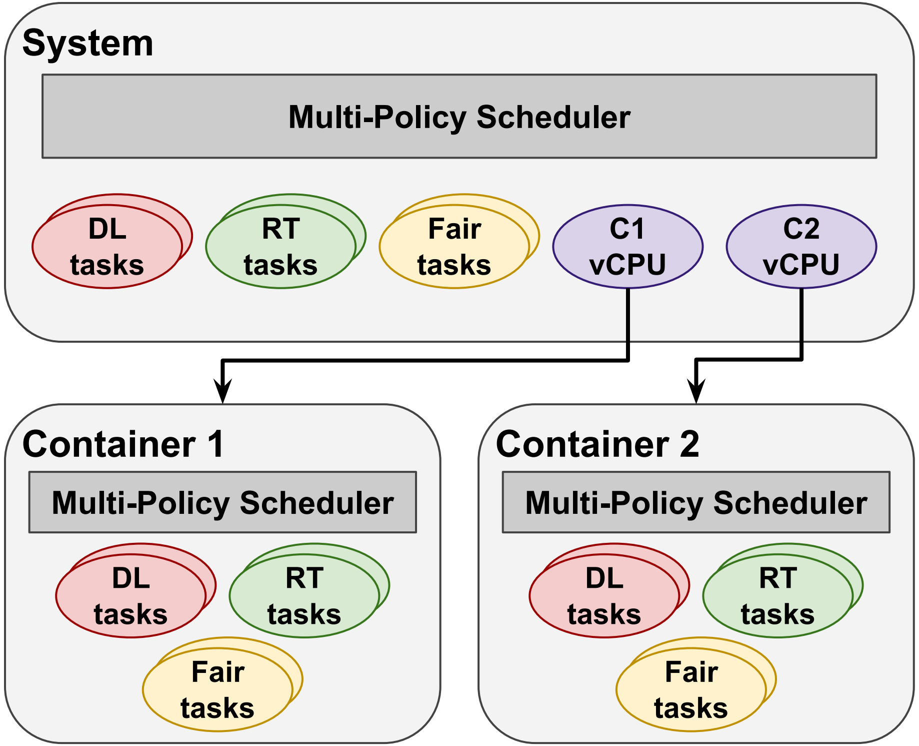
- **Temporal isolation** via bandwidth enforcement
- **Multi-policy flexibility** for heterogeneous workloads

4. Contribution

4.1. TGBS: Task Group Bandwidth Server

Overview

- TGBS **enforces** container-level **temporal isolation** while **preserving multiple** scheduling **policy** within each container
- The core idea behind TGBS is to:
  - **Recreate scheduling** at the container level by **virtualizing CPUs**.
  - **Guarantee temporal isolation** between virtual CPUs.



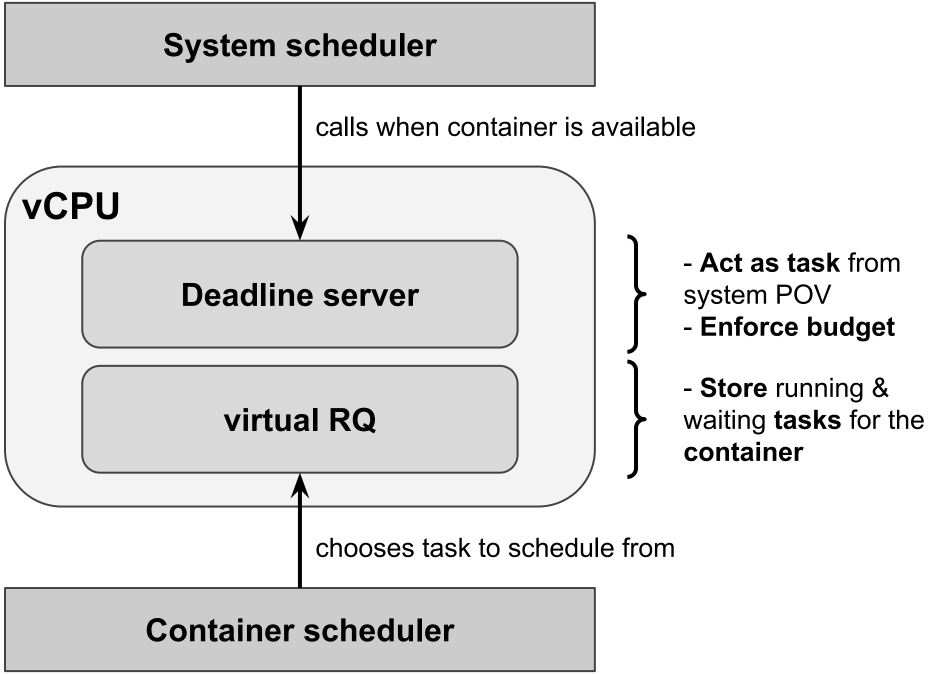
Limitation

This work assumes a **SINGLE-CORE** environment. Multi-core scheduling is not yet supported and remains a direction for future research.

TGBS : https://github.com/mkooshmanian/TGBS

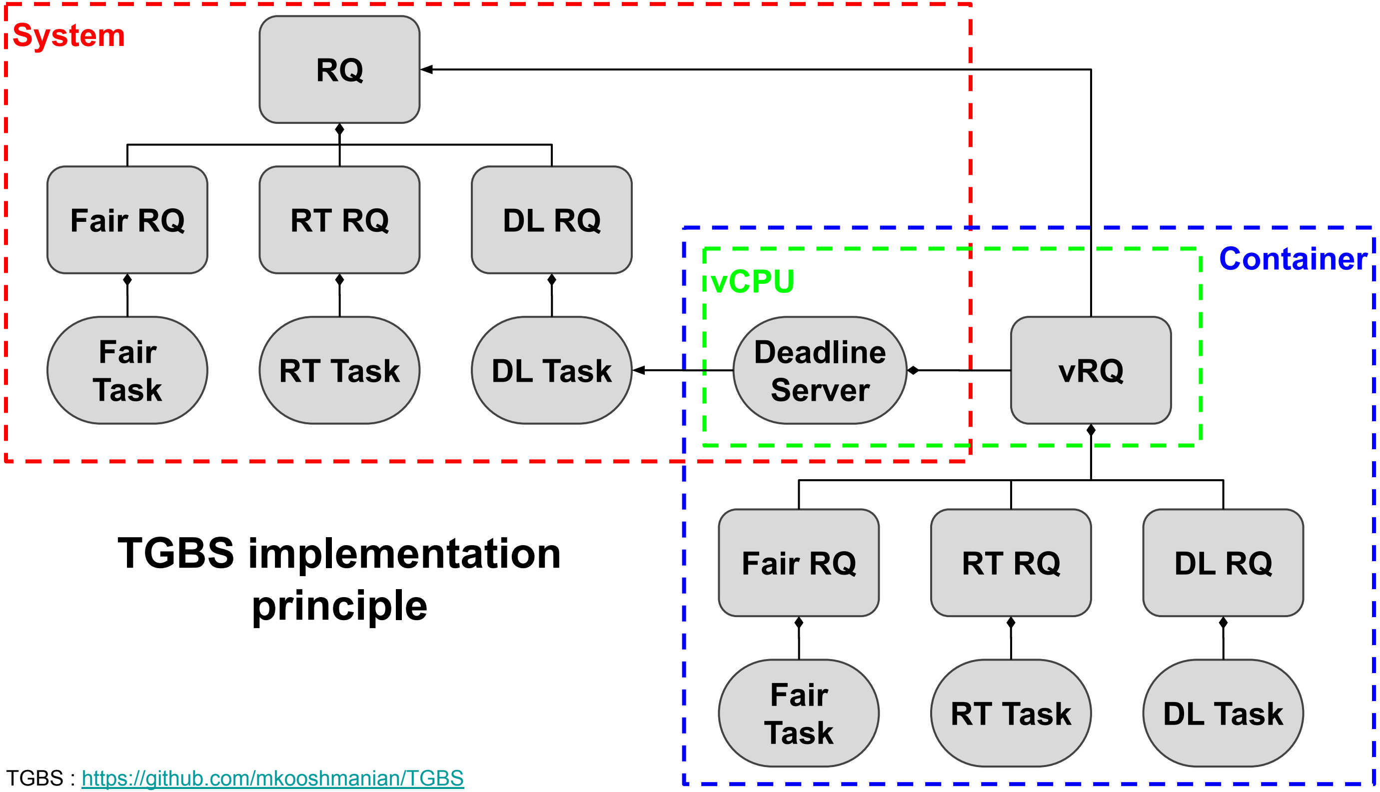
4.2. TGBS: CPU Virtualisation

- Rely on **deadline server** and **virtual runqueue**
- ***deadline server***
  - is a system **scheduling entity**
  - **schedule** by the **Constant Bandwidth Server** (CBS) : BW enforcement
  - designed to **serve** other **tasks**
- ***virtual runqueue***
  - **lightweight** version of a real **runqueue**



TGBS : https://github.com/mkooshmanian/TGBS

4.3. TGBS: Implementation Details

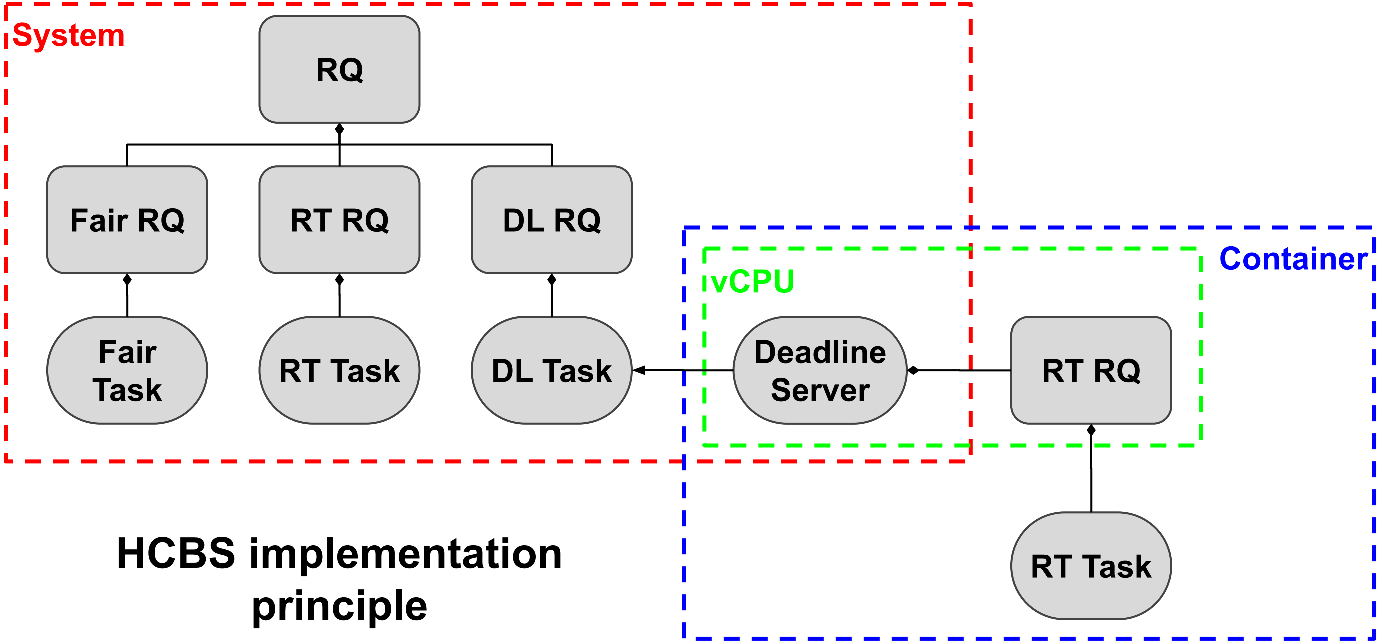


TGBS implementation principle

- TGBS **mirrors** Linux's **native scheduling structure** within containers.
- **Virtual runqueues replicate class-specific queuing** methods for all scheduling policies.

TGBS : https://github.com/mkooshmanian/TGBS

4.4. HCBS: Implementation Details (comparison)



HCBS implementation principle

- HCBS attaches **only** an **RT runqueue**, restricting it to real-time tasks.
- This **limits** HCBS to **mono-policy** scheduling.

HCBS (updated): https://github.com/Yurand2000/HCBS-patch/tree/rt-cgroups

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

5. Experimental Evaluation

cnes

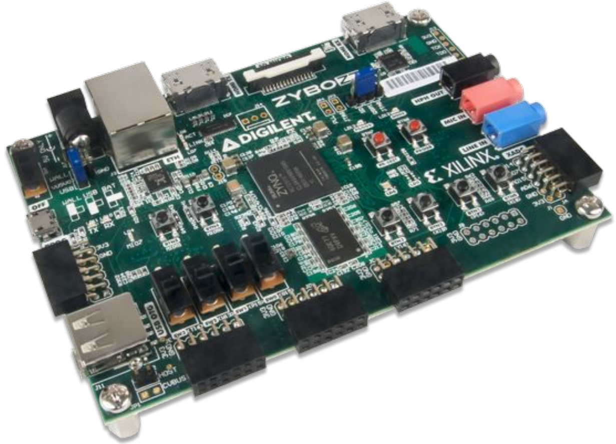
5.1. Objective and Setup

Objective

- **Evaluate TGBS** patch
  - Assess functional correctness and real-time performance
  - Compare with Vanilla Linux and HCBS

Setup

- **Zybo Z7** development board
  - Dual-core ARM Cortex-A9 (666 MHz) → **disabled 1 CPU**
  - 1 GB DDR3 memory
- Linux kernel **v6.17** configured with **PREEMPT-RT** extension
- **TGBS** patch applied for hierarchical and multi-policy scheduling



Zybo Z7 development board (Source: Digilent)

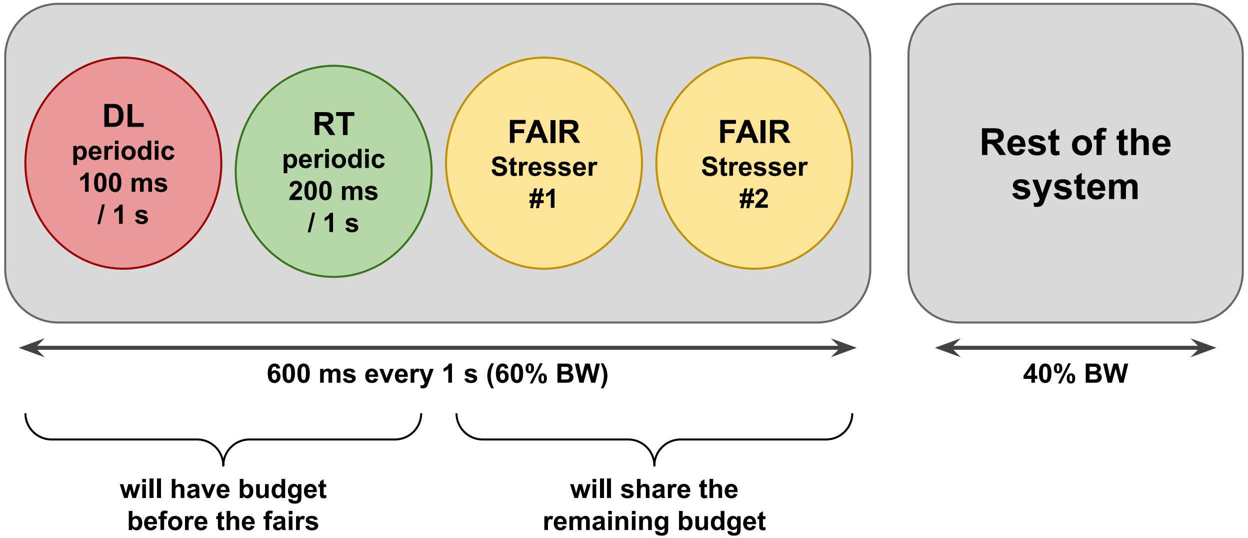
5.2. Functional Validation - Objectives

Objective

- Verify **correct prioritization** between **policies**
- **Ensure budget** enforcement is working properly

Setup

- 1 container with 4 threads
- 1 thread / policy → check multi-policy
- Stresser → check budget enforcement



Expected results

- **Prioritization**
  DL > RT > Fair
- **Budget enforcement**
  $U_{tot} \approx 60\%$
- **Fairness**
  U(stress#1)≈ U(stress#2)

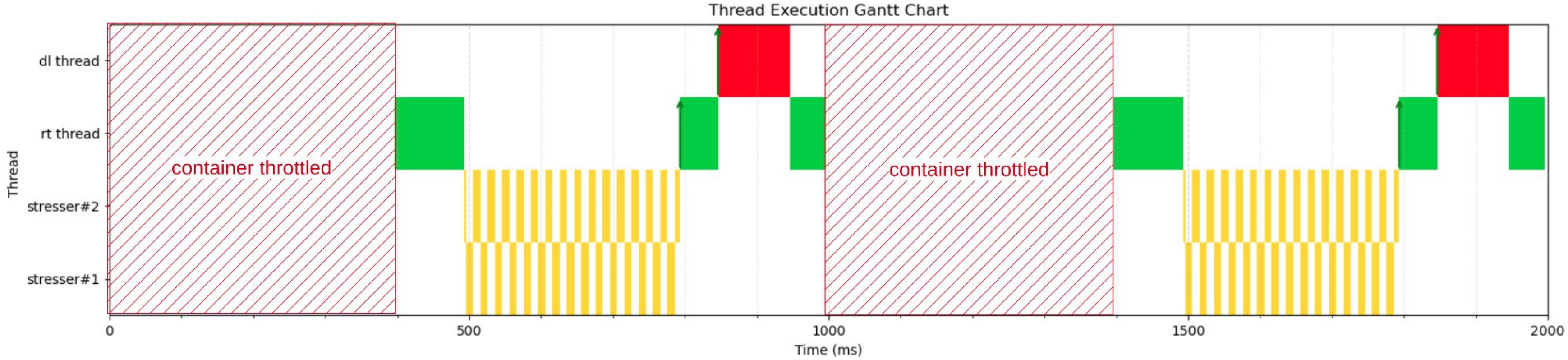
5.3. Functional Validation - Results

Results

- Measured **CPU shares** match **expected proportions**
- **Correct** enforcement of **container reservation**
- **Preemption** between different **policies** is **respected**

CPU usage under a 600 ms / 1 s container reservation.

Task	Policy	CPU share
DL task (100ms / 1s)	DEADLINE	≈ 10%
RT task (200ms / 1s)	FIFO	≈ 20%
FAIR task (stresser #1)	OTHER	≈ 15%
FAIR task (stresser #2)	OTHER	≈ 15%



5.4. Overhead Evaluation - Objectives

Objective

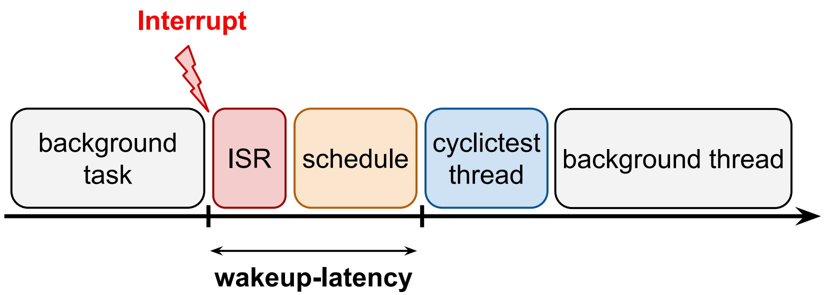
- Measure TGBS **scheduling overhead**
- **Compare** TGBS with **Vanilla** Linux and **HCBS**

Methodology

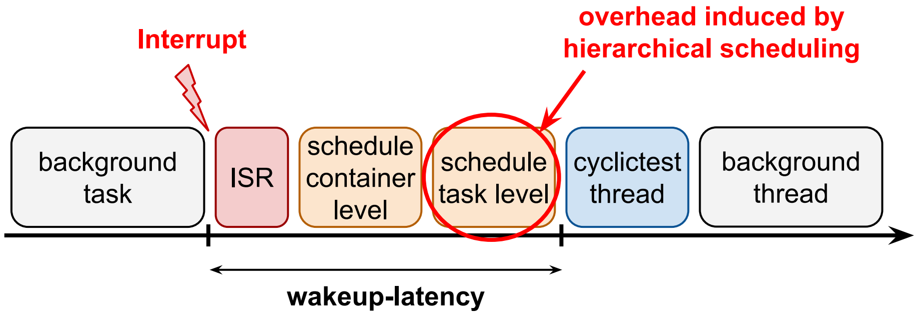
- Use **cyclictest** (rt-tests library) : evaluate **wake-up latency** (IRQ trigger to CPU execution)
- Run inside container for HCBS and TGBS

Expectations

Vanilla Linux



TGBS or HCBS Linux



5.5. Overhead Evaluation - Results

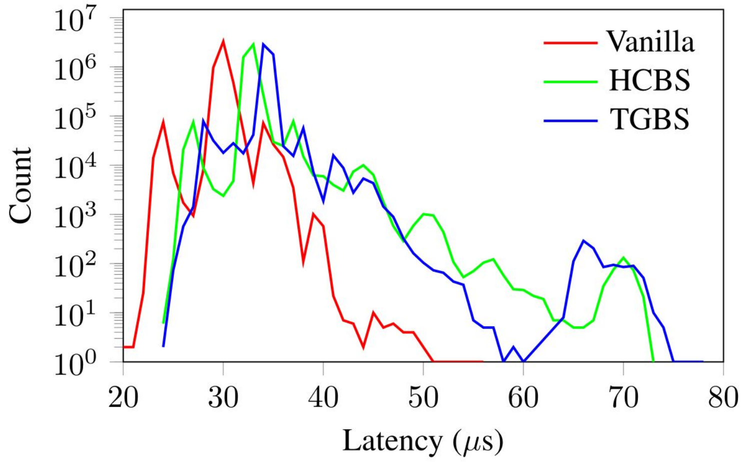
Results

- Avg latency: 30 µs (Vanilla) → 34 µs (TGBS) ≈ +15%
- Max latency: 56 µs (Vanilla) → 78 µs (TGBS) ≈ +40%
- **TGBS** overhead **comparable** to **HCBS**
- **No pathological behavior** observed

Cyclictest Latency Statistics (µs)

System	Min	Avg	Max	Std Dev
Vanilla	20	30	56	1.3
HCBS	24	33	73	1.7
TGBS	24	34	78	1.5

Cyclictest Latency Distribution



RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

6. Conclusion

cnes

6. Conclusion and Future Work

Summary

- Demonstrated the **necessity** for **multi-policy** and **hierarchical scheduling** in future spacecraft flight software
- **Developed** and **tested** a Linux **kernel patch** (**TGBS**) to provide **temporal isolation** for **heterogeneous workloads**

Future Work

- **Address** the **current gap** in literature regarding multi-policy and hierarchical models
- Develop a **formal model** of **TGBS** to prove its **correctness** and **timing guarantees**
- **Extend** TGBS to support **multicore** architectures

RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

Thanks for listening !
Any questions ?

TGBS patch GitHub link

contact : merlin.kooshmanian@onera.fr

cnes

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

Appendix

cnes

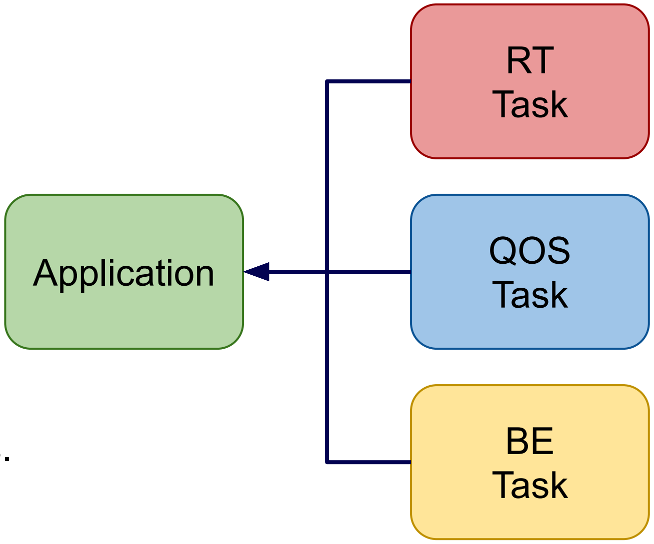
A. Task Classes

Diverse Operational Requirements

- **Heterogeneous** timing, reliability, and **operational demands** across space system functions
  - Predictability (critical control tasks)
  - Responsiveness (event driven tasks)
  - Fair sharing (variable workloads tasks)
  - Opportunistic execution (background tasks)

Task Types

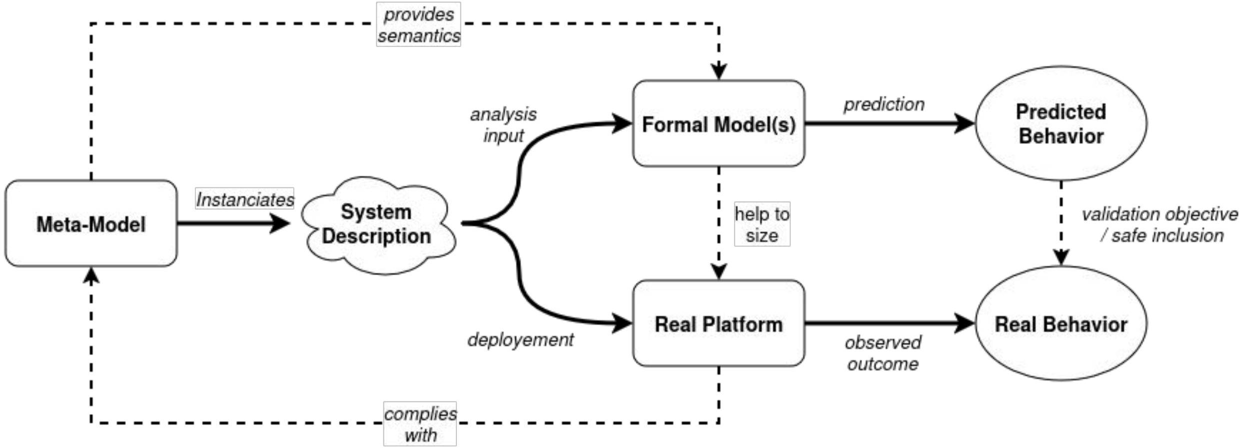
- **Real-Time (RT):** Hard deadlines, zero tolerance for delays.
- **Best-Effort (BE):** no needs, runs when possible
- **Quality-of-Service (QoS):** Predictable progress, flexible timing.



B. Methodology Overview

Structured Process

- **Links** conceptual **scheduling** systems to real platform **execution**
- **Ensures coherence** from **concept** definition to **implementation**



Core Elements

- **Meta-model / System Description**
  Defines and instantiates concepts
- **Real Platform**
  Executes the system description, yielding observed behavior

- **Formal Models :**
  - Provide conservative predictions for schedulability analysis
  - Provide real platform sizing methods

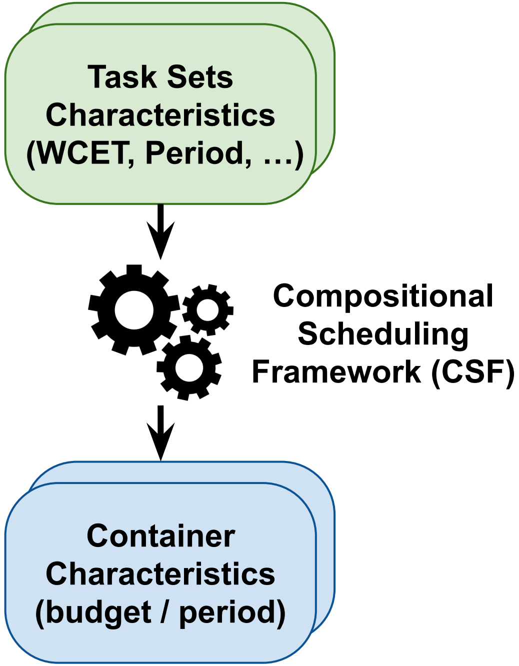
C. Container Sizing – CSF

Problem

- Need to **ensure** hard (RT) and soft (QoS) task **timing constraints** within container
- Guarantee **container schedulability** in **all situations** (migration, appearance, disappearance)

Solution - Compositional Scheduling Framework

- Designed for hierarchical systems; **compositional**, ensuring container **schedulability independently**
- Based on **demand /supply mechanism**
- But **pessimistic** and assumes **mono-political scheduling** (each component has a mono-policy scheduler)



D. Validation of the Container Sizing

Objective

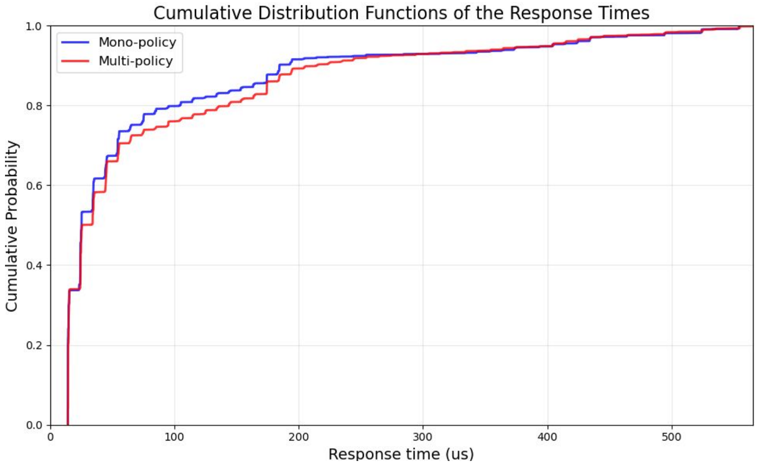
- **Validate** end-to-end **methodology**.
- Ensure **correct execution** under **TGBS**

Methodology

- **Synthetic workload** of **10 periodic tasks**, total utilization 50% ± 5%
- Tasks distributed across **two containers** (A and B) sized with **CSF** assuming mono-policy.
- **Mix** of **RT** tasks (SCHED_FIFO, RM priorities) and **QoS** tasks (SCHED_OTHER)

Results

- **No** deadline **misses** observed
- Analytically sized **reservations sufficient**
- **Correct execution** under both mono-policy and multi-policy configurations



Note : Mono-policy demonstrates slightly better response times than multi-policy, as expected since mono-policy prioritizes only system responsiveness, whereas multi-policy focuses also on fairness.