Supporting Low-Latency, Low-Criticality Tasks in a Certified Mixed-Criticality OS

Manohar Vanga

Andrea Bastoni

Björn Brandenburg

Henrik Theiling

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

SYSGO EMBEDDING INNOVATIONS

Mixed-Criticality Systems

Increasing **SWaP constraints** → **Consolidation of components with different criticalities** onto shared hardware platforms → **Mixed-Criticality Systems**

Mixed-Criticality Systems

Increasing **SWaP constraints**

Consolidation of components with different criticalities onto shared hardware platforms

Mixed-Criticality Systems

Key Challenges

Ensuring **isolation** between tasks of different criticalities.

Ensuring that **throughput and latency requirements** of all tasks are met.

What is Criticality?

What is Criticality?

What is Criticality?

What is Criticality?

Level of **failure assurance** a task is certified for.

e.g. **DAL levels** in DO-178B/C

Criticality vs. Importance

What is Criticality?

Level of **failure assurance** a task is certified for.

e.g. **DAL levels** in DO-178B/C

What is "importance"?

Any feature that is crucial to the **commercial success** of a product.

e.g. **touch GUI** in cars

Criticality vs. Importance

What is Criticality?	What is "importance"?
Level of **failure assurance** a task is certified for.	Any feature that is crucial to the **commercial success** of a product.
e.g. **DAL levels** in DO-178B/C	e.g. **touch GUI** in cars

Key point: a task may be of low criticality but still important!

A Case Study

RTOS consisting of a hypervisor-based **separation microkernel** designed for the highest levels of safety and security.

Deployed across **many safety-critical domains** including avionics, automotive, and transportation applications.

Certified on a wide range of projects using various **certification standards**, including DO-178B/C, IEC 61508, EN 50128.

A Case Study

PikeOS

Strong, battle-tested support for **high-criticality tasks**.

the highest levels of safety and security.

Deployed across
domains
and transportation applications

Certified on a wide range of projects
using various
including DO-178B/C, IEC 61508, EN 50128

A Case Study

PikeOS

Strong, battle-tested support for **high-criticality tasks**.

the highest levels of safety and security.

Deployed across

How can we integrate support for **low-criticality tasks**?

Certified on a wide range of projects using various
including DO-178B/C, IEC 61508, EN 50128

Our Paper: A Summary

Identify **deficiencies in low-criticality support** in PikeOS

Highlight **key design constraints** required in a commercial context, and **typically not addressed in academic designs**.

Present a **minimally-invasive extension of the PikeOS scheduler** to address the determined deficiencies.

Design and implementation of a **prototype in PikeOS**, with results from a **freely-shareable re-implementation in LITMUS^{RT}**.

Rest of this Talk

I *The problem of low-criticality, low-latency tasks*

II *Working within real-world design constraints*

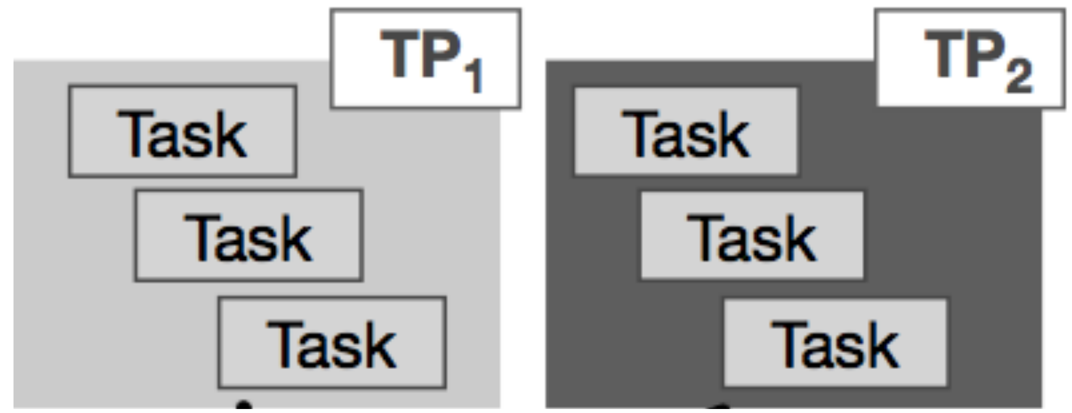
III *Our proposed scheduler extensions*

Rest of this Talk

I	*The problem of low-criticality, low-latency tasks*
II	*Working within real-world design constraints*
III	*Our proposed scheduler extensions*

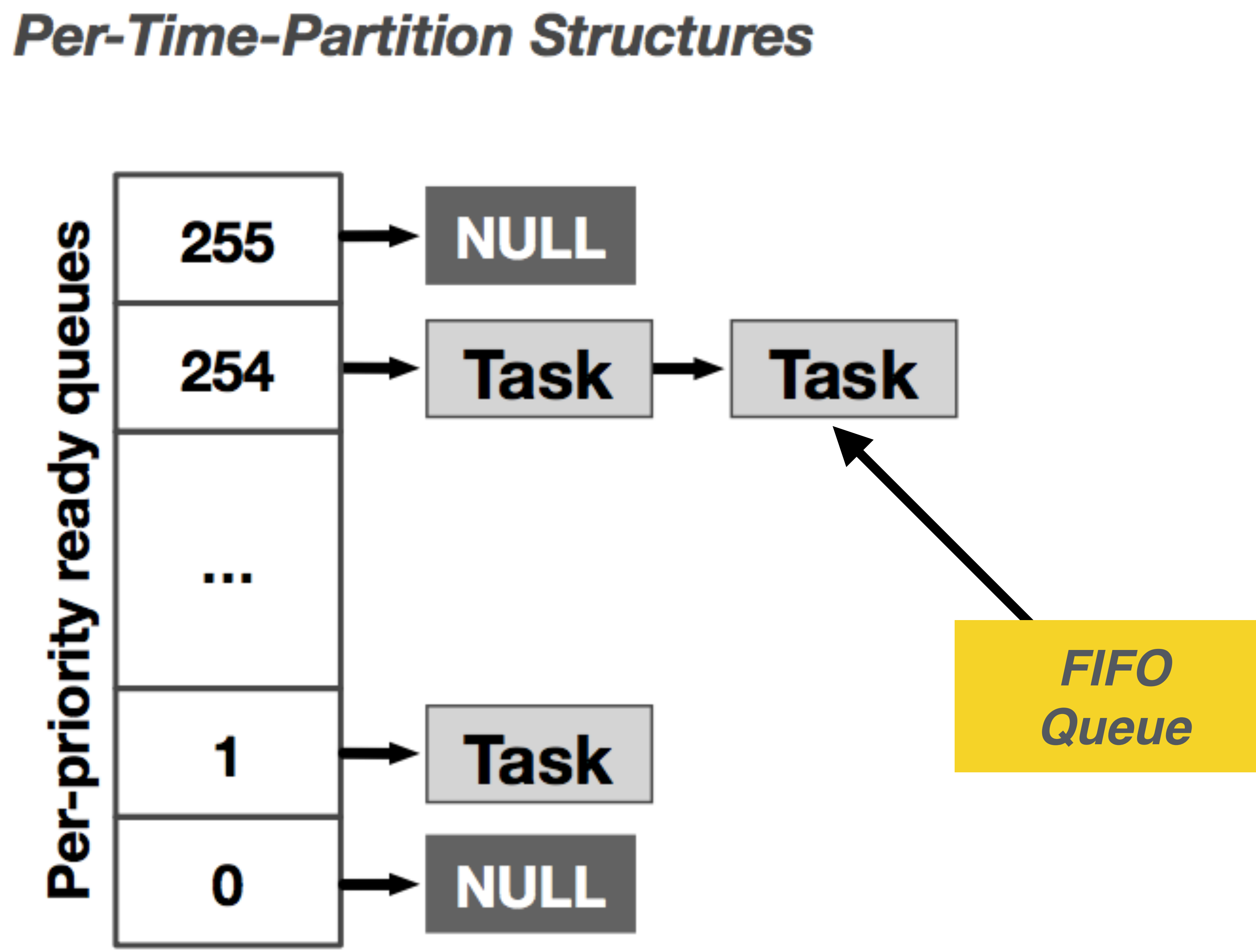
Application Tasks are Assigned to Time Partitions

Time partitioned scheduling

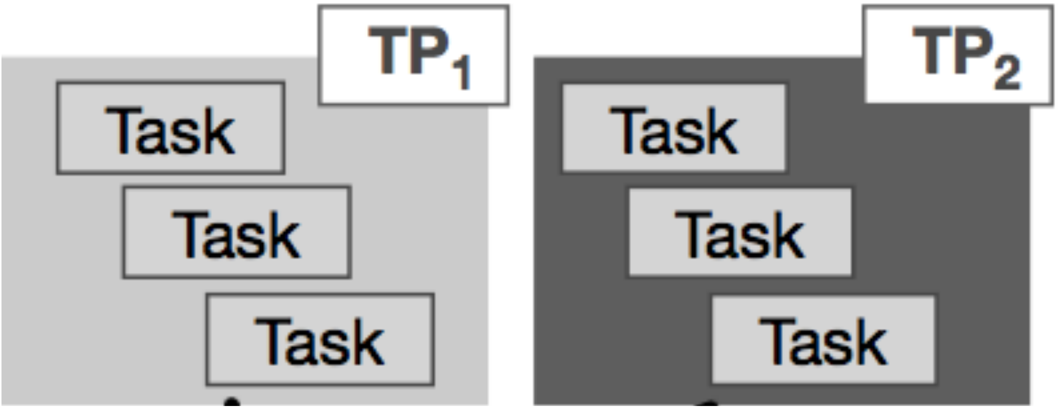


Every task must be certified to the same level of assurance as the highest criticality task it interferes with.

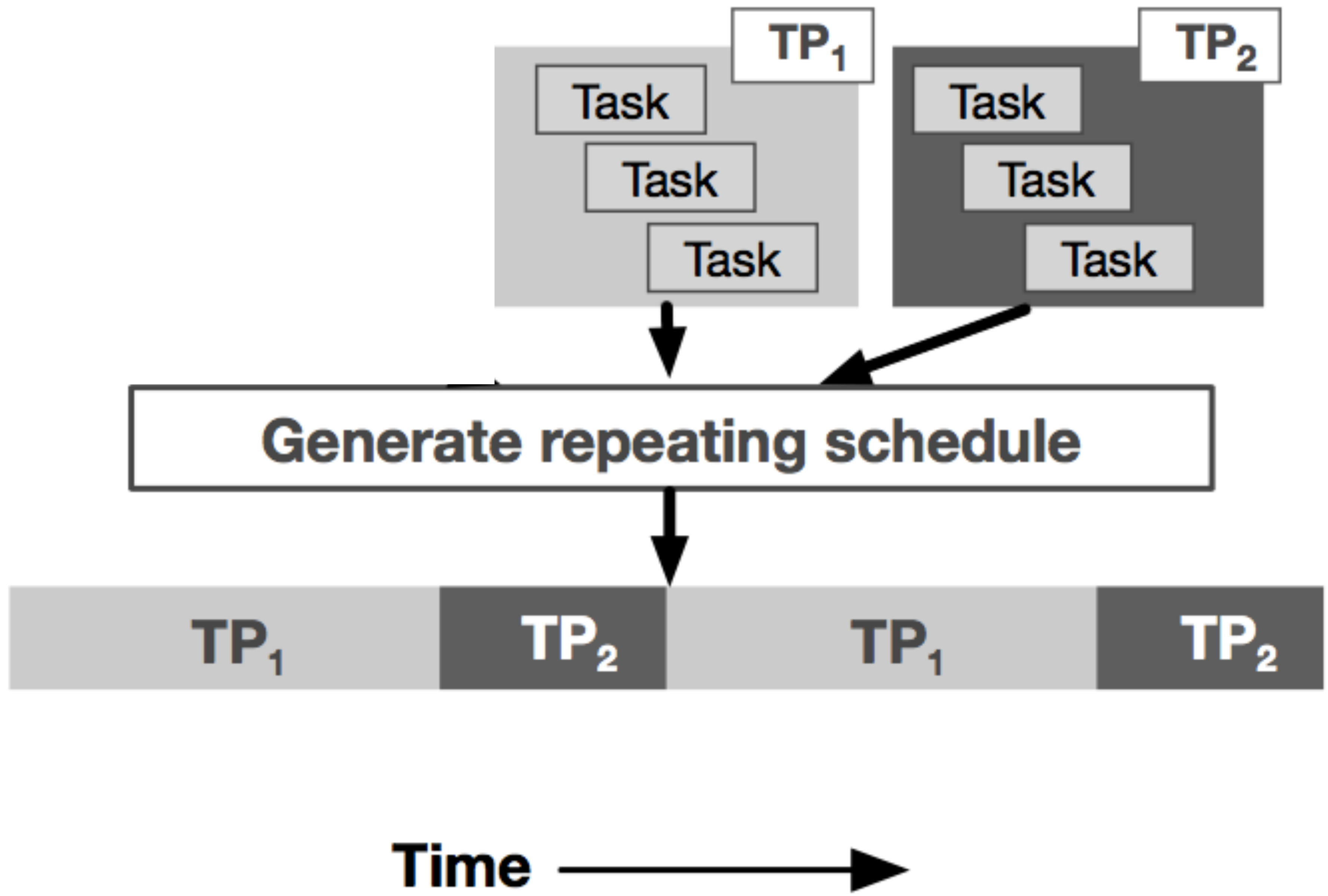
Fixed-Priority Scheduling Within Time Partitions



Application Tasks are Assigned to Time Partitions

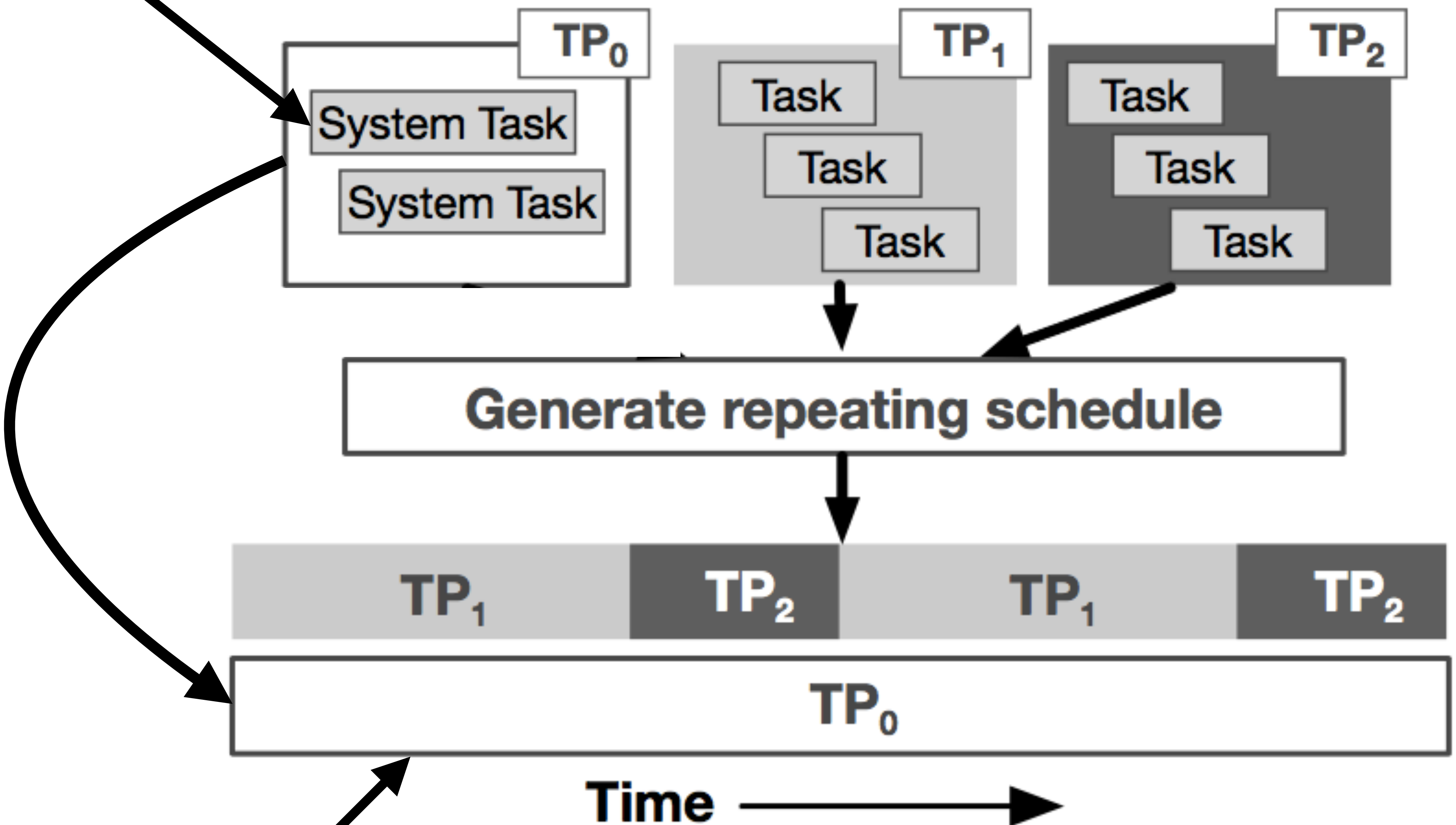


A Static Scheduling Table is Generated



System Tasks are Placed in Always-Eligible TP0

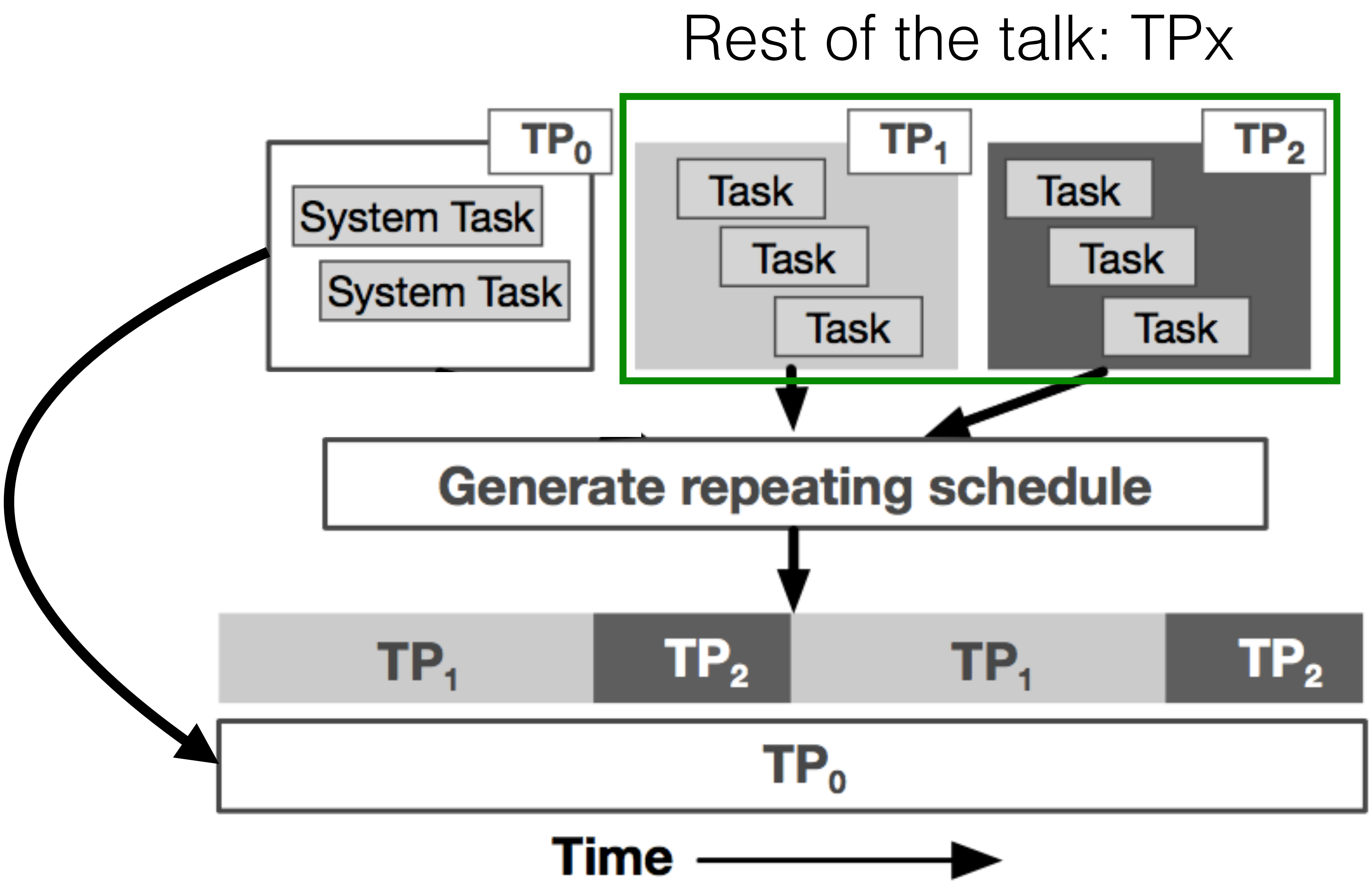
High criticality



Always eligible to run.

TP0 tasks are prioritized over app tasks ***at the same priority****.*

System Tasks are Placed in Always-Eligible TP0



Criticality and Latency Requirements

	Low Latency Required	High Latency Acceptable
High Criticality		
Low Criticality		

High-Criticality Tasks + Low Latency

	Low Latency Required	High Latency Acceptable
High Criticality	TP_0	
Low Criticality		

Examples

Safety-critical event handlers

High-rate, sensor-data retrieval tasks

High-Criticality Tasks + High Latency

	Low Latency Required	High Latency Acceptable
High Criticality	***TP_0***	***TP_0*** or ***high-criticality TP_x***
Low Criticality		

Examples

Computation-heavy, mission-critical planning tasks

High-Criticality Tasks + High Latency

Choice depends on tradeoff between acceptable latency bound and system performance.

	Low Latency Required	High Latency Acceptable
High Criticality	***TP_0***	***TP_0* or *high-criticality TP_x***
Low Criticality		

Examples

Computation-heavy, mission-critical planning tasks

Low-Criticality Tasks + High Latency

	Low Latency Required	High Latency Acceptable
High Criticality	***TP_0***	***TP_0*** or ***high-criticality TP_x***
Low Criticality		***Low-criticality TP_x***

Examples

Navigation or route planning tasks

Low Latency + Low-Criticality Tasks (L3C Tasks)

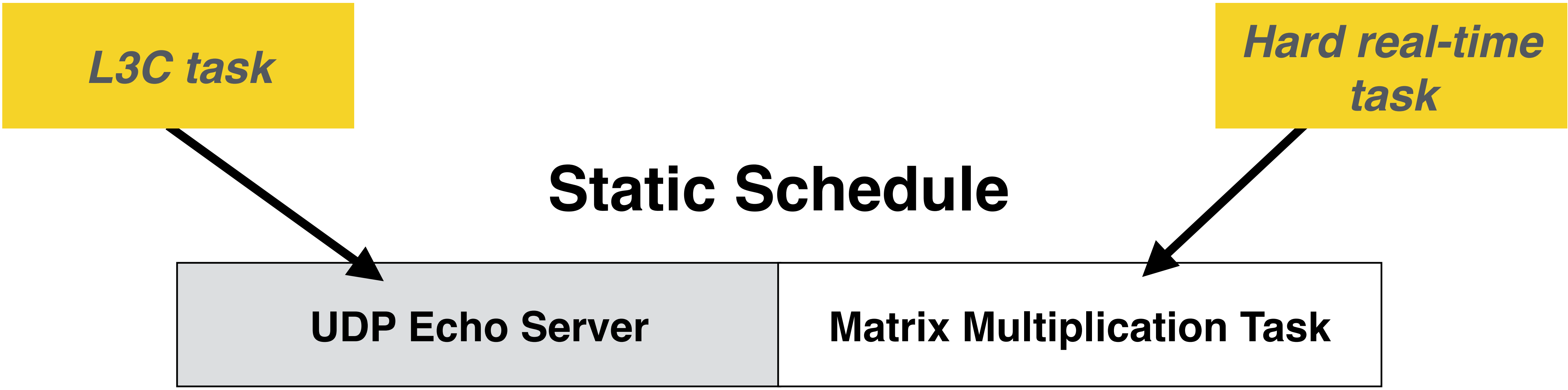
	Low Latency Required	High Latency Acceptable
High Criticality	***TP_0***	***TP_0*** or ***high-criticality TP_x***
Low Criticality	**?**	***Low-criticality TP_x***

L3C Tasks

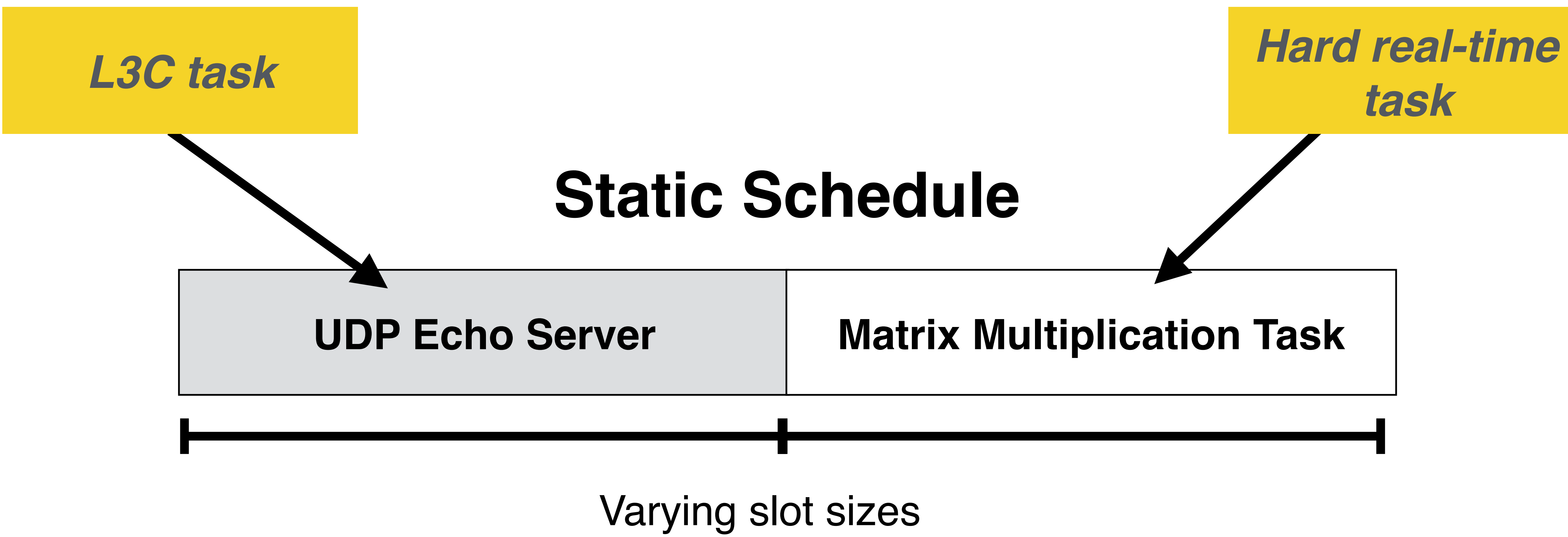
Examples

Low-latency, user interface tasks

Experiment: Latency vs. System Performance



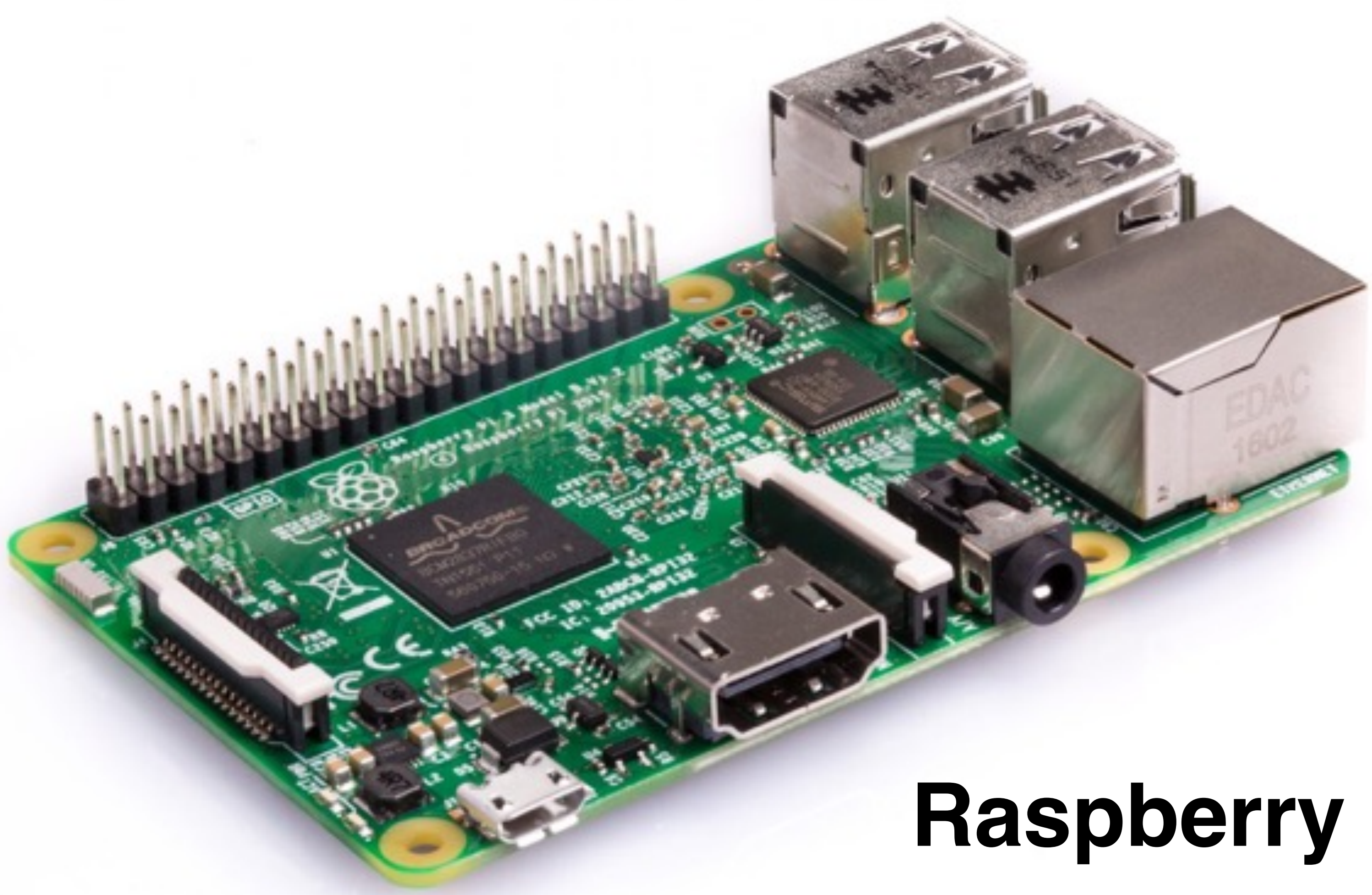
Experiment: Latency vs. System Performance



We measured: ***UDP echo latency***

We measured: ***LLC misses***

Evaluation Setup

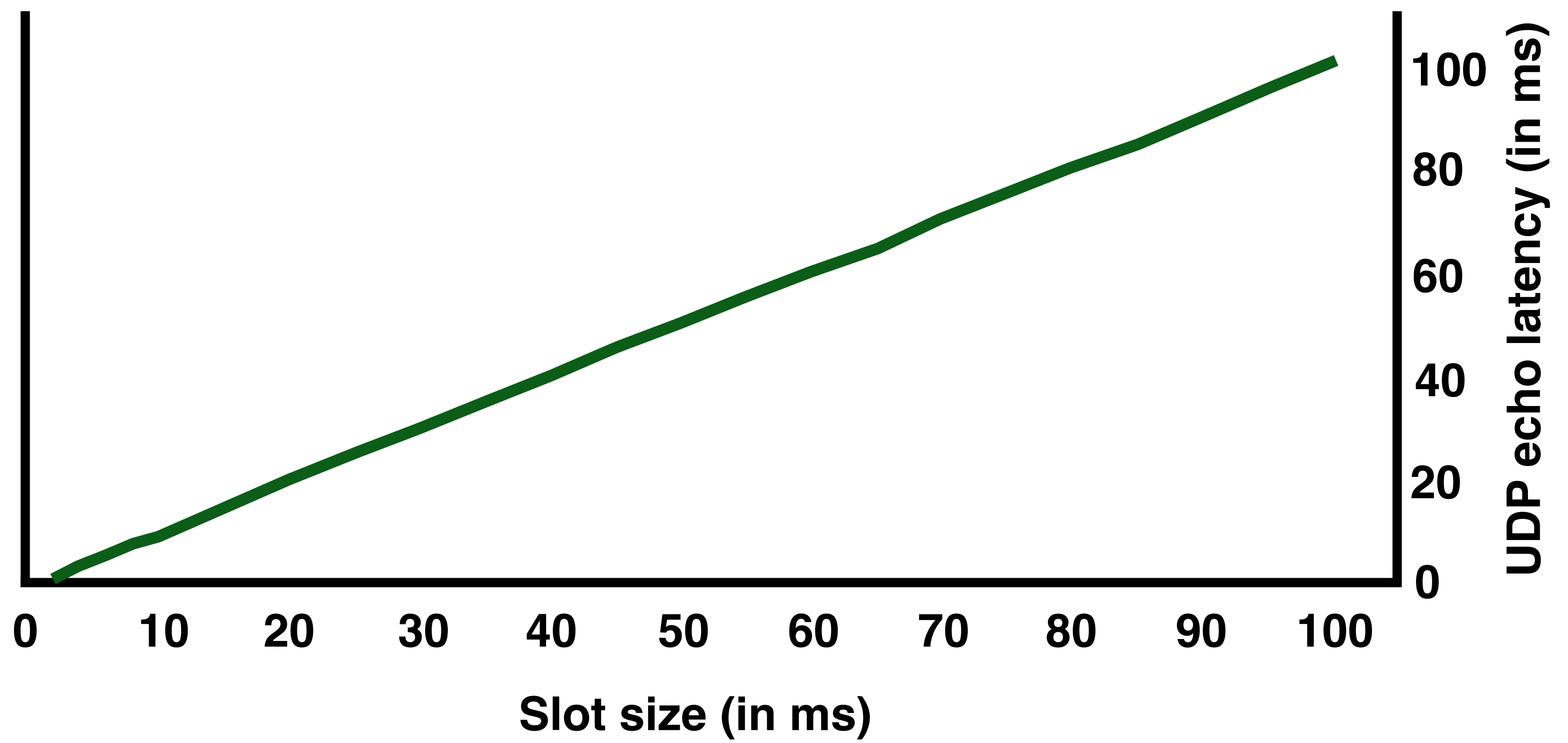


Raspberry Pi 3b

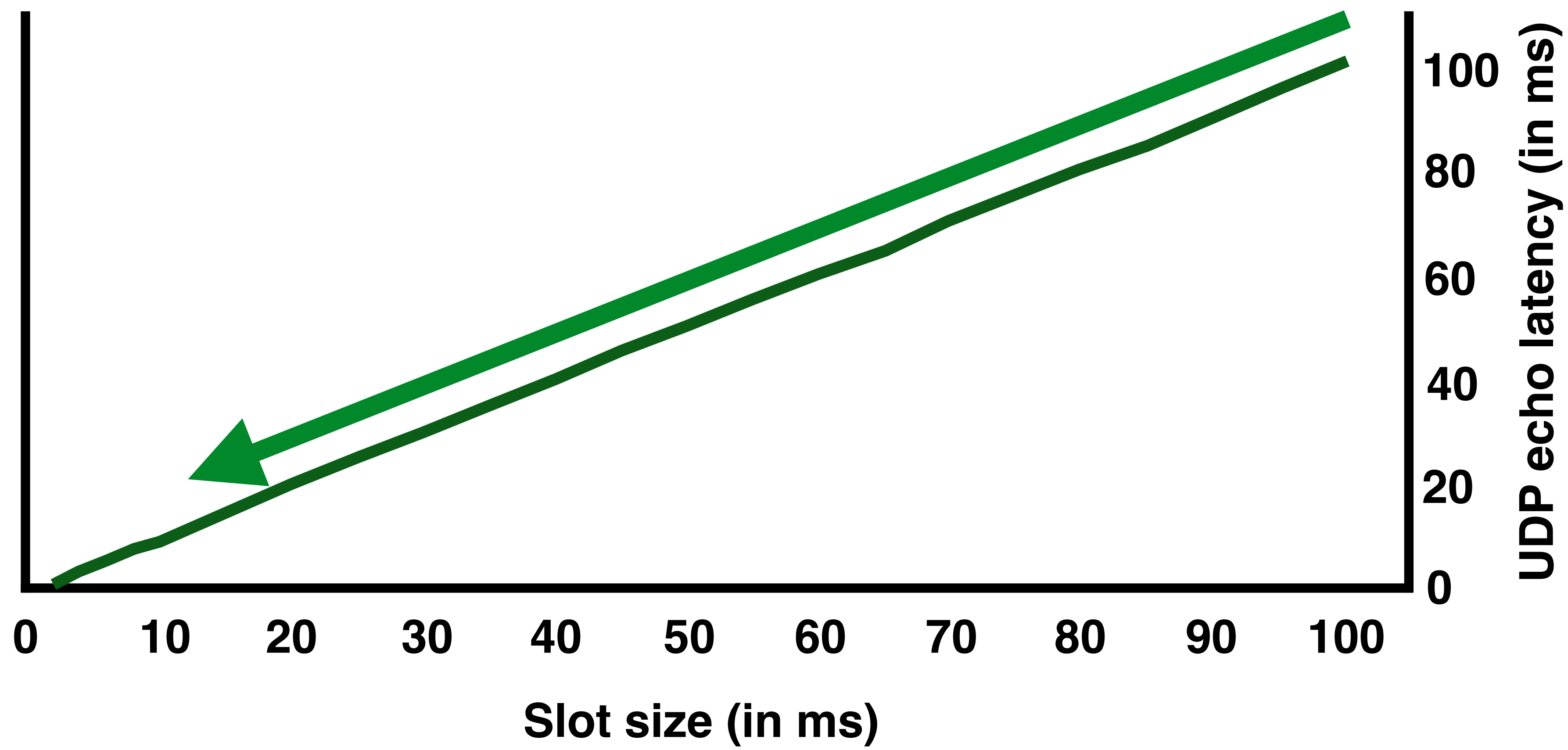
- Broadcom BCM2837.
- 64-bit ARM Cortex-A53 @ 1.2GHz.
- 1GB RAM

LITMUS^{RT} 2017.1 with Linux 4.9.30

Experiment: Latency vs. System Performance



Experiment: Latency vs. System Performance



UDP echo latency reduces with smaller slot size

Tradeoff: Latency vs. System Performance

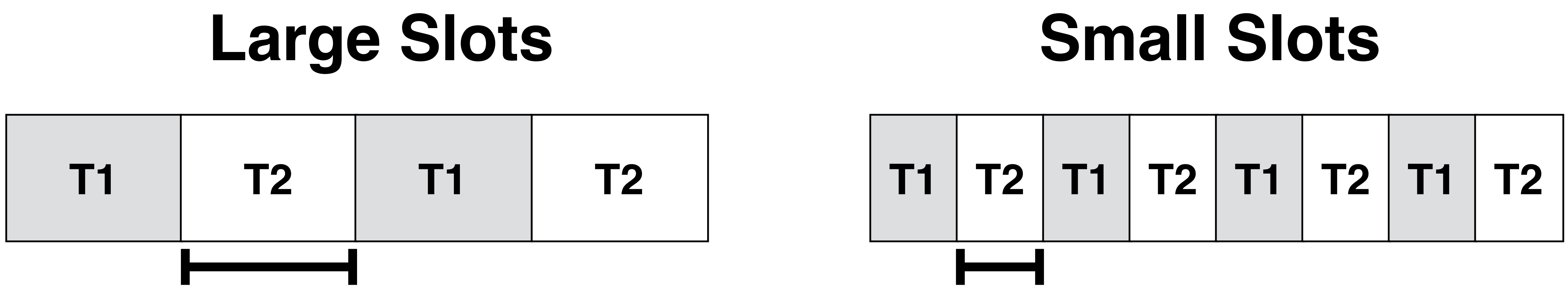
Large Slots

T1	T2	T1	T2

Small Slots

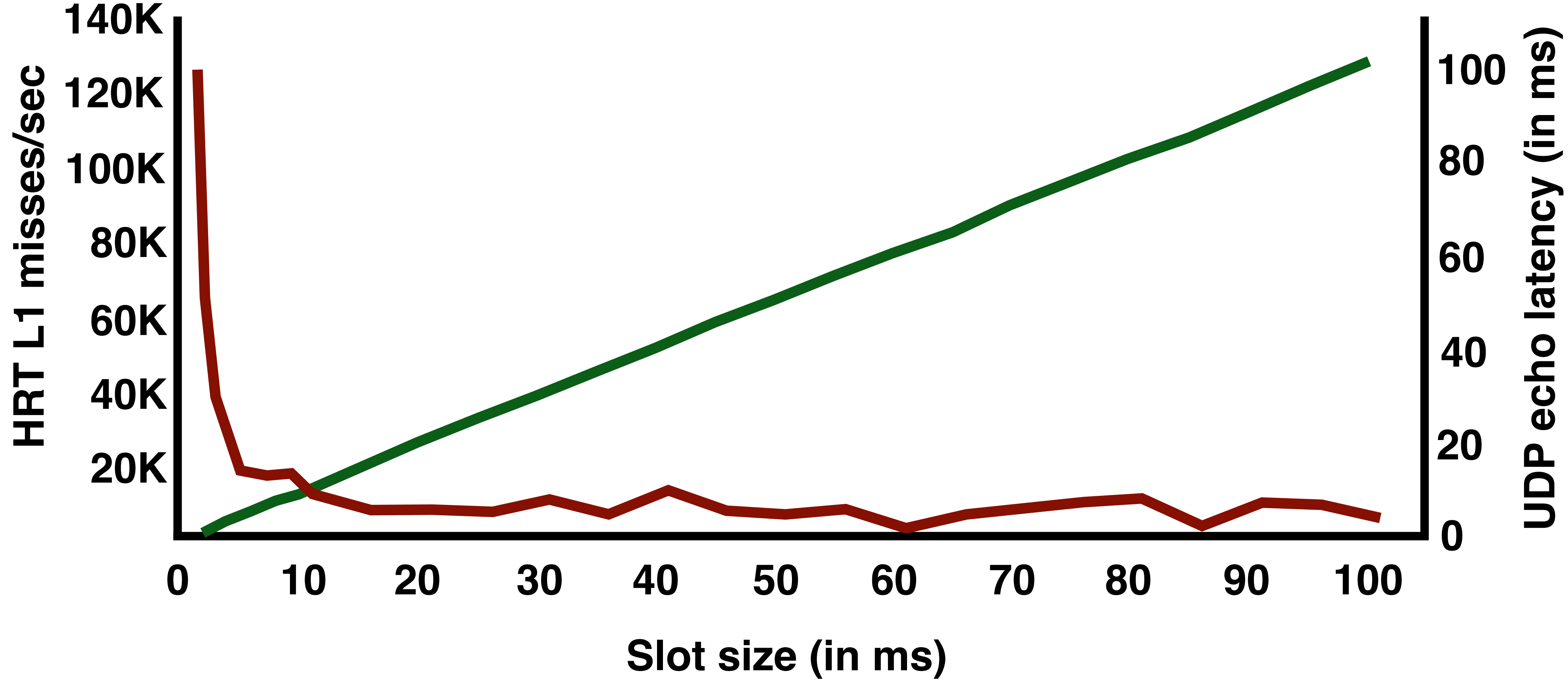
T1	T2	T1	T2	T1	T2	T1	T2

Tradeoff: Latency vs. System Performance

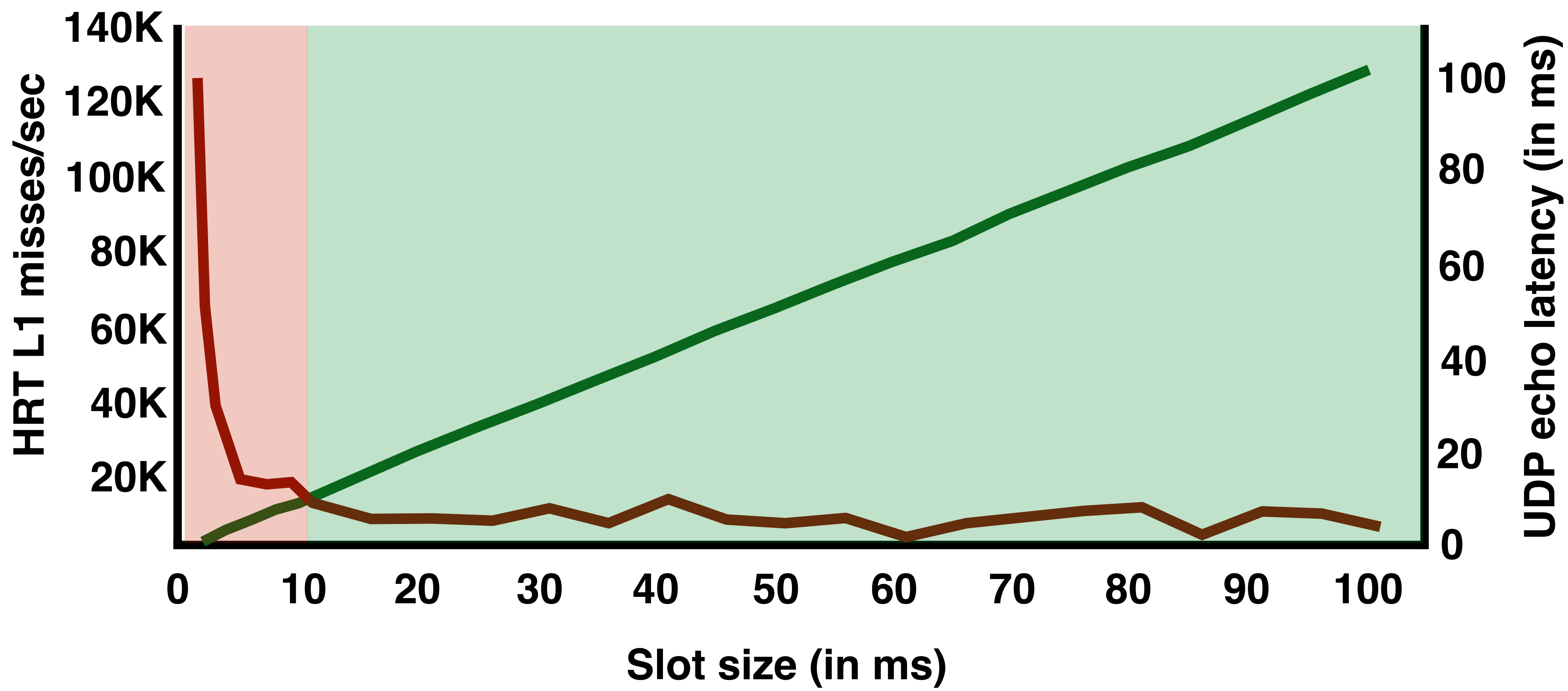


Latency incurred by tasks reduces with smaller slot sizes

Experiment: Latency vs. System Performance

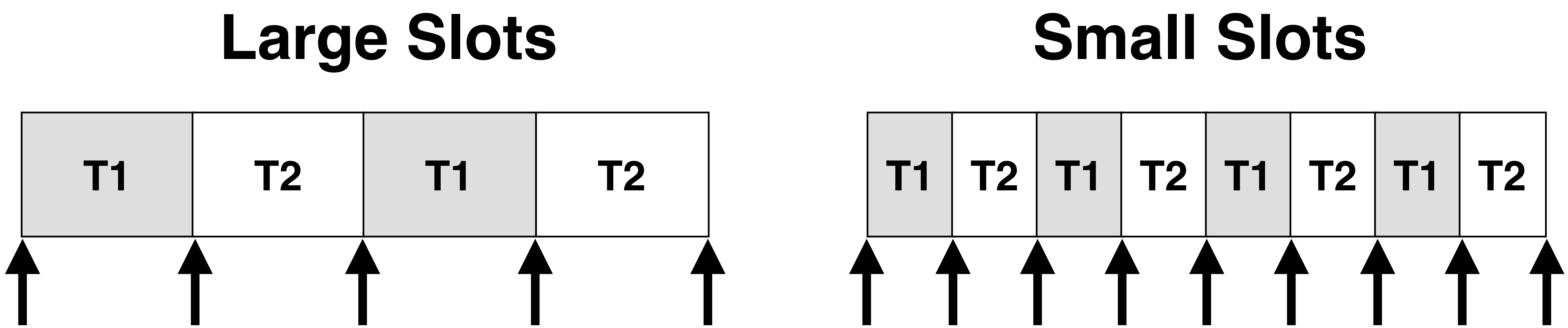


Experiment: Latency vs. System Performance



L3C tasks requiring < 10ms latency **cannot be placed within a TPx** without affecting the rest of the system.

Tradeoff: Latency vs. System Performance



Number of preemptions increases with smaller slot sizes
(loss of cache affinity, more scheduler invocations)

Other Possibilities?

Other Possibilities?

L3C tasks **cannot be placed at a high priority in TP0** without causing potentially unbounded interference on high-criticality tasks

Other Possibilities?

L3C tasks **cannot be placed at a high priority in TP0** without causing potentially unbounded interference on high-criticality tasks

L3C tasks **cannot be placed at a low priority in TP0** (criticality-monotic priority assignment) without itself incurring significant interference.

Rest of this Talk

I ***The problem of low-criticality, low-latency tasks***

II ***Working within real-world design constraints***

III ***Our proposed scheduler extensions***

Design Constraints

We present nine key design constraints in the paper!

Design Constraints

We present nine key design constraints in the paper!

- Strictly opt-in for OS customers.
- Minimally intrusive for the OS vendor.
- Optionally strict freedom-from-interference.

Design Constraints

We present nine key design constraints in the paper!

- **Strictly opt-in for OS customers.**
- Minimally intrusive for the OS vendor.
- Optionally strict freedom-from-interference.

Strictly Opt-In for Customers

Customers bear significant costs

Design and implementation of product components.

Configuration, testing, and certification of production systems.

Adoption of specific **workflows and tools**.

Strictly Opt-In for Customers

Customers bear significant costs

Design and implementation of product components.

Configuration, testing, and certification of production systems.

Adoption of specific **workflows and tools**.

By default, customers should require no changes. All L3C support features **should be strictly opt-in**.

Strictly Opt-In for Customers

Customers bear significant costs

Design and implementation of product components.

Configuration, testing, and certification of production systems.

Adoption of specific **workflows and tools**.

By default, customers should require no changes. All L3C support features **should be strictly opt-in**.

It should be **easy to opt-in incrementally** to new L3C-support features. That is, **without significant changes to designs and workflows.**

Design Constraints

- Strictly opt-in for OS customers.
- **Minimally intrusive for the OS vendor.**
- Optionally strict freedom-from-interference.

Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

Documentation efforts for certification process.

Adoption of specific **workflows and tools**.

Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

Documentation efforts for certification process.

Adoption of specific **workflows and tools**.

Cannot radically re-design core parts of the system triggering an expensive re-certification process.

Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

Documentation efforts for certification process.

Adoption of specific **workflows and tools**.

Cannot radically re-design core parts of the system triggering an expensive re-certification process.

Need to provide continued legacy support for **existing customers** with products out in the market.

Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

Documentation efforts for certification process.

Adoption of specific **workflows and tools**.

Cannot radically re-design core parts of the system triggering an expensive re-certification process.

Need to provide continued legacy support for **existing customers** with products out in the market.

Any introduced changes **must minimize re-certification costs** if they hope to be adopted.

Design Constraints

- Strictly opt-in for OS customers.
- Minimally intrusive for the OS vendor.
- **Optionally strict freedom-from-interference.**

Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bounds are known and enforced by OS (and enforcement mechanism validated at a high level of assurance).

Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bounds are known and enforced by OS (and enforcement mechanism validated at a high level of assurance).

For certain maximum-importance tasks, it must be possible to achieve **strict freedom from (all) interference**.

Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bounds are known and enforced by OS (and enforcement mechanism validated at a high level of assurance).

For certain maximum-importance tasks, it must be possible to achieve **strict freedom from (all) interference**.

Even when opting in to L3C support features, for certain tasks, it **should be possible to selectively opt-out.**

Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bou… d at

Example
EDF-VD requires all tasks in the system to be under the EDF-VD scheduler.

For certain maximum-importance tasks, it must be possible to achieve **strict freedom from (all) interference**.

Even when opting in to L3C support features, for certain tasks, it **should be possible to selectively opt-out.**

Rest of this Talk

I *The problem of low-criticality, low-latency tasks*

II *Working within real-world design constraints*

III *Our proposed scheduler extensions*

Key Problem

Ideally: want to place L3C tasks at a **high priority in TP0**

Key Problem

Ideally: want to place L3C tasks at a **high priority in TP0**

Tasks achieve low latency
(TP0 always eligible to run, immune even from higher-criticality interference)

Key Problem

Ideally: want to place L3C tasks at a **high priority in TP0**

Tasks achieve low latency
(TP0 always eligible to run, immune even from higher-criticality interference)

Can cause **potentially unbounded interference** on high-criticality tasks (both in TP0 and other TPx's)

Key Problem

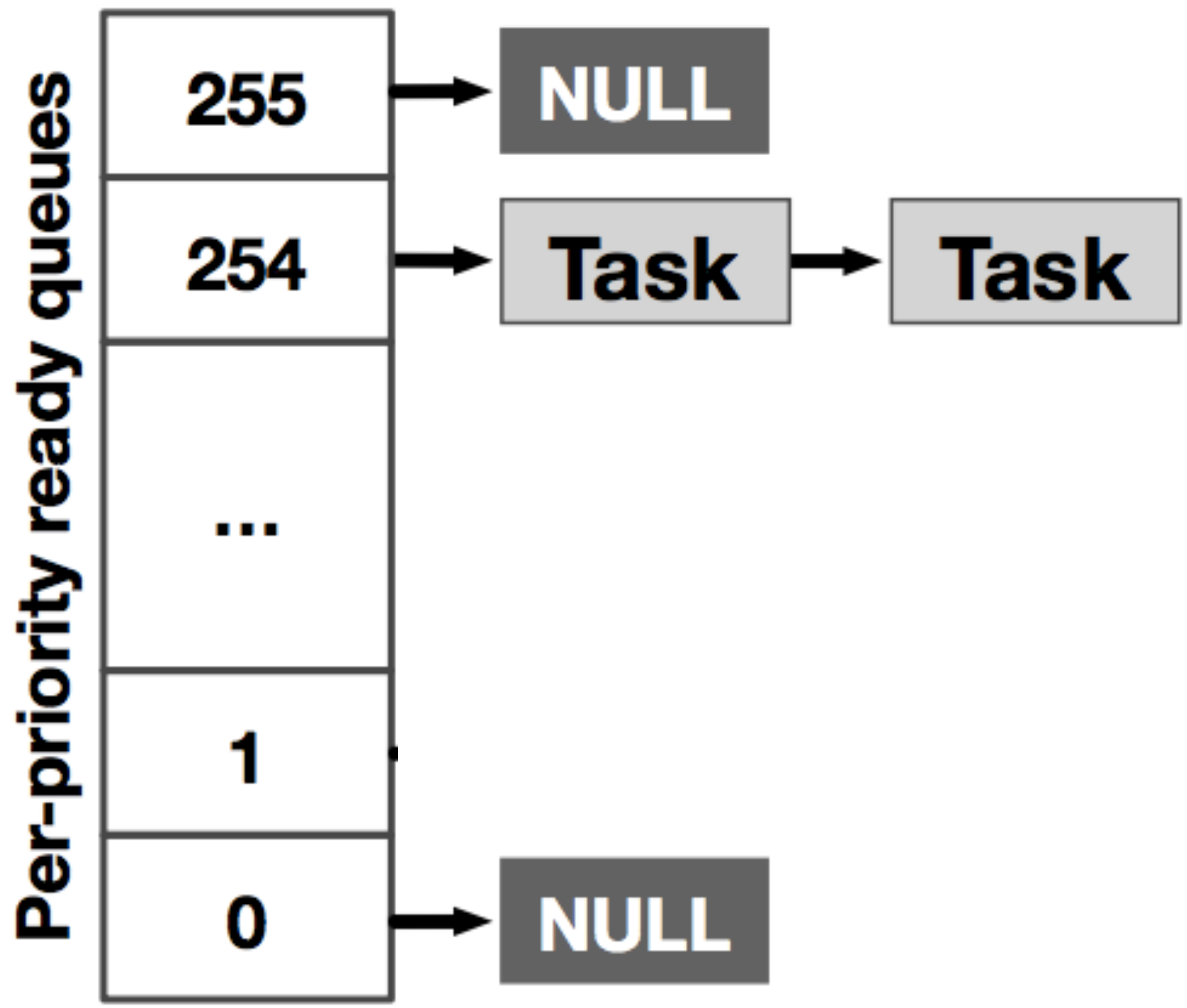
Ideally: want to place L3C tasks at a **high priority in TP0**

Tasks achieve low latency
(TP0 always eligible to run, immune even from higher-criticality interference)

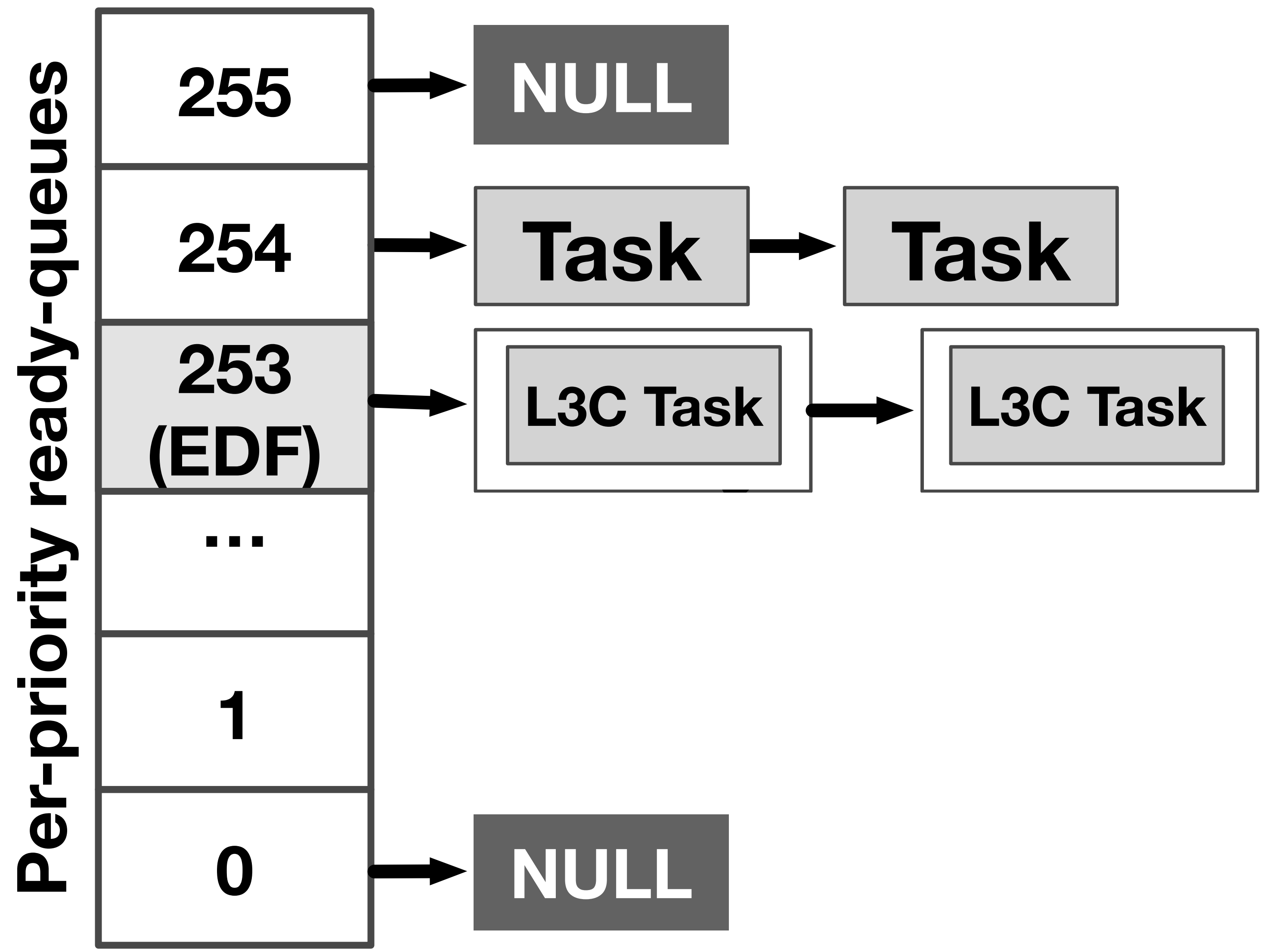
Can cause **potentially unbounded interference** on high-criticality tasks (both in TP0 and other TPx's)

Solution: bound interference from L3C tasks via reservations.

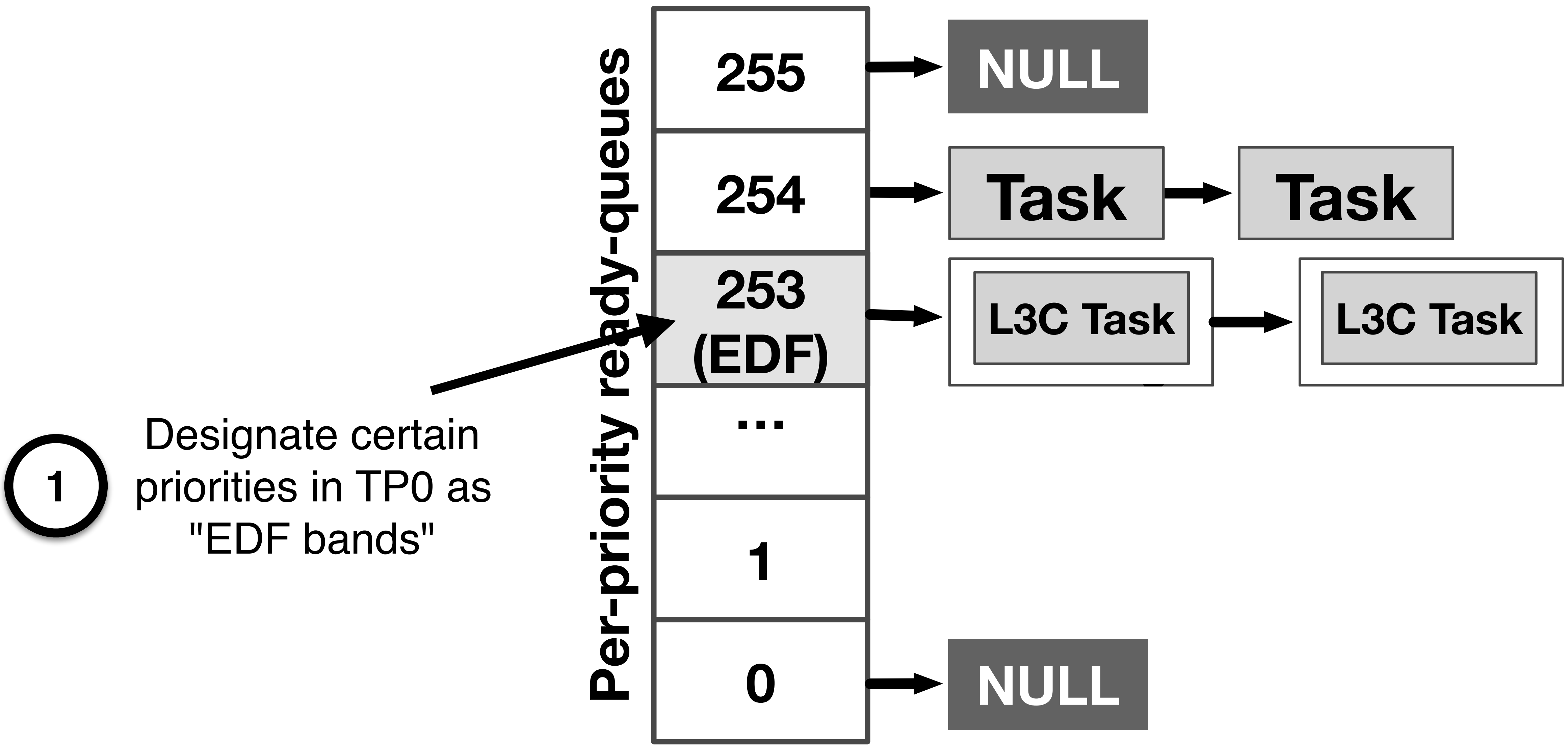
Fixed Priorities in TP0



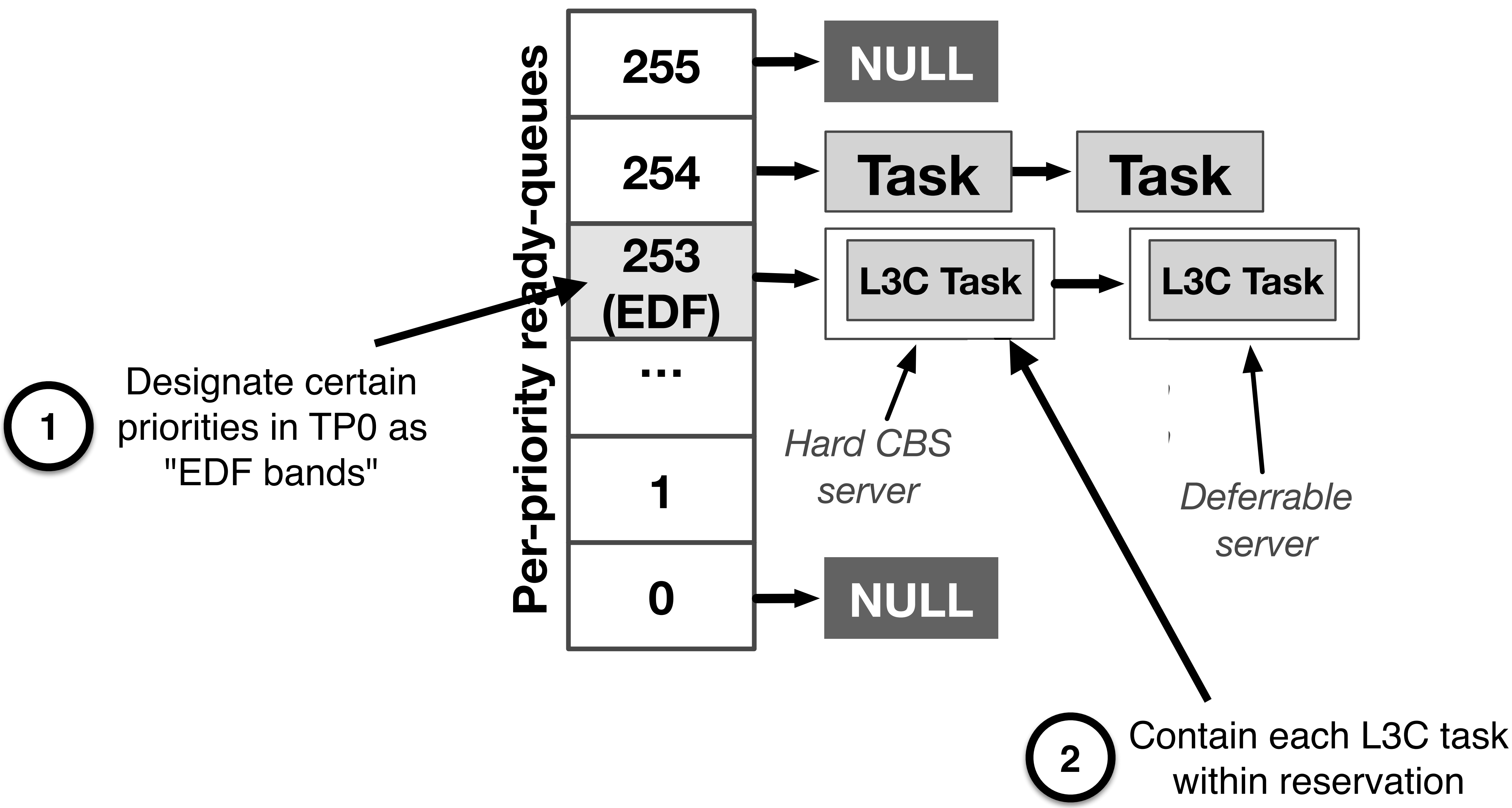
Adopted Solution



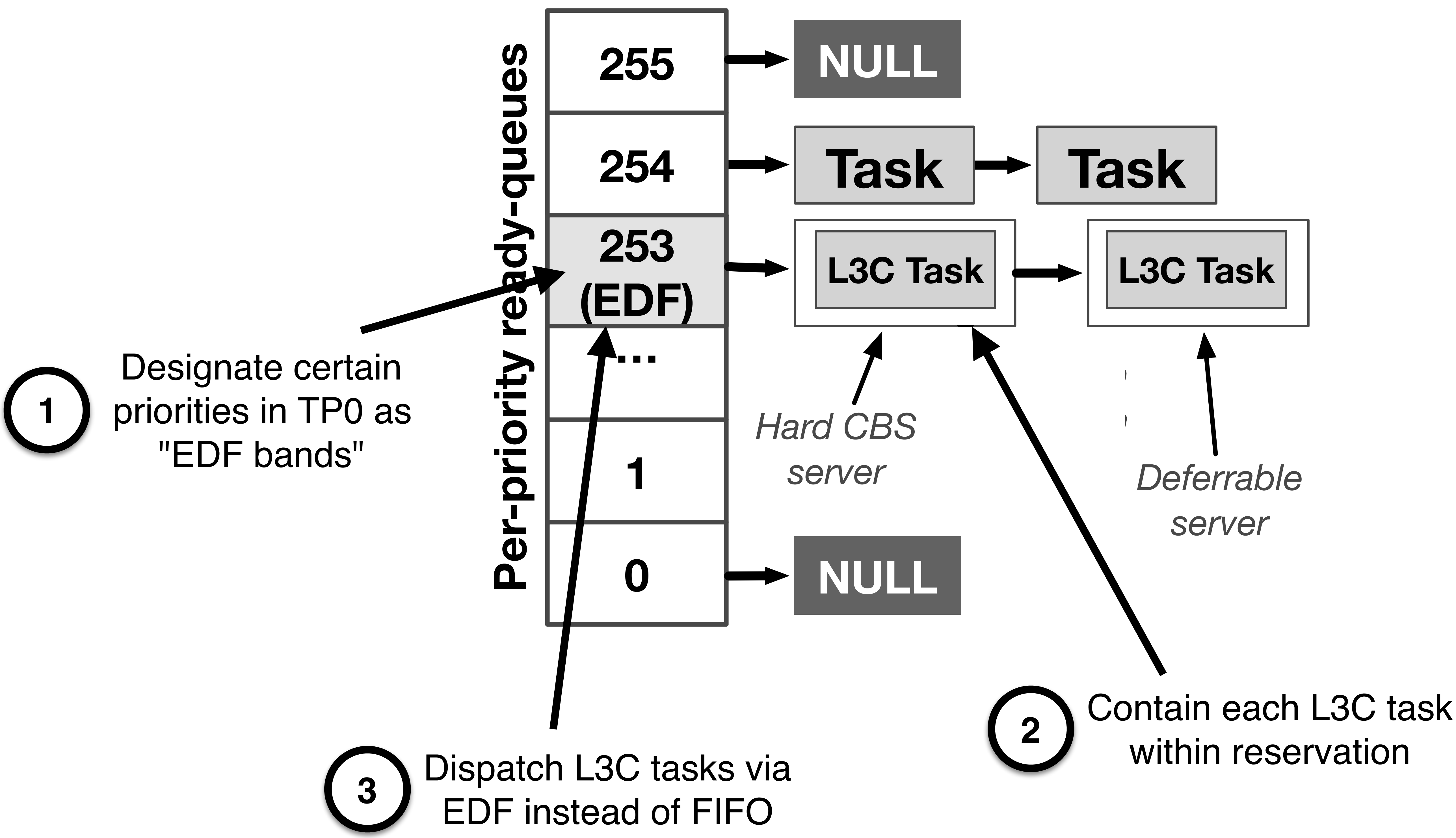
Adopted Solution



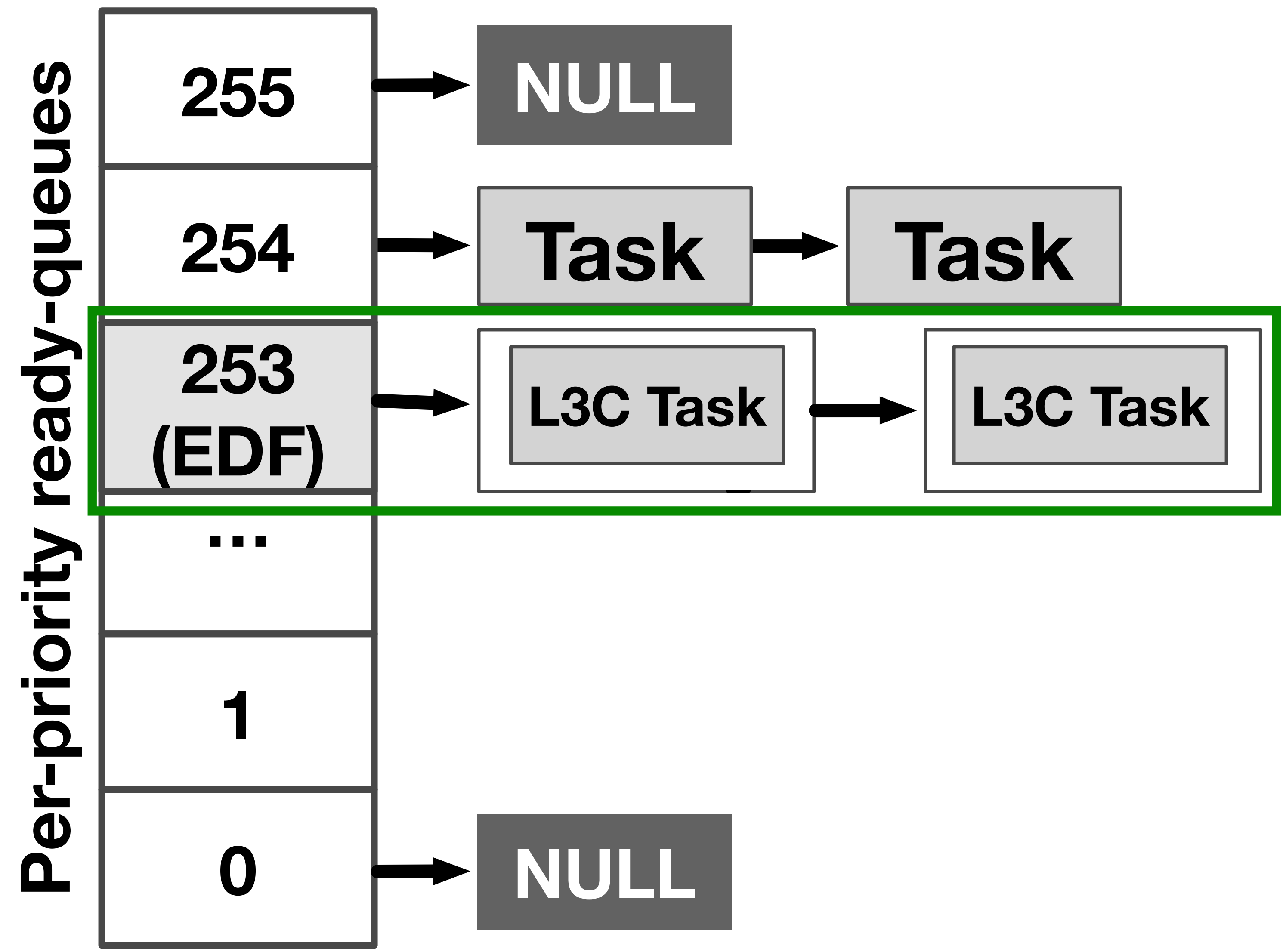
Adopted Solution



Adopted Solution



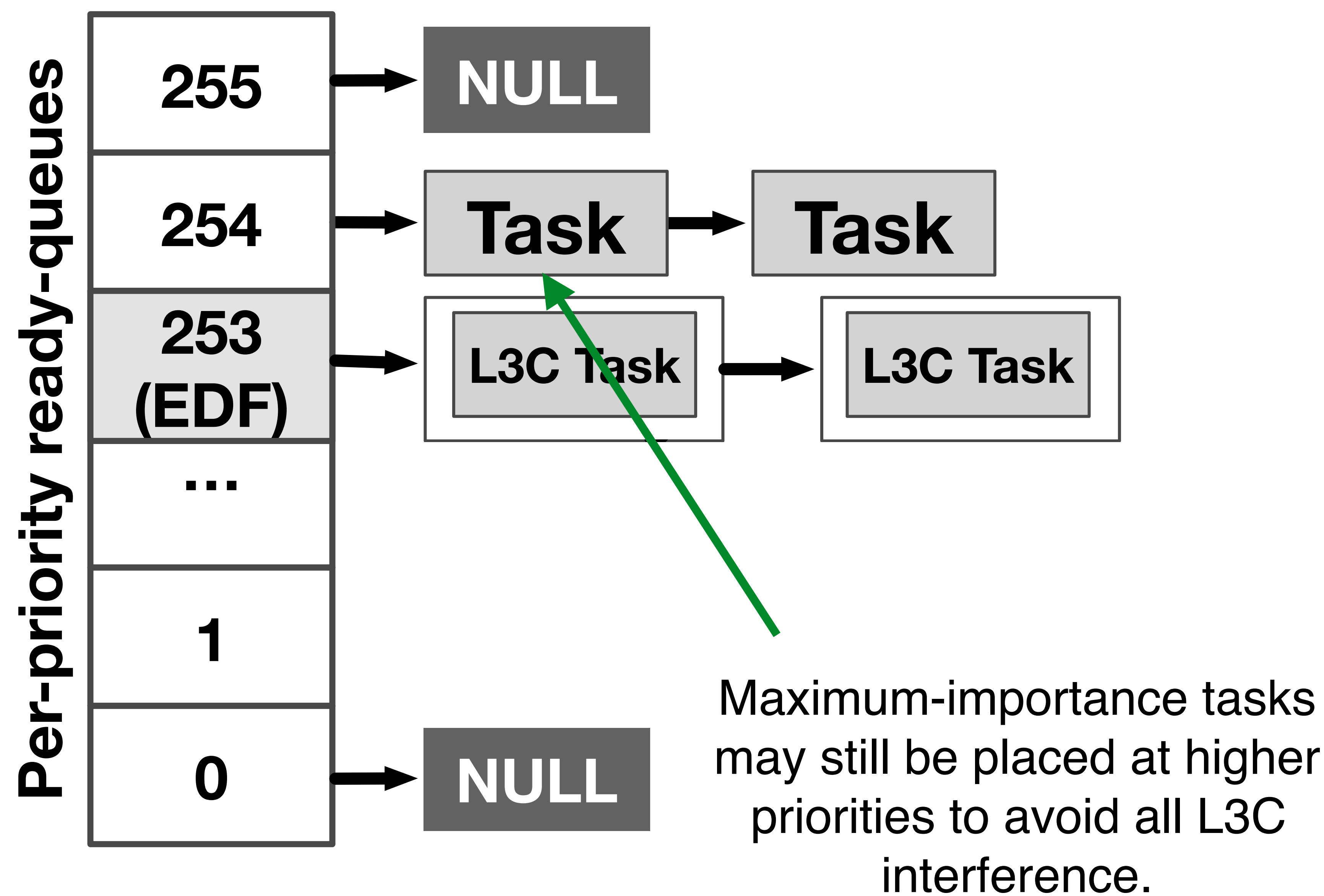
Adherence to Design Constraints



Strictly opt-in for OS customers

Designation of priorities in TP0 as EDF bands is strictly optional.

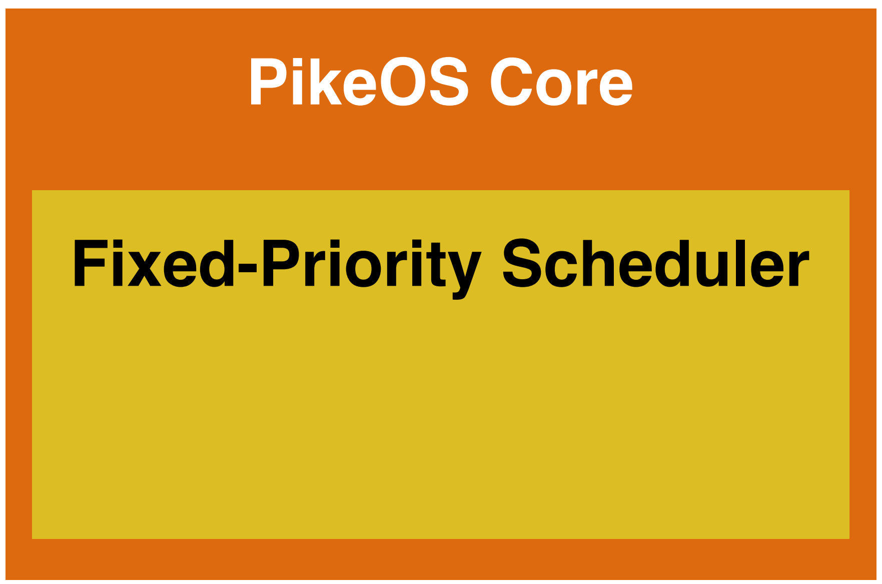
Adherence to Design Constraints



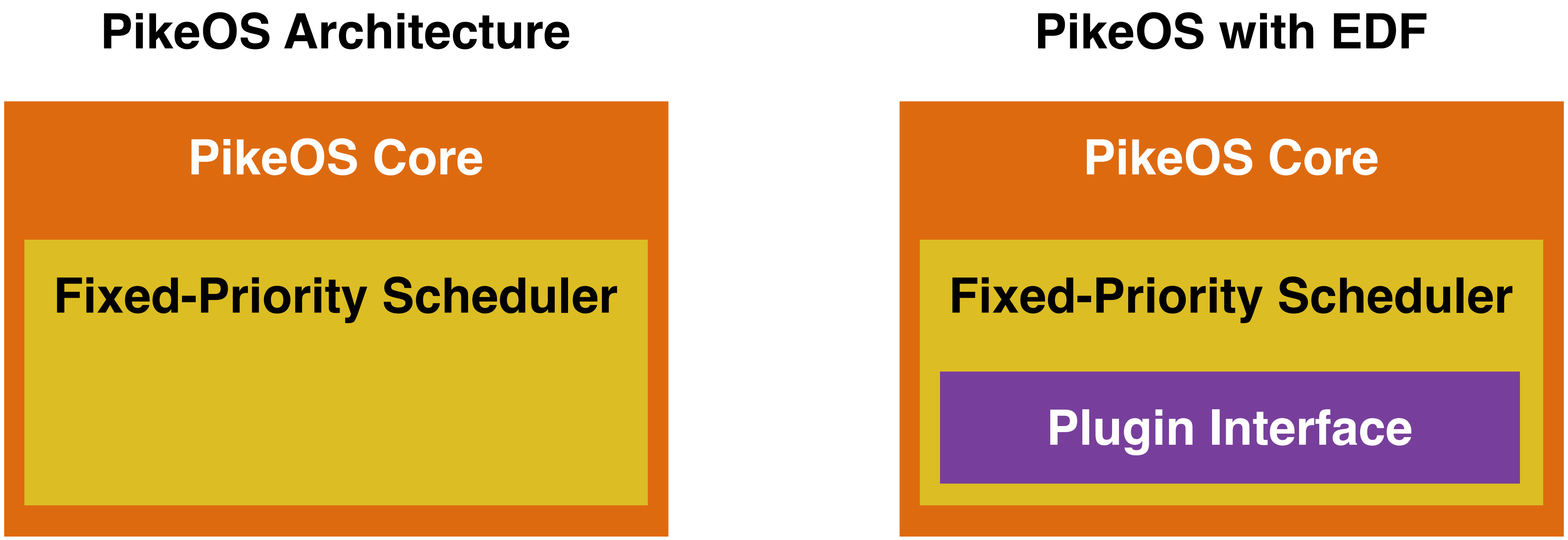
Strict freedom-from-interference from L3C tasks.

Adherence to Design Constraints

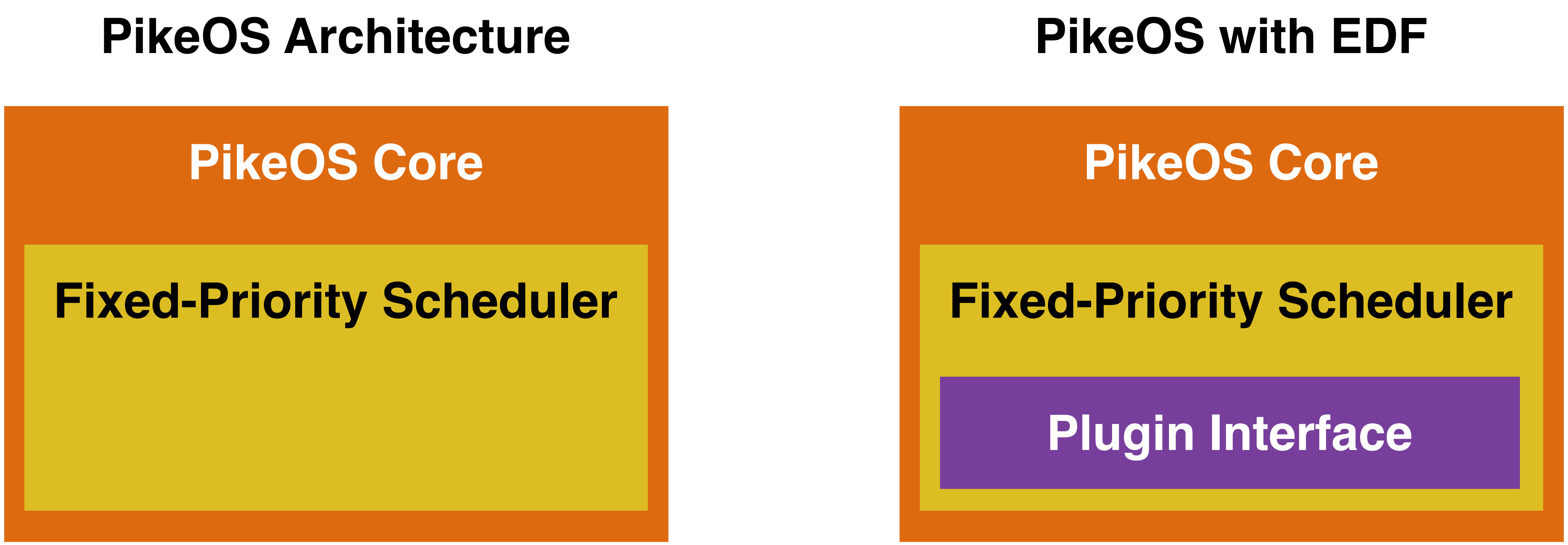
PikeOS Architecture



Adherence to Design Constraints



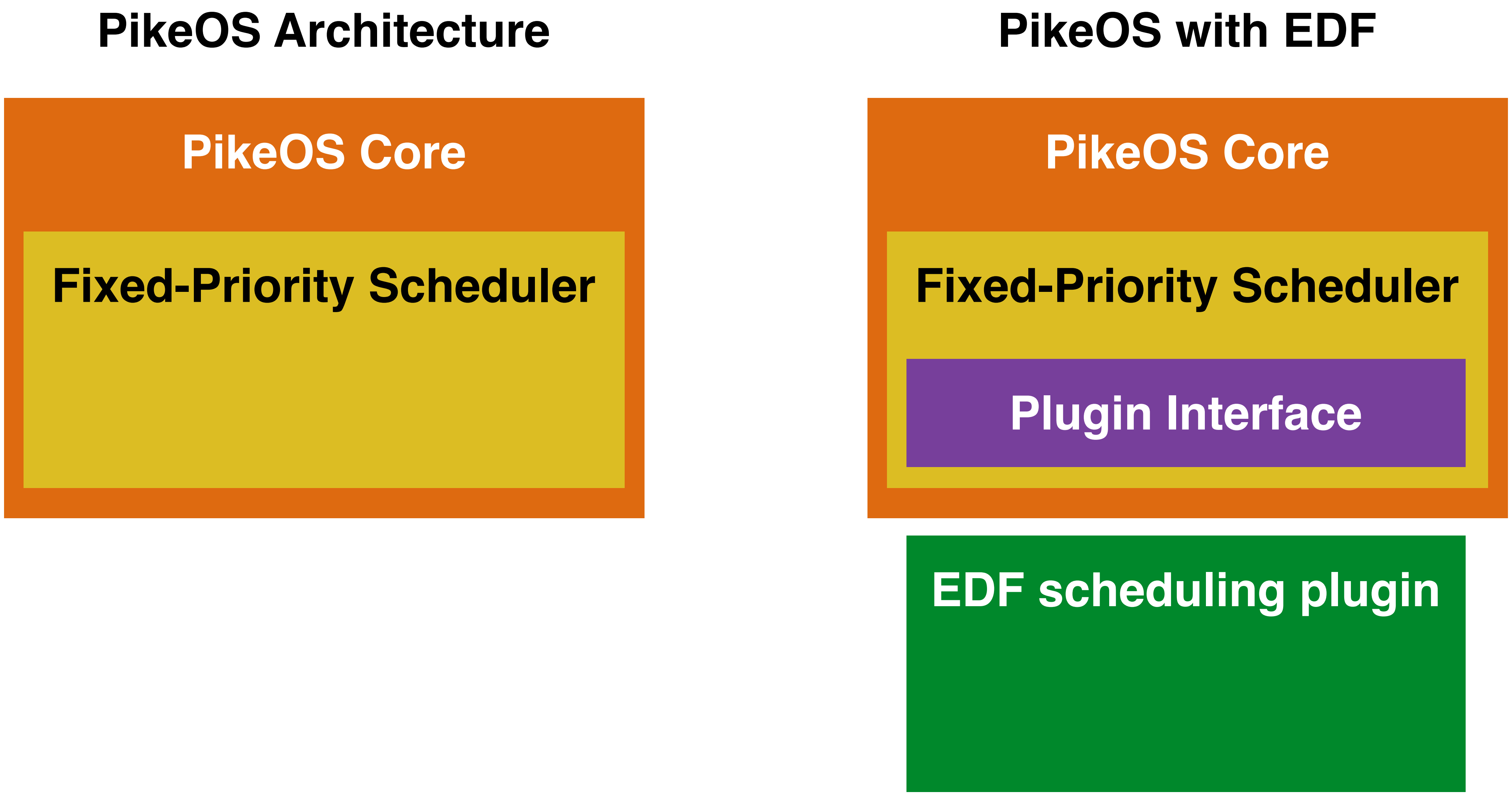
Adherence to Design Constraints



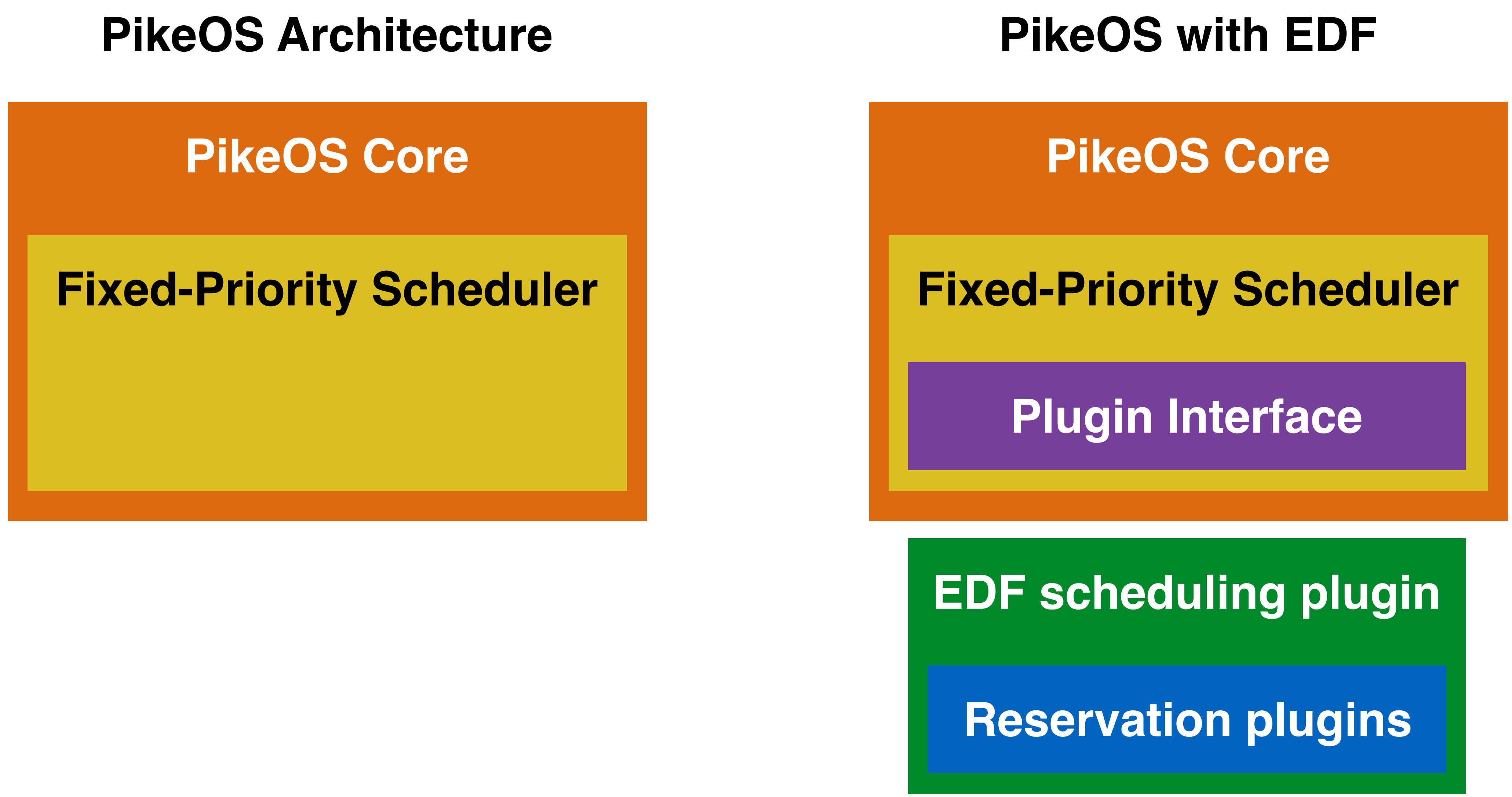
Minimally intrusive for OS vendors.

Can certify the plugin interface as part of the core kernel and amortize the cost across multiple customers.

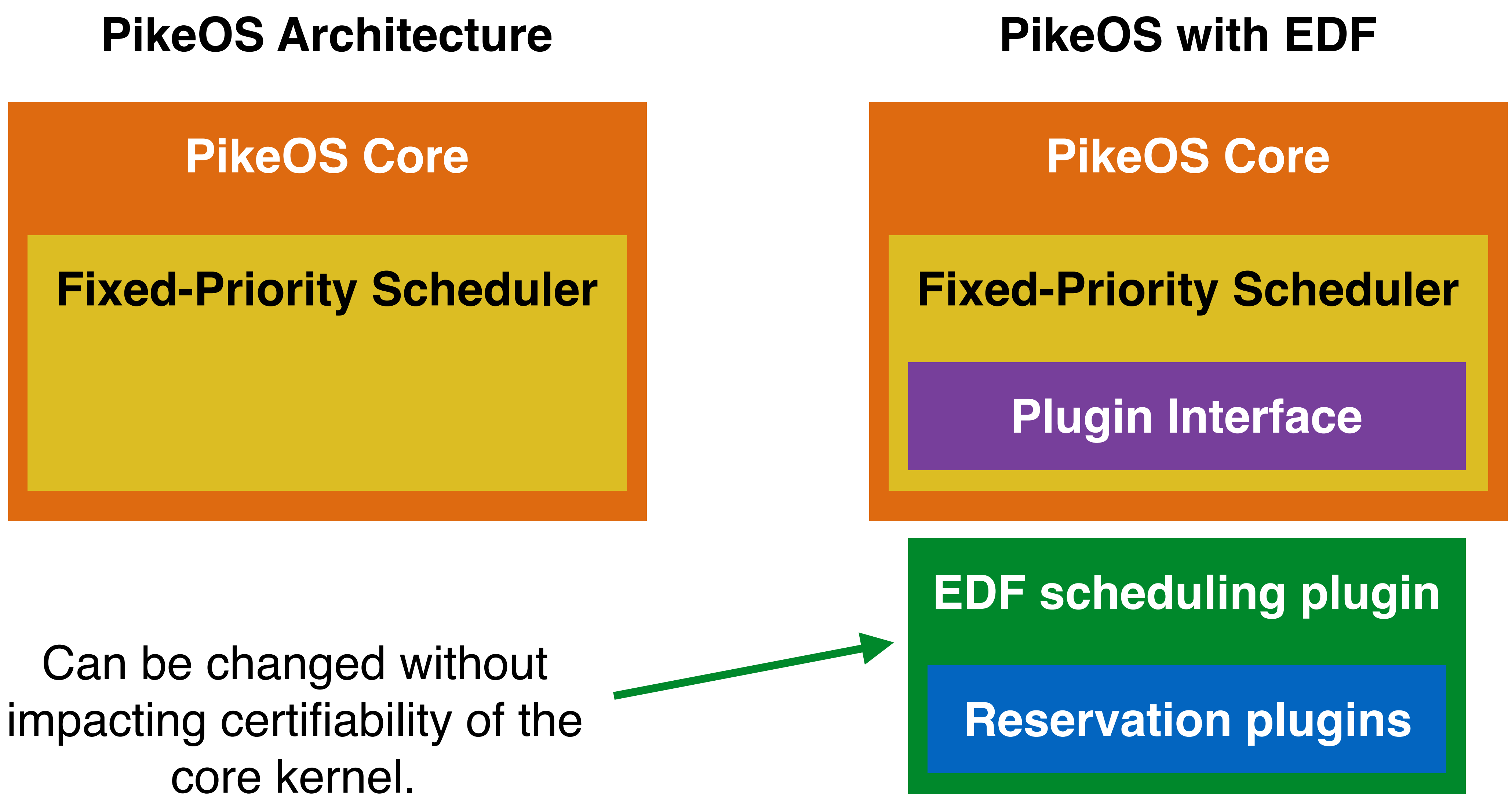
Adherence to Design Constraints



Adherence to Design Constraints



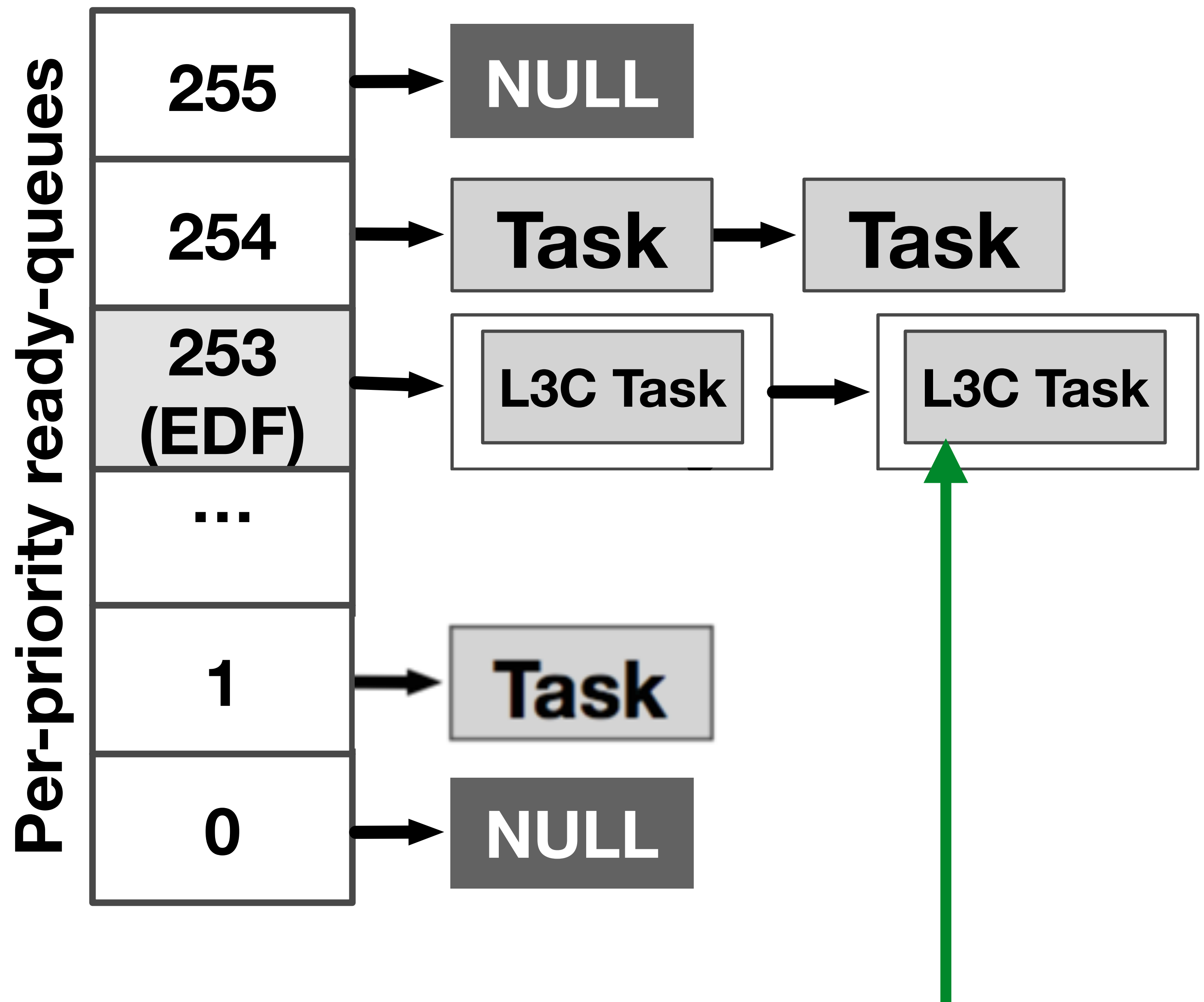
Adherence to Design Constraints



Can be changed without impacting certifiability of the core kernel.

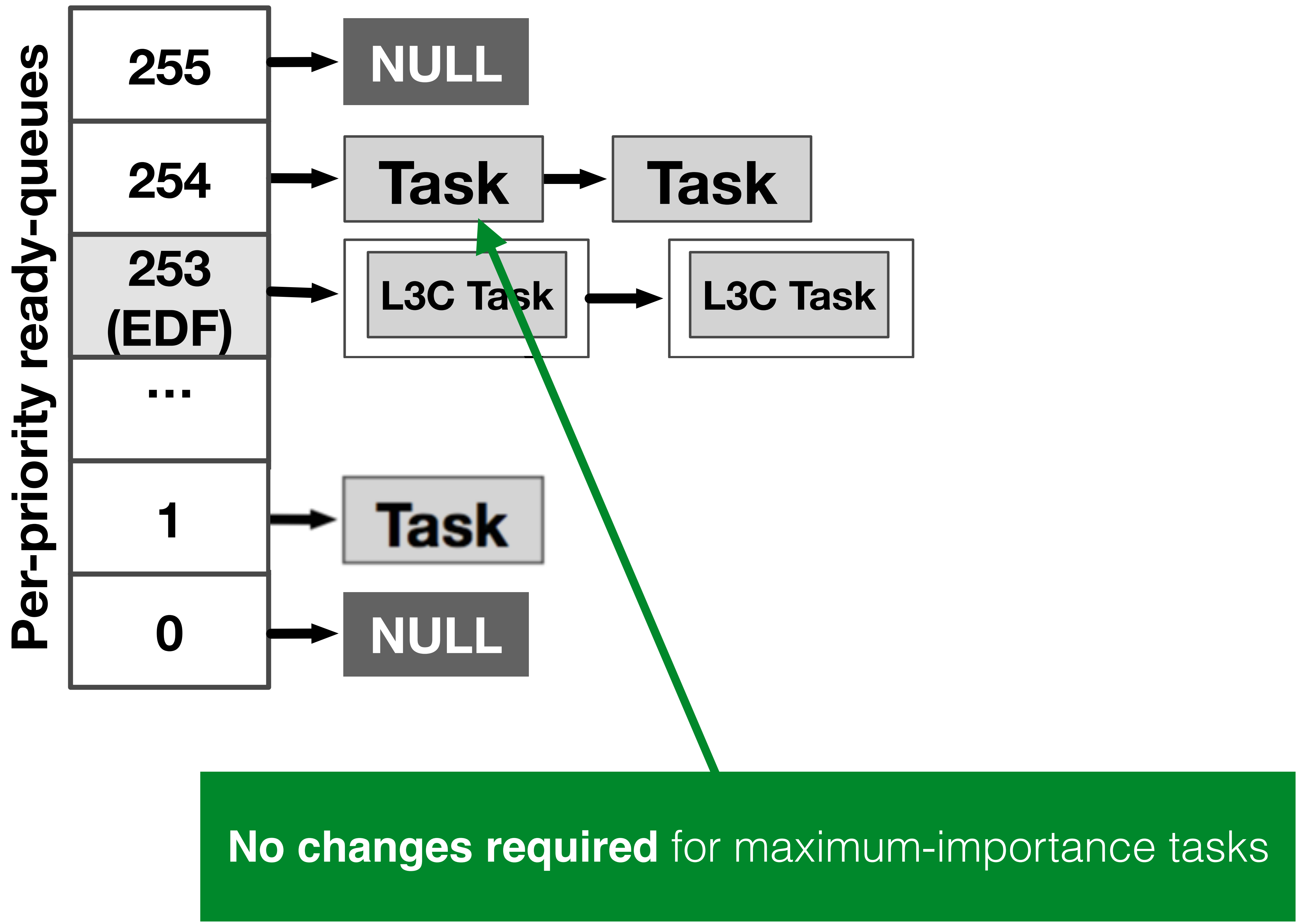
Can even be implemented in a separate address space or as a user-space thread.

System Configuration

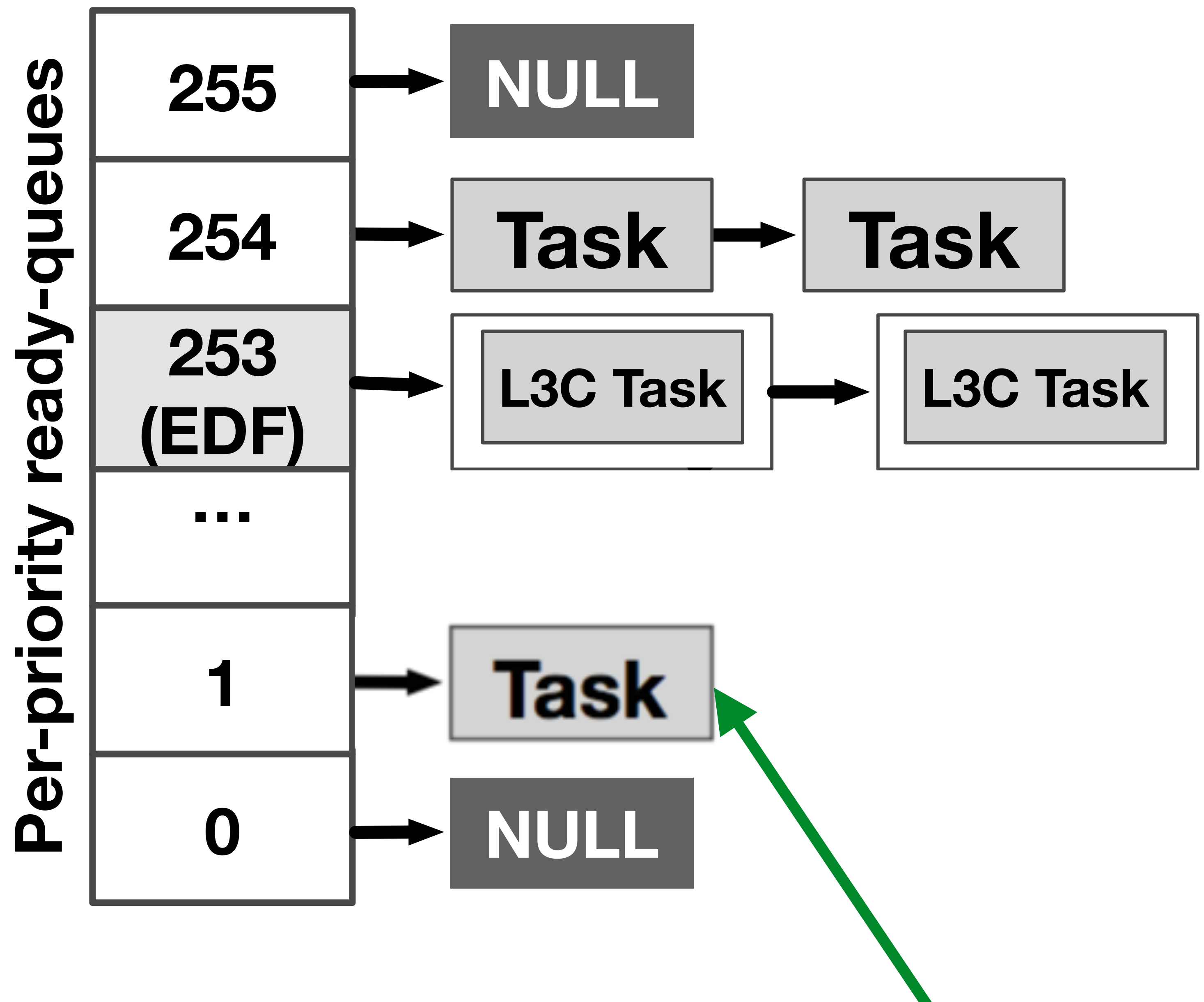


Individual L3C tasks **don't need to be certified**.
Certifying the enforcement mechanism is sufficient.

System Configuration



System Configuration

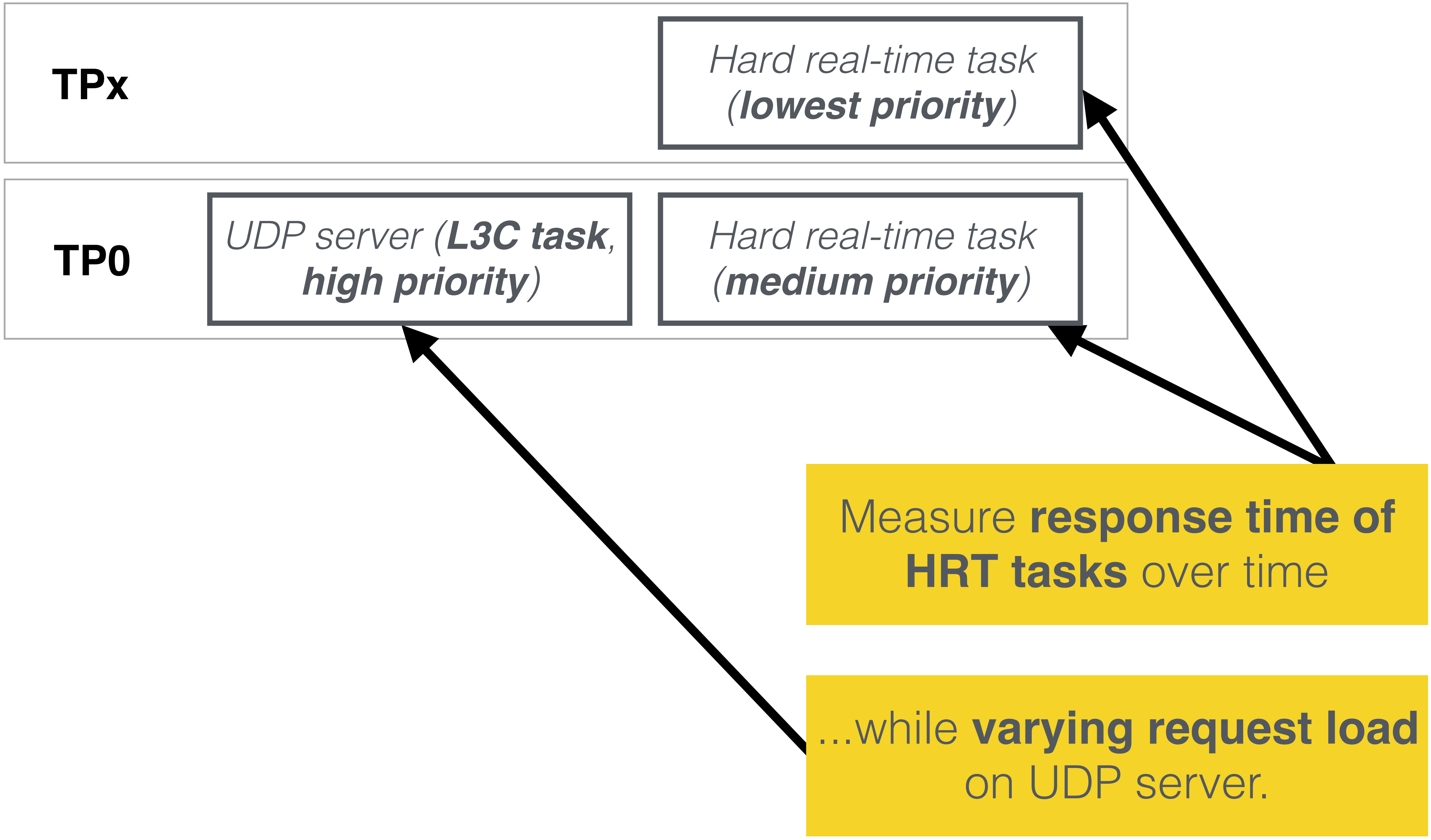


Need to **inflate budgets of lower- or equal-priority TP0 and TPx tasks (and change partition table)**.

Does it Work?

Does it Work?

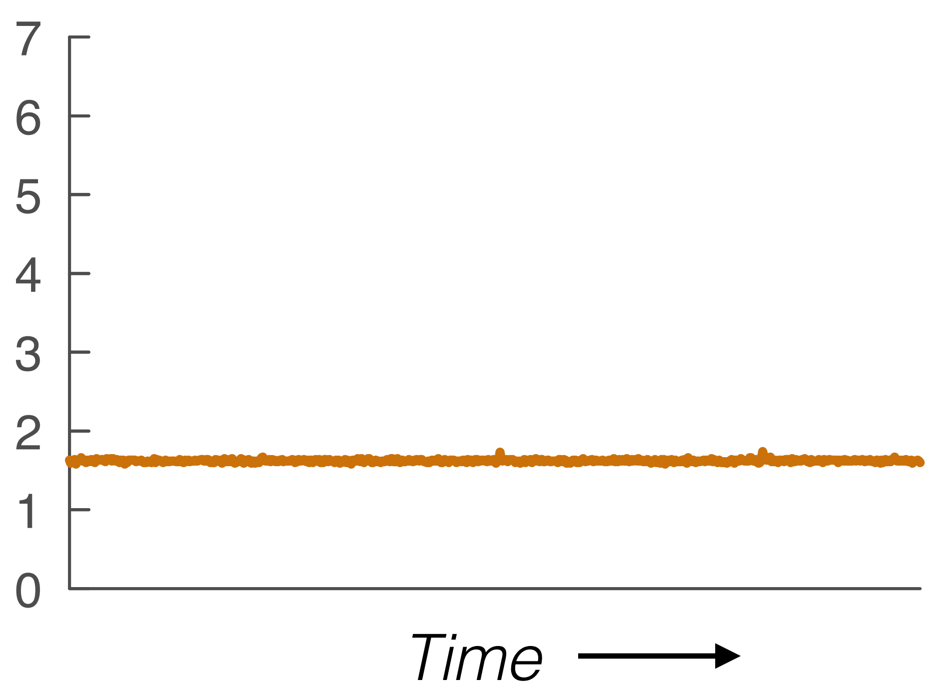
Experimental Setup



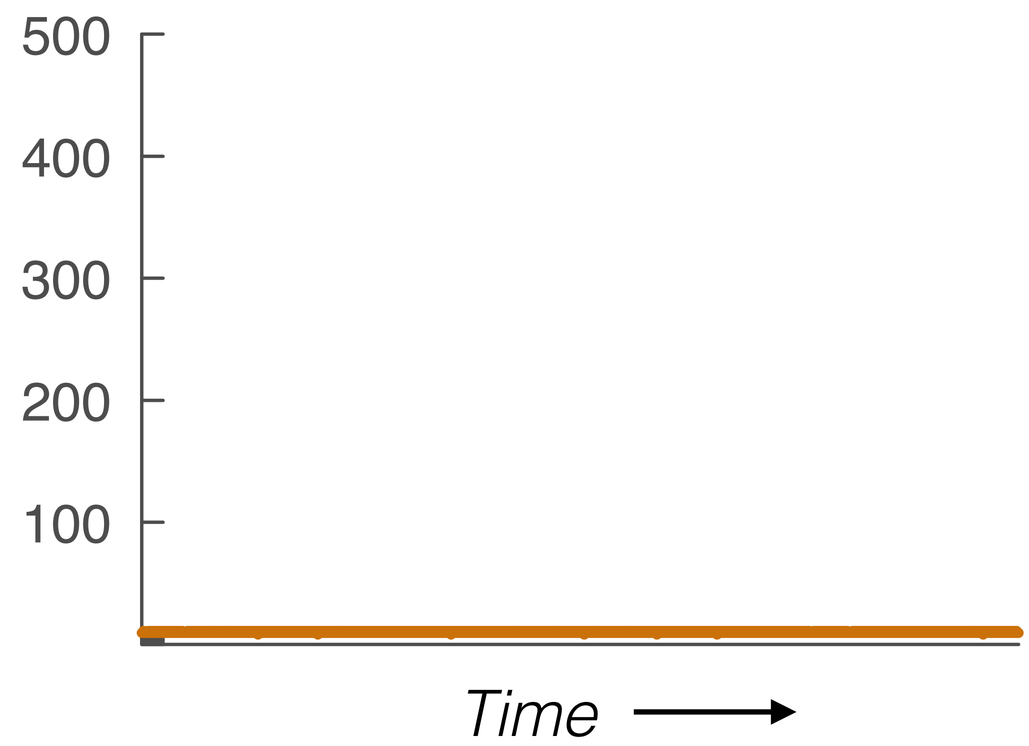
Does it Work?

L3C task at high priority in TP0

No load (zero requests per second)



Response time of TP0 task

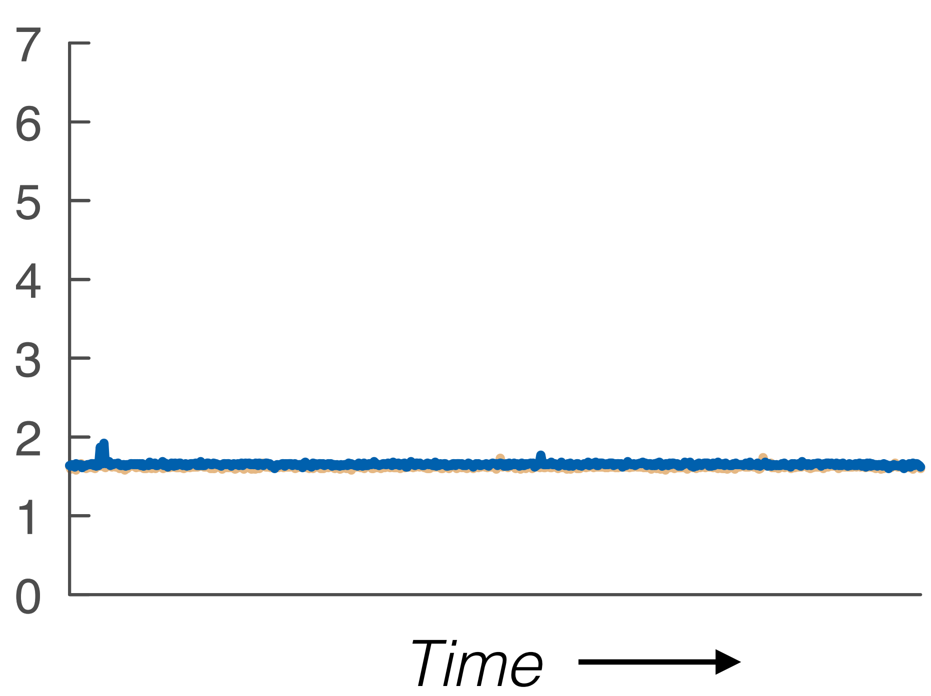


Response time of TPx task

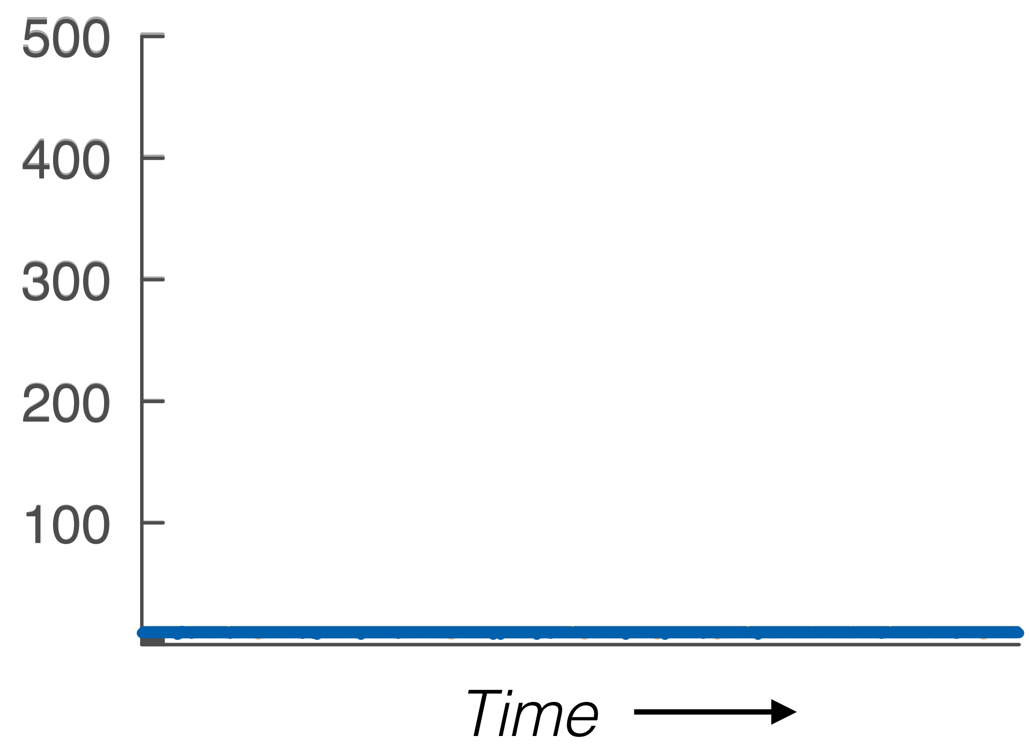
Does it Work?

L3C task at high priority in TP0

Normal load: 10 req/sec



Response time of TP0 task

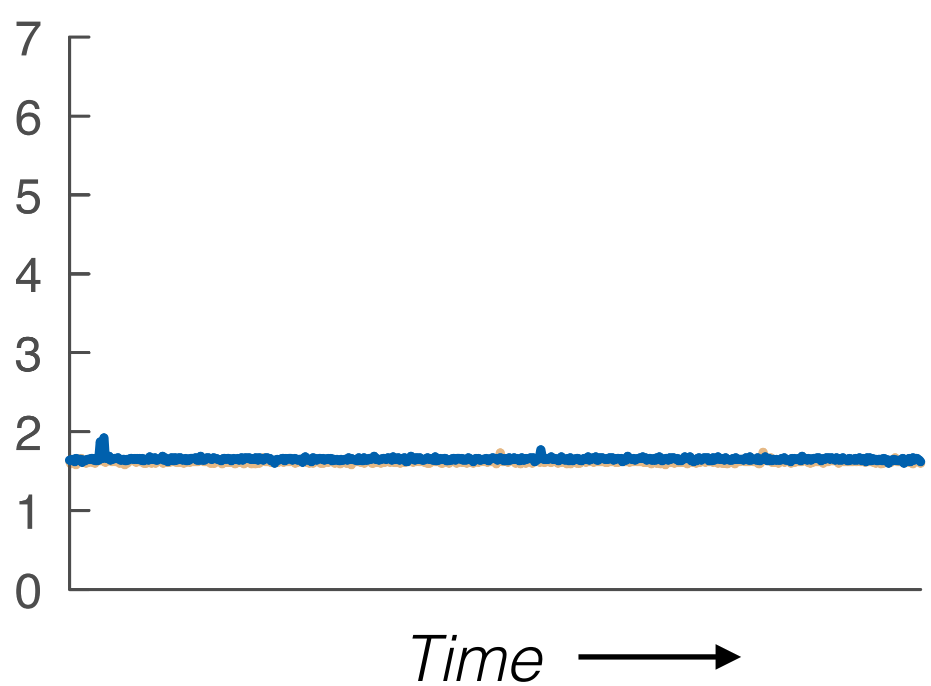


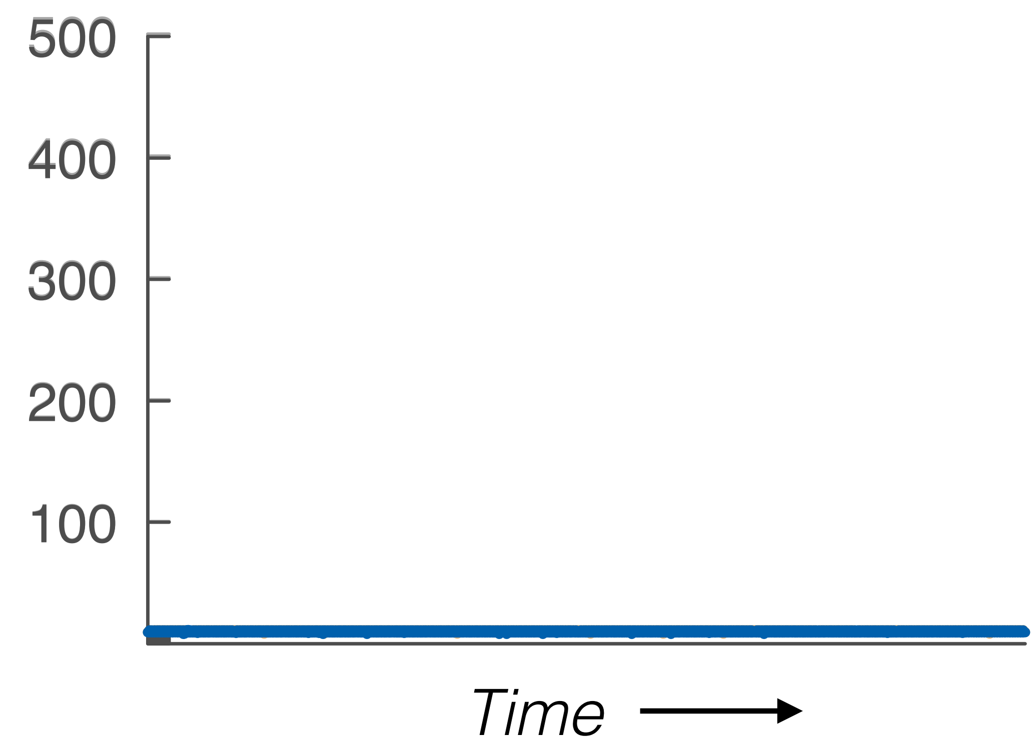
Response time of TPx task

Does it Work?

L3C task at high priority in TP0

Normal load: 10 req/sec





Response time o

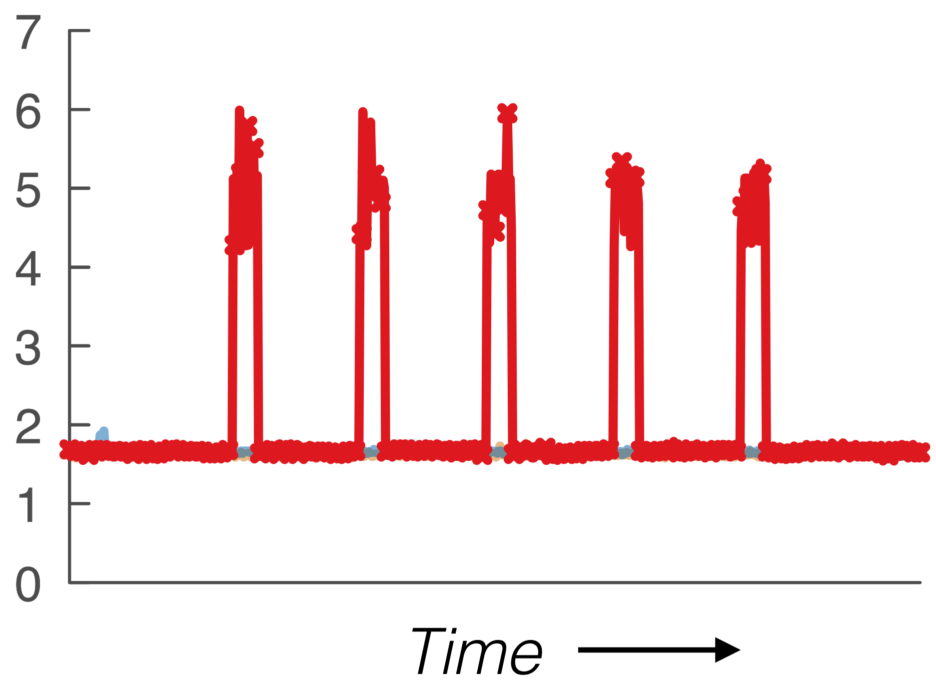
No undue interference on TP0 and TPx tasks under normal load.

time of TPx task

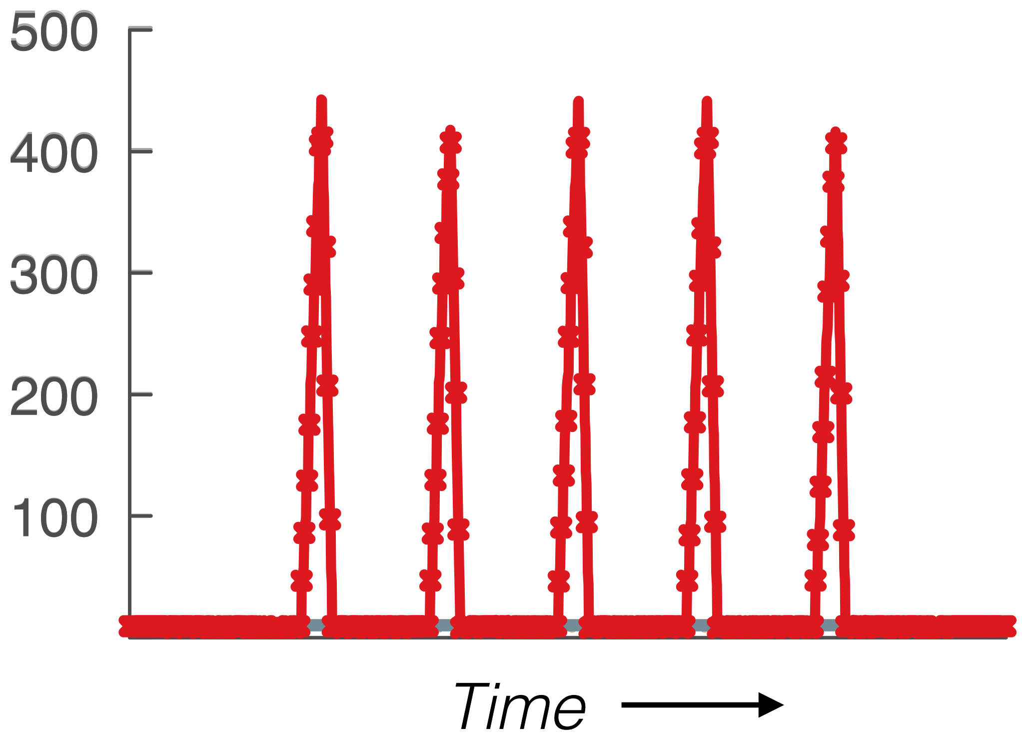
Does it Work?

L3C task at high priority in TP0

10 req/sec + bursts



Response time of TP0 task

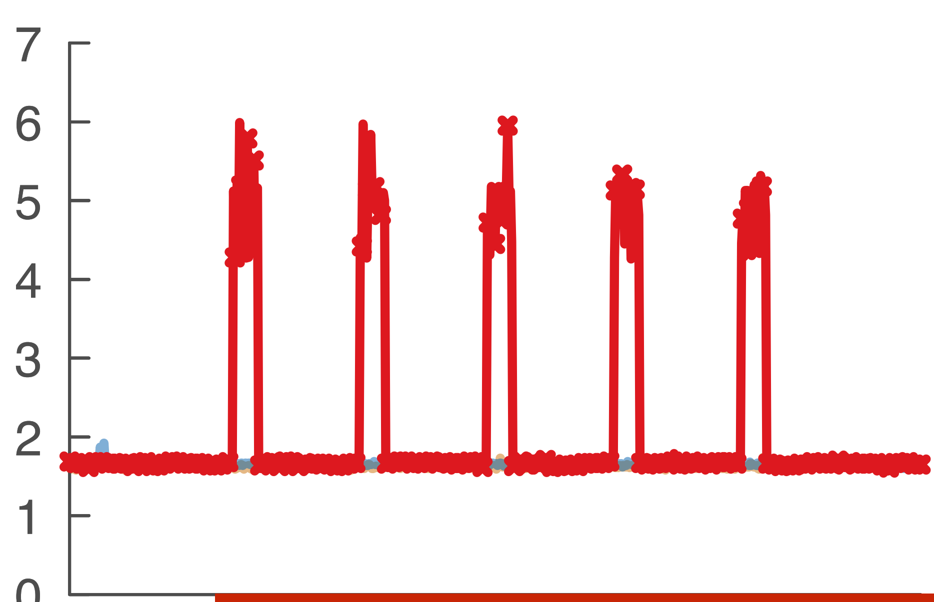


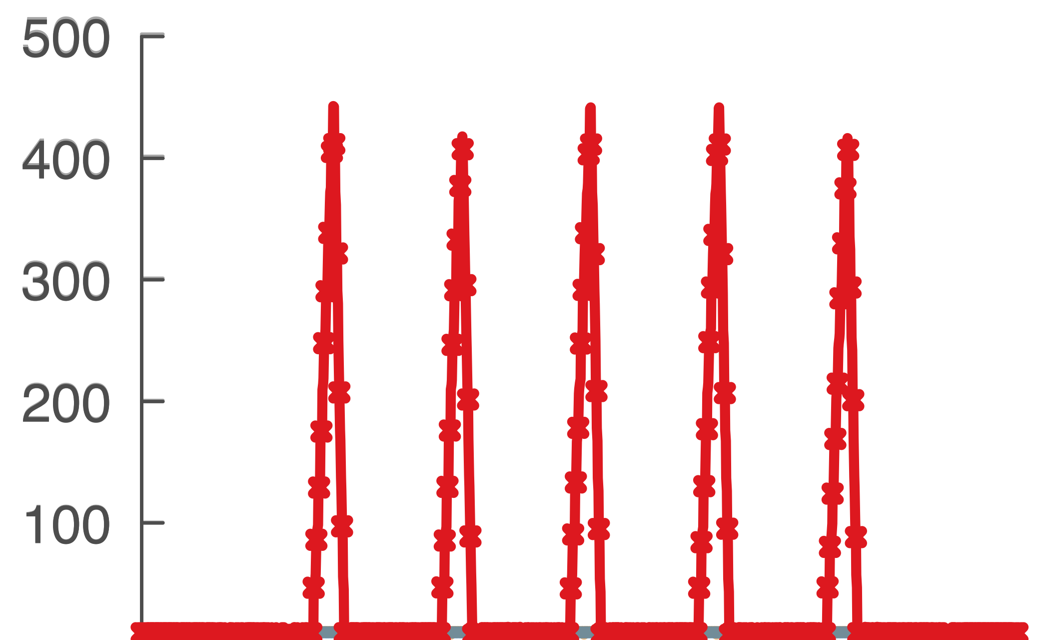
Response time of TPx task

Does it Work?

L3C task at high priority in TP0

10 req/sec + bursts



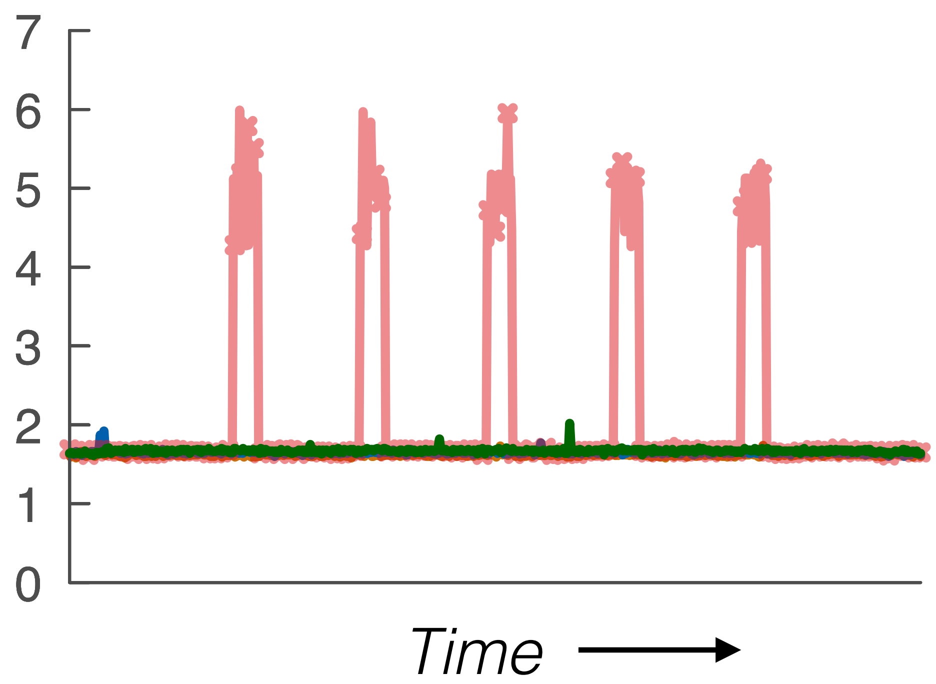


L3C task causes **response time spikes** in lower-priority tasks across the system

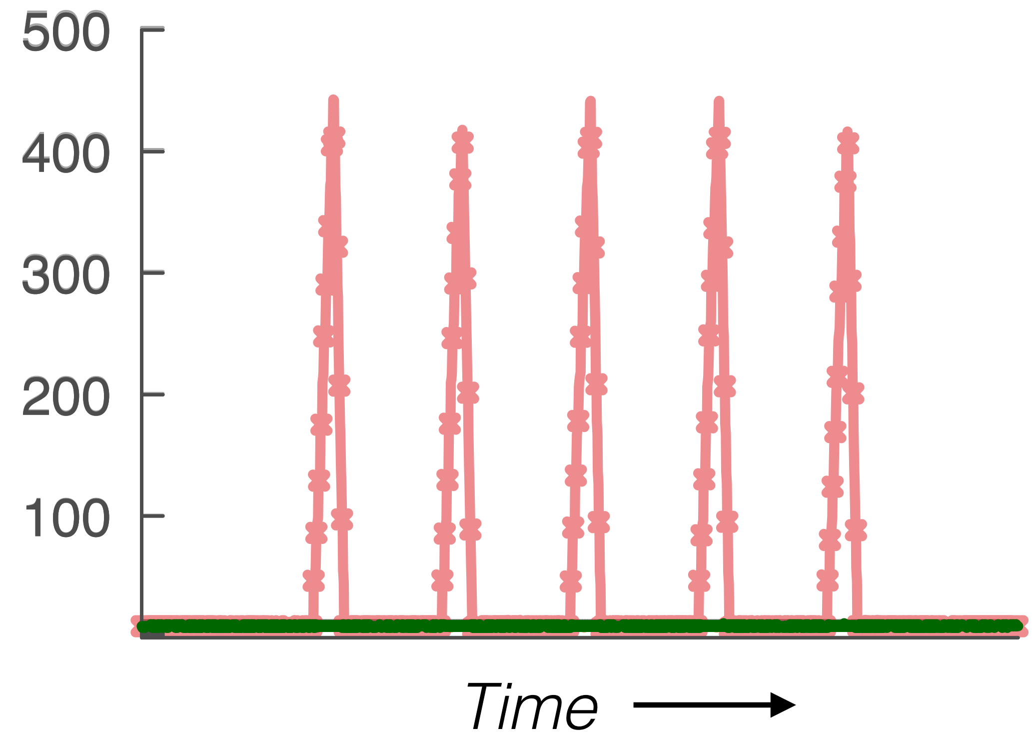
Does it Work?

L3C task at high priority in TP0 within reservation
(budget 1ms, period 50ms, determined empirically)

10 req/sec + bursts



Response time of TP0 task

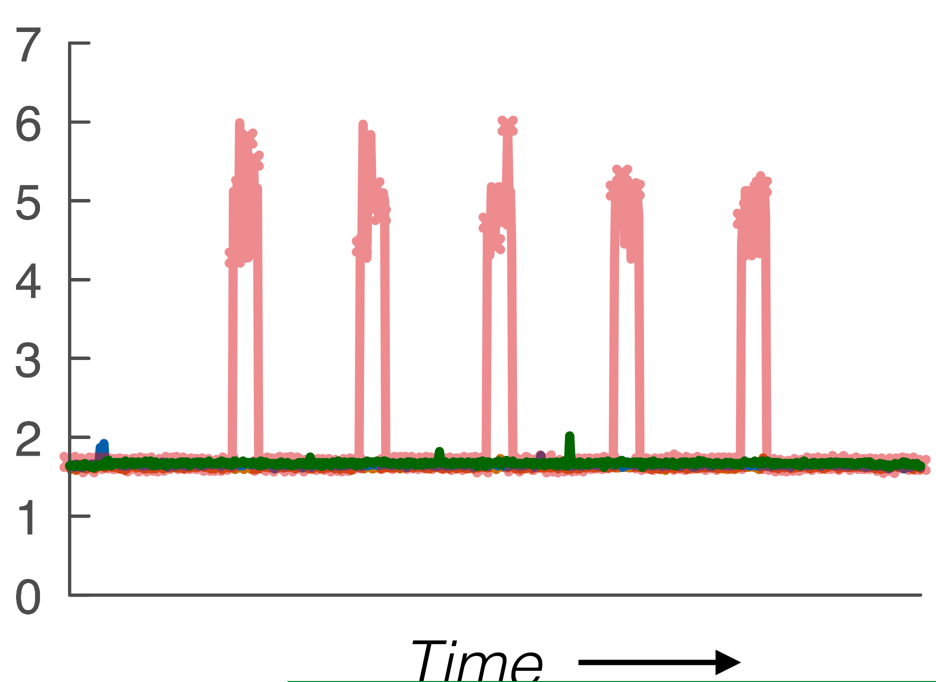


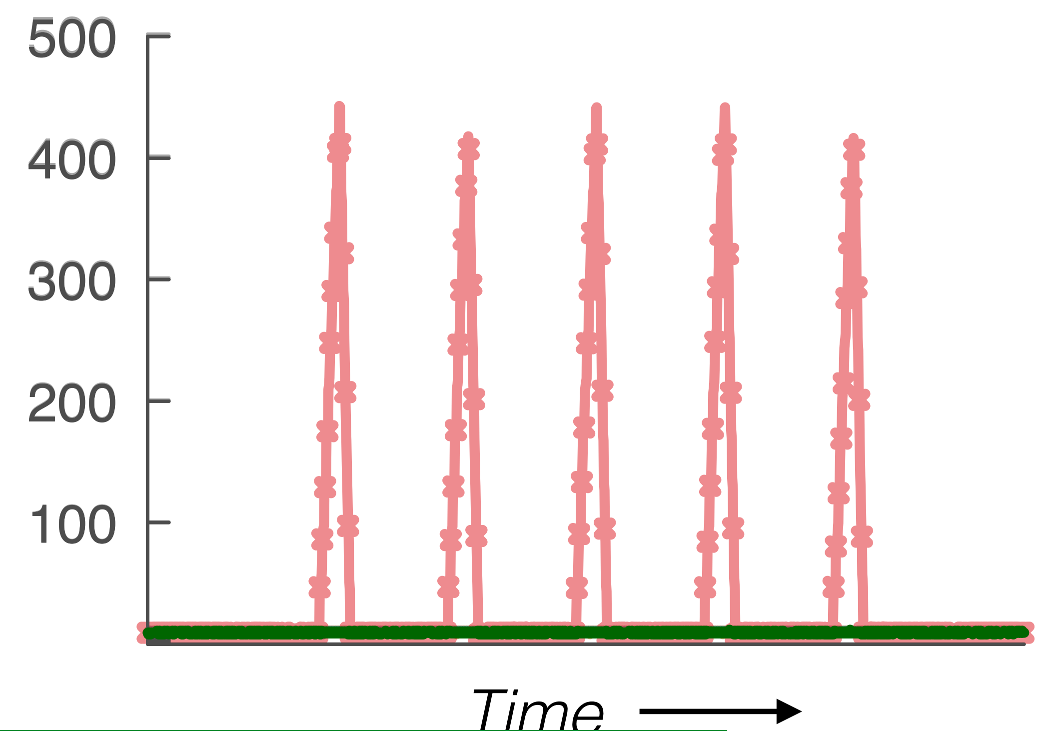
Response time of TPx task

Does it Work?

L3C task at high priority in TP0 within reservation
(budget 1ms, period 50ms, determined empirically)

10 req/sec + bursts





Resp

Px task

The encapsulating reservation **prevents bursts from affecting other parts of the system**.

A Case Study

Identified **a deficiency in L3C support** in PikeOS and showed why they are difficult to support under the existing scheduler.

Highlighted **key design constraints** required in a commercial context, **typically not addressed in academic designs**.

Showed that careful integration of **reservation-based scheduling** best suited the constraints of an existing high-criticality RTOS.

Implemented a **prototype in PikeOS**, and presented results from a **freely-shareable re-implementation in LITMUS^{RT}**.

https://people.mpi-sws.org/~bbb/papers/details/rtns17/index.html

Thanks!

LITMUS^{RT}

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

https://litmus-rt.org