TimerShield

Protecting High-Priority Tasks from Low-Priority Timer Interference

Pratyush Patel[1,2], Manohar Vanga[1], Björn Brandenburg[1]

[1]MPI-SWS, [2]Carnegie Mellon University

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS
Kaiserslautern, Germany

RTAS 2017
April 18, 2017
Pittsburgh, USA

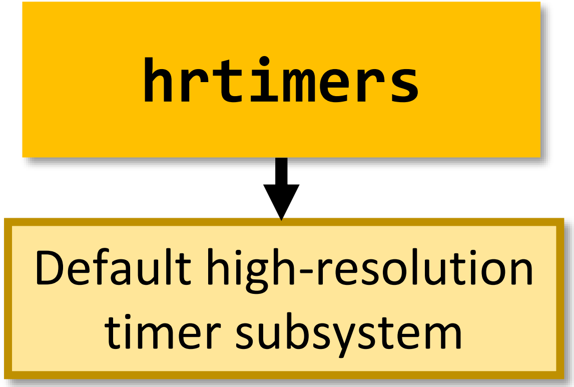
This Paper

This Paper

`hrtimers`

PREEMPT_RT

This Paper

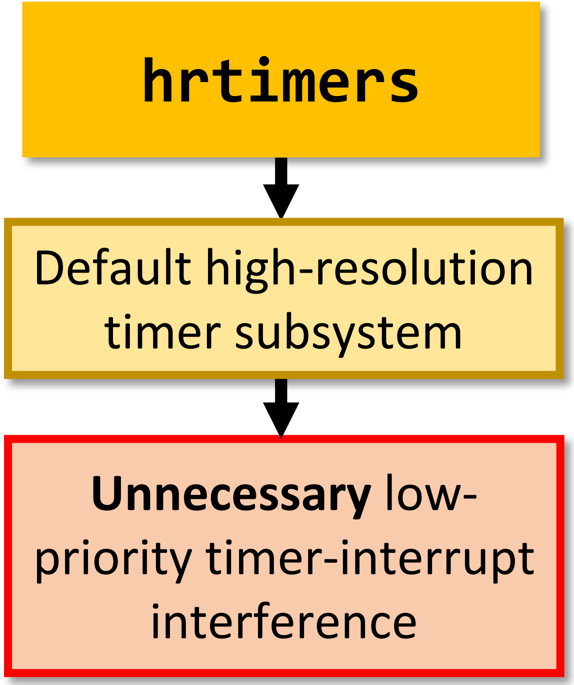


hrtimers

Default high-resolution timer subsystem

PREEMPT_RT

This Paper



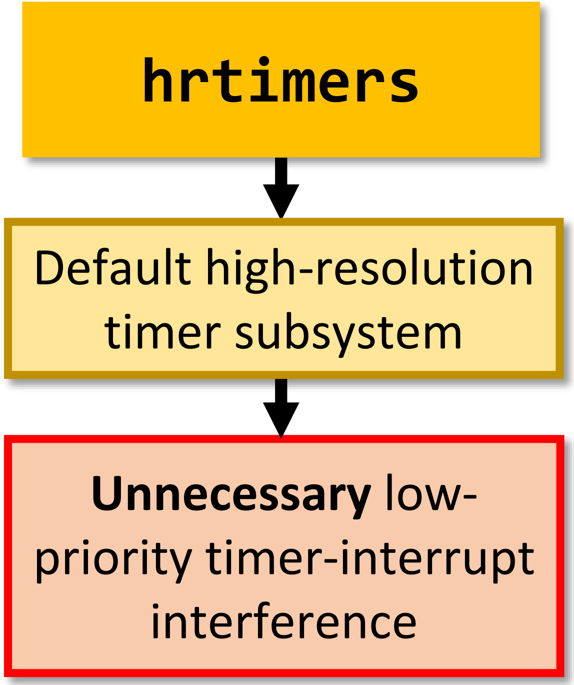
hrtimers

Default high-resolution timer subsystem

Unnecessary low-priority timer-interrupt interference

PREEMPT_RT

This Paper



`hrtimers`

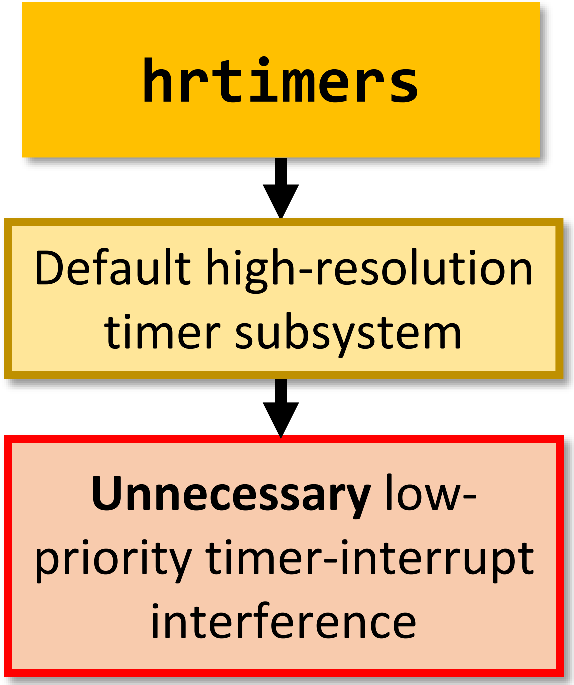
Default high-resolution timer subsystem

Unnecessary low-priority timer-interrupt interference

PREEMPT_RT

TimerShield

This Paper

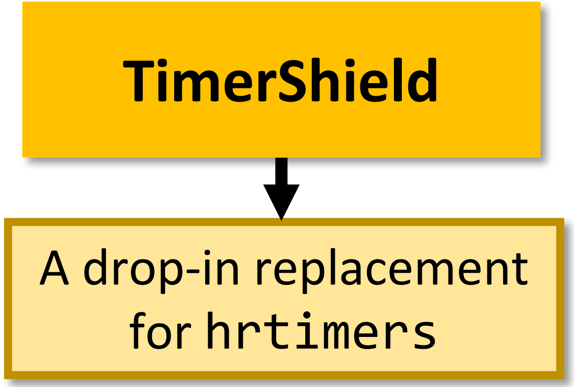


`hrtimers`

Default high-resolution timer subsystem

Unnecessary low-priority timer-interrupt interference

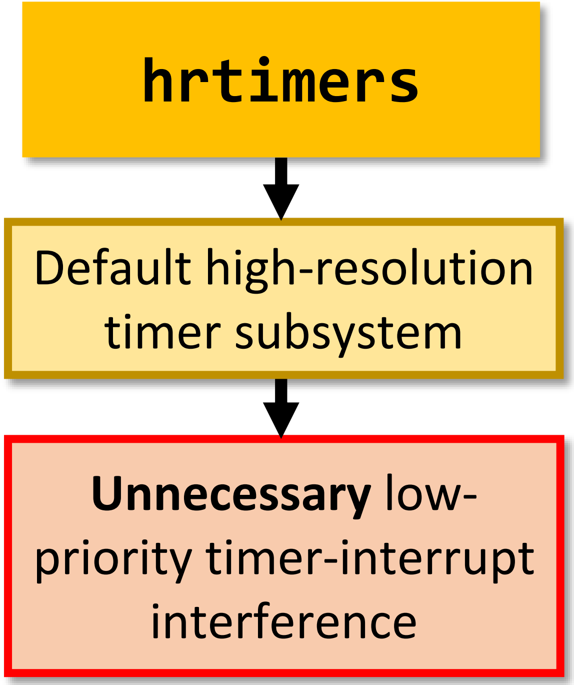
PREEMPT_RT



TimerShield

A drop-in replacement for `hrtimers`

This Paper



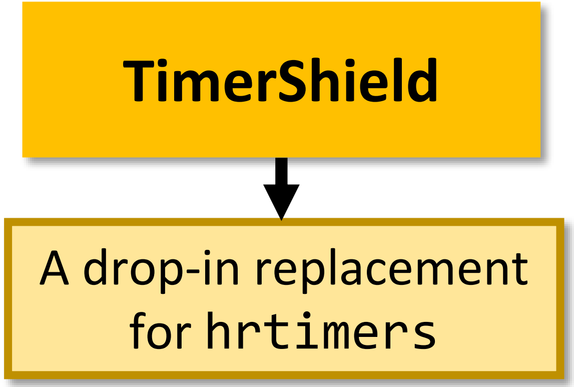
hrtimers

Default high-resolution timer subsystem

Unnecessary low-priority timer-interrupt interference

PREEMPT_RT

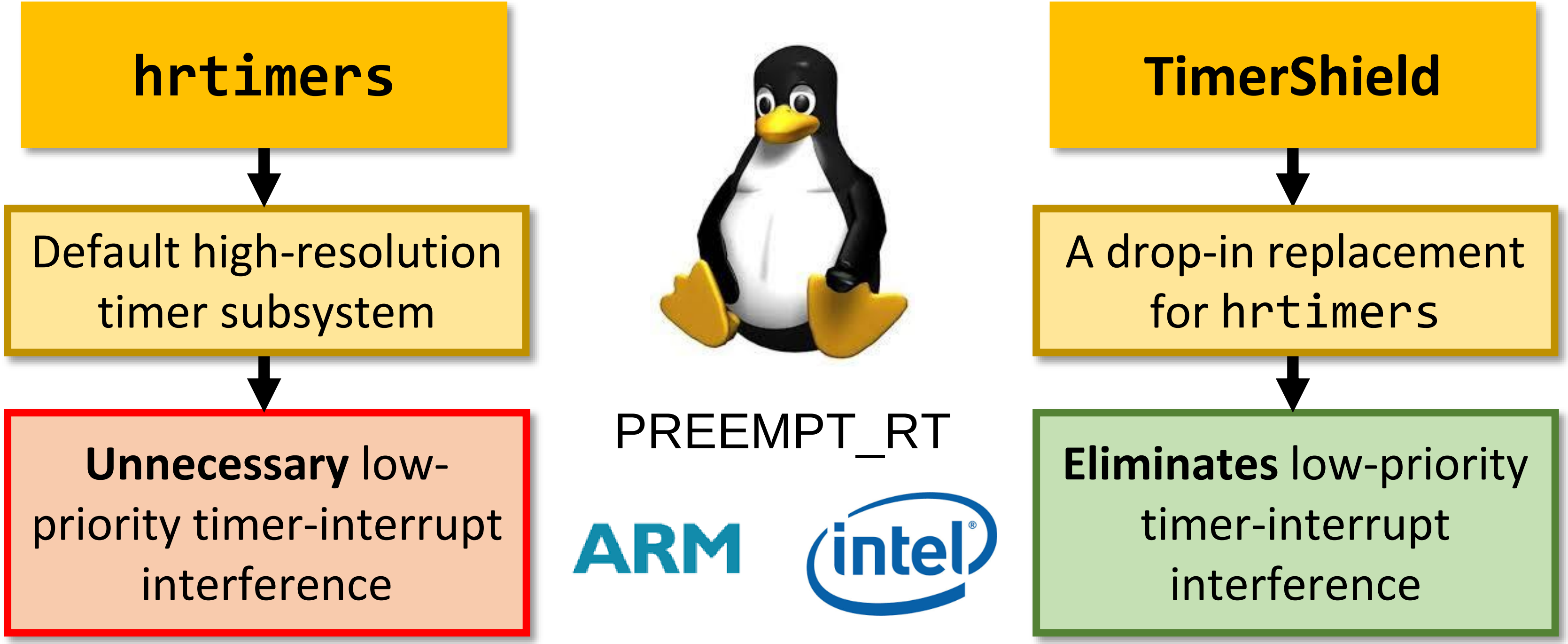
ARM intel



TimerShield

A drop-in replacement for hrtimers

This Paper



`hrtimers`

Default high-resolution timer subsystem

Unnecessary low-priority timer-interrupt interference

PREEMPT_RT

ARM

intel

TimerShield

A drop-in replacement for `hrtimers`

Eliminates low-priority timer-interrupt interference

Talk Overview

Timers and the Interference Problem

TimerShield Design

Evaluation

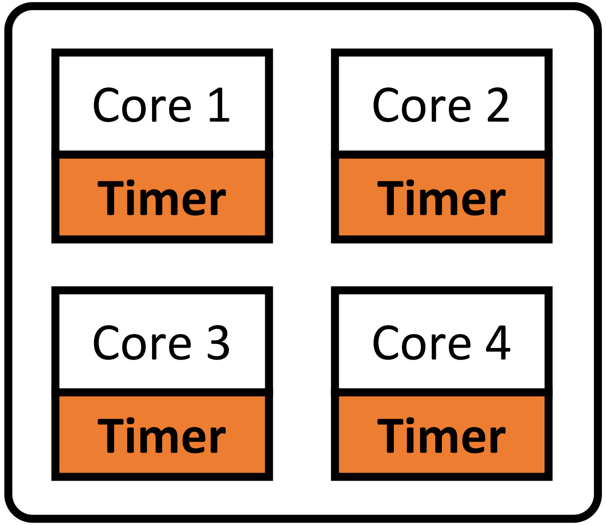
Talk Overview

Timers and the Interference Problem

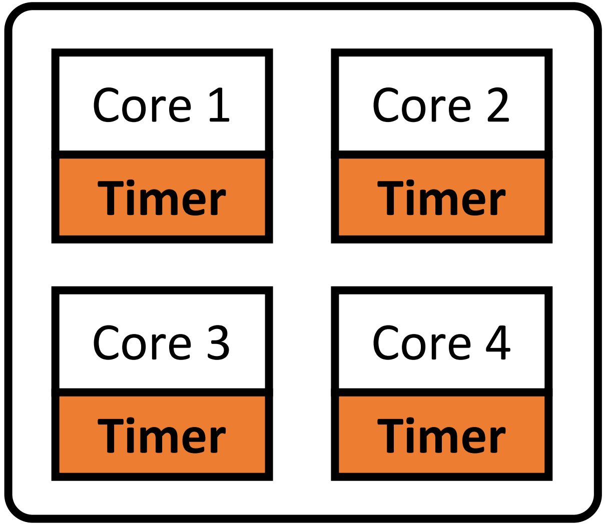
TimerShield Design

Evaluation

High-Resolution Timers

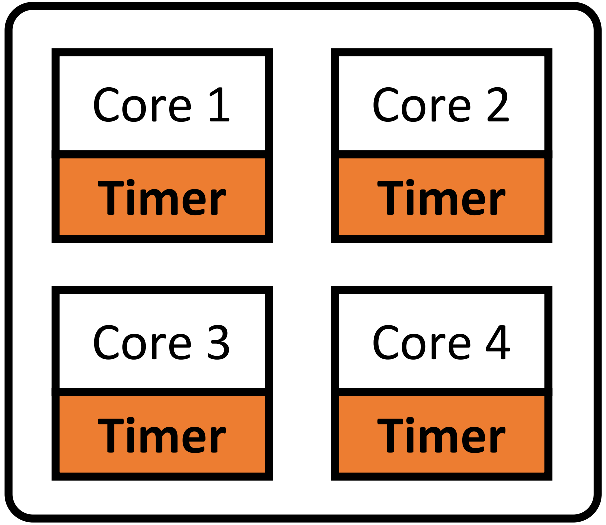


High-Resolution Timers



Core-local timers with **cycle precision**

High-Resolution Timers



Core-local timers with **cycle precision**

Can be **programmed** to raise an interrupt at a desired time

Timers in Real-Time OSes

Timers in Real-Time OSes

Job Releases

Tasks can be woken up periodically using timers

Timers in Real-Time OSes

Job Releases

Tasks can be woken up periodically using timers

Budget Enforcement

Schedulers use timers to prevent budget overruns

Timers in Real-Time OSes

Job Releases

Tasks can be woken up periodically using timers

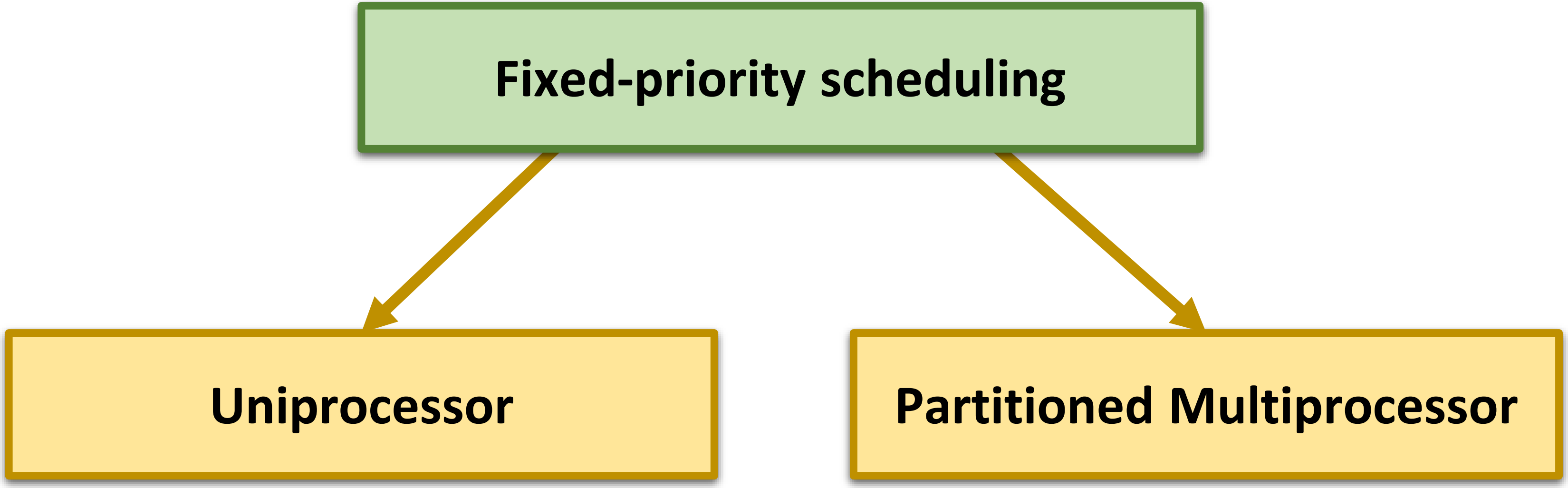
Budget Enforcement

Schedulers use timers to prevent budget overruns

Self-Suspensions

Tasks can use POSIX *clock_nanosleep()* to suspend themselves

Assumptions

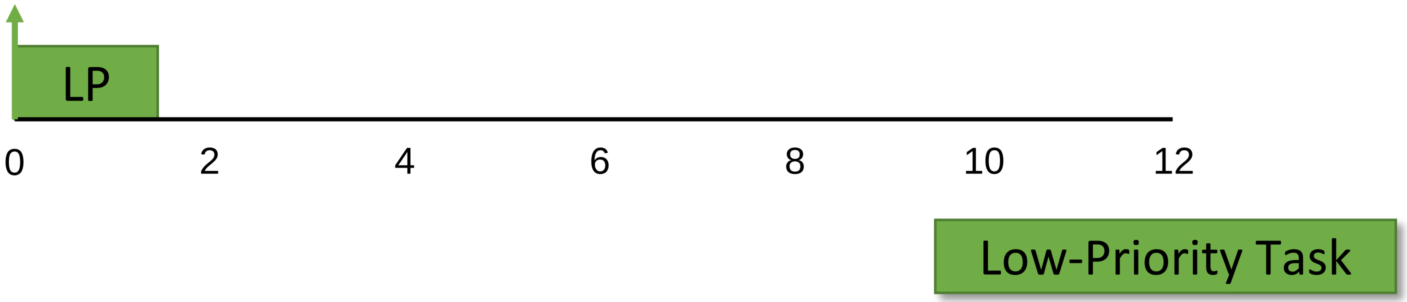


Fixed-priority scheduling

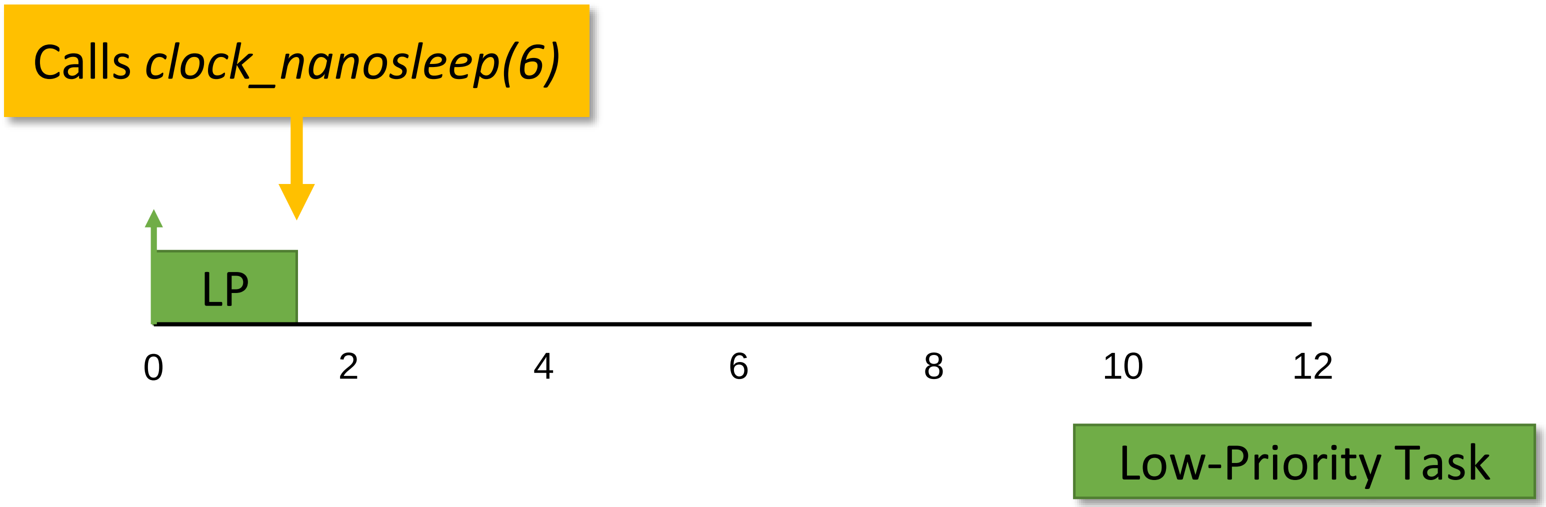
Uniprocessor

Partitioned Multiprocessor

Timer-Interrupt Interference



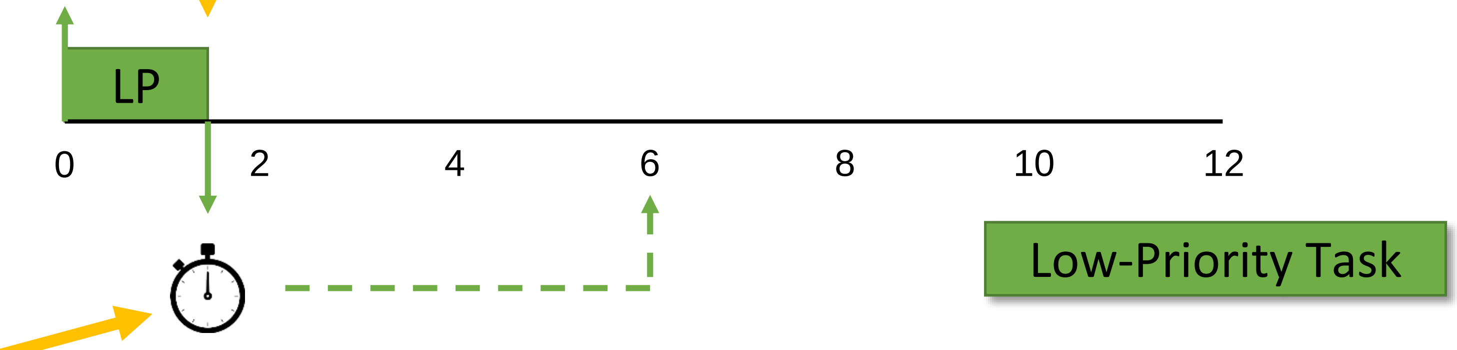
Timer-Interrupt Interference



Calls *clock_nanosleep(6)*

Low-Priority Task

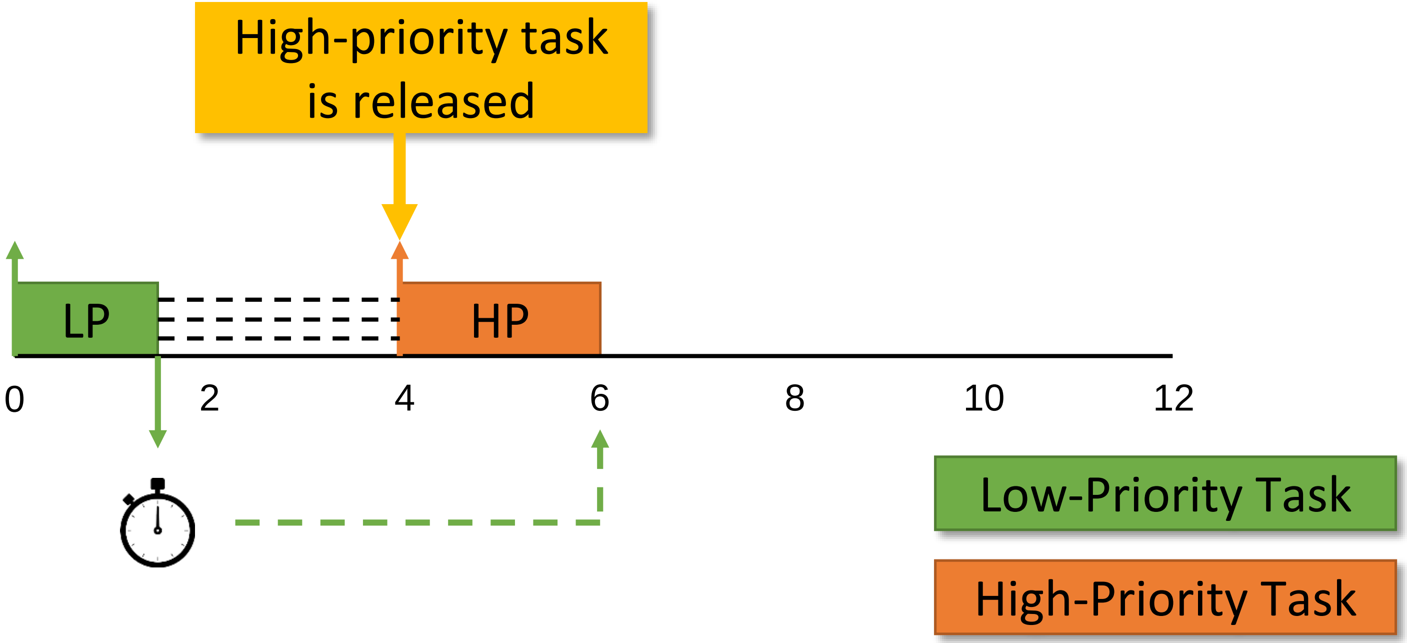
Timer-Interrupt Interference



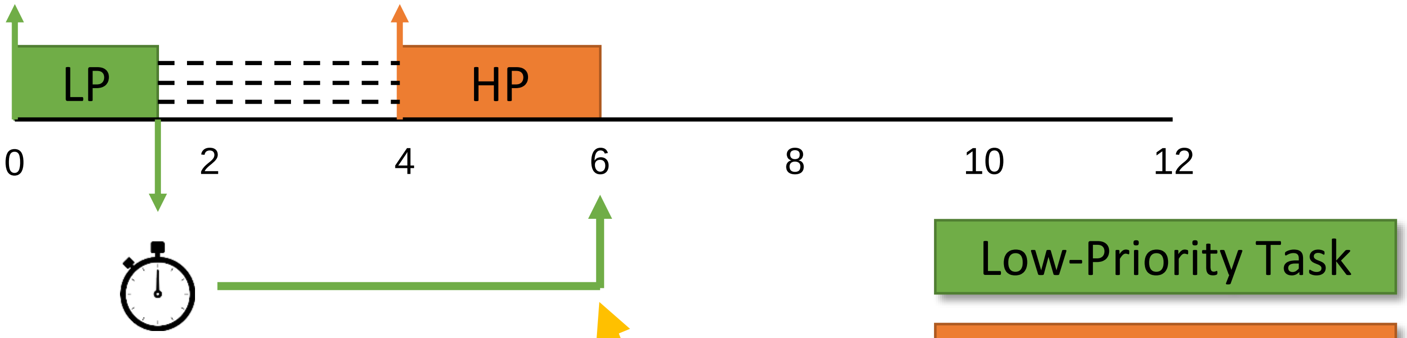
Calls *clock_nanosleep(6)*

Timer **hardware is programmed** to fire at the specified time

Timer-Interrupt Interference



Timer-Interrupt Interference



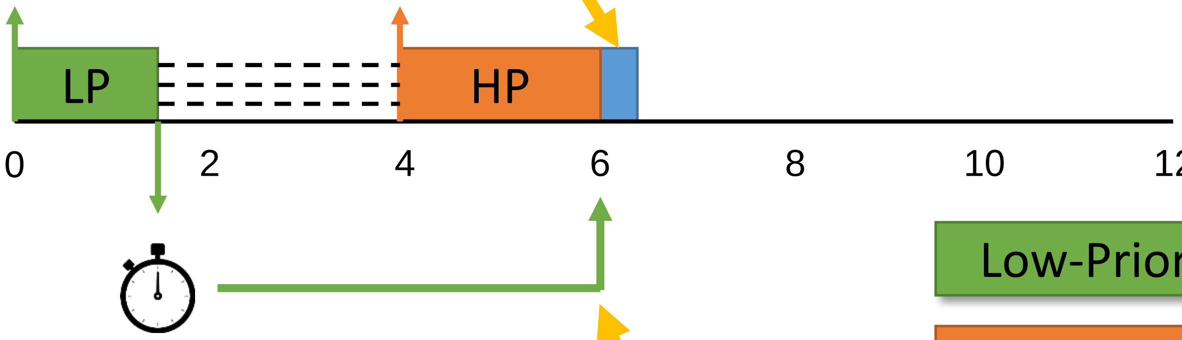
🟩 Low-Priority Task

🟥 High-Priority Task

At t = 6, timer hardware fires an **interrupt**

Timer-Interrupt Interference

HP is preempted to service the interrupt (LP task is woken up)



At t = 6, timer hardware fires an **interrupt**

🟩 Low-Priority Task

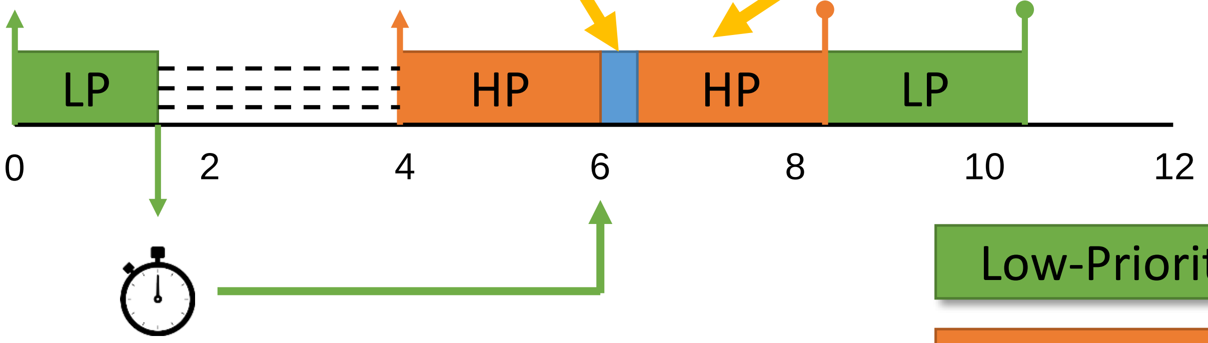
🟧 High-Priority Task

🟦 Timer Handler

Timer-Interrupt Interference

HP is preempted to service the interrupt (LP task is woken up)

HP task resumes

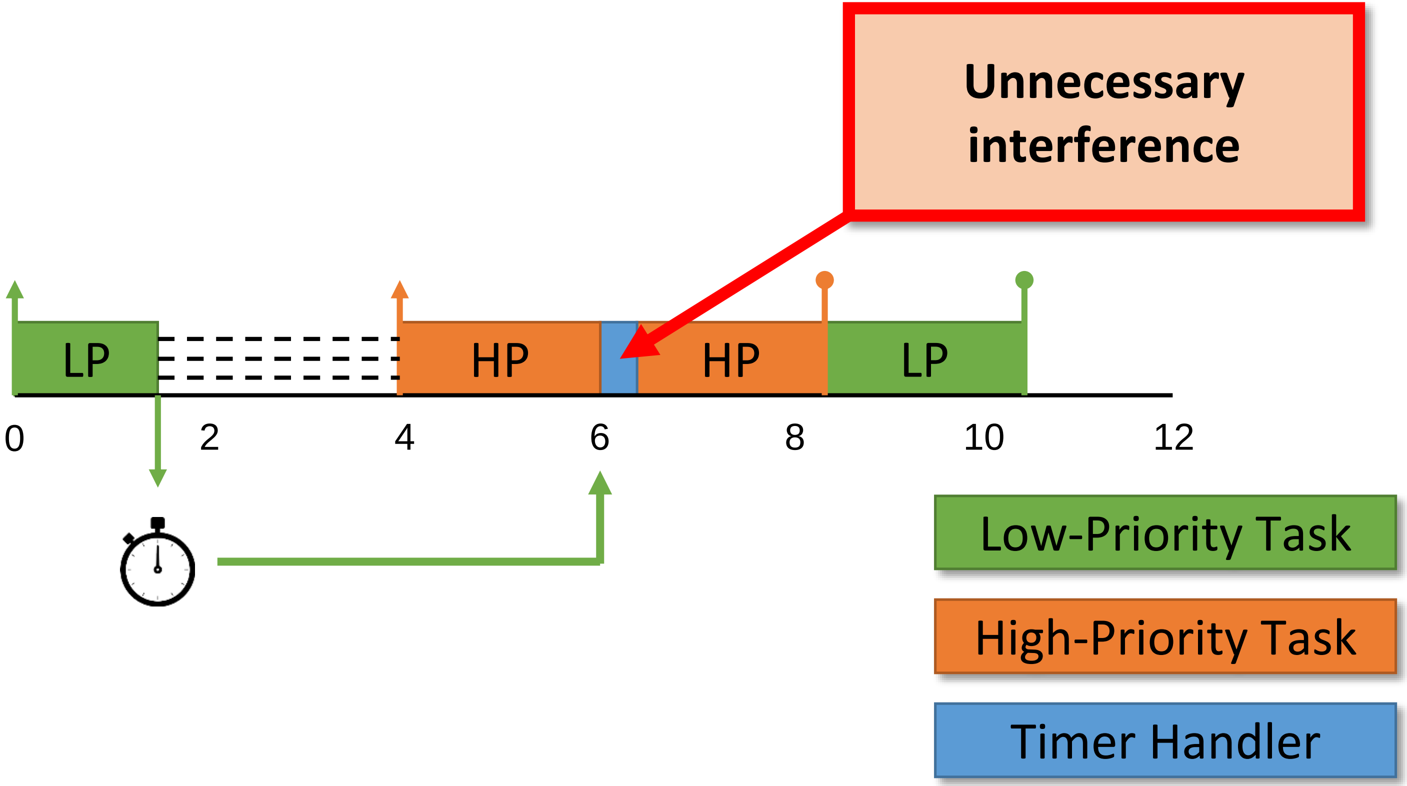


🟩 Low-Priority Task

🟧 High-Priority Task

🟦 Timer Handler

Timer-Interrupt Interference



Unnecessary interference

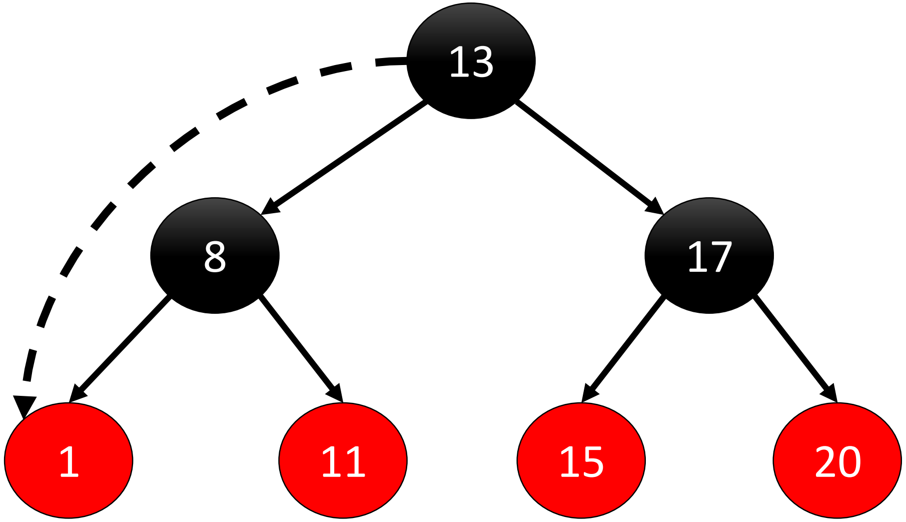
Low-Priority Task

High-Priority Task

Timer Handler

Why Does Interference Occur?

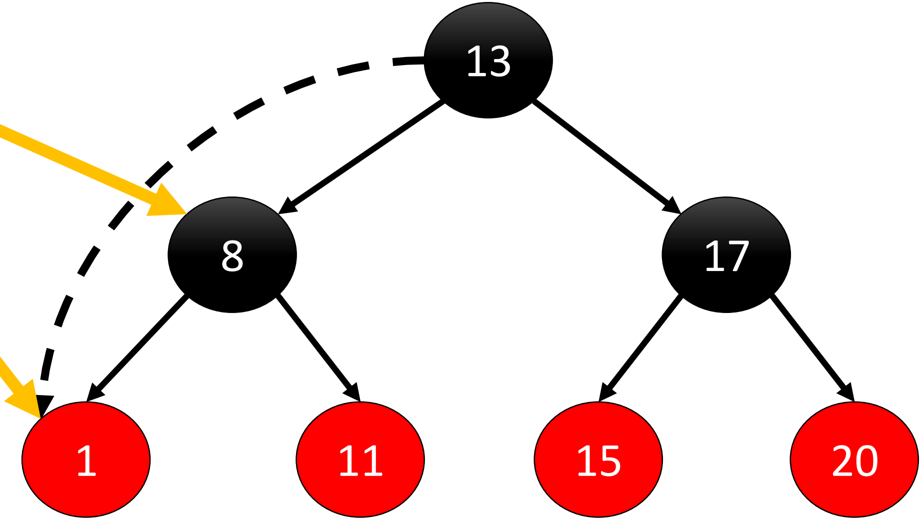
Linux's `hrtimer` subsystem



Why Does Interference Occur?

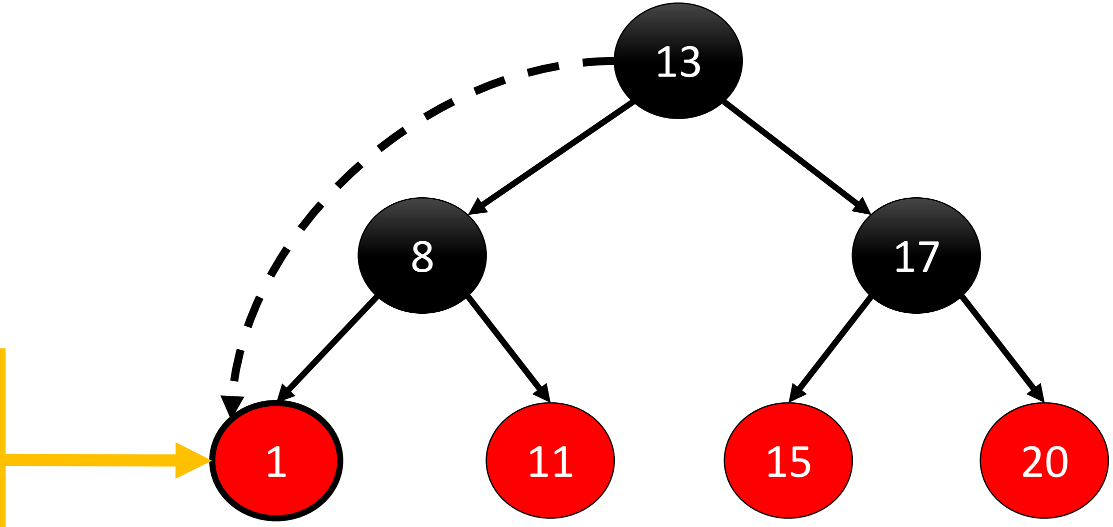
Multiplexes many software timers on a single hardware timer using a **time-ordered red-black tree**

Linux’s `hrtimer` subsystem



Why Does Interference Occur?

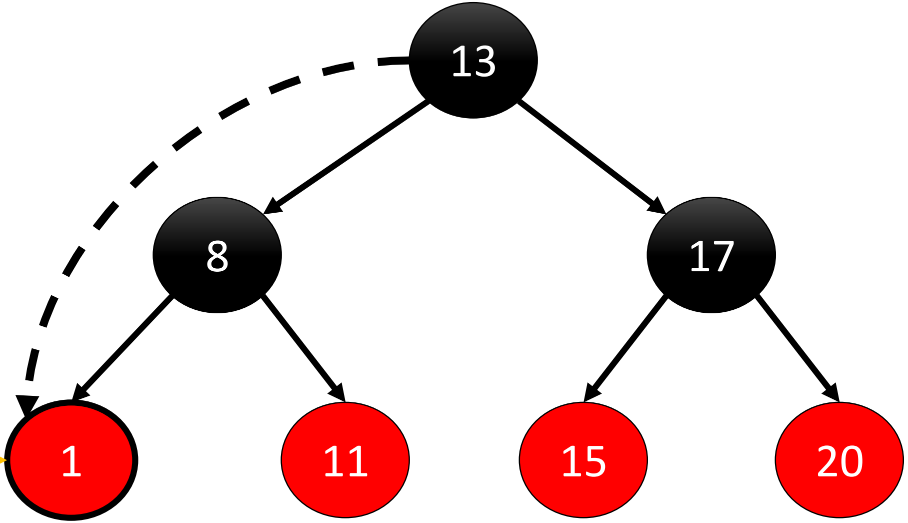
Linux's `hrtimer` subsystem



Earliest expiring timer is programmed into hardware

Why Does Interference Occur?

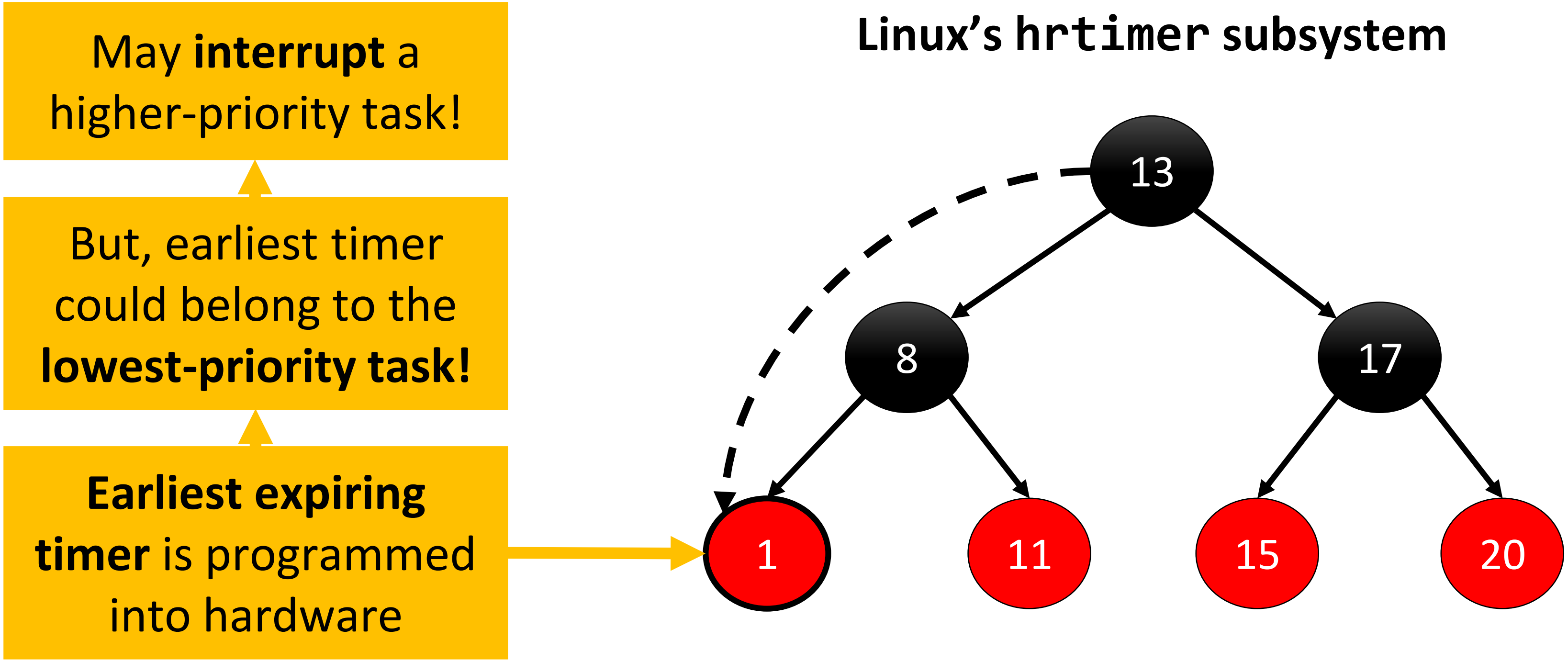
Linux's `hrtimer` subsystem



But, earliest timer could belong to the **lowest-priority task!**

Earliest expiring timer is programmed into hardware

Why Does Interference Occur?



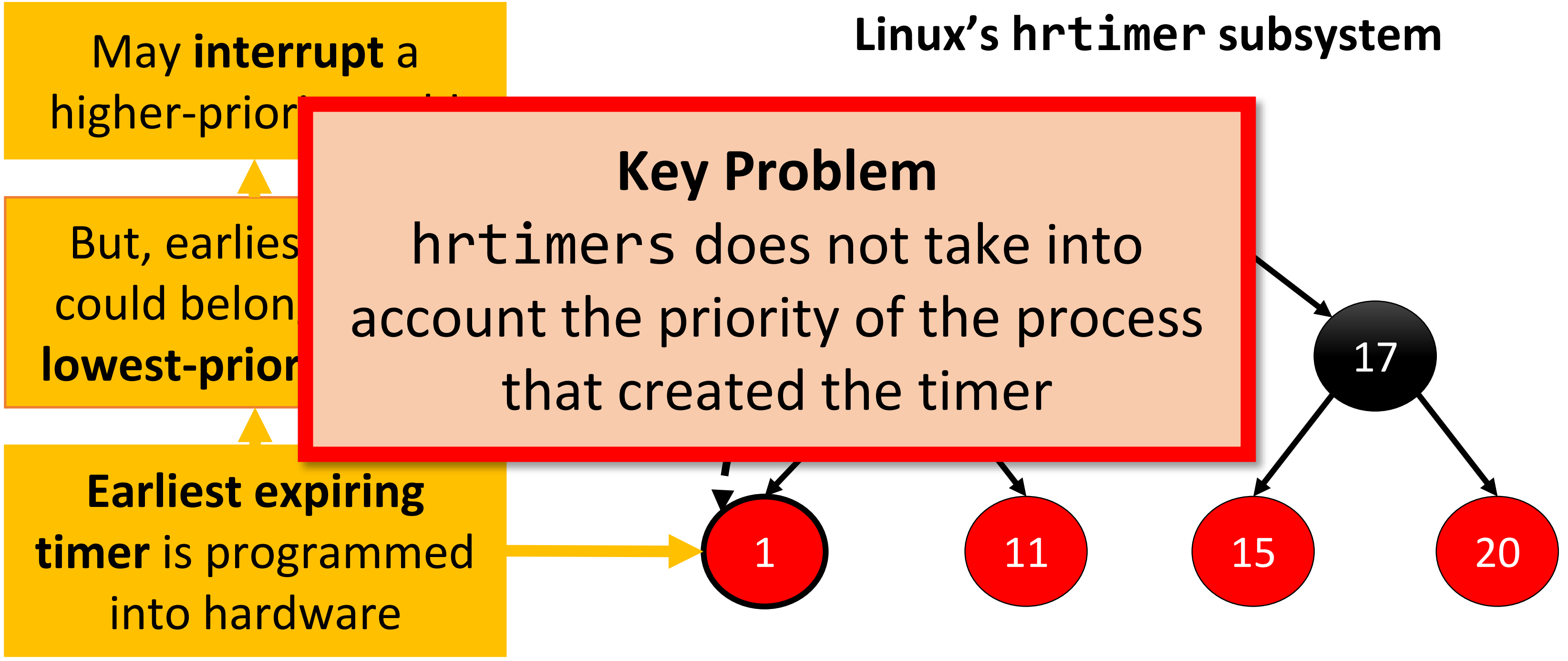
May **interrupt** a higher-priority task!

But, earliest timer could belong to the **lowest-priority task!**

Earliest expiring timer is programmed into hardware

Linux's `hrtimer` subsystem

Why Does Interference Occur?



May **interrupt** a higher-prior

But, earlies could belon **lowest-prior**

Earliest expiring timer is programmed into hardware

Linux's `hrtimer` subsystem

Key Problem

`hrtimers` does not take into account the priority of the process that created the timer

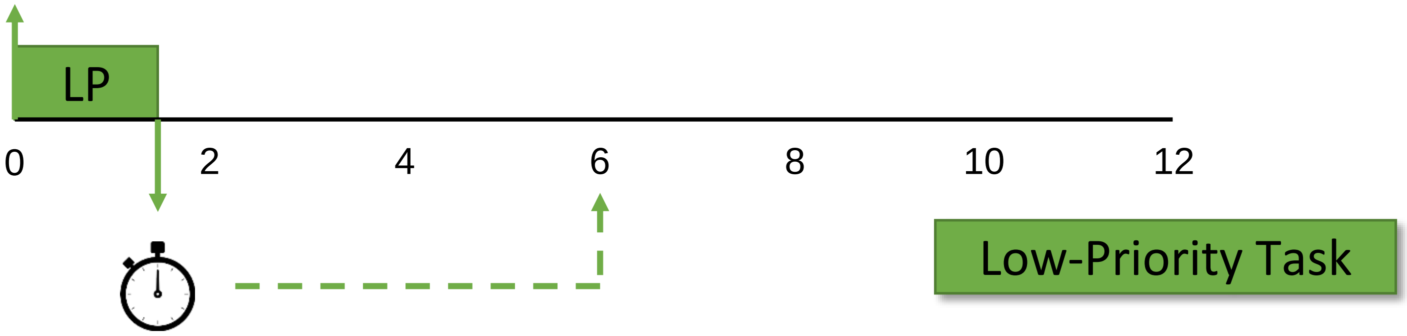
Talk Overview

Timers and the Interference Problem

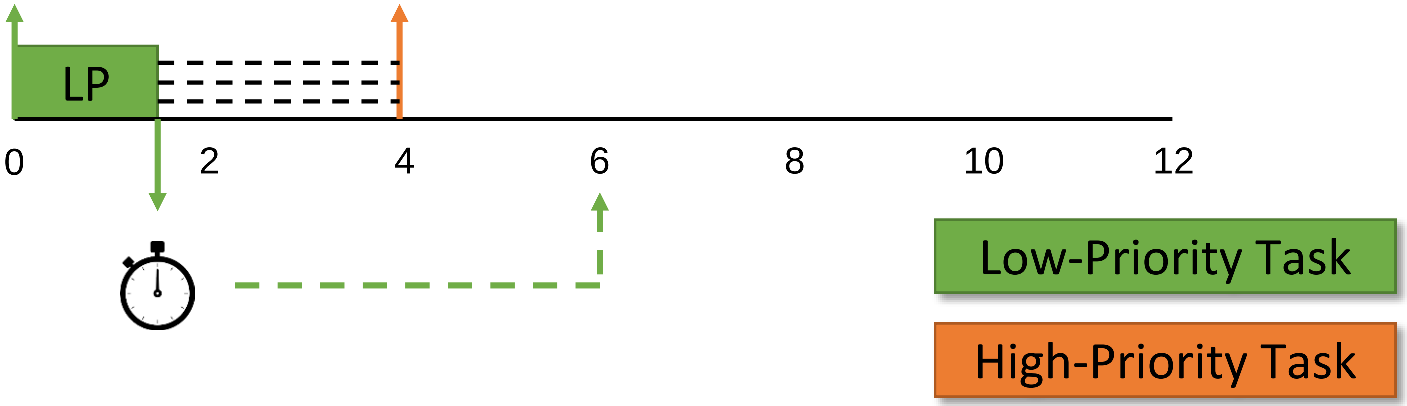
TimerShield Design

Evaluation

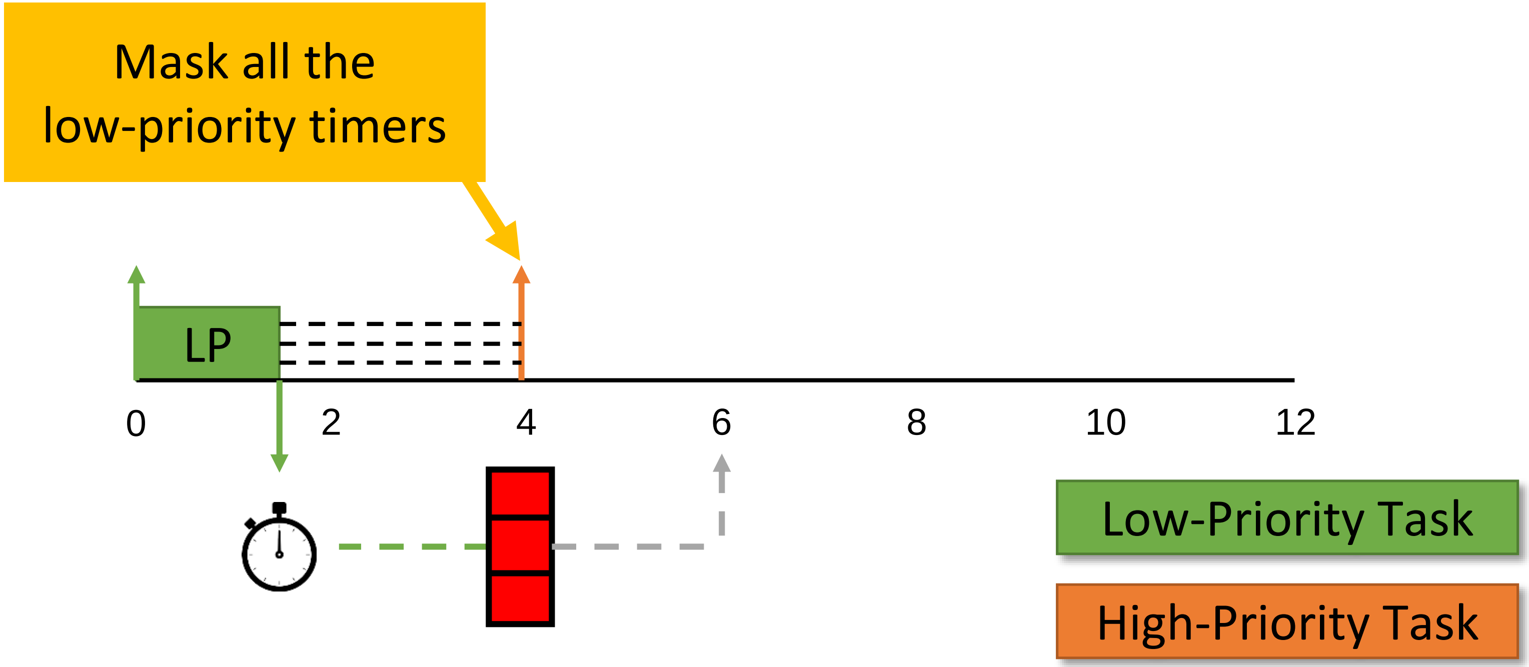
How Does TimerShield Work?



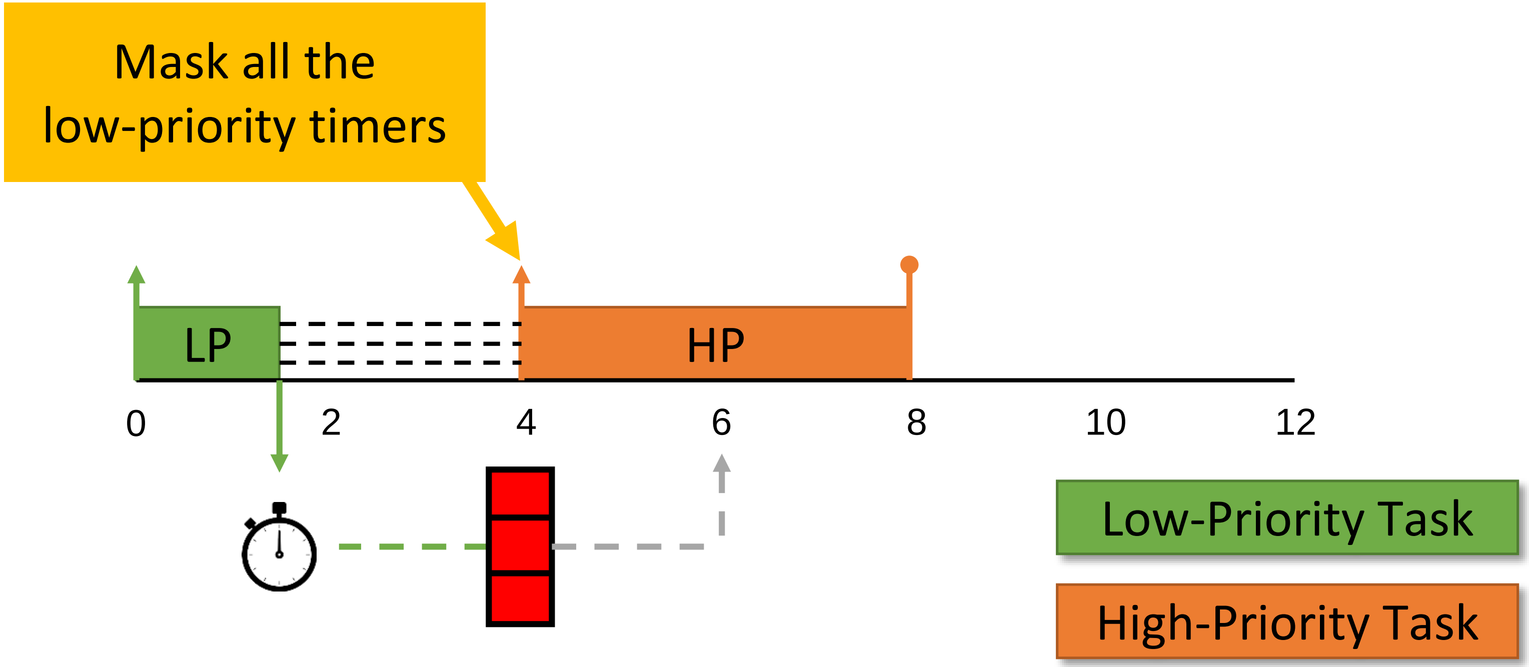
How Does TimerShield Work?



How Does TimerShield Work?


Mask all the low-priority timers

How Does TimerShield Work?

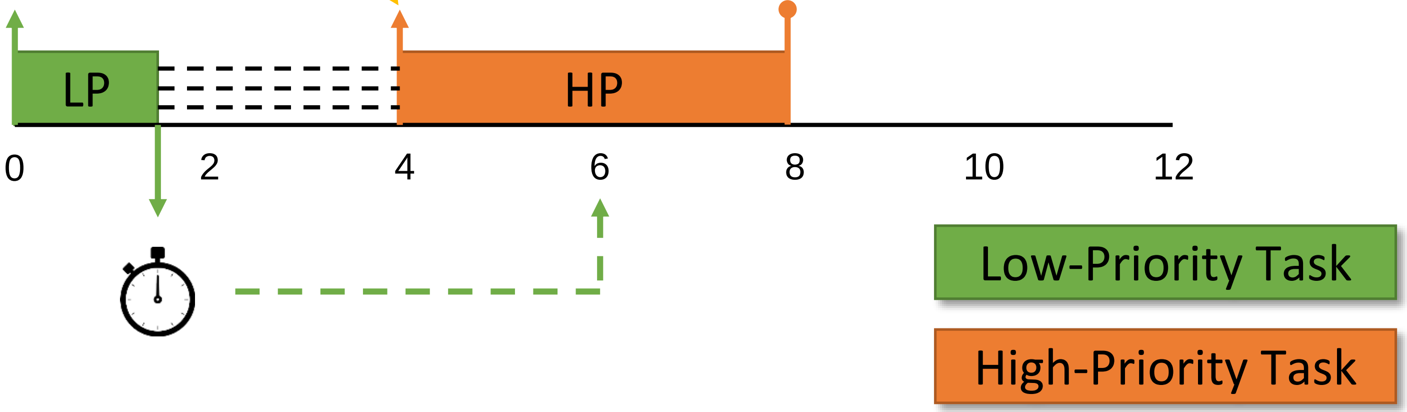


Mask all the low-priority timers

How Does TimerShield Work?

Mask all the low-priority timers

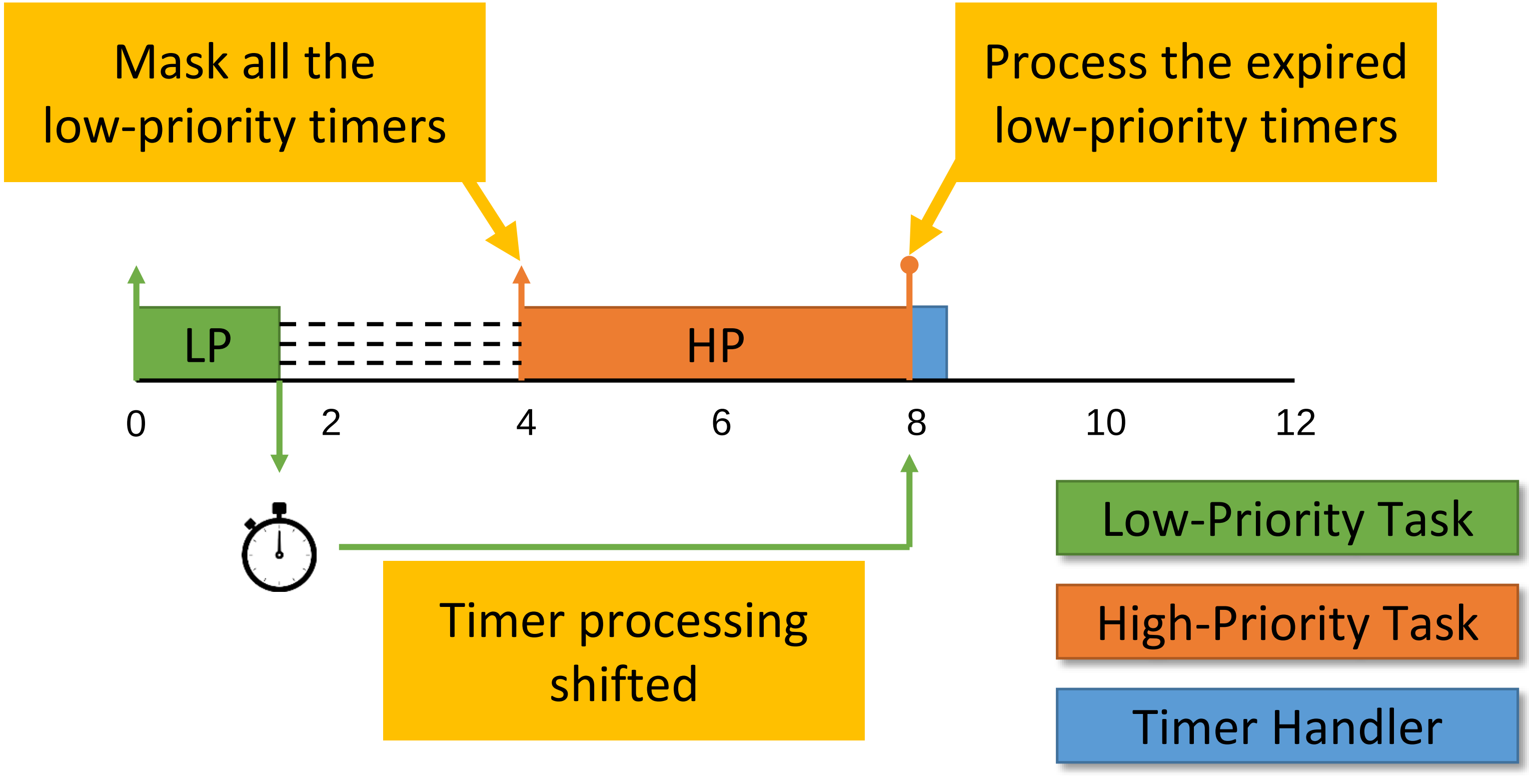
Process the expired low-priority timers



Low-Priority Task

High-Priority Task

How Does TimerShield Work?



Mask all the low-priority timers

Process the expired low-priority timers

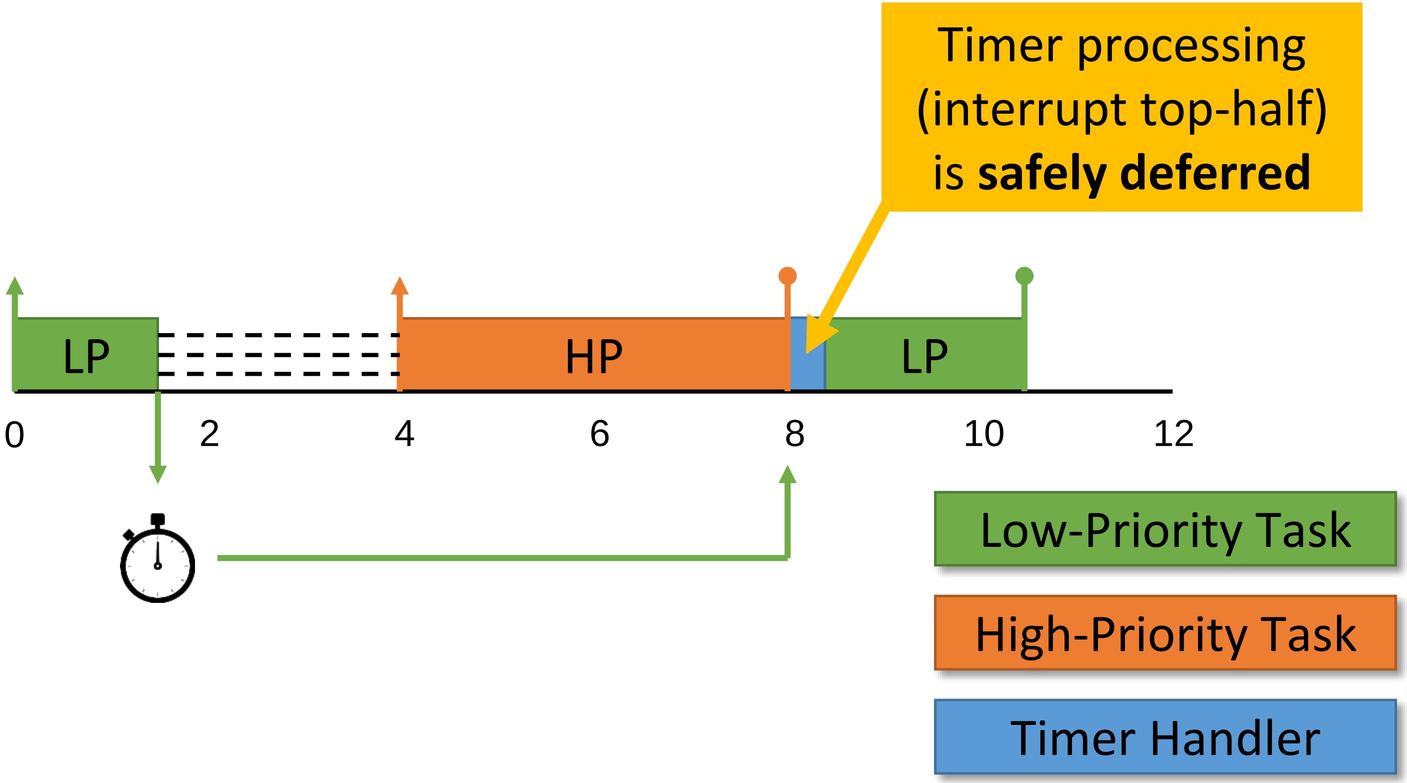
Timer processing shifted

🟩 Low-Priority Task

🟧 High-Priority Task

🟦 Timer Handler

How Does TimerShield Work?



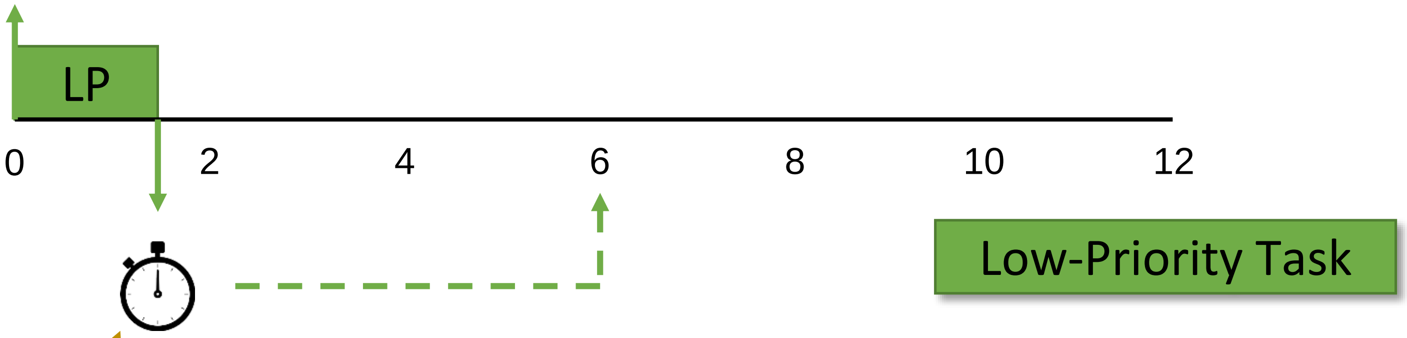
Timer processing (interrupt top-half) is **safely deferred**

Low-Priority Task

High-Priority Task

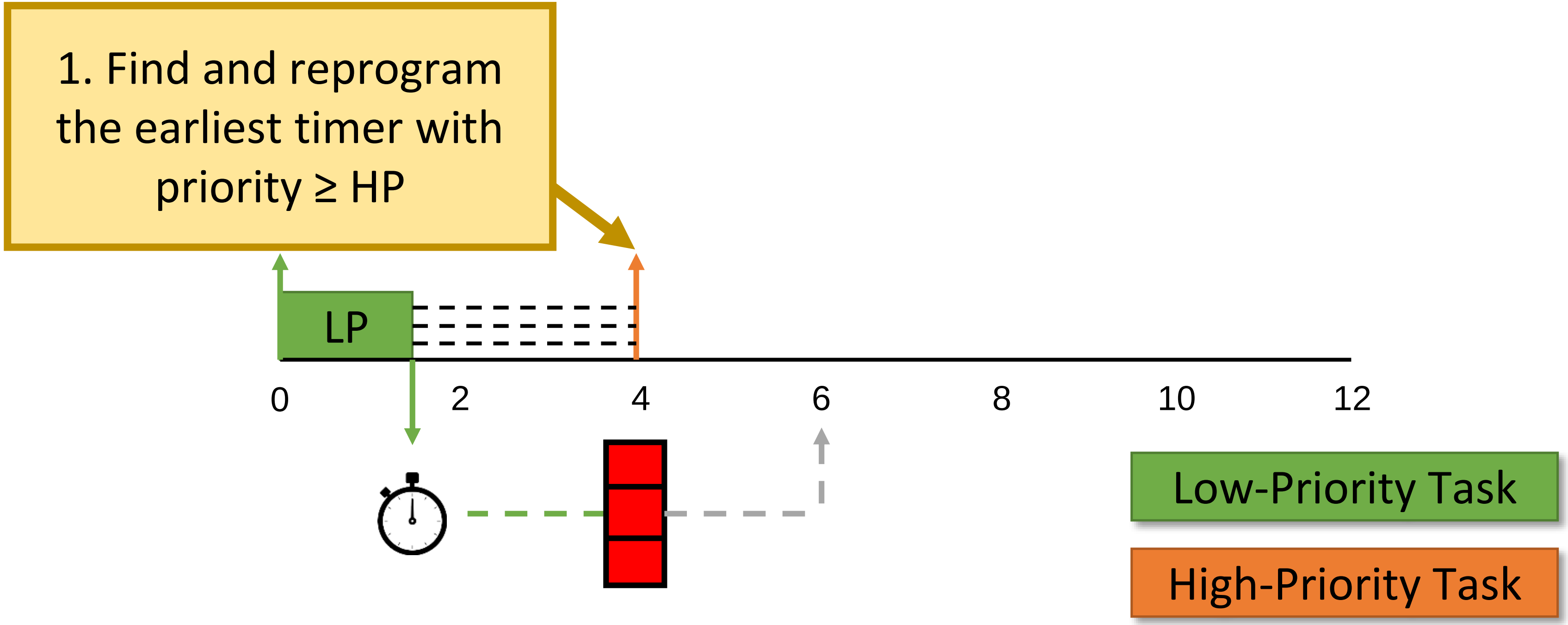
Timer Handler

How is TimerShield Implemented?



Timer inherits task priority

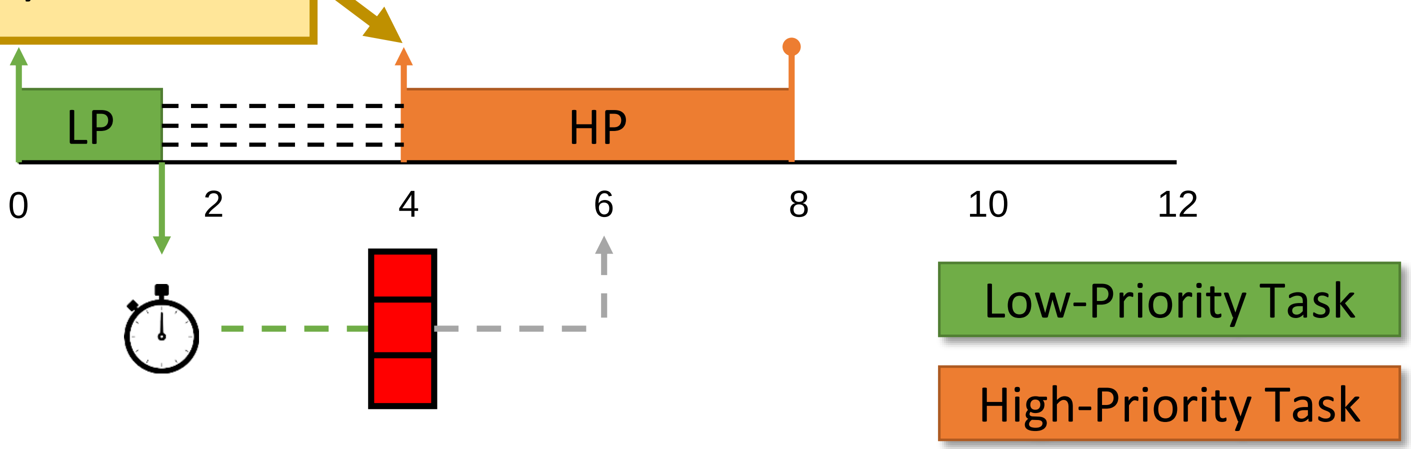
How is TimerShield Implemented?



1. Find and reprogram the earliest timer with priority ≥ HP

How is TimerShield Implemented?

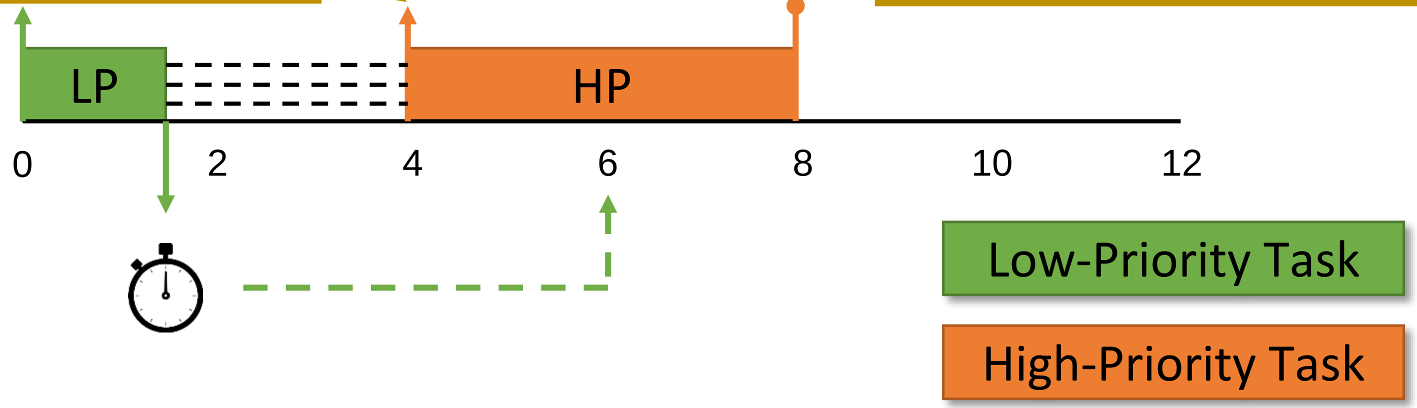
1. Find and reprogram the earliest timer with priority ≥ HP



How is TimerShield Implemented?

1. Find and reprogram the earliest timer with priority ≥ HP

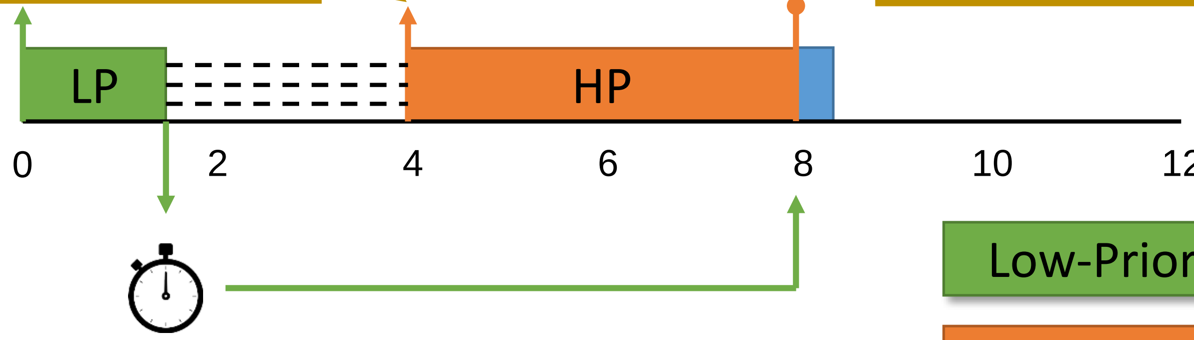
2. Process expired timers of the highest priority (lower priority ones can still be deferred)



How is TimerShield Implemented?

1. Find and reprogram the earliest timer with priority ≥ HP

2. Process expired timers of the highest priority (lower priority ones can still be deferred)



Low-Priority Task

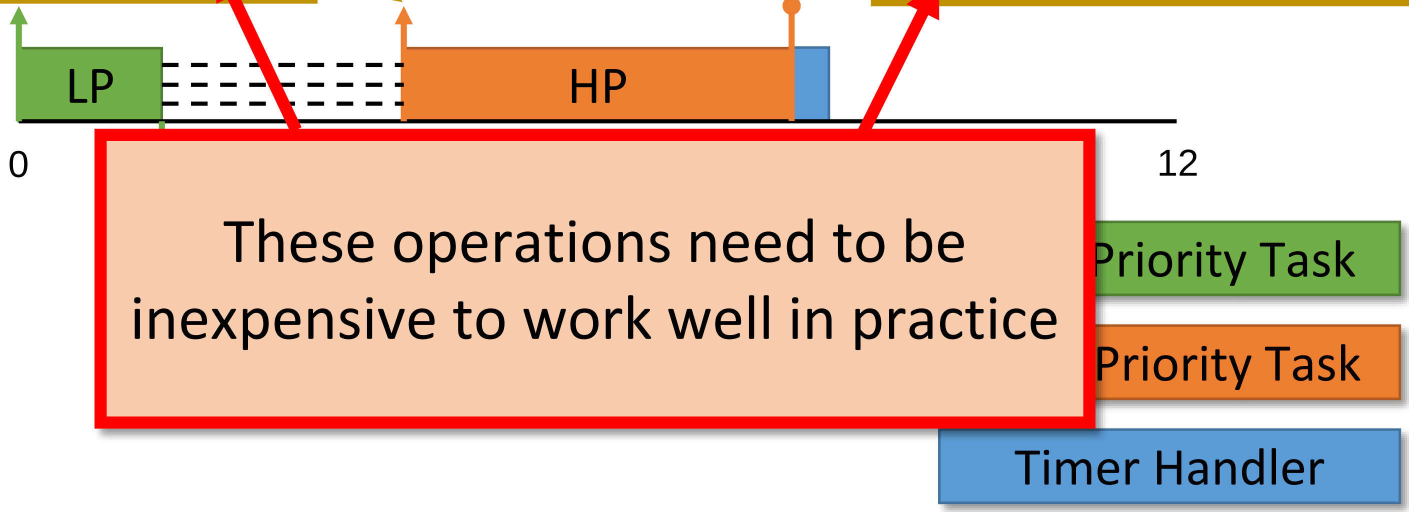
High-Priority Task

Timer Handler

How is TimerShield Implemented?

1. Find and reprogram the earliest timer with priority ≥ HP

2. Process expired timers of the highest priority (lower priority ones can still be deferred)



These operations need to be inexpensive to work well in practice

Priority-Based Earliest Timer

1: Find the earliest timer at each priority level

Priority-Based Earliest Timer

1: Find the earliest timer at each priority level

2: Among these, find the earliest timer in the priority range [curr_task_prio, max_system_prio]

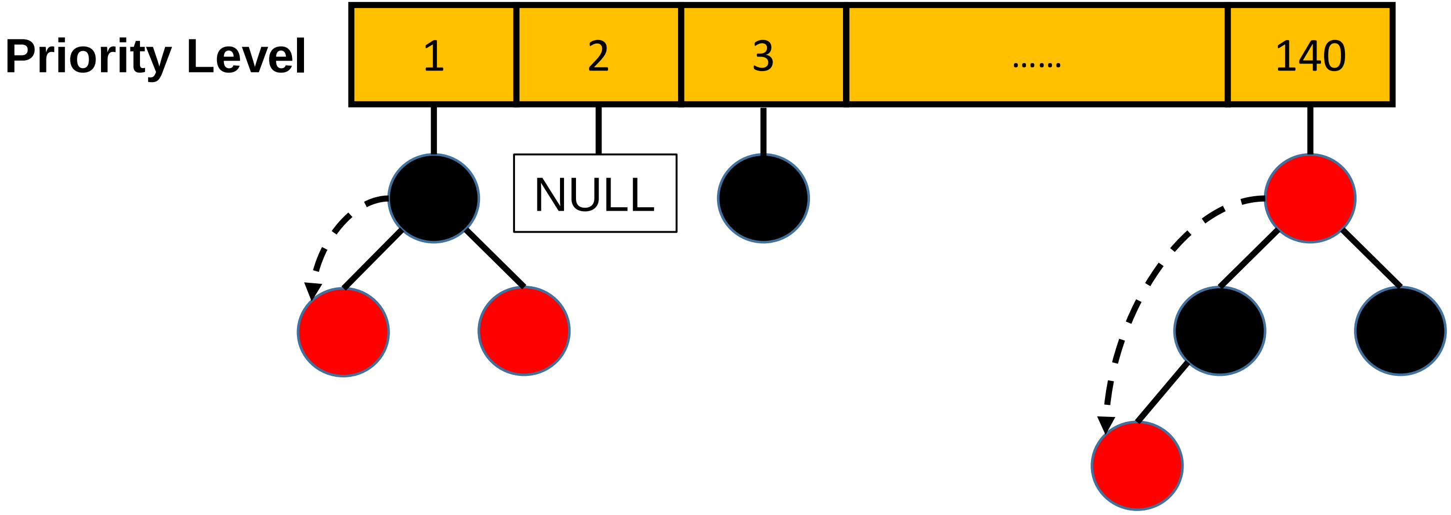
Priority-Based Earliest Timer

1: Find the earliest timer at each priority level

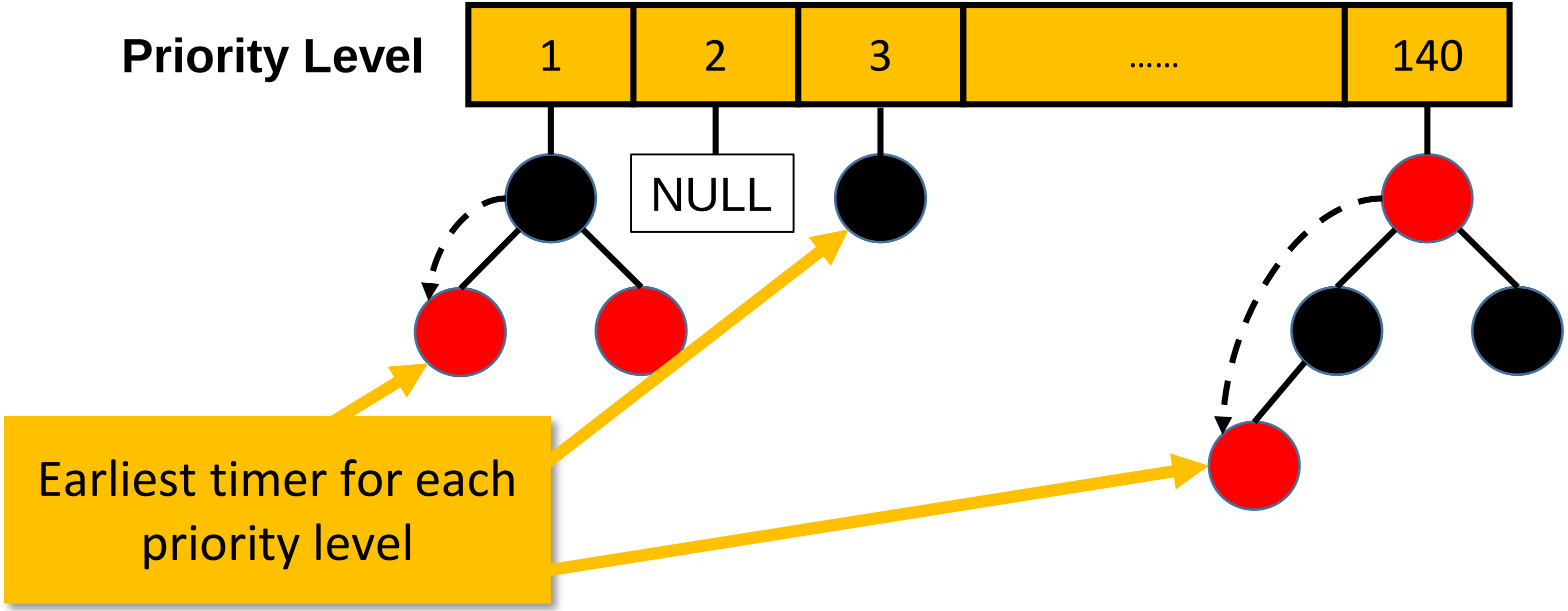
2: Among these, find the **earliest timer** in the priority **range [curr_task_prio, max_system_prio]**

A Range Minimum Query! (RMQ)

1: Replicating Red-Black Trees



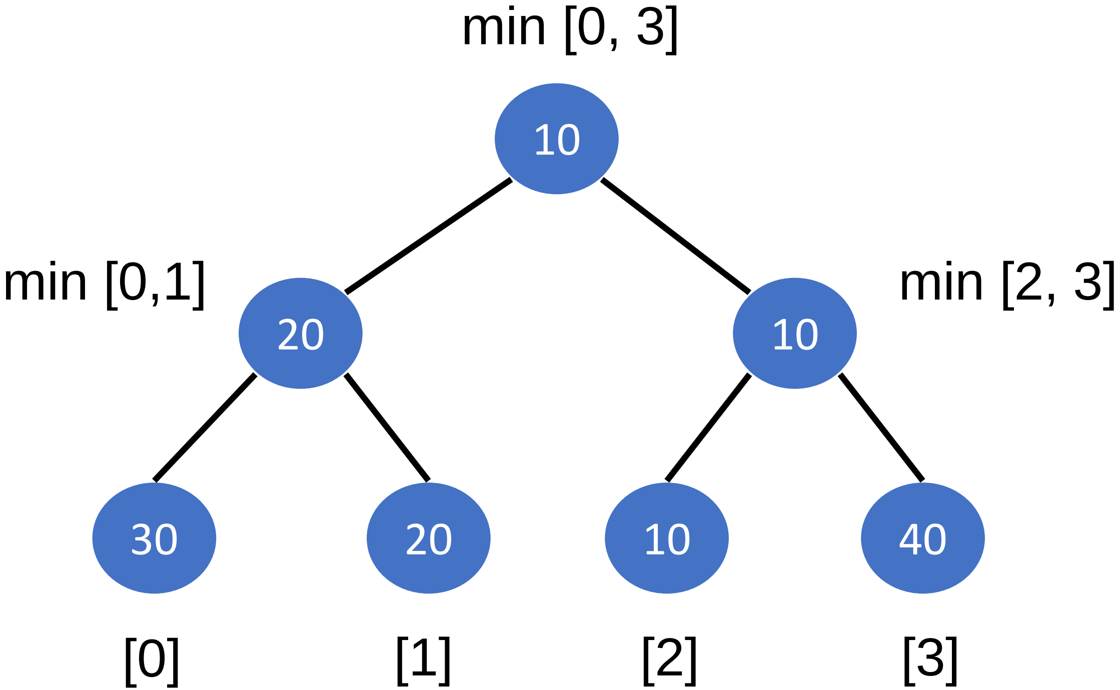
1: Replicating Red-Black Trees



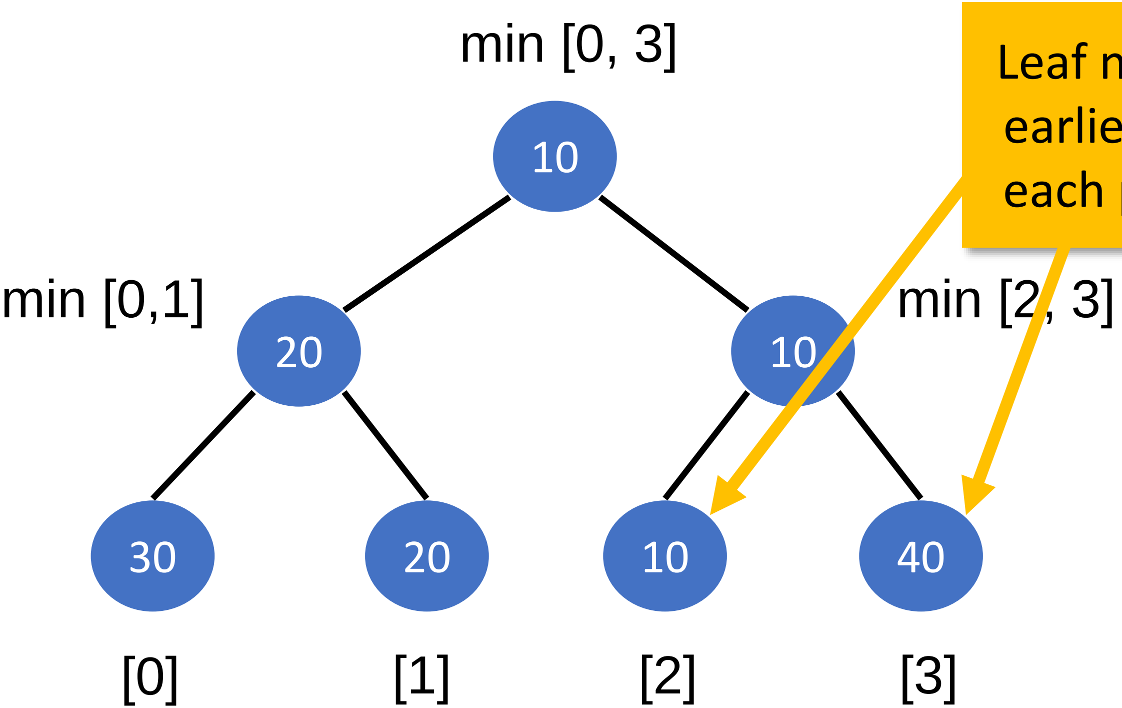
Priority Level	1	2	3	……	140

Earliest timer for each priority level

2: Range Minimum Query – Segment Tree

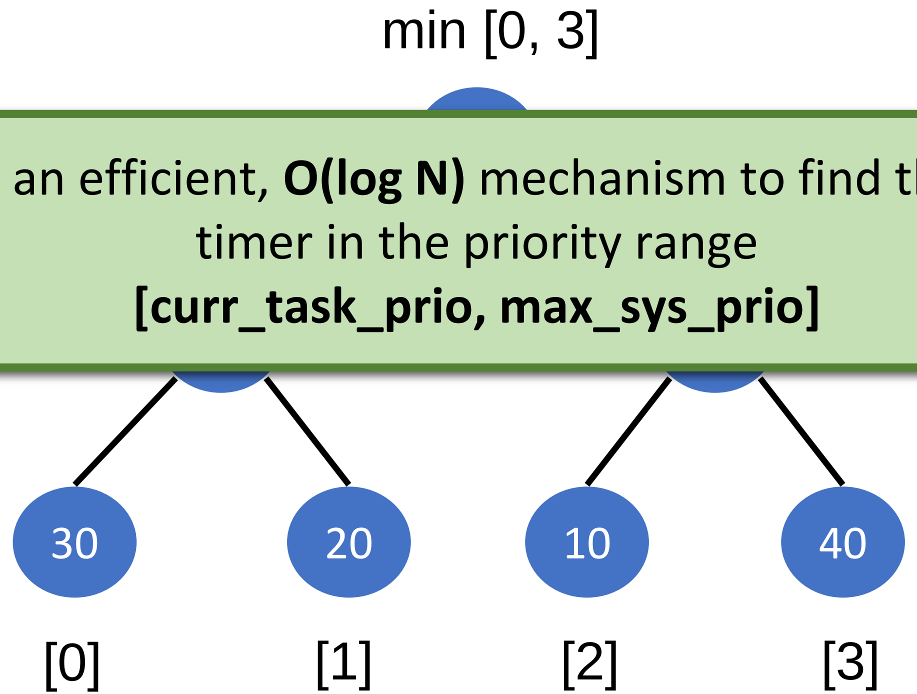


2: Range Minimum Query – Segment Tree



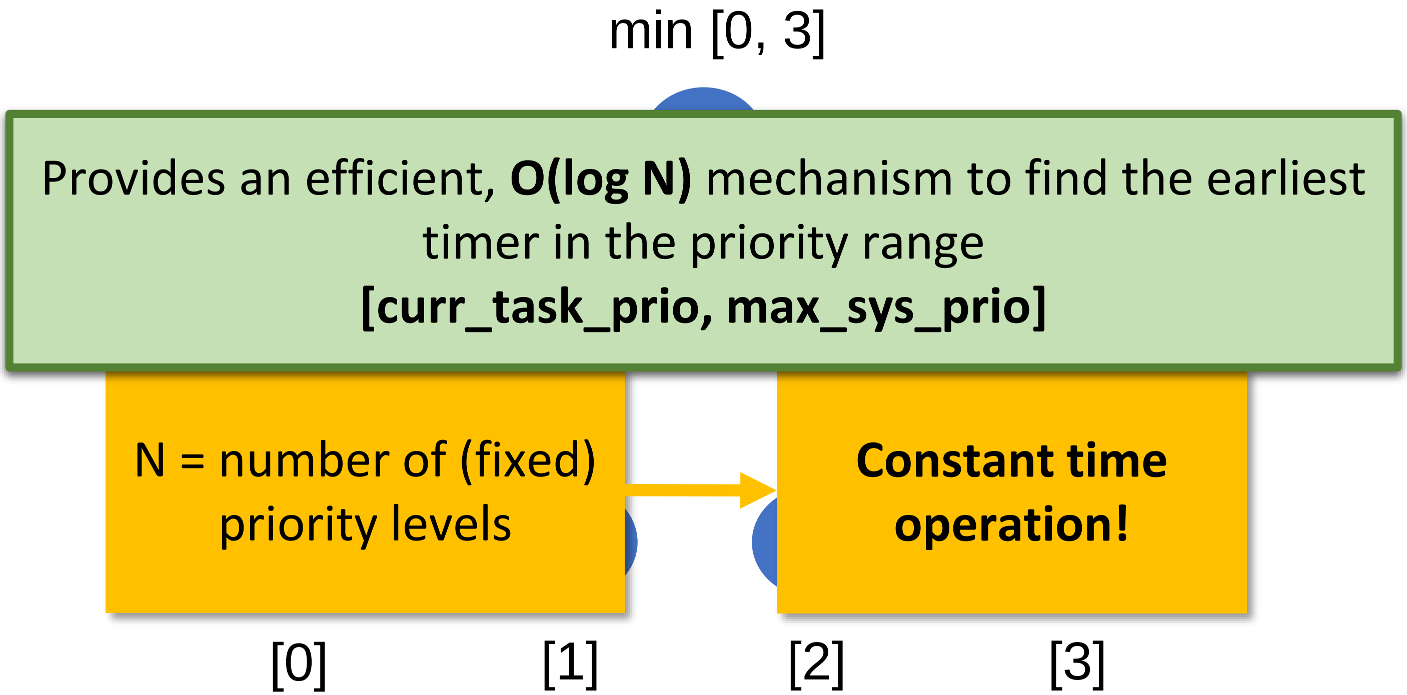
Leaf nodes are the earliest timers for each priority level

2: Range Minimum Query – Segment Tree



Provides an efficient, **O(log N)** mechanism to find the earliest timer in the priority range **[curr_task_prio, max_sys_prio]**

2: Range Minimum Query – Segment Tree



Provides an efficient, **O(log N)** mechanism to find the earliest timer in the priority range **[curr_task_prio, max_sys_prio]**

N = number of (fixed) priority levels

Constant time operation!

TimerShield Implementation

Further details in the paper!

Open-source implementation at
https://people.mpi-sws.org/~bbb/papers/details/rtas17p/

Talk Overview

Timers and the Interference Problem

TimerShield Design

Evaluation

Evaluation

Prototyped in
PREEMPT_RT

Intel Core-i5
4 x 3.2Ghz

ARM Cortex-A53
4 x 1.2Ghz

Evaluation

Prototyped in **PREEMPT_RT**

Intel Core-i5
4 x 3.2Ghz

Details in paper

ARM Cortex-A53
4 x 1.2Ghz

Evaluation

How effective is TimerShield at isolating high-priority tasks from low-priority timer interrupts?

How is the context-switch duration affected?

How costly are the new queueing data structures?

Evaluation

How effective is TimerShield at isolating high-priority tasks from low-priority timer interrupts?

How is the context-switch duration affected?

How costly are the new queueing data structures?

HP Task Response Time

We measured the response time of a high-priority task with varying number of low-priority, timer-using tasks

HP Task Response Time

1 KHz control loop with approx. 200µs computation time

We measured the response time of a **high-priority task** with varying number of low-priority, timer-using tasks

Taskset parameters based on S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmark for free,” in WATERS, 2015.

HP Task Response Time

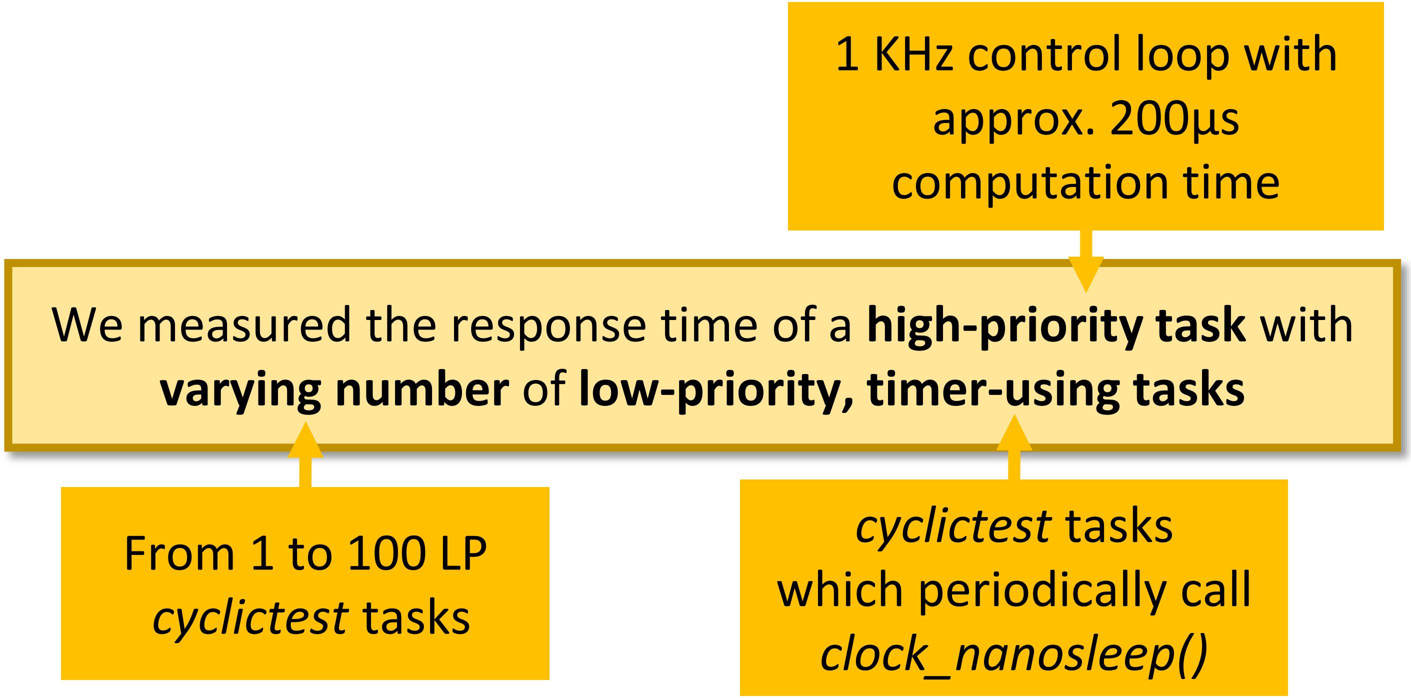
1 KHz control loop with approx. 200µs computation time

We measured the response time of a **high-priority task** with varying number of **low-priority, timer-using tasks**

cyclictest tasks which periodically call *clock_nanosleep()*

Taskset parameters based on S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmark for free,” in WATERS, 2015.

HP Task Response Time



1 KHz control loop with approx. 200µs computation time

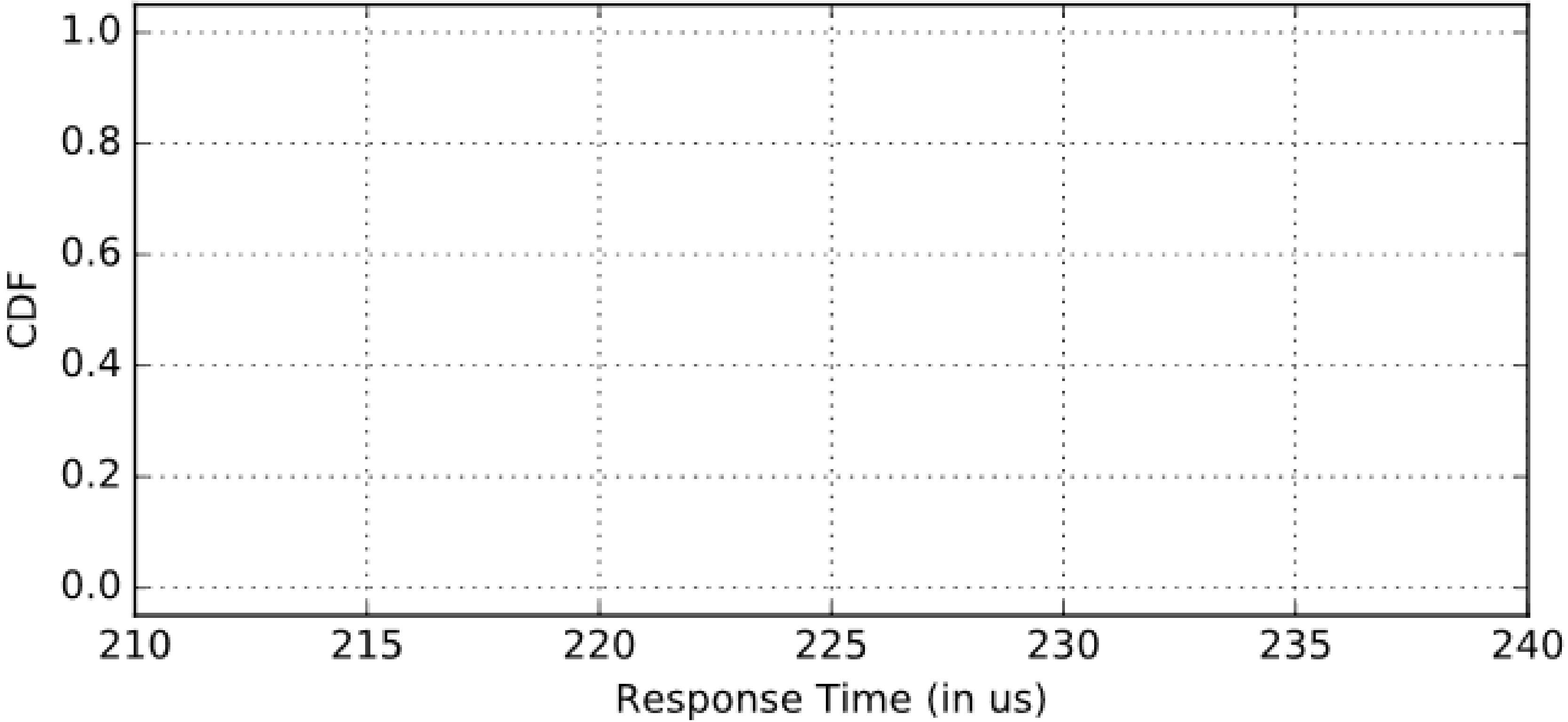
We measured the response time of a **high-priority task** with **varying number** of **low-priority, timer-using tasks**

From 1 to 100 LP *cyclictest* tasks

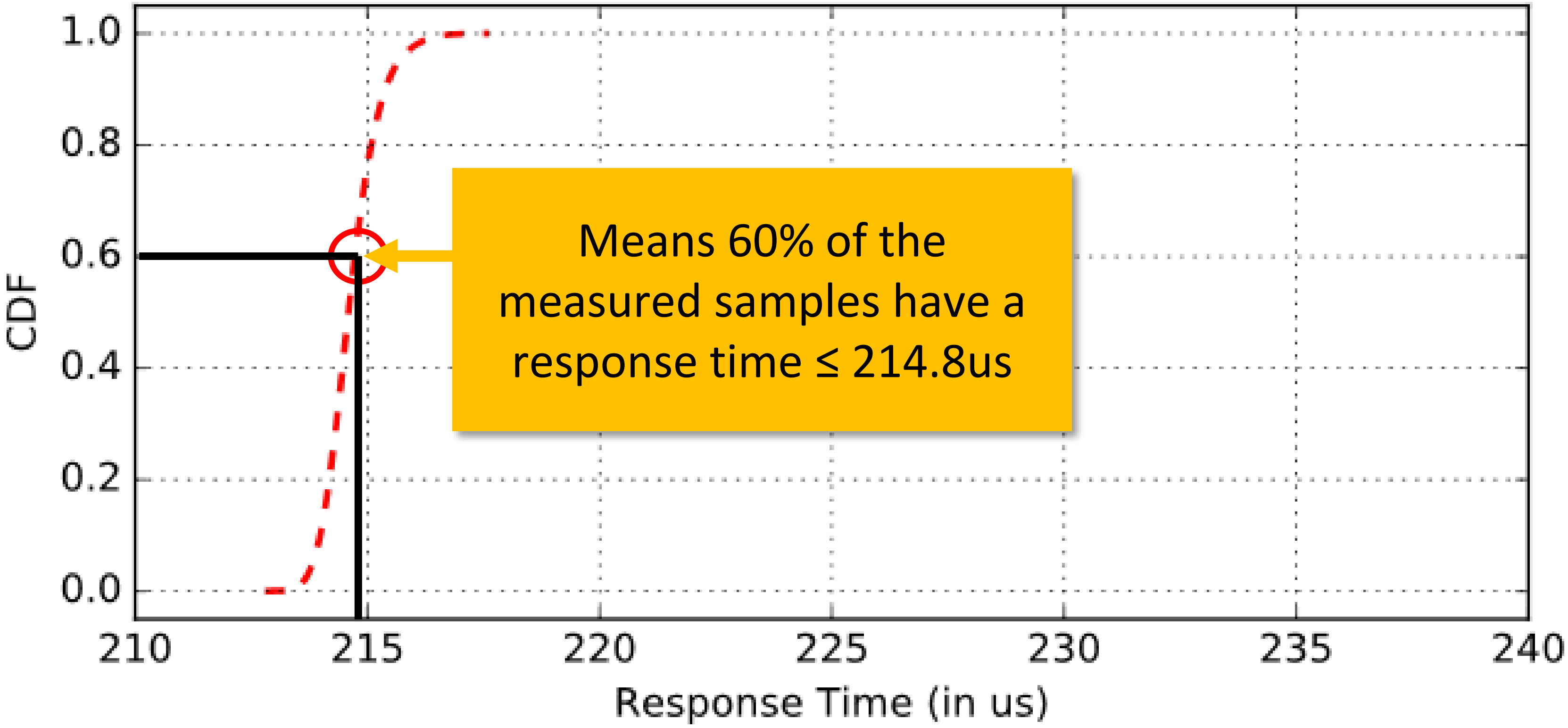
cyclictest tasks which periodically call *clock_nanosleep()*

Taskset parameters based on S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmark for free,” in WATERS, 2015.

HP Task Response Time

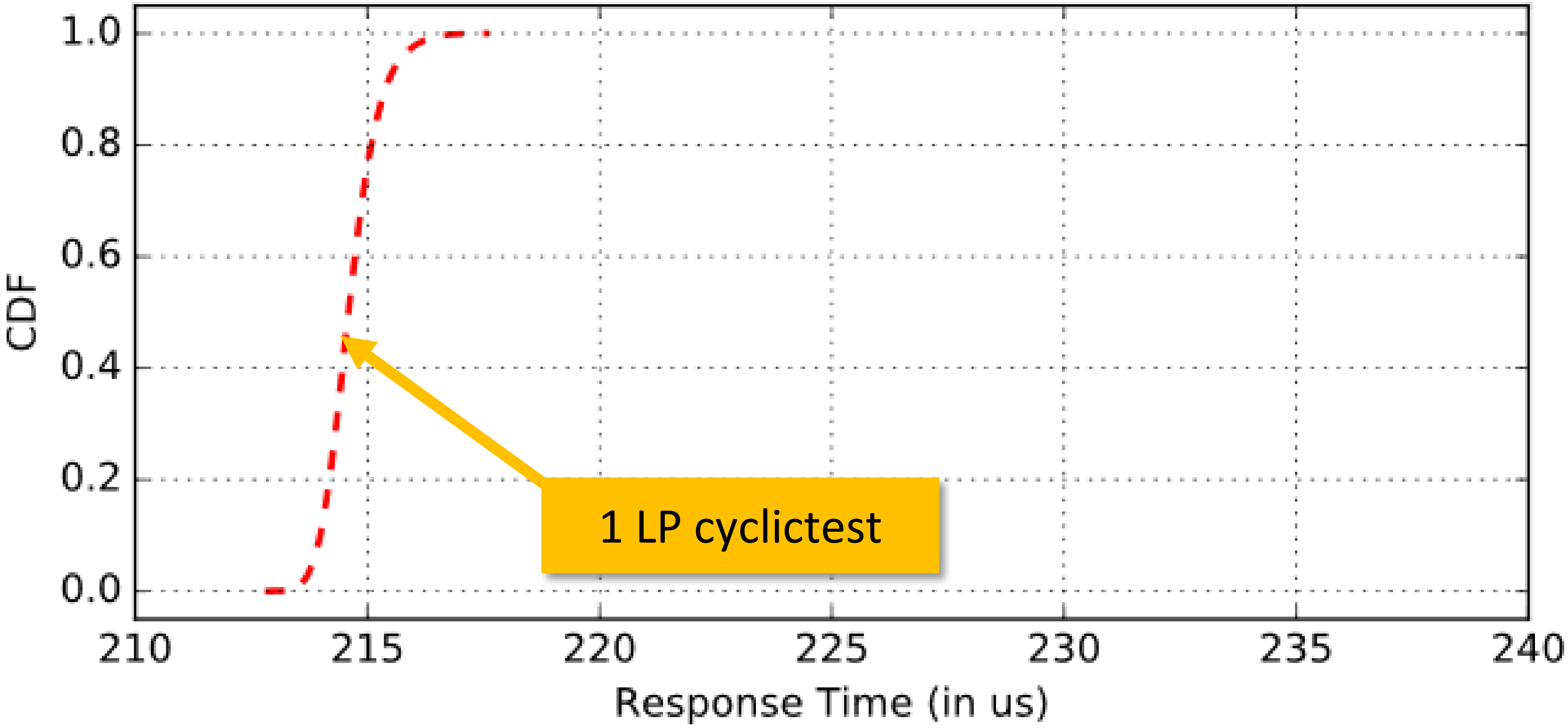


HP Task Response Time



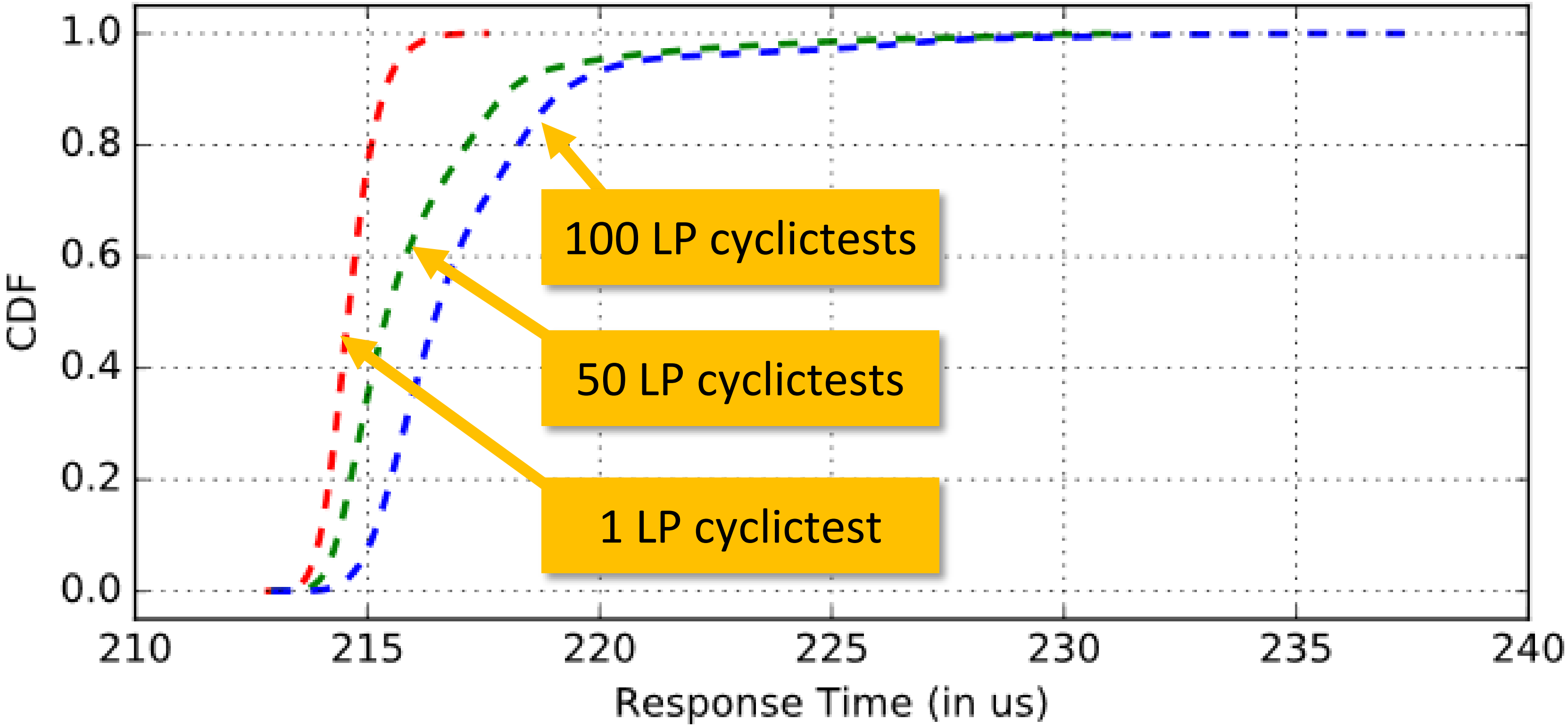
Means 60% of the measured samples have a response time ≤ 214.8us

Response Time - hrtimers



1 LP cyclictest

Response Time - hrtimers

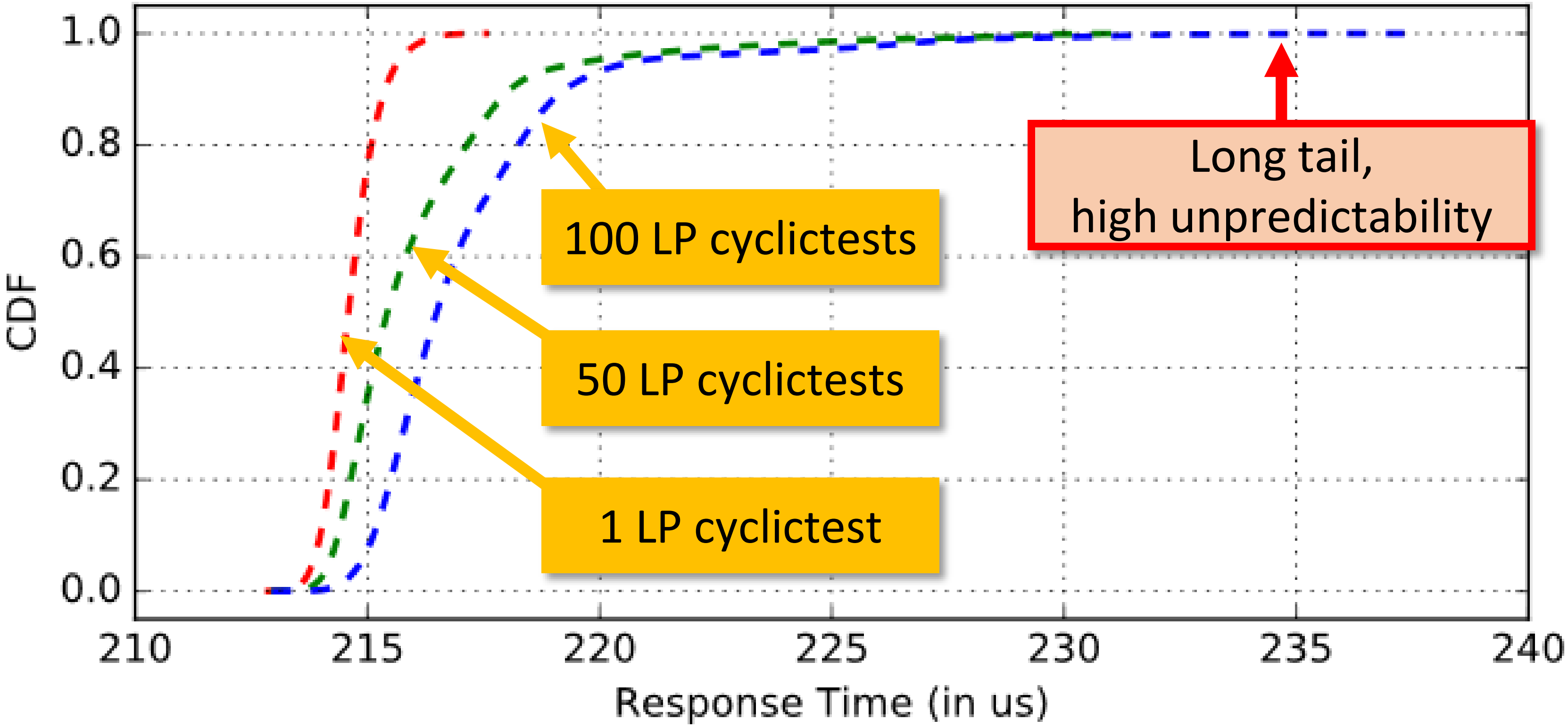


100 LP cyclictests

50 LP cyclictests

1 LP cyclictest

Response Time - hrtimers



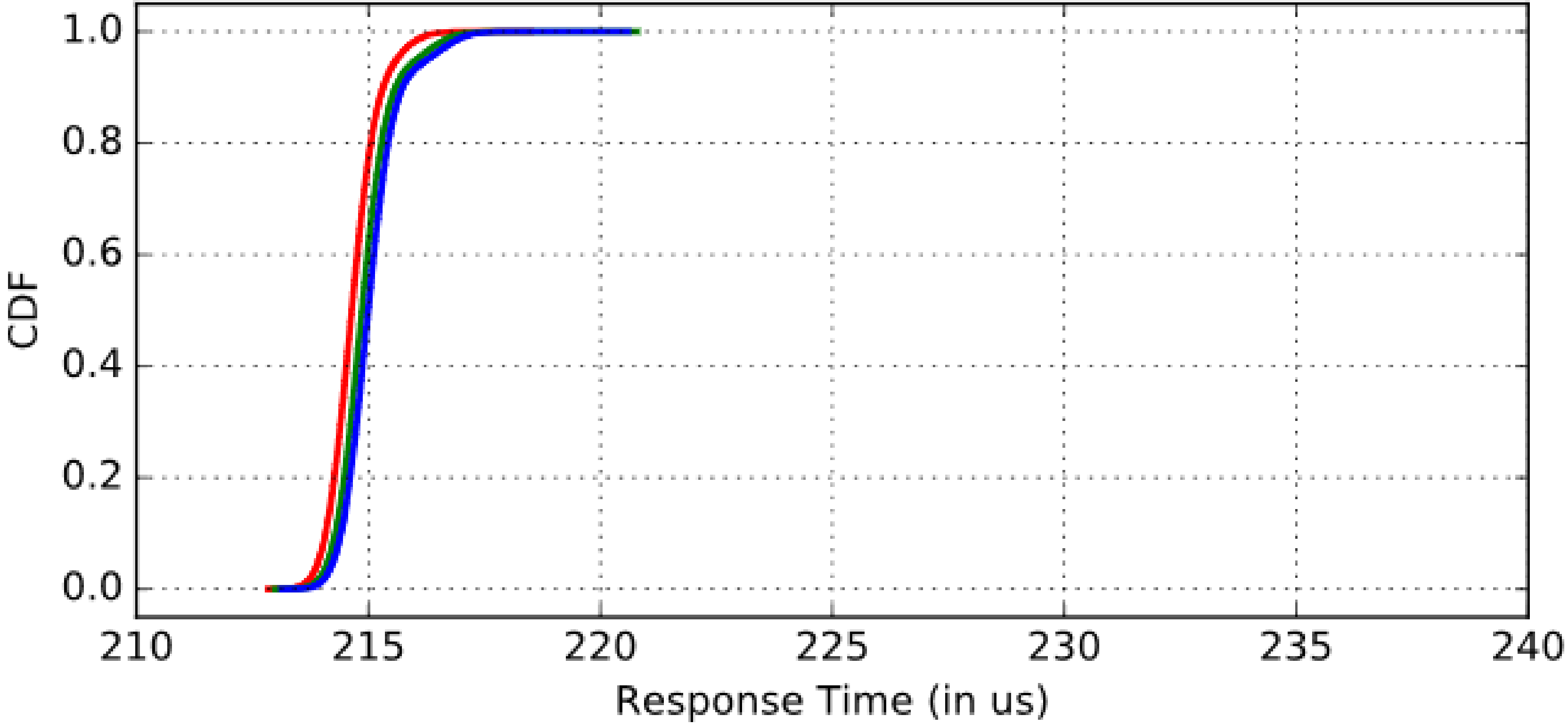
100 LP cyclictests

50 LP cyclictests

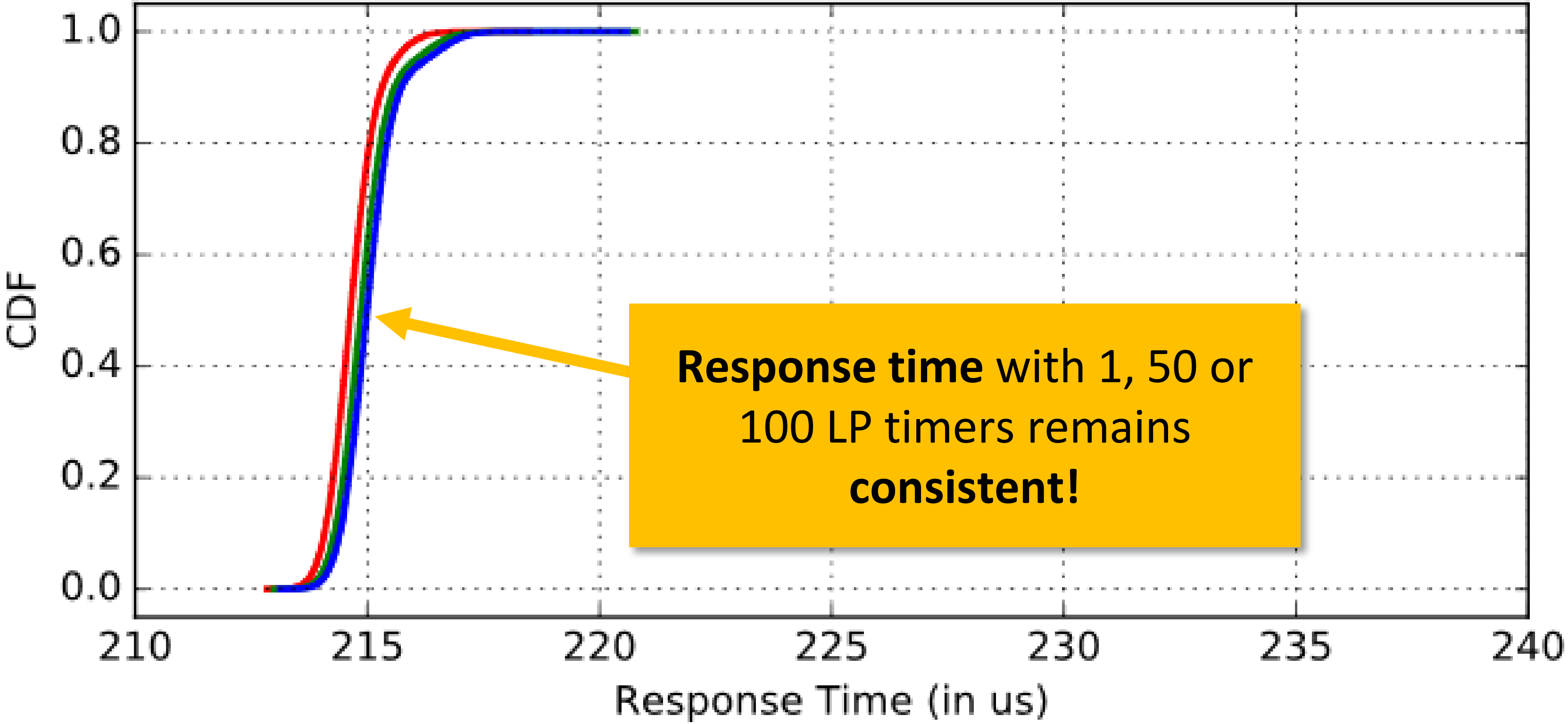
1 LP cyclictest

Long tail,
high unpredictability

Response Time - TimerShield

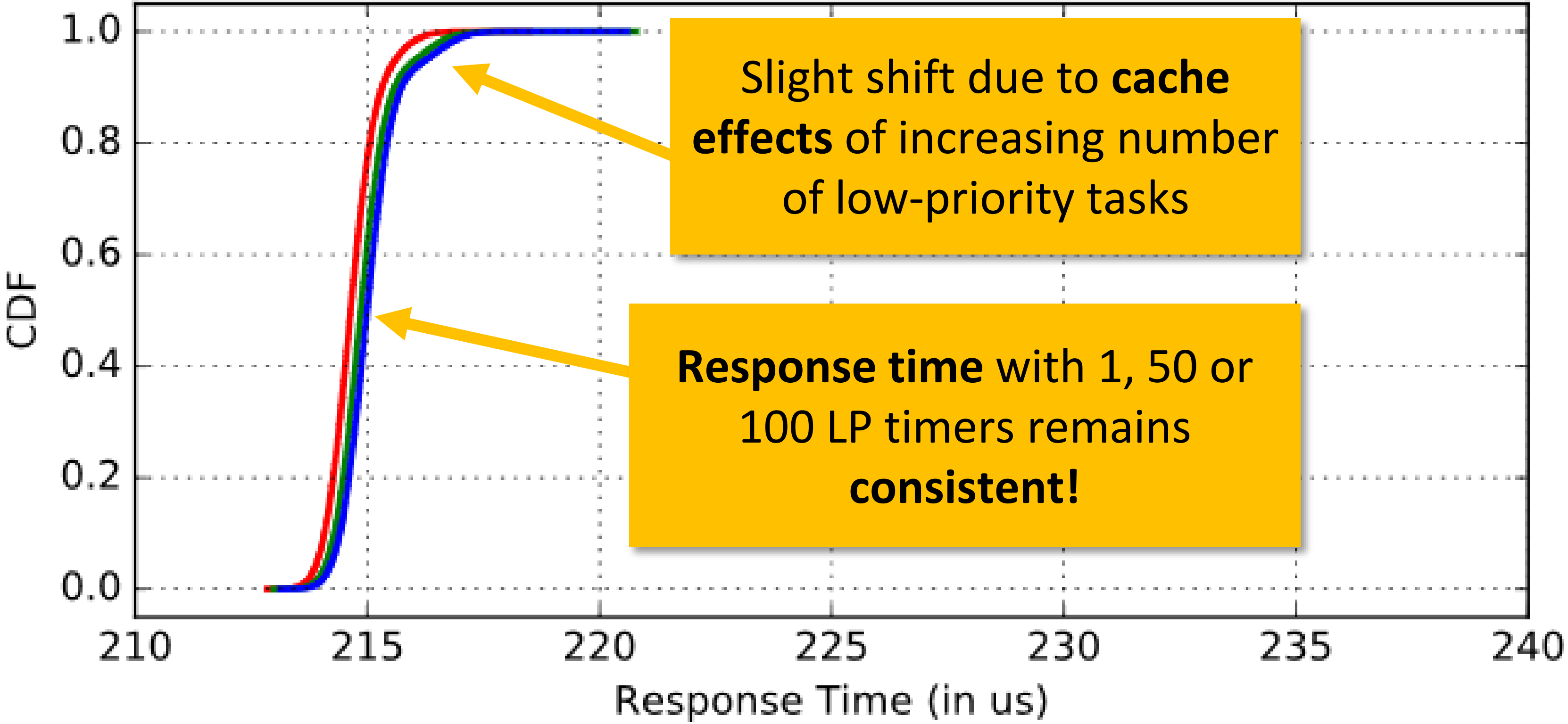


Response Time - TimerShield



Response time with 1, 50 or 100 LP timers remains **consistent!**

Response Time - TimerShield



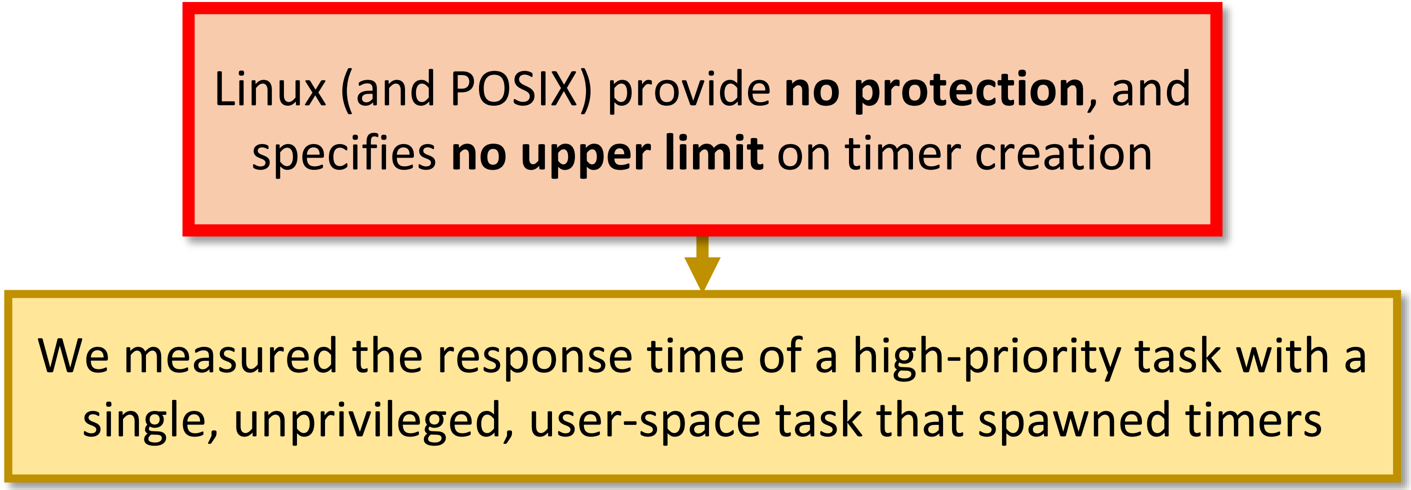
Slight shift due to **cache effects** of increasing number of low-priority tasks

Response time with 1, 50 or 100 LP timers remains **consistent!**

How Bad Can It Get?

Linux (and POSIX) provide **no protection**, and specifies **no upper limit** on timer creation

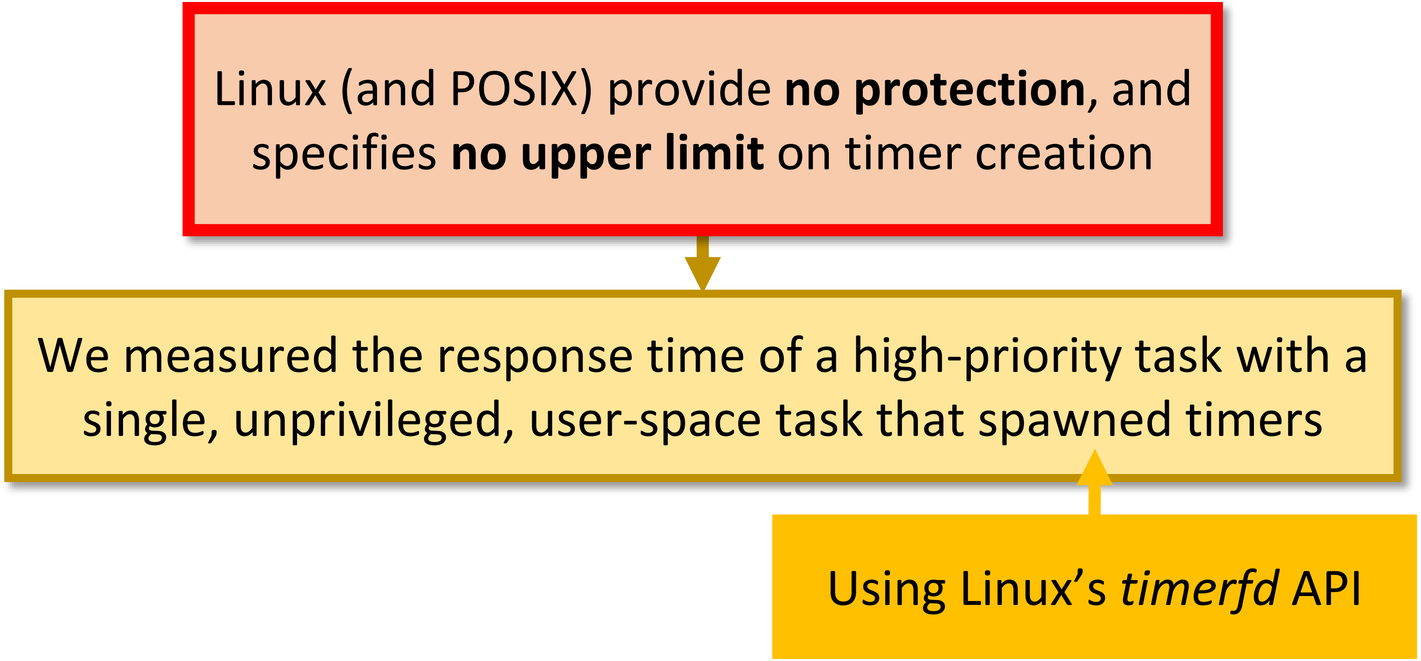
How Bad Can It Get?



Linux (and POSIX) provide **no protection**, and specifies **no upper limit** on timer creation

We measured the response time of a high-priority task with a single, unprivileged, user-space task that spawned timers

How Bad Can It Get?



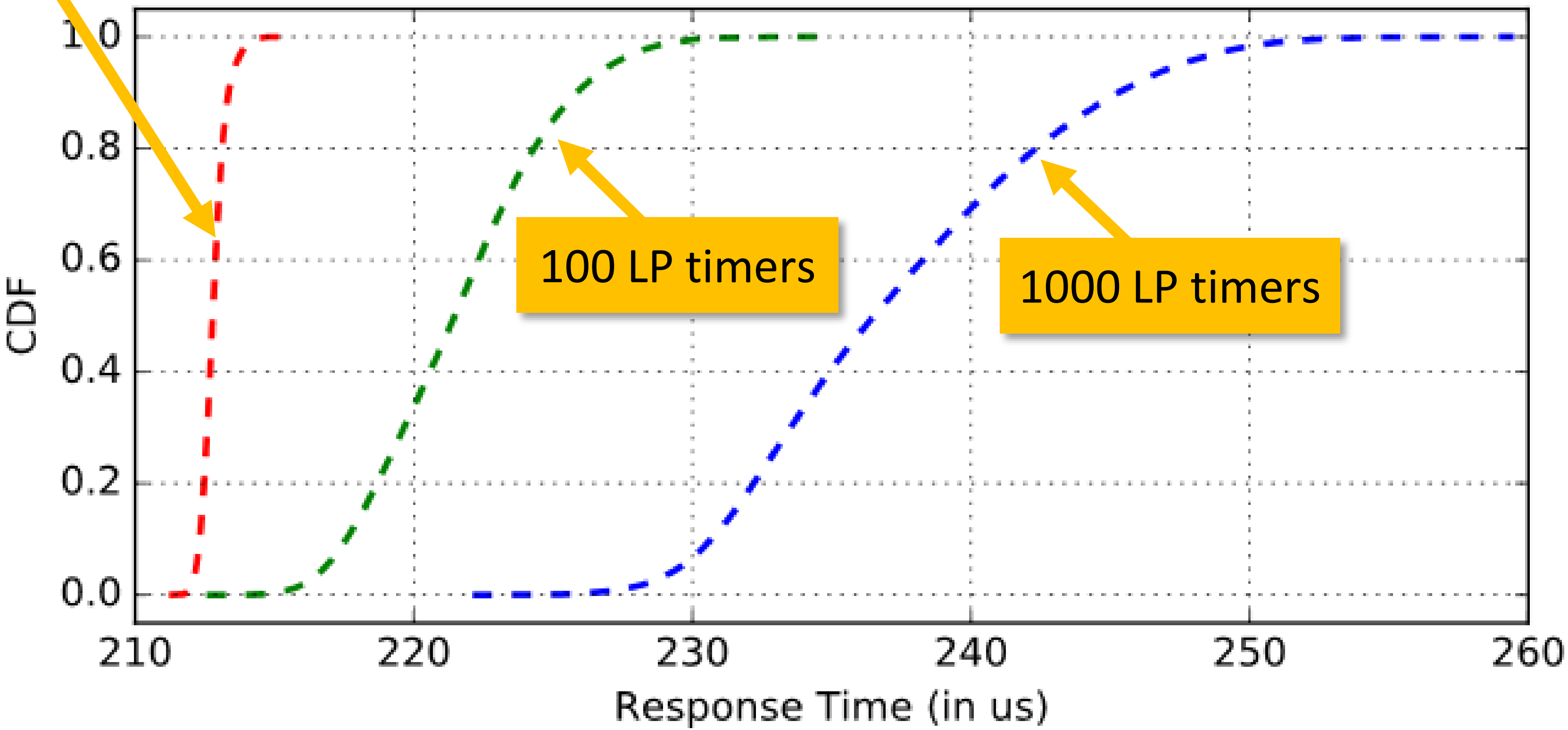
Linux (and POSIX) provide **no protection**, and specifies **no upper limit** on timer creation

We measured the response time of a high-priority task with a single, unprivileged, user-space task that spawned timers

Using Linux’s *timerfd* API

esponse Time - `hrtimers`

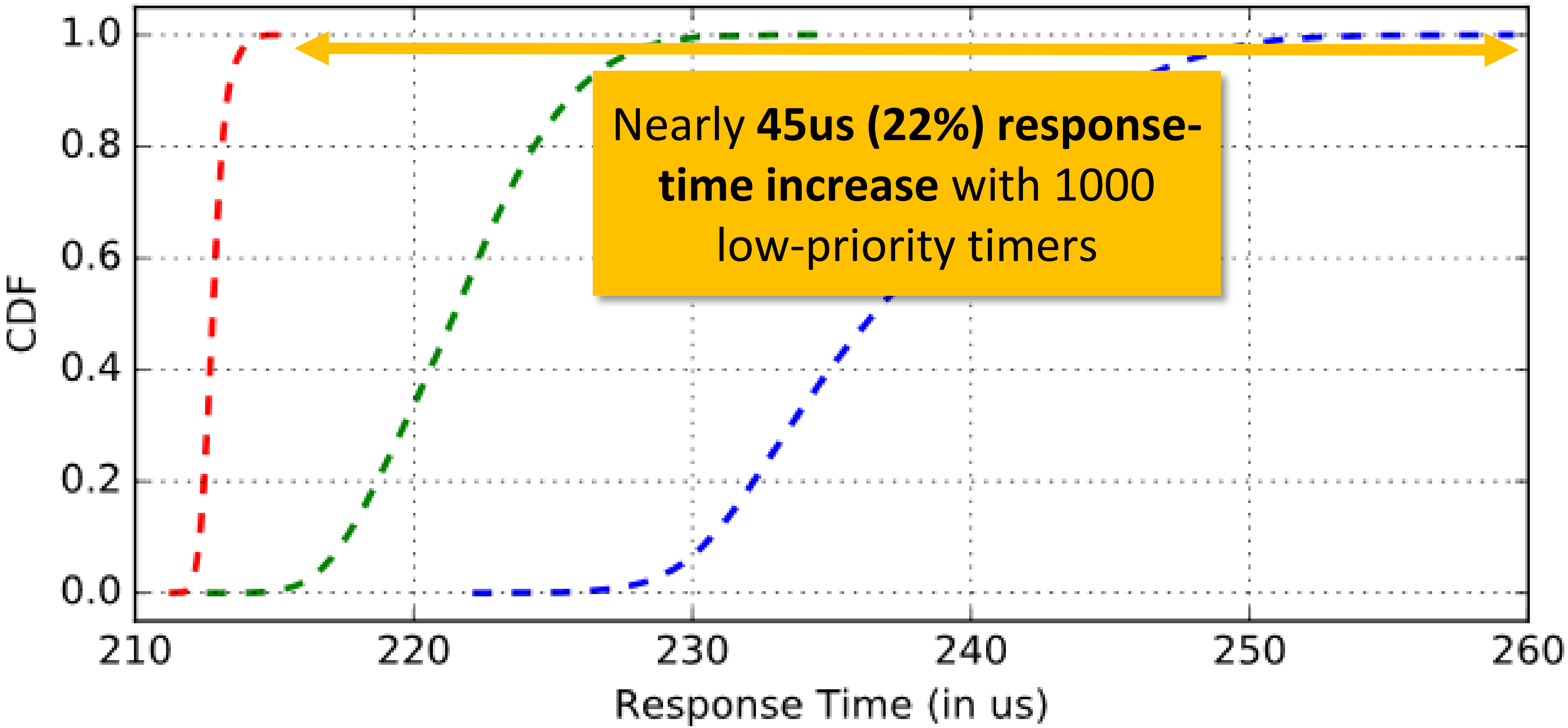
Idle system



100 LP timers

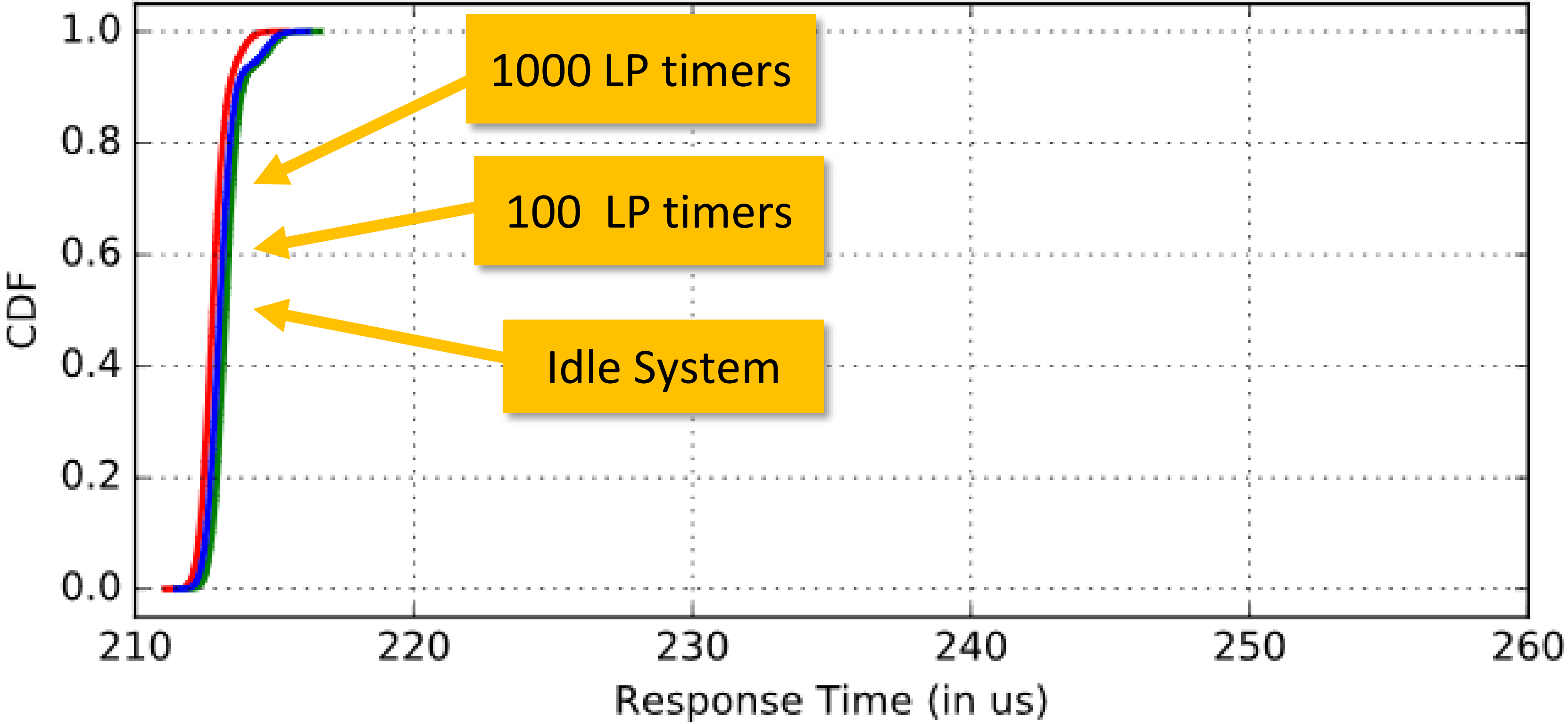
1000 LP timers

Response Time - `hrtimers`



Nearly **45us (22%) response-time increase** with 1000 low-priority timers

Response Time - TimerShield

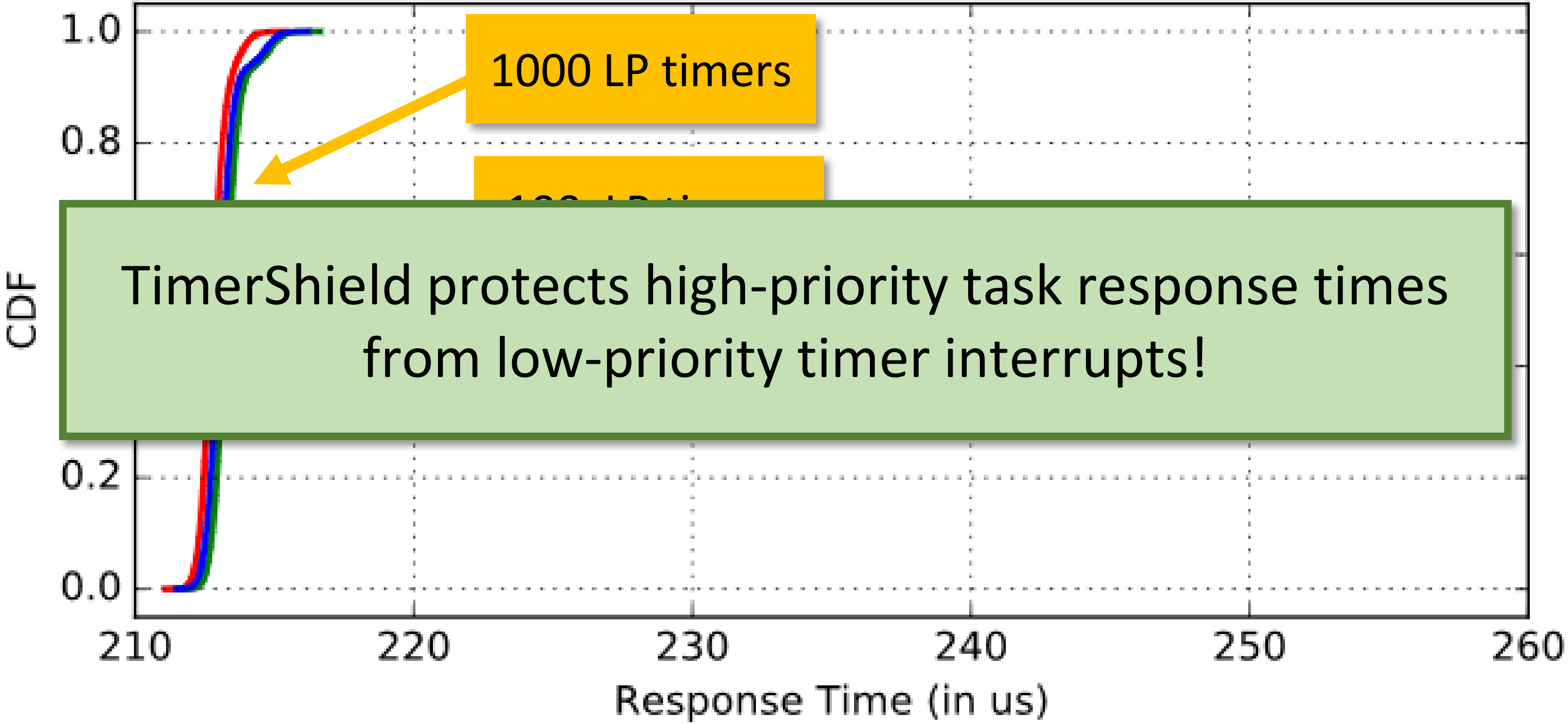


1000 LP timers

100 LP timers

Idle System

Response Time - TimerShield



1000 LP timers

TimerShield protects high-priority task response times from low-priority timer interrupts!

Evaluation

How effective is TimerShield at isolating high-priority tasks from low-priority timer interrupts?

How is the context-switch duration affected?

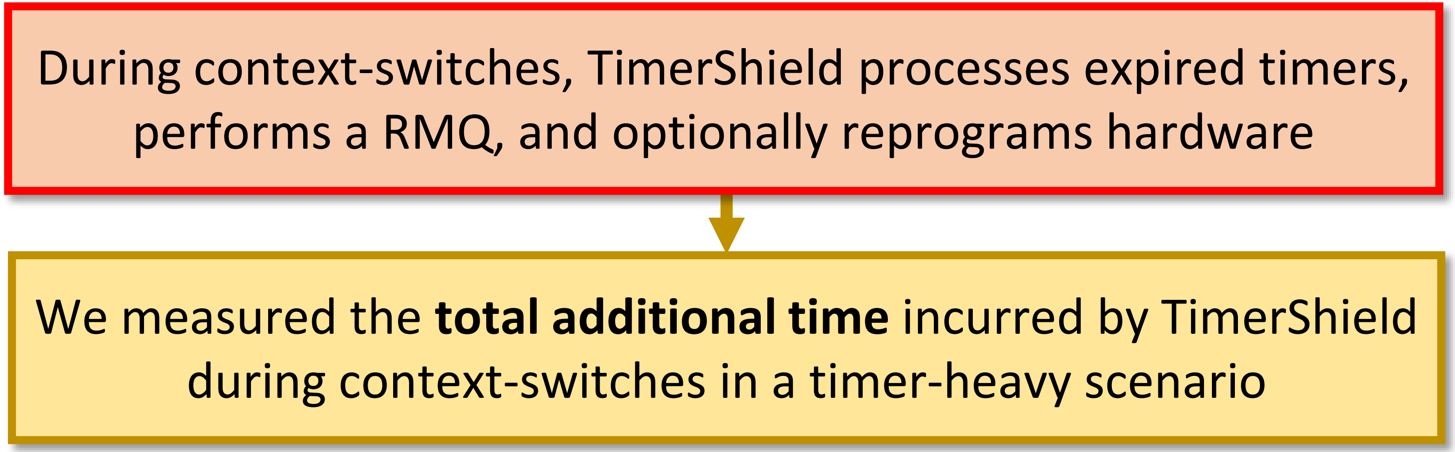
How costly are the new queueing data structures?

Additional Context-Switch Delay

During context-switches, TimerShield processes expired timers, performs a RMQ, and optionally reprograms hardware

Note: Results for a scenario without a timer-heavy load can be found in the paper.

Additional Context-Switch Delay

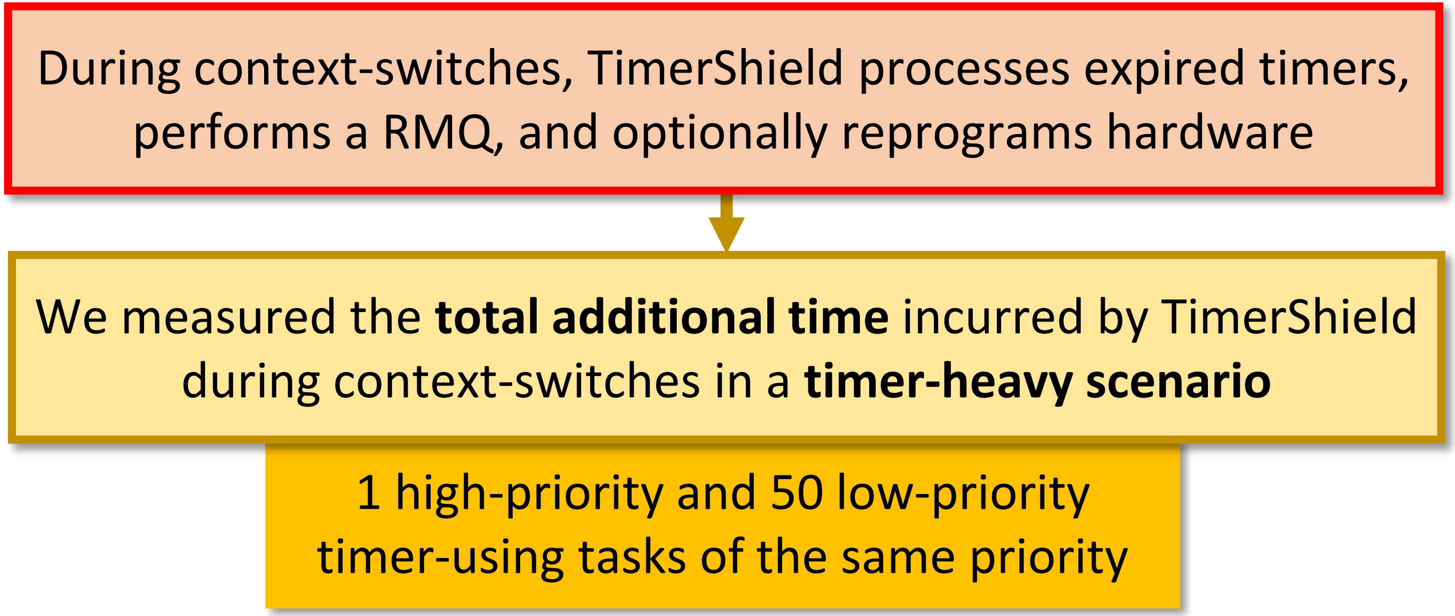


During context-switches, TimerShield processes expired timers, performs a RMQ, and optionally reprograms hardware

We measured the **total additional time** incurred by TimerShield during context-switches in a timer-heavy scenario

Note: Results for a scenario without a timer-heavy load can be found in the paper.

Additional Context-Switch Delay



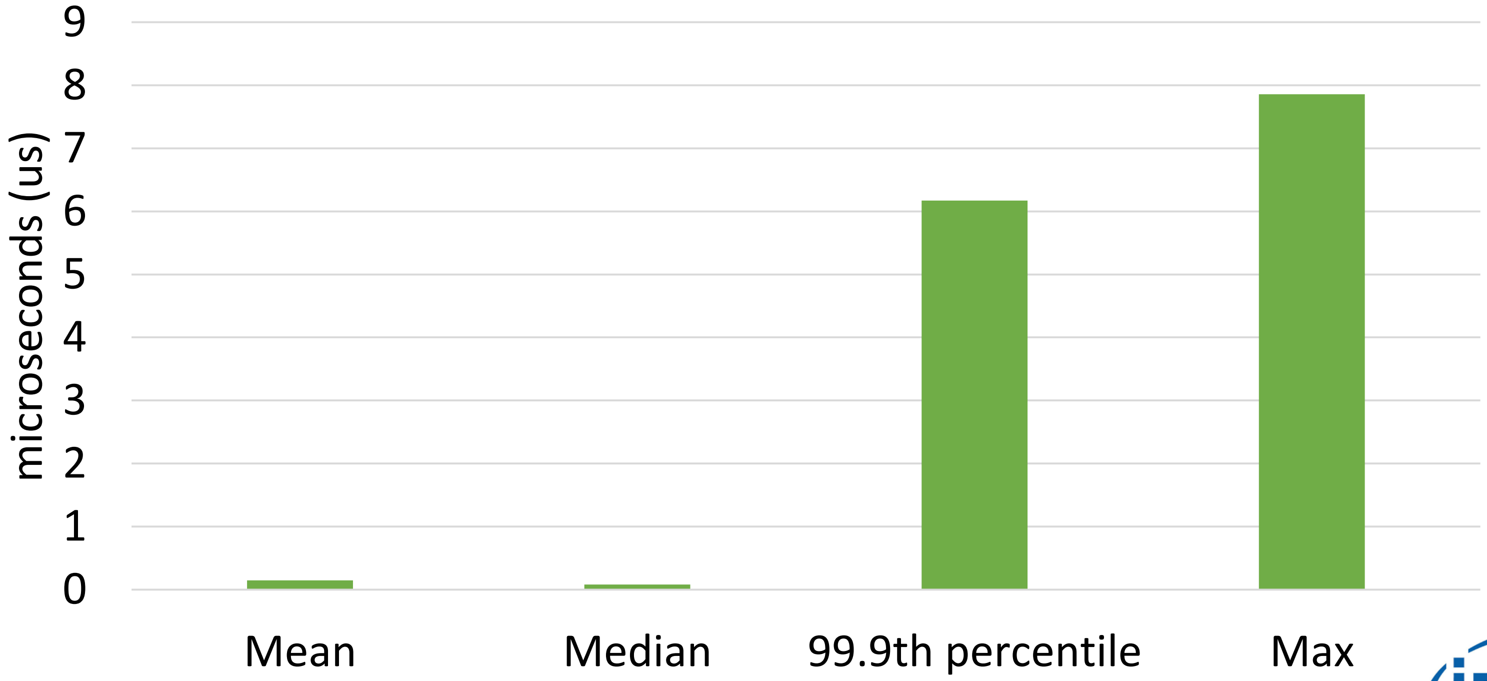
During context-switches, TimerShield processes expired timers, performs a RMQ, and optionally reprograms hardware

We measured the **total additional time** incurred by TimerShield during context-switches in a **timer-heavy scenario**

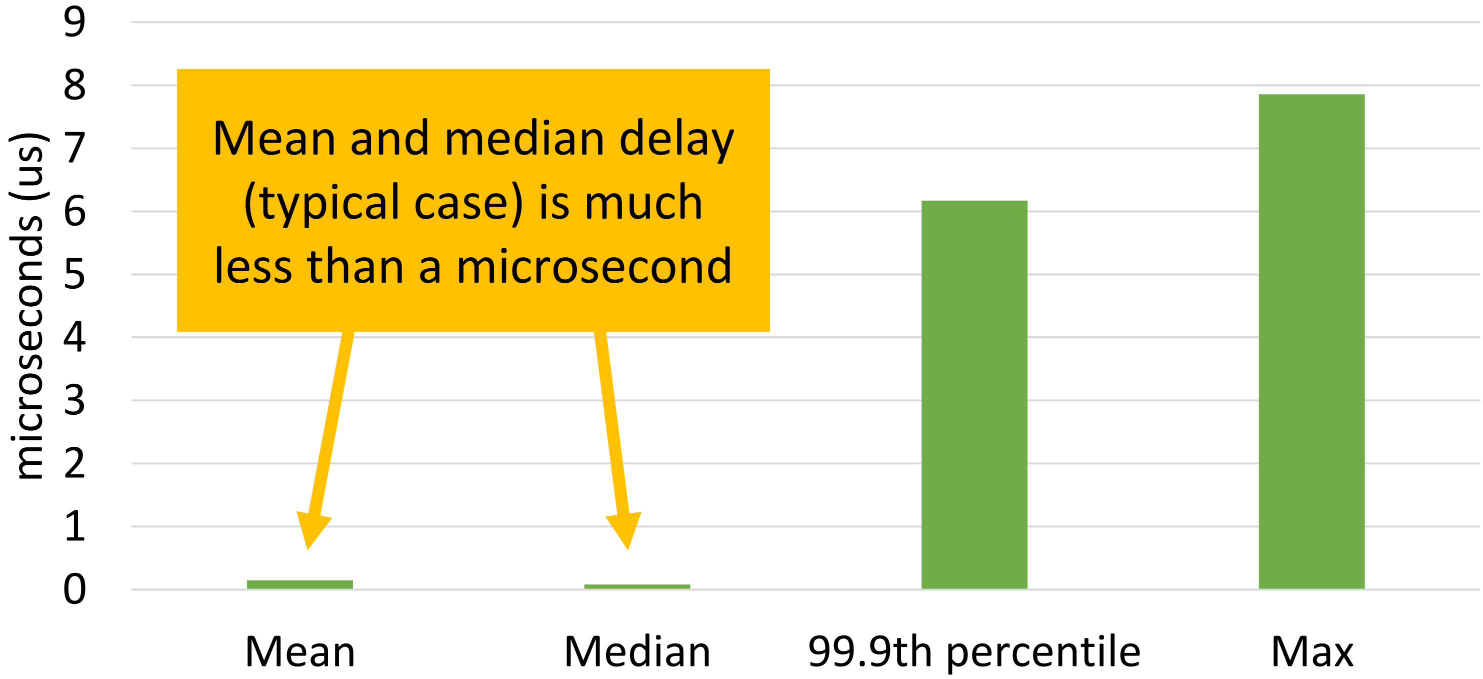
1 high-priority and 50 low-priority timer-using tasks of the same priority

Note: Results for a scenario without a timer-heavy load can be found in the paper.

Additional Context-Switch Delay

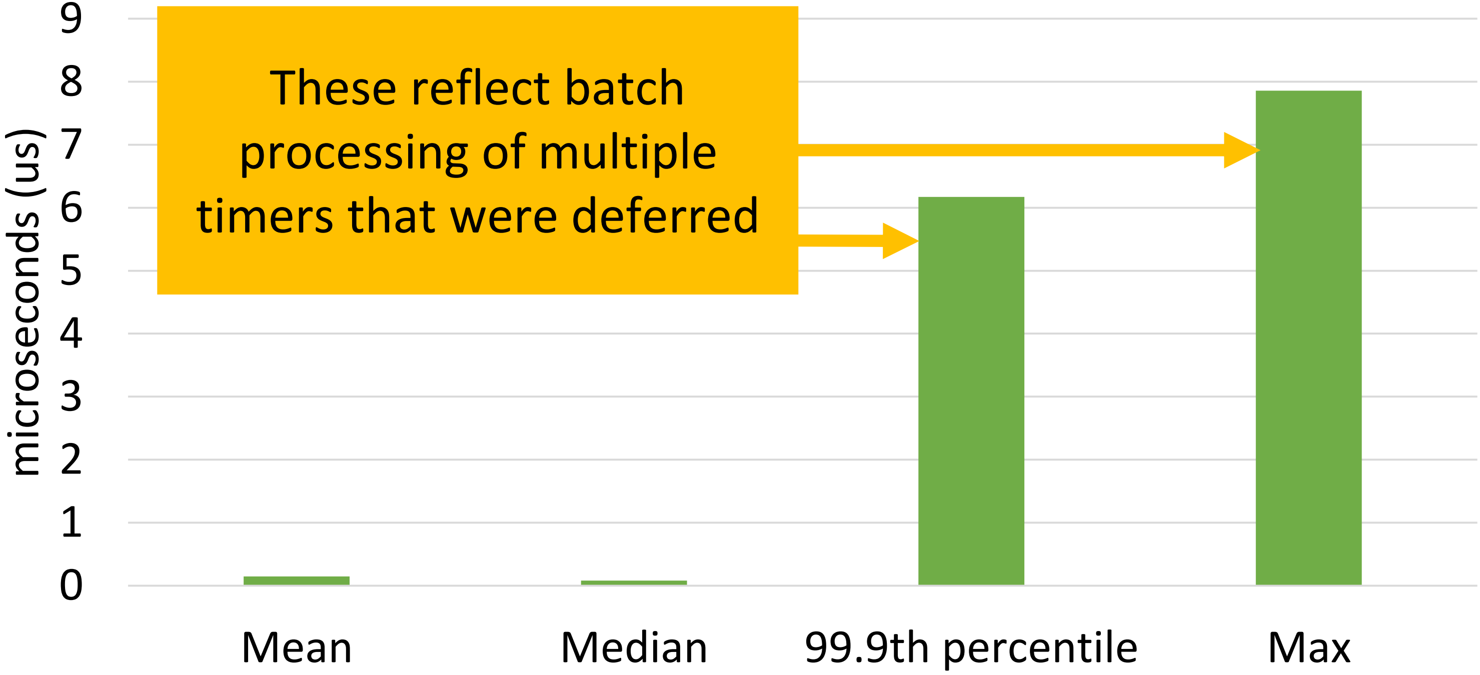


Additional Context-Switch Delay



Mean and median delay (typical case) is much less than a microsecond

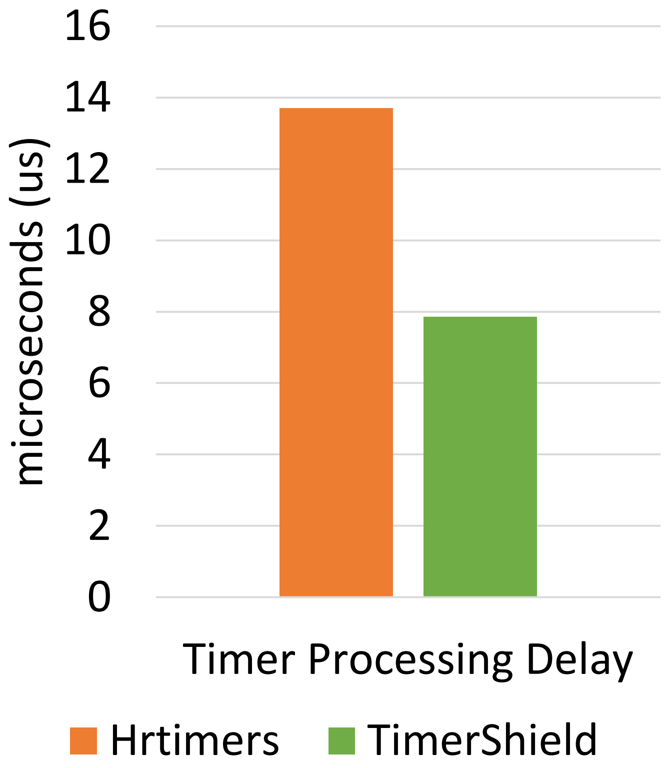
Additional Context-Switch Delay



These reflect batch processing of multiple timers that were deferred

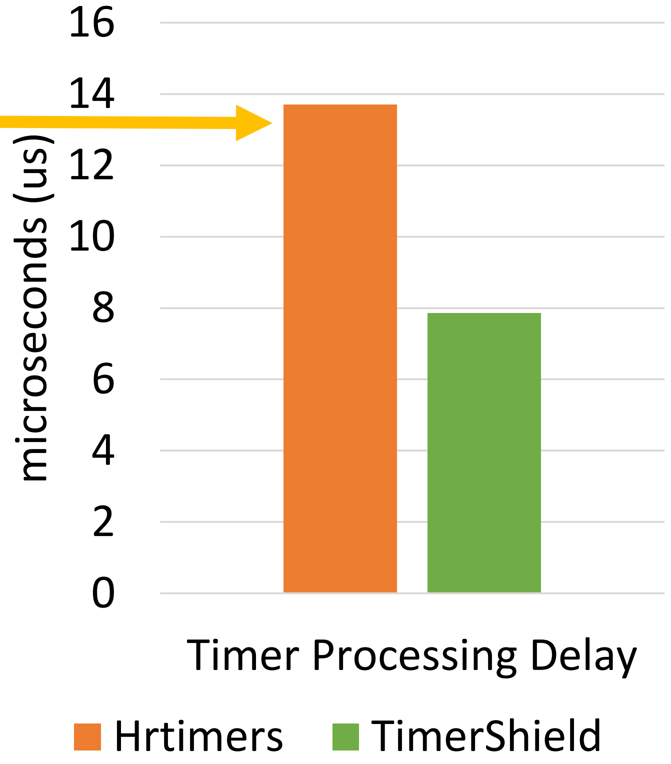
Timer Processing Delay

We measured the **worst-case increase** in HP task response time under `hrtimers` with the same experimental setup

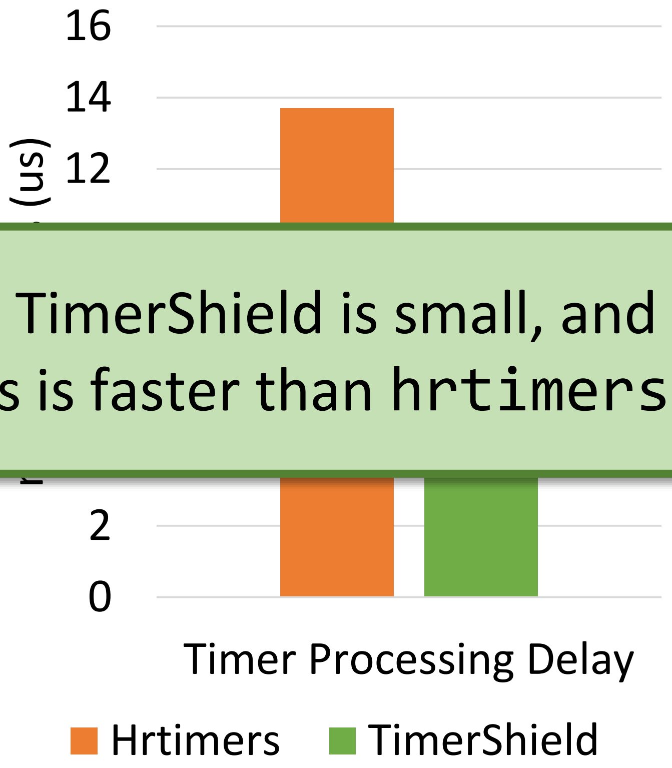


Batch Processing is Better!

`hrtimers` takes longer due to the **repetitive switches to interrupt context!**



Batch Processing is Better!



Context-switch delay due to TimerShield is small, and its batch processing of timers is faster than `hrtimers`

Evaluation

How effective is TimerShield at isolating high-priority tasks from low-priority timer interrupts?

How is the context-switch duration affected?

How costly are the new queueing data structures?

Data-Structure Overheads

The **worst case** for TimerShield’s data-structures is with a **single timer**, because each operation modifies the segment tree

Note: Results for a scenario with a timer-heavy load can be found in the paper.

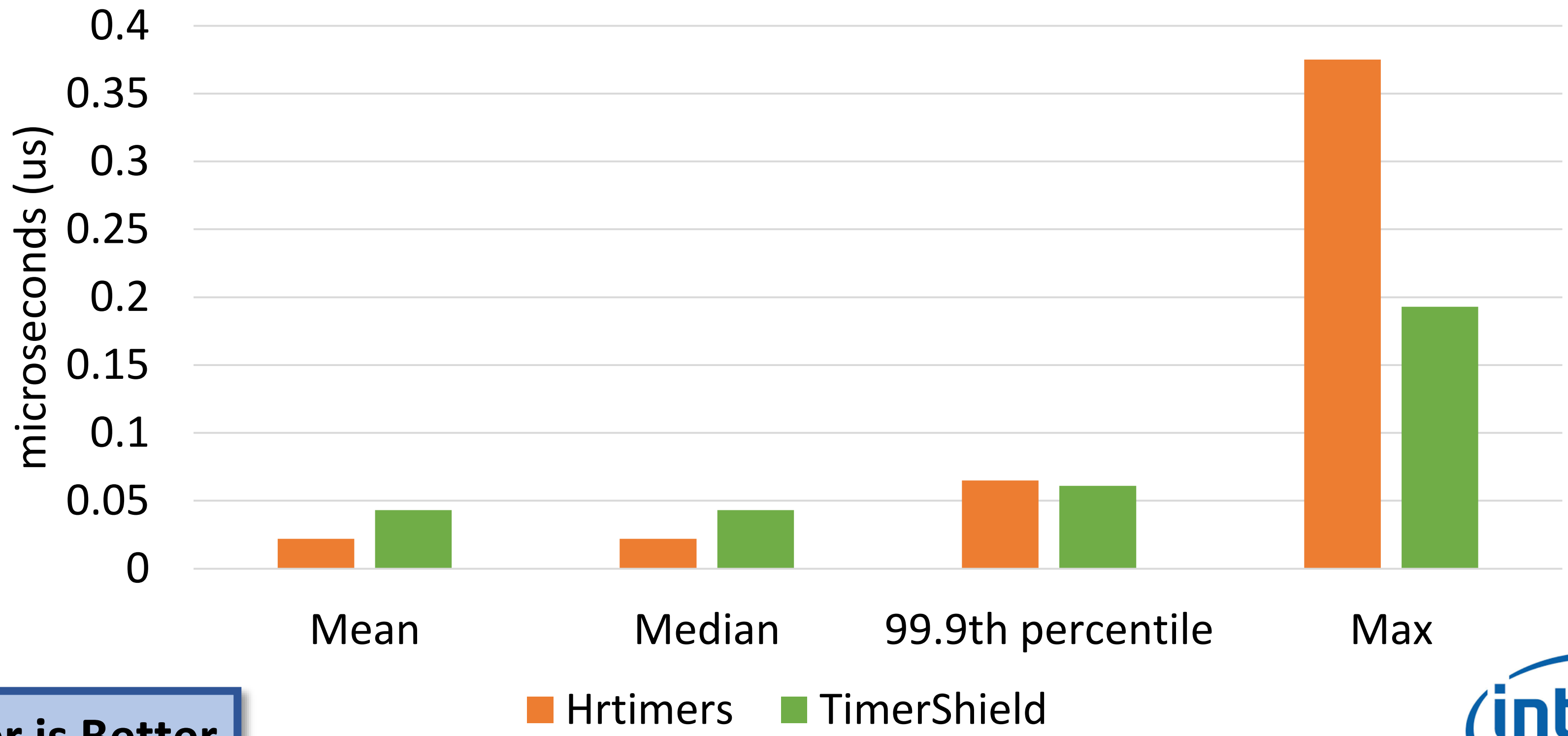
Data-Structure Overheads

The **worst case** for TimerShield’s data-structures is with a **single timer**, because each operation modifies the segment tree

We measured the timer **enqueue and dequeue cost** on both subsystems for this setup

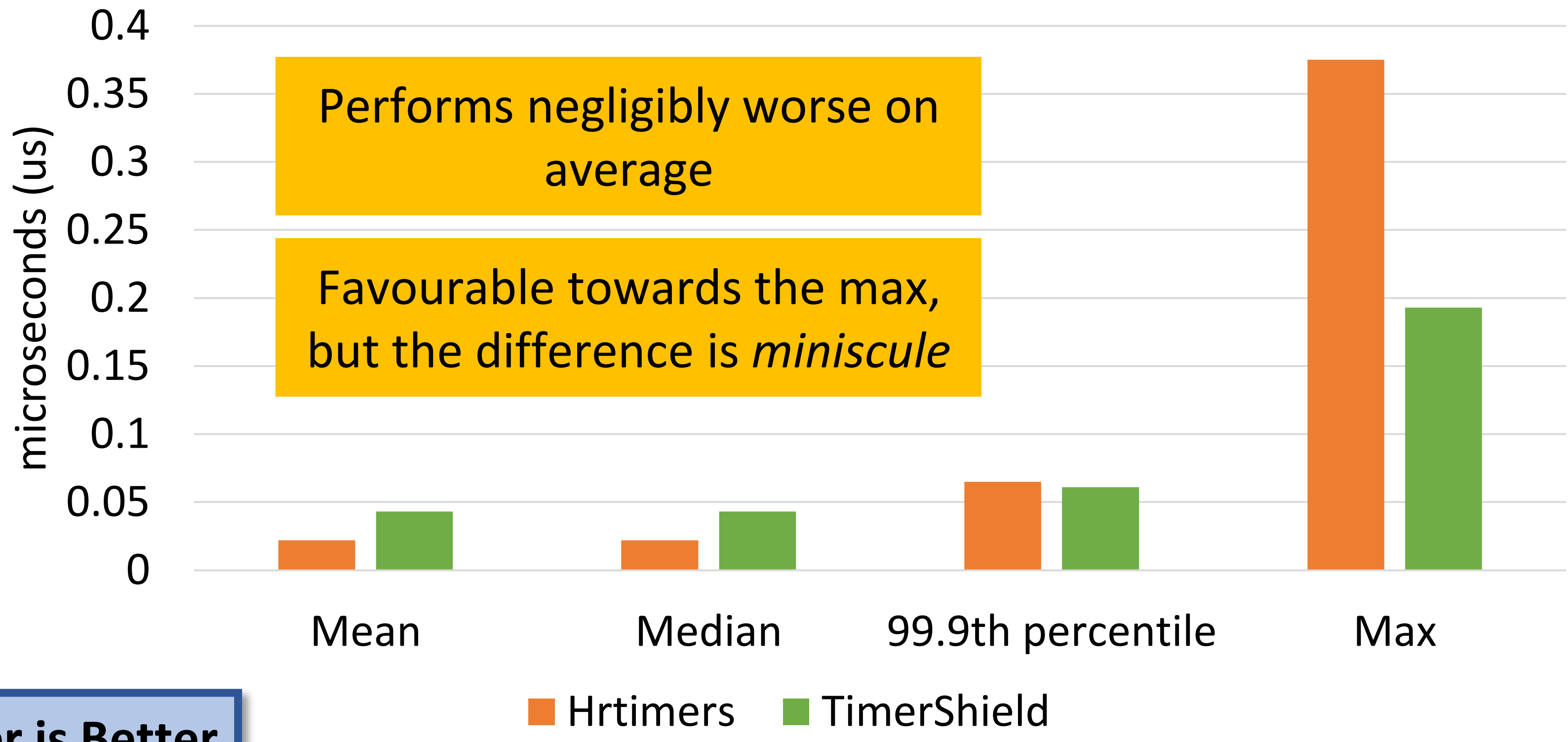
Note: Results for a scenario with a timer-heavy load can be found in the paper.

Timer Enqueue Cost



Lower is Better

Timer Enqueue Cost

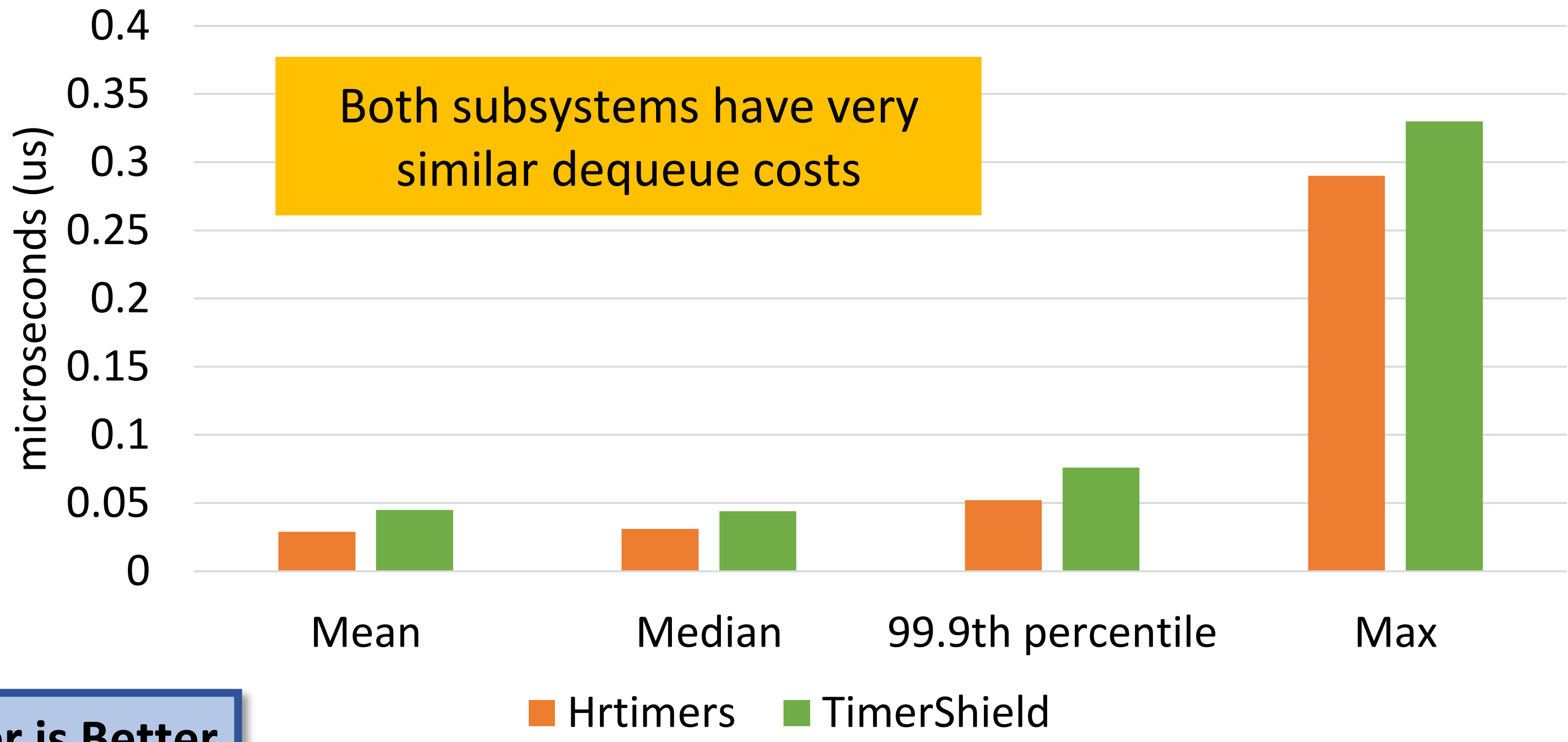


Performs negligibly worse on average

Favourable towards the max, but the difference is *miniscule*

Lower is Better

Timer Dequeue Cost



Both subsystems have very similar dequeue costs

Lower is Better

Evaluation Summary

Impossible for high-priority tasks to be interrupted by low-priority timers under TimerShield

Note: Further experiments, including results for ARM, can be found in the paper.

Evaluation Summary

Impossible for high-priority tasks to be interrupted by low-priority timers under TimerShield

Additional **context-switch delay is small**, and batch **timer processing is faster** with TimerShield

Note: Further experiments, including results for ARM, can be found in the paper.

Evaluation Summary

Impossible for high-priority tasks to be interrupted by low-priority timers under TimerShield

Additional **context-switch delay is small**, and batch **timer processing is faster** with TimerShield

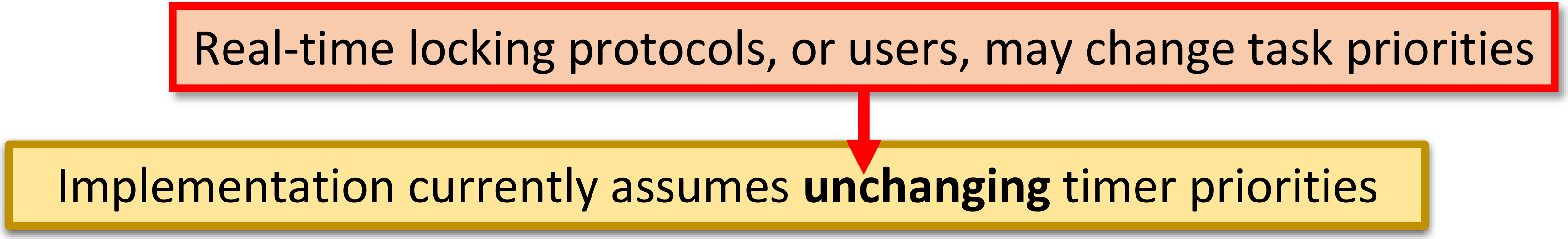
TimerShield’s data structure costs are **comparable to hrtimers**

Note: Further experiments, including results for ARM, can be found in the paper.

Dynamic Timer Priorities

Implementation currently assumes unchanging timer priorities

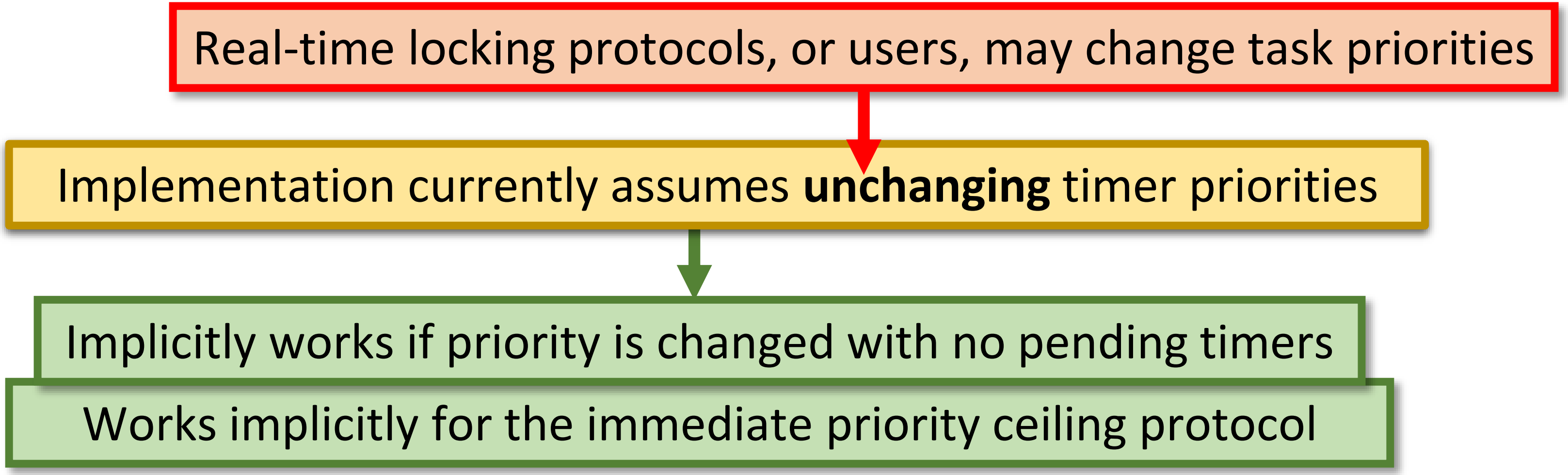
Dynamic Timer Priorities



Real-time locking protocols, or users, may change task priorities

Implementation currently assumes **unchanging** timer priorities

Dynamic Timer Priorities



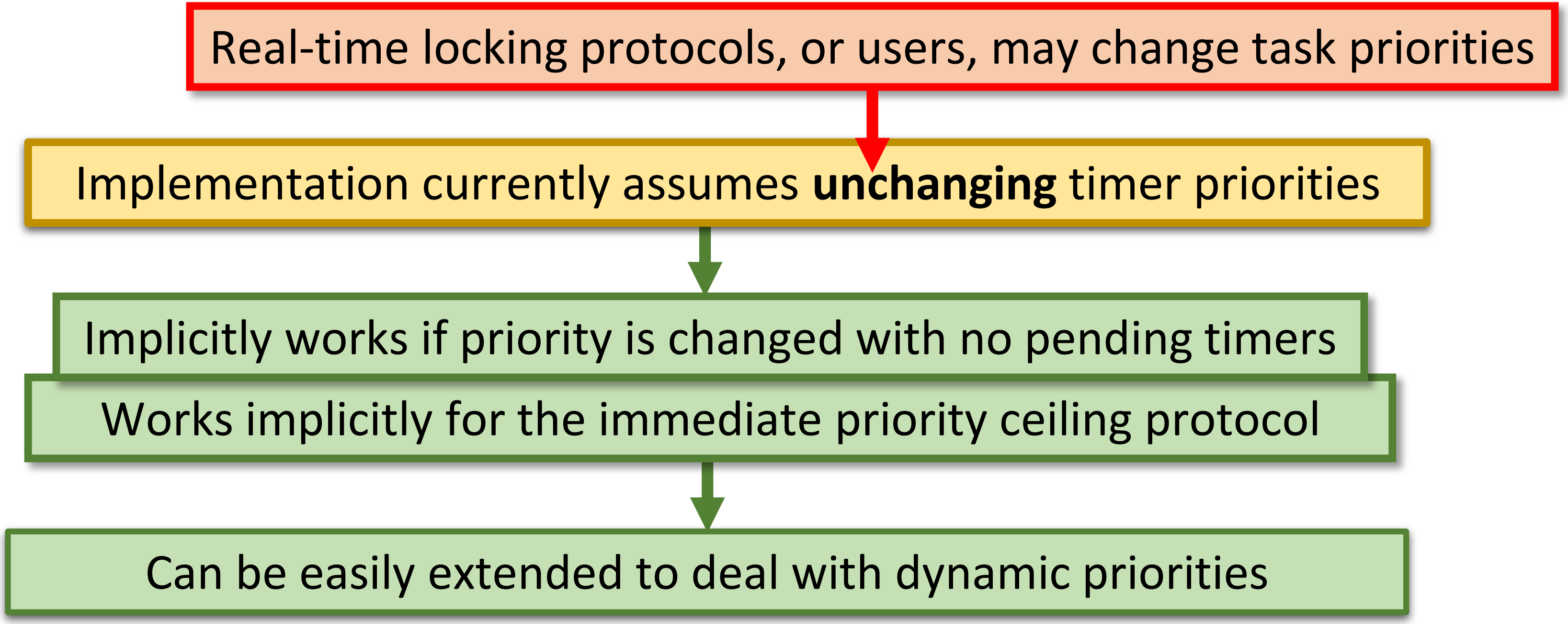
Real-time locking protocols, or users, may change task priorities

Implementation currently assumes **unchanging** timer priorities

Implicitly works if priority is changed with no pending timers

Works implicitly for the immediate priority ceiling protocol

Dynamic Timer Priorities



Real-time locking protocols, or users, may change task priorities

Implementation currently assumes **unchanging** timer priorities

Implicitly works if priority is changed with no pending timers

Works implicitly for the immediate priority ceiling protocol

Can be easily extended to deal with dynamic priorities

Future Work

Support for Earliest Deadline First (EDF) schedulers

Applying similar techniques to other, multiplexed interrupt sources such as network packet interrupts

Summary

FP scheduling on uniprocessor/partitioned multiprocessors

Summary

Low-priority timer interrupts have a **significant negative impact on high-priority task execution**

FP scheduling on uniprocessor/partitioned multiprocessors

Summary

Low-priority timer interrupts have a **significant negative impact on high-priority task execution**

Existing high-resolution timer subsystems, such as Linux `hrtimers`, **are not priority aware**

FP scheduling on uniprocessor/partitioned multiprocessors

Summary

Low-priority timer interrupts have a **significant negative impact on high-priority task execution**

Existing high-resolution timer subsystems, such as Linux `hrtimers`, **are not priority aware**

TimerShield completely avoids low-priority **timer interrupt interference** with **small overheads**

Thank you!

Source Code

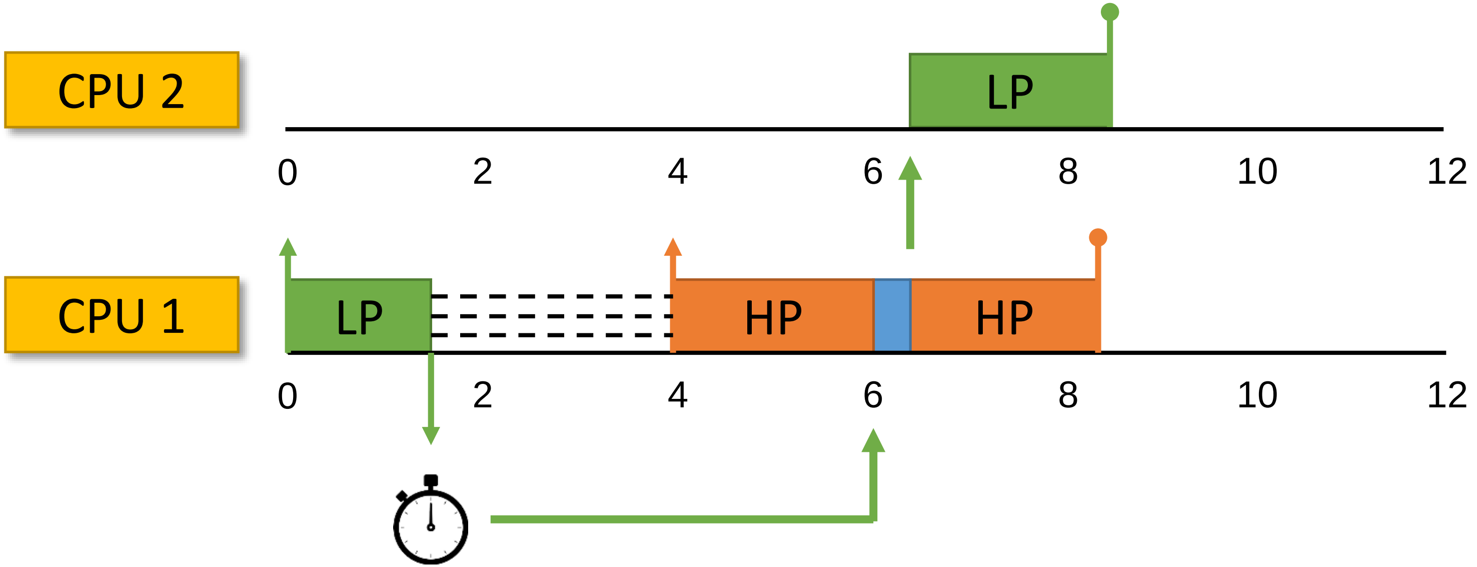
https://people.mpi-sws.org/~bbb/papers/details/rtas17p/

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

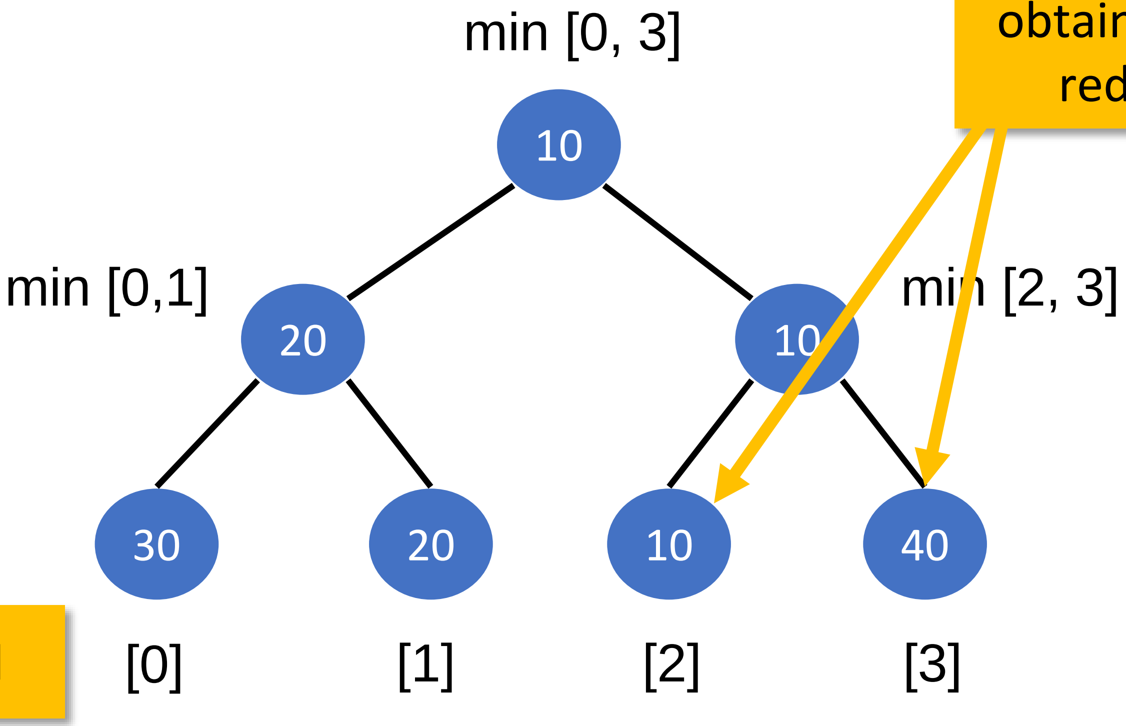
Appendix

Why Not Global Scheduling?

Not deferring the wakeup of a low-priority task might allow it to execute on a different, possibly idle CPU



Segment Tree



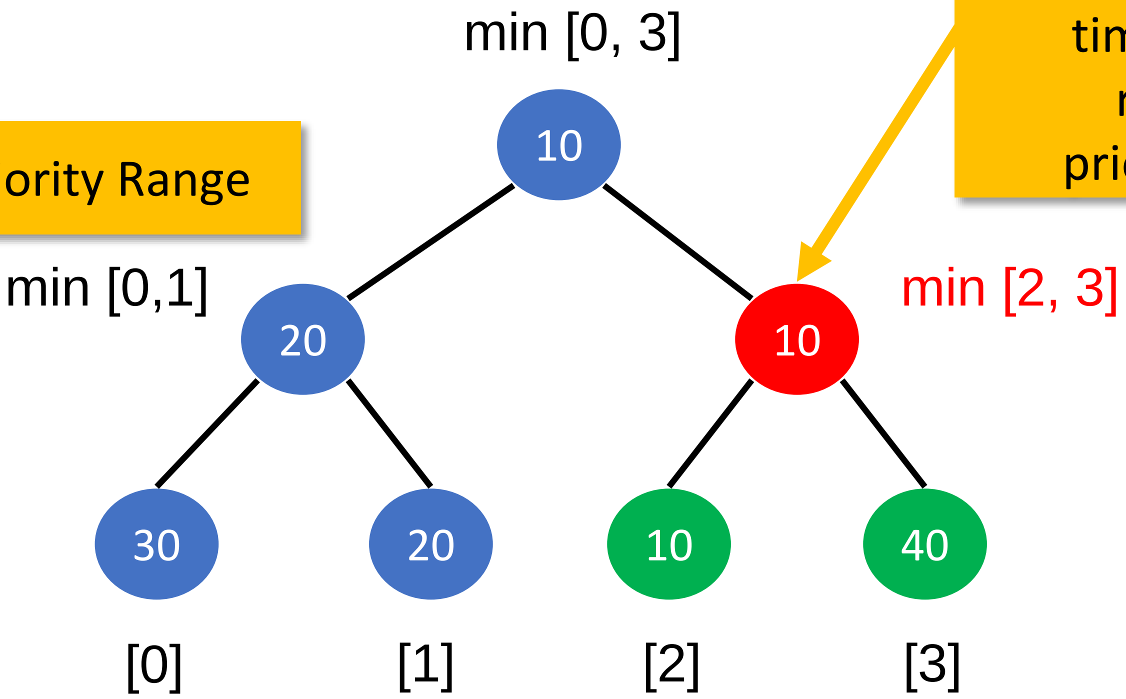
Leaf nodes correspond to the earliest timer obtained from each red-black tree

Priority Level

Segment Tree

Parent nodes store the minimum of their child nodes, and depict the earliest timer for the resulting priority range

Priority Range

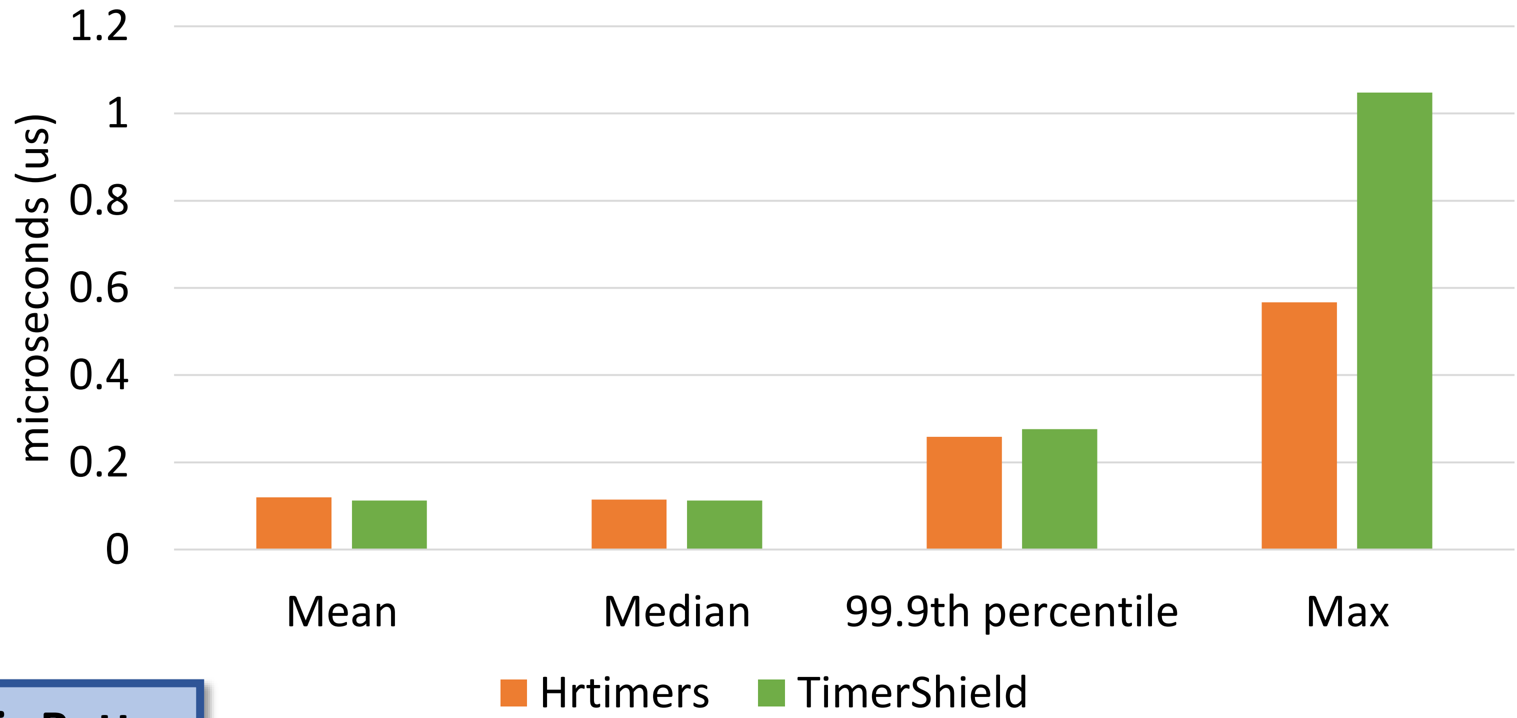


Code Size and Memory

How big is TimerShield code, and what are it’s memory requirements?

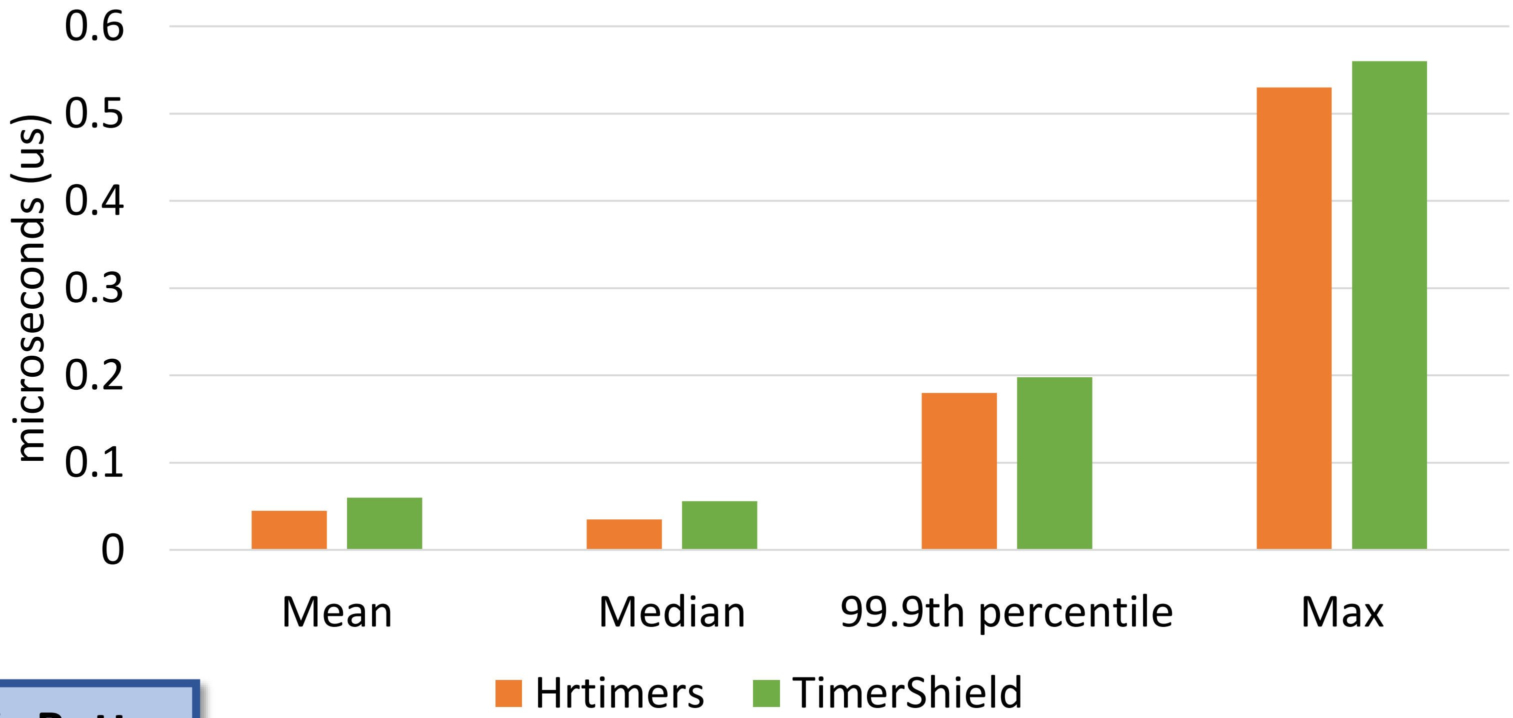
Increase in text segment	**2 KiB**
Increase in data segment	**35 KiB per core**

Timer Enqueue Cost (timer-heavy)



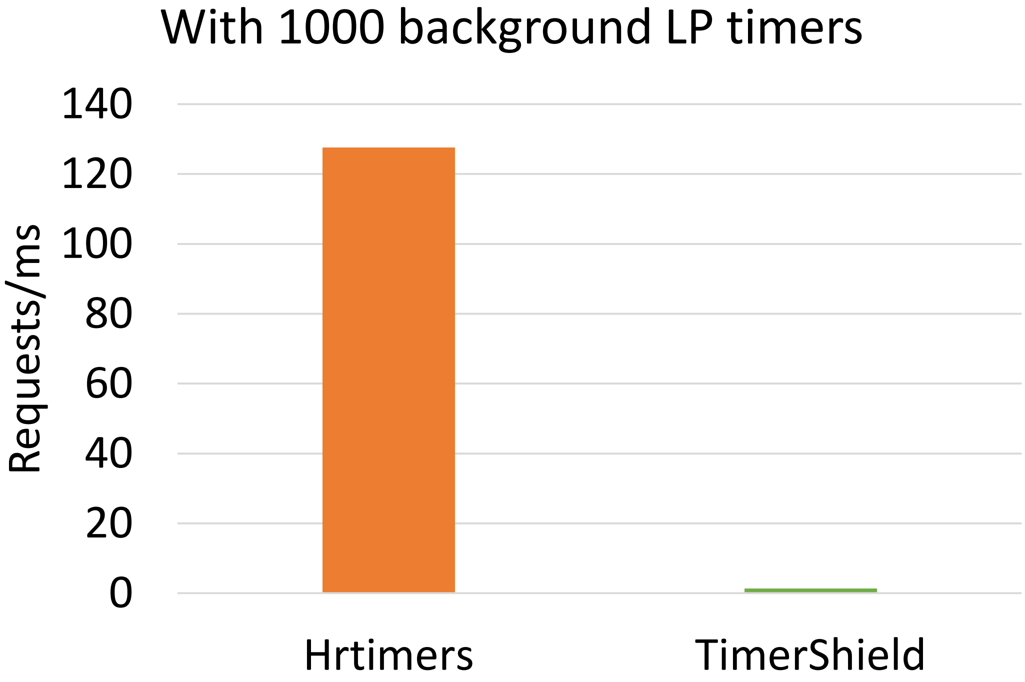
Lower is Better

Timer Dequeue Cost (timer-heavy)



Lower is Better

HP Task Throughput Reduction



Lower is Better

Idle Throughput: 7044.4 requests/ms