Microsoft Azure

Linux Kernel Real-Time Design & Challenges with Emerging Telco/5G RT Workloads

Srivatsa Bhat
Principal Software Engineer
Microsoft Linux Systems Group

Linux Kernel Meetup Bangalore
4 November 2023

What is Real-Time?

Understanding ‘Real-Time’

What is ‘Real-Time’ in Operating Systems?

- Key property: Must provide *predictable* low-latency *guarantees!*
- Key metric: Wakeup or Scheduling Latency

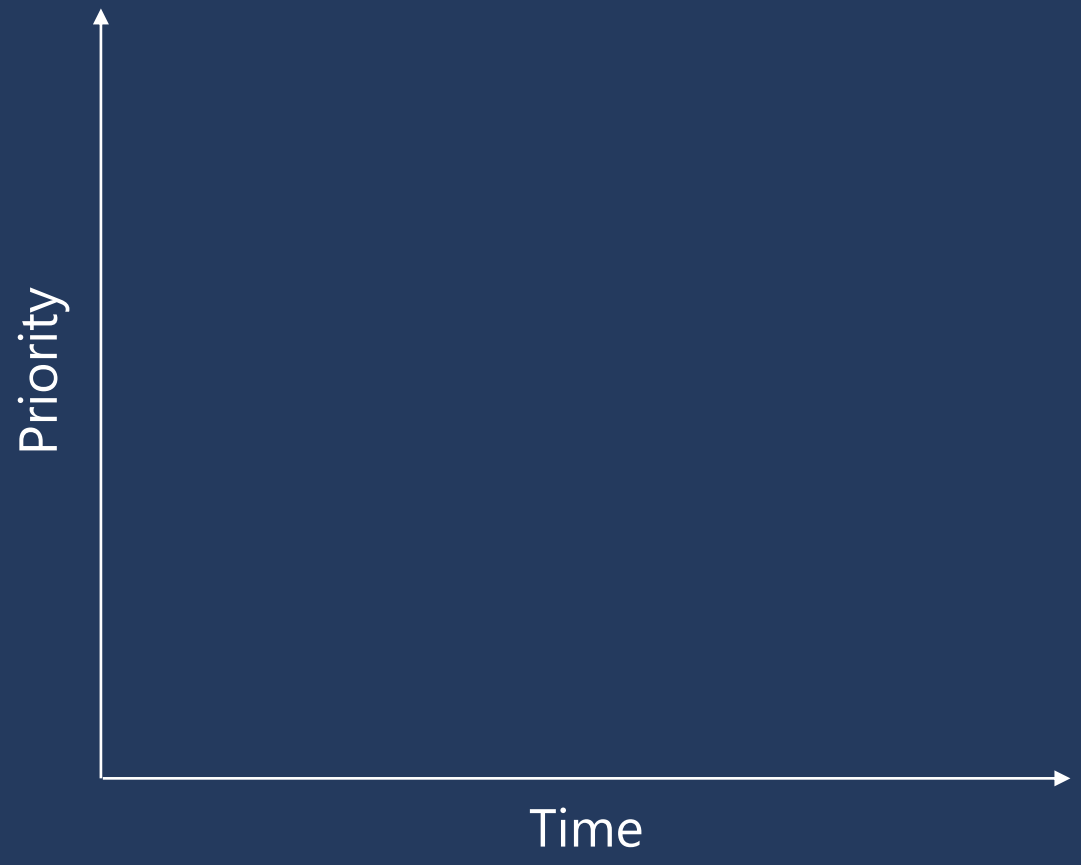
- Real-Time is NOT “real fast”
  - Trade-off between throughput vs per-task latency bounds

Who needs a Real-Time OS?

- Apps which have stringent latency requirements
- Catastrophic consequences if latency deadlines are missed
- Ex: Robotics & Industrial Automation, Telco/5G RAN

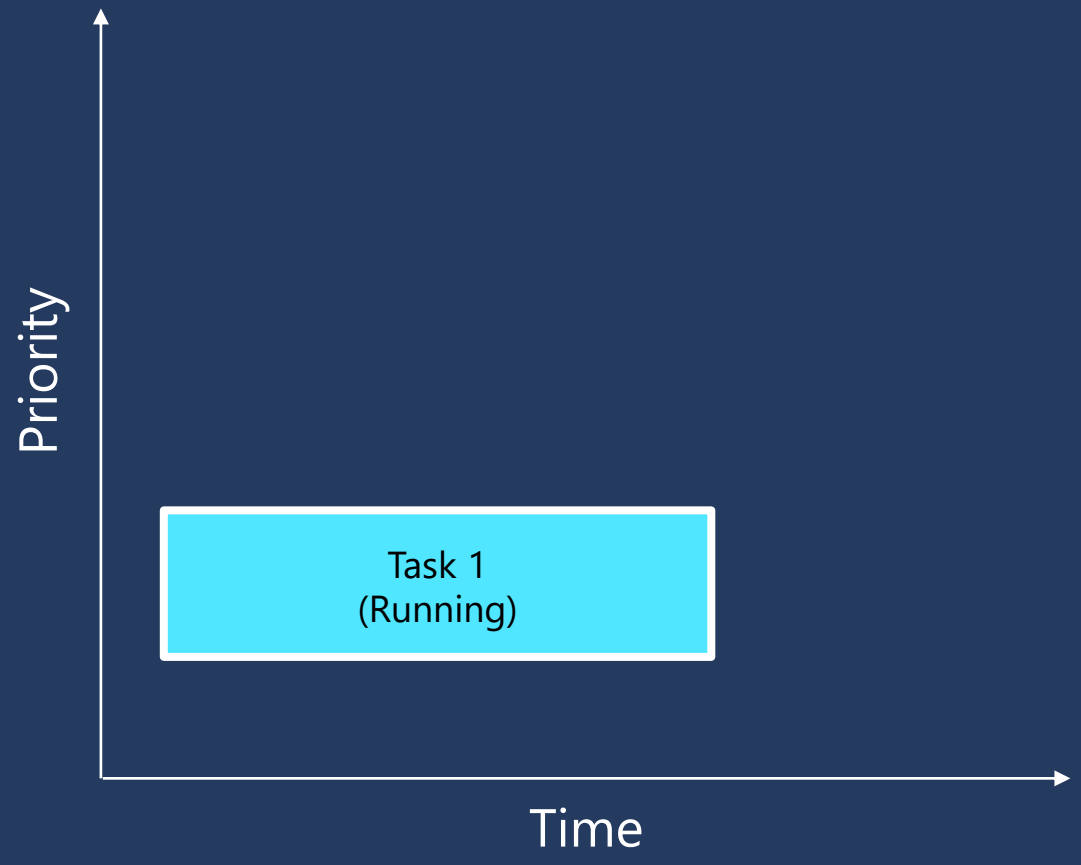
Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency in Linux Kernel: RT vs Non-RT



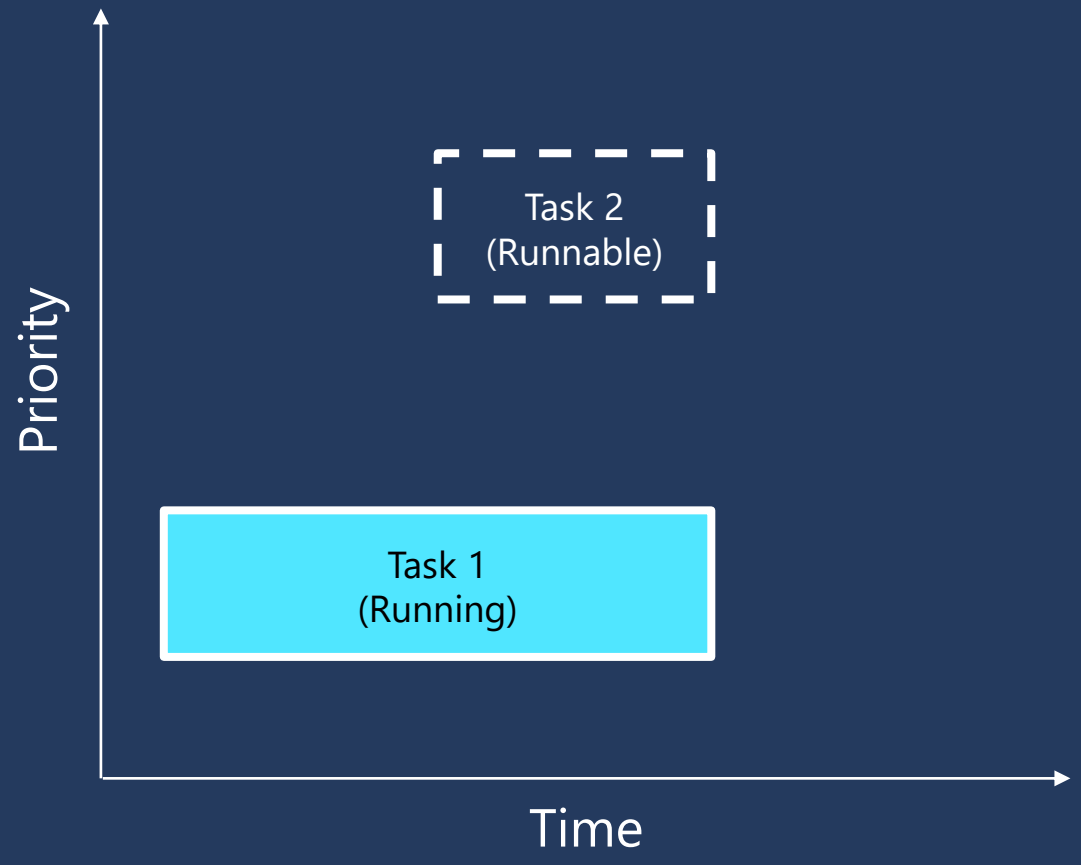
Non Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT



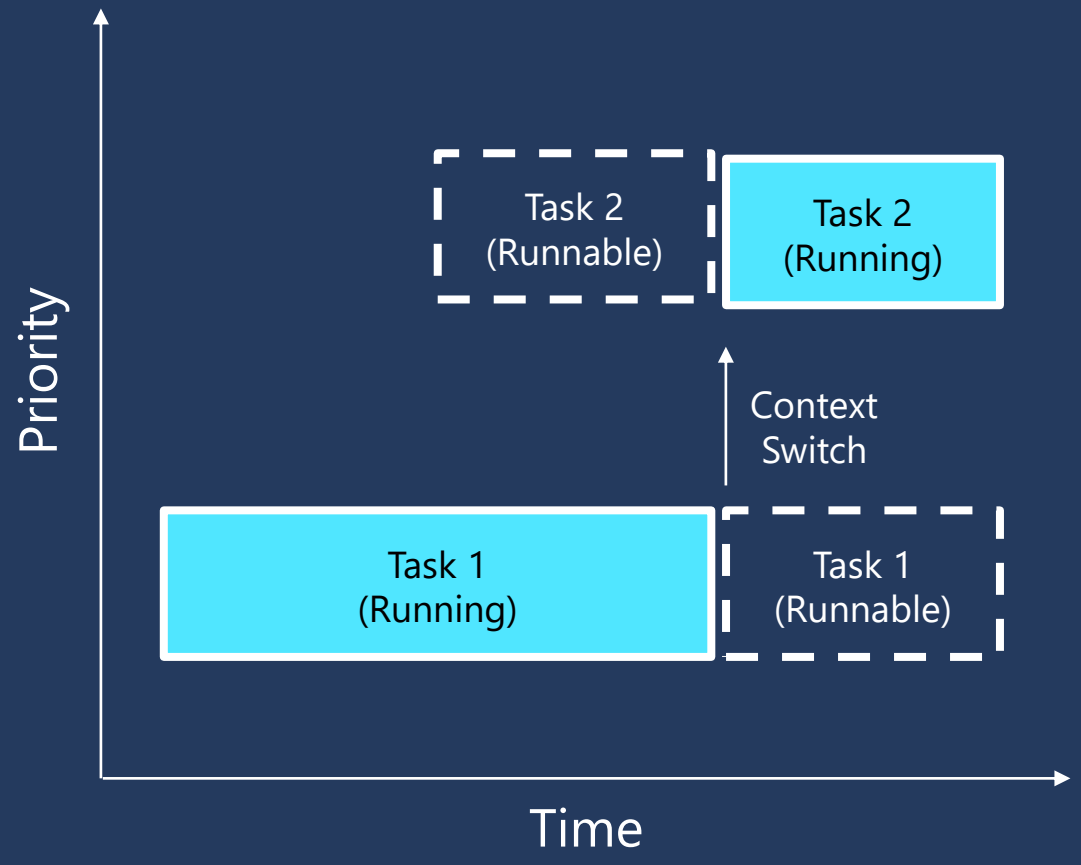
Non Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT



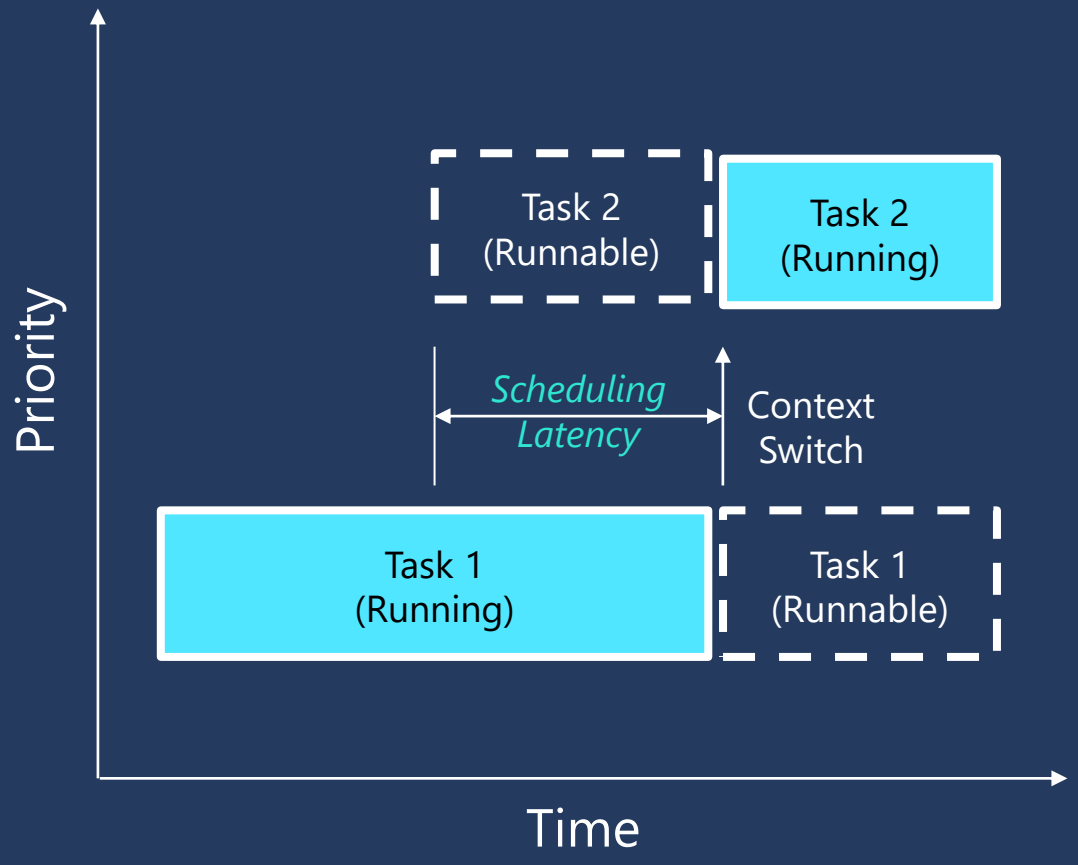
Non Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT



Non Real-Time Linux Kernel

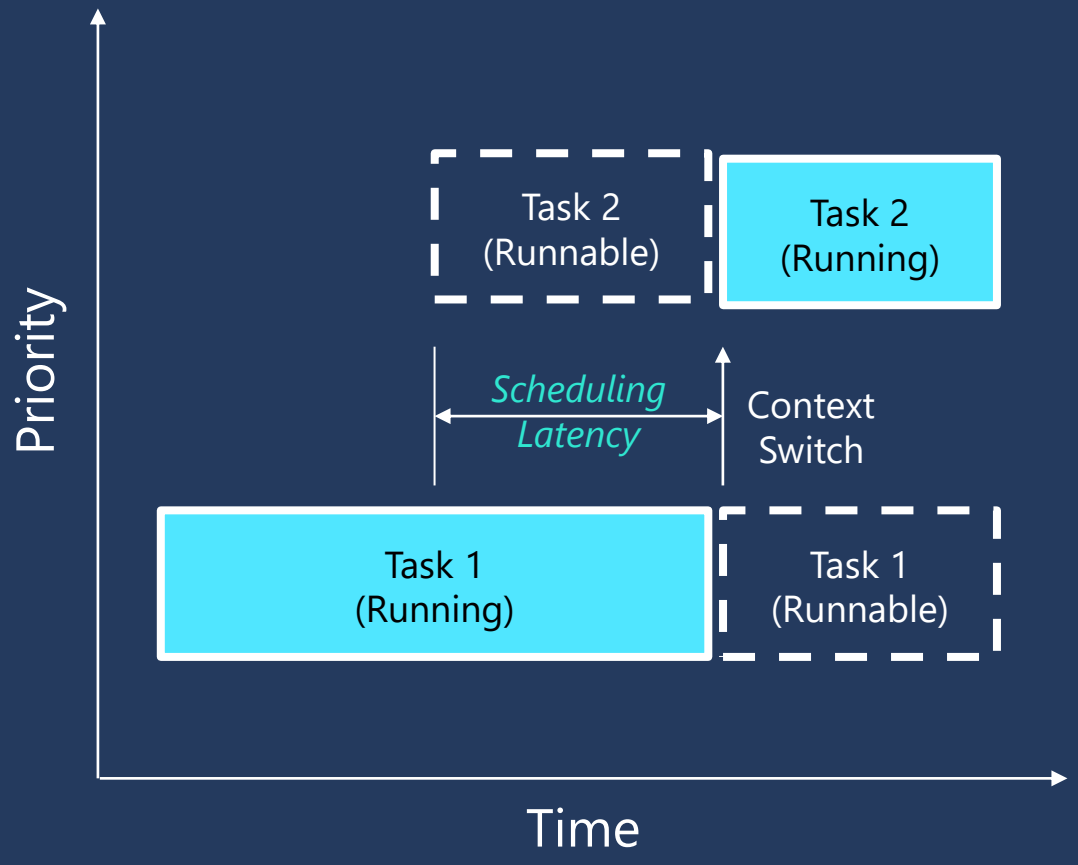
Scheduling Latency in Linux Kernel: RT vs Non-RT



Non Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT

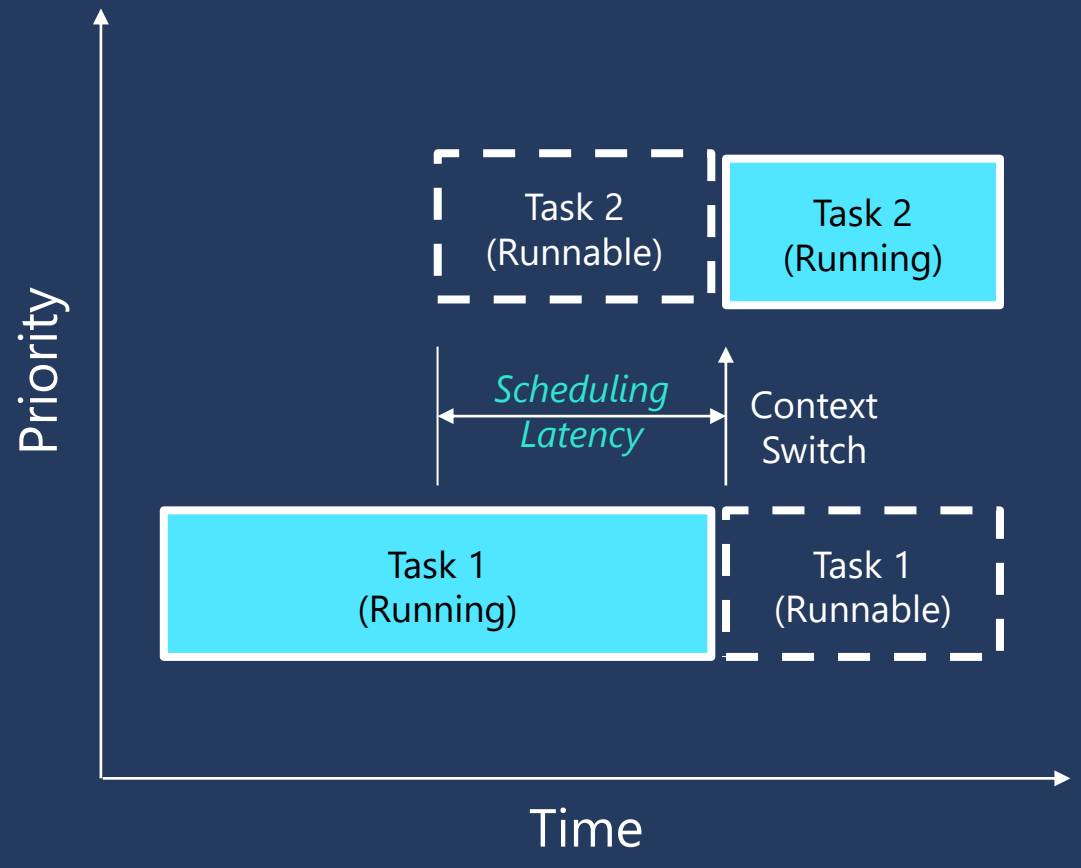
Scheduling Latency: ***Unbounded***



Non Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency: ***Unbounded***

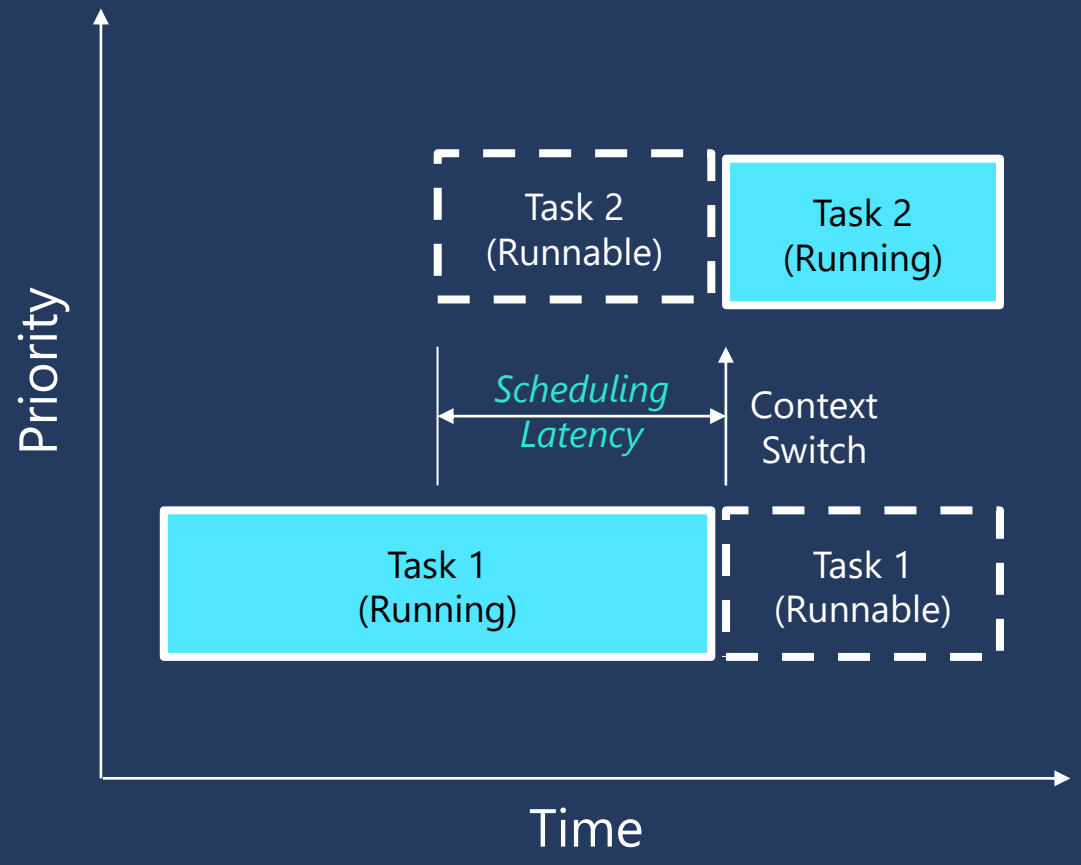


Non Real-Time Linux Kernel

Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency: ***Unbounded***

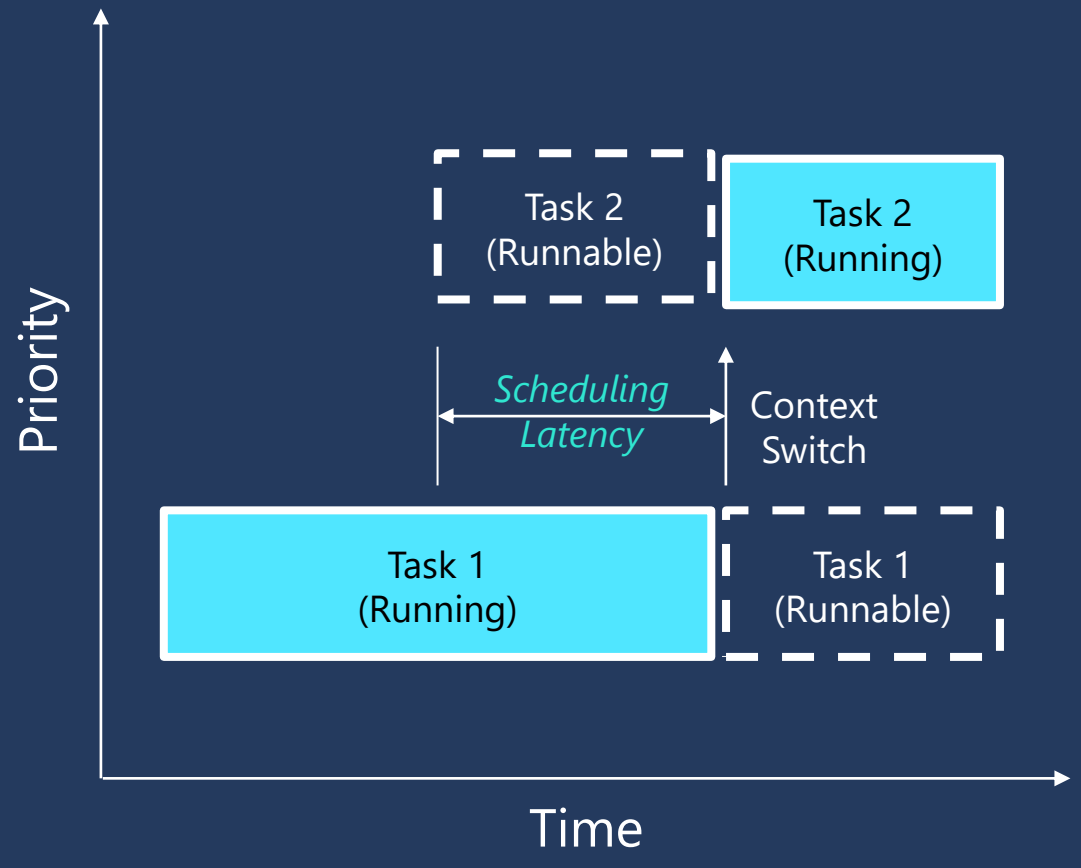


Non Real-Time Linux Kernel

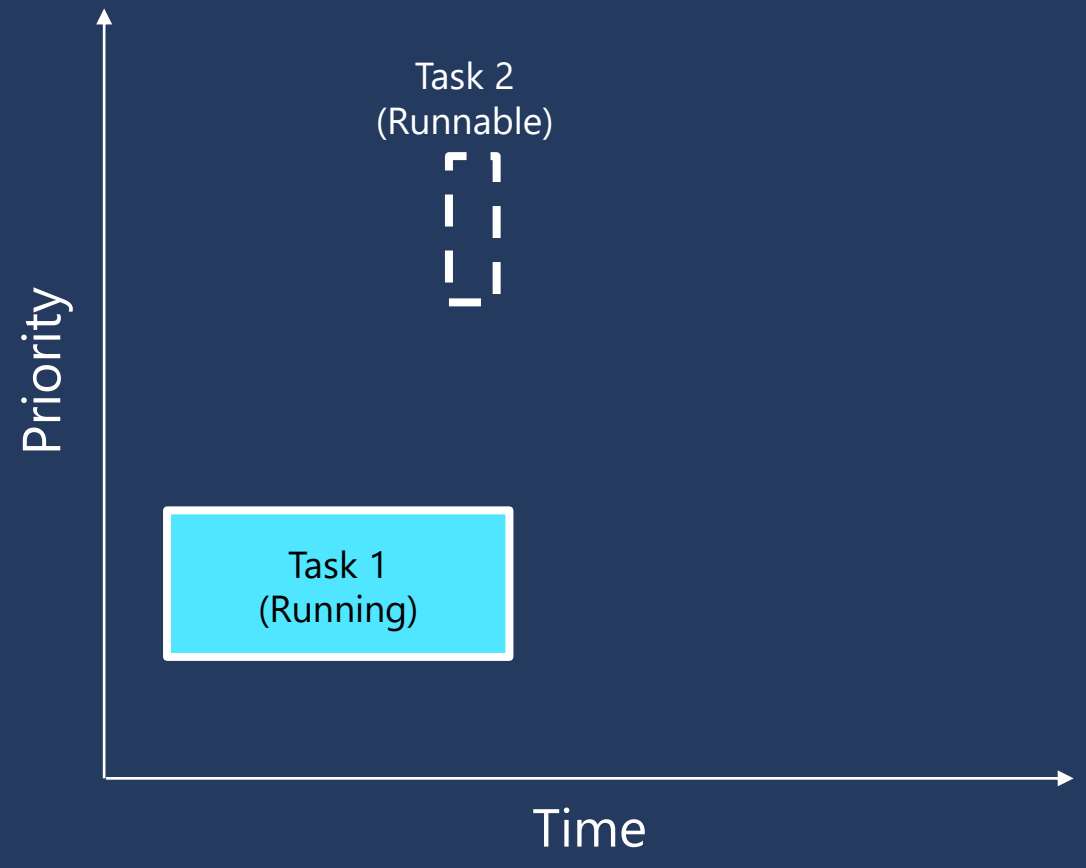
Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency: ***Unbounded***



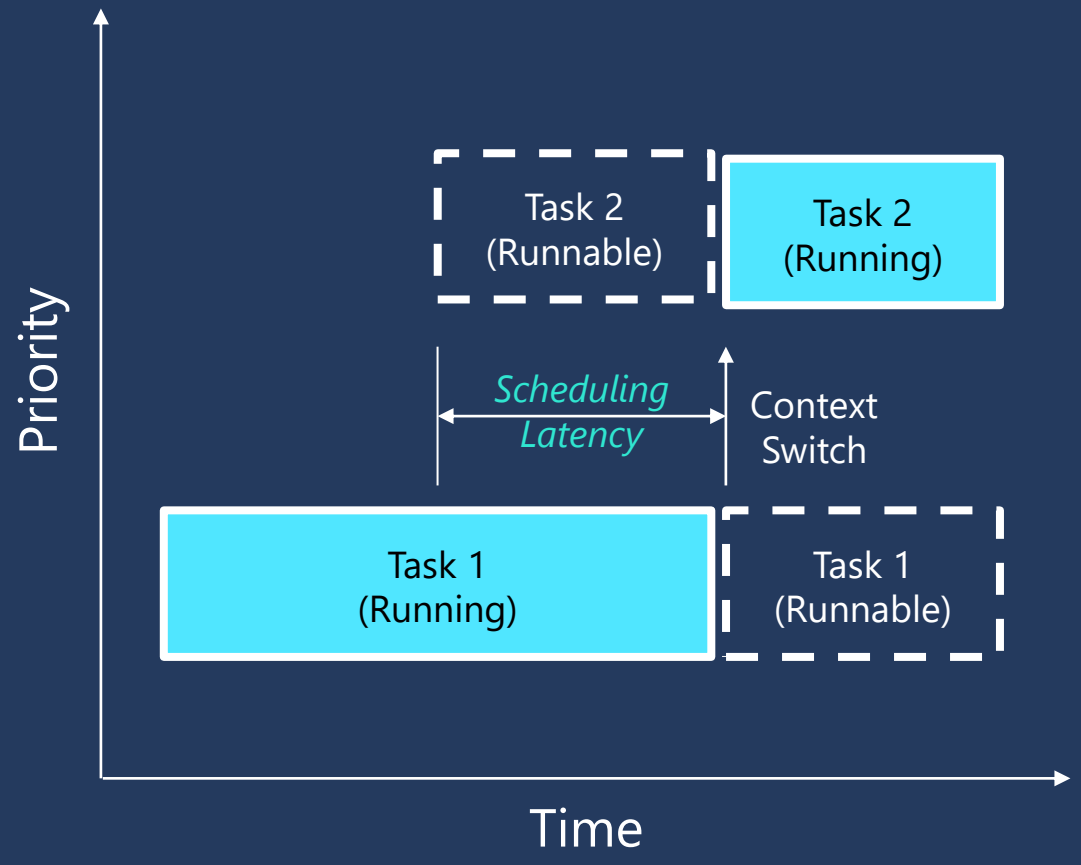
Non Real-Time Linux Kernel



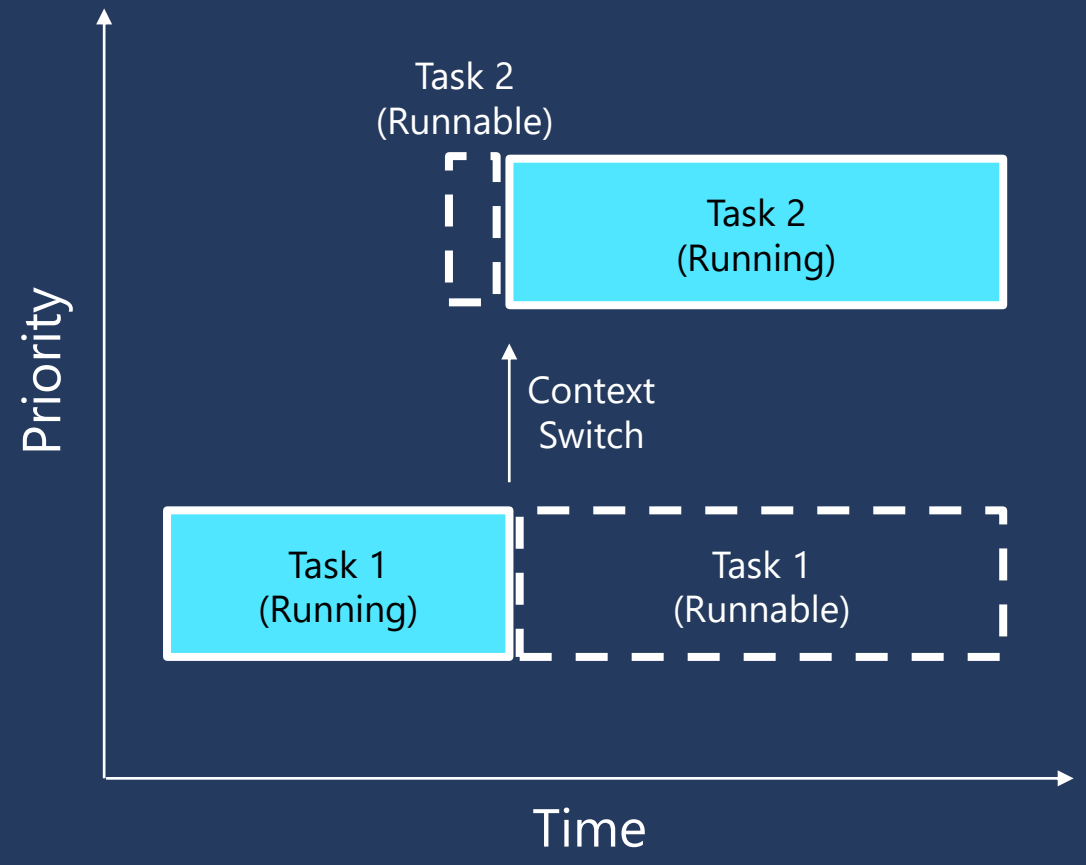
Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency: ***Unbounded***



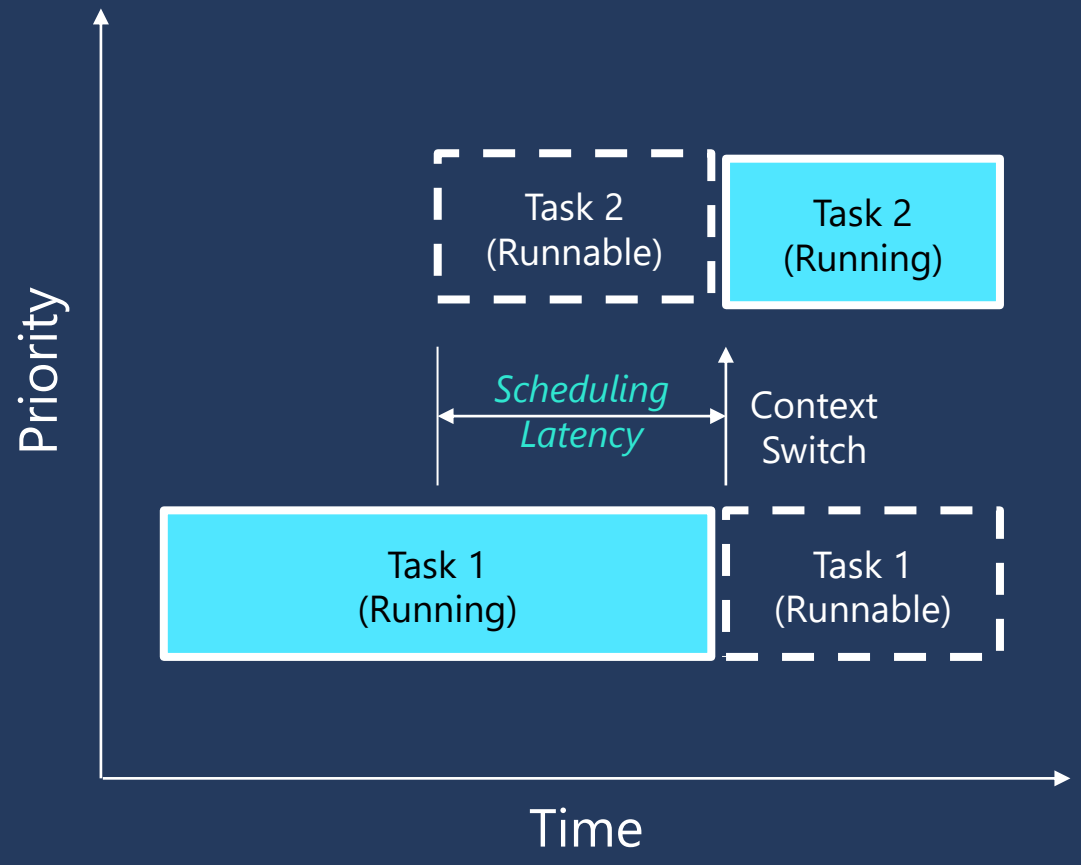
Non Real-Time Linux Kernel



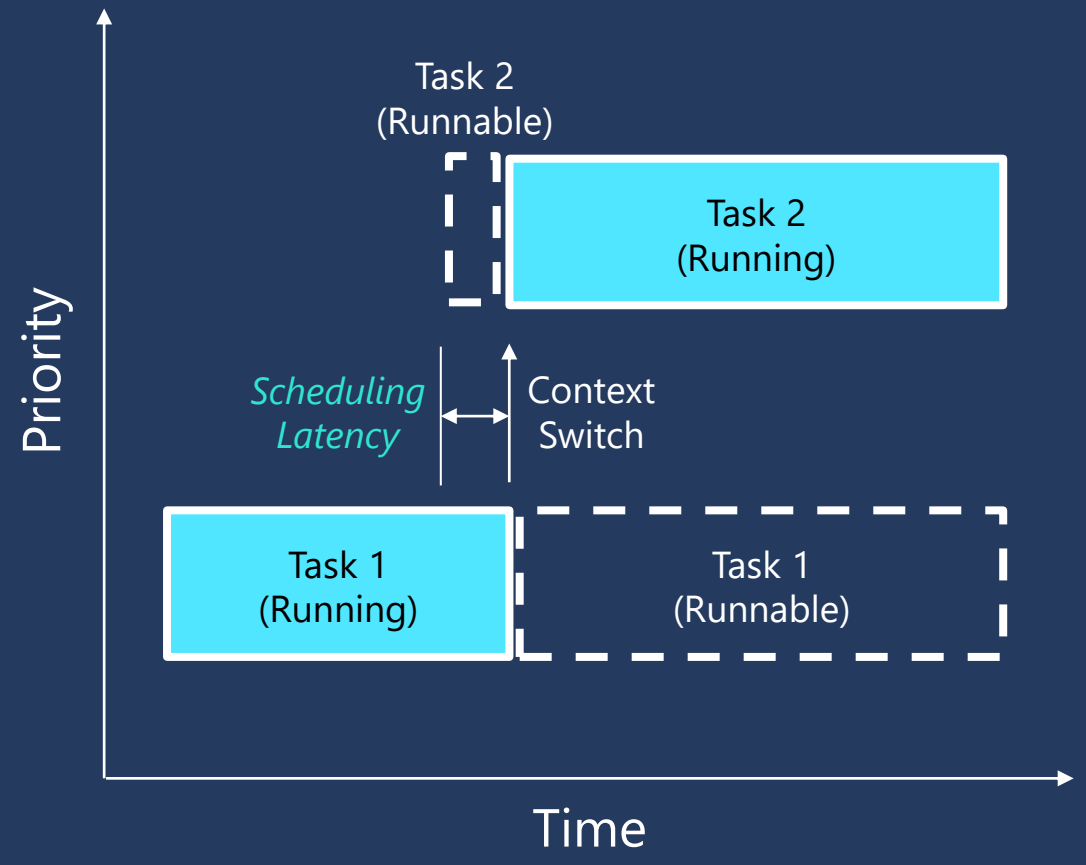
Real-Time Linux Kernel

Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency: ***Unbounded***



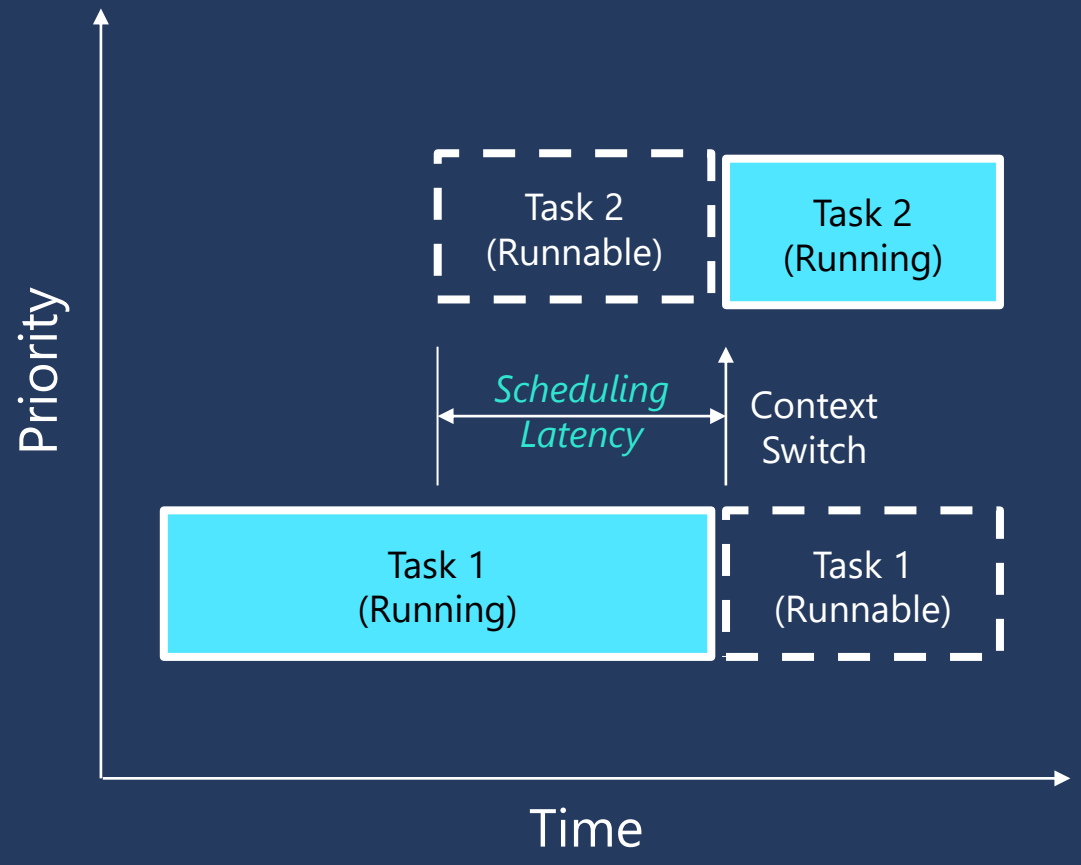
Non Real-Time Linux Kernel



Real-Time Linux Kernel

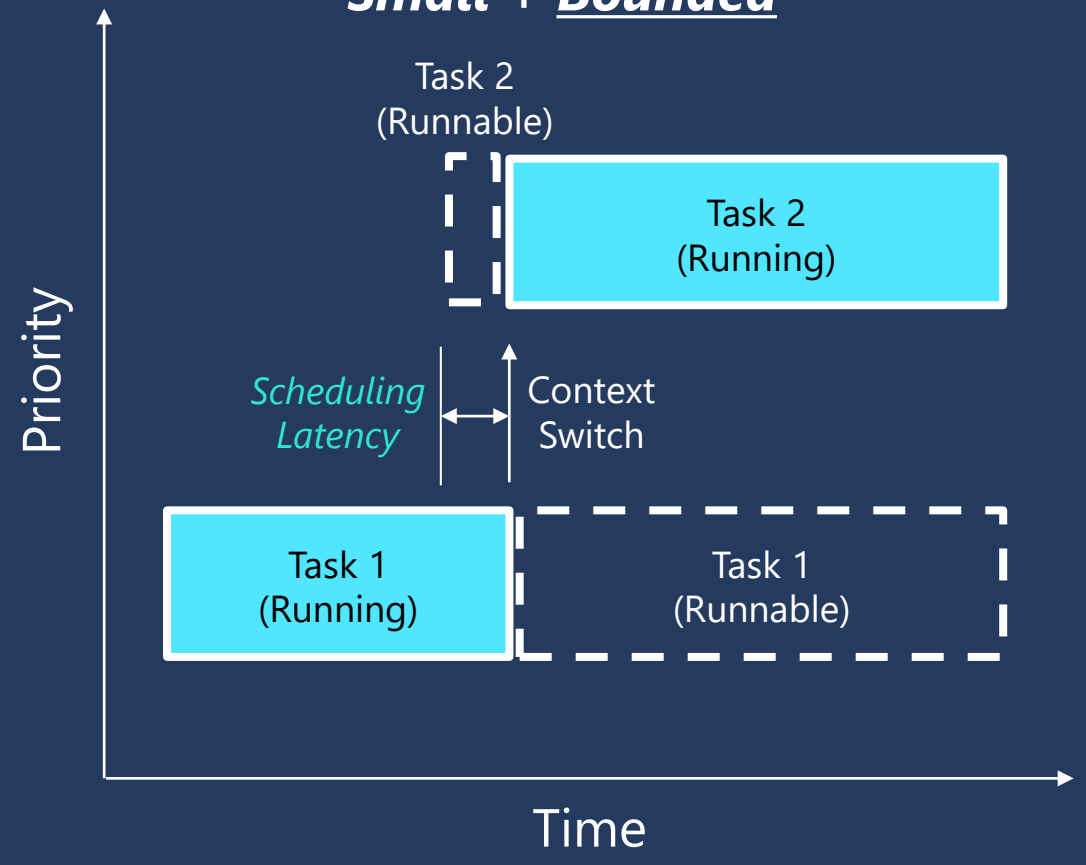
Scheduling Latency in Linux Kernel: RT vs Non-RT

Scheduling Latency: ***Unbounded***



Non Real-Time Linux Kernel

Scheduling Latency:
Small + Bounded



Real-Time Linux Kernel

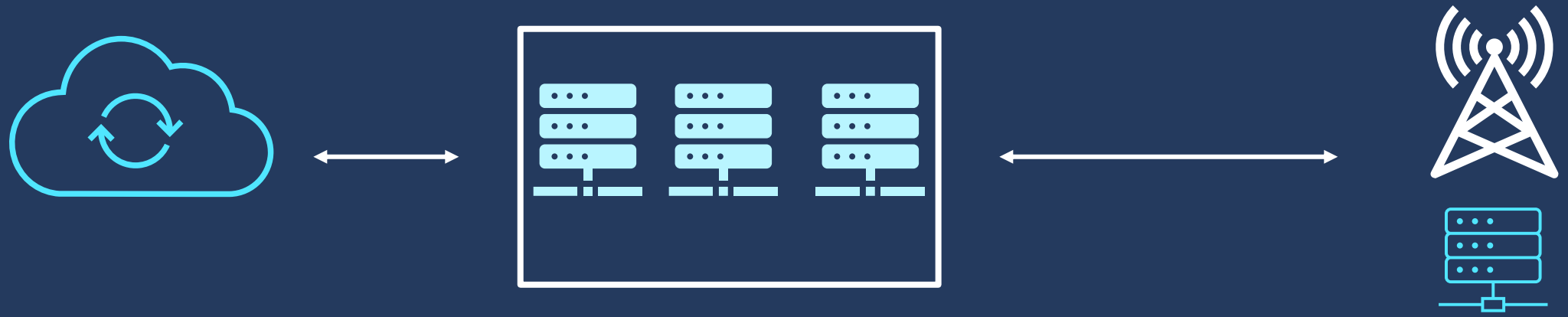
What makes Linux Kernel ‘Real-Time’?

Real-Time Preemption (PREEMPT_RT)

- Nearly all kernel code is made to be preemptible (ex: spinlocks, IRQ handlers)
- Preemptible code is available to priority scheduling
- Bounded execution for non-preemptible code in the critical paths (ex: no arbitrary loops)

Telco/5G RAN background

Radio Access Network (RAN) for Telco/5G



Core Network

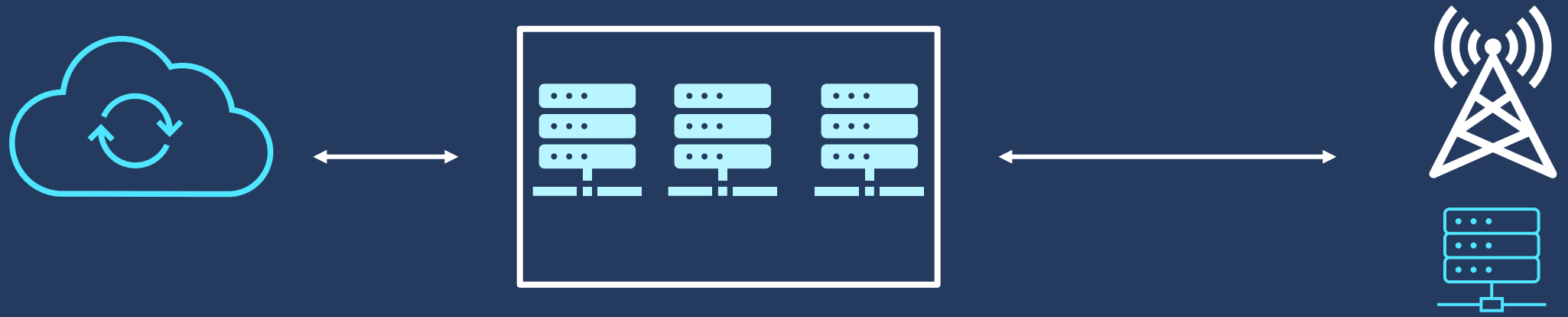
Local Data Center
(Near-Edge)

Workload: Non-RT app
CU or “Centralized Unit”

5G Radio Tower + Server
(Far-Edge)

Workload: RT app
DU or “Distributed Unit”

Radio Access Network (RAN) for Telco/5G



Core Network

Local Data Center
(Near-Edge)

Workload: Non-RT app
CU or “Centralized Unit”

5G Radio Tower + Server
(Far-Edge)

Workload: RT app
DU or “Distributed Unit”

OS latency requirements for RAN: < ***10 us scheduling latency***

Impact of exceeding latency constraints: call drops & retries

Telco/5G Real-Time app - Distributed Unit (DU)

- FlexRAN – Intel's reference implementation for RAN workload
- App's latency constraints are dictated by 5G protocol (3GPP spec)
- App characteristics:
  - Uses DPDK in polling mode to process network packets
  - Uses high real-time priority (SCHED_FIFO/90+)
  - Multi-threaded & CPU intensive
  - App aborts if latency exceeds acceptable thresholds

Linux kernel Real-Time design (PREEMPT_RT)

Understanding OS jitter

OS sources for latency spikes to RT apps

- Interrupts: Timers, device I/O, Inter-Processor Interrupts
- OS/kernel housekeeping: Kernel threads, workqueues, RCU

Understanding OS jitter

OS sources for latency spikes to RT apps

- Interrupts: Timers, device I/O, Inter-Processor Interrupts
- OS/kernel housekeeping: Kernel threads, workqueues, RCU

How does Linux PREEMPT_RT achieve low-latency guarantees?

- *Isolation:* Provides features to isolate app from OS housekeeping

Understanding OS jitter

OS sources for latency spikes to RT apps

- Interrupts: Timers, device I/O, Inter-Processor Interrupts
- OS/kernel housekeeping: Kernel threads, workqueues, RCU

How does Linux PREEMPT_RT achieve low-latency guarantees?

- *Isolation:* Provides features to isolate app from OS housekeeping
- *Real-Time Preemption*: Makes most kernel ops preemptible

Understanding OS jitter

OS sources for latency spikes to RT apps

- Interrupts: Timers, device I/O, Inter-Processor Interrupts
- OS/kernel housekeeping: Kernel threads, workqueues, RCU

How does Linux PREEMPT_RT achieve low-latency guarantees?

- *Isolation:* Provides features to isolate app from OS housekeeping
- *Real-Time Preemption*: Makes most kernel ops preemptible
- *Bounded Execution:* Non-preemptible code execution is finite + predictable

Understanding OS jitter

OS sources for latency spikes to RT apps

- Interrupts: Timers, device I/O, Inter-Processor Interrupts
- OS/kernel housekeeping: Kernel threads, workqueues, RCU

How does Linux PREEMPT_RT achieve low-latency guarantees?

- *Isolation:* Provides features to isolate app from OS housekeeping
- *Real-Time Preemption*: Makes most kernel ops preemptible
- *Bounded Execution:* Non-preemptible code execution is finite + predictable
- *Mitigation for Priority Inversion:* Solved using priority inheritance protocols

Understanding OS jitter

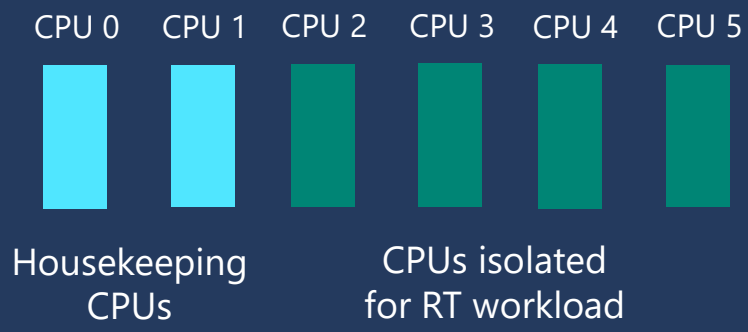
OS sources for latency spikes to RT apps

- Interrupts: Timers, device I/O, Inter-Processor Interrupts
- OS/kernel housekeeping: Kernel threads, workqueues, RCU

How does Linux PREEMPT_RT achieve low-latency guarantees?

- *Isolation:* Provides features to isolate app from OS housekeeping
- *Real-Time Preemption*: Makes most kernel ops preemptible
- *Bounded Execution:* Non-preemptible code execution is finite + predictable
- *Mitigation for Priority Inversion:* Solved using priority inheritance protocols
- *Real-Time Scheduling Algorithms:* SCHED_FIFO, SCHED_RR, SCHED_DEADLINE

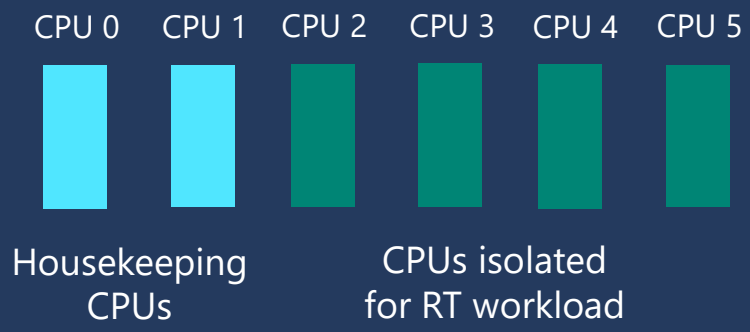
Isolation: Shielding RT app from OS jitter



Isolation: Shielding RT app from OS jitter

CPU isolation:

- Use '*isolcpus*' or cpusets to dedicate subset of CPUs to RT app
- Isolcpus takes specified CPUs out of the scheduler's purview
- Use CPU affinity to pin tasks of RT app to isolated CPUs



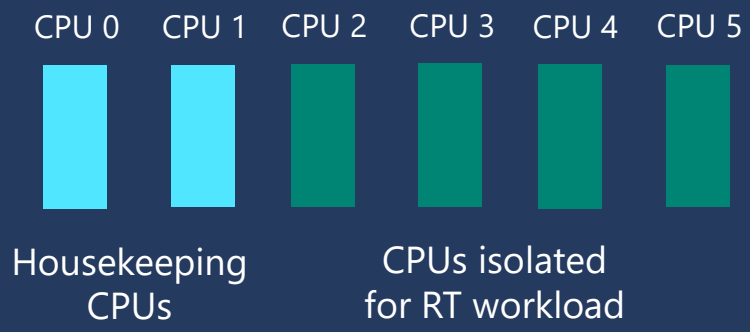
Isolation: Shielding RT app from OS jitter

CPU isolation:

- Use '*isolcpus*' or cpusets to dedicate subset of CPUs to RT app
- Isolcpus takes specified CPUs out of the scheduler's purview
- Use CPU affinity to pin tasks of RT app to isolated CPUs

Interrupt Affinity:

- Affine IRQs to housekeeping CPUs



Isolation: Shielding RT app from OS jitter

CPU isolation:

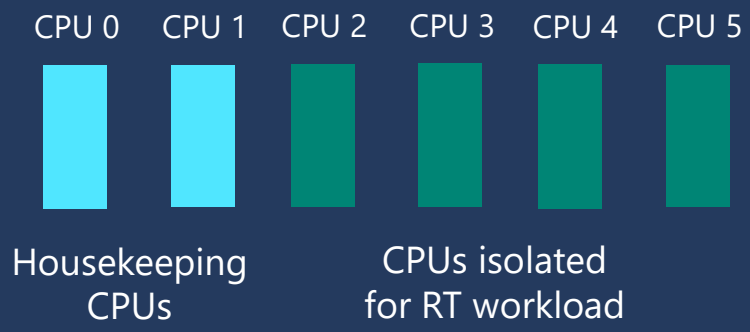
- Use '*isolcpus*' or cpusets to dedicate subset of CPUs to RT app
- Isolcpus takes specified CPUs out of the scheduler's purview
- Use CPU affinity to pin tasks of RT app to isolated CPUs

Interrupt Affinity:

- Affine IRQs to housekeeping CPUs

Full tickless execution:

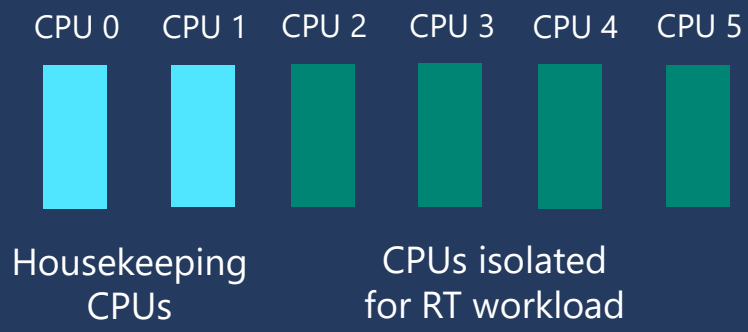
- Use '*nohz_full*' to complete disable periodic timer (scheduling) ticks on isolated CPUs



Isolation: Shielding RT app from OS jitter

CPU isolation:

- Use '*isolcpus*' or cpusets to dedicate subset of CPUs to RT app
- Isolcpus takes specified CPUs out of the scheduler's purview
- Use CPU affinity to pin tasks of RT app to isolated CPUs



Interrupt Affinity:

- Affine IRQs to housekeeping CPUs

Full tickless execution:

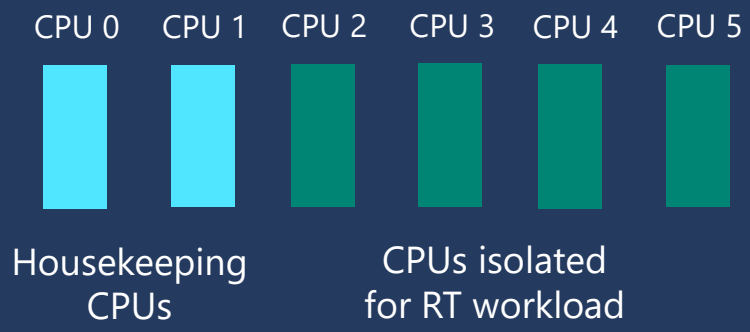
- Use '*nohz_full*' to complete disable periodic timer (scheduling) ticks on isolated CPUs

Adjust placement of deferred processing:

- Move OS housekeeping work *off* of the isolated CPUs: Ex: RCU callback processing

Isolation: Shielding RT app from OS jitter

CPU isolation:
- Use '*isolcpus*' or cpusets to dedicate subset of CPUs to RT app
- Isolcpus takes specified CPUs out of the scheduler's purview
- Use CPU affinity to pin tasks of RT app to isolated CPUs



Interrupt Affinity:
- Affine IRQs to housekeeping CPUs

Full tickless execution:
- Use '*nohz_full*' to complete disable periodic timer (scheduling) ticks on isolated CPUs

Adjust placement of deferred processing:
- Move OS housekeeping work *off* of the isolated CPUs: Ex: RCU callback processing

Automation: Use tuned package's 'real-time' profile

Real-Time Preemption

- CONFIG_PREEMPT_RT: Allows kernel to be configured as real-time
- Kernel code is now preemptible except for preempt-disabled critical sections
  - Achieved by redesigning fundamental kernel primitives to allow preemption
  - Ex: Sleepable spinlocks, threaded interrupt handlers
- Priority scheduling: RT app with high-prio can preempt low-prio kernel threads
- Non-preemptible code is audited to have bounded worst-case execution

Real-Time Scheduling Algorithms

Real-Time Scheduling Algorithms

SCHED_FIFO:

- First-In First-Out policy
- Fixed priority scheduling with prio range: 1 – 99 (highest)
- Runs highest prio task to completion (yield)

Real-Time Scheduling Algorithms

SCHED_FIFO:

- First-In First-Out policy
- Fixed priority scheduling with prio range: 1 – 99 (highest)
- Runs highest prio task to completion (yield)

SCHED_RR:

- Round Robin policy
- Same as SCHED_FIFO, except for RR with same-prio tasks

Real-Time Scheduling Algorithms

SCHED_FIFO:
- First-In First-Out policy
- Fixed priority scheduling with prio range: 1 – 99 (highest)
- Runs highest prio task to completion (yield)

SCHED_RR:
- Round Robin policy
- Same as SCHED_FIFO, except for RR with same-prio tasks

SCHED_DEADLINE:
- Not a fixed prio scheduling algorithm
- Tasks must specify params to describe their real-time demands
  - Params: Deadline D, Runtime R, Period P
  - Algo: Guarantees at least runtime ‘R’ within deadline ‘D’ in every period ‘P’

Stability challenges with CPU intensive RT apps

Stability challenges with CPU intensive RT apps

What happens if a high-prio RT app causes low-prio kernel threads to starve *permanently*?

Stability challenges with CPU intensive RT apps

What happens if a high-prio RT app causes low-prio kernel threads to starve *permanently*?

Problem

- Per-CPU kernel threads cannot be moved to housekeeping cores
- High prio RT tasks that never yield will starve kernel threads
- Impact: System hangs and instability

Stability challenges with CPU intensive RT apps

What happens if a high-prio RT app causes low-prio kernel threads to starve *permanently*?

Problem

- Per-CPU kernel threads cannot be moved to housekeeping cores
- High prio RT tasks that never yield will starve kernel threads
- Impact: System hangs and instability

Solution

- Short-term/Workaround: stalld package, which temporarily prio-boosts starving kthreads
- Long-term: Redesign Linux kernel housekeeping to allow for full isolation

Stability challenges with CPU intensive RT apps

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Stability challenges with CPU intensive RT apps

Run loop-rt on isolated CPU 2

$ taskset –c 2 ./loop-rt &

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Stability challenges with CPU intensive RT apps

Run loop-rt on isolated CPU 2

$ taskset –c 2 ./loop-rt &

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Before

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.03	2	S	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:1-mm_percpu_wq]

Stability challenges with CPU intensive RT apps

Run loop-rt on isolated CPU 2

$ taskset –c 2 ./loop-rt &

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Before

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.03	2	S	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:1-mm_percpu_wq]

No tasks runnable on CPU 2

Stability challenges with CPU intensive RT apps

Run loop-rt on isolated CPU 2

$ taskset –c 2 ./loop-rt &

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Before

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.03	2	S	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:1-mm_percpu_wq]

After

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.04	2	R	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1-mm_percpu_wq]
774	root	-91	0	2.1m	0.7m	99.9	0.0	3:36.42	2	R	./loop-rt

Stability challenges with CPU intensive RT apps

Run loop-rt on isolated CPU 2

$ taskset –c 2 ./loop-rt &

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Before

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.03	2	S	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:1-mm_percpu_wq]

After

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.04	2	R	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1-mm_percpu_wq]
774	root	-91	0	2.1m	0.7m	99.9	0.0	3:36.42	2	R	./loop-rt

loop-rt, ktimersoftd, ksoftirqd and kworker runnable on CPU2

Stability challenges with CPU intensive RT apps

Run loop-rt on isolated CPU 2

$ taskset –c 2 ./loop-rt &

loop-rt.c

```
struct sched_param s;
s.sched_priority = 90;

if (sched_setscheduler(getpid(), SCHED_FIFO, &s))
        perror("sched_setscheduler failed");

while (1)
        ;
```

Before

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.03	2	S	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:1-mm_percpu_wq]

After

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
149	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.04	2	R	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1-mm_percpu_wq]
774	root	-91	0	2.1m	0.7m	99.9	0.0	3:36.42	2	R	./loop-rt

ktimersoftd, ksoftirqd and kworker starved of CPU time!

Stability challenges with CPU intensive RT apps

Start & destroy a container from housekeeping CPU

$ docker run –it ubuntu /bin/bash

[ubuntu]$ // Attempt to exit the container

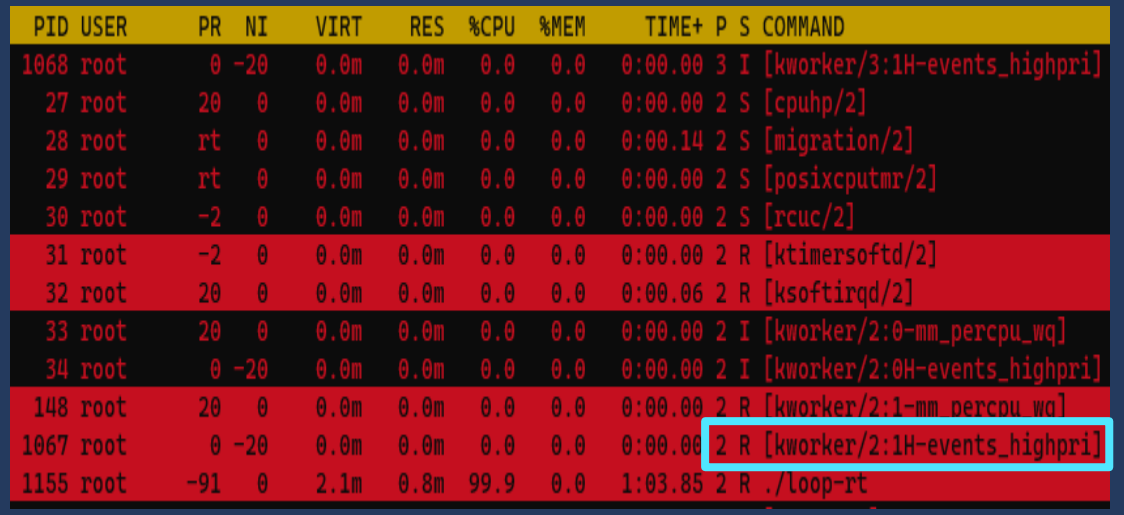
Stability challenges with CPU intensive RT apps

Start & destroy a container from housekeeping CPU

$ docker run –it ubuntu /bin/bash

[ubuntu]$ // Attempt to exit the container

events_highpri kworker also runnable on CPU 2



PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
1068	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1H-events_highpri]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.06	2	R	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H-events_highpri]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1-mm_percpu_wq]
1067	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1H-events_highpri]
1155	root	-91	0	2.1m	0.8m	99.9	0.0	1:03.85	2	R	./loop-rt

Stability challenges with CPU intensive RT apps

Start & destroy a container from housekeeping CPU

$ docker run –it ubuntu /bin/bash

[ubuntu]$ // Attempt to exit the container

events_highpri kworker also runnable on CPU 2

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
1068	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1H-events_highpri]
27	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[cpuhp/2]
28	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.14	2	S	[migration/2]
29	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[posixcputmr/2]
30	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	S	[rcuc/2]
31	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[ktimersoftd/2]
32	root	20	0	0.0m	0.0m	0.0	0.0	0:00.06	2	R	[ksoftirqd/2]
33	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0-mm_percpu_wq]
34	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	I	[kworker/2:0H-events_highpri]
148	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1-mm_percpu_wq]
1067	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	2	R	[kworker/2:1H-events_highpri]
1155	root	-91	0	2.1m	0.8m	99.9	0.0	1:03.85	2	R	./loop-rt

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
1068	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1H-events_highpri]
1069	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	4	I	[kworker/4:1H-events_highpri]
1070	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	5	I	[kworker/5:1H-events_highpri]
1071	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	7	I	[kworker/7:1H-events_highpri]
1072	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	6	I	[kworker/6:1H-events_highpri]
1164	root	20	0	0.0m	0.0m	0.7	0.0	0:00.22	1	I	[kworker/1:2-ata_sff]
1171	root	20	0	0.0m	0.0m	0.0	0.0	0:00.06	0	I	[kworker/u16:3-events_unbound]
1198	root	20	0	0.0m	0.0m	0.0	0.0	0:00.05	0	I	[kworker/0:2-events_power_effi
1202	root	20	0	0.0m	0.0m	0.0	0.0	0:00.03	1	I	[kworker/1:3-events_freezable_
25	root	20	0	0.0m	0.0m	0.0	0.0	0:00.60	1	D	[kworker/1:0+events]
465	systemd+	20	0	80.8m	4.6m	0.0	0.1	0:00.00	0	D	/lib/systemd/systemd-timesyncd
825	root	20	0	0.0m	0.0m	0.0	0.0	0:00.06	0	D	[kworker/0:0+ipv6_addrconf]
836	root	20	0	1022.5m	67.6m	0.0	0.8	0:01.29	0	D	/usr/bin/dockerd -H fd:// --co

kworker/events, kworker/ipv6_addrconf, systemd-timesyncd & dockerd all stuck in D state!!!

Stability challenges with CPU intensive RT apps

```
root@ph3-rt [ ~ ]# cat /proc/836/cmdline
/usr/bin/dockerd-Hfd://--containerd=/run/containerd/containerd.sock
root@ph3-rt [ ~ ]#
root@ph3-rt [ ~ ]# cat /proc/836/stack
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.119+0x12/0x90
[<0>] unregister_netdevice_many+0x16/0x20
[<0>] rtnl_delete_link+0x3f/0x50
[<0>] rtnl_dellink+0x121/0x2b0
[<0>] rtnetlink_rcv_msg+0x12a/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2c0/0x3f0
[<0>] sock_sendmsg+0x3e/0x50
[<0>] __sys_sendto+0x13f/0x180
[<0>] __x64_sys_sendto+0x28/0x30
[<0>] do_syscall_64+0x60/0x1b0
[<0>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
[<0>] 0xffffffffffffffff
```

Stability challenges with CPU intensive RT apps

```
root@ph3-rt [ ~ ]# cat /proc/836/cmdline
/usr/bin/dockerd-Hfd://--containerd=/run/containerd/containerd.sock
root@ph3-rt [ ~ ]#
root@ph3-rt [ ~ ]# cat /proc/836/stack
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.119+0x12/0x90
[<0>] unregister_netdevice_many+0x16/0x20
[<0>] rtnl_delete_link+0x3f/0x50
[<0>] rtnl_dellink+0x121/0x2b0
[<0>] rtnetlink_rcv_msg+0x12a/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2c0/0x3f0
[<0>] sock_sendmsg+0x3e/0x50
[<0>] __sys_sendto+0x13f/0x180
[<0>] __x64_sys_sendto+0x28/0x30
[<0>] do_syscall_64+0x60/0x1b0
[<0>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
[<0>] 0xffffffffffffffff
```

```
root@ph3-rt [ ~ ]# cat /proc/825/comm
kworker/0:0+ipv6_addrconf
root@ph3-rt [ ~ ]# cat /proc/825/stack
[<0>] rtnl_lock+0x15/0x20
[<0>] addrconf_verify_work+0xe/0x20
[<0>] process_one_work+0x1f4/0x470
[<0>] worker_thread+0x34/0x3f0
[<0>] kthread+0x160/0x180
[<0>] ret_from_fork+0x1f/0x40
[<0>] 0xffffffffffffffff
root@ph3-rt [ ~ ]#
```

Stability challenges with CPU intensive RT apps

```
root@ph3-rt [ ~ ]# cat /proc/836/cmdline
/usr/bin/dockerd-Hfd://--containerd=/run/containerd/containerd.sock
root@ph3-rt [ ~ ]#
root@ph3-rt [ ~ ]# cat /proc/836/stack
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.119+0x12/0x90
[<0>] unregister_netdevice_many+0x16/0x20
[<0>] rtnl_delete_link+0x3f/0x50
[<0>] rtnl_dellink+0x121/0x2b0
[<0>] rtnetlink_rcv_msg+0x12a/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2c0/0x3f0
[<0>] sock_sendmsg+0x3e/0x50
[<0>] __sys_sendto+0x13f/0x180
[<0>] __x64_sys_sendto+0x28/0x30
[<0>] do_syscall_64+0x60/0x1b0
[<0>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
[<0>] 0xffffffffffffffff
```

```
root@ph3-rt [ ~ ]# cat /proc/825/comm
kworker/0:0+ipv6_addrconf
root@ph3-rt [ ~ ]# cat /proc/825/stack
[<0>] rtnl_lock+0x15/0x20
[<0>] addrconf_verify_work+0xe/0x20
[<0>] process_one_work+0x1f4/0x470
[<0>] worker_thread+0x34/0x3f0
[<0>] kthread+0x160/0x180
[<0>] ret_from_fork+0x1f/0x40
[<0>] 0xffffffffffffffff
root@ph3-rt [ ~ ]#
```

```
void unregister_netdevice_many()
{
    ...
    rtnl_lock();
    ...
    flush_all_backlogs();
    ....
    rtnl_unlock();
}
```

Stability challenges with CPU intensive RT apps

```
root@ph3-rt [ ~ ]# cat /proc/836/cmdline
/usr/bin/dockerd-Hfd://--containerd=/run/containerd/containerd.sock
root@ph3-rt [ ~ ]#
root@ph3-rt [ ~ ]# cat /proc/836/stack
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.119+0x12/0x90
[<0>] unregister_netdevice_many+0x16/0x20
[<0>] rtnl_delete_link+0x3f/0x50
[<0>] rtnl_dellink+0x121/0x2b0
[<0>] rtnetlink_rcv_msg+0x12a/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2c0/0x3f0
[<0>] sock_sendmsg+0x3e/0x50
[<0>] __sys_sendto+0x13f/0x180
[<0>] __x64_sys_sendto+0x28/0x30
[<0>] do_syscall_64+0x60/0x1b0
[<0>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
[<0>] 0xffffffffffffffff
```

```
root@ph3-rt [ ~ ]# cat /proc/825/comm
kworker/0:0+ipv6_addrconf
root@ph3-rt [ ~ ]# cat /proc/825/stack
[<0>] rtnl_lock+0x15/0x20
[<0>] addrconf_verify_work+0xe/0x20
[<0>] process_one_work+0x1f4/0x470
[<0>] worker_thread+0x34/0x3f0
[<0>] kthread+0x160/0x180
[<0>] ret_from_fork+0x1f/0x40
[<0>] 0xffffffffffffffff
root@ph3-rt [ ~ ]#
```

```
void unregister_netdevice_many()
{
    ...
    rtnl_lock();
    ...
    flush_all_backlogs();
    ....
    rtnl_unlock();
}
```

```
void flush_all_backlogs()
{
    ...
    for_each_online_cpu(cpu) {
        queue_work_on(...);
    }
    ...
    for_each_online_cpu(cpu) {

        flush_work(...);
    }
}
```

Stability challenges with CPU intensive RT apps

```
root@ph3-rt [ ~ ]# cat /proc/836/cmdline
/usr/bin/dockerd-Hfd://--containerd=/run/containerd/containerd.sock
root@ph3-rt [ ~ ]#
root@ph3-rt [ ~ ]# cat /proc/836/stack
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.119+0x12/0x90
[<0>] unregister_netdevice_many+0x16/0x20
[<0>] rtnl_delete_link+0x3f/0x50
[<0>] rtnl_dellink+0x121/0x2b0
[<0>] rtnetlink_rcv_msg+0x12a/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2c0/0x3f0
[<0>] sock_sendmsg+0x3e/0x50
[<0>] __sys_sendto+0x13f/0x180
[<0>] __x64_sys_sendto+0x28/0x30
[<0>] do_syscall_64+0x60/0x1b0
[<0>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
[<0>] 0xffffffffffffffff
```

```
root@ph3-rt [ ~ ]# cat /proc/825/comm
kworker/0:0+ipv6_addrconf
root@ph3-rt [ ~ ]# cat /proc/825/stack
[<0>] rtnl_lock+0x15/0x20
[<0>] addrconf_verify_work+0xe/0x20
[<0>] process_one_work+0x1f4/0x470
[<0>] worker_thread+0x34/0x3f0
[<0>] kthread+0x160/0x180
[<0>] ret_from_fork+0x1f/0x40
[<0>] 0xffffffffffffffff
root@ph3-rt [ ~ ]#
```

```
void unregister_netdevice_many()
{
    ...
    rtnl_lock();
    ...
    flush_all_backlogs();
    ....
    rtnl_unlock();
}
```

```
void flush_all_backlogs()
{
    ...
    for_each_online_cpu(cpu) {
        queue_work_on(...);
    }
    ...
    for_each_online_cpu(cpu) {

        flush_work(...);
    }
}
```

Similar problems exist across many Linux subsystems:
ext4, cgroups, ftrace, sysctl …

Stability challenges with CPU intensive RT apps

flush_all_backlogs issue has been addressed in later kernels

```
commit 2de79ee27fdb52626ac4ac48ec6d8d52ba6f9047
Author: Paolo Abeni <pabeni@redhat.com>
Date:   Thu Sep 10 23:33:18 2020 +0200

    net: try to avoid unneeded backlog flush

    flush_all_backlogs() may cause deadlock on systems
    running processes with FIFO scheduling policy.

    The above is critical in -RT scenarios, where user-space
    specifically ensure no network activity is scheduled on
    the CPU running the mentioned FIFO process, but still get
    stuck.

    This commit tries to address the problem checking the
    backlog status on the remote CPUs before scheduling the
    flush operation. If the backlog is empty, we can skip it.

    v1 -> v2:
     - explicitly clear flushed cpu mask - Eric

    Signed-off-by: Paolo Abeni <pabeni@redhat.com>
    Signed-off-by: David S. Miller <davem@davemloft.net>

diff --git a/net/core/dev.c b/net/core/dev.c
```

Stability challenges with CPU intensive RT apps

flush_all_backlogs issue has been addressed in later kernels

```
commit 2de79ee27fdb52626ac4ac48ec6d8d52ba6f9047
Author: Paolo Abeni <pabeni@redhat.com>
Date:   Thu Sep 10 23:33:18 2020 +0200

    net: try to avoid unneeded backlog flush

    flush_all_backlogs() may cause deadlock on systems
    running processes with FIFO scheduling policy.

    The above is critical in -RT scenarios, where user-space
    specifically ensure no network activity is scheduled on
    the CPU running the mentioned FIFO process, but still get
    stuck.

    This commit tries to address the problem checking the
    backlog status on the remote CPUs before scheduling the
    flush operation. If the backlog is empty, we can skip it.

    v1 -> v2:
     - explicitly clear flushed cpu mask - Eric

    Signed-off-by: Paolo Abeni <pabeni@redhat.com>
    Signed-off-by: David S. Miller <davem@davemloft.net>

diff --git a/net/core/dev.c b/net/core/dev.c
```

but even on 6.6, we still see per-CPU kthreads starving in other areas…

Stability challenges with CPU intensive RT apps

flush_all_backlogs issue has been addressed in later kernels

```
commit 2de79ee27fdb52626ac4ac48ec6d8d52ba6f9047
Author: Paolo Abeni <pabeni@redhat.com>
Date:   Thu Sep 10 23:33:18 2020 +0200

    net: try to avoid unneeded backlog flush

    flush_all_backlogs() may cause deadlock on systems
    running processes with FIFO scheduling policy.

    The above is critical in -RT scenarios, where user-space
    specifically ensure no network activity is scheduled on
    the CPU running the mentioned FIFO process, but still get
    stuck.

    This commit tries to address the problem checking the
    backlog status on the remote CPUs before scheduling the
    flush operation. If the backlog is empty, we can skip it.

    v1 -> v2:
     - explicitly clear flushed cpu mask - Eric

    Signed-off-by: Paolo Abeni <pabeni@redhat.com>
    Signed-off-by: David S. Miller <davem@davemloft.net>

diff --git a/net/core/dev.c b/net/core/dev.c
```

but even on 6.6, we still see per-CPU kthreads starving in other areas…

PID	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	P	S	COMMAND
103	root	20	0	0	0	0	0.0	0.0	0:00.00	3	I	kworker/3:1-events
26	root	20	0	0	0	0	0.0	0.0	0:00.00	2	S	cpuhp/2
27	root	-51	0	0	0	0	0.0	0.0	0:00.00	2	S	idle_inject/2
28	root	-2	0	0	0	0	0.0	0.0	0:00.00	2	R	irq_work/2
29	root	rt	0	0	0	0	0.0	0.0	0:00.07	2	S	migration/2
30	root	-2	0	0	0	0	0.0	0.0	0:00.00	2	R	rcuc/2
31	root	-2	0	0	0	0	0.0	0.0	0:00.00	2	R	ktimers/2
32	root	20	0	0	0	0	0.0	0.0	0:00.00	2	R	ksoftirqd/2
33	root	20	0	0	0	0	0.0	0.0	0:00.00	2	I	kworker/2:0-slub_flushwq
102	root	20	0	0	0	0	0.0	0.0	0:00.00	2	I	kworker/2:1-events
1545	root	-91	0	2640	1024	1024	99.9	0.0	1:13.23	2	R	loop-rt

Stability challenges with CPU intensive RT apps

flush_all_backlogs issue has been addressed in later kernels

```
commit 2de79ee27fdb52626ac4ac48ec6d8d52ba6f9047
Author: Paolo Abeni <pabeni@redhat.com>
Date:   Thu Sep 10 23:33:18 2020 +0200

    net: try to avoid unneeded backlog flush

    flush_all_backlogs() may cause deadlock on systems
    running processes with FIFO scheduling policy.

    The above is critical in -RT scenarios, where user-space
    specifically ensure no network activity is scheduled on
    the CPU running the mentioned FIFO process, but still get
    stuck.

    This commit tries to address the problem checking the
    backlog status on the remote CPUs before scheduling the
    flush operation. If the backlog is empty, we can skip it.

    v1 -> v2:
     - explicitly clear flushed cpu mask - Eric

    Signed-off-by: Paolo Abeni <pabeni@redhat.com>
    Signed-off-by: David S. Miller <davem@davemloft.net>

diff --git a/net/core/dev.c b/net/core/dev.c
```

but even on 6.6, we still see per-CPU kthreads starving in other areas…

PID	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	P	S	COMMAND
103	root	20	0	0	0	0	0.0	0.0	0:00.00	3	I	kworker/3:1-events
26	root	20	0	0	0	0	0.0	0.0	0:00.00	2	S	cpuhp/2
27	root	-51	0	0	0	0	0.0	0.0	0:00.00	2	S	idle_inject/2
28	root	-2	0	0	0	0	0.0	0.0	0:00.00	2	R	irq_work/2
29	root	rt	0	0	0	0	0.0	0.0	0:00.07	2	S	migration/2
30	root	-2	0	0	0	0	0.0	0.0	0:00.00	2	R	rcuc/2
31	root	-2	0	0	0	0	0.0	0.0	0:00.00	2	R	ktimers/2
32	root	20	0	0	0	0	0.0	0.0	0:00.00	2	R	ksoftirqd/2
33	root	20	0	0	0	0	0.0	0.0	0:00.00	2	I	kworker/2:0-slub_flushwq
102	root	20	0	0	0	0	0.0	0.0	0:00.00	2	I	kworker/2:1-events
1545	root	-91	0	2640	1024	1024	99.9	0.0	1:13.23	2	R	loop-rt

PID	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	P	S	COMMAND
360	root	20	0	0	0	0	0.0	0.0	0:00.11	1	I	kworker/u16:6-ev
1	root	20	0	166800	11776	8320	0.0	0.3	0:00.68	1	D	systemd
269	root	19	-1	86928	48256	46976	0.0	1.1	0:00.53	0	D	systemd-journal
585	root	20	0	241176	8704	7808	0.0	0.2	0:00.00	1	D	gdm3
1549	root	20	0	166804	6456	2944	0.0	0.2	0:00.00	0	D	(wnloader)

Stability challenges with CPU intensive RT apps

but even on 6.6, we still see per-CPU kthreads starving in other areas…

```
srivatsa@ubuntu2204:~$ sudo cat /proc/1/comm
systemd
srivatsa@ubuntu2204:~$ sudo cat /proc/1/stack
[<0>] __wait_rcu_gp+0x143/0x150
[<0>] synchronize_rcu+0x147/0x170
[<0>] rcu_sync_enter+0x58/0xf0
[<0>] cgroup_procs_write_start+0x105/0x180
[<0>] __cgroup_procs_write+0x5d/0x190
[<0>] cgroup_procs_write+0x17/0x30
[<0>] cgroup_file_write+0x8c/0x190
[<0>] kernfs_fop_write_iter+0x15f/0x1f0
[<0>] vfs_write+0x2f8/0x420
[<0>] ksys_write+0x6a/0xf0
[<0>] __x64_sys_write+0x19/0x30
[<0>] do_syscall_64+0x59/0x90
[<0>] entry_SYSCALL_64_after_hwframe+0x6e/0xd8
srivatsa@ubuntu2204:~$
```

```
srivatsa@ubuntu2204:~$ sudo cat /proc/269/comm
systemd-journal
srivatsa@ubuntu2204:~$ sudo cat /proc/269/stack
[<0>] rt_mutex_schedule+0x24/0x50
[<0>] __rt_mutex_slowlock_locked.constprop.0+0xf1/0x260
[<0>] proc_cgroup_show+0x4c/0x450
[<0>] proc_single_show+0x56/0x110
[<0>] seq_read_iter+0x132/0x4e0
[<0>] seq_read+0xa5/0xe0
[<0>] vfs_read+0xb1/0x320
[<0>] ksys_read+0x6a/0xf0
[<0>] __x64_sys_read+0x19/0x30
[<0>] do_syscall_64+0x59/0x90
[<0>] entry_SYSCALL_64_after_hwframe+0x6e/0xd8
srivatsa@ubuntu2204:~$
```

```
srivatsa@ubuntu2204:~$ sudo cat /proc/585/comm
gdm3
srivatsa@ubuntu2204:~$ sudo cat /proc/585/stack
[<0>] rt_mutex_schedule+0x24/0x50
[<0>] __rt_mutex_slowlock_locked.constprop.0+0xf1/0x260
[<0>] proc_cgroup_show+0x4c/0x450
[<0>] proc_single_show+0x56/0x110
[<0>] seq_read_iter+0x132/0x4e0
[<0>] seq_read+0xa5/0xe0
[<0>] vfs_read+0xb1/0x320
[<0>] ksys_read+0x6a/0xf0
[<0>] __x64_sys_read+0x19/0x30
[<0>] do_syscall_64+0x59/0x90
[<0>] entry_SYSCALL_64_after_hwframe+0x6e/0xd8
srivatsa@ubuntu2204:~$
```

```
srivatsa@ubuntu2204:~$ sudo cat /proc/1549/comm
(wnloader)
srivatsa@ubuntu2204:~$ sudo cat /proc/1549/stack
[<0>] rt_mutex_schedule+0x24/0x50
[<0>] __rt_mutex_slowlock_locked.constprop.0+0xf1/0x260
[<0>] cgroup_kn_lock_live+0x47/0xf0
[<0>] __cgroup_procs_write+0x3e/0x190
[<0>] cgroup_procs_write+0x17/0x30
[<0>] cgroup_file_write+0x8c/0x190
[<0>] kernfs_fop_write_iter+0x15f/0x1f0
[<0>] vfs_write+0x2f8/0x420
[<0>] ksys_write+0x6a/0xf0
[<0>] __x64_sys_write+0x19/0x30
[<0>] do_syscall_64+0x59/0x90
[<0>] entry_SYSCALL_64_after_hwframe+0x6e/0xd8
srivatsa@ubuntu2204:~$
```

Resources for more info…

Linux PREEMPT_RT patches

- realtime:start [Wiki] (linuxfoundation.org)
- realtime:preempt_rt_versions [Wiki] (linuxfoundation.org)
- realtime:documentation:start [Wiki] (linuxfoundation.org)

Linux Plumbers Conference

- Real-Time & Scheduling Microconference

Q & A