Linux Kernel Support for Kernel Thread Starvation Avoidance

Real-Time MC, Linux Plumbers Conference 2021

Sharan Turlapati (sturlapati@vmware.com)
Srivatsa Bhat (srivatsa@csail.mit.edu)

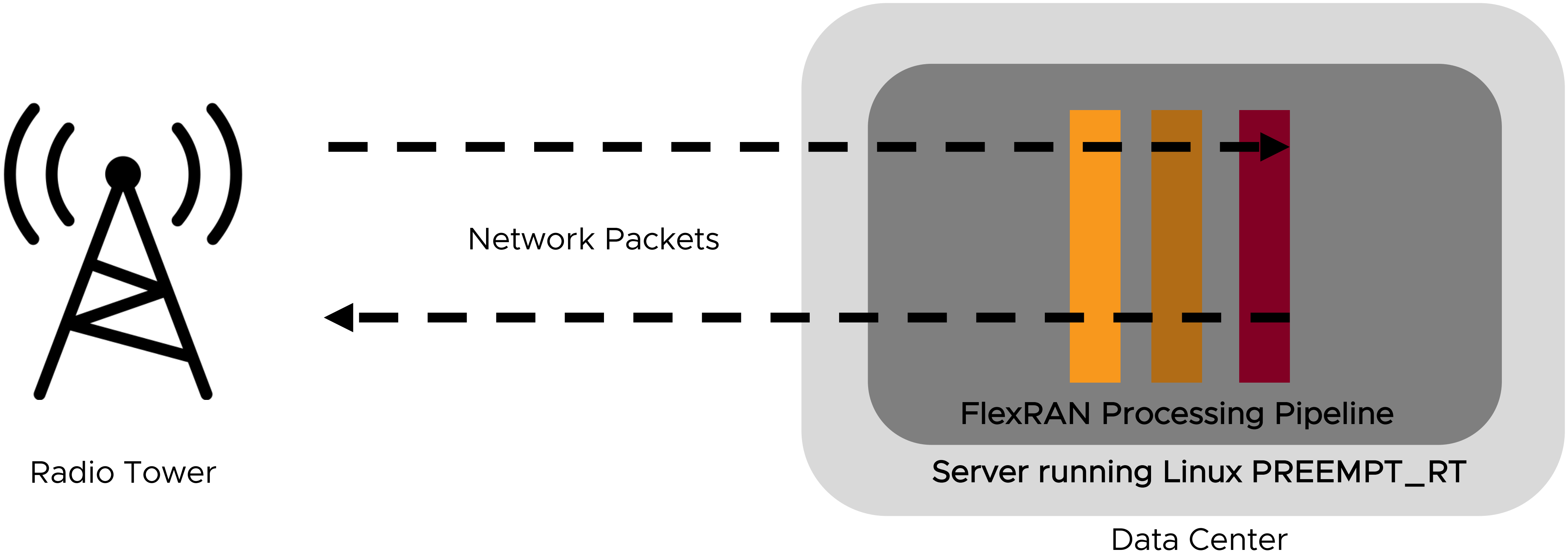
VMware Photon OS Team

21 Sep 2021

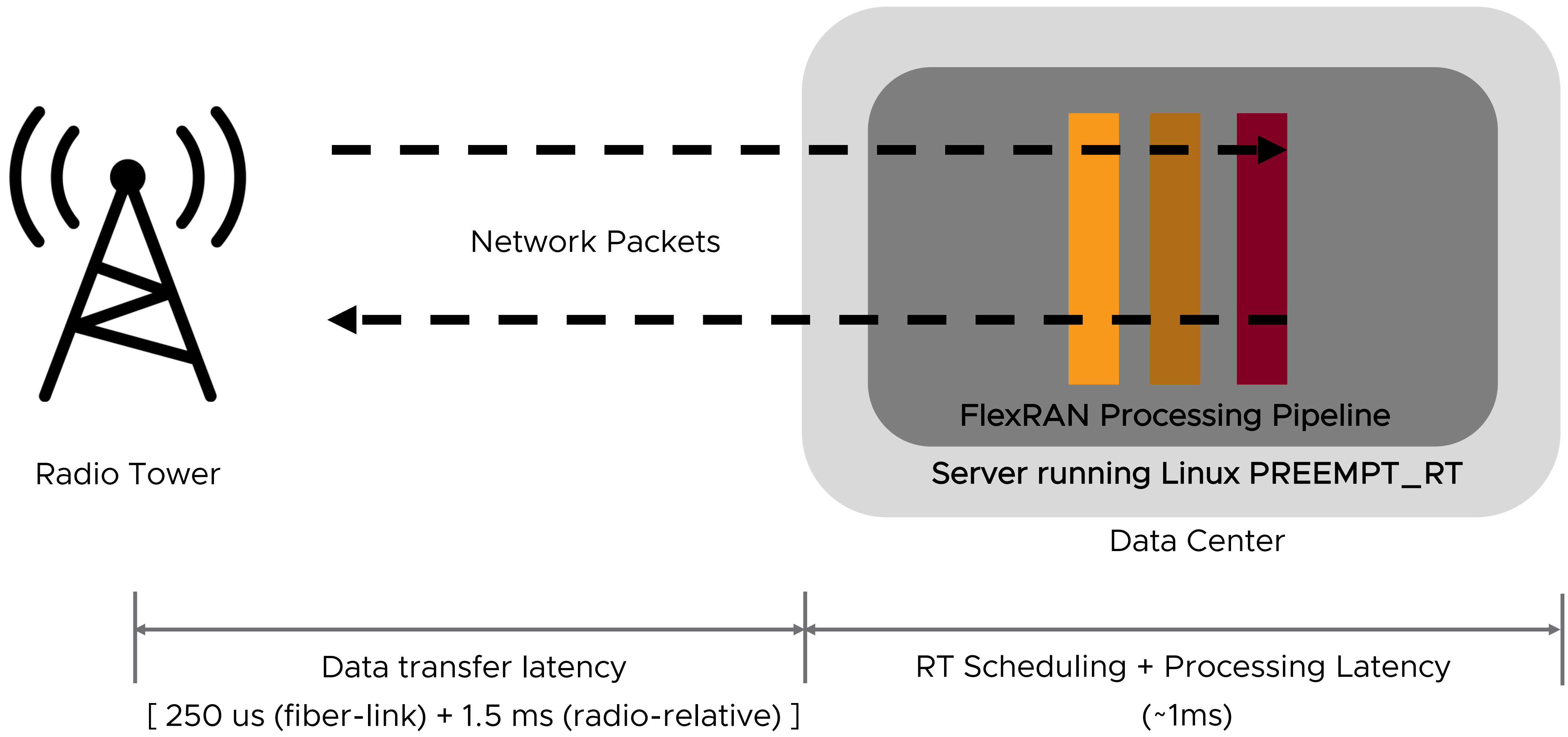
vmware

Agenda

Overview of Telco/RAN : Radio Access Network for 5G

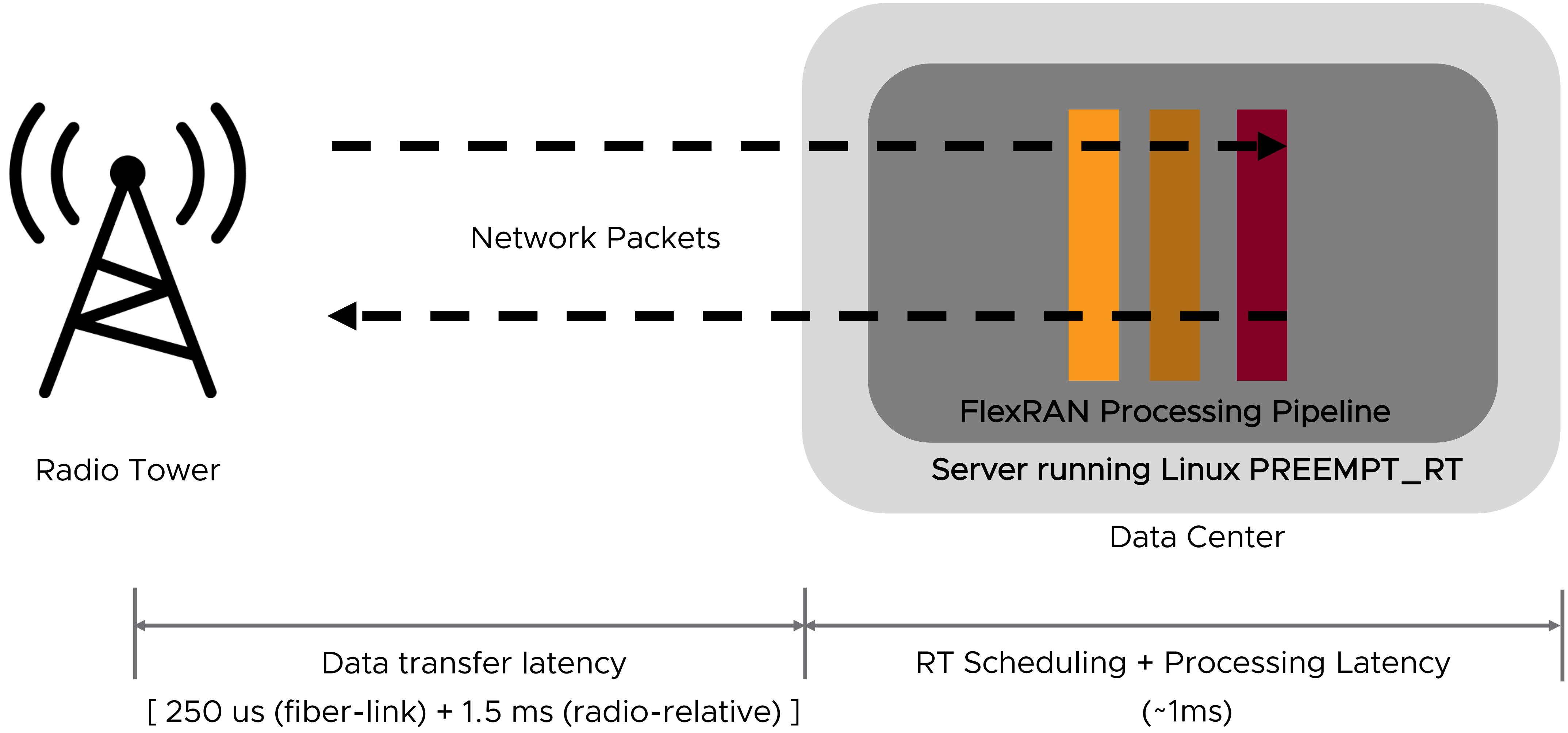


Overview of Telco/RAN : Radio Access Network for 5G



vmware

Overview of Telco/RAN : Radio Access Network for 5G

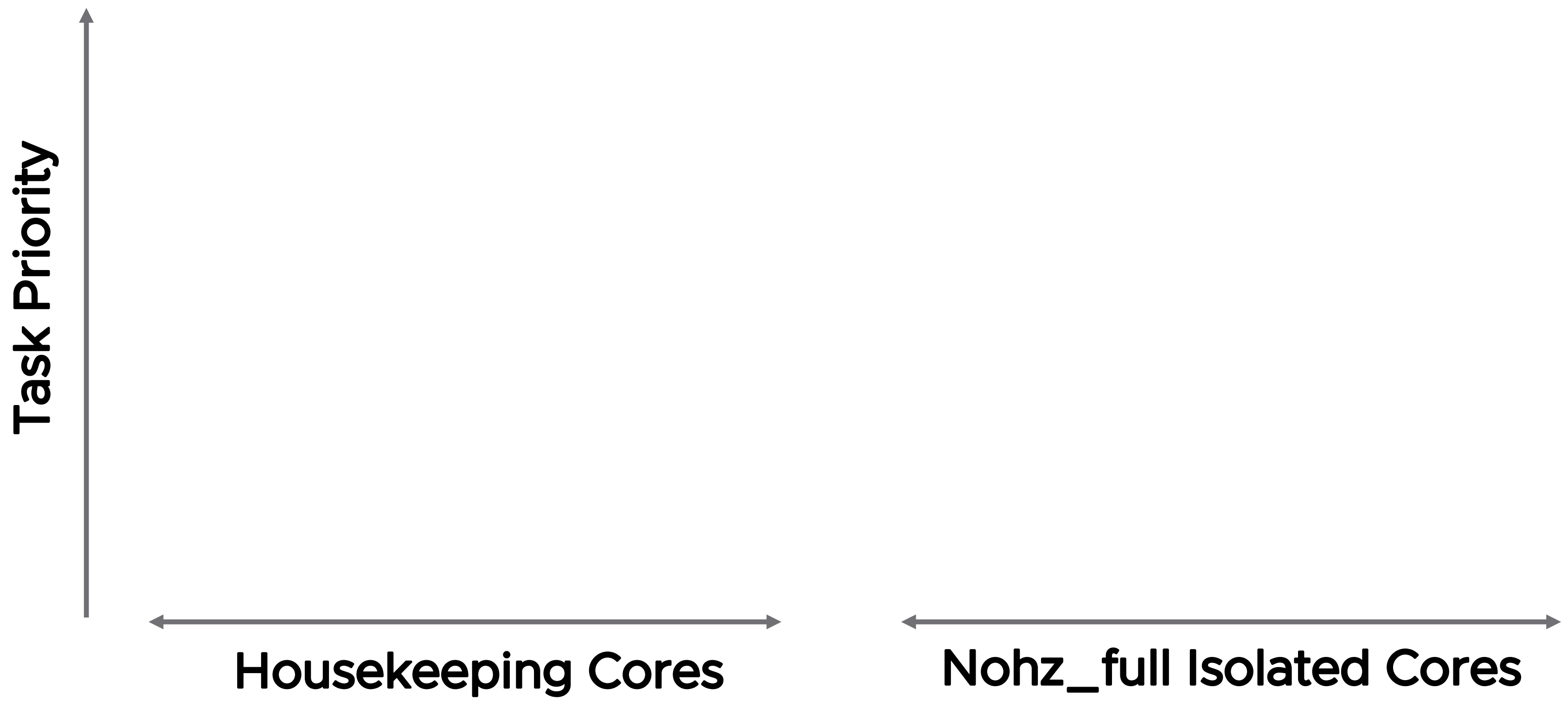


Fixed total latency budget for packet Tx + processing + ack (< 3ms)

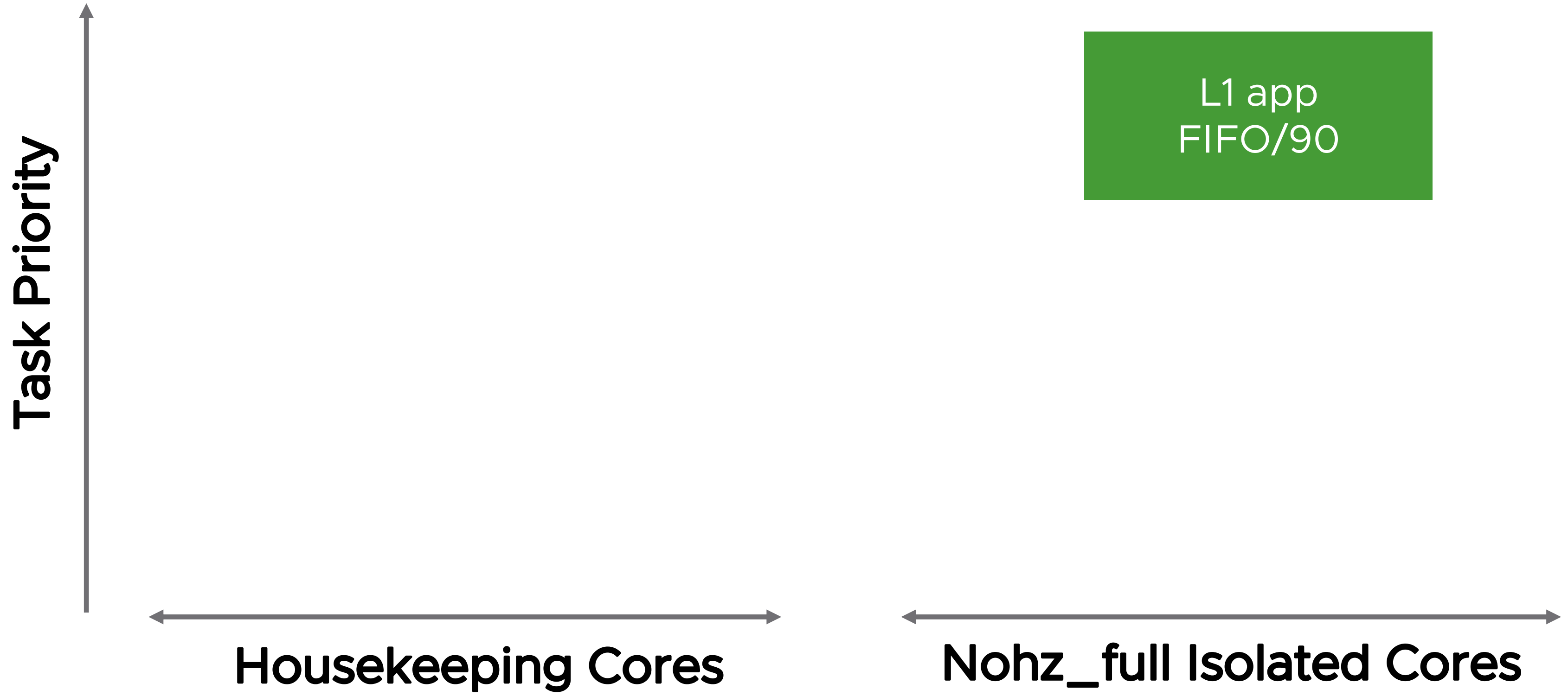
Cyclictest latency < 10us

vmware

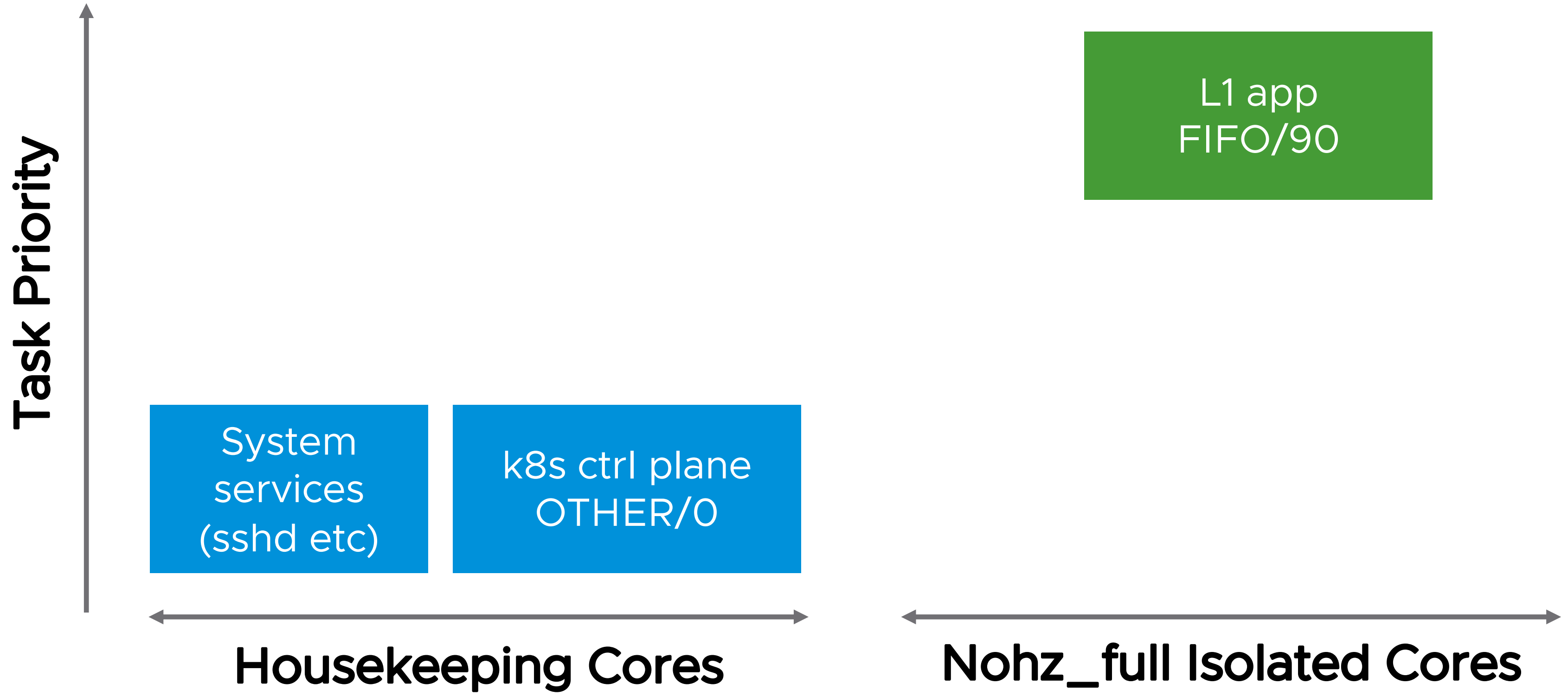
Problem Statement



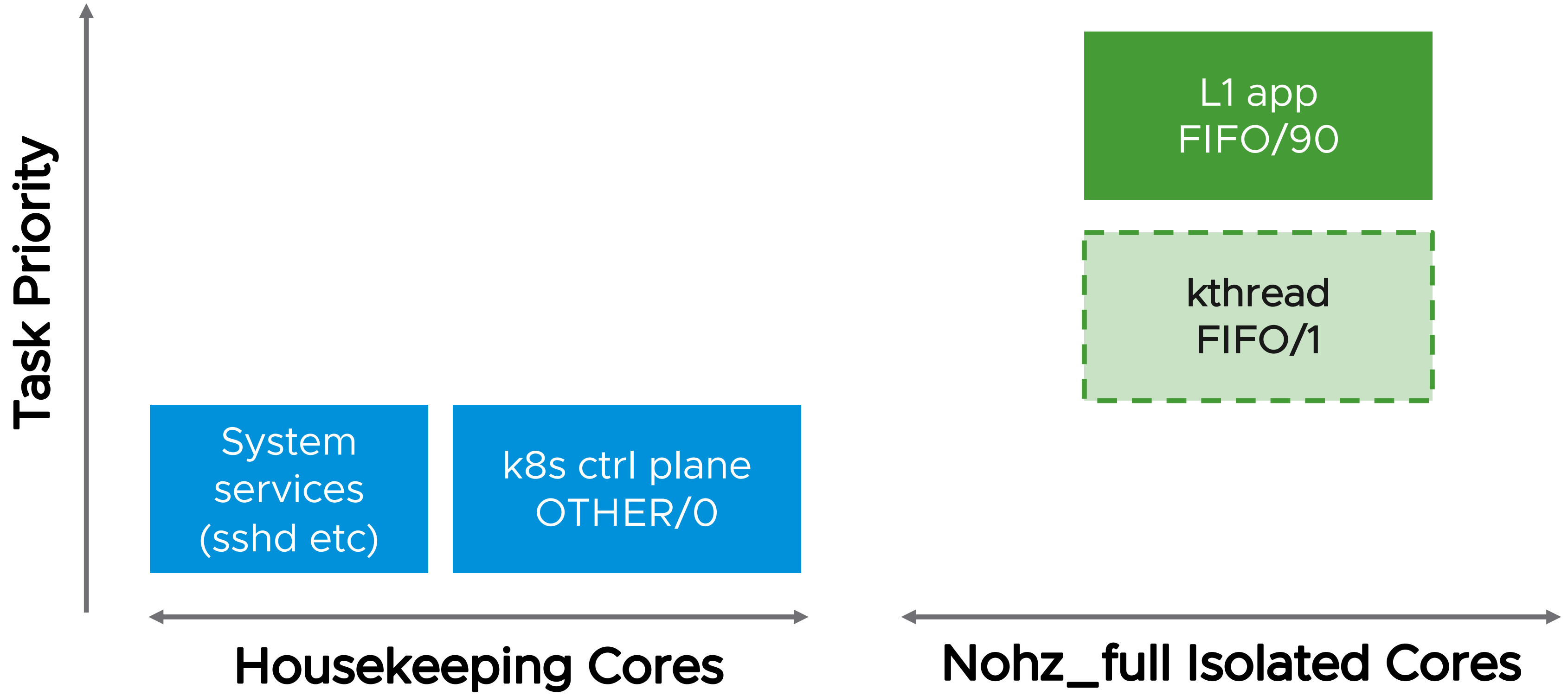
Problem Statement



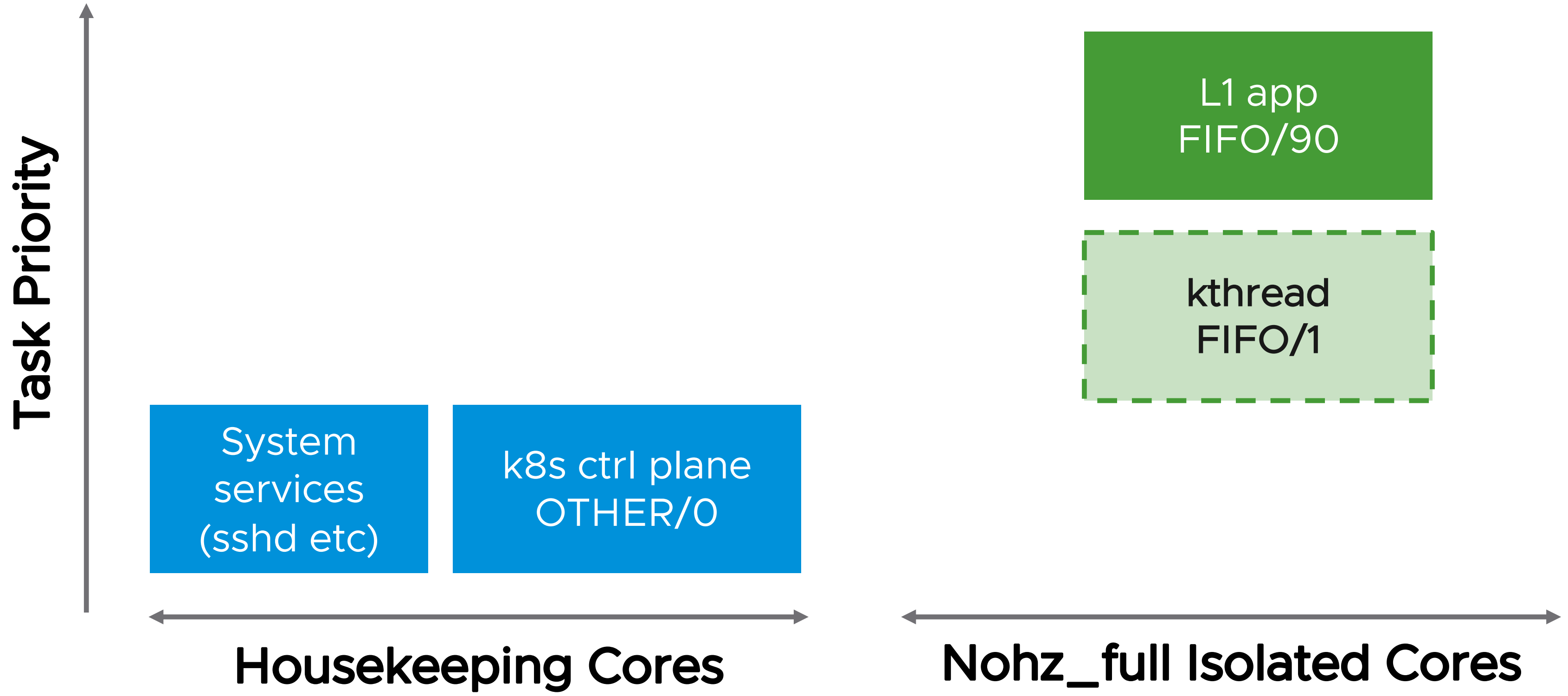
Problem Statement



Problem Statement

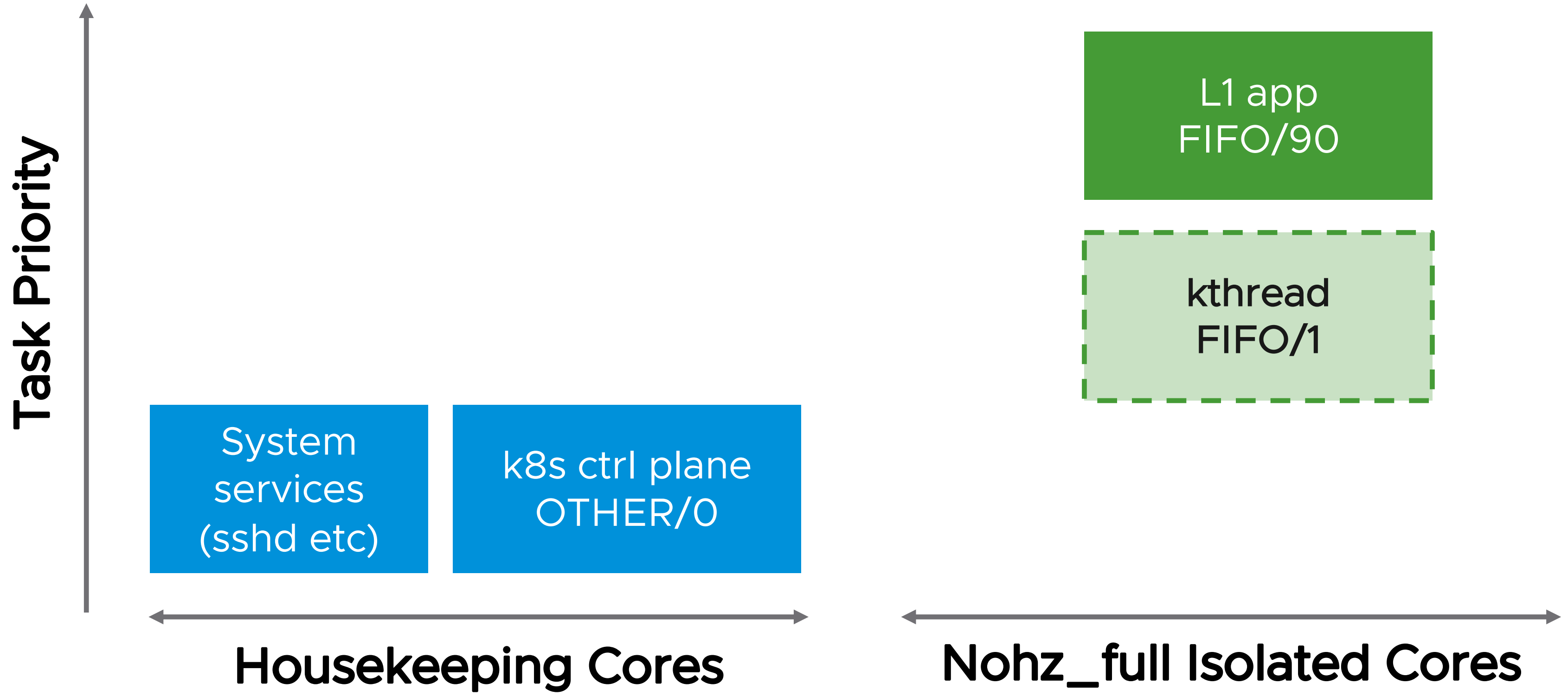


Problem Statement



Problem: Starved kthreads lead to cascading lockups (hang)

Problem Statement



Problem: Starved kthreads lead to cascading lockups (hang)

Goal: OS must remain stable, limiting the fault-domain to the RT app

vmware

Problem Statement Example: Container destroy causes hang

Problem Statement Example: Container destroy causes hang

Reproducer:

1. Run high prio CPU hog on an isolated CPU
2. Create & destroy a docker container on a housekeeping CPU

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

CPU 3 is nohz_full isolated

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

loop-rt has high RT prio (SCHED_FIFO/55)

Two runnable tasks on CPU 3: loop-rt and kworker/3

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

loop-rt hogs the CPU

kworker/3 is starved

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

```
Stalld DEBUG: Dumping Stack for dockerd(PID = 1021
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.124+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+0x2b5/0x3b0
[<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
```

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

```
Stalld DEBUG: Dumping Stack for dockerd(PID = 1021
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.124+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+0x2b5/0x3b0
[<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
```

```
static int rtnetlink_rcv_msg(…)
{
    rtnl_lock();
    ->flush_all_backlogs();
    rtnl_unlock();
}
```

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

```
Stalld DEBUG: Dumping Stack for systemd-network
[<0>] rtnetlink_rcv_msg+0xda/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2b5/0x3b0
[<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
```

```
Stalld DEBUG: Dumping Stack for dockerd(PID = 1021
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.124+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+0x2b5/0x3b0
[<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
```

```
static int rtnetlink_rcv_msg(…)
{
    rtnl_lock();
    ->flush_all_backlogs();
    rtnl_unlock();
}
```

Problem Statement Example: Container destroy causes hang

PID	USER	PR	NI	VIRT	RES	%CPU	%MEM	TIME+	P	S	COMMAND
37	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[cpuhp/3]
38	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.09	3	S	[migration/3]
39	root	rt	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[posixcputmr/3]
40	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[rcuc/3]
41	root	-2	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ktimersoftd/3]
42	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	S	[ksoftirqd/3]
43	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0-mm_percpu_wq]
44	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:0H-events_highpri]
270	root	20	0	0.0m	0.0m	0.0	0.0	0:00.00	3	I	[kworker/3:1-mm_percpu_wq]
1334	root	0	-20	0.0m	0.0m	0.0	0.0	0:00.00	3	R	[kworker/3:1H-events_highpri]
3068	root	-56	0	2.1m	0.7m	99.9	0.0	1:32.52	3	R	./loop-rt

```
Stalld DEBUG: Dumping Stack for dockerd(PID = 1021
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.124+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+0x2b5/0x3b0
[<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
```

```
Stalld DEBUG: Dumping Stack for systemd-network
[<0>] rtnetlink_rcv_msg+0xda/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rtnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2b5/0x3b0
[<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
```

```
static int rtnetlink_rcv_msg(…)
{
    rtnl_lock();
    ->flush_all_backlogs();
    rtnl_unlock();
}
```

Problem pattern is pervasive in Linux. Ex: ext4, cgroups, ftrace, sysctl etc.

vmware®

Existing solutions & limitations: stalld

Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)

Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
- Critical to RAN deployments to maintain stability

Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
- Critical to RAN deployments to maintain stability

Limitations of stalld

Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
- Critical to RAN deployments to maintain stability

Limitations of stalld

Limitation	Reasons
Scalability	Stallds threads run on housekeeping CPUs
Stalld can get starved itself	Competes for time on housekeeping CPUs RT prio stalld is risky – can ***cause*** stalls itself!
Unreliable logging	systemd-journald can get stuck Verbose logging gets stalld itself stuck
Trade-off: Response-time vs CPU consumption	Per-CPU threads vs single-threaded mode

vmware®

Design Goals for Stall Monitor

And why a kernel-based implementation can meet them

Design Goals for Stall Monitor

And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain

vmware®

Design Goals for Stall Monitor

And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain
- Ensure scalability
  - Per-cpu kthreads most susceptible to starvation
  - Per-cpu based scheduler hooks scale well

Design Goals for Stall Monitor

And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain
- Ensure scalability
  - Per-cpu kthreads most susceptible to starvation
  - Per-cpu based scheduler hooks scale well
- Monitor and boost efficiently
  - Avoid unnecessary periodic monitoring
  - sched events like wakeup and dequeue equip the scheduler to take decisions efficiently

vmware

Design Goals for Stall Monitor

And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain
- Ensure scalability
  - Per-cpu kthreads most susceptible to starvation
  - Per-cpu based scheduler hooks scale well
- Monitor and boost efficiently
  - Avoid unnecessary periodic monitoring
  - sched events like wakeup and dequeue equip the scheduler to take decisions efficiently
- Guarantee responsiveness
  - We must be able to prevent starvation under any scenario
  - Scheduler invocations inevitably offer the opportunity to monitor for starvation

vmware®

Design Features of Stall Monitor

Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU

Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)

    OR
  - Track the boosted priority duration (boost_duration_time)

Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)

    OR
  - Track the boosted priority duration (boost_duration_time)

- Boost only one starving kthread on a CPU at any given time

vmware

Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)

    OR
  - Track the boosted priority duration (boost_duration_time)
- Boost only one starving kthread on a CPU at any given time
- Boost or deboost happens in hardirq context of the hrtimer

Design Features of Stall Monitor

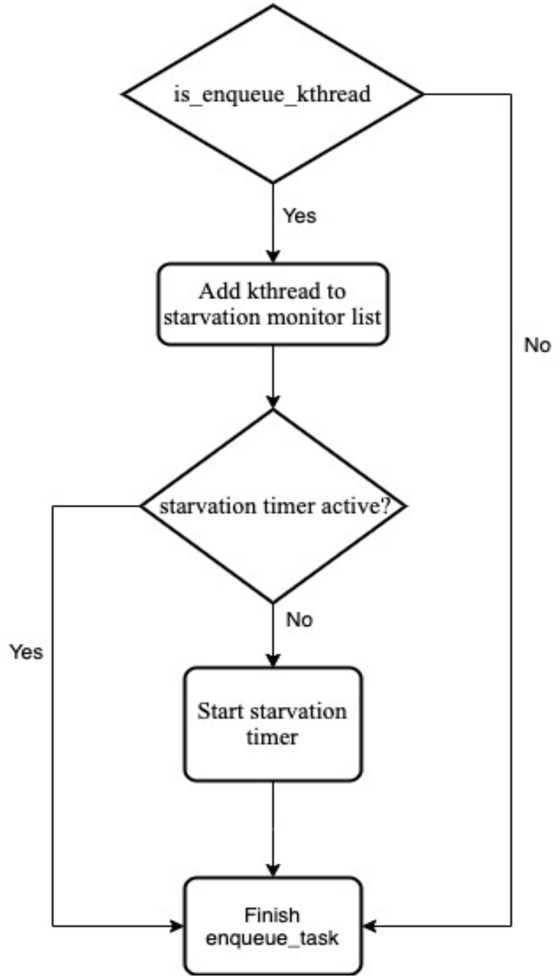
- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)

    OR
  - Track the boosted priority duration (boost_duration_time)

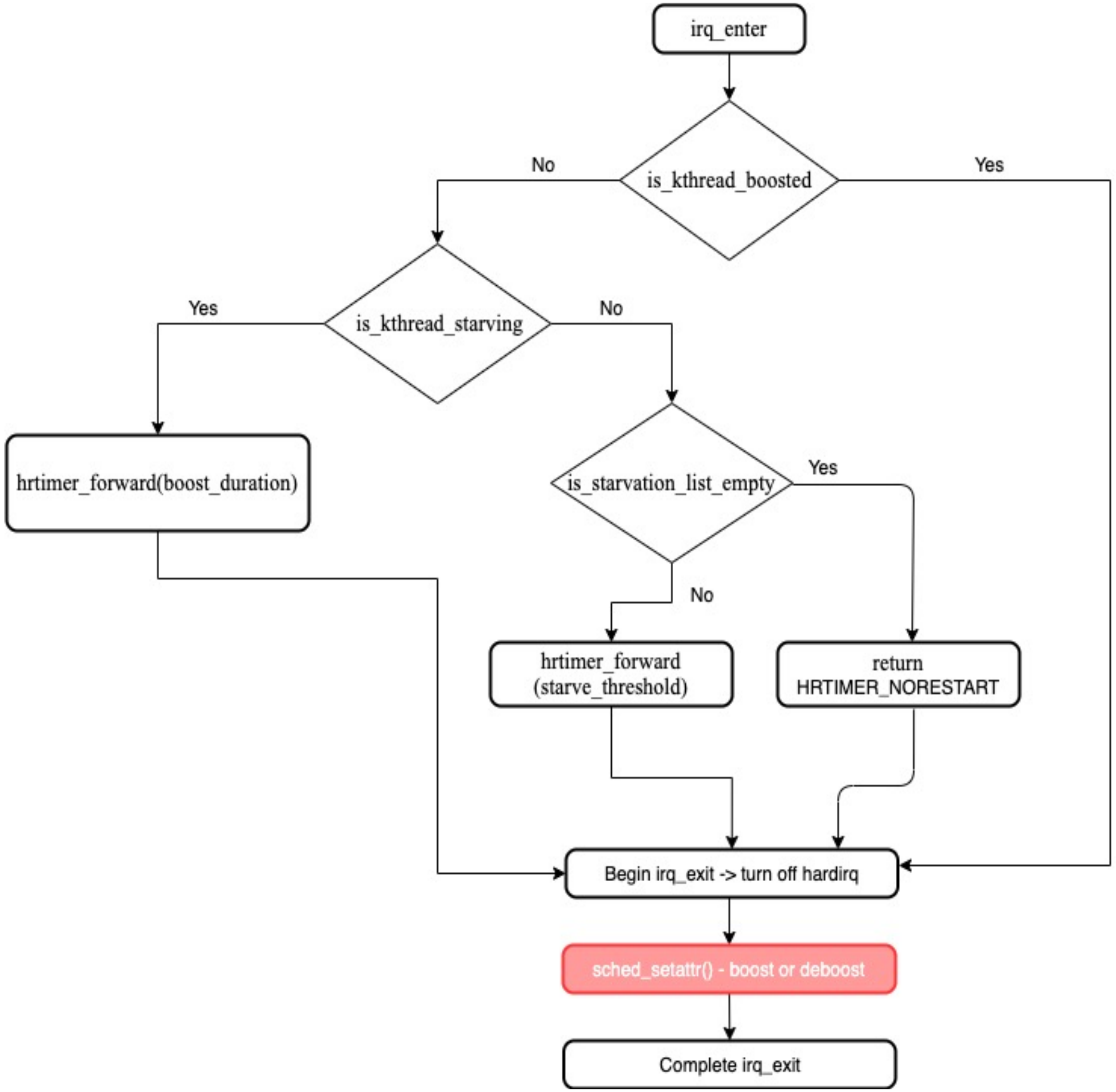
- Boost only one starving kthread on a CPU at any given time
- Boost or deboost happens in hardirq context of the hrtimer
- User defined OS jitter
  - With user configurable starvation_threshold_time, boost_duration_time as well as SCHED_DEADLINE parameters

Implementation of Stall Monitor

Enqueue Task



Hrtimer callback



vmware®

Challenges & Open Questions

- Priority boosting must happen in hardirq context
  - Cannot create more kthreads. Or can we use CPU stopper threads?
  - Better alternatives?
- Restrict the monitoring and boosting to isolcpus only?
- How much latency does it introduce?

Thank you!

Additional Data Points

- CFS code already has functions to track wait times spent by task on the runqueue –
  - Handled by update_stats_wait_start() and update_stats_wait_end()
  - This needs to be added to RT (SCHED_FIFO and SCHED_RR)
- __sched_setscheduler invoked by sched_setattr() has checks on pi being invoked from interrupt context. This is suspectedly due to rt_mutex_adjust_prio_chain() that enables interrupts using raw_spin_unlock_irq(&task->pi_lock) unconditionally