How Linux RT_PREEMPT Works

A common observation about real time systems is that the cost of the increased determinism of real time is decreased throughput and increased average latency.

This presentation enumerates some of the design choices and implementation that enable Linux PREEMPT_RT_FULL real time and the resulting performance implications.

Frank Rowand, Sony Network Entertainment

December 9, 2011

111209_1100

Assertion

The cost of the increased determinism of real time

- Increased average latency

- Decreased throughput

Some Random Data

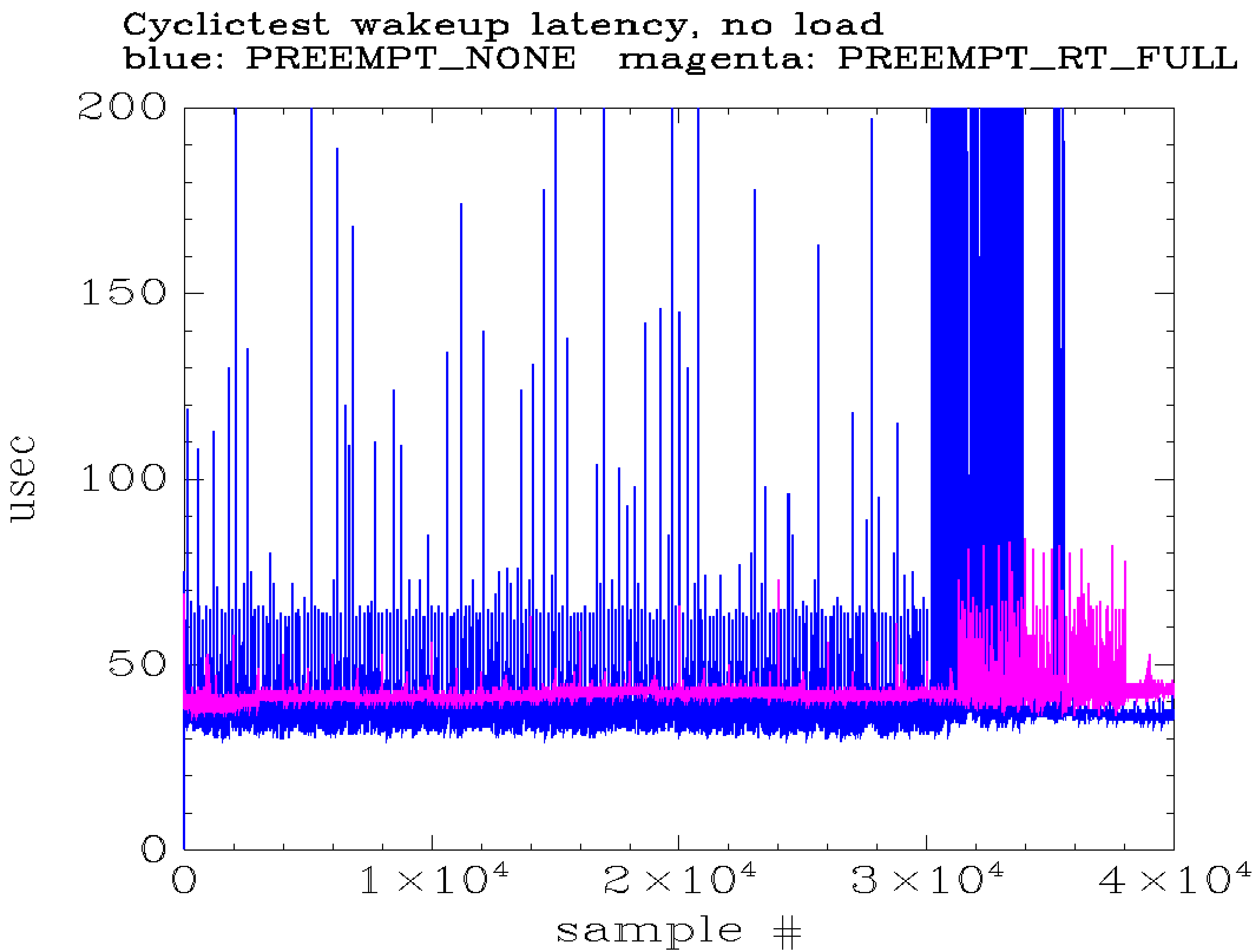
Compare the latency of an application on kernels built with:

1) CONFIG_PREEMPT_NONE
2) CONFIG_PREEMPT_RT_FULL

Some Random Data

Test System:

- ARM11 MPCore development system
- 4 cpus
- 210 Mhz processor clock
- L1 cache 64 Kbyte per processor
- L2 cache 1 Mbyte unified
- Linux 3.0.6-rt17


Cyclictest wakeup latency, no load
blue: PREEMPT_NONE magenta: PREEMPT_RT_FULL
usec
0
50
100
150
200
sample #
0
1×10⁴
2×10⁴
3×10⁴
4×10⁴

Latency (Response Time)

Kernel without RT patchset:

+ smaller average
+ smaller minimum
- larger maximum

PREEMPT RT enabled:

- larger average
- larger minimum
+ smaller maximum
+ more consistent

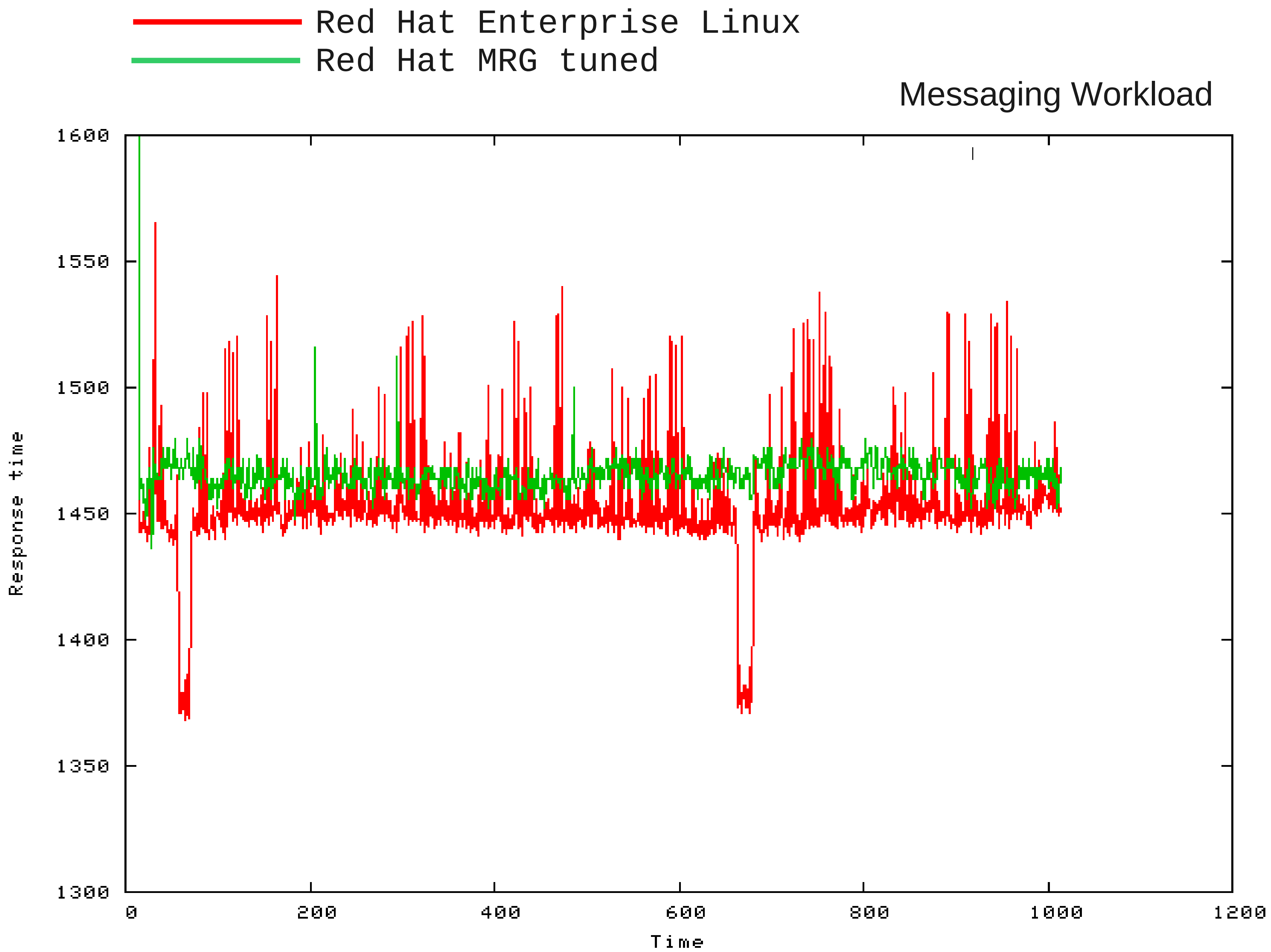
Statistics

	Min	Avg	Max
	----	----	----
PREEMPT_NONE	29	38	9186
PREEMPT_RT_FULL	35	41	95

Latency (Response Time)

Next graph shows an old kernel, circa 2009

Hardware configuration: unknown, server class



source: Red Hat

Latency

The previous graphs illustrate real results on:

- variety of kernel versions
- range of hardware, from embedded to server

Some Random Data

Compare the throughput of an application on kernels built with:

1) CONFIG_PREEMPT_NONE
2) CONFIG_PREEMPT_RT_FULL

Some Random Data

The workload I used for the throughput results

- is not realistic
- is not reasonable
- violates real time application design rules
- is stupid!
- but was easy to implement...

Some Random Data

Test System (same as first test system):

- ARM11 MPCore development system
- 4 cpus
- 210 Mhz processor clock
- L1 cache 64 Kbyte per processor
- L2 cache 1 Mbyte unified
- Linux 3.0.6-rt17

Test Variables

UP vs SMP

SMP, maxcpus=4 vs SMP maxcpus=1

workload: SCHED_FIFO vs SCHED_NORMAL

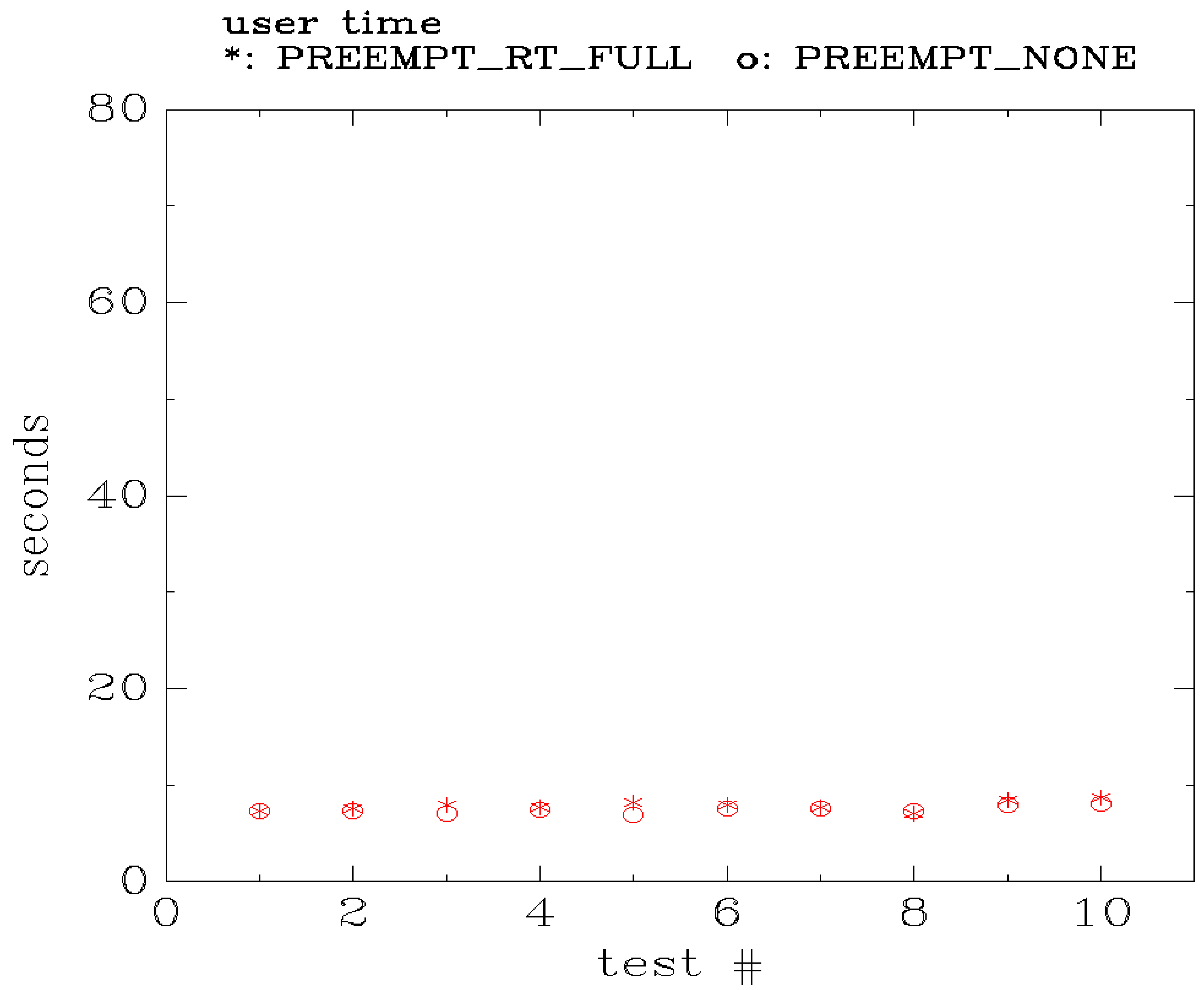
1, 2, or 4 instances of the workload

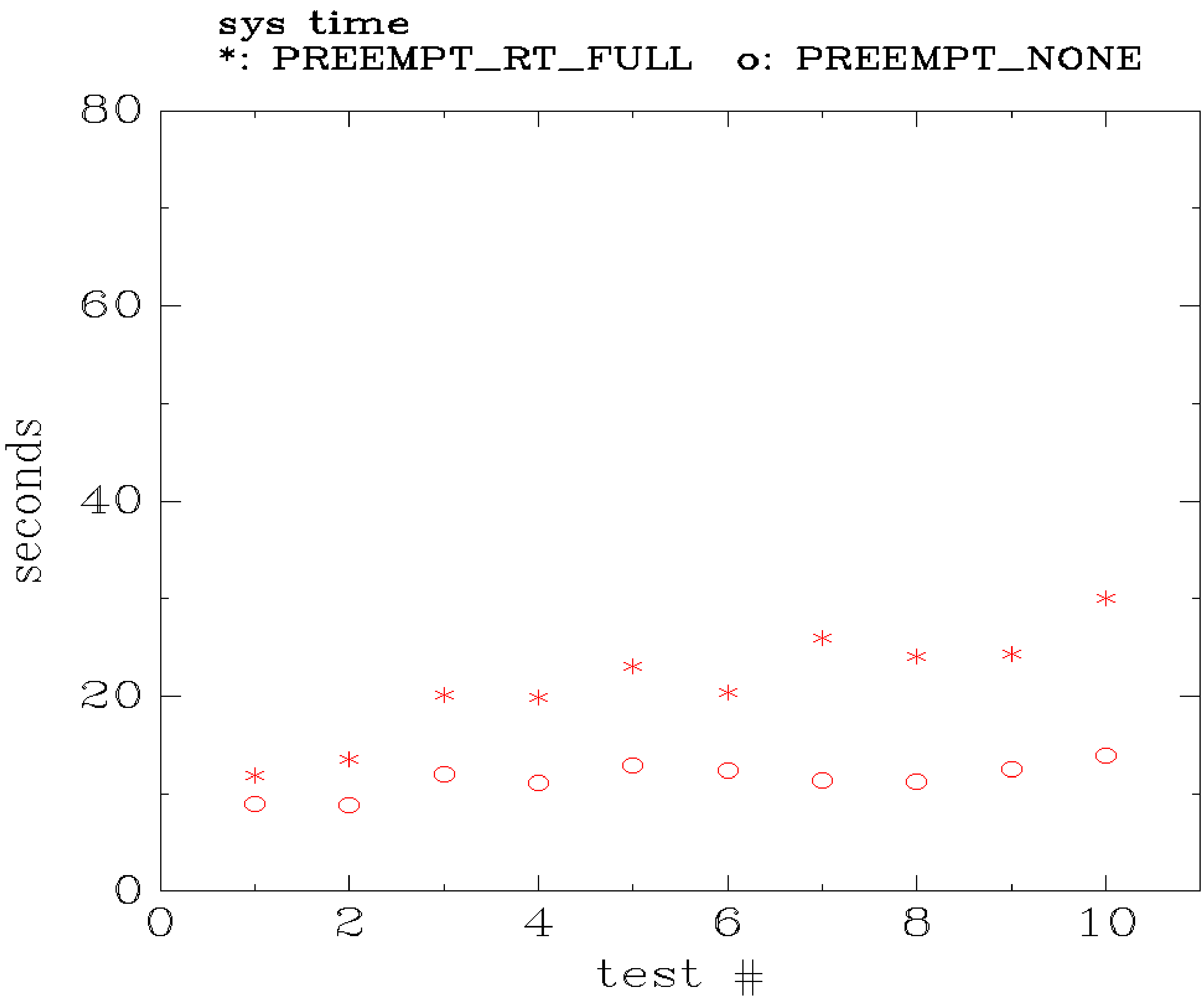
Permutations of variables results in 10 tests

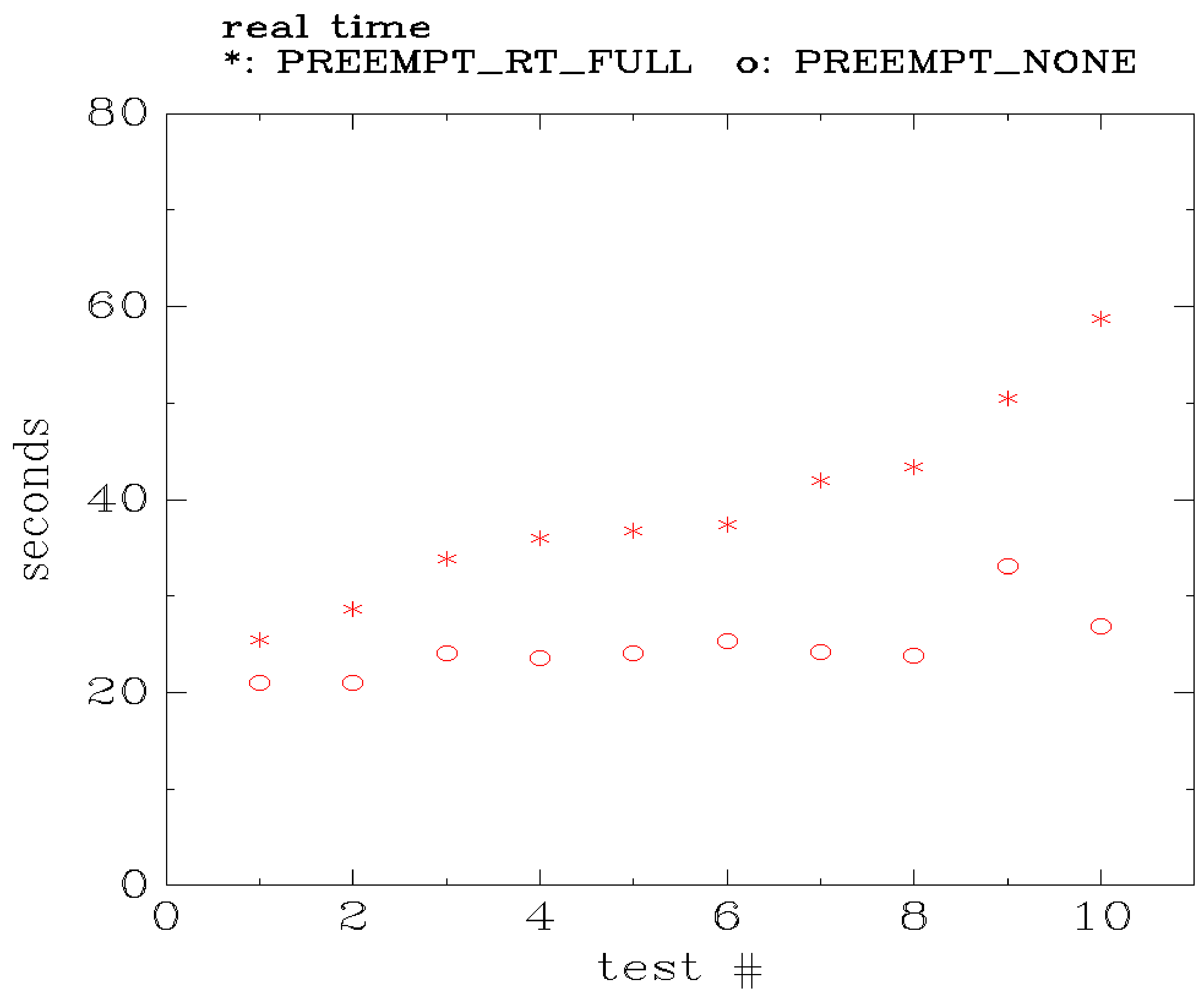
Test Data

The following graphs show the duration of each test.

Longer duration means lower throughput, so larger duration is worse throughput.


user time
*: PREEMPT_RT_FULL o: PREEMPT_NONE
80
60
40
20
0
seconds
0
2
4
6
8
10
test #


sys time
*: PREEMPT_RT_FULL o: PREEMPT_NONE
80
60
40
20
0
seconds
0
2
4
6
8
10
test #



real time
*: PREEMPT_RT_FULL o: PREEMPT_NONE
80
60
40
20
0
seconds
0
2
4
6
8
10
test #

Throughput

The previous graphs illustrate real results on:

- variety of test loads
- range of hardware, UP and SMP

Assertion

The cost of the increased determinism of real time

- Increased average latency

- Decreased throughput

This is true for Linux PREEMPT_RT_FULL.

Part 2

This presentation enumerates some of the design choices and implementation that enable Linux PREEMPT_RT_FULL real time and the resulting performance implications.

Enabling real-time Linux

- preemptible kernel
- locking
- threaded interrupt handlers
- threaded softirq

Non-Preemptible Kernel

When a task invokes a system call, the system call must complete (or sleep due to blocking on a resource) before another task can be scheduled.

Preemption can not occur during the execution of the system call.

Non-Preemptible Kernel

Preemption can not occur during the execution of the system call.

Scheduling may occur on:

- completion of system call
- system call sleeping

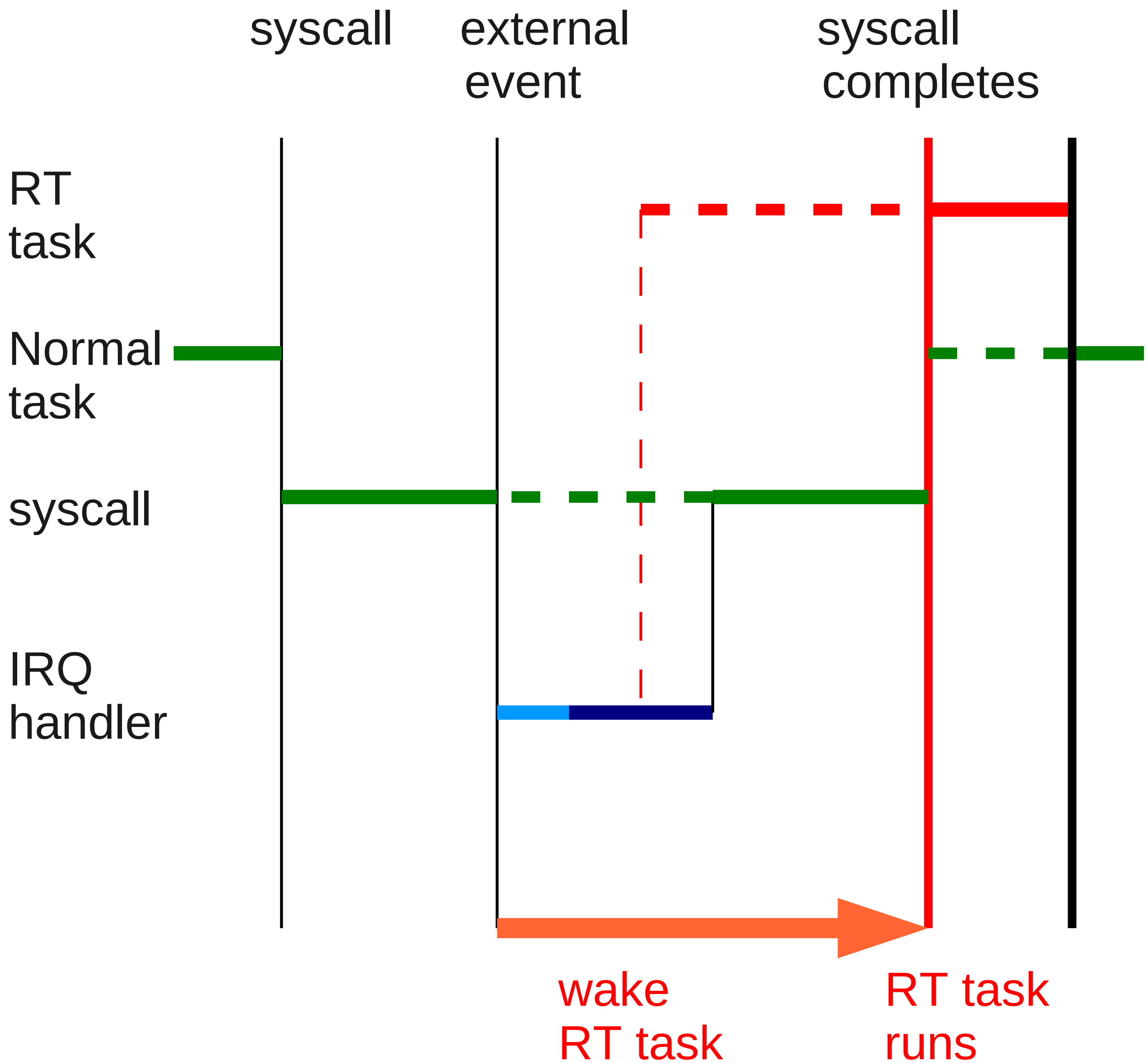
Non-Preemptible Kernel

Problems of typical non-preemptible kernel:

- kernel path lengths non-deterministic
- longest kernel path has long duration
- large variance in kernel path length

Non-Preemptible Kernel

Next slide illustrates non-preemptible kernel.

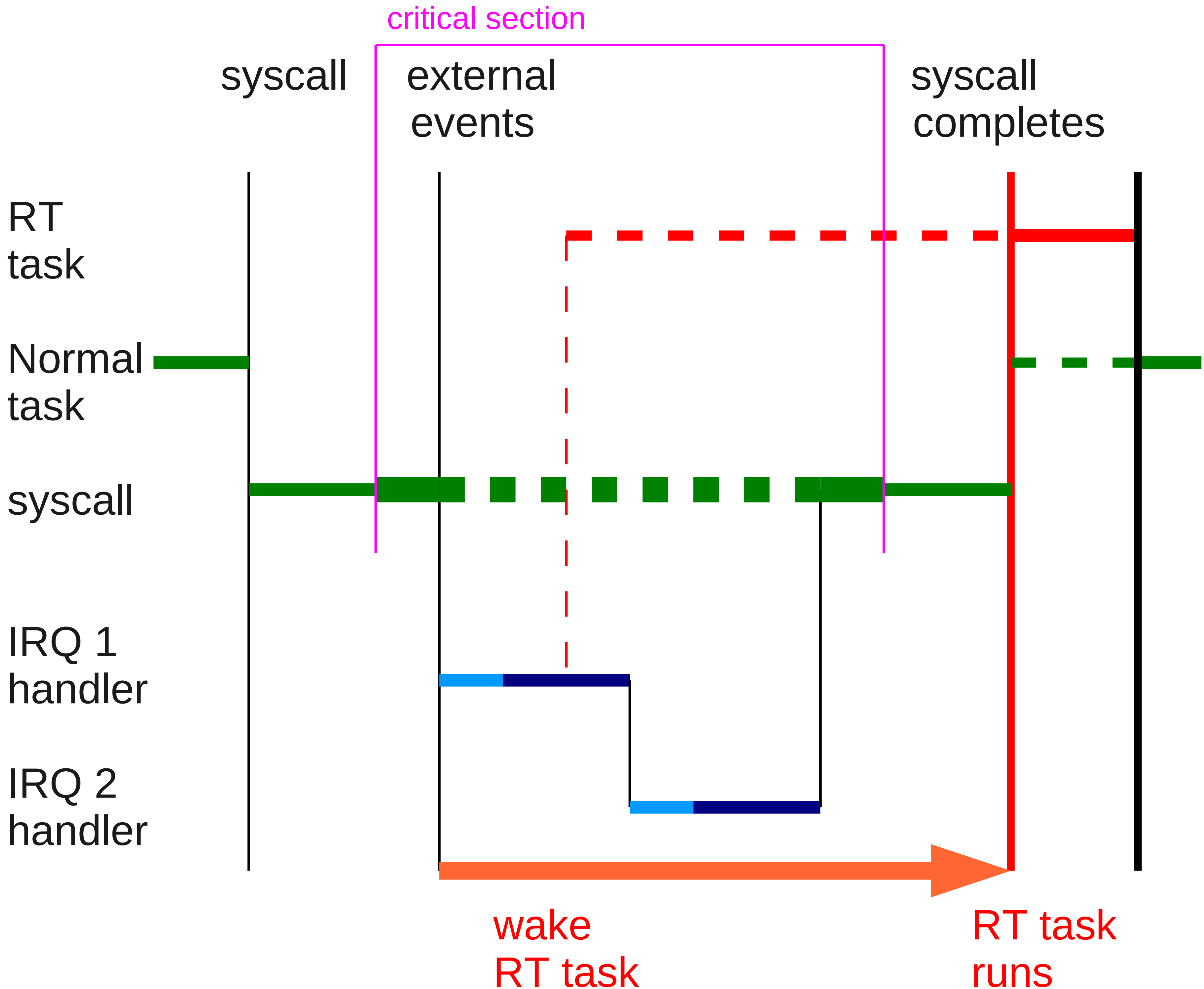

syscall
external event
syscall completes
RT task
Normal task
syscall
IRQ handler
wake RT task
RT task runs

Non-Preemptible Kernel

Next slide illustrates non-preemptible kernel.

Adding some complexity:

- 2 external events occur
- lock (critical section) during syscall


critical section
syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1 handler
IRQ 2 handler
wake RT task
RT task runs

Preemptible Kernel

Mainline 2.6 and 3.0 kernel

CONFIG_PREEMPT_NONE

- No forced kernel preemption

CONFIG_PREEMPT_VOLUNTARY

- Explicit preemption points in kernel

CONFIG_PREEMPT

- All kernel code (not in critical section) preemptible

Preemptible Kernel

RT_PREEMPT patch renames config option:

- Vanilla 2.6 kernel
  - CONFIG_PREEMPT
- RT_PREEMPT 2.6 kernel
  - CONFIG_PREEMPT_DESKTOP

Preemptible Kernel

RT_PREEMPT patch renames config option:

- Vanilla 3.0 kernel
  - CONFIG_PREEMPT
- RT_PREEMPT 3.0 kernel
  - CONFIG_PREEMPT_LL

Preemptible Kernel

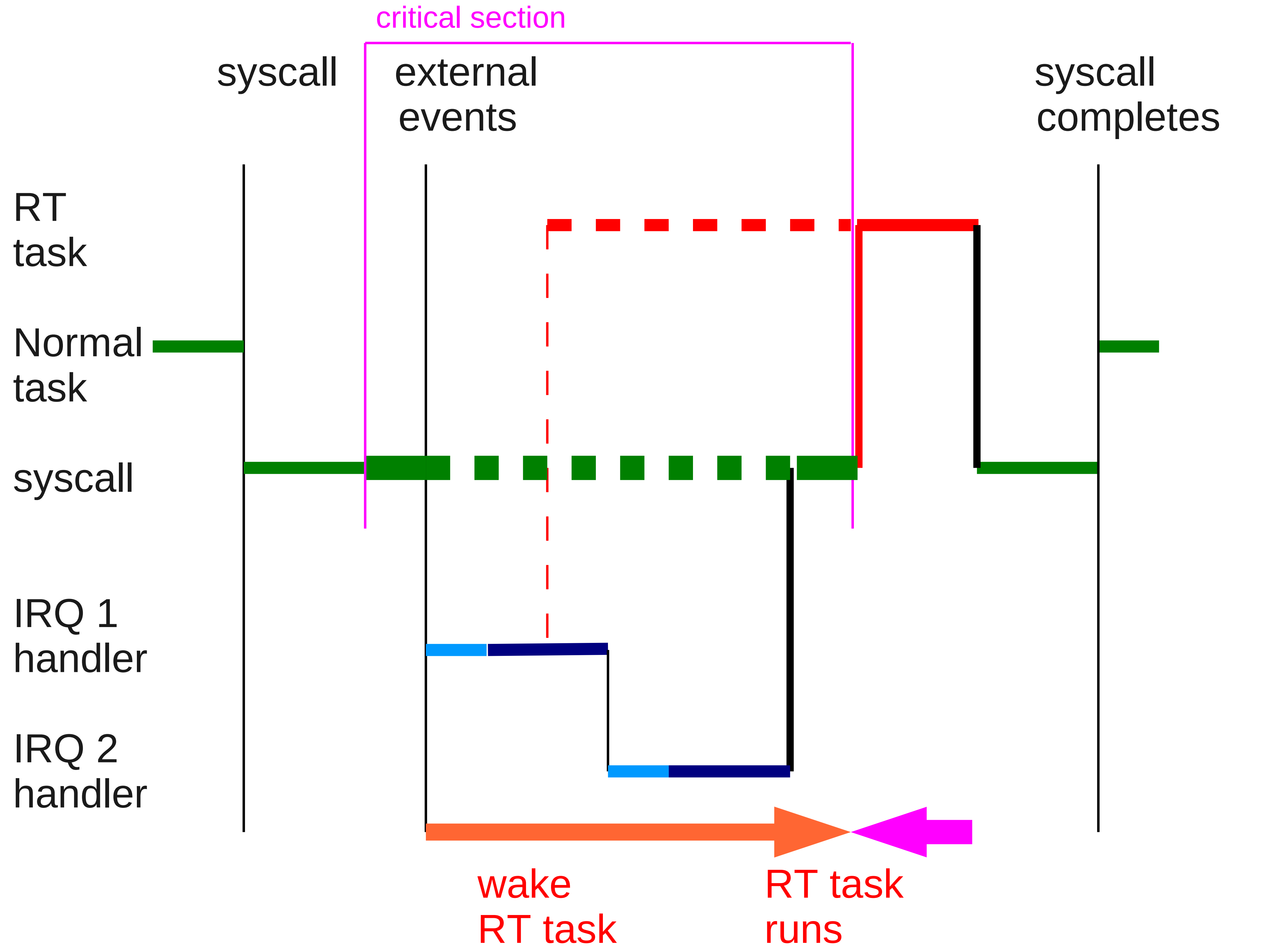
Mainline 2.6 and 3.0 kernel

CONFIG_PREEMPT   “fully preemptible”

- except when preemption is explicitly disabled
- except when interrupts are explicitly disabled
- except when a lock is held
  (“in a critical section”)

Preemptible Kernel

Next slide illustrates preemptible kernel.


critical section
syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1 handler
IRQ 2 handler
wake RT task
RT task runs

Score

- added 0 schedule with context switch

+ shorter maximum wakeup latency

Preemptible Kernel

RT_PREEMPT 2.6 kernel

CONFIG_PREEMPT_RT

“fully preemptible”

- except when preemption is explicitly disabled
- except when interrupts are explicitly disabled
- except when a raw spinlock is held

Preemptible Kernel

RT_PREEMPT 3.0 kernel

CONFIG_PREEMPT_RT_FULL

“fully preemptible”

- except when preemption is explicitly disabled
- except when interrupts are explicitly disabled
- except when a raw spinlock is held

Preemptible Kernel

CONFIG_PREEMPT_RT
CONFIG_PREEMPT_RT_FULL

Most kernel locks are converted to preemptible priority inheritance mutex.

Some kernel locks are converted to non-preemptible raw spinlock.

Score: Spinlock vs Mutex Acquire, Release

Spinlock

- shorter code path

Mutex

- longer code path

Score: Spinlock vs Mutex Contention

Spinlock

    spin until lock is available

Mutex

    if owner is executing
        spin until lock is available
    else
        sleep, wake when lock is available
        (2 schedules)

Score: Spinlock vs Mutex Result with contention

Spinlock is cheaper than mutex if:

spinlock lock path + spinlock unlock path
+ lock contention time <

mutex lock path + mutex unlock path
+ 2 schedules

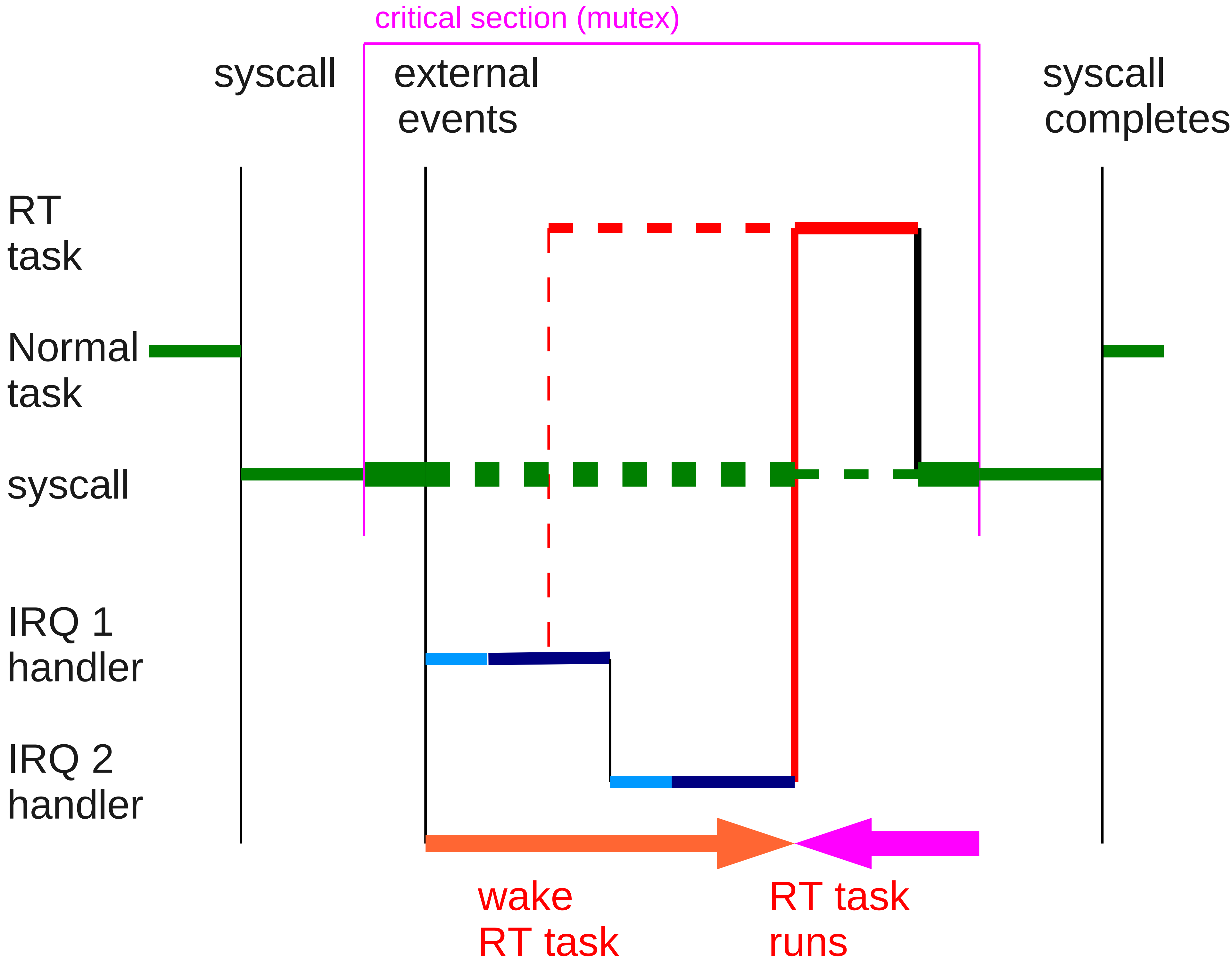
Score: Spinlock vs Mutex
Result with contention

Previous slide ignores:

- extra schedules due to priority inheritance
- cache effects

Preemptible Kernel

Next slide illustrates preemptible kernel with spinlocks converted to mutexes.


critical section (mutex)
syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1 handler
IRQ 2 handler
wake RT task
RT task runs

Score

- added 0 schedule with context switch

+ shorter maximum wakeup latency

Priority Inheritance Mutex

- May result in more schedule events.
- Avoids priority inversion.
- Typically larger execution cost than spinlock.
- Reader-Writer lock limited to one concurrent reader to minimize PI complexity.
  - Limits scalability of multiple readers.

Threaded Interrupt Handler

Overview Of
Interrupt handling algorithm

- Save context
- Handle “highest priority” interrupt
  - Interrupt handler executes in interrupt mode
  - irq_exit() may process softirq or wake softirqd
- Iterate over active interrupts (arch dependent)
- Schedule
- Restore context
  - returning either to previous process or
    to newly scheduled process

Overview Of Threaded Interrupt handling algorithm

- Save context
- Handle “highest priority” interrupt
  Wake Interrupt handler thread.
  irq_exit() may process softirq or wake softirqd
- Iterate over active interrupts (arch dependent)
- Schedule
- Restore context
  returning either to previous process or
  to newly scheduled process

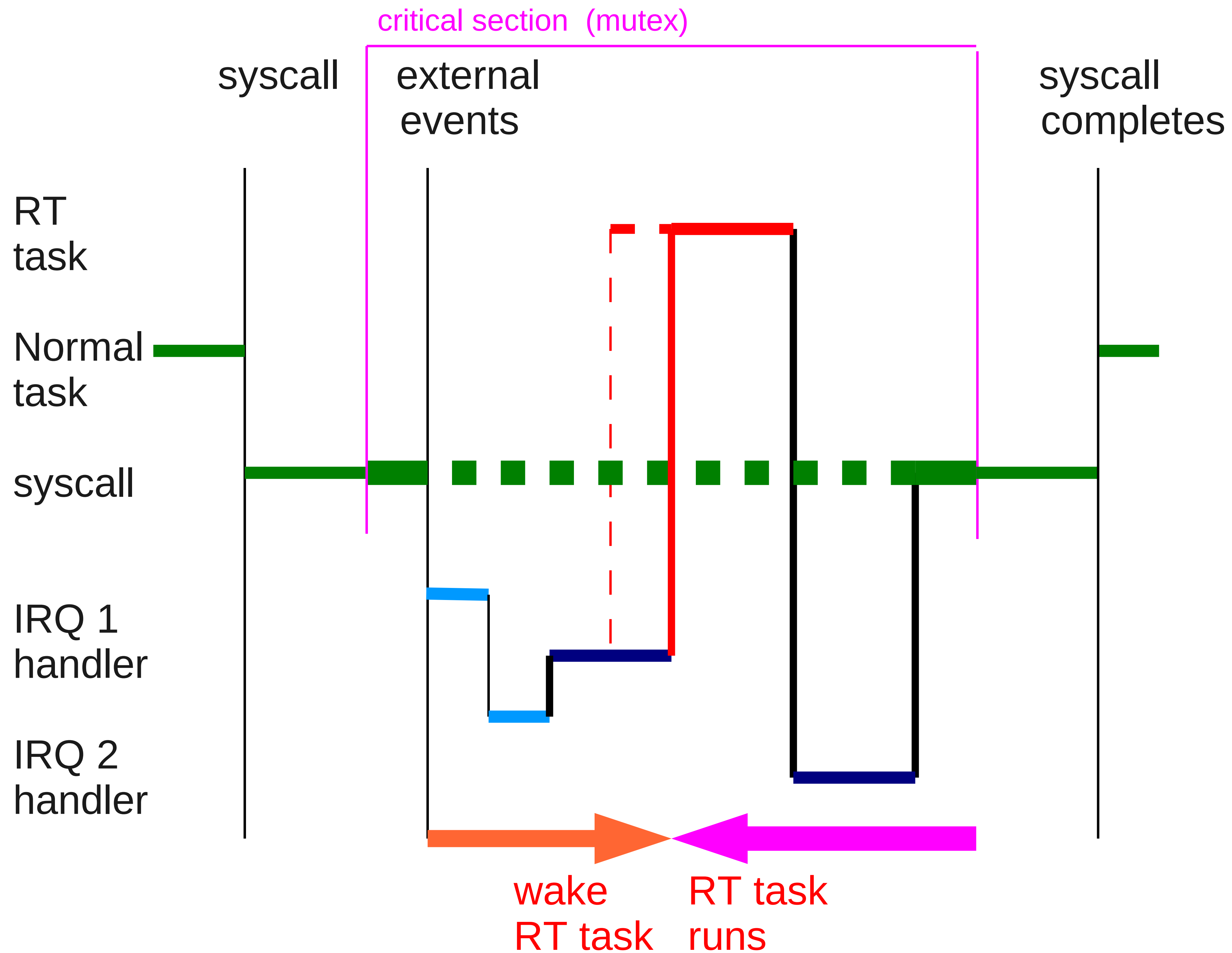
Interrupt handler thread executes when scheduled.

Threaded Interrupt Handler

RT_PREEMPT patchset converts almost all drivers to threaded model. (Timer handler executes in interrupt context.)

Preemptible Kernel

Next slide illustrates preemptible kernel with interrupt threads.



critical section (mutex)
syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1 handler
IRQ 2 handler
wake RT task
RT task runs

Score

- added 2 schedule with context switch

+ shorter maximum wakeup latency

Other Interrupt Overhead

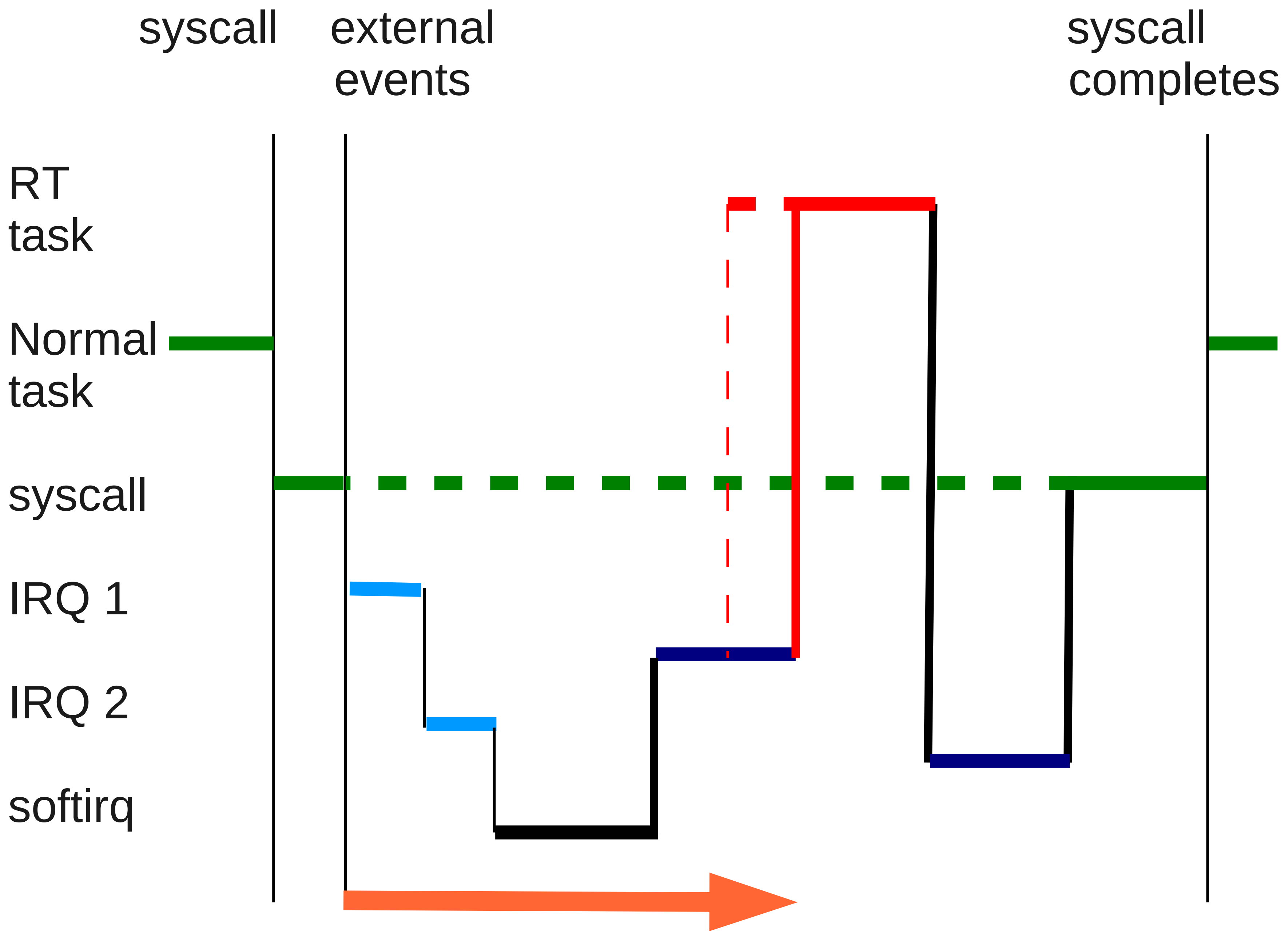
Other Interrupt Overhead

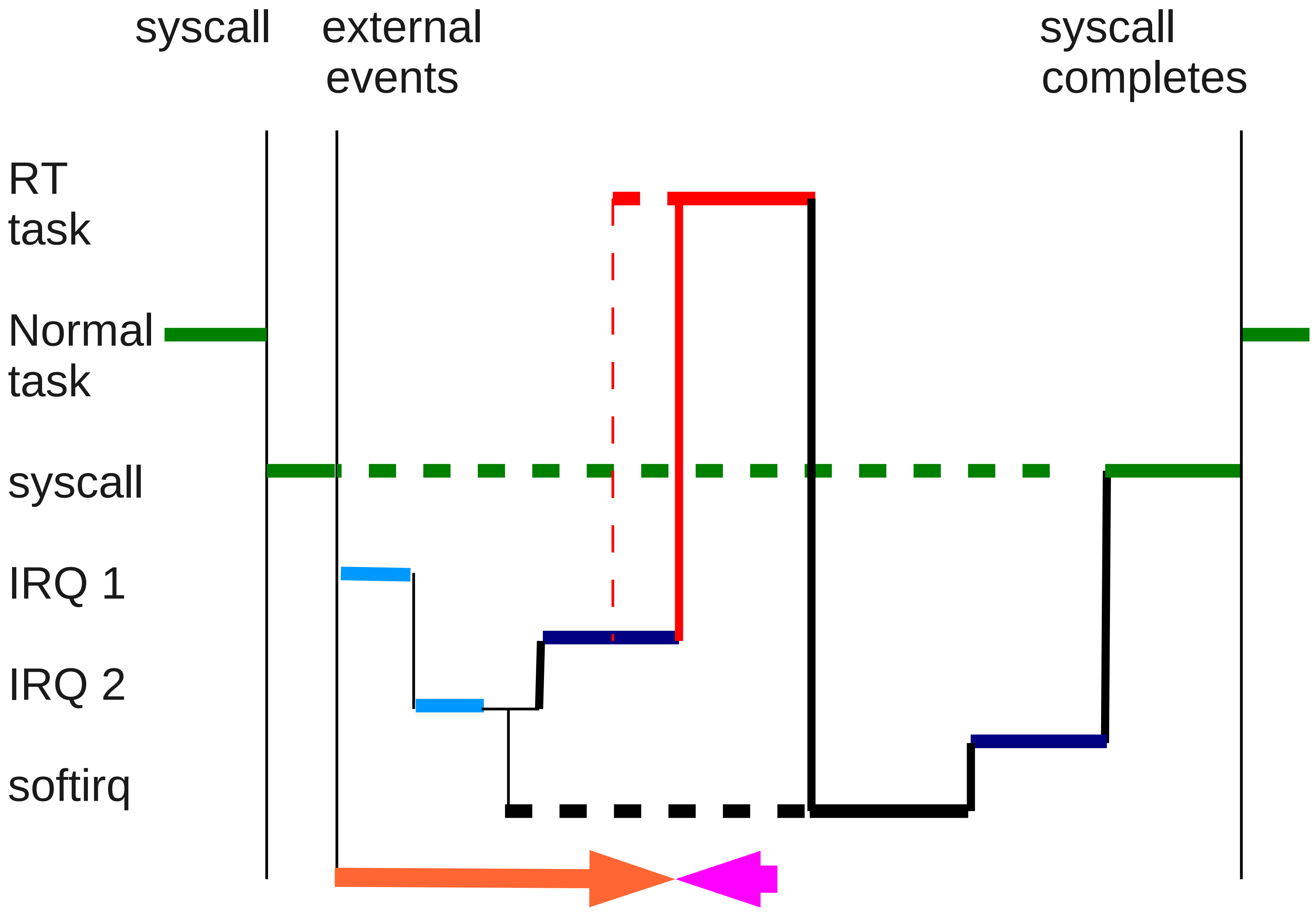
CONFIG_PREEMPT_RT and CONFIG_PREEMPT_RT_FULL changes:

- irq_exit() may process softirq or wake ksoftirqd thread

to:

- irq_exit() may wake ksoftirqd thread


syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1
IRQ 2
softirq



syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1
IRQ 2
softirq

Score

- added 1 schedule with context switch

+ shorter maximum wakeup latency

Other Interrupt Overhead

raise_softirq(), (softirq trigger) typically called from:

- irq context (timer softirq)
- interrupt thread

but can be called from anywhere in the kernel.

Previous slides show trigger from irq context.

Other Locking Overhead

CONFIG_TREE_PREEMPT_RCU

- Evolving in the early 3.0 RT patches...
- Not analyzed in this presentation.

Other Locking Overhead

local_lock

get_local_var() uses migrate_disable() instead of preempt_disable().

Evolving in the early 3.0 RT patches...

Not analyzed in this presentation.

Real Life

Real systems are much more complicated than the previous diagrams.

Other scenarios can generate different performance improvements or penalties.

Real Life

The flow of control diagrams are not an exhaustive analysis of why and how much PREEMPT_RT_FULL impacts performance metrics.

But instead provide scenarios illustrating the impacts.

Full Analysis of Overhead Would Include:

- Lock implementation overhead
- Reduced concurrency for Reader-Writer locks
- Increased lock contention, due to sleeping while holding lock
- Additional schedules
- Additional context switches

Recap: Enabling real-time Linux

- preemptible kernel
- locking
- interrupt handlers
- threaded softirq

Impact of Real-Time Features

+ Variance of real-time task latency decreased

+ Maximum real-time task latency decreased

- Average real-time task latency may be increased

- Throughput decreased

Questions?

How to get a copy of the slides

frank.rowand@am.sony.com