Real-Time Preemption Patch-Set

Manas Saksena

TimeSys

Real-Time Preemption Patch-Set

Development done by Ingo Molnar

Patches Available at:

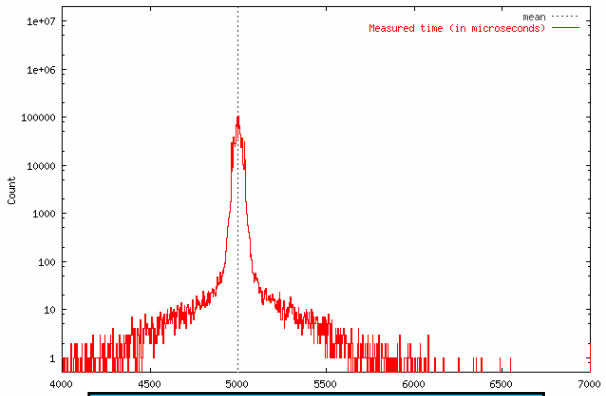
- http://people.redhat.com/mingo/realtime-preempt/
- LKML archives

Historical Perspective:

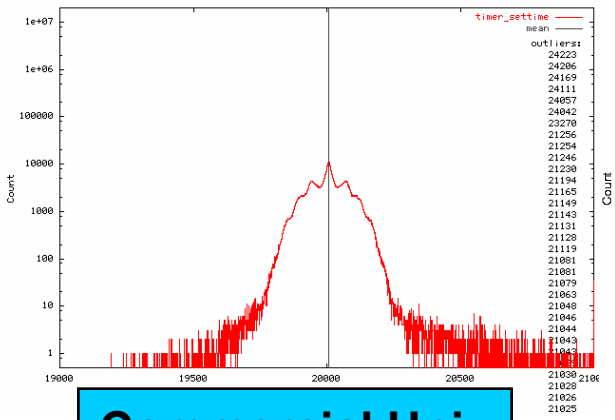
- TimeSys 2.4 Kernels implemented most of these features
- Scott Wood developed patches (posted to LKML) for IRQ & SoftIRQ Threading that were part of CELF 1.0 specification

TimeSys®

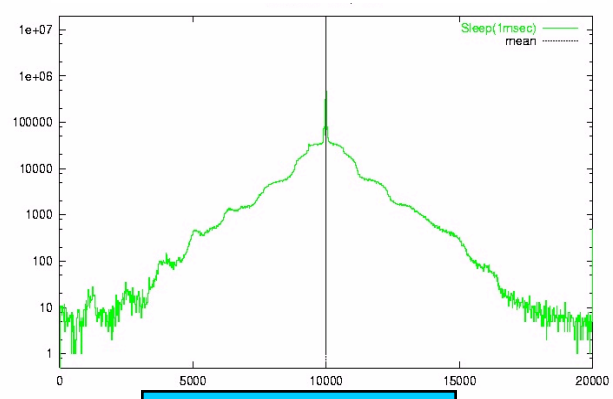
True RTOS Performance in Linux?



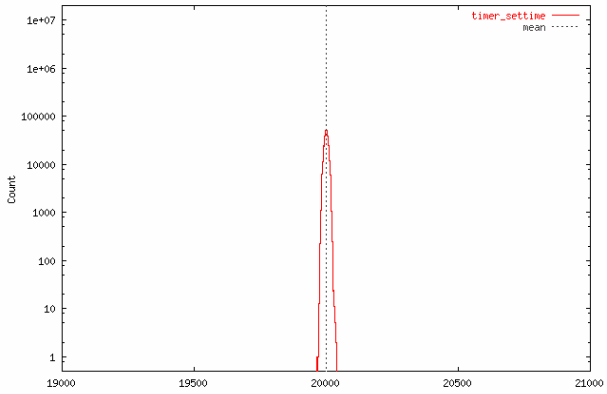
2.4 Preempt Linux



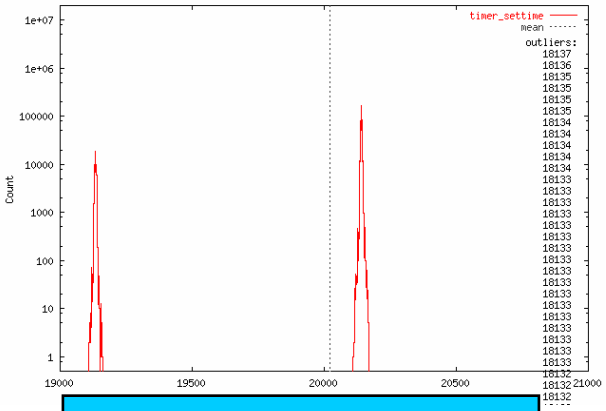
Commercial Unix



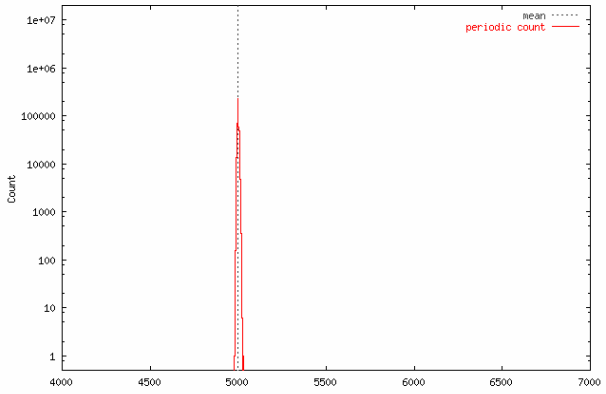
Desktop OS



Real-Time Unix



Widely-Used RTOS



TimeSys 2.4 Linux

TimeSys®

Real-Time Preemption PatchSet

Goal:

- Make Fixed priority preemptive scheduling (i.e., POSIX SCHED_FIFO and SCHED_RR classes) as close as possible to their ideal behavior

Tactics:

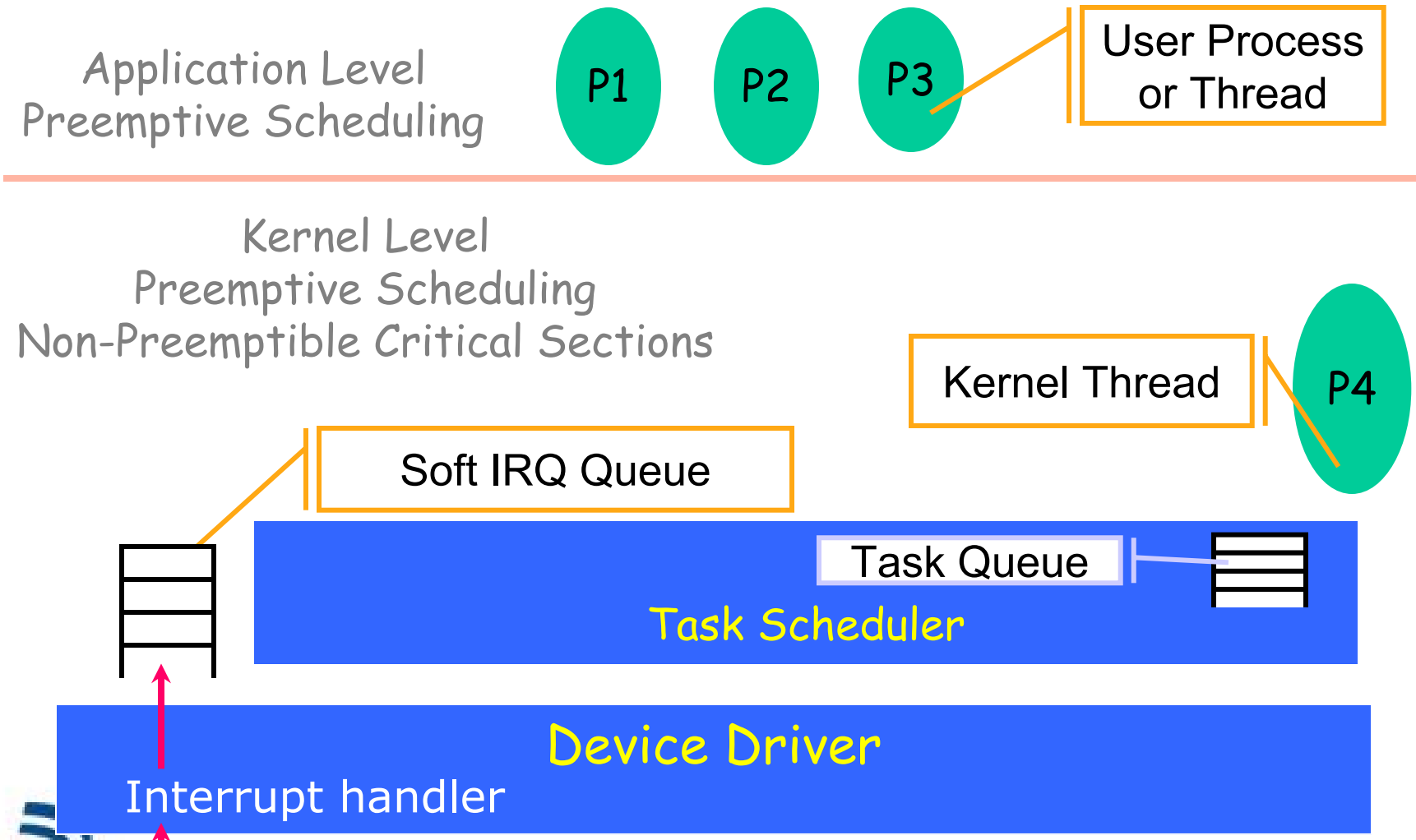
- Execute all activities in “schedulable/thread” context
- Make the system preemptible as much as possible

Other Goals:

- No impact on users not interested in real-time
- Support for degrees of real-time behaviors (Latency vs Throughput tradeoff)

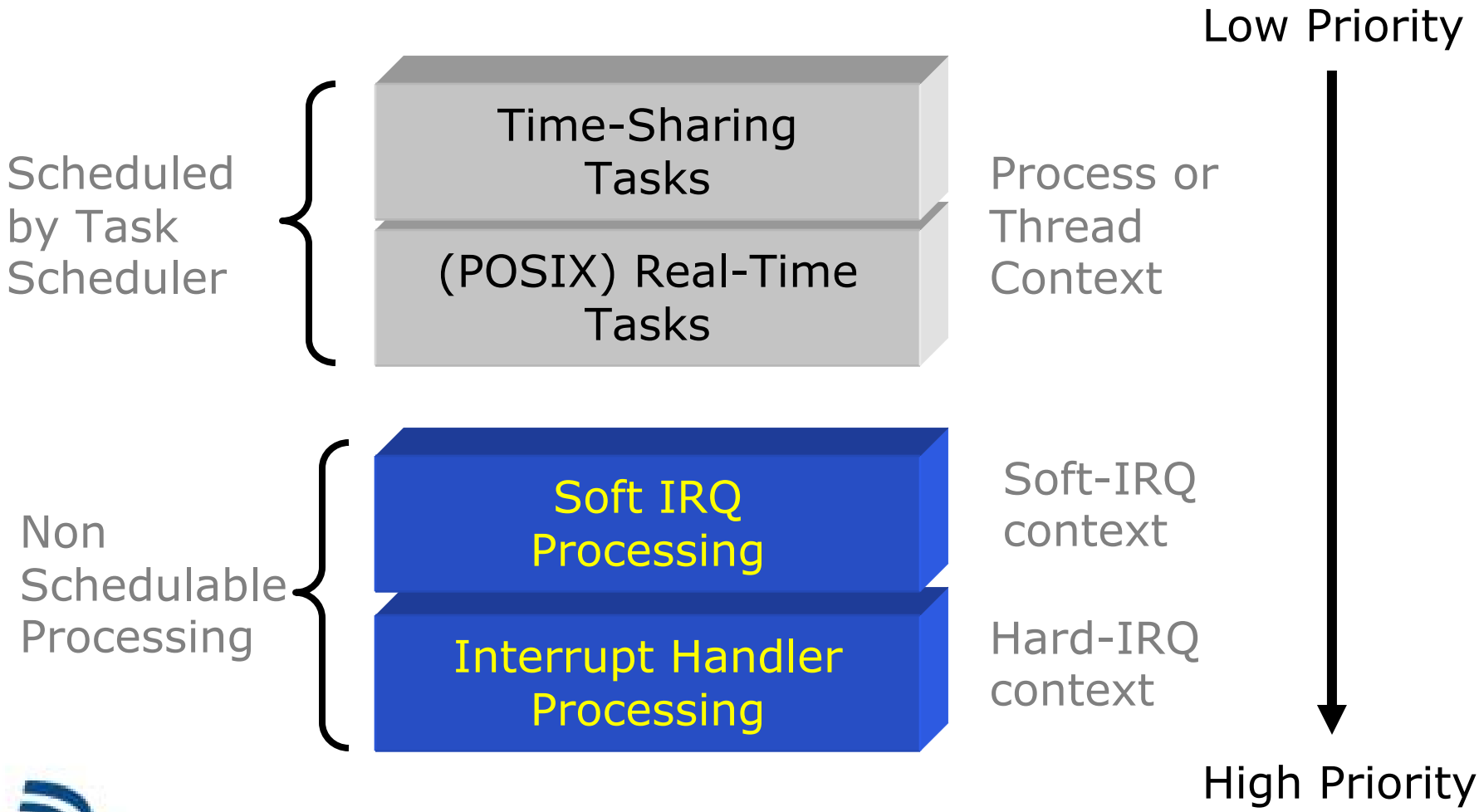
TimeSys®

Linux Tasking & Scheduling Architecture (2.6 kernel)



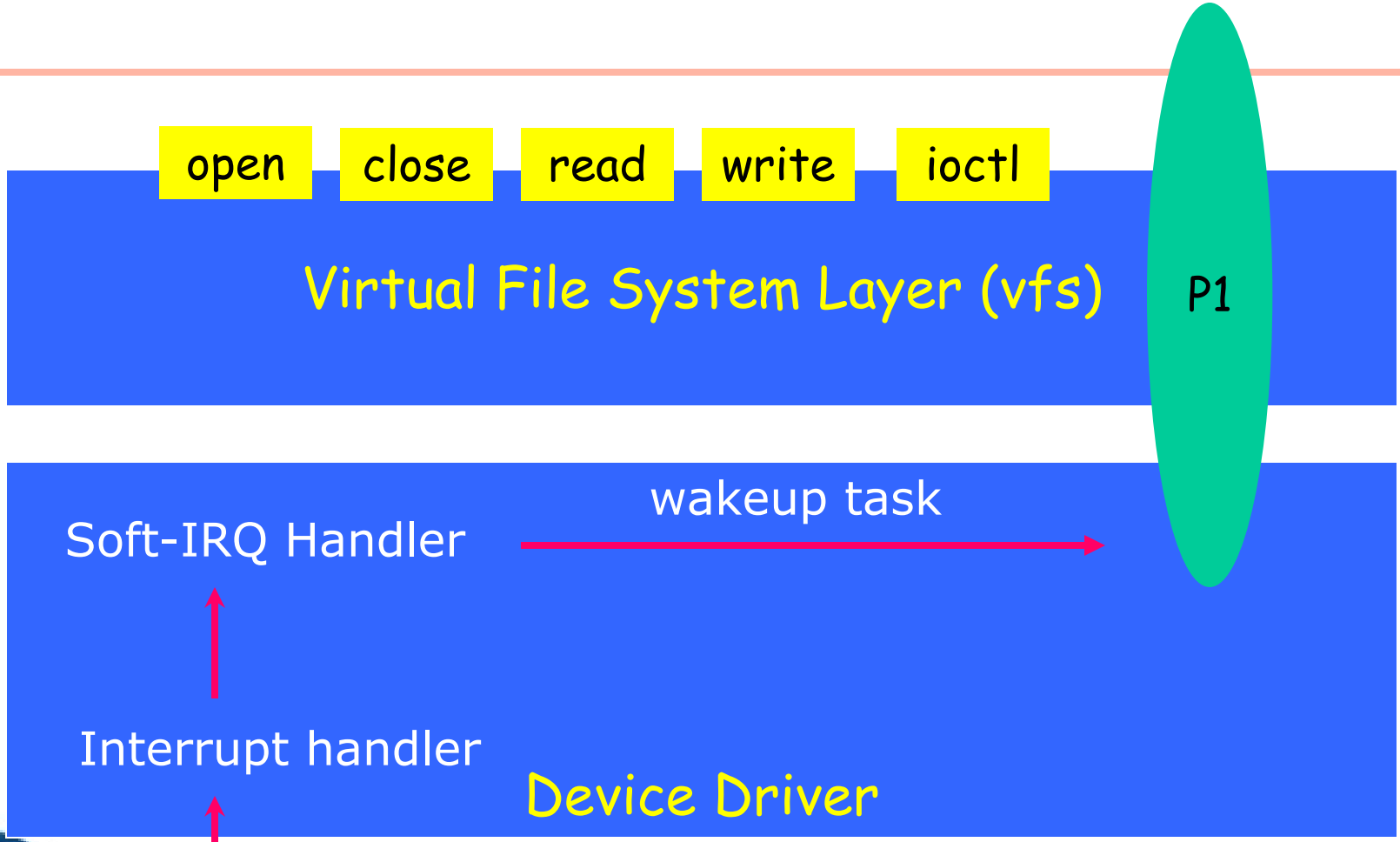
TimeSys®

Linux Scheduling Architecture



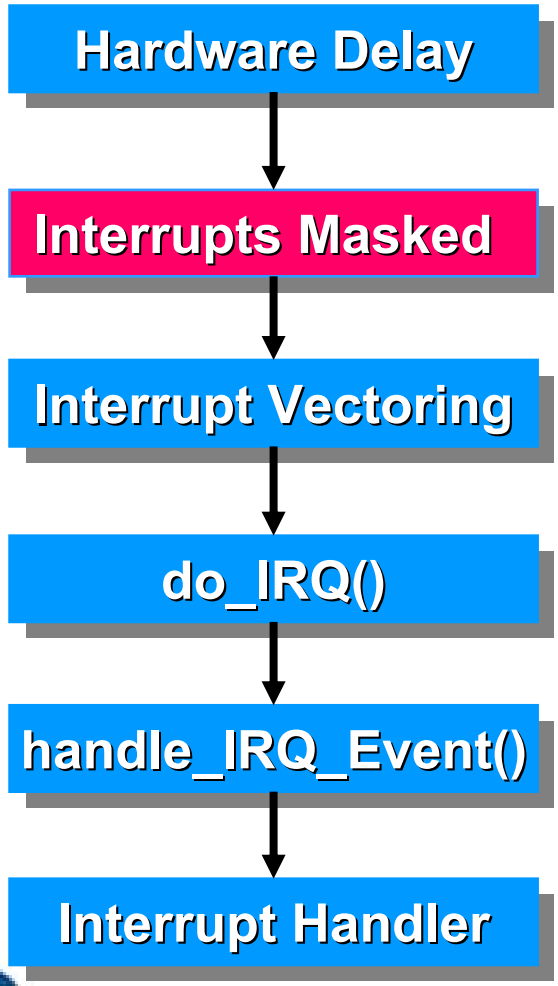
TimeSys®

Wakeup Latency



TimeSys®

Interrupt Latency



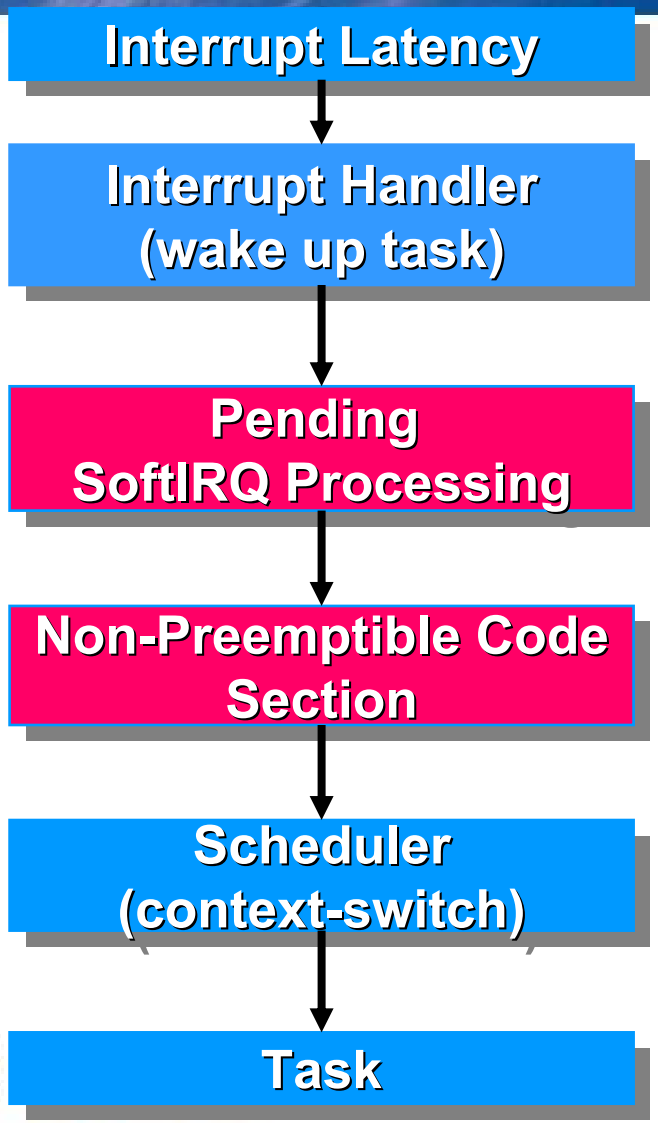
priority inversion

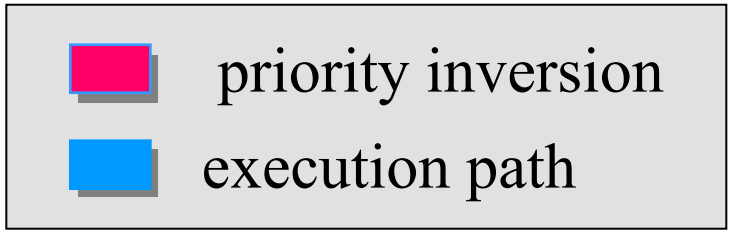
execution path

Nested Interrupts can cause additional delays here

TimeSys®

WakeUp Latency





OS defined higher priority work – may not match with application's needs

Non-preemptible kernel or Critical Sections

TimeSys®

Summary of Issues

Non-Prioritized Activities

- Interrupt Handling
- SoftIRQ Handling

Non-Preemptible Code Sections

- All Critical Sections (protected by spin locks) in the kernel
- Special Big Kernel Lock protected critical sections

Prioritized/Threaded Interrupt Handling

```
+config PREEMPT_HARDIRQS
+ bool "Thread Hardirqs"
+ default n
+#         depends on PREEMPT
+ help
+         This option reduces the latency of the kernel by 'threading'
+         hardirqs. This means that all (or selected) hardirqs will run
+         in their own kernel thread context. While this helps latency,
+         this feature can also reduce performance.
+
+         The threading of hardirqs can also be controlled via the
+         /proc/sys/kernel/hardirq_preemption runtime flag and the
+         hardirq-preempt=0/1 boot-time option. Per-irq threading can
+         be enabled/disable via the /proc/irq/<IRQ>/<handler>/threaded
+         runtime flags.
+
+   Say N if you are unsure.
```

Threaded SoftIRQ Handling

```
+config PREEMPT_SOFTIRQS
+ bool "Thread Softirqs"
+ default n
+#         depends on PREEMPT
+ help
+         This option reduces the latency of the kernel by 'threading'
+         soft interrupts. This means that all softirqs will execute
+         in softirqd's context. While this helps latency, it can also
+         reduce performance.
+
+         The threading of softirqs can also be controlled via
+         /proc/sys/kernel/softirq_preemption runtime flag and the
+         sofirq-preempt=0/1 boot-time option.
+
+   Say N if you are unsure.
```

TimeSys®

Summary of Issues

Non-Prioritized Activities

- Interrupt Handling
- SoftIRQ Handling

Non-Preemptible Code Sections

- **All Critical Sections (protected by spin locks) in the kernel**
- **Special Big Kernel Lock protected critical sections**

Kernel Preemptibility Options

No Preemption

- Non Preemptible Kernel

Voluntary Preemption

- Non Preemptible Kernel; Voluntary Preemption

Preemptible Kernel

- Preemptible Kernel, but non preemptible critical sections

Real-Time Preemptible Kernel

- Fully Preemptible Kernel

No Preemption

```
+config PREEMPT_NONE
+	bool "No Forced Preemption (Server)"
+	help
+	  This is the traditional Linux preemption model geared towards
+	  throughput. It will still provide good latencies most of the
+	  time but there are no guarantees and occasional long delays
+	  are possible.
+
+	  Select this option if you are building a kernel for a server or
+	  scientific/computation system, or if you want to maximize the
+	  raw processing power of the kernel, irrespective of scheduling
+	  latencies.
```

TimeSys®

Voluntary Preemption

```
+config PREEMPT_VOLUNTARY
+bool "Voluntary Kernel Preemption (Desktop)"
+help
+  This option reduces the latency of the kernel by adding more
+  "explicit preemption points" to the kernel code. These new
+  preemption points have been selected to minimize the
  maximum
+  latency of rescheduling, providing faster application reactions,
+  at the cost of slighly lower throughput.
+
+  This allows reaction to interactive events by allowing a
+  low priority process to voluntarily preempt itself even if it
+  is in kernel mode executing a system call. This allows
+  applications to run more 'smoothly' even when the system is
+  under load.
```

Voluntary Preemption

Basic Idea

- Introduce preemption points on long kernel paths
- Useful for getting low latencies when not using preemptible kernels

Voluntary Preempt in RT Patchset

- Reuse existing (but inactive) scheduling points in the kernel
- Introduce additional preemption points through instrumentation
  - Use lock-breaking to break long critical sections

Preempt Desktop

```
+config PREEMPT_DESKTOP
+ bool "Preemptible Kernel (Low-Latency Desktop)"
+ help
+   This option reduces the latency of the kernel by making
+   all kernel code that is not executing in a critical section
+   preemptible.  This allows reaction to interactive events by
+   permitting a low priority process to be preempted involuntarily
+   even if it is in kernel mode executing a system call and would
+   otherwise not about to reach a preemption point.  This allows
+   applications to run more 'smoothly' even when the system is
+   under load, at the cost of slighly lower throughput and a
+   slight runtime overhead to kernel code.
+
+   (According to profiles, when this mode is selected then even
+   during kernel-intense workloads the system is in an immediately
+   preemptible state more than 50% of the time.)
```

Real-Time Preemption

```
+config PREEMPT_RT
+ bool "Complete Preemption (Real-Time)"
+ select PREEMPT_SOFTIRQS
+ select PREEMPT_HARDIRQS
+ help
+   This option further reduces the scheduling latency of the
+   kernel by replacing almost every spinlock used by the kernel
+   with preemptible mutexes and thus making all but the most
+   critical kernel code involuntarily preemptible. The remaining
+   handful of lowlevel non-preemptible codepaths are short and
+   have a deterministic latency of a couple of tens of
+   microseconds (depending the the hardware).  This also allows
+   applications to run more 'smoothly' even when the system is
+   under load, at the cost of lower throughput and runtime
+   overhead to kernel code.
```

TimeSys®

Preemptible Kernels: Two Approaches to Protecting Critical Sections

PREEMPT-LOCK:

- Disable preemption during critical sections
- PREEMPT_DESKTOP does this
- Kernel is preemptible everywhere except when inside a critical section
- Optionally enable IRQ/SoftIRQ Threading
- Optionally enable Voluntary Preemption

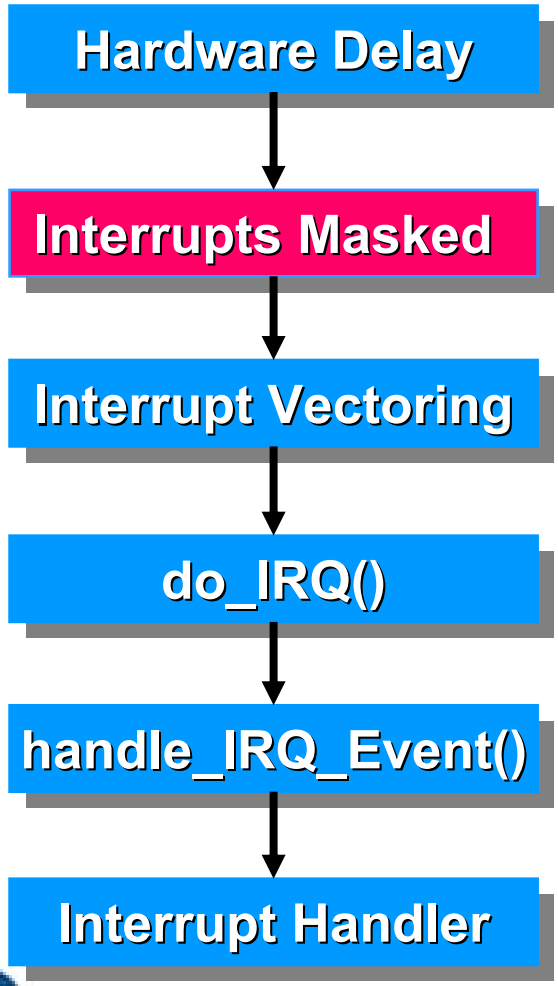
MUTEX-LOCK:

- PREMPT_RT does this
- Kernel is preemptible inside (most) critical sections
  - Still need some small non-preemptible critical sections
- Needs IRQ/SoftIRQ Threading

BKL Preemption

```
+config PREEMPT_BKL
+ bool "Preempt The Big Kernel Lock"
+ depends on PREEMPT || SMP
+ default y
+ help
+    This option reduces the latency of the kernel
       by making the big kernel lock preemptible.
+
+ Say Y here if you are building a kernel for a desktop
  system.
+ Say N if you are unsure.
```

Interrupt Latency with RT PREEMPT



priority inversion

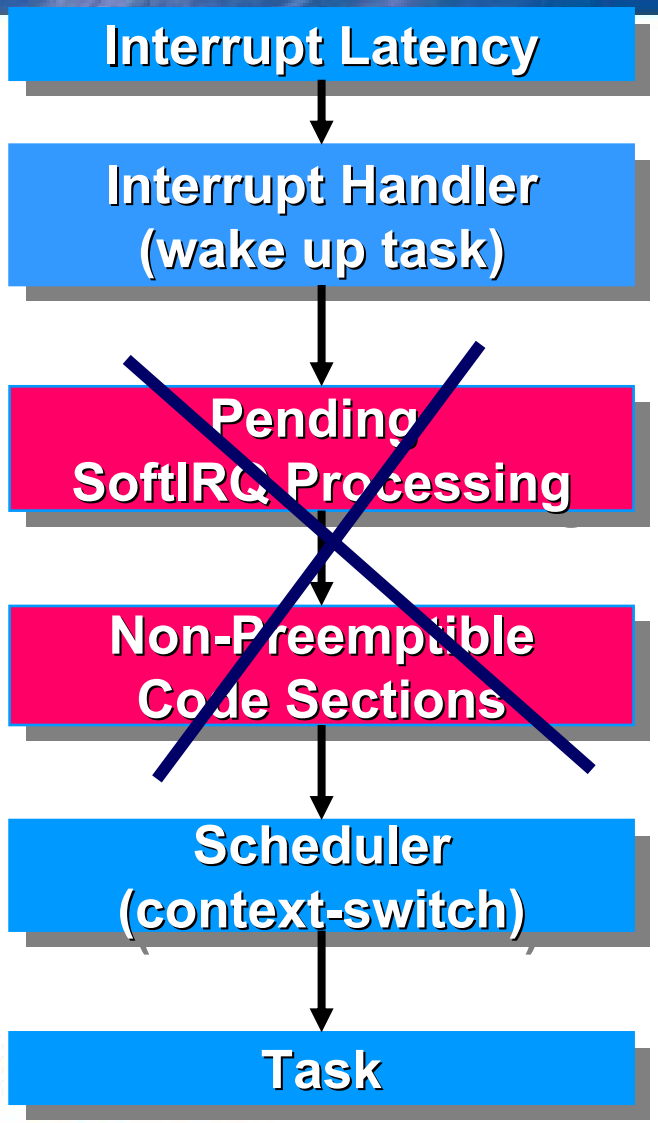
execution path

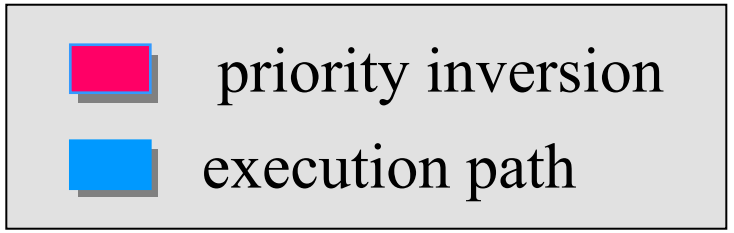
Reduced Interrupt Masking Time

Nested Interrupts can cause additional delays here

TimeSys®

Wakeup Latency with RT PREEMPT





OS defined higher priority work – may not match with application's needs

Non-preemptible kernel or Critica Sections

TimeSys®