The Linux Real Time patch

- Klaas van Gend
- FAE Europe, MontaVista Software

- T-DOSE conference,
- Eindhoven, December 3, 2006

Shameless plug

montavista

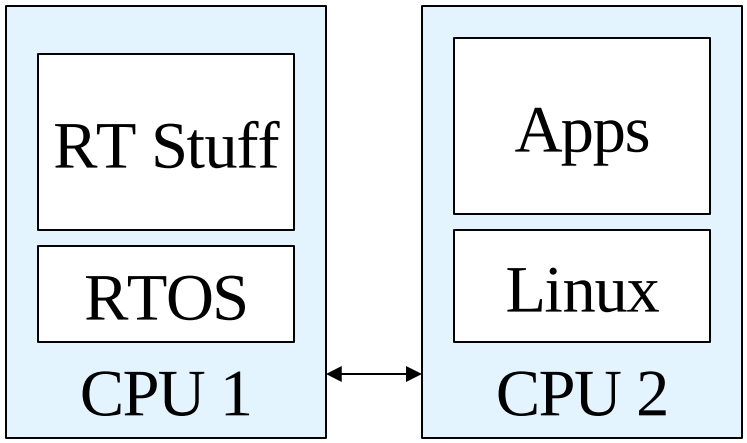
NLUUG

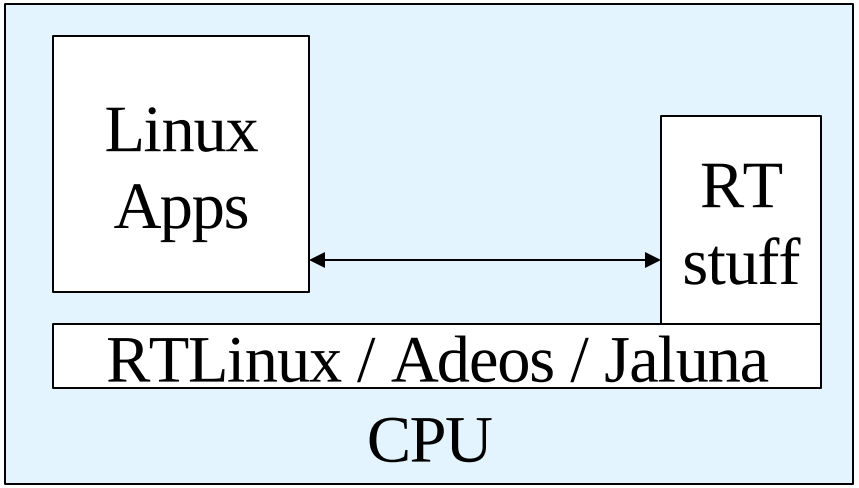
NLUUG – Open Systems, Open Standards.

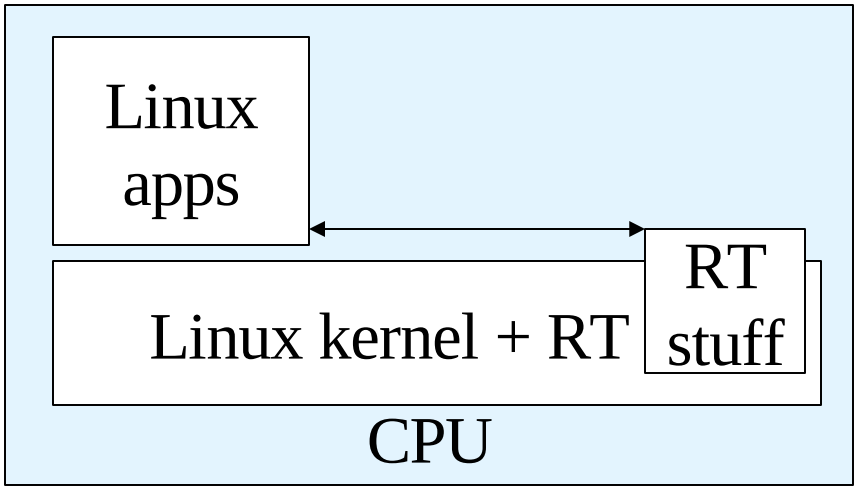
Spring conference on **virtualization** : May 10, 2007.

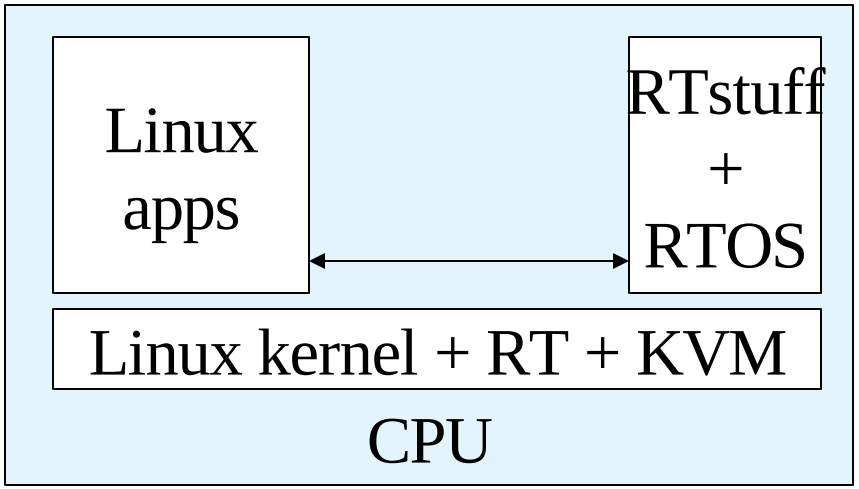
Virtualization and Real Time?

montavista









Real Time patch BOF on OLS

- Friday, July 21, 2006
- 13:00 – 13:45
- Room D

- Steven Rostedt
- Klaas van Gend

Rights to Copy

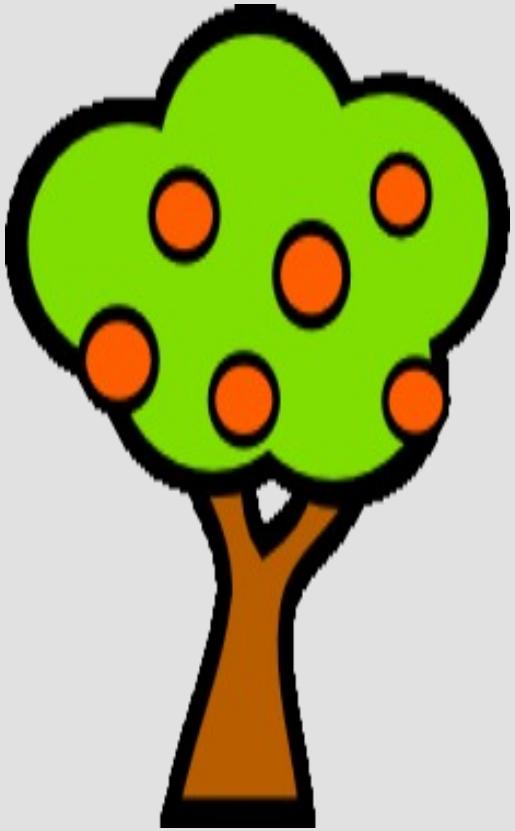
The RT patch

- Ingo Molnar's Real-time Patch
  - Maintainer: Ingo Molnar
  - Several developers: Thomas Gleixner and others
  - Download at: http://people.redhat.com/mingo/realtime-preempt/

What is UNIX?

- UNIX is by design fair:
- Share resources:
  - Memory
  - Disk space
  - cpu

The Car example



What is RT?

- RT is about determinism

  NOT performance!

- RT gives you a *guaranteed* maximum time of what can happen

- Sometimes this might actually slow down the system slightly

Repeat after me:

RT is NOT about performance

What's in the RT patch?

- Interrupts as threads (both Hard and Soft)
- Sleeping spinlocks !
- Priority Inheritance of the Sleeping Spinlocks
- high-res timers

Hard interrupts as Threads

- The “top half” is now a kernel thread
  - Runs in a loop
  - Calls *all* the ISR for the IRQ
- Processes can run at a higher priority than an ISR
- Ctrl-C ???
  - Klaas will talk about that later

?

PID	TID	CLS	RTPRIO	NI	PRI	PSR	%CPU	STAT	WCHAN	COMMAND
1	1	TS	-	0	23	0	0.6	S	select	init
2	2	FF	99	-	139	0	0.0	S	migration_thre	migration/0
3	3	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-high/0
4	4	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-timer/0
5	5	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-net-tx/0
6	6	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-net-rx/0
7	7	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-block/0
8	8	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-tasklet
9	9	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-hrtreal
10	10	FF	1	-	41	0	0.0	S	ksoftirqd	softirq-hrtmono
11	11	FF	99	-	139	0	0.0	S	watchdog	watchdog/0
12	12	TS	-	-10	34	0	0.0	S<	desched_thread	desched/0
24	24	FF	1	-	41	0	0.0	S<	worker_thread	events/0
26	26	TS	-	-5	29	0	0.0	S<	worker_thread	khelper
27	27	TS	-	-5	28	1	0.0	S<	worker_thread	kthread
30	30	TS	-	-5	29	0	0.0	S<	worker_thread	kblockd/0
32	32	TS	-	-5	29	1	0.0	S<	worker_thread	kacpid
33	33	FF	49	-	89	0	0.0	S<	irqd	IRQ 9
150	150	TS	-	0	23	0	0.0	S	pdflush	pdflush
151	151	TS	-	0	24	0	0.0	S	pdflush	pdflush
153	153	TS	-	-5	28	0	0.0	S<	worker_thread	aio/0
152	152	TS	-	0	22	1	0.0	S	kswapd	kswapd0
741	741	TS	-	-5	29	0	0.0	S<	serio_thread	kseriod
746	746	FF	48	-	88	1	0.0	S<	irqd	IRQ 12
774	774	FF	47	-	87	0	0.1	S<	irqd	IRQ 14
789	789	FF	45	-	85	0	0.0	S<	irqd	IRQ 1
793	793	TS	-	0	24	1	0.0	S	kjournald	kjournald
894	894	TS	-	-4	26	0	2.6	S<s	select	udevd

Soft IRQs

- Are separated
- Every softIRQ has its own thread
- hrtimer softirq has dynamic priority

Sleeping spinlocks

CONFIG_PREEMPT

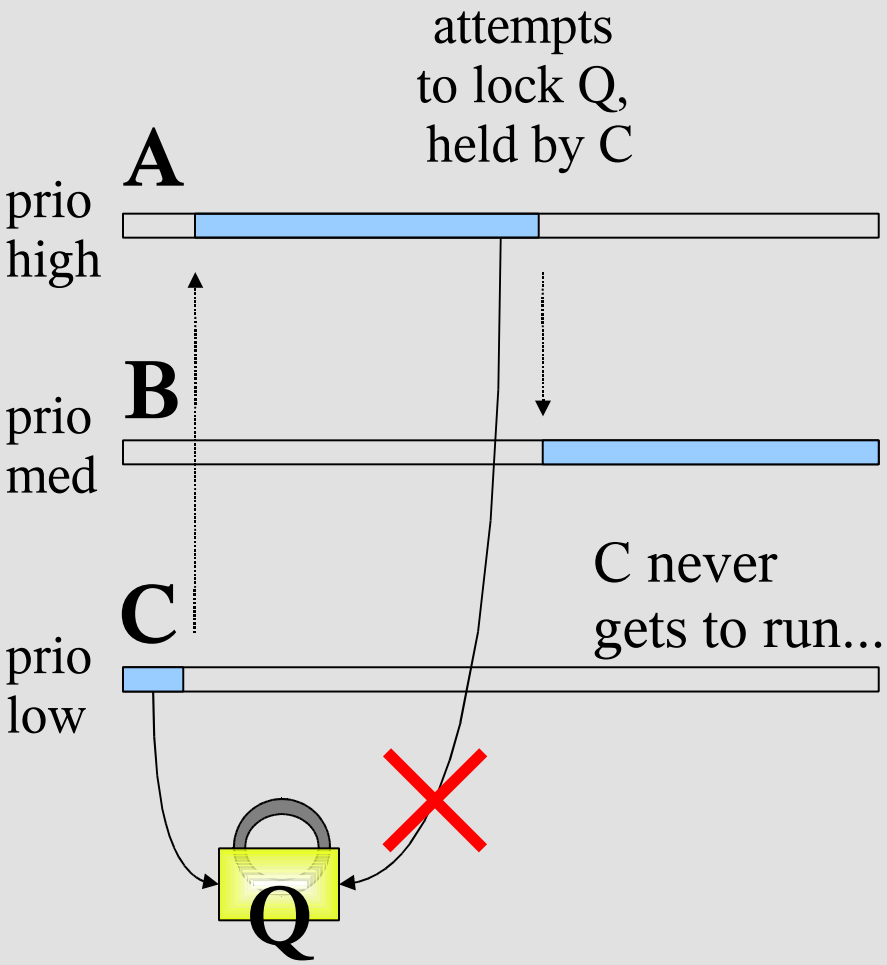
- spinlock disables preemption
- global blocking
- IRQs in interrupt context

CONFIG_PREEMPT_RT

- spinlocks are mutexes
- localized critical sections
- must have IRQs as threads

Priority Inheritance

- Fact:
  priority inversion happens
- *Even if Linus says it doesn't.*
- *Unbounded* priority inversion is preventable



BOF Part II:

MontaVista customers and RT

why all customers make the same mistakes

Klaas van Gend

FAE Europe

July 20, 2006

Why this presentation?

- Klaas van Gend
  - Works for MontaVista Linux Software
  - Has been shipping RT on 2.6.10 since August 2005
  - Several customers use RT
- They all made the same (type of) mistakes
- They all had the same confusions
  - Even if Klaas told them beforehand
- But we can learn from their confusion!

First mistake: do you need RT?

"I need real time because my system needs to be fast"

"I want to have the best performance Linux can do"

NO!

REAL TIME DOES NOT MEAN HIGHEST PERFORMANCE

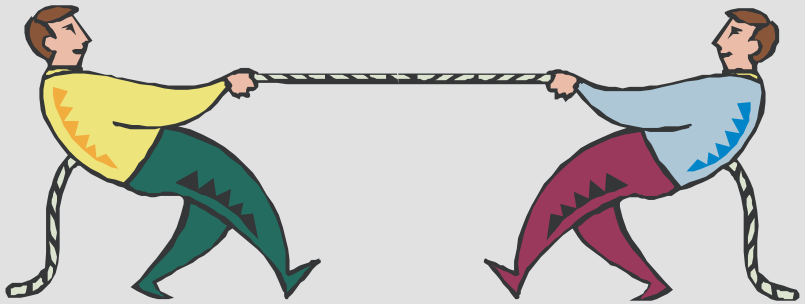
Real-Time Response vs. Throughput

Efficiency and Responsiveness are Inversely Related

- Overhead for Real-Time Preemption
  - Mutex Operations more complex than Spinlock Operations
  - Priority Inheritance on Mutex increases Task Switching
  - Priority Inheritance increases Worst-Case Execution Time

- Design flexibility allows much better worst case scenarios
  - Real-time tasks are designed to use kernel resources in managed ways then delays can be eliminated or reduced

Throughput



High responsiveness

Second mistake: prio 99

testrt.c:

```
#include <pthread.h>
int main(void)
{
    set_my_priority_to_highest();
    while (true)
      {;}
    return 0;
}
```

or:

```
while (someVolatile!=-1)
{
    sched_yield();
}
```

Third mistake: don't tell you use RT

- This really happened!
- *"NFS client stops working after 4-6 minutes"*

- Customer didn't provide kernel config (even after asking 4x)
- Support engineer started checking all kinds of configuration on both NFS client and server side

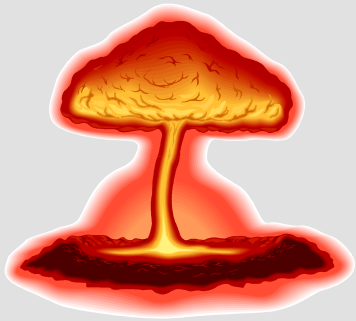
- (The real solution was improper locking in their UP-only network driver)

Fourth mistake: “I only have a single CPU!”

- In RT any process can be preempted at any time
- Thus very similar to multi-processor:
  - same code can run simultaneously at different cores
- All requirements for SMP-safeness also apply to RT
- RT and SMP share the same advanced locking
- Using deadlock detection in RT
  - already led to 100s of SMP bug fixes in the kernel

Fifth mistake:
RT process swapped to disk

- What happens if:
  - Your system is low on memory AND your RT task's code pages are freed or were swapped to disk?



- Solution:

```
mlockall(MCL_CURRENT | MCL_FUTURE)
```

- Only do this on small processes!
  - ALL memory pages in the process space will be locked into memory
  - Imagine what this does to a big multithreaded app ☺

Sixth mistake:
Expect someone else to test it

- Linux RT comes with NO WARRANTY

Hardware configuration impacts RT

- YOU have to verify it works well

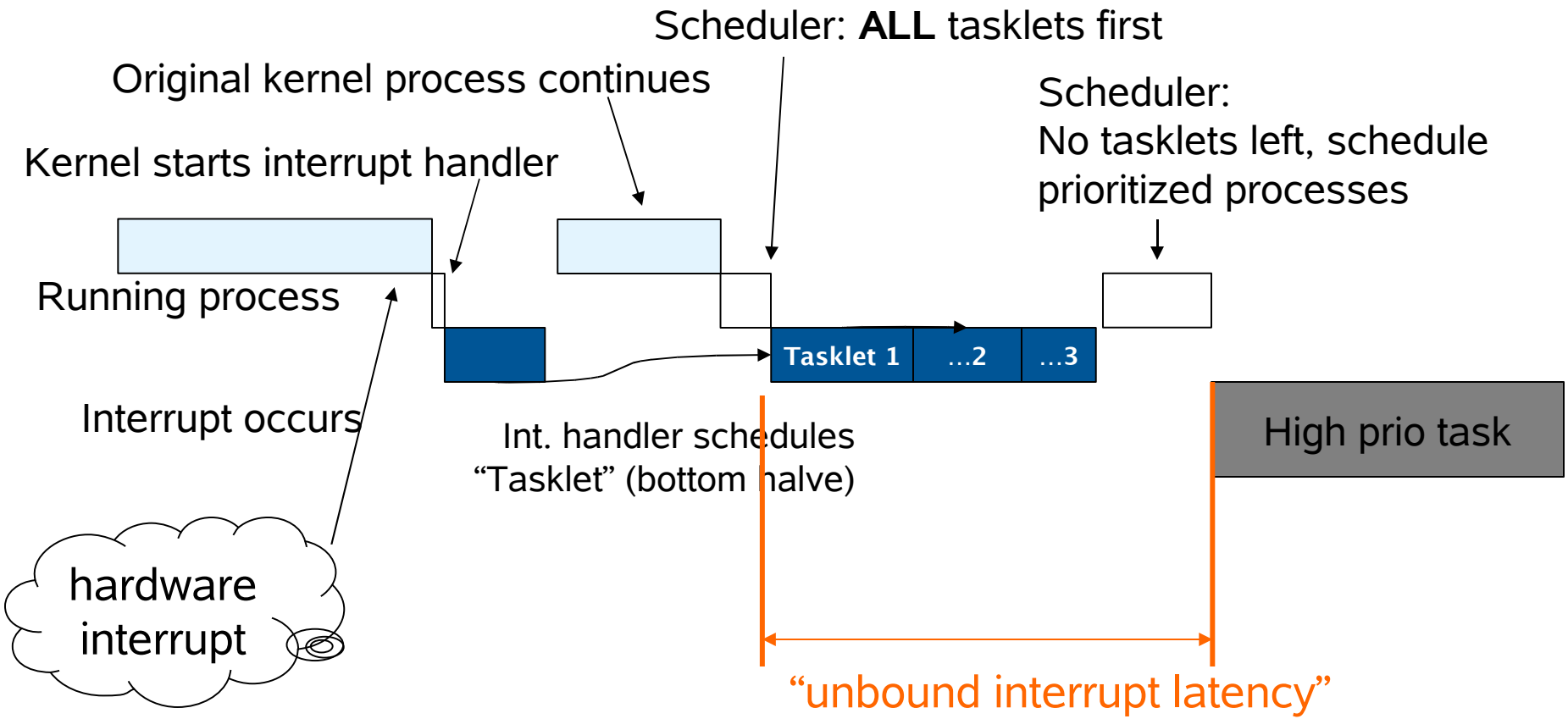
- Some random tips:
  - test with caches off
  - test with extra system load
  - ~~“quiet” on command line – no printk()~~
  - run at least once with IRQ latency tracing and fix it

(This slide was not in the original presentation)

Responses from the participants in the BOF

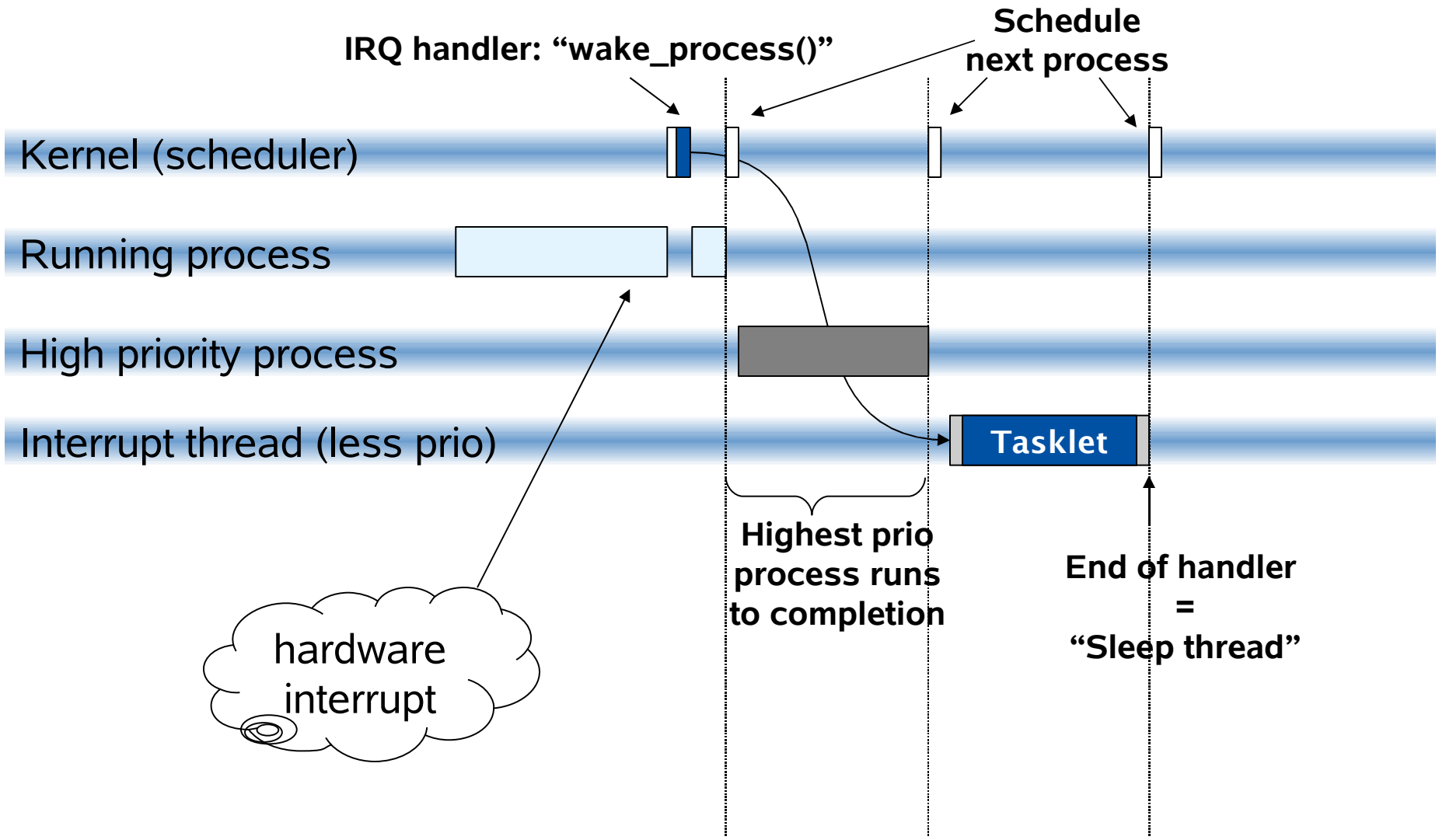
- 3rd Party binary drivers are not compatible with the locking mechanisms of the RT kernels, they need a recompile!!!
- Using soft floating point (and/or floating point kernel emulation) is not compatible with RT at the moment
- The current mechanism to distinguish between raw spinlocks and sleeping spinlocks works at compile time. It however confuses **`gdb`** and **`ctags`**
- Due to **`lockdep`**, future wrong usage of spinlocks or **`irq_disable()`** will be trapped before any patch enters the kernel tree
- Someone made the statement that up to today, the RT tree has caused almost 2000 patches (SMP bug fixes, but also the feature patches) to be accepted in the mainline kernel. Thus far more than on slide 21.

Standard Linux Interrupt Handlers



RT-patch Thread Context Interrupt Handlers

montavista




Back