montavista

Using Real-Time Linux

Common pitfalls, tips & tricks

Klaas van Gend,
Senior Solutions Architect,
MontaVista Software, Europe

Who is Klaas van Gend?

montavista

Klaas-the-Geek:

- Started programming age 13
- First encountered Linux 1993
- Software Engineer since 1998
- Lead developer of umtsmon
- Program Committee member for various open source conferences

Klaas-the-Sales-Guy:

- Joined MontaVista as FAE (not sales) 2004
- UK/Benelux/Israel territory
- Senior Solutions Architect for Europe
- Awarded FAE of the year 2006

Images do not necessarily depict reality

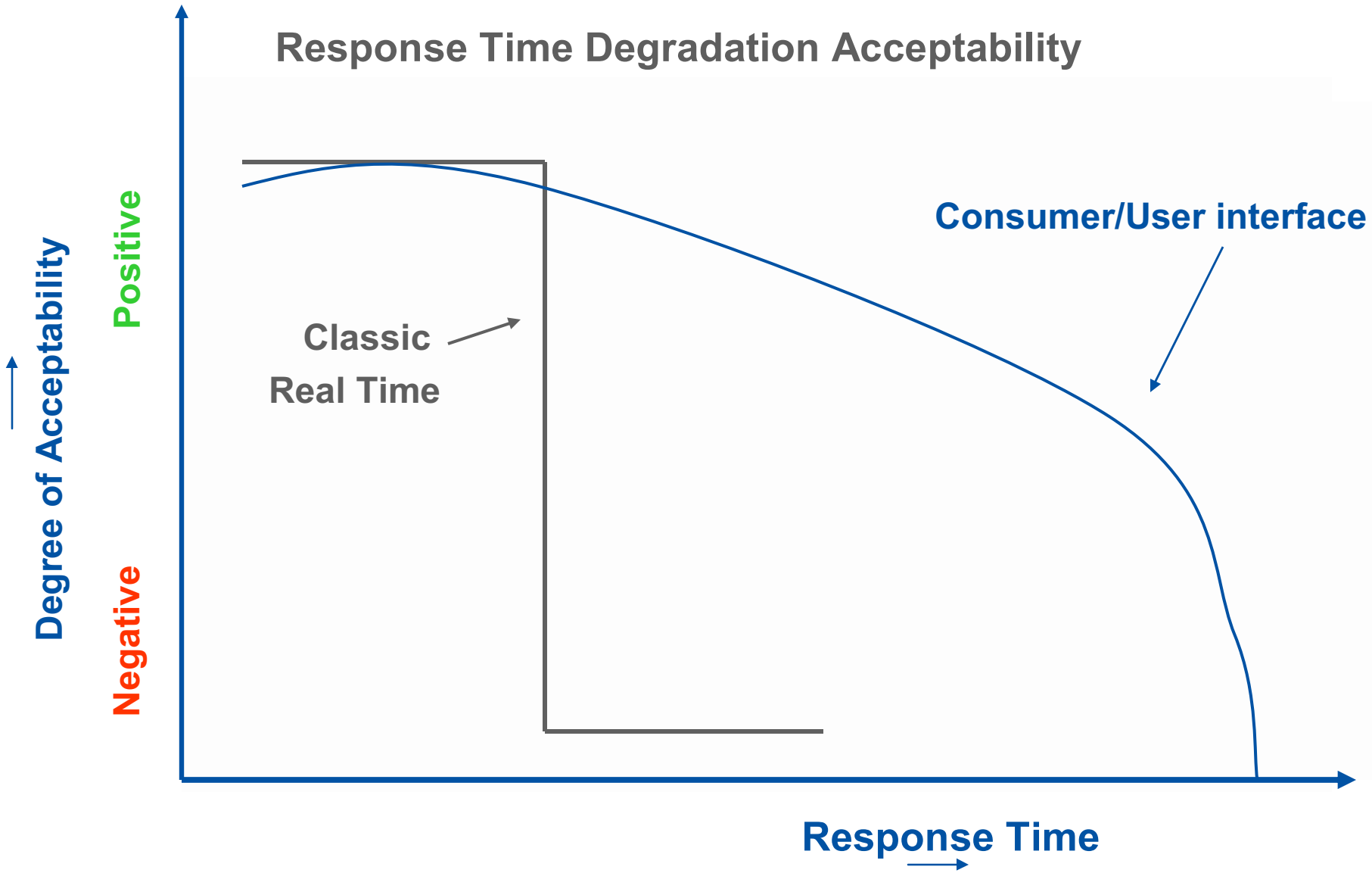
History of Linux and Real Time

montavista

- Fairness
- Preemption in user space
- Fixed Overhead / O(1) Scheduler
- Robert Love's Preemption in kernel
- Ingo Molnar's Voluntary preemption

Two types of Real-Time Expectations

montavista



Main assumption:

The highest priority task goes first

ALWAYS

Thus:

- **Everything should be pre-emptable**
- **Nothing should keep higher priority things from executing**

Key Elements of Real-Time Linux

montavista

- **Making Linux Real-time required addressing:**
  - Minimized interrupt disable times
  - Interrupt handling via schedulable threads
  - Fully pre-emptable kernel
    - Short critical sections
  - Perform synchronization via mutexes (not spin locks)
    - Allows involuntary pre-emption
  - Mutex support for priority inheritance
  - High Resolution timers

Sleeping Spinlocks

montavista

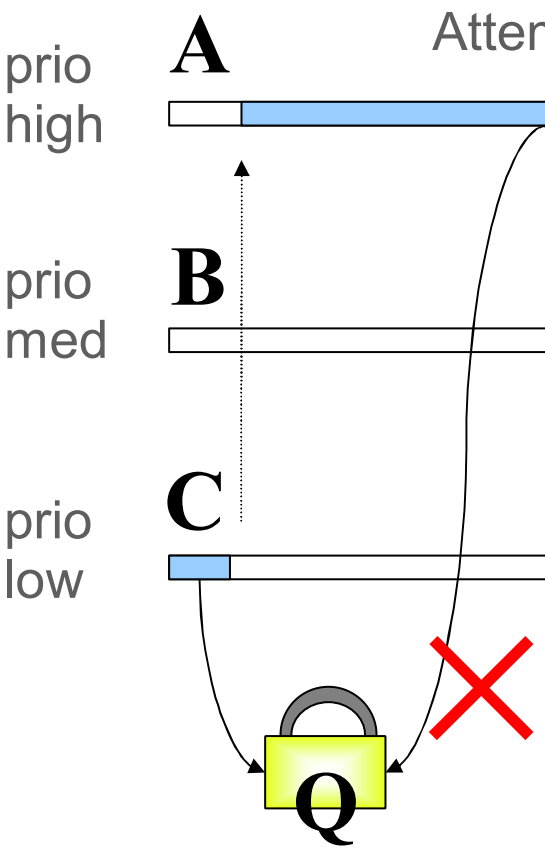
- **Original Linux** UP **Spinlock:**
  - *IRQ disable* on lock – nothing else can interrupt
  - Not RT friendly
- **Original Linux** SMP **Spinlock:**
  - Spinning (busy wait)
  - Not performance friendly

Solution:

“Sleeping Spinlock”

Problem: Priority Inversion

montavista



Solution:

“Priority Inheritance”

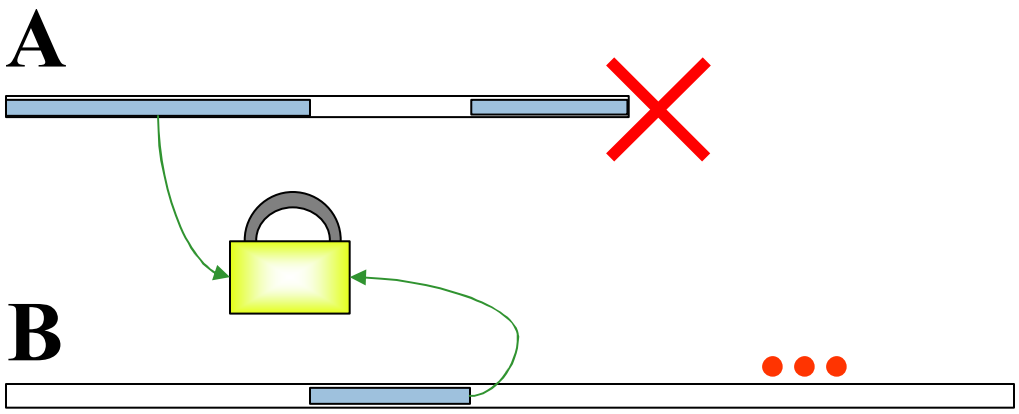
Robust Mutexes

montavista

Problem:

- **Inter *process* semaphores (“named ~”)**
- **Process A holds semaphore and dies**
- **Process B blocks on the same semaphore**
- **On regular Linux:** *mutex locked forever*
  - Thus waiting process B held forever
  - …until reboot

Solution:
“Robust Mutex”



Priority Queues

montavista

Problem:

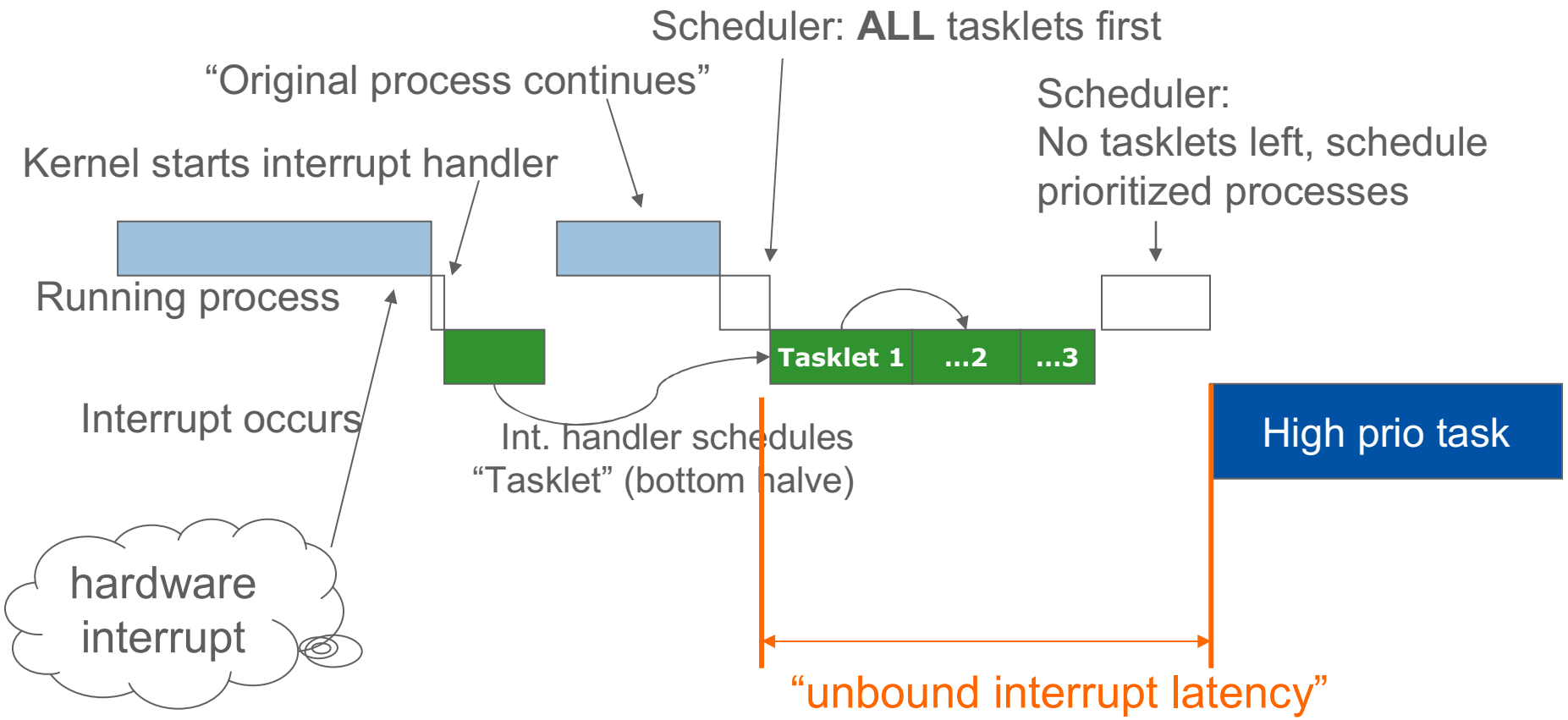
- 1000 processes waiting for a locked mutex
- Mutex gets unlocked – ***who will go first?***
- On regular Linux, the first waiting process 'gets' the mutex
- On RT Linux, the *highest priority* process should wake up and get the lock

Solution:
"Priority Queues"

Real Time is NOT fair, remember?

What's wrong with the standard IRQ mechanism?

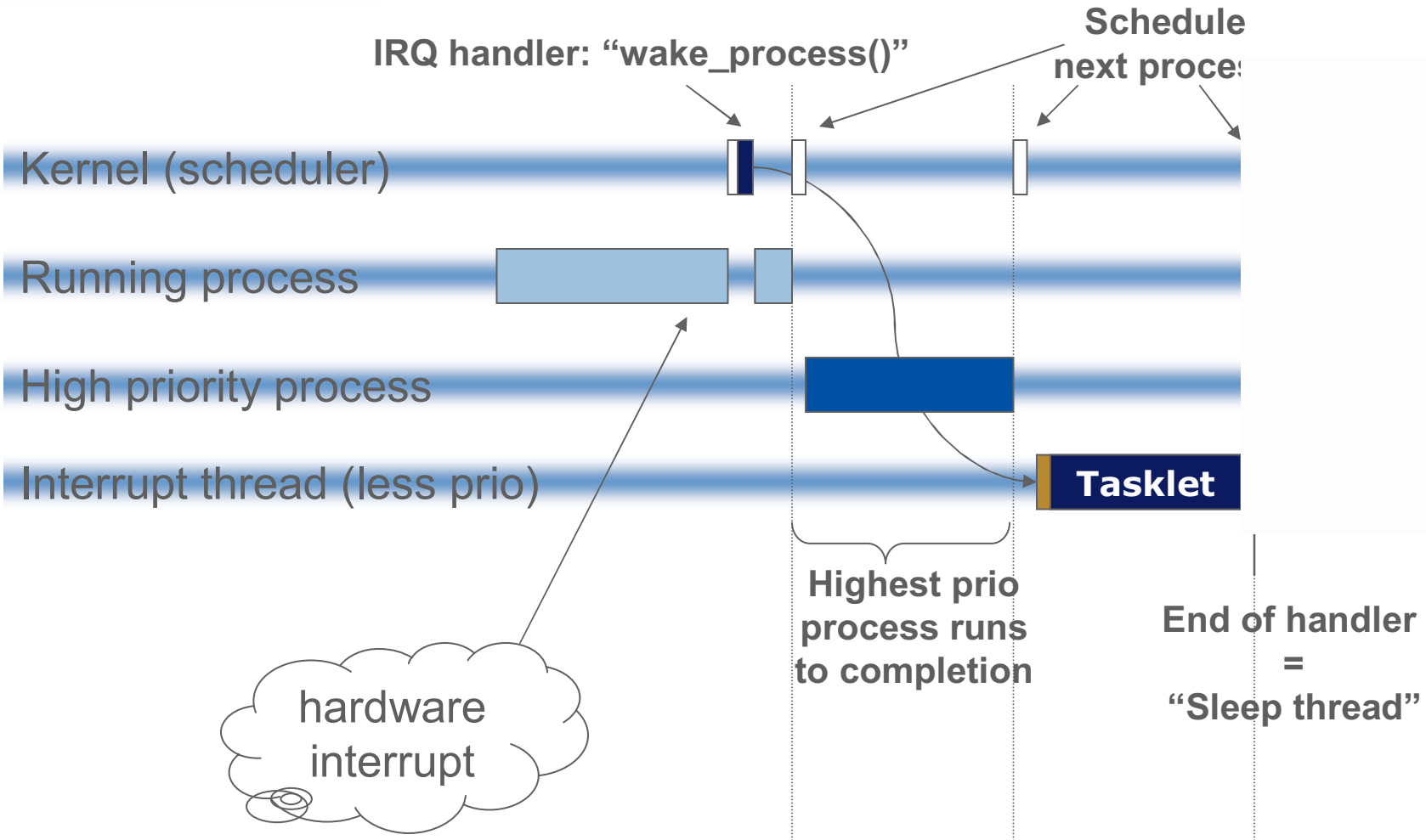
montavista



Solution:

"Threaded IRQs"

RT-patch Thread Context Interrupt Handlers



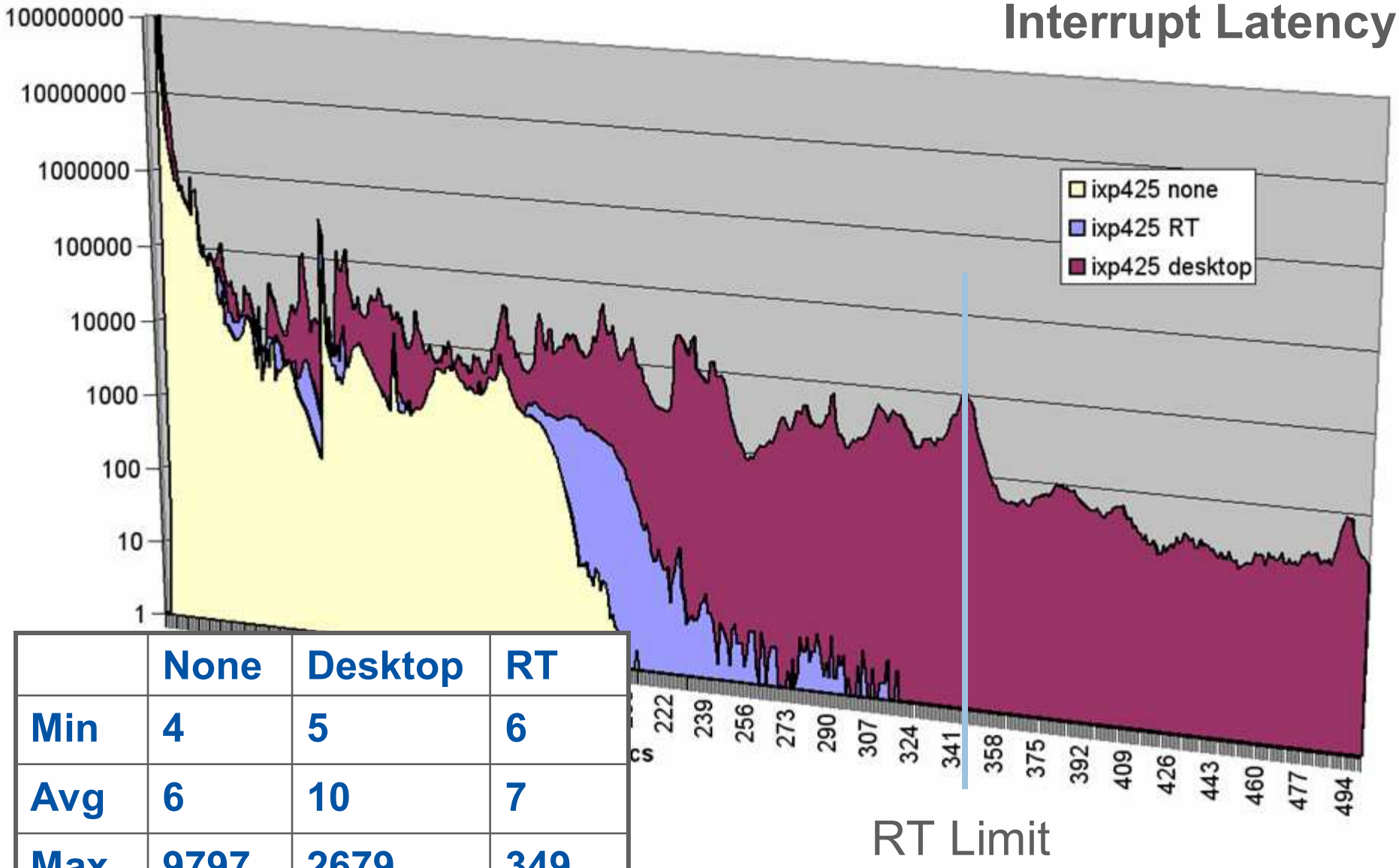
montavista™

Some Results

Intel IXP425 @ xxx Mhz, 2.6.18+

montavista

Interrupt Latency

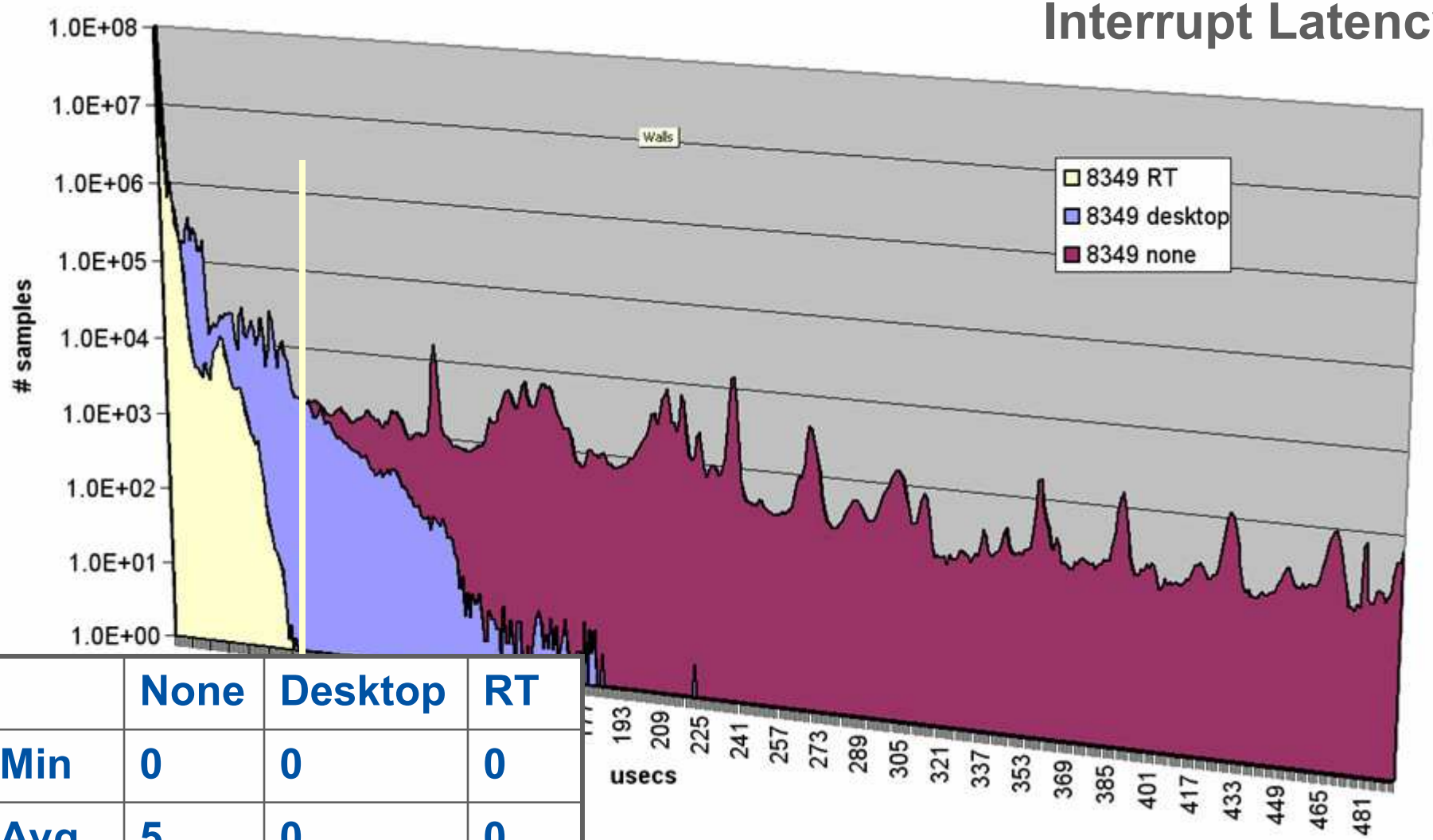


	None	Desktop	RT
Min	4	5	6
Avg	6	10	7
Max	9797	2679	349

FreeScale 8349 mITX @xxxMHz, 2.6.18+

montavista

Interrupt Latency



	None	Desktop	RT
Min	0	0	0
Avg	5	0	0
Max	3968	1604	53

More Results?

montavista

- **Request Real Time whitepaper**
  - By Bill Weinberg
  - http://www.mvista.com/

montavista

Common Mistakes & Myths

+ Tips & Tricks on Real Time

Mistake: “Fast” vs Determinism

montavista

“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 THROUGHPUT

Real-Time Response vs. Throughput

montavista

Efficiency and Responsiveness are Inversely Related

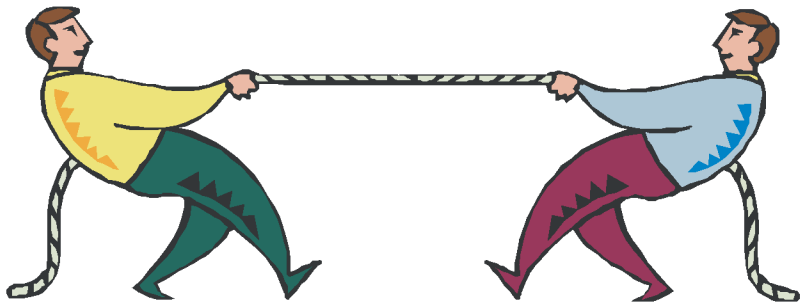
Overhead for Real-Time Preemption

- 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

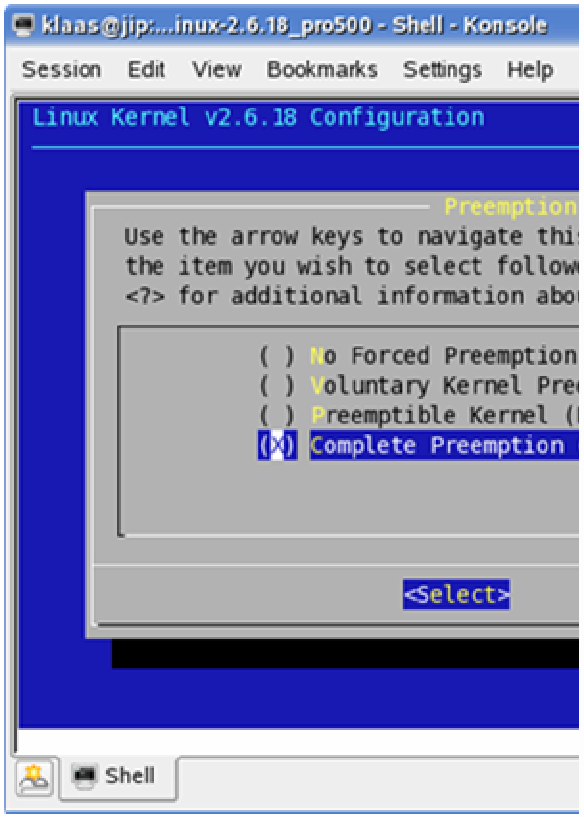
Throughput



High responsiveness

Mistake: forgetting to recompile

montavista



- **All kernel files need a recompile**
  - Function calls change
  - The scheduler gets extra code
  - IRQ mechanisms change
    - (even though the tasklet code doesn't change!)
  - Macros change
- **Syscalls do not change**
  - No need to recompile glibc
- **This also is true for out-of-tree modules**
  - You'll get very weird issues at module insertion or later…

RT doesn't mix with 3rdParty binary kernel modules !

Mistake: Forget to enable Robustness/PI in userland

montavista

```
#include <pthread.h>

// create the mutex
pthread_mutex_t mutex1;
pthread_mutex_init(&mutex1, NULL);

// create attributes struct
pthread_mutexattr_t myAttr;
pthread_mutexattr_init(&myAttr);

// set the corresponding fields
pthread_mutexattr_setprotocol (&myAttr, PTHREAD_PRIO_INHERIT);
pthread_mutexattr_setrobust_np (&myAttr, PTHREAD_MUTEX_ROBUST_NP);

// and apply to the mutex
pthread_mutex_init(&mutex1, &myAttr);
```

Mistake: “running at prio 99 froze my system”

montavista

testrt.c:

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

or:

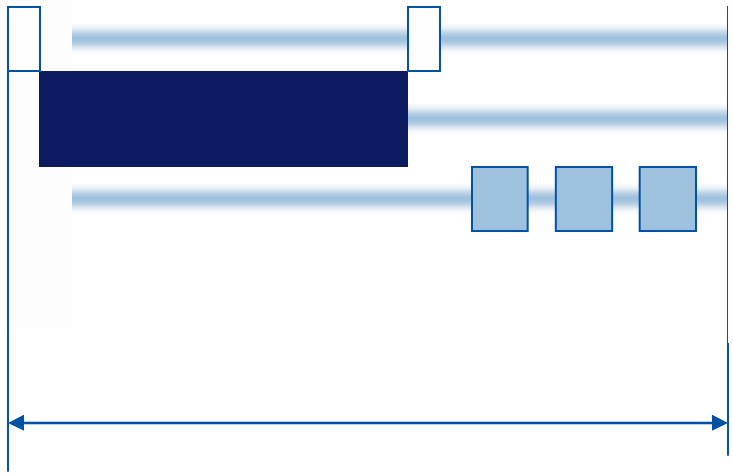
```
while (someVolatile != -1)
{
   sched_yield();
}
```

System Design Theory

montavista

- **You should only have one highest priority process***
  - IO-bound
    - control algorithms are IO-bound: they start and end with IO
  - Finite time running guarantee on your process
    - Definitely NO infinite loops!
  - sum(total running time + 2 x scheduler run) < latency req.

Scheduler
Highest Priority
Other tasks



Myth: “RT is difficult”

montavista

<<this page intentionally left blank>>

Myth: “RT is for embedded only”

montavista

- **RT pushed by audio community**
  - Audio not just a problem on Linux…
  - Ever used iTunes on a busy Windows XP laptop?

- **Games, Games, Games!**
  - Audio without pause/clicks/breaks/etc
  - Direct response to game controllers
  - Screen updates in hard real time - never missing frames

- **RT is the “true way”**
  - Voluntary pre-emption is “Windows95 in the kernel”
    - Not a good design, extra code, yada-yada
  - In 2 years from now, maybe only NONE and RT left?

Mistake: “But it works on normal Linux!”

montavista

- **Customer switched to real time:**
  - Geode x86-like board
  - Was missing bytes on serial ports
- **And he was missing even more bytes…**
  - When things ‘happened’
  - When he used alt+Fx to switch between X and text
- **PC BIOS:**
  - Scrolling a VGA buffer / switching VGA resolution
  - **Syslog** - by default logs to `/dev/tty8` or so

Mistake: “A Faster CPU will solve my problem”

montavista

- **Software becomes slower *faster* than hardware runs faster**
- **RT has been used as a “bugfix” to fix slowness**

This UART chip only had a 1 byte buffer!!!

Mistake: RT vs SMP in driver development

montavista

In RT *any* process can be preempted at *any* time

Thus very similar to multi-processor / multi-core:

- 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

Mistake: RT task swapped to disk

montavista

- **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 – *code* + *data* + *library!*
  - Imagine what this does to a big multithreaded app

- **Not just swap, page faults happen everywhere**
  - see http://rt.wiki.kernel.org/ and http://lwn.net/Articles/259710/

Myth/Mistake: “Linux Real Time is fully tested”

montavista

a.k.a. *“Gleixner did it – so it must work”*

- **Kernel community has spend many years developing / testing RT**
- **MontaVista *has* performed testing on all released RT-enabled Linux Support Packages**

But:

- There are 10M lines of code in the Linux kernel
- Linux RT comes with NO WARRANTY
- Hardware configuration significantly impacts RT, as do different code paths
- YOU have to verify it works well

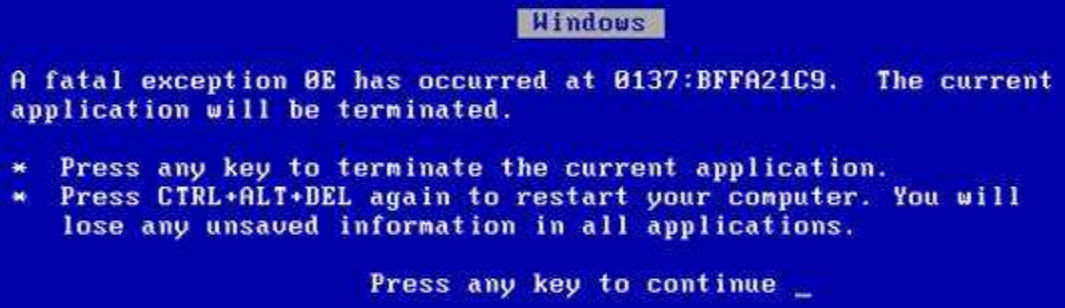
SUMMARY

montavista

- **Linux used to be fair – not good for RT**
- **MontaVista has worked on RT behavior since 1999**
- **True real time appeared in 2004**
  - Linux can be used for hard real time now
  - Interrupt latency on certain platforms always below 50 us
- **RT patch is still being merged into mainline kernel**
- **RT system design has its challenges**
  - But that's also true for programming in COBOL
  - This presentation uncovers *some* pitfalls and mistakes

"Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT." – Linus Torvalds

Fortunately, I run Linux ☺



Questions ???