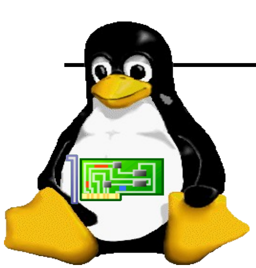
RTMux:

A Thin Multiplexer To Provide Hard Realtime Applications For Linux

Jim Huang (黃敬群) <jserv.tw@gmail.com>

Oct 15, 2014 / Embedded Linux Conference Europe



Agenda

- Mission: Build lightweight real-time environments for Linux/ARM
- Review of existing technologies
- RTMux: Resource-Multiplexing Real-time Executive
- Linux-friendly remote communication mechanisms

- Full source available: `https://github.com/rtmux`
- This work is sponsored by ITRI Taiwan and Delta Electronics

Mission:

Build Lightweight Real-time environments for Linux/ARM Applications

In short words, it is LOVER

LOVER =

Linux Optimized for Virtualization, Embedded, and Realtime

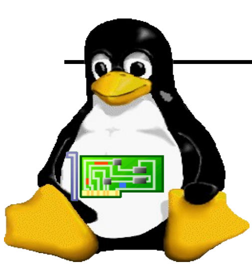
Use Case for RTMux

Quadcopter with Computer Vision

Use Case for RTMux

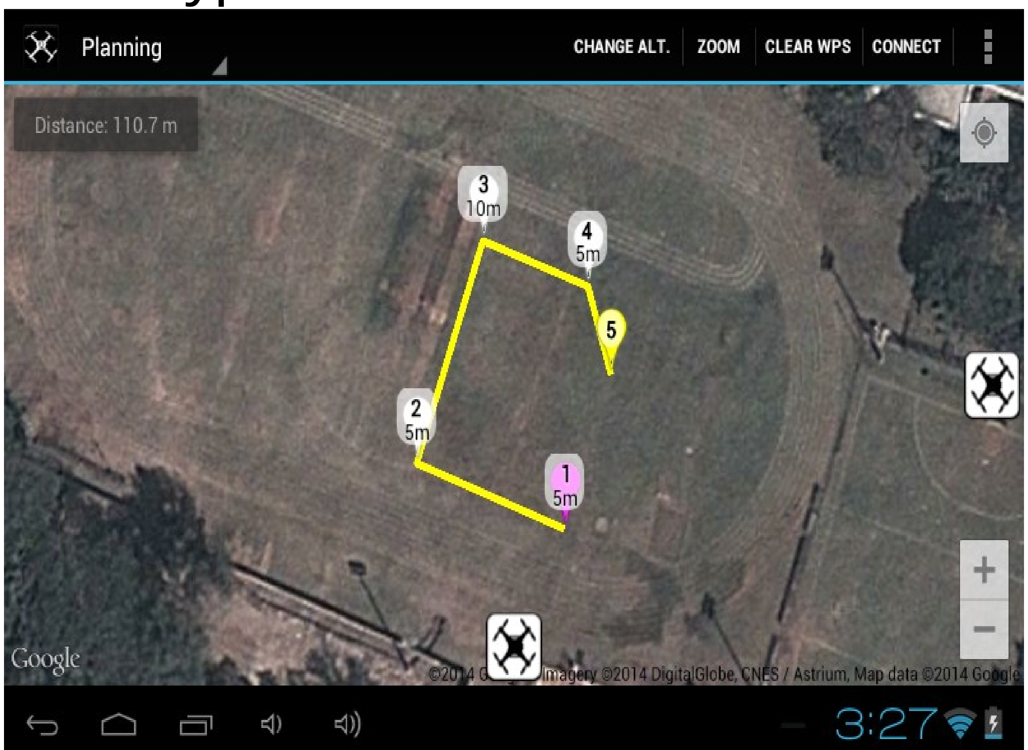
Quadcopter with Computer Vision

- Hard real-time
  - Autonomous Flight Modes (Landing/Take-off)
    - altitude control, feedback-loop control, RC
  - Autopilot, autonomous navigation
- Soft real-time
  - Stream real-time flight data on-screen over video
  - Parallel Tracking and Mapping (PTAM) , and the detected walls are visualized in 3D with mapped textures.
    - Source: https://github.com/nymanjens/ardrone-exploration



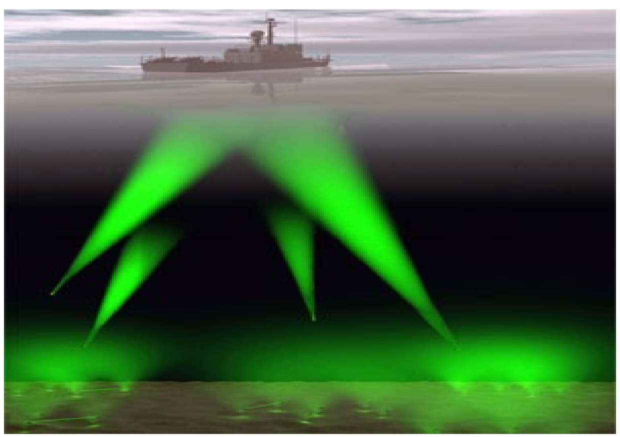
External Autonomous Navigation

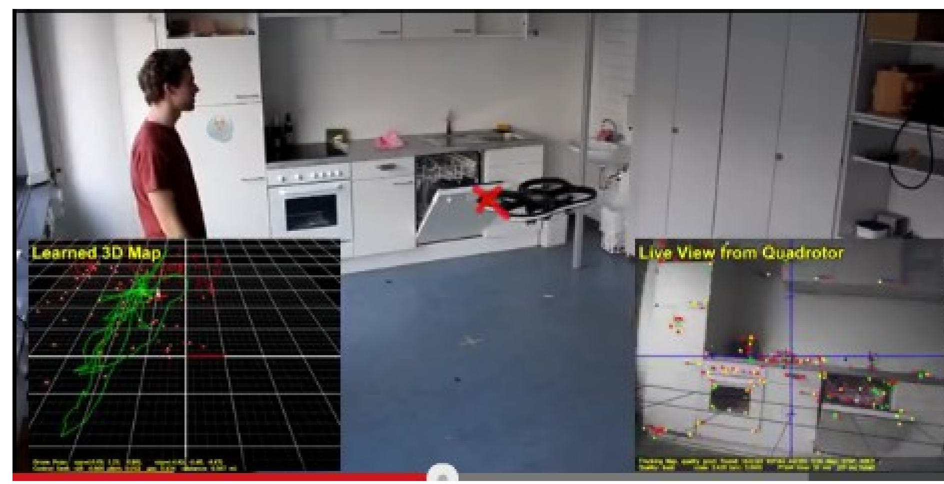
- Various Flight Modes-Stabilize, Alt Hold, Loiter, Auto Mode.
- For the AUTO mode, GPS is necessary.
- Waypoints are set in advance.



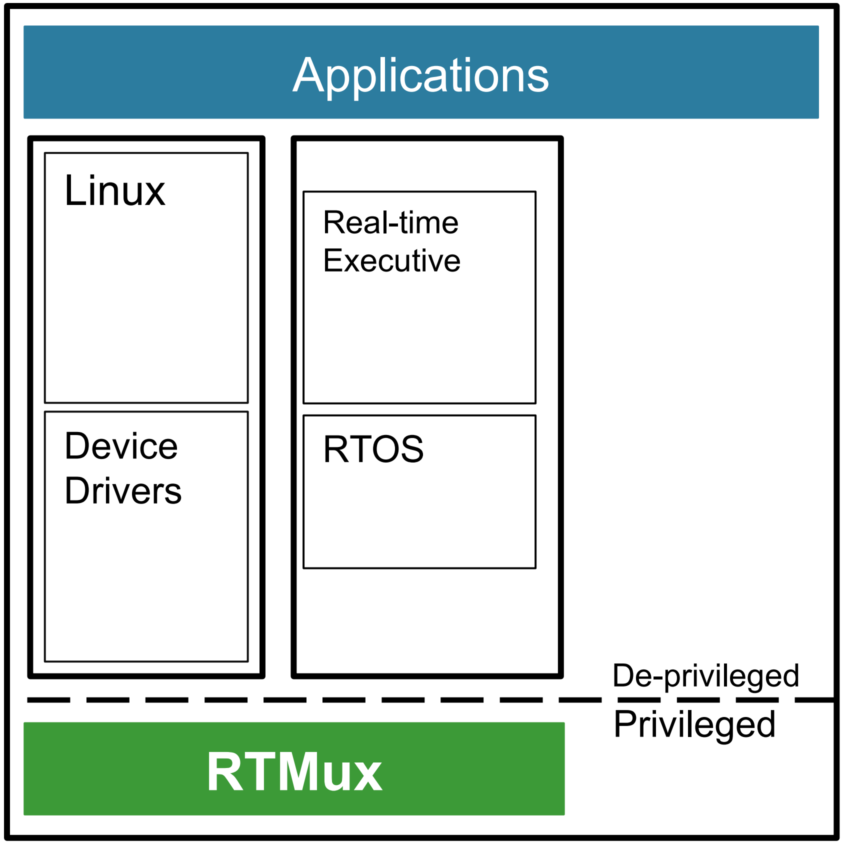
Internal Autonomous Navigation

- GPS fails in a closed-door environment.
  - Detect a door/window and go out where GPS access is present.
- Design a controller for navigation of quadcopter from indoor to outdoor environsments.
  - SONAR and Computer vision





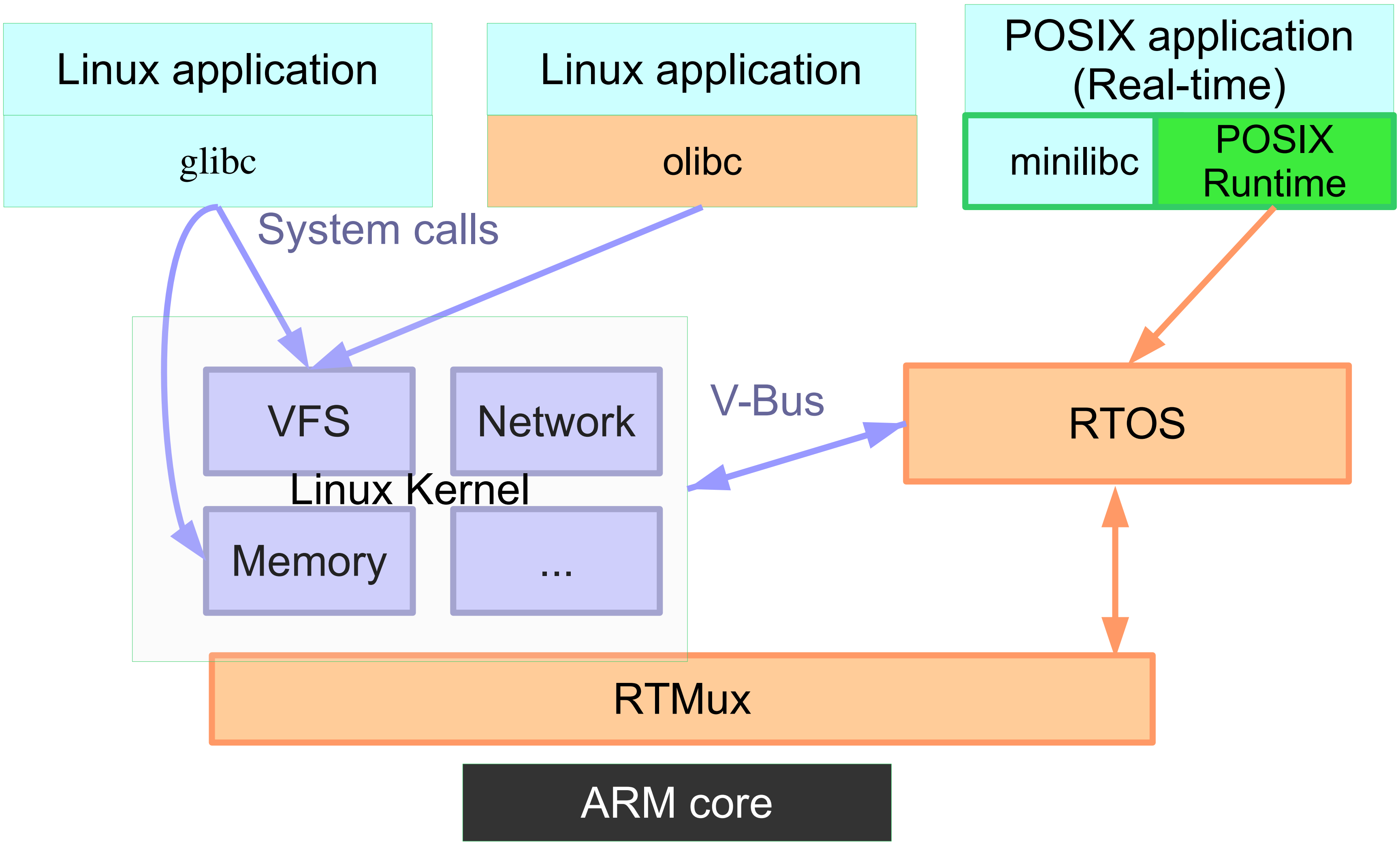
Source: http://wiki.ros.org/tum_ardrone

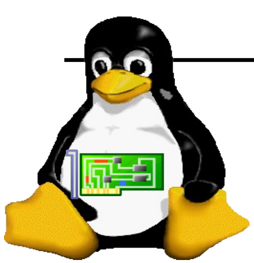


RTMux:

Multiplexer for Linux-based Real-time Applications

Powered by Open Source Stack





olibc: http://www.slideshare.net/jserv/olibc

Review of Existing Technologies

Realtime Performance

Application program

100% source compatibility
better binary compatibility

API

Customized Realtime kernel

Real-time Linux

(Standard) Linux

Response time: < 10 µs

One (physical)
address space with n-tasks

All tasks have
one common API
implemented by
one common library

Response time: ~1ms

One process address space
with n-threads

All threads of a process have
one common API
implemented by
one common library

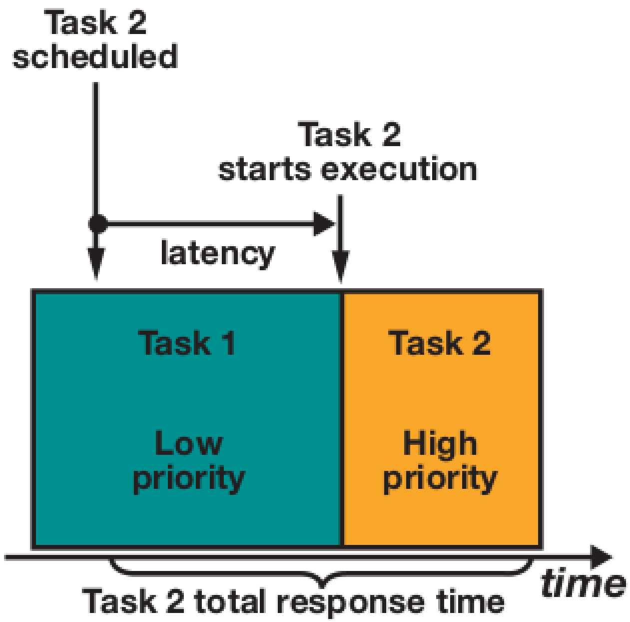
Response time: < 10 µs
same
homogeneity

Real-time Approaches

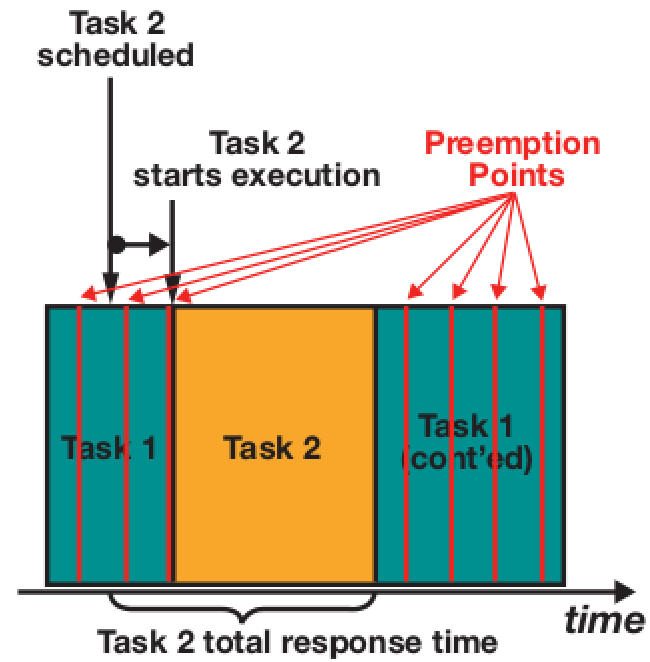
Two major approaches real time Linux

- rt-preempt (PREEMPT_RT patch)
  - Allows preemption, so minimize latencies
  - Execute all activities (including IRQ) in “schedulable/thread” context
  - Many of the RT patch have been merged
- Linux (realtime) extensions
  - Add extra layer between hardware and the Linux kernel to manage real-time tasks separately

Preemptive Kernel



non-preemptive system



preemptive system

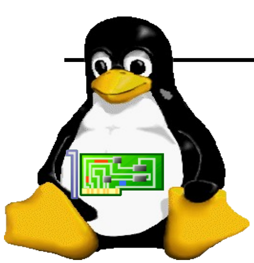
A concept linked to that of real time is preemption: the ability of a system to interrupt tasks at many “preemption points”.The longer the non-interruptible program units are, the longer is the waiting time (‘latency’) of a higher priority task before it can be started or resumed. GNU/Linux is “user-space preemptible”: it allows user tasks to be interrupted at any point. The job of real-time extensions is to make system calls preemptible as well.

Part I: Linux real-time preemption

http://www.kernel.org/pub/linux/kernel/projects/rt/

- led by kernel developers including Ingo Molnar, Thomas Gleixner, and Steven Rostedt
  - Large testing efforts at RedHat, IBM, OSADL, Linutronix
- Goal is to improve real time performance
- Configurable in the `Processor type and features` (`x86`), `Kernel Features` (`arm`) or `Platform options` (`ppc`)...

Preemption Mode	
○ No Forced Preemption (Server)	PREEMPT_NONE
○ Voluntary Kernel Preemption (Desktop)	PREEMPT_VOLUNTARY
○ Preemptible Kernel (Low-Latency Desktop)	PREEMPT_DESKTOP
⦿ Complete Preemption (Real-Time)	PREEMPT_RT
Thread Softirqs	PREEMPT_SOFTIRQS
Thread Hardirqs	PREEMPT_HARDIRQS



Wrong ideas about real-time preemption

- *It will improve throughput and overall performance*
  Wrong: it will degrade overall performance.
- *It will reduce latency*
  Often wrong. The maximum latency will be reduced.

The primary goal is to make the system predictable and deterministic.

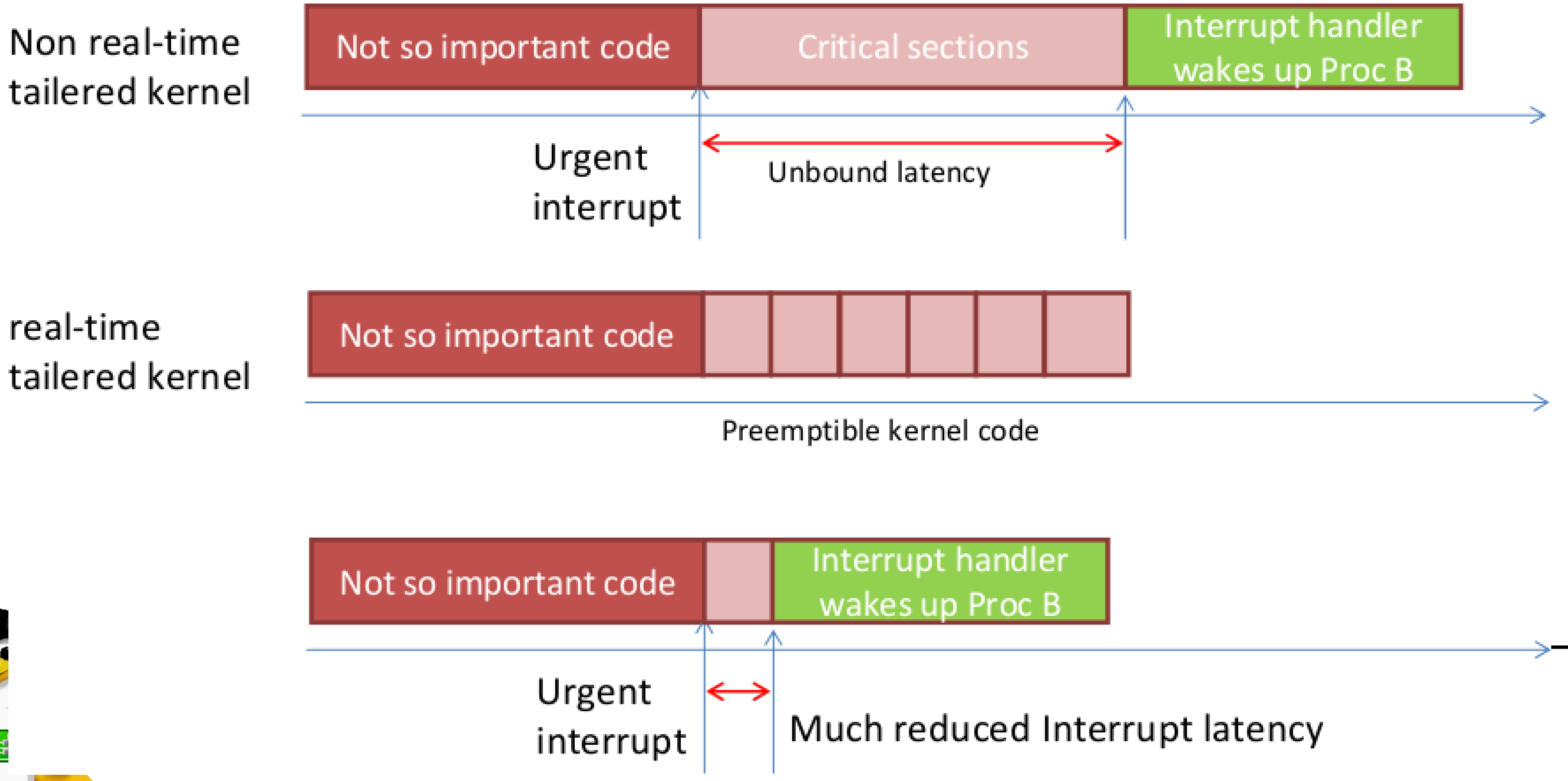
PREEMPT_RT: complete RT preemption

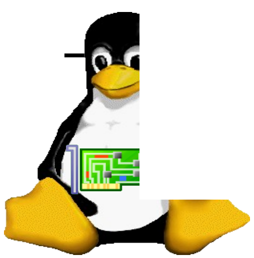
Replace non-preemptible constructs with preemptible ones

- Make OS preemptible as much as possible
  - except preempt_disable and interrupt disable
- Make Threaded (schedulable) IRQs
  - so that it can be scheduled
- spinlocks converted to mutexes (a.k.a. sleeping spinlocks)
  - Not disabling interrupt and allows preemption
  - Works well with thread interrupts

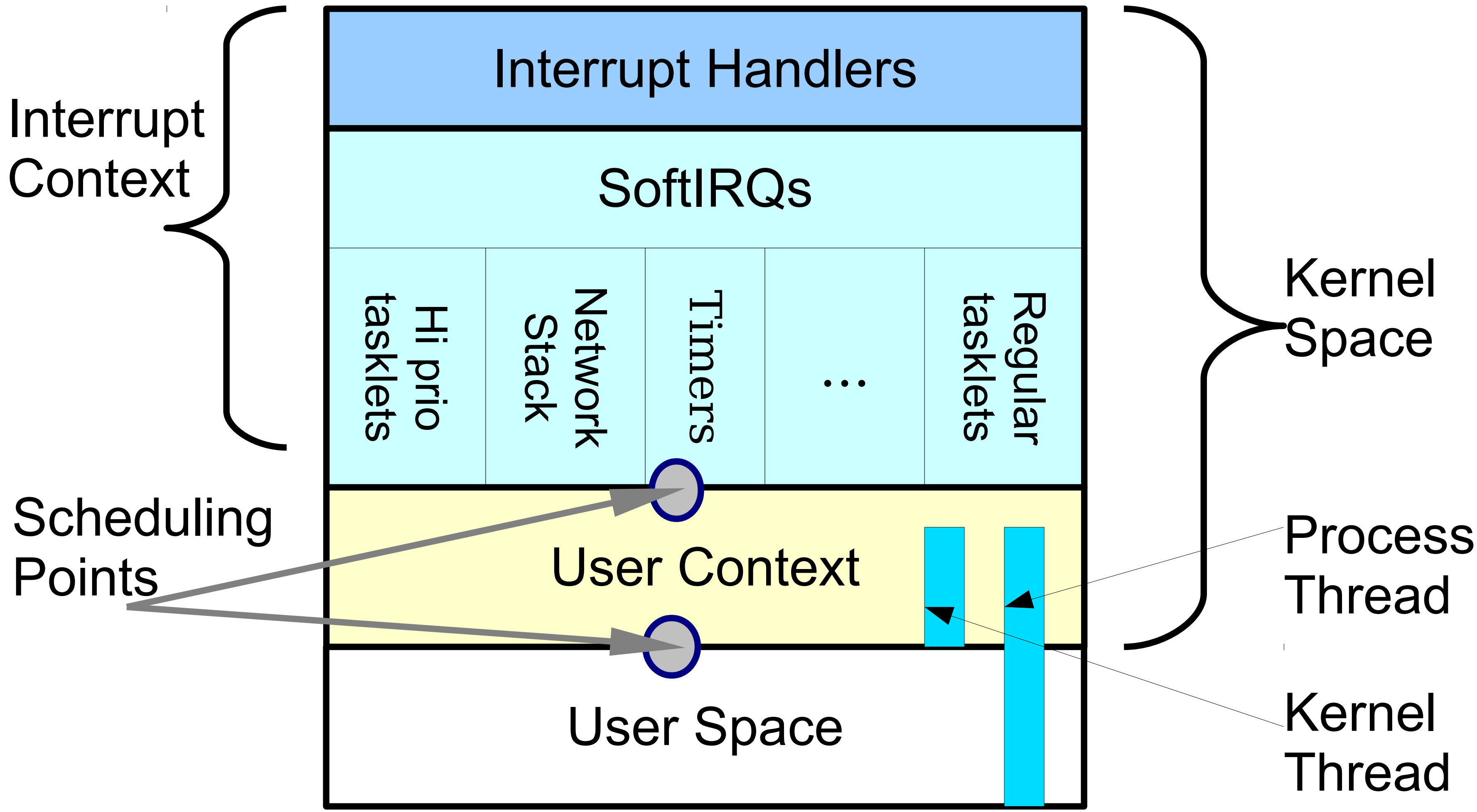
Toward complete RT preemption

- Most important aspects of Real-time
  - Controlling latency by allowing kernel to be preemptible everywhere

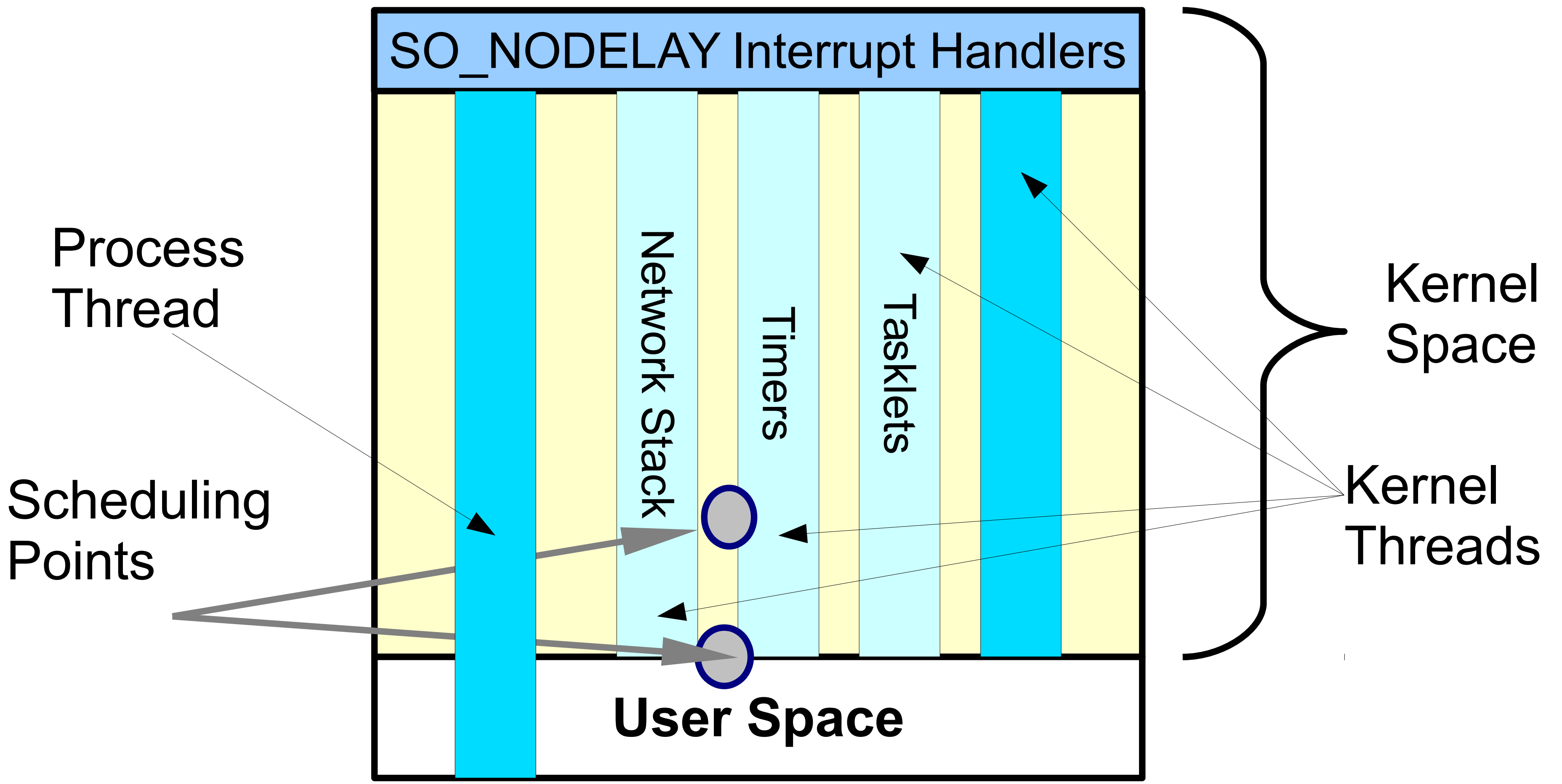




original Linux Kernel



PREEMPT_RT



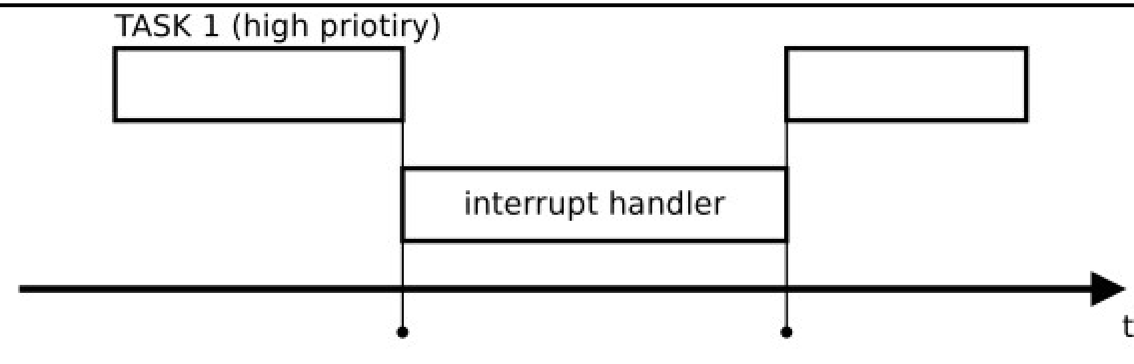
SO_NODELAY Interrupt Handlers
Network Stack
Timers
Tasklets
Process Thread
Scheduling Points
Kernel Space
Kernel Threads
User Space

Threaded Interrupts

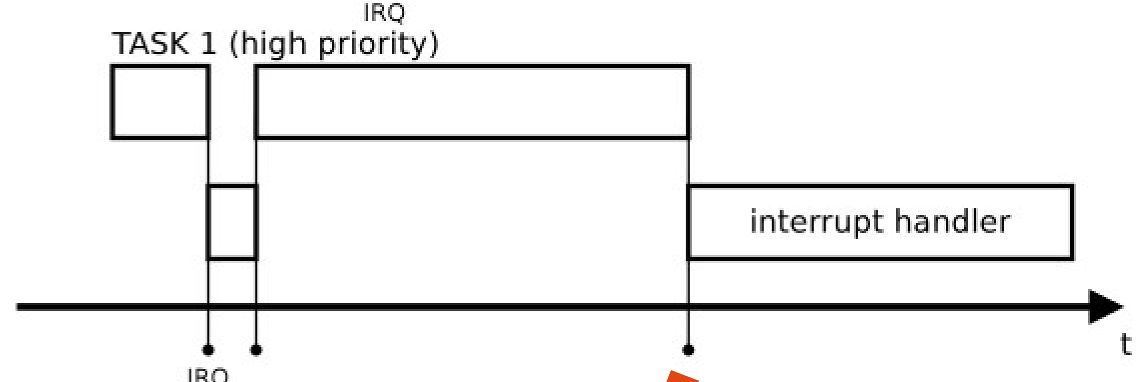
- Handle interrupt by interrupt handler thread
- Interrupt handlers run in normal kernel threads
  - Priorities can be configured
- Main interrupt handler
  - Do minimal work and wake-up the corresponding thread
- Thread interrupts allows to use sleeping spinlocks
- in PREEMPT_RT, all interrupt handlers are switched to threaded interrupt

Threaded Interrupts

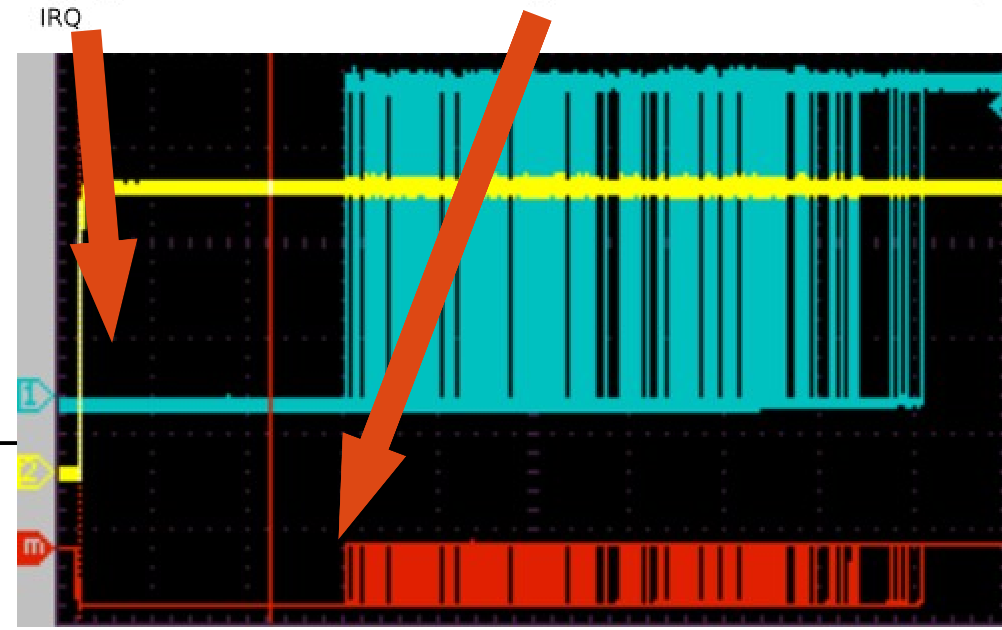
- The vanilla kernel



- Interrupts as threads



- Real world behavior



Benchmarking

cyclictest

- measuring accuracy of sleep and wake operations of highly prioritized realtime threads
- https://rt.wiki.kernel.org/index.php/Cyclictest

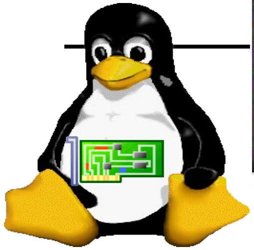
```
insop@chai:~/Projects/rt-tests$ uname -a
Linux chai 3.2.0-24-generic-pae #39-Ubuntu SMP Mon May 21 18:54:21 UTC 2012 i686 i686 i386 GNU/Linux
insop@chai:~/Projects/rt-tests$ sudo ./cyclictest -a -t -n -p99
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.54 0.69 0.67 6/417 3256

T: 0 ( 2772) P:99 I:1000 C:1008249 Min:      4 Act:   18 Avg:   11 Max:     701
T: 1 ( 2773) P:99 I:1500 C: 672166 Min:      4 Act:   35 Avg:   11 Max:     491
T: 2 ( 2774) P:99 I:2000 C: 504124 Min:      4 Act:    9 Avg:   11 Max:     363
T: 3 ( 2775) P:99 I:2500 C: 403299 Min:      4 Act:   14 Avg:   11 Max:    2013
T: 4 ( 2776) P:99 I:3000 C: 336082 Min:      4 Act:   14 Avg:   14 Max:     804
T: 5 ( 2777) P:99 I:3500 C: 288071 Min:      3 Act:   13 Avg:    9 Max:     190
T: 6 ( 2778) P:99 I:4000 C: 252062 Min:      3 Act:    9 Avg:    9 Max:     343
T: 7 ( 2779) P:99 I:4500 C: 224055 Min:      3 Act:   13 Avg:   10 Max:     224
```

vanilla kernel

Worst case latency: hundreds of usec

```
T: 0 ( 2995) P:99 I:1000 C:1030921 Min:      5 Act:    7 Avg:   12 Max:      32
T: 1 ( 2996) P:99 I:1500 C: 687280 Min:      5 Act:    7 Avg:   13 Max:      53
T: 2 ( 2997) P:99 I:2000 C: 515455 Min:      4 Act:    6 Avg:   13 Max:      34
T: 3 ( 2998) P:99 I:2500 C: 412364 Min:      5 Act:    7 Avg:   13 Max:      34
T: 4 ( 2999) P:99 I:3000 C: 343637 Min:      6 Act:   11 Avg:   16 Max:      31
T: 5 ( 3000) P:99 I:3500 C: 294546 Min:      3 Act:    4 Avg:   11 Max:     338
T: 6 ( 3001) P:99 I:4000 C: 257727 Min:      4 Act:    5 Avg:   11 Max:      24
T: 7 ( 3002) P:99 I:4500 C: 229091 Min:      3 Act:    5 Avg:   11 Max:      29
```

PREEMPT_RT



Worst case latency: tens of usec

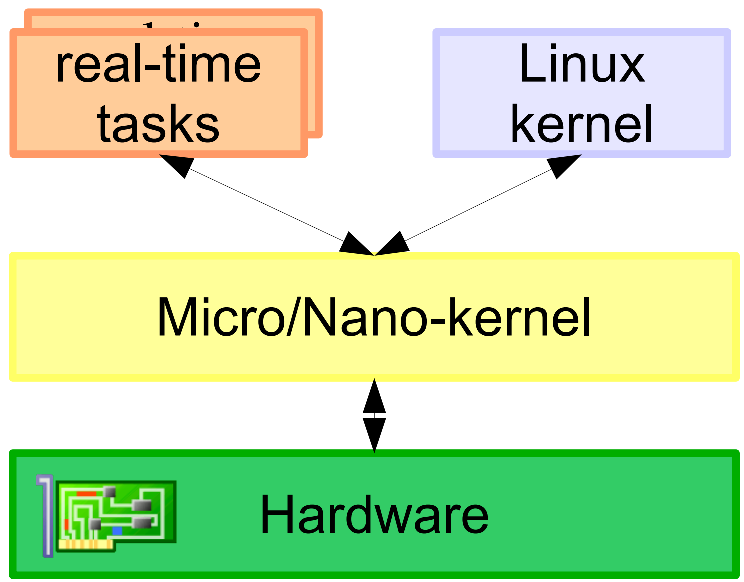
Part II: Linux hard real-time extensions

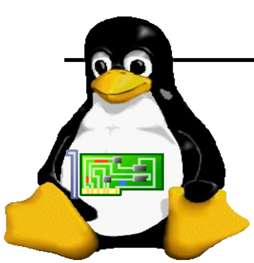
Three generations

- RTLinux
- RTAI
- Xenomai

A common principle

- Add a extra layer between the hardware and the Linux kernel, to manage real-time tasks separately.





Interrupt Response Time

PREEMPT: standard kernel with CONFIG_PREEMPT (“Preemptible Kernel (Low-Latency Desktop)) enabled
`cyclictest -m -n -p99 -t1 -i10000 -1360000`

XENOMAI: Kernel + Xenomai 2.6.0-rc4 + I-Pipe 1.18-03
`cyclictest -n -p99 -t1 -i10000 -1360000`

Configuration	Avg	Max	Min
XENOMAI	43	58	2
PREEMPT	88	415	27

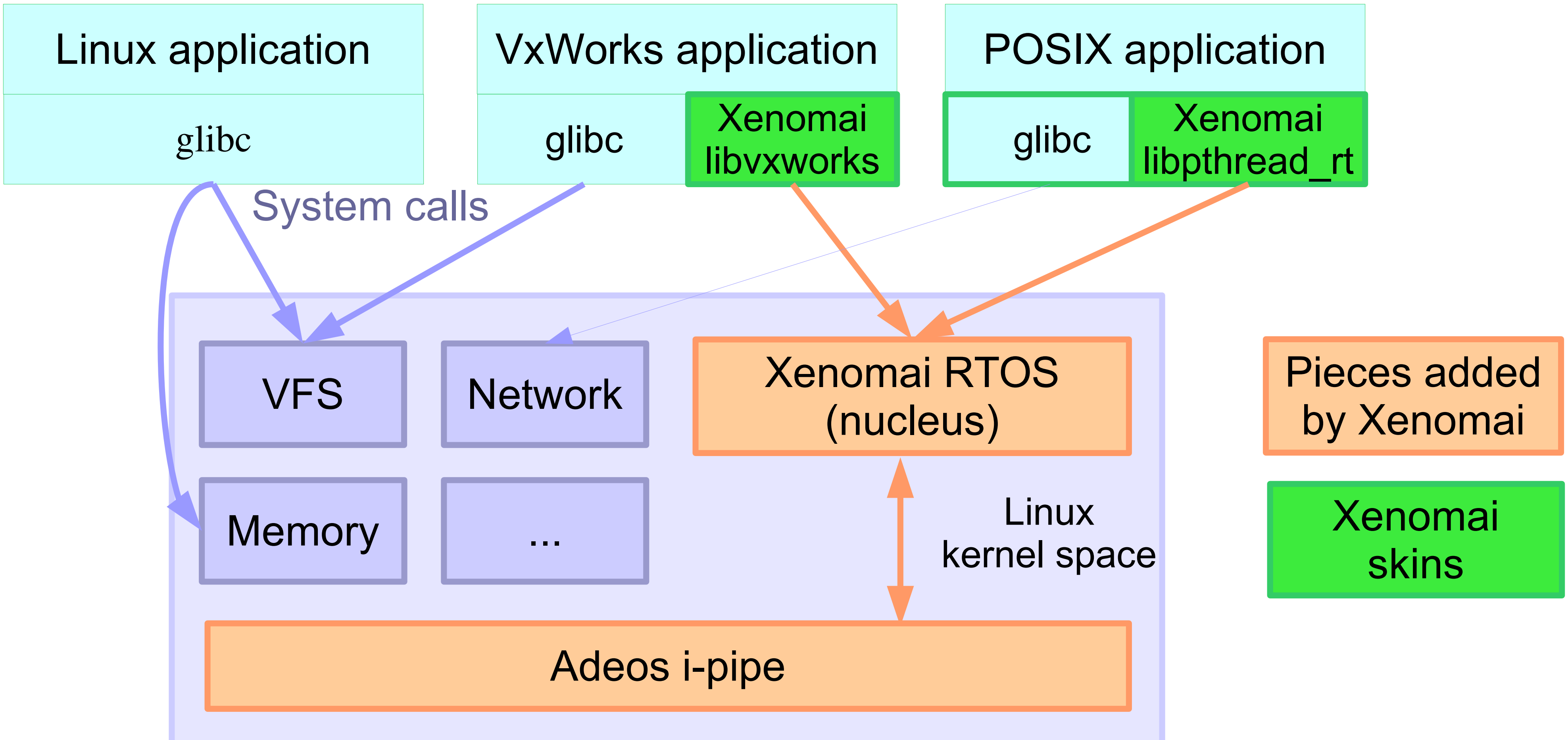
Hardware: Freescale i.MX53 ARM Cortex-A8 processor operating at 1GHz.
Time in micro second.

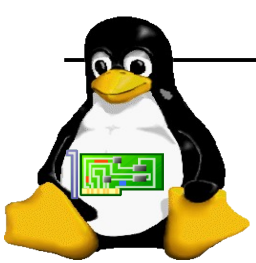
Xenomai project

http://www.xenomai.org/

- Started in the RTAI project (called RTAI / fusion).
- Skins mimicking the APIs of traditional RTOS such as VxWorks, pSOS+, and VRTXsa.
- Initial goals: facilitate the porting of programs from traditional RTOS to RTAI on GNU / Linux.
- Now an independent project and an alternative to RTAI. Many contributors left RTAI for Xenomai, frustrated by its goals and development style.

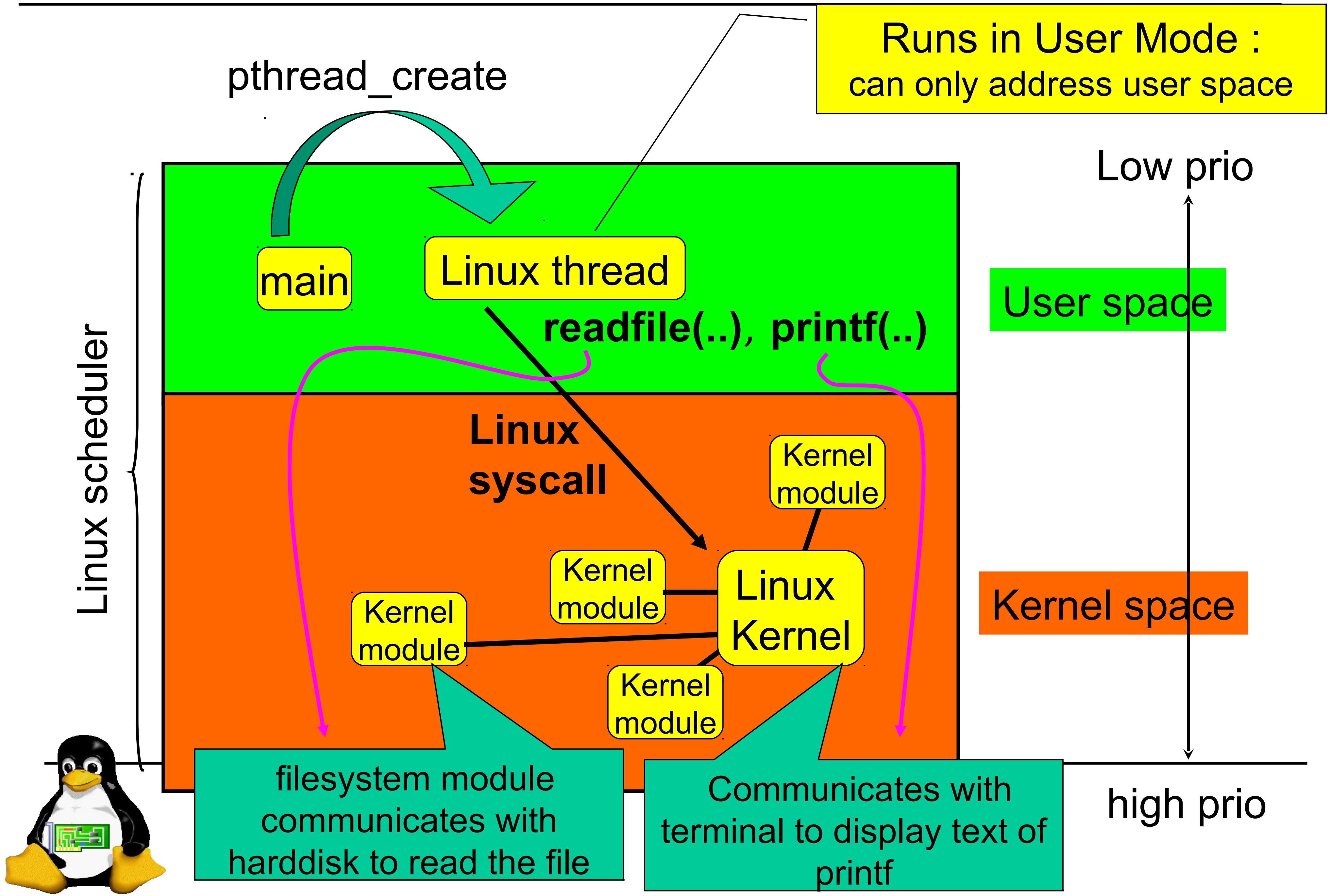
Xenomai architecture



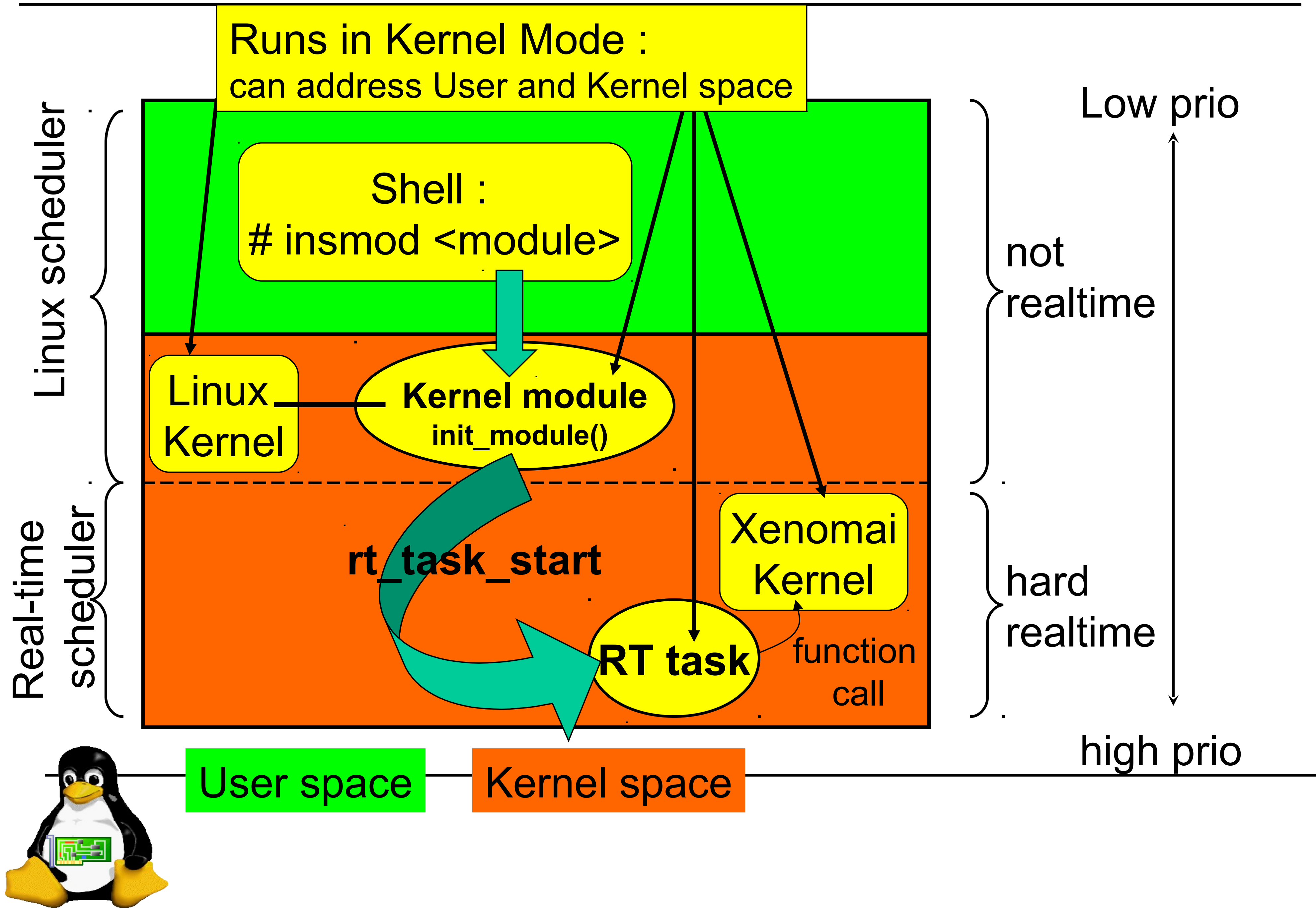


ipipe = interrupt pipeline

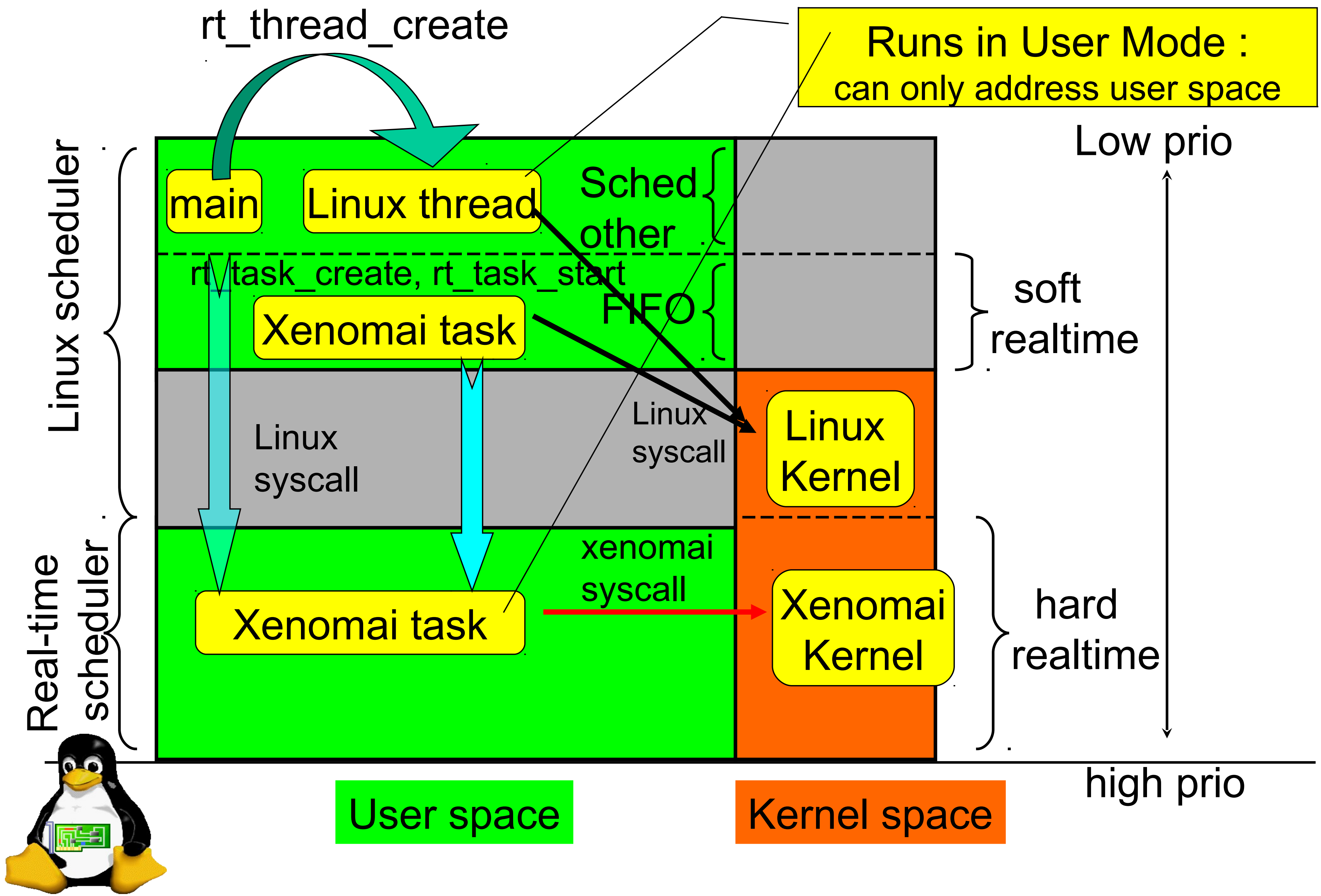
Original Linux



Xenomai (kernel space)

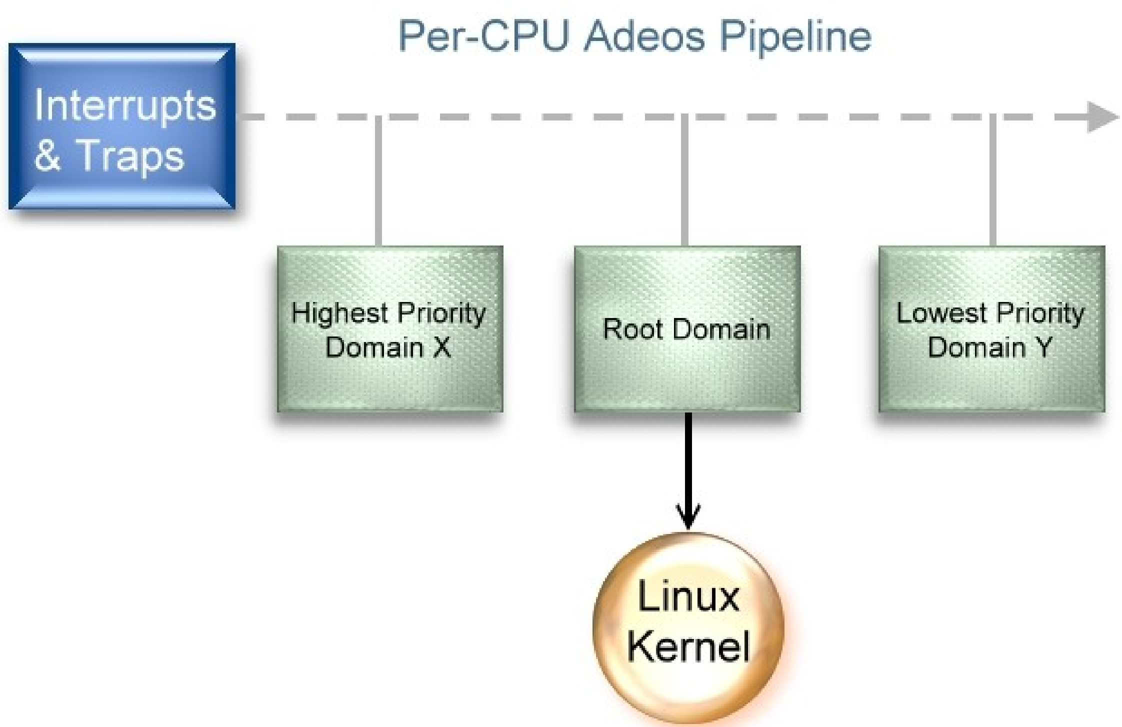


Xenomai (user space)



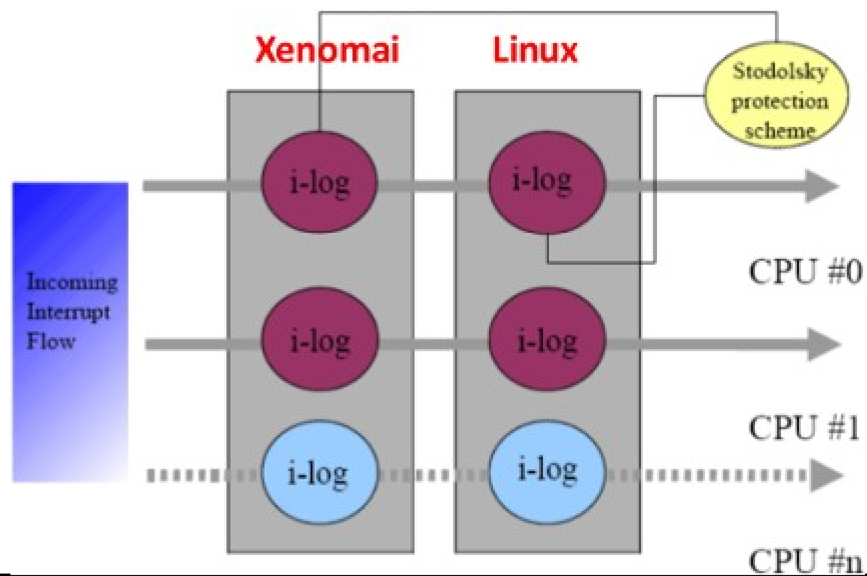
Xenomai internals: ipipe

- ipipe = Interrupt pipeline abstraction
  - guest OSes are regarded as prioritized domains.
- For each event (interrupts, exceptions, syscalls, ...), the various domains may handle the event or pass it down the pipeline.



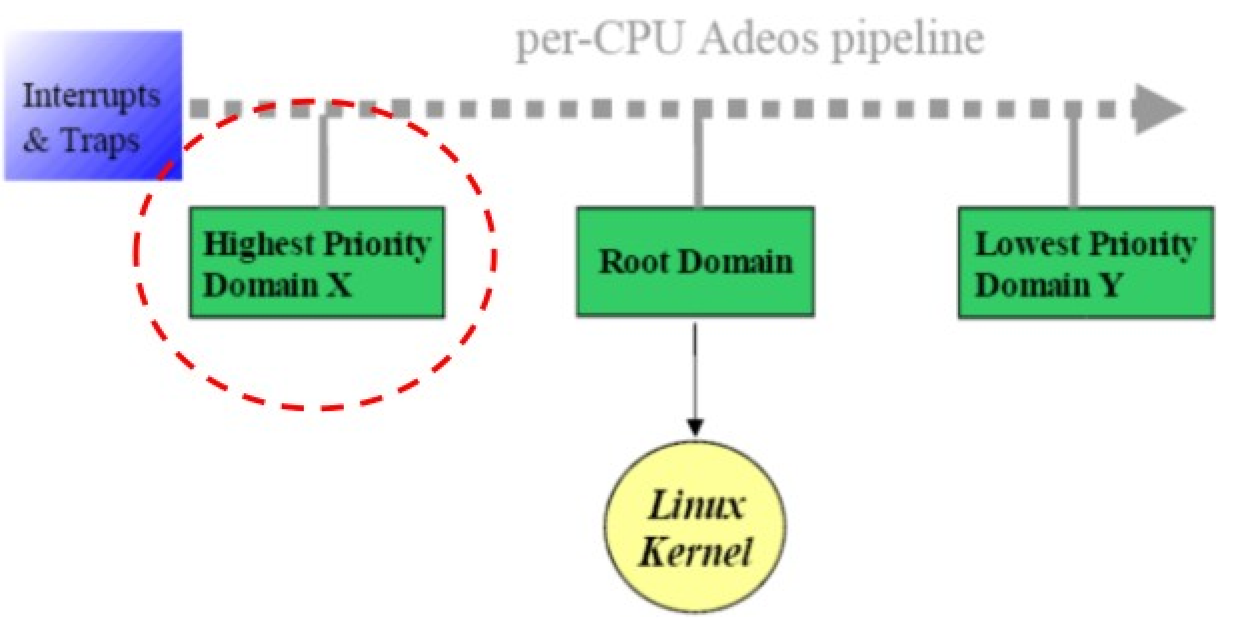
i-pipe: Optimistic protection scheme

- If a real time domain (like Xenomai) has higher priority it is the first in the pipeline
  - It will receive interrupt notification first without delay (or at least with predictable latency)
  - Then it can be decided if interrupts are propagated to low priority domains (like Linux) or not



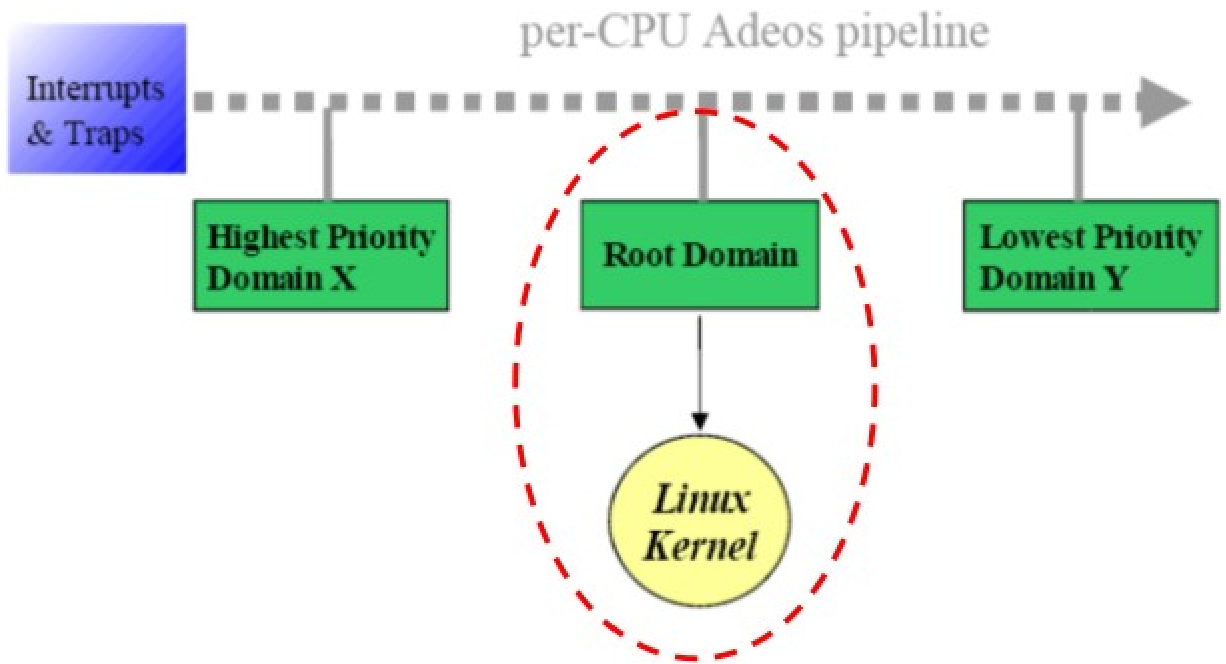
Interrupt pipeline (1)

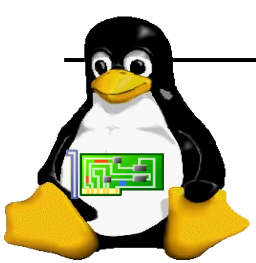
- The high priority domain is at the beginning of the pipeline, so events are delivered first to it
- This pipeline is referred as interrupt pipeline or I-pipe
- There is a pipeline for each CPU



Interrupt pipeline (2)

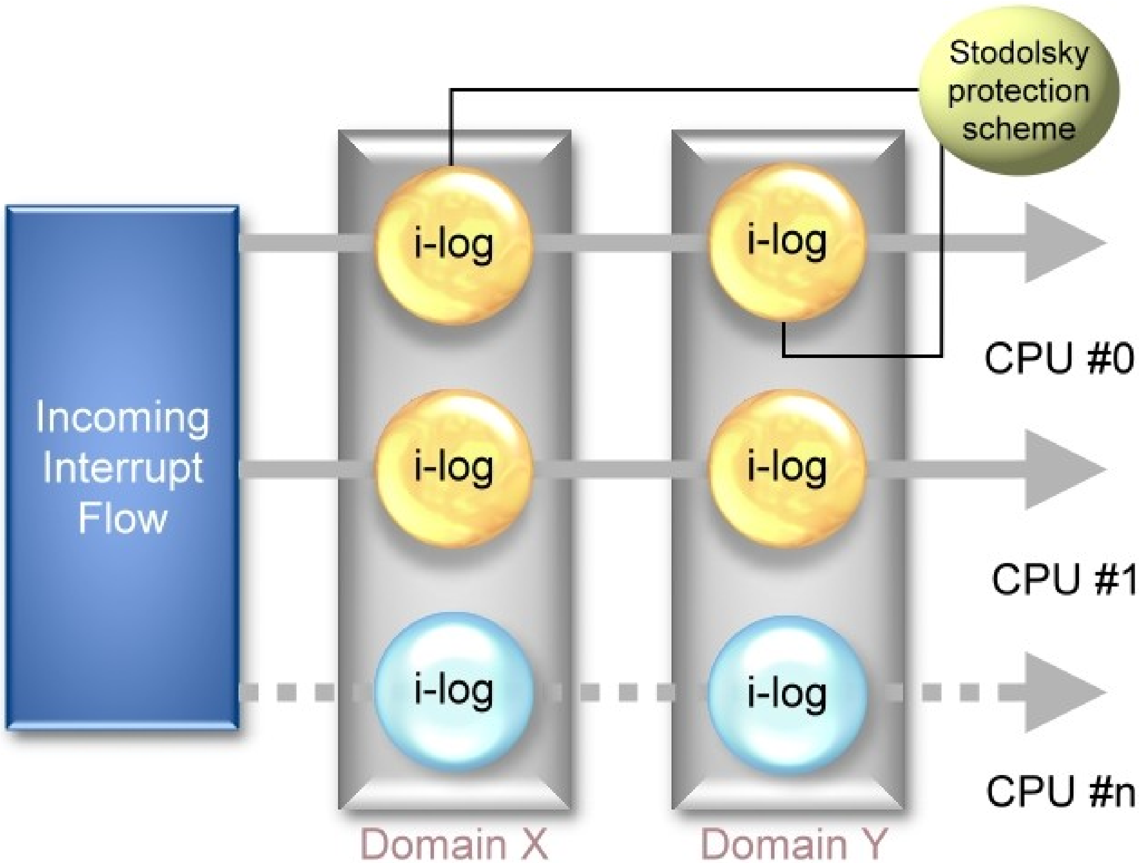
- The Linux domain is always the root domain, whatever is its position in the pipeline
- Other domains are started by the root domain
- Linux starts and loads the kernel modules that implement other domains

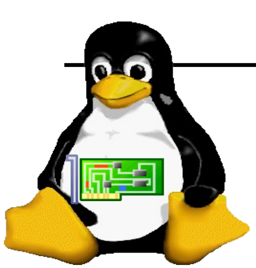




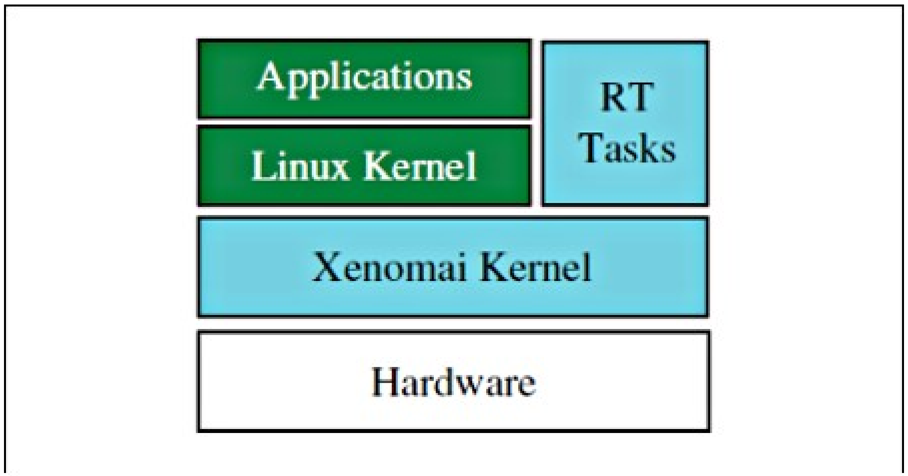
virtualized interrupts disabling

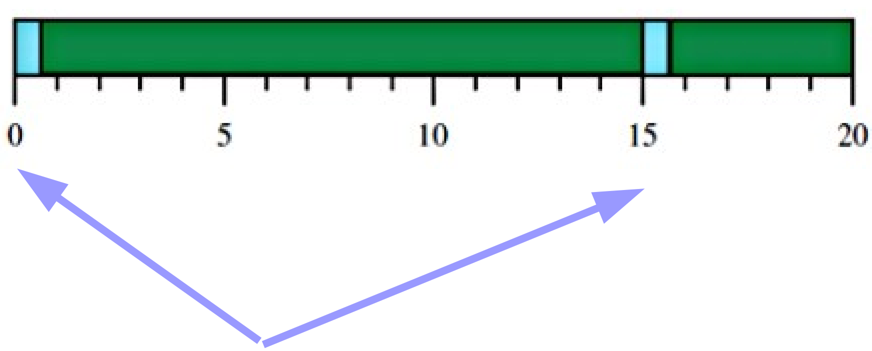
- Each domain may be “stalled”, meaning that it does not accept interrupts.
- Hardware interrupts are not disabled however (except for the domain leading the pipeline), instead the interrupts received during that time are logged and replayed when the domain is unstalled.

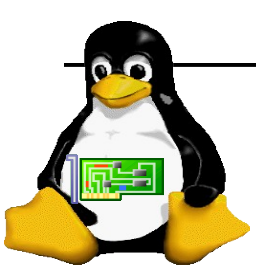




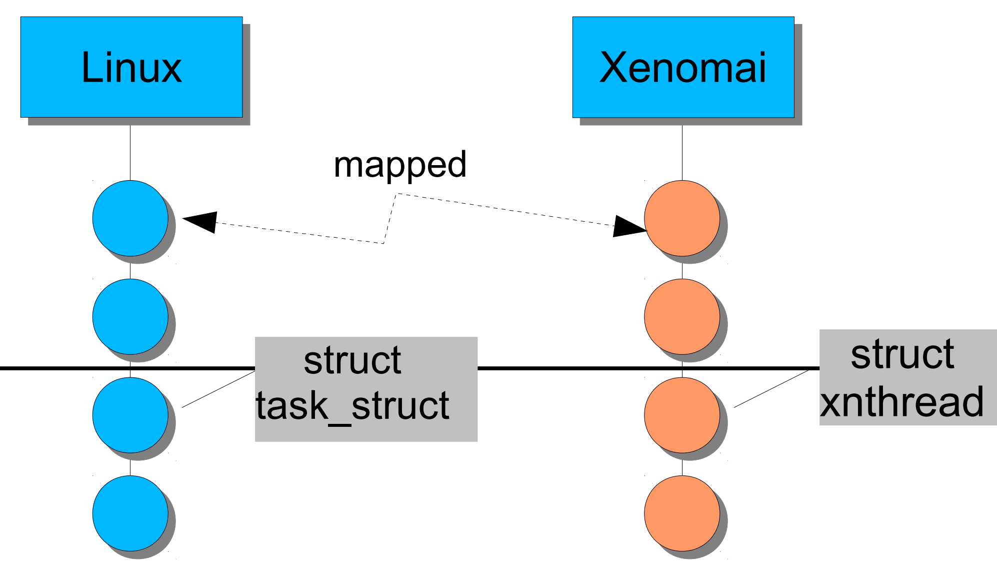
Real-Time Scheduler







- Xenomai extends the Linux kernel and is integrated as part of OS.
- A task with a period = 15 us, shown in light blue.
- While this real-time task is not being executed, Xenomai invokes the regular Linux scheduler which executes tasks as normal, shown in dark green.



Problems about Xenomai 2

- Large Linux modifications are required to enable ipipe

  (diffstat output)

  ```
  ksrc/arch/arm/patches/ipipe-core-3.14.17-arm-4.patch
  271 files changed, 14218 insertions(+), 625 deletions(-)
  ```

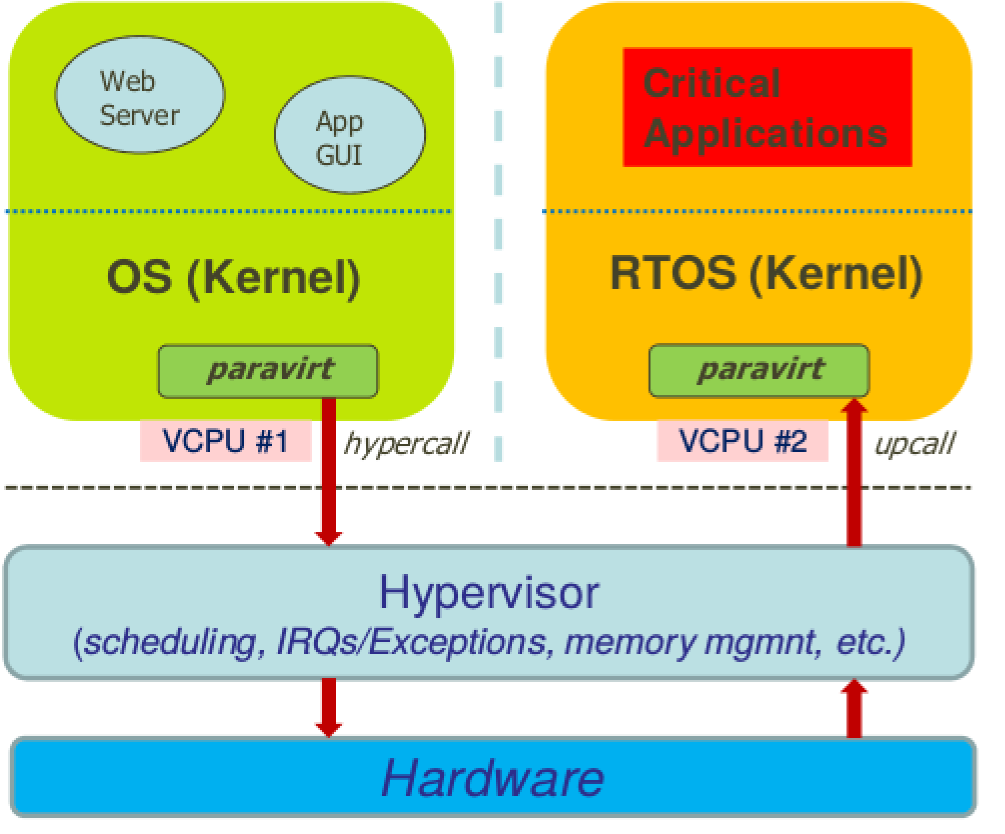
- Maintenance and incompatibility issues
  - POSIX skin
- Xenomai 3 is supporting PREEMPT_RT, but the real-time performance is as good as dual-kernel approach

RTMux: Our Real-time Solution

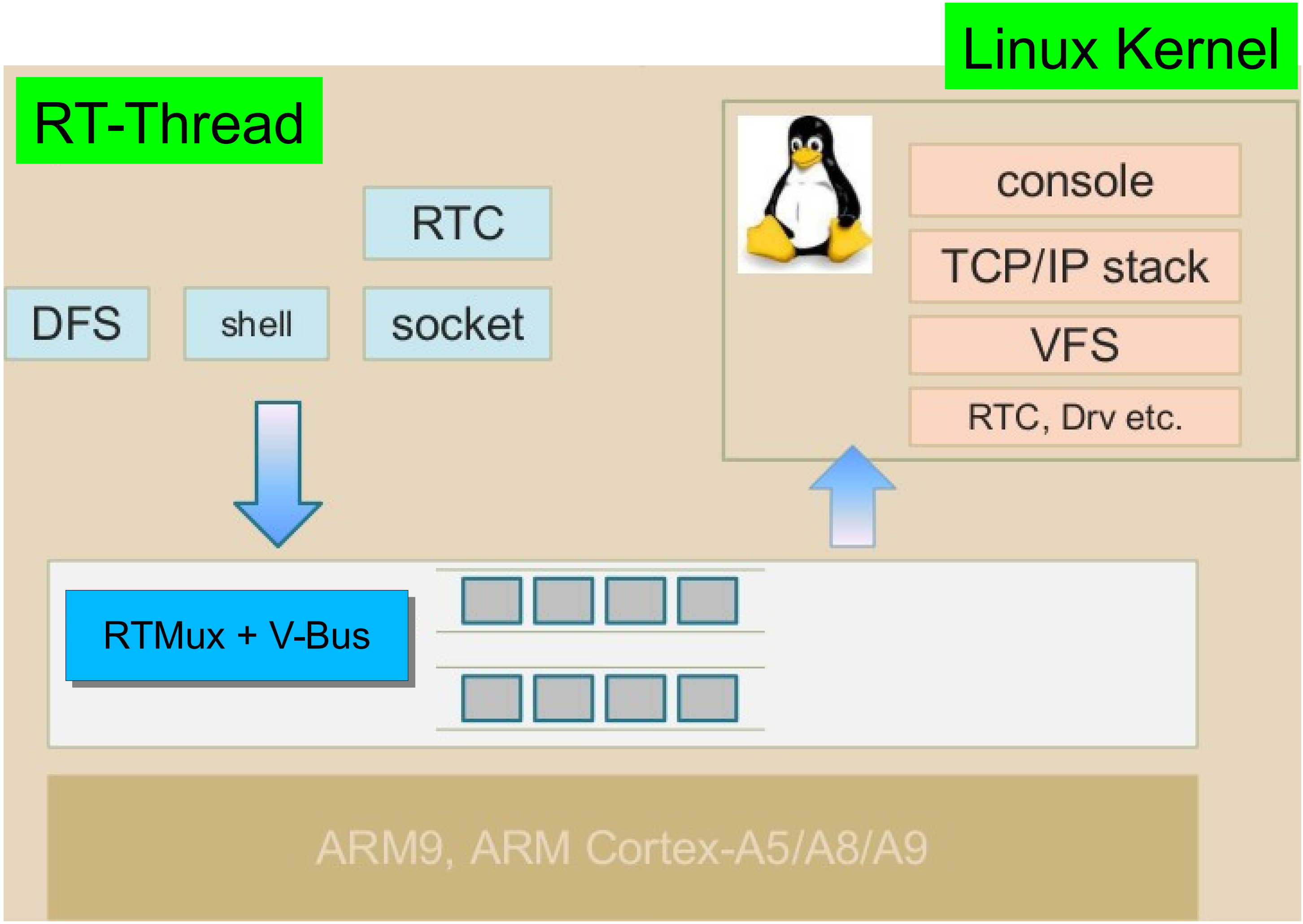
(Lightweight and easier to maintain)

RTMux Goals

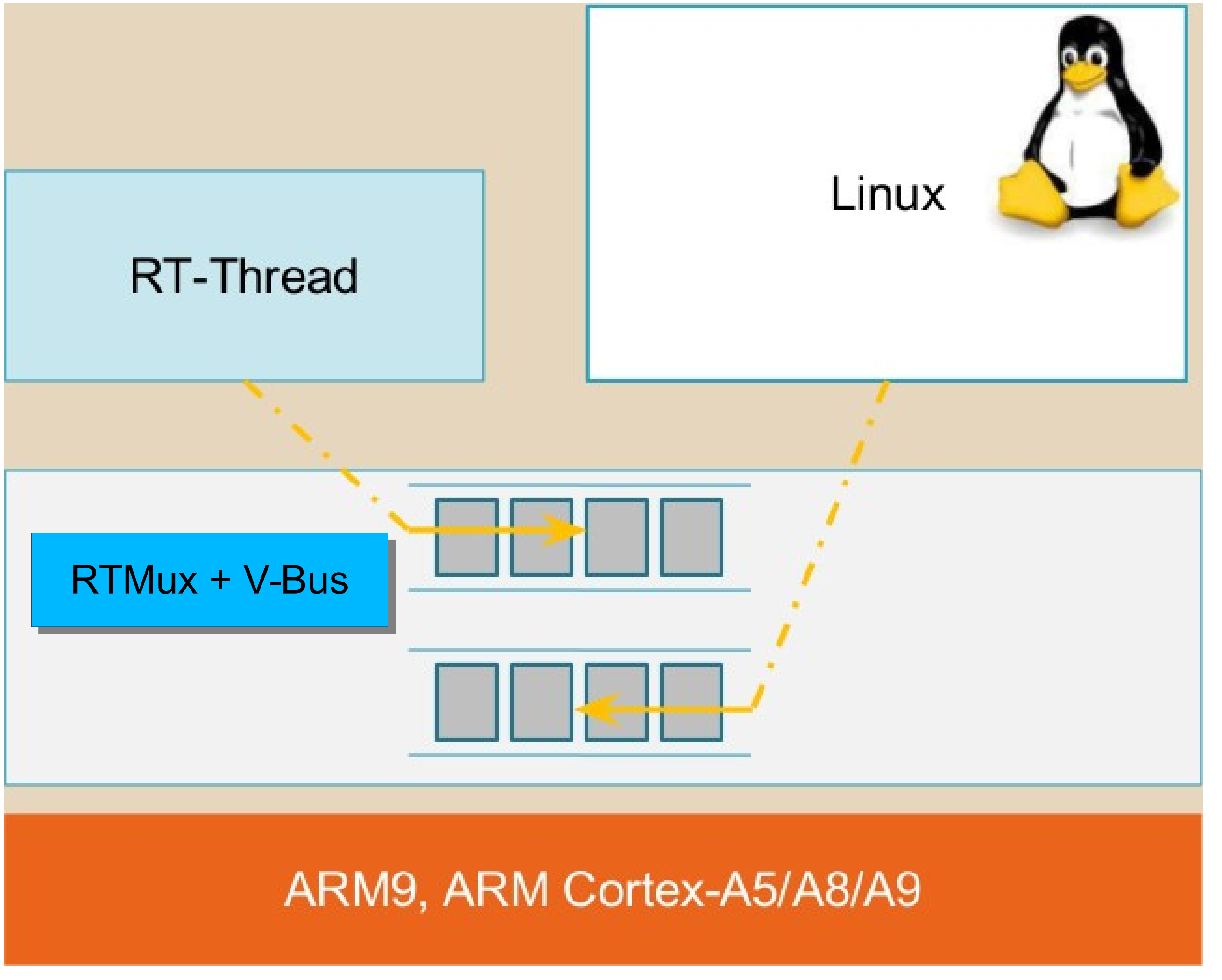
- Utilize the existing Linux mechanisms as possible
  - 400 LoC modifications!
- Lightweight hypervisor for both Linux and RTOS
- Of course, open source: https://github.com/rtmux
  - Hypervisor: GPLv2
  - RT-Thread: GPLv2



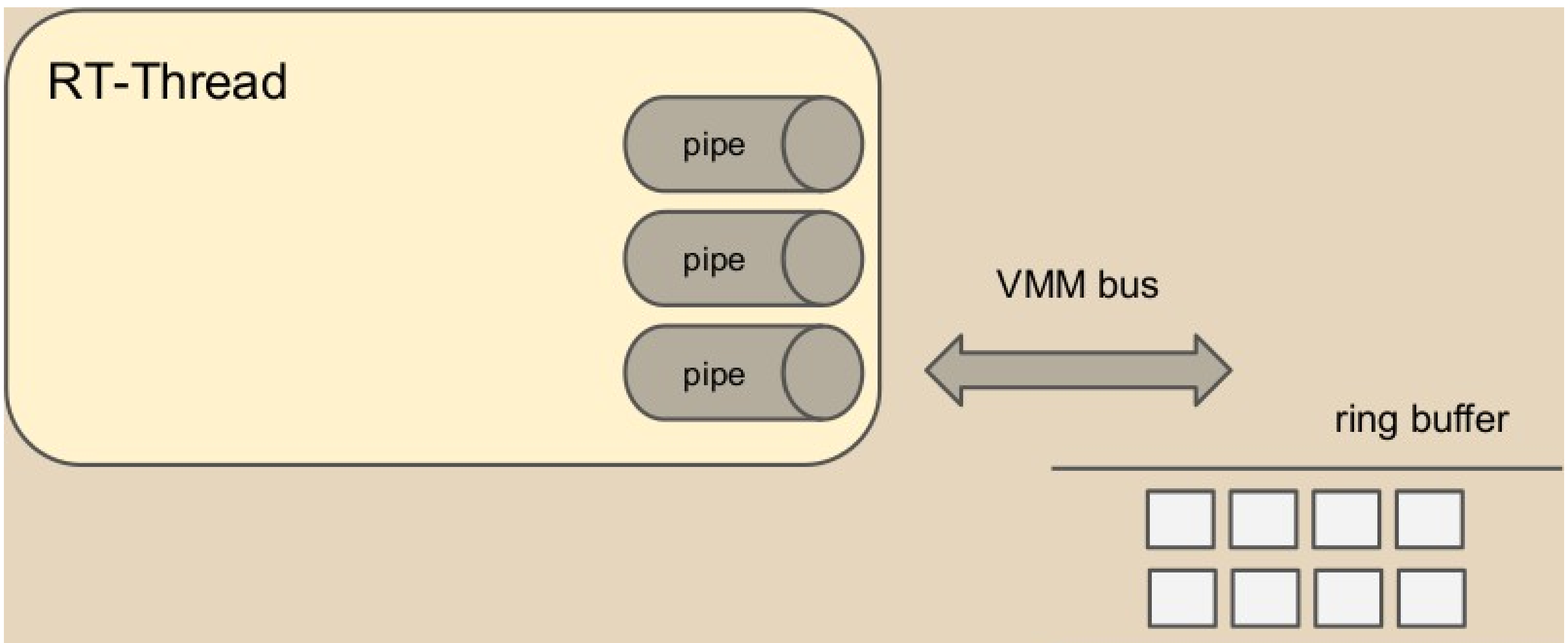
Real-time domain vs. Linux

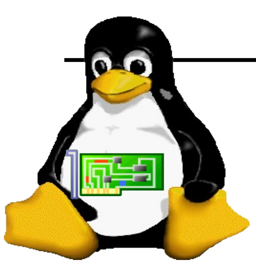


V-Bus: cross Virtual-machine Bus

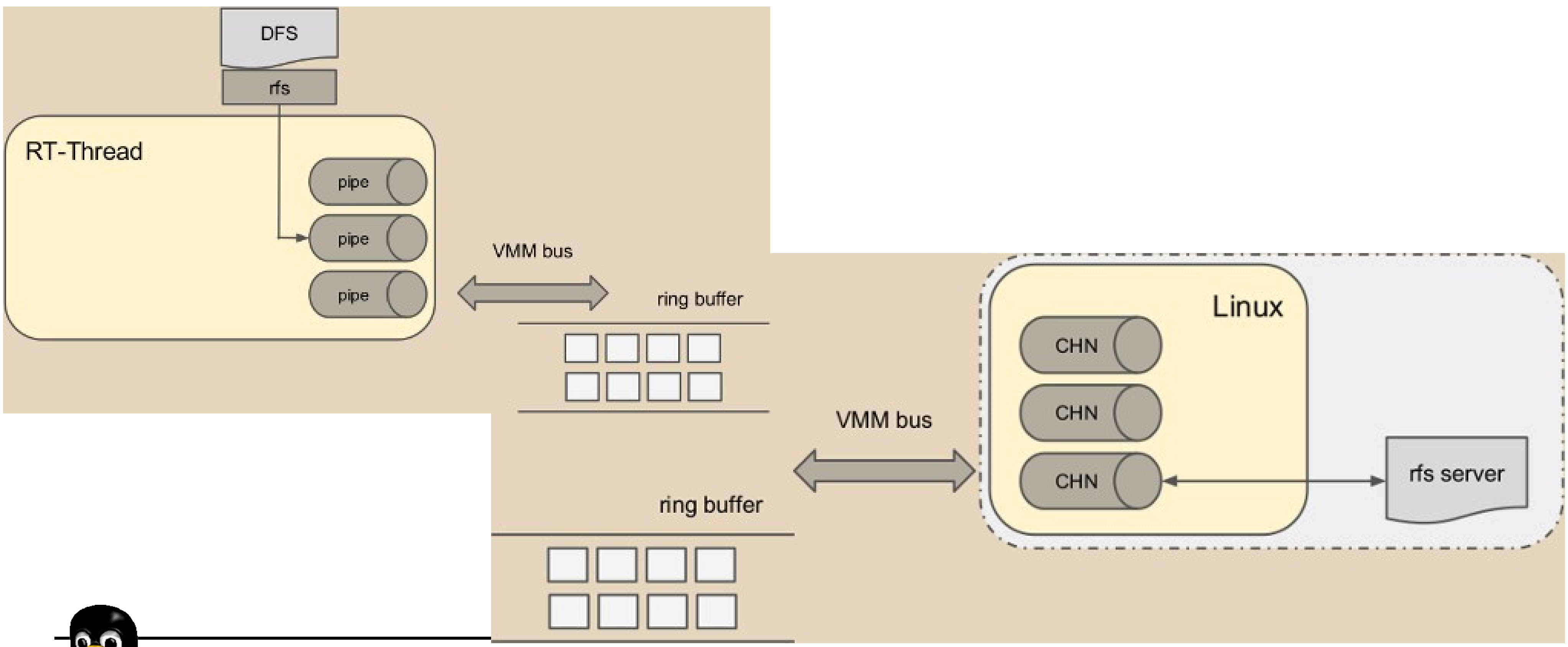


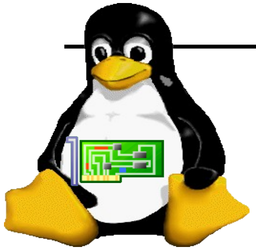
Ring-buffer for V-Bus





Linux communications via V-Bus



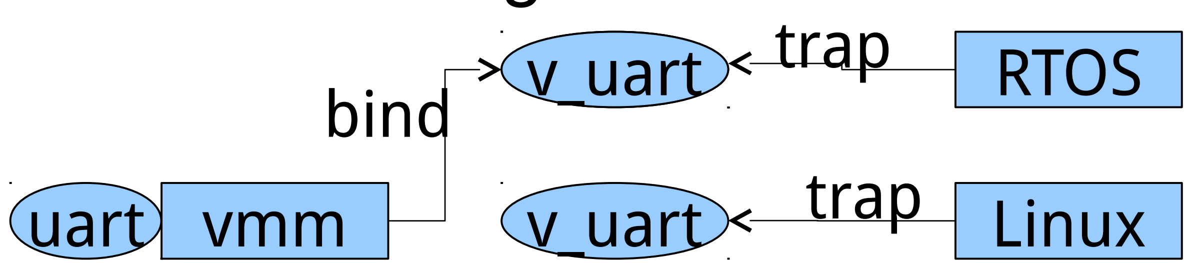


Minimal patch is required to enable RTMux

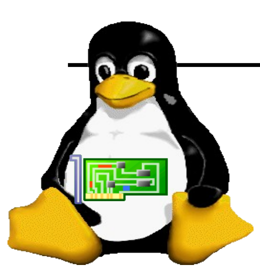
```
$ diffstat rtmux/patches/0001-RTMux.patch
 Kconfig                      |    1
 Makefile                     |    1
 common/gic.c                 |   67 +++++++++++++++++++++++
 include/asm/assembler.h      |    8 ++
 include/asm/domain.h         |    7 ++
 include/asm/irqflags.h       |   69 ++++++++++++++++--------
 include/asm/mach/map.h       |    5 +
...
 21 files changed, 568 insertions(+), 27 deletions(-)
```

Hardware Support

- ARM Cortex-A8 is supported
  - Verified on **Realview Cortex-A8** and **Beaglebone Black**
  - No VE (virtualization extension) required
- Virtual IRQ
- Create mappings for VMM, which shares memory regions with Linux
- Since the device is actually a plain memory with its functionalities emulated, the multiplex could be easily implemented as following:



Guest OS runs in pure user-mode, and RTMux applies the domain field in the page table to emulate the privilege level for the guest OS.



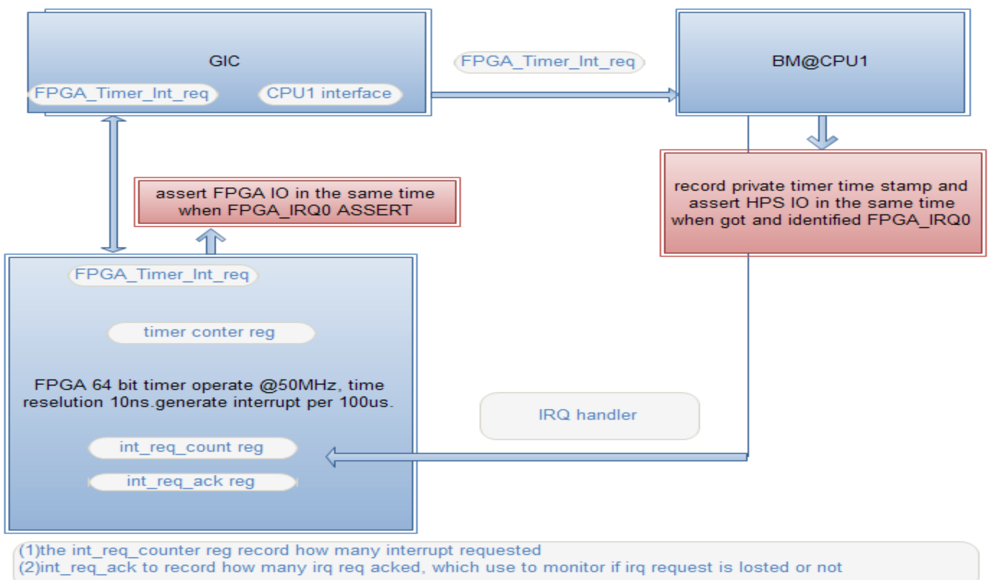
Reference Hardware: Beaglebone Black

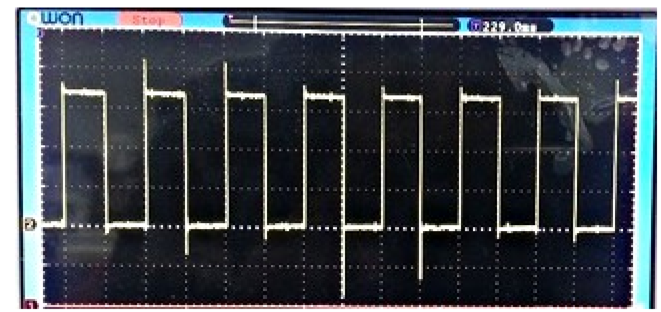
- 1GHz TI Sitara ARM Cortex-A8 processor
- 512MB DDR3L 400MHz memory
- 2 x 46 pin expansion headers for GPIO, SPI, I2C, AIN, Serial, CAN
- microHDMI, microSD, miniUSB Client, USB Host, 10/100 Ethernet
- PRU (Programmable Real-time Unit) can access I/O at 200MHz
  - one instruction takes 5ns, be very careful about the timing
  - write code in assembly

write an integer to the PRU register R30 which takes one instruction (5ns), do some calculations and checks and repeat the write instruction. The data are immediately (within 5ns) available at the output pins and get converted into an analog signal.

Interrupt Latency and Jitter Test

- Background
  - Measure RT interrupt latency while Linux domain is running vision programs.
- Approach





Procedure:

1. MCU generates IRQ request per 100us(10K/s).
2. Assert MCU IO in the same time when IRQ generated.
3. ARM identifies IRQ request and send ack to MCU. Assert IO in the same time
4. Totally, send 100K times IRQ

- Result
  - Max/Average interrupt latency: 3.567us / 582ns (no load)
  - Max/Average interrupt latency: 5.191us / 806ns (normal load)

Reference Results with Xenomai

- User-mode latency

```
== Sampling period: 1000 us
 == Test mode: periodic user-mode task

 RTT|  00:00:01  (periodic user-mode task, 1000 us period, priority 99)

 RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst

 RTD|      8.791|      8.999|     22.416|       0|     0|      6.874|     28.333
```

- Kernl-mode latency

```
 RTT|  00:00:00  (in-kernel periodic task, 100 us period, priority 99)

 RTH|-----lat min|-----lat avg|-----lat max|-overrun|----lat best|---lat worst

 RTD|      -0.920|      -0.804|       3.372|       0|      -4.250|       5.167
```

Conclusion

- Linux was not designed as a RTOS
- You can get **soft real-time** with the standard kernel preemption mode. **Most** of the latencies will be reduced, offering better quality, but probably not all of them.
- However, using **hard real-time extensions will not guarantee that your system is hard real-time**.
  - Your system and applications will also have to be designed properly (correct priorities, use of deterministic APIs, allocation of critical resources ahead of time...).
- RTMux demonstrates the ability to isolate the real-time domain from Linux kernel base in minimal changes with simplified partitioning techniques, suitable for power-efficient ARM cores.

Reference

- Soft, Hard and Hard Real Time Approaches with Linux, Gilad Ben-Yossef
- A nice coverage of Xenomai (Philippe Gerum) and the RT patch (Steven Rostedt):http://oreilly.com/catalog/9780596529680/
- Real-time Linux, Insop Song
- Understanding the Latest Open-Source Implementations of Real-Time Linux for Embedded Processors, Michael Roeder