Using real-time with Linux

Pierre Ficheux (pierre.ficheux@smile.fr)

September 2021

Linux and real-time

Linux and real-time

- Linux a free kernel based on POSIX standard
- Covered by GPLv2 license
- GNU/Linux is a family of OS based on Linux kernel
  - Standard distributions (Debian, Ubuntu, Fedora, etc.)
  - Produced by a “build system” (Yocto, Buildroot)
- Some “ready to use” RT distributions (RHEL for RT)
- Time-sharing scheduling (SCHED_OTHER)
  - All processes run with same priority (0)
  - One can change the process “priority” with the `nice()` system call (or the `nice` command) from +19 to -20
- SCHED_FIFO/RR support (with “bad” results)

Testing RT on Linux

- Procedure:
  - Starting a periodic task
  - Comparing measured deadline with theoretical one
  - Difference is called “jitter”
  - Don’t forget to test system with a heavy load !
- Tools:
  - Periodic task → `cyclictest`, `latency` (Xenomai)
  - GPIO
  - System load → “flood ping”, `hackbench`, `stress`, `dohell`
  - Measures and results → Oscilloscope, Gnuplot

Linux (UNIX) principles

- Initially based on processes (not threads)
- First threads implementation available in 2.6 kernel
- Execution modes of a process
  - User space
  - Kernel space (IO, system calls)
- Complex memory management
  - Hardware addresses (check with `lspci` command)
  - Kernel addresses (MMU), is the area above CONFIG_PAGE_OFFSET
  - Virtual address (user space)
- Lots of RTOS (such as RTEMS) don’t use MMU
  - Faster management
  - No memory protection (user / kernel)

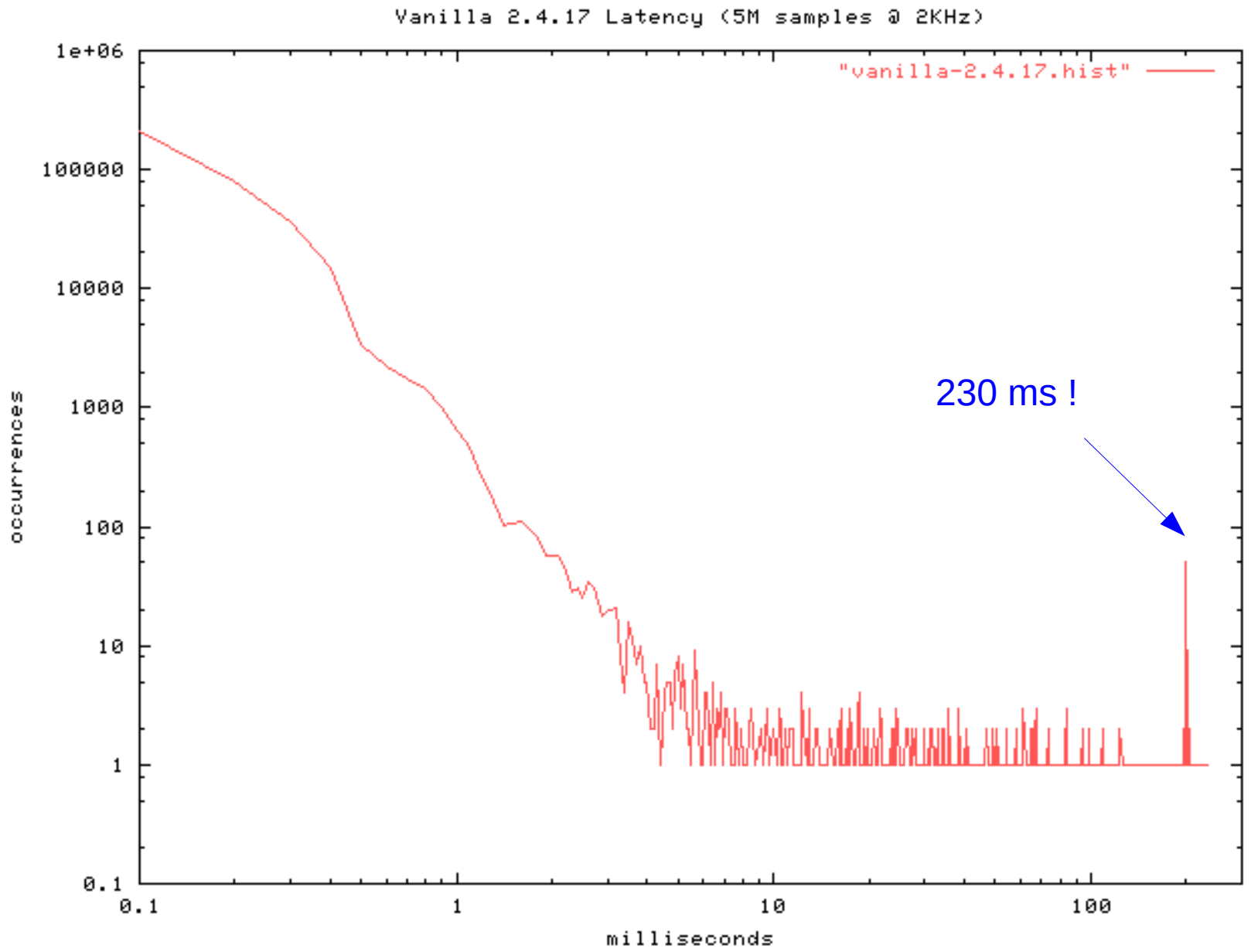
Scheduling issues on Linux

- No “full” preemption in kernel mode
- Process can’t be interrupted in the interrupt handler (top-half)
- Preemption by scheduler:
  - Fixed period with HZ constant (`CONFIG_HZ_xx`) → 100, 250, 1000 Hz
  - Recent kernel uses “tickless” mode (`NO_HZ_IDLE`), i.e. “on demand interrupt” but waking up is *not* deterministic
- RT tasks limitation by “Real Time Scheduler Throttling” in `/proc/sys/kernel`

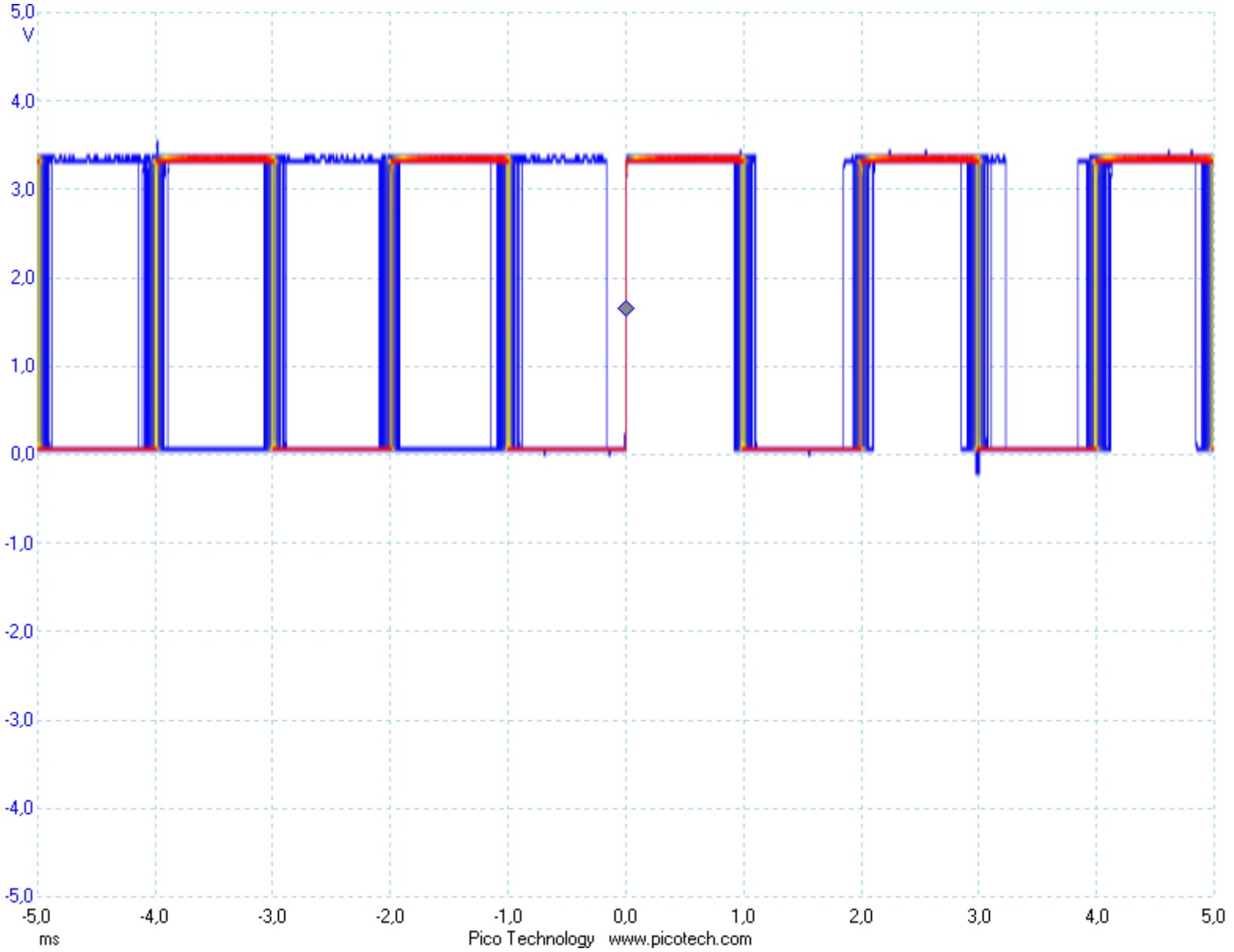
  `sched_rt_period_us` = value for 100% CPU bandwidth (1 s)

  `sched_rt_runtime_us` = RT tasks bandwidth (95 %)

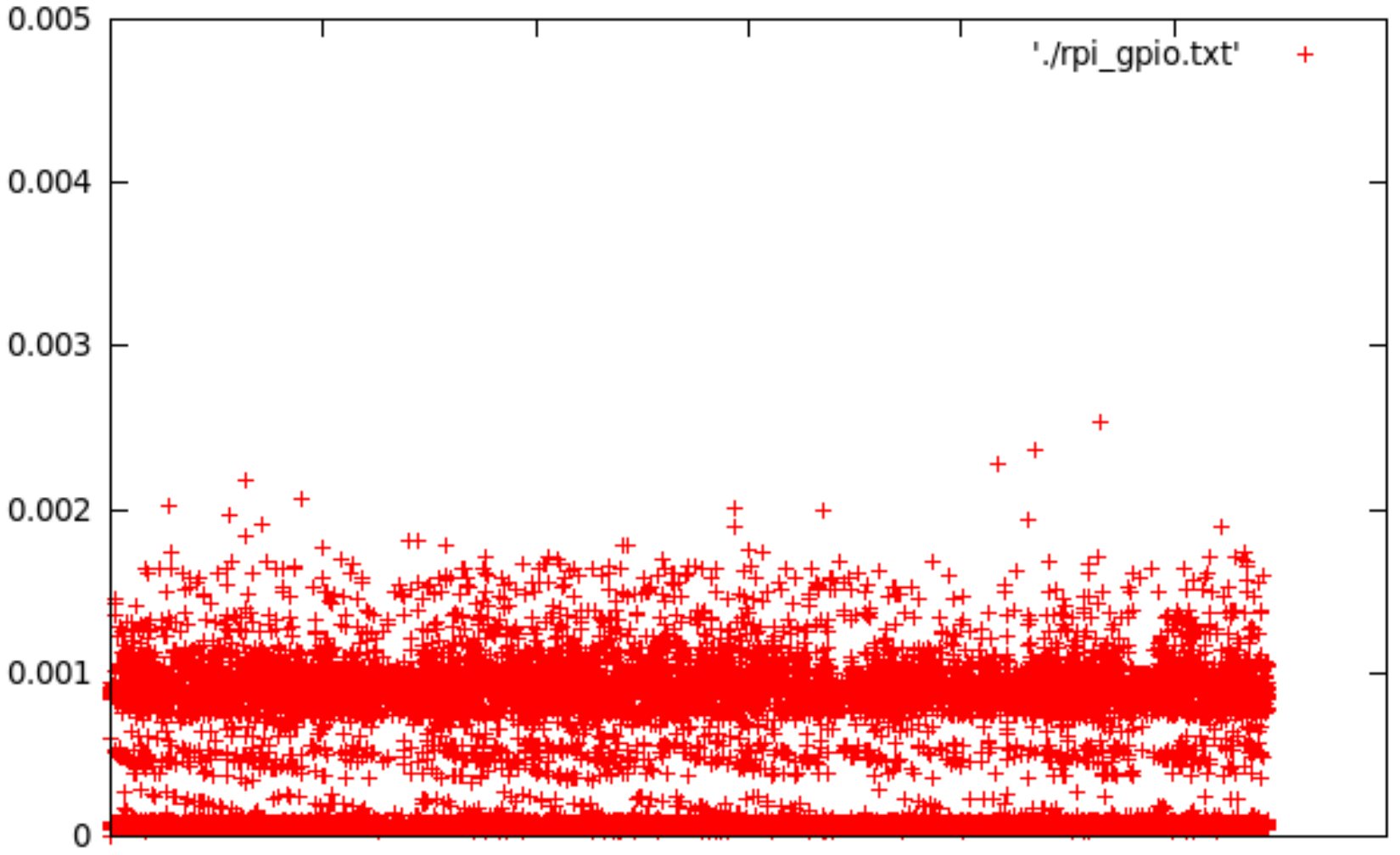
Linux 2.4 for x86 (2002)



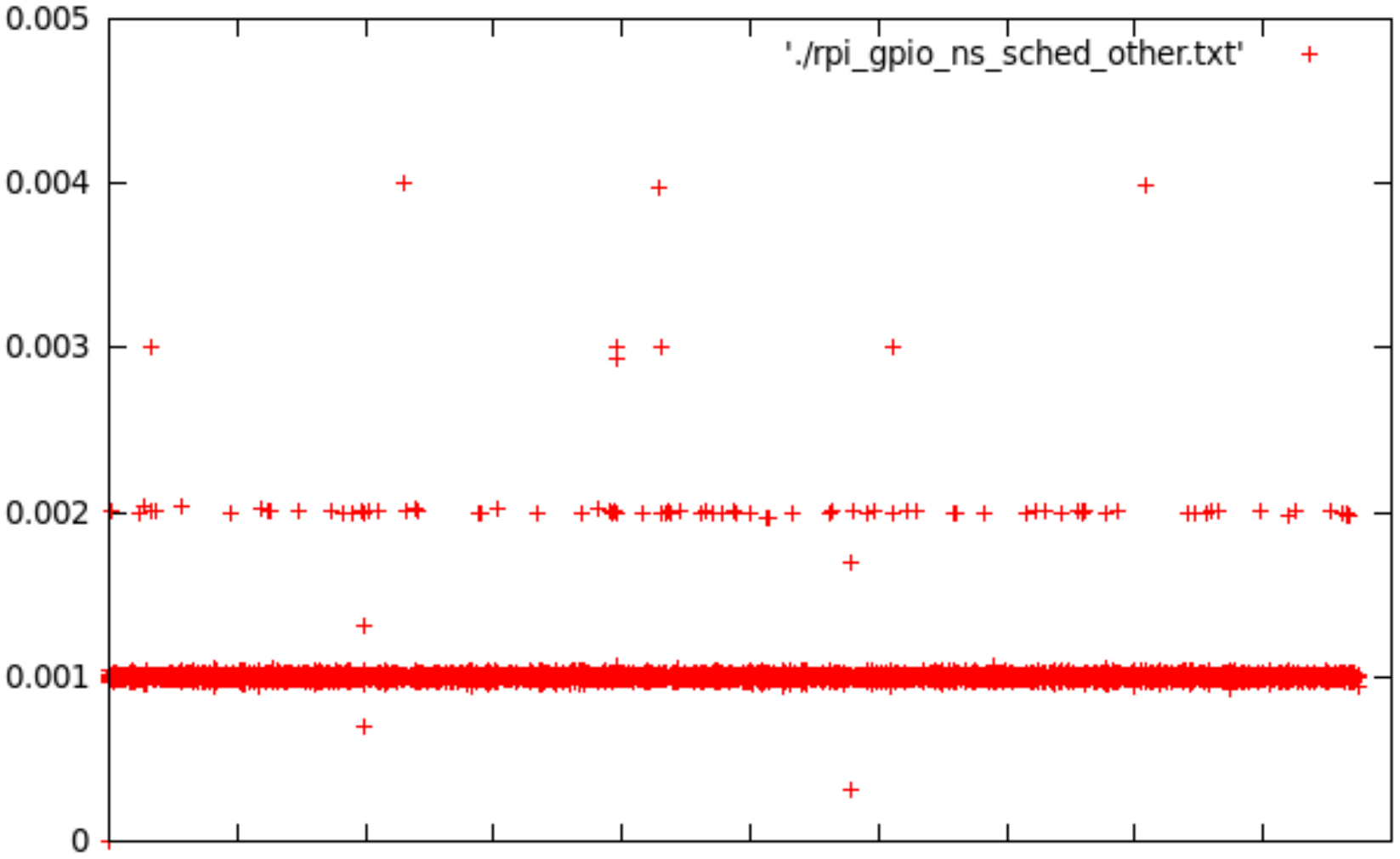
BB Black (Cortex-A8), 1 ms period



Pi B+, 1 ms period (signal based)



Same with `clock_nanosleep()`



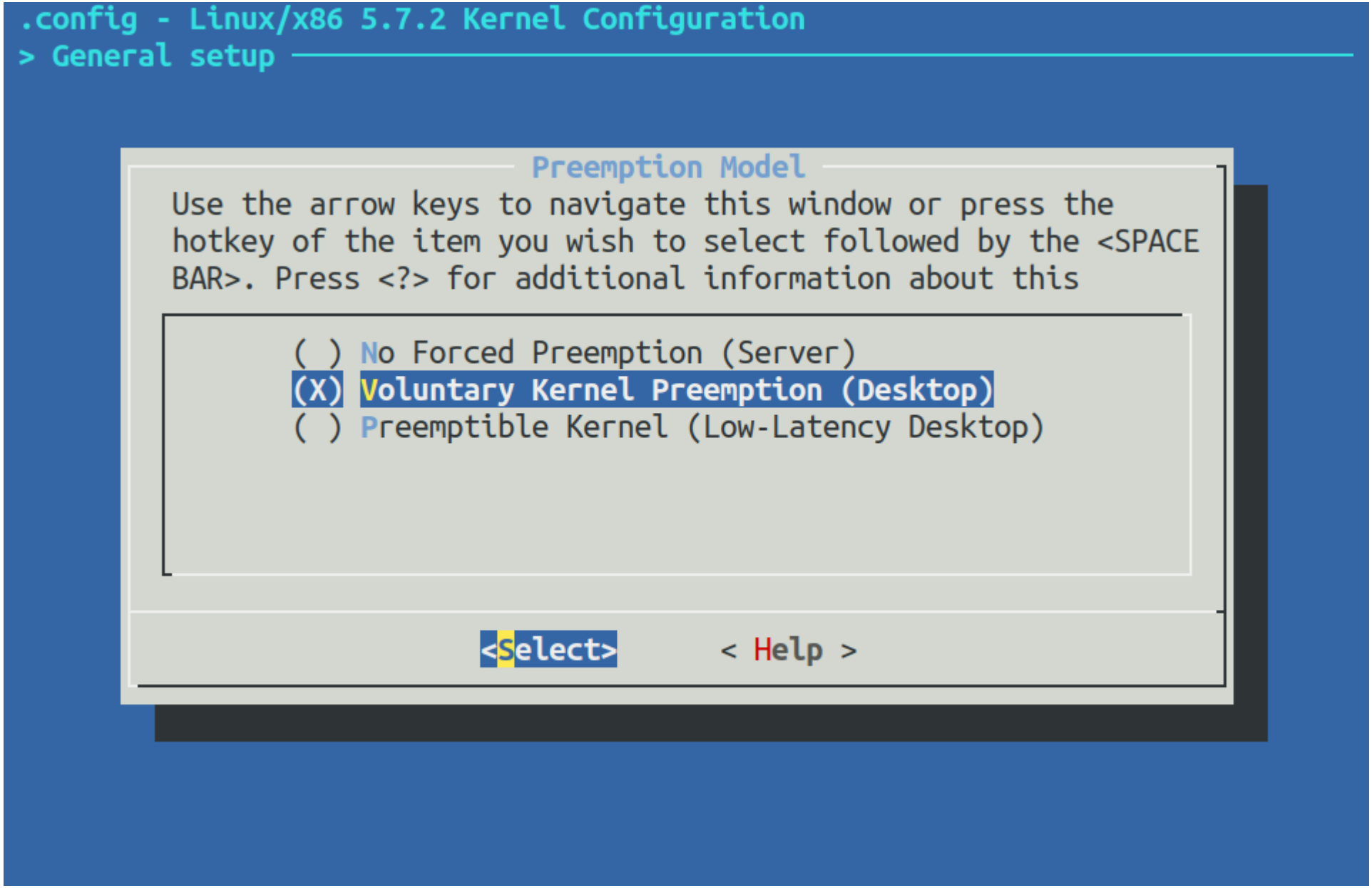
Improving RT kernel performances

- Standard preemption options (mainline but obsolete)
- PREEMPT_RT (Linux foundation, finally integrated in version 5.15)
- Co-kernel approach (RTLinux, RTAI, Xenomai)
- Use SCHED_FIFO/RR policy
- Use `mlockall()` system call to lock memory !
- Use CPU affinity
- Don’t use Linux signal handling (not deterministic) !

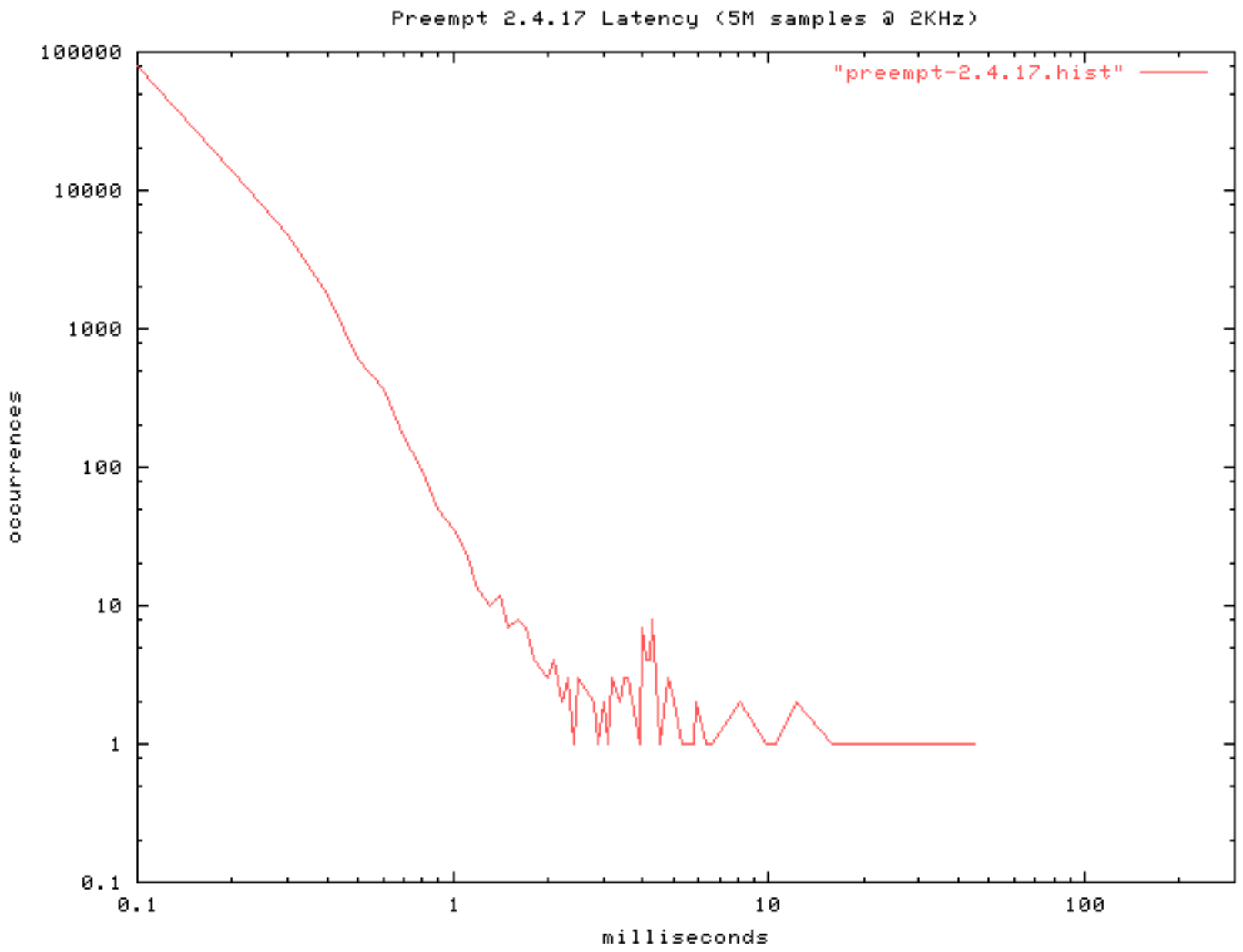
real-time patches

- “preempt-kernel” by Robert Love / MontaVista
- “low-latency” by Andrew Morton
- Patches integrated to mainline since 2.6
- Available for “all” architectures but mostly used on x86
- Preemptive kernel excepting:
  - SMP critical sections
  - Interrupt context (ISR)
- Menu *General Setup / Preemption Model*
  - PREEMPT original preemption model for desktop (based on preempt-kernel patch)
  - PREEMPT_VOLUNTARY “explicit” preemption points (middle point and default value)
- Same programming API (user / kernel)
- Not usable anymore (use PREEMPT_RT !)

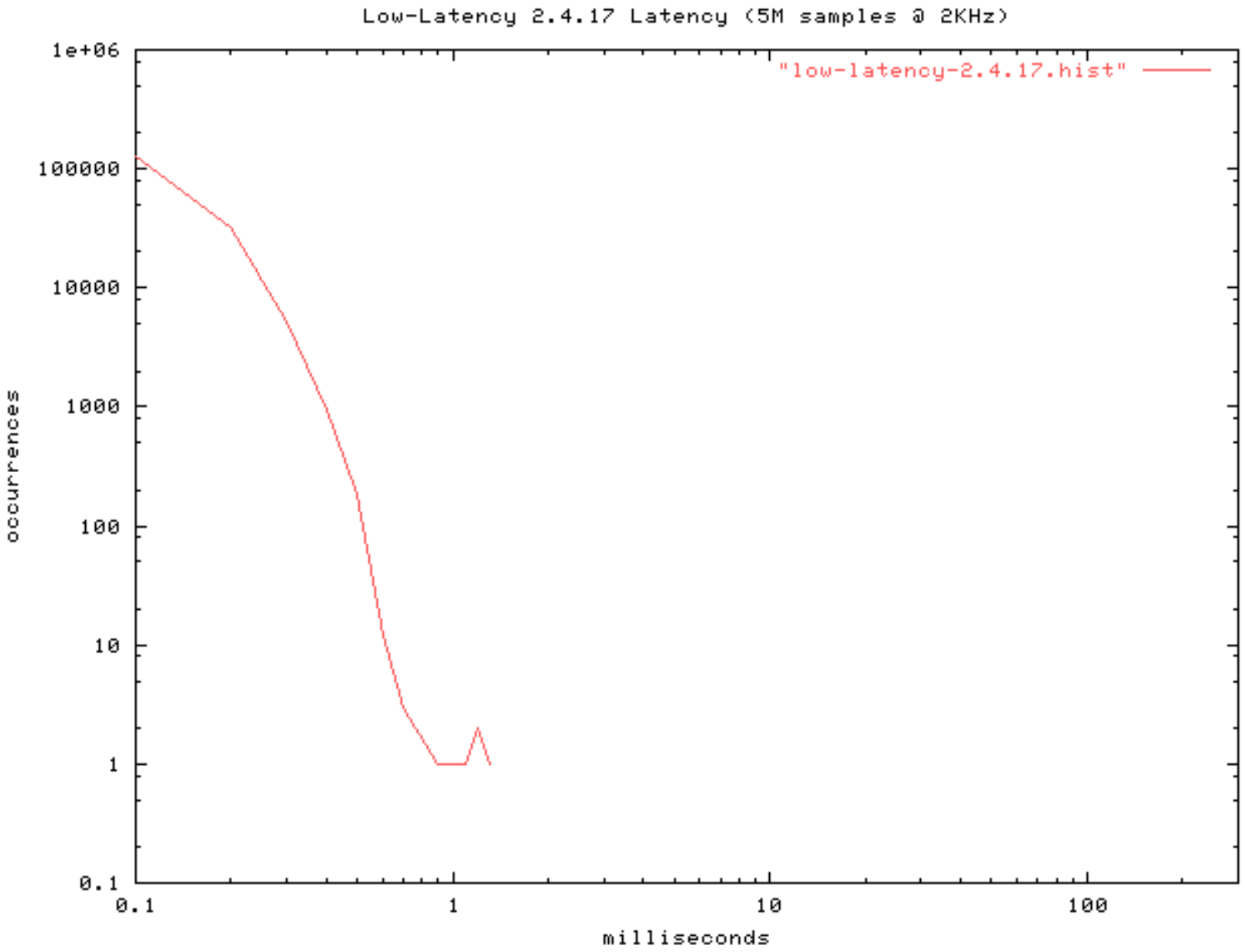
Linux kernel configuration



Preempt-kernel x86 (2002)



Low-latency x86 (2002)



PREEMPT_RT

Real-time patches history

- During autumn 2004 MontaVista, Timesys, Lynuxworks post real-time related patch fragments
- Ingo Molnar (major kernel maintainer) re-implements parts from scratch and posts the real-time preemption patch
- A core team forms
- Kernel Summit 2006 accepts a plan to merge all components into mainline over time

PREEMPT_RT

- Experimental branch for 2.6 kernel
- Started by Ingo Molnar
- First patch for 2.6.9 (2004)
- Maintained by Thomas Gleixner (Linutronix)
- Official Linux foundation project since 2015
- Works on platforms defining ARCH_SUPPORTS_RT
- Published as a patch for mainline kernel or specific branch (BSP)
- Finally merged in 5.15, 31/8/2021 !
- Same programming API as standard kernel

PREEMPT_RT features

- High resolution timers support (hrtimer) + dynamic ticks
- Sleeping spinlocks (preemptible protected regions) based on rt_mutex
- Adds priority inheritance to `spin_lock()` and `mutex_lock()`
- “Threaded interrupt model” → interrupt handling is based on kernel thread (with priority !)

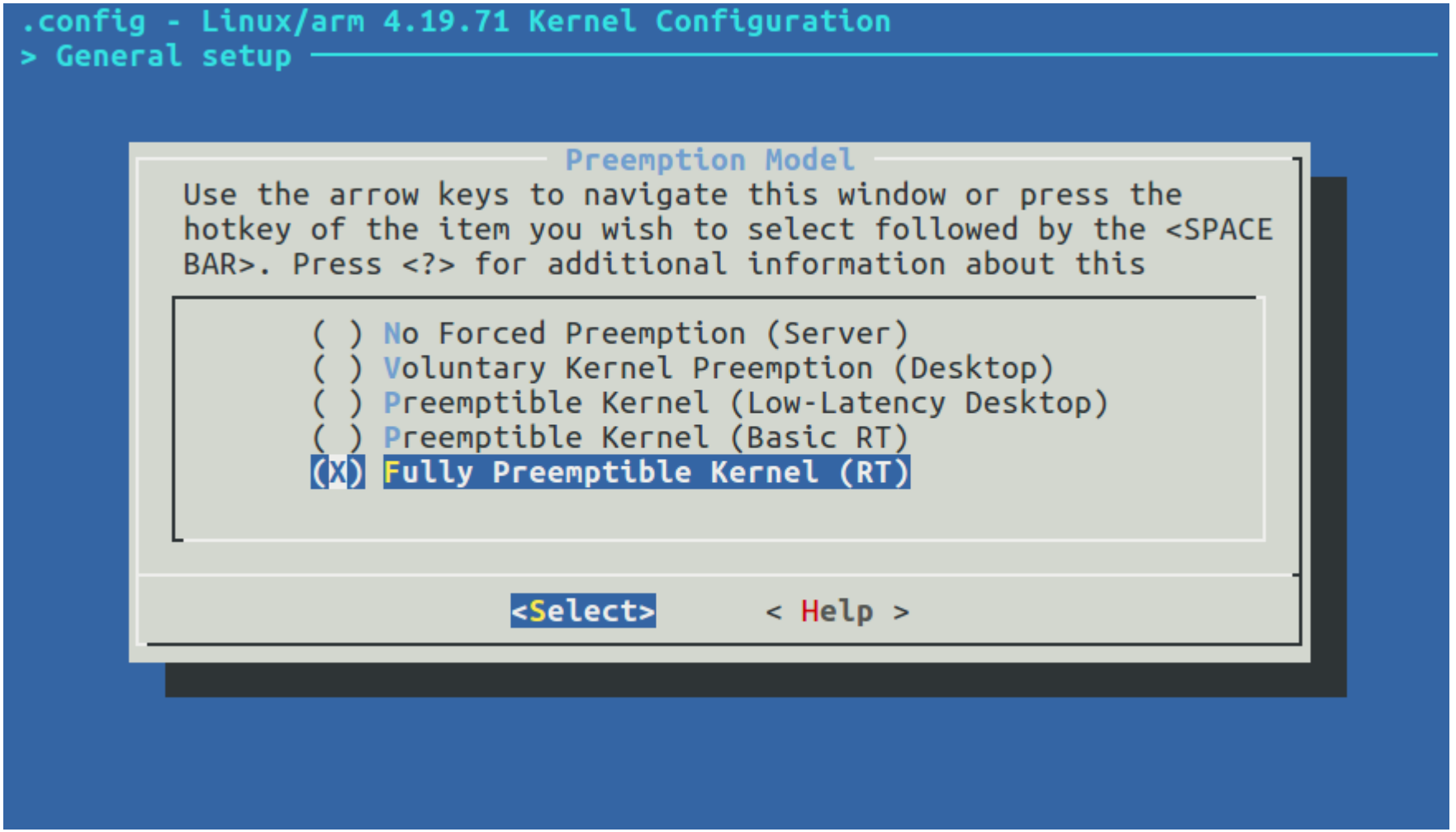
```
69 root          0 SW   [irq/62-dwc_otg]
71 root          0 SW   [irq/62-dwc_otg_]
72 root          0 SW   [irq/62-dwc_otg_]
```

- Still a full featured Linux kernel (Perf, Valgrind, Ftrace, etc.)

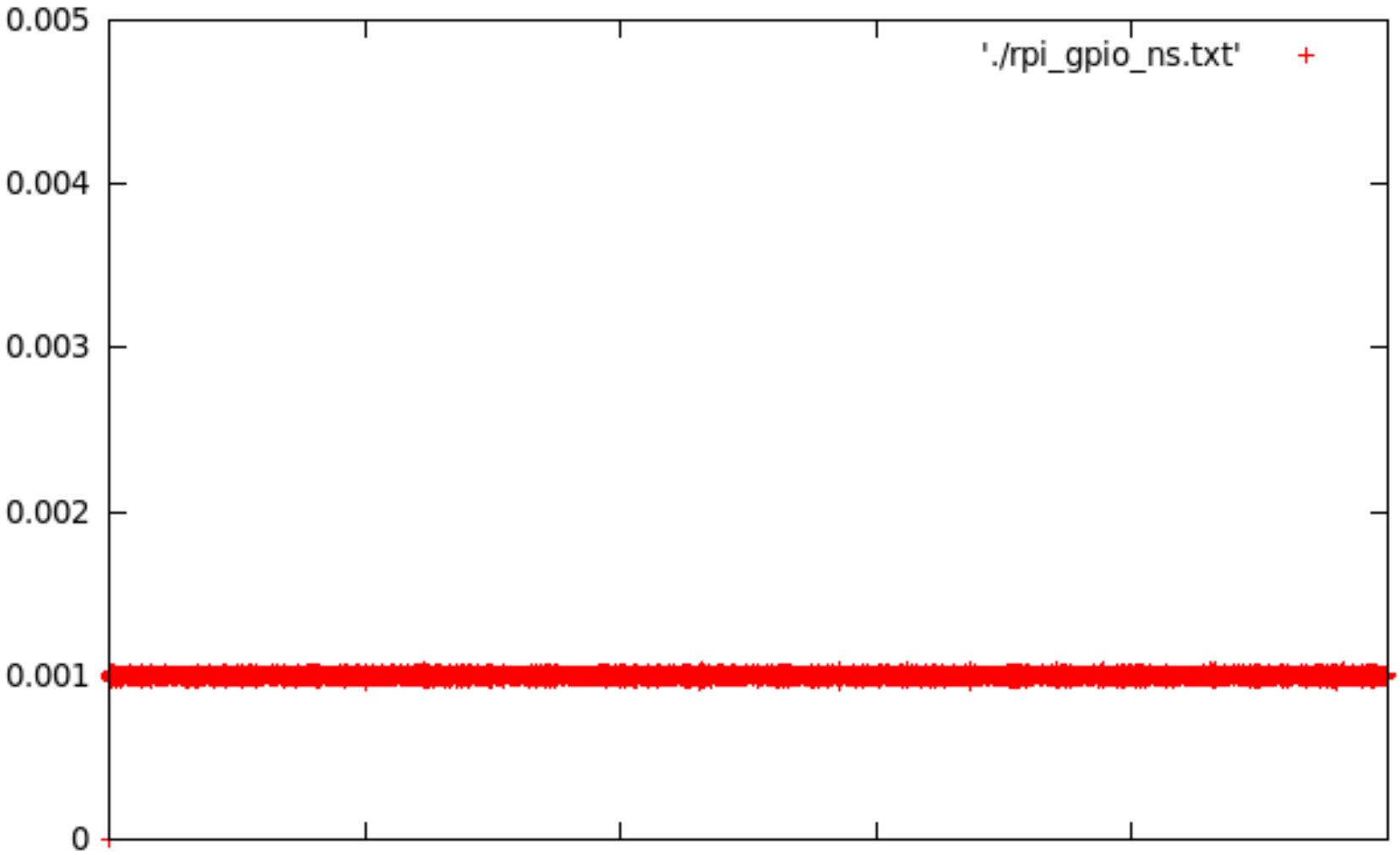
Spinlocks family

- Spinlocks (Linux/UP)
  - IRQ disables on lock, nothing else can interrupt
  - Not RT friendly
- Spinlocks (Linux/SMP)
  - Spinning (busy wait)
  - Not RT friendly
- Sleeping spinlocks (PREEMPT_RT)
  - Sections protected by spinlock may be preempted :-)
  - “Contended” spinlock may sleep (bad for IRQ, use kthread)
- Raw spinlocks (PREEMPT_RT)
  - Disables preemption while held
  - Does NOT sleep
  - Should be a “lightweight operation”

PREEMPT_RT configuration (4.x)



Raspberry Pi + PREEMPT_RT



Testing PREEMPT_RT performances

- PREEMPT_RT based system is easy to build with Buildroot or Yocto
- Use “rt-tests” package
- Use `cyclictest` to create 5 threads (period = 1 ms)

```
# cyclictest -p 99 -t 5 -n
```

- Use `hackbench` to stress the system

```
# hackbench -p -g 20 -l 1000
```

- Test on Raspberry Pi 3
  - jitter with standard kernel = 3 ms
  - jitter with PREEMPT_RT = 100 µs

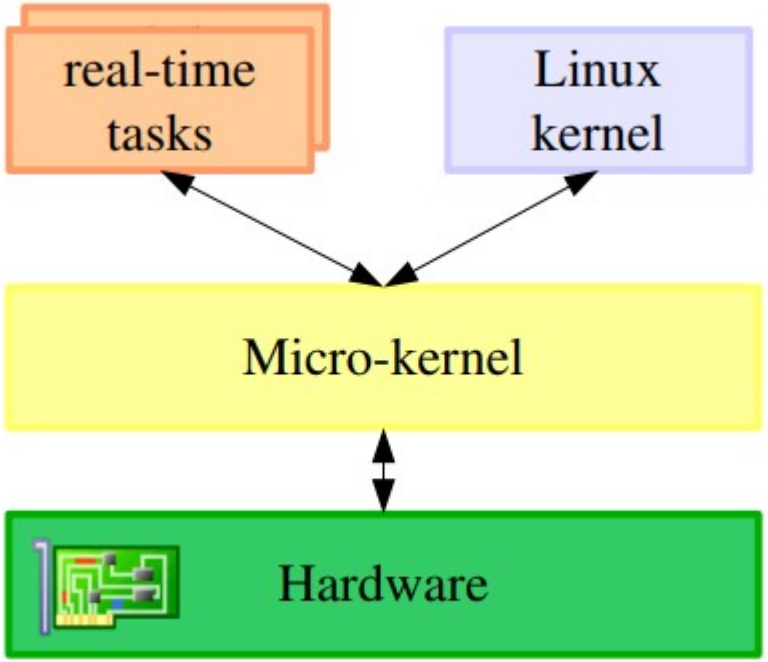
Using a co-kernel (RTLinux & sons)

Linux + co-kernel

- A very different approach !
- Add a dedicated RT kernel to the Linux kernel
  - RT sub-system based on kernel modules
  - IRQ virtualization
- Several models
  - Kernel only (RTLinux)
  - Kernel and (partial) user space (RTAI)
  - Kernel and (full) user space (Xenomai)
- User space support is very important regarding licensing (GPL vs LGPL) !

Co-kernel principles

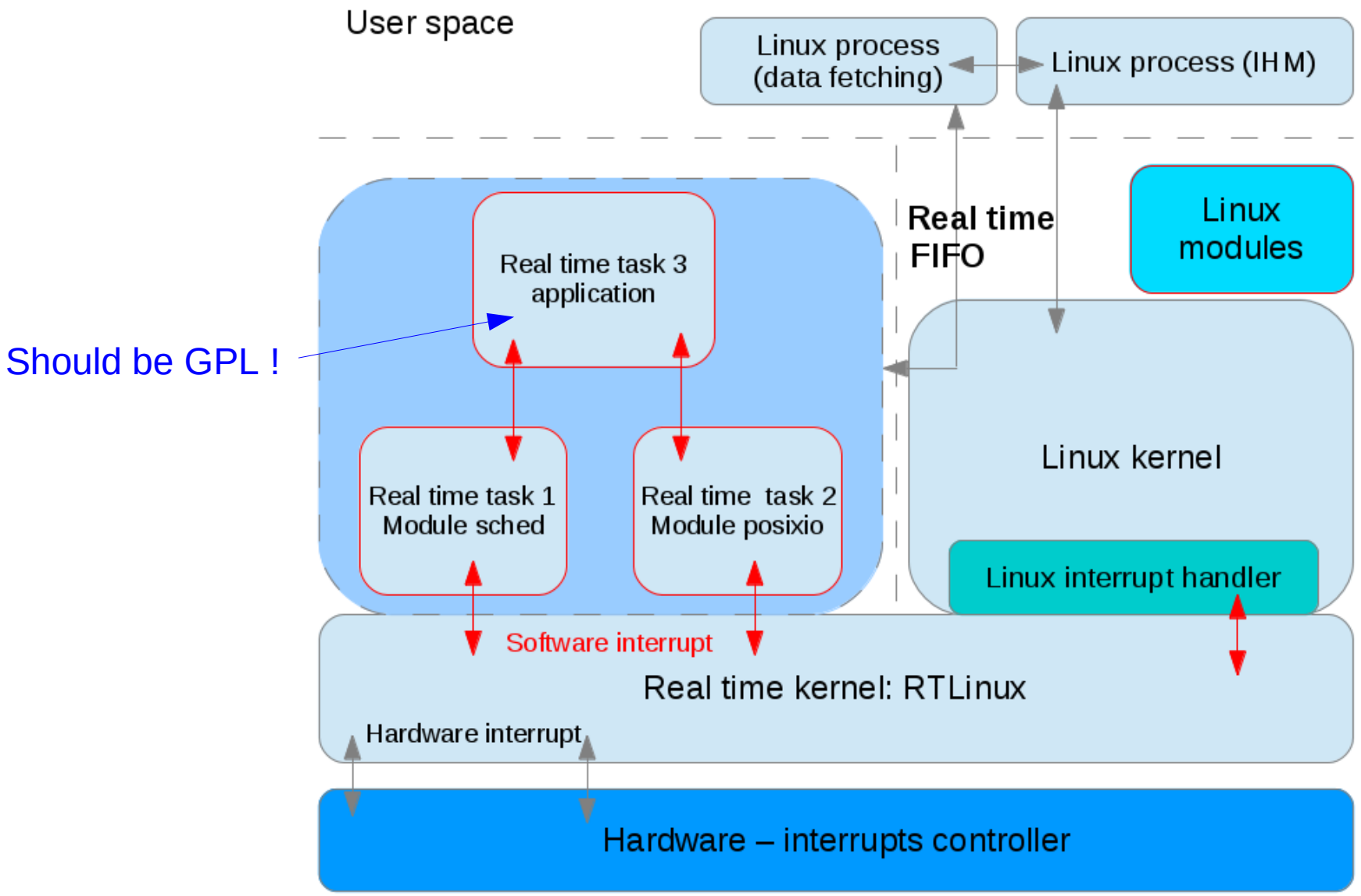
- RT specific scheduler
- Interrupt handling virtualization by a “micro kernel”
  - Kernel should not mask interrupt (CLI/STI on x86)
  - RT interrupts have higher priority
- Linux is an “idle task” for RT kernel (i.e. any Linux task has lower priority than any RT task)



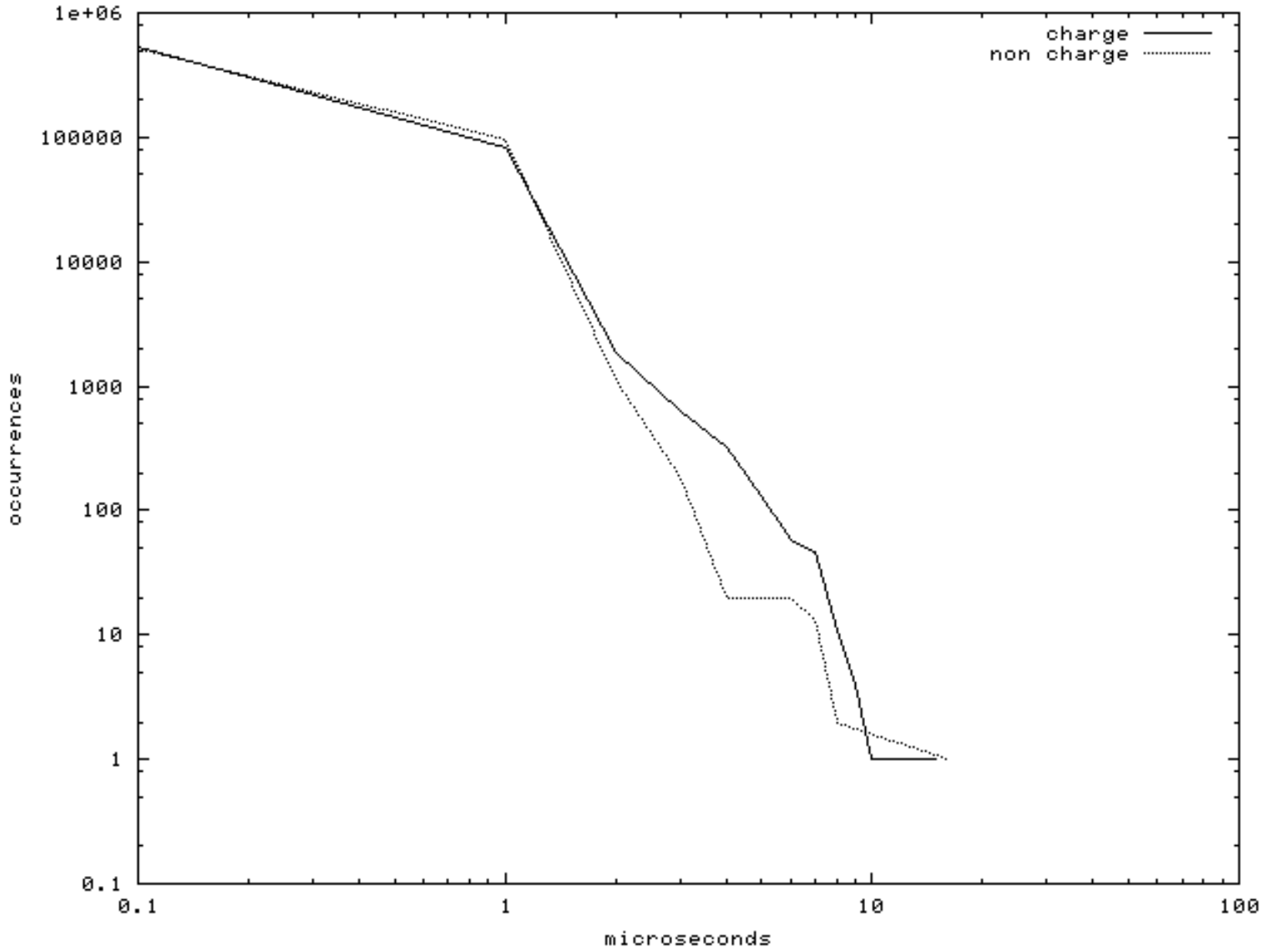
RTLinux

- NMT (New Mexico Institute of Technology) free project created par Victor Yodaiken et Michael Barabanov in 1998
- Licensed under GPL v2 (kernel space)
- Commercial version by FSMLabs
- Patented virtualization technology (5995745)
- Conflict with FSF regarding licensing, agreement with “open patented license”
- RTLinux sold to Wind River in 2007
- Last GPL version (obsolete) retired by Wind River
- Lots of RTLinux users switched to RTAI or Xenomai !

RTLinux architecture



RTLinux/GPL x86 (2002)



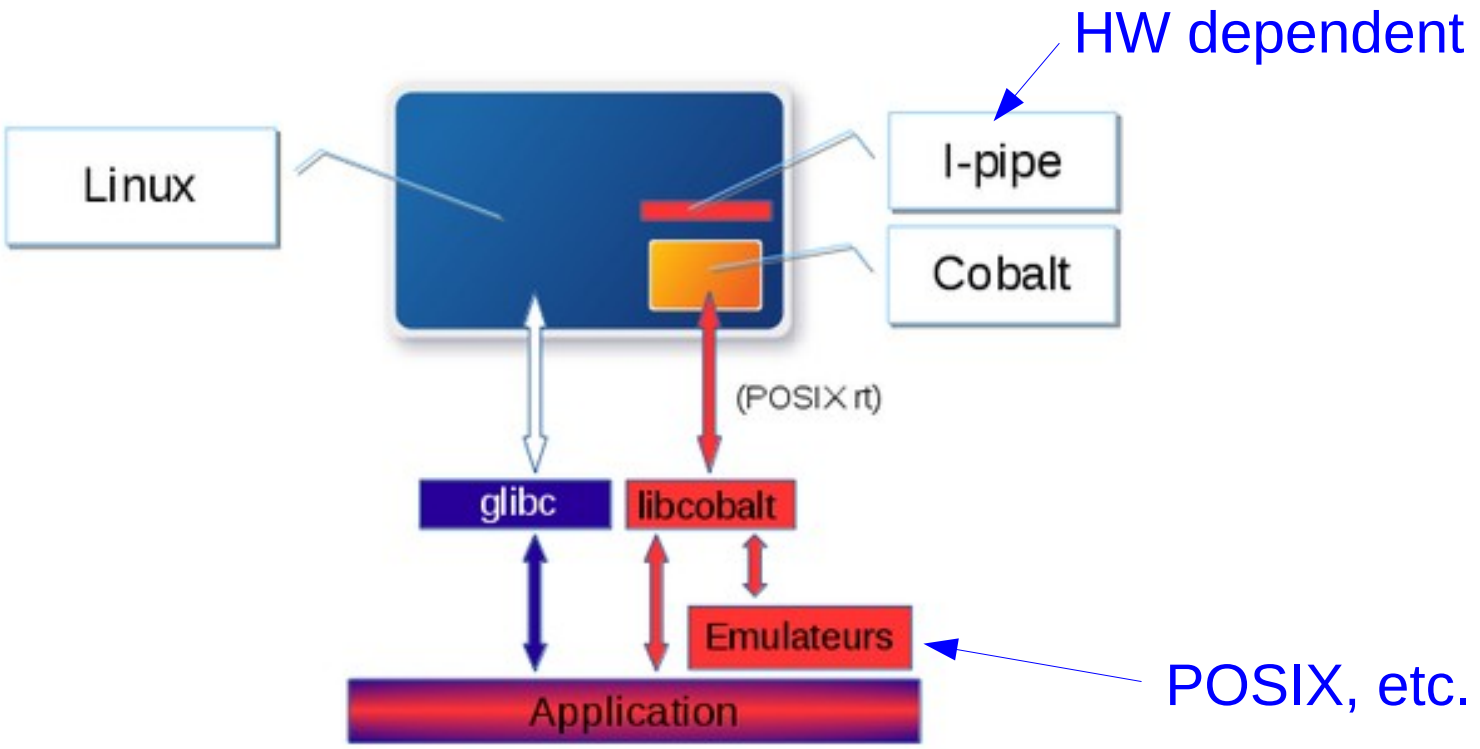
RTAI

- Real Time Application Interface
- RTLinux “fork” created by Paolo Mantegazza from DIAPM (Politecnico Milano)
- Designed for teaching and research (no patent issue)
- Focused on x86
- Introduced LXRT for RT task user space execution
- Collaboration with Xenomai between 2003 and 2005 (RTAI/fusion in 2004)
- Current version is 5.2 (may 2019)

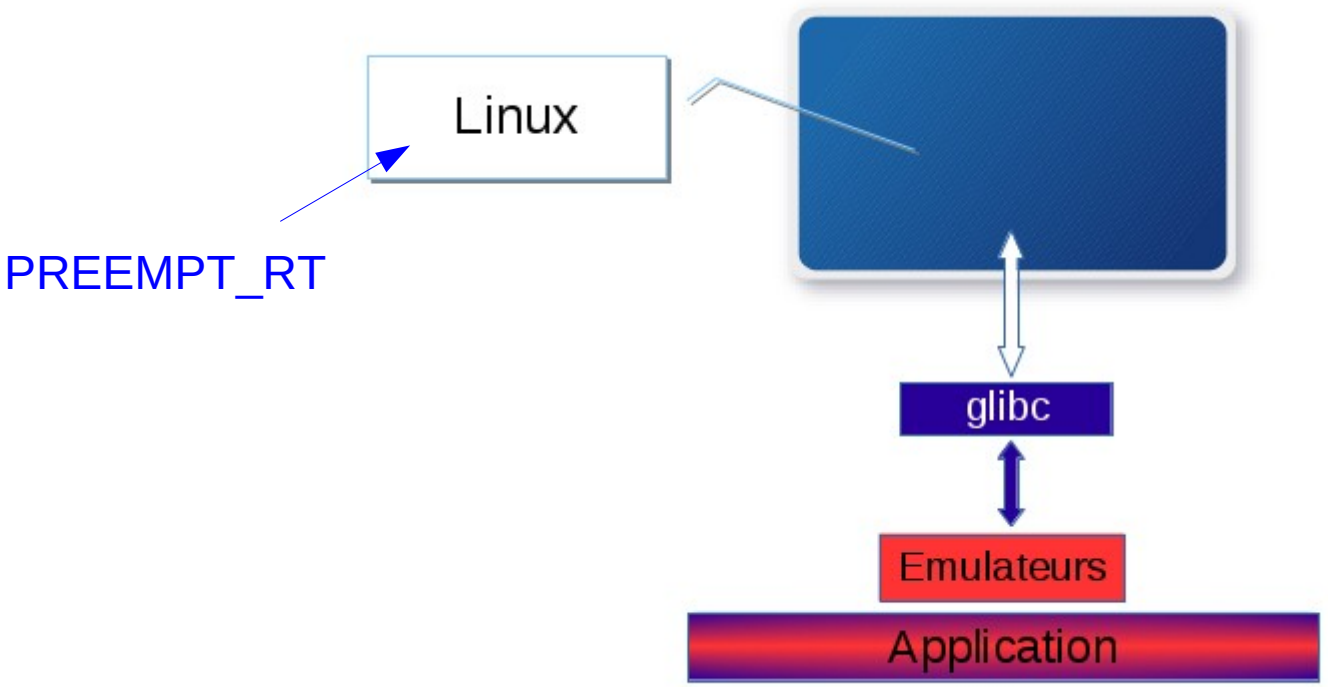
Xenomai

- Designed by Philippe Gerum in 2001 for RTOS emulation
- Close to RTLinux and RTAI (Linux + co-kernel)
- RTAI collaboration to “avoid” problems with RTLinux patent
- Based on micro-kernel (ADEOS) and interrupt pipeline (I-pipe)
- New architecture since 3.x with 2 choices:
  - co-kernel (Cobalt), most used
  - single kernel (Mercury) → PREEMPT_RT
- Current stable version is 3.1, legacy version is 2.6.5

Xenomai 3 (Cobalt)



Xenomai 3 (Mercury)



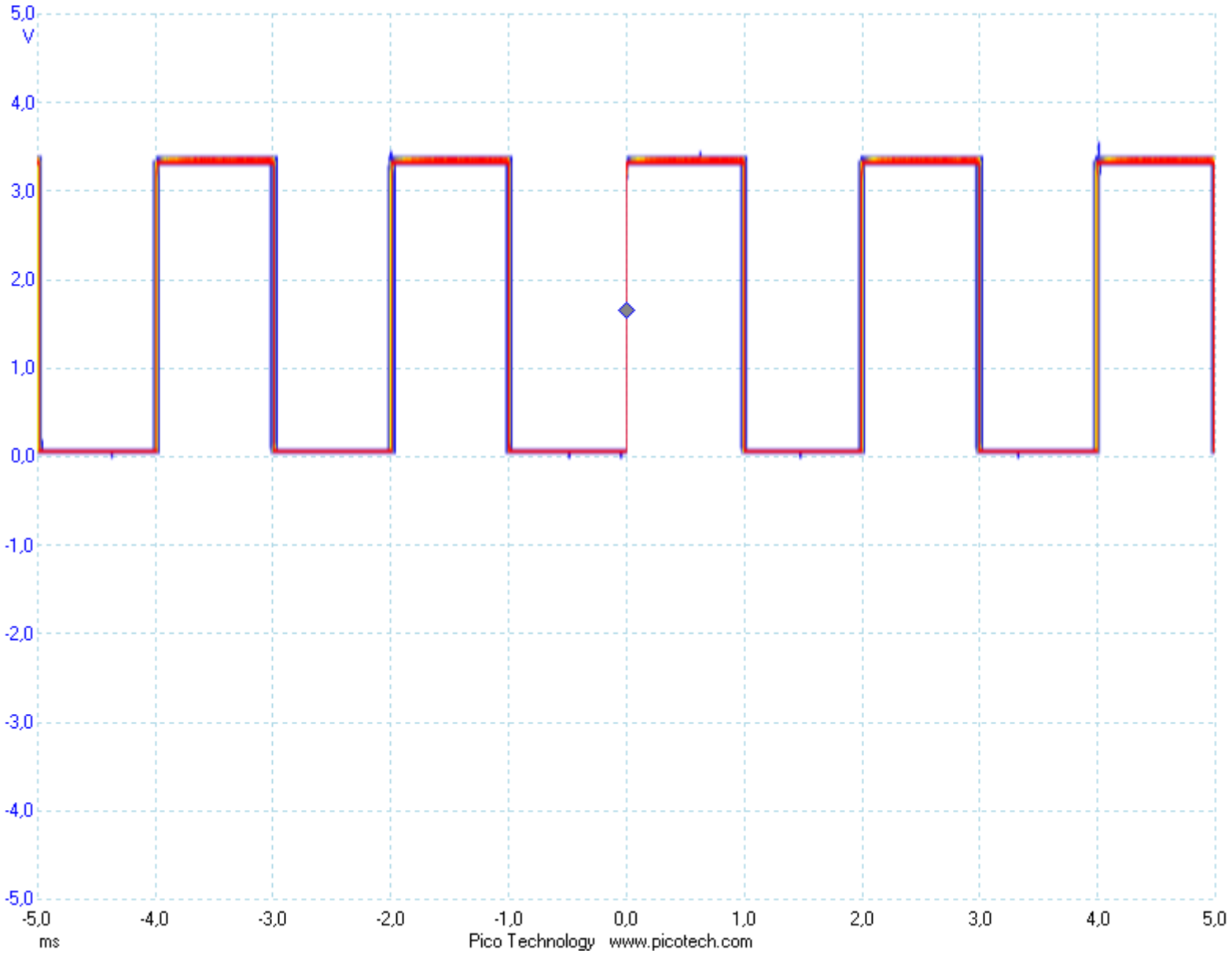
Xenomai :-)

- Most efficient RT extension for Linux kernel (30 % to 300 % better than PREEMPT_RT)
- RT tasks run in user space (no licensing problem → LGPL) !
- Provides “skins” for RTOS API (POSIX, VxWorks, VRTX, etc.)
- Used (and maintained) by big companies such as SIEMENS

Xenomai :-(

- Very small community
- Lacks of documentation and examples
- Lacks of internal documentation → difficult to adapt to a new architecture (I-pipe)
- Cobalt is more difficult to set up because of:
  - User / kernel installation
  - RTDM (specific drivers)
- Some standard debug tools should be adapted
- Not “mainline” (no support from Linux foundation)

BB Black Xenomai (2013)



Installing Xenomai

- Slightly more complex than PREEMPT_RT, as you need:
  - Kernel sources
  - Xenomai sources
  - I-pipe sources (patch)
- Create a Xenomai compatible kernel

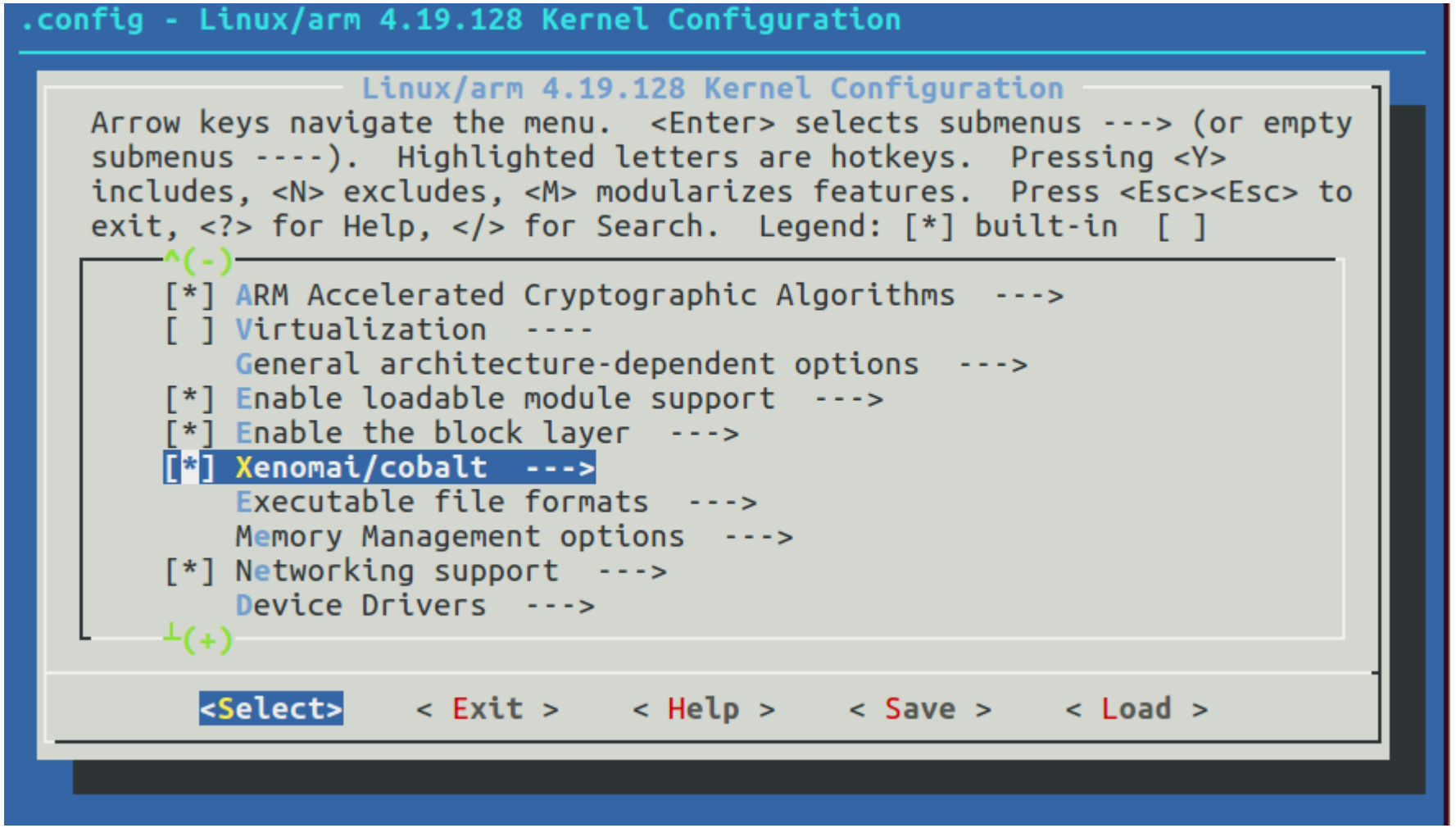
```
$ cd <xenomai-src-path>
$ ./scripts/prepare-kernel.sh --linux=<kernel-path> --arch=<arch> --ipipe=<ipipe-path>
```

- Compile user space libraries, tools, etc. (Autotools)

```
$ configure --host=<cross-toolchain-name> --enable-smp
$ make
```

- No “Xenomai ready” distribution
- Buildroot support (mainline)
- Yocto support with “meta-xenomai”

Kernel configuration



Testing Xenomai performances

- Testing is very close to PREEMPT_RT procedure
  - Periodic task
  - System stress (Linux domain)
- Xenomai provides `latency` and `cyclictest`
- Start `latency` (default period is 1 ms)

```
# latency &
```

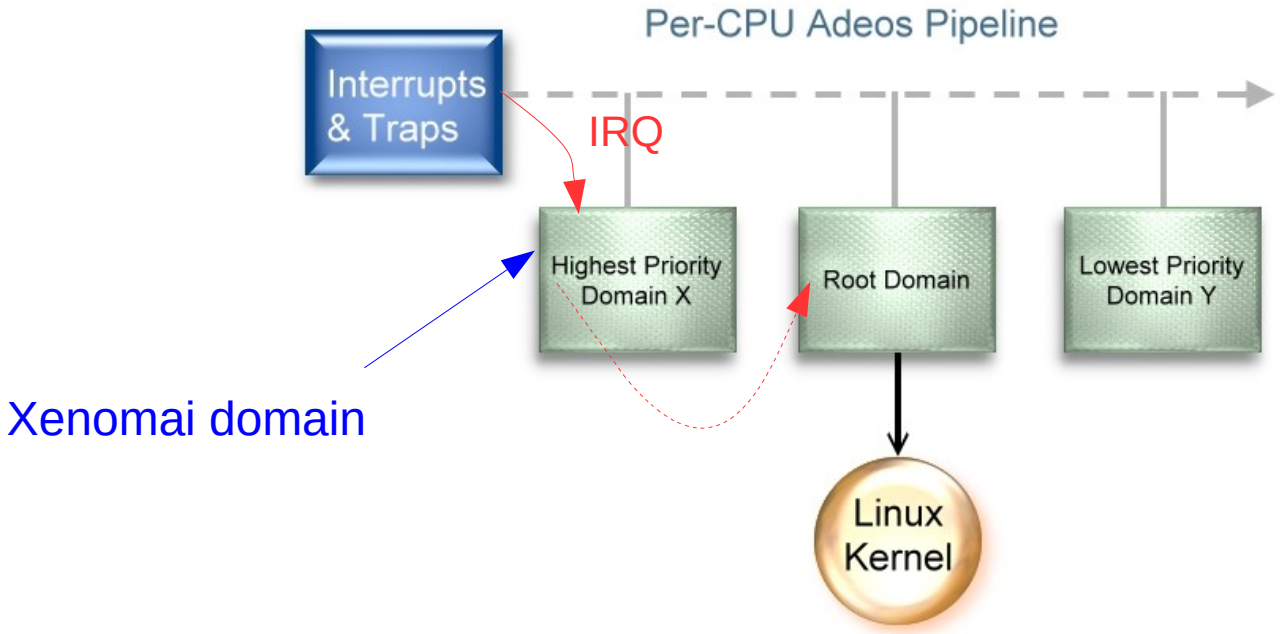
- Use `dohell` to stress the system (10 mn)

```
# dohell 600
```

- Maximum jitter for Raspberry Pi 3 is about 40 µs
- Maximum jitter for x86 is about 5/10 µs

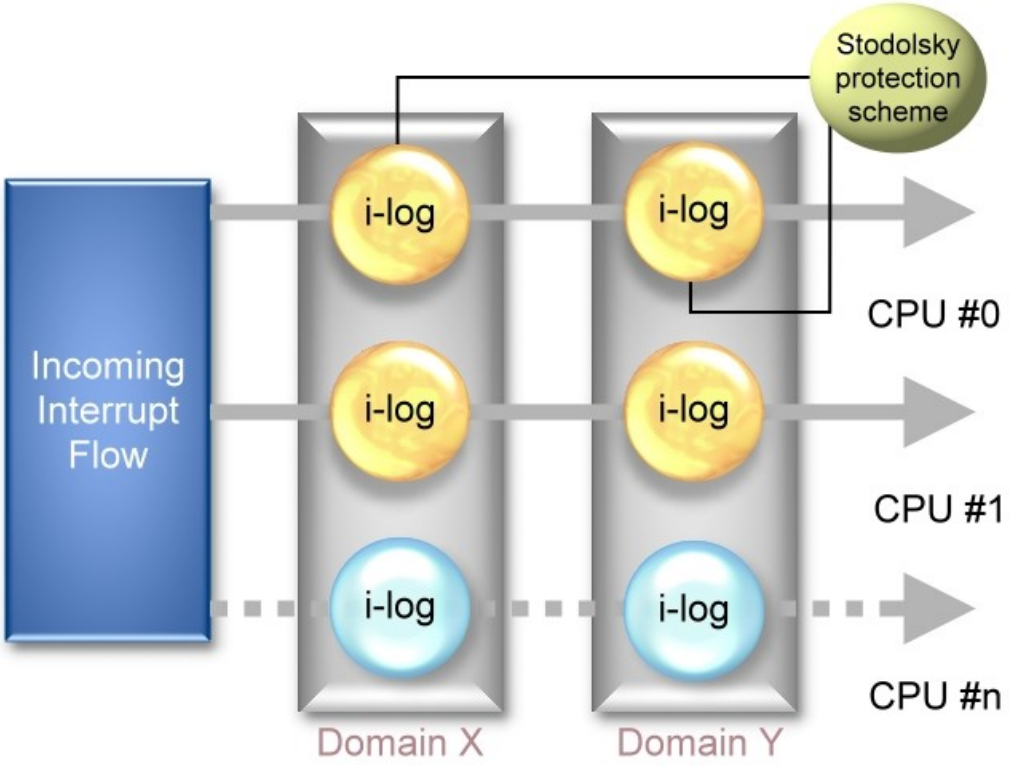
Domains and I-pipe

- Guest OS (Linux, Xenomai) are prioritized “domains”
- Xenomai is high priority domain
- Linux (aka “root”) is lowest priority domain
- For each event (interrupts, exceptions, syscalls, etc.), the domains may handle the event or pass it down the pipeline
- Calls to generic IRQ handlers should be replaced with calls to I-pipe functions (such as “ipipe_handle_domain_irq”)



Interrupt dispatching and protection

- Pipeline occupied by any given domain can be “stalled”
- Next incoming IRQ will NOT be delivered to the domain's handler (and domain with lower priority)
- Pending IRQ accumulate in the domain's interrupt log (i-log)
- Domains with higher priority (Xenomai) will still receive IRQ



Bibliography

- Virtual Memory and Linux https://events.static.linuxfound.org/sites/events/files/slides/elc_2016_mem.pdf
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/power_management_guide/tickless-kernel
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/real_time_throttling
- Linux scheduler latency https://www.eetimes.com/linux-scheduler-latency (old)
- Using realtime Linux https://elinux.org/images/8/8e/Using_Real-Time_Linux.KlaasVanGend.ELC2008.pdf
- The real-time preemption patch ftp://ftp.polsl.pl/pub/linux/kernel/people/tglx/preempt-rt/rtlws2006.pdf
- PREEMPT_RT / KR 2019 https://embedded-recipes.org/2019/talks/rt-is-about-to-make-it-to-mainline-now-what
- PREEMPT_RT mainlining in 5.x https://lwn.net/ml/linux-kernel/20190715150402.798499167@linutronix.de
- The magic behind PREEMPT_RT https://www.automateshow.com/filesDownload.cfm?dl=Haris-MagicBehindPREEMPTRT.pdf
- Open Patent License (RTLinux) https://www.gnu.org/press/2001-09-18-RTLinux.txt
- Xenomai 3 by Philippe Gerum (french) https://connect.ed-diamond.com/Open-Silicium/OS-016/Xenomai-3-hybride-et-cameleon
- Practical real-time Linux https://elinux.org/images/d/d7/Practical-Real-Time-Linux-ELCE15.pdf
- Porting Xenomai to a new ARM SoC https://source.denx.de/Xenomai/xenomai/-/wikis/Porting_Xenomai_To_A_New_Arm_SOC
- 3D printing with Linux & Xenomai https://elciotna18.sched.com/speaker/kendall_auel.1xog4usl
- Power PMAC (running Xenomai) http://www.deltatau.com/DT_PowerPMAC/PowerPMACHome.aspx
- RTAI/fusion https://lwn.net/Articles/106016
- Fast Interrupt Priority Management in OS Kernel (Stodolsky protection scheme) https://www.usenix.org/legacy/publications/library/proceedings/micro93/full_papers/stodolsky.txt
- Realtime preemption locking core merged https://lwn.net/Articles/867919