open wide INGENIERIE

Capitole du Libre 14-15-16 Novembre 2014

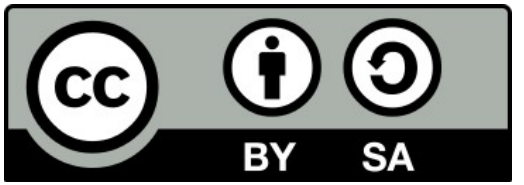
Linux temps réel sur Raspberry Pi (RPi)

Pierre Ficheux (pierre.ficheux@openwide.fr)

Novembre 2014

Agenda et licence

- Présentation de la carte RPi
- Distribution Raspbian
- Construction de distribution dédiée + exemple avec Buildroot
- Solutions temps réel sous Linux
  - PREEMPT-RT
  - Xenomai
- Nombreuses démonstrations

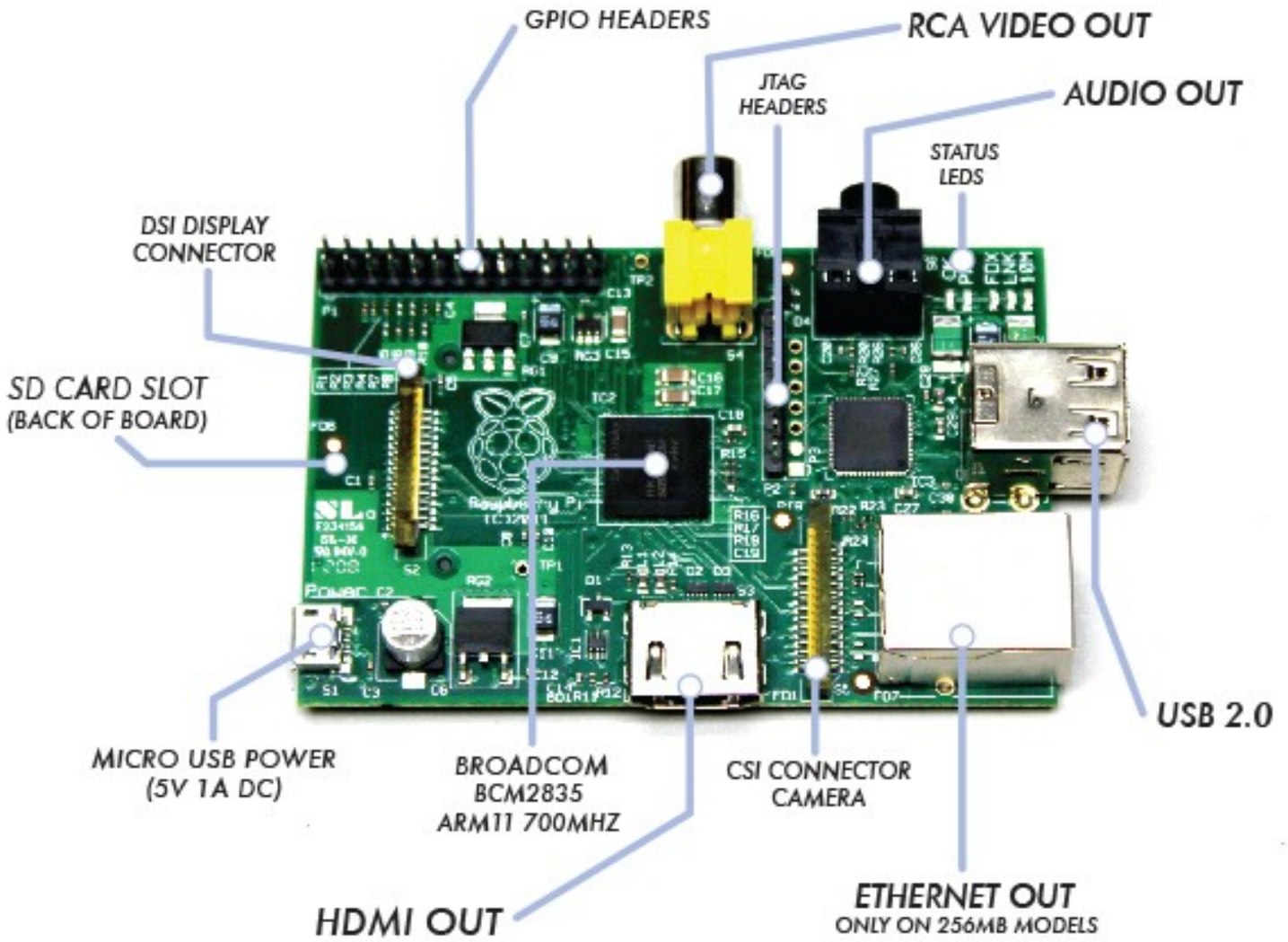


Présentation de la RPi

La Raspberry Pi

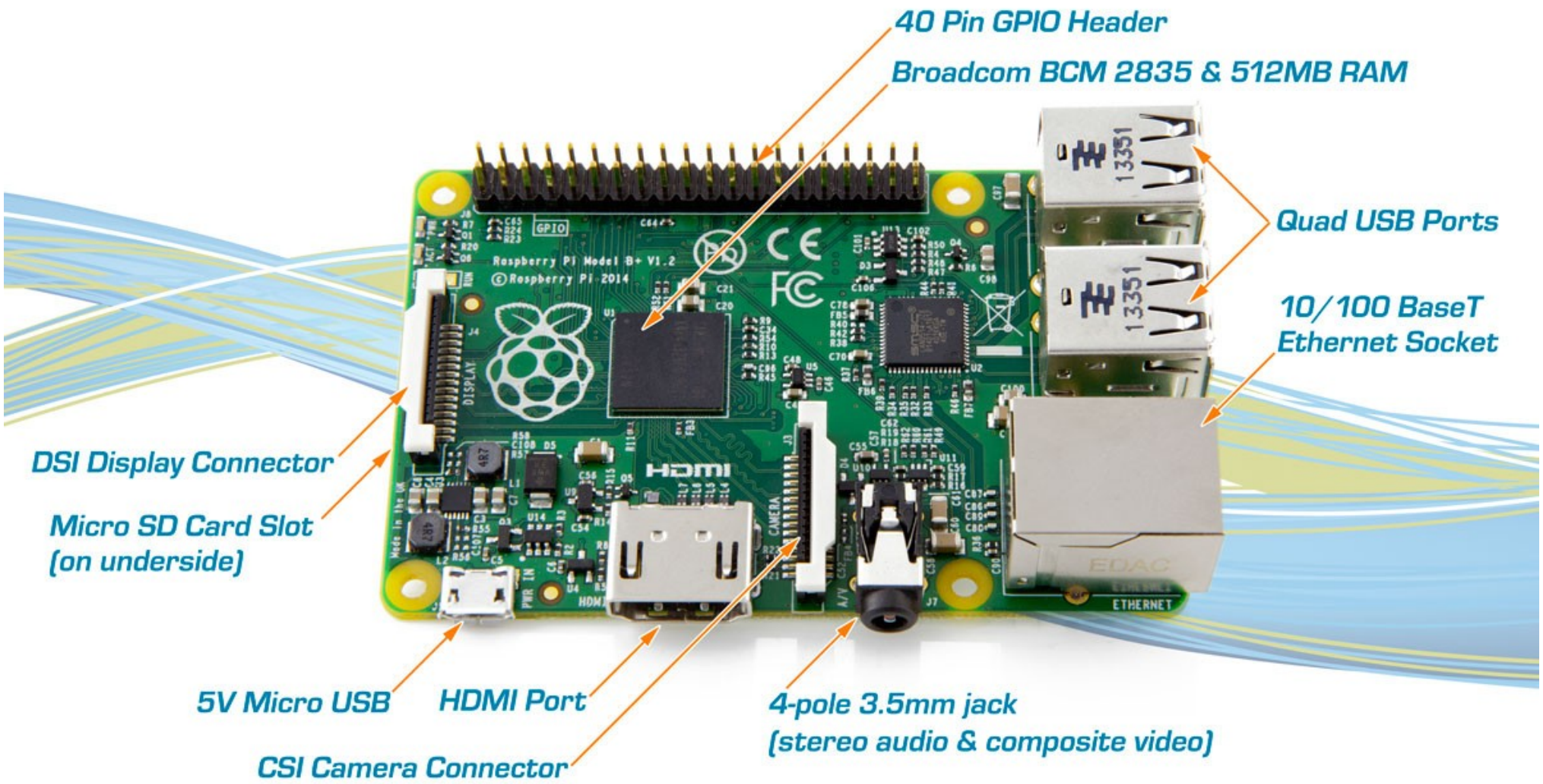
- Développée en UK à des fins d'enseignement « PC bon marché »
- Caractéristiques techniques
  - Processeur BCM2835 (Broadcom) à 700 Mhz
  - GPU performant, sorties vidéo composite et HDMI
  - 512 Mo de RAM (GPU+CPU)
  - Carte SD (pas de bootloader classique)
  - USB, Ethernet (sur USB)
  - GPIO, SPI, I2C
  - Audio
  - DSI/CSI (Display/Camera Serial Interface)
- Près de 4 millions d'exemplaires vendus à ce jour !
- Disponible à moins de 30€

Carte Raspberry Pi B (RPi)



open wide INGENIERIE

Raspberry Pi B+



La distribution Raspbian

- La Raspbian est un portage de la Debian 7.0 (wheezy) pour la RPi
- Distribution « par défaut » pour la RPi
- Il s'agit d'une distribution *classique* et non pas *embarquée* ni « temps réel »

```
# df -H
Filesystem      Size  Used Avail Use% Mounted on
rootfs          2.6G  2.0G  469M  82% /
...
/dev/mmcblk0p1   56M   24M   33M  30% /boot
```

- La distribution contient l'environnement de développement, X11, etc.
- La plupart des utilisateurs développent en Python !

Structure de la carte SD

- La carte SD contient au départ 2 partitions (ou plus)
  - Une partition VFAT de démarrage
  - Une partition EXT4 contenant la distribution
- La partition VFAT contient les fichiers de démarrage, configuration et les images noyau disponibles

  `kernel.img` (ou bien `zImage`) : noyau par défaut

  `kernel_emergency.img` : noyau de secours (statique)

  `cmdline.txt` : configuration du noyau Linux

  `config.txt` : configuration RPI (matérielle)

  `bootcode.bin` : initialisation du GPU (étape 1)

  `start.elf + fixup.dat` : initialisation du GPU (étape 2)

  `start_cd.elf + fixup_cd.dat` : initialisation du GPU (étape 2) si < 16Mo
- Sur Raspbian, la configuration peut être modifiée avec la commande `raspi-config`

Création de distribution

- Produit d'éditeur (WIND RIVER, MV, …) → €€€ 
- Utiliser un outil de génération : Buildroot, OpenEmbedded, OpenWrt, LTIB 
- Adapter une distribution Linux classique
  - Limité au niveau matériel
  - Empreinte mémoire importante 
  - Cas très particulier (Raspbian)
- Créer la distribution « from scratch »
  - Complexe mais pédagogique 
  - Difficile/impossible à industrialiser:
    - compilation croisée
    - gestion des dépendances
    - évolutions

Outil de création (build system)

- Un tel outil crée la distribution à partir des sources des composants adaptés en appliquant des « patch »
  - Il ne s'agit pas de *distribution* mais d'outil de *création de distribution*
- L'outil ne fournit pas les sources mais les *règles de production* et prend en compte les dépendances
- L'outil peut produire la chaîne croisée
- L'outil produit les différents éléments de la distribution
  - Image du bootloader (`u-boot.bin`)
  - Noyau Linux (`zImage`, `uImage`)
  - Images du root-filesystem (`rootfs.tar`, `rootfs.jffs2`, **`rpi-sdcard.img`**, ...)

Les principaux outils disponibles

- Yocto/OpenEmbedded
  - Moteur écrit en Python
  - Très puissant mais (très) lourd
  - Basé sur des fichiers de configuration (?)
- Buildroot
  - Basé sur la commande `make`
  - Au départ un démonstrateur pour uClibc
  - Désormais un véritable outil, bien maintenu !
- OpenWrt
  - Dérivé de BR
  - Orienté vers les IAD (Internet Access Device)
  - Gère les paquets binaires
- Autres: LTIB (Freescale), PTXdist (Pengutronix)

Using Buildroot « in a nutshell »

- Aide en ligne par :

```
$ make help
```

- Utiliser une configuration pré-définie (`configs`) :

```
$ make <myboard>_defconfig
```

- Consulter/ajuster éventuellement la configuration par :

```
$ make menuconfig
```

- Compiler par :

```
$ make
```

- Le résultat est dans le répertoire `output/images` :
  - Bootloader (si nécessaire) → `u-boot.bin`
  - Noyau Linux → `zImage`, `uImage`
  - Image(s) du root-filesystem → `rootfs.*`

Test avec la RPi

- Configuration et compilation :

```
$ make raspberrypi_std_defconfig
$ make
```

- Création de la SD

```
$ cd output/images
$ sudo dd if=rpi-sdcard.img of=/dev/sdb
```

- La distribution fonctionne en mode texte mais démarre en quelques secondes
- Pas de gestion de paquets sur Buildroot !

Linux temps réel

Rappels et définitions

- FAQ www.faqs.org/faqs/realtime-computing/faq/
- Le temps-réel est une notion de *garantie* et non pas de performance
  - qualité de fonctionnement (stabilité)
  - temps de réponse → réaction *appropriée* en un temps *borné* à un événement
- Ces exigences sont remplies par un « RTOS »
- « Tous » les systèmes embarqués classiques sont *temps réel* (VxWorks, LynxOS, QNX, ...)
- Le temps réel est lié à des processus *sensibles* (militaire, spatial, énergie, médical, transport)

open wide INGENIERIE

Le besoin temps réel

- Initialement : contexte industriel (*hard* RT = TR *dur*)
  - Pilotage matériel
  - Acquisition de données
- Dans le cas du temps réel *dur*, il n'y a *pas de tolérance* sur le comportement temps réel !
- Plus récemment : logiciel (*soft* RT = TR *mou*)
  - Piles protocolaires
  - Multimédia
- Portage de code depuis un RTOS vers Linux (qui lui n’est PAS un RTOS)

Mesures des performances TR sur Linux

- Principe :
  - Programmer une tâche périodique
  - Comparer la date d'échéance théorique avec la date réelle
  - La différence correspond au « jitter » (gigue)
  - Tester le système avec *ET* sans charge !
- Outils :
  - Tâche périodique: `cyclictest`, `latency`
  - Manipulation de « GPIO »
  - Stimulation: `hackbench`, `stress`
  - Mesure : Oscilloscope, Gnuplot (tracé de courbes)

Linux standard

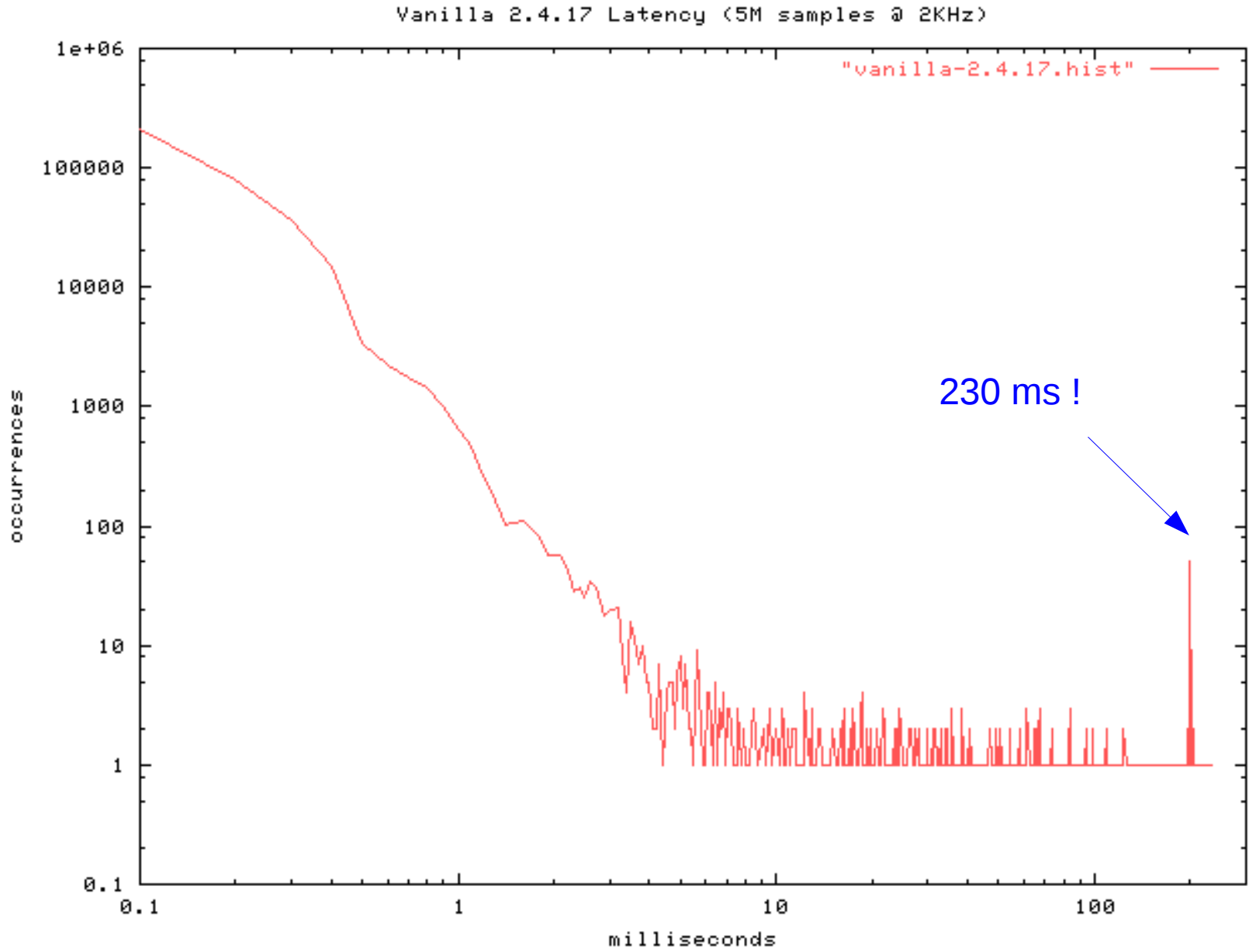
- Traitements souvent effectués par des *processus* (et non des *threads*)
- Modes d'exécution d'un processus
  - espace utilisateur (applicatif)
  - espace noyau (entrées-sorties)

    → gestion complexe de la mémoire peu fréquente sur un RTOS
- Préemption du traitement *applicatif*
  - L'ordonnancement est effectué par le noyau (UNIX)
  - Très différent du multitâche « collaboratif » de Windows 3.1 (et partiellement Windows 95) ou MacOS (< X)

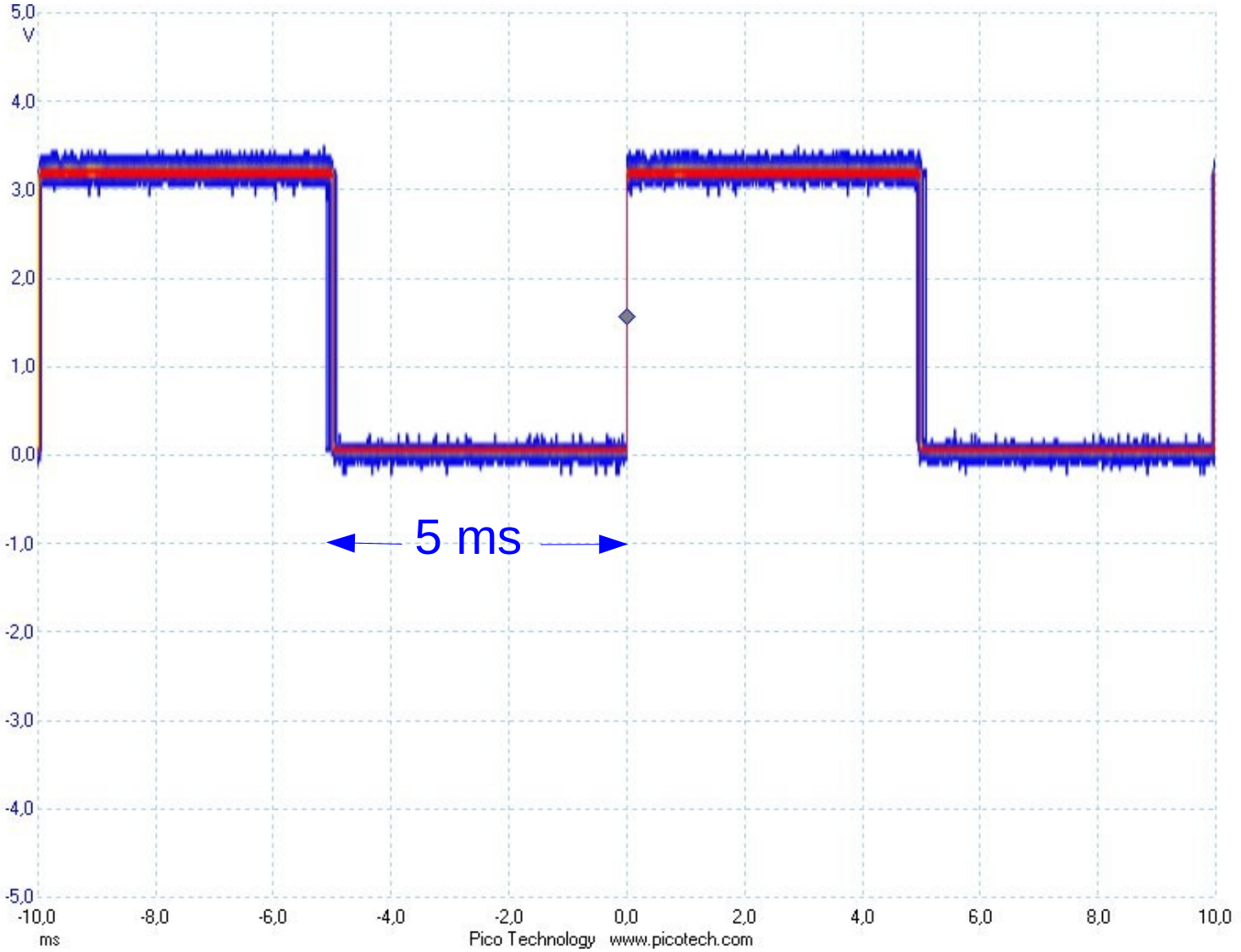
Linux standard

- Pas de préemption « complète » en mode noyau → un processus ne peut être interrompu dans une routine de traitement d'interruption (ISR)
- Préemption par l'ordonnanceur
  - Sur interruption *timer*
  - Fréquence *timer* fixe (constante HZ = nombre de ticks par seconde = 1000) →
- La précision de l'ordonnanceur = 1 ms (au mieux)
- Ordonnancement par niveau de priorité (POSIX)
  - Priorité dynamique standard (0, ajustable avec « nice ») → SCHED_OTHER
  - Priorité statique « temps réel » SCHED_FIFO/RR (1 à 99)
- La puissance du CPU n'améliore pas directement les performances

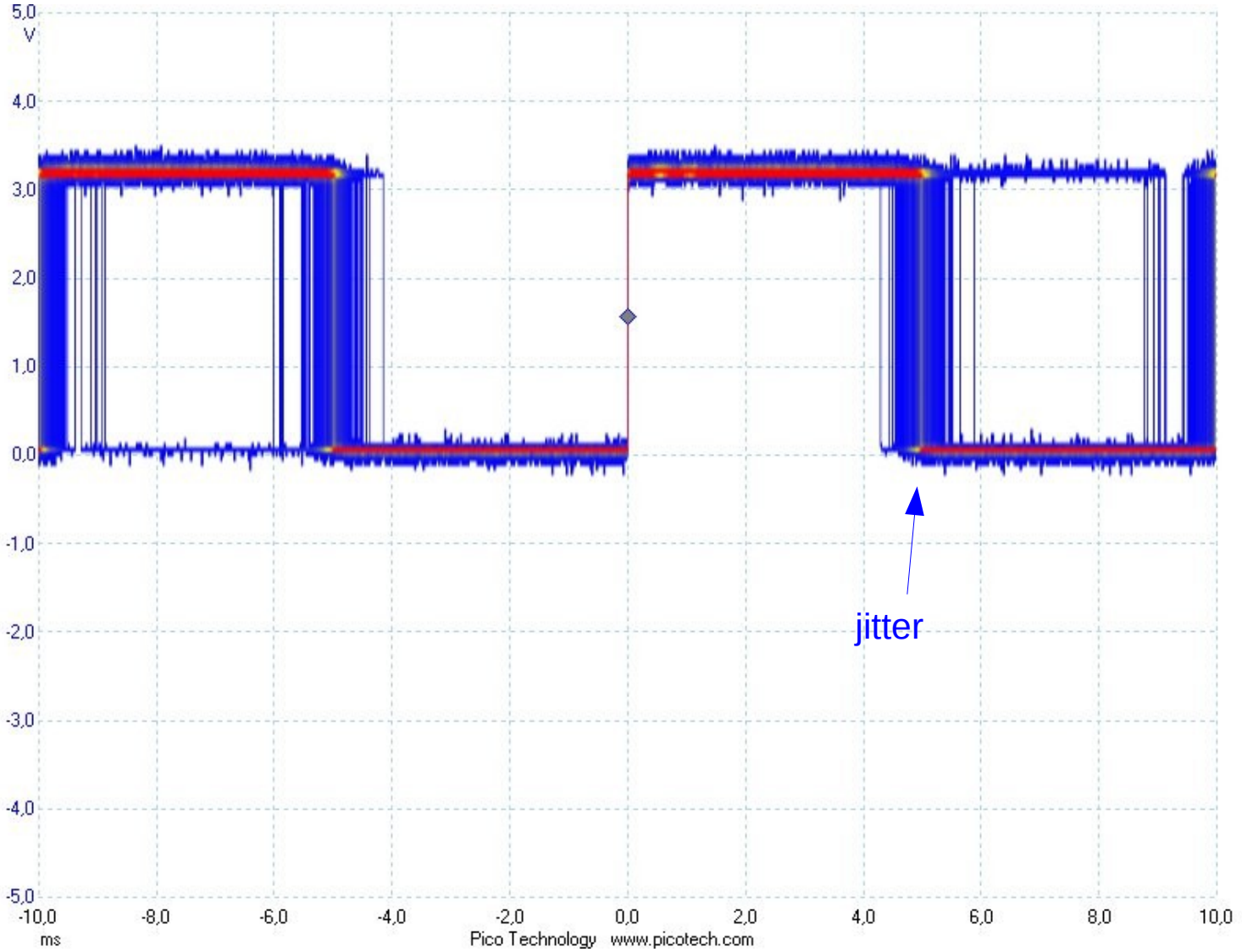
Test avec noyau standard 2.4.17 (x86)



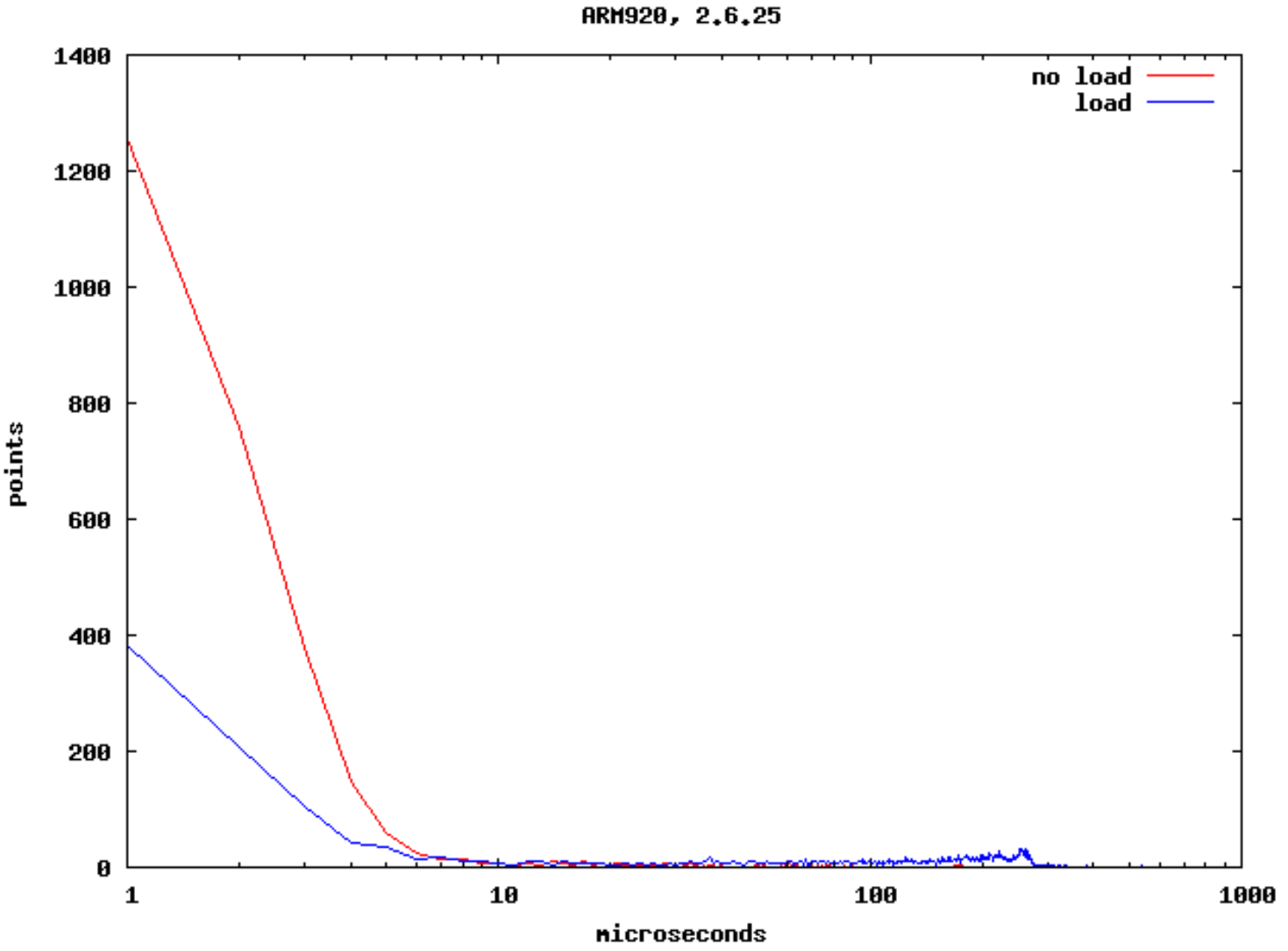
Test un noyau 2.6.25 (ARM9), non chargé



Test avec un noyau 2.6.25 (ARM9), chargé

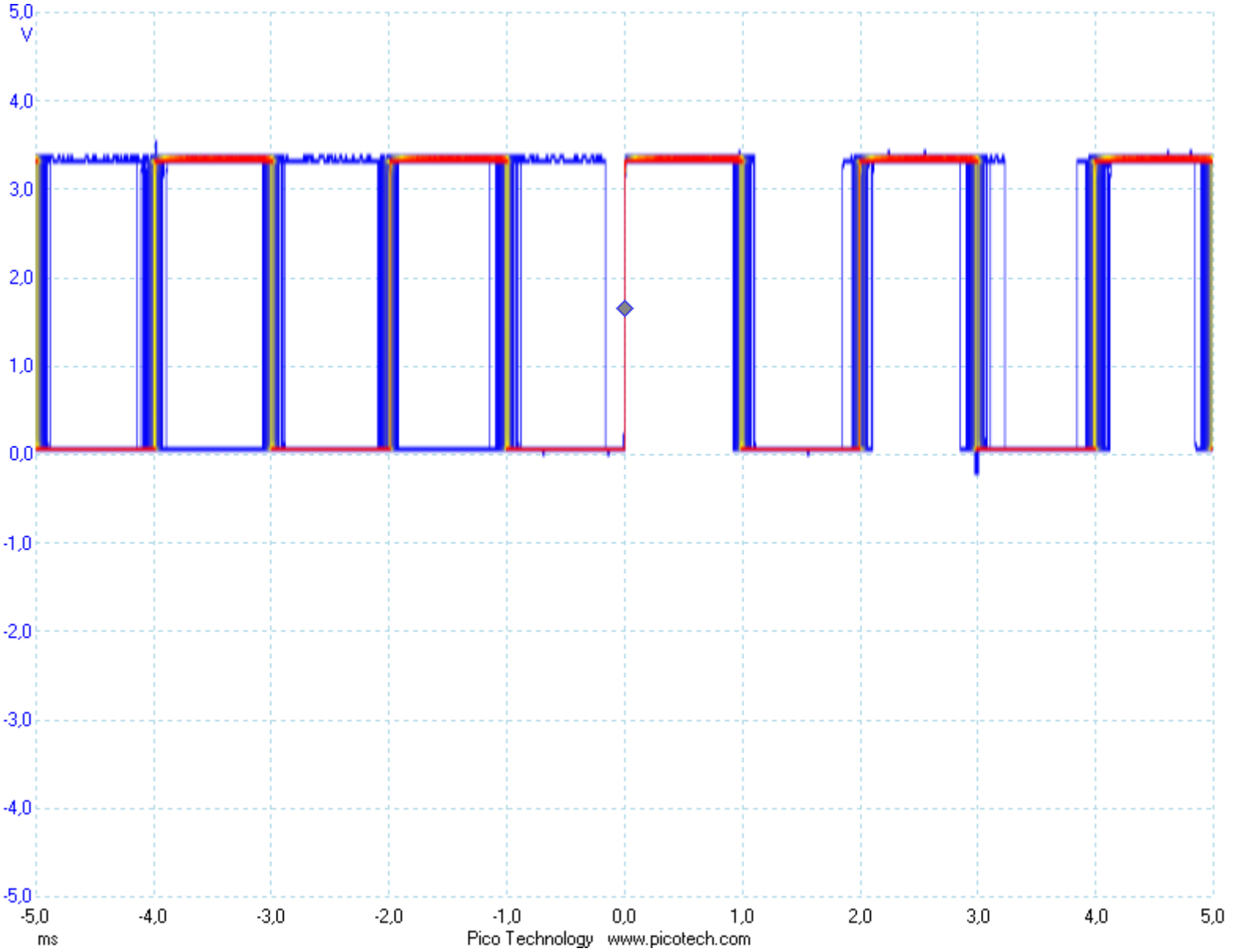


Test avec un noyau 2.6.25 (ARM9), enregistrement

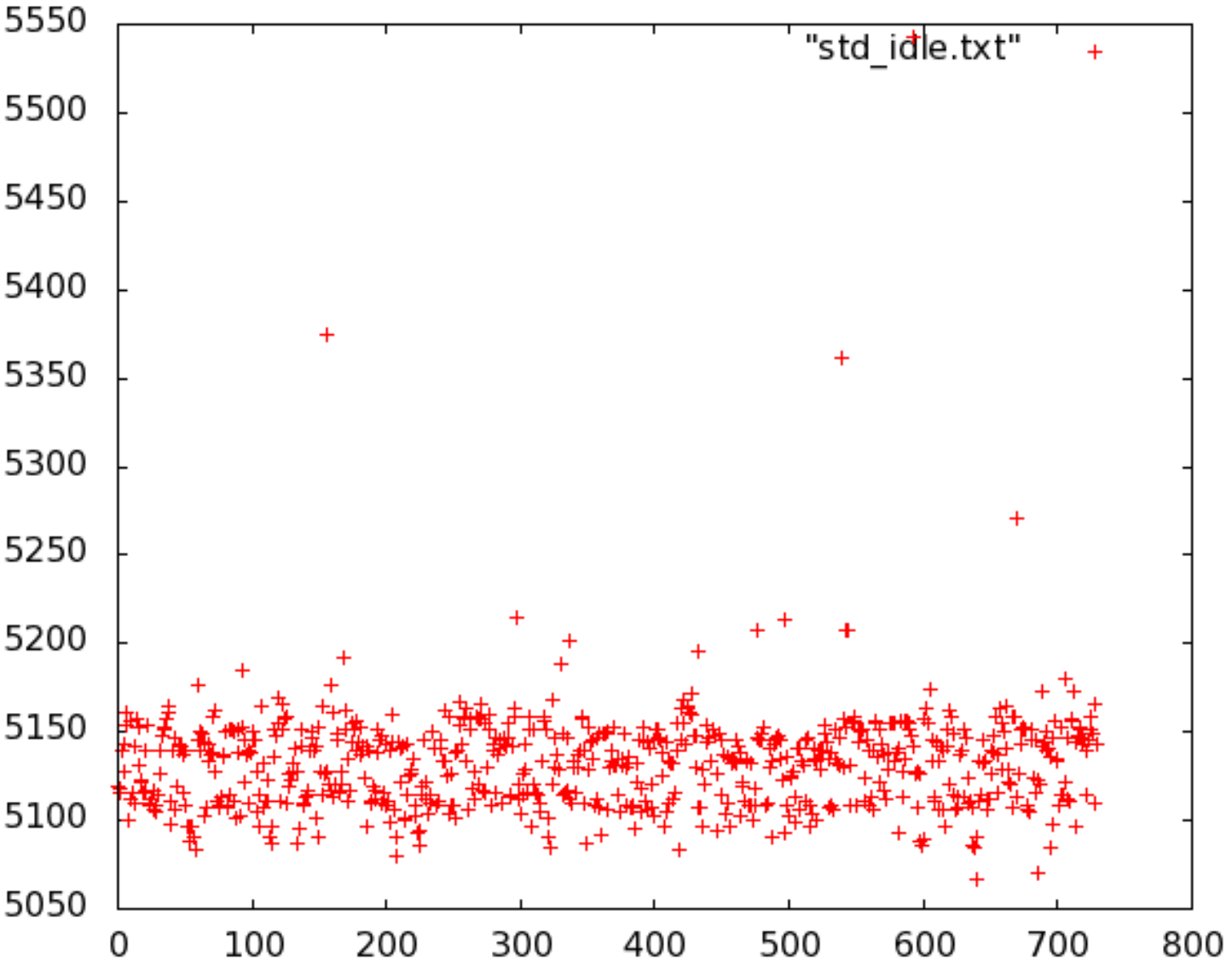


open wide INGENIERIE

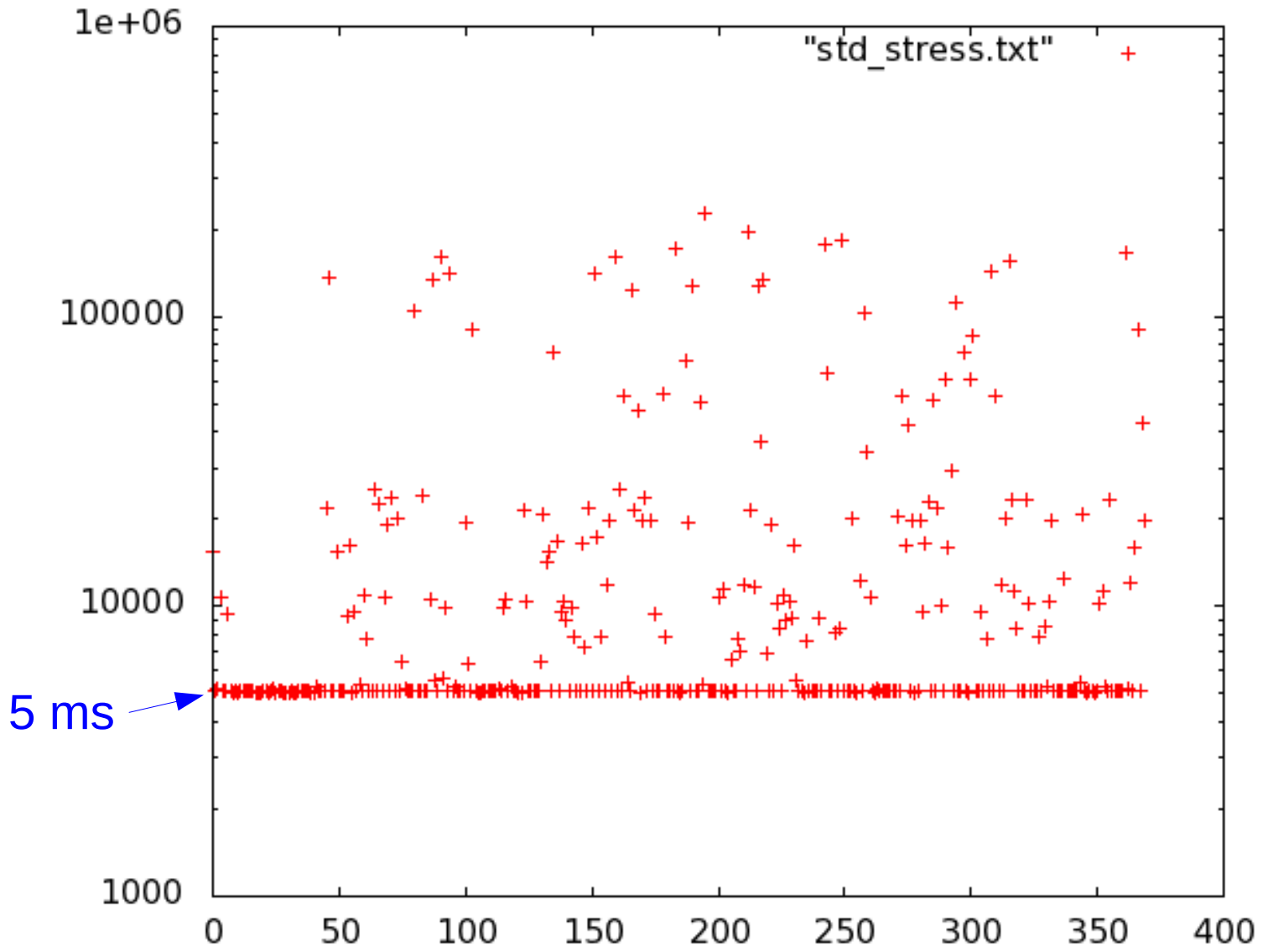
Raspberry Pi, noyau 3.6.11 (2013)



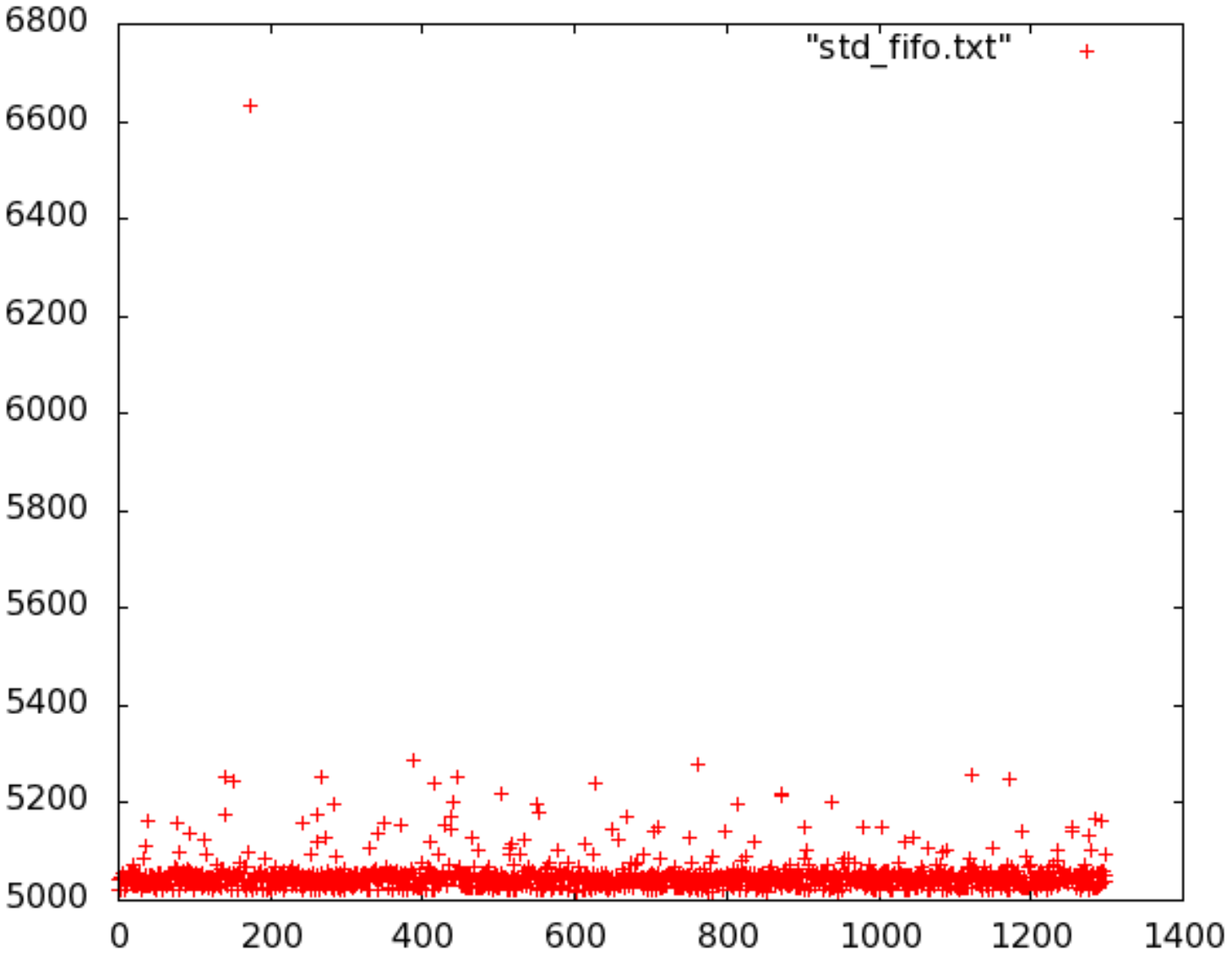
Test avec Ftrace (x86 non chargé)



Mesure avec Ftrace (x86 chargé)



Mesure avec Ftrace (x86 SCHED_FIFO, chargé)



PREEMPT-RT

- Branche expérimentale pour le noyau 2.6, voir https://rt.wiki.kernel.org
- Initié par Ingo Molnar, contributeur majeur du noyau
- Maintenu par Thomas Gleixner
- Aucun lien avec l'antique « preempt-kernel » !
- Surtout utilisé sur x86 et des processeurs performants (TSC = Time Stamp Counter)
- Fonctionne également sur ARM (9 ou plus), Nios II, Microblaze, ...
- Nécessite un noyau « mainline » (ou proche) mais ne sera probablement jamais intégré à la branche officielle
- Mise en place très simple (application d'un patch)
- Mêmes API de programmation que Linux standard

PREEMPT-RT, modifications

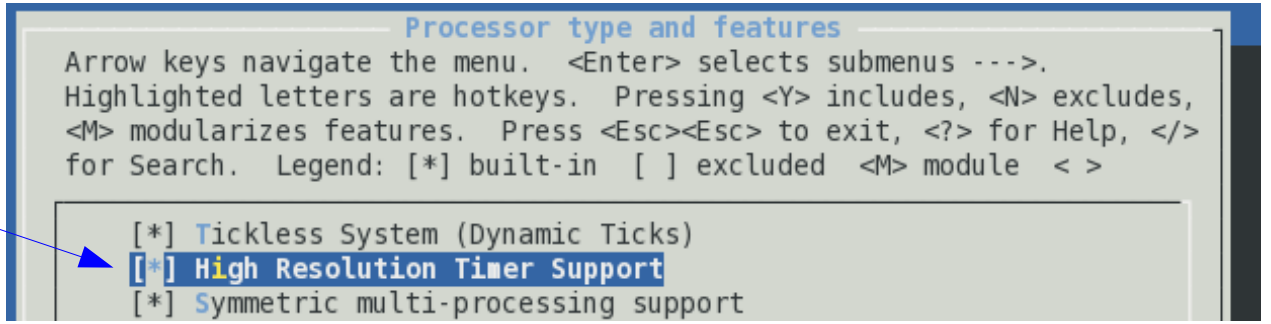
- *Threaded interrupt model* → utilisation d'un *thread noyau* (interruptible) pour le traitement de chaque interruption

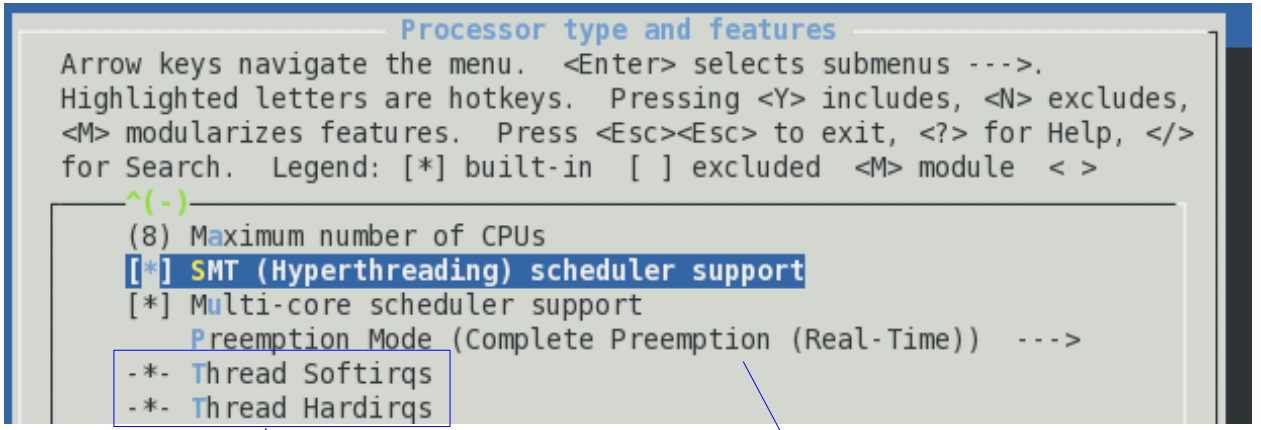
```
4     2 root     SW<      0   0%   0% [sirq-high/0]
5     2 root     SW<      0   0%   0% [sirq-timer/0]
...
6     2 root     SW<      0   0%   0% [sirq-net-tx/0]
```

- Prévention des inversions de priorité (par héritage)
- Timers noyau haute précision (API *hrtimer*)
- Réécriture complète des mécanismes de synchronisation (spinlock → mutex)
- Compatibilité avec *Ftrace* pour la mise au point
- La quasi-totalité du noyau est préemptible, mais reste un noyau Linux, donc très volumineux !

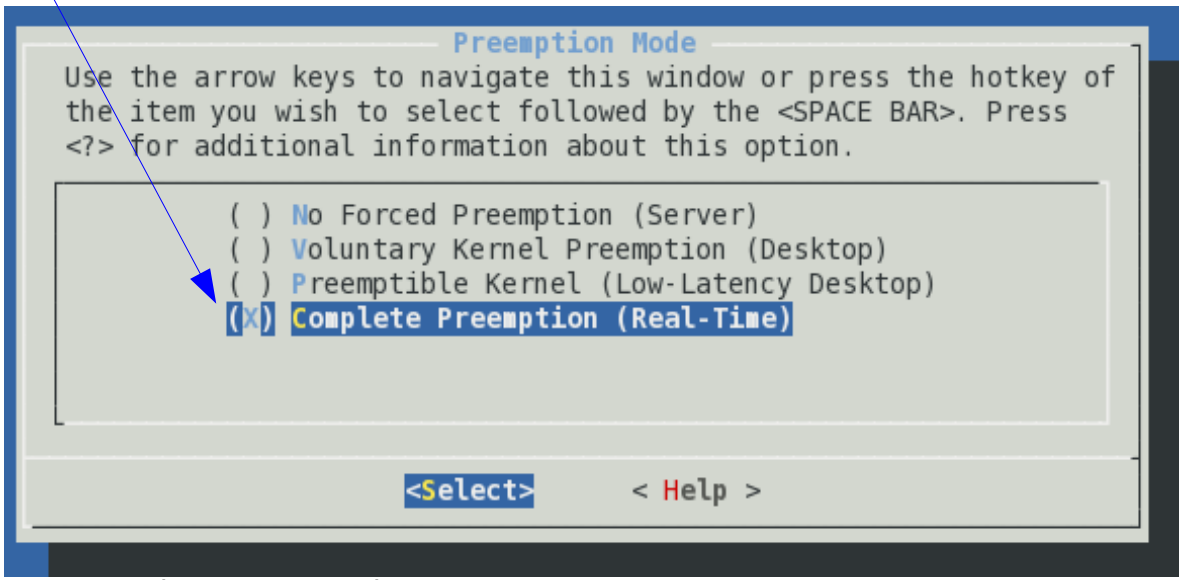
Configuration PREEMPT-RT

HRTIMER





IRQ = kernel thread



Linux PREEMPT-RT
Utilisation et performances

- Changements significatifs du code noyau
  - Sémantique de verrouillage des sections critiques
  - Inspection nécessaire de tous les composants
  - Volume du patch important
- Utilisation de `mlockall()` → verrouillage des pages mémoire en RAM
- Le coût de la préemption peut être important si le nombre de tâches TR augmente
- Temps de latence maximum nettement amélioré
  - dépend *largement* de la plate-forme matérielle (TSC)
  - dépend de la configuration logicielle
  - Bons résultats sur x86 pour les versions récentes, 2.6.30 et plus

PREEMPT-RT, mesures de performances

- PREEMPT-RT peut être intégré à Buildroot et utilisé avec le paquet *rt-tests*
- Utilisation du programme `cyclictest` avec 5 threads temps réel (période = 1 ms) :

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

- Charge avec `hackbench` (création de N processus communiquant entre eux par pipe)

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

1000 messages

800 tâches !

- Plate-forme Atom/x86 (N550)
  - gigue avec noyau standard : **16 ms** !
  - gigue avec PREEMPT-RT : < 50 µs :-)
- Sur RPi, la gigue tourne autour de 100 à 150 µs

Test PREEMPT-RT sous RPi

- Sans charge

```
# cyclictest -p 99 -t 5 -n -q
T: 0 (   67) P:99 I:1000 C:   4120 Min:     14 Act:   23 Avg:   23 Max:       55
T: 1 (   68) P:98 I:1500 C:   2747 Min:     14 Act:   22 Avg:   32 Max:       83
T: 2 (   69) P:97 I:2000 C:   2177 Min:     15 Act:   21 Avg:   22 Max:       53
T: 3 (   70) P:96 I:2500 C:   1741 Min:     15 Act:   27 Avg:   24 Max:       60
T: 4 (   71) P:95 I:3000 C:   1451 Min:     15 Act:   25 Avg:   21 Max:       55
```

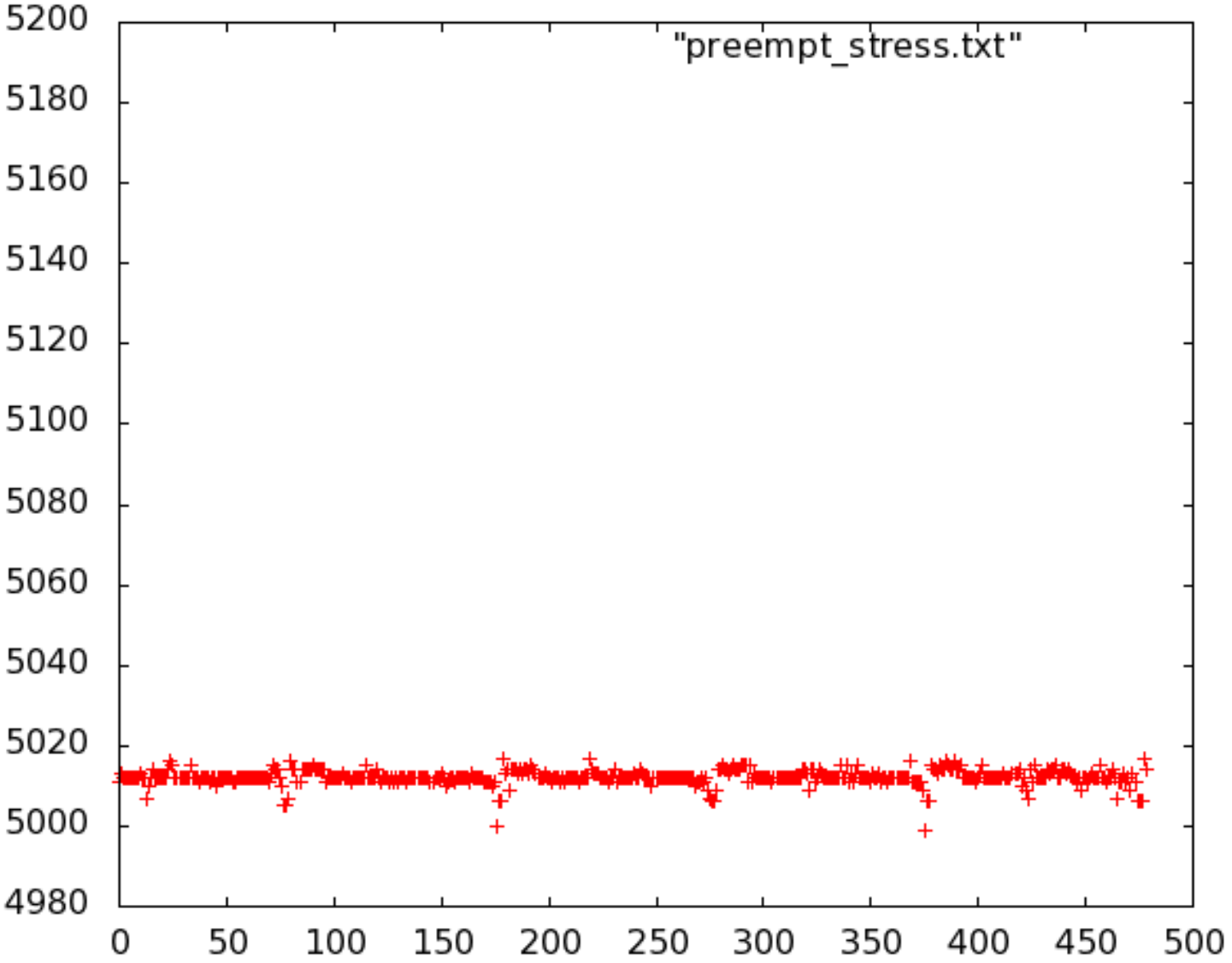
- Avec charge hackbench

```
T: 0 (   81) P:99 I:1000 C:  28485 Min:     14 Act:   21 Avg:   26 Max:       77
T: 1 (   82) P:98 I:1500 C:  18990 Min:     14 Act:   24 Avg:   28 Max:       88
T: 2 (   83) P:97 I:2000 C:  14243 Min:     15 Act:   41 Avg:   32 Max:      156
T: 3 (   84) P:96 I:2500 C:  11394 Min:     15 Act:   28 Avg:   36 Max:      165
T: 4 (   85) P:95 I:3000 C:   9495 Min:     15 Act:   28 Avg:   35 Max:       79
```

- Noyau standard

```
T: 0 (   73) P:99 I:1000 C:  22392 Min:     15 Act:   28 Avg:   28 Max:     2042
T: 1 (   74) P:98 I:1500 C:  14928 Min:     16 Act:   20 Avg:   32 Max:     1616
T: 2 (   75) P:97 I:2000 C:  11196 Min:     16 Act:  115 Avg:   38 Max:     1376
T: 3 (   76) P:96 I:2500 C:   8957 Min:     16 Act:   37 Avg:   37 Max:     1718
T: 4 (   77) P:95 I:3000 C:   7464 Min:     17 Act:   38 Avg:   36 Max:     1523
```

Mesure avec Ftrace (x86 PREEMPT-RT, chargé)



Linux + co-noyau

- Ajout d'un « co-noyau » pour la gestion du temps-réel
  - Sous-système temps-réel intégré à un module noyau
  - Patch de « virtualisation » des interruptions
- Différents modèles :
  - Noyau uniquement (RTLinux/Free, www.rtlinuxfree.com) → version libre obsolète
  - Noyau & espace utilisateur, semi-intégration Linux (RTAI, www.rtai.org)
  - Noyau & espace utilisateur, intégration Linux complète (Xenomai, www.xenomai.org)

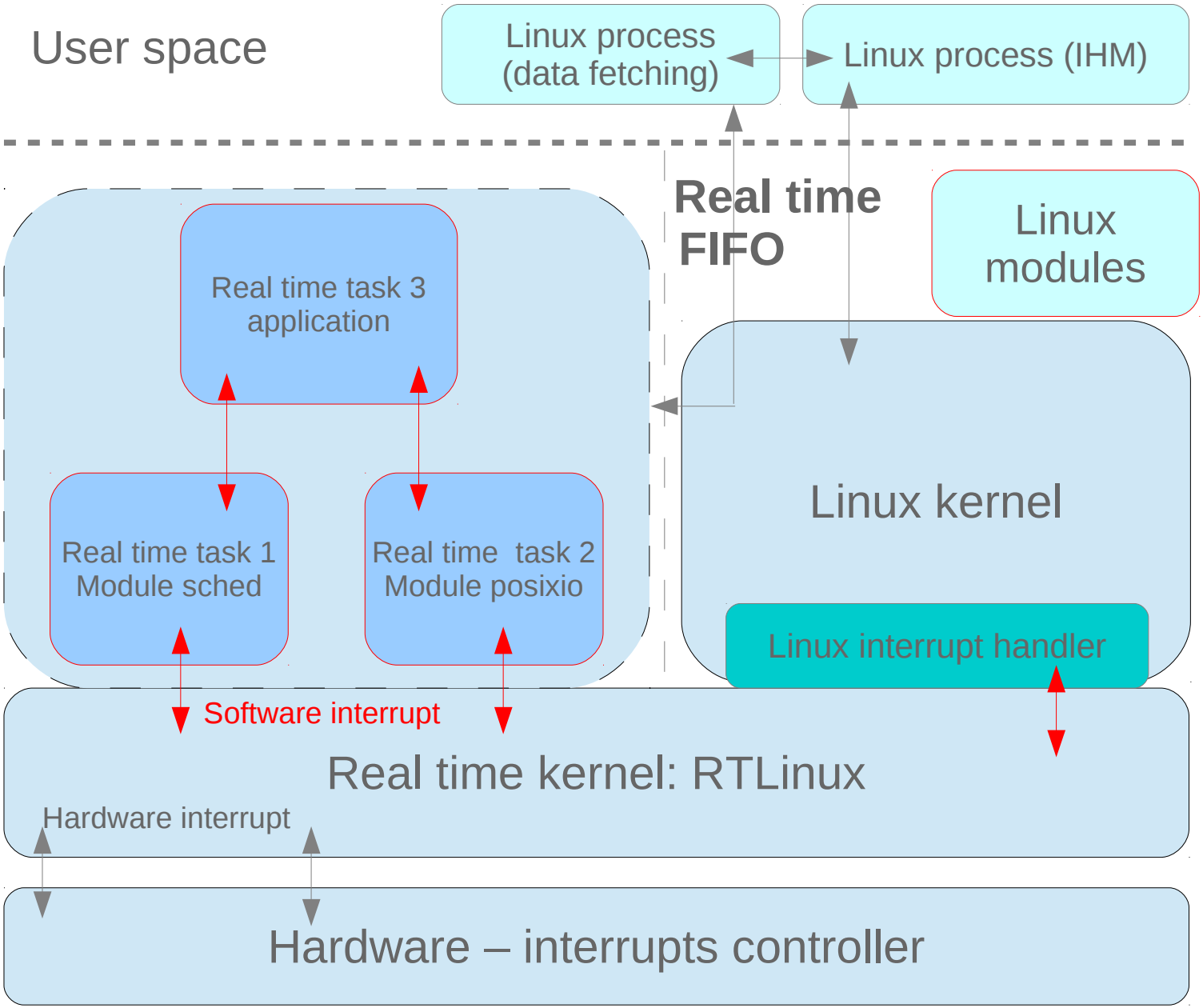
Linux + co-noyau Fonctionnement

- Séparation entre le composant temps-réel et Linux
  - Ordonnanceur temps-réel spécifique
  - Pas de dépendance sur les sections critiques Linux :-)
- Virtualisation de la gestion d'interruptions Linux
  - Routage prioritaire des IRQs vers le co-noyau
- Linux comme tâche *idle* du co-noyau
- Volume du patch noyau plus faible qu'avec PREEMPT-RT → 11000 lignes pour Xenomai/ADEOS
- Se rapproche de la technique de « para-virtualisation » des hyperviseurs (adaptation de l'OS)

RTLinux

- Projet universitaire (NMT) développé par Victor Yodaiken et Michael Barabanov en 1999
- Produit commercial développé par FSMLabs
- Dépôt d’un brevet logiciel → conflit avec la FSF
- Vendu à WIND RIVER en 2007
- Développement en espace noyau (?)
- Version GPL obsolète (2.6.9) retirée par WIND RIVER

Architecture RTLinux



RTAI

- Real Time Application Interface
- Un « fork » de RTLinux développé au DIAPM de l'école polytechnique de Milan → Dipartimento di Ingegneria Aerospaziale (Paolo Montegazza)
- Utilisé au DIAPM pour des travaux d'enseignement et de recherche
- Position douteuse / brevet logiciel FSMLabs
- Collaboration avec Xenomai entre 2003 et 2005 → RTAI/Fusion
- Fonctionne surtout sur x86
- Toujours actif mais peu d'évolution → version 3.8 en février 2010, 3.9 en août 2012, 4.0 en décembre 2013

Xenomai

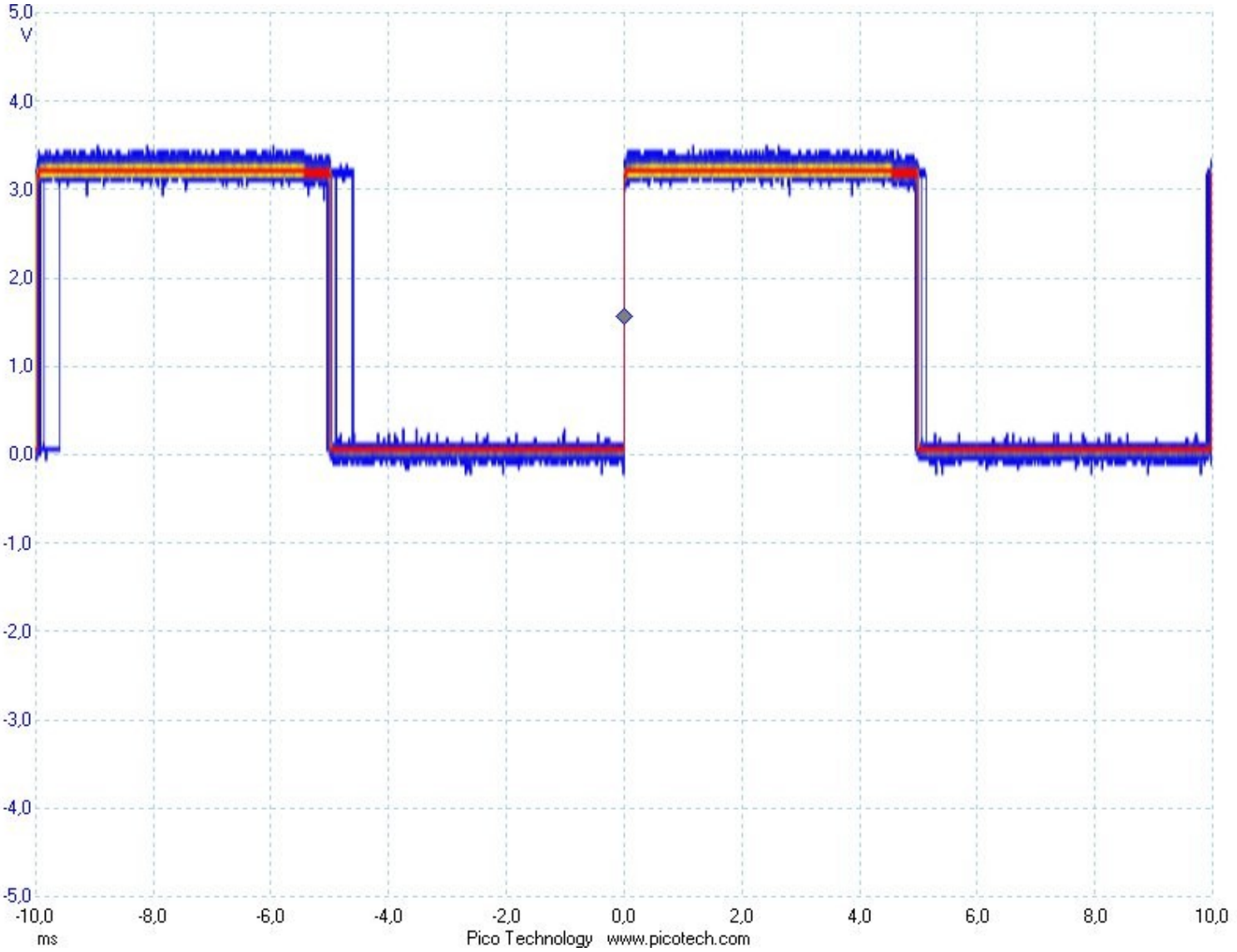
- Autrefois lié à RTAI
- Indépendant depuis 2005
- Développement de tâches temps réel en espace utilisateur :-)
- API de pilotes noyau → RTDM (Real Time Driver Model)
- Projet Linux/TR le plus avancé (et performant) à ce jour

Linux + co-noyau, synthèse

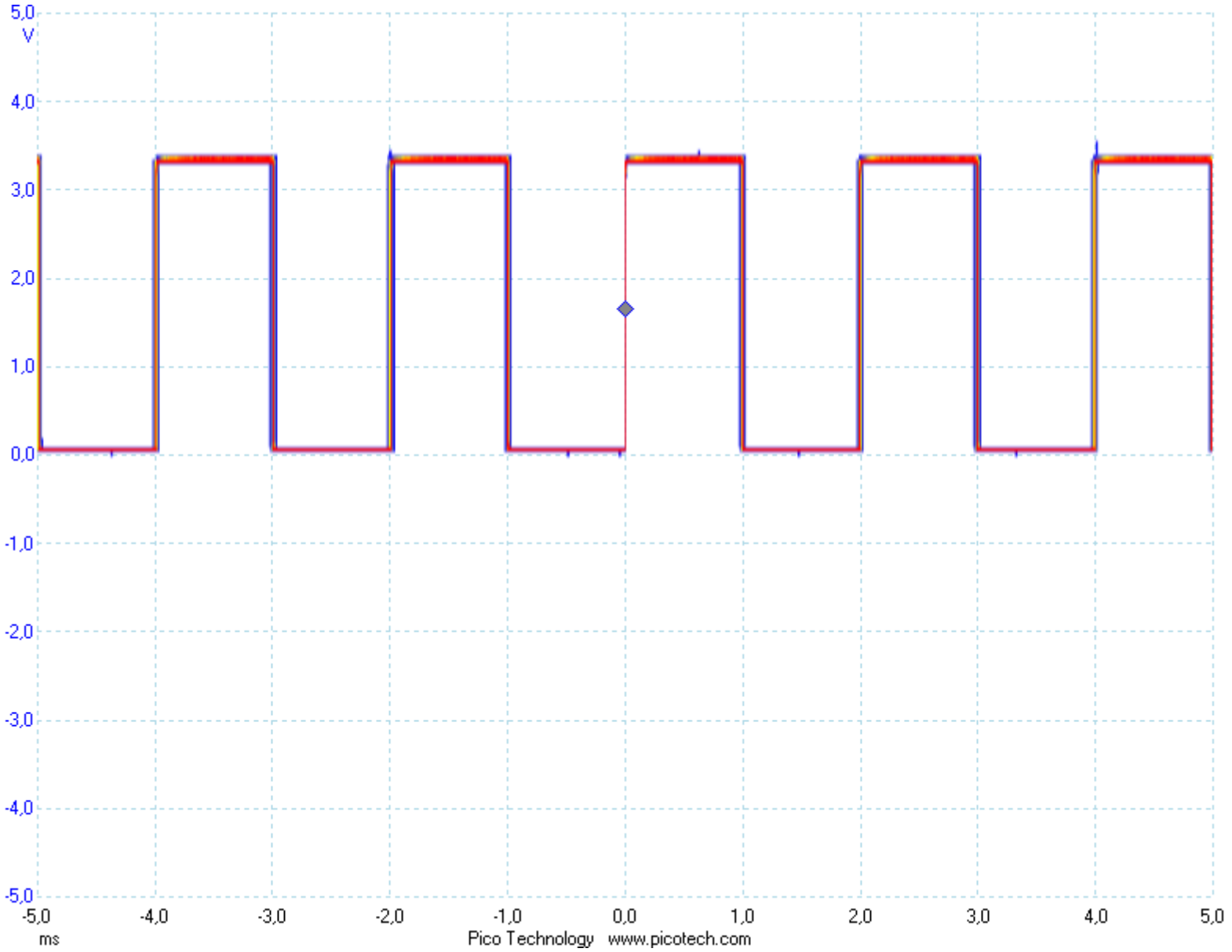
- Changement minimal sur le noyau Linux
  - *patch* de virtualisation d'interruptions
  - pas d'impact sur l'écriture de code noyau classique
- Impact sur l'écriture de code temps-réel !
  - utilisation d'APIs fournies par le co-noyau
  - notion de domaine d'exécution (temps-réel / normal)

- Garanties temps-réel fortes
  - ordonnanceur spécifique *indépendant*
  - sous-système temps-réel bien délimité
  - Avec Xenomai ou RTAI, gigue maximale de l'ordre de 10 µs sur Atom N550, 40 µs sur RPi !

Test Xenomai 2.6.25/ARM9 (2011)



RPi sous Xenomai (2013)



Historique Xenomai

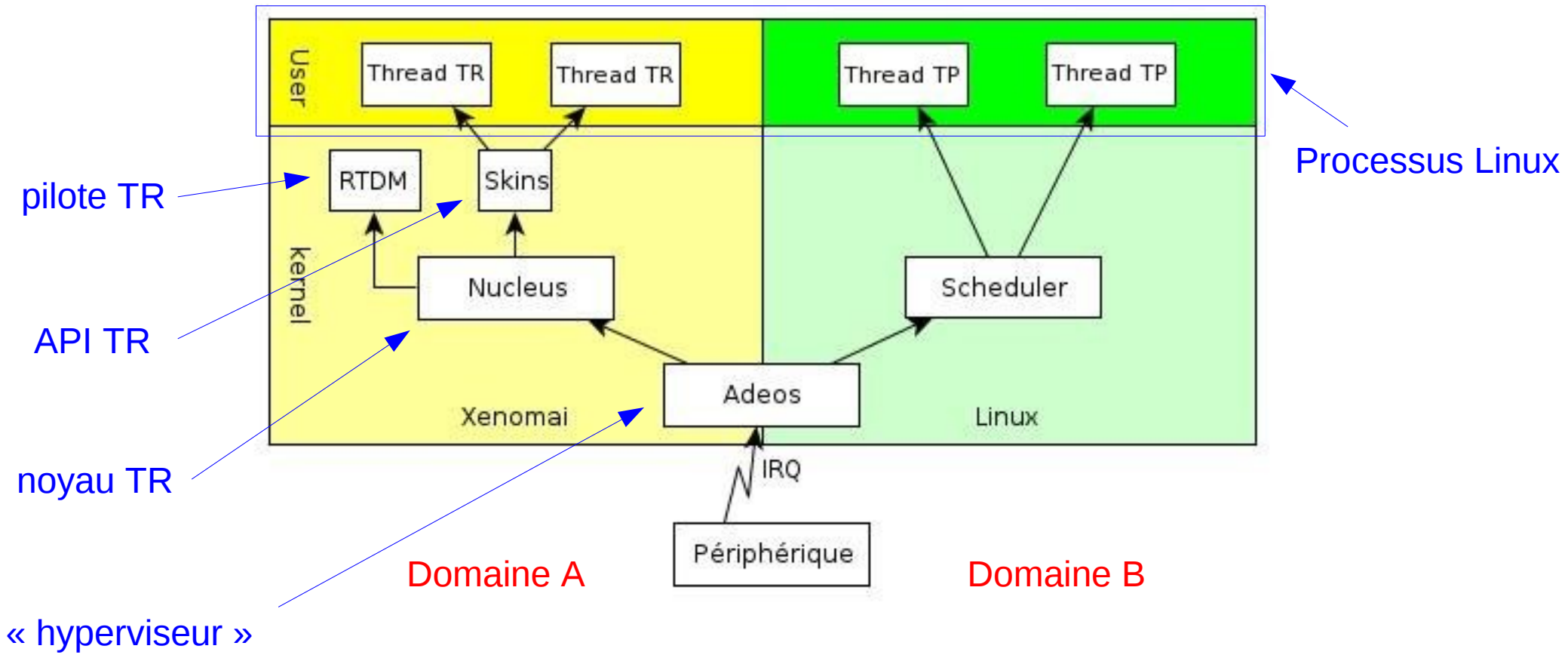
- Créé en 2001 par Philippe Gerum → émulation d'API RTOS (Xenodaptor)
  - Xenomai v0.5 – septembre 2001
  - Xenomai v1.1.1 – décembre 2002
- Intégration à RTAI en 2003
  - RTAI/fusion v0.1 – juin 2004
- Indépendance reprise en 2005
  - Xenomai 2.0 – octobre 2005 (RTAI/fusion v0.9.1)
  - Xenomai 2.1 – mars 2006
  - ...
  - Xenomai 2.6.0 – novembre 2011
  - Xenomai 2.6.3 – octobre 2013
  - Xenomai 3.0 – 2014 (Git)

What is Xenomai ?

- Xenomai est un sous-système temps-réel de Linux
  - Faible latence
  - Programmation de tâches possible en espace utilisateur
  - API de pilotes temps réel (RTDM) ! 
- Intégré à Linux → « Real-time sub-system »
- Supporte de nombreuses architectures
- Dispose de « skins » permettant d'émuler des API temps réel (POSIX, VxWorks, VRTX, uITRON, ...)
- Faible empreinte mémoire
- Plus complexe à mettre en œuvre que PREEMPT-RT mais performances supérieures (5 à 10 fois)
- Licence GPL (cœur), LGPL (interfaces, espace utilisateur)

Architecture générale

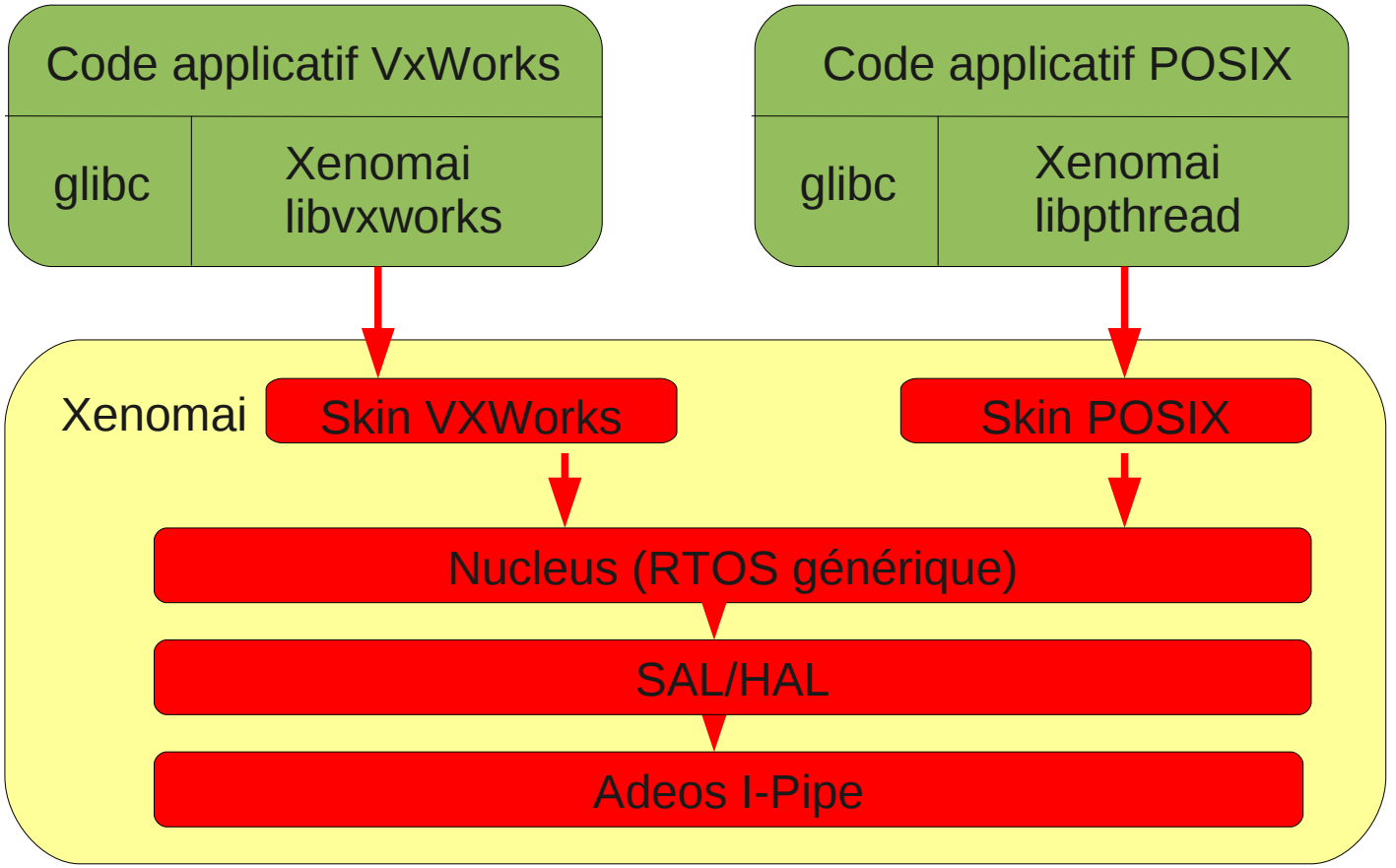
- Xenomai utilise un « hyperviseur » (ADEOS) pour *partager* le matériel avec le noyau Linux
- Un processus contient des threads TR et TP (Linux)



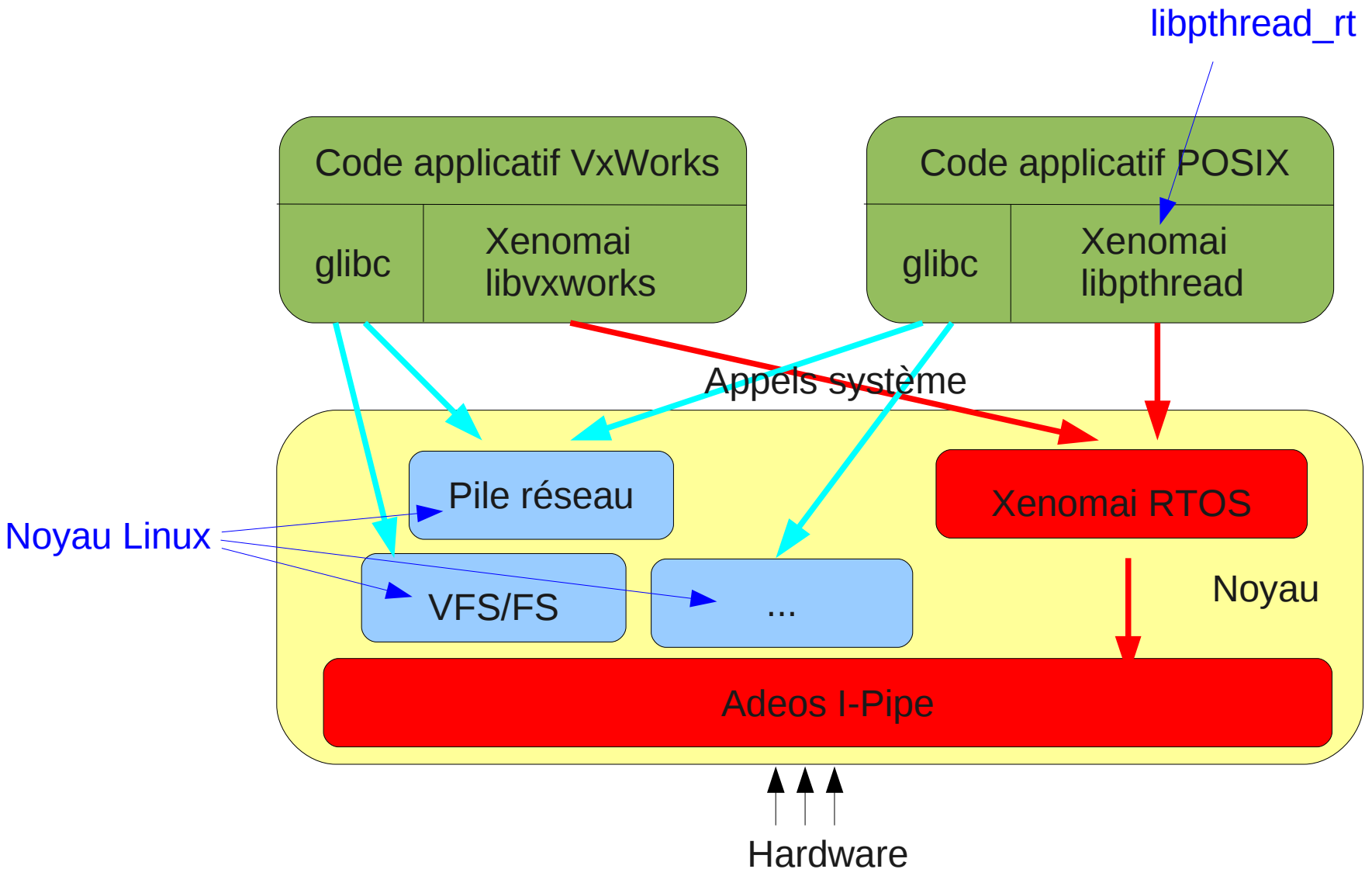
Architecture générale

- Adaptabilité
  - cœur de RTOS générique → « nucleus »
  - spécialisation d'interfaces ou *skins* (souvent POSIX)
- Modèle de programmation
  - espace noyau → le plus souvent pour les pilotes RTDM
  - espace utilisateur standard par les « skins »
  - L'application Linux est *répartie* sur les deux noyaux (Linux et Xenomai) ou « domaines »
- Couches d'abstraction d'architecture hôte SAL/HAL
  - rassemble les dépendances matérielles de la cible
- Multi plates-formes
  - *arm, blackfin, i386, ia64, powerpc32, powerpc64, nios2, sh4*

Architecture générale

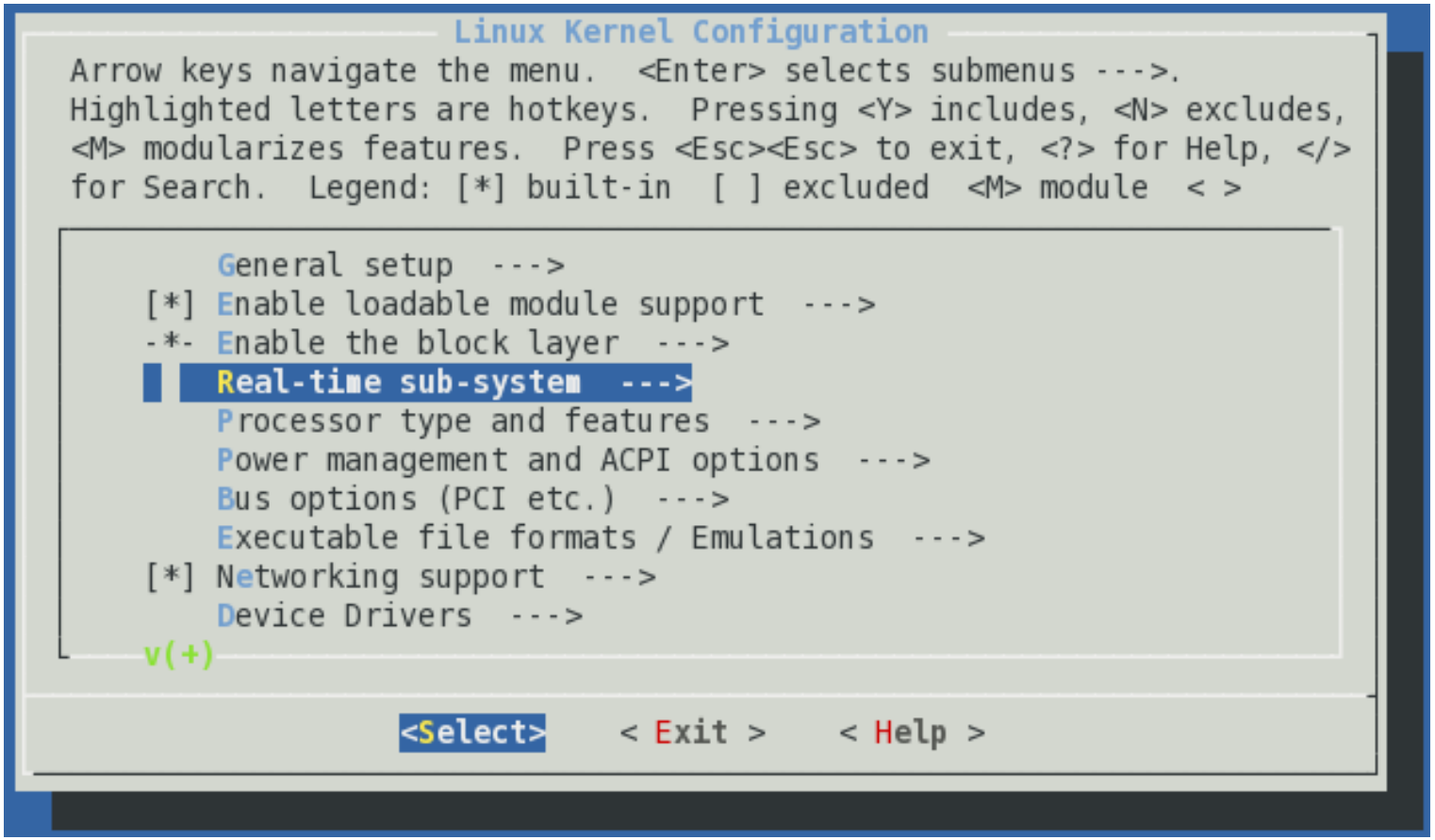


Utilisation des domaines



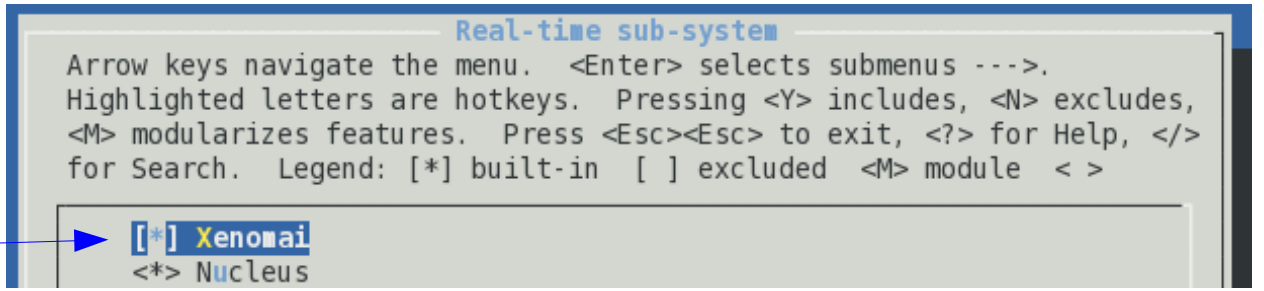
Configuration du noyau

- Nouvelle entrée *Real-time sub-system* dans le menu principal



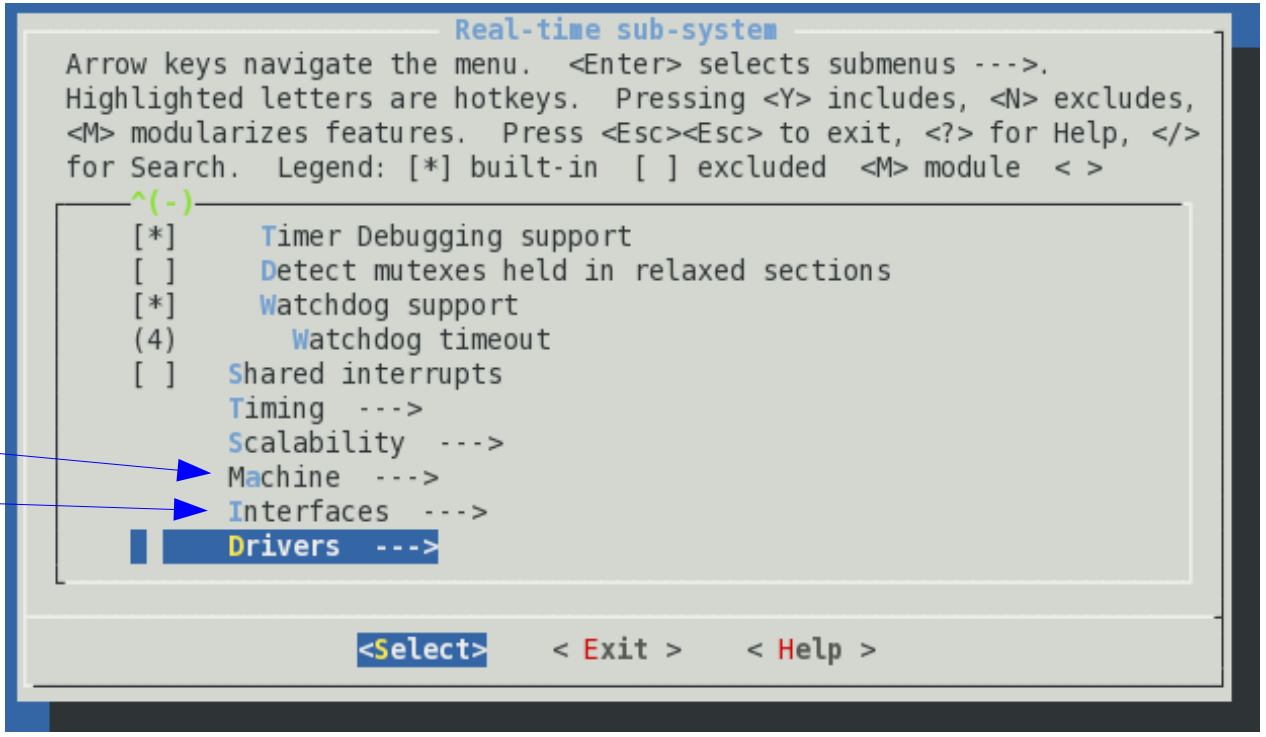
Configuration du noyau, suite

sélection « statique »

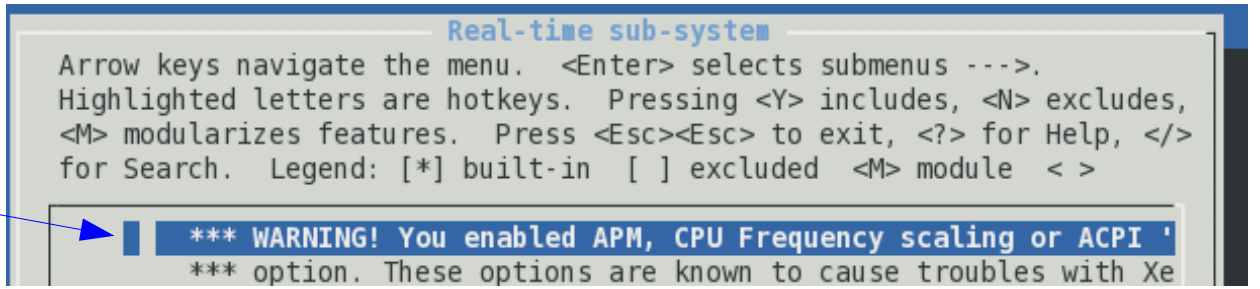


spécificités matérielles

« skins »



incompatibilité de config



Test de fonctionnement

- On peut utiliser `latency` fourni avec Xenomai
  - tâche périodique à partir de 100 µs (1ms sur RPi)
  - affichage du jitter (min, avg, max, best, worst)
  - « Overrun » et migration de mode (msw)
- Exemple sur RPi

```
# echo 0 > /proc/xenomai/latency
# latency
== Sampling period: 1000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
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|      2.000|      5.000|     31.000|       0|     0|      2.000|     31.000
RTD|      2.000|      5.000|     21.000|       0|     0|      2.000|     31.000
RTD|      2.000|      5.000|     19.000|       0|     0|      2.000|     31.000
...
```

Test de fonctionnement

- On peut charger le système Linux avec la commande hackbench ou avec un « flood ping ».

```
# hackbench -p -g 20 -l 1000
# ping -f adresse_RPi
```

- On note la variation sur le domaine Xenomai

```
RTD|      2.000|      5.000|     24.000|        0|        0|      2.000|     25.000
RTD|      2.000|      5.000|     25.000|        0|        0|      2.000|     25.000
RTD|      3.000|     21.000|     36.000|        0|        0|      2.000|     36.000
RTD|      3.000|     21.000|     36.000|        0|        0|      2.000|     36.000
RTD|      3.000|     21.000|     32.000|        0|        0|      2.000|     36.000
RTD|      3.000|     21.000|     33.000|        0|        0|      2.000|     36.000
RTD|      2.000|     19.000|     33.000|        0|        0|      2.000|     36.000
```

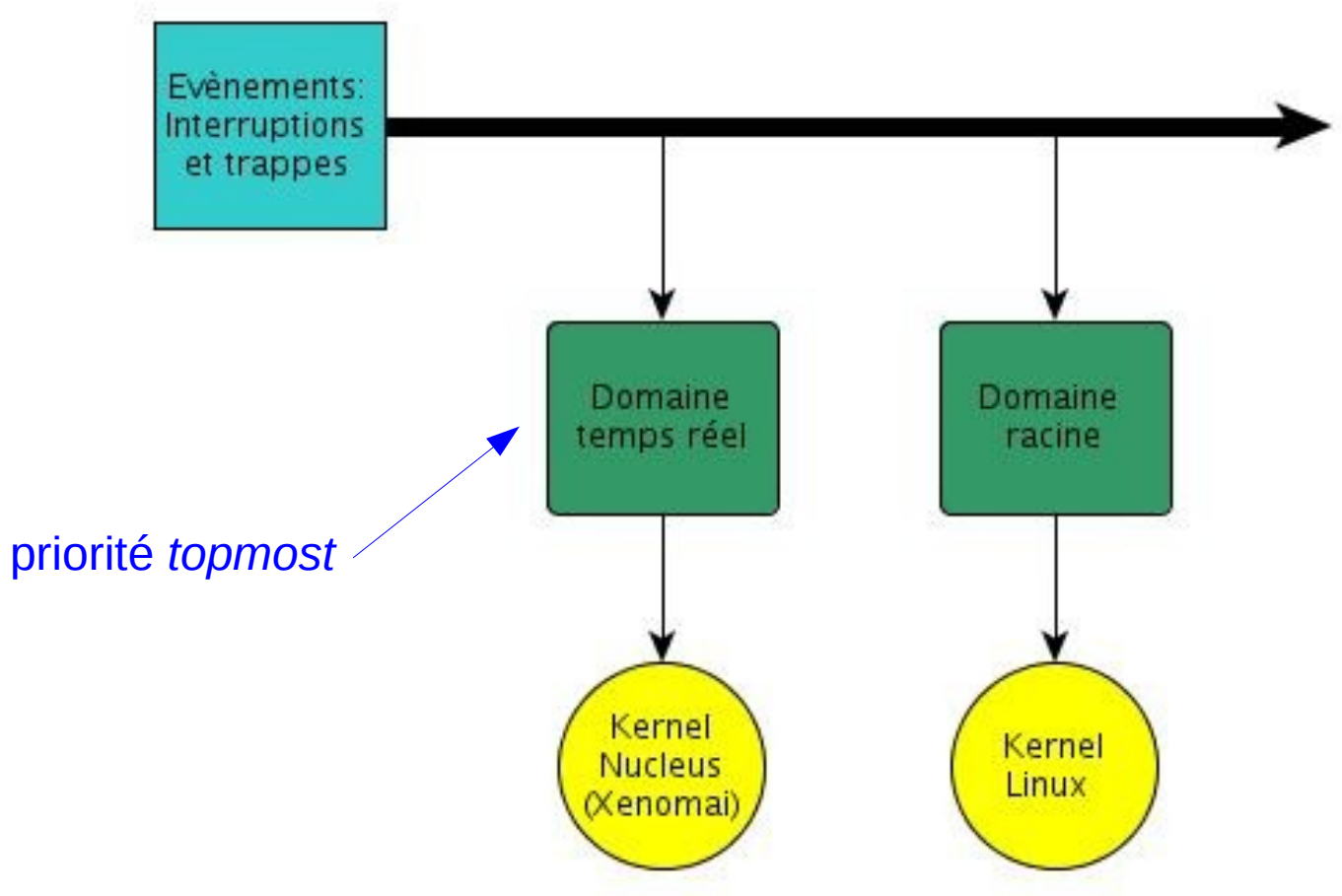
- On note le retour à la valeur moyenne lors de l'arrêt de la charge

```
RTD|      2.000|      5.000|     21.000|        0|        0|      2.000|     36.000
RTD|      2.000|      5.000|     23.000|        0|        0|      2.000|     36.000
...
```

ADEOS

- **A**daptative **D**omain **E**nvironment for **O**perating **S**ystems
- Proposé par Karim Yaghmour en 2002 pour *contourner* le brevet logiciel de RTLinux / FSMLabs
- Virtualisation de ressources matérielles → cohabitation de plusieurs noyaux sur une même machine
- Basé sur un principe de « pipeline d'interruptions » (I-Pipe)
- Organisation en *domaines* avec *priorité* (*topmost* pour Xenomai) → ordonnancement
  - Composant logiciel basé sur un micro-noyau
  - Notifié par ADEOS lors d'événements (interruption, trappe, appel système, ...)
- Existe uniquement sous la forme d'un « patch » pour le noyau Linux

I-Pipe ADEOS


Evènements:
Interruptions
et trappes
Domaine
temps réel
Domaine
racine
priorité *topmost*
Kernel
Nucleus
(Xenomai)
Kernel
Linux

Principe du I-Pipe

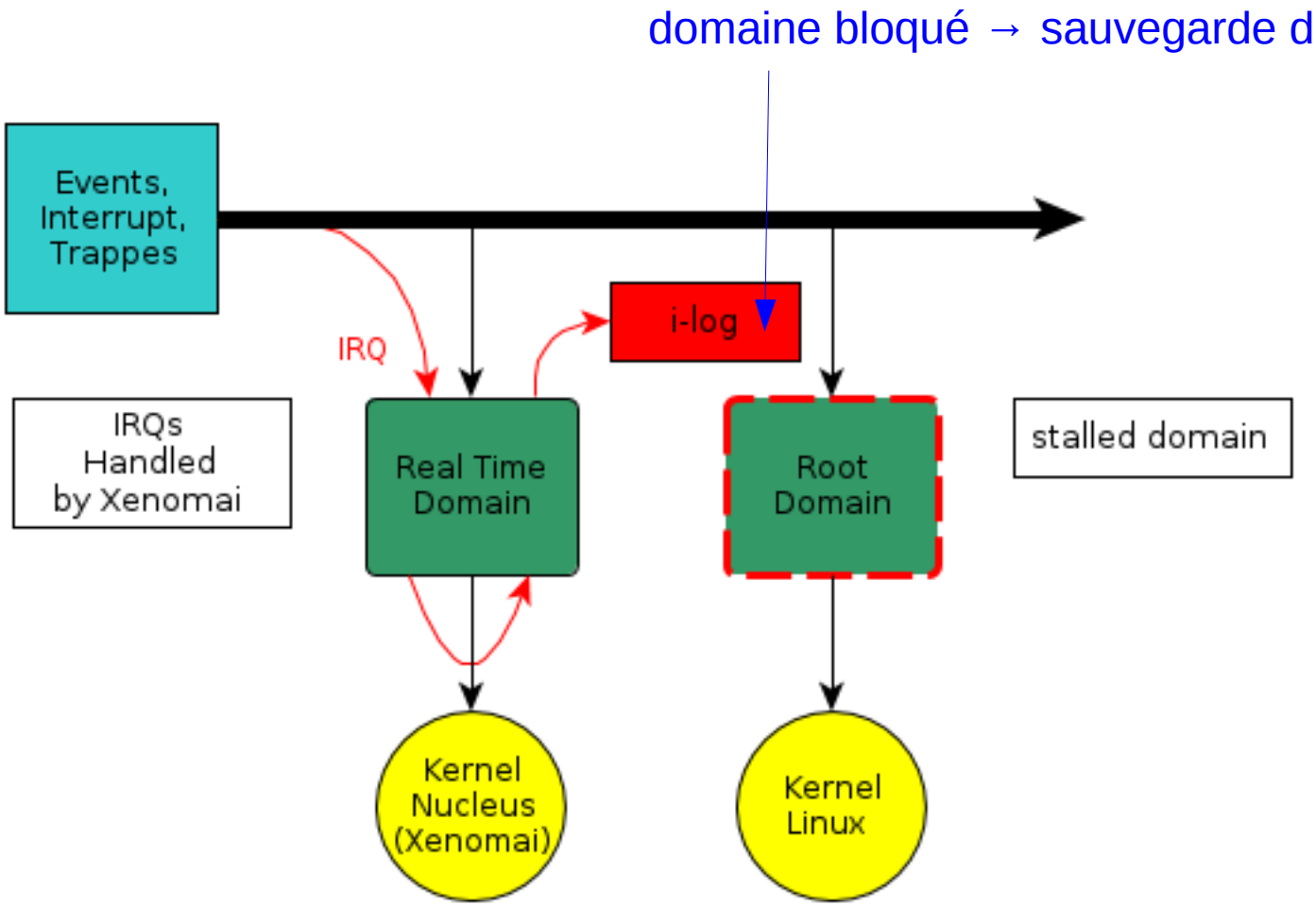
- Le I-Pipe est démarré par le domaine « racine » (Linux) → initialisation dans `start_kernel()`
- Le contrôleur d'interruption matériel est *uniquement* piloté par ADEOS
- Pour chaque domaine, masque d'interruption au niveau *logiciel* et non plus au niveau *matériel*
- Pour chaque interruption, le domaine peut :
  - Accepter → traitement immédiat
  - Ignorer → traitement *différé* → *blocage* du domaine
  - Écarter → propagée au domaine suivant
  - Terminer → non propagée
- La réception d'interruption *matérielle* est possible pendant le *blocage* du domaine → pas de perte d'IRQ :-)

Principe du I-Pipe, suite

- Traitement des interruptions (x86)
  - `cli` (CLear Interrupt) → `ipipe_stall()`
  - `sti` (SeT Interrupt) → `ipipe_unstall()`
- Propagation au domaine suivant
  - `ipipe_propagate_irq()`

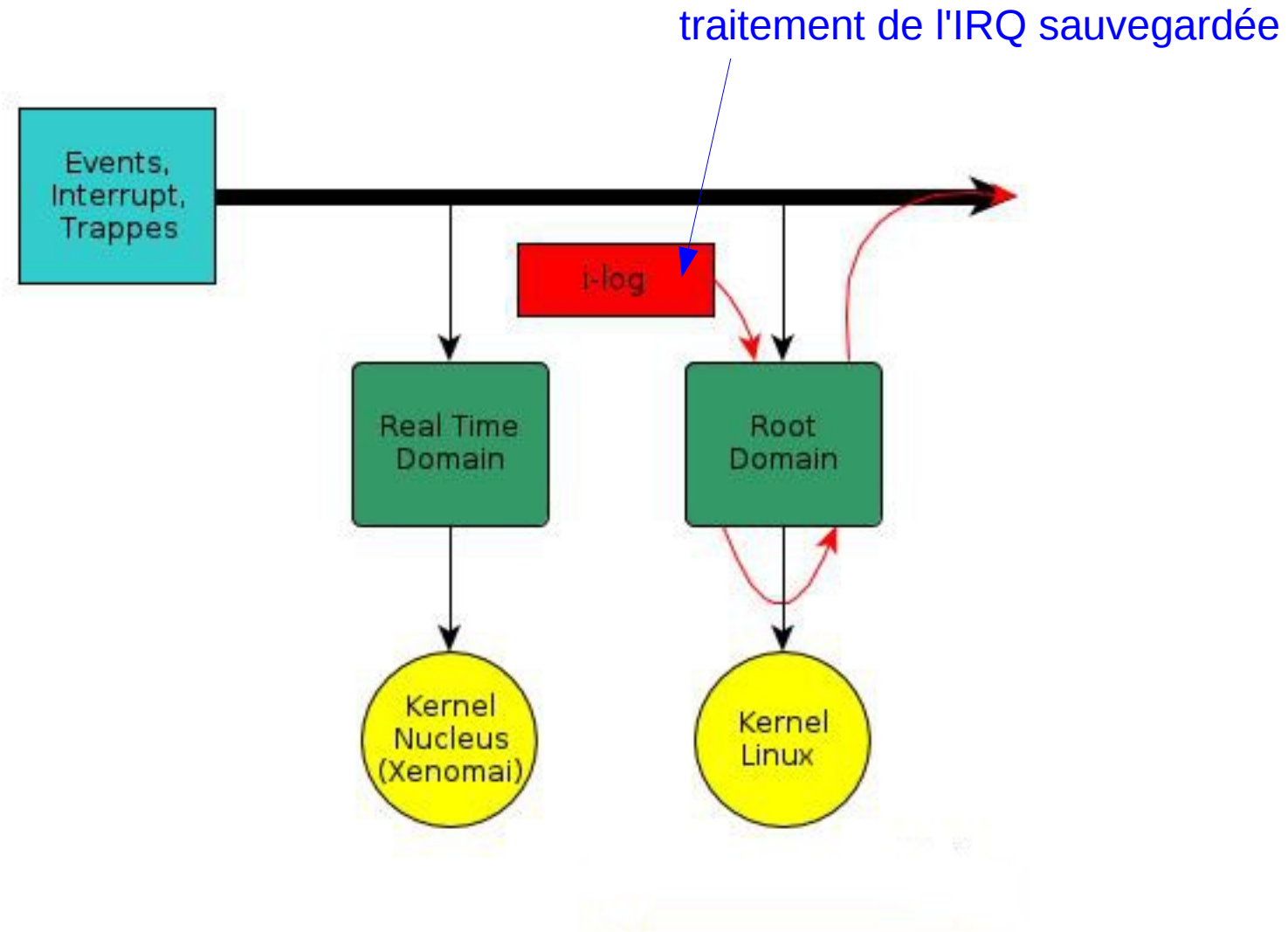
Blocage du domaine racine

`ipipe_stall_pipeline_from()`



Déblocage du domaine racine

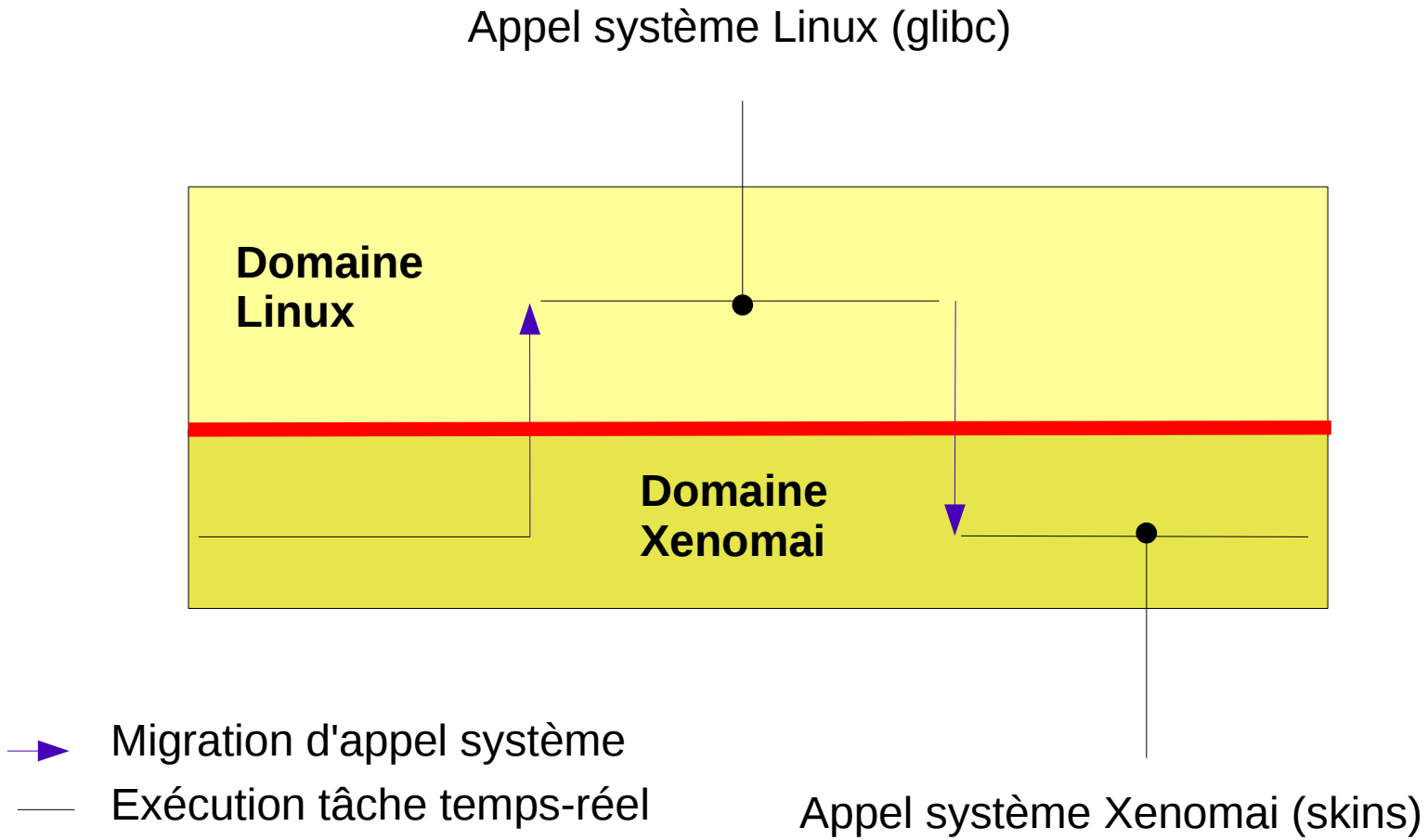
```
ipipe_unstall_pipeline_from()
```



Domaines d'exécution

- Dualité d’ordonnancement sur 2 domaines
  - Domaine Xenomai, déterministe
  - Domaine Linux, *non* déterministe
- Exécution d'une tâche temps-réel
  - Mode primaire (domaine Xenomai)
  - Mode secondaire (domaine Linux)
- Migration de modes
  - Automatique sur appels système
  - Interceptée par le signal `SIGDEBUG`
  - Source de « jitter » !

Migration de domaine sur syscall



Conclusions

- La RPi n'est pas une carte industrielle mais permet de tester des fonctionnalités disponibles sur d'autres cartes plus performantes (Cortex-A8/A9, i.MX6)
- Xenomai est beaucoup mieux adapté à l'embarqué que PREEMPT-RT
- La mise en place reste cependant plus délicate et nécessite une architecture particulière pour l'application ainsi que des pilotes dédiés !

Bibliographie

- http://www.framboise314.fr/38-millions-de-raspberry-pi-vendus
- http://buildroot.uclibc.org
- http://www.yoctoproject.org
- http://www.xenomai.org
- https://rt.wiki.kernel.org
- http://www.blaess.fr/christophe/livres/solutions-temps-reel-sous-linux
- http://www.blaess.fr/christophe/category/raspberry-pi
- http://www.linuxembedded.fr/2013/01/preempt-rt-sur-raspberry-pi
- http://www.linuxembedded.fr/2013/07/rtems-sur-raspberry-pi
- http://www.linuxembedded.fr/2014/07/electronique-simple-pour-gpio
- http://xenomai.org/2014/06/porting-a-posix-application-to-xenomai
- http://www.xenomai.org/documentation/trunk/html/api
- http://francois.touchard.perso.luminy.univ-amu.fr/3/LinuxAvance/C1-intro.pdf
- http://francois.touchard.perso.luminy.univ-amu.fr/3/LinuxAvance/C4-RTDM.pdf
- https://delog.wordpress.com/2014/09/16/embedded-linux-system-for-raspberry-pi-with-yocto-project/