open wide
INGENIERIE

Linux et le temps réel

Jérémy Rosen (jeremy.rosen@openwide.fr)

Septembre 2013

Présentation Open Wide

- SSII/SSLL créée en septembre 2001 avec THALES et Schneider
- Indépendante depuis 2009
- Environ 120 salariés sur Paris, Lyon et Toulouse
- Industrialisation de composants open source
  - Développement
  - Formation
  - Expertise
- Trois activités :
  - **OW S**ystème d'**I**nformation (Java/PHP)
  - **OW O**utsourcing: hébergement
  - **OW I**ngénierie: informatique industrielle

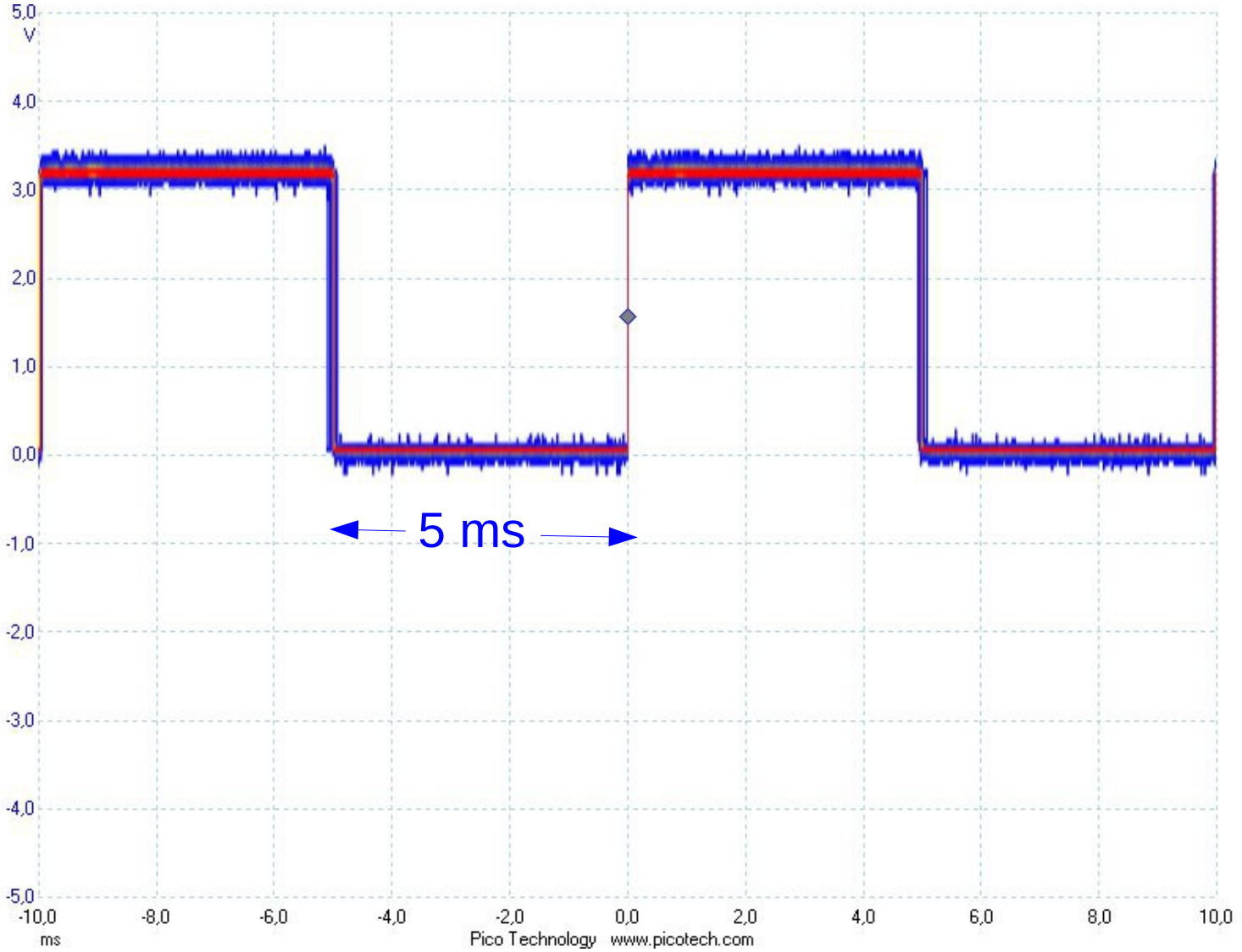
open wide
INGENIERIE

Qu'est que le temps réel

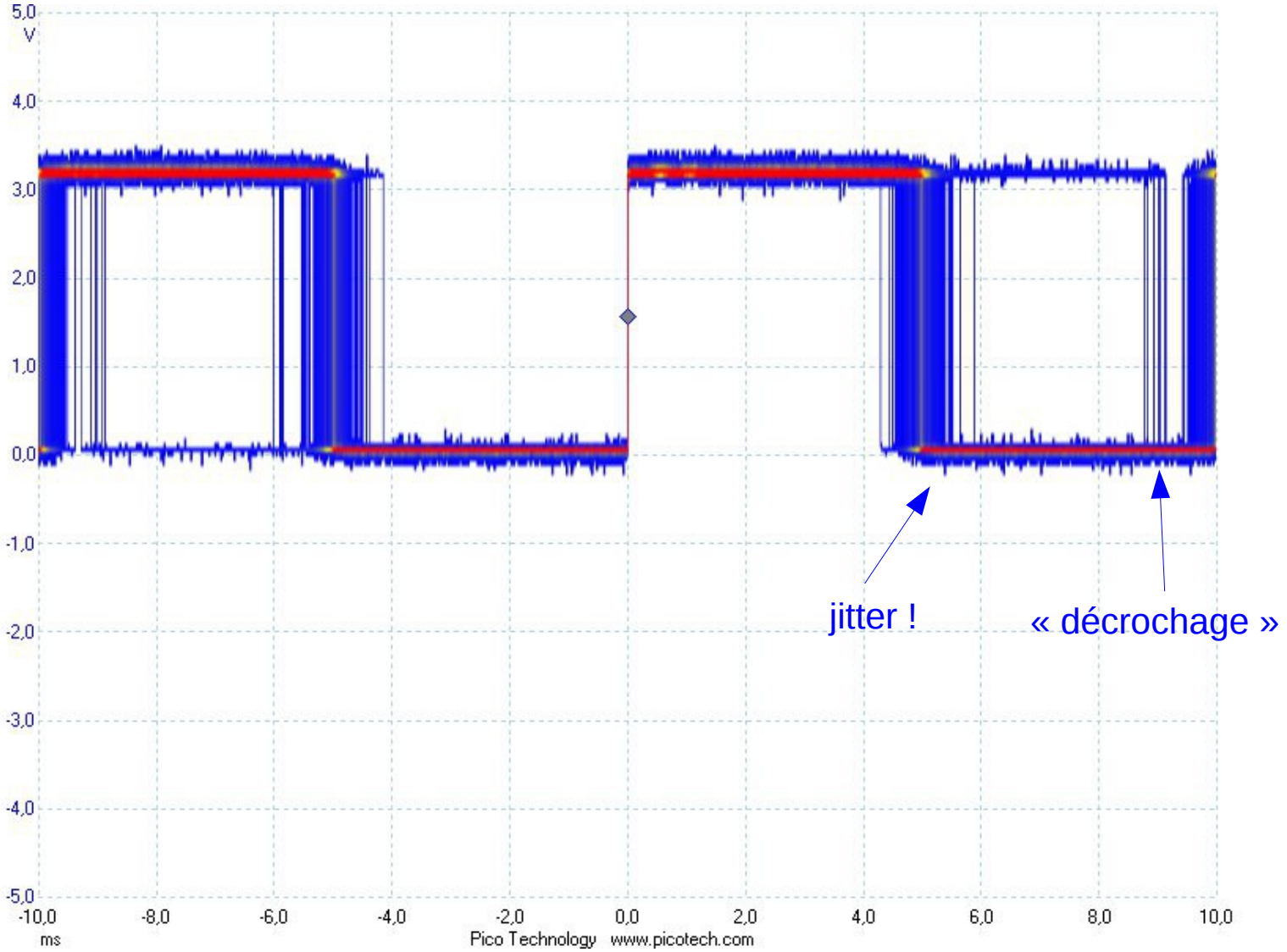
Définition

- Les spécifications d'un logiciel temps réel contiennent des contraintes de temps de réponse fortes.
  - Il y a un temps maximum imposé entre un événement et son traitement
  - La validité du résultat est influencée par le temps de traitement
  - Le logiciel doit avoir une vitesse de traitement adapté au processus physique qu'il gère
- Le temps de réponse doit être *garanti*. En particulier en cas de charge du système.
  - Ajouter des ressources au système n'es pas une solution
  - Les algorithmes ne peuvent pas se dégrader en durée infinie... ou indéfinie
- Le compromis performance/respect d'échéance est fortement biaisé vers le respect des échéances.
  - Un logiciel temps-réel a un moins bon débit qu'un logiciel non contraint.

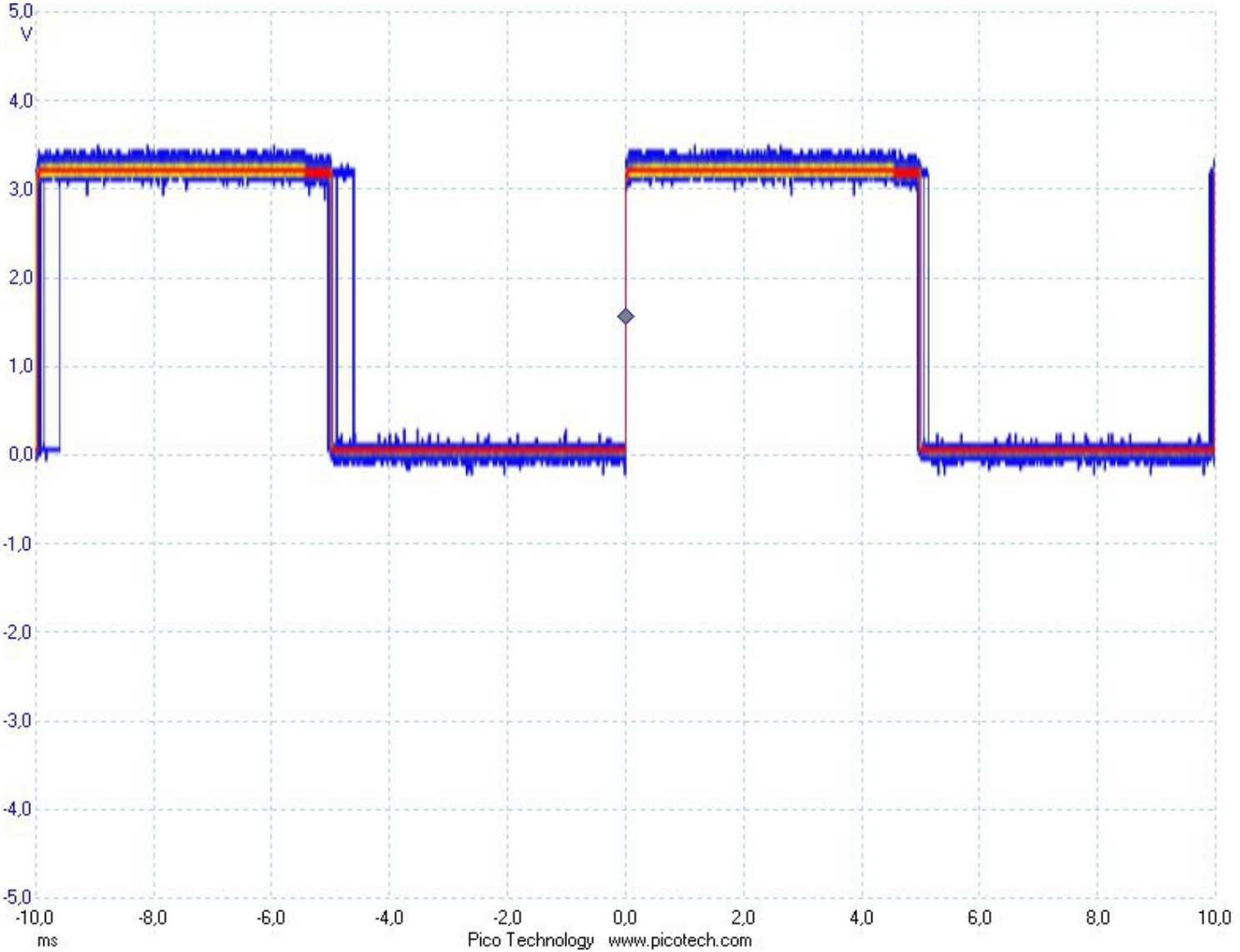
Tâche périodique, Linux non chargé



Système chargé par « flood ping »



Flood ping avec Xenomai



Les deux types de temps réel

- Le temps réel **strict (hard)**
  - Rater une échéance est un bug.
  - Exemple : ABS.
- Le temps réel **souple (soft)**
  - Rater une échéance dégrade la qualité du résultat.
  - On estime souvent la qualité en pourcentage d'échéances respectés.
  - Exemple : Vidéo.

Les cousins du temps réel

- Les systèmes **critiques**
  - La criticité est estimée par les conséquences de la perte du système.
  - Exemples : DO178/SIL
  - De nombreuses normes et processus permettent d'évaluer et de valider ces systèmes
- Les systèmes **certifiés**
  - Les algorithmes doivent être prouvés correct statiquement.
  - Dans le cas du temps réel, les échéances doivent donc pouvoir être vérifiés au niveau algorithmique.

Logiciel embarqué = temps réel ?

- Les applications embarquées historiques étaient TR
- Les systèmes d'exploitation embarqués propriétaires sont TR (VxWorks, …) → RTOS
- L'apparition (généralisation) des OS libres dans l'industrie et dans l'embarqué en général modifie la donne !
  - Linux est utilisable dans l'industrie
  - Linux n'est pas TR
  - Linux peut être modifié pour être TR (PREEMPT-RT, Xenomai)
  - Il existe des systèmes TR légers et libres (RTEMS, FreeRTOS, ...)

GPOS / RTOS

- GPOS (Windows, Linux, ...)
  - Lissage des priorités (dynamique) → complexité
  - Grand nombre de tâches concurrentes (> 200 sur un PC Linux « inactif »)
  - 15M lignes de code pour le noyau Linux 3.x !
- RTOS
  - Prévisible : évaluer le pire des cas
  - Déterministe : faible influence de la charge
  - Pas ou peu de notion de « performances moyennes » Tout est dimensionné pour le pire cas
  - Peu de tâches concurrentes
  - Souvent mono-process
  - Léger (< 100K lignes de code pour FreeRTOS)
  - Certification possible de certains RTOS dédiés

open wide
INGENIERIE

Les différentes approches des RTOS

Ordre de présentation

- Distance au hardware croissante.
- Expertise logicielle nécessaire décroissante.
- Performances RT décroissantes.
- Taille de code total croissantes.
- Confort de développement croissant.
- Difficulté de certification croissante.
- Support matériel disponible croissant.
- Existant réutilisable croissant.

Bare metal

- « Bare metal » = pas de système d'exploitation
- Application exécutée *directement* sur matériel
- Système critique ou très réduit
- Faible portabilité
- Facile à évaluer/vérifier de bout en bout
- Généralement pas de MMU/cache/IRQ
- Le confort de développement et l'outillage sont très limités

Approches synchrone

Modélisation statique du comportement

- Pas d'interruption ni d'allocation dynamique.
- Tout l'ordonnancement est fixé à la compilation.
- Nécessité de « séparer » les applications à plusieurs niveaux (partitionnement)
  - Spatial : espace mémoire.
  - Temporel : ordonnancement.
- Toutes les communications sont déterminées à la compilation.
- Convient aux systèmes *critiques.*
- Langages dédiés, génération de code (ESTEREL, SIGNAL, SynDEx)
- Normalisation ARINC 653.

Quelques RTOS ARINC 653

- VxWorks 653 (Wind River)
- LynxOS-178, LynxOS-SE (LynuxWorks)
- INTEGRITY-178 (Green Hills)
- LithOS, basé sur XstratuM (hyperviseur TR de l'UPV)
- POK (Telecom ParisTech, Julien Delange), utilisé pour la recherche

Approches asynchrone

Version temps réel de l'approche classique

- Les tâches de priorité TR ne peuvent être interrompues que par des tâches de plus haute priorité
  - Pas de priorité dynamiques pour les tâches TR.
  - Pas d'interruption par le kernel (sauf héritage de priorité).
  - Les tâches s'activent instantanément lorsqu'elles deviennent RUNNABLE.
  - Généralement un scheduler différent (FIFO ou RR)
- Nécessite de la préemption à tous les niveaux
  - Drivers spécifiques (interruptible dans les IRQ).
  - Peu/pas de sections critiques ou sections critiques avec gestion de priorité.
- Modélisation complexe (nombreux cas de préemption, MMU, IRQ, Caches)

eCos

- **e**mbeddable **C**onfigurable **OS** (CYGNUS 1997)
- Supporte de nombreux CPU (16), 32 et 64 bits
- Empreinte mémoire de 10 à 100 Ko
- Outils de configuration avancé, gestion de « packages »
- Version « pro » par eCosCentric
- Utilisé dans le multimédia :
  - http://www.ecoscentric.com/ecos/examples.shtml

RTEMS

- RTEMS = **R**eal **T**ime **E**xecutive for **M**ultiprocessor **S**ystems
- Initialement « Missile Systems » puis « Military Systems »
- Exécutif temps réel embarqué diffusé sous licence libre (GPL avec exception)
- Ce n'est pas exactement un système d'exploitation car l'application est « liée » au noyau → *un seul* processus mais *plusieurs* « threads »
- Programmation C, C++, Ada
- Plus de 100 BSP disponibles pour 20 architectures
- API RTEMS « classique » ou **POSIX**
- Utilisé par EADS Astrium, ESA, NASA, NAVY, ...

open wide
INGENIERIE

Linux et le temps réel

Linux comme (RT)OS

- Réservé aux systèmes *complexes*
  - 32 bits minimum
  - Gestion complexe de la mémoire (MMU, pagination+segmentation)
  - Empreinte mémoire importante: 2 Mo pour µCLinux (MMU-less), 4 Mo pour Linux
  - Consommation mémoire vive : 16 Mo minimum
- Incompatible avec les systèmes *critiques*
- Souvent utilisé pour les outils, les simulateurs et architectures « mixtes » (banc de test)

Linux (vanilla) et le Temps Réel

- Linux est un UNIX, donc pas un système temps réel
- Pas de préemption « complète » en mode noyau → un processus ne peut être interrompu dans une routine de traitement d'interruption (top half)
- Préemption par l'ordonnanceur
  - Sur interruption *timer*
  - Fréquence *timer* fixe (constante *HZ* = 1-10 ms) → précision de l'ordonnanceur (granularité)
  - Latence (non-garantie) de l'ordre de la milliseconde.
- 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) mais non-garanties

Extensions TR pour Linux

- L'utilisation de Linux comme RTOS est souvent intéressante
  - Approche hybride avec quelques tâches TR
  - On conserve le confort d'un système classique
- Deux approches possibles :
  - Modifier le noyau Linux afin d'améliorer ses performances TR (PREEMPT-RT)
  - Ajouter un « co-noyau » TR qui partage le matériel avec le noyau Linux (RTLinux, RTAI, Xenomai) → approche « virtualisation »

PREEMPT-RT

- Branche expérimentale pour la version 2.6 et 3.x, voir https://rt.wiki.kernel.org
- Initié par Ingo Molnar, contributeur majeur du noyau
- Surtout utilisé sur x86 et des processeurs performants (nécessite 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 (mais des contraintes de développement existent)
- Permet de garantir 100 µs de jitter (sur x86, dépends largement du matériel)

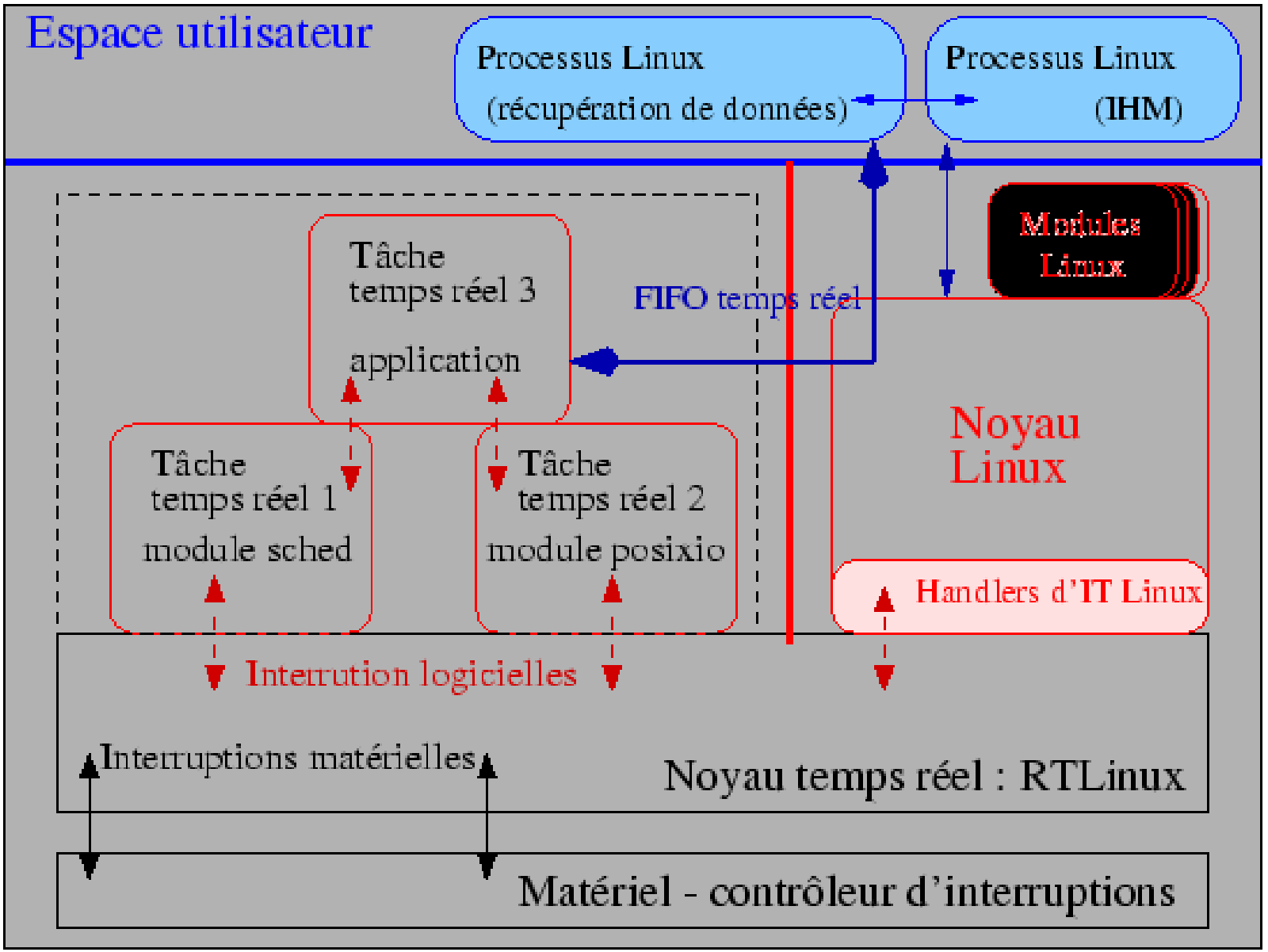
Linux avec co-noyau

- 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
- Se rapproche de la technique de « para-virtualisation » des *hyperviseurs* (adaptation de l'OS)
- Excellentes performances TR
  - ordonnanceur spécifique *indépendant*
  - sous-système temps-réel bien délimité
  - jitter maximal de l’ordre de 10 µs sur Atom/x86 !

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 initiale RTLinux



RTAI

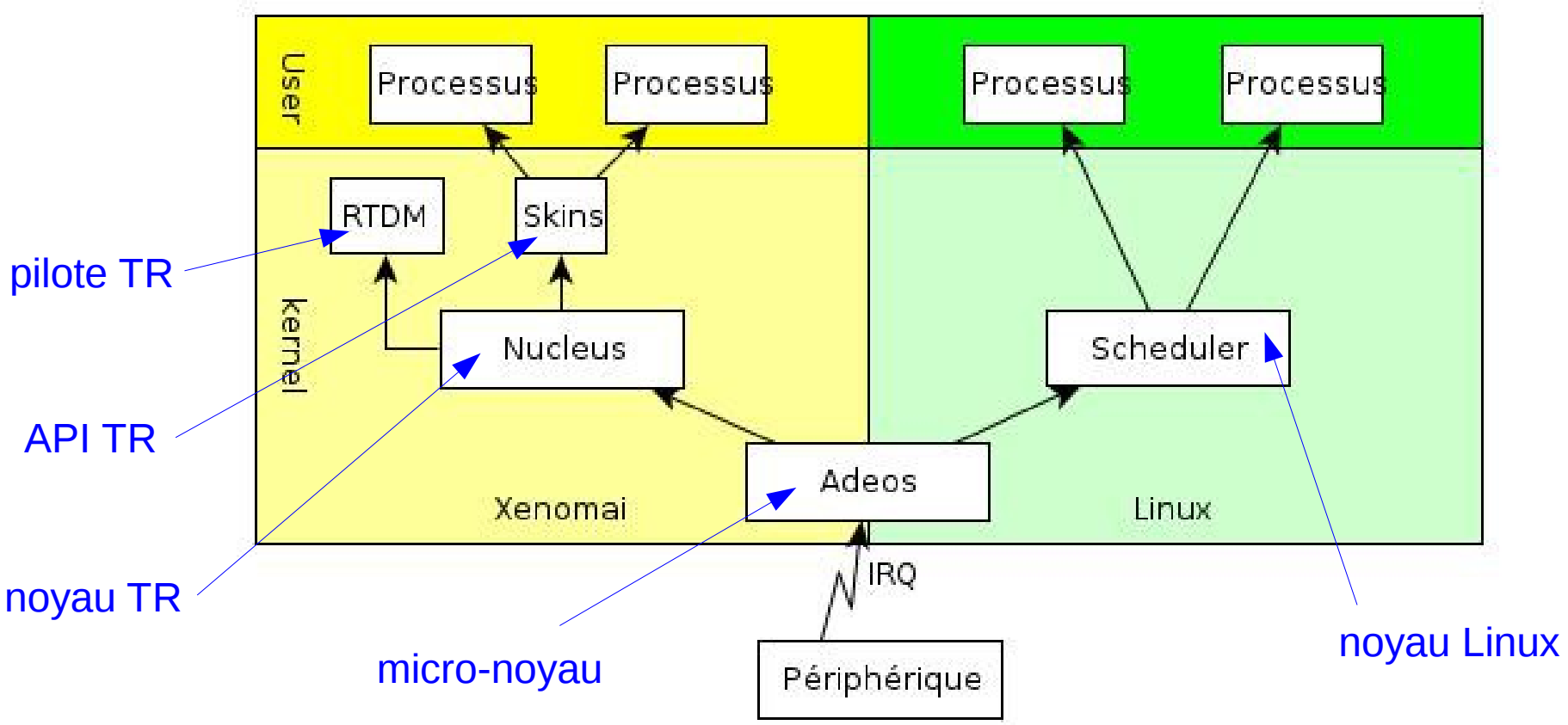
- **R**eal **T**ime **A**pplication **I**nterface
- Un « fork » de RTLinux développé au DIAPM de l'école polytechnique de Milan → Dipartimento di Ingegneria Aerospaziale (Paolo Montegazza)
- Utilisé au DIAPM pour des travaux d'enseignement et de recherche
- Quelques utilisations industrielles
- Position douteuse / brevet logiciel FSMLabs
- Toujours actif mais peu d'évolution → version 3.8 en février 2010, 3.9 en août 2012

Xenomai

- Xenomai est un sous-système temps-réel de Linux
  - Programmation de tâches en espace utilisateur
  - API d'application et de pilotes temps réel (RTDM) dédiées
- Supporte de nombreuses architectures
- Dispose de « skins » permettant d'émuler des API temps réel (**POSIX**, VxWorks, VRTX, uITRON, ...)
- Plus complexe à mettre en œuvre que PREEMPT-RT mais performances 5 à 10 fois supérieures
- Licence GPL (cœur), LGPL (interfaces, espace utilisateur)

Xenomai, architecture

- Xenomai utilise un micro-noyau (ADEOS) pour partager le matériel avec le noyau Linux



Android

- Difficile de ne pas l'évoquer !
- Système d'exploitation « libre » basé sur un noyau Linux modifié par Google
- Pas de support RT sur un Android de base
- Peut utiliser du code natif (via NDK)
- Utilisable en guest sur un RTOS ?
- PREEMPT-RT porté pour Android par OpenWide (intégré au projet Android-x86)

Questions ?