État de l'art des systèmes embarqués

Pierre Ficheux (pierre.ficheux@openwide.fr)

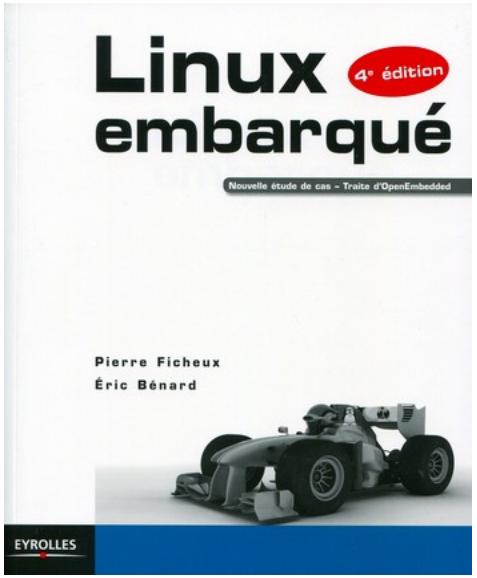
Septembre 2013

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
- Trois activités :
  - **OW S**ystème d'**I**nformation (Java/PHP)
  - **OW O**utsourcing: hébergement
  - **OW I**ngénierie: informatique industrielle

Présentation PF

- Ingénieur Arts et Métiers + Sup'Aéro
- Utilisateur de logiciels libres depuis 1989
- Utilisateur de Linux depuis 1992
- Auteur des 4 éditions de l'ouvrage « Linux embarqué » (Eyrolles), 4ème édition parue en juin 2012 avec E. Bénard
- Auteur GNU Linux Magazine et Open Silicium
- CTO Open Wide Ingénierie, enseignant EPITA



open wide INGENIERIE

Programme

- Introductions aux systèmes embarqués
- Systèmes temps réel
- Systèmes critiques
- Logiciel libre dans l’embarqué
- OS libres pour l’embarqué
- Extension temps réel pour Linux
- Les outils (Buildroot, OpenEmbedded, …)
- Quelques statistiques du marché
- Démonstrations

open wide
INGENIERIE

Introduction aux systèmes embarqués

Système / logiciel embarqué

- Un système peut être l'association matériel + logiciel
- Logiciel utilisé dans un *équipement industriel* ou un *bien de consommation*
- On dit aussi logiciel *dédié* ou *intégré* (embedded software)
- L'équipement est valorisé pour son coté *fonctionnel* et *non pas* pour le logiciel !
- Un bon logiciel embarqué *doit savoir se faire oublier* !
- On parle parfois de logiciel *enfoui* ou *profondément enfoui* (deeply embedded)
- Très vaste domaine d'application !

Logiciel embarqué

- Ciblé: limité aux fonctions pour lesquelles il a été créé
- Fiable et sécurisé: autonomie, processus sensibles
- Longue durée de vie: militaire, spatial (20, 30 ans ou +)
- IHM spécifique ou déportée :
  - Affichage réduit ou inexistant
  - Pas forcément de clavier/souris
- Optimisé: empreinte mémoire, performances
  - Gagner quelques € sur le matériel peut être important
  - Processeurs (encore) moins puissants que dans un « PC »
  - Temps d'attente peu acceptable pour l'utilisateur non spécialiste
- Pas obligatoirement un OS (taille, criticité)

open wide INGENIERIE

Domaines d'application

- 1960-70 : remplacer/compléter des systèmes analogiques (spatial)
- 1980 : RTOS (Real Time OS) génériques
- 2000 : OS libres, grand public
- Domaines historiques/industriels
  - Militaire, spatial (RTOS/360, VRTX sur Hubble)
  - Contrôle de processus industriel
  - Transport : AUTOSAR/OSEK, ARINC 653 → certification (DO-178, ...)
  - Internet/Telecom : routeurs, PABX (Chorus)
- « Nouveaux » domaines
  - Multimédia : audio, vidéo, TV, automobile (GENIVI), Téléphonie (Android), médical

Les « nouveaux » domaines

- Équipement grand public jusque la *isolé* (multimédia, domotique, …)
- Téléphonie → 1 milliards de téléphones Android !
- *Infotainment* transport: automobile, aéronautique
  - Ajout de fonctions *communicantes* → utilisation de protocoles standards de type IP et dérivés (HTTP, DHCP, etc.)
  - Difficile d'intégrer ces couches dans des logiciels embarqués propriétaires → utilisation d'un OS
- « Boite noire » dédiée à un ensemble de fonctions (PABX, passerelle médicale, set-top box avec services étendus)

Logiciel ou OS embarqué?

- Définition d'un OS

  « Assure un ensemble de services en présentant aux utilisateurs une interface mieux adaptée à leurs besoins que celle de la machine physique »

- OS parfois trop complexe pour certains systèmes très simples
- Dès que le nombre de tâches/services devient important, l'utilisation d'un OS est rentable (tendance actuelle)
- Classification RTOS/**GP**OS (General Purpose)

Bare metal

- « Bare metal » = pas de système d'exploitation
- Application exécutée *directement* sur matériel
- Système critique ou très réduit
- Faible portabilité
- Éventuellement un simple « exécutif » temps réel (Sysgo)
- Utilisation d'un (RT)OS
  - Plus de souplesse et portabilité mais plus d'empreinte mémoire
  - Certification possible de certains OS : VxWorks 653 (Wind River), LynxOS-178 / LynxOS-SE (LynuxWorks), INTEGRITY-178 (Green Hills)

Avantages d'un OS

- Affranchit le développeur d'un travail d'adaptation au matériel pour les interfaces de base (PCI, USB, Ethernet...)
- Permet de bénéficier des dernières avancées technologiques et de faire évoluer le système: protocoles réseau, multimédia, etc.
- Recrutement des développeurs plus simple (Linux, Android) !
- Utilisation de matériel « standard » (suivant l'OS)
- Focalisation sur le métier !

Inconvénients d'un OS

- Empreinte mémoire (donc coût matériel)
- Consommation d'énergie (nombreux travaux dans ce domaine, dont Android)
- Obsolescence des composants
- Criticité (sauf OS spécialisés)
- Perte de maîtrise du système (Boeing 787...)

open wide
INGENIERIE

Systèmes temps réel

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 »
  - Peu de tâches concurrentes
  - Léger (< 100K lignes de code pour FreeRTOS)
  - Certification possible de certains RTOS dédiés

Les différents types de temps-réel

- Temps-réel *dur* : un retard dans l'obtention du résultat rend le système inutile ou dangereux (pilotage matériel, embarqué historique)
- Temps-réel *mou* : un retard dans l'obtention du résultat est acceptable (industriel, multimédia)
- La plupart des systèmes temps-réel *modernes* sont « hybrides » (exemple : Linux/Xenomai)
  - Certaines tâches sont temps-réel (dur)
  - D'autres ne le sont pas ou peu (mou)
- Frontière mou/dur parfois difficile à définir !
- La puissance CPU masque les problèmes de TR :-(

Quelques RTOS célèbres

- VxWorks de WindRiver
- QNX, UNIX *like*
- LynxOS de LynuxWorks
- Nucleus (Mentor Graphics)
- µC/OS (micro-C OS) et µC/OS II: RTOS pour micro-contrôleurs
- VRTX
- **ultron**
- pSOS
- *Windows CE / XP Embedded, Windows Mobile*
- **RTEMS**
- **FreeRTOS**

open wide
INGENIERIE

Systèmes embarqués « critiques »

Approches asynchrone / synchrone

- Asynchrone
  - Système TR classique
  - Interaction avec l'environnement (interruption), dérive d'horloge entre calculateurs
  - Modélisation complexe (nombreux cas de préemption)
- Synchrone
  - Modélisation statique du comportement
  - Pas d'interruption ni d'allocation dynamique
  - Convient aux systèmes *critiques*
  - Langages dédiés, génération de code (ESTEREL, SIGNAL, SynDEx)
  - Exemple du métro automatique parisien M14/M1

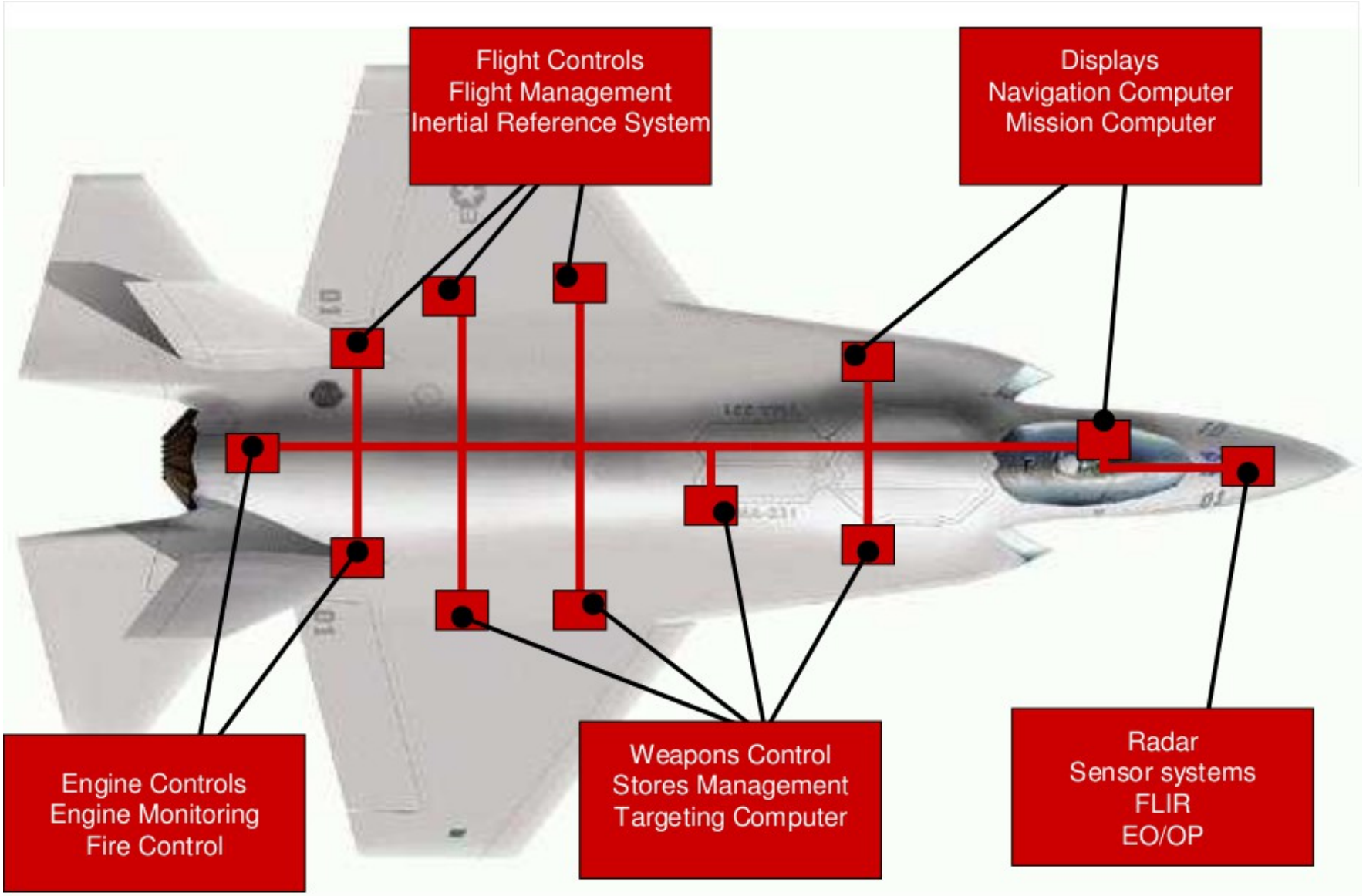
Certification et normes

- La certification concerne
  - Tests fonctionnels et couverture de code
  - Pratiques dangereuses (ex : allocation dynamique)
  - Traçabilité
  - Documentation
- Exemples de normes
  - DO178 : aéronautique
    - 5 niveaux de criticité (A, B, C, D, E) allant de « catastrophic » à « no effect »
  - CEI 61508 : sûreté de fonctionnement en général
    - SIL (System Integrity Level) de 4 à 1
  - CEI 60601 : médical

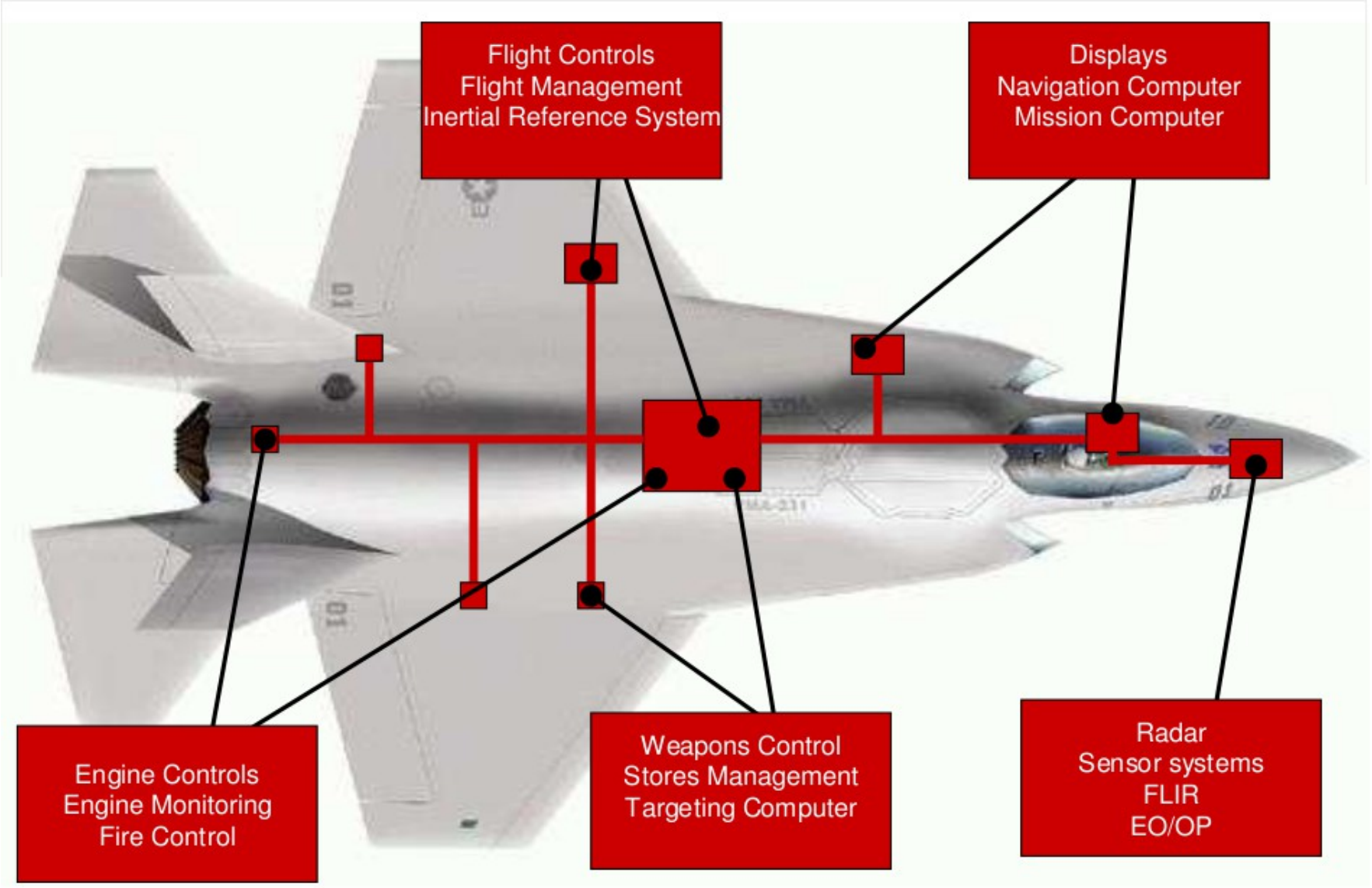
Utilisation de standards

- Certification par rapport aux normes
- Évolution de l'architecture des systèmes
  - Omniprésence de l'électronique et du logiciel
  - Moins de calculateurs et plus de fonctionnalités (plusieurs applications sur un calculateur)
  - Réduction des composants, du câblage, donc du coût
  - Sous-traitance
  - De plus en plus de trafic, mais autant (plus) de sécurité requise !
- Exemple : architecture *IMA* (Integrated Modular Avionics) pour A380, A350, A400M, Falcon 900, ...

Architecture classique



Architecture « IMA »

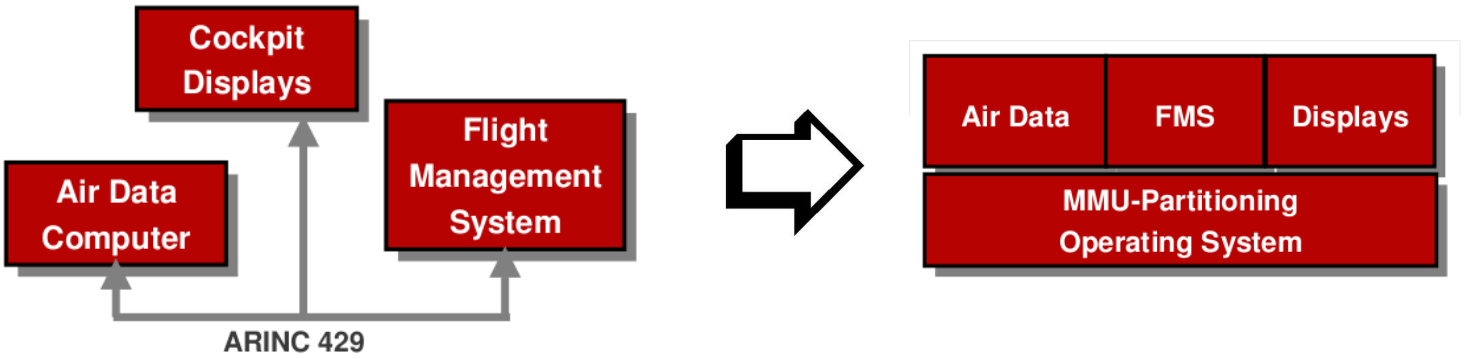


Système *partitionné*

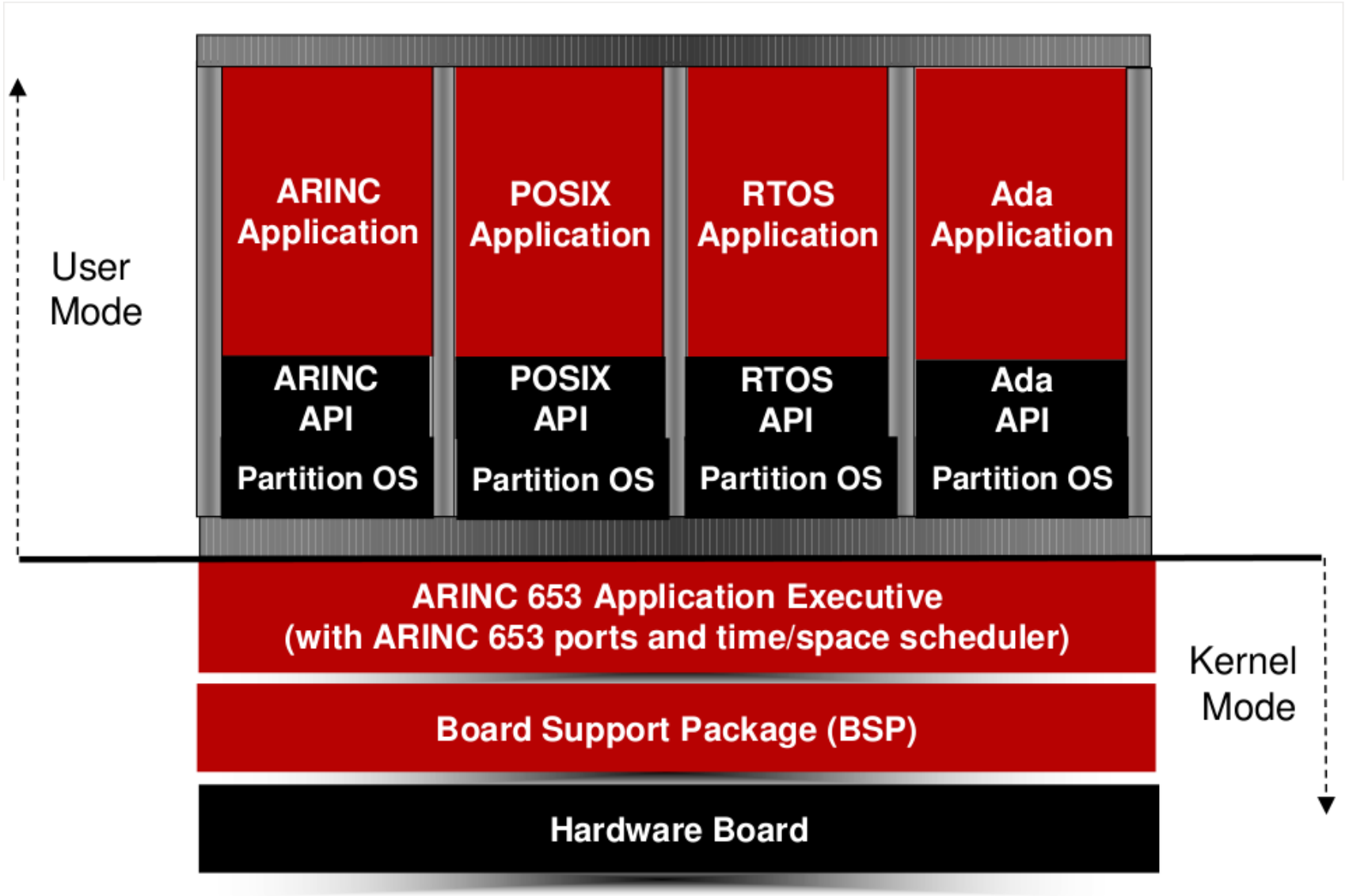
- Un calculateur peut héberger plusieurs applications à plusieurs degrés de criticité
- Nécessité de « séparer » les applications à plusieurs niveaux (partitionnement)
  - Spatial : espace mémoire
  - Temporel : ordonnancement
- La communication est souvent nécessaire mais maîtrisée
  - Intra-partition : entre tâches d'une même partition
  - Inter-partition : entre tâches de partitions différentes
- Normalisation ARINC 653

ARINC

- **A**eronautical **R**adio **INC**orporated, 1929
- Définitions de standards pour l'aéronautique dont :
  - ARINC 429 : bus de communication aéronautique (1974)
  - ARINC 653 : RTOS partitionné (1996)
  - ARINC 664 : Ethernet déterministe
  - ARINC 7xx : équipements (702A, 704, ...)
  - ...



Architecture ARINC 653



ARINC 653, description

- Évolution : 653 (1996), 653-1 (2003), 653-2 (2006)
- Noyau temps réel, abstraction du matériel par BSP
- Services :
  - Partition management
  - Process management
  - Time management
  - Inter/Intra partition communication
  - Error handling
- API de programmation : APEX (**AP**plication **EX**ecutive) pour accéder aux services (en C et Ada) => portabilité

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

open wide
INGENIERIE

Le logiciel libre dans l'embarqué

Importance du logiciel libre

- Le logiciel libre a pris un part importante des systèmes embarqués
  - Outils (GNU toolchain)
  - OS (Linux, RTEMS, ...)
  - Ateliers de développement (Eclipse)
- La plupart des éditeurs commerciaux fournissent également (ou uniquement) des composants basés sur du logiciel libre (VxWorks, Adacore, LynuxWorks, ...)
- Attention aux licences !

Quelques éléments sur le LL

- A peu près équivalent à la notion d'*open source*, voir http://www.opensource.org
- *Libre* ne veut pas (forcément) dire *gratuit*
- La confusion vient de la signification anglaise
  - free = libre / gratuit
- Différents types de logiciels
  - Le *freeware* ou *graticiel*: gratuit mais sources *non disponibles*, pas forcément de licence (abandon de la « paternité » du code)
  - Le *shareware*: sources non disponibles, coût modique, licence souvent propriétaire
  - Le *logiciel libre*: sources disponibles, licence *open source,* non liée à la notion de gratuité (on peut vendre un logiciel libre)

Avantages/inconvénients du logiciel libre

- Avantages
  - Disponibilité du code source: *maîtrise* et *maintenabilité* dans le temps
  - Redistribution *sans royalties*
  - Développements *dérivés* de code source existant
  - Outils de développement souvent « gratuits » !
- Inconvénients
  - Méfiance des décideurs/acheteurs, modèle *décentralisé* apparaissant comme « flou »
  - Contraintes de certaines licences (GPL, LGPL)
  - Support de certains matériels
  - Manque d'outils et de documentation

Les licences libres

- Licences compatibles avec le modèle libre défini sur http://www.opensource.org
- Problème de compatibilité avec les droits nationaux
- Exemples de licences :
  - GPL / LGPL
  - BSD
  - MIT/X11
  - Mozilla
  - FDL (pour la documentation)
  - CeCILL (France)
- Voir http://fr.wikipedia.org/wiki/Licence_libre

Quelques projets libres majeurs

- X Window System (X11), MIT (MIT/X11)
- Projet GNU (GPL/LGPL) → GCC, EMACS, ...
- FreeBSD (BSD)
- Apache (Apache)
- **Noyau Linux (GPL)**
- **RTEMS (GPL)**
- Eclipse (EPL)
- Mozilla / Thunderbird (MPL)
- WebKit (BSD/LGPL)
- Qt (LGPL)
- Android (partiellement Apache 2 + GPL)
- ...

Outils « libres » pour l'embarqué critique

- OCARINA (TPT) : « compilateur » AADL (Architecture Analysis and Design Language)
- SynDEx (INRIA) : Synchronized Distributed Executive, générateur de code « synchrone »
- POK (TPT) : RTOS ARINC 653
- TASTE (ESA) : The Assert Set of Tools for Engineering
- TOPCASED (Airbus) : Toolkit in OPen source for Critical Applications & SystEms Development
- B événementiel (CLEARSY)
- Initiative Open-DO (www.open-do.org) Adacore

open wide
INGENIERIE

Les OS libres pour l'embarqué

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
- Migration temps réel des anciens RTOS complexe car Linux *n'est pas* TR → évolution avec les extensions PREEMPT-RT et Xenomai
- Incompatible avec les systèmes *critiques*
- Souvent utilisé pour les outils, les simulateurs et architectures « mixtes » (banc de test)

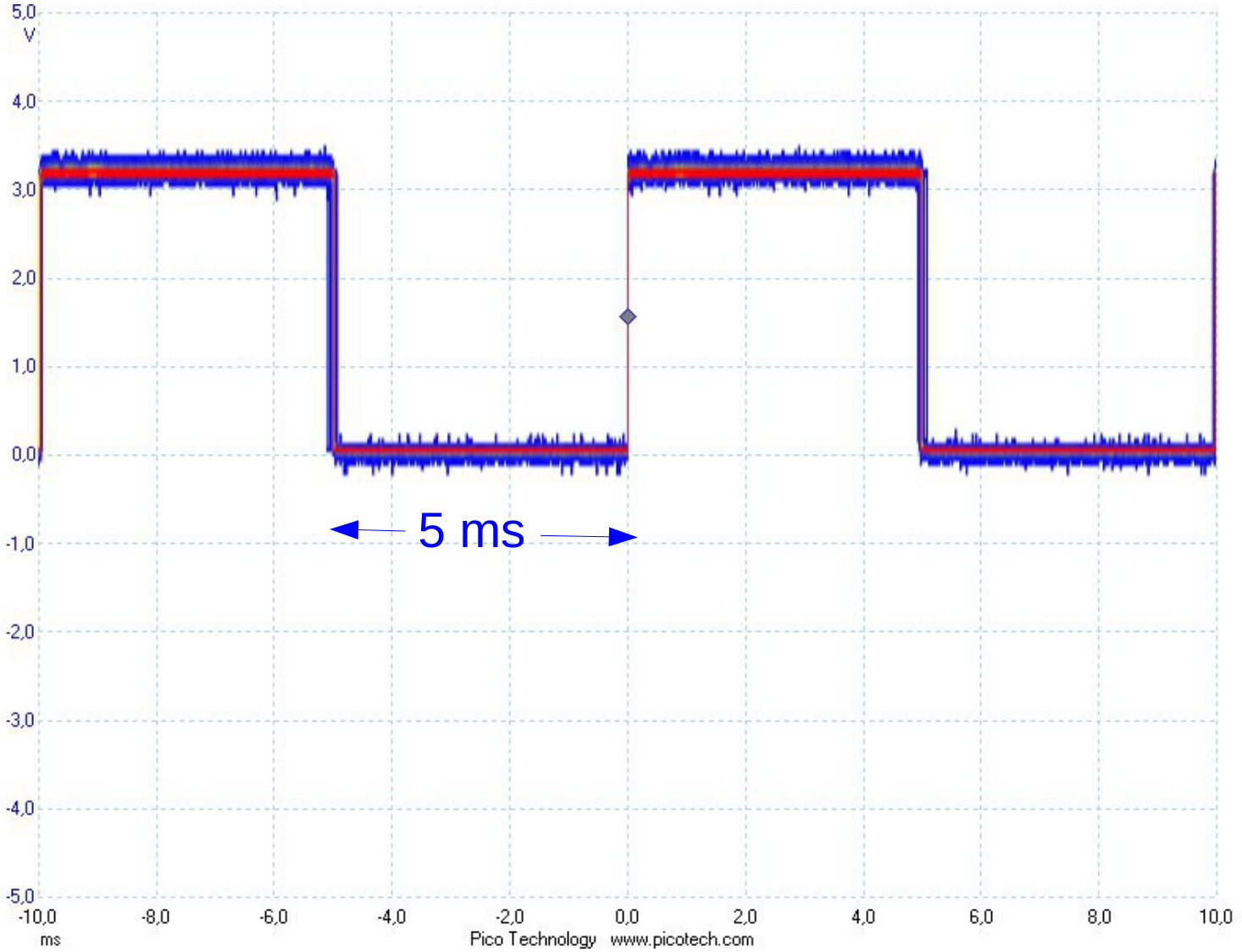
Linux comme (RT)OS , suite

- Désormais, « embarqué » n’implique pas forcément « temps réel » et/ou très faible empreinte mémoire (80% / 20 %)
- Dans les autres cas on peut utiliser d'autres systèmes plus légers et TR
  - eCos / Lepton (libre + POSIX)
  - RTEMS (libre + POSIX)

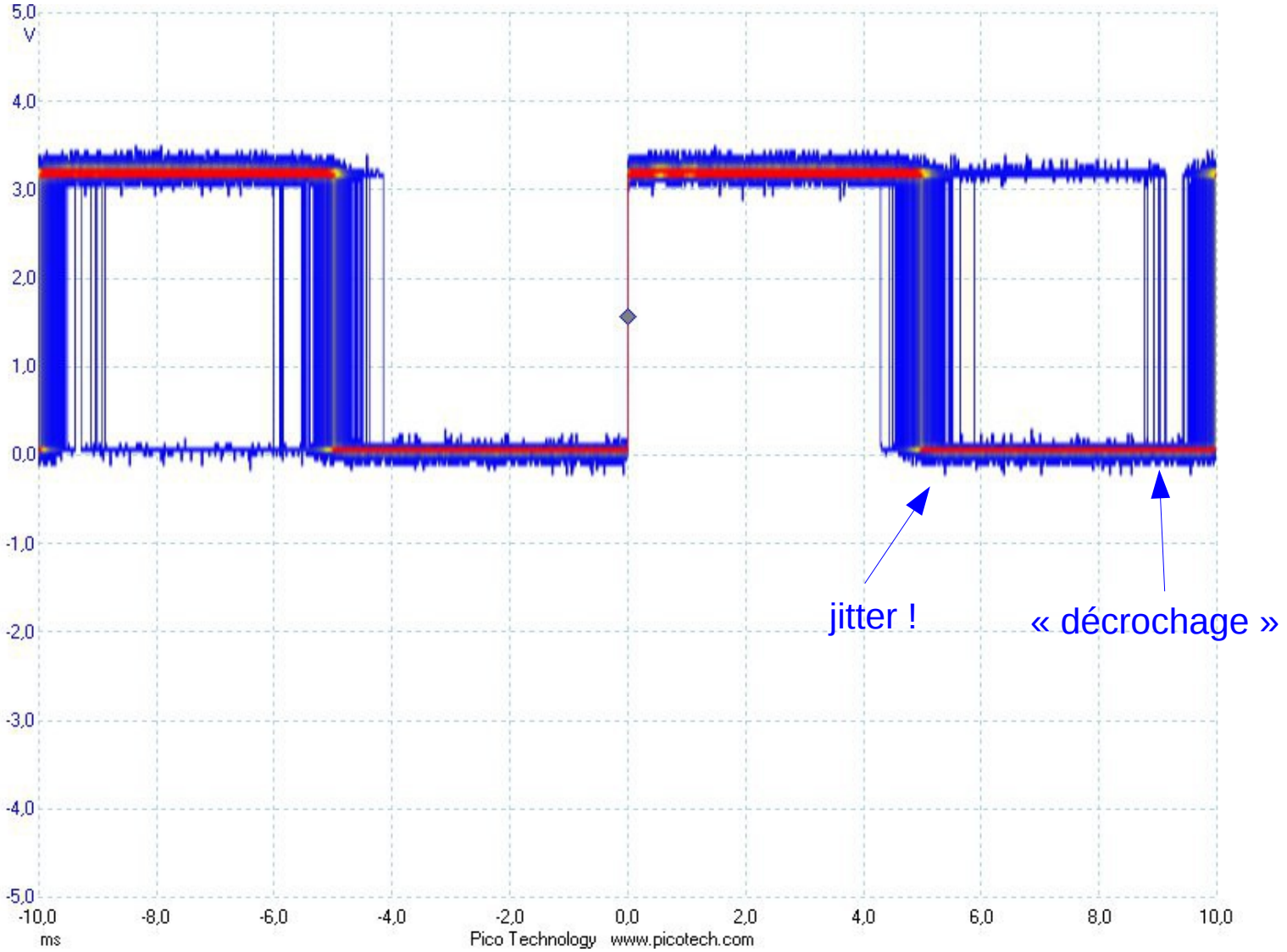
Linux et le TR

- 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)
- 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 mal prise en compte

Tâche périodique, Linux non chargé



Système chargé par « flood ping »



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-kernel, low-latency, PREEMPT-RT)
  - Ajouter un « co-noyau » TR qui partage le matériel avec le noyau Linux (RTLinux, RTAI, Xenomai) → approche « virtualisation »

Preeempt-kernel, low-latency

- Patch pour la version 2.4
- Intégrés à la version 2.6 et supérieures
- Technologies obsolètes
- Utiliser plutôt PREEMPT-RT !

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
- Aucun lien avec « preempt-kernel » !
- 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

PREEMPT-RT, modifications

- *Threaded interrupt model* → utilisation d'un *thread noyau* (donc 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)
- Le résultat est un noyau (presque) « préemptif », mais reste un noyau Linux
- Sections critiques avec des tâches non TR

PREEMPT-RT, conclusion

- Changements significatifs du code noyau
  - 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
- Permet de garantir 100 µs de jitter (x86)

Linux avec 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 (entre autres)
- Différents modèles de programmation
  - Noyau uniquement (RTLinux, version libre)
  - Noyau et espace utilisateur, semi-intégration Linux (RTAI, www.rtai.org)
  - Noyau & espace utilisateur, intégration Linux complète (Xenomai, www.xenomai.org)

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)

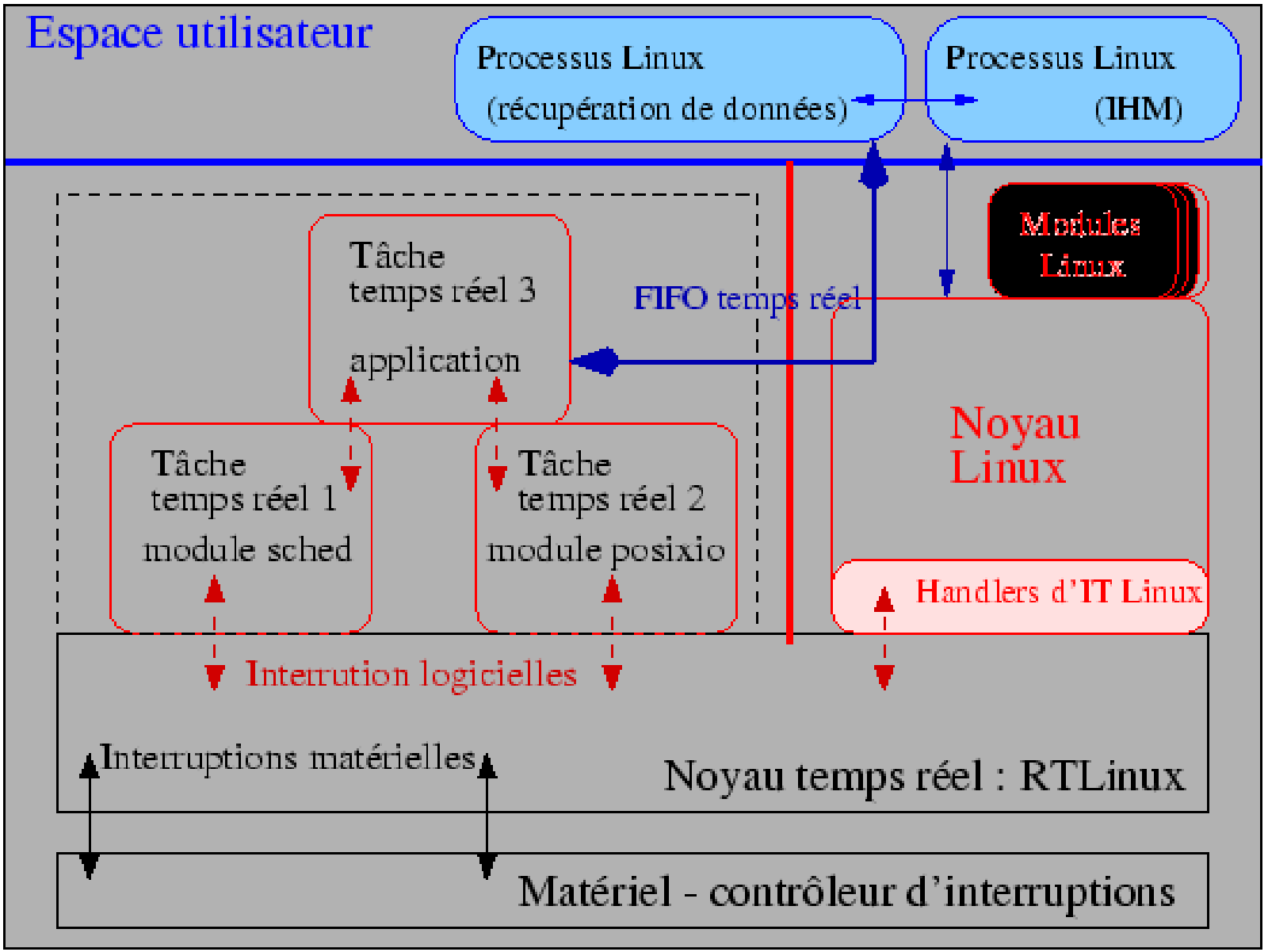
Linux + co-noyau, suite

- Peu de modifications sur le noyau Linux
  - *patch* de virtualisation (très bas niveau)
  - notion de domaine d'exécution (temps-réel / normal)
- Pas d'impact sur l'écriture de code noyau classique
- Impact sur l'écriture de code temps-réel !
  - utilisation des API fournies par le co-noyau
- 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

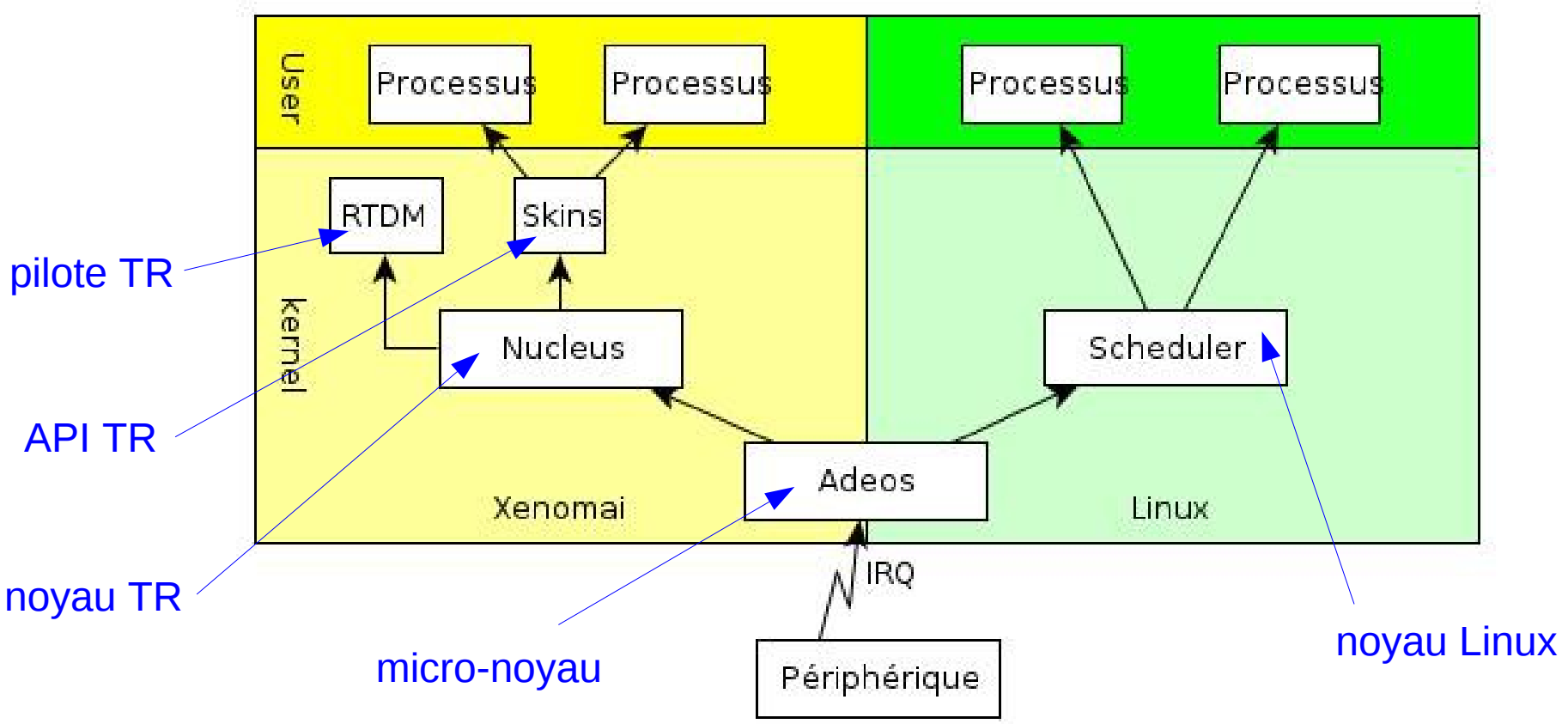
- **R**eal **T**ime **A**pplication **I**nterface
- Un « fork » de RTLinux développé au DIAPM de l'école polytechnique de Milan → Dipartimento di Ingegneria Aerospaziale (Paolo Montegazza)
- Utilisé au DIAPM pour des travaux d'enseignement et de recherche
- Quelques utilisations industrielles
- Position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

Xenomai

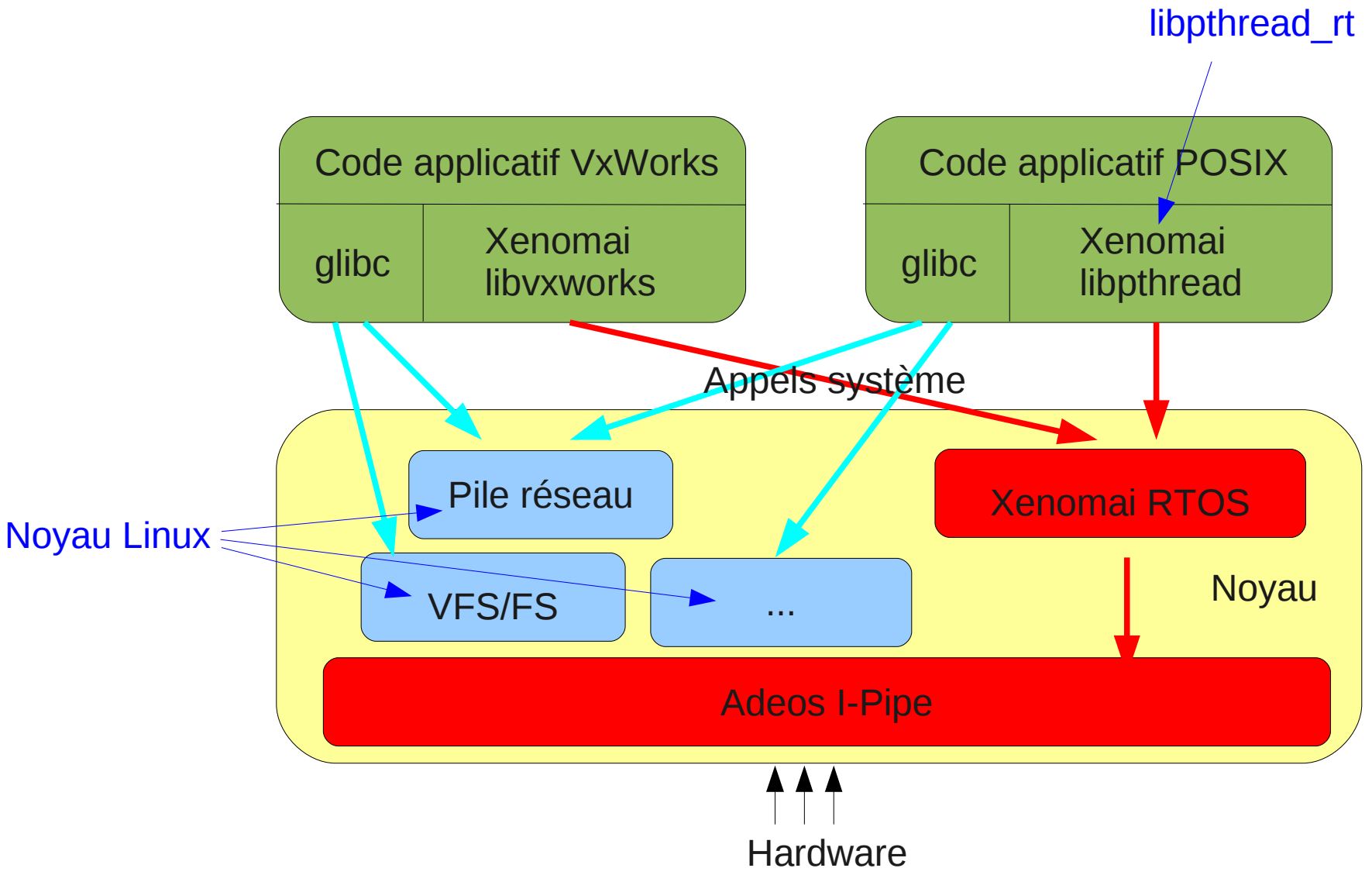
- Xenomai est un sous-système temps-réel de Linux
  - Programmation de tâches en espace utilisateur
  - API d'application et de pilotes temps réel (RTDM) dédiées
- Intégré au noyau Linux → « Real-time sub-system »
- Supporte de nombreuses architectures
- Dispose de « skins » permettant d'émuler des API 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

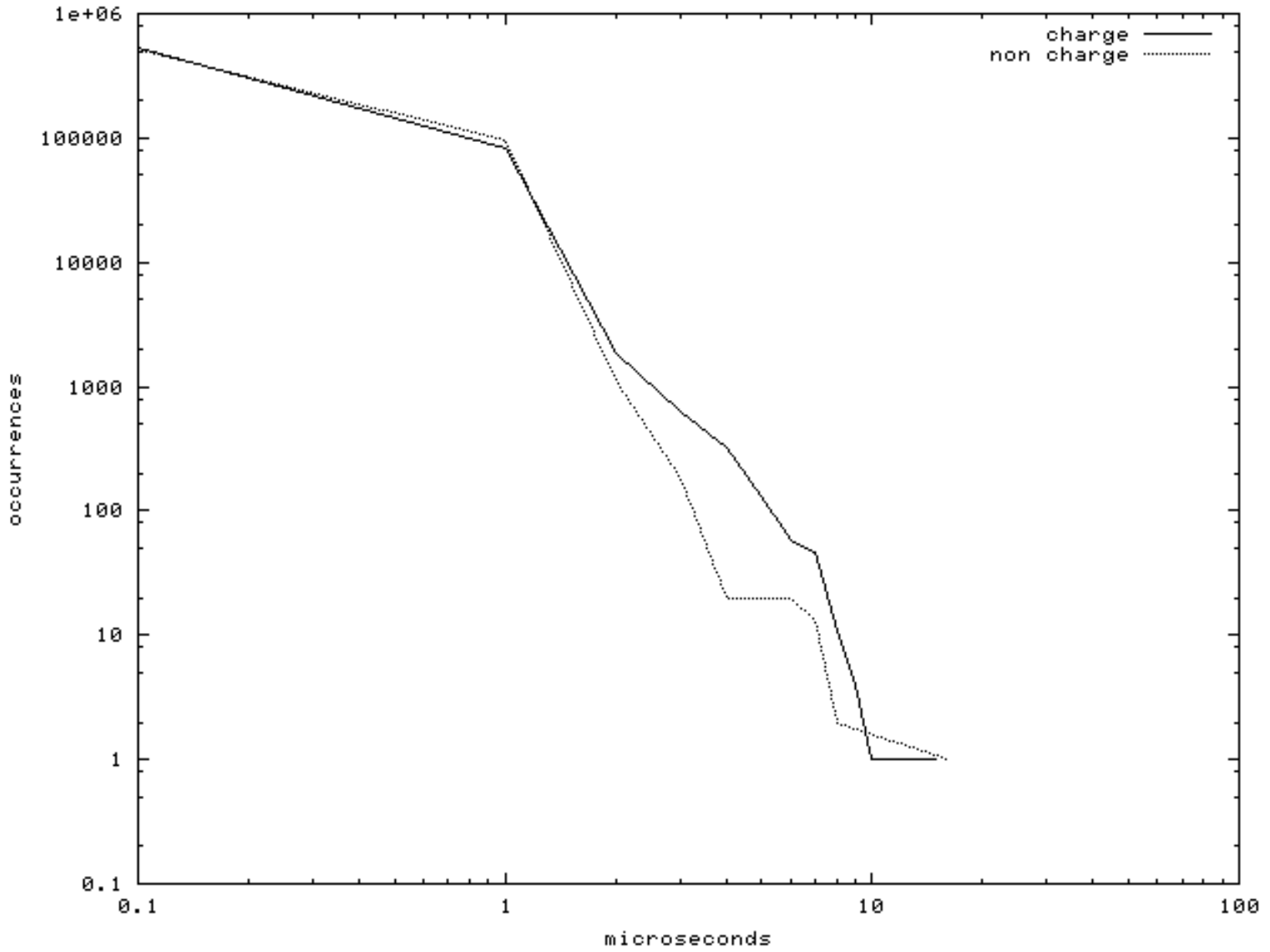
- Xenomai utilise un micro-noyau (ADEOS) pour partager le matériel avec le noyau Linux



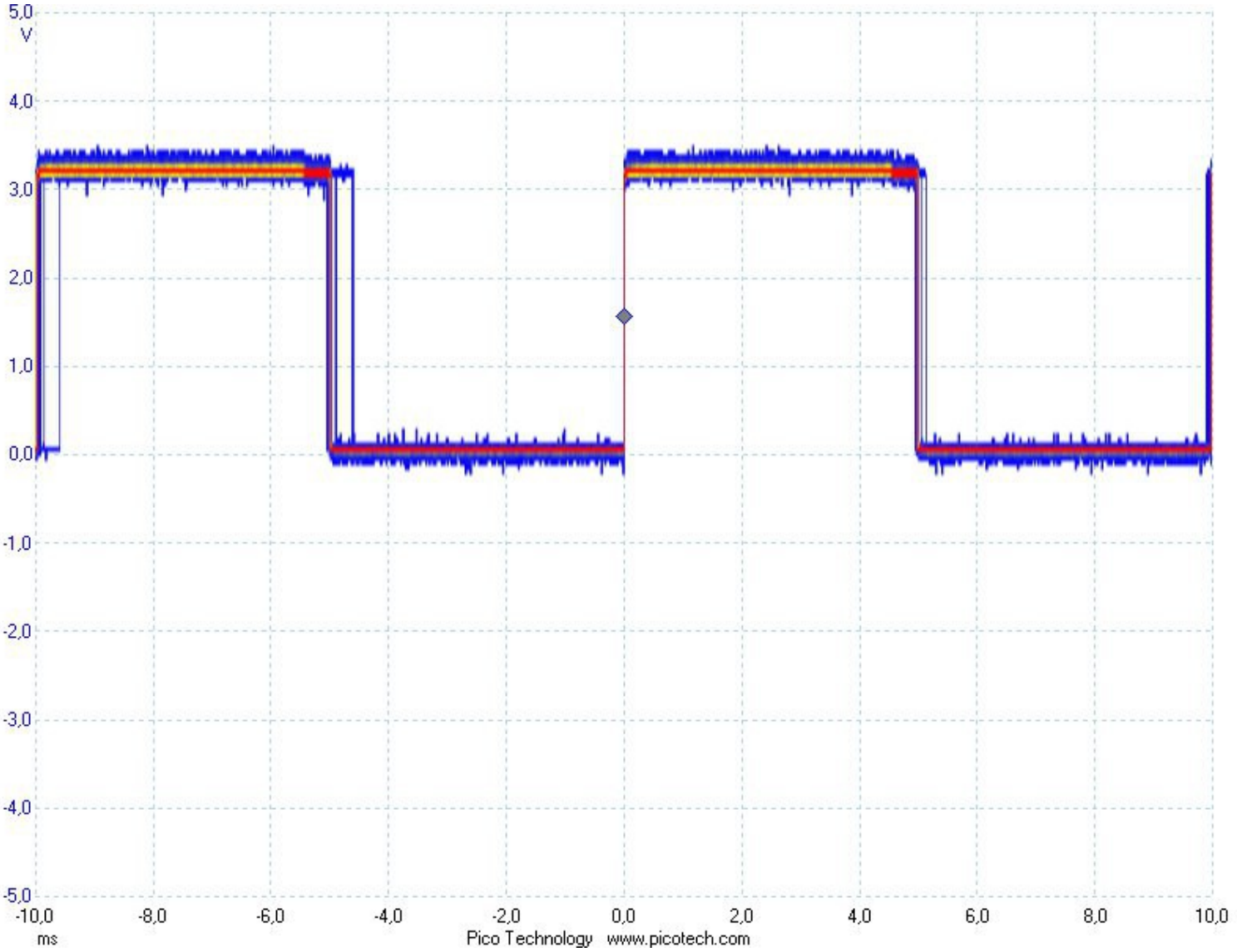
Répartition sur les deux domaines



Test de RTLinux, noyau 2.4.17 (2002)



Flood ping avec Xenomai



Android

- Difficile de ne pas l'évoquer !
- Système d'exploitation « libre » basé sur un noyau Linux modifié par Google
- Basé sur *Dalvik*, une machine virtuelle Java (très) *optimisée* pour le mobile
- Navigateur web basé sur Webkit (open source)
- Graphique optimisé en 2D ou 3D basé sur OpenGL ES
- Base de données SQLite
- Support des principaux formats audio, image, vidéo (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- Environnement de développement intégré (EDI) basé sur Eclipse (émulateur, débogueur, ...) → ADT

Android et l'industrie

- Android est conçu pour la téléphonie + tablettes !
- Les téléphones ressemblent de plus en plus à des ordinateurs personnels (voir marché du PC laminé par les tablettes)
- Les projets industriels abandonnent les RTOS propriétaires pour Linux (embarqué)
- Typologie des projets
  - Avec ou sans temps réel « dur »
  - Avec ou sans interfaces graphique (headless)
- Android est utilisable/conseillé si :
  - IHM (plus simple et standard car Java)
  - Pas de TR dur (pour l'instant, mais voir OS#8 !)

Android industriel, avantages/inconvénients

- Avantages
  - Programmation Java (simple et répandue)
  - IHM évoluée
  - Communauté importante
  - Fait rêver les managers et les comptables (tablette = grand public = bon marché)
- Inconvénients
  - Compatibilité POSIX partielle
  - Système de « build » statique assez rudimentaire par rapport à ceux de Linux (Buildroot, OE)
  - Pas réellement un projet libre ni communautaire
  - Noyau Linux non standard (même si la situation évolue)
  - Des interfaces matérielles mal supportées

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

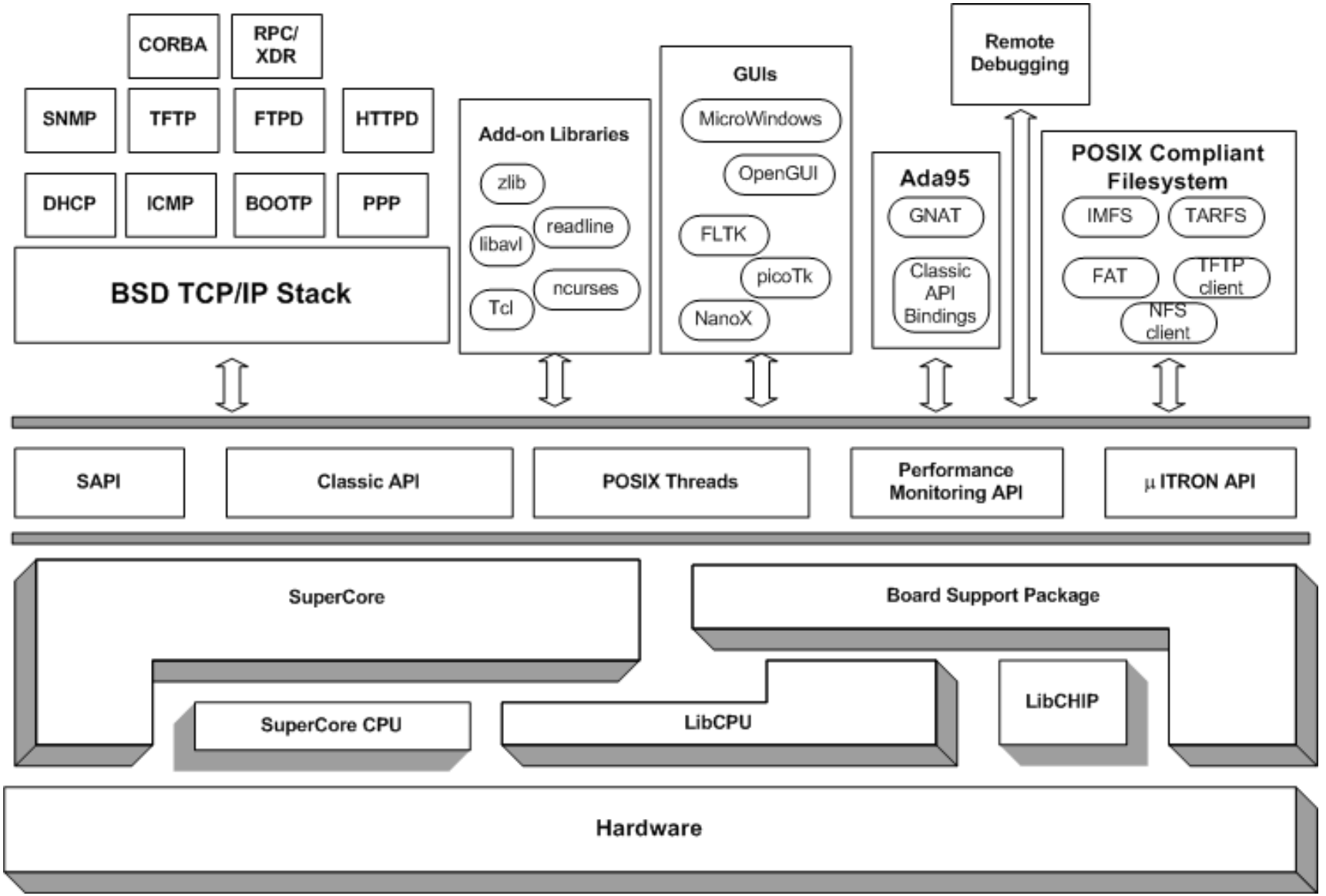
RTEMS

- 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, ...

RTEMS, suite

- RTEMS est un *exécutif* TR :
  - L'utilisation est plus complexe → configuration par fichiers, pas de « shell » comme sur un OS évolué
  - Un seul processus
  - Beaucoup plus petit qu'un OS → en sélectionnant les composants on peut arriver à une taille de quelques dizaines de Ko :-)
  - Pas (peu) de support MMU
  - De nombreuses fonctionnalités sont optionnelles : réseau, système de fichiers, etc.
  - Configuration statique de l'application
- Permet d'ajouter « facilement » API, ordonnanceur
- Léger, environ 320K lignes (`.c` et `.h`) pour la version 4.10.2

RTEMS, architecture



open wide
INGENIERIE

Les outils libres

Typologie des outils nécessaires

- Développement: compilation, mise au point (GNU Toolchain)
- EDI / IDE (Eclipse)
- Construction de distribution, paquets logiciels, production
- Émulation / simulation
- Sondes « matérielles » (JTAG)
- Gestion de version

  → La plupart de ces outils existent dans le monde du logiciel libre

Outils de développement

- Intégration des outils GNU (compilation, mise au point) dans un EDI (Environnement de Développement Intégré)
  - Eclipse + plugin CDT (C/C++)
  - QtCreator
- Produits commerciaux liés à un éditeur (Eclipse + outils)
  - Workbench (Wind River → CDT)
  - DevRocket (Montavista)
- Libre ou commercial ?
  - Outils additionnels intégrés (profiling, mise au point)
  - Support technique !

Production de distribution (Linux)

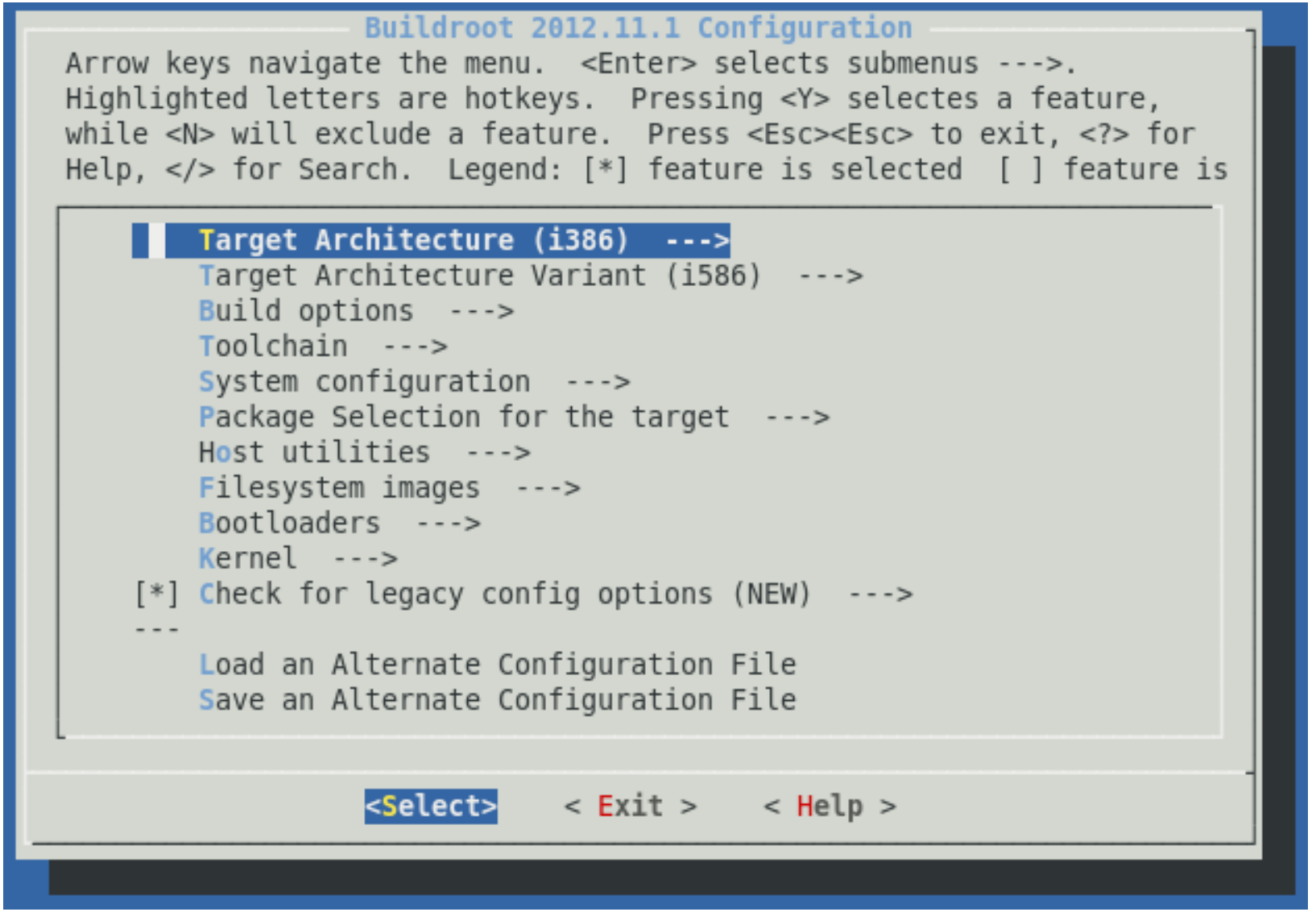
- Un outil crée la distribution à partir des sources des composants adaptés en appliquant des « patch »
  - Il ne s'agit pas de *distribution* mais d'outil de *création de distribution*
- L'outil ne fournit pas les sources mais les *règles de production* et prend en compte les dépendances
- L'outil peut produire la chaîne croisée GCC
- L'outil produit les différents éléments de la distribution
  - Image du bootloader
  - Noyau Linux
  - Images du root-filesystem
- Atelier JDEV2013 traitant ce sujet !

Les principaux outils disponibles

- 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)

Buildroot

- Lié au projet uClibc (micro-C-libc) : libC plus légère que la Glibc
- But initial: produire des images de test de uClibc
- Moteur basé sur des fichiers `Makefile` et des scripts-shell
- Outil de configuration « graphique »
- Peut désormais utiliser Glibc, Eglibc, ...
- Repris en 2009 par Peter Korsgaard et Thomas Petazzoni
- Une version officielle tous les 3 mois: 2009.02, ..., 2011.02...2013.08


Buildroot 2012.11.1 Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is
Target Architecture (i386) --->
Target Architecture Variant (i586) --->
Build options --->
Toolchain --->
System configuration --->
Package Selection for the target --->
Host utilities --->
Filesystem images --->
Bootloaders --->
Kernel --->
[*] Check for legacy config options (NEW) --->

Load an Alternate Configuration File
Save an Alternate Configuration File
<Select> < Exit > < Help >

OpenEmbedded

- Une « généralisation » de l'approche utilisée dans BR
- Utilise un moteur écrit en Python (bitbake) et un ensemble de règles utilisant un principe d'héritage → « recipe » (recette)
- Pas d'interface de configuration
- Processus lourd → plusieurs heures pour la première compilation (environ 30 mn pour BR)
- TRES puissant, recommandé dans le cas ou l'on gère un grand nombre de configurations ou des distributions complexes
- Gère la notion de paquet binaire, contrairement à BR
- Utilisé par Yocto, qui devient une référence

Émulateur QEMU

- Émulateur de matériel initialement développé par Fabrice Bellard, diffusé sous GPL v2
- Exécuté dans l'espace utilisateur de Linux
- Permet d'émuler diverses architectures: x86, PowerPC, ARM, etc.
- Émulation de carte complète → outil de développement, mise au point, test automatique
  - Outil de certification DO-178 (Couverture) http://www.open-do.org/projects/couverture
- Désormais, large communauté avec dépôt Git sur http://git.savannah.gnu.org/cgit/qemu.git

Conclusions/marchés

- L'embarqué est un marché en croissance (oui ça existe!)
- Augmentation (significative) de l'activité « embarqué » entre 2007 et 2012 chez les SII et éditeurs
- Les marchés traditionnels restent leaders
  - spatial/défense (62%)
  - Automobile (47%)
- Augmentation de l'utilisation de l'open source (outils)
  - Éditeurs (65 %)
  - SSII (42%)
- http://www.syntec-numerique.fr/sites/default/files/related_docs/CP-etude-PAC-Embarque-27Nov12.pdf

open wide INGENIERIE

Démonstrations

- Comparaison Linux standard, PREEMPT-RT, Xenomai
- Linux sur Raspberry Pi (Raspbian, Buildroot, Yocto)
- RTEMS sur Raspberry Pi (une led clignotante !)
- PREEMPT-RT et Xenomai sur Android-x86

open wide
INGENIERIE

Questions ?