http://www.isen.fr/

Introduction au développement de systèmes embarqués avec Linux

Julien Gaulmin

<julien23@gmail.com> / @julien23

Plan

1. L'embarqué :
   - Définitions,
   - Marché et perspectives,
   - Topologie d'un système embarqué,
   - Architecture matérielle,
   - Architecture logicielle.
2. Pourquoi GNU/Linux ?
   - Les raisons technologiques,
   - Les raisons économiques,
   - Les raisons personnelles,
   - Les autres *OS*,
   - Les licences,
   - Limites.
3. Solutions :
   - Types de solutions,
   - Plate-formes orientées produits,
   - Briques de base logicielles,
   - Références.

4. Notions essentielles :
   - Concepts et orthodoxie Unix,
   - Analyse du processus de démarrage de Linux,
   - Processus de compilation,
   - Édition de liens binaires,
   - Exécutables,
   - μClinux vs Linux.
5. Méthodes et outils de développement :
   - Terminologie,
   - Méthodologies de développement,
   - Compilation croisée (*cross-compilation*),
   - Débogage et optimisation,
   - Émulation et virtualisation logicielles.
6. Étude de cas ;
7. Références.

L’embarqué

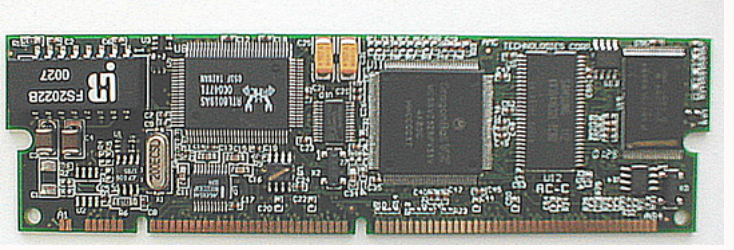
Définitions

- Combinaison de matériels et logiciels permettant de remplir une ou plusieurs fonctions spécifiques avec des **contraintes** plus ou moins sévères (consommation, température, taille, performances...) ;
- Système électronique et informatique **autonome** ne possédant **pas d'entrées/sorties standards** (écran, clavier, souris...) ;
- Ordinateur non visible et **intégré** dans un système ou équipement d'utilité différente ;
- Se dit plus généralement de tout système qui n'appartient pas aux différents domaines traditionnels de l'informatique (bureautique, web, gestion, gros systèmes) ;
- On parle aussi de systèmes **"enfouis"** quand le rapport à l'informatique est peu discernable.

Exemples

- PDA, *smartphone*, tablette, *diskless-PC* ;
- Balladeur numérique, platine DVD, décodeur/enregistreur TV ;
- Automate industriel, robot, machine outil ;
- Routeur, *box* Internet, μCsimm, *Raspberry Pi* ;
- Autoradio, machine à laver, ABS voiture.






Embarqué et Temps-réel

- Système temps-réel :
  - l'information après acquisition et traitement est encore **pertinente**,
  - capacité de répondre à une sollicitation donnée pour produire une réaction appropriée en un temps déterminé $\Rightarrow$ **déterminisme**,
  - pas forcément synonyme de puissance de calcul ni de vitesse d'exécution.
- Certains systèmes embarqués sont soumis à des contraintes temporelles plus ou moins fortes nécessitant l'emploi de noyaux temps-réel (RTOS [1]) ;

1. RTOS : ***R**eal **T**ime **O**perating **S**ystem*

- Deux formes principales d'applicatifs temps-réel :
  - **temps-réel "dur"** (*hard real-time*) $\Rightarrow$ le système doit **absolument** répondre à un événement donné dans un **temps déterminé** (ABS, système militaire...),
  - **temps-réel "mou"** (*soft real-time*) $\Rightarrow$ le système est soumis à des **containtes temporelles** mais le retard ou l'annulation d'une échéance n'est **pas catastrophique** pour autant (jeux vidéo, autoradio, VoIP [1]...).

1. VoIP : ***Voice over IP***

Marché et perspectives

L'essor de l'embarqué

- Couplé à l'essor du "tout numérique" et du multimédia ;
- Convergence des médias (voix, vidéo, données...) ;
- Intelligence à tous les niveaux (domotique, robotique...) ;
- Produits communiquants, mobilité ;
- Miniaturisation et baisse des coûts des composants ;
- Les prochains axes de croissance : IoT [1], wearables, etc ;
- Évolution du marché total de l'embarqué de 32 M$ en 1998 à 92 M$ en 2008 et 2000 M$ en 2015 [2].

1. IoT : ***I**nternet **o**f **T**hings*
2. Estimation de *IDC* en 2011

Les us et coutumes

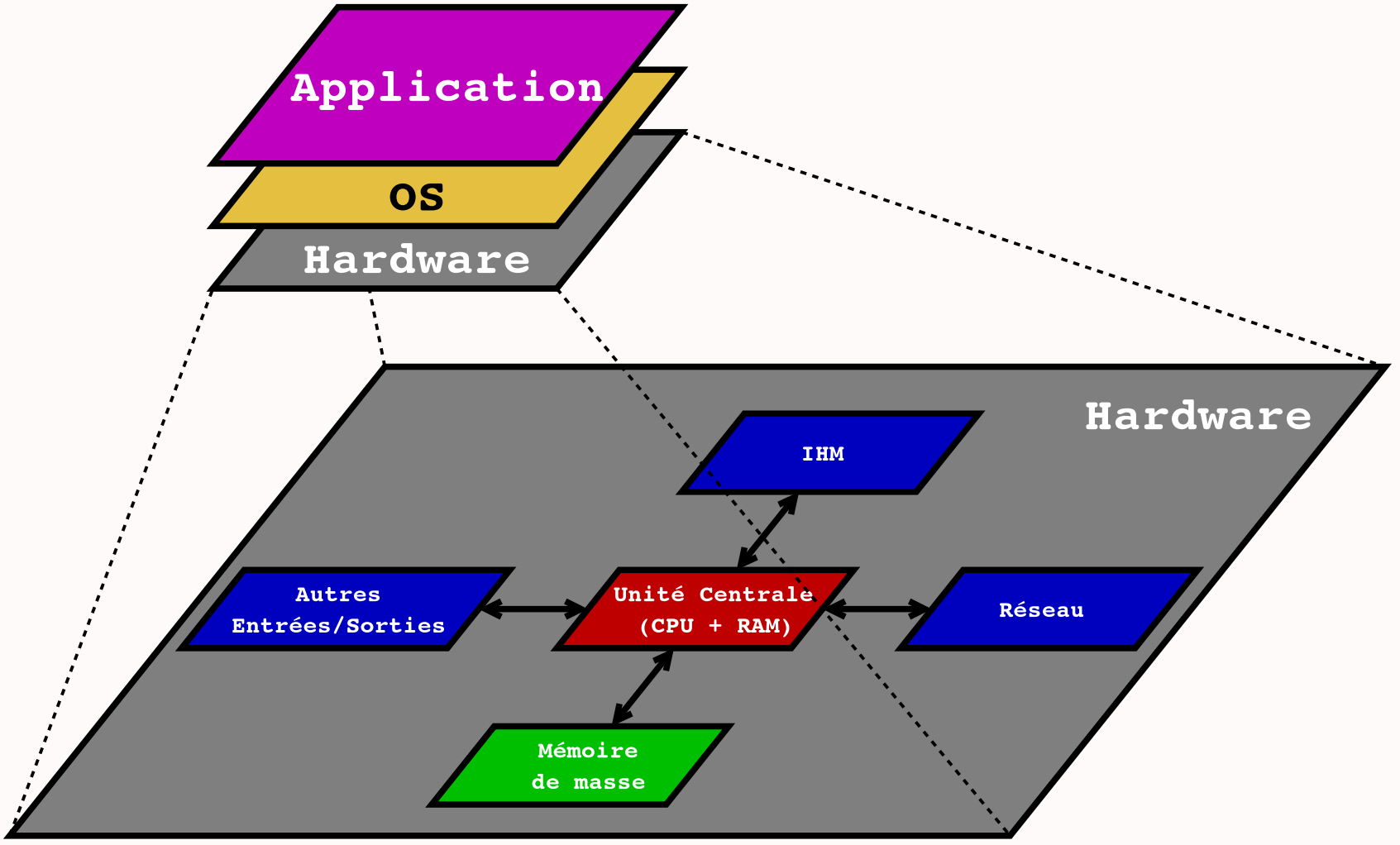
✗ Marché très fermé des *OS* propriétaires :
— pas compatibles entre eux,
— kits de développement coûteux et figés,
— fortes royalties,
— dépendance vis à vis d'un éditeur.
✗ *OS* développés en interne :
— développement et mise au point long et coûteux,
— évolutivité et pérennité délicates,
— portabilité réduite.
✓ Depuis 2000, Linux et les logiciels libres se sont imposés comme alternative à tous ces monopoles, du prototypage au produit fini.

Les acteurs

- Communautés de développeurs ;
- Éditeurs de logiciels ;
- Sociétés de service ;
- Fabriquants de composants ;
- Industriels ;
- Scientifiques, universitaires et étudiants ;
- Organisations (CELF [1], Linux Foundation, TV Linux Alliance, RTLF [2], LDPS [3], FHSG [4], LSB [5], FSF [6], OpenGroup...)
- Médias (portails web, éditeurs, presse...).

1. CELF : ***CE** **L**inux **F**orum*
2. RTLF : ***R**eal-**T**ime **L**inux **F**oundation*
3. LDPS : ***L**inux **D**evelopment **P**latform **S**pecification*
4. FHSG : ***F**ilesystem **H**ierarchy **S**tandard **G**roup*
5. LSB : ***L**inux **S**tandard **B**ase*
6. FSF : ***F**ree **S**oftware **F**oundation*

Topologie d'un système embarqué



Architecture matérielle

- Souvent dédiée dans les systèmes à fortes contraintes de consommation, d'encombrement ou de coût ;
- Aujourd'hui, la tendance s'inverse avec l'apparition de *system off-the-shelves* de plus en plus intégrés (SOB [1], SOC [2]...) ;
- Adaptée aux ressources nécessaires ⇒ pas de superflu (économies d'échelle).

1. SOB : ***S**ystem **O**n **B**oard*
2. SOC : ***S**ystem **O**n **C**hip*

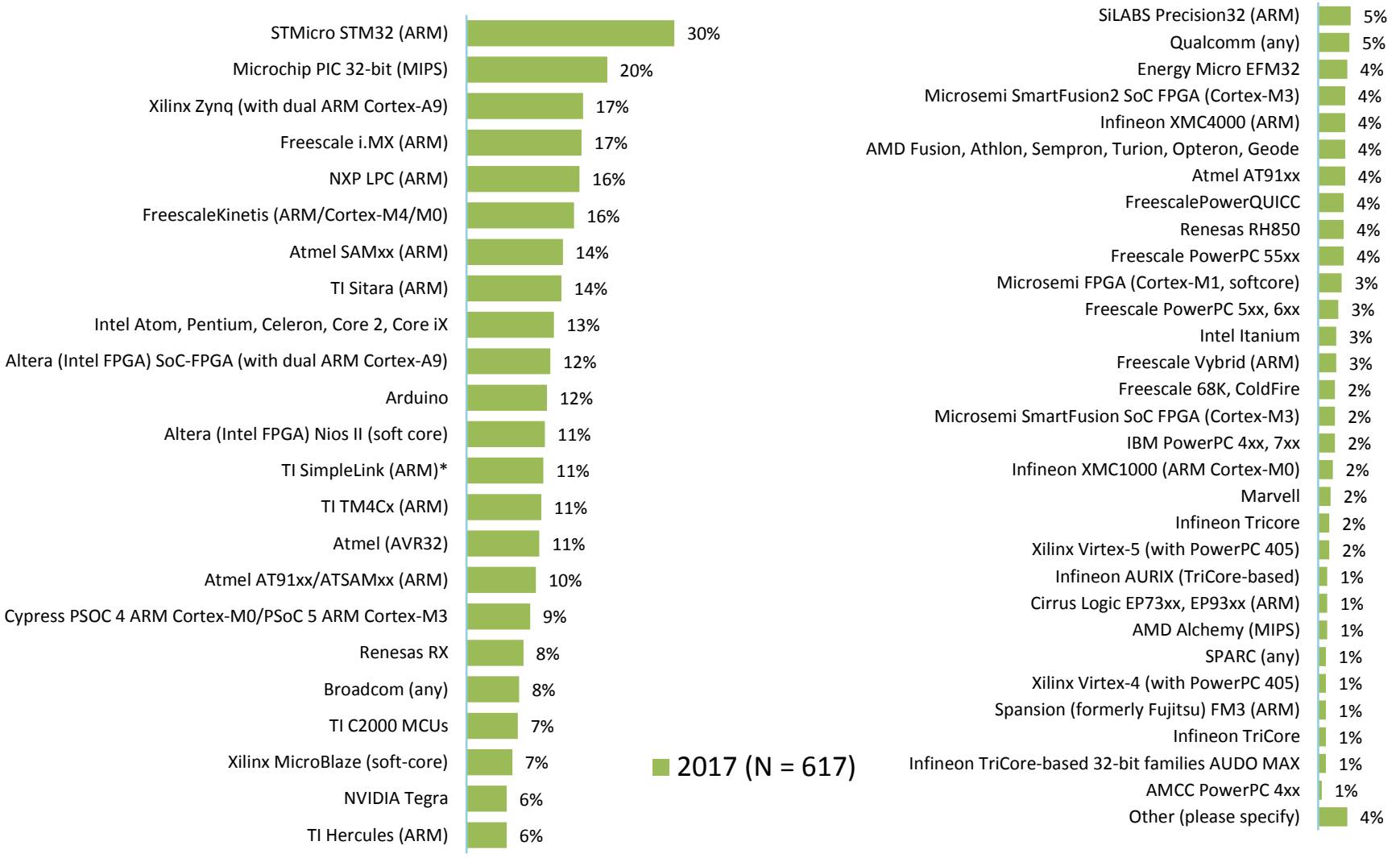
Familles de processeurs

- **Généralistes** : x86, ia64, x64, PowerPC, Sparc...
- **Basse consommation** : ARM [1] (ARMx, Cortex, XScale), SuperH, MIPS [2], PowerPC...
- **SOC** : 68k (Motorola DragonBall et ColdFire), x86 (AMD Geode, VIA Nano, Intel Atom), ARM (NVidia Tegra, Qualcomm Snapdragon, Samsung Hummingbird, Apple Ax, Intel PXA), MIPS, PowerPC, Etrax...
- **ASIC ou FPGA** avec coeur(s) ARM, MIPS, PowerPC...

1. ARM : ***A**dvanced **R**ISC **M**achine*
2. MIPS : ***M**icroprocessor without **I**nterlocked **P**ipeline **S**tages*

Which of the following 32-bit chip families would you consider for your next embedded project?

ASPENCORE

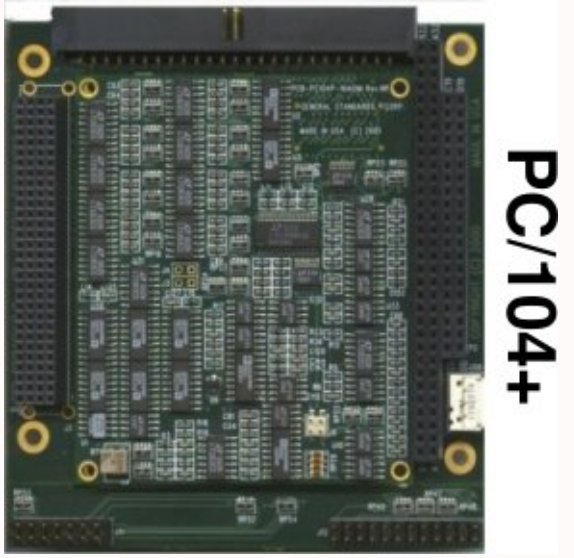


Bus de communication

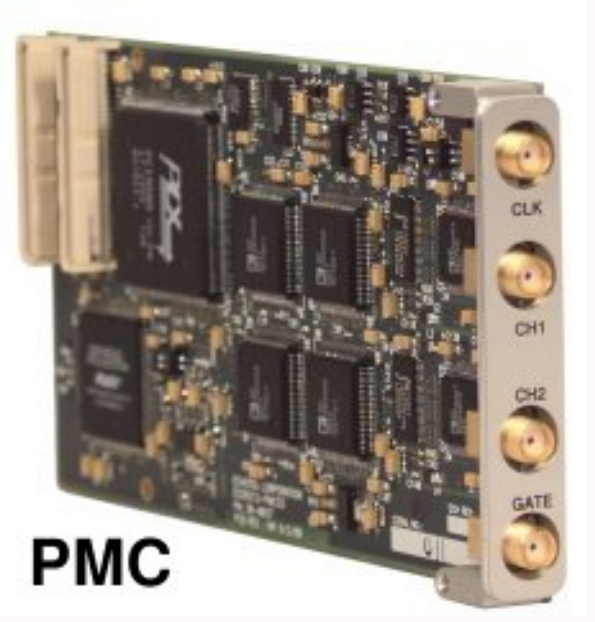
- VME [1] $\Rightarrow$ VMEbus, VME64, VME64x, VME320, VXI [2], IP-Module, M-Module...
- PCI [3] $\Rightarrow$ CompactPCI (cPCI), PCI-X, PXI [4], PMC [5], PC/104+, PCI-104, MiniPCI...
- PCIe [6] $\Rightarrow$ XMC, AdvancedTCA, AMC, ExpressCard, MiniCard, PCI/104-Express, PCIe/104...
- PCMCIA [7] $\Rightarrow$ PCMCIA, PC Card, CardBus...
- ISA [8] $\Rightarrow$ PC/104...

1. VME : ***V**ersa **M**odule **E**urocard*
2. VXI : ***VME**bus e**X**tension for **I**nstrumentation*
3. PCI : ***P**eripheral **C**omponent **I**nterconnect*
4. PXI : ***PCI** e**X**tension for **I**nstrumentation*
5. PMC : ***PCI M**ezzanine **C**ard*
6. PCIe : ***PCI eXPRESS***
7. PCMCIA : ***P**ersonnal **C**omputer **M**emory **C**ard **I**nternational **A**ssociation*
8. ISA : ***I**ndustry **S**tandard **A**rchitecture*


VME


PC/104+


PXI
PDXI


CLK
CH1
CH2
GATE
PMC

- Parallèle $\Rightarrow$ ATA/ATAPI [1] (IDE [2]), SCSI [3], Centronics/IEEE1284...
- Série $\Rightarrow$ I^2C [4], RS232, RS485, USB [5], IEEE1394, Serial ATA...
- Réseau $\Rightarrow$ Ethernet, FDDI [6], X.25, WiFi/802.11, BlueTooth/ZigBee/WUSB/Wibree/802.15.x/ANT, IrDA [7], ATM [8], Token Ring, GSM [9]/GPRS [10]/UMTS [11]/LTE [12]...

1. ATAPI : ***AT** **A**ttachment **P**acket **I**nterface*
2. IDE : ***I**ntergated **D**rive **E**lectronics*
3. SCSI : ***S**mall **C**omputer **S**ystems **I**nterface*
4. I^2C : ***I**nter-**I**ntegrated **C**ircuit*
5. USB : ***U**niversal **S**erial **B**us*
6. FDDI : ***F**ibber **D**istributed **D**ata **I**nterface*
7. IrDA : ***I**nfra**r**ed **D**ata **A**ssociation*
8. ATM : ***A**synchronous **T**ransfert **M**ode*
9. GSM : ***G**lobal **S**ystem for **M**obile communications*
10. GPRS : ***G**eneral **P**acket **R**adio **S**ervice*
11. UMTS : ***U**niversal **M**obile **T**elecommunications **S**ystem*
12. LTE : ***L**ong **T**erm **E**volution*

Mémoires de masse

- Supports Magnétiques ⇒ 2,5", 3,5", microdrive, bande...
- Mémoires Flash ⇒ SSD [1], FlashDisk, DiskOnChip, CompactFlash, CFI [2], SD Card, Memory Stick, USB Mass Storage...
- ROM [3], EPROM, EEPROM, UVPROM...
- Supports optiques ⇒ CD, DVD, Blu-ray...
- Combinaison des précédentes.

1. SSD : ***S**olid **S**tate **D**evice*
2. CFI : ***C**ommon **F**lash **I**nterface*
3. ROM : ***R**ead **O**nly **M**emory*

Entrées/Sorties

- Entrées :
  - entrées TOR [1] (collecteurs ouverts, optocoupleurs...) ou GPIO [2],
  - capteurs/convertisseurs (pression, audio, température...),
  - clavier, boutons poussoirs, dalles tactiles, télécommandes (infrarouge, radio...),
  - capteurs optiques (photo/vidéo), lecteurs radio (tags), lecteurs laser (codes barres).
- Sorties :
  - sorties TOR (relais, optocoupleurs, logique...) ou GPIO,
  - LEDs, écrans et afficheurs LCD,
  - bips, synthèse vocale, alarmes,
  - imprimantes en tous genres (papier, étiquettes, photos...).

1. TOR : **T**out **O**u **R**ien
2. GPIO : ***G**eneral **P**urpose **I**nput **O**utput*

Réseau

- Technologies :
  - classiques ⇒ Ethernet, ATM, X.25...
  - de terrain (*fieldbus*) ⇒ CAN [1], RS232, RS485, CPL [2], ARCnet [3],
  - sans fil ⇒ WiFi/802.11, IrDA, BlueTooth/ZigBee/WUSB/Wibree/802.15.x, GSM/GPRS/UMTS...
- Pour quoi faire ?
  - communiquer,
  - partager des informations,
  - contrôler et superviser.

1. CAN : ***C**ontroller **A**rea **N**etwork*
2. CPL : **C**ourant **P**orteur en **L**igne
3. ARCnet : ***A**ttached **R**essource **C**omputer **net**work*

Architecture logicielle

Rappels sur les systèmes d'exploitation (*OS*)

- Composé d'un noyau et de *drivers* qui réalisent l'**abstraction du matériel** ;
- De **bibliothèques** qui formalisent l'API [1] pour accéder aux services du noyau ;
- D'un ensemble variable d'**outils de base** (configuration du matériel, gestion des fichiers, interface graphique, etc).

1. API : ***A**pplication **P**rogramming **I**nterface*

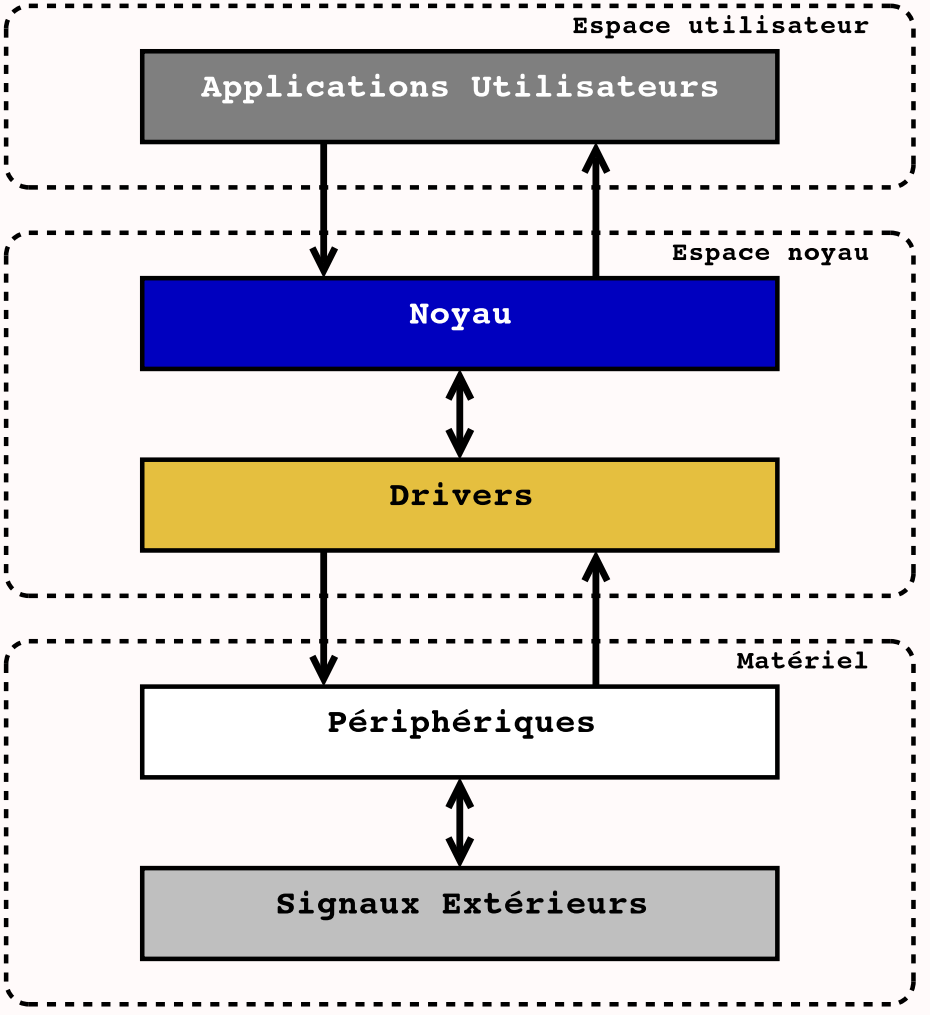
Le noyau (*kernel*)

- Premier programme exécuté lors de la mise en marche ;
- Utilise généralement un mode d'exécution privilégié du CPU ;
- Réalise une **abstraction du matériel et des services** :
  - offre une suite de services généraux qui facilitent la création de logiciels applicatifs,
  - sert d'intermédiaire entre ces logiciels et le matériel,
  - apporte commodité, efficacité et capacité d'évolution,
  - permettant d'introduire de nouvelles fonctions et du nouveau matériel sans remettre en cause les logiciels.

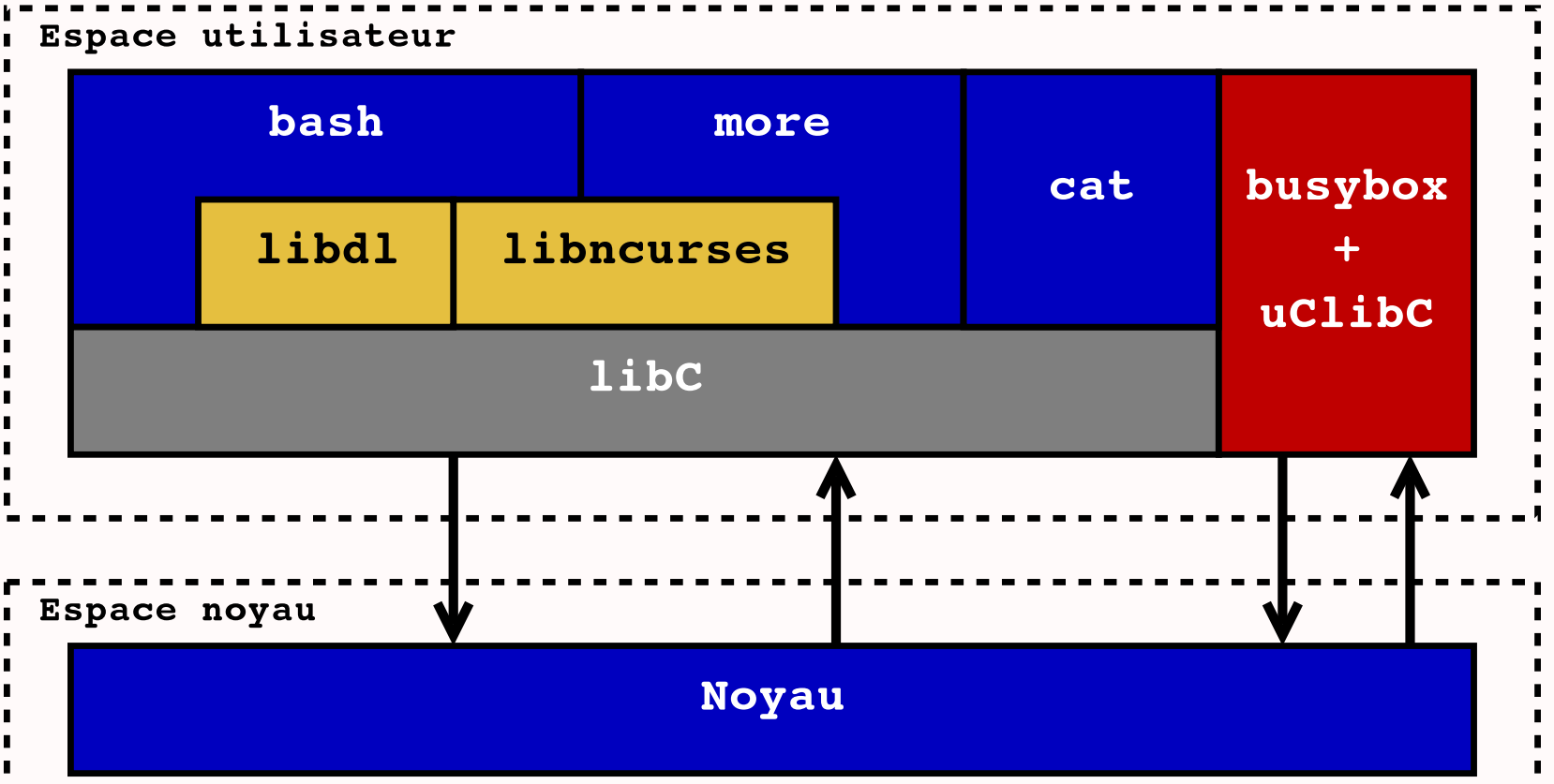
- Les fonctionnalités offertes diffèrent d'un modèle à l'autre, et sont typiquement en rapport avec :
  - l'exécution et l'**ordonnancement des programmes**,
  - l'utilisation de la **mémoire centrale et des périphériques**,
  - la manipulation et l'organisation des fichiers (**systèmes de fichiers**),
  - la communication et le réseau, et
  - la détection d'erreurs.

Source: Wikipedia (http://fr.wikipedia.org/wiki/Systeme_d'exploitation)

Structure d'un *OS* monolithique



Structure de l'espace utilisateur



Pourquoi GNU/Linux ?

Les raisons technologiques

- ✓ Disponibilité des **sources** ⇒ maîtrise du système à tous les étages ;
- ✓ Standards ouverts (formats, protocoles...) ⇒ **interoperabilité** ;
- ✓ **Performances**, **fiabilité** ;
- ✓ **Portabilité** ⇒ diversité des architectures et matériels supportés ;
- ✓ **Connectivité réseau** native ;
- ✓ Mise à l'échelle (*scalability*) ⇒ empreinte mémoire réduite ;
- ✓ Diversité et multiplicité des logiciels disponibles.

Les raisons économiques

- ✓ **Gratuit**, pas de royalties ;
- ✓ Facilité et rapidité de mise en oeuvre ;
- ✓ **Indépendance** vis à vis des fournisseurs/prestataires ;
- ✓ Communauté de développeurs $\Rightarrow$ support gratuit et sans limites, possibilité de s'adresser directement aux concepteurs ;
- ✓ **Multiplicité des acteurs** $\Rightarrow$ l'inertie ou les axes d'évolution d'un produit ne sont pas dictés par une seule entité.

Les raisons personnelles

✓ Logiciels libres et leurs **4 libertés** essentielles :

— **utiliser** ⇒ pas de restrictions,

— **étudier** ⇒ *"use the source Luke"*,

— **redistribuer** ⇒ vendre ou donner le logiciel et son code source,

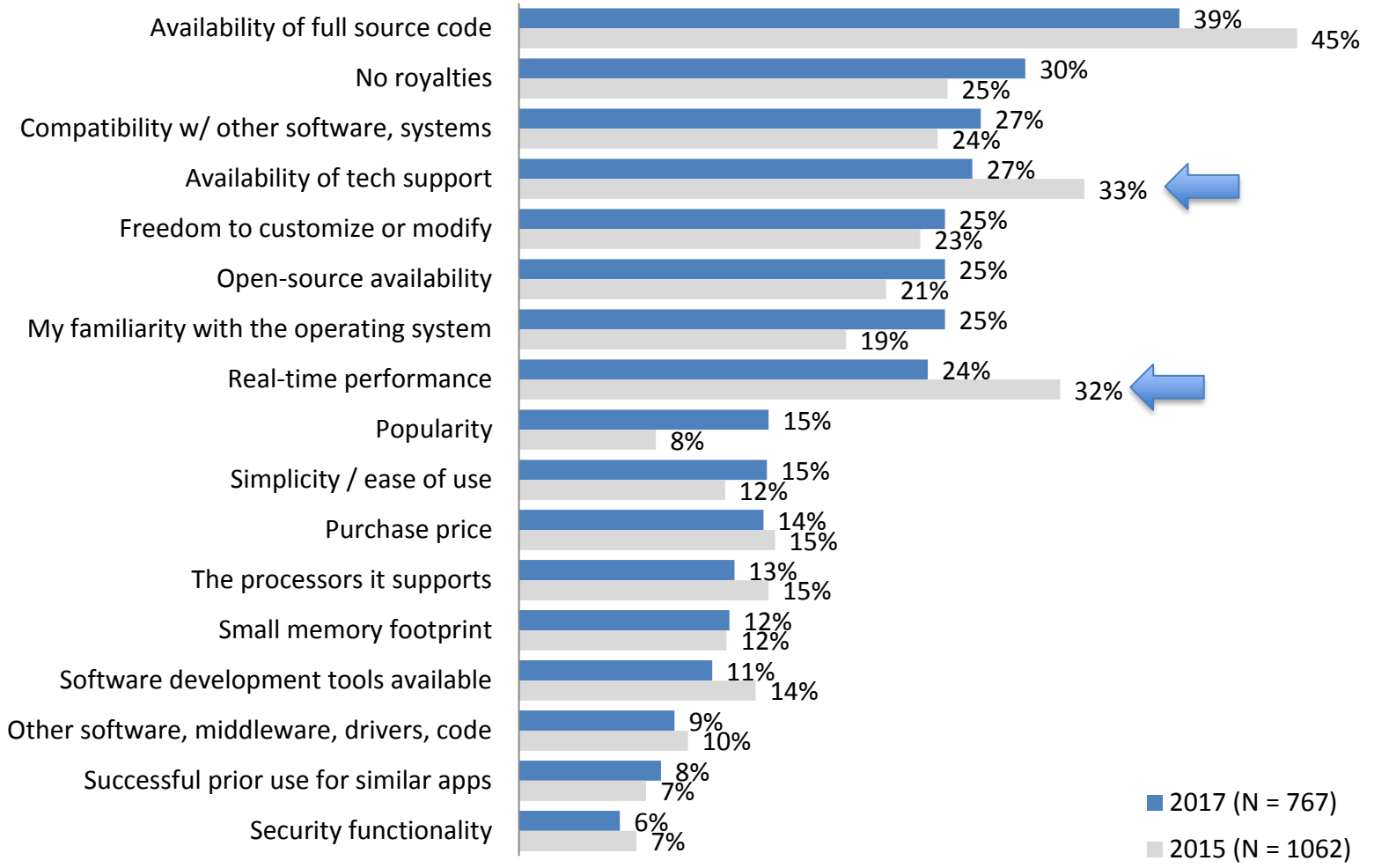
— **améliorer** ⇒ corriger ou ajouter des fonctionnalités.

✓ Participer à un des plus grand **projet communautaire** ;

✓ Se jauger par rapport aux meilleurs,

✓ Étoffer son CV et s'adapter aux demandes du marché.

What are the most important factors in choosing an operating system?

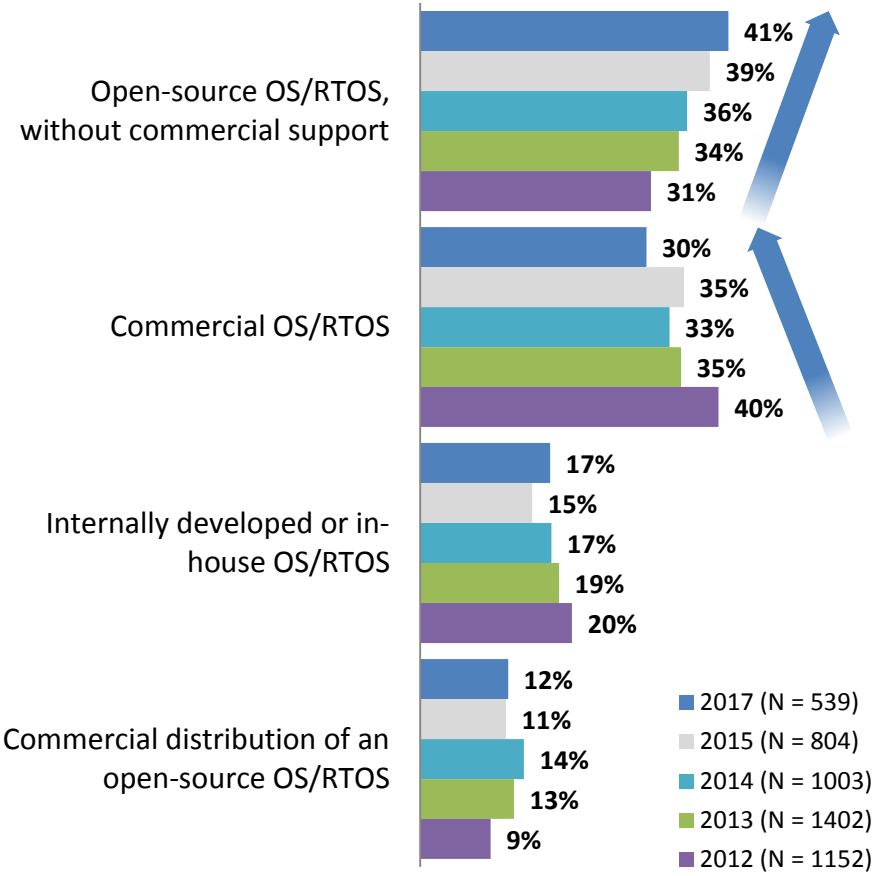


Base: Currently using an operating system

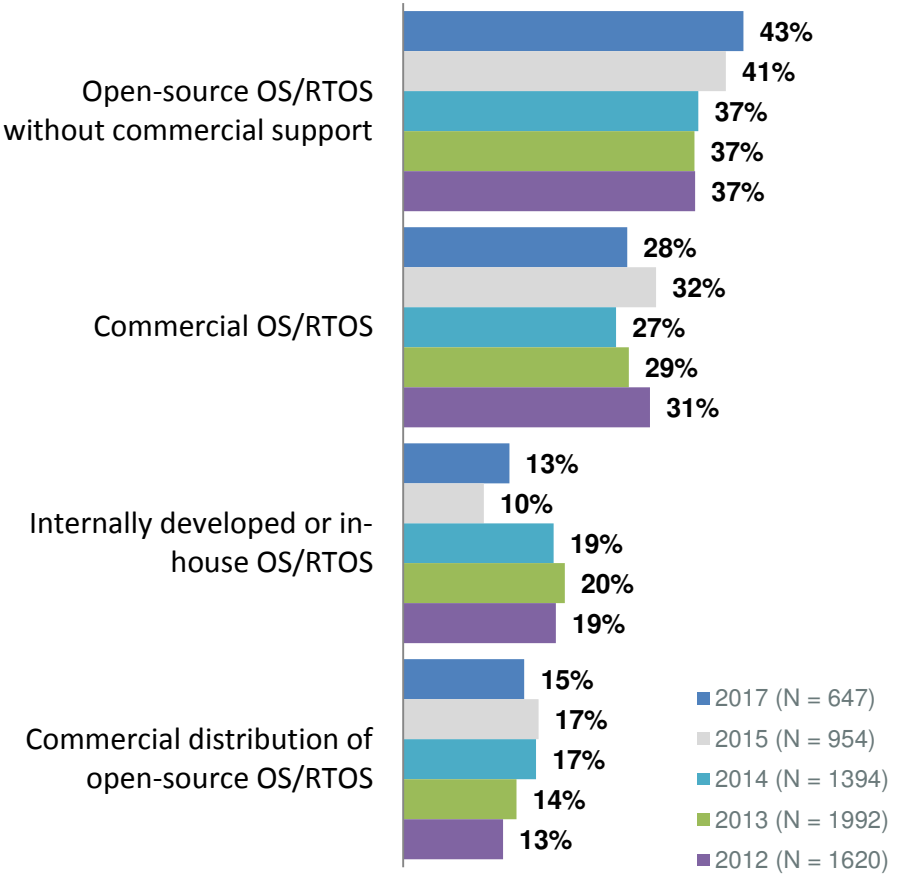
Les autres *OS*

ASPENCORE

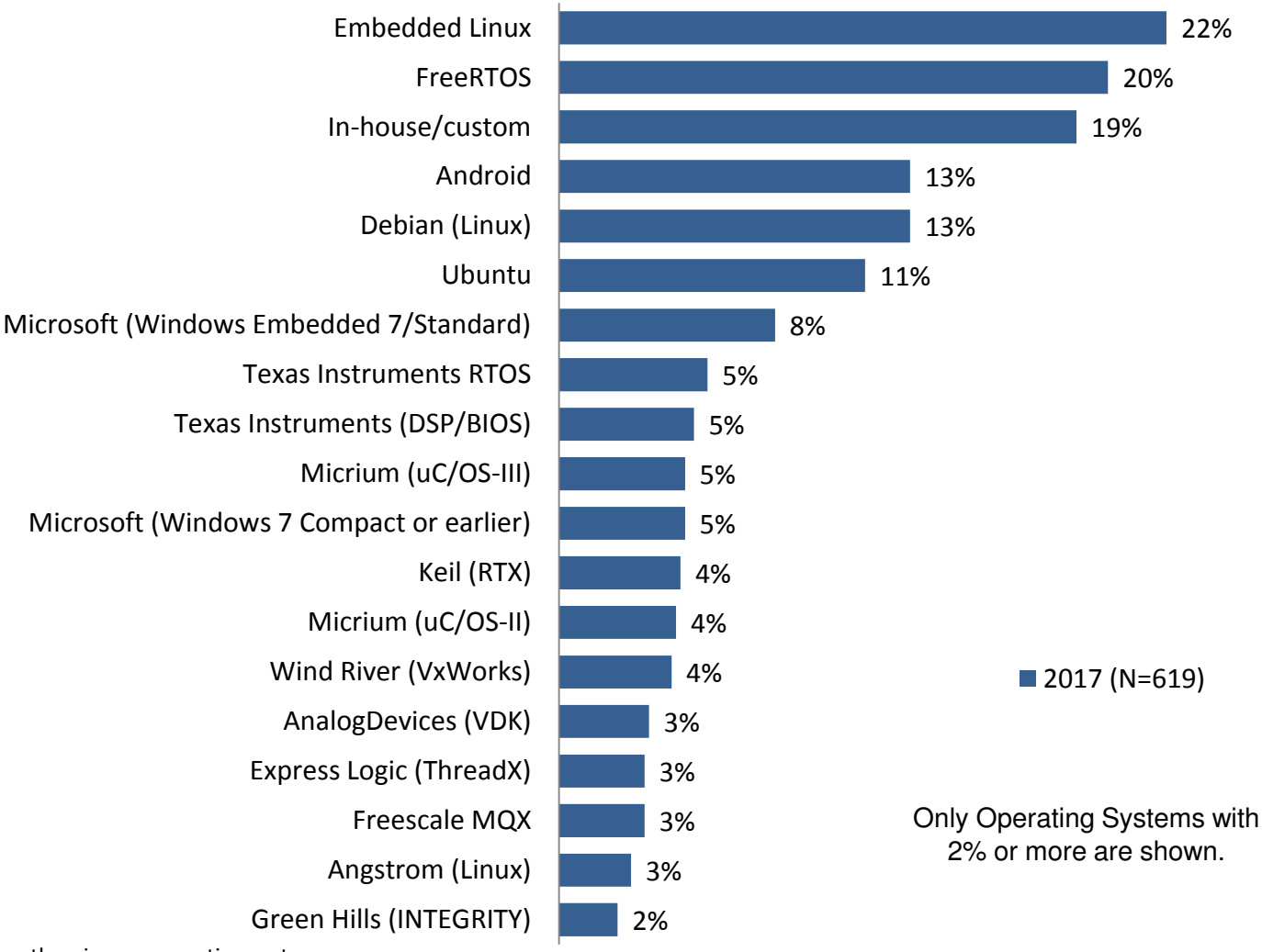
My current embedded project uses:



My next embedded project will likely use:



Please select ALL of the operating systems you are currently using.



Base: Currently using an operating system

Les licences

- GNU GPL [1] $\Rightarrow$ la plus répandue, la plus stricte, la plus éprouvée et la plus contagieuse, elle est basée sur la notion de *copyleft* [2] ;
- GNU LGPL [3] $\Rightarrow$ autorise l'édition de liens dynamique avec du code non libre, très utilisée pour les bibliothèques ;
- X11/MIT/BSD $\Rightarrow$ très permissives, autorisent l'exploitation propriétaire du code ;
- *PL $\Rightarrow$ de nombreux éditeurs ont créé leurs propres licences pour distribuer leurs logiciels *open source* (Netscape, IBM, Sun...), la FSF dresse un bilan de compatibilité de ces licences avec celles de GNU (http://www.gnu.org/licenses/license-list.html) et l'OSI [4] certifie leur conformité avec l'OSD [5] (http://www.opensource.org/).

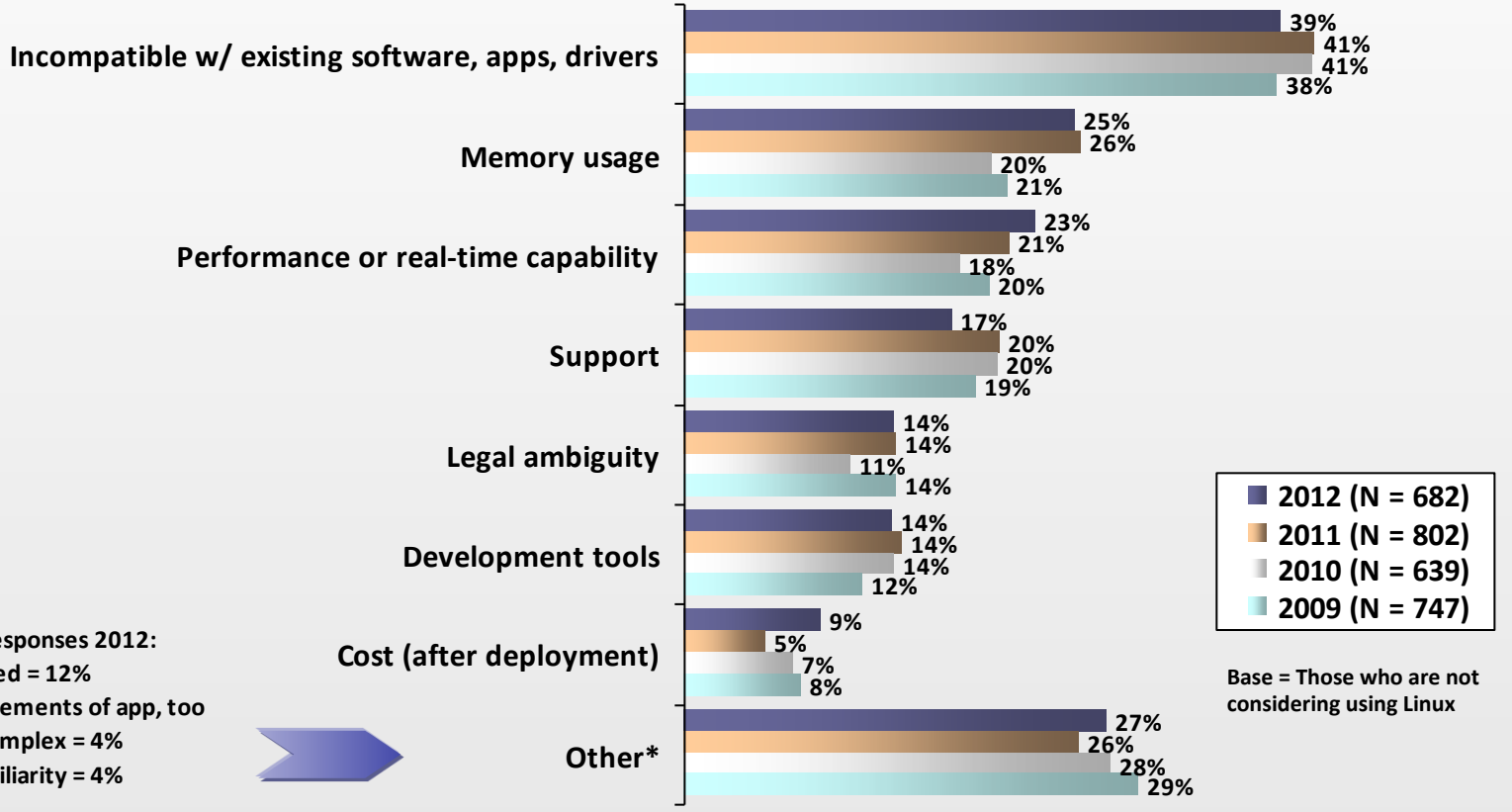
1. GPL : ***G**eneral **P**ublic **L**icense*
2. *copyleft* : utilisation des droits d'auteur / *copyright* pour assurer la liberté (utiliser, étudier, redistribuer et améliorer) du logiciel
3. LGPL : ***L**esser **G**eneral **P**ublic **L**icense*
4. OSI : ***O**pen **S**ource **I**nitiative*
5. OSD : ***O**pen **S**ource **D**efinition*

Limites

- ✗ Pas adapté pour les systèmes de quelques dizaines de kilo-octets (électroménager "classique", HiFi, télécommande...) ;
- ✗ Code fourni *AS-IS* ;
- ✗ Pas de relation contractuelle avec un fournisseur/prestataire ;
- ✗ Pas d'assistanat automatique (RTFM [1]) ;
- ✗ Réticences des développeurs face aux changements (principe d'inertie face à la nouveauté et à la liberté de choix) ;
- ✗ Diversité des acteurs.

1. RTFM : ***R**ead **T**he **F*****ing **M**anual*

Why are you not interested in embedded Linux?



Base = Those who are not considering using Linux

*Other responses 2012:
- No need = 12%
- Requirements of app, too big/complex = 4%
- Unfamiliarity = 4%

UBM Electronics

Solutions

Types de solutions

Trois approches s'offrent aux développeurs de systèmes embarqués sous Linux :

- **Système "fait maison"** :
  - construction/portage d'une chaîne de compilation croisée pour l'hôte (*host cross-platform development environment*),
  - choix et intégration, dans la chaîne de compilation, des briques de base logicielles composant le système (+ applications dédiées),
  - construction/alimentation d'un système de fichiers principal pour la cible (*build and feed target rootfs*),
  - automatisation des procédures précédentes (scripts, makefiles...) permettant la reconstruction rapide d'une image binaire contenant le noyau et le *rootfs* après modifications.

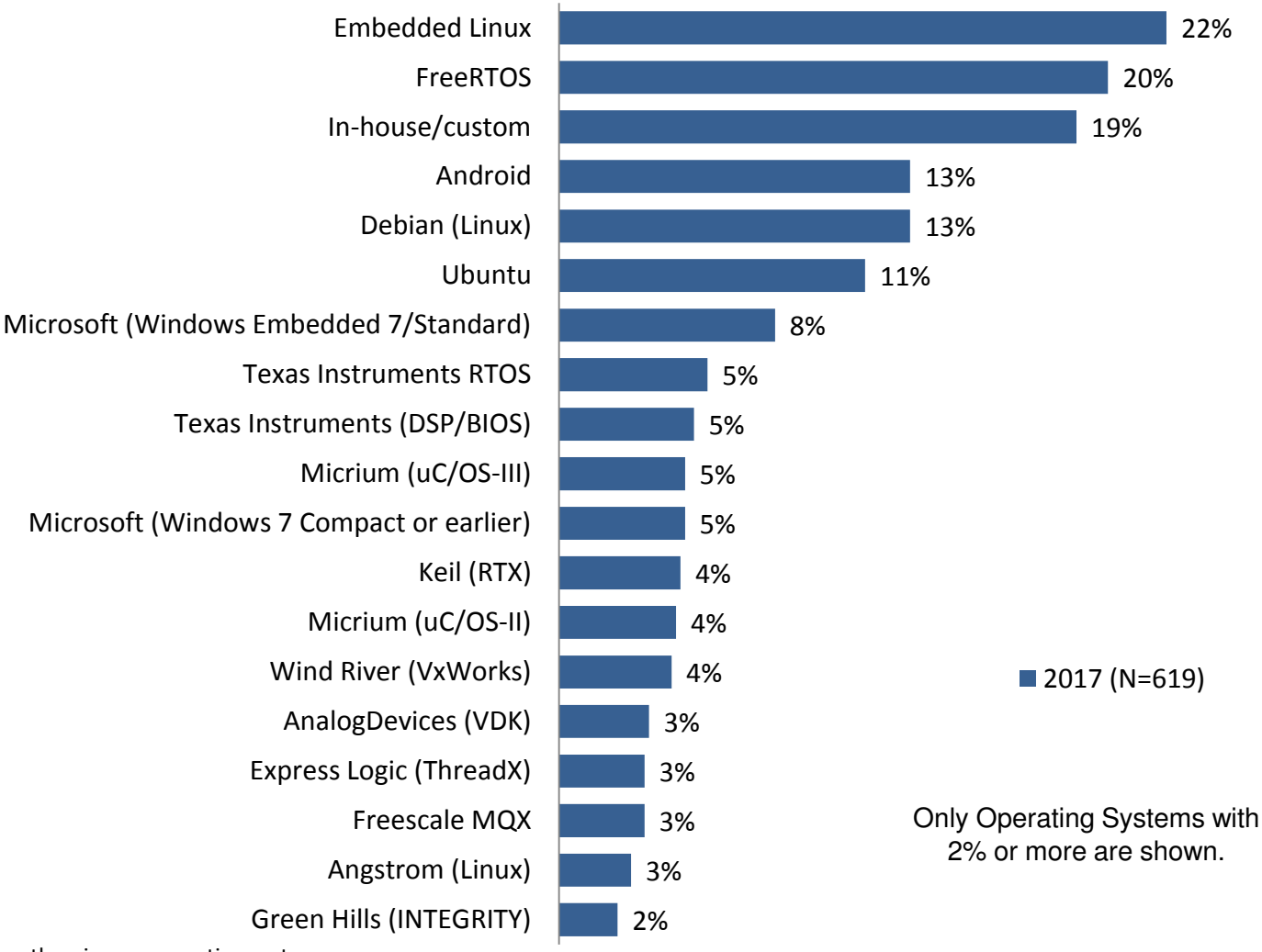
- **Distribution libre** :
  - s'assurer que son matériel est supporté (architecture, carte de développement...),
  - sinon adapter la distribution,
  - choisir parmi les logiciels proposés ceux qui apparaîtront sur le système final.
- **Distribution commerciale** :
  - s'assurer que son matériel est supporté (architecture, carte de développement...),
  - acheter une ou plusieurs licences des outils de développement propriétaires,
  - choisir parmi les logiciels proposés ceux qui apparaîtront sur le système final.

Bien sûr, les trois approches ont leurs avantages et leurs inconvénients selon le profil des développeurs impliqués :

- **Système "fait maison"** :
  - ✗ mise en place plus longue,
  - ✗⇒✓ nécessite une connaissance plus profonde des outils et mécanismes mis en jeu ce qui peut s'avérer handicapant au début mais très avantageux par la suite,
  - ✗ pas d'IDE tout intégré,
  - ✓ maîtrise complète du système et de ses constituants,
  - ✓ indépendance totale.

- **Distribution libre** :
  - ✓ prise en main rapide,
  - ✓ base d'utilisateurs importante,
  - ✗ maîtrise complète possible mais moins évidente,
  - ✗ pas toujours d'IDE,
  - ✓ indépendance totale.
- **Distribution commerciale** :
  - ✓ prise en main rapide,
  - ✓ IDE souvent très complet,
  - ✓ assistance personnalisée (payante),
  - ✗ prix du système de développement,
  - ✗ moins de maîtrise,
  - ✗ quasi dépendance vis à vis du distributeur et des outils qu'il fournit.

Please select ALL of the operating systems you are currently using.



Base: Currently using an operating system

Plate-formes orientées produits

- *Middleware* orienté vers un segment de marché (ex : smartphones, tablettes, routeurs, IVI [1]...) ;
- SDK [2] en langage de haut niveau :
  - faciliter et accélérer les développements,
  - fédérer les développeurs.
- Simplification d'accès aux ressources métier ;
- Libres ou commerciales ;
- Liées ou pas à une plate-forme matérielle.

1. IVI : ***I**n-**V**ehicle **I**nfotainment*
2. SDK : ***S**oftware **D**evelopment **K**it*

Briques de base logicielles

Noyaux Linux

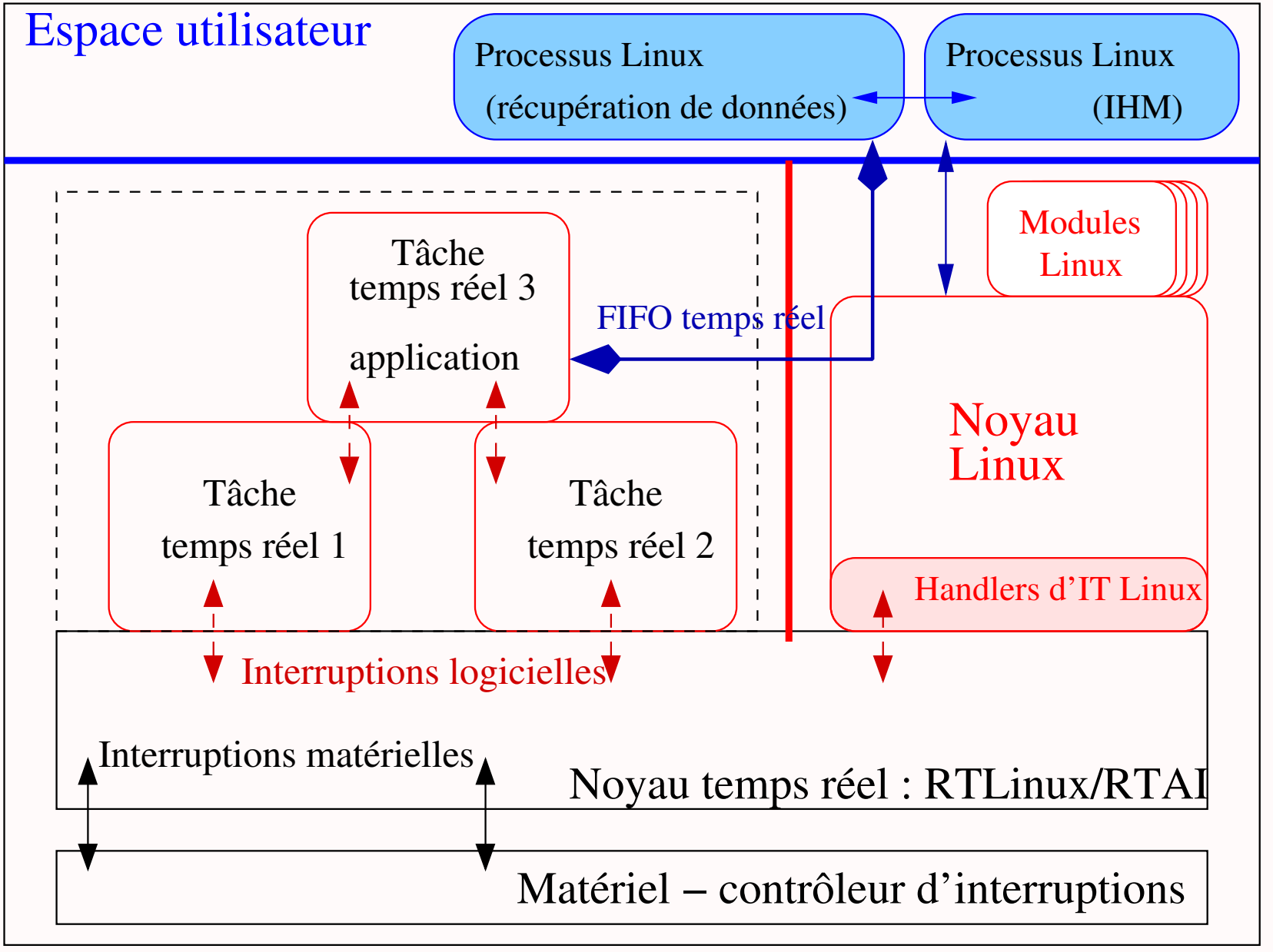
- **Linux *vanilla*** (http://www.kernel.org/) $\Rightarrow$ le noyau Linux standard ;
- **μClinux** (http://www.uclinux.org/) $\Rightarrow$ *patch* du noyau Linux pour supporter les architectures sans MMU [1] ainsi que quelques matériels périphériques spécifiques à ces architectures (intégré à partir des noyaux 2.6).

1. MMU : ***M**emory **M**anagement **U**nit*

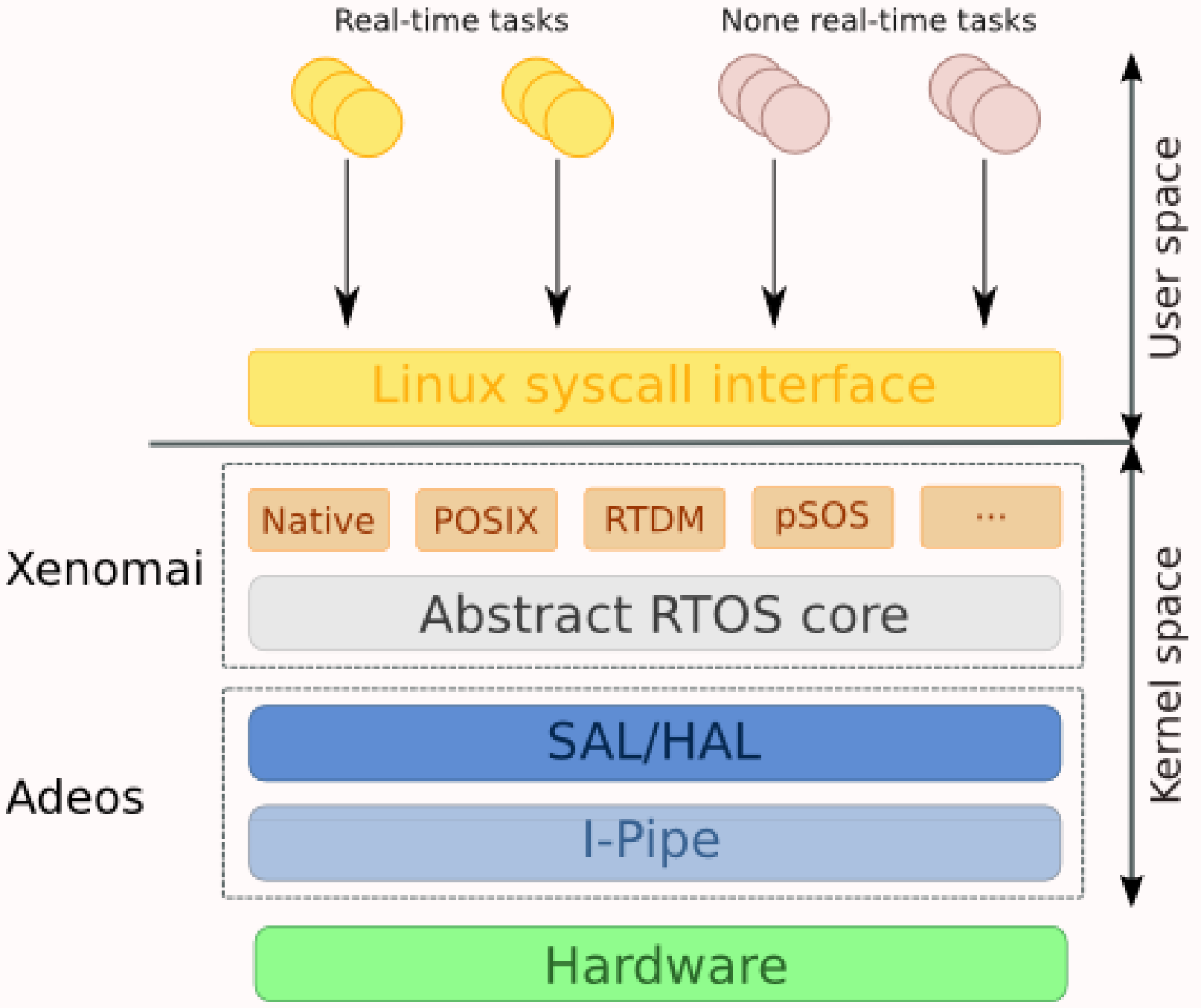
Extensions temps-réel

- **Open RTLinux/Wind River Real-Time Core** (http://fr.wikipedia.org/wiki/RTLinux) $\Rightarrow$ pionnier dans la technique du temps-réel "dur" sous Linux à base de micro-noyau mais technologie brevetée et version libre beaucoup moins évoluée que la version commerciale ;
- **RTAI** [1]**/Linux** (http://www.rtai.org/) $\Rightarrow$ API POSIX [2] et micro-noyau temps-réel libre permettant la cohabitation de tâches temps-réel dures avec un noyau Linux comme tâche de plus basse priorité (aujourd'hui basé sur l'approche sans brevet ADEOS) ;
- **Xenomai** (http://www.xenomai.org) $\Rightarrow$ concurrent/successeur libre de RTAI dont les API de programmation sont basées sur le concept de *skins* pour permettre une réutilisation maximale de code existant (POSIX, *OS* propriétaires, etc) et dont l'ordonnancement repose au choix sur une approche micro-noyau (Nucleus) ou sur ***PREEMPT_RT*** ;

1. RTAI : ***R**eal **T**ime **A**pplication **I**nterface*
2. POSIX : ***P**ortable **O**perating **S**ystem **I**nterface*



Source: Nicolas Ferre (http://noglitch.free.fr/)



Source: Xenomai.org

- *Patches* ***low latency***, ***O(1) scheduler***, ***preemtible kernel***...
  $\Rightarrow$ *patches* du noyau Linux améliorant la latence des appels système, la réactivité de l'ordonnanceur ou ajoutant la préemptibilité au niveau du noyau, ils sont souvent développés ou sponsorisés par des distributions commerciales (RedHat, MontaVista, TimeSys...) ;
- *Patch* ***PREEMPT_RT*** $\Rightarrow$ *Patch* pour apporter la préemptibilité native et le support des contraintes temps-réel dures au noyau Linux (maintenu par la communauté des développeurs de l'embarqué).

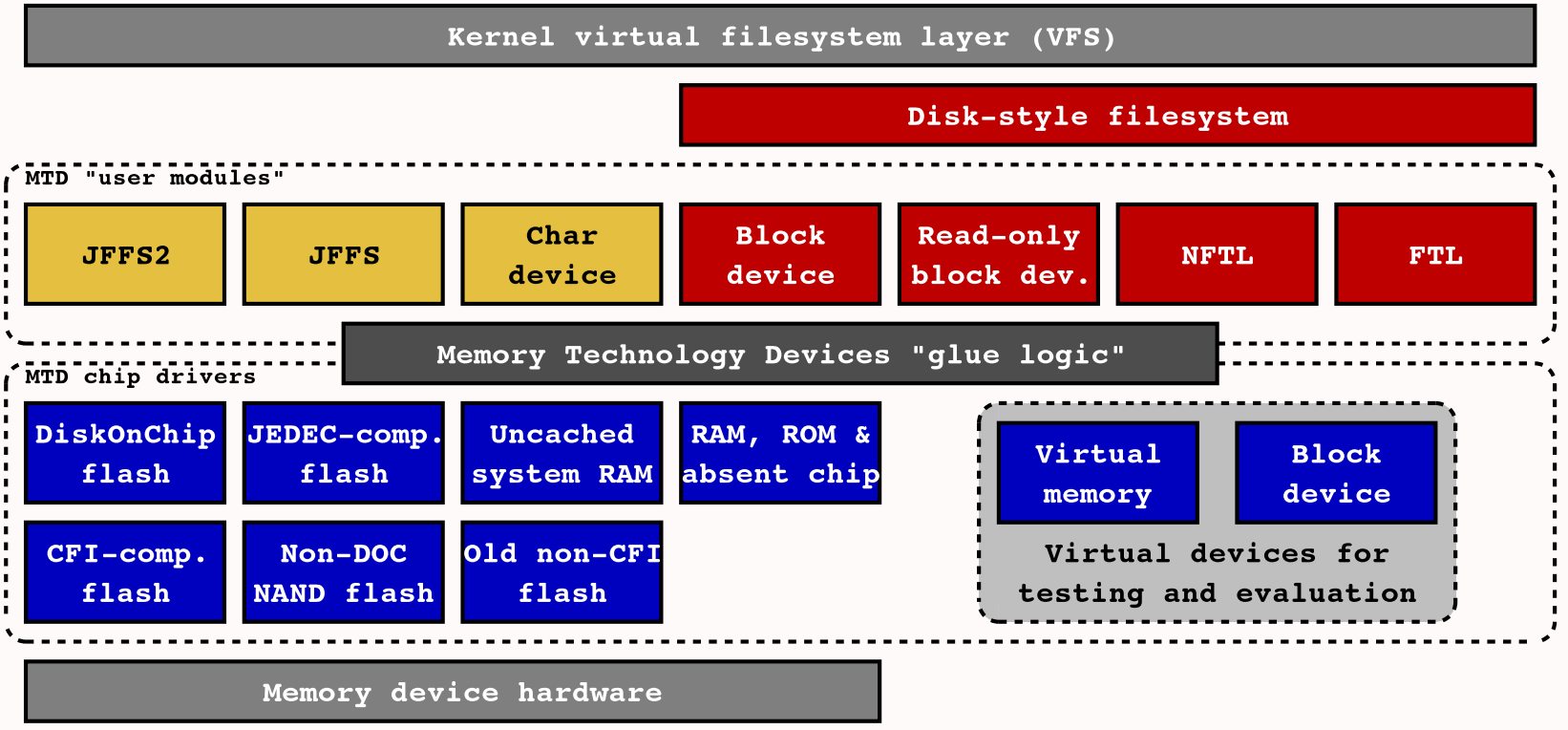
Systèmes de fichiers pour l'embarqué

- **YAFFS2** (http://www.yaffs.net/) ⇒ système de fichiers robuste (journalisation, correction d'erreur) sur mémoires Flash NAND [1] pour lesquelles il a été conçu et optimisé ;
- **JFFS2** (http://sources.redhat.com/jffs2/) ⇒ système de fichiers compressé sur mémoires Flash permettant un stockage résistant aux *crashes* et coupures brutales de courant et prenant en compte les spécificités des supports de stockage mémoire via la couche MTD [2] de Linux (voir aussi UBIFS) ;
- **ROMFS** (http://romfs.sf.net/)**/CRAMFS** [3] (http://sf.net/projects/cramfs/) / **SquashFS** (http://squashfs.sourceforge.net/) ⇒ systèmes de fichiers en lecture seule fournissant un stockage statique (construit sur le système de développement puis placé en ROM, Flash ou RAM) avec des fonctionnalités et un encombrement minimums (pas de permissions, de dates de modification, compression pour CRAMFS et SquashFS...) ;

1. NAND/NOR : accessible par pages et blocs (∼ Disque) / de manière aléatoire (∼ RAM)
2. MTD : ***M**emory **T**echnology **D**evices*
3. CRAMFS : ***C**ompressed **ROM** **F**ile**S**ystem*

- **LogFS** (http://logfs.org) $\Rightarrow$ système de fichiers *log-structured*, performant même sur les stockages Flash de grande taille (voir aussi NILFS) ;
- **AuFS** (http://aufs.sourceforge.net/) **/ OverlayFS** $\Rightarrow$ *Overlay FileSystem* et service noyau rendant un système de fichiers en lecture seule virtuellement inscriptible par union logique avec un système réellement inscriptible ;
- **TMPFS/RAMFS** $\Rightarrow$ systèmes de fichiers résidant en RAM, *swappable* (TMPFS - requiert une MMU) et redimentionnables dynamiquement, ils sont souvent utilisés pour stocker les données non conservées après redémarrage (/tmp, /var, logs...) ;
- **PRAMFS** [1] (http://pramfs.sourceforge.net/) $\Rightarrow$ système de fichiers résidant en RAM non volatile et persistant même après redémarrage, il est particulièrement adapté aux systèmes mobiles (téléphones, PDA...) où on retrouve parfois ce type de mémoires.

1. PRAMFS : ***P**rotected and **P**ersistent **RAM** **F**ile**S**ystem*



Source: Karim Yaghmour - *Building Embedded Linux Systems* (http://www.embeddedtux.org/)

Systèmes de fichiers conventionnels

- **Ext2** (http://e2fsprogs.sf.net/ext2.html) ⇒ ancien système de fichiers par défaut de Linux qui peut être monté avec écritures synchrones (`mount -o sync`) pour assurer l'intégrité des données aux dépens des performances ;
- Journalisés (**Ext3/4**, **ReiserFS**, **XFS**, **JFS**...) ⇒ les opérations sont décrites dans un journal (transactions) avant d'être effectivement effectuées permettant ainsi de conserver l'intégrité du système de fichiers et de redémarrer rapidement en cas de coupure de courant ;
- *Copy-on-write* (**BtrFS**, **ZFS**) ⇒ systèmes de fichiers modernes avec fonctionnalités de *snapshots*, multi-supports, correction d'erreur...
- **MS-DOS FAT 12/16/32** ⇒ le système de fichiers originel des *OS* Microsoft, ses différentes déclinaisons permettent de l'adapter aux différentes tailles de supports (très utilisé sur les stockages Flash des appareils électroniques grand public).

Bibliothèque C (libc)

- **Glibc** (http://www.gnu.org/software/libc/libc.html) / **EGlibc** (http://www.eglibc.org) $\Rightarrow$ bibliothèque C officielle de tous les systèmes Linux, elle est complète, performante, multi plate-forme et très bien documentée mais néanmoins volumineuse et peu adaptée dans les systèmes à faible empreinte mémoire ($\Rightarrow$ EGlibc) ;
- **μClibc** (http://www.uclibc.org/) $\Rightarrow$ presque entièrement compatible (code source) avec la Glibc, elle est beaucoup plus adaptée à l'embarqué parce que conçue pour être la plus petite possible, elle supporte également le noyau μClinux (systèmes sans MMU) ;
- **diet libc** (http://www.dietlibc.org/) $\Rightarrow$ bibliothèque C très légère permettant de créer principalement des binaires liés statiquement (non compatible Glibc) ;
- **Newlib** (http://www.sourceware.org/newlib/) $\Rightarrow$ association de plusieurs bibliothèques embarquées pouvant être utilisée en dehors de tout OS (BSP [1]).

1. BSP : ***B**oard **S**upport **P**ackage*

Outils de base

- **BusyBox** (http://www.busybox.net/) $\Rightarrow$ le couteau Suisse pour Linux embarqué, il réimplémente plus de 200 des principaux utilitaires disponibles sur les systèmes Linux en un seul exécutable léger (shells, console-tools, procps, util-linux, modutils, netutils...) ;
- **TinyLogin** (http://tinylogin.busybox.net/) $\Rightarrow$ complément parfait de BusyBox (aujourd'hui intégré) pour les systèmes embarqués utilisant les services d'authentification (contrôle d'accès et gestion des utilisateurs, groupes, mots de passe, droits...) ;
- **EmbUtils** (http://www.fefe.de/embutils/) $\Rightarrow$ ensemble d'outils Unix courants optimisés pour la taille et basés sur la diet libc ;
- **Outils GNU** (http://www.gnu.org/directory/GNU/) $\Rightarrow$ outils standards du projet GNU.

Serveurs réseau

- Web :
  - **Boa** (http://www.boa.org/),
  - **BusyBox::httpd** (http://www.busybox.net/),
  - **LightTPD** (http://www.lighttpd.net/),
  - **thttpd** (http://www.acme.com/software/thttpd/),
  - **Mbedthis AppWeb** (http://www.mbedthis.com/),
  - **GoAhead WebServer** (http://www.goahead.com/products/web_server.htm),
  - **Apache** (http://www.apache.org/)...
- FTP :
  - **sftpd** (http://safetp.cs.berkeley.edu/),
  - **ProFtpd** (http://proftpd.linux.co.uk/),
  - **tftpd**...

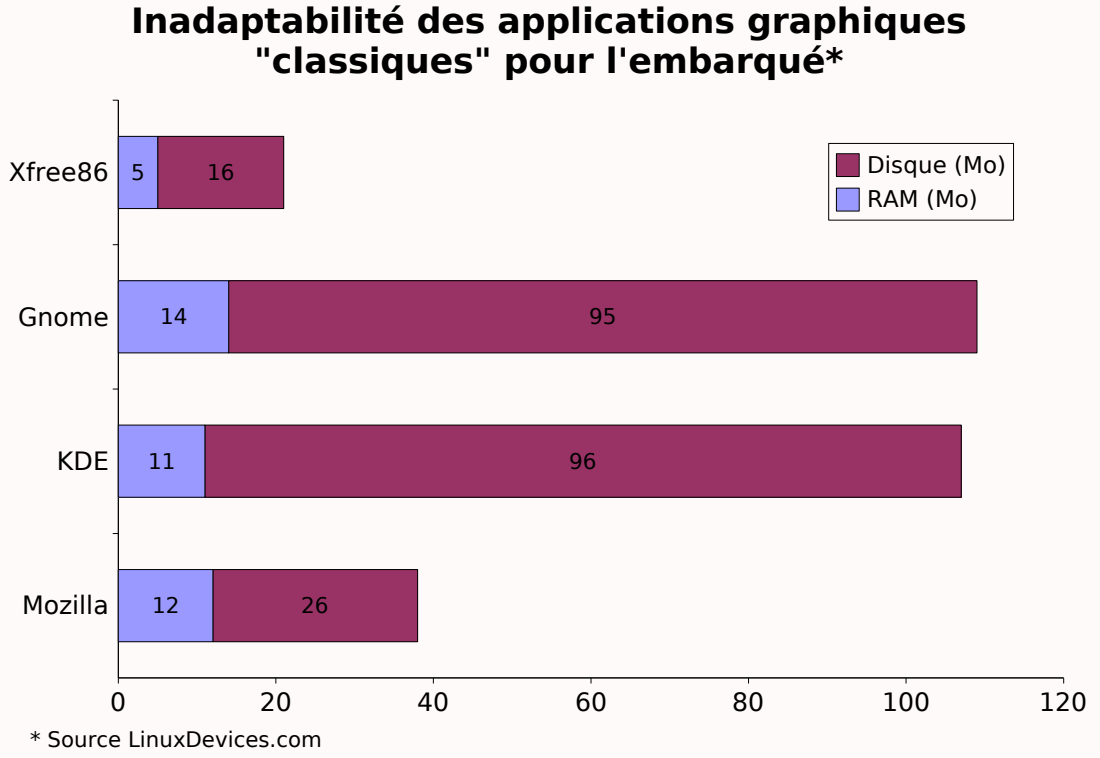
- Accès distant :
  - **OpenSSH** (http://www.openssh.com/),
  - **telnetd**,
  - **utelnetd**,
  - **gettyd**,
  - **pppd**...
- DHCP :
  - **BusyBox::udhcp** (http://www.busybox.net/),
  - **dhcpd**...
- Autres :
  - **Zebra** (http://www.zebra.org/),
  - **snmpd**...

Bases de données

- **Berkeley DB** (http://www.sleepycat.com/) $\Rightarrow$ *open source*, compatible GPL, commerciale, non relationnelle, simple, moins de 500 ko, multi plate-forme ;
- **MySQL** (http://www.mysql.com/) $\Rightarrow$ *open source* (GPL ou commerciale), relationnelle, SQL, transactionnelle, rapide, multi plate-forme, très répandue ;
- **SQLite** (http://www.sqlite.org/) $\Rightarrow$ *open source*, relationnelle, SQL, transactionnelle, sans serveur (bibliothèque autonome), multi plate-forme, fichier unique, la plus répandue dans l'embarqué, moins de 275 ko ;
- **DB2 Everyplace** (http://www.ibm.com/software/data/db2/everyplace/) $\Rightarrow$ commerciale, multi plate-forme, moins de 200 ko.

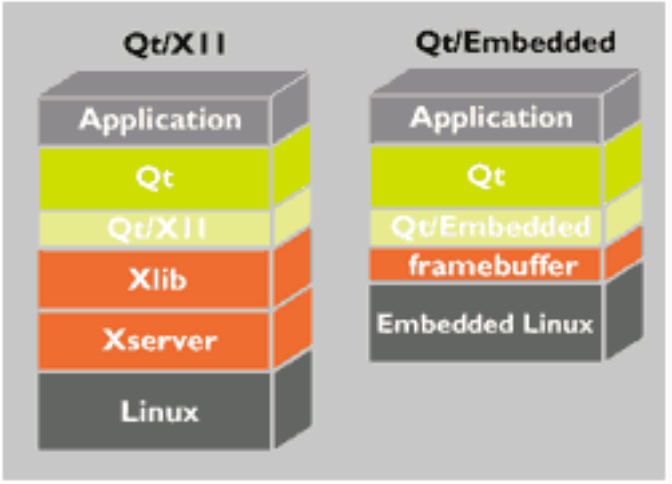
IHM [1]

- Les outils graphiques "classiques" utilisés sur les ordinateurs de bureau sont peu adaptés à l'embarqué du fait de leur taille sur stockage non volatil et en mémoire.



1. IHM : ***I**nterface **H**ome **M**achine*

- **DirectFB** (http://www.directfb.org/) $\Rightarrow$ surcouche au *framebuffer* Linux permettant de construire des IHM rapides et légères avec une abstraction complète du matériel ;
- **Gtk+** (http://www.gtk.org/) $\Rightarrow$ une version de cette bibliothèque (très répandue dans le monde Unix) utilise DirectFB pour s'affranchir de l'utilisation d'un serveur X Window ;
- **Qt/Embedded** (http://www.trolltech.com/products/embedded/) $\Rightarrow$ couche permettant l'utilisation de la bibliothèque Qt au dessus du *framebuffer* Linux (de 800 ko à 3 Mo) ;

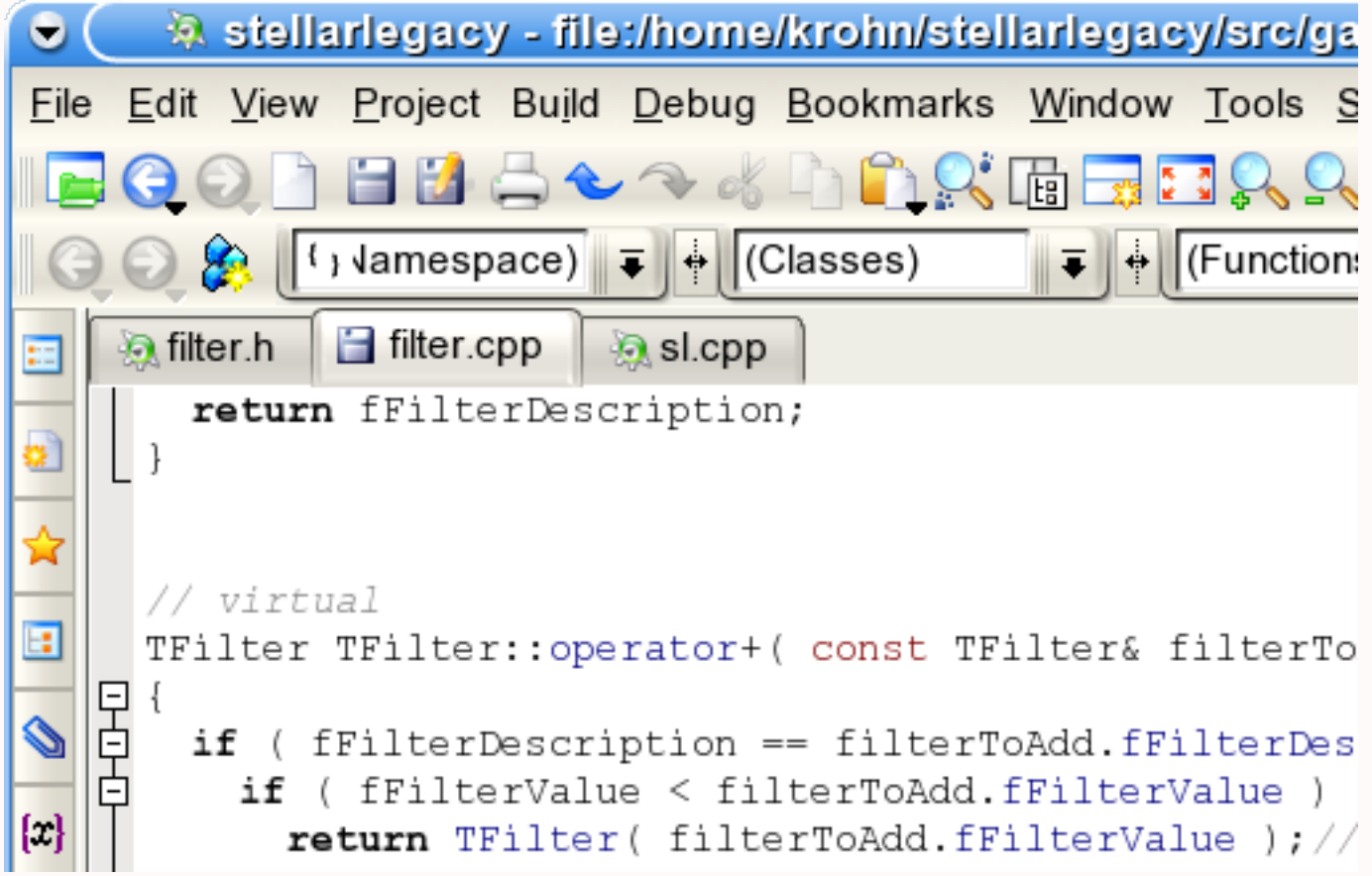


- **MicroWindows/Nano-X** (http://www.microwindows.org/)
  $\Rightarrow$ environnement graphique multi plate-forme complet possèdant sa propre API mais également une bibliothèque de compatibilité X Window (100 ko) ;
- **X Window System (XFree86/X.org)** $\Rightarrow$ (http://www.xfree86.org/ / http://www.x.org/) les implémentations libres du serveur graphique historique de tous les Unix, rapides, optimisées pour de nombreux *chipsets* graphiques mais gourmandes en mémoire et en espace disque ;
- **Tiny-X** (http://www.xfree86.org/) $\Rightarrow$ réimplémentation d'un serveur X Window pour plates-formes embarquées (1 Mo) ;
- **SDL** [1] (http://www.libsdl.org/) $\Rightarrow$ bibliothèque multi plate-forme pour le développement d'applications graphiques multimédias.

1. SDL : ***S**imple **D**irectMedia **L**ayer*

IDE [1]

- Eclipse (http://www.eclipse.org/) ;
- KDevelop (http://www.kdevelop.org/) ;
- Vim (http://www.vim.org/) ;
- Emacs (http://www.emacs.org/).



1. IDE : ***I**ntergated **D**evelopment **E**nvironment*

Références

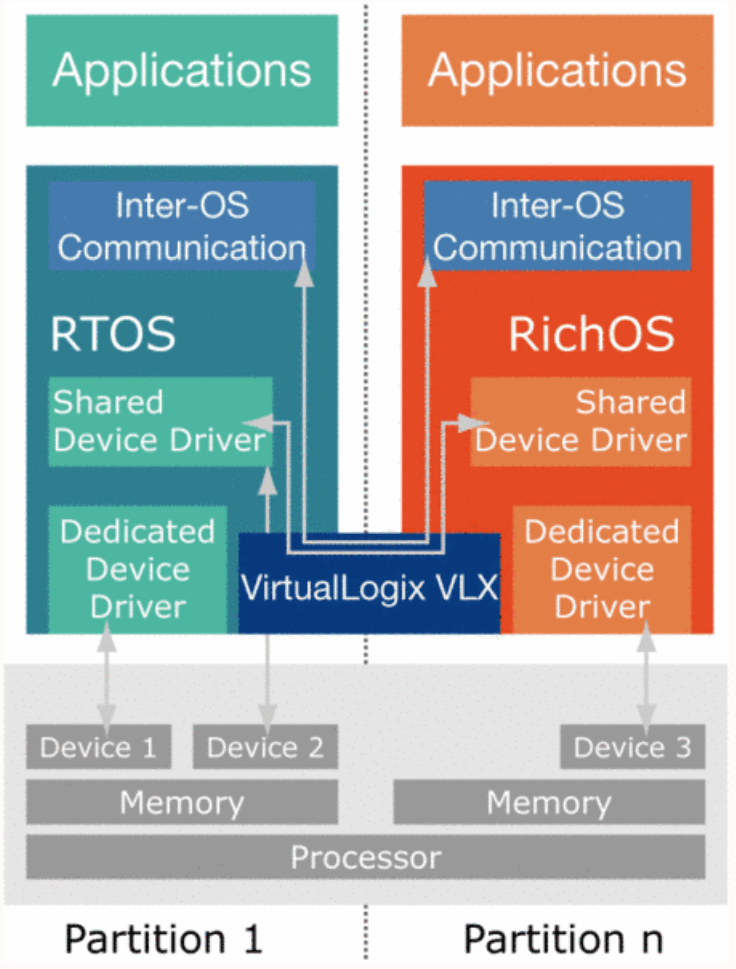
Distributions libres

- μClinux-dist (http://www.uclinux.org/pub/uClinux/dist/) ;
- SnapGear Embedded Linux (http://www.snapgear.org/) ;
- Yocto Project / OpenEmbedded ;
- Pengutronix PTXdist (http://www.pengutronix.de/software/ptxdist_en.html) ;
- Denx ELDK [1] (http://www.denx.de/ELDK/).

1. ELDK : ***E**mbedded **L**inux **D**evelopment **K**it*

Distributions commerciales

- FSMLabs (http://www.rtlinux.com/) ;
- VirtualLogix (http://www.virtuallogix.com/) ;
- Koan (http://www.klinux.org/) ;
- LynuxWorks (http://www.lynuxworks.com/) ;
- Intel/Windriver (http://www.windriver.com/) ;
- MontaVista (http://www.mvista.com/) ;
- SysGo (http://www.elinos.com/) ;
- TimeSys (http://www.timesys.com/).


Applications
Applications
Inter-OS Communication
Inter-OS Communication
RTOS
RichOS
Shared Device Driver
Shared Device Driver
Dedicated Device Driver
VirtualLogix VLX
Dedicated Device Driver
Device 1
Device 2
Device 3
Memory
Memory
Processor
Partition 1
Partition n

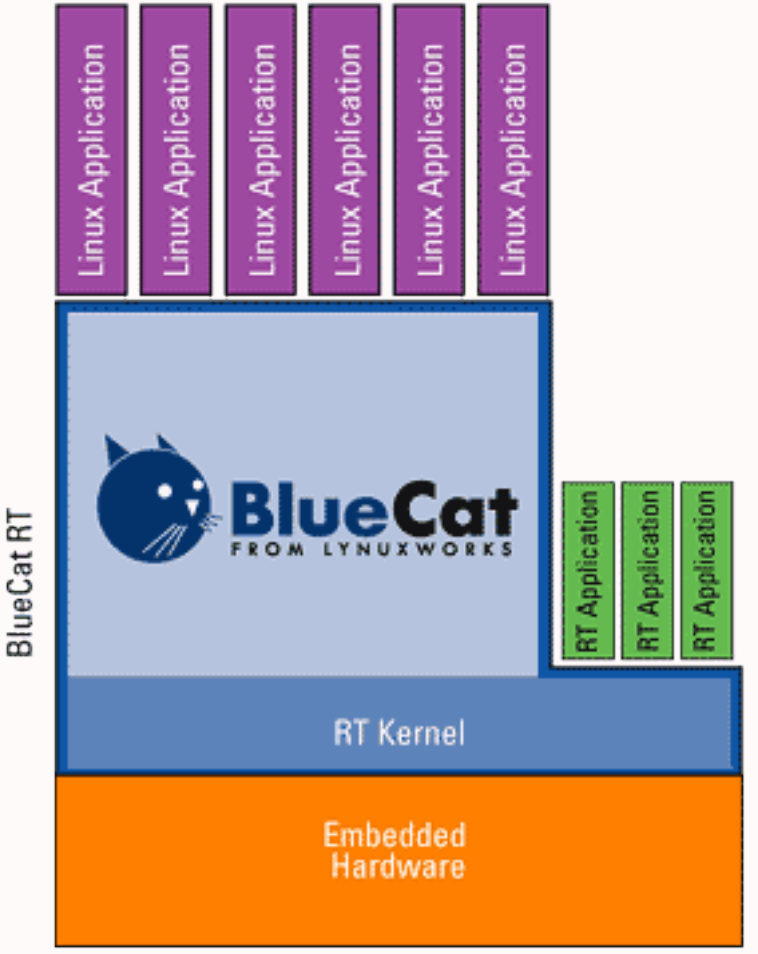

Linux Application
Linux Application
Linux Application
Linux Application
Linux Application
Linux Application
BlueCat RT
BlueCat
FROM LYNUXWORKS
RT Application
RT Application
RT Application
RT Kernel
Embedded Hardware

Plate-formes orientées produits

- Android (http://developer.android.com/) ;
- Tizen (http://developer.tizen.org/) ;
- Bada (http://developer.bada.com/) ;
- WebOS (http://developer.palm.com/) ;
- Zeroshell (http://www.zeroshell.net/).

Notions essentielles

Concepts et orthodoxie Unix

"Unix est simple. Il faut juste être un génie pour comprendre sa simplicité." - Dennis Ritchie

- Tout est fichier (données, *drivers*, liens, *pipes*, *sockets*) ;
- Modularité et *pipes* "|" :
  - programmes simples qui effectuent une seule chose et qui le font bien,
  - programmes qui collaborent et se cumulent pour faire des choses complexes,
  - privilégier les flux de texte comme interface universelle,
  - ex :`du -ks * | sort -n`.

Analyse du processus de démarrage de Linux

- *Firmware* (*bootstrap*) $\Rightarrow$ placé dans une ROM/Flash à la première adresse accédée par le processeur après un *reset*, il initialise ce dernier et passe la main au *bootloader* ;
- *Bootloader* $\Rightarrow$ chargé de lancer le noyau en le plaçant en RAM ou en l'exécutant sur place (XIP [1]) après l'avoir récupéré :
  - à une adresse prédéterminée sur un support de stockage (ROM, Flash, disque dur, CDROM...),
  - sur un système de fichiers qu'il sait accéder,
  - par le réseau (BOOTP/TFTP [2]).

1. XIP : *eXecute In Place*
2. TFTP : *Trivial File Transfert Protocol*

- Noyau $\Rightarrow$ après une phase d'initialisation de tous ses composants, il monte le système de fichiers racine (*rootfs*) disponible :
  — sur un support de stockage, ou
  — en mémoire, préchargé par le *bootloader*, ou
  — via le réseau (NFS [1]).
  avant de lancer finalement le premier processus (`init`) ;
- Le processus `init` lance les applicatifs et autres services du système...

1. NFS : ***N**etwork **F**ile**S**ystem*

Processus de compilation

`gcc -v helloworld.c -o helloworld`

- **Préprocesseur** (CPP [1]) ⇒ traite les macro-commandes des fichiers C (`#include`, `#define`, `#ifdef`, `__FUNCTION__`...) ;
- **Compilateur** (CC [2]) ⇒ transforme les fichiers sources C en fichiers sources d'assemblage dédiés à une plate-forme ;
- **Assembleur** (AS [3]) ⇒ transforme les fichiers sources d'assemblage en objets binaires (bibliothèque BFD [4]) ;
- **Édition de liens** (LD [5]) ⇒ fabrique un exécutable à partir des objets binaires et des bibliothèques statiques (archives).

1. CPP : ***C PrePr**ocessor*
2. CC : ***C C**ompiler*
3. AS : ***AS**sembler*
4. BFD : ***B**inary **F**ile **D**escriptor*
5. LD : ***L**ink e**D**itor*

Édition de liens binaires

Édition statique

```
gcc -Wall -O2 -o libmisc.o -c libmisc.c
ar -rc libmisc.a libmisc.o
gcc -Wall -O2 -o app.o -c app.c
gcc -L. -static -o app_static app.o -lmisc
```

- Toutes les dépendances sont résolues lors de l'édition des liens ;
- L'exécutable résultant est beaucoup plus gros car il contient les parties du code des bibliothèques qu'il utilise ;
- Il s'exécutera quelles que soient les versions des bibliothèques partagées présentes sur la cible (il ne s'en sert pas).

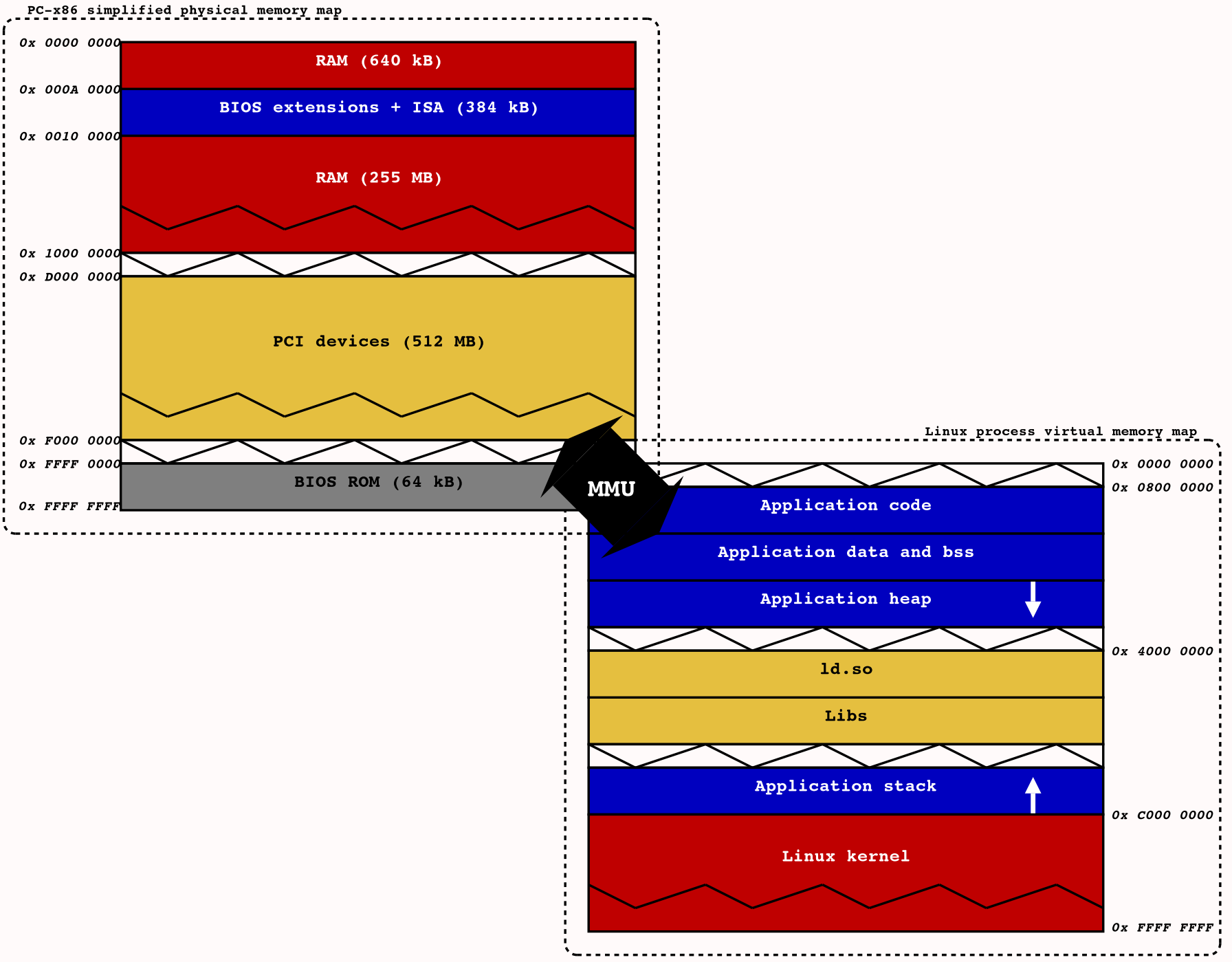
Édition dynamique

```
gcc -Wall -O2 -fpic -o libmisc.po -c
libmisc.c
gcc -shared -o libmisc.so libmisc.po
gcc -Wall -O2 -o app.o -c app.c
gcc -L. -o app_dynamic app.o -lmisc
```

- Type d'édition de liens par défaut sur les plates-formes qui supportent ce mécanisme ;
- L'édition de liens finale est réalisée au chargement de l'exécutable ;
- Si N exécutables utilisent la même version d'une bibliothèque partagée, celle-ci n'est chargée qu'une seule fois en mémoire ;
- L'exécutable est plus petit puisqu'il ne contient que son propre code.

Processus de chargement d'un exécutable lié dynamiquement

- Quand le processus est créé, le noyau charge en mémoire (`mmap()`) le fichier exécutable et le chargeur dynamique (`ld-linux.so` pour les binaires ELF) ;
- Le contrôle est donné au chargeur dynamique ;
- Le chargeur inspecte l'exécutable et les bibliothèques disponibles sur le système (via `ld.so.cache` et `ld.so.conf`) pour résoudre les dépendances (données et fonctions) et trouver les bibliothèques nécessaires ;
- Il charge ensuite en mémoire toutes les bibliothèques nécessaires, à des adresses prédéfinies, dans l'espace mémoire virtuel du processus ;
- Le chargeur saute enfin au point de départ du programme qui commence alors son exécution.


PC-x86 simplified physical memory map
0x 0000 0000
RAM (640 kB)
0x 000A 0000
BIOS extensions + ISA (384 kB)
0x 0010 0000
RAM (255 MB)
0x 1000 0000
0x D000 0000
PCI devices (512 MB)
0x F000 0000
0x FFFF 0000
BIOS ROM (64 kB)
0x FFFF FFFF
MMU
Linux process virtual memory map
0x 0000 0000
0x 0800 0000
Application code
Application data and bss
Application heap
0x 4000 0000
ld.so
Libs
Application stack
0x C000 0000
Linux kernel
0x FFFF FFFF

Les outils associés aux bibliothèques partagées

- `ldd` ⇒ affiche les bibliothèques partagées dont dépend un exécutable lié dynamiquement ou une autre bibliothèque partagée ;
- `ldconfig` ⇒ génère les liens logiques et le fichier de cache `ld.so.cache` utilisés par le chargeur dynamique en fonction des bibliothèques présentes dans `/lib`, `/usr/lib` et autres répertoires listés dans `ld.so.conf` ;
- `ltrace` ⇒ intercepte et affiche les appels aux bibliothèques partagées réalisés par un exécutable.

Statique ou dynamique ?

- Statique si :
  - dynamique pas supportée (souvent le cas avec les plates-formes sans MMU),
  - peu d'exécutables partagent les mêmes bibliothèques,
  - seulement quelques fonctions de chaque bibliothèque sont utilisées.
- Dynamique si :
  - les ressources mémoire disponibles sont très restreintes,
  - beaucoup d'exécutables sur la cible,
  - besoin de faire évoluer ou corriger les bibliothèques sans mettre à jour toute la cible.

Exécutables

Formats les plus courants

- ELF [1] $\Rightarrow$ format binaire pour les exécutables, les objets et les bibliothèques, il fait office de standard pour la plupart des Unix (dont Linux) ;
- a.out [2] $\Rightarrow$ le format de sortie par défaut de l'assembleur et de l'éditeur de liens des systèmes Unix ;
- bFLT [3] (Flat) $\Rightarrow$ format de fichiers exécutables léger, dérivé du format a.out et utilisé par le projet μClinux, il supporte la compression ;
- COFF [4] $\Rightarrow$ format binaire objet issu de l'ABI [5] d'Unix System V, il est l'ancêtre du format ELF.

1. ELF : ***E****xecutable and* ***L****inkable* ***F****ormat*
2. a.out : ***a****ssembler* ***out****put*
3. bFLT : ***b****inary* ***FLaT*** *format*
4. COFF : ***C****ommon* ***O****bject* ***F****ile* ***F****ormat*
5. ABI : ***A****pplication* ***B****inary* ***I****nterface*

Manipulations sur les exécutables

- **allègement** ⇒ l'utilitaire `strip` supprime les symboles, informations de débogage et autres contenus superflus d'un fichier binaire (exécutable ou bibliothèque) ;
- **conversion** ⇒ l'utilitaire `elf2flt` permet de convertir un binaire ELF en bFLT ;
- **compression** ⇒ le format bFLT supporte la compression (totale ou seulement données) avec décompression au moment de l'exécution par le noyau (`elf2flt [-z|-d]`).

μClinux vs Linux

Différences fondamentales

- μClinux est adapté aux plates-formes sans MMU :
  - pas de protection mémoire,
  - pas de mécanisme de mémoire virtuelle (modèle mémoire plat).

Conséquences

- L'appel système `fork()` n'est pas implémenté ⇒ utilisation de `vfork()` (API BSD) :
  — père et fils partagent **entièrement** leur espace mémoire (y compris la pile), et
  — le père est suspendu jusqu'à ce que son fils appelle `execve()` ou `exit()`.
- Fragmentation rapide si nombreuses allocations/libérations dynamiques de mémoire (`malloc()`/`free()`) ⇒ préférer l'allocation d'un *pool* au démarrage de l'application ;

- Pile de taille fixée à la compilation ;
- Utilisation d'exécutables relogeables :
  - adressage relatif (PIC [1]) $\Rightarrow$ code limité à 32 ko (jump 16 bits du 68k), ou
  - adressage absolu complètement relogeable (références modifiées au chargement par le noyau) $\Rightarrow$ plus lourd et plus lent au chargement.
- Pas de mécanisme de *swap*.

1. PIC : ***P**osition **I**ndependant **C**ode*

Méthodes et outils de développement

Terminologie

- On distingue deux entités :
  - la **cible** (*target*) est la plate-forme matérielle qui va accueillir l'*OS* et le ou les applicatifs embarqués,
  - l'**hôte** (*host*) est la plate-forme de développement sur laquelle sont mis au points les différentes parties logicielles de la cible.
- L'hôte et la cible sont rarement basés sur la même architecture matérielle et n'utilisent pas forcément le même *OS* ;
- Un même hôte peut servir à développer plusieurs cibles différentes en même temps.

Méthodologies de développement

- On distingue habituellement 4 méthodologies de développement pour les systèmes embarqués :
  - développement connecté,
  - développement par stockage amovible,
  - développement sur cible,
  - développement par prototypage.
- Elles sont plus ou moins dictées par les contraintes de stockage, de performances et d'accessibilité du système cible.

Développement connecté

- La cible est reliée à l'hôte par un lien physique (Ethernet, USB, série, JTAG...) ;
- Le lien permet :
  - de mettre à jour la cible à distance, et/ou
  - de déboguer la cible, et/ou
  - à la cible de récupérer le noyau et le *rootfs* dynamiquement (TFTP, NFS...).
- C'est la configuration la plus rencontrée.

Développement par stockage amovible

- La cible comporte un *bootloader* minimaliste ;
- Le développeur place le noyau et le *rootfs* sur le stockage amovible (CompactFlash, EEPROM...) via un programmateur adéquat installé sur l'hôte ;
- Le support est ensuite installé sur la cible ;
- Un émulateur de ROM permet d'améliorer le processus pour le faire ressembler à un développement connecté.

Développement sur cible

- Possible uniquement sur les systèmes embarqués disposant de suffisamment d'espace de stockage et de mémoire pour pouvoir faire fonctionner un compilateur (ex : systèmes SOB avec disque dur) ;
- La cible comporte ses propres outils de développement natifs (éditeur, compilateur, débogueur...) ;
- Le développeur accède à la cible soit directement, à l'aide d'un clavier et d'un écran, soit par le réseau à partir de l'hôte (ssh, telnet...) ;
- Généralement basée sur une distribution Linux du marché, celle-ci peut éventuellement être "allégée" lors du passage au système final (suppression des outils de conception, documentation, paquets inutiles...).

Développement par prototypage

- Le développement du ou des applicatifs est réalisé sur cible ou sur une plate-forme de développement (d'architecture similaire à la cible) à partir d'une distribution Linux classique sur disque dur ;
- Développement (en parallèle ?) d'un système de fichiers racine léger qui :

— contient uniquement l'arborescence ainsi que les utilitaires et bibliothèques (attention aux versions) nécessaires aux applicatifs,

— sera placé par la suite sur le système cible (ex : CompactFlash pour s'affranchir d'un disque dur) après intégration des applicatifs.

Compilation croisée (*cross-compilation*)

Chaîne de compilation

- On parle de *Cross-Platform Development Toolchain* ;
- Elle se compose des éléments suivants :

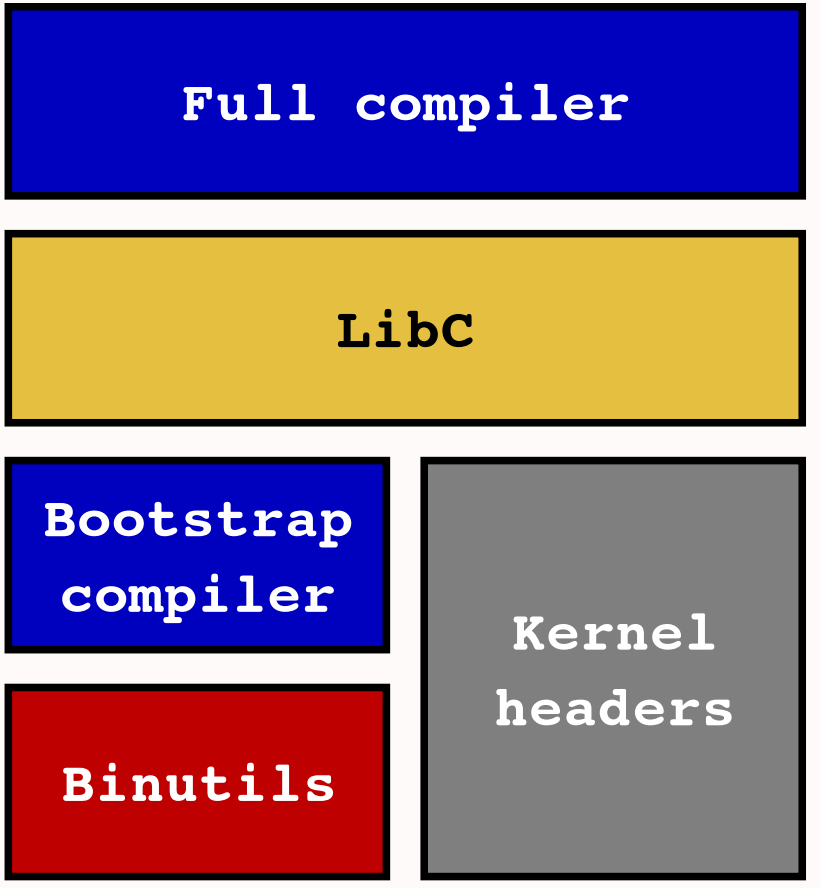
— un ensemble d'outils de manipulation de fichiers binaires (binutils),

— un compilateur C/C++ (GCC),

— un noyau (Linux),

— une bibliothèque C (μClibc).

Interdépendances dans la chaîne de compilation



Fabrication de la chaîne de compilation croisée

- L'hôte doit déjà posséder une chaîne de compilation locale (voir distributions) lui permettant de compiler des applicatifs natifs (pour lui même) ;
- Désignation de la cible au format GNU :

— ARM ⇒ `arm-linux`
— PowerPC ⇒ `powerpc-linux`
— MIPS (big endian) ⇒ `mips-linux`
— MIPS (little endian) ⇒ `mipsel-linux`
— i386 ⇒ `i386-linux`...

- Configuration de la cible et des chemins :

```
$ export TARGET=arm-linux
$ export PREFIX=/usr/local
$ export INCLUDE=$PREFIX/$TARGET/include
```

- Installation des binutils :

```
$ tar zxvf binutils-2.10.1.tar.gz
$ mkdir build-binutils
$ cd build-binutils
$ ../binutils-2.10.1/configure
  -target=$TARGET -prefix=$PREFIX
$ make
$ make install
```

Le répertoire `$PREFIX/bin` contient alors `arm-linux-ar`, `arm-linux-as`, `arm-linux-ld`, `arm-linux-nm`, `arm-linux-objdump`, `arm-linux-strip`...

- Installation du *bootstrap cross-compiler* :

```
$ tar zxvf gcc-2.95.3.tar.gz
$ mkdir build-bootstrap-gcc
$ cd build-bootstrap-gcc
$ ../gcc-2.95.3/configure -target=$TARGET
  -prefix=$PREFIX -without-headers
  -with-newlib -enable-languages=c
$ make all-gcc
$ make install-gcc
```

- Configuration des en-têtes du noyau Linux :

```
$ tar jxvf linux-2.4.24.tar.bz2
$ cd linux-2.4.24
$ make ARCH=arm CROSS_COMPILE=$TARGET-
  menuconfig
$ mkdir $INCLUDE
$ cp -r linux/ $INCLUDE
$ cp -r asm-generic/ $INCLUDE
$ cp -r asm-arm/ $INCLUDE/asm
```

- Installation de la bibliothèque C :

```
$ tar jxvf uClibc-0.9.16.tar.bz2
$ cd uClibc-0.9.16
$ make CROSS=$TARGET- menuconfig
$ make CROSS=$TARGET-
$ make CROSS=$TARGET- PREFIX="" install
```

- Installation complète du compilateur :

```
$ mkdir build-gcc
$ cd build-gcc
$ ../gcc-2.95.3/configure -target=$TARGET
  -prefix=$PREFIX -enable-languages=c,c++
$ make all
$ make install
```

Utilisation de la chaîne de compilation croisée

```
$ arm-linux-gcc exemple.c -o exemple
$ arm-linux-size exemple
$ arm-linux-strip exemple
...
```

ScratchBox/Crosstool-NG/buildroot

- http://www.scratchbox.org, http://crosstool-ng.org, http://buildroot.uclibc.org ;
- Boites à outils simplifiant la mise en place de chaînes complètes de compilation croisée ;
- Fonctionnalités et techniques variées :

— gestion de la compilation croisée et la configuration croisée,

— mécanisme de *sandbox* (QEMU + chroot) pour isoler la cible de l'hôte,

— construction noyau et *rootfs* avec *buildroot*.

Yocto Project / OpenEmbedded

- http://www.yoctoproject.org, http://www.openembedded.org ;
- *Frameworks* de création de sa propre distribution embarquée ;
- Basés sur une base commune *OpenEmbedded-Core* et l'outil *BitBake* ;
- Gestion de la compilation croisée ;
- Supportent plusieurs gestionnaires de packages ;
- Tests via QEMU ;
- Plusieurs sous projets :
  - intégration Eclipse,
  - génération d'un SDK pour sa ditribution (ADK [1]),
  - EGlibc...

1. ADK : ***A**pplication **D**evelopment **K**it*

Débogage et optimisation

Débogage distant avec GDB [1]

- Débogage symbolique ;
- Processus de la cible contrôlé à distance depuis l'hôte avec GDB (ou surcouche graphique comme DDD [2]) ;
- Sur la cible, deux possibilités :

— `gdbstub` : ensemble de points d'interception (*hooks*) et de gestionnaires d'évènements (*handlers*), disponibles dans le *firmware* ou le noyau de la cible, et permettant de déboguer cette dernière à distance en manipulant directement le matériel, ou

— `gdbserver` : client léger installé sur la cible et permettant de déboguer, à distance, une application de cette dernière en utilisant l'*OS* (appel système `ptrace()` des Unix).

1. GDB : ***G**NU **De**Bugger*
2. DDD : ***D**ata **D**isplay **D**ebugger*

gdbserver

- Il ne permet de déboguer que les applications mais il est plus simple et plus répandu dans le monde Linux ;
- La partie `gdbserver`, disponible sur la cible, récupère les commandes de débogage en provenance du GDB de l'hôte et lui transmet les résultats ;
- Plusieurs liens de communication sont disponibles (lien série, TCP/IP...) ;
- Exemple de connexion via TCP/IP :

target> `gdbserver :2222 hello`

host> `arm-linux-gdb hello` **ou** `ddd -gdb -debugger arm-linux-gdb hello`

(gdb) `target remote 192.168.0.10:2222`

(gdb) `list`

(gdb) `break 10`

(gdb) `cont`

strace

- Utilitaire permettant d'intercepter (utilise `ptrace()`) tous les appels système réalisés par un processus et de les afficher de manière humainement lisible ;
- Possibilité de filtrer les appels système à intercepter (ex : `strace -e trace=open,close,read ls`);
- Peut être installé sur la cible le temps des développements, puis retiré de la version finale ;
- Se trouvant à la frontière entre l'espace utilisateur et l'espace noyau, il permet de savoir qui, de l'application ou du noyau (plus rare ;-), se comporte mal.

LTTng [1] / *SystemTap*

- Logiciels d'analyse système et noyau complets ;
- Comprennent une partie noyau (génération des traces) et une partie utilisateur (acquisition des traces) ;
- Les traces générées sur la cible peuvent ensuite être analysées sur l'hôte à l'aide d'applications graphiques dédiées ;
- Permet l'analyse complète des problématiques de synchronisation, communication inter-processus, *timing* utilisateur et noyau...

1. LTT : ***L**inux **T**race **T**oolkit **n**ext **g**eneration*

Profiling

- Technique consistant à réaliser des statistiques sur les temps d'exécution des différentes partie d'une application à des fins de débogage et/ou d'optimisation ;
- Le noyau Linux comporte son propre système de *profiling* (option de démarrage `profile=n`, `/proc/profile` et utilitaire `readprofile`) qui analyse le pointeur d'instruction à chaque interruption du *timer* système et tient ainsi à jour une table des fonctions du noyau les plus utilisées ;
- Pour les applications, GCC comporte aussi son propre système de *profiling* (option `-pg`) qui génère un fichier de statistiques à chaque instanciation de l'application (exploitable *a posteriori* avec l'outil `gprof`).

Débogage par le matériel

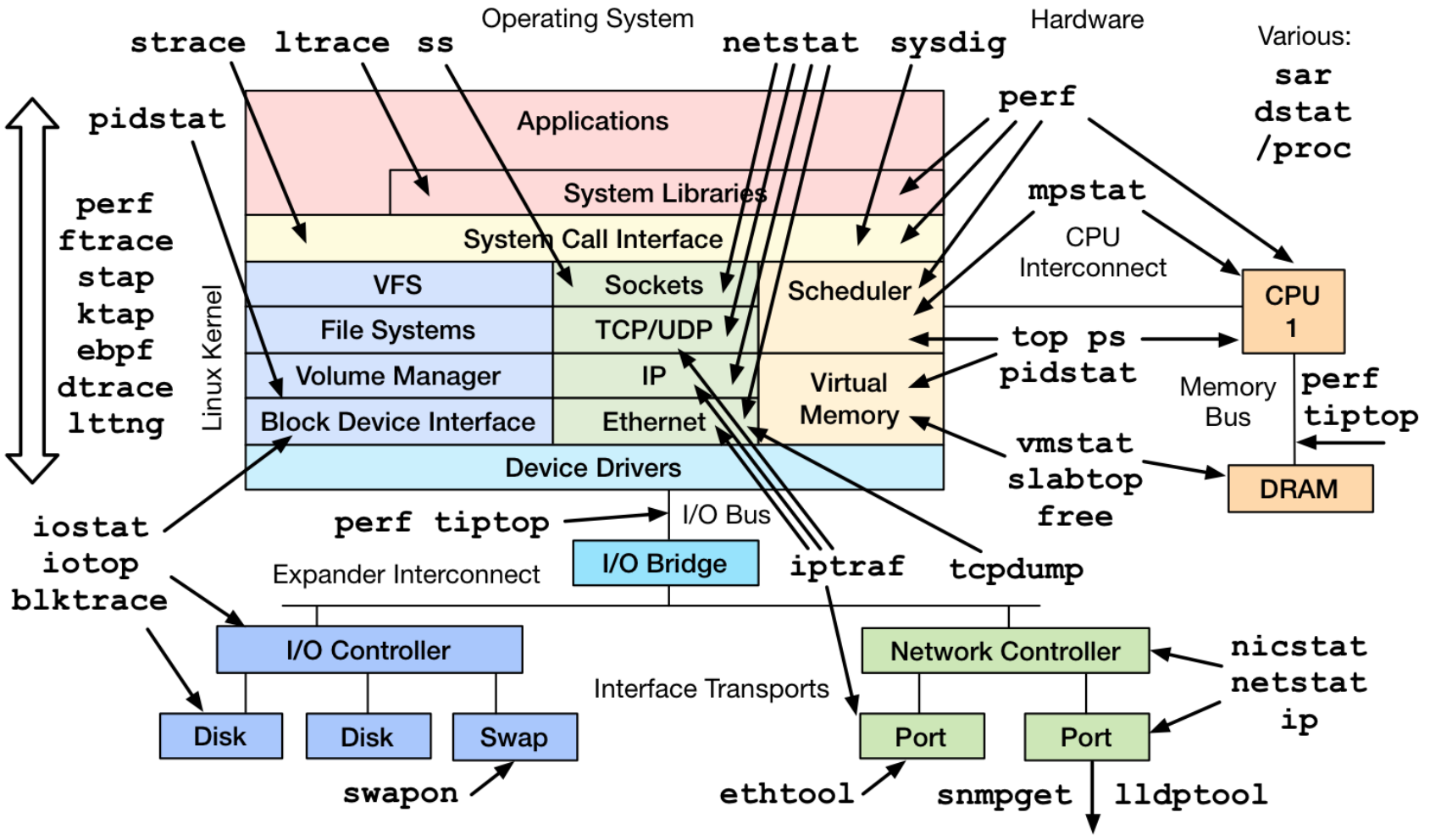
- Émulateur ROM $\Rightarrow$ composant RAM (*overlay RAM*) piloté par un lien série et inséré en lieu et place de la ROM du système cible via un adaptateur, il permet de télécharger du code rapidement sur la cible, de tracer les accès ROM (points d'arrêt) ;
- ICE [1] $\Rightarrow$ émulateur matériel qui combine les fonctionnalités d'un émulateur CPU, d'un émulateur ROM et d'un analyseur logique, il remplace physiquement (via un adaptateur) le processeur d'une carte de développement et permet de simuler jusqu'aux entrées/sorties ;

1. ICE : ***I**n **C**ircuit **E**mulator*

- JTAG [1] $\Rightarrow$ sur un microcontrôleur il est utilisé en OCD [2] et permet, via un lien série ou USB, de placer des points d'arrêt, de lire et d'écrire les registres et de simuler des sorties, il est souvent utilisé pour programmer la Flash embarquée dans les composants ;
- BDM [3] $\Rightarrow$ intermédiaire entre le ICE et le JTAG spécifique aux processeurs Motorola, il permet un contrôle total de ceux-ci via un connecteur 10 ou 26 points.

1. JTAG : ***J**oint **T**est **A**ction **G**roup*
2. OCD : ***O**n **C**hip **D**ebugger*
3. BDM : ***B**ackground **D**ebug **M**onitor*

Linux Performance Observability Tools



Émulation et virtualisation logicielles

- QEMU (http://www.bellard.org/qemu/) $\Rightarrow$ émulateur multi plate-forme de processeurs (x86, ARM, SPARC, PowerPC) incluant deux modes de fonctionnement (système complet / applicatifs Linux) ;
- ARMulator (http://www.gnu.org/software/gdb/) $\Rightarrow$ extension du débogueur GNU (GDB) permettant d'émuler les différents coeurs ARM (*big endian*, *little endian* et *thumb*) ;
- Xcopilot (http://www.uclinux.org/pub/uClinux/utilities/) $\Rightarrow$ émulateur PalmPilot complet (68k, timers, ports série, écran tactile...) ayant servi au développement de la version initiale de μClinux ;
- POSE [1] (http://sf.net/projects/pose/) $\Rightarrow$ émulateur multi plate-forme de PDA Palm, il est une amélioration de Copilot apportée par Palm Software ;

1. POSE : ***P**alm **OS** **E**mulator*

- UML [1] (http://user-mode-linux.sf.net/) $\Rightarrow$ noyau Linux utilisant le noyau Linux comme plate-forme matérielle, il permet de faire fonctionner plusieurs noyaux comme des processus standards du noyau hôte ;
- VMware (http://www.vmware.com/) / VirtualBox (http://www.virtualbox.org/) $\Rightarrow$ machine virtuelle multi plate-forme (commerciale/libre) émulant une architecture x86/PC complète (CPU, BIOS, disques, réseau...) qui supporte la majeure partie des *OS* fonctionnant sur cette architecture ;
- Bochs (http://bochs.sf.net/) $\Rightarrow$ émulateur x86/PC multi plate-forme sous licence LGPL ;
- MAME [2] (http://www.mame.net/) $\Rightarrow$ émulateur de machines arcades aussi diverses que variées, il émule des processeurs encore utilisés dans l'embarqué (z80, M6809, 68k...).

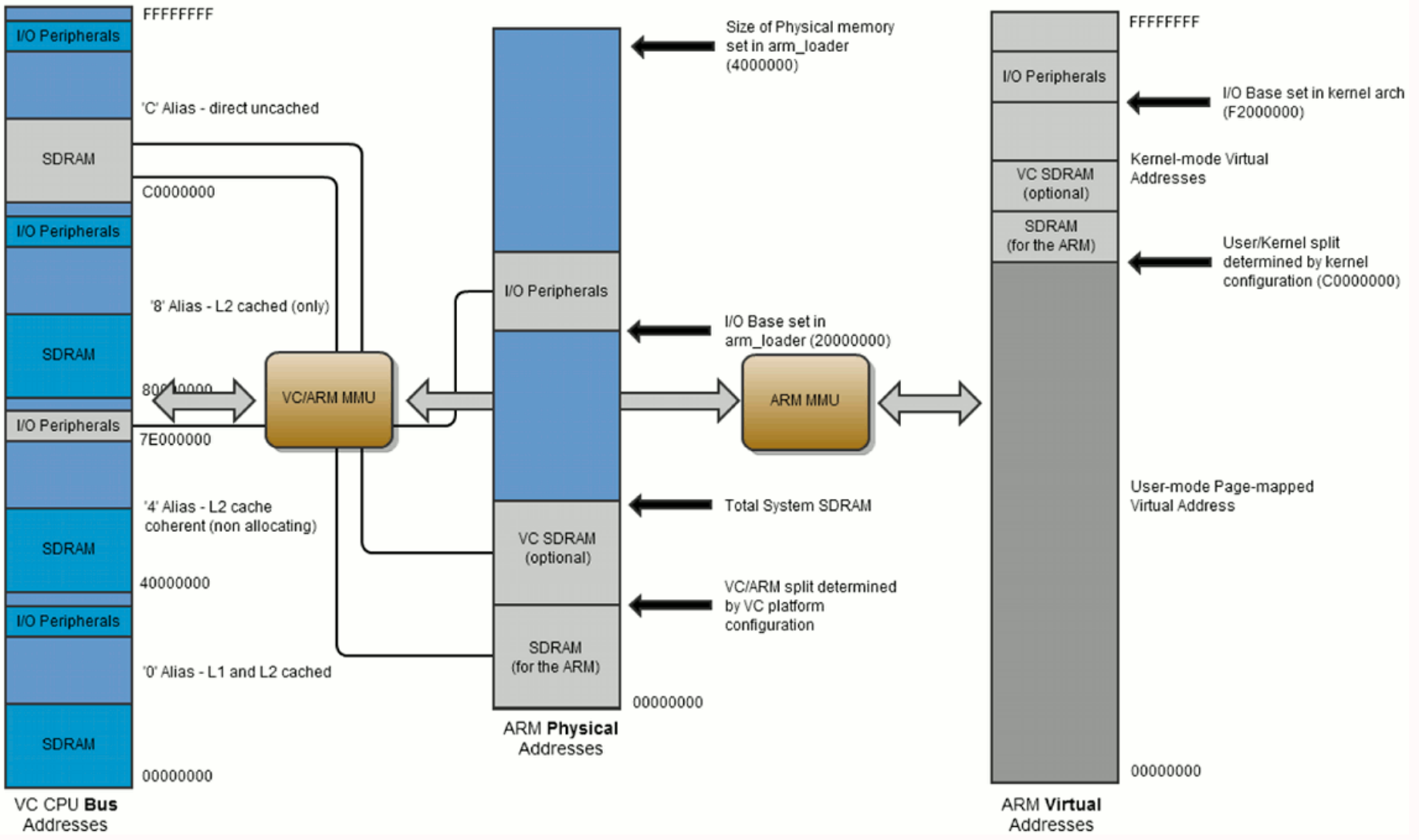
1. UML : ***U**ser **M**ode **L**inux*
2. MAME : ***M**ultiple **A**rcade **M**achine **E**mulator*

Étude de cas

Raspberry Pi

- Mise en oeuvre d'un système embarqué sur la plate-forme *Raspberry Pi* (ARM11, Flash SD, 256Mo SDRAM et contrôleur Ethernet 100 Mb) :

— mise en place de la distribution Raspbian,
— développement croisé d'applicatifs,
— débogage distant via gdbserver,
— serveur web et PHP,
— création d'un système *from scratch* avec *buildroot*.

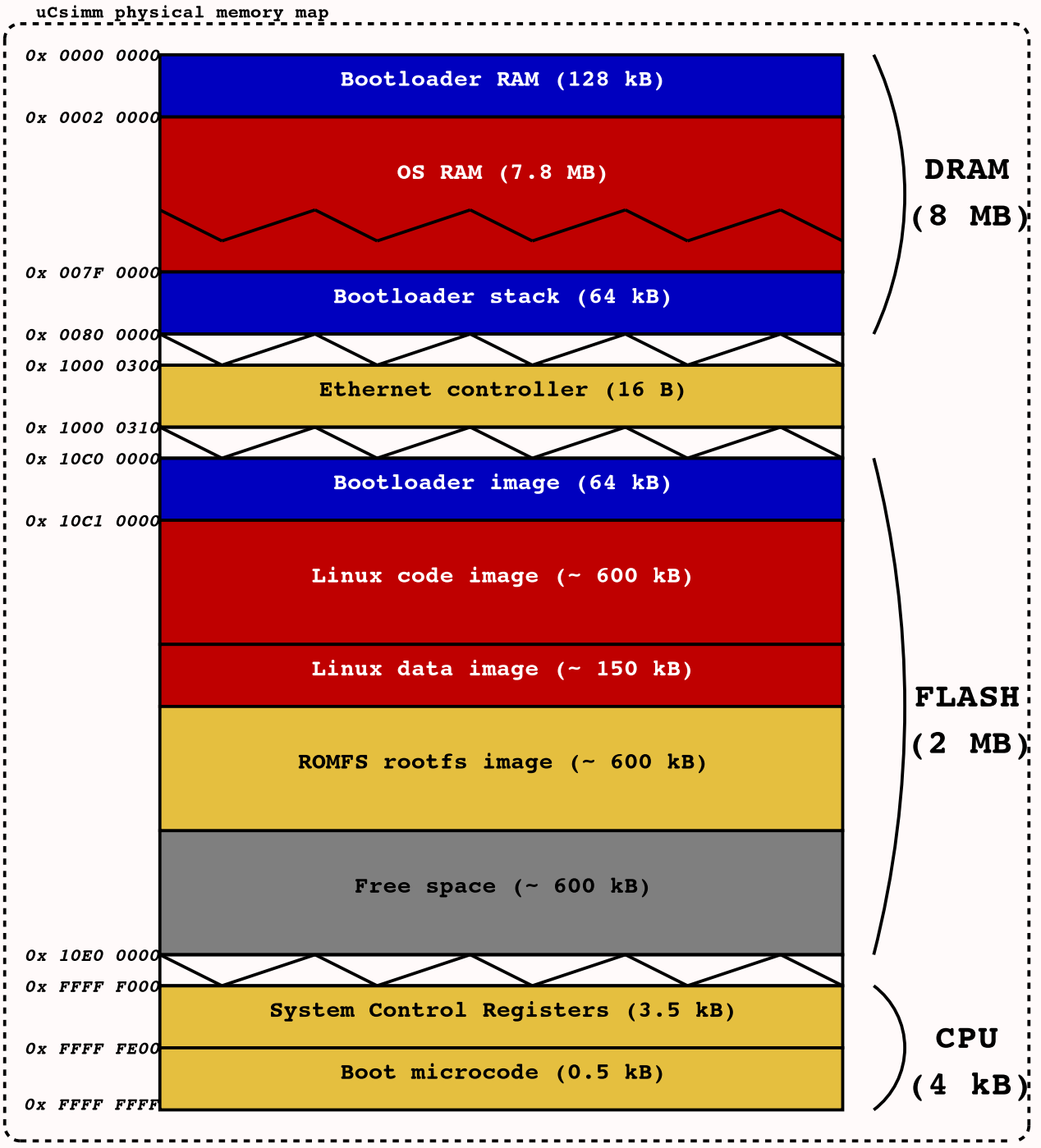

FFFFFFFF
I/O Peripherals
'C' Alias - direct uncached
SDRAM
C0000000
I/O Peripherals
'8' Alias - L2 cached (only)
SDRAM
80000000
I/O Peripherals
7E000000
'4' Alias - L2 cache coherent (non allocating)
SDRAM
40000000
I/O Peripherals
'0' Alias - L1 and L2 cached
SDRAM
00000000
VC CPU Bus Addresses
VC/ARM MMU
I/O Peripherals
VC SDRAM (optional)
SDRAM (for the ARM)
00000000
ARM Physical Addresses
Size of Physical memory set in arm_loader (4000000)
I/O Base set in arm_loader (20000000)
Total System SDRAM
VC/ARM split determined by VC platform configuration
ARM MMU
FFFFFFFF
I/O Peripherals
I/O Base set in kernel arch (F2000000)
Kernel-mode Virtual Addresses
VC SDRAM (optional)
SDRAM (for the ARM)
User/Kernel split determined by kernel configuration (C0000000)
User-mode Page-mapped Virtual Address
00000000
ARM Virtual Addresses

μCsimm

- Mise en oeuvre d'un système embarqué sur la plate-forme μCsimm (DragonBall EZ, 2 Mo Flash, 8 Mo DRAM et contrôleur Ethernet 10 Mb) :

— mise en place de la distribution μClinux,
— développement croisé d'applicatifs,
— débogage distant via gdbserver,
— serveur web et CGI [1].

1. CGI : ***C**ommon **G**ateway **I**nterface*


uCsimm physical memory map
0x 0000 0000
Bootloader RAM (128 kB)
0x 0002 0000
OS RAM (7.8 MB)
0x 007F 0000
Bootloader stack (64 kB)
0x 0080 0000
0x 1000 0300
Ethernet controller (16 B)
0x 1000 0310
0x 10C0 0000
Bootloader image (64 kB)
0x 10C1 0000
Linux code image (~ 600 kB)
Linux data image (~ 150 kB)
ROMFS rootfs image (~ 600 kB)
Free space (~ 600 kB)
0x 10E0 0000
0x FFFF F000
System Control Registers (3.5 kB)
0x FFFF FE00
Boot microcode (0.5 kB)
0x FFFF FFFF
DRAM
(8 MB)
FLASH
(2 MB)
CPU
(4 kB)

Références

Livres

- ***Building Embedded Linux Systems*** - Karim Yaghmour (http://www.embeddedtux.org/) ;
- **Linux Embarqué** - Pierre Ficheux (http://pficheux.free.fr/) ;
- ***Embedded Linux*** - John Lombardo ;
- ***Embedded Linux*** - Craig Hollabaugh ;
- ***Linux for Embedded and Real-time Applications*** - Doug Abbott.

Portails web

- LinuxGizmos (http://linuxgizmos.com/) ;
- *Embedded Linux Wiki* (http://elinux.org/) ;
- *The Linux Foundation* (http://www.linuxfoundation.org/) ;
- *The Linux Documentation Project* (http://www.tldp.org/).

Web

- Cours de Patrice Kadionik à l'ENSEIRB (http://www.enseirb.fr/ kadionik/) ;
- Pages perso. de Bill Gatliff (http://billgatliff.com/) ;
- Pages perso. de Nicolas Ferre (http://nferre.free.fr/) ;
- *The μClinux directory* (http://uclinux.home.at/) ;
- *Embedded Debian* (http://www.emdebian.org/) ;
- Systèmes de fichiers (http://en.wikipedia.org/wiki/List_of_file_systems).

Hardware

- OpenHardware (http://www.openhardware.net/) ;
- LART (http://www.lart.tudelft.nl/) ;
- OpenCores (http://www.opencores.org/) ;
- GumStix (http://www.gumstix.com/) ;
- ArmadeouS (http://www.armadeus.com/) ;
- Raspberry Pi (http://www.raspberrypi.org/).

Fin