

ISEN-Toulon

<http://www.isen.fr/>



Introduction au développement de systèmes embarqués avec Linux

Julien Gaulmin

<julien23@free.fr>

Version 2007r1.

Cette présentation est diffusable selon les termes de la *Creative Commons License* (<http://creativecommons.org/licenses/by-sa/2.0/fr/deed.fr>)

Attribution-ShareAlike 2.0



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 philosophiques,
- Les autres *OS*,
- Les licences,
- Limites.

3. Solutions :

- Briques de base logicielles,
- Distributions libres,
- Distributions commerciales.

4. Notions essentielles :

- Analyse du processus de démarrage de Linux,
- Processus de compilation,
- Édition de liens binaires,
- Exécutables,
- μ Clinux 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.

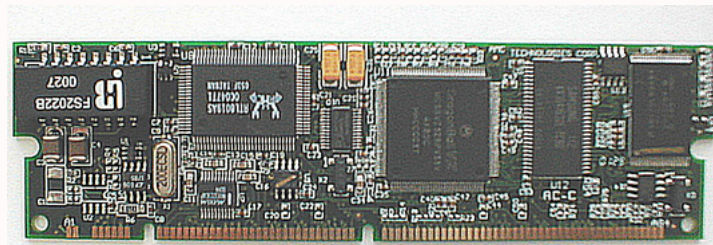
L'embarqué

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, borne d'accès Internet, *tablet-PC*, *diskless-PC* ;
- Balladeur numérique, platine DVD, décodeur/enregistreur TV ;
- Automate industriel, robot, machine outil ;
- Routeur, point d'accès *WiFi*, téléphone IP, μ Csimm ;
- Téléphone GSM, 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
⇒ **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 (RTOS¹);

¹RTOS : *Real Time Operating System*

- Deux formes principales d'applicatifs temps-réel :
 - **temps-réel "dur"** (*hard real-time*) ⇒ le système doit **absolument** répondre à un événement donné dans un **temps déterminé** (ABS, système anti-missile...),
 - **temps-réel "mou"** (*soft real-time*) ⇒ 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, VoIP¹ ...).

¹VoIP : *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 communicants ;
- Miniaturisation et baisse des coûts des composants ;
- Évolution du marché total de l'embarqué¹ de 32 milliards de dollars en 1998 à 67 milliards en 2004.

¹Estimation de *Business Communications Co. Inc.* en 1999

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.
- ✓ Aujourd'hui, émergence de Linux et des logiciels libres comme alternative sérieuse à tous ces monopoles, du prototypage au produit fini.

Les acteurs

- Communautés de développeurs ;
- Éditeurs de logiciels ;
- Sociétés de service ;
- Fabricants de composants ;
- Industriels ;
- Scientifiques, universitaires et étudiants ;
- Organisations (ELC¹, Emblix, TV Linux Alliance, RTLF², LDPS³, FHSG⁴, LSB⁵, FSF⁶, OpenGroup...)
- Médias (portails web, éditeurs, presse...).

¹ELC : *Embedded Linux Consortium*

²RTLF : *Real-Time Linux Foundation*

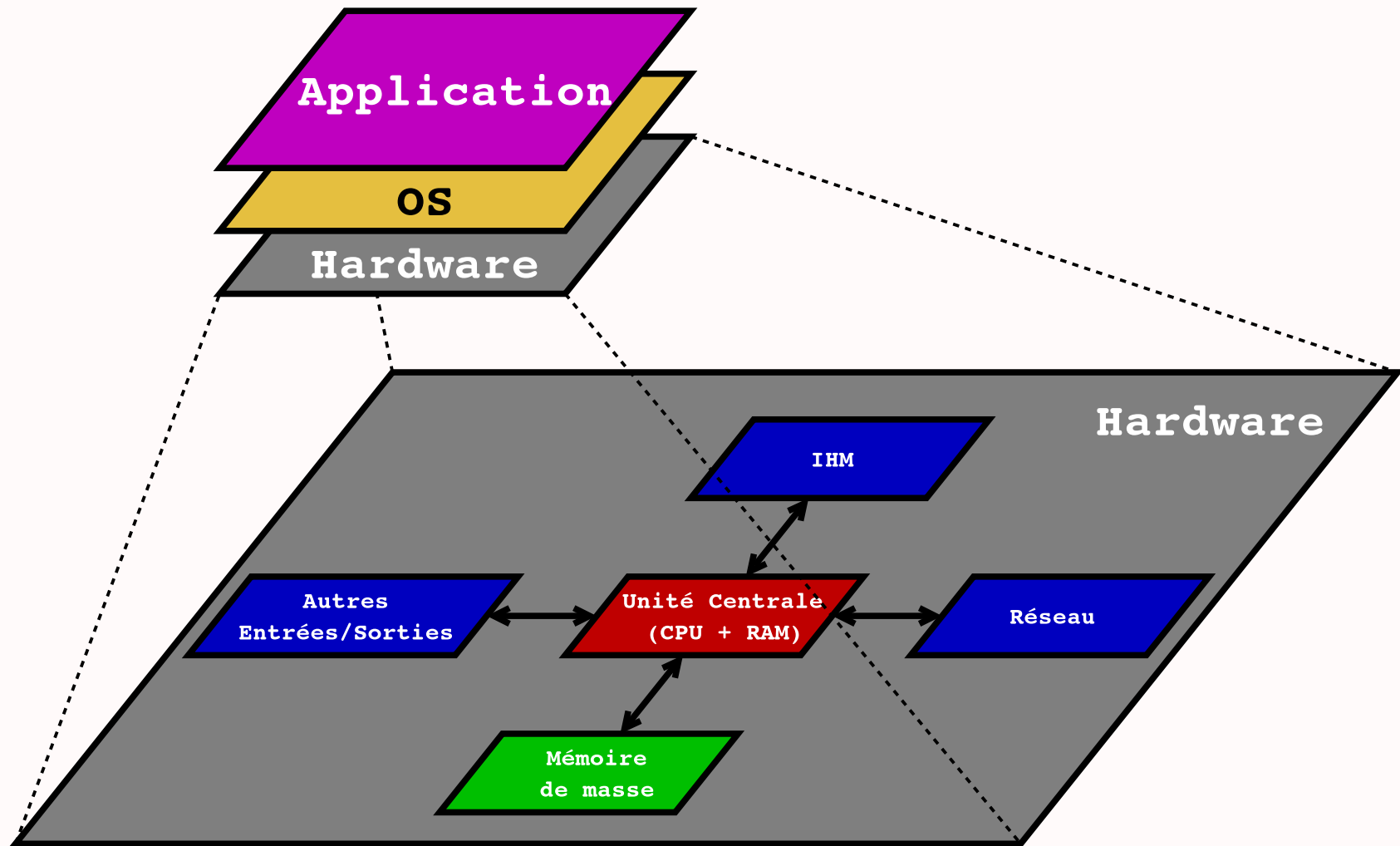
³LDPS : *Linux Development Platform Specification*

⁴FHSG : *Filesystem Hierarchy Standard Group*

⁵LSB : *Linux Standard Base*

⁶FSF : *Free Software Foundation*

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 (SOB¹, SOC²...);
- Adaptée aux ressources nécessaires \Rightarrow pas de superflu (économies d'échelle).

¹SOB : *System On Board*

²SOC : *System On Chip*

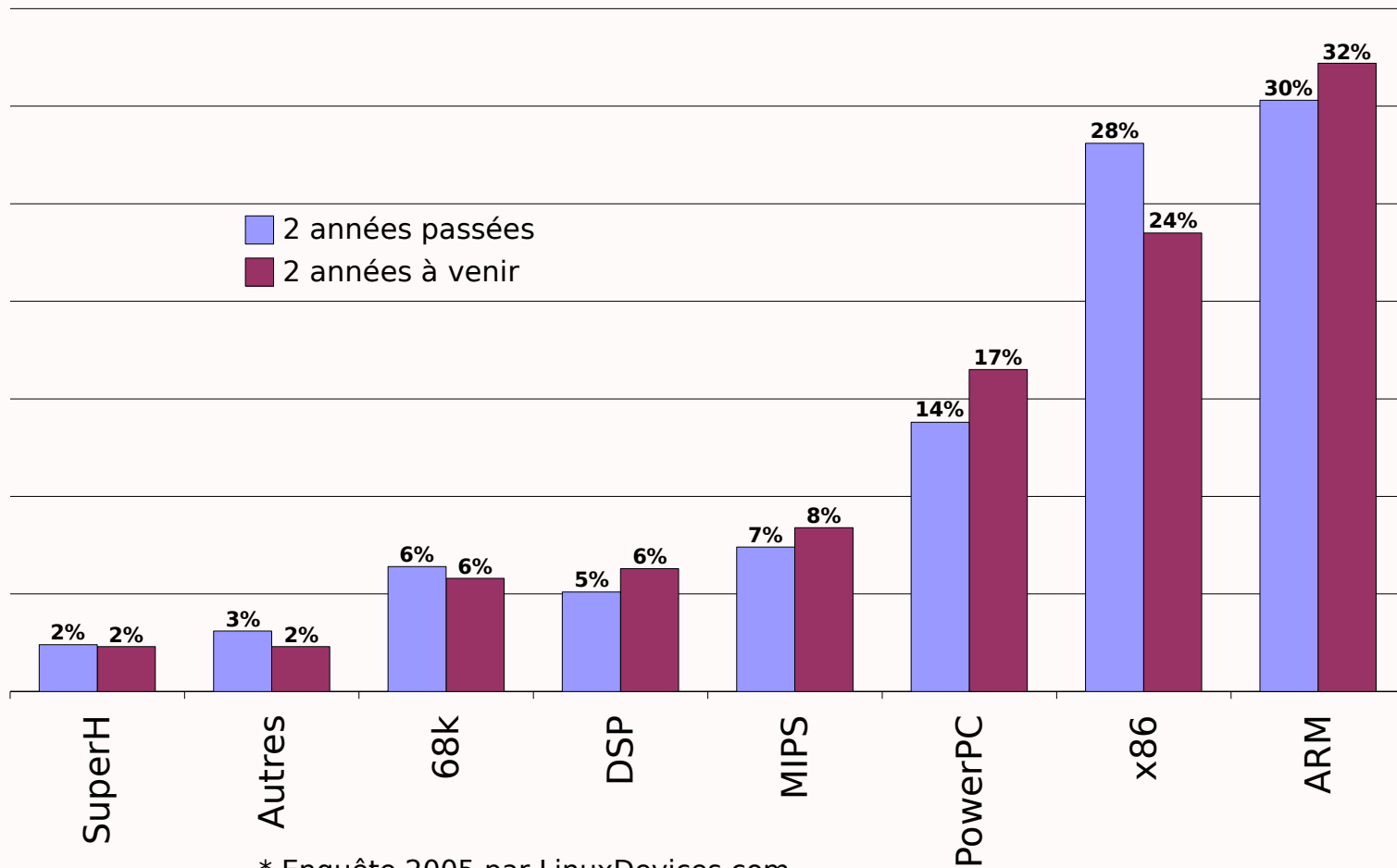
Familles de processeurs

- **Généralistes** : x86, ia64, x64, PowerPC, Sparc...
- **Basse consommation** : ARM¹ (ARMx, XScale, StrongARM), SuperH, MIPS², PowerPC, PARISC, i960...
- **SOC** : 68k (Motorola DragonBall et ColdFire), x86 (Geode, Elan, ZFx86, Eden, C7), ARM (Altera EPXAx, Intel SA-11x, Samsung S3Cx), PowerPC (IBM 405GP, Motorola PowerQUICC et MPC500), Etrax...
- **ASIC ou FPGA** avec coeur(s) ARM, MIPS, PowerPC...

¹ARM : *Advanced RISC Machine*

²MIPS : *Microprocessor without Interlocked Pipeline Stages*

Les CPU utilisés dans les systèmes embarqués*



* Enquête 2005 par LinuxDevices.com

Bus de communication

- VME¹ ⇒ VMEbus, VME64, VME64x, VME320, VXI², IP-Module, M-Module...
- PCI³ ⇒ CompactPCI (cPCI), PCI-X, PXI⁴, PMC⁵, PC/104+, MiniPCI...
- PCIe⁶ ⇒ XMC, AdvancedTCA, AMC, ExpressCard, MiniCard...
- PCMCIA⁷ ⇒ PCMCIA, PC Card, CardBus...
- ISA⁸ ⇒ PC/104...

¹VME : *Versa Module Eurocard*

²VXI : *VMEbus eXtension for Instrumentation*

³PCI : *Peripheral Component Interconnect*

⁴PXI : *PCI eXtension for Instrumentation*

⁵PMC : *PCI Mezzanine Card*

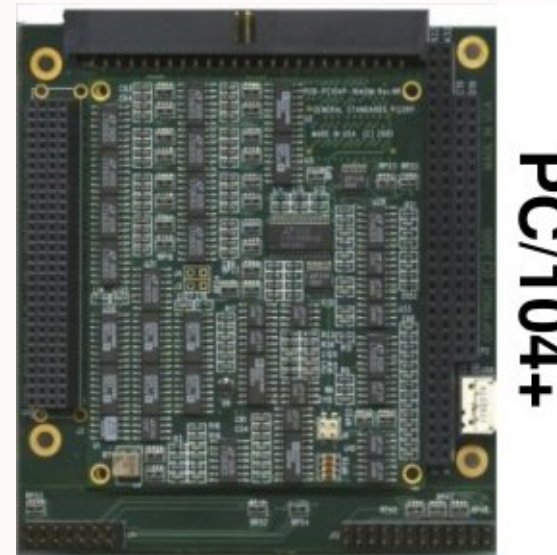
⁶PCIe : *PCI eXPRESS*

⁷PCMCIA : *Personnal Computer Memory Card International Association*

⁸ISA : *Industry Standard Architecture*



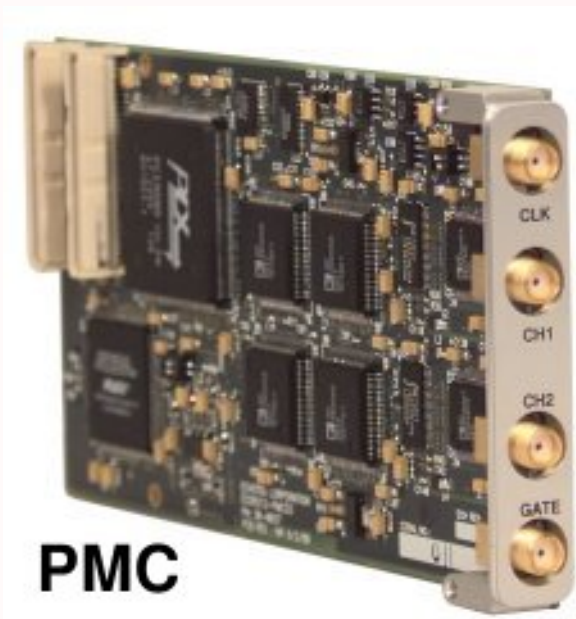
VME



PC/104+



PXI



PMC

- Parallèle \Rightarrow ATA/ATAPI¹ (IDE²), SCSI³, Centronics/IEEE1284...
- Série \Rightarrow I²C⁴, RS232, RS485, USB⁵, IEEE1394, Serial ATA...
- Réseau \Rightarrow Ethernet, FDDI⁶, X.25, WiFi/802.11, BlueTooth/ZigBee/WUSB/Wibree/802.15.x, IrDA⁷, ATM⁸, Token Ring, GSM⁹/GPRS¹⁰/UMTS¹¹ ...

¹ATAPI : *AT Attachment Packet Interface*

²IDE : *Intergated Drive Electronics*

³SCSI : *Small Computer Systems Interface*

⁴I²C : *Inter-Integrated Circuit*

⁵USB : *Universal Serial Bus*

⁶FDDI : *Fibber Distributed Data Interface*

⁷IrDA : *Infrared Data Association*

⁸ATM : *Asynchrounous Transfert Mode*

⁹GSM : *Global System for Mobile communications*

¹⁰GPRS : *General Packet Radio Service*

¹¹UMTS : *Universal Mobile Telecommunications System*

Mémoires de masse

- Disques Durs ⇒ 2,5”, 3,5”, microdrive...
- Mémoires Flash ⇒ FlashDisk, DiskOnChip, CompactFlash, CFI¹, SD Card, Memory Stick, USB Mass Storage...
- ROM², EPROM, EEPROM, UVPROM...
- CD, DVD, disquette...
- Combinaison des précédentes.

¹CFI : *Common Flash Interface*

²ROM : *Read Only Memory*

Entrées/Sorties

- Entrées :
 - entrées TOR¹ (collecteurs ouverts, optocoupleurs...),
 - capteurs/convertisseurs (pression, audio, température...),
 - clavier, boutons poussoirs, dalles tactiles, télécommandes (infrarouge, radio...),
 - lecteurs de tags ou de codes barres.
- Sorties :
 - sorties TOR (relais, optocoupleurs, logique...),
 - écrans et afficheurs LCD,
 - bips, synthèse vocale, alarmes,
 - imprimantes en tous genres (papier, étiquettes, photos...).

¹TOR : Tout Ou Rien

Réseau

- Technologies :
 - classiques \Rightarrow Ethernet, ATM, X.25...
 - de terrain (*fieldbus*) \Rightarrow CAN¹, RS232, RS485, CPL², ARCnet³,
 - sans fil \Rightarrow WiFi/802.11, IrDA, BlueTooth/ZigBee/WUSB/Wibree/802.15.x, GSM/GPRS/UMTS...
- Pour quoi faire ?
 - communiquer,
 - partager des informations,
 - contrôler et superviser.

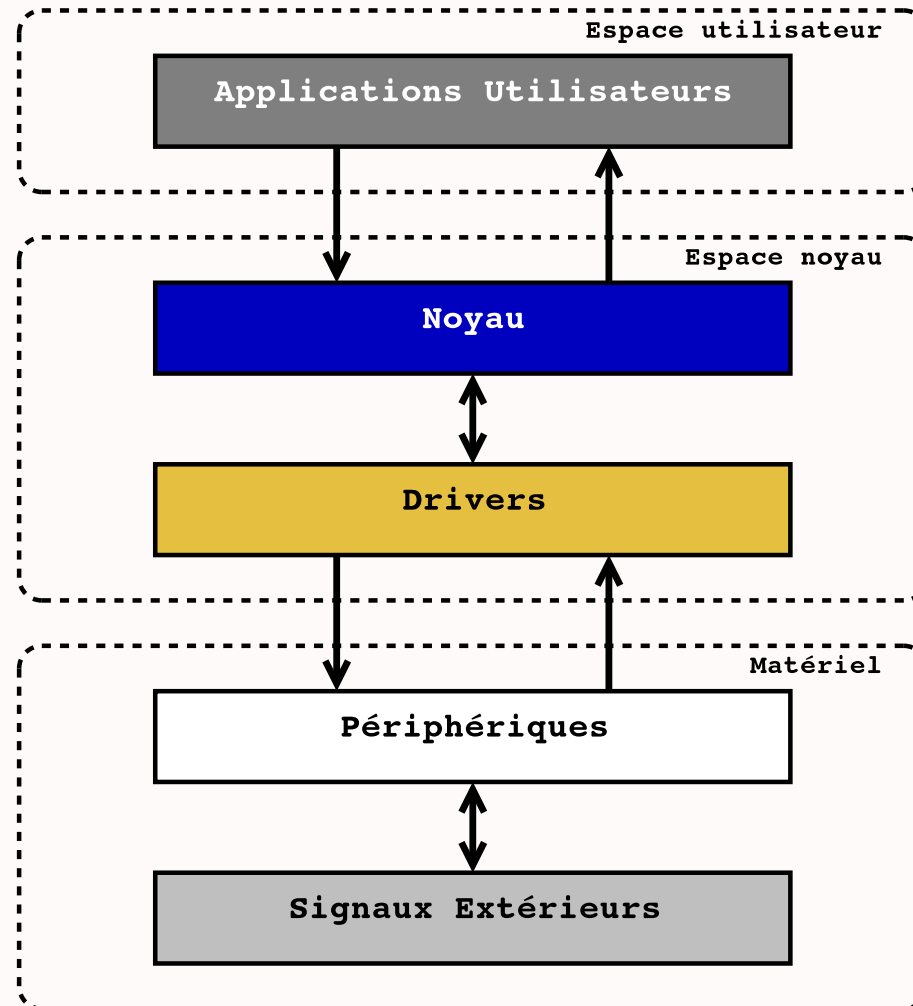
¹CAN : *Controller Area Network*

²CPL : *Courant Porteur en Ligne*

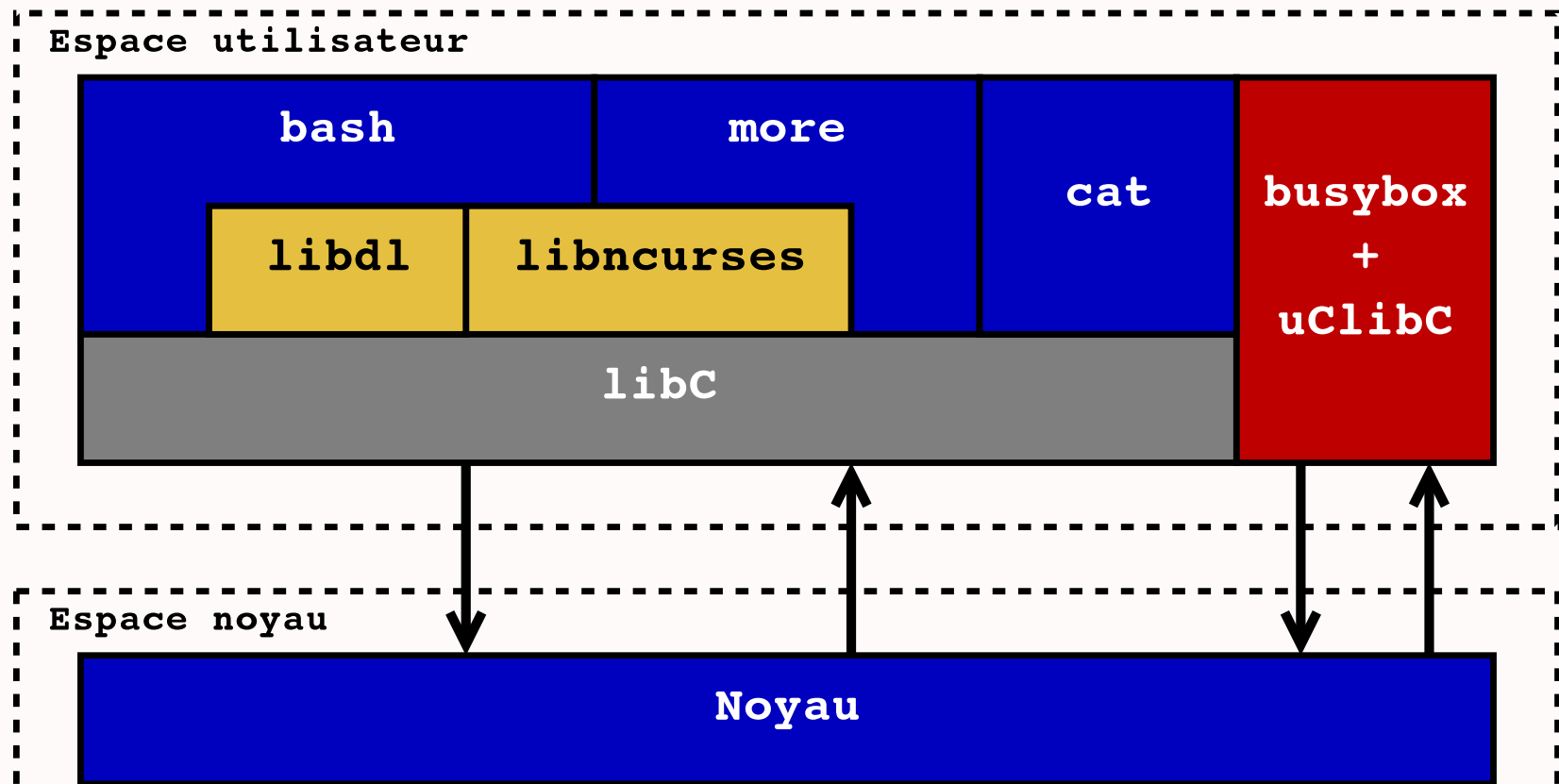
³ARCnet : *Attached Ressource Computer network*

Architecture logicielle

Rappels sur la structure d'un OS



Structure de l'espace utilisateur



Pourquoi GNU/Linux ?

Les raisons technologiques

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

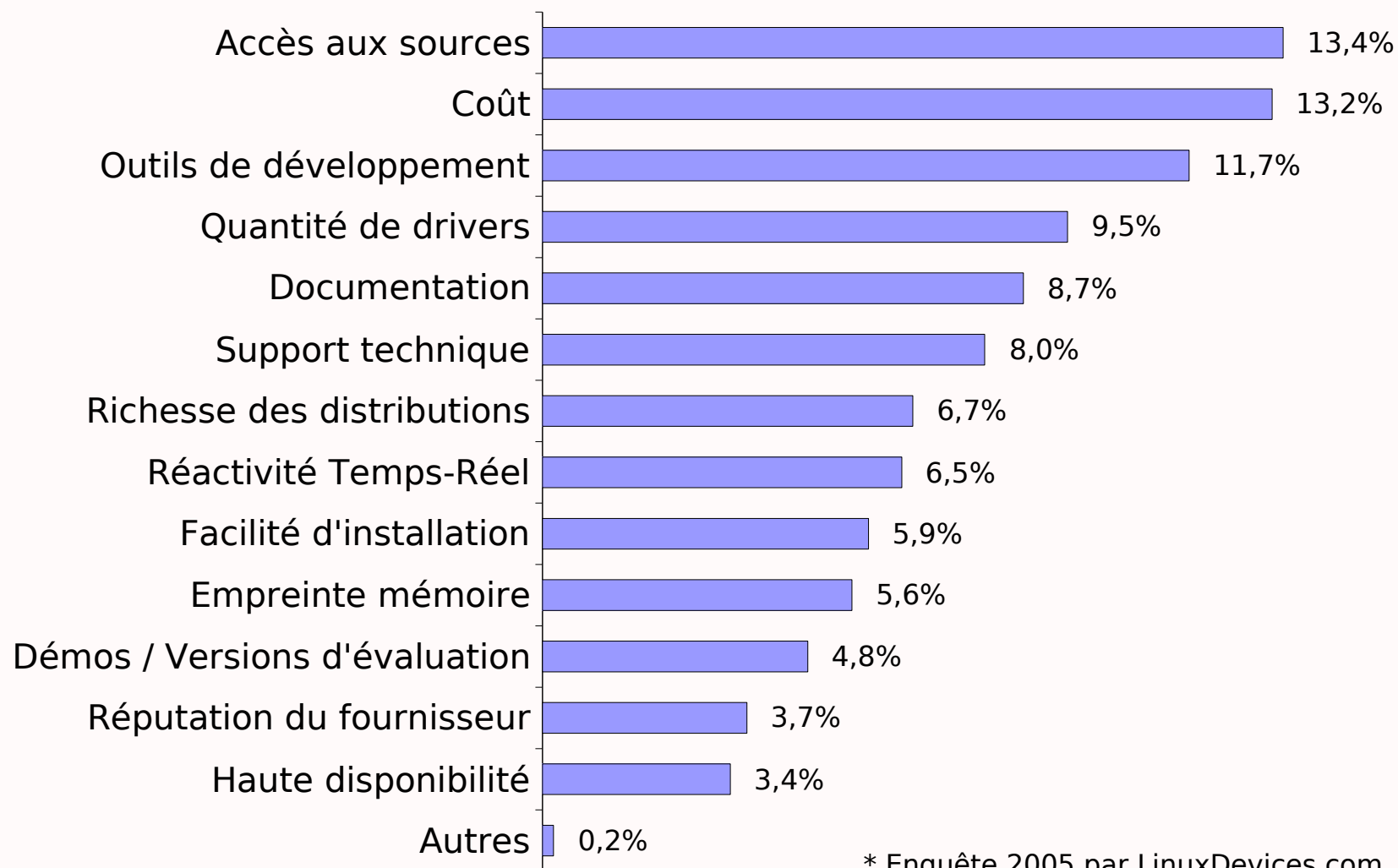
Les raisons économiques

- ✓ **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 philosophiques

- ✓ 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 au plus grand projet communautaire de tous les temps ;
- ✓ Contribution à une oeuvre commune.

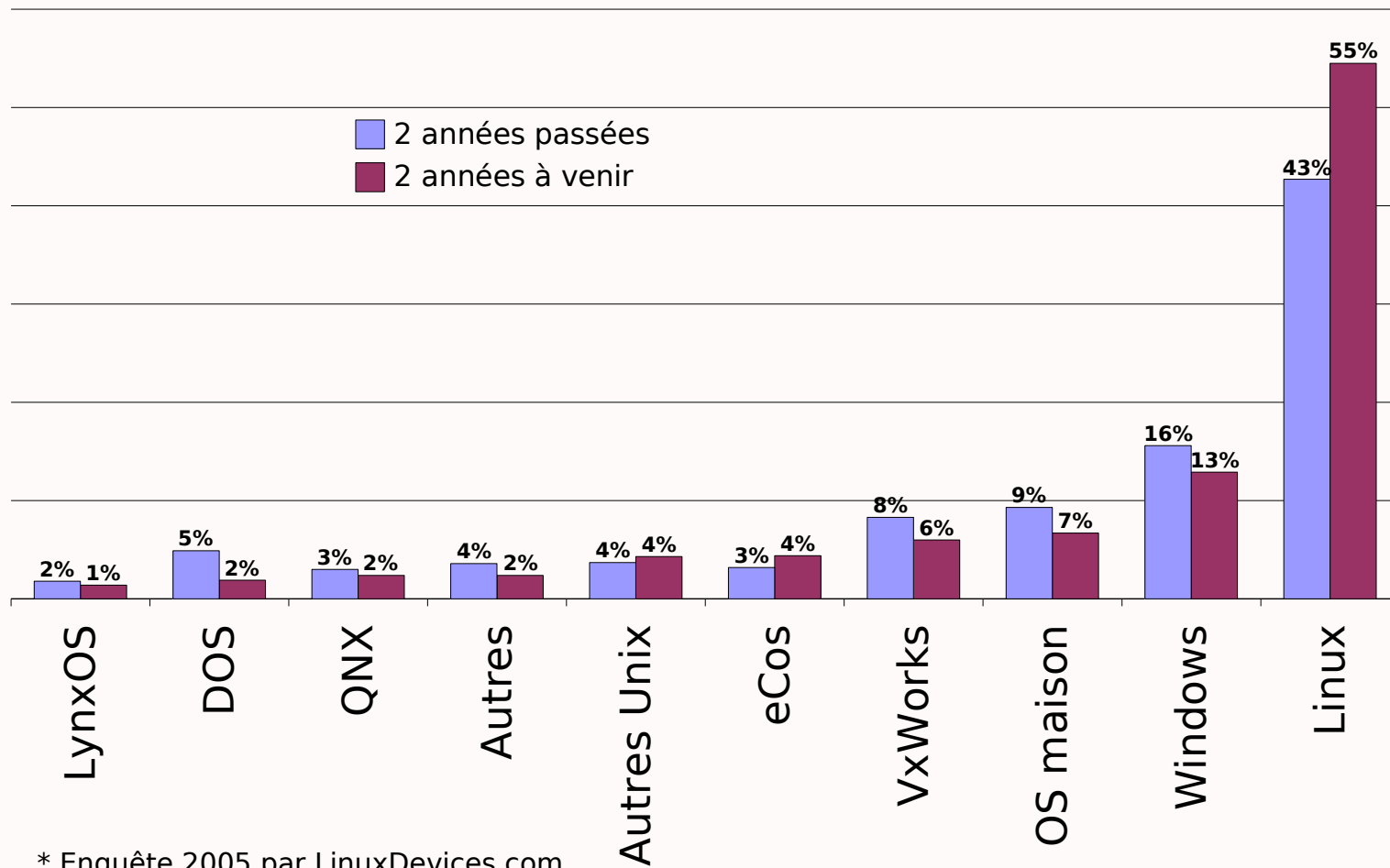
Principaux critères de choix pour une distribution embarquée*



* Enquête 2005 par LinuxDevices.com

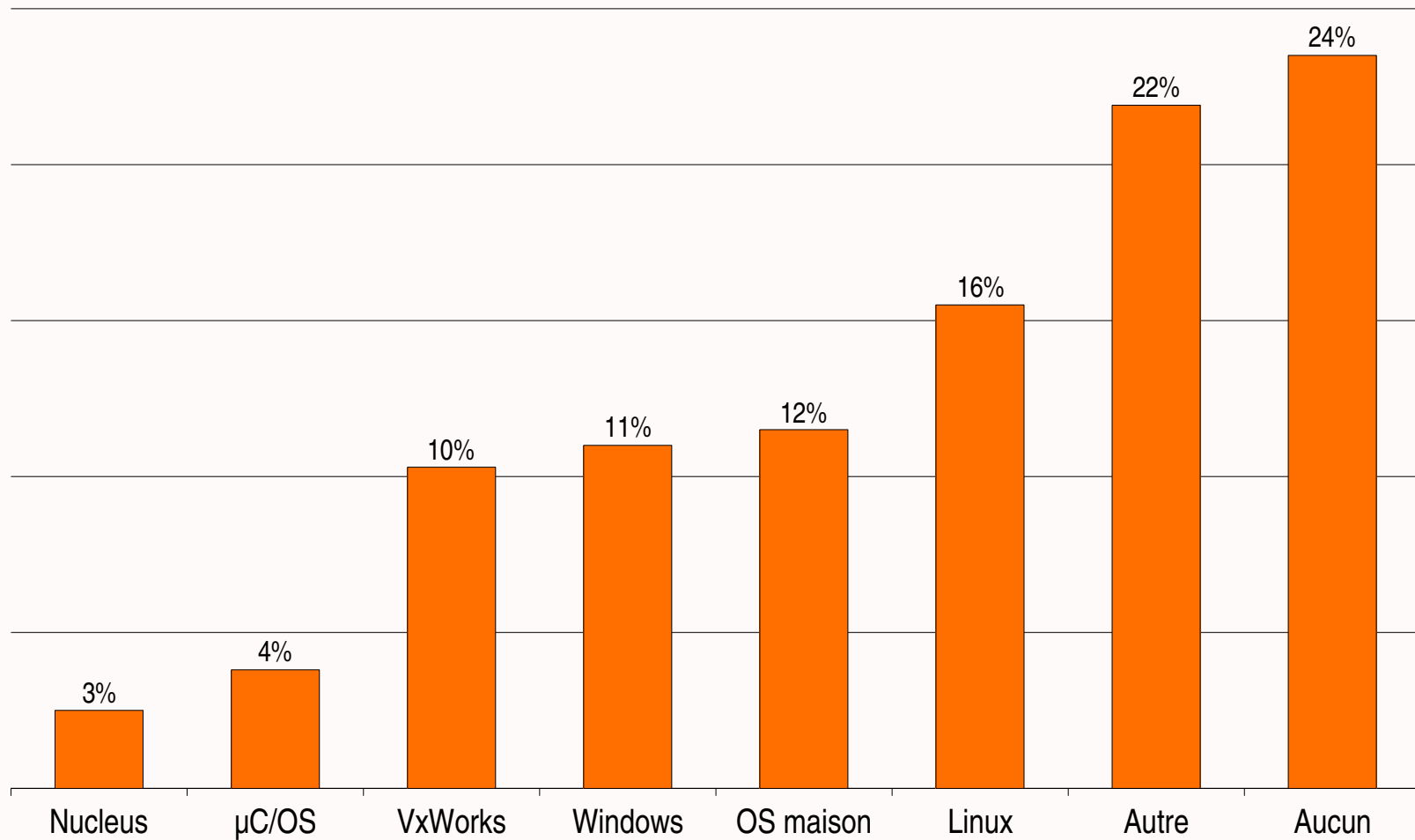
Les autres OS

Les OS utilisés dans les systèmes embarqués*



* Enquête 2005 par LinuxDevices.com

Les OS utilisés dans les systèmes embarqués*



* Enquête 2004 par VDC

Les licences

- GNU GPL¹ ⇒ la plus répandue, la plus stricte, la plus éprouvée et la plus contagieuse, elle est basée sur la notion de *copyleft*² ;
- GNU LGPL³ ⇒ autorise l'édition de liens dynamique avec du code non libre, très utilisée pour les bibliothèques ;
- X11/MIT/BSD ⇒ très permissives, autorisent l'exploitation propriétaire du code ;
- *PL ⇒ 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'OSI⁴ certifie leur conformité avec l'OSD⁵ (<http://www.opensource.org/>).

¹GPL : *General Public License*

²*copyleft* : utilisation des droits d'auteur / *copyright* pour assurer la liberté (utiliser, étudier, redistribuer et améliorer) du logiciel

³LGPL : *Lesser General Public License*

⁴OSI : *Open Source Initiative*

⁵OSD : *Open Source Definition*

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 (RTFM¹) ;
- ✗ Réticences des développeurs face aux changements (principe d'inertie face à la nouveauté et à la liberté de choix) ;
- ✗ Diversité des acteurs.

¹RTFM : *Read The F***ing Manual*

Solutions

Trois types de solutions 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 :

- **Systeme "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.

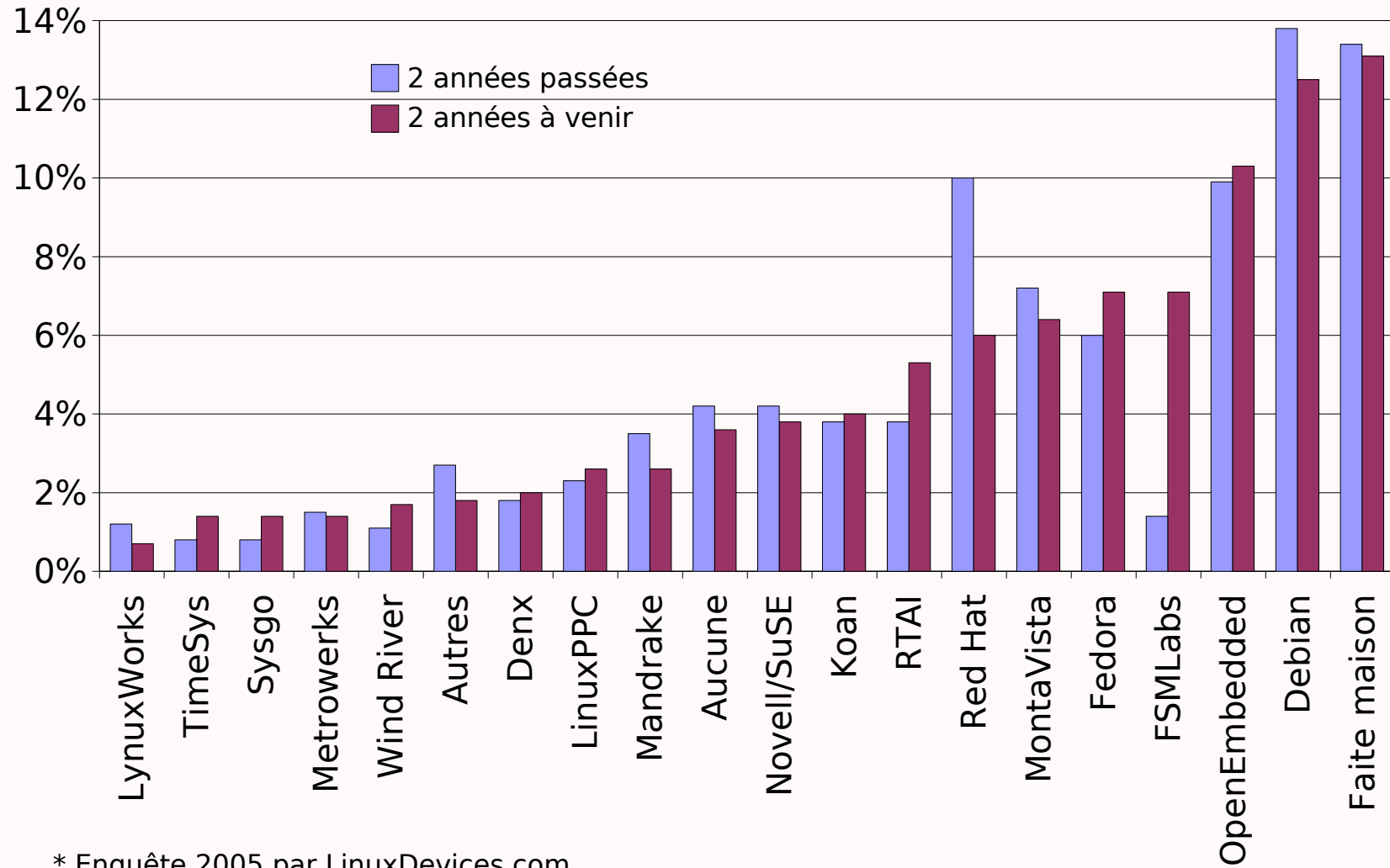
- **Distribution libre :**

- ✓ 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.

Fournisseurs/Distributions Linux préférés dans l'embarqué*



* Enquête 2005 par LinuxDevices.com

Briques de base logicielles

Noyaux Linux

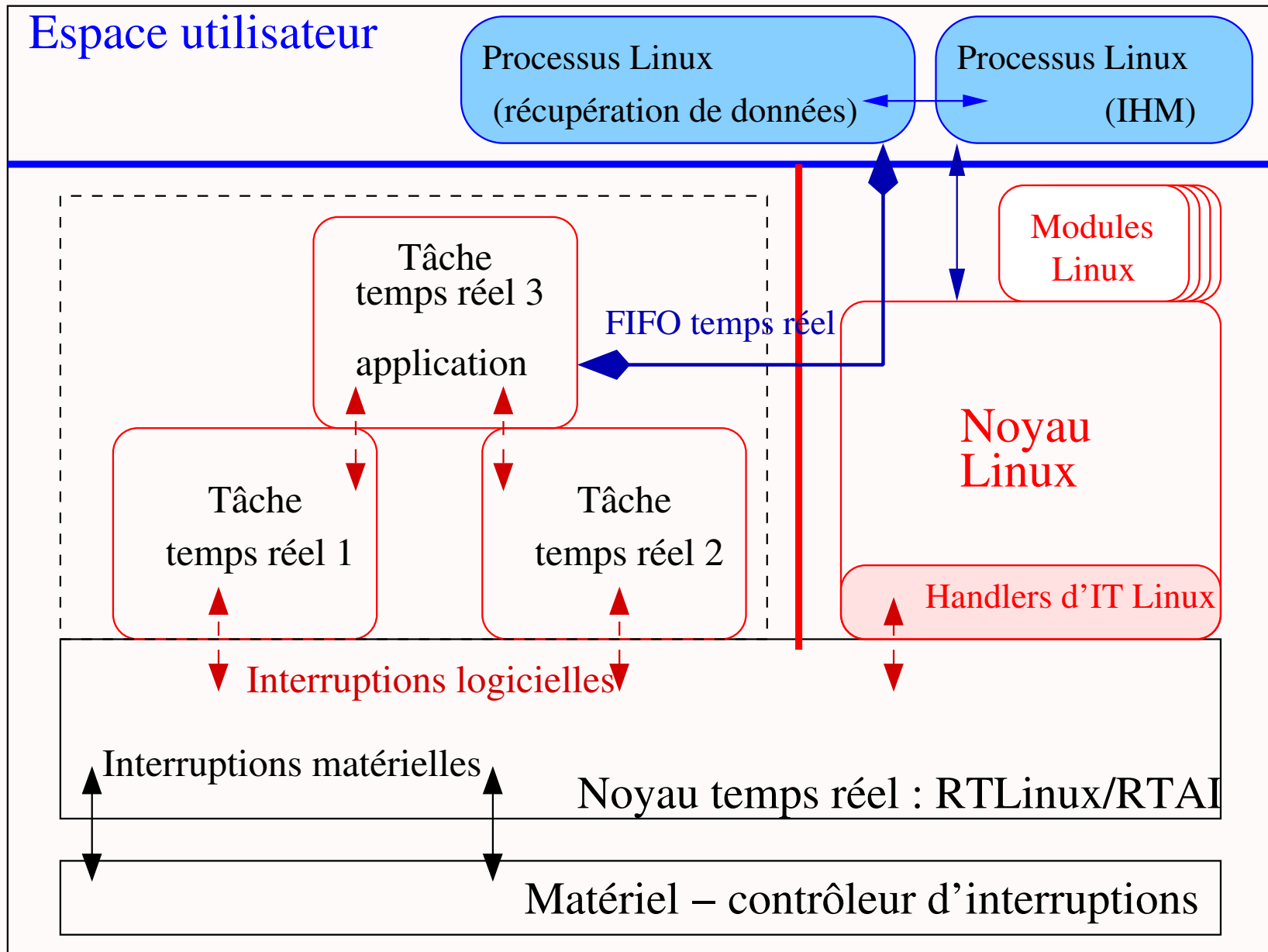
- **Linux vanilla** (<http://www.kernel.org/>) \Rightarrow le noyau Linux standard ;
- **μ Clinux** (<http://www.uclinux.org/>) \Rightarrow *patch* du noyau Linux pour supporter les architectures sans MMU¹ ainsi que quelques matériels périphériques spécifiques à ces architectures (intégré à partir des noyaux 2.6).

¹MMU : *Memory Management Unit*

Extensions temps-réel

- **RTAI¹/Linux** (<http://www.aero.polimi.it/~rtai/>) \Rightarrow micro-noyau temps-réel (LGPL et basé sur l'approche sans brevet ADEOS) permettant la cohabitation de tâches temps-réel dures avec un noyau Linux comme tâche de plus basse priorité ;
- **RTLinuxFree/RTLinuxPRO** (<http://www.fsmlabs.com/>) \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 ;
- *Patches low latency, O(1) scheduler, preemptible 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...).

¹RTAI : *RealTime Application Interface*



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

Systemes de fichiers pour l'embarqué

- **JFFS¹** (<http://developer.axis.com/software/jffs/>) ⇒ système de fichiers 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 MTD² de Linux ;
- **JFFS2** (<http://sources.redhat.com/jffs2/>) ⇒ successeur de JFFS ajoutant à celui-ci le support de la compression et quelques améliorations ;
- **ROMFS** (<http://romfs.sf.net/>) ⇒ système 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...);

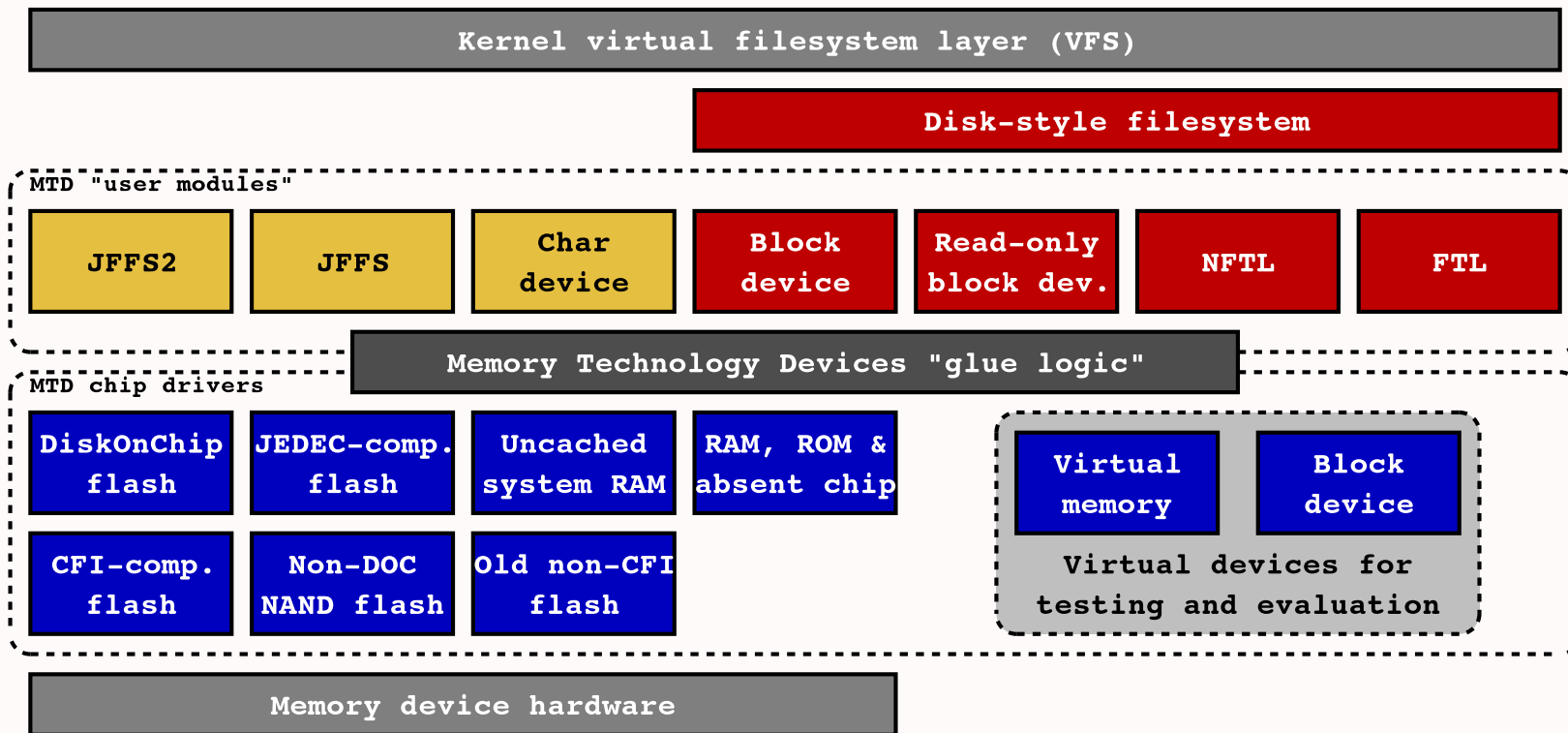
¹JFFS : *JournalingFlash File Système*

²MTD : *Memory Technology Devices*

- **CRAMFS**¹ (<http://sf.net/projects/cramfs/>) / **SquashFS** (<http://squashfs.sourceforge.net/>) ⇒ même but que ROMFS mais avec plus d'optimisations et compression des fichiers (zlib) ;
- **UnionFS** (<http://www.unionfs.org/>) / **mini fo** (<http://www.denx.de/>) ⇒ *Overlay FileSystem* rendant un système de fichiers en lecture seule virtuellement inscriptible par union logique avec un système réellement inscriptible ;
- **TMPFS/RAMFS** ⇒ systèmes de fichiers résidant en RAM, *swappable* (TMPFS - requiert une MMU) et redimensionnables dynamiquement, ils sont souvent utilisés pour stocker les données non conservées après redémarrage (/tmp, /var, logs...) ;
- **PRAMFS**² (<http://pramfs.sourceforge.net/>) ⇒ 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 souvent ce type de mémoires.

¹CRAMFS : *Compressed ROM FileSystem*

²PRAMFS : *Protected and Persistent RAM FileSystem*



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

Systemes de fichiers conventionnels

- **Ext2** (<http://e2fsprogs.sf.net/ext2.html>) \Rightarrow le systeme de fichiers par defaut de Linux qui peut etre monte avec ecritures synchrones (`mount -o sync`) pour assurer l'integrite des donnees aux depens des performances ;
- Systemes journalises (**Ext3, ReiserFS, XFS, JFS...**) \Rightarrow les operations sont decrites dans un journal (transactions) avant d'etre effectivement effectuees permettant ainsi de conserver l'integrite du systeme de fichiers et de redemarrer rapidement en cas de coupure de courant ;
- **MS-DOS FAT 12/16/32** \Rightarrow le systeme de fichiers originel des OS Microsoft, ses differentes declinaisons permettent de l'adapter aux differentes tailles de supports (tres utilise sur les stockages Flash des appareils electroniques grand public).

Bibliothèque C (libc)

- **Glibc** (<http://www.gnu.org/software/libc/libc.html>) \Rightarrow la bibliothèque C officielle de tous les systèmes Linux, elle est complète, performante, multi plate-forme et très bien documentée mais est néanmoins volumineuse et peu adaptée dans les systèmes à faible empreinte mémoire ;
- **μ Clibc** (<http://www.uclibc.org/>) \Rightarrow presque entièrement compatible 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 μ Clinux (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).

Outils de base

- **BusyBox** (<http://www.busybox.net/>) ⇒ 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/>) ⇒ complément parfait de BusyBox 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/>) ⇒ ensemble d'outils Unix courants optimisés pour la taille et basés sur la diet libc;
- **ASMutils** (<http://linuxassembly.org/asmutils.html>) ⇒ nombreux utilitaires Unix réécrits en assembleur x86;
- **Outils GNU** (<http://www.gnu.org/directory/GNU/>) ⇒ 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/>),
 - **dhcpcd...**
- Autres :
 - **Zebra** (<http://www.zebra.org/>),
 - **snmpd...**

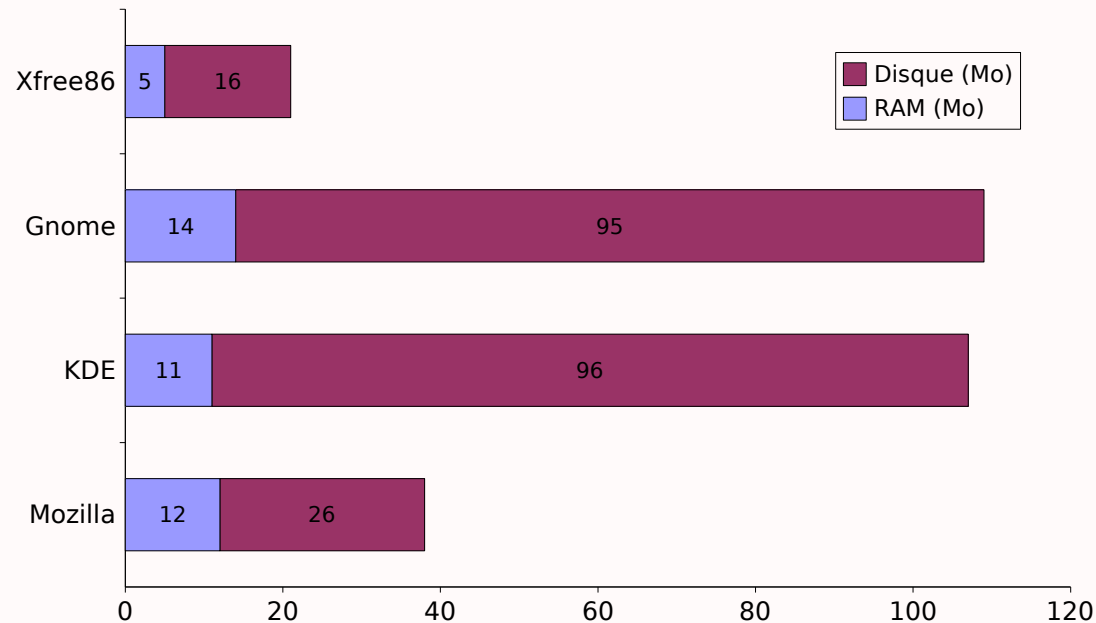
Bases de données

- **Berkeley DB** (<http://www.sleepycat.com/>) ⇒ *open source*, compatible GPL, commerciale, non relationnelle, simple, moins de 500 ko, multi plate-forme ;
- **MySQL** (<http://www.mysql.com/>) ⇒ *open source* (GPL ou commerciale), relationnelle, SQL, transactionnelle, rapide, multi plate-forme, très répandue ;
- **DB2 Everyplace** (<http://www.ibm.com/software/data/db2/everyplace/>) ⇒ commerciale, multi plate-forme, moins de 200 ko.

IHM¹

- 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.

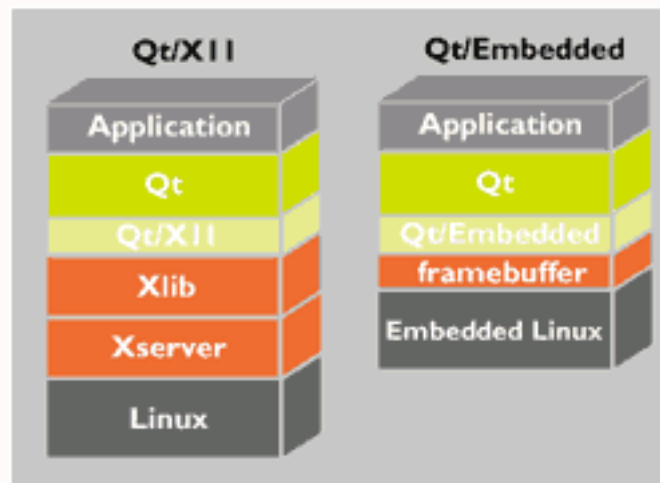
**Inadaptabilité des applications graphiques
"classiques" pour l'embarqué***



* Source LinuxDevices.com

¹IHM : *Interface Home Machine*

- **DirectFB** (<http://www.directfb.org/>) ⇒ surcouche au *framebuffer* Linux permettant de construire des IHM rapides et légères avec une abstraction complète du matériel ;
- **Gtk+** (<http://www.gtk.org/>) ⇒ 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/>) ⇒ couche permettant l'utilisation de la bibliothèque Qt au dessus du *framebuffer* Linux (de 800 ko à 3 Mo) ;



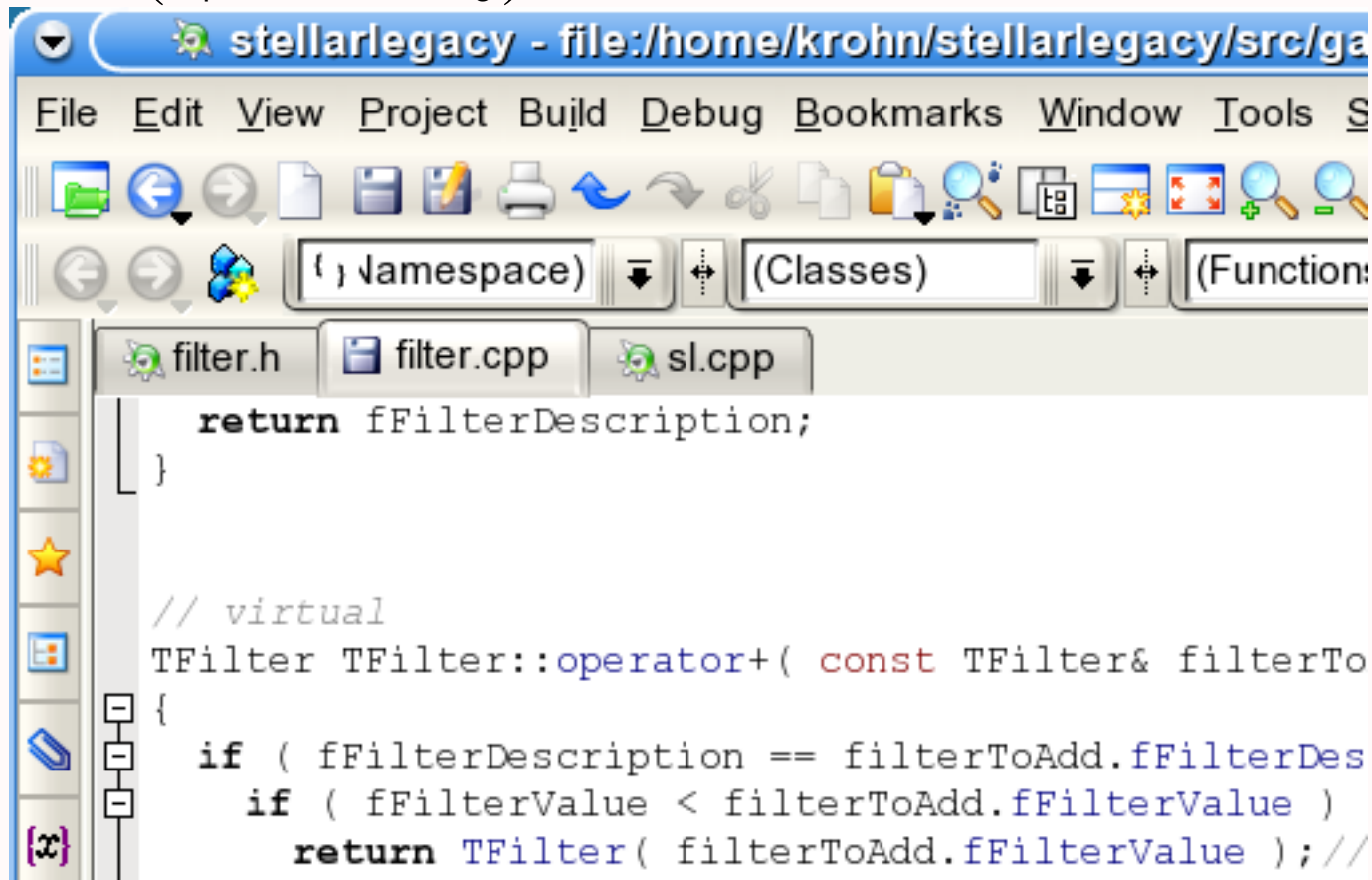
- **MicroWindows/Nano-X** (<http://www.microwindows.org/>)
⇒ environnement graphique multi plate-forme complet possédant sa propre API¹ mais également une bibliothèque de compatibilité X Window (100 ko) ;
- **X Window System (XFree86/X.org)** ⇒ (<http://www.xfree86.org/> / <http://www.x.org/>) les implémentations 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/>) ⇒ réimplémentation d'un serveur X Window pour plates-formes embarquées (1 Mo) ;
- **SDL**² (<http://www.libsdl.org/>) ⇒ bibliothèque multi plate-forme pour le développement d'applications graphiques multimédias.

¹API : *Application Programming Interface*

²SDL : *Simple DirectMedia Layer*

IDE¹

- Eclipse (<http://www.eclipse.org/>) ;
- KDevelop (<http://www.kdevelop.org/>) ;
- VIM (<http://www.vim.org/>) ;
- EMACS (<http://www.emacs.org/>).



¹IDE : *Intergated Development Environment*

Distributions libres

- μ Clinux-dist (<http://www.uclinux.org/pub/uClinux/dist/>) ;
- SnapGear Embedded Linux (<http://www.snapgear.org/>) ;
- OpenEmbedded (<http://oe.handhelds.org/>) ;
- Pengutronix PTXdist (http://www.pengutronix.de/software/ptxdist_en.html) ;
- Denx ELDK¹ (<http://www.denx.de/ELDK/>).



μ Clinux

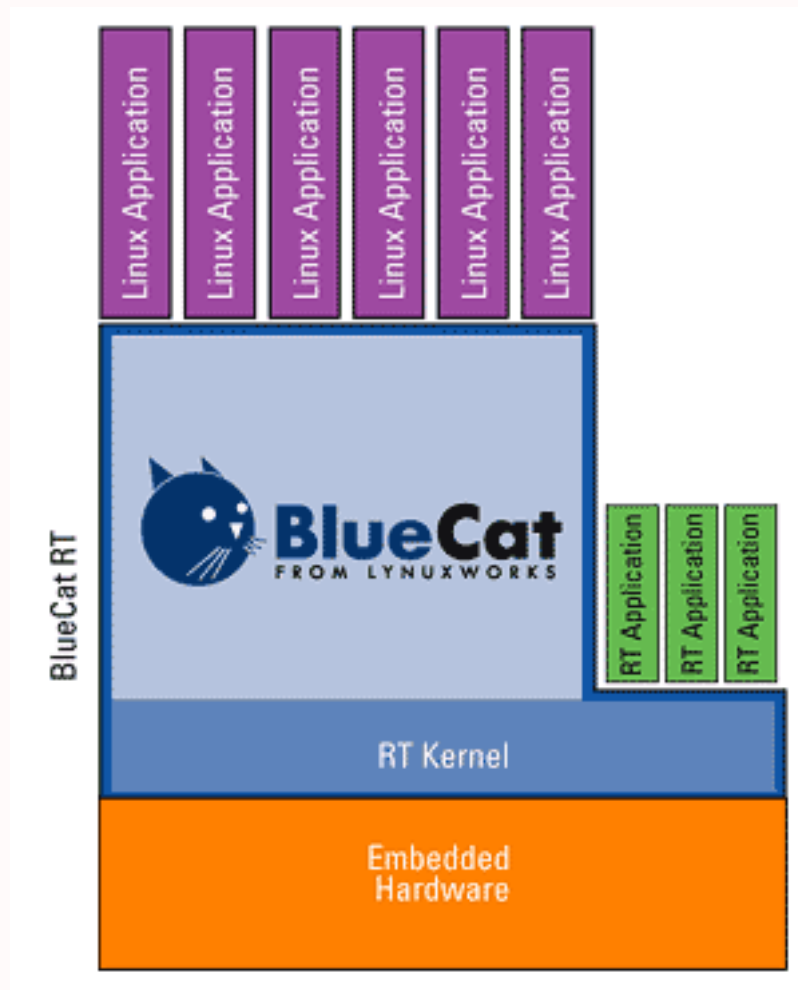
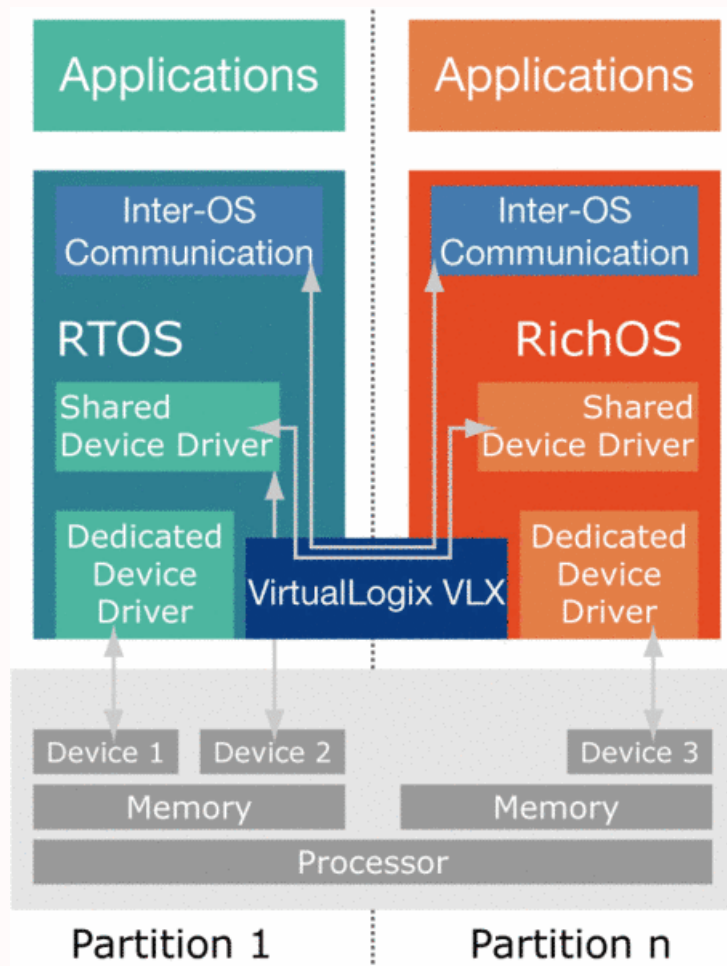


¹ELDK : *Embedded Linux Development Kit*

Distributions commerciales

- FSMLabs (<http://www.rtlinux.com/>) :
- VirtualLogix (<http://www.virtuallogix.com/>) :
- Koan (<http://www.klinux.org/>) :
- LynuxWorks (<http://www.lynuxworks.com/>) :
- Metrowerks/Lineo (<http://www.metrowerks.com/>) ;
- MontaVista (<http://www.mvista.com/>) ;
- SysGo (<http://www.elinos.com/>) ;
- TimeSys (<http://www.timesys.com/>).

The logo for ELinOS SysGo, featuring the text "ELinOS" in a blue serif font above "SYSGO" in a blue sans-serif font.The logo for LynuxWorks, featuring the text "LYNEXWORKS™" in a blue sans-serif font next to a yellow and blue circular graphic.The logo for TimeSys, featuring a blue circular graphic with a white swoosh and the text "TimeSys®" in a blue sans-serif font.The logo for MontaVista, featuring a grey arc above the text "montavista™" in a blue sans-serif font.The logo for Wind River, featuring the text "WIND RIVER" in a red sans-serif font.The logo for VirtualLogix, featuring the text "virtualLogix" in a blue sans-serif font.The logo for FSMLabs RTLinux, featuring a blue graphic of three slanted lines above the text "FSMLabs" in a blue sans-serif font and "RTLinux®" in a smaller blue sans-serif font below it.The logo for Klinux, featuring the text "K linux" in a red sans-serif font with a green square above the "K".The logo for Metrowerks, featuring the text "metrowerks™" in a white sans-serif font on a black background.



Notions essentielles

Analyse du processus de démarrage de Linux

- *Firmware (bootstrap)* ⇒ 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* ⇒ chargé de lancer le noyau en le plaçant en RAM ou en l'exécutant sur place (XIP¹) 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/TFTP²).

¹XIP : *eXecute In Place*

²TFTP : *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 (NFS¹).avant de lancer finalement le premier processus (*init*);
- Le processus *init* lance les applicatifs et autres services du système...

¹NFS : *Network FileSystem*

Processus de compilation

```
gcc -v helloworld.c -o helloworld
```

- **Préprocesseur** (CPP¹) ⇒ traite les macro-commandes des fichiers C (`#include`, `#define`, `#ifdef`, `__FUNCTION__`...);
- **Compilateur** (CC²) ⇒ transforme les fichiers sources C en fichiers sources d'assemblage dédiés à une plate-forme ;
- **Assembleur** (AS³) ⇒ transforme les fichiers sources d'assemblage en objets binaires (bibliothèque BFD⁴) ;
- **Édition de liens** (LD⁵) ⇒ fabrique un exécutable à partir des objets binaires et des bibliothèques statiques (archives).

¹CPP : *C PreProcessor*

²CC : *C Compiler*

³AS : *ASsembler*

⁴BFD : *Binary File Descriptor*

⁵LD : *Link eDitor*

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
```

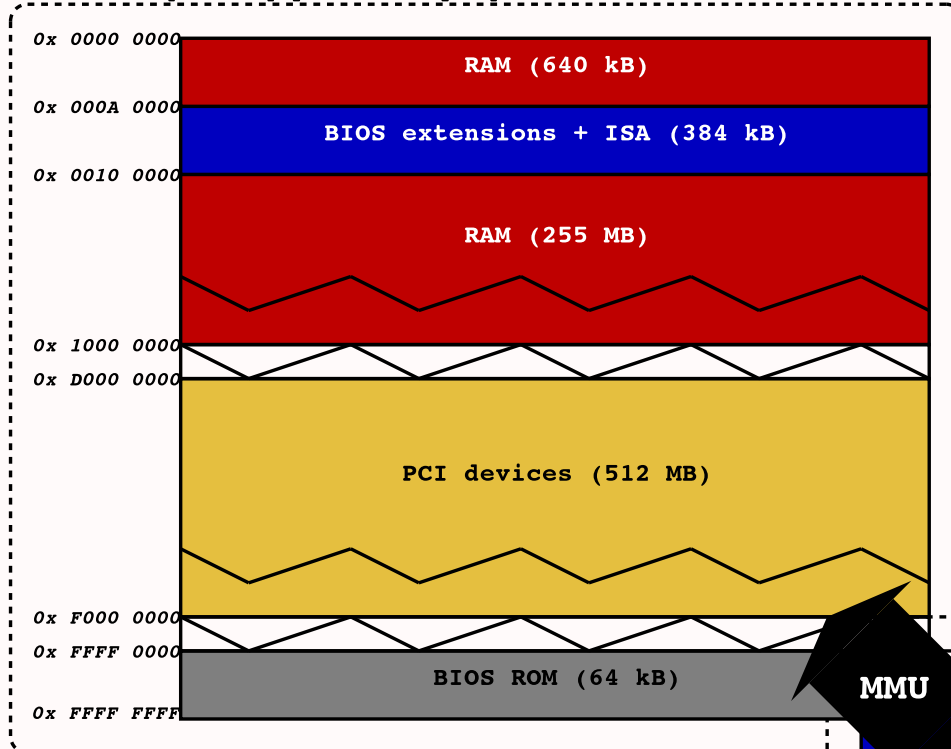
```
gcc -L. -o app_dynamic app.o -lmisc
```

- 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.

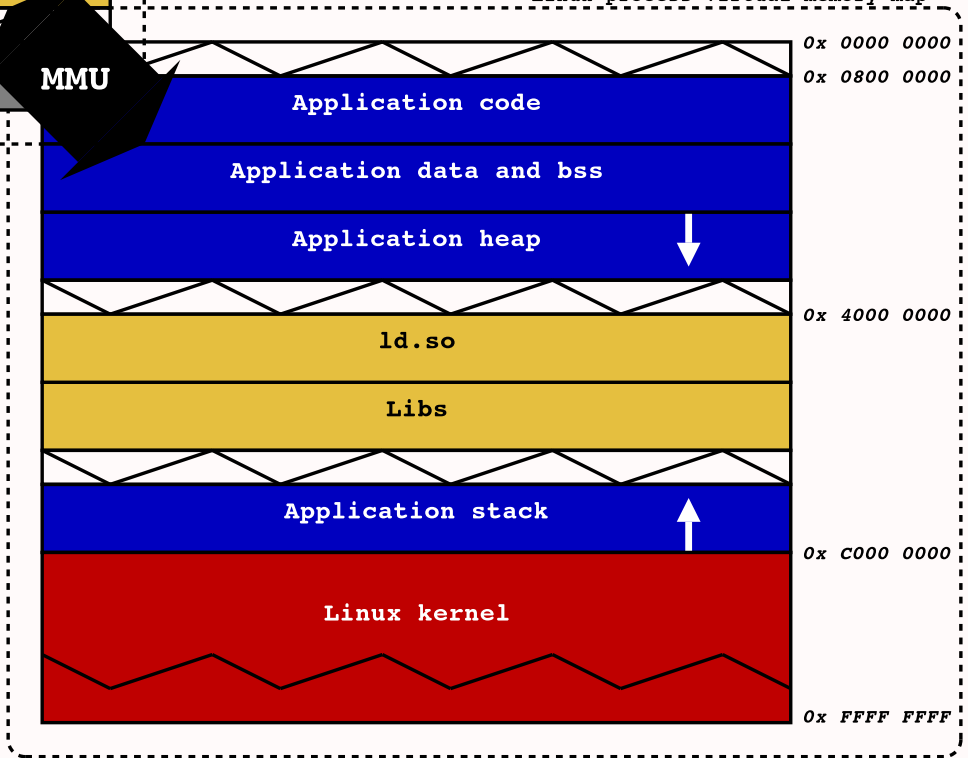
Processus de chargement d'un exécutable lié dynamiquement

- 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



Linux process virtual memory map



Les outils associés aux bibliothèques partagées

- `ldd` \Rightarrow affiche les bibliothèques partagées dont dépend un exécutable lié dynamiquement ou une autre bibliothèque partagée ;
- `ldconfig` \Rightarrow 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` \Rightarrow 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

- ELF¹ ⇒ format binaire pour les exécutables, les objets et les bibliothèques, il fait office de standard pour la plupart des Unix (dont Linux) ;
- a.out² ⇒ le format de sortie par défaut de l'assembleur et de l'éditeur de liens des systèmes Unix ;
- bFLT³ (Flat) ⇒ format de fichiers exécutables léger, dérivé du format a.out et utilisé par le projet μ Clinux, il supporte la compression ;
- COFF⁴ ⇒ format binaire objet issu de l'ABI⁵ d'Unix System V, il est l'ancêtre du format ELF.

¹ELF : *Executable and Linkable Format*

²a.out : *assembler output*

³bFLT : *binary FLAT format*

⁴COFF : *Common Object File Format*

⁵ABI : *Application Binary Interface*

Manipulations sur les exécutable

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

μ Clinux vs Linux

Différences fondamentales

- μ Clinux 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 \Rightarrow 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()`) \Rightarrow préférer l'allocation d'un *pool* au démarrage de l'application ;

- Pile de taille fixée à la compilation ;
- Utilisation d'exécutables relogeables :
 - adressage relatif (PIC¹) ⇒ code limité à 32 ko (jump 16 bits du 68k), ou
 - adressage absolu complètement relogeable (références modifiées au chargement par le noyau) ⇒ plus lourd et plus lent au chargement.
- Pas de mécanisme de *swap*.

¹PIC : *Position Independent Code*

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, 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 programmeur 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

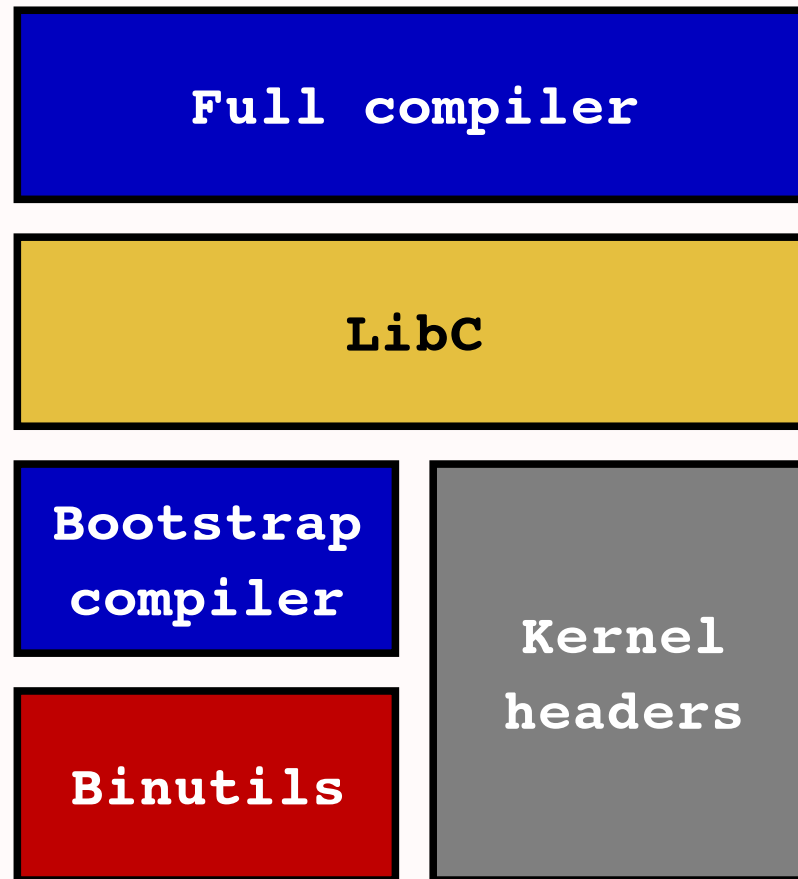
- Le développement du ou des applicatifs est réalisé sur cible ou sur une plate-forme de développement (d'architecture similaire à la cible) à partir d'une distribution Linux classique sur disque dur ;
- Développement (en parallèle ?) d'un système de fichiers racine léger qui :
 - contient uniquement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

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) ;
- Designation de la cible au format GNU :
 - ARM \Rightarrow `arm-linux`
 - PowerPC \Rightarrow `powerpc-linux`
 - MIPS (big endian) \Rightarrow `mips-linux`
 - MIPS (little endian) \Rightarrow `mipsel-linux`
 - i386 \Rightarrow `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

- <http://www.scratchbox.org>;
- Boite à outils simplifiant la mise en place de chaînes complètes de compilation croisée ;
- Gère à la fois la compilation croisée et la configuration croisée ;
- Utilise un mécanisme de *sandbox* (UML + chroot) pour isoler la cible de l'hôte ;
- Supporte les cibles de type ARM, x86 et PowerPC à partir d'un hôte x86 uniquement.

Débogage et optimisation

Débogage distant avec GDB¹

- Débogage symbolique ;
- Processus de la cible contrôlé à distance depuis l'hôte avec GDB (ou surcouche graphique comme DDD²) ;
- 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).

¹GDB : *GNU DeBugger*

²DDD : *Data Display Debugger*

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.

LTT¹

- Logiciel d'analyse système complet ;
- Comprend 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'une application graphique dédiée ;
- Permet l'analyse complète des problématiques de synchronisation, communication inter-processus, *timing* utilisateur et noyau...

¹LTT : *Linux Trace Toolkit*

Profiling

- Technique consistant à réaliser des statistiques sur les temps d'exécution des différentes parties 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 instantiation 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) ;
- ICE¹ \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 ;

¹ICE : *In Circuit Emulator*

- JTAG¹ ⇒ sur un microcontrôleur il est utilisé en OCD² et permet, via un lien série, 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 ;
- BDM³ ⇒ 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.

¹JTAG : *Joint Test Action Group*

²OCD : *On Chip Debugger*

³BDM : *Background Debug Monitor*

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 μ Clinux ;
- POSE¹ (<http://sf.net/projects/pose/>) \Rightarrow émulateur multi plate-forme de PDA Palm, il est une amélioration de Copilot apportée par Palm Software ;

¹POSE : *Palm OS Emulator*

- UML¹ (<http://user-mode-linux.sf.net/>) ⇒ 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/>) ⇒ machine virtuelle multi plate-forme commerciale 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/>) ⇒ émulateur x86/PC multi plate-forme sous licence LGPL ;
- MAME² (<http://www.mame.net/>) ⇒ émulateur de machines arcades aussi diverses que variées, il émule des processeurs encore utilisés dans l'embarqué (z80, M6809, 68k...).

¹UML : *User Mode Linux*

²MAME : *Multiple Arcade Machine Emulator*

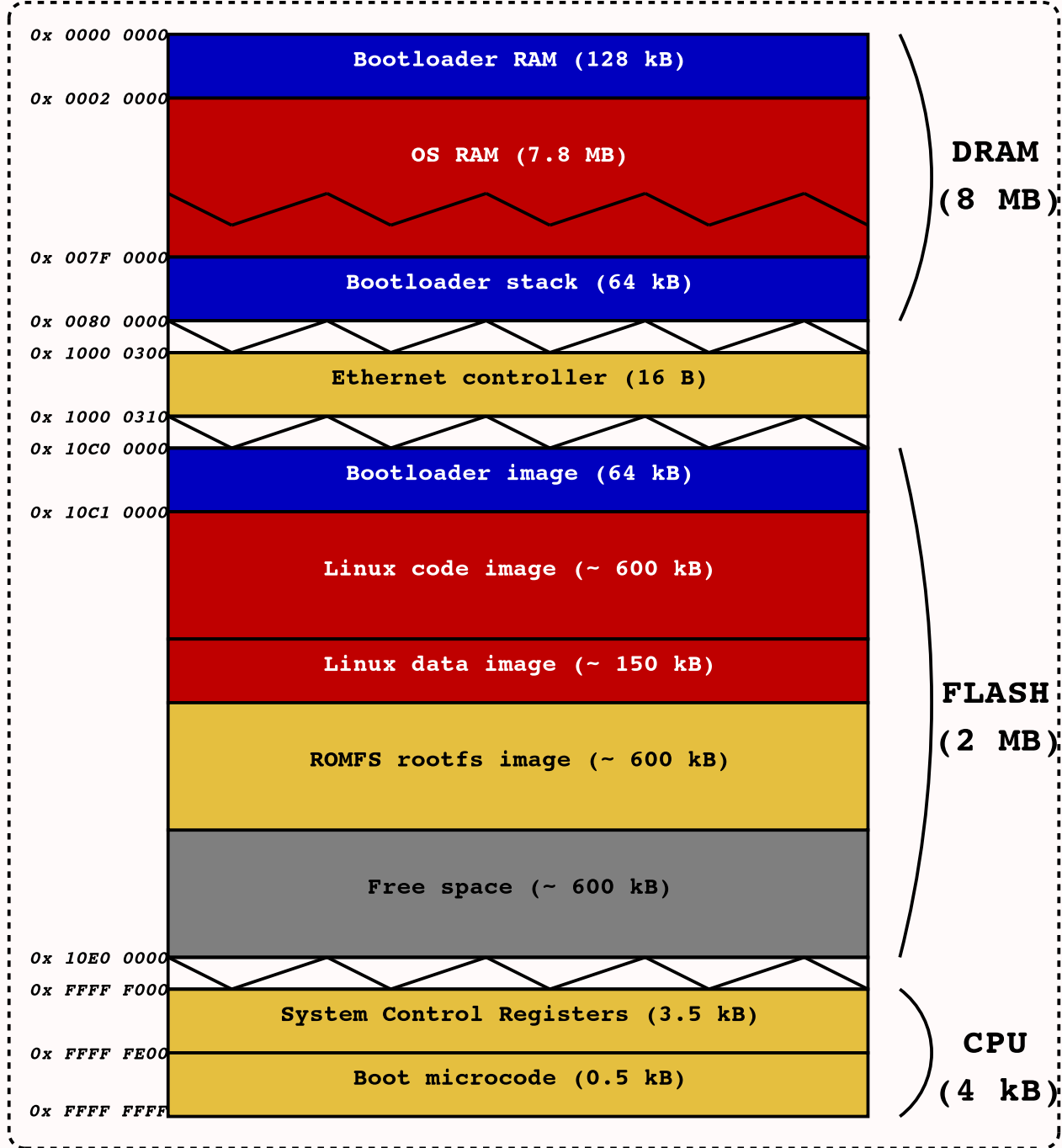
Étude de cas

μ Csimm

- Mise en oeuvre d'un système embarqué sur la plate-forme μ Csimm (DragonBall EZ, 2 Mo Flash, 8 Mo DRAM et contrôleur Ethernet 10 Mb) :
 - mise en place de la distribution μ Clinux,
 - développement croisé d'applicatifs,
 - débogage distant via gdbserver,
 - serveur web et CGI¹.

¹CGI : *Common Gateway Interface*

uCsim physical memory map



Références

Livres

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

Portails web

- LinuxDevices (<http://www.linuxdevices.com/>) ;
- RealTimeLinux (<http://www.realtimelinux.org/>) ;
- uCdot (<http://www.ucdot.org/>) ;
- HandHelds (<http://www.handhelds.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 μ Clinux 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/>) ;
- LinuxDevices Quick Reference Guides
(<http://www.linuxdevices.com/articles/...>) :
 - Embedded Processor and System-on-Chip (...AT4313418436.html),
 - Single Board Computer (...AT2614444132.html),
 - Embedded Linux "Cool Devices" (...AT4936596231.html).