

Introduction à Linux embarqué

Thomas Petazzoni
Bootlin
<https://bootlin.com/>



Intervenant

- ▶ Thomas Petazzoni
- ▶ Ingénieur Linux embarqué chez Bootlin
 - ▶ Spécialisé Linux embarqué : services de développement et formation
- ▶ Impliqué dans la communauté
 - ▶ Développeur Buildroot
 - ▶ Fondateur et membre CA de Toulibre
 - ▶ Membre CA de l'April
 - ▶ Créateur et animateur de l'Agenda du Libre
 - ▶ Co-auteur de MapOSMatic.org

Embedded Linux Conference Europe

- ▶ **LA** conference Linux embarqué pour les techniciens
- ▶ Jeudi et vendredi à Grenoble
- ▶ Des dizaines de conférences
- ▶ 80 € pour les hackers

Embarqué ?

Un système embarqué peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise. Ses ressources disponibles sont généralement limitées. Cette limitation est généralement d'ordre spatial (taille limitée) et énergétique (consommation restreinte).

Wikipédia, http://fr.wikipedia.org/wiki/Systeme_embarque

Embarqué ?

- ▶ Une définition assez générale
 - ▶ Recouvre des systèmes de types très différents
 - ▶ Frontière floue avec les systèmes « classiques »
- ▶ Produits de grande consommation
 - ▶ Routeurs personnels, lecteurs de DVD, appareils photos numériques, GPS, caméscopes, téléphones, micro-onde, four
- ▶ Produits industriels
 - ▶ Commande de robot, alarmes, systèmes de surveillance, contrôle de machines, voiture, avion, satellite

Linux embarqué

- ▶ Le monde du Logiciel Libre offre toute une palette d'outils pour le développement de systèmes embarqués
- ▶ Avantages
 - ▶ Réutilisation de composants existants pour le système de base, permet de se focaliser sur sa valeur ajoutée
 - ▶ Composants de bonne qualité, pour certains
 - ▶ Contrôle complet sur les composants, modifications sans contraintes
 - ▶ Support de la communauté: tutoriels, listes
 - ▶ Faible coût, et notamment pas de royalties par unité vendue
 - ▶ Potentiellement moins de problèmes juridiques
 - ▶ Accès plus facile aux logiciels et outils

Linux Embarqué

- ▶ Parts de marchés actuelles des OS embarqués
 - ▶ OS propriétaire: 39%
 - ▶ Linux embarqué gratuit: 29%
 - ▶ Linux embarqué avec support commercial: 11%
 - ▶ OS maison: 7%
 - ▶ Pas d'OS: 11%
- ▶ Pour les projets futurs
 - ▶ Linux embarqué gratuit: 71%
 - ▶ Linux embarqué avec support commercial: 16%
 - ▶ OS propriétaire: 12%
 - ▶ OS maison: 1%
- ▶ Source: Venture Development Corp, octobre 2007

Linux Embarqué

- ▶ GPS: Tomtom et Garmin
- ▶ Routeurs personnels: Linksys
- ▶ PDA: Zaurus, Nokia N8x0
- ▶ Téléviseurs, caméscopes, lecteurs de DVDs, etc: Sony
- ▶ Téléphone: Motorola, OpenMoko
- ▶ Machines industrielles
- ▶ Et plein d'autres usages que l'on imagine même pas...

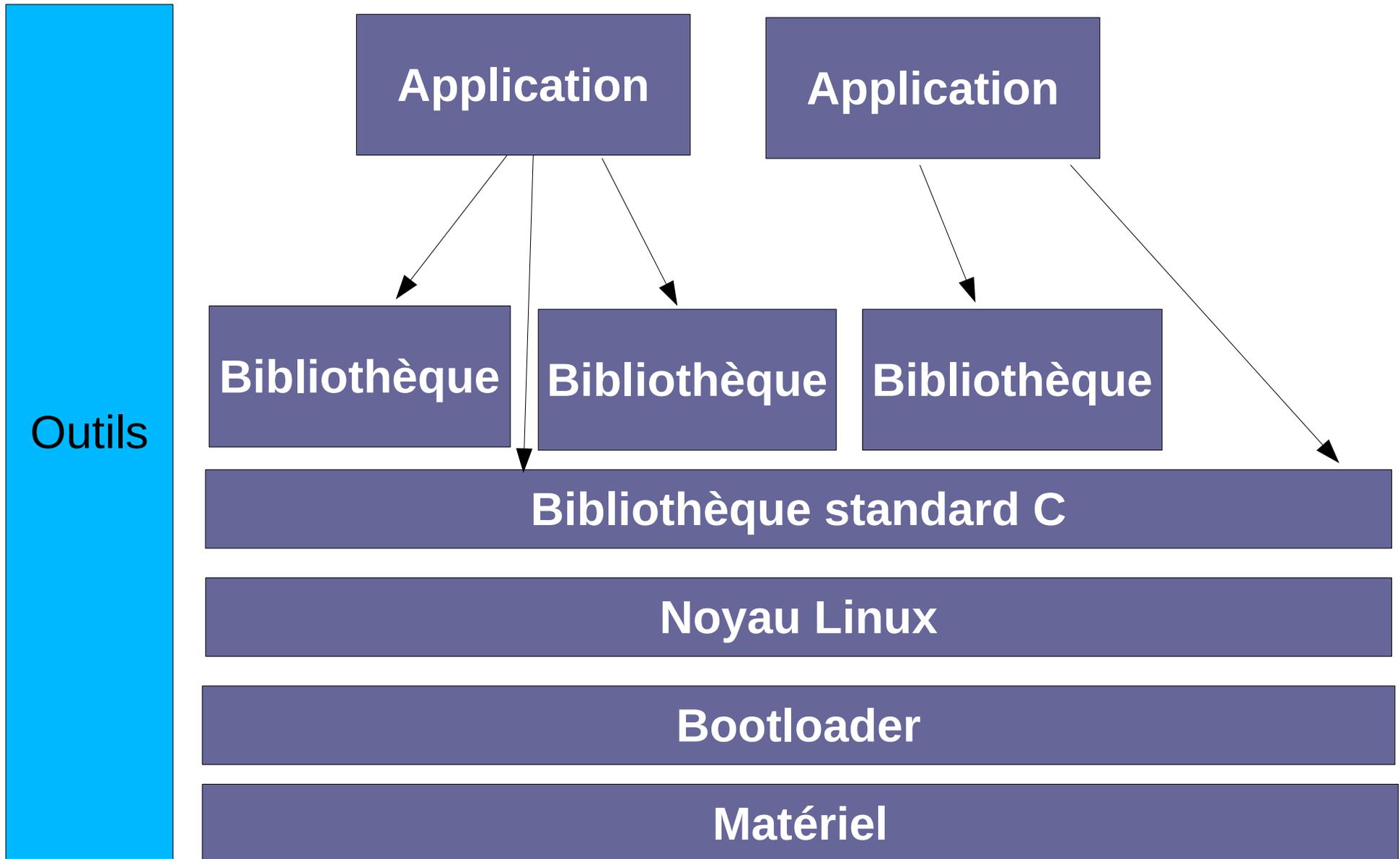


Ça fonctionne avec Linux, mais à quoi ça sert ?



À traire les vaches !

Architecture de base



Matériel pour l'embarqué

- ▶ Le matériel des systèmes embarqués est souvent différent de celui d'un système classique
 - ▶ Architecture processeur différente. Souvent ARM, MIPS ou PowerPC. x86 est aussi utilisé.
 - ▶ Stockage sur mémoire Flash, de type NOR ou NAND, de capacité souvent relativement réduite (quelques Mo à quelques centaines de Mo)
 - ▶ Capacité mémoire réduite (quelques Mo à quelques dizaines de Mo)
 - ▶ De nombreux bus d'interconnexion peu courants sur le desktop: I2C, SPI, SSP, CAN, etc.
- ▶ Cartes de développement à partir d'une centaine d'Euros
 - ▶ Souvent utilisées comme base pour le design final de la carte qui sera utilisée

Exemples

▶ Picotux 100

- ▶ ARM7 55 Mhz, Netsilicon NS7520
- ▶ 2 Mo de Flash
- ▶ 8 Mo de RAM
- ▶ Ethernet
- ▶ 5 GPIO
- ▶ Série



▶ OpenMoko

- ▶ ARM 920T 400 Mhz, Samsung 2442B
- ▶ 2 Mo de Flash NOR
- ▶ 128 Mo de RAM
- ▶ 256 Mo de Flash NAND
- ▶ Touchscreen 640x480, Bluetooth, GSM, série, GPS, son, deux boutons, Wifi, USB, etc.



Exemples

▶ Carte Calao Systems USB A 9263

- ▶ Processeur Atmel AT91SAM9263 à 180 Mhz
- ▶ 64 Mo de RAM
- ▶ 256 Mo de NAND
- ▶ Ethernet
- ▶ Série/JTAG/alim sur USB
- ▶ USB host et device
- ▶ Plein de cartes d'extension
- ▶ Fabricant sur Grenoble!
- ▶ ~ 150 €

▶ Carte Armadeus APF27

- ▶ Processeur Freescale iMX27
- ▶ 128 Mo de RAM
- ▶ 256 Mo de NAND
- ▶ Ethernet
- ▶ Série
- ▶ USB host
- ▶ FPGA !

Émulation

- ▶ Qemu permet l'émulation de nombreuses architectures CPU
 - ▶ X86, bien sûr, mais aussi PowerPC, ARM, MIPS, SPARC, etc.
 - ▶ Commande: `qemu - system - ARCH`
- ▶ Émulation complète: processeur, mémoire et périphériques
- ▶ Pour chaque architecture, plusieurs plateformes sont proposées
 - ▶ Pour ARM: Integrator, Versatile, PDA Sharp, Nokia N8x0, Gumstix, etc.
 - ▶ `qemu - system - arm -M ?`
- ▶ Idéal pour commencer à bricoler autour de Linux embarqué, même si le contact avec le matériel est plus « fun »
- ▶ Web: <http://bellard.org/qemu/>

Besoins minimaux

- ▶ Un processeur supporté par le compilateur gcc et le noyau Linux
 - ▶ Processeur 32 bit
 - ▶ Les processeurs sans MMU sont également supportés, au travers du projet uClinux
- ▶ Quelques mega-octets de RAM, à partir de 4 Mo, 8 Mo sont nécessaires pour pouvoir faire vraiment quelque chose.
- ▶ Quelques mega-octets de stockage, à partir de 2 Mo, 4 Mo pour faire vraiment quelque chose.
- ▶ Linux n'est donc pas fait pour les petits micro-contrôleurs qui ne possèdent que quelques dizaines ou centaines de Ko de Flash et de RAM
 - ▶ Sur le métal, pas d'OS
 - ▶ Systèmes réduits, type FreeRTOS, RTEMS, etc.

Chaîne de cross-compilation

- ▶ L'outil indispensable pour le développement embarqué sur des architectures non-x86.
- ▶ Contient des outils
 - ▶ S'exécutant sur une machine hôte (la machine du développeur, généralement x86)
 - ▶ Générant/manipulant du code pour machine cible (généralement non x86)
- ▶ « Outils binaires »
 - ▶ Binutils
 - ▶ Ld, as, nm, readelf, objdump, etc.
 - ▶ Bibliothèque standard C
 - ▶ glibc, uClibc ou eglibc
 - ▶ Compilateur C/C++
 - ▶ gcc
 - ▶ Bibliothèques mathématiques
 - ▶ gmp, mpfr
 - ▶ Débogueur
 - ▶ gdb

Chaîne de cross-compilation

▶ À la main

- ▶ Implique de configurer et compiler tous les éléments dans le bon ordre
- ▶ Les architectures non-x86 ne sont pas toujours parfaitement fonctionnelles dans une version stable de binutils ou gcc, besoin d'appliquer des patches

▶ Pré-compilées

- ▶ Solution la plus simple
- ▶ Code Sourcery est un fournisseur réputé, sous contrat avec ARM et MIPS

▶ Génération avec des scripts

- ▶ Crosstool-ng,
<http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool>
- ▶ Buildroot, <http://buildroot.uclibc.org>

Bootloader

- ▶ Sur PC : LILO ou Grub
 - ▶ Le BIOS fait une bonne partie du travail et met à disposition du bootloader des routines pour le chargement de données depuis le disque
- ▶ Sur les architectures embarquées, pas de BIOS
 - ▶ Le bootloader doit tout faire, y compris l'initialisation du contrôleur mémoire
 - ▶ Lors de la mise sous tension, le CPU commence l'exécution à une adresse fixée
 - ▶ Le design matériel fait en sorte qu'une partie de la Flash soit mappée à cette adresse
 - ▶ Le point d'entrée du bootloader est stocké à cette adresse dans la Flash: il prend le contrôle du matériel dès sa mise sous tension
 - ▶ Parfois un bootloader de premier niveau de petite taille

Bootloader: U-Boot

- ▶ Il existe de nombreux bootloaders pour les architectures non-x86, certains plus ou moins spécifiques à certaines architectures ou plateformes
- ▶ Le bootloader libre le plus utilisé est **Das U-Boot**
- ▶ Il supporte un grand nombre d'architectures, est aisément configurable et modifiable
- ▶ Fonctionnalités de base
 - ▶ Téléchargement du noyau et du système de fichiers par le réseau (protocole TFTP)
 - ▶ Protection, effacement et écriture sur la Flash
 - ▶ Exécution du noyau
 - ▶ Outils de diagnostics: lecture/écriture mémoire, test de périphériques, etc.
- ▶ <http://www.denx.de/wiki/U-Boot>

Noyau Linux

- ▶ Composant essentiel d'un système embarqué
- ▶ Les éléments de base du système: gestion des processus, de la mémoire, systèmes de fichiers, protocoles réseau, etc.
- ▶ Contient les pilotes pour la plupart des périphériques
- ▶ Le noyau distingue trois niveaux pour le support du matériel embarqué
 - ▶ L'architecture: ARM, MIPS, PowerPC
 - ▶ Le processeur: Samsung SC2442 par ex.
 - ▶ La machine: OpenMoko Freerunner

Noyau Linux

- ▶ Le noyau est le plus souvent porté sur une carte par le vendeur de celle-ci, sinon il faut s'adresser à une société spécialisée, ou mettre les mains dans le camboui.
- ▶ Un fichier de configuration est fourni pour chaque machine, il est personnalisable
- ▶ Après compilation, le noyau c'est
 - ▶ Une image binaire (le noyau lui-même), le plus souvent compressée, d'une taille de ~600 Ko à plusieurs Mo. C'est cette image qui est chargée et exécutée par le bootloader
 - ▶ Optionnellement des modules noyau

Bibliothèque standard C

- ▶ La bibliothèque de base qui s'intercale entre d'un côté toutes les autres bibliothèques et applications et d'un autre côté le noyau
- ▶ Elle fait partie de la chaîne de cross-compilation
- ▶ Trois solutions
 - ▶ GNU Libc, la version standard utilisée sur tous les systèmes desktop/serveur. Fonctionnalités complètes, mais grosse.
 - ▶ uClibc, une ré-écriture complète d'une libc plus simple, optimisée pour la taille et configurable en fonctionnalités
 - ▶ eglibc, une reprise de la GNU Libc ajoutant plus de flexibilité au niveau de la configuration
- ▶ Avec la libc sur un système embarqué, on a déjà une API de programmation “riche” pour des applications non-graphiques
 - ▶ Threads, IPC, entrées-sorties, réseau, etc.

Busybox

- ▶ Besoin d'un ensemble d'outils de base pour la cible
- ▶ cp, ls, mv, mkdir, rm, tar, mknod, wget, grep, sed et tous les autres
- ▶ Une solution: utiliser les outils GNU classiques
 - ▶ fileutils, coreutils, tar, wget, etc.
 - ▶ Inconvénient: beaucoup d'outils, pas conçus pour l'embarqué
- ▶ Une meilleure solution: Busybox
 - ▶ Tous les outils dans un seul programme binaire
 - ▶ Des outils aux fonctionnalités réduites... et à taille réduite
 - ▶ Extrêmement configurable
 - ▶ Des liens symboliques pour les utiliser comme d'habitude
 - ▶ <http://www.busybox.net>
 - ▶ Utilisé dans de très nombreux de produits du marché

Bibliothèques graphiques

- ▶ Des solutions graphiques de bas niveau
 - ▶ Le framebuffer géré par le noyau Linux
 - ▶ DirectFB, qui offre une interface de programmation plus abordable
 - ▶ X.org Kdrive, un serveur X simplifié
 - ▶ Nano-X
- ▶ Les solutions graphiques de plus haut niveau
 - ▶ Qt, peut fonctionner directement sur le framebuffer du noyau ou au-dessus d'un serveur X
 - ▶ Gtk, peut fonctionner au-dessus de DirectFB ou d'un serveur X
 - ▶ WxEmbedded, au-dessus de X, DirectFB ou Nano-X

Bibliothèques et outils

- ▶ En théorie, toutes les bibliothèques et tous les outils libres peuvent être cross-compilés et utilisés sur une plateforme embarquée
 - ▶ Une fois le système en place, c'est juste du Linux !
- ▶ En pratique, la cross-compilation n'est pas toujours aisée, car pas prévue par les développeurs originaux
 - ▶ Bien qu'étant horribles, les autotools bien utilisés sont la meilleure voie vers un logiciel cross-compilation-aware
- ▶ Des outils plus spécifiquement destinés aux plateformes limitées
 - ▶ dropbear à la place d'OpenSSH, BusyBox, Lighttpd ou Boa comme serveur HTTP

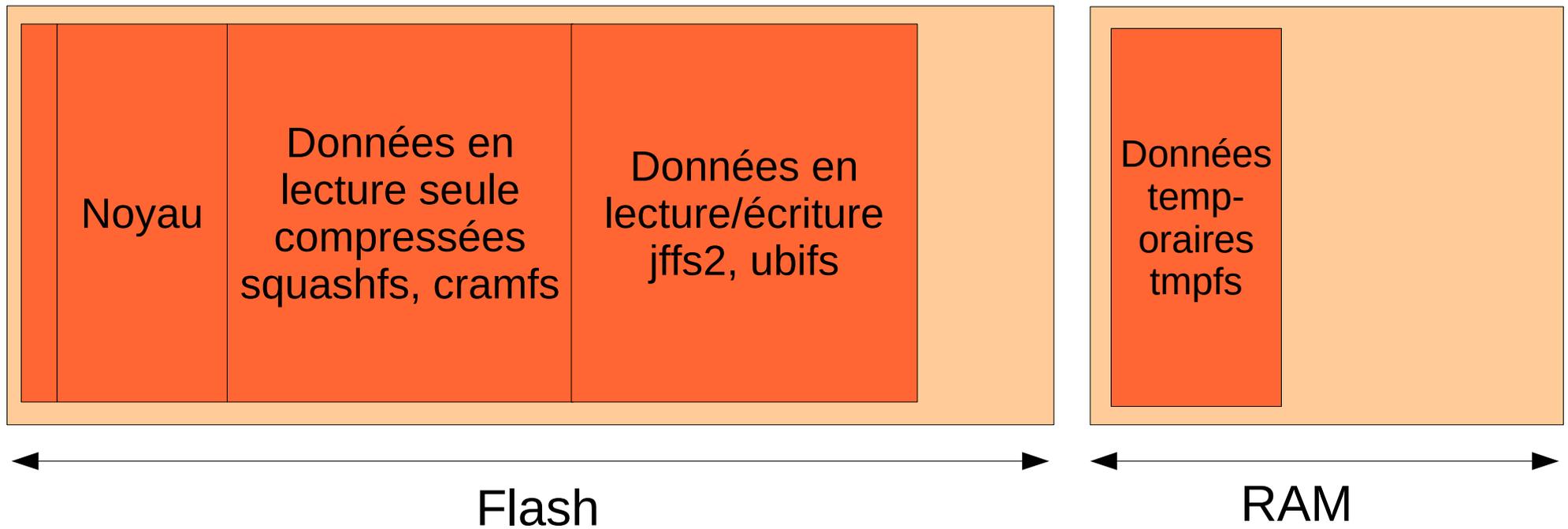
Outils de construction

- ▶ Plusieurs approches pour la construction d'un système embarqué
 - ▶ À la main
 - ▶ Pénible, peu reproductible, difficulté de trouver les bonnes options, d'appliquer les bons patches, etc.
 - ▶ Par des outils de construction
 - ▶ Buildroot
 - ▶ OpenEmbedded
 - ▶ PTXdist
 - ▶ Par des distributions
 - ▶ Gentoo Embedded
 - ▶ Debian Embedded

Stockage

- ▶ Les Flash des systèmes embarqués sont généralement des flash brutes
- ▶ Aucun matériel ne s'occupe du « wear leveling », c'est à dire la répartition des écritures
- ▶ Ces flash sont considérées par le noyau comme des MTD, Memory Technology Devices
- ▶ Des systèmes de fichiers spécifiques doivent être utilisés
- ▶ Aujourd'hui: JFFS2, UBIFS
- ▶ Demain: LogFS, AXFS
- ▶ Autres systèmes de fichiers pour l'embarqué: squashfs et cramfs, pour les parties en lecture seule, tmpfs pour les données temporaires

Stockage



Gestion de l'énergie

- ▶ CPUfreq
 - ▶ Similaire à ce qu'on a sur x86, variation de la fréquence et de la tension d'alimentation du processeur
- ▶ CPUidle
 - ▶ Gère les différents niveaux de sommeil d'un processeur
- ▶ Suspend
 - ▶ Vers la RAM le plus souvent
- ▶ « Clock framework »
 - ▶ Gère l'ensemble des horloges du CPU et connaît quelles horloges sont utilisées par quels périphériques
- ▶ Régulateurs
 - ▶ Des CPUs récents intègrent des régulateurs de courant et tension contrôlables par logiciel

Temps réel

- ▶ Une contrainte dans certains systèmes embarqués est la prédictabilité du temps de réponse
 - ▶ Capture de données ou réaction suite à un évènement qui doivent avoir lieu dans un temps borné
- ▶ Nécessite un système dit « temps réel »
- ▶ Le temps réel n'a **rien à voir** avec la performance
 - ▶ Les bornes temporelles à garantir peuvent être larges, mais doivent être garanties
 - ▶ Un système temps réel est parfois globalement plus lent qu'un système à temps partagé classique

Temps réel et Linux

- ▶ Linux n'est pas conçu à l'origine comme un système temps réel, mais comme un système à temps partagé
 - ▶ Objectif: maximiser l'utilisation des ressources pour maximiser le rendement global des applications
- ▶ Deux approches pour apporter le temps réel au noyau
 - ▶ Une approche dans le noyau. Objectif: réduire au fur et à mesure les zones pendant lesquelles le noyau ne réagit pas à un évènement extérieur.
 - ▶ Linux-rt, patch maintenu par Ingo Molnar et Thomas Gleixner
 - ▶ Une approche de co-noyau, où un mini-noyau temps réel se charge de traiter les évènements importants, en garantissant des temps de réponse. Linux tourne en tâche de fond.
 - ▶ RTAI, Xenomai

Débogage

- ▶ Pour les parties bas-niveau, bootloader et noyau, utilisation du JTAG
 - ▶ Bus qui permet de contrôler directement le processeur
 - ▶ Besoin d'une sonde qui se connecte à la carte
 - ▶ Coté machine de développement, s'utilise en général au travers de gdb (cross-compilé)
- ▶ Pour les applications utilisateur
 - ▶ Utilisation de gdbserver sur la machine cible
 - ▶ Et d'un gdb cross-compilé sur la machine de développement, qui contrôle l'exécution de l'application à distance grâce à gdbserver

En savoir plus

- ▶ Supports de formation de Bootlin, disponibles sous licence libre
 - ▶ Intégration de système Linux embarqué
 - ▶ Développement de code noyau
- ▶ Livre « Building Embedded Linux Systems », O'Reilly, 2008
- ▶ Livre « Embedded Linux Primer », Prentice Hall, 2006
- ▶ Embedded Linux Wiki, <http://elinux.org>
- ▶ Conférences ELC et ELCE, vidéos mises à disposition par Bootlin
- ▶ Assez peu de littérature en français
 - ▶ Articles dans Linux Magazine
 - ▶ Livre « Linux Embarqué » par Pierre Fichoux, un peu obsolète car publié en 2002