

Systemes embarqués Linux temps réel

Environnements de développement
Fiabilité et tolérance aux fautes

Fabrice MONTEIRO

Laboratoire Interfaces Capteurs Microélectronique
Université Paul Verlaine — Metz

Logiciels Embarqués Temps Réel

Qu'est-ce qu'un logiciel embarqué ?

“Embedded Software” en anglais

- Programme utilisé dans un équipement industriel ou bien de consommation grand public.
- Les différences essentielles avec un logiciel classique sont :
 - la complète intégration du logiciel embarqué dans cet équipement
 - il n'a pas de raison d'être en dehors de l'équipement pour lequel il est conçu ;
 - l'équipement est valorisé uniquement par son aspect fonctionnel ;
 - un bon logiciel intégré l'est au point de se faire oublier !

Temps réel

En informatique industrielle,
on parle de **système embarqué temps réel** lorsque :

- ce système contrôle (surveillance et/ou pilote) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé ;
- le respect des contraintes temporelles est aussi important que l'exactitude du résultat → **un résultat délivré en dehors des délais imposés est un résultat faux.**

⇒ **le système d'exploitation embarqué doit être temps réel**

Systemes embarqués temps réel

- **Exemples de système temps réel**
 - **systemes de contrôle de procédé**
(usines, centrales nucléaires) ;
 - **systemes aéronautiques**
(guidage missiles, avions, satellites) ;
 - **systemes automobiles**
(injection électronique, ABS, EPS, airbags) ;
 - **systemes multimédia mobiles**
(téléphonie, vidéo, réalité virtuelle).

Caractéristiques pour les OS temps réel

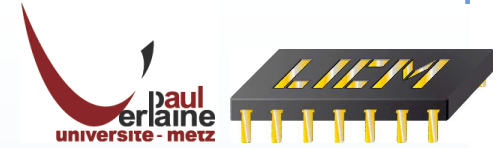
- **Rapidité :**
 - commutations de contexte rapides ;
 - performance des algorithmes d'ordonnancement ;
 - accessibilité des structures de données...
- **Prédictibilité :**
 - prévoir les futures violations d'échéance ;
 - y faire face (techniques de tolérance aux fautes, migration...).
- **Adaptativité :**
 - reconfigurabilité dynamique (sous contraintes temps réel) ;

Sociétés éditrices généralement de taille moyenne

- Difficultés à suivre l'évolution technologique :
 - durée de vie du processeur \approx 12 / 24 mois ;
 - évolution encore plus rapide des standards logiciels ;

⇒ **problèmes d'obsolescence**
- Segment de marché très spécialisé :
 - coûts des licences et droits de redistribution (très) élevés.
- Disparition possible de l'éditeur, du système et du support technique correspondant :
 - **pratique dangereuse** du reverse engineering ;
 - ou bien, achat à **coût élevé** des sources.

Pourquoi LINUX est-il adapté à l'embarqué ?



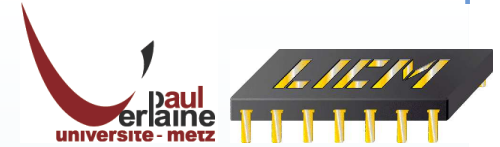
- **Fiabilité**

- LINUX est réputé pour sa fiabilité. L'écran bleu de la mort (blue screen of death) est inconnu des utilisateurs de LINUX.

- **Faible coût**

- LINUX est exempt de royalties ;
- les outils de développement sont disponibles sous licence GPL ;
- le seul effort financier nécessaire à l'adoption de LINUX se situe sur **la formation et le support technique.**

Pourquoi LINUX est-il adapté à l'embarqué ?



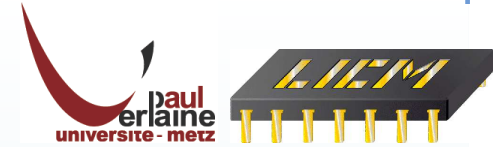
- **Performances**

- les performances de LINUX ne sont plus à prouver ;
- c'est même dans des situations de faibles performances matérielles que LINUX se révèle le plus efficace comparé à d'autres systèmes.

- **Portabilité et adaptabilité**

- porté sur un très grand nombre d'architectures matérielles (y compris de faible puissance, comme le prouve le projet μ Clinux) ;
- large support des architectures x86 et ARM (très présente dans le domaine de l'embarqué) ;
- réalisation aisée d'une distribution réduite et fiabilisée, adaptée aux applications embarquées à partir d'une distribution standard.

Pourquoi LINUX est-il adapté à l'embarqué ?



- **Ouverture**

- entièrement open source, ce qui facilite l'adaptation et le test de nouveaux protocoles ;
- forte inter-opérabilité avec d'autres systèmes d'exploitation ;
- adoption préférentielle des nouvelles technologies constituant des standards ouverts.

- **La tendance du marché**

- chiffre d'affaire généré par Linux embarqué 65 M\$ en 2003, environ 120 M\$ en 2006 ;
- part importante des revenus du marché de Linux embarqué réalisée sur des prestations de services, d'intégration ou de transfert technologique.

Caractéristiques dérivées du caractère open source

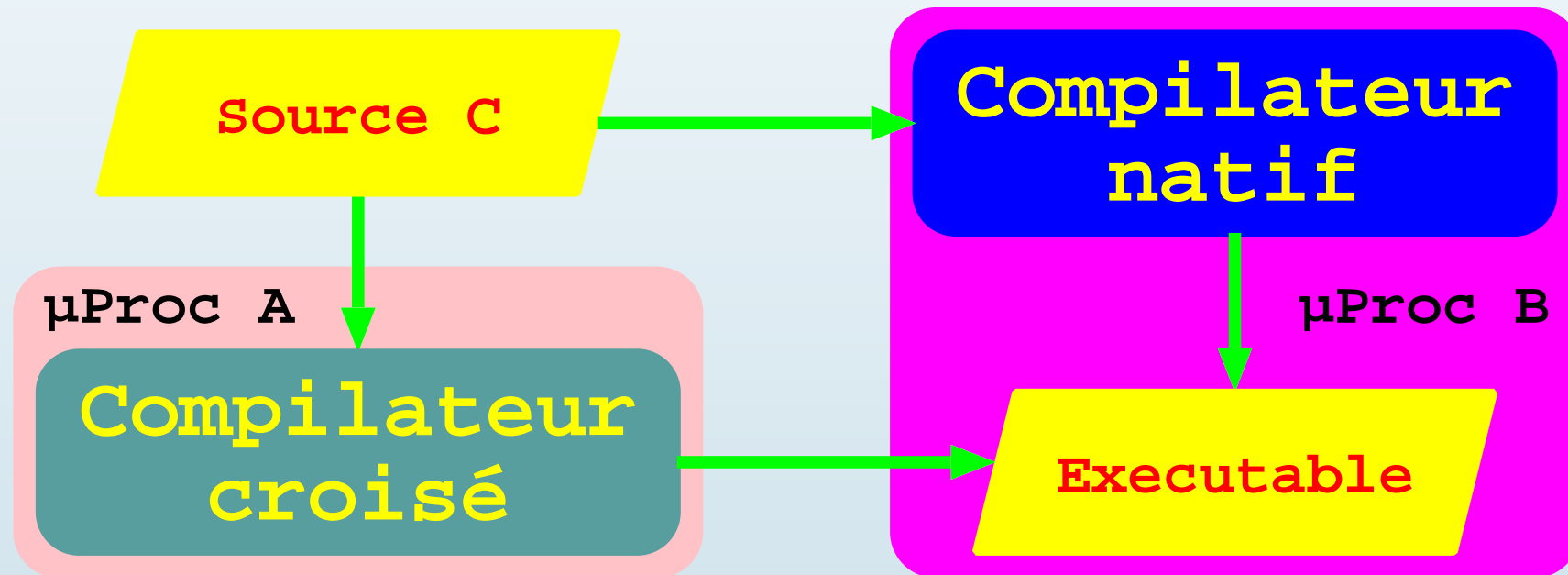
- (+) Redistribution sans royalties.
- (+) Disponibilité du code source.
- (+) Réalisation possible de développements dérivés de ce code source.
- (+/-) Modèle de développement communautaire, non centralisé
- (-) Multiplicité des types de licence.
- (-) Problème de crédibilité (méfiance des décideurs).
- (-) Responsabilités juridiques.

Linux embarqués

- Solutions professionnelles (commerciales) clé en main pour construire un Linux embarqué sur des cibles diverses :
 - Montavista
 - LinuxWorks
 - RedHat
 - **TimeSys/Linux RTOS...**
- Solutions libres
 - projet LRP, projet Tom's Boot Root, projet ELKS, projet Small Linux, projet PeeWeeLinux, projet μ Clinux, **RTLinux, RTAI, eCOS...**

Environnements de travail et outils de développement

Développement natif et développement croisé.



Compilation croisée

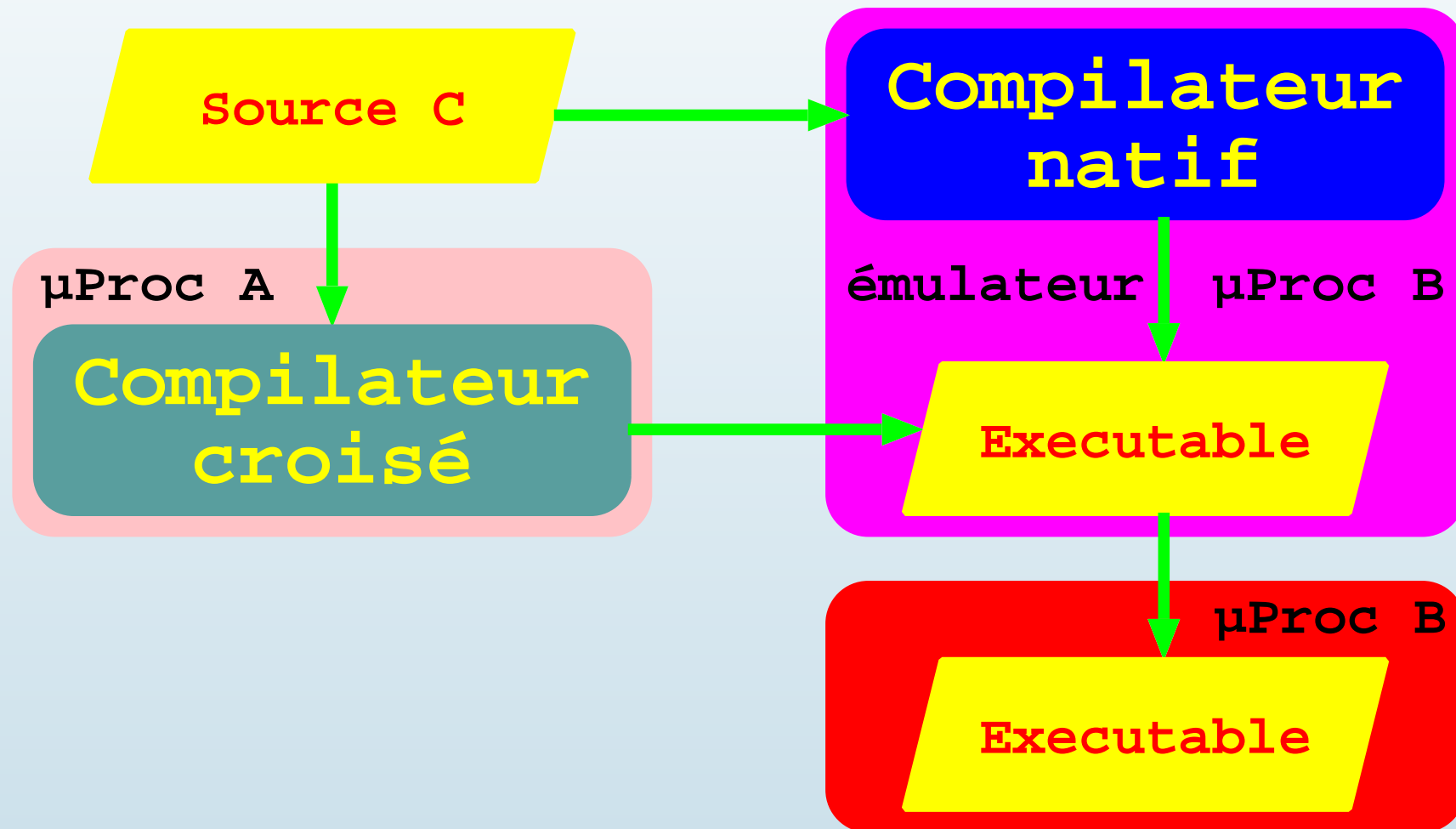
Soient μ ProcA et μ ProcB 2 architectures matérielles \neq

- Un **compilateur croisé** (*cross compiler*) traduit un **code source** sur l'architecture μ ProcA en **code objet** pour l'architecture μ ProcB

Intérêt :

- **Ressources limités sur architecture μ ProcB**
 - exemple \rightarrow architecture ARM ;
 - environnement de travail potentiellement limité ;
 - temps de compilation prohibitif.
- **Ressources abondantes sur μ ProcA**
 - exemple \rightarrow architecture de type PC ;
 - environnement de travail conventionnel ;
 - temps de compilation courts.

Utilisation d'un émulateur



Principe d'un émulateur de système

- une machine virtuelle (ou émulateur de système)
 - fournit une implantation virtuelle de l'architecture cible ;
 - permet de faire tourner un ou plusieurs systèmes d'exploitation (ou seulement des processus) sur un système d'exploitation déjà installé sur la machine.

Intérêt (+) / (-)

- (+) développement logiciel en amont du système matériel possible ;
- (+) environnement de travail "homogène" ;
- (+) outils de débogage approfondi ;
- (-) vitesse d'exécution plus faible (voire beaucoup) ;
- (-) caractéristiques du système plus ou moins fidèles.

Construction d'un Linux embarqué (1)

Construire un petit système embarqué à partir de rien en 40 minutes, c'est possible ! Et en plus, c'est assez simple !

- Configuration et compilation du noyau Linux
- Création du système de fichiers racine
- Compilation et installation de Busybox
- Création des fichiers spéciaux pour les périphériques
- Scripts de démarrage du système : systèmes de fichier virtuels, réseau
- Configuration d'une interface HTTP simple

extrait de "Linux from Scratch en 40 minutes" de "Free Electrons"

<http://free-electrons.com/publications/lfs/>

Construction d'un Linux embarqué (2)

Approche du plus vers le moins pour construire un système embarqué

- Commencer à partir d'un environnement de bureau complet sous GNU/Linux (Debian, Fedora...) et supprimer tous les éléments inutiles.
- Travail très complexe : besoin de vérifier un très grand nombre de fichiers et de paquetages. Besoin de comprendre l'(in)utilité de chaque fichier avant de le supprimer.
- Conserver des scripts et des fichiers de configuration inutilement complexes.
- Le résultat final est quand même trop gros car les outils et les bibliothèques standards sont utilisés. De plus, beaucoup de bibliothèques partagées sont nécessaires.

Construction d'un Linux embarqué (3)

Approche du moins vers le plus pour construire un système embarqué

- Commencer avec un système de fichiers minimal, voire vide, en ajoutant seulement les éléments qui vous sont nécessaires.
- Bien plus facile à réaliser ! Vous passez du temps sur ce dont vous avez besoin.
- Plus facile à contrôler et à maintenir : vous développez une compréhension des outils que vous utilisez.
- Vous avez seulement besoin de scripts de configuration basiques.
- Le résultat final peut être extrêmement petit, d'autant plus que vous utilisez des outils légers.

Construction d'un Linux embarqué (4)

Utilisation d'un émulateur logiciel de processeur

→ QEMU (<http://qemu.org>)

- Émulateur rapide de processeur utilisant un traducteur dynamique portable.

Deux modes opérationnels

- Émulation d'un système complet : processeur et périphériques divers (support de x86, x86_64, ppc).
- Émulation du mode utilisateur (seulement sur un hôte Linux) : peut exécuter des applications compilées pour d'autres CPU (support de x86, ppc, arm, sparc).

Construction d'un Linux embarqué (5)

Une boîte à outils complète : <http://www.busybox.net/> de Codepoet Consulting

- La plupart des outils Unix en ligne de commande avec un seul exécutable ! Inclut même un serveur web !
- Fait moins d'1 Mo (compilé statiquement avec glibc), moins de 500 Ko (compilé statiquement avec μ Clibc)
- Configuration aisée des fonctionnalités à inclure
- Le meilleur choix pour :
 - Initrds avec des scripts complexes.
 - Tout système embarqué !

glibc — <http://www.gnu.org/software/libc/>

- Bibliothèque C du projet GNU (présente sur GNU/Linux).
- Performance, respect des standards et portabilité.
- Assez volumineuse pour des petits systèmes embarqués :
→ près de 1,7 Mo sur les iPAQs Linux (libc+libm)
- Taille du programme exemple “hello world” :
12 Ko / 350 Ko (lien dynamique/statique)

μ Clib — <http://www.uclibc.org/>

- Bibliothèque C légère pour petits systèmes embarqués, mais intègre néanmoins beaucoup de fonctionnalités.
- Taille réduite : approx. 360 Ko (libuClibc+libm)
- Taille du programme exemple “hello world” :
2 Ko / 18 Ko (lien dynamique/statique).

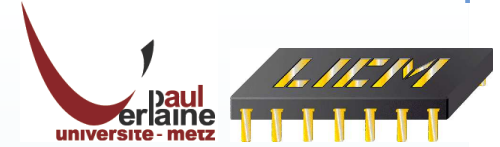
Compétences nécessaires

Un système embarqué est :

- un système mixte (matériel+logiciel) ;
- soumis à des contraintes temps réel ;
- en interaction avec son environnement (mécatronique, transmissions, multimédia mobile) ;
- soumis à des contraintes de sûreté de fonctionnement.

⇒ **connaissances pluridisciplinaires nécessaires**

Systemes embarqués numériques complexes



- Préconisation de technologies électroniques programmables
 - Circuits programmables performants (nb portes logiques et pins, fréquences élevés) ;
 - Méthodes de développement conjoint (codesign) ;

⇒ évolution vers SoC/SPoC, NoC, MPSoC

System On Chip, Network on Chip, Multi-Processor on Chip

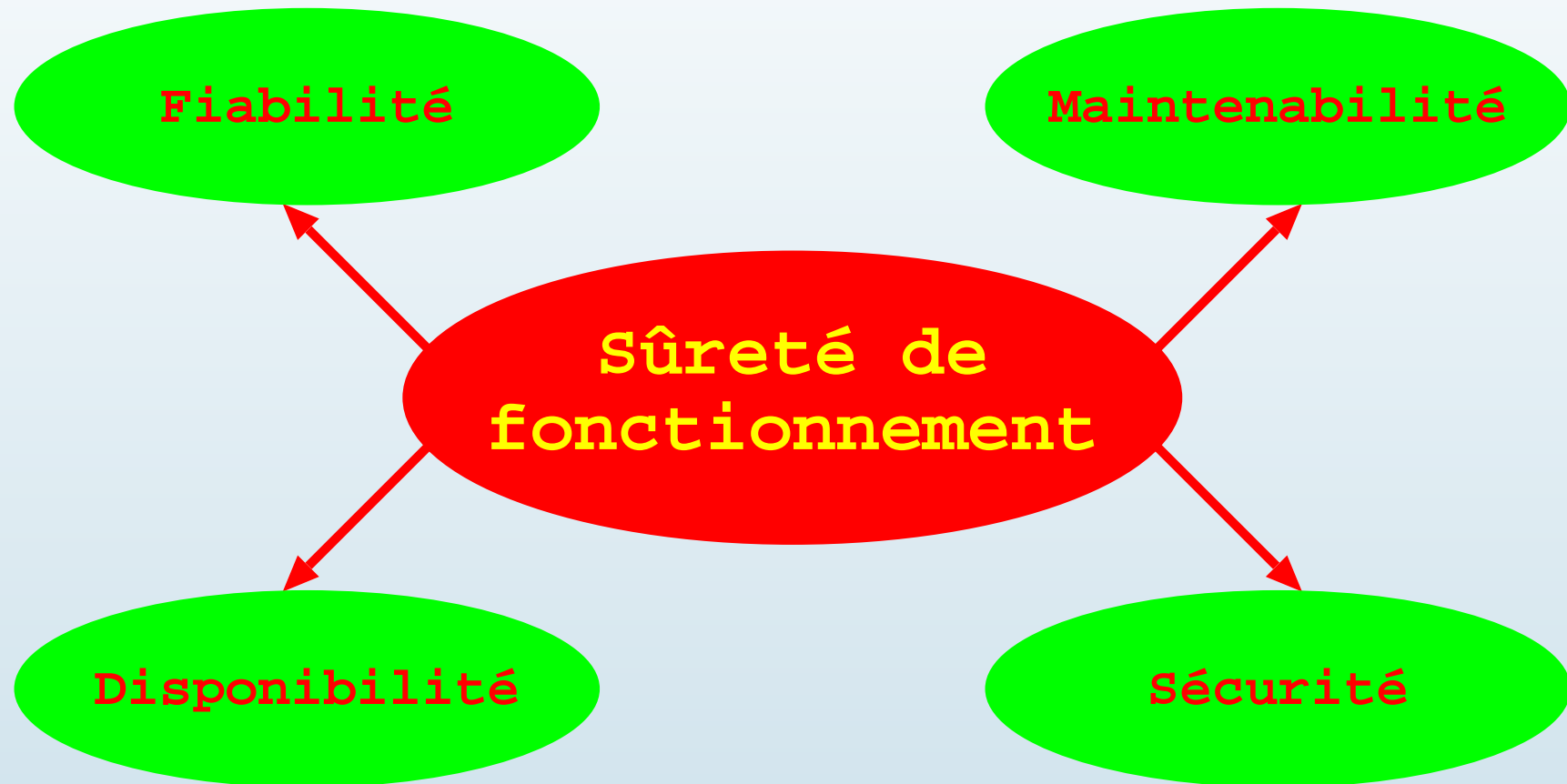
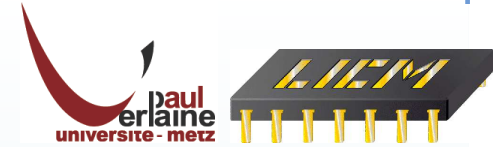
⇒ **algorithmique “matérielle”**

- langages de spécification matérielle (VHDL, Verilog)
- langages de spécification système (SystemeC)

Problématique de la Sûreté de Fonctionnement

- Systèmes logiciels larges
 - 20000 bugs pour chaque million de lignes de code
 - 90% des ces bugs trouvées en faisant des tests
 - 200 bugs détectés durant la 1ère année d'utilisation
 - 1800 bugs non détectés
- Maintenance logicielle
 - fixation de 200 bug
 - introduction de 200 nouvelles fautes !

Sûreté de fonctionnement : FMDS



Définition de la FMDS

- **Fiabilité** : (Reliability)
probabilité qu'un système informatique fonctionne sans tomber en panne pendant un certain temps, dans des conditions d'utilisation bien définies (continuité de service)
- **Maintenabilité** : (Maintainability)
aptitude d'un système défaillant à être maintenu ou réparé
- **Disponibilité** : (Availability)
être prêt à l'utilisation
- **Sûreté** : (Safety/Security)
pas de conséquences catastrophiques pour l'environnement, protection contre accès et/ou manipulations non autorisés de l'information

Défaillances

- **Défaillance (panne)** : survient quand le comportement d'un système viole sa spécification de service
- Les défaillances résultent de problèmes inattendus internes au système qui se manifestent éventuellement dans le comportement externe du système
- Ces problèmes sont appelés **erreurs** et leur causes mécaniques ou algorithmiques sont appelées **fautes**

Fautes → Erreurs → Défaillances

Sources des fautes

- Différentes sources de fautes peuvent être à l'origine d'une défaillance d'un système
- Notamment :
 - spécification inadéquate des fonctionnalités
 - erreurs de conception dans le logiciel
 - défaillance du processeur
 - interférence au niveau du sous-système de communication

Types de fautes

- Classification des fautes :
 - **Transitoire :**
se produit de manière isolée (une fois et disparaît)
 - **Intermittente :**
se reproduit sporadiquement (transitoires multiples)
 - **Permanente :**
persiste indéfiniment (jusqu'à réparation) après son occurrence

Types de défaillances

- **Crash :**
le programme s'arrête, mais il fonctionnait correctement avant l'arrêt
- **Panne d'omission :**
le programme ne répond pas à une requête
- **Panne de temporisation :**
la réponse survient hors des délais temps réel spécifiés
- **Panne de réponse :**
la réponse est simplement incorrecte
- **Panne arbitraire (Byzantine) :**
des réponses arbitraires sont produites à des instants arbitraires

- **Prévention de fautes :**
comment prévenir l'occurrence ou l'introduction de fautes
- **Tolérance aux fautes :**
comment fournir un service en dépit des fautes
- **Élimination de fautes :**
comment réduire la présence (nombre et sévérité) de fautes
- **Prévision de fautes :**
comment estimer la présence, la création et les conséquences des fautes

Prévention de Fautes

Stratégies pour l'**évitement de fautes**

- limiter l'introduction de fautes pendant la conception du système
 - spécification rigoureuse des besoins, formellement si possible ;
 - utilisation de composants plus fiables ;
 - mise en oeuvre de méthodologies de conception strictes
 - emploi de langages moins permissifs (abstraction de données élevé et modularité) ;
 - mise en oeuvre de méthodes formelles de vérification.

Tolérance aux Fautes (1)

Un système est tolérant aux fautes :

- s'il peut masquer la présence de fautes en utilisant la redondance
 - le service rendu sera éventuellement dégradé, mais dans ce cas les contraintes de sûreté devront être respectées
 - système *fail-safe* par exemple
 - la redondance n'a d'autre but que de favoriser la tolérance

Tolérance aux Fautes (2)

La redondance peut arborer plusieurs formes :

- **Redondance matérielle :**
ajout de composants matériels
- **Redondance logicielle :**
ajout d'instructions, routines, programmes
- **Redondance temporelle :**
ajout de temps pour répéter des tâches (instructions)
- **Redondance d'information :**
en fait, il s'agit d'un effet direct des redondances précédentes

Niveau de Tolérance aux Fautes

Niveau de tolérance défini par l'application

- **Tolérance aux fautes complète :**
poursuite du service en présence de fautes sans perte significative de fonctionnalités ou de performance
- **Dégradation gracieuse (failsoft) :**
dégradation partielle des fonctionnalités ou de performance accepté durant le recouvrement ou la réparation
- **Fail-Safe :**
le système doit maintenir son intégrité quitter à accepter un arrêt temporaire de son fonctionnement
→ les actions issues du systèmes sont soit exactes, soit sûres (sans danger pour l'application / l'environnement)

Détection de Fautes / Erreurs

Tolérance aux Fautes

→ basée sur la capacité à détecter les fautes

- Détection de fautes réalisée via la détection d'erreur (effets "visibles" des fautes)

Détection d'erreurs réalisée en ligne

(*on-line testing, on-line checking*)

- Surveillance et détection externe
- Auto-surveillance et détection systèmes *self-checking, totally self-checking...*