

Introduction aux systèmes temps réel

Iulian Ober

IRIT

ober@iut-blagnac.fr

Définition

Systèmes dont la **correction** ne dépend pas seulement des *valeurs* des résultats produits mais également des *délais dans lesquels les résultats sont produits*

- le système doit répondre à des stimuli externes sous un **délat spécifié**
- L'absence de réponse est aussi grave qu'une réponse erronée, voire plus

Exemples :

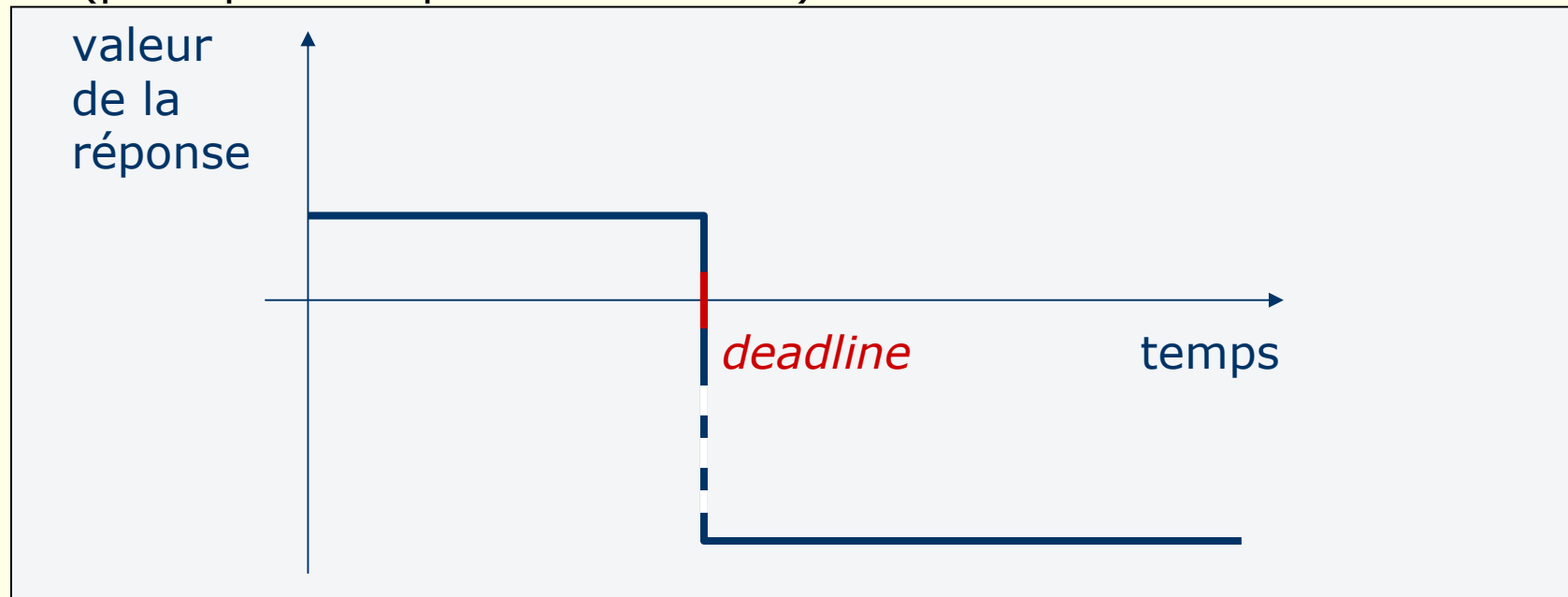
- contrôle de véhicules (automobiles, trains, avions...),
- robotique
- systèmes militaires de contrôle / commande
- contrôle du trafic (aérien, terrestre)
- contrôle de processus industriels
- ...

Catégories de systèmes temps réel

- Temps réel dur (*hard real time*)
- Temps réel ferme (*firm real time*)
- Temps réel mou (*soft real time*)

Temps réel dur

La réponse du système dans les délais est **vitale**.
L'absence de réponse est catastrophique
(plus qu'une réponse incorrecte)

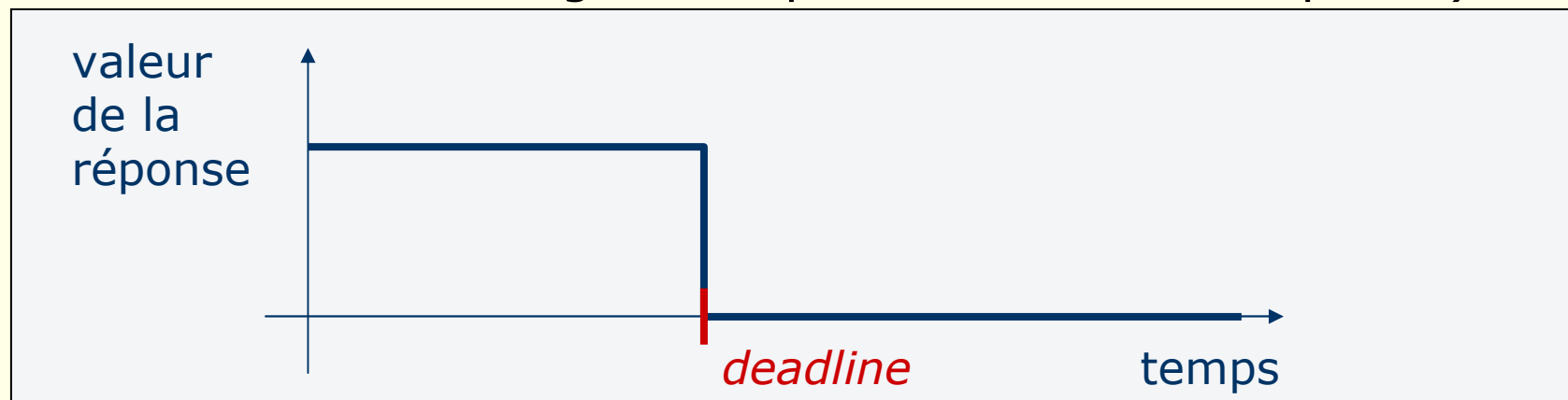


Exemples : contrôle aérien, contrôle d'une centrale nucléaire...

Temps réel ferme

La réponse du système dans les délais est **essentielle**. **Le résultat ne sert plus à rien** une fois le *deadline* passé.

(définition alternative : la pénalité de non-réponse est dans le même ordre de magnitude que la valeur de la réponse)

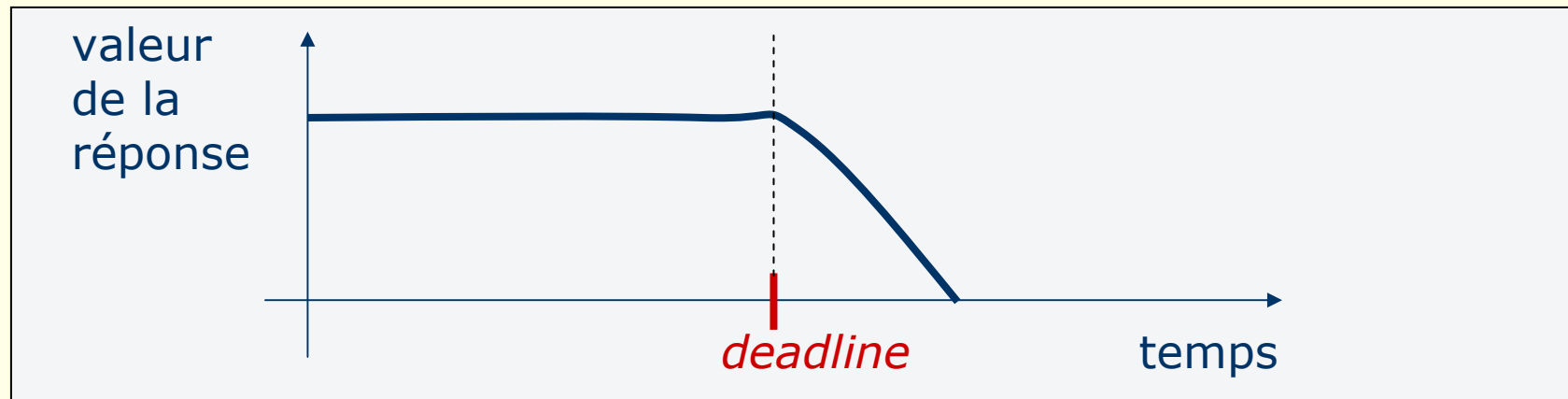


Exemples : transactions en bourse...

(c'est subjectif : le temps réel ferme de l'un peut être le temps réel dur de l'autre)

Temps réel mou

La réponse du système après les délais **réduit progressivement sont intérêt.**
Les pénalités ne sont pas catastrophiques.



Exemples : VoD, logiciel embarqué de votre téléphone, iPod, etc.

Classification – mot de fin

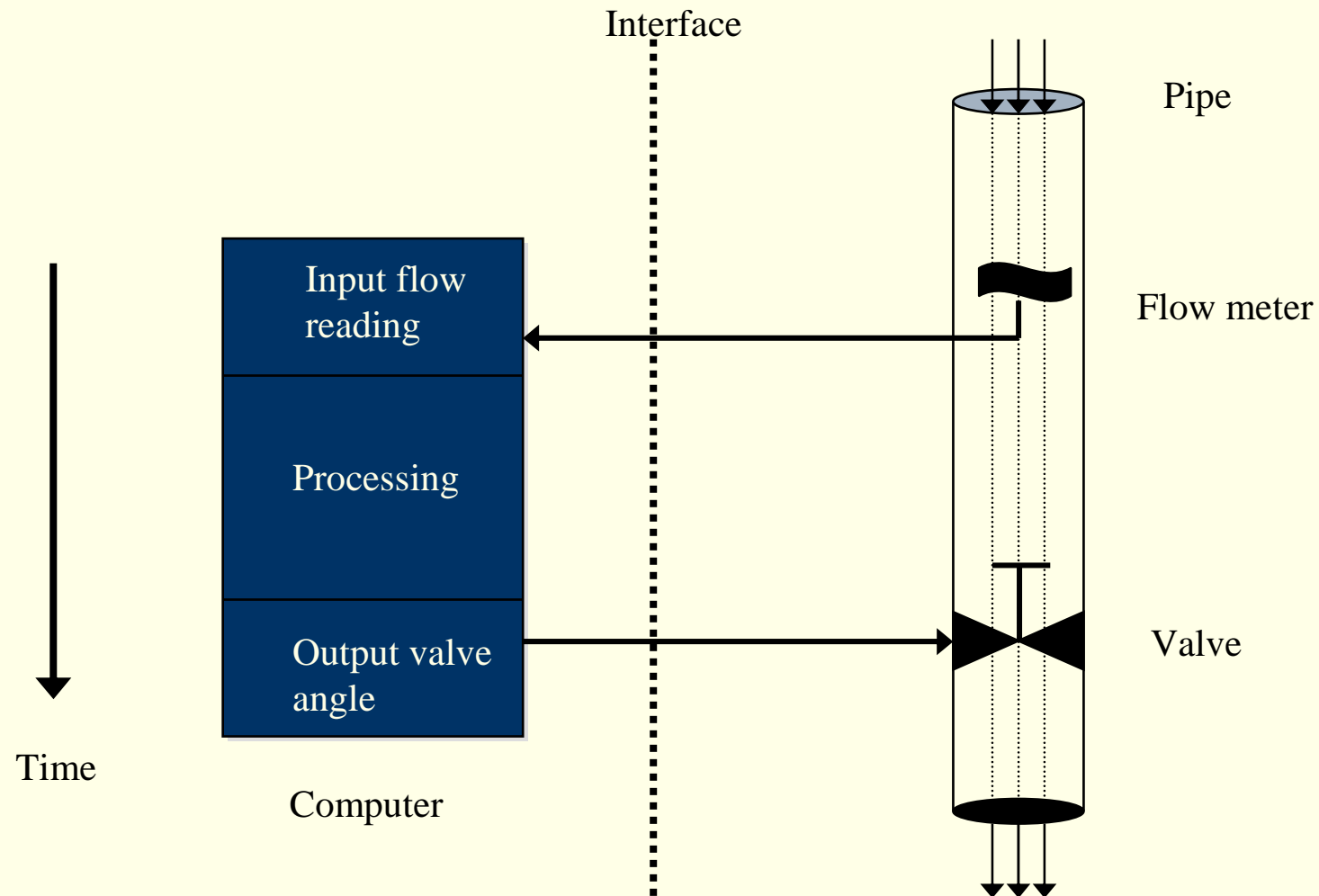
- Un même système peut avoir des sous-systèmes TR-dur, TR-ferme ou TR-mou
- ...en fait, la fermeté est une caractéristique de chaque échéance (deadline)

Sommaire

- Généralités
 - Caractéristiques récurrentes des STR
 - Types de problèmes soulevées
- Programmation des STR
 - Prog. concurrente
 - Synchronisation et communication
 - Facultés temporelles

Exemples typiques

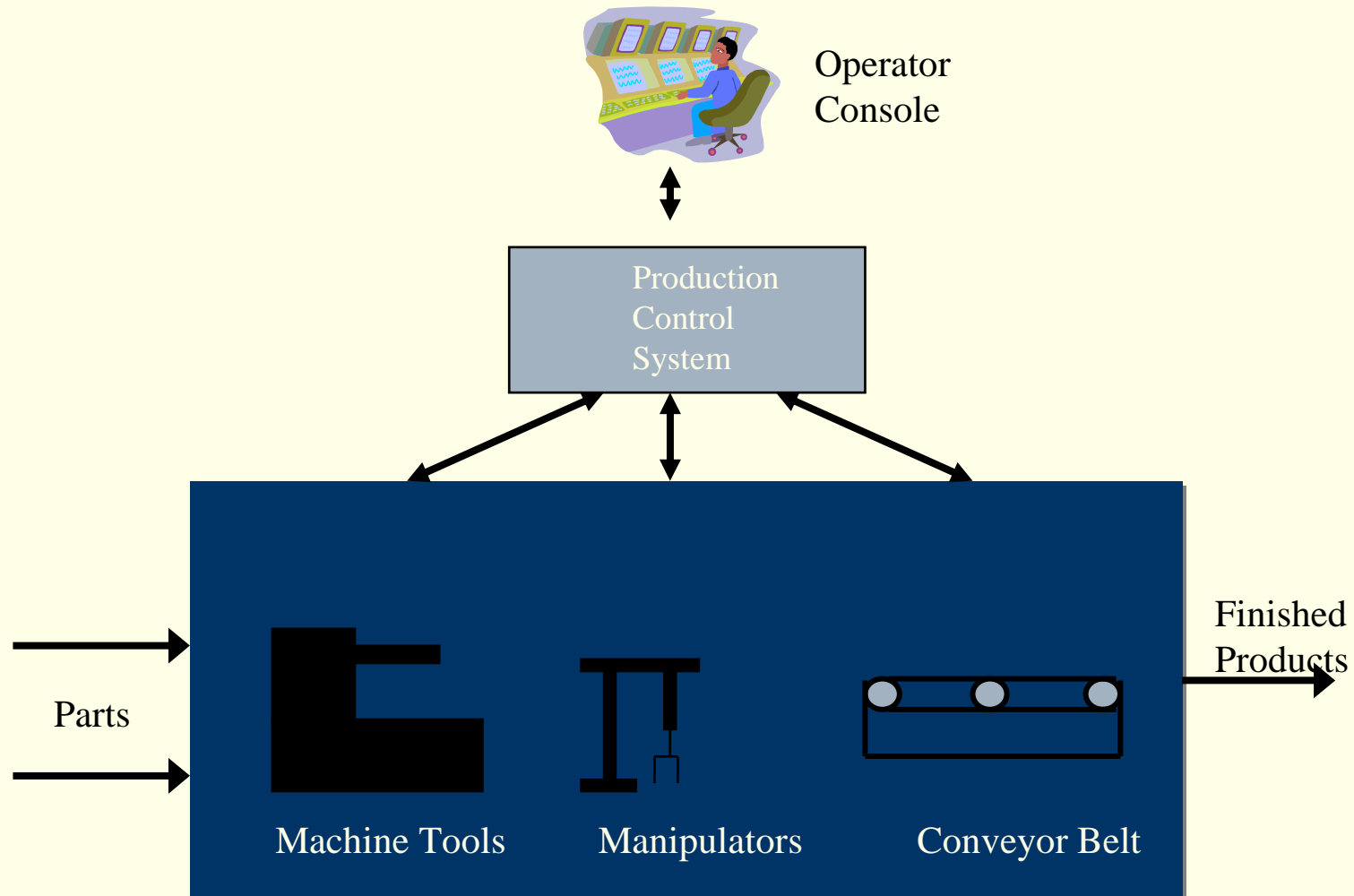
Un système de contrôle du débit (*)



(*) Source: "Real Time Systems and Programming Languages"
© Alan Burns and Andy Wellings, 2001

Exemples typiques

Un système de contrôle de la production (*)

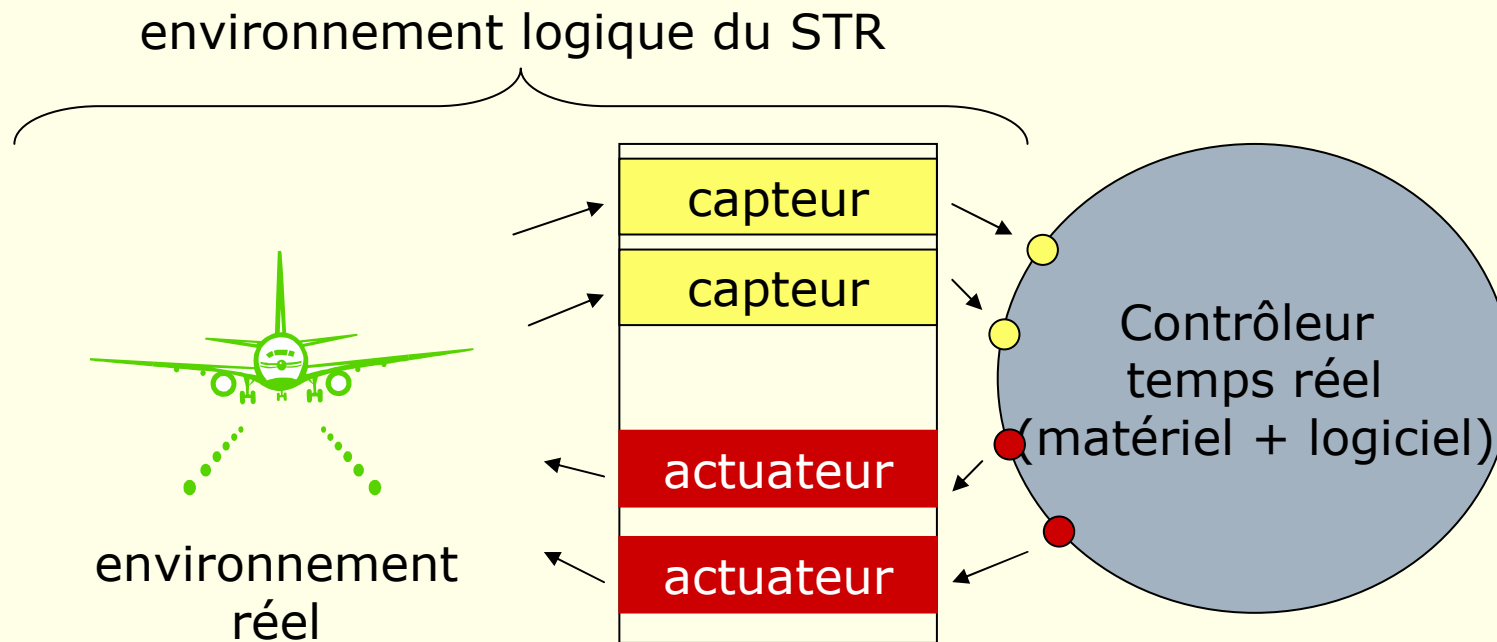


(*) Source: "Real Time Systems and Programming Languages"
© Alan Burns and Andy Wellings, 2001

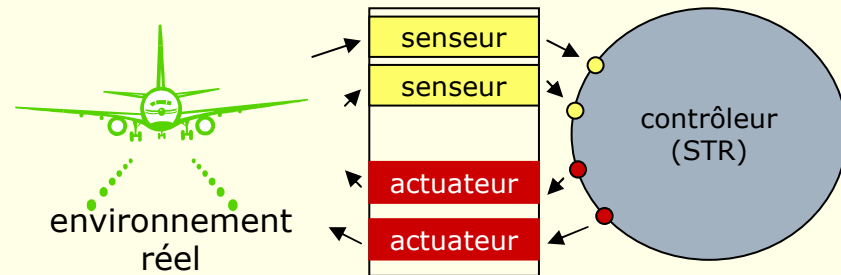
Cas typique : le contrôleur

Les systèmes temps réel sont souvent **embarqués** dans un équipement spécialisé, leur but étant de **contrôler** l'équipement et/ou son *environnement*

- Environ 99% des processeurs produits dans le monde sont dédiés aux systèmes embarqués



Systemes réactifs



systeme **ouvert** répondant **constamment** aux sollicitations de son environnement en produisant des actions sur celui-ci

- par opposition : systèmes transformationnels (e.g., indexation d'une base de données...)
- abstraction => système qui tourne à l'infini
- vie du système divisée en modes d'exécution (e.g., sol, décollage, croisière, ...)

Contraintes temporelles

- sont souvent dérivées de lois physiques
- se présentent souvent sous forme de **fréquences**

E.g., pour contrôler un avion en croisière à la vitesse V , il faut ajuster la position des gouvernes à la fréquence ν

=> le contrôleur doit produire une sortie vers actuateurs toutes les $1/\nu$ sec.

=> les calculs de chaque cycle doivent terminer en moins de $1/\nu$ sec.

Concurrence

La tâche du système se décompose souvent en plusieurs activités, qui doivent souvent être exécutées **en parallèle**.

- ...parce que **l'environnement est déjà parallèle**:
quand le vent bat, la gravitation ne s'arrête pas...
(et des fois, il y a des choses encore plus surprenantes qui arrivent...)
- ça peut être implanté par
 - des tâches concurrentes sur un mono-processeur
 - un système réparti (avec en plus de la concurrence dans les nœuds)
- les \neq activités sont **plus ou moins critiques**
ordinateur de bord de votre voiture gère : ESP, ABS, auto-radio...

PRÉDICTIBILITÉ

« LA » caractéristique requise avant tout d'un système temps réel:

- sous un ensemble **d'hypothèses** concernant *la charge* (e.g., fréquence des entrées) et *les erreurs non-contrôlables* (e.g., fréquence des erreurs de bit sur le bus), arriver à **prouver que...**
- ...toute les contraintes (notamment les délais) **sont respectées...**
- ...au moins pour les tâches critiques du système

Que préférez-vous: un ordinateur de bord qui déclanche l'ABS à 1ms du blocage des roues dans 99% de cas, ou un qui le déclanche à 10ms, mais dans 100% de cas ?

Systemes embarqués

Cela impose des contraintes supplémentaires

- Haute disponibilité requise
 - resets coûteux ou impossibles
 - updates – idem
 - Capacités de calcul et stockage prédéterminées et souvent limitées
 - les limites sont toutefois très variables
e.g. téléphone GSM vs. satellite militaire
 - Autres contraintes *non-fonctionnelles*
 - e.g., consommation (durée de vie des batteries),...
- ⇒ Processeurs et couches système personnalisés
- *overhead* dans le processus de développement

Autres caractéristiques(*)

- Grande taille et complexité parfois
 - E.g. millions of LoC for the ISS
 - => importance de la modularité
- Interaction avec du matériel dédié
 - Nécessité de programmer et d'interagir avec des pilotes à un niveau abstrait et sûr
- Échantillonnage des entrées
 - => manipulation des valeurs continues dans un domaine discret
 - algorithmes numériques, gestion d'erreurs
- Rapidité et efficacité parfois importante

(*) Source: "Real Time Systems and Programming Languages"
© Alan Burns and Andy Wellings, 2001

Malentendus fréquents (*)

- ❑ système temps réel = système rapide et performant
- ❑ la programmation temps réel veut dire assembleur, interruptions et pilotes
- ❑ il n'y a aucune science derrière le développement des systèmes temps réel, tout est une question de bidouillage
- ❑ toutes les problèmes du temps réel ont été déjà résolus dans d'autres domaines de l'informatique
- ❑ ...et l'augmentation de la vitesse des processeurs va résoudre ce qui reste

(*) inspiré de J. Stankovic, « Misconceptions about real-time computing », IEEE Computer, 21(10):10--19, Oct. 1988

Structure du cours

1. Programmation des STR

Concepts et exemples (Ada, Java)

- Concurrency
- Synchronisation et communication
- Aspects temporels

2. Ordonnement

Politiques statiques/dynamiques mono-processeur

- Modèle de tâches, hypothèses
- Construction de l'EDT processeur
- Critères de décision

3. Sûreté de fonctionnement

Méthodes de conception, d'analyse, de V&V

Sommaire

- Généralités
 - Caractéristiques récurrentes des STR
 - Types de problèmes soulevées
- Programmation des STR
 - Prog. concurrente
 - Synchronisation et communication
 - Facultés temporelles

Problèmes rencontrés

- allocation de ressources
 - ordonnancement, tolérance aux fautes, récupération de ressources...

- architecture
 - caractéristiques du processeur, du système de communication, des E/S

- méthode de développement
 - spécification de besoins, validation, langages et modèles

Allocation de ressources

Problème principal : l'**ordonnancement**

- allocation des tranches de temps processeur
- i.e., les moments où une tâche est livrée au processeur (*dispatch*) / suspendue (*preempt*)

Objectif de l'ordonnancement : assurer le respect des échéances *de manière prouvable*

Une **méthode d'ordonnancement** est caractérisée par:

- le **critère de test** d'ordonnabilité (**offline**)
- la méthode effective de construction de ***l'emploi du temps*** (**schedule**) du processeur (**online** ou **offline**)

Problèmes d'architecture

Pour supporter un logiciel temps-réel, l'architecture (matériel + OS) doit être **prédictible**:

- temps d'exécution des instructions
- changement de contexte
- opérations avec la mémoire
- traitement des interruptions
- ...

Implications

- ⇒ Les mémoires cache et les processeurs supéscalaires *sont évités*
- ⇒ La tolérance aux fautes est supportées dans le matériel (self-checking, voting and monitoring...)
- ⇒ Support pour des communications rapides et prédictibles (e.g., pas de Ethernet, ou alors amélioré)

Problèmes d'architecture (cont)

- ❑ support pour algorithmes d'ordonnancement dans l'OS, préemption rapide, contextes multiples, priorités...
- ❑ Fault Containment Regions
- ❑ gestion des interruptions
- ❑ synchronisation d'horloges
- ❑ etc.

Problèmes de méthode de développement

- **Spécification des besoins**
 - fonctionnelle : ce que le système doit calculer
 - non-fonctionnelle : e.g., contraintes temporelles
 - la spécification des deux aspects doit être **conjointe** et suffisamment **précise (formelle?)**
(pour permettre d'examiner ses implications et ses propriétés indépendamment de l'implantation)

- **Conception et développement**
 - constructions de langage pour gérer le **temps**, les **ressources**
 - support de la **communication** et de la **concurrence**
 - constructions pour supporter la vérification du **critère d'ordonnabilité** à la compilation