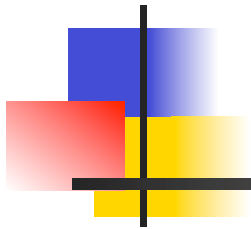


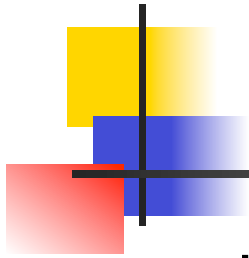
# Introduction au temps réel



Laurent Pautet

[Laurent.Pautet@enst.fr](mailto:Laurent.Pautet@enst.fr)

Version 2.0



# Définition d'un système temps réel

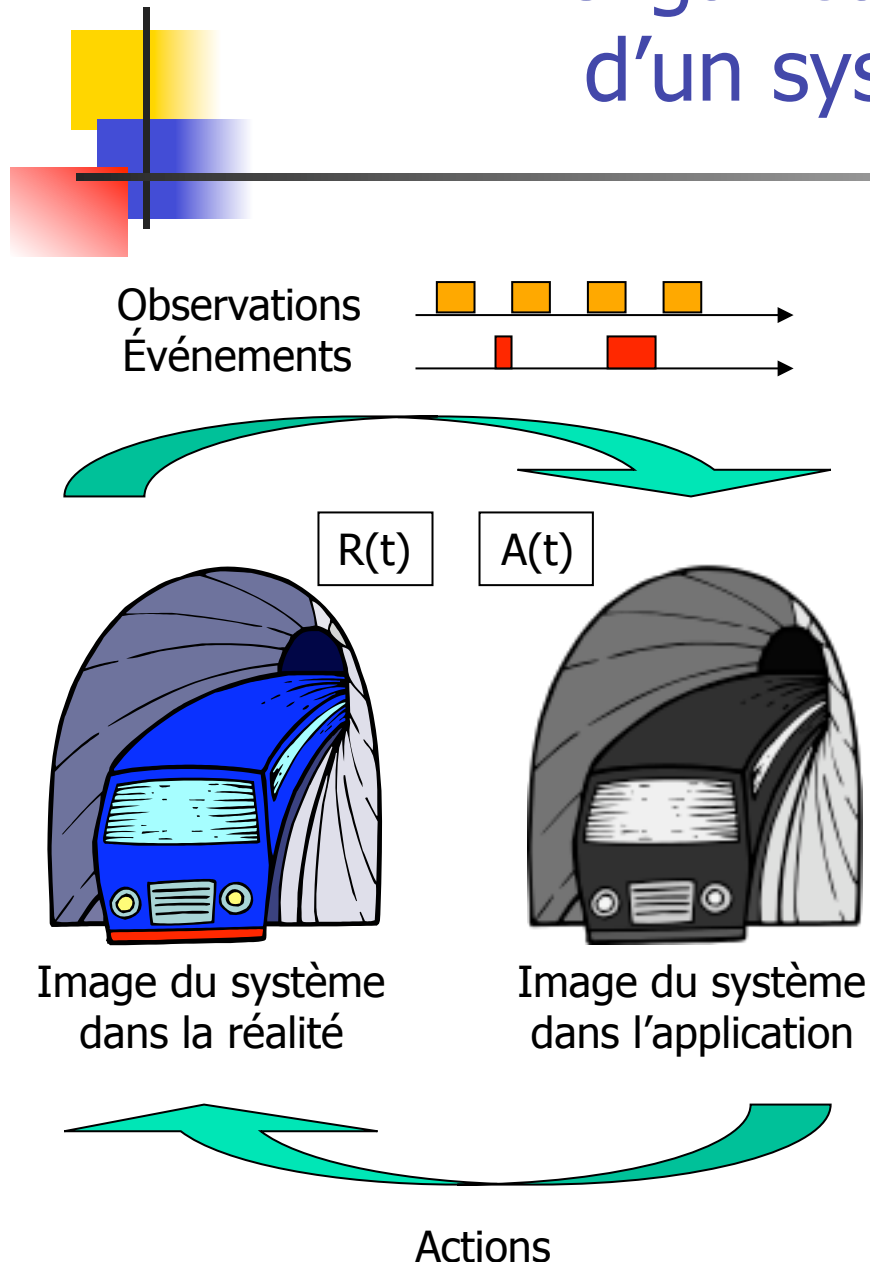
---

Un système temps réel se compose d'un ou plusieurs sous-systèmes devant répondre en un temps fini et spécifié à des stimuli générés par le monde extérieur

Un résultat hors échéance reste faux même s'il s'avère logiquement correct

Un contrôleur de vitesse pour voiture, une machine à laver, un robot d'entrepôt, un pilotage automatique de TGV, une centrale nucléaire, un système international de routage aérien, un système de gestion de salle de marché, etc.

# Organisation schématique d'un système temps réel



- L'application doit disposer d'une image précise et cohérente de la **réalité** au cours du **temps**
- L'objectif du temps réel vise donc à borner la différence entre l'image du système dans la réalité et celle dans l'application ( $|R(t)-A(t)| < \epsilon$ )
- Pour actualiser l'image dans l'application, celle-ci lit notamment des capteurs périodiquement (granularité temporelle pendant laquelle les mesures évoluent significativement)



# Contraintes du temps réel

---

- Concordance des images du système
  - Échelles de temps propres au système :
    - Avion (m-sec), Auto (sec), Bateau (min)
  - Régularité des scrutations et de leur propagation
    - Échantillonnage, périodes, giges, ...
  - Précisions logique et temporelle des calculs
    - Justesse des calculs, disponibilité dans fenêtre de temps
- Cohérence temporelle
  - Ensemble de mesures faiblement dispersées
- Réaction aux événements
  - Calcul des actions vs réaction à un événement extérieur
- Réaction aux pannes, aux surcharges, aux conflits
  - Importance # Urgence # Priorité

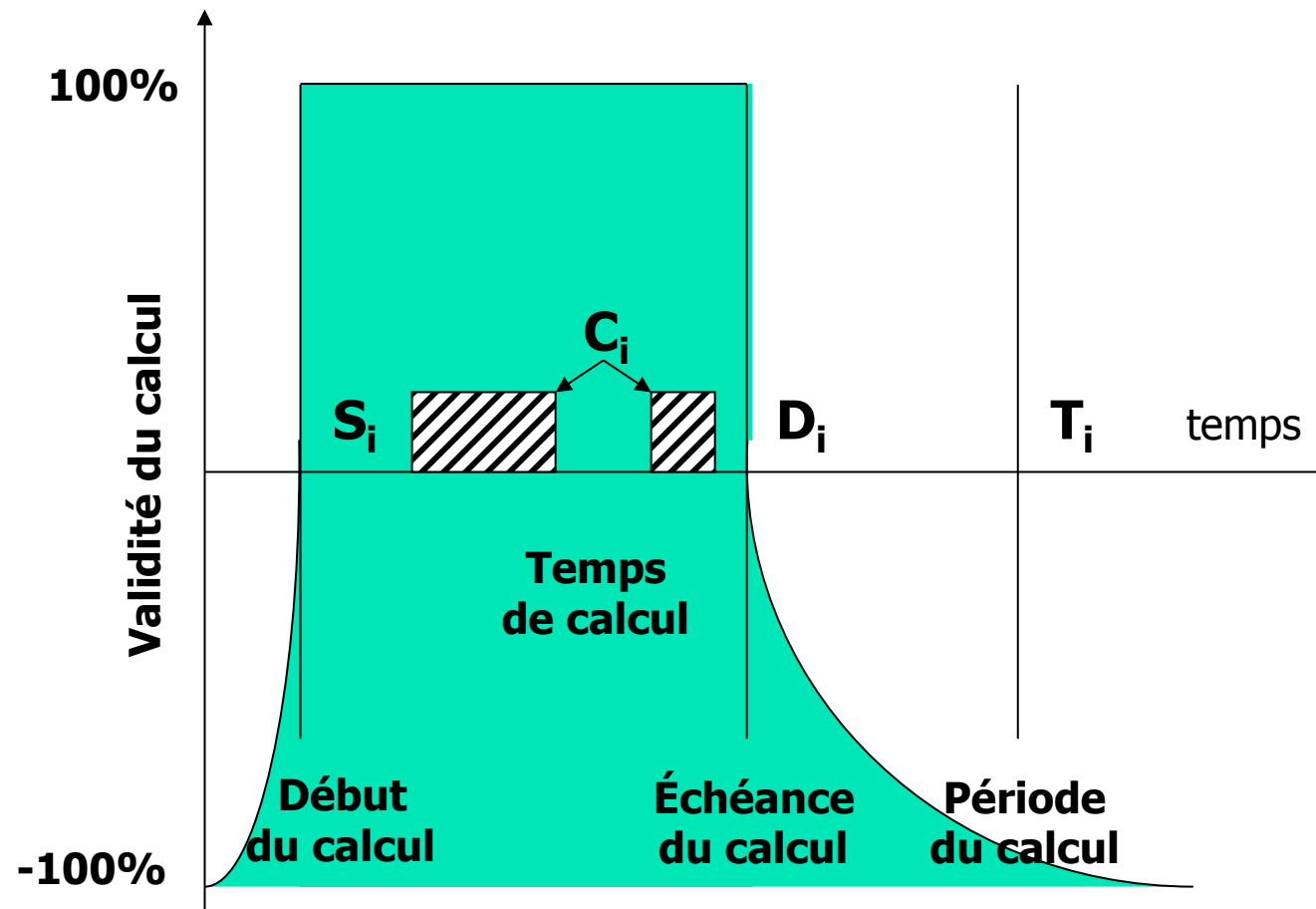


# Notations

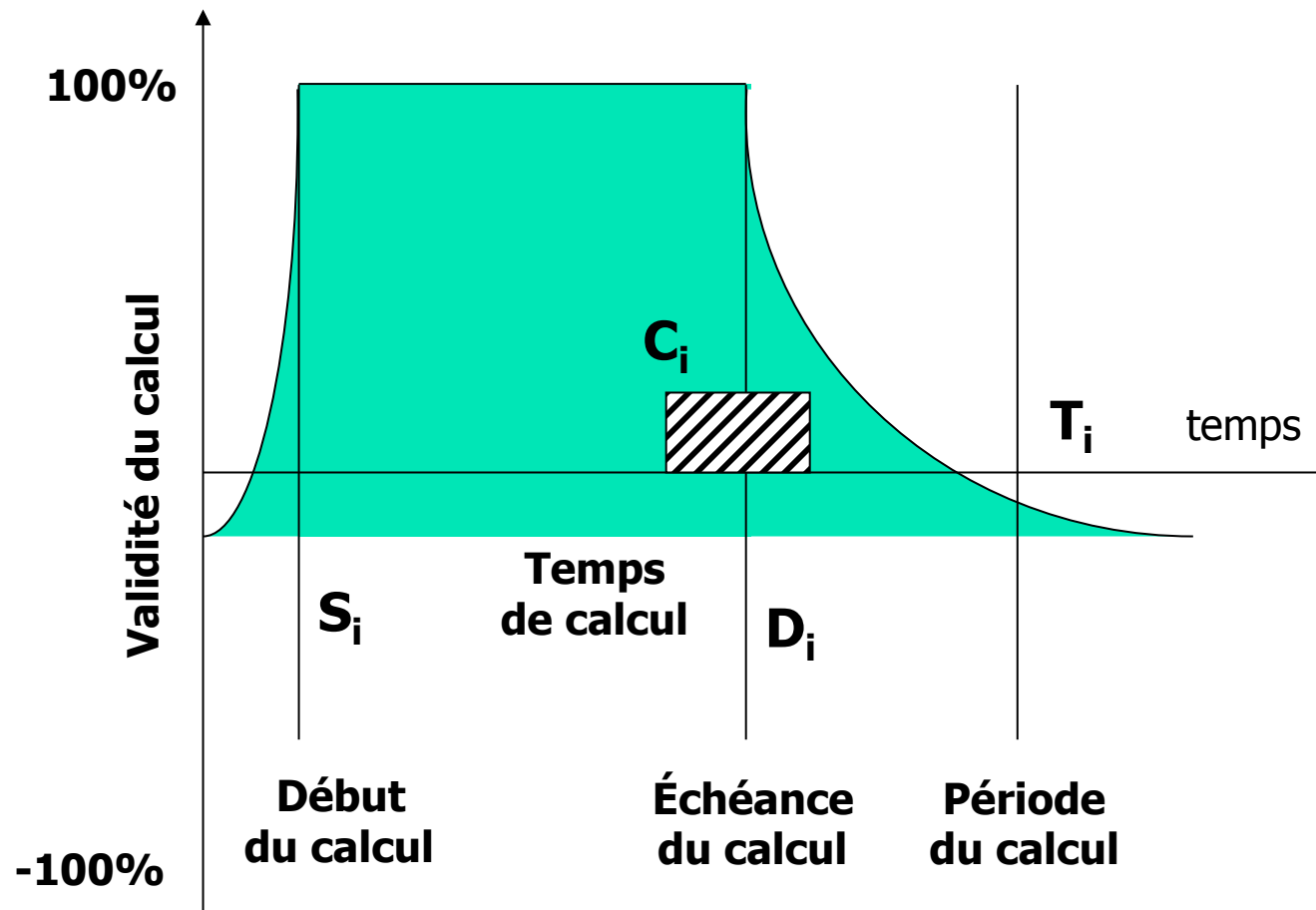
---

- Caractéristiques de la tâche  $t_i$ 
  - $C_i$  : durée de calcul de  $t_i$
  - $S_i$  : date de démarrage de la tâche  $t_i$ 
    - Avant l'heure ce n'est pas l'heure
  - $D_i$  : date d'échéance de  $t_i$ 
    - Après l'heure ce n'est plus l'heure
  - $T_i$  : période de  $t_i$
  - $U_i = C_i / T_i$  = utilisation du processeur pour  $t_i$
- $S_i + C_i < D_i$  mais ...
- $D_i < T_i$  n'est pas obligatoire
- $S_i$  peut être différent de 0 (problème d'écho)

# Hard Real-Time - Temps réel strict (*dur*)



# Soft Real-Time - Temps réel souple (*mou*)



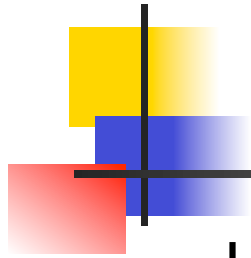


# Échec du respect de l'échéance

---

- Dans un système de temps réel strict, le non-respect de l'échéance ne peut survenir
  - Déterminisme maximal des opérations
    - WCET : *Worst Case Execution Time*
  - Réduction des points non-déterministes
    - Pré-allocation des ressources
    - Sur-dimensionnement du système
- Dans un système de temps réel souple, le non-respect de l'échéance est acceptable
  - Dans un certain pourcentage
  - Un certain nombre de fois
  - Avec une certaine fréquence
  - Peut aboutir à un traitement dégradé





# Sous-systèmes du système temps réel

---

Un système temps réel se compose de plusieurs sous-systèmes répondant à des critères de temps réel.

Cependant, tous les sous-systèmes d'un système temps réel ne répondent pas à des critères de temps réel !

- Les tâches temps réel critiques doivent absolument respecter les échéances prévues.
- Les tâches temps réel non-critiques peuvent répondre avec un certain retard sachant que la cohérence est dégradée.
- Les autres tâches répondront au mieux.

# Différentes catégories de systèmes exemples de l'aérospatial

Safety oriented system		<p>Life time in safety mode: 10 years              Interruption of service: 1 second              Error detection coverage: 100 %              Full segregation              Ground intervention not allowed</p>
Availability oriented system		<p>Limited life time: few hours or days              Interruption of service: 10 ms              Error detection coverage: 100%              No survival mode at system level              Ground intervention is not possible</p>
Reliability oriented system		<p>Long life time: 15 years in orbit              Interruption of service: 1 minute              Error detection coverage: 90%              Survival mode at system level              Ground intervention always possible</p>
Ground technology oriented system		<p>Medium life time: 3 years              Interruption of service allowed              Error detection coverage: 90%              Switch payload to safe state when error              Ground intervention always required</p>
Cost oriented system		<p>Medium life time: 3 years              Interruption of service allowed              Error detection coverage: 80%              Robust survival mode at system level              Ground intervention always required</p>



# Caractéristiques majeures

---

- Exécution prévisible
- Exécution déterministe
- Contrôle en temps réel
- Activités concurrentes
- Précision des données
- Sécurité des données
- Fiabilité des données
- Tolérance aux pannes
- Complexité de l'application
- Mise en œuvre efficace
- Placement à l'exécution
- Contraintes sur ressources
- Interaction avec le matériel
- Ordre entre actions
- Hiérarchie entre les tâches
- Projection priorité – criticité



# Modèles mathématiques

---

- Les problèmes d'ordonnancement de recherche opérationnelle (RO) sont différents de ceux de temps réel (TR), le but étant pour la RO de minimiser (hors ligne) le temps de réponse et pour le TR de satisfaire (en ligne) des échéances
- Ordonnancements de tâches périodiques comme
  - *Ordonnancement à base de table*
  - Ordonnancements préemptif à priorité fixe
  - Ordonnancements préemptif à priorité dynamique
- Ordonnancements de tâches apériodiques comme
  - Serveur sporadique
- Synchronisation des tâches comme
  - Héritage de priorité
  - *Priorité plafonnée*

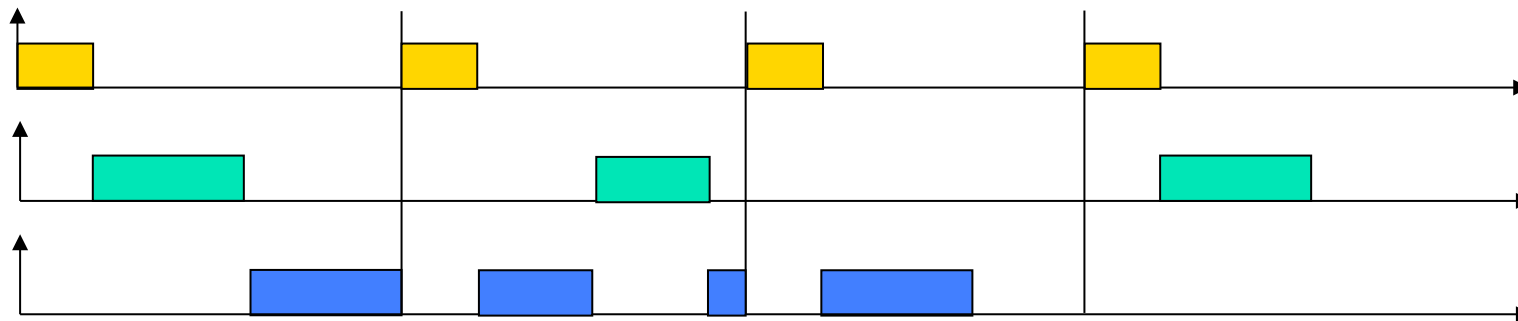
On restreint ici le problème d'ordonnancement  
en supposant  $D_i = T_i$ ,  $S_i = 0$  et tâches indépendantes

# Ordonnancement par priorité statique

## Rate Monotonic Scheduling

- Principe : une période courte (fréquence forte) indique une tâche évoluant vite et donc prioritaire
- La priorité est l'inverse de la période ( $P_i = 1/T_i$ )
- Condition suffisante :  $\sum U_i = \sum C_i/T_i \leq n (2^{1/n} - 1)$

	période	calcul	utilisation
$\tau_1$	10	2	0.200
$\tau_2$	15	4	0.267
$\tau_3$	36	12	0.333

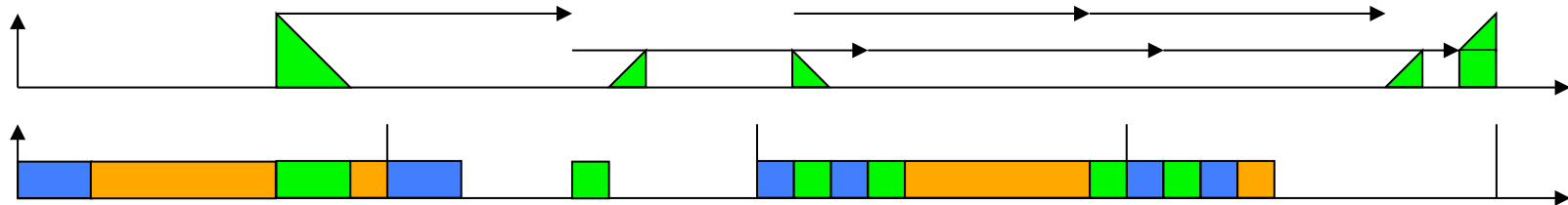


# Ordonnancement des tâches apériodiques

## Serveur sporadique

- Principe : Faire traiter par une tâche périodique de forte priorité les événements apériodiques tout en limitant le budget de temps de traitement alloué
- Le ticket de temps pris sur le budget pour traiter un événement est restitué après une période

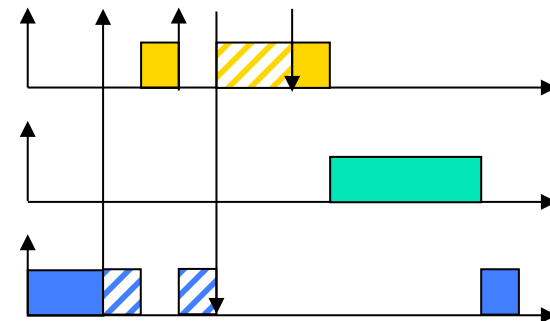
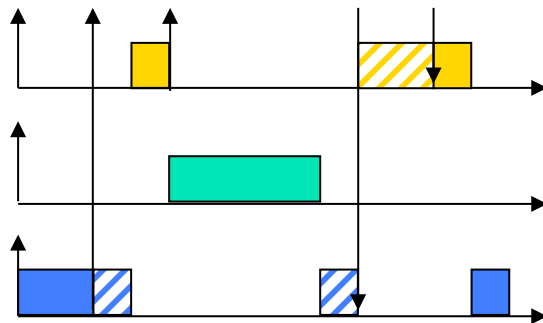
	période/activation	calcul	priorité	utilisation
événement $e_1$	7	3		
événement $e_2$	11	4		
tâche $t_1$	20	6	3	0,300
tâche $t_2$	10	2	2	0,200
serveur sporadique	8	2	1	0,250



# Synchronisation des tâches

## Push Through Blocking

- Si une tâche  $t_3$  de faible priorité dispose d'un verrou qu'une tâche  $t_1$  de forte priorité souhaite obtenir,  $t_3$  élève sa priorité à celle de  $t_1$  pendant l'exclusion mutuelle
- Sinon, une fois  $t_1$  bloquée, une tâche  $t_2$  de moyenne priorité pourrait obtenir le processeur provoquant une mise en attente infinie de  $t_1$  (*inversion de priorité*)





# Systèmes d'exécution

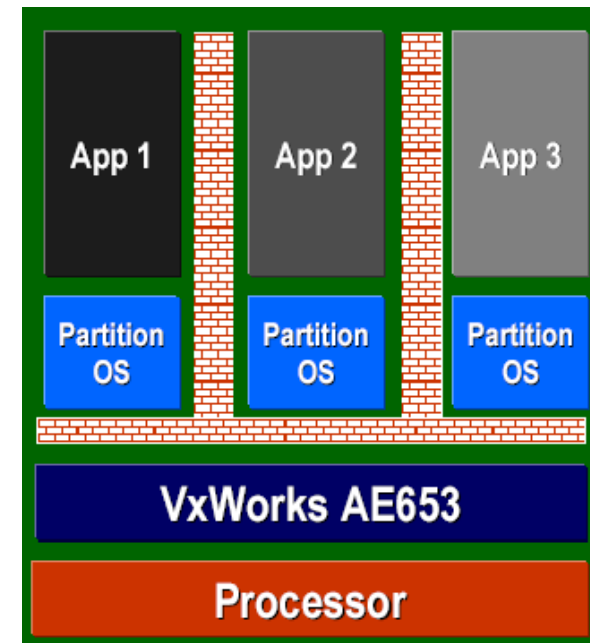
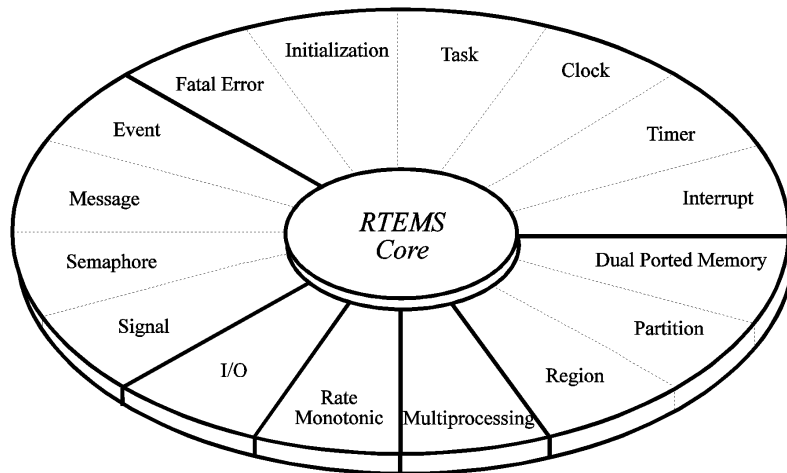
---

- Noyau préemptif réentrant
  - déterministe
  - « embarqué » et configurable
- Interfaçage matériel
  - Interruptions
  - Entrées-sorties
- Ordonnancement
  - Priorité statique (RMS)
  - Priorité dynamique (EDF)
- Services
  - Gestion du temps
  - Horloge temps réel
  - Synchronisation entre tâches
  - Partage de ressources
  - Serveur périodique de tâches apériodiques
  - Communication par messages
  - Base de données (mémoire)



# Exemples de système d'exécution

- VxWorks, VRTX, Lynx, RTEMS, QNX, ...
- Normalisés POSIX temps réel P1003.4
- Unices difficiles à adapter au temps réel (RT-Linux)
- Configuration classique
  - Une tâche, un processus léger
  - Pré-allocation des données (pas d'allocation dynamique)
  - Ordonnancement à priorité statique



Laurent Pautet



# Spécification, preuve et vérification

---

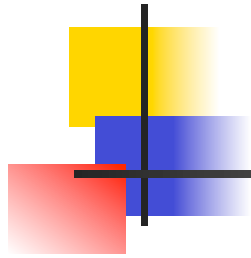
- La spécification, la preuve et la vérification du comportement temporel demeurent des défis fondamentaux pour le domaine du temps réel
- Il faut différencier prouver (*a priori*) et vérifier (*a posteriori*)
- Des progrès ont été faits en matière de preuves mais
  - Systèmes hybrides synchrones et asynchrones
  - Propriétés issues d'une composition de systèmes
  - Introduction de la répartition et de la communication
- Des progrès ont été faits en matière de vérification mais
  - Explosion combinatoire des systèmes réels
  - Introduction et expression des contraintes temporelles



# Outils et langages de conception

---

- Les outils de conception permettent de décrire le système de manière détaillée et à plusieurs niveaux :
  - niveau fonctionnel (contraintes de haut niveau)
  - niveau non-fonctionnel (description du comportement)
  - niveau implantation (description du matériel et du système)
- Un langage de description comme AADL ou UML-MARTE peut donner lieu à la production automatique de code dans un langage de programmation de bas niveau comme C ou de haut niveau comme Ada
- La conception peut aboutir à une preuve, à une simulation ou une vérification
- Cependant, une modification peut remettre en cause l'ensemble de ces résultats



# Modèles synchrone et asynchrone

---

- Dans le modèle synchrone, le temps de réaction à un stimulus est considéré comme nul ou négligeable
- Un stimulus n'arrive pas pendant la réaction en cours
- Dans le modèle asynchrone, il n'est pas nul et même variable en fonction du stimulus
- L'avantage du modèle synchrone vient des langages de description offerts, de sa simplicité de mise en œuvre et des ressources limitées qu'il requiert
- Or, en fonction du contexte, l'hypothèse du modèle synchrone peut ne pas être réaliste



# Langages réactifs

## Approche synchrone

---

- On isole ici la partie comportementale de celle fonctionnelle du système pour modéliser
- Plusieurs solutions ont vu le jour
  - Estérel (langage impératif)
  - Lustre, Signal (langages à flots de données)
  - Statecharts (approche graphique)
- Compilation de la description
- Mise en œuvre de preuve
- Génération d'automates ou de circuits

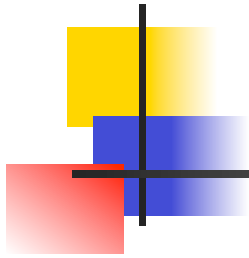


# Langages de programmation

## Approche asynchrone

---

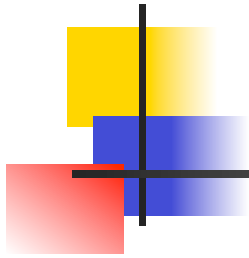
- Le langage de haut niveau :
  - spécification d'un projet de grande taille,
  - expression aisée des contraintes temporelles,
  - expression aisée des mécanismes de tolérance aux pannes,
  - encapsulation des détails d'implantation (compilateur),
  - connaissance minimale de l'architecture sous-jacente.
- Il propose des outils d'analyse :
  - temps d'exécution du pire cas
  - ressources nécessaires
  - vérification de règles spécifiques
- Exemple de langage de programmation
  - Ada (normalisé) et environnement (validé)
  - Ada et annexe pour le temps réel
  - Ada avec profile Ravenscar



# Bases de données

---

- La gestion de stock ou de production, les salles de marché, ... constituent des exemples de systèmes temps réel avec bases de données
- Il faut cependant s'affranchir ou redéfinir certains aspects :
  - Importance de la cohérence logique vs la cohérence temporelle
  - Remise en cause de la permanence des BD en temps réel
  - Pas d'atomicité et pas de traitement en séquence
  - Combinaison des processus temps réel et des triggers des bases de données



# Intelligence artificielle

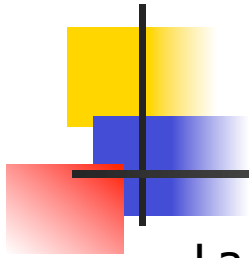
---

- La reconnaissance de la parole, l'analyse d'image ou l'aide à la décision constituent des exemples de systèmes temps réel avec intelligence artificielle
- De nombreux problèmes de temps réel sont NP complets ou nécessitent l'intervention d'un expert
- Cependant, les solutions proposées ne prennent que peu en compte les préoccupations du temps réel. Il faut intégrer le fait que :
  - la récursivité et le *backtracking* sont bornés
  - l'utilisation mémoire est maîtrisée (sans ramasse-miettes)
  - les contraintes de temporelle sont satisfaites (convergence)
  - la préemption d'un calcul est possible (concurrency)
  - les temps d'exécution sont respectés (fonction créée)



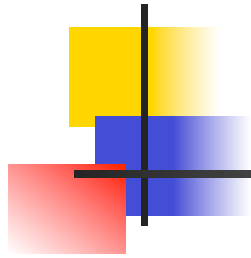
# Tolérance aux pannes

## généraux byzantins



La tolérance aux pannes peut être exprimée de façon abstraite en terme de généraux byzantins qui campent autour d'une cité ennemie. Ne communiquant qu'à l'aide de messagers, ceux-ci doivent se mettre d'accord sur un plan de bataille, sinon la défaite est assurée. Or un ou plusieurs de ces généraux peuvent être des traîtres, qui sèment la confusion parmi les autres. Il s'agit de trouver un algorithme pour s'assurer que les généraux loyaux arrivent tout de même à se mettre d'accord sur un plan de bataille.

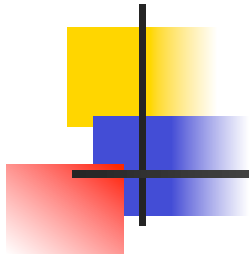
- On ne peut pas trouver de solution au problème de consensus sous hypothèse de panne pour un modèle de système asynchrone avec un seul fautif
- Avec des propriétés sur les messages, le vote majoritaire fonctionne si le nombre de généraux est strictement 3 fois supérieur aux nombre de traîtres



# Tolérance aux pannes

---

- La tolérance aux pannes doit être intégrée dès la conception dans le matériel comme dans le logiciel tout en prenant en compte les contraintes temporelles.
- La tolérance aux pannes accompagnée de reconfiguration dynamique a techniquement peu progressé et la prise en compte du temps dans ces systèmes demeure difficile.
- Les outils fournis par le système d'exécution sont déterminants notamment en ce qui concerne l'implantation de points de reprises, de duplication matérielle et logicielle et de techniques de recouvrement après fautes.



# Architecture et matériel

---

- La conception d'un système temps réel associe le logiciel et le matériel, un composant remplissant une fonction donnée pouvant devenir soit l'un soit l'autre
- Une fausse croyance pousse à tout intégrer dans le matériel mais le non-déterminisme existe aussi dans le matériel
- Changer l'application ne doit plus impliquer de changer le matériel (qui devient ainsi banalisé)
- Les évolutions du matériel doivent porter sur l'intégration de nouvelles fonctions notamment au travers des system-on-chip (validés) ou des mécanismes de communication nouveaux



# Réseaux et communication

---

- Évaluer le pire cas requiert des réseaux spécifiques
  - Réseau à jeton (anneau logique, FDDI)
  - Réseau synchrone (CAN synchrone)
  - Réseau à protocole spécifique
- Il faut savoir évaluer le pire cas pour
  - délais entre systèmes d'exécution et de communication
  - délais d'attente avant émission et réception
  - délais de transmission dans le système de communication
- Il faut des intergiciels spécifiques qui généralisent les propriétés des systèmes d'exécution à une plate-forme intégrant la répartition, le réseau et la communication