Introduction aux Systèmes embarqués temps-réel

Laurent Pautet

Laurent.Pautet@enst.fr

Version 2.0

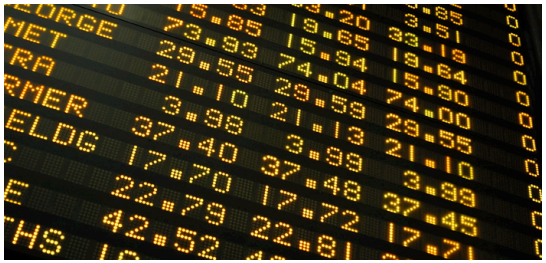
Exemples d'applications temps réel

Commandes de vol numériques

Codage des communications

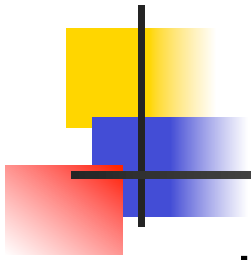
Métro automatisé

Syst. Informatique de la bourse de Paris



?

Quel est le lien



Exigences temporelles

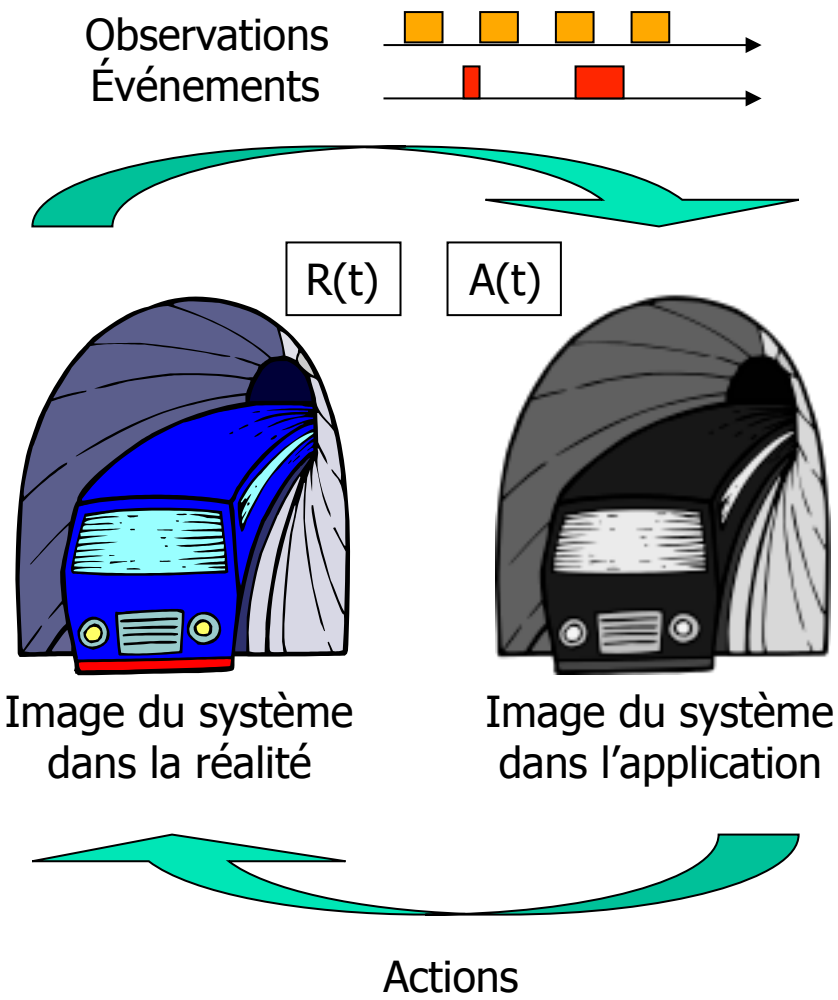
Un système temps réel se compose d'un ou plusieurs sous-systèmes devant répondre en un temps fini et spécifié à des stimuli générés par le monde extérieur

Une réponse hors échéance est invalide

Même si son contenu semble 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.

But des contraintes temporelles



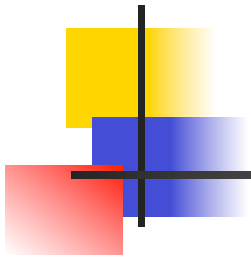
- L' application doit disposer d'une image précise et cohérente de la **réalité** au cours du **temps**
- L'objectif du temps réel vise donc à borner la différence entre l'image du système dans la réalité et celle dans l'application ($|R(t)-A(t)|<\varepsilon$)
- Pour actualiser l'image dans l'application,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

Contraintes Non Fonctionnelles

PB : Ces systèmes doivent être prévisibles

- Réactivité & cohérence temporelle
  - Définir des fenêtres temporelles pour la validité des données
  - Respecter l'échelle de temps du système -> ni trop vite, ni trop lent (avion = msec, voiture = sec)
  - Assurer des traitements exécutés en un temps borné (connu)
- Fiabilité & Disponibilité
  - Assurer l'exactitude « numérique » du résultat des traitements
  - Assurer la continuité de la fonction en cas de conditions adverses (tolérance aux pannes, aux malveillances ...)

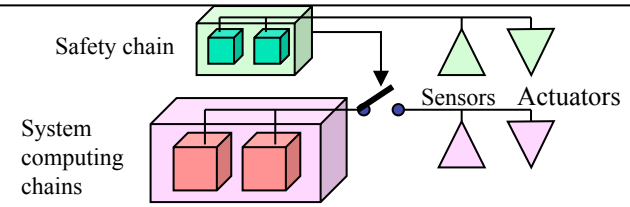
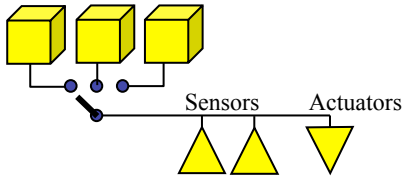
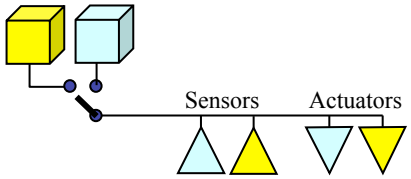
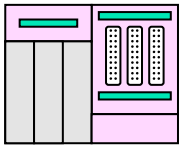
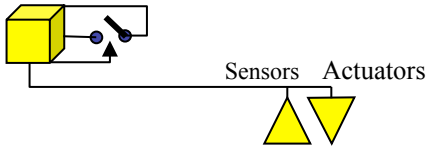
Contraintes non fonctionnelles -> temporelles & structurelles



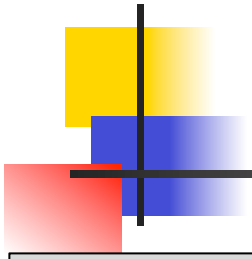
Des besoins applicatifs vers l'ingénierie

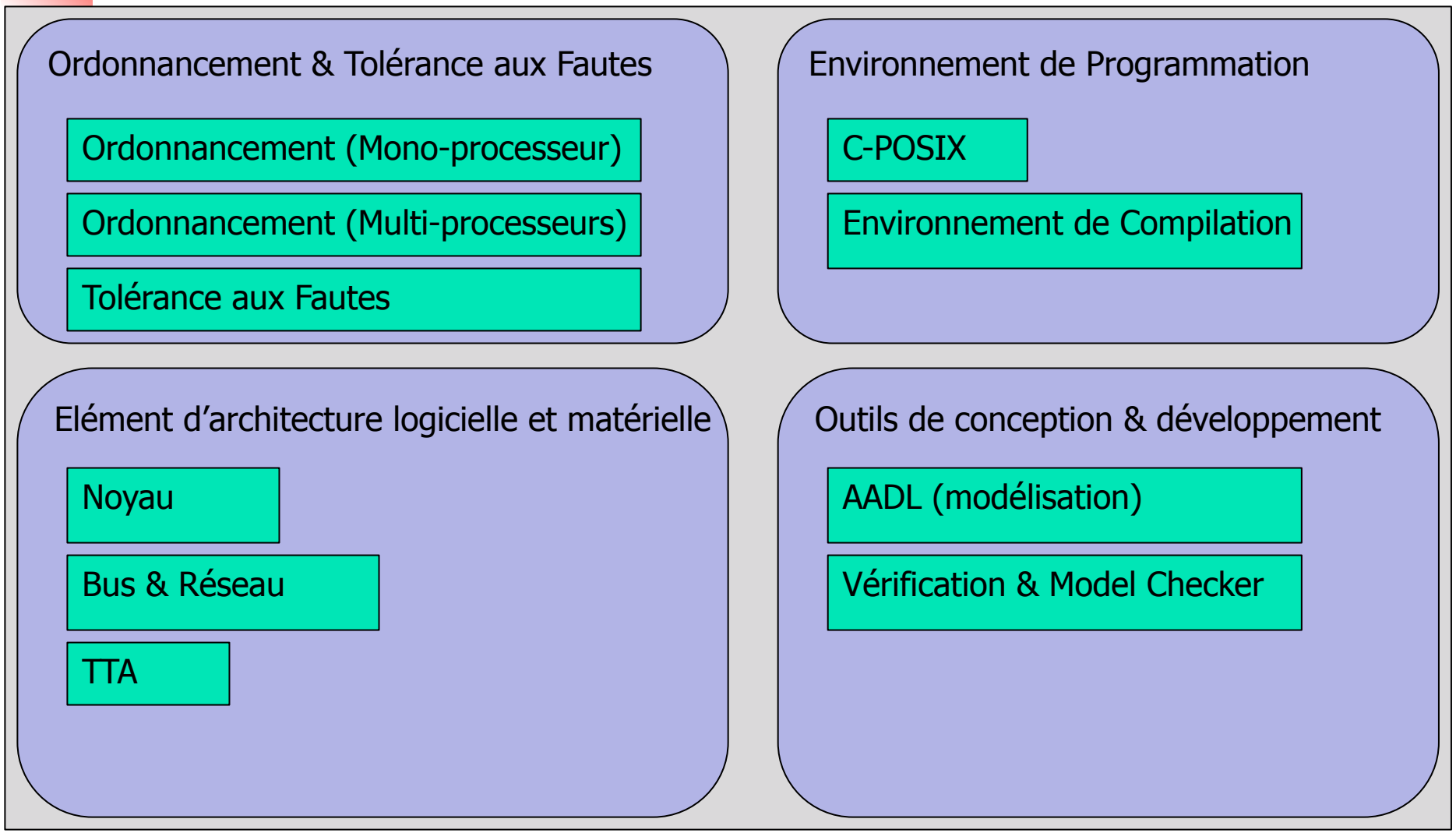
- Besoins
  - Réactivité & cohérence temporelle
  - Fiabilité & disponibilité
- Les moyens
  - Des architectures et des services pour aider la conception (Noyaux, intergiciels, Bus et réseaux, TTA, BDTR)
  - Des modèles et méthodes pour être prévisible (Ordonnancement, Tolérance aux Fautes)
  - Des langages de programmation adaptés (Java->Java-RT, Standard POSIX 1003.1c et , Run-time RT Ada)
  - Des outils pour assurer la continuité entre la conception et la réalisation (AADL, Marte, Env. développement, modèles et vérification)

Différentes architectures de systèmes
exemples de l'aérospatial

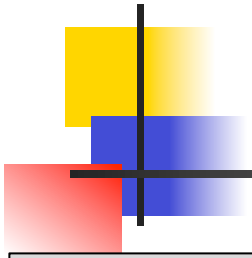
Safety oriented system		Life time in safety mode: 10 years Interruption of service: 1 second Error detection coverage: 100 % Full segregation Ground intervention not allowed
Availability oriented system		Limited life time: few hours or days Interruption of service: 10 ms Error detection coverage: 100%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

Contenu du cours





Contenu du cours



Ordonnancement & Tolérance aux Fautes

- Ordonnancement (Mono-processeur)
- Ordonnancement (Multi-processeurs)
- Tolérance aux Fautes

Environnement de Programmation

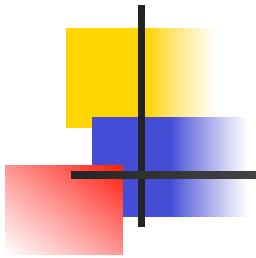
- C-POSIX
- Environnement de Compilation

Elément d'architecture logicielle et matérielle

- Noyau
- Bus & Réseau
- TTA

Outils de conception & développement

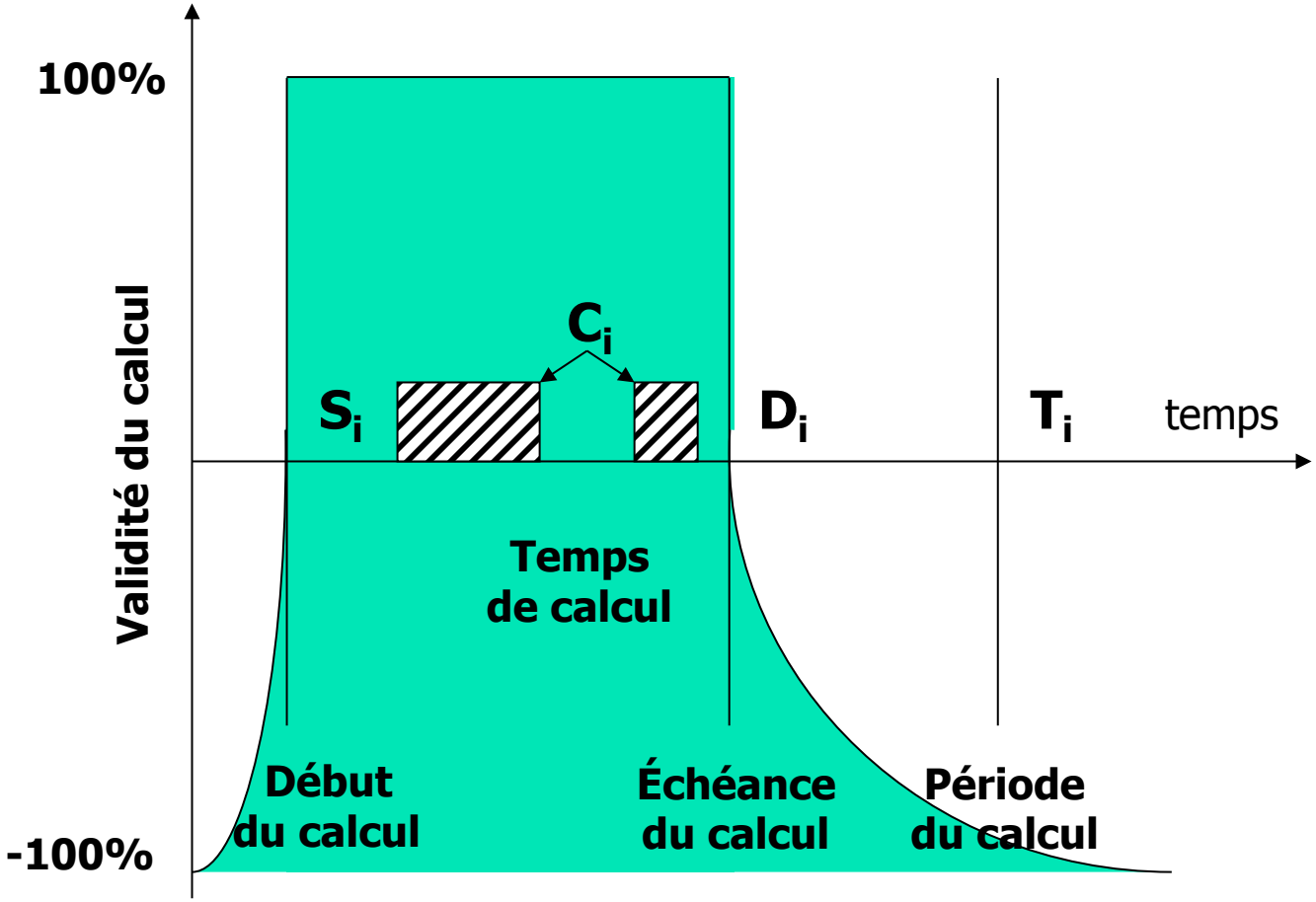
- AADL (modélisation)
- Vérification & Model Checker



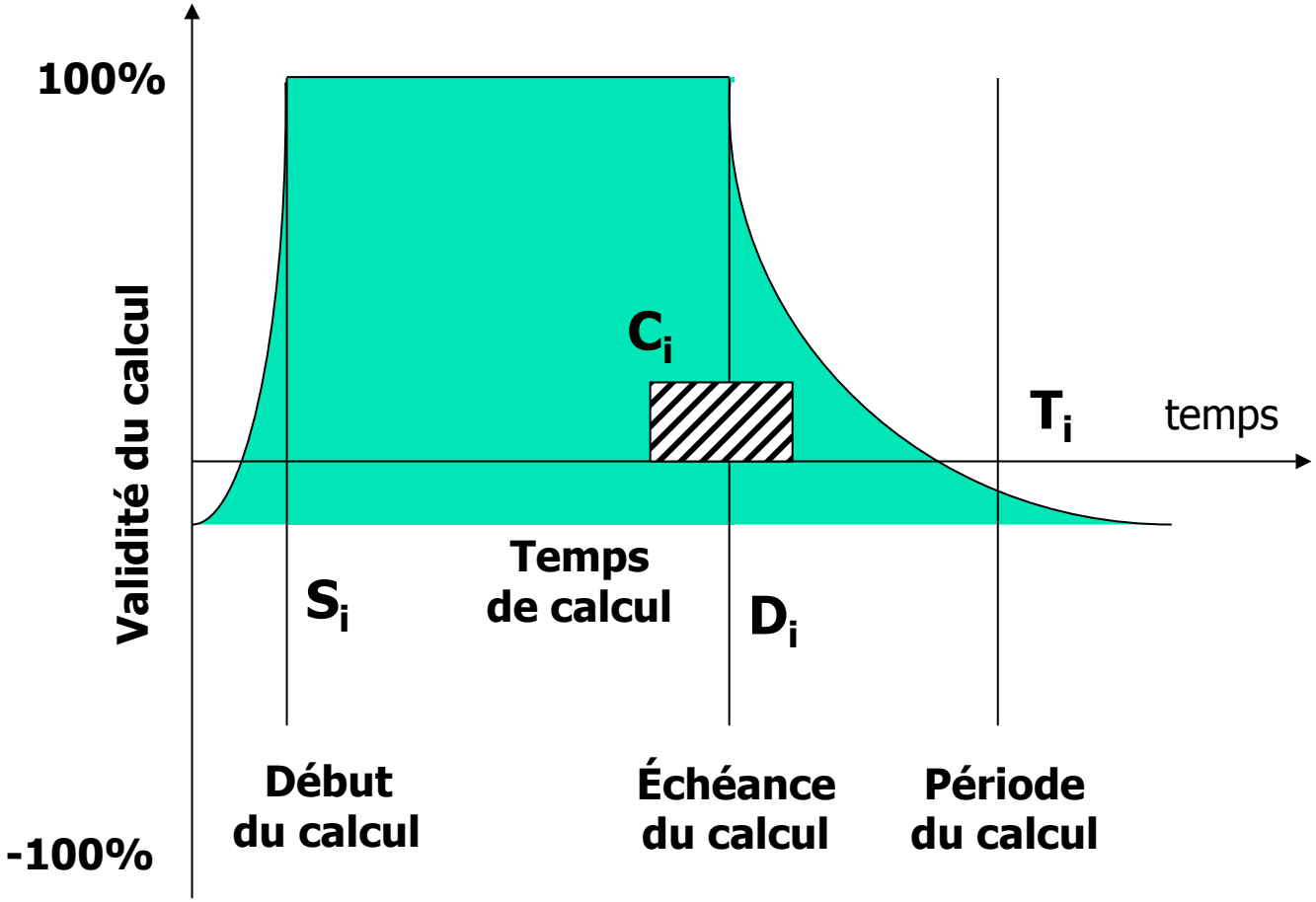
Les notations pour l'ordonnancement

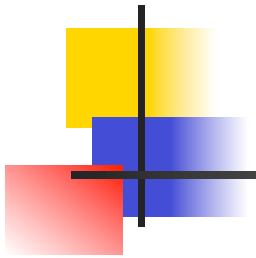
- Caractéristiques de la tâche t_i
  - C_i : durée de calcul de t_i
  - S_i : date de démarrage de la tâche t_i
    - Avant l'heure ce n'est pas l'heure
  - D_i : date d'échéance de t_i
    - Après l'heure ce n'est plus l'heure
  - T_i : période de t_i
  - $U_i = C_i / T_i$ = utilisation du processeur 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 (dépendances)

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



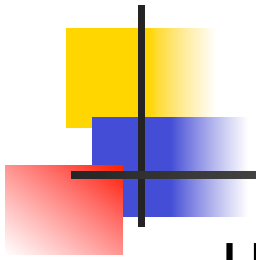
Soft Real-Time - Temps réel souple *(mou)*





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

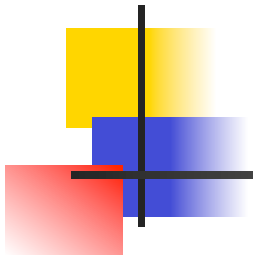


Sous-systèmes du système temps réel

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.



Modèles mathématiques

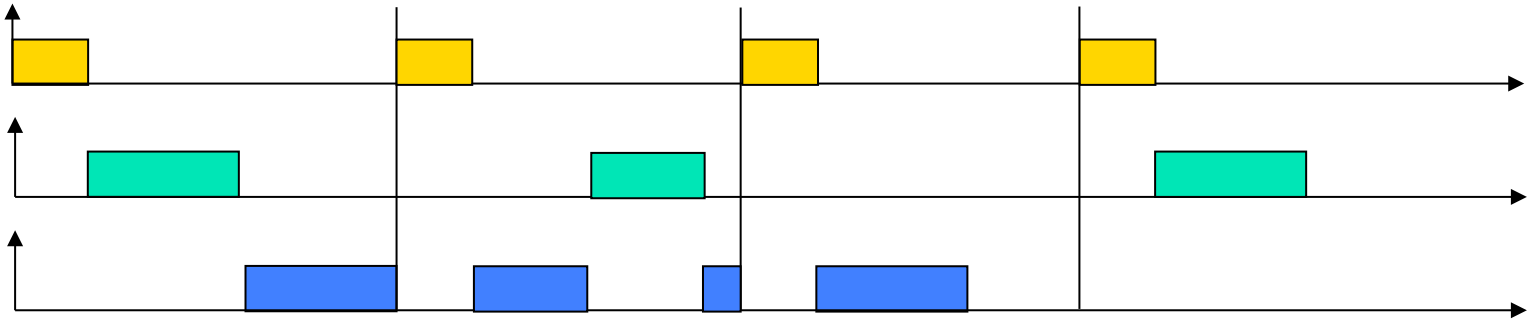
- Les problèmes d'ordonnancement de recherche opérationnelle (RO) sont différents de ceux de temps réel (TR), le but étant pour la RO de minimiser (hors ligne) le temps de réponse et pour le TR de satisfaire (en ligne) des échéances
- Ordonnancements de tâches périodiques comme
  - *Ordonnancement à 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

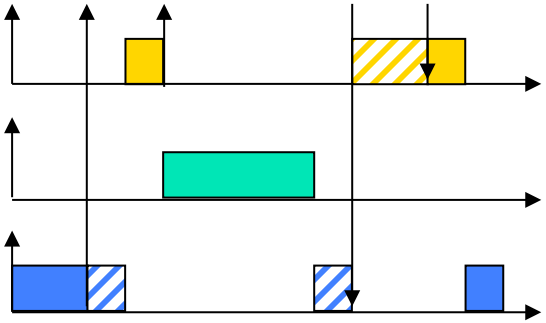
- Principe : une période courte (fréquence forte) indique une tâche évoluant vite et donc prioritaire
- La priorité est l'inverse de la période ($P_i = 1/T_i$)
- Condition suffisante : $\sum U_i = \sum C_i / T_i <= n\,(2^{1/n} - 1)$

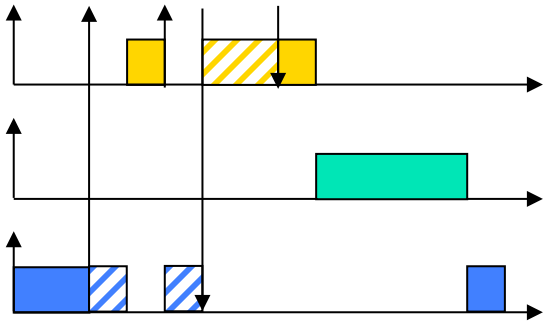
	période	calcul	utilisation
T_1	10	2	0.200
T_2	15	4	0.267
T_3	36	12	0.333



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*)



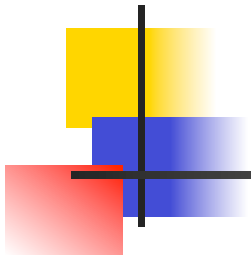


Tolérance aux fautes
principe

- Approche zéro défaut :
  Tout marche comme prévu - les WCET sont respectés, les programmes ne bouclent pas…
- Tolérance aux fautes :
  le pire peu arriver mais doit être maitrisé
  - Identification d'un ensemble de conditions adverses (la foudre brule un capteur…)
  - Altération de l'architecture du système pour « limiter la casse »

Les deux approches sont complémentaires en pratique

Tolérance aux fautes principes et pratiques



- La tolérance aux fautes (TaF) affecte fortement l'implémentation du système
  => à prendre en compte le + tôt possible

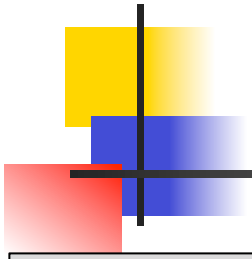
- Les principes :
  - Deux suretés valent mieux qu'une (redondance)
  - Un tiens vaut mieux que deux tu l'auras (mode dégradé)

- En pratique :
  - Architectures distribuées pour résister aux pannes

  (architecture à 3 calculateurs utilisée dans les avions)
  - Méthodes de codage et de génie logiciel pour confiner les comportements illicites

  (code CRC sur les CD, codes de retours d'erreurs en C)

Contenu du cours



Ordonnancement & Tolérance aux Fautes

- Ordonnancement (Mono-processeur)
- Ordonnancement (Multi-processeurs)
- Tolérance aux Fautes

Environnement de Programmation

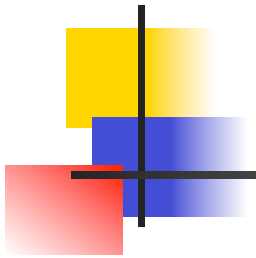
- C-POSIX
- Environnement de Compilation

Elément d'architecture logicielle et matérielle

- Noyau
- Bus & Réseau
- TTA

Outils de conception & développement

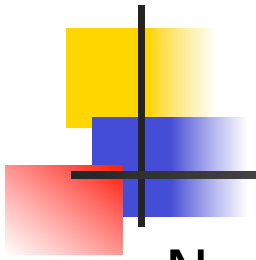
- AADL (modélisation)
- Vérification & Model Checker



Architecture du 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

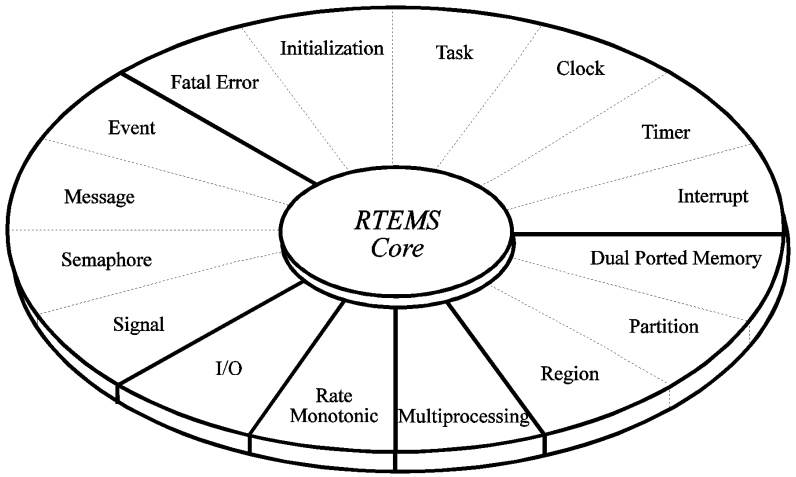
Support d'exécution

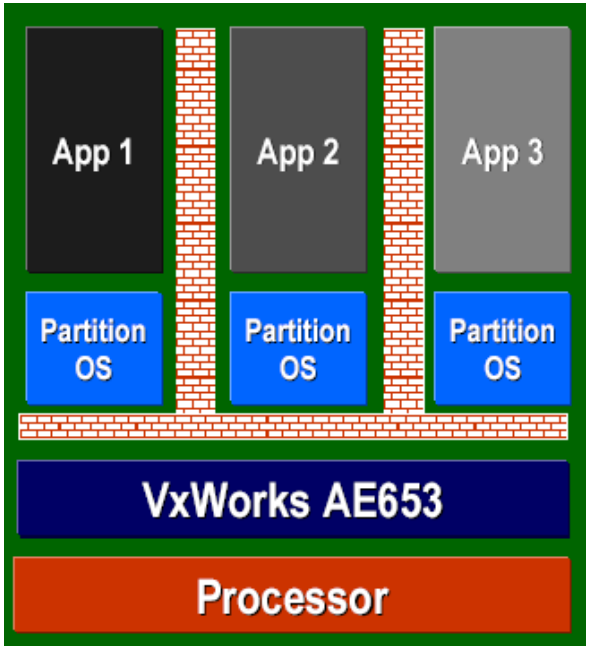


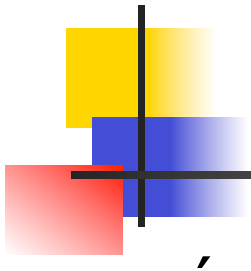
- Noyau préemptif réentrant
  - déterministe
  - « embarqué » et configurable
- Interfaçage matériel
  - Interruptions
  - Entrées-sorties
- Ordonnancement
  - Priorité statique (RMS)
  - Priorité dynamique (EDF)
- Services
  - Gestion du temps
  - Horloge temps réel
  - Synchronisation entre tâches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

Exemples de supports 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



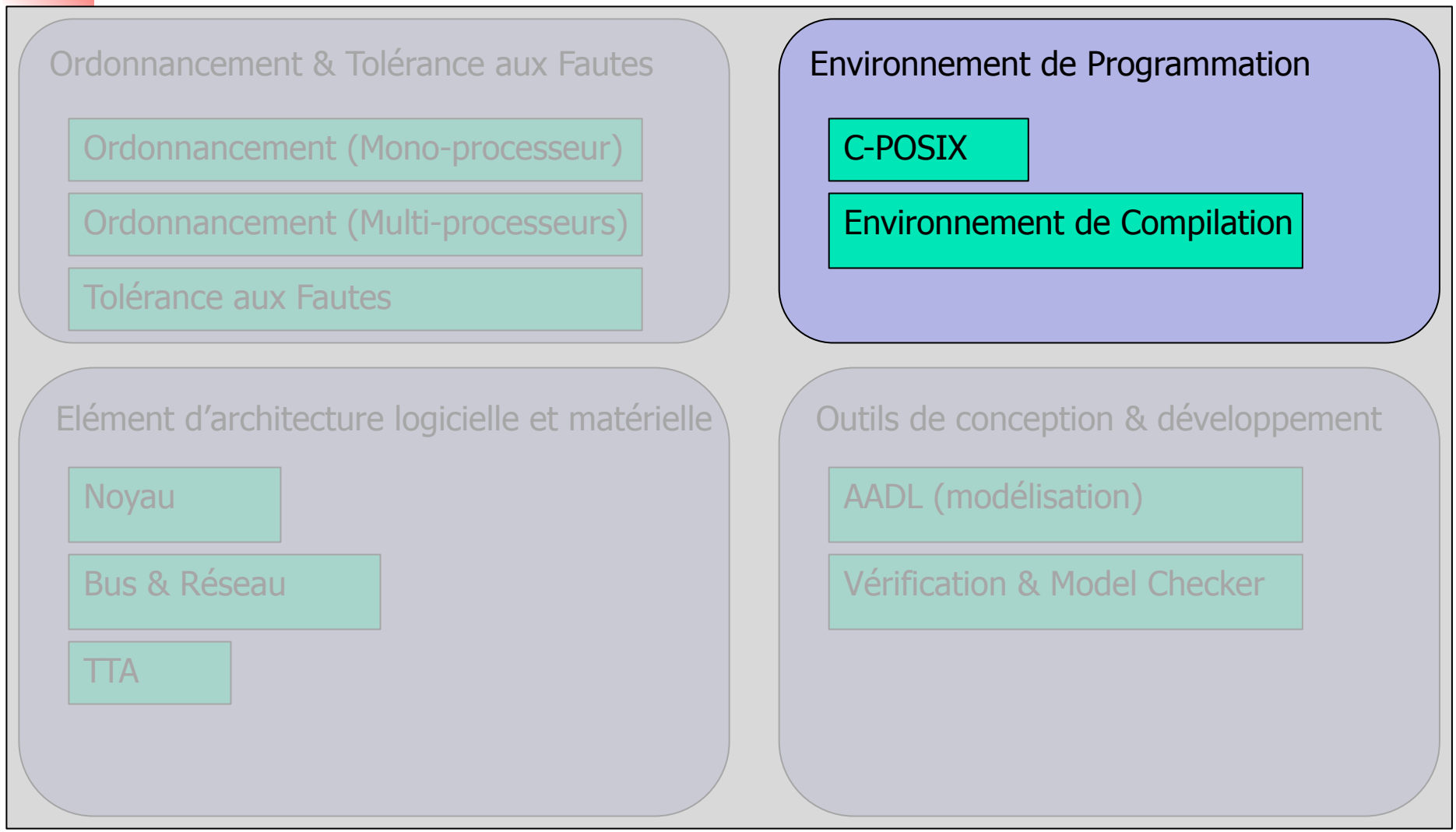


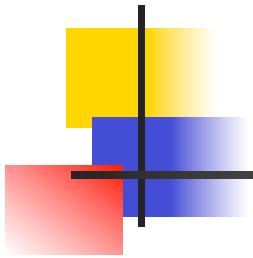


Réseaux et communication

- 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

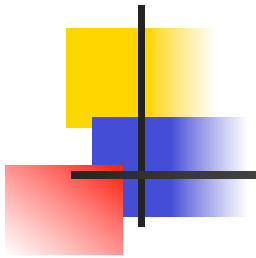
Contenu du cours





Programmation Temps réel

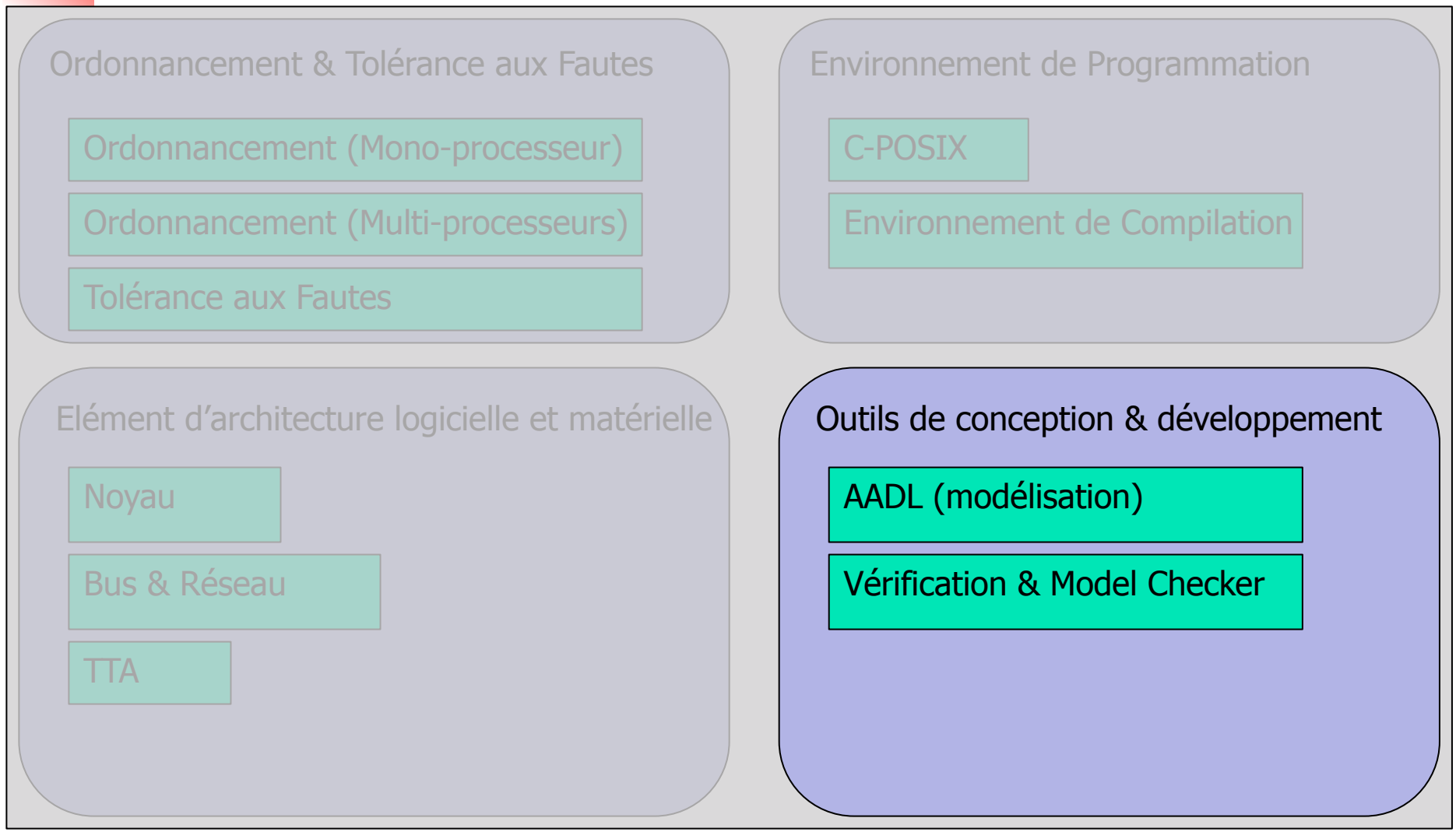
- But : programmer le déterminisme
  - Ordonnancement
  - Contrôle des ressources
- Moyens :
  - Les threads POSIX 1003.4b / Java / fsu (Ada).
  - les motifs de programmation pour RMS (ex. sous RTEMS)
  - Les fonctions de synchronisation adaptées au temps réel
  - La gestion de la mémoire pour assurer le déterminisme (Java RT)

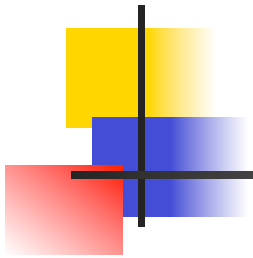


Environnement de Développement

- Des outils spécifiques au temps réel (TR) :
  - Run-time temps réels (Ada, JVM Temps Réel)
  - Evaluation du WCET (Bound-T)
- Des outils spécifiques aux systèmes embarqués :
  - Compilation et édition de lien « croisées ».
  - Interface de contrôle / observation de la cible
- Une application TR est souvent embarquée =>
  - Adapter la runtime à la cible
  - Intégrer la description du matériel dans l'évaluation du WCET

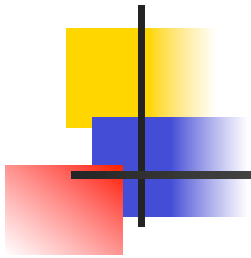
Contenu du cours





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 nécessiter des preuves, ou une validation par simulation ou vérification
- Cependant, une modification peut remettre en cause l'ensemble de ces résultats