Systèmes embarqués/ Embedded Systems

2023/2024

Chapitre 1: Système embarqués

2023/2024

I. Définitions

1. Un système embarqué (SE) est un système électronique dont lequel on embarque de l'informatique. Il peut être une partie intégrante d'un système plus large ou une machine. Tous les systèmes qui ont des interfaces digitales (i.e. montre, caméra, voiture…) peuvent être considérés comme des SE.

- Certains SE ont un OS
- D'autres sans OS, pour ces derniers toute leur logique peut être implantée en un seul programme.

2. Un système embarqué est une combinaison de logiciel et matériel, avec des capacités fixes ou programmables, qui est spécialement conçu pour un type d'application.

Les distributeurs automatiques de boissons, les automobiles, les équipements médicaux, les caméras, les avions, les jouets, les téléphones portables et les PDA sont des exemples de systèmes qui abritent des SE.

3. Un système embarqué peut être une composante primordiale d'un système complexe (i.e. un avion, une voiture, Train, drone, etc…) dont l'objectif est de commander, contrôler et superviser ce système.

'Embedded System' : Système Embarqué

I. Définitions

❖ Other definition

An embedded system is a combination of computer hardware and software designed for a specific function. Embedded systems may also function within a larger system. The systems can be programmable or have a fixed functionality. Industrial machines, consumer electronics, agricultural and processing industry devices, automobiles, medical equipment, cameras, digital watches, household appliances, airplanes, vending machines and toys, as well as mobile devices, are possible locations for an embedded system.

While embedded systems are computing systems, they can range from having no user interface (UI) -- for example, on devices designed to perform a single task -- to complex graphical user interfaces (GUIs), such as in mobile devices. User interfaces can include buttons, LEDs (light-emitting diodes) and touchscreen sensing. Some systems use remote user interfaces as well

II. Domaines d'application des systèmes embarqués

❑ Domaines « traditionnels »

- Avionique
- Robotique
- Automobile
- Militaire

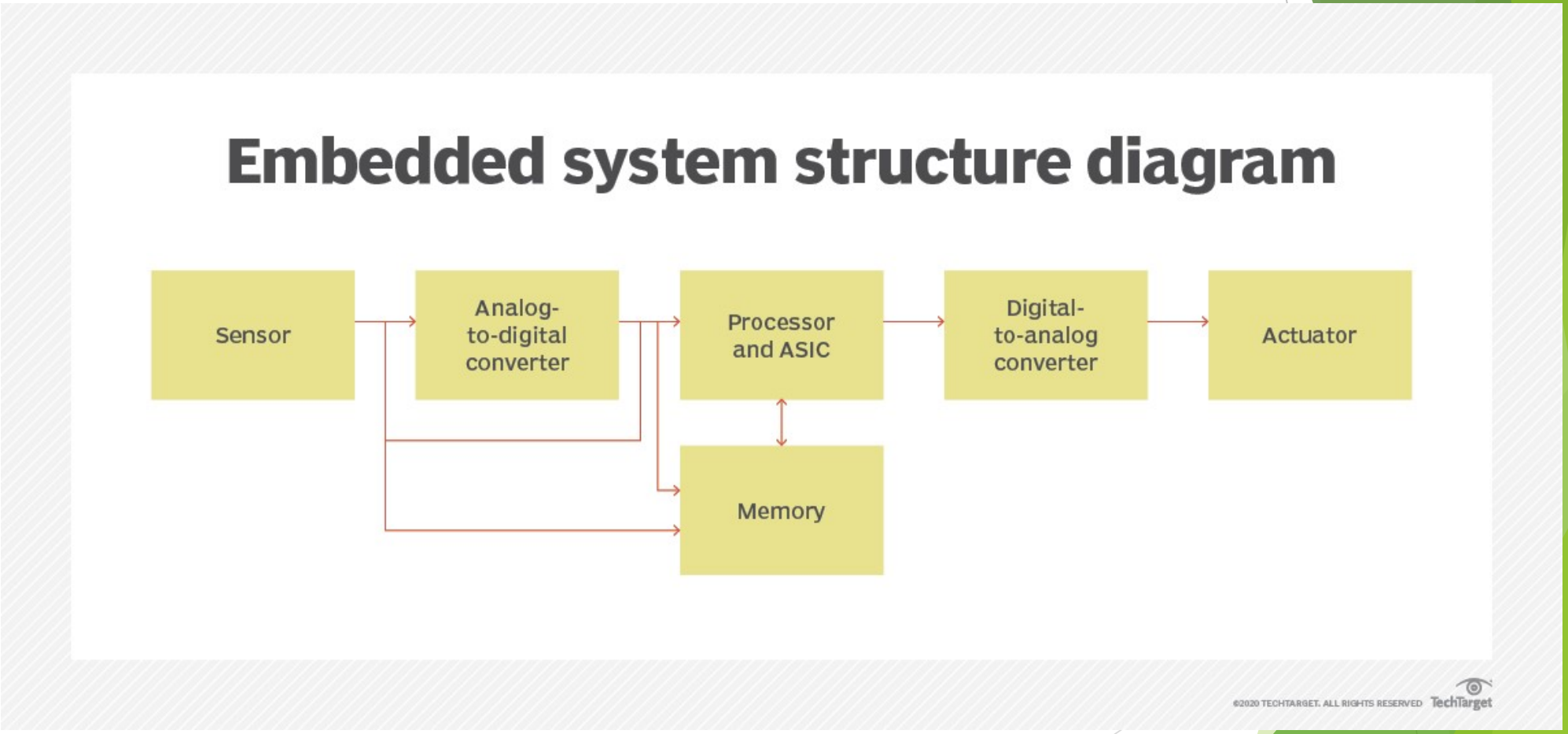
❑ Domaines « nouveaux »

- Jeux et loisirs
- Téléphonie, Internet mobile
- Implants (santé sécurité)
- Domotique,
- immeubles intelligents,
- Villes intelligentes
- Vêtements…

Irone est capable de corriger sa
ion, de se **stabiliser** en restant en vol
nnaire. Il **exécute** aussi en temps réel
dres envoyés par la radiocommande.

La **voiture sans conducteur** est capabl
de transporter des usagers su
n'importe quelle route en tout
autonomie et sécurité.

III. Architecture de base d´un système embarqué



IV. Les types des systèmes embarqués

❑ Types selon leurs exigences fonctionnel

Il existe quelques types de systèmes embarqués de base, qui diffèrent par leurs exigences fonctionnelles. Ils sont:

- **Les systèmes embarqués mobiles / Mobile embedded systems** :sont des systèmes de petite taille conçus pour être portables. Les appareils photo numériques en sont un exemple.

- **Les systèmes embarqués en réseau / Networked embedded systems** : sont connectés à un réseau pour fournir une sortie à d'autres systèmes. Les exemples incluent les systèmes de sécurité domestique et les systèmes de point de vente (POS).

- **Les systèmes embarqués autonomes / Standalone embedded systems** : ne dépendent pas d'un système hôte. Comme tout système embarqué, ils effectuent une tâche spécialisée. Cependant, ils n'appartiennent pas nécessairement à un système hôte, contrairement aux autres systèmes embarqués. Une calculatrice ou un lecteur MP3 en sont un exemple.

- **Les systèmes embarqués en temps réel / Real Time embedded systems**: fournissent le résultat requis dans un intervalle de temps défini. Ils sont souvent utilisés dans les secteurs médical, industriel et militaire car ils sont responsables de tâches urgentes. Un système de contrôle du trafic en est un exemple.

IV. Les types des systèmes embarqués

- Types selon leurs performances

Les systèmes embarqués peuvent également être classés selon leurs exigences de performances :

- Les systèmes embarqués à petite échelle n'utilisent souvent qu'un microcontrôleur 8 bits.

- Les systèmes embarqués à moyenne échelle utilisent un microcontrôleur plus grand (16-32 bits) et relient souvent les microcontrôleurs entre eux.

- Les systèmes embarqués sophistiqués utilisent souvent plusieurs algorithmes qui entraînent des complexités logicielles et matérielles et peuvent nécessiter des logiciels plus complexes, un processeur configurable et/ou un réseau logique programmable.

IV. Les types des systèmes embarqués

❑ Types selon leurs architecture logiciel

Il existe plusieurs architectures logicielles de systèmes embarqués courantes, qui deviennent nécessaires à mesure que les systèmes embarqués se développent et deviennent plus complexes. Ceux-ci inclus:

- **Les boucles de contrôle simples / simple control loop** : appellent des sous-programmes qui gèrent une partie spécifique du matériel ou de la programmation embarquée.

- **Les systèmes contrôlés par interruption / Interrupt controlled systems :** ont deux boucles : une principale et une secondaire. Les interruptions dans les boucles déclenchent des tâches.

- **Le multitâche préemptif ou le multithreading / Preemptive multitasking or multithreading** : est souvent utilisé avec un *RTOS /Real Time Operating System* et propose des stratégies de synchronisation et de changement de tâches.

VI. Principales caractéristiques des systèmes embarqués

❑ Fonctionnalité unique

La plupart des systèmes embarqués ont des fonctionnalités uniques. Le système présente à plusieurs reprises le comportement programmé. La plupart des appareils électroniques grand public comme une horloge, un chronomètre, un simple dispositif d'alarme, un réfrigérateur et une machine à laver en sont des exemples classiques. Ils exécutent tous des fonctionnalités préprogrammées.

❑ Exigences axées sur l'utilisateur

Le système se caractérise par une conception très contraignante. Les exigences de l'utilisateur peuvent ne pas être techniques mais il peut y avoir plusieurs contraintes. Les contraintes continueront d'évoluer d'un type d'appareil à l'autre. Certains appareils doivent être très peu coûteux, en particulier les applications grand public. Une simple montre-bracelet numérique ou un petit thermomètre numérique en sont des exemples typiques. Mais certains utilisateurs peuvent avoir besoin de montres-bracelets pouvant résister à 10 m de profondeur d'eau. La conception et l'ingénierie des produits changent pour la même fonctionnalité.

VI. Principales caractéristiques des systèmes embarqués

❑ Efficacité énergétique

Les appareils doivent consommer très peu d'énergie car la plupart d'entre eux peuvent fonctionner sur batterie. C'est le critère majeur pour la plupart des appareils portables. Les appareils peuvent être aussi placés dans des endroits où ils peuvent devoir être auto-alimentés.

❑ Compact

Les appareils grand public doivent être suffisamment compacts pour pouvoir être tenus dans les poches et la main et emportés. La plupart des appareils portables peuvent être répertoriés dans cette classe d'appareils.

❑ Réactif

Tous les systèmes embarqués ont une propriété réactive. Ils réagissent en permanence aux changements de l'environnement du système et répondent aux entrées fournies par l'utilisateur final ou l'environnement. Exemple : Un climatiseur, une machine de contrôle de l'humidité dans votre maison. Ils réagissent à l'environnement. Le temps nécessaire à la réaction dépend du temps de réponse et de la rapidité souhaitée de l'appareil.

VI. Principales caractéristiques des systèmes embarqués

- **Temps de réponse**

Les appareils doivent être suffisamment rapides pour fournir la réponse souhaitée. La rapidité dépend du type d'application. La rapidité est toujours relative et dépend des exigences.

- **Temps réel**

La majorité des appareils ont besoin d'un comportement en temps réel. Une manière très approximative de définir le temps réel est que le temps est un facteur de validité de la sortie. Cela signifie que la sortie doit être disponible à un moment précis. Le temps est un facteur dans la conception du système. Les gens définissent également cela comme un système plus rapide car il devrait calculer les résultats en temps réel sans délai. Mais ce n'est pas vraiment une vraie définition.

- **Mobilité** (Fixe, mobilité : faible, moyenne et grande)
- **Communication** (échange, supervision, contrôle, etc…..

la communication affecte bcp l'énergie du SE

- **Contraintes de sécurité** (militaire, e-health, space, finance, etc…)

Confidentialité – intégrité – disponibilité - …..

- **Coût de produits en relation avec le secteur cible** (Gd public – Militaire- etc)

VI. Principales caractéristiques des systèmes embarqués

- **Temps de réponse**

Les appareils doivent être suffisamment rapides pour fournir la réponse souhaitée. La rapidité dépend du type d'application. La rapidité est toujours relative et dépend des exigences.

- **Temps réel**

La majorité des appareils ont besoin d'un comportement en temps réel. Une manière très approximative de définir le temps réel est que le temps est un facteur de validité de la sortie. Cela signifie que la sortie doit être disponible à un moment précis. Le temps est un facteur dans la conception du système. Les gens définissent également cela comme un système plus rapide car il devrait calculer les résultats en temps réel sans délai. Mais ce n'est pas vraiment une vraie définition.

- **Mobilité** (Fixe, mobilité : faible, moyenne et grande)
- **Communication** (échange, supervision, contrôle, etc…..

la communication affecte bcp l'énergie du SE

- **Contraintes de sécurité** (militaire, e-health, space, finance, etc…)

Confidentialité – intégrité – disponibilité - …..

- **Coût de produits en relation avec le secteur cible** (Gd public – Militaire- etc)

VII. Métriques de qualité dans la conception des systèmes embarqués

❑ Frais d'ingénierie non récurrents (NRE)

Lorsque vous commencez à développer un produit, vous faites une conception conceptuelle pour développer le prototype. Les outils nécessaires au développement seront achetés, une conception sera mise en œuvre et la fonctionnalité sera vérifiée. Vous obtenez l'approbation formelle du client si le système est destiné à un client spécifique. Les coûts impliqués dans le développement d'une version technique du système qui est prête pour la production sont des coûts d'ingénierie non récurrents. (NRE) Ces coûts doivent être absorbés dans le coût de commercialisation du produit. Si vous souhaitez que le produit soit mis sur le marché rapidement, vous pouvez prévoir des outils de développement rapide. Si le prototype initial est conçu avec une haute qualité (mesures quantifiées élevées), le coût et le temps de développement augmenteront. En effet, Un équilibre doit être atteint grâce à une optimisation appropriée.

❑ Coût unitaire

Le coût unitaire est calculé à partir du coût total NRE et des coûts de production. En effet, le coût unitaire dépend de la quantité requise. Le coût NRE et la quantité déterminent les coûts unitaires.

VII. Métriques de qualité dans la conception des systèmes embarqués

❑ Performance

Chaque consommateur a besoin de systèmes hautement performants, qu'il utilise ou non cette fonctionnalité. Nous devons optimiser correctement les principales fonctionnalités en fonction de la précision et des temps de réponse réellement souhaités par les utilisateurs. Les performances peuvent être en termes de temps de réponse, de fonctionnalité et de plusieurs facteurs.

❑ Efficacité énergétique

La métrique souhaitée pour les applications grand public telles que les appareils portables, les mobiles, les appareils portables, etc. doit être conçue de manière à consommer très moins d'énergie et à réduire le taux de charge. Les consommateurs ont besoin de systèmes alimentés par le secteur pour consommer moins d'énergie afin de réduire les coûts énergétiques. C'est pourquoi vous trouvez cinq étoiles à deux étoiles attribuées aux climatiseurs, téléviseurs, réfrigérateurs, etc.

❑ Sécurité

La sécurité personnelle est la plus haute priorité. Vous avez peut-être entendu dire que certains appareils mobiles ont explosé pendant leur utilisation. Les aspects de sécurité souhaités doivent être introduits dans la conception elle-même. La sécurité est le facteur le plus important pour que les utilisateurs et le site installé ne soient pas endommagés et qu'il n'y ait aucune perte de vie humaine. Il existe certaines normes et réglementations internationales à adapter dans la conception des produits par lesquelles les aspects de sécurité doivent être respectés.

VII. Métriques de qualité dans la conception des systèmes embarqués

❑ Mises à jour fonctionnelles

Les clients ont besoin de mises à jour fonctionnelles sur les systèmes existants. La conception doit être suffisamment flexible pour ajouter de nouvelles fonctionnalités au produit, auxquelles on n'a pas pensé au moment de la conception. Ce concept était difficile dans les premiers jours. Mais maintenant, comme les systèmes sont pilotés par processeur et connectés via Internet et intelligents, de nouvelles fonctionnalités peuvent être ajoutées en silence via des mises à jour logicielles.

❑ Interface utilisateur

Les clients ont besoin d'une interface facile avec le système. Ils attendent une facilité d'interaction.

❑ Taille

Les utilisateurs ont autant que possible besoin d'appareils compacts. Cependant, il nécessite une optimisation importante en termes de coût. Lorsque vous essayez de miniaturiser en utilisant des ASIC et des dispositifs logiques programmables, les coûts NRE augmentent. Alors, trouvez un équilibre entre la taille, le coût et le délai de mise sur le marché.

VII. Métriques de qualité dans la conception des systèmes embarqués

❑ Maintenabilité

Une fois que le produit est opérationnel dans un site, il doit être maintenu de tous les défauts et apporter des améliorations mineures dans le site lui-même. Aujourd'hui, diverses techniques sont disponibles pour rendre le système maintenu à distance et la conception elle-même devrait avoir des caractéristiques suffisantes grâce auxquelles nous pouvons facilement maintenir les systèmes sur le marché très rapidement.

❑ Rugosité

Généralement, il est considéré comme la robustesse physique du système. Mais dans les systèmes embarqués, il est mesuré en robustesse fonctionnelle, à savoir, la récupération de conditions inattendues, l'exactitude des mesures dans des environnements difficiles, etc.

❑ Véracité

La justesse du système est principalement observée par l'exactitude des mesures. Dans une certaine mesure, cela dépend de la précision. La précision dépend du coût. Par exemple, si vous fabriquez une simple balance numérique, la précision doit être juste suffisante pour obtenir de vrais résultats du client. Si vous voulez le rendre trop précis, ce qui n'est pas vraiment requis par le consommateur, vous augmentez effectivement le coût du système. Voici donc le scénario d'optimisation des exigences.

VII. Métriques de qualité dans la conception des systèmes embarqués

❑ Délai de mise sur le marché

Le temps de mise sur le marché est une mesure importante. Un produit doit concurrencer des produits similaires existants sur le marché. La durée de vie d'un produit ressemble à une courbe en forme de cloche. Dans un premier temps, il doit ramasser des produits concurrents. Les ventes augmenteront si elles disposent de mesures polyvalentes par rapport aux produits concurrents. Cependant, la consommation du produit s'estompe lentement à mesure que les consommateurs trouvent de nouveaux produits compétitifs avec des mesures améliorées. Par conséquent, vous constatez que le cycle de vie d'un produit de consommation suit à peu près une courbe en forme de cloche. Si l'entrée d'un produit sur le marché est retardée, les ventes totales diminuent. Le revenu obtenu est l'aire de la courbe en cloche.

Cette métrique dépend du temps nécessaire pour prototyper l'unité, les tests et la vérification. Se rendre ensuite à la production et à la mise sur le marché. Ce processus doit être optimisé afin que le produit soit disponible sur le marché au bon moment avant même que vos concurrents n'essayent d'apporter un article similaire. L'optimisation dépend du type de conception que vous planifiez et du temps nécessaire à ce processus. Par exemple, si vous souhaitez développer un appareil compact et à faible consommation, vous pouvez envisager de développer un ASIC. Mais cela implique que le time-to-market soit très élevé. Au moment où vous créez un prototype et le mettez sur le marché, vous risquez de perdre le marché ou de réduire votre part de marché. Par conséquent, vous devez optimiser correctement la conception. C'est un défi de conception.

VIII. Cycle de développement d'un système embarqué

Désigne toutes les étapes du développement d'un système embarqué (logiciel et matériel) , de la conception à la commercialisation.

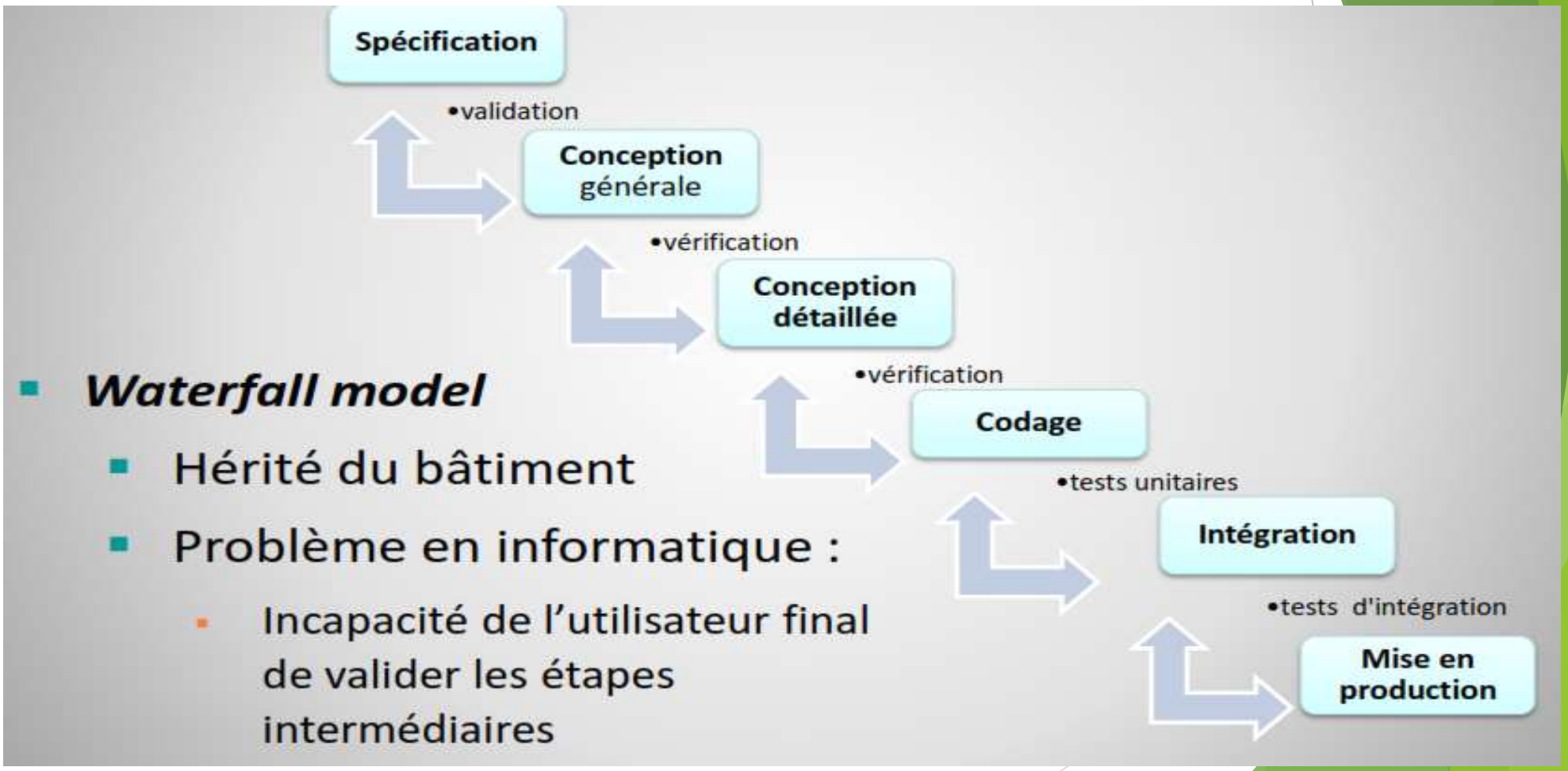
▪ Objectif :

– Valider le développement logiciel et son intégration au matériel cible, en assurant la conformité avec les besoins exprimés

– Vérifier durant chaque étape du cycle, le processus de développement (adéquation des méthodes mises en œuvre)

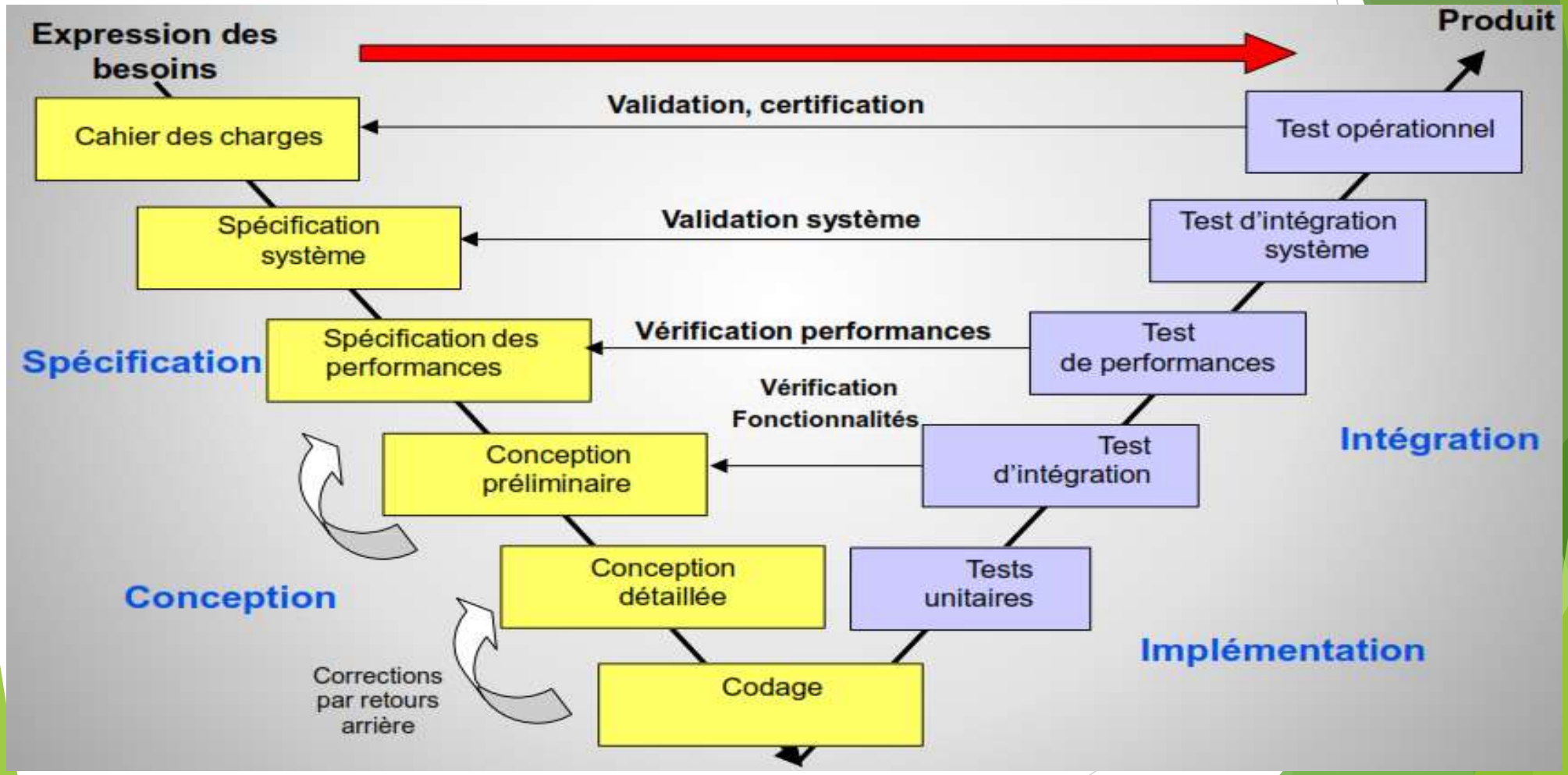
VIII. Cycle de développement d'un système embarqué

❑ Modèle de développement en Cycle Cascade



VIII. Cycle de développement d'un système embarqué

❑ Modèle de développement en Cycle V



VIII. Cycle de développement d'un système embarqué

❑ Activités du Cycle en V

- Expressions des besoins : Consiste à définir les objectifs, la finalité du projet et son inscription dans une stratégie globale.
  - ▪ Une description précise de ce que veut l'utilisateur (client) et de ce qu'il espère obtenir
  - ▪ Besoins fonctionnels :
    - ✓ Sorties en fonction des entrées et des paramètres.
  - ▪ Besoins non fonctionnels :
    - ✓ temps nécessaire pour calculer la sortie,
    - ✓ taille, poids, etc.,
    - ✓ consommation,
    - ✓ fiabilité,
    - ✓ etc.

Comprendre le besoin du client et savoir aussi l'identifier !

VIII. Cycle de développement d'un système embarqué

❑ Activités du Cycle en V

- Spécification : c-à-d , l'expression, le recueil, l'analyse et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes.

- Conception générale : Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel et du matériel.

- Conception détaillée : consistant à définir précisément chaque sous ensemble du Logiciel Et matériel.

Codage : (Implémentation ou programmation), soit la traduction dans un langage de programmation des fonctionnalités définies lors des phases de conception.

VIII. Cycle de développement d'un système embarqué

❑ Activités du Cycle en V

- Tests unitaires : permettant de vérifier individuellement que chaque sous-ensemble du logiciel est ce qu´il est implémenté conformément aux spécifications.

- Intégration : dont l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document.

- Qualification (ou recette): c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales.

- Documentation: visant à produire les informations nécessaires pour l'utilisation des ressources pour des développements ultérieurs.

VIII. Cycle de développement d'un système embarqué

❑ Activités du Cycle en V

Mise en production : déploiement du matériels et logiciels en masse, en vue d'une va industrialisation . Les mises en production se font de manière industrielle, précédées de batteries de tests

- Test et validation

✓ Tests unitaires : interviennent au niveau modules ou sous-systèmes de chaque brique logicielle modélisée puis codée durant les étapes précédentes. Ces tests assurent que ces briques respectent de manière individuelle leur cahier des charges.

✓ Les tests d'intégration : vérifient l'intégralité du produit et le valide techniquement.

VIII. Cycle de développement d'un système embarqué

❑ Activités du Cycle en V

- Mise en production : déploiement du matériels et logiciels en masse, en vue d'une va industrialisation . Les mises en production se font de manière industrielle, précédées de batteries de tests

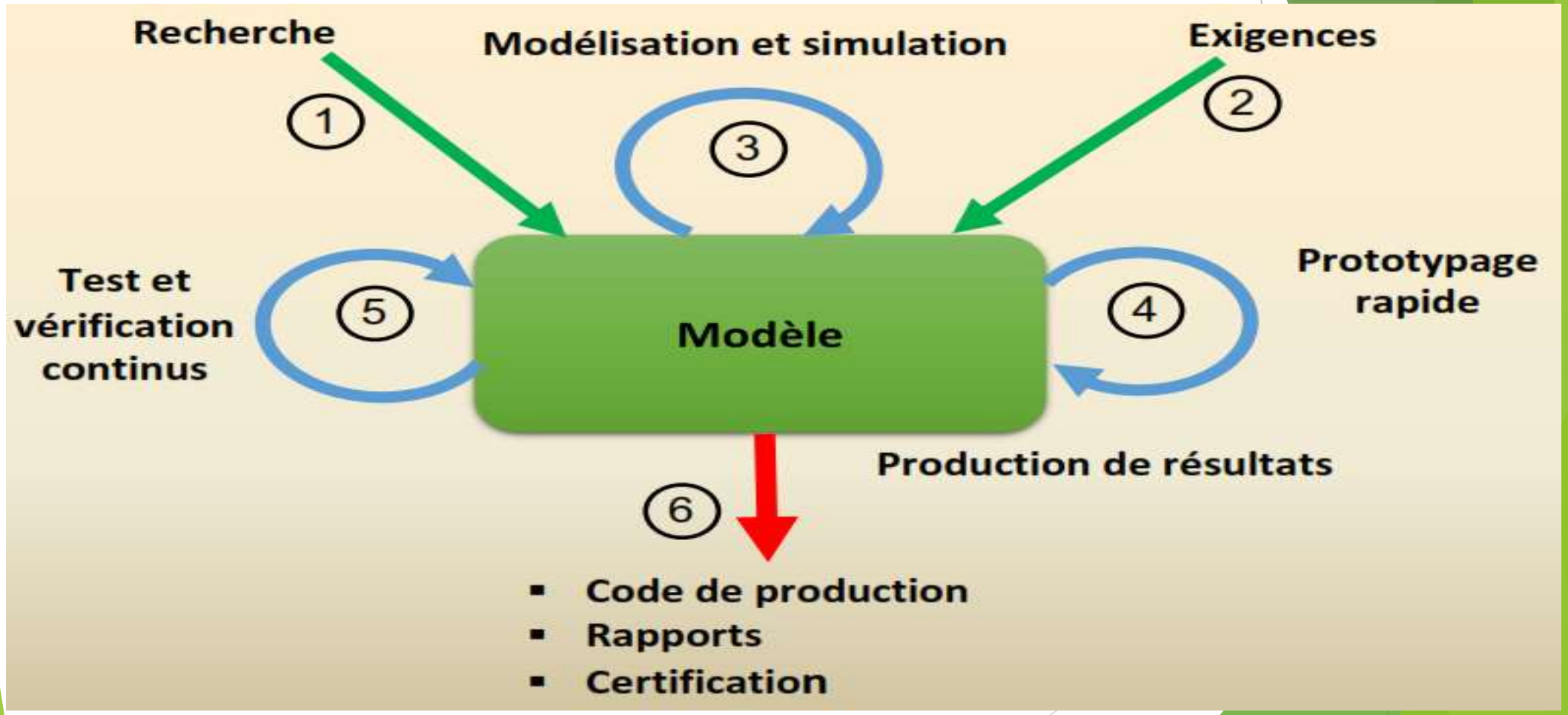
- Test et validation

✓ Tests unitaires : interviennent au niveau modules ou sous-systèmes de chaque brique logicielle modélisée puis codée durant les étapes précédentes. Ces tests assurent que ces briques respectent de manière individuelle leur cahier des charges.

✓ Les tests d'intégration : vérifient l'intégralité du produit et le valide techniquement.

VIII. Cycle de développement d'un système embarqué

❑ Méthodologie de Conception : Conception basée sur modèle / Model based design



VIII. Cycle de développement d'un système embarqué

❑ Méthodologie de Conception : Conception basée sur modèle / Model based design

▪ La conception basée sur le modèle est une approche axée sur le modèle pour les systèmes de contrôle, de traitement du signal, des communications et d'autres systèmes dynamiques.

▪ Plutôt que de s'appuyer sur des prototypes physiques et des spécifications textuelles, La conception basée sur le modèle utilise un modèle tout au long du cycle de développement.

VIII.Cycle de développement d'un système embarqué

❑ Méthodologie de Conception : Conception basée sur modèle / Model based design

- Le modèle comprend tous les composants pertinents pour le comportement du système :
  - ✓ Algorithmes,
  - ✓ logique de contrôle,
  - ✓ composants physiques et propriété intellectuelle (IP).
- Une fois que le modèle est développé (élaboré), il devient la source de nombreuses sorties, y compris les rapports et le code généré (code C ou HDL).
- La conception basée sur le modèle permet la conception, la simulation au niveau du système et des composants, la génération automatique de code, le test et la vérification en continu

VIII. Cycle de développement d'un système embarqué

❑ Pourquoi la conception basée sur un modèle ?

La conception basée sur un modèle permet de rationaliser de nombreux aspects du développement, par exemple:

- Gérer les systèmes complexes.
- Automatiser les tâches fastidieuses et faibles.
- Explorer rapidement de nouvelles idées.
- Créer une langue commune qui favorise la communication et collaboration.
- Augmenter la qualité du produit.
- Réduire le risque.

VIII. Cycle de développement d'un système embarqué

❑ Concepts fondamentaux de la conception basée sur modèle

La conception basée sur modèle repose sur sept concepts fondamentaux :

1. Spécification exécutable.
2. Simulation au niveau du système.
3. Analyse Quoi-si ?.
4. Elaboration du modèle.
5. Prototypage virtuel.
6. Test et vérification continus.
7. Automatisation.

VIII. Cycle de développement d'un système embarqué

❑ Simulation au niveau du système.

- Utilisée au niveau du système pour valider les exigences, vérifier la faisabilité d'un projet et effectuer des tests et des vérifications anticipées.
- L'ensemble du modèle du système est simulé pour étudier les performances du système et les interactions des composants.
- La simulation fournit un moyen de vérifier les systèmes multi –domaines qui sont plus complexes à comprendre et à maitriser.
- Les problèmes de conception et les incertitudes peuvent faire l'objet d'une enquête précoce, bien avant de créer un matériel coûteux.
- Les simulations sont sécurisées : il n'y a aucun dommage sur le matériel ou d'autres dangers si la conception ne fonctionne pas.

IX. Processeurs Embarqués

❑ Les unités de calcul

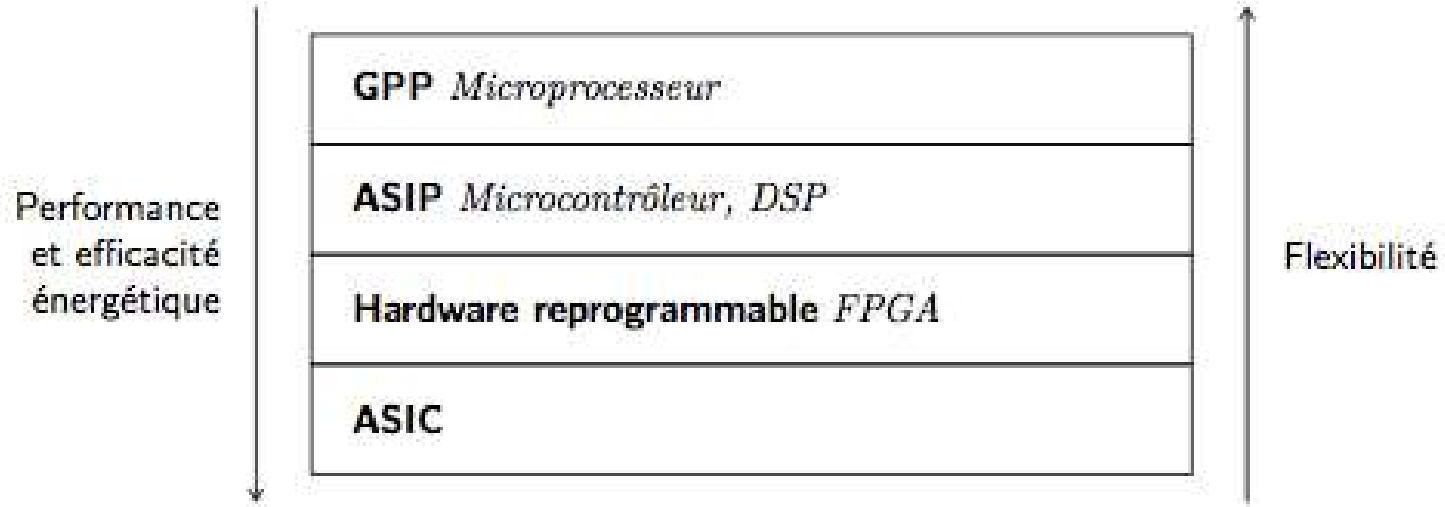
Plusieurs types d'implémentations sont possibles pour les unités de calcul, allant de systèmes plus génériques comme l'Intel 4004 vers des solutions beaucoup plus spécifiques. On distingue essentiellement quatre grandes catégories :

- Le ***General Purpose Processor*** **(processeur à usage général) (GPP)** propose un jeu d'instructions assez large et varié permettant de réaliser beaucoup de choses. C'est le genre de processeur que l'on retrouve notamment dans les ordinateurs *« traditionnels »*.
- ***L'Application-Specific Instruction set Processor*** (processeur dont le jeu d'instructions est spécifique à l'application) (ASIP) possède un jeu d'instructions qui est spécifiquement conçu pour un certain ensemble d'opérations.
- Vient ensuite le ***hardware reprogrammable*** dont la fonction peut être modifiée après sa fabrication, une fois sorti d'usine donc. Pour être précis, il s'agit plus d'une reconfiguration que d'une reprogrammation étant donné que ce type d'unité de calcul n'exécute pas d'instructions.
- Enfin, l'unité de calcul la plus spécifique est ***l'Application-Specific Integrated Circuit*** (circuit intégré propre à une application et spécialisé pour cette dernière) (ASIC) qui est un circuit intégré réalisant les fonctions nécessaires pour une application donnée.

IX. Processeurs Embarqués

❑ Les unités de calcul

Une unité de calcul générique sera plus flexible, mais au détriment de moins bonnes performances et d'une efficacité énergétique moindre, et inversement pour les unités de calcul spécifiques.



Processeurs Embarqués

❑ Processeur

Le processeur (microprocesseur) est le composant hardware le plus connu d'un système microprogrammé. C'est l'unité intelligente de traitement des informations. Son travail consiste à:

• lire des programmes (des suites d'instructions),
• les décoder et
• les exécuter.

Son architecture et brochage sont résumés dans ***:***

• *L'unité de commande.*

Elle permet de "séquencer" le déroulement des instructions. Elle effectue la recherche en mémoire de l'instruction, le décodage, l'exécution et la préparation de l'instruction suivante. L'unité de commande élabore tous les signaux de synchronisation internes ou externes (bus des commandes) au microprocesseur.

• *L'unité arithmétique et logique (UAL).* C'est l'organe qui effectue les opérations:

- *Arithmétiques* : addition, soustraction, multiplication, ...
- Logiques : et, ou, non, décalage, rotation, ...
- Des registres.
- registres de données;
- registres de segment ;
- registres pointeurs;
- registre d'état

Processeurs Embarqués

❑ Processeur

Types des µp : Architecture

a) Type CISC

La majorité des microprocesseurs ont une architecture CISC (*ex Complex Instruction Set Computer), *ce qui signifie "ordinateur avec jeu d'instructions complexes". C'est le cas des* processeurs de type x86, c'est-à-dire les processeurs fabriqués par Intel, AMD, Cyrix, ex: Vax, Motorola 68000, Intel x86/Pentium

b) Type RISC

Reduced instruction set computer ou RISC (*en français « microprocesseur à jeu d'instructions réduit* ») est un type d'architecture matérielle de µp qui se caractérise par un jeu d'instructions réduit, facile à décoder et comportant uniquement des instructions simples.

- Petites instructions simples, toutes de même taille, ayant toutes (presque) le même temps d'exécution
- Pas d'instruction complexe
- Accélération en pipelinant l'exécution (entre 3 et 7 étages de pipeline pour une instruction) augmentation de la vitesse d'horloge
- Code plus simple à générer, mais moins compact
- Tous les microprocesseurs modernes utilisent ce paradigme:
- SPARC, MIPS, ARM, PowerPC, MC, etc.

IX. Processeurs Embarqués

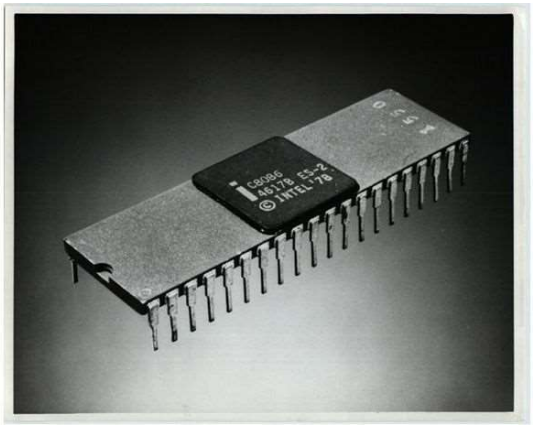
❑ Processeur

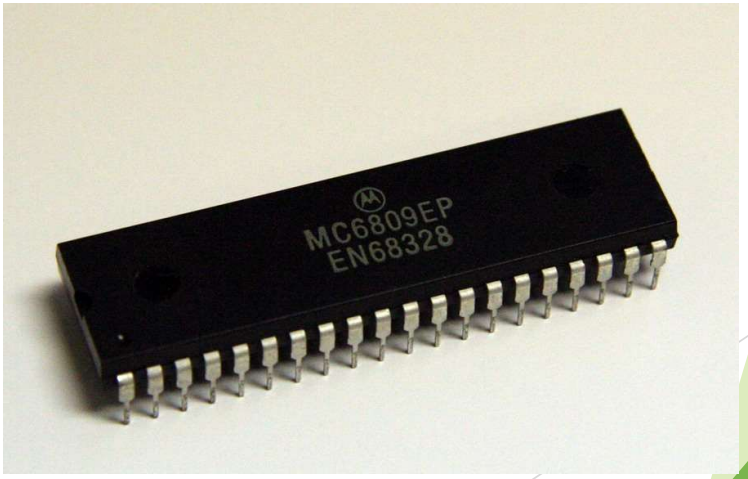
Catégories de microprocesseur

On distingue quatre classes de µp:

1- Usage général - processeurs de haute performance

- Pentium, SPARC, Intel 80x86, Motorola 68HCxxx ...
  - Utilisé pour les logiciels d'usage général
  - Général sous contrôle d'un système d'exploitation (Windows .., UNIX ..,)
  - Poste de travail, PC





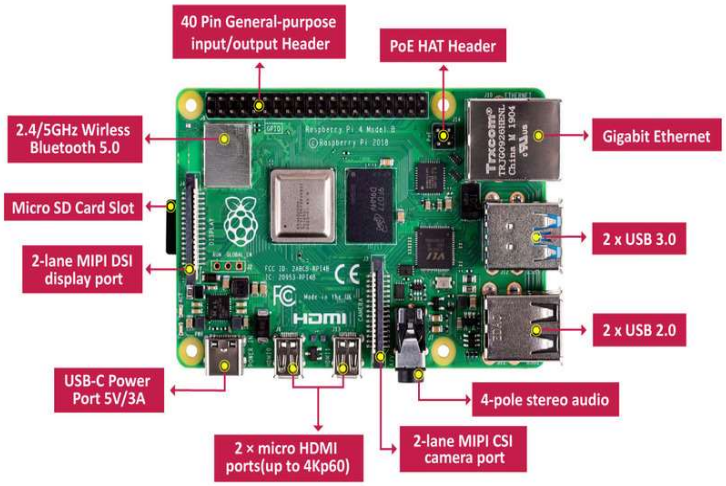
Ne sont plus utiliser dans les systèmes embarqués

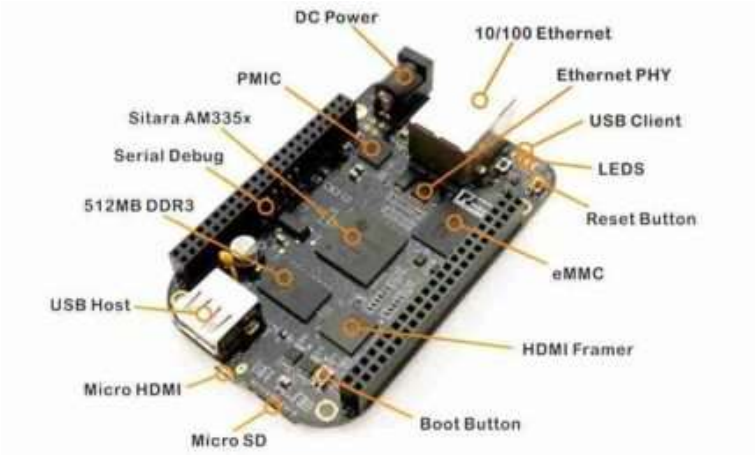
IX. Processeurs Embarqués

❑ Processeur

2- processeurs embarqués et noyaux spéciaux

- 486SX, ARM, Hitachi SH7000, NEC V800, Motorola PowerPC, ...
  - programme unique
  - Léger, souvent sous OS temps réel (Windows CE, QNX, OS-9, PSOS, ...)
  - Prise en charge de traitement numérique du signal (léger)
  - consommation électroniques, les téléphones cellulaires





IX. Processeurs Embarqués

❑ Processeur

3- Microcontrôleurs

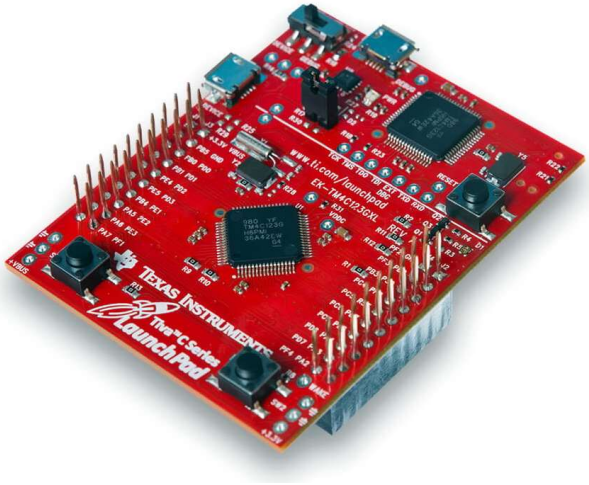
- Chipsets spécifiques 8, 16, (32) Bit, avec des caractéristiques différentes
  - A / D, D / A, entrée, sortie, Timer, Interfaces, ...
  - Coût sensibles
  - Les plus hauts processeurs de volume pour les machines de l'automobile, de lavage, ...
  - DSP à usage général et la conception de base 'Gate Arrays‘

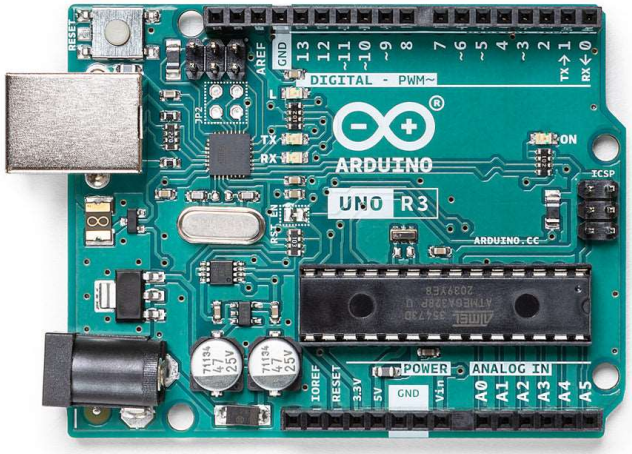
la famille les plus connu : Microchip (PIC16f xx, PIC18fxx, PIC32, …….)
Atmel/ avr : Atmega328p (ARDUINO UNO), Atmega644Pa, AT32……..
STmciroelectronics: STM32F0, STM32F 4, STM32F7……
Motorola : 68HC16, 68HCxx……
NXP freescale: 683xx
Texas instrument: MSP 430

IX. Processeurs Embarqués

❑ Processeur

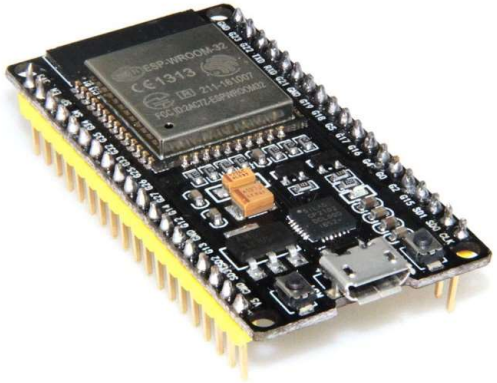
3- Microcontrôleurs

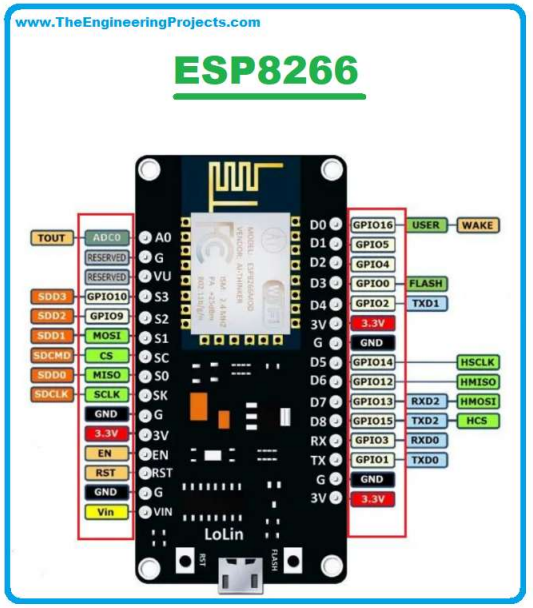




ESP32 38PIN

WiFi + Bluetooth



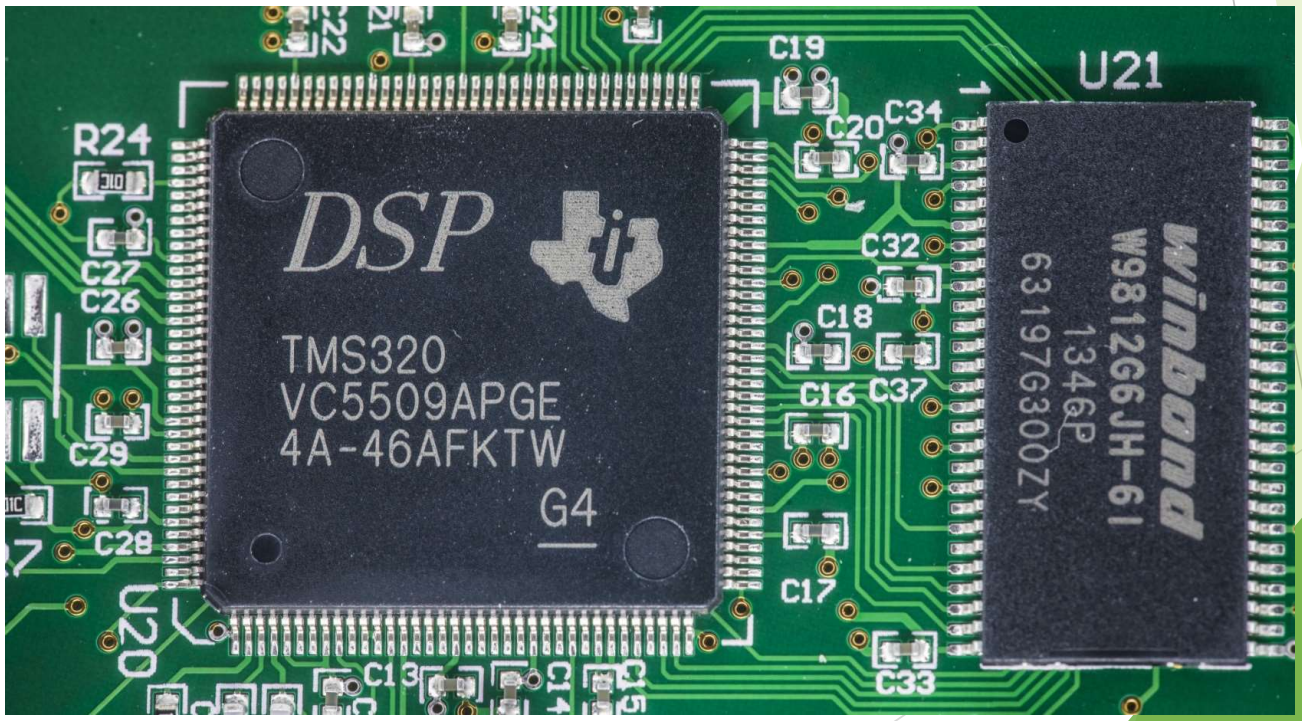


IX. Processeurs Embarqués

❑ Processeur

4 – Processeur de traitement numerique du signal / DSP.

-Un DSP pour Digital Signal Processor en anglais est un processeur spécialisé dans le traitement numérique du signal. Son architecture est optimisée pour traiter une grande quantité de données en parallèle à chaque cycle d'horloge. Ce mode de fonctionnement est très efficace pour traiter des signaux numériques (filtrage, compression, extraction, etc.) comme de la vidéo ou de la musique.



IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

-Un **circuit logique programmable** ou **PLD** (*Programmable Logical Device*), est un circuit intégré logique qui peut être programmé après sa fabrication.

Il se compose de nombreuses cellules logiques élémentaires contenant des bascules logiques librement connectables. L'utilisateur doit donc programmer le circuit avant de l'utiliser. Les différentes logiques de programmation (unique, reprogrammable, etc.) et d'architecture ont conduit à la création de sous-familles dont les plus connues sont les FPGA et les CPLD.

La notion de programmation des PLD revient à définir une table de connexion et d'interconnexion des portes logiques. Ce n'est donc pas une programmation algorithmique (c.-à-d. une série d'instructions faite pour tourner sur un processeur) mais une programmation matérielle.

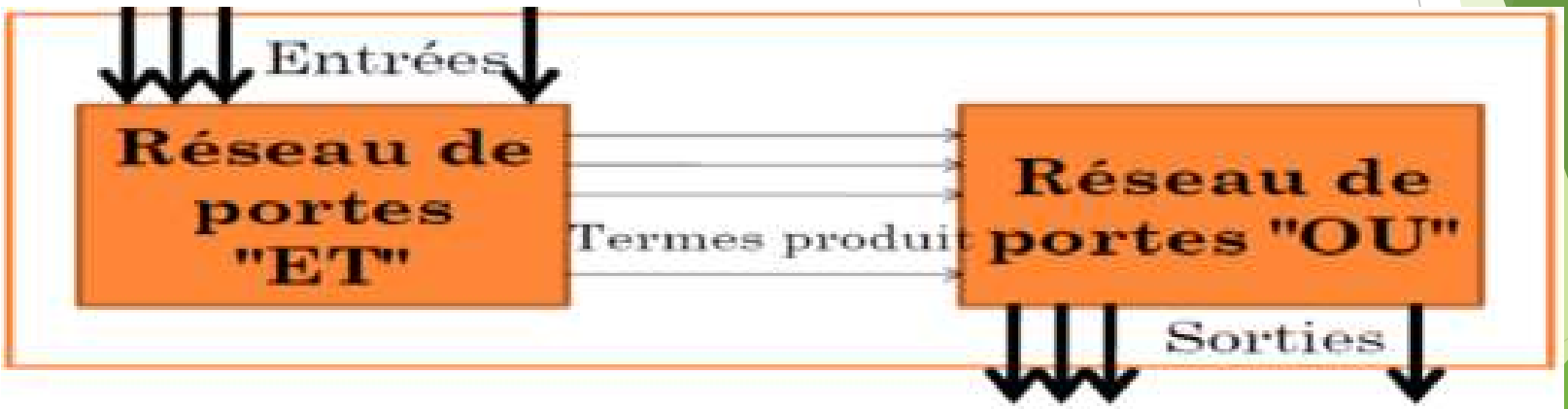
Il existe trois principaux circuits programmables : ??

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ **SPLD (Simple PLD)** Les Simple Programmable Logic Device sont des circuits programmables élémentaires, constitué d'un ensemble de portes « ET » sur lesquelles viennent se connecter les variables d'entrée et leurs compléments. Ainsi qu'un ensemble de portes « OU » sur lesquelles les sorties des opérateurs « ET » sont connectées les variables d'entrée. Comme le montre la figure suivante :



Architecture global d'un SPLD

Il existe trois type des SPLD :

IX. Processeurs Embarqués

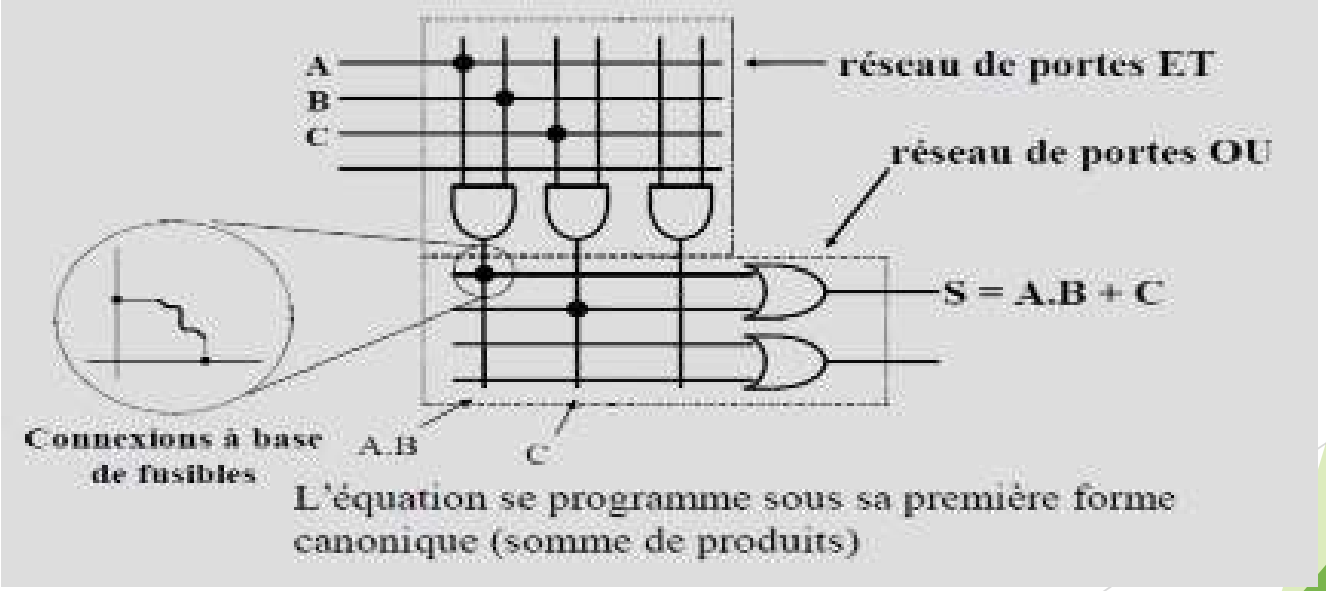
❑ Processeur

4 – Les circuits logiques programmable

❖ SPLD (Simple PLD):

✓ PAL (Programmable Array Logic) : réseaux logiques programmables

- Développés au début des années 70 par MMI (ex-AMD).
- La programmation se fait par destruction de fusibles.
- Aucun fusible n'est grillé à l'achat de la PAL.



Architecture d'un PAL

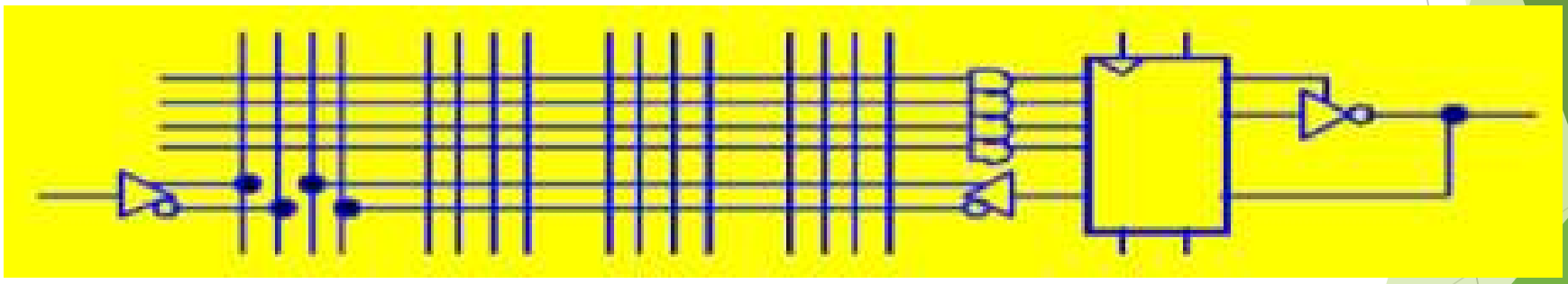
IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ SPLD (Simple PLD):

✓ GAL (Généric Array Logic) L'inconvénient majeur des PAL est qu'ils ne sont programmables qu'une seule fois. Ce qui a ramené la firme LATTICE a pensé de remplacer les fusibles irréversibles des PAL par des transistors MOSFET qui peuvent être régénérés. Ceci a donné naissance aux GAL « Réseau Logique Générique ». Ces circuits peuvent donc être reprogrammés à volonté,



Implémentation ET-OU-bascule D d'une cellule de base d'un circuit GAL

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ CPLD (Complexe PLD):

Les CPLD peuvent être vu comme une intégration de plusieurs PLD simples (SPLD) dans une structure à deux dimensions, ils sont composés de blocs logiques répartis autour d'une matrice d'interconnexion PI (Programmable Interconnect).

Dans les années 80, Altera sont les premiers à fournir une solution utilisable de CPLD:

o Utilisé pour le prototypage rapide,

o Un problème restait pourtant au niveau de la matrice d'interconnexion entreles SPLD,

o Limitant la taille des designs à prototyper

o En 1984 Xilinx, lance le premier Field Gate Array (FPGA), le XC2064.

o La principale différence est sa flexibilité sans perte de performances.

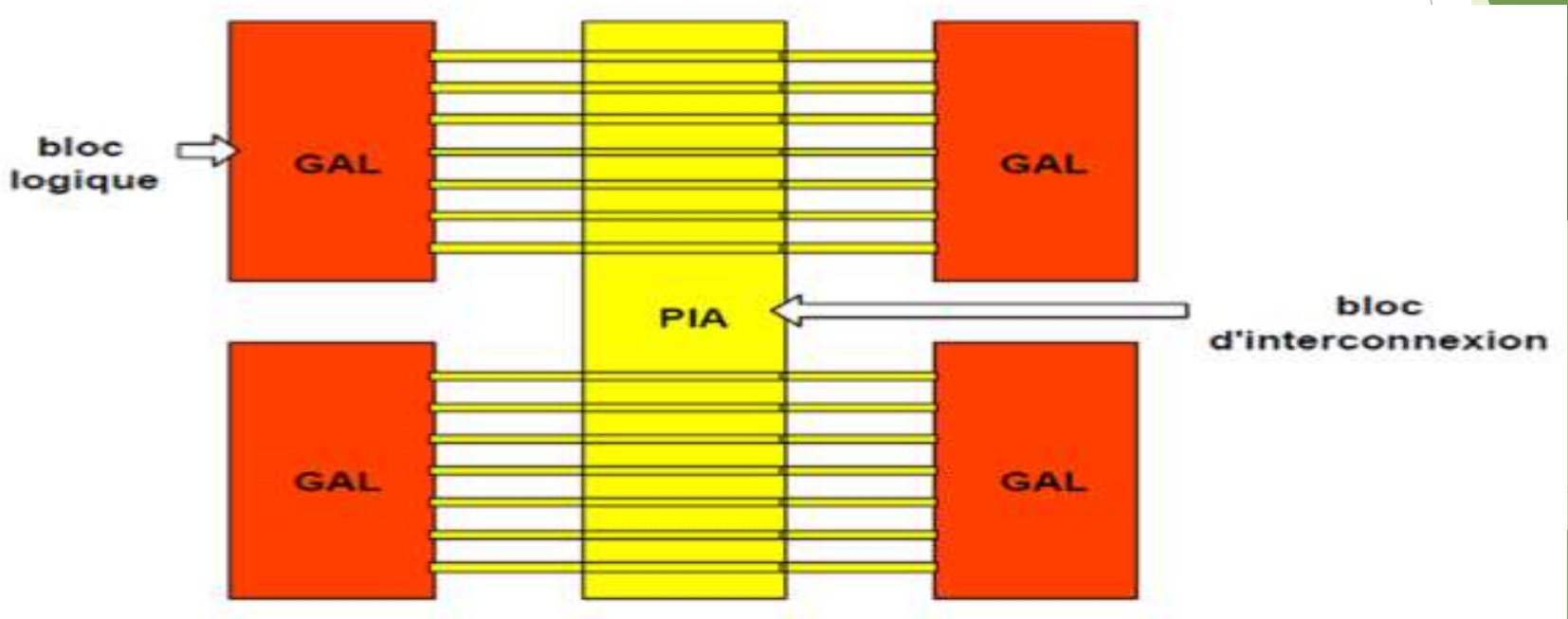
o Son inconvénient est le manque de prédictibilité des temps d'interconnexion.

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ CPLD (Complexe PLD):



Circuit CPLD

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ FPGA (Field Programmable Gate Array):

- Circuit programmable composé d'un réseau de blocs logiques élémentaires (plusieurs milliers de portes), de cellules d'entrée-sortie et de ressources d'interconnexion totalement flexibles

- La puissance de ces circuits est telle qu'ils peuvent être composés de plusieurs milliers voire millions de portes logiques et de bascules. Les dernières générations de FPGA intègrent même de la mémoire vive (RAM). Les deux plus grands constructeurs de FPGA sont XILINX et ALTERA.

- Ce sont des circuit ultra rapide

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ FPGA (Field Programmable Gate Array):

✓ Différence entre CPLD et FPGA

- FPGA contient jusqu'à 100 000 de minuscules blocs logiques, tandis que CPLD ne contient que quelques blocs de logique allant jusqu'à quelques milliers.
- En termes d'architecture, les FPGA sont considérés comme des dispositifs à « grain fin » alors que les CPLD sont des « grains grossiers ».
- Les FPGA sont parfaits pour des applications plus complexes, tandis que les CPLD sont mieux pour les plus simples.
- Les FPGA sont constitués de minuscules blocs logiques tandis que les CPLD sont constitués de blocs plus gros.
- FPGA est une puce logique numérique basée sur la RAM, tandis que CPLD est basé sur EEPROM.
- Normalement, les FPGA sont plus coûteuses alors que les CPLD sont beaucoup moins chers.
- Les délais sont beaucoup plus prévisibles dans les CPLD que dans les FPGA.

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ FPGA (Field Programmable Gate Array):

	Xilinx	*Altera*	*Actel*
Hautes performances	Veritex	Startix	
Rentable	Kintex (ex SPARTAN)	Arria	Pro Aisc 3
Economie d´energie	Artix	Cyclone	Igloo

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ FPGA (Field Programmable Gate Array):

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ Le Langage VHDL

VHDL est l'acronyme de **VHSIC HDL** (***Very High Speed Integrated Circuit Hardware Description Language***), c'est un langage de description matérielle qui a été créé dans les années 1980 à la demande du département de la défense américaine (**DOD**).

De nos jours, le langage **VHDL** devient un outil indispensable pour la conception des systèmes électroniques intégrés, il est proposé par la grande majorité des sociétés de développement et la programmation des composants programmables du type **PLD**, **CPLD** et **FPGA**

Le **VHDL** es utilisé aussi pour concevoir des modèles de simulations numériques ou des bancs de tests. Avec un langage de description matérielle et un **FPGA** (Field Programmable Gate Array), un concepteur peut développer rapidement et simuler un circuit numérique sophistiqué, de l'implémenter sur une carte de prototypage, et de vérifier son fonctionnement

NB: il y a aussi le langage VERILOG

IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ Le Langage VHDL

- Structure d´une description VHDL

L'accès aux bibliothèques

library *nom de la bibliothèque ;*

use permet l'accès aux fonctions de la bibliothèque **all** ;

La déclaration d'entité

entity *Nom_de_l'entité* **is**

generic *déclaration des paramètres (éventuellement) ;*
port *description des E/S de la structure en explicitant pour chacune d'entre elles,* ***le nom****, la direction ou* ***mode*** *et* ***le type*** *;*

end *Nom_de_l'entité ;*

L'architecture

architecture *Nom_de_l'architecture* **of** *Nom_de_l'entité* **is**

zone de déclaration optionnelle

begin

description de la structure logique

end *Nom_de_l'architecture ;*

IX. Processeurs Embarqués

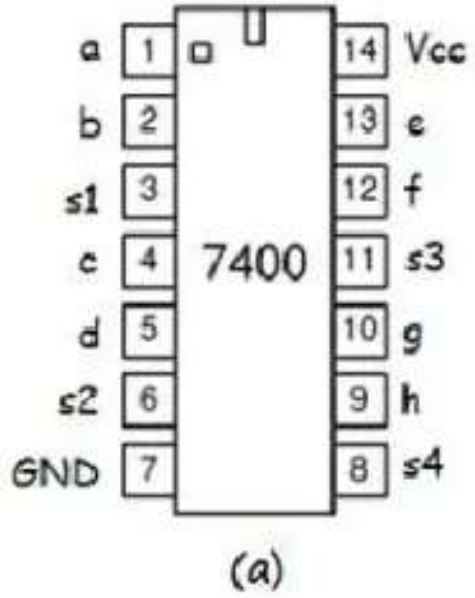
❑ Processeur

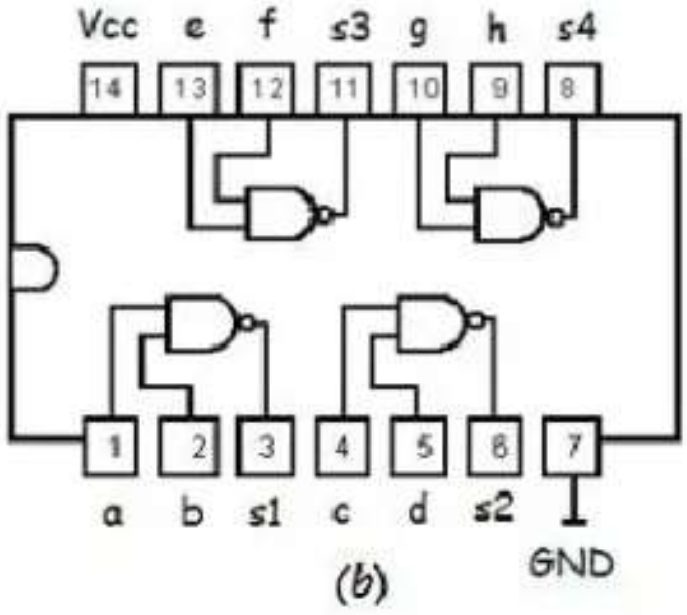
4 – Les circuits logiques programmable

❖ Le Langage VHDL :

Exemple:

Donner la description VHDL de circuit intégré 7400. Ce dernier est constitué de quatre portes logiques NAND à deux entrées





IX. Processeurs Embarqués

❑ Processeur

4 – Les circuits logiques programmable

❖ Le Langage VHDL :

Exemple:

Donner la description VHDL de circuit intégré 7400. Ce dernier est constitué de quatre portes logiques NAND à deux entrées

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity CI_7400 is
5      port ( a, b, c, d, e, f, g, h : in std_logic;
6             s1, s2, s3, s4 : out std_logic
7           );
8   end CI_7400;
9
10  architecture behavioral of CI_7400 is
11  begin
12     s1 <= a nand b;
13     s2 <= c nand d;
14     s3 <= e nand f;
15     s4 <= g nand h;
16  end behavioral;
```

IX. Processeurs Embarqués

❑ SOC System On Chip

Un **système sur une puce**, souvent désigné dans la littérature scientifique par le terme anglais « ***system on a chip*** » est un système complet embarqué sur un seul circuit intégré (« puce »), pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue.

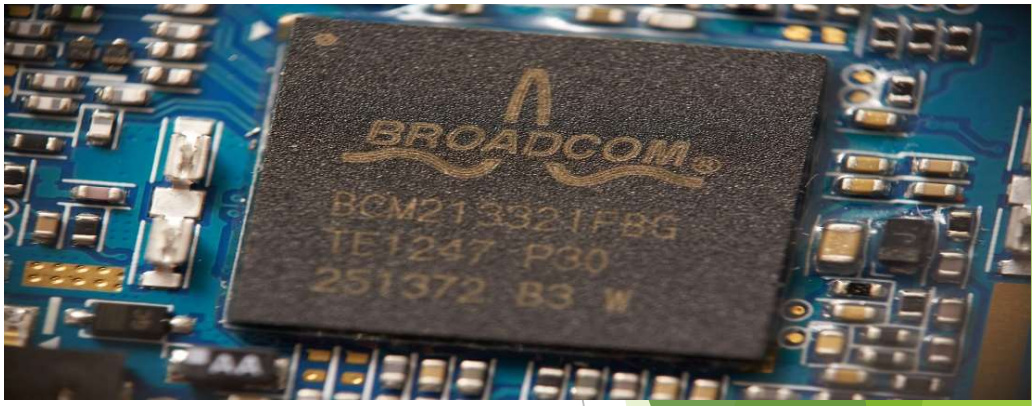
Il peut également comprendre de la logique, de la mémoire (statique, dynamique, flash, ROM, PROM, EPROM ou EEPROM), des dispositifs (capteurs) mécaniques, opto-électroniques, chimiques ou biologiques et des circuits radio.

Exemple

- Architecture ARM: Toutes les puces contenant des processeurs ARM Cortex-A, que ce soit Allwinner, Exynos de Samsung, MediaTek, OMAP de Texas Instruments, Rockchip, Tegra de Nvidia, Snapdragon de Qualcomm, Xgene d'APM (destiné aux serveurs) etc. ou les microcontrôleurs ARM Cortex-M, comme le STM32 de STMicroelectronics
- Architecture Xtensa de Tensilica , principalement utilisée dans l'ESP8266 et l'ESP32.

IX. Processeurs Embarqués

❑ SOC System On Chip



❖ Avantages de SoC

- Taille
- Tous les composants sont maintenant sur la même puce, donc un gain de place important
- Vitesse :
  - Les distances entre les composants sont réduites, donc la fréquence d'horloge peut augmenter considérablement .
  - Les composants sur puce et leur connexion sont plus facilement prévisibles, moins de marge
- Coût
  - Un circuit imprimé moins complexe
  - Moins de soudage

❖ Inconvénient

- Le défaut principal d'un système sur une puce est sa **non-évolution** : on peut en effet faire évoluer un ordinateur en ajoutant de la mémoire, en changeant une carte graphique, une carte réseau, etc.
- Le système sur une puce (SoC) ne peut par contre pas être modifié, donc la seule possibilité est la mise à jour logiciel. En cas de panne, on change donc le SoC et parfois même la carte mère sur laquelle le SoC est fixé.

IX. Processeurs Embarqués

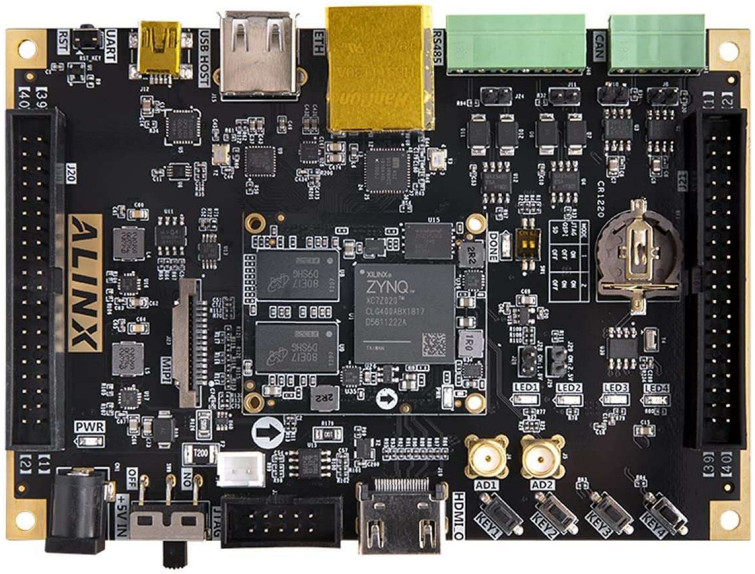
❑ PSOC : Programmable System On Chip

❖ Deux types

- Le processeur lui-même est implanté dans le FPGA par un programme (VHDL, Verilog, etc.)
- Le processeur est gravé physiquement sur le FPGA, on programme des périphérique et autres logiques sur la partie programmable du FPGA

❖ Exemples chez xilinx

- Sur le FPGA Artix-7, on peut implanter le microprocesseur 32 bits MicroBlaze (soft core)
- Le FPGA Zynq contient en dur le microprocesseur ARM cortex-A9 double cœur



X. Hardware / Software Co-Design

❑ Quand le matériel rejoint le logiciel .

- La capacité de conception des systèmes numériques permet aujourd'hui de tout intégrer dans un même composant (concept du single chip).
- On travaille donc au niveau système et non plus au niveau porte élémentaire ou schématique. On parle de système sur silicium SoC (System on Chip) ou SoPC (System on Programmable Chip).
- On utilise maintenant des langages de description du matériel (VHDL, Verilog) pour synthétiser et aussi tester les circuits numériques.
- On a ainsi une approche logicielle pour concevoir du matériel.
- Avec l'augmentation de l'intégration, les systèmes numériques se sont complexifiés alors que la mise sur le marché doit être la plus rapide possible :
  - Prise en compte du *Time To Market* (TTM).

I. Hardware / Software Co-Design

❑ Hardware ou Software ?

- La difficulté est maintenant de savoir comment implémenter une fonctionnalité. Exemple d'un algorithme de compression vidéo :
  - Plus rapide par hardware mais plus cher.
  - Plus flexible par software mais plus lent.

- On doit être capable de jongler en plus avec les paramètres suivants :
  - Coût.
  - Rapidité.
  - Robustesse

X. Hardware / Software Co-Design

❑ Approche traditionnelle

1. Choix du matériel (composants électroniques, processeur…) pour le système embarqué.
2. Donner le système ainsi conçu aux programmeurs.

3. Les programmeurs doivent réaliser un logiciel qui « colle » au matériel en n'exploitant que les ressources offertes.

X. Hardware / Software Co-Design

❑ Complexité grandissante

- MAIS les systèmes embarqués sont de plus en plus complexes.
- Il est de plus en plus difficile de penser à une solution globale optimisée du premier jet.
- Il est de plus en plus difficile de corriger les bugs.

- Il est de plus en plus difficile de maintenir le système au cours du temps (obsolescence des composants…).
- En conséquence, l'approche traditionnelle de développement d'un système embarqué doit évoluer...

X. Hardware / Software Co-Design

❑ Solution a la Complexité

- Dans le processus de conception du système, on doit garder un niveau d'abstraction important le plus longtemps possible.

- Le système doit pouvoir être décomposé en sous-systèmes suivant une hiérarchie logique (approche objet).
  - *Si le design change, on doit pouvoir en réutiliser une bonne partie (design reuse).*
  - *Il ne faut pas jeter à la poubelle un précédent design et repartir from scratch !*

- On doit pouvoir utiliser des outils de vérification automatique.

X. Hardware / Software CoDesign

❑ Co-design hardware/software

- Le Co-design dans la méthodologie de conception d'un système embarqué est de plus en plus utilisé.
- Le Co-design permet de concevoir en même temps à la fois le matériel et le logiciel pour une fonctionnalité à implémenter.
- Le Co-design permet de repousser le plus loin possible dans la conception du système les choix matériels à faire contrairement à l'approche classique où les choix matériels sont faits en premier lieu !

X. Hardware / Software Co-Design

❑ Conception et Co-design hardware/software

Les facteurs à l'origine de la nécessité d'une co-conception/co-design hw-sw sont:

- ❑ La plupart des systèmes actuels incluent à la fois des unités matérielles dédiées et des unités logicielles s'exécutant sur des microcontrôleurs ou des processeurs à usage général.
- ❑ L'utilisation croissante des processeurs programmables utilisés dans les systèmes qui autre fois été tout matériel.
- ❑ La disponibilité de microcontrôleurs bon marché à utiliser dans les systèmes embarqués et la disponibilité de cœurs de processeur pouvant être facilement intégrés dans une conception ASIC.
- ❑ Efficacité accrue des compilateurs de langage de niveau supérieur (C et C++) qui rendre l'écriture de code efficace pour les processeurs embarqués beaucoup plus facile et moins long.

X. Hardware / Software Co-Design

❑ Conception et Co-design hardware/software

Plusieurs avantages apparaissent lors de l'adaptation de la stratégie de co-conception pour les systèmes embarqués.

- ❑ Le co-design oblige les développeurs à se pencher sur le problème de manière holistique
- ❑ Le cycle de vie de la conception est bien défini.
- ❑ La complexité croissante des systèmes embarqués nécessite cette méthodologie pour améliorer la performance du système, qualité, temps de cycle de conception, fiabilité et rentabilité.
- ❑ La tendance actuelle de la conception de systèmes sur puce SoC nécessite des principes de co-design.
- ❑ Le temps de cycle de conception s'améliore considérablement en raison du nombre réduit d'itérations.
- ❑ Profitez des progrès des outils et des technologies.
- ❑ Réduit le temps d'intégration et de test.

X. Hardware / Software Co-Design

❑ Les étapes dans Co-design hardware/software

- **Spécifications** : liste des fonctionnalités du système de façon abstraite.
- **Modélisation** : conceptualisation et affinement des spécifications produisant un modèle du matériel et du logiciel.
- **Partitionnement** : partage logiciel matériel.

- **Synthèse et optimisation** : synthèse matérielle et compilation logicielle.
- **Validation** : co-simulation
- **Intégration** : rassemblement des différents modules
- **Tests d'intégration** : vérification du fonctionnement.

X. Hardware / Software Co-Design

❑ Partitionnement

❑ **Le partitionnement matériel-logiciel est l'activité critique de la co-conception/ co-design.**

❑ ***Le partitionnement HW-SW est le processus consistant à décider si la fonctionnalité requise est plus avantageusement implémentée dans le matériel ou le logiciel. Cet exercice doit être fait pour chaque fonction principale à mettre en œuvre sur le sous-système partitionné.***

X. Hardware / Software Co-Design

❑ Partitionnement

L'objectif du partitionnement doit être clair dès les premières étapes du co-design .

- ❑ partitionnement pour augmenter la vitesse d'exécution
- ❑ réduire les latences dans le changement de tâche ;
- ❑ réduire la taille du matériel et transférer la fonctionnalité au logiciel sans compromettre la rapidité d'exécution ;
- ❑ réduire le coût global du système ;
- ❑ planifier l'exécution des tâches du système pour répondre à toutes les contraintes de temps ;
- ❑ modéliser le système tout au long du processus de conception pour valider qu'il répond aux objectifs et fonctionnalités d'origine.
- ❑ les activités d´ordonnancement en co-design sont:
  - ➢ Ordonnancement des opérations dans le matériel,
  - ➢ Ordonnancement des instructions dans le logiciel
  - ➢ Ordonnancement des processus dans le RTOS

X. Hardware / Software Co-Design

❑ Partitionnement

Les types de partitionnement dans le Co-Design

- ***Partitionnement fonctionnel*** : si le partitionnement fonctionnel est effectué, la partition peut être implémentée dans le matériel ou le logiciel.

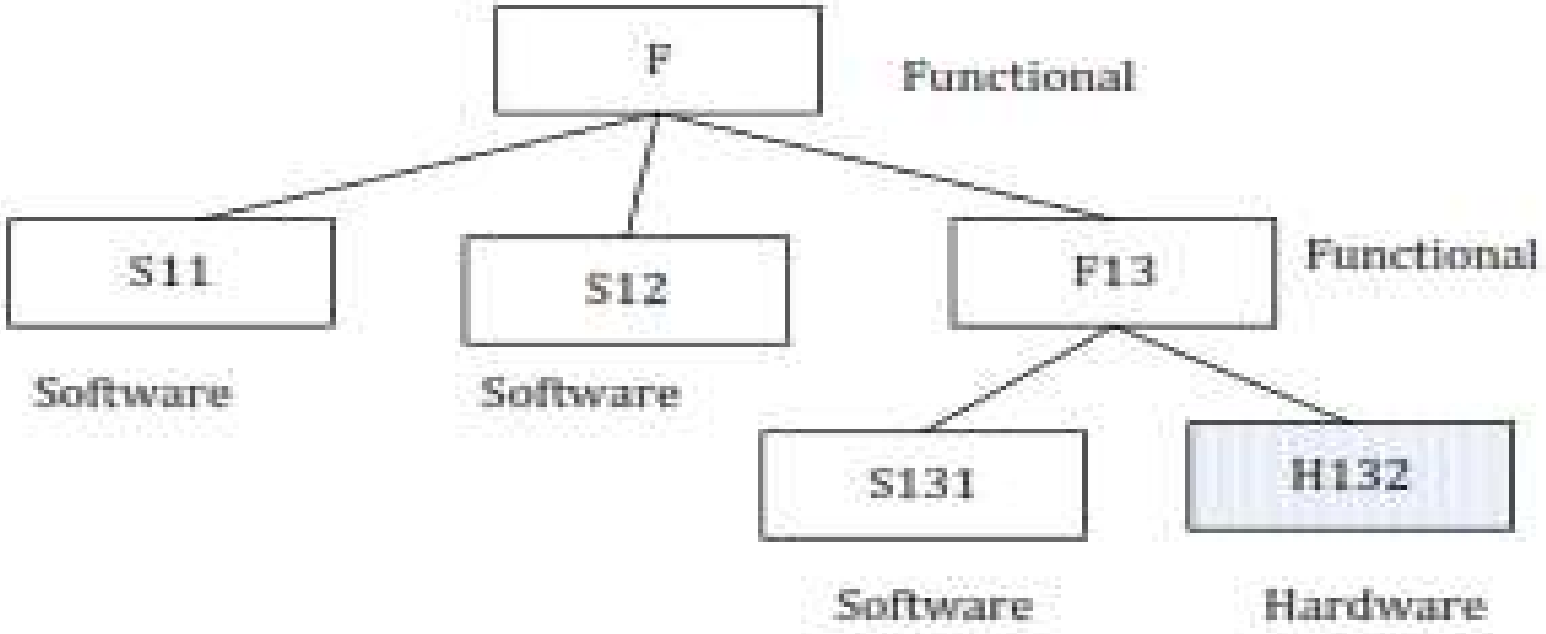
- ***Partitionnement orienté logiciel*** : appliquez une approche orientée logiciel où vous commencez avec toutes les fonctionnalités dans la partie logiciel et déplacez certaines parties critiques dans le temps sur le matériel.

- ***Partitionnement orienté matériel*** : une autre approche consiste à démarrer toutes les fonctionnalités dans le matériel et déplacer des parties vers le logiciel lorsque l'implémentation matérielle n'est pas rentable, et il n'y a pas de criticité temporelle.

X. Hardware / Software Co-Design

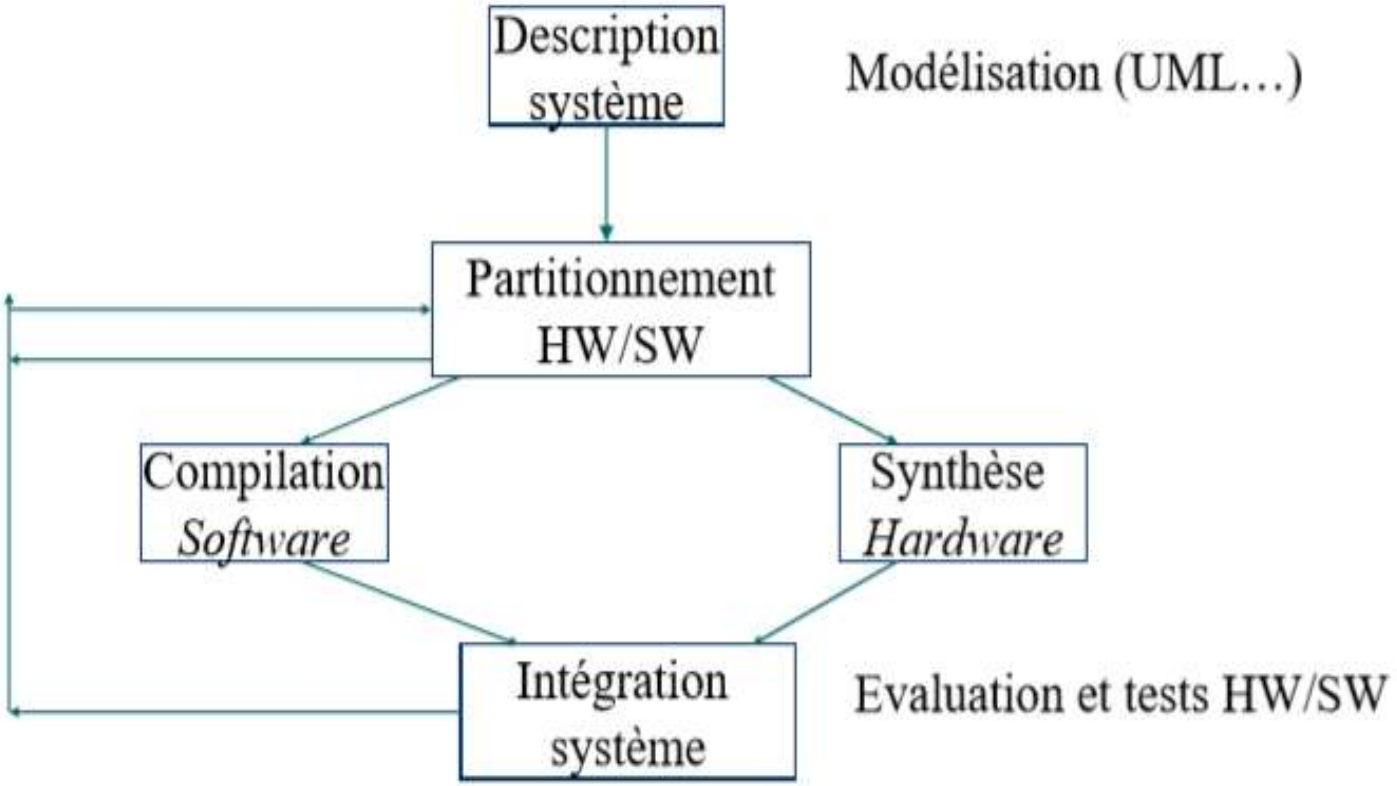
❑ Partitionnement

Exemple de partitionnement



X. Hardware / Software Co-Design

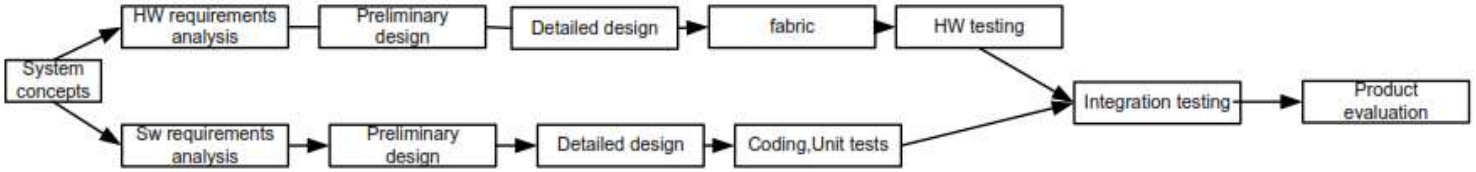
❑ Les étapes dans le Co-design



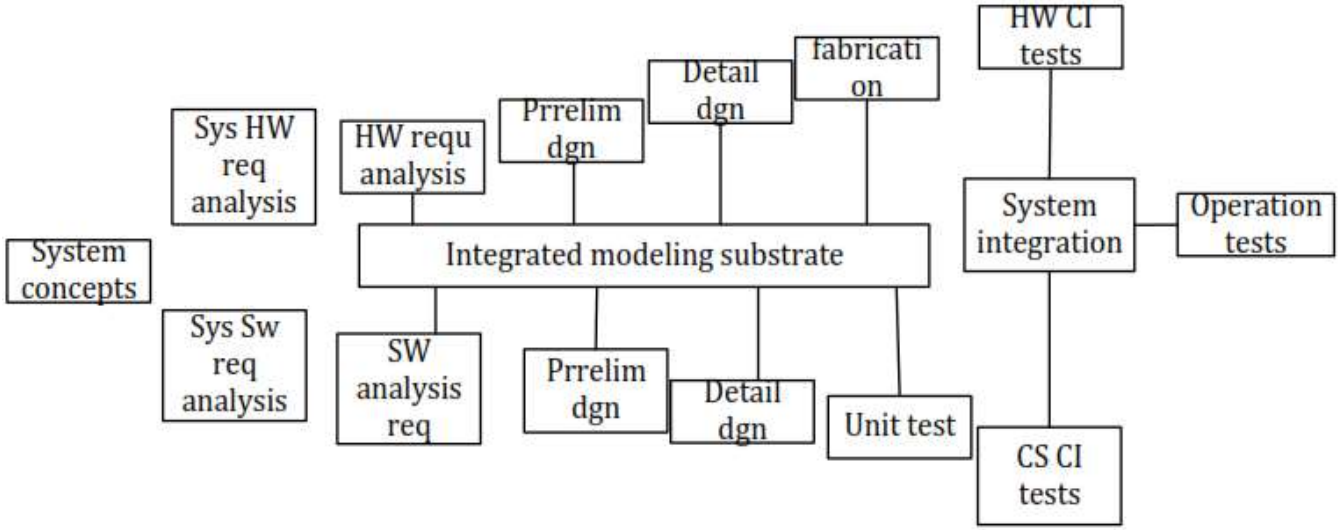
X. Hardware / Software Co-Design

❑ Partitionnement

Model conventionnel du Co-design



Process Intégrer du Co-design



Système d´exploitation embarqués

❖ Qu'est-ce qu'un système d'exploitation embarqué ?

❑ Un système d'exploitation embarqué est un système d'exploitation (OS) spécialisé conçu pour effectuer une tâche spécifique pour un appareil qui n'est pas un ordinateur. Le travail principal d'un système d'exploitation embarqué est d'exécuter le code qui permet à l'appareil de faire son travail. Le système d'exploitation embarqué rend également le matériel de l'appareil accessible aux logiciels qui s'exécutent sur le système d'exploitation.

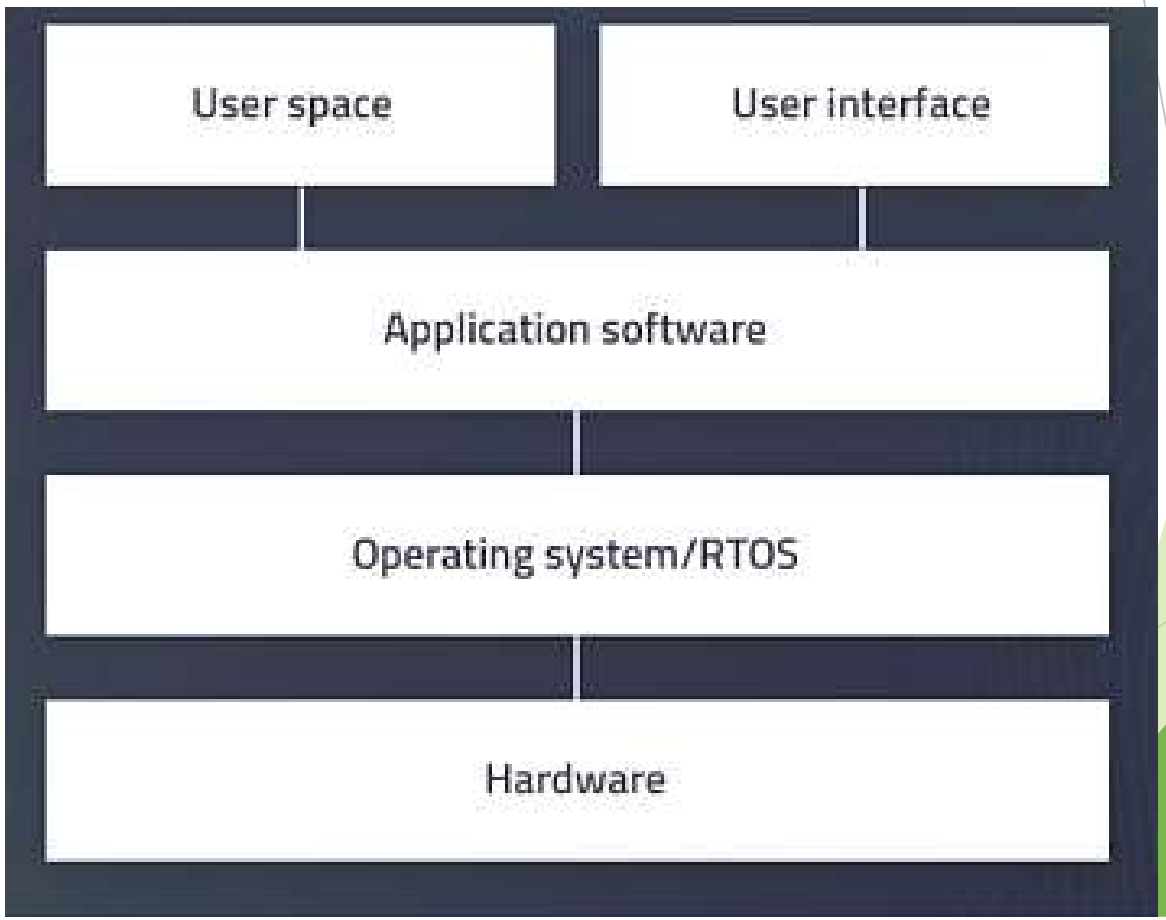
❑ Un système d'exploitation embarqué est un système d'exploitation conçu et optimisé pour :

- ➢ améliorer l'efficacité de la gestion des ressources matérielles
- ➢ réduire les temps de réponse spécifiquement pour la tâche pour laquelle l'appareil a été créé

Système d´exploitation embarqués

- Comment le système d'exploitation s'intègre dans un système embarqué

OS Embarqué



❖ Comment fonctionne un OS Embarquée ?

- Un système d'exploitation Embarqué permet à un système Embarqué de faire son travail au sein d'un système plus large. Il communique avec le Hardware du système embarqué pour exécuter une fonction spécifique. Par exemple, un ascenseur peut contenir un système Embarqué, tel qu'un microprocesseur ou un microcontrôleur, qui lui permet de comprendre sur quels boutons le passager appuie. Le logiciel embarqué qui s'exécute sur ce système est le système d'exploitation Embarqué.

- Contrairement à un système d'exploitation pour un ordinateur à usage général, un système d'exploitation Embarqué a des fonctionnalités limitées. Selon le système en question, le Os peut n'exécuter qu'une seule application embarquée. Cependant, cette application est probablement cruciale pour le fonctionnement du systèmes . Compte tenu de cela, un système d'exploitation embarqué doit être fiable et capable de fonctionner avec des contraintes de mémoire et de puissance de traitement.

❖ Avantage et désavantage

❑ Avantages du système d'exploitation Embarqué :

- ➢ Le système d'exploitation est souvent peu coûteux.
- ➢ Le système d'exploitation a tendance à utiliser peu de ressources, y compris une puissance minimale.
- ➢ La performance est généralement sans problème.

❑ Inconvénients du système d'exploitation embarqué :

- ➢ Le système d'exploitation ne peut généralement exécuter qu'une seule ou très peu d'applications.
- ➢ Il est difficile de modifier le système d'exploitation une fois que vous avez établi un cadre/Framework et que vous l'avez Embarquée
- ➢ Dépanner le système d'exploitation lorsqu'il y a des problèmes peut être difficile.

❖ OS Embarqué Vs OS Ordinaire:

caractéristique	*OS Embarqué*	*OS ordinaire*
But principal	Exécuté une seule application	Exécuté plusieurs applications
Distribution d´application	Généralement l´application et le system sont distribuées comme une seul image	L´application et le système sont distribuées séparément
Architecture	Désignée pour une application spécifique	Désignée pour un usage générale
Processeur	CPU ou MCU	CPU

❖ Classement des systèmes d´exploitations Embarquées

Il existe des dizaines de systèmes d'exploitation embarqués. Certains sont des systèmes qui fonctionnent pour des gammes variées de systèmes . De nombreux systèmes d'exploitation embarqué sont plus populaires pour certains appareils et industries. La conception et les sélections de votre matériel auront un impact considérable sur la sélection du système d'exploitation .Deux façons de classer les systèmes d'exploitation embarqués sont les suivantes :

- s'ils fonctionnent sur des microprocesseurs ou des microcontrôleurs
- si ils sont utilisés, en particulier pour des industries ou des appareils spécifiques.

❖ Systèmes d'exploitation embarqués : fonctionnant sur des microprocesseurs ou des microcontrôleurs

❑ OS embarqués sur microprocesseurs :

- Embedded Linux, especially Yocto and various vendor specific derivates of it
- WebOS
- Android
- Desktop Linux/Windows
- QNX
- Integrity
- VxWorks
- Ubuntu and Debian
- Windows for IoT
- Embedded Configurable Operating System (eCos)

❖ Systèmes d'exploitation embarqués : fonctionnant sur des microprocesseurs ou des microcontrôleurs

❑ OS embarqués sur microprocesseurs :

- FreeRTOS
- Zephyr
- QNX
- Integrity
- VxWorks
- Embedded Configurable Operating System (eCos)
- "Bare metal" on microcontrollers

❖ Types de systèmes d'exploitation embarqués

Les différents types de systèmes d'exploitation sont les suivants :

- **Single system control loop** :est le type le plus simple de système d'exploitation embarqué. C'est tellement comme le système d'exploitation mais il est conçu pour exécuter une seule tâche unique. Il est encore en débat que ce système doit être classé comme un type de système d'exploitation ou non. Un exemple serait le contrôle de la température dans une maison intelligente. Un thermostat intelligent mesure la température dans la maison et si elle dépasse la limite fixée par l'utilisateur, éteint le chauffage.

- **Multi-Tasking Operating System** :Comme son nom l'indique, ce système d'exploitation peut effectuer plusieurs tâches. Dans un système d'exploitation multitâche, plusieurs tâches et processus s'exécutent simultanément. Plusieurs fonctions peuvent être exécutées si le système possède plusieurs cœurs ou processeurs. Le système d'exploitation est commuté entre les tâches. Certaines tâches attendent des événements tandis que d'autres reçoivent des événements et sont prêtes à être exécutées. Si l'on utilise un système d'exploitation multitâche, le développement logiciel est simplifié car les différents composants du logiciel peuvent être rendus indépendants les uns des autres.

❖ Types de systèmes d'exploitation embarqués

- **Rate Monotonic Operating System**: Il s'agit d'un type de système d'exploitation qui garantit que les tâches exécutées dans un système peuvent s'exécuter pendant un intervalle de temps spécifique et pendant une période de temps spécifique. Lorsqu'il n'est pas assuré, il y a une notification d'échec au logiciel système pour prendre les mesures appropriées.

- **Preemptive Operating System:** Un système d'exploitation préemptif est un type de système d'exploitation multitâche qui interprète la prédominance préemptive pour les tâches. Une tâche de priorité plus élevée est toujours définie et exécutée avant une tâche de priorité inférieure. De tels systèmes d'exploitation multitâches sont efficaces pour augmenter la réponse du système aux événements et simplifient également le développement logiciel rendant le système plus fiable. Le concepteur du système peut être en mesure de calculer le temps pris par le planificateur pour changer de tâche.

Types de systèmes d'exploitation embarqués

- **Real-time operating system:** Un système d'exploitation temps réel est conçu pour être réactif. Il traite les entrées lorsqu'elles sont reçues et répond dans un délai spécifique. Si le temps de réponse tombe en dehors de la période spécifiée, le système peut tomber en panne. Les systèmes d'exploitation en temps réel utilisent parfois une planification monotone, qui attribue des priorités aux tâches.

Choix d´un OS Embarqué:

Les Éléments importants à évaluer lors du choix d'un système d'exploitation pour un système embarqué :

- Quelles sont les compétences et l'expérience de votre équipe de développement existante ? cela peut être une considération importante lors de la comparaison de deux systèmes d'exploitation qui pourraient autrement fonctionner pour votre projet. "Vous ne trouverez peut-être pas (votre équipe a) d'expérience avec QNX ou un autre système d'exploitation en temps réel. Mais vous trouverez toujours des personnes qui connaissent Linux, au moins un peu.
- De quelles API aurez-vous besoin ? Les API permettent au système de communiquer avec les applications.
- Quelle est la sécurité du système d'exploitation ?
- Le système d'exploitation doit-il prendre en charge les périphériques dont votre système embarqué peut avoir besoin ?
- Le systèmes aura-t-il une interface graphique ou un écran ?
- Quelle est l'histoire et la force du système d'exploitation ?. "Donc, cela a également un impact sur votre prise de décision : pensez-vous que ce fournisseur sera toujours là dans 10 ans ? Ou pensez-vous que vous pouvez trouver quelqu'un qui sait quelque chose à propos de ce (système) ?"

Choix d´un OS Embarqué:

- Quelles sont les limites de mémoire et les besoins du système?

- Quelles fonctionnalités le système d'exploitation fournit-il, et lesquelles avez-vous ou n'avez-vous pas besoin ? Quel impact pouvez-vous avoir sur les fonctionnalités suivantes ?
  - Système de fichiers
  - Gestion des threads et des processus
  - Nettoyage et gestion de la mémoire
  - API pour les graphiques et la gestion du processeur
  - Gestion du cycle de vie des applications,
  - lanceur d'applications, éléments de l'interface utilisateur du système
  - Aspects de sécurité (divisés en plusieurs autres bits)
  - Capacité de mise à jour (à la fois OS + BSP (package de support de carte)); Support OTA
  - Disponibilité d'autres Middleware, par ex. un serveur HTTP ou une base de données SQL
  - Les périphériques prennent en charge les API (appareil photo, Bluetooth, Wi-Fi, etc.)

- Comment l'intégration matérielle fonctionnera-t-elle avec le système d'exploitation ?

❖ Exemple :OS Embarqué

❖ Linux Embarqué:

- Embedded Linux fait référence à l'utilisation de Linux dans des robots, des routeurs, des cartes de prototypage ou tout autre appareil électronique doté d'un microcontrôleur (MCU). Embedded Linux diffère de Linux principalement par sa taille, car une grande partie du système n'est pas nécessaire pour votre appareil embarqué.

- Avec Embedded Linux, vous avez une variété d'options. Vous pouvez construire vous-même l'image système en utilisant Yocto Project ou Buildroot. Vous pouvez également utiliser une image prédéfinie comme OpenWrt si vous travaillez avec des routeurs, ou Ångström si vous avez une carte de prototypage comme BeagleBoard ou Raspberry Pi. En outre, vous pouvez utiliser une distribution plus connue telle que Debian, Ubuntu ou même Android (veuillez noter qu'Android n'est pas une distribution Linux, mais utilise Linux comme noyau). De plus, selon la carte que vous choisissez d'utiliser, vous pouvez avoir une image Linux déjà distribuée par le fabricant.

- Lorsque nous avons besoin d'un système d'exploitation avec suffisamment de flexibilité et d'options de personnalisation, nous pouvons choisir la version Linux qui correspond le mieux à nos besoins.

❖ Exemple :OS Embarqué

❖ Linux Embarqué:

❑ Sécurité, performances et fonctionnalités du développement Linux embarqué

Vous pouvez obtenir d'excellentes performances avec Embedded Linux. Cependant, en raison d'une variété d'options, cela dépendra de la distribution que vous choisissez ou de la manière dont vous souhaitez la construire.

En termes de sécurité, Linux est un système d'exploitation très sécurisé. Néanmoins, si vous choisissez de le construire, vous devrez le configurer vous-même dans la plupart des cas.

- Licence gratuite et logiciel open source
- Prend en charge environ 150 processeurs
- Véritable multitâche
- Noyau temps réel
- Prise en charge de nombreux systèmes de fichiers
- Mode protégé pour que les programmes ou les utilisateurs ne puissent pas accéder aux zones non autorisées.
- Mise en réseau avec TCP/IP et d'autres protocoles.
- Capacité multi-utilisateurs.
- De nombreuses bibliothèques partagées et outils de développement

❖ Exemple :OS Embarqué

❖ Linux Embarqué:

❑ Avantages

- licence gratuite
- pas cher
- Il a beaucoup de documentation
- Il offre une assistance gratuite et payante
- Il est hautement personnalisable

❑ Inconvénients

- Il peut être difficile à utiliser
- Vous devez implémenter la sécurité
- Si vous choisissez de construire votre propre Linux, les performances dépendront de la façon dont vous configurez votre système

❖ Exemple :OS Embarqué

❖ QNX:

Développé à l'origine par Quantum Software Systems au début des années 80. Maintenant, il appartient à Blackberry. QNX est un Real Time OS embarqué destiné à développer des systèmes de mission critiques et est un système d'exploitation basé sur un micro-noyau de type UNIX.

Certaines des caractéristiques QNX les plus pertinentes sont :

- Architecture micro-noyau
- Prise en charge du multicœur
- Prend en charge l'entrée multi-touch et la capture vidéo
- Démarrage sécurisé
- Système de fichiers crypté
- Système de fichiers à vérification automatique
- Certifié POSIX PSE52
- Wi-Fi 802.11 a/b/g/n
- Prise en charge complète des piles IPv6 et IPv4
- Prise en charge USB 3.x, hôte
- Bluetooth v4.2 classique et basse consommation
- Prise en charge des derniers processeurs ARMv8 et x86-64Prise en charge continue de 32 bits pour ARMv7 et x86

❖ Exemple :OS Embarqué

❖ QNX:

❑ Avantages du système embarqué QNX

- C'est un système d'exploitation en temps réel
- Il supporte de nombreuses architectures
- Vous pouvez porter l'application Linux sur QNX

❑ Inconvénients du système embarqué QNX

- Il est sous licence privée
- C'est assez cher

❑ QNX est utilisé dans les domaines suivants :

- Automobile
- Machinerie lourde
- Contrôle industriel
- Équipement médical
- Robotique

❖ Exemple :OS Embarqué

❖ Integrity:

Integrity est un RTOS certifié POSIX développé par Green Hills. L'un des aspects les plus pertinents de ce système est le fait qu'il s'agit d'un système d'exploitation en temps réel dur.

INTÉGRITÉ est livré avec :

- Prise en charge Wi-Fi : WPA2, Bluetooth, 3G
- Pv4/IPv6 et piles de routage
- FFS, FAT, NFS
- Graphiques 2D, 3D et OpenGL
- Certification de sécurité et de sûreté pour les dispositifs médicaux, les systèmes de contrôle ferroviaires, les systèmes de contrôle industriels, l'automobile, etc.
- Prend en charge de nombreuses architectures telles que x86, Power Architecture, ARM, MIPS, OMAP, etc.
- Temps réel dur
- Prise en charge multicœur

❖ Exemple :OS Embarqué

❖ Integrity:

INTEGRITY propose son propre kit de développement : MULTI. Il prend en charge C, C++, Embedded C++ et ADA comme langages de programmation. Nintendo a utilisé MULTI comme environnement de développement pour la console de jeu Wii U

❑ Avantages du système d'exploitation Integrity

- C'est un système d'exploitation dur et en temps réel
- Il est teste sur beaucoup d'appareils
- Il prend en charge une grande variété d'architectures CPU et MCU

❑ Inconvénients du système d'exploitation Integrity

- C'est assez cher (l'environnement de développement MULTI a un coût de 6000 $ par développeur)
- Il peut être fastidieux de trouver la bonne source pour la documentation
- Le MULTI IDE est difficile à utiliser et est plus lent que ses concurrents

❖ Exemple :OS Embarqué

❖ Integrity:

Le système d'exploitation INTEGRITY est utilisé dans les domaines aérospatial, automobile, militaire, industriel, médical et grand public. Voici quelques exemples d'appareils qui utilisaient le système d'exploitation :

- Boeing 787 Dreamliner
- Avion à réaction F-22 Raptor
- Toyota Prius
- Ford Lincoln Aviateur
- Système de gestion de contrôle des trains de Bombardier

❖ Exemple :OS Embarqué

❖ VxWorks:

Il s'agit d'un RTOS développé par Wind River Systems. VxWorks est un système propriétaire conçu pour plusieurs industries. VxWorks présente les caractéristiques suivantes :

- Prise en charge de monocœur et multicœur
- Séparation entre le noyau et les environnements d'espace utilisateur
- Conforme POSIX
- Prise en charge des architectures ARM, IA-32, Intel 64, MIPS, PowerPC, SH-4, StrongARM, xScale
- Prise en charge des langages de programmation C++ 11, 17 et 20, Python et Rust
- Prise en charge des systèmes de fichiers dos-FS et HRFSPi
- IPv4/IPv6 à usage général et en temps réel
- Certifications de sécurité

❖ Exemple :OS Embarqué

❖ VxWorks:

❑ Avantages du système d'exploitation VxWorks

- ➢ Il a un simulateur
- ➢ Il prend en charge les bibliothèques multiplateformes
- ➢ Il prend en charge de nombreuses architectures 32 et 64 bits
- ➢ Il teste sur beaucoup d'appareils
- ➢ La documentation est disponible sur leur site

❑ Inconvénients du système d'exploitation VxWorks

- ➢ Seuls quelques langages de programmation sont disponibles
- ➢ Il a un léger problème de sécurité

❖ Exemple :OS Embarqué

❖ VxWorks:

VxWorks est utilisé dans une longue liste d'appareils intégrés. Un exemple est le robot Asimo de Honda ou les Curiosity Rovers de la NASA. Voici quelques autres domaines où le système embarqué a été utilisé jusqu'à présent :

- Vaisseau spatial : Curiosity Rover, le Mars Reconnaissance Orbiter, la sonde spatiale Deep Impact, le SpaceX Dragon, entre autres.
- Avion : Airbus A400M Airlifter, Boeing 787
- Télescopes spatiaux
- Automobile : systèmes de navigation européens Volkswagen RNS 510, système de télémétrie pour voitures de course Bosch Motorsports
- Robots industriels
- Test et mesure, Transport, Contrôleurs, Systèmes de stockage…….

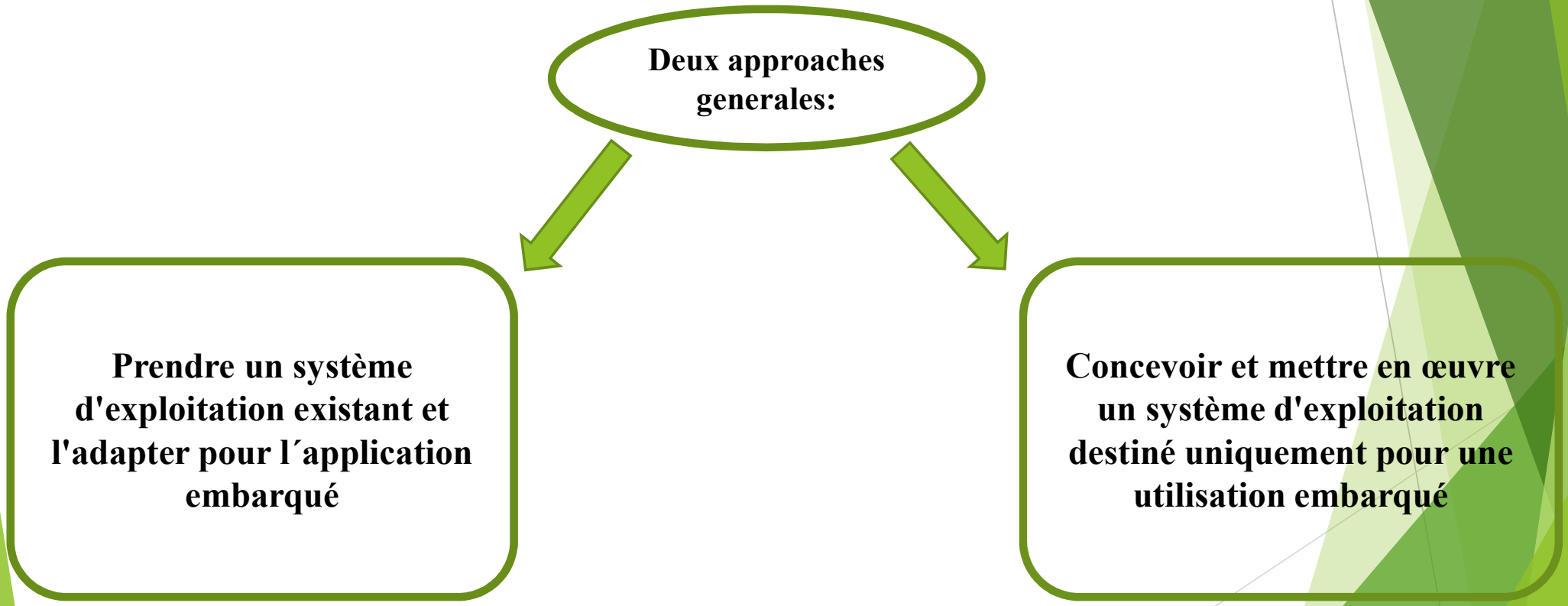
جـامعة الحسن الثـانـي بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

ENSEM

Chapitre2 : Architecture et développement des OS embarqué

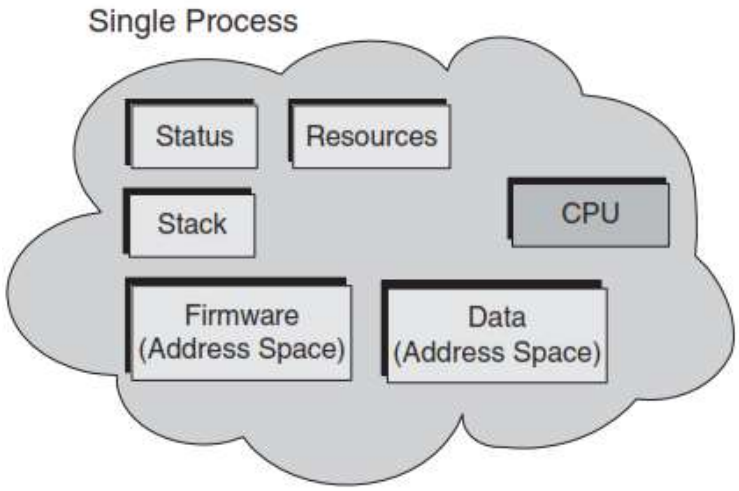
Pr. MOUZOUNA YOUSSEF

❖Développement d´un OS Embarqué:



❖Process et Tasks:

- Un ***process*** ou ***Task*** est un programme en cours d'exécution. Un programme peut être défini comme un ensemble d'instructions. Le programme est une entité passive et le processus est une entité active
- Un ***Process*** ou ***Task*** sont deux termes interchangeable dans les systèmes d´exploitation
- Lorsqu'un processus est créé, un certain nombre de ressources lui sont allouées par le système d'exploitation. Ceux-ci peuvent inclure une pile de processus, un espace d'adressage mémoire, des registres (via le CPU), un compteur de programme, des ports d'E/S, des connexions réseau,, etc. Ces ressources ne sont généralement pas partagées avec d'autres processus.
- L'instruction en cours d'exécution (identifiée par la valeur du compteur de programme) et les valeurs actuelles des données associées en mémoire ou dans les registres sont collectivement appelées ***Process state***.



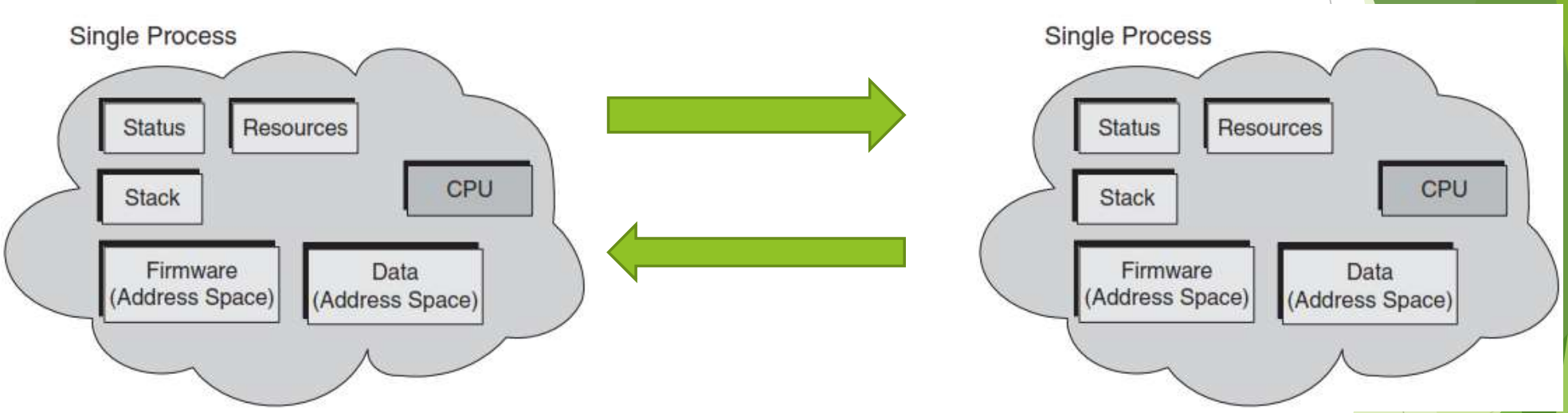
❖Process et Tasks:

❑ Si un second Task est ajoutée au system

des problèmes potentiels de conflit de ressources surviennent. Généralement, il n'y a qu'un seul processeur et les ressources restantes sont limitées.

Le CPU, est donnée à une tâche pendant un court instant puis à l'autre. Si chaque tâche partage les ressources du système dans les deux sens, chacune peut terminer son travail. Si le processeur est passé assez rapidement entre les tâches, il semblera que les deux tâches l'utilisent en même temps. Le temps d'exécution du programme sera prolongé. Un tel schéma est appelé *Multitasking* (multitâche) ; on dit que les tâches s'exécutent *Concurretly*

❖Process et Tasks:



Multitasking

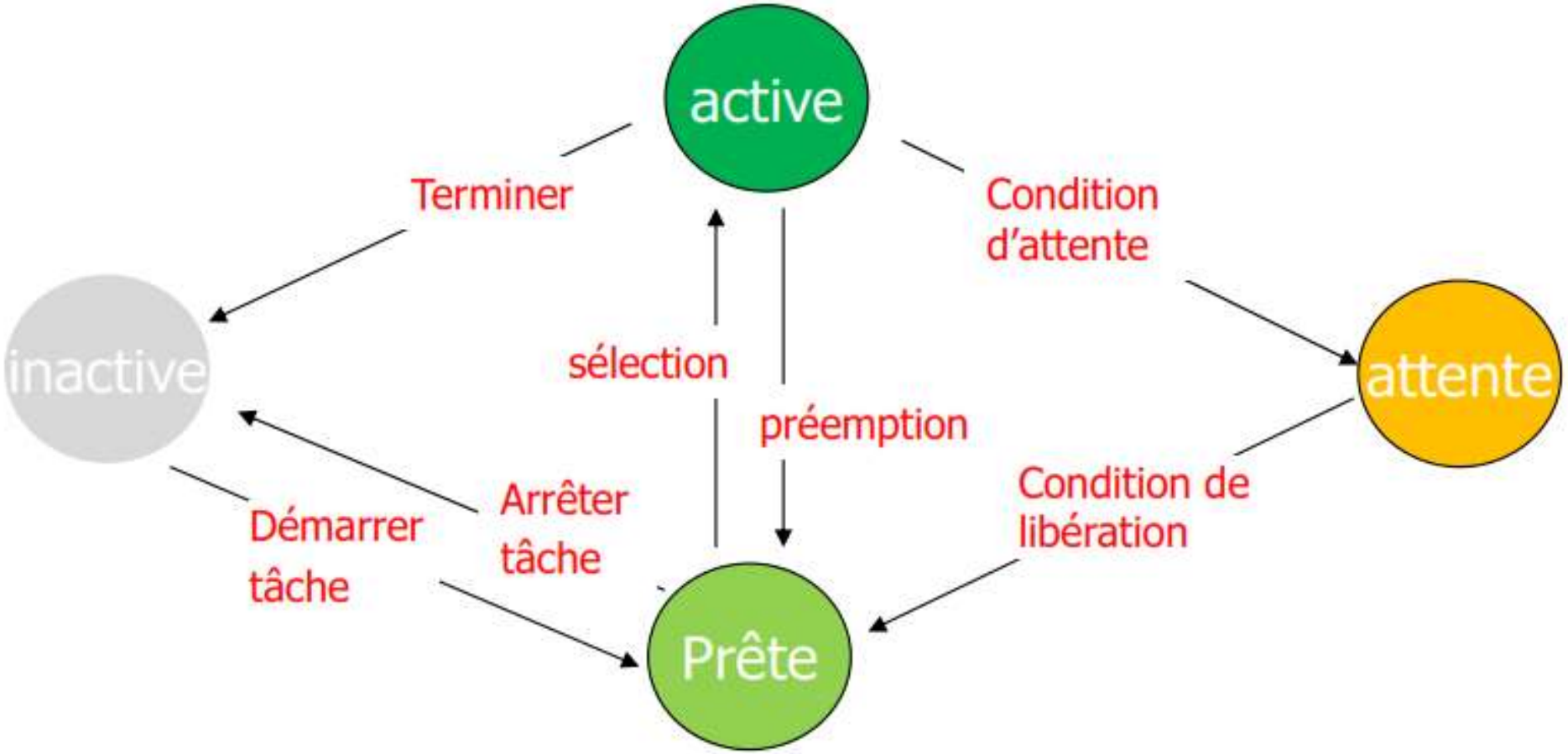
❖Process et Tasks: Changing context/ Dynamique des taches

Le noyau du SE place chaque tâche dans 4 états d'exécution possibles

- ***Active*** : est en cours d'exécution (1 seule à la fois et par CPU)
- ***Prête*** : en attente de temps CPU (il peut y avoir plusieurs)
- ***Suspendue*** : en attente d'un évènement pour la rendre prête (il peut y en avoir plusieurs)
- ***Inactive***: ne participe plus à l'activité du système (il peut y en avoir plusieurs).

un ***ordonnancement / Schedule*** est mis en place pour spécifier quand, dans quelles conditions et pendant combien de temps chaque tâche recevra l'utilisation du CPU (et d'autresRessources).

❖Process et Tasks: Changing context/ Dynamique des taches



❖Process et Tasks: Changing context/ Dynamique des taches

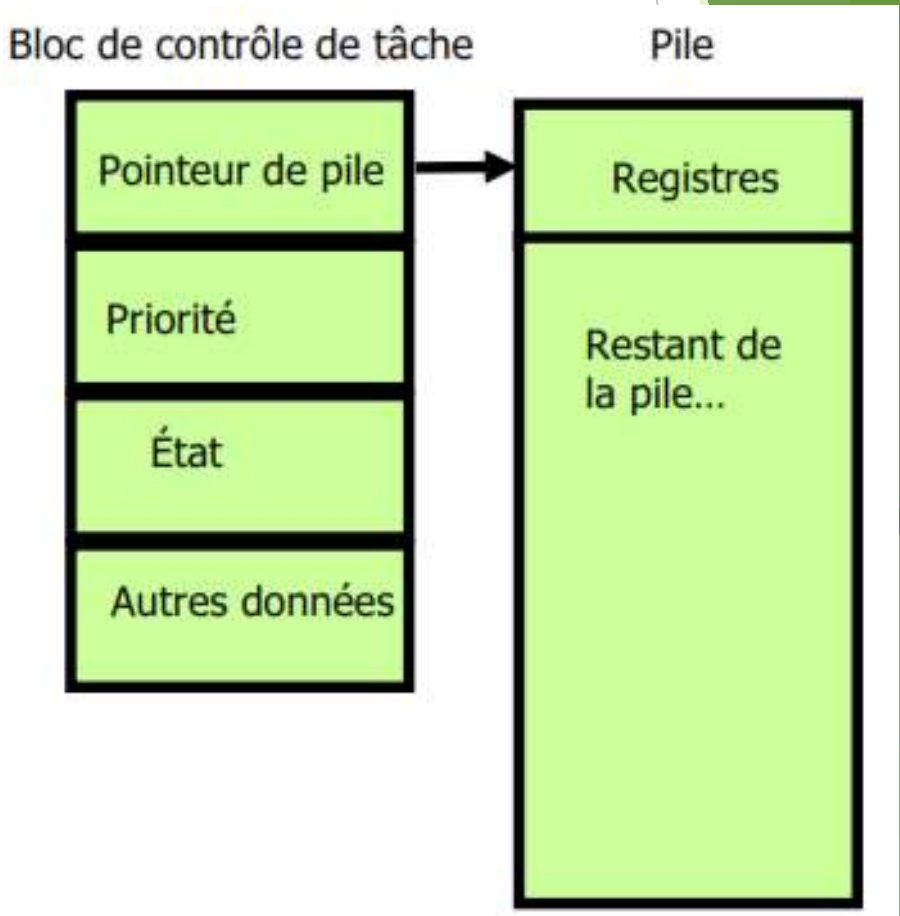
Les critères pour décider quelle tâche doit être exécutée ensuite sont collectivement appelés une planification stratégie. Ces stratégies se répartissent généralement en trois catégories :

1. ***Multiprogrammation/ Multiproraming***, dans laquelle la tâche en cours d'exécution continue jusqu'à ce qu'elle exécute une opération nécessitant l'attente d'un événement externe (par exemple, l'attente d'un événement d'E/S ou de l'expiration d'un temporisateur).
2. ***Temps réel/ Real Time***, dans lequel les tâches avec des délais temporels spécifiés sont garanties de se terminer avant l'expiration de ces délais. Les systèmes utilisant un tel schéma nécessitent une réponse à certains événements dans un temps bien défini et contraint.
3. ***Le temps partagé/ Time Sharing***, dans lequel la tâche en cours d'exécution doit abandonner le CPU afin qu'une autre tâche puisse avoir un tour. Dans le cadre d'une stratégie à temps partagé, un Timer est utilisé pour anticiper la tâche en cours d'exécution et rendre le contrôle au système d'exploitation. Un tel schéma permet de s'assurer de manière fiable que chaque processus dispose d'une tranche de temps pour utiliser le système d'exploitation.

❖Process et Tasks: Bloc de contrôle des taches

Structure indiquant le statut d'une tâche dans l'environnement d'exécution

- Contenus des registres du CPU
- Contenu des pointeurs de programme, de données et de la pile
- Identifiant, priorité, droits d'accès, et état d'exécution courant



❖Process et Tasks: Ordonnancement/ Scheduling

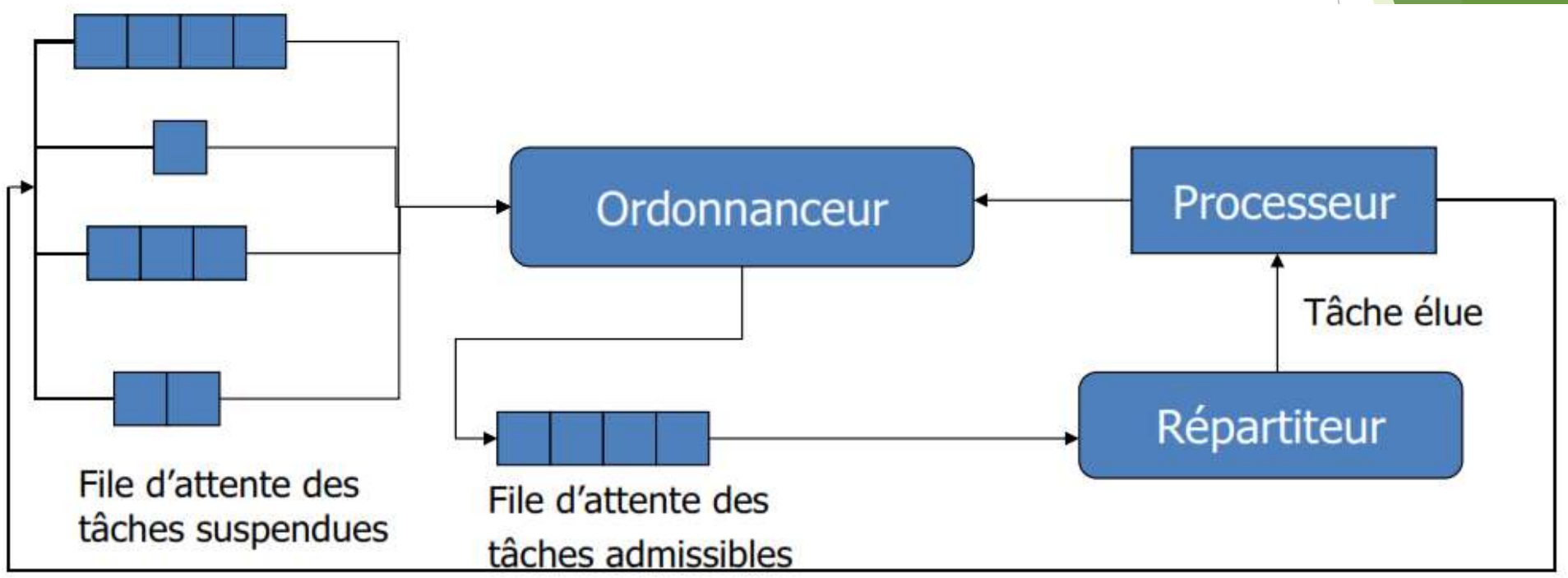
- ❖ Fonction majeure du noyau
- ❖ Fait par un répartiteur logiciel qui intervient à chaque fois qu'une modification d'état doit a voir lieu
  - ➢ Alloue le CPU à une tâche parmi celles admissibles
  - ➢ Gère le déblocage de tâches en attente et la commutation de tâches actives

- ❖ Mécanismes
  - ➢ Préemption de la tâche active suite à un évènement (droit de priorité)
  - ➢ Allocation de temps de CPU égalitaire (round robin)
  - ➢ Commutation de contexte programmée

❖Process et Tasks: Ordonnancement/ Scheduling

Repartiteur / Dispatcher

❖ Réalise le choix de la tâche active et le changement de contexte associé



❖Process et Tasks: Ordonnancement/ Scheduling

Commutation de tache et commutation de context

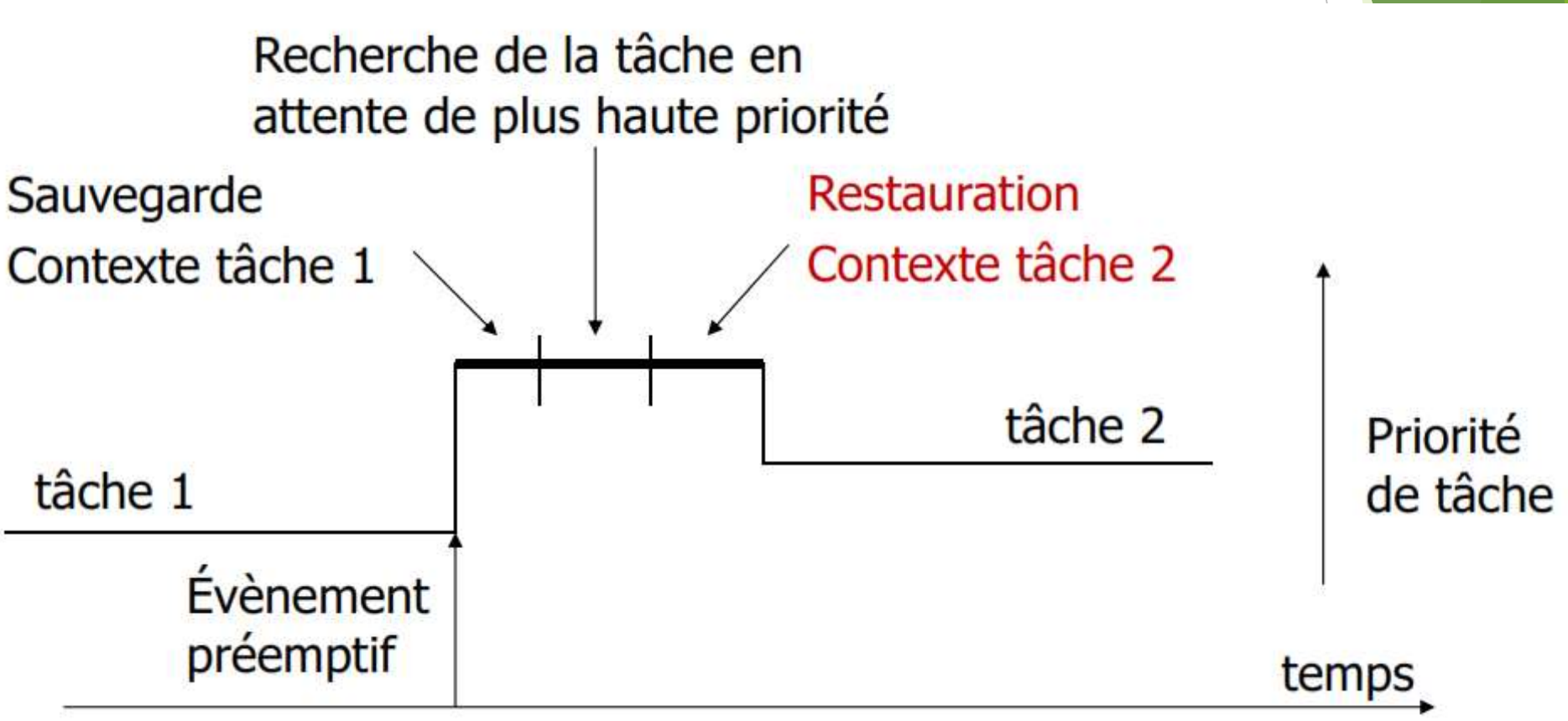
- ❖ Commutation des tâches
  - ➢ Arrêt du traitement de la tâche en cours au profit d'une autre
  - ➢ Provoquée par :
    - demande explicite de la tâche en cours (attente,..)
    - décision de l'ordonnanceur (tâche plus prioritaire active)
    - réponse prioritaire à un phénomène externe

- ❖ Commutation de contexte
  - ➢ Sauvegarde de l'état des registres du processus courant et restauration de l'état des registres d'un nouveau processus
  - ➢ Effectuée lors d'un appel au noyau ou lors de l'occurrence d'une interruption

❖Process et Tasks: Ordonnancement/ Scheduling

Commutation de tache et commutation de context

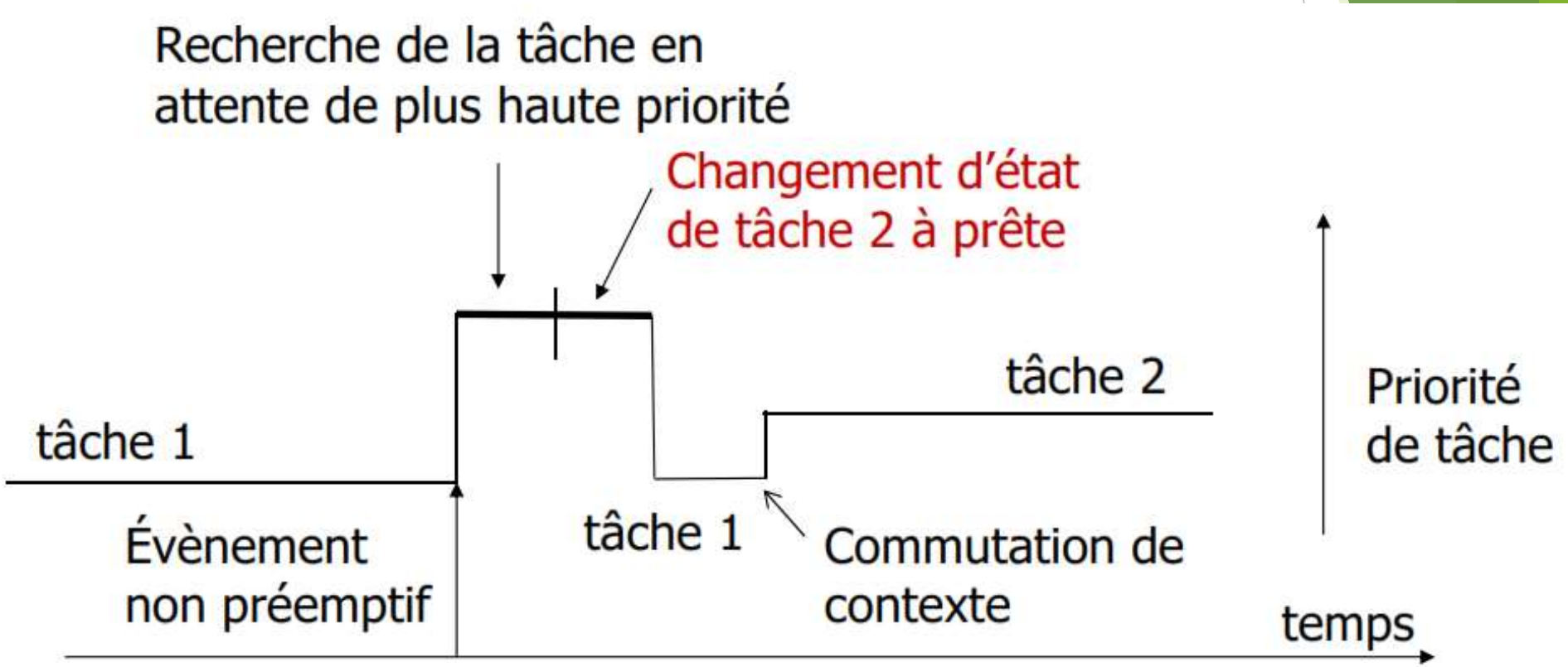
❖ Modèle Préemptif:



❖Process et Tasks: Ordonnancement/ Scheduling

Commutation de tache et commutation de context

❖ Modèle Non Préemptif:



❖Process et Tasks: Ordonnancement/ Scheduling

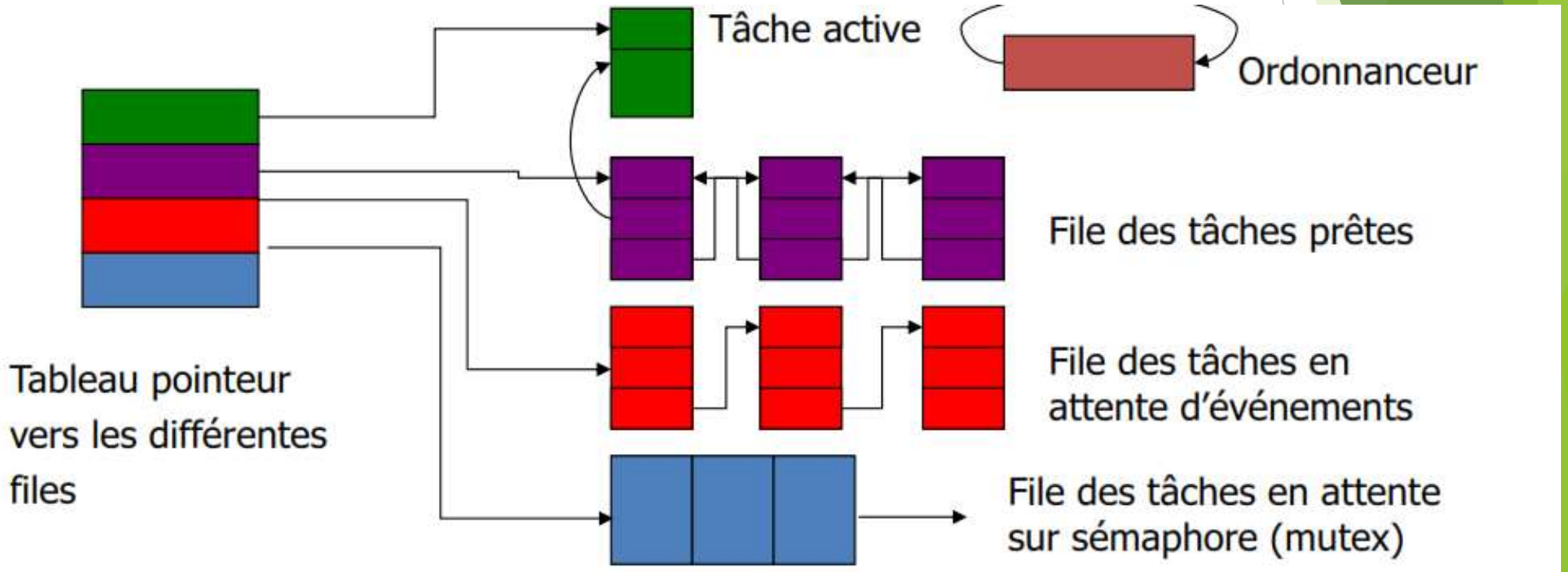
Critères de sélection :

- La politique de sélection de l'ordonnanceur répond à des contraintes spécifiques :
  - Temps de CPU donné pour chaque tâche
  - Respect de l'ordre de priorité des tâches
  - Respect d'un temps de réponse donné
  - Préemption d'une tâche qui monopolise le processeur

- En général, l'invocation de l'ordonnanceur s'effectue sur:
  - Mise en attente ou réveil d'une tâche
  - Changement de priorité d'une tâche
  - Réquisition du CPU au profit d'une tâche plus prioritaire

❖Process et Tasks: Ordonnancement/ Scheduling

- ❖ La manière de gérer les files d'états à une incidence sur les performances d'un système.
- ❖ LIFO (pile) ou FIFO (liste, file, queue)
  - ➢Les tâches prêtes sont organisées en files doublement chaînées par l'intermédiaire de leur blocs descripteurs



❖Gestion d´interruption

- ❖ Gestion de plusieurs périphériques: minuterie (timer), moteurs, capteurs, disques, etc.
- ❖ Requêtes asynchrones signalées par des interruptions

- ❖ 2 types d'interruptions:
  - Interruptions matérielles
  - Interruptions logicielles

- ❖ Le code exécuté lors d'une interruption est dicté par le CPU à l'aide du vecteur d'interruption. Mais l'OS intervient pour:
  - Connecter une adresse mémoire à chaque ligne d'interruption
  - Que faire après avoir servi une interruption

- ❖ Gestion de l'aspect temps réel.

❖Process et Tasks: Communication Inter process

- ❖ Sémaphores:
  - Synchro à travers 2 opérations atomiques P et V
  - Bas niveau
  - Exclusion mutuelle assurée par le programmeur

- ❖ Moniteurs
  - Mécanisme de haut niveau
  - Exclusion mutuelle assurée par le compilateur

- ❖ Passage de messages
  - Transfert de données entre processus
  - Mise en tampon des messages

❖Gestion de la mémoire

- ❖ Allocation: Allouer à chaque tâche la mémoire dont elle a besoin
- ❖ Mapping: Faire la correspondance entre la mémoire physique et l'adressage utilisé par les tâches.
- ❖ Protection: Etablir un ensemble de comportements à adopter lorsqu'une tâche utilise de la mémoire non allouée.
- ❖ Implémentation des mécanismes de gestion.

❖Un OS Embarqué:

Un système d'exploitation embarqué fournit un environnement dans lequel les tâches qui composent une application embarquée, sont exécutés. Le moyen le plus simple de voir un système d'exploitation est peut-être du point de vue des services qu'il peut fournir. En interne, les systèmes d'exploitation varient considérablement tant dans la conception que dans la stratégie de prestation de ces services.

Il doit assurer ou supporter trois fonctions spécifiques:

1. Planifiez (schedule) l'exécution de la tâche.
2. Distribuez (Dispatch) une tâche à exécuter.
3. Assurer la communication et la synchronisation entre les tâches.

The Scheduler détermine quelle tâche s'exécutera et quand elle le fera. ***The Dispatcher*** effectue les opérations nécessaires pour démarrer la tâche, et ***the intertask ou interprocess communication*** est le mécanisme d'échange de données et d'informations entre tâches ou processus sur la même machine ou sur des machines différentes. ***The kernel*** est la plus petite partie du système d'exploitation qui fournit ces fonctions.

❖OS Embarqué: Caractéristiques

- Real-time operation
- Reactive operation
- Configurability
- I/O device flexibility
- Streamlined protection mechanisms
- Direct use of interrupts

❖Un OS Embarqué:

Dans un système d'exploitation embarqué, ces fonctions doivent être assurées:

• Gestion des processus ou des tâches

la gestion des tâches implique la création et la suppression de processus utilisateur et système ainsi que la suspension et la reprise de ces processus. La façon dont la gestion des tâches est gérée détermine, en grande partie, si le système d'exploitation peut être défini comme temps réel ou pas. Les responsabilités supplémentaires incluent la gestion de la communication inter-processus et des interblocages (Deadlocks). Les Deadlokcs surviennent lorsque deux tâches ou plus ont besoin d'une ressource détenue par une autre tâche.

• Gestion de la mémoire

les services de gestion de la mémoire incluent le suivi et le contrôle des tâches chargées en mémoire, la surveillance des parties de la mémoire utilisées et par qui, l'administration de la mémoire dynamique si elle est utilisée.

❖Un OS Embarqué:

Dans un système d'exploitation embarqué, ces fonctions doivent être assurées:

• Gestion du système d'E/S

La gestion des entrées et sorties du système peut inclure un large éventail de responsabilités. Une application embarquée doit interagir avec une grande variété d'appareils différents. Dans les systèmes plus complexes, une telle interaction se produit via un logiciel spécial appelé pilote de périphérique (Device Driver). Dans un système bien conçu, le côté interne de ce logiciel possède ce qu'on appelle une interface d'appel commune - API. La motivation d'une telle approche est de permettre au logiciel d'application d'interagir avec chacun des différents dispositifs de la même manière. Le système d'exploitation doit gérer l'interaction entre chacun de ces appareils et les utilisateurs ou les tâches de l'application.

Gestion du système de fichiers

Comme son nom l'indique, les responsabilités de gestion du système de fichiers sont dirigées vers la création, la suppression et la gestion des fichiers et des répertoires.

❖Un OS Embarqué:

Dans un système d'exploitation embarqué, ces fonctions doivent être assurées:

• Protection du système

Comme indiqué précédemment, assurer la protection des données et des ressources dans le contexte de processus simultanés est un devoir important et essentiel pour le système d'exploitation. Une telle obligation est plus aiguë dans le contexte d'une machine de von Neumann.

• La mise en réseau

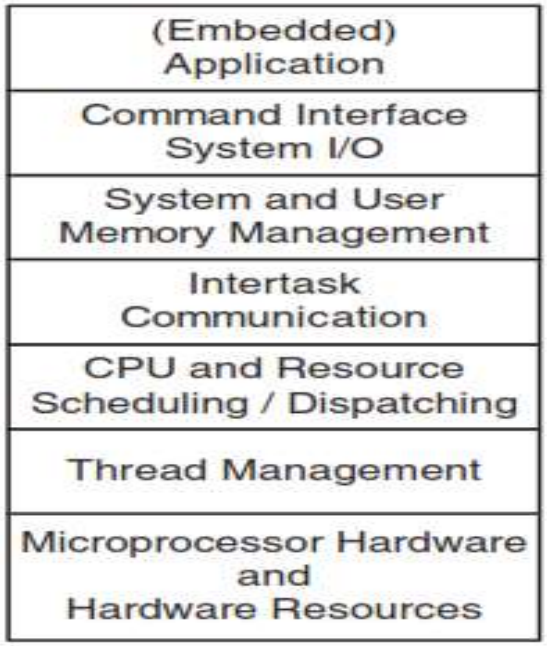
Dans le cadre d'une application distribuée, le système d'exploitation doit également prendre en charge la gestion de la communication distribuée intra système et l'ordonnancement à distance des tâches.

• Interprétation des commandes

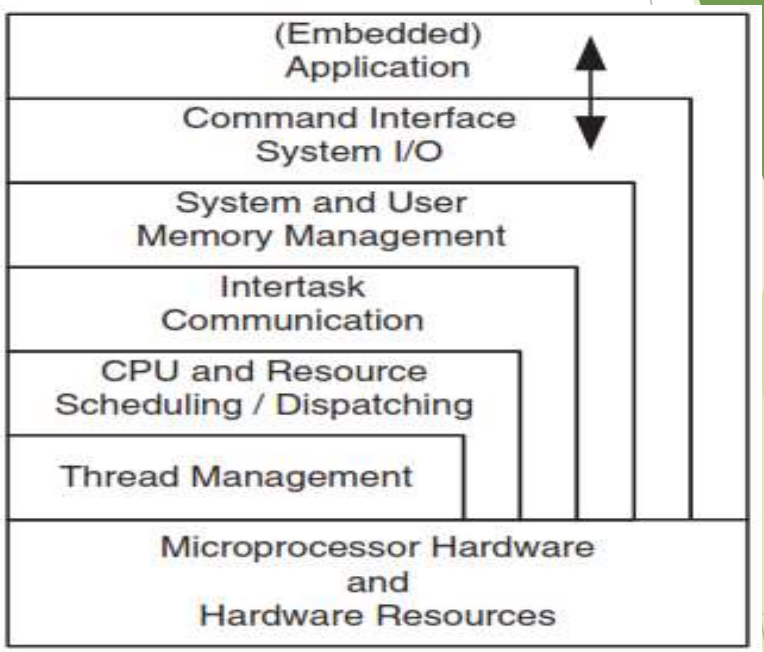
Au fur et à mesure que les commandes et les directives entrent dans le système, elles doivent être analysées, vérifiées pour l'exactitude grammaticale et dirigées vers la tâche cible.

❖Architecture :

Deux grande architectures



OS virtual machine model



Typical Embeded OS architecture

❖Noyaux embarqués : Monolithic

- Le SE monolithique : **ensemble procédures, chacune pouvant appeler toute autre à tout instant.**
- Pour effectuer un appel système, on dépose dans un registre les paramètres de l'appel et on exécute une instruction spéciale appelée **appel superviseur** ou **appel noyau.**
- Son exécution commute la machine du mode utilisateur au mode superviseur ou noyau et **transfère le contrôle au SE.**
- Le SE analyse les paramètres déposés dans le registre mentionné plus haut et en déduit la procédure à activer pour satisfaire la requête.
- A la fin de l'exécution (procédure système), le SE rend le contrôle au programme appelant.

❖Noyaux embarqués : Monolithic

Un système monolithique est composé de trois couches:

C1- Une **procédure principale qui identifie la procédure de service** requise.

C2- Des **procédures de service qui exécutent les appels système.**

C3- Des **procédures utilitaires qui assistent les procédures système.**

Une procédure utilitaire peut etre appelée par plusieurs procédures systèmes.

❖Noyaux embarqués : Monolithic

OS Monolithique (plus ancien):

- Simple/ne consomme pas beaucoup de ressources
- Convient aux « petits systèmes » ou quelques portions de systèmes temps réel complexe
- OS → entièrement en mode privilégié
- L'application utilise un appel système pour accéder aux services de l'OS → procédure exécutée
- Gestion de l'interruption : optimisée car pas de changement de contexte entier (prioritaire car l'ordonnanceur est désactivé)
- Impossible de mettre à jour l'application « à chaud » (remplacement + reboot)

❖Noyaux embarqués : Monolithic

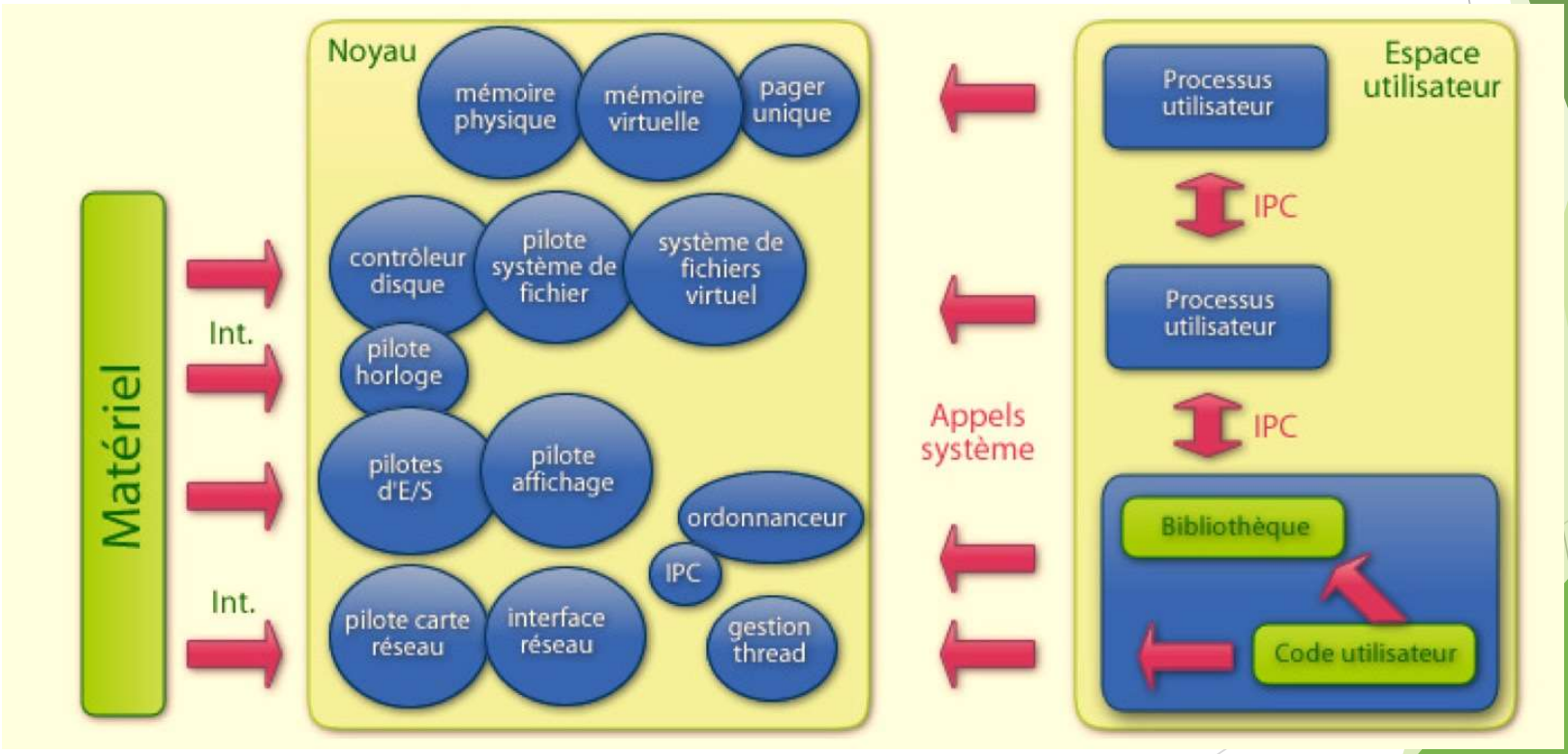
❑ Structure de base:

- Un programme principal qui invoque la procédure du service
- Un ensemble de procédures de services qui gèrent les appels système
- Un ensemble de procédures utilitaires auxiliaires des précédentes

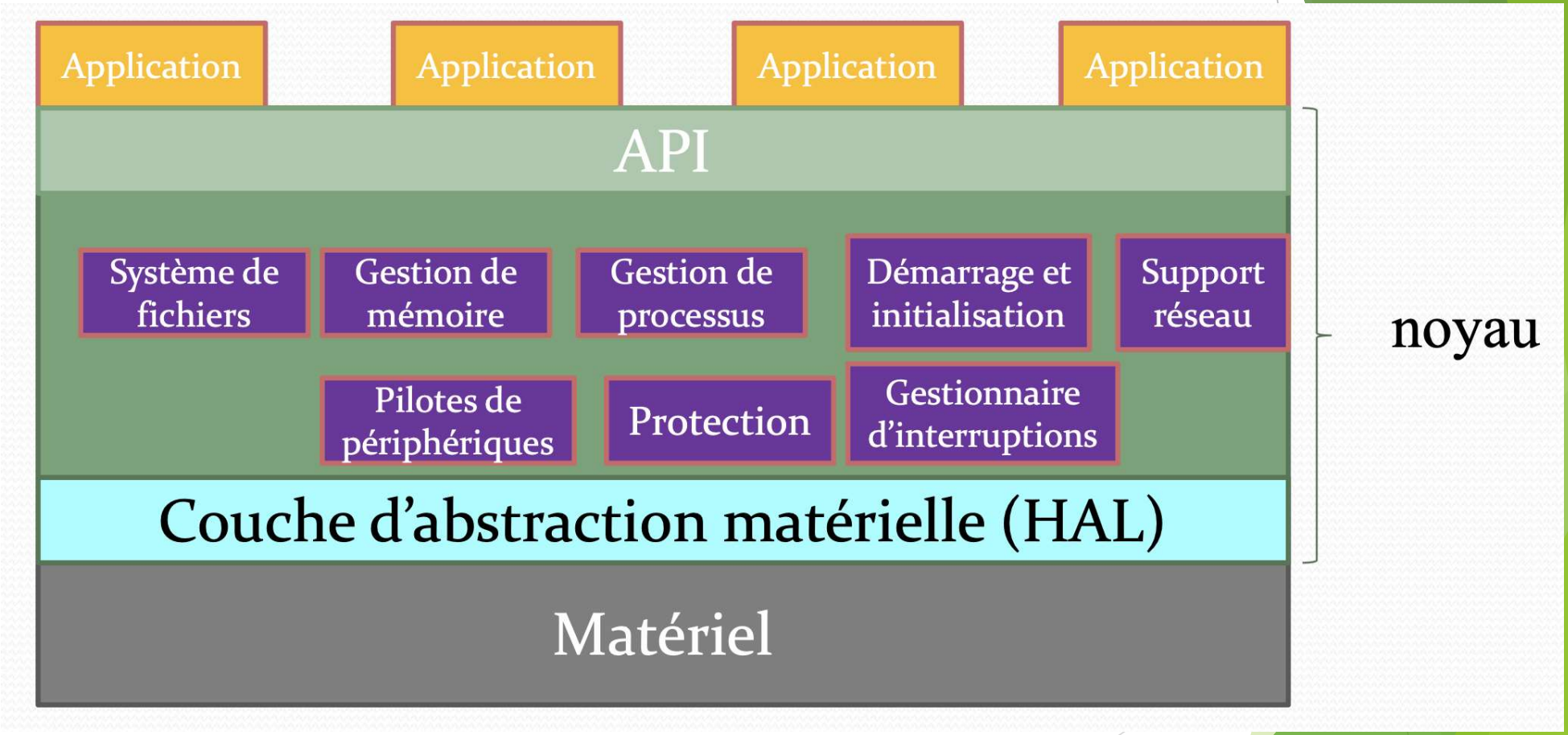
Exemples : Vielles version d'UNIX (FreeBSD, SOLARIS), DOS, ….

❖Noyaux embarqués : Monolithic

Certains OS (qlq versions de **Linux, ou certains vieux Unix) ont un noyau monolithique.** CAD l'ensemble des fonctions du système et des pilotes sont regroupés dans **un seul bloc de code et un seul bloc binaire généré à la compilation**



❖Noyaux embarqués : Monolithic



❖Noyaux embarqués : Monolithic

❑ Avantages

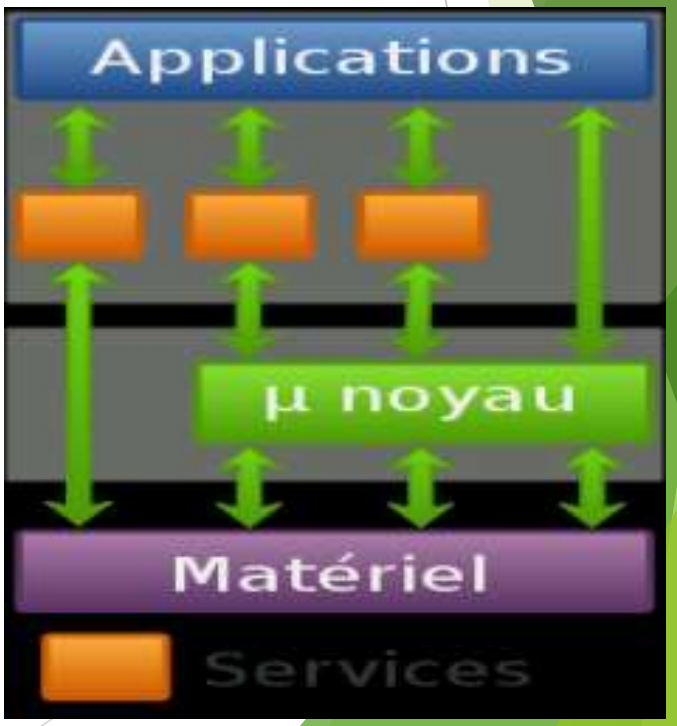
- De meilleures performances
- Vite développe´ ...
- Évolution: chargement dynamique (et donc sélectif) des modules

❑ Inconvénients

- Extension difficile
- Code non modulaire
- Plus c'est gros, moins c'est performant!
- Nid de bugs
- Peu fiable (un bug " redémarrage)

❖Noyaux embarqués : Micronoyau

Syst u-Noyau à cherchent à min les fonctionalités dépendantes du noyau
En placant la + grde partie de l'OS à l'extérieur du noyau à user space
Ces fonctions seront assurées par des petits SRVs indépendants
Chaque petit SRV a son propre espace d'adressage et de ressources

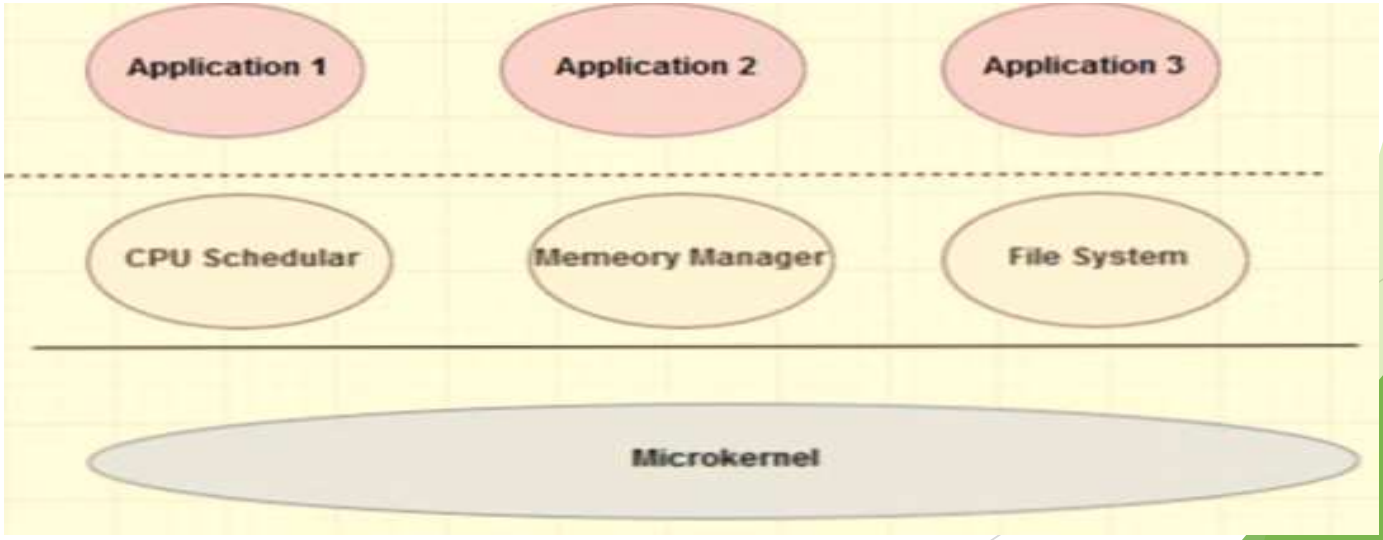


❖Noyaux embarqués : Micronoyau

Ce principe a pour avantage:

On éloigne les services 'A risque' des parties critiques de l'OS regroupées dans le noyau (à gain en robustesse , fiabilité & facilité de maintenance et évolutilivité)

Mais:
Les mécanismes de communications (IPC) nécessaires pour assurer le passage de messages entre petits SRVs peuvent devenir très lourdes et peuvent limiter les performances



❖Noyaux embarqués : Micronoyau

OS Micronoyau

- Déplace plusieurs fonctions de l'OS vers des «processus serveurs» s'exécutant en mode utilisateur à réduction au maximum de la taille du code privilégié (mode Kernel).

- Gérer les communications entre applications et serveurs pour:
  - Renforcer la politique de sécurité
  - Permettre l'exécution de fonctions système (accès aux registres d'E/S, etc.).

- Fiabilité augmentée : si un processus serveur «crash», le système continue à fonctionner et il est possible de relancer ce service sans redémarrer.

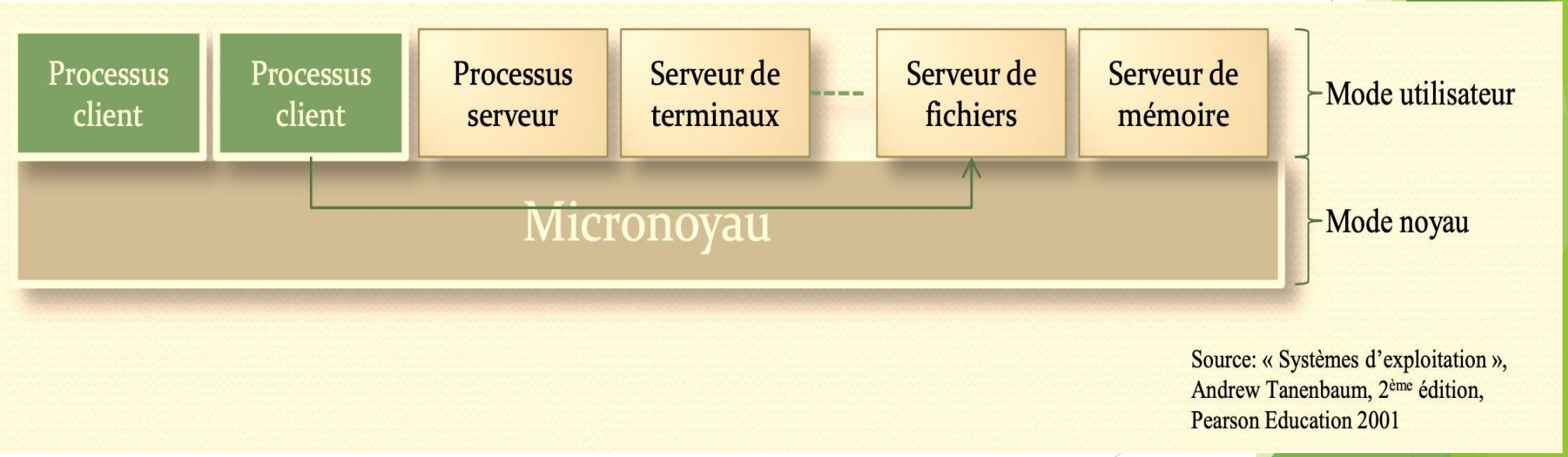
- Modèle facilement étendu à des systèmes distribués.

- Gestion de l'interruption : commutation de tache (moins efficace que le modèle monolithique).

❖Noyaux embarqués : Micronoyau

Le noyau gèrent les communications entre clients et serveurs.
Certains services sont impossibles à exécuter en mode utilisateur (pilotes de périphériques d'E/S):

- Garder certains processus serveur critiques en mode noyau
- Garder une partie du mécanisme en mode noyau en laissant le choix des politiques aux serveurs en mode utilisateur.



Source: « Systèmes d'exploitation », Andrew Tanenbaum, 2ème édition, Pearson Education 2001

❖Noyaux embarqués : Micronoyau

❑ Avantages

- Extensibilite
- Minimise le code du noyau
- Sécurite :
  - Un serveur (mode utilisateur) crashe, il sera le seul à redémarrer
- Fiabilite
  - Micronoyau: code plus petit Ž moins de bugs

❑ Inconvénients

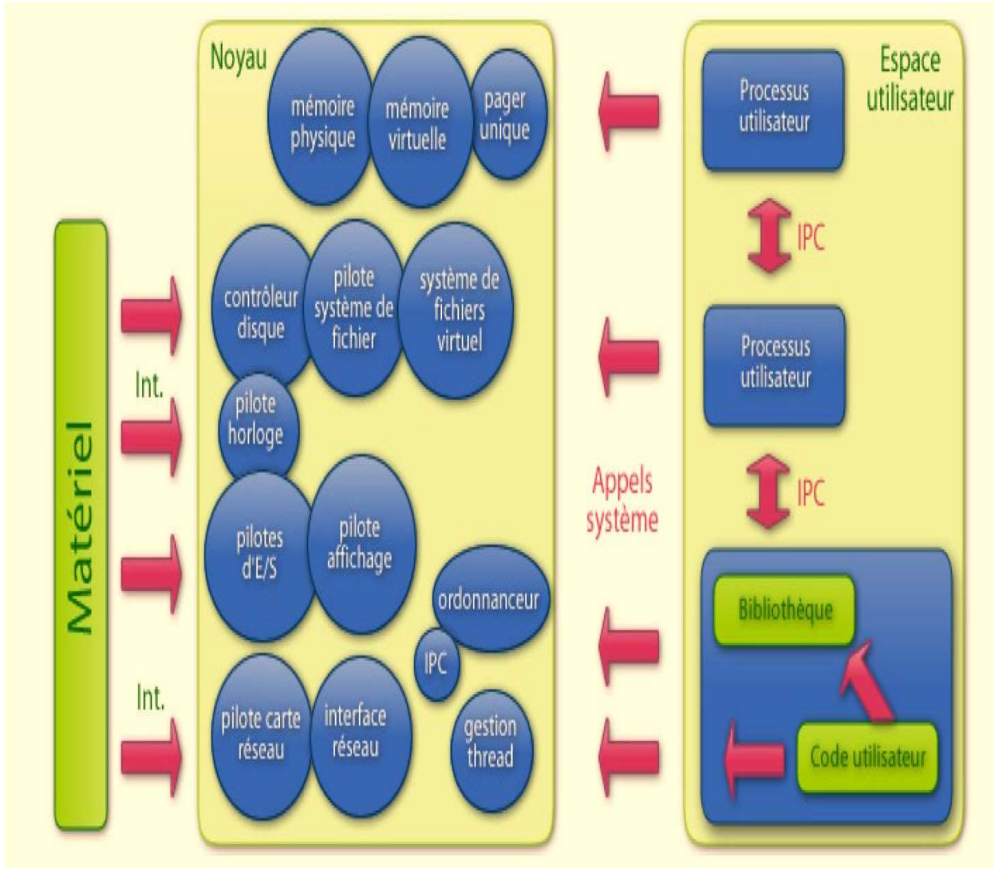
- Souvent tenté de rajouter des choses dans le noyau (vu qu'il est petit...)
- Mauvaises performances
- Requière beaucoup de prudence lors de la conception

❖Noyaux embarqués : Noyau modulaire

- Ensemble des fonctions + pilotes à regroupés dans un BLOC (code)
- Concept simple
- Excellente vitesse d'exécution
- Évolution: chargement dynamique (et donc sélectif) des modules

  - Extension difficile
  - Augmentation en taille (OS) à difficulté de maintenance
  - Augment du Nbre de pilotes hard compilés ds le noyau à Augment de l'empreint mémoire ds le noyau
  - Code de l'OS (monolithique) non modulaire à Architecture Modulaire

❖Noyaux embarqués : Noyau modulaire



❖Noyaux embarqués : Noyau modulaire

❑ C est un Noyau:

- hybride entre le noyau monolithique et le micronoyau
- performances stables et globalement bonnes
- comprend de nombreux modules, chacun dédié à une tâche spécifique
- chargement des modules en cas de besoin

❖Noyaux embarqués : Noyau modulaire

Dans le Noyau:

- Seules les parties fondamentales du noyau regroupés ds un BLOC unique (Monolithique)

Ø Les autres fonctions (pilotes matériels, ….) sont regroupés en # modules :

Ø Ces modules peuvent être séparés de point de vue code / binaire

v Cette modularité du noyau permet de charger à la demande les fonctionnalités à Améliorer les possibilités de configurations.

v La majorité des OS utilisent cette technologie

v Ces noyaux concervent les atouts des noyaux monolithiques

v La facilité de conception et de développement à maintenance et vitesse d'exécution reste excellente

v L'usage de module à découpage du code source du noyau en blocs indépendants

v Ces Blocs améliorent l'organisation et la clarté du code source

❖Noyaux embarqués : Noyau modulaire

❑ Remarque

- Ces noyaux conservent un important défaut des noyaux monolithiques purs:
- Une erreur dans un Module à mettre en danger la stabilité de tout le système
- Problème de portabilité lorsque le code augmente considérablement (Nbre de lignes de code Important).

❖Noyaux embarqués : Noyau modulaire

❑ Avantage:

- réduit le temps pour reconstruire un noyau
- diagnostique plus facile des problèmes du système
- charge les modules en cas de besoin
- réduire l'empreinte mémoire du noyau
- augmentation des performances globales et petite taille du noyau

❑ Inconvénients:

- possibilité instabilité
- vulnérabilité de sécurité
- Peut être difficile à maintenir

❖Les normes: La norme POSIX

- Portable Operating System Interface
- pour aider à produire des codes portables sans trop d'efforts
- initié par l'IEEE et organisé par l'ISO
- à partir d'UNIX, pour le C ANSI
- différents standards pour différents problèmes
- POSIX.4 : extensions temps réel
- POSIX.4a : extensions pour les threads
- PSE51: profile de système temps réel minimaliste : 1 seul processus POSIX pouvant exécuter plusieurs threads POSIX pouvant utiliser le passage de messages POSIX pour communiquer avec d'autres systèmes PS5x .
- PSE52: profile de système de controleur temps réel: PSE51+support pour un système de fichiers + E/S asynchrones
- PSE53: profile de système temps réel dédie´ : PSE51+support multiprocessus
- PSE54: profile de système temps réel polyvalent: englobe les autres profils. Il consiste de POSIX.1, POSIX.1b, POSIX.1c, et/ou POSIX.5b

important : l'interface POSIX.4 ne fournit pas un environnement temps réel, mais uniquement des interfaces qui peuvent être utilisées dans le cadre d'un OS Temps Réel

❖Les normes:

- Autres normes:

- UNIX98: normalisation de l'OS UNIX. Cette norme incorpore plusieurs des normes de POSIX
- EL/IX: API pour les systèmes embarqués. Se veut un sous ensemble des normes POSIX et ANSI.
- ITRON: norme japonaise pour les systèmes embarqués
- OSEK: norme allemande pour une architecture ouverte reliant les divers contrôleurs électroniques d'un véhicule.
- RT Spec pour Java: spécification pour un run time qui édicte des prescriptions (ramasse miettes, certaines politiques d'ordonnancement, etc.)
- Ada95: (ex: MarteOS, OpenRavenscar).
- RT Corba: un ensemble de spécification temps réel

❖Contiki OS:

- ❑ créé en 2002
- ❑ noyau modulaire
- ❑ systèmes d'exploitation embarqués open source
- ❑ conçu pour les systèmes en réseau et à mémoire limitée
- ❑ se concentrant sur les appareils à faible consommation d'énergie et à l'internet des objets
- ❑ contiki est suffisamment léger pour répondre aux exigences d'un appareil embarqué équipé d'un microcontrôleur bas de gamme, d'une petite mémoire et d'une alimentation par batterie
- ❑ un bon exemple de cela est l'empreinte de code de contiki, qui ne nécessite qu'environ 10 Ko de RAM et 30 Ko de ROM
- ❑ développement par de nombreuses équipes de développement (TI, Atmel, université d'oxford ...)

❖Contiki OS:

❑ Caractéristiques

- Protothreads partageant les fonctionnalités du multithreading et de la programmation événementielle
- Mise en réseau TCP/IP (IPv4, IPv6 via uIP et Rime)
- prise en charge des normes de faible puissance (6LowPAN, RPL, CoAP)
- chargement dynamique des modules
- Estimation de la consommation de puissance
- fonctionnant sur des appareils sans fil à faible consommation
- Simulateur : Cooja

❖Tiny OS:

❑ Caractéristiques

- ❑ open source
- ❑ basé sur un noyau monolithique
- ❑ développé pour la première fois par l'université de berkeley en 1999
- ❑ conçu pour les systèmes en réseau et à mémoire limitée
- ❑ se concentrant sur les appareils à faible consommation d'énergie et à l'internet des objets
- ❑ contiki est suffisamment léger pour répondre aux exigences d'un appareil embarqué équipé d'un microcontrôleur bas de gamme, d'une petite mémoire et d'une alimentation par batterie
- ❑ écrit en NesC
- ❑ nesC est un langage conçus pour incarner les concepts structurant et le modèle d'exécution de TinyOS. C'est une extension du langage C orientée composant ; il support alors la syntaxe du langage C et il est compilé vers le langage C avant sa compilation en binaire.
- ❑ Inexistence d´un kernel sous TinyOS
  - ➢ OS oriente composants et pas de protection mémoire
  - ➢Pas de processus, ni de systeme d´allocation de mémoire
  - ➢Gestion d´iinterruptions depend du peripherique
  - ➢Non bloquant avec quelques primitives synchronisation

❖Tiny OS: But

- ❑Permettre la forte concurrence entre plusieurs flux de donnees
  - ✓ ***Flux régulier de données captées et traitées***
- ❑Ressources limitees (memoire, puissance calcul, energie)
  - ✓ ***Utilisation efficace des ressources et communication low-power***
- ❑Adaptation aux evolutions hardware
  - ✓ ***Portabilité la plus grande possible sur différent hardware***
- ❑Supporter une tres large gamme d´applications
  - ✓ ***OS embarqué le plus modulaire possible***
- ❑Supporter un ensemble de plateformes variees
  - ✓ ***OS embarqué plutôt orienté general-purpose***
- ❑Robustesse du systeme d´exploitation
  - ✓ ***Réseau de senseurs sans surveillance pendant des mois ou années***
- ❑Limitation du buffering dans le reseau pour eviter latences
  - ✓ ***Et mémoire limitée et communication sur distance courte***

❖Tiny OS: Composant

- Systeme logiciel pour TinyOS est un ensemble de composants
  - ***Petit module qui réalise une tâche ou ensemble de tâches simple***
- Interactions entre composants et avec le hardware
  - ***De manière limitée et très bien définie***
- Seul module software toujours present est l´ordonnanceur
  - ***Il n´y a pas de kernel et donc pas vraiment d´OS***
- ***Architecture software rigide et simplifiee pour gerer le reseau***

- Plusieurs composants open source pour TinyOS
  - ***Besoins de base pour le reseau de senseurs connectee***
  - ***Single-hop networking, ad-hoc routing, power management...***
- Collection de composants standardisees forme TinyOS
  - ***La collection est l'OS avec lequel applications sont développées***

❖Tiny OS: Ordonnanceur

- Ordonnanceur qui opère a travers plusieurs composants
  - ***Une seule tâche choisie car système uniprocesseur***

- Algorithme par defaut est une simple FIFO
  - ***Peut mettre le processeur en sleep car power aware***
  - ***Un slot par tache dans la file***
  - ***Possibilité pour l´utilisateur de remplacer le scheduler***

- Deux origines possibles pour les taches a exécuter
  - ***Placée en file comme résultat d´un évènement***
  - ***Requête spécifique d´une tache en cours d´execution***

جـامعة الحسن الثـانـي بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

ENSEM

Chapitre3 :
Systèmes d'exploitation embarqués Temps Réel-RTOS

Pr. MOUZOUNA YOUSSEF

❖ Système d´exploitation temps réel?

- Définition : un programme qui ordonnance l'exécution des tâches de façon ponctuelle, gère les ressources du système et fournit les éléments de base pour développer les applications

- souvent organisé autour
  - ➢ d'un noyau fournissant les algorithmes minimum pour l'ordonnancement et la gestion des ressources éventuellement
  - ➢ de services, fournis par exemple par des modules, pour gérer les systèmes de fichier, les accès réseau, ou tout autre service requis par l'application

❖ Système d´exploitation temps réel?

Classification

- Temps réel dur ou critique (hard real-time): le non respect des contraintes temporelles entraîne la faute du système. Ex.: contrôle de trafic aérien, système de conduite de missile, etc.

- Temps réel souple (soft real-time): le respect des échéances est important mais le non respect des échéances ne peut occasionner de graves conséquences. Ex.: projection vidéo (décalage entre le son et l'image). Ex.: un robot qui capte des infos sur des objets défilant sur un convoyeur .

- Temps réel ferme (firm real-time): temps réel souple avec le manquement occasionnel des échéances. Ex.: projection vidéo (perte de quelques trames d'images).

❖ Système d´exploitation temps réel?

les composants de base du noyau d'un OS Temps Réel sont :

- l'ordonnanceur, contenu dans tous les noyaux et qui va fournir les algorithmes de base pour choisir la tâche qui va accéder au processeur

- des objets du noyau pour aider les développeurs à créer les applications : tâches ,sémaphores ,files de message...

- des services, qui sont les opérations effectuées par le noyau sur un objet ou plus généralement des opérations comme les mesures de temps, la gestion des interruptions, la gestion des ressources...

❖ Caractéristiques d'un OS Temps Réel?

5 propriétés essentielles

- ❑ fiabilité (reliability)
  - ➢ faculté de pouvoir être utilisé sans intervention extérieure
  - ➢ s'exprime en fonction du nombre de 9 de la disponibilité:

Nombre de 9	***Temps de non-disponibilité par an***	***Application typique***
3 (99,9%)	~ 9 heures	bureautique
4 (99,99%)	~ 1 heure	serveur d'entreprise
5 (99,999%)	~ 5 minutes	serveur à haute disponibilité
6 (99,9999%)	~ 31 secondes	serveur à très haute disponibilité

  - ➢ va dépendre du hardware autant que du software
  - ➢a un coût certain...

❖ Caractéristiques d'un OS Temps Réel?

5 propriétés essentielles

- prédictabilité (predictability)
  - spécifique des systèmes temps réels
  - spécifie le degré de confiance que l'on peut avoir dans la prévision du temps de réponse
- performances
  - caractérise la vitesse à laquelle le système va fournir sa réponse
  - dépend du hardware autant que du software
- compacité de l'empreinte (compactness)
  - utilisation optimale des ressources
- possibilité d'appliquer un facteur d'échelle (scalability)
  - utilisation optimale des ressources en cas de modification de la taille du système
  - optimisation des conditions de développement

❖ Tout RTOS doit avoir certaines propriétés souhaitables.

- Comme le RTOS est intégré à l'application, il fonctionne comme un système embarqué compact, principalement sur le Firemware et la mémoire sans aucun disque dur externe. L'espace mémoire occupée par RTOS doit être très petit.

- L'application intégrée et le RTOS doivent fonctionner sur différentes plates-formes matérielles avec différentes architectures de processeur. Par conséquent, le RTOS doit prendre en charge différents processeurs.

- Le RTOS doit fournir une API standard et des outils de débogage. En effet, RTOS est intégré à l'application. Le débogage doit être transparent sur l'ensemble du système.

❖ Classification des RTOS

❑ ***RTOS à noyau RT***

- ➢ Généralement conçu pour la rapidité de réponse.
- ➢ Peut être inadapté aux systèmes complexes, car priorisation de la rapidité sur la prédic tibilité
- ➢ En général propriétaire : QNX, PDOS, VCOS, VTRX32, VxWORKS

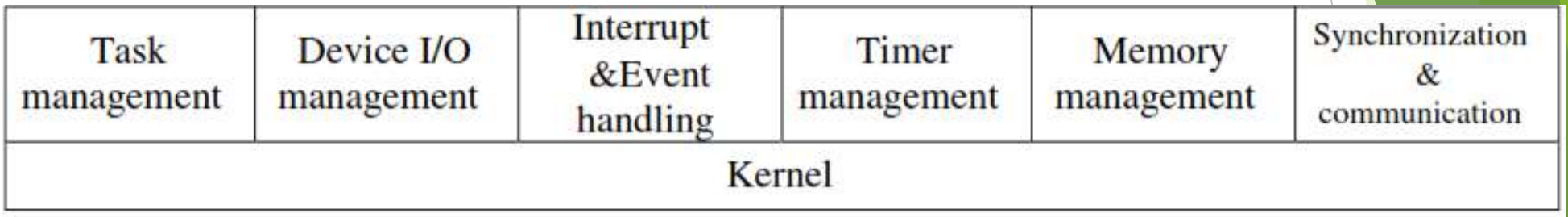
❑ ***SE standard à noyau non-RT modifié***

- ➢ Extension RT gère l'exécution des tâches et considère le noyau standard comme l'une d'elles pour les réponses en temps réel

❑ Interface de programmation d'applications (API) standard versus dédiés

- ➢ p. ex. POSIX extension-RT d'Unix, ITRON, OSEK

❖ Architecture and Organization



Real Time Operating Systems organization

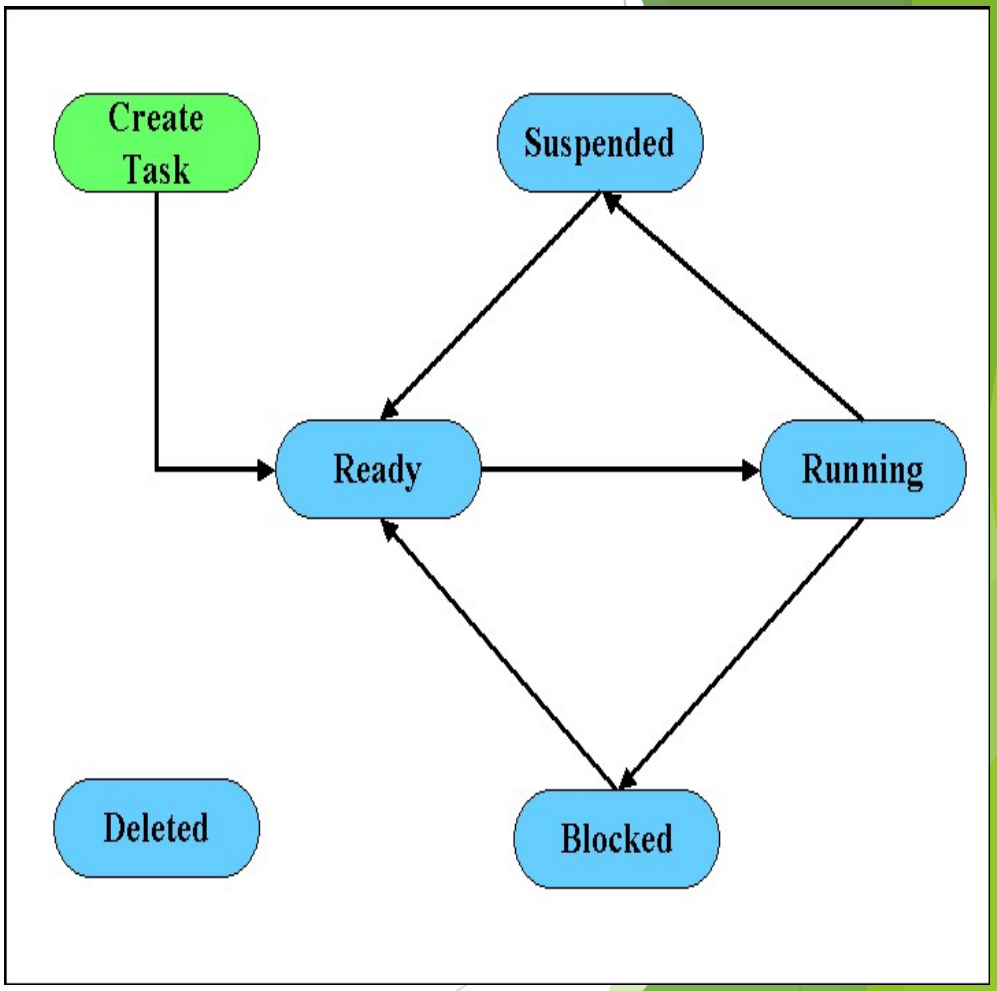
❖ Etat des Taches

Prête (Ready) : une tâche qui possède toutes les ressources nécessaires à son exécution. Elle lui manque seulement le processeur.

Active (Running) : Tâche en cours d'exécution, elle est actuellement en possession du processeur.

Attente (Blocked) : Tâche en attente d'un événement (queue de messages, sémaphores, timeout ...). Une A l'arrivé de l'événement, la tâche concernée repasse alors à l'état prêt.

Suspendu (Suspend) : tâche à l'état dormant ; elle ne fait pas partie de l'ensemble des tâches ordonnançables



❖ Priorité

chaque tache se voit attribuer une priorité qui est fonction de son caractère critique. Plus une tache est importante, plus sa priorité est élevée. Il existe deux types de priorité de taches :

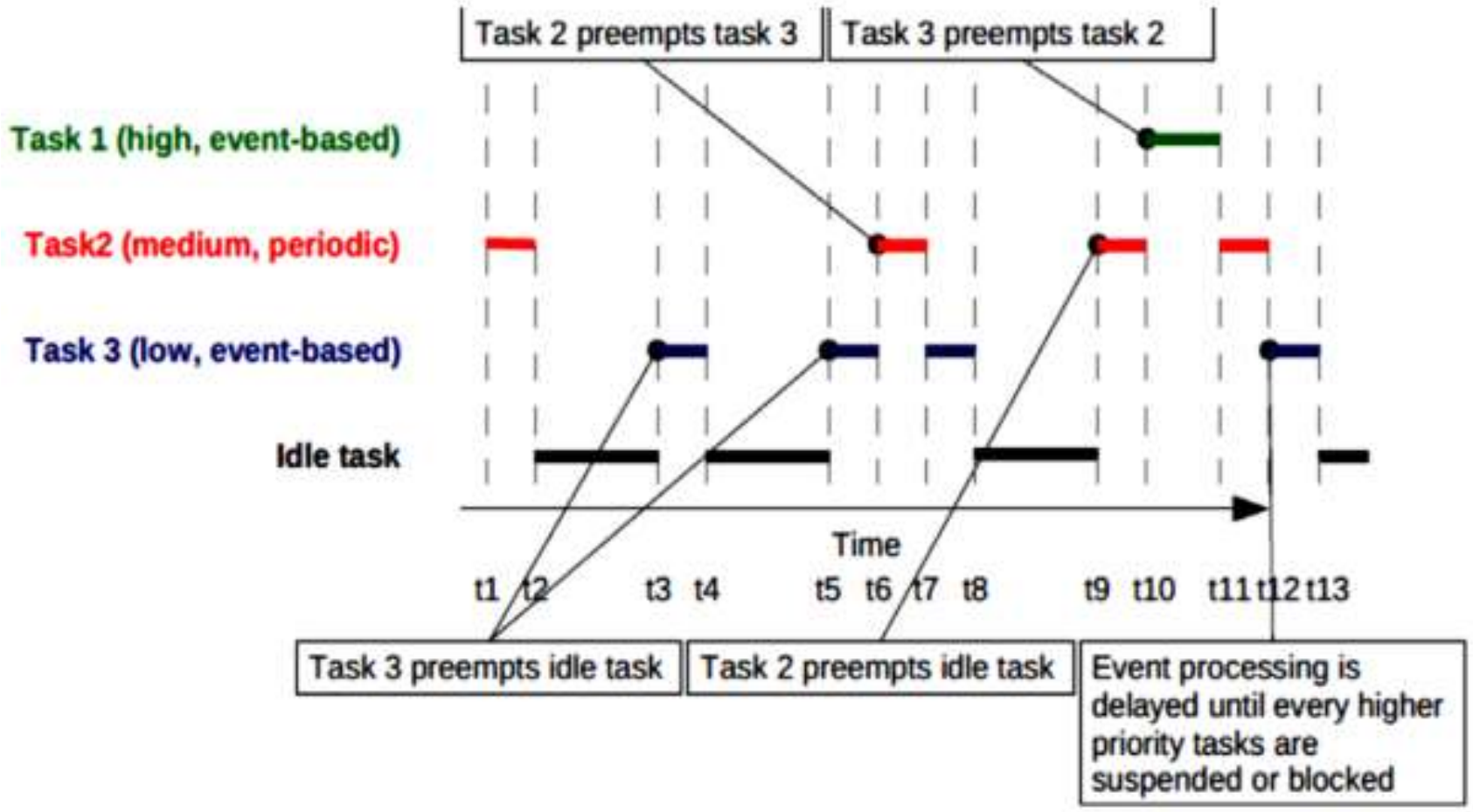
Priorité statique : la priorité de chaque tâche n'évolue pas durant l'exécution. Chaque tache reçoit une priorité fixe lors de sa création. Toutes les taches et leurs contraintes temporelles sont connues à la compilation.

Priorité dynamique : La priorité de chaque tache peut évoluer durant l'exécution. Cette caractéristique des RTOS permet d'éviter l'inversion de priorité.

Inversion de priorité

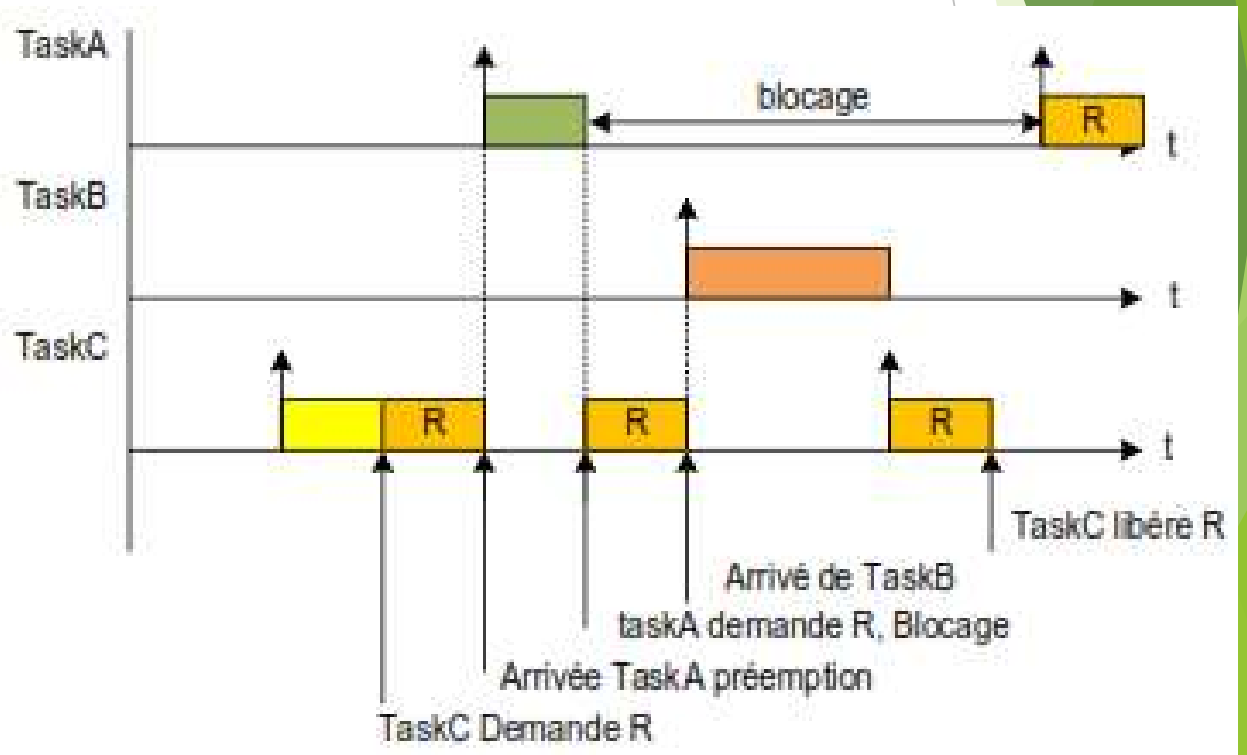
L'inversion de priorité est un problème qui survient lorsqu'une tache est suspendue dans l'attente d'une ressource contrôlée par une tache moins prioritaire. Par exemple, s'il existe deux taches T1 et T2 avec T1 plus prioritaire que T2. T1 attend qu'un événement se produise dans T2 (par exemple la libération d'un sémaphore). Dans ce cas, la priorité effective de T1 est réduite à celle de T2 car elle est en attente d'une ressource détenue par T2.
Pour résoudre ce problème, il suffit d'élever temporairement le priorité de T2 en la rendant égale à celle de T1 pendant le laps de temps qu'elle utilise la ressource en ensuite de restaurer sa priorité normale.

❖ Priorité


Task 2 preempts task 3
Task 3 preempts task 2
Task 1 (high, event-based)
Task2 (medium, periodic)
Task 3 (low, event-based)
Idle task
Time
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13
Task 3 preempts idle task
Task 2 preempts idle task
Event processing is delayed until every higher priority tasks are suspended or blocked

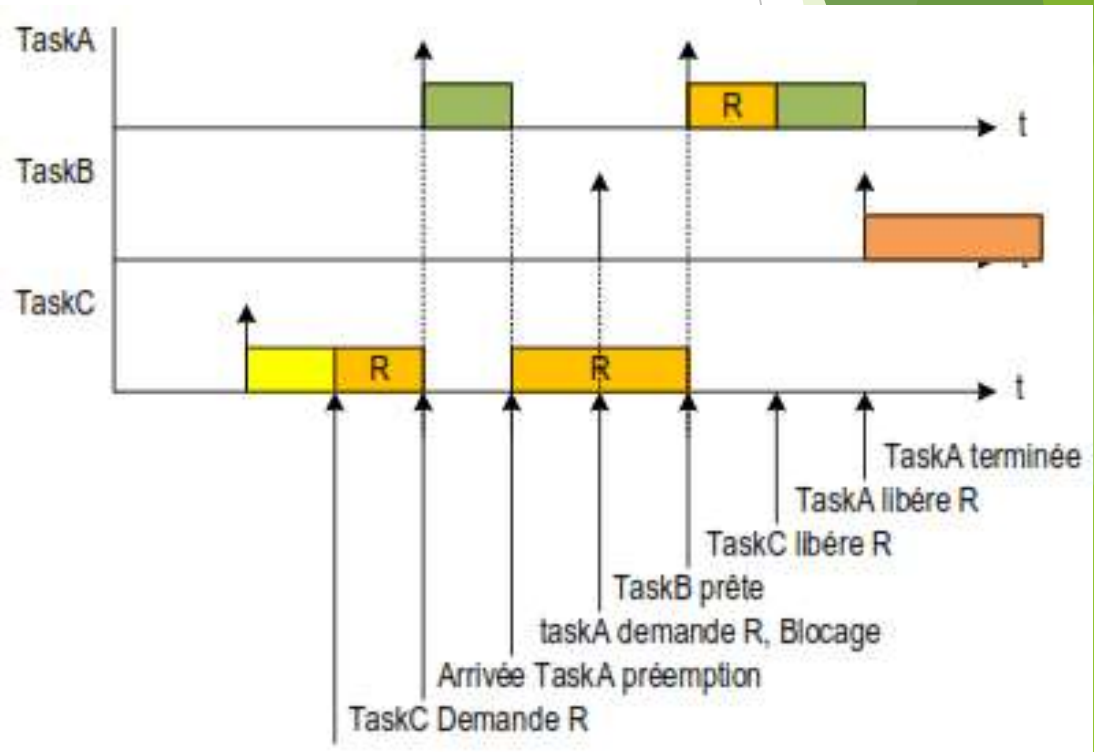
❖ Problème d´inversion de Priorité

Soient trois tâches TaskA (haute priorité), TaskB et taskC (Basse priorité) et R une ressource partagée par les trois tâches. Supposons que TaskC est en possession de la ressource R ; TaskC est préemptée par TaskA plus prioritaire, lorsque TaskA souhaite prendre la ressource R elle sera bloquée à cause de la non disponibilité de R (déjà pris par TaskC), dans ce cas TaskB peut être exécutée si elle est prête. Ce phénomène est appelé ***inversion de priorité***.



❖ Problème d´inversion de Priorité

Pour pallier ce défaut, on attribue à la tâche en possession de ressource (TaskC), *la priorité de la tâche la plus prioritaire des tâches demandant la ressource (héritage de priorité). Dans ce cas, lorsque TaskA est* bloquée, c'est la TaskC qui reprend son exécution et non pas TaskB (TaskC à la même priorité que TaskA). TaskC revient à sa priorité ordinaire quand elle sort de la section critique (libère la ressource).



❖ Gestion des ressources et communication entre taches

❖ Problématique

- Le transfert correct de données entre taches dans un programme temps-réel soulève des problèmes plus importants que dans le cas d'un programme unique communiquant avec des routines d'interruption car :

  - chacune des taches peut voir son exécution suspendue après chaque instruction afin de transférer l'usage du processeur à l'autre tâche,
  - les changements de contexte ne peuvent être évités qu'en interagissant avec l'ordonnceur.

- Pour communiquer entre différentes taches, le noyau fourni des services destinés à :

  - assurer la synchronisation des taches communicantes,
  - organiser le transfert de données d'une tache à l'autre.

❖ Gestion des ressources et communication entre taches

❖ Semaphore

Considérez une situation où il y a deux personnes qui veulent partager un vélo. À la fois, une seule personne peut utiliser le vélo. Celui qui a la clé du vélo aura la chance de l'utiliser. Et lorsque cette personne donne la clé à la 2ème personne, alors seule la 2ème personne peut utiliser le vélo.

Le sémaphore est comme cette clé et le vélo est la ressource partagée. Chaque fois qu'une tâche veut accéder à la ressource partagée, elle doit d'abord acquérir le sémaphore. La tâche doit libérer le sémaphore après avoir terminé avec la ressource partagée. Jusqu'à ce moment, toutes les autres tâches doivent attendre si elles ont besoin d'accéder à une ressource partagée car le sémaphore n'est pas disponible. Même si la tâche essayant d'acquérir le sémaphore est d'une priorité plus élevée que la tâche acquérant le sémaphore, elle sera en état d'attente jusqu'à ce que le sémaphore soit libéré par la tâche de priorité inférieure.

❖ Gestion des ressources et communication entre taches

❖ Utilisation du sémaphore

- ❑ Gestion des ressources partagées
- ❑ Synchronisation des tâches

Outre la gestion des ressources partagées, la synchronisation des tâches peut également être effectuée à l'aide d'un sémaphore. Dans ce cas, le sémaphore sera comme un drapeau et non une clé.

- ➢ Rendez-vous unilatéral

Il s'agit d'une synchronisation à sens unique qui utilise un sémaphore comme indicateur pour signaler une autre tâche.

- ➢ Rendez-vous bilatéral

Il s'agit d'une synchronisation bidirectionnelle effectuée à l'aide de deux sémaphores. Un rendez-vous bilatéral est similaire à un rendez-vous unilatéral, sauf que les deux tâches doivent se synchroniser l'une avec l'autre avant de continuer.

❖ Gestion des ressources et communication entre taches

❖ Type du sémaphore

Mutex : un sémaphore binaire pour l'exclusion mutuelle entre les tâches, pour protéger une section critique. En interne, il fonctionne à peu près de la même manière qu'un sémaphore binaire, mais il est utilisé d'une manière différente. Elle est « prise » avant la section critique et « donnée » juste après, c'est-à-dire dans la même tâche. Un mutex stocke généralement la tâche "propriétaire" actuelle et peut augmenter sa priorité de planification pour éviter un problème appelé "inversion de priorité".

❖ Gestion des ressources et communication entre taches

❖ Type du sémaphore

Mutex : un sémaphore binaire pour l'exclusion mutuelle entre les tâches, pour protéger une section critique. En interne, il fonctionne à peu près de la même manière qu'un sémaphore binaire, mais il est utilisé d'une manière différente. Elle est « prise » avant la section critique et « donnée » juste après, c'est-à-dire dans la même tâche. Un mutex stocke généralement la tâche "propriétaire" actuelle et peut augmenter sa priorité de planification pour éviter un problème appelé "inversion de priorité".

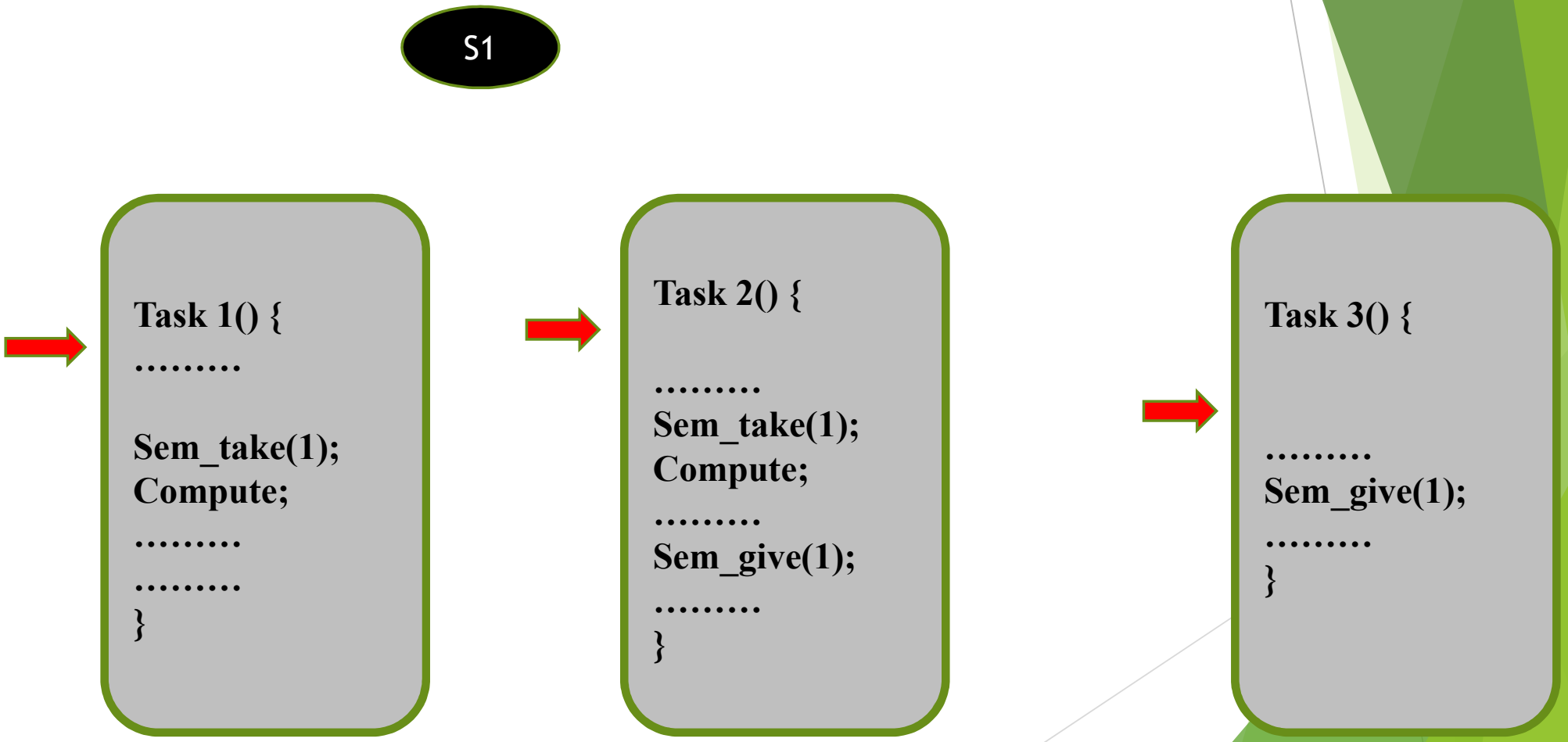
❖ Gestion des ressources et communication entre taches

❖ Type du sémaphore

- Sémaphore de comptage : un sémaphore qui contient un compteur avec une borne supérieure. Cela permet de garder une trace des ressources partagées limitées. Chaque fois qu'une ressource doit être allouée, une tentative de "prendre" le sémaphore est faite et le compteur est incrémenté s'il est inférieur à la borne supérieure spécifiée, sinon la tentative d'allocation bloque la tâche (éventuellement avec un délai d'attente) ou échoue directement, selon le paramètres au service de sémaphore RTOS. Lorsque la ressource doit être libérée, une opération « donner » est effectuée qui décrémente le compteur.

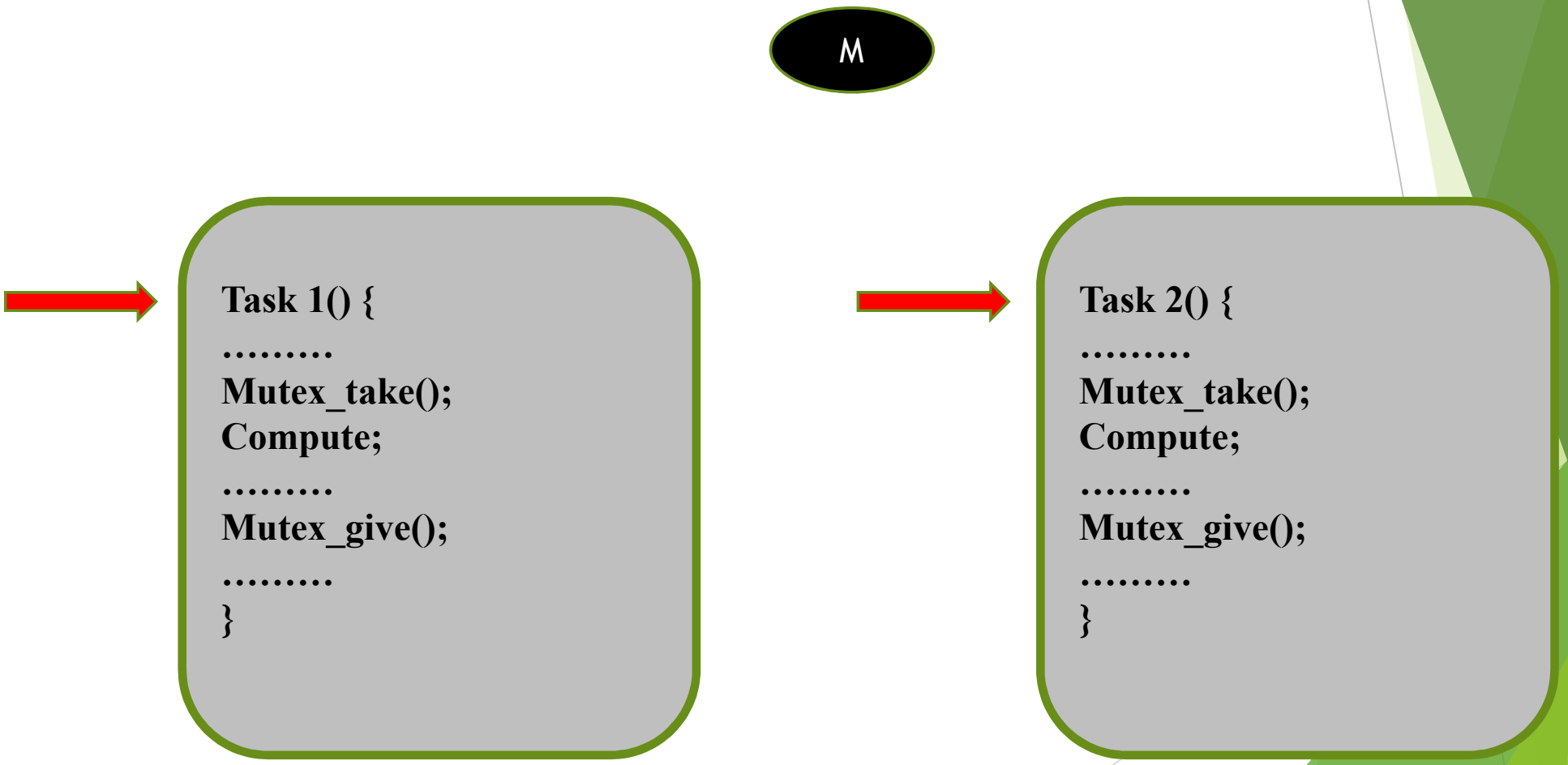
❖ Gestion des ressources et communication entre taches

❖ Sémaphore



❖ Gestion des ressources et communication entre taches

❖ Mutex



❖ FreeRTOS

- **FreeRTOS** est un système d'exploitation temps réel (RTOS) faible empreinte, portable, préemptif et Open source pour microcontrôleur. Il a été porté sur plus de 40 architectures différentes. Créé en 2003 par Richard Barry et la FreeRTOS Team, il est aujourd'hui parmi les plus utilisés dans le marché des systèmes d'exploitation temps réel.

- Le nombre de tâches exécutées simultanément et leur priorité ne sont limités que par le matériel. L'ordonnancement est un système de file d'attente basé sur les Sémaphores et les Mutex. Il est basé sur le modèle Round-Robin avec gestion des priorités. Conçu pour être très compact, il n'est composé que de quelques fichiers en langage C, et n'implémente aucun pilote matériel.
- Les domaines d'applications sont assez larges, car les principaux avantages de FreeRTOS sont l'exécution temps réel, un code source ouvert et une taille très faible. Il est donc utilisé principalement pour les systèmes embarqués qui ont des contraintes d'espace pour le code, mais aussi pour des systèmes de traitement vidéo et des applications réseau qui ont des contraintes de temps réel.