Informatique embarquée

Architecture des logiciels embarqués

Philippe.Plasson@obspm.fr

v1.0

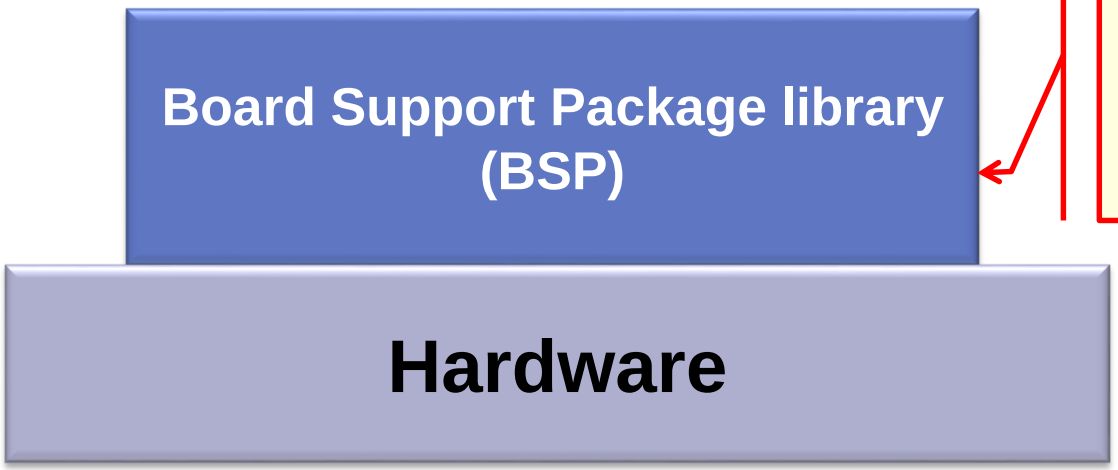
Architecture des logiciels embarqués

1. Les principales couches d'un logiciel embarqué
2. Types d'architecture : boucle de contrôle simple, système piloté par les interruptions, multi-tâches, RTOS
3. Architectures multi-coeurs : AMP, SMP
4. Hyperviseurs et virtualisation

Architecture des logiciels embarqués

1. Les principales couches d'un logiciel embarqué
2. Types d'architecture : boucle de contrôle simple, système piloté par les interruptions, multi-tâches, RTOS
3. Architectures multi-coeurs : AMP, SMP
4. Hyperviseurs et virtualisation

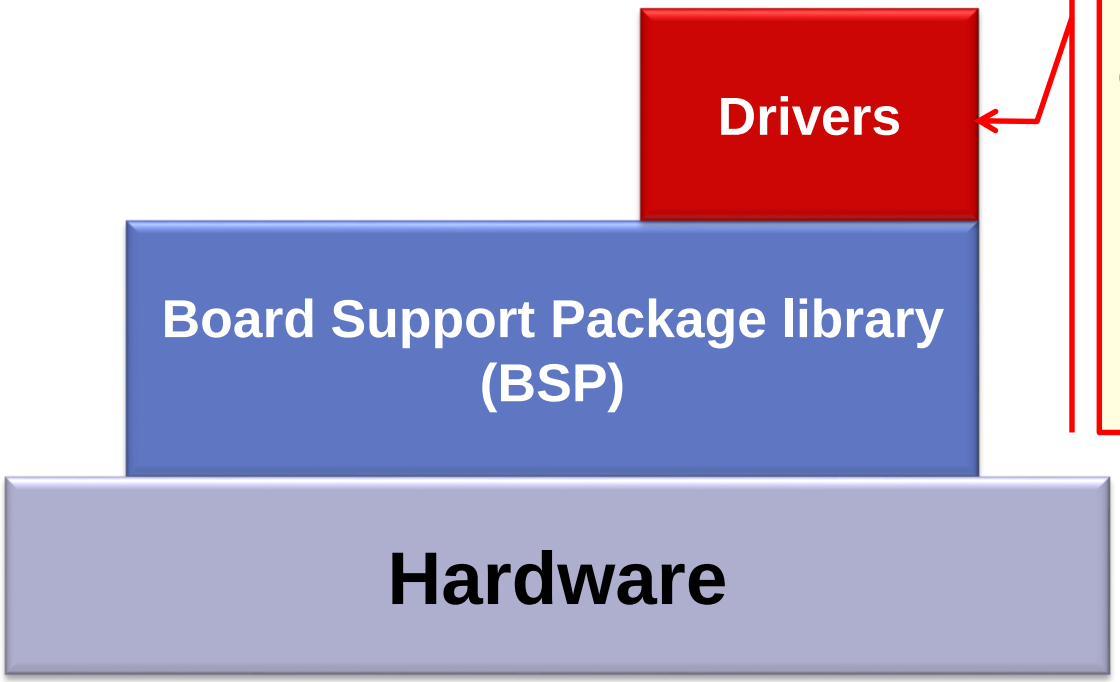
Les principales couches d'un logiciel embarqué



BSP = Board Support Package
Ensemble de modules logiciels bas niveau prenant en charge un processeur :

- Service d'initialisation de la trap table / vector table
- Service d'initialisation des registres du processeur
- Service d'initialisation du contrôleur d'interruptions
- Service d'installation des gestionnaires d'interruptions
- Service de configuration du contrôleur mémoire
- Service d'initialisation de la RAM

Les principales couches d'un logiciel embarqué

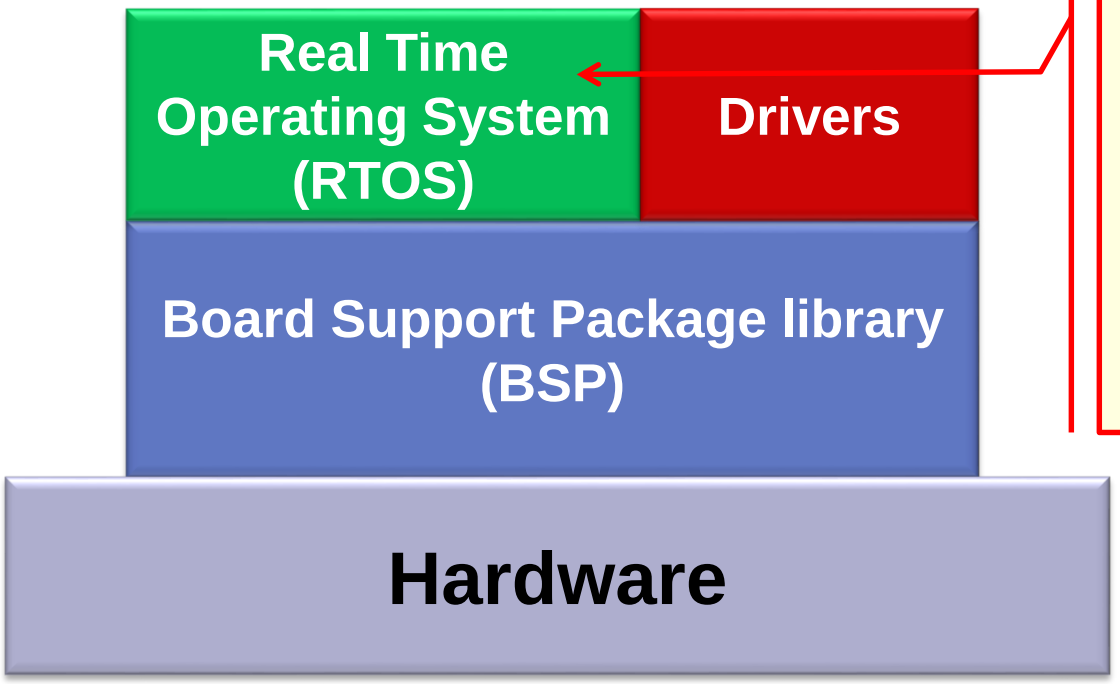


Drivers = pilotes

Modules logiciels bas niveau permettant aux couches applicatives d'interagir avec le matériel (périphériques) :

- Fonctions de configuration
- Fonctions de gestion des entrées / sorties : émission / réception de données
- Gestionnaire d'interruption

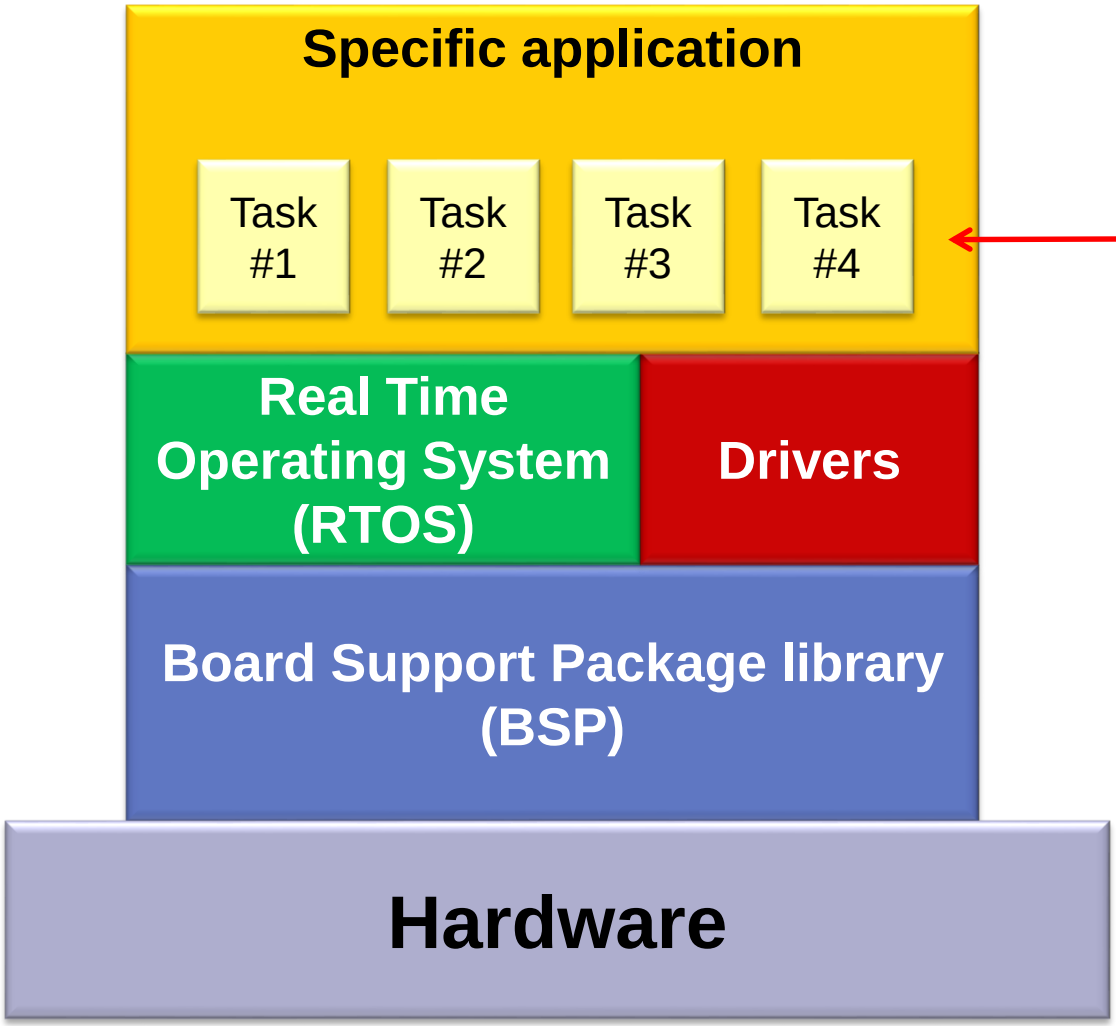
Les principales couches d'un logiciel embarqué



RTOS = Real Time Operating System

- Support du multi-tâches (ordonnanceur, création / destruction des tâches, …)
- Services de communication (queues de message) et de synchronisation inter-tâches (sémaphore, mutex, event flags)
- Services d'allocation de la mémoire
- Timers

Les principales couches d'un logiciel embarqué

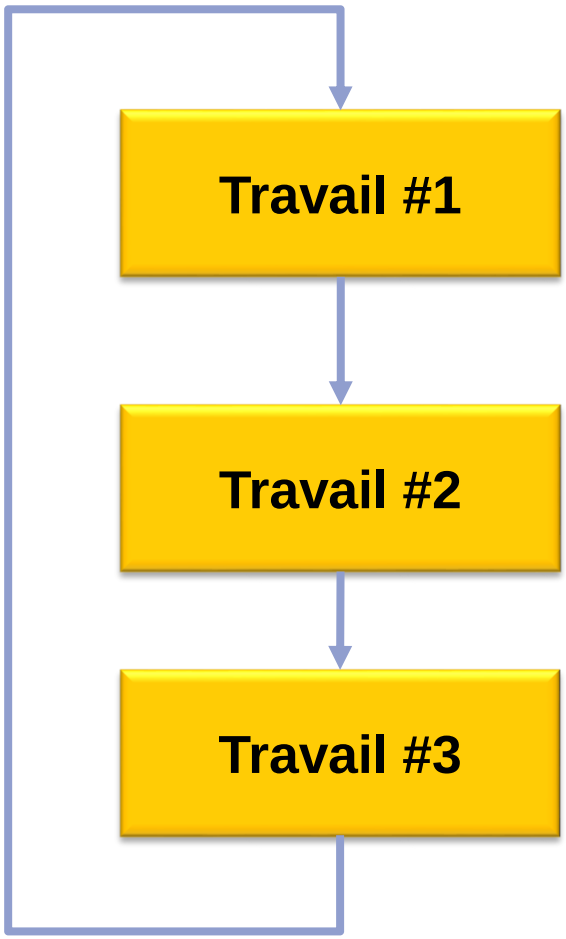


Couche applicative basée sur une architecture multi-tâche

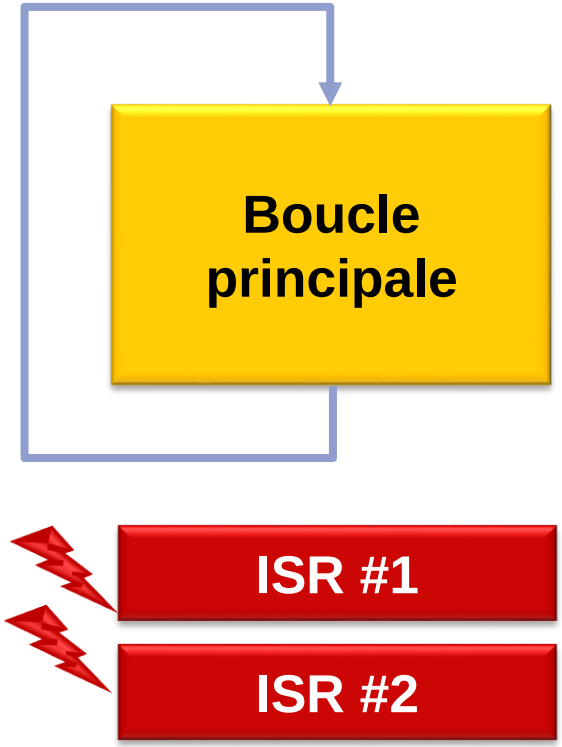
Architecture des logiciels embarqués

1. Les principales couches d'un logiciel embarqué
2. Types d'architecture : boucle de contrôle simple, système piloté par les interruptions, multi-tâches, RTOS
3. Architectures multi-coeurs : AMP, SMP
4. Hyperviseurs et virtualisation

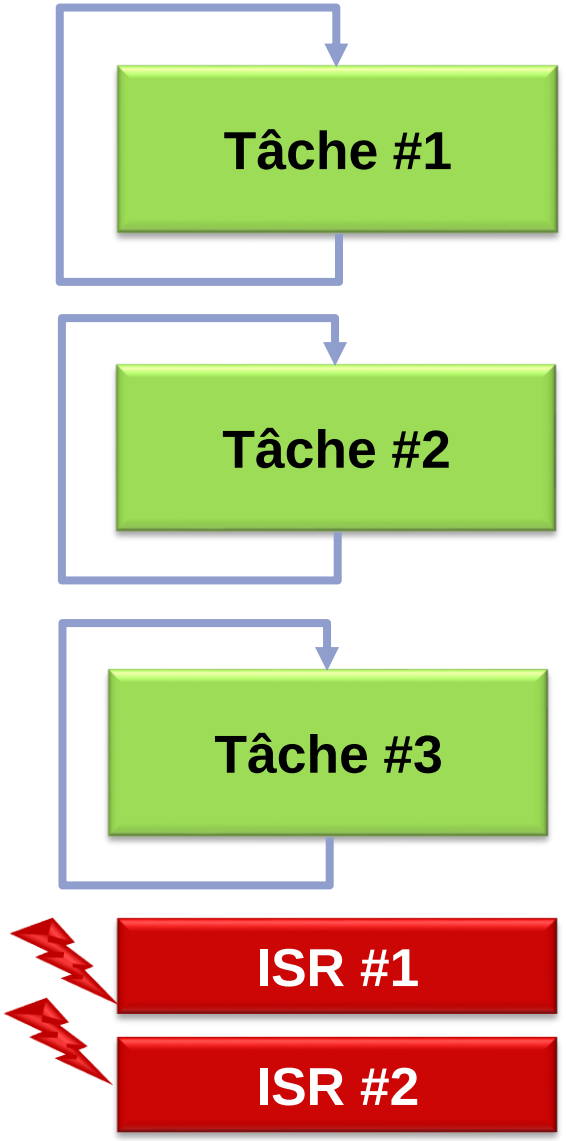
Types d'architecture



Boucle simple

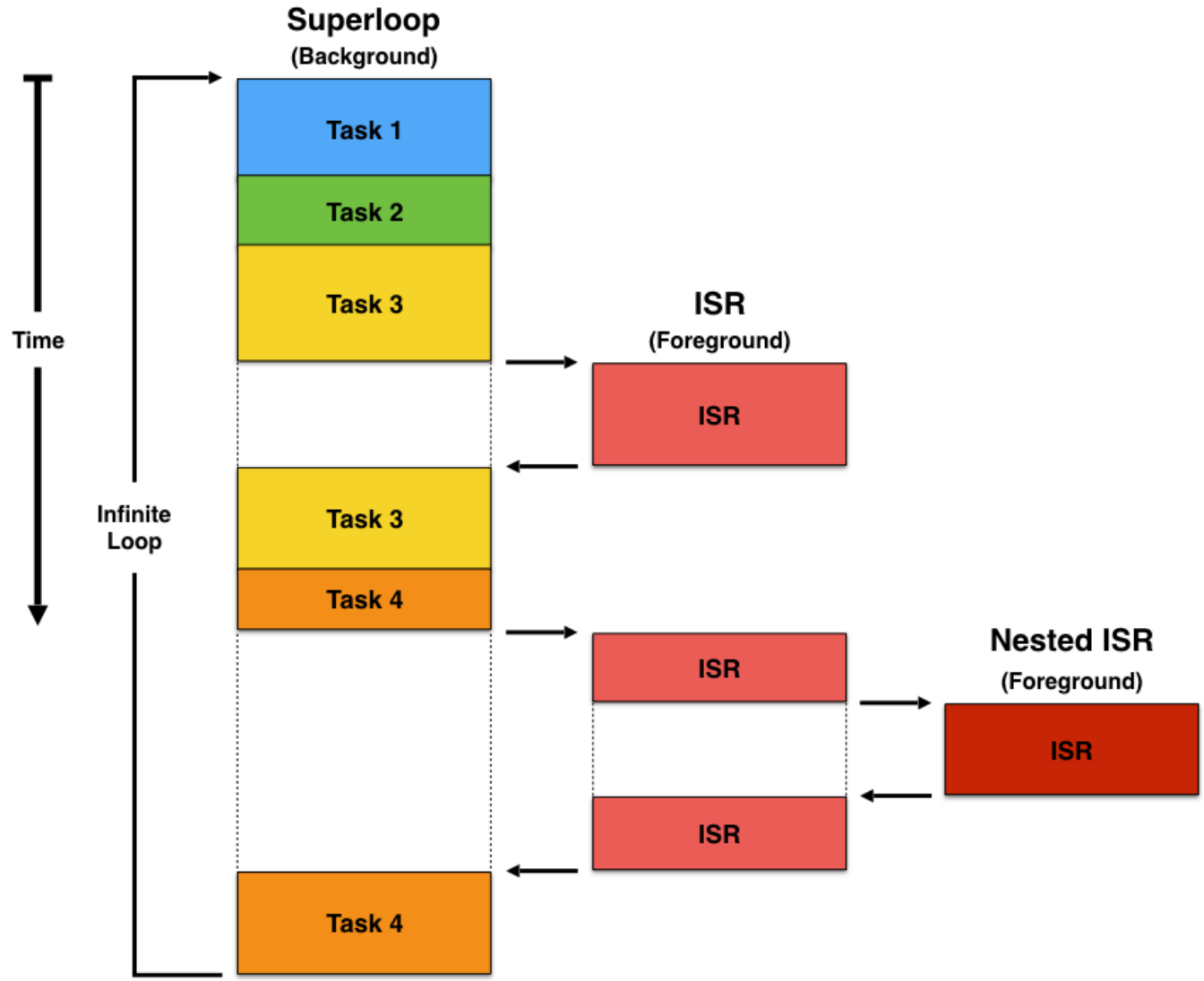


Boucle simple et ISR



RTOS (Multi-tâches)

Type d'architecture
Boucle simple et interruptions



Crédit : http://micro-os-plus.github.io/user-manual/basic-concepts/

Type d'architectures
Les systèmes multi-tâches

- Beaucoup d'approches possibles dans l'implémentation de l'ordonnancement des tâches :
  - Multi-tâches coopératif vs. Multi-tâches préemptif
  - Ordonnancement selon les priorités (« event driven ») vs. Time slicing (« time sharing »)
  - Priorités fixes vs. Priorités dynamiques
  - Etc.
- L'utilisation d'un OS n'est pas forcément nécessaire si l'application est très simple et que les ressources matérielles sont très contraintes.
- Cependant, l'approche consistant à utiliser un RTOS offrant le support au multi-tâches est la plus répandue.

Architectures multi-taches
Définition

- Un OS temps réel (Real-Time OS = RTOS), ou encore noyau temps réel, apporte un certain nombre de services facilitant la conception et la mise au point des applications embarquées temps réel.
- En particulier, un OS temps réel apporte la notion de programmation multi-tâches et de pseudo-parallélisme.
- La programmation multi-tâches permet de concevoir une application sous la forme d'un ensemble de tâches indépendantes, c'est-à-dire ayant leur propre fil d'exécution
  - Chaque tâche a sa propre pile et son propre contexte d'exécution
  - Synonyme de tâche = thread (fil) ➔ on parle aussi de multi-threading
- Les tâches interagissent entre elles via l'échange de messages asynchrones (c'est-à-dire non bloquants) transitant par des queues de messages et via des mécanismes de synchronisation.

Architectures multi-taches
Définition

- La programmation multi-tâches est particulièrement adaptée au développement des applications devant gérer des signaux et des données provenant simultanément de différentes sources (cas généralement des applications embarquées).
- La programmation multi-tâches permet aussi de découpler :
  - la partie d'une application gérant les entrées / sorties (et donc dépendantes des interruptions et du monde extérieur) qui doit être extrêmement réactive et donc avoir la plus haute priorité,
  - de la partie gérant le traitement de données proprement dit qui est moins prioritaire.

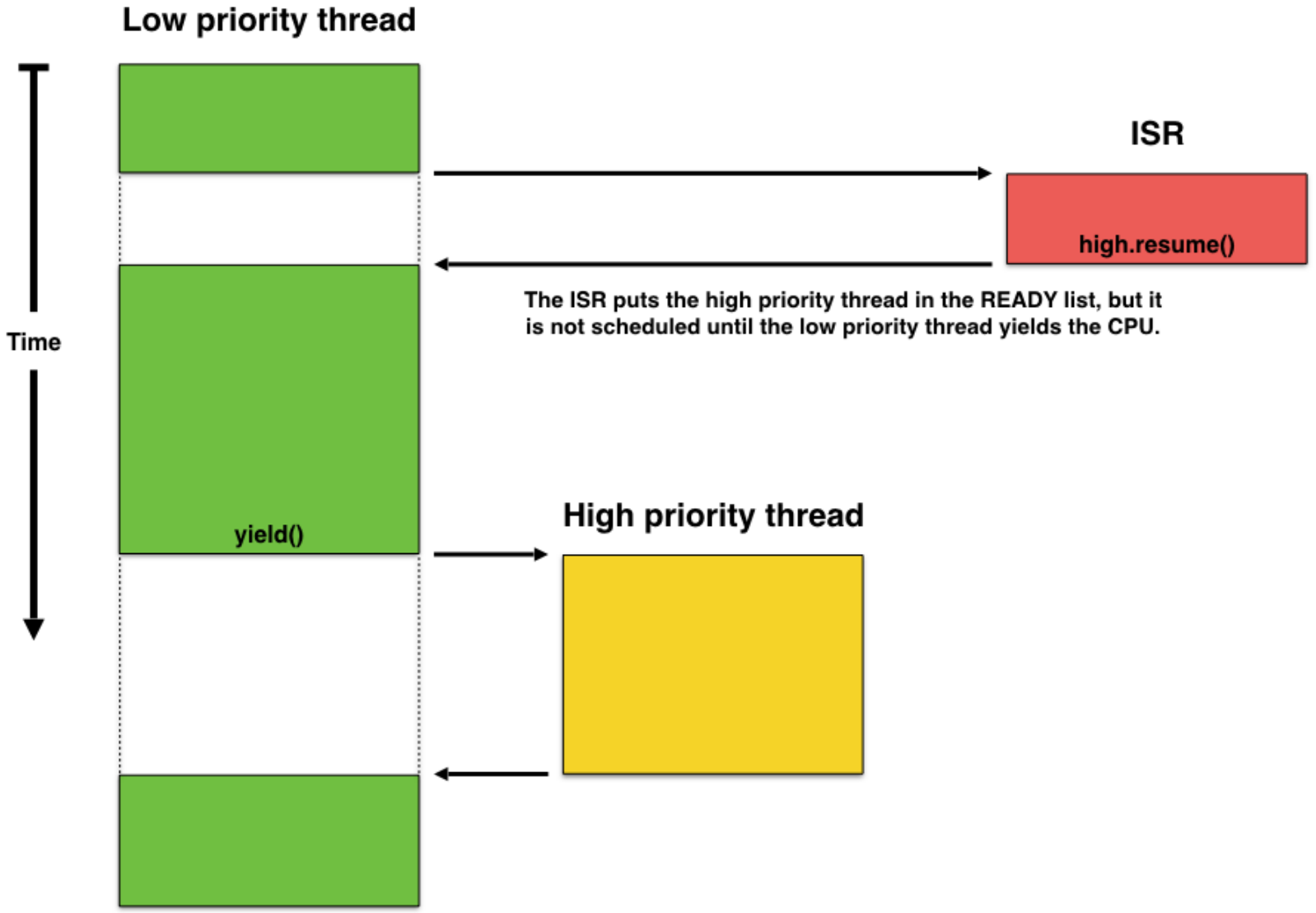
Architectures multi-taches
Définition

- Les traitements associés à une tâche sont déclenchés par la réception de messages asynchrones ou par le changement d'état d'un objet de synchronisation.
- On parle de pseudo-parallélisme, car, à un instant donné, seule une tâche est active.
- L'ordonnanceur est le chef d'orchestre d'un noyau temps réel. Si plusieurs tâches sont prêtes à être activées à un instant donné, c'est l'ordonnanceur qui va décider laquelle de ces tâches va prendre la main et devenir active.
- Il existe plusieurs politiques d'ordonnancement :
  - La plus courante étant celle basée sur les priorités : on assigne à chaque tâche une priorité. Si plusieurs tâches sont prêtes à être activées à un instant donné, c'est la tâche la plus prioritaire qui sera activée.
  - Autre approche = time-slicing : on donne la main à tour de rôle à chaque tâche selon un cadencement périodique

Multi-tâches coopératif

- Appelé aussi multi-tâches non préemptif
- L'OS n'initie jamais de lui-même un changement de contexte d'une tâche vers une autre.
  - Chaque tâche s'exécute jusqu'à ce qu'elle ait terminé son travail ou rende la main explicitement à l'ordonnanceur de tâches.
  - Ce sont les tâches elles-mêmes qui décident de passer la main périodiquement via une fonction de l'OS (ex. : taskYIELD() dans FreeRTOS) ou qui se suspendent en attente d'un message ou d'un événement en provenance d'une autre tâche (appel bloquant).
  - Toutes les tâches doivent coopérer entre elles pour assurer l'ordonnancement global de l'application.
- Ce type de multi-tâches peut être difficile à mettre en œuvre et peut mener à des applications peu réactives et dont les ressources CPU sont mal exploitées.

Multi-tâches coopératif

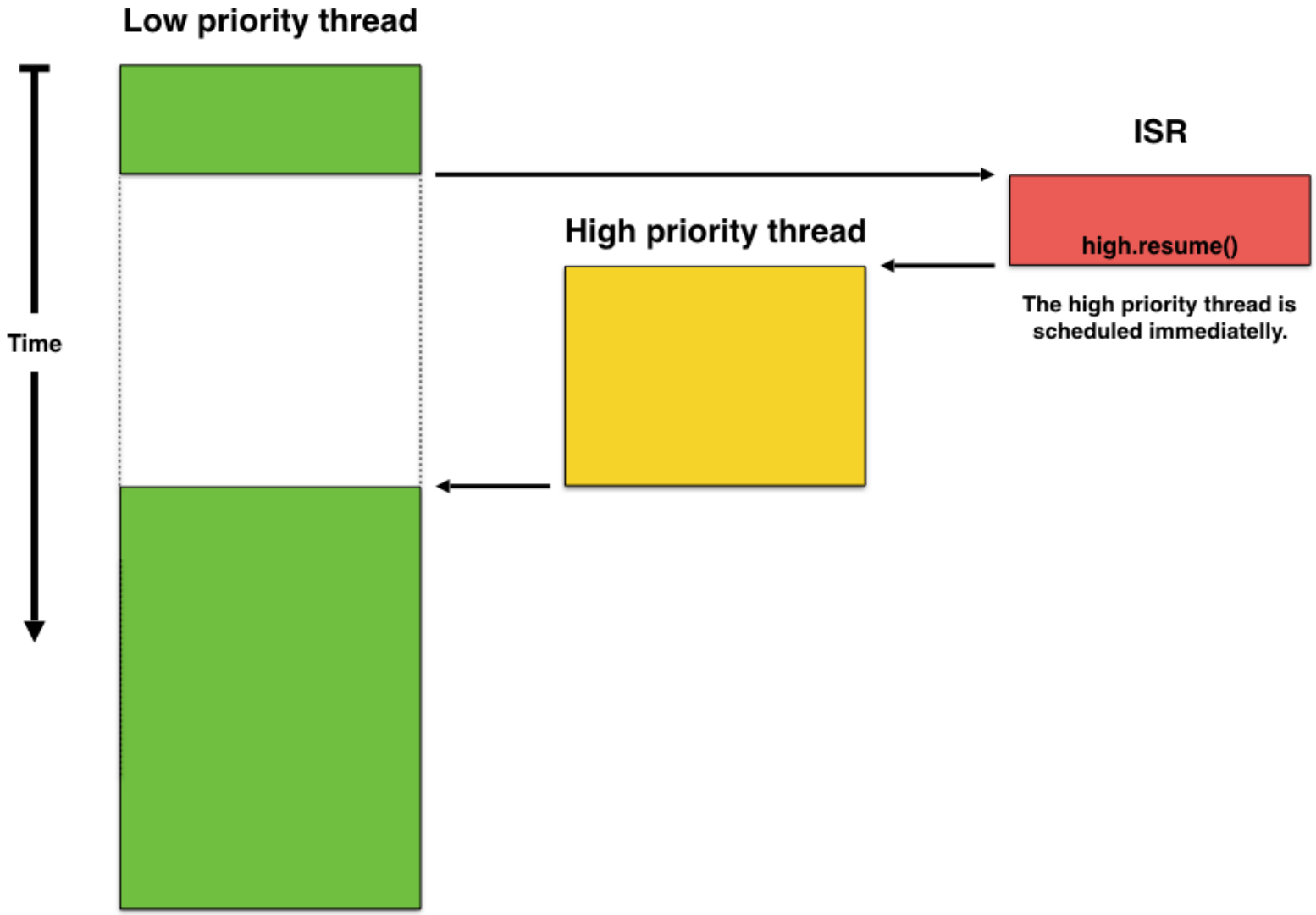


Crédit : http://micro-os-plus.github.io/user-manual/basic-concepts/

Multi-tâche préemptif

- L'ordonnanceur d'un OS préemptif peut interrompre à tout moment une tâche en cours d'exécution pour permettre à une autre tâche de s'exécuter.
  - Une préemption est l'interruption temporaire de l'exécution d'une tâche due à un événement extérieur à la tâche.
  - Passer, lors d'une préemption, de l'exécution d'une tâche à une autre s'appelle une commutation de contexte.

Multi-tâches préemptif
Ordonnancement par les priorités

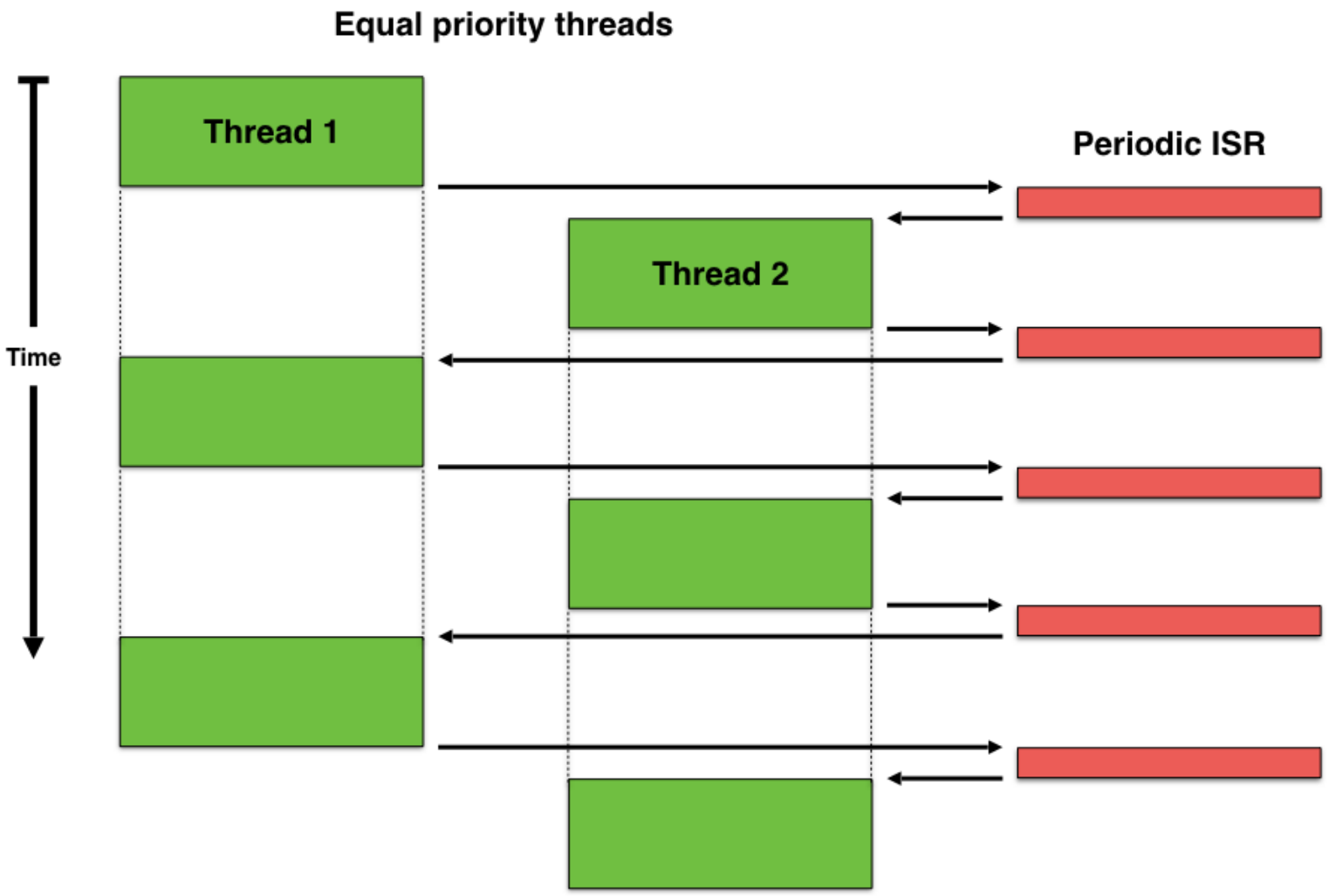


Crédit : http://micro-os-plus.github.io/user-manual/basic-concepts/

Multi-tâches préemptif

- Ce type de multi-tâches, quand il est associé à un ordonnancement sur priorité, permet de construire des systèmes beaucoup plus réactifs aux événements extérieurs.
- Mais il pose aussi des problèmes liés aux accès concurrents aux ressources qu'il faut gérer avec des mécanismes de synchronisation et d'exclusion mutuelle.

Multi-tâches préemptif
Time-slicing



Crédit : http://micro-os-plus.github.io/user-manual/basic-concepts/

RTOS et architectures monolithiques

- La plupart des RTOS du marché dédiés aux applications embarquées « contraintes » (ressources matérielles limitées) sont fournies sous forme de bibliothèques C.
- Les fonctions de ces RTOS (gestions des tâches, gestion des messages, gestion des mécanismes de synchronisation, etc.) sont utilisées par les modules applicatifs du logiciel embarqués.
  - Exemple :
    - ThreadX: https://rtos.com/solutions/threadx/real-time-operating-system/
    - FreeRTOS: https://freertos.org/
    - RTEMS: https://www.rtems.org/
    - eCos: http://ecos.sourceware.org/
    - VxWorks: https://www.windriver.com/products/vxworks/

RTOS et architectures monolithiques

- L'application embarquée et le RTOS forment donc un tout : on parle d'architecture monolithique :
  - Le RTOS est linkée à l'application.
  - Le RTOS ne fonctionne pas indépendamment de l'application.
  - Le RTOS est lié à une seule application.
- A comparer aux architectures basées sur l'utilisation d'OS à usage généraliste comme Linux ou Android pour lesquelles les applications sont clairement dissociées du système d'exploitation.

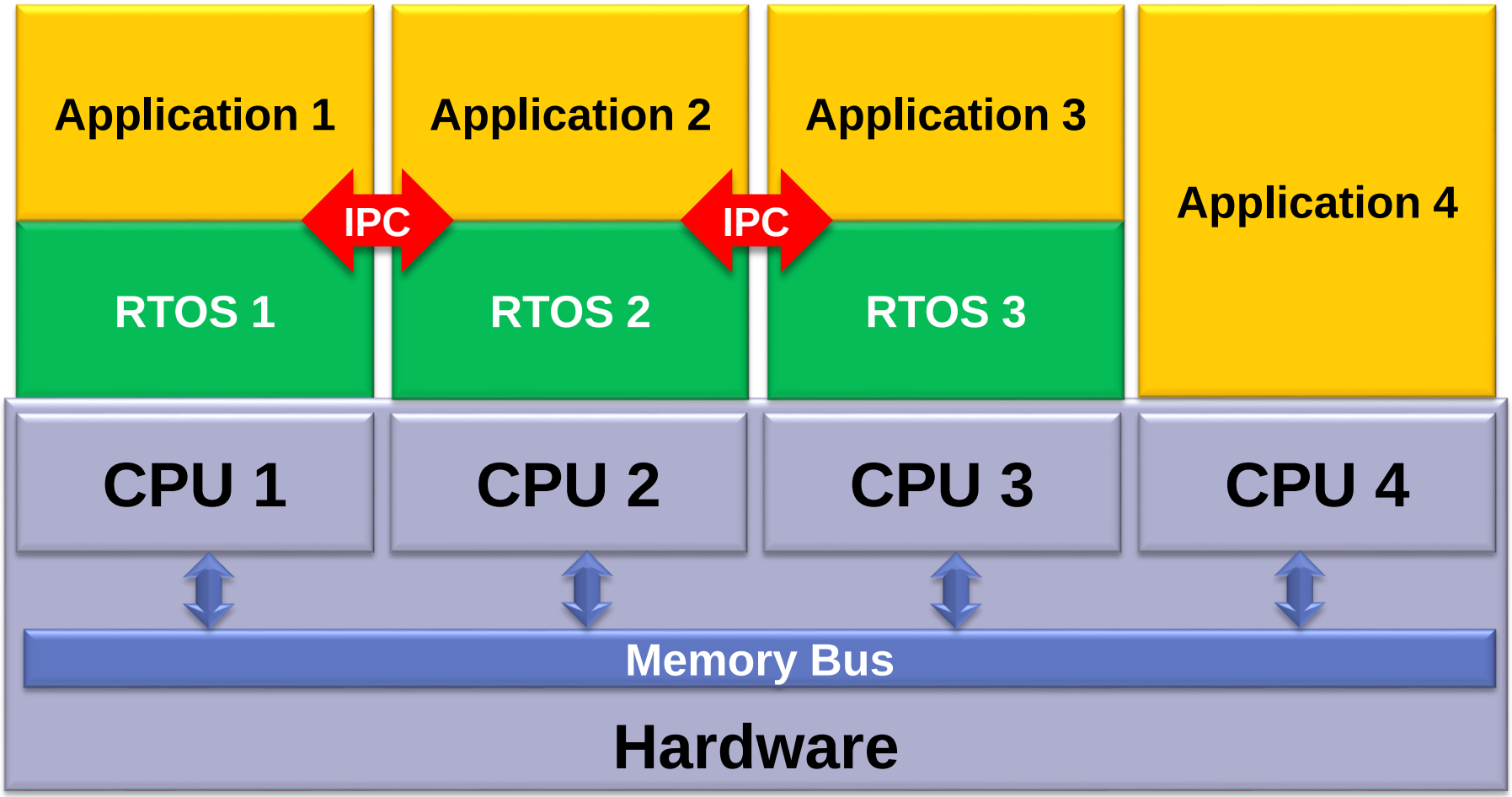
Architecture des logiciels embarqués

1. Les principales couches d'un logiciel embarqué
2. Types d'architecture : boucle de contrôle simple, système piloté par les interruptions, multi-tâches, RTOS
3. Architectures multi-coeurs : AMP, SMP
4. Hyperviseurs et virtualisation

Architectures multi-core
Assymetric Multi-Processing (AMP)

- Approche la plus simple.
- Chaque CPU agit indépendamment des autres.
  - Il y a une image applicative par CPU (donc un OS par CPU si l'application utilise un RTOS).
  - Dans le cas d'un plantage de l'application sur un CPU, les autres applications sur les autres CPU ne sont pas nécessairement affectées.
- Des mécanismes IPC (Inter-Processor Communication) sont utilisés pour échanger des données et des signaux entre les CPU :
  - Mémoires partagées
  - Interruptions dédiées
  - Spin lock basées sur des instructions du processeur réalisant des opérations "compare-and-swap" atomiques et permettant de rendre sûrs les accès concurrents à la mémoire réalisés entre CPU.

Architectures multi-core
Assymetric Multi-Processing (AMP)



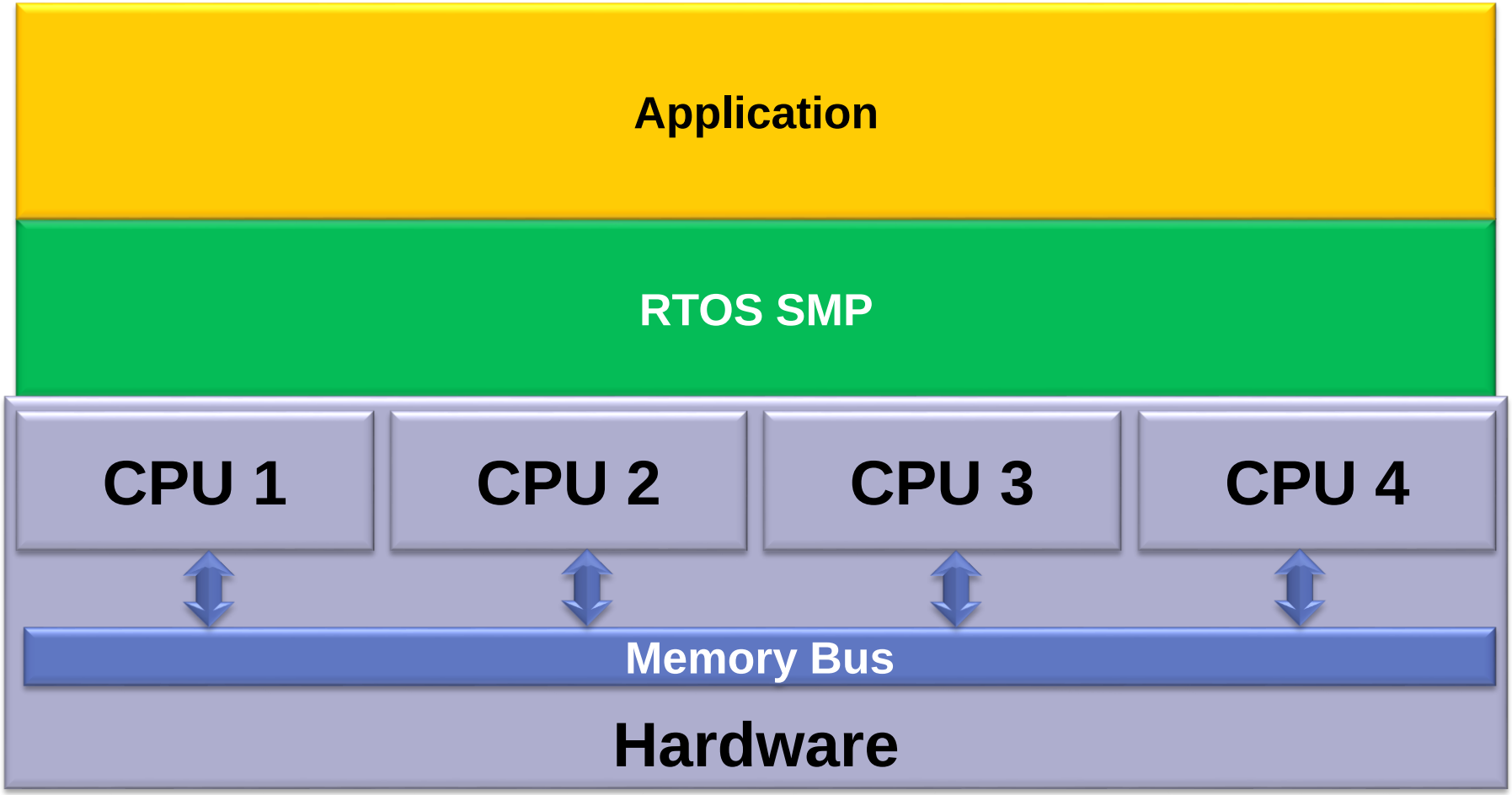
Architectures multi-core
Assymetric Multi-Processing (AMP)

- L'approche AMP est aussi bien adaptée aux SoC mettant en œuvre des processeurs hétérogènes (ex. : un SoC contenant un cœur de processeur LEON et des DSP).
- Le standard OpenAMP est en train d'émerger et vise à définir un framework pour les systèmes AMP :
  - https://www.multicore-association.org/workgroup/oamp.php

Architectures multi-core
Symetric Multi-Processing (SMP)

- Les différents CPU sont gérés par un seul OS.
- Il y a une seule image mémoire (une seule application).
- Les tâches peuvent être explicitement liées à un CPU spécifique (concept d'affinité).
- L'OS peut dispatcher les tâches vers n'importe quel CPU pendant l'exécution.
  - Les taches peuvent flotter d'un CPU à l'autre en fonction des priorités et de la disponibilité des CPU.
  - Concept dit de « load balancing »
- L'approche SMP est adaptée aux processeurs « homogènes » (intégrant des CPU de même nature).

Architectures multi-core
Symetric Multi-Processing (SMP)



Architecture des logiciels embarqués

1. Les principales couches d'un logiciel embarqué
2. Types d'architecture : boucle de contrôle simple, système piloté par les interruptions, multi-tâches, RTOS
3. Architectures multi-coeurs : AMP, SMP
4. Hyperviseurs et virtualisation

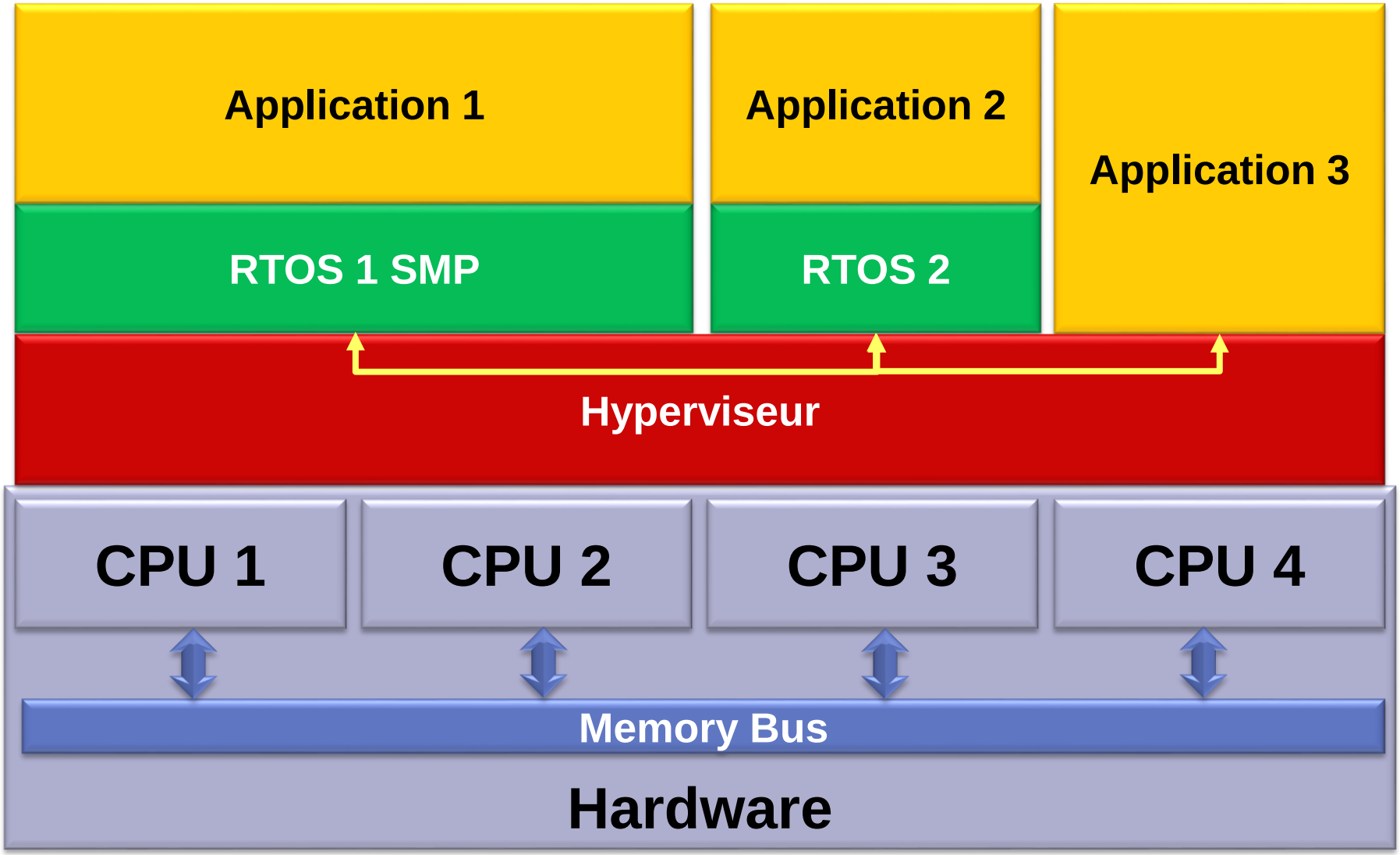
Hyperviseurs / Virtualisation

- Les hyperviseurs sont de plus en plus utilisés dans le domaine de l'embarqué.
- Un hyperviseur fournit une infrastructure permettant d'exécuter plusieurs systèmes d'exploitation ou plusieurs applications temps-réel dans un environnement partitionné robuste.
- L'hyperviseur prend en charge la virtualisation :
  - des ressources CPU
  - de la mémoire
  - des interruptions
  - des timers
  - de certains périphériques

Hyperviseurs / Virtualisation

- Un hyperviseur permet de définir des partitions dans lesquelles vont pouvoir s'exécuter :
  - Des applications compilées pour s'exécuter dans le support d'un OS
  - Des applications temps-réel et leur RTOS
  - Des OS à usage général et leurs applications
- Pour accéder aux ressources matérielles, les applications ou OS s'exécutant dans les partitions doivent faire appels aux services de l'hyperviseur.
- Un hyperviseur offre aussi des mécanismes de communication entre partitions.

Hyperviseurs



Hyperviseurs / Virtualisation

- Exemple :
  - XtratuM : http://www.xtratum.org/
  - PikeOS (RTOS + plate-forme de virtualisation) : https://en.wikipedia.org/wiki/PikeOS