programmation multi-cœur avec l'OS Trampoline

Sébastien Dubé, Jean-Luc Béchennec, Mikaël Briday

Jessica - 27 septembre 2013

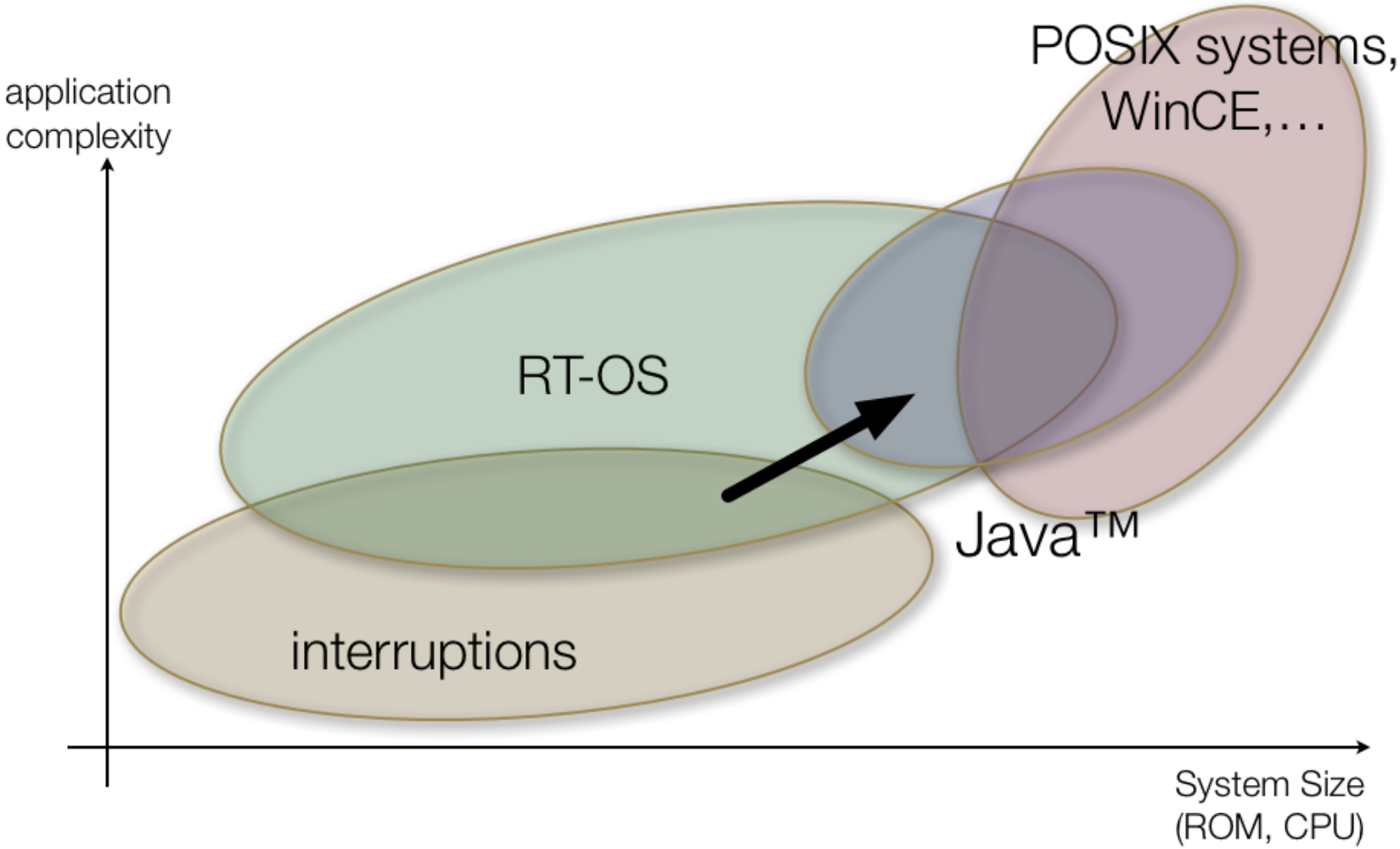
Contexte

- Electronique embarquée dans les véhicules avec des contraintes temps-réel plus ou moins dures
  - Moteur ;
  - Chassis ;
  - Habitacle.
- Contraintes économiques fortes : Petits calculateurs (peu de RAM)
- Systèmes répartis interconnectés par des bus standards : CAN, LIN et maintenant FlexRay
- Fiabilité élevée requise : ABS, ESP, AirBag, ...

Buts d'un OS Temps réel

- Motivations
  - Coût important de développement ou de portage ;
  - Pas d'inter-opérabilité des systèmes construits par divers équipementiers
- Approche choisie : spécification de l'architecture et des briques logicielles.
  - Interfaces indépendantes du matériel ;
  - Comportements bien définis pour faciliter les portages ;
  - Une approche dédiée pour tenir compte des spécificités du domaine (embarqué, temps-réel, ... et le coût)
- Avantages attendus
  - Réutilisation du logiciel de Base ;
  - Portabilité des applications.

Positionnement



OSEK/VDX

- OSEK/VDX : « Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug / Vehicle Distributed Executive » (Systèmes ouverts et leurs interfaces pour l'électronique automobile) ;
- Consortium industriel et académique (de l'industrie automobile et de la recherche). Comité de pilotage : Opel, BMW, Daimler-Chrysler, PSA, Renault, Volkswagen, Robert Bosch, Siemens, Université de Karlsruhe ;
- Spécification système (architecture, interfaces et comportement) pour les systèmes embarqués électroniques automobiles (`http://www.osek-vdx.org`) ;
- Fondation d'AUTOSAR ;
- Standard ISO 17356.

Spécifications OSEK/VDX

OS Noyau temps-réel "conduit par les événements" ;

COM Couche de communication applicative ;

NM Gestion du réseau ;

OIL Langage de description et de configuration hors-ligne ;

ORTI Interface de déverminage ;

ttOS and FTCOM Architecture et composants "conduits par le temps" pour les systèmes les plus critiques.

Spécifications OSEK/VDX

OS Noyau temps-réel "conduit par les événements" ;

COM Couche de communication applicative ;

NM Gestion du réseau ;

OIL Langage de description et de configuration hors-ligne ;

ORTI Interface de déverminage ;

ttOS and FTCOM Architecture et composants "conduits par le temps" pour les systèmes les plus critiques.

Spécifications OSEK/VDX

OS Noyau temps-réel "conduit par les événements" ;

COM Couche de communication applicative ;

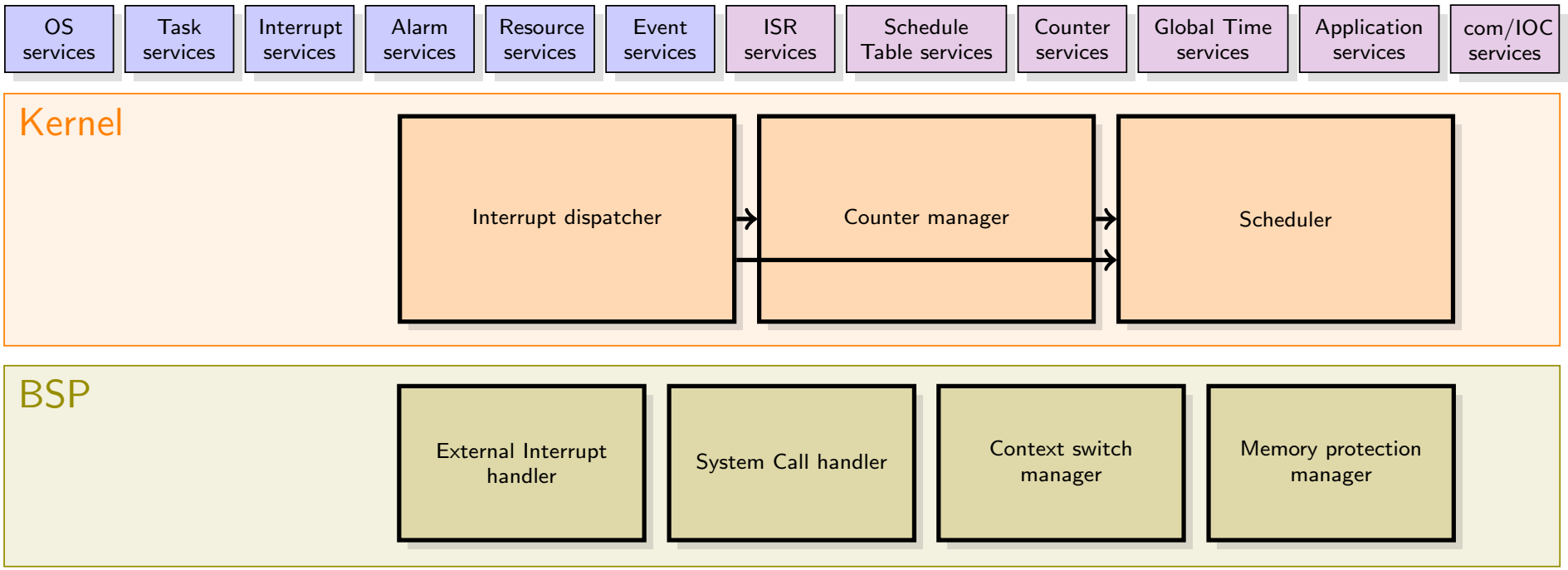
NM Gestion du réseau ;

OIL Langage de description et de configuration hors-ligne ;

ORTI Interface de déverminage ;

ttOS and FTCOM Architecture et composants "conduits par le temps" pour les systèmes les plus critiques.

Architecture de Trampoline



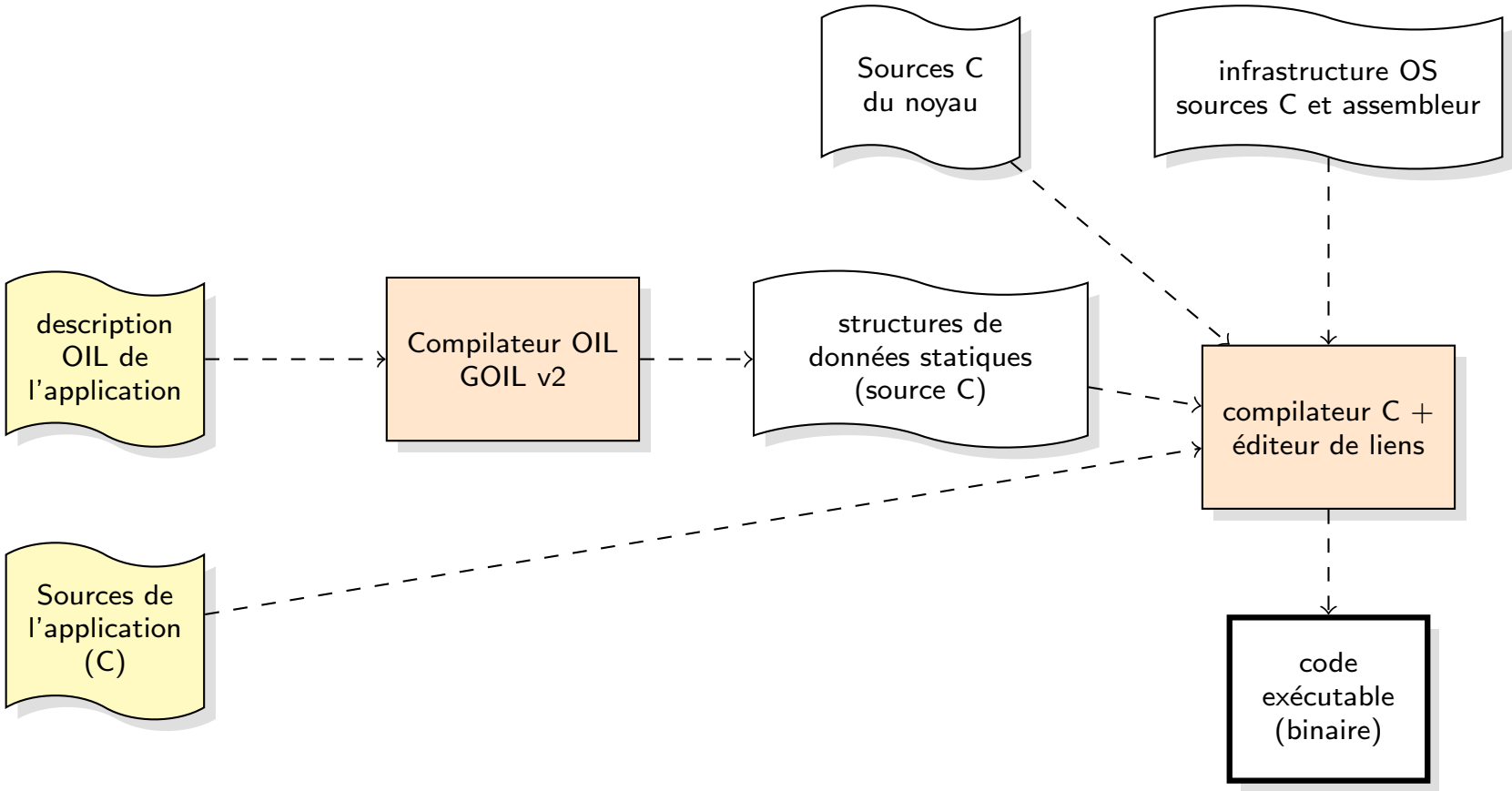
Conçu pour le domaine automobile

- fondé sur peu (mais suffisamment) de concepts ;
- configuration statique (hors ligne) : L'architecture de l'application est complètement connue. Ceci simplifie beaucoup la conception du noyau ;
- permet de n'embarquer que les fonctions de l'OS effectivement utilisées ;
- permet de stocker les descripteurs statiques et la configuration en ROM ;
- comportement prédictible. Correspond aux besoins des applications temps-réel.

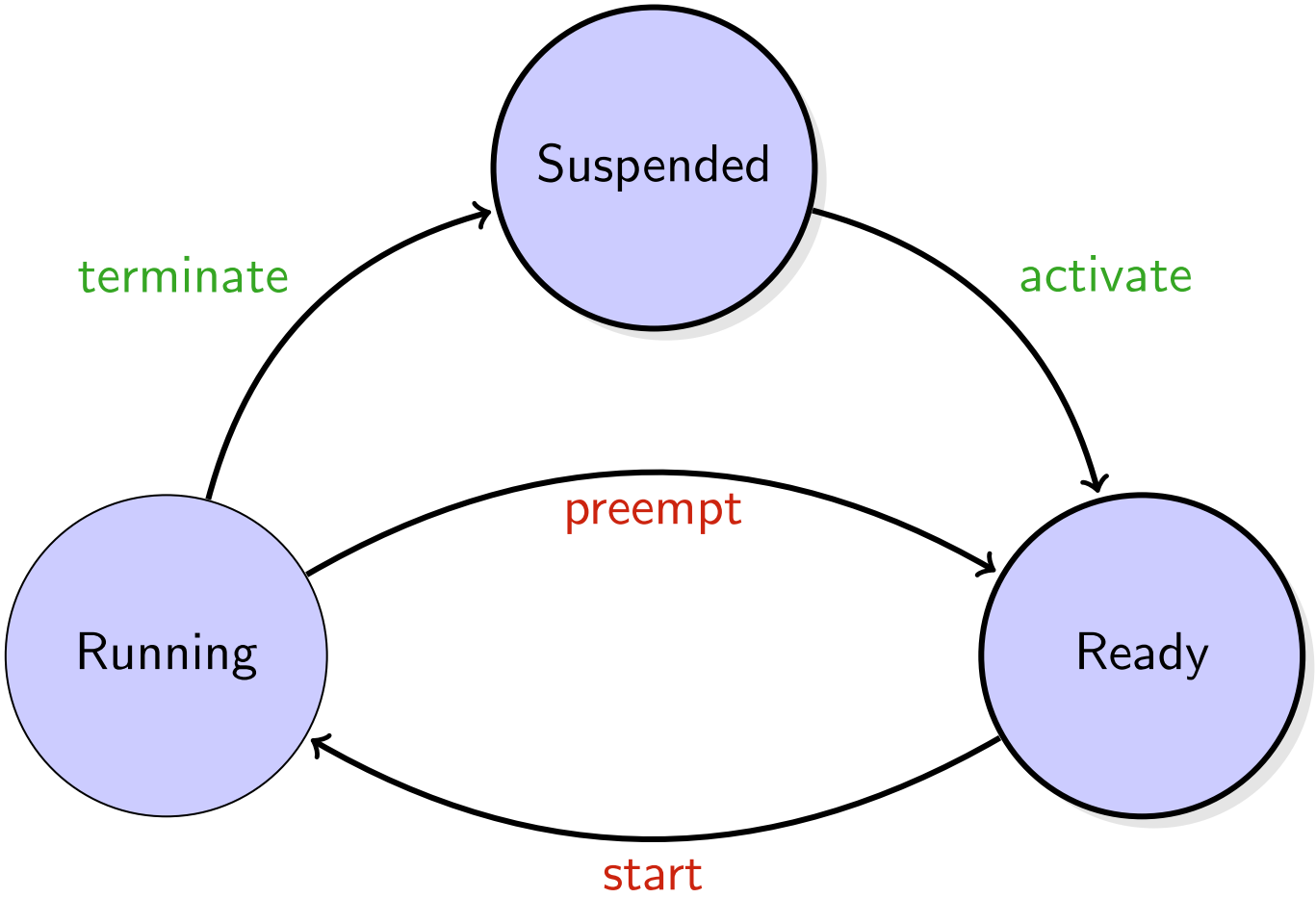
Processus de développement OSEK OS + OIL

- Les objets d'une application OSEK sont tous définis quand l'application est conçue : les objets sont statiques, pas de création/suppression de tâche, resource, ... dynamiquement pendant l'exécution de l'application.
- La syntaxe OIL est simple : un ensemble d'objets (tasks, resources, ...) avec une valeur pour chaque attribut ;
- Certains attributs ont des sous-attributs.
- À partir de cette description (fichier texte), un compilateur génère les structures de données automatiquement :
  - rapide ;
  - moins sujet aux erreurs ;
  - indépendant du concepteur de l'OS ;
  - facile à mettre à jour.

Processus de développement OSEK OS + OIL



Modèle de tâche basique



Utilisation d'une tâche basique

Déclaration statique de la tâche (fichier OIL)

```
TASK myTask {
  PRIORITY = 5;
  AUTOSTART = FALSE;
  SCHEDULE = FULL;
  STACK = 256;
};
```

Implémentation de la tâche en langage C

```
int main()
{
  initHw();
  StartOS(OSDEFAULTAPPMODE);
  return 0;
}

TASK(myTask)
{
  computeSth();
  TerminateTask();
}
```

Les alarmes

- Objectif : faire une *action* après un certain nombre de *ticks* d'un périphérique matériel
- *l'action* peut être :
  - la signalisation d'un évènement ;
  - l'activation d'une tâche ;
  - l'appel d'une fonction (callback). Retiré dans AUTOSAR.
- le périphérique matériel peut-être
  - un *timer* ;
  - une source d'interruption périodique (régime moteur par exemple).

L'alarme est définie statiquement (dans l'OIL), mais ses caractéristiques peuvent être modifiées dynamiquement (arrêt/marche, périodicité).

Synchronisation dans OSEK/OS - AUTOSAR monocœur

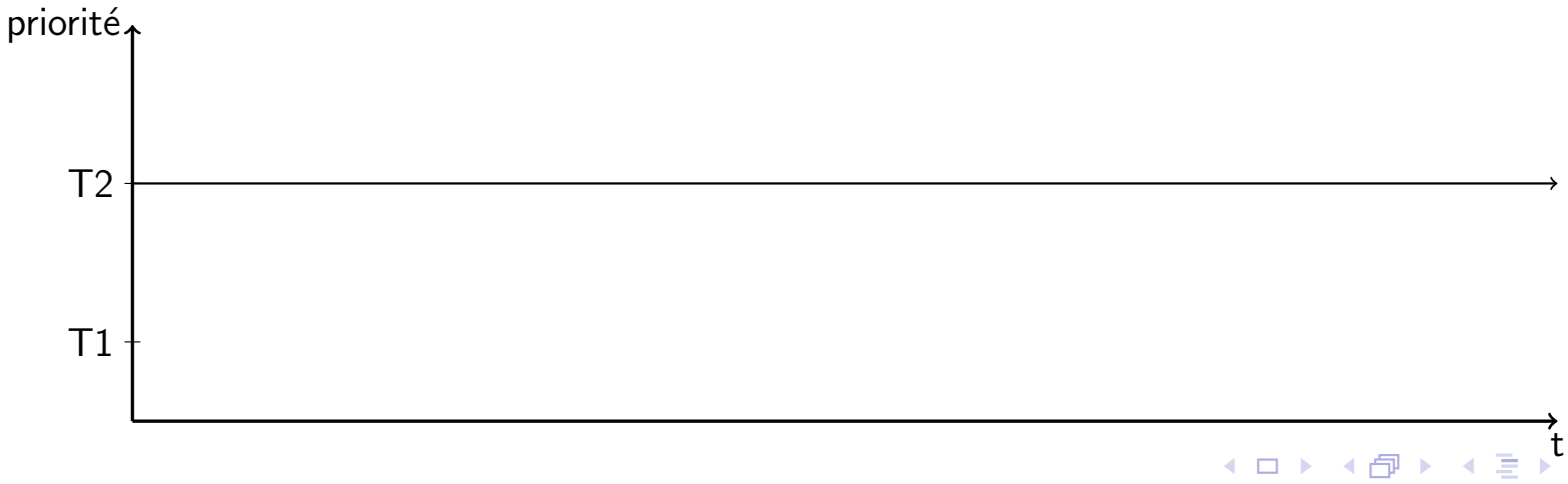
Pas de sémaphore :

- inversion de priorité ;
- appel système bloquant ;
- deadlock (en cas de mauvaise utilisation).

Exemple de deadlock...

```
1  TASK(T1)
2  {
3     mutex_lock(S1);
4     mutex_lock(S2);
5     //section critique
6     //avec S1 et S2
7     mutex_unlock(S2);
8     mutex_unlock(S1);
9  }
```

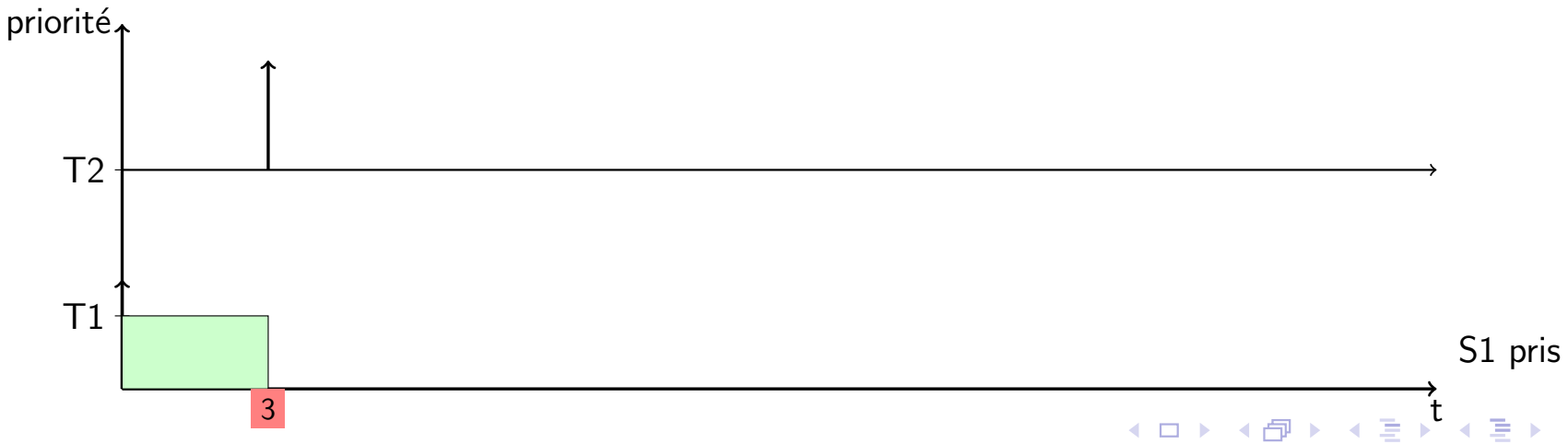
```
10 TASK(T2)
11 {
12    mutex_lock(S2);
13    mutex_lock(S1);
14    //section critique
15    //avec S1 et S2
16    mutex_unlock(S1);
17    mutex_unlock(S2);
18 }
```



Exemple de deadlock...

```
1  TASK(T1)
2  {
3     mutex_lock(S1);
4     mutex_lock(S2);
5     //section critique
6     //avec S1 et S2
7     mutex_unlock(S2);
8     mutex_unlock(S1);
9  }
```

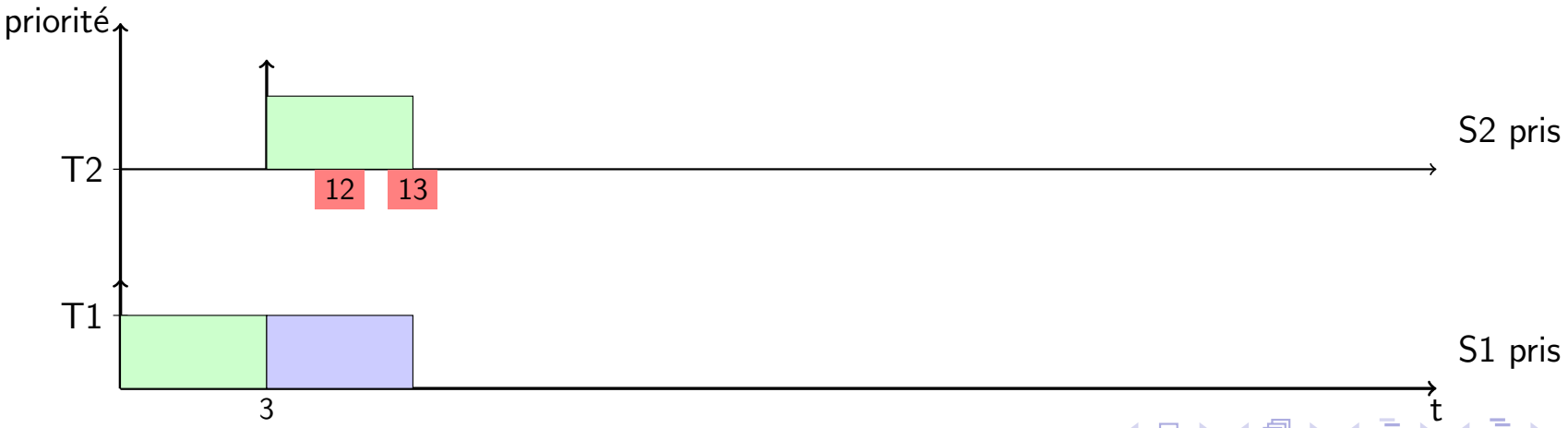
```
10 TASK(T2)
11 {
12    mutex_lock(S2);
13    mutex_lock(S1);
14    //section critique
15    //avec S1 et S2
16    mutex_unlock(S1);
17    mutex_unlock(S2);
18 }
```



Exemple de deadlock...

```
1  TASK(T1)
2  {
3     mutex_lock(S1);
4     mutex_lock(S2);
5     //section critique
6     //avec S1 et S2
7     mutex_unlock(S2);
8     mutex_unlock(S1);
9  }
```

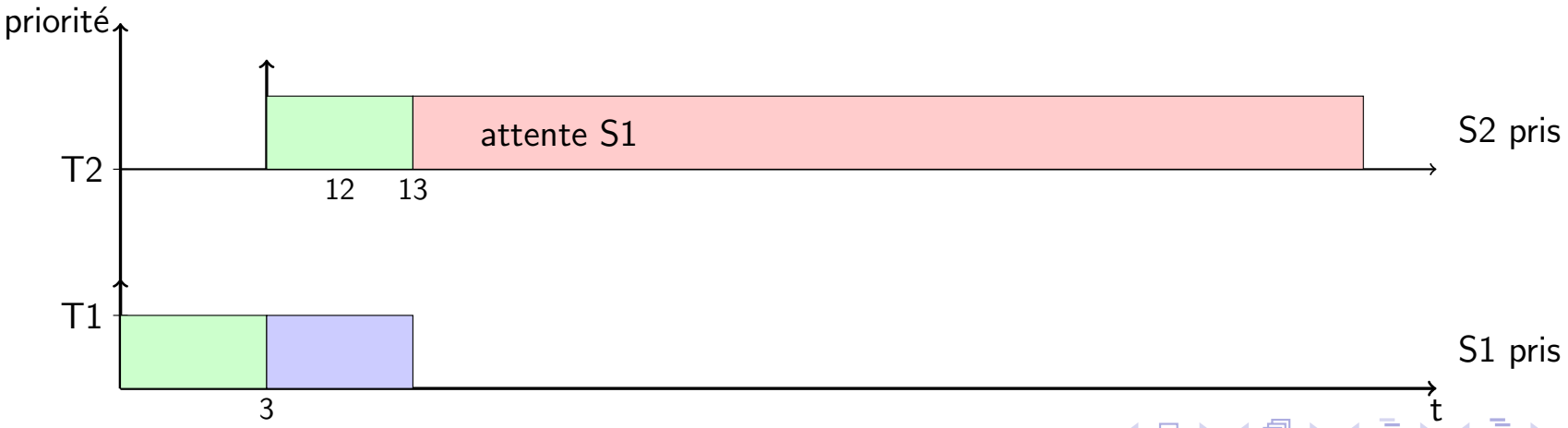
```
10 TASK(T2)
11 {
12    mutex_lock(S2);
13    mutex_lock(S1);
14    //section critique
15    //avec S1 et S2
16    mutex_unlock(S1);
17    mutex_unlock(S2);
18 }
```



Exemple de deadlock...

```
1  TASK(T1)
2  {
3     mutex_lock(S1);
4     mutex_lock(S2);
5     //section critique
6     //avec S1 et S2
7     mutex_unlock(S2);
8     mutex_unlock(S1);
9  }
```

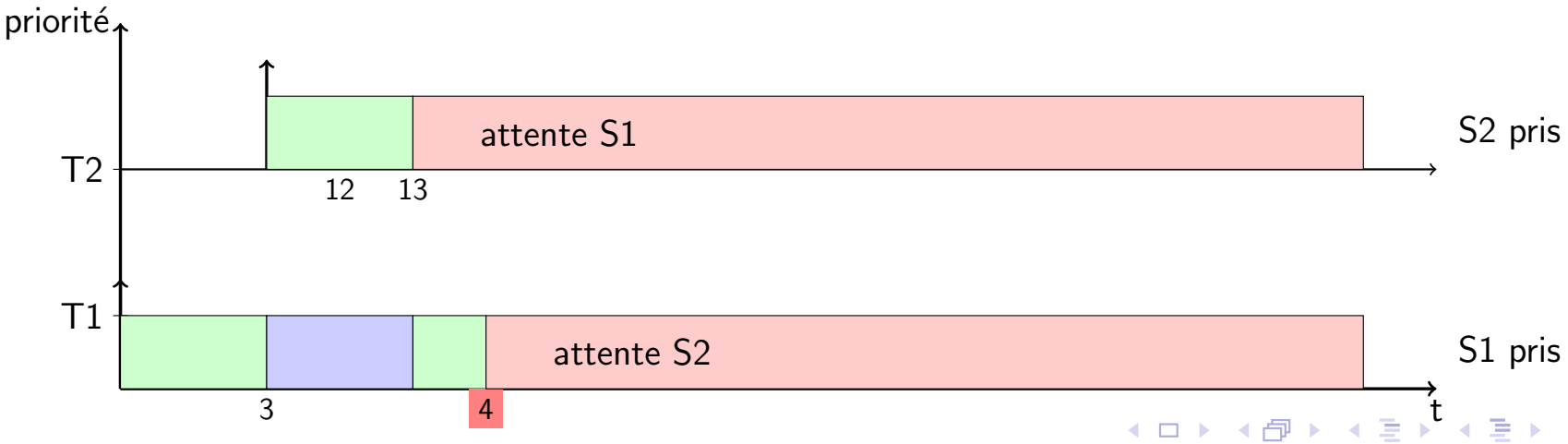
```
10 TASK(T2)
11 {
12    mutex_lock(S2);
13    mutex_lock(S1);
14    //section critique
15    //avec S1 et S2
16    mutex_unlock(S1);
17    mutex_unlock(S2);
18 }
```



Exemple de deadlock...

```
1  TASK(T1)
2  {
3     mutex_lock(S1);
4     mutex_lock(S2);
5     //section critique
6     //avec S1 et S2
7     mutex_unlock(S2);
8     mutex_unlock(S1);
9  }
```

```
10 TASK(T2)
11 {
12    mutex_lock(S2);
13    mutex_lock(S1);
14    //section critique
15    //avec S1 et S2
16    mutex_unlock(S1);
17    mutex_unlock(S2);
18 }
```

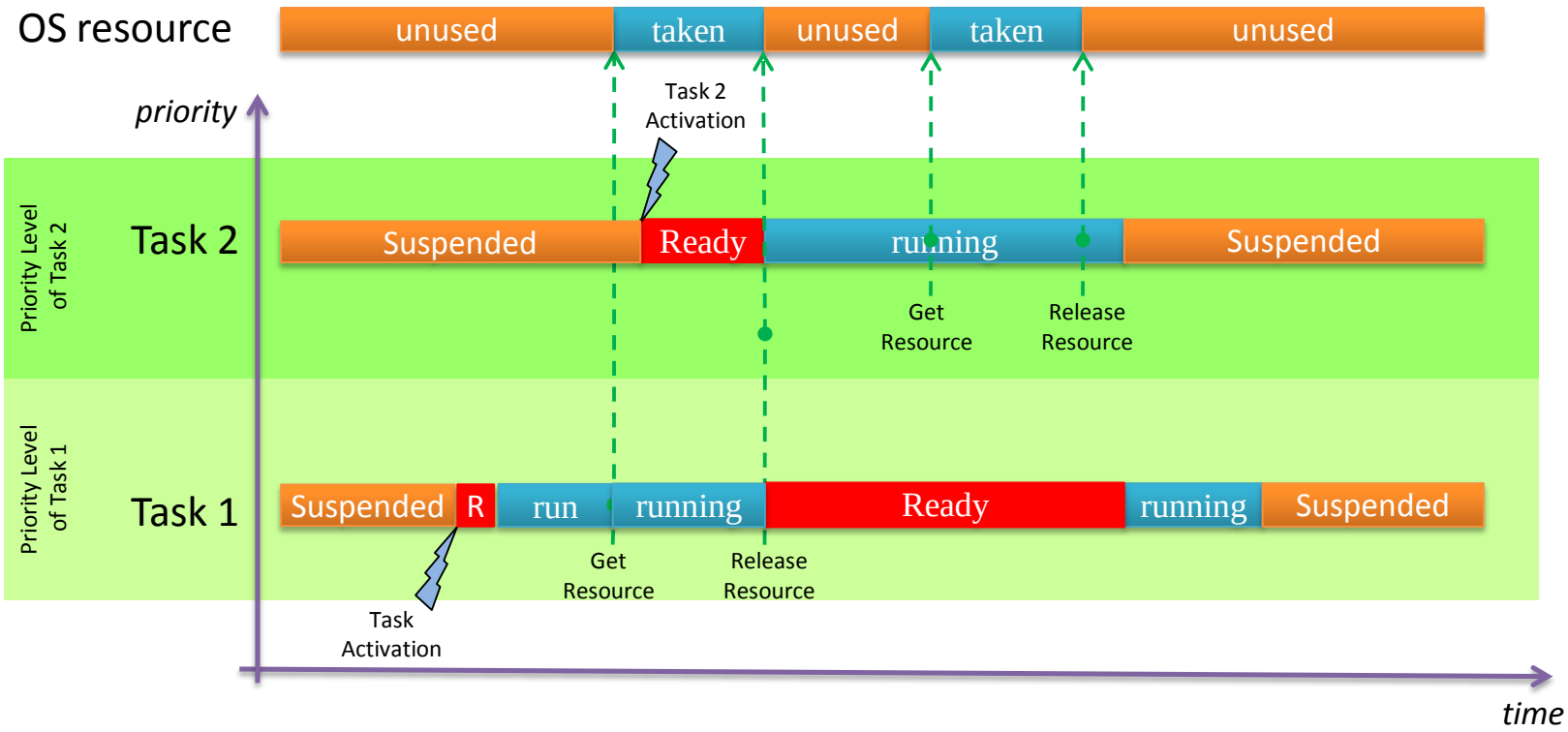


Synchronisation dans OSEK/OS - AUTOSAR monoœur

Utilisation du protocole IPCP : Immediat Priority Ceiling Protocol

- on définit une *ressource* (jeton), à laquelle on associe une priorité
  $\Rightarrow$ la priorité est le *max* (plafond) des priorités des tâches qui peuvent prendre la ressource ;
- une tâche qui prend la ressource hérite de la priorité de la ressource
  $\Rightarrow$ elle devient ponctuellement non preemptible par les autres tâches susceptibles d'être en concurrence sur la ressource ;
- lors du relâchement de la ressource, la tâche retrouve sa priorité initiale.

Synchronisation dans OSEK/OS - AUTOSAR monoœur

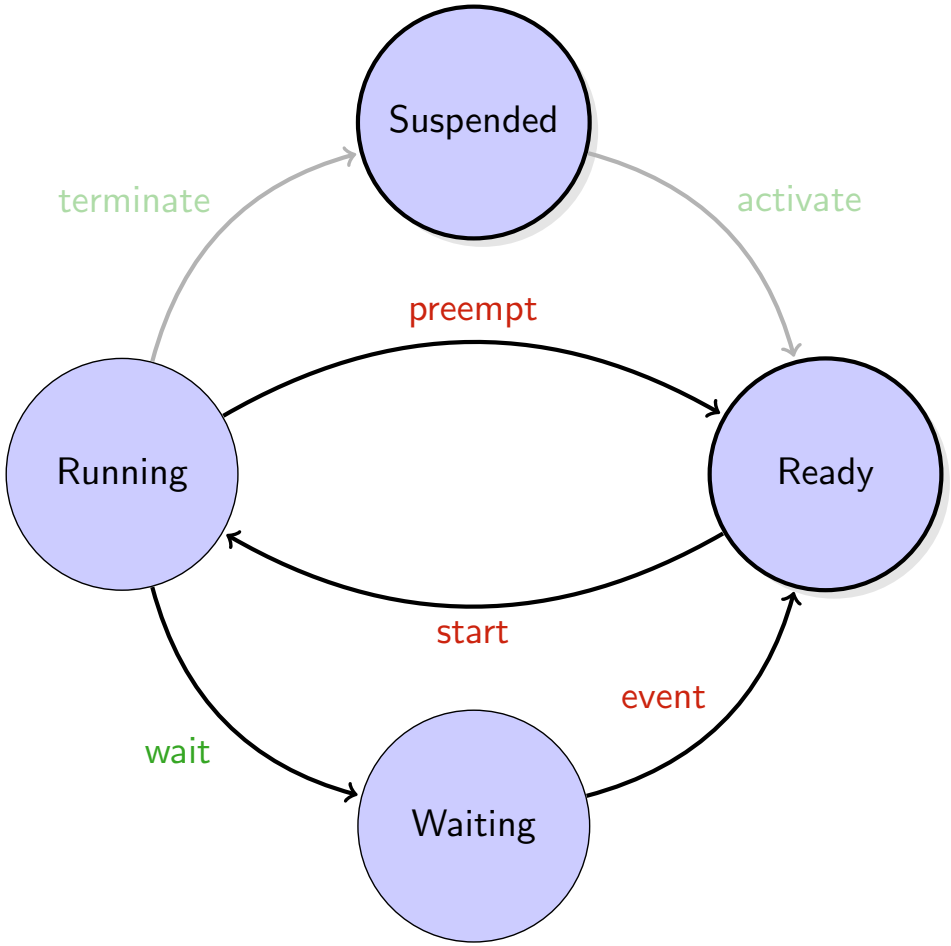


Hyp : Task1 and Task2 contain runnable(s) which share the same resource

Gestion des évènements

- un `event` est privé : c'est la propriété d'une seule tâche
- les évènements (`event`) sont basés sur le modèle N producteurs/1 consommateur ;
  - n'importe quelle tâche/ISR2 peut envoyer un évènement à une tâche ;
  - une tâche (et une seule) peut réceptionner l'évènement.
- 4 appels systèmes associés :
  - `setEvent(TaskId, eventMask)` ;
  - `waitEvent(eventMask)` ;
  - `getEvent(TaskId,eventMaskRef)` ;
  - `clearEvent(eventMask)`.

Modèle de tâche étendue



utilisation des évènements

```
TASK(Task1)
{
    doSth1();
    SetEvent(Task2, EV1);
    TerminateTask();
}

TASK(Task3)
{
    doSth3();
    SetEvent(Task2, EV2);
    TerminateTask();
}
```

```
TASK(Task2)
{
    EventMaskType event_got;
    init();
    while(1){
        WaitEvent(EV1 | EV2);
        GetEvent(Task2, &event_got);
        if (event_got & EV1) {
            ClearEvent(EV1);
            //manage EV1
        }
        if (event_got & EV2) {
            ClearEvent(EV2);
            //manage EV2
        }
    }
}
```

L'arrivée d'un évènement entre le `WaitEvent` et le `ClearEvent` est perdu !

AUTOSAR - les OS Applications

But : intégrer sur un calculateur des applications distinctes

- Regroupe des objets (tâches, ISR, alarmes, . . .) ;
- Possède un espace mémoire propre, l'IOC [1] permet une communication entre OS Applications ;
- Peut être arrêtée et démarrée ;
- Par défaut les objets d'une OS Application ne sont pas accessibles d'une autre OS Application ;
- Une OS Application est comparable à un processus mais elle est attribuée à un seul cœur ;
- Notion d'OS Application *Trusted* qui s'affranchie de la protection mémoire.

1. **I**nter **O**S Application **C**ommunication

AUTOSAR - La protection mémoire

AUTOSAR propose une protection mémoire des OS Applications

- Une OS Application possède un espace mémoire privé ;
- Pas d'espace mémoire commun à toutes les OS Application ;
- Notion de *Trusted Function* qui s'affranchie de la protection mémoire.

Optionnellement, une protection mémoire entre tâches peut être mise en œuvre (c'est le cas pour Trampoline)

- Système statique $\Longrightarrow$ toutes les allocations mémoires connues à l'édition de lien ;
- Utilisation d'une allocation mémoire indépendante du compilateur C via le préprocesseur.

```
#define APP_Task_task1_START_SEC_CONST_32BIT
#include "tpl_memmap.h"
CONST(int, AUTOMATIC) myVar = 0;
#define APP_Task_task1_STOP_SEC_CONST_32BIT
#include "tpl_memmap.h"
```

AUTOSAR - La protection temporelle

Le bon fonctionnement de l'application dépend du temps d'exécution maximum des tâches. $\Longrightarrow$ mécanisme de contrôle de non dépassement du temps d'exécution maximum.

- Chaque tâche a un budget ;
- Le budget est consommé quand la tâche s'exécute ;
- Si le budget est dépassé, l'OS peut :
  - Ne rien faire ;
  - Tuer l'instance de la tâche ;
  - Tuer l'OS Application à laquelle appartient la tâche ;
  - Redémarrer l'OS Application à laquelle appartient la tâche ;
  - Arrêter l'OS.

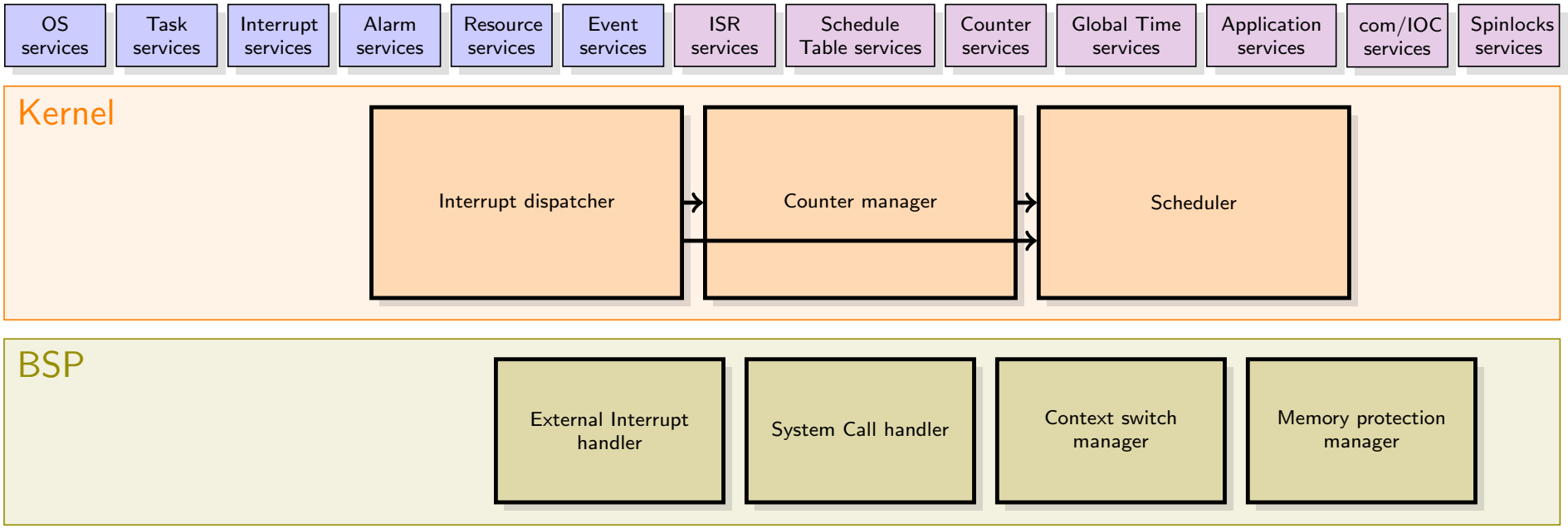
Ordonnancement

- ordonnancement *partitionné* : Chaque tâche est affecté (statiquement) à un cœur ;
  $\Longrightarrow$ il y a une liste des tâches prêtes pour chaque cœur
- ordonnancement *global* : les tâches peuvent migrer d'un cœur à l'autre dynamiquement.
  $\Longrightarrow$ il y a une liste des tâches prêtes unique
- ordonnancement *mixte* : combinaison des 2 : des tâches sont affectées statiquement à un cœur, d'autres peuvent migrer.

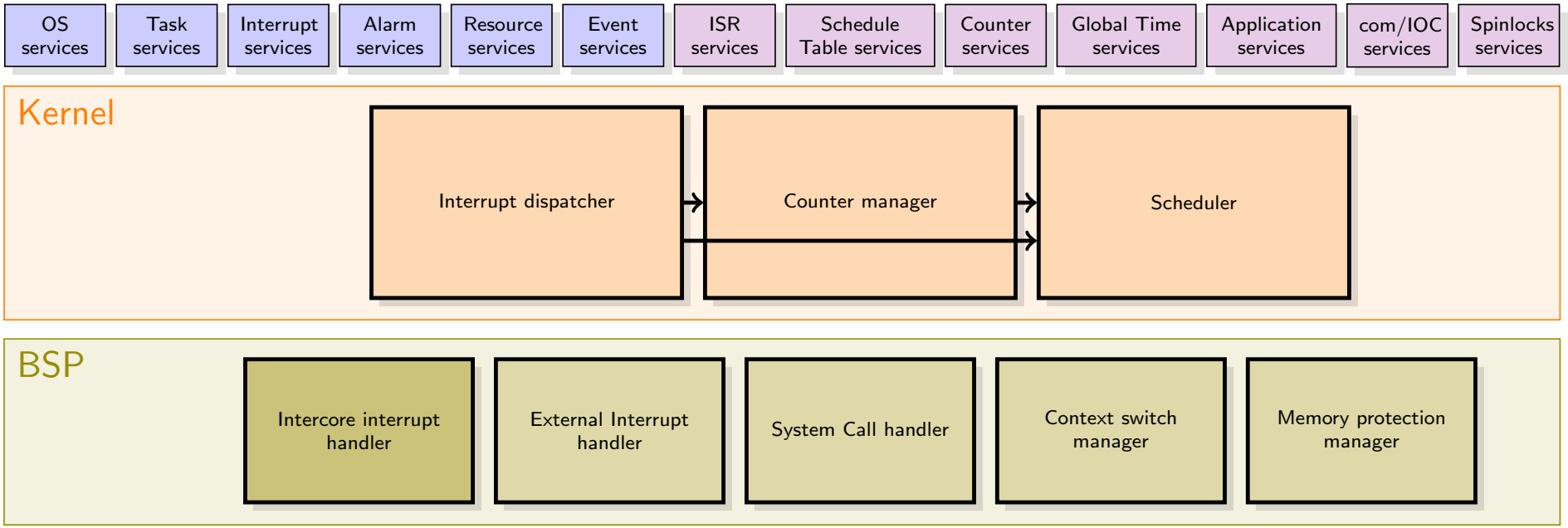
Ordonnancement

- ordonnancement *partitionné* : Chaque tâche est affecté (statiquement) à un cœur ;
  $\Longrightarrow$ il y a une liste des tâches prêtes pour chaque cœur
- ordonnancement *global* : les tâches peuvent migrer d'un cœur à l'autre dynamiquement.
  $\Longrightarrow$ il y a une liste des tâches prêtes unique
- ordonnancement *mixte* : combinaison des 2 : des tâches sont affectées statiquement à un cœur, d'autres peuvent migrer.

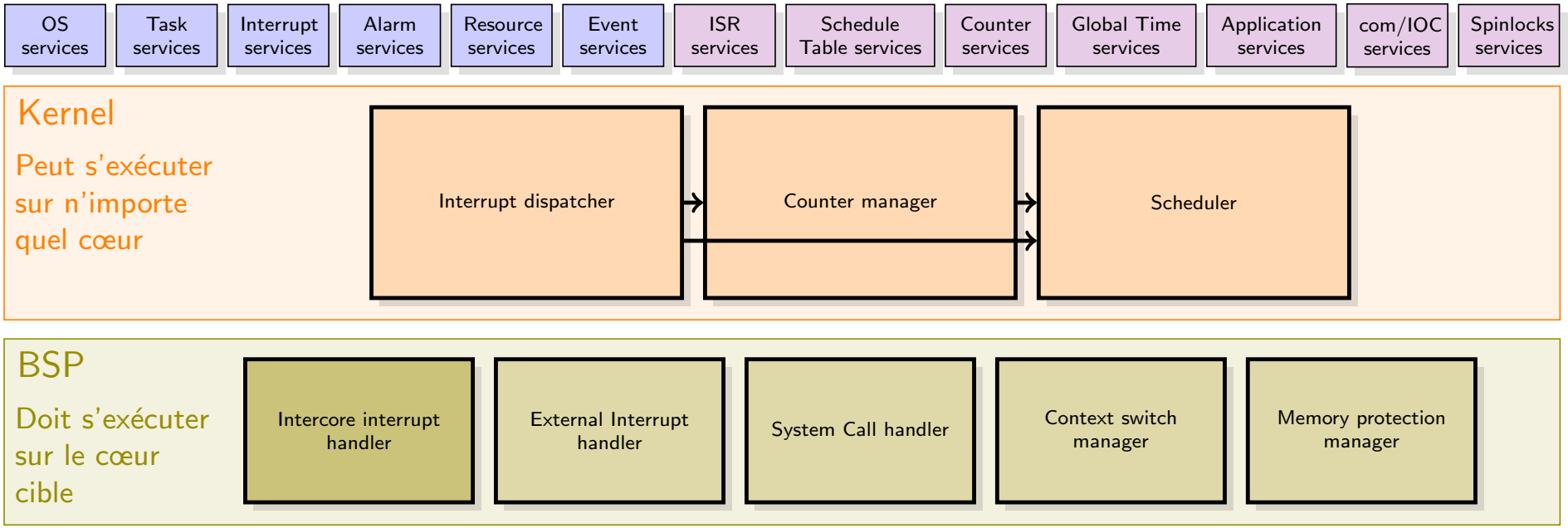
Architecture multicœur de Trampoline



Architecture multicœur de Trampoline



Architecture multicœur de Trampoline



Les Contraintes Automobile

- Conception des fonctions du véhicule (vue système) ;
- Méthodes et outillages de répartition des fonctions sur calculateurs ;
- Répartition des responsabilités / intégration de logiciels tiers ;
- Contraintes de Coût ;
- Contraintes de Sureté de fonctionnement/Fiabilité ;

Répartition de fonctions sur le véhicule

SwitchEvent

switch_event (event)

request_light (type, mode)

AUTOSAR Interface

LightRequest

switch_event(event)

request_light (type, mode)

AUTOSAR Interface

Front-Light Manager

request_light(type, mode)

set_light(type, mode)

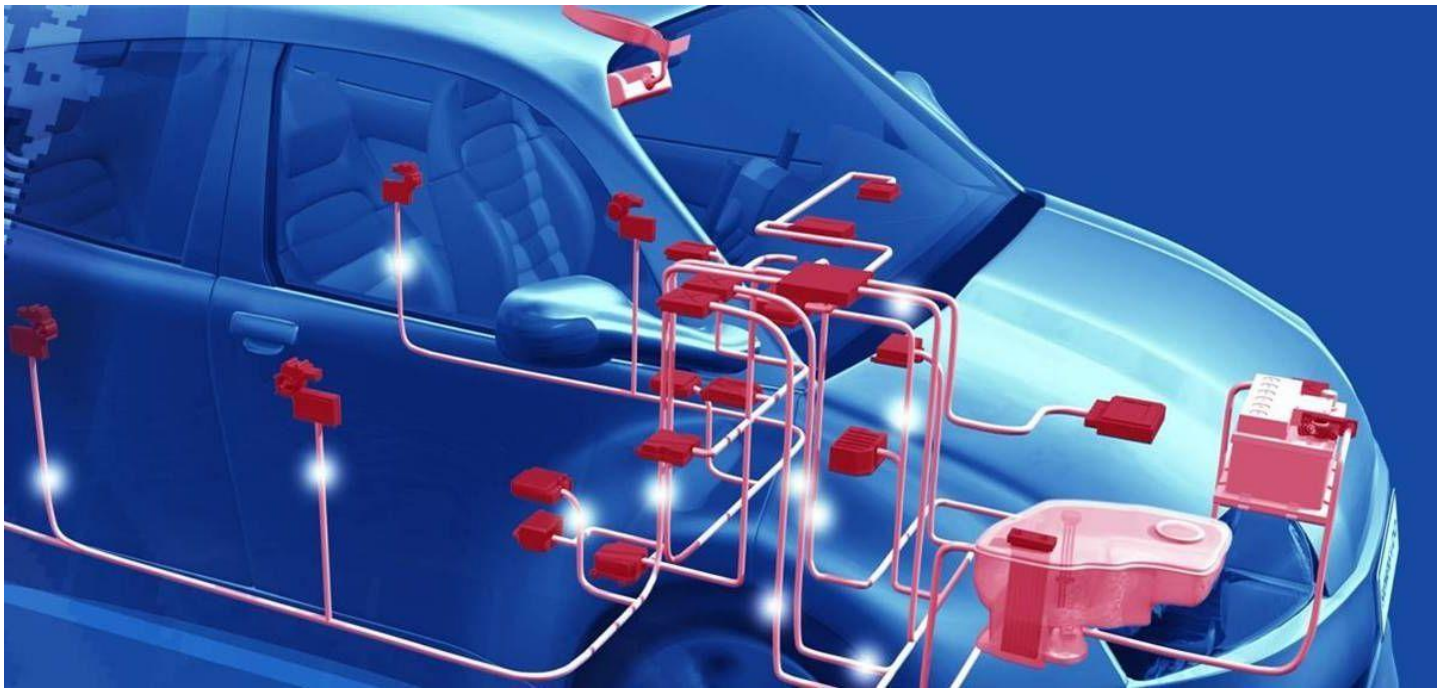
AUTOSAR Interface

Xenonlight

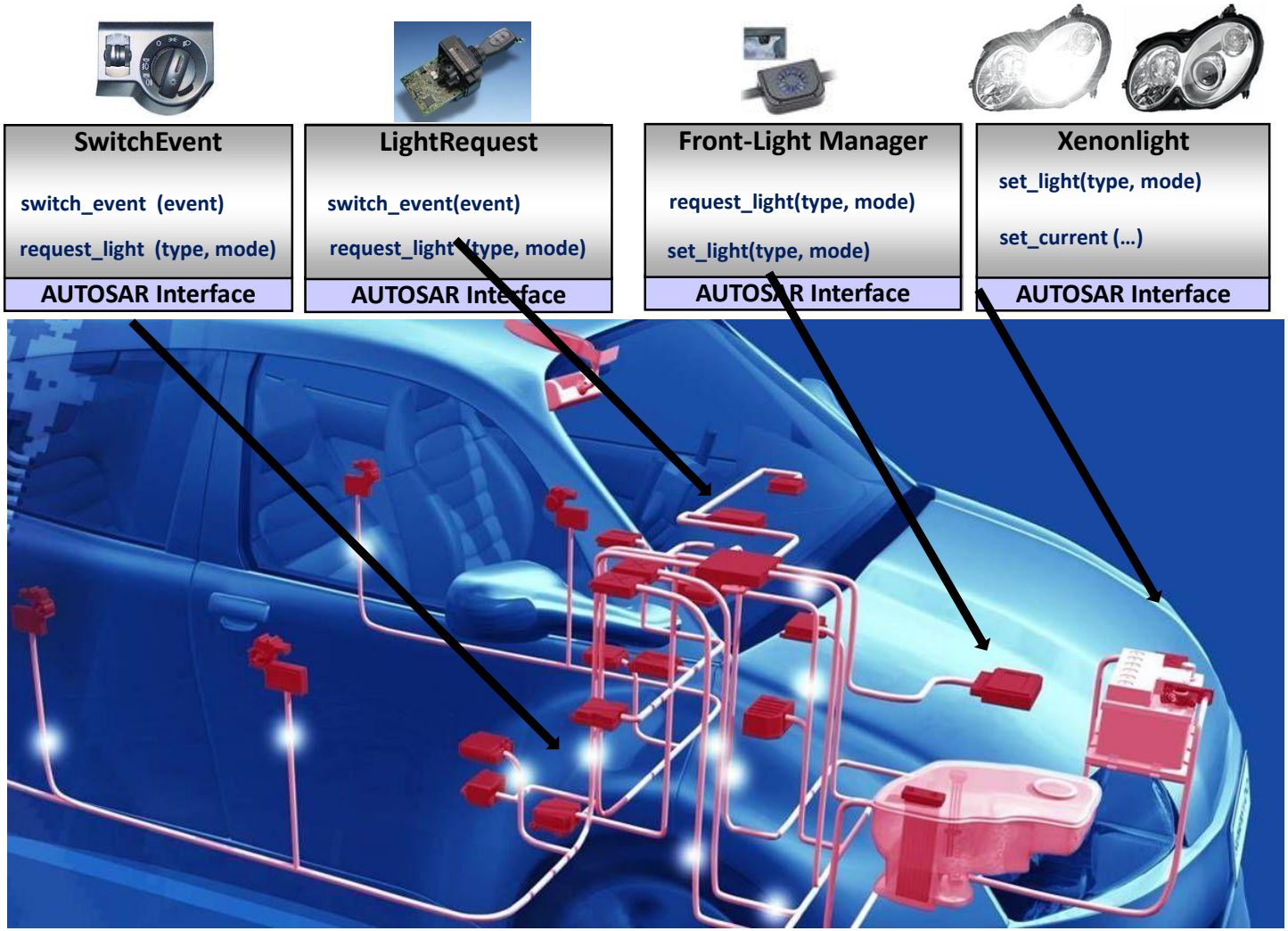
set_light(type, mode)

set_current (...)

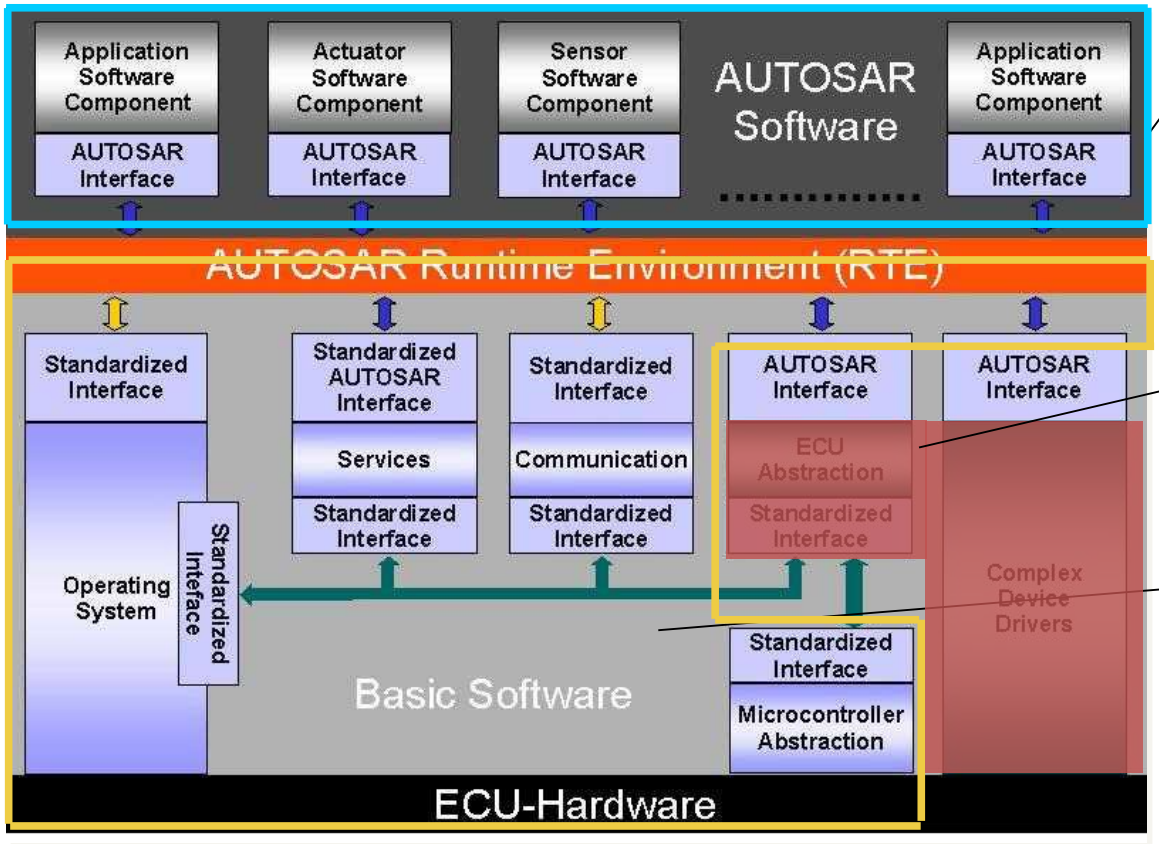
AUTOSAR Interface



Répartition de fonctions sur le véhicule



Architecture d'un calculateur - Architecture AUTOSAR



Application Software (ASW)

Standardization of interfaces

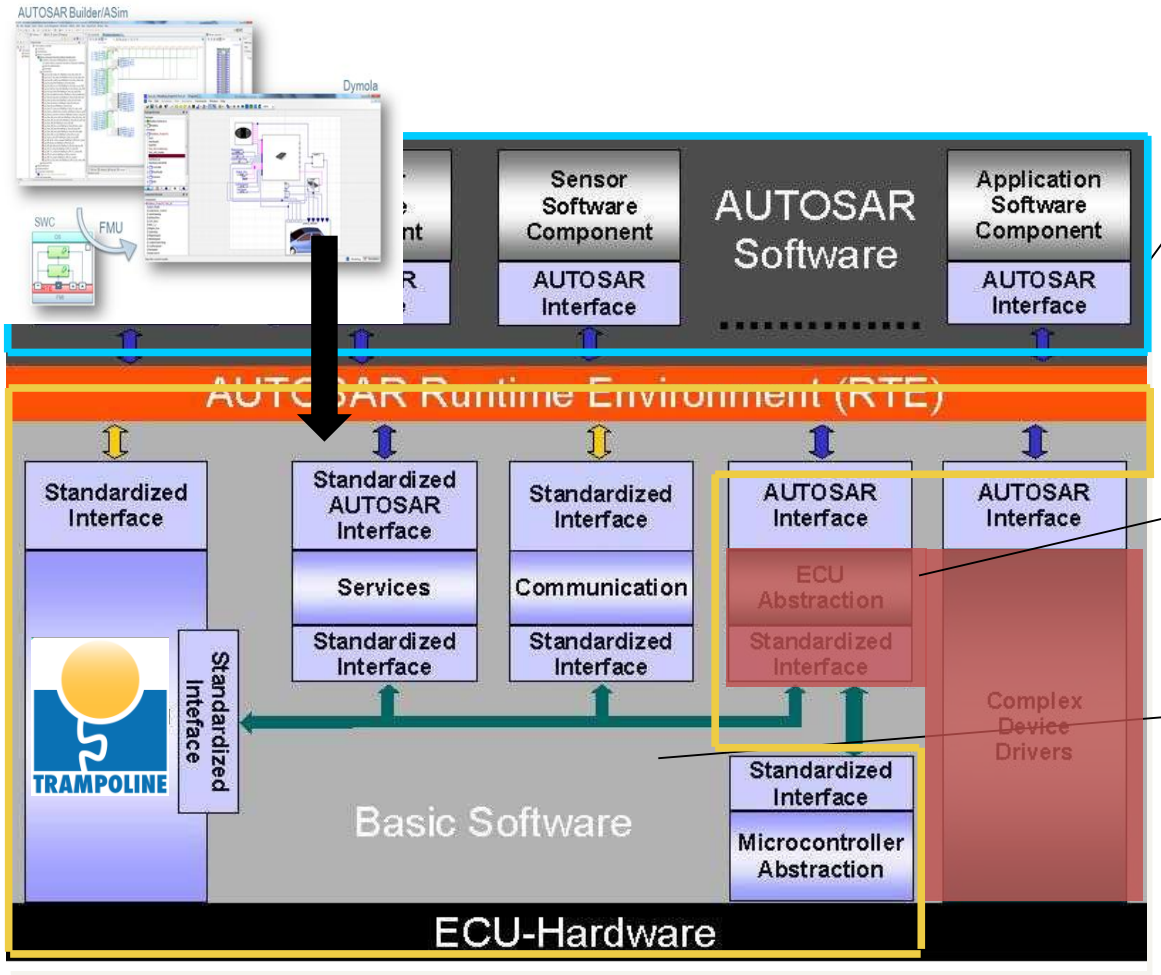
Not standardized in AUTOSAR

Basic Software (BSW) +RTE

Standardization of interfaces and behavior

Objectives: Basic SW: Decoupling of Hardware and Application Software
Application SW: Relocation / Reuse of SW-Components between ECUs

Architecture d'un calculateur - Architecture AUTOSAR



Application Software (ASW)

Standardization of interfaces

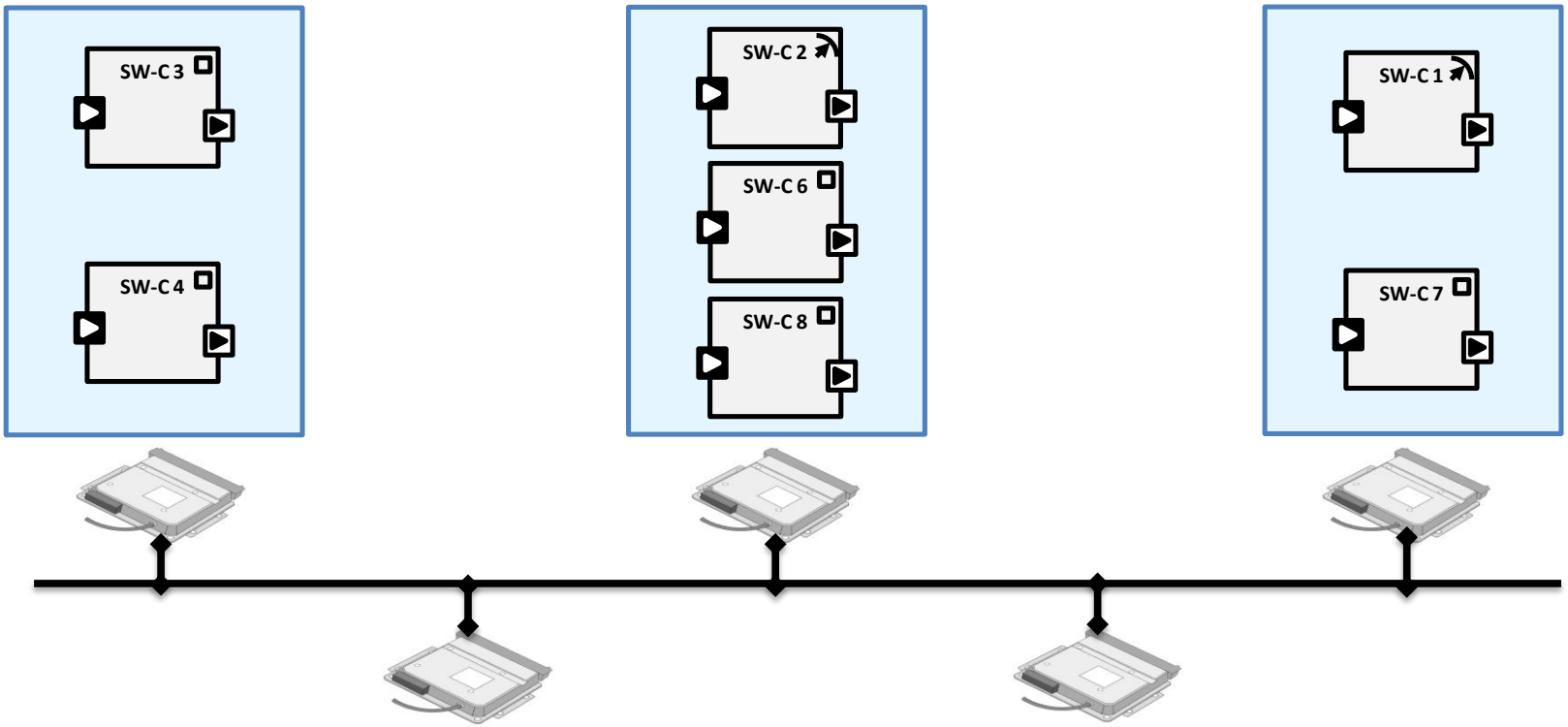
Not standardized in AUTOSAR

Basic Software (BSW) +RTE

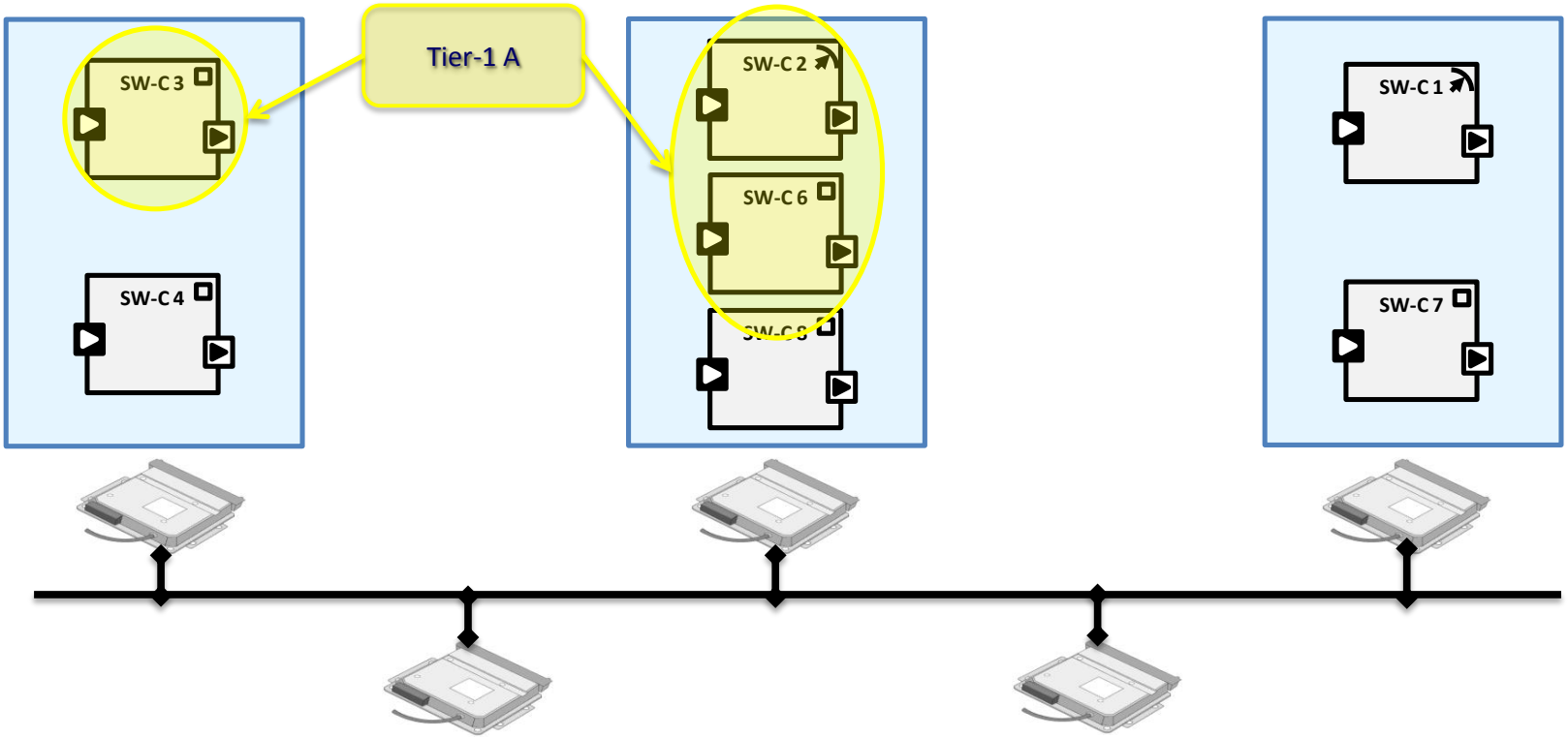
Standardization of interfaces and behavior

Objectives: Basic SW: Decoupling of Hardware and Application Software
Application SW: Relocation / Reuse of SW-Components between ECUs

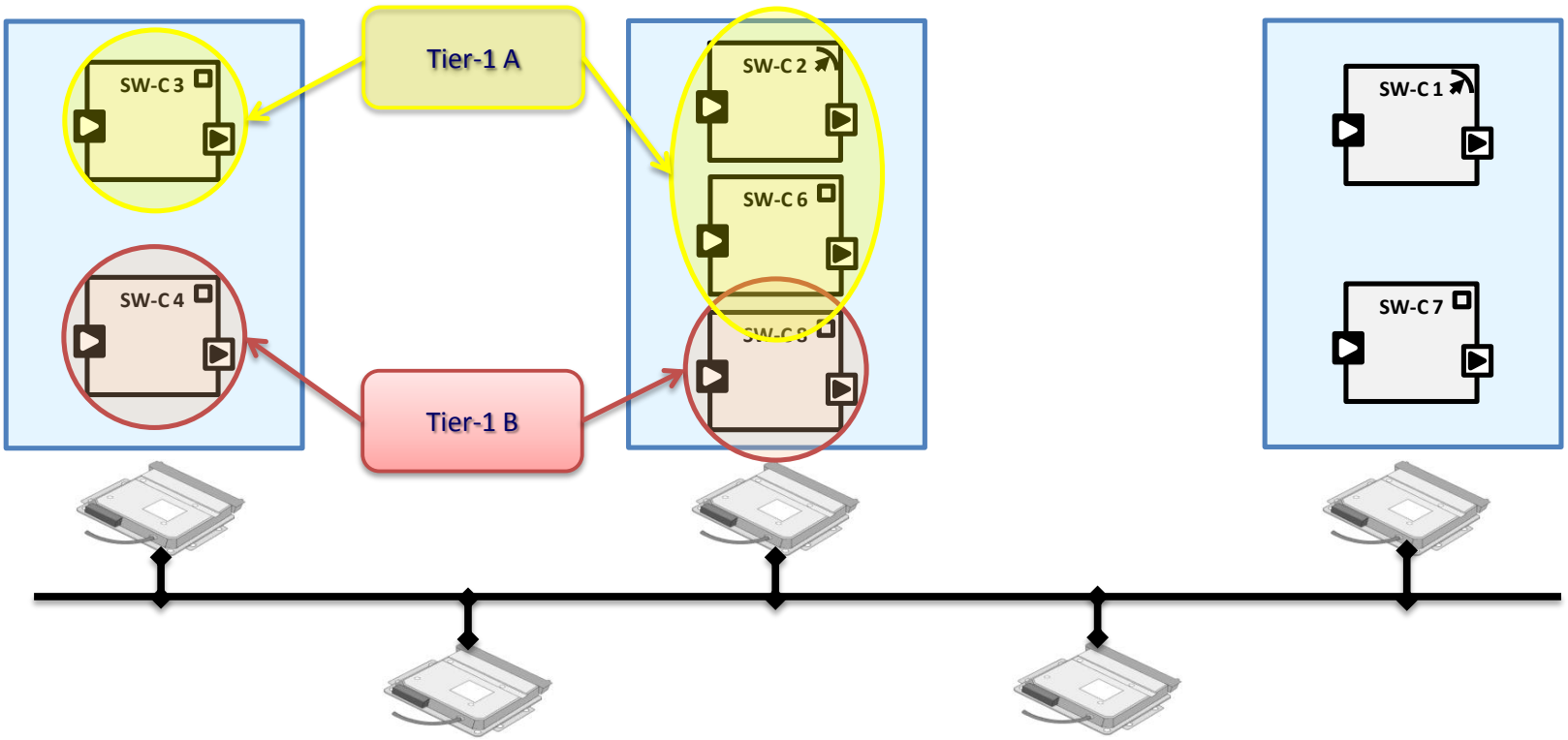
Répartition/Allocation de responsabilités 1/2



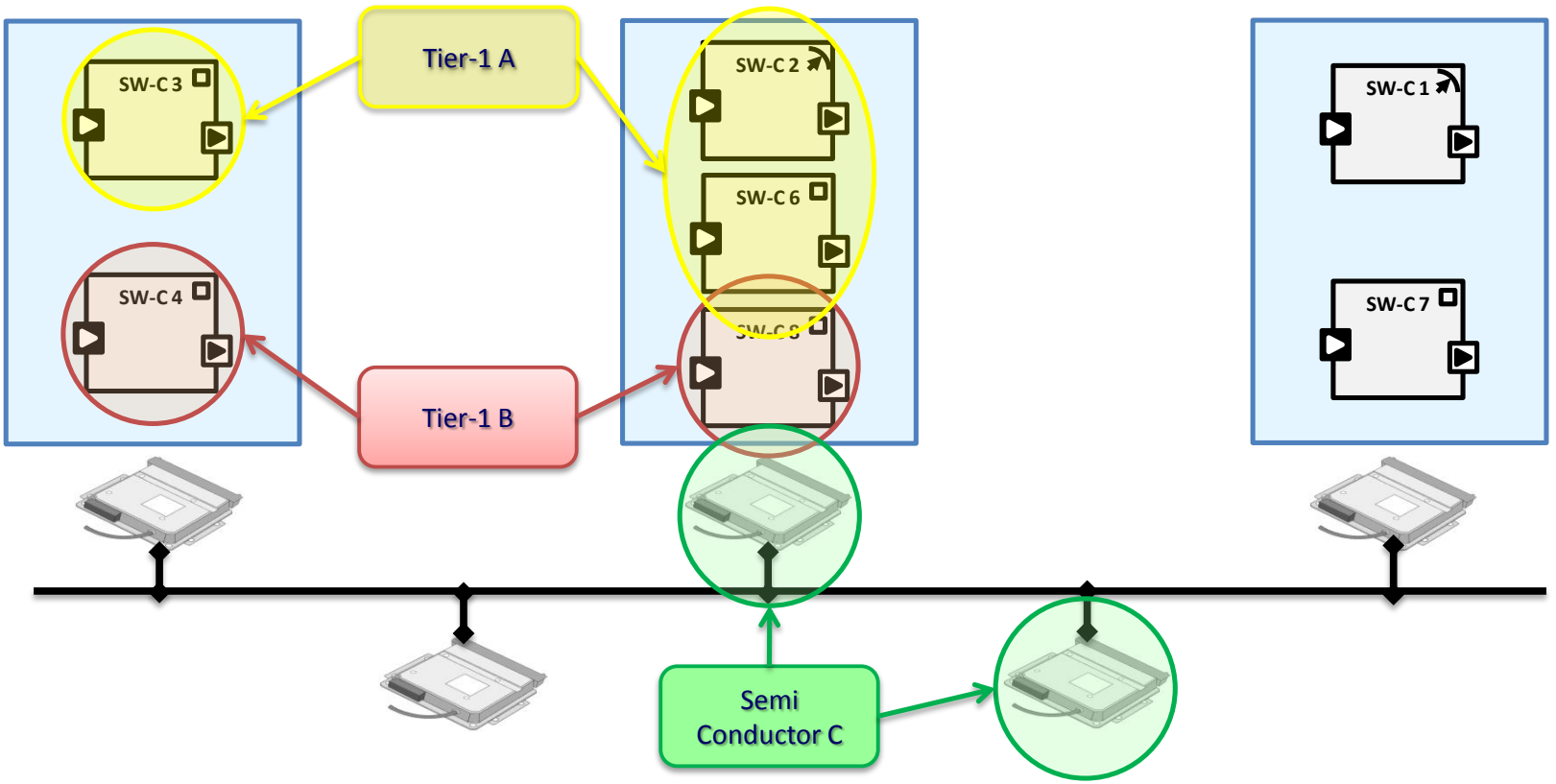
Répartition/Allocation de responsabilités 1/2



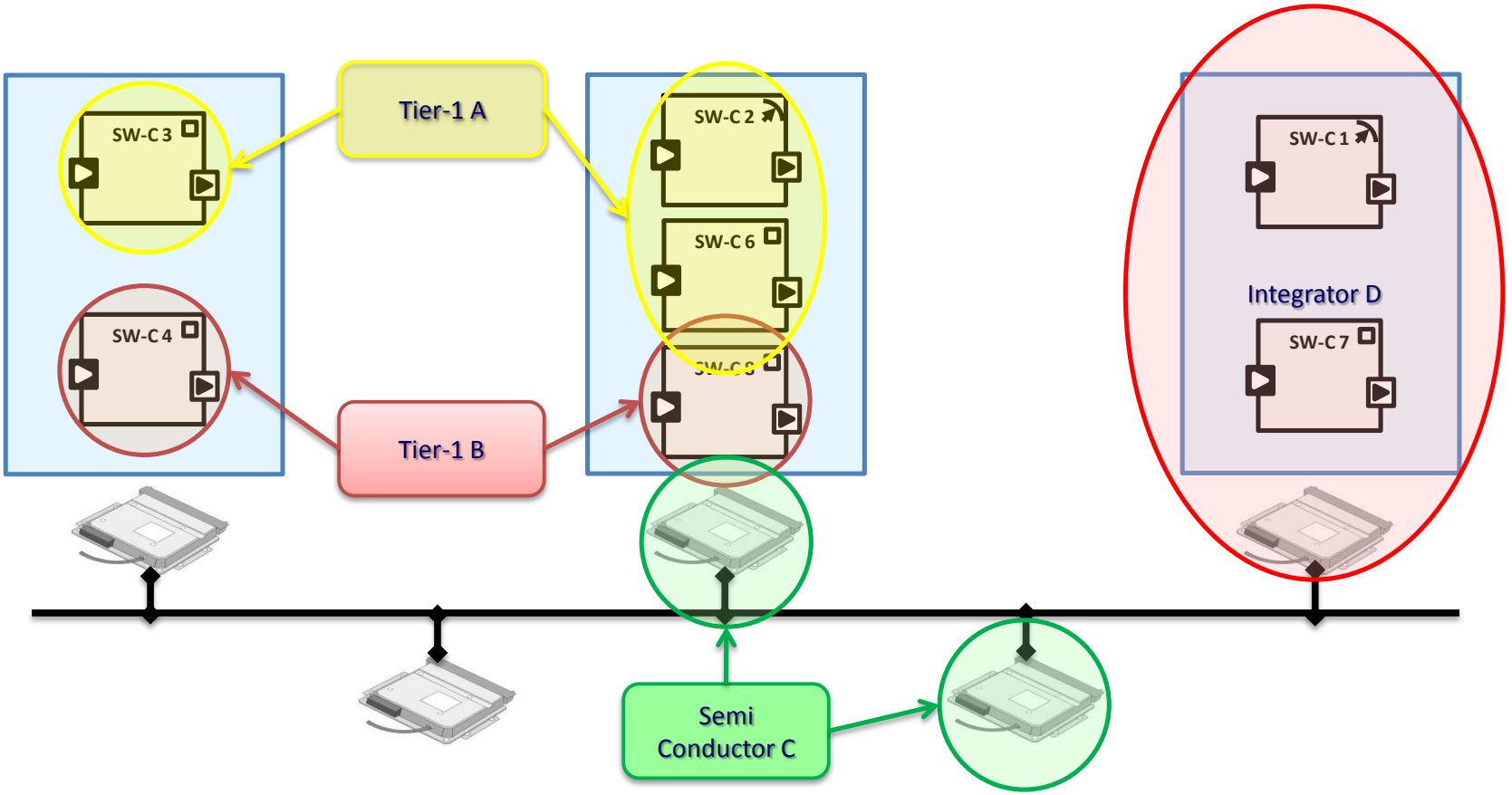
Répartition/Allocation de responsabilités 1/2



Répartition/Allocation de responsabilités 1/2



Répartition/Allocation de responsabilités 1/2



Répartition/Allocation de responsabilités 2/2

Avantages du standard AUTOSAR

- Interfaces standardisées entre applications provenant de différentes sources ;
- Interfaces standardisées vers les couches basses (OS, pilotes de périphériques) ;
- Applications logicielles indépendantes du matériel ;
- Utilisation des principes OSEK de génération de code statiquement par configuration ;
- Outillages disponible pour la répartition des applications logicielles sur les cibles.

Définitions AUTOSAR 1/2

Partitionnement - Définition AUTOSAR

partitionnement :
consiste à grouper un ensemble d'applications logicielles au sein d'une partition. Tous les éléments associés à cette partition (tâches, isr, événements, ...) peuvent partager une même zone mémoire. Si un accès mémoire à une zone non autorisée pour une partition est réalisée, l'OS utilise alors des mécanismes de protection mémoire mis à disposition par le microcontroleur.

Définitions AUTOSAR 2/2

IOC - Définition AUTOSAR

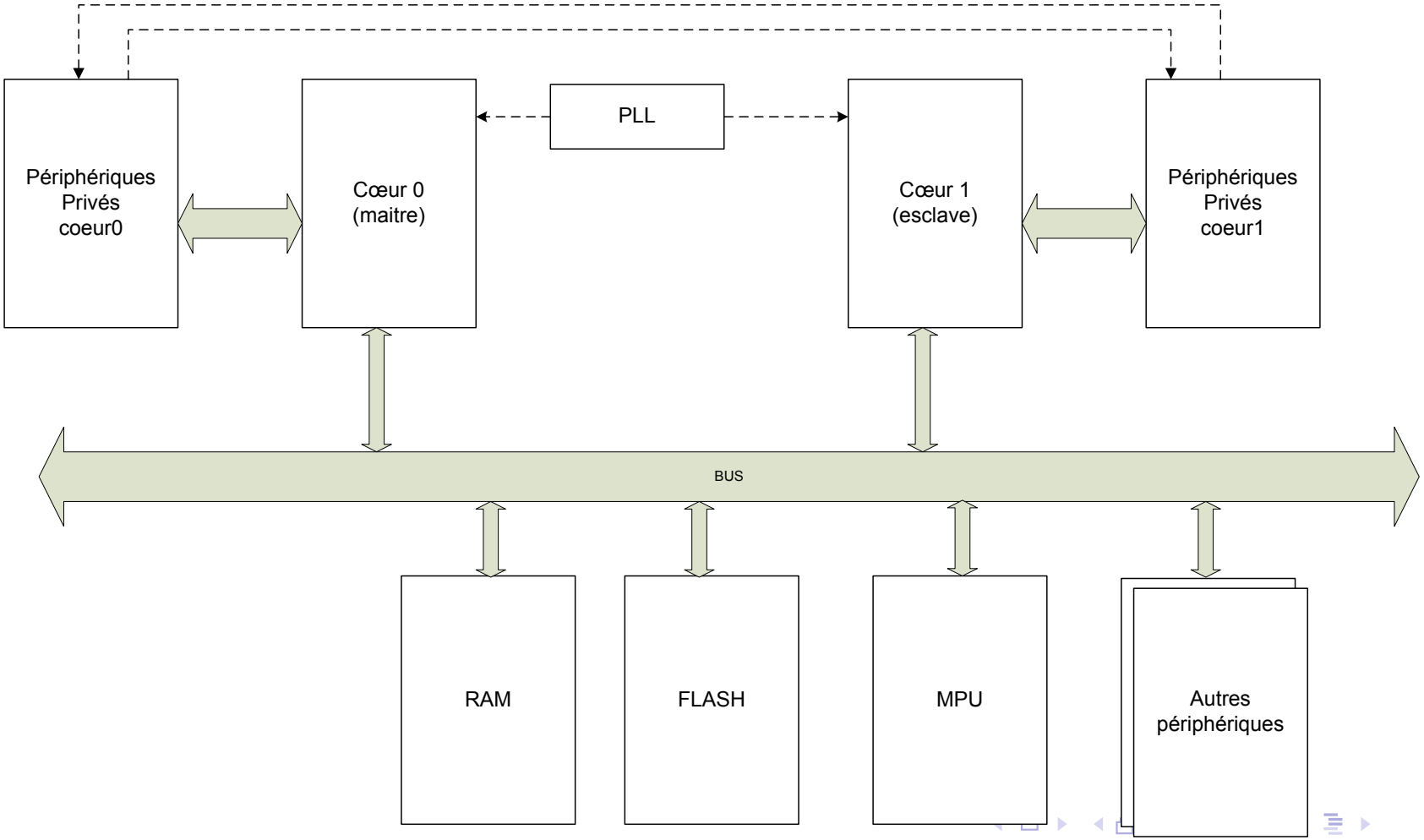
Inter-OsApplications Communication permet d'envoyer et de recevoir des messages entre des tâches appartenant à 2 partitions différentes.

Trusted / Not Trusted

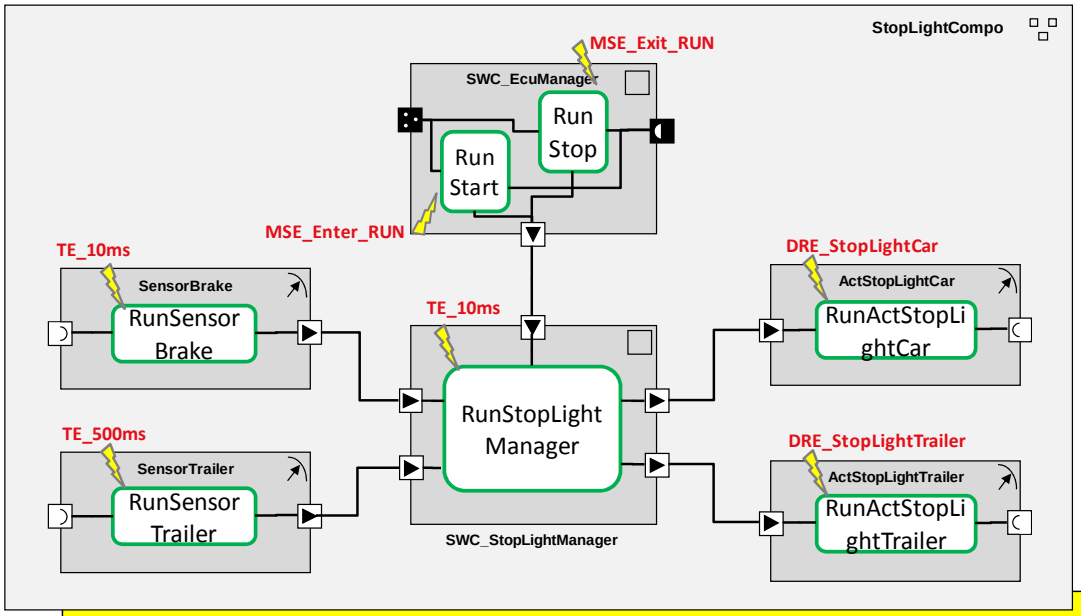
Trusted Une application Trusted s'exécute en mode *superviseur* et dispose d'un accès libre à la mémoire ;

Not Trusted Une application Not Trusted s'exécute en mode *utilisateur* et dispose d'un accès limité à la mémoire.

Architecture matérielle - principes

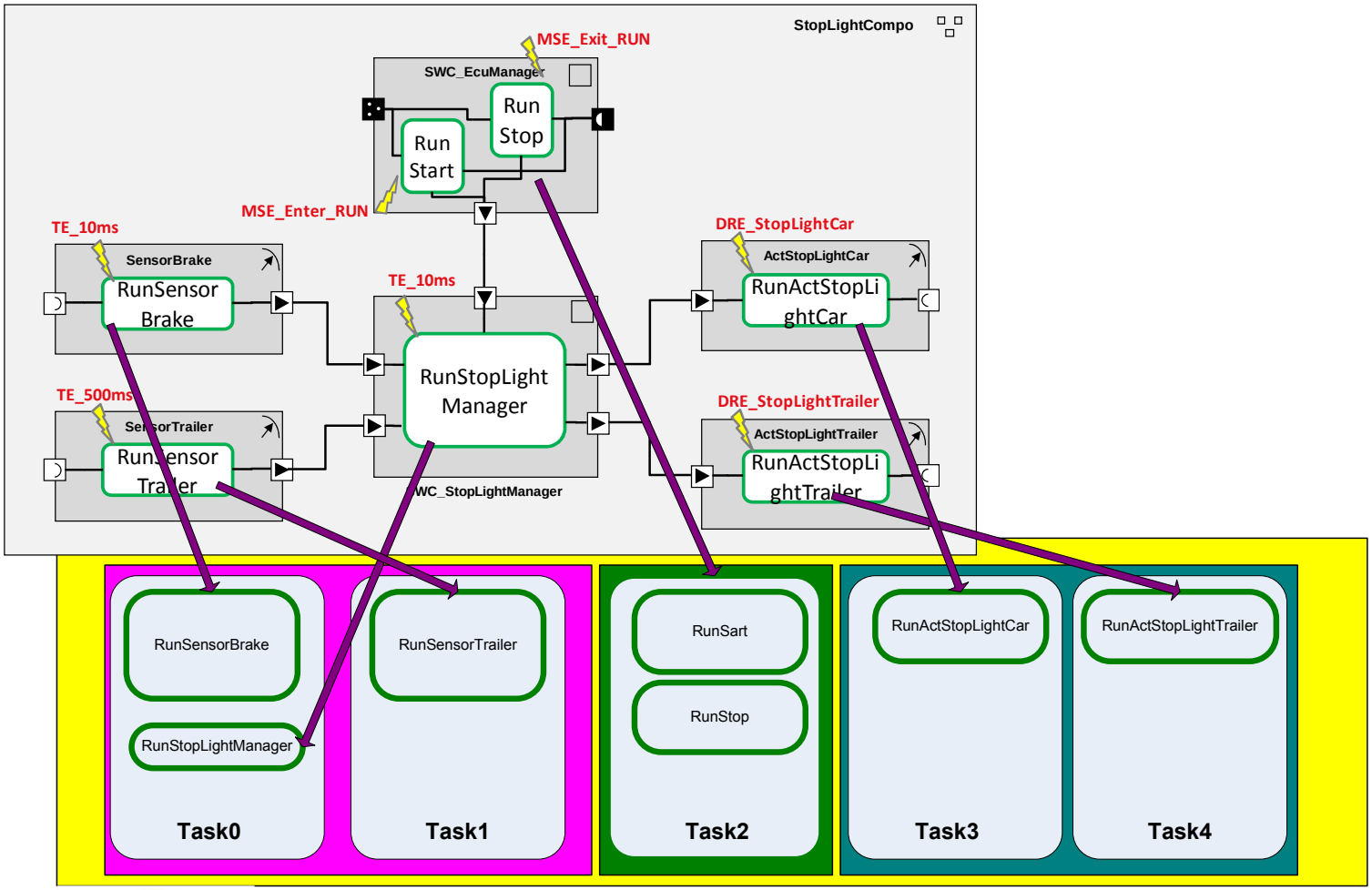


Les standard AUTOSAR OS et RTE

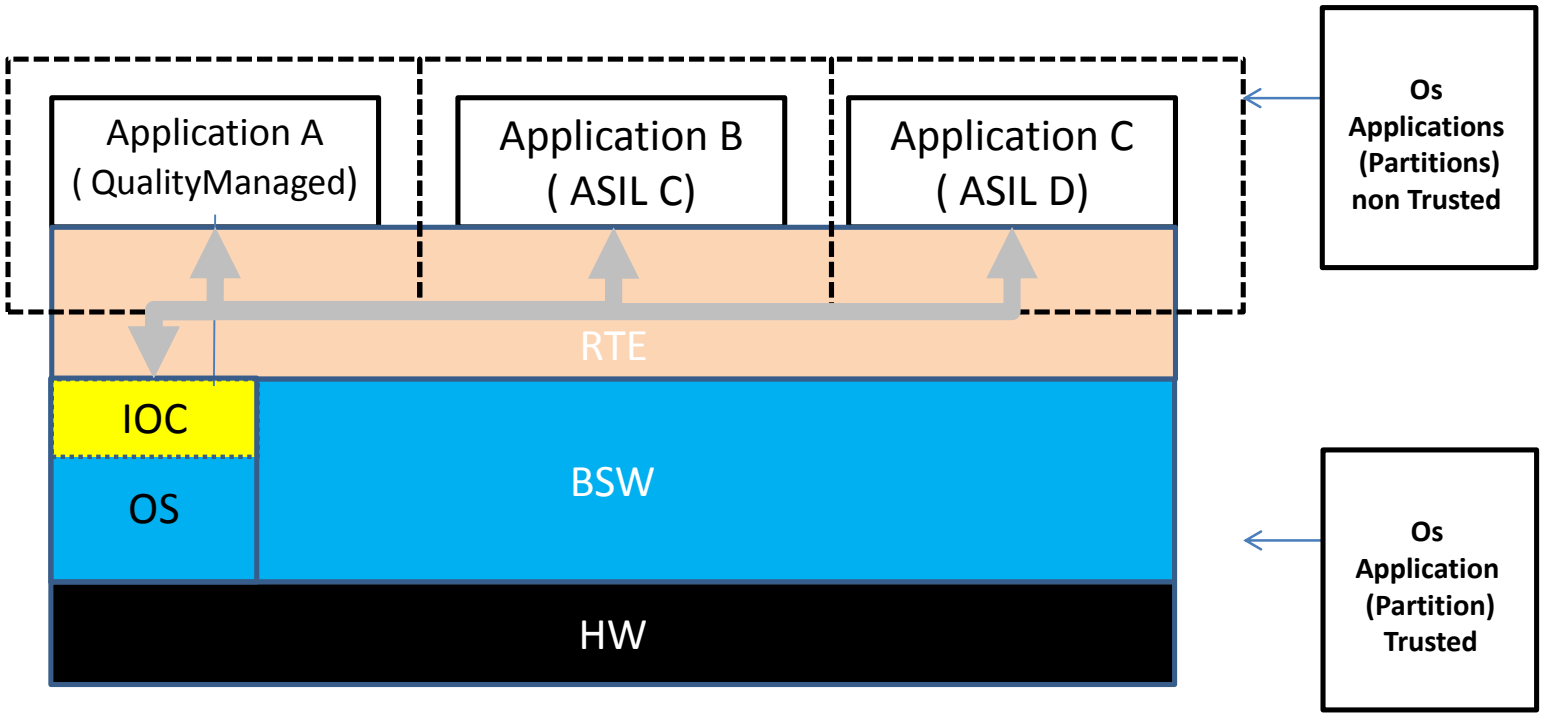


RTE

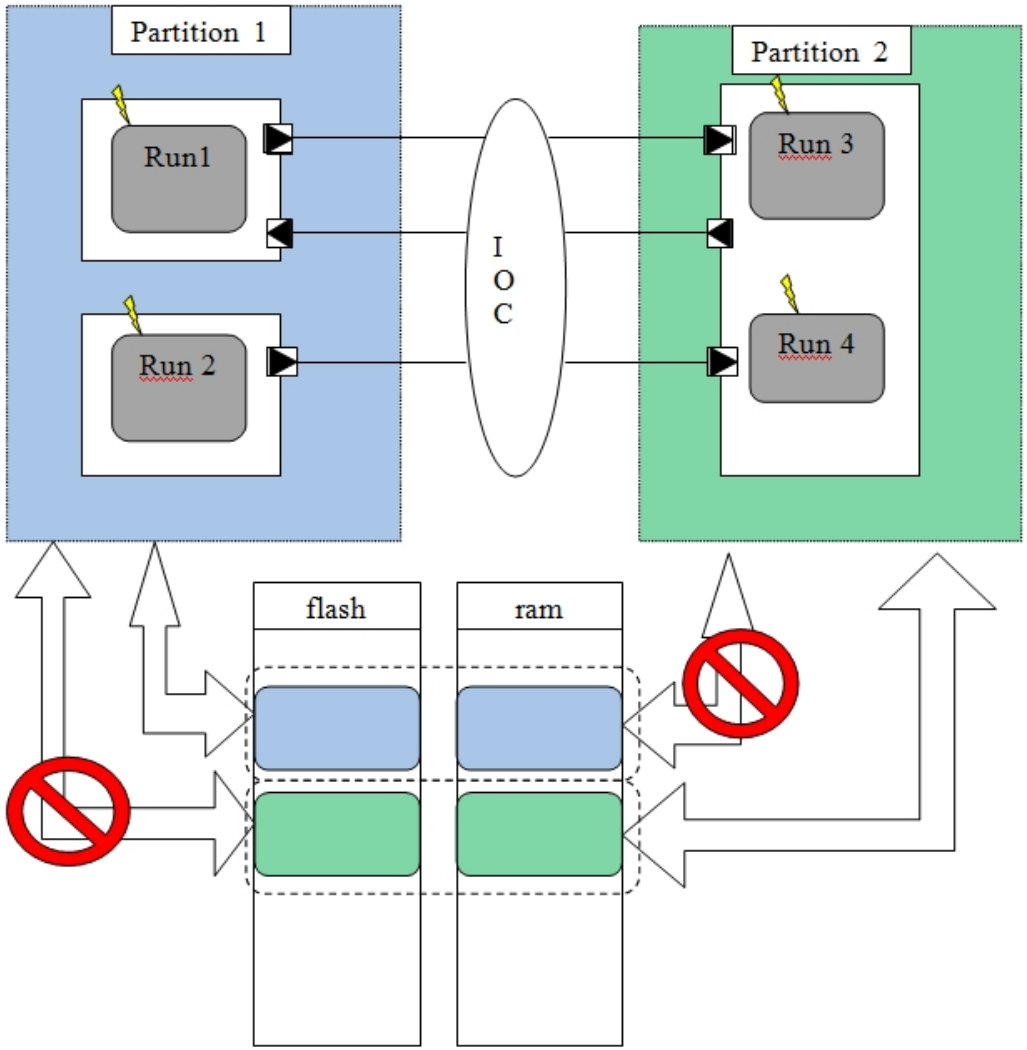
Les standard AUTOSAR OS et RTE



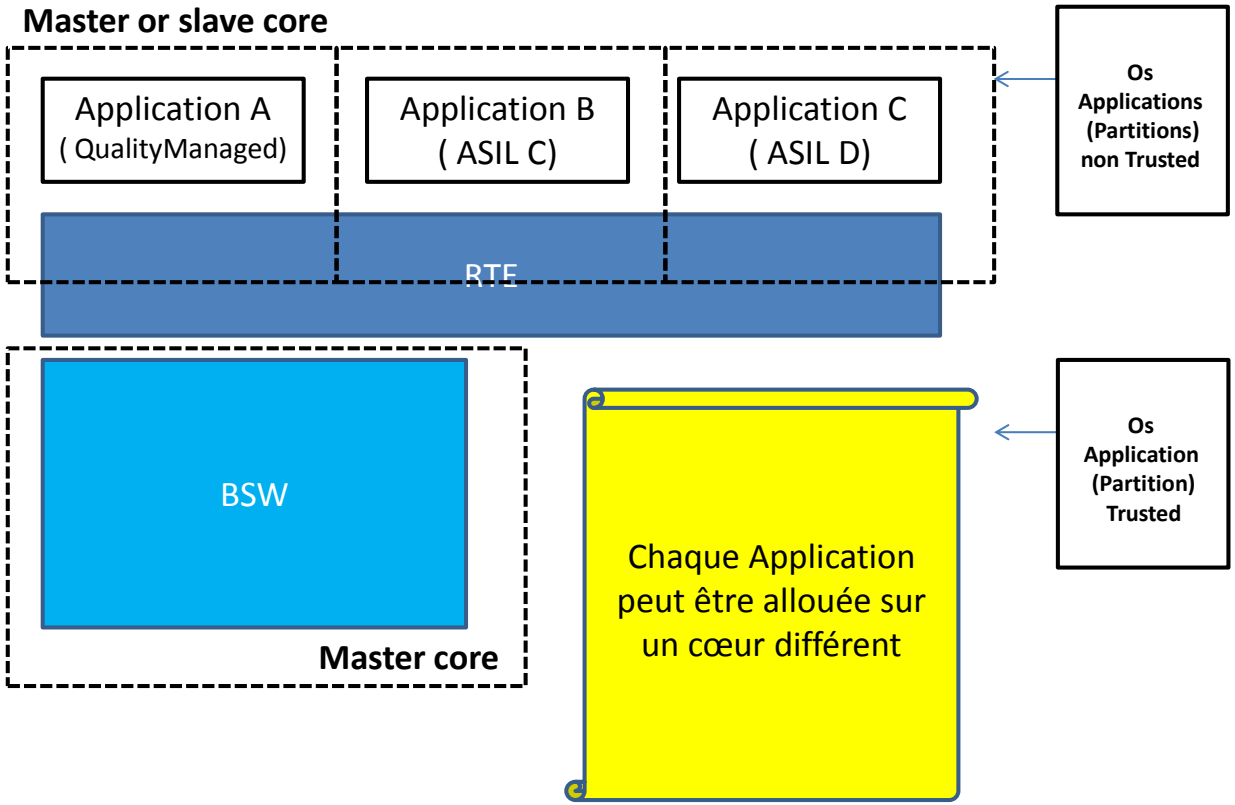
Le partitionnement d'applications - Isolation d'applications de différents niveaux de sécurité



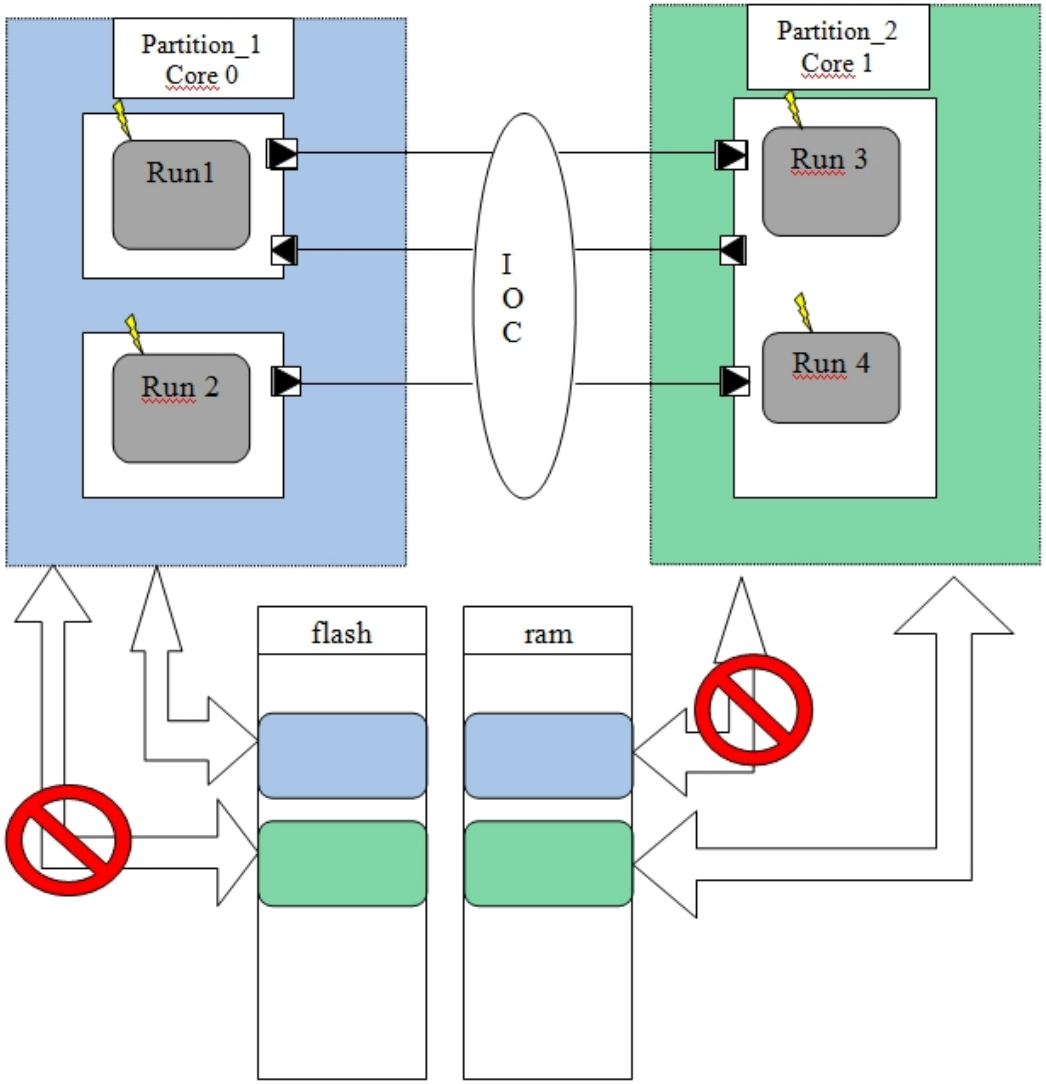
Protection Mémoire entre applications logicielles



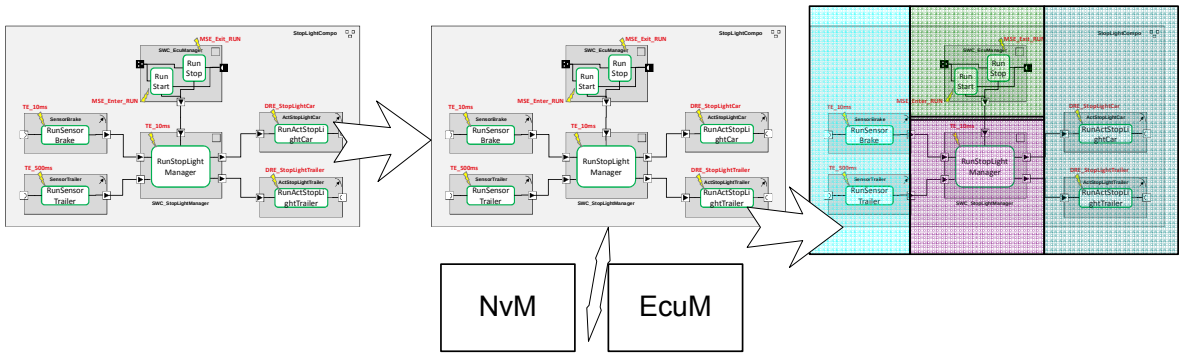
Application Multicœur partitionnée



Application Multicœur partitionnée



Chaîne outillée



Architecture fonctionnelle Non partitionnée monocoeur

Interconnection de l'application sur les services fournis par le calculateur (accès mémoire non volatile, réseaux can/lin/flexray, entrées/sorties,...)

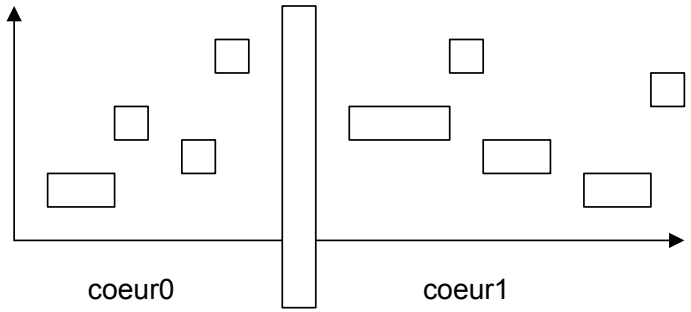
« Partitionnement » de l'application (définition des OsApplications, répartition sur les cœurs)

Génération de code BSW / OS / RTE

sources

Simulation Temps-réel

Analyse d'ordonnançabilité Analyse des contraintes de temps



Partitionnement, Multicœur et AUTOSAR

Pourquoi le partitionnement en automobile ?

- Achat de logiciels tiers ;
- Ségrégation spatiale entre applications de différents niveaux de sureté.

Partitionnement, Multicœur et AUTOSAR

Pourquoi le partitionnement en automobile ?

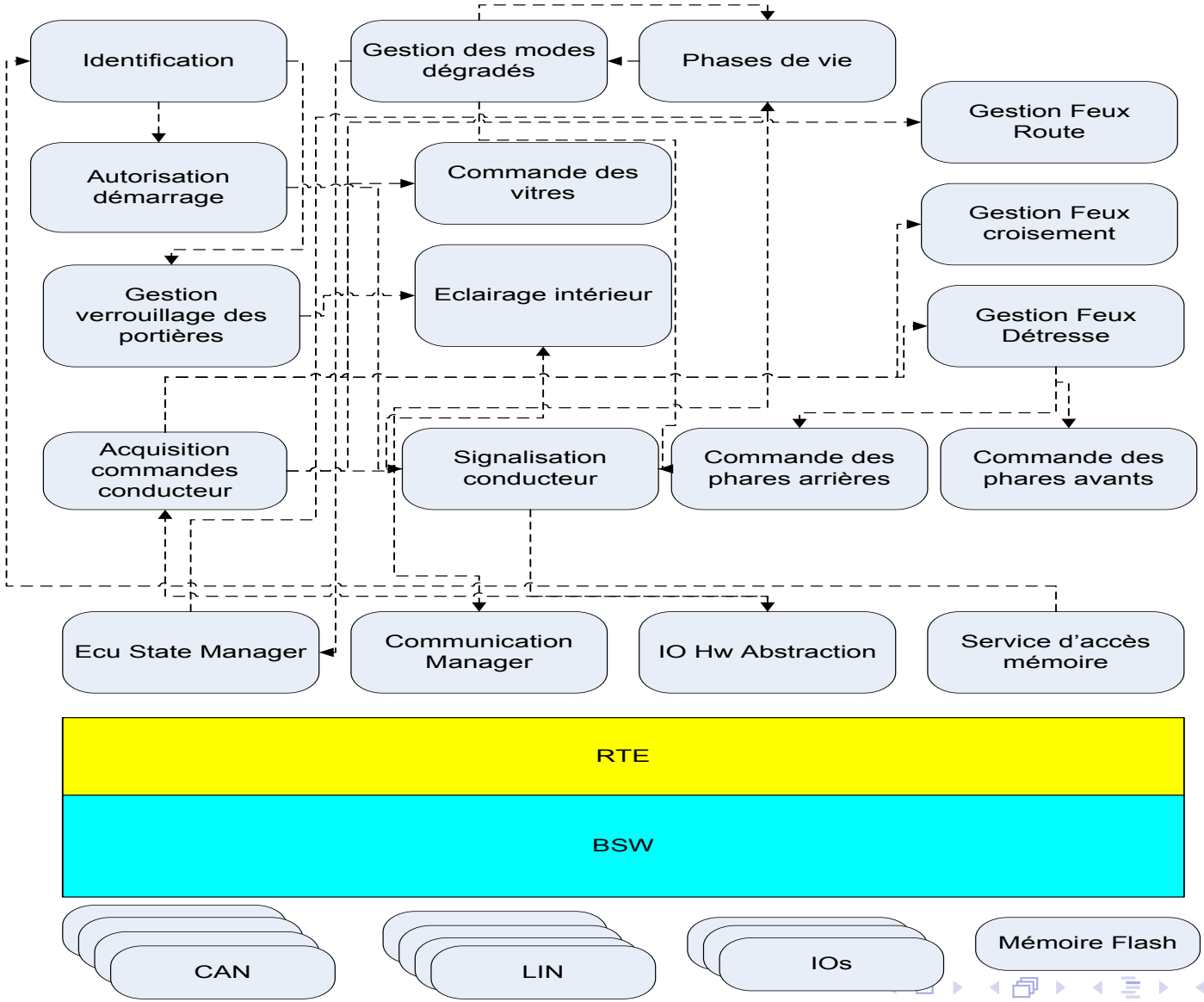
- Achat de logiciels tiers ;
- Ségrégation spatiale entre applications de différents niveaux de sureté.

Partitionnement, Multicœur et AUTOSAR

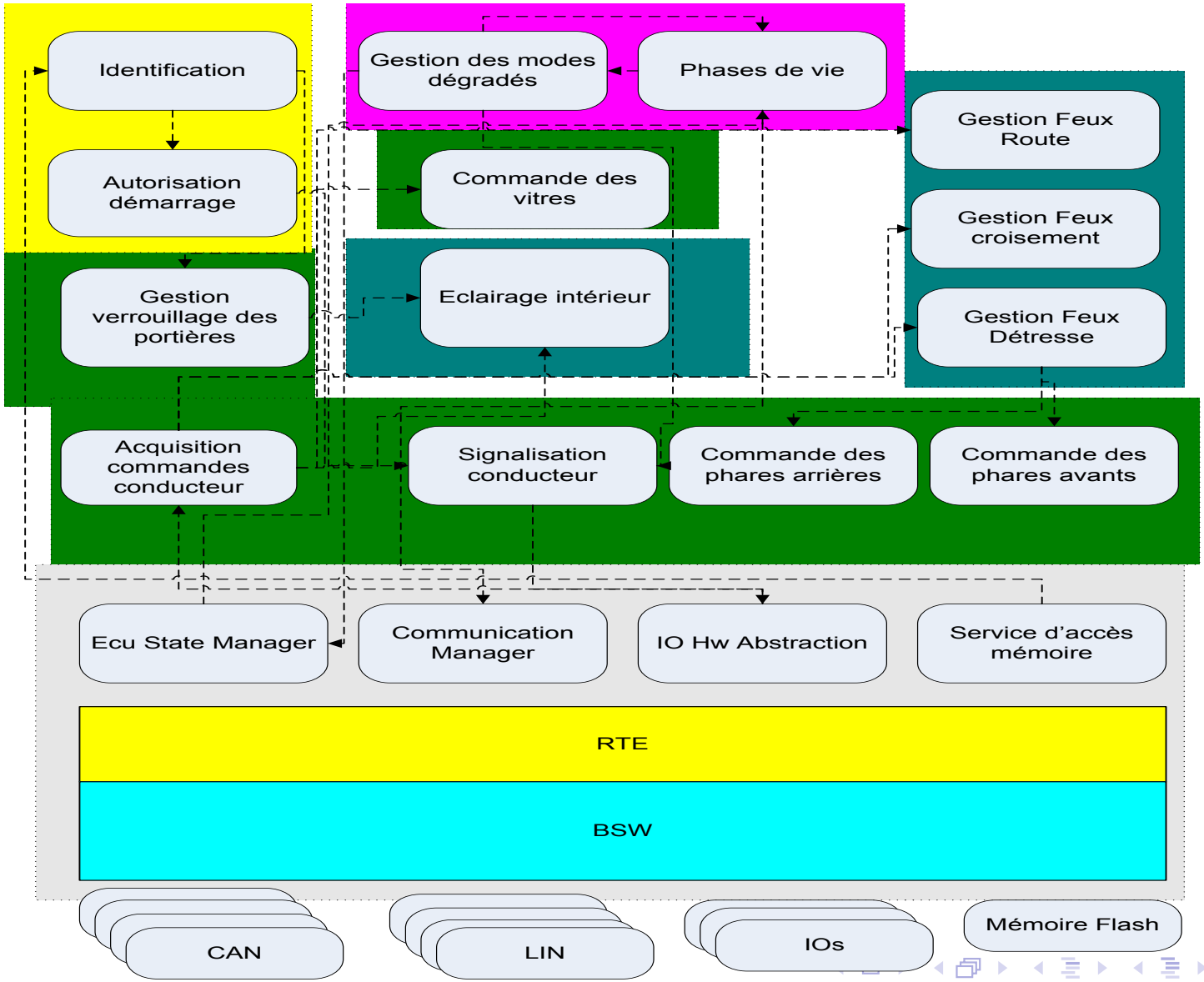
Pourquoi le multicœur en automobile ?

- Gain de de Performances ;
- Fusion de fonctions ;
- **Note :** Des applications situées sur 2 cœurs différents doivent être localisées dans 2 partitions différentes.

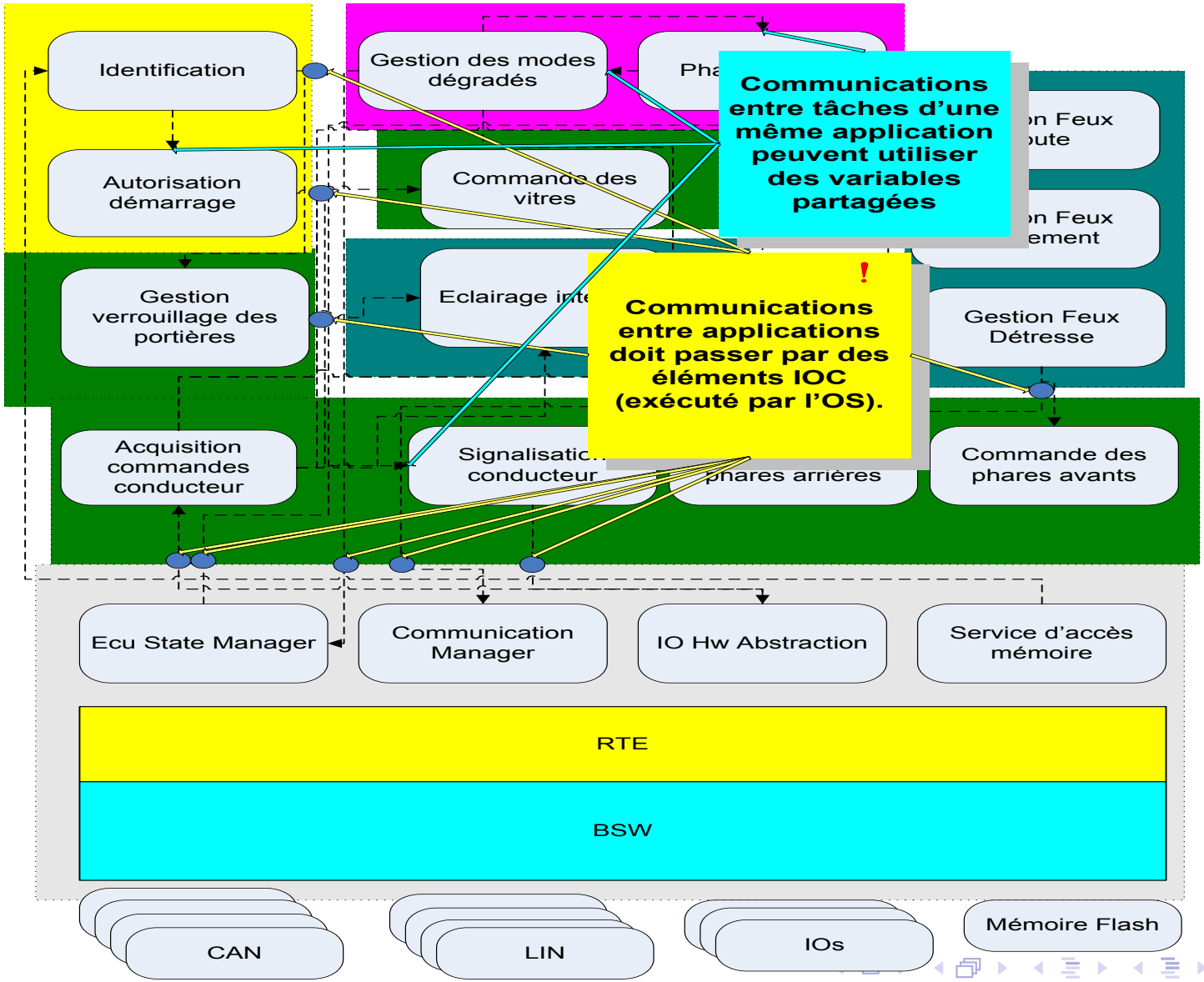
Exemple d'application : calculateur d'habitacle



Exemple d'application : calculateur d'habitacle



Exemple d'application : calculateur d'habitacle



Constats entre partitionné / non partitionné

Application Habitacle Non Partitionnée

- Nombre de tâches faible, contraintes de temps faibles ;
- Communication inter-tâches sans messagerie OS (variables partagées en mémoire) ;
- Corps des fonctions/tâches généré par des outils ;
- Une erreur arrivant dans une tâche impacte directement les autres tâches.

Application Habitacle Partitionnée

- Plus de tâches (1 tâche ne peut etre répartie entre 2 OsApplications) ;
- Communication inter-OsApplication par messagerie : ajout de délais de latence ;
- Possibilité de surveiller les temps d'exécution des tâches individuellement ;
- Les zones mémoires de chacune des partitions **not trusted** sont protégées.

Constats entre multicœur / monocœur

Application Habitacle Monocœur

- Nombre de tâches faible, contraintes de temps faibles ;
- Communication inter-tâches sans messagerie OS (variables partagées en mémoire) possible ;
- Tâches ne s'exécutent pas en parallèle ;
- Analyse d'ordonnancabilité *classique* d'un ordonnanceur à priorité fixe (RMA) ;
- Echéances des tâches directement impactées par priorités choisies et temps passé dans les interruptions.

Application Habitacle Multicœur

- Plus de tâches (1 tâche ne peut etre répartie entre 2 OsApplications) ;
- Communication inter-OsApplication par messagerie obligatoire entre tâches localisées sur 2 cœurs ;
- Tâches s'exécutent en parallèle sur les 2 cœurs (gain de performance, accès concurrent mémoire) ;
- Possibilité de *répartir* la charge entre les 2 cœurs de processeurs ;
- *Analyse d'ordonnancabilité très délicate*.

Status

Etat de l'Art

- AUTOSAR défini l'OS et couches basses comme support au multicœur/partitionnement ;
- Les chaînes d'outils actuelles implémentent les OS multicœur AUTOSAR mais rarement le RTE/couches basses associées ;
- Analyse d'ordonnancabilité difficilement maitrisée et argument de Sureté difficile à démontrer dans ces environnements ;
- Besoins croissants des constructeurs pour diminuer le nombre de calculateurs ;
- **Une mauvaise conception en multicœur peut conduire à des performances inférieures à une conception monocœur**.

Travaux futurs

Travaux futurs

- Trampoline en cours de migration vers Multicœur sur architecture PowerPC ;
- Extensions/Études sur autres politiques d'ordonnancement partitionnés et globaux ;
- Analyse temporelles et Vérifications.

Question ?

http://trampoline.rts-software.org
http://www.see4sys.com

Licence LGPL

Cibles supportées par Trampoline

Monocœur : PowerPC, ARM, AVR 8 bits, C166, Posix ;

Multicœur : PowerPC, Posix.