Trampoline
Un système d'exploitation temps réel pour l'informatique embarquée

Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou

IRCCyN – UMR CNRS 6597
CNRS, École Centrale de Nantes, École des Mines de Nantes, Université de Nantes

École d'été Temps Réel
Rennes, 26 août 2015

Section 1

Contexte, définition, architectures

Système embarqué

Système électronique et informatique intégré à un système mécatronique en vue de piloter ses organes

Principales caractéristiques

Ressources matérielles limitées

Matériel adapté aux contraintes opérationnelles (thermiques, vibratoires, etc) et économiques $\longrightarrow$ micro-contrôleur

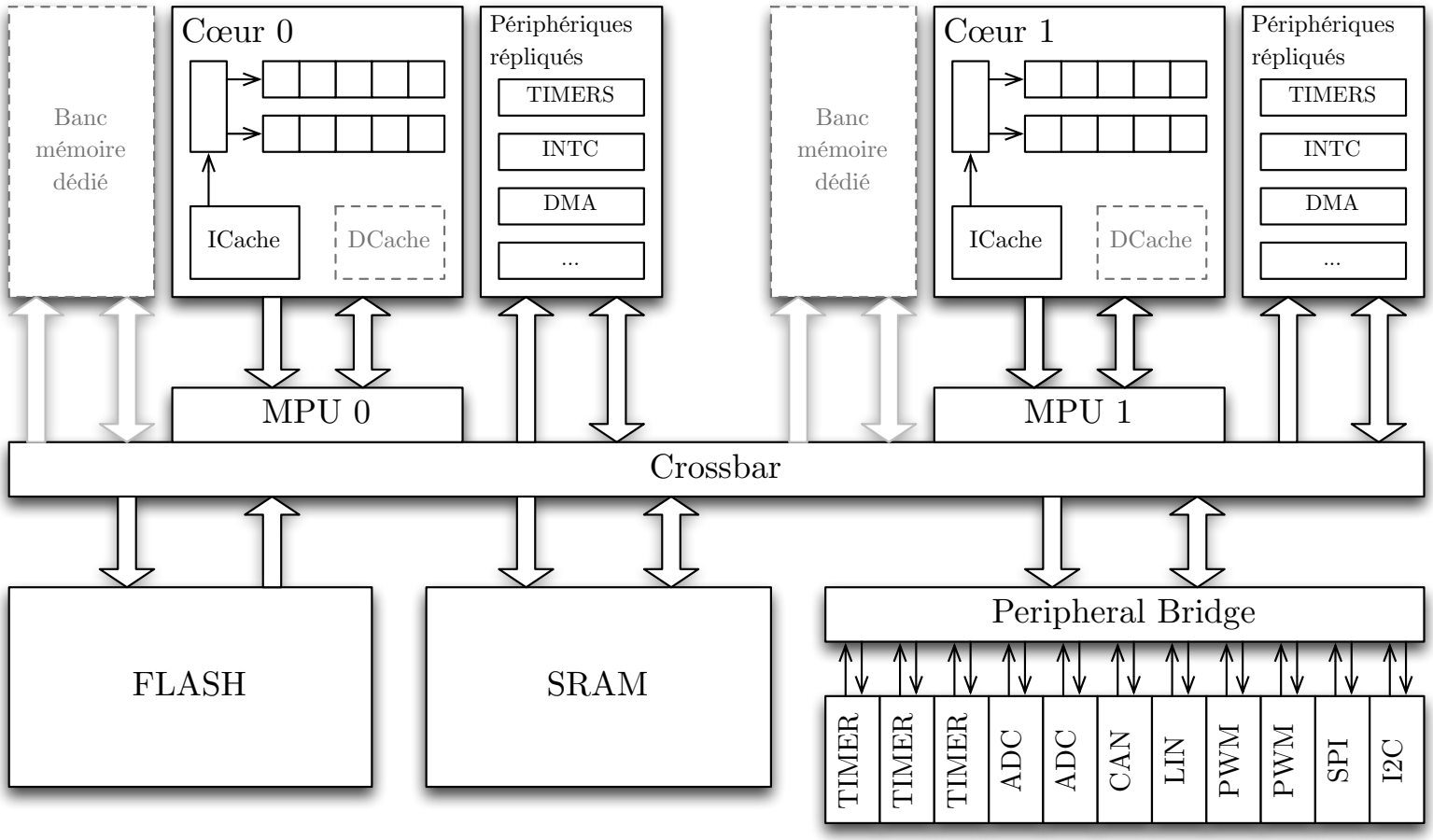
Temps réel

- Contraintes *hard*, *firm*, *soft*
- Dynamique parfois très rapide : les temps de réponse exigés peuvent s'exprimer en centaines de micro-secondes

Contexte industriel

- Normes, standards, certification

Architecture matérielle type : organisation



Architecture matérielle type : caractéristiques

- Mono ou Multicœur
- ISA de type RISC 32 bits
- Micro-architecture superscalaire
- Cache : L1 uniquement, souvent I-Cache uniquement
- Protection mémoire : MPU, plus rarement MMU
- Pas d'assistance matérielle à la virtualisation

Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Fonctions

- Partie commande : régulateurs, filtres, machine à états
- Partie système : pilotes, pile de communication, monitoring, ...

Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Temps réel

- Activations périodiques ou sporadiques
- Échéances contraintes ou sur requêtes
- Criticités multiples

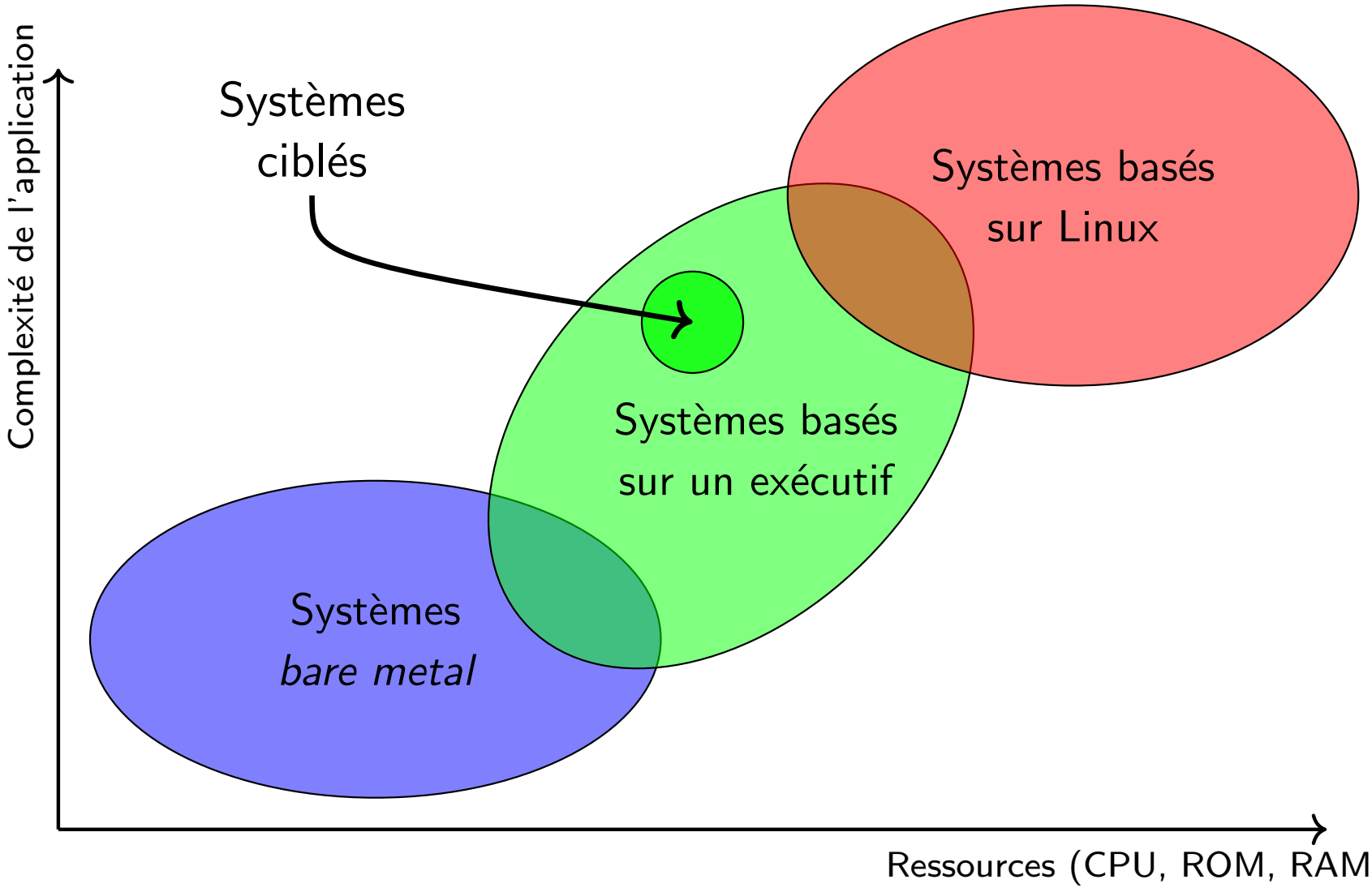
Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Communication (et synchronisation)

- Signaux booléens ou numériques, scalaires ou vectoriels
- Utilisation de queues de messages et/ou de blackboard
- Ressources partagées (périphériques)

Positionnement



Système d'exploitation temps réel

Système d'exploitation temps réel (SETR)

Un système d'exploitation (SE) est qualifié de temps réel s'il permet de construire un système aux temps de réponses **prédictibles** et **compatibles avec les dynamiques de l'environnement.**

Chemin critique

- Prise en compte des interruptions
- Ordonnancement
- Changement de contexte

Impact sur la conception

Déterminisme

- Restriction du nombre de services en mode noyau

Prédictibilité

- Éviction des algorithmes / services dont le pire cas est pathologique
- Minimiser le pire-cas, éventuellement au détriment d'autres objectifs : équité, flexibilité, sécurité, etc.

Les services de base

Temps réel

- Ordonnancement
- IPC : synchronisation, signalisation, communication
- Prise en compte des interruptions
- (Mesure du temps : horloges, réveils, chien de garde)

Tolérance aux fautes

- Isolation mémoire
- Isolation temporelle

Et le reste ?

Reléguée hors du noyau

- Piles de communication
- Pilotes de périphériques

Parfois/souvent/toujours inutiles

- Système de gestion de fichiers
- Gestion des utilisateurs

Difficilement temps réel

- Mémoire virtuelle (*swapping*)

Architecture d'un système d'exploitation temps réel

Comme pour les GPOS, différentes architectures sont envisageables, en fonction des contraintes du domaine applicatif

- Exécutif
- Separation Kernel
- Micro-noyau
- Hyperviseur
- etc.

Exécutif

Définition

Un exécutif est un ≪ petit ≫ SETR, adapté aux ressources limitées des micro-contrôleurs, et incluant un nombre minimal de services

Relation avec les SETR

Exécutif $\subsetneq$ SETR

Exemples

FreeRTOS, Trampoline

Separation Kernel

Separation Kernel, Partitioning Kernel

≪ *The task of a separation kernel is to create an environment which is indistinguishable from that provided by a physically distributed system* ≫ (Rushby, 1981).
Le concept de *separation kernel* a été développé pour des objectifs de sécurité. Le concept de *partiotioning kernel* en est l'adaptation au temps réel

Relation avec les SETR

Un partiotioning kernel est un SETR dédié aux systèmes critiques

Exemples

Integrity-178B, standard ARINC 653

Micro-noyau

Micro-noyau

≪ *A concept is tolerated in the μ-kernel only if moving it outside the kernel [...] would prevent the implementation of the system's required functionality* ≫ (Liedtke, 1995)

- Manipulations basique de l'espace d'adressage
- Identification et manipulations des threads, IPC
- (aujourd'hui unifié sous la forme d'un gestionnaire de capacités)

Relation avec les SETR

Micro-noyau $\neq$ SETR $\wedge$ Micro-noyau $\cap$ SETR $\neq \emptyset$

Exemples (micro-noyaux temps réel)

L4/Fiasco, Composite

Hyperviseur

Hyperviseur (pour l'informatique embarquée)

Hyperviseur de type I (*bare metal*) orienté prédictibilité et sécurité.
Technologiquement similaire aux micro-noyaux

Relation avec les SETR

Un hyperviseur temps réel a les caractéristiques d'un SETR

Exemples

PikeOS, Fiasco.OC, OKL4 Microvisor

Section 2

Trampoline : un exécutif temps réel

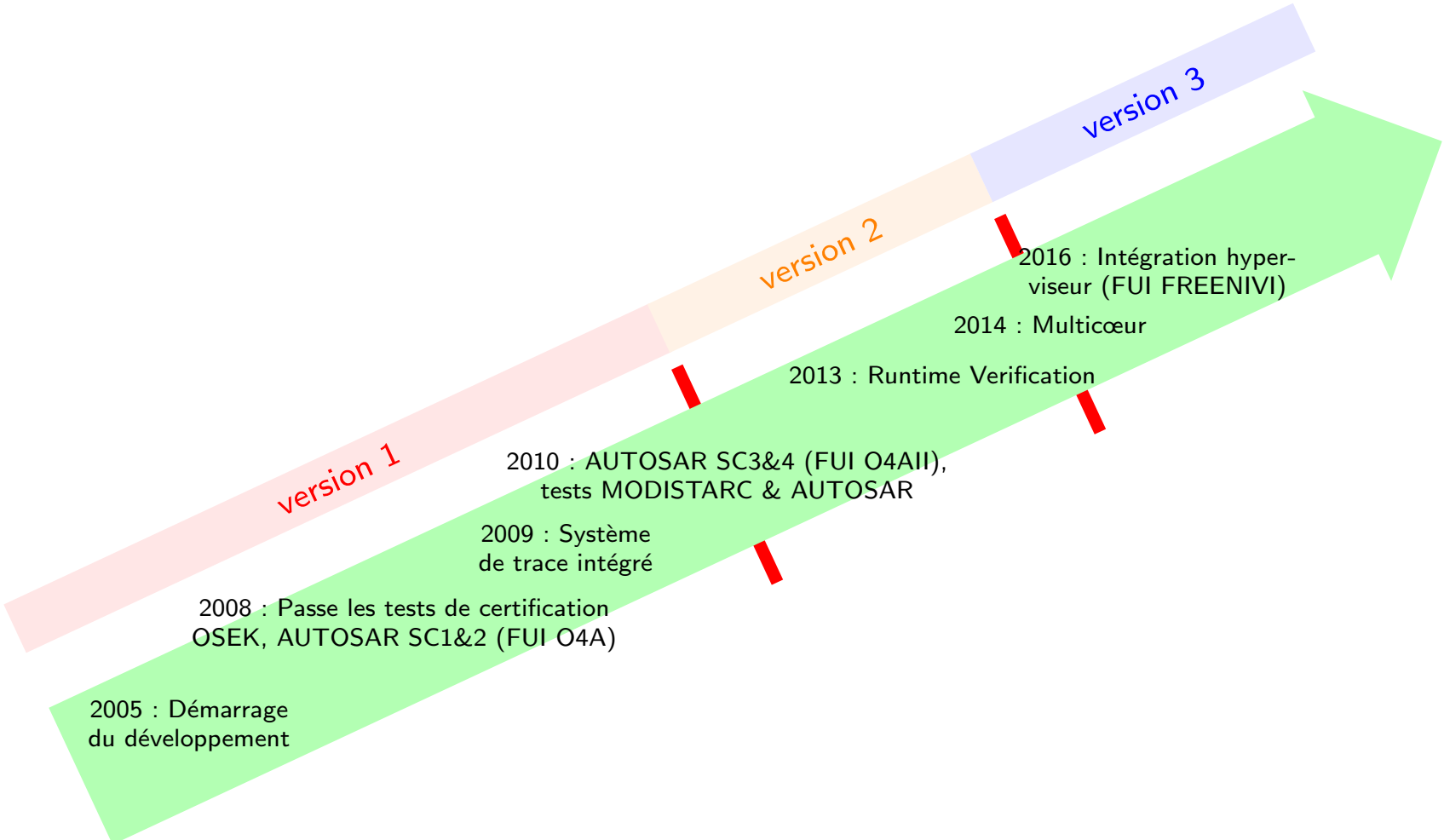
Trampoline : un exécutif temps réel

- Présentation
- Processus de construction d'une application
- Architecture interne
- Travaux autour de Trampoline

Trampoline

- Exécutif temps réel, développé par l'équipe Systèmes Temps Réel de l'IRCCyN
- Modulaire et portable
- Licence libre (GPL) + version industrialisée
- Conforme OSEK/VDX OS et AUTOSAR OS
- Nombreuses cibles dont Arduino, Posix
- Support expérimental des architectures multicœurs

Historique


version 1
version 2
version 3
2005 : Démarrage du développement
2008 : Passe les tests de certification OSEK, AUTOSAR SC1&2 (FUI O4A)
2009 : Système de trace intégré
2010 : AUTOSAR SC3&4 (FUI O4AII), tests MODISTARC & AUTOSAR
2013 : Runtime Verification
2014 : Multicœur
2016 : Intégration hyperviseur (FUI FREENIVI)

Utiliser Trampoline : pourquoi ? comment ?

Pourquoi ?

Disposer d'un composant libre pour construire des bancs d'essais

Support d'enseignement

Expérimenter des nouveaux algorithmes / services

Comment ?

Site web : `http://trampoline.rts-software.org`

Dépôt : `http://trampoline.rts-software.org/svn/trunk`

- susceptible de changer

Système statique

Un système d'exploitation est statique s'il n'offre pas la possibilité de créer des objets (tâche, ressource, boîte aux lettres, etc.) en ligne. Il doit donc être entièrement configuré à la compilation.

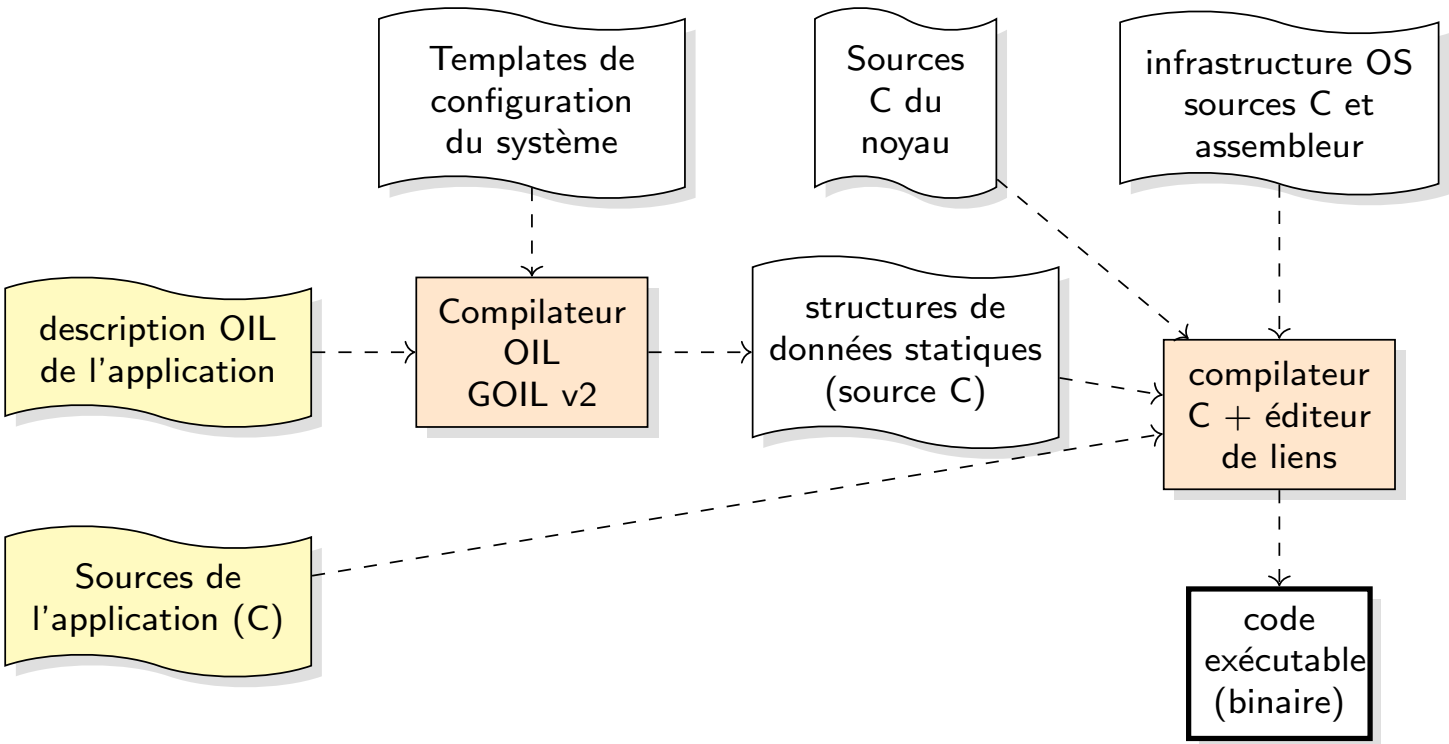
Le noyau d'un système statique peut être « taillé sur mesure » pour l'application

- Services à inclure
- Structure de données
- Et plus encore : chemins d'exécution, variables, etc.

Trampoline est un système statique

Trampoline utilise un langage dédié

- dérivé de la norme OIL
- extensible (basé sur un moteur de template)



Exemple de description

```
ALARM one_second {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK { TASK = t0; };
  AUTOSTART = TRUE { APPMODE = std; ALARMTIME = 100; CYCLETIME = 100; };
};

TASK t0 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  MESSAGE = s00;
};

MESSAGE s00 {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
      CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};
```

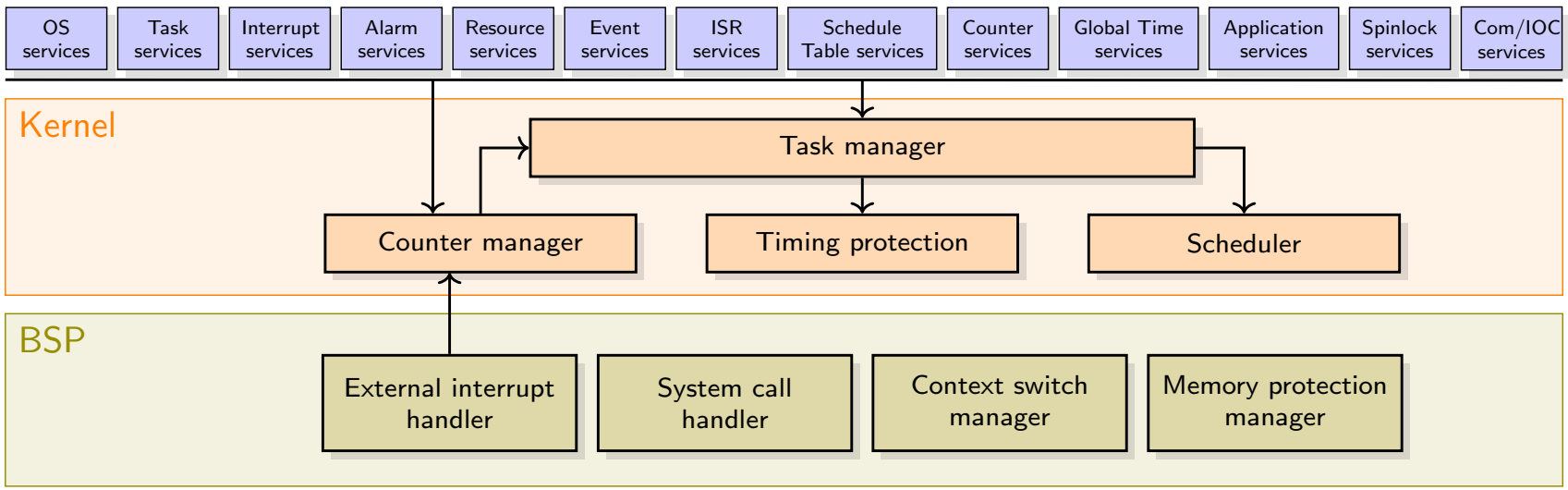
Exemple de template goil

```
/*============================================================================
 * Definition and initialization of process tables (tasks and isrs)
 */
CONSTP2CONST(tpl_proc_static, AUTOMATIC, OS_APPL_DATA)
tpl_stat_proc_table[TASK_COUNT+ISR_COUNT+% ! OS::NUMBER_OF_CORES %] = {
%
foreach proc in PROCESSES do
    %  &% !proc::NAME %_% ![proc::KIND lowercaseString] %_stat_desc,
%
end foreach
if OS::NUMBER_OF_CORES == 1 then
%  &IDLE_TASK_task_stat_desc%
else
  loop core from 0 to OS::NUMBER_OF_CORES - 1 do
    %  IDLE_TASK_% ! core %_task_stat_desc%
  between %,
%
  end loop
end if
%
};
```

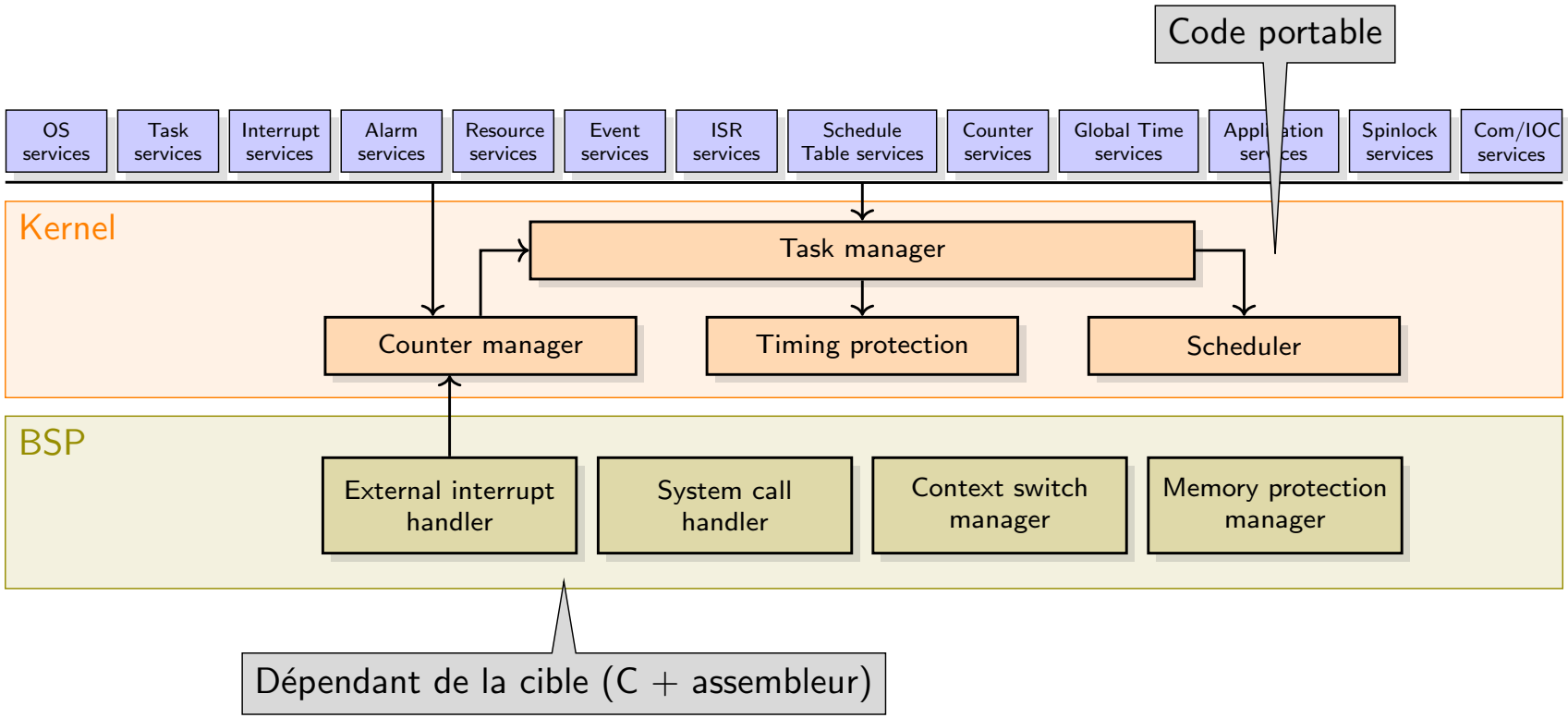
Avantages de `goil` sur une API

- Séparation des préoccupations
- Niveau d'abstraction
- Plus simple à faire évoluer
- Automatisation de la spécialisation du noyau
- Génération automatique : Code, Makefile, *link script*

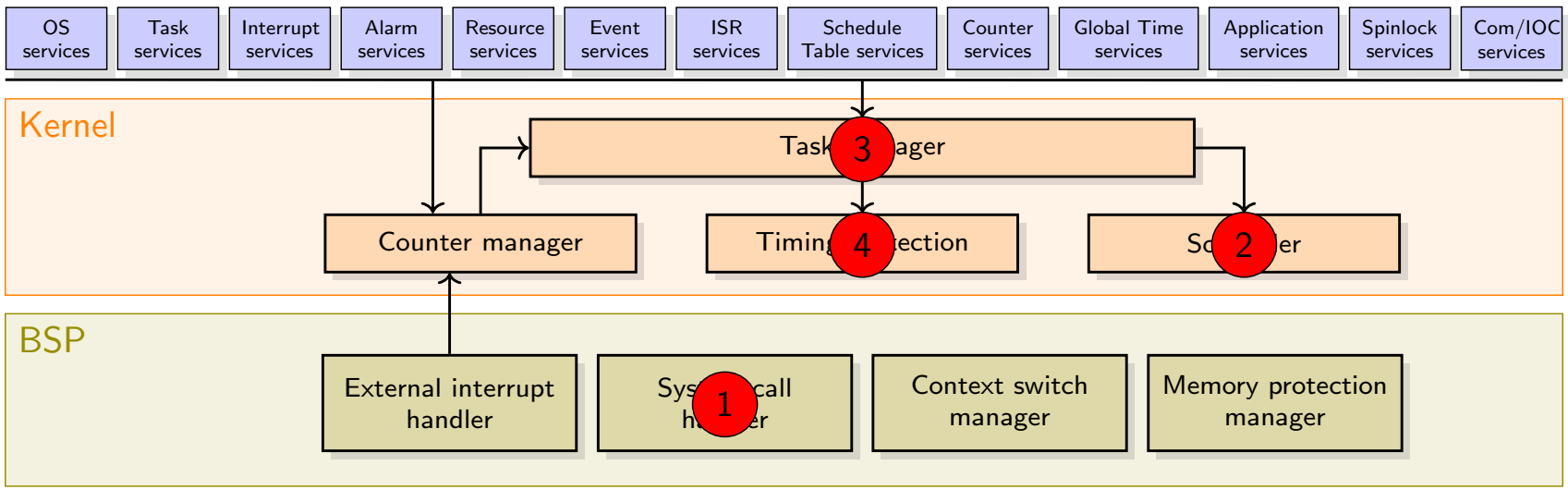
Architecture



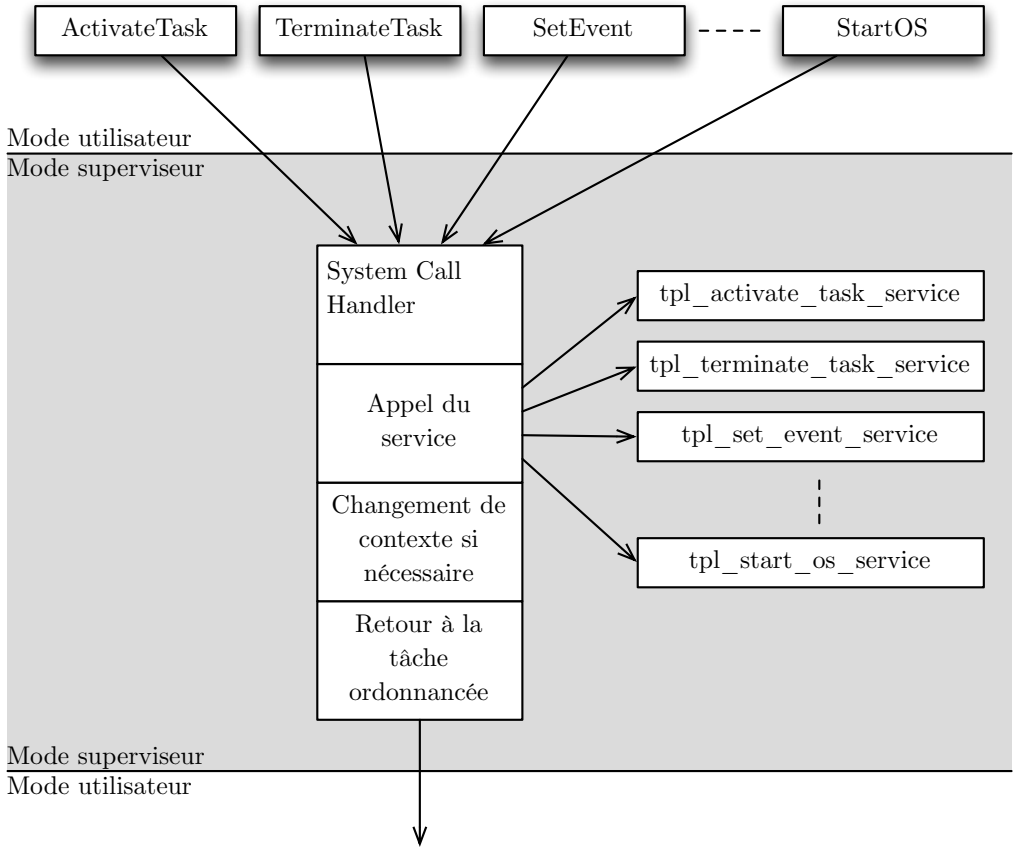
Architecture



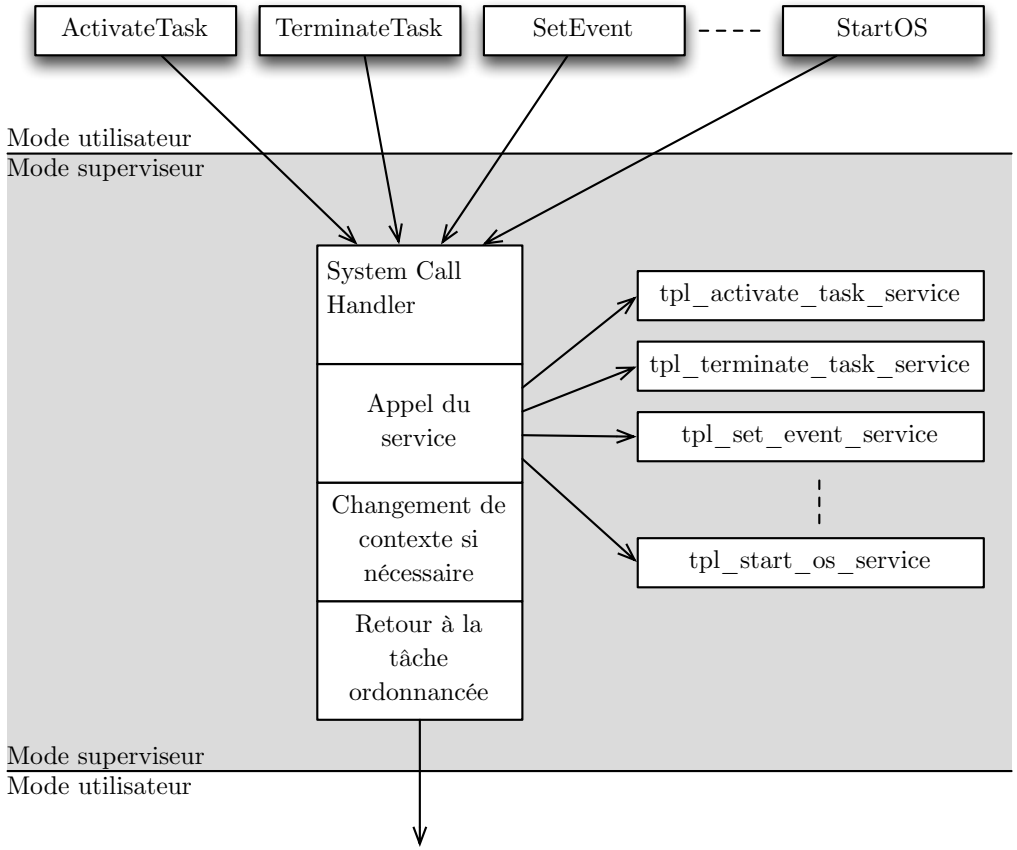
Architecture



Module *System Call Handler*



Module *System Call Handler*



Une fois passé en mode superviseur, le noyau est non préemptible. Il faut donc minimiser la durée de ce passage pour assurer la réactivité du système

Module *System Call Handler*

Passage en mode superviseur (exemple sur PowerPC)

```
  .global SuspendOSInterrupts
SuspendOSInterrupts:
  li   r0,OSServiceId_SuspendOSInterrupts  /* service id in r0 */
  sc                                       /* system call      */
  blr                                      /* returns          */
```

Selon l'ABI utilisée, les paramètres et le code de retour de l'appel système sont passés dans des registres ou sur la pile

Module *Scheduler*

Scheduler

Assure la manipulation de la liste des tâches prêtes grâce à quatre fonctions

- `tpl_put_new_proc`, `tpl_put_preempted_proc`, `tpl_front_proc`, `tpl_remove_front_proc`

Implémentation

Dans la v3 de Trampoline, la structure utilisée est un tas binaire

- Insertion et suppression en $\log_2(n)$,
- Consultation de la tête en temps constant

Module *Scheduler*

Politique d'ordonnancement

FP/FIFO avec seuil de préemption.

En pratique, toute politique *fixed job priority* qui assure un ordre *FIFO* entre les *jobs* successifs d'une même tâche peut être implémentée

Modifier la politique d'ordonnancement

1. Ajouter les informations nécessaires aux descripteurs de tâches à l'aide des *templates* `goil`
2. Adapter le calcul de la priorité dans `tpl_put_new_proc`
3. Adapter le protocole de synchronisation pour l'accès aux ressources partagées (IPCP par défaut)

Module *Task manager*

Task Manager

Fournit les fonctions permettant de faire évoluer l'état des ≪ tâches ≫.

Utilise les fonctions du module *Scheduler* pour manipuler les *jobs* engendrés.

Tâche ?

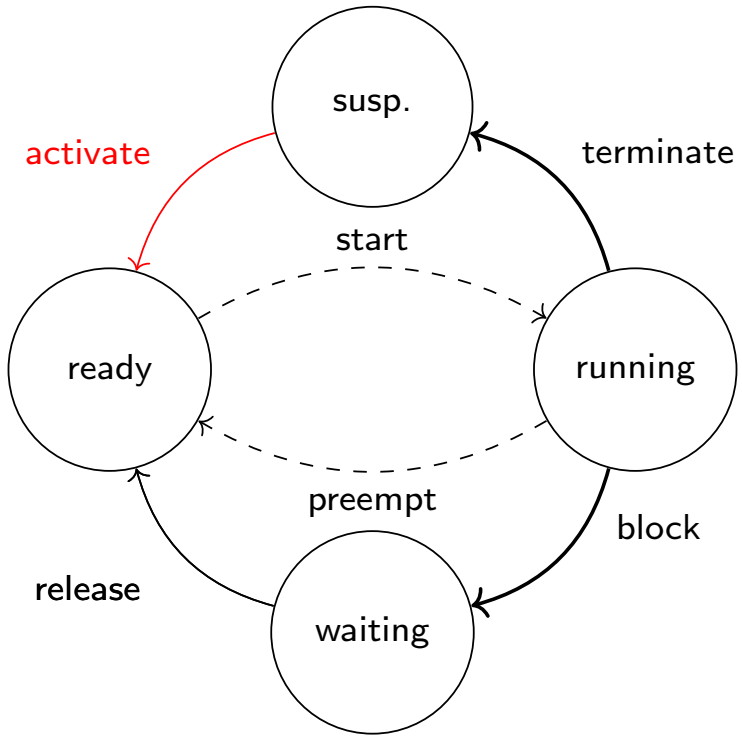
L'API propose 3 types de tâches

- tâche basique : pas d'appel bloquant
- tâche étendue
- ISR : similaire aux tâches basiques mais activée sur une IRQ

En interne, un seul type est utilisé : `tpl_proc`

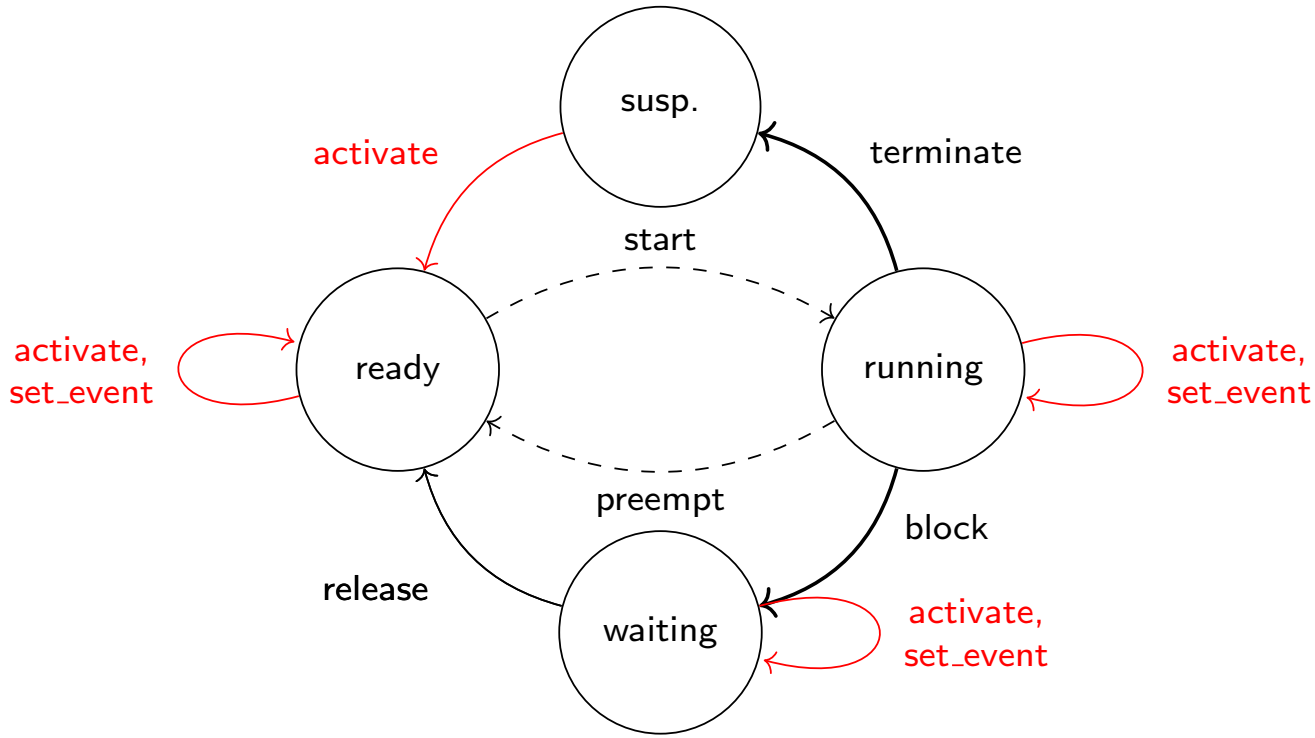
Module *Task Manager*

Le module *Task Manager* fait évoluer l'état des objets `tpl_proc`



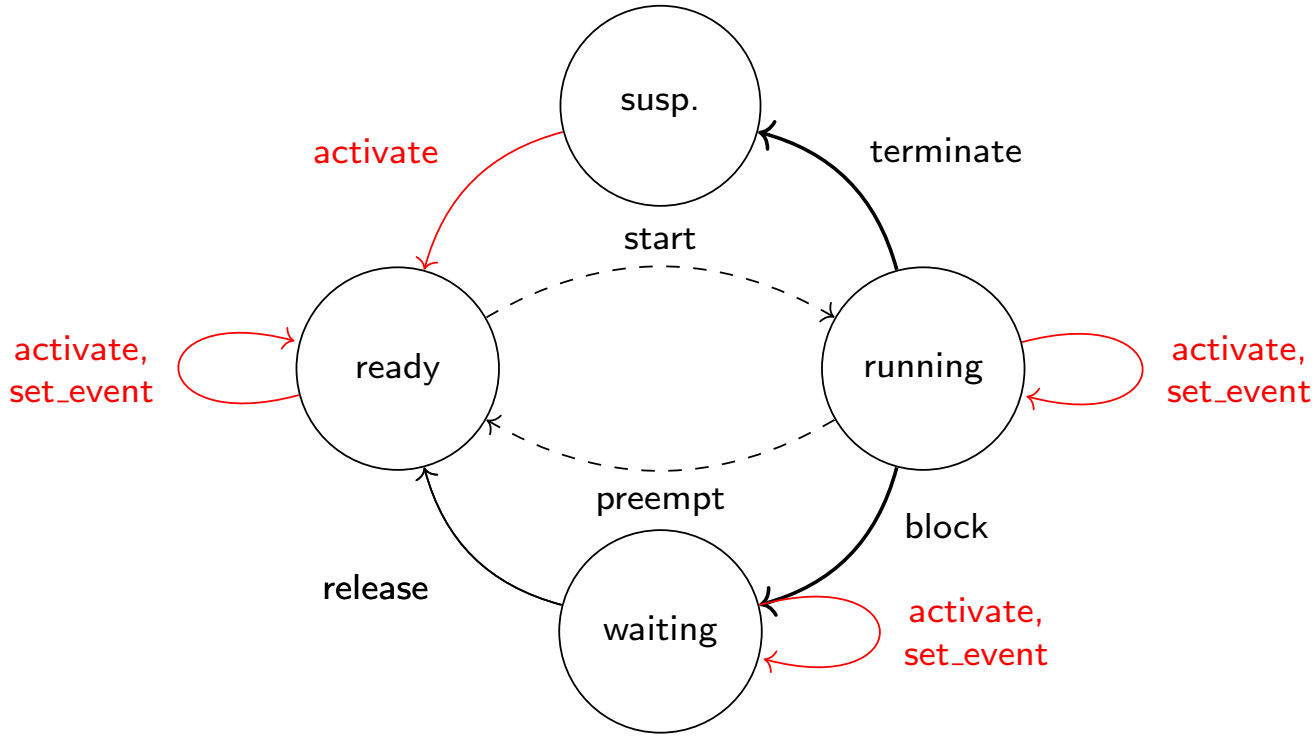
Module *Task Manager*

Le module *Task Manager* fait évoluer l'état des objets `tpl_proc`



Module *Task Manager*

Le module *Task Manager* fait évoluer l'état des objets `tpl_proc`



- Enregistrement des activations jusqu'à une borne fixée
- Enregistrement des événements dans un vecteur de bit

Module *Task Manager*

L'API du module se déduit du modèle

- `tpl_activate_task`
- `tpl_preempt`
- `tpl_start`
- `tpl_terminate`
- `tpl_block`
- `tpl_release`
- `tpl_set_event`

Module *Timing Protection*

Timing protection

Met en œuvre une forme d'isolation temporelle en assurant le respect de trois invariants

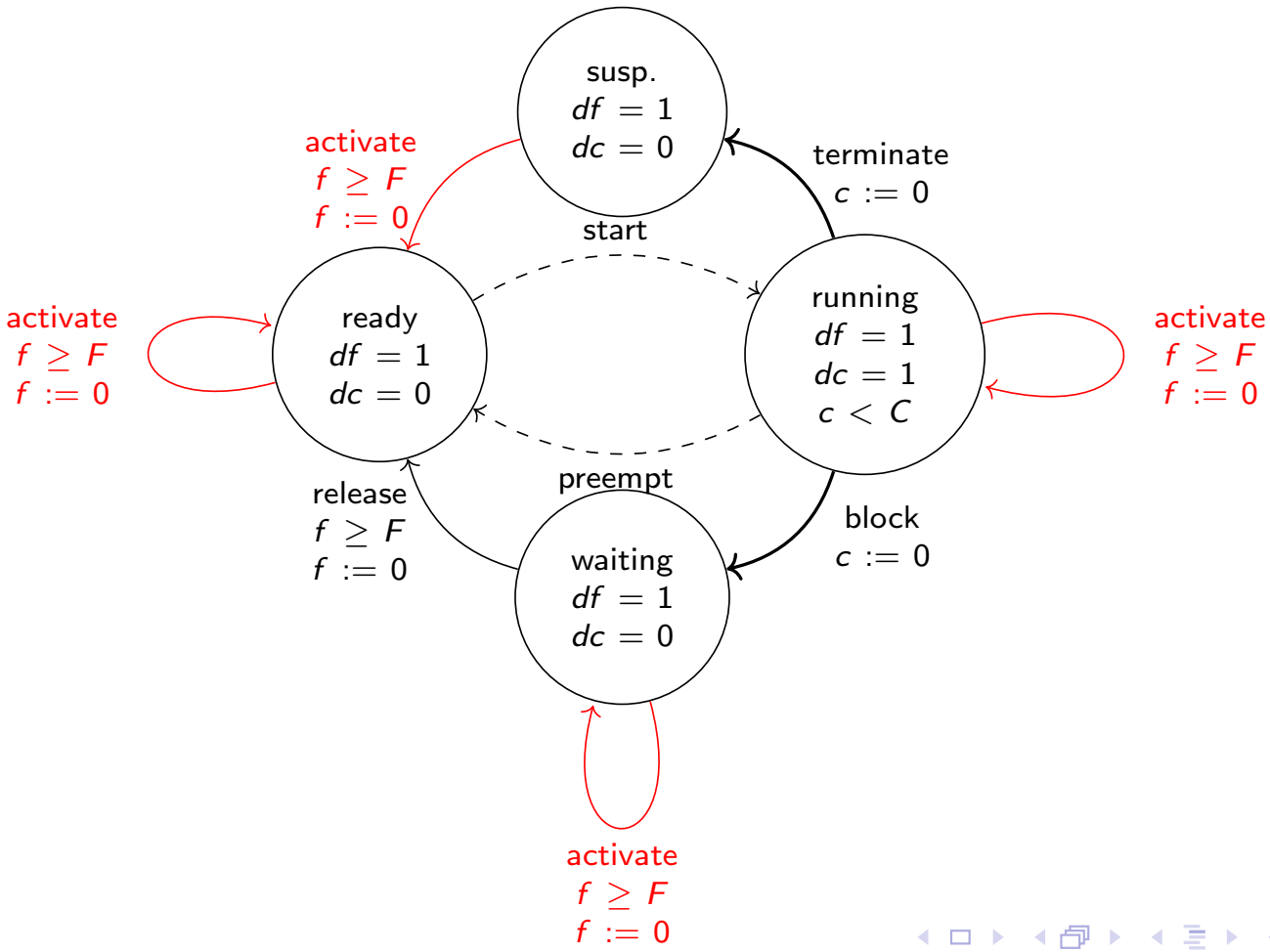
- le temps d'exécution des *jobs* d'une même tâche est inférieur ou égal à un budget C
- le temps d'exécution de chaque section critique des *jobs* d'une même tâche est inférieur ou égal à un budget B
- le délai entre la création de deux *jobs* successifs par une même tâche est supérieur ou égal à une borne F

Ces trois invariants renvoient aux attributs du modèle de tâche sporadique

Module *Timing Protection*

L'horloge c mesure le temps d'exécution du *job* $\ll$ actif $\gg$ de la tâche

L'horloge f mesure le temps écoulé depuis la dernière activation d'un *job* de la tâche



Module *Timing Protection*

Recouvrement d'erreur

Appel d'une fonction utilisateur qui choisit :

- Tuer la tâche
- Tuer l'OS-Application
- Re-démarrer l'OS-Application
- Éteindre le système

Travaux passés et en cours dans l'équipe

Thèse de Dominique Bertrand

- Dimensionnement des paramètres du module de protection temporelle

Thèse de Sylvain Cotard

- Développement d'un service de vérification en ligne
- Développement d'un service de communication *wait free*

Thèse de Toussaint Tigori

- Génération de noyau spécialisé pour l'application

Thèse de Louis-Marie Givel

- Test de propriétés temps réel par injection de délais

Travaux passés et en cours dans l'équipe

Thèse de Dominique Bertrand

- Dimensionnement des paramètres du module de protection temporelle

Thèse de Sylvain Cotard

- Développement d'un service de vérification en ligne
- Développement d'un service de communication *wait free*

Thèse de Toussaint Tigori

- Génération de noyau spécialisé pour l'application

Thèse de Louis-Marie Givel

- Test de propriétés temps réel par injection de délais

Un service de vérification en ligne pour AUTOSAR

Mécanismes de protection AUTOSAR OS

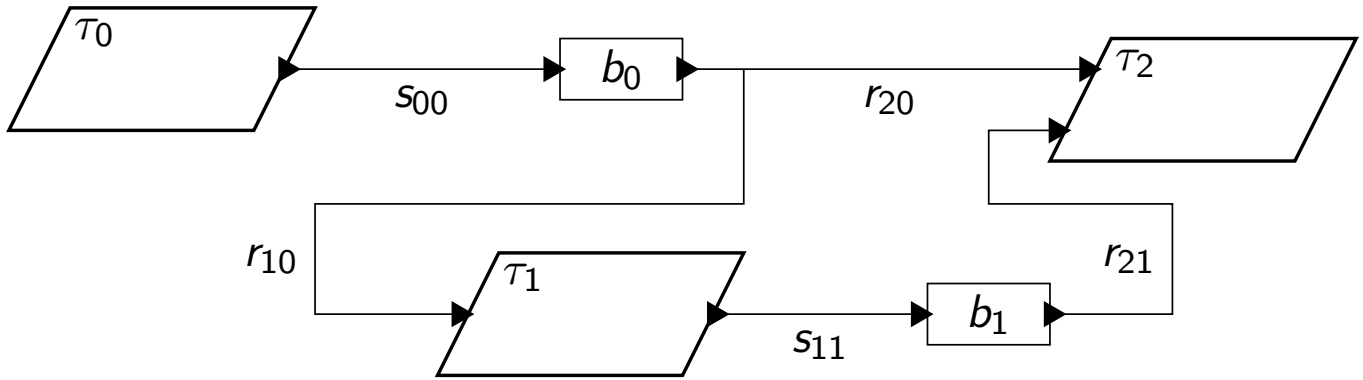
AUTOSAR OS ajoute à OSEK/VDX OS un ensemble de mécanismes de protection

- Protection mémoire
- Protection temporelle
- Protection du noyau (services et ressources matérielles)

Nos travaux

- Protection fonctionelle : propriétés chronologiques inter-tâches
- Implantation interne au noyau

Exemple

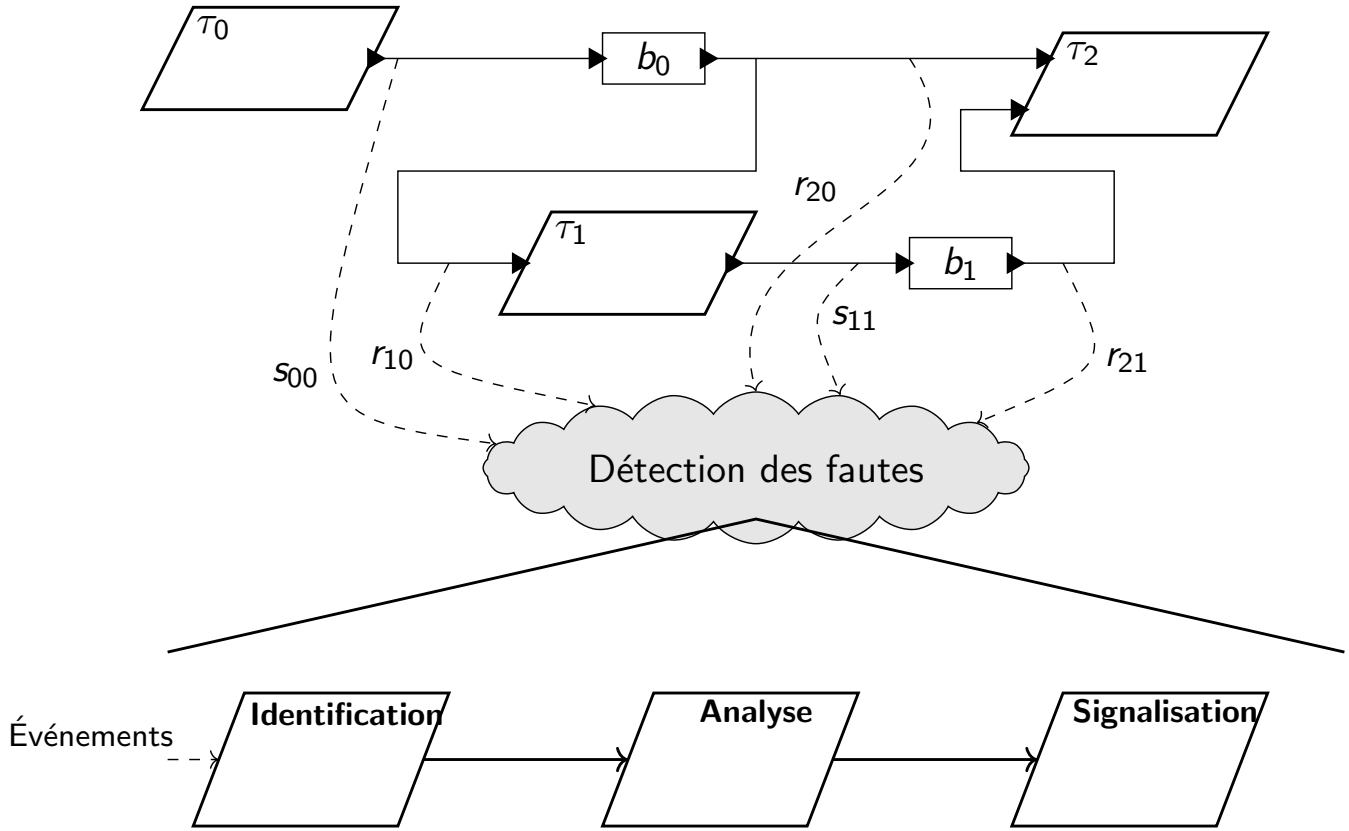


τ_1, τ_2 et τ_3 sont trois tâches, b_0 et b_1 sont deux buffers de type blackboard. τ_2 effectue un test de cohérence entre les entrées (lues dans b_0) et les sorties (écrites dans b_1) de τ_1

Propriété à vérifier : cohérence des données

Quand une instance de τ_2 lit une donnée, les buffers sont synchronisés et le restent jusqu'à la terminaison de cette instance.

Objectif


τ_0
b_0
τ_2
r_{20}
τ_1
b_1
s_{11}
r_{10}
s_{00}
r_{21}
Détection des fautes
Événements
Identification
Analyse
Signalisation

Who watches the watchmen ?

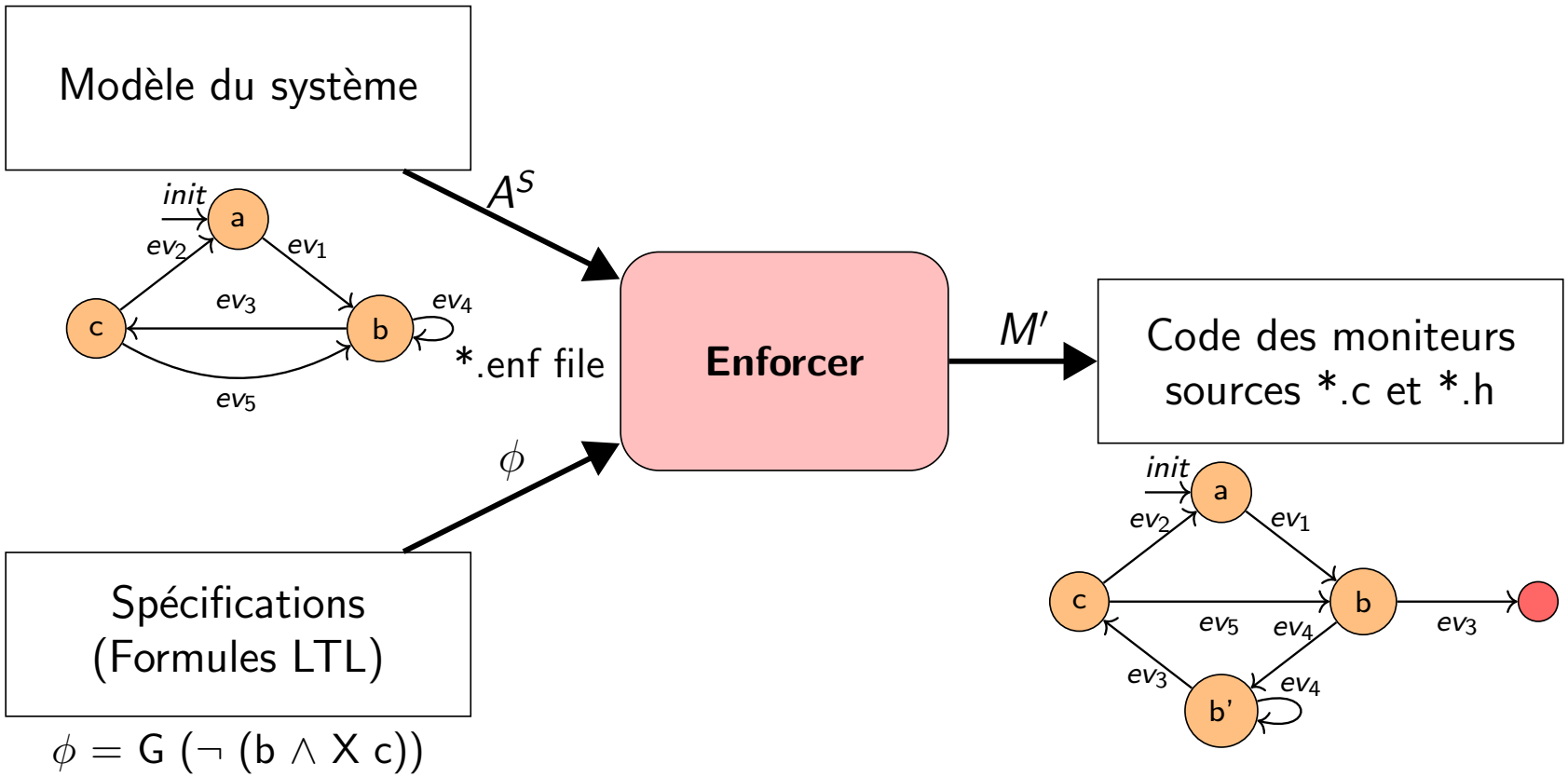
Exigences

- Mécanisme de détection
  - au moins aussi critique que le système surveillé
- Interne au noyau :
  - Limiter l'impact sur le chemin critique
  - Noyau au moins aussi critique que le composant le plus critique

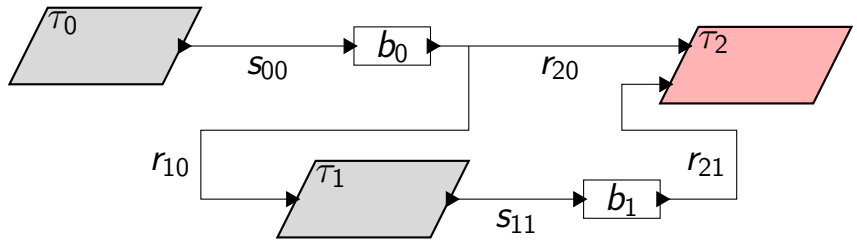
Vérification en ligne

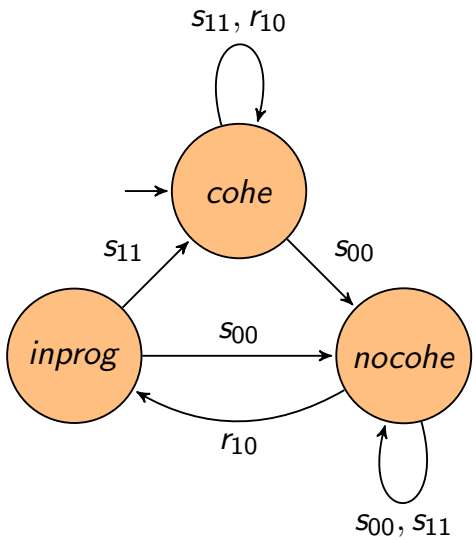
Domaine des méthodes formelles qui porte sur la construction automatisée et prouvée de moniteurs à partir de spécifications formelles.

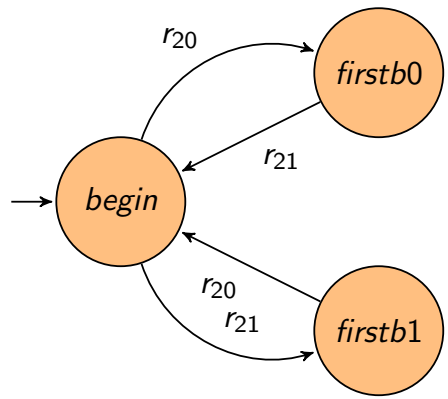
Enforcer : un outil pour la génération de moniteurs



Exemple : entrées



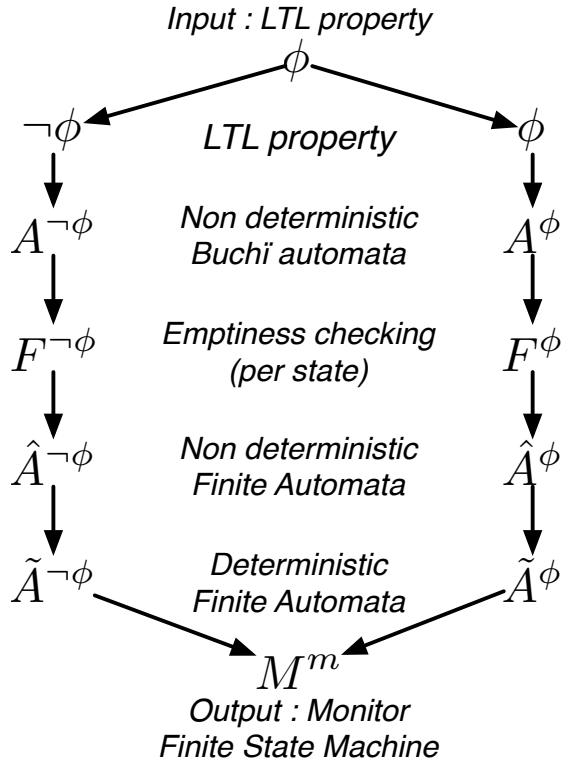




$$\phi = \mathbf{G}\ ((m\_t2.firstb0 \vee m\_t2.firstb1) \implies (m\_buf.cohe\ \mathbf{U}\ m\_t2.begin))$$

Aperçu de la technique de calcul des moniteurs

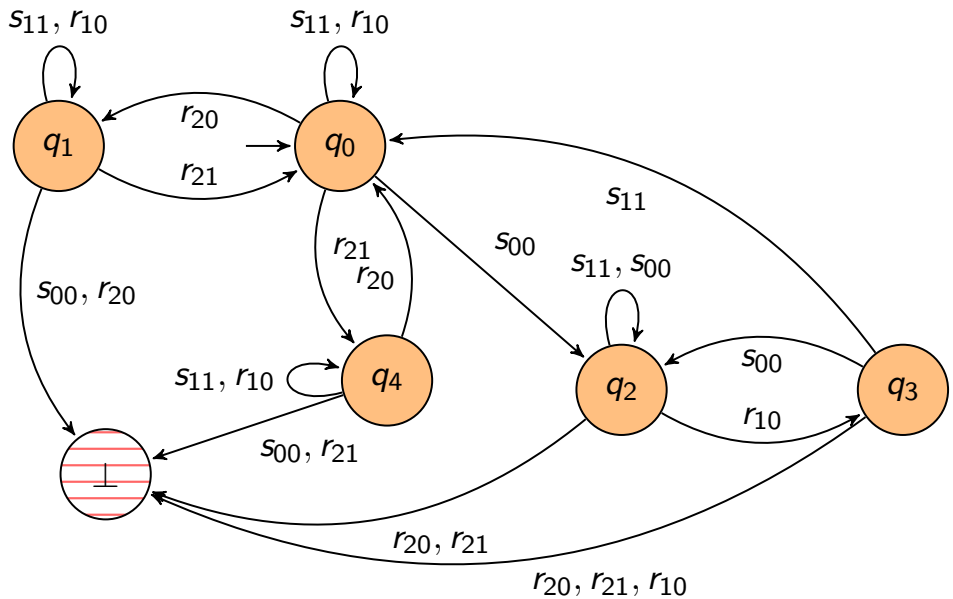
Étape n° 1 – Calculer un moniteur (machine de Moore) à partir des spécifications (Bauer *et al.*, 2011)



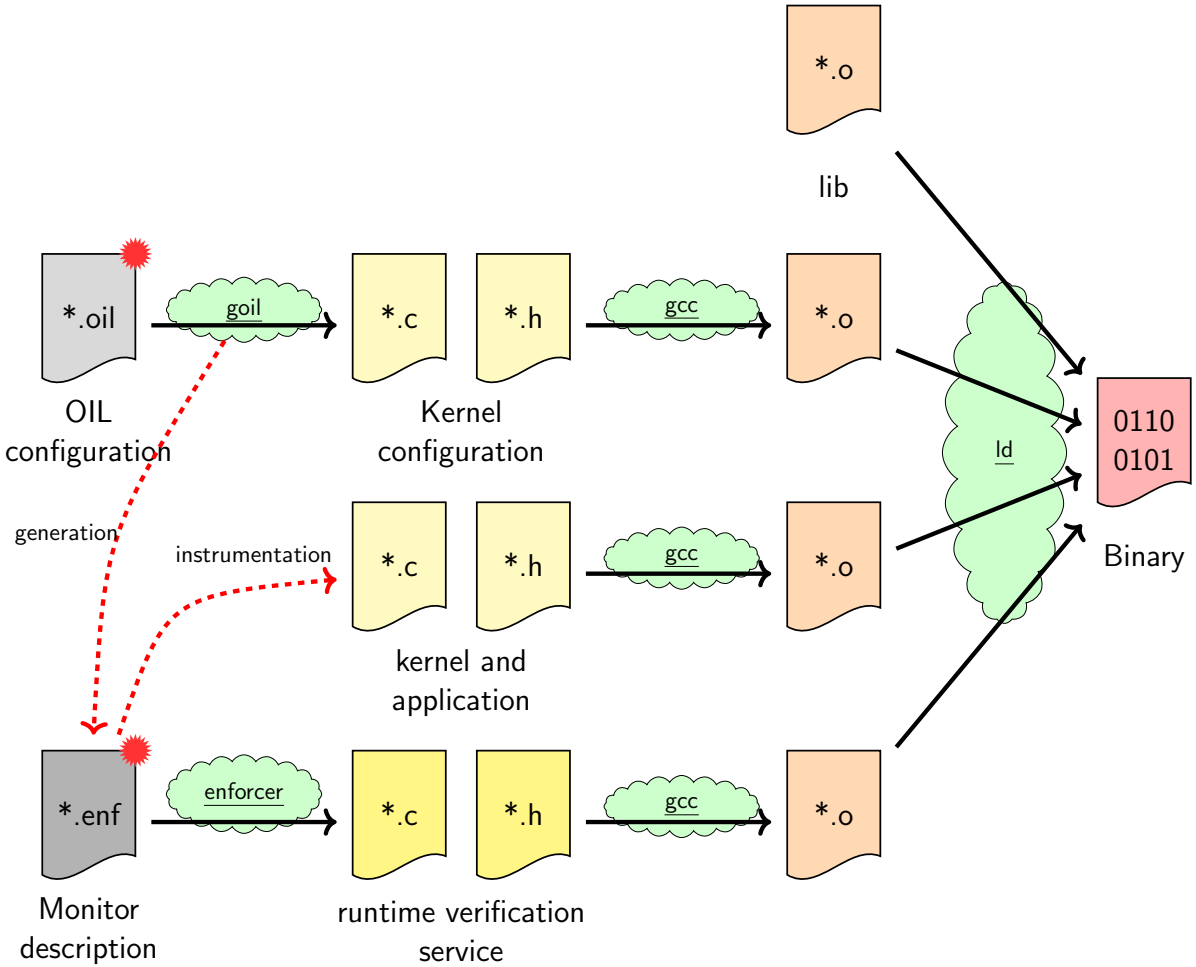
1. Calcul du NBA qui reconnait les traces (infinies) valides
2. Test du langage vide pour chaque état
3. Calcul du NFA qui reconnait les préfixes (finis) des traces valides
4. Calcul d'un DFA par déterminisation
5. Calcul d'une machine de Moore par composition des DFA et calcul de la fonction de sortie

Étape n° 2 – Composer le résultat avec un modèle du système pour faciliter l'identification des événements

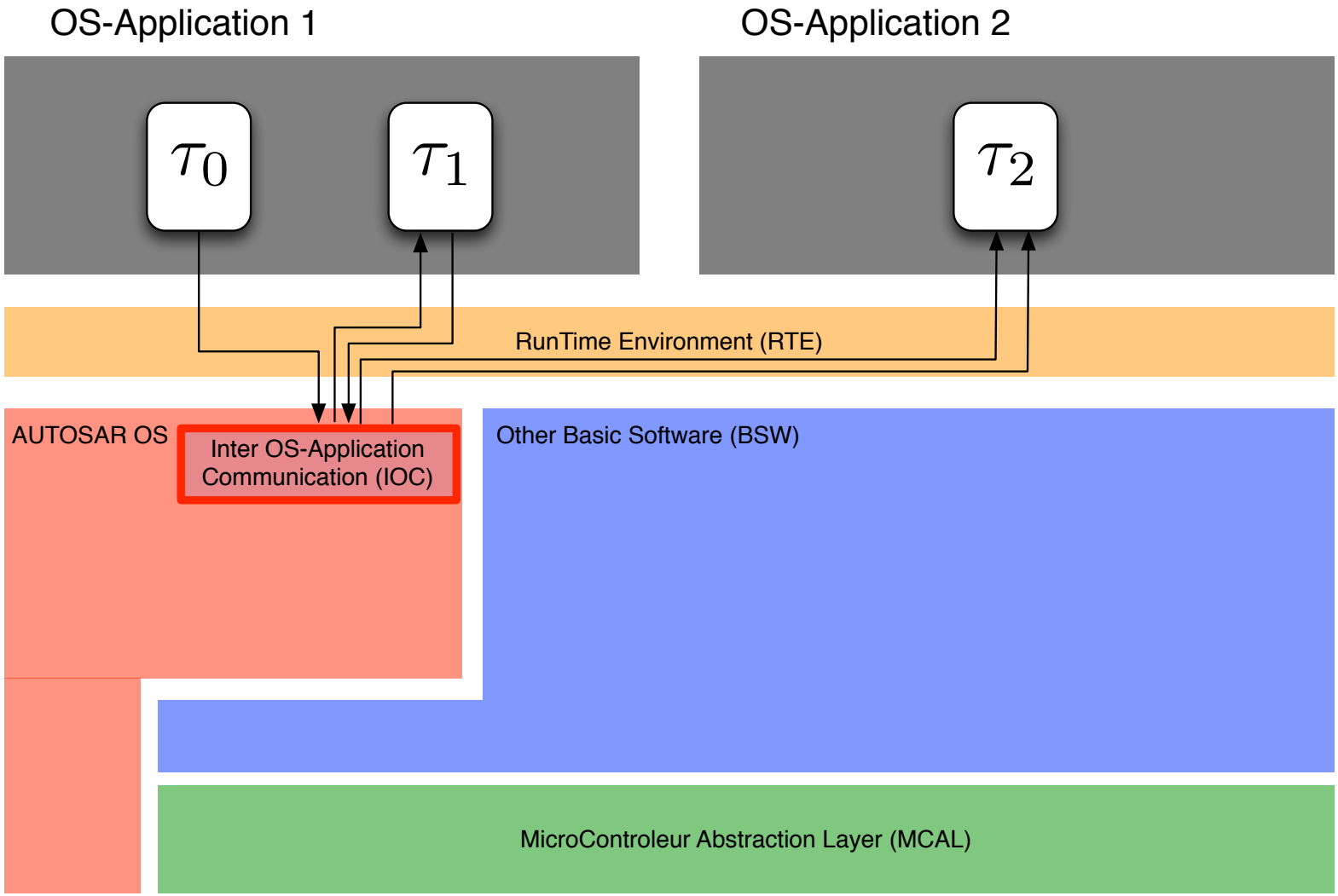
Exemple : résultat



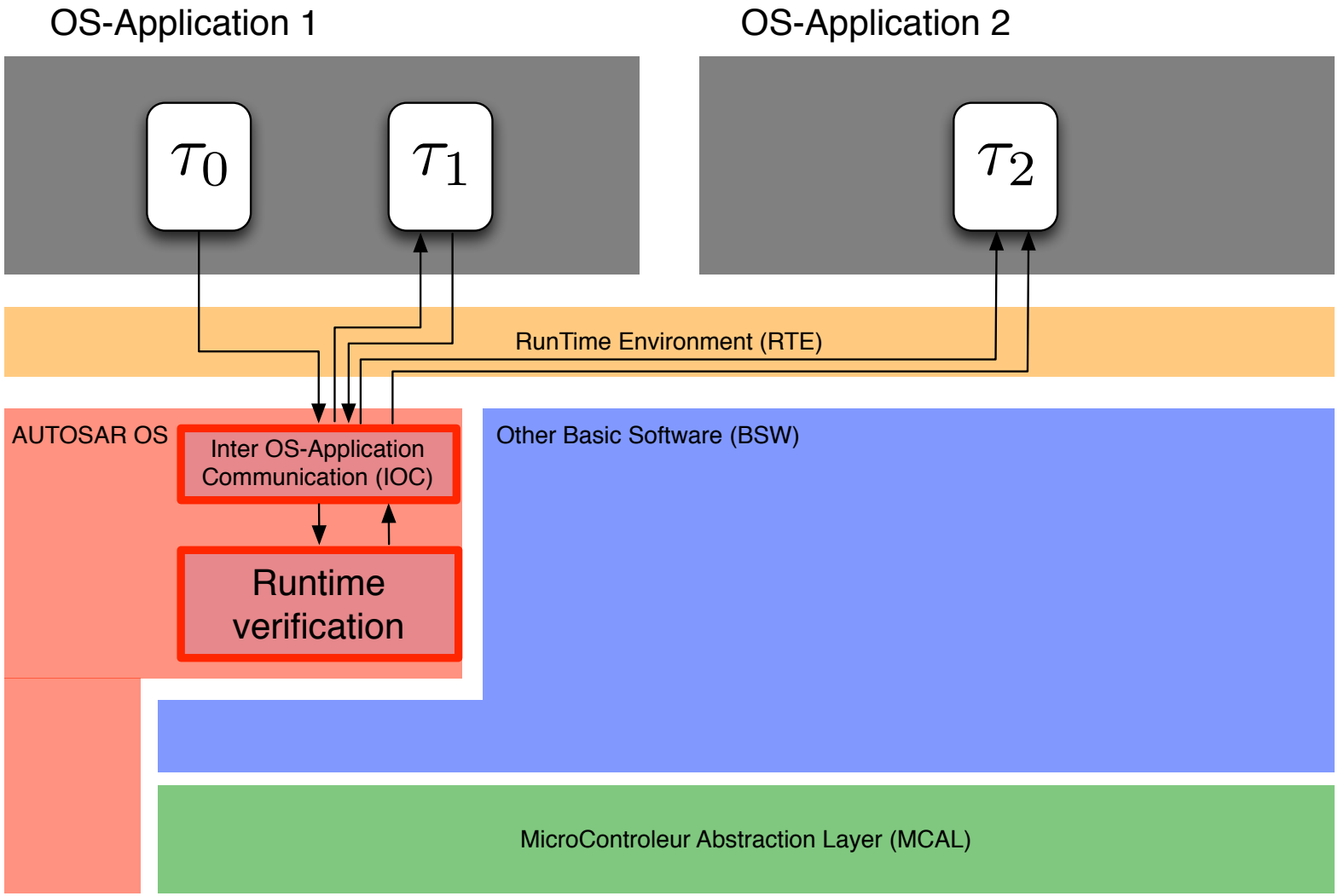
Intégration dans la chaîne d'outils pour Trampoline



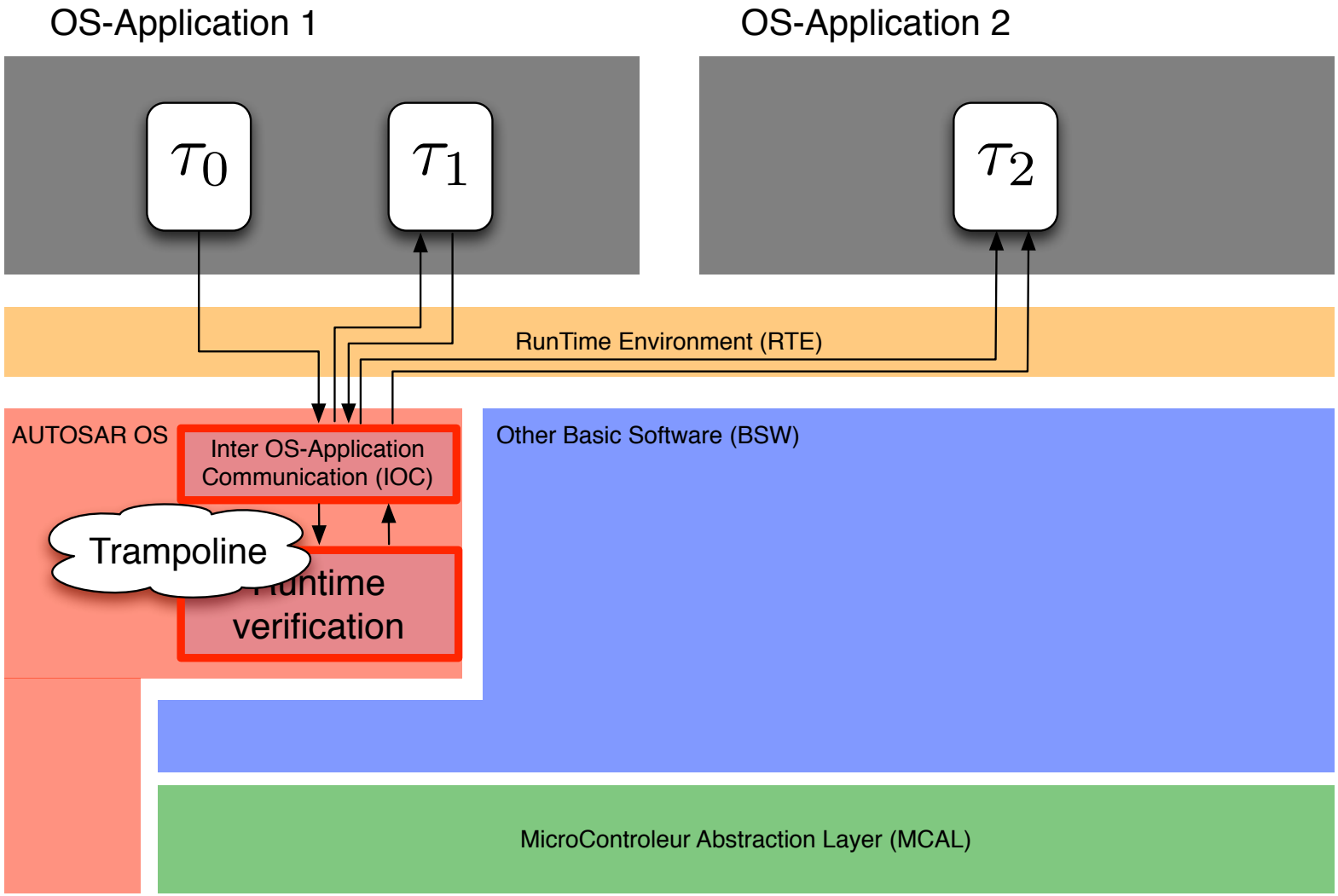
Injection dans Trampoline



Injection dans Trampoline



Injection dans Trampoline



Instrumentation du noyau

```
FUNC(tpl_status, OS_CODE) tpl_receive_static_internal_unqueued_message(
    CONSTP2CONST(void, AUTOMATIC, OS_CONST)       rmo,
    P2CONST(tpl_com_data, AUTOMATIC, OS_VAR) data)
{
    VAR(tpl_status, AUTOMATIC) result = E_COM_FILTEREDOUT;

    /*  cast the base receiving mo to the correct type of mo                */
    CONSTP2CONST(tpl_internal_receiving_unqueued_mo, AUTOMATIC, OS_CONST)
    rum = rmo;
    /*  get the destination buffer                                          */
    P2VAR(tpl_com_data, AUTOMATIC, OS_VAR)
    mo_buf = rum->buffer.buffer;

    #if WITH_MONITOR == YES
    #if MONITORING_SENDING_MESSAGE == YES
    sending_event_table[rum->base_mo.monitor.buffer_id][tpl_kern.running_id]();
    #endif
    #endif

    /*  reception filtering                                                 */
    if (tpl_filtering(mo_buf, data, rum->base_mo.filter))
    {
        /*  get the size of the buffer
*/
        VAR(int, AUTOMATIC) size = rum->buffer.size;
        result =  E_OK;
        /*  copy the data from the source (data)
        to the message object buffer
        */
        ...
```

Instrumentation du noyau

```
FUNC(tpl_status, OS_CODE) tpl_receive_static_internal_unqueued_message(
    CONSTP2CONST(void, AUTOMATIC, OS_CONST)       rmo,
    P2CONST(tpl_com_data, AUTOMATIC, OS_VAR) data)
{
    VAR(tpl_status, AUTOMATIC) result = E_COM_FILTEREDOUT;

    /*  cast the base receiving mo to the correct type of mo                */
    CONSTP2CONST(tpl_internal_receiving_unqueued_mo, AUTOMATIC, OS_CONST)
    rum = rmo;
    /*  get the destination buffer                                          */
    P2VAR(tpl_com_data, AUTOMATIC, OS_VAR)
    mo_buf = rum->buffer.buffer;

    #if WITH_MONITOR == YES
    #if MONITORING_SENDING_MESSAGE == YES
    sending_event_table[rum->base_mo.monitor.buffer_id][tpl_kern.running_id]();
    #endif
    #endif

    /*  reception filtering                                                 */
    if (tpl_filtering(mo_buf, data, rum->base_mo.filter))
    {
        /*  get the size of the buffer
*/
        VAR(int, AUTOMATIC) size = rum->buffer.size;
        result =  E_OK;
        /*  copy the data from the source (data)
        to the message object buffer
        */
        ...
```

Implantation de la vérification en ligne

1 – Identification de l'événement à partir des paramètres et du contexte de l'appel système (ici l'id de la tâche et du buffer)

```
sending_event_table[rum->base_mo.monitor.buffer_id][tpl_kern.running_id]();
```

	b_0	b_1
τ_0	`s00()`	`empty()`
τ_1	`empty()`	`s11()`
τ_2	`empty()`	`empty`

Implantation de la vérification en ligne

1 – Identification de l'événement à partir des paramètres et du contexte de l'appel système (ici l'id de la tâche et du buffer)

```
sending_event_table[rum->base_mo.monitor.buffer_id][tpl_kern.running_id]();
```

	b_0	b_1
τ_0	s00()	empty()
τ_1	empty()	s11()
τ_2	empty()	empty

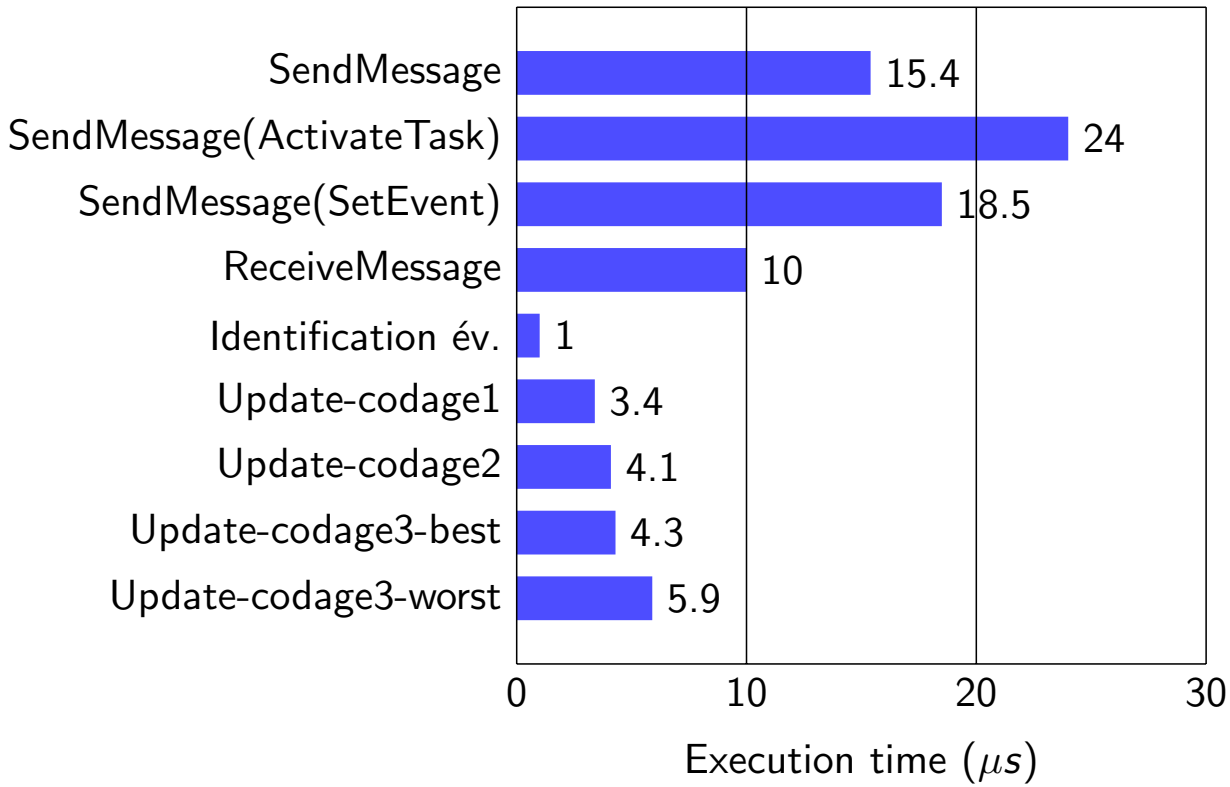
2 – Pour chaque moniteur concerné par cet événement

- Calcul du nouvel état courant à partir de la matrice de transition
- Signalisation d'un état final par appel d'une fonction fournie par l'intégrateur

```
FUNC(void, OS_CODE) s00()
{
    change_state(&prop1_desc, s00_prop1);
    ...
    change_state(&propN_desc, s00_propN);
}
```

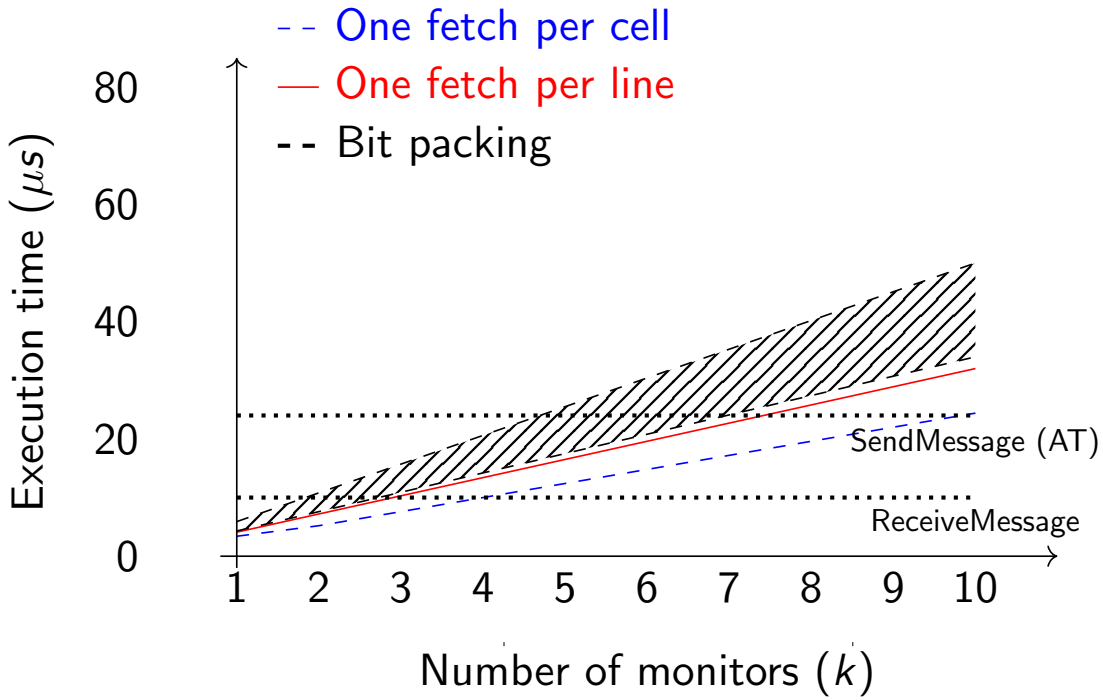
Performances : pour 1 moniteur

- Carte Olimex lpc2294, arm7TDMI, 60 MHz, *gcc -O2*



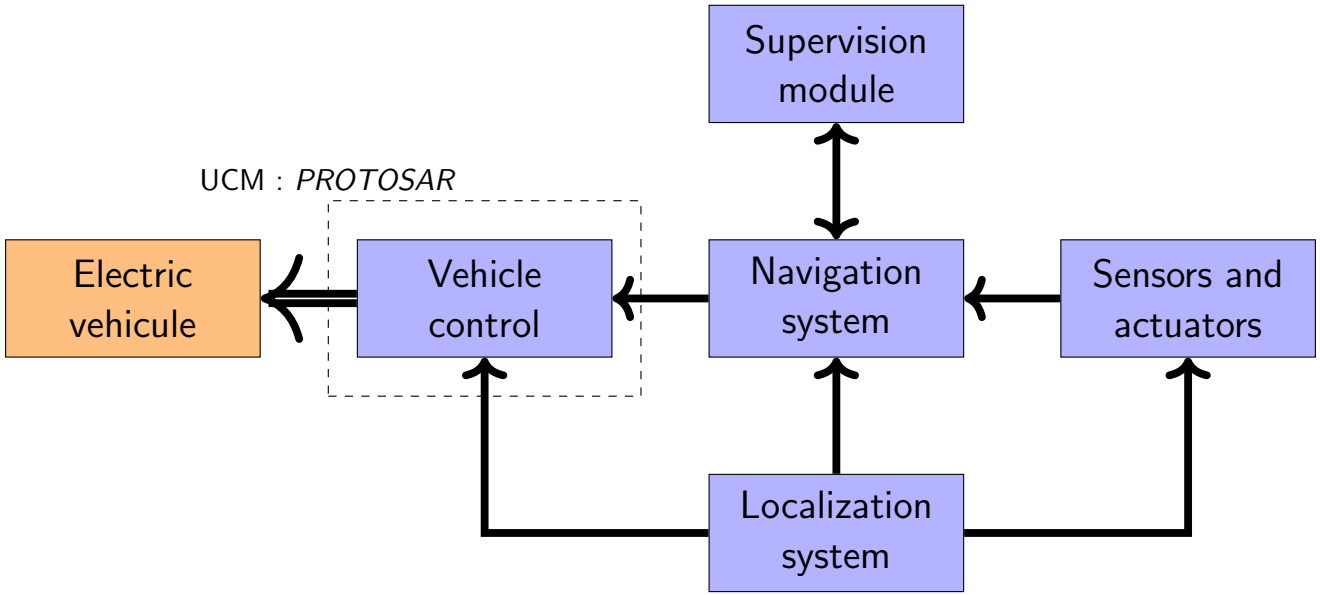
Performances : pour n moniteurs

- Carte Olimex lpc2294, arm7TDMI, 60 MHz, *gcc -O2*



Étude de cas : le projet PROTOSAR

Projet prototype de véhicule autonome pour les déplacements intra-site

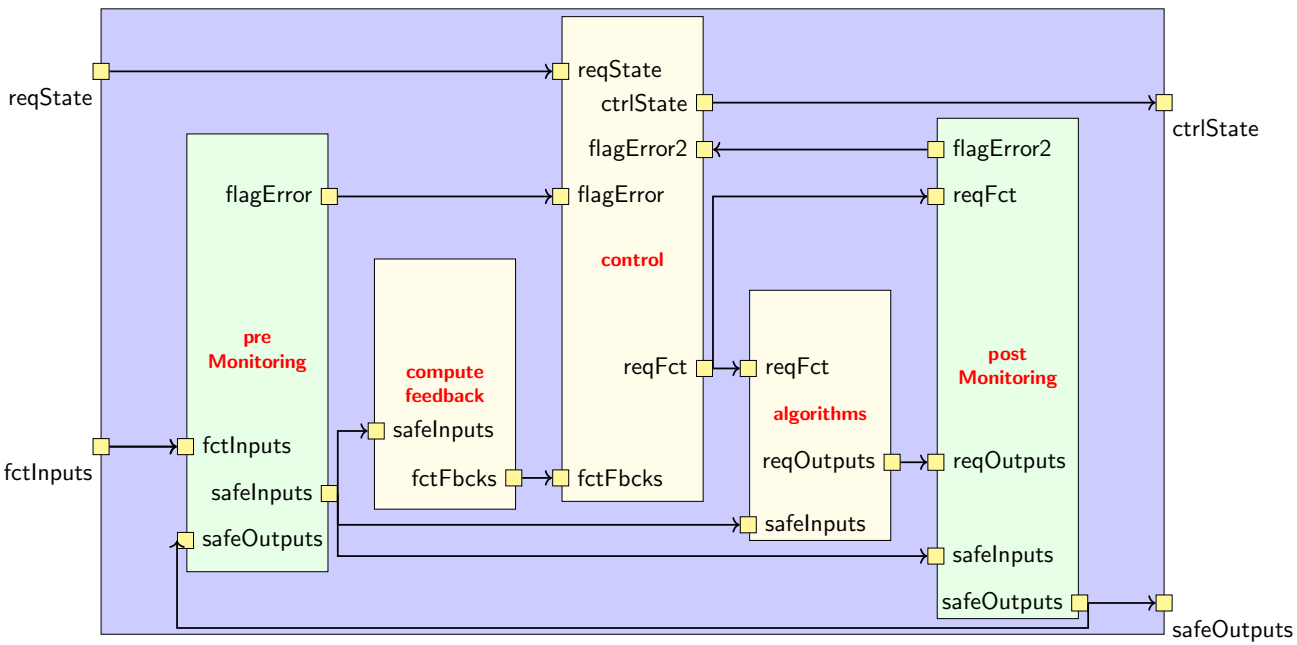


Étude de cas : le module Vehicle control

Architecture du module Vehicule control

7 composants : Trajectoire, Vitesse, Frein Parking, Levier de vitesse, Angle volant, Accélération et Frein

Chaque composant utilise le même patron, comportant un bloc fonctionnel (compute feedback, control, algorithms) et un bloc de monitoring des valeurs consommées et produites (pre-Monitoring, post-Monitoring)



Exemples de propriétés inter-tâches

Production et consommation

Chaque donnée produite par *preMonitoring* sur *safeInputs* est lue par *algorithms*

Propagation

Dès qu'une erreur est détectée par *preMonitoring*, *algorithms* n'émet plus sur *reqOutputs*

Cohérence

Le contrôle de vraisemblance réalisé par *postMonitoring* est fait sur des données cohérentes entre elles

...

Total : 284 événements à intercepter, 304 moniteurs

Empreinte mémoire

Implantation considérée : 1 tâche par bloc (donc 2 par composant)

Approche naïve

- 27.72 kB en ROM,
  - Moniteurs (tables de transition) = 14.52 kB
  - Code (fonctions de traitement des év.) = 13.2 kB
- 5.2 kB en RAM (descripteurs)

Optimisations

- Moniteurs codés par *Bit packing* : -50%
- Compositions de propriétés (1 moniteur pour plusieurs propriétés)
- Utilisation de modèles du système plus détaillés
- 6.5 kB en ROM, 2.3 kB en RAM

Section 3

Conclusions

Trampoline

- Exécutif temps réel pour « petit » micro-contrôleurs, conforme OSEK/VDX OS et AUTOSAR OS
- Système statique : ne pas hésiter à abuser de cette propriété
- Architecture modulaire, chaîne de développement extensible
- Logiciel libre sous licence GPL

Hot topics

- Isolation spatiale et temporelle dans les systèmes multicœur
- Virtualisation de système basé sur un exécutif sur des plateformes orientées multimédia embarqué
- Ré-intégrer la sécurité comme préoccupation de premier plan