Systèmes d'exploitation temps réel

-

Application au cas OSEK-VDX

Yvon Trinquet

IRCCyN - UMR CNRS 6597

École Centrale de Nantes, École des Mines de Nantes

Université de Nantes

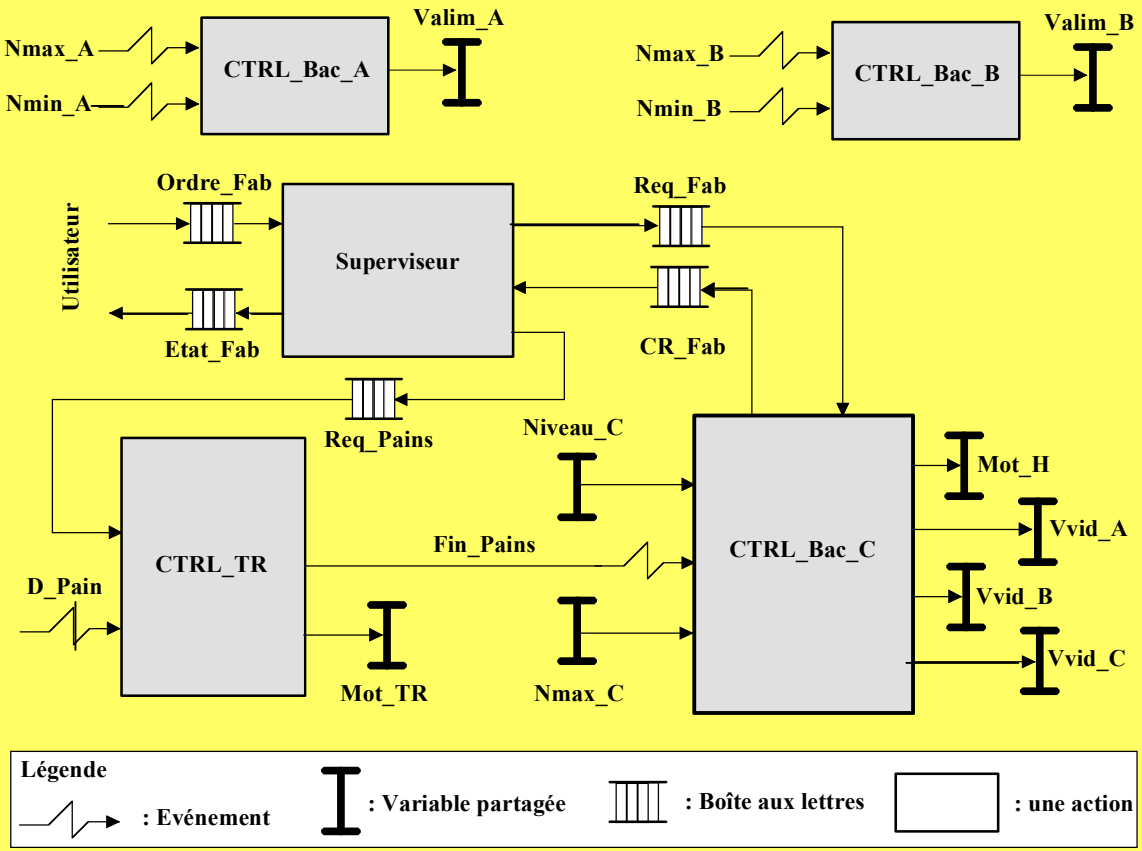
Équipe « Systèmes Temps Réel »

Yvon.Trinquet@irccyn.ec-nantes.fr

Plan

- **Généralités**
- **Le noyau temps réel OSEK/VDX OS**
- **La communication OSEK/VDX COM**
- **Évolutions, modèles d'application**

Exemple d'architecture fonctionnelle (ou logicielle) d'une application



❄ Quelle implémentation ?

- choix de la configuration matérielle
  - contraintes topologiques
  - contraintes de sûreté de fonctionnement
  - contraintes temporelles
- lien « Tâche – Événement activeur »
  - approche « **synchrone** »
  - approche « **asynchrone** »

- la démarche **synchrone** :
  - ✓ Hypothèse « temps nul » pour les calculs et les réactions
  - ✓ Simultanéité possible des événements
  - ✓ Cadre formel ➔ vérification formelle

mais … le « monde » est asynchrone

- la démarche **asynchrone** :
  - ✓ Les calculs prennent du temps
  - ✓ Les événements ne sont jamais simultanés
  - ✓ Il faut accepter la préemption et ordonnancer, protéger les ressources partagées
  - ✓ Modèles beaucoup plus complexes, plus durs à vérifier

Un support d'exécution : **l'exécutif temps réel**

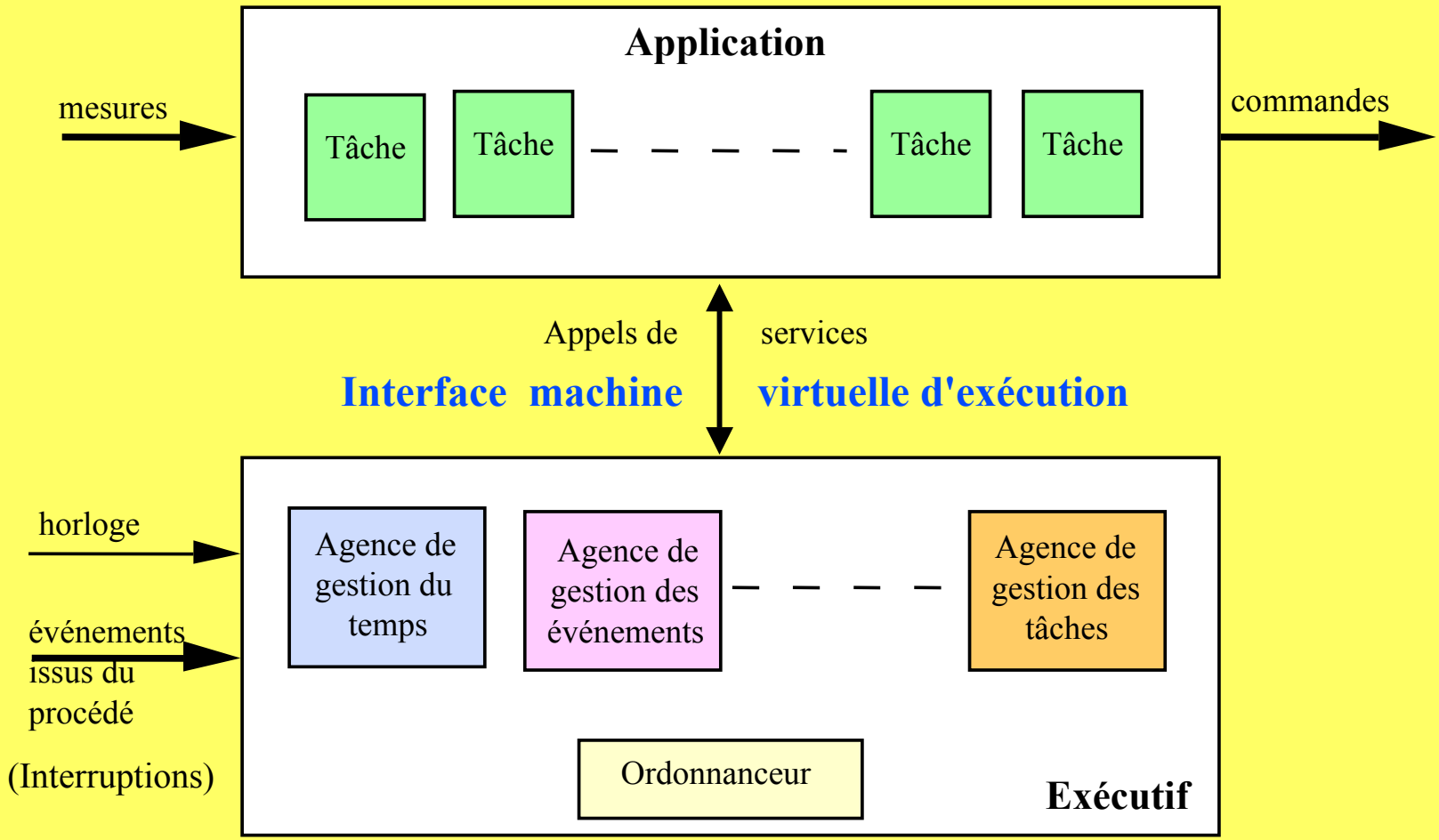
Types d'exécutifs

- les exécutifs « généralistes »
  beaucoup de produits commerciaux (VxWorks …)
  un standard OSEK-VDX pour de petits systèmes embarqués

- les exécutifs Unix TR
  - approche Posix (standards IEEE) : extensions temps réel pour Unix (ordonnancement, sémaphores, multi-threads …)
  - approche Linux temps réel : RT-Linux, RTAI

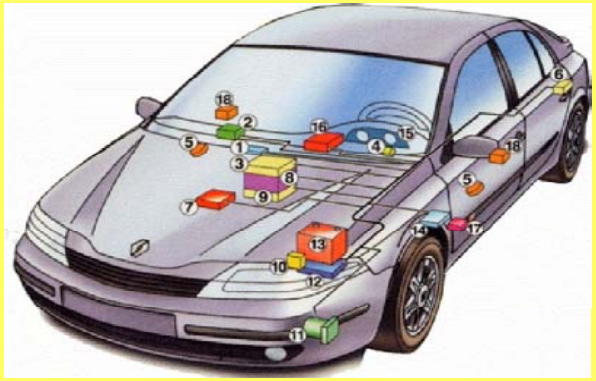
- les exécutifs « recherche » : Mars (Kopetz), Spring kernel (Stankovic), ARTS (Lehoczky) …

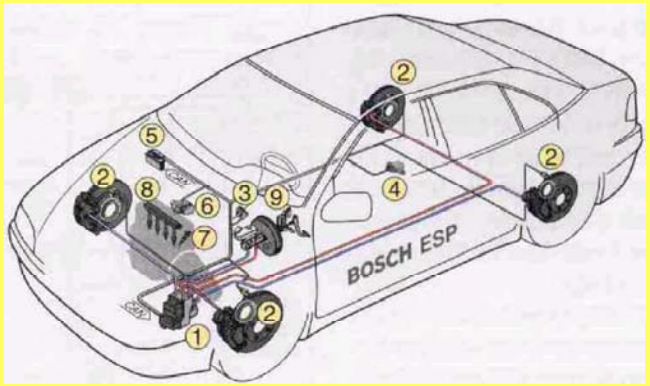
L 'exécutif et l 'application



❄ contexte d'OSEK/VDX (1)

celui de « l'électronique » embarquée dans les véhicules





- contraintes **temps réel** (dures et souples) :

  (powertrain, chassis, body, telematics)

- **sûreté de fonctionnement** élevée
  - ✓ mais mission en principe interruptible
  - ✓ avènement du X by Wire

❄ contexte d'OSEK/VDX (2)

- support matériel minimal (peu de RAM, ECU 8 et 16 bits)
- **architecture distribuée** autour de ≠ réseaux :
  CAN, LIN … puis TTP/C, FlexRay …
- fonctions transversales :
  inter-opérabilité des sous-systèmes pour réaliser la fonction
- flexibilité de l'architecture :
  - ✓ ajout de fonctions,
  - ✓ **portabilité** et **réutilisabilité** des fonctions logicielles)

Et tout cela … au moindre coût !

❄ OSEK/VDX : historique

- Proposition du groupe OSEK, constitué :
  - ✓ de constructeurs (BMW, DaimlerChrysler, Renault, PSA, etc.)
  - ✓ d'équipementiers (Bosch, Siemens, etc.)
  - ✓ d'universitaires (Univ. Karlsruhe)

- Fusion des projets **OSEK** (Allemand) et **VDX** (GIE PSA-Renault)

- Travaux commencés en 1995 pour OSEK, vers 1992 pour VDX

❄ OSEK/VDX :

OSEK/VDX OS	version 2.2,	septembre 2001
	noyau du système d'exploitation	
OSEK/VDX COM	version 3.0.1,	janvier 2003
	services pour la communication	
OSEK/VDX NM	version 2.5.1	Mai 2000
	gestion et surveillance du réseau	
OSEK/VDX OIL	version 2.4.1	janvier 2003
	langage de description des applications	

Le site Web de référence : **http:// www . osek-vdx . org**

Le noyau temps réel

OSEK-VDX OS

Principaux services d'OSEK/VDX OS

- Services pour les **tâches**
- Services de synchronisation (**événements**)
- Services d **'exclusion mutuelle**
- Services pour les phénomènes récurrents (**compteurs et alarmes**)
- Service pour la **communication** (dans OSEK/VDX COM)
- Services pour la gestion des **interruptions**
- Services **systèmes** et gestion des **erreurs**

Tous les objets sont statiques :

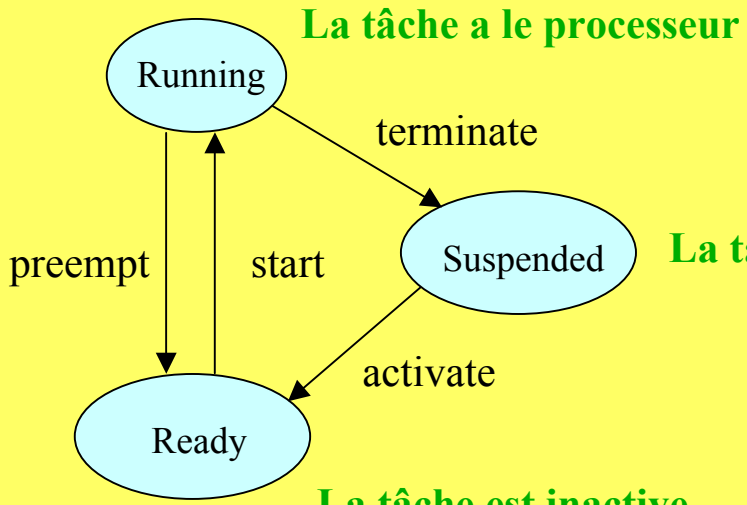
- pas de création dynamique d'objets
- pas de suppression dynamique d'objets

Les tâches (1)

Tâche = objet actif de l'application

- **La tâche « Basique »** :
  - module sans point bloquant
  - doit se terminer
  - 3 états : suspended / ready / running
- **activations multiples** : mise en file des requêtes d'activation si la tâche est active (pas d'instanciation)

Diagramme des états



Les tâches (2)

ActivateTask

TerminateTask

ChainTask

Schedule

```
Task T1
{
  ---
  ---
  Activate (T2)
  ---
  TerminateTask ()
}
```

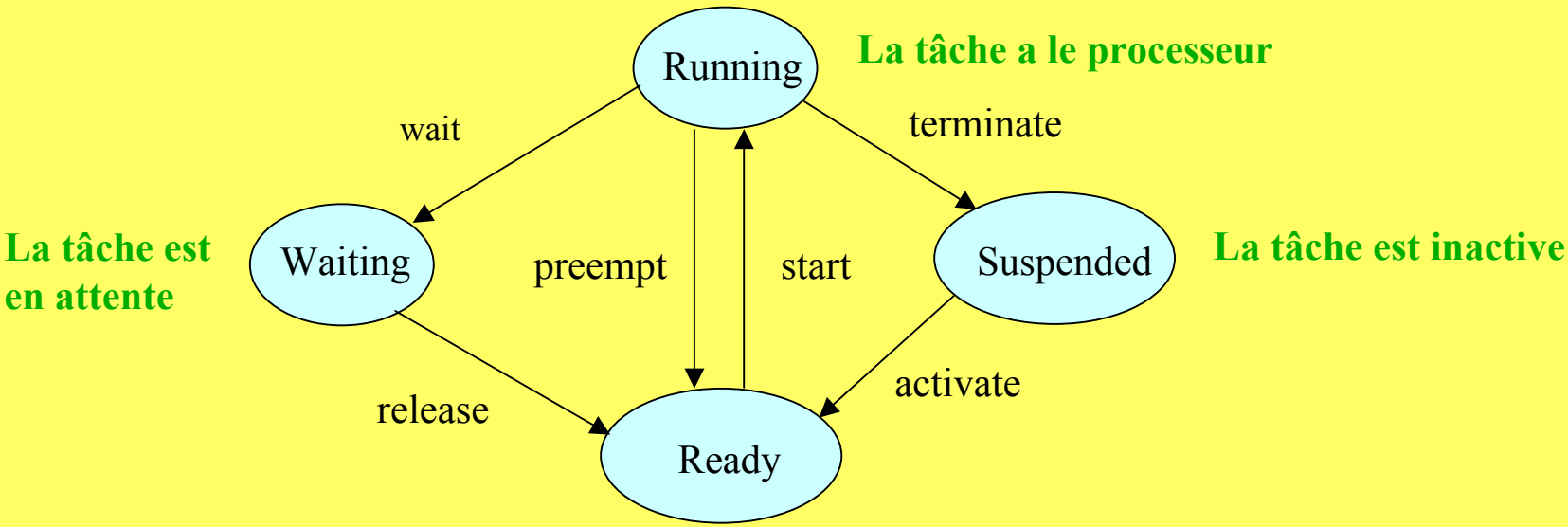
```
Task T2
{
  ---
  ---
  ---
  ChainTask (T3)
}
```

```
Task T3
{
  ---
  ---
  ---
  TerminateTask ()
}
```

+ d'autres services …

- **La tâche « Étendue »** :
  - un ou plusieurs modules
  - peut attendre une occurrence d'événement
  - 4 états : waiting en plus

Diagramme des états



ordonnancement (1)

- **priorité fixe** non modifiable (la valeur 0 est la plus faible priorité)
- **3 modes** d 'ordonnancement :
  - ✓ **préemptif** : toutes les tâches en mode préemptif
    - ➢ Une tâche peut être interrompue au profit d'une autre plus prioritaire
  - ✓ **non préemptif** : toutes les tâches en mode non-préemptif
    - ➢ Une tâche perd le processeur lorsqu'elle le demande explicitement
  - ✓ **mixte** (attributs des tâches : préemptif ou non préemptif)
    - ➢ Intérêts du non-préemptif dans un contexte préemptif :
      - si temps d'exécution faible, peu différent du temps de commutation de contexte
      - engendre moindre consommation de RAM
      - Non préemption pour protection d'une ressource

ordonnancement (2)

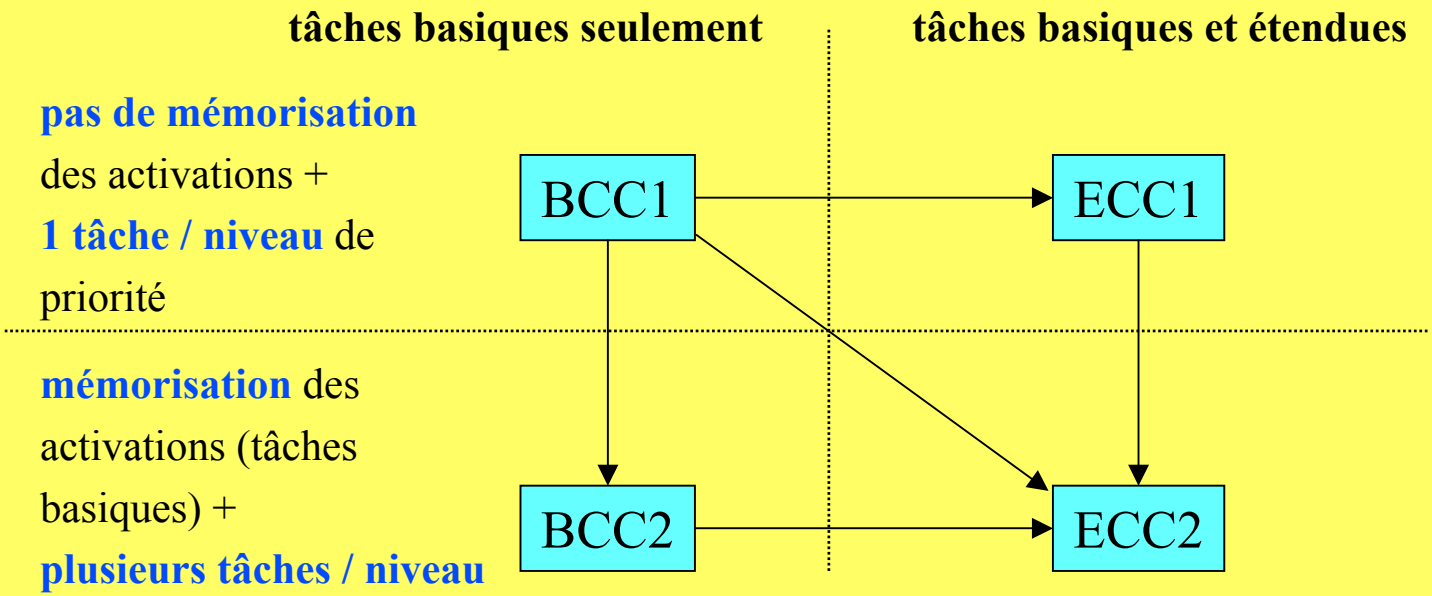
- gestion des **ressources partagées** avec **« Priority Ceiling Protocol »** qui influe sur l'ordonnancement.

- notion de **groupe de tâches** :
  - ✓ Un groupe = un ensemble de tâches liées par une ressource interne
  - ✓ Les tâches internes au groupe, partageant une ressource ne peuvent se préempter :
    - ➢ C'est intéressant pour des tâches coopérantes (à l'aide d'une même ressource)
    - ➢ C'est plus simple que la gestion classique de ressource car complètement transparent à l'application
  - ✓ Pour les tâches externes qui ont une priorité ≤ à la plus haute priorité du groupe, les tâches du groupe sont non-préemptives
  - ✓ Pour les tâches externes qui ont une priorité > à la plus haute priorité du groupe, les tâches du groupe sont préemptives

❄ classes de conformité (1)

créées pour adapter le noyau :

- aux besoins de l'application
- aux ressources matérielles disponibles



❄ synchronisation par événements

- événements **privés** : 1 événement est la propriété d'une tâche étendue
  (c'est l'implémentation qui fixe le nombre d'événement d'une tâche)

  ➔ seul le propriétaire peut invoquer le service d'attente

- modèle **n** producteurs / **1** consommateur

  ➔ n tâches peuvent invoquer le service de signalisation
  1 tâche invoque le service d'attente

- **attente explicite** des consommateurs (synchrone)

  par opposition à la signalisation asynchrone d'Unix

❄ synchronisation par événements

- occurrences **mémorisées, effacement explicite**

  ➔ l'effacement de l'occurrence est à la charge du propriétaire

- attente possible sur une **liste** d'événements en **OU**

  ➔ - la première occurrence réveille la tâche
  - si elle est survenue lors de l'appel la tâche n'est pas bloquée

- **pas de chien de garde** dans les services (attente non gardée)

  ➔ utilisation des alarmes pour cela

Événements (3)


```
Task T3
{
  ---
  ---
  SetEvent (T2, EV2)
  ---
  TerminateTask ()
}
```

Tâche basique

Tâche basique

```
Task T1
{
  ---
  ---
  SetEvent (T2, EV1)
  ---
  TerminateTask ()
}
```

```
Task T2
{
  Loop
    WaitEvent (EV1, EV2)
    ---
    ClearEvent(EV1,EV2)
    ---
  EndLoop
}
```

Tâche étendue

❄ Les ressources et l'exclusion mutuelle

- Ressources à usage unique : code, périphérique …
- Besoin d'un accès coordonné aux **ressources à usage exclusif** :

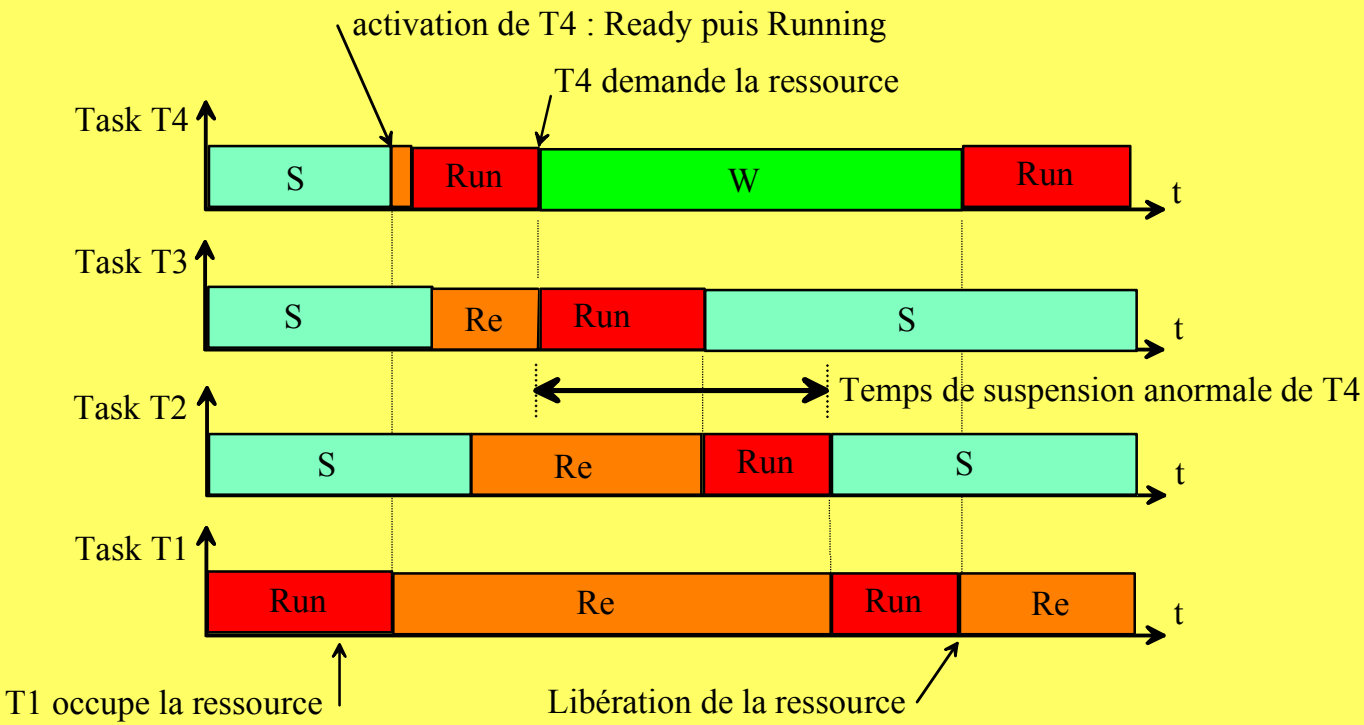
  section critique ➔ **exclusion mutuelle d'accès**

- La gestion des ressources doit assurer que :
  - ✓ 2 tâches n'occupent pas la ressource en même temps
  - ✓ il n'y a pas d'**inversion de priorité**
  - ✓ il n'y a pas **d'interblocage** entre tâches

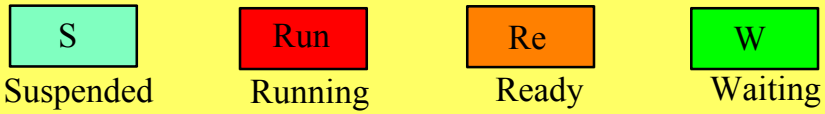
L'inversion de priorité

Priorité T4 > Priorité T3 > …

T1 et T4 utilisent la même ressource

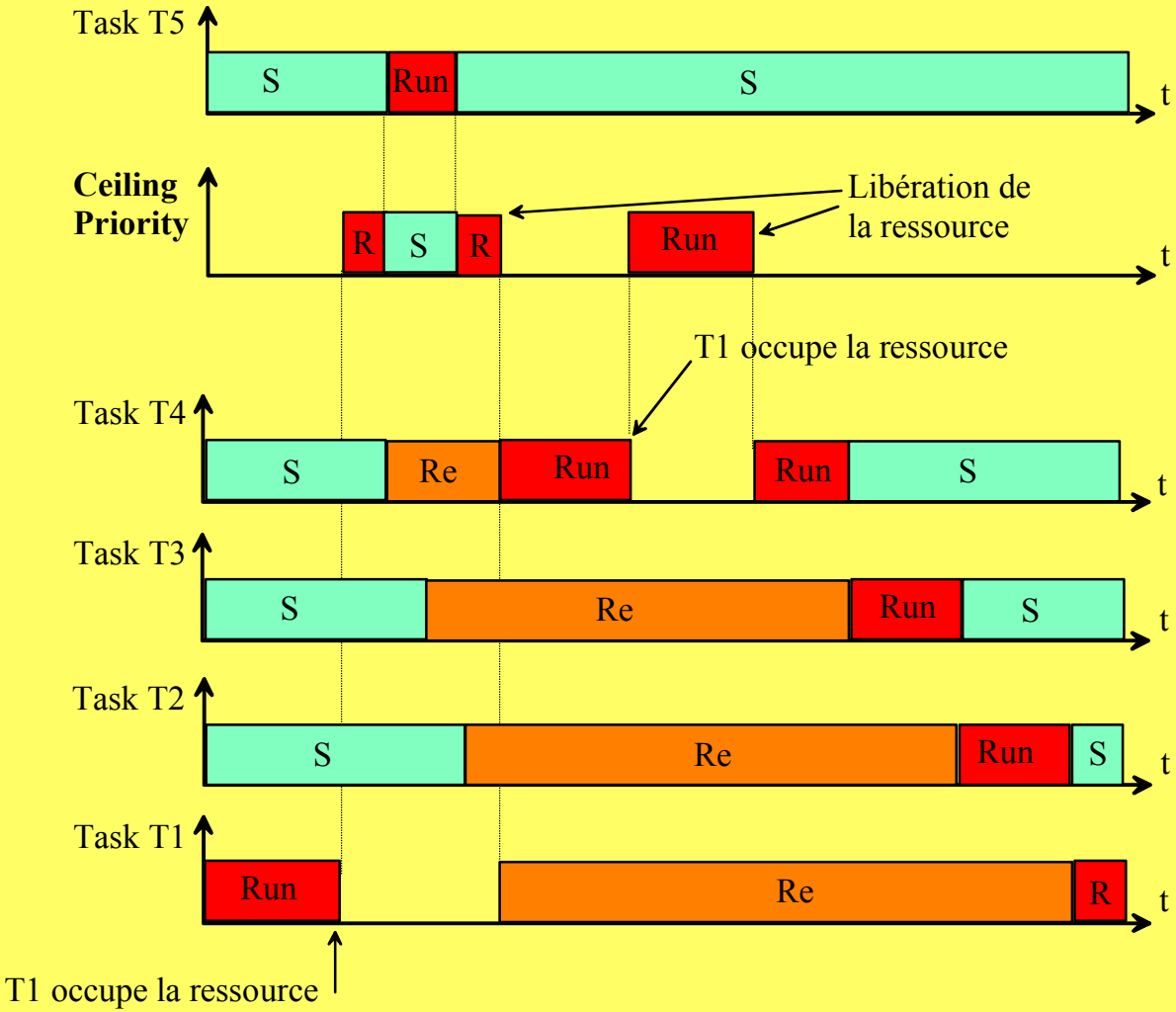


Légende



Le protocole « Priority Ceiling Protocol »

Ressources (3)



Priorité T5
> Priorité T4
> Priorité T3
> …

T1 et T4 utilisent la même ressource

Légende

S — Suspended

Run — Running

Re — Ready

W — Waiting

Les ressources et l'exclusion mutuelle

- services pour **prendre** et **relâcher** explicitement une ressource

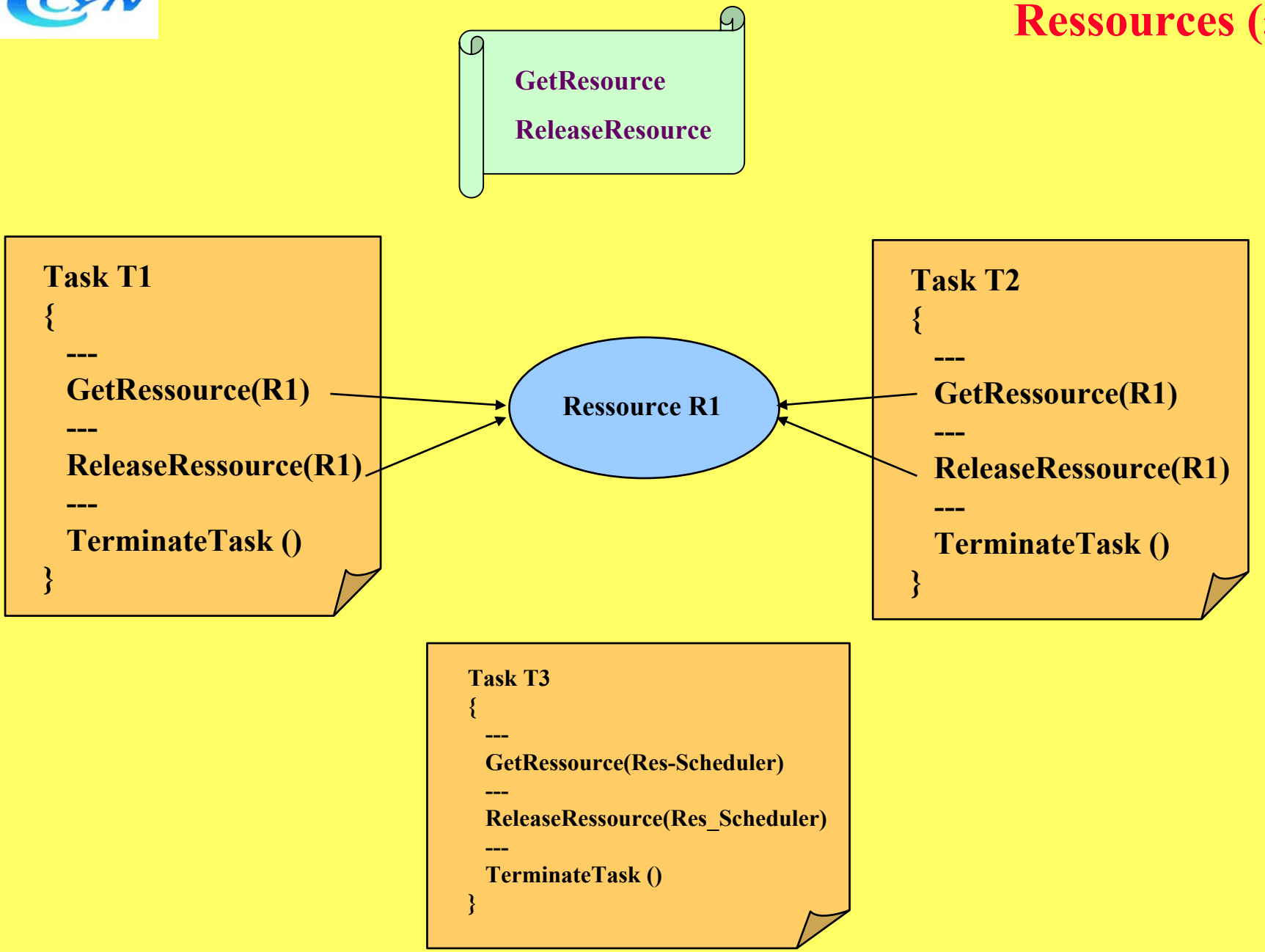
- utilisation du protocole **PCP** pour éviter :
  - les inversions de priorités
  - les blocages inter-tâches

- une ressource standard : **Res_scheduler**
  (passage en mode non-préemptif)

- **restriction** d'appels de services à l'intérieur d'une section critique

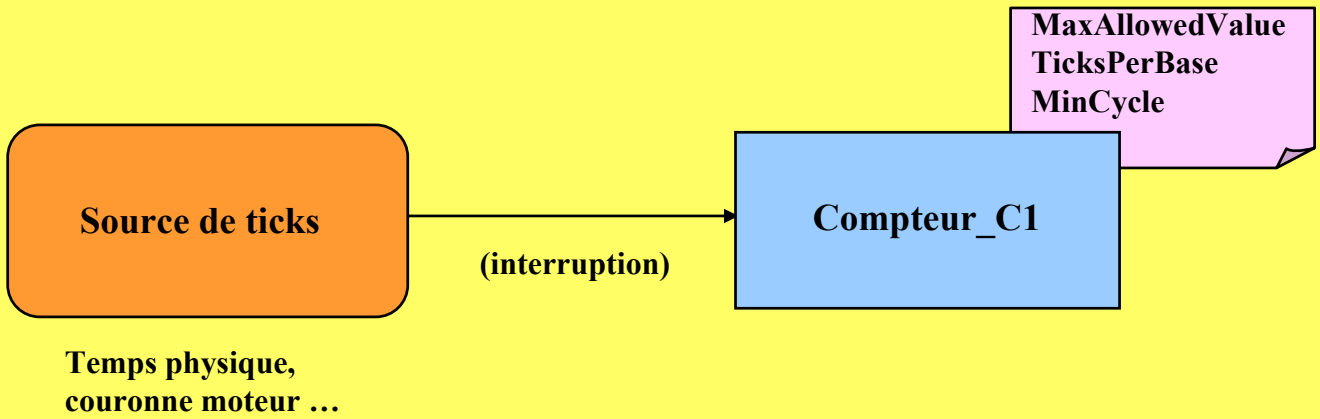
  TerminateTask, ChainTask, Schedule, WaitEvent :
  ne doivent pas être appelés dans une section critique

Ressources (5)



❄ alarmes et compteurs

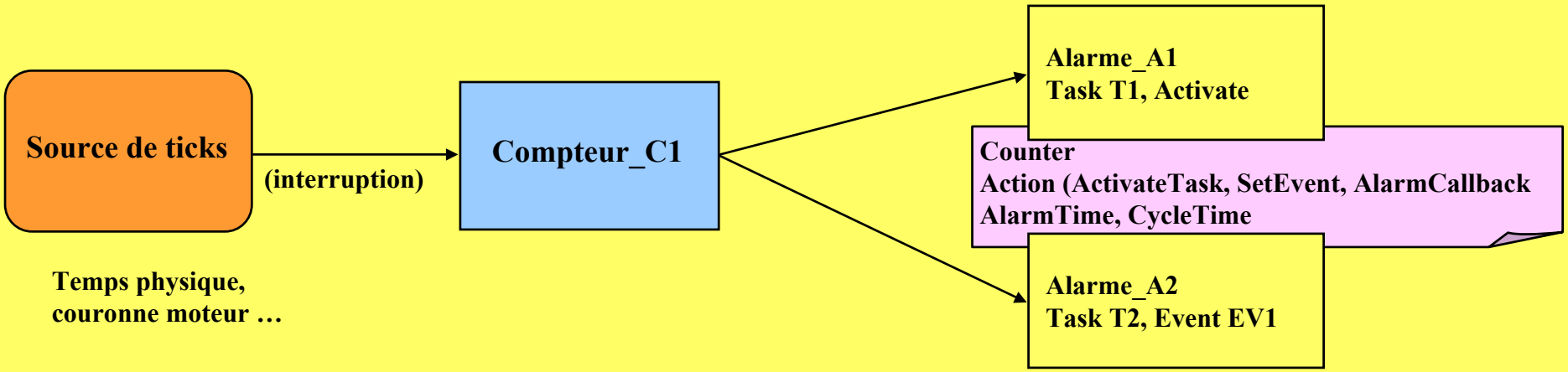
- **compteur**
  - ✓ manipulable au travers de OIL (Osek Implementation Language)
  - ✓ enregistrement de « ticks » externes, éventuelle pré-division
  - ✓ compteur **fini** avec RAZ automatique



❄ alarmes et compteurs

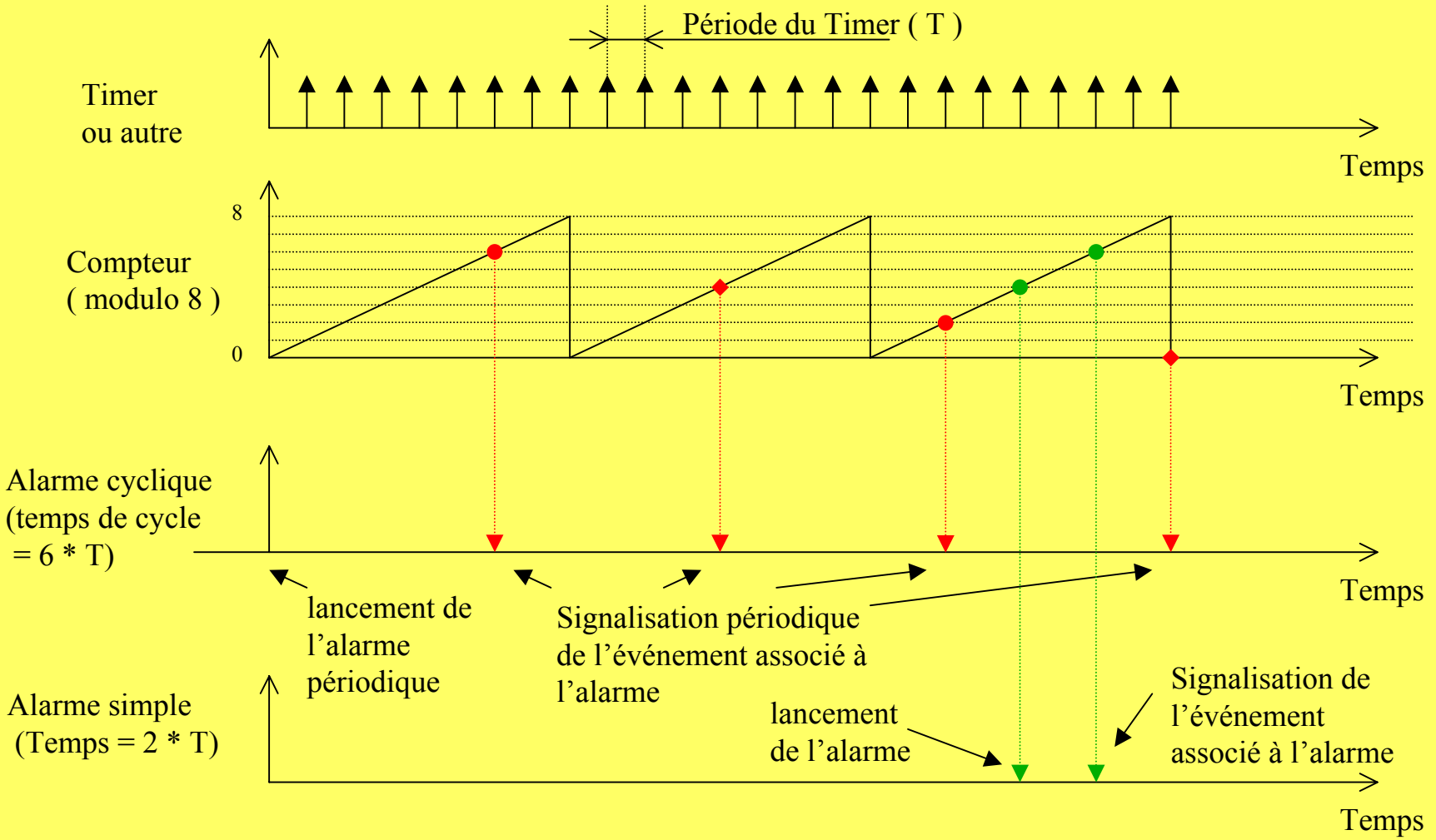
- **alarme**

  - ✓ attachée à **1 compteur** et **1 tâche**
  - ✓ **unique** ou **cyclique,** déclenchement **absolu** ou **relatif**
  - ✓ l'alarme est déclenchée lorsque le **compteur** atteint une **valeur de référence**
  - ✓ sur expiration :
    - **activation** de la tâche, ou
    - **signalisation** d'un événement, ou
    - exécution d'une routine « **Callback** ».



Alarmes (3)

exemples

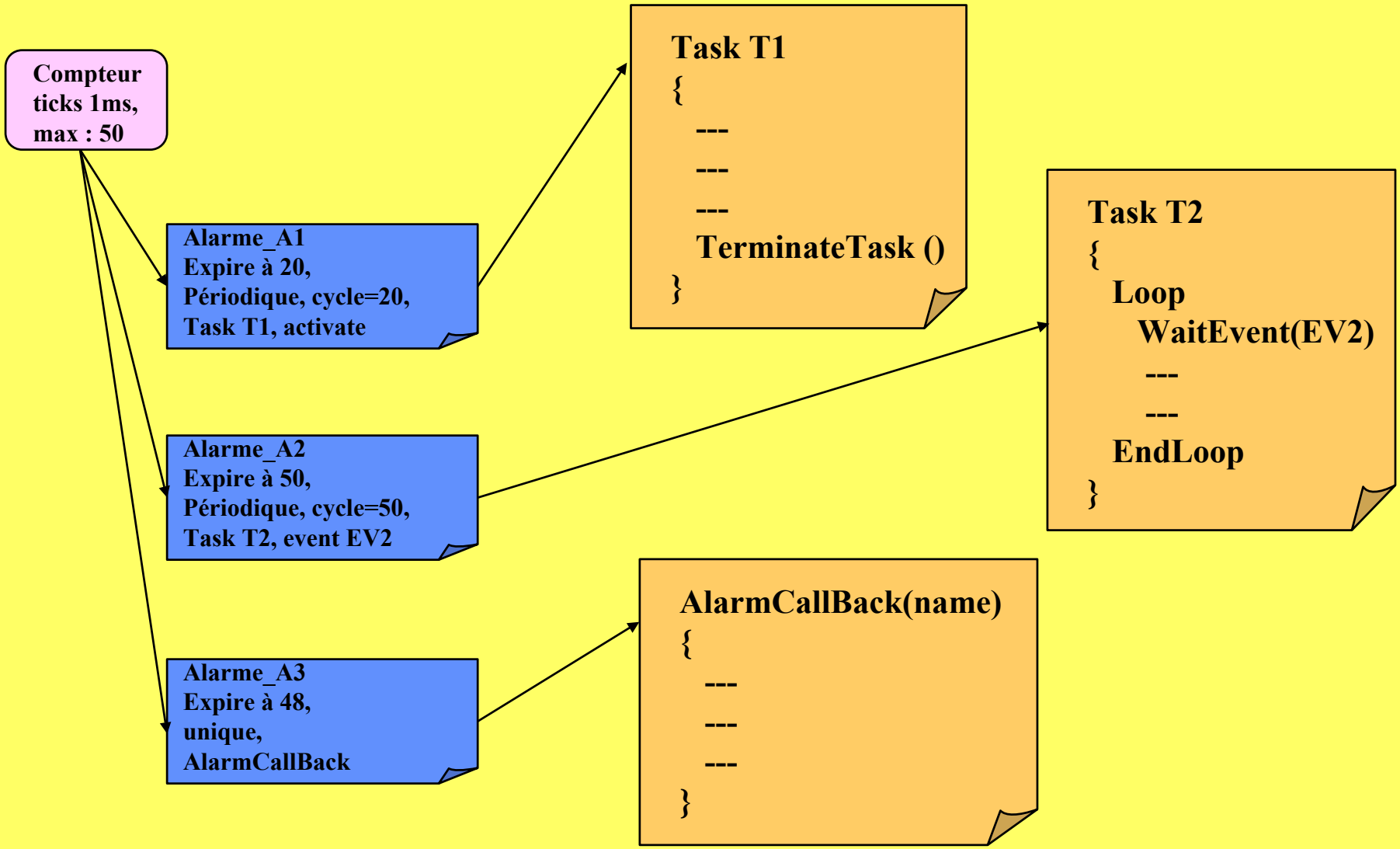


Applications

- activation de **tâches périodiques** ou non
- **signalisation** d ’occurrences d **’événement** périodiques ou non périodiques
- **garde temporelle**

SetRelAlarm **SetAbsAlarm** **GetAlarm** **CancelAlarm**

Alarmes (5)

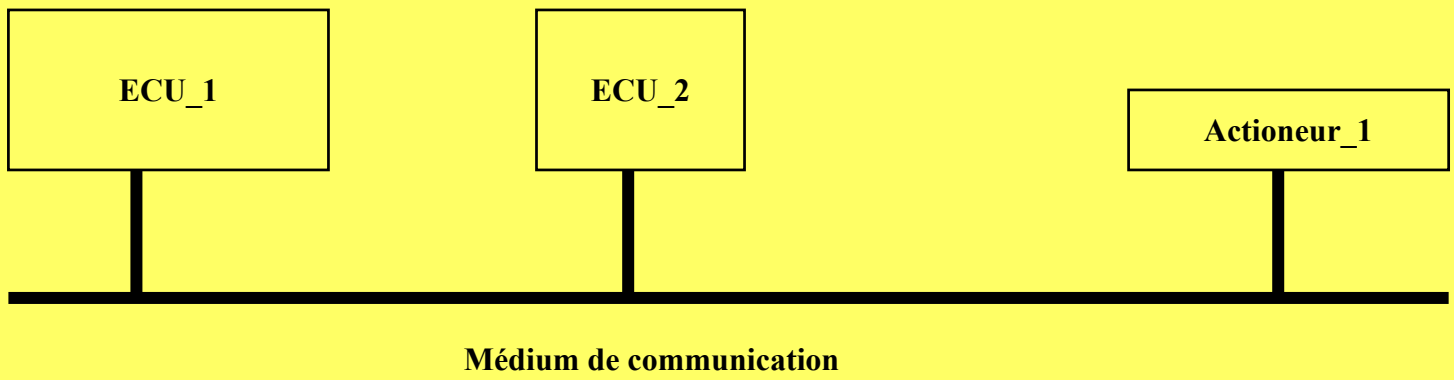

Compteur ticks 1ms, max : 50
Alarme_A1 Expire à 20, Périodique, cycle=20, Task T1, activate
Alarme_A2 Expire à 50, Périodique, cycle=50, Task T2, event EV2
Alarme_A3 Expire à 48, unique, AlarmCallBack
Task T1 { --- --- --- TerminateTask () }
Task T2 { Loop WaitEvent(EV2) --- --- EndLoop }
AlarmCallBack(name) { --- --- --- }

La communication

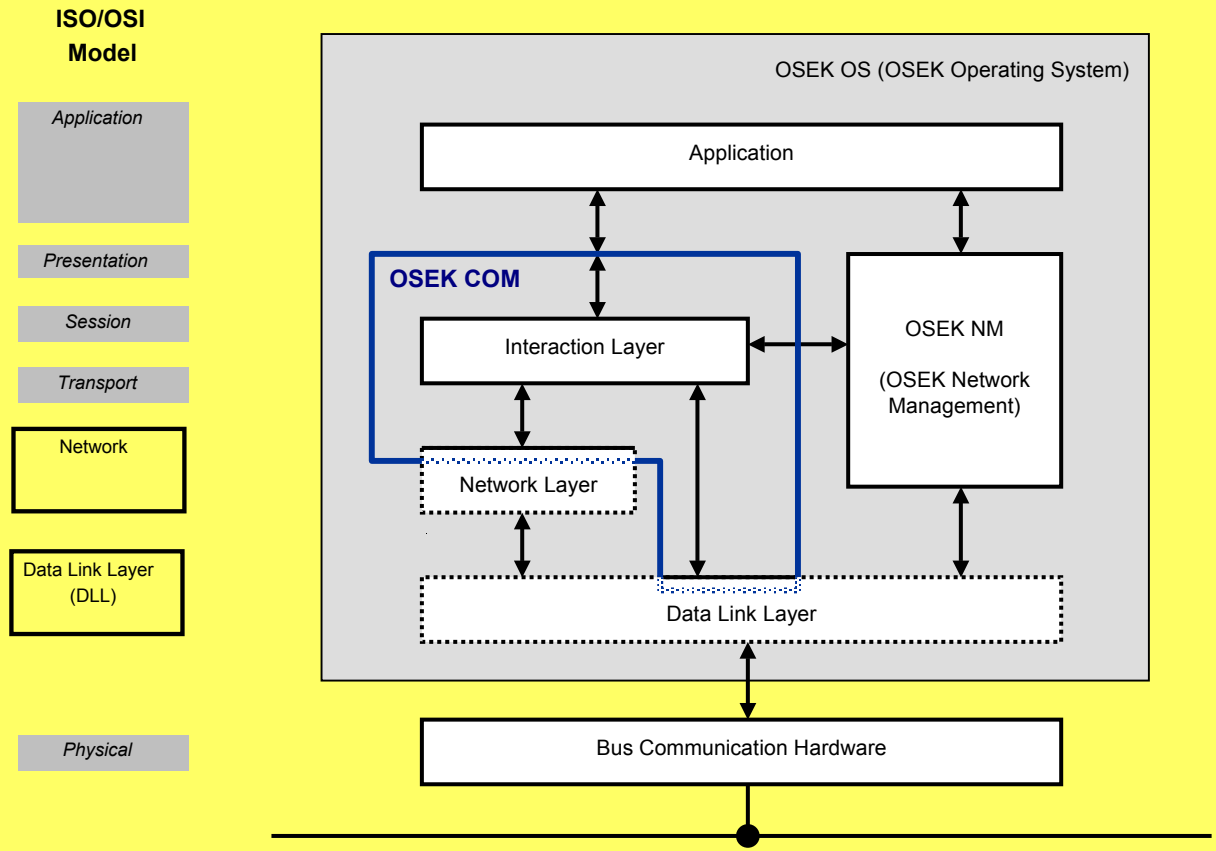
OSEK-VDX COM

La communication

- **Portabilité**, **réutilisation** et **inter-opérabilité** des logiciels d'application
  - ✓ Même interface pour les communications internes et externes
  - ✓ Indépendance vis-à-vis du protocole de communication sous-jacent
- Adaptation au contexte (**scalability**)
- **Configuration statique** de la communication



OSEK COM (2)



OSEK-VDX / COM v3.0

(courtoisie Christophe Marchand (PSA))

OSEK COM (3)

- services basés autour des objets **messages**

  1 message = 1 nom + 1 type de données + des attributs

- modèle de communication **m : n** :

  ✓ Écriture dans le message : plusieurs écrivains possibles (même site)

  ✓ Lecture du message : plusieurs lecteurs possibles (un site, sites ≠)

- des **classes de conformité** pour s'adapter au contexte

  ✓ CCCA et CCCB : communication **interne** à un site seulement

  ✓ CCC0 et CCC1 : communication **intra** et **inter-sites**

OSEK COM (4)

- Modèle de communication **asynchrone**

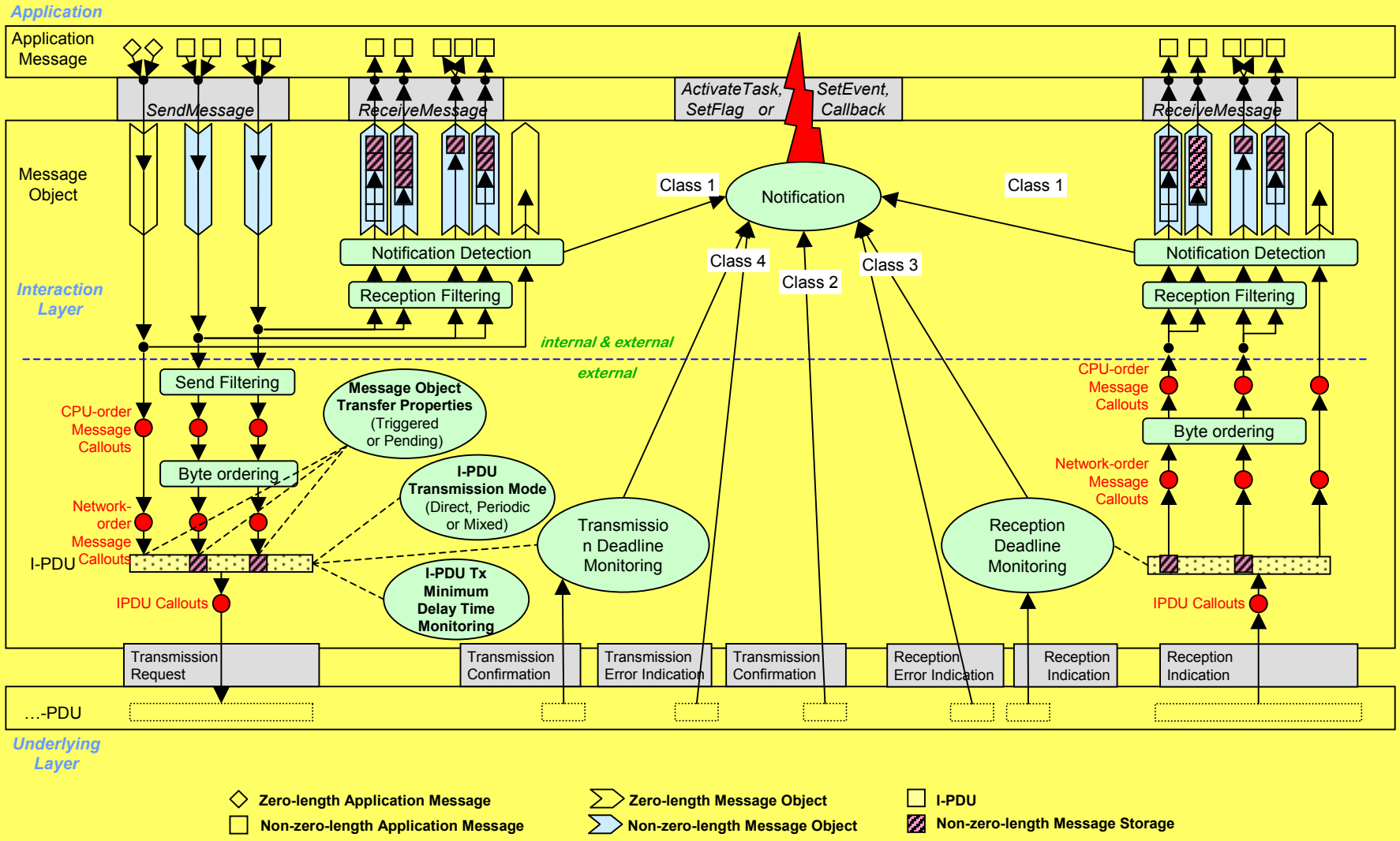
  - ✓ structure « **tableau noir** » (**Unqueued message**), variable rafraîchie
  - ✓ structure à **file FIFO** (**Queued message**), boîte aux lettres classique

- **pas de suspension** de la tâche émettrice pendant la transmission
  pas de blocage si message non disponible pour la tâche réceptrice

⇒ resynchronisation par

  - **polling** (sur une variable d'état)
  - **activation** tâche sur **fin d'envoi ou réception**
  - **signalisation** d'occurrence **sur fin d'envoi ou réception**
  - exécution routine **Callback**
  - **alarme** sur **garde temporelle** (émission / réception)

OSEK COM (10)



(courtoisie Christophe Marchand (PSA))

Filtrage sur les messages

Source doc OSEK-VDX COM

Algorithm Reference	Algorithm	Description
F_Always	True	No filtering is performed so that the message always passes
F_Never	False	The filter removes all messages
F_MaskedNewEqualsX	*(new_value&mask) == x*	Pass messages whose masked value is equal to a specific value
F_MaskedNewDiffersX	*(new_value&mask) != x*	Pass messages whose masked value is not equal to a specific value
F_NewIsEqual	*new_value == old_value*	Pass messages which have not changed
F_NewIsDifferent	*new_value != old_value*	Pass messages which have changed
F_MaskedNewEqualsMaskedOld	*(new_value&mask) == (old_value&mask)*	Pass messages where the masked value has not changed
F_ MaskedNewDiffersMaskedOld	*(new_value&mask) != (old_value&mask)*	Pass messages where the masked value has changed
F_NewIsWithin	*min <= new_value <= max*	Pass a message if its value is within a predefined boundary
F_NewIsOutside	*(min > new_value) OR (new_value > max)*	Pass a message if its value is outside a predefined boundary
F_NewIsGreater	*new_value > old_value*	Pass a message if its value has increased
F_NewIsLessOrEqual	*new_value <= old_value*	Pass a message if its value has not increased
F_NewIsLess	*new_value < old_value*	Pass a message if its value has decreased
F_NewIsGreaterOrEqual	*new_value >= old_value*	Pass a message if its value has not decreased

Exemples de services

Evolutions : OSEKtime, OSEK FT Com

Modèles d'application

Évolutions

Évolution conjointe avec une approche TT (Time-Triggered)

- **OSEKtime OS** 1.0 juillet 2001
- **OSEK FT com** 1.0 juillet 2001
  (Fault-Tolerant Communication)

❄ Aperçu d'OSEKtime OS

Contexte : TTCAN - TTP/C - FlexRay

- **2 types** de tâches :
  - Les tâches TT
  - Les tâches OSEK-VDX OS (classiques, services précédents)
- **Tâche TT** = un module (point d'activation, point de sortie) (suspended, running, preempted)
- **Plan statique d'ordonnancement** calculé hors-ligne
- Surveillance des **échéances** (marques dans le plan d'ordonnancement)
- Utilisation des temps libres pour l'exécution des tâches OSEK-VDX classiques

❄ Modèles d'application (1)

Classes de conformité : souci d'implémentation, pas de vérification

➔ Modèles d'application

- du plus simple (analysable formellement)
- au plus complexe (validation par simulation)

Exemples de modèles

- Tâches basiques, périodiques, indépendantes
  (la coopération par message « unqueued » n'est pas une dépendance)
- Tâches basiques, périodiques, indépendantes avec offset initial
  (l'offset peut être géré par une alarme non périodique)

Modèles d'application (2)

Exemples de modèles (suite)

- Tâches basiques, périodiques, avec partage de ressource
- Tâches basiques, périodiques + apériodiques (gestion par un serveur de tâches), indépendantes
- Tâches basiques, périodiques, dépendantes par messages (« queued ») internes à un site ou inter-sites
- Tâches basiques périodiques, tâches étendues périodiques (infos sur les causes et temps de blocage)
- - - - - -
- Tâches étendues cycliques, coopérant via le réseau, partageant des ressources, préemptives et non-préemptives …

Conclusion

- OSEK/VDX OS, COM, NM, OIL

  une proposition mature et complète

- OSEKtime OS et FTCom

  une architecture TT + communication FT (à promouvoir)

- Des produits industriels (Motorola, WindRiver …) et des produits propriétaires (équipementiers)

- Un pas important vers la portabilité des applications, la réutilisation de composants dans les ECU …