# APPLICATIONS EMBARQUEES DISTRIBUEES


Filière « Systèmes embarqués »

R.KOCIK

# PLAN

I. Mise en oeuvre d'applications distribuées

1. Introduction
2. Architectures parallèles
3. Limites du parallélisme
4. Parallélisation d'algorithmes

II. Applications distribuées et Temps réel

5. Introduction
6. Techniques de communication entre tâches
7. OSEK
8. Ordonnancement de communications
9. Synchronisation d'horloges

# I. Mise en oeuvre d'applications distribuées

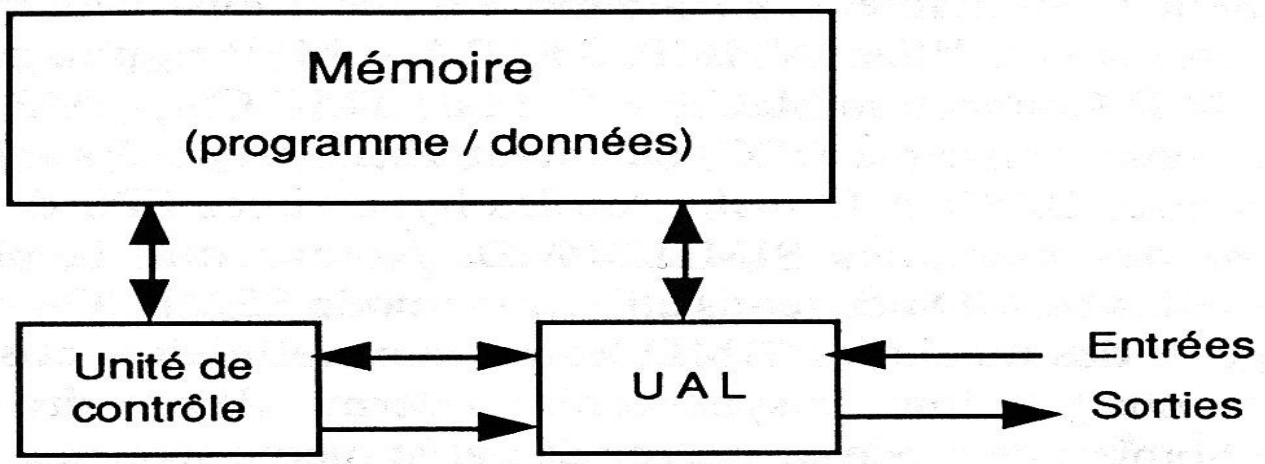
1. Introduction
   - Notion de parallélisme
   - Formes de parallélisme
   - Le // pour répondre à quels besoins ?
2. Architectures parallèles
3. Limites du parallélisme
4. Parallélisation d'algorithmes

# Notion de parallélisme

- « un ordinateur parallèle est une machine composée de plusieurs processeurs couplés qui travaillent en même temps et coopèrent à la résolution d'un même problème »
- Système distribué, système réparti
- Depuis 80 développement ➚

# Modèle d'architecture

- 1 processeur : séquenceur d'instruction (UC) , Unité de traitement (UAL) et d'une mémoire
- Exécution des instructions arithmétiques ou logiques du programme une par une sur des valeurs en mémoire



# Caractéristiques des architectures //

- Plusieurs unités de traitement
- Interconnections : des entrées-sorties servent à connecter les processeurs
- Etat partagé : les processeurs coopèrent pour maintenir un état commun, c.a.d que les opérations sont coordonnées.

# Exemple introductif

La famille Dupont organise un pique-nique composé de sandwiches. Avant l'arrivée des convives, la famille entreprend la confection de ces sandwichs. Comment s'y prendre ?

- Mme Dupont fait tout le travail,

- L'un coupe le pain, le deuxième beurre la tranche, le troisième met la viande et le quatrième range le sandwich sur un plat,

- Mme D. supervise les opérations, les autres se placent autour de la table avec du pain, du beurre de la viande. Sur ordre de Mme D. chacun coupe la tranche, la beurre et met la viande en même temps et donne le sandwich à Mme D.

- Chaque personne se charge indépendamment des autres de la confection des sandwichs

# Formes de parallélisme

- **Contrôle** : chaque processeur exécute des actions, processus, tâches qui lui sont propre
- **Données** : réalisation d'une action identique sur tous les processeurs mais sur des données différentes
- **Pipe-line** : travail à la chaîne

# Domaines d'utilisation

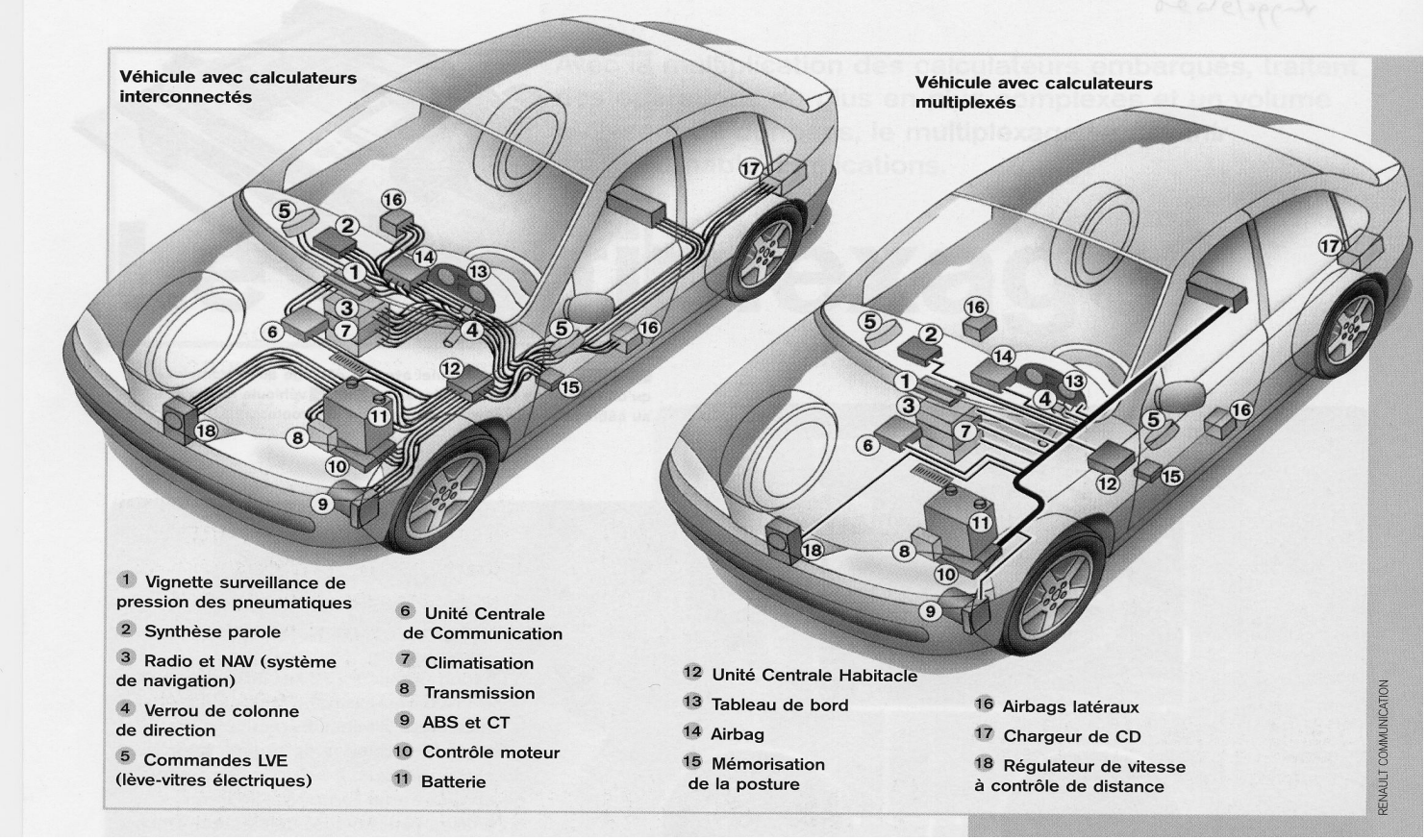
- Médical : génétique, imagerie médicale
- Physique : mécanique quantique, modélisation de plasmas, mécanique des fluides ...
- Industrie : simulations, CAO, chaîne de fabrication
- Aéronautique : contrôle aérien, dynamique des fluides ...
- Météorologie, géologie
- Chimie
- Nucléaire
- Serveurs de bases de données multimédias ...
- ...

# Pour répondre à quels besoins ?

- Puissance de calcul
- Problème géographiquement distribué
- Réduction de coûts
- Tolérance aux pannes

# Exemple dans l'automobile

1. **Application distribuée par nature**
2. **Réduction des coûts**
3. Puissance de calcul
4. Tolérance aux pannes




Source : Magazine RD Renault

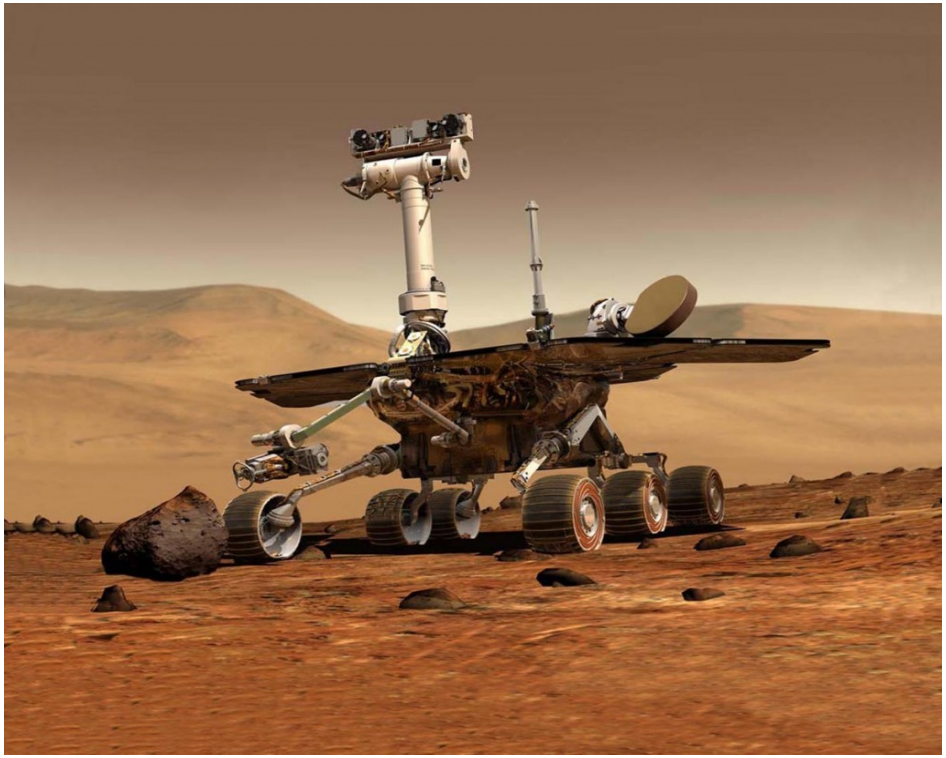
# Exemple dans l'aérospatial & l'aéronautique

1 **Nature distribuée de l'application**

2 **Tolérance aux pannes**

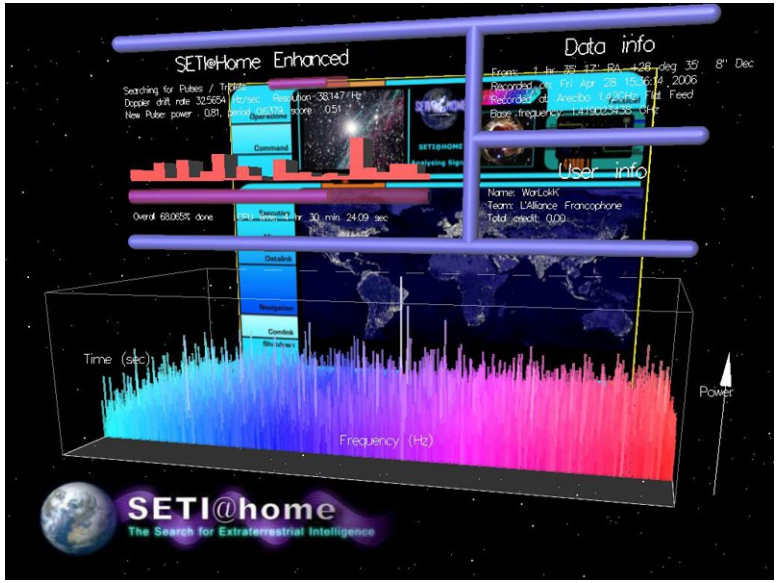
3 Puissance de calcul

4 Coûts

# Exemple dans l'astrophysique

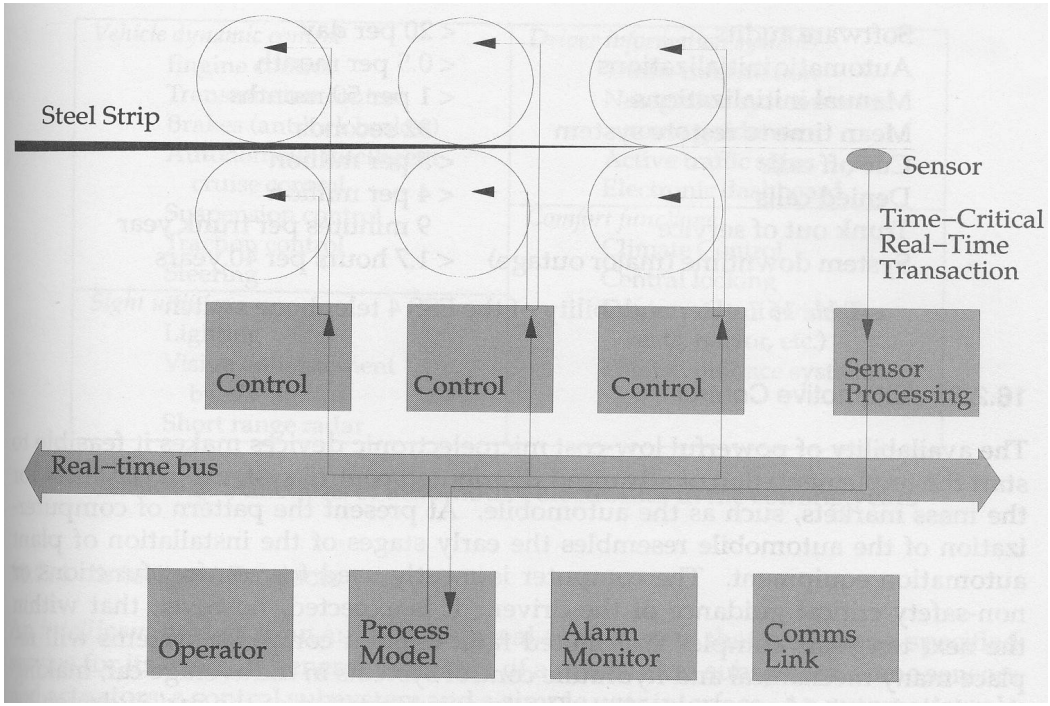
1. **Puissance de calcul**
2. nature distribuée de l'application
3. Réduction des coûts
4. Tolérance aux pannes

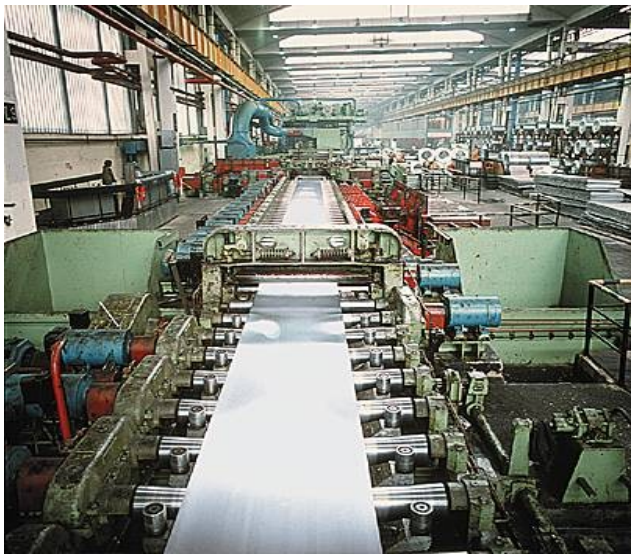




http://setiathome.ssl.berkeley.edu/

# Exemple dans l'industrie métallurgique

1. **nature distribuée de l'application**
2. Tolérance aux pannes
3. Réduction des coûts
4. Puissance de calcul



# Problèmatique des architectures //

- Problème d'algorithmique : répartir intelligemment le travail sur plusieurs unités de traitement
- Problème d'architecture matérielle : quelle architecture adopter pour interconnecter efficacement de nombreux processeurs et réaliser “au mieux” le travail

# I. Mise en oeuvre d'applications distribuées

1.Introduction

2. Architectures parallèles
   - Classif ication de Flynn
   - Classif ication selon type de communications
   - Classif ication selon réseau d'interconnexion
   - Routage

3. Limites du parallélisme

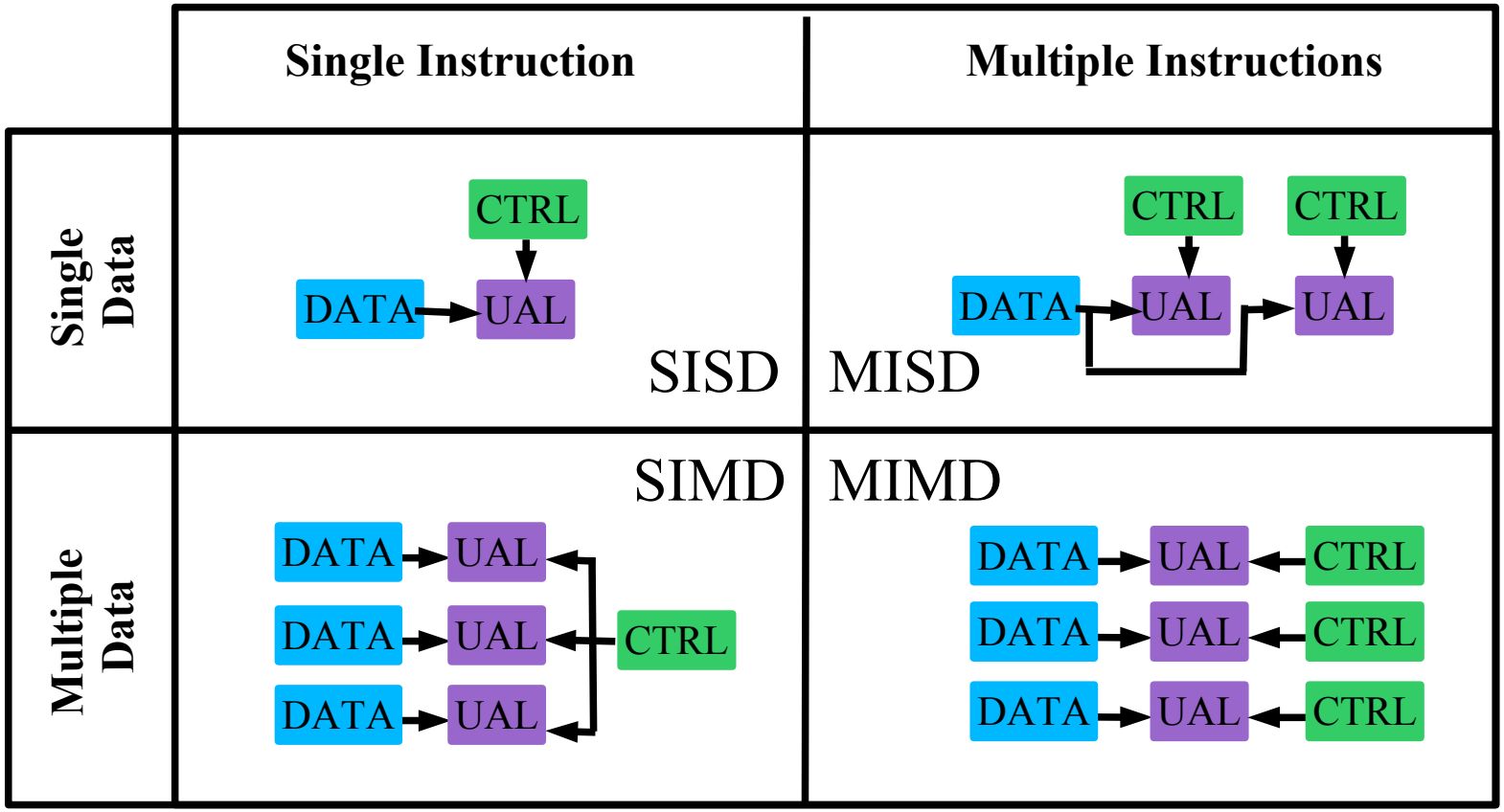
4. Parallélisation d'algorithmes

# Classif ication des architectures //

- Types de parallélisme : Flynn
- Types de communications
- Réseau d'interconnexion
- Caractéristiques architecturales particulières ....

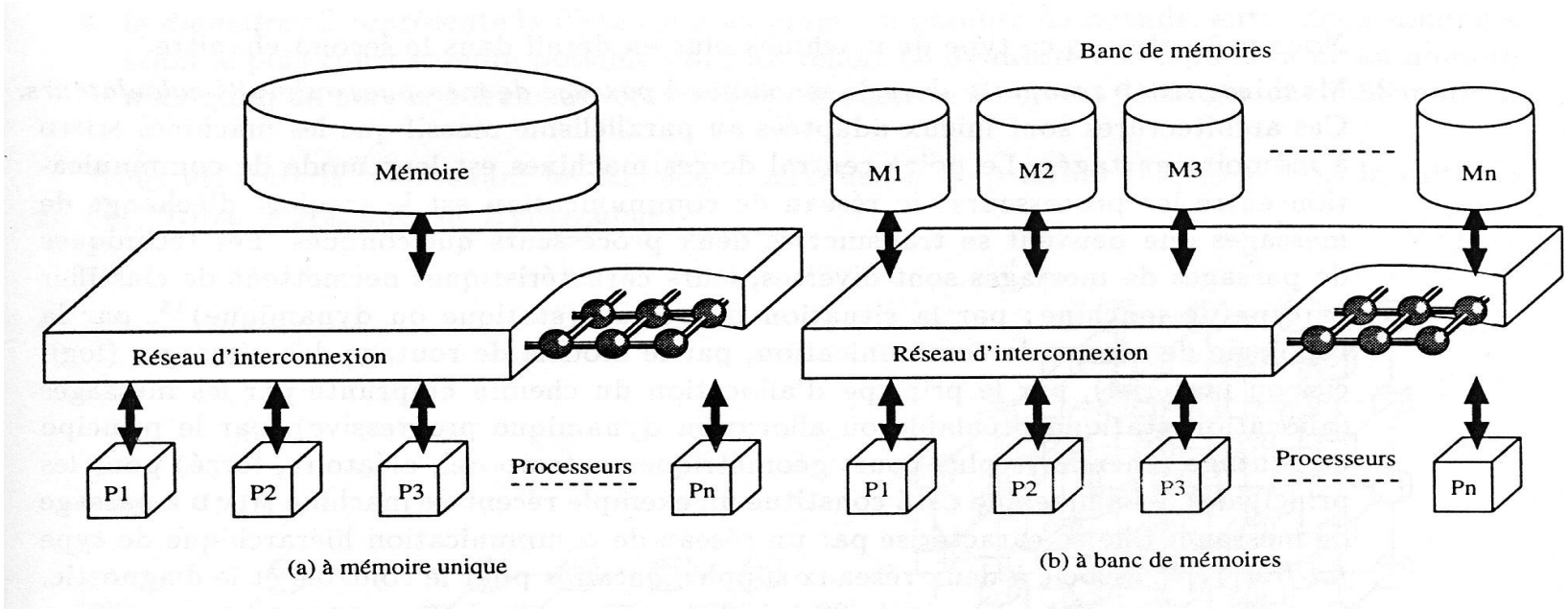
# Classif ication de Flynn

**4 catégories d'architecture**



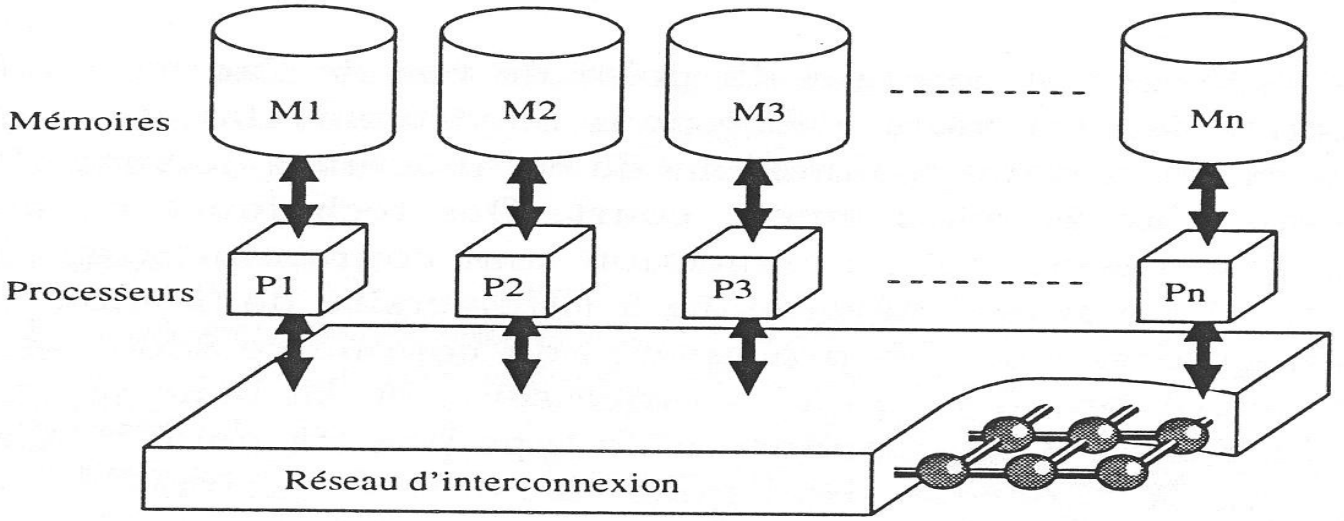
# Suivant le type de communication (1)

- Communication par partage de données
  - Machines à mémoire partagée
  - Réseau d'interconnexion multipoint RAM



# Suivant le type de communication (2)

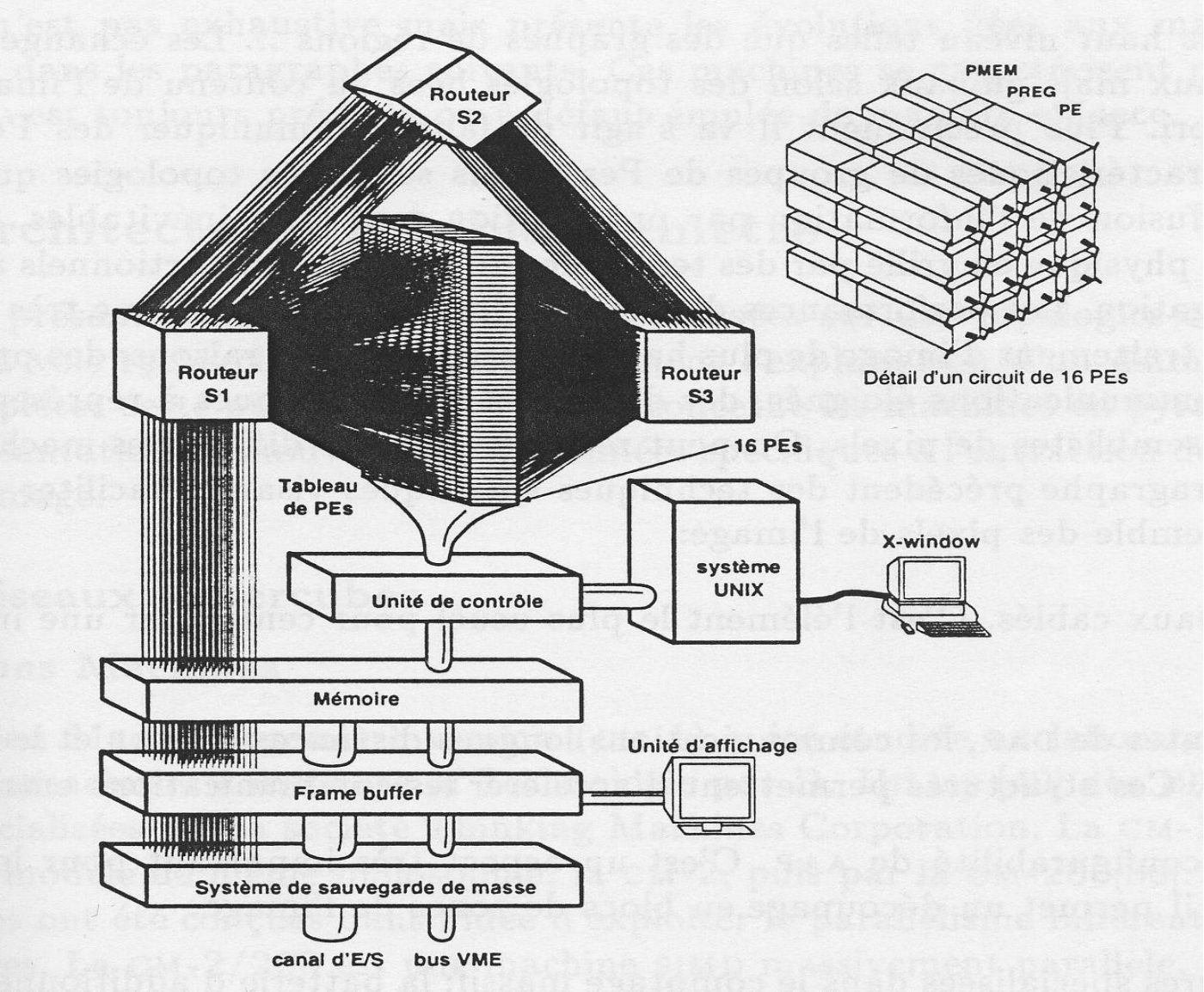
- Communication par passage de messages
  - Machines à mémoire distribuée
  - Réseau d'interconnexion SAM
    - Point à point (lien)
    - Multipoint (bus)



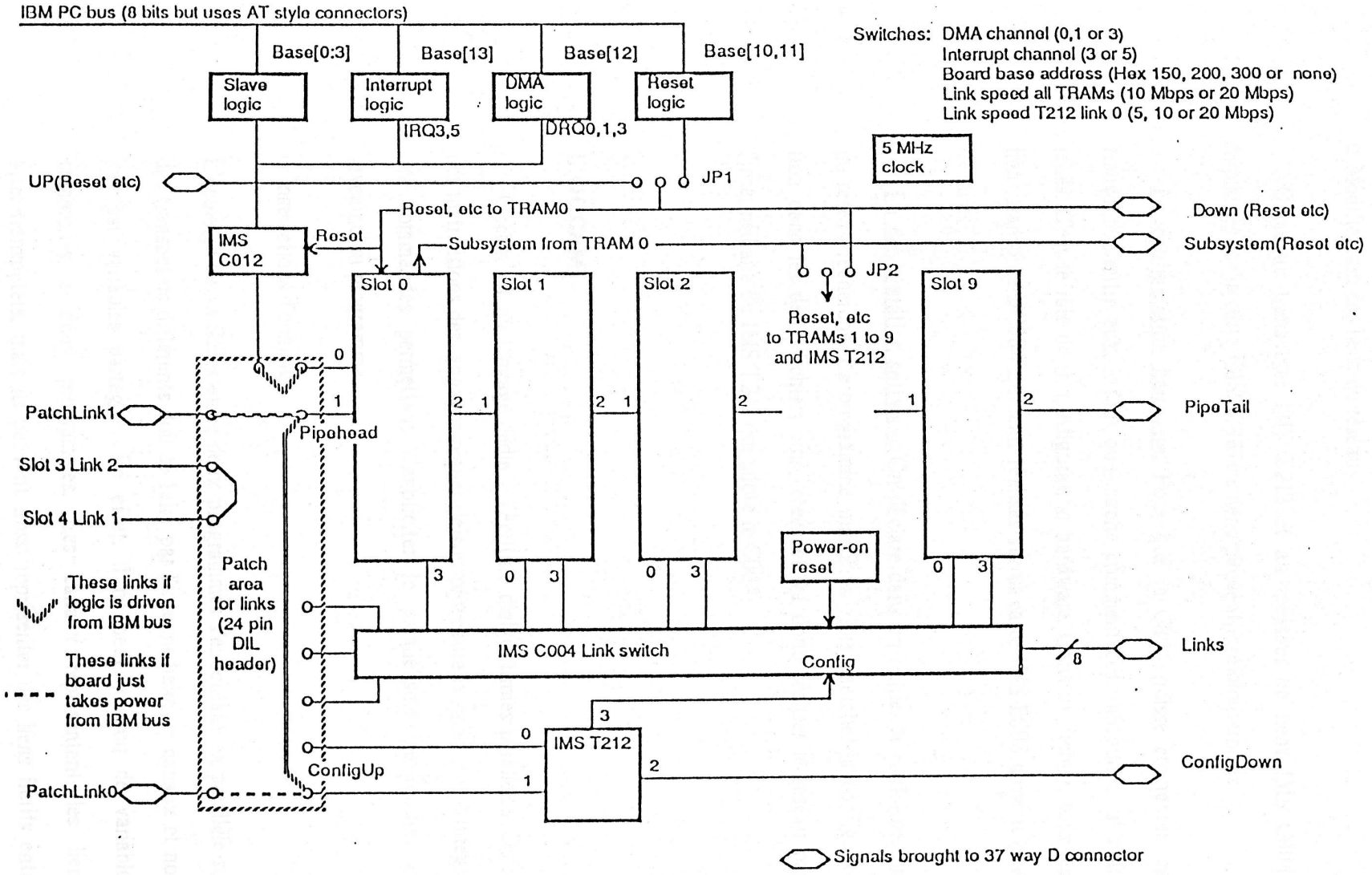
# Suivant le réseau d'interconnexion

- Réseau statique : connexions physiques réalisées à la conception de la machine
- Réseaux dynamiques : reconfiguration des connexions
  - Pendant l'exécution
  - Avant l'exécution

# Exemple : MASPAR



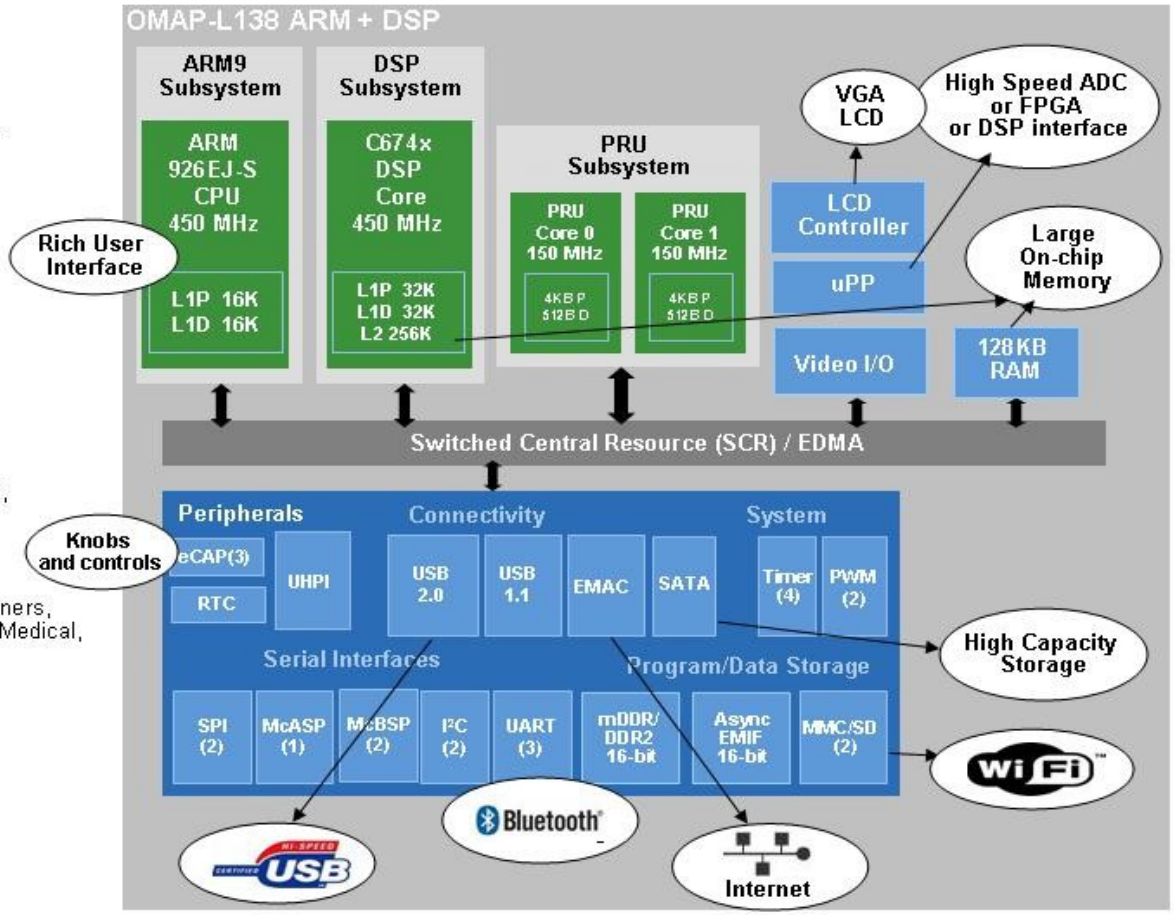
# Exemple : Transputer



# Exemple : OMAP

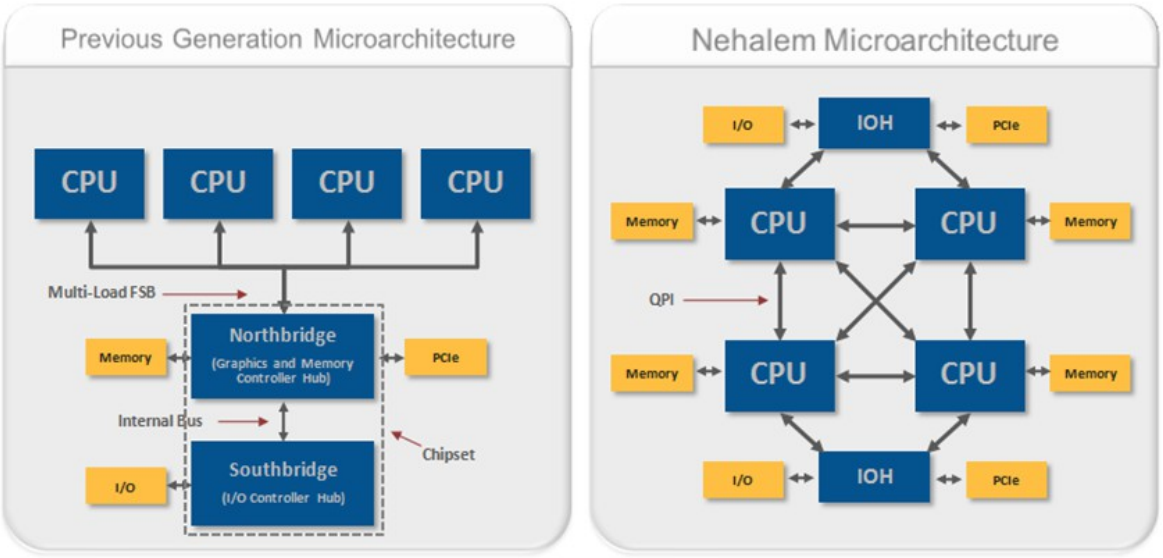
## OMAP-L138 (ARM9 + C674x DSP)

- **CPU Cores**
  - ARM926EJ-S™ (MPU) 450/375MHz
  - C674x DSP Core 450/375MHz
  - 2 PRU Cores upto 150 MHz each
- **Peripherals (1.8/ 3.3V IOs)**
  - 10/100 Ethernet MAC
  - EMIFA - SDRAM/NAND Flash
  - EMIFB – DDR (mDDR/DDR2)
  - Video Port I/F, SATA, uPP, LCDC
- **Package**
  - 13 x13mm nFBGA (0.65mm), 16x16mm BGA (0.8mm)
  - Pin to pin compatible with C6748/6/2, AM1808/6
- **Applications**
  - Power Protection Systems, Test & Measurement, SDR, Bar Code Scanners, Portable Communications, Portable Medical, Portable Audio

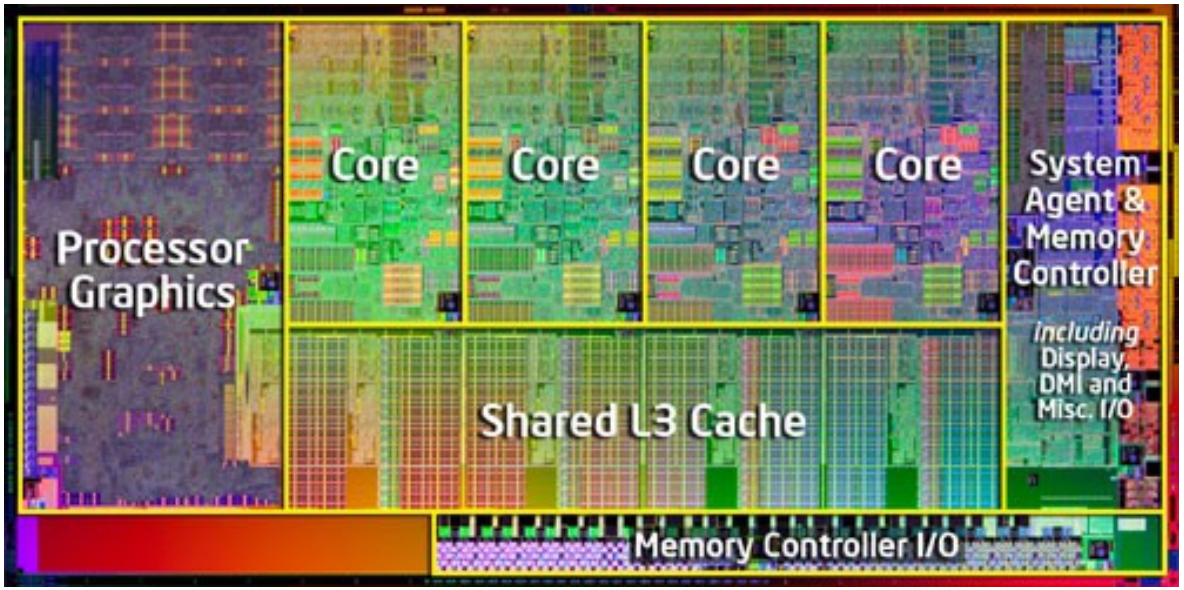



Source : Texas Instruments *www.ti.com*

# Exemple : Intel Core i7

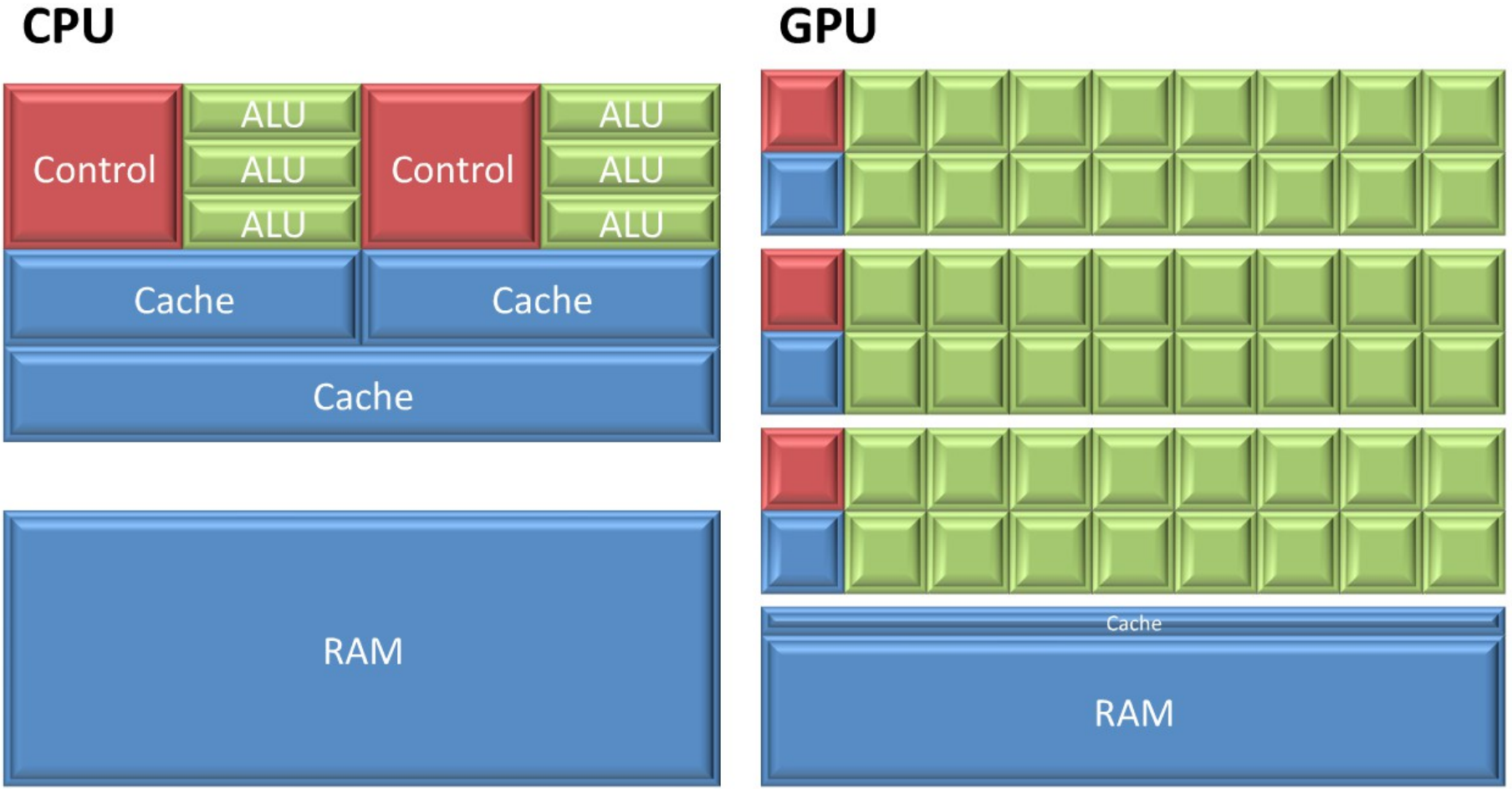



Source : National Instruments
*www.ni.com*




Source : *www.hardware.fr*

# Exemple : GPU




Généraliste, logique complexe, gros cache

Calcul Flottant, logique réduite, bcp de registres

Cours GPU gael.guennebaud:
*http://www.labri.fr/perso/guenneba/pghp_2015/Cours_PGHP_2015_01-IntroArchitectureGPU.pdf*

# Exemple : GPU NVIDIA

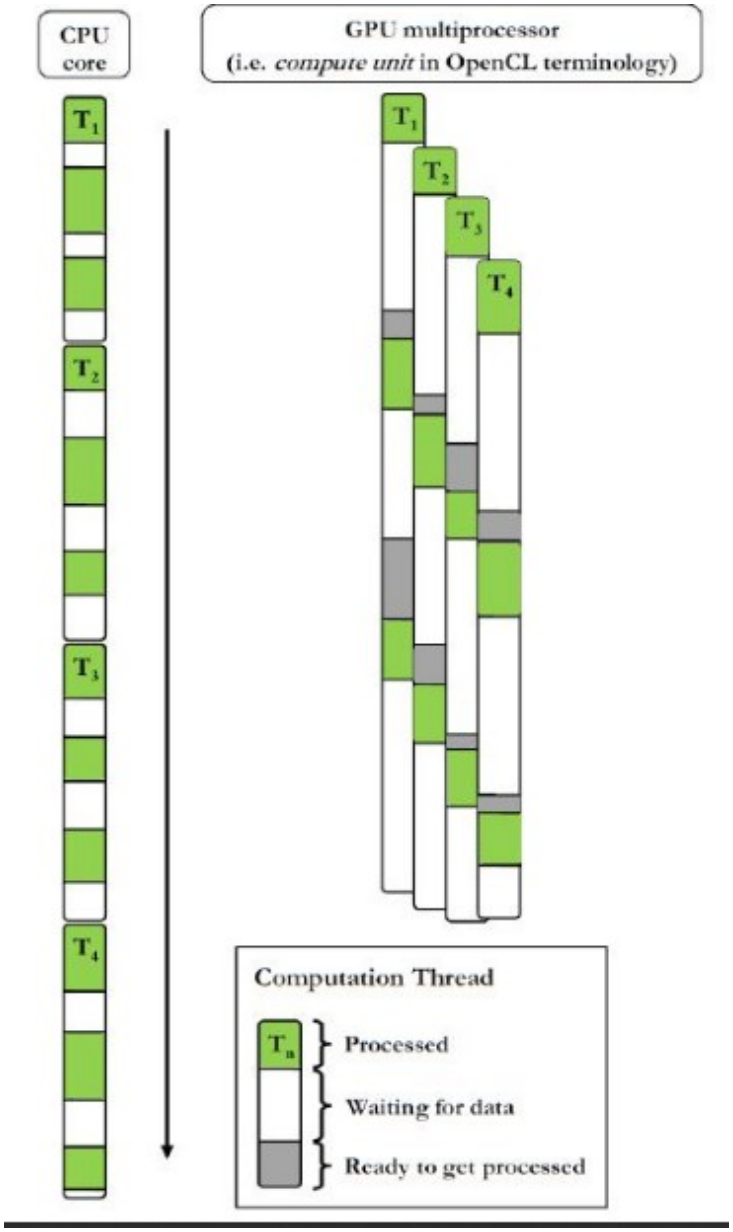
- Streaming Multiprocessors en cluster
- Chaque SM peut exécuter un programme différent MIMD


# Exemple : GPU

CPU : optimisé pour exécuter au plus vite une tâche

GPU : optimisé pour exécuter au plus vite un ensemble de tâches

**Exploitation parallélisme de données et parallélisme pipe-line**



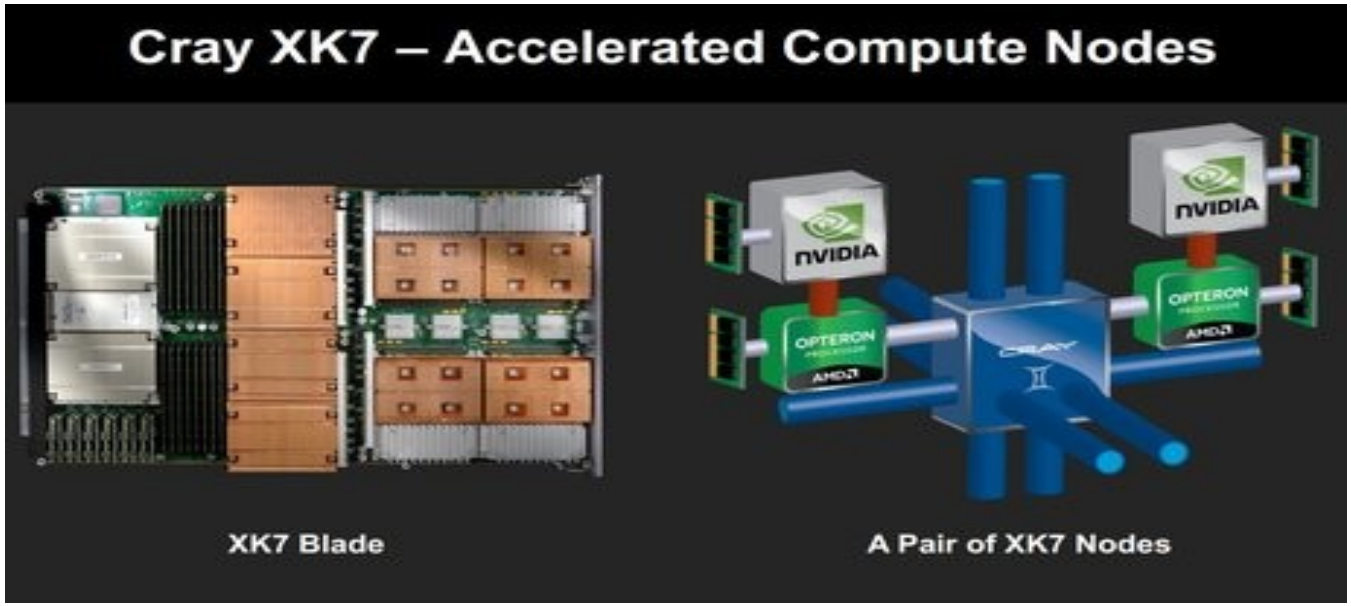
# Exemple : CRAY Titan (1)



www.olcf.ornl.gov/titan/

- Super ordinateur construit par Cray Inc. Pour le Laboratoire national d'Oak Ridge
- capable d'atteindre 27 PFLOPS en performance de pointe.
(1 petaFLOPS = $10^{15}$ opérations virgule flottante par seconde)
- Sa mémoire vive est de 710 TB3 (598 TB CPU et 112 TB GPU).
- Système d'exploitation : Cray Linux Environment (CLE).
- Puissance électrique : 8,2 MW.
- Coût : 97 millions de dollars.
- 404 $m^2$

# Exemple : CRAY Titan (2)



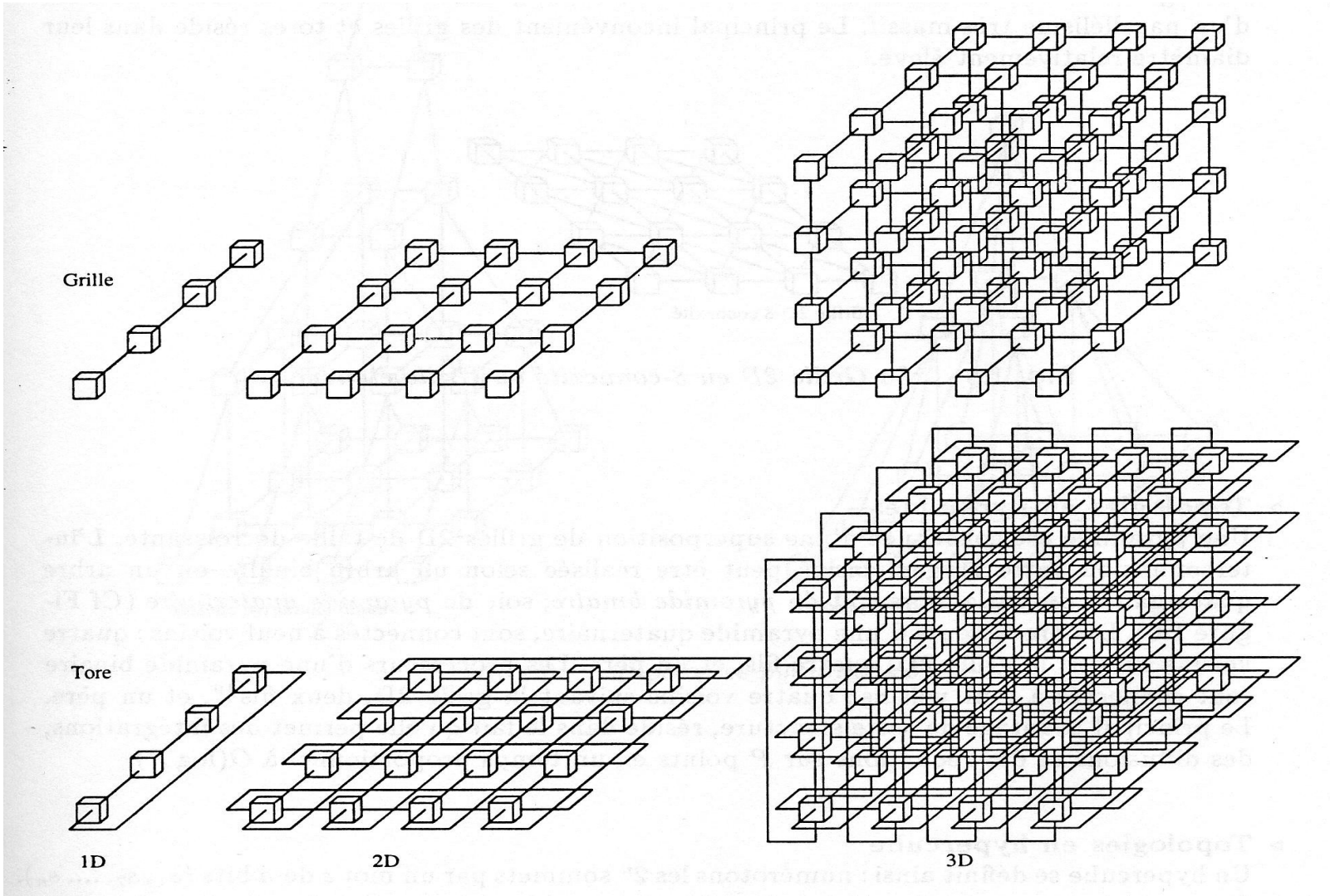

- ~200 armoires de 24 cartes contenant 4 nœuds de calcul, 1 CPU et 1 GPU par nœud
- CPU : AMD Opteron 6200 Interlagos series 16 core CPU,  16 ou 32 Go ECC DDR3 par CPU
- GPU : Nvidia Tesla K20 Kepler series, 5 ou 6 Go ECC DDR5 par GPU
- **=> 18688 CPU + 18688 GPU**

#5 au classement www.top500.org (#2 en 2016)

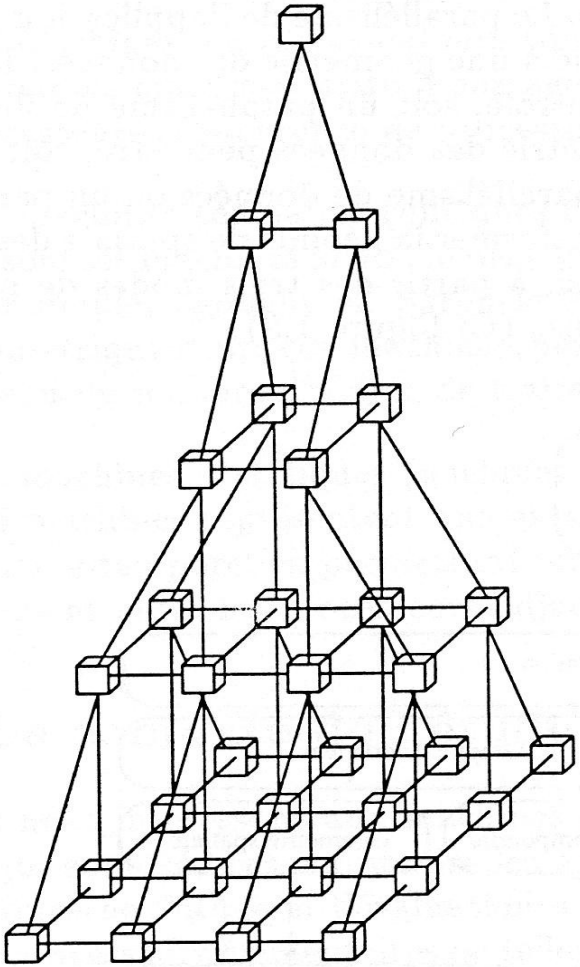
# Caractérisation des réseaux

- Diamètre : distance maximum entre deux processeurs quelconques selon le plus court chemin pour les relier
- Connectivité (ou degré) : nombre de voisins directement accessibles de chaque processeur
- Latence : temps de transit d'un signal à travers le réseau
- Bande passante : débit d'information qu'un processeur peut transmettre dans le réseau

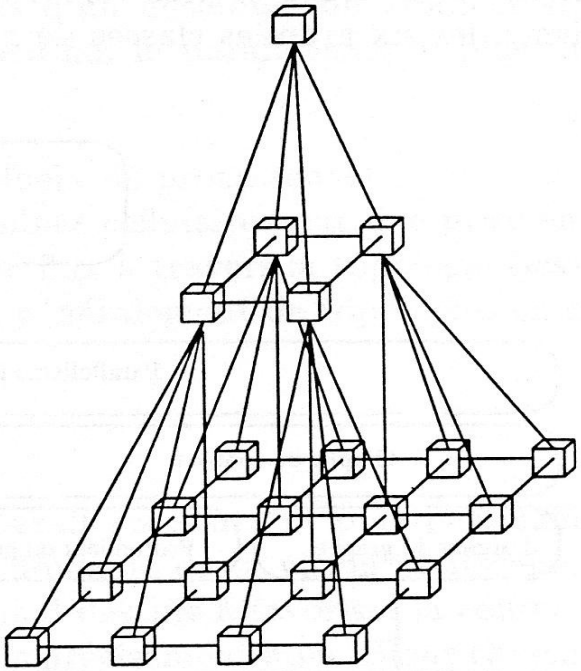
# Topologie de réseaux (1)



# Topologie de réseaux (2)

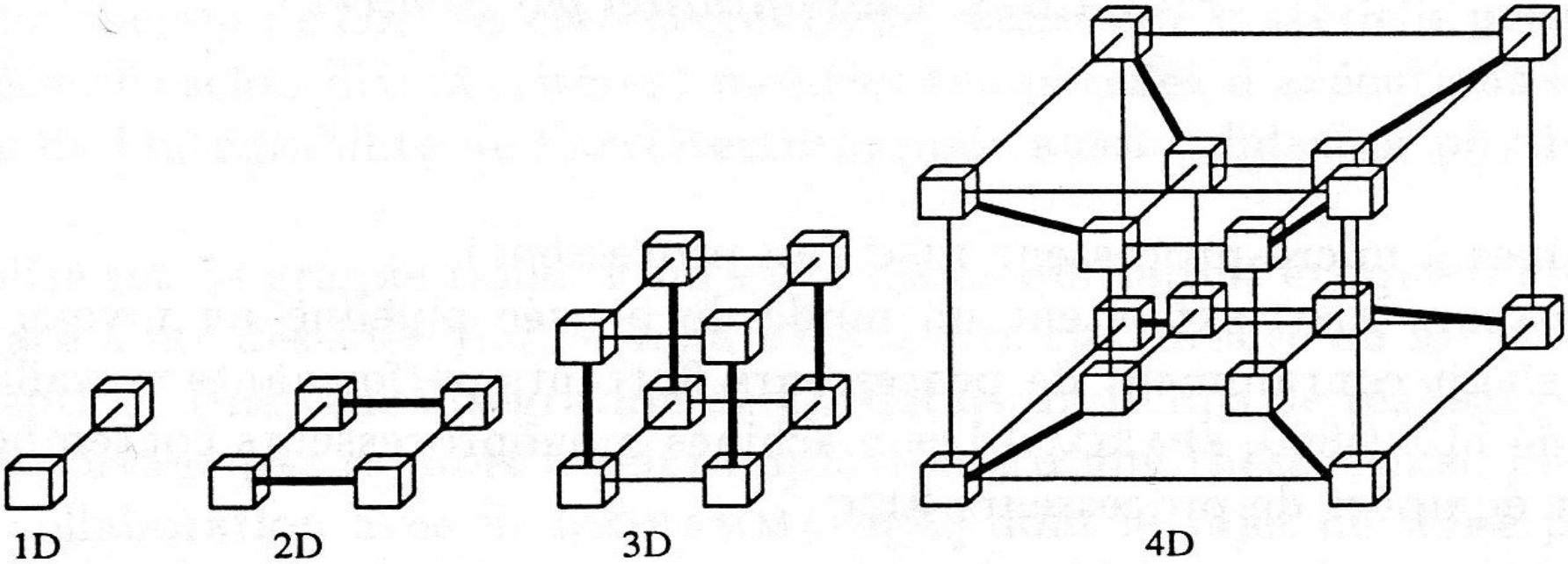


Réseau pyramidal binaire



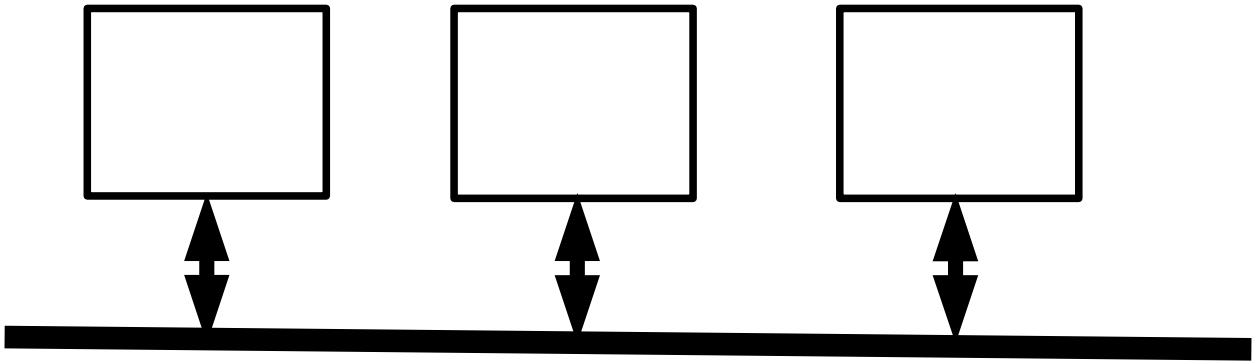
Réseau pyramidal quaternaire

# Topologie de réseaux (3)



# Topologie de réseaux (4)

- SAM multipoint (bus)
  - Connection possible de tous les processeurs
  - Nécessite un arbitre (matériel ou logiciel)



# Routage

- Communication entre processeurs non directement connectés
  - Choix du chemin
  - Gestion des conf łts
  - Garantir ordre
- Matériel ou logiciel

# Routage déterministe

- Association d'un seul chemin à chaque couple de processeur (x,y)
- Routage (x,y) peut être différent du routage (y,x)
- Avantage : chemin le plus court
- Désavantage : conf lts de routes

# Routage non déterministe

- Glouton : dès qu'un chemin approche de la cible , on le prend
- Aléatoire : choix aléatoire de la prochaine route
- Forcé : choix du chemin le plus court, s'il est occupé, choix d'un autre chemin

# Techniques de routage

- Commutation de messages (Store & Forward)
- Commutation de circuits
- Commutation de paquets
- Routage Wormhole

# Machines hétérogènes

- Processeurs
  - Dédiés au parallélisme
  - Microcontroleurs
  - DSP
  - ...
- Réseaux d'interconnexions
  - Liens point à point
  - Bus SAM
  - Bus RAM

# I. Mise en oeuvre d'applications distribuées

1.Introduction

2. Architectures parallèles

3. Limites du parallélisme

4. Implantation d'algorithmes sur architecture //

# Performances

Facteur d'accélération : Mesure globale de la qualité d'une implantation parallèle d'un algorithme

$$S_p = \frac{t_1}{t_p}$$

$t_1$ : temps d'exécution séquentiel

$t_p$ : temps d'exécution sur une architecture à $p$ processeurs

*Théorème :* $\forall\ p, 1 \leq S_p \leq p$

Efficacité : Mesure du taux moyen d'utilisation des processeurs = rendement

$$E_p = \frac{S_p}{p}$$

$\forall\ p, 0 \leq E_p \leq 1$

# Loi d'Amdahl

Si on considère $t_p = \alpha t_1 + \frac{(1-\alpha) t_1}{p}$ le temps d'exécution sur p processeurs

Alors

$$\Rightarrow \quad S_p = \frac{1}{\alpha + \frac{(1-\alpha)}{p}} \leqslant \frac{1}{\alpha} \qquad (0 \leqslant \alpha \leqslant 1)$$

- Le facteur d'accélération est quelque soit p inférieur à la proportion de code séquentiel. Ex: si 10% du programme est séquentiel alors $S_p < 10 \quad \forall \; p$

➡ Le facteur d'accélération est borné par une borne indépendante du nombre de processeurs et de la structure de la machine

# Limites du //

- Lois d'Amdahl(1967), de Lee(1980), de Minsky(1990), ...
- Table de Stone (1973)

| $S_P$ | Exemples |
|---|---|
| $\alpha p$ | Calculs matriciels<br>Discrétisation |
| $\frac{\alpha p}{\log_2(p)}$ | Tris<br>Systèmes tridiagonaux<br>Récurrences linéaires<br>Evaluation de polynômes |
| $\alpha \log_2(p)$ | Recherche d'un élément<br>dans un ensemble |
| $\alpha$ | Certaines récurrences non linéaires<br>Compilation de programmes |

Dans ce tableau, $\alpha$ est un nombre positif plus petit que 1, et dépendant de la machine.

# Surcoût lié à la parallélisation (overhead)

- Communications
- Attentes de synchronisation
- Modifications des algorithmes dues à la //
- Coordination par l'OS (ressources partagées)

# Exemple

- Construction de la pyramide des ages d'une population
  - $N$ nombres entiers $0<N<M$
  - $N$ données réparties sur $p$ processeurs: on suppose $M$ et $N$ multiples de $p$
  - Le processeur $P_i$ $(0 \leqslant i \leqslant p-1)$, construit le morceau d'histogramme correspondant aux valeurs comprises entre i(M/p) et (i+1)(M/p)
  - Topologie anneau

# Exemple (2)

- Algorithme

  (a) En // chaque $P_i$ examine le bloc de *N/p* données qu'il a reçu et construit sa portion locale de l'histogramme

  (b) En // chaque $P_i$ envoie à son voisin de gauche son bloc de *N/p* valeurs et reçoit de son voisin de droite le bloc suivant

  (c) On répète *p-1* fois (a) et (b)

# Exemple (3)

- Overhead étape (a) :
  - soit ! la fraction des valeurs comprises entre *i(M/p)* et *(i+1)(M/p)*,
  - *T* temps pour placer une donnée dans l'histogramme,
  - *t* temps pour seulement tester l'intervalle d'appartenance d'une valeur

$$temps_{(a)} = \alpha \frac{N}{p} T + \boxed{(1-\alpha)\frac{N}{p} t}$$

# Exemple(4)

- Overhead étape (b) :
  - Soit C le temps de communication pour transmettre une donnée

$$temps_{(b)} = (N/p)C$$

  - Remarque : il suffirait en principe d'envoyer au Pi voisin seulement les données non placées
    - Algo + compliqué
    - Pb avec les machines SIMD où les messages qui circulent en // doivent être de même longueur

# Exemple(5)

- Temps total d'exécution (on néglige t)

$$T_{par} = \alpha NT + \frac{p-1}{p} NC$$

- Cas idéal ! $= 1/p$

$$T_{par} = \frac{NT}{p} + \frac{p-1}{p} N C$$

- En séquentiel

$$T_{seq} = N T$$

- Accélération

$$S_p = \frac{p}{1 + \frac{C}{T}(p-1)} \leqslant \frac{p}{p-1} \frac{T}{C}$$

# Exemple

- Intéressant si $T >> C$ et $t$ négligeable
- Overhead lié à la parallélisation :
  - Découpage en $N/p$ blocs de données
  - Chargement initial des Pi processeurs
  - Reconstruction de l'histogramme

# I. Mise en oeuvre d'applications distribuées

1.Introduction

2. Architectures parallèles

3. Limites du parallélisme

4. Implantation d'algorithme sur architecture //

- Principe
- Ordonnancement
- Algorithmes de placement et d'ordonnancement

# 2 APPROCHES COMPLEMENTAIRES

- Etudier au mieux la parallélisation d'un algorithme existant:
  - Choix d'une architecture adaptée (MIMD,MISD,SIMD)
  - Utiliser au mieux l'architecture : problème de placement et d'ordonnancement
- Imaginer une nouvelle version de l'algorithme adaptée à la machine parallèle
  - Modifier l'algorithme afin de réduire la partie séquentielle
  - Modifier l'algorithme pour exploiter au mieux le parallélisme offert par l'architecture (contrôle, donnée, pipe-line)

# Principe de parallélisation

a) Découpage en tâches

b) Construction d'un graphe de précédences

c) Affectation des tâches aux processeurs en respectant les contraintes de précédence : **placement et ordonnancement**

➡ **Problème d'optimisation**

**Maximiser *Sp* et *Ep*, obtenir $t_{opt}$ , minimiser *p***

# Découpage en tâches

- Tâche = unité de traitement indivisible caractérisée par son comportement extérieur :
  - Entrées/Sorties
  - Fonction : suite indivisible d'instructions
  - Temps d'exécution
- Pas de préemption
- Temps d'exécution maximal (branchements)
- Granularité du découpage

# Granularité : exemple

Produit de 2 matrices A et B de taille (n,n) :

- Granularité fine

$$C(i,j) \leftarrow C(i,j) + A(i,k) * B(k,j)$$

- Granularité moyenne

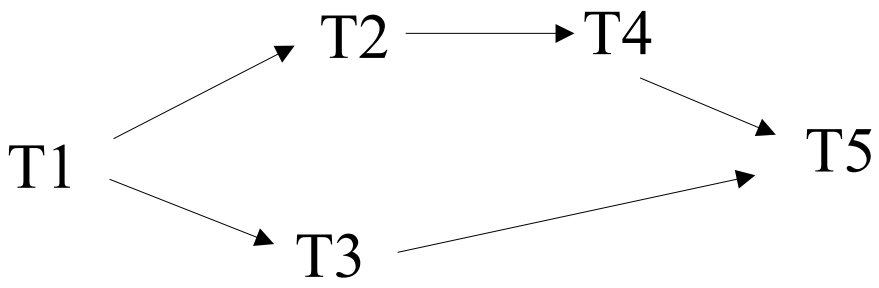
$$\begin{array}{l} pour\ k \leftarrow 1\ jusqu'à\ n \\ \quad C(i,j) \leftarrow C(i,j) + A(i,k) * B(k,j) \end{array}$$

- Granularité grossière

$$\begin{array}{l} pour\ j \leftarrow 1\ jusqu'à\ n \\ \quad pour\ k \leftarrow 1\ jusqu'à\ n \\ \quad\quad C(i,j) \leftarrow C(i,j) + A(i,k) * B(k,j) \end{array}$$

# Graphe de précédence

- Sommet = tâche
- Arc = relation de précédence (s'exécute après)
- Relation de précédence : définie par relation entrées sorties entre les tâches
- Pas de cycle => DAG (Directed Acyclic graph)
- !! variables globales



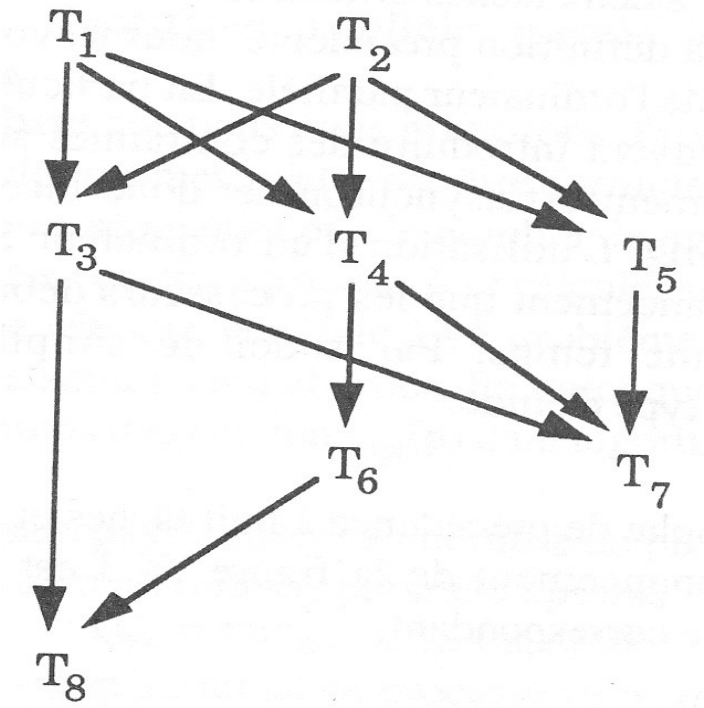
# Ordonnancement

- **« QUAND la tâche doit s'exécuter » :** allocation temporelle d'une ressource (processeur, média de com) à une tâche
- Hypothèses simplificatrices :Temps d'exécution unitaire des tâches, architecture MIMD, mémoire partagée, temps d'accès mémoire négligeable, pas de conflit d'accès
- Hauteur du graphe H(G) : nombres de tâches du plus long chemin
- But : minimiser le temps d'exécution du graphe en respectant les précédences

# Décomposition en niveaux

- Décomposition par prédécesseurs
  - Niveau 1 : constitué des tâches n'ayant pas de prédécesseur
  - Niveau k : constitué des tâches dont tous les prédécesseurs sont dans des niveaux supérieurs
  - Niveau H(G) : constitué des tâches sans successeurs
- Décomposition par successeurs
  - Niveau H(G) : tâches sans successeur
  - Niveau k : tâches dont tous les successeurs sont dans des niveaux supérieurs
  - Niveau 1 : tâches sans prédécesseurs
- Largeur de décomposition L(G) = nombre de processeurs

# Exemple de décomposition




- Par prédécesseurs :
  - Niveau(1) = {T1,T2},
  - N(2) = {T3,T4,T5},
  - N(3)= {T6,T7},
  - N(4) = {T8}
- Par successeurs :
  - N(1) = {T1,T2},
  - N(2) = {T4},
  - N(3) = {T3,T5,T6},
  - N(4)={T7,T8}

# Optimisations

## Temps de calcul optimal $t_{opt}$

*théorème :* avec un nombre illimité de processeurs,
topt=H(G)=chemin critique du graphe de précédences

## Nombre minimal de processeurs $p_{opt}$ permettant d'exécuter en $t_{opt}$

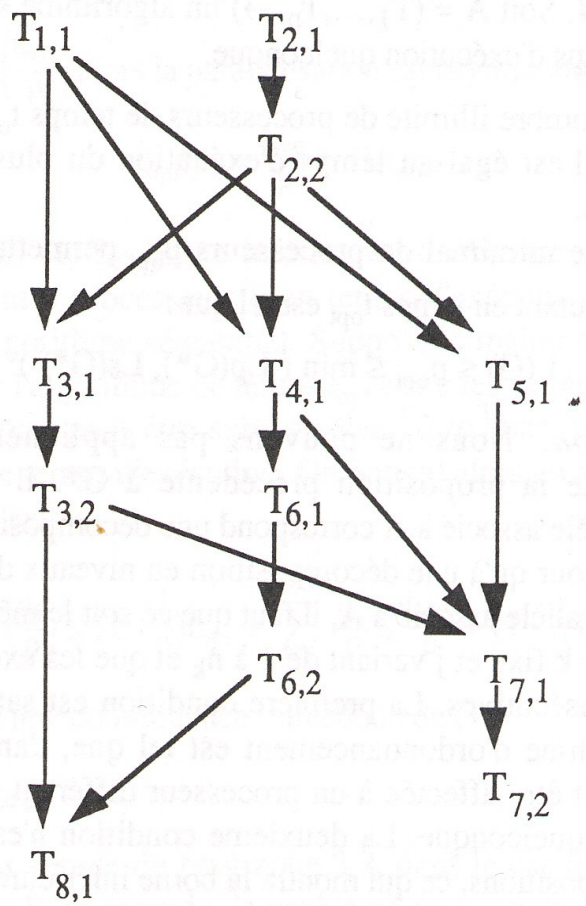
$$p_{opt} \leqslant min(L_{pred}(G), L_{succ}(G))$$

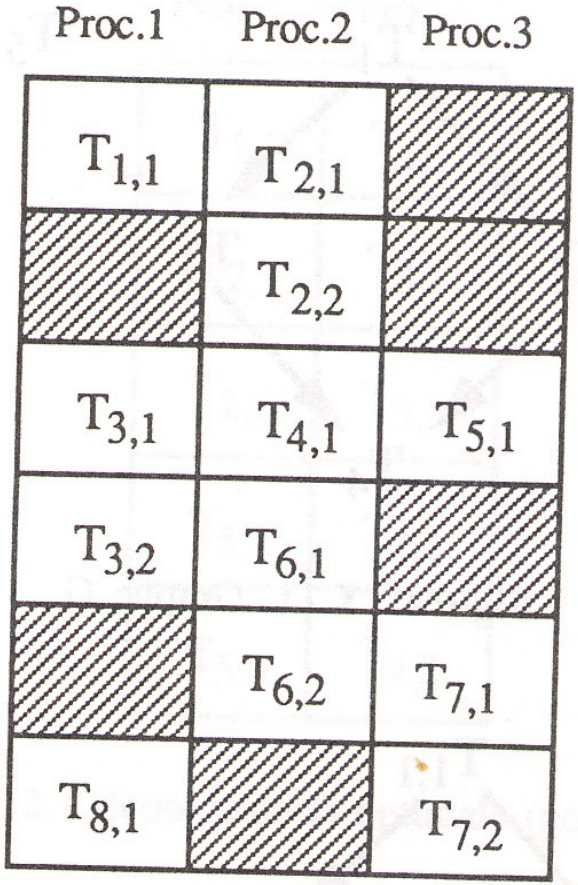
= réduire au minimum la largeur de décomposition sans en augmenter la hauteur

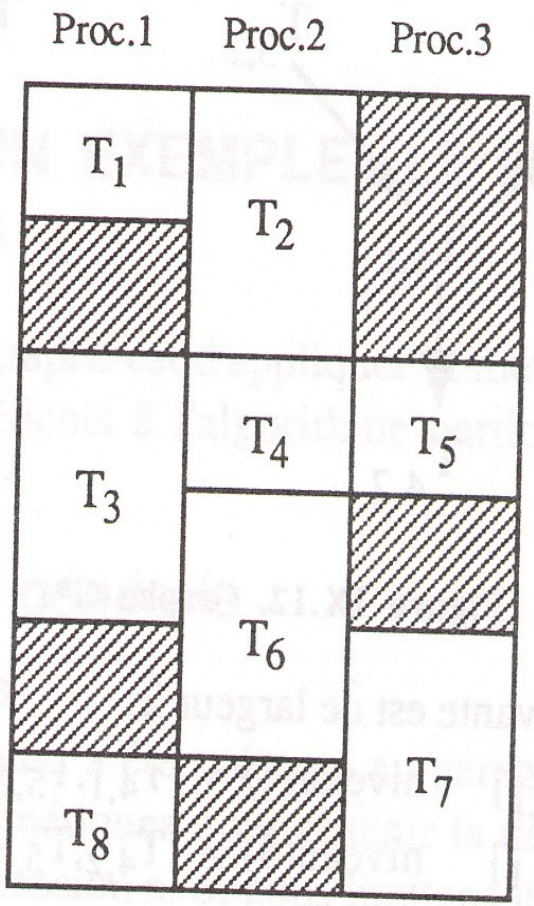
# Temps d'exécution non unitaires

- Découpage des tâches en sous-tâches unitaires
- Construction du graphe de précédence
- Décomposition avec contrainte de placement sur chaque morceaux d'une même tâche
- Reconstitution des tâches non unitaires

# Exemple


T1,1
T2,1
T2,2
T3,1
T4,1
T5,1
T3,2
T6,1
T6,2
T7,1
T7,2
T8,1


Proc.1
Proc.2
Proc.3
T1,1
T2,1
T2,2
T3,1
T4,1
T5,1
T3,2
T6,1
T6,2
T7,1
T8,1
T7,2


Proc.1
Proc.2
Proc.3
T1
T2
T3
T4
T5
T6
T7
T8

# Placement et ordonnancement optimal

- Placement : « **Où les tâches doivent être exécutées** », allocation spatiale d'une ressource (proc/média de com) à une tâche ou à une communication
- Modèles
  - Algorithme
  - Architecture
- Problème NP-complet
  - Solutions optimales
  - Solutions approchées

# Modèles pour le placement et l'ordonnancement

- Qualité du placement et de l'ordonnancement dépend de la finesse des modèles
- Algorithme : graphe de précédence
  - Tâche (sommet): temps d'exécution, coût mémoire, ...
  - Dépendance de donnée (arc) : quantité d'information à échanger entre les tâches
- Architecture : graphe
  - Processeur (sommet) : mémoire cache, calcul flottant,..
  - Média de communication (arc) : débit, protocole, arbitrage d'accès, ...
  - Ressources partagées, // calcul/communication

# Placement et ordonnancement optimal

Problème de minimisation d'une fonction de coût :

- Temps d'exécution total, équilibrage de charges de calcul, de com,...
- Contraintes : précédences, mémoire, E/S ...

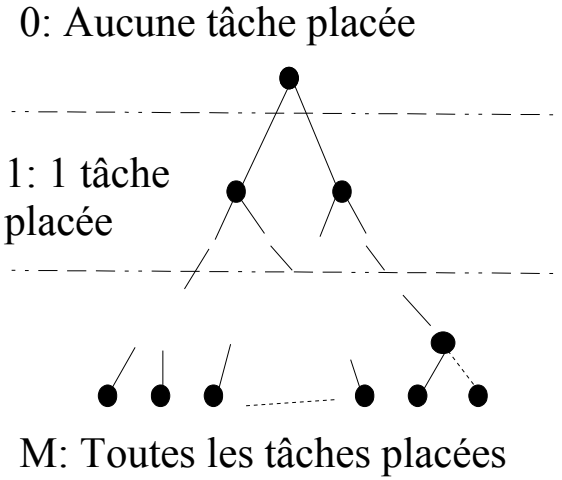
- Statique : avant exécution
- Dynamique : pendant l'exécution (migration de tâches, reconfiguration du réseau,...)

# Placement et ordonnancement statique

- Ploblème NP-difficile
- Méthodes exactes : exploration combinatoire, **solution optimale**
- Méthodes approchées : **solution sous-optimale**
  - Algorithmes gloutons
  - Algorithmes itératifs
  - Algorithmes Mixtes

# Méthodes exactes

- Exploration de toutes les solutions possibles




- En pratique on cherche à éliminer des branches
  - Branch & Bound :fonction d'estimation du coût de placement, on supprime les branches dont le coût est supérieur à une branche completement développée
  - Théorie des graphes

# Méthodes approchées :algorithmes gloutons

Principe

- le placement d'une tâche i se fait sous certains critères à partir du placement réalisés sur les (i-1) premières
- Pas de remise en cause des choix déjà effectués

Intérêt

- Permet de trouver très rapidement une solution grossière

# Algorithmes gloutons (2)

Algorithme de placement modulo

- Distribution équitable des tâches sur les processeurs
- Algorithme
  - `Numéroter les tâches et les processeurs par une méthode quelconque`
  - `tâche i placée sur le processeur (i modulo p)`
- Simplicité : mise en oeuvre fréquente dans les OS
- Valable si Tcom << Tcalcul

# Algorithmes gloutons (2)

Algorithme LPT (largest processing time first)

- Algorithme
  - `Placer la tâche 1 sur le premier processeur qui possède assez de mémoire`
  - `Tâche i placée sur le processeur possédant la plus petite charge de calcul`
- Simplicité
- Amélioration du modulo
- Valable si Tcom<<Tcalcul

# Algorithmes gloutons (3)

à critère structurel

- Algorithme
  - `Placer la tâche possédant le + grand nombre d'entrées-sorties sur le premier proc`
  - `Tâche i placée sur le processeur possedant la plus petite charge globale (calcul + communication)`
- Prise en compte des communications

# Algorithmes gloutons (4)

Algorithme glouton amical

- Réduire les communications
- Algorithme
  - Placement arbitraire de la première tâche
  - Choix de la tâche i qui communique le plus avec les tâches déjà placées
  - Choix du placement en minimisant une fonction de coût (par exemple minimiser les communications)

# Algorithmes gloutons (5)

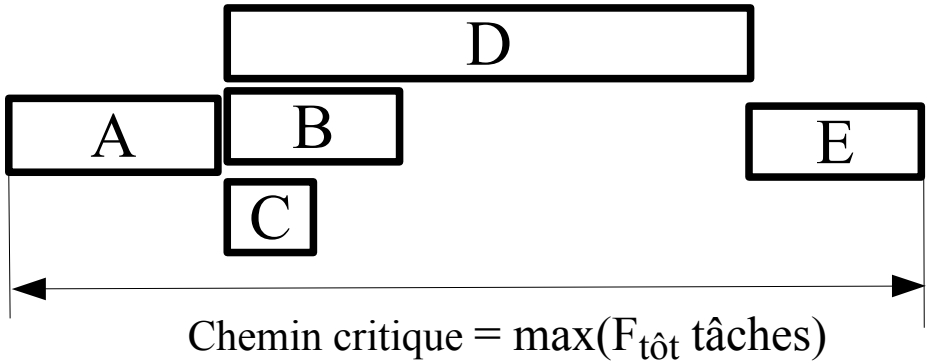
Algorithme à critère quantitatif

- Réduire les communications
- Algorithme
  - Choix de la tâche la plus coûteuse (coût de calcul + coût de com)
  - Choix du placement en considérant que les coûts des coms interne à un proc son nuls.

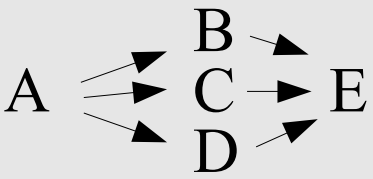
.....

# Algorithme glouton: exemple LLF(1)

- Dates exécution au plus tôt

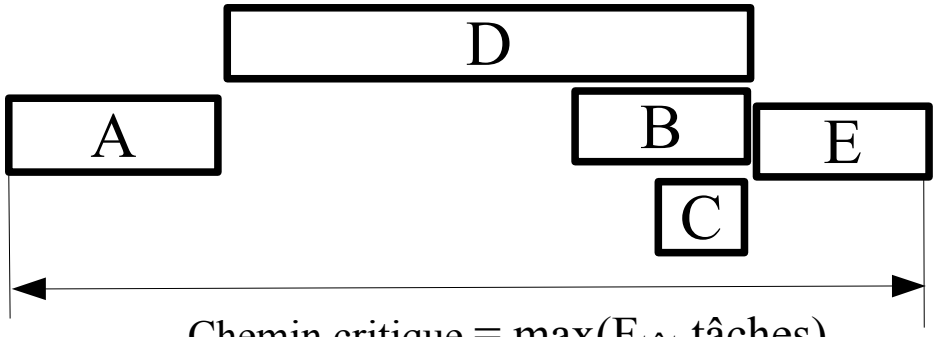



DAG :



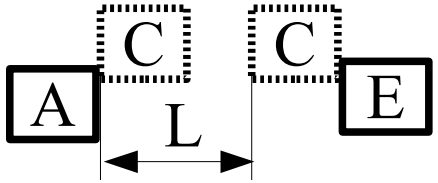

- $D_{tôt}$ :dates de début d'exécution au plus tôt
  - ! $F_{tôt} = D_{tôt} + D_{exec}$
  - ! $D_{tôt} = \max(F_{tôt}$ prédécesseurs)

- Dates exécution au plus tard




- Ftard : dates de fin exécution au plus tard
  - ! $D_{tard} = F_{tard} - D_{exec}$
  - ! $F_{tard} = \min(D_{tard}$ successeurs)

- Laxité = $F_{tard} - F_{tôt}$ (ou $D_{tard}$-$D_{tôt}$)



# Algorithme glouton: exemple LLF (2)

a) Calcul dates au plus tôt et chemin critique

- ! décomposition par niveau prédécesseurs

b) Calcul dates au plus tard et Laxité

- ! décomposition par niveau successeurs

c) Tant que toutes les tâches ne sont pas placées et ordonnancées

- Ordonnancement de la tâche de plus faible laxité dont les prédécesseurs sont déjà placés
- Choix du placement minimisant l'allongement du chemin critique
- Mise à jour des dates et laxité

# Algorithmes itératifs

- Démarrage d'une solution initiale complète
- Amélioration itérative de cette solution
  - Permutations de tâches en ne retenant que les permutations qui améliorent une fonction de coût
  - Aléatoire nécessaire de temps en temps pour pour trouver des solutions meilleurs
- Recuit simulé, Réseaux de neurones, algorithmes génétiques

# Algorithmes mixtes

- Intermédiaire entre glouton et exact
- Calcul d'une solution approchée
- Pas d'exploration de sous-ensembles de solution moins intéressante que la solution approchée

# Placement et ordonnancement: conclusion

- Placement et ordonnancement liés
- Solution optimale difficile à atteindre

! Architectures hétérogènes

! Modèles (// calcul/communication, mémoire ...)

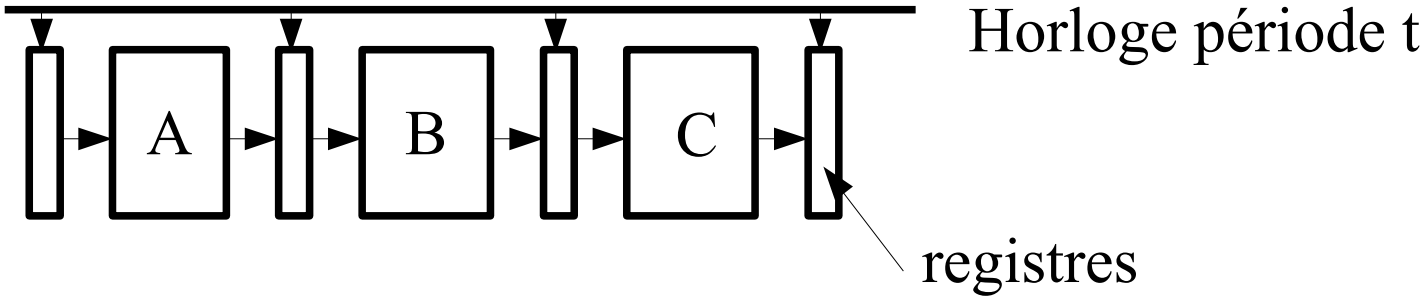
# Implantation parallèle d'un algorithme

- Réduire au maximum la structure séquentielle d'un algorithme
- Choix d'un grain adapté
- Exploiter le // de données, contrôle ou pipe-line (cf. famille Dupont)
- Essayer autant que possible d'obtenir une bonne répartition des charges sur les ressources matérielles

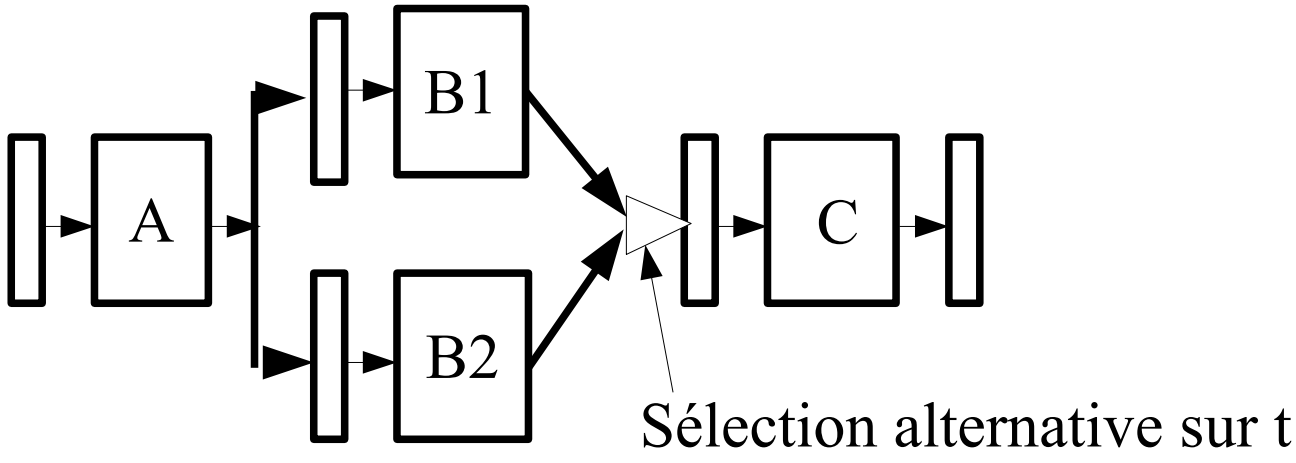
➔ **Méthodologies, savoir faire, règles empiriques, essais ...**

# Exploitation matérielle du // pipe-line

- Etages à durées identiques

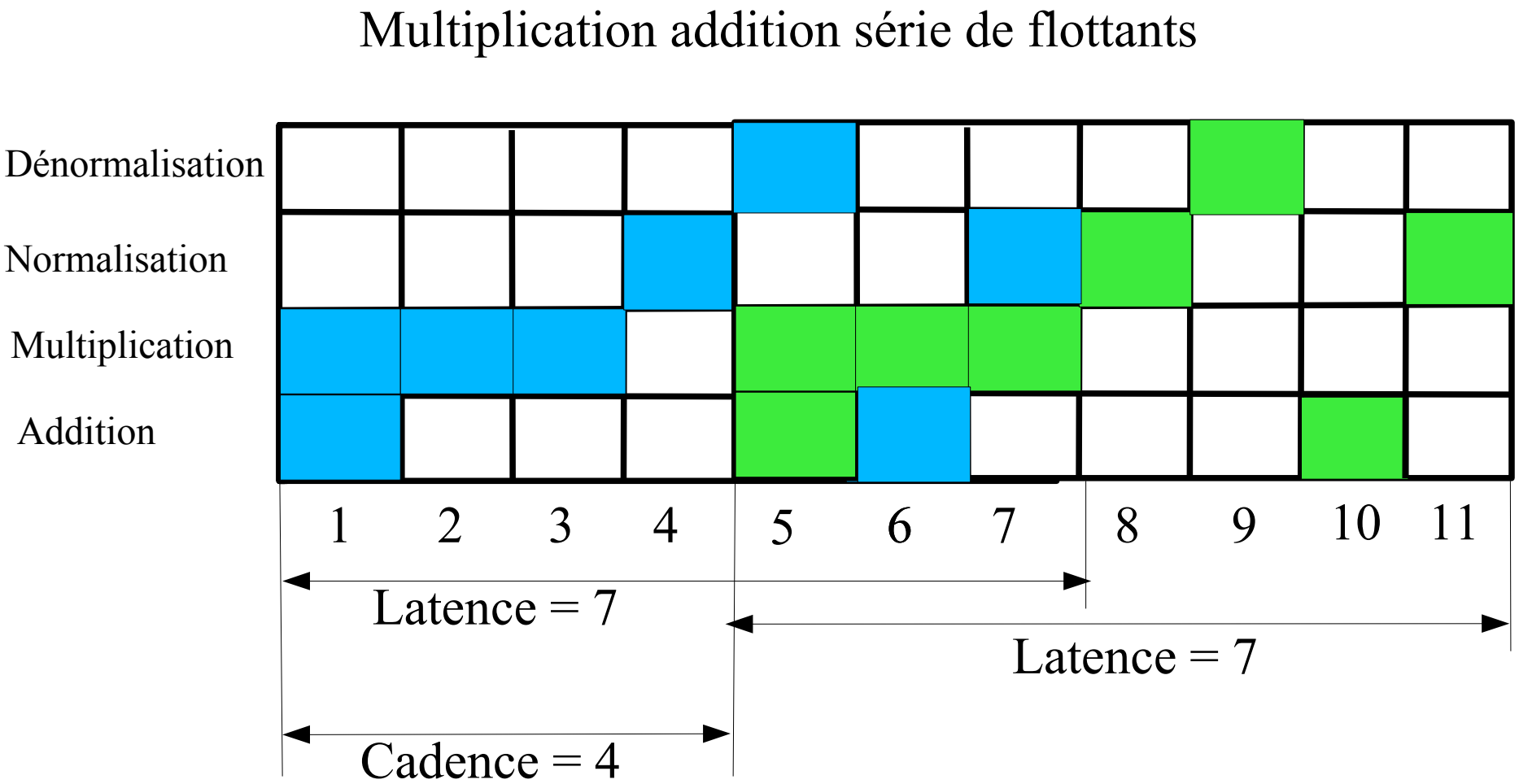



- Etages à durées multiples

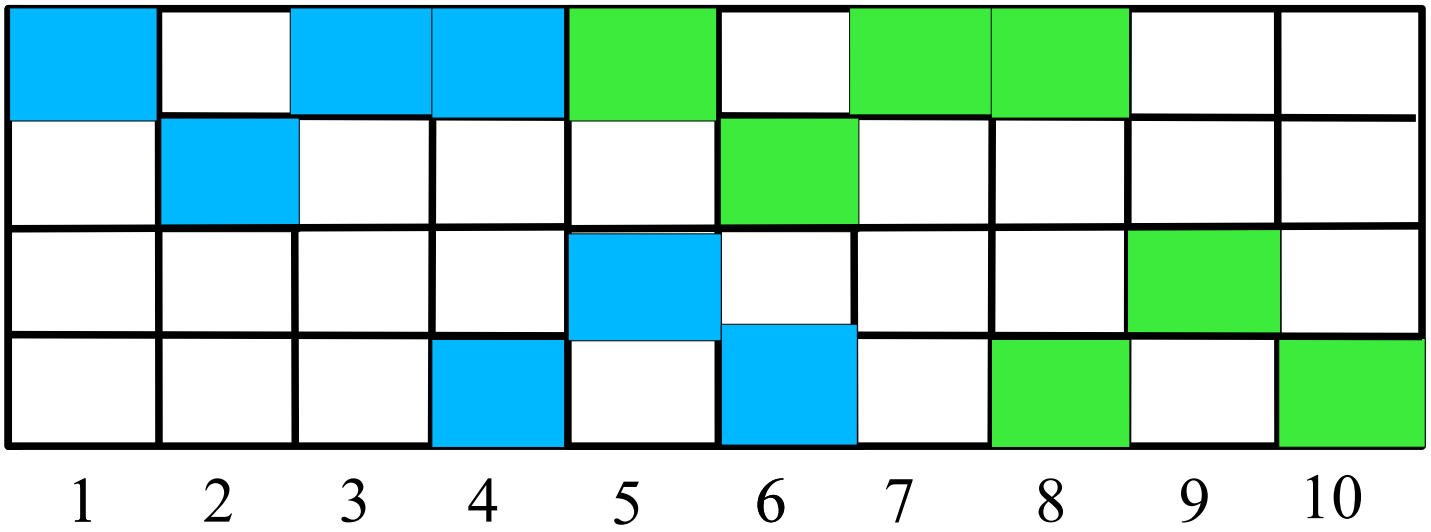


# Exploitation logicielle // pipe-line (1)

- Table de réservation



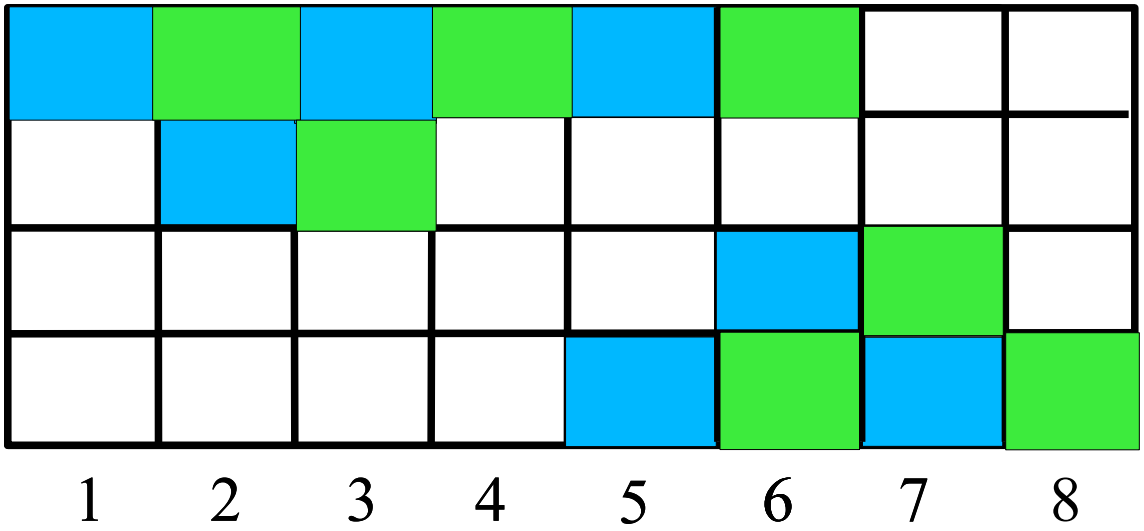
# Exploitation logicielle // pipe-line(2)




Latence = 6

Cadence = 4T

- Re-timing




Latence = 6 + 1

ou

Cadence : 2 exec pour 6T

Cadence : 3T

# II. Temps Réel Distribué

1. Introduction

2.Techniques de communication entre tâches

3.OSEK-VDX

4. Ordonnancement de messages

5. Synchronisation d'horloges

# Temps réel

- Réactivité
- Contraintes temporelles
  - Multi-échantillonnage
  - Maîtrise des retards E/S
- Sûreté de fonctionnement

**ordonnancement de tâches**

# Temps Réel Distribué

- Temps réel
- Communications
  - Coordination des actions entre processeurs
  - Cohérence temporelle des données traitées
  - Temps de communications

**ordonnancement de tâches**

**+**

**ordonnancement de messages**

# Conception des STR Distribués (1)

- STR
  - Méthodes d'analyse permettant de s'assurer que les contraintes temps réel seront satisfaites (RMA, ...)
  - Implantation à l'aide d'un OS temps réel
- //
  - Techniques de placement et d'ordonnancement ne prenant pas en compte les contraintes temporelles multiples (multi-périodes)
  - OS, langage, librairies de gestion des communications

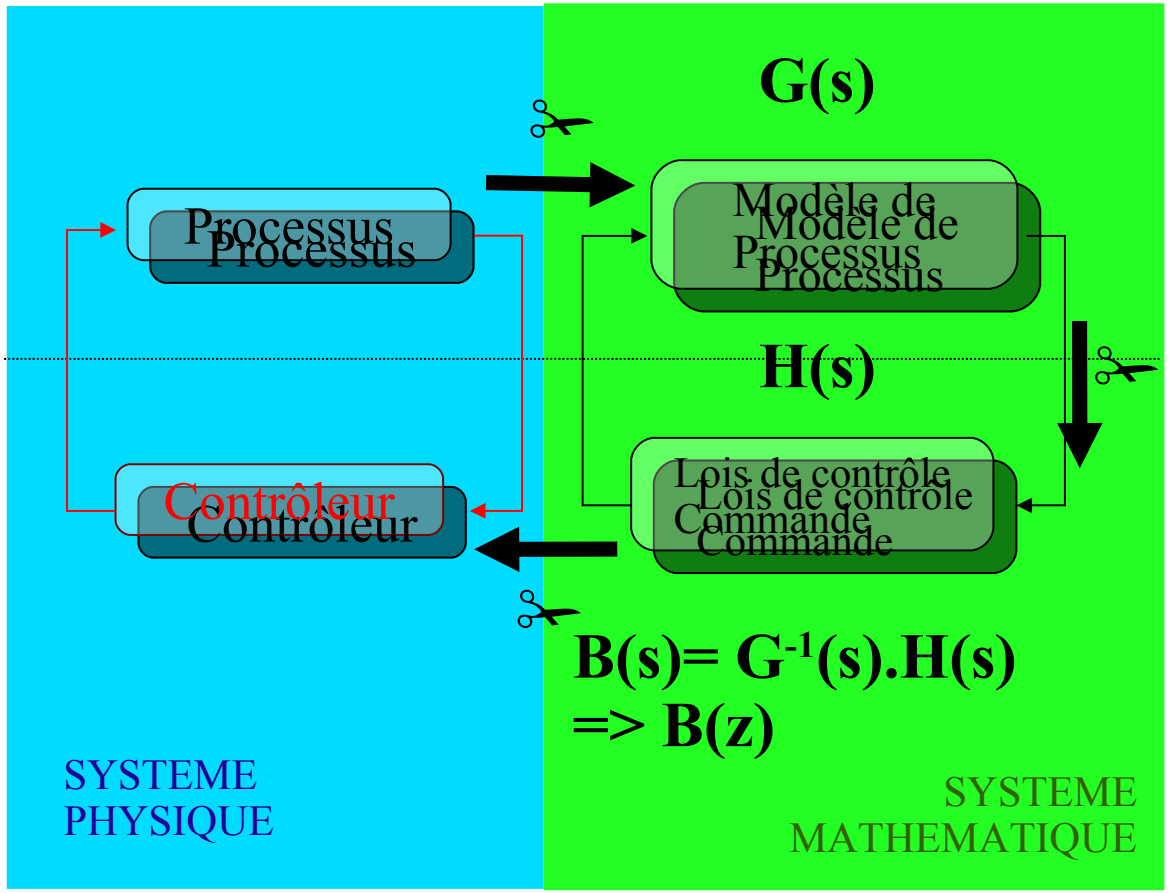
# Conception des STR Distribués (2)

- STR + coms =Ensemble de systèmes communiquants
  - Ordonnancement de tâches : idem STR (analyse + OS)
  - Ordonnancement de messages
  - Utilisation peu efficace de l'architecture //
  - **Approche la plus courante**
- STR //
  - Placement et ordonnancement optimal
  - Ordonnancement conjoints des tâches et messages
  - **Domaine de la recherche**

# II. Temps Réel Distribué

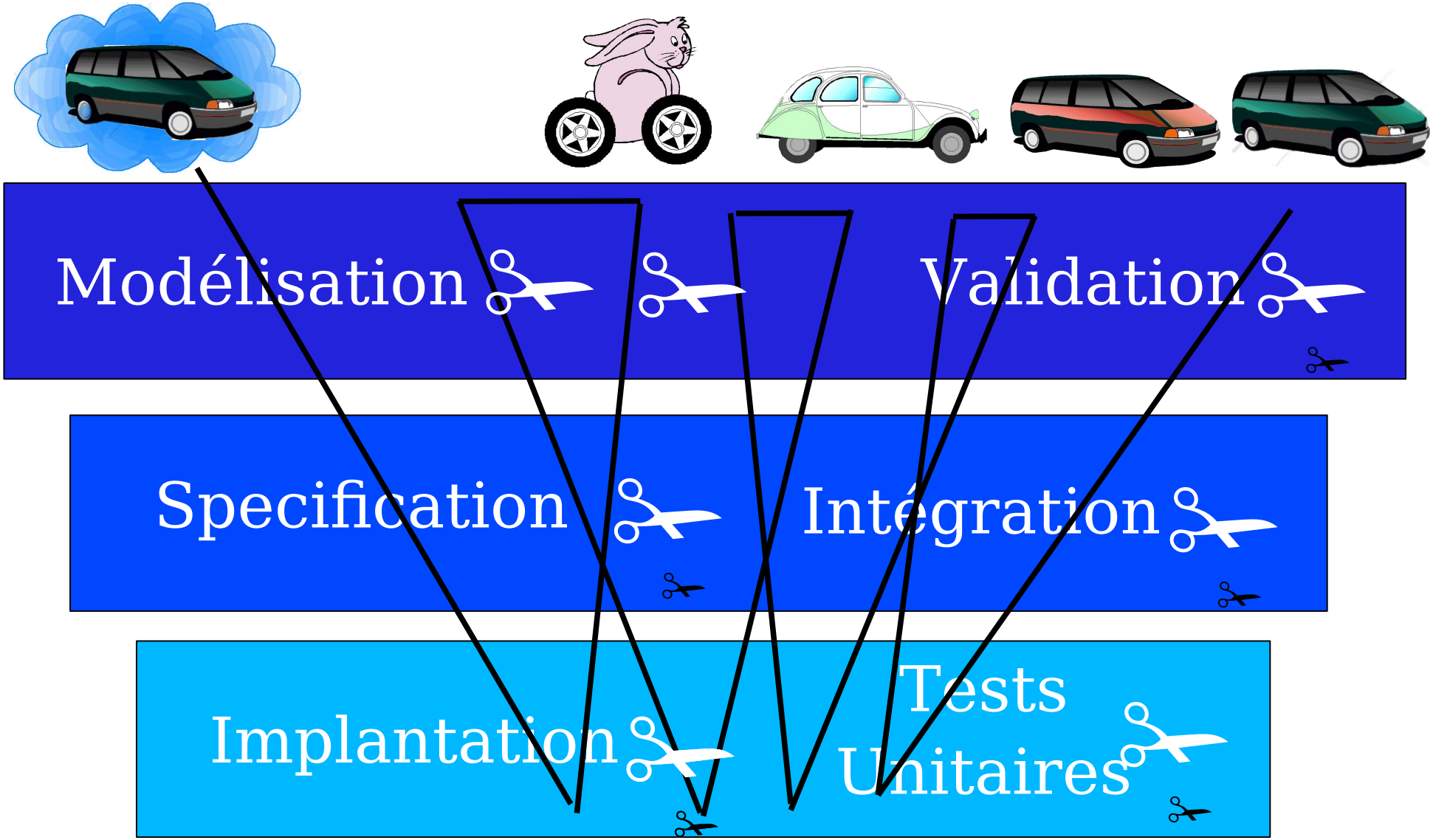
1. Introduction
2. Techniques de communication entre tâches
3. OSEK-VDX
4. Ordonnancement de messages
5. Synchronisation d'horloges

# Conception d'un système de contrôle



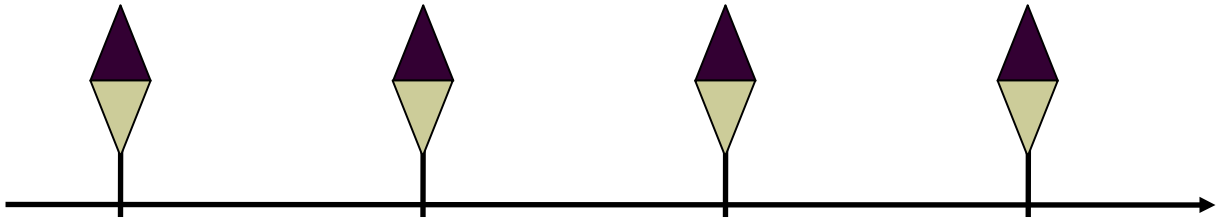

- Modélisation du processus
- Synthèse et identification des lois de contrôle-commande
- Réalisation du Contrôleur et validation

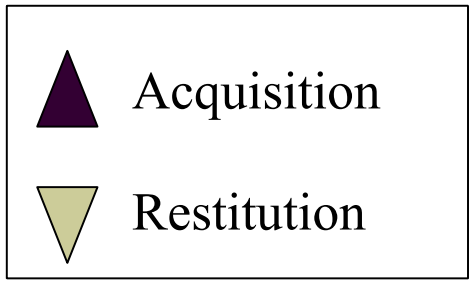
# Cycle de développement en V



# Etape de modélisation

- Loi de commandes : domaine de l'automatique
- Synthèse de lois de commandes échantillonnées ou discrétisation de lois de commandes continues
- Choix des périodes empirique (10 * constante de temps du système)






Périodicité
Synchronisme E/S
Multi-échantillonnage difficile

# Etape d'implantation

- Gestion d'activités parallèles
- Respecter des contraintes temporelles
- Minimiser les coûts

! Allouer **spatialement** (multiprocesseurs) et **temporellement** des ressources de manière efficace en garantissant le comportement temporel

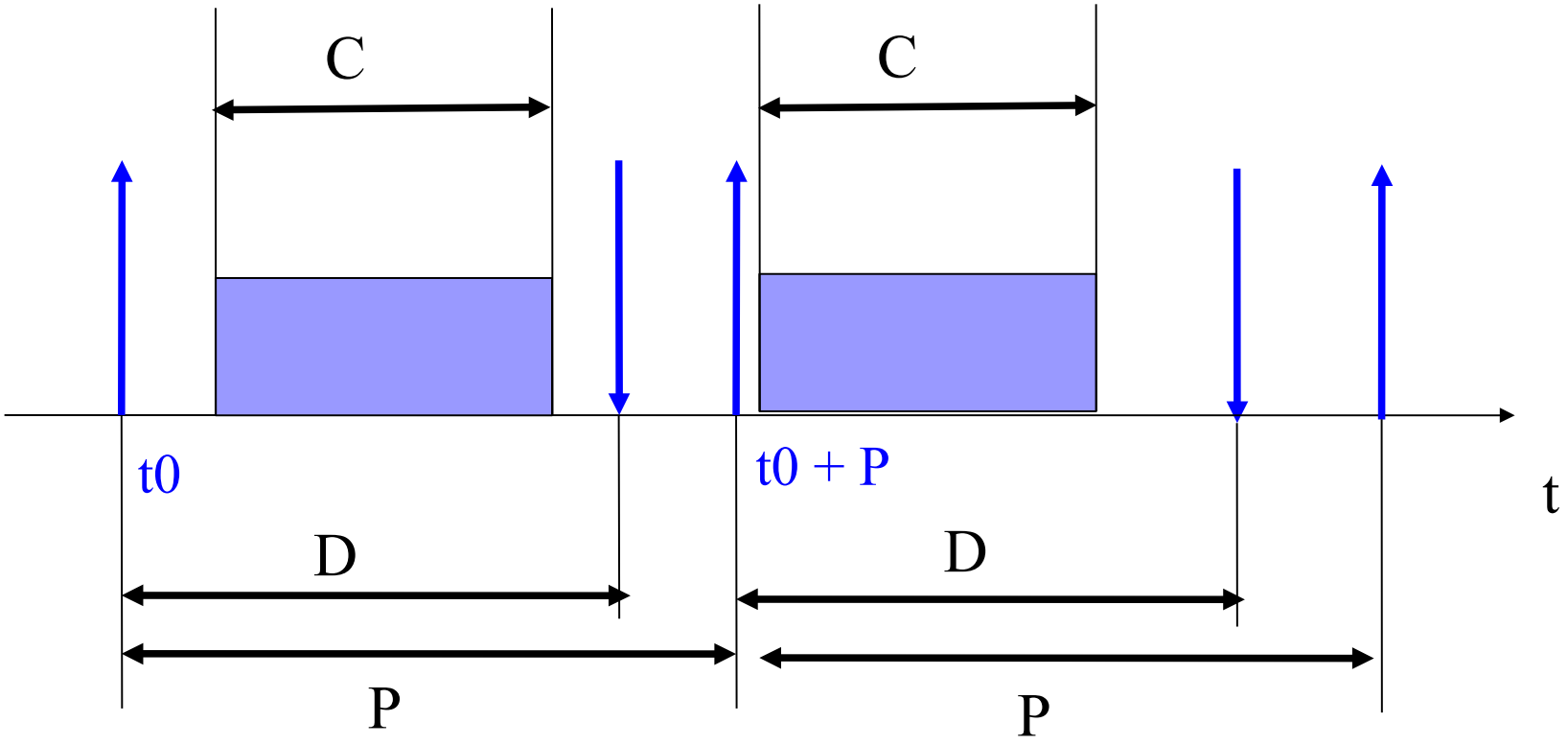
- Ordonnancement : allocation temporelle sur une ressource
  - En ligne, hors ligne
  - préemptif, non-préemptif
  - priorités fixes, priorités dynamiques

# Implantation : ordonnancement

- Découpage en tâches : choix des caractéristiques
- Analyse de l'ordonnancement
  - choix d'un algorithme d'ordonnancement : RM, IDM, EDF, préemption, ...
  - analyse d'ordonnançabilité
- Implantation à l'aide d'un OS

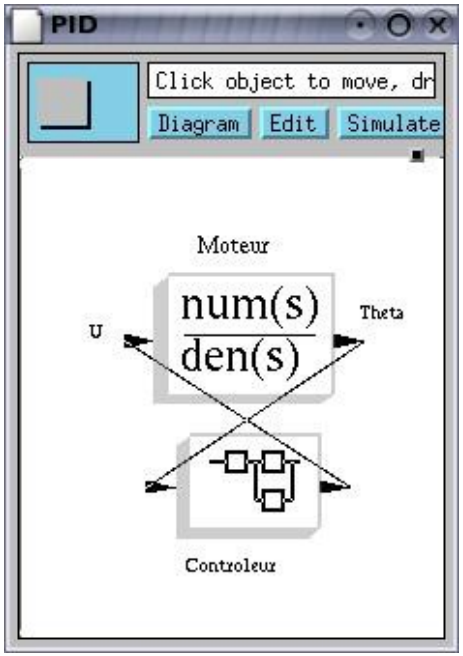
# Implantation : modèle de tâche

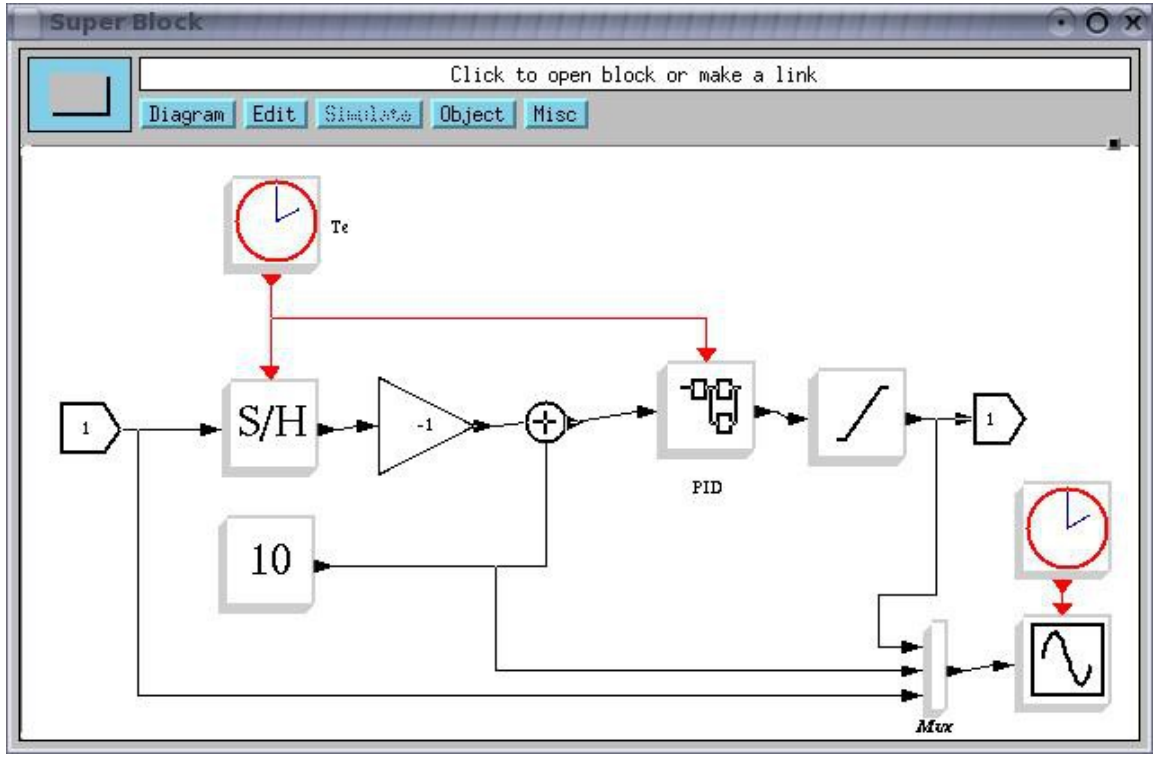
- Tâches indépendantes
- T(t0,C,P,D)



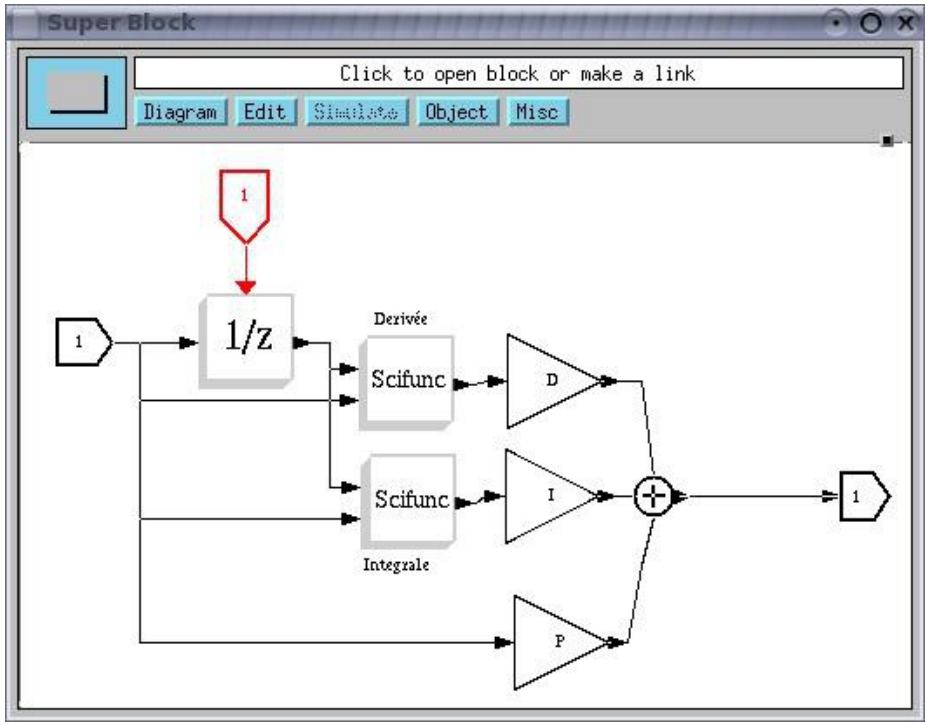
# Exemple : asservissement position

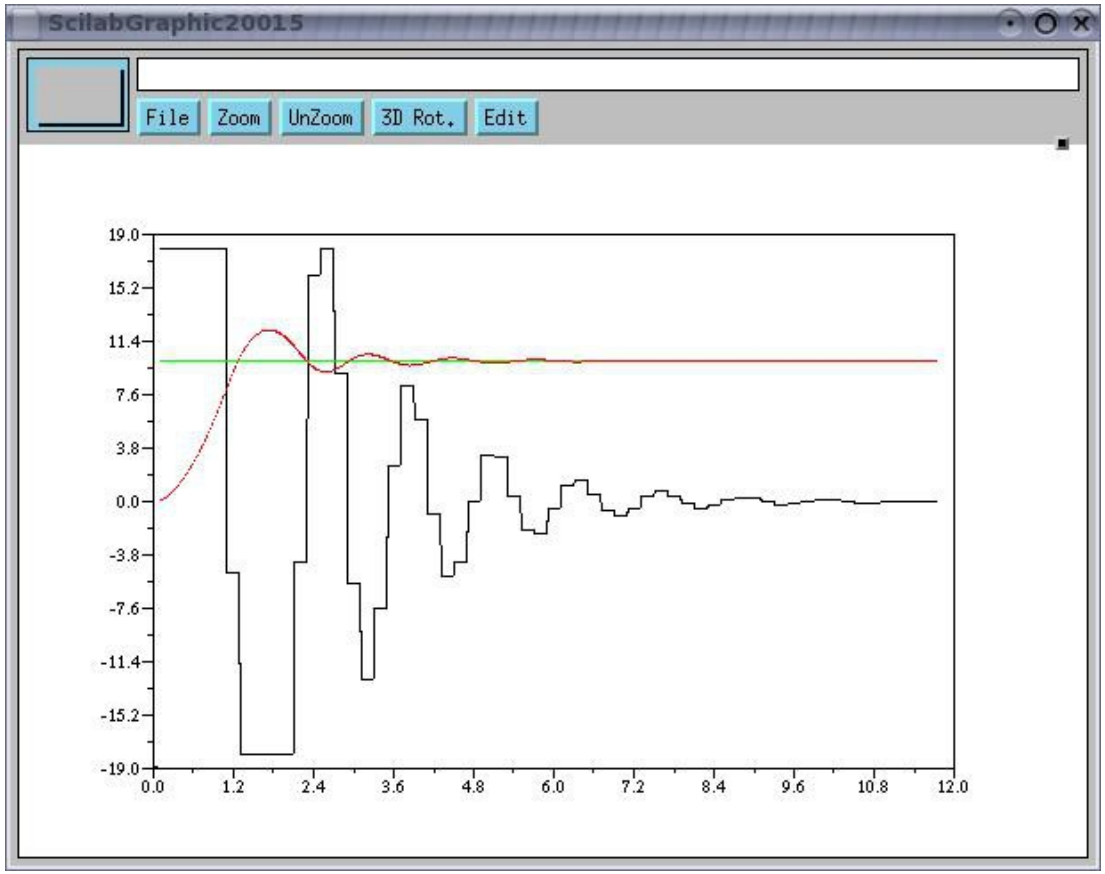
- Modélisation du système :
  moteur à courant continu => 1/(s(s+a)), a=0.9
- Synthèse de la loi de commande:
  système boucle fermée à PID discretisé Te=0.2s
- Simulation



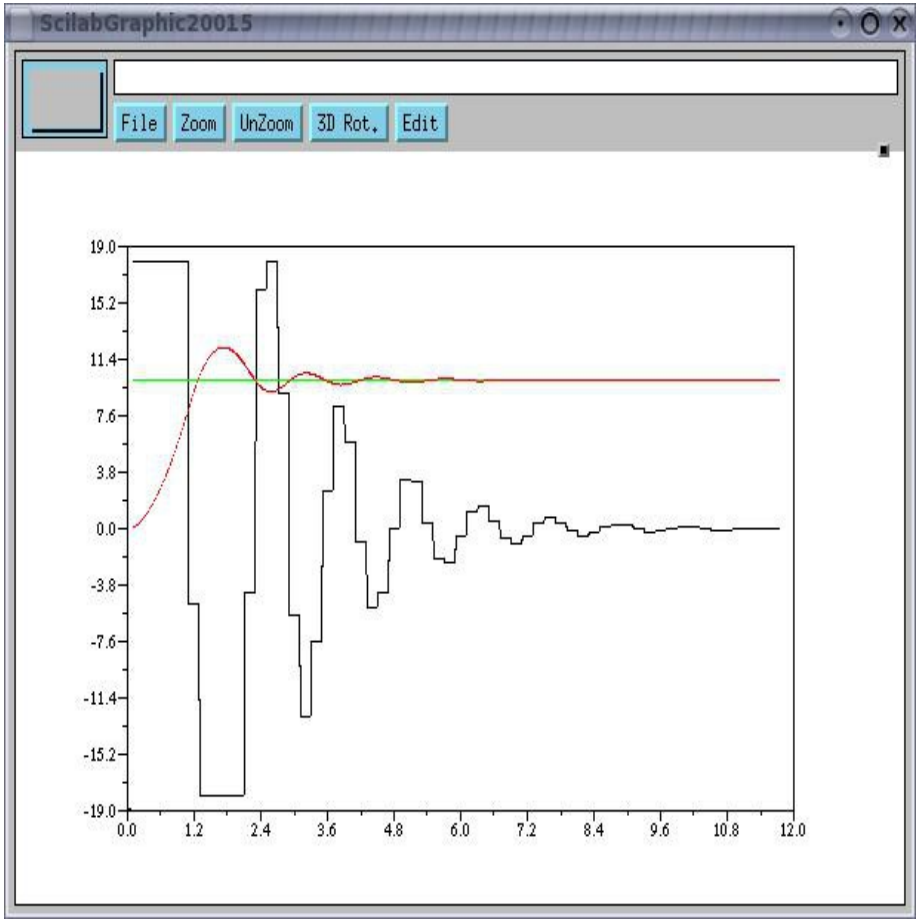
# Asservissement position (2)



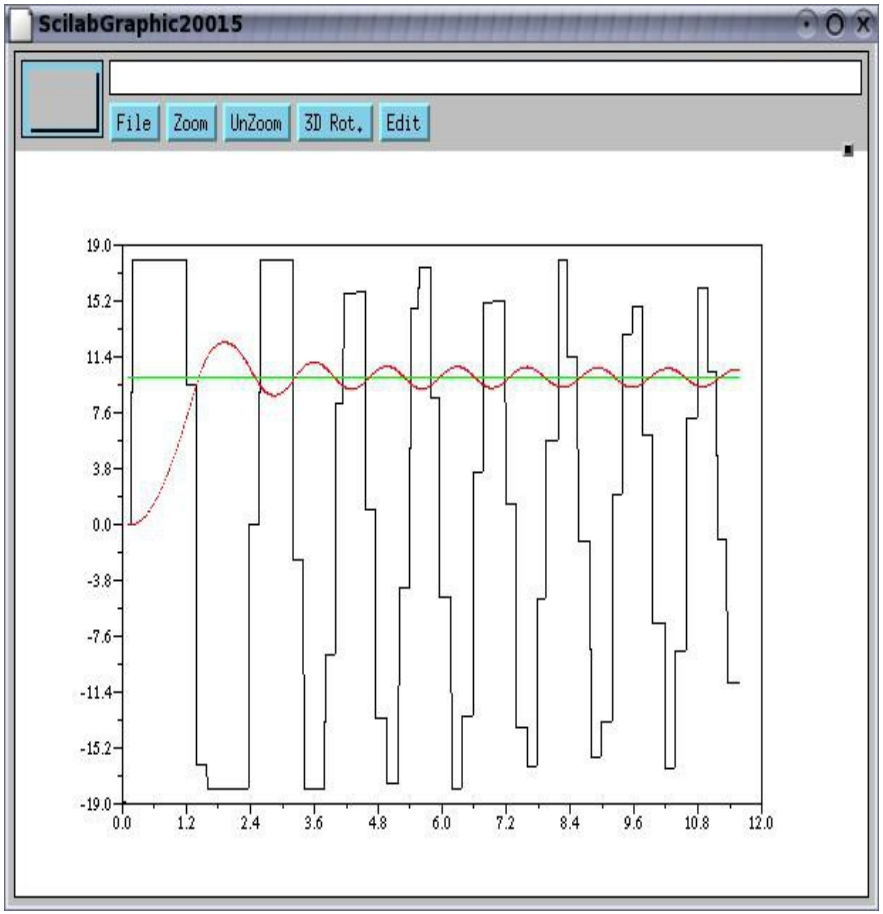




a=0.9, P=13, I=9.8, D=3.1, Te=0.2s

# Asservissement position : implantation



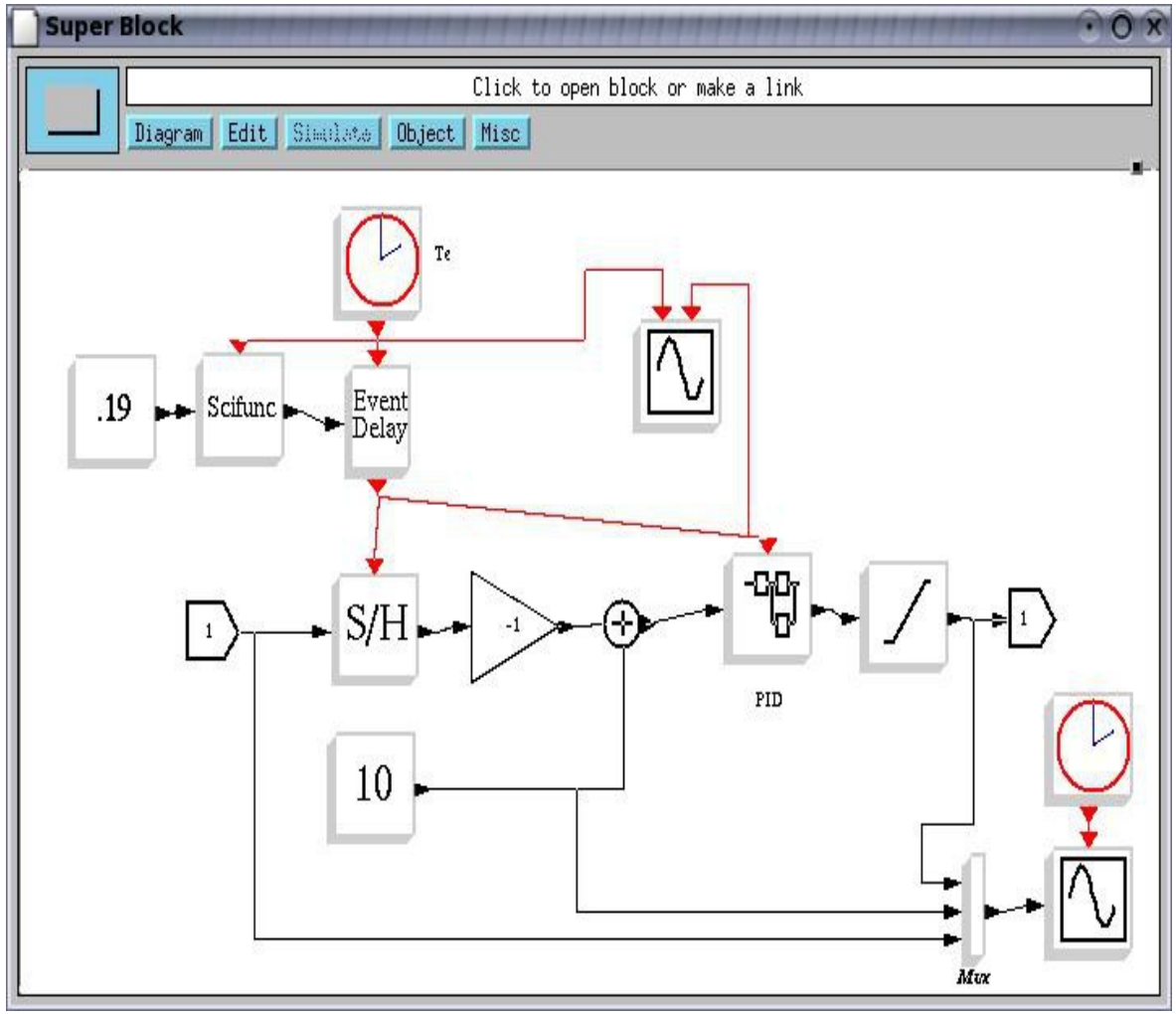

Simulation Scicos

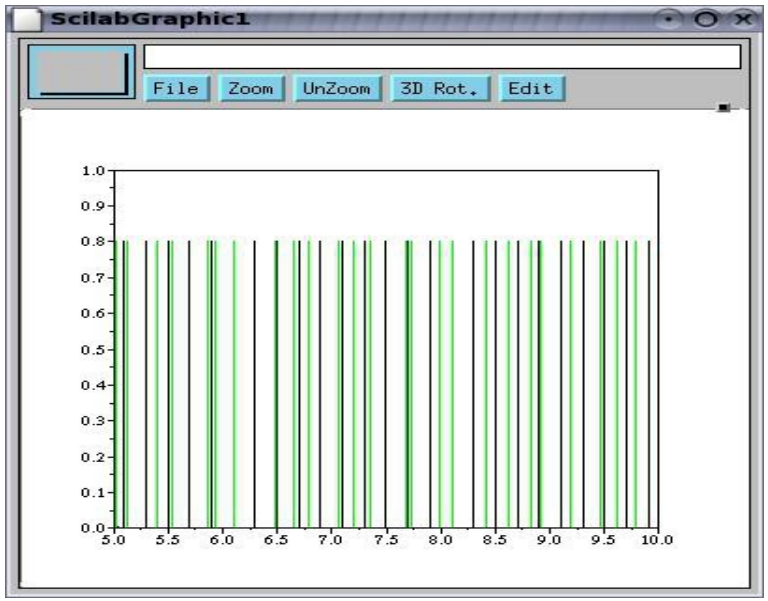


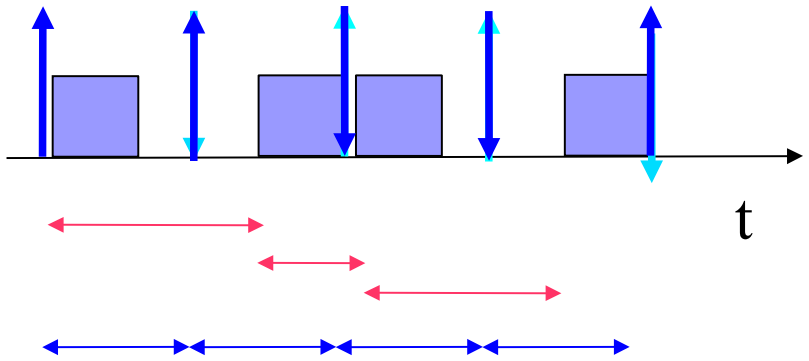

Exécution Monoprocesseur

1 tâche T(t0,C,D,P)
t0=0, C=60ms,P=Te, D=P

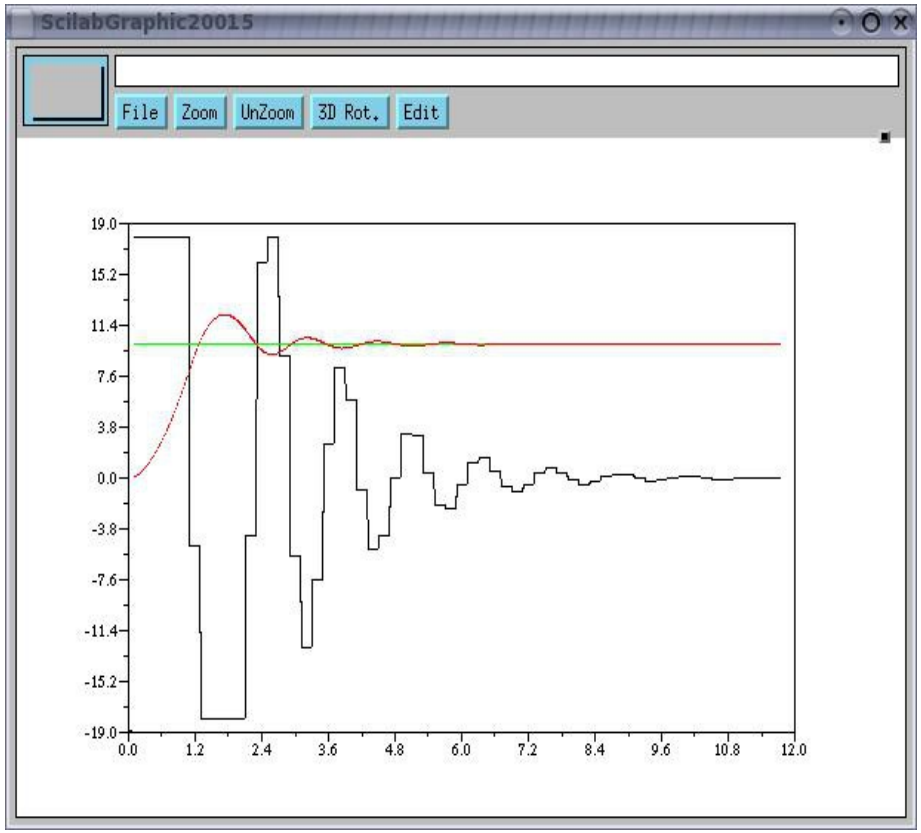
# Gigue échantillonnage



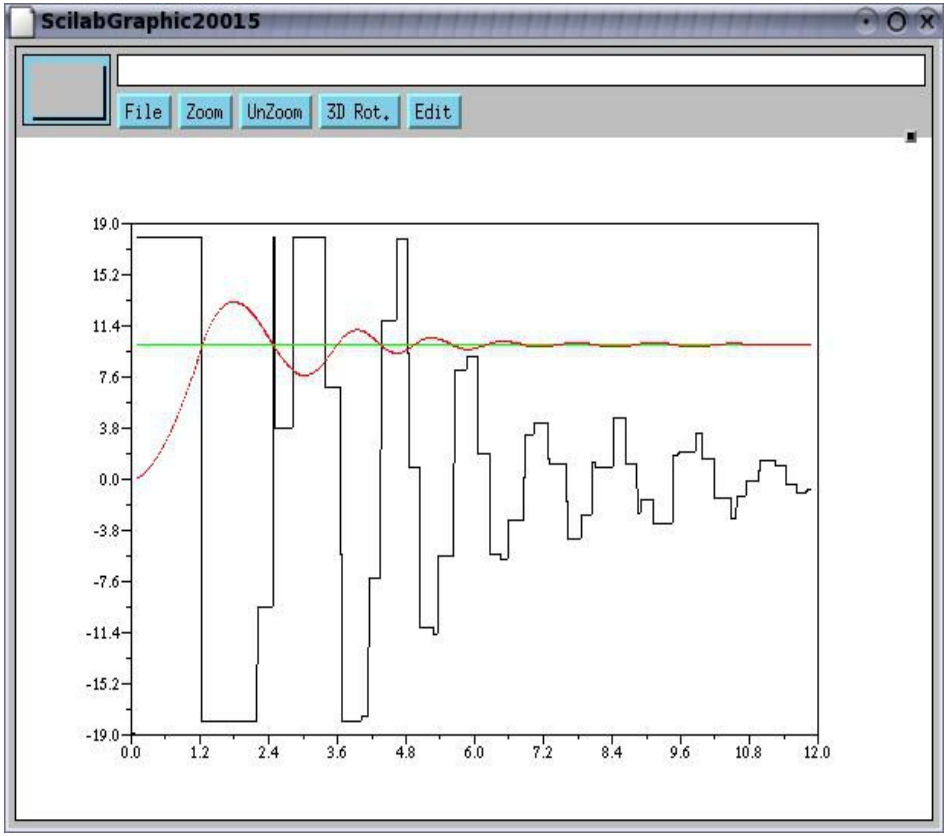



# Simulation gigue d'échantillonnage

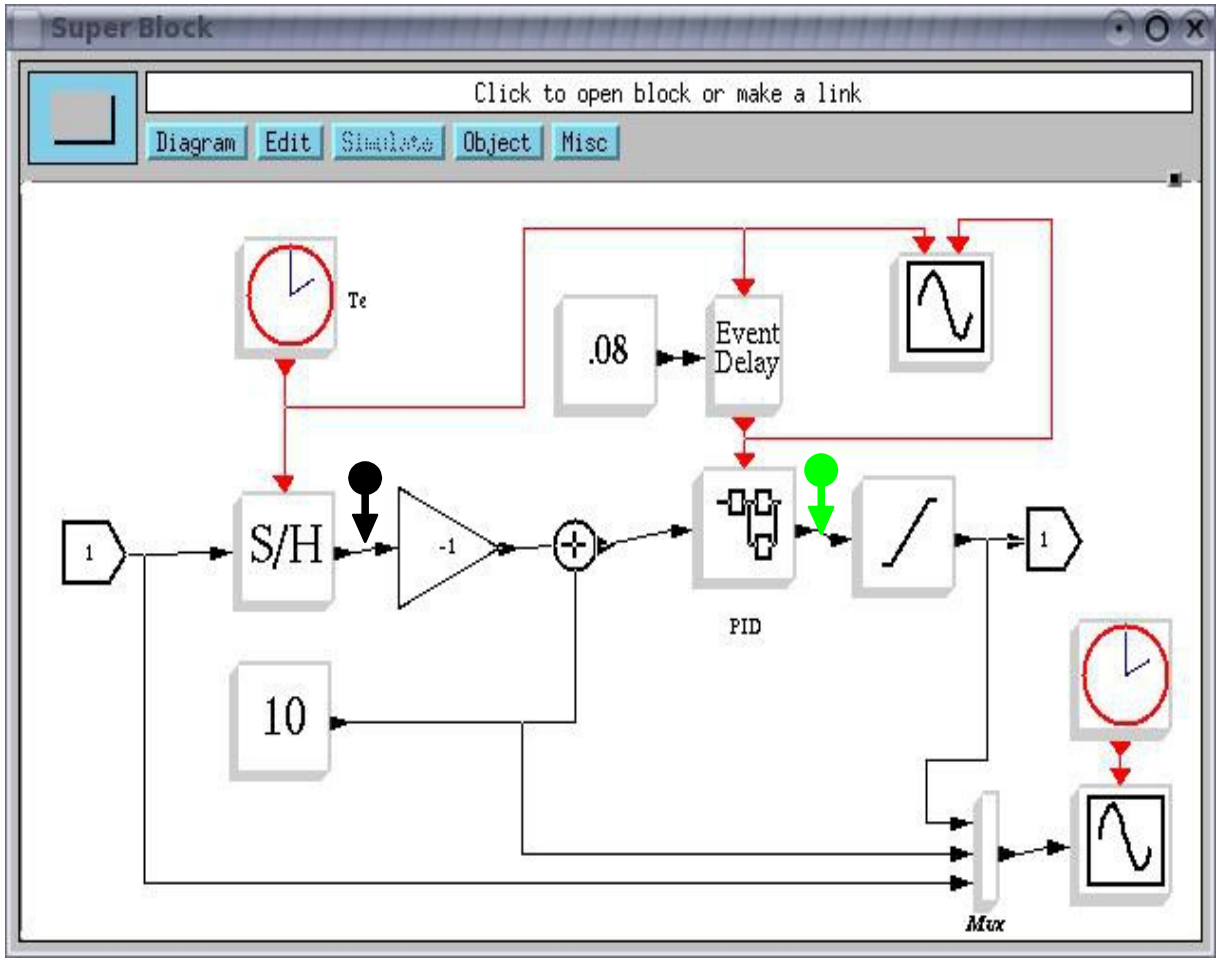


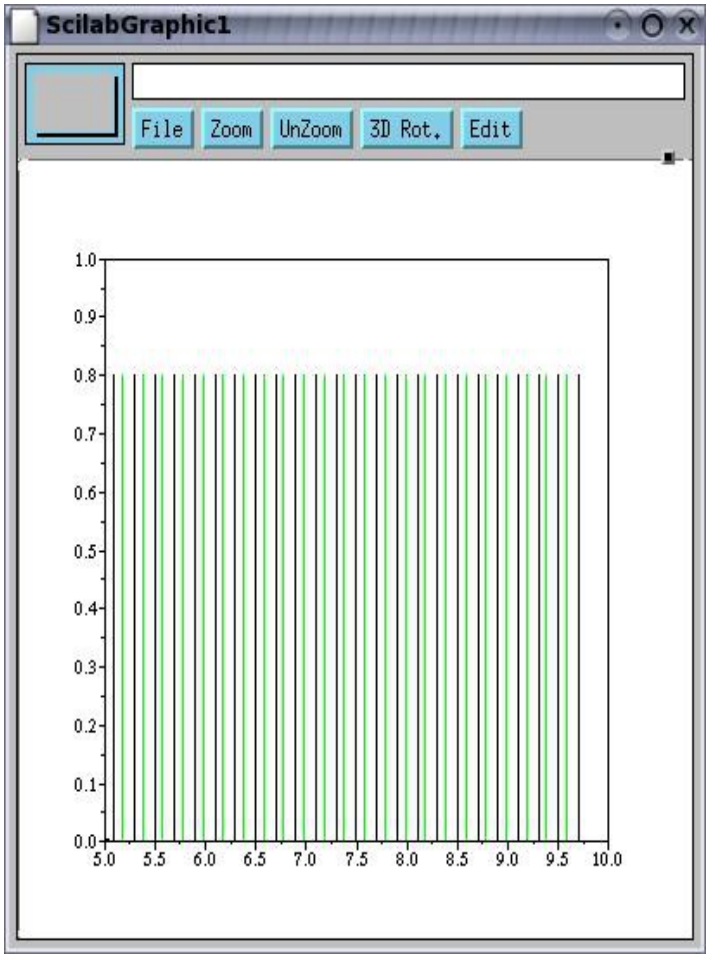

Simulation Scicos
de référence



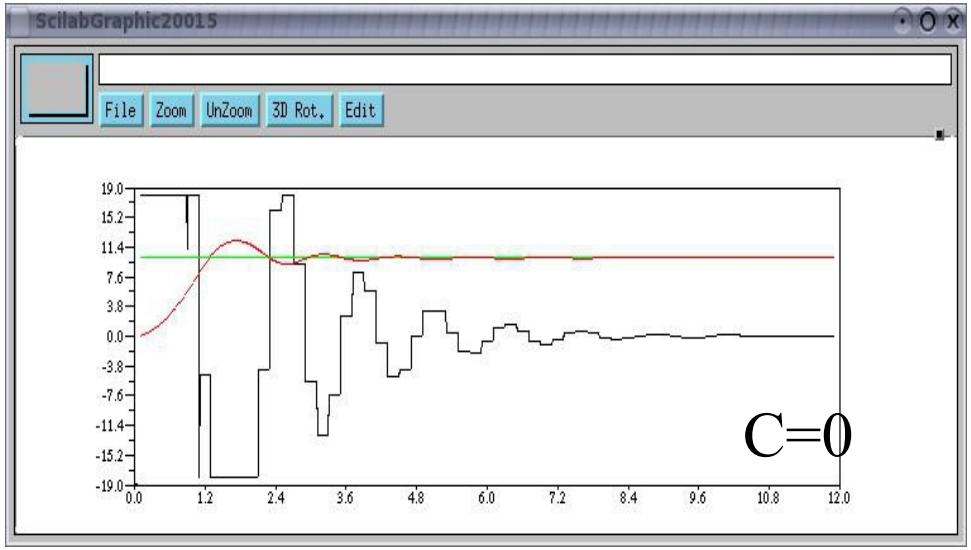

Simulation Scicos
avec gigue

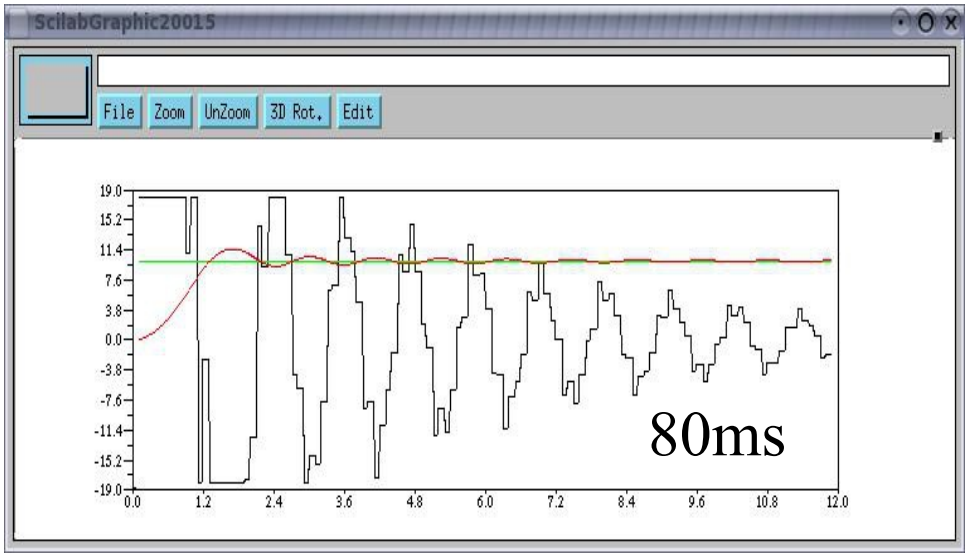
# Temps de calcul

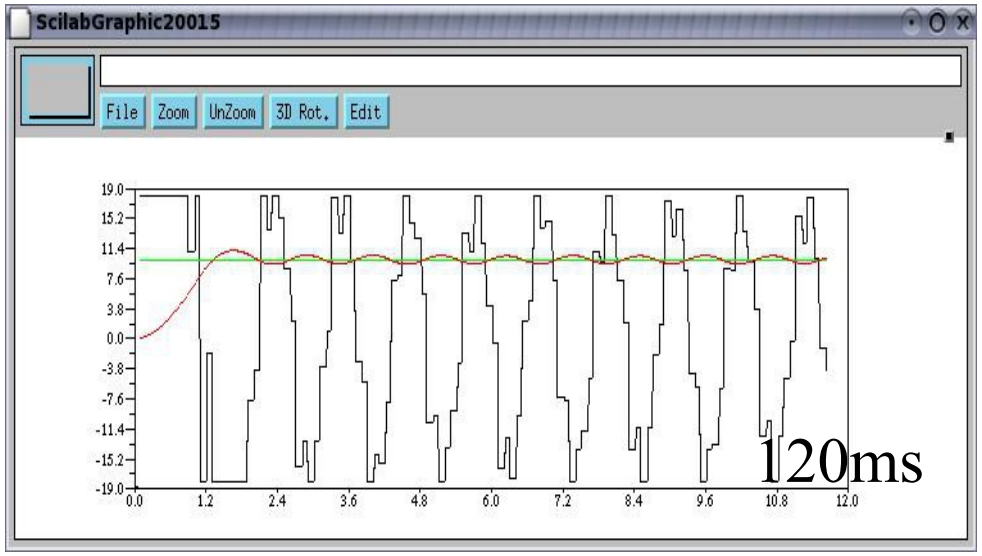


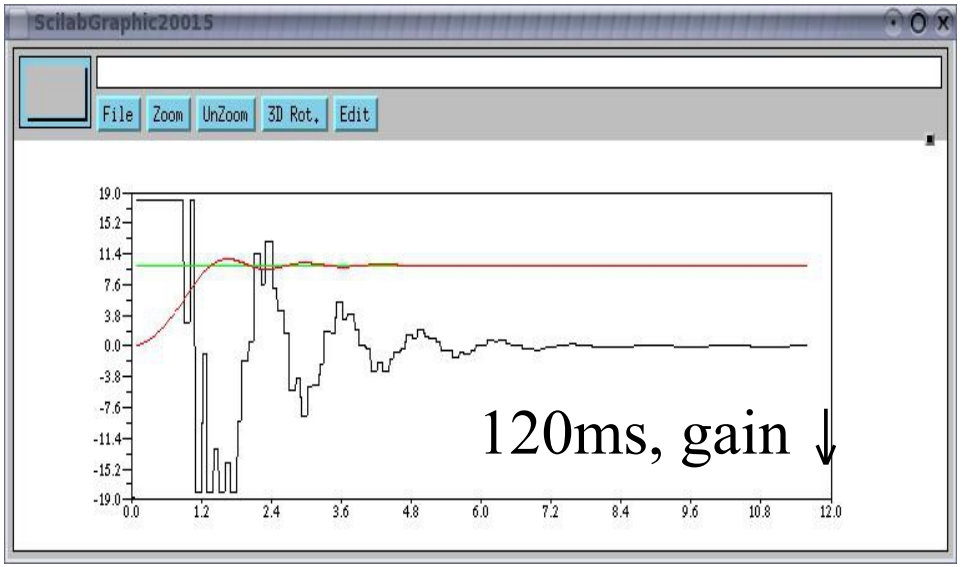



# Simulations temps de calcul

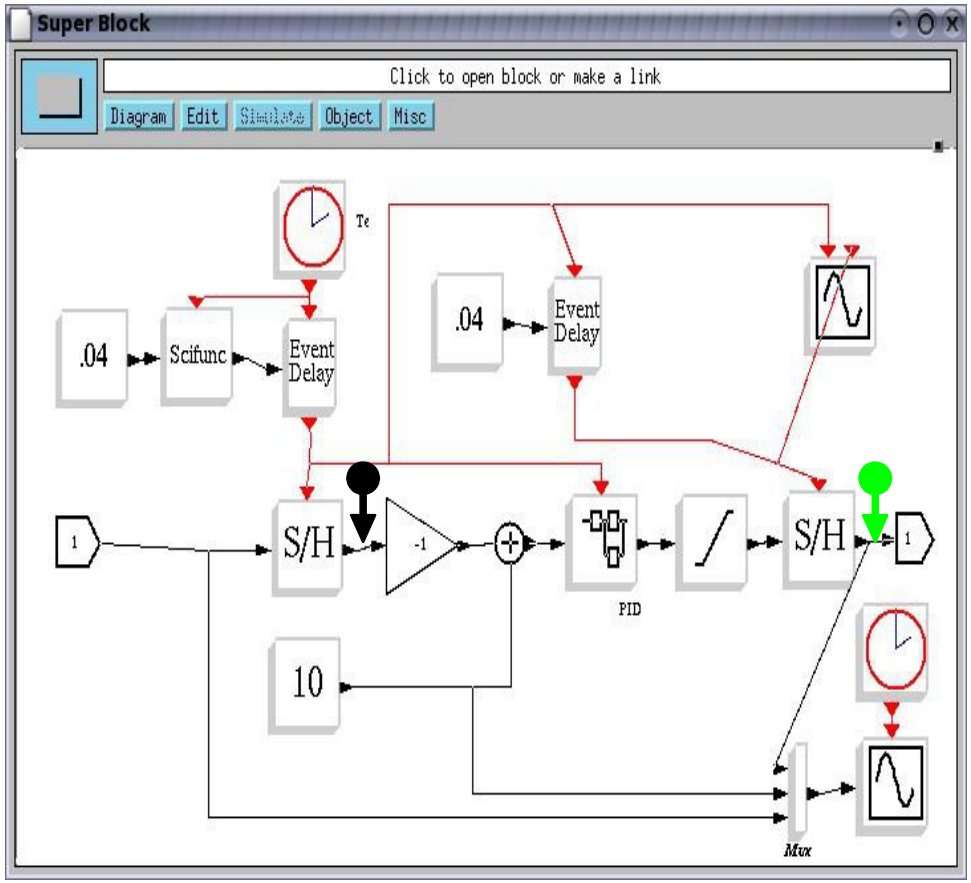


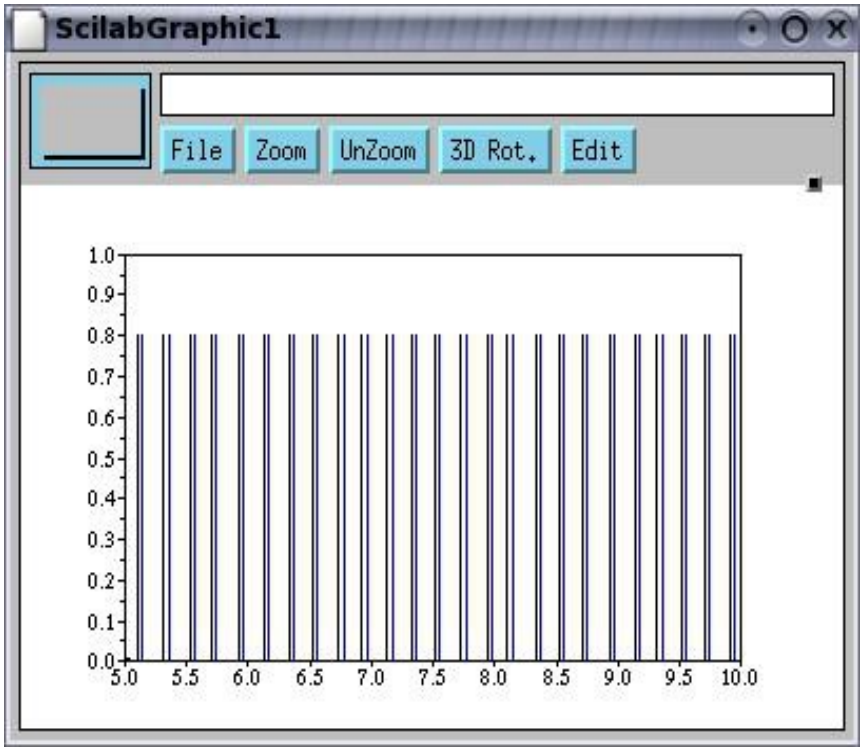



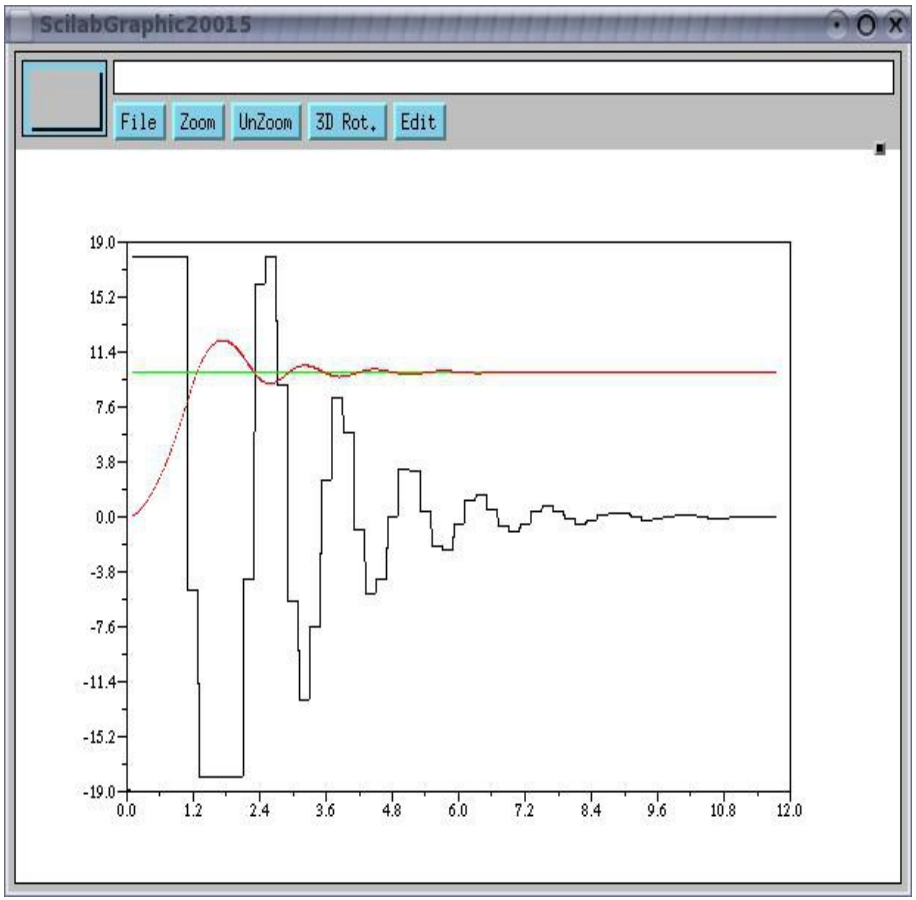



# Gigue + temps de calcul

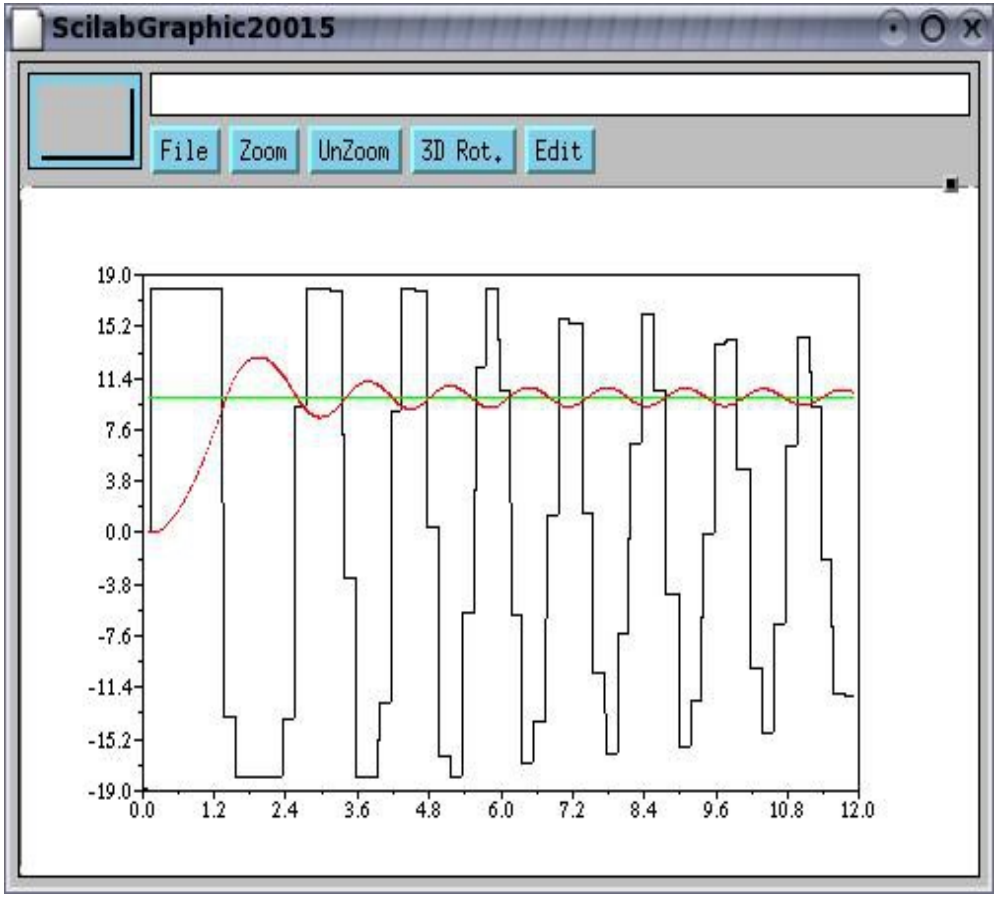


# simulation gigue + retard



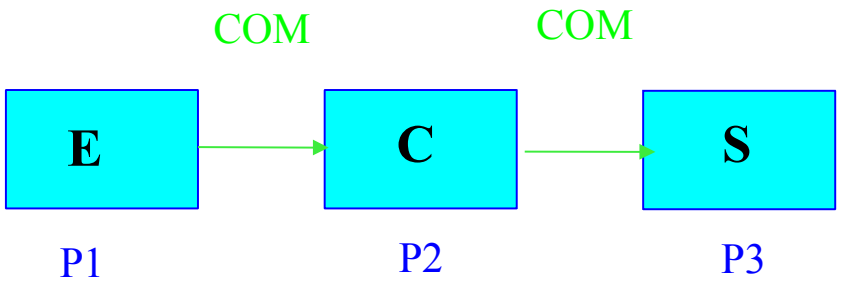

Simulation Scicos
de référence

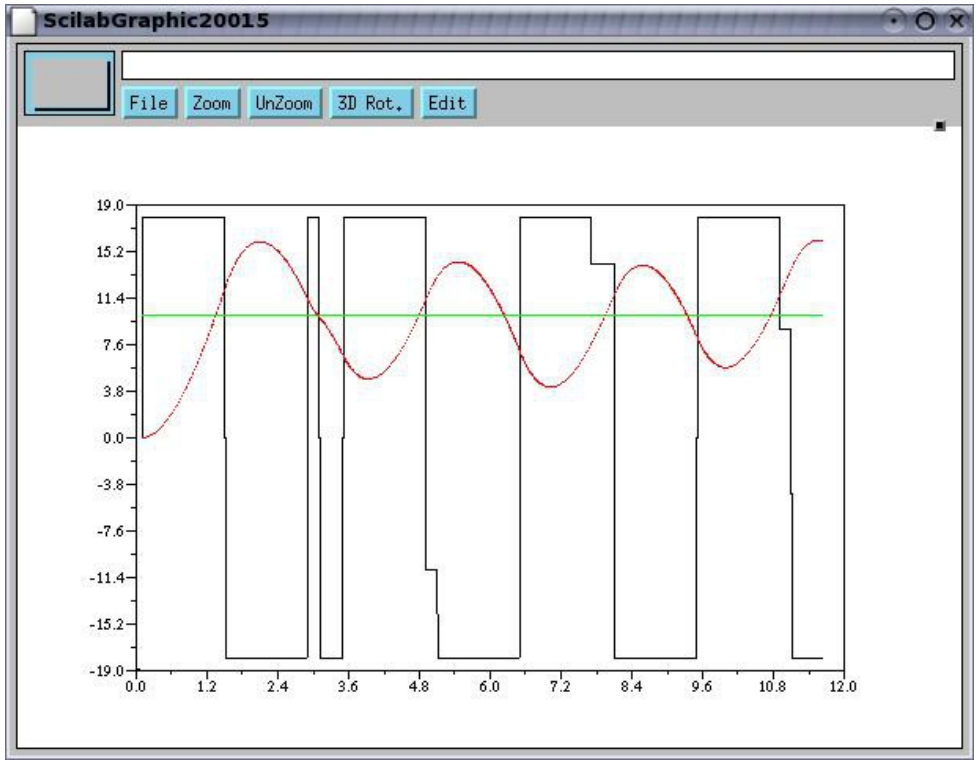



Gigue 40ms, retard 40ms

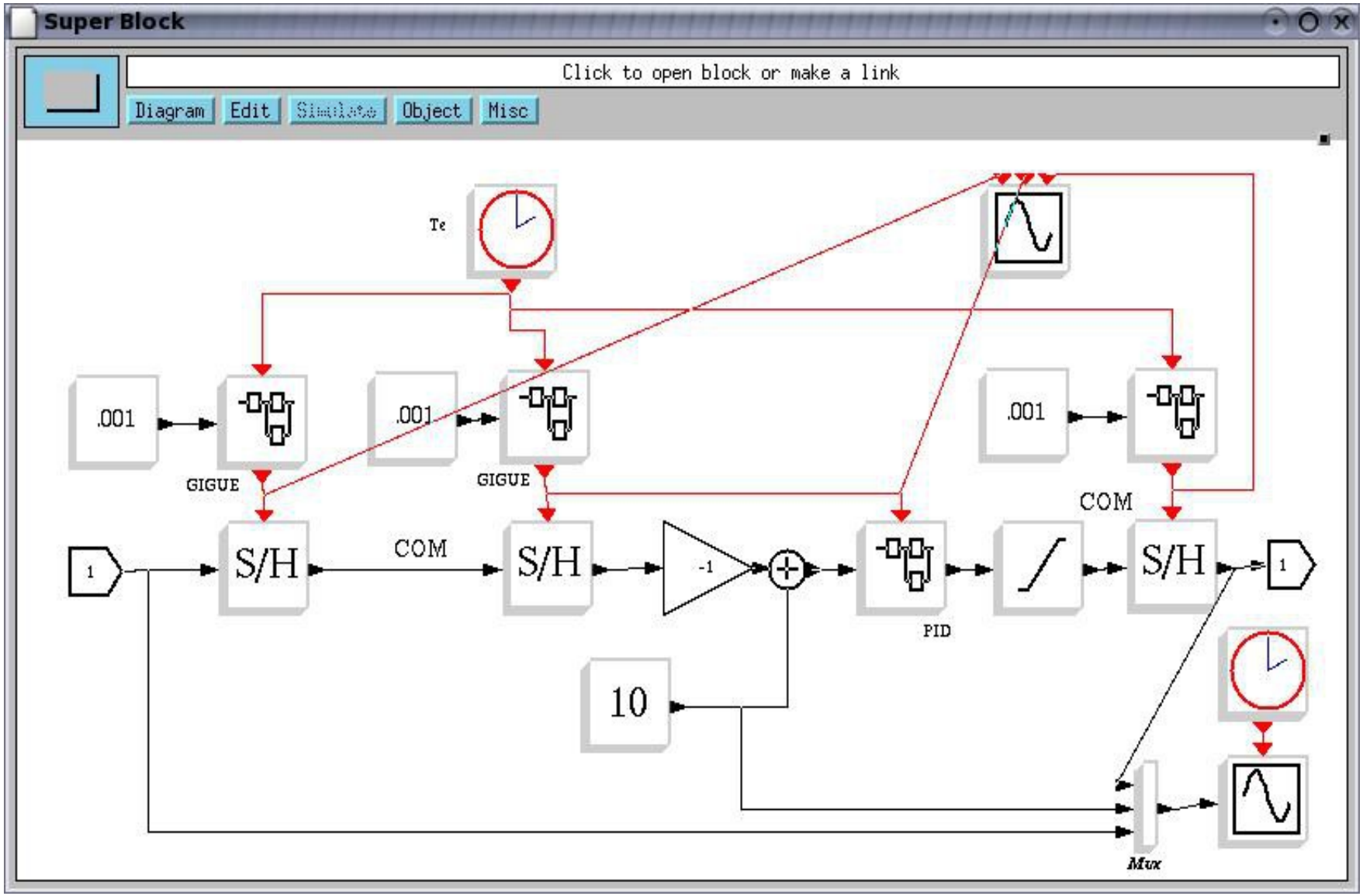
# Implantation distribuée

Hypothèses :
- Temps de Comunication nul
- Temps de calcul faible
- Gigue faible



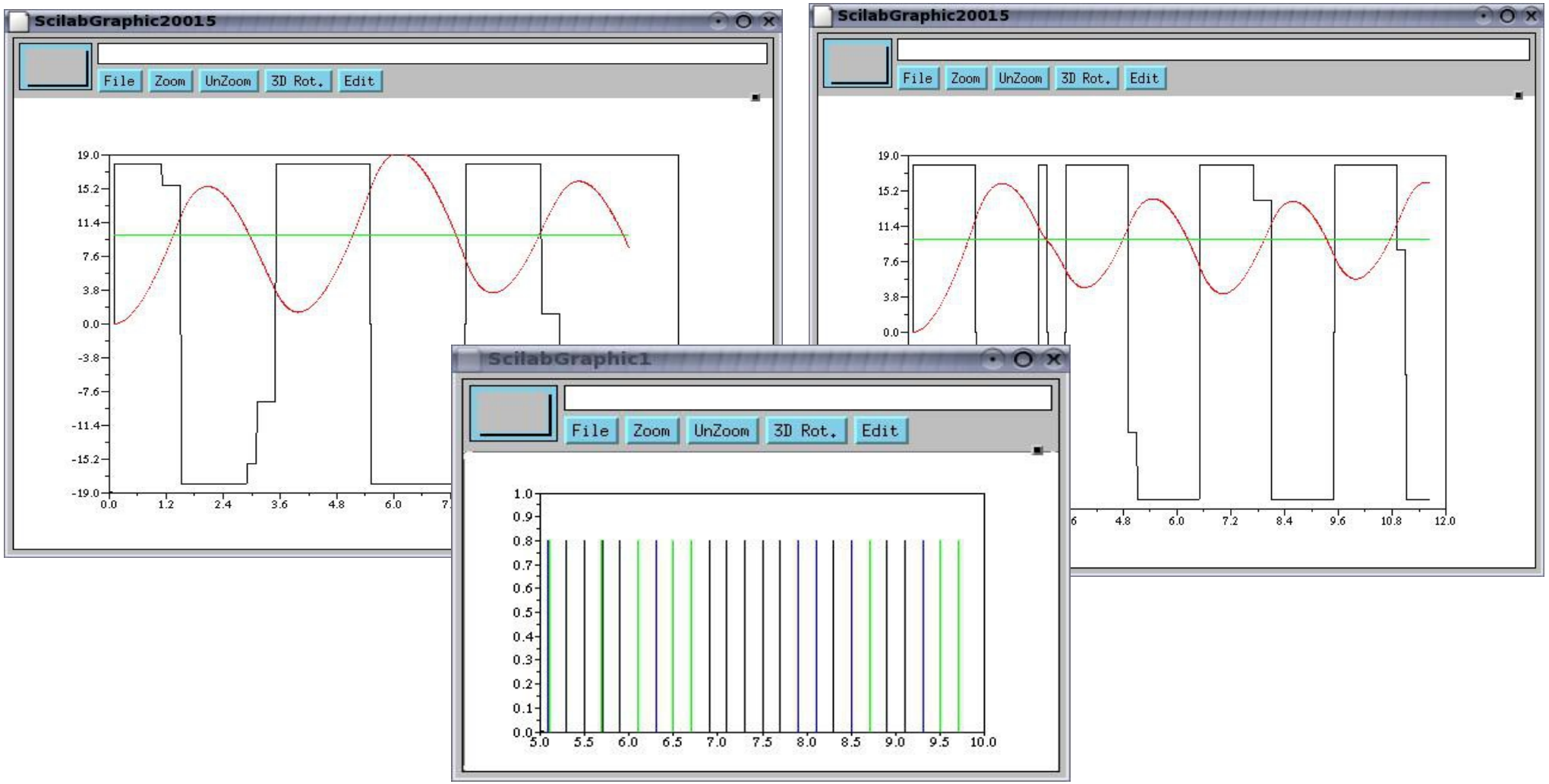
# Implantation distribuée



# Simulations implantation distribuée

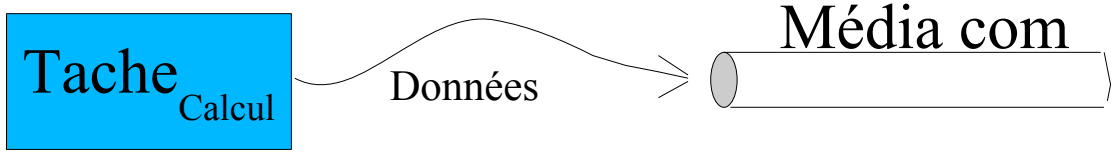
- **Instabilité**
- **Indéterminisme**

Pourquoi ?

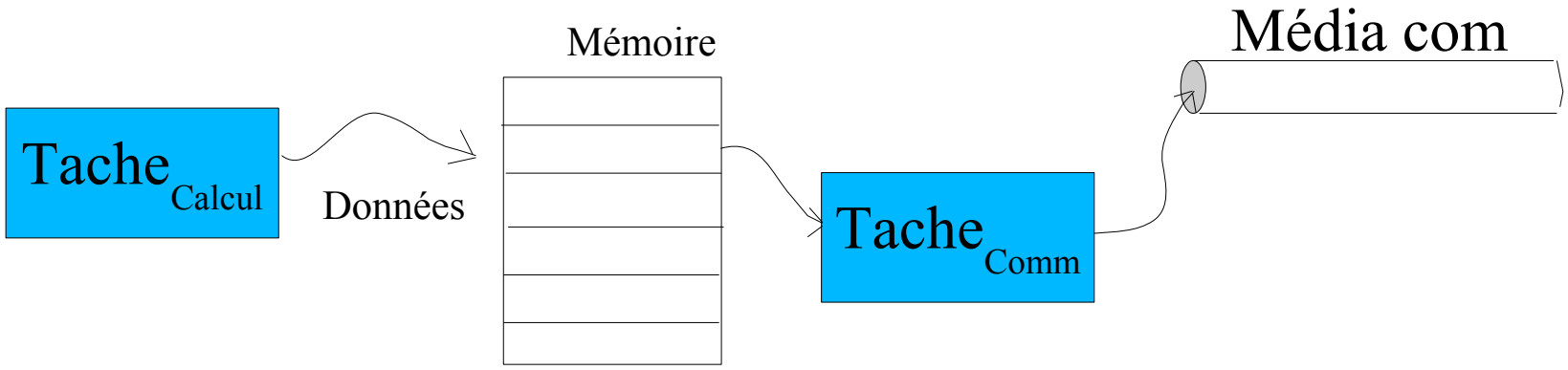


# Mécanismes de communication (1)

- Emission
  - Synchrone



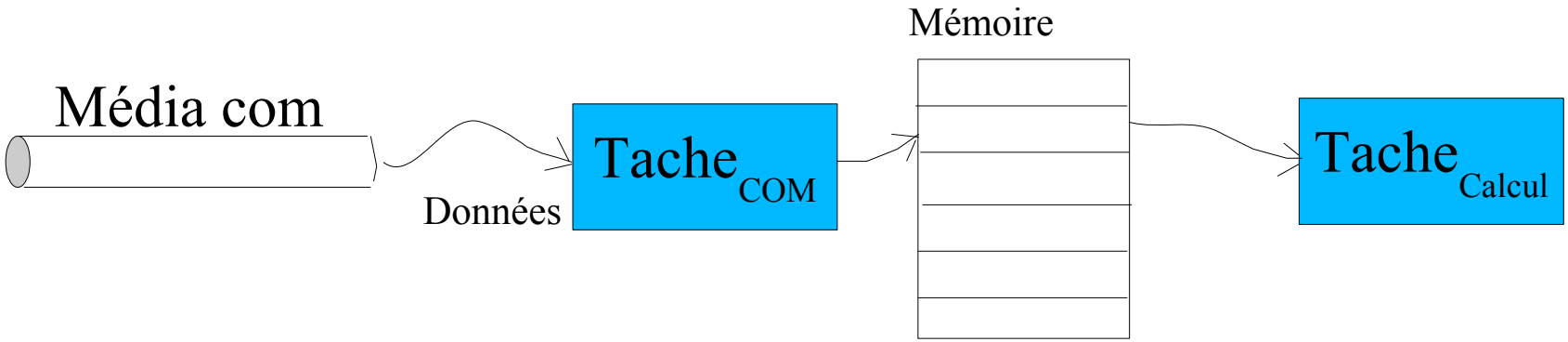

  - Asynchrone



# Mécanismes de communication (2)

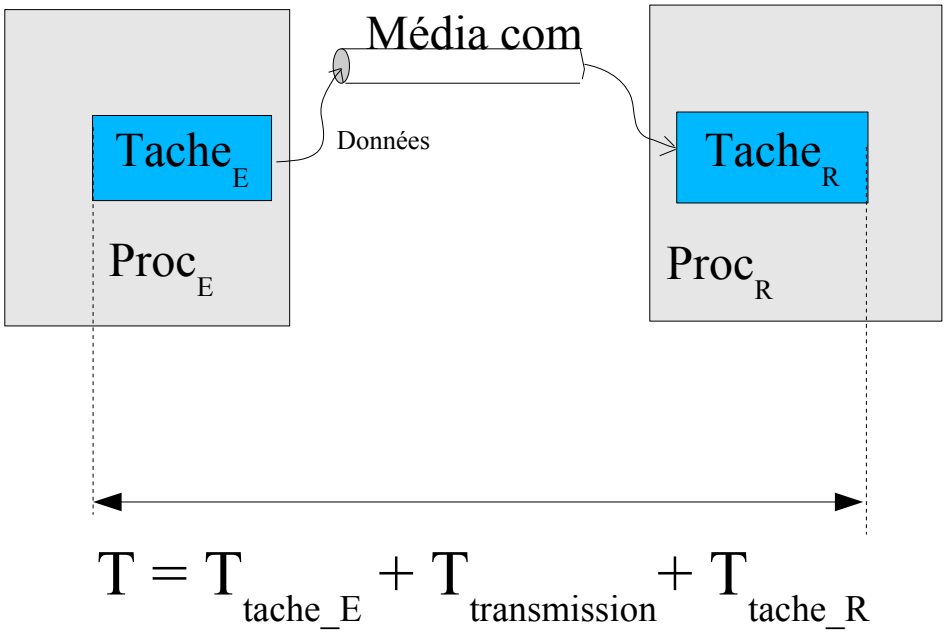
- Réception
  - Synchrone


  - Asynchrone



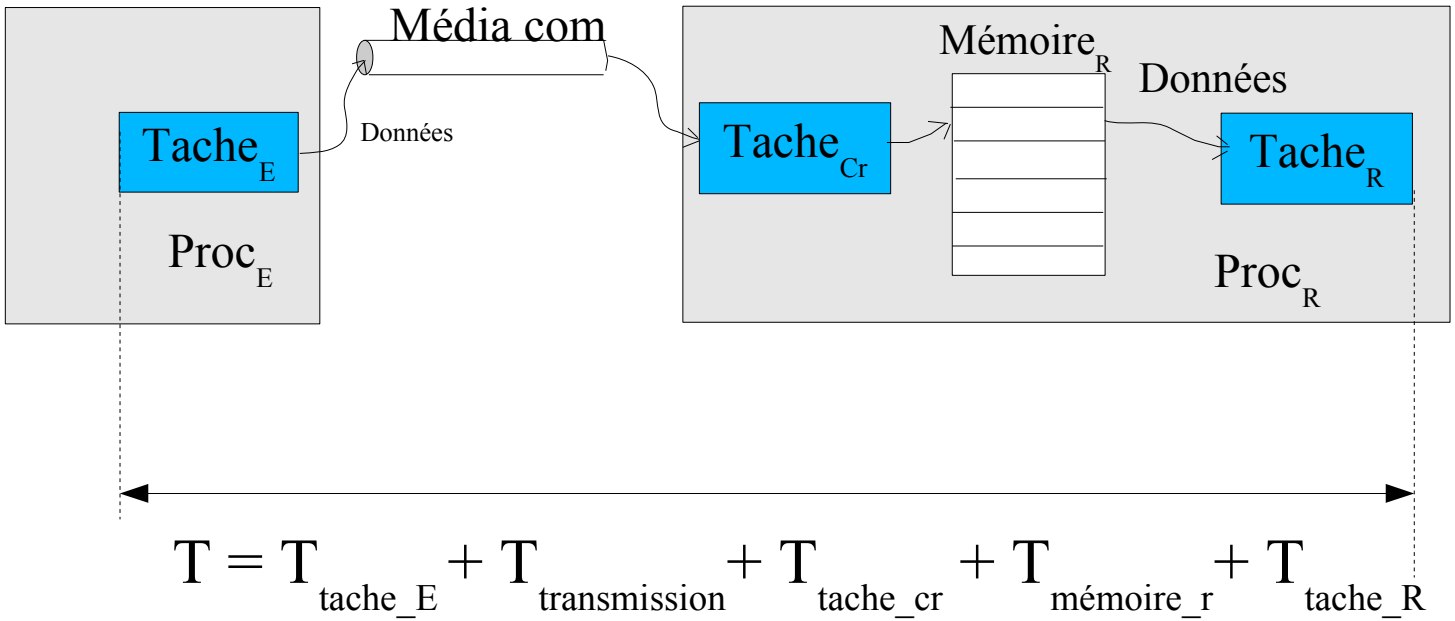
# Techniques de communication entre tâches (1)

- SYN-SYN



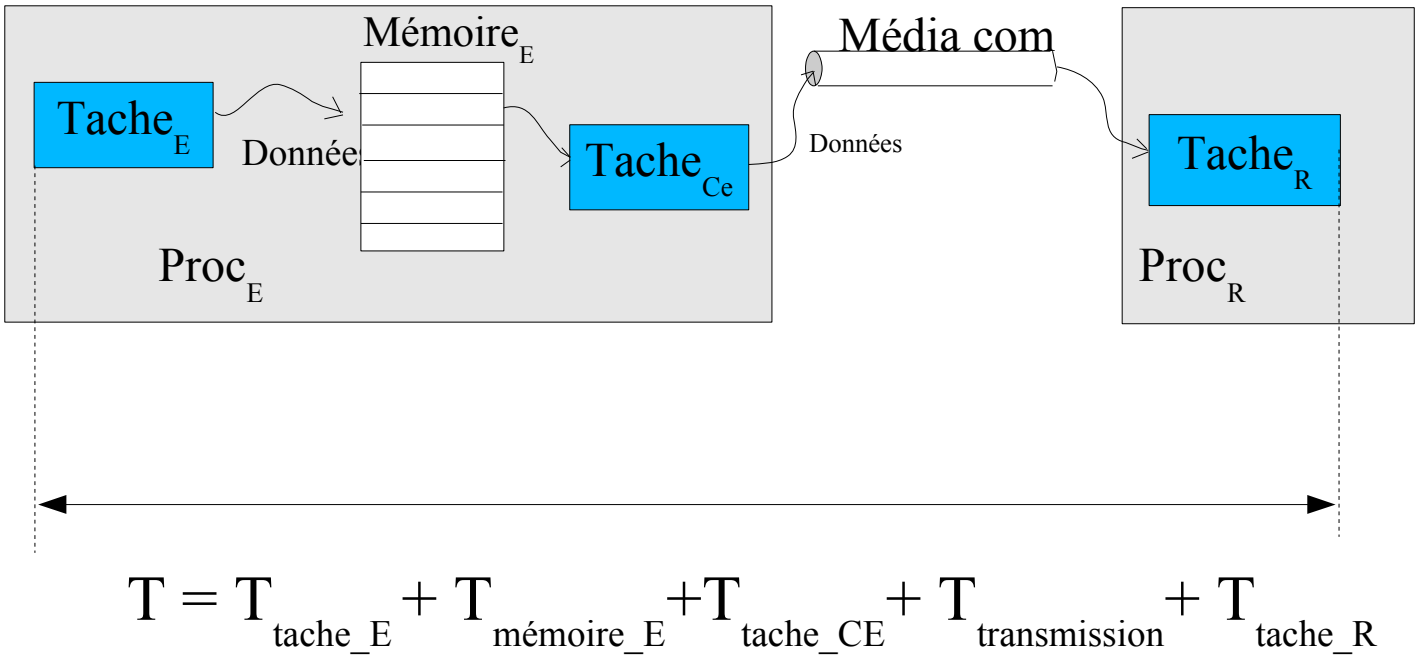
# Techniques de communication entre tâches (4)

- SYN-ASYN



# Techniques de communication entre tâches (5)

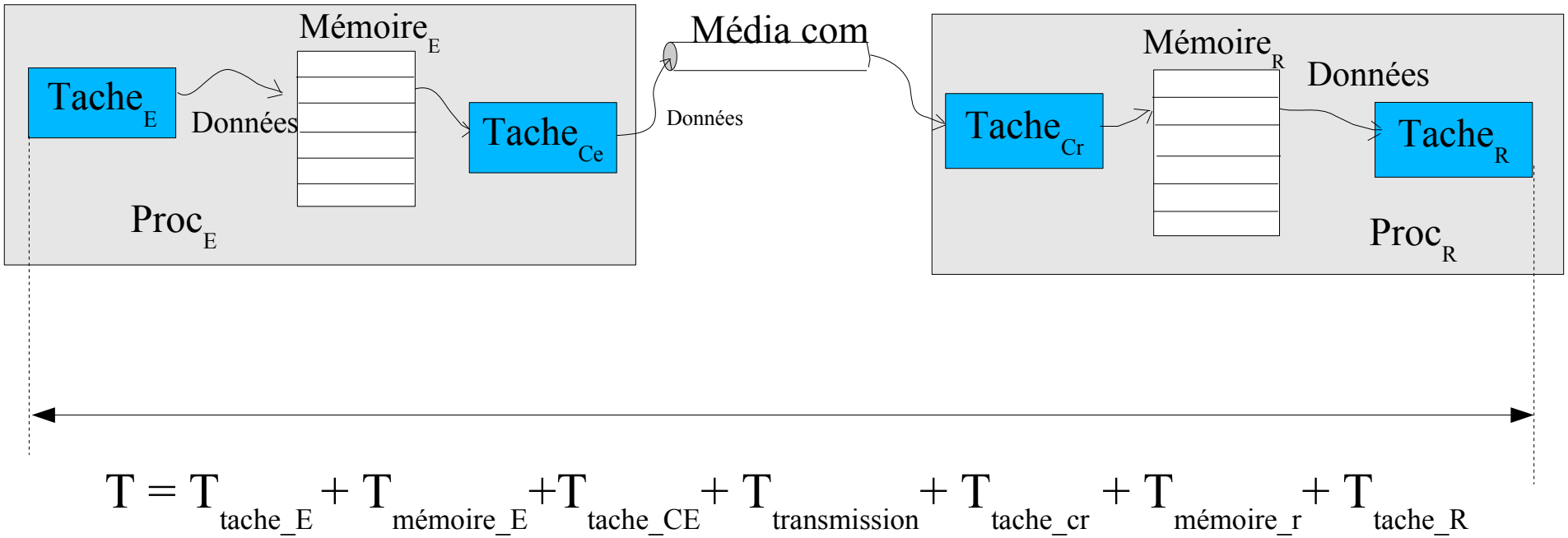
- ASYN-SYN




$$T = T_{tache\_E} + T_{mémoire\_E} + T_{tache\_CE} + T_{transmission} + T_{tache\_R}$$

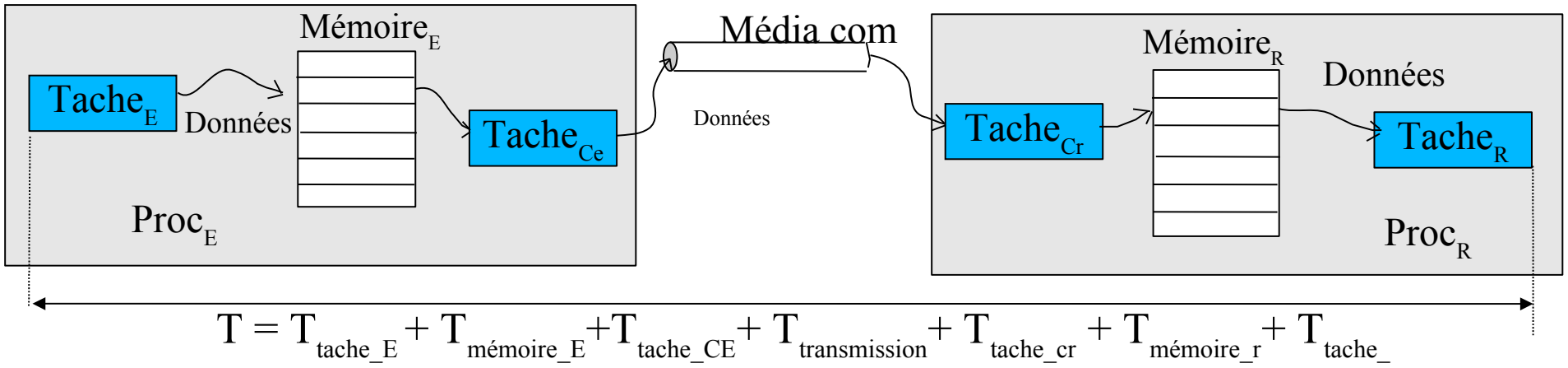
# Techniques de communication entre tâches (6)

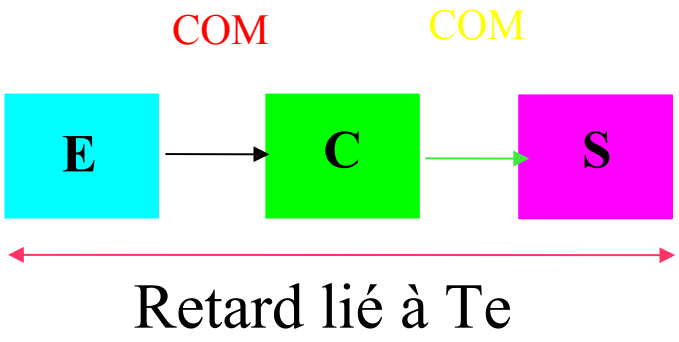
- ASYN-ASYN

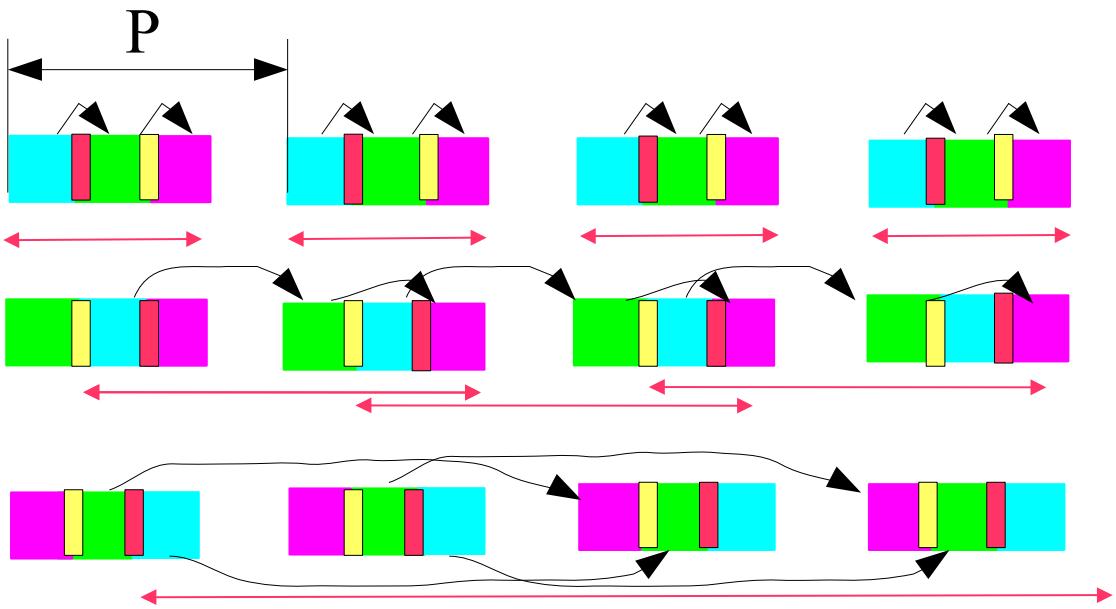



$$T = T_{tache\_E} + T_{mémoire\_E} + T_{tache\_CE} + T_{transmission} + T_{tache\_cr} + T_{mémoire\_r} + T_{tache\_R}$$

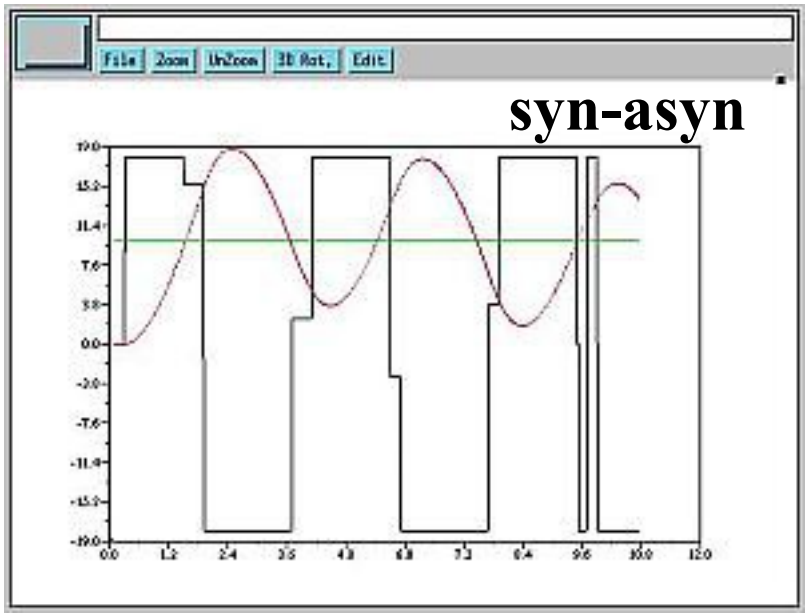
# Retard E/S

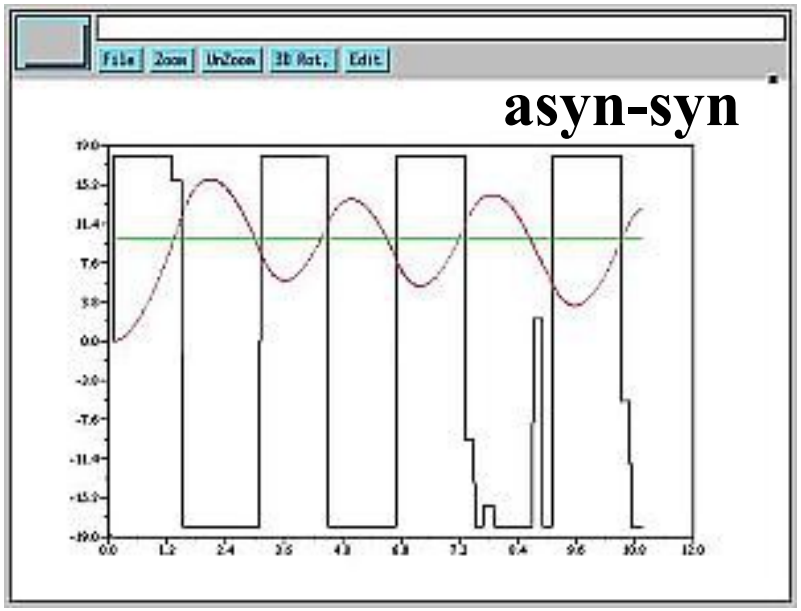

Mémoire$_E$
Média com
Mémoire$_R$
Tache$_E$
Données
Tache$_{Ce}$
Données
Tache$_{Cr}$
Données
Tache$_R$
Proc$_E$
Proc$_R$
$T = T_{tache\_E} + T_{mémoire\_E} + T_{tache\_CE} + T_{transmission} + T_{tache\_cr} + T_{mémoire\_r} + T_{tache\_}$

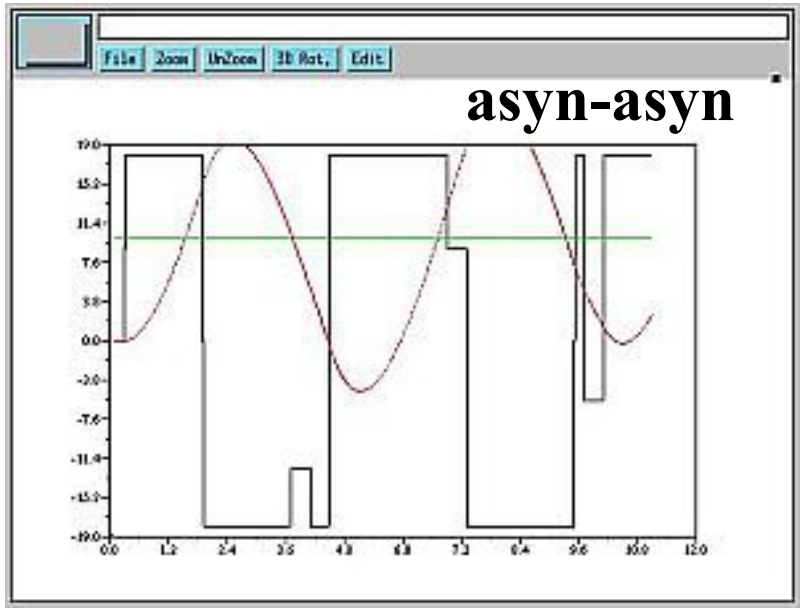

COM
COM
E
C
S
Retard lié à Te

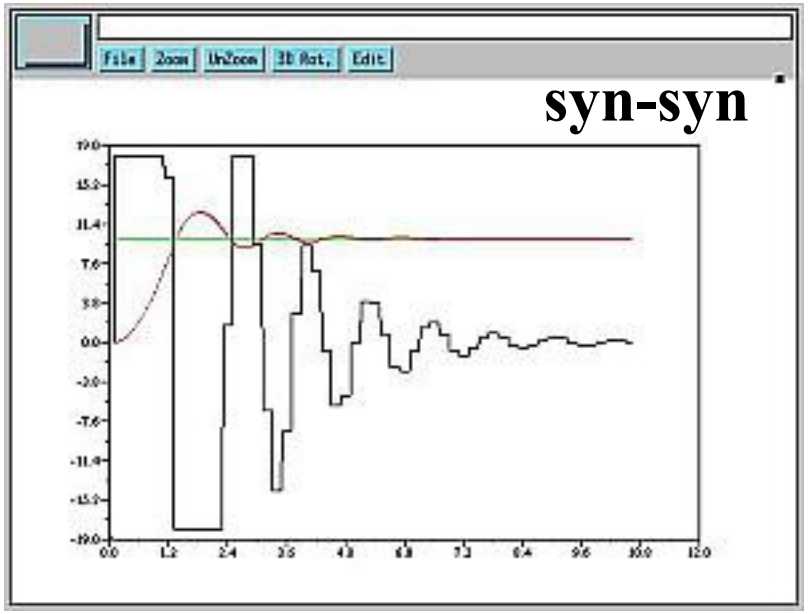

P

# Impacte des com. sur les performances

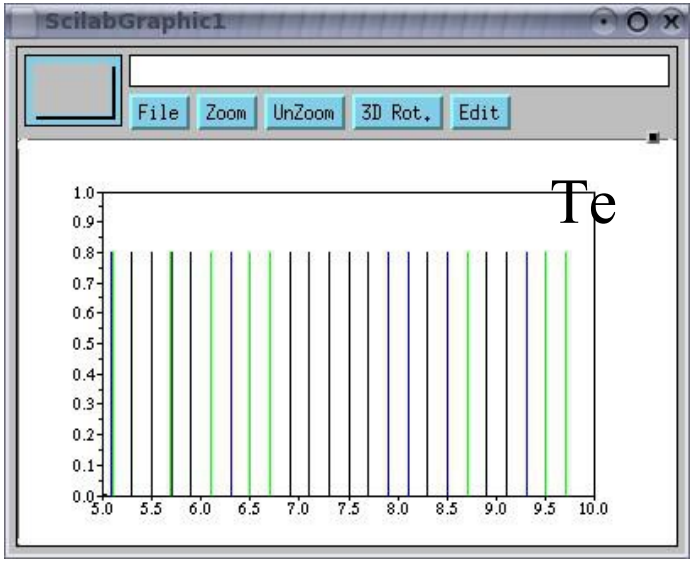


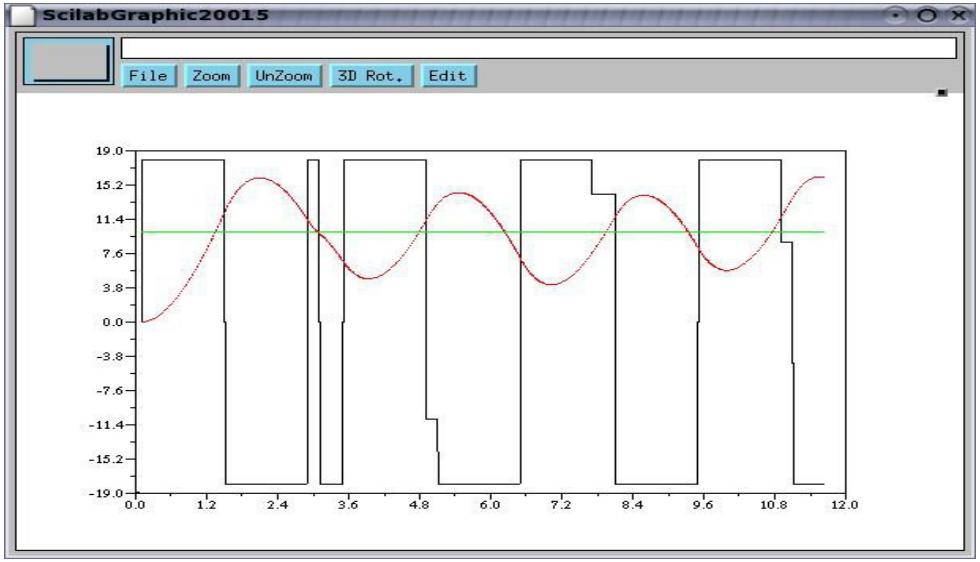



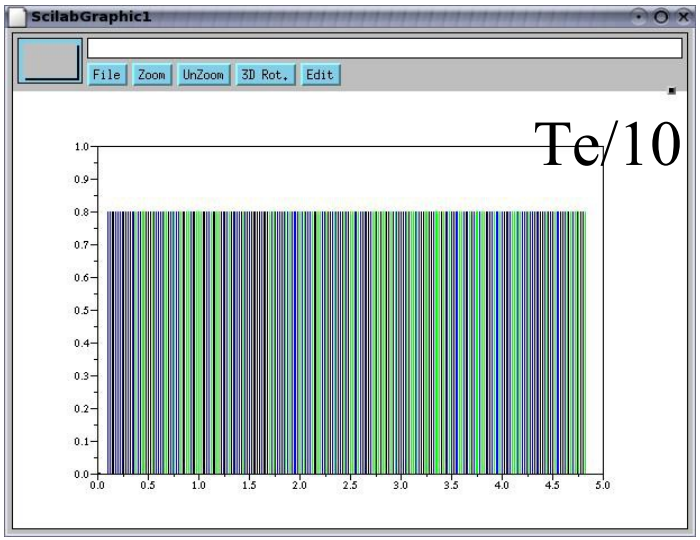



# Influence de Te (Asyn-Asyn)




# Conclusion

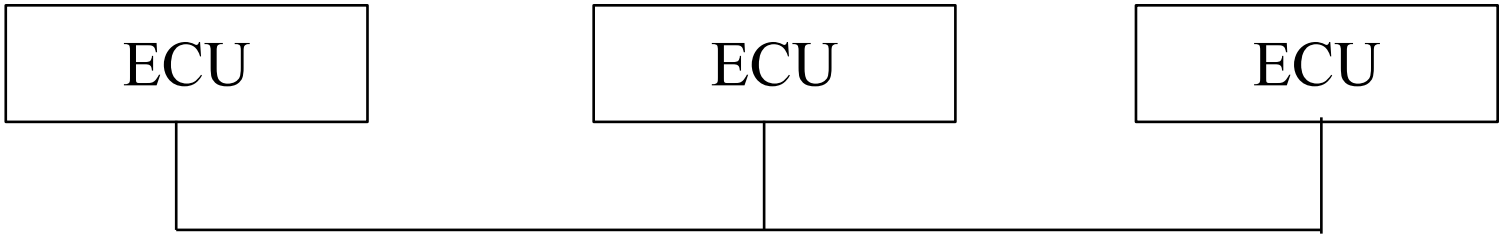
- Choix d'implantation => conséquence sur les performances du contrôleur
- Choix des Mécanismes de communication, réduire les retards :
  - synchroniser ( -- complexité, ++ charge)
  - augmenter les périodes ( ++ simplicité, -- charge)

# II. Temps Réel Distribué

1. Introduction
2. Techniques de communication entre tâches
3. OSEK-VDX
   - OSEK-OS
   - OSEK-COM
   - OSEK-NM
   - OSEK-OIL
   - Certification
4. Ordonnancement de messages
5. Synchronisation d'horloges

# Système ouvert en automobile

- Plusieurs ECUs (Electronic Control Unit) reliée entre elles par un moyen de communication
- ECUs proviennent de différents équipementiers et possèdent fonctionnellement différentes architectures internes microcontrôlées
- nécessité d'interfaces et de protocoles standards
- nécessité de disposer d'un administrateur de réseau uniforme pour garantir le sécurité du fonctionnement du système distribué



# OSEK/VDX

- Open Systems and the corresponding interfaces for Automotive Electronics
  - Génèse
    - 1993 OSEK : BMW, Daimler/Mercedes-Benz, Opel, Volkswagen, Bosh, Siemens, Université de Karlsruhe
    - 1994 Vehicle Distributed eXecutive : PSA, Renault
  - Objectif

    Standard industriel d'architecture logicielle, pour applications distribuées dans les véhicules automobiles

    Temps réel (implantation efficace), embarquabilité (mémoire réduite)

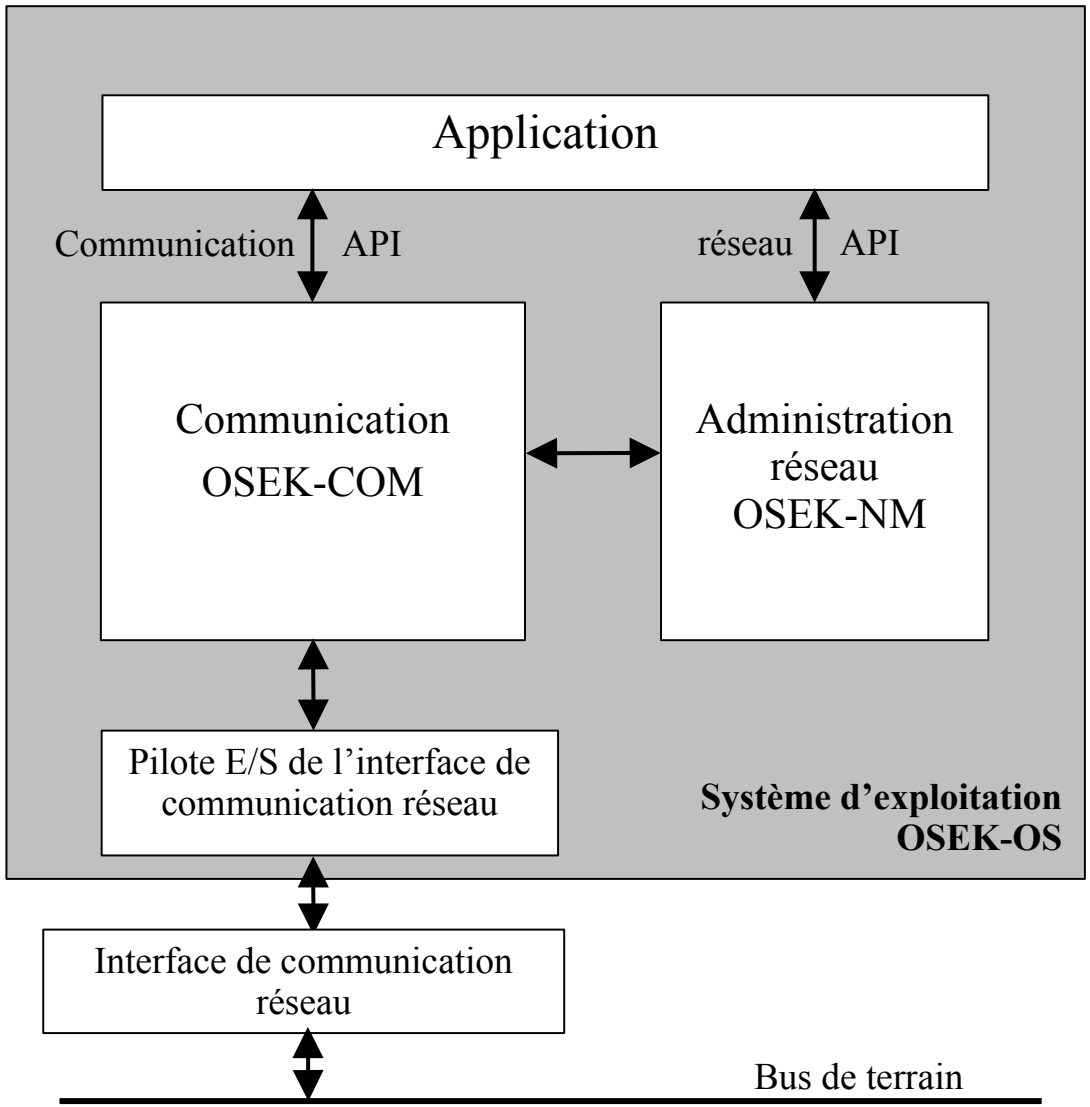
    Adaptabilité
- http://www.osek-vdx.org/

# Pourquoi OSEK/VDX ?

- Réduction significative des temps et coûts de développement
- réutilisation des logiciels existants
- intégration de modules logiciels fonctionnels
- moyens de communication entre ECUs avec différents contrôleurs et architectures de réseaux
- accroissement de performances fonctionnelles du véhicule en utilisant toutes les ressources qu ’offre une architecture distribuée
- qualité améliorée des  logiciels des ECUs
- fiabilité plus accrue et disponibilité des fonctions  réseaux
- portabilité plus élevée des logiciels applicatifs

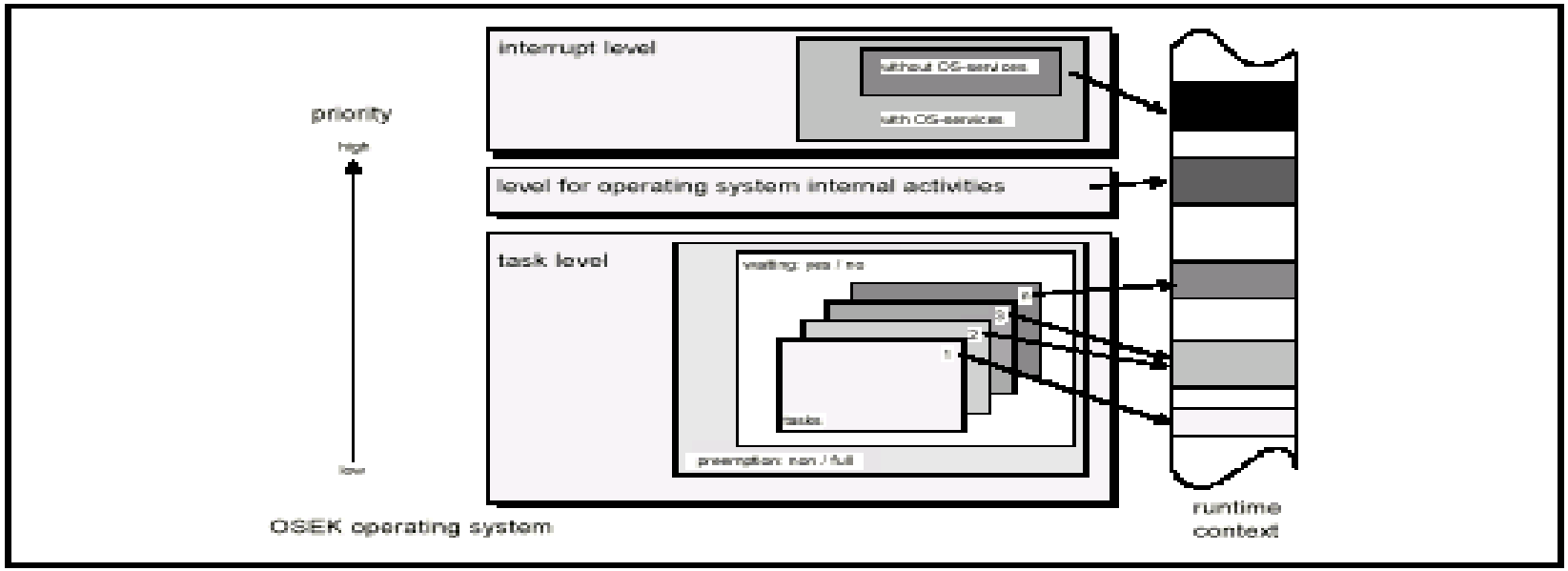
# Architecture OSEK/VDX

- **3 entités**
  - OSEK -OS
    Operating System
    v2.2.1 jan 2003
  - OSEK – COM
    Communication
    v3.0.1 jan 2003
  - OSEK - NM
    Network Management
    v2.5.2 jan 2003



# OSEK - OS

- Système d'exploitation temps réel
  - supporte temps critique pour tout hardware
  - 3 niveaux de traitement
    - niveau d 'interruption
    - niveau activités du système d'exploitation
    - niveau pour la tâche



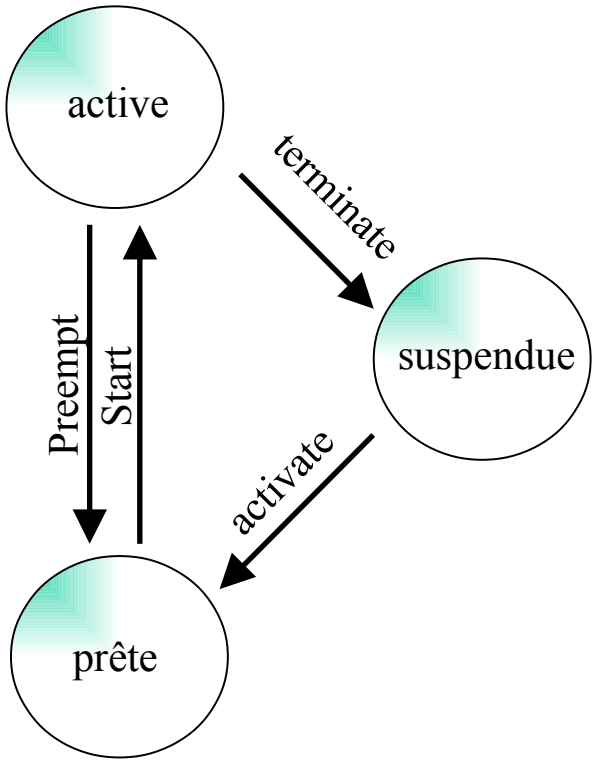
# OSEK – OS : Classes de conformance

- Répondre aux différents besoins de fonctionnalité et de capacité de l'OS
- Compatibilité ascendante entre classes
- Classes Basiques BCC
  - BCC1 tâches de type basique, une requête par tâche, une tâche par priorité
  - BCC2 : BCC1 + plus d'une tâche par priorité
  - BCC3 : BCC2 + autorise plusieurs requête par tâches
- Classes Etendues ECC
  - ECC1 comme BCC3, mais tâches étendues sans requêtes multiples
  - ECC2 : ECC1 + requêtes multiples des taches étendues

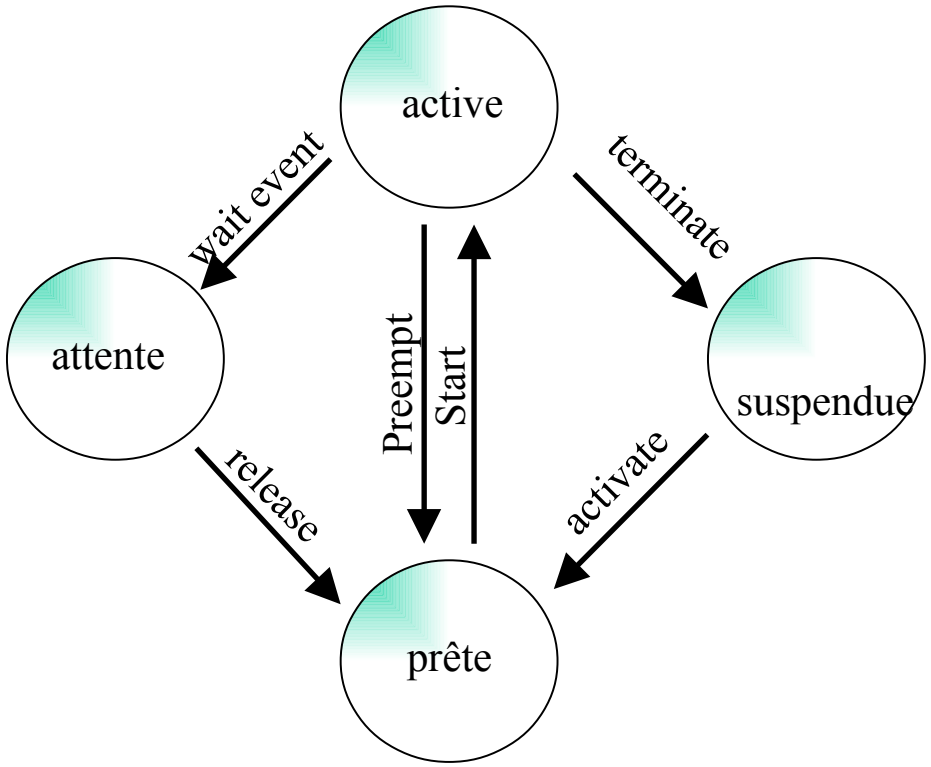
# OSEK – OS : tâches

➔ 2 types :

BASIQUE




ETENDUE




➔ Politique d'ordonnancement : non-préemptive, préemptive, mixte

# OSEK - OS - Services

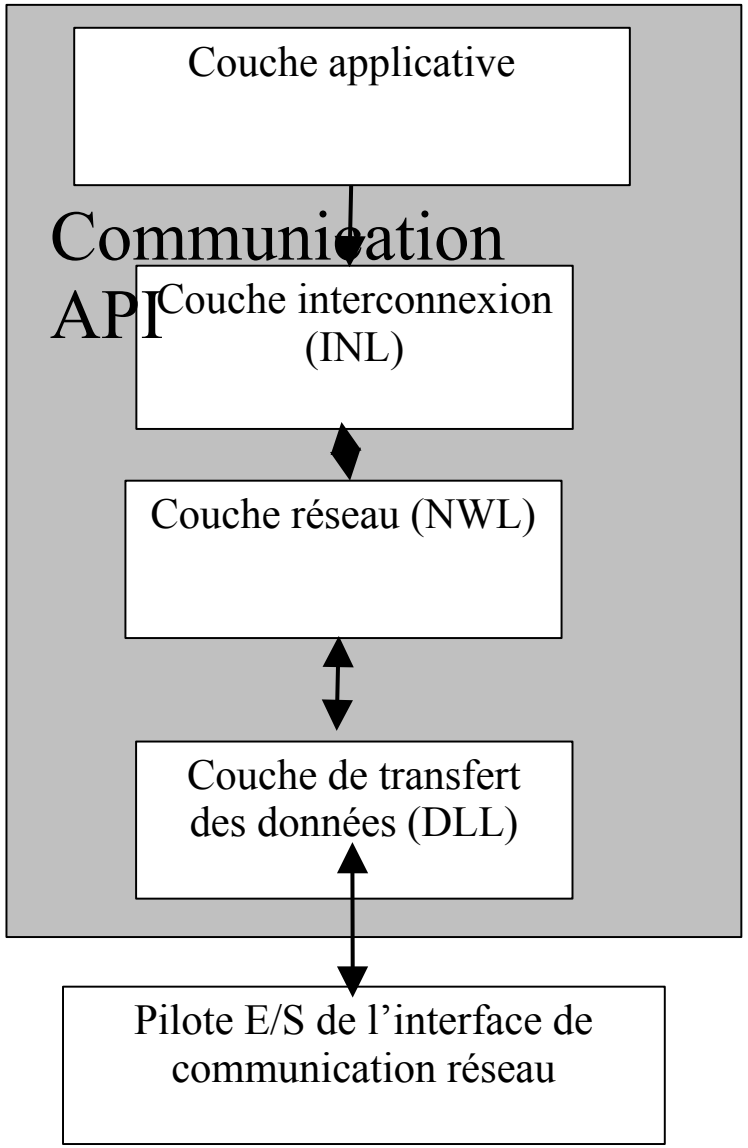
- Services système -> applicatif
  - Gestion de tâches
    - dés/activation, état, commutation
    - Ordonnancement priorités fixes
  - Gestion des IT
  - synchronisation événementielle
  - Gestion accès concurrent à des ressources par exclusion mutuelle avec protocole d'héritage de priorité plafond
  - compteurs et alarmes
    - événements récurrents / interruption horloges, capteurs
  - mécanismes de traitement d'erreur
  - Mécanismes de déboggage (non standardisé)

# OSEK - COM

- Interface de communication standardisé (API)
  - Entre ECUs
  - Au sein d'un ECU
  - Entre l'ECU et les périphériques
- Avantages
  - Indépendance vis à vis du hardware
  - Réduction du temps de développement
  - Assure la cohérence des données
  - Portabilité

# OSEK - COM – Couches

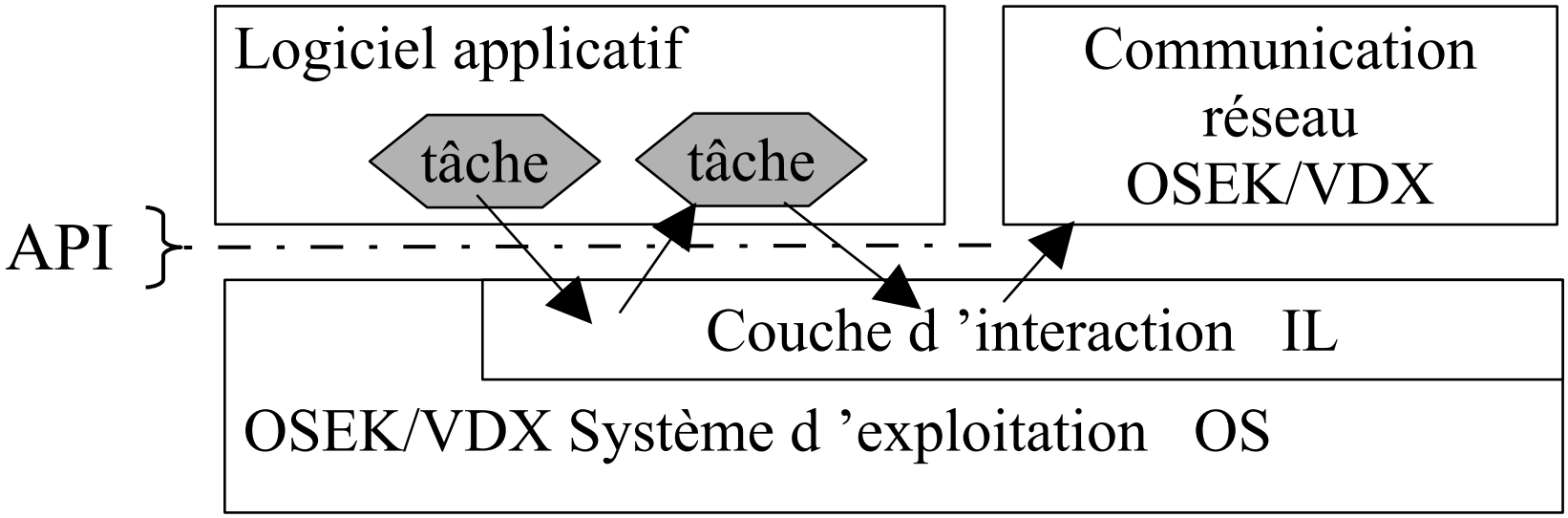
- **couche de communication de données -DLL**
  - Gestion de la communication matérielle - système du bus et pilotes associés
- **couche réseau – NWL**
  - échange et dé/assemblage en segments compatibles avec couche DLL et accusé de réception
- **couche d'interconnexion - IL**
  - interfaçage avec le programme d'application - API



# OSEK – COM – Modèle d'implantation

- Communication inter-tâches
  - Tâches au sein d'une même ECU
    géré par l'OS localement
  - Tâches sur des ECUs différents

    OS local + OSEK COM



# OSEK-COM Services

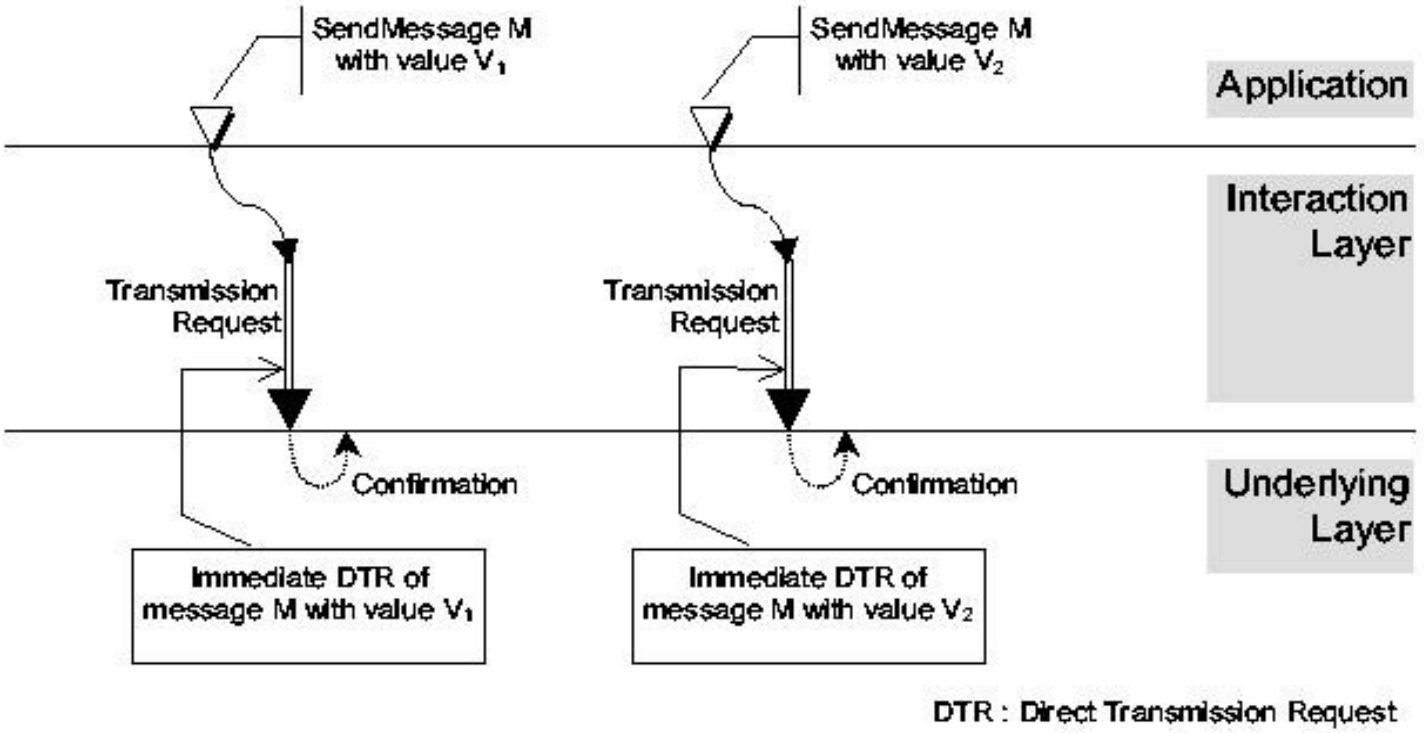
- Communication interne
  - Couche IL rend le message directement disponible pour la tâche réceptrice
- Communication externe
  - Couche IL assemble un ou plusieurs messages dans un I-PDU (Interaction Layer Protocol Data Unit)
  - Communication non bloquante
- Message m:n (émis par m tâches et reçus par n tâches)
- Messages bufferisés (FIFO): consommés (retirés) à la reception
- Messages non bufferisés : écrasés par nouvelles valeurs

# OSEK - COM – Services (2)

- Transmission Directe, Périodique ou Mixte
- Monitoring des échéances temporelles (reception et émission)
- Gestion d'erreurs
- Filtrage de messages
- Initialisation/arrêt des communications
- Configuration des messages définie statiquement à la conception

# OSEK-COM – Modes de transmission

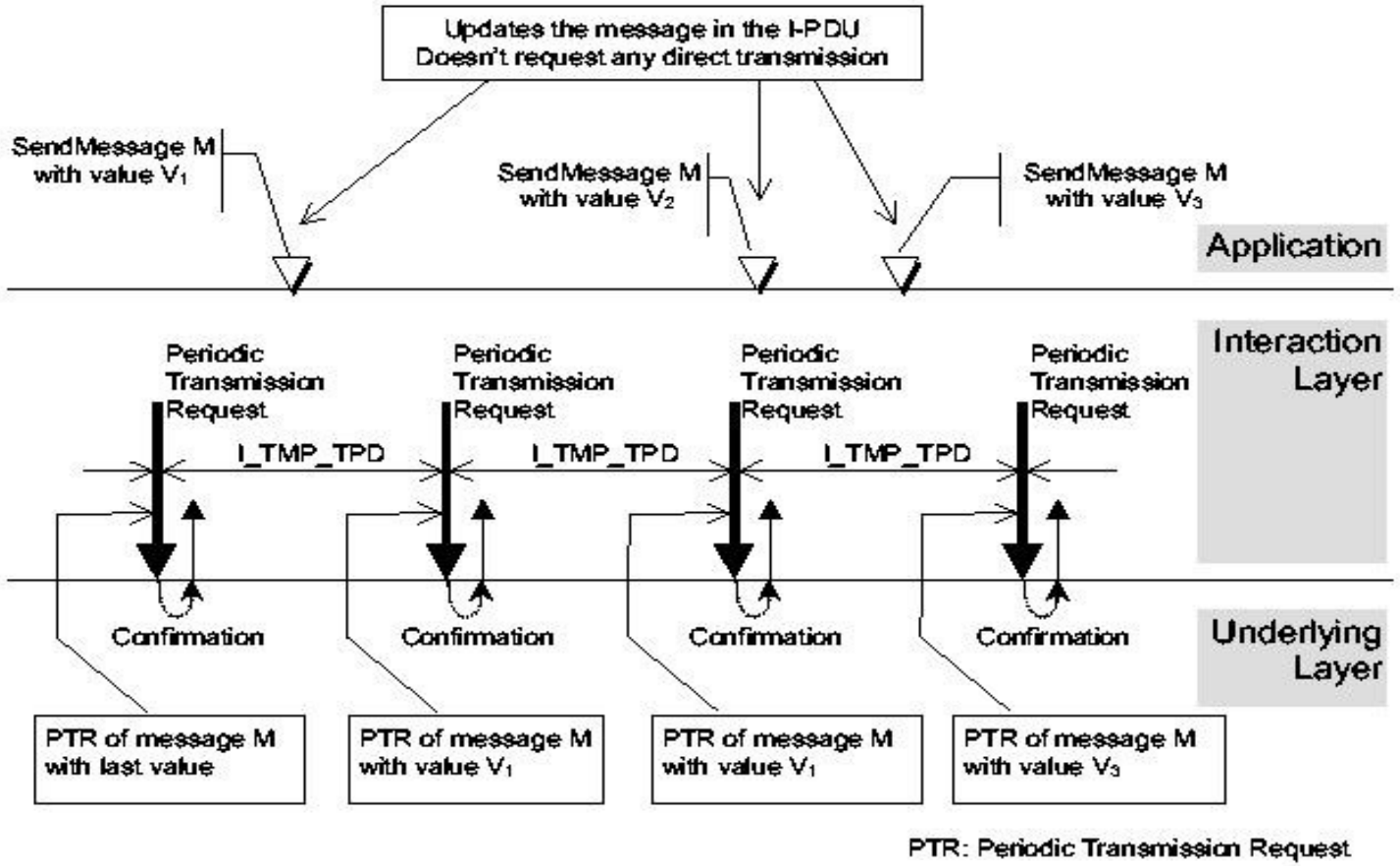
- Transmission directe




DOCUMENT OSEK-COM SPECIFICATION

# OSEK-COM – Modes de transmisssion (2)

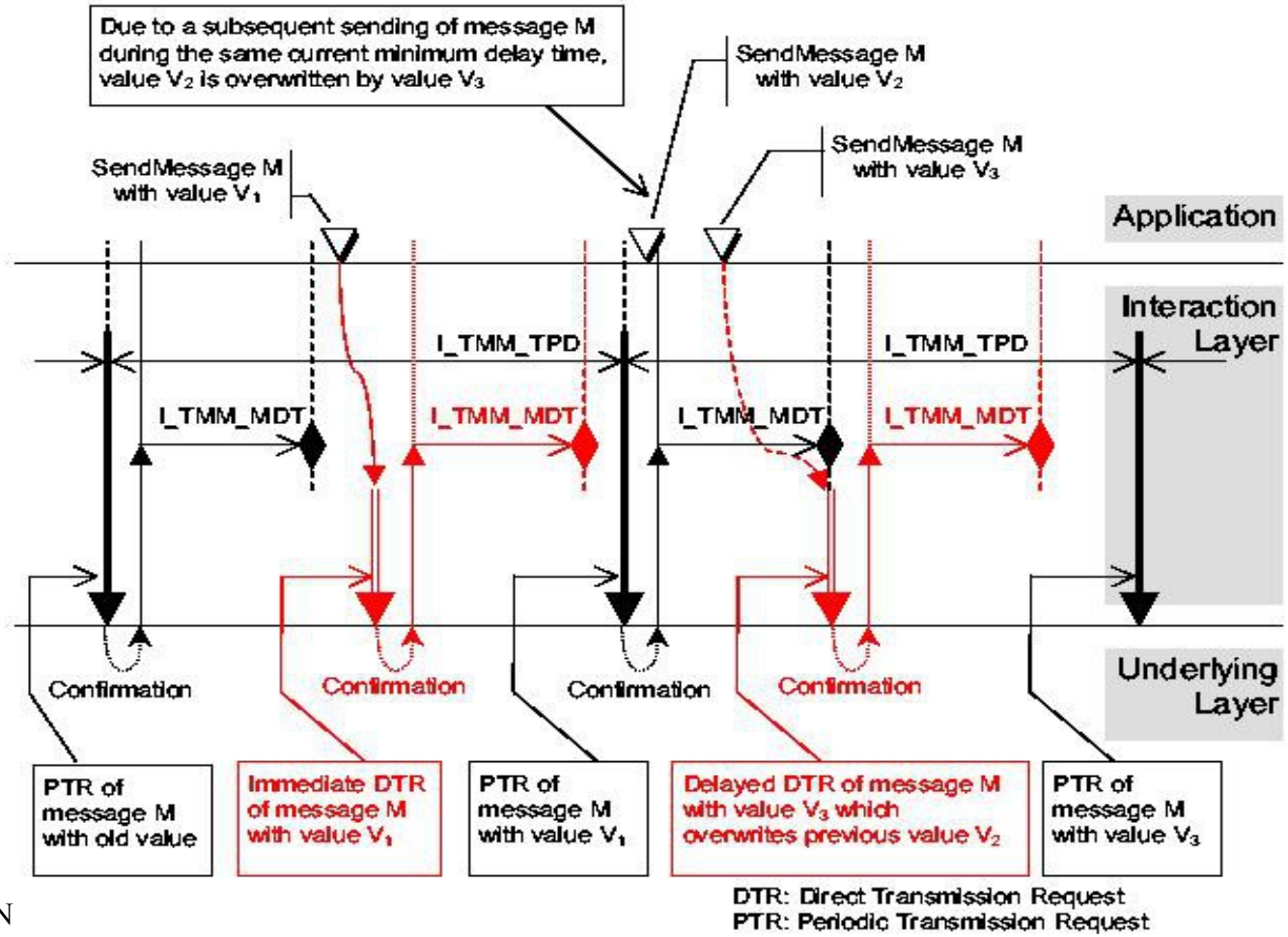
- Transmission périodique




DOCUMENT OSEK-COM SPECIFICATION

# OSEK-COM – Modes de transmission (3)

- Transmission mixte

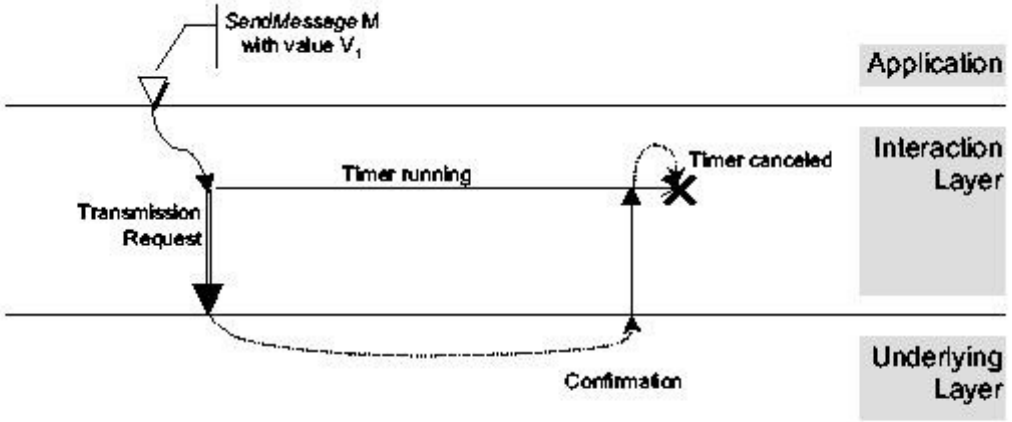



DOCUMENT OSEK-COM SPECIFICATION

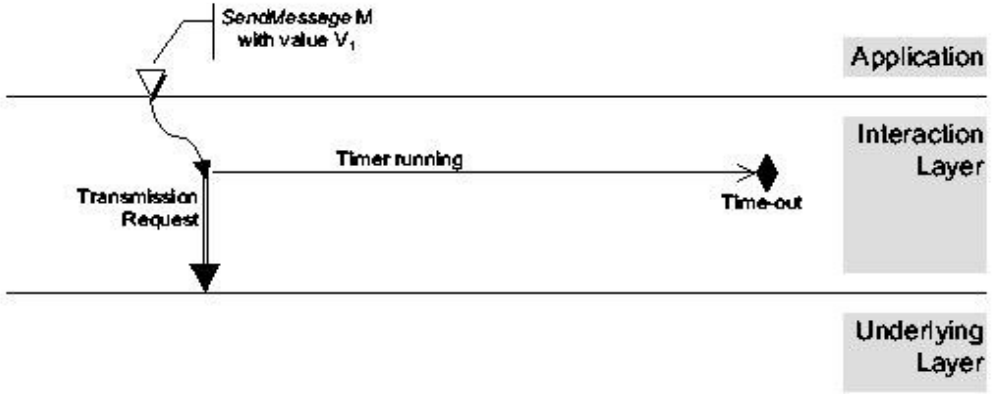
# OSEK-COM – Monitoring

- Transmission directe

OK




ERREUR

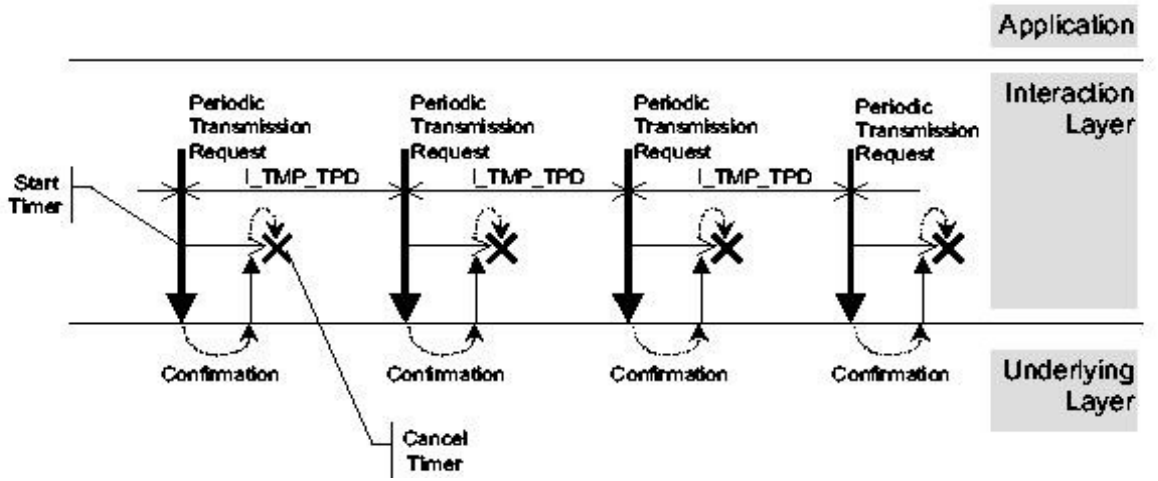



DOCUMENT OSEK-COM SPECIFICATION

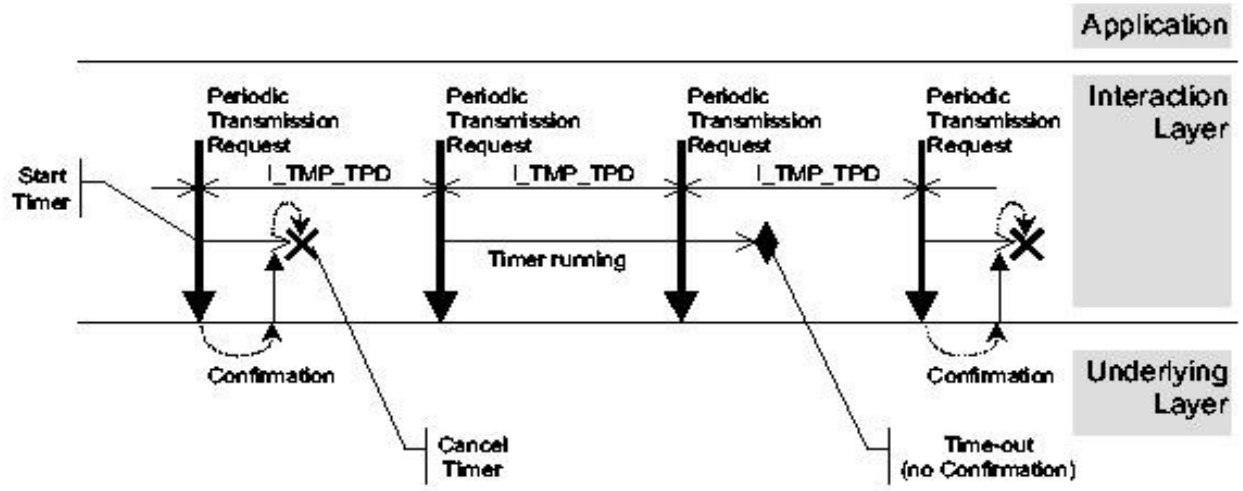
# OSEK-COM – Monitoring (2)

- Transmission périodique

OK




ERREUR

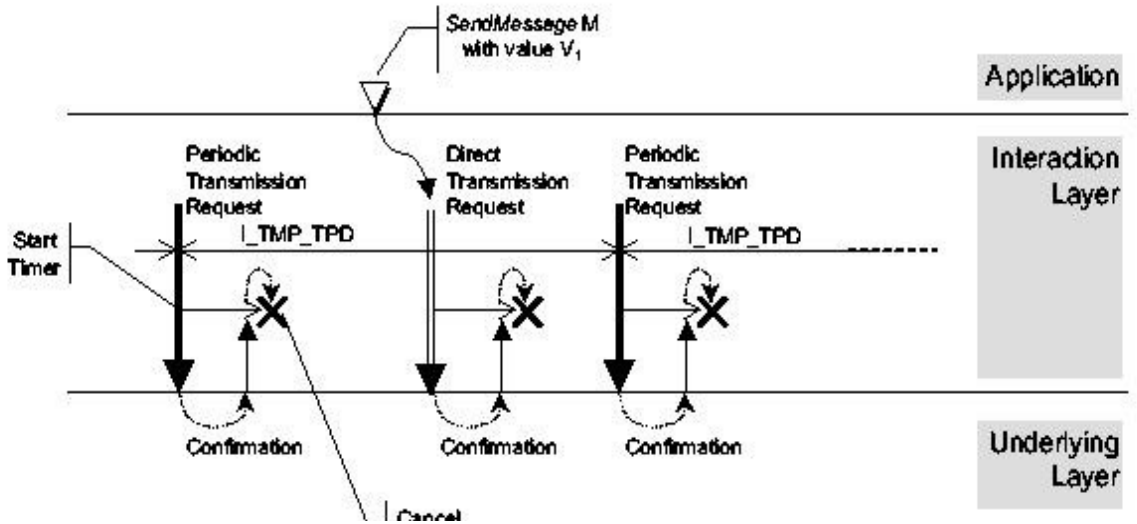



DOCUMENT OSEK-COM SPECIFICATION

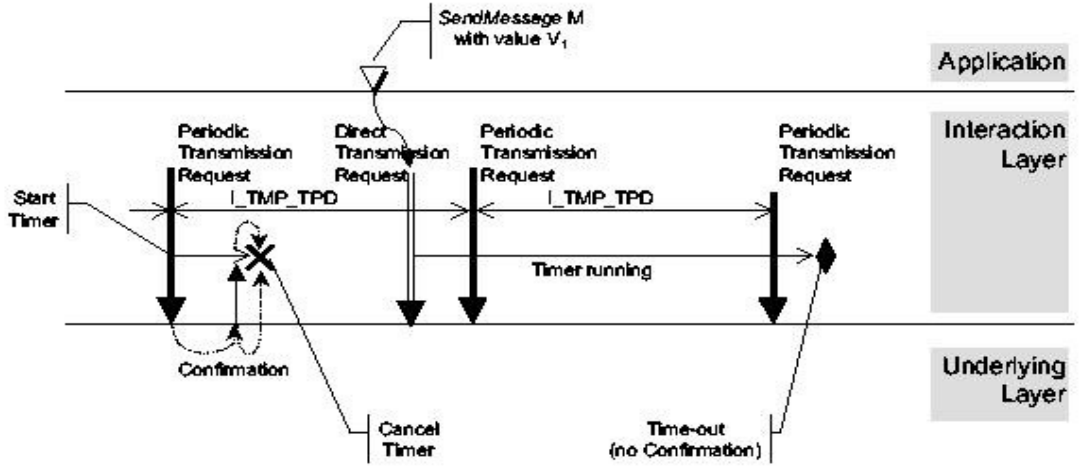
# OSEK-COM – Monitoring (3)

- Transmission mixte

OK



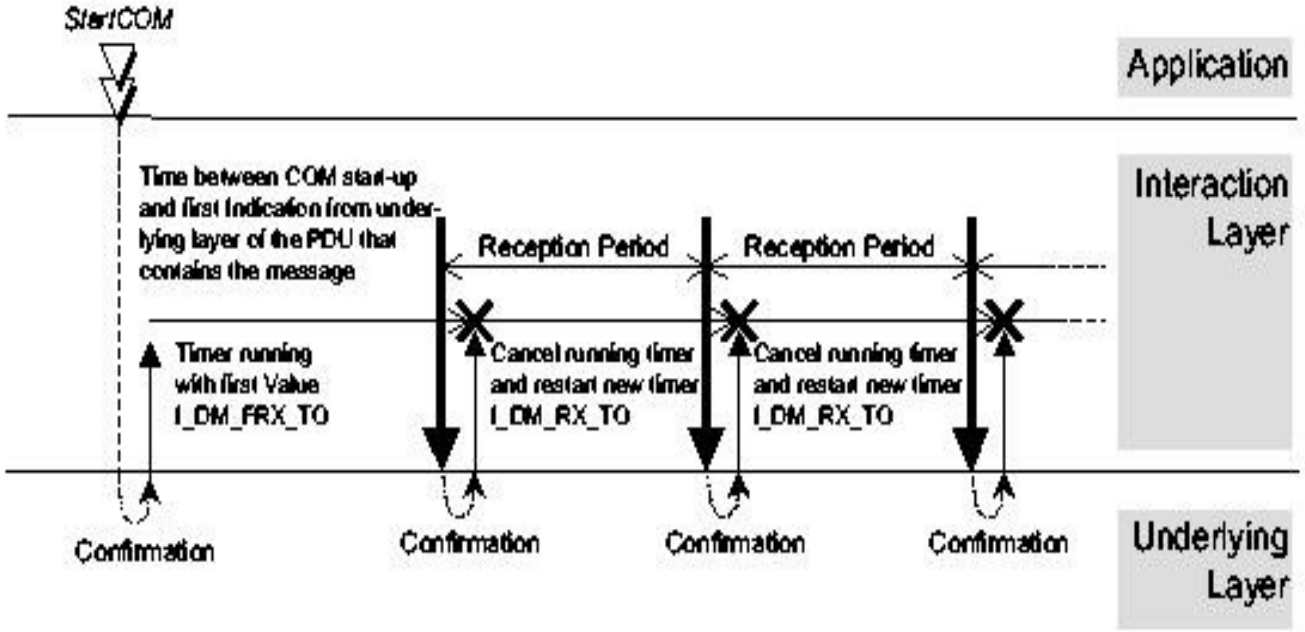

ERREUR




DOCUMENT OSEK-COM SPECIFICATION

# OSEK-COM – Monitoring (3)

- Réception périodique



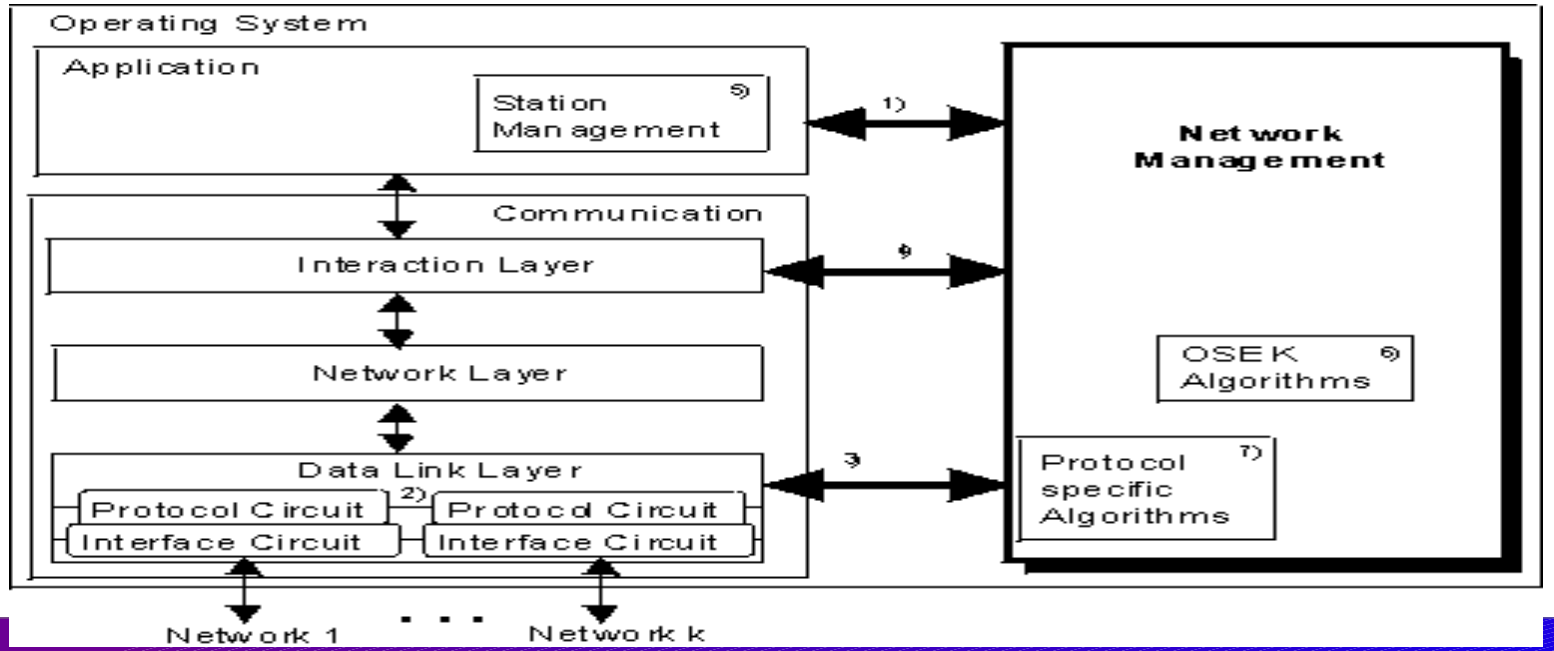

DOCUMENT OSEK-COM SPECIFICATION

# OSEK-COM - Classes de conformance

- CCCA
  - Communications internes
  - Messages non bufferisés
  - Services de monitoring réduits
- CCCB
  - CCCA + messages bufferisés + monitoring complet
- CCC0
  - CCCA + communications externes
- CCC1
  - Implementation complète de tout les services

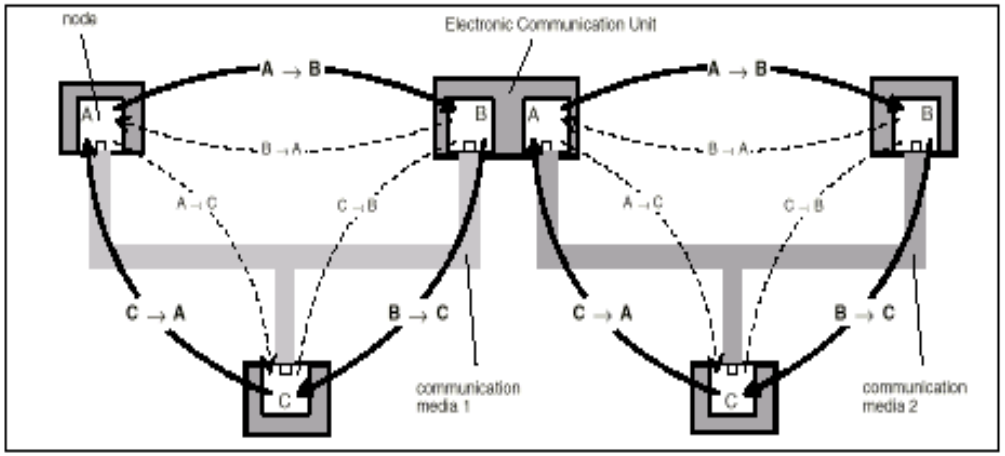
# OSEK - NM

- Administration réseau
  - tâches d ’administration
    - initialisation des communications réseau
    - monitoring de la configuration réseau
    - administration des modes opératoires locaux et globaux des stations du réseau
    - support au diagnostic



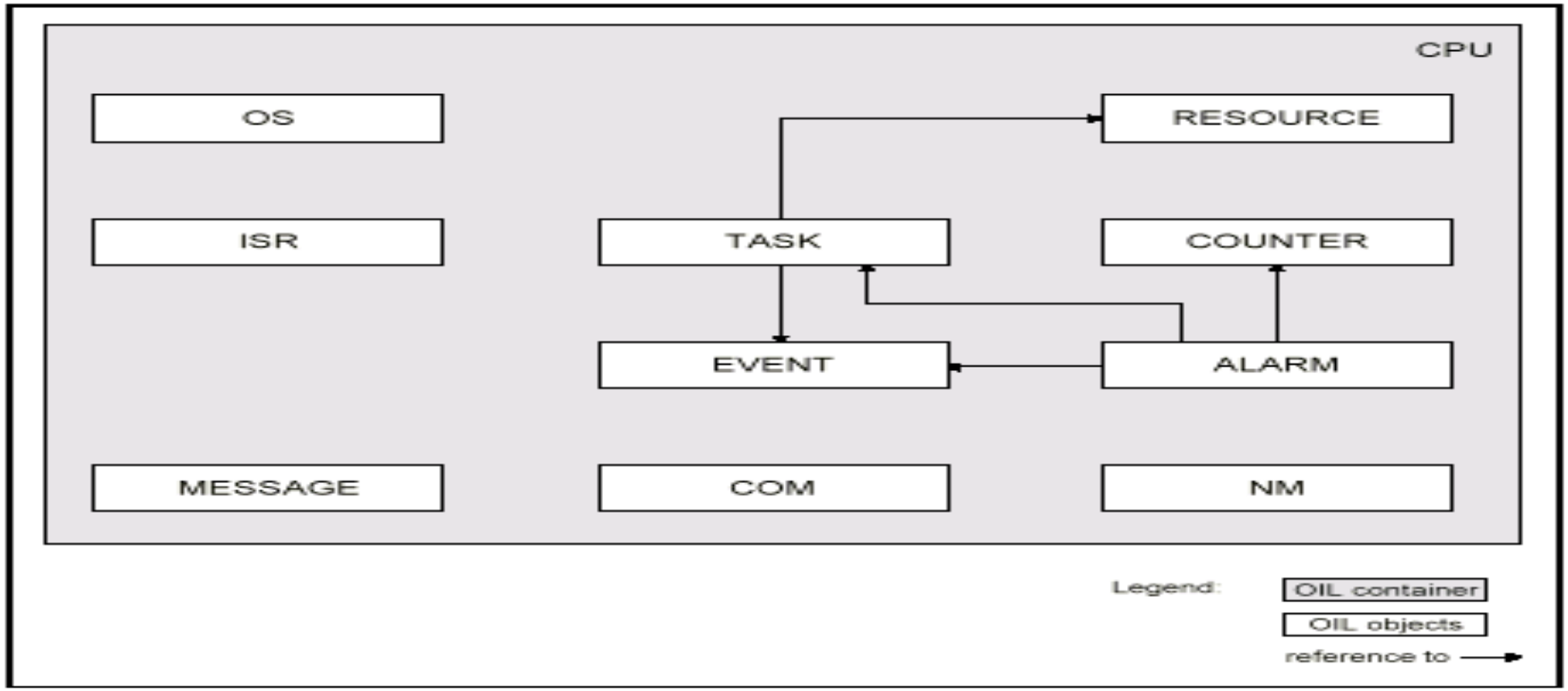
# OSEK - NM - Monitoring

- 2 mécanismes
  - contrôle direct
    - chaque station émet un message explicite « I am alive »
    - ordre d ’émission selon un anneau logique
  - contrôle indirect
    - écoute et analyse de messages spécifiques émis périodiquement par les stations
    - réception de tels messages sont interprétés comme indication d ’existence de la station
- Déconnexion de station
  - si retard (timeout) trop important
  - reconfiguration de l ’anneau logique
  - indication à l ’application



# OSEK - OIL

- Osek Implementation Language
  - Définition formelle d'une implémentation OSEK
- Objets Standards OIL



# Certification

- Projet ESPRIT : Modistarc
  - spécification de la procédure de certification
    - outils
    - modèles de référence
    - plans de tests
  - définition de critères de performance et méthode de mesure
    - compte rendu de performance pour chaque implémentation OSEK / VDX
  - création et organisation d'une autorité de certification

# II. Temps Réel Distribué

# Ordonnancement de messages

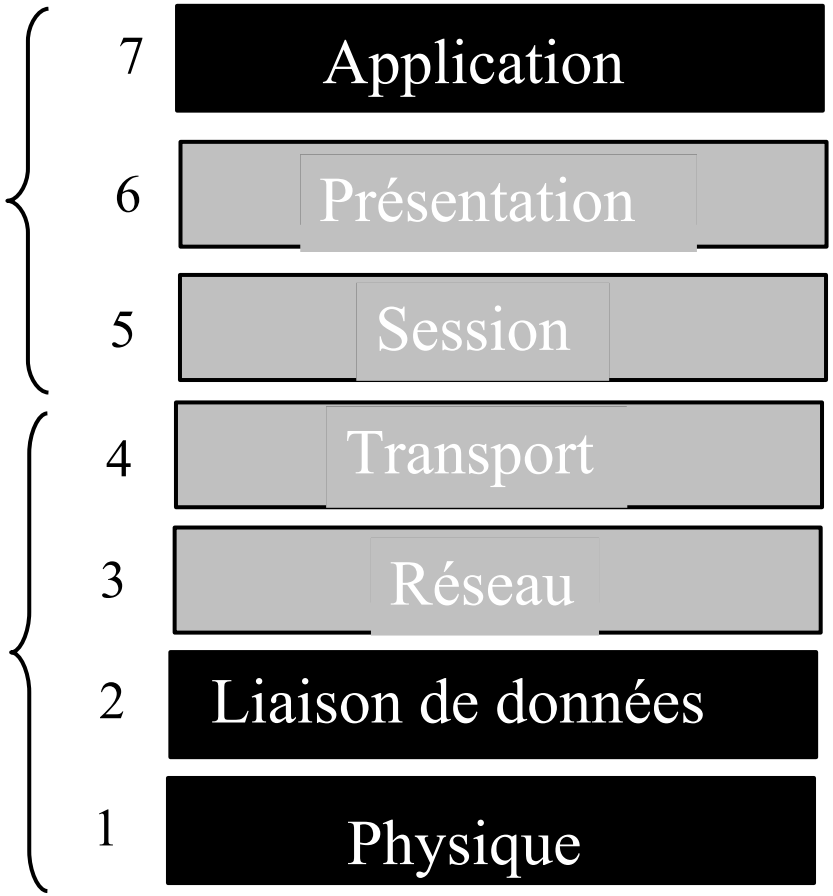
- Choisir un média de communication : réseau temps réel
- Définir les messages : fonction du placement des tâches
- Gérer le hardware : OS (cf. OSEK), écriture de drivers
- Garantir les contraintes temporelles de l'application : **Choisir un protocole de communication adapté au réseau choisi, permettant de garantir l'ordonnançabilité de l'ensemble des messages**

# Propriétés des réseaux de communication TR

- Borne maximale pour le délai de transfert,
- Faible gigue,
- Haut degré d'ordonnançabilité,
- Possibilité d'échanges périodiques et/ou apériodiques de messages,
- Stabilité,
- Overhead faible,
- Adaptabilité,
- Fiabilité
- ...

# Rappel: modèle OSI

- Application
- Liaison de données
  - MAC (Medium Access Control)
  - LLC (Logical Link Control)
- Physique



# Caractéristiques des messages

- Criticité : contrainte stricte, relative, sans contrainte
- Longueur
- Type : périodique, apériodique
- Relations entre messages : précédence, synchronisation
- Destinataires : 1:1,1:N

# Hypothèses d'ordonnancement

- Messages : apériodiques, périodiques (avec échéance = période), indépendants
- Protocole : sur couche MAC
  - Multiplexage fréquentiel: émission sur une bande de fréquence réservée
  - Multiplexage temporel: émission dans un intervalle de temps réservé
  - Compétition: émission libre, gestion des collisions
  - Consultation: droit d'émission donné par un jeton ou une station maître du média

# Protocoles MAC (1) : Ethernet

- Protocole CSMA/CD
- Principe
  - Chaque station, si elle détecte que le bus est libre peut émettre dès qu'elle le souhaite
  - Si pendant la transmission la station détecte une collision, le message est réémis au bout d'un temps aléatoire
- Compétition

# Protocoles MAC (2) : CAN

- Protocole CSMA/CA
- Principe
  - Chaque station, si elle détecte que le bus est libre peut émettre dès qu'elle le souhaite
  - Message identifiés à priorité
  - La station qui émet le message le plus prioritaire gagne l'accès au média
  - La station qui perd l'accès au média tente de réémettre le message dès que le bus se libère
- Compétition

# Protocoles MAC (4) : MAP, Profibus, FDDI

- Protocole de bus à jeton
- Principe
  - Circulation d'un jeton dans un anneau virtuel,
  - Temps d'utilisation du bus : configuré pour chaque station
  - TTRT : temps maximal entre 2 passages successifs d'un jeton sur une station
- Consultation

# Protocoles MAC (5): boucle à jeton

- Principe
  - A l'init. mise en circulation d'un jeton de faible priorité
  - Lorsqu'une station détecte un jeton elle le piège si elle a des données à émettre de priorité supérieure ou égale
  - Lorsque la trame émise revient à sa source, la station met en service un jeton à la priorité demandée
  - Anneau physique
  - Jetons multiples : 8 niveaux de priorité (3 bits)
- Consultation

# Protocoles MAC (6) : FIP

- Principe
  - Une station arbitre de bus
  - Table de réservation du bus définie statiquement
  - L'arbitre utilise la table pour définir le message à transmettre, il diffuse le nom de l'objet à échanger
  - Le producteur transmet la valeur de l'objet, tous les consommateurs concernés le récupèrent
- Consultation

# MAC du point de vue de l'ordonnancement

- Fonction de la couche MAC
  - Arbitrage d'accès : détermine le moment où une station a le droit d'utiliser le media
  - Contrôle de transmission: détermine le temps pendant lequel la station peut utiliser le média
- Selon protocole MAC une des deux fonctions offre davantage de capacité que l'autre pour prendre en compte des contraintes temporelles

➔ **Pour un protocole MAC, trouver un algo d'ordonnancement tirant au mieux parti de ces 2 fonctions**

# MAC du point de vue de l'ordonnancement (2)

- Exemple bus à jeton
  - Arbitrage accès : droit d'accès passe de station en station selon ordre de l'anneau logique
  - Contrôle de transmission : une station émet pendant son quantum de temps alloué à la configuration
  - → **Contrôle de transmission bien adapté à la prise en compte de contraintes de temps**
- Exemple CSMA/CD
  - Arbitrage d'accès : toute station peut utiliser le média si libre
  - Contrôle de transmission : une station peut émettre sans restriction tant qu'elle détecte le média libre

→ **Aucune fonction réellement adapté à la prise en compte du temps**

# Stratégies d'ordonnancement de messages

- Stratégies
  - avec garantie (contraintes strictes)
  - meilleur effort (contraintes relatives)
- Inspirées des stratégies d'ordonnancement de tâches (RM, EDF ...)

Pas de préemption sur la transmission d'une trame

# Propriétés souhaitées d'une stratégie d'ordonnancement

- Taux Utilisation Potentielle (TUP) élevé: TUP =/ si l'algorithme permet d'ordonnancer des messages périodiques tant que le taux d'utilisation total est </ . PCTUP = pire des cas du TUP, plus petite borne supérieure du TUP
- Robustesse
- Tolérance aux fautes
- Possibilité de transfert de messages sans contrainte stricte
- Overhead minimal

**➔ Compromis en fonction de l'application**

# Ordonnancement de messages périodiques

- Échéance = période
- Contraintes strictes ⇒ stratégie avec garantie

1. MAC à contrôle d'accès décentralisé
   a) Approche fondée sur l'arbitrage d'accès : Rate Monotonic
   b) Approche fondée sur le contrôle de transmission
2. MAC à contrôle d'accès centralisé (FIP)

# MAC à contrôle d'accès décentralisé (1) : Rate Monotonic

a) Utilisation de l'arbitrage d'accès :

- Rate Monotonic
- Principe : priorité du message fonction de la période
  - efficace sur réseaux à mécanisme global de priorités ( boucle à jeton ou CAN )
  - Inefficace sur réseaux sans mécanisme global de priorités (CSMA/CD, FDDI, bus à jeton) car il doit y avoir échange de messages pour déterminer le message le plus prioritaire

# MAC à contrôle d'accès décentralisé : RM sur b. à jeton

- $PCTUP \geqslant 0.69 - Bmax / Pmin$ , Bmax = temps maximum de blocage d'un message par un autre de priorité plus faible, Pmin minimum des périodes des messages
- Robustesse : périodes modifiables tant que taux utilisation < PCTUP
- Tolérance aux fautes : propagation des fautes aux autres stations
- Messages sans contrainte stricte : priorité fonction de leur échéance
- Overhead : dépend de la longueur du réseau et de son débit

# MAC à contrôle d'accès décentralisé (3) : RM sur CAN

- Identificateurs fonctions des périodes
- PCTUP : découle de RM avec tâches $\simeq$ 69%
- Robustesse : tant que taux utilisation < PCTUP
- Tolérance aux fautes : identificateurs définis de manière statique => pas de modification de caractéristiques de messages => tolérant aux fautes
- Messages sans contraintes strictes : identificateurs priorité faible
- Overhead : négligeable (arbitrage sans destruction pendant l'émission du message)

# MAC à contrôle d'accès décentralisé (4)

## b) Approche fondée sur le contrôle de transmission

- Principalement sur réseaux FDDI et bus à jeton
- Principe : à chaque tour du jeton on alloue un certain temps Qi à chaque station i : jeton temporisé
  - Allocation pour un message complet: $Q_i = R_i$
  - Allocation équitable : $Q_i = (TTRT - \alpha)/n$
  - Allocation proportionnelle: $Q_i = (TTRT - \alpha) \cdot \frac{R_i}{T_i}$
  - Allocation proportionnelle normalisée: $Q_i = (TTRT - \alpha) \cdot \frac{R_i}{T_i} \cdot \sum_{i=1}^{n} \frac{R_i}{T_i}$
  - Allocation locale : $Q_i = \frac{R_i}{\lceil T_i / TTRT \rceil - 1}$
  - Allocation locale améliorée : $Q_i = \frac{R_i}{max(\lceil T_i / TTRT \rceil - 1, 1)}$

$R_i$ = durée du message i, $\alpha$ = temps du passage du jeton, n= nombre de stations, TTRT= temps de rotation du jeton, $T_i$= période du message i

# MAC à contrôle d'accès décentralisé (5) : jeton temporisé

- PCTUP : 0 pour message complet et proportionnelle, 33% pour les autres techniques
- Robustesse : taux utilisation < PCTUP
- Tolérance aux fautes : meilleure que sur l'arbitrage d'accès, temps alloués constants, seule la station fautive est affectée
- Messages sans contrainte stricte : réservation d'une fenêtre temporelle
- Overhead : plus faible que celui engendré par l'approche fondée sur l'arbitrage d'accès

# MAC à contrôle d'accès centralisé : FIP

- Construction hors ligne d'une table de scrutation à partir des périodes de consommation des variables et messages (PPCM des périodes)
- L'arbitre de bus scrute en ligne la table pour déterminer la variable à transmettre
- PCTUP : $\sum_{i=1}^{n} Xi/Ti \leq 1$ avec Xi temps d'échange complet du message (transmission identifieur + transmisssion valeur)
- Robustesse : non, reconfiguration complète du réseau
- Tolérant aux fautes, sauf si problème lié à l'arbitre
- Overhead : important (messages de services)

# Ordonnancement de messages apériodiques

- Caractéristiques des messages non connues à priori
- Messages critiques : tailles minimales, maximales, lois d'arrivée et échéances souhaitées ⇒ $U_a$ taux moyen d'utilisation du média

1. MAC à contrôle d'accès décentralisé
   a) Stratégie de garantie
   b) Stratégie du meilleur effort
2. MAC à contrôle d'accès centralisé (FIP)

# MAC à ctrle d'accès décentralisé : stratégie de garantie

- Méthode de réservation dynamique (arbitrage d'accès)
  - demande de réservation du bus par une station à toutes les stations
  - le message est transmis dès que la réservation a réussi
  - overhead important
- Méthode du serveur périodique (contrôle de transmission)
  - message périodique fictif
  - Le temps alloué au message fictif est utilisé pour traiter les messages apériodiques

# MAC à contrôle d'accès décentralisé : meilleur effort (1)

- Earliest Deadline First sur CAN (arbitrage d'accès)
  - Première partie de l'identificateur : identificateur du message
  - Seconde partie de l'identificateur fonction de la deadline
  - Le message le plus urgent a l'identificateur le plus bas
- Minimum Laxity First sur boucle à jeton (arbitrage d'accès)
  - $L(t) = TA_{MAC} + D_{MAC} - DEP - t$
    - $TA_{MAC}$ instant d'arrivée du message au niveau MAC
    - $D_{MAC}$ deadline au niveau MAC
    - DEP délai de transmission
  - La station recevant le jeton transmet celui dont la laxité est la plus faible

# MAC à contrôle d'accès décentralisé : meilleur effort (1)

- Applications multimédia (contrôle de transmission)
  - Plusieurs versions du message avec quantité d'informations différentes (ex: image de résolutions différentes)
  - Transmission de la version la plus adaptée en fonction de la charge du réseau
  - Charge du réseau définie
    - en fonction de la file d'attente locale
    - En fonction d'un historique du nombre de messages n'ayant pas respecté les échéances

# MAC à contrôle d'accès centralisé : FIP

- Messages apériodiques non critiques
- Gestion des messages périodiques/apériodiques à l'aide de deux FIFO
  - Les demandes d'accès pour messages apériodiques sont faites par des messages  périodiques réservés et sont mis en file d'attente
  - Demandes servies lorsqu'il n'y a pas de messages périodiques à échanger

# II. Temps Réel Distribué

1. Introduction

2.OSEK-VDX

3. Techniques de communication entre tâches

4. Ordonnancement de messages

5. Synchronisation d'horloges

# Synchronisation d'horloges

- Prendre en compte de manière cohérente des événements sur des machines distinctes : activer ou terminer des tâches à des instants précis, dater des échantillons ...
- Horloges physiques locales
- Déviation des horloges (tolérances, températures,...)
- Garantir un temps global commun
  - Équipement physique de réception d'une horloge standard
  - **synchronisation physique ou logicielle pour garantir un temps global commun**

# Objectifs de la synchronisation d'horloges

- Délivrer un temps global commun approximativement identique pour tous les calculateurs malgré :
  - Les dérives des horloges physiques des calculateurs
  - Les défaillances des calculateurs et de leurs horloges
  - Les défaillances des réseaux de communication
- Temps physique (mesure) doit être une bonne approximation du temps réel

# Propriétés des horloges synchronisées

- Granulat d'une horloge physique : intervalle de temps séparant deux tops successifs de l'horloge

$$Si \quad |t_1 - t_2| < gr \Rightarrow H(t_1) = H(t_2)$$

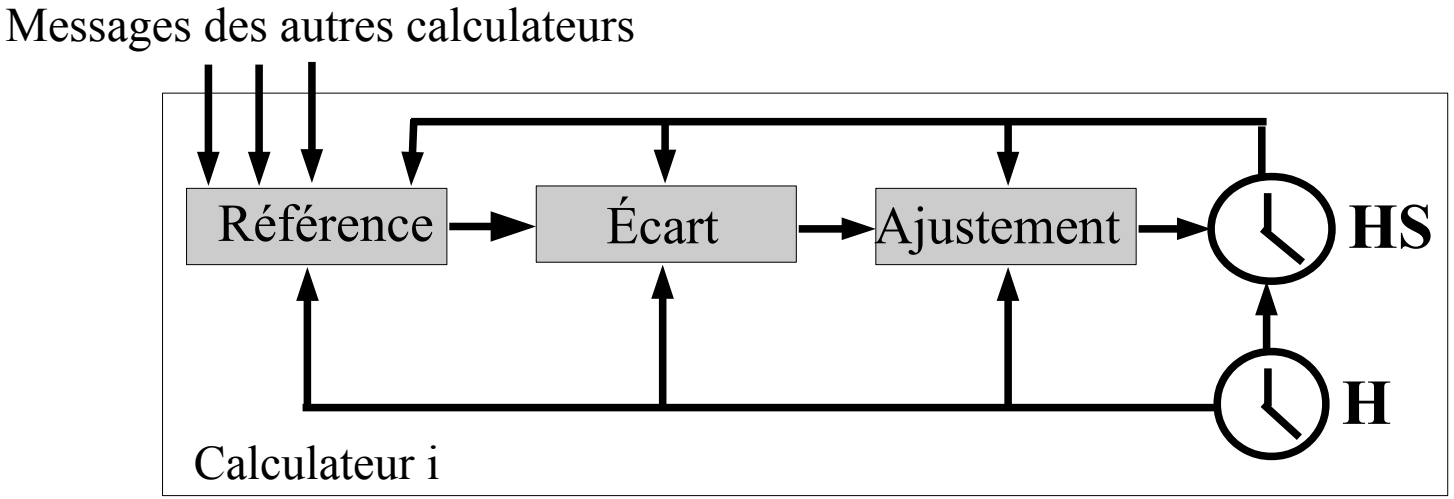
- Propriété d'accord : la différence entre deux horloges synchronisées doit être bornée (précision d'accord)

$$\left| HS_i(t) - HS_k(t) \right| \leqslant \delta$$

- Propriété d'exactitude : la déviation d'une horloge synchronisée par rapport au temps réel doit être bornée

$$\exists \rho < 1, \quad 1 - \rho \leqslant \frac{HS_i(t_2) - HS_i(t_1)}{t_2 - t_1} \leqslant 1 + \rho \,, \quad t_2 \leqslant t_1 + gr$$

# Principe de la synchronisation d'horloges




- Référence : fournit la valeur de référence pour calculer la donnée d'ajustement et détermine le moment d'ajustement
- Ecart : calcule la donnée d'ajustement, écart entre la valeur de référence et la valeur de l'horloge locale
- Ajustement : modifie la valeur de l'horloge locale au moment de l'ajustement

# Structure de la synchronisation

- Décentralisée
  - Fonctions de synchronisation dupliquées à l'identique sur chaque calculateur
  - Approche naturellement tolérante aux fautes
- Centralisée
  - Un calculateur privilégié calcule la valeur de référence pour tout le système
  - Totalement centralisée : fonction écart réalisée par le calculateur privilégié
  - Partiellement centralisé : fonction écart dans chacun des calculateurs « esclaves »
  - Approche peu ou pas tolérante aux fautes

# Critères de performance

- Précision d'accord ⇒réduire l'écart entre horloges synchronisées
- Exactitude de synchronisation ⇒ réduire l'écart entre l'horloge synchronisée et le temps réel
- Nombre de messages ⇒ minimiser le trafic induit par l'algorithme
- Tolérance aux fautes ⇒ trouver un compromis entre complexité, overhead, robustesse ...
- Temps d'exécution de l'algorithme ⇒minimiser l'overhead

# Fonction référence : principe

- Calcule la *valeur de référence* et le *moment d'ajustement*
- 4 combinaisons possibles
  - Calcul global de la référence, calcul local du moment d'ajustement : algo à **CONVERGENCE DE VALEURS**
  - Calcul local de la référence, calcul global du moment d'ajustement : algo à **CONVERGENCE DE MOMENT**
  - Calcul global de la référence et du moment d'ajustement : algo à **CONVERGENCE DE VALEURS ET MOMENT**
  - Calcul local de la référence et du moment d'ajustement : **PAS DE SYNCHRONISATION POSSIBLE**

# Calcul de la valeur de référence

- Calcul local

$$R_i(tr_i) = V \cdot j + \beta$$

V, durée d'une vague de synchronisation,
$\beta$ , temps maximum nécessaire au calcul du moment

- Calcul global

$$R_i(tr_i) = F(X_{i1}(tr_i), X_{i2}(tr_i), \cdots, X_{in}(tr_i))$$

$X_{ik}(tr_i)$ valeur de l'horloge du calc k estimée par i à $tr_i$
F, fonction de convergence (moyenne, ...)

# Lecture des horloges à distance

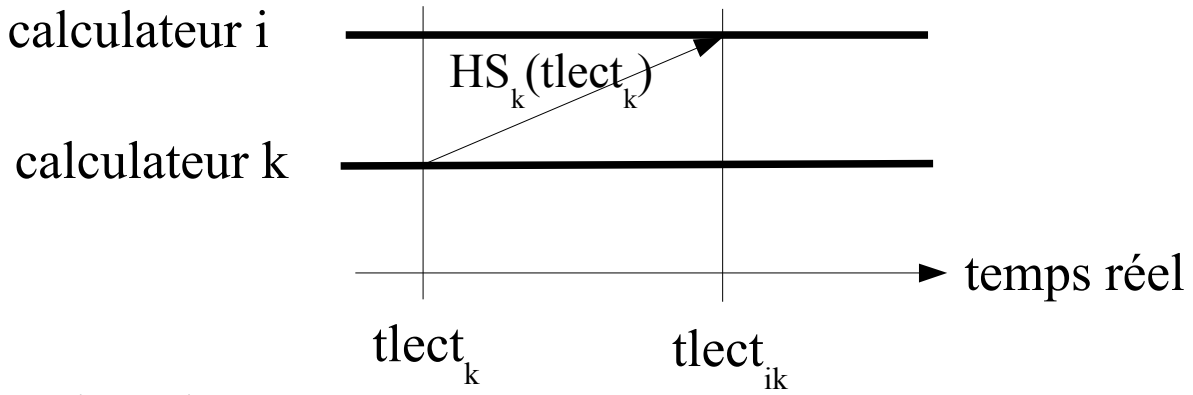
- Problème majeur de la synchronisation
- Réduire l'erreur de lecture

$$\epsilon = |HS_k(t) - HS_{ik}(t)|$$

- Techniques de lecture
  - Déterministes
  - Probabilistes
  - Statistiques

# Lecture d'horloges à distance: approche déterministe (1)

- erreur directement liée au délai de communication
- délais de com nécessairement bornés
- sans demande de lecture



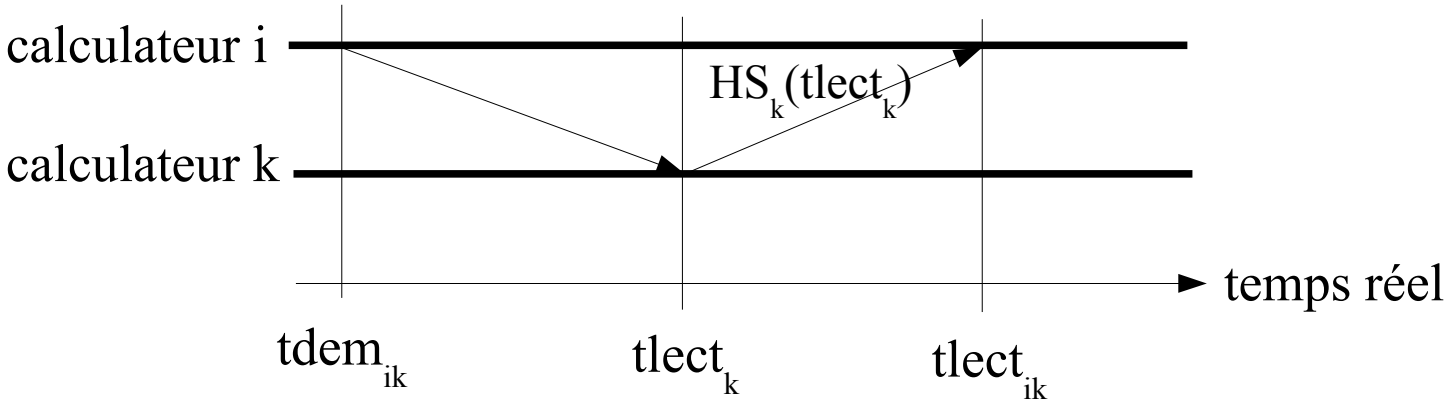

Modèle d'estimation :

$$HS_{ik}(tlect_{ik}) = HS_k(tlect_k) + \tau$$

$\tau = (D_{max} + D_{min})/2$ , délai moyen de communication

# Lecture d'horloges à distance: approche déterministe (2)

- avec demande de lecture




$$AR = HS_i(tlect_{ik}) - HS_i(tdem_{ik})$$

Modèle d'estimation :

hypothèse : délais de communication i→k et k→i identiques

$$HS_{ik}(tlect_{ik}) = HS_k(tlect_k) + AR/2$$

# Lecture d'horloges à distance: approche non déterministe (1)

- délais de communication non bornés
- précision d'accord aussi petite que souhaitée
- probabilité de succès < 1
- délais de communication considéré comme une variable aléatoire (distribution, minimum, moyenne, variance, ...)
- caractéristiques des délais utilisés pour déduire le nombre de messages pour obtenir le succès de l'algorithme avec la probabilité souhaitée
- + complexes que les algos déterministes

# Lecture d'horloges à distance: approche non déterministe (2)

- technique de limitation du délai max d'aller-retour
  - dérivé d'une lecture déterministe avec demande
  - délai max d'aller retour fixé en fonction de la précision souhaitée
  - une valeur obtenue après le délai n'est pas retenue, dans ce cas on recommence une lecture z unité de temps plus tard
  - p probabilité de dépassement du délai, (1-p) probabilité de réussite
  - r nombre maximal de tentatives ⇒ proba. réussite $=(1-p^r)$
  - choix judicieux de r et z  fonction du réseau rend la probabilité de succès proche de 1

# Lecture d'horloges à distance: approche non déterministe (2)

- Technique utilisant la régression linéaire
  - l'horloge distante que l'on cherche à lire est représentée par une fonction linéaire y=ax+b de la valeur de l'horloge locale
  - les paramètres a et b sont estimés et affinés statistiquement en prenant en compte les temps de communications mesurés
- Augmenter la précision d'accord d'un algo de synchronisation non déterministe ⇒ augmenter les communications
- Compromis entre précision d'accord, probabilité de succès et nombre de messages échangés

# Calcul du moment d'ajustement

- Calcul local

$$T_j = T_0 + j * V$$

V période de la vague de synchronisation
$T_j$ date de la jème synchronisation
$T_0$ date de la première synchronisation

- Calcul global
  - chaque calculateur émet un message pour signaler l'éminence du moment d'ajustement
  - un calculateur n'effectue l'ajustement que s'il sait qu'au moins x calculateurs sont prêts à le faire
- Calcul global centralisé
  - un calculateur privilégié diffuse périodiquement un message aux autres

# Fonction écart

- Calcul décentralisé de l'écart

$$A_i(te_i) = R_i(tr_i) - HS_i(te_i)$$

- Calcul centralisé de l'écart: le calculateur privilégié calcule l'écart et le transmet au calculateur « esclave »

$$A_i(te_i) = R_p(tr_p) - X_{pi}(tr_p)$$

Rp valeur de référence du calculateur privilégié
Xpi estimation de l'horloge du calculateur esclave

# Fonction ajustement

- Ajustement discontinu
- Ajustement continu

# Introduction à la tolérance aux pannes (défaillances) dans les systèmes répartis

# Introduction

- **Problème :** défaillance = comportement non souhaité (généralement imprévisible) d'un composant matériel ou logiciel
- **Objectif :** tolérance aux défaillances, c.à.d. transformer un comportement non souhaité soit :
  - Comportement prévisible
  - De manière à masquer le comportement non souhaité
- **Solution :** Transformer les composants d'un système en composants satisfaisant certaines hypothèses

# Défaillance

- Défaut (Fault) ! Erreur ! Défaillance (Failure)
- Défaut
  - Nature : accidentelle, intentionnelle
  - Origine : physique, humaine, interne au système, design
  - Persistance : permanent, temporaire
- Erreur : manifestation du défaut
- Défaillance
  - Sur des valeurs : le composant délivre un service ou une valeur incorrects
  - Temporelle: non satisfaction de contraintes temporelles

# Classes de défaillances

- *Défaillance par omission* : le composant omet de produire une sortie relative à une entrée
- *Défaillance franche:* défaillance par omission permanente
- *Défaillance d'arrêt:* pas d'activité observable du composant, une valeur constante est fournie en sortie
- *Défaillance byzantine :* les contraintes temporelles ou les valeurs ne sont pas toujours correctes

# Tolérance aux défaillances : principes

- Calcul d'erreurs : supprimer les erreurs si possible avant défaillance
  - Recouvrement d'erreur : ramener le système danc un état correct (état précédent ou suivant)
  - Compensation d'erreur : l'erreur est masqué (utilisation de redondance)
- Traitement de défauts : éviter qu'un défaut ne se reproduise
  - Localisation de l'erreur,
  - Masquage de l'erreur

# Redondance

- Utilisé par toutes les méthodes de tolérance aux pannes
- 3 types :
  - Matérielle : ensemble des moyens matériels ajoutés au système pour le rendre tolérant aux pannes
  - Logicielle : ensemble des moyens logiciels utilisés pour la tolérance aux pannes
  - Temporelle : allocation de temps supplémentaire pour exécuter des tâches pour la tolérance aux pannes

# Tolérance aux pannes : 4 phases

- Détection d'erreur :
  - une défaillance est déduite par détection d'une erreur dans l'état du système
- Isolement de l'erreur
  - les conséquences de la défaillance sont délimitées
- Recouvrement d'erreur
  - le système est replacé dans un état acceptable
- Traitement de la défaillance
  - le composant défaillant est identifié et le système continu à s'exécuter de manière à ce que celui-ci ne soit plus utilisé.

# Tolérance aux pannes : propriétés recherchées (1)

- Consensus byzantin
  - Accord entre processeurs susceptibles de produire une défaillance *byzantine*
  - Consensus possible si pour n processeurs, le nombre de processeurs défaillant est inférieur à n/3.
  - Il existe différents protocoles pour résoudre ce problème

# Propriétés recherchées (2)

- Synchronisation d'horloges
  - Garder les horloges des différents noeuds d'un système réparti synchronisées
  - Approche déterministe : permet de garantir une précision sur la synchronisation
  - Approche probabiliste : assure la synchronisation avec une probabilité forte, mais pas de garantie

# Propriétés recherchées (3)

- Stockage stable de données
  - Assurer la lecture ou écriture des données même si la mémoire (RAM, disque dur ...) est défaillante
  - Sans redondance matérielle : tolérance aux fautes byzantines
  - Avec redondance matérielle

    ex : disques RAID

# Propriétés recherchées (4)

- Processeur arrêté par défaillance (fail stop processor)
  - Arrêt de l'exécution de processeurs
  - Le contenu de la mémoire volatile (RAM, registre processeurs) est perdu,
  - Etat de la mémoire non volatile n'est pas affectée
  - Le processeur défaillant peut être détecté par les autres processeurs
  - k-fail-stop processor

# Propriétés recherchées (5)

- Détection de défaillance et diagnostic  d'erreur
  - Détection d'une défaillance d'un processeur par les autres processeurs en un temps fini
  - Nécessaire pour le recouvrement d'erreur
  - Méthode simple :
    - détection par timeout
    - Overhead important sur les gros systèmes (tests de tous les processeurs)

# Propriétés recherchées (5b)

- Détection de défaillance (suite)
  - Algorithme de PMC
    - Décomposition du système en unités
    - Chaque unité teste le status d'autres unités
    - Graphe représentant relations de test
      - Arcs labélisés par le résultat du test :
        - Aij =ui tests uj
        - Aij = x si ui est défaillant,
        - Aij = 1 si ui non défaillant et uj défaillant
        - Aij = 0 si ui et uj non défaillant
      - Ensemble des labels =syndrome
      - Syndrome analysé et centralisé par un superviseur, processeur très sûre
      - Conditions sur le nombre de processeur fautifs et sur le graphe de connection pour que le diagnostic puisse être réalisé

# Propriétés recherchées (5c)

- Détection de défaillance (suite)
  - Exemple : OSEK NM
    - Timeout sur messages périodiques
    - Anneau logique : chaque processeur émet un message « I'm alive »

# Propriétés recherchées (6)

- fiabilité de communication :
  - un message envoyé par un processeur à un autre processeur arrive non corrompu au récepteur
  - l'ordre des messages entre deux processeurs est respecté
  - Protocoles pour garantir cette propriété généralement supporté par le protocole de communication
  - Exemple : couche MAC du bus CAN

# Propriétés recherchées (7)

- Atomicité, fiabilité et causalité de diffusion de message
  - diffusion de message possible
  - Diffusion répond à trois propriétés :
    - fiabilité : un message diffusé doit être reçu par tous les processeurs opérationnels
    - Ordre garanti : différents messages envoyés par différents processeurs sont délivrés à tous les noeuds dans le même ordre
    - Causalité : cohérence entre l'ordre des messages délivrés et les événements qui ont déclenchés l'envoi de ces messages
  - 3 types de primitives pour la diffusion
    - Diffusion fiable : supporte seulement la propriété de sûreté
    - Diffusion atomique : supporte sûreté et ordre
    - Diffusion causale : supporte la causalité

# Techniques pour la tolérance aux pannes

- Recouvrement d'un état stable
  - Points de recouvrement périodiques (*checkpointing)*
  - En fonctionnement normal, l'état du système est sauvegardé à chaque checkpoint
  - Lorsqu'une erreur est détectée, le système est remis dans le dernier état stable sauvegardé (*rollback*)
  - Simple en monoprocesseur, mais en réparti :
    - Sauvegarde et restauration réalisés localement

    ➔ Problème de consistance de l'état global

# Techniques pour la tolérance aux pannes (2)

- Recouvrement d'un état stable (suite)
  - 2 approches possibles :
    - Checkpointing asynchrone :
      - Indépendamment sur chaque station, mais enregistrement des communications
      - Simple car pas de synchro
      - Possibilité de ne pas trouver d'état consistant
    - Checkpointing synchrone
      - Coopération pour établir les checkpoint
      - Checkpoint plus complexes, mais recouvrement simplifié
      - Overhead de communications

# Techniques pour la tolérance aux pannes (3)

- Duplication de données
  - Masquage à l'utilisateur des défaillances de communication et/ou de processeurs dans le système
  - Différentes techniques
    - Site primaire
    - Duplication active
  - Trouver le degré optimal de duplication, car les coûts de gestion des replicas peuvent être importants

# Techniques pour la tolérance aux pannes (4)

- Duplication de données
  - Technique du site primaire
    - Chaque donnée dupliquée à un site primaire désigné et des sites de backup
    - Requêtes d'opérations sur la donnée faites au site primaire
    - Le site primaire vérifie régulièrement la consistance des données avec les backups
    - Lorsque le site primaire est défaillant, un algorithme d'élection d'un nouveau site primaire

# Techniques pour la tolérance aux pannes (5)

- Duplication de données (suite)
  - Technique de duplication active
    - Tous les copies sont actives, la requête d'opération sur la donnée est envoyée à toutes les copies
    - N'importe quelle copie peut servir la requête
    - La première réponse est utilisée

# Techniques pour la tolérance aux pannes (6)

- Résistance aux défaillances de processus
  - Un calcul distribué est achevé malgré la défaillance d'une ou plusieurs taches qui le constituent
  - On utilise généralement de la duplication de processus