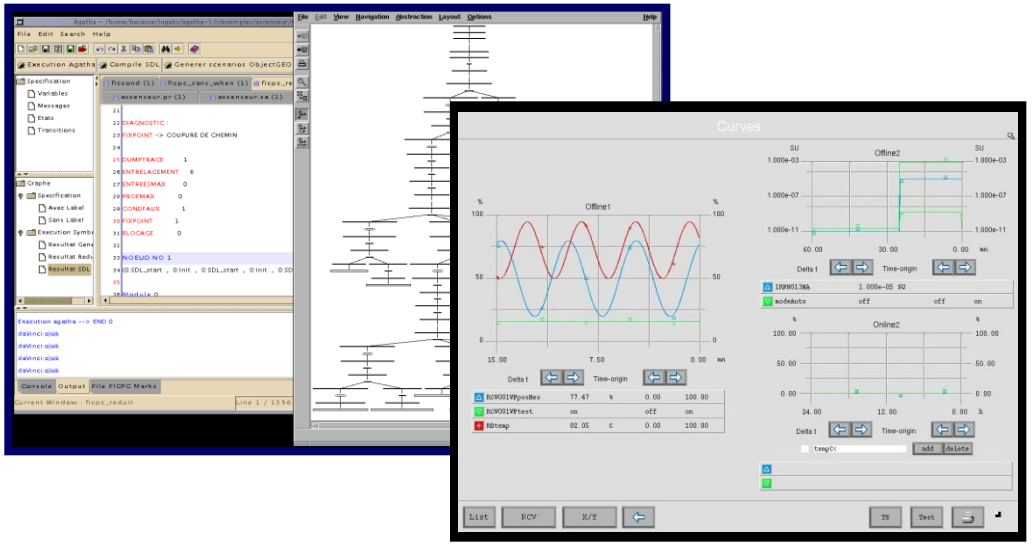
Paradigme et technologie Time-triggered pour la réalisation des systèmes temps-réel multitâche et déterministe

V. David
Ingénieur-Chercheur, IRSN
Expert Systèmes Temps-Réel, Sûreté de Fonctionnement
Précédemment Krono-Safe, Directeur Technique
Fondateur du LaSTRE du CEA LIST
Pr. INSTN



- **Objectifs, contexte, concepts**
  - Le déterminisme et la sûreté de fonctionnement
  - L'approche OASIS
- La mise en œuvre dans une chaîne d'outils
  - Compilation et génération de code
  - Runtime et noyau d'exécution
- Focus sur dimensionnement, testabilité et mise au point
- Perspectives avancées
  - Aspects distribués

Bref historique

- Réacteur 1300 MW : système de CC numérique (1ère génération avec logiciels classés de sûreté 1E)
  - 1985, 8 unités P4, 12 unités P'4 et P''4
  - plus de 350 années de fonctionnement réacteur cumulées (2003)
- Réacteur 1450 MW : système de CC numérique (2ème génération)
  - 1996, 4 unités N4
  - plus de 30 années de fonctionnement réacteur cumulées (2003)

- Etapes principales du projet OASIS
  - 1993 → 1995 : analyse des différentes approches
  - 1996 → 2000 : élaboration des concepts OASIS et outils prototypes
  - 2001 → 2003 : pré-industrialisation
  - 2003 → 2006 : industrialisation dans le cadre du projet QDS/OASIS
- Partenaires
  - 1993 → 2000 : CEA – EDF – FRAMATOME
  - 2001 → 200x : CEA – FRAMATOME (AREVA-NP)
  - 2012 → 200x : Krono-Safe (startup du CEA), Safran, etc.

Objectifs : performances et sûreté

- Diminuer les coûts de développement, de qualification et d'architecture
  - plus faciles à concevoir, à réaliser, à maintenir (productivité)
  - plus faciles à tester, à vérifier et à valider (productivité et qualification)
  - plus performants, trouver le bon dimensionnement (meilleure intégration)

- Meilleures intégration, modularité et flexibilité
  - permettre un fonctionnement « multitâches »
    - avec des échelles de temps multiples
  - « composants » temps-réel, interface et comportement parfaitement définis et maîtrisés
    - interactions déterministes des « composants »
    - (pré-)certification générique de systèmes à logiciel prépondérant
  - réduire le nombre de composants matériels (ECU, réseaux…)

- Meilleure sûreté de fonctionnement des ECU
  - garantir un fonctionnement déterministe, indépendamment des problèmes d'ordonnancement
  - supporter des mécanismes de détection et de confinement d'anomalies de fonctionnement (ségrégation)

=> Approche multitâche orientée temps-réel et sûreté de fonctionnement

Sûreté de fonctionnement

- Confiance fondée sur quelques grands principes
  1. robustesse, tolérance aux fautes
     (détection et confinement d'anomalies, modes dégradés, etc.)
  2. maîtrise des mécanismes mis en œuvre
     (ex : problèmes des logiciels à interruptions)
  3. déterminisme comportemental
     (reproductibilité des tests : les mêmes causes impliquent les mêmes effets)
  4. application du principe de diversification
     (prévenir les pannes de cause commune, ex : redondance ou div. fonct., etc.)
  5. défense en profondeur

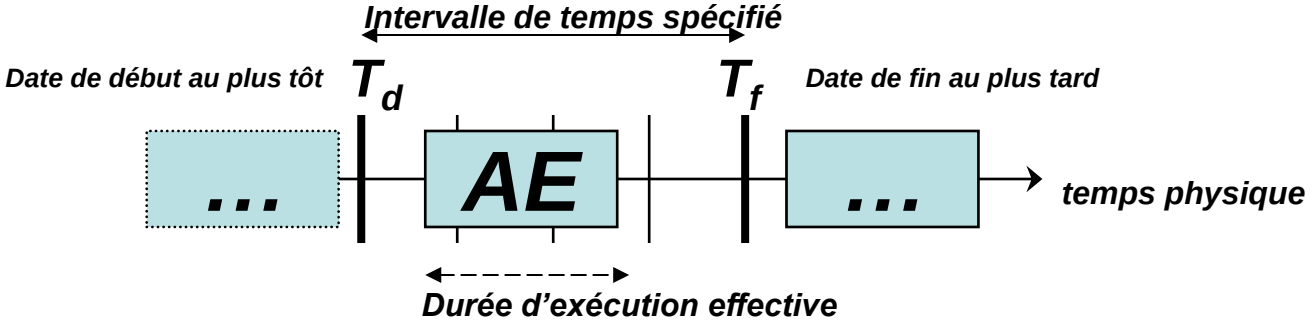
- OASIS s'attaque aux points 1, 2, 3 et facilite la mise en œuvre du point 4 (meilleure intégration)

S'inspirer du multitâche, mais viser le déterminisme système

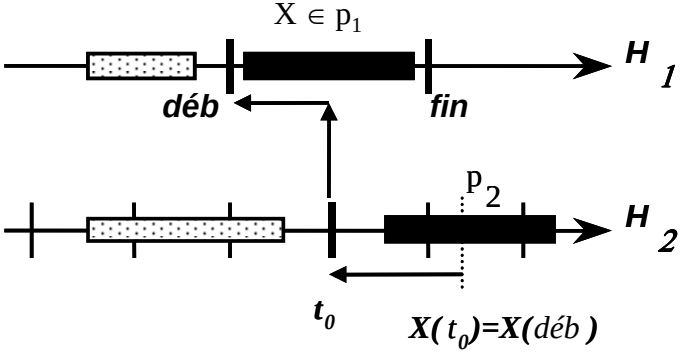
- Ne pas introduire de sources d'asynchronismes dans la conception des applications (e.g. pas d'entrées/sorties sous interruptions)
  - dominer les asynchronismes dus aux temps d'exécution variables
    - pour garantir le déterminisme
    - pour maîtriser le dimensionnement (performances)
- Proposer un modèle multitâche communiquant
  - avec un partage sécurisé des ressources
  - sur des cibles matérielles standards
- Simplifier les problèmes de coordination des activités exprimées en conception
- Définir une approche modulaire/compositionnelle
  - interactions entre modules totalement spécifiées et maîtrisées
  - réduire les difficultés de mise au point, de tests et de validation (test et déterminisme)

Le fonctionnement TT dans OASIS

- Le système est cadencé par le temps réel (Time-Triggered)
  - tous les traitements prennent place entre deux instants du système (mais pas forcément consécutifs) :
    - les exécutions asynchrones sont synchronisées (synchronisme) au début et à la fin des traitements : isochronisme réel indépendant du matériel
  - tous les transferts de données entre « tâches » ont lieu à la transition entre deux traitements

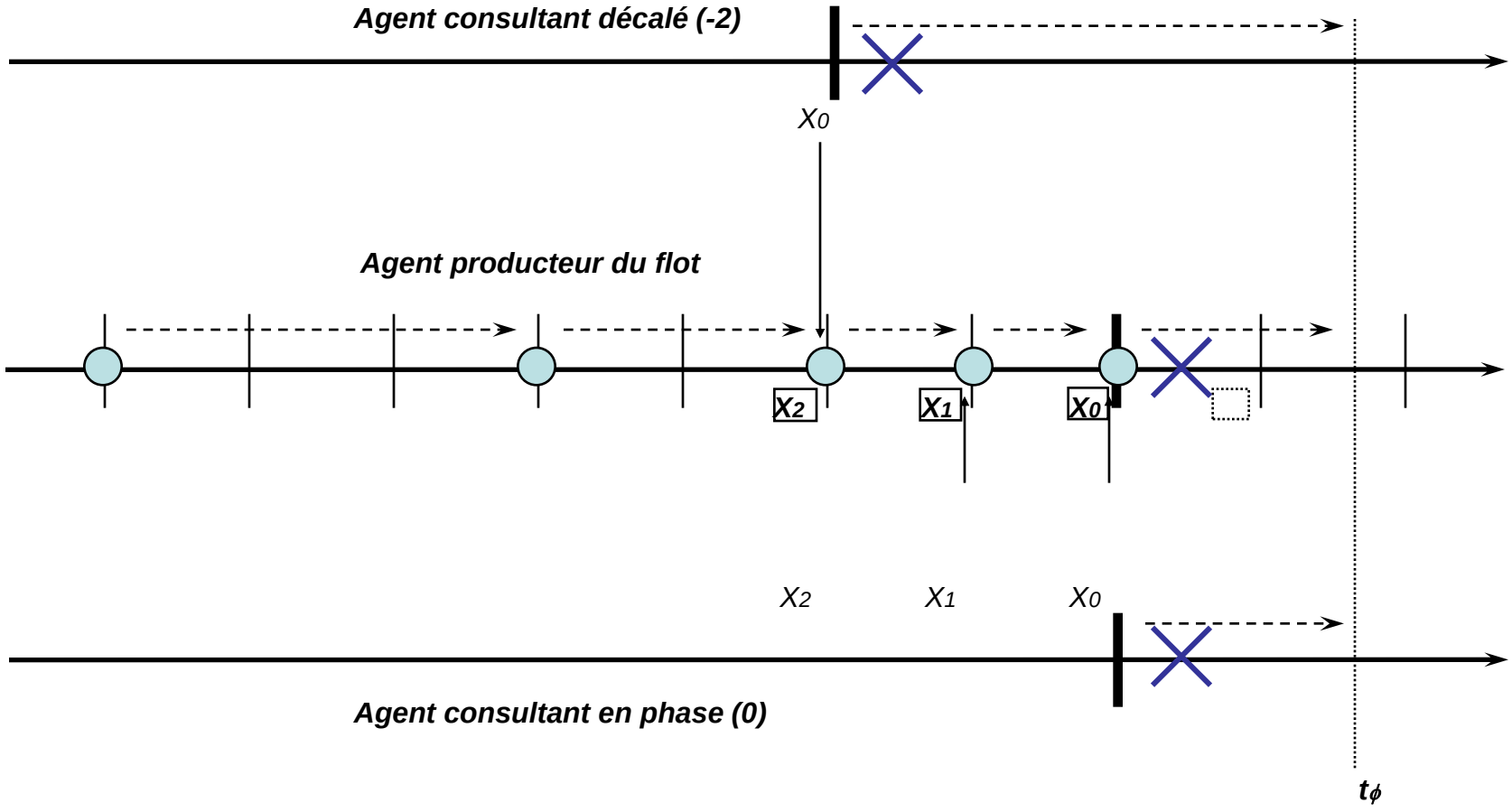


- Cohérence temporelle des échanges (liée aux mécanismes de communication OASIS)
  - encapsulation des données et des traitements (donnée protégée et un seul producteur)
  - les valeurs des données sur lesquelles travaille une AE ne sont pas sensibles entre T_d et T_f



  - principe d'observabilité strict
  - mise à jour implicite et automatique aux rythmes définis
  - dimensionnement automatique et protection des tampons, optimaux et sûrs

Cohérence temporelle

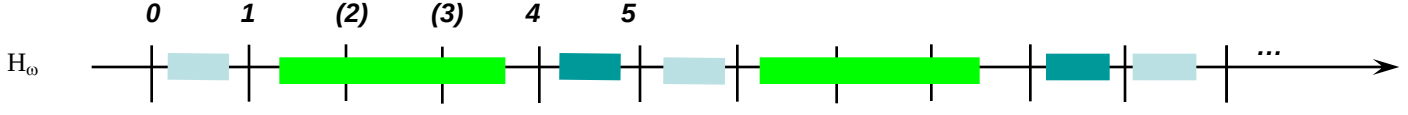


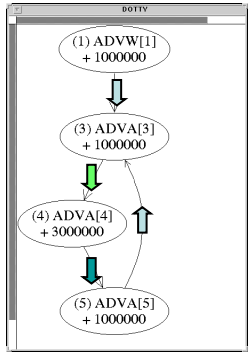
Ordonnancement et dimensionnement

- Les valeurs des données sont indépendantes
  - des durées d'exécution réelles, de la politique d'ordonnancement, de la fragmentation des AE

=> Le comportement temps-réel est indépendant de l'ordonnancement

- Une condition nécessaire et suffisante de dimensionnement
  - un surgraphe représente tous les comportements temporels possibles d'une tâche du point de vue des synchronisations temporelles





  - la composition de ces surgraphes permet de connaître tous les intervalles temporels partagés par les traitements de chaque tâche
    - le produit synchronisé est l'opération mathématique de composition des surgraphes
    - le « graphe » obtenu contient TOUS les entrelacements possibles des traitements

  - engendre un système d'équations et d'inéquations temporelles

=> Démonstration du dimensionnement

[ICECCS 98, ISOLA 07]

- Objectifs, contexte, concepts
  - Le déterminisme et la sûreté de fonctionnement
  - L'approche OASIS
- **La mise en œuvre dans une chaîne d'outils**
  - Compilation et génération de code
  - Runtime et noyau d'exécution
- Focus sur dimensionnement, testabilité et mise au point
- Perspectives avancées
  - Aspects distribués

Mise en œuvre d'OASIS et choix primordiaux

- Pas de rupture culturelle de l'entreprise, mais une rupture technologique pour la sûreté
  - utilisation d'environnement et de plateforme standard : C, gcc, gld, etc.
  - portabilité et pérennité des développements
  - qualité des outils (taille des problèmes traités, performances, traçabilité, etc.)

- Description aisée d'une conception OASIS
  - Le temps, les agents, les communications, etc.
  - Fusionner les aspects OASIS et les aspects algorithmes classiques

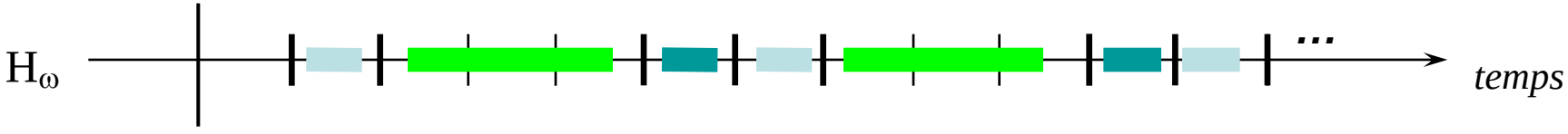
- Limiter les erreurs
  - Approche déclarative (spécification du besoin) plutôt qu'impérative (appels de fonctions ou de primitives)
  - Contrôle de la conception (cohérence du modèle) et moyens d'analyses d'une conception (à différents niveaux)
  - Dégager le concepteur des calculs (sur le temps, le dimensionnement, etc.), des problèmes de codage

↳ **Définition du langage ΨC et compilation de la conception OASIS décrite**

- Extension complète du langage C :
  - Multitâche : définition des tâches (*agent*), de leur code algorithmique dont leur point d'entrée (*body start*)
  - Communications inter-agents : interfaces et manipulations des flots de données temps-réel et des messages temporels
  - Comportement temps-réel : au sein des algorithmes de calcul, expression du temps et des synchronisations temporelles (*advance*, *before*, …)
  - Le code ΨC et l'algorithmique des agents se base sur les instructions du langage C (fonctions C, intr. préprocesseur, etc.)
  - Englobe la norme C ISO/CEI 9899:1989 (ANSI-C) et sa bibliothèque standard

Exemple d'écriture du ΨC

- Un agent avance dans le temps par sauts successifs le long d'une échelle de temps (horloge de base de la tâche) définie par le concepteur pour chaque agent
  - Les sauts peuvent être de longueurs différentes (comme ci-dessous 1-3-1)

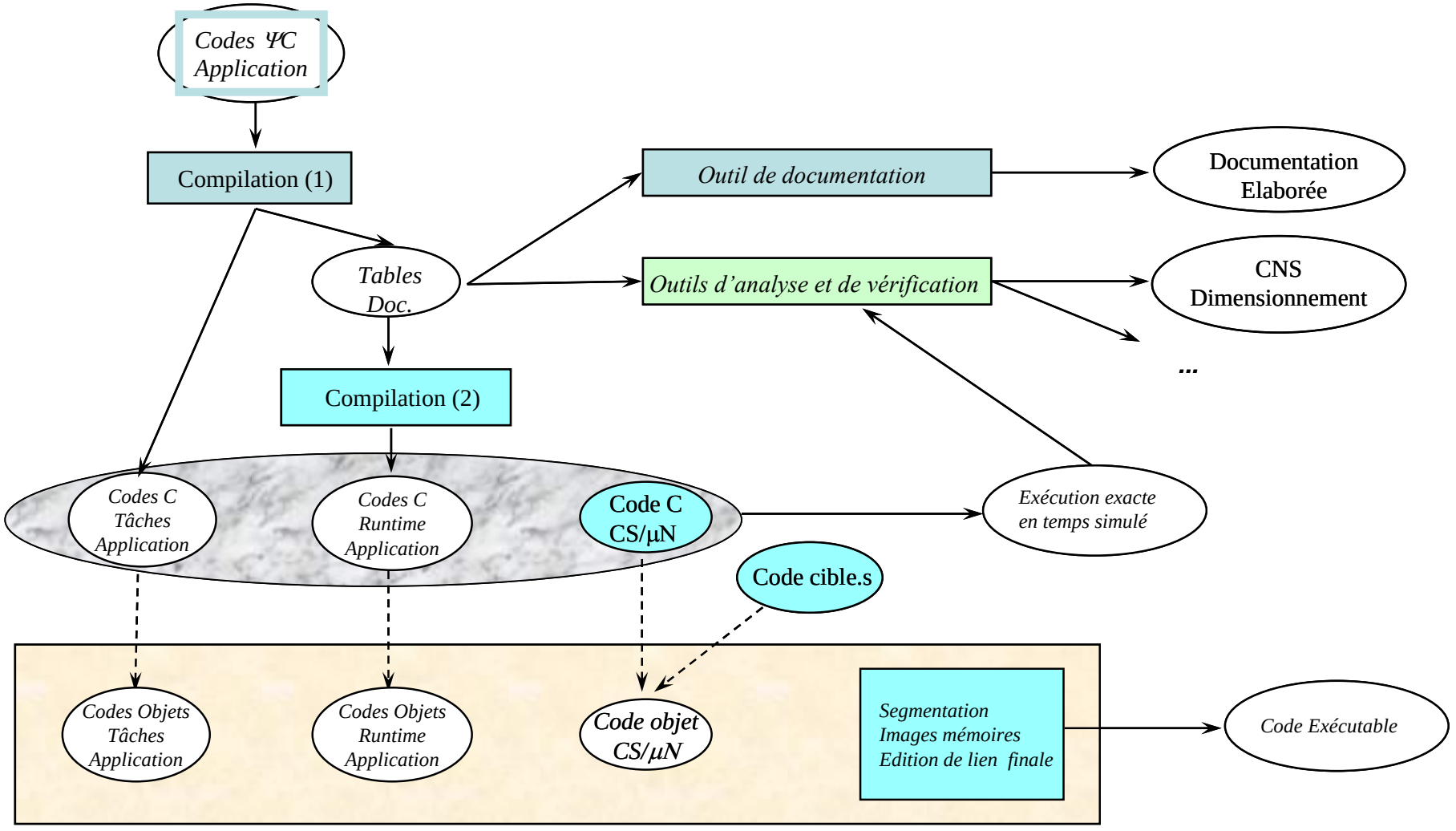



```
clock HBASE= gtc1(0, 1); /* 1 ms */
application demo(inittime=99) with HBASE;
agent ag1tst(starttime= 1) with HBASE
{
  body start
  {
    /* proc 1 */ advance(1);
    /* proc 2 */ advance(3);
    /* proc 3 */ advance(1);
  }
}
```

Initialisation TT :

- Chaque tâche démarre à partir d'un instant spécifié de son horloge de base après le démarrage TT de l'application
- La valeur initiale du temps TT de l'application est elle aussi prédéfinie par le concepteur

Chaîne de génération de code OASIS simplifiée

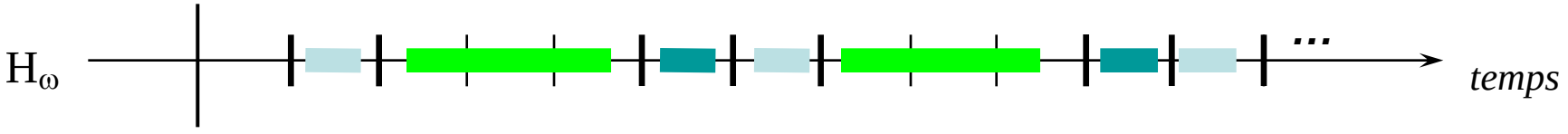


Compilation de graphes

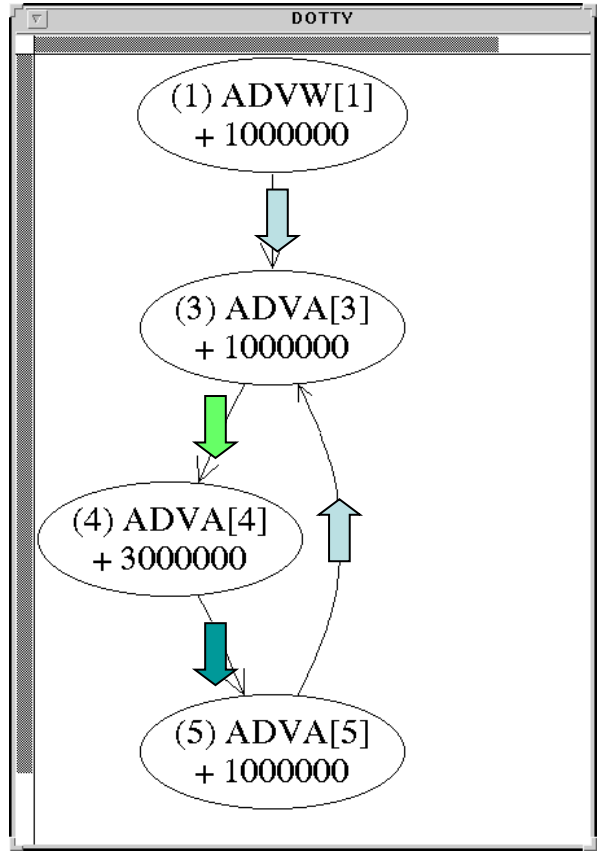
- Point de départ : code ΨC de chaque agent
- Le compilateur a construit pour chaque fichier .psy, son arbre de syntaxe abstraite
  - il contient toutes les règles de grammaire utilisées
    - dont les instructions C conditionnelles et toutes les instructions Ψ (advance, …)
  - et leur agencement dans le code (tient compte des boucles, branchements conditionnels)
- Des graphes intermédiaires sont engendrés (Psy1 et Psy1g fic.psy) pour pouvoir
  - se limiter aux seuls nœuds/arcs intéressants du point de vue du flot d'exécution et des contraintes temporelles
  - calculer les contraintes temporelles de chaque portion de code (date de début au plus tôt, date de fin au plus tard ou échéances)
- Après réduction, on obtient un graphe minimal par agent contenant
  - Les contraintes temporelles de chaque traitement de l'agent
  - Tous leurs enchaînements possibles dans le code l'agent

  C'est le graphe de contrôle d'exécution de l'agent

Notion de surgraphe d'un agent

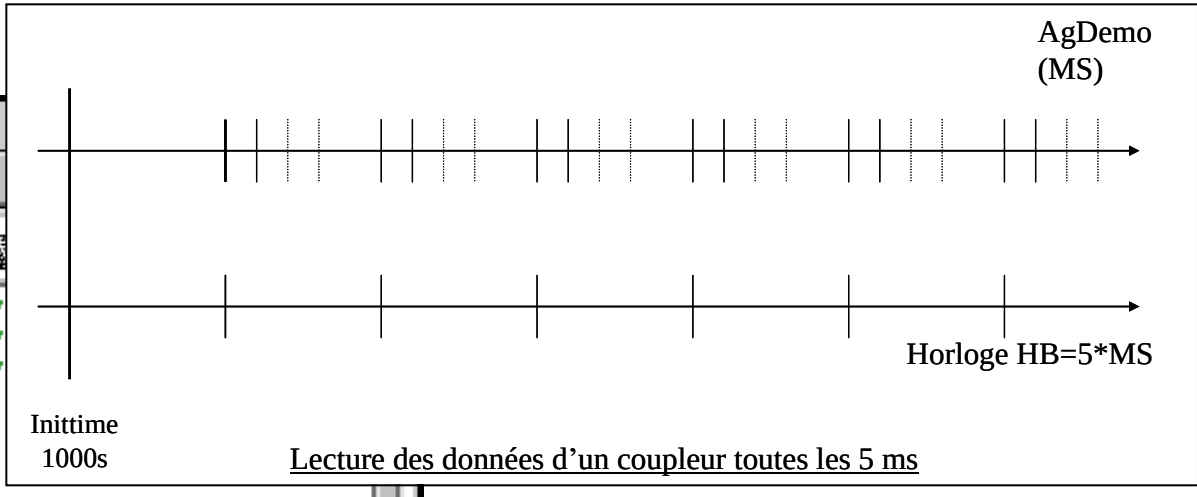



```
clock HBASE= gtc1(0, 1); /* 1 ms */
application demo(inittime=99) with HBASE;
agent ag1tst(starttime= 1) with HBASE
{
  body start
  {
    /* proc 1 */ advance(1);
    /* proc 2 */ advance(3);
    /* proc 3 */ advance(1);
  }
}
```



compilation

Calcul du surgraphe d'un agent (1/2)



Lecture des données d'un coupleur toutes les 5 ms

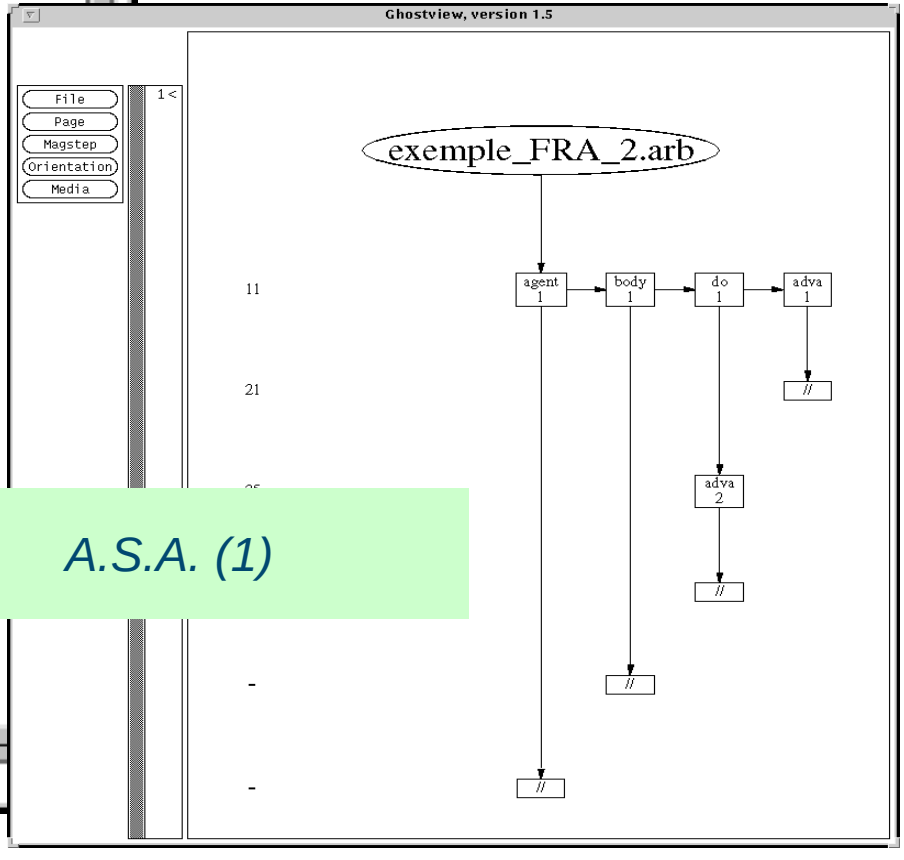
emacs: exemple_FRA_2.psy

File Edit Apps Options Buffers Tools C

```
clock MS = gtcl( 0, 1 ); /* horloge de période 1 millisecondes */
clock HA =  1*MS;        /* horloge de période 1 millisecondes */
clock HB =  5*HA;        /* horloge de période 5 millisecondes */

/* déclaration de la date de début de l'application */
application demo (inittime=1000*1000) with MS;

#include "simu_coupleur.h"

/* code source de l'agent AgDemo */
agent AgDemo (starttime=1 with HB) with HA
{
  body start
    {
      long indParc=0;
      long maxParc;

      maxParc=lec_donn();

      do [3]  {
        advance(1);
        indParc++;
      } while (indParc < maxParc);

      advance(1) with HB;
    }
}
```

-----XEmacs: exemple_FRA_2.psy (C PenDel Font)----L1--C1--All----------

Fontifying exemple_FRA_2.psy... done.

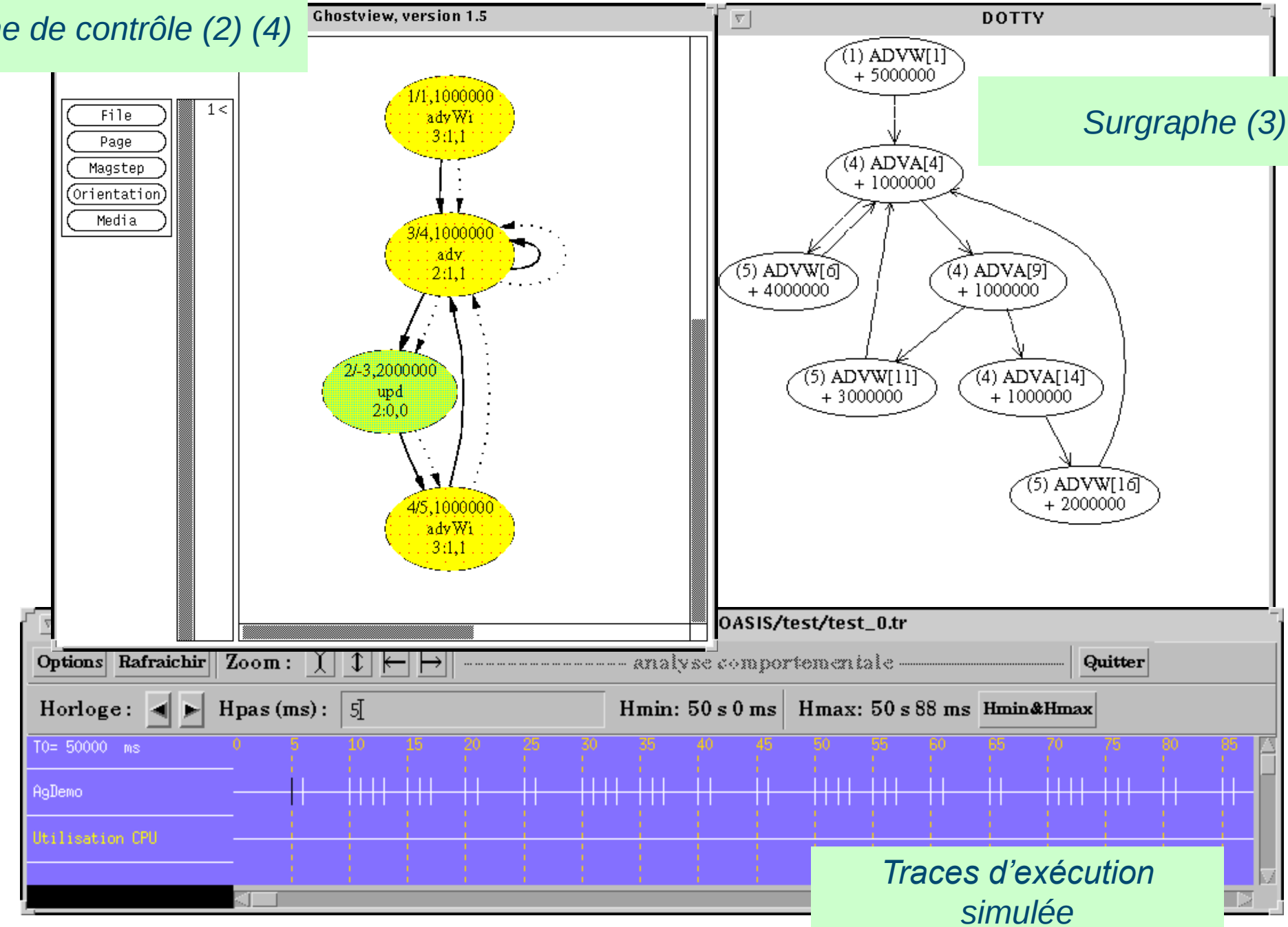


A.S.A. (1)

Calcul du surgraphe d'un agent (2/2)

Graphe de contrôle (2) (4)

Surgraphe (3)



Traces d'exécution simulée

La compilation ΨC et la génération de code C

- Chaîne de compilation :
  - gère différentes unités de compilation, les numéros de lignes pour les codes d'erreurs (cf. traçabilité), etc.
  - calcule les contraintes temporelles
  - calcule les dimensions des tampons
- Code exécutable engendré directement à partir de la conception en ΨC
  - génération des données et codes de runtime pour chaque agent
  - génération des données de runtime de l'application pour le Noyau
- En ΨC, il n'existe que des primitives de programmation déclaratives :
  - pas d'appel noyau direct, tout est compilé à partir de la description haut niveau
- Après compilation, une seule primitive d'exécution (!) :
  - demander le changement de nœud dans le graphe de contrôle
  - la sémantique est compilée dans la runtime avec ses tables
  - le graphe de contrôle de l'automate de chaque agent est fourni à la runtime
  - le contrôle de cohérence est assuré par le Noyau

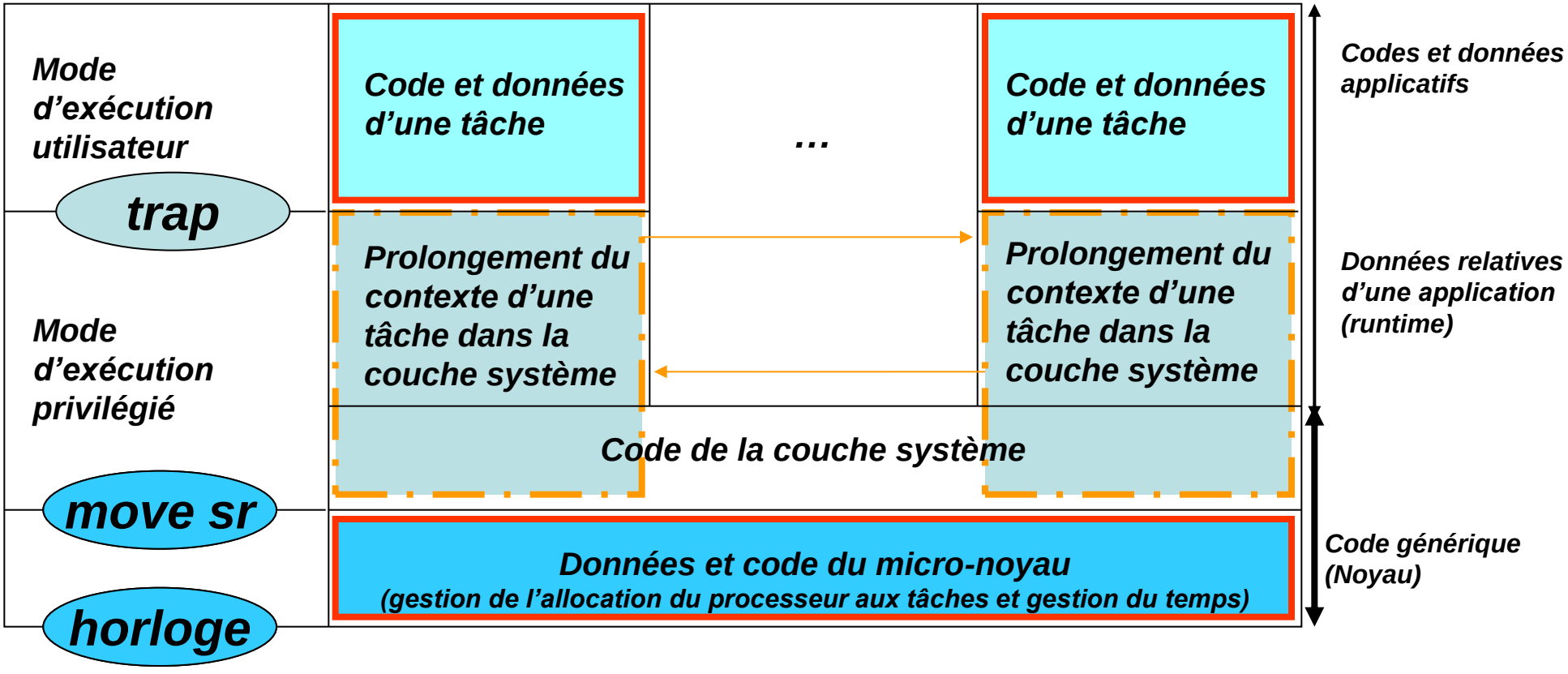
Anomalies, déterminisme et modularité

- Quelques défauts résiduels peuvent toujours être présents
  - le nombre de défauts non révélés doit être aussi bas que possible
  - certains défauts peuvent être activés sans être détectés par le test
    - Exemple de non déterminisme : lecture d'une mauvaise adresse contenant une valeur dépendante de l'instant physique de l'accès (dans ce cas, le comportement observé sera dépendant de l'implantation, de la vitesse du processeur, de l'ordonnancement)

- Principe de déterminisme fort dans OASIS
  - la conséquence d'une anomalie doit être déterministe
    - il faut confiner les anomalies et maîtriser leurs conséquences
    - forte intégration => nouveaux mécanismes de détection et de confinement déterministes
  - il faut
    - détecter à l'exécution toutes les tentatives de violation de la séparation des tâches
    - contrôler le dimensionnement des ressources d'exécution (tampons, quotas CPU, etc.)
    - contrôler l'enchaînement séquentielle des actions, etc.
  - l'impact d'une anomalie de fonctionnement est maîtrisé et déterministe

- Définit une architecture logicielle modulaire
  - généricité des tâches au niveau du code objet (binaire)
    - pas de recompilation
    - seulement le runtime à engendrer et l'édition de liens à refaire
  - permet une approche multi-fournisseur

Organisation défensive des programmes exécutables



- contextes d'exécution distincts hors de CS et différents internes à CS, contexte μN indépendant
- segmentation particulière des exécutables et des données
  - conception orienté sûreté et exploitation spécifique des mécanismes de protection liés au H/W
- contrôle de conformité de l'exécution de chaque tâche
- contrôle toutes les hypothèses de dimensionnement (quotas, tampons, etc.)
- les préemptions de tâches sont toutes contrôlées par le μNoyau atomique
- CS 100% préemptible

Noyau générique d'exécution (1/2)

- Le Noyau n'est pas un moniteur standard
  - c'est un exécuteur orienté sûreté des graphes de contrôle d'exécution de chaque tâche
- Des mécanismes de détection et confinement d'anomalies de fonctionnement
  - les tentatives de violation d'un quota de temps imparti ou la violation d'un graphe de contrôle ou la violation du dimensionnement d'un tampon
  - conception orienté sûreté et exploitation spécifique des mécanismes de protection liés au H/W (MMU –descripteurs statiques- et mode privilégié) (brevet d'invention)
  - l'impact d'une anomalie de fonctionnement est maîtrisé et déterministe
- Le μNoyau est un automate cadencé par le temps (programme séquentiel non préempté) :
  - la seule source d'interruption (l'horloge système) est utilisée pour la détection et le confinement d'anomalies et les synchronisations temporelles
  - les préemptions de tâches sont toutes contrôlées par le μNoyau
  - la CS est entiérement lock-free/wait-free (100% préemtible)
    - activité incluse dans la gestion des quotas

Noyau d'exécution sur IA32 (2/2)

- Portabilité
  - Couche système écrite en C
  - Micronoyau écrit en C pour *toutes* les parties indépendantes du matériel (assembleur sinon)
- Mise en œuvre des E/S matérielles spécifiques sur Intel IA32/PC-AT (IO_S)
  - Dont les fonctions spécifiques avec droits superviseur restreints pour effectuer les opérations d'E/S (in/out)
  - Pour que chaque agent puisse faire appel aux fonctions d'E/S qui nécessitent ces privilèges avec contrôle d'accès
- Contrôle de l'initialisation et de la configuration des chipsets au démarrage (définition du mode INIT_S)
  - Une fois que le BIOS a donné la main au bootloader OASIS, il n'est plus jamais utilisé
  - Bootloader OASIS
    - Permet de télécharger les images noyau séparément des images applicatives sur mémoire statique de la carte via les ports réseaux
      - Gestion d'un protocole de communication pour le téléchargement par paquets
      - Contrôle des réceptions via un protocole spécifique et cryptage des binaires
    - Contrôles sur les chipsets
      - Non utilisés : mis en mode désactivé
      - Utilisés : configuration puis avant la fin de la séquence de démarrage vérification de la conformité de la configuration mise en œuvre
- Performance
  - Temps de passage dans le micronoyau de moins d'une microseconde

Editions de liens, segmentation mémoire (1/2)

- Psyld réalise l'édition de liens de tous les _.o d'une application pour la plateforme cible
  - Première édition de liens entre chaque agent avec ses librairies et sa partie de runtime correspondant
    - Segmentation mémoire
      - Implantation du pont (interface entre le noyau générique et une application)
      - Implantation de chaque agent
      - Alignement de tous les segments pour pagination
    - genMMU fait les tables MMU correspondantes
      - Compilation des tables
      - Implantation
  - Seconde édition de liens entre tous les morceaux
    - Produit le résultat _.ex
  - NB : le Noyau n'est pas recompilé, ni relinké
- Ldint réalise l'édition de liens de tous les _.o d'une application pour la simulation exacte sur POSIX
  - Idem avec programme principal (main en bibliothèque : inter.o) et sans implantation mémoire

Editions de liens, segmentation mémoire (2/2)

- Organisation de la mémoire telle que
  - Les piles des agents sont séparées
    - Placées dans des segments tels que tout débordement est détecté par la protection mémoire
  - Pas de translation d'adresse : mapping direct (bijection) entre adresses logiques et physiques
    - Facilite la détection d'erreur
    - Permet une mise au point aisée
  - Pas de dépendances cachées que ce soit
    - Via les librairies applicatives
    - Via le code couche système
  - Toutes les tables de symboles sont conservées
    - Globales (tables systèmes, tables de protection mémoire, etc.)
    - Locales aux agents (données/fonctions locales, en bibliothèque, etc.)

Synthèse

- Conciliation du déterminisme et du multitâche
  - objectifs de sûreté et de performances des systèmes critiques embarqués
  - maîtrise des applications temps-réel développées
  - maîtrise du dimensionnement

- Avantages
  - permet autant de rythmes de temps de cycles que nécessaire
  - permet de tenir plus facilement la charge sur les temps de cycle
  - permet des démonstrations de sûreté dès l'étape de conception
  - difficultés de conception (diagrammes temporels) séparées des difficultés de dimensionnement (ordonnancement)
  - portabilité et pérennité grandement facilitées
    - coûts de modification et de maintenance mieux maîtrisés
    - véritable approche modulaire pour la composition de tâches temps-réel
  - exécution réelle en temps-réel ou simulé sans machine cible
  - nombreux mécanismes de détection et de confinement d'anomalies de fonctionnement
    - ségrégation, comportements extrêmes mieux maîtrisés

- Transfert industriel dans le domaine nucléaire pour Framatome / AREVA NP (2003-2007)
  - respect des normes et recommandations en vigueur
    - standard industriels : C, IA32, etc.
    - exigences de sûreté : CEI 60880, RFS
  - plateforme innovante QDS/OASIS
  - qualification : système classé de sûreté, catégorie A
- Startup Krono-Safe depuis 2012

- Objectifs, contexte, concepts
  - Le déterminisme et la sûreté de fonctionnement
  - L'approche OASIS
- La mise en œuvre dans une chaîne d'outils
  - Compilation et génération de code
  - Runtime et noyau d'exécution
- **Focus sur dimensionnement, testabilité et mise au point**
- Perspectives avancées
  - Aspects distribués

Dimensionnement / tampons de communication

- Principe
  - Le concepteur exprime les besoins en communication, les outils de génération de code calcule la taille des tampons pour la mise en œuvre
  - Calculer automatiquement un majorant le plus proche possible du maximum atteignable par analyse des informations statiques recueillies lors de la compilation

- Pour les variables temporelles
  - Calcul réalisé grâce aux tables extraites de la compilation
  - La taille d'un tampon local par consultant a une taille égale à celle spécifiée dans son interface
  - La taille du tampon global est calculé en fonction des sauts maximums dans le temps du producteur et des consultants vis-à-vis de la période de l'horloge associée à la variable temporelle (indépendamment de l'ordonnancement)

Dimensionnement CPU / principes

- Cadencement TT d'une tâche décrit dans la conception
  - Encapsulation du comportement temporel dynamique dans le surgraphe issu de la compilation
    - Enchaînement dynamique des AE d'une tâche
    - Echanges de données inter-tâches

- Ordonnancement dynamique de l'exécution
  - Qui respecte le cadencement TT issu de la conception
    - Exécution du graphe de chaque tâche séparément contrôlé par le noyau
  - Ordonnancement guidé par les échéances
    - Optimal pour le modèle OASIS
    - Obtention d'une CNS pour savoir si la propriété de ponctualité est vérifiée
  - Démonstration hors-ligne du dimensionnement
    - Basée sur la connaissance des quotas des AE (majorants des temps d'exécution)
    - Surveillance en ligne
      - Des quotas (dimensionnement AE)
        - » Instrumentation du noyau possible pour évaluation des temps d'exécution
      - Et des échéances (dimensionnement application)

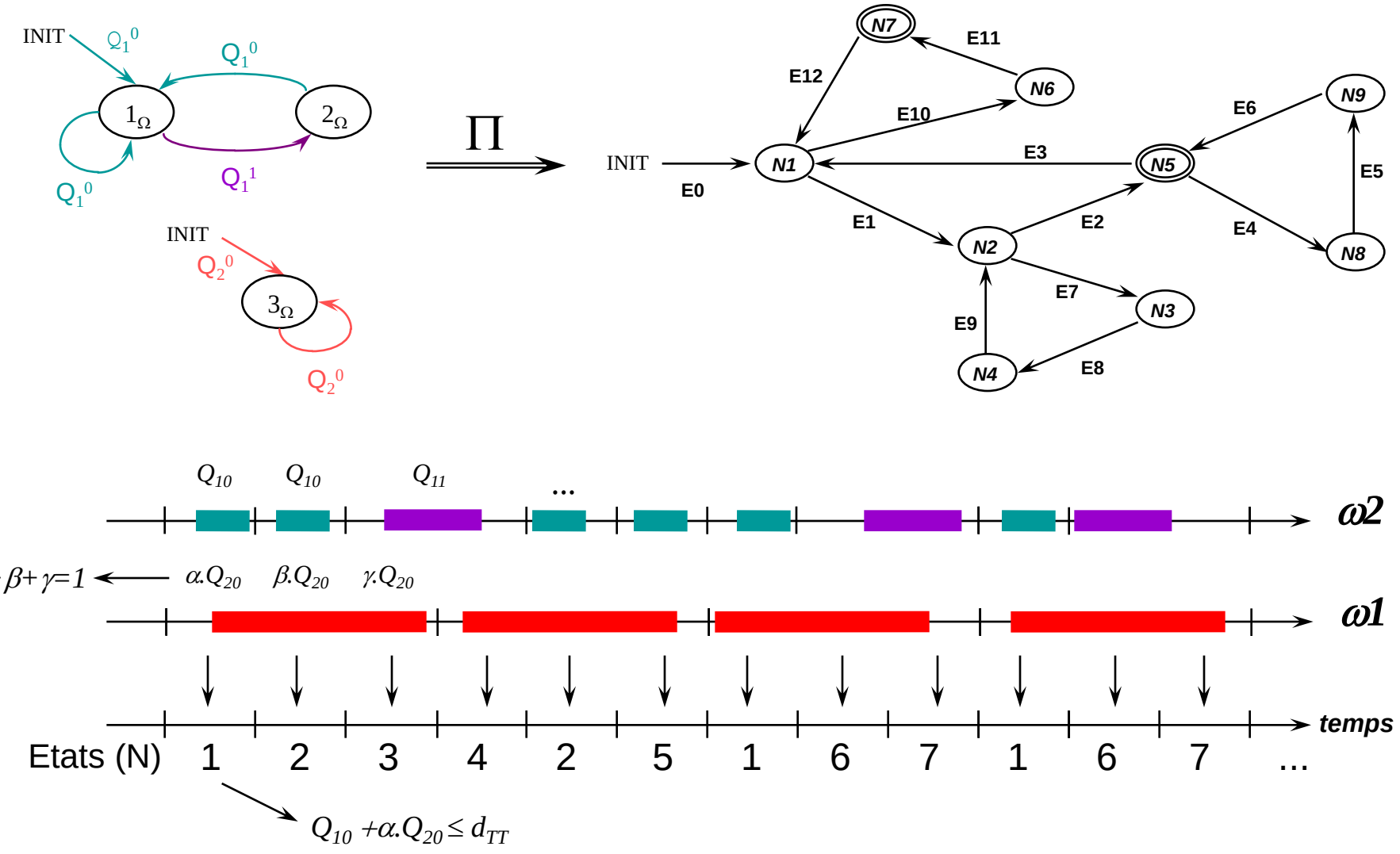
Dimensionnement CPU / mise en oeuvre

- Construction mathématique de la CNS
  - Composition des tâches : dépliage « temporel » de chaque surgraphe
  - La composition des surgraphes des tâches permet de connaître tous les intervalles temporels partagés par les traitements de chaque tâche
    - Le produit synchronisé est l'opération mathématique de composition des surgraphes
    - Le « graphe » obtenu contient TOUS les entrelacements possibles des traitements

- A partir de ce graphe, on peut engendrer un système d'équations et d'inéquations temporelles
  - les équations traduisent le fait qu'un traitement peut être réparti sur plusieurs intervalles TT (quelque soit la politique d'ordonnancement)
  - les inéquations dites « de charge » traduisent que sur un intervalle TT, le temps d'exécution pris par les traitements (ou portions de) est inférieur à la durée de l'intervalle

  ↳La condition formelle à vérifier est le système d'équations et inéquations linéaires

Dimensionnement CPU / illustration (1/3)



Dimensionnement CPU / illustration (2/3)

- La propriété de ponctualité à vérifier représente le fait que :
  - Sur chaque fenêtre temporelle, la somme des temps d'exécution des parties d'AE de chaque tâche doit être inférieure à la durée de la fenêtre temporelle
  - Mieux que la condition suffisante classique : la somme des facteurs de charge normalisés par tâche
    - Lorsque les maxima de chaque AE ne sont pas atteints simultanément

- Cela engendre les contraintes suivantes
  - Pour chaque arc du produit synchronisé
    - Une inéquation sur les portions d'AE exécutés dans l'intervalle (identifié par j)

    $\Sigma_{[i=1..n]} \alpha_{ij} . Q(AE_i) \leq k$ with α_{ij} in [0,1]
  - Pour les nœuds de terminaison des AE du produit synchronisé
    - Une équation sur les variables libres α par AE (identifié par i)

    $\Sigma_{[j=1..p]} \alpha_{ij} = 1$

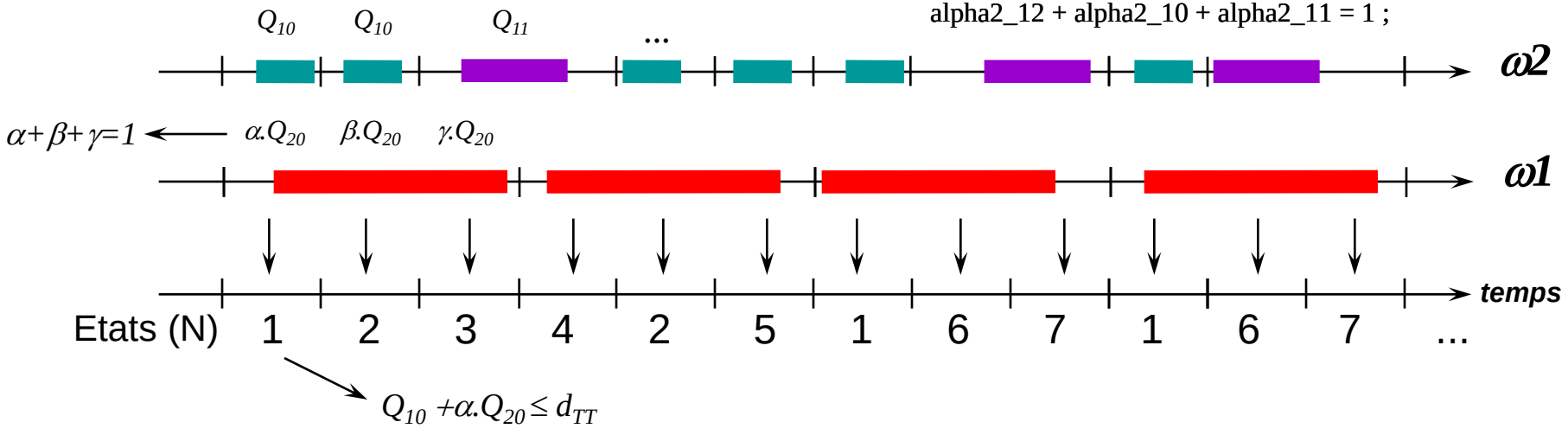
Dimensionnement CPU / illustration (3/3)

INÉQUATIONS TEMPORELLES

E0 : Q1_0 alpha1_0 + Q2_0 alpha2_0 <= k ;
E1 : Q1_0 alpha1_1 + Q2_0 alpha2_1 <= k ;
E2 : Q1_0 alpha1_2 + Q2_0 alpha2_2 <= k ;
E3 : Q1_0 alpha1_3 + Q2_0 alpha2_3 <= k ;
E4 : Q1_1 alpha1_4 + Q2_0 alpha2_4 <= k ;
E5 : Q1_1 alpha1_5 + Q2_0 alpha2_5 <= k ;
E6 : Q1_0 alpha1_6 + Q2_0 alpha2_6 <= k ;
E7 : Q1_1 alpha1_7 + Q2_0 alpha2_7 <= k ;
E8 : Q1_1 alpha1_8 + Q2_0 alpha2_8 <= k ;
E9 : Q1_0 alpha1_9 + Q2_0 alpha2_9 <= k ;
E10:Q1_1 alpha1_10 + Q2_0 alpha2_10 <= k ;
E11:Q1_1 alpha1_11 + Q2_0 alpha2_11 <= k ;
E12:Q1_0 alpha1_12 + Q2_0 alpha2_12 <= k ;

ÉQUATIONS

alpha1_0 = 1 ;
alpha2_0 + alpha2_1 + alpha2_2 = 1 ;
alpha2_0 + alpha2_1 + alpha2_7 = 1 ;
alpha2_0 + alpha2_10 + alpha2_11 = 1 ;
alpha1_1 = 1 ;
alpha1_2 = 1 ;
alpha1_3 = 1 ;
alpha2_3 + alpha2_1 + alpha2_2 = 1 ;
alpha2_3 + alpha2_1 + alpha2_7 = 1 ;
alpha2_3 + alpha2_10 + alpha2_11 = 1 ;
alpha1_4 + alpha1_5 = 1 ;
alpha2_4 + alpha2_5 + alpha2_6 = 1 ;
alpha1_6 = 1 ;
alpha1_7 + alpha1_8 = 1 ;
alpha2_8 + alpha2_9 + alpha2_2 = 1 ;
alpha2_8 + alpha2_9 + alpha2_7 = 1 ;
alpha1_9 = 1 ;
alpha1_10 + alpha1_11 = 1 ;
alpha1_12 = 1 ;
alpha2_12 + alpha2_1 + alpha2_2 = 1 ;
alpha2_12 + alpha2_1 + alpha2_7 = 1 ;
alpha2_12 + alpha2_10 + alpha2_11 = 1 ;

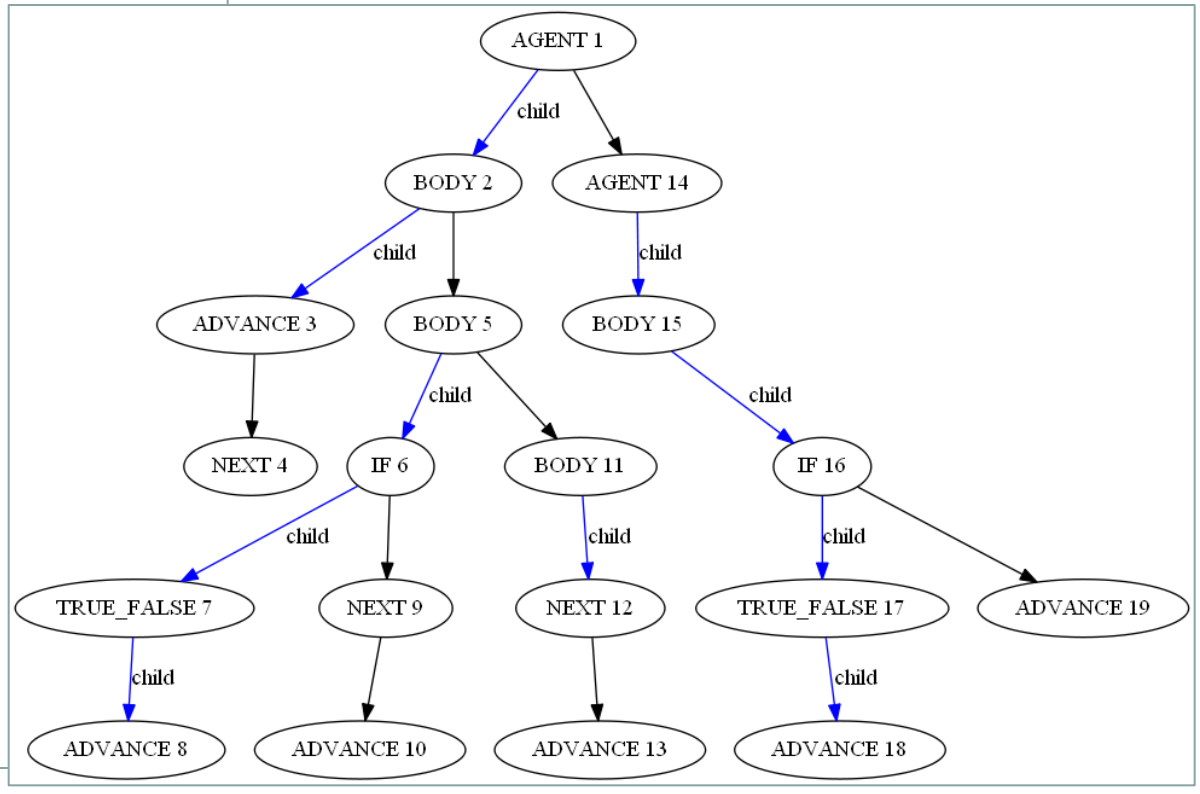


Dimensionnement CPU / résolution

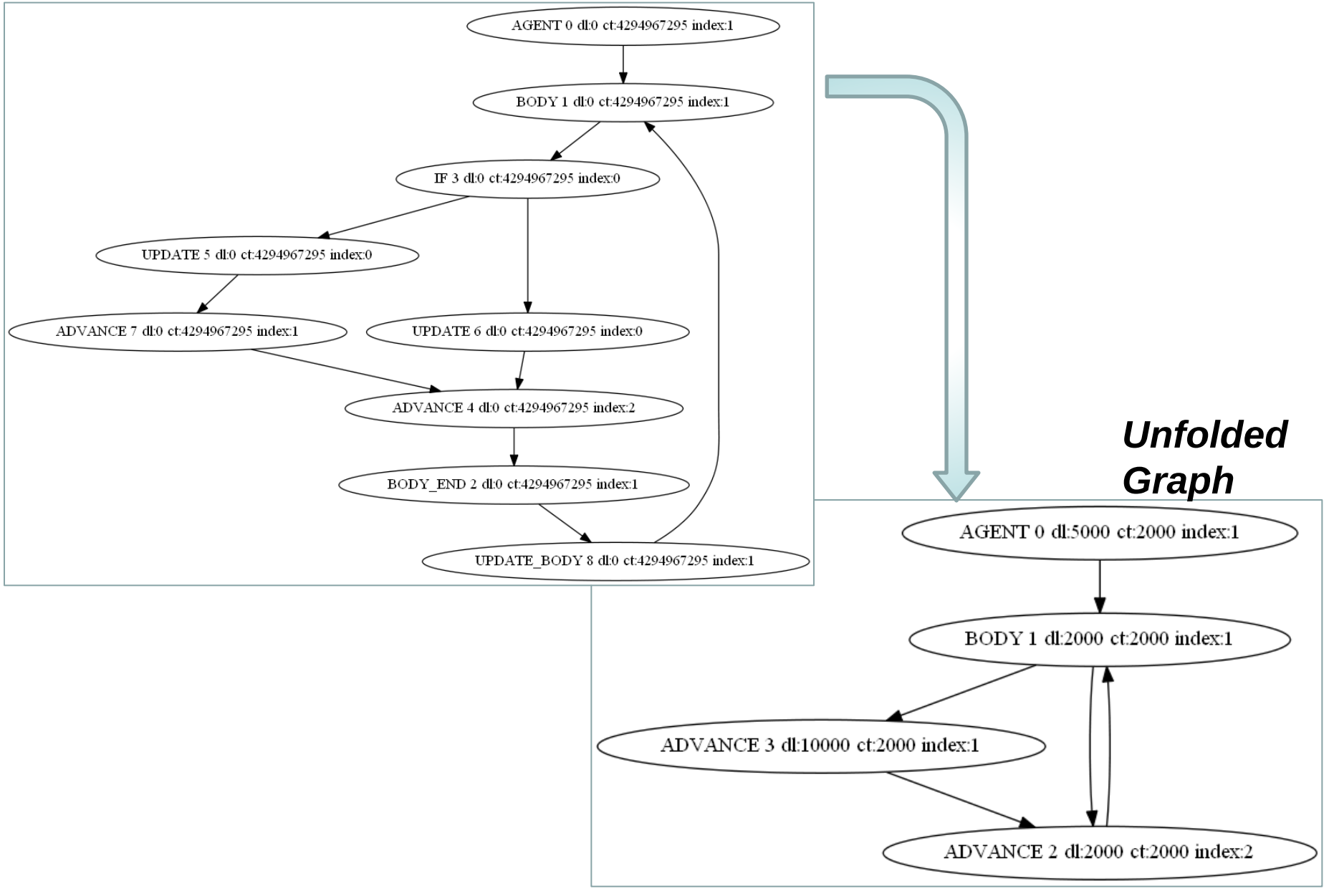
- Résolution d'un problème classique de résolution de contraintes
  - L'existence d'une solution garantit un dimensionnement correct
- En fait, mieux qu'un critère binaire, ce système permet de caractériser le dimensionnement optimal :
  - Remplacer le membre droit de chaque inéquation h_0
  - Rechercher sa valeur minimum $(h_0)_{min}$ telle que le système ait une solution
    - Si $(h_0)_{min} < 1$, garantie de dimensionnement et obtention de la valeur de réserve CPU
    - Sinon, on obtient l'évaluation de la surcharge CPU
    - Dans tous les cas, on a aussi un cas d'exécution des AE des tâches atteignant cette charge
- Le produit synchronisé représente la complexité exacte de l'entrelacement des AE des tâches dans le temps
  - Mais toujours quelque soit le découpage dynamique qui sera réalisé par l'algorithme d'ordonnancement implanté dans le noyau
    - Maîtrise fine des asynchronismes liés aux temps d'exécution (quotas) et au parallélisme d'exécution
  - Complexité selon l'application, combinatoire (cf. option RATS/RSF)

Exemple (option RATS et RSF) (1/4)

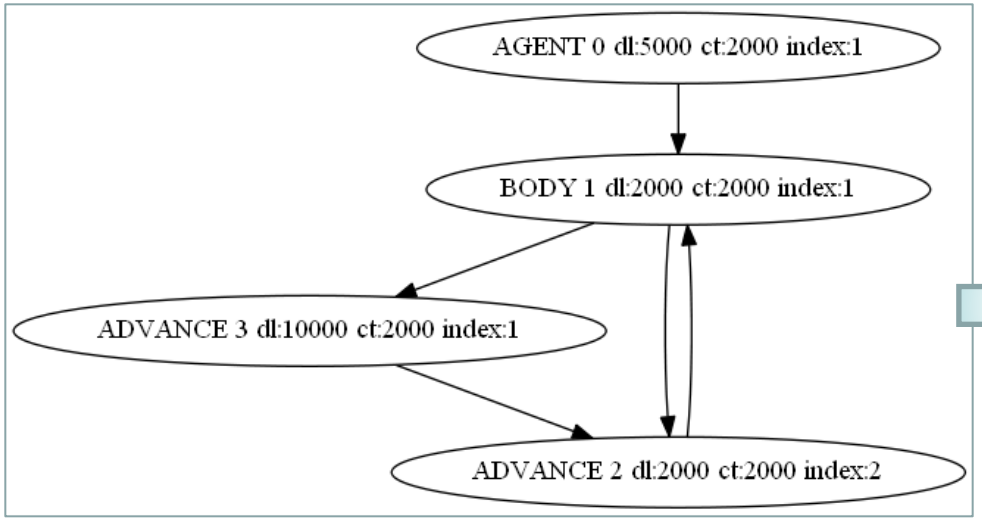
```
agent ping(starttime = 1 with cping) with
cbase
{
  body start
  {
    edit("Ping start\n");
    value = INIT_VALUE;
    advance 1 with cping;
    next send_ping;
  }
  body send_ping
  {
    int val = value;
    if (value == 10)
    {
      value = 0;
      advance 10;
    }
    next receive_ping;
    advance 1;
  }

(...)
```

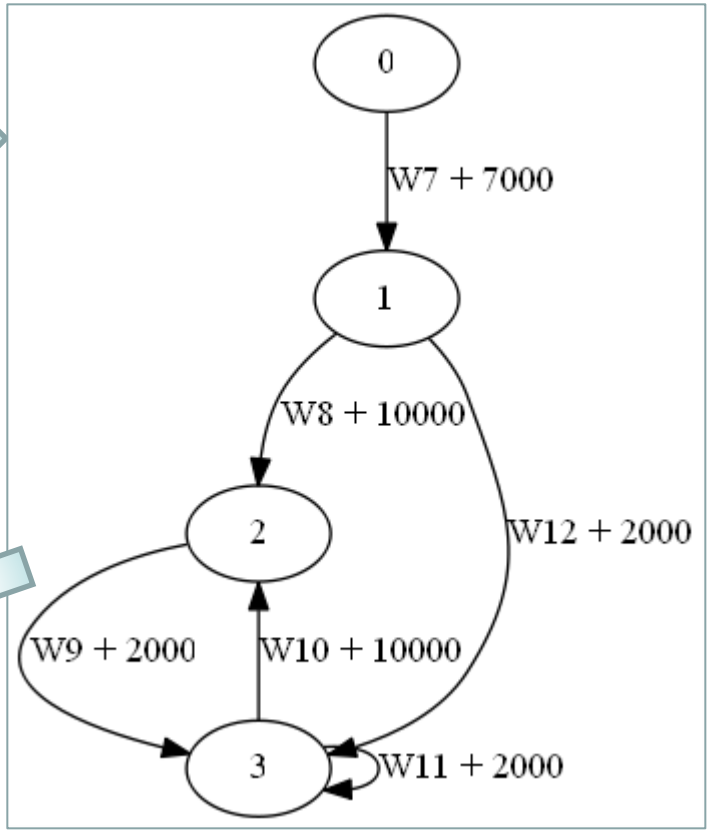


Exemple (option RATS et RSF) (2/4)

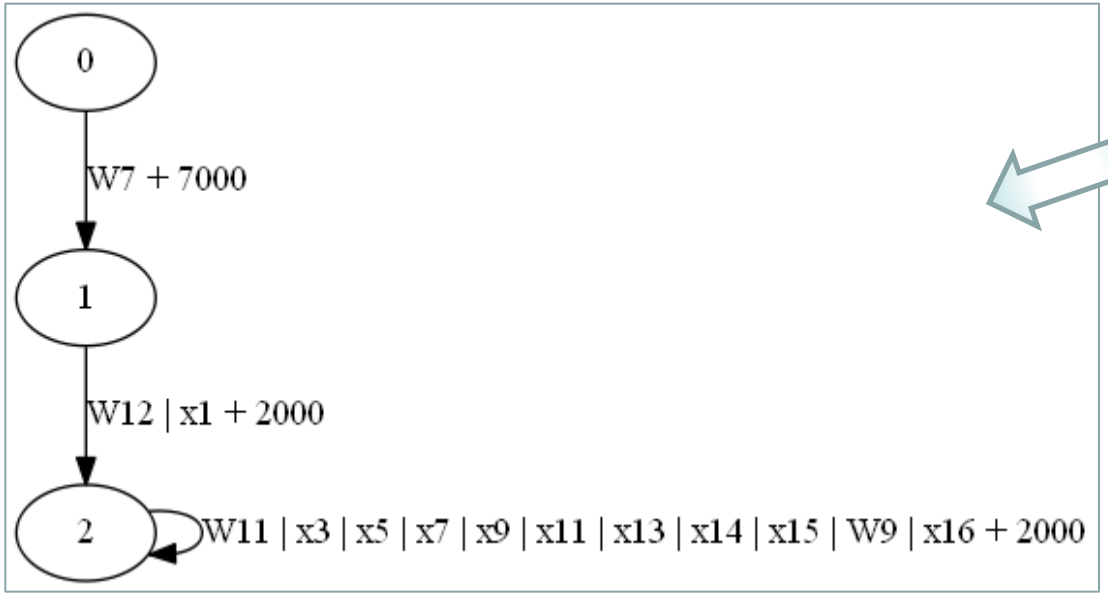


Exemple (option RATS et RSF) (3/4)



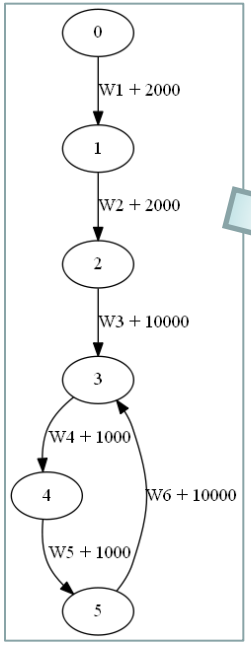


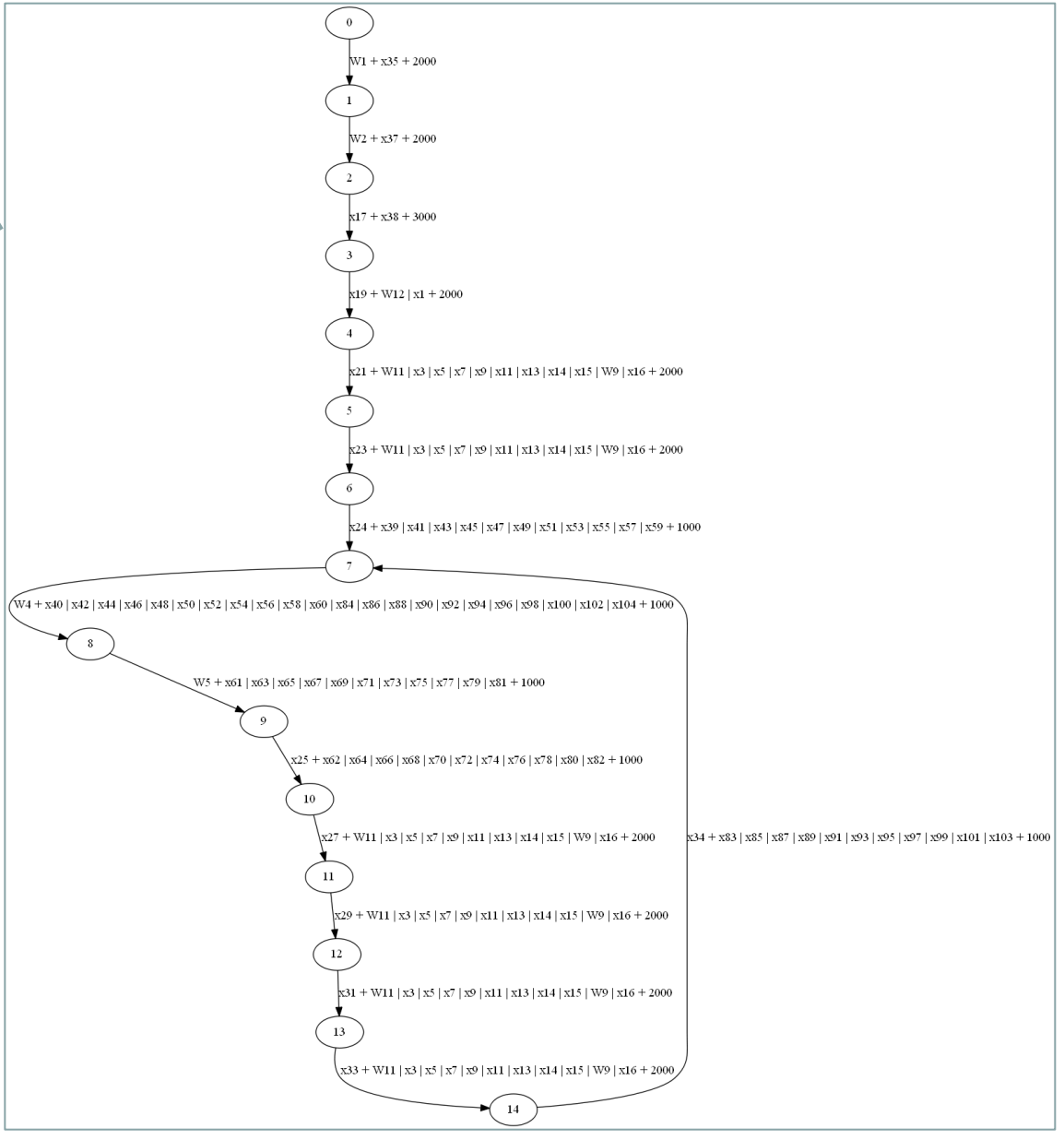
GRAPE

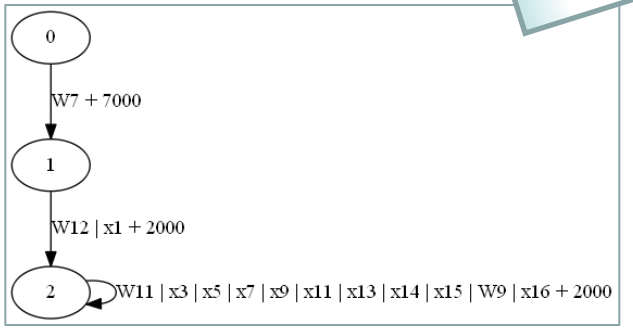


Exemple (option RATS et RSF) (4/4)

RATS ping



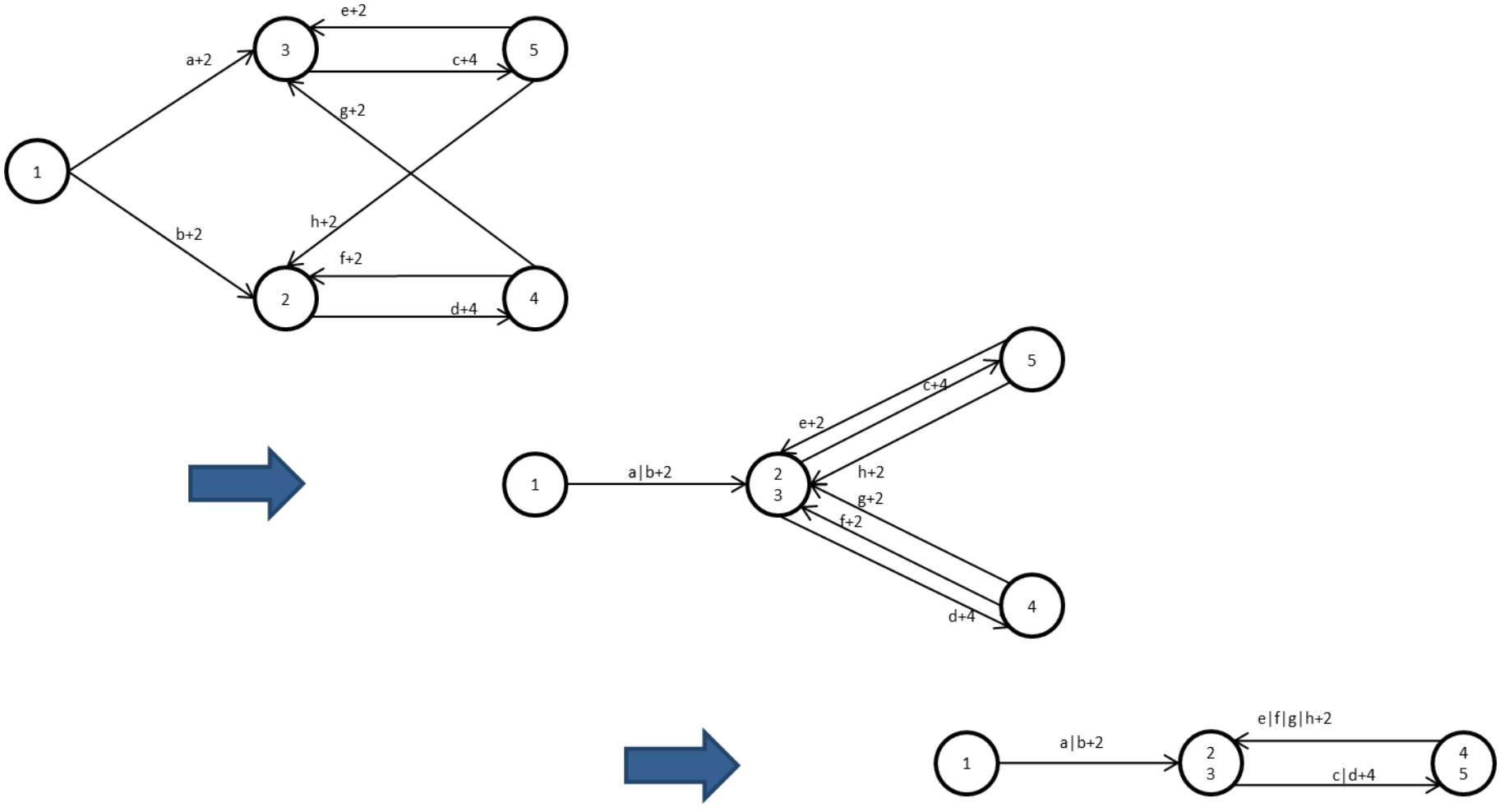




RATS pong

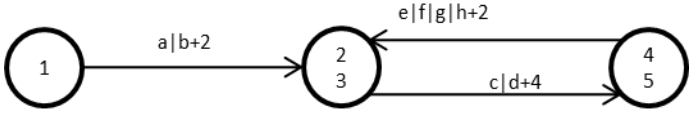
Option « Cyclic Graph Reduction » (1/5)

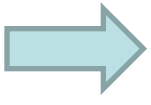
Ex 1 reduction

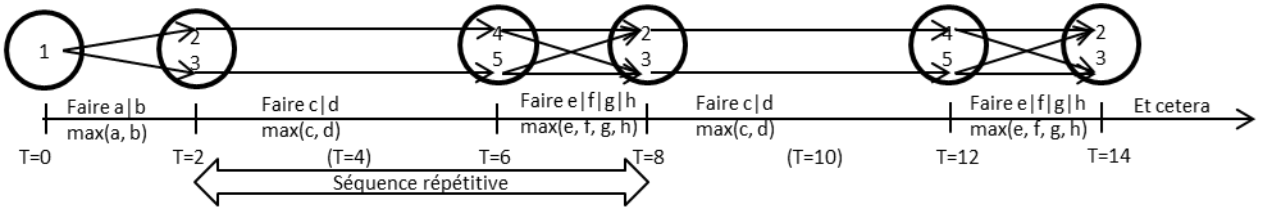


Option « Cyclic Graph Reduction » (2/5)

Ex 1 reduction

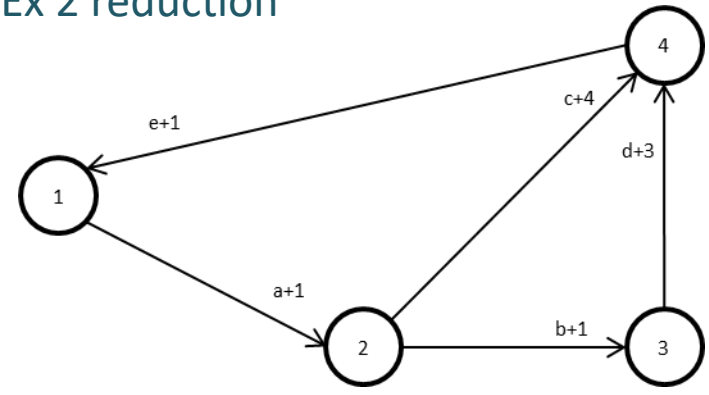


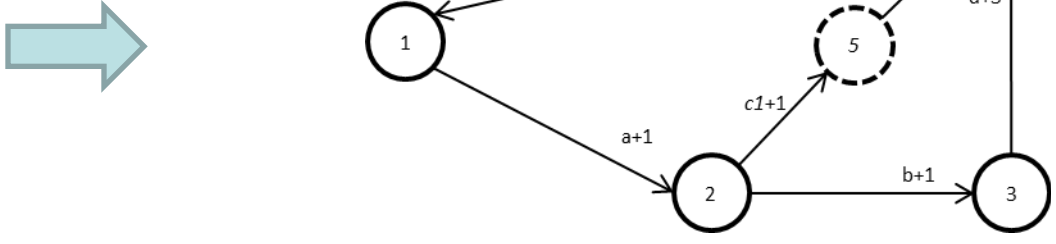


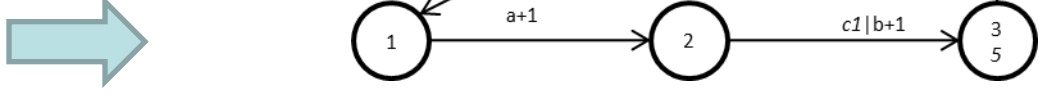


Option « Cyclic Graph Reduction » (3/5)

Ex 2 reduction

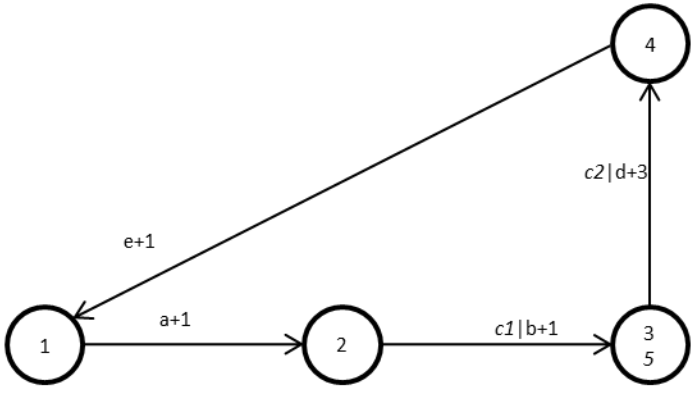


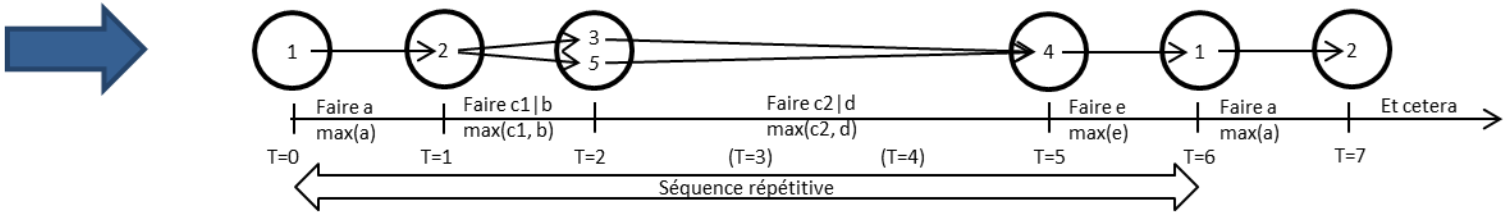




Option « Cyclic Graph Reduction » (4/5)

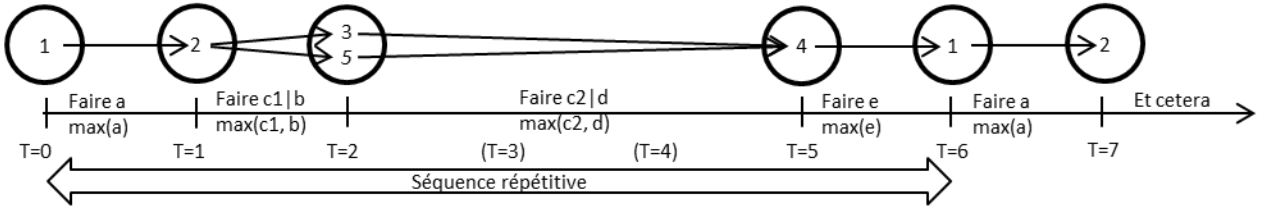
Ex 2 reduction

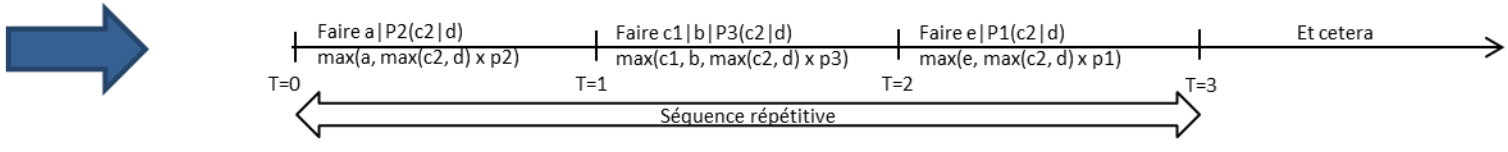




Option « Cyclic Graph Reduction » (5/5)

Ex 2 arbitrary reduction





Ce qu'apporte OASIS pour le test

- Mêmes objectifs et potentiel de test que pour les approches séquentielles
  - Le comportement dynamique du système temps-réel est prédictible et invariant :
    - Tous les tests sont reproductibles
  - Recherche exhaustive du pire des cas : tous les comportements sont accessibles et testables
  - Chaque agent est testable séparément (facilite la réutilisation)
    - Modularité du code source jusqu'au binaire
- Capacité d'évaluer et de tester
  - Les performances temps-réel (cf. dimensionnement)
  - Le comportement fonctionnel d'une application (tout ou en partie)
- Le modèle inclue les aspects temps-réel
  - Les graphes de contrôle représentent le comportement dynamique des agents et contiennent les contraintes temporelles
  - Le déterminisme permet de faire des campagnes de tests reproductibles
    - Maîtrise temporelle de l'injection des données d'entrées
    - Invariance du comportement de l'application entière

- Objectifs, contexte, concepts
  - Le déterminisme et la sûreté de fonctionnement
  - L'approche OASIS
- La mise en œuvre dans une chaîne d'outils
  - Compilation et génération de code
  - Runtime et noyau d'exécution
- Focus sur dimensionnement, testabilité et mise au point
- **Perspectives avancées**
  - Aspects distribués

Contexte multicalculateur avec composants standards

- Problèmes posés par le calcul distribué
  - recherche d'un placement optimal (performance) et sûr (SFC, etc.)
  - garantie de l'intégrité et l'uniformité des informations
  - dérive non uniforme des horloges locales

- Composants standards (Components Off-The-Shelf - COTS)
  - diffusion à grande échelle, robustesse de certains composants
  - pas entièrement conforme avec les exigences (effort d'intégration et d'adaptation)

=> Objectif : garantir le transport de l'information bout à bout (TR+SdF)

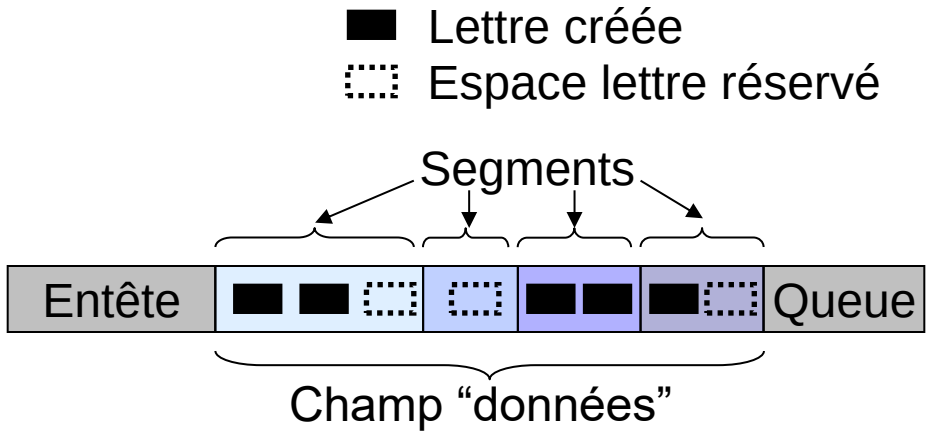
- Continuité et intégration avec le modèle OASIS
  - intègre déjà de manière réaliste les contraintes temporelles
    - délais de communication jamais estimés comme nuls
    - diffusion avant l'échéance, approvisionnement après l'échéance
  - couches systèmes et matérielles transparentes pour le développeur
  - application conçue indépendamment de l'architecture

- Interface TDMA avec Ethernet standard (physical layer)
  - aucune collision par construction (une collision devient une erreur)
  - garantir par construction la ponctualité des communications
  - dimensionnement et ordonnancement statique du réseau
    - pour vérifier et valider le réseau hors-ligne
    - pour détecter un comportement fautif

=> Cadencement global des communications connu de tous

Contenu et contenant des paquets

- Cycle réseau calculé automatiquement pour chaque application
- Contenu (Lettres)
  - structure d'encapsulation des variables temporelles et des messages
- Contenant (Paquets & Segments)
  - dimensionnement hors-ligne et constant



↳ Durant l'exécution, la longueur du contenant est indépendante de la quantité de son contenu

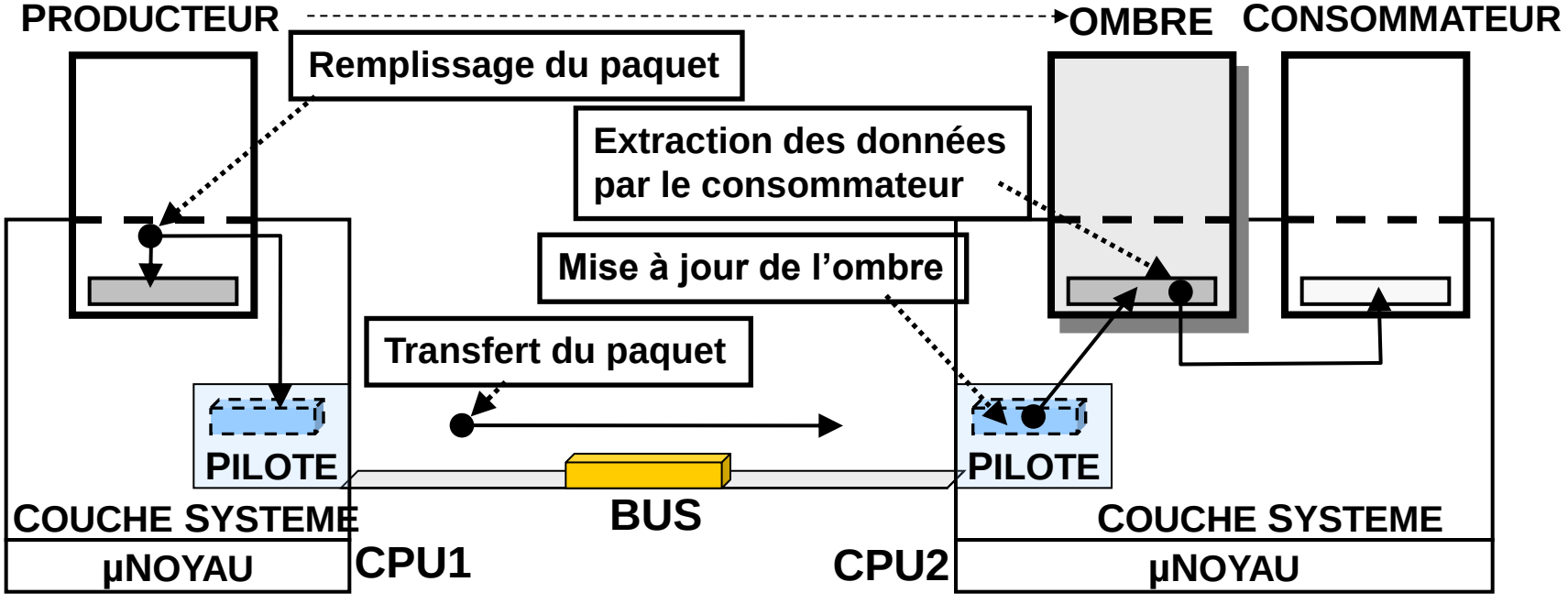
↳ Communications répétitives et prédictibles

- Facilite la vérification du dimensionnement
  - théorie (résolution du système d'équations et d'inéquations)
    - le pire des cas est calculable
  - pratique
    - le pire des cas est systématique

Concept de communications intégrées

- Préserver la couche système et le µNoyau
  - maintenir le niveau de sûreté atteint
  - disposer d'une solution modulaire
- Concept d'ombre
  - clone mémoire du producteur
  - localisée sur chaque CPU consommateur
- Concept de pilote
  - tâche TT de la couche système
  - contrôle les préparations et réceptions
- Cadencement du réseau
  - par construction, un seul CPU occupe le média de communication à un instant donné
- Optimisation possible (A-TDMA)
  - utilisation du mécanisme de détection de porteuse
  - « chevauchement » de plages par construction
    - seulement entre deux émetteurs consécutifs
    - inversion de paquets impossible

  ↳ espace entre deux paquets consécutifs optimal



Apports de la solution

- Extension transparente et intégrée

- Gestion des communications performante et sûre
  - parallélisme des communications
    - pas de problème de sections critiques
    - pas d'opérations séquentielles d'élaboration des paquets

    ↳ les tâches communiquent à leur propre rythme en parallèle

    ↳ le pilote se limite à indiquer l'emplacement des paquets en corrélation avec la plage prévue

  - Occupation du réseau optimale
    - pas de perte de bande passante inutile
      - µNoyau déclenche précisément la transmission
    - plages de longueur nécessaire et suffisante

↳ Ponctualité toujours garantie pour toutes les données transférées