Méthodologie Modélisation

Bertrand Granado

Biblio

Cours de Christophe Jego

De quoi on parle ?

- **De plein de choses comme :**
  - Qu'est qu'une méthodologie
  - Qu'est qu'un modèle
  - Quelles sont les outils pour décrire un modèle

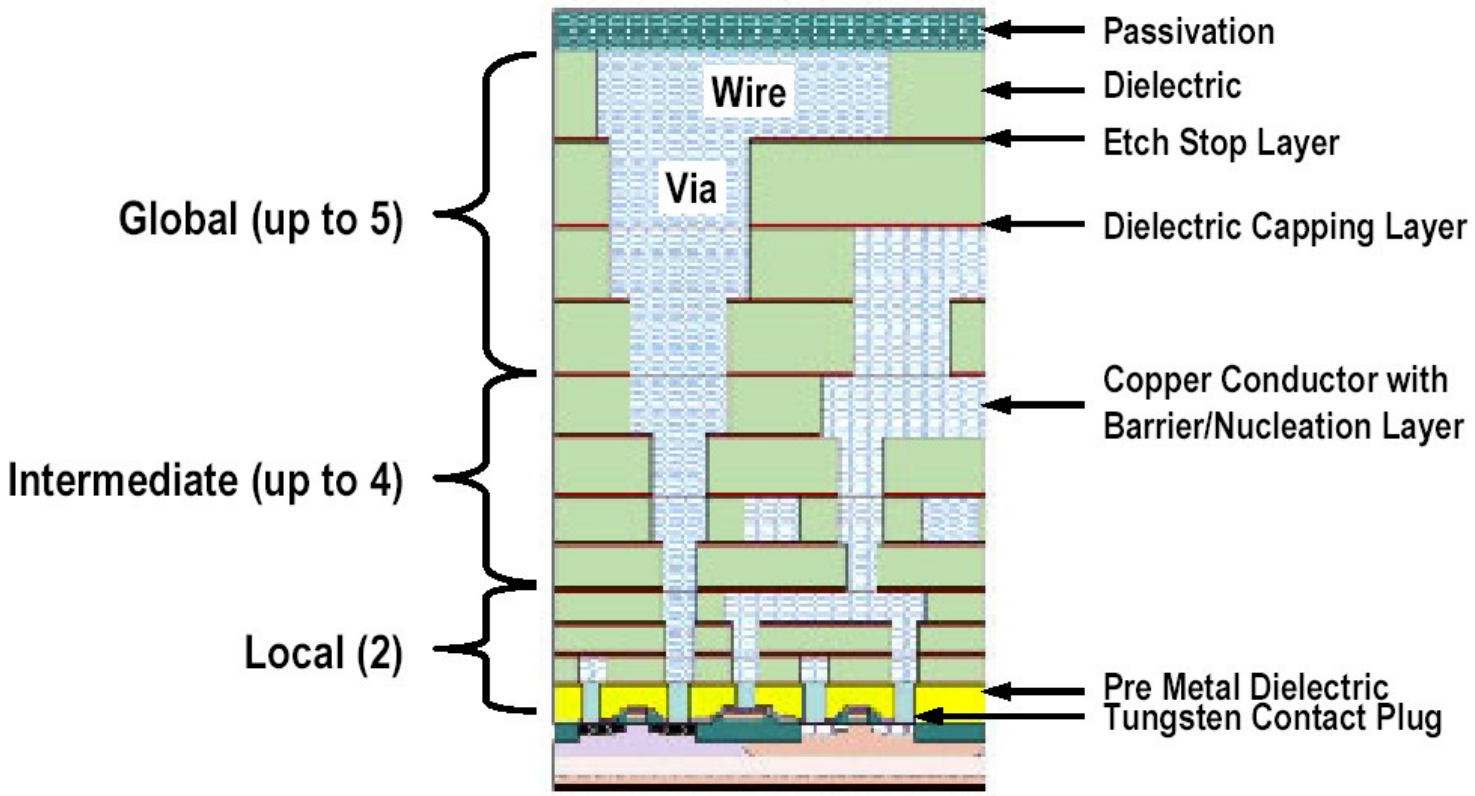
Le constat

International Technology Roadmap for Semiconductors

	2010	2011	2013	2015	2020	2024
Technologie (υ m)	45	38	27	21	11,9	7,5
Taille des puces ∑ P & ASIC (mm^2)	99	140	140	88	111	88
Densité d'intégration υ P & ASIC (Mtr/cm^2)	781	1104	2209	3506	11130	28047
Fréquence d'horloge SOC (GHz)	5,9	6,3	7,3	8,5	12,4	16,6
Tension d'alimentation (V)	0,97	0,93	0,87	0,81	0,68	0,6

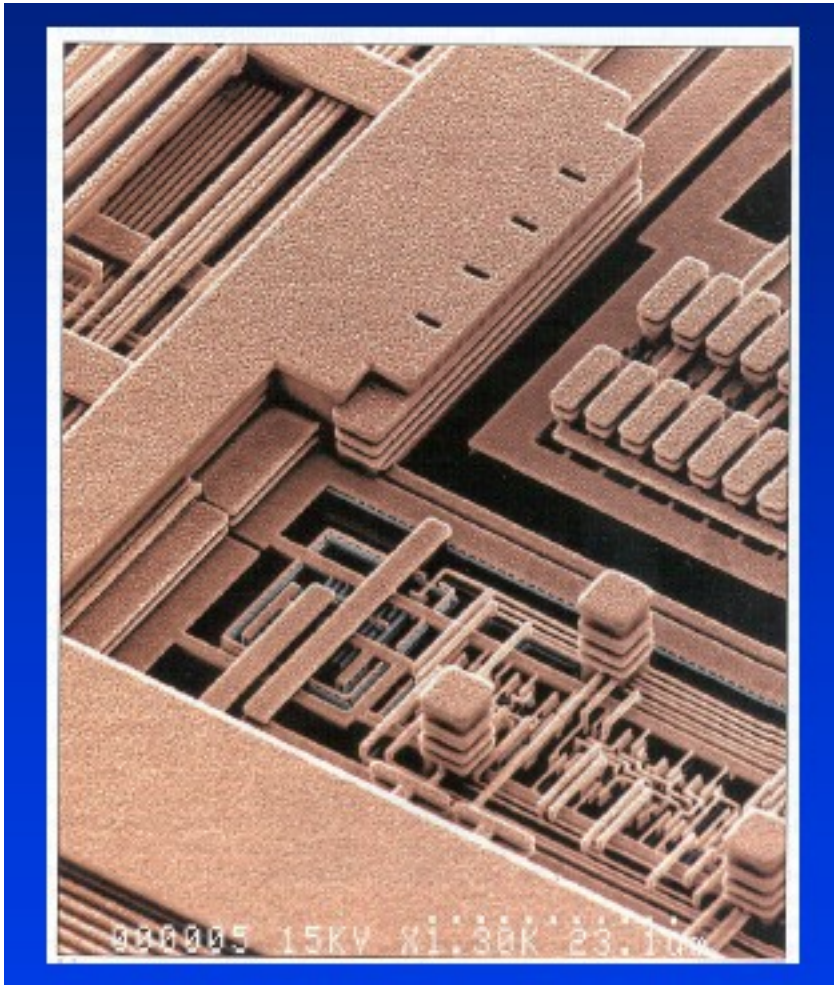
Source : November 2009 ITRS -http://public.itrs.net/

La technologie aujourd'hui

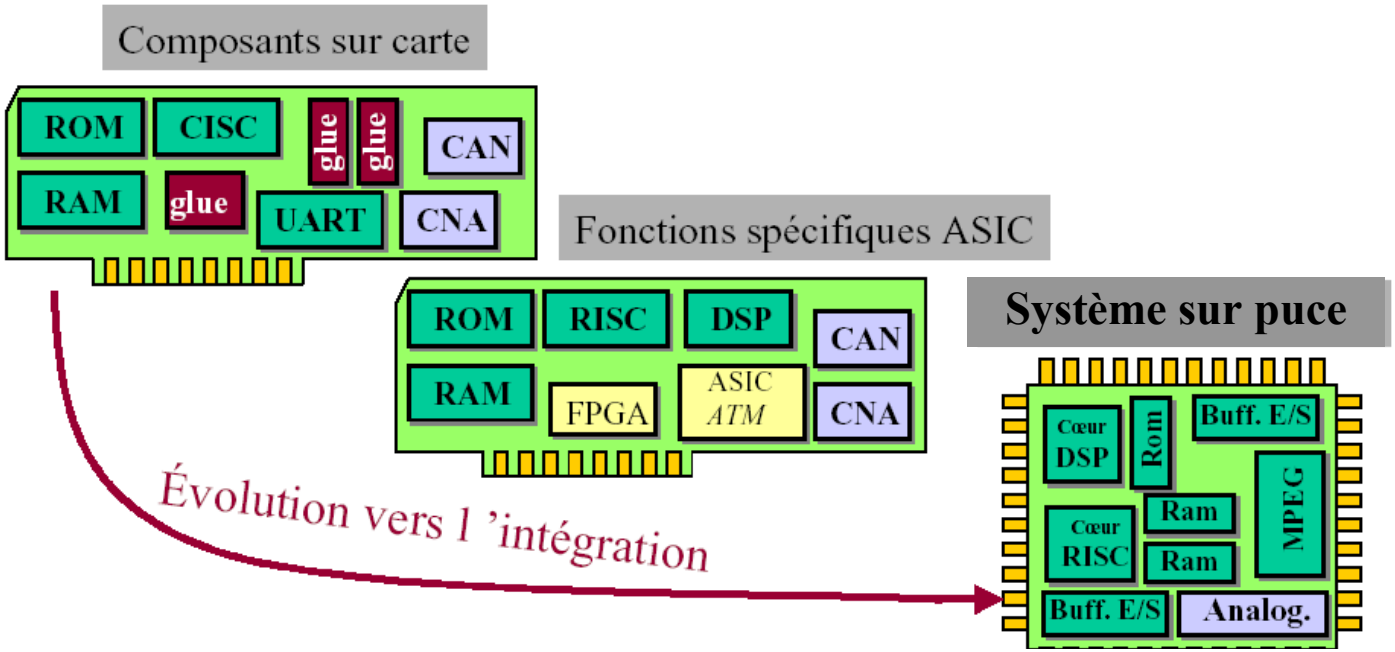


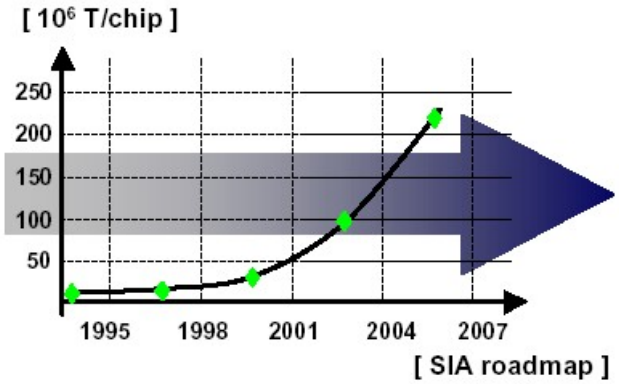
Source : 29 November 2001 ITRS Release Conference http://public.itrs.net/

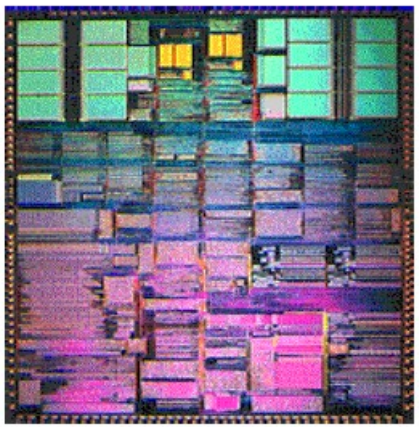
La technologie aujourd'hui



Perspective Système sur Puce

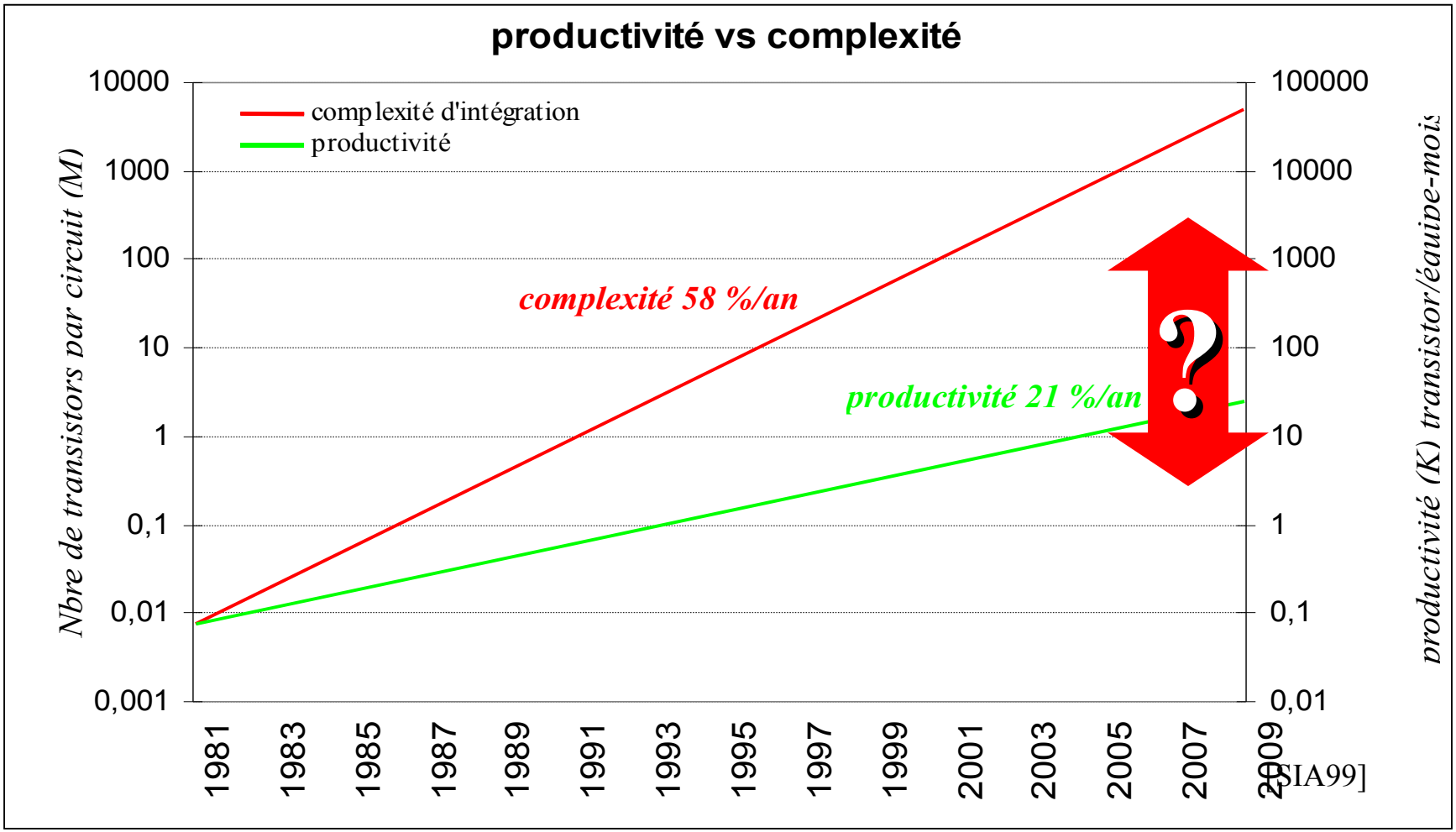






UNIVERSITÉ de Cergy-Pc

La Faille de l'ingénieur

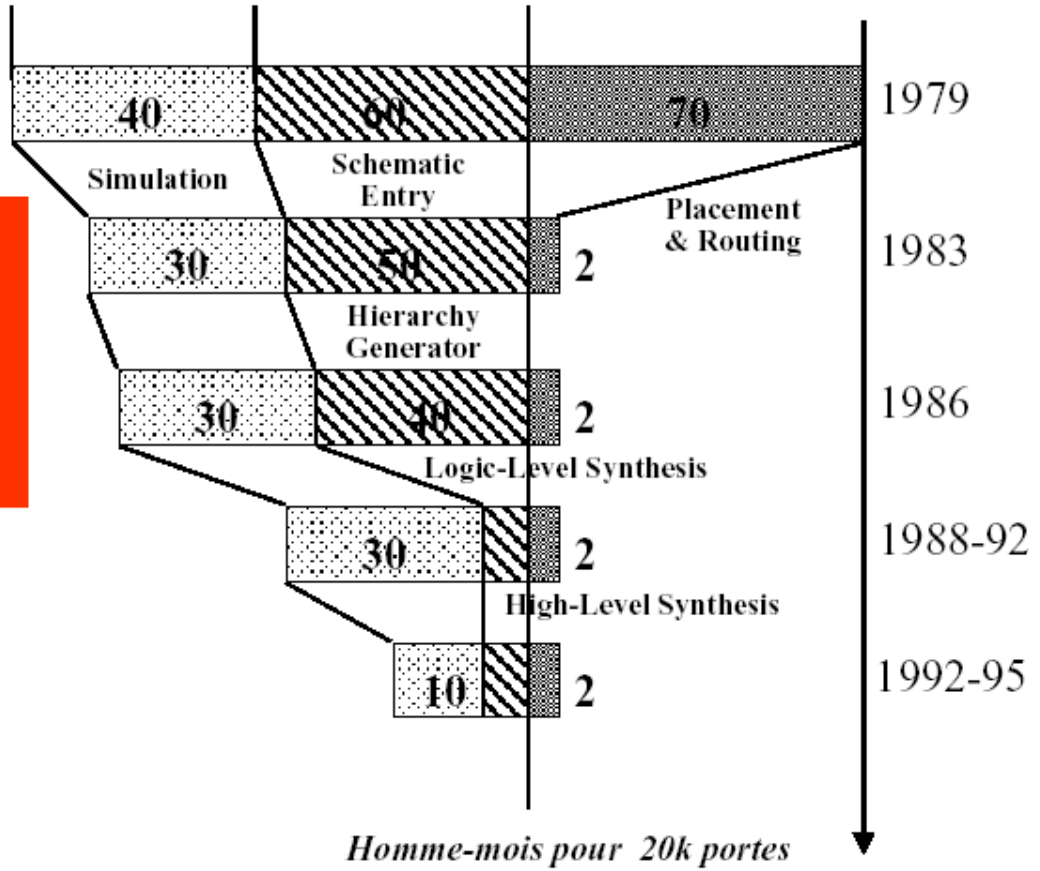


Introduction : Implications de l'évolution technologique

Par jour un ingénieur
- ecrit 10 lignes de code corrects
- dessine 10 portes
- place 10 cellules

Evolution de la technologie :
+ 58% Tr/an
Productivité concepteur :
+ 21% /an

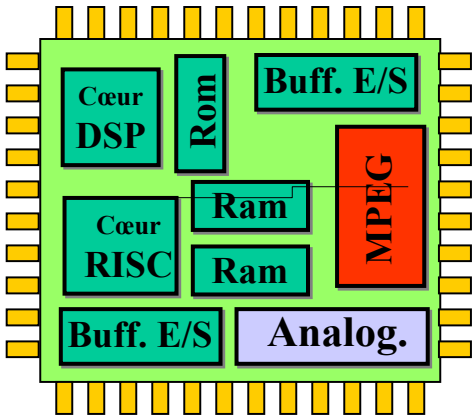
Actuellement :
- *Spécifications système*
- *Synthèse architecturale*
  Synthèse logique et RTL
- *CAO (saisie de schéma, simulation, P&R,...)*



Introduction : Implications de l'évolution technologique

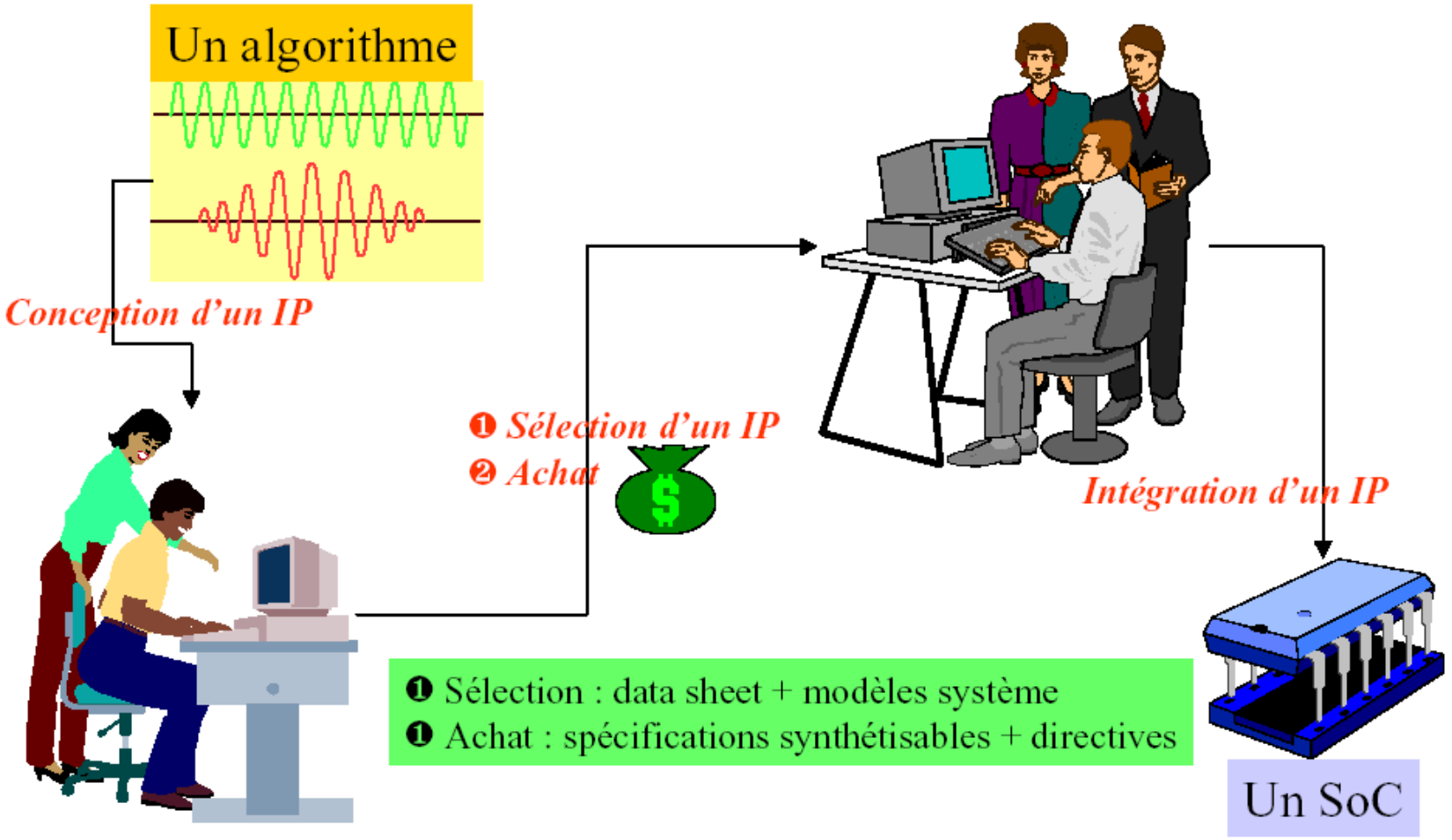
Notion d'IP (Intellectual Property)

- Réutiliser les blocs déjà conçus dans la société ;
- Utiliser les générateurs de macro-cellules (Ram, multiplieurs,…)
- Acheter des blocs conçus hors de l'entreprise.

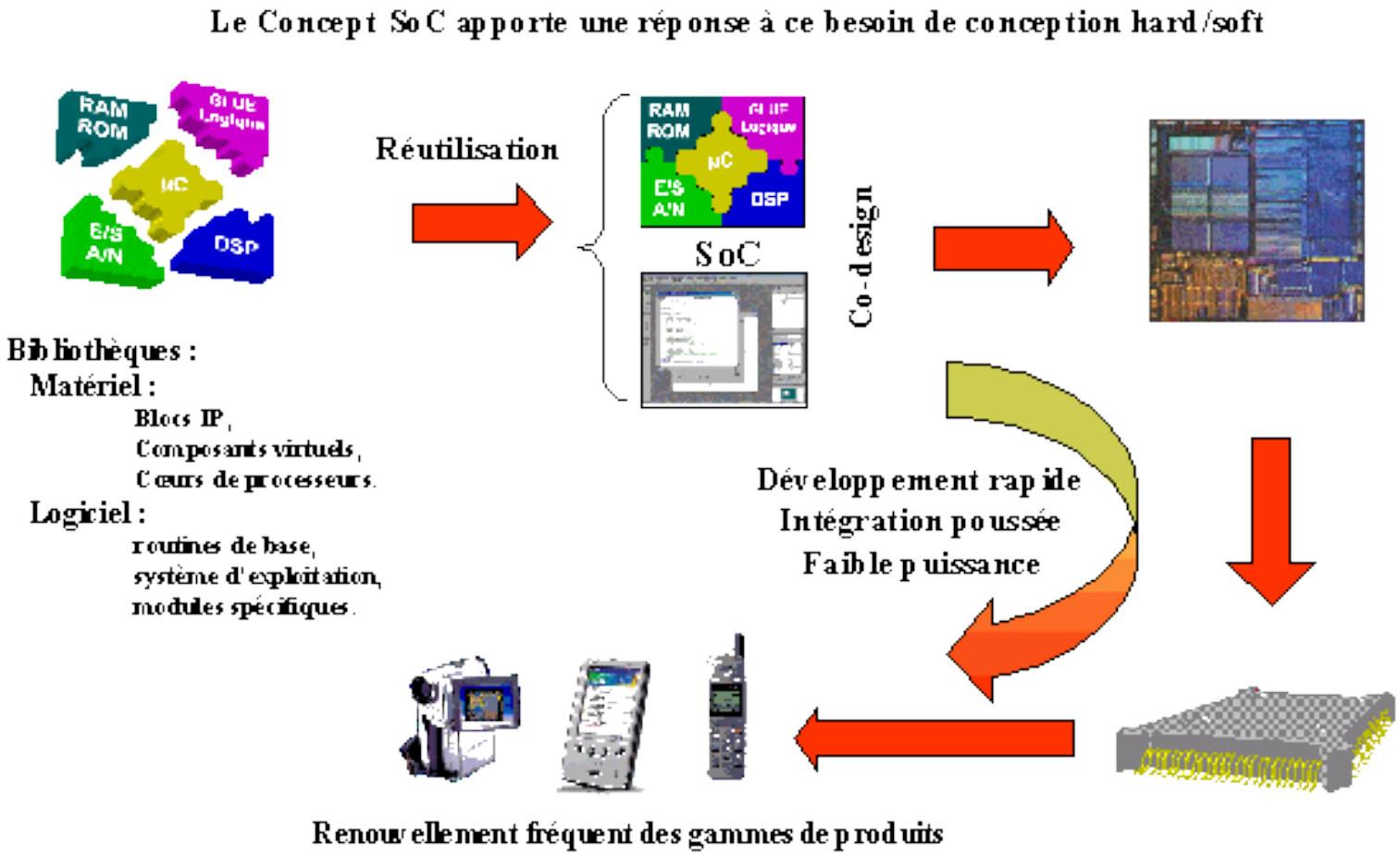


25% de conception personnalisée

Introduction : Implications de l'évolution technologique



Conception de SoC



Réalisation d'un SoC

- Réutiliser les blocs déjà conçus dans la société ;
- Utiliser les générateurs de macro-cellules (Ram, multiplieurs,…) ;
- Acheter des blocs conçus hors de l'entreprise.

Notion d'IP (*Intellectual Property*)

- **Blocs fonctionnels complexes réutilisables**
  - Matériel : déjà implanté, dépendant de la technologie, fortement optimisé
  - Logiciel : dans un langage de haut niveau (VHDL, Verilog, C++…), paramétrables
- **Normalisation des interfaces (OCP)**
- **Environnement de développement (co-design, co-specification, co-vérificatin)**
- **Performances moyennes (peu optimisé)**

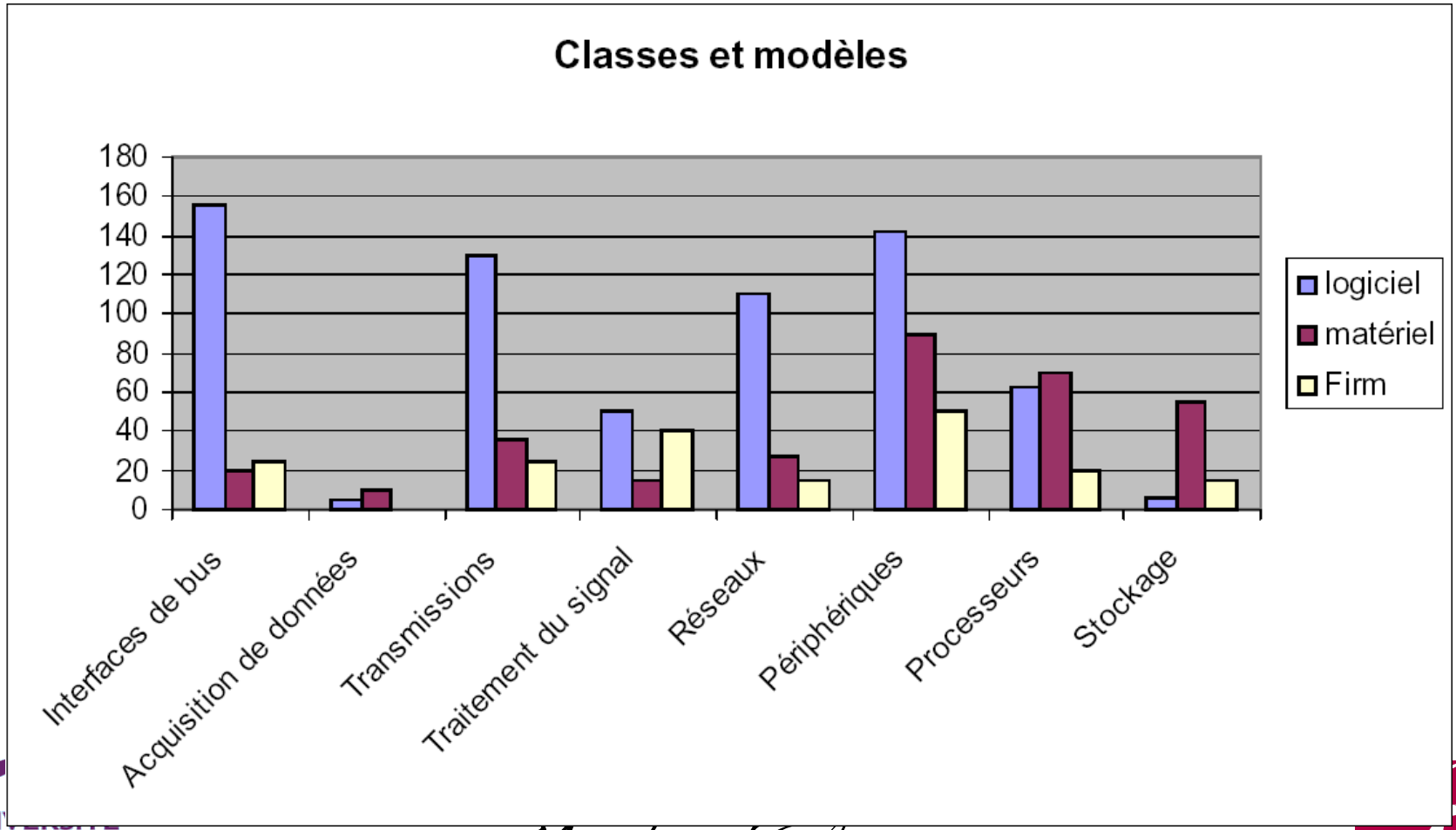
Utilisation d'IP

- **Bloc réutilisable (IP)**
  - connaître les fonctionnalités
  - estimer les performances dans un système
  - être sûr du bon fonctionnement de l'IP
  - intégrer cet IP dans le système
  - valider le système

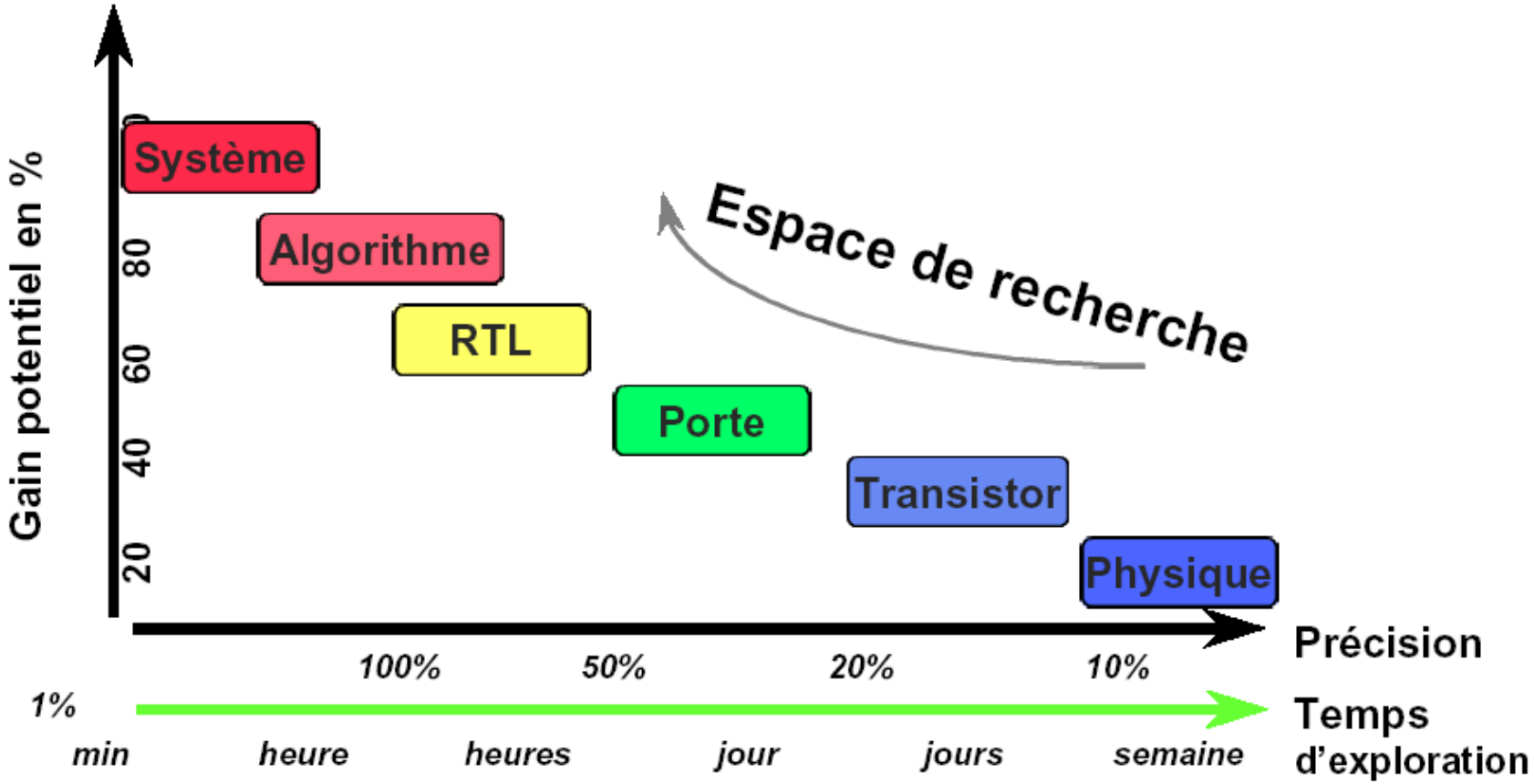
IP Type

	Flot de conception	Représentation	Librairies	Technologie	Portabilité
Logiciel (soft) Non prédictible Très flexible	Conception système Conception RTL	Comporte-mental RTL	-	Indépendant technologie	Illimitée
Firm Flexible Prédictible	Synthèse de plan de masse Placement	RTL & blocs Netlist	De référence • Temps • Dessin	Technologie générique	Portable sur librairies
MatÈriel (hard) Pas flexible Très prédictible	Routage Vérification	PolygÙnes Données	Spécifique process Règles de dessin	Technologie fixe	Dépendant du process

Commerce d ’IP « design & reuse »



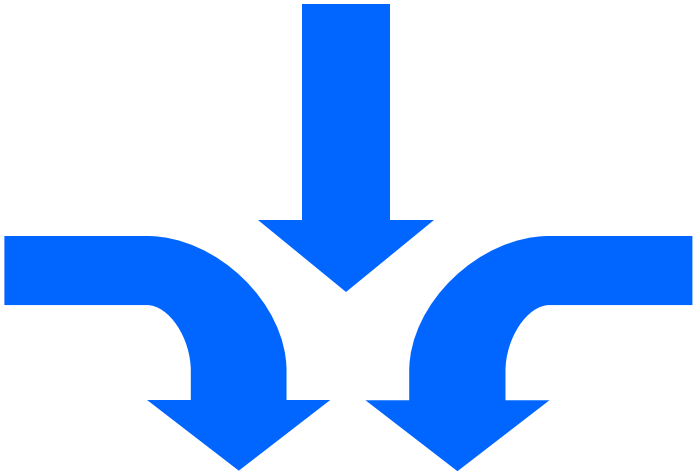
Introduction : Implications de l'évolution technologique



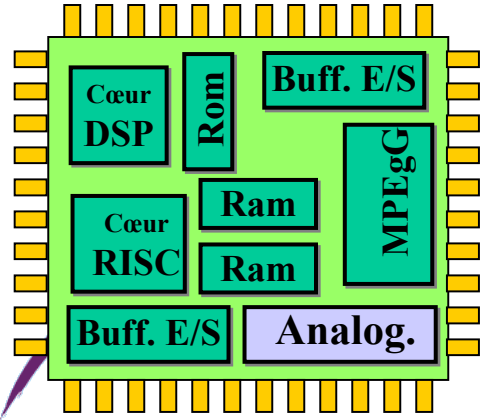
Introduction : Implications de l'évolution technologique

un flot de conception traditionnel

des capacités technologiques

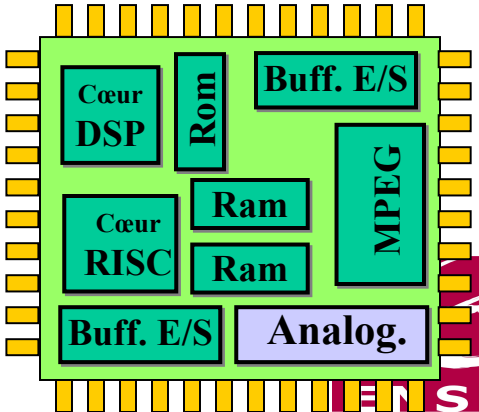


des orientations méthodologiques



Définition d'un flot de conception pour les Systèmes sur Puce

Master ESA



Méthodologie de conception : Nécessité & Objectifs

Nécessité d'une méthodologie

- Systèmes de grande complexité à intégrer
- Capacité d'intégration croissante
- Espace de recherche de solutions extrêmement vaste
- Temps de conception de plus en plus court (Pb de Time-to-Market)

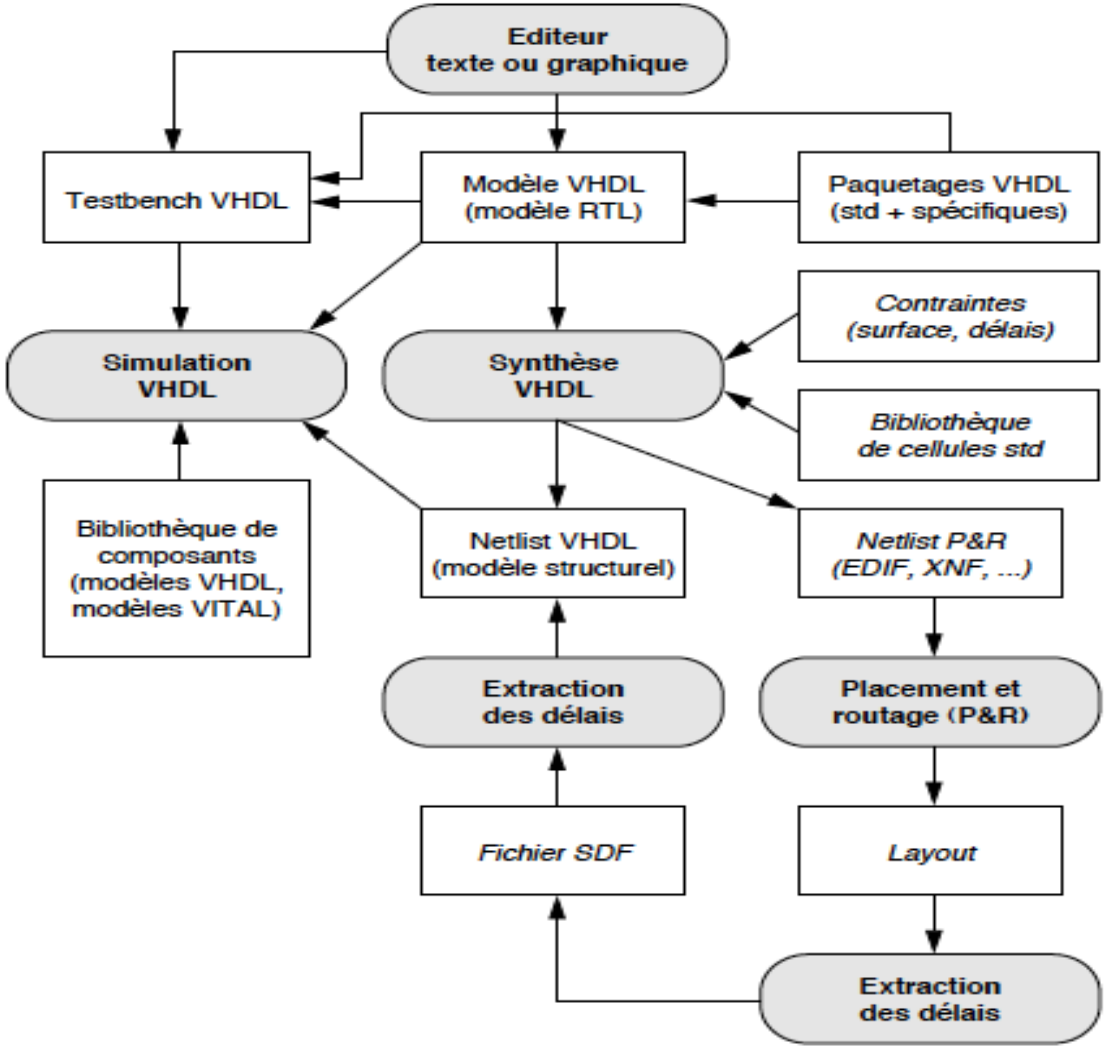
Objectifs

- Favoriser l'exploration de l'espace architectural
- Limiter les erreurs de conception (sûreté de fonctionnement)
- Utiliser les outils de CAO pour les tâches répétitives et laborieuses
- Rester indépendant le plus longtemps possible vis à vis de la technologie
- Prendre en compte les contraintes matérielles au plus tôt

Définition

- **Méthode:**
  - technique de résolution de problème caractérisée par un ensemble de règles bien définies qui conduisent à une solution correcte
- **Méthodologie:**
  - un ensemble structuré et cohérent de modèles, méthodes, guides et outils permettant de déduire la manière de résoudre un problème
- **Modèle:**
  - une représentation d'un aspect partiel et cohérent du «monde» réel
  - précède toute décision ou formulation d'une opinion
  - est élaboré pour répondre à la question qui conduit au développement d'un système

Méthodologie VHDL



Méthodologie de conception : Niveaux d'abstraction

- **Conception d'un circuit ou d'un système : passer d'un cahier des charges à une réalisation**

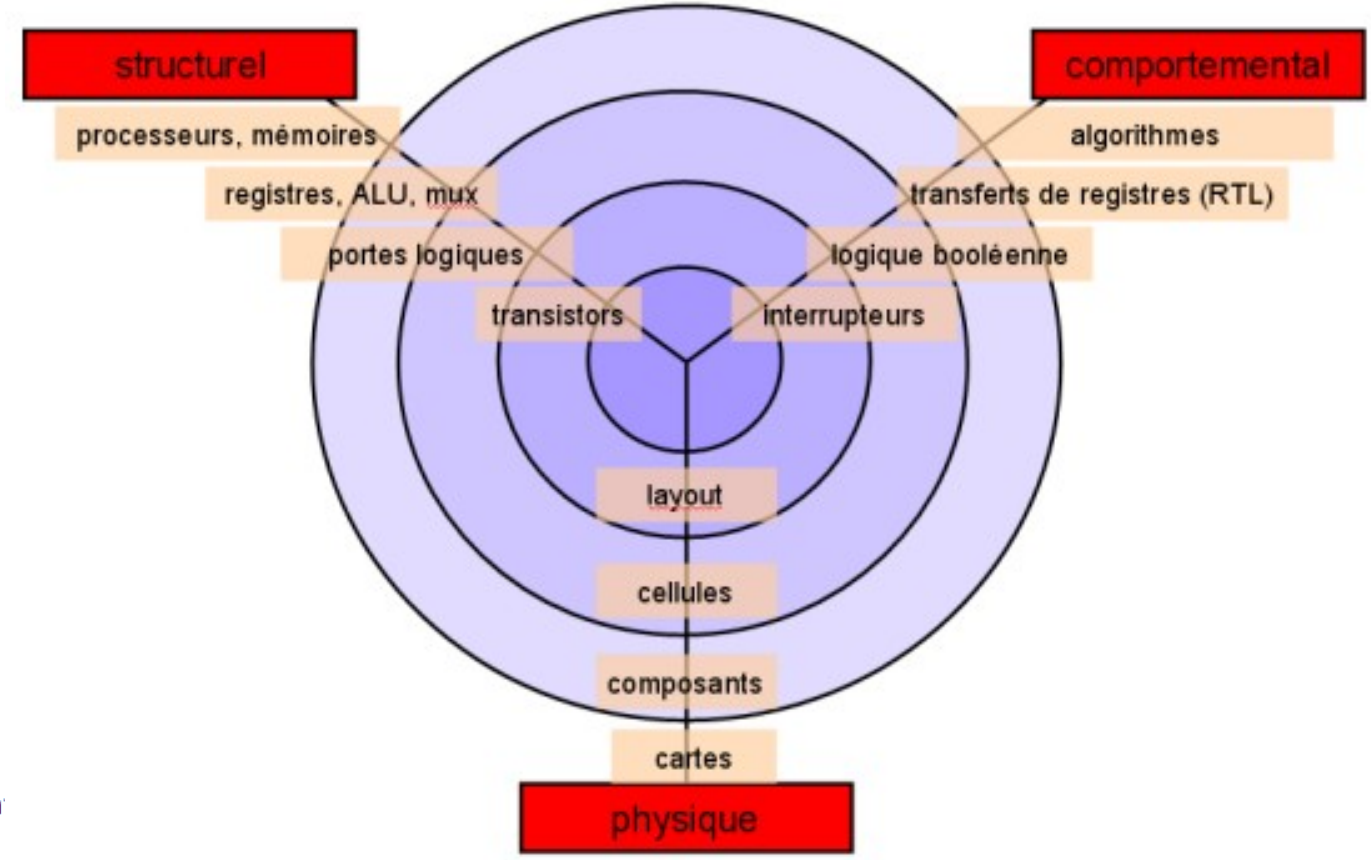
- **Quatre grands niveaux de conception :**
  - Niveau *Spécification* (ou système) : définition du problème
  - Niveau *Architectural* : agencement général de la réalisation
  - Niveau *Logique* (ou logiciel) : conception détaillée
  - Niveau *Implantation* : réalisation physique

- **La réalisation peut être matérielle, logicielle ou conjointe**

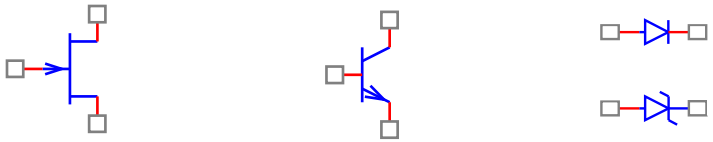
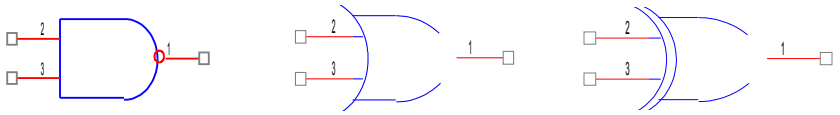
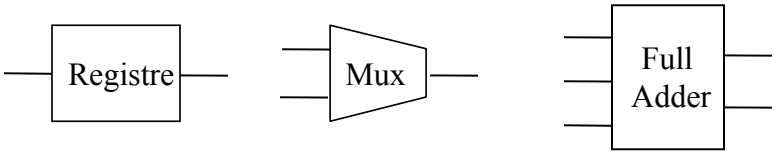
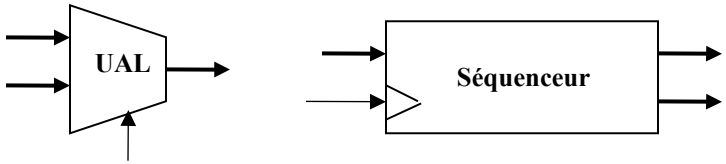
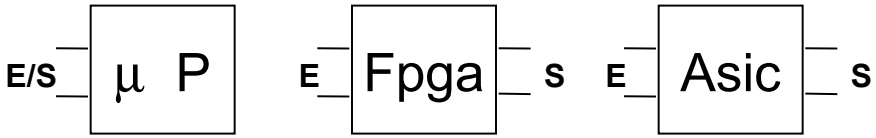
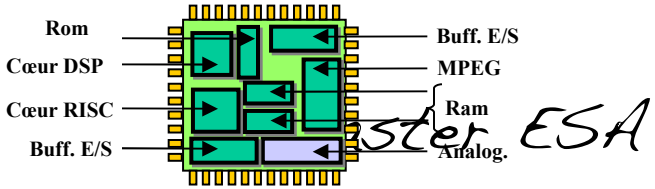
Méthodologie de conception : Niveaux d'abstraction

Diagramme en Y : *" The basic idea underlying the Y-chart is that each element of an electronic system can be described within three different domain [Gajski83] "*

A l'intérieur de chaque domaine, les éléments peuvent être décrit à différents niveaux d'abstraction. Les transitions à l'intérieur du diagramme en Y entre les domaines définissent les étapes de conception.



Méthodologie de conception : Niveaux d'abstraction

Type	Eléments de base	Conception
Composants discrets		**Conception électrique** Optimisation des caractéristiques électriques
SSI Small Scale Integration		**Conception logique** Optimisation des équations logiques
MSI Middle Scale Integration		**Conception numérique** Optimisation des traitements
LSI Large Scale Integration		**Conception architecturale** Choix des fonctionnalités
VLSI Very Large Scale Integration		**Conception fonctionnelle** Optimisation des implémentations matérielles et logicielles
ULSI Ultra Large Scale Integration		**Conception système** Optimisation conjointe des implémentations matérielles et logicielles

Méthodologie de conception : Flot de conception

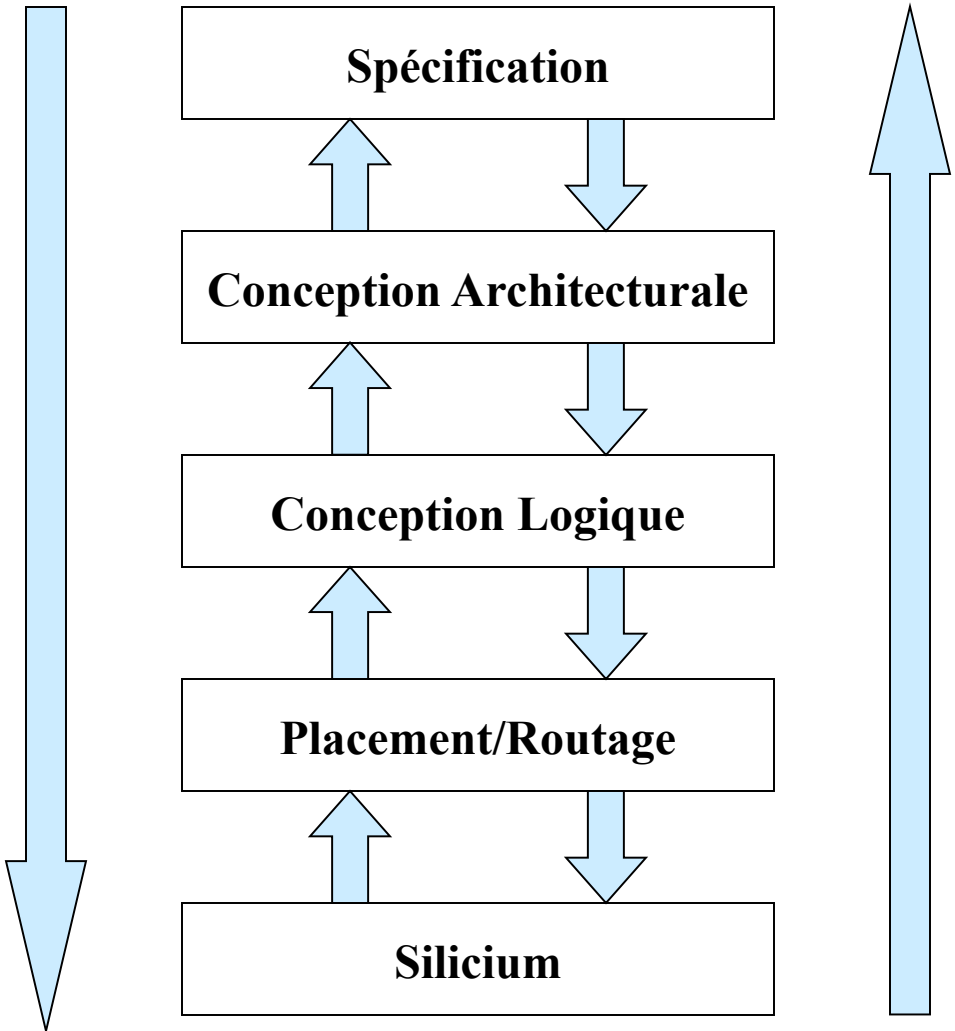
Démarche de conception

Top-Down Design

Bottom-Up Design

Démarche descendante

Raffinement de chaque constituant



Démarche ascendante

Abstraction sur un ensemble de constituants

Méthodologie de conception : Flot de conception

Démarche ascendante

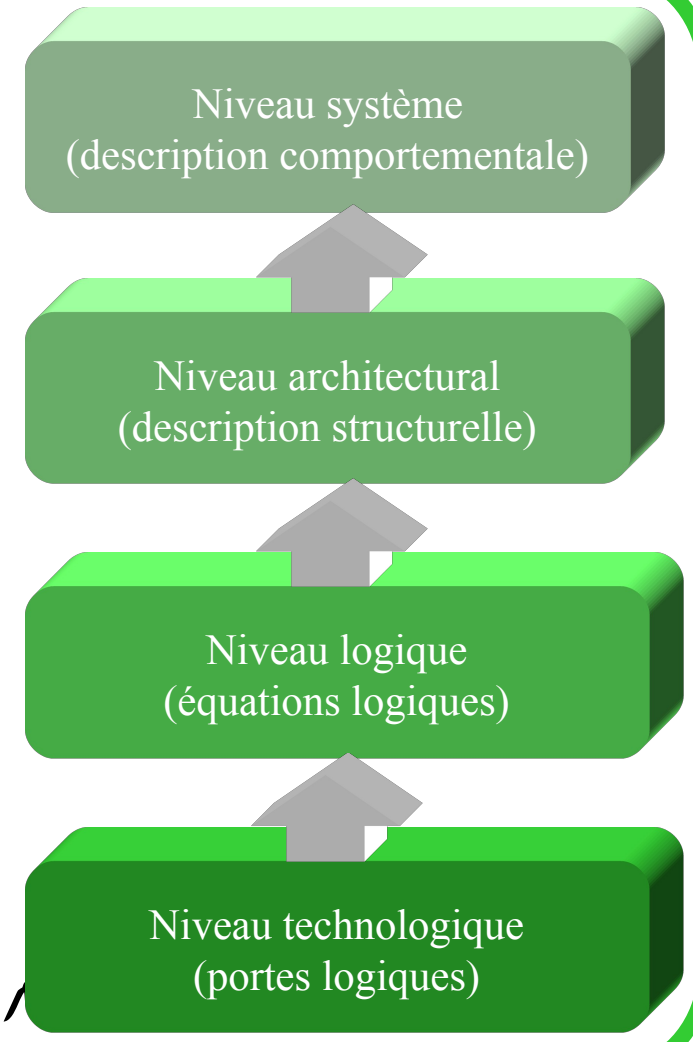
Domaine de travail

Algorithmie
Traitement du signal et de l'image

Ordonnancement
Sélection de composants
Machine d'états

Simplifications logiques
Equations booléennes

Physique du composants
Dimensionnement des transistors



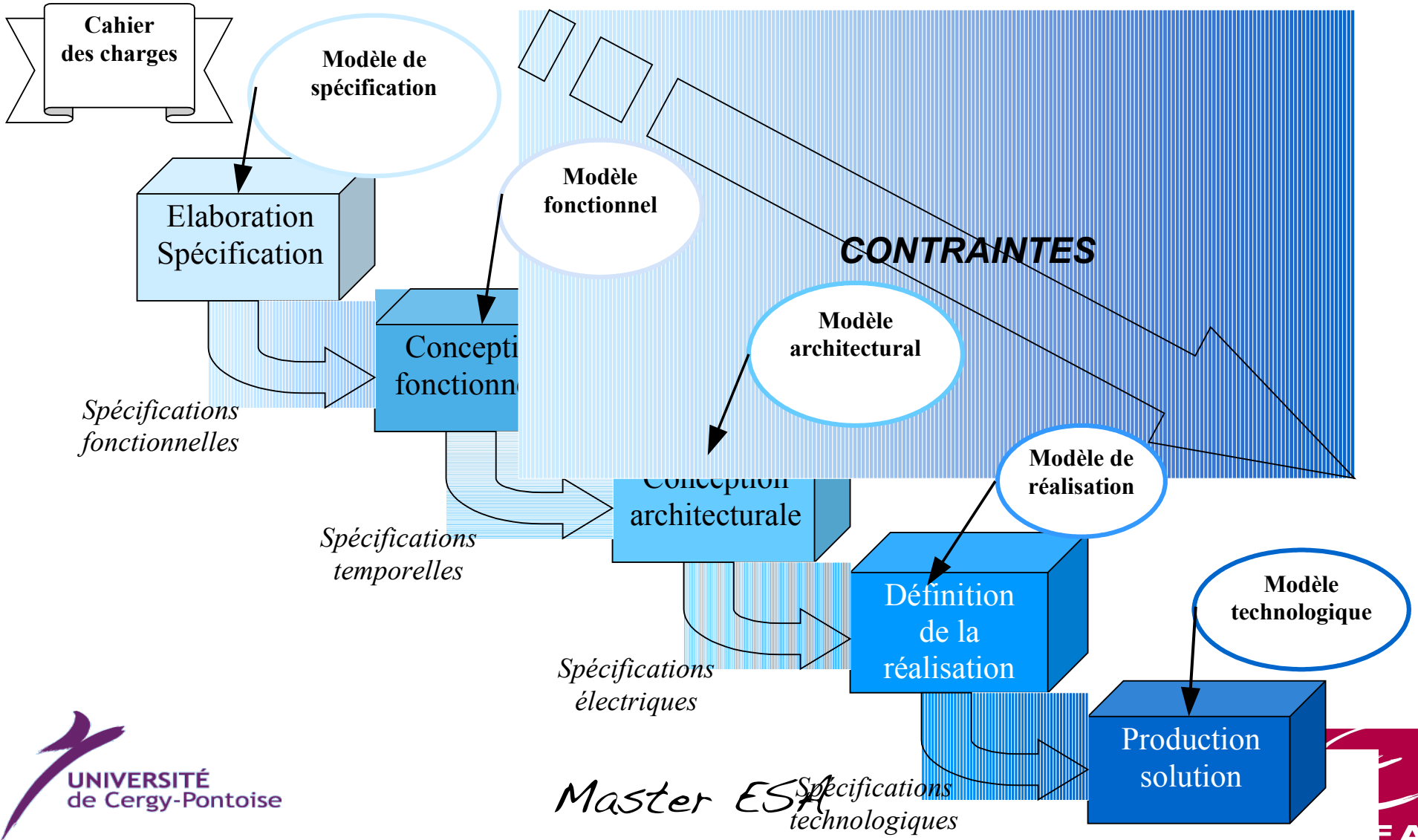
Boites à outil
pour le concepteur
du niveau supérieur

Méthodologie de conception : Flot de conception

- ***Caractéristiques d'une démarche ascendante:***
  - La conception débute par une phase d'optimisation électrique.
  - Une bibliothèque d'éléments optimisés est construite à chaque niveau (utilisée au niveau d'abstraction supérieur).
  - A chaque composant d'un niveau considéré, est associé une phase d'abstraction (délimitation du composant auquel est associé une fonction au niveau supérieur).
- ***Contraintes de la démarche ascendante:***
  - Nécessite une étude au niveau électrique longue et coûteuse qui ne peut être justifiée que pour des réalisations à grande diffusion.
  - Implique des contraintes de conception système difficiles à appréhender car on commence par la spécification de sous-ensembles en espérant progresser vers une solution architecturale pour le système global.

Méthodologie de conception : Flot de conception

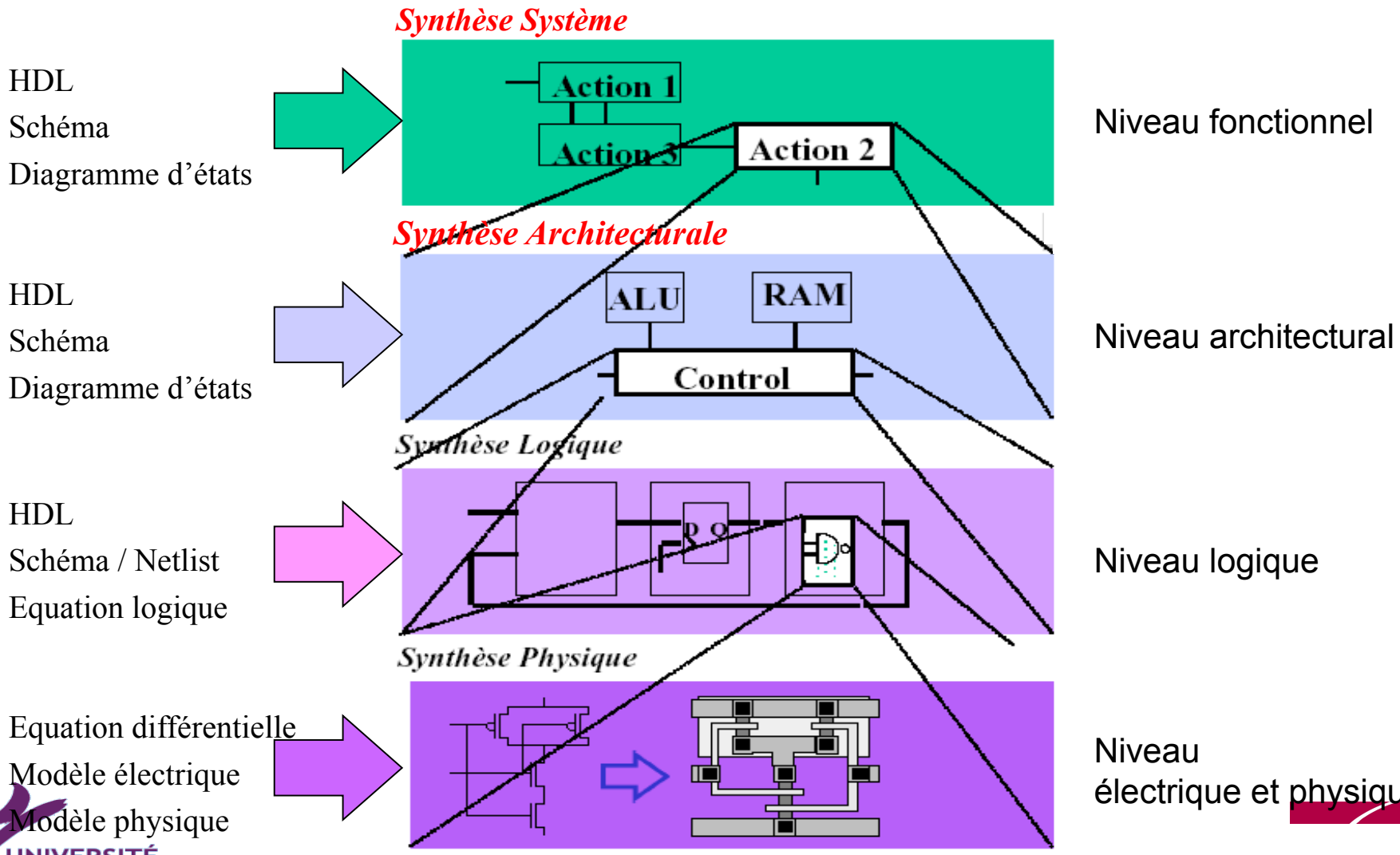
z



Méthodologie de conception : Flot de conception

- ***Caractéristiques d'une démarche descendante:***
  - ***Spécification*** : définition des différentes tâches fonctionnelles et des contraintes de conception (débit, surface, consommation…)
  - ***Conception fonctionnelle*** : détermination des fonctions internes, des protocoles et des échanges entre fonctions
  - ***Conception architecturale*** : détermination des opérateurs, gestion des chemins de données et du contrôle
  - ***Conception détaillée*** : transformation de la solution architecturale en circuit. Optimisation des caractéristiques temporelles
  - ***Réalisation*** : optimisation de l'implémentation au niveau des performances électriques et structurelles

Méthodologie de conception : Flot de conception



Méthodologie de conception : Flot de conception

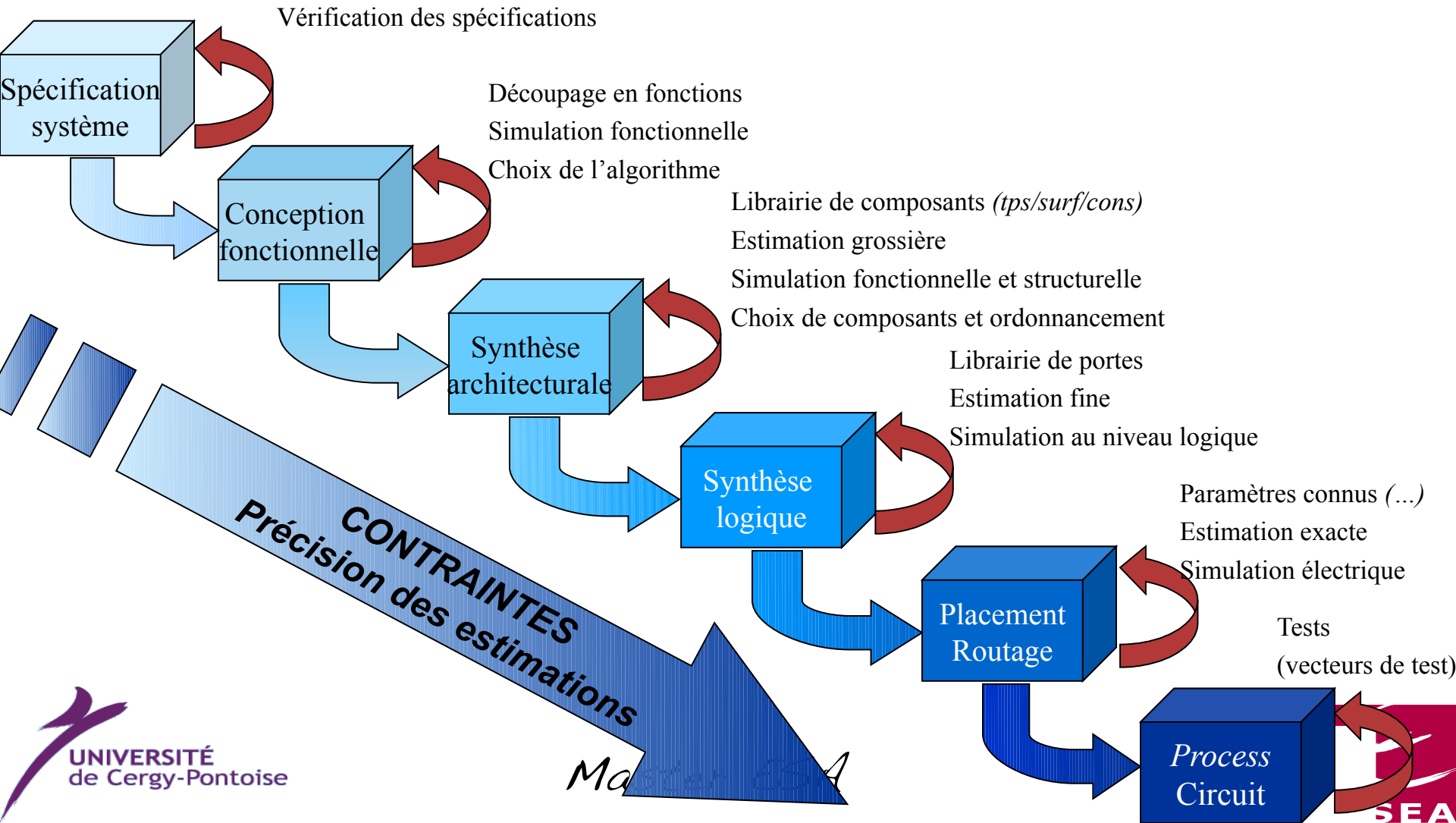
Méthode globale

- ✓ **Optimisations: gain dominant au niveau système**
  - transformations algorithmiques
  - choix des fonctions
  - organisation des données
- ✓ **Exploration de l'espace architectural**
  - éviter les remises en cause des décisions (rebouclage)
  - ordonner les prises de décision => heuristique de réduction du tps de conception
  - limiter l'espace de recherche des différents niveaux
- ✓ **Affinement progressif du circuit**
  - flot continu du niveau système au circuit
  - combler le fossé entre les concepteurs de système et les concepteurs de circuit
  - définition de modèles globaux qui peuvent être affinés
- ✓ **MAIS**
  - **Rebouclage complet du flot** => processus long et coûteux donc **à proscrire**
  - **Flot descendant sans *feedback* => illusoire**

Solution : optimisations locales et rebouclages partiels

Méthodologie de conception : Flot de conception

Méthode globale



UNIVERSITÉ de Cergy-Pontoise

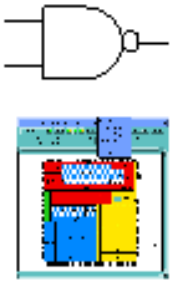
Méthodologie de conception : Tendance

Evolution des techniques de conception

- **Années 70/80 : Conception "Full Custom"**
  - Simulation : Spice

- **Années 80/00 : Conception "cell based" + FPGA...**
  - Réutilisation du matériel (briques : "standard cell")
  - Modélisation, Simulation, Synthèse

VHDL



- **Années 00/xx : Conception "SOCs"**
  - Réutilisation du matériel et du logiciel ("IP")
  - "Co-Design"
  - Vérification
  - "Time to Market", coûts,...

IP ?

Logiciel Embarqué ?

Niveau d'intégration ?

Méthodologie de conception : Tendance

- Mise en place d'un flot de conception continu des spécifications systèmes à l'implantation électrique
- Systèmes embarqués

  ⇒ (gestion de la consommation et de la mémorisation)
- Définir des stratégies pour concevoir des circuits en technologies sub-microniques
- Conception d'ASIP (*Application Specific Instruction set Processor*) et de leur compilateur associé
- Hardware/Software CoDesign

Le flot de conception CoDesign : Présentation générale

Le CoDesign logiciel/matériel propose une approche structurée pour la conception d'une classe de systèmes numériques ⇒ *les systèmes dédiées*

Quatre grands étapes sont distinguées dans l'approche CoDesign :

1-) Spécification

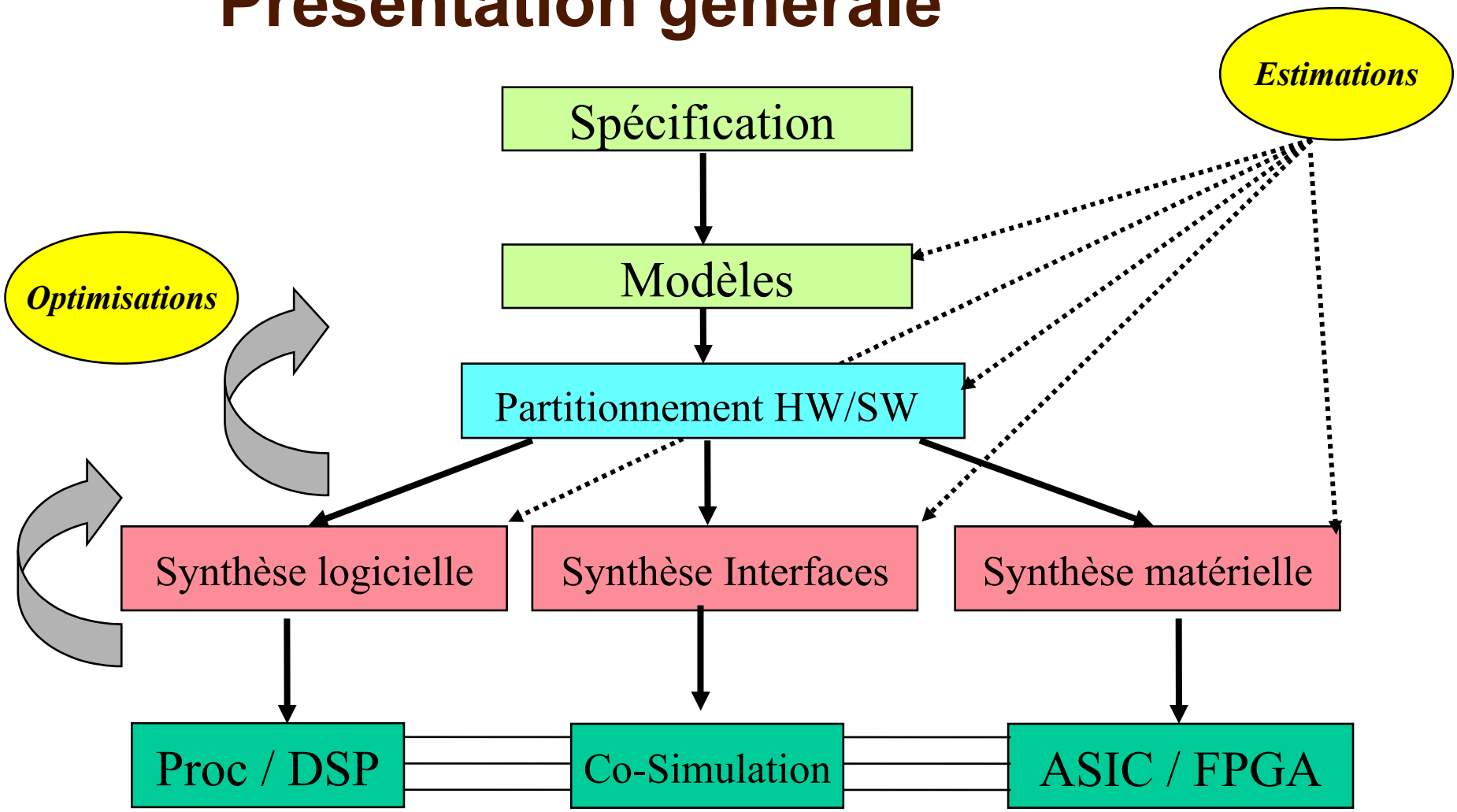
2-) Partitionnement

3-) Synthèse logicielle & matérielle

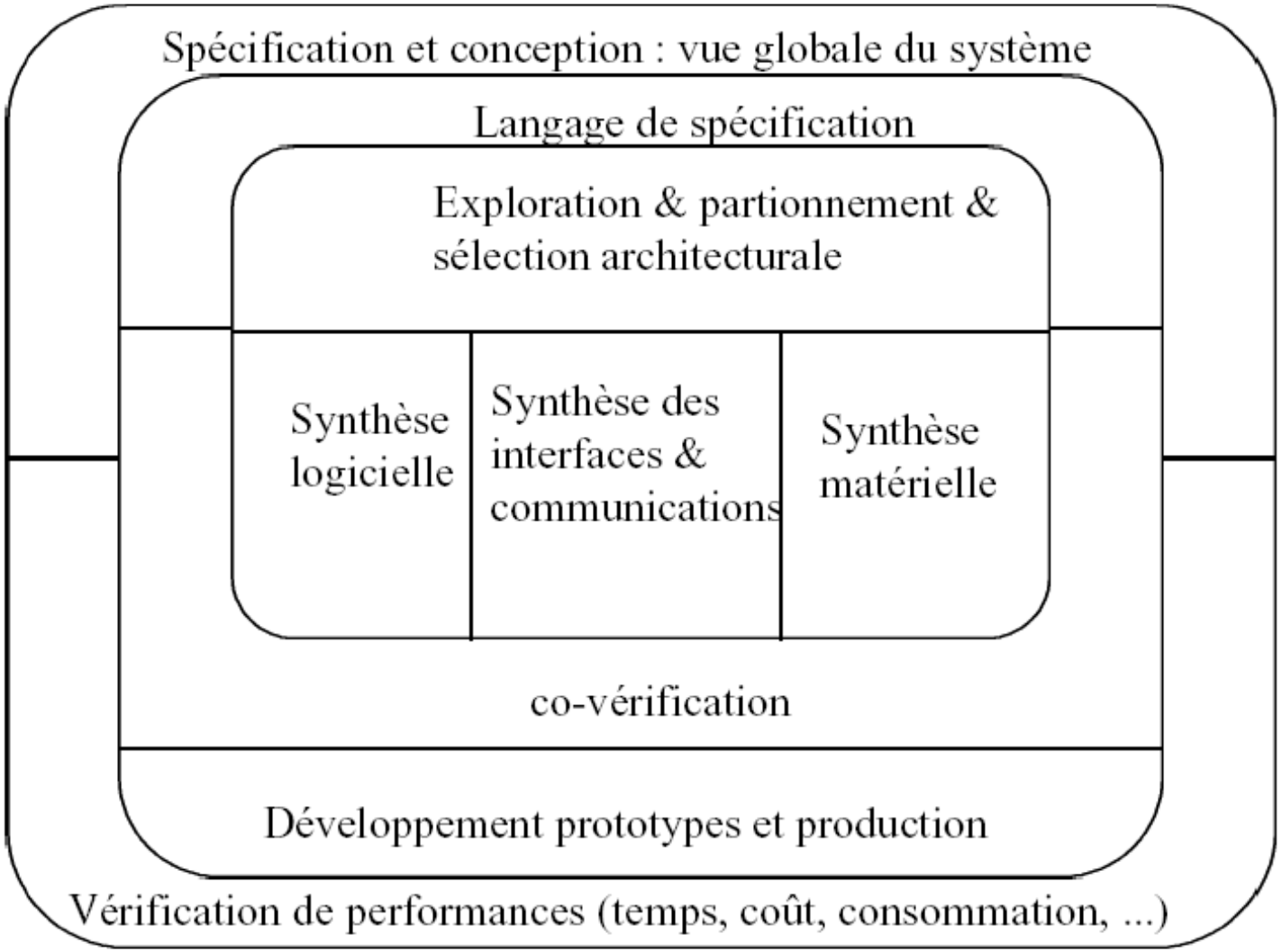
4-) Co-simulation

La réalisation finale est une réalisation conjointe matérielle/logicielle

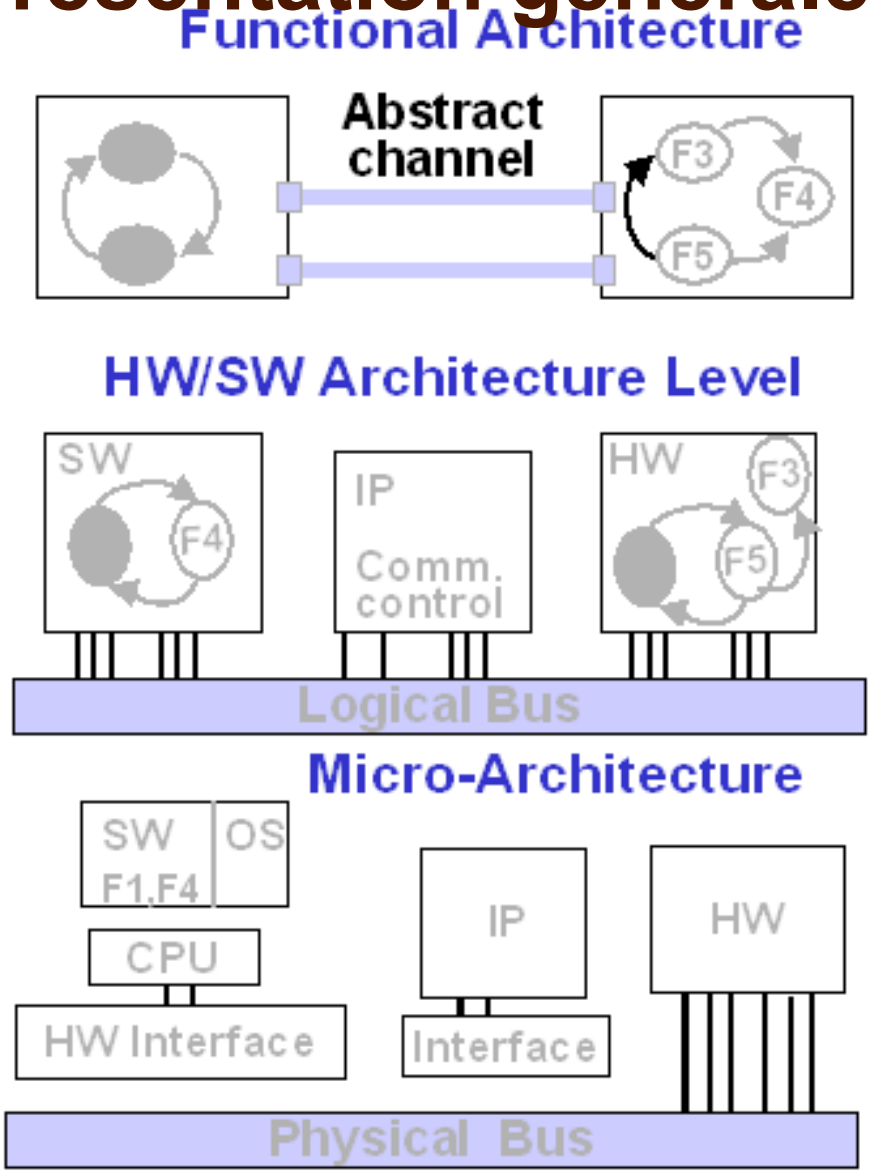
Le flot de conception CoDesign : Présentation générale



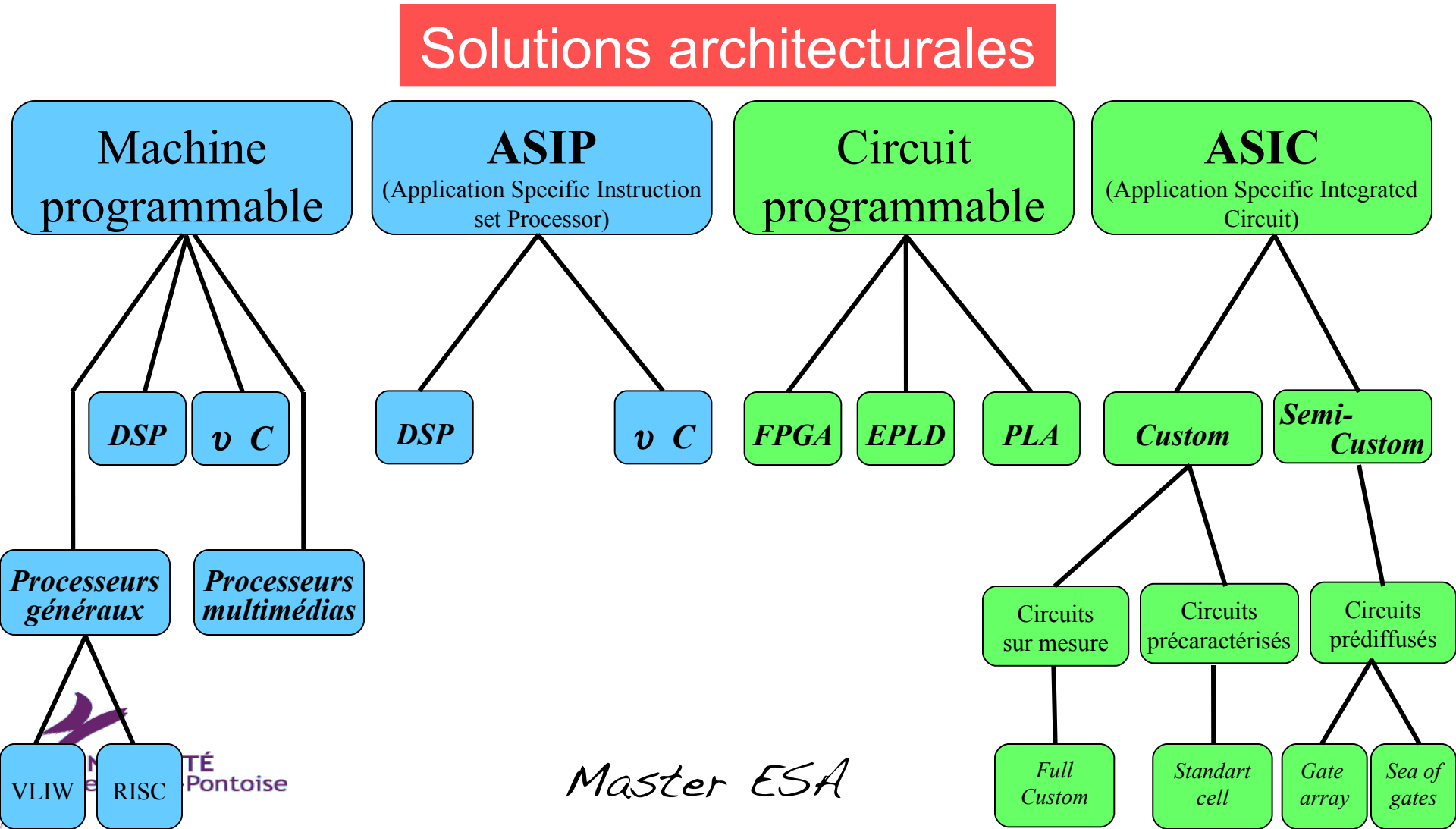
Le flot de conception CoDesign : Présentation générale



Le flot de conception CoDesign : Présentation générale

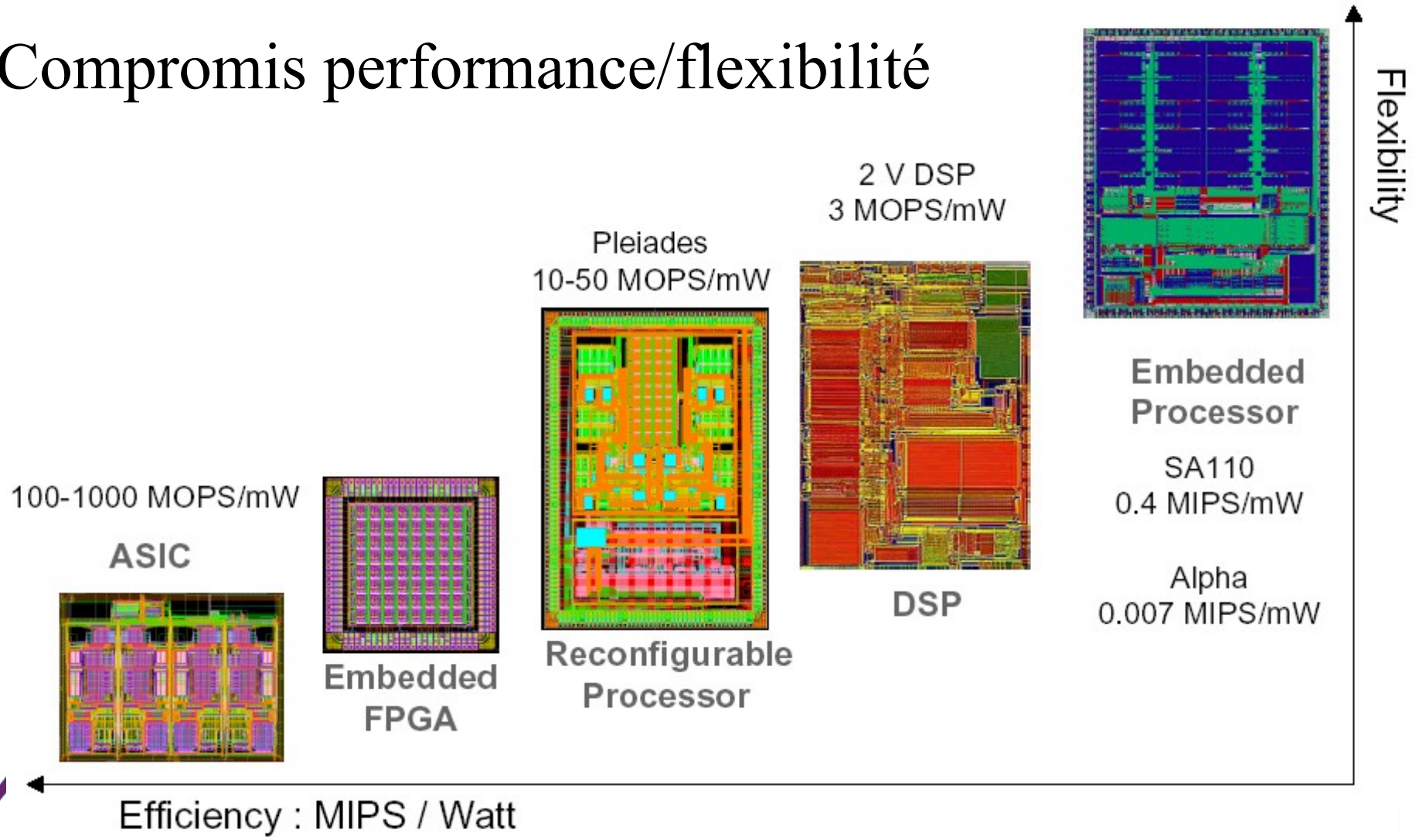


Le flot de conception CoDesign : Présentation générale



Le flot de conception CoDesign : Présentation générale

Compromis performance/flexibilité



Le flot de conception CoDesign : Les différentes étapes

Spécification

- ✓ **Evaluation d'un cahier des charges**
  - ❑Aspects techniques
    - Evaluation de la complexité (nbre de portes, nbre E/S, …)
    - Performances électriques recherchées (surface, consommation, alimentation…)
    - Types de fonction à développer (numérique, analogique, RF)
    - Encapsulation : type de boîtier
    - …..
  - ❑Aspects économiques
    - Délais de conception et de fabrication
    - Marché => Quantité de pièces prévues
    - Budget
    - ….
- ✓ **Choix de la cible matérielle**
  - ❑Une technologie
  - ❑Un type de circuit (FPGA, ASIC, DSP, Processeurs…)
  - ❑Un flot de développement (outils CAO)

Le flot de conception CoDesign : Les différentes étapes

Partitionnement

Objectif

L'objectif du partitionnement est de **regrouper les variables et les comportements fortement dépendants**, puis de décider pour chaque regroupement d'une **réalisation logicielle ou matérielle**.

Problèmatique

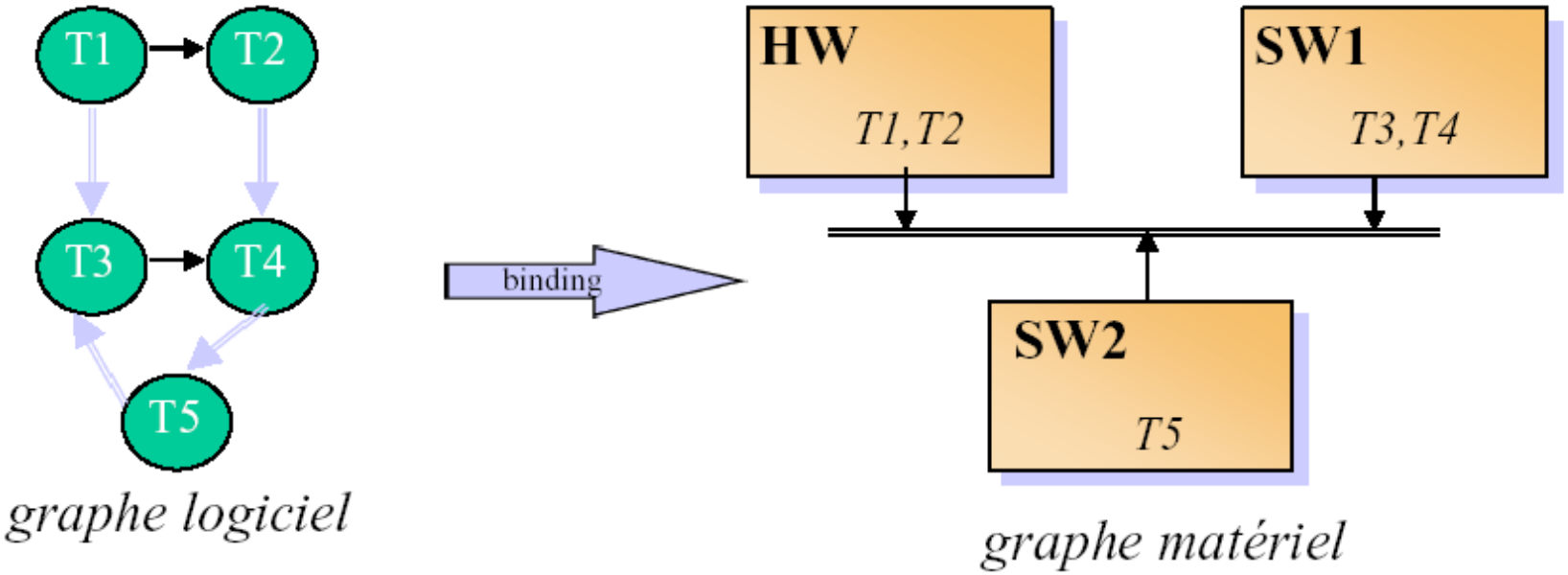
Les **performances réelles** de la réalisation ne peuvent être connues **qu'après l'étape de co-simulation**.

Approche

Afin de permettre le partitionnement, des **estimateurs de performance** rapides et précis doivent prédire les **performances** et le **coût** d'une réalisation architecturale.

Le flot de conception CoDesign : Les différentes étapes

Partitionnement



- Problèmes principaux :
  - Fonction coût
  - Granularité, parallélisme potentiel
  - Estimation de la communication

Le flot de conception CoDesign : Les différentes étapes

Partitionnement

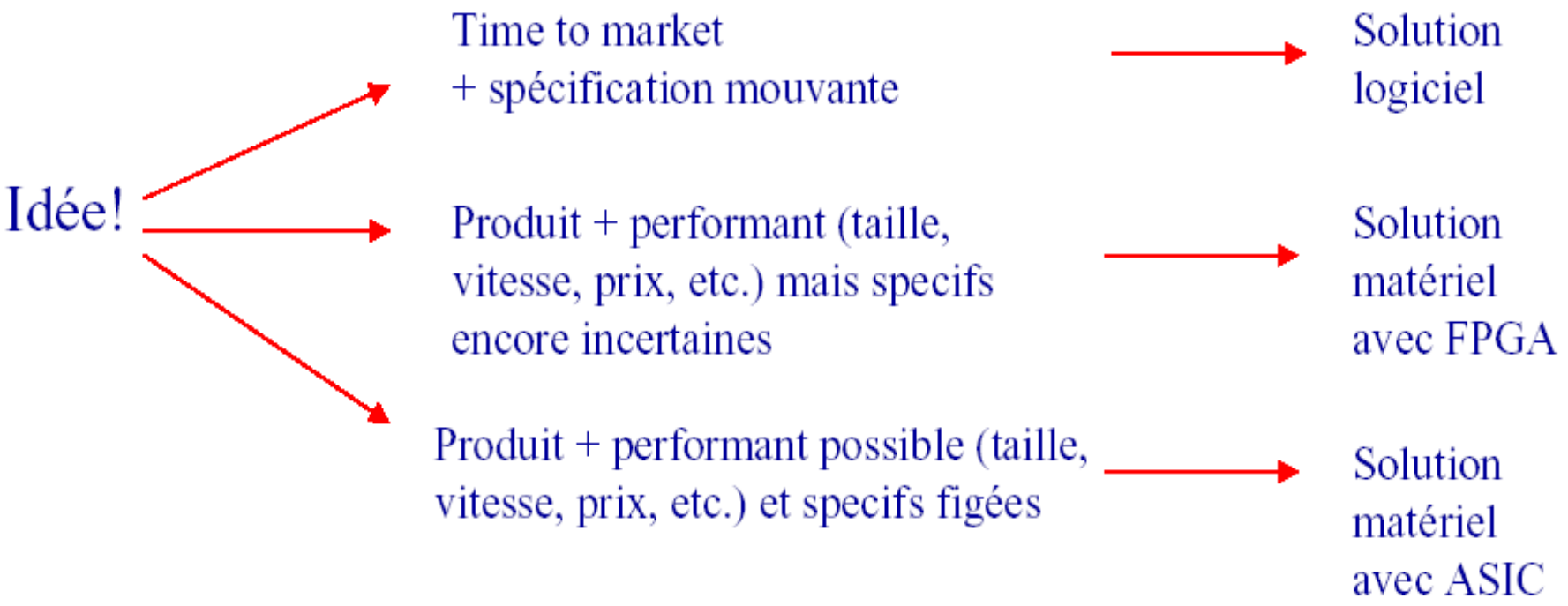
Comparaison de quelques techniques automatiques

	Modèle/Granularité	Architecture	Objectif	Technique	Contrôle	Nombre solutions
COSYMA [Ernts 93]	C étendu/Instruction	Figée: 1 proc. 1 Asic	Temps	Recuit Simulé, SW -> HW	nonpréemptif compile time	1
Vulcan [De Micheli 94]	C/Instruction	Figée: 1 proc. 1 Asic	Temps	HW -> SW Glouton	nonpréemptif	1
GCLP [Kalavade 94]	DFG/Tâche	Figée: 1 proc. 1 Asic	Temps/Surface	List scheduling modifié	nonpréemptif compile time	1
[Vahid 97]	C/Fonction	Figée: 1 proc. 1 Asic	Temps/Surface/ IO	Min Cut	nonpréemptif compile time	Exploration
[Teich 97]	C/Fonction	SOC	Temps	Algorithme génétique	?	Exploration
COSYN [Dave 97]	Graphe de tâches	Multiprocesseur	Temps/Consommation	Clustering	préemptif/non-préemtif	1
[Karkowski 98]	C/boucles	multi-ASIP	Temps/nb proc.	Enumération	séquentiel	Exploration
[Kuchcinski 99]]	DFG/Tâche	SOC	Temps/Surface	Prog. par contraintes	compile time	Exploration
[Li 00]	C/boucles	1 proc. 1 FPGA	Temps	Clustering	compile time	1
SynDEx [Sorel]	DFG/Tâche	multiprocesseur	Temps	List Scheduling modifé	nonpréemptif compile time	1
[Oh 99]	Graphe de Tâches	multiprocesseur	Temps/Surface	Ordonnançabilité	préemptif	1
[Wolf 95]	Graphe de Tâches	multiprocesseur	Temps/Surface	Ordonnançabilité	préemptif	1

Le flot de conception CoDesign : les différentes étapes

Partitionnement

La répartition logiciel/matériel peut varier durant la vie d'un produit.



Le flot de conception CoDesign : Les différentes étapes

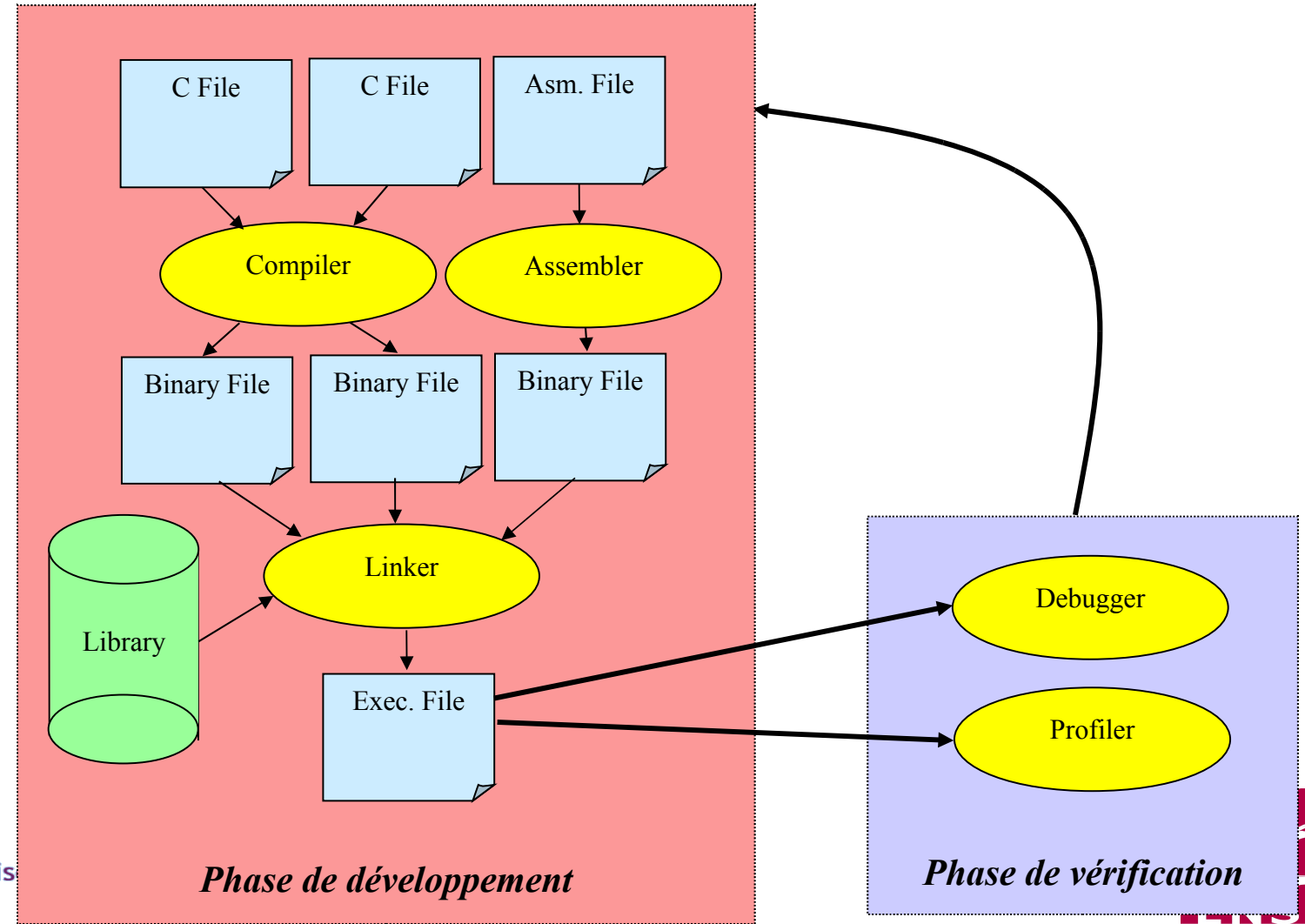
Synthèse

Cette étape regroupe les synthèses de la partie logicielle, de la partie matérielle et des interfaces et protocoles de communication.

- *Synthèse logicielle* : correspond à la conversion d'une description décrite dans un **langage fonctionnel** en un code **exécutable par un processeur**.
- *Synthèse matérielle* : correspond à la conversion d'une description décrite dans un **langage fonctionnel** en un ensemble d'équations différentielles décrivant la **structure d'un layout**.
- *Synthèse des communications* : étape essentielle, elle doit **garantir les transferts de données entre les différents blocs matériels et logiciels**. Les **protocoles** et les **modes de communication** sont définis durant cette étape.

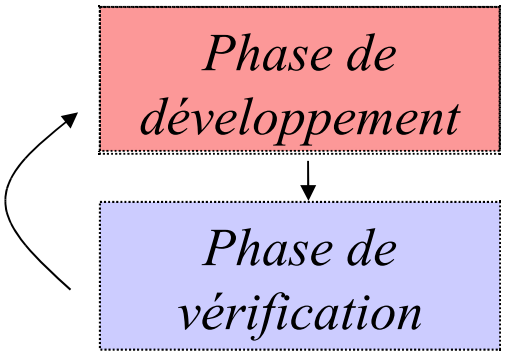
Le flot de conception CoDesign : Les différentes étapes

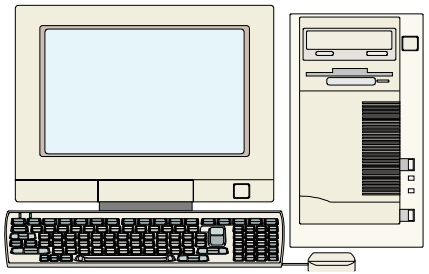
Synthèse logicielle



Le flot de conception CoDesign : Les différentes étapes

Synthèse logicielle

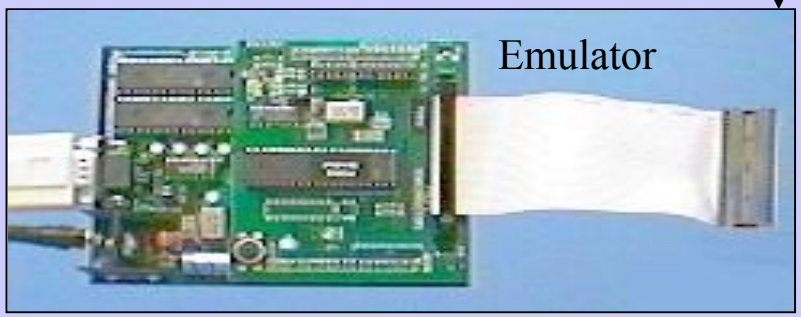




Development processor

Phase de développement

Debugger



Programmer

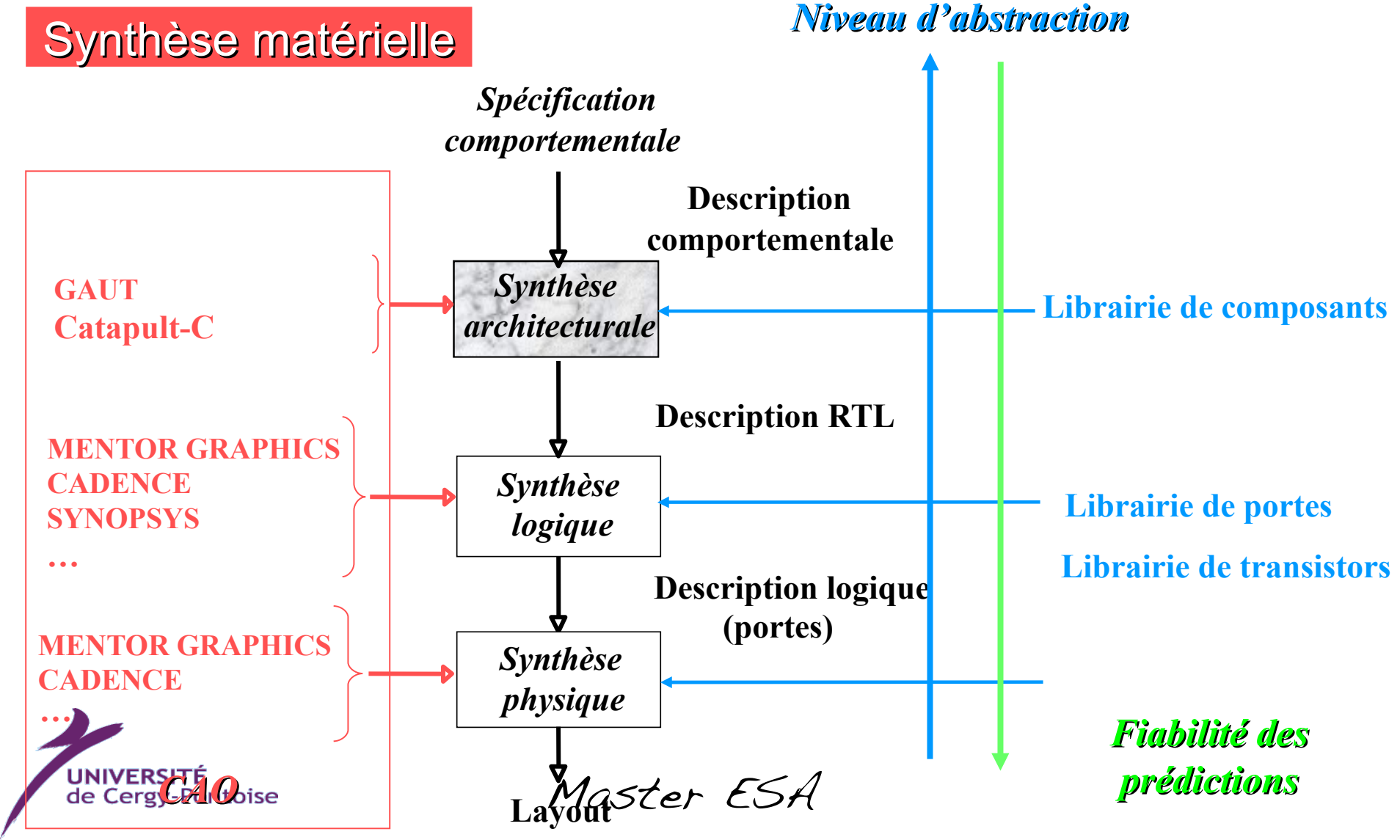
Outils de mesure

Maste

Phase de vérification

UNIVERSITÉ de Cergy-Pontoise

Le flot de conception CoDesign : Les différentes étapes

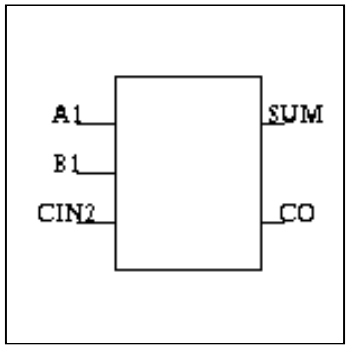


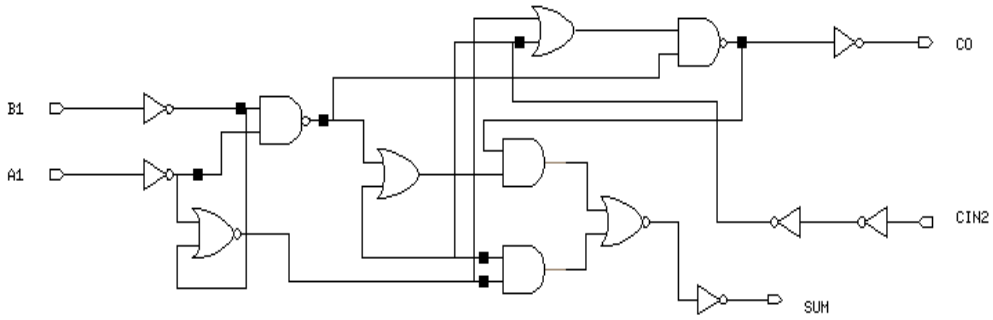
Le flot de conception CoDesign : Les différentes étapes

Synthèse matérielle

SUM :=

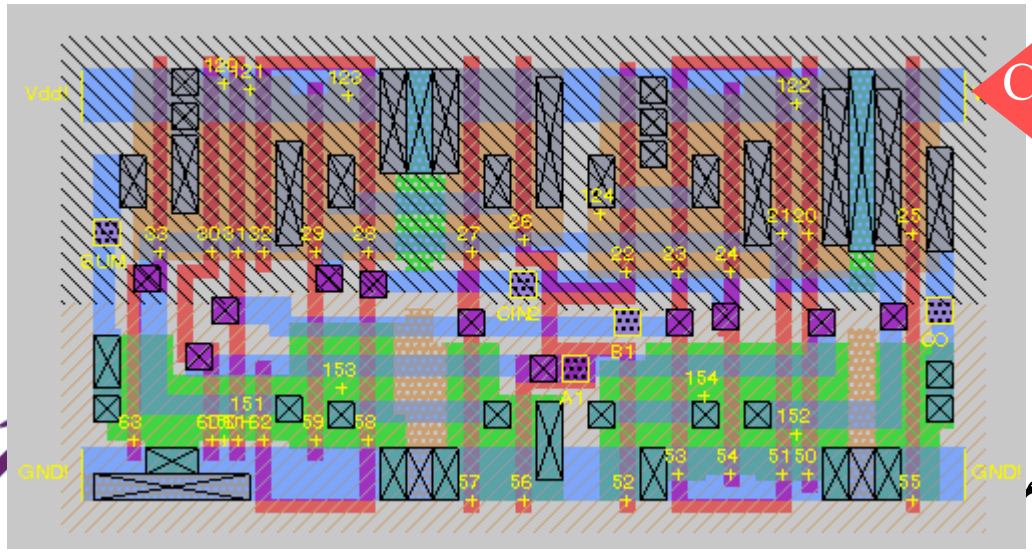
A1+B1

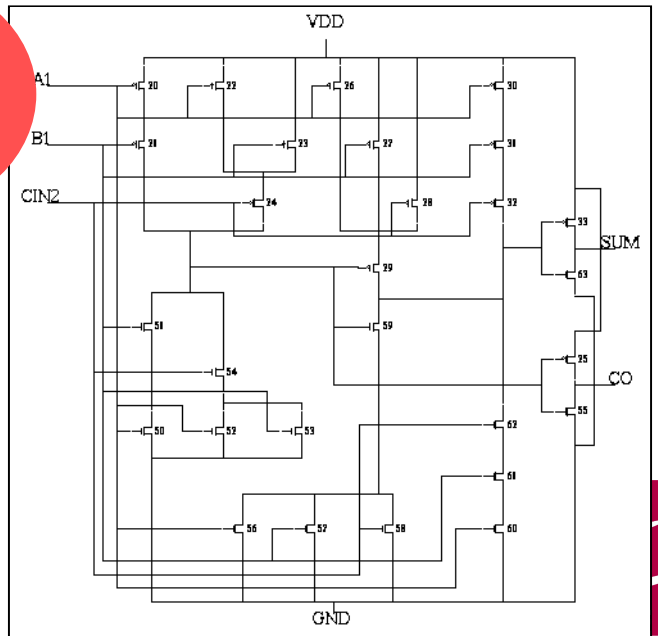




Algorithme

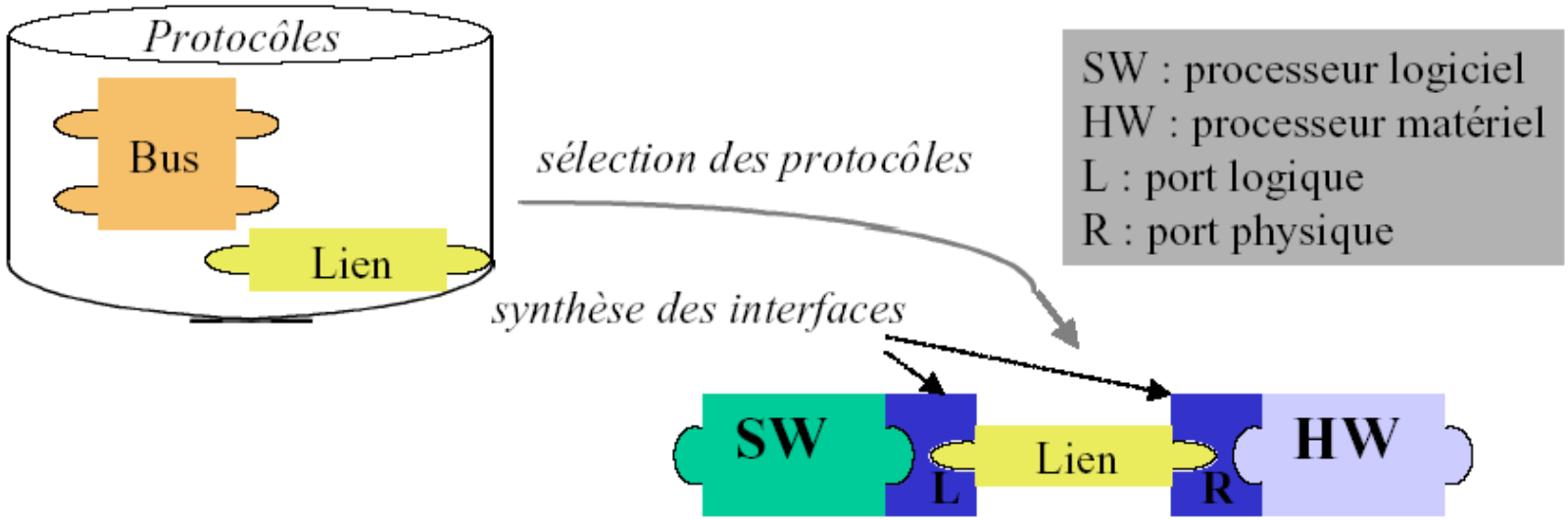
Circuit





Le flot de conception CoDesign : Les différentes étapes

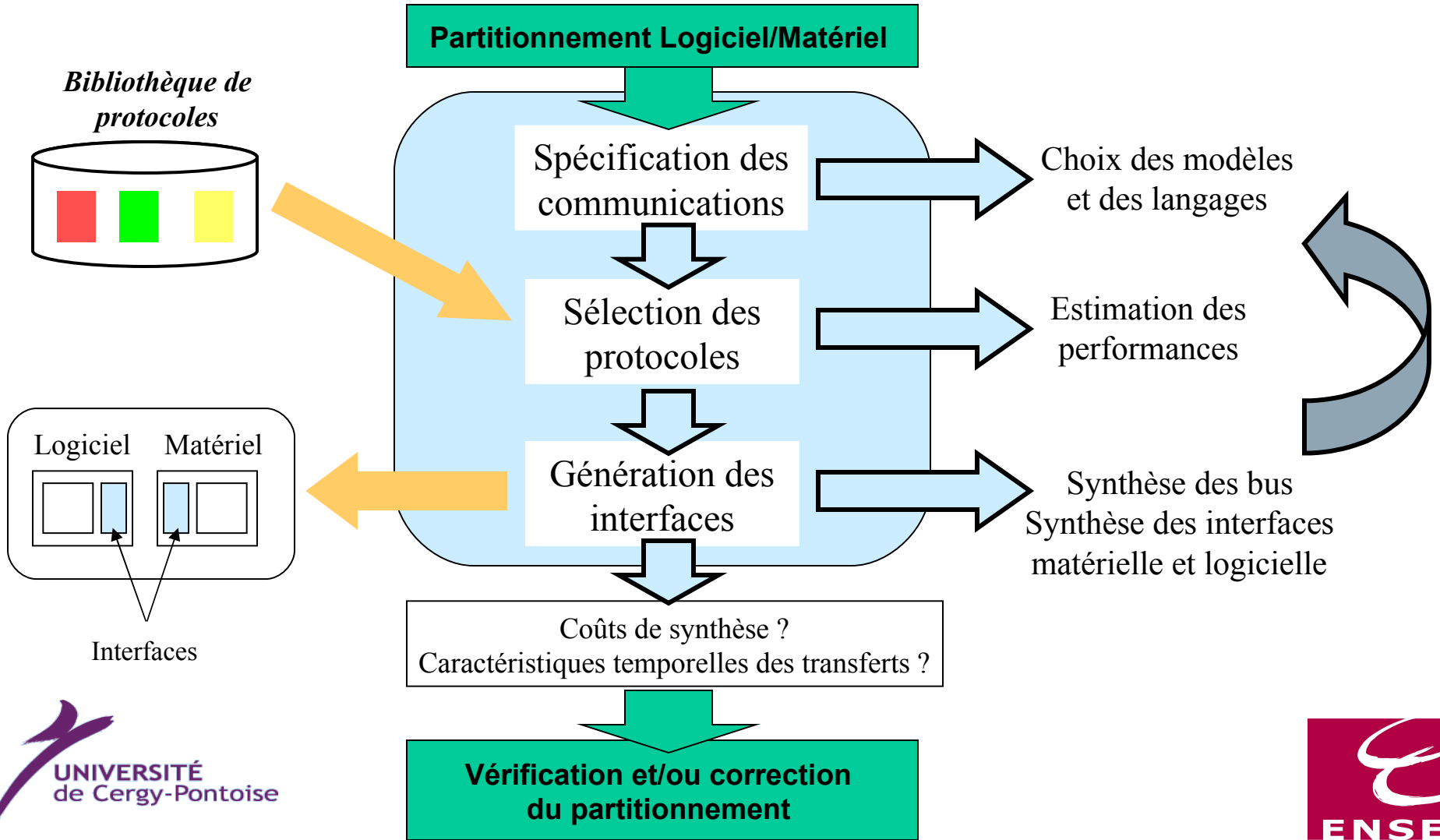
Synthèse des communications



- **Problèmes :**
  - minimiser les coûts des ports (FIFO, ...)
  - optimiser les temps de transfert

Le flot de conception CoDesign : Les différentes étapes

Synthèse des communications

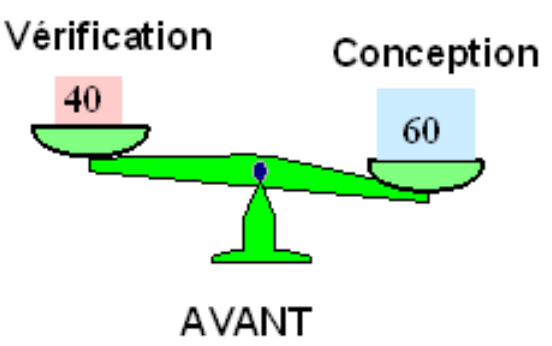


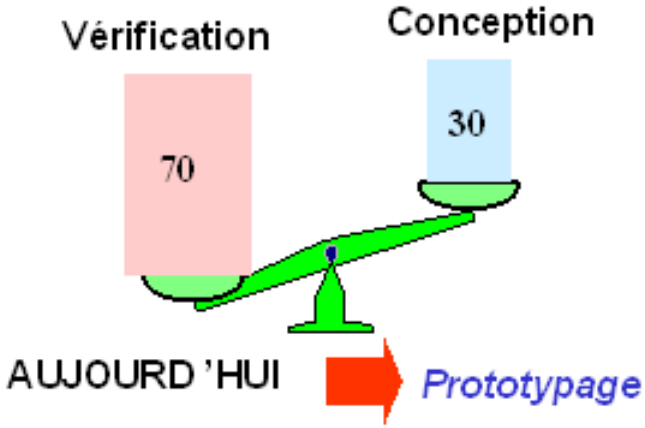
UNIVERSITÉ de Cergy-Pontoise

ENSEA

Le flot de conception CoDesign : Les différentes étapes

Co-simulation **Une nécessité !!**





PENTIUM 100MHZ
(f. prototypage)

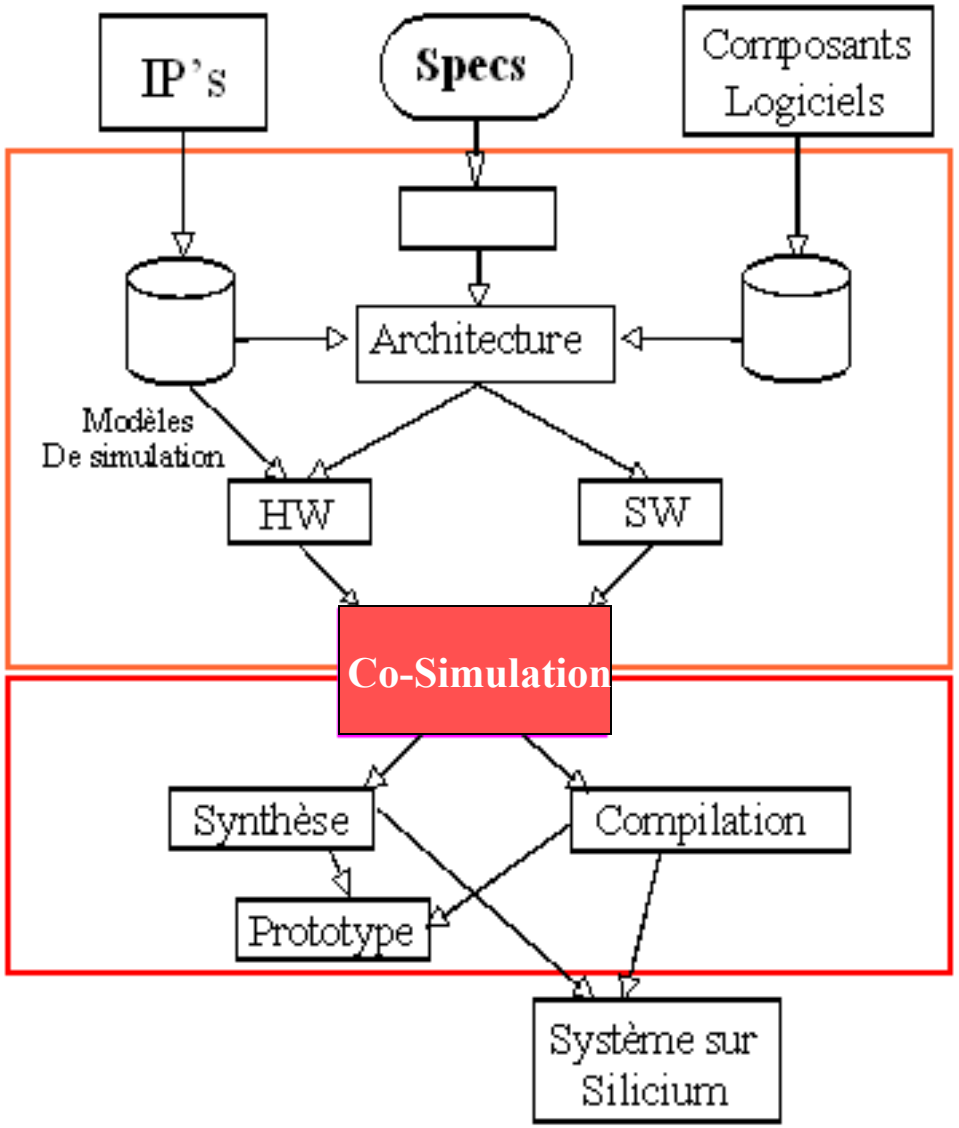
Détection de contour
sobel :
w=3x3 image 256x256

SYSTEME CDMA

Simulation	
logique	19 ans
Fonct.	20 heures
VHDL	15 mois

Prototypage
quelques semaines
0,5 s
1s

Le flot de conception CoDesign : Les différentes étapes



Co-simulation

Prototype Virtuel :

- *Haut niveau d'abstraction*
- *Modifications aisées*

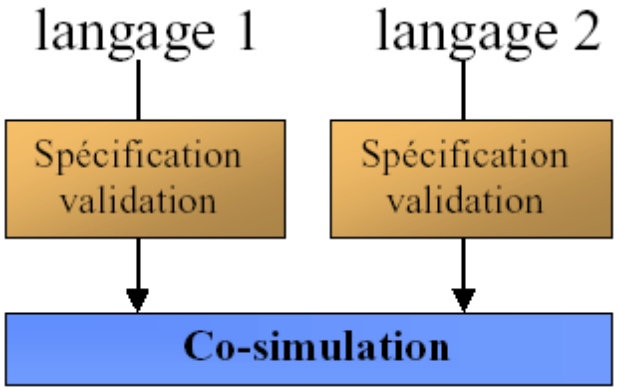
Prototype Matériel :

- *Détails d'implantation*
- *Rapidité d'exécution*

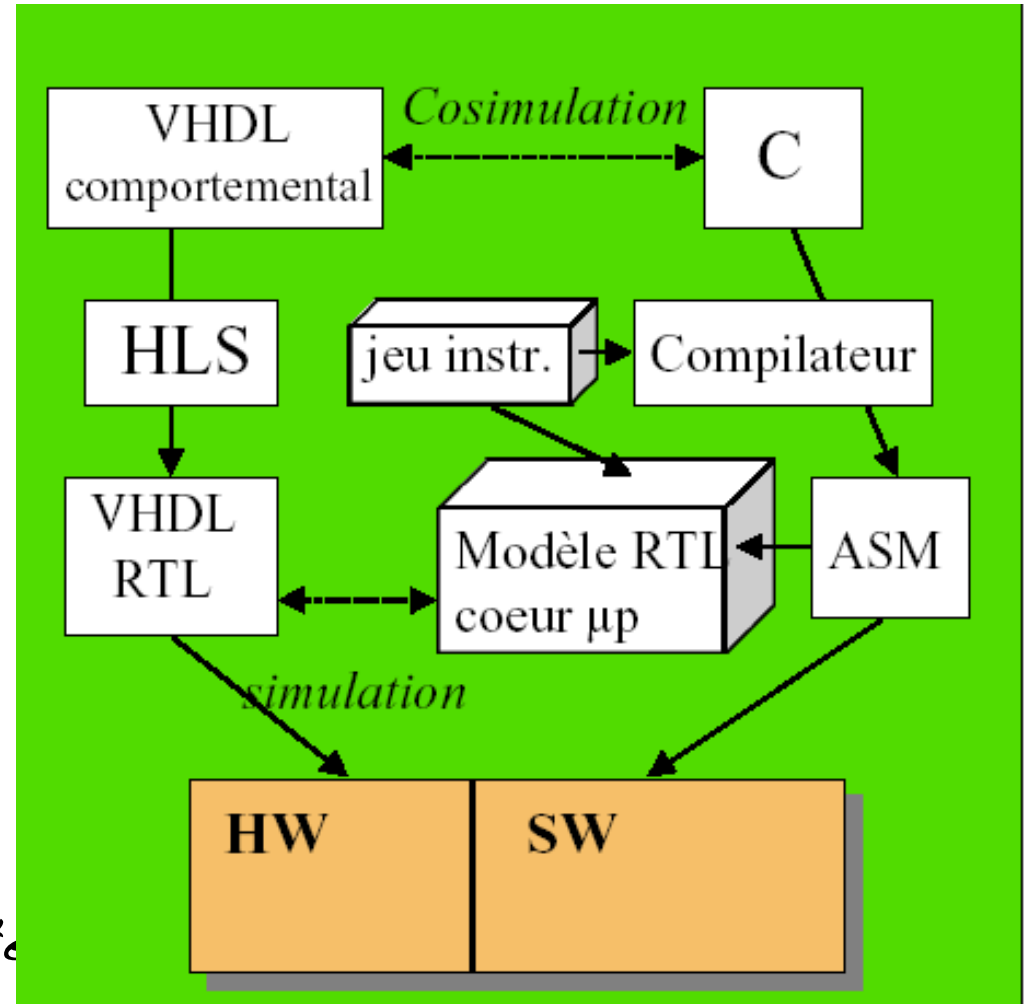
Le flot de conception CoDesign : Les différentes étapes

Co-simulation

Principe



Exemple



Définition – Co-Design Logiciel/Matériel

- Conception de systèmes macro ou micro (System on Chip) qui intègrent à la fois des processeurs généralistes capables d'exécuter des programmes (LOGICIEL) et des IP (Intellectual Property) (MATERIEL).
- Conception *conjointe* des composant logiciels et matériel
- L'unification de chemins matériels et logiciels couramment séparés.

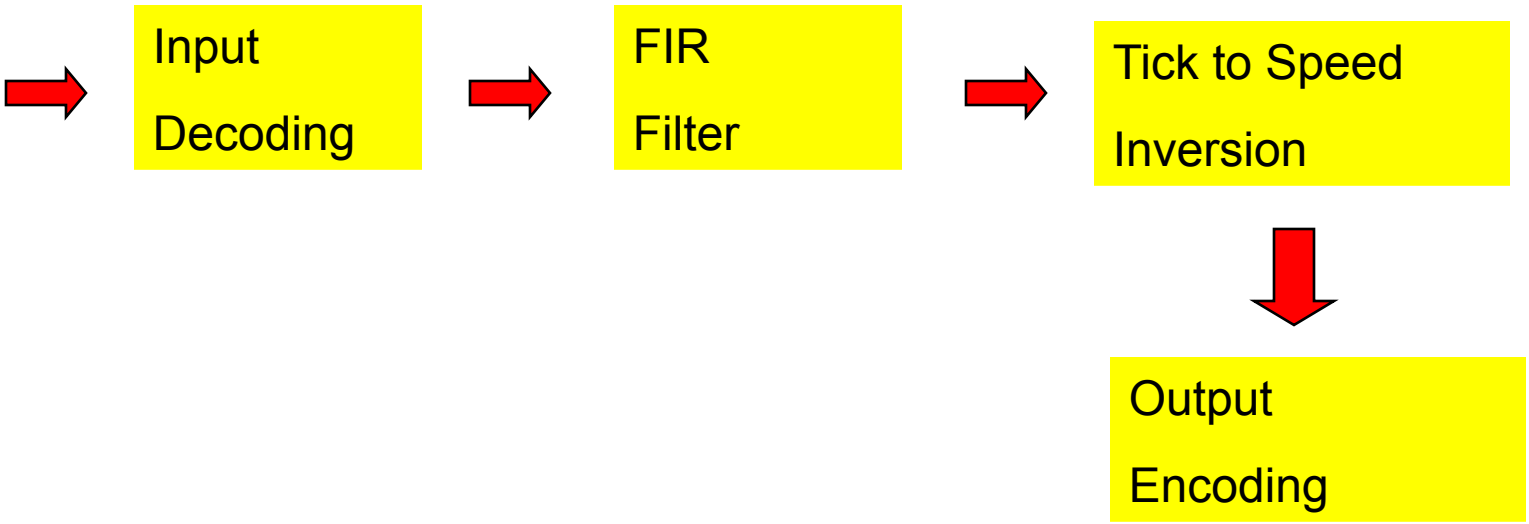
Définition – Co-Design Logiciel/Matériel

Une méthodologie de conception qui support le développement coopératif et concurrent du matériel et du logiciel (***co-spécification, co-développement, et co-vérification***) afin d'obtenir des fonctionnalités partagées et d'atteindre les performances espérées[1].

1. Gupta, R. and De Micheli, G., "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design & Test of Computers*, September 1993, pp. 29-41.

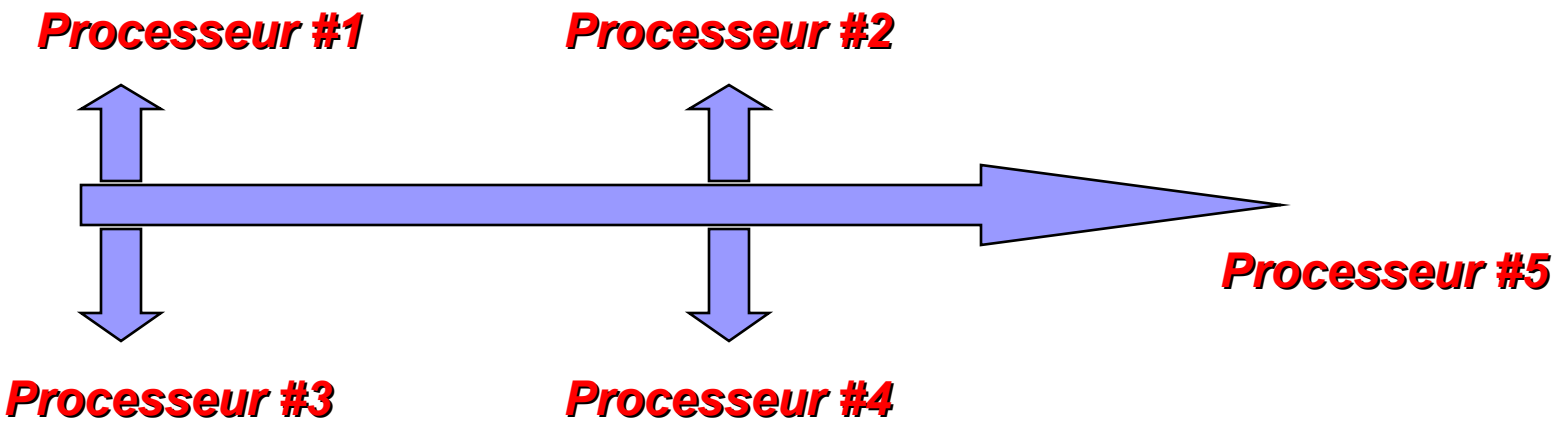
Co-design: Exemple

- Capteur optique de vitesse de roue.
- Contraintes du systèmes → Surface – 40 unités, temps – 100 cycles
- Cela peut être mis en oeuvre avec des processeurs généralistes, du matériel spécialisé ou une combinaison des 2.



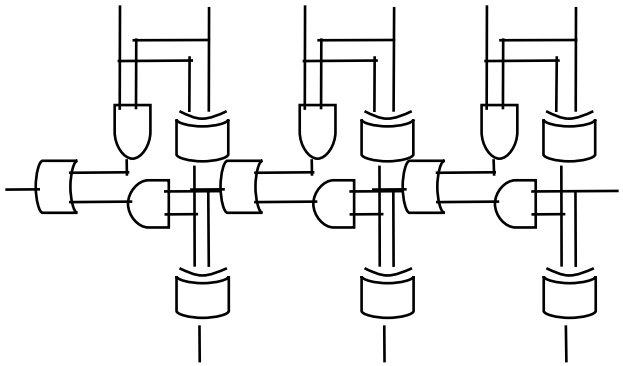
Co-design: Logiciel

- Mise en oeuvre logicielle
- Contraintes
  - *Surface – 48 unités* > 40 unités
  - *Temps – 132 cycles* > 100 cycles
- Développement : 2 mois



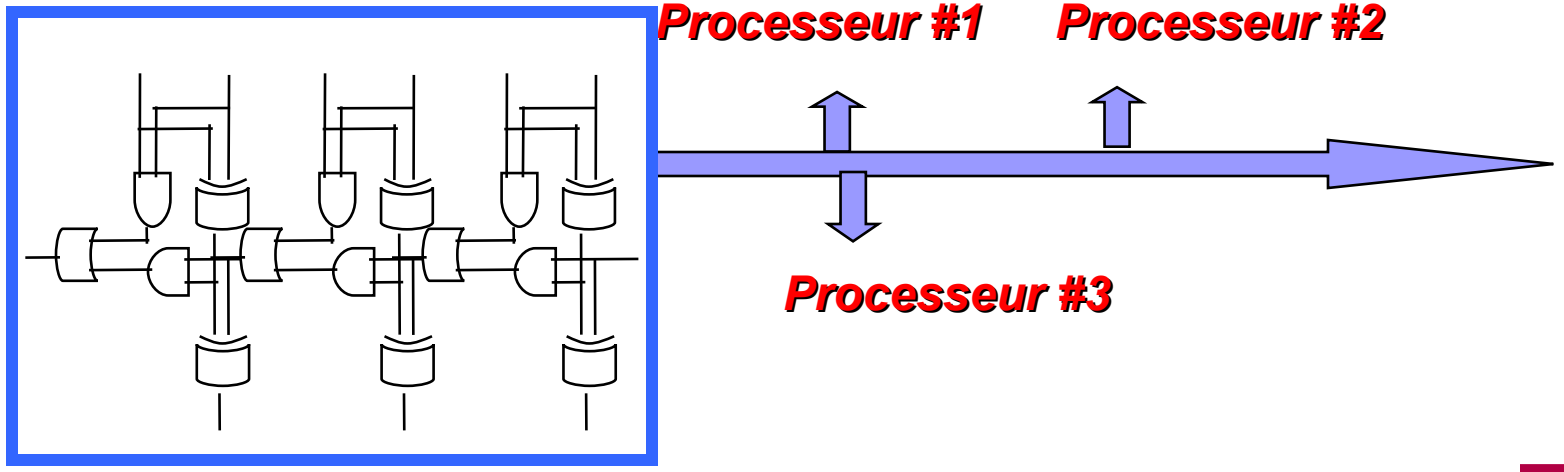
Co-design: Matériel

- Mise en oeuvre matérielle
- Contraintes
  - *Surface – 24 unités*, < 40 unités
  - *Temps – 52 cycles* < 100 cycles
- Dépasse les espérances en surface et en temps de *40%*
- Développement : 9 mois
  - Délai trop long, surtout dans un monde très compétitif



Co-design : Logiciel/Matériel

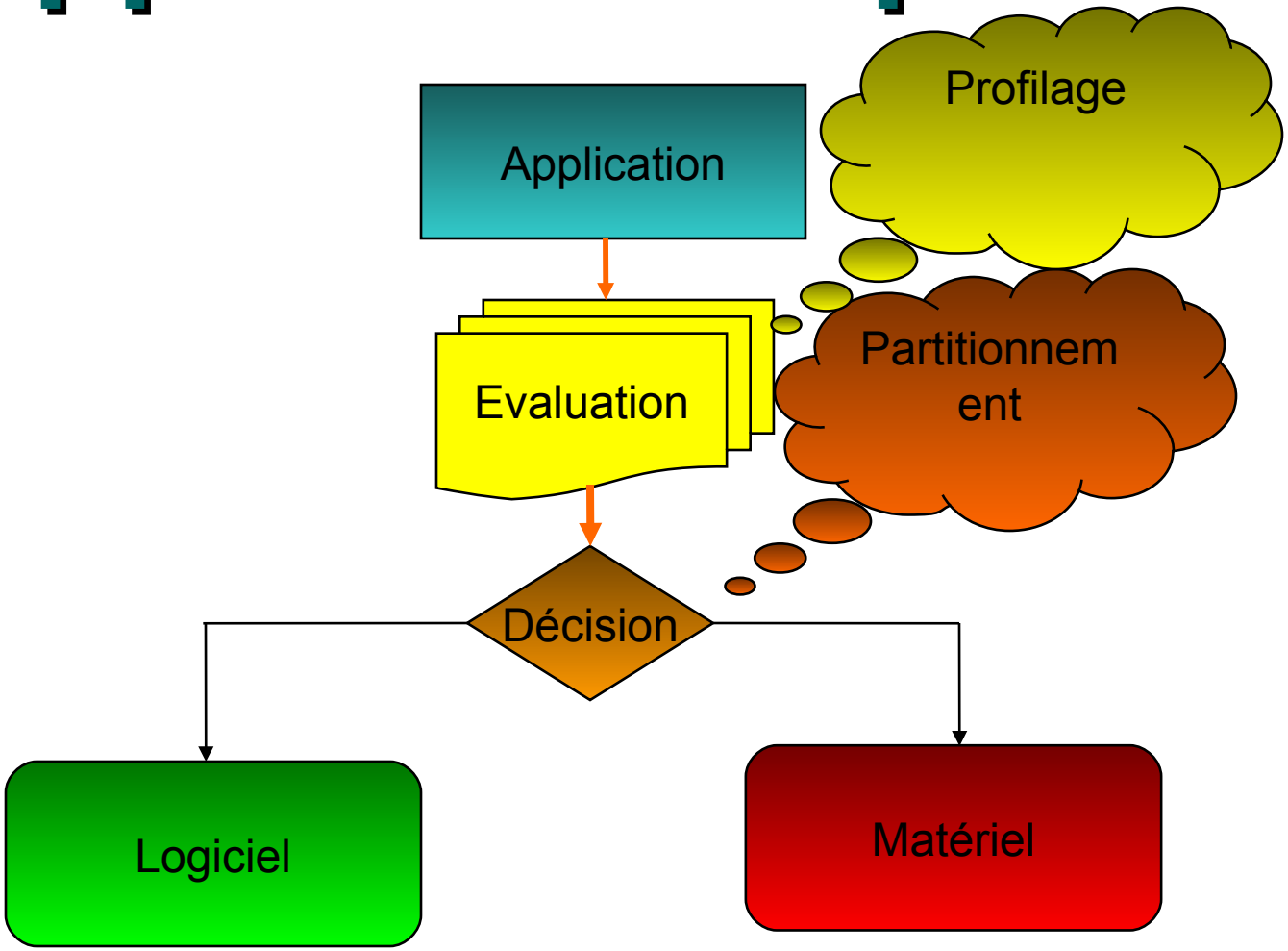
- Mise en oeuvre Logicielle et Matérielle
- System constraints
  - *Surface – 37 unités*, < 40 unités
  - *Temps – 95 cycles* < 100 cycles
- Développement : 3,5 mois
- Pas aussi efficace que la mise en oeuvre purement matérielle
- Réalise un bon compromis entre les deux extrèmes



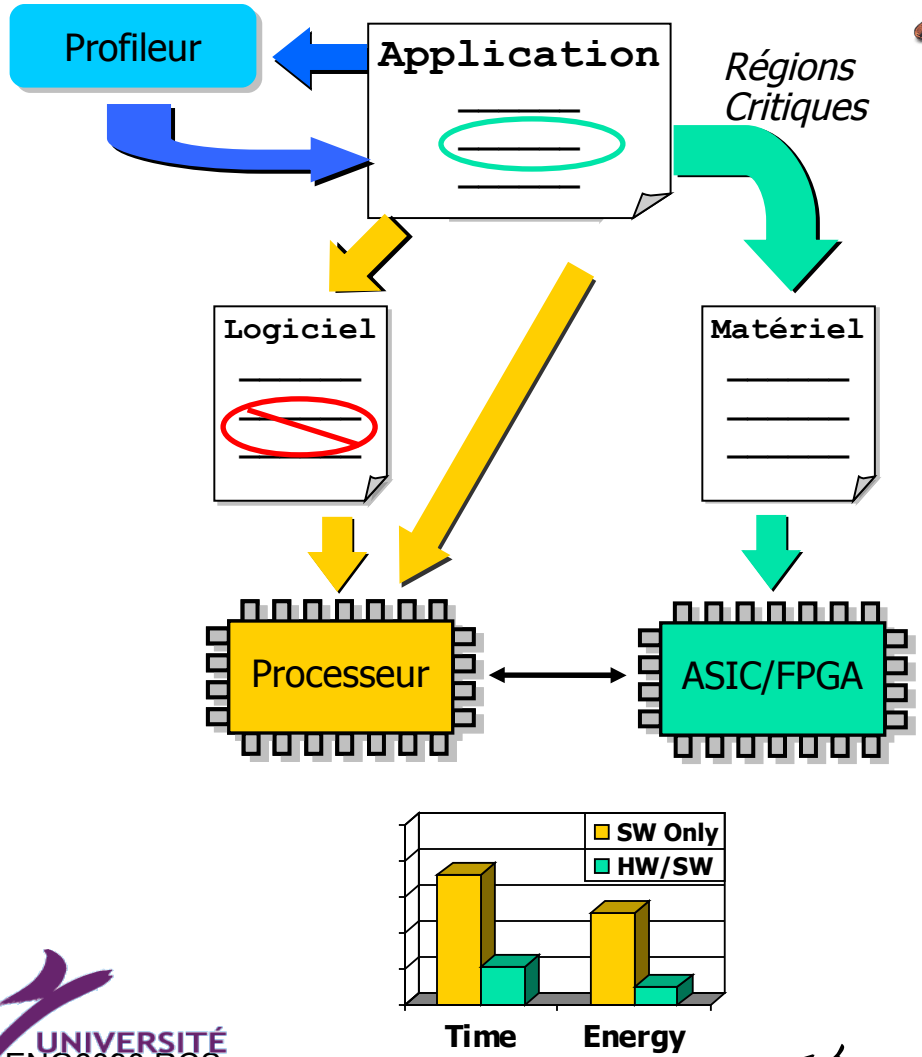
Co-Design: Recherche

- La recherche en codesign traverse plusieurs champs de compétences tels que :
  - *Spécification système et modélisation*
  - *Exploration du design*
  - *Partitionnement*
  - Ordonnancement
  - *Co-verification et Co-simulation*
  - Génération de code matériel et logiciel
  - Interfaçage matériel/logiciel
- L'objectif commun ici est de développer une méthodologie unifiée pour créer des systèmes qui contiennent à la fois du matériel et du logiciel.

Approche Simple



Profilage et Partitionnement



- **Bénéfices**
  - Accélération de 10 à 200 fois
  - Accélération possible de 800 fois
  - Beaucoup plus de potentiel que les optimisations dynamiques logicielles (internes au processeur, déroulage de boucle, pipeline logiciel,...)
  - Réduction de la consommation d'énergie de 25 à 95%

Profilage

- Le Profilage permet d'apprendre les endroits, en terme de code, où le programme passe son temps. Quelle fonction appelle quelle autre durant son exécution.
- Le profilage s'effectue via des données collectée lors de l'exécution de l'application. Cette méthode peut donc être utilisée pour analyser des programmes trop complexe pour une analyse via la lecture des sources.
- Ces informations de profil, montre les bouts de code où le programme est plus lent qu'attendu.
- Ces bouts de code sont de bons candidats à :
  - une réécriture optimisées
  - une transformation matérielle

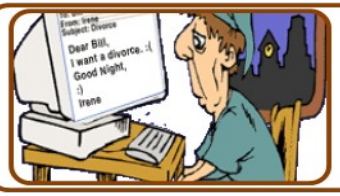
Comment réaliser un profilage ?

- **Echantillonage :** interruption périodique de l'OS ou lecture des compteurs matériels du processeur
- **Instrumentation:** insertion de code espion pour mesurer

Echantillonage / Instrumentation

	Echantillonage	Instrumentation
Overhead	Typically about 1%	High, may be 500% !
System-wide profiling	Yes, profiles all app, drivers, OS functions	Just application and instrumented DLLs
Detect unexpected events	Yes , can detect other programs using OS resources	No
Setup	None	Automatic ins. of data collection stubs required
Data collected	Counters, processor an OS state	Call graph , call times, critical path
Data granularity	Assembly level instr., with src line	Functions, sometimes statements
Detects algorithmic issues	No, Limited to processes , threads	Yes – can see algorithm, call path is expensive

Méthode d'analyse des performances d'un programme



Comprehension of program behavior

Measure the Performance of the program using Analysis tools

Identify **HotSpots** (Processes and routines that eat most execution time)

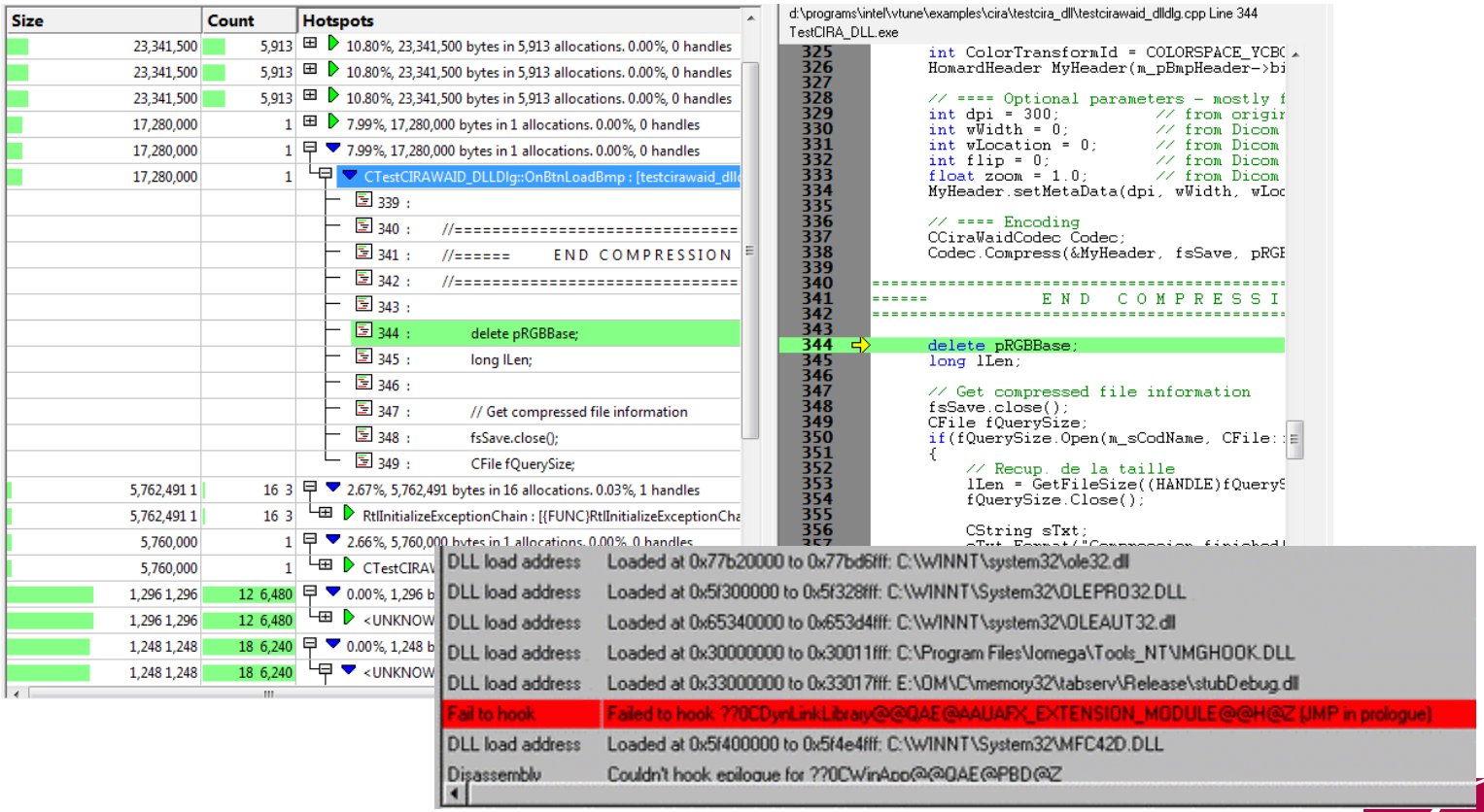
Diagnosis of the issue & identify the cause of HotSpots

Remove Bottlenecks by modifying program & refining HotSpots

Exemple d'outil de profilage

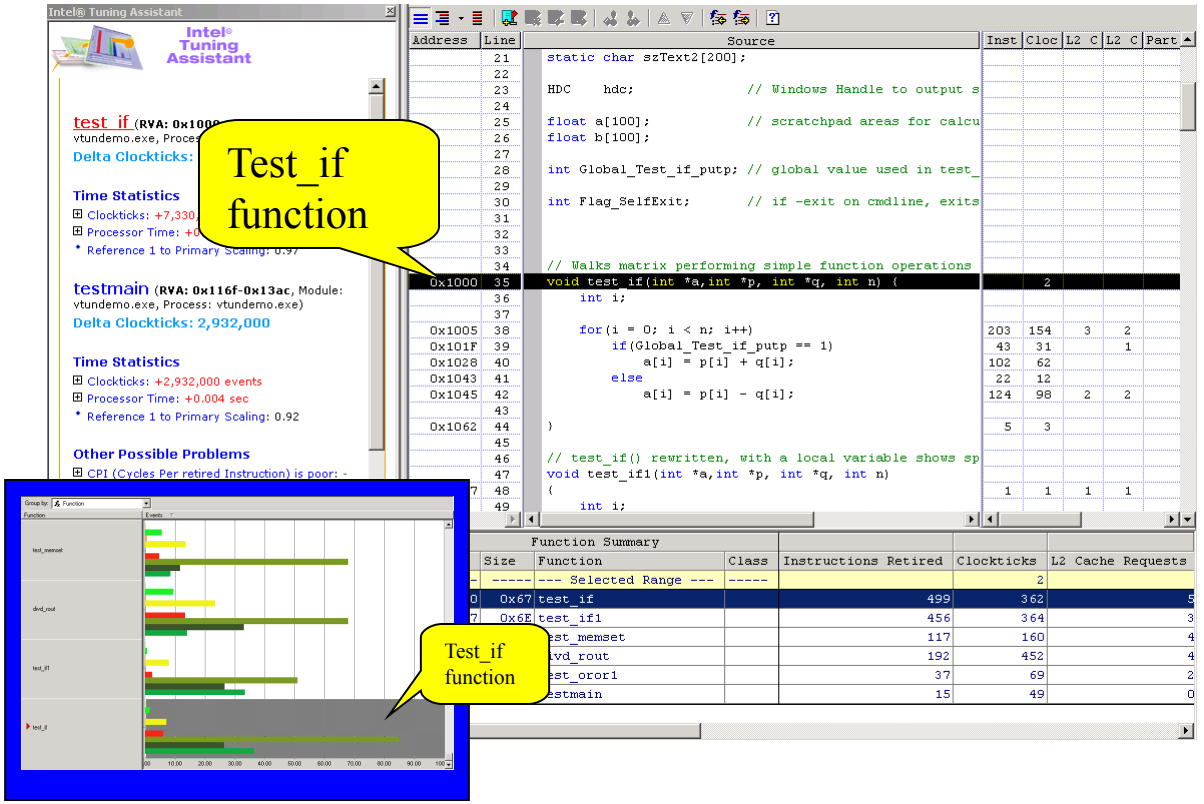
- Memory & Performance Validator

(Software Verification Ltd. - UK)



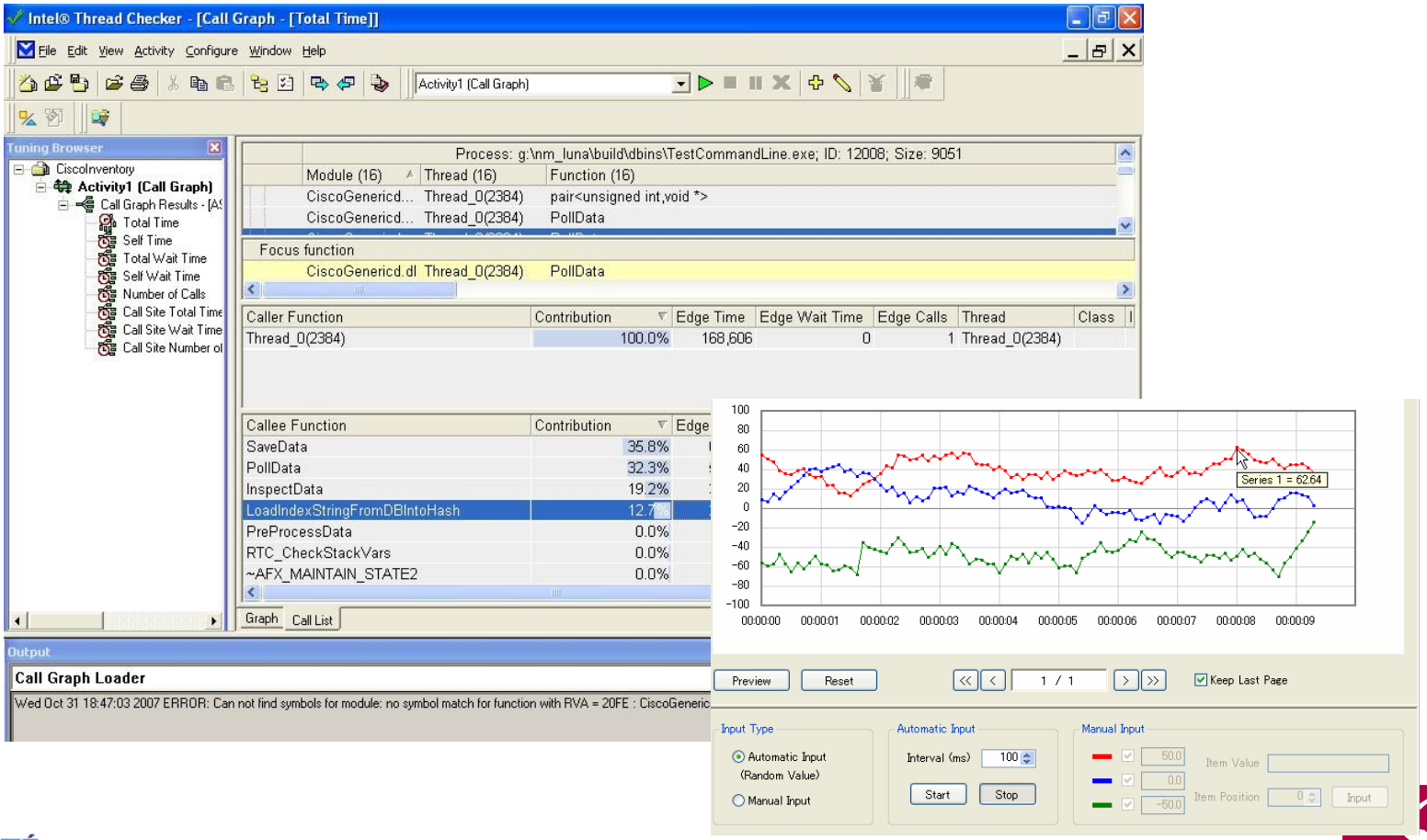
Exemple d'outil de profilage

- V-Tune - Intel



Exemple d'outil de profilage

- Visual C++ Performance Profiler (Microsoft)



Evaluation et Comparaison

	Mem & Perf Validator	V-Tune	VS Perf. Analyzer	ValGrind	Google Profiler
Platforms					
Linux Win32 X64	☐ ☒ ☐	☒ ☒ ☐	☒ ☒ ☒	☒ ☐ ☐	☒ ☐ ☐
Line level code	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Graphical o/p	**Yes**	**Yes**	**Yes**	**No**	**No**
Diagnostic	**Yes**	**Yes**	**No**	**No**	**No**
Strongness	• Interesting graphical environment • No need to instrument code	• quickly identify critical functions • Helps identify system level perf. Issues • No need to instrument code • Important diagnostic feature	• Easy to use • Visual studio	• Strong in memory profiling • all reads and writes of memory are checked • Easy to use • Freeware	• Freeware • Increase cache efficiency
Weakness	• Weak performance Profiling	• Not Stable for X64 PF • Monitors all active software on system	• Is Not a stand alone profiler • Very expensive • No memory profiling	• Weak performance profiling • No windows PF	• Old • Buggy • Inaccurate
Notes & Users Evaluation	Weak perf. profiling		Is Not a stand alone profiler	Massif in Heap Profiling	Old, buggy, Inaccurate
Price (€)	**407**	**695**	**9470**	**Freeware**	**Freeware**

Exemple de profilage via gprof

- Avec gcc, il faut tout d'abord compiler et lier le programme avec les options de profilage autorisées :
  - *gcc -o myprog.exe myprog.c utils.c –g –pg*
- Il faut ensuite exécuter le programme pour collecter les donnée du profil d'exécution
  - Le programme écrit les données collectées dans un fichier `*gmon.out*' juste avant de finir.
- Il est possible après d'utiliser gprof pour analyser les données collectées :
  - *gprof options* myprog.exe *gmon.out* > outfile
  - *gprof* créé un fichier de profil et un graphe d'exécution

Exemple de profilage via gprof

- *Options:*
  - **-e *function_name* : indique à gprof de ne pas générer d'information sur la fonction *function_name* (et ses enfants …) dans le graphe d'appel**
  - **-f *function_name*: provoque une limitation de l'analyse dans le graphe des appels à la fonction *function_name* et ses enfants**
  - **-b : gprof ne renvoie pas d'informations explicatives à propos des champs renseignés dans les tables.**

Exemple de profilage via gprof

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
```

- ***% time*** **: pourcentage du temps total d'exécution que le program a passer dans cette fonction.**
- ***cumulative seconds*****: Temps cumulatif en seconde que le processeur a passé a exécuter cette fonction ainsi que toutes les fonctions appelées dans cette fonction.**
- ***self seconds*****: Temps en secondes utilisé pour cette seule fonction.**
- ***calls*****: Nombre de fois total où cette fonction a été appelée.**
- ***self ms/call*****: Temps moyen en milliseconde pris par chaque appel de la fonction.**
- ***total ms/call*****: Temps moyen en milliseconde pris par chaque appel de la fonction et de ses descendants.**
- ***name*****: Nom de la fonction.**

Faiblesse de cette première approche

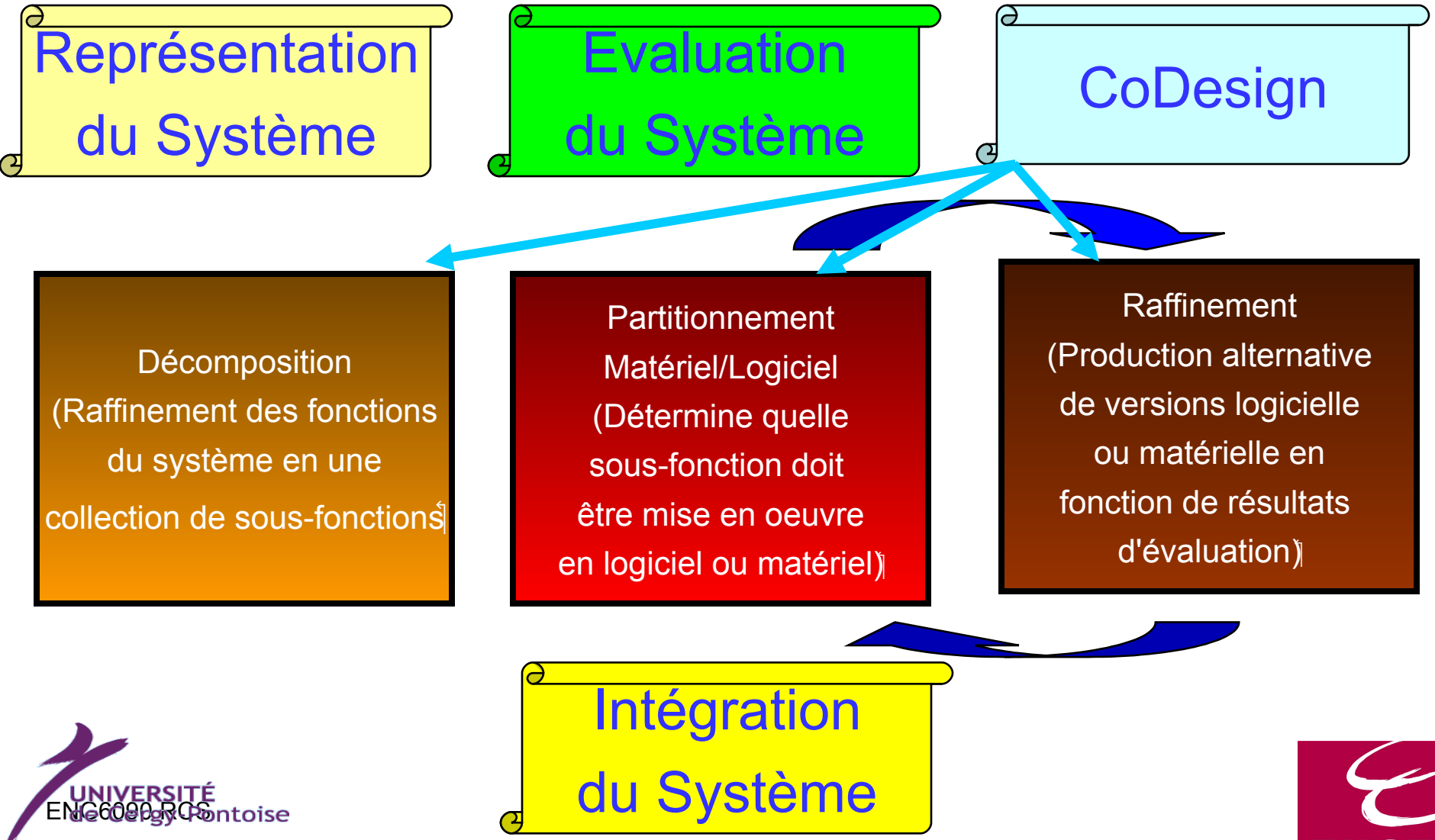
I. Certaines fonctions ne sont pas triviales à réaliser en matériel.

II. Les décisions prises trop tôt dans le flot risque de ne pas être optimales

III. Aucune considération pour la communication et l'interfaçage.

IV. Si l'application change alors il faut ré-exécuter un profilage et ensuite un partitionnement.

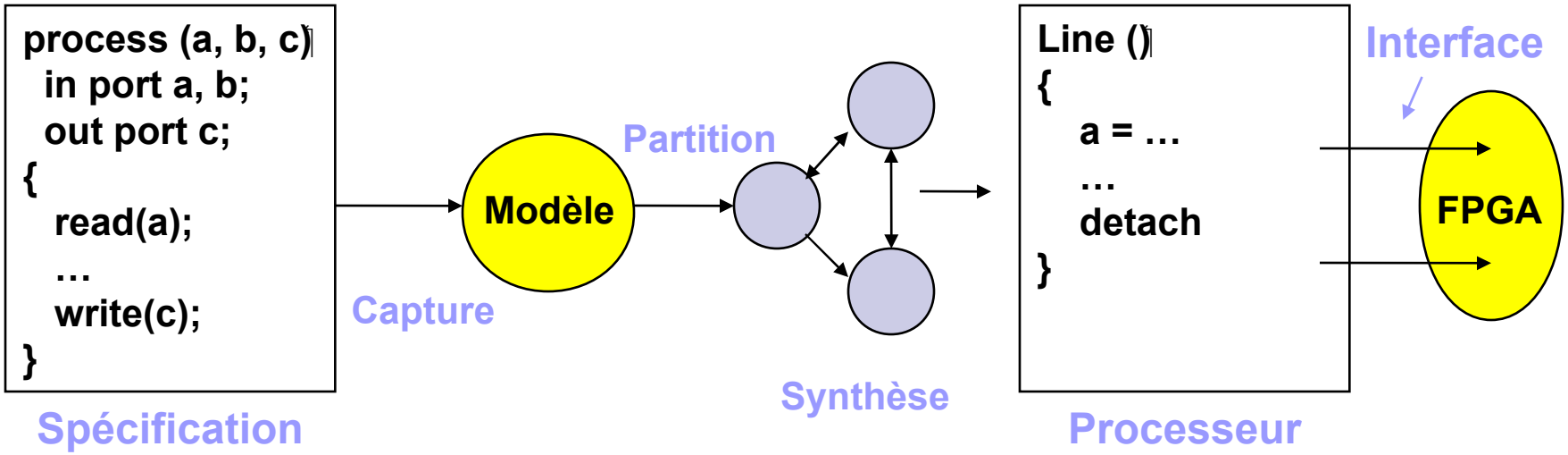
Codesign: Un atelier



Partitionnement et Ordonnancement

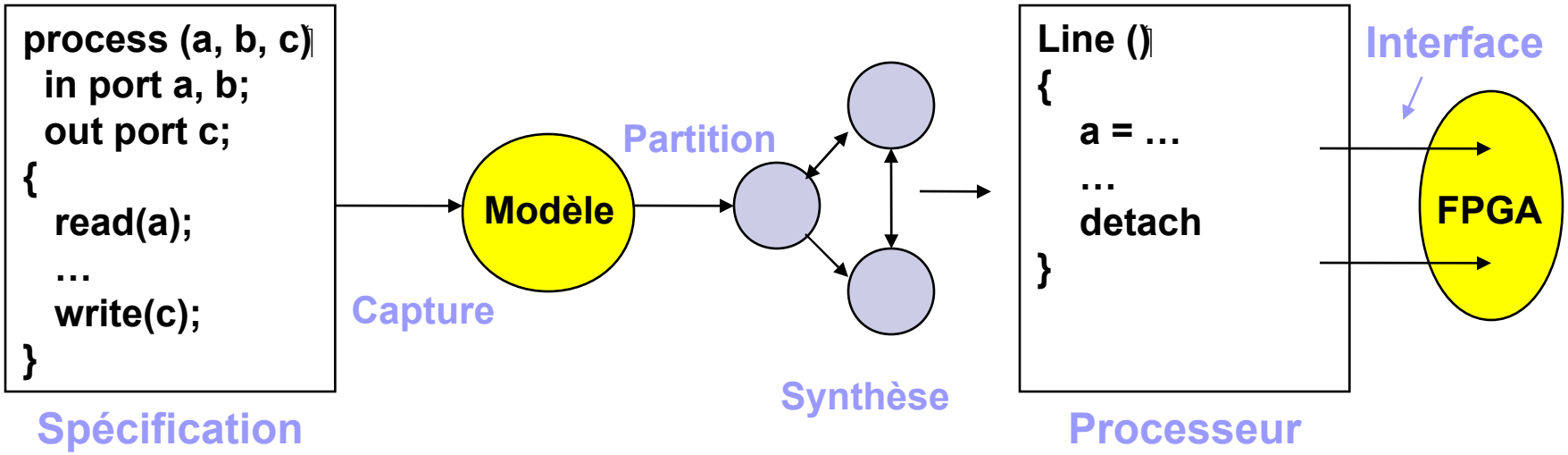
- Le ***partitionnement et l'ordonnancement de tache*** est impératif dans beaucoup d'applications, en codesing de système, pour les multi-processeur et les systèmes reconfigurables.
- Les tache identifiées de la description initiale de l'application doivent être mise en oeuvre :
  - Au bon ***endroit*** (***partitionnement***)
  - Au bon ***moment*** (***ordonnanceur***)
- Ces problèmes bien connus, le partitionnement et l'ordonnancement, ont été identifiés comme des problèmes ***NP-Complets***.
- Les techniques d'optimisations basées sur des ***heuristiques*** sont généralement employées pour explorer l'espace des possibilités où des solutions quasi-optimales peuvent être trouvées.

Partitionnement



- Les mécanismes à optimiser lors d'un partitionnement :
  1) Minimiser les communication à travers un bus
  2) Extraire le maximum de parallélisme -> Faire exécuter simultanément le matériel (FPGA/ASIC) et le logiciel (Processeur)
  3) Extraire le maximum de performances du processeur

Partitionnement



- Les mécanismes à optimiser lors d'un partitionnement :
  1) Minimiser les communication à travers un bus
  2) Extraire le maximum de parallélisme -> Faire exécuter simultanément le matériel (FPGA/ASIC) et le logiciel (Processeur)
  3) Extraire le maximum de performances du processeur

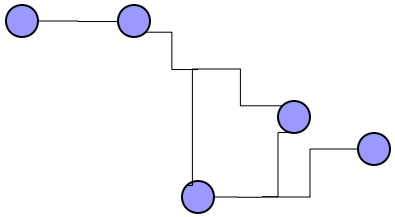
Terminologie: Hypergraphes

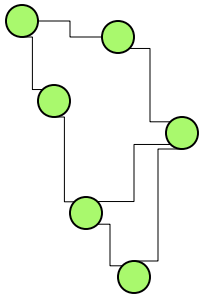
un hypergraphe H = <**V**, **E**^{h}>

V est un ensemble de sommets
h $\in$ **E**^{h} est un sous-ensemble de sommets, 2^V

un graph G = <**V**, **E**>

V est un ensemble de sommets
e $\in$ **E** est un couple de sommets (u,v)





- **Une netlist est un hypergraphe**
- **Un hypergraphe peut être approximé par des graphes en mettant à plat chaque hyperarc dans une clique d'arcs**

Problème du Bi-partitionnement

- **Soit un graphe/hypergraphe G**
- **Trouver une partition *P* de V**

  V_1, V_2 s.t $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ dans laquelle autant de sommets sont dans V_1 et V_2.

- **Minimiser le nombre d'arc qui coupent la frontière**

  $\min c(P) = \sum_{\text{all } h} w(h)$ if $\exists$ ($u \in V_1$ and $v \in V_2$)

  où u et v sont connectés par un arc h

Approches Bipartitionnement

- **min-cut / max-flow (Ford-Fulkerson 1962)**
  - maximum flow through graph = minimum cut
  - useful for establishing unconstrained bound
- **Kernighan-Lin (1970)**
  - operates on graphs
  - swap all nodes once, in pairs that yield max. gain
  - choose greatest gain over pass
  - repeat until no improvement
  - $O(n^2 \log n)$
- **recuit simulé**
  - select some random moves based on “temperature”
  - design hopefully “cools” into optimal solution
  - computationally intensive
- **Fiduccia-Mattheyses (1982)**
  - operates on hypergraphs
  - O(p), linear time!

Fiduccia-Mattheyses

- **génération d'une partition initiale**
- **calcul du gain g(c) en déplaçant chaque sommet, le gain est la différence entre le coût extérieur et le coût intérieur**

tant *qu'il y a gain* faire

{

déverrouiller;

tant que max g(c) > 0 | c ∉ verrouillé

{

sélectionner sommet with max g(c) | c ∉ verrouillé;

déplacer c à travers la frontière;

c → verrouillé;

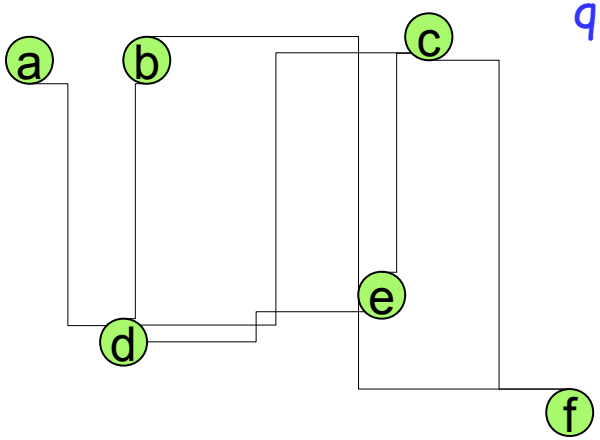
mise à jour de g(c) pour tous les voisins de c;

}

}

une itération O(p)

Exemple

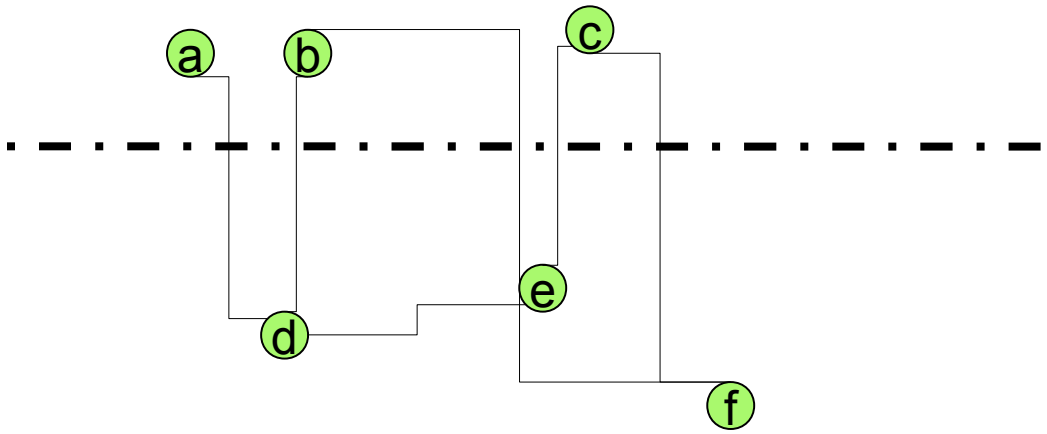


but: créer une partition du graphe en deux parties disjointes qui minimise le nombre d'arc qui coupe la frontière entre les deux parties.

- tous les arcs ont le même poids
- Critère :

$|V_1| - 1 \geq |V_2| \geq |V_1| + 1$

Exemple

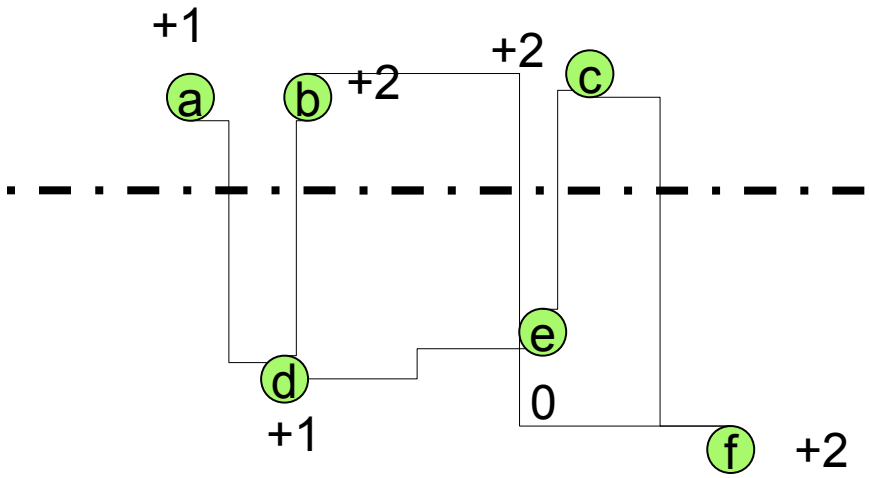


nombre de coupures = 5

Etape 1.

Partition aléatoire avec un nombre de sommets identiques dans chaque partie

Exemple

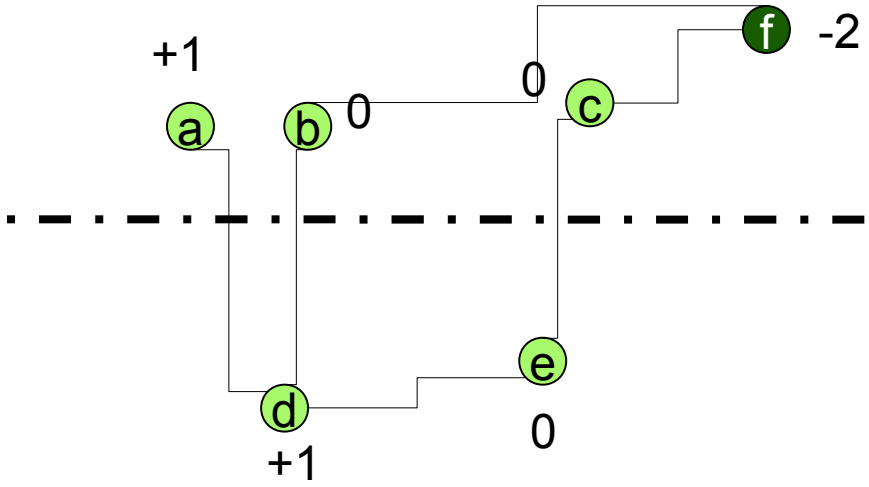


Etape 2

Calcul des gains initiaux po
chaque sommet.

Gain = 8
nombre de coupures = 5

Exemple



Etape 3

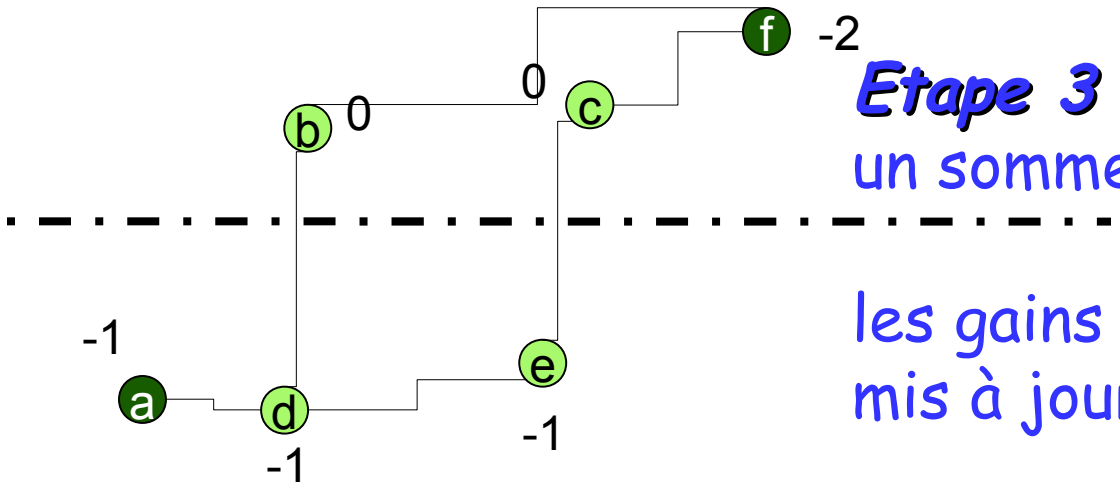
sélection d'un sommet

Les gains critiques sont mis à jour

le sommet est verrouillé pour empécher un futur déplaceme

Gain = 0
nombre de coupure = 3

Exemple



Etape 3

un sommet est sélectionné

les gains critiques sont mis à jour

le sommet est verrouillé pour empêcher de futurs déplaceme

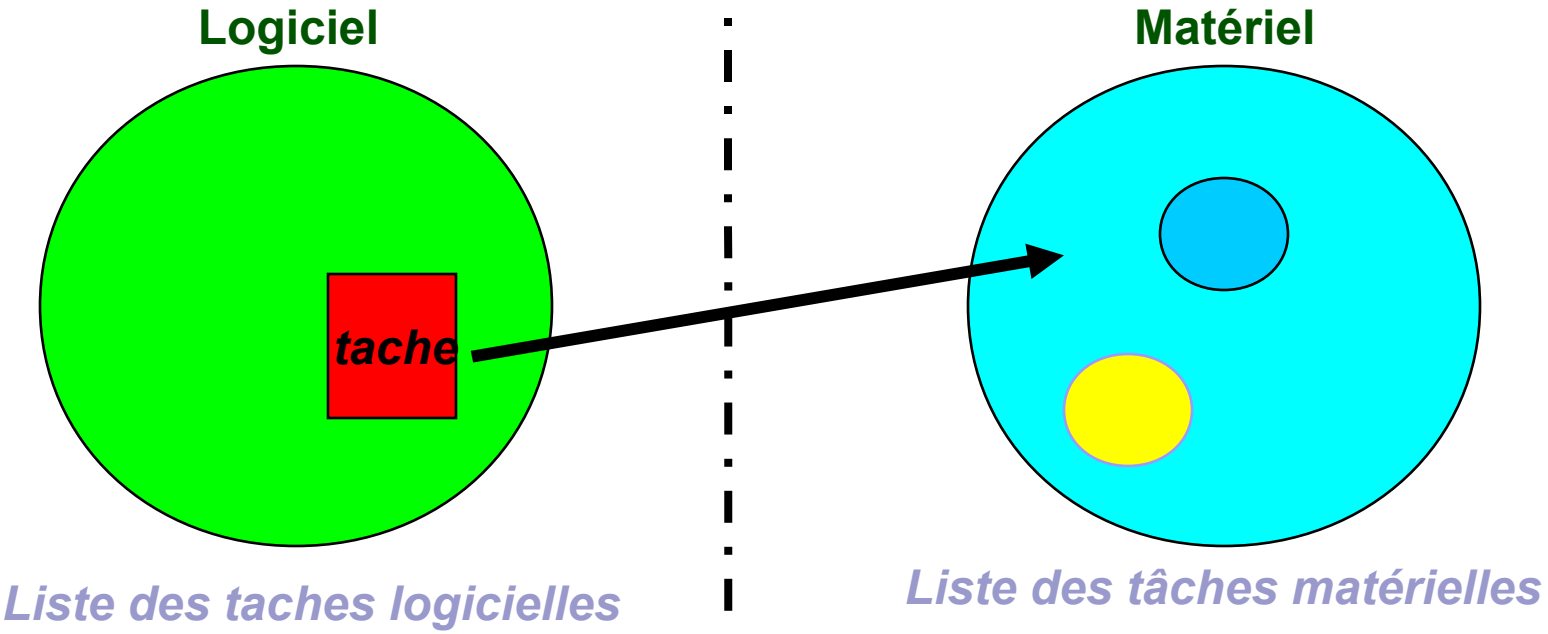
Gain = -5

nombre de coupures = 2

Co-Design: Approches

- **Stratégies opposées :**
  - Vulcan (“primal” approach)
    - Functionality all in HW (HardwareC) initially
    - Move some to CPU to reduce architecture cost
  - Cosyma (“dual” approach)
    - Functionality all in SW (C^x) initially
    - Move some to ASIC to meet performance goals
- **Lycos**
  - Convert all functionality to neutral form

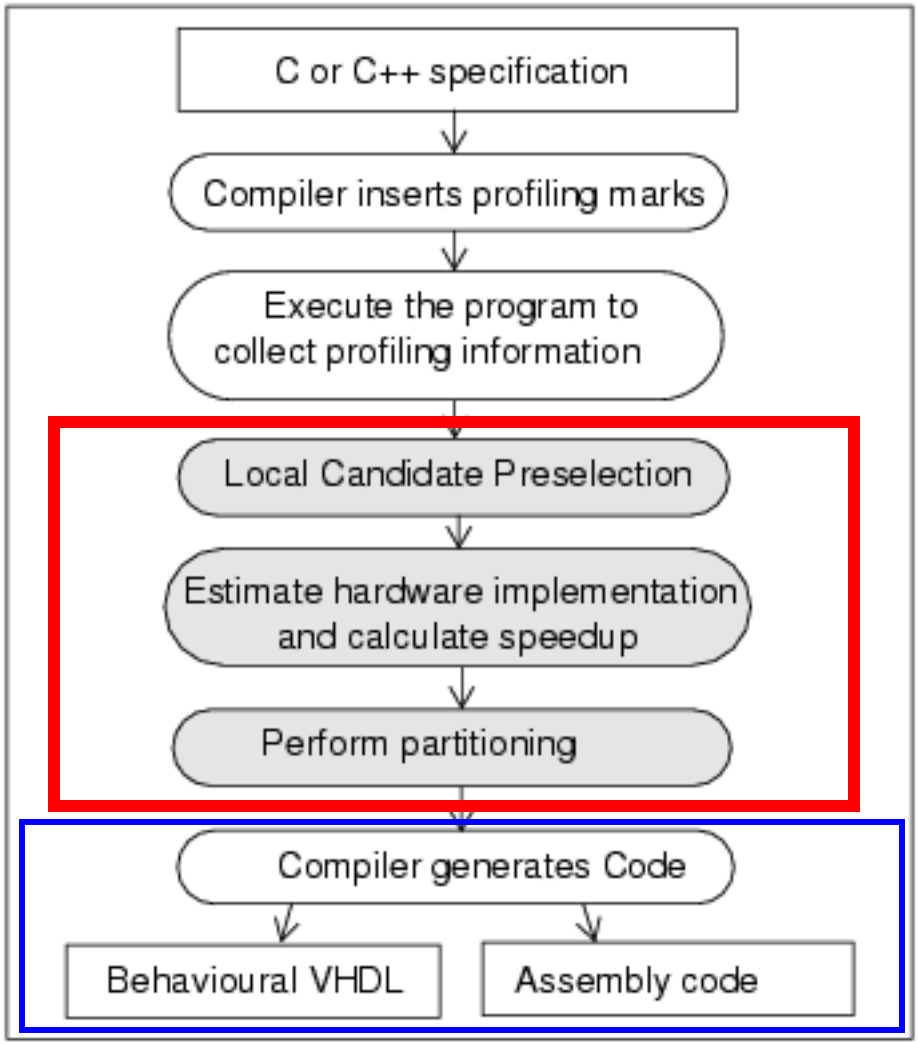
Algorithmes de Partitionnement



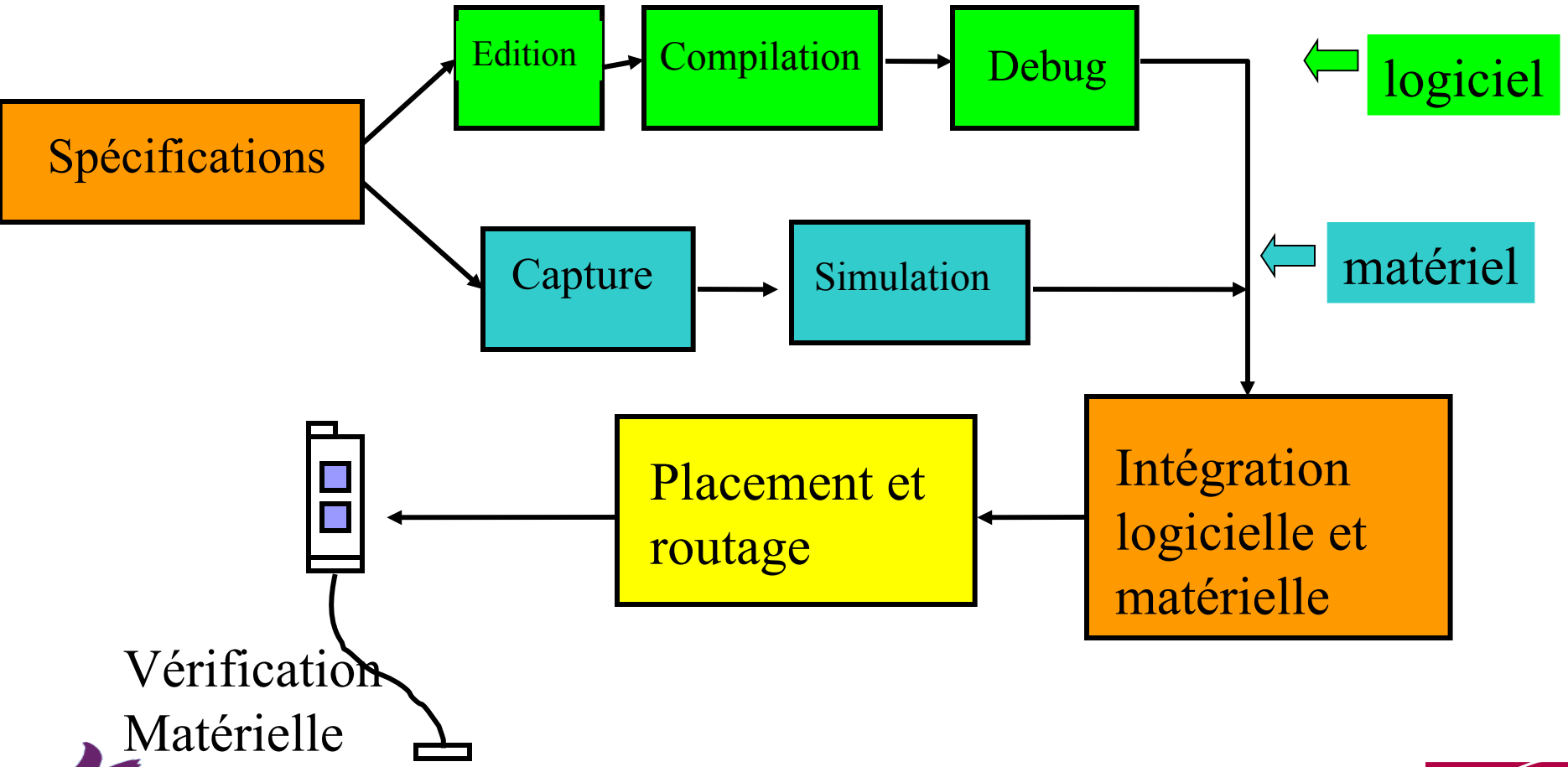
- Part du principe qu'initialement tout est fait en logiciel
- Sélection d'une tache pour échange logiciel vers matériel
- Migration vers le matériel et évaluation :
  - Temps, ressources matérielles, programmation et stockage, surplus temporel du à la synchronisation
- Evaluation des coût et évaluation des déplacement similaires à la minimisation du nombre de coupures et de la simulated annealing.

Analyse du Partitionnement

- Le profileur détermine les dépendances et permet une estimation des performances.
- Les résultat de la “compilation” est un code HDL synthétisable et un binaire exécutable par un processeur.



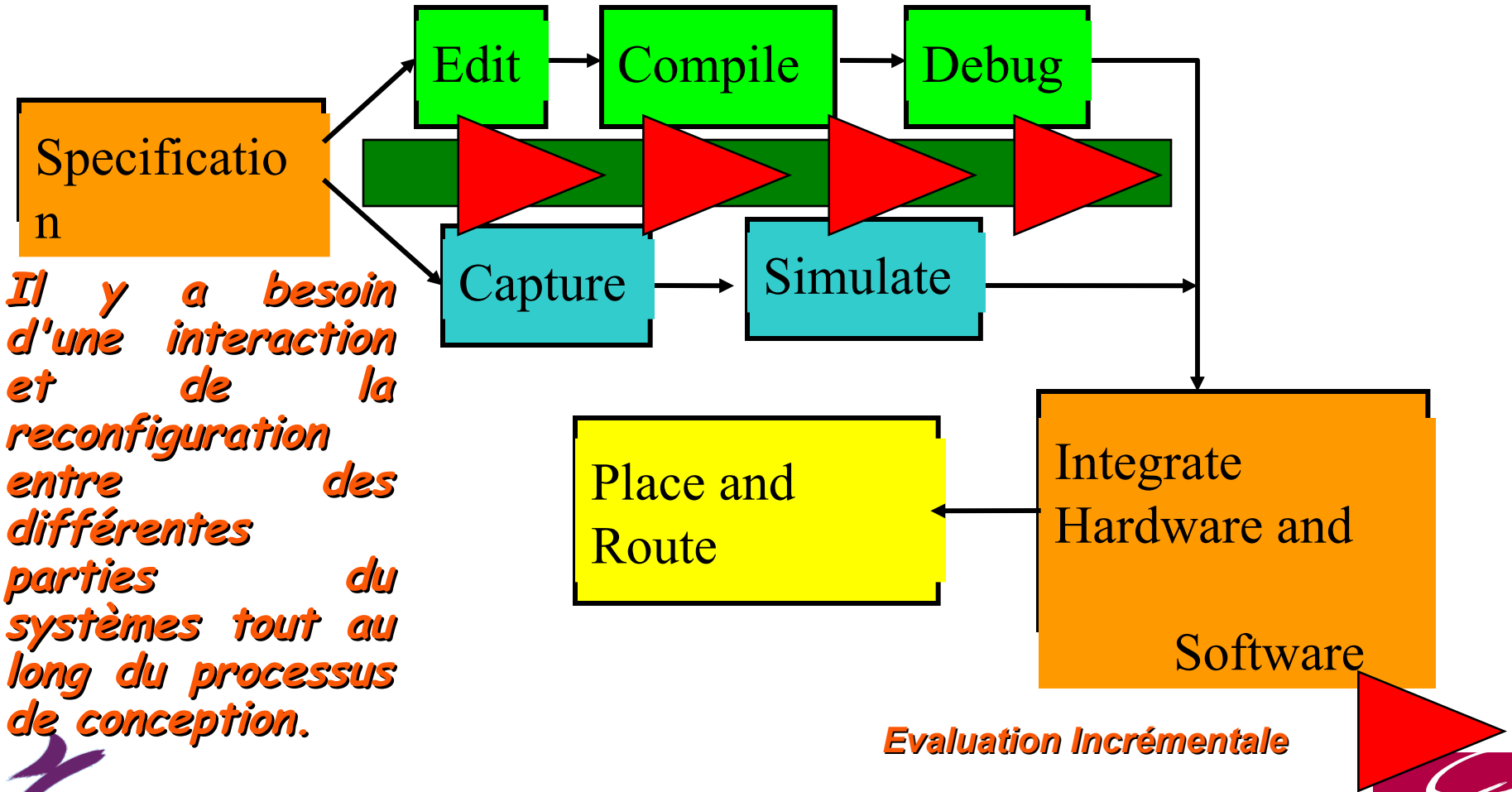
Processus du Co-design



Pratiques Courantes

- **Dans les flots conventionnels de conception, la décision de partage entre composants logiciels et matériel est décidée en amont très tôt, usuellement basée sur des méthodes ad-hoc, créant ainsi un problème communément appelé : "*Problème de la continuité du modèle*".**
- **Problèmes rencontrés :**
  - La possibilité d'explorer le compromis logiciel/matériel est limité (càd. le déplacement de fonctionnalité entre le logiciel et le matériel à travers la modification d'interfaces)
  - Les coûts augmentent et les fabrications sont reportées dû aux décisions tardives dans le cycle de conception (on ne s'aperçoit réellement de l'adéquation qu'à la fin du flot de conception)

Environnement de conception unifié



Il y a besoin d'une interaction et de la reconfiguration entre des différentes parties du systèmes tout au long du processus de conception.

Modélisation

Modèles Textuels

- Langage C
- VHDL
- SystemC
- …

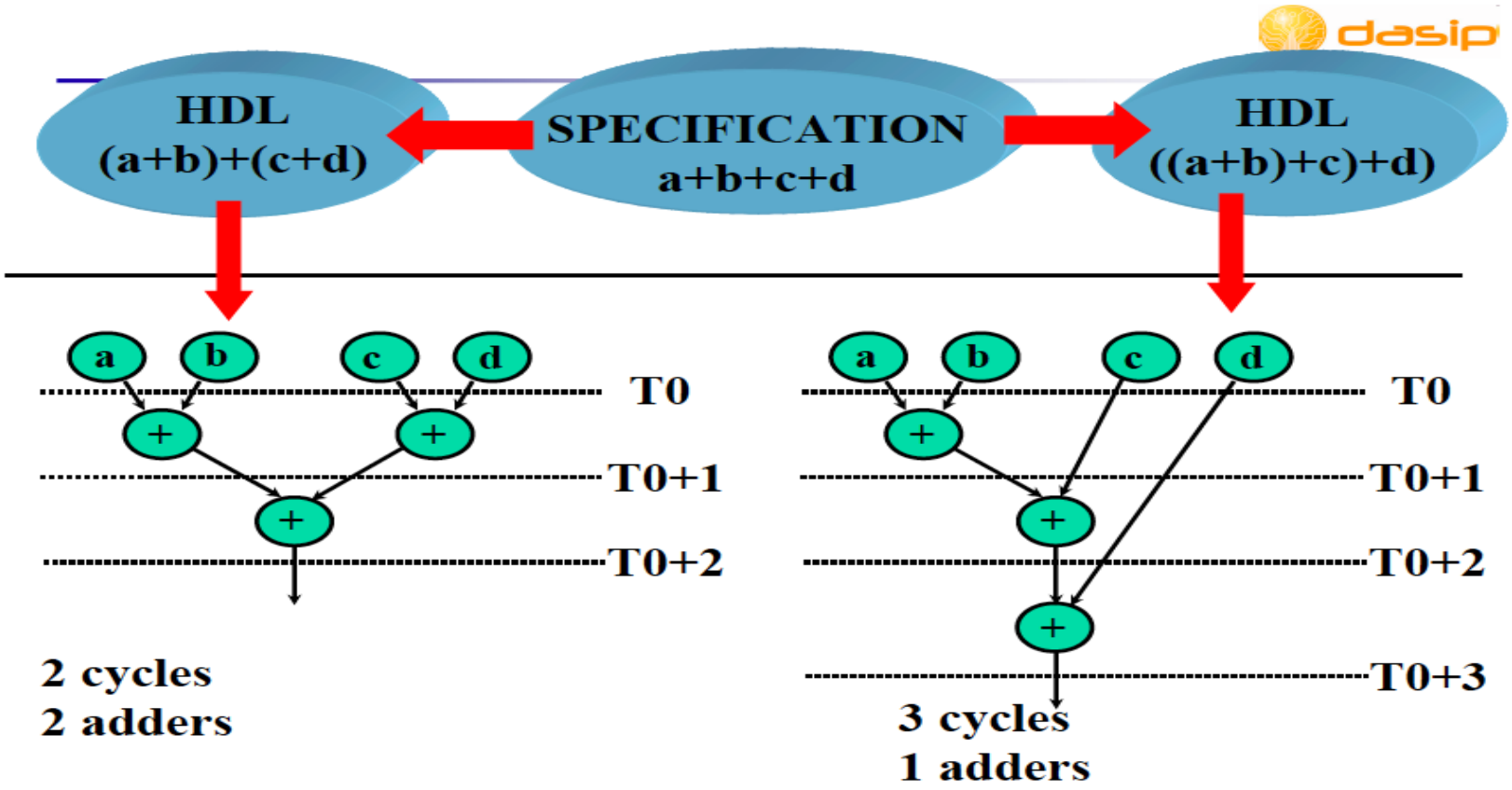
Modèles Graphiques

Modèles Graphiques

Graphes Flot de Données

Modèles Graphiques

Graphes Flot de Données



Modèles Graphiques

KPN : Khan Processus Network

Modèles Graphiques

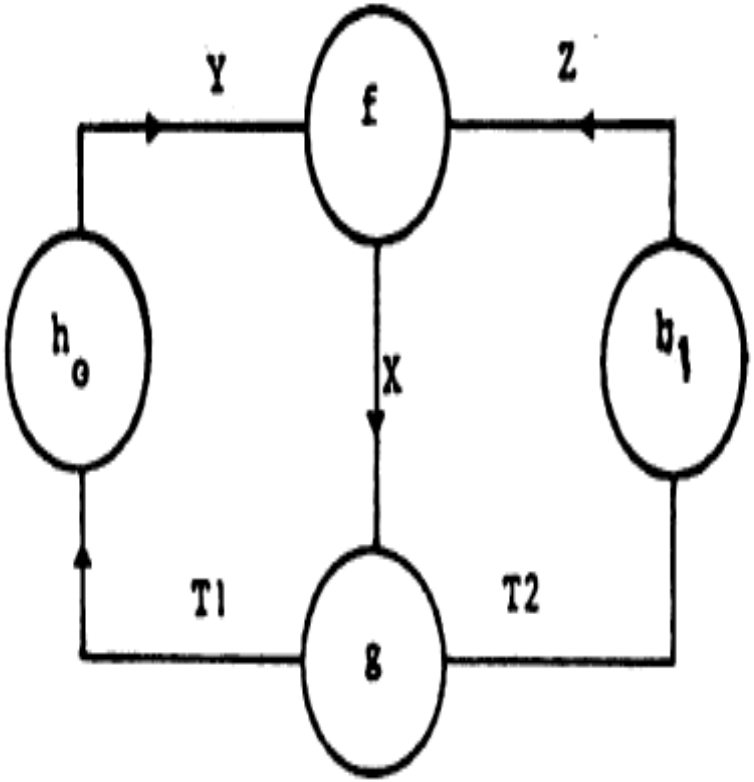
KPN : Khan Processus Network

```
 Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer  in U,V; integer out W) ;
    Begin integer I ; logical B ;
          B := true ;
          Repeat Begin
(4)         I := if B then wait(U) else wait(V) ;
(7)         print (I) ;
(5)         send I on W ;
            B := ¬B ;
            end ;
    End ;
  Process g(integer in U ; integer out V, W) ;
    Begin integer I ; logical B ;
      B := true ;
      Repeat Begin
        I := wait (U) ;
        if B then send I on V else send I on  W ;
        B := ¬B ;
        End ;
    End ;
(3) Process h(integer in U;integer out V; integer INIT);
    Begin integer I ;
      send INIT on V ;
      Repeat Begin
        I := wait(U) ;
        send I on V ;
        End ;
    End ;
  Comment : body of mainprogram ;
(6) f(Y,Z,X) par g(X,T1,T2) par h(T1,Y,0) par h(T2,Z,1
  End ;
```

Fig.1. Sample parallel program S.

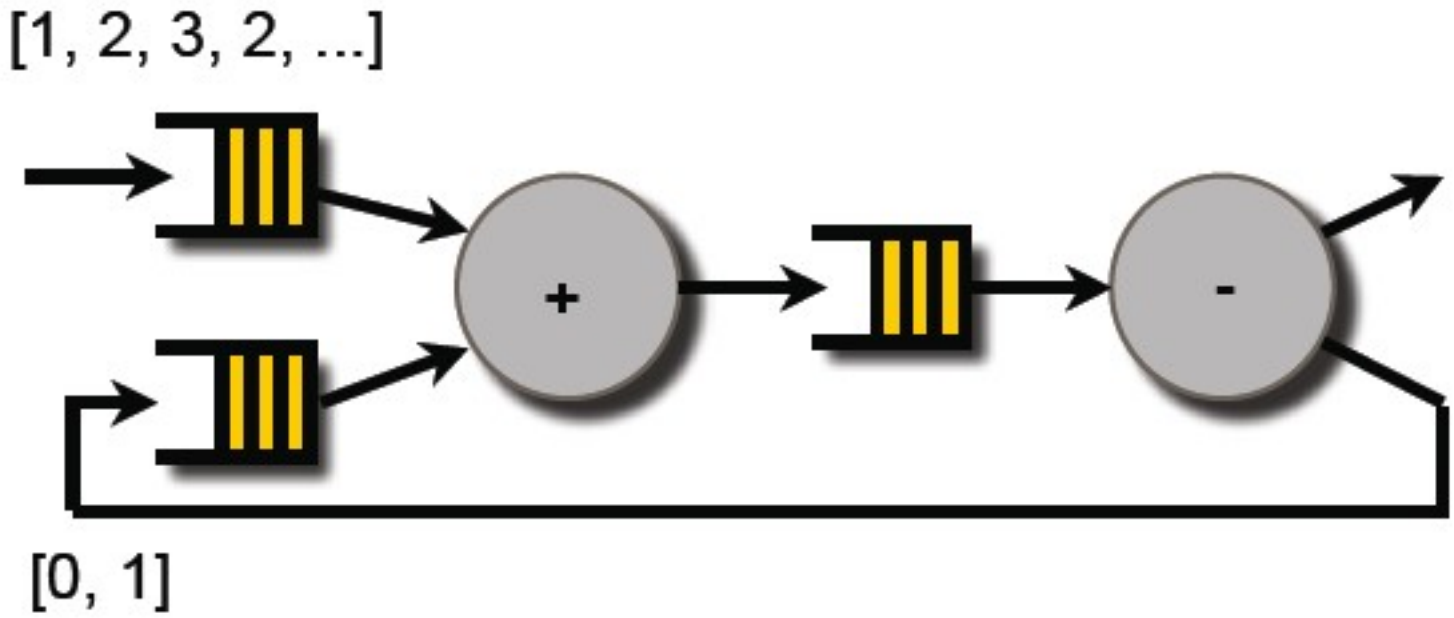
Modèles Graphiques

KPN : Khan Processus Network



Modèles Graphiques

KPN : Khan Processus Network



Modèles Graphiques

FSM : Finite State Machine

Modèles Graphique

Les réseaux de Pétri

RdP : origine et domaines d'application

Origine :

Idées de départ de Carl Adam Petri (thèse en 1962) :

Un ensemble d'automates à états finis qui communiquent

Avoir à la fois la représentation des automates

Et celle des communications par les mêmes primitives

- communications asynchrones par échange de messages
- communication synchrones par rendez-vous, synchronisations, ressources partagées

RdP : origine et domaines d'application

Domaines d'application :

Systèmes de production, Autom. Prog. Ind., Grafcet

- Evaluation des performances, simulation à événements discrets

Validation de protocoles de communication

Systèmes temps réels, systèmes distribués, génie logiciel

Systèmes d'information, gestion, interfaces homme-machine

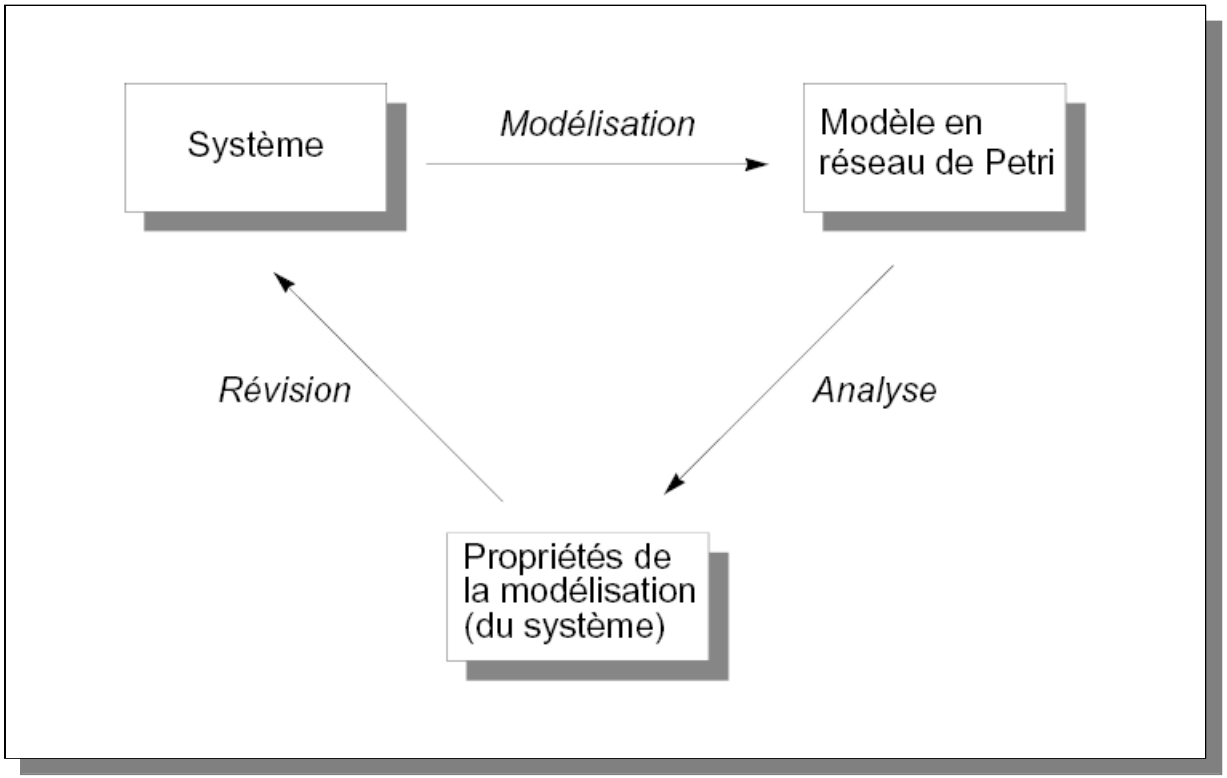
Modèles de raisonnement, planification

RdP : présentation informelle

- Le formalisme des réseaux de Petri est un outil permettant l'étude de systèmes dynamiques et discrets.
- Il s'agit d'une représentation mathématique permettant la modélisation d'un système.
- L'analyse d'un réseau de Petri peut révéler des caractéristiques importantes du système concernant sa structure et son comportement dynamique.
- Les résultats de cette analyse sont utilisés pour évaluer le système et en permettre la modification et/ou l'amélioration le cas échéant.

RdP : présentation informelle

Démarche générale



RdP : présentation informelle

Concepts de base

Condition

- Une condition est un prédicat ou une description logique d'un état du système.
- Une condition est vraie ou fausse.
- Un état du système peut être décrit comme un ensemble de conditions.

Evénement

- Les événements sont des actions se déroulant dans le système.
- Le déclenchement d'un événement dépend de l'état du système.

Déclenchement, précondition, postcondition

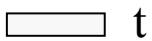
- Les conditions nécessaires au déclenchement d'un événement sont les *préconditions* de l'événement.
- Lorsqu'un événement se produit, certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres conditions, appelées *postconditions* de l'événement deviennent vraies.

4.2. RdP : présentation informelle

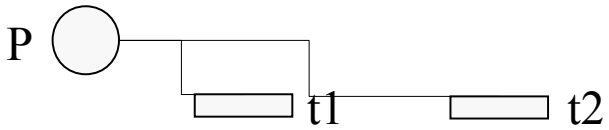
Concepts de base

Condition = Place

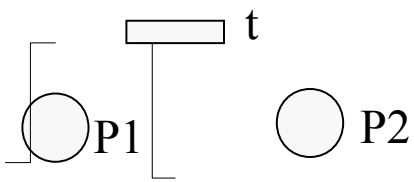
Evénement = Transition



précondition = arc Place -> Transition



postcondition = arc Transition -> Place



RdP : présentation informelle

Concepts de base

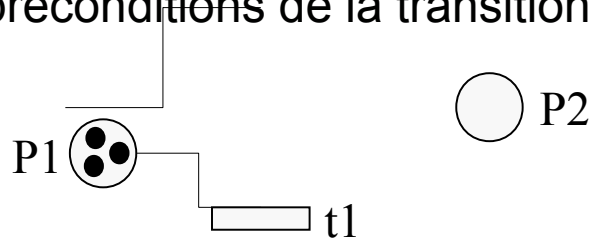
Satisfaction d'une Condition = Jeton dans une Place

Remarque : on peut avoir un nombre quelconque non borné de jetons dans une place

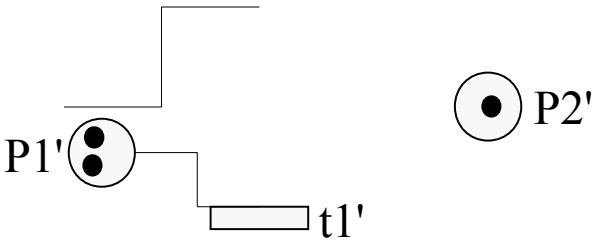
RdP : présentation informelle

Concepts de base

Condition de franchissement d'une transition = satisfaction de toutes les places préconditions de la transition



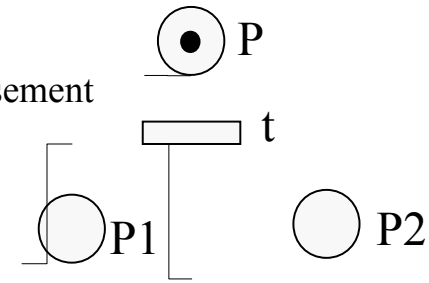
non franchissable

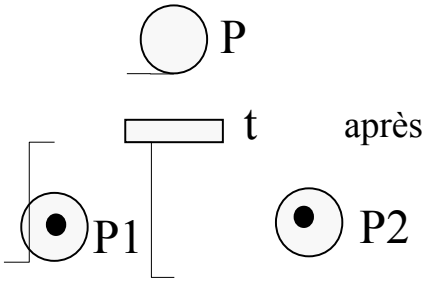


franchissable

Effet du franchissement d'une transition = satisfaction de toutes les places postconditions de la transition

avant franchissement





après franchissement

RdP : présentation informelle

Modélisation de systèmes avec *ressources*

Pour certains systèmes, il est plus juste de raisonner en termes d'ensemble de ressources, au sens large, qu'en termes de conditions-événements.

=> un jeton = une ressource

Le nombre de jetons contenus dans une place reflète le nombre de ressources qu'elle possède.

Les jetons d'une place n'ont pas d'identité individuelle, autrement dit ils sont indiscernables.

Ces ressources sont consommées et produites par les événements du système.

Les arcs entrants d'une transition peuvent être valués par un entier quelconque (non nul)

=> valuation = nombre de jeton nécessaires dans la place pour franchir la transition

=> si k est la valuation d'un arc d'une place P vers une transition T, le tir de la transition T retire k jetons dans la place P

Les arcs sortants d'une transition peuvent être valués par un entier quelconque (non nul)

=> valuation = nombre de jeton produits dans la place située après la transition

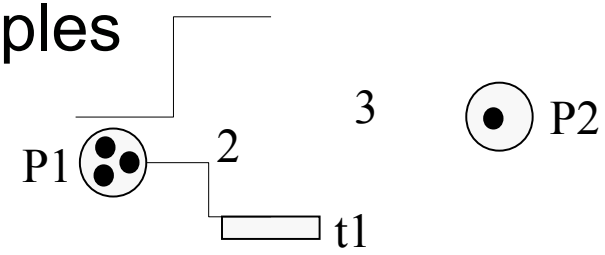
=> si k est la valuation d'un arc d'une transition T vers une place P, le tir de la transition T dépose k jetons dans la place P

Par défaut, les arcs sont valués par 1.

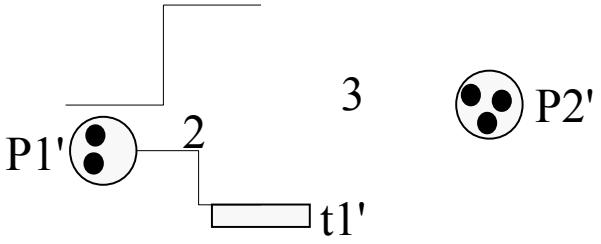
RdP : présentation informelle

Concepts de base

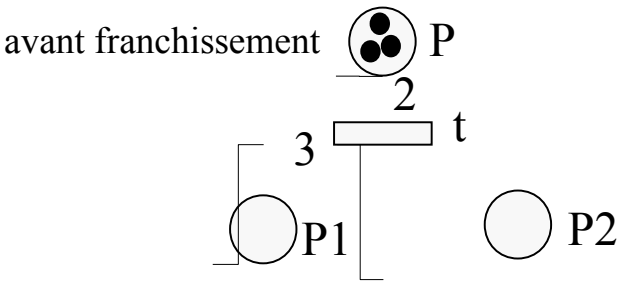
Exemples

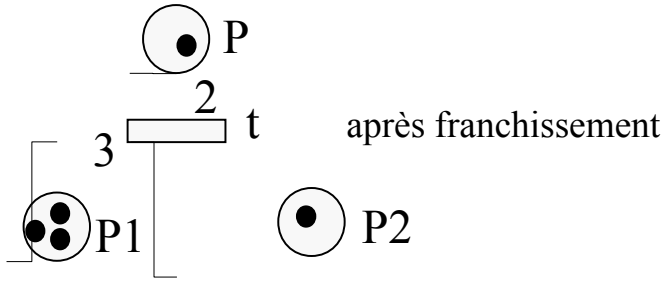


non franchissable



franchissable

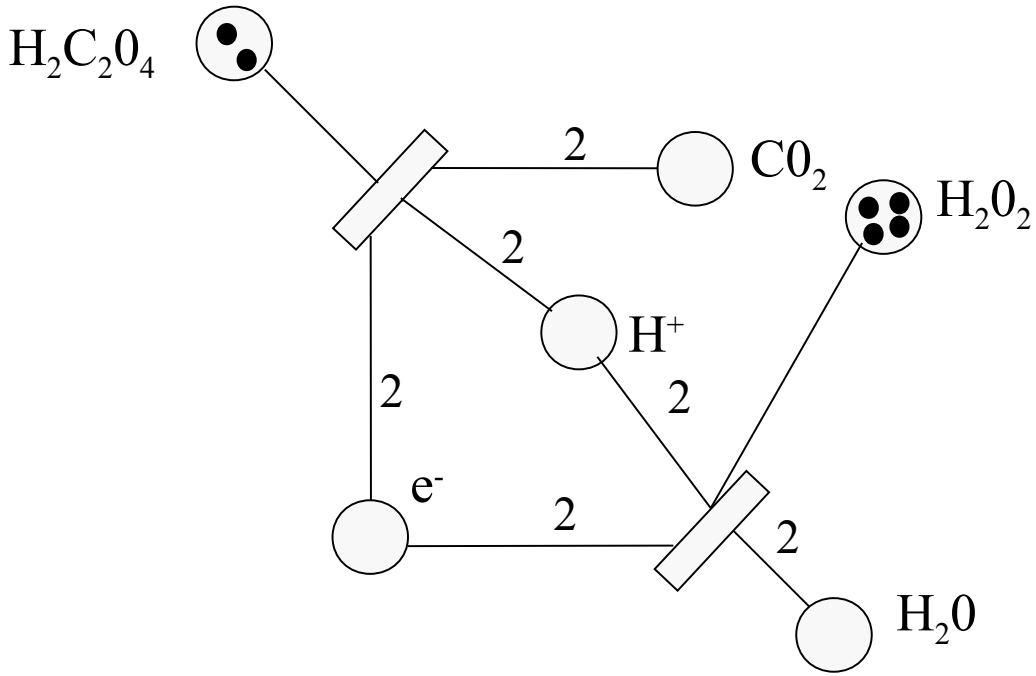




RdP : présentation informelle

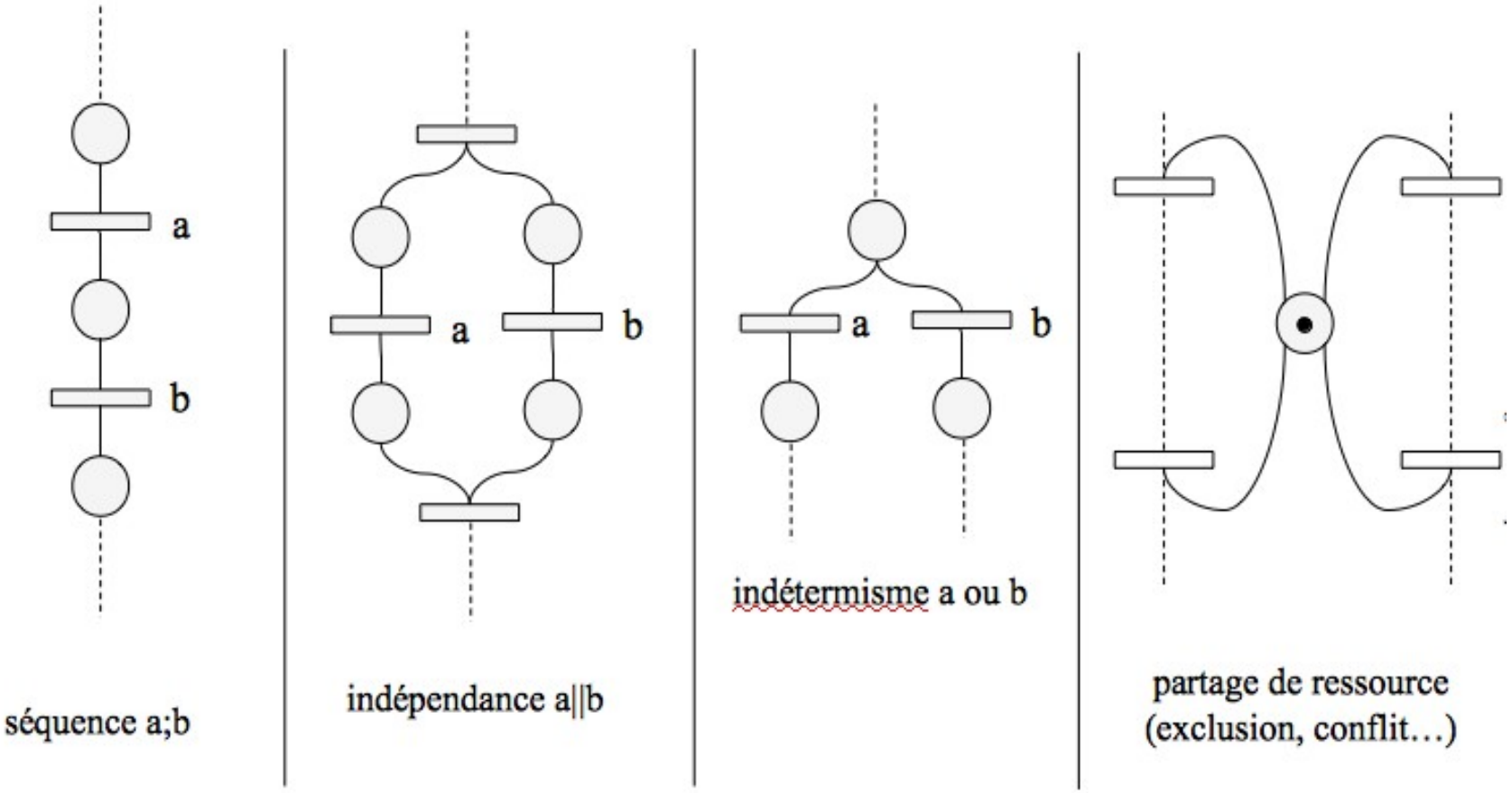
Exemple…

une réaction chimique d'oxydo-réduction



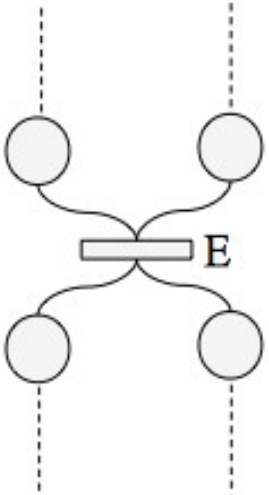
RdP : présentation informelle

Schémas particuliers

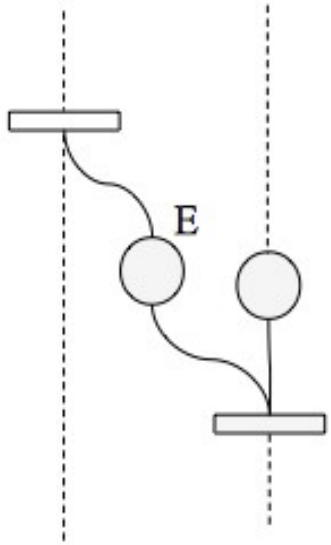


RdP : présentation informelle

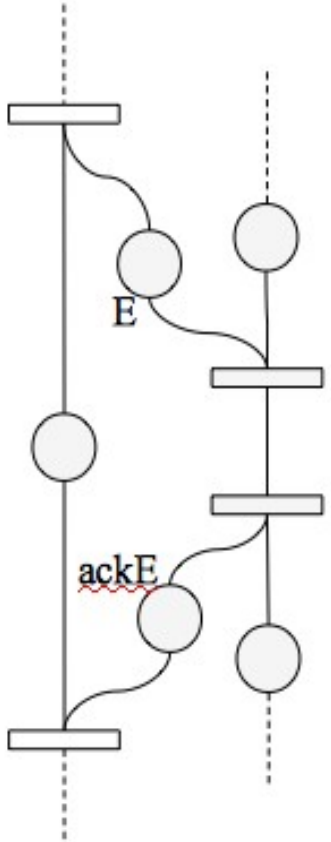
Schémas particuliers



synchonisation a=b,
envoisynchrone
d'un message E



sémaphore,
envoi asynchrone
d'un message E



envoi asynchrone
d'un message E
avec acquittement

4.2. RdP : présentation informelle

Notions complémentaires

Une transition-puit est une transition ayant une sortie vide.

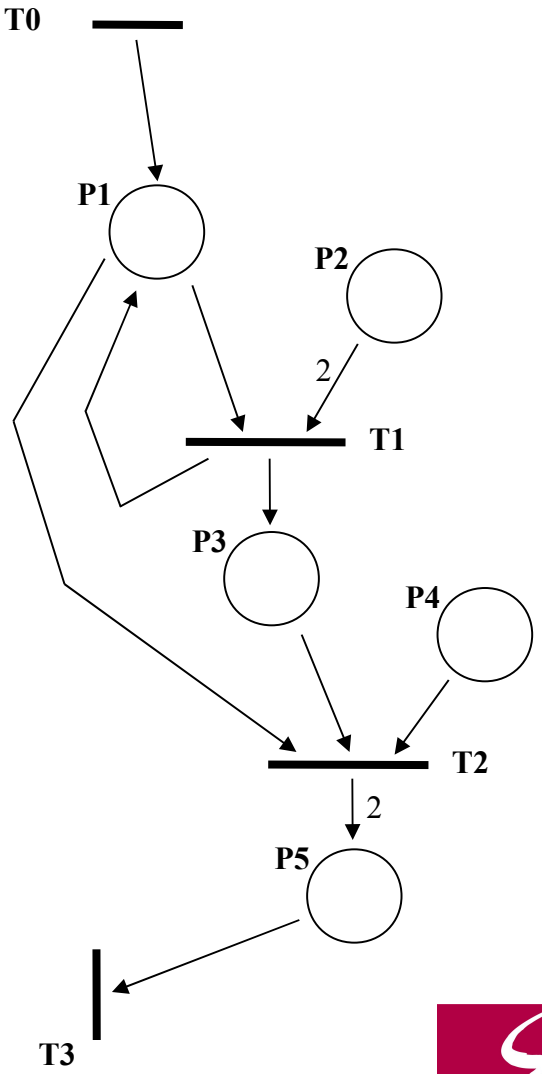
Une transition-source est une transition ayant une entrée vide.

Une boucle est un circuit constitué d'une seule place et d'une seule transition.

Un RdP sans boucle est dit pur

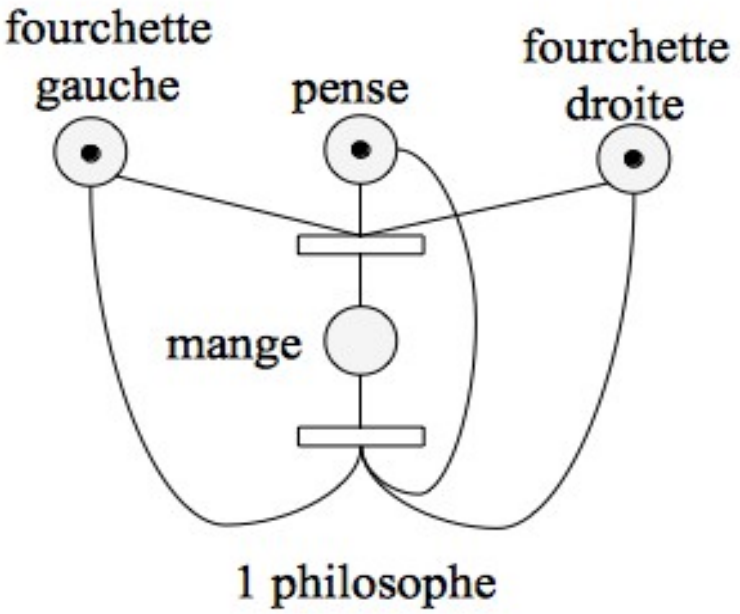
Exemple :

- T0 est une transition-source.
- T3 est une transition-puit.



RdP : présentation informelle

Exemple : les 5 philosophes

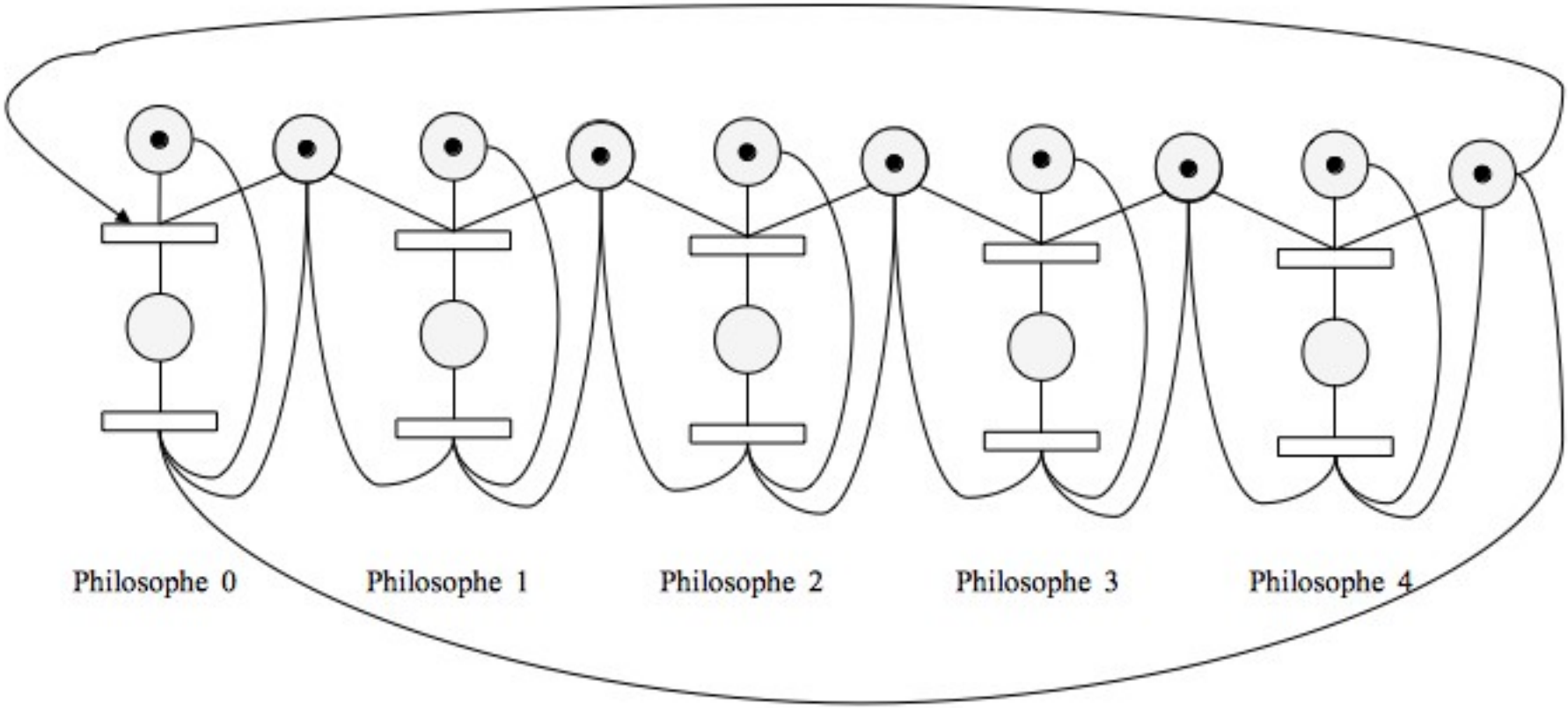


=> composition par mise en commun des fourchettes

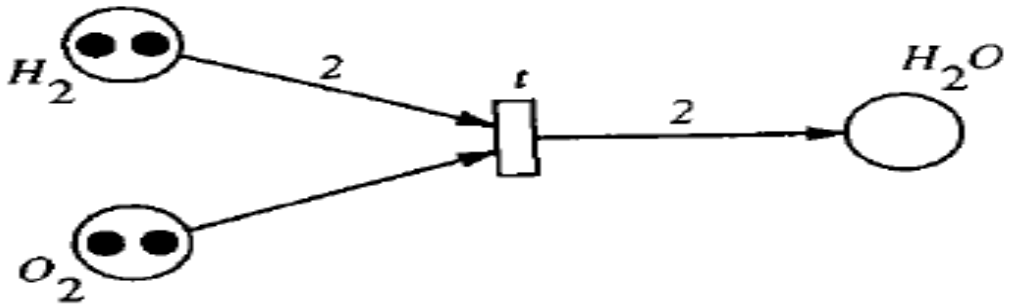
=> fusion des places « fourchette droite » du philosophe N et « fourchette gauche » du philosophe N+1 (modulo 5)

RdP : présentation informelle

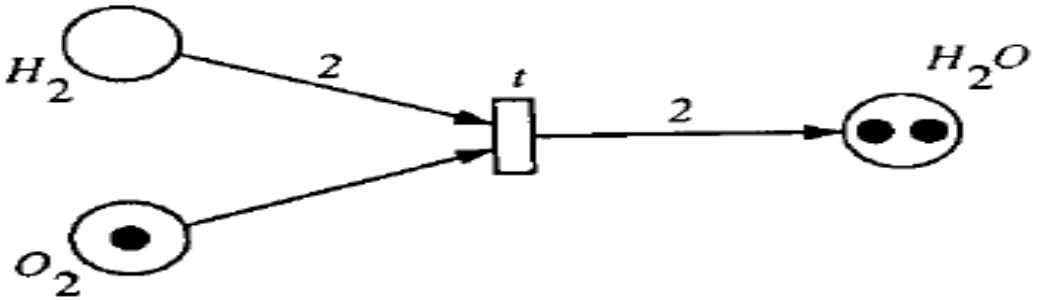
Exemple : les 5 philosophes



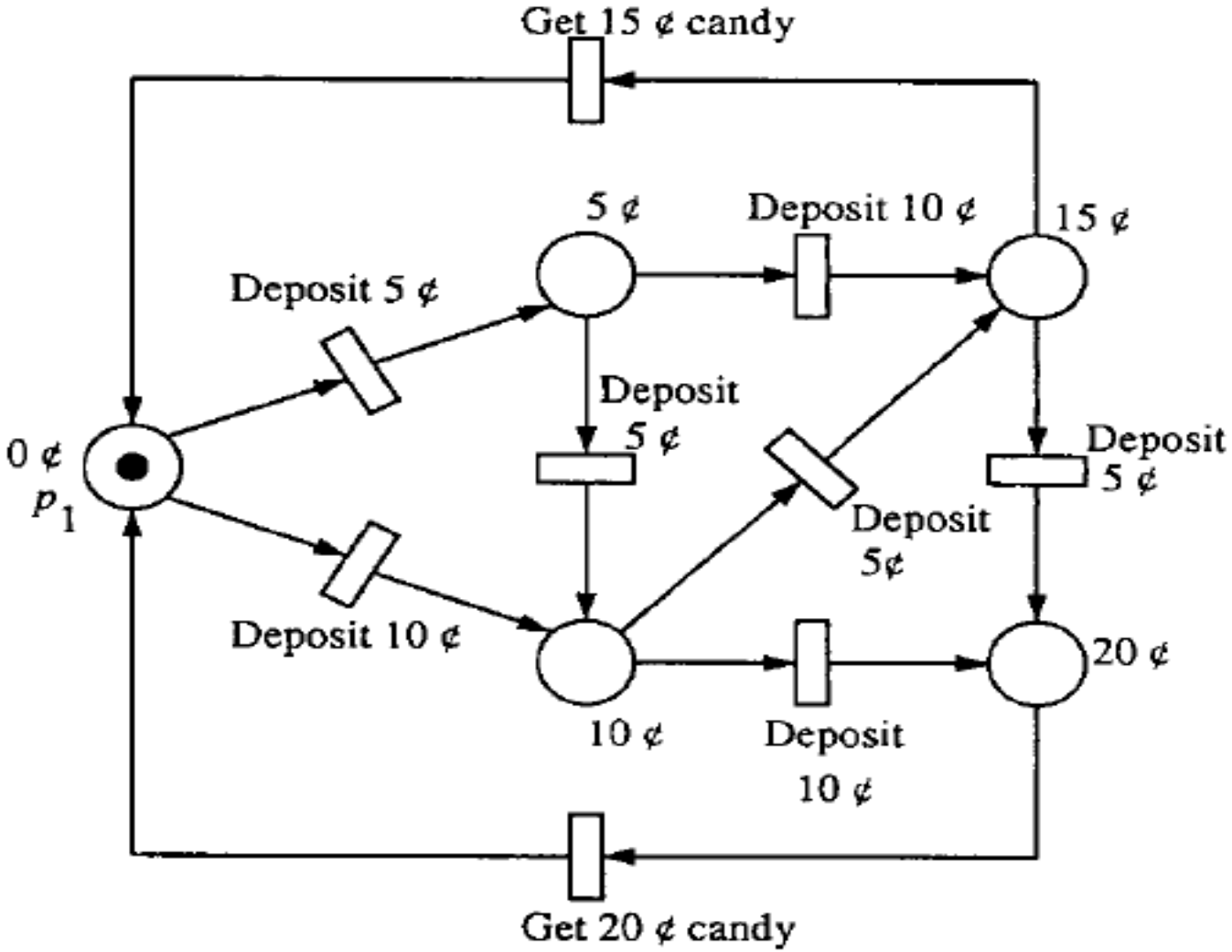
RdP : Exemples – l'eau



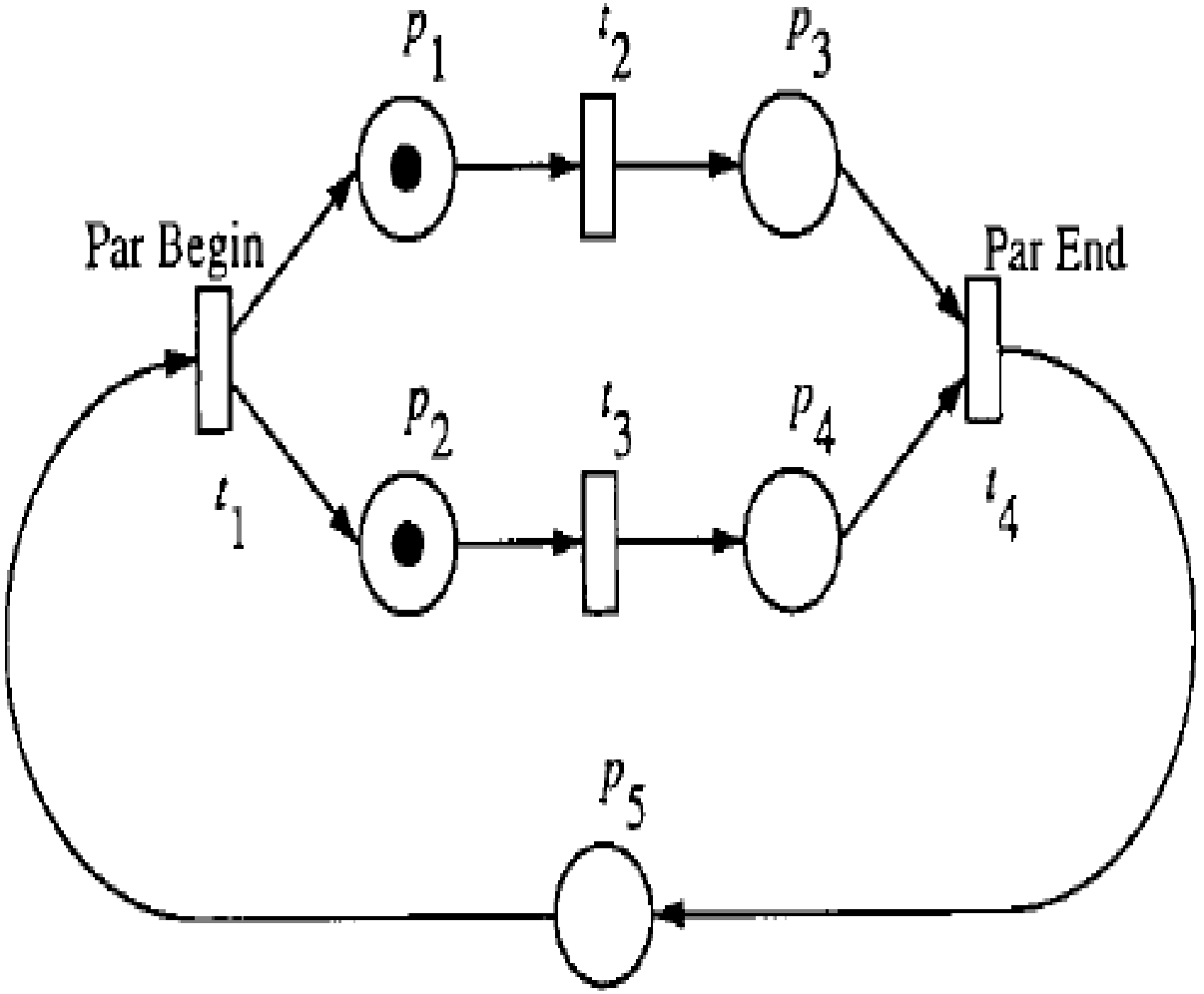
(a)



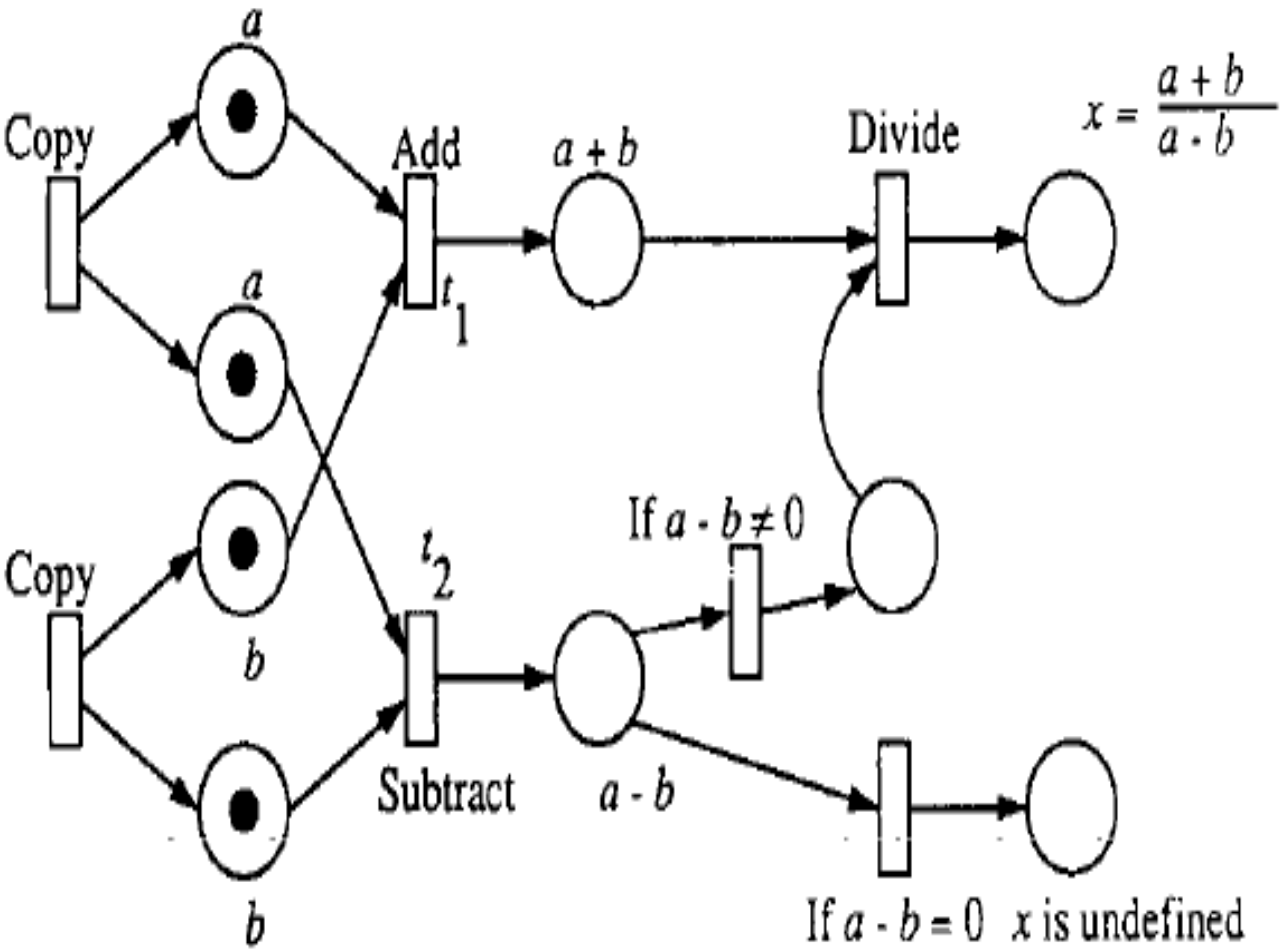
RdP : Exemples – distributeur



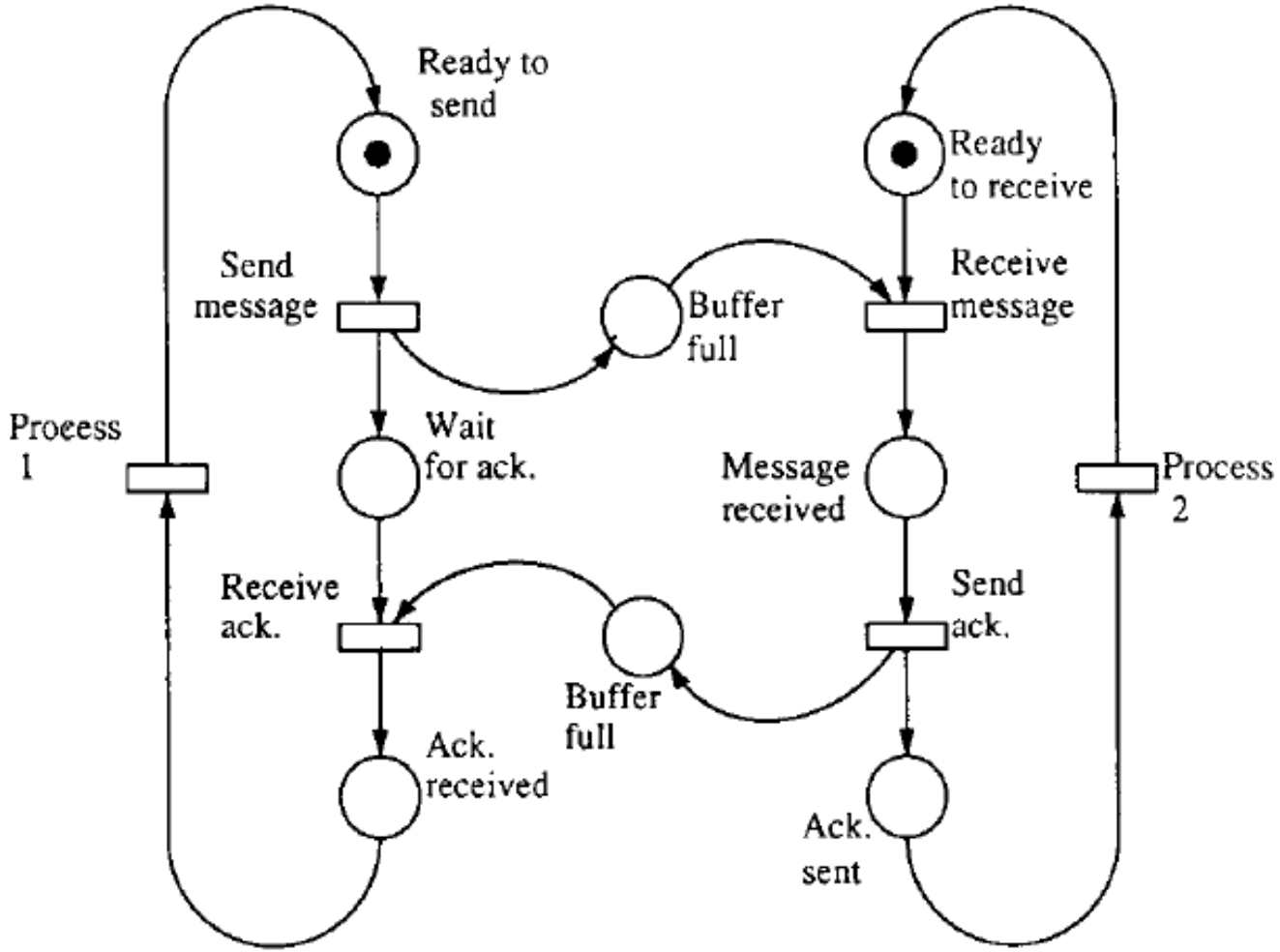
RdP : Exemples – parallélisation



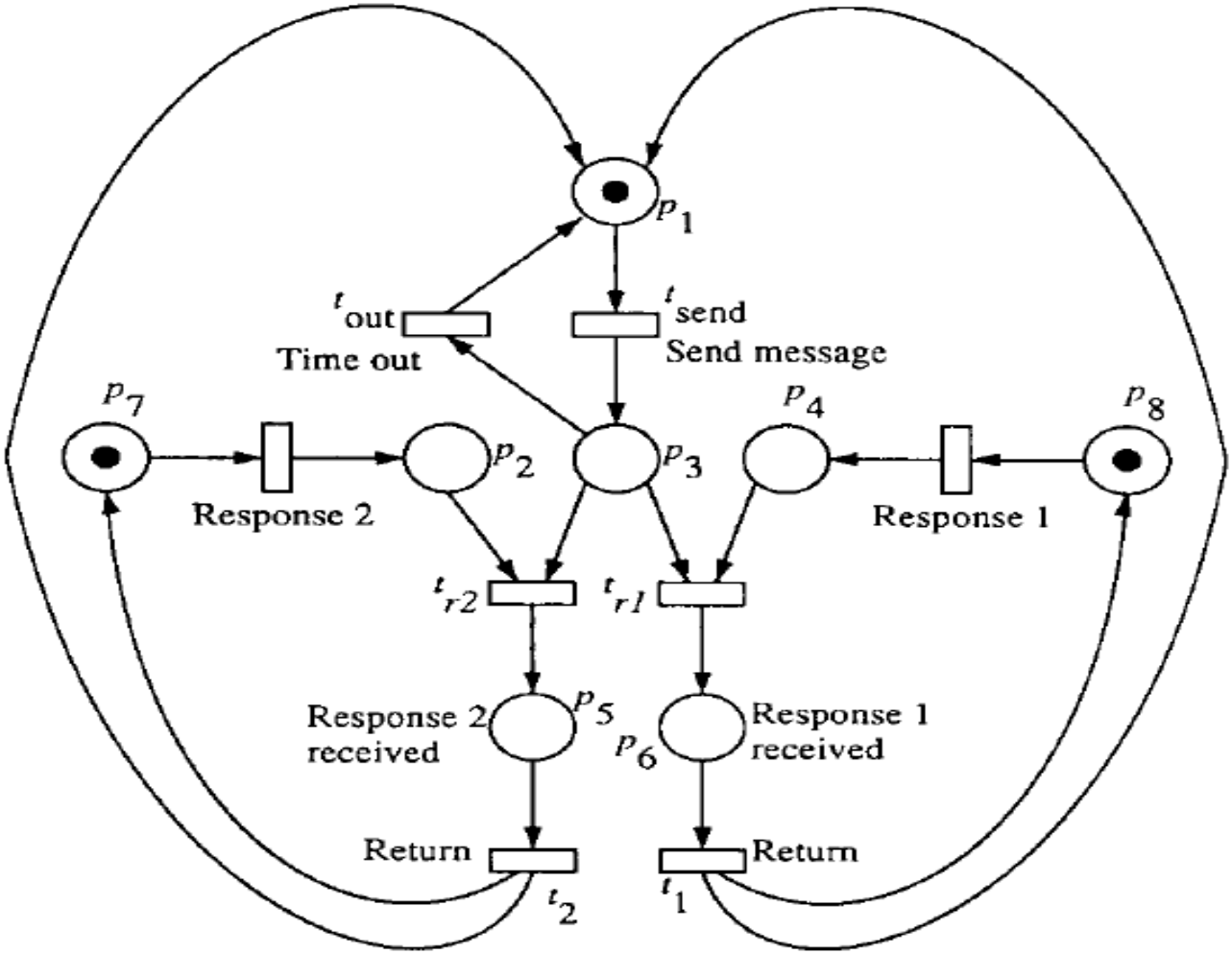
RdP : Exemples – UAL



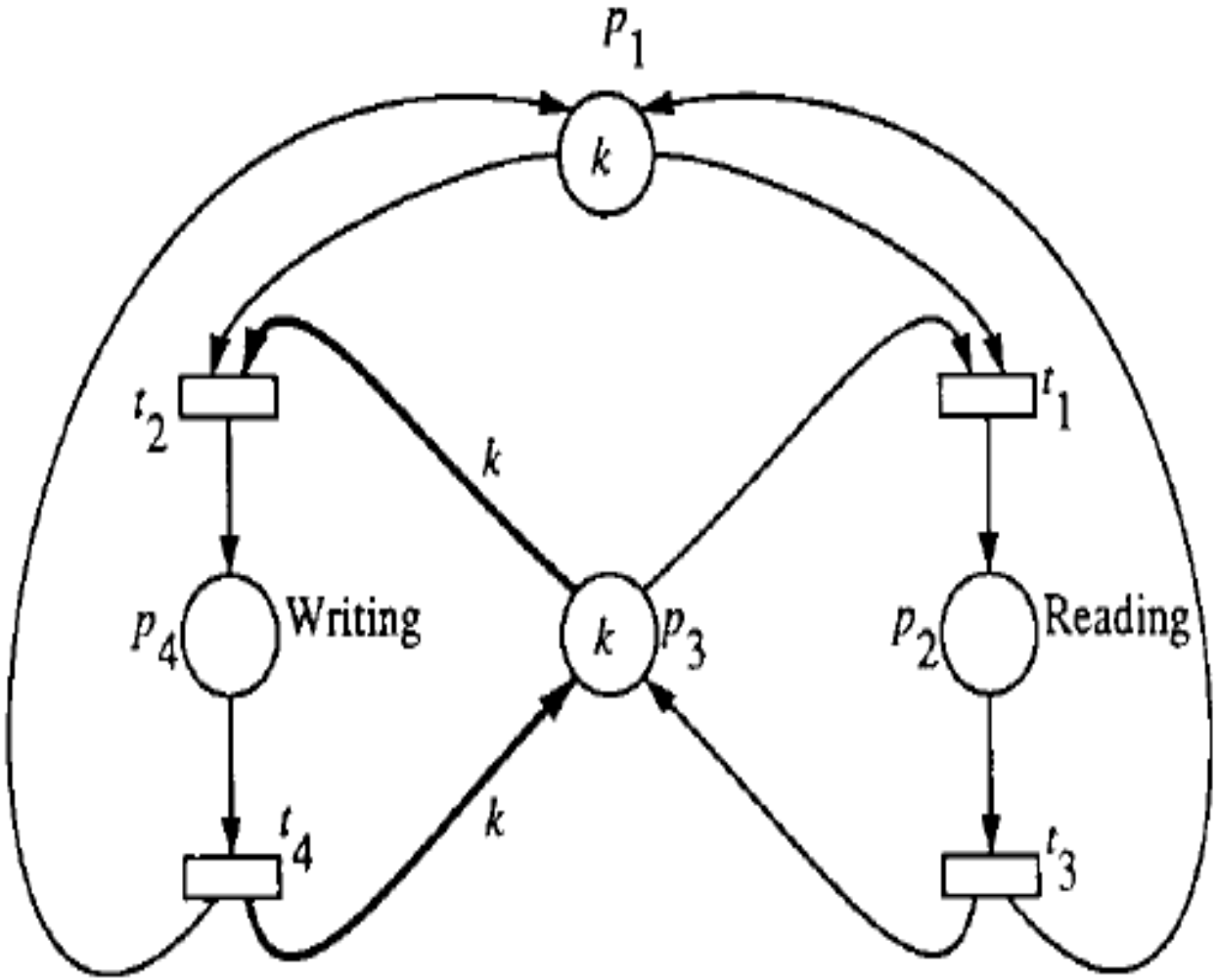
RdP : Exemples – Emission/Réception



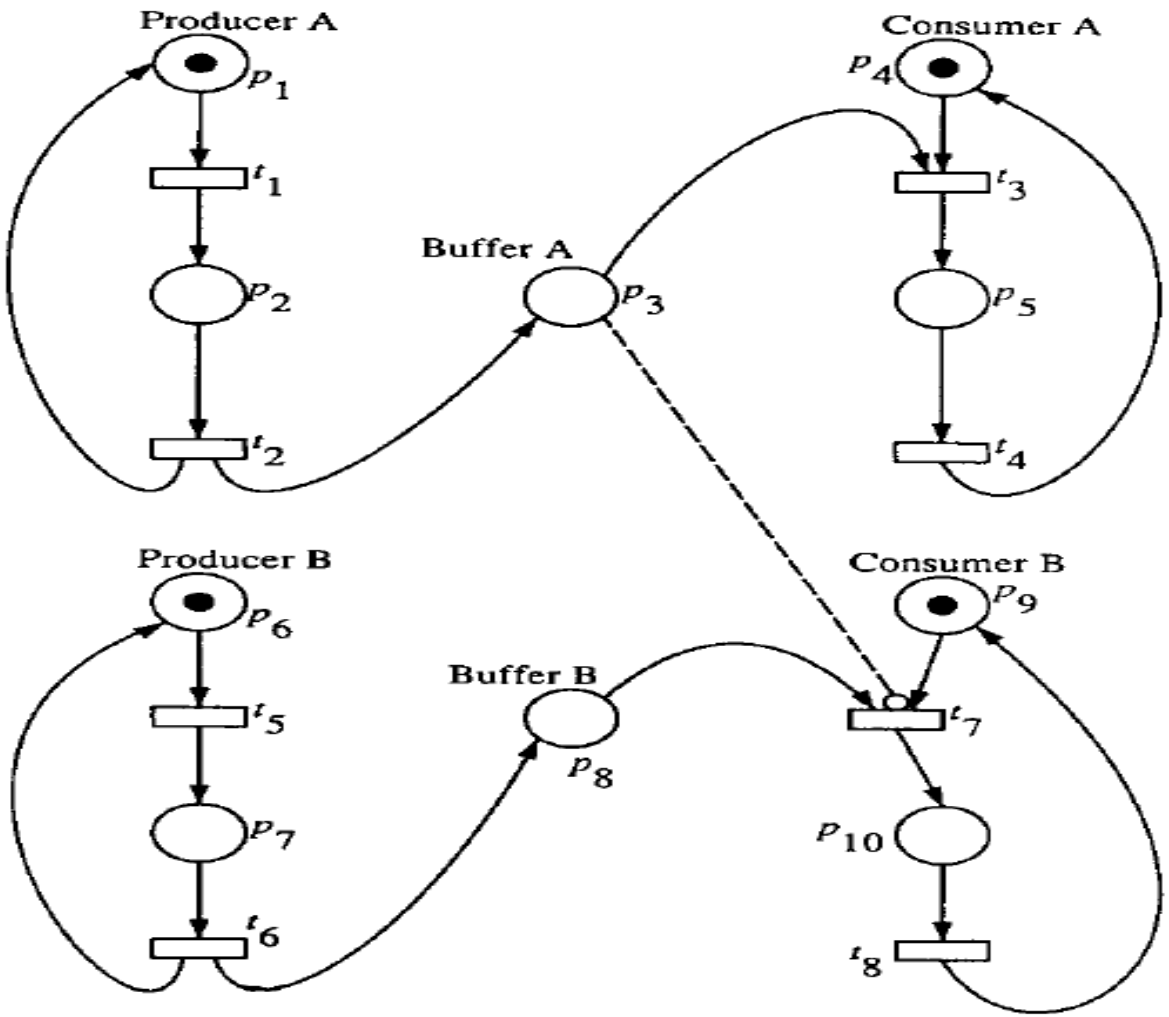
RdP : Exemples – Emission/Réception



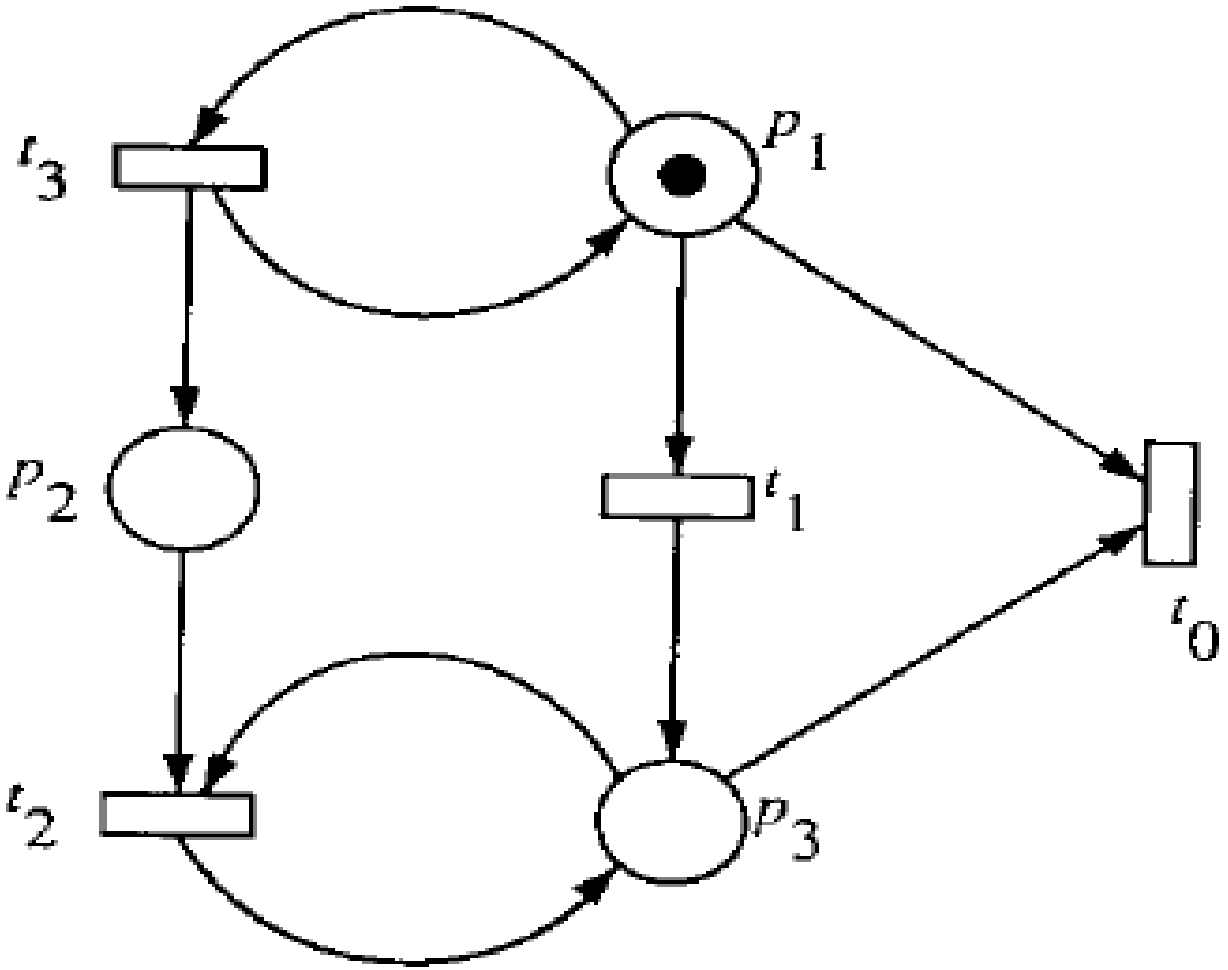
RdP : Exemples – Mémoire partagée



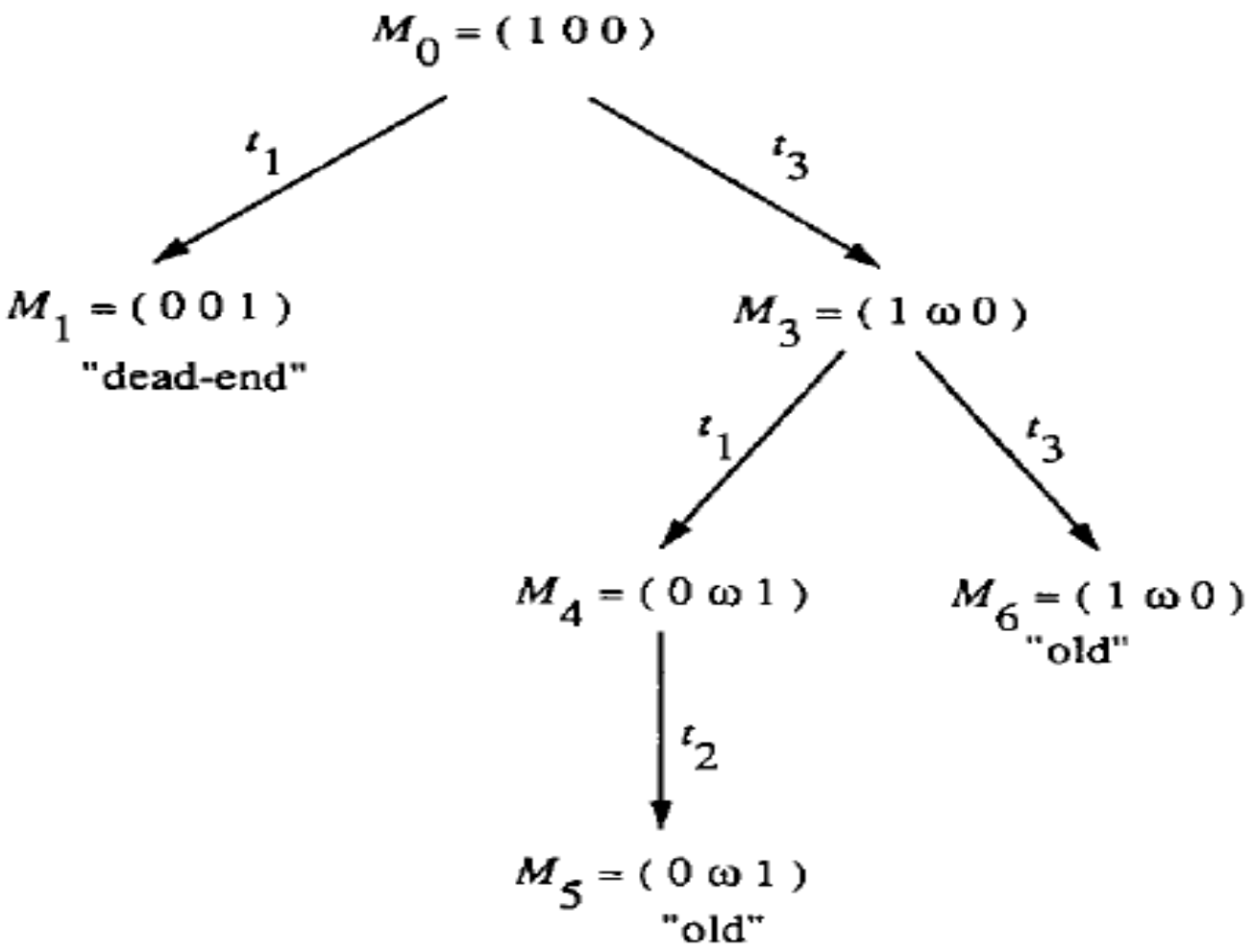
RdP : Exemples – Mémoire partagée



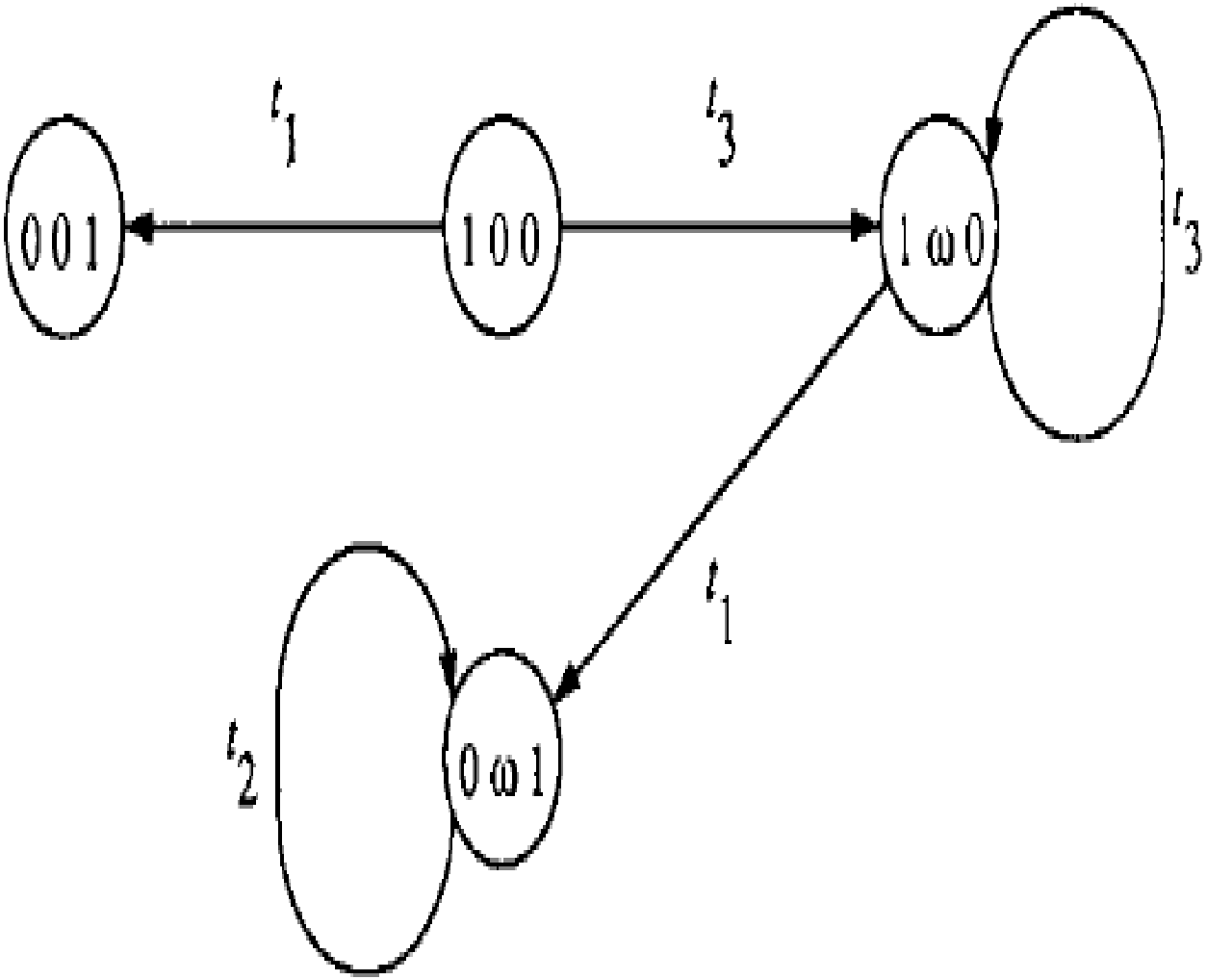
RdP : Quelques Propriétés



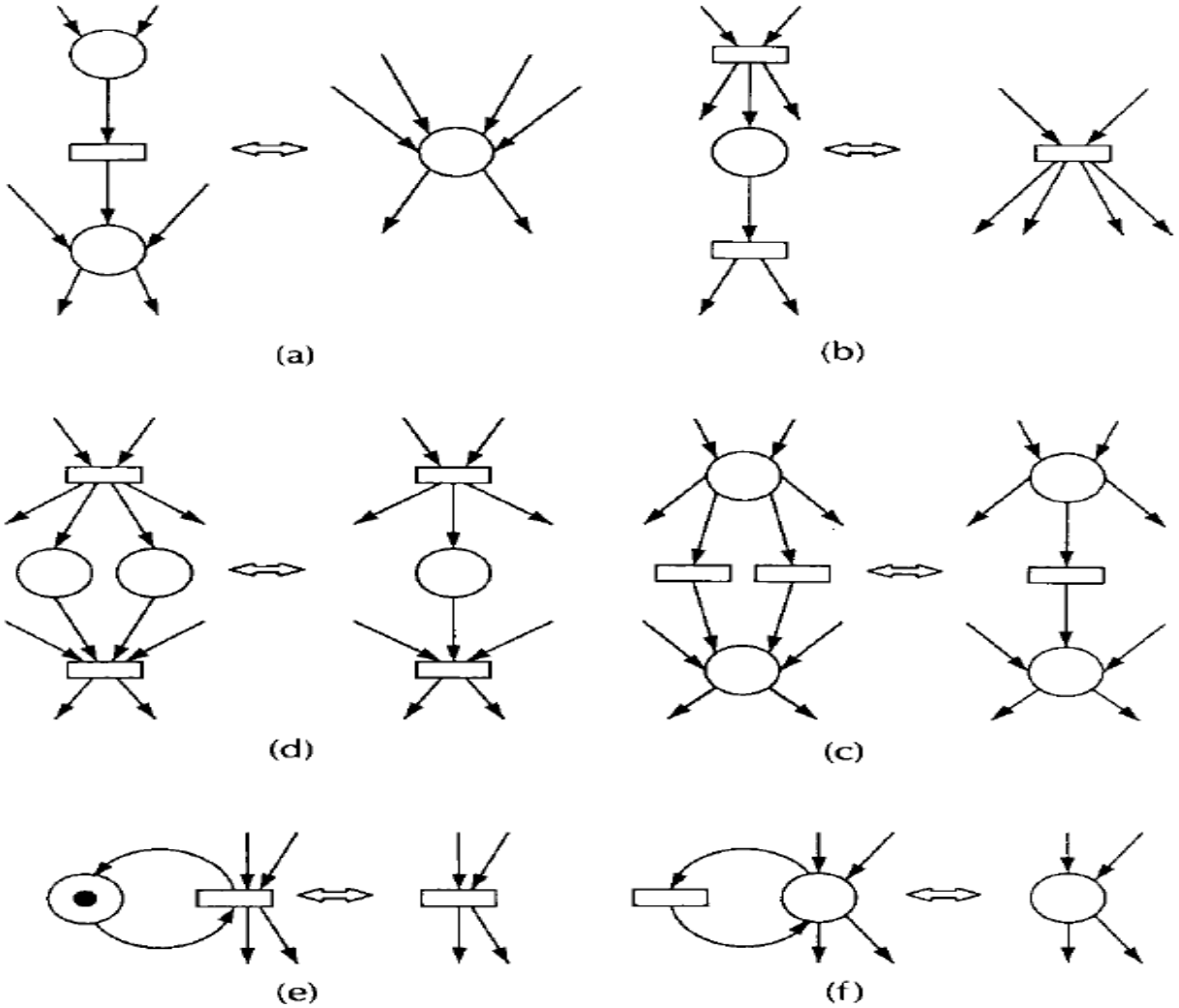
RdP : Reachabilité



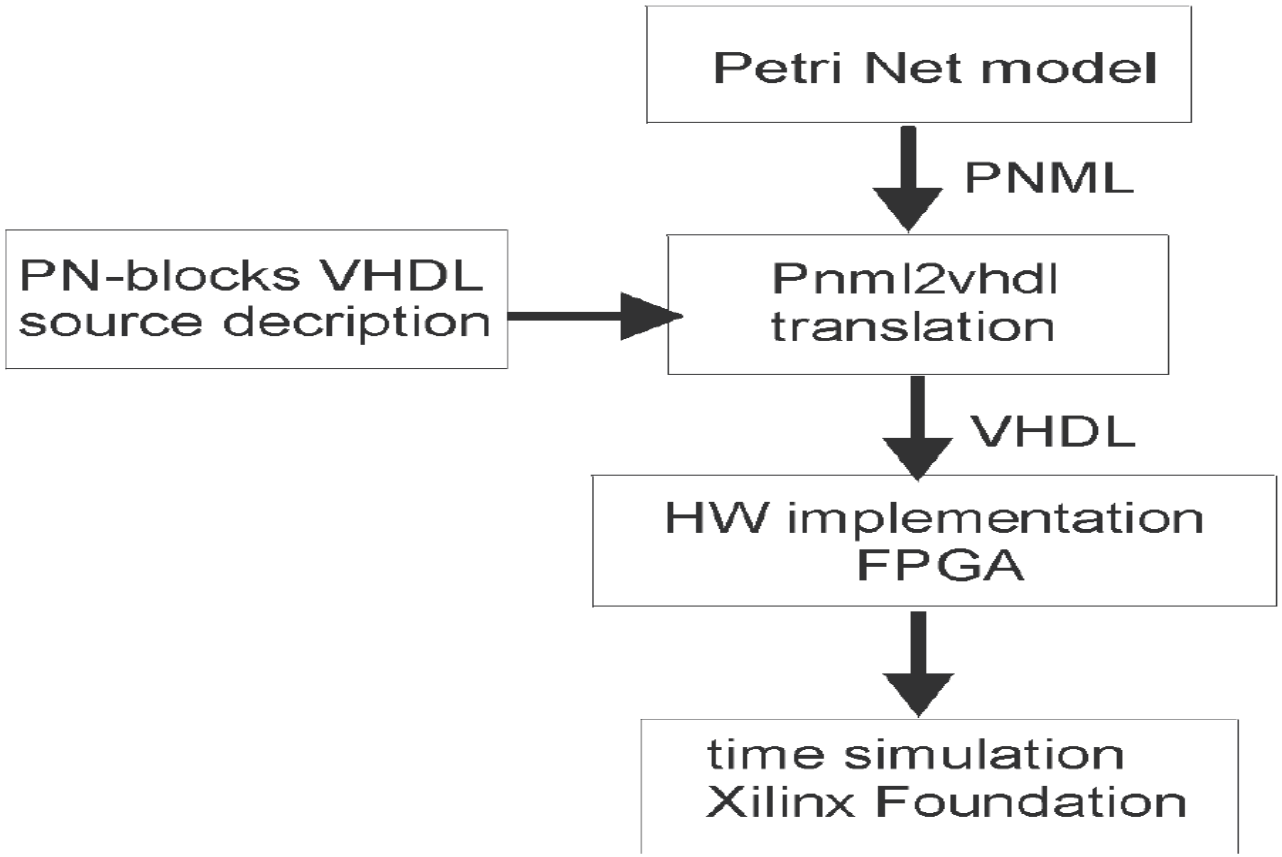
RdP : Graphes de Reachabilité



RdP : Factorisation



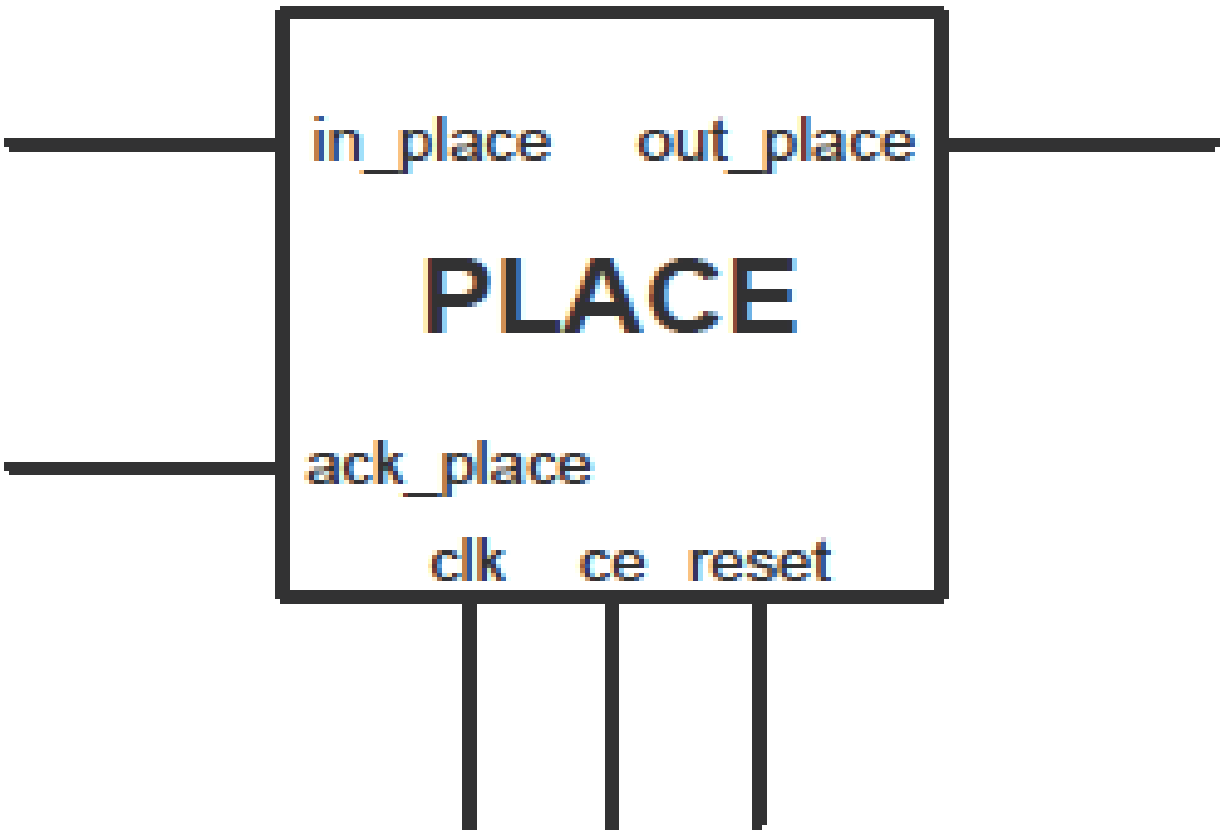
RdP : génération code VHDL (H. Kubatova – CTU Prague)



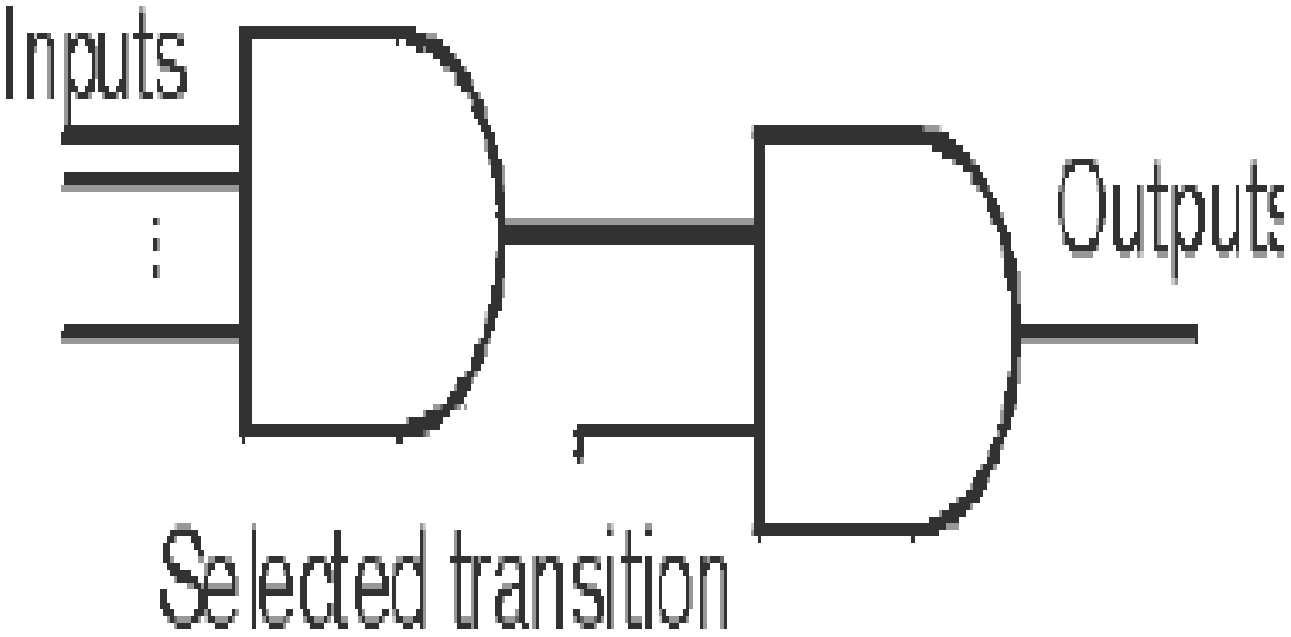
RdP : génération code VHDL (H. Kubatova – CTU Prague)

```
<place id="p1">
<graphics>
<position x="-20" y="10"/>
</graphics>
<name>
<value>ready to produce</value>
<graphics>
<offset x="0" y="0"/>
</graphics>
</name>
<initialMarking>
<value>1</value>
<graphics>
<offset x="0" y="0"/>
</graphics>
</initialMarking>
</place>
```

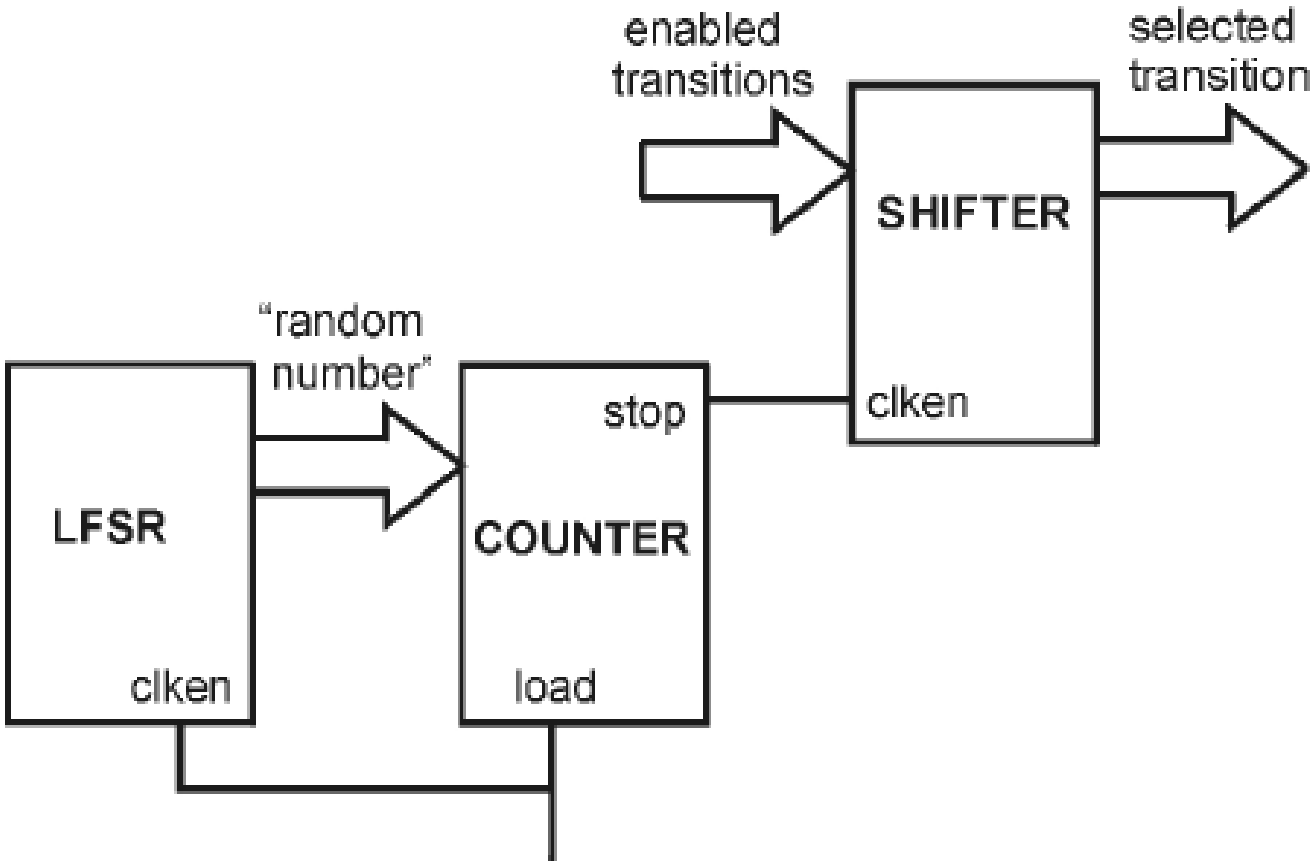
RdP : génération code VHDL (H. Kubatova – CTU Prague)



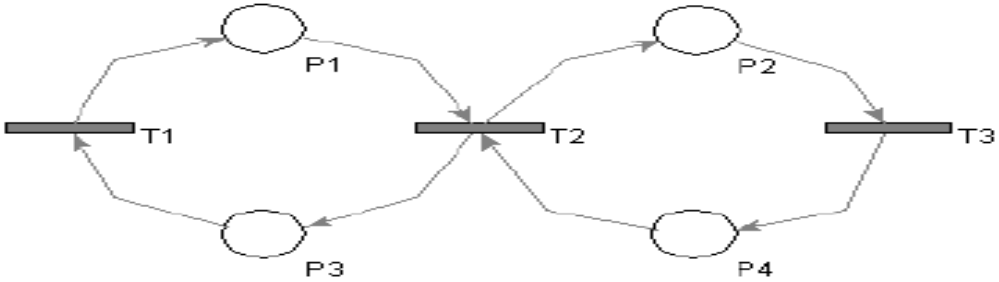
RdP : génération code VHDL (H. Kubatova – CTU Prague)

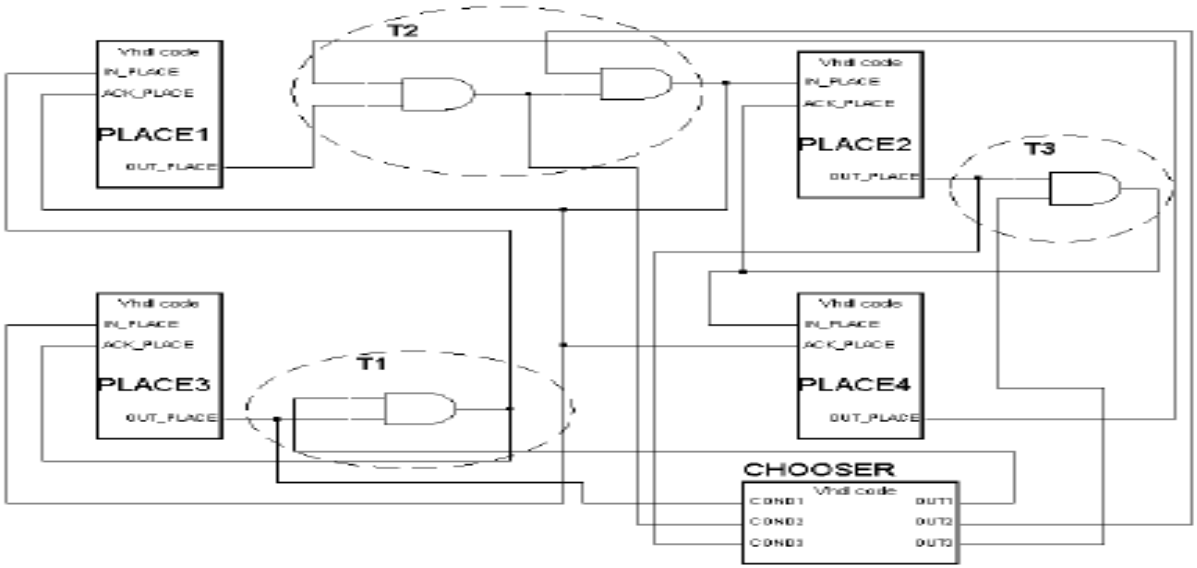


RdP : génération code VHDL (H. Kubatova – CTU Prague)

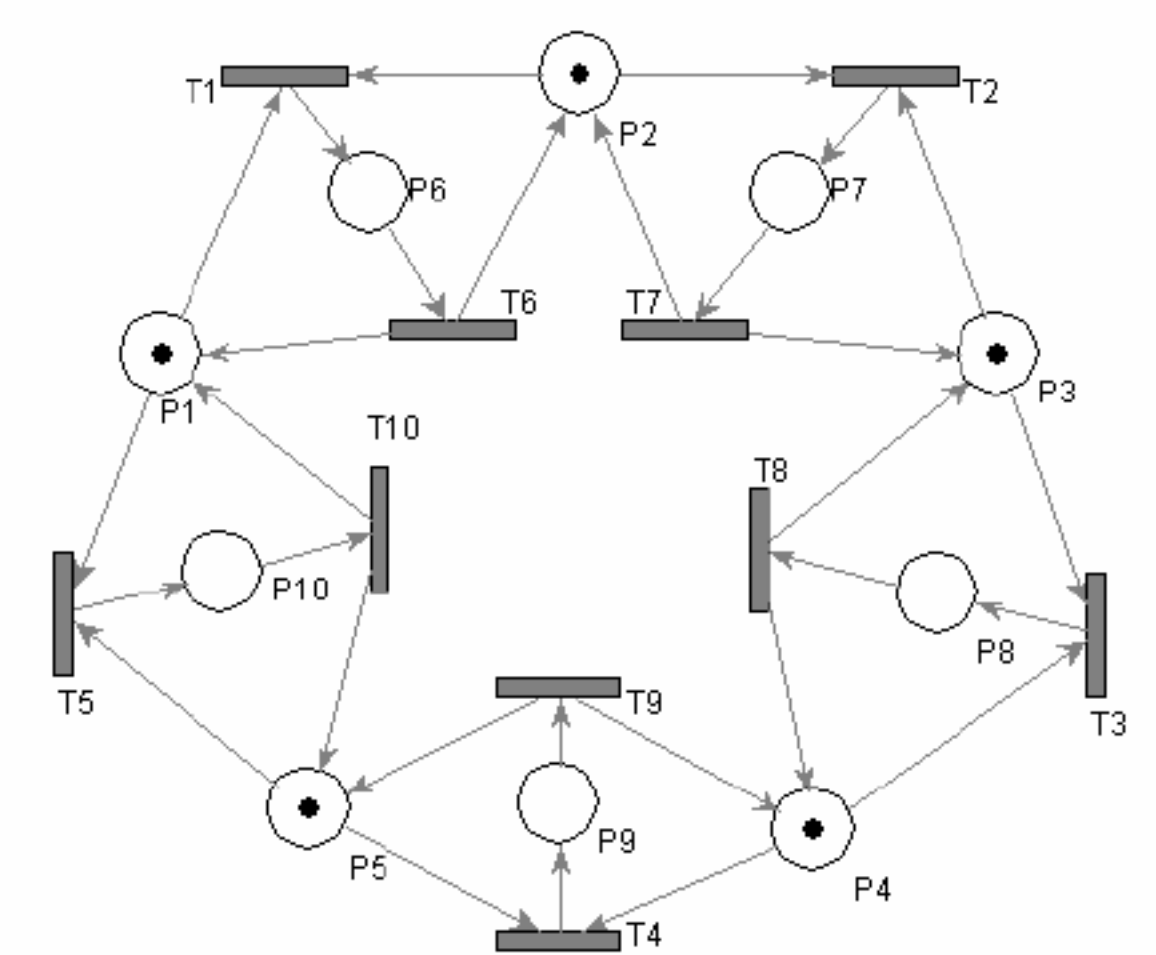


RdP : génération code VHDL (H. Kubatova – CTU Prague)

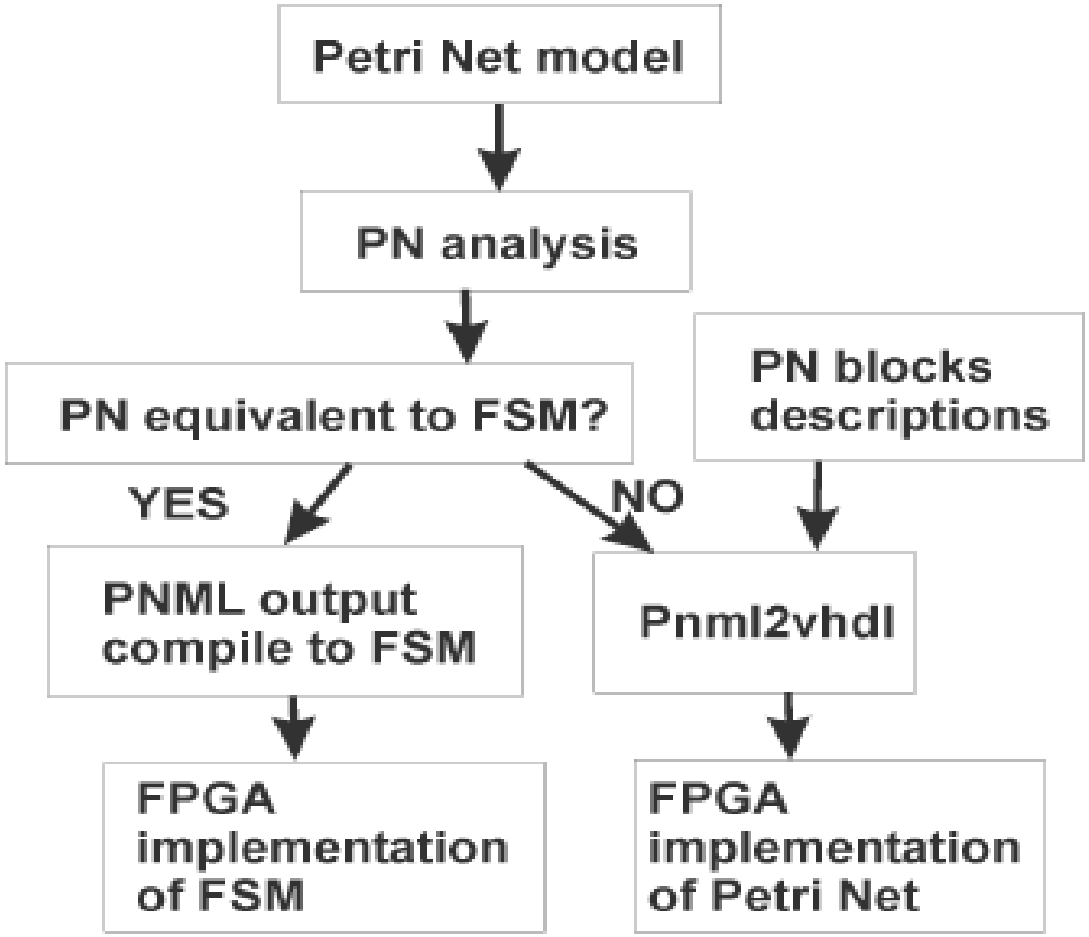




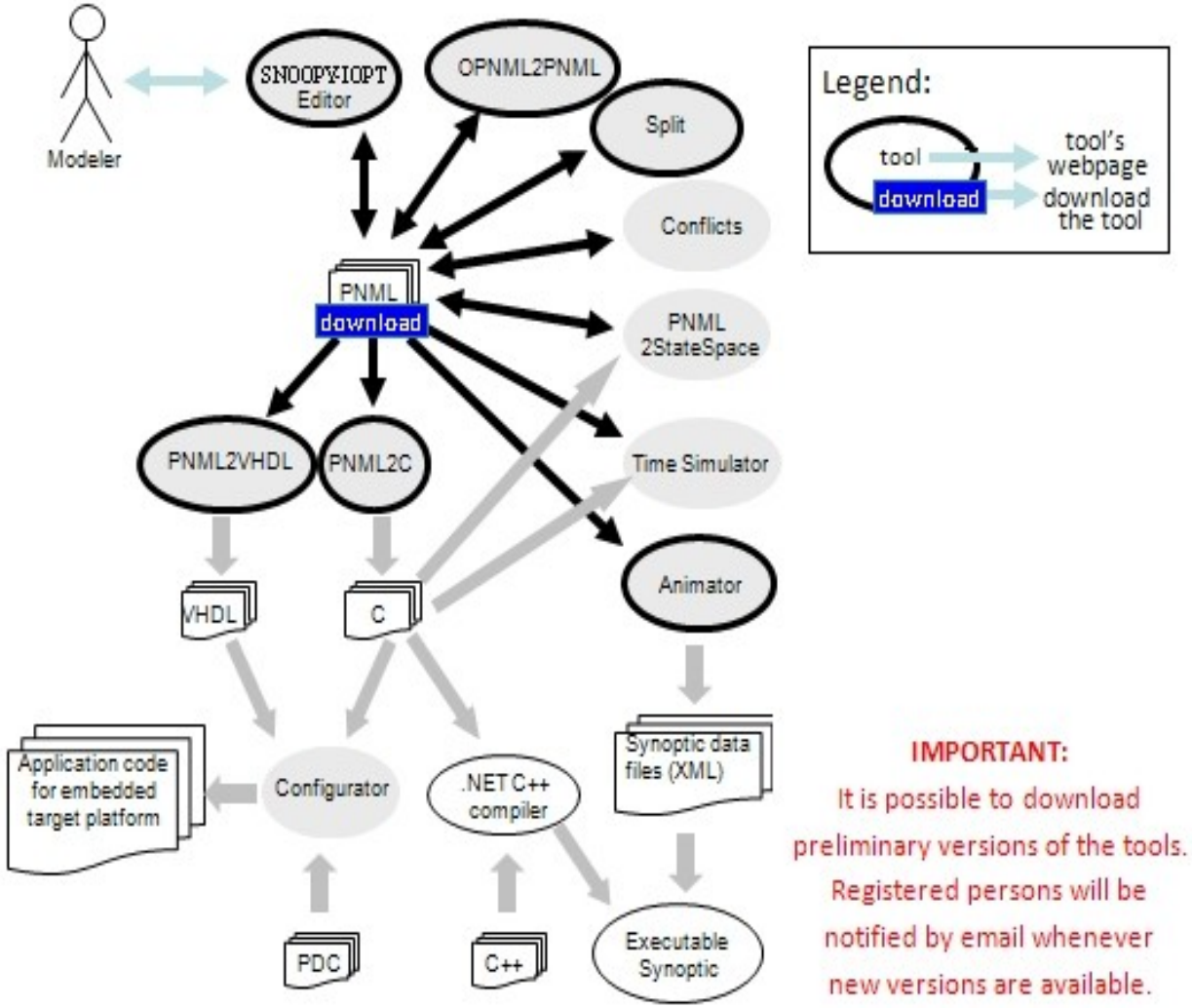
RdP : génération code VHDL



RdP : génération code VHDL



RdP : génération code VHDL (Projet FORDESING - Portugal)



IMPORTANT:

It is possible to download preliminary versions of the tools. Registered persons will be notified by email whenever new versions are available.

RdP : génération code VHDL