Conception et systèmes embarqués complexes

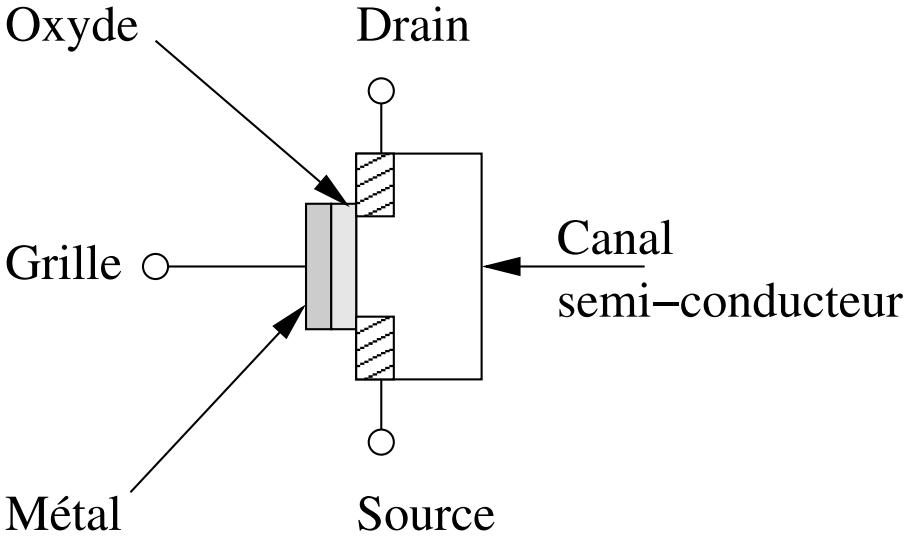
Master 2004

Antoine Fraboulet, Tanguy Risset
antoine.fraboulet@insa-lyon.fr, tanguy.risset@ens-lyon.fr
Lab CITI, INSA de Lyon, Lab LIP, ENS de Lyon

Conception de circuits

Le transistor

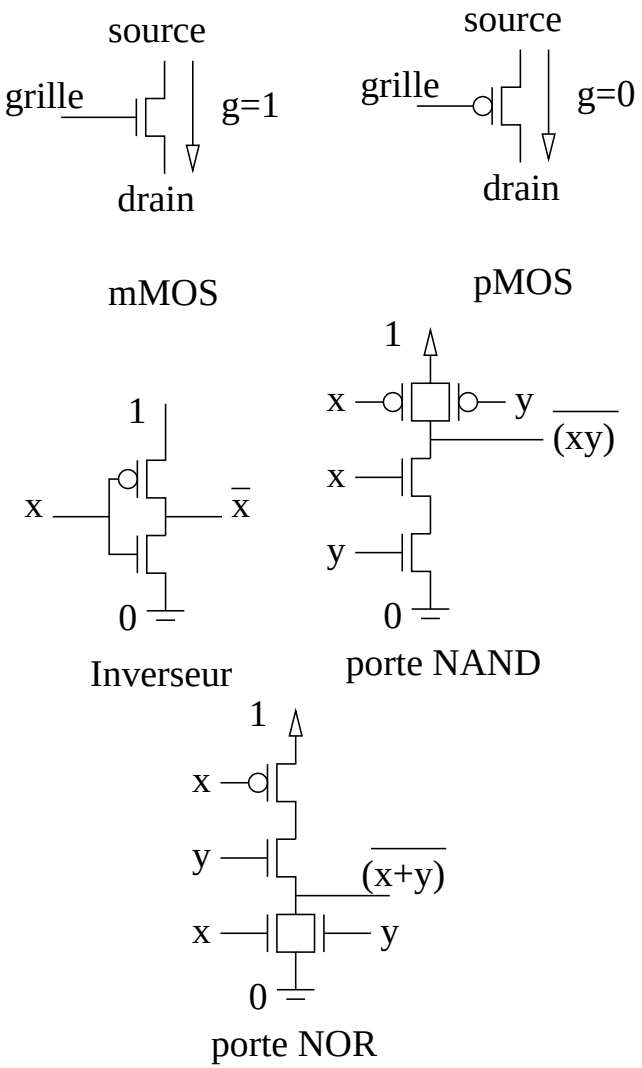
- Composant électronique de base
- Portes logiques ON/OFF



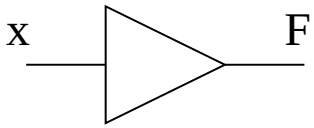
Technologie CMOS

- Complementary Metal Oxide Semiconductor
- Niveaux logiques : 0 = 0V et 1 = 3V
- Deux types de portes
  - nMOS : conducteur si la grille=1
  - pMOS : conducteur si la grille=0
- Réalisation de quelques portes de base

  Inverseur, NAND, NOR



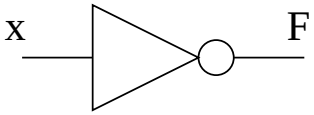
Portes élémentaires



Amplificateur:

$F = x$

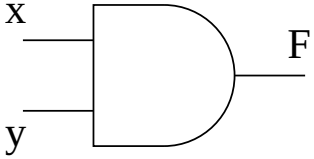
x	F
0	0
1	1



Inverseur:

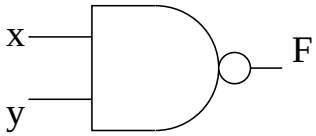
$F = \overline{x}$

x	F
0	1
1	0



ET: $F = x\ y$

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

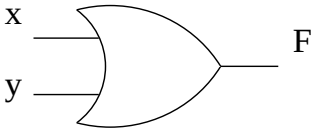


NON ET:

$F = \overline{(x\ y)}$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

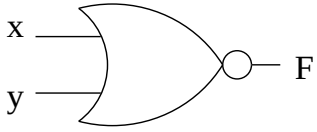
Portes élémentaires



OU:

$F = x + y$

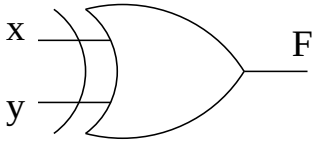
x	y	F
0	0	0
0	1	1
1	0	1
1	1	1



NOR:

$F = \overline{(x + y)}$

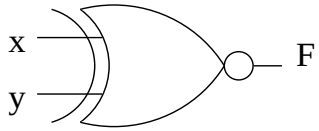
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0



XOR:

$F = x \oplus y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0



XNOR:

$F = x \odot y$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Conception de circuits combinatoires

1. Description du problème:

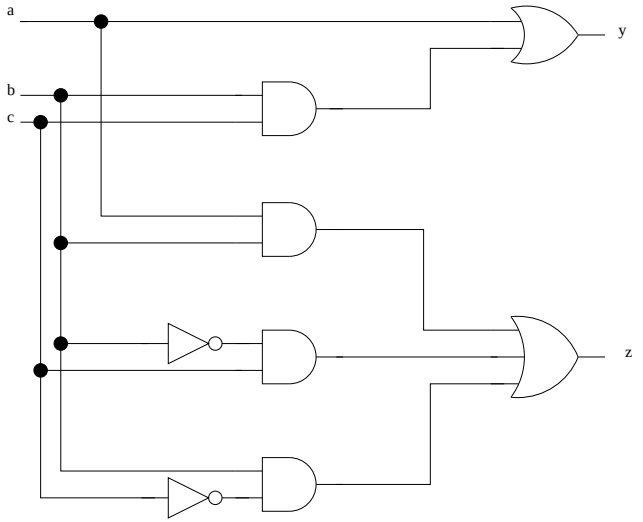
   y vaut 1 si a vaut 1 ou b et c valent 1.

   z vaut 1 si b ou c valent 1 (mais pas les deux) ou si a, b et c valent 1.
2. Table de vérité $\rightarrow$
3. Équations logiques

   $$y = \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c + ab\overline{c} + abc$$
   $$z = \overline{a}\overline{b}c + \overline{a}b\overline{c} + a\overline{b}c + ab\overline{c} + abc$$
4. Equations optimisées

   $$y = a + bc$$
   $$z = ab + \overline{b}c + b\overline{c}$$
5. Portes logiques $\rightarrow$

entrées			sorties	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

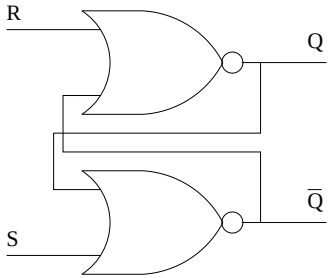


Composants combinatoires fréquents

- Multiplexeur à n entrées
- Décodeur $log(n) \rightarrow n$
- Additionneur n bits
- Comparateur n bits
- ALU n bits
- etc.

Logique séquentielle

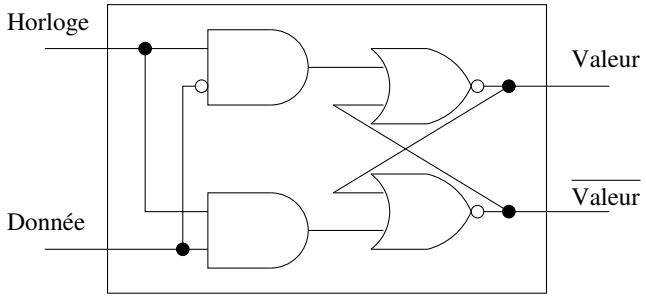
- Bascule RS: lorsque R et S passe à 0, Q et $\overline{Q}$ conservent leur valeur antérieure.



Bascule RS

S	R	Q	$\overline{Q}$
0	1	0	1
1	1	interdit	interdit
1	0	1	0
0	0	Q_{n-1}	$\overline{Q_{n-1}}$

- Bascule D: permet d'échantillonner la valeur de la donnée lorsque l'horloge passe à 1 (front montant de l'horloge) et de conserver cette valeur lorsque l'horloge passe à 0.



Composants séquentiels fréquents

- Registre n bits (avec $reset$ et eventuellement $load/ClockEnable$)
- Registre à décalage n bits
- Compteur n bits
- Machine à états

Conception de circuits séquentiels

- Extrêmement complexe en général.
- De nombreux modèles de calculs:
  - Séquentiel
    - Machine à états
    - Contrôleur + chemin de données
  - Parallélisme de tâches
    - Processus communicants
  - Parallélisme de données
  - Calcul sur des flots de données
  - Circuits multi-horloge
  - Paradigme synchrone
  - Circuits assynchrone
- Notion sous-jacente très utilisée: automate à états finit

Le “Paradigme” Ptolemy

- **Ptolemy** (http://ptolemy.berkeley.edu/) est un outil de modélisation et de conception de systèmes concurrents développé par l’université de Berkeley.
- **Ptolemy** permet d’avoir une vision globale d’un système et de concevoir différentes parties selon différents flots de conception.
- Vise particulièrement les systèmes enfouis, les systèmes hétérogènes (analogique/digital, calcul/contrôle)
- Ed Lee affirme que “La clé du succès de Ptolemy est l’utilisation de *modèles de calculs* parfaitement définis pour gouverner les intéractions entre les composants”

Modélisation

- La modélisation est l’acte de représenter un système (ou un sous-système) formellement.
- Il existe beaucoup de formes de modèles:
  - Modèle mathématique, ensemble d’assertion portant sur les propriétés du système (fonctionnalité, dimensions physique, etc.).
  - Modèle constructif/exécutable: définit une procédure qui mime un ensemble de propriétés du système (utilisé pour décrire les réactions aux stimulis externes)

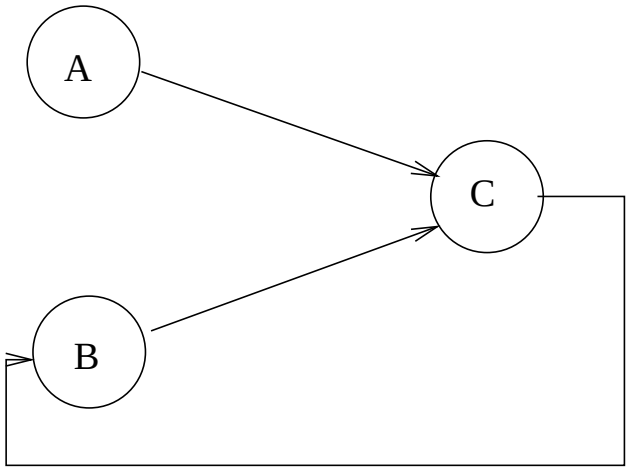
Conception

- La **conception** est l’acte de réaliser un système (ou un sous-système). Cela nécessite:
  - de définir un ou plusieurs modèles du système ;
  - de raffiner ces modèles jusqu’à obtenir les fonctionnalités désirées.
- Conception et modélisation sont fortement couplées.
- la **simulation** (exécution du modèle exécutable) est un terme approprié lorsque le modèle est clairement distinct du système. Pour les systèmes enfouis, cette distinction diminue au fur et à mesure des raffinements (simulation $\longrightarrow$ implémentation logicielle).

Modèle de Calcul

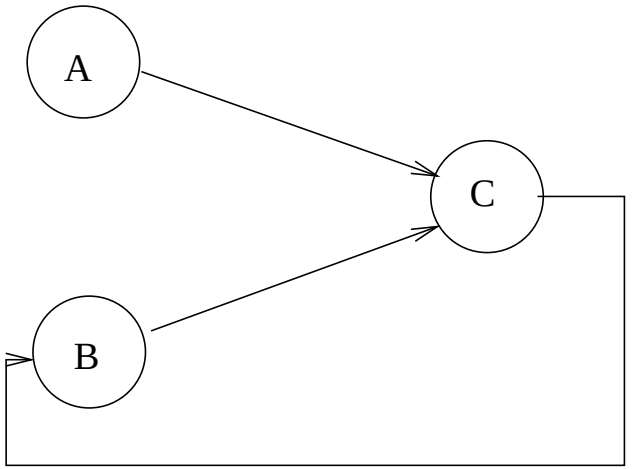
- Le modèle exécutable est construit à partir d’un **modèle de calcul**: ensemble de règles (formant la sémantique) qui permettent au concepteur de concevoir le modèle.
- Le choix du modèle de calcul dépend du système à implémenter:
  - Traitement purement calculatoire sur des données: sémantique impérative (C, Java, Matlab, ...).
  - Système mécanique: parallélisme et temps continu (Simulink, VHDL-AMS)
- Le choix du modèle de calcul affecte fortement la qualité de conception d’un système.
- pour les systèmes enfouis: les concepts de *parallélisme* et de *temps* doivent être pris en compte.

Différentes modèles de calcul



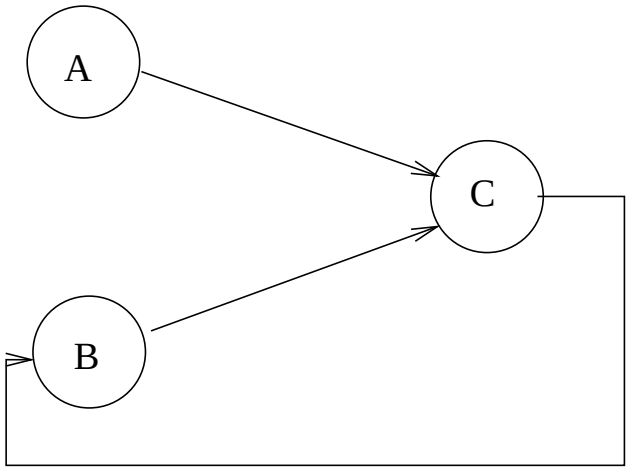
- Une représentation graphique
- plusieurs sémantiques associés dépendant du modèle de calcul choisi (les *domaines* dans Ptolemy)

Équations différentielles



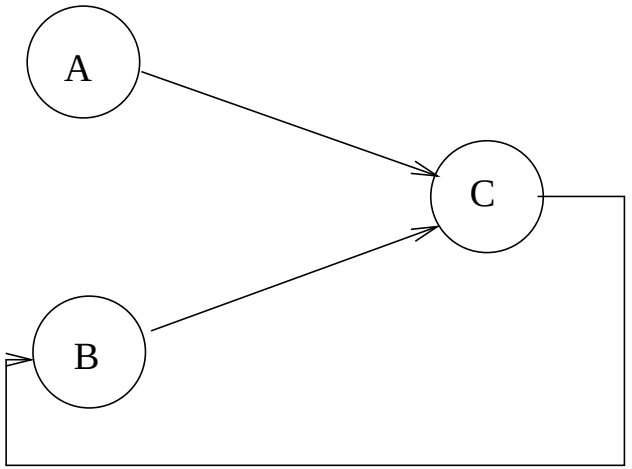
- Arc=fonction continue, sommet=relation entre ces fonctions.
- Simulation=point fixe du système.
- Modélisation de circuits analogiques, systèmes physiques.
- Problèmes: proche de l’implémentation, simulation complexe, interaction avec des systèmes d’évènement discrets.
- Exemple: Simulink, VHDL-AMS, . . .

Machines à états finis (FSM)



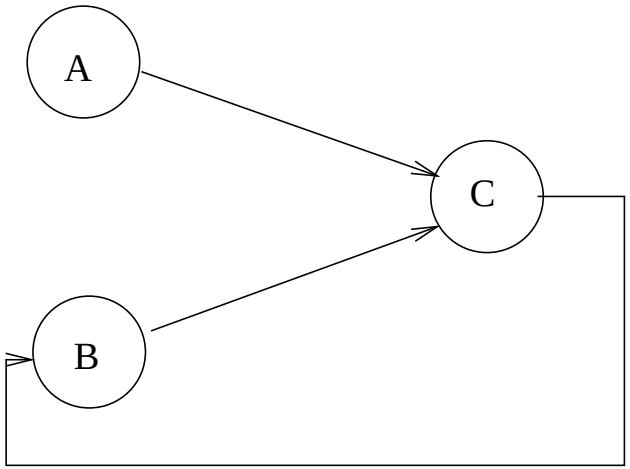
- Sommet=état, arc=transition.
- Adapté au contrôle des systèmes enfouis (en particulier les systèmes critiques).
- Réalisation en matériel ou logiciel.
- Problèmes: modèle pauvre, nombre d’états.
- Exemples: *StateChart*

Modèles flot de données



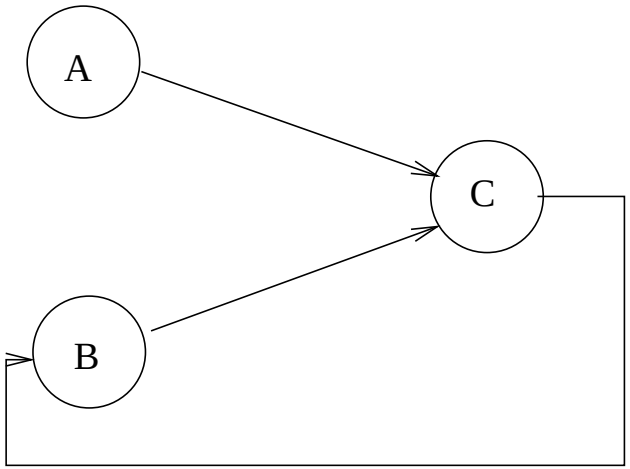
- Arcs=données cadencées par des horloges, Sommets=relations entre les données.
- Adapté au contrôle complexe des systèmes enfouis.
- Réalisation en matériel ou logiciel.
- Problèmes: synchronisation globale nécessaire (systèmes sur-spécifiés).
- Exemples: Les modèles flot de données synchrone (SDF de Ptolemy, langages Signal, Esterel, Lustre, équations récurrentes).

Modèles à évènements discrets



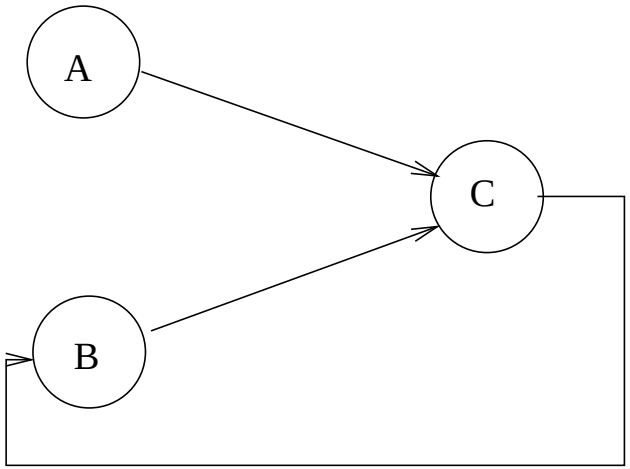
- arcs=évènements placés dans le temps (action et date).
- Spécification de systèmes matériels parallèles et simulation de systèmes de télécommunication.
- Problèmes: notion globale de temps (gros chips), simulation lente.
- Exemples: Les simulateurs VHDL et Verilog.

Modèles à passage de messages synchrone



- Des processus communiquent lors d’actions atomiques, instantanées (rendez-vous: les envois sont bloquants).
- Problèmes de partage de ressources (client-server, multiplexage de ressources matériels).
- Problèmes: ne modélise que des systèmes déterministes.
- Exemples: langage à base de CSP ou CCS (Lotos, Occam).

Modèles à passage de messages assynchrone



- Les envois sont non bloquant (tampons). Arc=séquences de valeurs, Sommets=fonctions entre entrées et sorties.
- Modélisation d’algorithmes de traitement du signal parallèles
- Problèmes: modélise mal le contrôle logique.
- Exemples: Processus de Kahn, Process Network de Ptolemy.

Paradigme Ptolemy

- Quel modèle de calcul utiliser?
- Un modèle hybride (mélange de plusieurs modèle):
  - Le modèle devient trop complexe.
- Émuler tous les modèles par un seul:
  - certaines parties du systèmes sont traitées inefficacement à cause des carences du modèle choisi.

$\longrightarrow$ le “paradigme” Ptolemy:

- **utiliser différents modèles pour différentes parties d’un système et spécifier précisément leurs interfaces.**

VHDL

Qu’est ce que VHDL?

- VHSIC (Very High Speed Integrated Circuit) Hardware Description Langage.
- Langage pour décrire la structure *et* le comportement de systèmes électroniques, en particulier des circuits digitaux (ASIC, FPGA, . . .).
- Standard IEEE.
- Indépendant de la technologie cible.
- Indépendant de la méthodologie de conception.
- Indépendant des outils de conception.
- Langage très général $\rightarrow$ très complexe ($\rightarrow$ dépendent de tout!)
- VHDL **n’est pas un langage de programmation**

  c’est un langage de description (specification) de système.

Historique

- 1980: le département de défense américain lance un appel d'offre pour un langage qui permettrait de décrire tous les systèmes électroniques utilisés. Motivation affichée: réutilisabilité et réduction des coûts de conception.
- 1983 trois compagnies (Intermetics, IBM, Texas Instruments) commencent le développement.
- 1985: première version officielle de VHDL (version 7.2).
- 1986: VHDL est donné à IEEE pour en faire un standard.
- 1987: Standard IEEE 1076-1987.
- 1993: Standard IEEE 1076-1993.
- 1999: Standard IEEE 1076.6-1999

niveaux description VHDL

- Description structurelle.
  - Proche des schématiques traditionnelles des concepteurs
  - Blocs inter-connectés.
- Description comportementale.
  - Proche de la programmation traditionnelle
  - exécution séquentielle d'instructions
- Cette classification ne correspond pas exactement à notre décomposition intuitive: description algorithmique/description architecturale

VHDL de niveau transfert de registre

- Une description de niveau transfert de registre est une description de la structure du circuit, mais qui abstrait les opérateurs.
- On peut parler de "registre" (sans dire exactement quel type), d'additionneur, etc.
- N'est pas forcément du VHDL structurel.
- Ce sous ensemble de VHDL doit pouvoir être synthétisé par les outils commerciaux (Synopsis, Cadence, etc.) selon une *sémantique définie par le standard*.

$\longrightarrow$ C'est une plate forme solide pour la synthèse haut niveau.

Principe de la simulation événementielle

- Le système est représenté comme un ensemble de **processus** (*process*) qui s'exécutent en parallèle
- On veut simuler des processus parallèles sur une machine séquentielle.
- Nouvel objet pour communiquer entre programmes séquentiels: le signal
- Moteur de simulation:
  - Le simulateur exécute tous les processus dans un ordre quelconque.
  - Lorsqu'un signal partagé par plusieurs processus est modifié, on enregistre sa nouvelle valeur, le signal conservant temporairement sa valeur.
  - Lorsque tous les processus ont été exécutés, on modifie les valeurs des signaux partagés
  - on incrément le temps symbolique ($+1\Delta$) et on recommence jusqu'à convergence

Le temps symbolique Δ

- Le temps symbolique permet d'ordonner des événements simultanés à partir de leurs dépendances.
- Une affectation à un signal: $Sig_1 <= Sig_2$ est instantanée mais la valeur de Sig_1 est modifiée après 1 Δ
- Un "événement" possède donc une date complète composée d'une date physique (ex: 1h 04m 17s) et d'une date symbolique (ex: 4Δ)

Signal et variable

- Du fait de sa sémantique spécifique, le signal se comporte très différemment d'une variable
- On peut en avoir une vision intuitive comme representant la "valeur d'un fil physique au cours du temps"
- Pour qu'un signal conserve une valeur d'un cycle à un autre il faut mettre en place explicitement un mécanisme de mémorisation qui sera interprété comme un registre.
- Pour faciliter la simulation on peut aussi introduire des variables dans les processus.
- Les variables sont locales aux processus, leur affectation est instantanée, elles conservent leur valeur au cours du temps comme dans un langage de programmation

$$Sig_1 <= Sig_2;$$
$$Sig_3 <= 3;$$

$$Var_1 := Var_2;$$
$$Var_3 := 3;$$

Le temps physique

- Le temps physique permet de simuler l'écoulement du temps réel
- Il ne peut avancer qu'avec les instruction `wait` et `after` (points de synchronisation, point d'arrêt):

```
wait 10 ns
Sig1 <= 25 after 100 s

wait until rising_edge(Clk)
```

- Entre deux points de synchronisation, le temps physique n'avance pas (l'affectation des signaux est différée).
- Le processus est (implicitement) une boucle infinie avec au moins un point d'arrêt.

```
process ...
  ...
  begin
  S <= A+B;
  wait on A;
  R:=S;
  A<= R;
  B<= A;
  wait until rising_edge(Clk)
  if (R > 1023) then
    Counter <= 127
  else
    COUNTER <=Counter-1;
  end if;
  wait until rising_edge(Clk)
  ISO <=COUNTER * 7
end process;
```

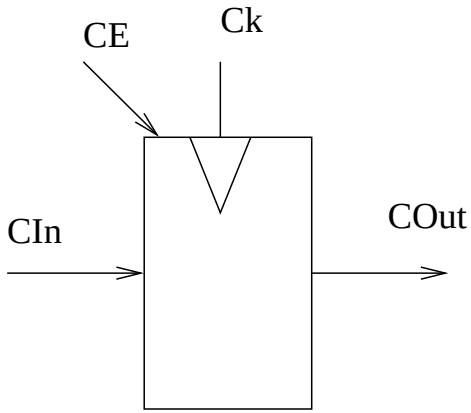
Principe du moteur de simulation

- Chaque processus possède une liste de sensibilité qui indique si il doit être "réveillé" ou pas.
  Changement d'une valeur d'un signal utilisé
  Changement du temps physique
- Le moteur effectue de manière répétitive les tâches suivantes:
  1. Choisit la date courante
  2. Positionne les signaux à leurs nouvelles valeurs
  3. Pour chaque processus
     teste la condition de reprise du point d'arrêt
     Si elle est vérifiée, le processus est réveillé et exécuté jusqu'au prochain point d'arret

Exemple: un registre

- Un registre avec clock enable.
- Le processus est réveillé lors d'une transition sur `Ck`
- Les `If` internes permettent d'exprimer la condition: front montant de l'horloge avec `CE` à 1.

```
PROCESS(ck)
BEGIN
IF (ck = '1' AND ck'EVENT)
THEN
    IF CE='1' THEN Out <= In;
    END IF;
END IF;
END PROCESS;
```



Structure du langage

- Il existe "5 unités de compilation" permettant de décrire des composants.
  - L'**entité**: description de l'interface du composant: le nom et ses ports d'entrée/sortie
  - L'**architecture** décrit l'intérieur du composant. Il peut y avoir plusieurs architectures pour le même composant (ex: une pour la simulation efficace, une pour la synthèse). L'architecture contient les processus.
  - La **déclaration de paquetage**. Un paquetage est une collection d'objets réutilisables (constantes, types, composants, procédures)
  - Le **corps de paquetage**
  - La **configuration** indiquant quelle architecture utiliser pour chaque entité

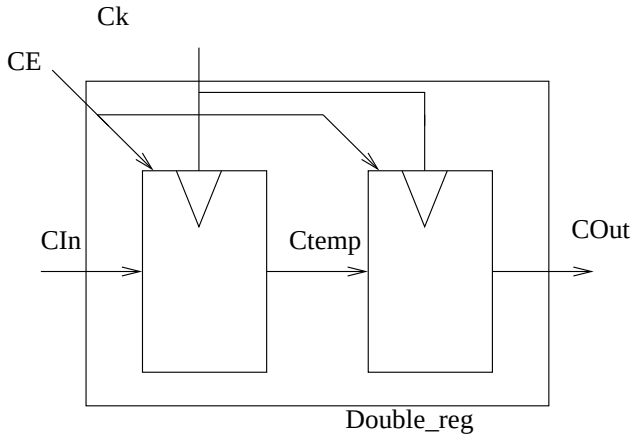
Exemple: double registre (declaration)


```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY DOUBLE_REG IS
    port(CIn, Ck, CE: IN STD_LOGIC;
         COut: out STD_LOGIC);
END ENTITY DOUBLE_REG;

ARCHITECTURE behavioural OF DOUBLE_REG IS
    -- Declaration de signaux du composant

BEGIN
    PROCESS(ck)
       Variable CTemp: STD_LOGIC;
        -- Declaration de signaux
        -- ou variables du process
    BEGIN
    IF (ck = '1' AND ck'EVENT)
    THEN
        IF CE='1' THEN
            COut <= CTemp;
            CTemp := CIn;
        END IF;
    END IF;
    END PROCESS;
END behavioural;
```



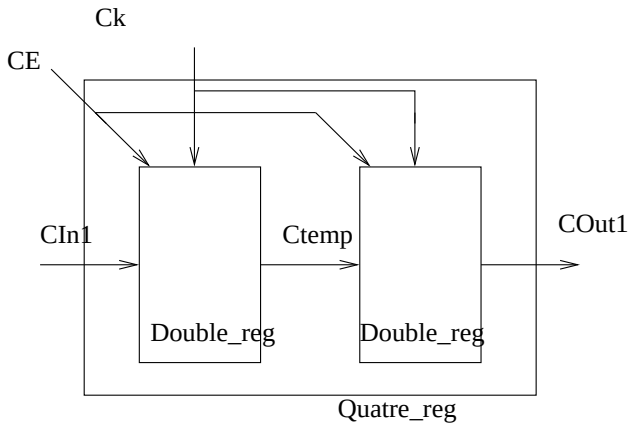
Exemple: double registre (instanciation)


```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY QUATRE_REG IS
    port(CIn1, Ck, CE: IN STD_LOGIC;
         COut1: out STD_LOGIC);
END ENTITY QUATRE_REG;

ARCHITECTURE behavioural OF QUATRE_REG IS
       Signal CTemp: STD_LOGIC;
       COMPONENT DOUBLE_REG IS
            port(CIn, Ck, CE: IN STD_LOGIC;
            COut: out STD_LOGIC);
        END COMPONENT;
    Begin
    I0: DOUBLE_REG port map (Cin => Cin1,
                             Ck  => Ck,
                             CE  => CE,
                             Cout => CTemp);
    I1: DOUBLE_REG port map (Cin => CTemp,
                             Ck  => Ck,
                             CE  => CE,
                             Cout => COut1);

END behavioural;
```



Les types

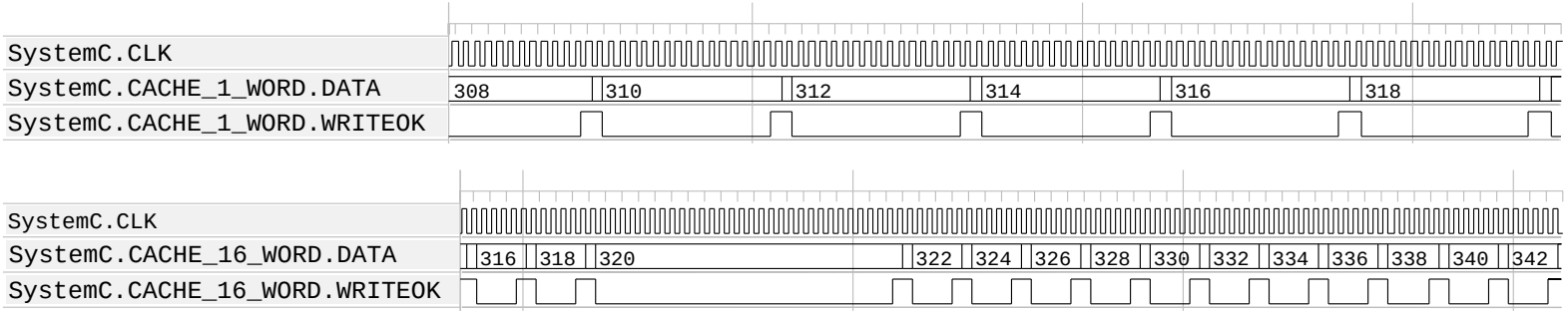
- Les librairies IEEE définissent des types de base pour
  - Les valeurs booléennes (`IEEE.std_logic_1164.all;`)
  - Les entiers, les chaînes de caractères
  - Il vaut mieux utiliser les `STD_LOGIC_VECTOR` pour la synthèse.
- Les type `STD_LOGIC` est "resolu", il peut prendre les valeurs suivantes:
  - `'U', - uninitialized`
  - `'X', - Uknown`
  - `'0', - 0`
  - `'1', - 1`
  - `'Z', - High Impedance`
  - `'W', - Weak Uknown`
  - `'L', - Weak '0'`
  - `'H', - Weak '1'`
  - `'-', - Don't care`
- Si le signal est piloté par plusieurs processus, la fonction de résolution va décider quelle sera sa valeur

Compilation de Vhdl

- Pour être simulé, un programme VHDL est
  - Compilé
  - "Élaboré" ($\simeq$édition de liens). Lors de l'élaboration les opérations élémentaires utilisées sont implémentées par des librairies fournies par l'outil de simulation, leur implémentation peut poser problème.
  - Simulé. Pour cela il nécessite un "test bench" qui fournit les entrées et l'horloge au composant simulé (stimuli).
- Si il est écrit de manière *synthétisable*, il peut être:
  - synthétisé (synthèse logique)
  - simulé après synthèse
  - placé (placement-routage)
  - simulé après placement routage.
- Une description destinée à la simulation efficace d'un circuit est très différente d'une description destinée à la réalisation. VHDL synthétisable propose un compromis entre les deux.

Conception en VHDL

- Le mécanisme du temps symbolique rend extrêmement délicat le débugage de VHDL
- On utilise les *waveforms* qui ne permettent pas de voir les transitions de granularité Δ.



- VHDL permet énormément de constructions difficiles à comprendre (circuits asynchrones, simulation, synthèse, etc...).
- Aujourd'hui il existe des langages plus rapides pour la simulation (SystemC, etc.), VHDL est donc essentiellement utilisé pour la synthèse et la simulation bas niveau.
- $\Rightarrow$ **Se contraindre fortement lors de l'écriture de programmes VHDL**

Contraintes pour l'écriture de VHDL

- Penser *composant* (objet) plutôt que fonctionnalité (procédure).
- Distinguer les éléments de mémorisation (mémoire, bancs de registre) et les composants standards (opérations arithmétiques, filtres numériques) pour prévoir l'utilisation de bibliothèques.
- Décomposer les composants sous forme d'automate à états finis
- On peut alors soit coder directement l'automate en VHDL soit décomposer à nouveau cet automate en un contrôleur et un chemin de donnée
- Pour les traitements hautement pipelinés (*stream processing*), on utilisera plutôt une description des différents étages de pipeline interconnectés.

Codage d'un automate en VHDL (exemple)

- N'utiliser que des signaux (pas de variables). Que des types simples (`STD_LOGIC_*`), de préférence non resolu.
- Décomposer le matériel en l'état (tous les éléments mémorisés entre deux cycle d'horloge) et les autres signaux.
- Écrire un processus sensible à l'horloge qui met à jour les registres d'état
- C'est la fonction de **transition** de l'automate.

```
PROCESS(clk,reset)
     BEGIN
       IF clk = '1' AND clk'event THEN
         IF reset = '1' THEN
           state           <= INIT;
         ELSE
           CASE state IS
             WHEN INIT => IF start = '1' THEN
                            state <= START_S;
                          else
                            state <=INIT;
                          END IF;
             WHEN START_S => IF start = '0' THEN
                               state <= EXEC_INIT_PHASE;
                             else
                               if (NBDataRdy_Sig='1' and
                                      NBDataAck='1') then
\                                 state <= START_S_2;
                               else
                                 state <= START_S;
                               end if;
                             END IF;
            ................
            ................
             when OTHERS => state <= ERROR;
           END CASE;
         END IF;
       end if;
     END PROCESS;
```

Codage d'un automate en VHDL (suite)

- Reste la fonction de **génération** de l'automate (calcul des sorties en fonction des entrées et de l'état).
- Écrire un processus sensible aux entrées et à l'état qui calcule les sorties.

```
S1 <= a + b * C;
S2 <= a - b;

PROCESS(S1,S2,state)
     BEGIN
         IF state = 'INIT' THEN
            Out1 <= S1;
         ELSE
            Out1 <= S2;
         END IF;
     END;
```

- La manière la plus propre:

  Décrire le data-path comme des affectations entre signaux en dehors des processus (inférence systématique du matériel pour le calcul sur les données)

  Décrire l'affectation des sorties comme une série de if imbriqués contenant une seule affectation de signal simple par corps (toutes les branches doivent contenir une affectation au signal de sortie: inférence des multiplexeurs de contrôle).

Synthèse de VHDL

- La plaie: l'inférence accidentelle de registres.
- dès qu'un signal n'est pas affecté dans tous les chemin de contrôle.

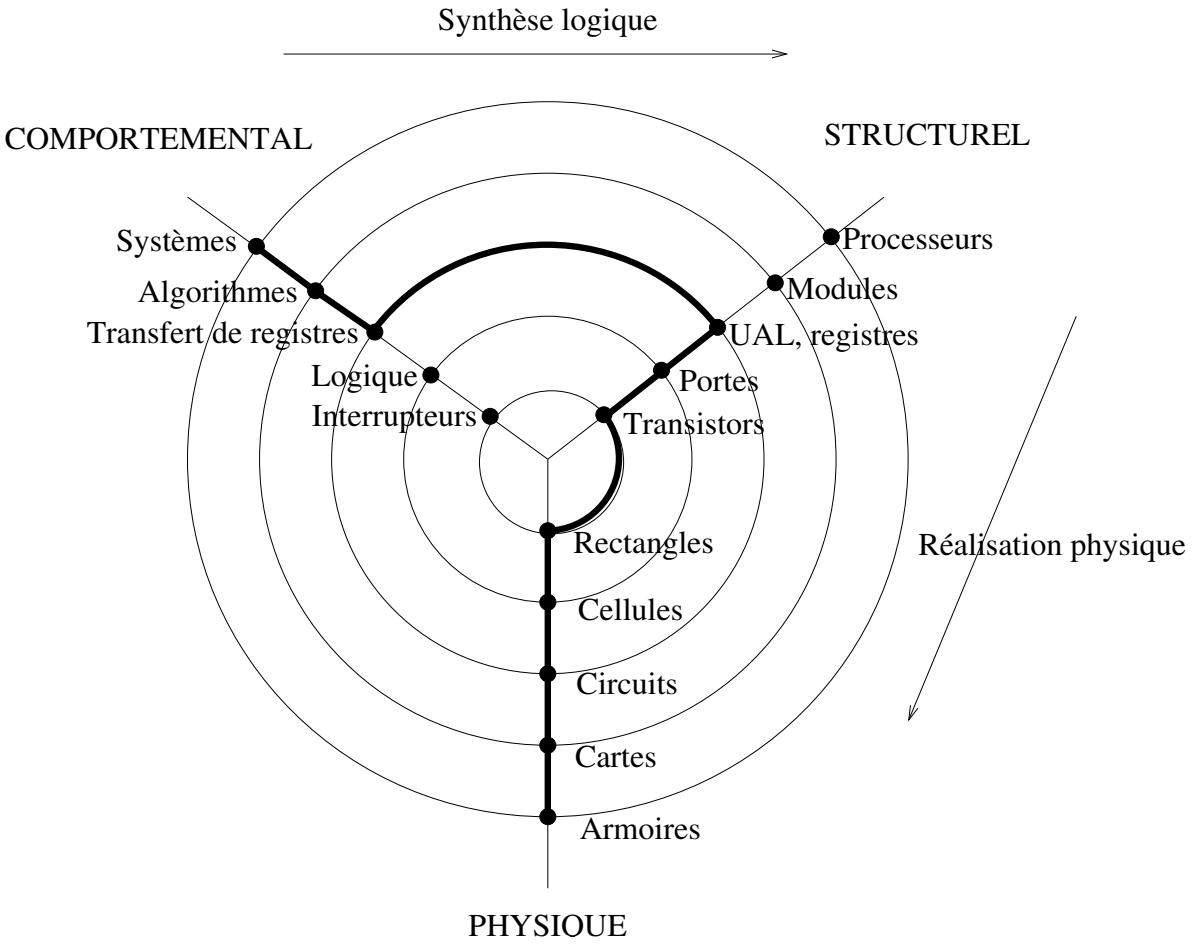
```
PROCESS(S1,state)
    BEGIN
        IF state = 'INIT' THEN
           Out1 <= S1;
        END IF;
    END;
```

```
PROCESS(S1,state)
    BEGIN
        IF state = 'INIT' THEN
           Out1 <= S1;
        ELSE
           Out1 <= '0';
        END IF;
    END;
```

$\Rightarrow$ Compter les registres

- Boucle à contrôle constant acceptée (déroulées à la compilation).
- Clause `Wait` acceptée uniquement pour l'horloge.
- En principe, il existe un standard IEEE pour le VHDL synthétisable. En pratique, chaque outil synthetise des sous-ensembles légèrement différents.
- Principaux outils de synthèse pour les Asics: Mentor Graphics, Synopsys, Cadence.

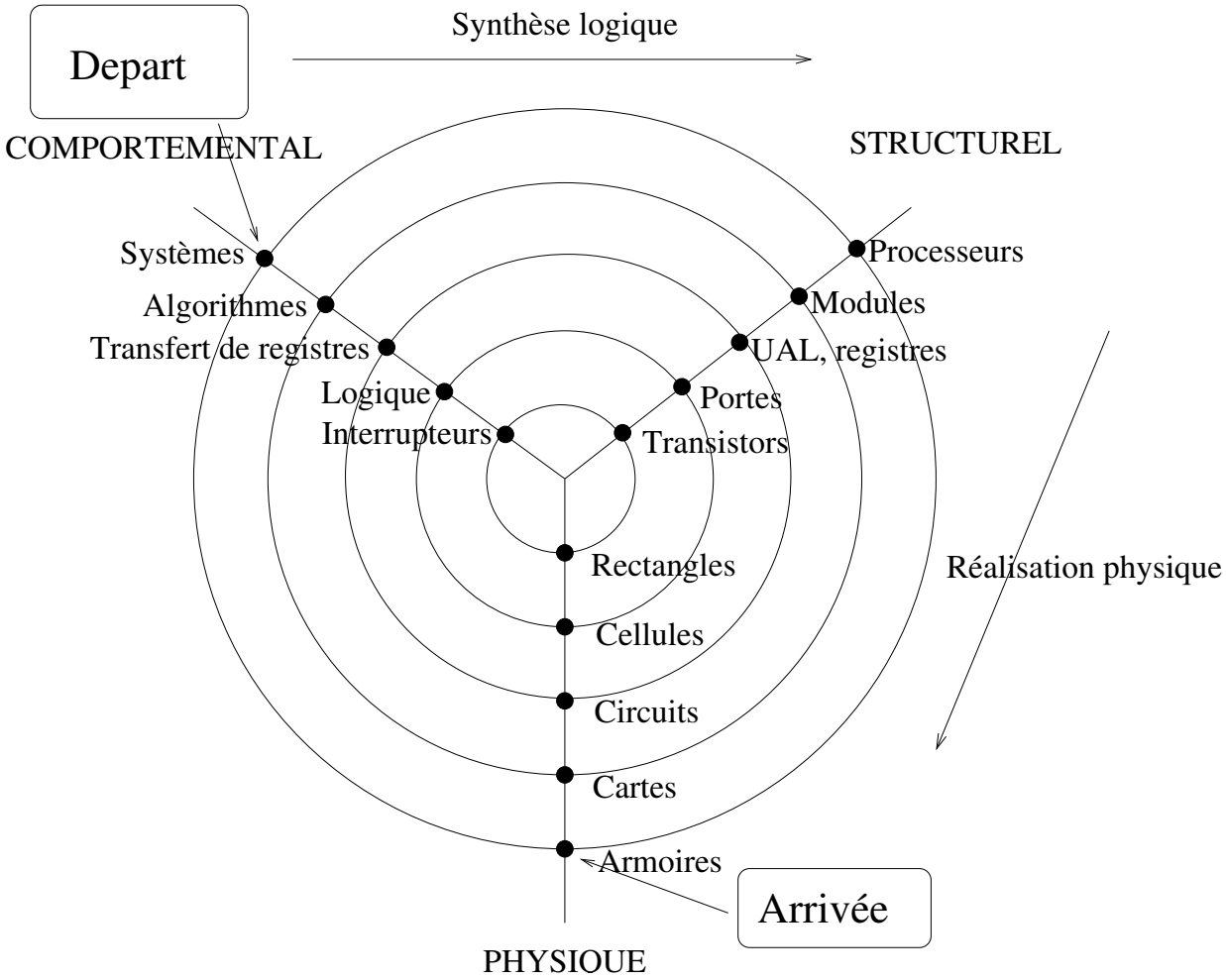
Synthèse de haut niveau



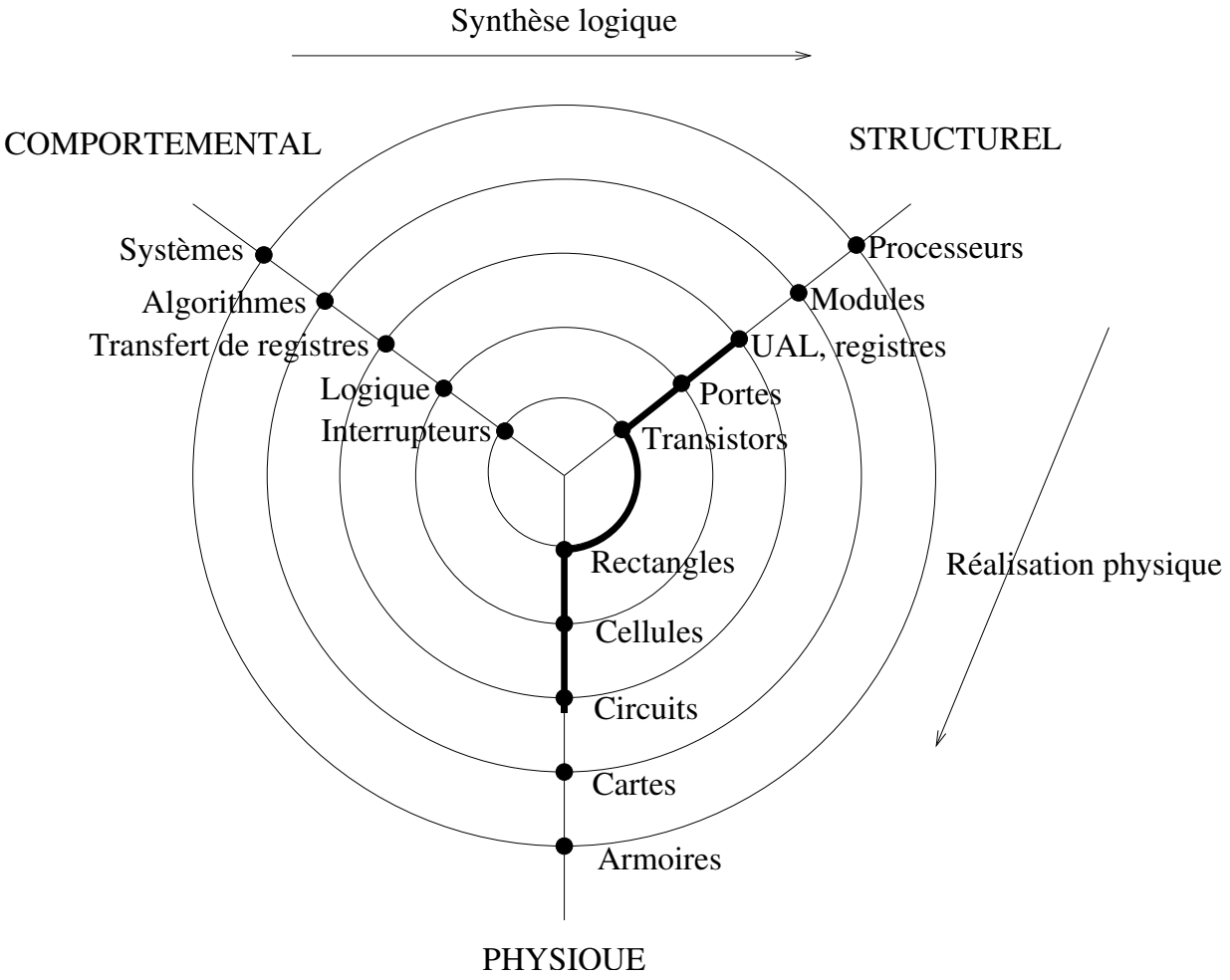
- Aujourd’hui : synthèse de VHDL structurel.
- Demain : synthèse de haut niveau: C, Matlab,VHDL $\rightarrow$ VHDL synthétisable.

Synthèse de haut niveau

Quel chemin prendre?



Synthèse logique “structurelle”



Synthèse logique “RTL”