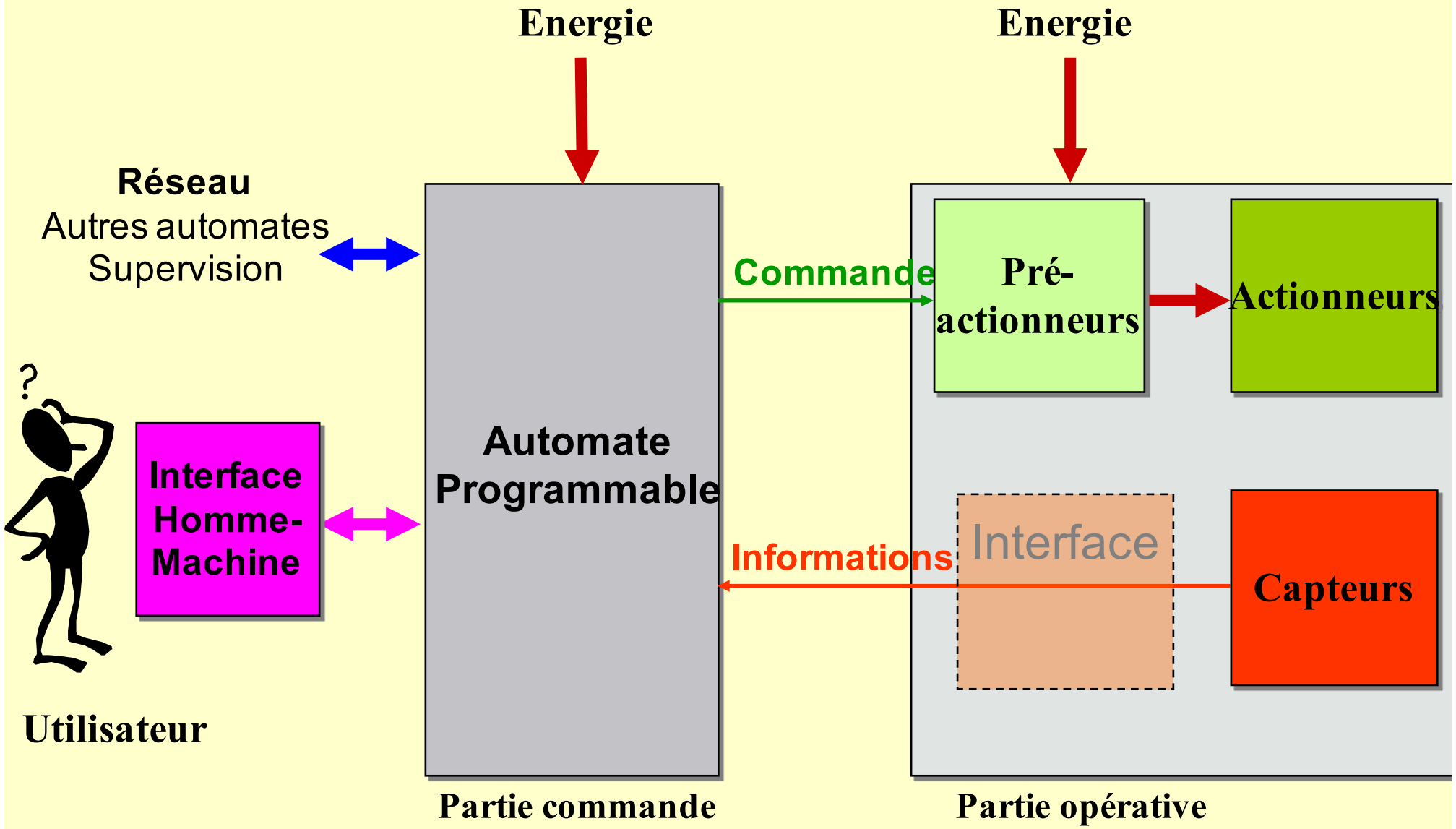
Automate Programmable industriel

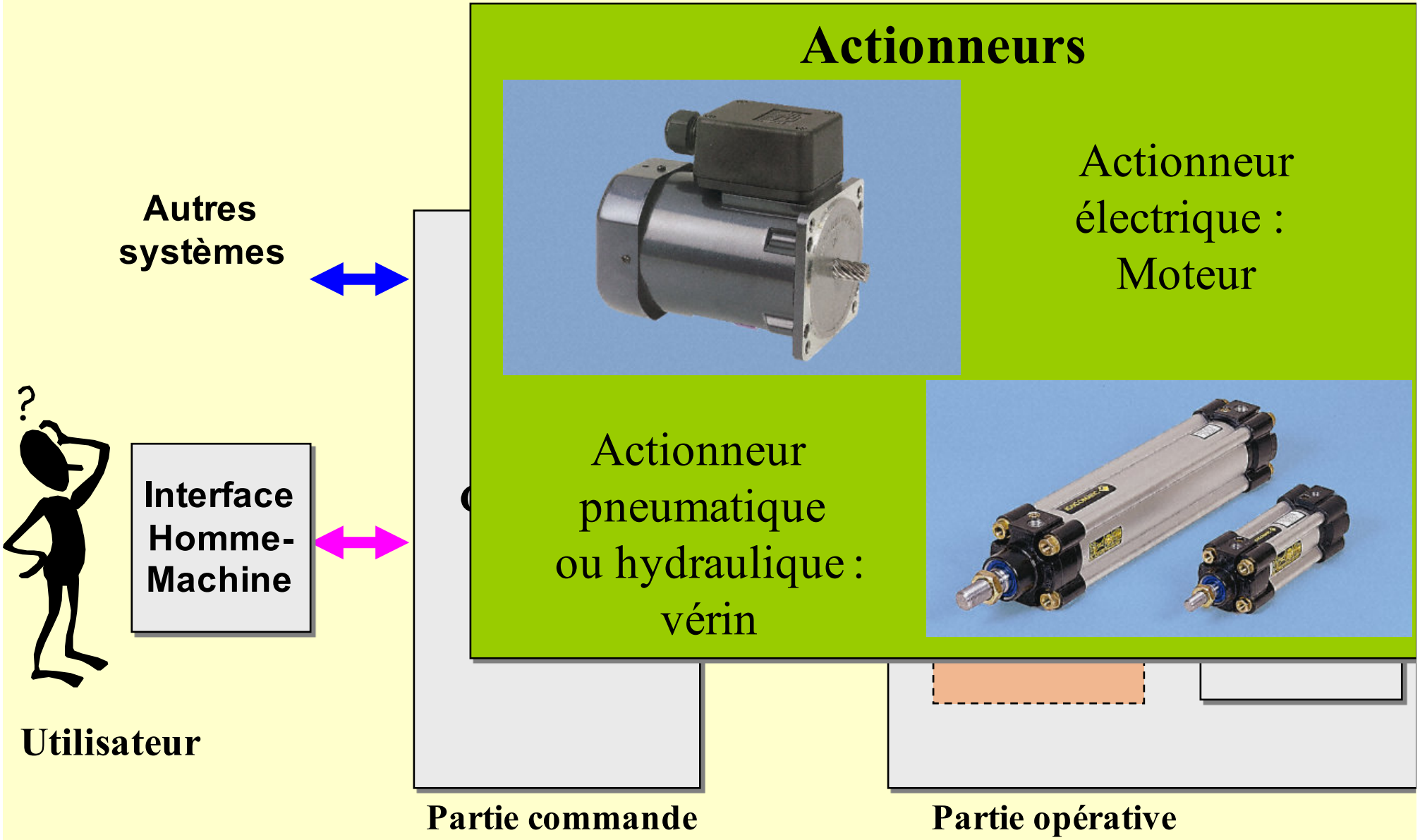
1 Introduction

L’automate programmable au cœur du système automatisé de production

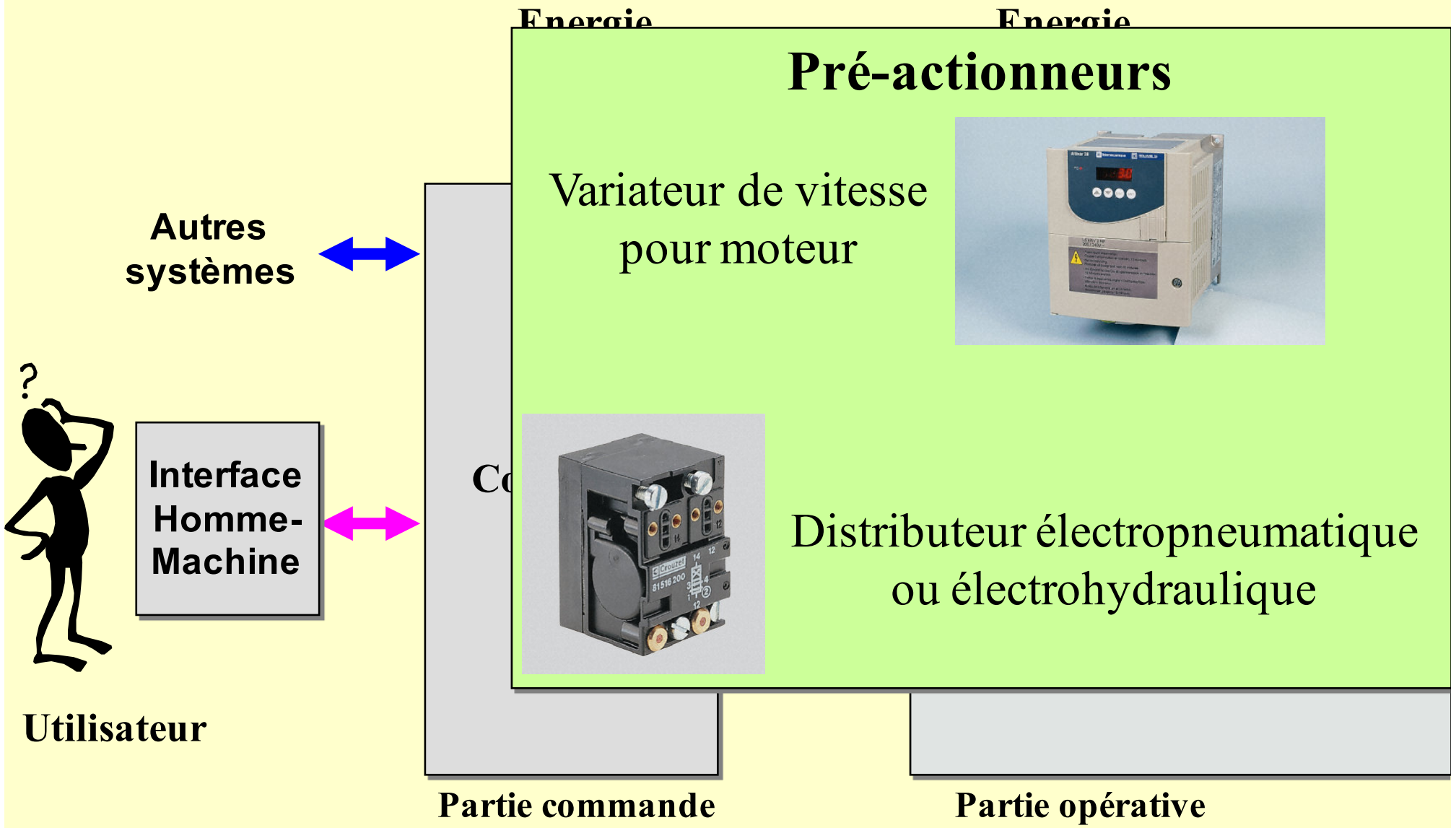
1.1 Structure d'un système automatisé



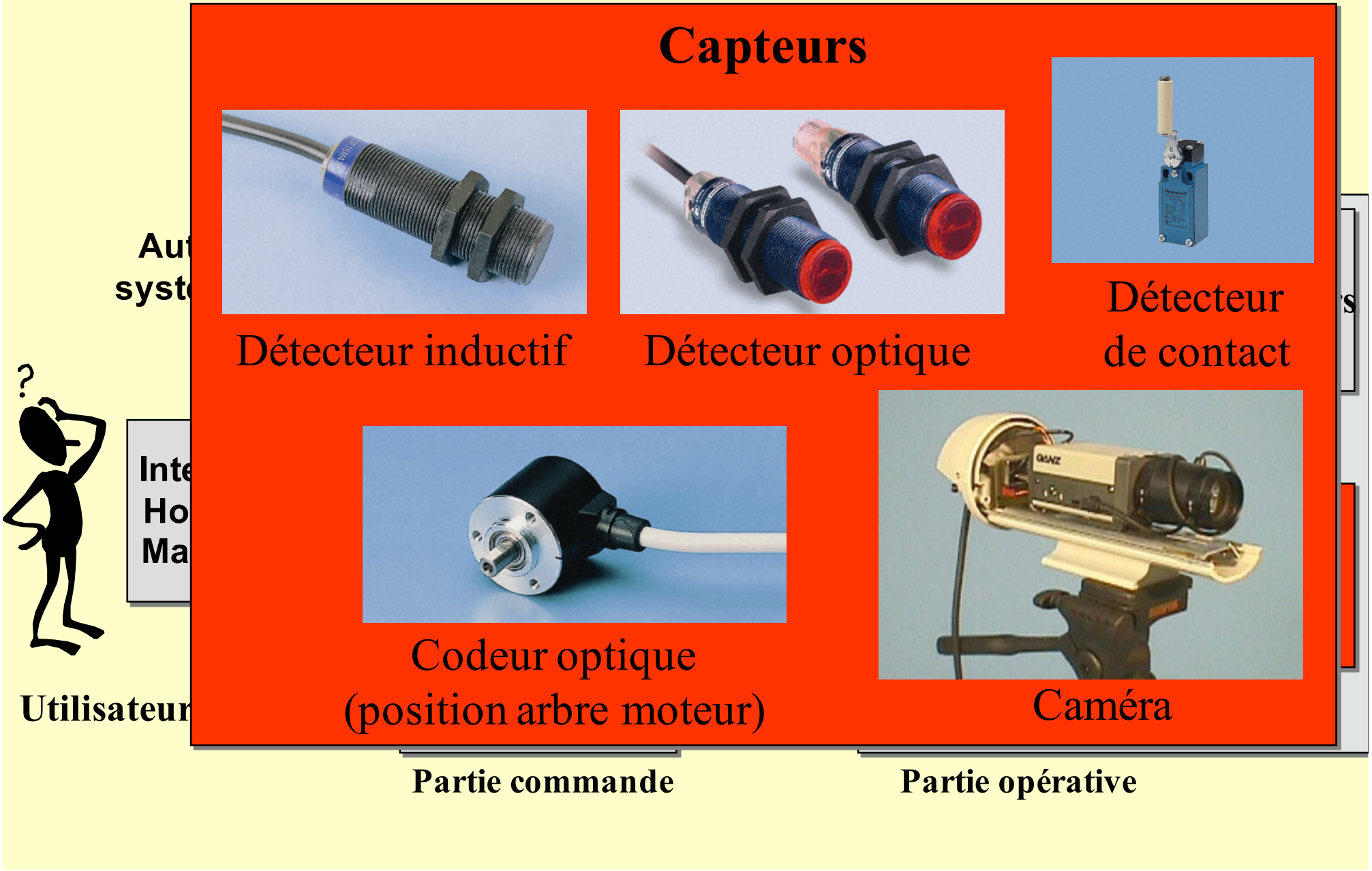
1.1 Structure d'un système automatisé



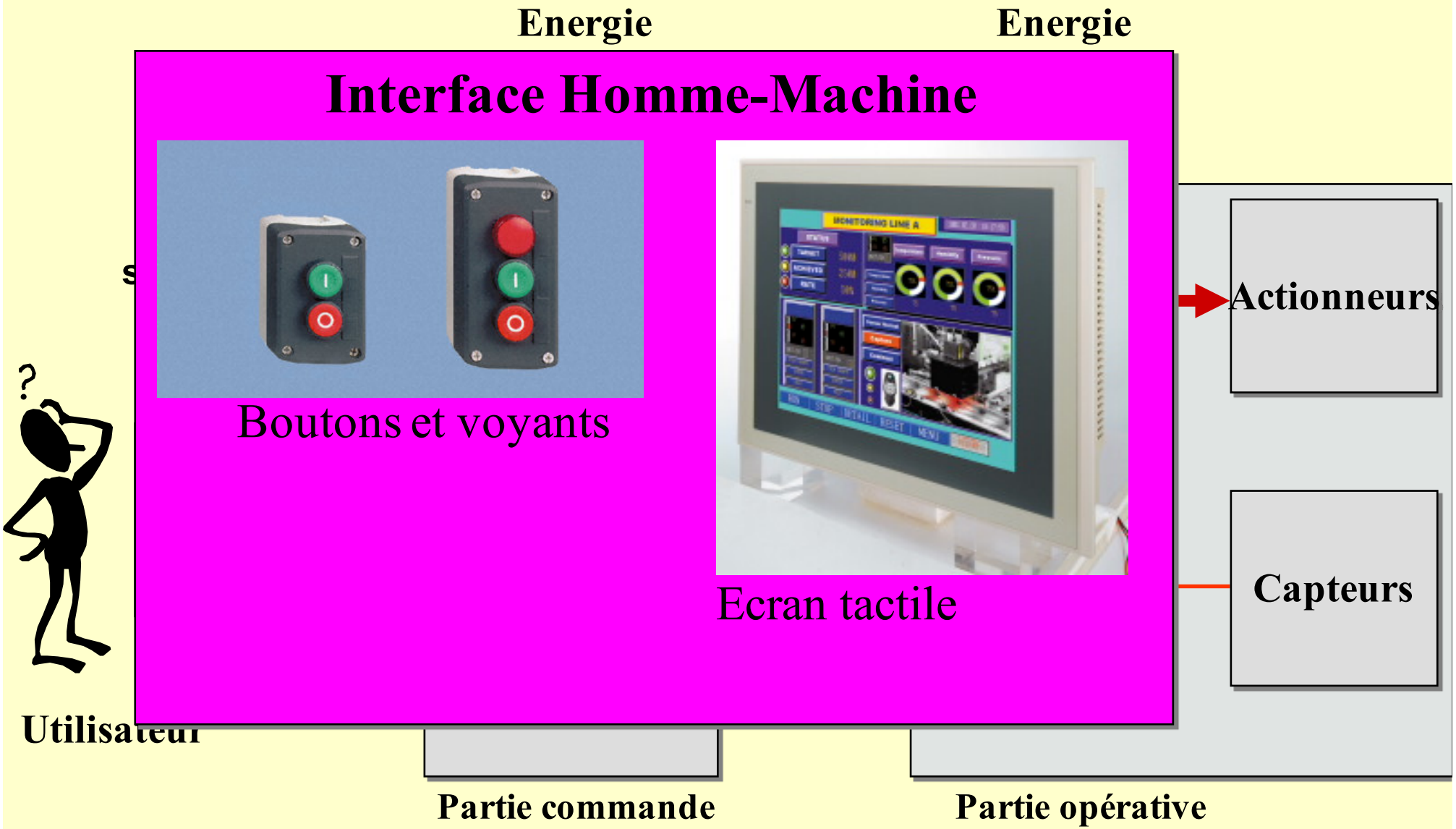
1.1 Structure d'un système automatisé



1.1 Structure d'un système automatisé



1.1 Structure d'un système automatisé



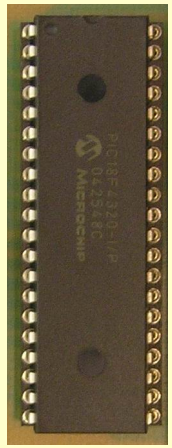
1.1 Structure d'un système automatisé

Qui peut concurrencer l'automate ?

Peu d'éléments à produire
On recherche :
- Un faible coût de développement
- Un développement rapide et aisé

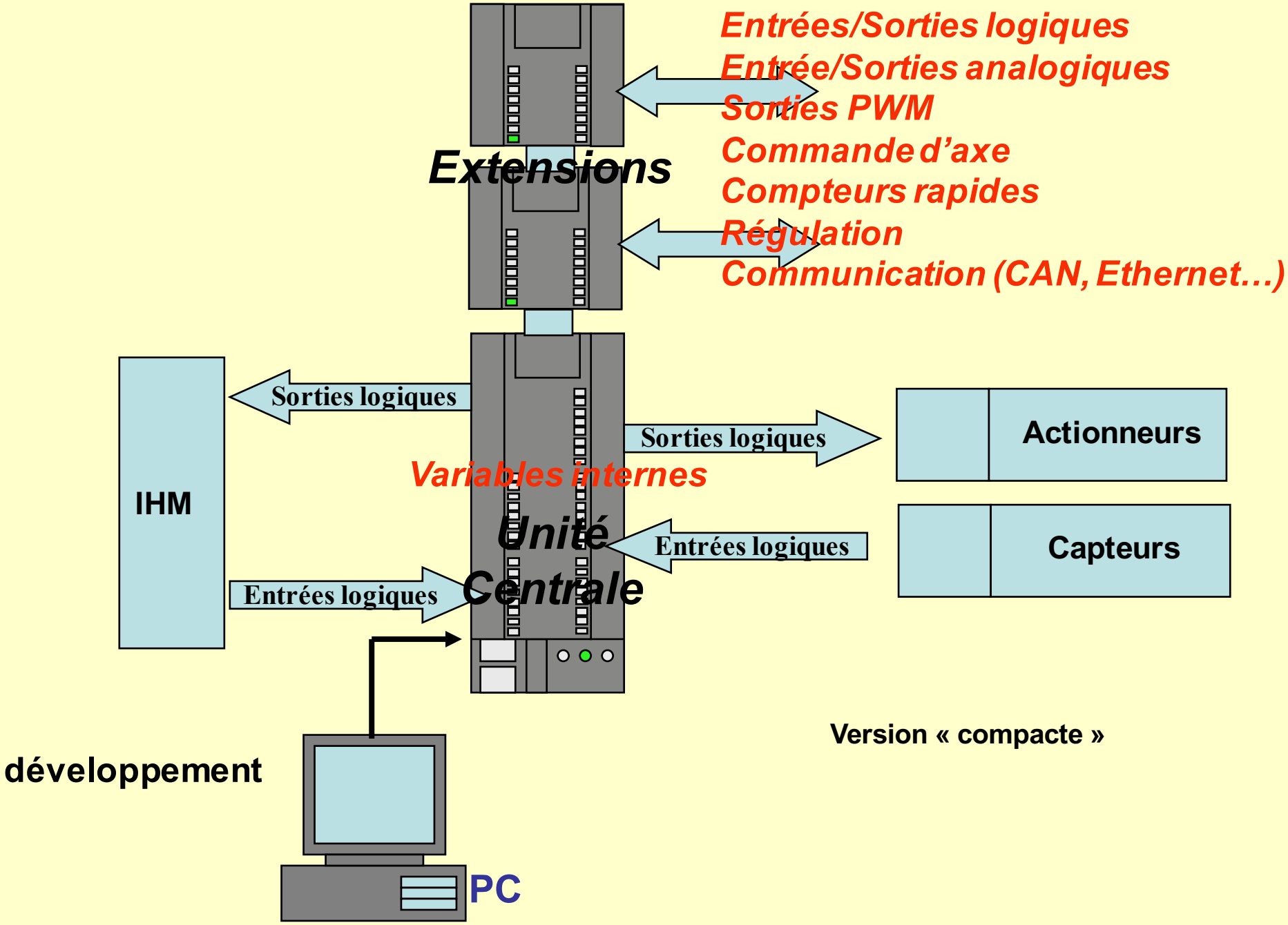
Automate Programmable Industriel

Beaucoup d'éléments à produire (ex : ABS d'une voiture)
On recherche :
- Un faible coût unitaire du composant



Microcontrôleur

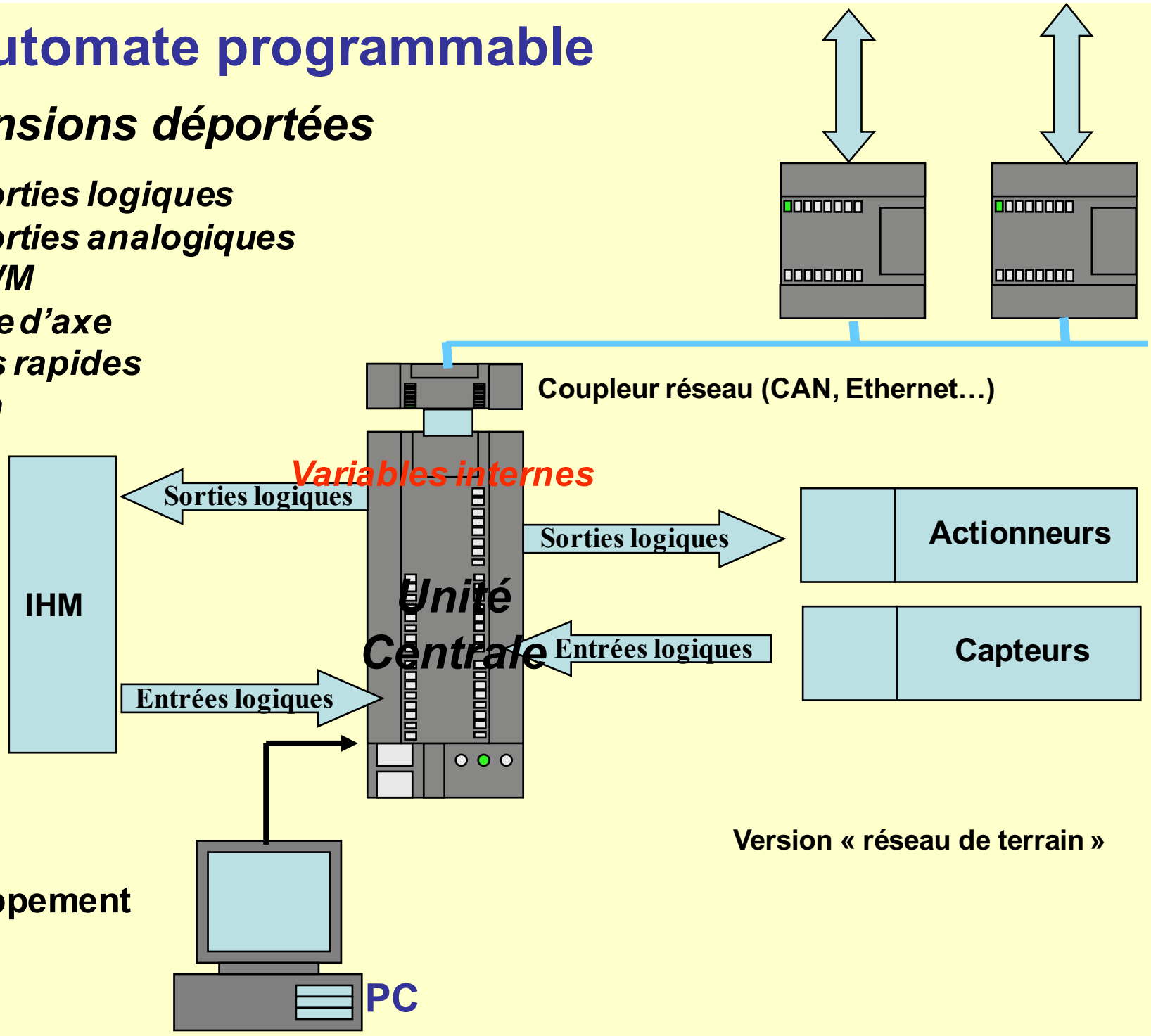
1.2 L'automate programmable



1.2 L'automate programmable

Extensions déportées

Entrées/Sorties logiques
Entrées/Sorties analogiques
Sorties PWM
Commande d'axe
Compteurs rapides
Régulation



Version « réseau de terrain »

1.3 Problématiques combinatoires et séquentielles

- Selon le cahier des charges, les habitudes, les contraintes de développement et de mise au point, les taches sont programmées de façon combinatoire ou séquentielle

1.3 Problématiques combinatoires et séquentielles

Exemples combinatoires

- Le voyant s'allume si le capteur d'usure est à 1 ou si la pièce a été utilisée 1000 fois
- Le compteur s'incrémente à chaque cycle de production
- Les conditions de démarrage sont réunies si le capot est fermé et le bain d'huile à la bonne température
- L'alimentation est coupée une présence est détectée dan le champ opératoire

1.3 Problématiques combinatoires et séquentielles

Exemple séquentiel

- Suite à un ordre de marche, le vérin sort. Quand le noyau est totalement sorti, il faut attendre 10s. Ensuite, le vérin rentre. Quand le noyau est totalement rentré, on attend un nouvel ordre de marche.
- Les systèmes séquentiels sont généralement décrits par un grafcet

2 Les langages de programmation

- La norme IEC 1131-3 définit entre autres choses, cinq langages qui peuvent être utilisés pour la programmation d'applications d'automatisme. Les cinq langages sont :
  - SFC « Sequential Function Char »
  - FB « function block diagram »
  - ST « Structured Text »
  - LD « ladder diagram »
  - IL « instruction list»

2.1 Les différents langages

- SFC (« sequential function char ») : issu du langage GRAFCET, ce langage, de haut niveau, permet la programmation aisée de tous les procédés séquentiels ;
- FBD (« function block diagram », ou schéma blocs) : ce langage permet de programmer graphiquement à l'aide de blocs, représentant des variables, des opérateurs ou des fonctions. Il permet de manipuler tous les types de variables ;
- LD (« ladder diagram », ou schéma à relais) : langage graphique dédié à la programmation d'équations booléennes (true/false), très utilisé malgré qu'il soit très bas niveau;

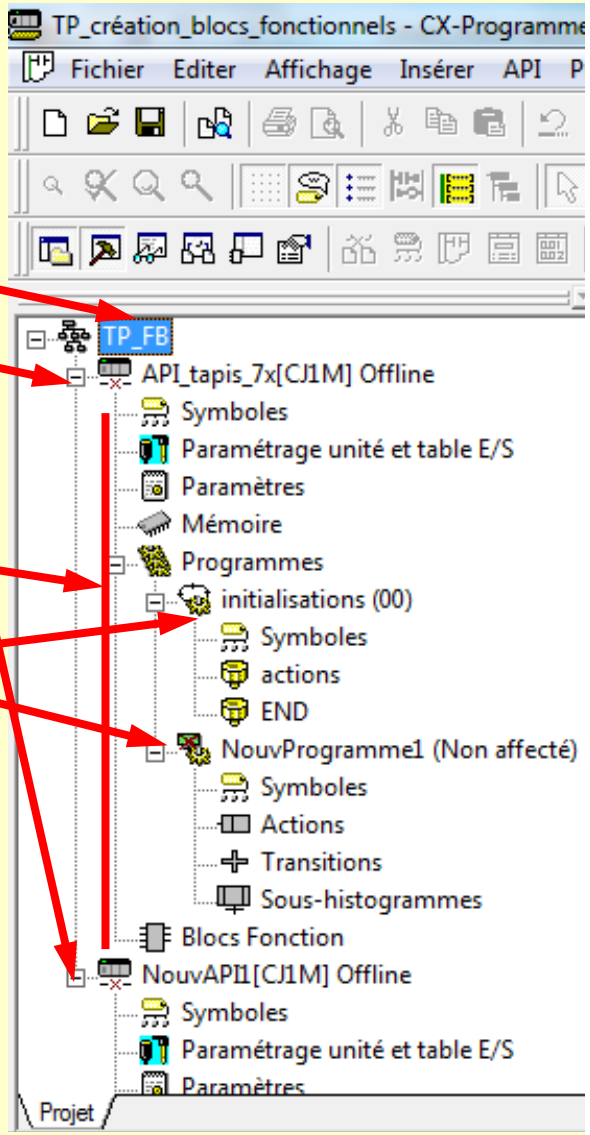
2.1 Les différents langages

- ST (« structured text » ou texte structuré) : ce langage est un langage textuel de haut niveau. Il permet la programmation de tout type d'algorithme plus ou moins complexe ;
- IL (« instruction list », ou liste d'instructions) : ce langage textuel de bas niveau est un langage à une instruction par ligne. Il peut être comparé au langage assembleur.

2.2 Architecture d'un projet

- **Un projet** est composé de **configurations.**
  - **Une configuration:** plate-forme matérielle composée d'une ou plusieurs **ressources**.
    - **Une ressource** représente un cycle automate, divisée en plusieurs unités de programmation appelées **POUs (program organization unit).**
    - **Les POUs** sont organisés en une architecture hiérarchisée. Les programmes peuvent être décrits avec les langages graphiques ou textuels **SFC, FBD, LD, ST ou IL.** Les POUs peuvent être des programmes, des fonctions ou des blocs fonctionnels.

Omron



2.3 les Variables

- Les variables peuvent être locales à un POU Une variable locale ne peut être utilisée que par un seul POU.
- Les variables peuvent être globales elles peuvent alors être utilisées par tous les POUs d'une ressource.

2.3 les Variables

Type de données	Taille	Etendue
Boolean **BOOL**	1	[0(false),1(true)]
Short integer **SINT**	8	[-128,127]
Integer **INT**	16	[-32768,32767]
Double integer **DINT**	32	[-2.147.483.648,2.147.483.647]
Unsigned short integer**USINT**	8	[0,255]
Unsigned integer **UINT**	16	[0,65535]
Unsigned double integer **UDINT**	32	[0,4.294.967.295]
Real numbers **REAL**	32	$[-2^{128}, 2^{128}]$
Bit string of length 8 **BYTE**	8	[16#00,16#FF]
Bit string of length 16 **WORD**	16	[16#0000,16#FFFF]
Bit string of length 32 **DWORD**	32	[16#0000,16#FFFFFFFF]
TIME(*)		T#0,00s à T#21 474 836,47s

2.3 les Variables

- Une variable représentée directement peut être utilisée dans les programmes pour accéder à une voie d'E/S (entrée/sortie). L'identificateur d'une telle variable commence toujours par le caractère « % »

%	Attribut	type	i.j.k
	I: Entrée	X: boolean	i: numéro de rack
	Q : Sortie	B: 8 bits	j: numéro de carte
	M : mémoire	W: 16 bits	k: *numéro de voie*
	K : constante	D: 32 bits	
		F: flottant	

%IX1.5 Entrée TOR n°5 de la carte n°1 du rack par défaut n°0
%QX2.4.F Sortie TOR voie F du module 4 du rack 2
%MX128 bit interne n°128
%MW4 entier 16 bits n°4

2.3 Table des entrées/sorties et mémoire interne

En fonction de sa puissance, un API possède

Des signaux d'entrées/sorties physiques

- Des entrées logiques (accessibles bit/bit ou par mot)
- Des sorties logiques (accessibles bit/bit ou par mot)
- Des entrées analogiques
- Des sorties PWM
- Des entrées de comptage
- …..

De la mémoire interne pour le stockage de données (transition, étapes, calculs intermédiaires…)

- Bits ou mots « système » (ex First_Cycle)
- Bits ou mots réservés pour certains périphériques (ex FinTimer)
- Bits ou mots utilisables pour les données, mémorisés ou non suite à une coupure d'alimentation.

2.3 Table des entrées/sorties et mémoire interne

La Table des symboles permet d'associer à chaque symbole un emplacement physique ou en mémoire.

Name	Type	Address	Comment
VC4	EBOOL	%Q0.3.9	Voyant cabine niveau 4
VC3	EBOOL	%Q0.3.15	Voyant cabine niveau 3
VC2	EBOOL	%Q0.3.10	Voyant cabine niveau 2
VC1	EBOOL	%Q0.3.11	Voyant cabine niveau 1
VA4	EBOOL	%Q0.3.12	Voyant d'appel niveau 4
VA3	EBOOL	%Q0.3.14	Voyant d'appel niveau 3
VA2	EBOOL	%Q0.3.13	Voyant d'appel niveau 2
VA1	EBOOL	%Q0.3.8	Voyant d'appel niveau 1
t10_11	BOOL		
p1	EBOOL	%I0.1.4	Presence niveau 1
N	INT		
DP4	EBOOL	%I0.1.0	Presence niveau 4
DP3	EBOOL	%I0.1.1	Presence niveau 3
DP2	EBOOL	%I0.1.2	Presence niveau 2
Cde_Mont	EBOOL	%Q0.3.1	Commande de montée du moteur cabine
Cde_Desc	EBOOL	%Q0.3.0	Commande de descente du moteur cabine
Bp_Manu	EBOOL	%I0.1.12	Bp marche manu
BP_bl	EBOOL	%I0.2.0	Bouton Blanc pupitre
Bp_Auto	EBOOL	%I0.1.11	Bp marche Auto
BP4d	EBOOL	%I0.1.5	Appel niveau descente 4
BP4	EBOOL	%I0.1.14	Appel Cabine niveau 4

Sortie 1 bit

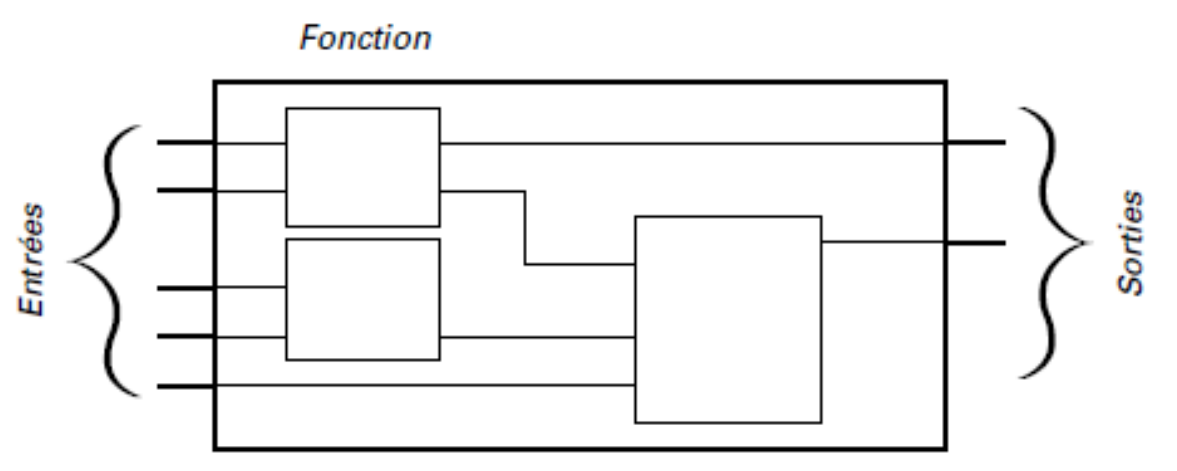
Bit interne

Mot interne

Entrée 1 bit

2.4 Le langage FDB

- Le langage FBD (function block diagram) est un langage graphique.
Il permet la construction d'équations complexes à partir des opérateurs standards, de fonctions ou de blocs fonctionnels.



2.4 Langage FDB

- Les Blocs Fonctionnels sont des POUs avec plusieurs paramètres d'entrée et de sortie.
- L'instanciation est possible. Il est possible de créer plusieurs instances (copies) nommées d'un blocs. Intérêt sur des taches répétitives, exemple: Programmation d'un bloc fonctionnel réalisant le démarrage étoile/triangle d'une MAS pour piloter 10 moteurs.
-Les blocs fonctionnels peuvent être programmé en ST, LD, IL.
-Un bloc fonctionnel peut appeler un autre bloc (pas toujours possible)

2.4 Le langage Strucured Text ST

- Le langage ST (structured text) est un langage textuel de haut niveau dédié aux applications d'automatisation. Ce langage est principalement utilisé pour décrire les procédures complexes, difficilement modélisables avec les langages graphiques. C'est le langage par défaut pour la programmation des actions dans les étapes et des conditions associées aux transitions du langage SFC.

```
IF manual AND not (alarm) THEN
        level := manual_level;
        bx126 := bi12 OR bi45;
ELSIF over_mode THEN
        level := max_level;
ELSE
        level := (lv16 * 100) / scale;
END_IF;
```

2.5 Langage ST:

Opérateurs	Description
:=	Affectation (ex a:=5;)
>, <, =, >=, <=; <>	Opérateurs de comparaison.
-, +, *, /	Opérateurs mathématiques.
NOT, OR, AND, XOR	Opérateurs booléens.

Les types d'énoncés du langage ST sont :

— assignation ;
— sélection IF-THEN-ELSIF-ELSE ;
— énoncé RETURN ;
— sélection CASE ;
— itération WHILE ;
— itération REPEAT ;
— itération FOR ;
— énoncé EXIT.

2.5 Langage ST:

Enoncé IF-THEN-ELSIF-ELSE :

```
IF <expression_booléenne> THEN
<énoncé> ;
<énoncé> ;
...
ELSIF <expression_booléenne> THEN
<énoncé> ;
<énoncé> ;
...
ELSE
<énoncé> ;
<énoncé> ;
...
END_IF;
```

```
IF (condition)
THEN instruction 1;
ELSE instruction 2;
END_IF;
```

Exécution de l'instruction 1 si la condition est vraie et de l'instruction 2 sinon.

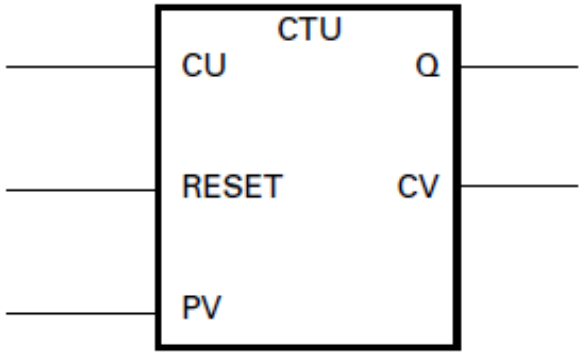
```
(* Programme ST avec énoncé IF *)
IF manual AND not (alarm) THEN
        level := manual_level;
        bx126 := bi12 OR bi45;
ELSIF over_mode THEN
        level := max_level;
ELSE
        level := (lv16 * 100) / scale;
END_IF;
(* Structure IF sans ELSE *)
If overflow THEN
        alarm_level := true;
END_IF;
```

2.5 Langage ST:

Enoncé Return : termine l'exécution du programme

Exemple :

(* Spécifications FBD du programme : compteur programmable *)




```
(* implémentation ST du programme, avec énoncé RETURN *)
If not (CU) then
        Q := false;
        CV := 0
        RETURN; (* fin du programme *)
end_if;
if RESET then
        CV := 0;
else
        if (CV < PV) then
                CV := CV + 1;
        end_if;
end_if;
Q := (CV >= PV);
```

2.5 Langage ST:

Enoncé CASE :

```
CASE (variable) OF
1: (instruction 1);
2: (instruction 2);
3: (instruction 3);
ELSE
(instruction 4);
END_CASE;
```

Si variable prend la valeur 1, l'instruction 1 est exécutée. Si la valeur est 2, l'instruction est exécutée, etc... Si la valeur de la variable ne correspond à aucune des valeurs listées, l'instruction 4 est exécutée.

```
CASE error_code OF
        255:    err_msg := 'Division by zero';
                fatal_error := TRUE;
        1:      err_msg := 'Overflow';
        2, 3:   err_msg := 'Bad sign';
ELSE
                err_msg := 'Unknown error';
END_CASE;
```

2.5 Langage ST:

Enoncé WHILE :

```
WHILE (condition) DO
(instruction);
END_WHILE;
```

Tant que la condition est vraie, l'instruction est réalisée

Enoncé REPEAT : énoncé d'itération avec test d'itération en fin de boucle.

```
REPEAT
 <énoncé> ;
 <énoncé> ;
 ...
UNTIL <condition_booléenne>
END_REPEAT ;
```

2.5 Langage ST:

Enoncé FOR :

```
FOR (A := B TO C BY D)
DO (instruction);
END_FOR;
```

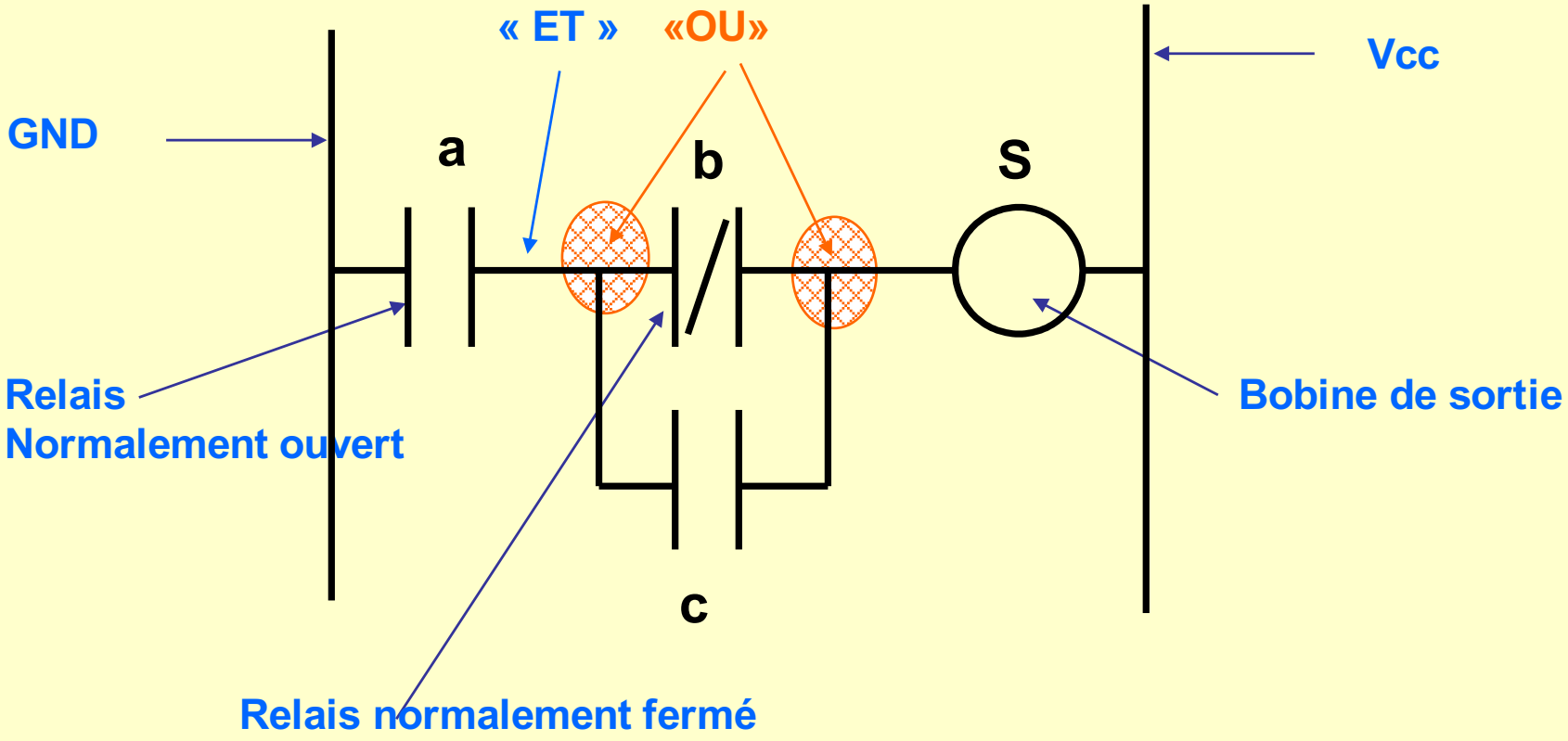
Pour A allant de B à C par pas de D, l'instruction est réalisée.

Enoncé EXIT : quitte une boucle d'itération FOR, WHILE ou REPEAT

```
(* ce programme recherche un caractère dans une chaîne *)
length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
        code := ascii (message, index);
        IF (code  = searched_char) THEN
                found := YES;
                EXIT;
        END_IF;
END_FOR;
```

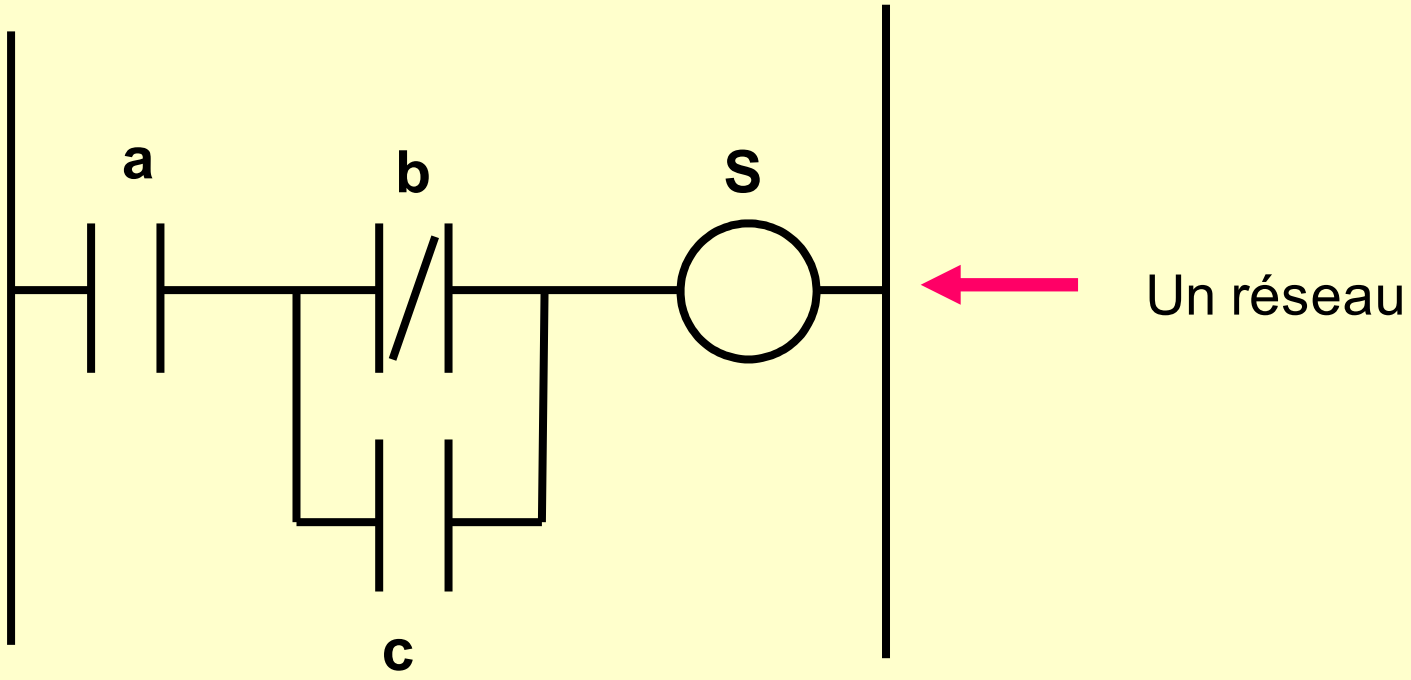
2.6 Le langage LADDER

Diagramme à contacts, utilisé pour programmer des éléments combinatoires



S = a . (/b + c)

2.6 Le langage LADDER



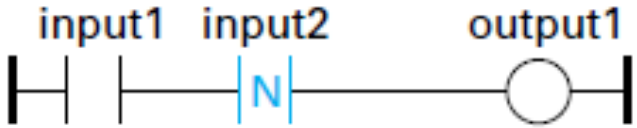
- On utilise un réseau par calcul
- Une section est un ensemble de réseaux
- Pour structurer le programme, on l'organise en sections

2.6 Le langage LADDER

Exemple de contact à détection de front positif :


```
(* Équivalence ST *)
ouput1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev est la valeur de input2 au cycle précédent *)
```

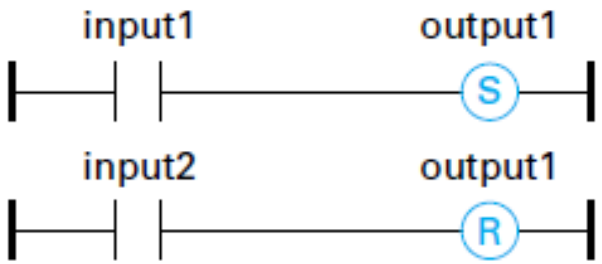
Exemple de contact à détection de front négatif :




```
(* Équivalence ST *)
ouput1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev est la valeur de input2 au cycle précédent *)
```

2.6 Le langage LADDER

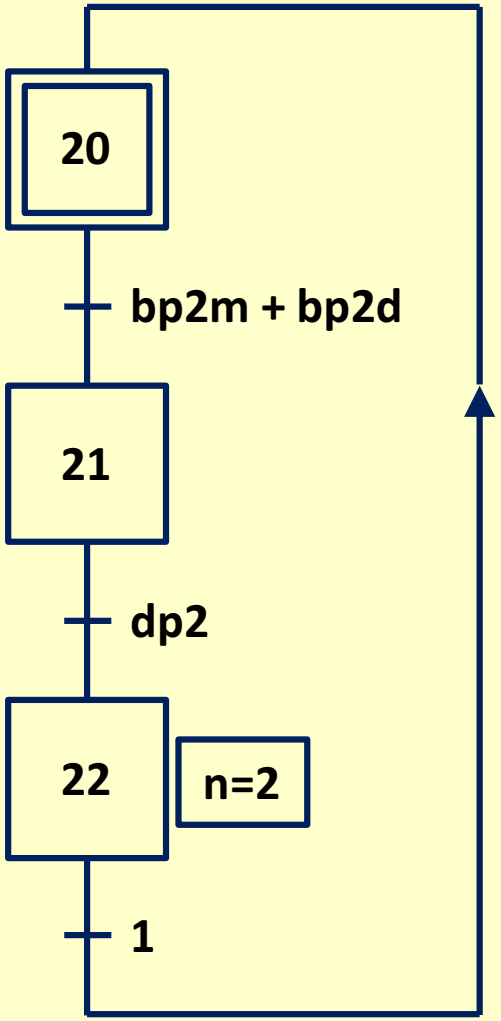
Exemple de relais à action SET et RESET :




```
(* Équivalence ST *)
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

2.7 traduction du SFC en ST

Exemple



❑ Pour chaque transition on définit une variable interne TOR

t20_21 t21_22 t22_20

Chacune vaut 1 si la transition est vrai, 0 sinon

❑ Pour chaque étape on définit une variable interne TOR

X20 X21 X22

Chacune vaut 1 si l'étape est active, 0 sinon

❑ Entrées physiques : bpm2m, bp2d, dp2 (TOR)

Variable interne : n (entier)

2.7 traduction du SFC en ST

Modèle en trois parties, partie 1 : les transitions

```
(* Ceci est un commentaire *)
(* Transitions *)

t20_21 :=  X20 AND (bp2m OR bp2d) ;

t21_22 :=  X21 AND dp2 ;

t22_20 :=  X22 ;
```

Affectation **:=**

Opérateurs
NOT, AND, OR

2.7 traduction du SFC en ST

Modèle en trois parties, partie 2 : les étapes

```
(* Etapes *)

(* Activation *)

IF %S13 OR t22_20 THEN X20 :=TRUE ; END_IF;

IF t20_21 THEN X21 :=TRUE; END_IF;

IF t21_22 THEN X22 :=TRUE; END_IF;
```

2.7 traduction du SFC en ST

Modèle en trois parties, partie 2 : les étapes

```
(* Etapes *)

(* Désactivation *)

IF t20_21 THEN X20:=FALSE; END_IF;

IF t21_22 OR %S13 THEN X21:=FALSE; END_IF;

IF T22_20 OR %S13 THEN X22 :=FALSE; END_IF;
```

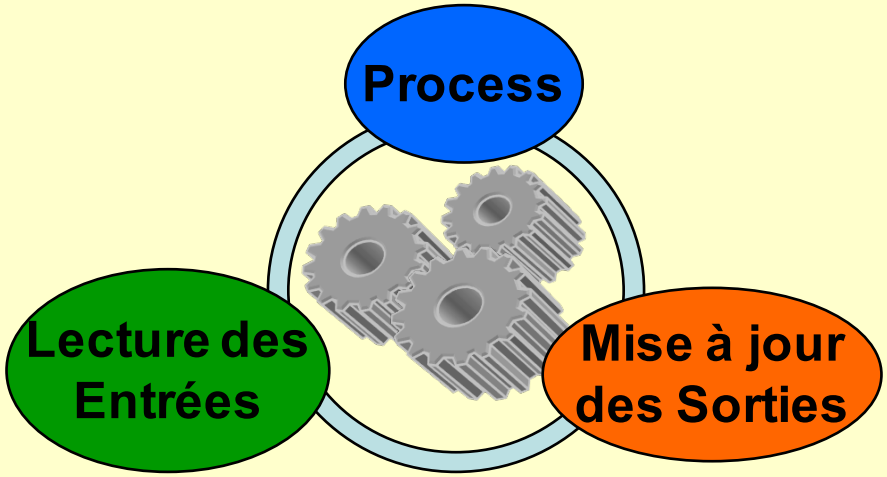
2.7 traduction du SFC en ST

Modèle en trois parties, partie 3 : les sorties

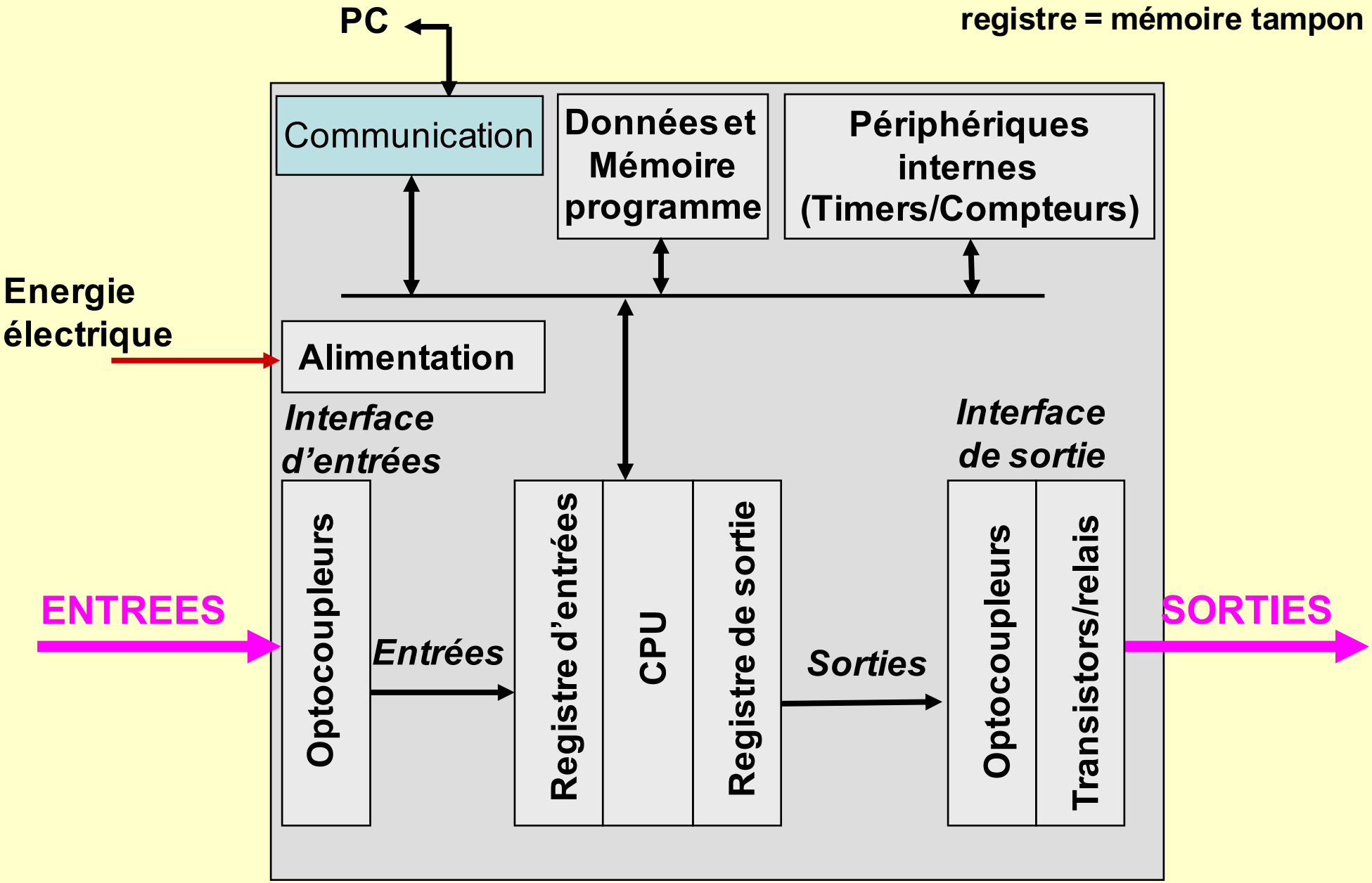
```
(* Sorties *)

IF X22 THEN N:=2;  END_IF;
```

Pour une sortie TOR : IF X22 THEN S:= TRUE;

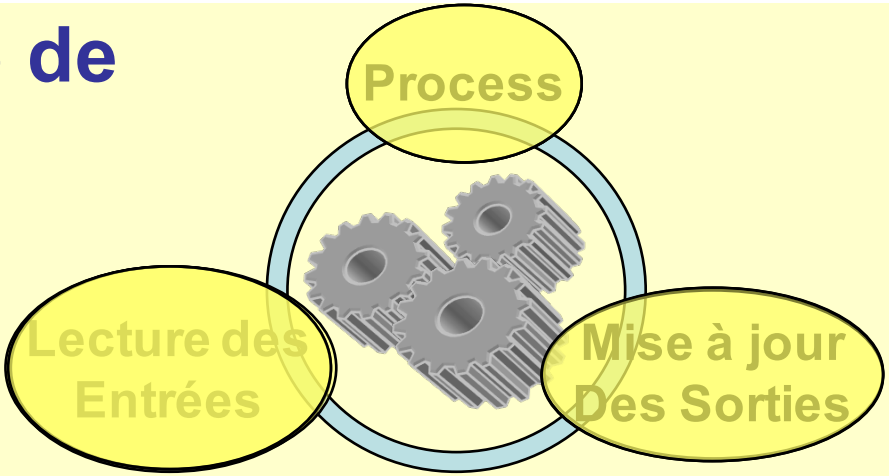
3 Structure interne d’un automate



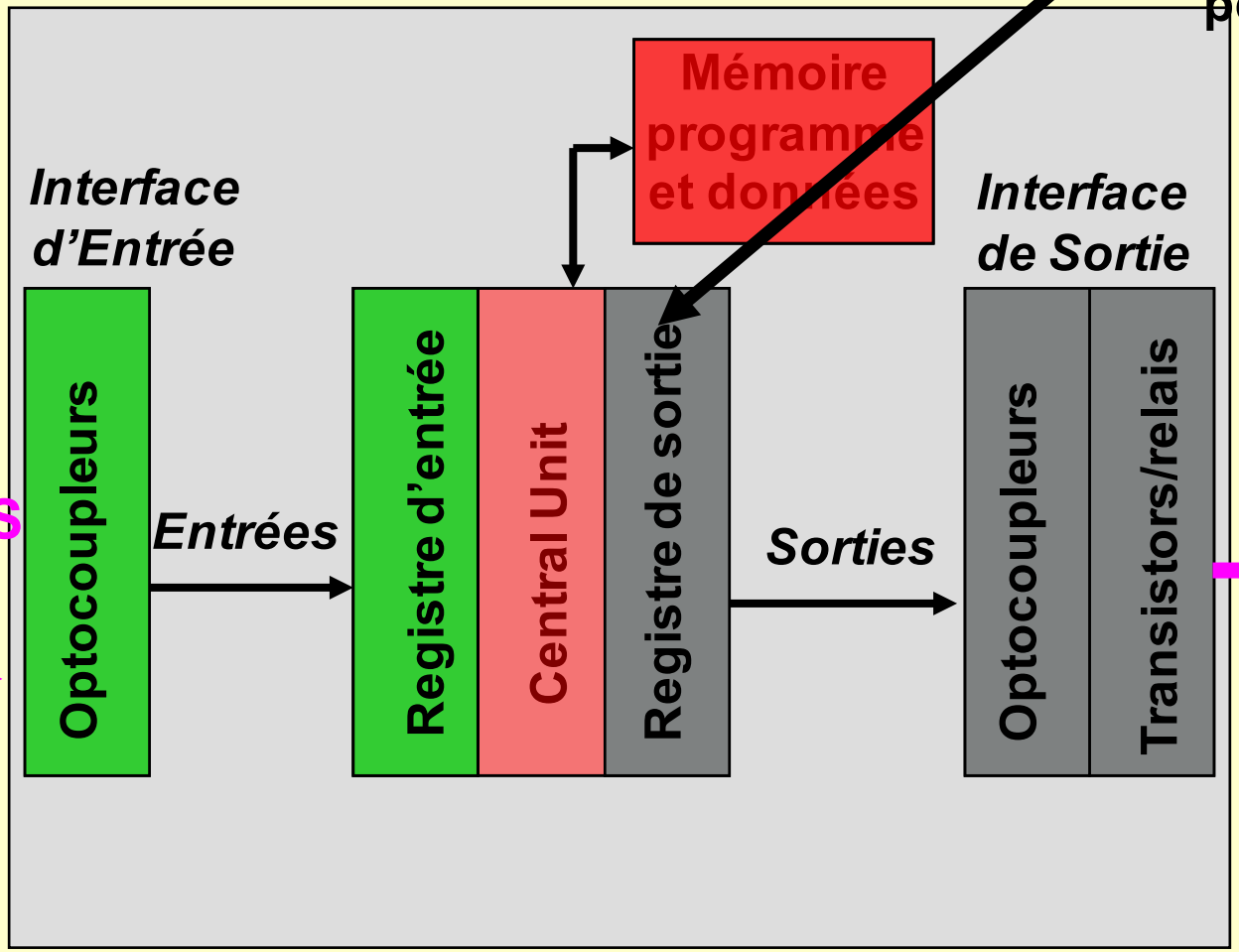
3.1 Structure interne d'un automate



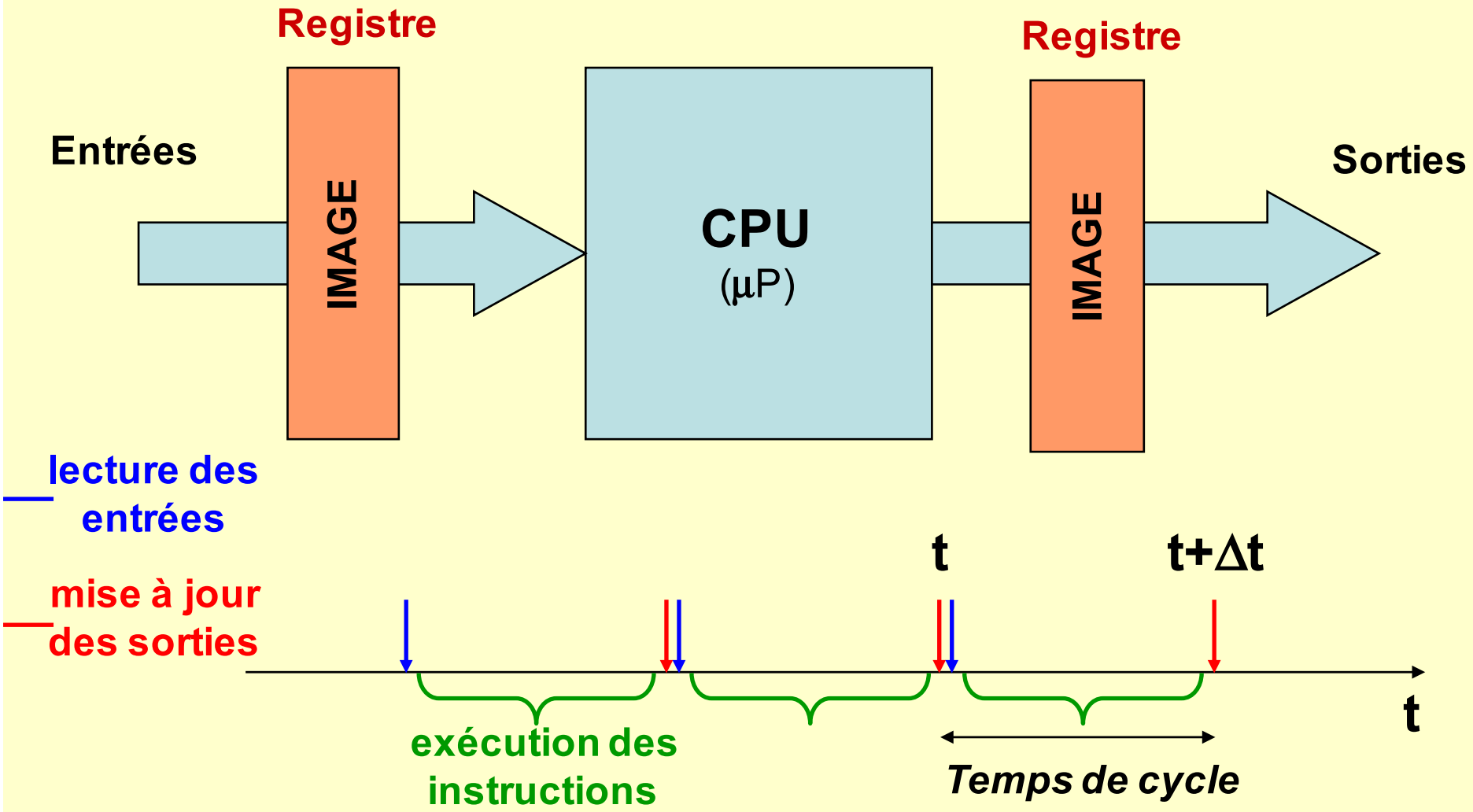
3.2 Le cycle de l'automate



Le contenu du registre change pendant le cycle



3.2 Le cycle de l'automate

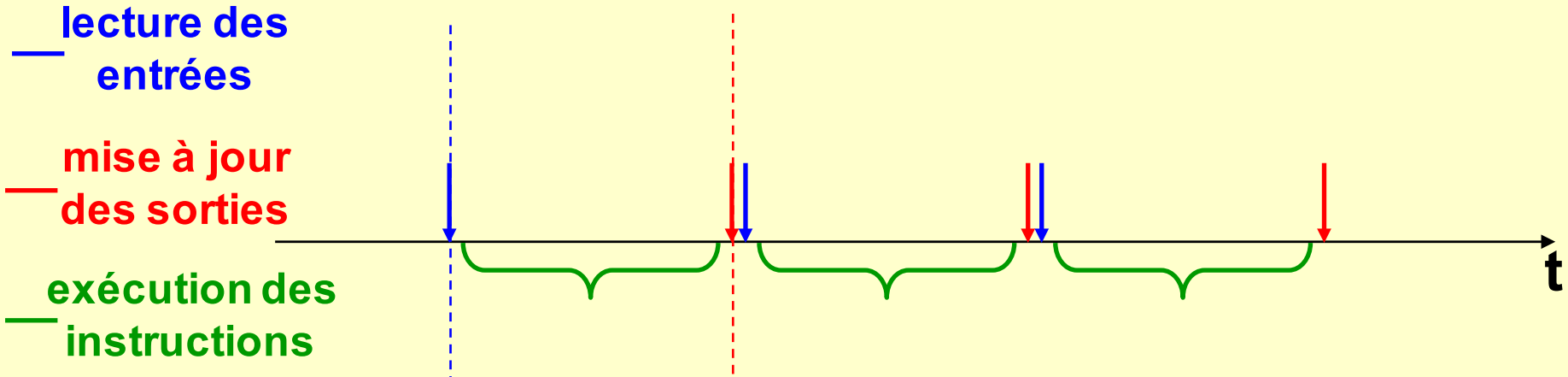


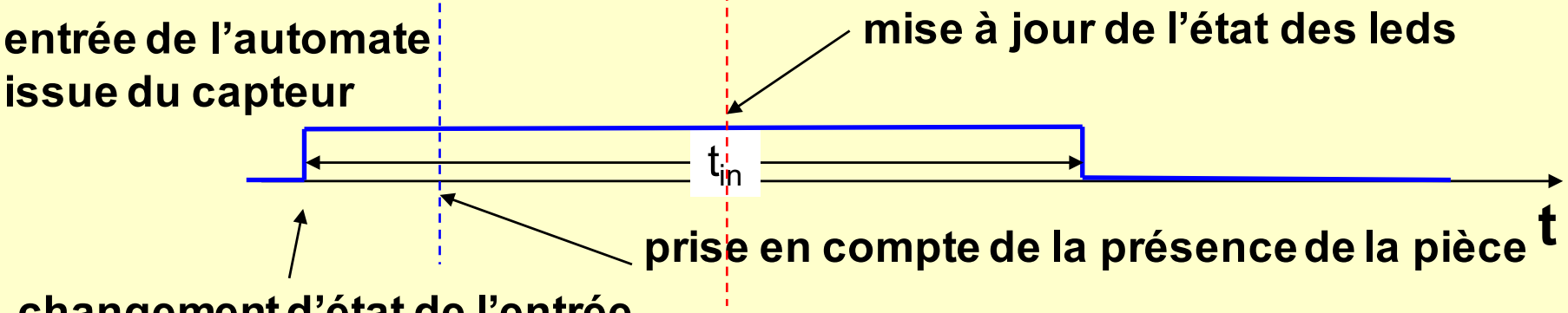
Sorties(t+Δt) = fonction(Entrées(t) , Sorties(t))

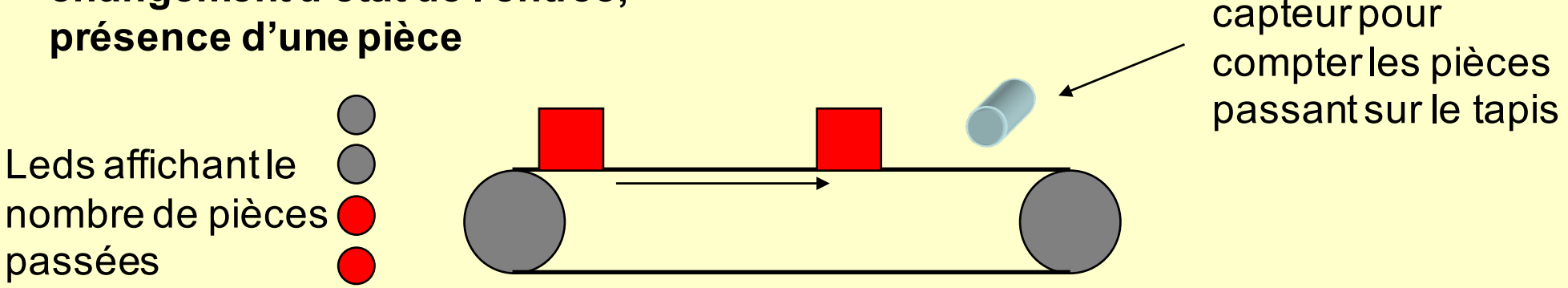
Ordre de grandeur du temps de cycle : entre 1 et 100 ms

3.2 Durée de vie des entrées

le tapis tourne doucement







3.2 Durée de vie des entrées

le tapis tourne plus vite.