Institut de recherche sur les lois fondamentales de l'Univers

Université Paris-Saclay

Conception multitâches et OS temps réel

Shebli Anvar, PhD
CEA Paris-Saclay
Irfu/Dedip/Lilas
91191 Gif-sur-Yvette
France
shebli.anvar@cea.fr
+33 1 69 08 78 32
+33 6 63 31 92 26

Système informatique temps réel

Le comportement d'un système informatique est qualifié de ***temps réel*** lorsqu'il est assujetti à ***l'évolution d'un procédé*** qui lui est connecté et qu'il doit piloter ou suivre en ***réagissant*** à tous ses changements d'état.

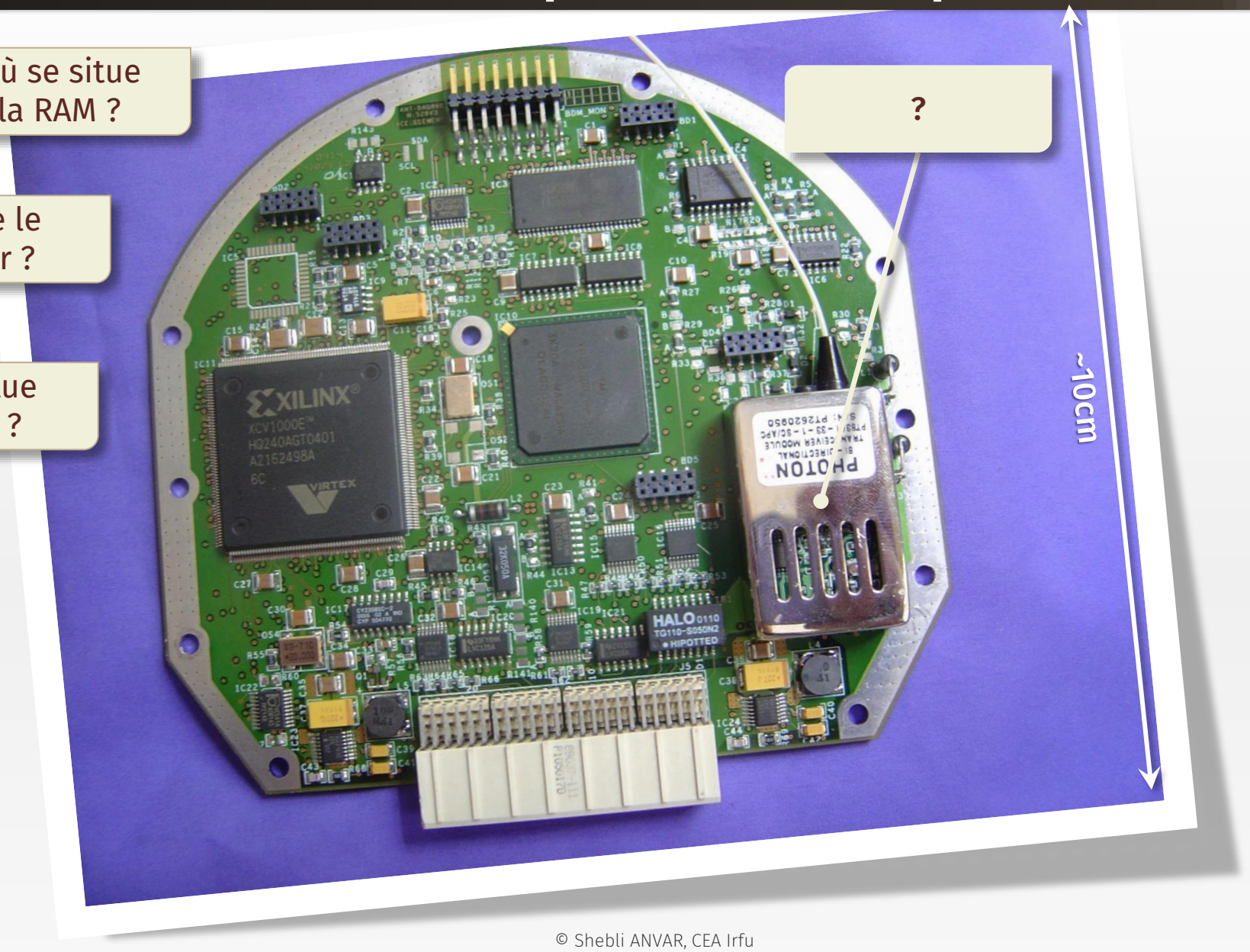
Institut de recherche sur les lois fondamentales de l'Univers

Université Paris-Saclay

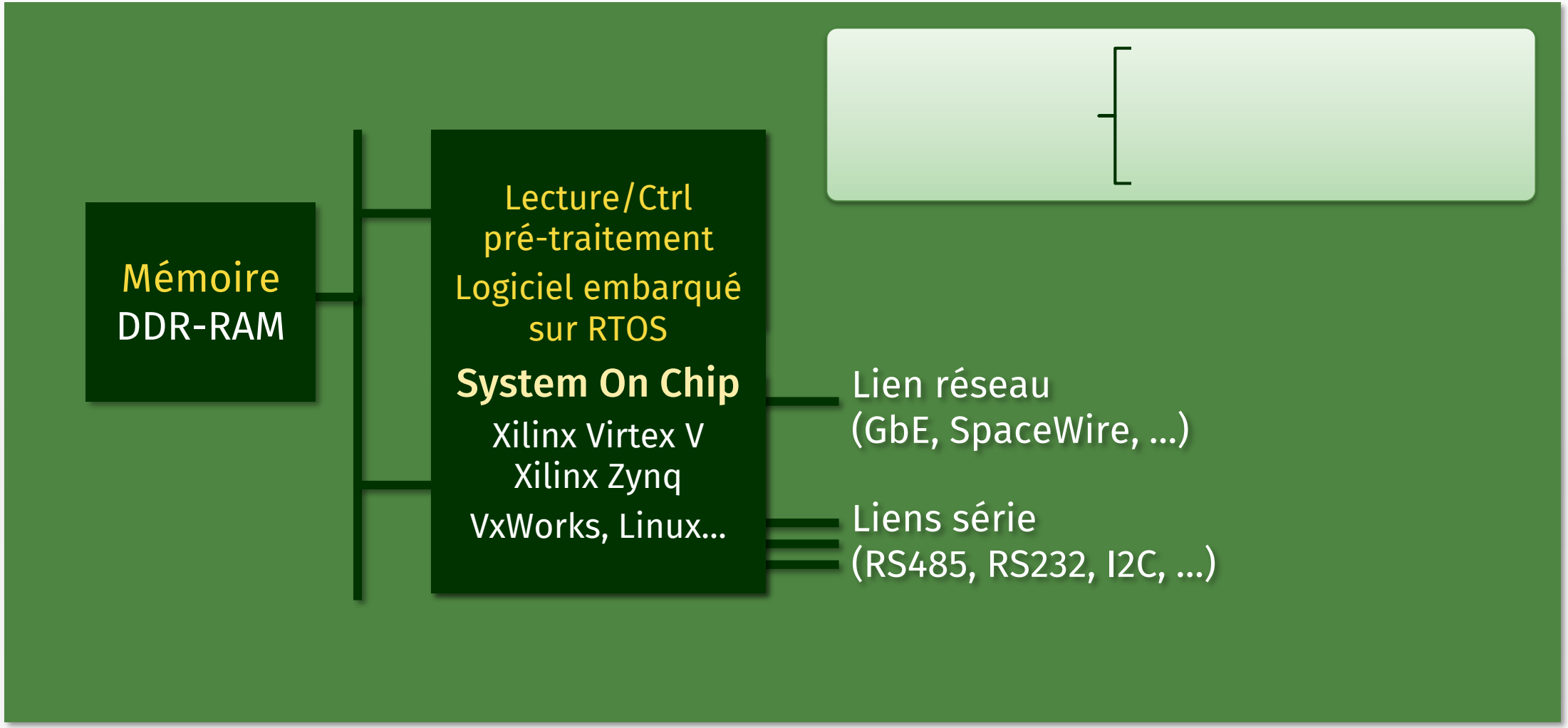
Mapping Mémoire

Shebli Anvar, PhD
CEA Paris-Saclay
Irfu/Dedip/Lilas
91191 Gif-sur-Yvette
France
shebli.anvar@cea.fr
+33 1 69 08 78 32
+33 6 63 31 92 26

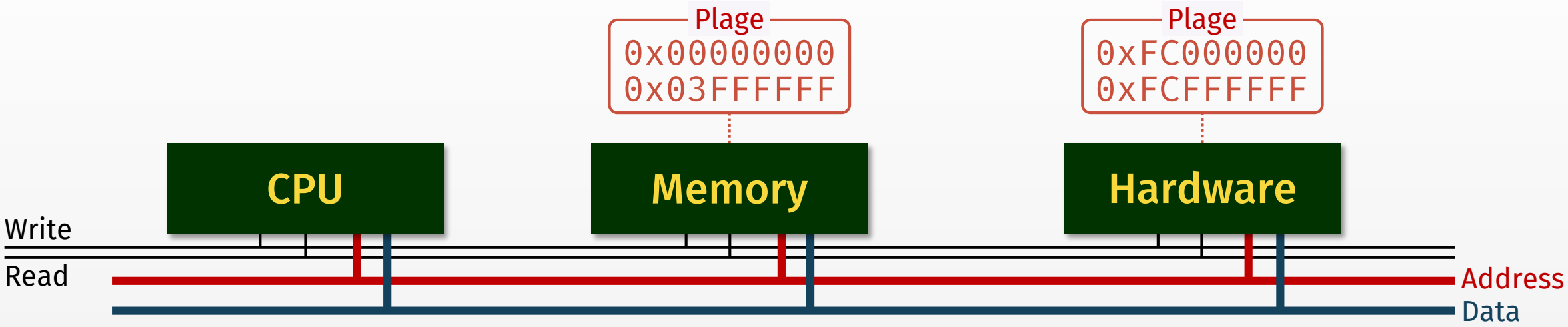
Carte d'acquisition embarquée



Carte d'acquisition embarquée : System On Chip (SoC)

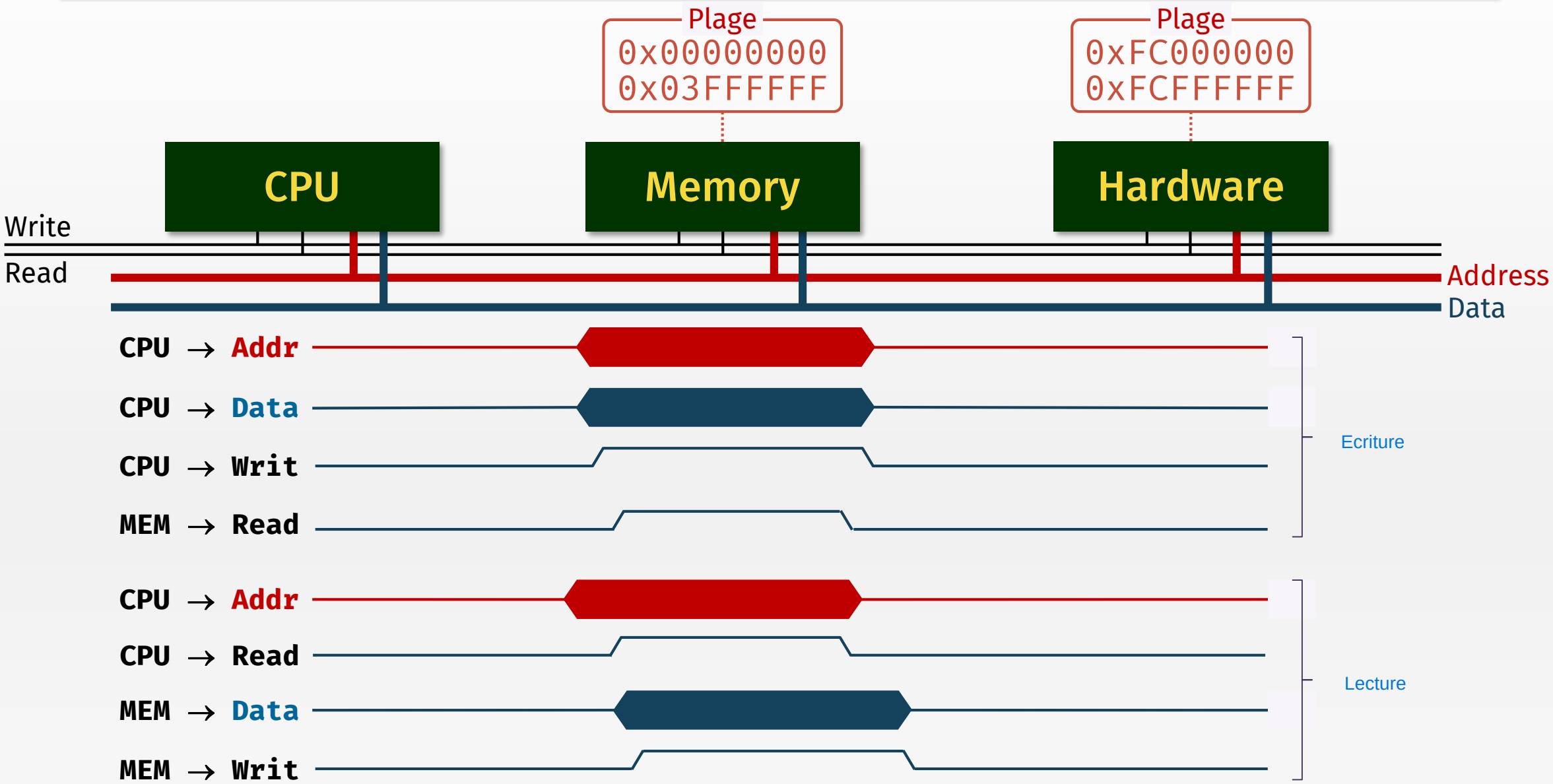


Principe de communication par bus



- **CPU: active l'adresse sur le bus Address (ex: `0xFC002000`).**
- **Périphérique : un seul est sensible à cette adresse**
- **Si écriture :**
  - CPU active la donnée sur le bus Data.
  - Périphérique : concerné traite la donnée sur le bus Data.
- **Si lecture :**
  - Périphérique : active la donnée sur le bus Data.
  - CPU : traite (lis) la donnée sur le bus Data.
- **Modes de lecture/écriture spéciaux (rafale synchrone, etc.)**
- **Contrôleur mémoire : intégré au CPU**

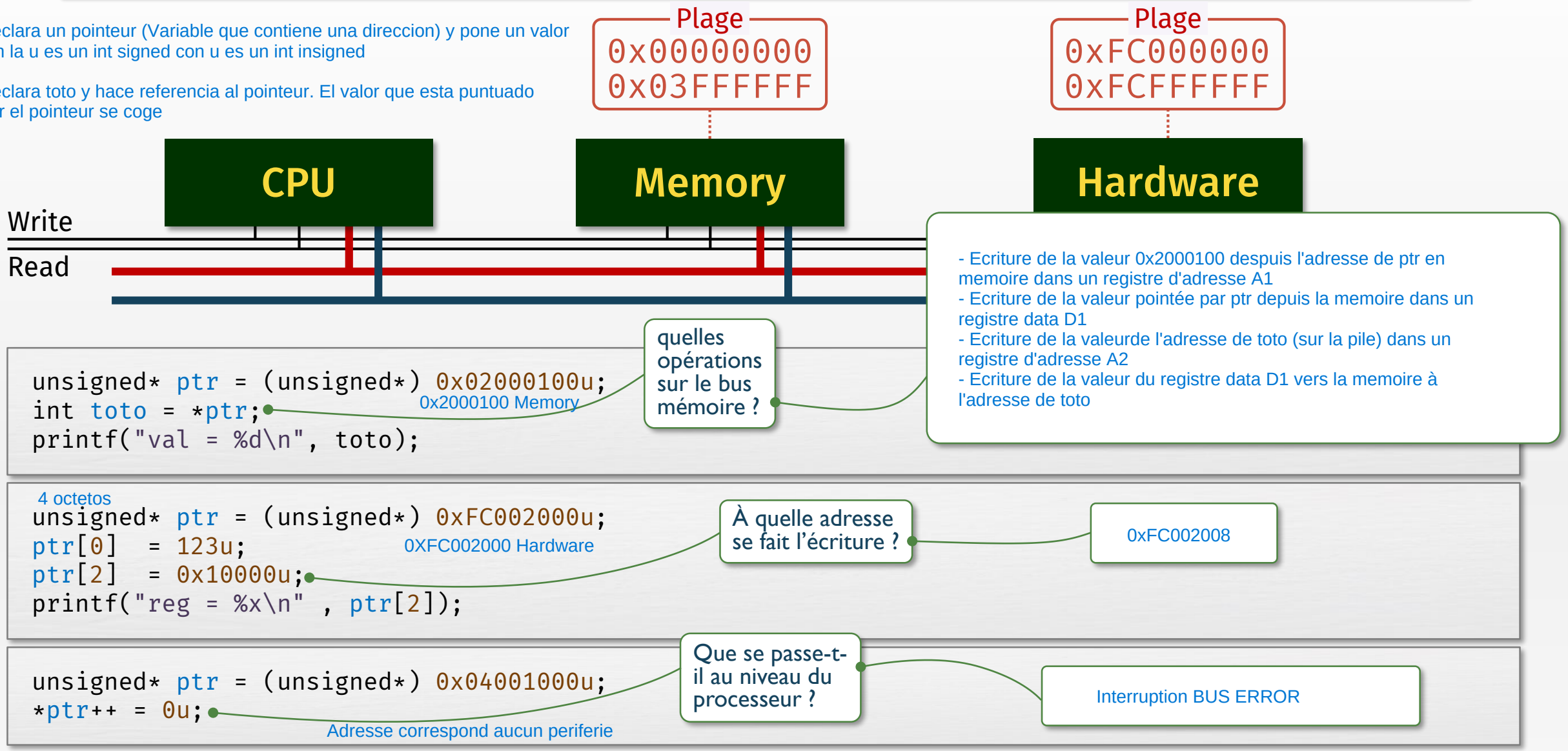
Principe de communication par bus



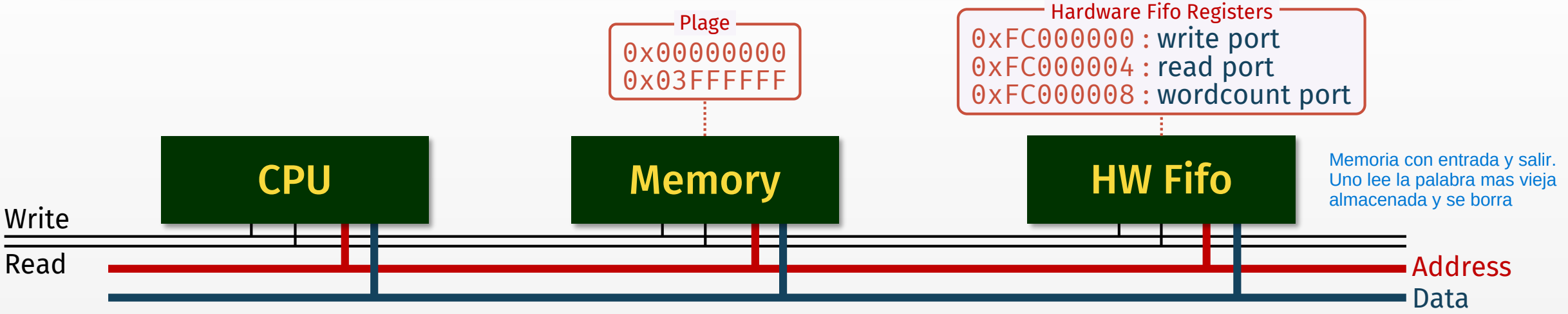
Communication par bus : versant logiciel

Declara un pointeur (Variable que contiene una direccion) y pone un valor
Sin la u es un int signed con u es un int insigned

Declara toto y hace referencia al pointeur. El valor que esta puntuado por el pointeur se coge



Communication par bus : précautions logicielles



Memoria con entrada y salir.
Uno lee la palabra mas vieja almacenada y se borra

Écrivez le code qui écrit les 1000 premiers nombres impairs dans la fifo hardware, puis qui vide les valeurs de la fifo dans un tableau

```
int* pFifo = (int*) 0xFC000000
for (int i=0; i < 1000; i++){
    pFifo[0] = 2*i+1; //La memoria Fifo guarda los valores anteriores
}

//Si jamais la taille de la fifo es plus petit, les mots vont disparaître

int size = pFifo[2]; //pFifo[2] trae el valor de l'adresse 0xFC00008 que es wordcount port
int* tab = new int[size];
for (int i=0;i<size; i++){
    tab[i]=pFifo[1];
}
delete [] tab;
```

```
// Los computadores estan muy avanzados y el optimizador dice jaja que gva esta guardando mil numeros
//  en la misma posicion de memoria, solo va alojar el ultimo valor

*volatile int* pFifo = (int*) 0xFC000000 //Le dice al optimizador: no suponga que este valor es estable
for (int i=0; i < 1000; i++){
    pFifo[0] = 2*i+1; //La memoria Fifo guarda los valores anteriores
}

//Si jamais la taille de la fifo es plus petit, les mots vont disparaître

int size = pFifo[2]; //pFifo[2] trae el valor de l'adresse 0xFC00008 que es wordcount port
int* tab = new int[size];
for (int i=0;i<size; i++){
    tab[i]=pFifo[1];
}
delete [] tab;
```

Taille d'une variable scalaire en mémoire

- **BYTE = unité d'information référencée par une adresse mémoire**
- **Taille en BYTES des éléments scalaires en langage C/C++ : « sizeof »**

Architectures 32 bits typiques		
type	sizeof	bits
char	1	8
short	2	16
int	4	32
long	4	32
long long	8	64
void* adresse	4	32

Architectures 64 bits typiques		
type	sizeof	bits
char	1	8
short	2	16
int	4	32
long	**8**	**64**
long long	8	64
void*	**8**	**64**

Validité de l'opérateur « cast »

```
char* pc = (char*) 0xFC000000;
int n = (int) pc;
void* pv = (void*) n;
short x = (short) pv;
int* pi = (int*) x;
```

Info perdu

Estoy alojando una adresse (32) en un short (16)

Validité de l'opérateur « cast »

```
char* pc = (char*) 0xFC0000001234;
int n = (int) pc;
void* pv = (void*) n;
short x = (short) pv;
int* pi = (int*) x;
```

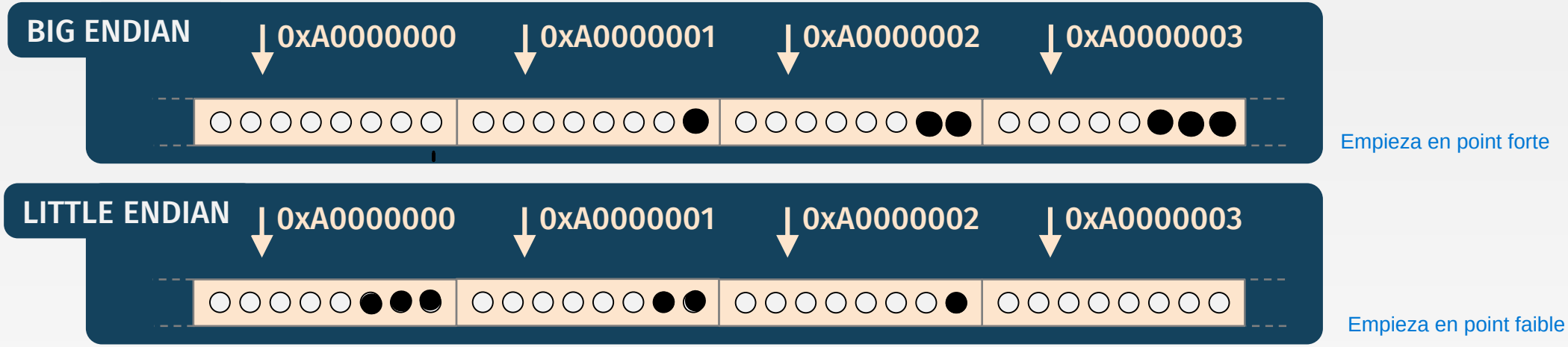
Perd

perd

Endianité (Endianness)

- **BYTE = unité d'information référencée par une adresse mémoire**
- **Lorsqu'un type de donnée primitif (char, short, int, long, etc.) est composé de plus d'un BYTE, la question de l'ordre de ces BYTES en mémoire se pose**
- **Deux grandes classes d'architecture sont aujourd'hui courantes :**
  - LITTLE ENDIAN (architectures Intel i86)
  - BIG ENDIAN (architectures PowerPC, SPARC, etc.)
- **Exemple : *entier 32 bits n* sur mémoire adressant des *BYTES de 8 bits***
  Supposons que *n* contient la valeur 66311 = 0x10307 = 0b10000001100000111
  Supposons que *&n* (adresse de *n* en mémoire) est égale à 0xA0000000

Le point forte est a gauche



Endianité (Endianness)

```
int n = 0x10307; // Élément 32 bits
char* p = (char*) &n; // Pointeur sur octet (8 bits)
printf("%d, %d, %d, %d\n", p[0] , p[1] , p[2] , p[3]);
```

p va pointer au meme endroit que l'adresse de n

Affichage sur architecture BIG ENDIAN

0, 1, 3, 7

Affichage sur architecture LITTLE ENDIAN

7, 3, 1, 0

```
int n = 0x10307; // Élément 32 bits
short* p = (short*) &n; // Pointeur sur élément 16 bits
printf("%d, %d\n", p[0] , p[1]);
```

Affichage sur architecture BIG ENDIAN

1, 775

Affichage sur architecture LITTLE ENDIAN

775, 1

Opérateurs binaires (bitwise operators)

Hexa: Cada letra representa 4 bits

0 0 1 0 0 0 1 1
0 0 0 1 0 0 0 1
_____
0 0 0 0 0 0 0 1

bitwise OR

```
int n = 0x23 | 0x11;
printf("%x, %d\n", n, n);
```

33, 51

bitwise AND

```
int n = 0x23 & 0x11;
printf("%x, %d\n", n, n);
```

1, 1

bitwise XOR

```
int n = 0x23 ^ 0x11;
printf("%x, %d\n", n, n);
```

32, 50

0 0 1 0 0 0 1 1
0 0 0 1 0 0 0 1
_____
0 0 1 1 0 0 1 0

Signed
0 0 0 0
1 1 1 1 Not
1 +
_____
0 0 0 0

Not de numbre +1

bitwise NOT

```
int n = ~0x23;
printf("%x, %d\n", n, n);
```

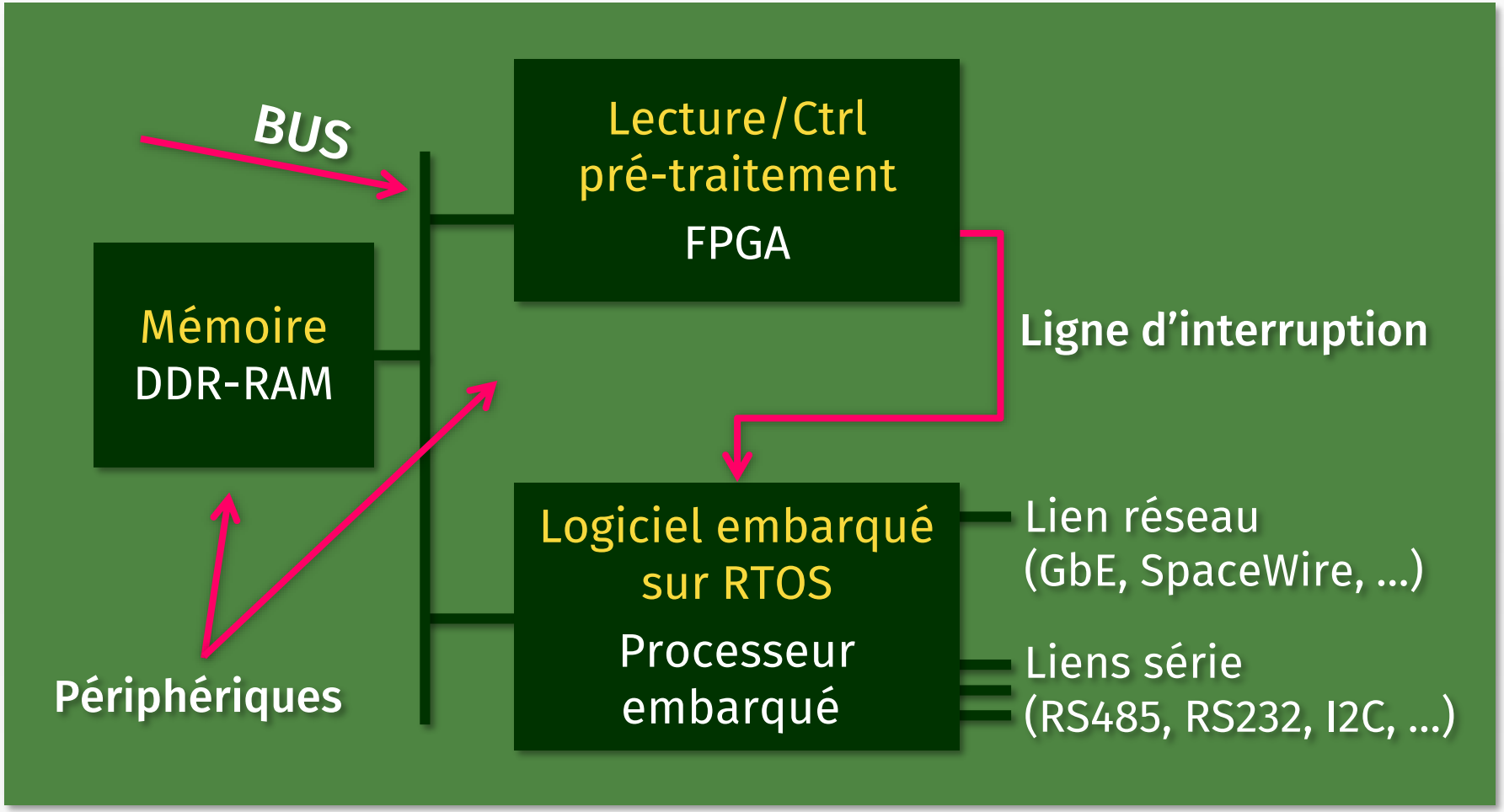
ffffffdc, -36

Nombre opposé = complément à 2 :

-N ⇔ ~N+1

Interruption matérielle

- **L'autre moyen de communication entre un périphérique et le CPU est l'interruption matérielle. C'est le seul moyen communication qui soit à l'initiative du périphérique.**



Interruption matérielle

- **L'autre moyen de communication entre un périphérique et le CPU est l'interruption matérielle. C'est le seul moyen communication qui qui soit à l'initiative du périphérique.**
- **Son fonctionnement exact dépend de l'architecture matérielle de la carte et des liaisons entre le périphérique et les broches d'interruption du CPU.**
- **Le CPU associe à une broche une fonction d'interruption. Dans cette fonction, l'interruption est traitée. Si le CPU est animé par un OS :**
  - Tous les appels susceptibles d'être bloquants sont interdits au sein de la fonction d'interruption.
  - Le temps d'exécution de l'interruption doit être minimisé.
  - Tout traitement lourd doit être effectué par une tâche standard en attente de libération d'un sémaphore. Typiquement, ce sémaphore est libéré par la fonction d'interruption.