LE PIPELINAGE

Chapitre 3

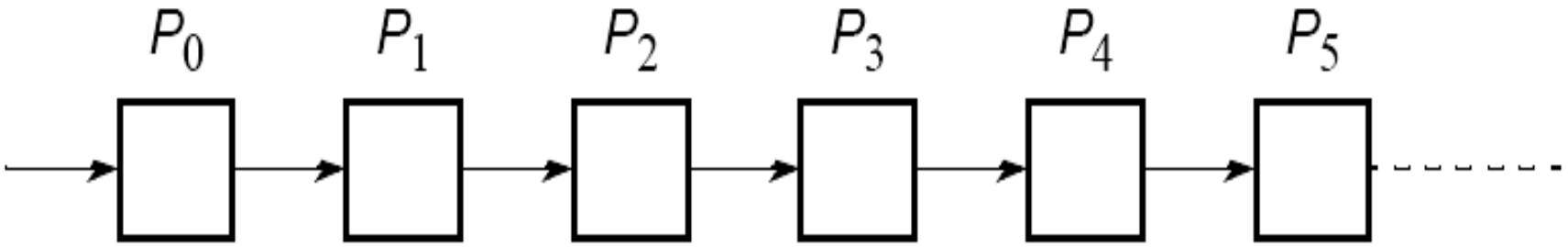
Programmation Parallèle – 3 Ing Info

Plan

1. Principe
2. Types
3. Exemples

Principe

- Décomposer l'opération à réaliser en une série d'opérations successives exécutées séparément.
- Chaque opération s'appelle étage.



Principe

- **Exemple 1 :**
- **Exécution d'une instruction**
- **Jeux d'instruction de J.V Neumann**
  - **F : Fetch**
  - **D : Decode**
  - **R : Read**
  - **E : Execute**
  - **W : Write**

Principe

- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:** 

Temps d'exécution = 5* Te avec Te : temps élémentaire par étage

Principe

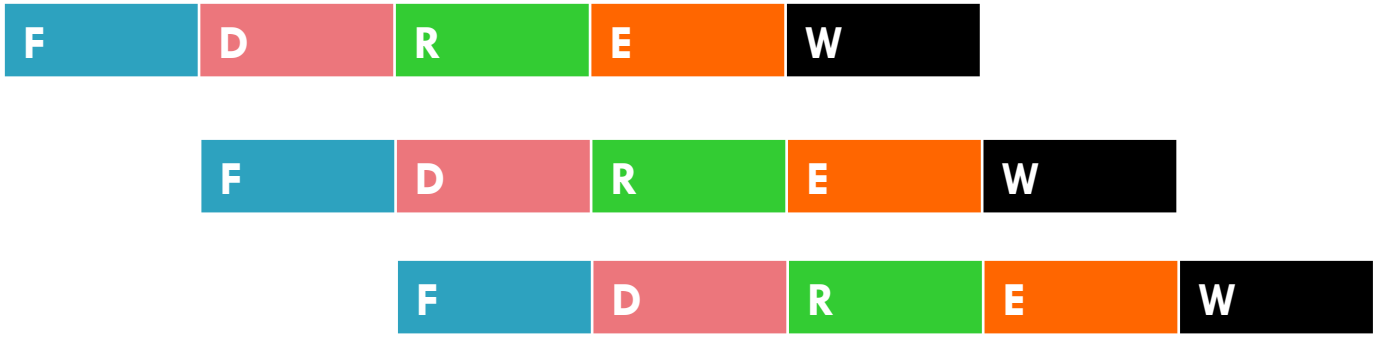
- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:**

- **Inst 2 :**

Temps d'exécution = 5* Te + Te

Principe

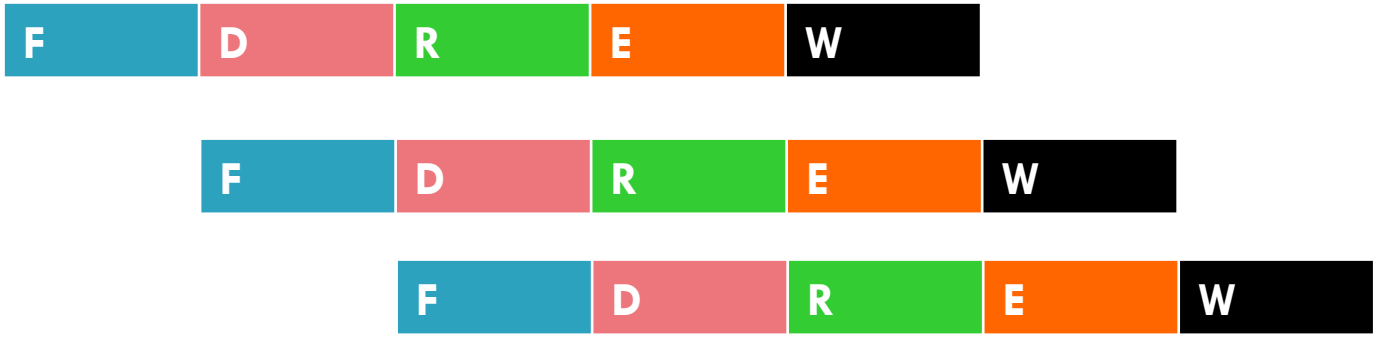
- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:**
- **Inst 2 :**
- **Inst 3 :**



Temps d'exécution = 5* Te + Te *2

Principe

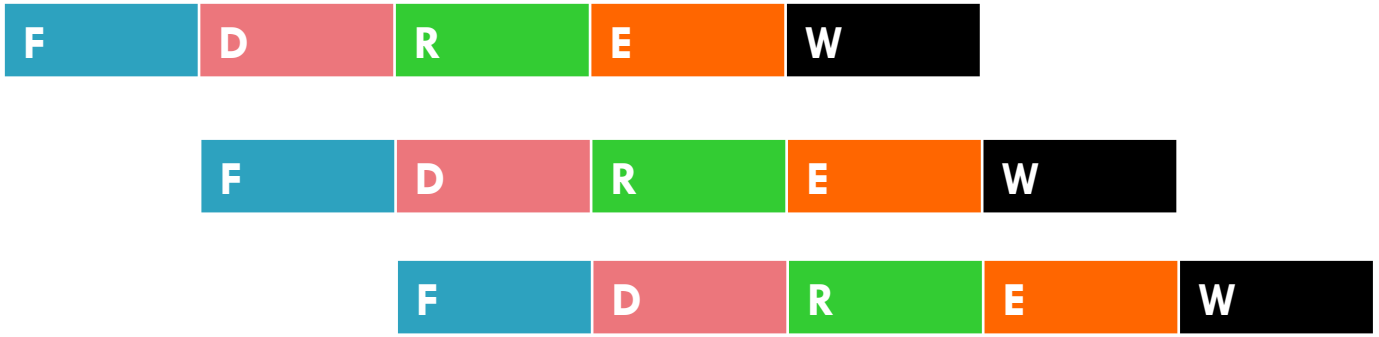
- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:**
- **Inst 2 :**
- **Inst 3 :**
- **Inst n :**



Temps d'exécution = 5* Te + Te * (n-1)

Principe

- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:**
- **Inst 2 :**
- **Inst 3 :**
- **Inst n :**

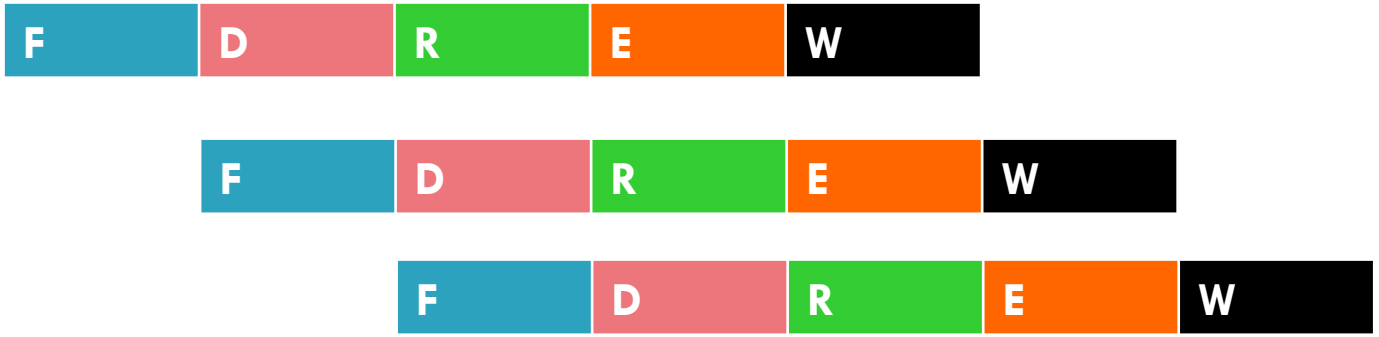


En général avec k étages

Temps d'exécution = K* Te + Te * (n-1)

Principe

- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:**
- **Inst 2 :**
- **Inst 3 :**
- **Inst n :**

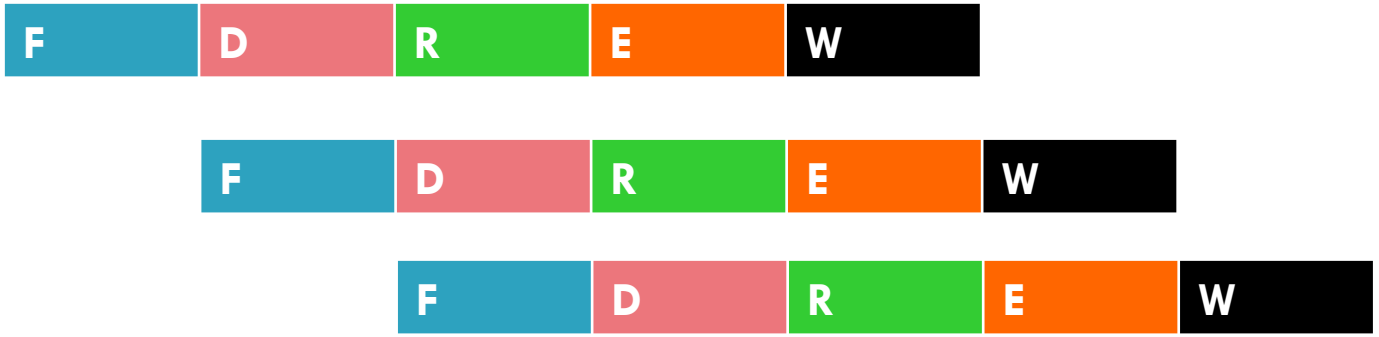


En général avec k étages

Temps d'exécution = (K-1)* Te + Te * n

Principe

- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:**
- **Inst 2 :**
- **Inst 3 :**
- **Inst n :**



En général avec k étages

Temps d'exécution = (K-1)* Te + Te * n

Principe

- **Exemple 1 :**
- **Exécution d'une instruction**
- **Pipelinage :**
- **Inst 1:** 

- **Inst 2 :** 

- **Inst 3 :** 

- **Inst n :**

En général avec k étages

(K-1)* Te : **temps d'amorce (pipeline non productif)**

Principe

- **Exemple 2 :**
- **Ajouter tous les éléments d'un tableau A à une somme de cumul:**

```
for (i = 0; i < n; i++)
                    sum = sum + A[i];
```

- **Cette boucle peut être décortiquée pour donner:**

```
sum = sum + A[0];
sum = sum + A[1];
sum = sum + A[2];
…
sum = sum + A[n-1];
```

Principe

- **Exemple 2 :**

Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension)

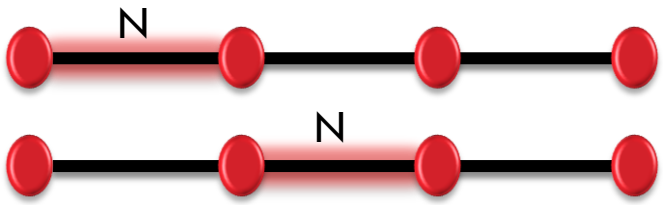
- **1ère étape**

Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension)

- **1ère étape**
- **2ème étape**

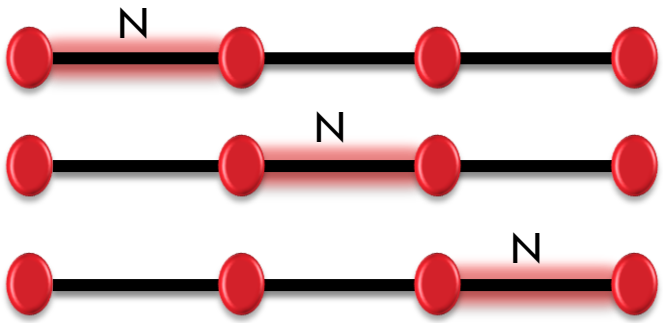


Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension)

- **1ère étape**
- **2ème étape**
- **3ème étape**

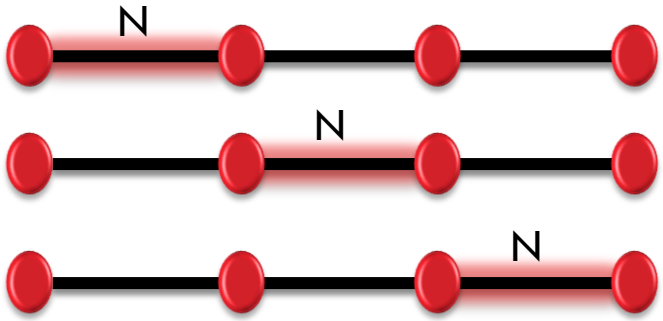


Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension)

- **1ère étape**
- **2ème étape**
- **3ème étape**



$$T_{comm} = 3*(\beta + N * \tau)$$

Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension) /pipeline

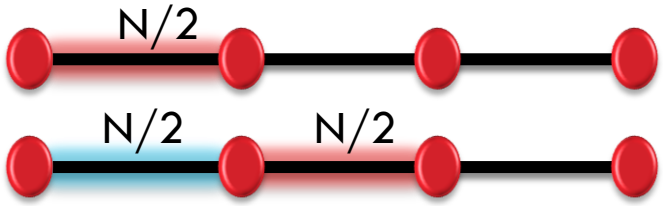
- **1ère étape**

Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension) /pipeline

- **1ère étape**
- **2ème étape**

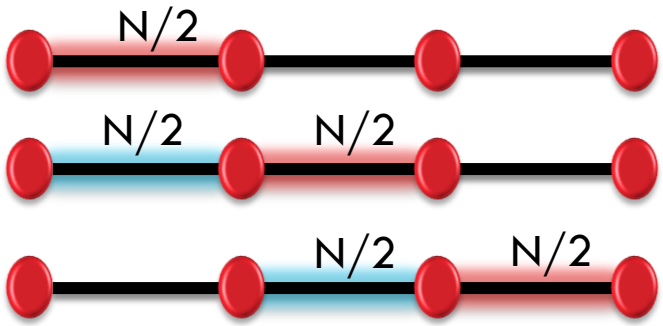


Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension) /pipeline

- **1ère étape**
- **2ème étape**
- **3ème étape**

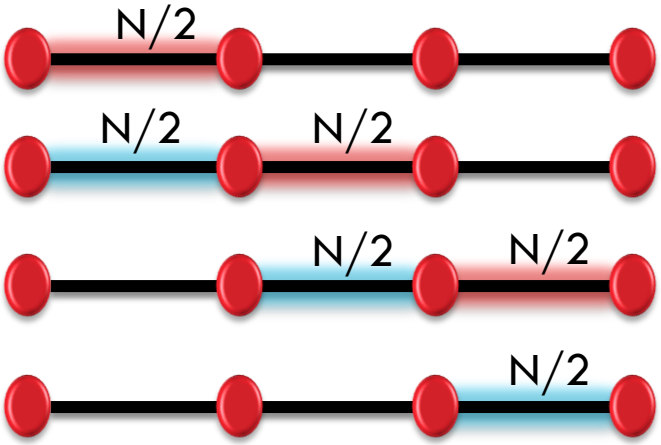


Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension) /pipeline

- **1ère étape**
- **2ème étape**
- **3ème étape**
- **4ème étape**

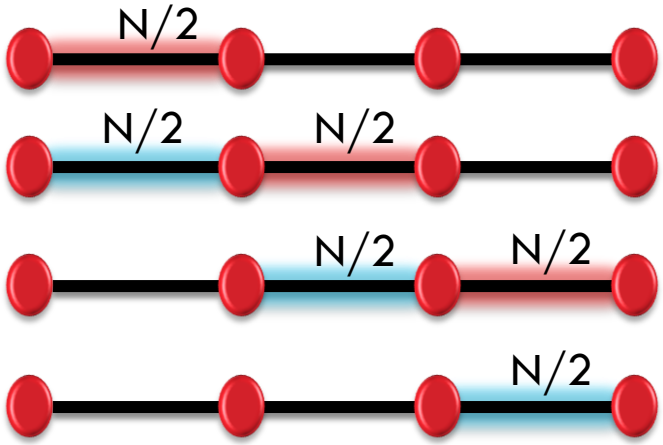


Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension) /pipeline

- **1ère étape**
- **2ème étape**
- **3ème étape**
- **4ème étape**



$$T_{comm} = 4*(\beta + N/2 * \tau)$$

Principe

- **Exemple 3 :**

Diffusion dans un réseau linéaire (grille à une dimension) /pipeline

$T_{comm} = 3*(\beta + N * \tau)$ $T_{comm} = 4*(\beta + N/2 * \tau)$

Dans la majorité des cas $\beta << \tau$

L'utilisation du pipeline est plus intéressante

Types

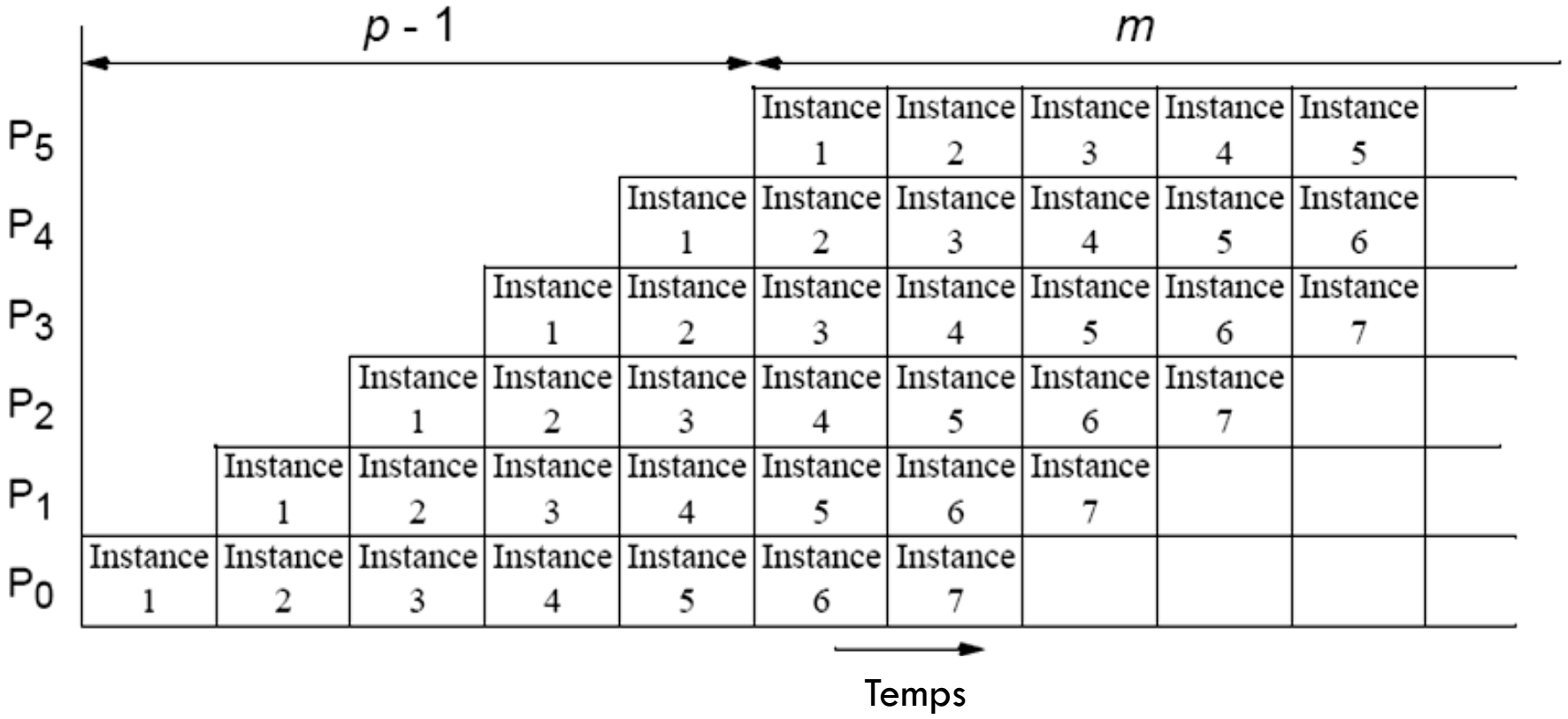
- **Type 1 : plusieurs instances d'un même programme**

C'est le SPMD

Types

- **Type 1 : plusieurs instances d'un même programme**

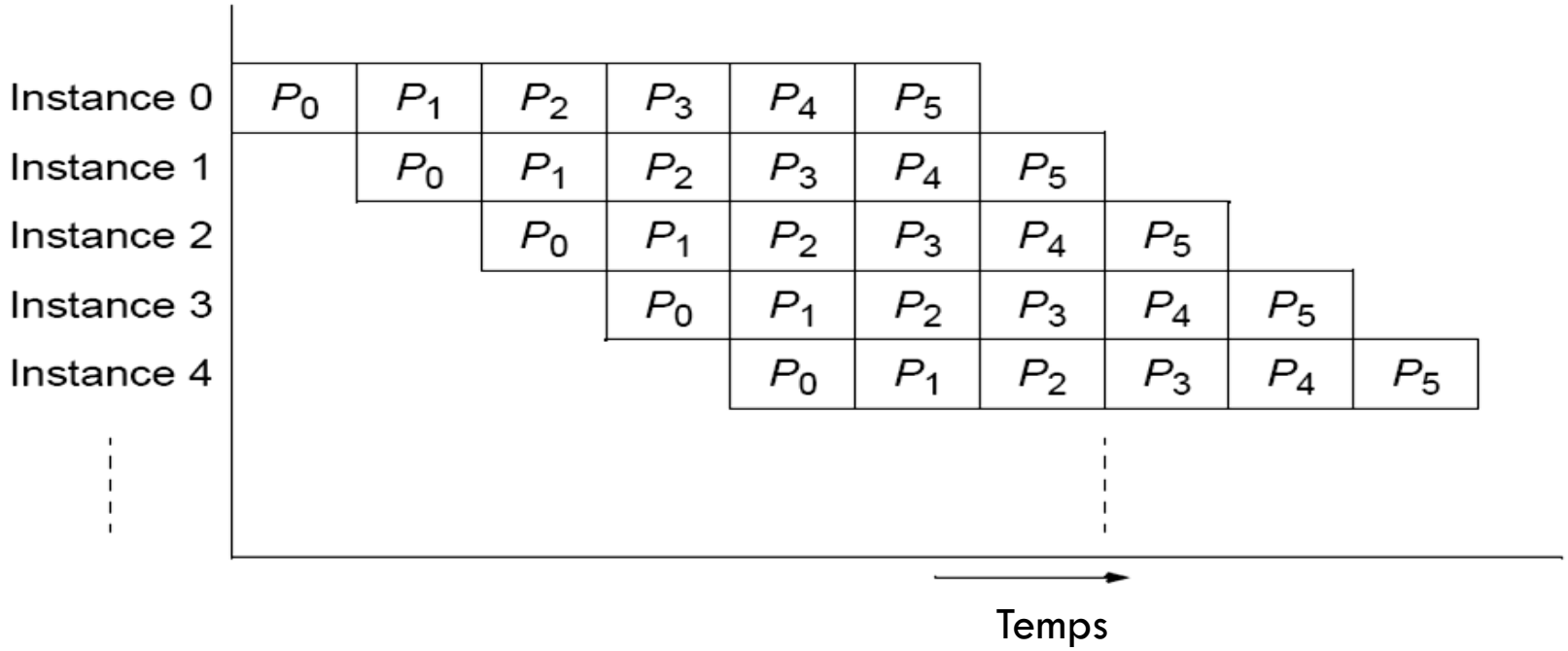
Diagramme de Gantt (espace-temps)



Types

- **Type 1 : plusieurs instances d'un même programme**

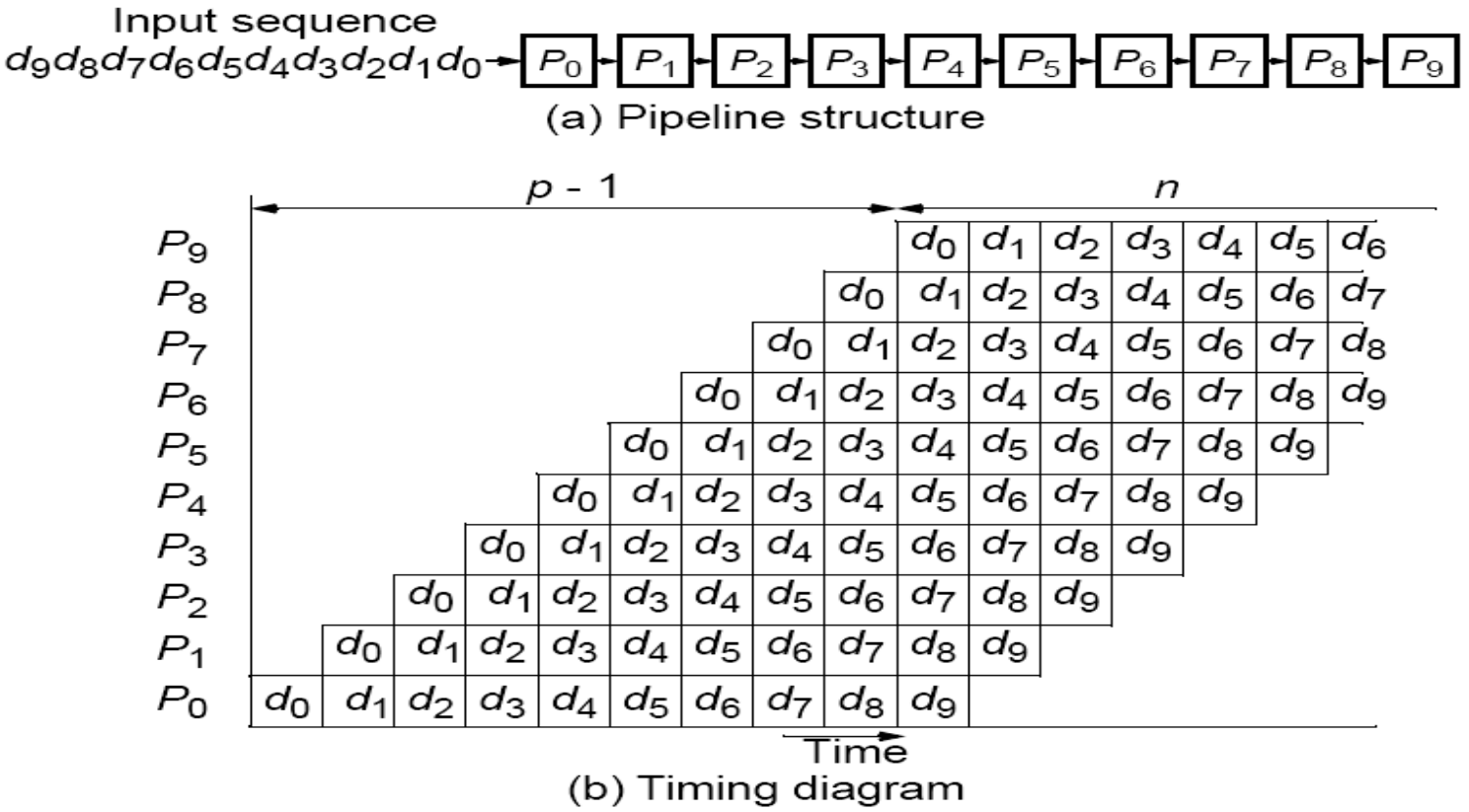
Espace-temps alternatif



Types

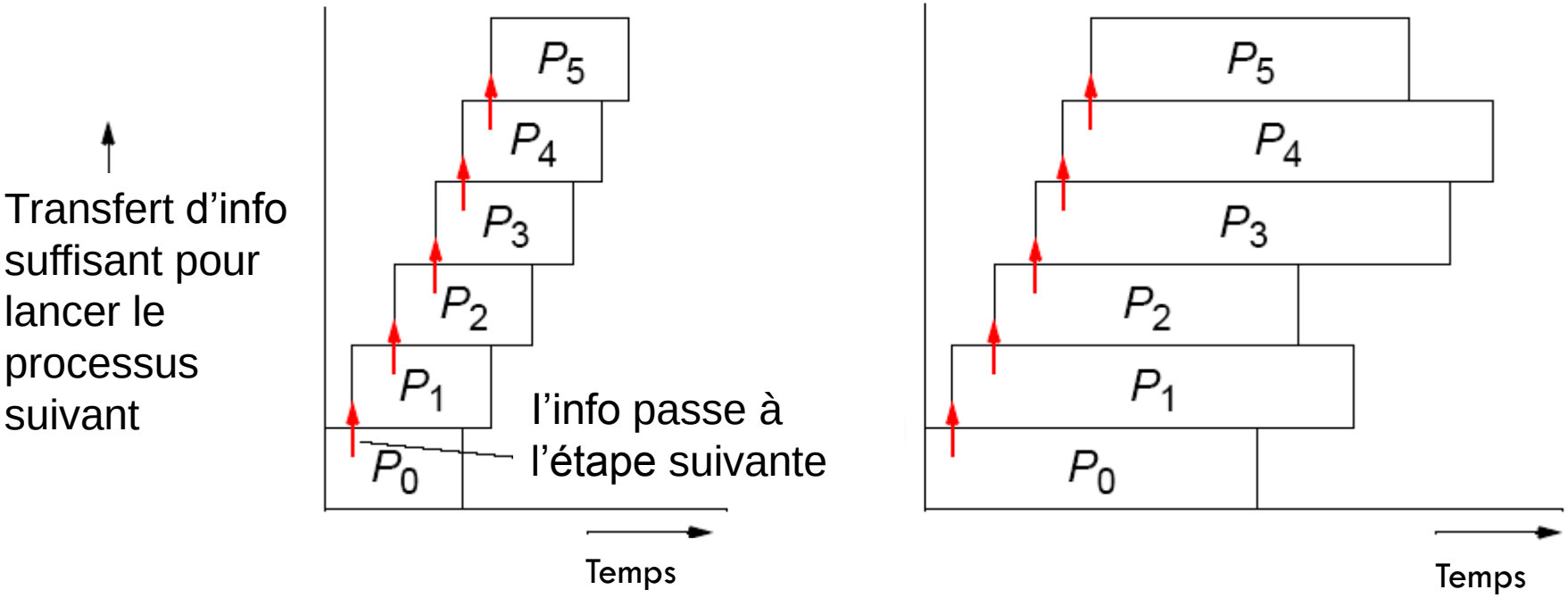
- **Type 2 : une série des données exige des opérations multiples**

Espace-temps



Types

- **Type 3 : dès qu'une information est prête, on la transmet à l'étage suivant**



(a) Processus avec temps d'executions egaux

(b) Processus avec temps d'executions differents

Types

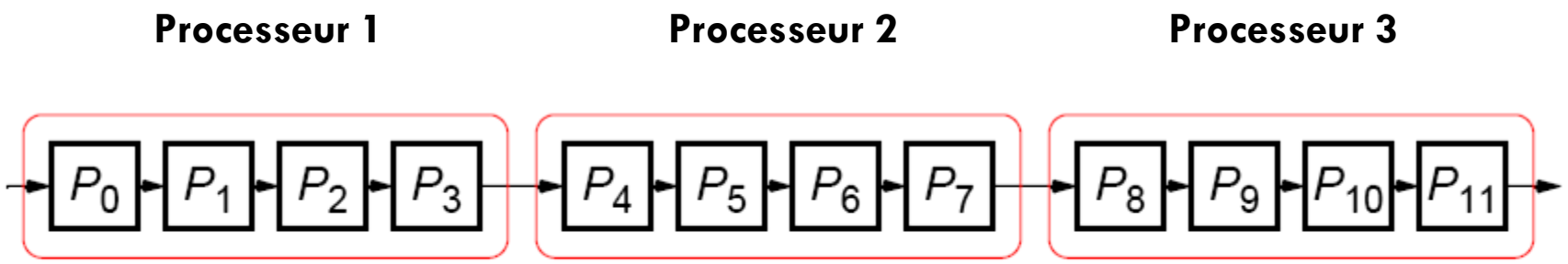
- **Remarque**

Si le nombre d'étapes est plus large que le nombre de processeurs dans un pipeline,

Types

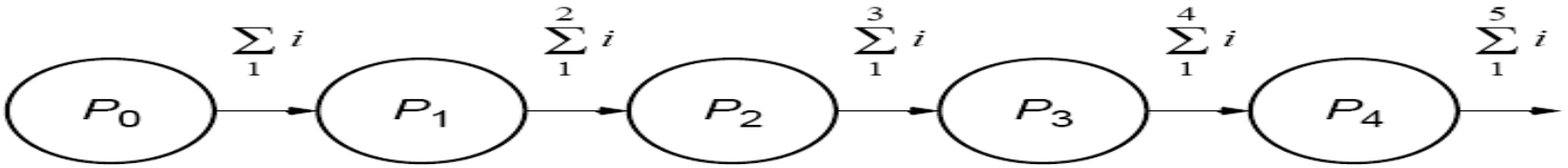
- **Remarque**

Si le nombre d'étapes est plus large que le nombre de processeurs dans un pipeline, un groupe d'étapes peut être affecter à chaque processeur.



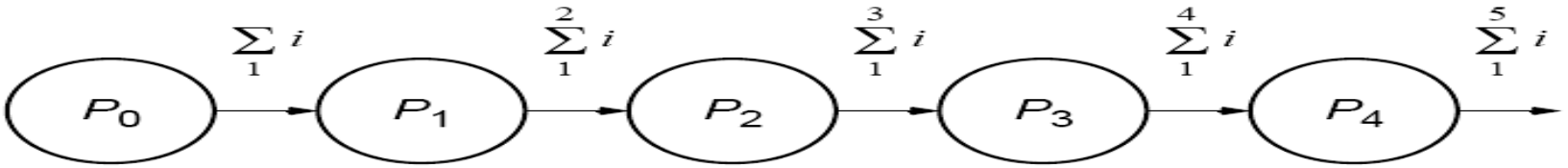
Exemples

- **Cumul des nombres**



Exemples

- **Cumul des nombres**

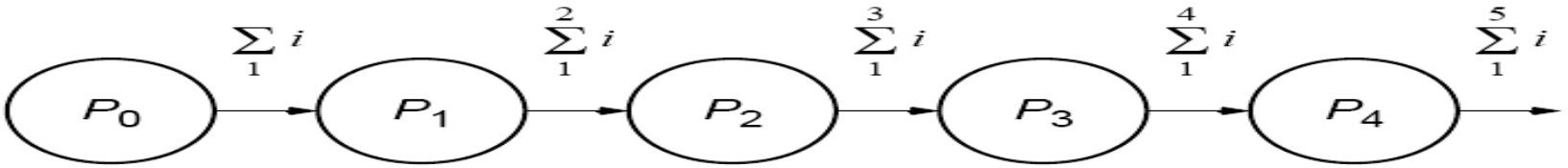


Code de base pour le processus P_i :

```
recv(&accumulation, Pi-1);
accumulation = accumulation + number;
send(&accumulation, Pi+1);
```

Exemples

- **Cumul des nombres**

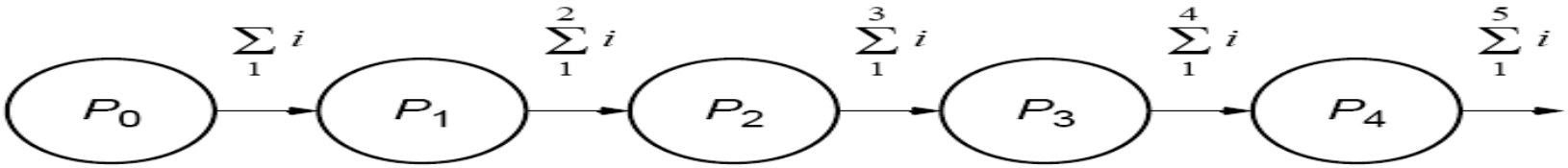


Sauf pour le 1er processus, P_0, qui a pour code

```
accumulation = accumulation + number;
send(&number, P1);
```

Exemples

- **Cumul des nombres**



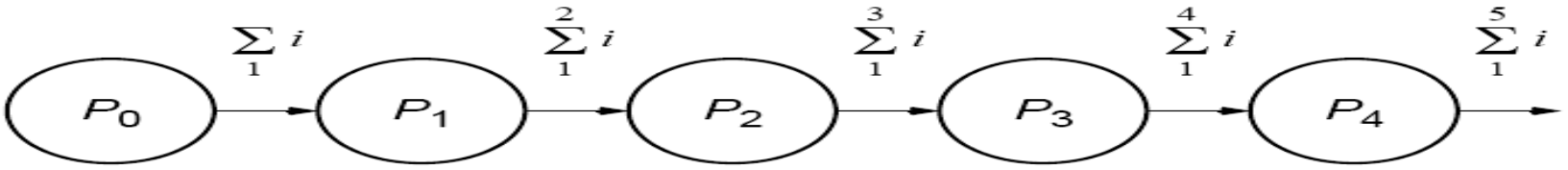
Et le dernier processus, P_{n-1}, qui a pour code

```
recv(&number, Pn-2);
accumulation = accumulation + number;
```

Exemples

- Cumul des nombres

Type 1 SPMD




```
if (process > 0)
        recv(accumulation, Pi-1);

accumulation = accumulation + number;

if (process < n-1)
        send(accumulation, Pi+1);
```