# MÉTHODOLOGIE DE PARALLÉLISATION DES TÂCHES DANS UN ENVIRONNEMENT HOMOGÈNE

APP– 3 Ing Info Systèmes Embarqués

**Chapitre 4**

**Elaboré par: Raja TLILI-KHEMIRI**

# Plan



1. Notion de système de tâches
2. Graphe de précédence
3. Ordonnancement de tâches

# Notion de système de tâches



## 1. Notion de tâche

**La décomposition (segmentation) d'un programme en tâches n'est pas unique. Elle dépend de la granularité demandée (taille de tâche).**

# Notion de système de tâches



## 1. Notion de tâche

**La décomposition (segmentation) d'un programme en tâches n'est pas unique. Elle dépend de la granularité demandée (taille de tâche).**

```
pour i=1 à n faire
        pour j=1 à n faire
                Y(i)= Y(i)+ A(i,j)* X(j)
        fin pour j
fin pour i
```

# Notion de système de tâches



## 1. Notion de tâche

**La décomposition (segmentation) d'un programme en tâches n'est pas unique. Elle dépend de la granularité demandée (taille de tâche).**

```
pour i=1 à n faire
        pour j=1 à n faire
                Y(i)= Y(i)+ A(i,j)* X(j)
        fin pour j
fin pour i
```

| Tâche | Granularité |
|---|---|
| Y(i)= Y(i)+ A(i,j)* X(j) | Moyenne (=2) |
| Boucle interne | Grande (=2n) |

# Notion de système de tâches



## 1. Notion de tâche

**La décomposition (segmentation) d'un programme en tâches n'est pas unique. Elle dépend de la granularité demandée (taille de tâche choisie).**

```
pour i=1 à n faire
        pour j=1 à n faire
                Y(i)= Y(i)+ A(i,j)* X(j)
        fin pour j
fin pour i
```

**Une granularité fine correspond à une seule opération élémentaire**

**$\rightarrow 2n^2$ tâches**

# Notion de système de tâches



## 1. Notion de tâche

**La décomposition (segmentation) d'un programme en tâches n'est pas unique. Elle dépend de la granularité demandée (taille de tâche).**

```
pour i=1 à n faire
        pour j=1 à n faire
                Y(i)= Y(i)+ A(i,j)* X(j)
        fin pour j
fin pour i
```

**Plus la granularité est fine, plus nous pouvons faire apparaitre le parallélisme s'il existe.**

# Notion de système de tâches



## 1. Notion de tâche

**Tâche c'est une unité indivisible définie par les propriétés suivantes**

$E_i$ : les entrées de la tâche $T_i$

$S_i$ : les sorties de la tâches $T_i$

$F_i$ : opération sur $T_i$; $(E_i) \rightarrow S_i$

$\Theta_i$ :coût de $T_i$ (nombre d'opérations élémentaires) $P_i$

**Ainsi la tâche est représentée par** $(E_i , S_i , F_i, \Theta_i)$

# Notion de système de tâches



## 1. Notion de tâche

**Remarques**

- **La tâche ne peut s'exécuter que sur un seul processeur**
- **L'exécution d'une tâche est non préemptive**
- **Le modèle est dit statique si on connait à l'avance de ($E_i$ , $S_i$ , $F_i$, $\Theta_i$)**
- **PS le programme séquentiel, une fois segmenté PS=($T_1$,…,$T_n$,$\rightarrow$)**

  **« $\rightarrow$ » relation d'ordre total puisque le programme est séquentiel**

# Notion de système de tâches



## 2. Système de tâches

**Un système de tâches S =($T_1$,…,$T_n$,<<) est un ensemble de tâches muni d'une relation d'ordre partiel notée <<.**

**$T_i$ << $T_j$ ($i \neq j$) signifie que l'exécution de la tâche $T_j$ ne peut commencer que si $T_i$ a terminé son exécution**

# Notion de système de tâches



## 2. Système de tâches

**Un système de tâches $S = (T_1, \ldots, T_n, <<)$ est un ensemble de tâches muni d'une relation d'ordre partiel notée $<<$.**

**$T_i << T_j$ $(i \neq j)$ signifie que l'exécution de la tâche $T_j$ ne peut commencer que si $T_i$ a terminé son exécution**

**Comparaison de deux tâches :**

**Deux tâches $T_i$ et $T_j$ sont indépendantes (non interférentes) si**

$$S_i \cap S_j = S_i \cap E_j = E_i \cap S_j = \emptyset$$

**C'est la relation de Bernstein**

# Notion de système de tâches



## 2. Système de tâches

**Un système de tâches S =($T_1$,…,$T_n$,<<) est un ensemble de tâches muni d'une relation d'ordre partiel notée <<.**

**$T_i$ << $T_j$ ($i \neq j$) signifie que l'exécution de la tâche $T_j$ ne peut commencer que si $T_i$ a terminé son exécution**

**Comparaison de deux tâches :**

**Deux tâches $T_i$ et $T_j$ sont indépendantes (non interférentes) si**

**$S_i \cap S_j = S_i \cap E_j = E_i \cap S_j = \emptyset$ : c'est une condition nécessaire mais non suffisante pour que les deux tâches puissent s'exécuter en //.**

# Notion de système de tâches



## 2. Système de tâches

$S=(T_1,\ldots,T_n,<<)$ :

- $T_i << T_j$
- $T_j << T_i$
- $T_i$ et $T_j$ indépendantes

**Remarques**

- $T_i$ est prédécesseur immédiat de $T_j$ (ou $T_j$ est successeur immédiat de $T_i$) s'il n'existe aucune autre tâche $T_k$ telle que $T_i<<T_k<<T_j$.
- Cette relation d'ordre partiel $<<$ , est transitive mais non symétrique.

# Notion de système de tâches



## 2. Système de tâches

$S=(T_1,\ldots,T_n,\ll)$ :

**pour tout couple de tâches Ti et Tj, on a l'un des 3 cas suivants :**

- $T_i \ll T_j$
- $T_j \ll T_i$
- $T_i$ et $T_j$ indépendantes

**Remarques**

- **Cette relation d'ordre partiel << , est transitive mais non symétrique.**
- $T_i \ll T_j \Leftrightarrow T_i \rightarrow T_j$ et $(S_i \cap S_j) \cup (S_i \cap E_j) \cup (E_i \cap S_j) \neq \emptyset$

# Graphe de précédence



## 1. Graphe de tâches

GT: $T_i \longrightarrow T_j$ si $T_i << T_j$

**Remarques**

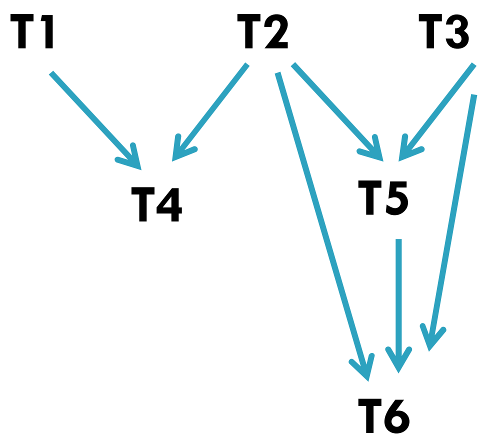
- **GT (graphe de tâches) est un graphe orienté non cyclique (car les arcs suivent tous le même sens, celui de l'ordre séquentiel)**
- **On associe à chaque programme un graphe de tâches**

# Graphe de précédence



## 1. Graphe de tâches

- Exp
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $T_2 << T_6$
- $T_3 << T_5$ , $T_3 << T_6$
- $T_5 << T_6$

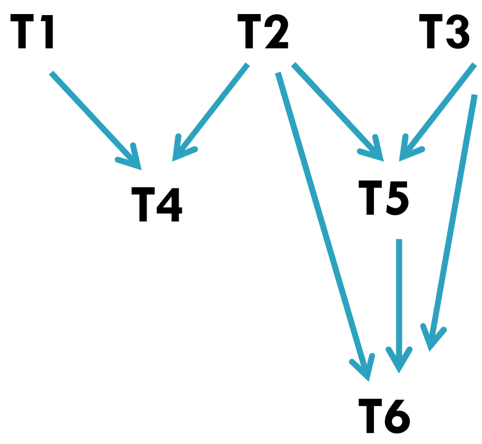


# Graphe de précédence



## 2. Graphe de précédence

- Pour construire le GP il faut éliminer les redondances du GT.
- Exp
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $T_2 << T_6$
- $T_3 << T_5$ , $T_3 << T_6$
- $T_5 << T_6$

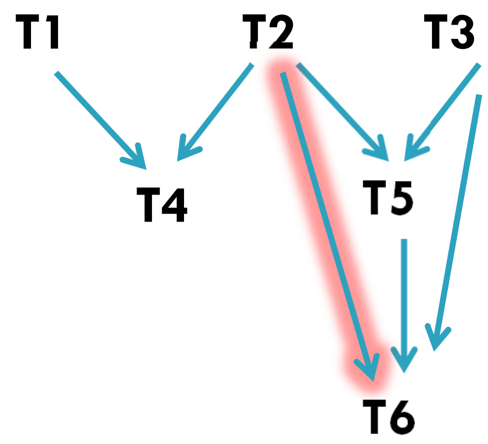


# Graphe de précédence



## 2. Graphe de précédence

- Pour construire le GP il faut éliminer les redondances du GT.
- Exp
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $T_2 << T_6$
- $T_3 << T_5$ , $T_3 << T_6$
- $T_5 << T_6$

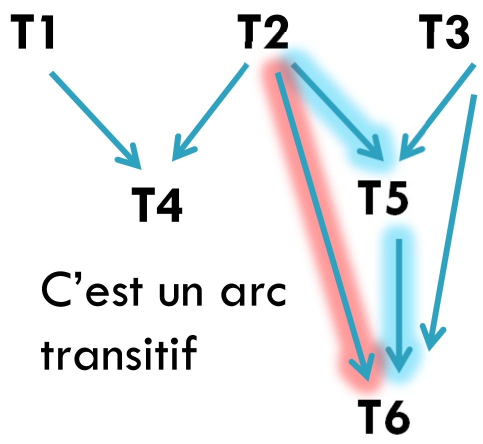


# Graphe de précédence



## 2. Graphe de précédence

- **Pour construire le GP il faut éliminer les redondances du GT.**
- **Exp**
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $T_2 << T_6$
- $T_3 << T_5$ , $T_3 << T_6$
- $T_5 << T_6$

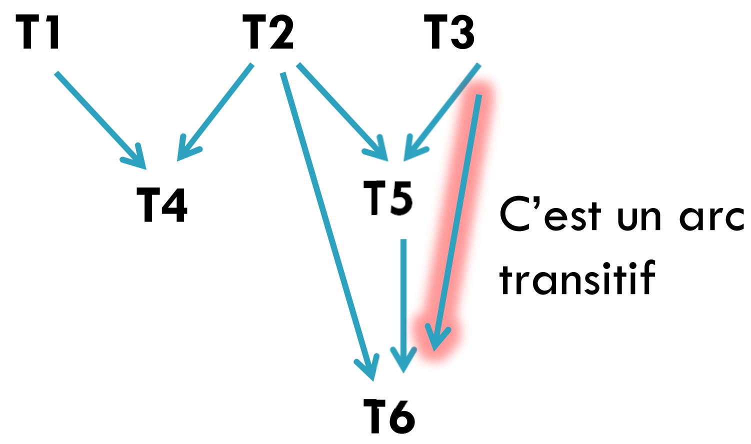


# Graphe de précédence



## 2. Graphe de précédence

- Pour construire le GP il faut éliminer les redondances du GT.
- Exp
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $T_2 << T_6$
- $T_3 << T_5$ , $T_3 << T_6$
- $T_5 << T_6$

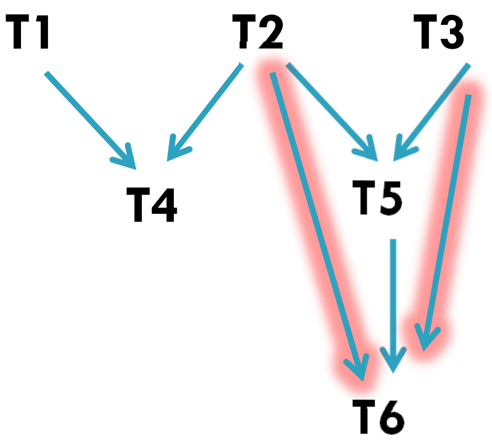


# Graphe de précédence



## 2. Graphe de précédence

- Pour construire le GP il faut éliminer les redondances du GT.
- Exp
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $\cancel{T_2 << T_6}$
- $T_3 << T_5$ , $\cancel{T_3 << T_6}$
- $T_5 << T_6$

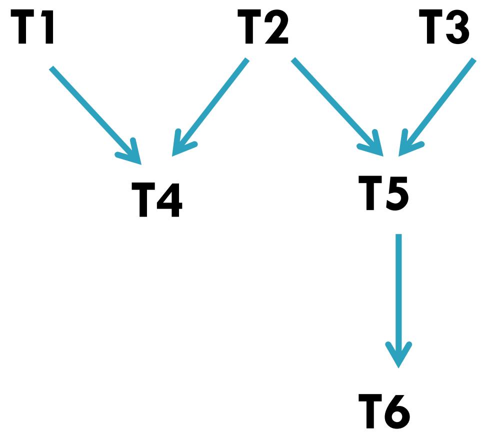


# Graphe de précédence



## 2. Graphe de précédence

- Pour construire le GP il faut éliminer les redondances du GT.
- Exp
- $T_1 << T_4$
- $T_2 << T_4$ , $T_2 << T_5$ , $\cancel{T_2 << T_6}$
- $T_3 << T_5$ , $\cancel{T_3 << T_6}$
- $T_5 << T_6$




**Il faut donc supprimer les arcs transitifs pour avoir le GP.**

# Graphe de précédence



## 2. Graphe de précédence

### ❖ Remarques

- GT : graphe de tâches
- GP : graphe de précédence
- GFT : graphe de fermeture transitive

# Graphe de précédence



## 2. Graphe de précédence

❖ **Remarques**

- **GT : graphe de tâches**
- **GP : graphe de précédence : on supprime tous les arcs transitifs**
- **GFT : graphe de fermeture transitive : on complète tous les arcs transitifs**

# Graphe de précédence



## 2. Graphe de précédence

❖ **Remarques**

- **GT : graphe de tâches**
- **GP : graphe de précédence**
- **GFT : graphe de fermeture transitive**

**la relation de Bernstein : $S_i \cap S_j = S_i \cap E_j = E_i \cap S_j = \emptyset$**

**pour savoir si deux tâches peuvent s'exécuter en //.**

# Graphe de précédence



## 2. Graphe de précédence

❖ **Remarques**

- **GT : graphe de tâches**
- **GP : graphe de précédence**
- **GFT : graphe de fermeture transitive**

**La connaissance de GFT rends nécessaire et suffisante**

**la relation de Bernstein : $S_i \cap S_j = S_i \cap E_j = E_i \cap S_j = \emptyset$**

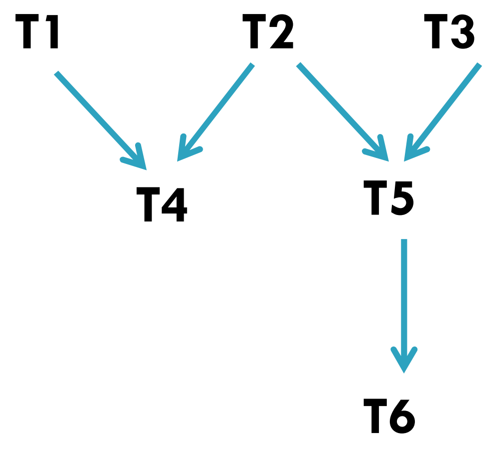
**pour savoir si deux tâches peuvent s'exécuter en //.**

# Graphe de précédence



## 3. Matrice d'adjacence

$S=(T_1,\dots,T_n,<<) \rightarrow GP$



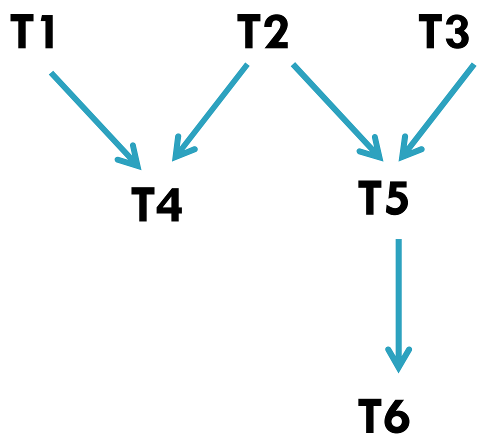

- A matrice d'adjacence d'ordre n
- $a_{ij}$ =1 ssi $T_i<<T_j$

# Graphe de précédence



## 3. Matrice d'adjacence

$S=(T_1,\ldots,T_n,<<) \rightarrow GP$




- A matrice d'adjacence d'ordre n
- $a_{ij}=1$ ssi $T_i<<T_j$

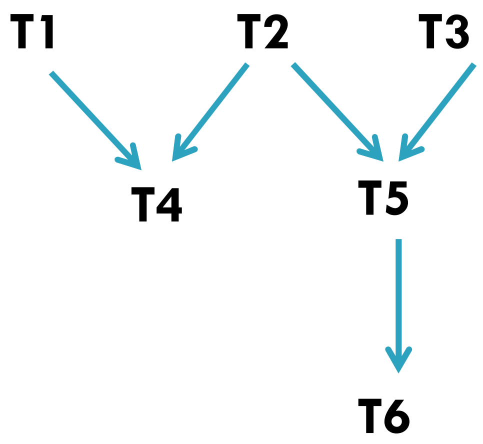
$$\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

# Graphe de précédence



## 3. Matrice d'adjacence

$S=(T_1,\dots,T_n,<<) \rightarrow GP$




- A matrice d'adjacence d'ordre n
- C'est une matrice creuse

$$\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

# Graphe de précédence



## 3. Matrice d'adjacence

**Il existe un algorithme dit tri topologique qui permet de numéroter les nœuds d'un graphe pour avoir : si $T_i << T_j \Rightarrow i<j$**

# Graphe de précédence



## 3. Matrice d'adjacence

**Il existe un algorithme dit tri topologique qui permet de numéroter les nœuds d'un graphe pour avoir : si $T_i << T_j \Rightarrow i<j$**

**La matrice A sera une matrice triangulaire supérieure**

# Graphe de précédence



## 3. Matrice d'adjacence

**Il existe un algorithme dit tri topologique qui permet de numéroter les nœuds d'un graphe pour avoir : si $T_i << T_j \Rightarrow i<j$**

**La matrice A sera une matrice triangulaire supérieure**

**Si la matrice A a des 1 sur toute la sur-diagonale ($T_1<<T_2<<\ldots<<T_n$) alors le programme est séquentiel**

# Graphe de précédence



## 3. Matrice d'adjacence

Il existe un algorithme dit tri topologique qui permet de numéroter les nœuds d'un graphe pour avoir : si $T_i << T_j \Rightarrow i<j$

La matrice A sera une matrice triangulaire supérieure

Si la matrice A a des 1 sur toute la sur-diagonale ($T_1<<T_2<<…<<T_n$) alors le programme est séquentiel

Ainsi, nous devons comparer i avec i+1, .., n pour avoir une information sur l'ordre et l'indépendance des tâches.

# Graphe de précédence



## 3. Matrice d'adjacence

Il existe un algorithme dit **tri topologique** qui permet de numéroter les nœuds d'un graphe pour avoir : si $T_i << T_j \Rightarrow i<j$

La matrice A sera une matrice triangulaire supérieure

Si la matrice A a des 1 sur toute la sur-diagonale ($T_1 << T_2 << \ldots << T_n$) alors le programme est séquentiel

Ainsi, nous devons comparer i avec i+1, .., n pour avoir une information sur l'ordre et l'indépendance des tâches.

Il est important aussi de préciser le **type de dépendance** sur le GP

# Graphe de précédence



## 4. Types de dépendance

$$T_i \quad : \quad S_i \quad = \quad F_i\,(E_i)$$

$$T_j \quad : \quad S_j \quad = \quad F_i\,(E_j)$$

# Graphe de précédence



## 4. Types de dépendance

$$T_i \quad : \quad S_i \quad = \quad E_i$$

$$T_j \quad : \quad S_j \quad = \quad E_j$$

# Graphe de précédence



**4. Types de dépendance**

$$T_i \quad : \quad S_i \quad = \quad E_i$$

$$T_j \quad : \quad S_j \quad = \quad E_j$$

**δ : Dépendance de flux**

**ou Dépendance Producteur-Consommateur**

**RAW (Read After Write)**

# Graphe de précédence



4. **Types de dépendance**

$T_i$ : $S_i = E_i$

$T_j$ : $S_j = E_j$

$\bar{\delta}$ : **Anti-Dépendance**

**ou Dépendance Consommateur-Producteur**

**WAR (Write After Read)**

# Graphe de précédence



## 4. Types de dépendance

$T_i \quad : \quad S_i \quad = \quad E_i$

$T_j \quad : \quad S_j \quad = \quad E_j$

**$\delta^{\circ}$: Dépendance de sortie**

**ou Dépendance Producteur-Producteur**

**WAW (Write After Write)**

# Graphe de précédence



## 4. Types de dépendance

**$\delta$ : communication**

**$\overline{\delta}$ : pas de communication**

**$\delta^{\circ}$ : pas de communication**

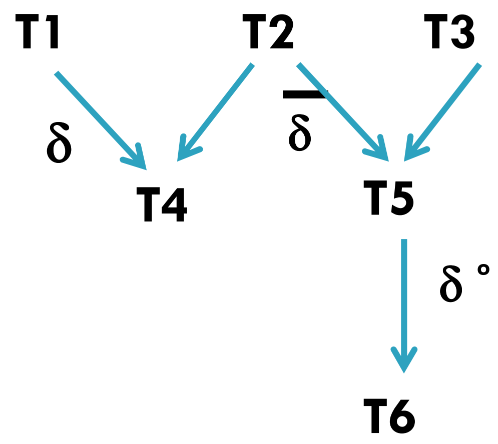
**Dans la pratique nous nous intéressons plus à la dépendance $\delta$ qui engendre des coûts de communication.**

# Graphe de précédence



## 4. Types de dépendance

**Sur le GP**

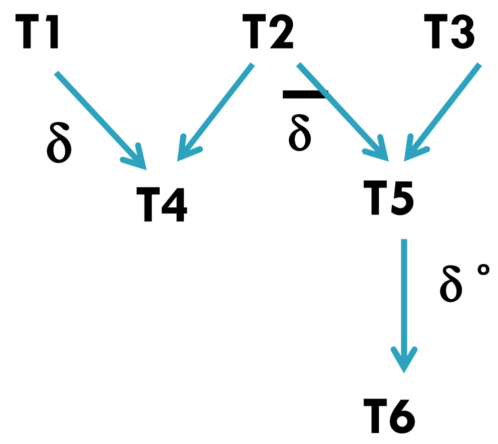


# Graphe de précédence



## 4. Types de dépendance

**Sur le GP**



# Ordonnancement des tâches



## 1. Généralités

**PS=($T_1$,...,$T_n$,$\rightarrow$)**

**S =($T_1$,...,$T_n$,<<)**

**Pr =($P_1$,...,$P_p$)**

**L' ordonnancement est définit comme suit ORD :**

$$T_i \xrightarrow{\quad\ominus\quad} (\,tdb(T_i\,),\ Proc(T_i))$$

**tdb($T_i$ )** : temps de début d'exécution de $T_i$

**Proc($T_i$)** : processeur qui va exécuter $T_i$

**$\theta_i$** : durée de la tâche $T_i$

# Ordonnancement des tâches



## 1. Généralités

$PS=(T_1,\ldots,T_n,\rightarrow)$

$S\ =(T_1,\ldots,T_n,\ll)$

$Pr=(P_1,\ldots,P_p)$

**L' ordonnancement est définit comme suit ORD :**

$$T_i \longrightarrow (\ tdb(T_i),\ Proc(T_i))$$

Si $T_i \ll T_j$ alors $tdb(T_i)+\theta_i \le tdb(T_j)$

Si $[tdb(T_i),tdb(T_i)+\theta_i\,] \cap [tdb(T_j),tdb(T_j)+\theta_j\,] \neq \emptyset$

alors $Proc\ (T_i) \neq Proc(T_j)$

# Ordonnancement des tâches



## 1. Généralités

**$PS=(T_1,\ldots,T_n,\rightarrow)$**

**$S=(T_1,\ldots,T_n,\ll)$**

**$Pr=(P_1,\ldots,P_p)$**

**L' ordonnancement est définit comme suit ORD :**

$$T_i \longrightarrow (\ tdb(T_i),\ Proc(T_i))$$

Si $Proc(T_i) = Proc(T_j)$ alors $tdb(T_i)+\theta_i \leq tdb(T_j)$

ou $tdb(T_j)+\theta_j \leq tdb(T_i)$

# Ordonnancement des tâches



## 1. Généralités

$PS=(T_1,\ldots,T_n,\rightarrow)$

$S =(T_1,\ldots,T_n,<<)$

$Pr =(P_1,\ldots,P_p)$ **p étant le nombre de processeurs**

**L' ordonnancement est définit comme suit ORD :**

$$T_i \xrightarrow{\quad\theta\quad} (\ tdb(T_i),\ Proc(T_i))$$

$T_p$ = **durée de l'exécution des n tâches (makespan) sur p processeurs**

= $\max_i \ [tdb(T_i)+\theta_i]$

# Ordonnancement des tâches



## 1. Généralités

**PS=$(T_1,\ldots,T_n,\rightarrow)$**

**S =$(T_1,\ldots,T_n,<<)$**

**Pr =$(P_1,\ldots,P_p)$**

**L' ordonnancement est définit comme suit ORD :**

$$T_i \longrightarrow (\ \mathrm{tdb}(T_i),\ \mathrm{Proc}(T_i))$$

**$T_p$ il faut le minimiser pour obtenir l'ORD optimal $T_{opt}$**

$$= \max_i \ [\mathrm{tdb}(T_i)+\theta_i]$$

# Ordonnancement des tâches



## 1. Généralités

**PS=($T_1,\ldots,T_n,\rightarrow$)**

**S =($T_1,\ldots,T_n,\ll$)**

**Pr =($P_1,\ldots,P_p$)**

**L' ordonnancement est définit comme suit ORD :**

$$T_i \longrightarrow ( tdb(T_i), Proc(T_i))$$

**$T_{opt}$ longueur du chemin critique (le plus long) dans GP**

**où la longueur du chemin est la somme des durées des tâches le constituant**

$$T_{opt} = \max_i \quad [tdb(T_i)+\theta_i]$$

# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

- **$tet_i$ = temps au plus tôt de la tâche i**
- **$teT_i$ = temps au plus tard de la tâche i**

# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

- $tet_i$ = temps au plus tôt de la tâche i
- $teT_i$ = temps au plus tard de la tâche i

### a) Cas de << vide

- $T_1$ : $\theta_1 = 3$, $tet_1 = 0$, $teT_1 = 3$
- $T_2$ : $\theta_2 = 4$, $tet_2 = 0$, $teT_2 = 2$
- $T_3$ : $\theta_3 = 6$, $tet_3 = 0$, $teT_3 = 0$

# Ordonnancement des tâches



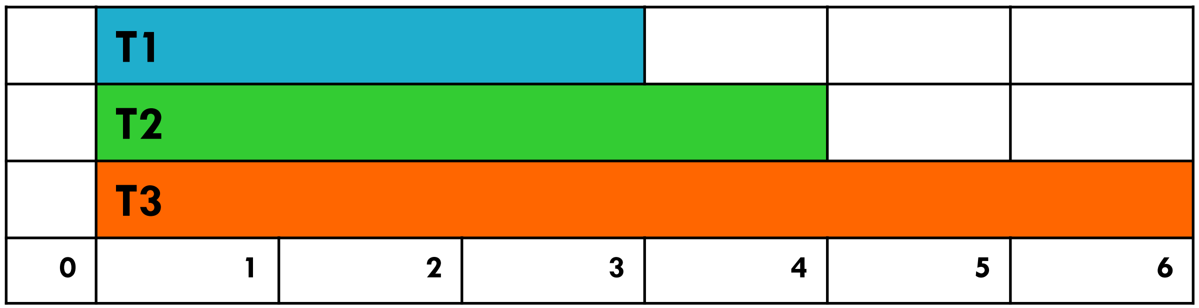
## 2. Détermination de temps au plus tôt et au plus tard

- $tet_i$ = temps au plus tôt de la tâche i
- $teT_i$ = temps au plus tard de la tâche i

### a) Cas de << vide

- $T_1$ : $\theta_1 = 3$, $tet_1 = 0$, $teT_1 = 3$
- $T_2$ : $\theta_2 = 4$, $tet_2 = 0$, $teT_2 = 2$
- $T_3$ : $\theta_3 = 6$, $tet_3 = 0$, $teT_3 = 0$

Au plus tôt



# Ordonnancement des tâches



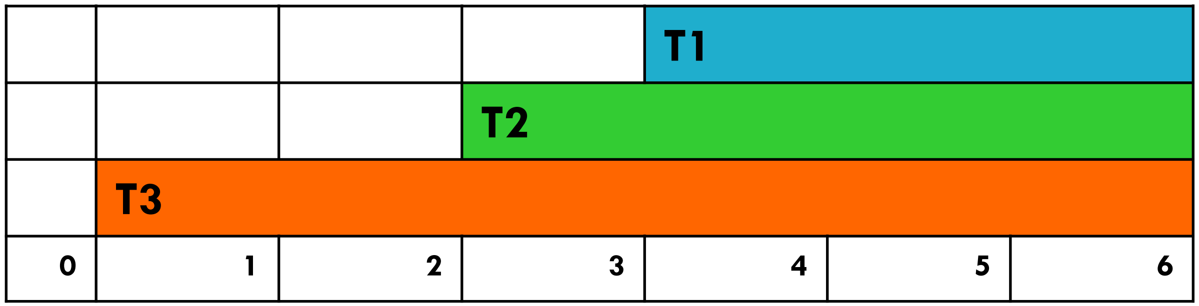
## 2. Détermination de temps au plus tôt et au plus tard

- $tet_i$ = temps au plus tôt de la tâche i
- $teT_i$ = temps au plus tard de la tâche i

### a) Cas de << vide

- $T_1$ : $\theta_1 = 3$, $tet_1 = 0$, $teT_1 = 3$
- $T_2$ : $\theta_2 = 4$, $tet_2 = 0$, $teT_2 = 2$
- $T_3$ : $\theta_3 = 6$, $tet_3 = 0$, $teT_3 = 0$

Au plus tard



# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

- $tet_i$ = temps au plus tôt de la tâche i
- $teT_i$ = temps au plus tard de la tâche i

### a) Cas de << vide

- $T_1$ : $\theta_1 = 3$, $tet_1 = 0$, $teT_1 = 3$
- $T_2$ : $\theta_2 = 4$, $tet_2 = 0$, $teT_2 = 2$
- $T_3$ : $\theta_3 = 6$, $tet_3 = 0$, $teT_3 = 0$

**Il existe 12 possibilités d'exécuter ces tâches avec un même coût $T_{opt} = 6$.**

# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

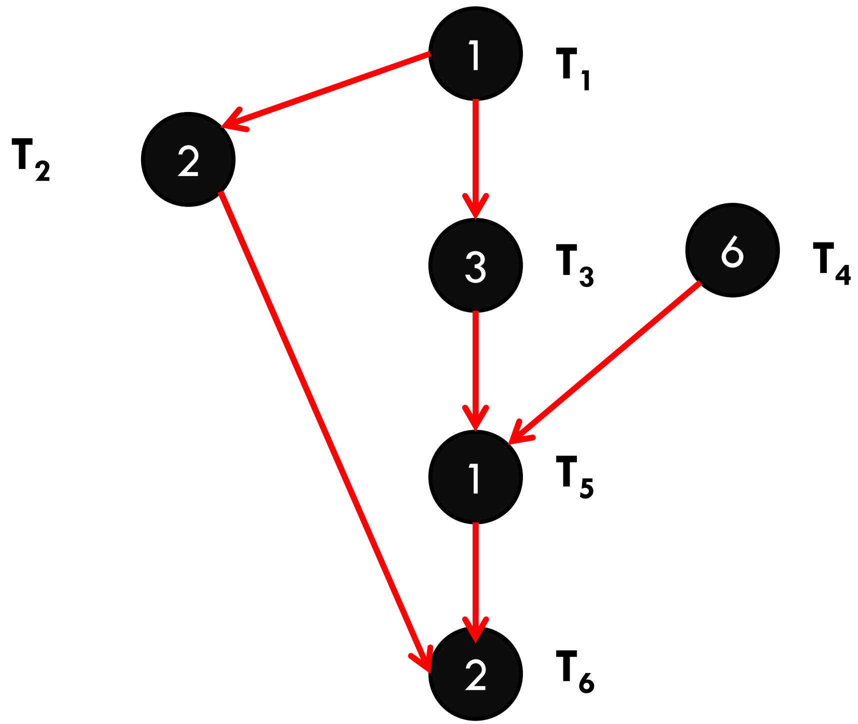
- **lorsqu'il ya dépendance entre les tâches, il faut déterminer $T_{opt}$ et déterminer pour chaque tâche tet et puis teT pour pouvoir définir l'ORD optimal.**

# Ordonnancement des tâches



2. **Détermination de temps au plus tôt et au plus tard**

b) **Cas général**



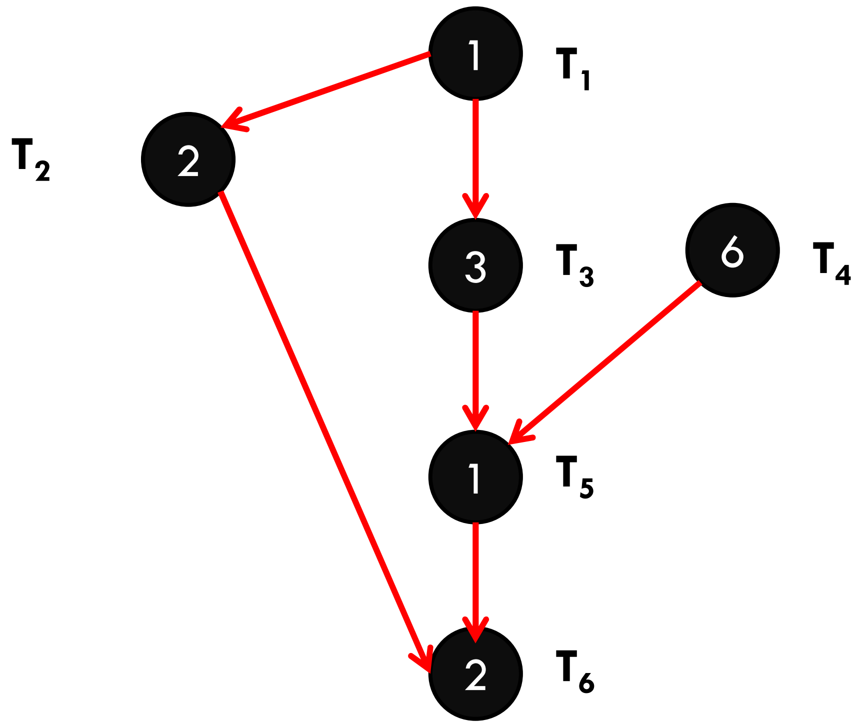
# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

**Matrice d'adjacence**




| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 |

# Ordonnancement des tâches

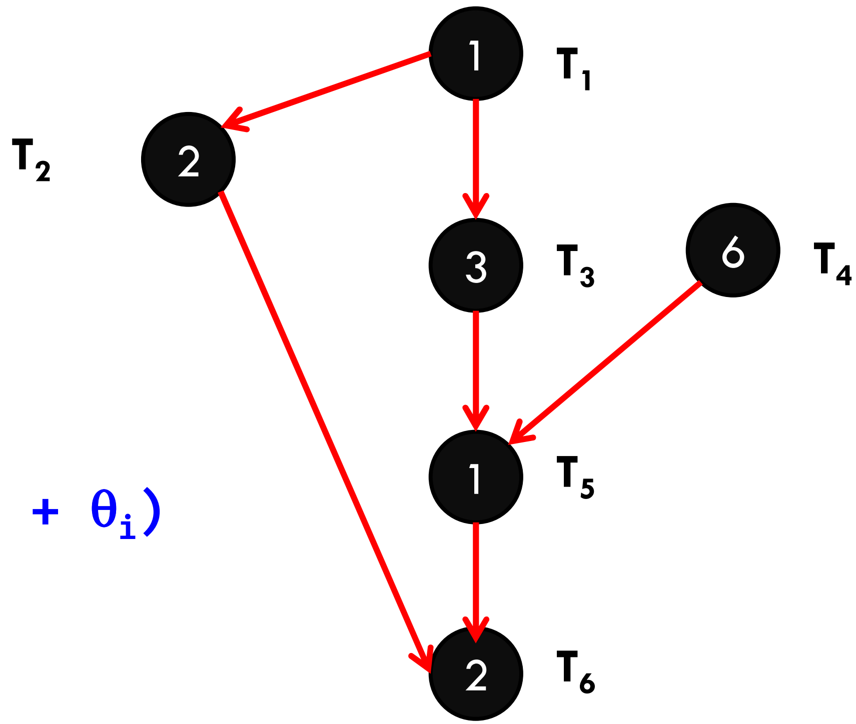


**2. Détermination de temps au plus tôt et au plus tard**

**b) Cas général**

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



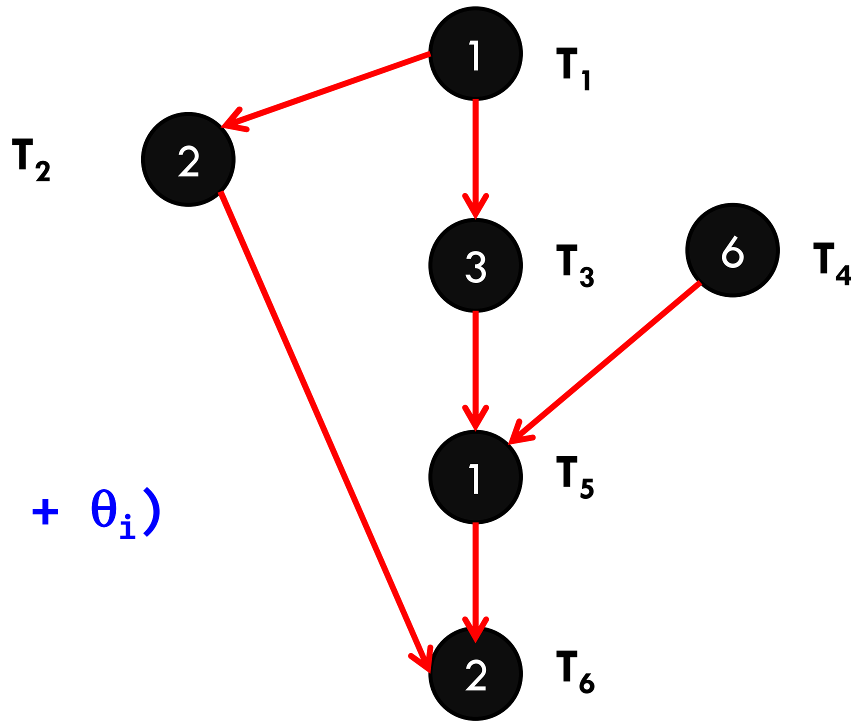
# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

- **Détermination de $T_{opt}$**




```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```

- **C'est l'algorithme de balayage par prédécesseur**

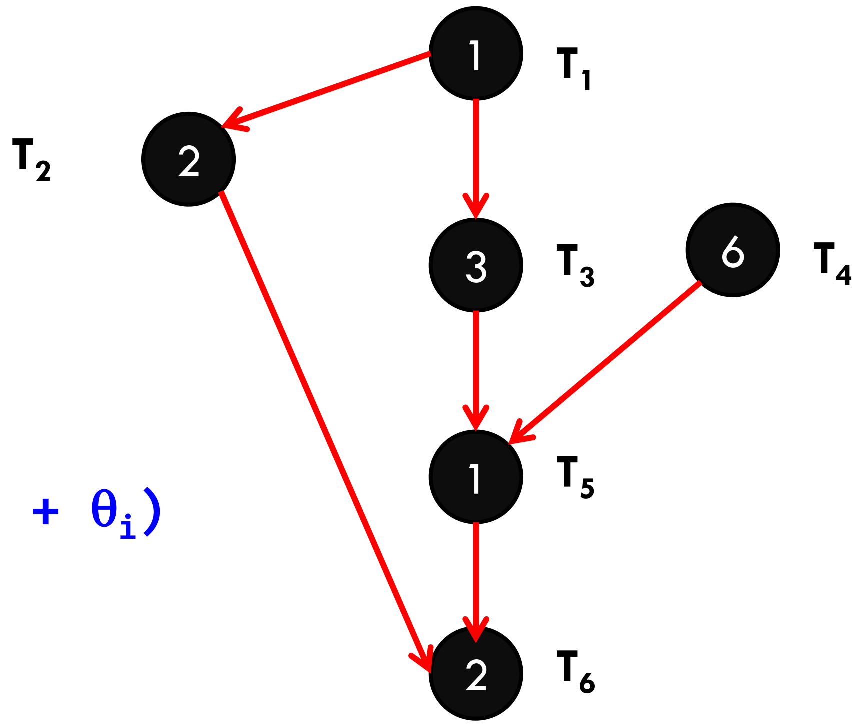
# Ordonnancement des tâches



2. **Détermination de temps au plus tôt et au plus tard**

b) **Cas général**

- **Détermination de $T_{opt}$**




```
tet=0 T_opt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si a_ik =1
   alors tet_k =max(tet_k, tet_i + θ_i)
 fin pour i
   T_opt =max (T_opt, tet_k + θ_k)
fin pour k
```

- **Les tâches qui n'ont pas de prédécesseurs commencent à 0**

# Ordonnancement des tâches

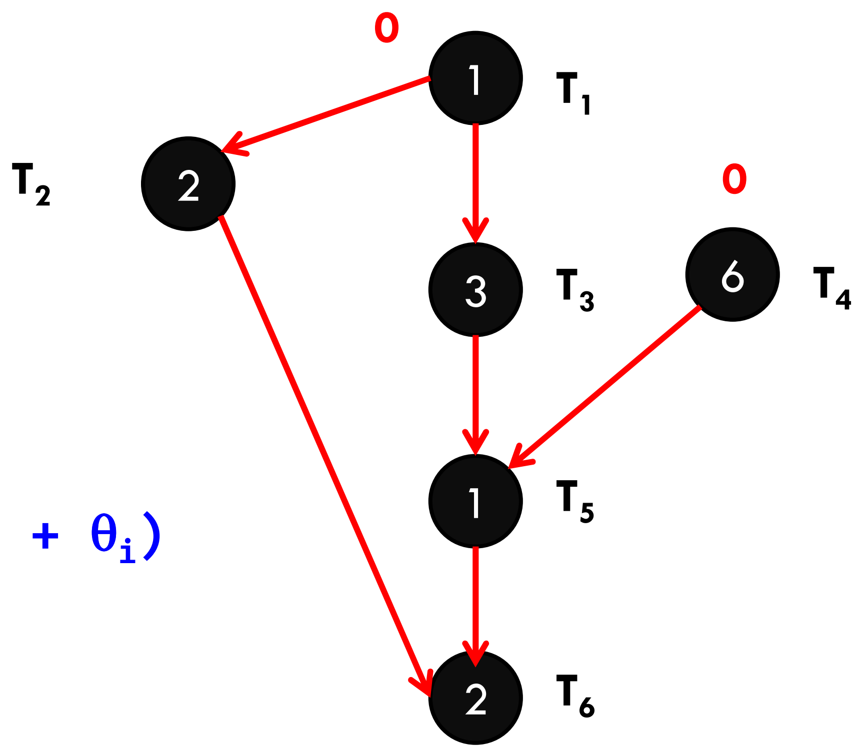


2. Détermination de temps au plus tôt et au plus tard

b) Cas général

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



# Ordonnancement des tâches

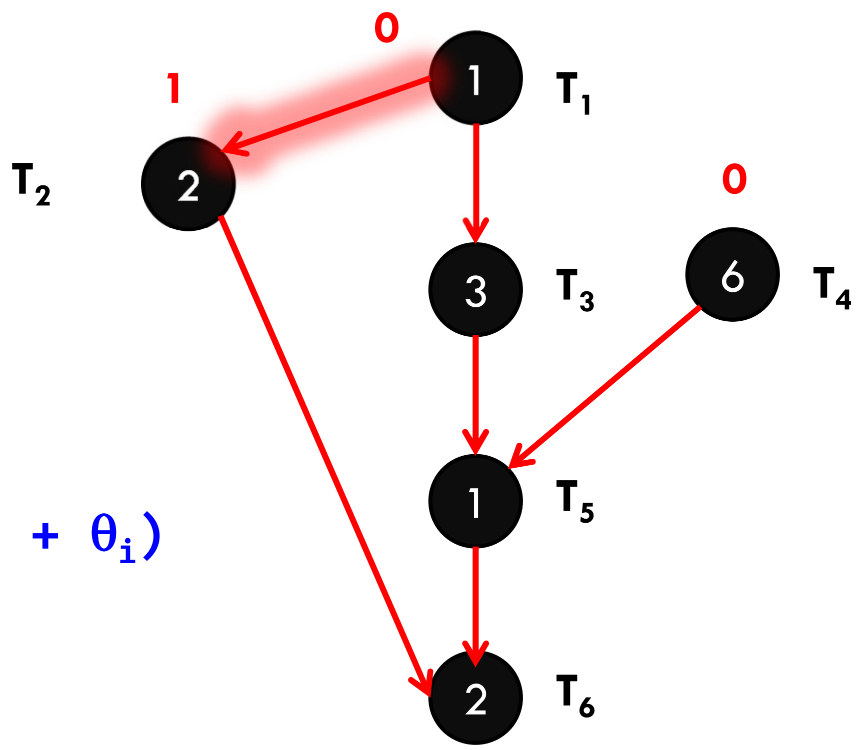


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



# Ordonnancement des tâches

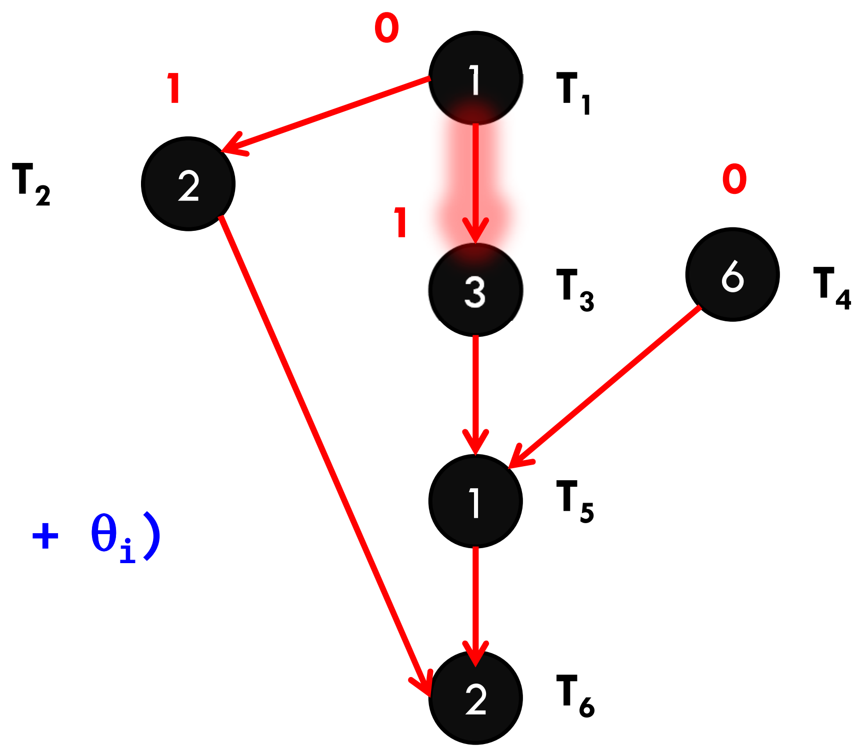


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

- **Détermination de $T_{opt}$**

```
tet=0 T_opt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si a_ik =1
   alors tet_k =max(tet_k, tet_i + θ_i)
 fin pour i
   T_opt =max (T_opt, tet_k + θ_k)
fin pour k
```



# Ordonnancement des tâches

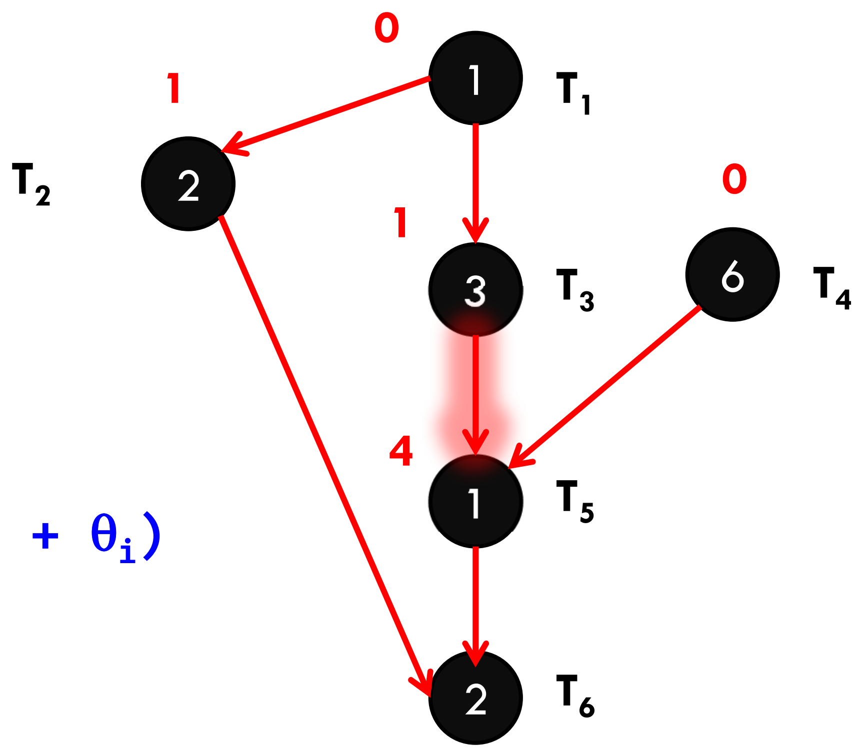


**2. Détermination de temps au plus tôt et au plus tard**

**b) Cas général**

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



# Ordonnancement des tâches

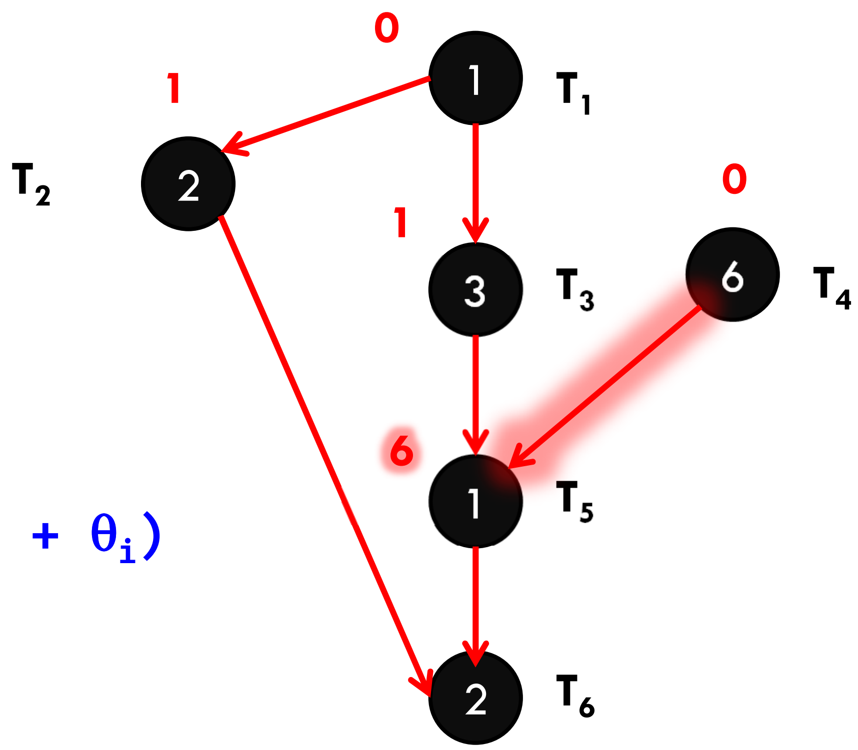


**2. Détermination de temps au plus tôt et au plus tard**

**b) Cas général**

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



# Ordonnancement des tâches

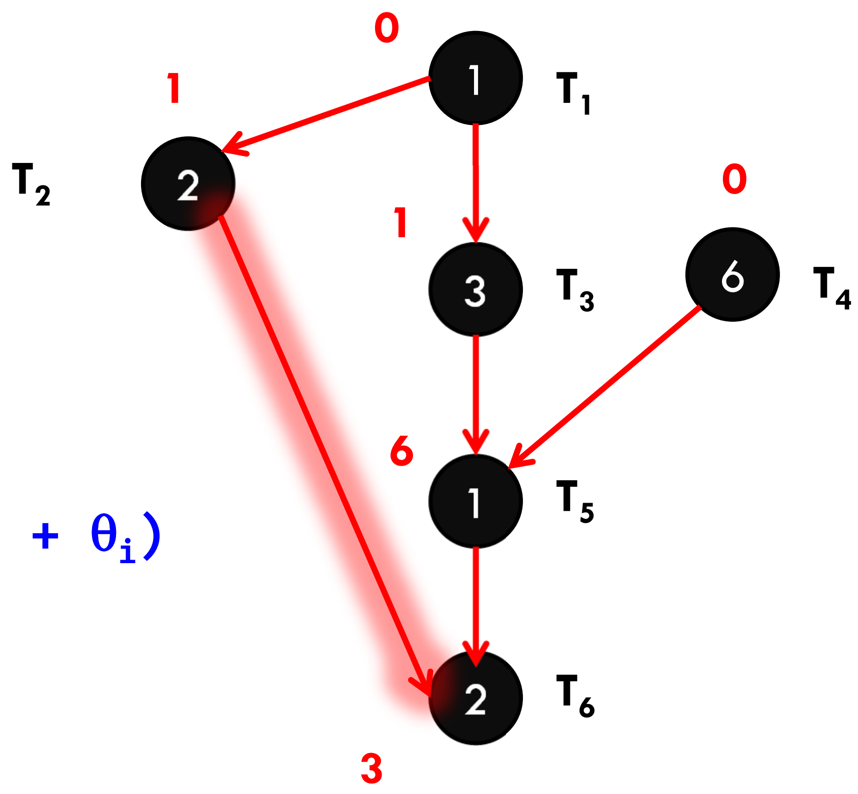


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



# Ordonnancement des tâches

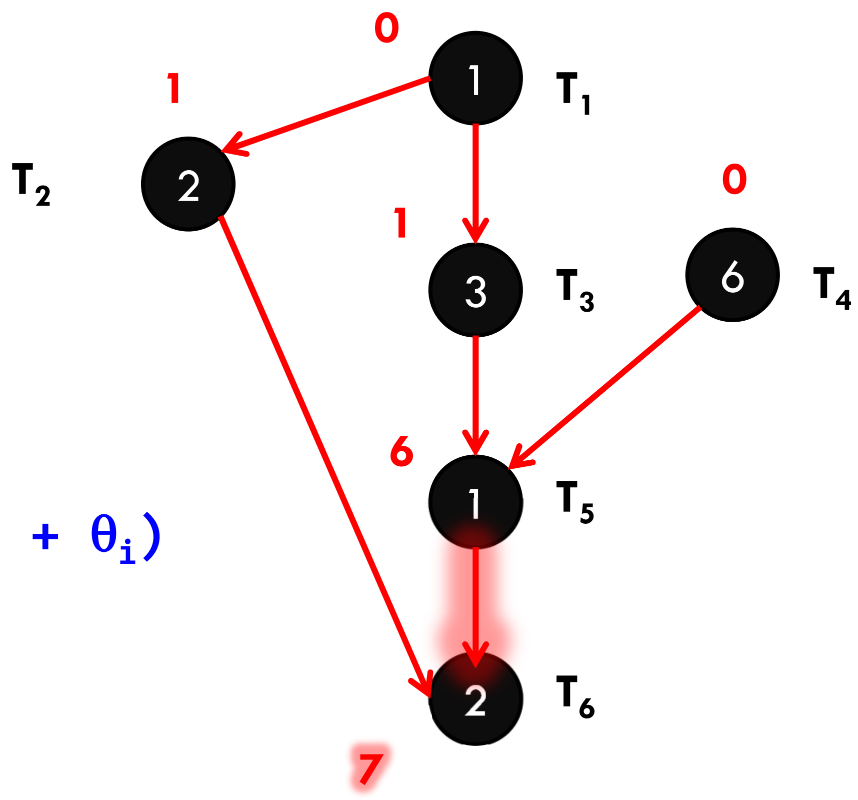


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

- **Détermination de $T_{opt}$**

```
tet=0 Topt =0
pour k=2 à n faire
 pour i=1 à k-1 faire
   si aik =1
   alors tetk =max(tetk, teti + θi)
 fin pour i
   Topt =max (Topt, tetk + θk)
fin pour k
```



# Ordonnancement des tâches

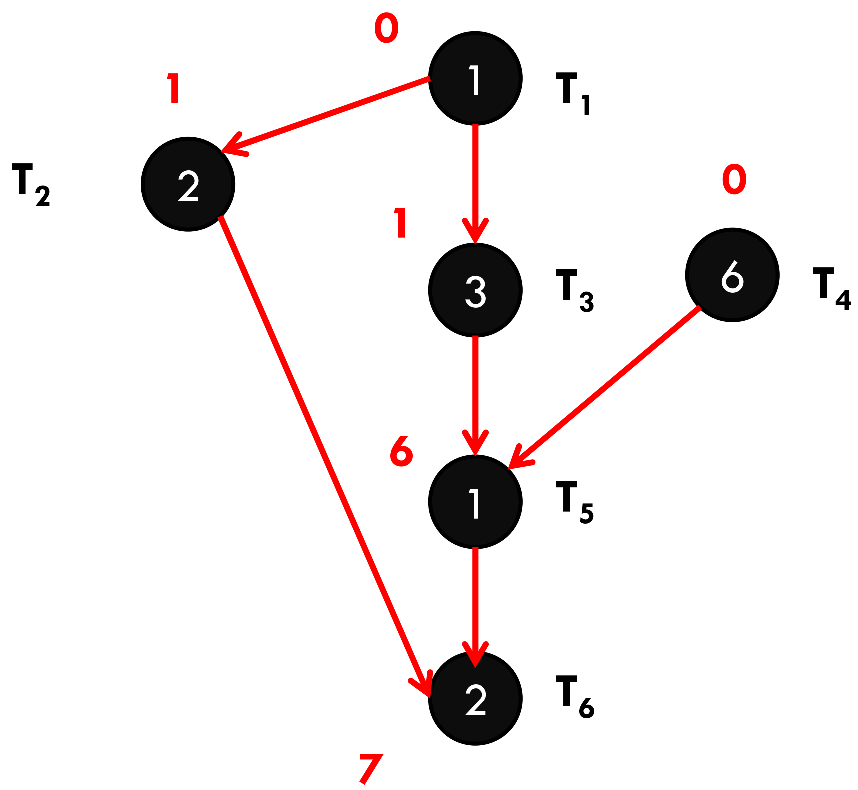


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

- Détermination de $T_{opt}$

Donc $T_{opt} = 9$



# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

teT=$T_{opt}$ pour toutes les tâches

pour k=n à 1 faire

  pour i=k+1 à n faire

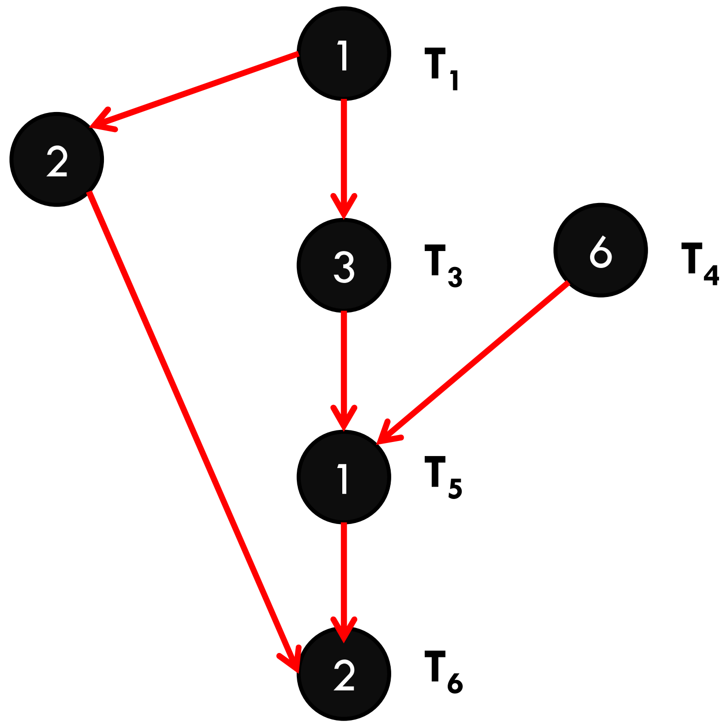
    si $a_{ik}$ =1

    alors $teT_k$ =min($teT_k$, $teT_i$)

  fin pour i

      $teT_k$= $teT_k$ - $\theta_k$

fin pour k



# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

$teT = T_{opt}$ pour toutes les tâches

pour k=n à 1 faire

  pour i=k+1 à n faire

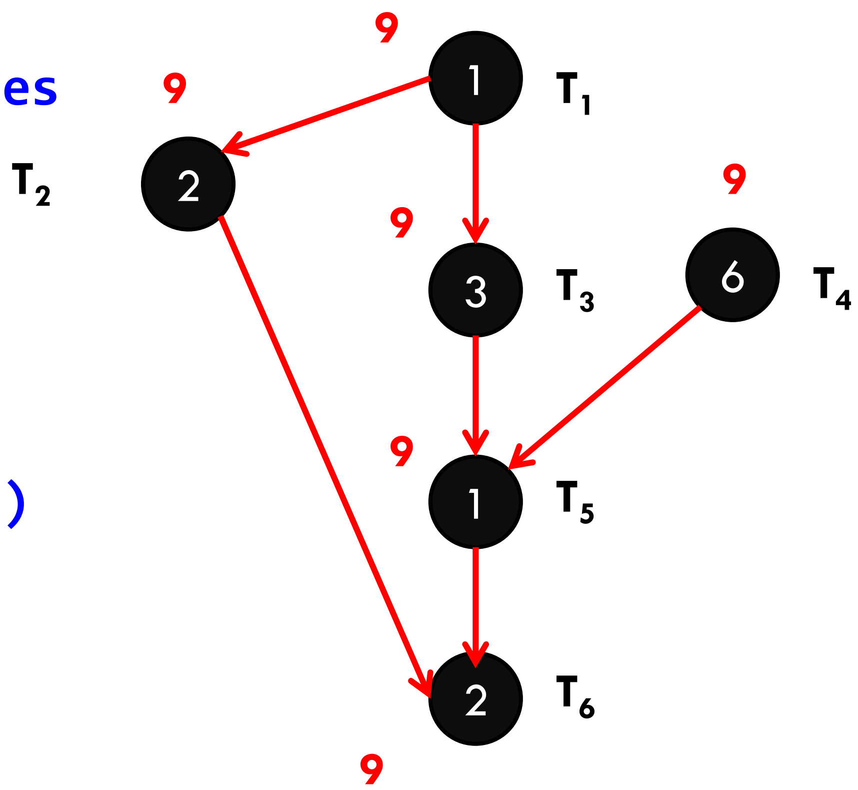
    si $a_{ik}$ =1

    alors $teT_k = \min(teT_k, teT_i)$

  fin pour i

      $teT_k = teT_k - \theta_k$

fin pour k



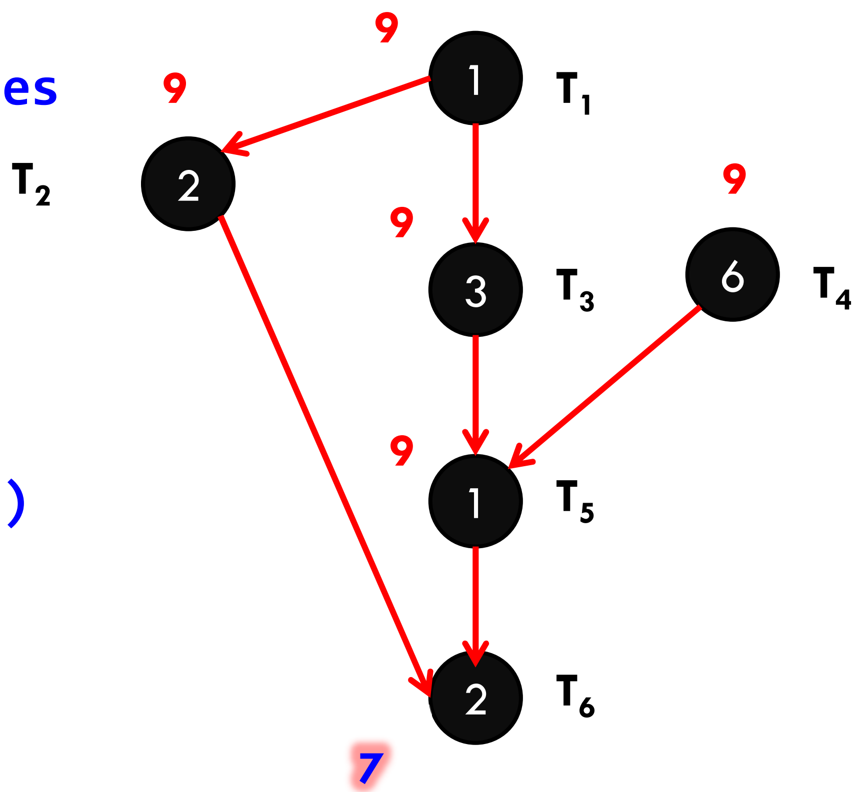
# Ordonnancement des tâches

## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

```
teT=Topt pour toutes les tâches

pour k=n à 1 faire
 pour i=k+1 à n faire
   si aik =1
   alors teTk =min(teTk, teTi)
 fin pour i
       teTk= teTk - θk
fin pour k
```



# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

$teT = T_{opt}$ pour toutes les tâches

pour $k=n$ à 1 faire

 pour $i=k+1$ à $n$ faire

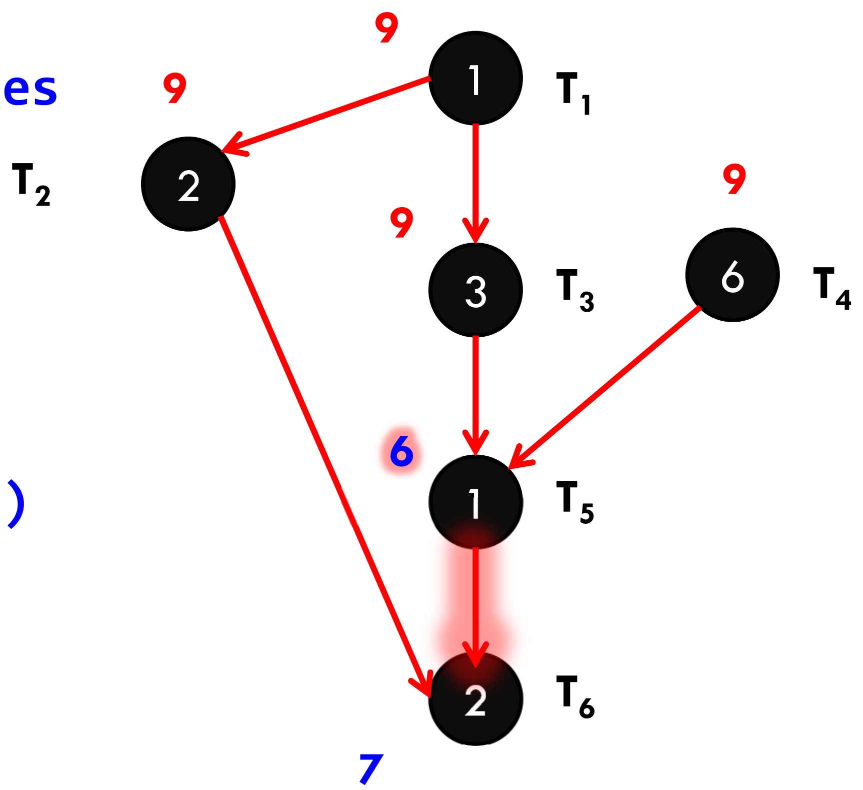
  si $a_{ik} = 1$

  alors $teT_k = \min(teT_k, teT_i)$

 fin pour i

   $teT_k = teT_k - \theta_k$

fin pour k



# Ordonnancement des tâches

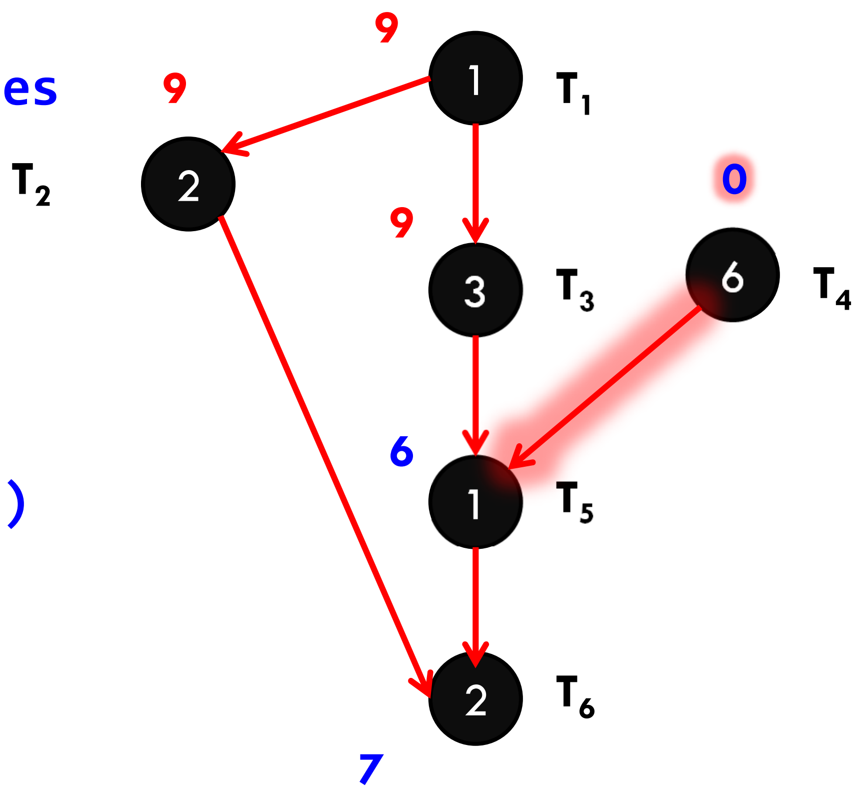


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

```
teT=Topt pour toutes les tâches

pour k=n à 1 faire
 pour i=k+1 à n faire
   si aik =1
   alors teTk =min(teTk, teTi)
 fin pour i
       teTk= teTk - θk
fin pour k
```



# Ordonnancement des tâches

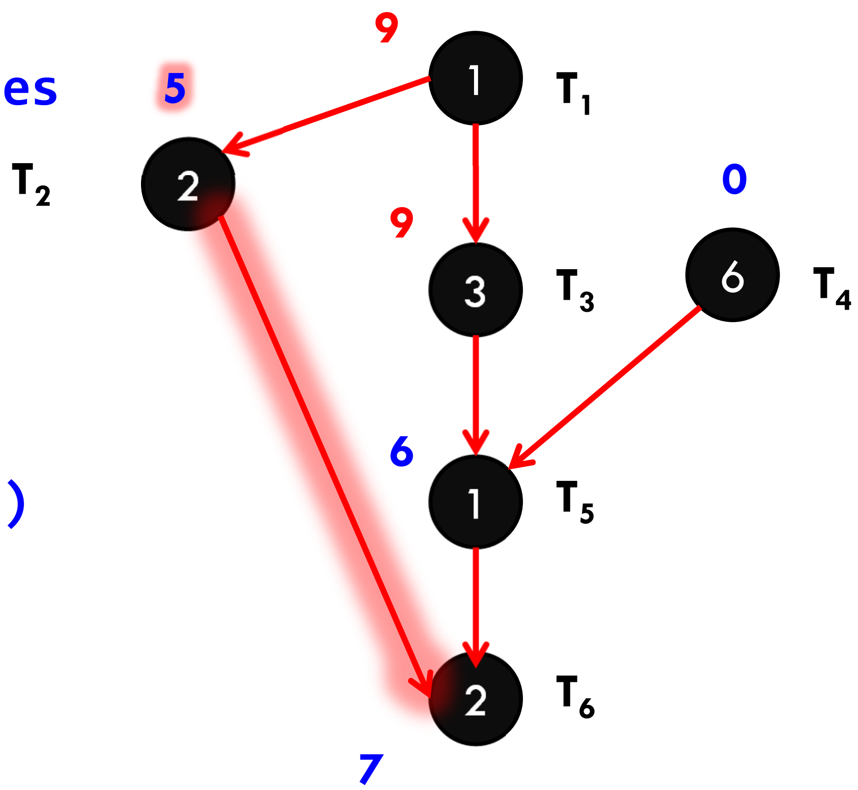


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

```
teT=Topt pour toutes les tâches

pour k=n à 1 faire
 pour i=k+1 à n faire
   si aik =1
   alors teTk =min(teTk, teTi)
 fin pour i
       teTk= teTk - θk
fin pour k
```



# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

$teT = T_{opt}$ pour toutes les tâches

pour $k$=$n$ à 1 faire

  pour $i$=$k$+1 à $n$ faire

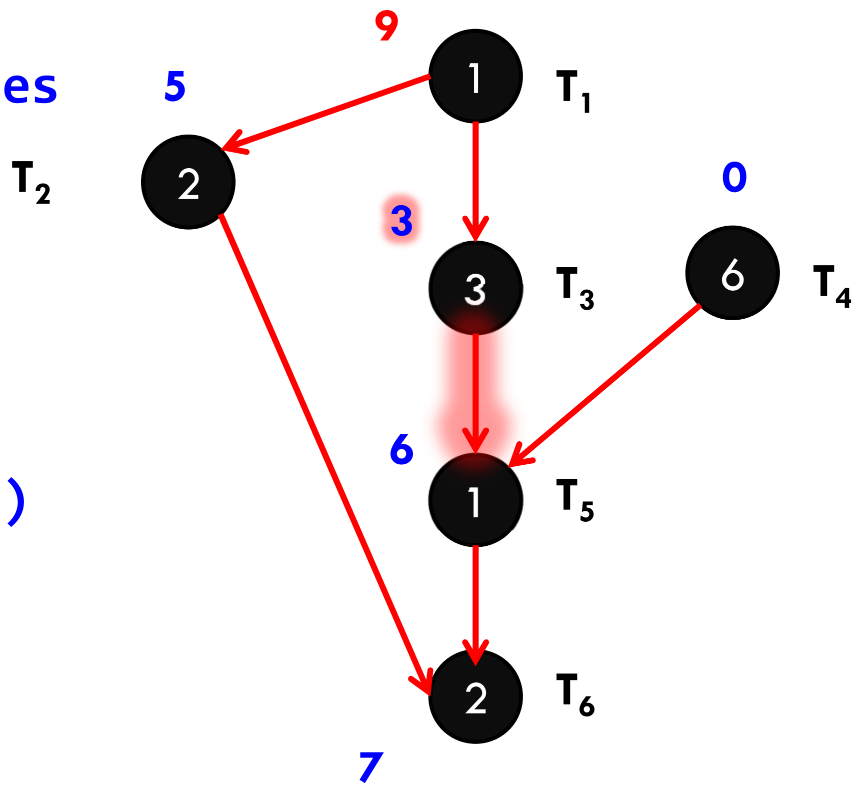
    si $a_{ik}$ =1

    alors $teT_k$ =min($teT_k$, $teT_i$)

  fin pour $i$

      $teT_k$= $teT_k$ - $\theta_k$

fin pour $k$



# Ordonnancement des tâches



## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général

$teT = T_{opt}$ pour toutes les tâches

pour k=n à 1 faire

 pour i=k+1 à n faire

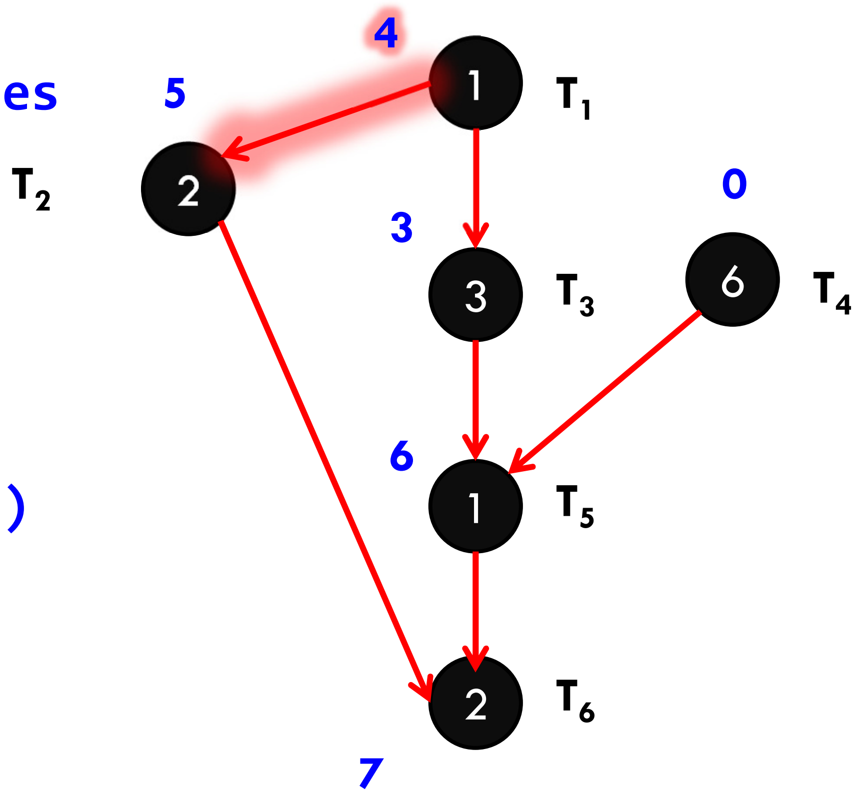
   si $a_{ik} = 1$

   alors $teT_k = \min(teT_k, teT_i)$

 fin pour i

      $teT_k = teT_k - \theta_k$

fin pour k



# Ordonnancement des tâches



2. Détermination de temps au plus tôt et au **plus tard**

b) Cas général

$teT = T_{opt}$ pour toutes les tâches

pour k=n à 1 faire

 pour i=k+1 à n faire

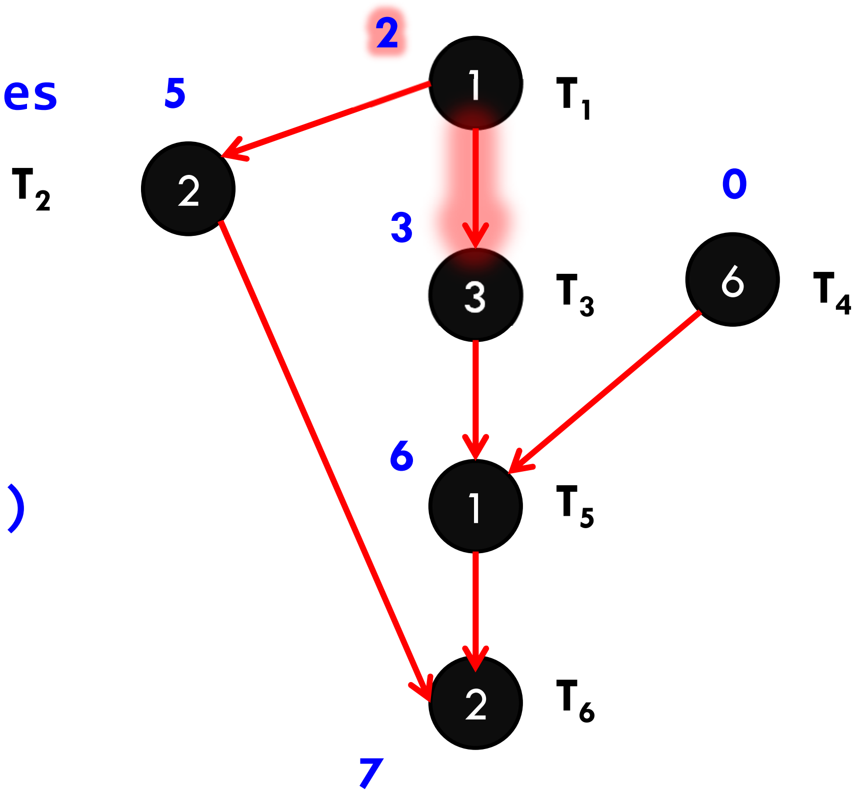
  si $a_{ik}$ =1

  alors $teT_k = \min(teT_k, teT_i)$

 fin pour i

   $teT_k = teT_k - \theta_k$

fin pour k

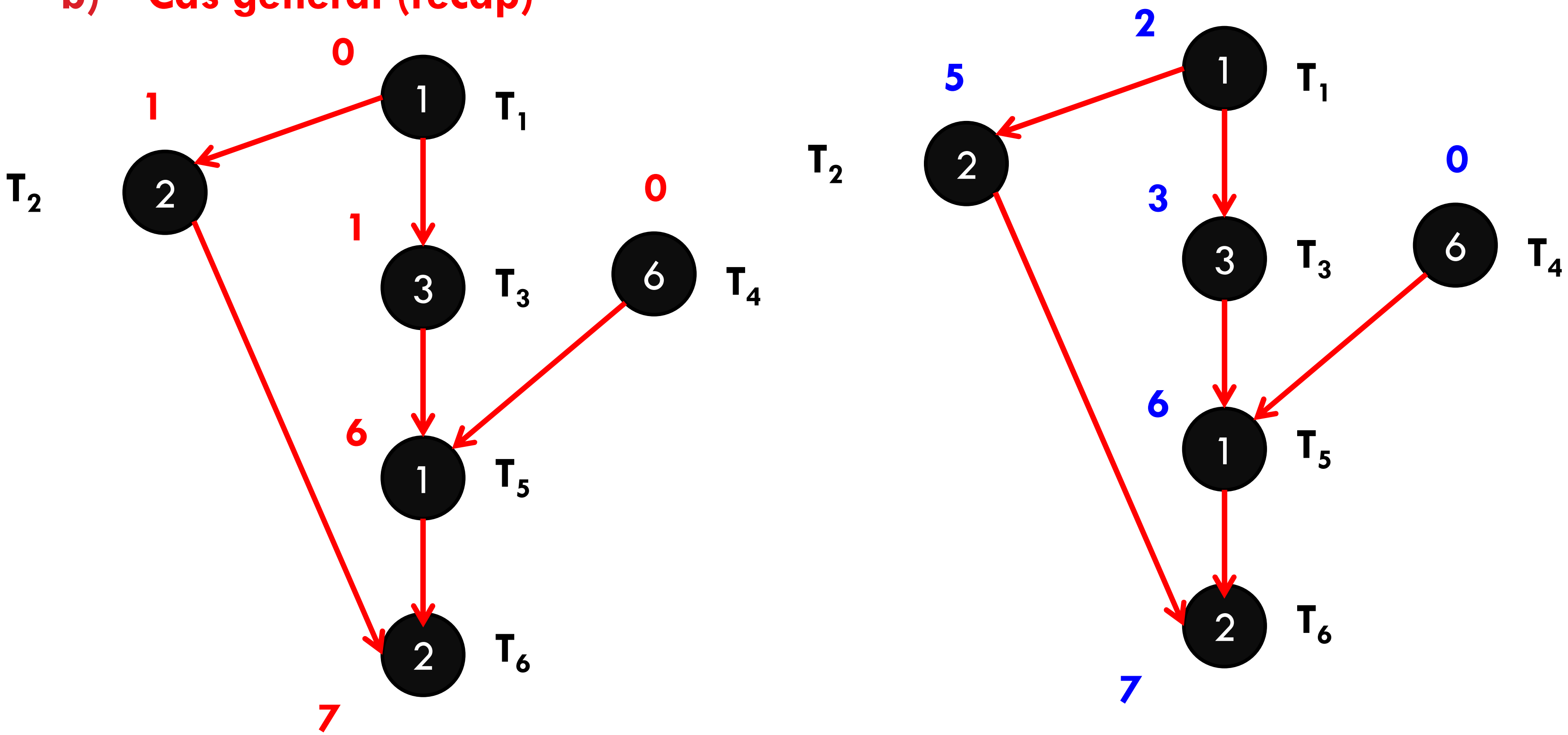


# Ordonnancement des tâches



2. **Détermination de temps au plus tôt et au plus tard**

b) **Cas général (recap)**

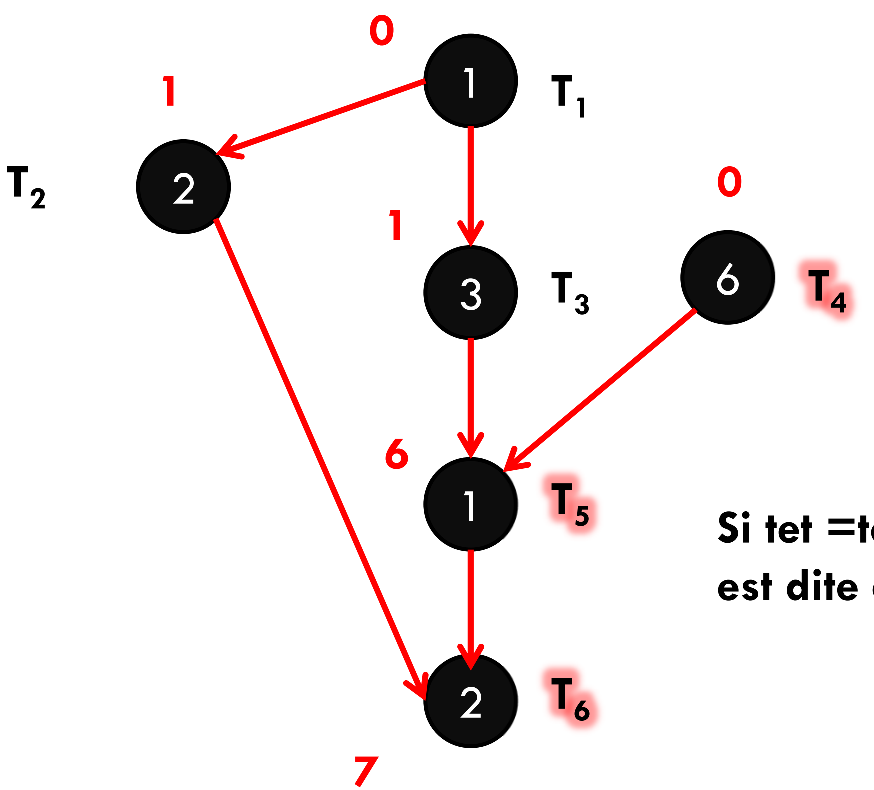


# Ordonnancement des tâches

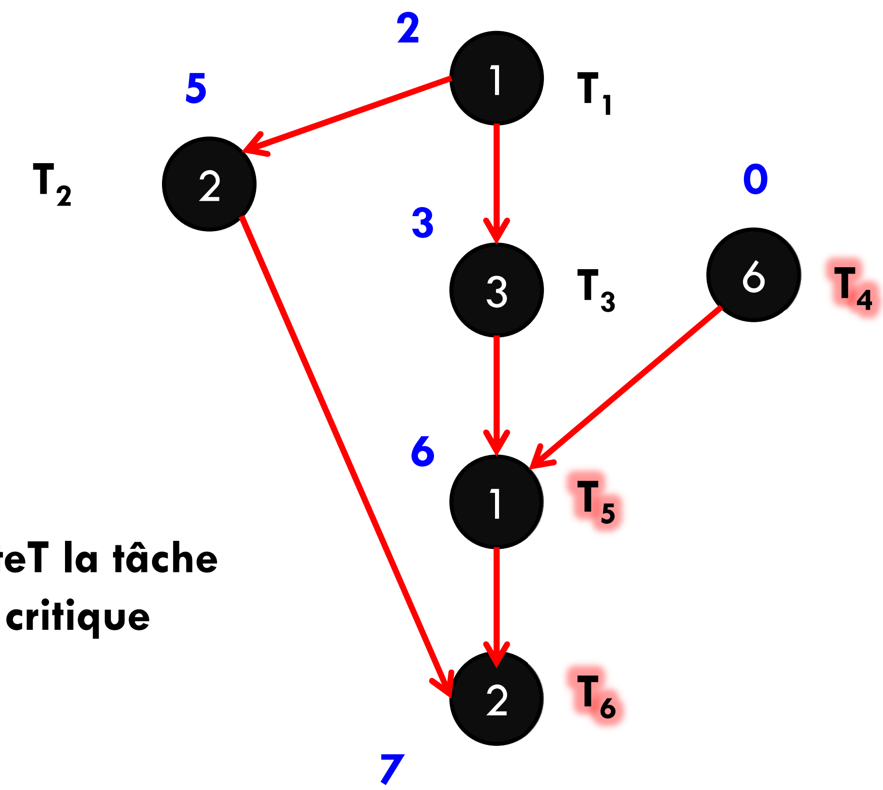


## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général (recap)




Si tet =teT la tâche est dite critique

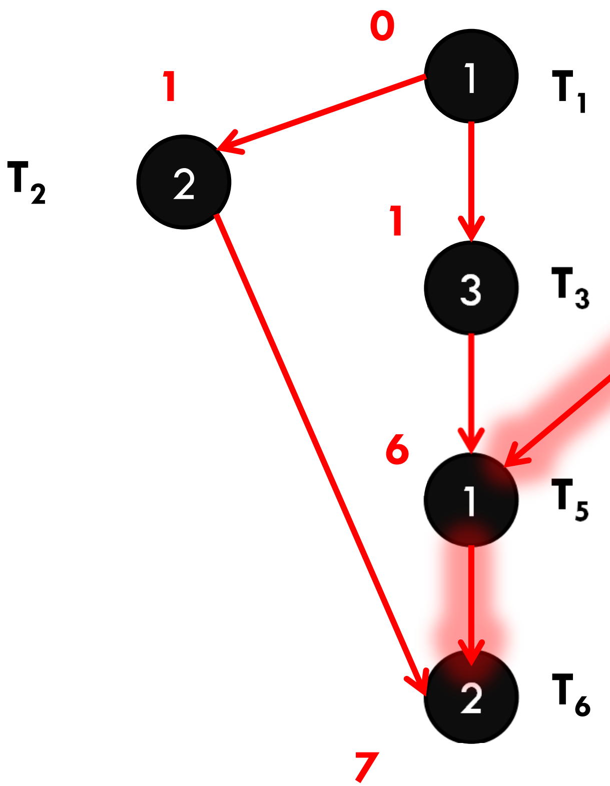


# Ordonnancement des tâches



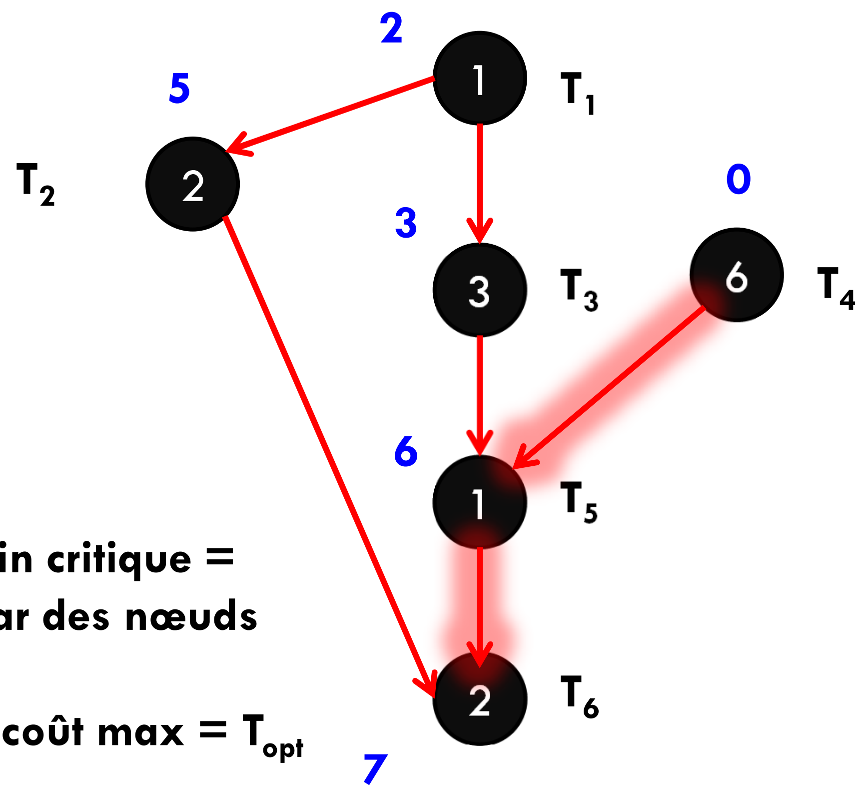
## 2. Détermination de temps au plus tôt et au plus tard

### b) Cas général (recap)




Le chemin critique = formé par des nœuds critiques

Il est de coût max = $T_{opt}$



# Ordonnancement des tâches

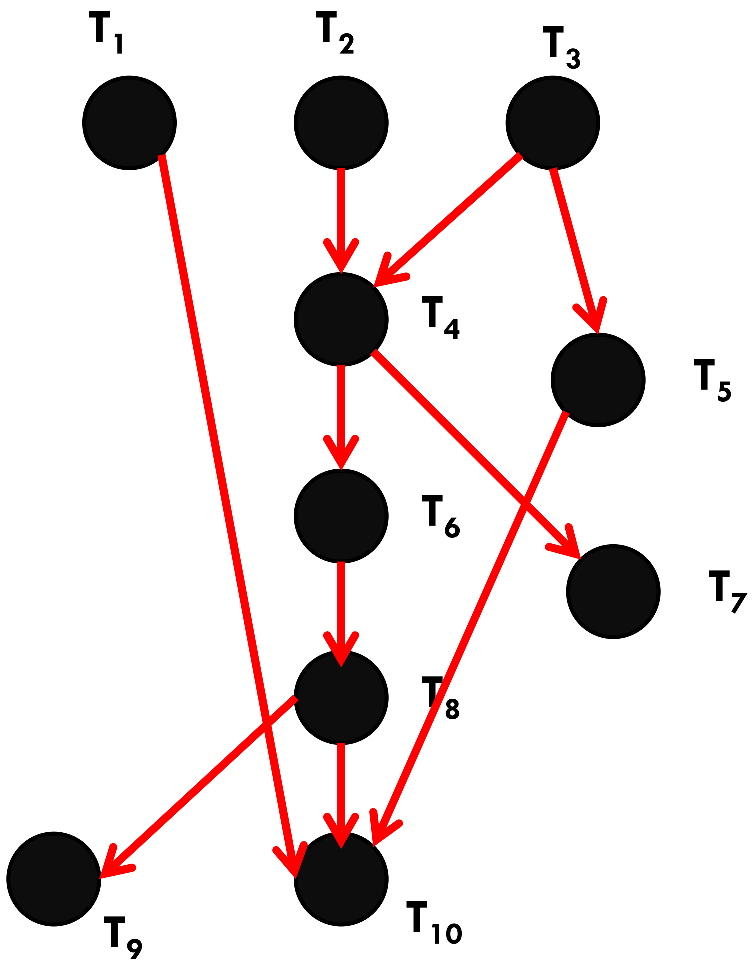


**3. Détermination de $P_{opt}$**

**Pour toutes les tâches**

**on suppose que $\theta_i$=1**

**Déterminer tet et teT**



# Ordonnancement des tâches

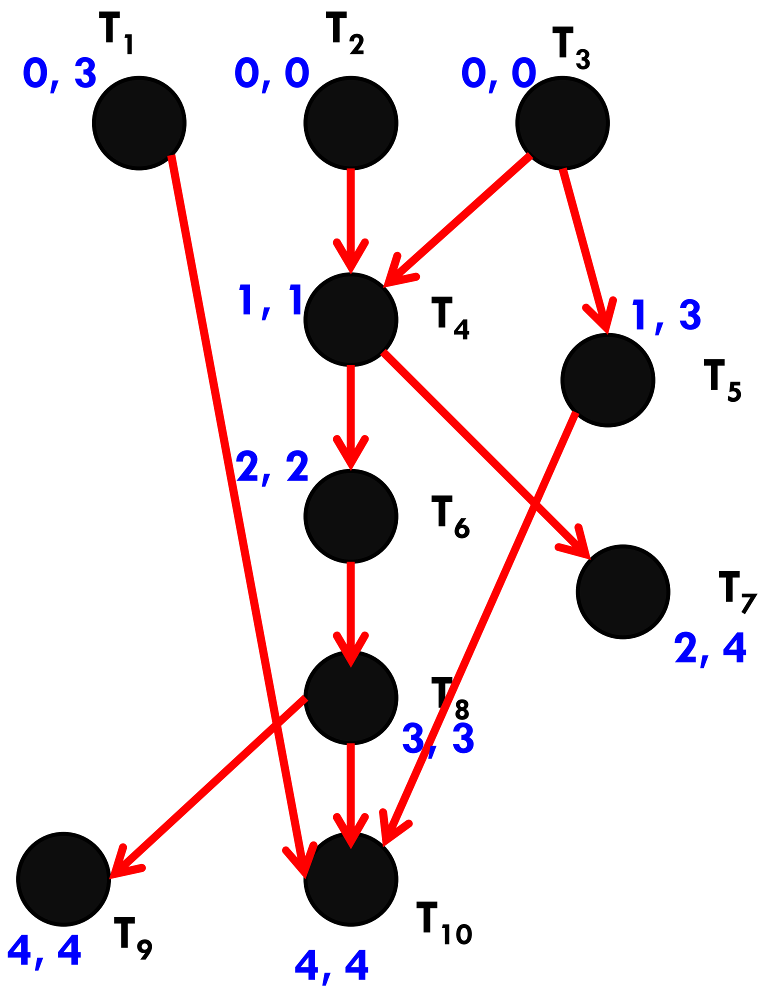


**3. Détermination de $P_{opt}$**

**Pour toutes les tâches**

**on suppose que $\theta_i$=1**

**$T_{opt}$ = 5**



# Ordonnancement des tâches

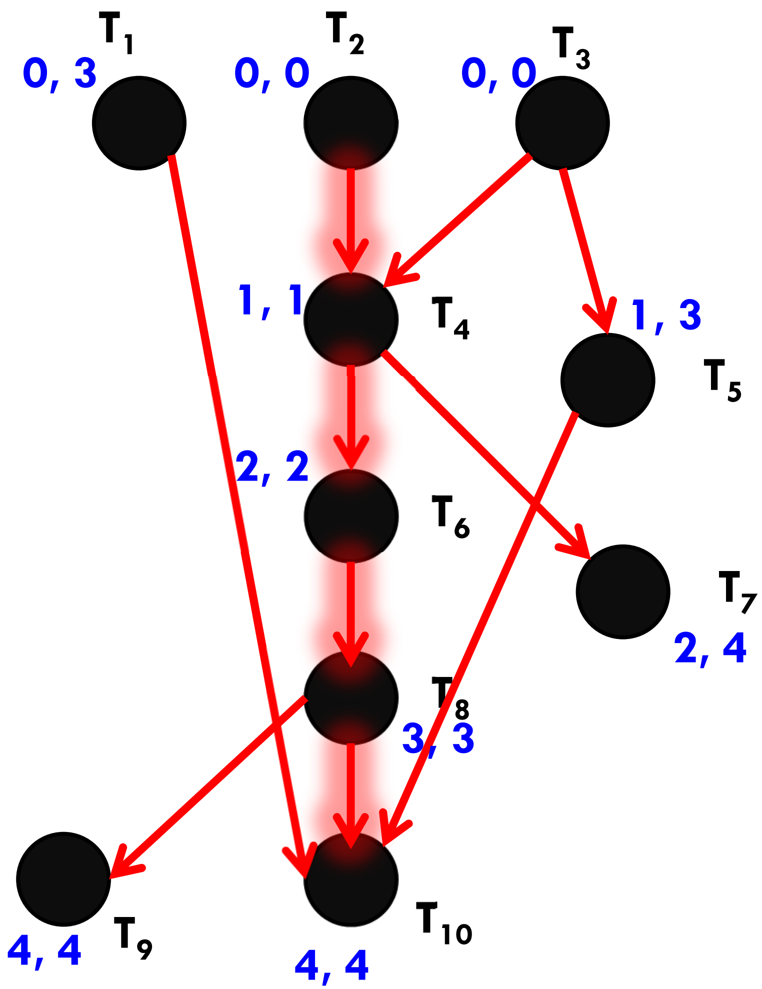


3. **Détermination de $P_{opt}$**

**Pour toutes les tâches**

**on suppose que $\theta_i$=1**

**Chemin critique**



# Ordonnancement des tâches

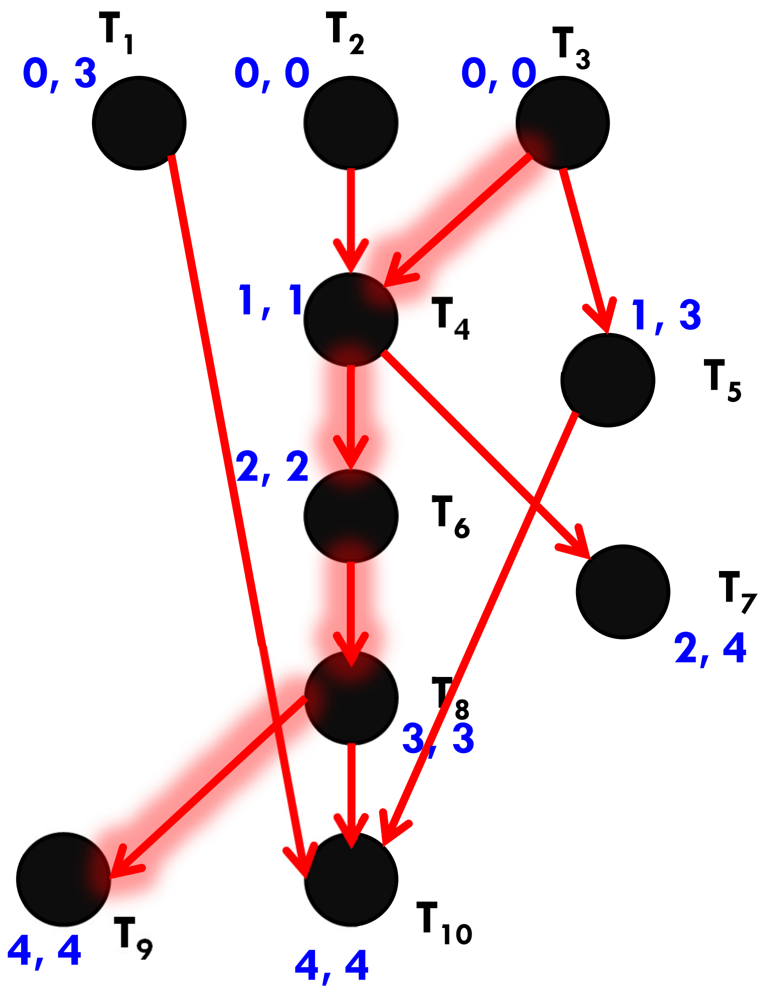


**3. Détermination de $P_{opt}$**

**Pour toutes les tâches**

**on suppose que $\theta_i$=1**

**Chemin critique**



# Ordonnancement des tâches

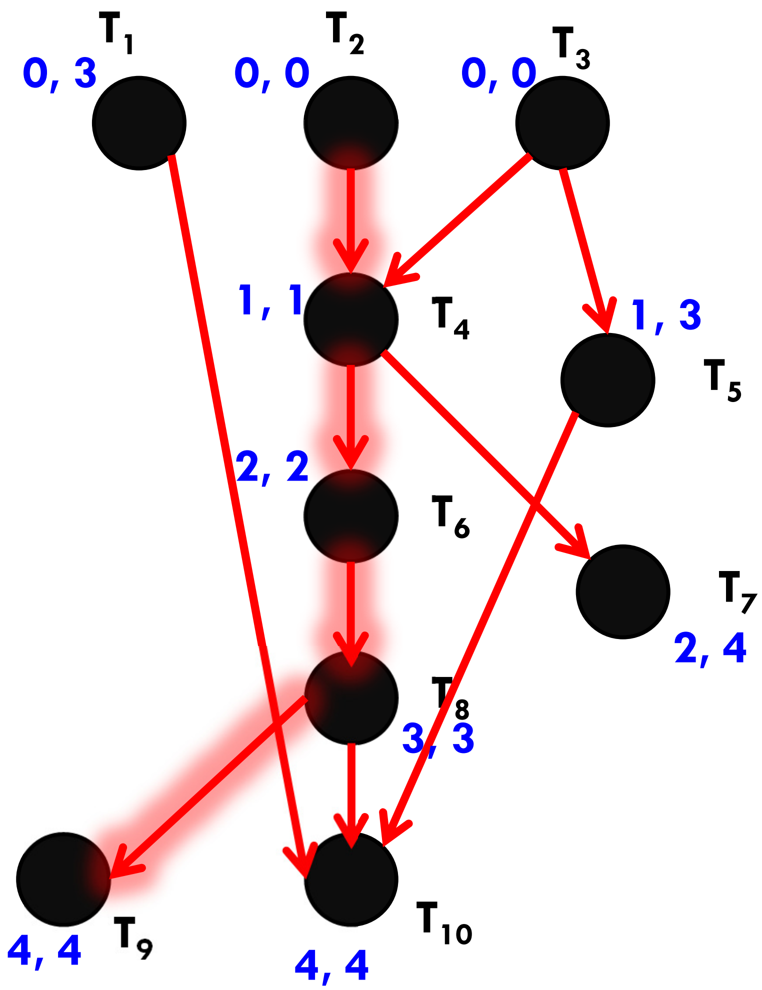


3. **Détermination de $P_{opt}$**

**Pour toutes les tâches**

**on suppose que $\theta_i$=1**

**Chemin critique**



# Ordonnancement des tâches

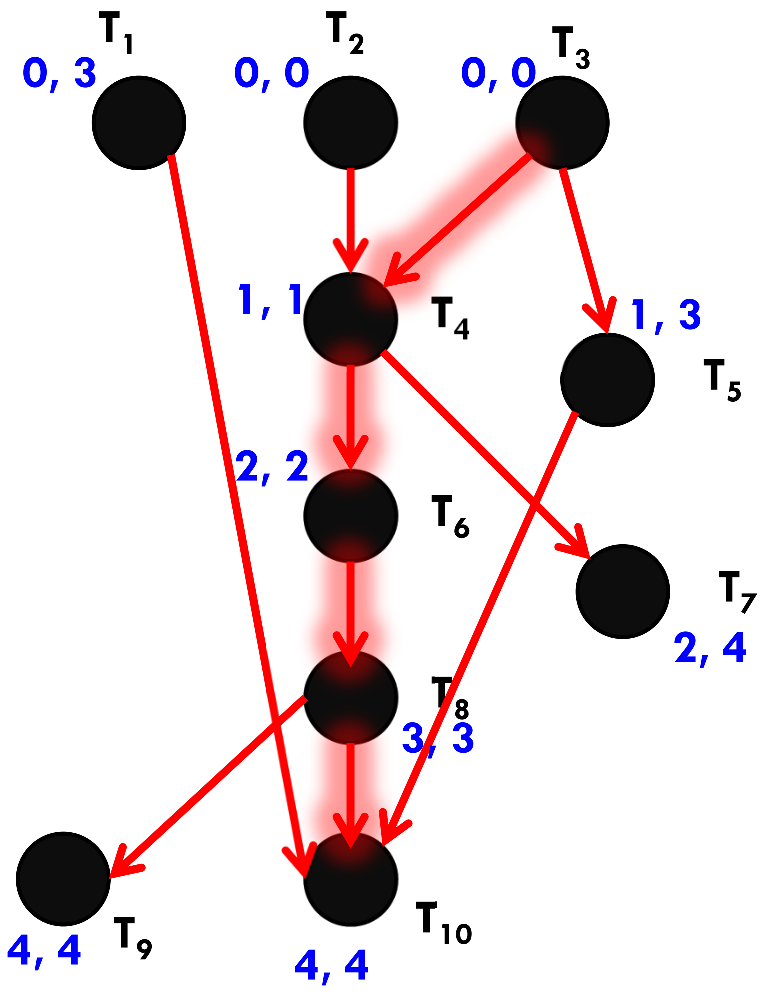


**3. Détermination de $P_{opt}$**

Pour toutes les tâches

on suppose que $\theta_i=1$

Chemin critique



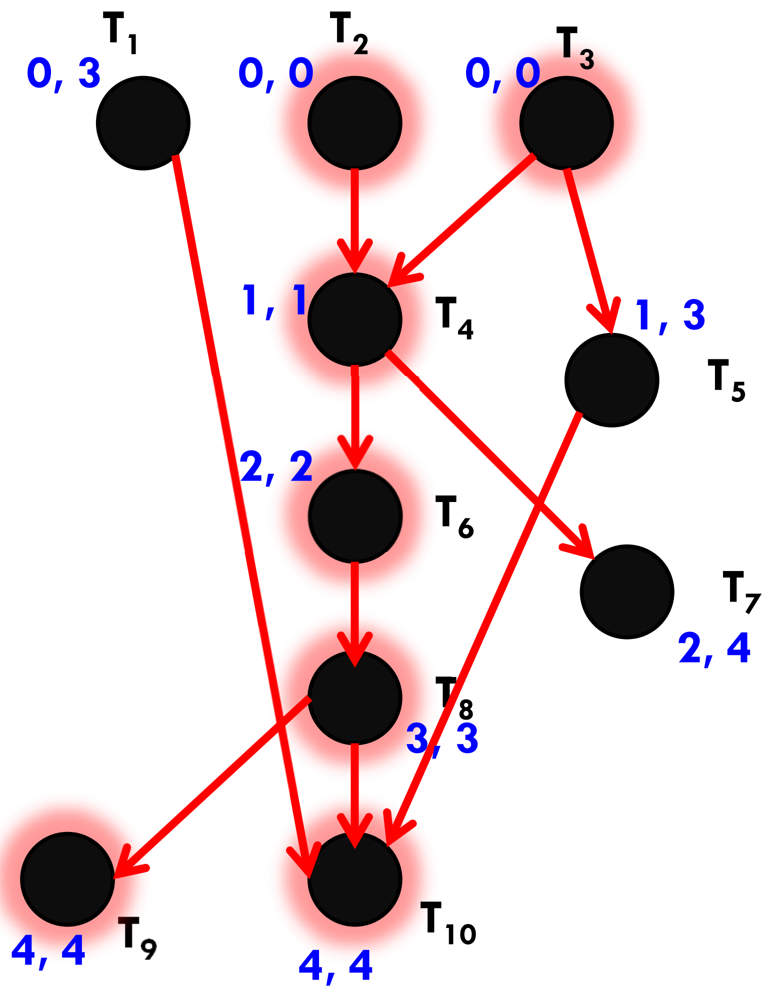
# Ordonnancement des tâches

**3. Détermination de $P_{opt}$**

**Pour toutes les tâches**

**on suppose que $\theta_i=1$**

**Nœuds critiques**



# Ordonnancement des tâches



**3. Détermination de $P_{opt}$**

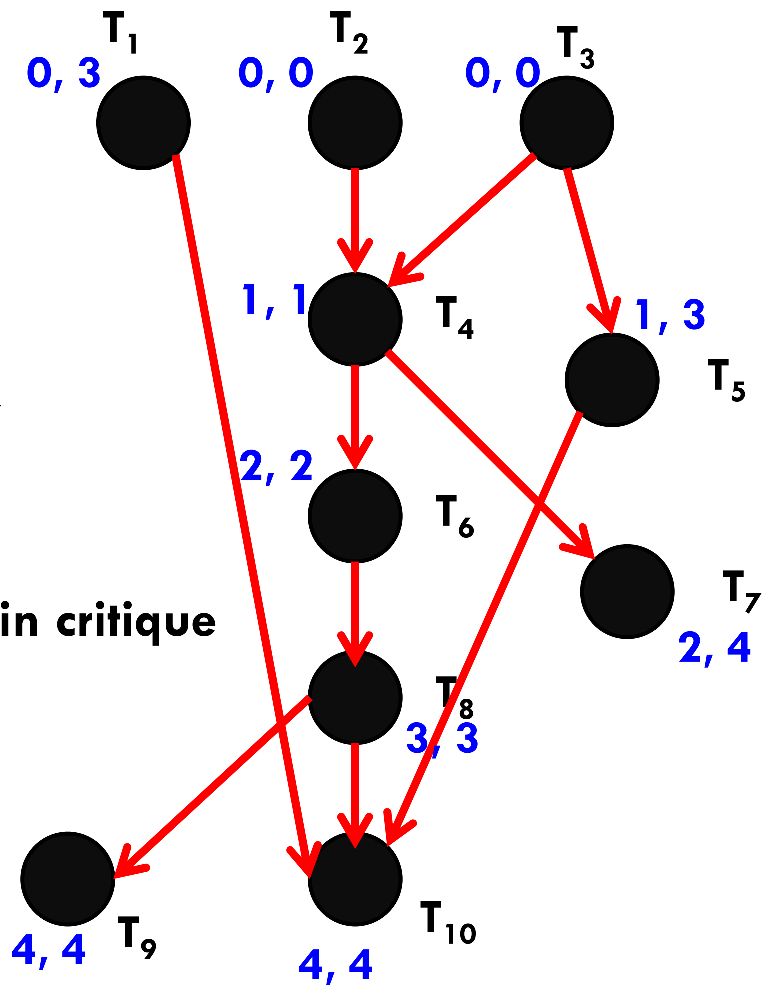
Pour toutes les tâches

on suppose que $\theta_i=1$

Décomposer le graphe en des niveaux

$N_k$ = niveau k

h(G) = hauteur du G = longueur chemin critique



# Ordonnancement des tâches



**3. Détermination de $P_{opt}$**

Pour toutes les tâches

on suppose que $\theta_i=1$

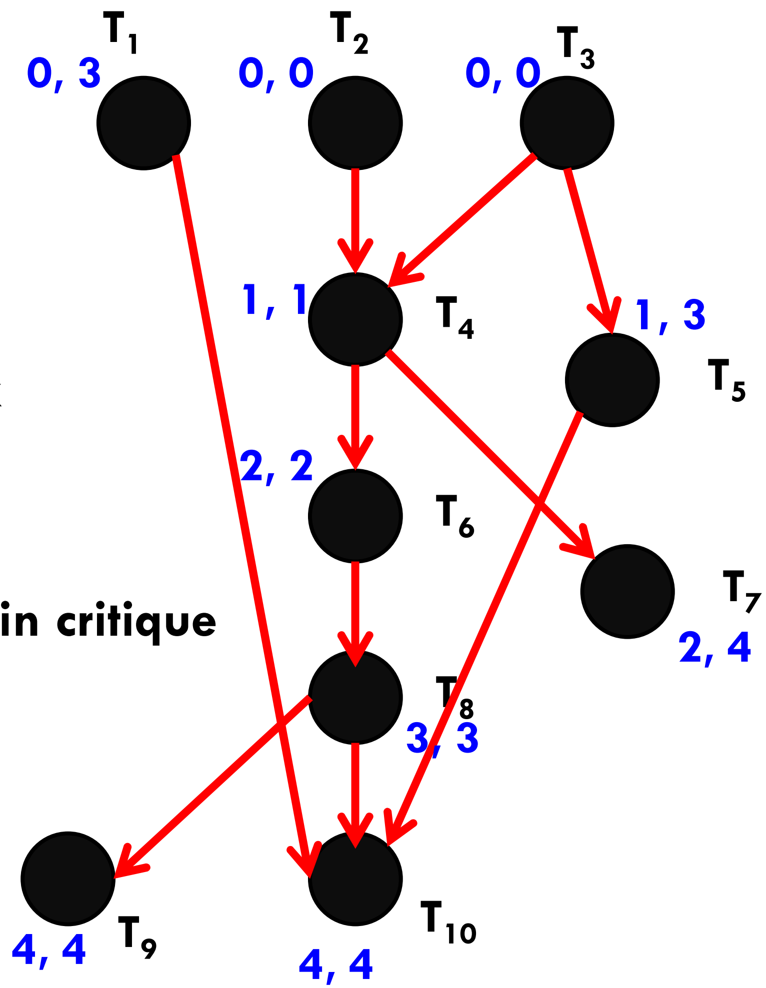
Décomposer le graphe en des niveaux

$N_k$ = niveau k

h(G) = hauteur du G = longueur chemin critique

$N_1$ = nœuds sans prédécesseurs

$N_{h(g)}$ = nœuds sans successeurs



# Ordonnancement des tâches



**3. Détermination de $P_{opt}$**

Pour toutes les tâches

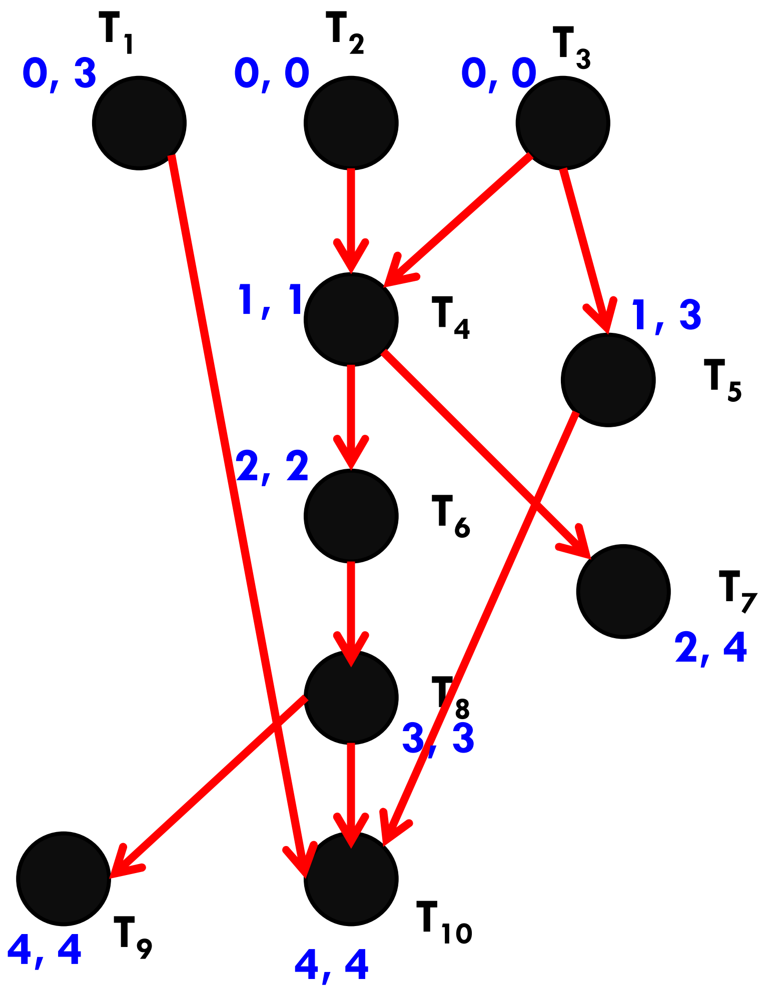
on suppose que $\theta_i=1$

**Décomposer le graphe en des niveaux**

**Deux décompositions :**

**$D_{tot}$ (G) : chaque $T_i$ se trouve $N_{tôt}$ (i)**

**$D_{tard}$(G) : chaque $T_i$ se trouve $N_{tard}$ (i)**



# Ordonnancement des tâches



## 3. Détermination de $P_{opt}$

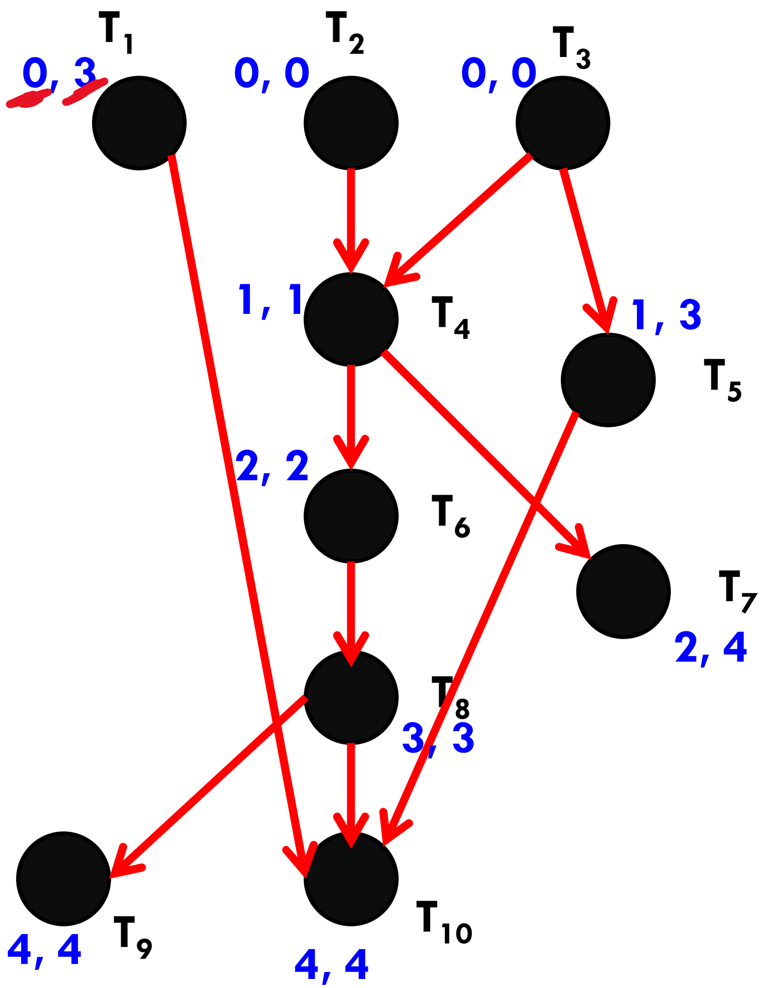
**Pour toutes les tâches**

**on suppose que $\theta_i = 1$**

**Décomposer le graphe en des niveaux**

$N_{tôt}(i) = (tet_i/\theta\ )+1$

$N_{tard}(i) = (teT_i/\theta\ )+1$

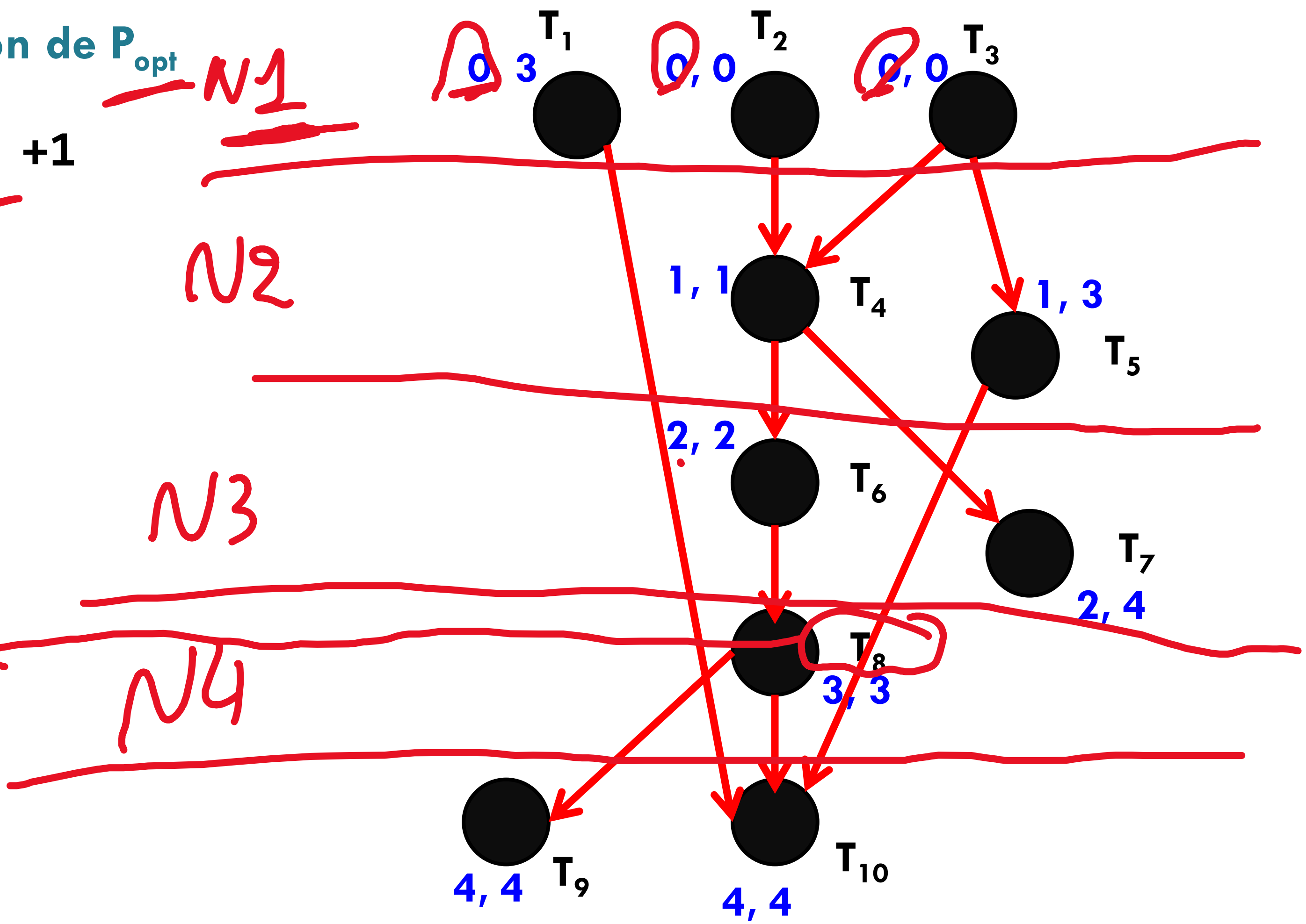


# Ordonnancement des tâches



3. **Détermination de $P_{opt}$**

$N_{tôt}(i) = (tet_i/\theta) + 1$




$\frac{3}{1}$ +1

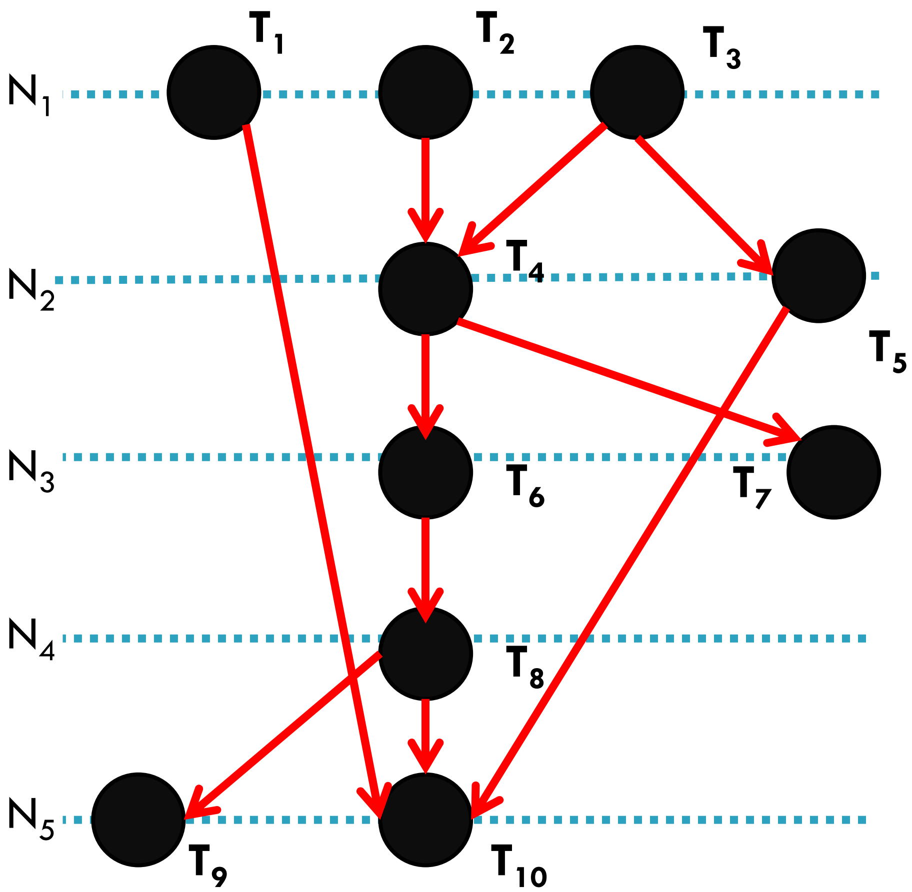
=> 4

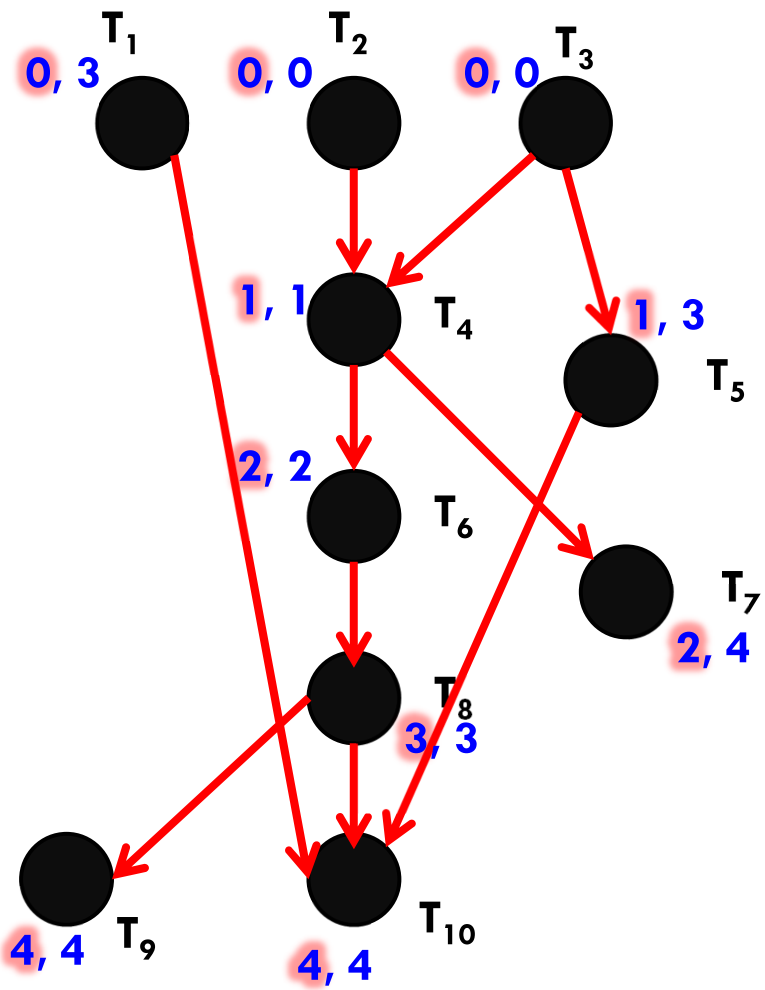
# Ordonnancement des tâches



3. Détermination de $P_{opt}$

$N_{tôt}(i) = (tet_i / \theta) + 1$



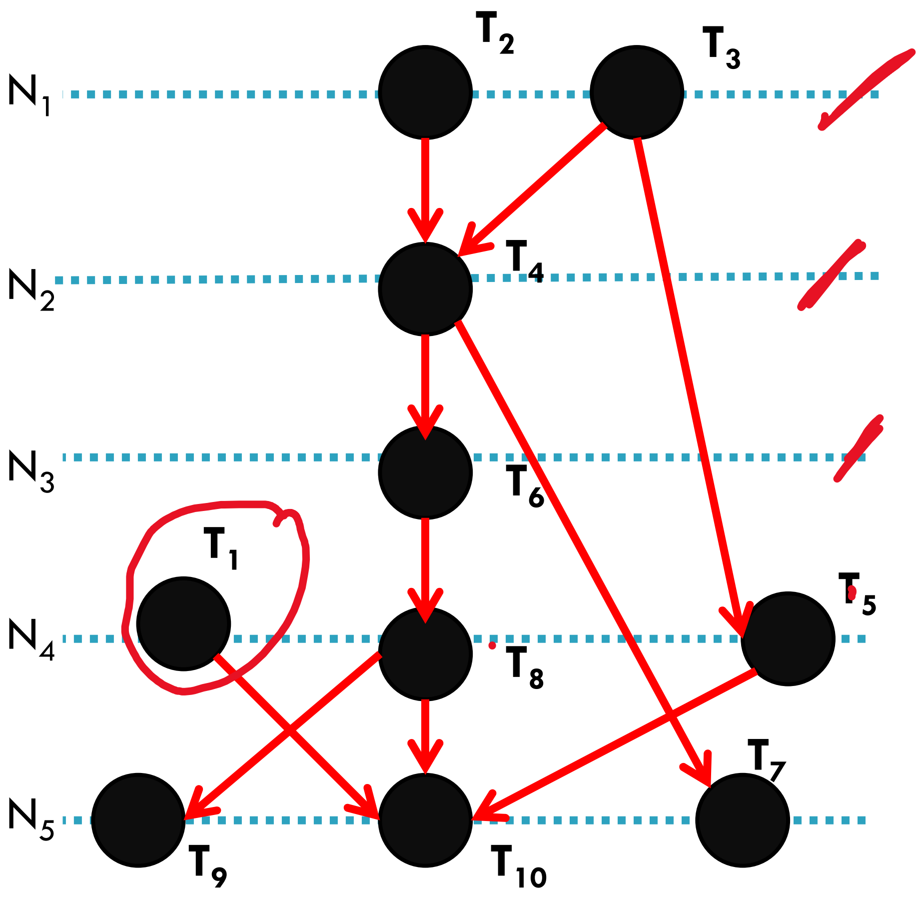



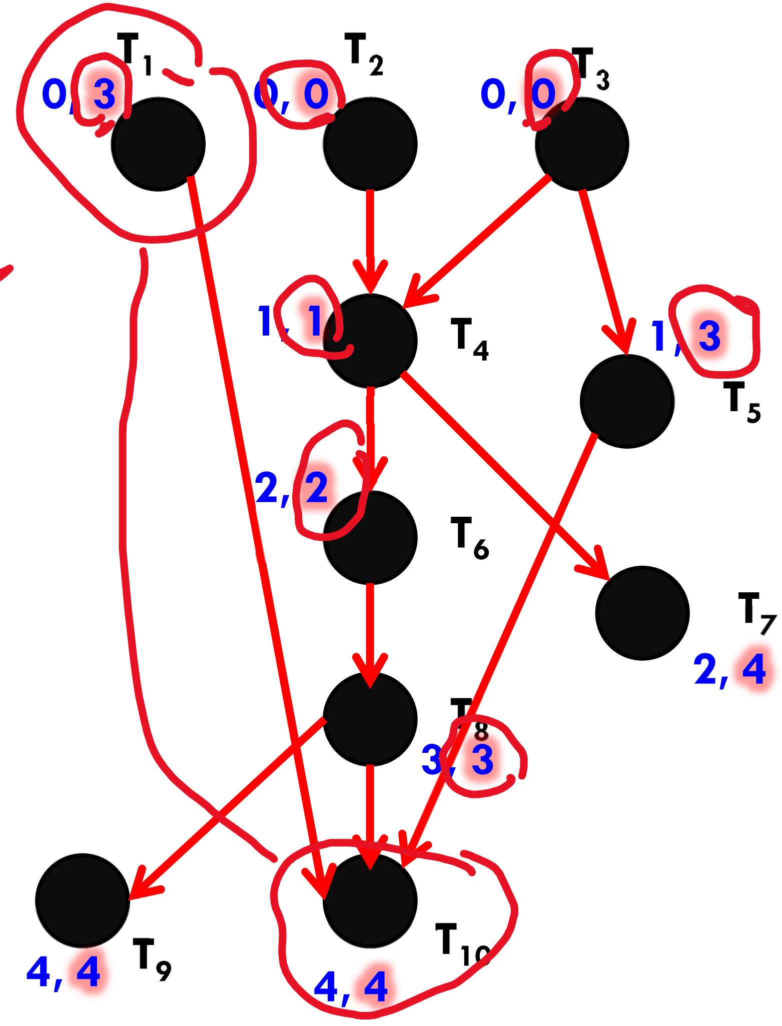
# Ordonnancement des tâches



## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta\ )+1$



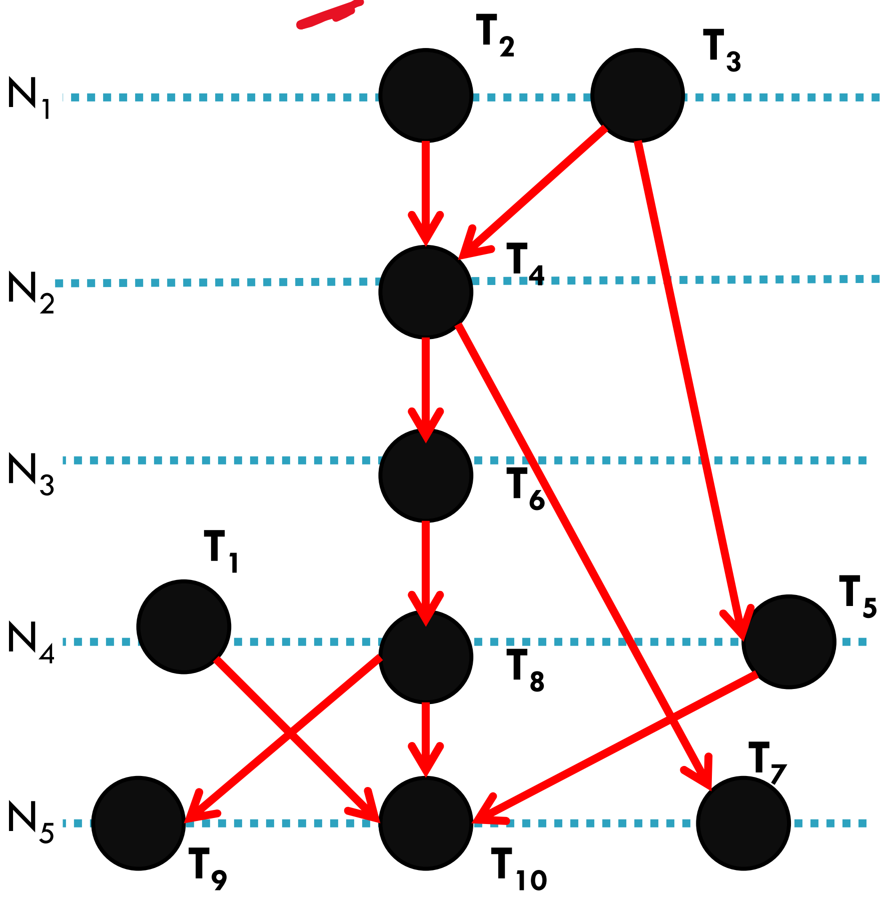



# Ordonnancement des tâches



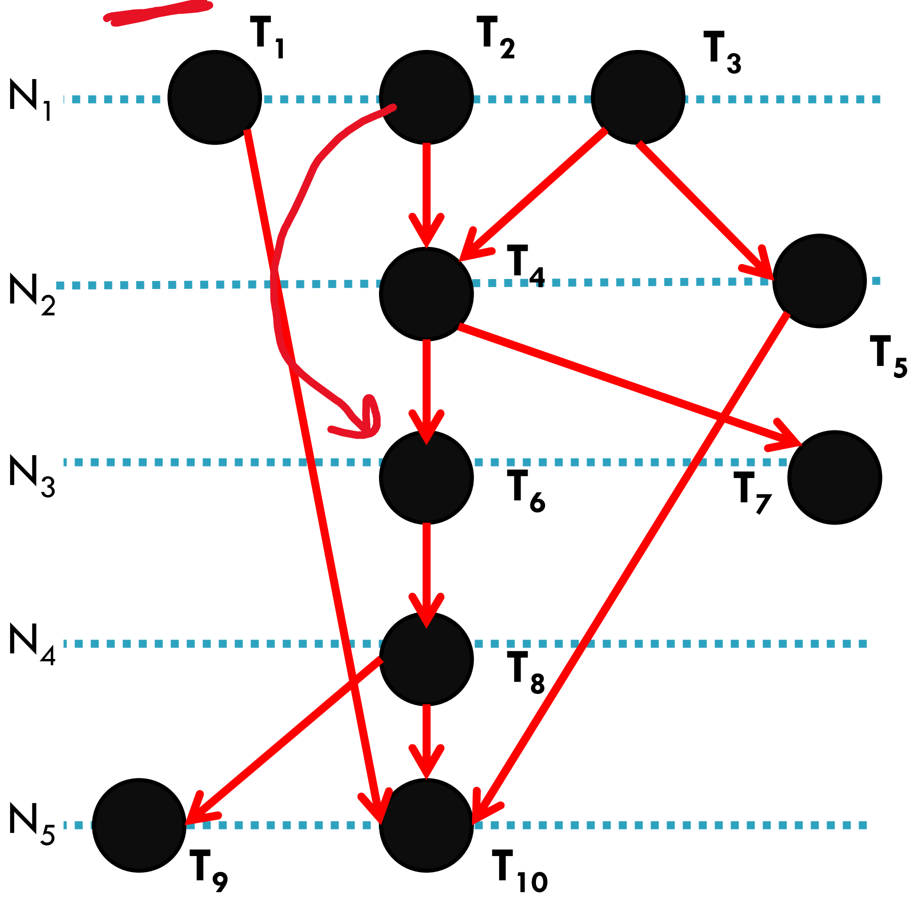
## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta\ )+1$





$N_{tôt}(i) = (tet_i\ /\theta)\ \ +1$



$teT_i = tet_i \Rightarrow$ tâche critique

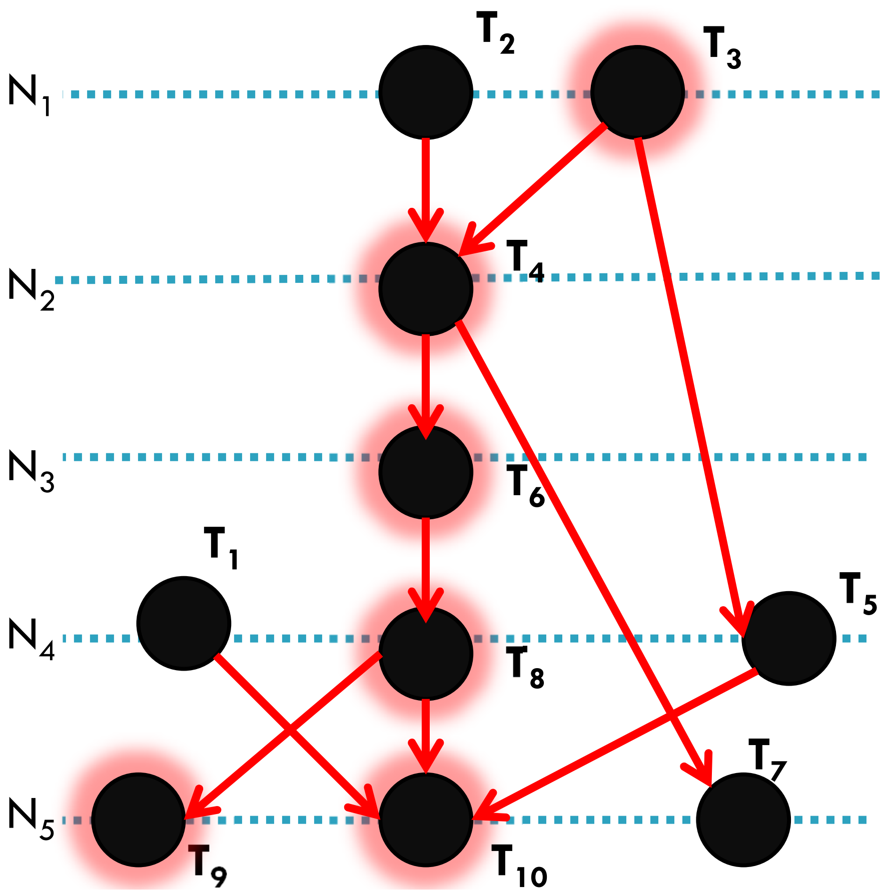
# Ordonnancement des tâches



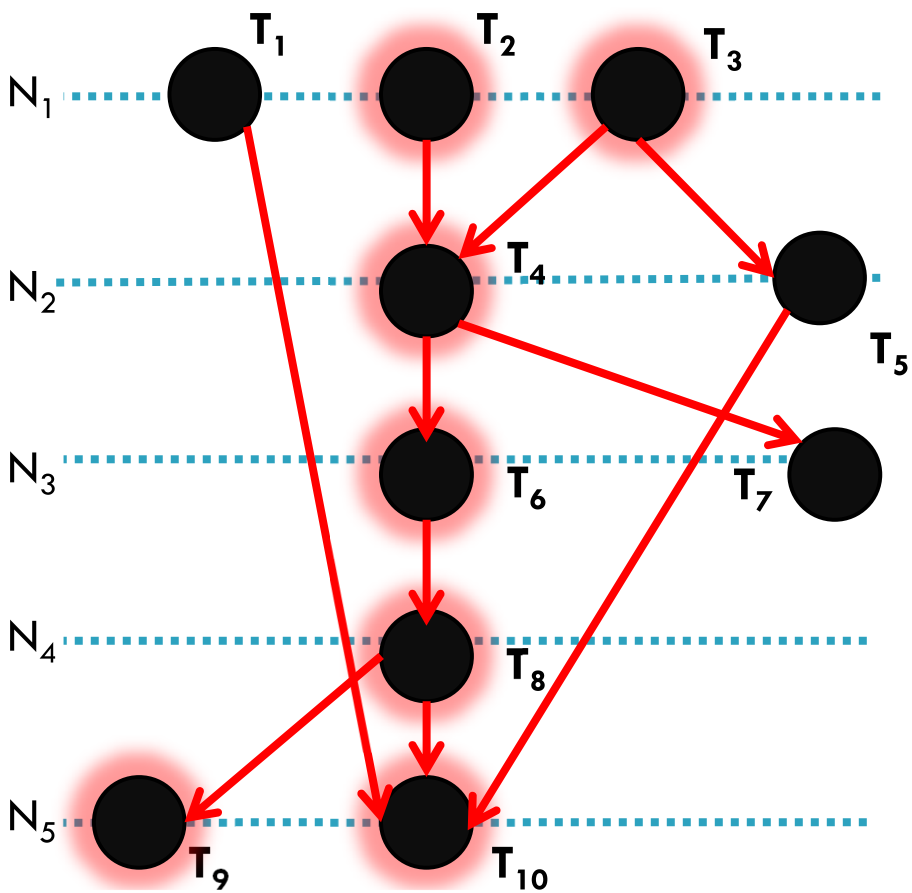
**Les tâches critiques sont toujours dans le même niveau**

## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta)+1$




$N_{tôt}(i) = (tet_i/\theta)+1$



# Ordonnancement des tâches

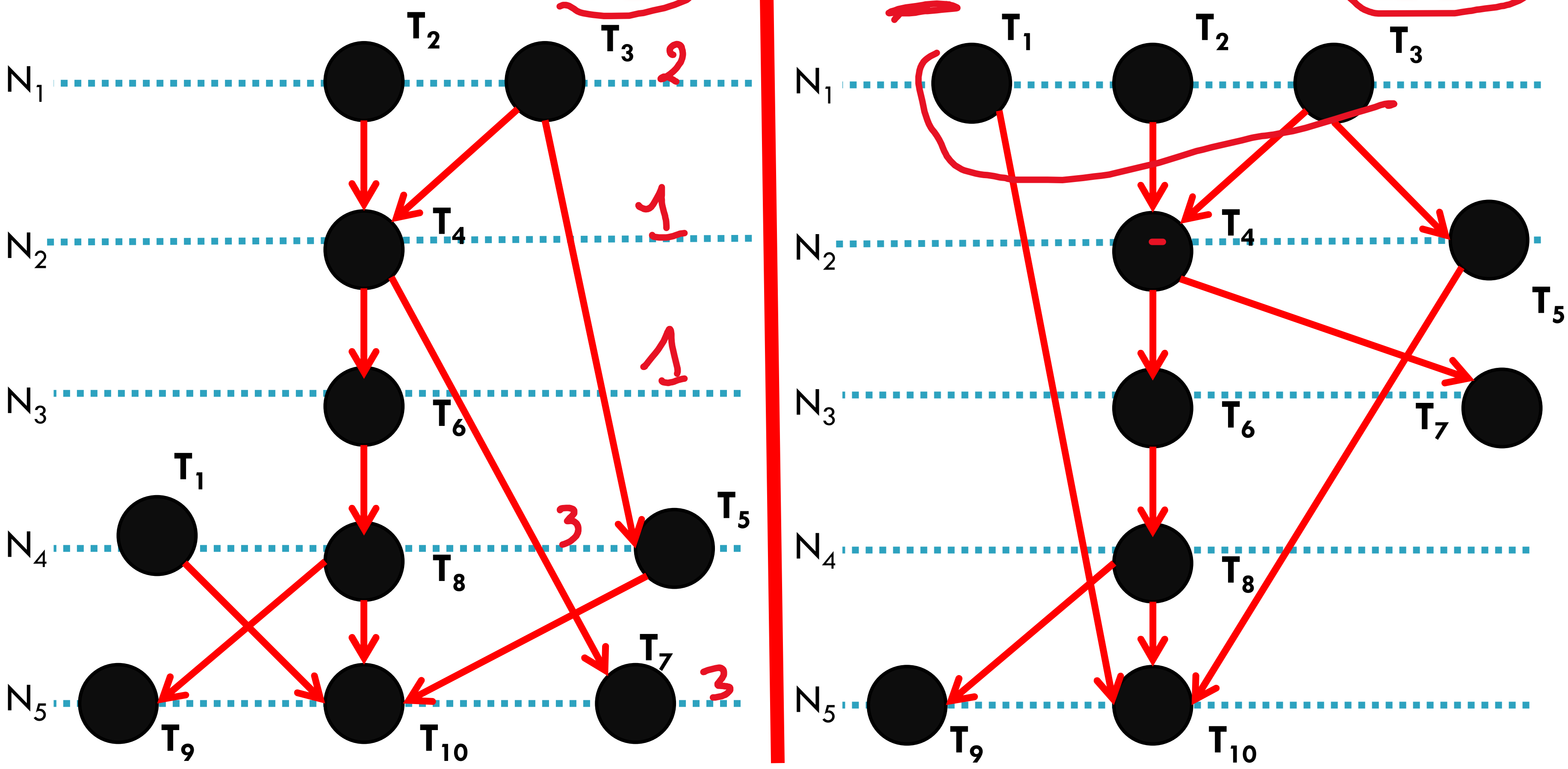


L : largeur du graphe = max du nombre de nœud dans un niveau

## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta)+1 \rightarrow$ LT=3

$N_{tôt}(i) = (tet_i/\theta)+1 \rightarrow$ Lt=3



# Ordonnancement des tâches

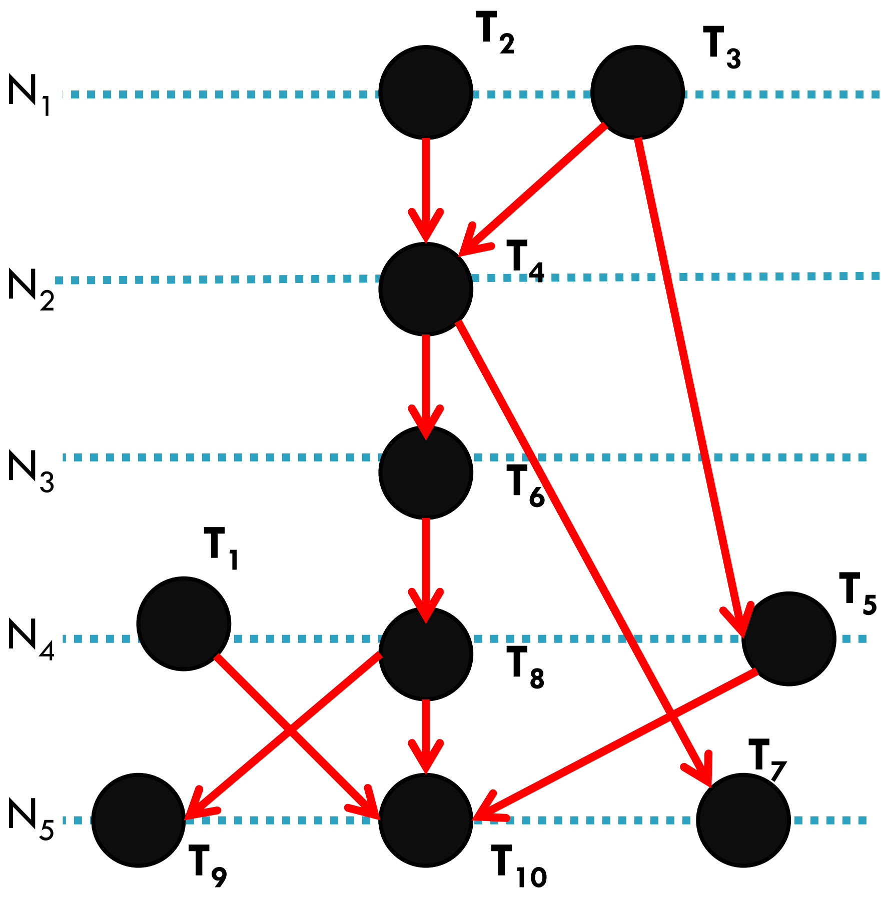


$P_{opt} \leq \min(\text{Lt}, \text{LT})$

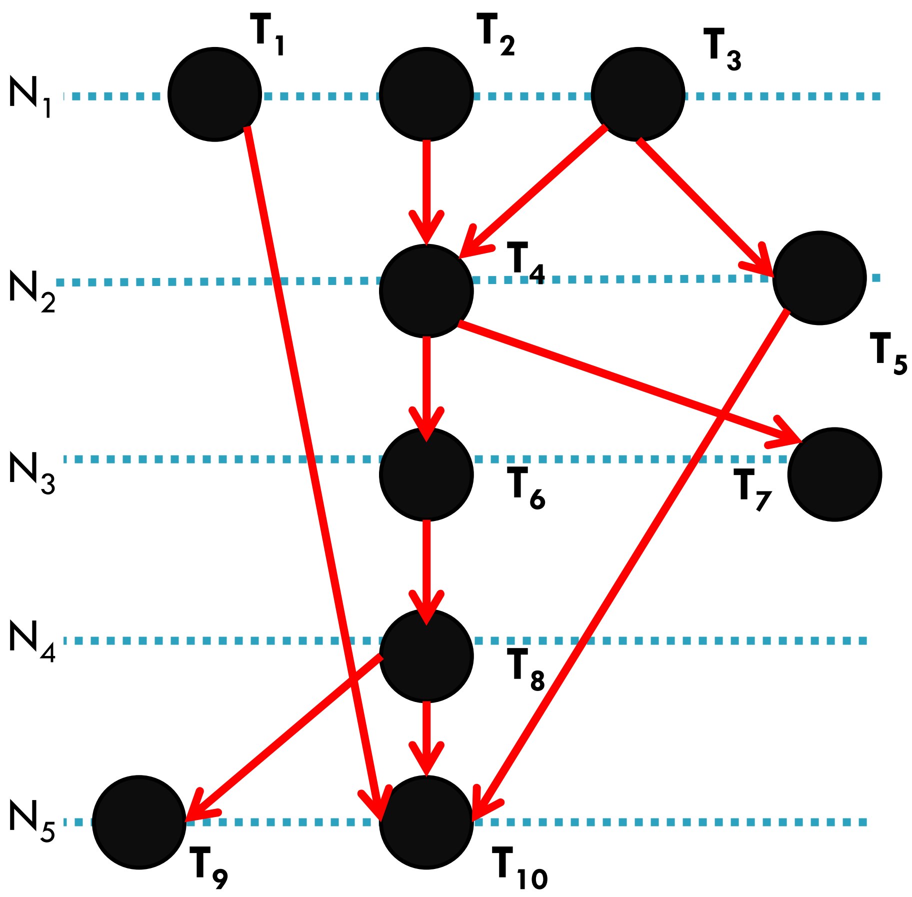
$\leq \min(3,3)$

## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta)+1$




$N_{tôt}(i) = (tet_i/\theta) + 1$



tôt de

# Ordonnancement des tâches



$T_{opt} = \theta * h(G)$

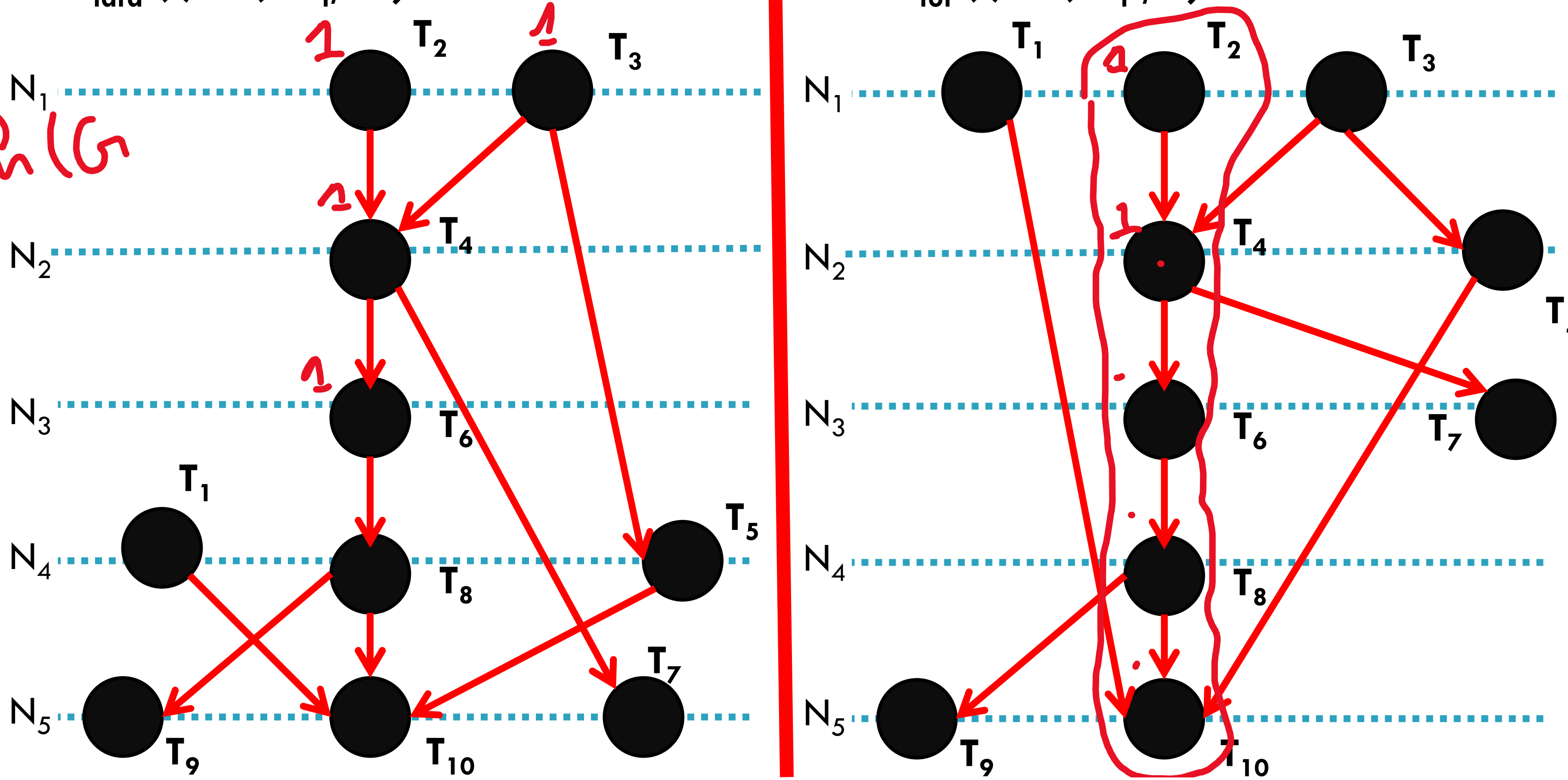
$T_{seq} = \theta * n \rightarrow S = n/h(G)$

**3. Détermination de $P_{opt}$**

$N_{tard}(i) = (teT_i/\theta) + 1$

$N_{tôt}(i) = (tet_i/\theta) + 1$

× h(G



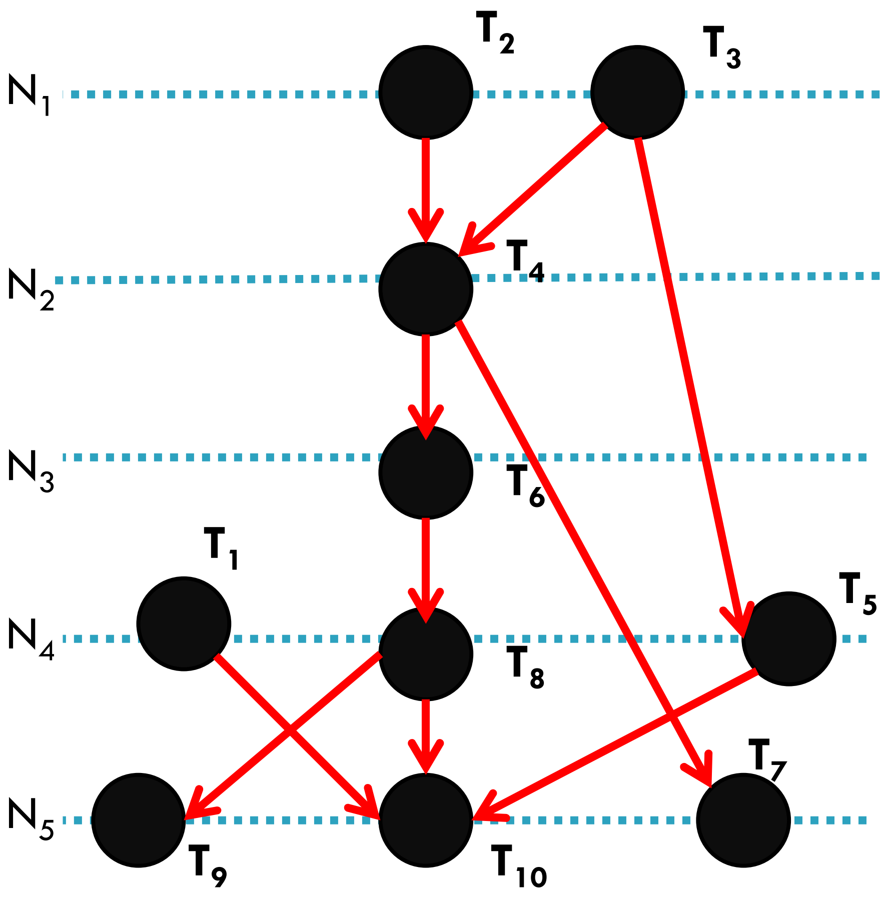
# Ordonnancement des tâches



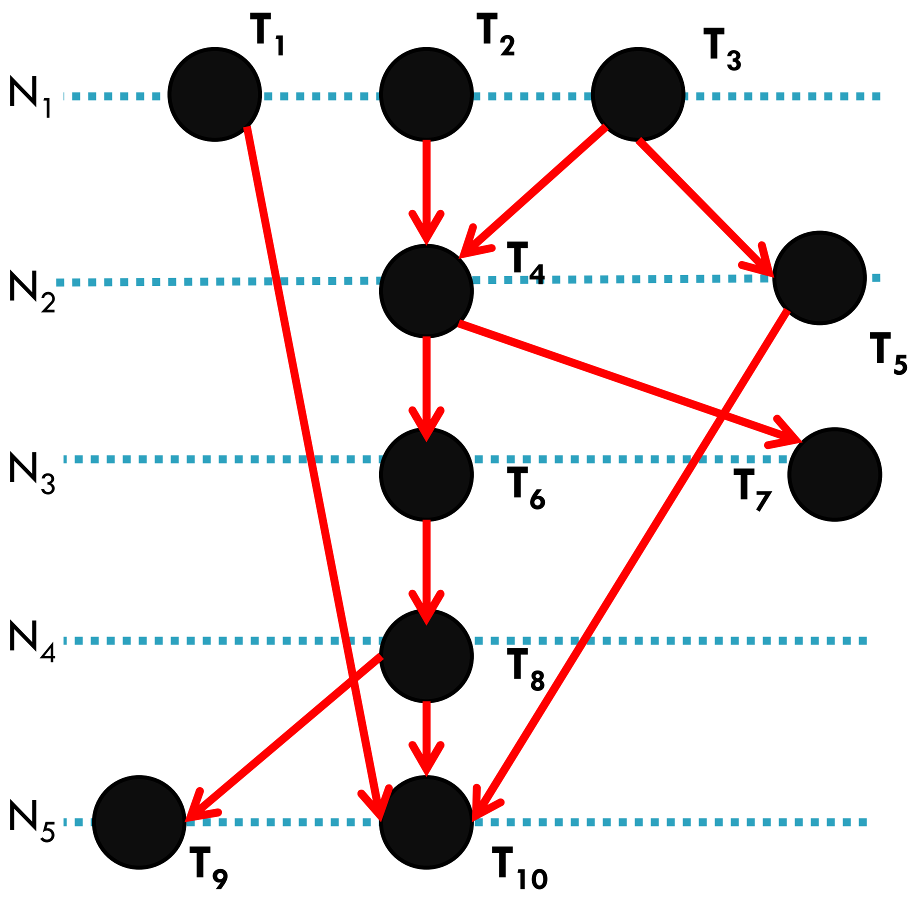
$n/h(G) \leq P_{opt} \leq \min(Lt, LT)$

## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta) + 1$




$N_{tôt}(i) = (tet_i/\theta) + 1$



# Ordonnancement des tâches

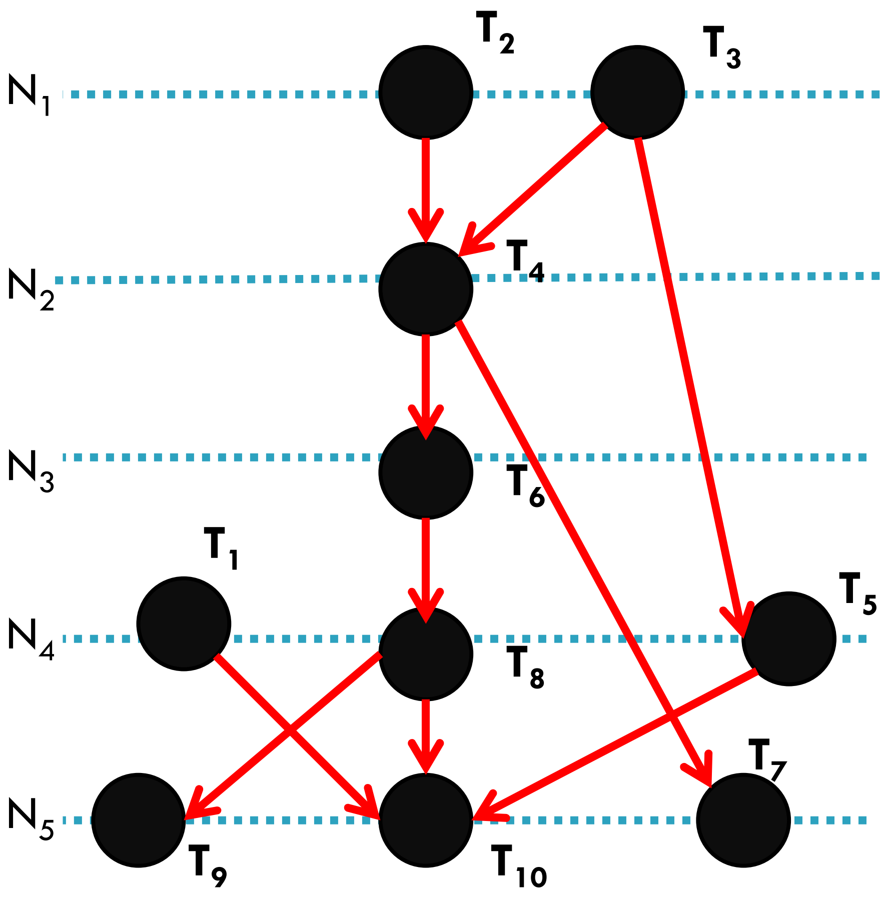


## 3. Détermination de $P_{opt}$

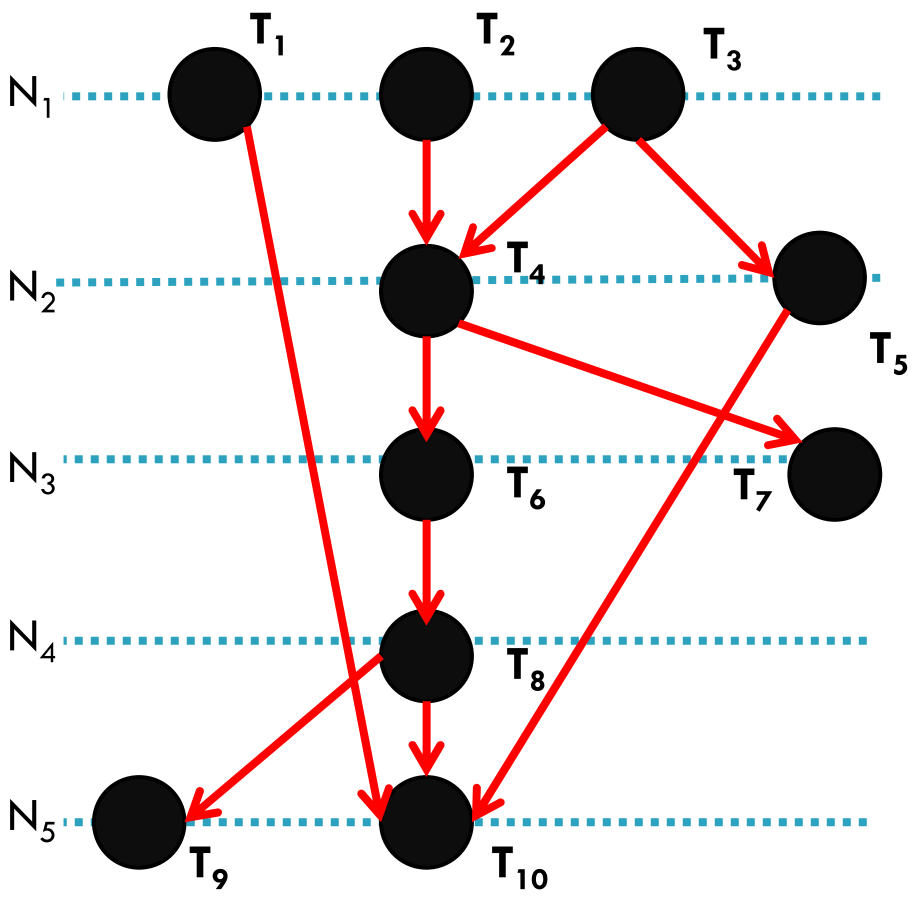
$n/h(G) \leq P_{opt} \leq \min(Lt, LT)$

Dans notre cas $2 \leq P_{opt} \leq 3$

$N_{tard}(i) = (teT_i/\theta) + 1$




$N_{tôt}(i) = (tet_i/\theta) + 1$

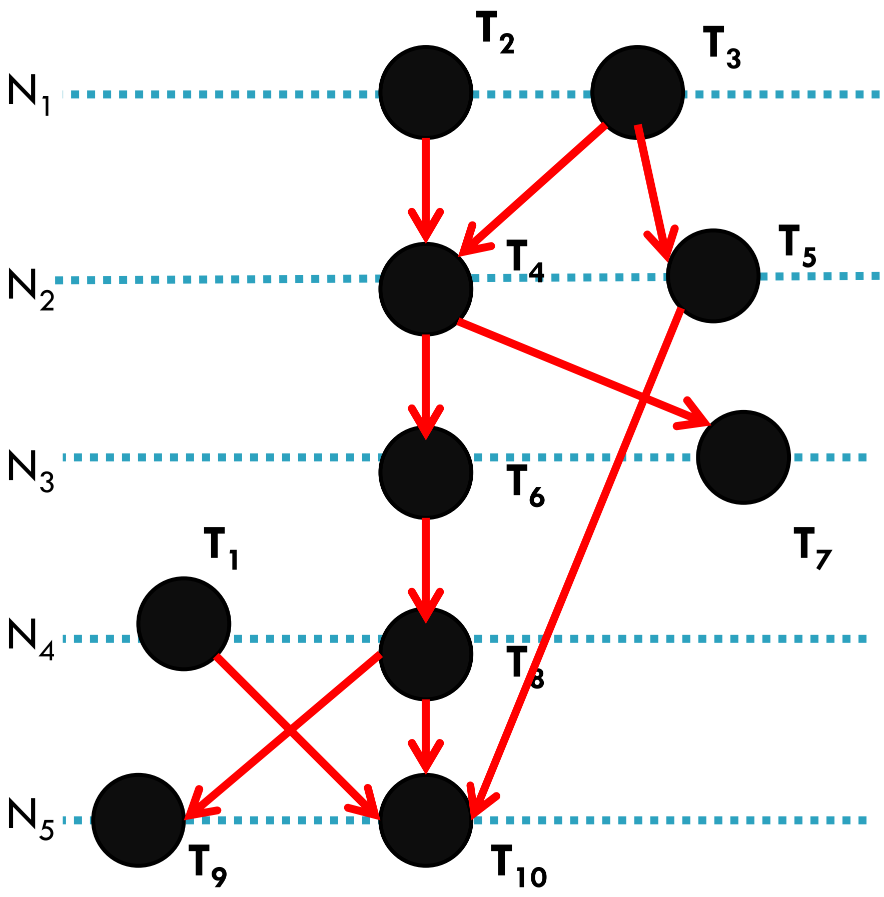


# Ordonnancement des tâches



$P_{opt}$ = 2

### 3. Détermination de $P_{opt}$



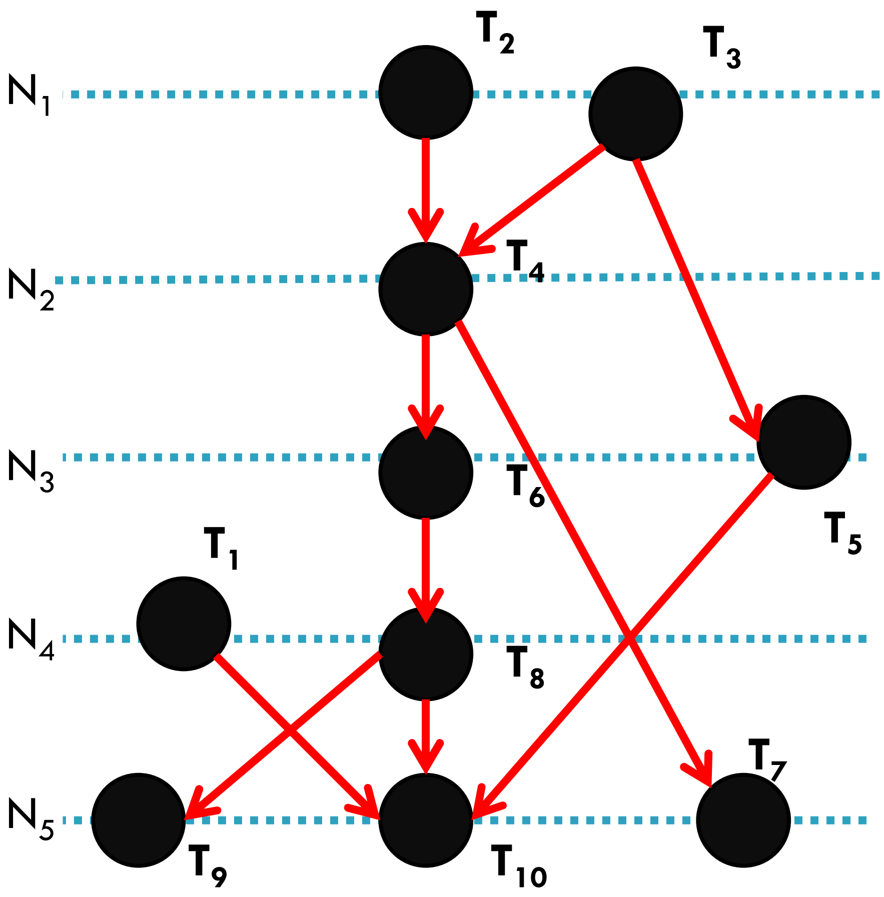
# Ordonnancement des tâches



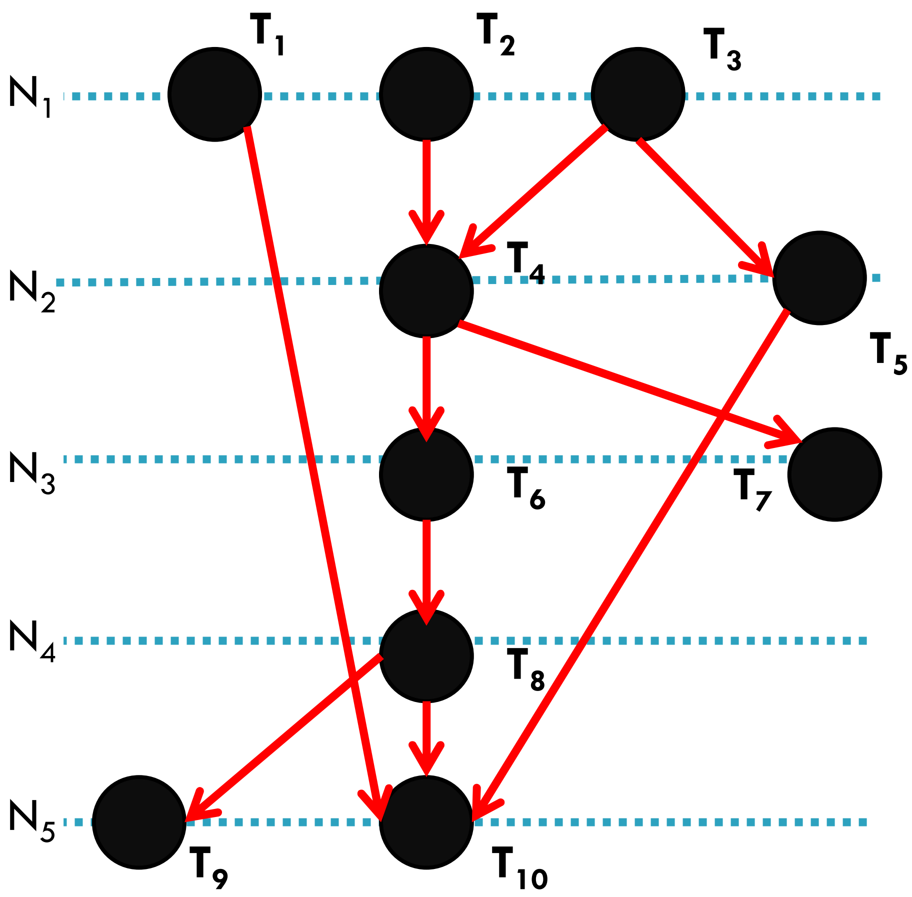
$\max(Lt, LT) \leq P_{max} \leq \max L(D(G))$

## 3. Détermination de $P_{opt}$

$N_{tard}(i) = (teT_i/\theta)+1$




$N_{tôt}(i) = (tet_i/\theta)+1$

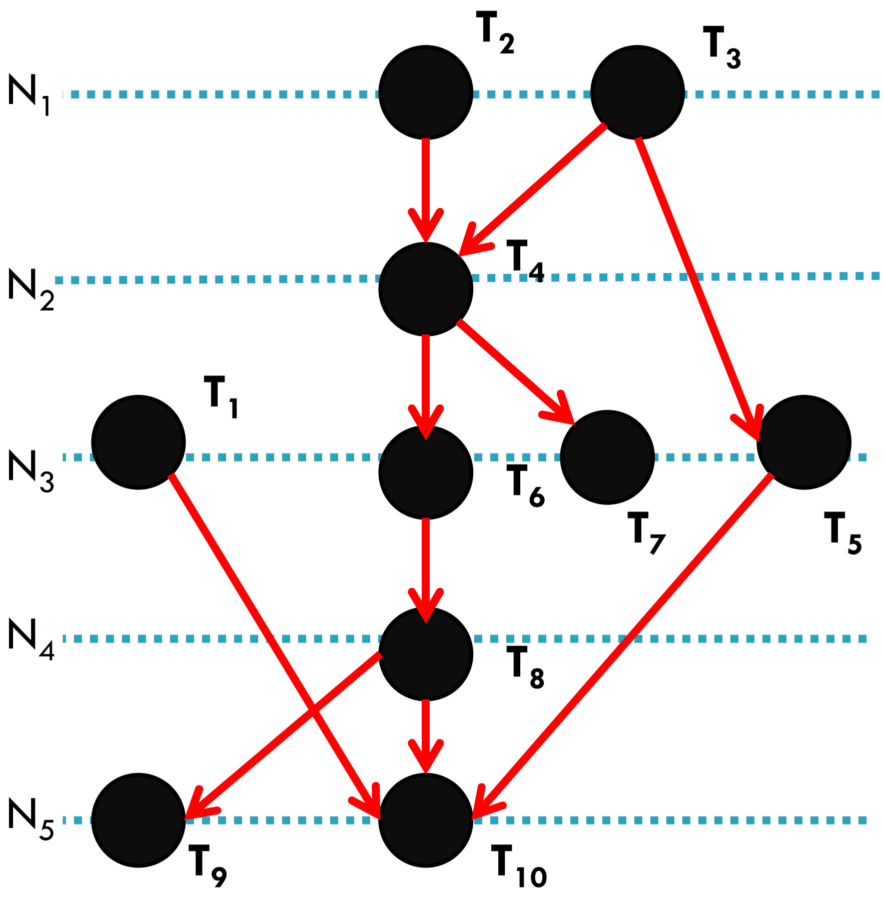


# Ordonnancement des tâches



**3. Détermination de $P_{opt}$**

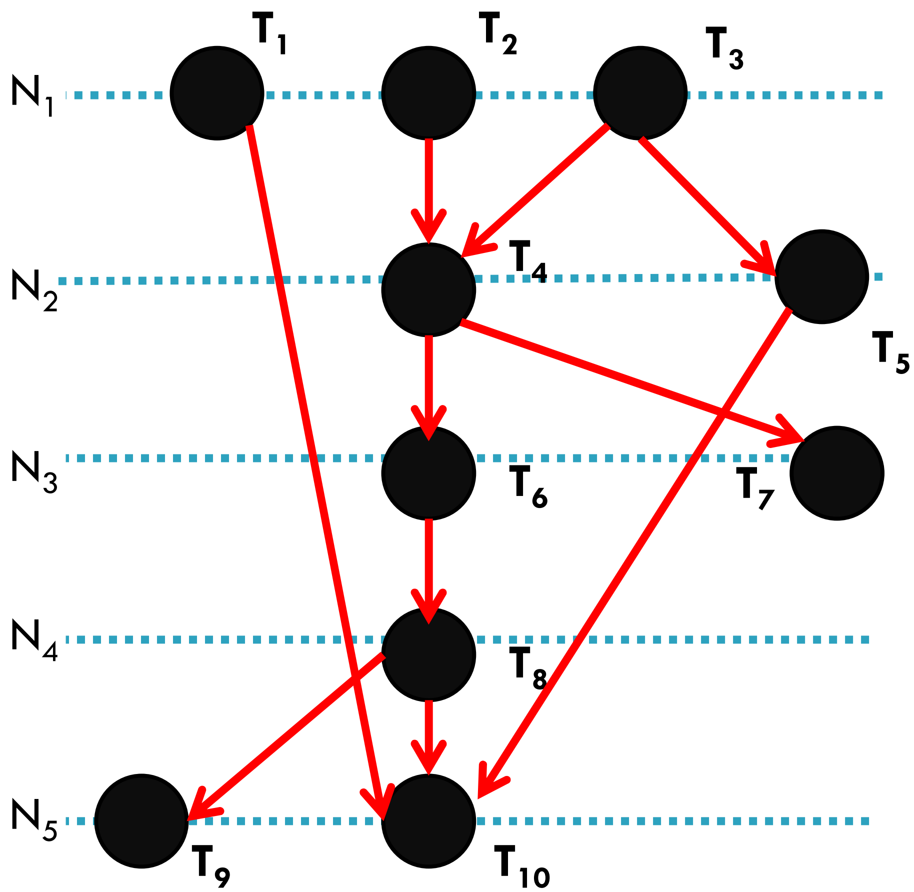
**$\max(Lt, LT) \leq P_{max} \leq \max L(D(G))$**
**Dans notre cas $P_{max} = 4$**



# Ordonnancement des tâches



**3. Détermination de $P_{opt}$**



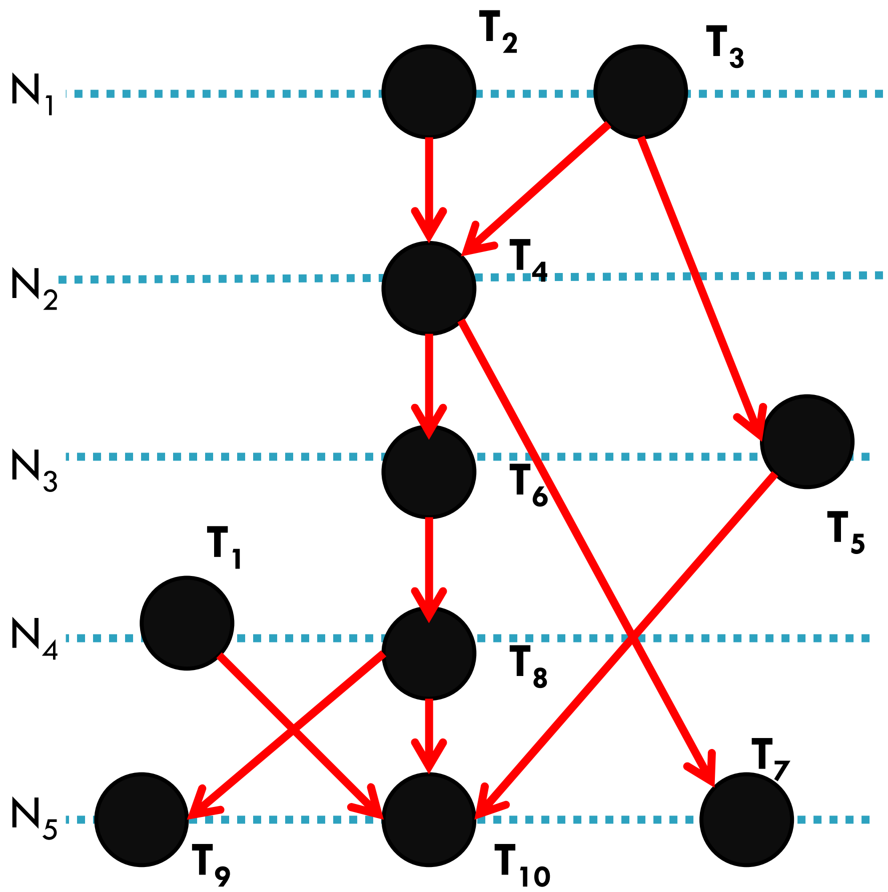

**ORD au plus tôt**

| | | | | | |
|---|---|---|---|---|---|
| Proc1 | T2 | T4 | T6 | T8 | T10 |
| Proc2 | T1 | T5 | T7 | | T9 |
| Proc3 | T3 | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 |

# Ordonnancement des tâches



## 3. Détermination de $P_{opt}$



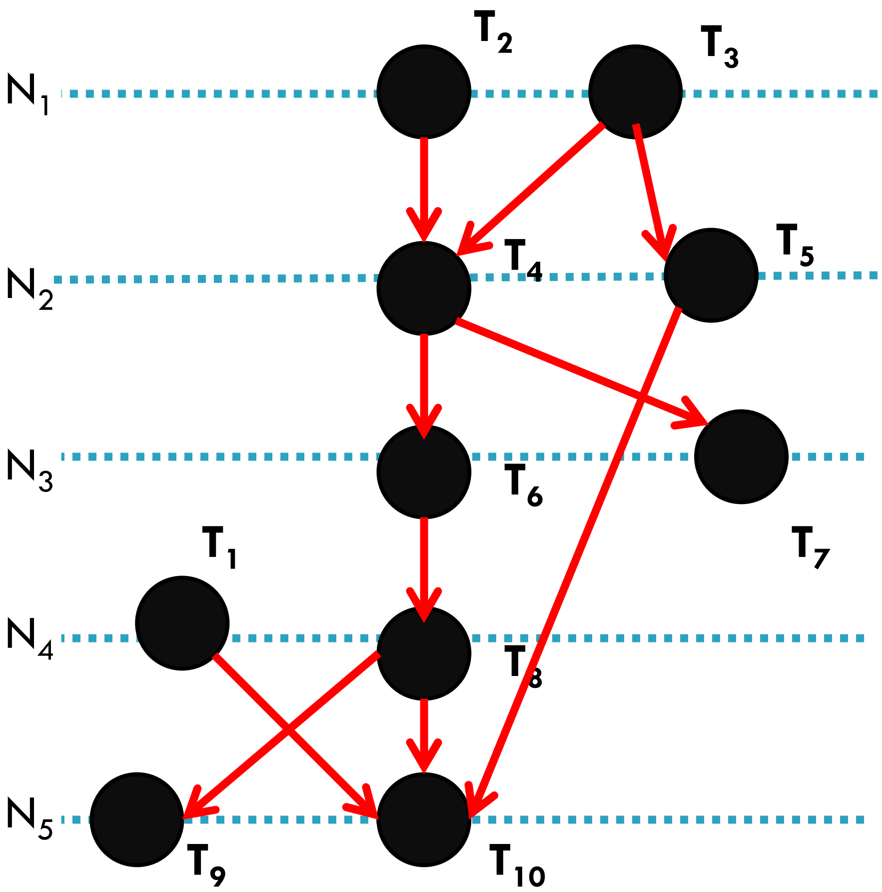

**ORD au plus tard**

| | | | | | |
|---|---|---|---|---|---|
| Proc1 | T2 | T4 | T6 | T8 | T10 |
| Proc2 | T3 | | T5 | T1 | T9 |
| Proc3 | | | | | T7 |
| 0 | 1 | 2 | 3 | 4 | 5 |

# Ordonnancement des tâches



## 3. Détermination de $P_{opt}$



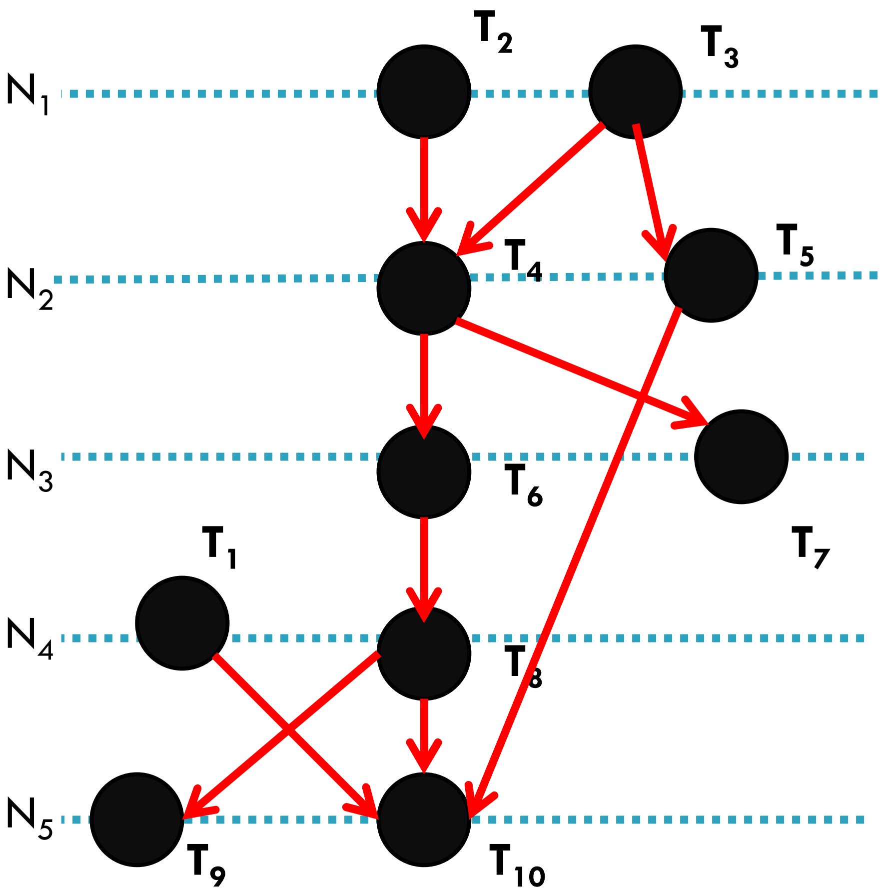

**ORD optimal**

| | | | | | |
|---|---|---|---|---|---|
| Proc1 | T2 | T4 | T6 | T8 | T10 |
| Proc2 | T3 | T5 | T7 | T1 | T9 |
| 0 | 1 | 2 | 3 | 4 | 5 |

# Ordonnancement des tâches



## 3. Détermination de $P_{opt}$




**ORD optimal**

| | | | | | |
|---|---|---|---|---|---|
| **Proc1** | **T2** | **T4** | **T6** | **T8** | **T10** |
| **Proc2** | **T3** | **T5** | **T7** | **T1** | **T9** |
| 0 | 1 | 2 | 3 | 4 | 5 |

Avec trois processeurs on a $T_{opt}$ mais le coût n'est pas optimal en nombre de processeur