

Formation Automatique et Informatique Industrielle

Master 1 S2

Matière : Systèmes Embarqués et Systèmes
Temps Réel SE-STR

Par : ATOUI Hamza

Plan du cours

- Politiques d'ordonnancement:
 - Partie 2 (Tâches apériodiques indépendance à contrainte souple):
 - **First Come First Served (FCFS).**
 - **Shortest Job First (SJF).**
 - Preemptive (Shortest Remaining Time –SRT-).
 - Non- Preemptive.
 - **Priority Based (PB).**
 - Preemptive.
 - Non- Preemptive.
 - **Round Robin (RR).**

Politiques d'ordonnancement

FIRST COME FIRST SERVED (FCFS)

First Come First Served (FCFS)

- **FCFS** : est un algorithme se base sur l'exécution de la tâche qui arrive en premier par une **Ready Queue** en **FIFO** (First In First Out).
- La **plus haute priorité** est donnée au tâche qui **arrive en premier**. Si on a plus d'une tâche **arrive au même temps**, dans ce cas, la priorité est l'**ordre** des tâches.

First Come First Served (FCFS)

- Le temps d'allocation du CPU au tâche choisi par FCFS est **coopératif (la tâche prend le CPU jusqu'à la terminaison de C)**.
- Le modèle utilisé par FCFS est : **Task(r0, C, O)**.
 - **r0** : Arrival Date.
 - **C** : Burst Time (Capacity).
 - **O** : Order of task.
- Temps de charge du noyau : Négligeable.
- On n'a pas de délai critique ($D = \infty$).

First Come First Served (FCFS)

- Etapes à suivre pour ordonnancer efficacement un ensemble de tâches par FCFS:
- **Etape 1** : Détermination de priorité des différentes tâches par : on va associer la plus haute **priorité** à la tâche qui arrive en premier. Si on a plus d'une tâche **arrive au même temps**, dans ce cas, la priorité est l'**ordre** des tâches.

First Come First Served (FCFS)

- **Etape 2** : de tracer les chronogrammes par :
 - De signaler sur long de chronogramme de chaque tâche : $\{r \uparrow\}$.
- D'ordonnancer les tâches selon l'algorithme dans le diapo suivant.
- De tracer le chrono **Gantt Chart (GC)**.

First Come First Served (FCFS)

- Algorithme d'ordonnancement par FCFS :

À chaque TICK_SYSTEM faire

- 1- Chercher les tâches à l'état READY (chargement de ReadyQueue)
- 2- Sélectionner la tâche qui arrive en premier dans le ReadyQueue.
- 3- Dessiner un rectangle sur le chrono de la tâche sélectionner au TICK_SYSTEM en cours.
- 4- Diminuer un TICK le BURST_TIME (C) de la tâche sélectionner.

Répéter les 4 étapes jusqu'à la fin de Burst Time de toutes les tâches

First Come First Served (FCFS)

- Exemple : ordonnancer les tâches suivantes par FCFS :

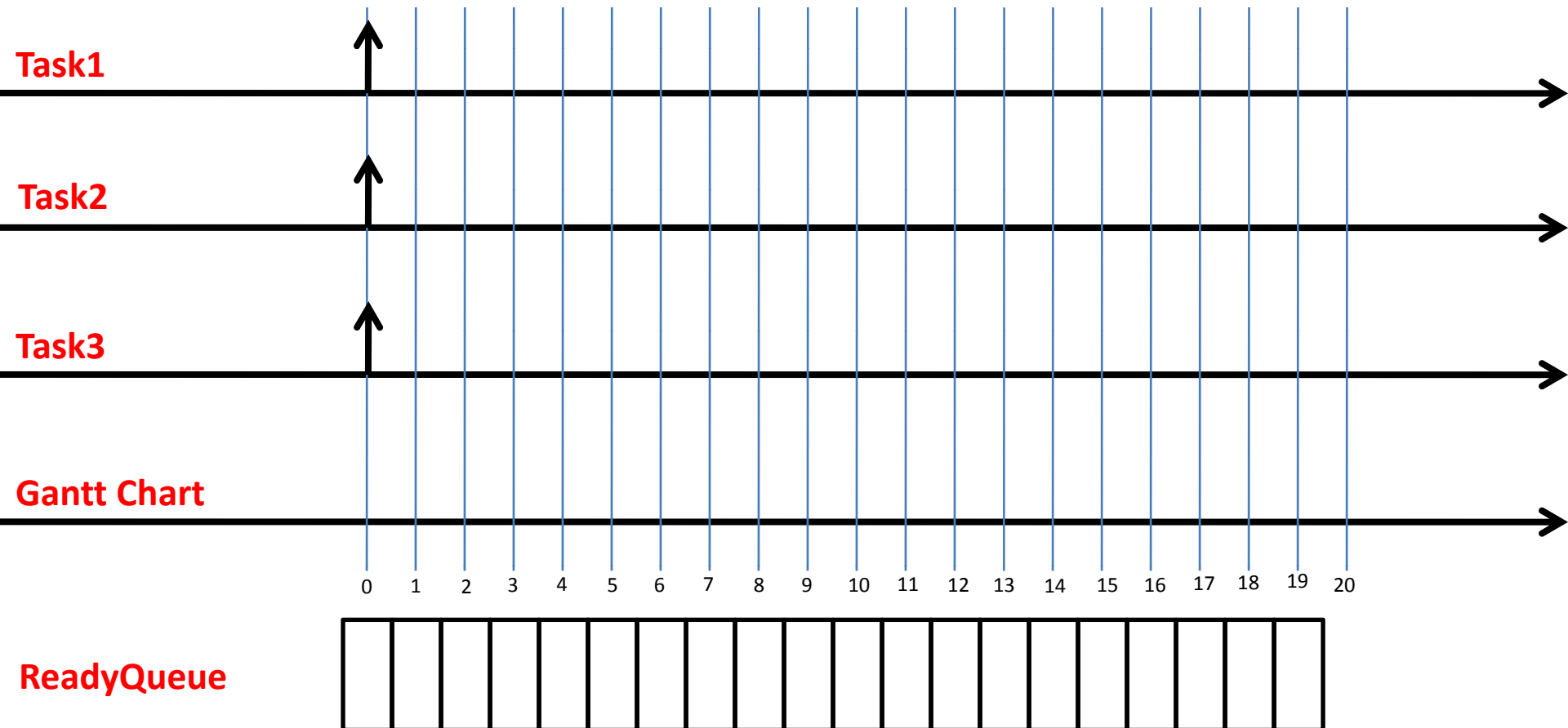
Task	r0	C	O
Task1	0	10	1
Task2	0	2	2
Task3	0	3	3

First Come First Served (FCFS)

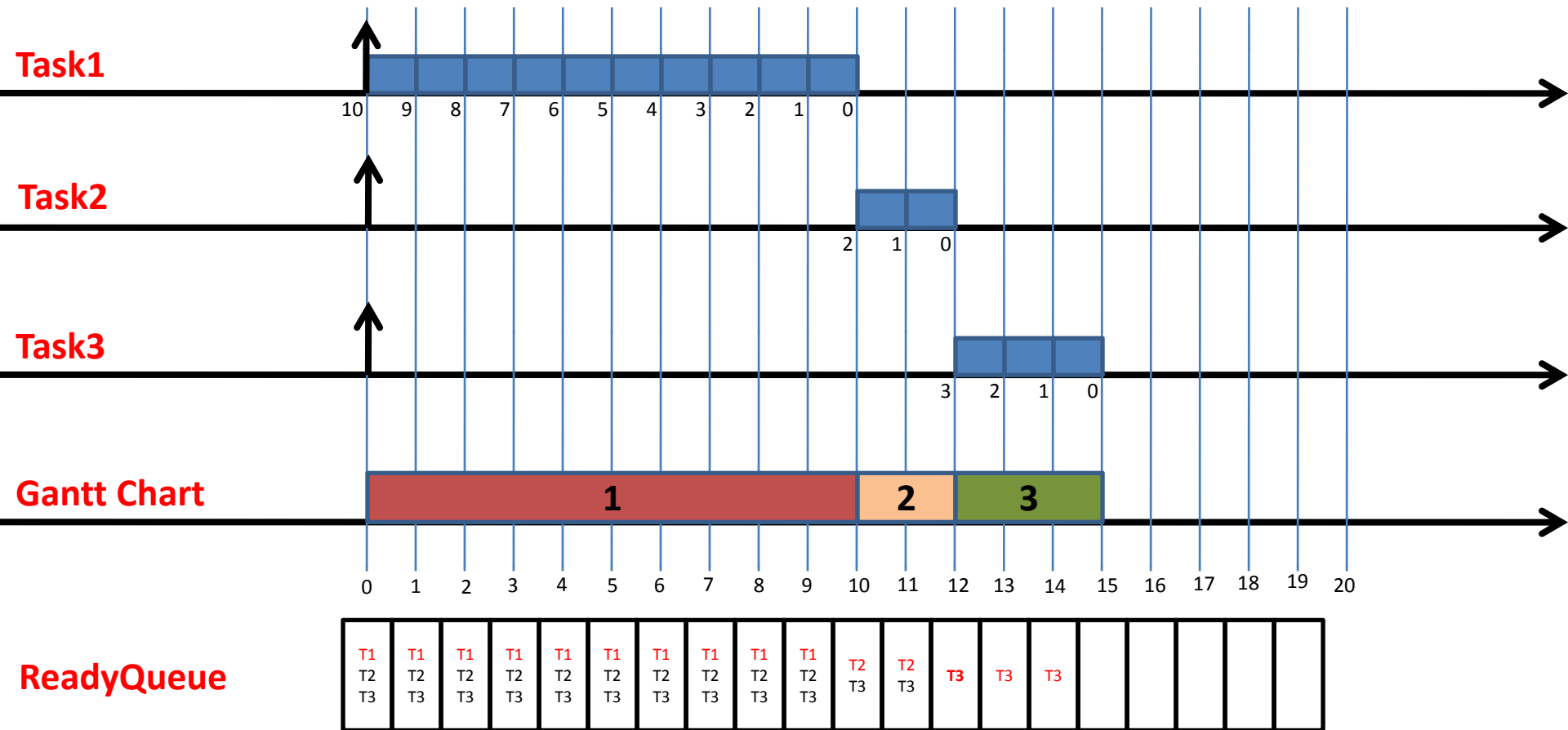
- D'après le tableau de diapo précédent, toutes les tâches arrivent en même temps, donc la priorité est l'ordre des tâches.
- Task1 prioritaire que Task2 et Task2 prioritaire que Task3.

First Come First Served (FCFS)

- Tracer les chronogrammes :



First Come First Served (FCFS)



First Come First Served (FCFS)

- Pour mesurer la performance des algorithmes qu'on va étudier dans cette partie du cours, nous sommes obligés de calculer :
- **Average Waiting Time (AWT)** : le temps moyen d'attente.
- **Average Turn Around Time (ATT)** : le temps moyen de charge (séjour).
- On prend le même exemple pour calculer AWT et ATT.

First Come First Served (FCFS)

Task	r0	C	O	Completion Date (CD)	Turn Around Time (TAT) (CD – r0)	Waiting Time (WT) (TAT - C)
Task1	0	10	1	10	10-0 = 10	10-10 = 0
Task2	0	2	2	12	12-0 = 12	12-2 = 10
Task3	0	3	3	15	15-0 = 15	15-3 = 12

- $AWT = \text{mean}(WT_i) = \text{mean}(0, 10, 12) = 7.33.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(10, 12, 15) = 12.33.$

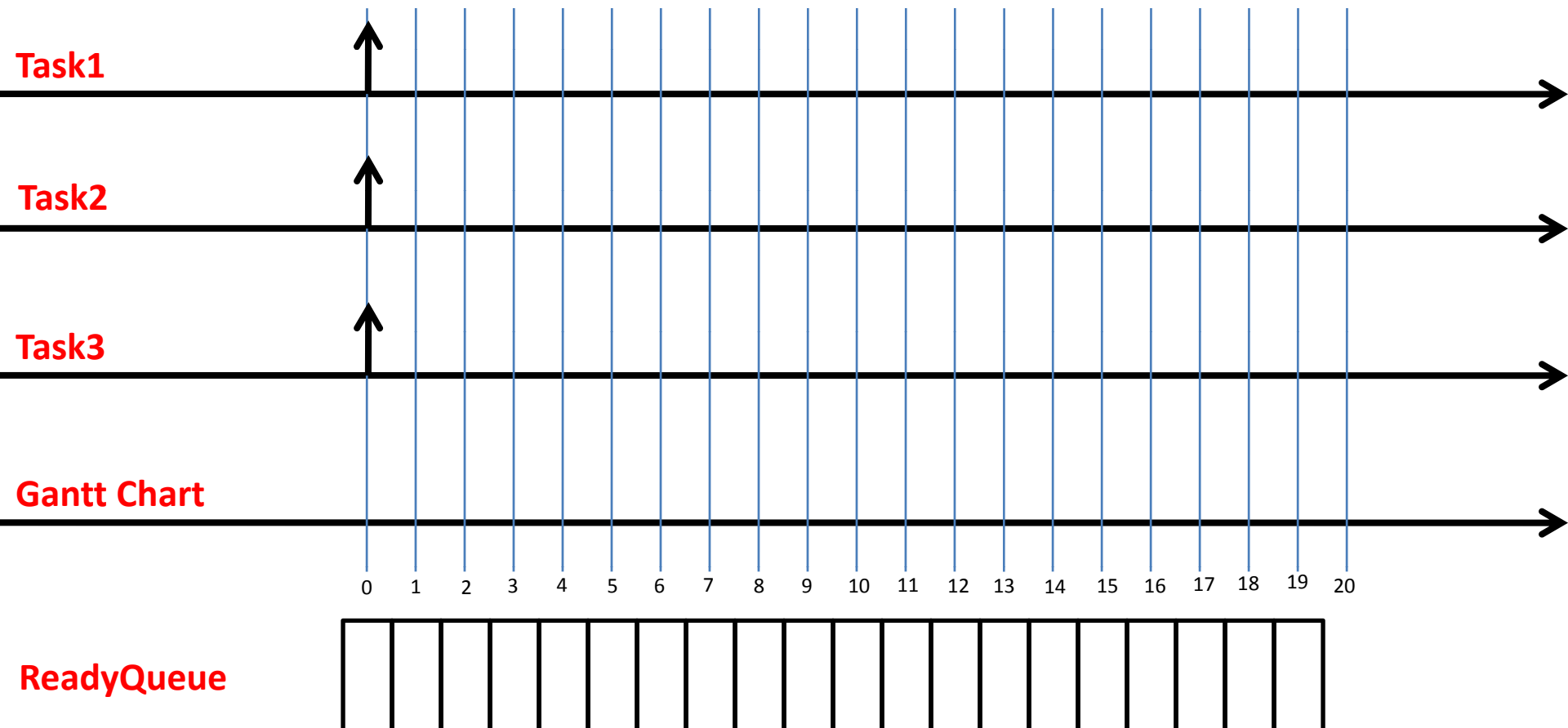
First Come First Served (FCFS)

- Pour augmenter la performance il faut garder le AWT et le ATT le plus faible possible.
- Faire le même exemple (ordonnancement + AWT et ATT) mais avec l'ordre suivante :

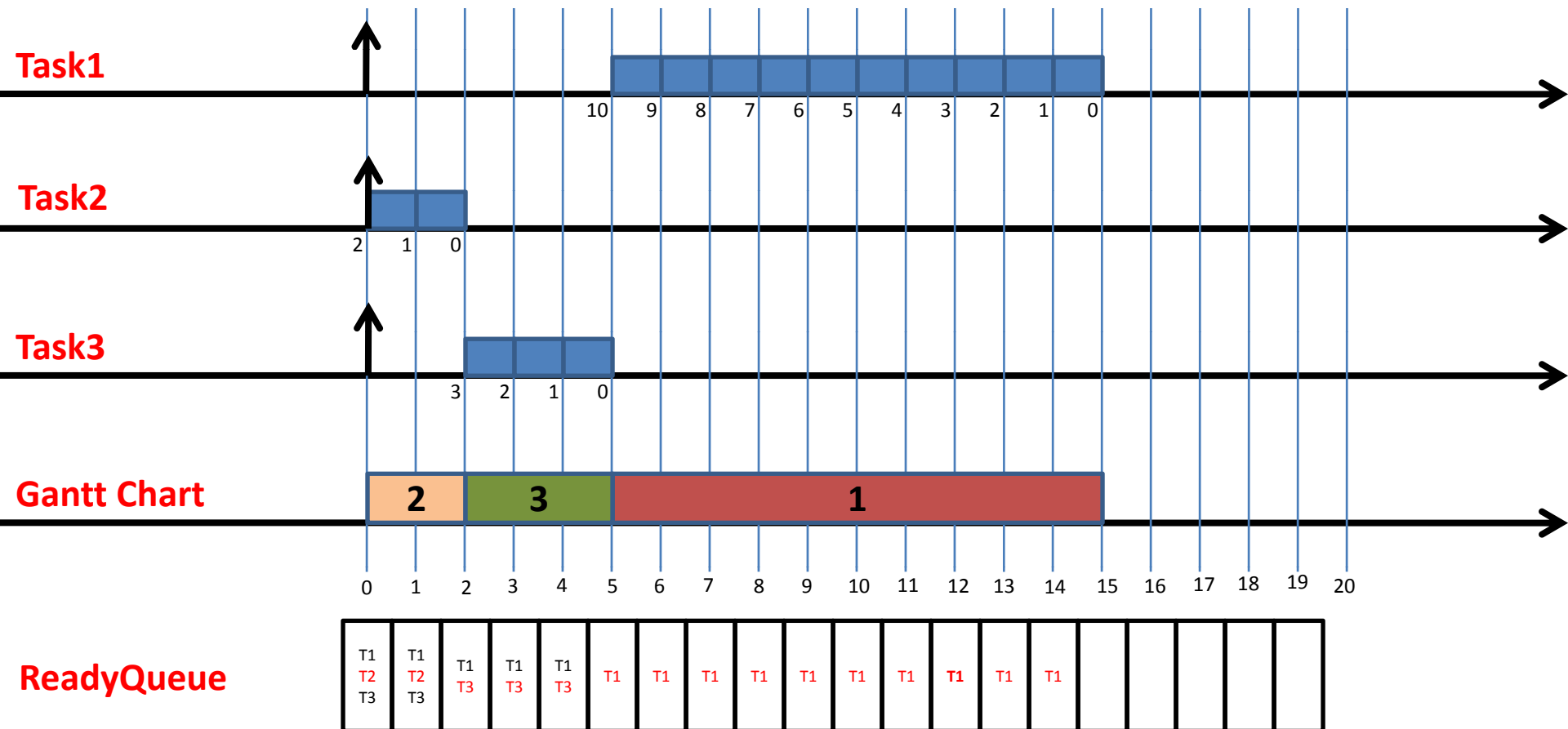
Task	r0	C	O
Task1	0	10	3
Task2	0	2	1
Task3	0	3	2

First Come First Served (FCFS)

- Tracer les chronogrammes :



First Come First Served (FCFS)



First Come First Served (FCFS)

Task	r0	C	O	Completion Date (CD)	Turn Around Time (TAT) (CD – r0)	Waiting Time (WT) (TAT - C)
Task1	0	10	1	15	15-0 = 15	15-10 = 5
Task2	0	2	2	2	2-0 = 2	2-2 = 0
Task3	0	3	3	5	5-0 = 5	5-3 = 2

- $AWT = \text{mean}(WT_i) = \text{mean}(5, 0, 2) = 2.33.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(15, 2, 5) = 7.33.$

First Come First Served (FCFS)

- Que remarquez-vous ???

First Come First Served (FCFS)

- On remarque que le nouveau ordre est performant par rapport au premier ordre.
- En plus, le nouveau ordre est inversement proportionnel à la capacité des tâches.
- Donc, pour améliorer le FCFS, on va associer la plus haute priorité à la tâche de la plus courte capacité.
- L'algorithme qui utilise cette stratégie est le **Shortest Job First Non-Preemptive**.

Politiques d'ordonnancement

SHORTEST JOB FIRST (SJF)

Shortest Job First (SJF)

- **SJF** : est un algorithme se base sur l'exécution de la tâche de la plus petite capacité.
- Si on a plus d'une tâche **a la même capacité et arrive au même temps**, dans ce cas, la priorité est l'**ordre** des tâches.

Shortest Job First (SJF)

- Le temps d'allocation du CPU au tâche choisi par SJF est **coopératif (la tâche prend le CPU jusqu'à la terminaison de C)**.
- Le modèle utilisé par SJF est : **Task(r0, C, O)**.
 - **r0** : Arrival Date.
 - **C** : Burst Time (Capacity).
 - **O** : Order of task.
- Temps de charge du noyau : Négligeable.
- On n'a pas de délai critique ($D = \infty$).

Shortest Job First (SJF)

- Etapes à suivre pour ordonnancer efficacement un ensemble de tâches par SJF:
- **Etape 1** : Détermination de priorité des différentes tâches par : on va associer la plus haute **priorité** à la tâche de la plus petite capacité. Si on a plus d'une tâche **a la même capacité et arrive au même temps**, dans ce cas, la priorité est l'**ordre** des tâches.

Shortest Job First (SJF)

- Algorithme d'ordonnancement par SJF :

À chaque TICK_SYSTEM faire

- 1- Chercher les tâches à l'état READY (chargement de ReadyQueue)
- 2- Sélectionner la tâche de la petite capacité dans le ReadyQueue.
- 3- Dessiner un rectangle sur le chrono de la tâche sélectionner au TICK_SYSTEM en cours.
- 4- Diminuer un TICK le BURST_TIME (C) de la tâche sélectionner.

Répéter les 4 étapes jusqu'à la fin de Burst Time de toutes les tâches

Shortest Job First (SJF)

- Exemple : ordonnancer les tâches suivantes par SJF est calculer le AWT et le ATT :

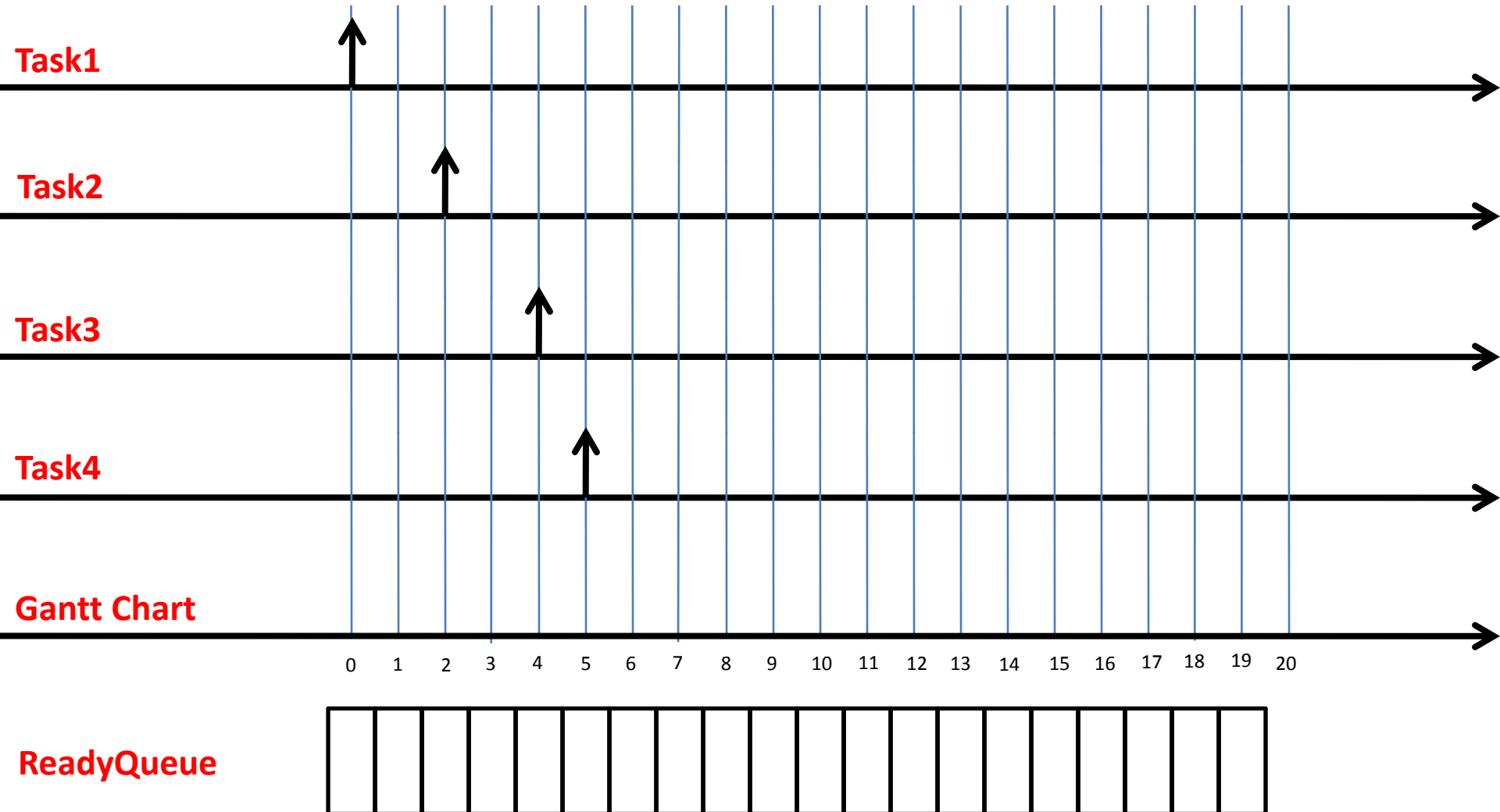
Task	r0	C	O
Task1	0	7	1
Task2	2	4	2
Task3	4	1	3
Task4	5	4	4

Shortest Job First (SJF)

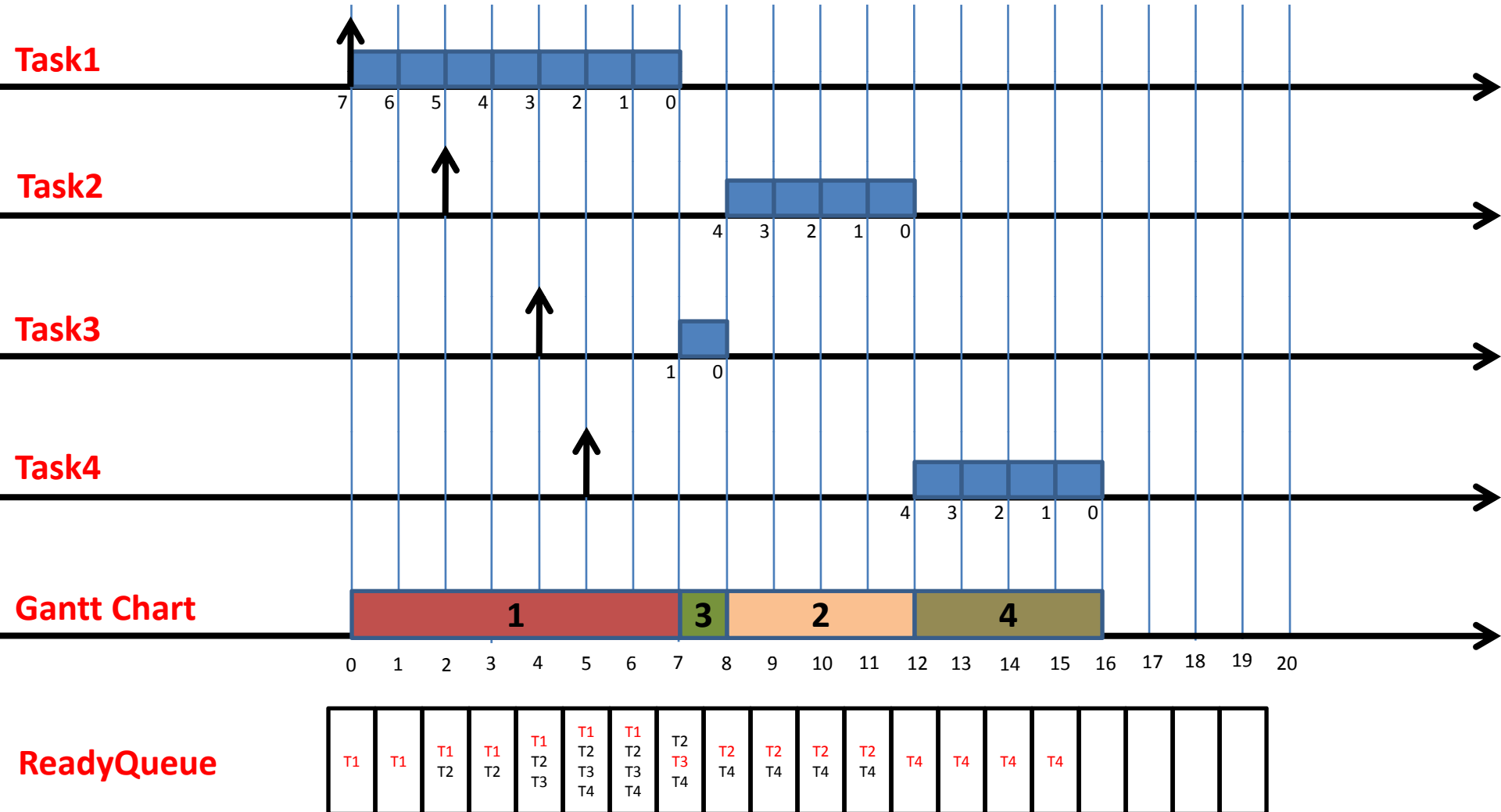
- D'après le tableau de diapo précédent les priorités sont les suivantes :
- Task3 prioritaire que Task2, Task2 prioritaire que Task4 et Task4 prioritaire que Task1.

Shortest Job First (SJF)

- Tracer les chronogrammes :



Shortest Job First (SJF)



Shortest Job First (SJF)

Task	r0	C	O	Completion Date (CD)	Turn Around Time (TAT) (CD - r0)	Waiting Time (WT) (TAT - C)
Task1	0	7	1	7	$7 - 0 = 7$	$7 - 7 = 0$
Task2	2	4	2	12	$12 - 2 = 10$	$10 - 4 = 6$
Task3	4	1	3	8	$8 - 4 = 4$	$4 - 1 = 3$
Task4	5	4	4	16	$16 - 5 = 11$	$11 - 4 = 7$

- $AWT = \text{mean}(WT_i) = \text{mean}(0, 6, 3, 7) = 4.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(7, 10, 4, 11) = 8.$

Shortest Job First (SJF)

- On remarque clairement que le SJF est plus optimal que le FCFS.
- Maintenant, ordonnancer et calculer le AWT et le ATT des deux tâches suivantes :

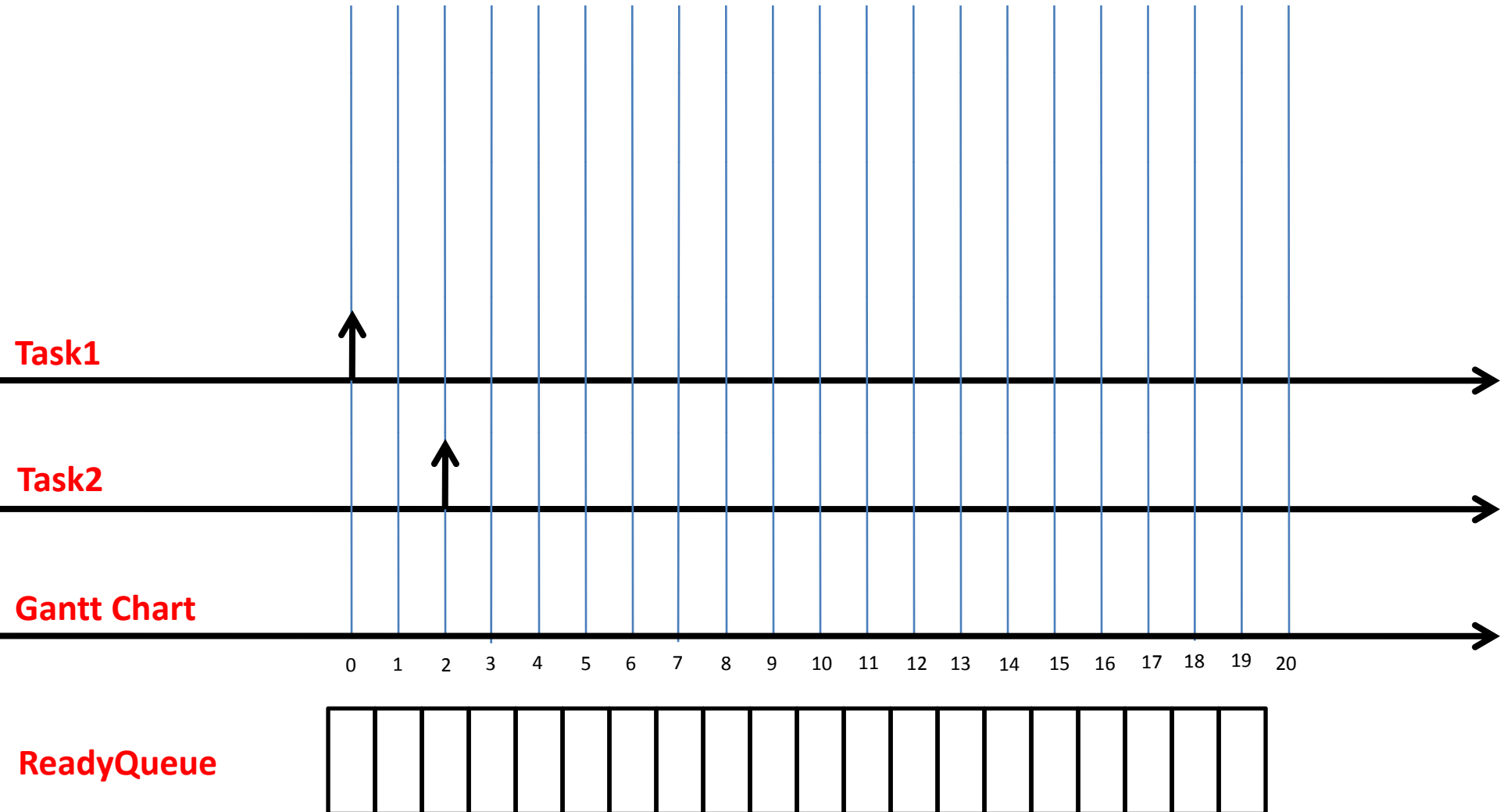
Task	r0	C	O
Task1	0	10	1
Task2	2	2	2

Shortest Job First (SJF)

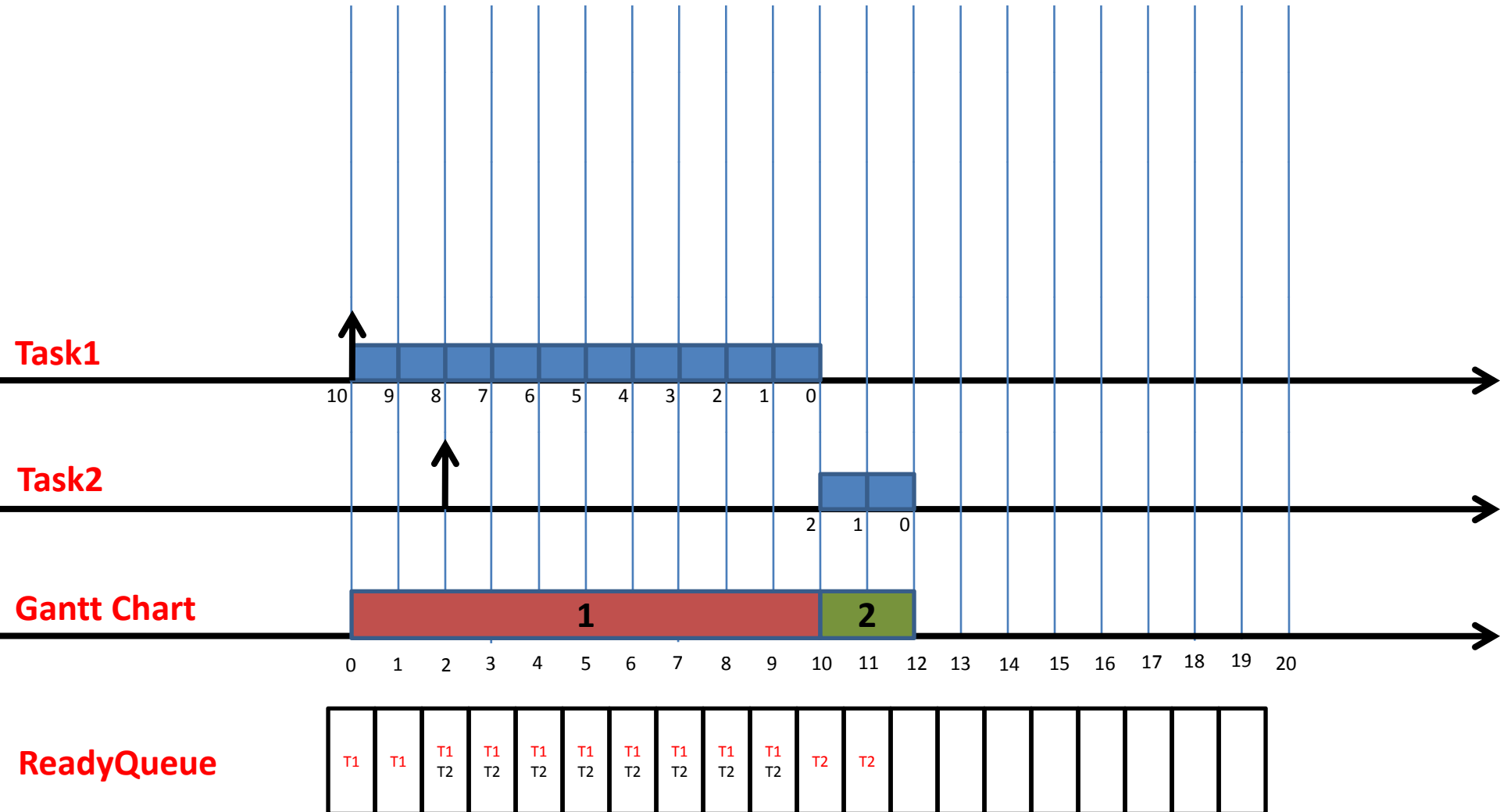
- Calcul de priorité : Task2 prioritaire que Task1.

Shortest Job First (SJF)

- Tracer les chronogrammes :



Shortest Job First (SJF)



Shortest Job First (SJF)

Task	r0	C	O	Completion Date (CD) (Start Date + C)	Turn Around Time (TAT) (CD – r0)	Waiting Time (WT) (TAT - C)
Task1	0	10	1	10	10-0 = 10	10-10 = 0
Task2	2	2	2	12	12-2 = 10	10-2 = 8

- $AWT = \text{mean}(WT_i) = \text{mean}(0, 8) = 4.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(10, 10) = 10.$

Shortest Job First (SJF)

- Malgré que Task2 est prioritaire que Task1, l'ordonnanceur laisser Task1 s'exécute jusqu'à la fin de son capacité (Burst Time) par ce que le temps d'allocation de CPU est coopératif.
- Ce type d'allocation **dégrade** les performances de l'algorithme SJF est devient comme FCFS **au moment des temps d'arrivé sont différents**.
- La plus simple amélioration qu'on va introduire sur cet algorithme est de changer le temps d'allocation de CPU **coopératif** vers **préemptif**.

Shortest Job First (SJF)

- Faire le même exemple mais avec un temps d'allocation préemptif.

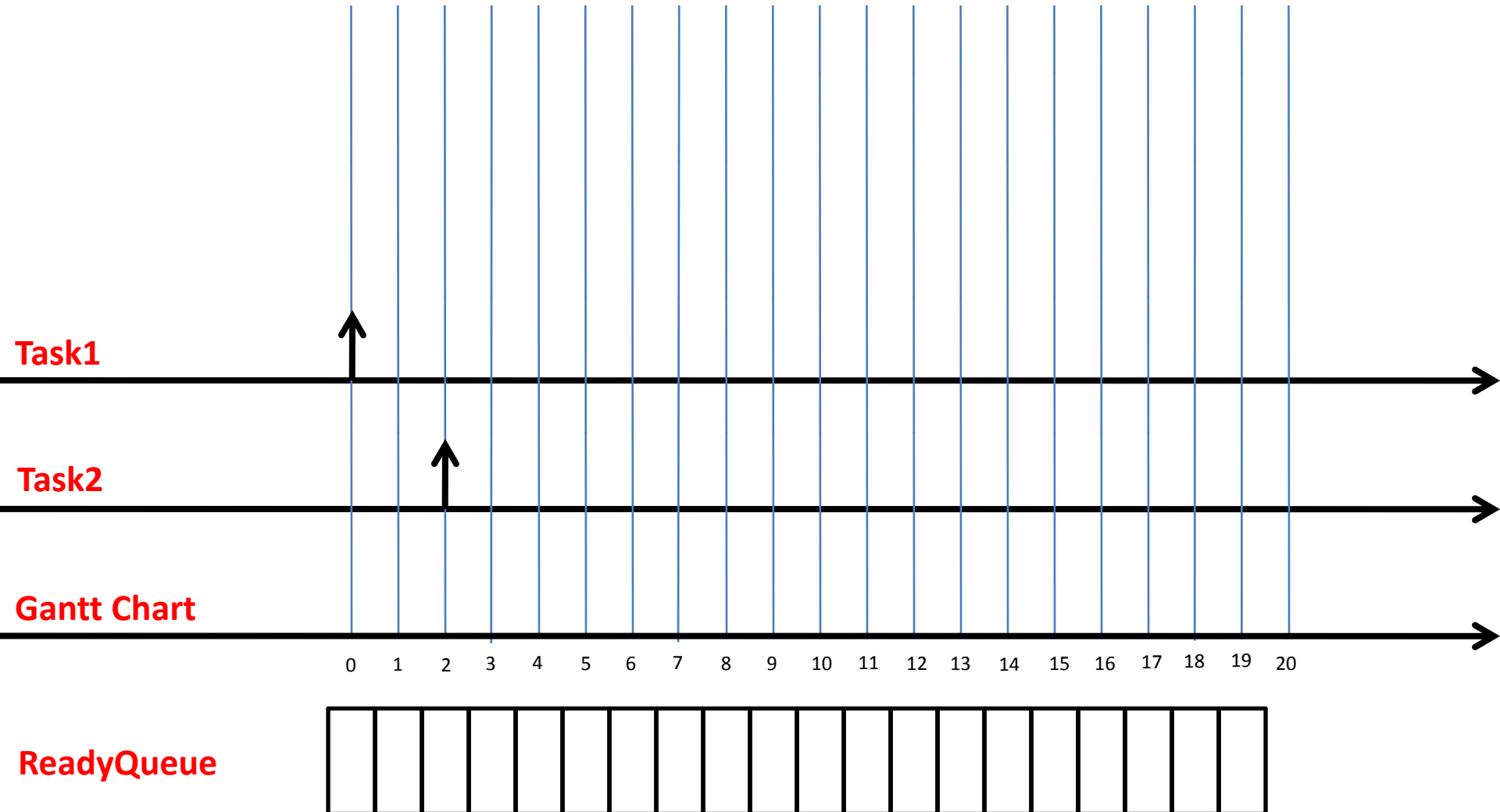
Task	r0	C	O
Task1	0	10	1
Task2	2	2	2

Shortest Job First (SJF)

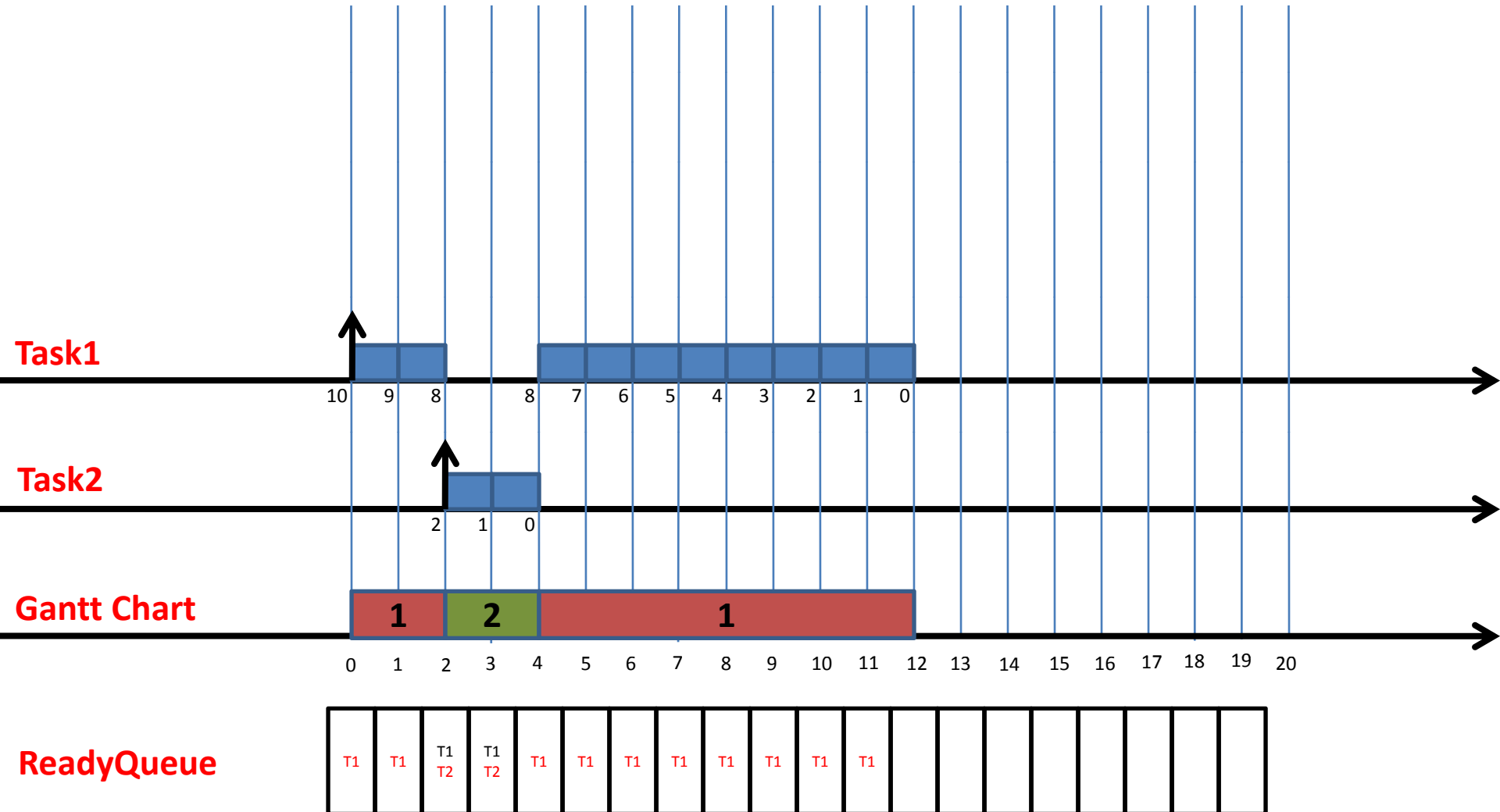
- Calcul de priorité : Task2 prioritaire que Task1.

Shortest Job First (SJF)

- Tracer les chronogrammes :



Shortest Job First (SJF)



Shortest Job First (SJF)

Task	r0	C	O	Completion Date (CD) (Start Date + C)	Turn Around Time (TAT) (CD – r0)	Waiting Time (WT) (TAT - C)
Task1	0	10	1	12	12-0 = 12	12-10 = 2
Task2	2	2	2	4	4-2 = 2	2-2 = 0

- $AWT = \text{mean}(WT_i) = \text{mean}(2, 0) = 1.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(12, 2) = 7.$

Shortest Job First (SJF)

- Cette amélioration prend le nom : **Short Job First Preemptive** ou **Shortest Remaining Time (SRT)**.

Politiques d'ordonnancement

PRIORITY BASED (PB)

Priority Based (PB)

- **PB** : est un algorithme se base sur l'exécution de la tâche de la plus haute priorité.
- La **plus haute priorité** est **fixée par l'utilisateur** selon son besoin.
- Si on a plus d'une tâche **à la même priorité**, dans ce cas, la priorité est donnée à la tâche qui **arrive en premier** et si les tâches de la même priorité arrivent en même temps, la priorité est donnée à l'**ordre** des tâches.

Priority Based (PB)

- Le temps d'allocation du CPU au tâche choisi par PB est **coopératif (la tâche prend le CPU jusqu'à la terminaison de C) ou préemptif**.
- Le modèle utilisé par PB est : **Task(r0, C, O, Pr)**.
 - **r0** : Arrival Date.
 - **C** : Burst Time (Capacity).
 - **O** : Order of task.
 - **Pr** : Priority.
- Temps de charge du noyau : Négligeable.
- On n'a pas de délai critique ($D = \infty$).

Priority Based (PB)

- Etapes à suivre pour ordonnancer efficacement un ensemble de tâches par PB:
- **Etape 1** : de tracer les chronogrammes par :
 - De signaler sur long de chronogramme de chaque tâche : $\{r\uparrow\}$.
 - D'ordonnancer les tâches selon l'algorithme dans le diapo suivant.
 - De tracer le chrono **Gantt Chart (GC)**.

Priority Based (PB)

- Algorithme d'ordonnancement par PB :

À chaque TICK_SYSTEM faire

- 1- Chercher les tâches à l'état READY (chargement de ReadyQueue)
- 2- Sélectionner la tâche de la plus haute priorité dans le ReadyQueue.
- 3- Dessiner un rectangle sur le chrono de la tâche sélectionner au TICK_SYSTEM en cours.
- 4- Diminuer un TICK le BURST_TIME (C) de la tâche sélectionner.

Répéter les 4 étapes jusqu'à la fin de Burst Time de toutes les tâches

Priority Based (PB)

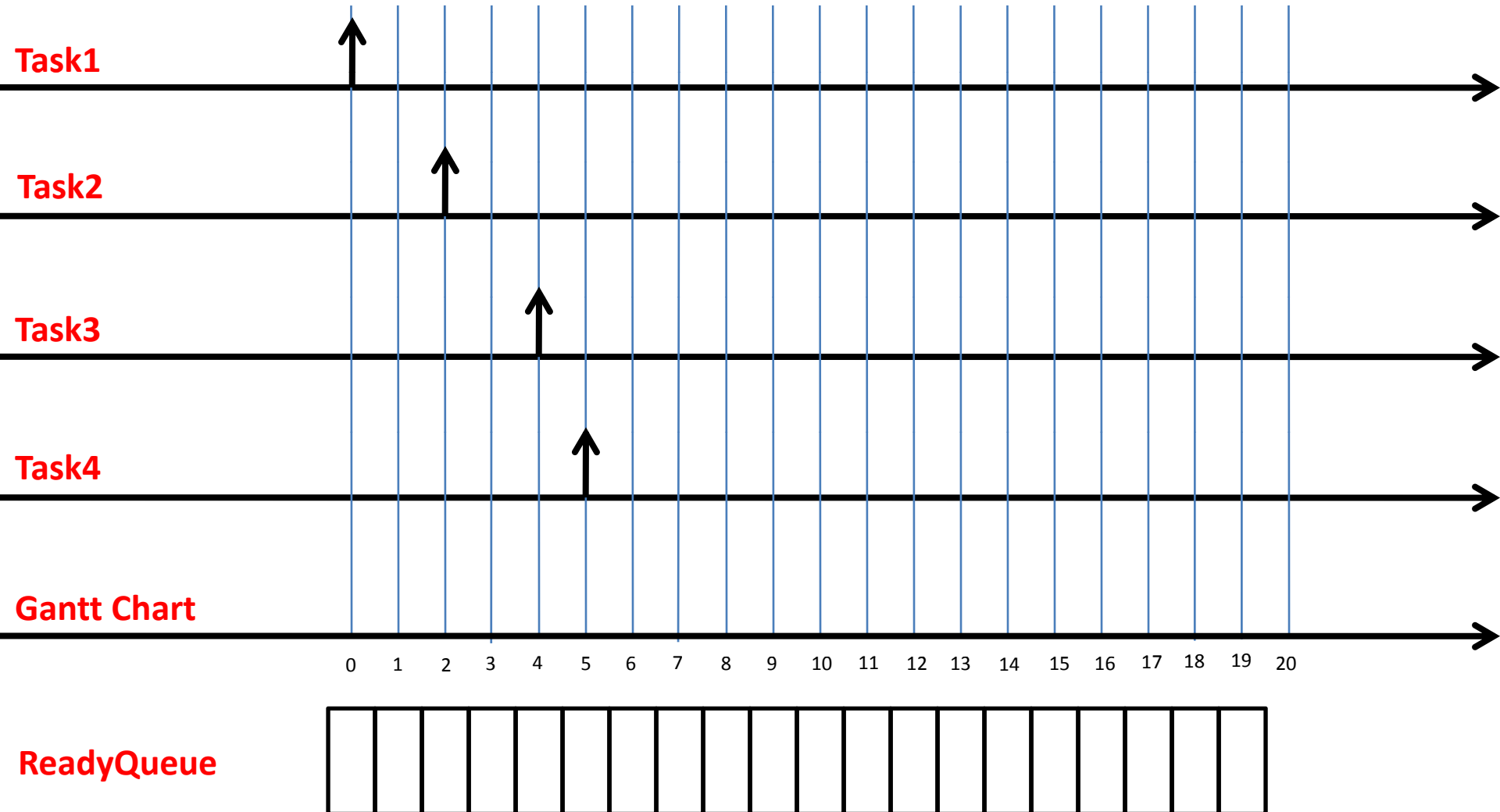
- Exemple : ordonnancer les 4 tâches suivantes par PB-Cooperative et calculer le AWT et le ATT.

Task	r0	C	O	Pr
Task1	0	7	1	3
Task2	2	4	2	2
Task3	4	1	3	4
Task4	5	4	4	1

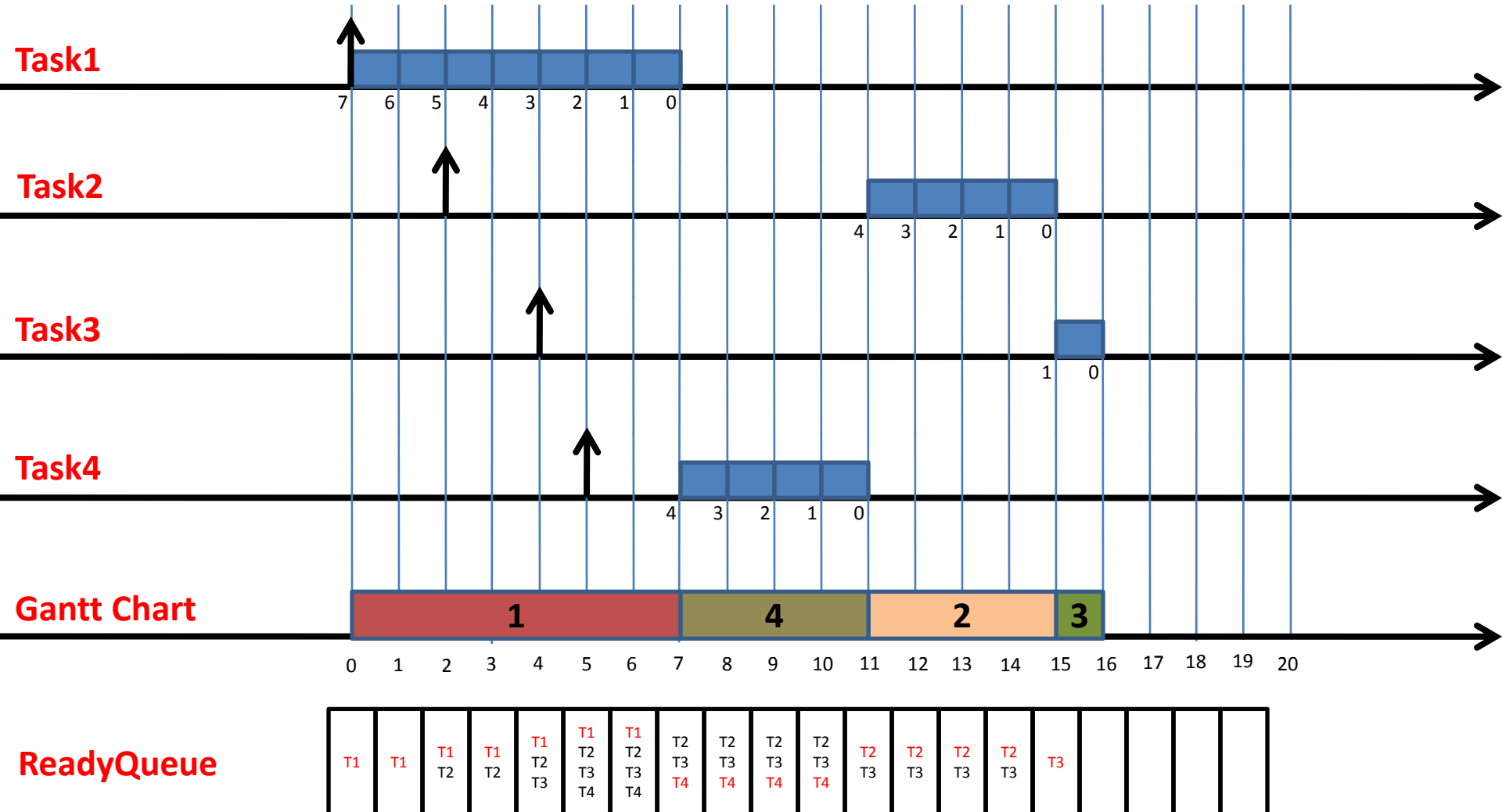
1 is a highest priority and 4 is a lowest priority

Priority Based (PB)

- Tracer les chronogrammes :



Priority Based (PB)



Priority Based (PB)

Task	r0	C	O	Pr	Completion Date (CD)	Turn Around Time (TAT) (CD – r0)	Waiting Time (WT) (TAT - C)
Task1	0	7	1	3	7	7-0 = 7	7-7 = 0
Task2	2	4	2	2	15	15-2 = 13	13-4 = 9
Task3	4	1	3	4	16	16-4 = 12	12-1 = 11
Task4	5	4	4	1	11	11-5 = 6	6-4 = 2

- $AWT = \text{mean}(WT_i) = \text{mean}(0, 9, 11, 2) = 5.50$.
- $ATT = \text{mean}(TAT_i) = \text{mean}(7, 13, 12, 6) = 9.50$.

Priority Based (PB)

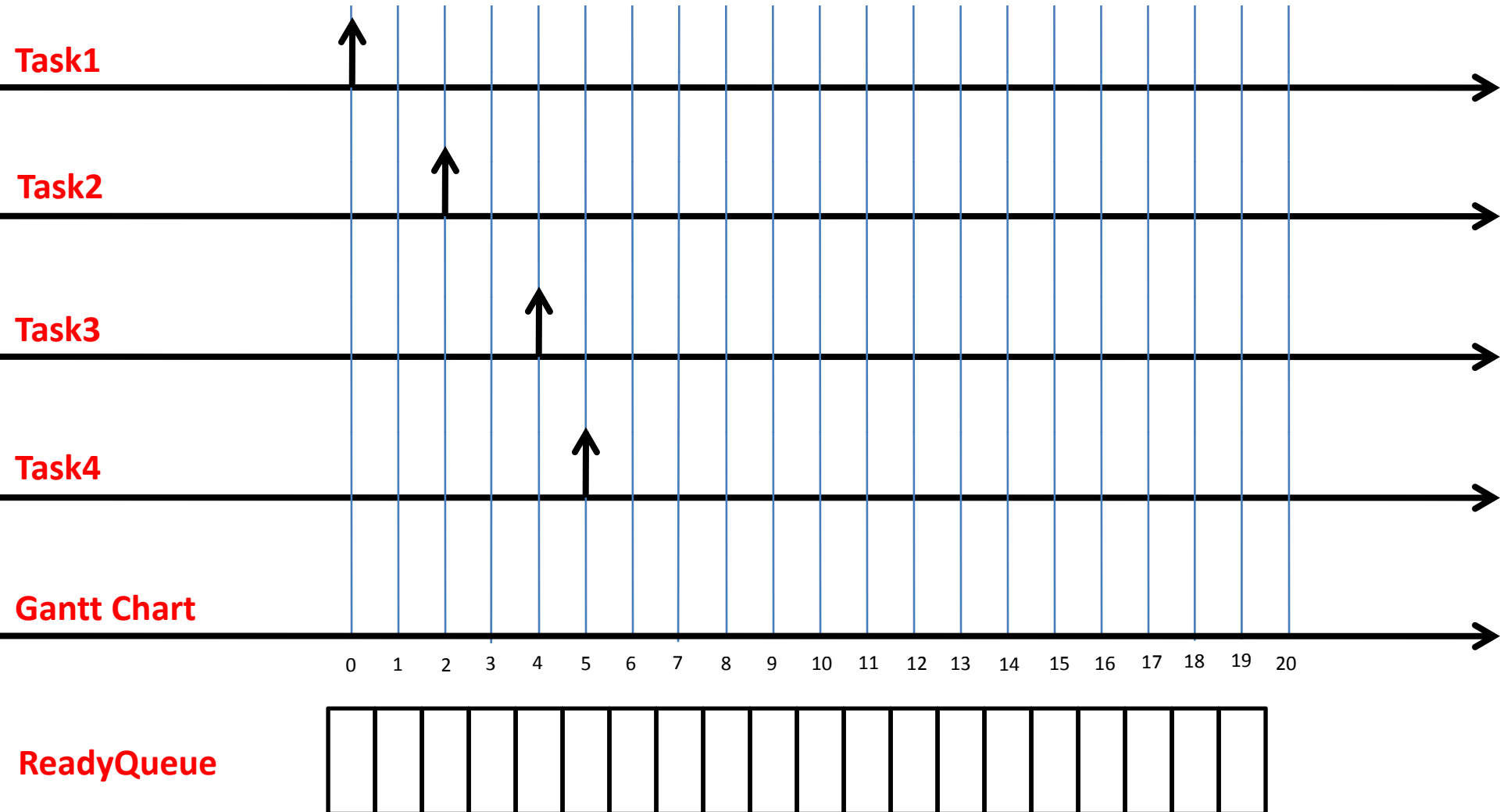
- Exemple : ordonnancer les 4 tâches suivantes par PB-Preemptive et calculer le AWT et le ATT.

Task	r0	C	O	Pr
Task1	0	7	1	3
Task2	2	4	2	2
Task3	4	1	3	4
Task4	5	4	4	1

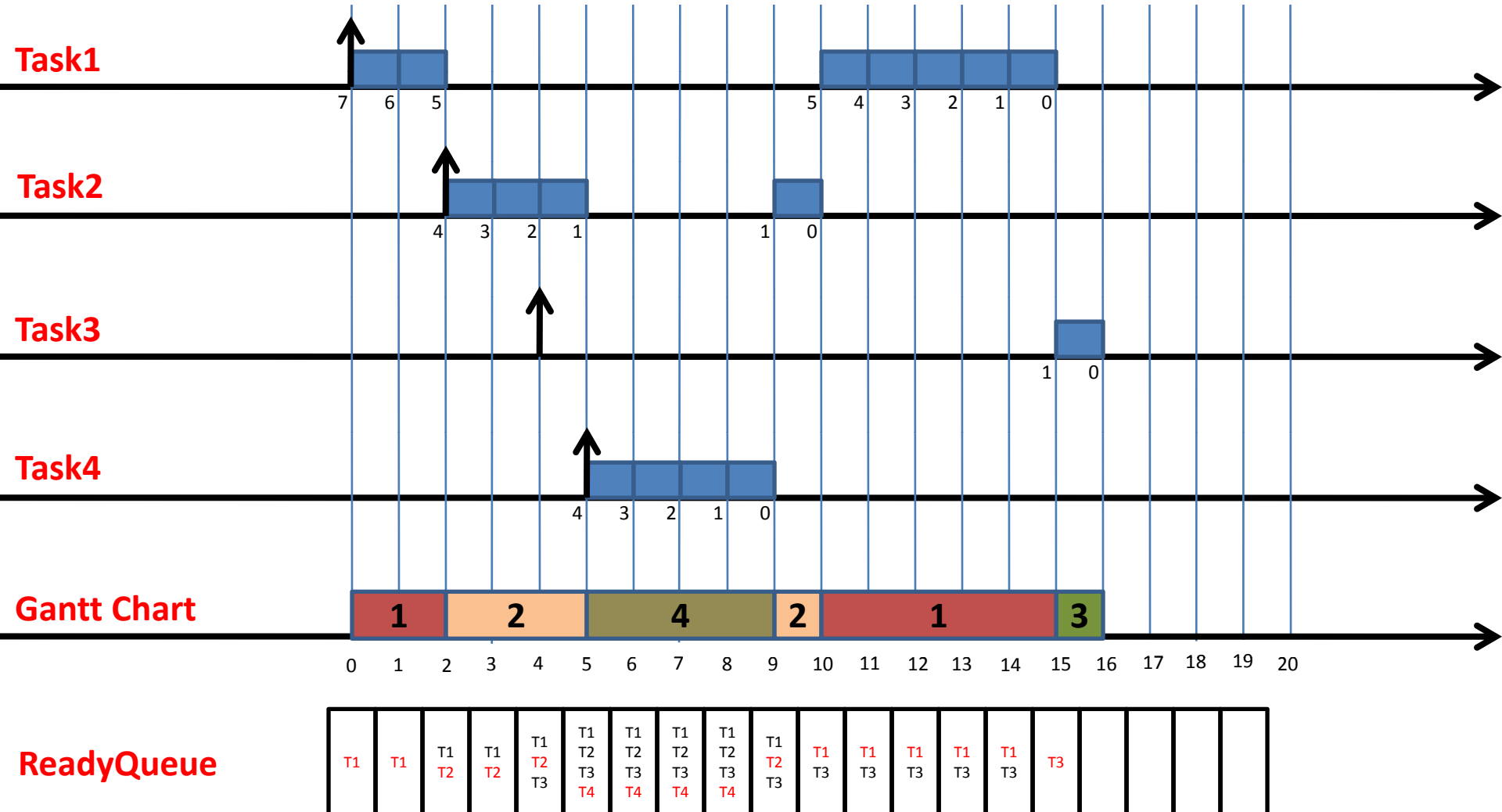
1 is a highest priority and 4 is a lowest priority

Priority Based (PB)

- Tracer les chronogrammes :



Priority Based (PB)



Priority Based (PB)

Task	r0	C	O	Pr	Completion Date (CD)	Turn Around Time (TAT) (CD - r0)	Waiting Time (WT) (TAT - C)
Task1	0	7	1	3	15	15-0 = 15	15-7 = 8
Task2	2	4	2	2	10	10-2 = 8	8-4 = 4
Task3	4	1	3	4	16	16-4 = 12	12-1 = 11
Task4	5	4	4	1	9	9-5 = 4	4-4 = 0

- $AWT = \text{mean}(WT_i) = \text{mean}(8, 4, 11, 0) = 5.75.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(15, 8, 12, 4) = 9.75.$

Politiques d'ordonnancement

ROUND ROBIN (RR)

Round Robin (RR)

- **Round-robin** (Tourniquet) est un algorithme d'ordonnancement adapté aux systèmes travaillant en temps partagés.
- Une petite unité de temps, appelé **time quantum** ou **time slice**, est définie. La file d'attente (Ready Queue) est gérée comme une file **circulaire**.
- L'ordonnanceur **parcourt** cette file et alloue un temps processeur à chacun des processus pour un intervalle de temps de l'ordre d'un quantum au maximum.
- La **performance** de round-robin dépend fortement du **choix du quantum** de base.

Round Robin (RR)

- Le RR est un algorithme d'ordonnancement **préemptif** se base sur le partage du temps entre tâches à travers une Ready Queue de type **FIFO** sans priorité associée aux tâches.
- Cet algorithme est inventé aussi pour lutter contre le phénomène de la **famine** (**STARVATION**).

Round Robin (RR)

- Le temps d'allocation du CPU au tâche choisi par RR est **préemptif**.
- Le modèle utilisé par RR est : **Task(r0, C, O)**.
 - **r0** : Arrival Date.
 - **C** : Burst Time (Capacity).
 - **O** : Order of task.
- Temps de charge du noyau : Négligeable.
- On n'a pas de délai critique ($D = \infty$).

Round Robin (RR)

- Etapes à suivre pour ordonnancer efficacement un ensemble de tâches par RR:
- La Ready Queue (**R.Q**) de RR est de type FIFO, l'**empilement** se fait sur **r0** et à **la fin de quantum** si la tâche n'est pas encore terminée. Le **dépilement** se fait par l'**ordonnanceur**.
- Si plus d'une tâche arrive au même temps, l'**empilement** dans le **R.Q** se fait par **ordre** de tâches.
- La valeur de **Time Quantum (T.Q)** est un **multiple de TICK système (T.S)**.

Round Robin (RR)

- **Etape 1** : de tracer les chronogrammes par :
 - De signaler sur long de chronogramme de chaque tâche : $\{r \uparrow\}$.
- D'ordonnancer les tâches selon l'algorithme dans le diapo suivant.
- De tracer le chrono **Gantt Chart (GC)**.

Round Robin (RR)

- Algorithme d'ordonnancement par RR :

A chaque T.Q faire

Si (R.Q n'est pas vide) alors

Dépiler la R.Q;

A chaque T.S faire

Dessiner un rectangle sur le chrono;

Diminuer C d'un T.S;

Diminuer T.Q d'un T.S;

Répéter si ($C \neq 0$ et $T.Q \neq 0$)

Si ($C \neq 0$) alors

Empiler la tache dépilée précédemment dans R.Q;

Fin si

Fin si

Répéter jusqu'à la fin de Burst Time de toutes les tâches

Round Robin (RR)

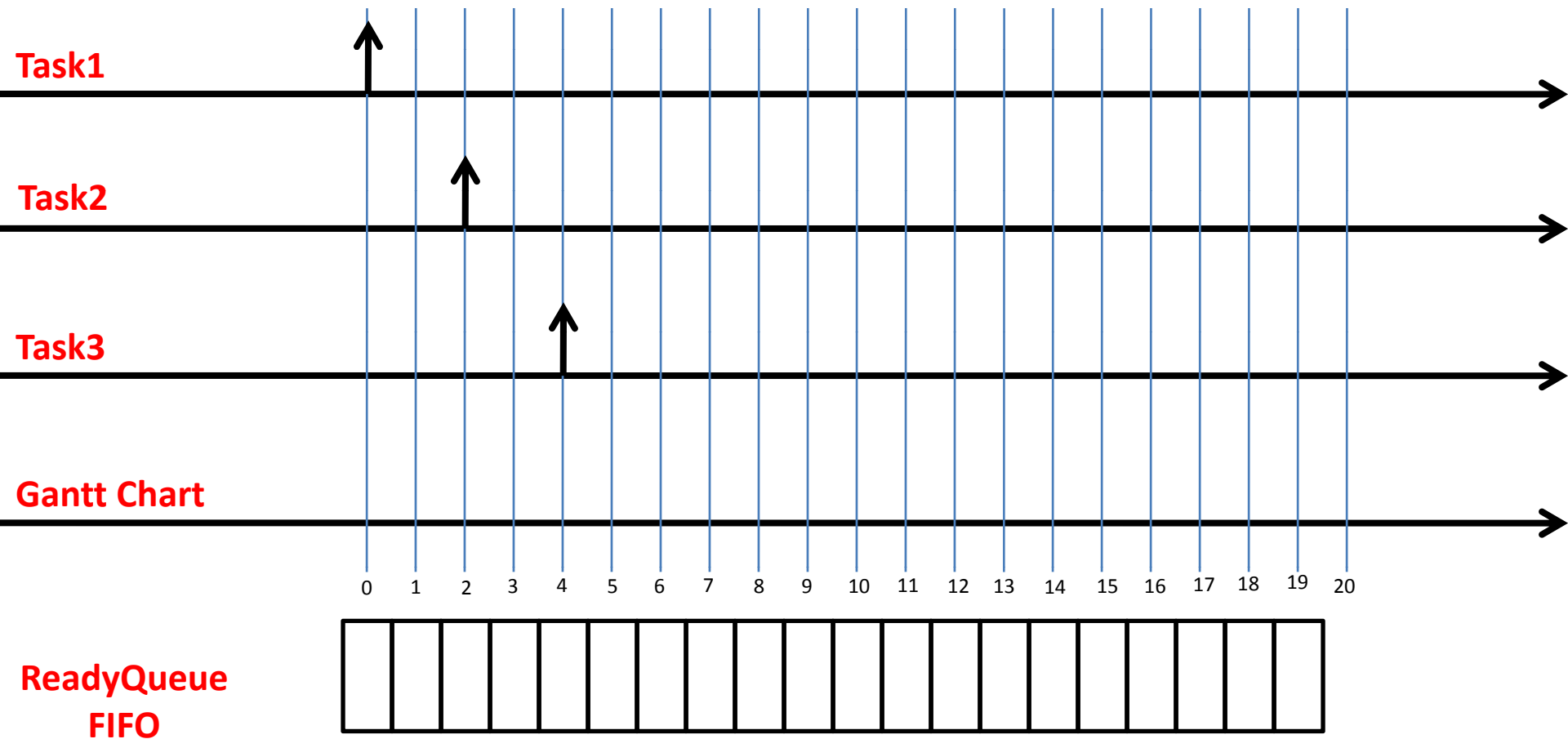
- Exemple : ordonnancer les 3 tâches suivantes par RR et calculer le AWT et le ATT.

Task	r0	C	O
Task1	0	7	1
Task2	2	4	2
Task3	4	1	3

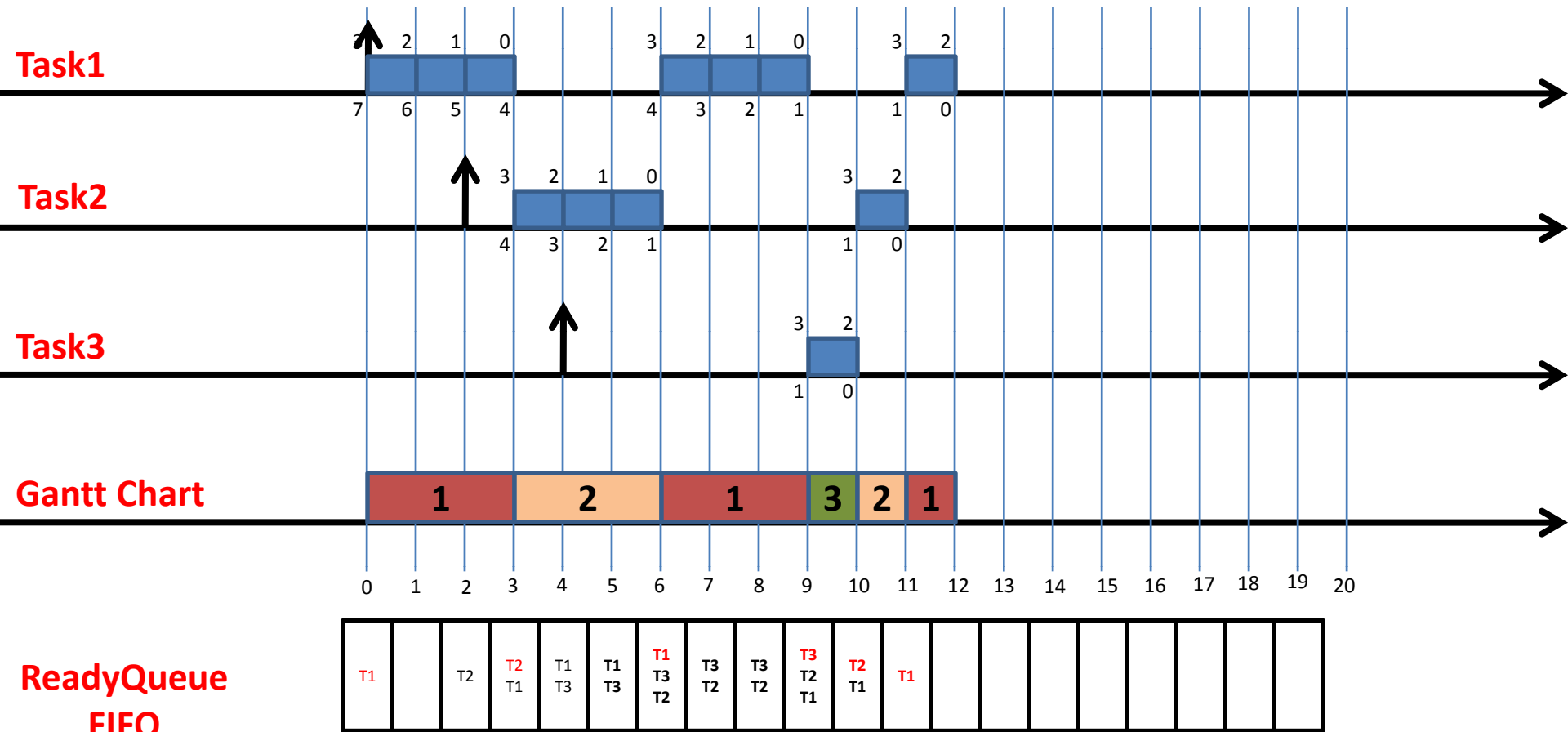
$$\text{T.Q} = 3 \text{ T.S}$$

Round Robin (RR)

- Tracer les chronogrammes :



Round Robin (RR)



Round Robin (RR)

Task	r0	C	O	Completion Date (CD)	Turn Around Time (TAT) (CD – r0)	Waiting Time (WT) (TAT - C)
Task1	0	7	1	12	$12 - 0 = 12$	$12 - 7 = 5$
Task2	2	4	2	11	$11 - 2 = 9$	$9 - 4 = 5$
Task3	4	1	3	10	$10 - 4 = 6$	$6 - 1 = 5$

- $AWT = \text{mean}(WT_i) = \text{mean}(5, 5, 5) = 5.$
- $ATT = \text{mean}(TAT_i) = \text{mean}(12, 9, 6) = 9.$