



# Lezione R7

## Schedulazione di job bloccanti e job aperiodici

Sistemi embedded e real-time

20 novembre 2012

Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica  
Università degli Studi di Roma Tor Vergata

Schema della lezione

Auto-sospensione

Non interrompibilità

Cambi di contesto

Test di schedulabilità

Schedulazione con tick

Job aperiodici

## Di cosa parliamo in questa lezione?



In questa lezione si continua a discutere di schedulazione priority-driven, con particolare riguardo alle condizioni di schedulabilità in presenza di job bloccanti

- 1 Blocchi dovuti ad auto-sospensione
- 2 Blocchi dovuti a non interrompibilità
- 3 Rallentamenti dei cambi di contesto
- 4 Test di schedulabilità con blocchi e rallentamenti
- 5 Scheduler basati su interruzioni periodiche
- 6 Schedulazione di job aperiodici

## Tempi di blocco e rallentamenti

Molti fattori di diversa natura contribuiscono a rallentare l'esecuzione di un job, potenzialmente provocando il mancato rispetto della sua scadenza

Due grandi classi di ritardi:

- I *tempi di blocco*, in cui il job pur essendo stato rilasciato non può essere eseguito per qualche motivo esterno

Ad esempio, un *blocco per non interrompibilità*, oppure l'esecuzione di una operazione che provoca una sospensione del job

Il *tempo massimo di blocco*  $b_i$  è la lunghezza massima dell'intervallo in cui un job di  $T_i$  può essere bloccato

- I *rallentamenti* sistematici che si sommano al tempo di esecuzione del job

Un esempio è il tempo richiesto per eseguire lo scheduler e per effettuare il cambio di contesto tra un job e l'altro



## Auto-sospensione

Spesso un job già rilasciato non può essere eseguito perché in attesa di eventi esterni: è sospeso e sostituito sul processore da un altro job (*auto-sospensione*)

Esempi di **operazioni bloccanti** che **auto-sospendono**:

- accesso al disco rigido
- attesa di dati da rete o da altro job
- attesa della scadenza di un timer

*Supponiamo che ogni job di un task  $T_i$  si auto-sospende per un tempo  $x$  non appena è rilasciato (ad es., attende dati di input). Come determinare se il job è schedulabile?*

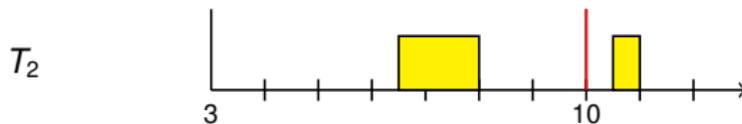
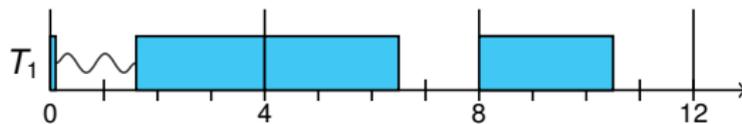
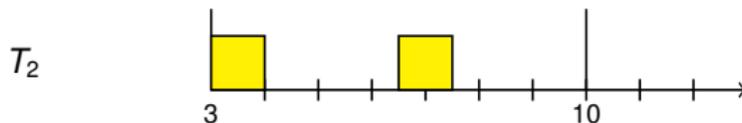
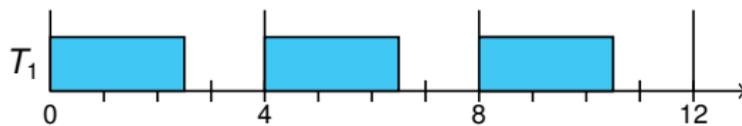
È sufficiente considerare come istante di rilascio  $p_i + x$  e come scadenza relativa  $D_i - x$

Se tutti i job possono variare quando e per quanto tempo si auto-sospendono, per ciascun task  $T_i$  si deve determinare il **tempo massimo di blocco per l'auto-sospensione**  $b_i(ss)$



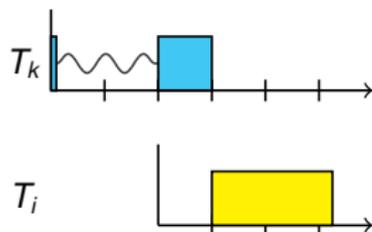
# Esempio di auto-sospensione (RM)

$$T_1 = (4, 2.5) \quad T_2 = (3, 7, 2, 7)$$



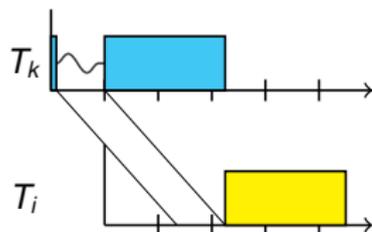
## Rallentamento dovuto all'auto-sospensione

Caso 1: il tempo di auto-sospensione di un job è maggiore della durata del job



Un job di  $T_i$  con priorità inferiore è rallentato al massimo per un tempo pari alla durata del job di  $T_k$

Caso 2: il tempo di auto-sospensione di un job è minore della durata del job



Un job di  $T_i$  con priorità inferiore è rallentato al massimo per un tempo pari alla durata dell'auto-sospensione



## Tempo massimo di blocco per auto-sospensione

Dato un task  $T_k$ , sia  $x_k$  il tempo massimo di auto-sospensione di ogni job di  $T_k$  (è un parametro noto)

Dato un task  $T_i$  ed un task di priorità maggiore  $T_k$ , il rallentamento inflitto ad un job di  $T_i$  da un job di  $T_k$  è minore o uguale a  $x_k$  e minore o uguale a  $e_k$

Di conseguenza, 
$$b_i(ss) = x_i + \sum_{k=1}^{i-1} \min(e_k, x_k)$$

*Il valore di  $b_i(ss)$  definisce in modo completo il rallentamento dovuto all'auto-sospensione per il task  $T_i$ ? **No!***

Anche il numero di volte massimo  $K_i$  in cui un job di  $T_i$  si auto-sospende è importante

Infatti per ogni sospensione e successiva riattivazione:

- è possibile che si verifichi un blocco da parte di un processo non interrompibile
- si ha un rallentamento dovuto allo scheduler ed al costo del cambio di contesto



## Non interrompibilità dei job

Per tutti i teoremi di schedulabilità ogni job è interrompibile nell'istante in cui un job di priorità maggiore è rilasciato

In pratica questa assunzione è irrealistica: esistono sempre istanti in cui un job **non** è interrompibile, ad esempio quando:

- il job utilizza una risorsa critica condivisa
- il job interagisce con un dispositivo hardware
- il job esegue una chiamata di sistema che, in quel momento, non è interrompibile
- il costo dell'interruzione è troppo elevato

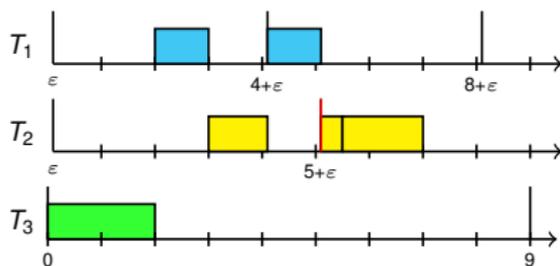
Un job  $J_i$  è *bloccato per non interrompibilità* quando è pronto all'esecuzione ma non può essere eseguito a causa di un job di priorità minore che non può interrompere l'esecuzione

Si dice che in un intervallo di tempo si verifica *inversione di priorità* se nell'intervallo viene eseguito un job di priorità minore di quella di un altro job pronto per l'esecuzione



## Esempio di non interrompibilità

- Consideriamo un sistema di tre task  $T_1=(\epsilon, 4, 1, 4)$ ,  $T_2=(\epsilon, 5, 1.5, 5)$ ,  $T_3=(9, 2)$  (con  $0 < \epsilon < 0.5$ )
- L'utilizzazione totale è  $U = 1/4 + 1.5/5 + 2/9 = 0.77$ , quindi è schedulabile sia con **EDF** che con **RM** ( $U_{RM}(3)=0.779$ ) *se tutti i job sono sempre interrompibili*
- Supponiamo che  $T_3$  sia **non interrompibile**:  $T_3$  ha fase  $0 < \epsilon$ , quindi  $J_{3,1}$  esegue nell'intervallo  $[0, 2]$
- In  $[\epsilon, 2]$   $J_{1,1}$  e  $J_{2,1}$  sono **bloccati** da  $J_{3,1}$ : **inversione di priorità**
- Nell'intervallo  $[2, 5+\epsilon]$  sono eseguiti  $J_{1,1}$ ,  $J_{2,1}$  e  $J_{1,2}$ , ma  $5+\epsilon-2 < 1.5+1+1$ :  $T_2$  manca la scadenza



## Blocco dovuto a non interrompibilità

Sia  $\theta_k$  il tempo d'esecuzione massimo della più lunga sezione non interrompibile dei job di  $T_k$

Sia  $b_i(np)$  il *tempo massimo di blocco per non interrompibilità* che un job di  $T_i$  può subire nel momento del suo rilascio

*Quanto vale  $b_i(np)$ ?*

$$b_i(np) = \max \{ \theta_k : \text{per ogni task } T_k \text{ di priorità minore di } T_i \}$$

*Il tempo massimo di blocco  $b_i$  dipende sia da  $b_i(np)$  che da  $b_i(ss)$ . Qual è la formula per RM/DM?*

$$b_i = b_i(ss) + (K_i + 1) \cdot b_i(np)$$

Oltre che in occasione del primo rilascio, il job può essere **bloccato per non interrompibilità** ad ogni attivazione seguente ad una auto-sospensione



## Cambi di contesto

L'overhead dovuto ai cambi di contesto è un rallentamento subito uniformemente da tutti i job in occasione di ogni attivazione

Sia  $CS$  il costo di un cambio di contesto tra due job, incluso il tempo necessario all'esecuzione dello scheduler

I test di schedulabilità possono essere applicati semplicemente includendo nei tempi di esecuzione dei task i costi dovuti ai cambi di contesto:

$$e'_i = e_i + 2 \cdot (K_i + 1) \cdot CS$$

*Quale algoritmo di scheduling è particolarmente inefficiente se  $CS$  è significativamente grande? **LST***

In una schedulazione **LST** vi è un gran numero di cambi di contesto, quindi un overhead significativo

Inoltre non è facile determinare il numero massimo di cambi di contesto di ciascun job, quindi è difficile validare il sistema



## Test di schedulabilità per job bloccanti



*Come estendere il test di schedulabilità per trattare i job che possono bloccare?*

Al tempo disponibile per l'esecuzione di ciascun job va sottratto il tempo massimo in cui il job può restare bloccato

Idea: il tempo a disposizione di un job per terminare l'esecuzione deve essere ridotto del **tempo massimo di blocco**

Perciò la **funzione di tempo richiesto** diventa

$$w_i(t) = e_i + b_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \quad \text{per } 0 < t \leq \min(D_i, p_i)$$

Analogamente per il test di schedulabilità generale:

$$w_{i,j}(t) = j e_i + b_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k \quad \text{per } (j-1)p_i < t \leq w_{i,j}(t)$$

## Condizioni di schedulabilità per task bloccanti a priorità fissa

Sia dato un sistema di  $n$  task  $\mathcal{T}$  ed un algoritmo  $X$  a priorità fissa con fattore di utilizzazione  $U_X(n)$

Sappiamo che il sistema è effettivamente schedulabile se  $U_{\mathcal{T}} \leq U_X(n)$ , a condizione che i task non blocchino mai

*Come si adatta la condizione di schedulabilità per task a priorità fissa bloccanti?*

- Ciascun job può bloccare in misura differente: applichiamo la condizione di schedulabilità un task alla volta
- Nel caso peggiore ogni job di  $T_i$  impiega tempo  $e_i + b_i$  per completare l'esecuzione
- Quindi  $T_i$  è schedulabile se

$$\sum_{k=1}^i \frac{e_k}{p_k} + \frac{b_i}{p_i} \leq U_X(i)$$





## Condizione di schedulabilità EDF per job bloccanti

La condizione di schedulabilità EDF in presenza di blocchi è analoga a quella degli algoritmi a priorità fissata

- Si applica su ciascun task singolarmente
- Il task  $T_i$  è schedulabile tramite EDF se

$$\sum_{k=1}^n \frac{e_k}{\min(D_k, p_k)} + \frac{b_i}{\min(D_i, p_i)} = \Delta_{\mathcal{T}} + \frac{b_i}{\min(D_i, p_i)} \leq 1$$

*Qual è la difficoltà? Cosa ci manca per applicare la formula?*

Il problema è come definire i tempi massimi di blocco  $b_i$ : non esiste più l'insieme dei job con priorità minore di  $T_i$

### Teorema (Baker 1991)

In una schedulazione EDF, un job con scadenza relativa  $D$  può bloccare un altro job con scadenza relativa  $D'$  solo se  $D > D'$

Dim.:  $d > d', r < r' \Rightarrow D = d - r > d' - r' = D'$

Soluzione: ordinare i task per scadenze relative crescenti, ed applicare la formula di  $b_i$  trovata per i task con priorità fissata

## Schedulazione basata su tick

Test e condizioni di schedulabilità assumono che lo scheduler è *event-driven*: viene eseguito quando si verifica un evento rilevante (un job viene rilasciato, si auto-sospende o termina)

In pratica, è più semplice realizzare uno scheduler *time-driven* che si attiva all'occorrenza di interruzioni periodiche (*tick*)

- Il riconoscimento di un evento come il rilascio di un job potrebbe essere differito fino al tick successivo
- Si distinguono due tipi di job rilasciati: quelli *pendenti* non ancora riconosciuti dallo scheduler e quelli *eseguibili*
- Esiste una coda di job *pendenti* ed una per i job *eseguibili*
- Lo scheduler sposta job dalla coda dei *pendenti* in quella degli *eseguibili* (nella posizione appropriata)
- Quando un job termina o sospende l'esecuzione, viene eseguito subito il prossimo job *eseguibile* senza invocare lo scheduler



## Test di schedulabilità per priorità fissata con tick

*Come è possibile applicare il test di schedulabilità ad uno scheduler a priorità fissata basato su tick?*



Consideriamo uno scheduler che si attiva con periodicità  $p_0$ , esegue in tempo  $e_0$  il controllo della coda di job pendenti, e trasforma un job da pendente ad eseguibile in tempo  $CS_0$

Schema della lezione

Auto-sospensione

Non interrompibilità

Cambi di contesto

Test di schedulabilità

Schedulazione con tick

Job aperiodici

Per controllare la schedulabilità di un task  $T_i$ :

- aggiungere un task  $T_0=(p_0, e_0)$  di priorità massima
- aggiungere un task  $T_{0,k}=(p_k, CS_0)$  con priorità maggiore di  $T_1$  per ogni  $k = i+1, \dots, n$
- aggiungere  $(K_k + 1) \cdot CS_0$  al tempo d'esecuzione  $e_k$  di ogni task  $T_k$ , per  $k = 1, 2, \dots, i$

- utilizzare: 
$$b_i(np) = \left( \left[ \max_{i+1 \leq k \leq n} \frac{\theta_k}{p_0} \right] + 1 \right) \cdot p_0$$

## Esempio di test di schedulabilità con tick

$T_1=(0.1, 4, 1, 4)$ ,  $T_2=(0.1, 5, 1.8, 5)$ ,  $T_3=(20, 5)$  non interromp. in  $[r_3, r_3+1.1)$ . Scheduler:  $p_0 = 1$ ,  $e_0 = 0.05$ ,  $CS_0 = 0.06$

Verifica di  $T_1$  Sistema equivalente:  $T_0=(1, 0.05)$ ,  $T_{0,2}=(5, 0.06)$ ,  $T_{0,3}=(20, 0.06)$ ,  $T_1=(4, 1.06)$ ,  $b_1 = 3$ :

$$w_1(t) = 1.06 + 3 + \lceil t/1 \rceil 0.05 + \lceil t/5 \rceil 0.06 + \lceil t/20 \rceil 0.06$$

$$w_1(4.06) = 4.43 = w_1(4.43) > 4 \quad \Rightarrow \text{no}$$

Verifica di  $T_2$  Sistema equivalente:  $T_0=(1, 0.05)$ ,  $T_{0,3}=(20, 0.06)$ ,  $T_1=(4, 1.06)$ ,  $T_2=(5, 1.86)$ ,  $b_2 = 3$ :

$$w_2(t) = 1.86 + 3 + \lceil t/1 \rceil 0.05 + \lceil t/20 \rceil 0.06 + \lceil t/4 \rceil 1.06$$

$$w_2(4.86) = 7.29, w_2(7.29) = 7.44 = w_2(7.44) > 5 \quad \Rightarrow \text{no}$$

Verifica di  $T_3$  Sistema equivalente:  $T_0=(1, 0.05)$ ,  $T_1=(4, 1.06)$ ,  $T_2=(5, 1.86)$ ,  $T_3=(20, 5.06)$ ,  $b_3 = 1$ :

$$w_3(t) = 5.06 + 1 + \lceil t/1 \rceil 0.05 + \lceil t/4 \rceil 1.06 + \lceil t/5 \rceil 1.86$$

$$w_3(6.06) = 12.25, w_3(12.25) = 16.53, w_3(16.53) = 19.65, \\ w_3(19.65) = 19.8 = w_3(19.8) \leq 20 \quad \Rightarrow \text{ok}$$





Schema della lezione

Auto-sospensione

Non interrompibilità

Cambi di contesto

Test di schedulabilità

Schedulazione con tick

Job aperiodici

## Condizione di schedulabilità con tick

Metodo per applicare una condizione di schedulabilità ad uno scheduler a priorità dinamica basato su tick

Per ciascun  $T_i$  da controllare (task ordinati per  $D$  crescenti):

- aggiungere un task  $T_0=(p_0, e_0)$  di priorità massima
- aggiungere  $(K_k + 1) \cdot CS_0$  al tempo d'esecuzione  $e_k$  di ogni task  $T_k$ , per  $k = 1, 2, \dots, n$
- utilizzare:  $b_i(np) = \left( \left[ \max_{i+1 \leq k \leq n} \frac{\theta_k}{p_0} \right] + 1 \right) \cdot p_0$

Nell'esempio precedente, il sistema equivalente è

$$T_0=(1, 0.05), T_1=(4, 1.06), T_2=(5, 1.86), T_3=(20, 5.06), \\ b_1 = b_2 = 3, b_3 = 1$$

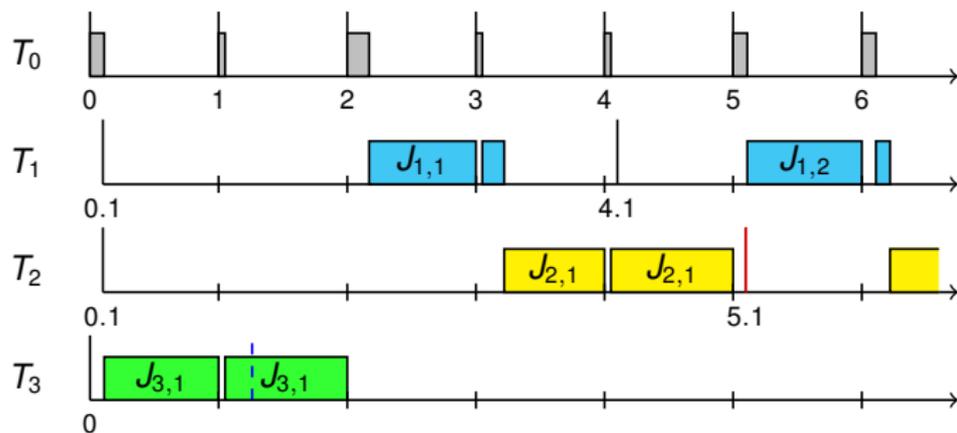
$$\text{Fattore di utilizzazione } U = \frac{0.05}{1} + \frac{1.06}{4} + \frac{1.86}{5} + \frac{5.06}{20} = 0.94$$

$$\text{Verifica di } T_1: U + 3/4 = 1.69 > 1 \quad \Rightarrow \text{no}$$

$$\text{Verifica di } T_2: U + 3/5 = 1.54 > 1 \quad \Rightarrow \text{no}$$

$$\text{Verifica di } T_3: U + 1/20 = 0.99 \leq 1 \quad \Rightarrow \text{ok per EDF}$$

## Esempio di schedulazione con tick



L'algorithmo di schedulazione potrebbe essere sia EDF che RM

In questo frammento  $T_1$  rispetta la scadenza anche se falliscono sia il test che la condizione di schedulabilità



## Schedulazione priority-driven di job aperiodici

Nei sistemi real-time basati su schedulazione priority-driven è spesso necessario eseguire oltre ai task periodici:

- Job **aperiodici soft RT**: con tempi di arrivo e di esecuzione sconosciuti, con scadenze “soft” o senza scadenze, ma comunque da completare nel più breve tempo possibile
- Job **aperiodici hard RT**: con tempi di arrivo sconosciuti, e con tempi di esecuzione e scadenze “hard” noti solo dopo il rilascio

Le due classi di job richiedono algoritmi differenti

Ogni algoritmo utilizzato deve essere corretto e ottimale:

- le scadenze dei task periodici devono essere rispettate
- i job aperiodici hard RT devono essere rifiutati se non è possibile garantire le loro scadenze
- le scadenze dei job aperiodici hard RT accettati devono essere rispettate
- i tempi di risposta dei job aperiodici soft RT devono essere minimizzati (singolarmente o mediamente)



## Schedulazione di job aperiodici soft RT in background

La *schedulazione in background* è l'algoritmo più semplice per i job aperiodici soft real-time

Una *coda* memorizza i job aperiodici che sono stati rilasciati

Il job aperiodico in testa alla coda viene eseguito durante gli intervalli di tempo in cui la schedulazione priority-driven dei task periodici lascia il processore idle

*L'algoritmo è corretto e ottimale?*

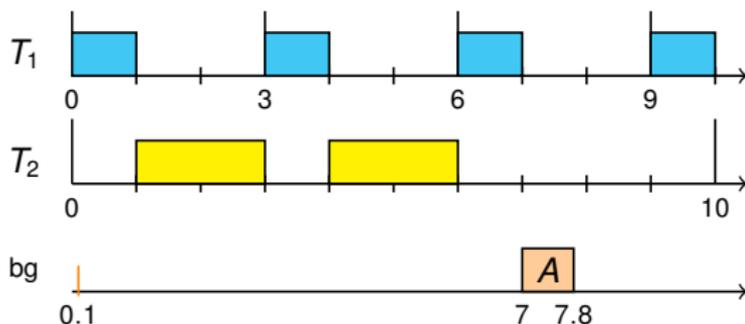
È **corretto**: i task periodici non sono influenzati

È **non ottimale**: i job aperiodici sono ritardati senza motivo

$$T_1 = (3, 1)$$

$$T_2 = (10, 4)$$

Job aper. A:  
rilascio a 0.1  
durata 0.8



## Schedulazione di job aperiodici soft RT interrupt-driven

L'algoritmo di *schedulazione interrupt-driven* impone l'esecuzione dei job aperiodici non appena vengono rilasciati

Ossia: i job aperiodici hanno sempre priorità massima

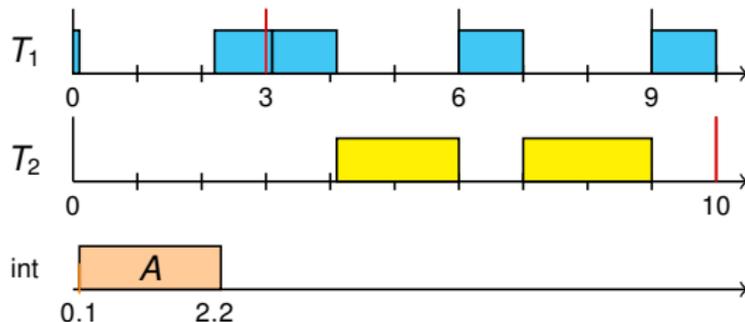
*L'algoritmo è corretto e ottimale?*

È **ottimale** per i job aperiodici: hanno tempi di risposta minimi  
È **non corretto**: i task periodici possono mancare le scadenze

$$T_1 = (3, 1)$$

$$T_2 = (10, 4)$$

Job aper. A:  
rilascio a 0.1  
durata 2.1



## Schedulazione di job aperiodici soft RT con slack stealing

L'algoritmo di *schedulazione con slack stealing* esegue i job aperiodici in anticipo rispetto ai task periodici finché il sistema ha **globalmente** slack positivo

*L'algoritmo è corretto e ottimale?*

È **corretto**: i task periodici non mancano le scadenze

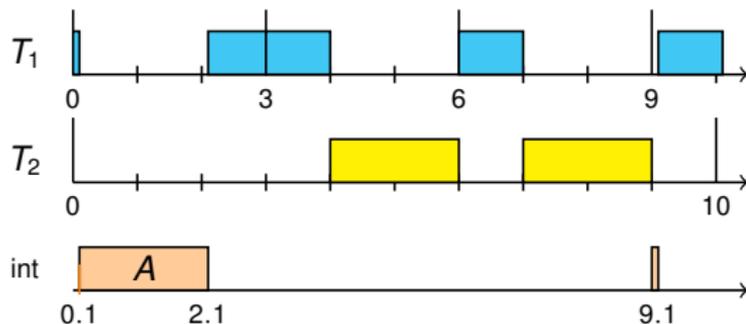
È **ottimale**, ma solo per il job aperiodico in cima alla coda

Il grande svantaggio di questo algoritmo è la difficoltà di implementazione in scheduler priority-driven

$T_1 = (3, 1)$

$T_2 = (10, 4)$

Job aper. A:  
rilascio a 0.1  
durata 2.1



## Schedulazione di job aperiodici soft RT con polling

L'algoritmo di *schedulazione con polling* è basato su un task periodico (*server di polling* o *poller*) con fase 0, periodo  $p_s$ , tempo d'esecuzione  $e_s$ , e priorità massima

Il *server di polling* controlla la *coda di job aperiodici*: se è vuota, si auto-sospende fino al prossimo periodo, altrimenti esegue il job in cima alla coda per max  $e_s$  unità di tempo

*L'algoritmo è corretto e ottimale?*

La *correttezza* dipende dai parametri del *poller*

È *non ottimale* (il job aperiodico può arrivare subito dopo l'inizio del periodo del *poller*)

$$T_P = (2.5, 0.5)$$

$$T_1 = (3, 1)$$

$$T_2 = (10, 4)$$

Job aper. A:  
rilascio a 0.1  
durata 0.8

