



Università degli Studi di Parma

Dipartimento di Ingegneria e Architettura

Sistemi operativi e in tempo reale - a.a. 2023/24

Problemi e tipologie di scheduling in tempo reale

prof. Stefano Caselli

stefano.caselli@unipr.it



Problemi nella schedulazione real-time

- ❑ Esempi che testimoniano che elaborazione *in tempo reale* non equivale ad elaborazione *veloce*
- ❑ Situazioni in cui modifiche ad un problema che dovrebbero facilitarne la soluzione producono invece tempi di risposta maggiori → *anomalie*
- ❑ Analisi di sequenze di *esecuzione in sistemi multiprocessore* e problemi correlati

Scenario

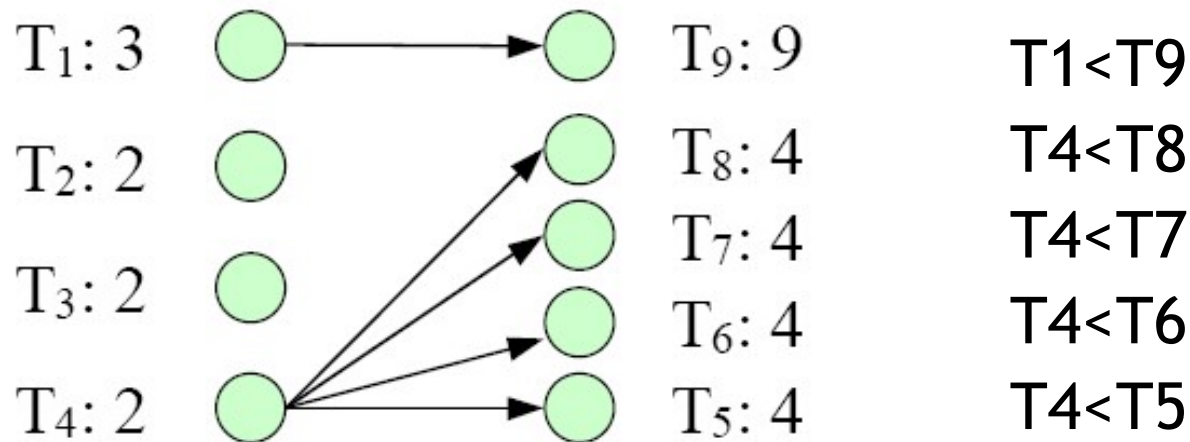


- ❑ Insieme di task con vincoli di precedenza, organizzati a grafo
- ❑ In questo caso: task aperiodici in esecuzione *one-shot*
- ❑ Per un insieme di *task aperiodici correlati*, una metrica spesso utilizzata di valutazione complessiva è il tempo di completamento dell'insieme di task, ovvero il *makespan*
 - Il makespan è il tempo che intercorre dal rilascio del primo task al completamento dell'ultimo
- ❑ Confronteremo il makespan con un valore *deadline*



Esempio

- Insieme di task: $T = \{T_1, \dots, T_9\}$
- Priorità: $\text{Pri}(T_i) > \text{Pri}(T_j)$ per $\forall i < j$
- Vincoli di precedenza e tempi di esecuzione:

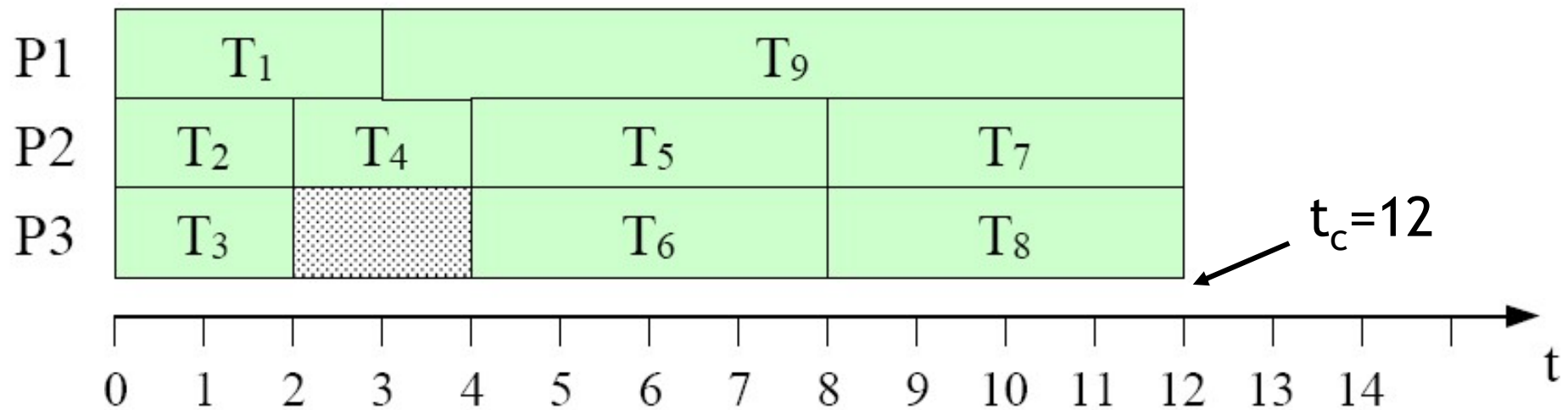
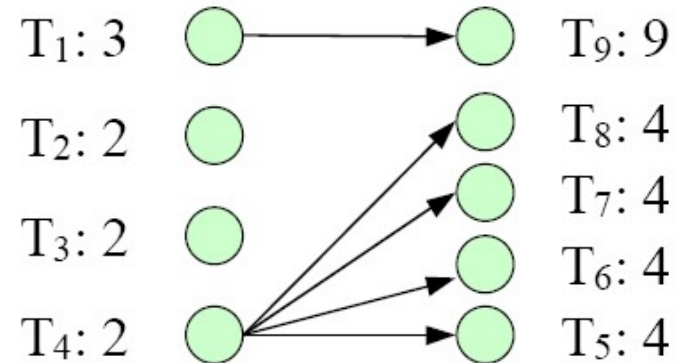


- Sistema di elaborazione: k processori P_1, \dots, P_k

Anomalie di scheduling



- Esecuzione su $k=3$ processori:



[Esercizio]



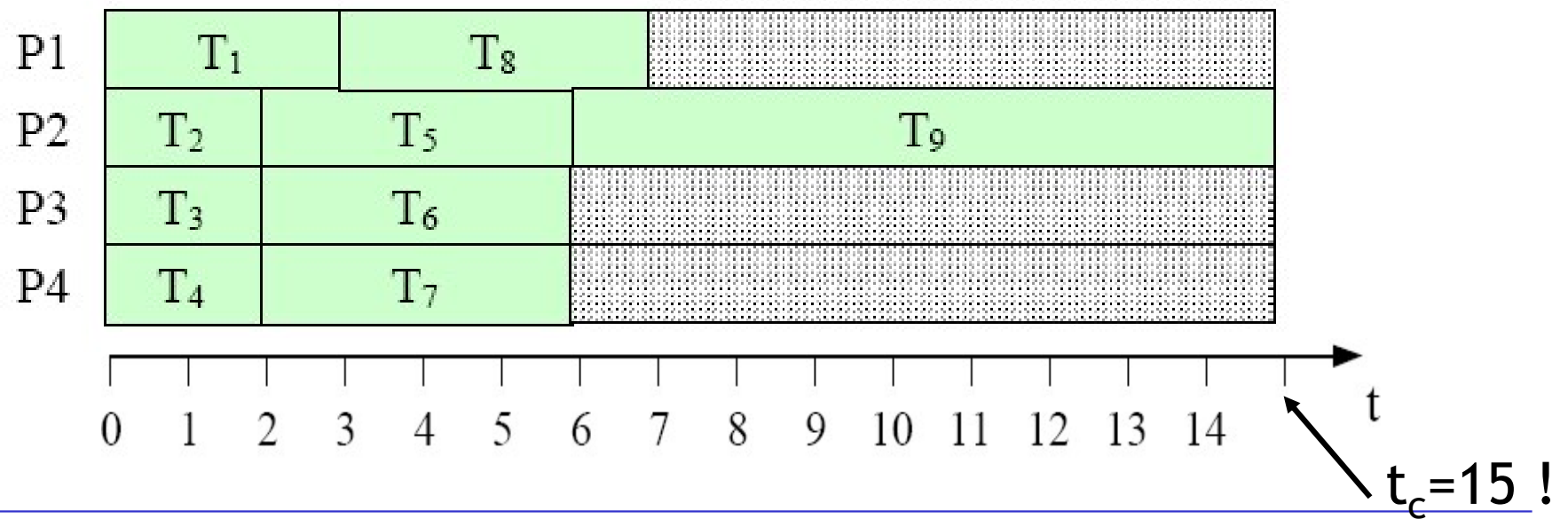
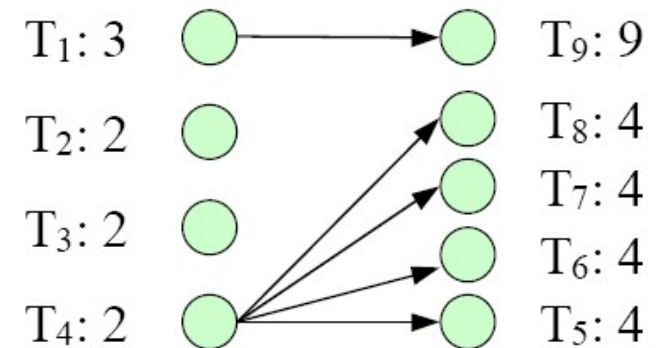
- ❑ Costruire la schedule per le stesse condizioni del caso precedente nell'ipotesi in cui $Pri(T_i) < Pri(T_j)$ per $\forall i < j$
- ❑ Quanto vale il *makespan*?

- ❑ La *priorità* attribuita influenza le prestazioni!
- ❑ E' un parametro libero o vincolato?

Anomalie di scheduling



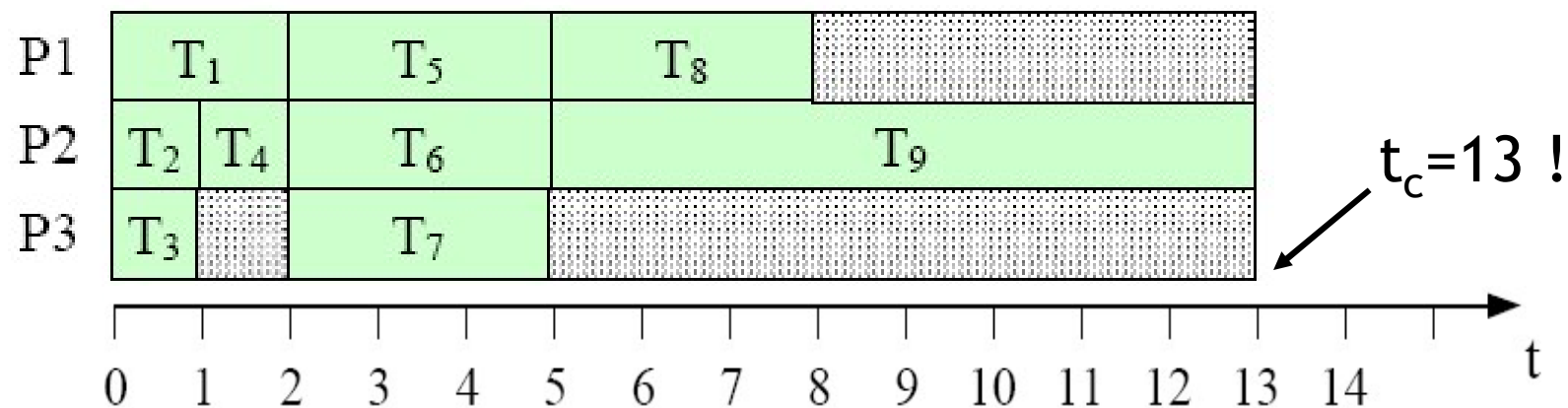
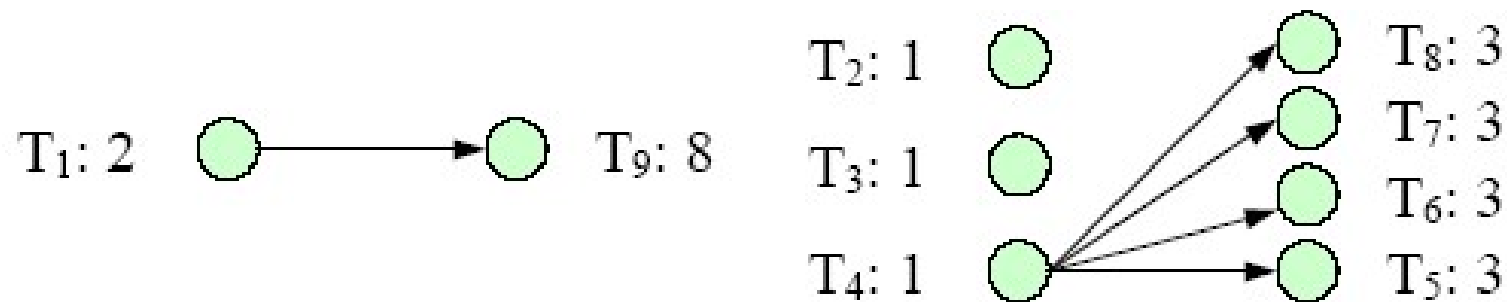
- Esecuzione su $K=4$ processori:





Anomalie di scheduling










- Esecuzione di task *più brevi*, $C_i \rightarrow C_i - 1 \ \forall i$ (ad es. per ottimizzazione del codice):

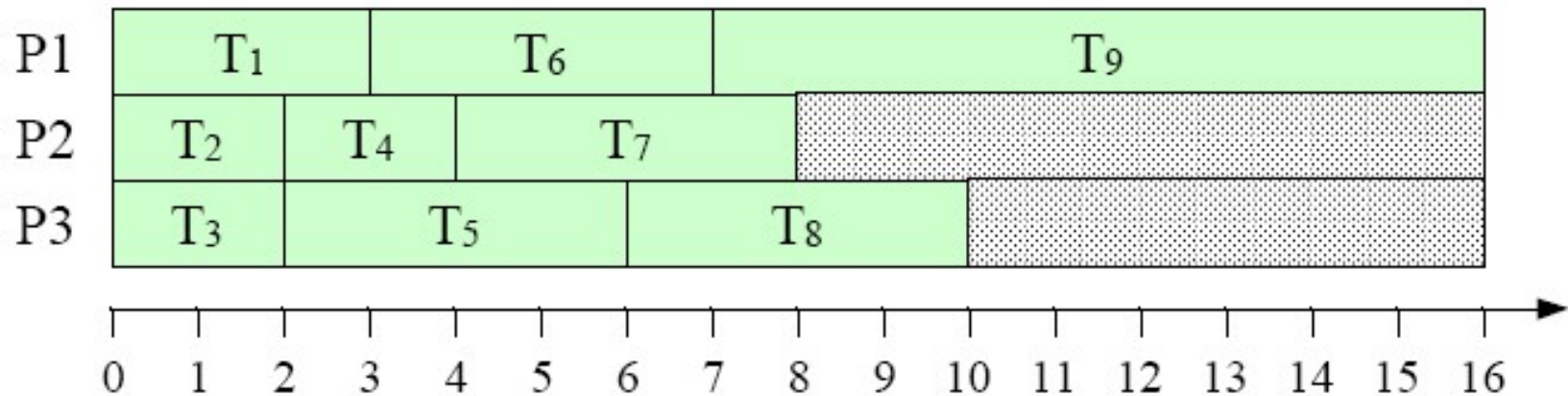




Anomalie di scheduling

□ Rimozione dei *vincoli di precedenza*:

$T_1: 3$		$T_4: 2$		$T_7: 4$	
$T_2: 2$		$T_5: 4$		$T_8: 4$	
$T_3: 2$		$T_6: 4$		$T_9: 9$	

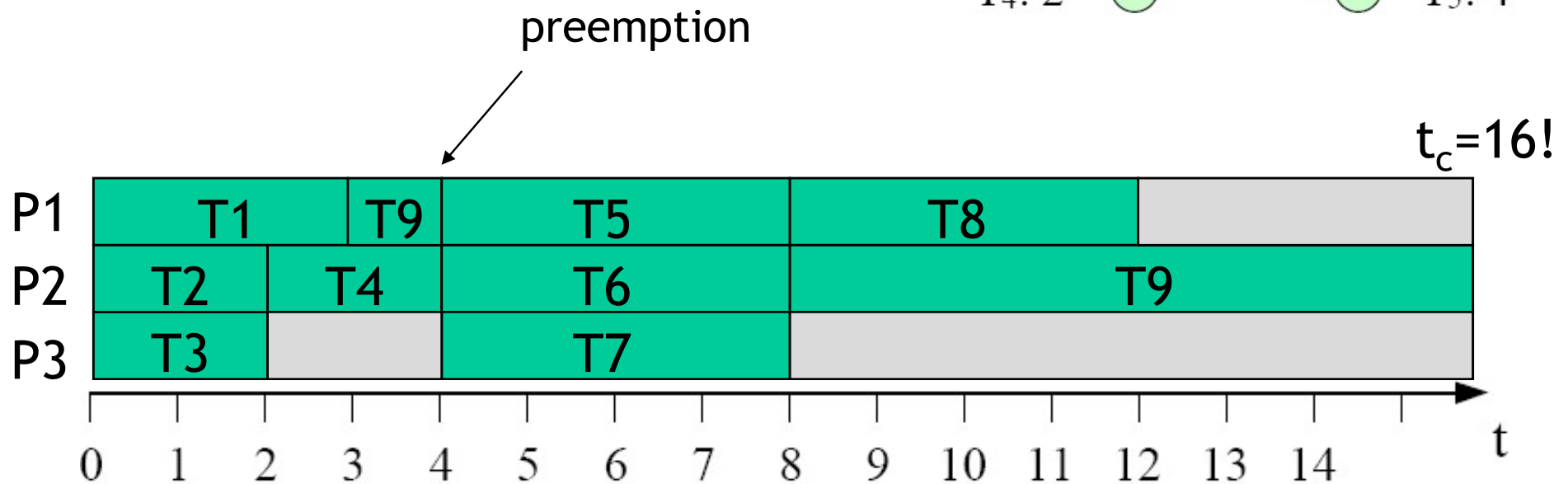
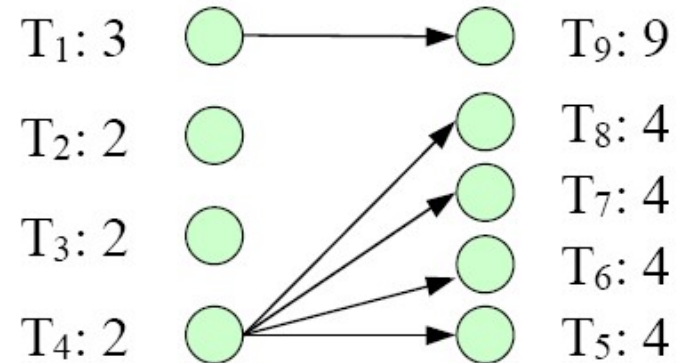


$t_c=16!$

Anomalie di scheduling



□ Introduzione di *preemption*:



Anomalie di scheduling



- ❑ *Anomalie di Richard*
- ❑ Teorema (Graham, 1969):

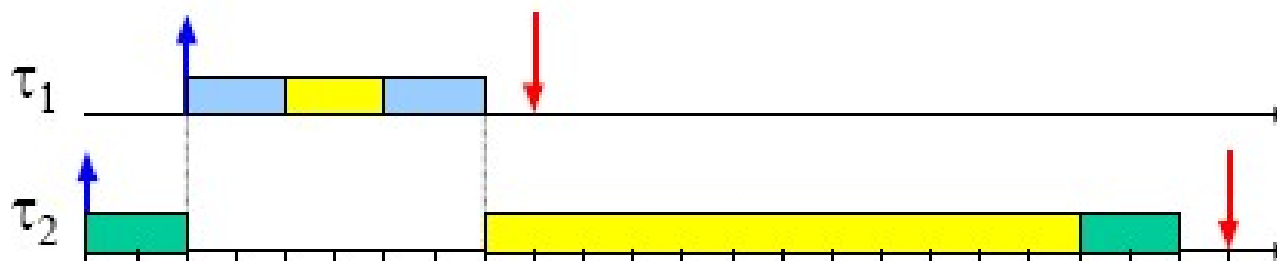
Per un task set schedulato in modo ottimale su un sistema multiprocessore con assegnamento di priorità, numero di processori, tempi di esecuzione e vincoli di precedenza fissati, l'aumento del numero di processori, la riduzione dei tempi di esecuzione ed il rilassamento dei vincoli di precedenza possono aumentare la lunghezza della schedule.

- ❑ ➔ modificando anche lievemente il problema in modo da rendere *più semplice* il rispetto delle scadenze, il tempo di risposta può divenire più elevato

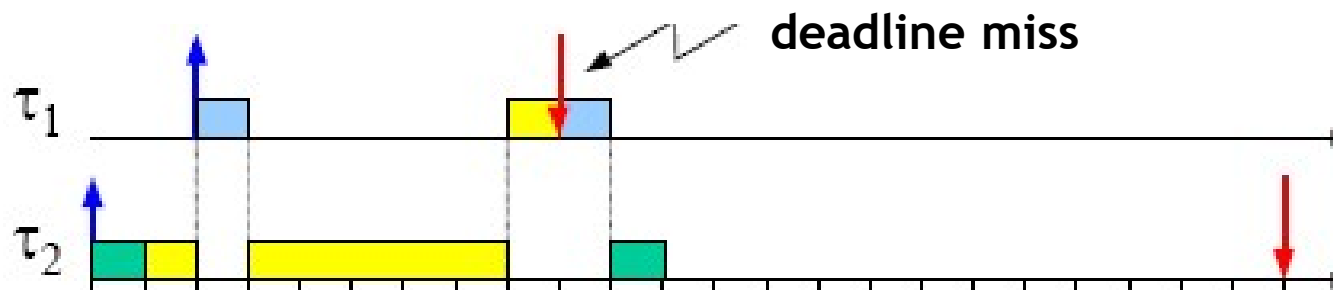
Anomalie di scheduling in presenza di vincoli su risorse



- Presenza di sezioni critiche: 



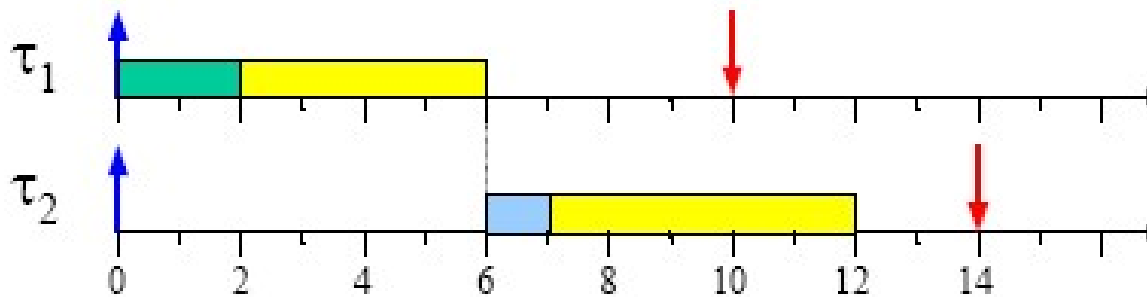
- Con velocità processore 2x:



Anomalie di scheduling in presenza di vincoli su risorse



- Introduzione di un ritardo D:



- Un ritardo D (es. sleep D) prima della sezione critica da parte di τ_1 può determinare un ritardo superiore a quello previsto o una deadline miss



Problema di scheduling generale

- ❑ Il problema di assegnare processori e risorse ai job soddisfacendo vincoli di precedenza e temporali è, nel caso generale, NP-completo e quindi *intrattabile*
- ❑ Nei sistemi real-time dinamici, le decisioni di scheduling devono essere prese *on-line*, durante l'esecuzione dei task
- ❑ Possibili elementi di semplificazione:
sistemi monoprocesso, assenza di vincoli di precedenza o di risorse, preemption, priorità statiche, task omogenei, rilascio simultaneo dei task

Scheduling multiprocessore



- ❑ Implicazioni dei risultati negativi sullo scheduling multiprocessore!
 - ❑ Nei sistemi multiprocessore e multicore real-time:
 - *task* partizionati e *assegnati staticamente* ai processori
 - scheduling con algoritmi specifici, anche dinamici, all'interno dei singoli processori
 - migrazione dei task non prevista o solo per situazioni speciali (cambiamenti di modo, sovraccarichi, emergenze, accettazione di nuovi task)
 - ottimizzazione delle performance anche in relazione agli aspetti di «*affinity*»
 - ❑ \Rightarrow focus su scheduling real-time su singolo processore
-

Classificazione degli algoritmi di scheduling



- *preemptive/non-preemptive*
 - *preemptive*: il task in esecuzione può essere interrotto per assegnare il processore ad un altro task;
 - *fully preemptive*: il task può essere interrotto in qualsiasi istante
 - *non-preemptive*: il task esegue fino al completamento; tutte le decisioni di scheduling sono prese quando il task completa l'esecuzione
- *statici/dinamici*
 - *statici*: le decisioni sono prese in base a parametri fissi, attribuiti ai task prima della loro attivazione
 - *dinamici*: le decisioni sono prese in base a parametri dinamici, variabili a tempo di esecuzione

Classificazione degli algoritmi di scheduling

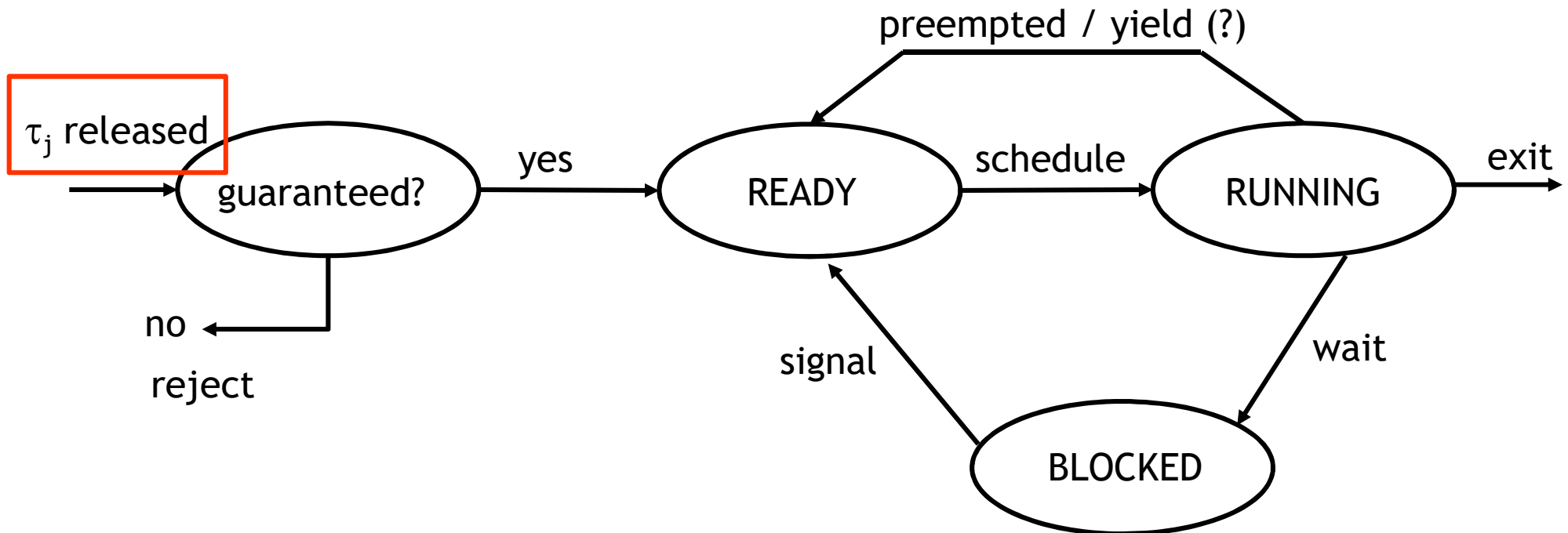


- *on-line/off-line*
 - *on-line*: le decisioni di scheduling sono prese on-line quando un nuovo task entra nel sistema o un task in esecuzione termina
 - *off-line*: l'algoritmo è eseguito sull'intero task set prima dell'esecuzione; la schedule è memorizzata in tabella ed utilizzata dal dispatcher
- *garantiti/best-effort*
 - *garantiti*: i task sono garantiti e rispettano le proprie deadline; nuovi task vengono *accettati* solo se l'intero task set resta garantito
 - *best-effort*: le deadline sono rispettate quando possibile; nuovi task vengono accettati indipendentemente dal loro impatto sulla schedule; in *media* forniscono, sul task set, risultati migliori rispetto a quelli garantiti

Sistemi con garanzia



- Per poter garantire task real time a tempo di esecuzione occorre un *sistema di accettazione*



Classificazione degli algoritmi di scheduling



- *ottimi/euristici*
 - *ottimi*: garantiscono di trovare una schedule fattibile se esiste; in presenza di una metrica di valore, trovano la schedule con il valore massimo
 - *euristici*: tendono a fornire buoni risultati senza garanzie di ottimalità della schedule

- *chiaroveggenti*
 - conoscono in anticipo gli istanti di arrivo dei task



Schedulabile?

- ❑ *Ripasso:*
- ❑ *schedule*: assegnamento di job ai processori disponibili
- ❑ *schedule fattibile*: nella schedule ogni job inizia l'esecuzione non prima dell'istante di rilascio e completa entro la sua deadline
- ❑ *ottimalità*: un algoritmo di scheduling è ottimo se è in grado di produrre sempre una schedule fattibile quando essa esiste

- ❑ → *insieme di task schedulabile*: insieme di task per il quale esiste una schedule fattibile



Valutazione di schedulabilità

- ❑ Non sempre la schedulabilità di un task set Ω in base ad un algoritmo di priorità o di scelta Γ può essere determinata agevolmente per ispezione:
 - la schedulabilità di Ω potrebbe dipendere da aspetti non noti o non modellati della applicazione o del sistema di elaborazione
 - la verifica per ispezione potrebbe essere troppo costosa
- ❑ \Rightarrow si utilizzano tecniche e criteri di analisi con diverso grado di accuratezza e costo di realizzazione
- ❑ Criteri efficienti possono essere integrati nei moduli di accettazione dei task
- ❑ Criteri utilizzati anche nel *design* del sistema hw/sw real-time



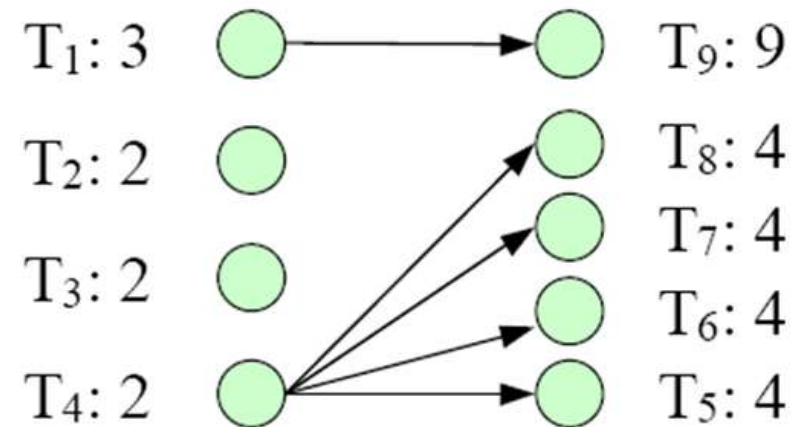
Schedulabile da Γ ?

- Un *insieme di task* Ω è *schedulabile dall'algoritmo* Γ se Γ produce una *schedule fattibile* per i task in Ω
 - L'algoritmo Γ potrebbe comunque non essere ottimo
- Il quesito sulla *schedulabilità* di un *insieme di task* Ω da parte di un algoritmo Γ in base ad un assegnato criterio di analisi ξ ammette in generale le risposte **sì | no | forse**
- Il quesito sulla *garanzia di un job o task* $\tau_i \in \Omega$ da parte di un algoritmo Γ sulla base di un criterio di analisi ξ ammette solo le risposte **sì | no**

Esercizi



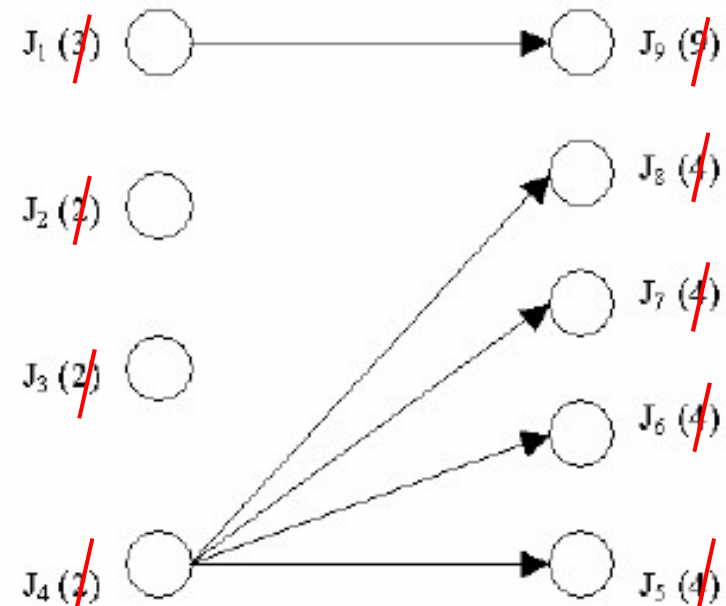
- Dato il task set aperiodico, quale è il minimo makespan ottenibile con un numero illimitato di processori?
- Quale è il numero minimo di processori che consente di ottenere il minimo makespan?
- Quale è il numero massimo di processori che è possibile impegnare nella computazione, nell'ipotesi che un processore libero debba essere impegnato se c'è un task pronto?



Esercizio 1



- Calcolare makespan con esecuzione di task *più brevi*, $C_i \rightarrow C_i - 1 \forall i$ e $K=4$ processori
- $Pri(T_i) > Pri(T_j)$ per $\forall i < j$
 - NB: correggere le durate rispetto a quelle iniziali, riportate in figura!





Esercizio 2

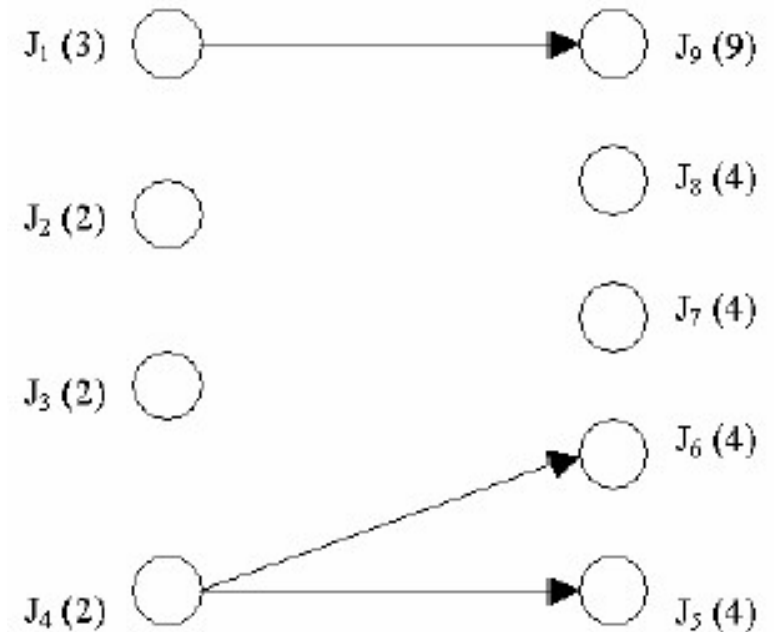
- Calcolare il makespan con 3 processori e *riduzione* dei vincoli di precedenza:

$J_1 < J_9$

$J_4 < J_6$

$J_4 < J_5$

- $Pri(T_i) > Pri(T_j)$ per $\forall i < j$



Compito per casa



- ❑ Risolvere esercizi 1 e 2 disegnando sequenze di esecuzione e riportando tempo di completamento t_c
- ❑ Inviare soluzione per mail entro **Giov. 21/3 h. 23.59**