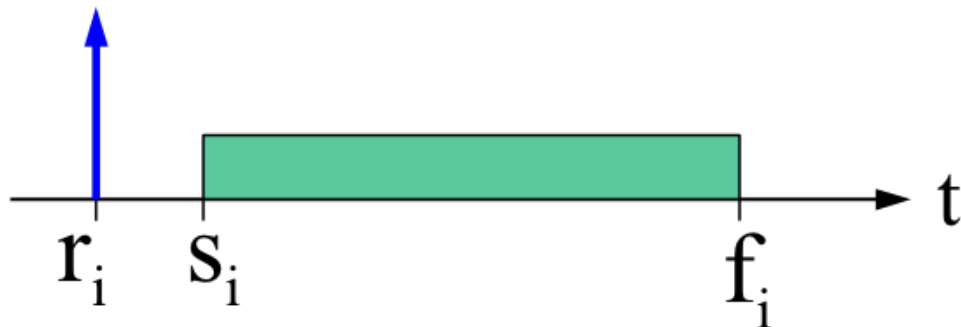


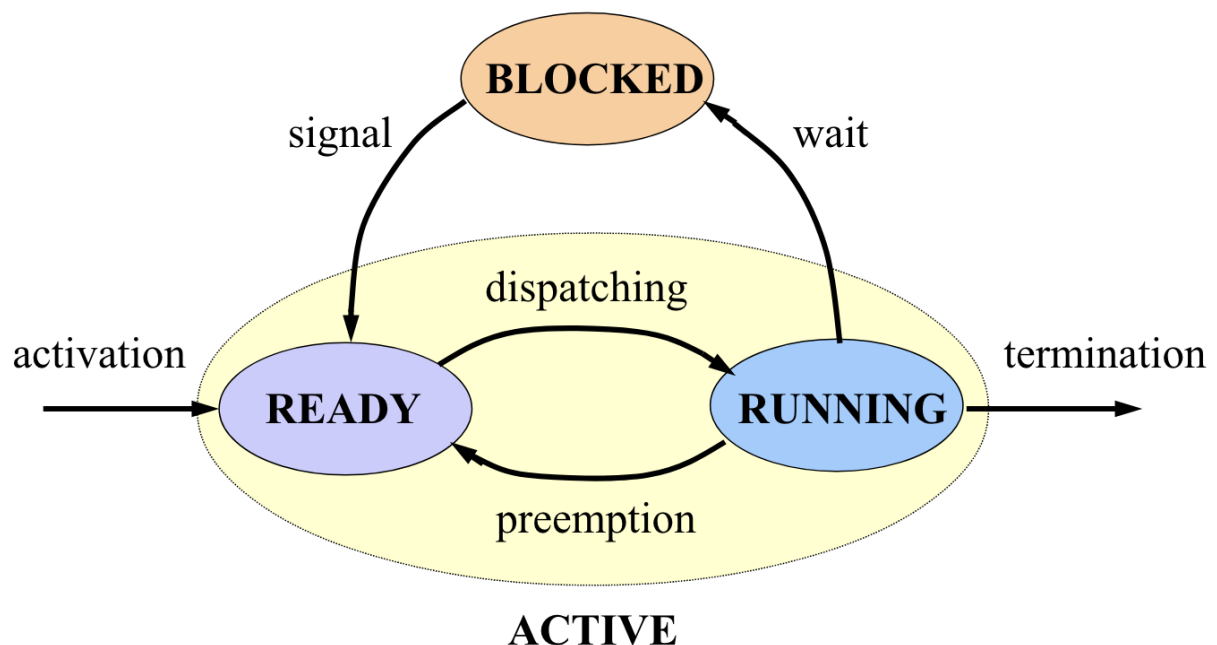
Introduction to real-time scheduling

Task

Sequenza di istruzioni che sono eseguite da un processore dall'inizio alla fine.



- $r_i \rightarrow$ quando il task mi arriva, indica lo stato ready
- $s_i \rightarrow$ quando inizia
- $f_i \rightarrow$ quando finisce



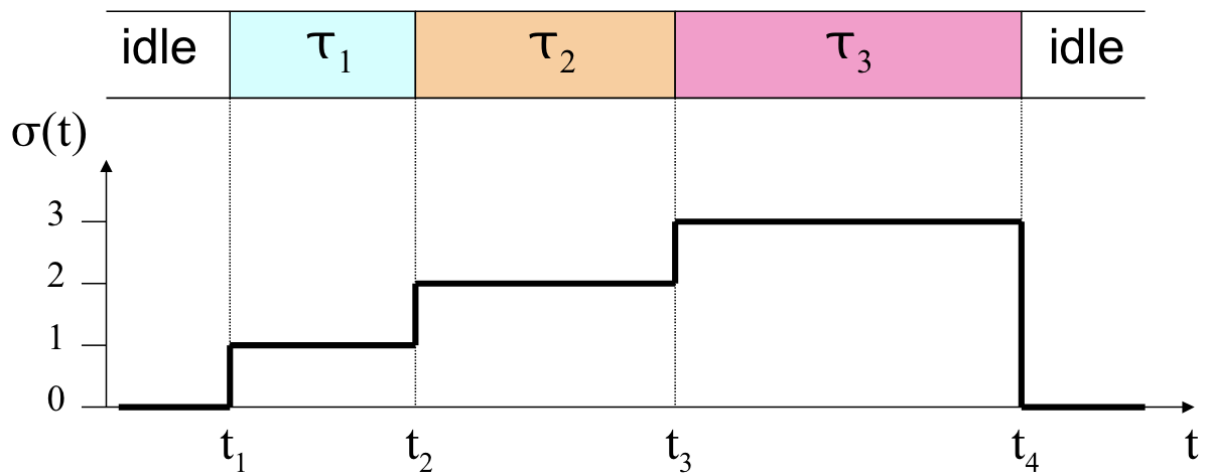
I task pronti vengono tenuti in una coda \rightarrow **READY QUEUE**

Schedule \rightarrow Particolare assegnamento dei task ai processori, chi va in esecuzione istante per istante.

Mapping che chiamiamo σ tale per cui:

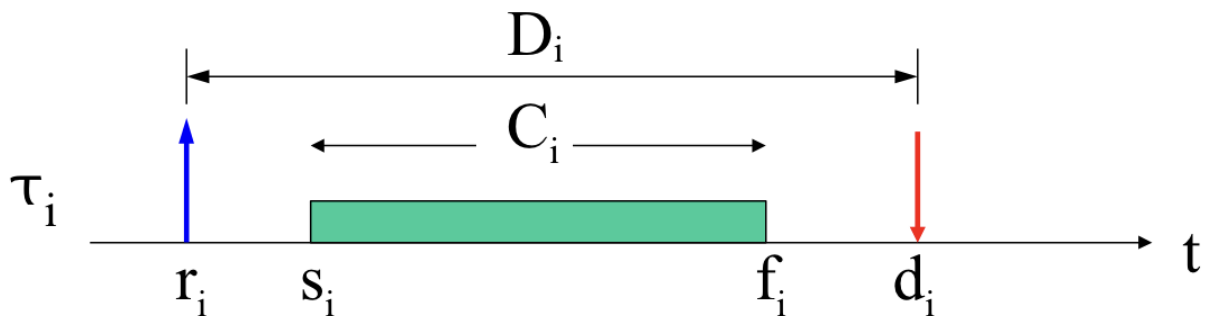
$$\sigma(t) \begin{cases} k > 0 & \text{if } \tau_k \text{ is running} \\ 0 & \text{if the processor is idle} \end{cases}$$

Idle = non sto eseguendo nessuno



- t_1, t_2, t_3 sono detti **CONTEXT SWITCH**
- Ogni intervallo dove viene eseguito un processo viene detto **time slice**

Real-time tasks



- $d_i \rightarrow$ deadline assoluta, tempo fisso.
- $D_i \rightarrow$ deadline relativa, delta di tempo.
- $C_i \rightarrow$ **Worst Case Execution Time - WCET** \rightarrow tempo di esecuzione di caso peggiore, se qualcosa andrà male dobbiamo considerare la configurazione che causa il ritardo maggiore

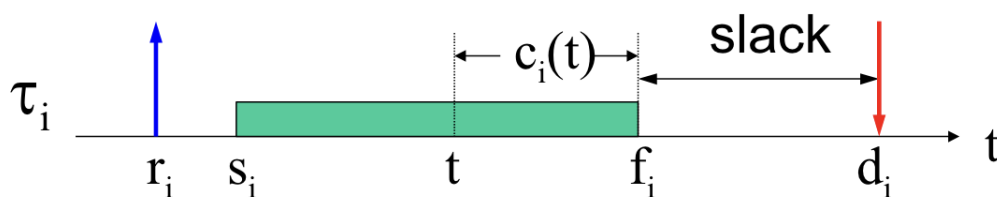
Come di ricava?

- **WCET tool** \rightarrow cercano per via sperimentale o analitica di capire quanto al peggio può durare il tempo di esecuzione di un vostro task.

Analitica \rightarrow creano un grafo con l'insieme dei Basic Block (insieme di istruzioni che sono sicuro avvengano una di fila all'altra) e valutano la peggiore

Sperimentale \rightarrow Non è safe, viene molto utilizzata in industria, non si presta a safety levels più critici. Sono necessari un mare di simulazioni 10^{20}

Ci interessa il concetto di deadline \rightarrow istante entro il quale devo assolutamente terminare prima che succedano cose più o meno disastrose



- **Lateness** = $f_i - d_i$, quando è arrivato tardi il tuo task? Se la lateness è negativa sono in anticipo
- **Tardiness** = $\max(0, L_i)$ → così che se il task finisce prima va comunque a 0
- **Tempo di esecuzione residuo - Residual WCET** → $c(t)$, quanto mi rimane da eseguire ad un istante t
- **Laxity o slack** → quanto tempo ho ancora a disposizione prima di incorrere in un deadline miss → $d_i - t - c_i(t)$,

Differenza tra task e job? Un job è un'esecuzione, un task è una sequenza infinita di jobs che si ripetono

La criticalità del task dipende da quali sono le conseguenze nel caso ho un deadline miss → HARD tasks (gravi conseguenze), SOFT task (può missare la deadline, ma cerchiamo di limitarlo), FIRM task lasciare che il task "missi" k ogni m deadline, ogni 10 periodi ne perde al massimo 2.

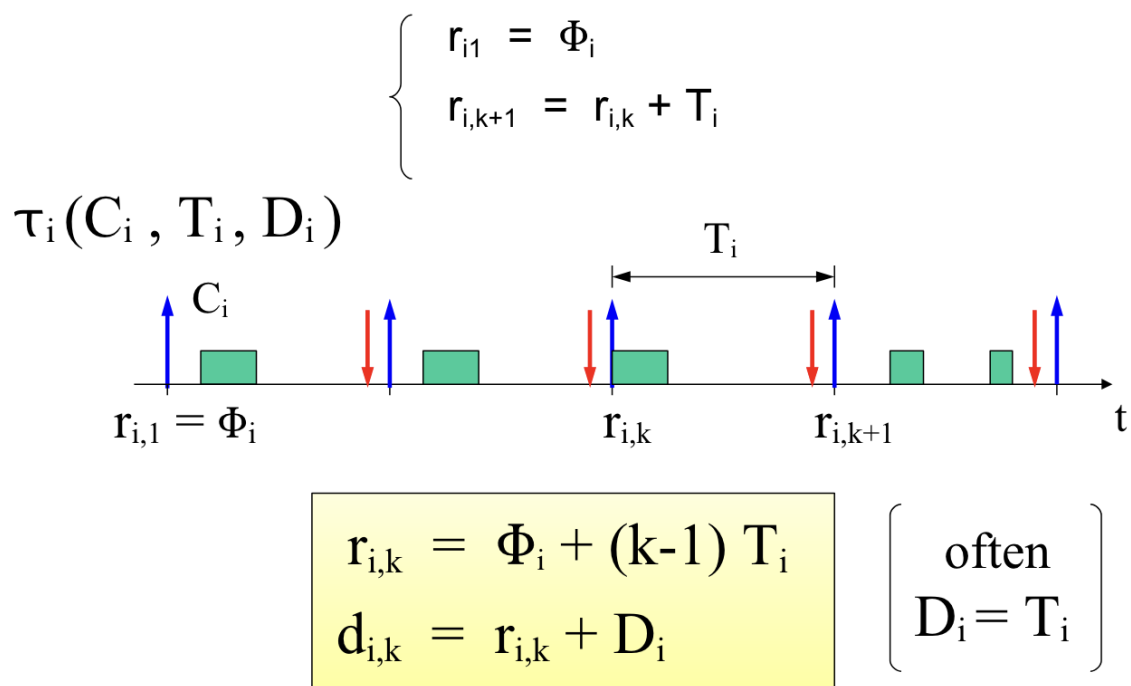
Un sistema operativo real-time è quello richiesto per task hard, perché a differenza di SO tipo linux ho più predicibilità

Hard tasks → controllo low level, sensory-motor planning

SOFT tasks → leggere la tastiera, mostrare messaggi

Modalità di attivazione

- **Time driven (Task periodico)** → un timer o un kernel che attiva ad intervalli regolari il task
 - **Task periodico**



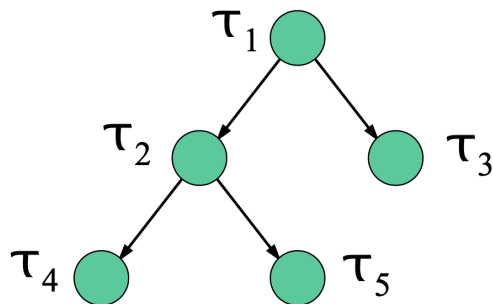
Ogni job si attiva regolarmente ogni T_i dopo quello prima.

- **Task Aperiodico**
 - **Aperiodico** → La prossima istanza mi può arrivare quando vuole (gestire allarmi)
 - **Sporadico** → I task arrivano sempre con una stessa periodicità, ma a volte con una maggiore
- **Event driven** → quando il task è attivato da qualcosa di asincrono, o ad esempio un input utente

Tipi di vincoli

- **Vincoli temporali** → deadline, attivazione, completion, jitter (quanto cambia il tempo di risposta del mio task, lo voglio più basso possibile)

- **Espliciti** → inclusi nelle specifiche di sistema → aprimi una valvola ogni 10 secondi
- **Impliciti** → non sono inclusi nelle specifiche del sistema, ma devono essere rispettati → evita gli ostacoli mentre vai ad una certa velocità
- **Vincoli di precedenza** → prima che un task possa eseguire devo aver finito il task precedente
 - Vado ad imporre un **Directed Acyclic Graph - DAG**



predecessor

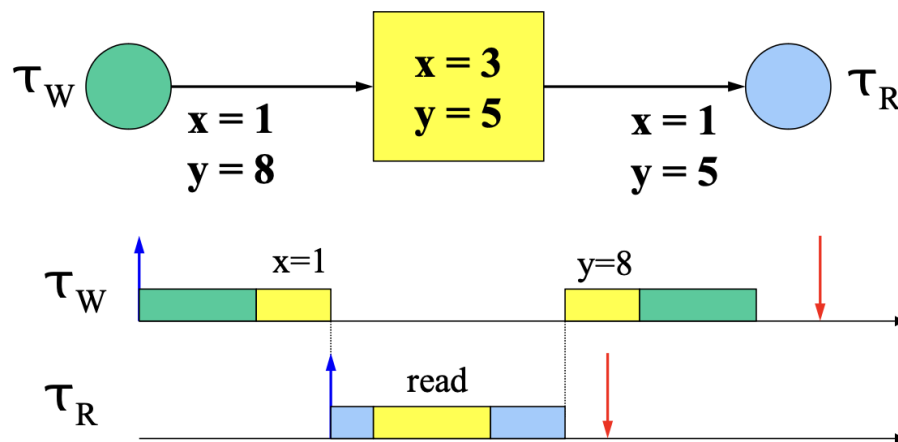
$$\tau_1 < \tau_4$$

immediate predecessor

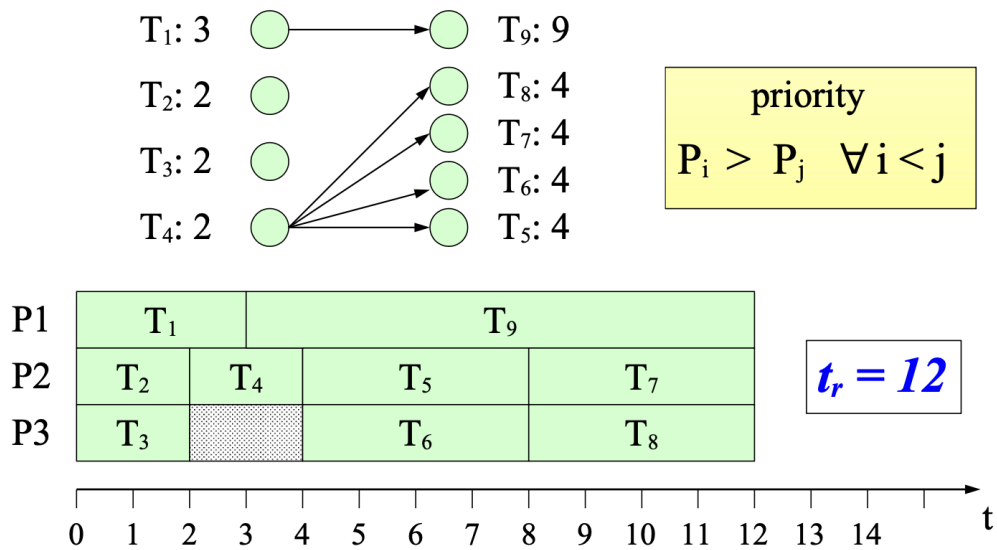
$$\tau_1 \rightarrow \tau_2$$

τ indica dei job con priorità diverse.

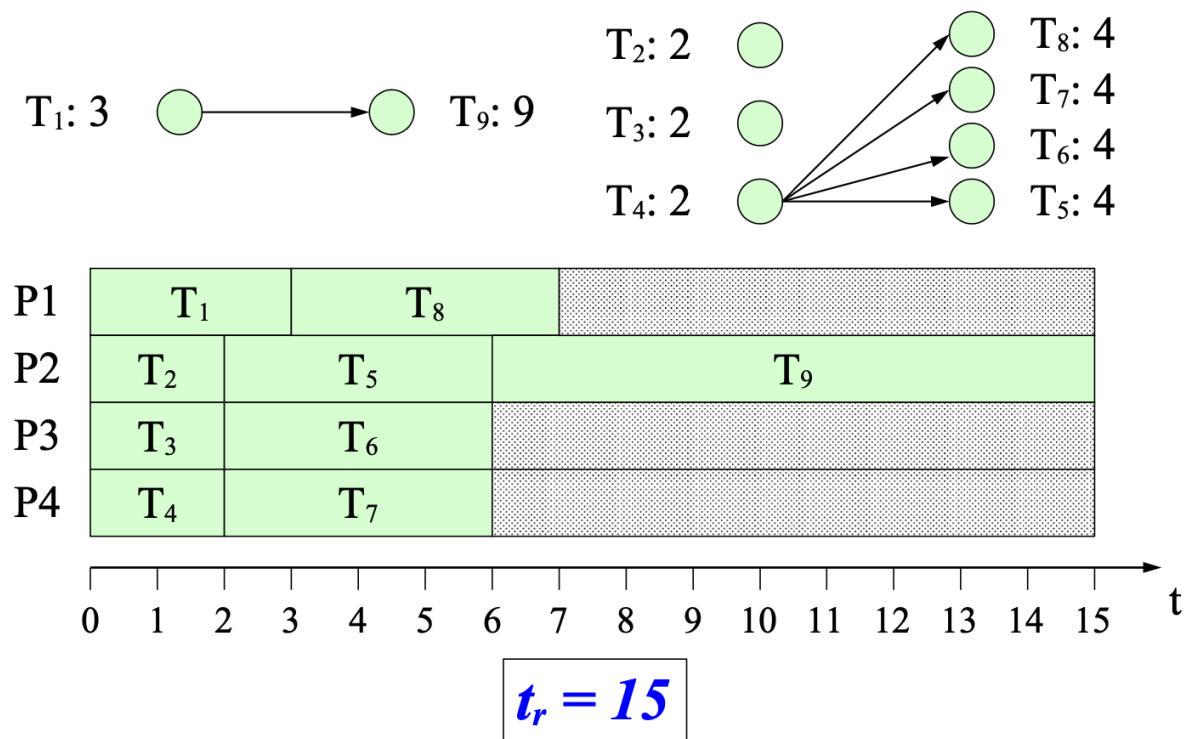
- **Vincoli sulle risorse** → ad esempio mutue esclusioni scaturite da wait e post che proteggono delle sezioni critiche
 - Prima di leggere fai una wait, finito fai una signal → questa mutua esclusione introduce ritardi.



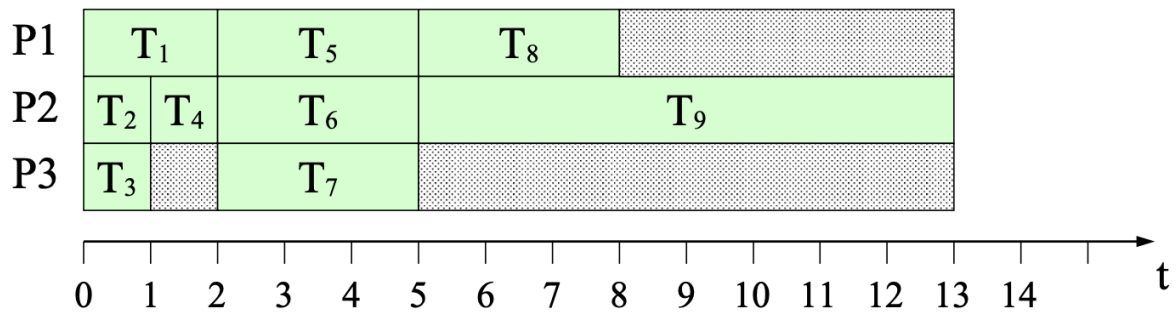
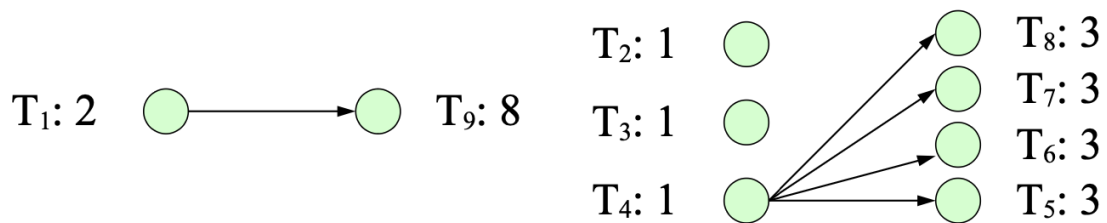
Anomalie di scheduling



- La priorità va data a chi ha indice più basso
- Scheduler non preemptive → T_9 non viene descheduled al termine di T_4

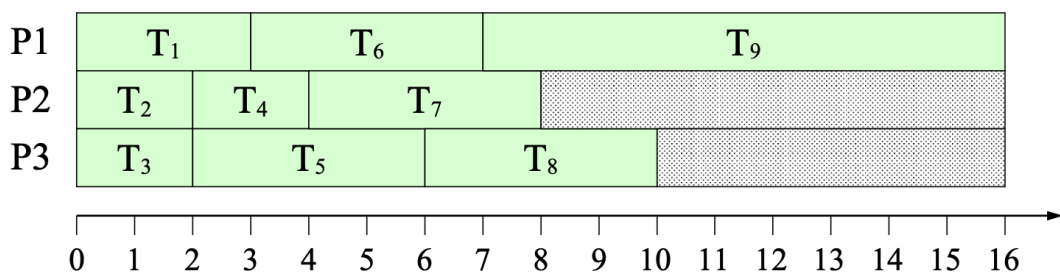
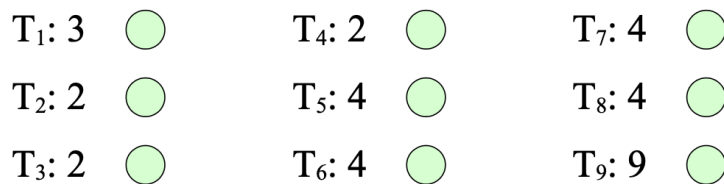


- **Bin packing** → mi causa un impacchettamento dei vari task sfortunato, invece di andare meglio aumentando i processori essi rallentano lo svolgimento, al posto di 12 impiego 15



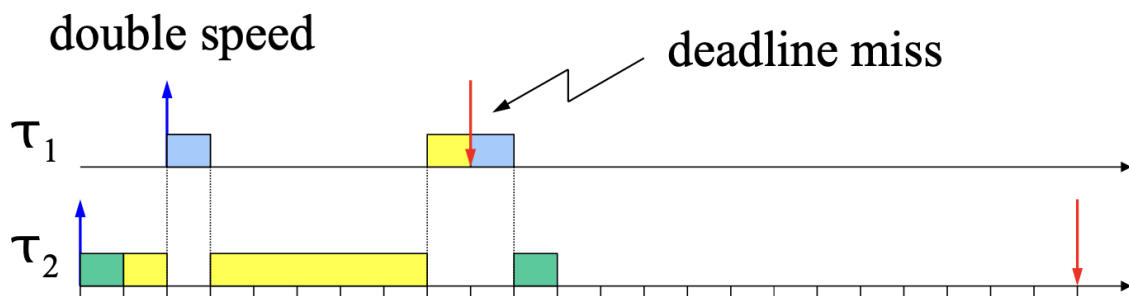
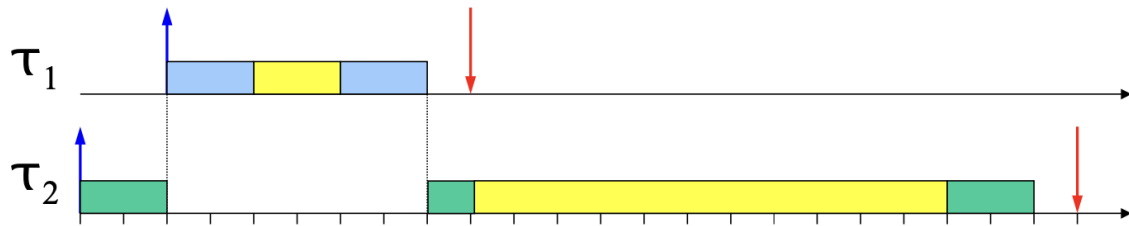
$$t_r = 13$$

- Anche diminuendo la durata dei task peggiora, invece di usare 3 processori a 2Ghrz ne uso 3 a 3 Ghrz

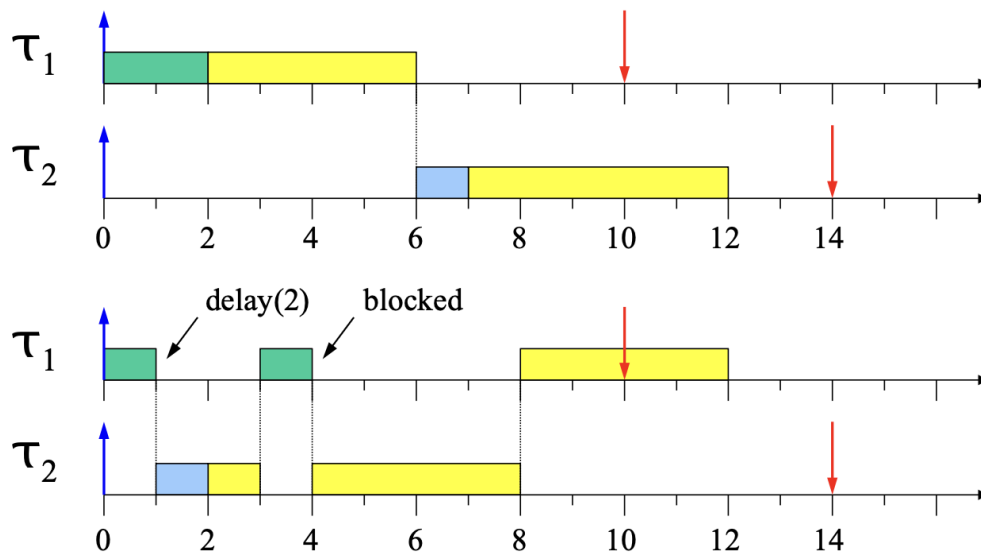


$$t_r = 16$$

- Senza vincoli di precedenza addirittura il tempo aumenta più degli altri esempi



- Passare da 1 a 2 GHz → Locko prima la risorsa gialla, ma quella azzurrina parte lo stesso, questo blocca la gialla e fa si di missare la deadline.



- Inserire dei ritardi → anche solo due istanti di tempo fanno la differenza, 6 istanze di tempo di ritardo
- Per tutto questo il sistema operativo deve essere più predicibile possibile.