Università degli Studi di Parma
Dipartimento di Ingegneria dell'Informazione
Sistemi operativi e in tempo reale – a.a. 2023/24

Sistemi in tempo reale
Definizioni

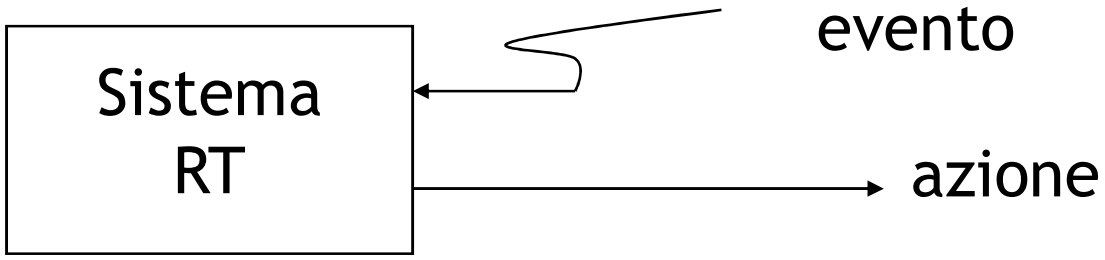
prof. Stefano Caselli

Esempi discussi

- Il generico sistema di controllo digitale multirate
- Il sistema avionico
- Il controllo di un robot mobile in ambiente domestico
- ...

- Q: Cosa succede in questi sistemi se non si reagisce *in tempo utile* agli eventi?

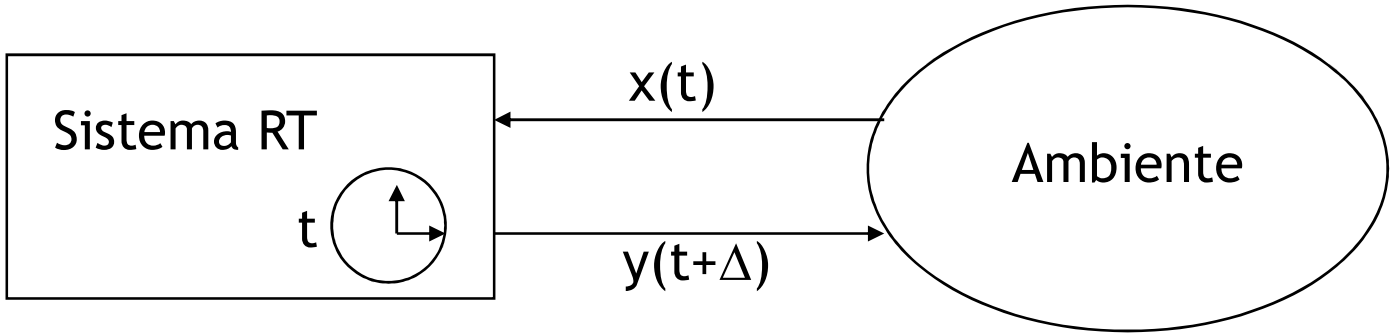
Sistema in tempo reale



- Un *Sistema in Tempo Reale* è un sistema di elaborazione in grado di rispondere agli eventi rispettando precisi vincoli temporali

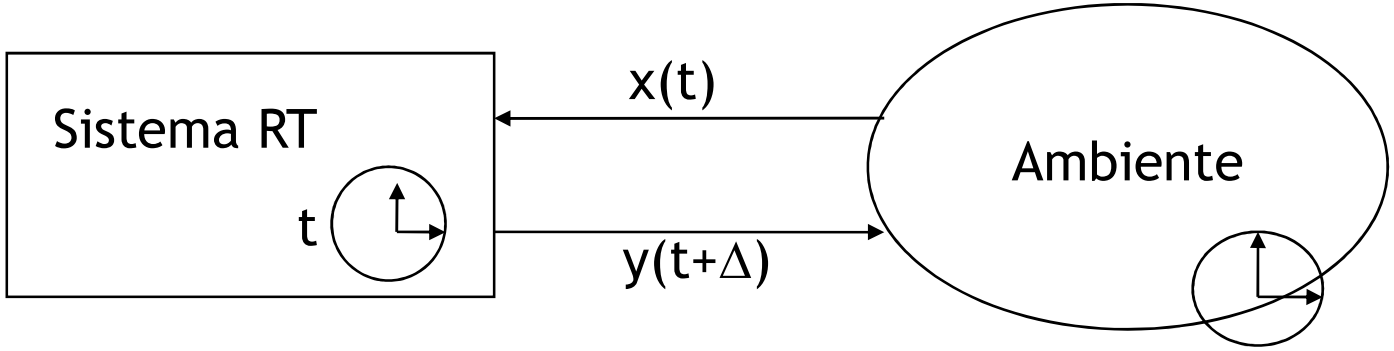
Sistema in tempo reale - definizione

- Un sistema in tempo reale è un sistema in cui la *correttezza* della elaborazione dipende *non solo dai risultati* prodotti in uscita, *ma anche dall'istante* in cui tali risultati vengono resi disponibili



Sistema in tempo reale

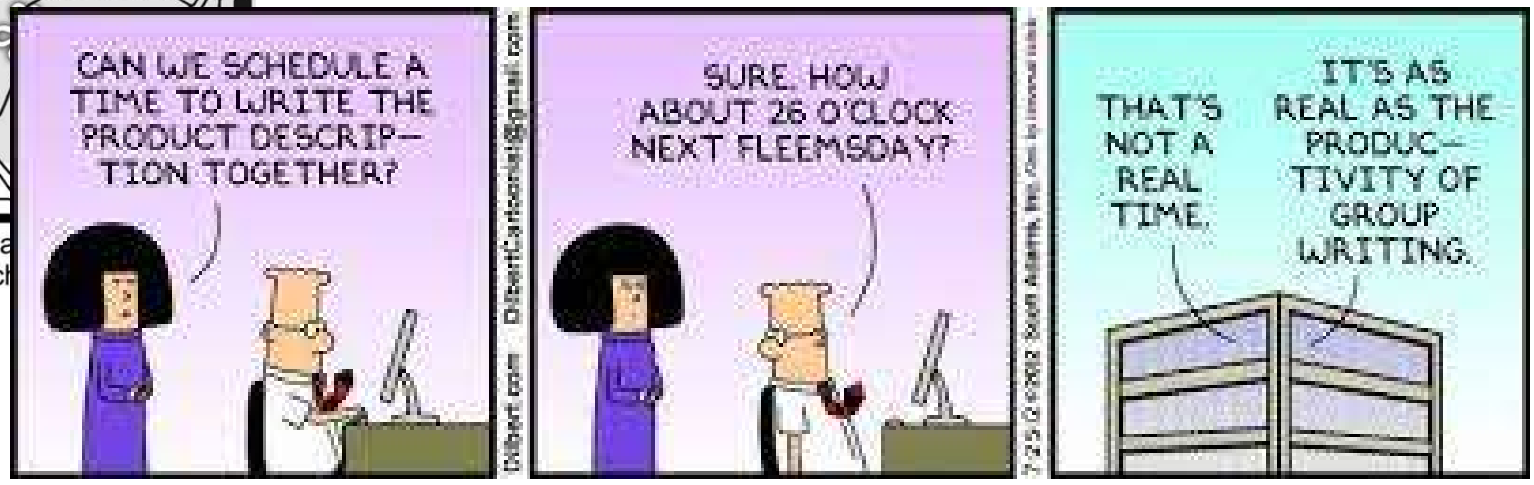
- Il tempo del sistema deve essere sincronizzato con il tempo dell'ambiente



- In un sistema non in tempo reale manca questa dipendenza da vincoli temporali

Altre definizioni di real-time

Shelly consults her basketba
to-real-time conversion ch



- Eventi sportivi in real-time? uhm

Velocità

- *Real-time ≠ veloce!*
- La velocità è sempre relativa a quella dello specifico ambiente in cui opera il sistema
- Un'esecuzione più veloce è in genere utile, ma *non garantisce* di per sé un comportamento corretto
- L'obiettivo di un sistema real-time è garantire il *comportamento temporale* di *ciascun task*
- L'obiettivo di un sistema veloce è minimizzare il *tempo medio di risposta* di un *insieme di task*
- I tempi medi tuttavia non garantiscono le prestazioni dei singoli task!

Predicibilità

- Requisito essenziale di un sistema RT è la *predicibilità*
- In presenza di elevate incertezze (sulle caratteristiche del task set e del sistema di elaborazione) è impossibile o difficile fornire garanzie
- La predicibilità al 100% esiste nelle analisi formali, mentre si trova di rado nei problemi reali
  → *analisi di caso peggiore* e inclusione degli *overhead*
- Se la variabilità è eccessiva, si può valutare se è comunque possibile e utile fornire *garanzie ad un sottoinsieme di task*

Fonti di non determinismo

- Architettura
  - cache, elaborazione in pipeline, esecuzione speculativa, branch prediction, interrupt, DMA
  - → trend verso architetture ad alte prestazioni ma con crescente non determinismo
- Sistema operativo
  - scheduling, sincronizzazione, comunicazione, memoria virtuale
- Linguaggio di programmazione
  - possibile assenza di supporti per la gestione del tempo
- Metodologia di progetto
  - assenza di tecniche di analisi e verifica

Definizioni (Liu)

- *Job*: unità di lavoro schedulata ed eseguita dal sistema
- *Task*: insieme di job correlati che realizzano collettivamente una funzionalità del sistema
- *Processore*: risorsa necessaria al job per l'esecuzione (CPU, disco, rete);
  - *server* in teoria delle code

- Astraiamo, in generale, dalla specifica elaborazione o trasmissione (job) e dalla specifica risorsa (processore)

Vincoli temporali

- *Istante di rilascio* (istante di attivazione, istante di richiesta): istante in cui il job diviene disponibile per l'esecuzione
- *Deadline* (scadenza): istante entro cui l'esecuzione deve essere completata
  - *Deadline relativa*: massimo tempo di risposta tra l'istante di rilascio e l'istante di completamento
  - *Deadline assoluta* (= Deadline):
    = istante di rilascio + deadline relativa

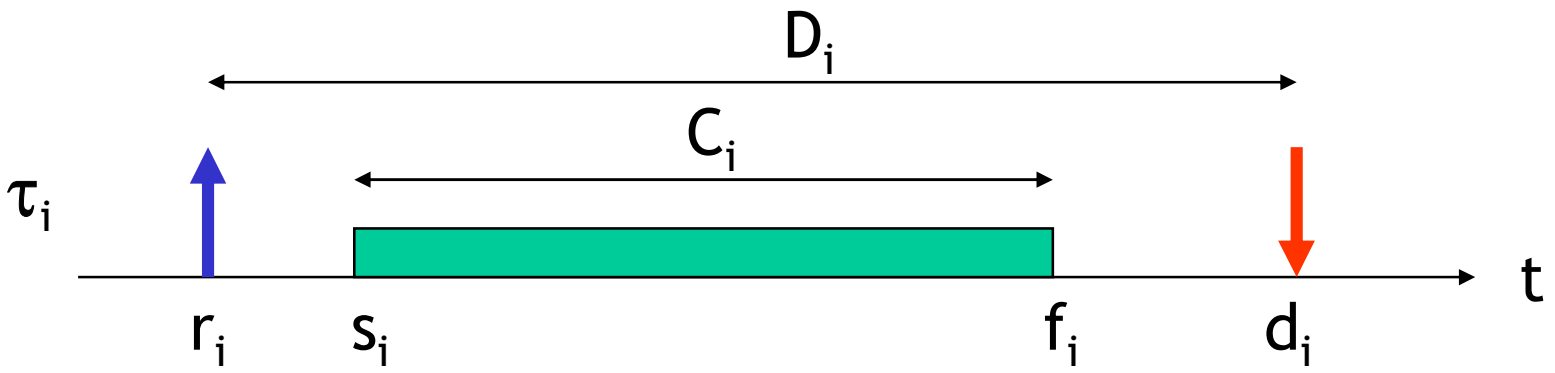
- Istante di rilascio e deadline consentono di specificare i vincoli temporali più frequenti

Deadline *hard* e *soft*

- *Deadline miss*: mancato rispetto di una deadline (produzione in ritardo del risultato, ovvero il risultato non è pronto nell'istante di deadline)
- Intuitivamente:
  - *Hard deadline*: una deadline miss è considerata un *guasto fatale* o che può provocare conseguenze disastrose
  - *Soft deadline*: una deadline miss è *indesiderabile*, ma tollerabile se non troppo frequente; miss ripetute rendono le prestazioni del sistema sempre più scadenti

- Problema: Definizioni di *deadline hard* e *soft* non formalizzate né riferite a valori quantificabili!

Parametri dei job real-time



r_i istante di rilascio (o di richiesta, tempo di arrivo a_i)

s_i istante di inizio *(start time)*

C_i tempo di esecuzione del job *(execution time)*

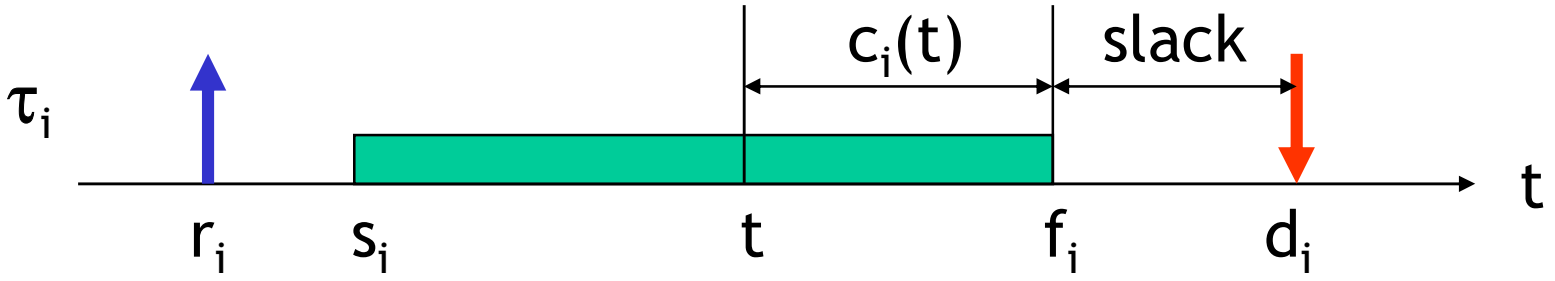
$WCET_i$ *(worst-case execution time)* valore max che può assumere C_i

d_i deadline assoluta *(scadenza)*

D_i deadline relativa

f_i ist. di completamento *(completion* o *finishing time)*

Altri parametri



Lateness: $L_i=f_i - d_i$

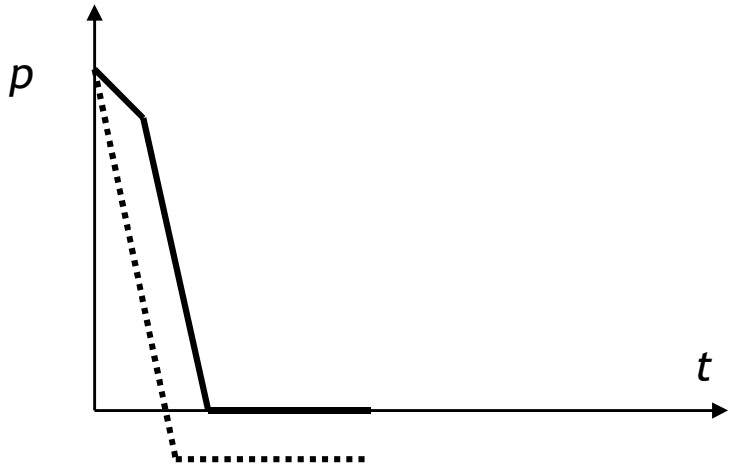
Tardiness: $\max (0, L_i)$

WCET residuo: $c_i(t)$, $c_i(r_i)=C_i$

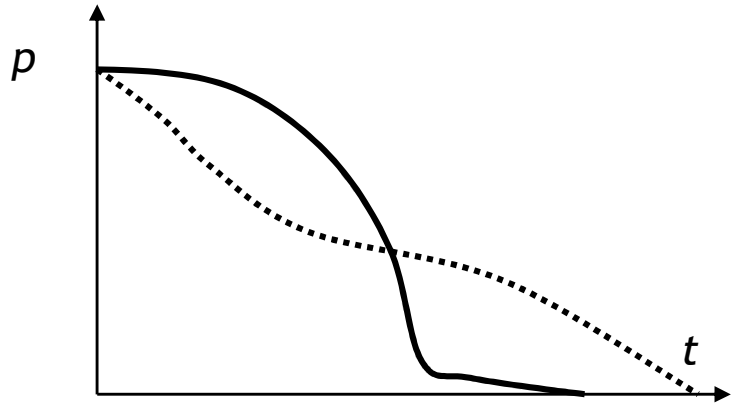
Lassità (slack): $d_i - t - c_i(t)$

Deadline *hard* e *soft*

- Misura quantitativa: *prestazione complessiva p* del sistema in funzione del ritardo dei job (*tardiness*, *t*)
- Tardiness: ritardo di completamento rispetto alla deadline, è 0 se il job completa entro la deadline



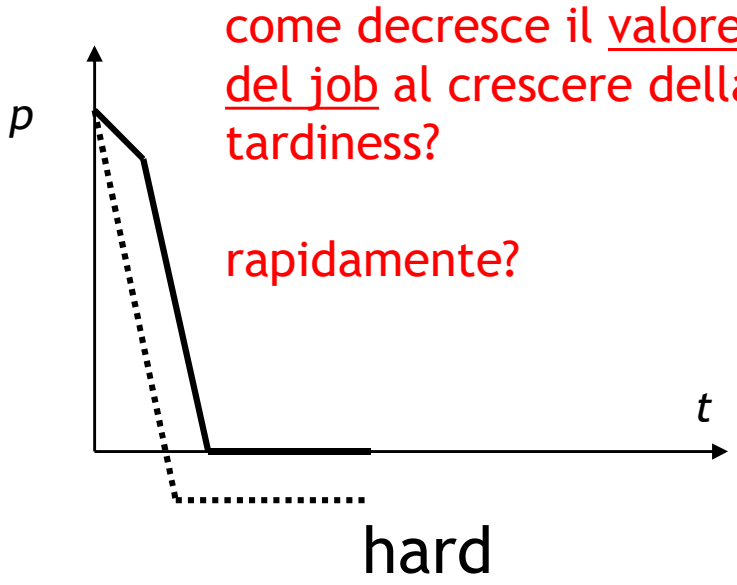
hard

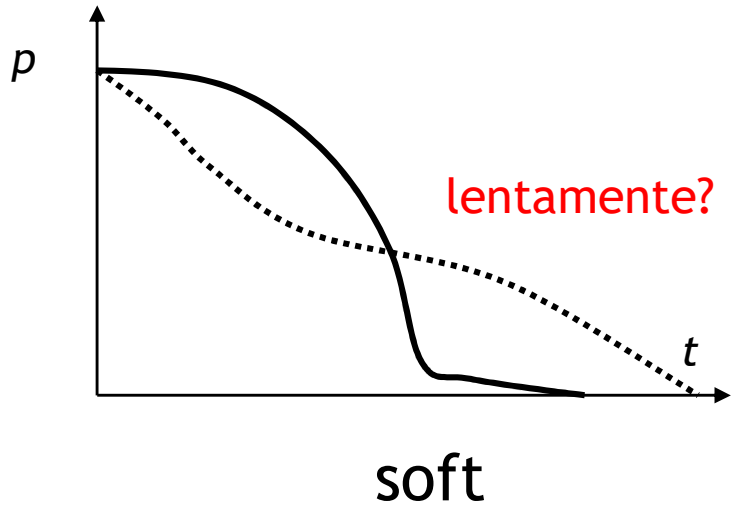


soft

Deadline *hard* e *soft*

- Misura quantitativa: *prestazione complessiva p* del sistema in funzione del ritardo dei job (*tardiness*, *t*)
- Tardiness: ritardo di completamento rispetto alla deadline, è 0 se il job completa entro la deadline





è un criterio convincente?

Deadline *hard* e *soft*

- Anche una definizione basata sul concetto di valore in funzione della tardiness *p(t)* non è risolutiva, per la difficoltà di stabilire metriche univoche

- Definizione *operativa*:

  Un *job* ha una deadline di tipo *hard* quando il progettista deve *dimostrare* che il job *rispetta sempre la deadline* nelle condizioni di funzionamento specificate

Sistemi hard real-time

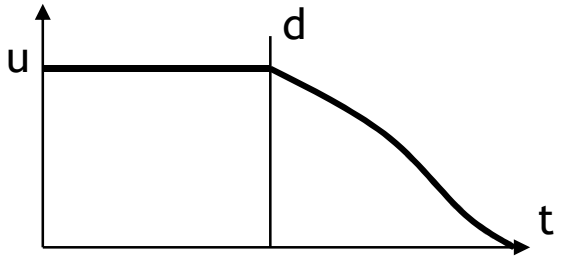
- Un *sistema* real-time viene definito *hard real-time* se *alcuni dei job* da cui è composto presentano *deadline di tipo hard*
  - Nota: almeno due job hard… garantire un job è banale!
- Esempi:
  - Sistemi embedded
  - Procedure di recovery in sistemi high-availability

- I sistemi hard real-time che devono gestire *dinamiche di impianto molto veloci* tipicamente richiedono *sistemi operativi specializzati* (esigenze di determinismo e vincoli di latenza max)

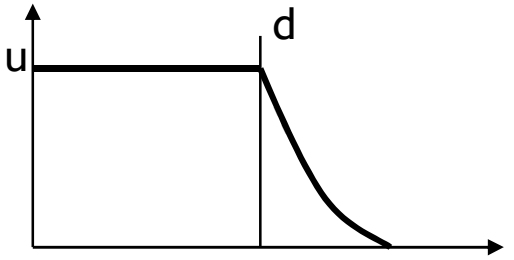
Sistemi soft real-time

- Un sistema real-time viene definito *soft real-time* se i job hanno deadline di tipo soft

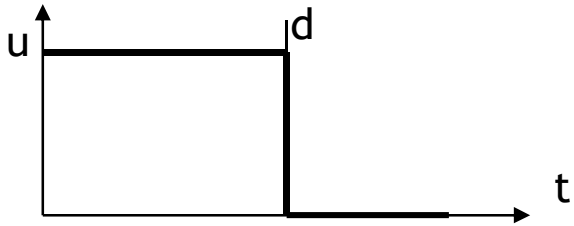
Vincoli temporali non stringenti:
- sistemi per transazioni on-line
- centraline di commutazione



Vincoli temporali più stringenti:
- sistema di borsa telematica



Vincoli temporali stringenti:
- multimedia



Sistemi soft real-time

- I *requisiti* sono spesso specificati in termini probabilistici
- La *validazione* avviene di solito mediante simulazione e prove sul sistema

Richiami e notazioni

- Da SisOp...

- Liu utilizza il termine *Job*; nella letteratura real-time vengono tuttavia utilizzati anche i termini *Task*, *Processo*, *Thread*
- Usiamo i termini Job, Task, Processo in modo intercambiabile, ove non sorga confusione
- Usiamo Thread in contrapposizione a Processo
- Più precisamente: il Thread è il *meccanismo* del sistema operativo per eseguire l'*attività*, Task o Job, oggetto dello scheduling in tempo reale

Thread non real-time

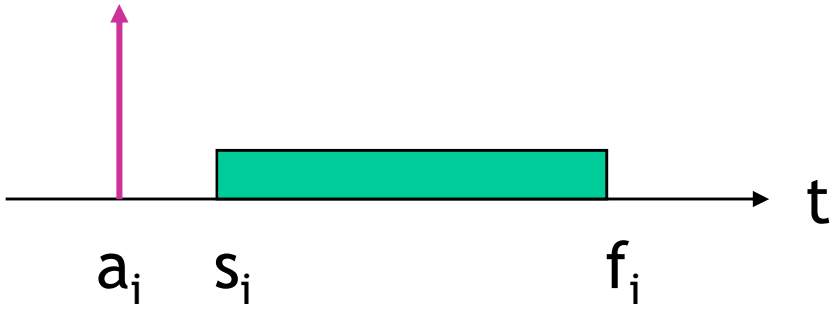
- Thread o task:

  sequenza di istruzioni che, in assenza di altre attività, viene eseguita in modo continuativo dal processore fino al suo completamento

a_i = tempo di arrivo (arrival time)

s_i = tempo di inizio esec. (start time)

f_i = tempo di fine esec. (finishing time)



Cosa manca in questa figura?

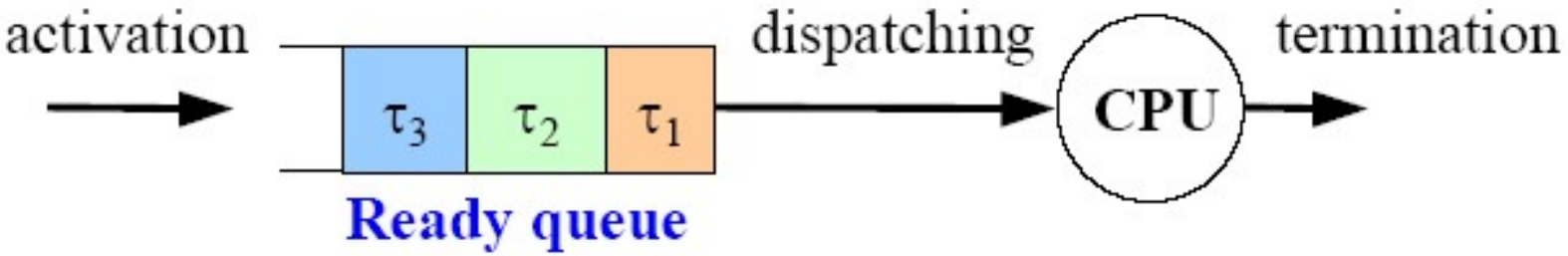
Stato di un thread

- Prescindendo dalla disponibilità della CPU:
  - *Attivo:* il task può essere eseguito dalla CPU
  - *Bloccato:* in attesa di un evento

- Un task *Attivo* può essere:
  - *Running*: in esecuzione dalla CPU
  - *Pronto:* in attesa della CPU

I thread pronti

- I descrittori dei thread pronti sono organizzati in una coda di attesa (o altra struttura dati più generale) denominata *ready queue* (coda dei thread pronti)
- La strategia per la scelta del task da porre in esecuzione sulla CPU è l'*algoritmo di scheduling*



Scheduling e revoca

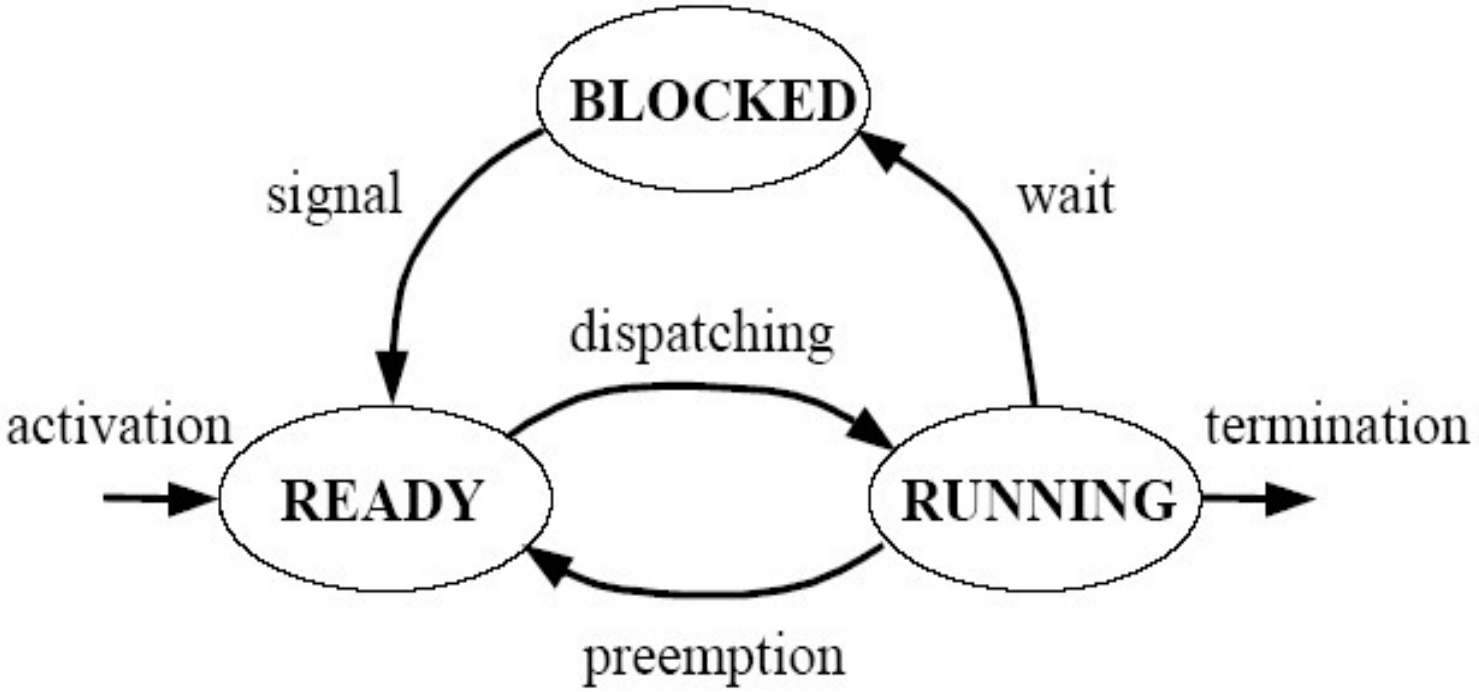
- Scheduling *preemptive*

  il task in esecuzione può essere temporaneamente sospeso, inserendolo nella ready queue, a favore di un task più importante

- Scheduling *non preemptive*

  il task in esecuzione non è soggetto a revoca fino al suo completamento

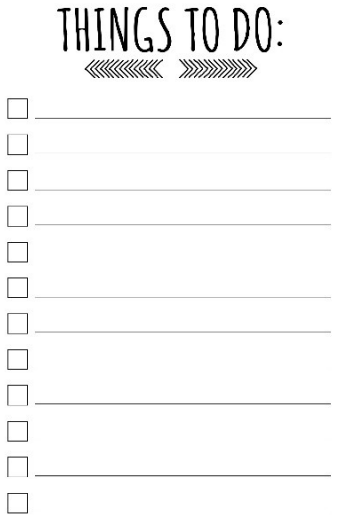
Transizioni di stato dei task



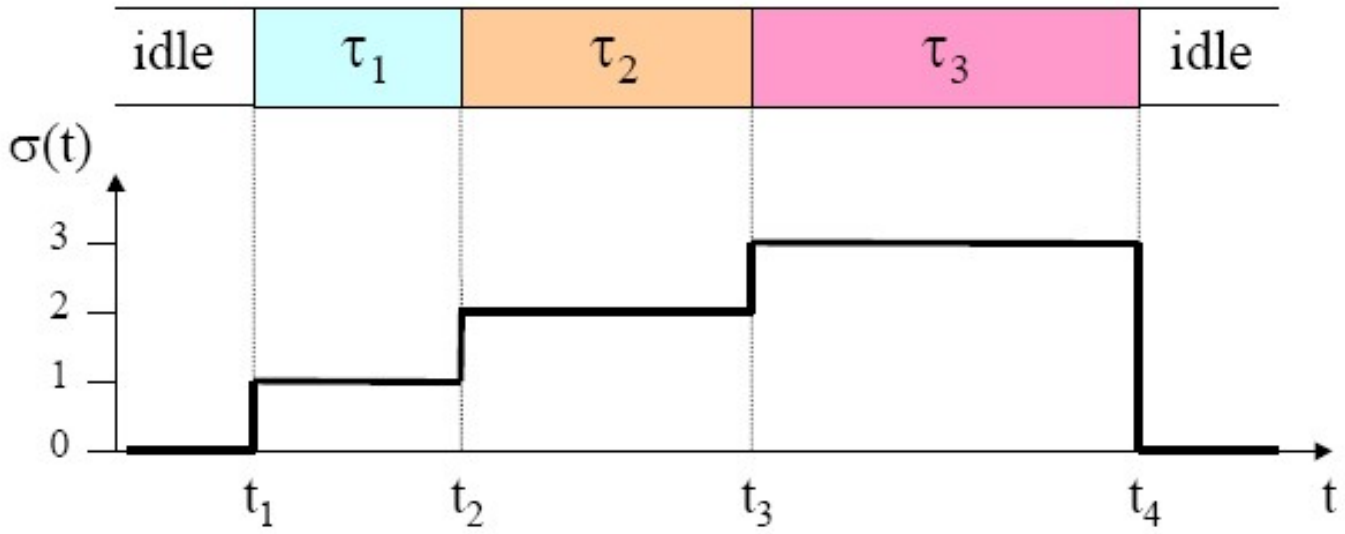
Schedule

- Una schedule (lista di assegnamento) è uno specifico assegnamento di task al processore
- Dato un insieme di task $\Gamma=\{\tau_1,\ldots,\tau_n\}$, una schedule è un mapping $\sigma : R^+ \rightarrow N$ tale che $\forall t \in R^+, \exists t_1, t_2 : t \in [t_1, t_2)$ e $\forall t' \in [t_1, t_2) : \sigma(t)=\sigma(t')$

$\sigma(t) = k > 0$ se τ_k è in esecuzione
$\sigma(t) = 0$ se il processore è inattivo (idle)

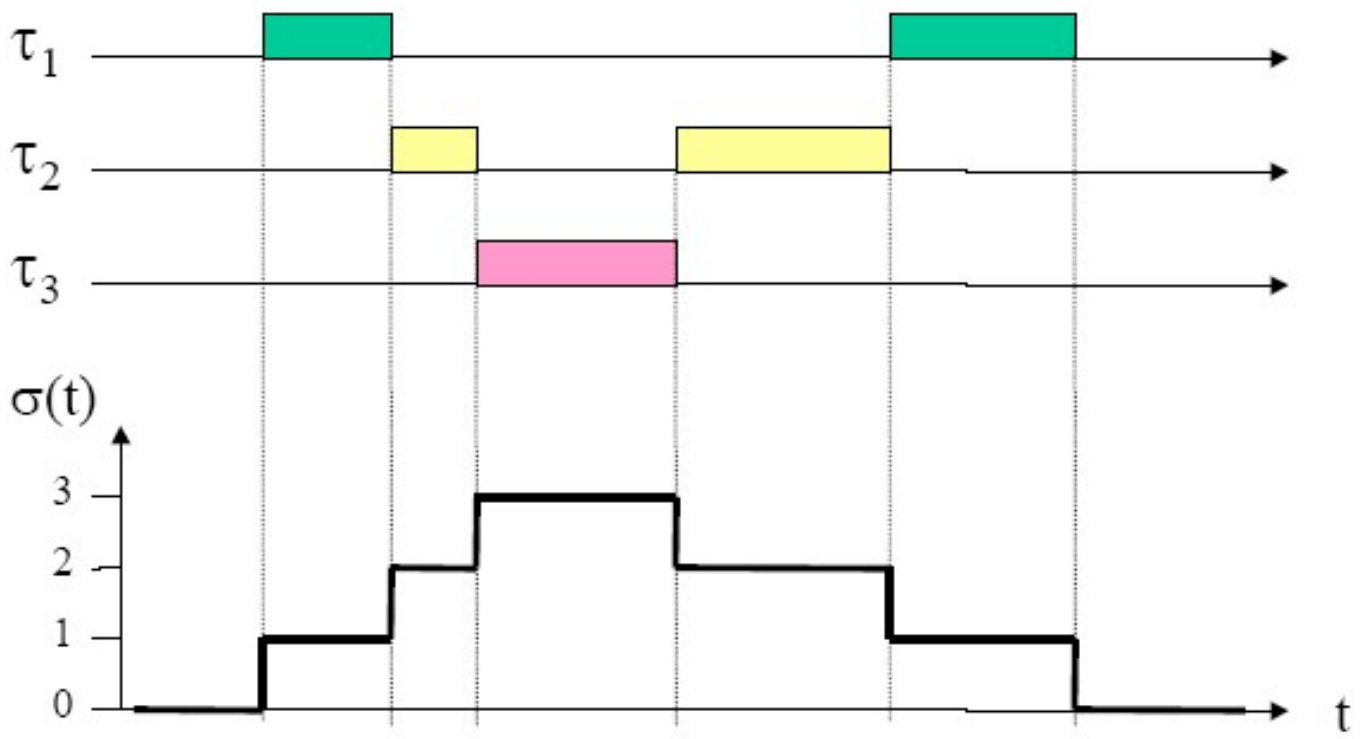


Un esempio di schedule



- Agli istanti t_1, t_2, t_3, t_4 viene eseguito un *thread switch*
- Ogni intervallo $[t_i, t_{i+1})$ è denominato *time slice*

Schedule con preemption



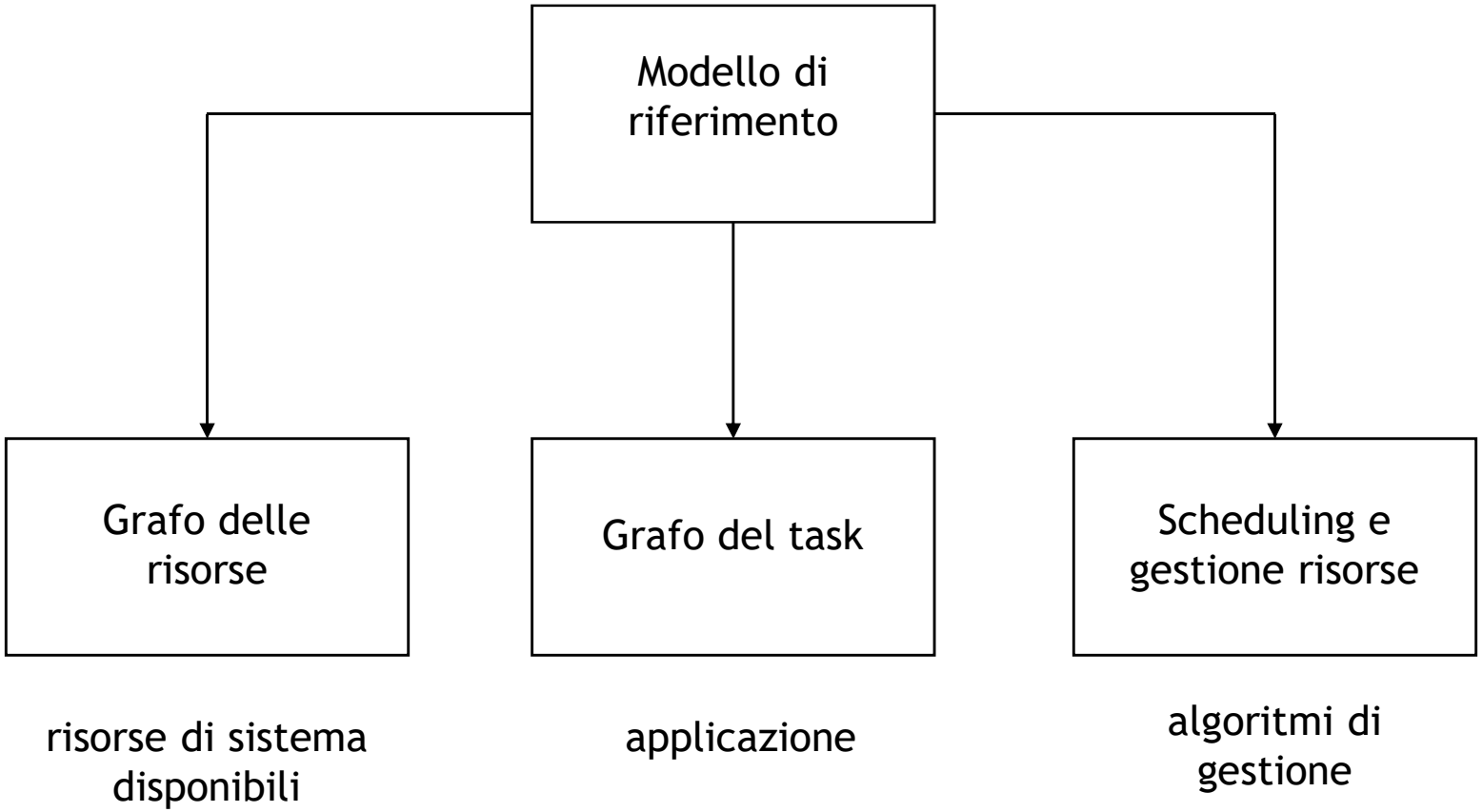
Università degli Studi di Parma
Dipartimento di Ingegneria dell'Informazione
Sistemi operativi e in tempo reale – a.a. 2023/24

Modello di riferimento per sistemi real-time

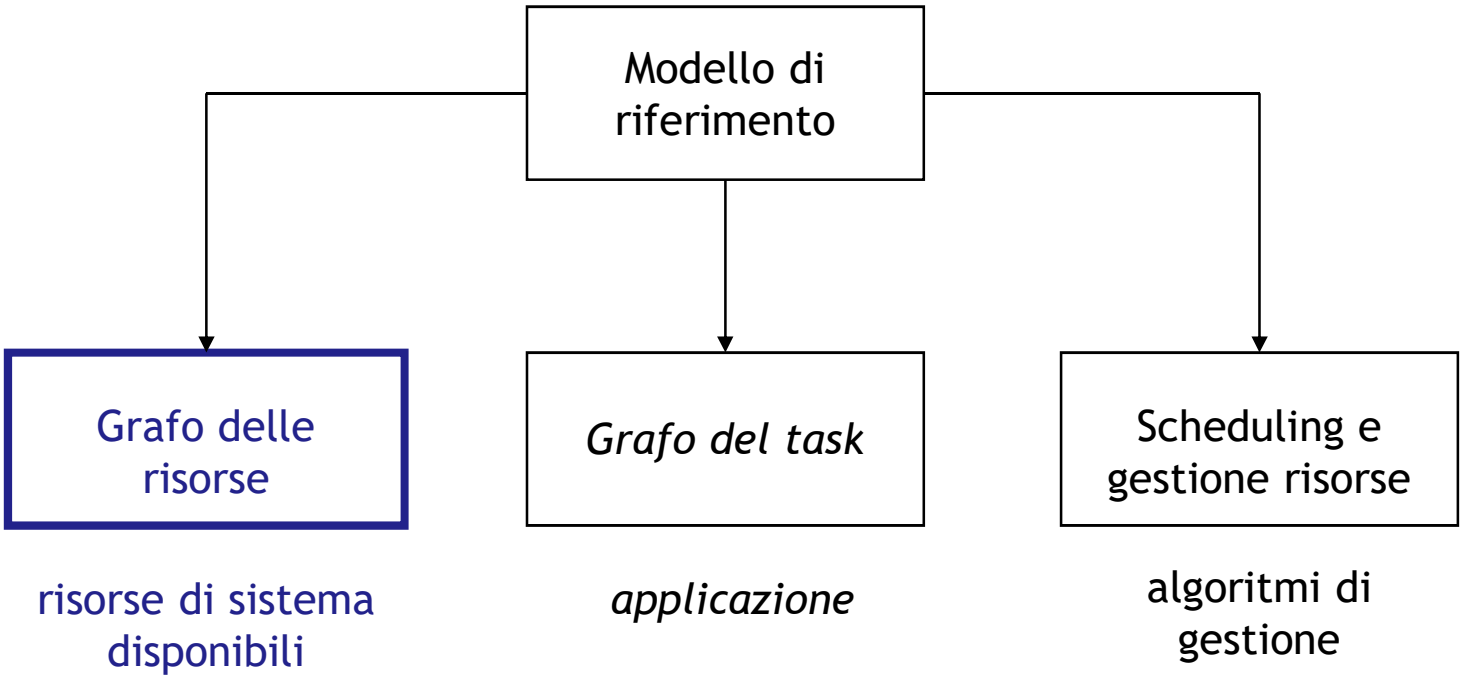
Obiettivi del modello

- Astrarre dalle caratteristiche funzionali dei sistemi
- Evidenziare le proprietà temporali ed i requisiti di risorse

Struttura del modello



Modello delle risorse



Processori e risorse

- *Processori*: server, risorse attive (CPU, tratte di rete, dischi, etc.)

- *Tipi di processori*:
  - due processori sono *dello stesso tipo* se funzionalmente identici e possono essere scambiati tra loro
  - Es.: tratte di rete tra due *peer* e con lo stesso transmission rate, CPU in sistemi SMP e multicore omogenei
  - processori *di tipo diverso* non possono essere scambiati tra loro
  - Es.: differenze funzionali (CPU vs. disco) o differenze di ruolo nella topologia del sistema
- $P_1,\ldots, P_m$

Processori e risorse

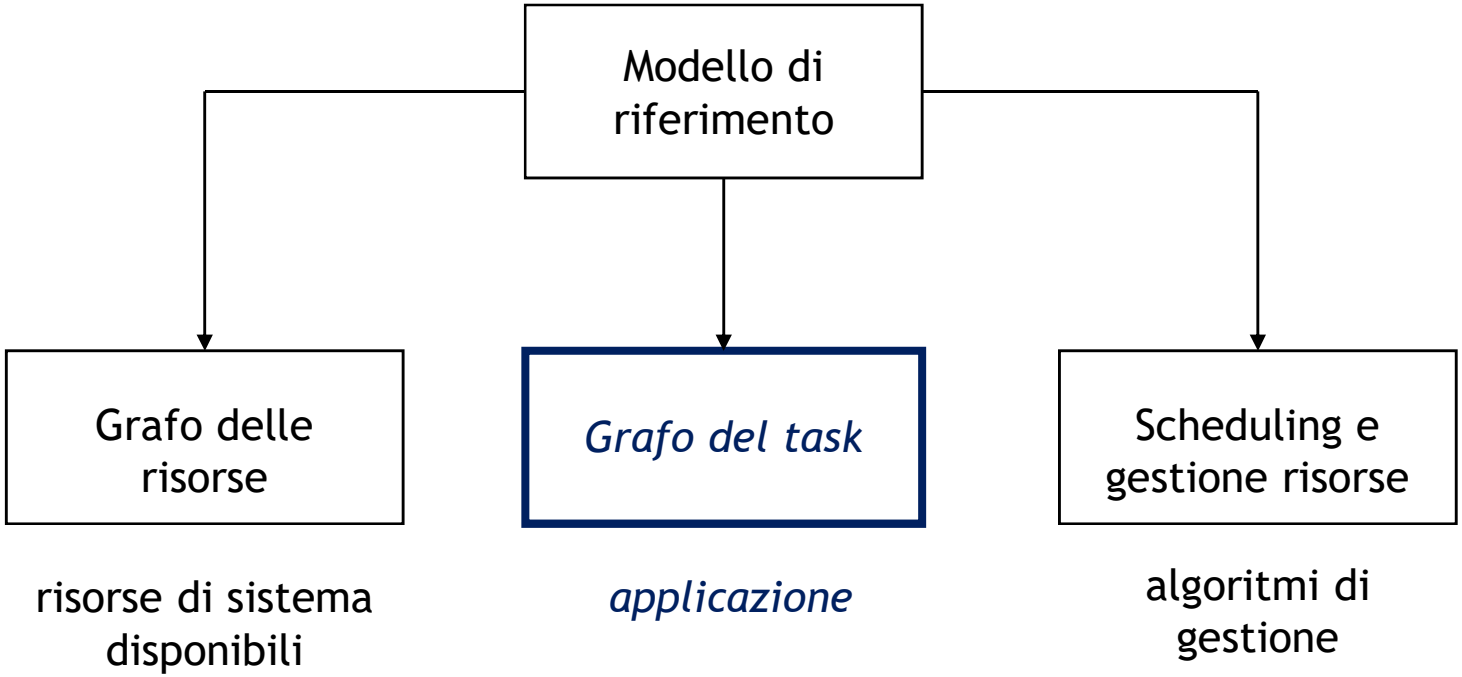
- *Risorse*: risorse passive
- Risorse necessarie, in aggiunta ai processori, per assicurare l'avanzamento della applicazione
- Non sono caratterizzate da un parametro di velocità (diversamente dai processori)
- Es.: semafori per accesso a sezione critica, lock
- Es.: data link gestito con finestra mobile
  - Job: trasmissione di un messaggio
  - Processore: data link
  - Risorsa: numero di sequenza valido
- $R_1, \ldots, R_s$

Risorse

- Risorse *riusabili*:
  - rese nuovamente disponibili dopo l'uso, riutilizzabili in modo sequenziale (*serially reusable*)
  - talvolta dette *serializzabili*, intendendo che devono essere assunte in modo sequenziale
  - es.: semafori, data lock, numeri di sequenza
  - possono essere divise in *tipi* ed *istanze per tipo*
- Risorse *consumabili*:
  - scompaiono dopo l'uso
  - es.: messaggi
- Risorse *abbondanti*:
  - nessun job è ritardato per l'attesa di queste risorse
  - solitamente trascurabili dai modelli
  - es.: memoria, se preallocata in modo statico e sufficiente

Modello della applicazione

- Viene detta anche carico di lavoro o *workload*



Parametri temporali

- J_i: *Job*, unità di lavoro
- T_j o τ_j: *Task*, insieme di job correlati
- r_i: *istante di rilascio* di J_i
- d_i: *deadline assoluta* di J_i
- D_i: *deadline relativa* di J_i
- e_i o C_i: *tempo di esecuzione* (massimo) di J_i, WCET

- Perchè WCET?
  - la variabilità dei C_i è tipicamente ridotta *nei task RT*
  - le frazioni di tempo e risorse non utilizzate sono rese disponibili a processi *soft* real-time o *non* real-time

Modello per task periodici

- Insieme di *task* : $\tau_1,\ldots \tau_n$
- Ogni task consiste di *job* : $\tau_i=\{J_{i1}, J_{i2}, \ldots\}$
- Φ_i : *fase* di τ_i, $\Phi_i = r_{i1}$ istante di rilascio del primo job
- T_i : *periodo* di τ_i , intervallo minimo tra due istanti di rilascio
- H : *iperperiodo*, $H=mcm(T_1, \ldots, T_n)$
- C_i : *tempo di esecuzione* di τ_i
- u_i : *utilizzazione* di τ_i , $u_i = C_i/T_i$
- D_i : *deadline relativa* di τ_i , spesso $D_i = T_i$

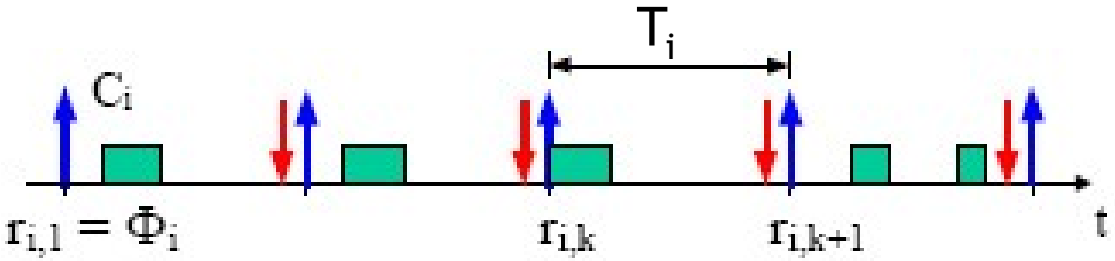
Modello per task periodici

- Insieme di *task* : $\tau_1,\ldots \tau_n$
- Ogni task consiste di *job* : $\tau_i=\{J_{i1}, J_{i2}, \ldots\}$
- Φ_i : *fase* di τ_i, $\Phi_i = r_{i1}$ istante di rilascio del primo job
- T_i : *periodo* di τ_i , intervallo minimo tra due istanti di rilascio
- H : *iperperiodo*, $H=mcm(T_1, \ldots, T_n)$
- C_i : *tempo di esecuzione* di τ_i
- u_i : *utilizzazione* di τ_i , $u_i = C_i/T_i$
- D_i : *deadline relativa* di τ_i , spesso $D_i = T_i$

E' un caso di interesse per le applicazioni?

Modello per task periodici

- $r_{i,1} = \Phi_i$
- $r_{i,k+1} = r_{i,k} + T_i$



$r_{i,k} = \Phi_i + (k-1)T_i$

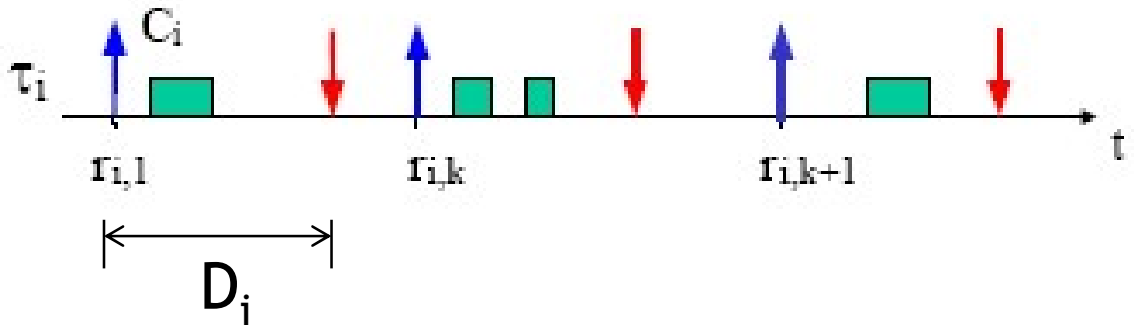
$d_{i,k} = r_{i,k} + D_i$

spesso $D_i = T_i$

τ_i caratterizzato da (C_i, T_i, D_i, Φ_i)

Modello per task aperiodici e sporadici

- Task *aperiodici*: $r_{i,k+1} > r_{i,k}$
- Task *sporadici*: $r_{i,k+1} > r_{i,k} + T_i$
- T_i è il *minimum inter-release time*



Task aperiodici e sporadici

- Spesso rappresentano eventi modellati, ma di cui non è noto l'istante di verifica
- Possono essere caratterizzati da distribuzioni dei tempi di interarrivo A(x) e dei tempi di esecuzione B(x)

- Classificazione formale:
  - i task *aperiodici* hanno deadline *soft* o sono privi di deadline ...
  - i task *sporadici* hanno o possono avere deadline relative di tipo *hard*
- Nel mondo reale? Impossibile fornire garanzie a task di tipo hard RT altrimenti...

Task con jitter

- jitter = variabilità nei tempi di rilascio e di esecuzione

- $r_i \in \{r_i^-, r_i^+\}$ *jitter nel tempo di rilascio* (task periodici)
- $e_i \in \{e_i^-, e_i^+\}$ *jitter nel tempo di esecuzione*

- Caso peggiore: $e_i=e_i^+$, $r_i=r_i^+$
- Per un'analisi di schedulabilità si può assumere
  $r_i= r_i^-$ e $WCET= e_i^+ + (r_i^+ - r_i^-)$
- Nei task periodici talvolta il *jitter nel tempo di completamento* $\{f_i^- - r_i, f_i^+ - r_i\}$ è un problema in sè e va minimizzato

Vincoli di precedenza

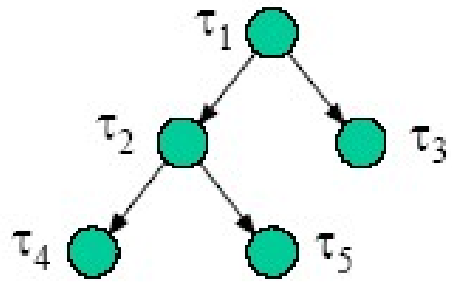
- Rappresentati con un *grafo di precedenza*
- Esprimono *dipendenze* tra dati e di controllo

- *Relazione di precedenza*: $<$ (ordinamento parziale)
- *Grafo di precedenza*: $G=(J,<)$
- Esempi di vincoli di precedenza: vincoli AND/OR

- Non tutti i vincoli di precedenza sono rappresentabili in un grafo di precedenza tra task (ad es. accesso esclusivo a dati condivisi)
- Esistono strumenti formali in grado di esprimere precedenze *e* sincronizzazioni → *Reti di Petri*

Grafo di precedenza

- Grafo orientato aciclico (DAG - Direct Acyclic Graph)



τ_1 *predecessore* di τ_4 : $\tau_1 < \tau_4$

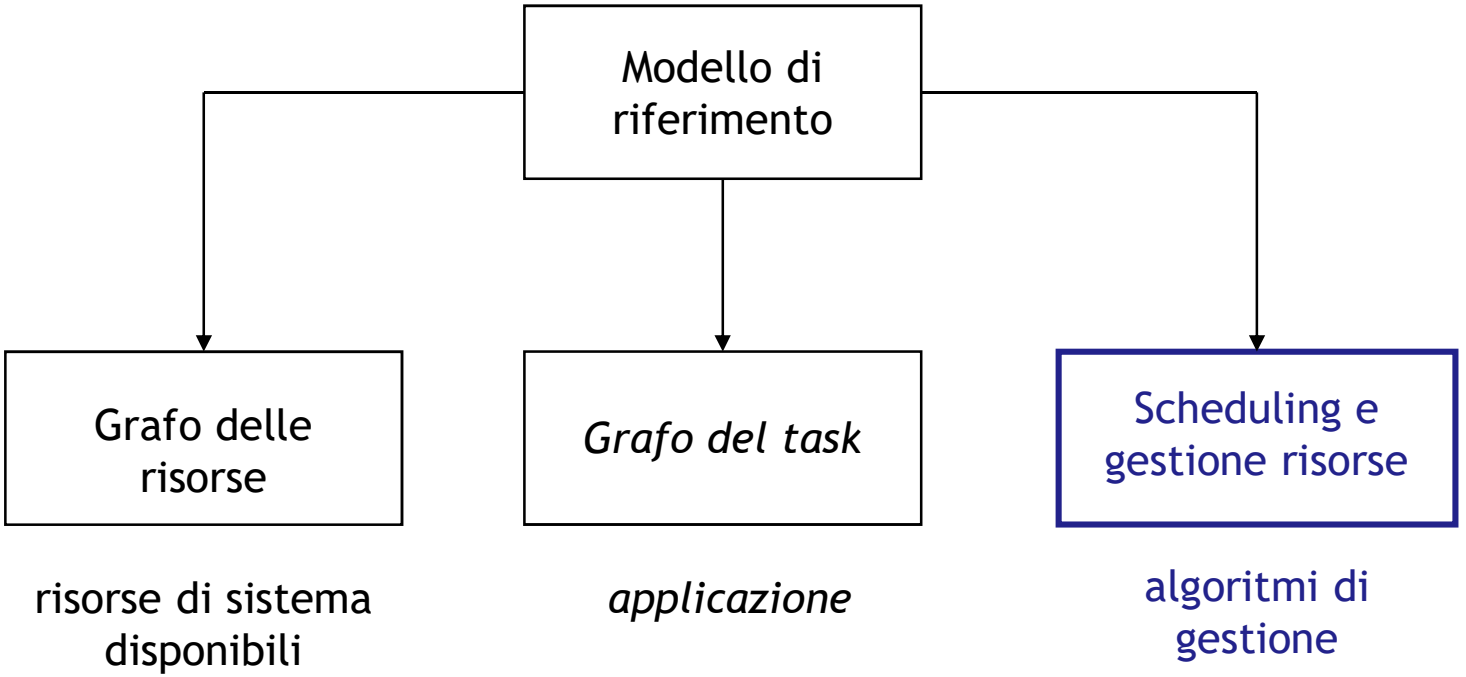
τ_1 *predecessore immediato* di τ_2 : $\tau_1 \rightarrow \tau_2$

Parametri funzionali

- *Revocabilità*
  - *revoca* o *preemption*: sospensione dell'esecuzione di un job per cedere il processore ad un job più urgente
  - la non-revocabilità è spesso legata ad una specifica risorsa; il job può essere ancora revocabile su altre risorse
  - la preemption ha un costo

- *Criticità*
  - possiamo associare un peso o valore ai job per indicarne la criticità relativa
  - schedulatori e protocolli di accesso alle risorse possono ottimizzare misure di prestazioni che tengano conto di tali pesi

Modello dell'algoritmo di gestione



Schedule ed algoritmi di scheduling

- *schedule*: assegnamento di job ai processori disponibili
- *schedule fattibile (feasible)*: 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
- *misure di prestazione*:
  - numero di job in ritardo (tardy jobs)
  - tardiness massima o media
  - tempo di risposta massimo o medio
  - makespan

Se schedule non fattibile?

- Valutare possibilità e fattibilità di una schedule con processore più veloce
- Valutare partizionamento dei task e assegnazione a core multipli
- Quanti e quali task risultano garantiti dall'algoritmo in esame?
- Modificare le caratteristiche dei task, se consentito dall'applicazione
  - Nel caso di task periodici, valutare se è possibile intervenire sui parametri di uno o più task: T_i ? C_i ? D_i ?
- Ridiscutere le specifiche con il committente
- Understand and optimize, before giving up!

Misure di prestazione per job soft RT

- La metrica più utilizzata è il *tempo di risposta medio*
- Nei sistemi RT hard/soft misti, l'obiettivo tipico è garantire il rispetto delle deadline dei job hard minimizzando il tempo di risposta medio dei job soft
- Non c'è vantaggio a completare in anticipo i job hard,
  → è possibile ritardarne l'esecuzione per migliorare la risposta ai job soft
- Altre metriche:
  - *miss rate*: percentuale di job completati in ritardo
  - *loss rate*: percentuale di job non eseguiti (ad es. scartati)
  - *invalid rate*: miss rate + loss rate