SAPIENZA
UNIVERSITÀ DI ROMA

Problemi di Scheduling

Automazione I

22/10/2014

Vincenzo Suraci

STRUTTURA DEL NUCLEO TEMATICO

- SCHEDULING A LIVELLO DI COORDINAMENTO
- CLASSIFICAZIONE DEGLI ALGORITMI DI SCHEDULING
- SCHEDULING DI TASK PERIODICI

SAPIENZA
UNIVERSITÀ DI ROMA

Corso di Laurea: INGEGNERIA
Insegnamento: AUTOMAZIONE I
Docente: DR. VINCENZO SURACI

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SCHEDULING A LIVELLO DI COORDINAMENTO

PROBLEMA DELLO SCHEDULING

Dato un sistema di automazione industriale in cui è necessario coordinare differenti task, avendo a disposizione un sistema di controllo real time dotato di una o più unità di calcolo e un insieme di risorse limitate ad accesso mutuamente esclusivo, **il problema di programmazione concorrente è risolto da un algoritmo di scheduling se esso**:

- Assicura la correttezza temporale ad una percentuale π dei task
- Assicura la correttezza logica di tutti i task

Opzionalmente si possono richiedere le ulteriori proprietà:

- Rispetta i vincoli di precedenza (PREEMPTIVE)
- Rispetta i vincoli di mutua esclusione

PROBLEMA DELLO SCHEDULING

DEFINIZIONE

Dato un insieme di task nell'ambito di un problema a livello di coordinamento di più elementi singoli, tale insieme è detto **SCHEDULABILE** se **esiste** almeno **un algoritmo di scheduling** che permetta il **rispetto di tutti i vincoli** del problema.

Non è assolutamente **detto** infatti che, **dato un insieme di task** che devono essere coordinati da un sistema di controllo real time, **qualsiasi algoritmo** di scheduling sia in grado di **soddisfare i vincoli di progetto** (correttezza temporale e logica).

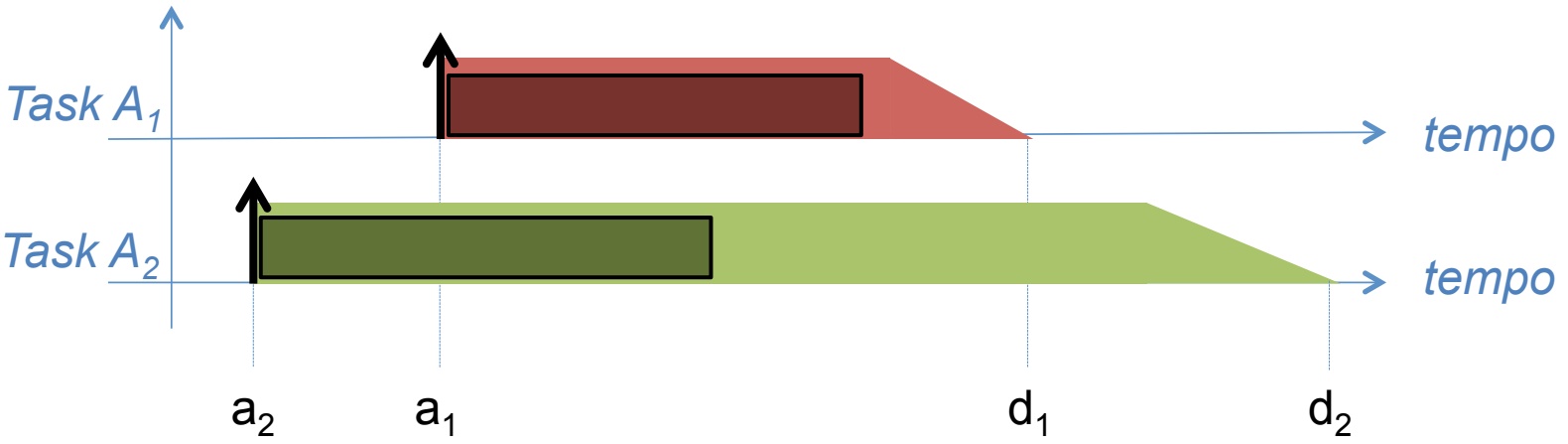
PROBLEMA DELLO SCHEDULING

ESEMPIO: sia dato un sistema monoprocessore e due task:

A_1, caratterizzato da $a_1 = 35s$, $C_1^{MIN} = 55s$ e $D_1 = 80s$ ($d_1 = 115s$)

A_2, caratterizzato da $a_2 = 10s$, $C_2^{MIN} = 60s$ e $D_1 = 145s$ ($d_2 = 155s$)

L'insieme dei due task può essere rappresentato in figura:

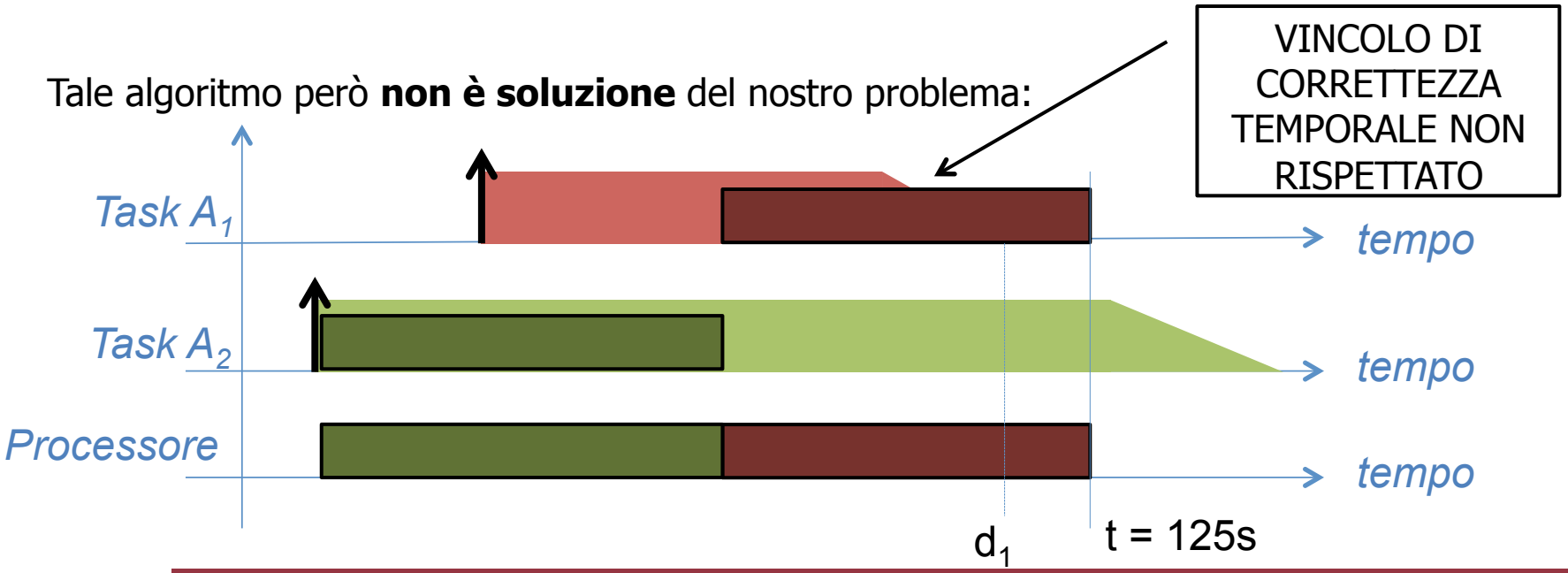


PROBLEMA DELLO SCHEDULING

Applichiamo un algoritmo di scheduling molto semplice chiamato **First In First Out (FIFO)**.

L'algoritmo FIFO è di tipo NON PREEMPTIVE: il primo task che diventa attivo viene mandato in esecuzione e terminato, il secondo task che diventa attivo viene mandato in esecuzione e terminato e così via.

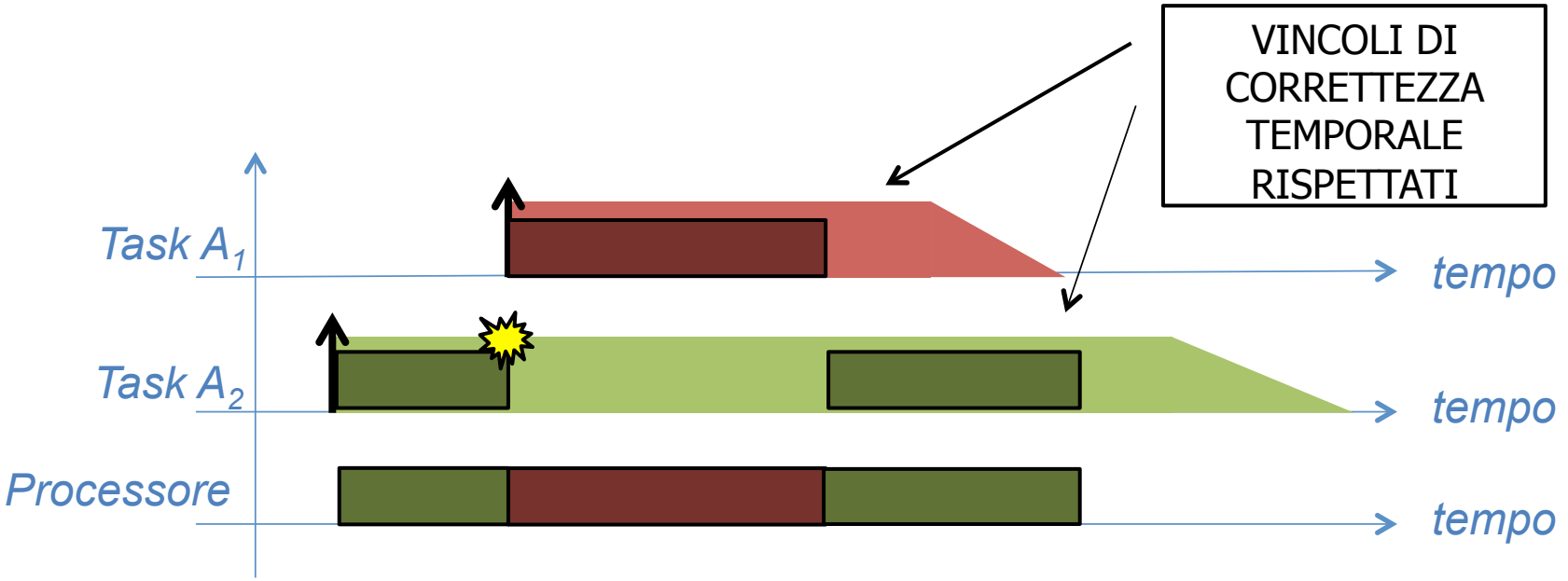
Tale algoritmo però **non è soluzione** del nostro problema:



PROBLEMA DELLO SCHEDULING

Se modifichiamo l'algoritmo di scheduling **FIFO** aggiungendo il vincolo di priorità del primo task rispetto al secondo, per questo insieme di task avremo una soluzione al problema.

Il PREEMPTIVE FIFO **è soluzione** del nostro problema:



COMPLESSITÀ

Sfortunatamente il problema della definizione di **un algoritmo di scheduling non è computazionalmente risolvibile in TEMPO POLINOMIALE rispetto al numero di Task**.

Nel 1975 è stato dimostrato che **il problema generale dello scheduling è di tipo NP-HARD**, ovvero non esistono algoritmi di scheduling di complessità computazionale temporale polinomiale nel numero di task che siano in grado di definire uno scheduling in grado di garantire tutti i vincoli del problema.

Quello che a noi interessa in **AUTOMAZIONE** non è l'aspetto teorico, di per sé estremamente interessante, ma le ricadute pratiche della teoria in scenari reali.

Fortunatamente nei sistemi di controllo finalizzati al coordinamento di elementi singoli, si possono fare notevoli **ipotesi semplificative** che portano alla definizione di algoritmi di scheduling hard e soft real time anche molto efficienti.

SAPIENZA
UNIVERSITÀ DI ROMA

Corso di Laurea: INGEGNERIA
Insegnamento: AUTOMAZIONE I
Docente: DR. VINCENZO SURACI

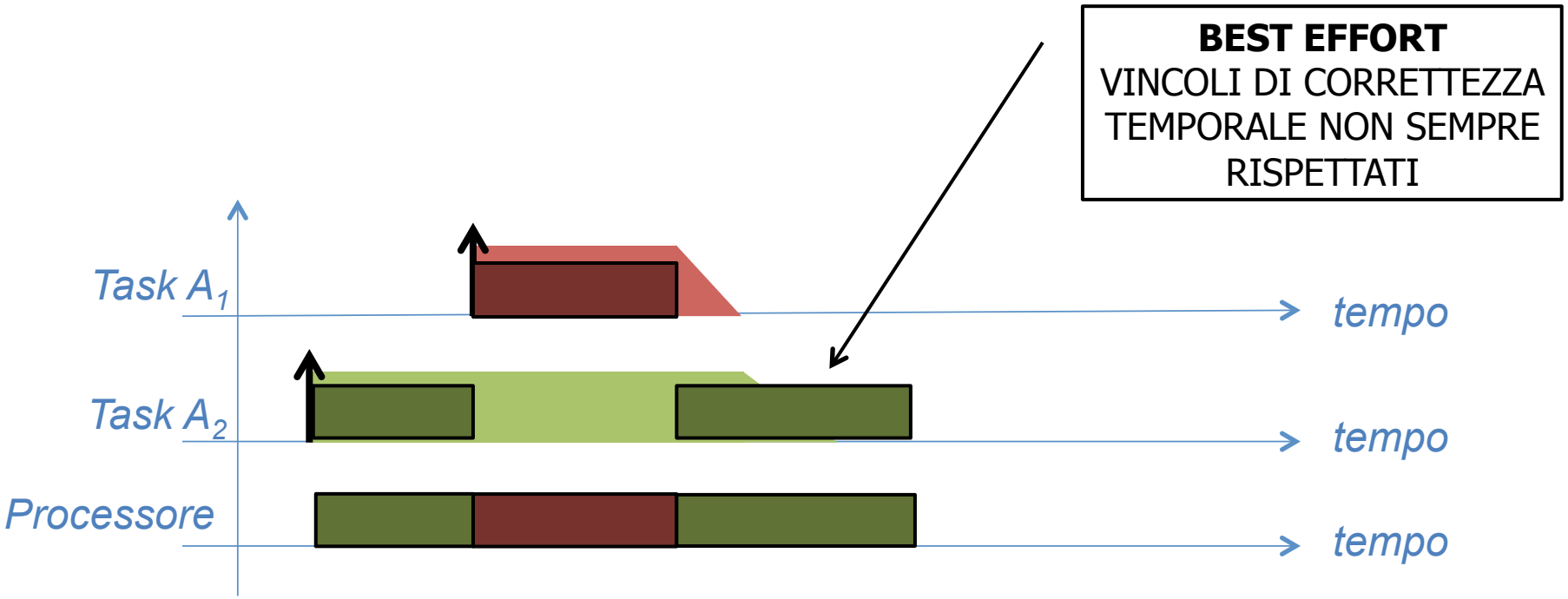
DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

CLASSIFICAZIONE DEGLI ALGORITMI DI SCHEDULING

CLASSIFICAZIONE

Dato un algoritmo di scheduling diremo che esso è:

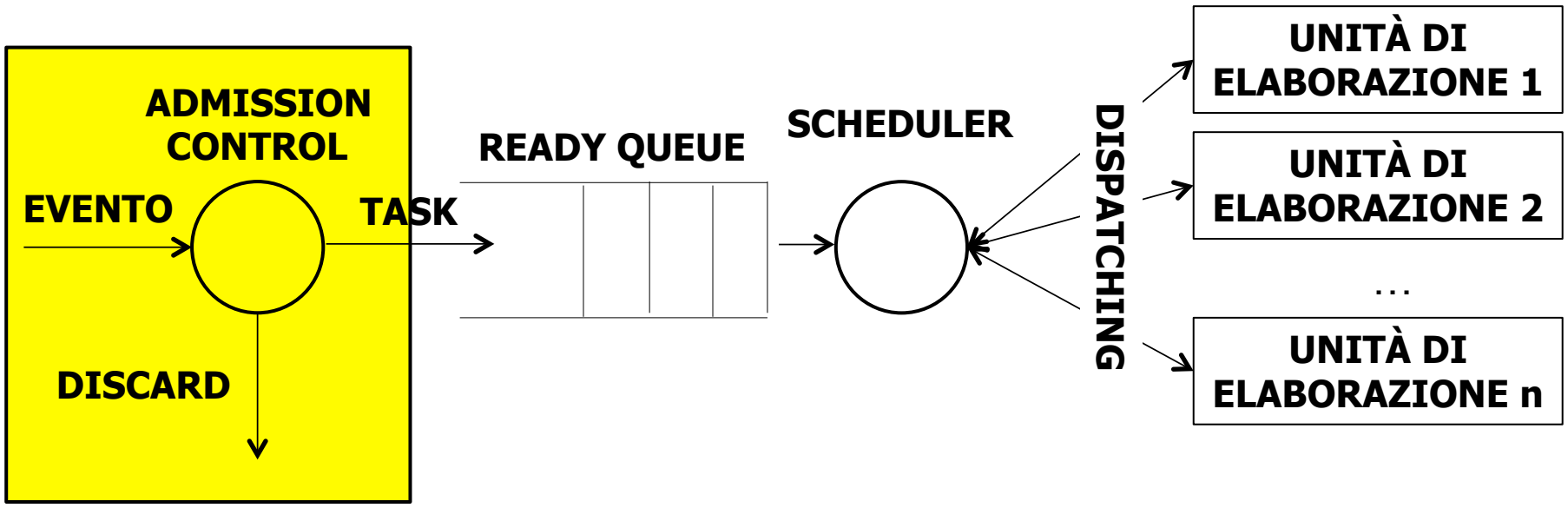
- **GUARANTEED** se esso ha $\pi = 1$ (HARD REAL TIME);
- **BEST EFFORT** se esso ha $\pi < 1$ (SOFT REAL TIME);



CLASSIFICAZIONE - SCHEDULING GUARANTEED

Un algoritmo di scheduling **GUARANTEED** deve rispettare il **vincolo di correttezza temporale per ogni task**.

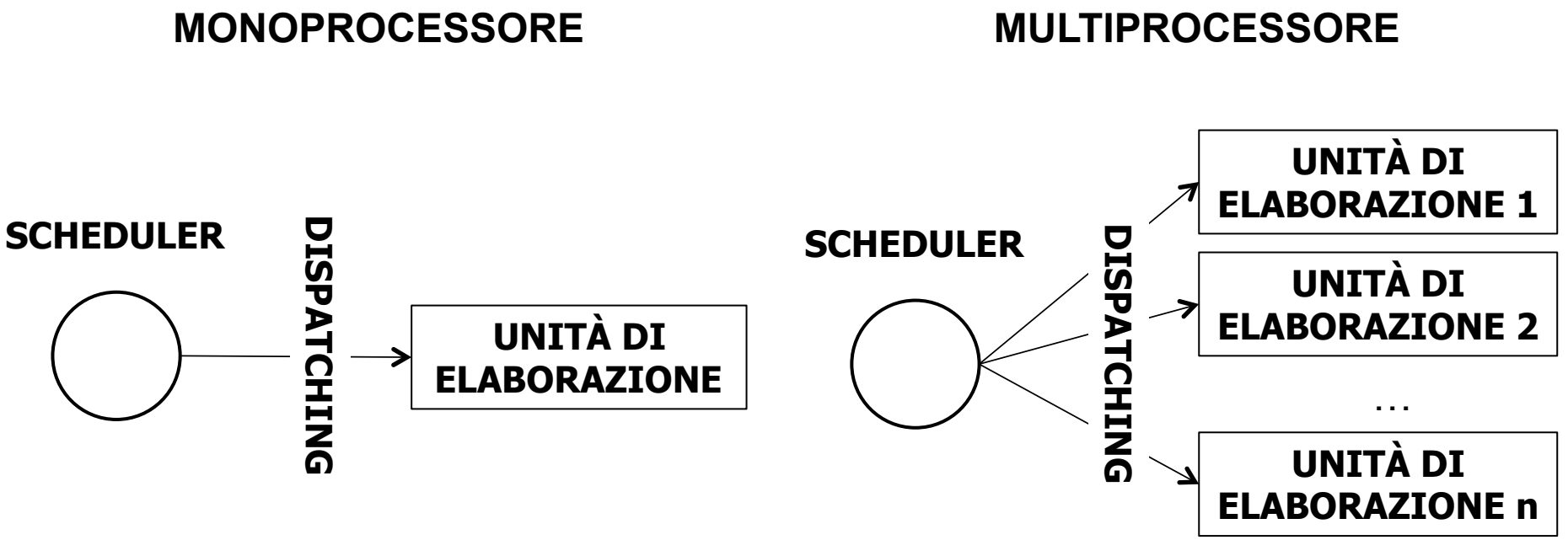
Questi algoritmi si basano su un «test di garanzia» chiamato **ADMISSION CONTROL**. Se il test assicura l'esecuzione del task allo scadere della deadline, il task viene attivato (e spostato nella READY QUEUE), altrimenti viene scartato (**DISCARD**).



CLASSIFICAZIONE

Dato un algoritmo di scheduling diremo che esso è:

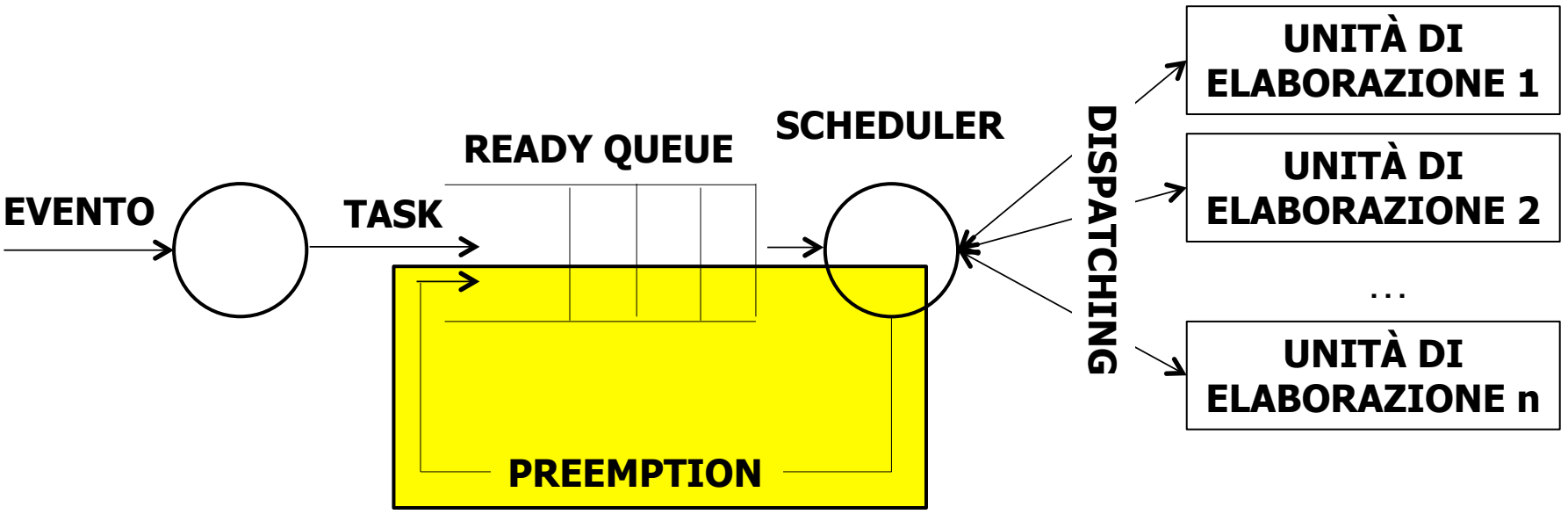
- **MONOPROCESSORE** se esso fa dispatching di task su un'unica unità di calcolo
- **MULTIPROCESSORE** se esso fa dispatching di task su più di una unità di calcolo



CLASSIFICAZIONE

Dato un algoritmo di scheduling diremo che esso è:

- **PREEMPTIVE** se esso è in grado di interrompere l'esecuzione in una delle unità di calcolo di un task a minor priorità a favore dell'esecuzione di un task a maggior priorità
- **NON PREEMPTIVE** se esso non è in grado di interrompe l'esecuzione di un task quando esso è stato inviato ad una delle unità di calcolo



CLASSIFICAZIONE

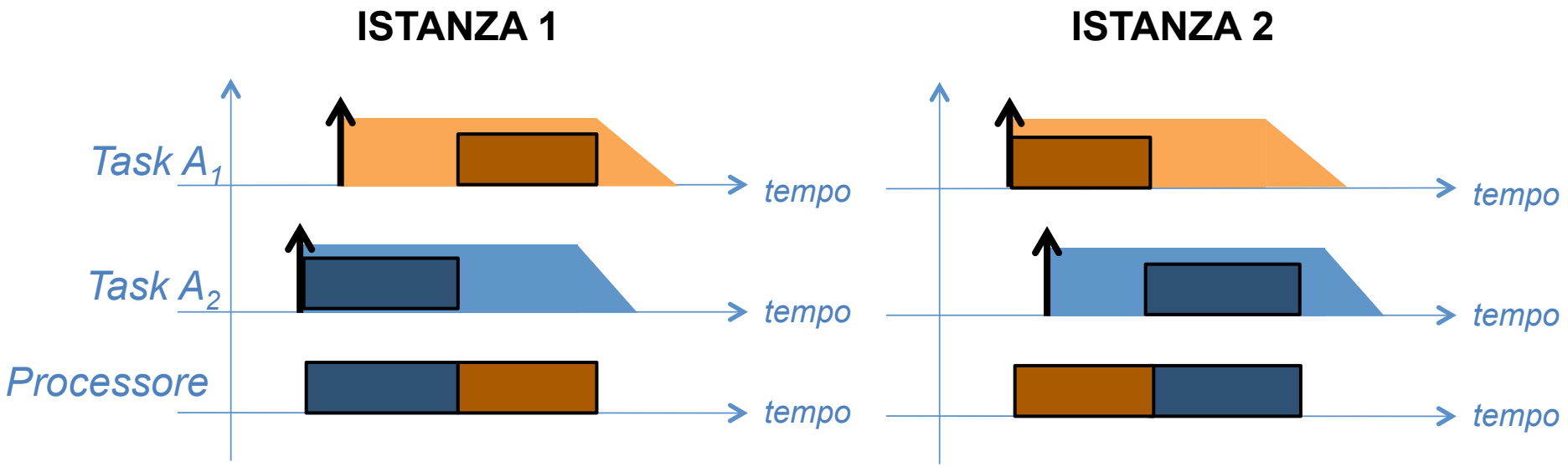
Dato un algoritmo di scheduling diremo che esso è:

- **OFF-LINE** se tutte le decisioni riguardanti lo scheduling sono prese **PRIMA** dell'attivazione dei task e quindi se **la sequenza di dispatching è nota a priori**

- **ON-LINE** se le decisioni riguardanti lo scheduling dipendono dall'ordine di attivazione dei task

CLASSIFICAZIONE

L'algoritmo **FIFO è uno scheduling ON-LINE** in quanto **dipende dall'ordine di attivazione dei task**.



CLASSIFICAZIONE

Dato un algoritmo di scheduling diremo che esso è:

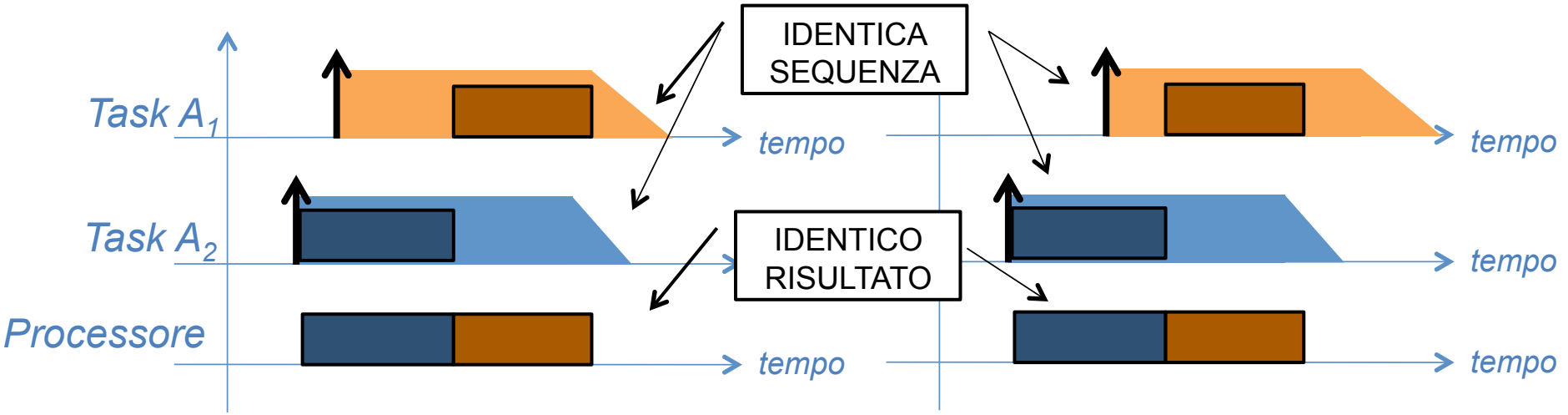
- **STATICO** se le **regole di dispatching** sono definite a partire da **PARAMETRI CHE NON VARIANO** durante l'esecuzione dell'algoritmo

- **DINAMICO** se le **regole di dispatching** sono definite a partire da **PARAMETRI CHE POSSONO VARIARE** durante l'esecuzione dell'algoritmo

Un esempio di **parametro** è la **priorità** associata ad un task.

CLASSIFICAZIONE

L'algoritmo **FIFO è uno scheduling ON-LINE STATICO**, in quanto, fissata la sequenza di attivazione dei task, la priorità dei task è nota a priori.



CLASSIFICAZIONE - CONSIDERAZIONI

PROPOSIZIONE

Un algoritmo di scheduling OFF-LINE è STATICO.

DIMOSTRAZIONE

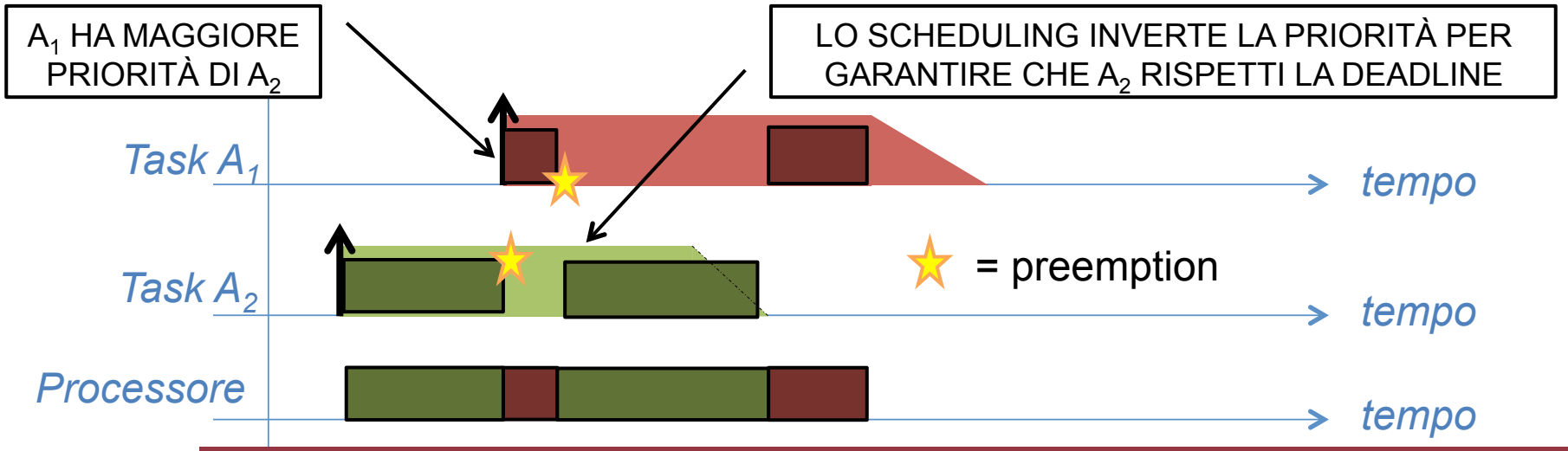
- Essendo l'algoritmo di scheduling OFF-LINE, **la sequenza è nota a priori** e non viene mai modificata durante l'esecuzione dell'algoritmo.
- Necessariamente **le regole di dispatching** (che determinano la sequenza dei task mandati in esecuzione) sono dipendenti da **parametri statici, che NON VARIANO** durante l'esecuzione dell'algoritmo.

CLASSIFICAZIONE - CONSIDERAZIONI

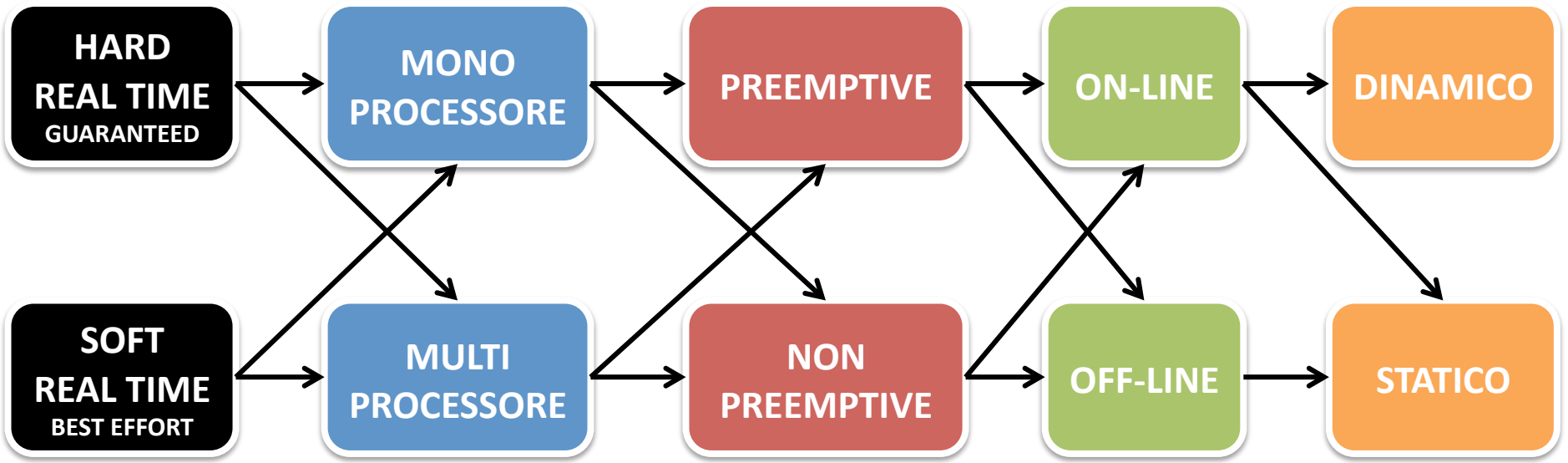
In generale (ma non sempre) un algoritmo di scheduling **GUARANTEED è ON-LINE e DINAMICO**. Infatti per garantire il vincolo di correttezza temporale per ogni task ($\pi = 1$) difficilmente la sequenza dei task sarà nota a priori (caso di scheduling OFF-LINE) e spesso **le regole di dispatching** dei task **possono dipendere da parametri** (ad esempio il livello di priorità) **che variano nel tempo** (scheduling DINAMICO).

ESEMPIO

Scheduling MONOPROCESSORE con priorità variabile.



CLASSIFICAZIONE – DIAGRAMMA SINOTTICO



SAPIENZA
UNIVERSITÀ DI ROMA

Corso di Laurea: INGEGNERIA
Insegnamento: AUTOMAZIONE I
Docente: DR. VINCENZO SURACI

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

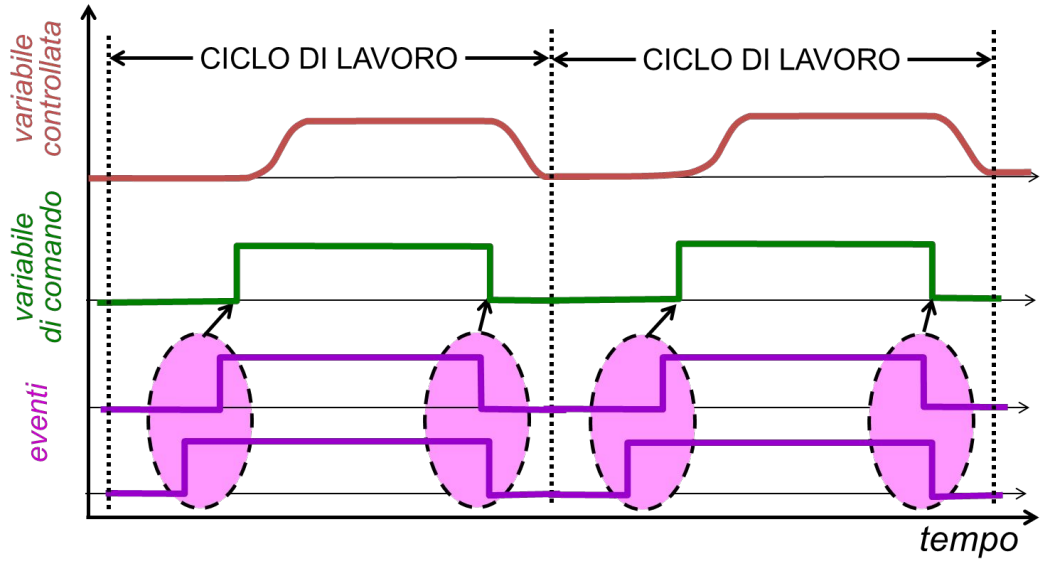
SCHEDULING DI TASK PERIODICI

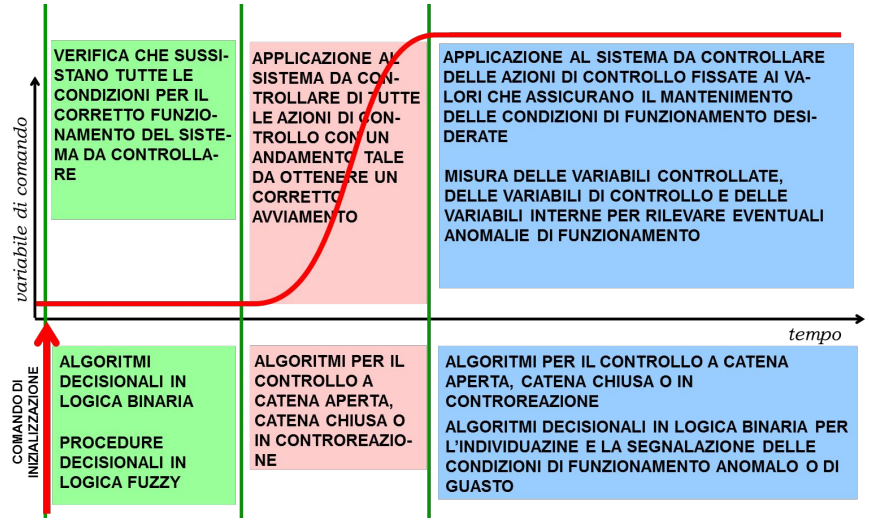
TASK PERIODICI NELL'AUTOMAZIONE INDUSTRIALE

Nel contesto dell'**AUTOMAZIONE** industriale ha molto senso considerare lo scenario di coordinare **task** che vengano **attivati periodicamente**.

Si pensi ad esempio ai task che compongono un **CICLO DI LAVORO** in un impianto ad **EVENTI PROGRAMMATI** che devono essere periodicamente ripetuti.

Si pensi anche ai task che periodicamente devono essere sequenzializzati per mantenere costanti le condizioni di funzionamento di un impianto di produzione **DI TIPO CONTINUO**.



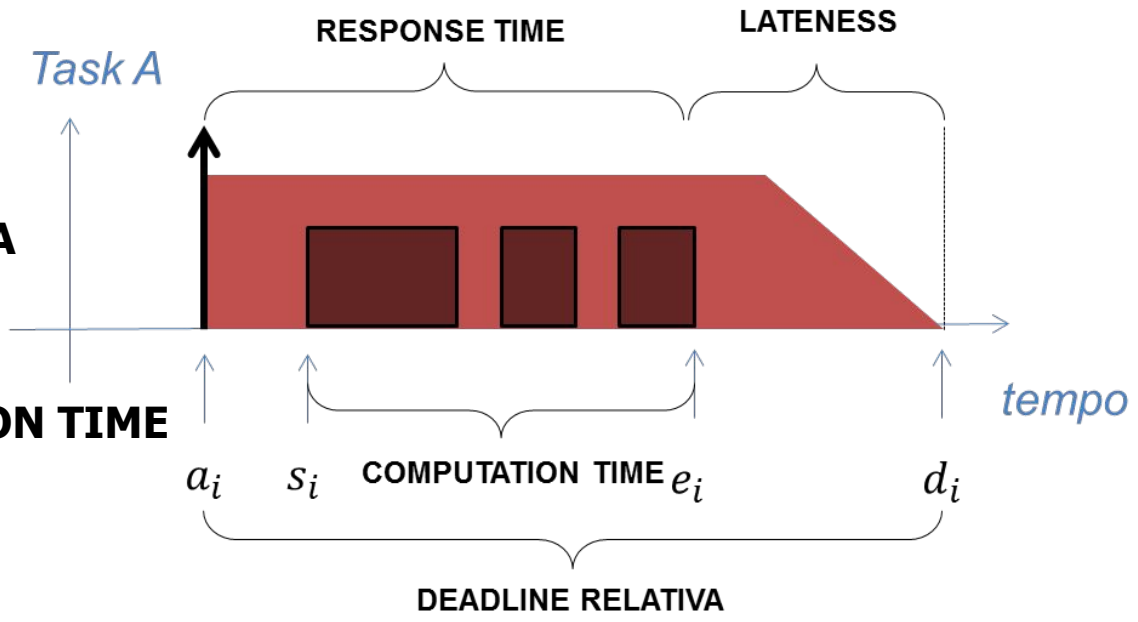


PERIODO DI ATTIVAZIONE

DEFINIZIONE

Dato un task (A) si definisce **ESECUZIONE** k-esima (**ISTANZA** k-esima) del task A, l'elemento caratterizzato dai seguenti parametri caratteristici:

- a_i (k) **ACTIVATION TIME**
- s_i (k) **START TIME**
- e_i (k) **END TIME**
- d_i (k) **DEADLINE ASSOLUTA**



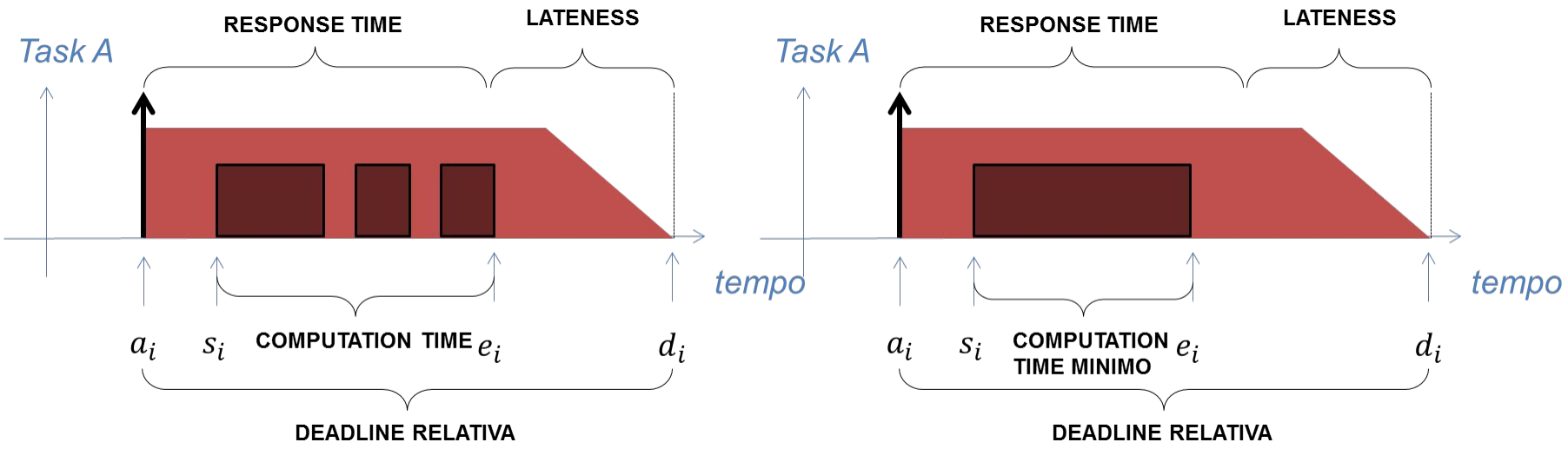
- C_i(k) $\triangleq C_i^{MIN}$(k) **COMPUTATION TIME**
- D_i(k) **DEADLINE RELATIVA**
- R_i(k) **RESPONSE TIME**
- L_i(k) **LATENESS**

PERIODO DI ATTIVAZIONE

OSSERVAZIONE

Da ora in avanti faremo riferimento al Computation Time $C_i(k)$, sempre come al valore relativo al minimo Computation Time [$C_i^{MIN}(k)$].

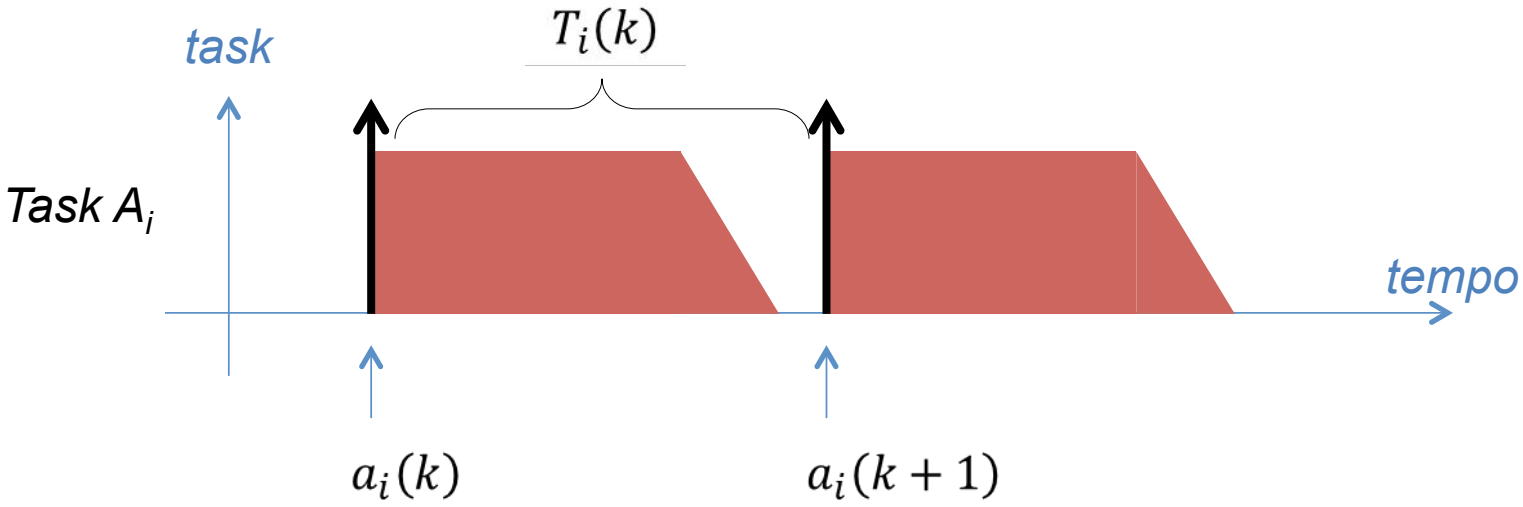
$C_i(k) = C_i^{MIN}(k)$ **COMPUTATION TIME**



PERIODO DI ATTIVAZIONE

DEFINIZIONE

Dato un insieme finito di n task (A_1, A_2, ..., A_n) si definisce **PERIODO DI ATTIVAZIONE** $T_i(k)$ della *k*-esima esecuzione del task A_i l'intervallo finito di tempo che intercorre dall'istante di attivazione del task A_i nella *k*-esima esecuzione, all'istante di attivazione dello stesso task A_i nella successiva (*k+1*)-esima esecuzione.

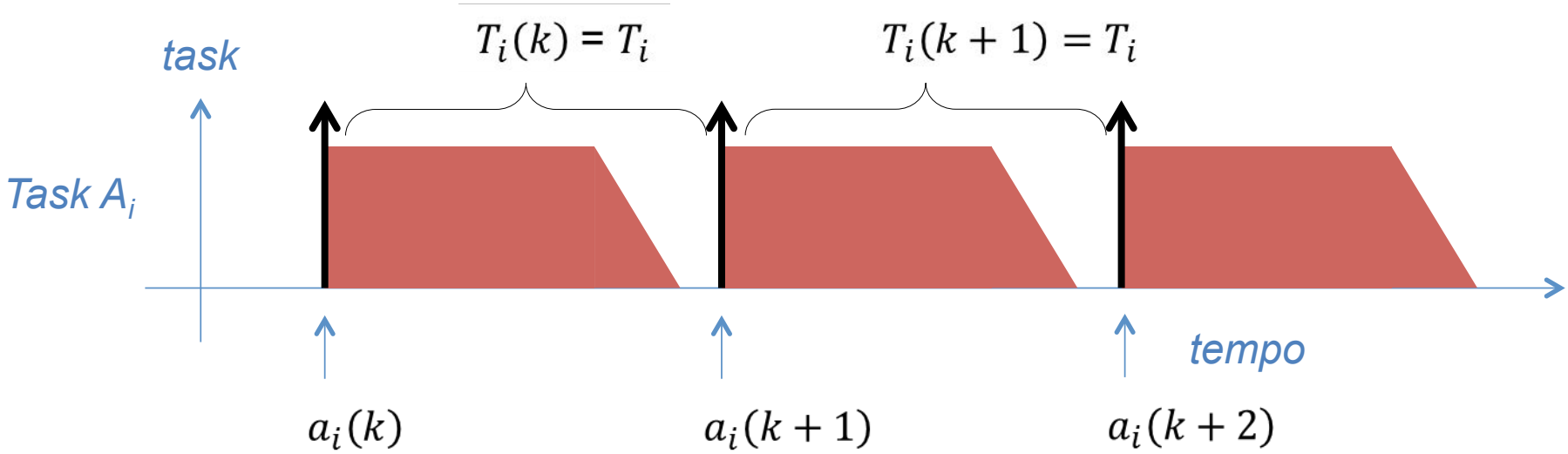


$$a_i(k+1) - a_i(k) = T_i(k) \;\; \forall\, i \in [1,..,n], \forall\, k \in \mathbb{N}$$

TASK PERIODICI

DEFINIZIONE

Dicesi **TASK PERIODICO** (o **PROCESSO PERIODICO**), un task A_i in cui il **PERIODO DI ATTIVAZIONE** (T_i) sia costante ad ogni esecuzione.



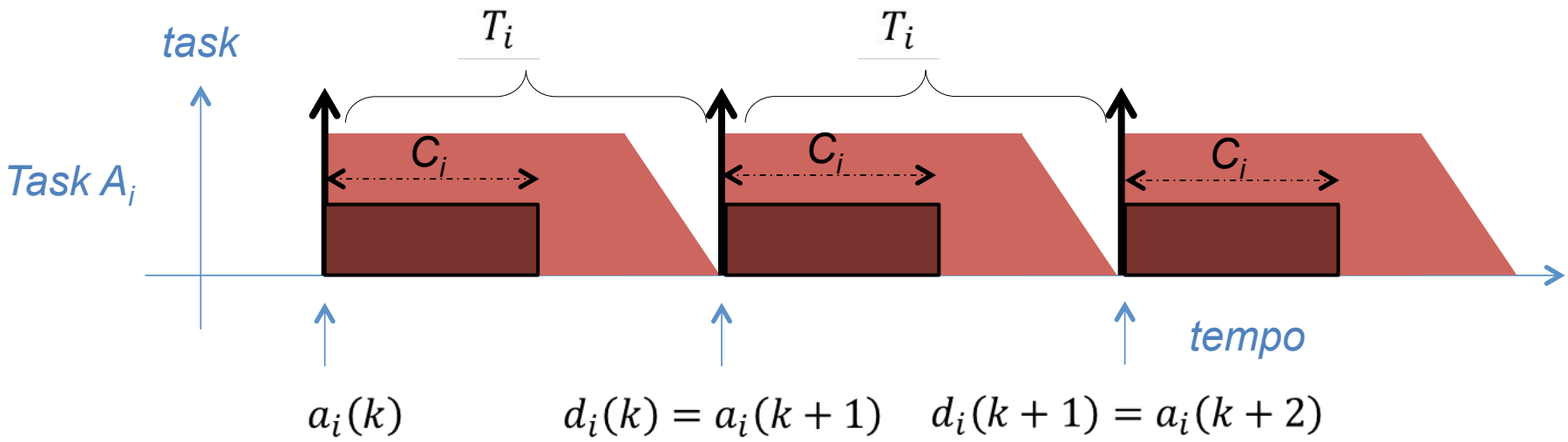
$$T_i(k) = T_i \quad \forall k \in \mathbb{N}$$

TASK PERIODICI

IPOTESI

Dato un insieme di n TASK PERIODICI (A_1, A_2, …, A_n) ipotizziamo che:

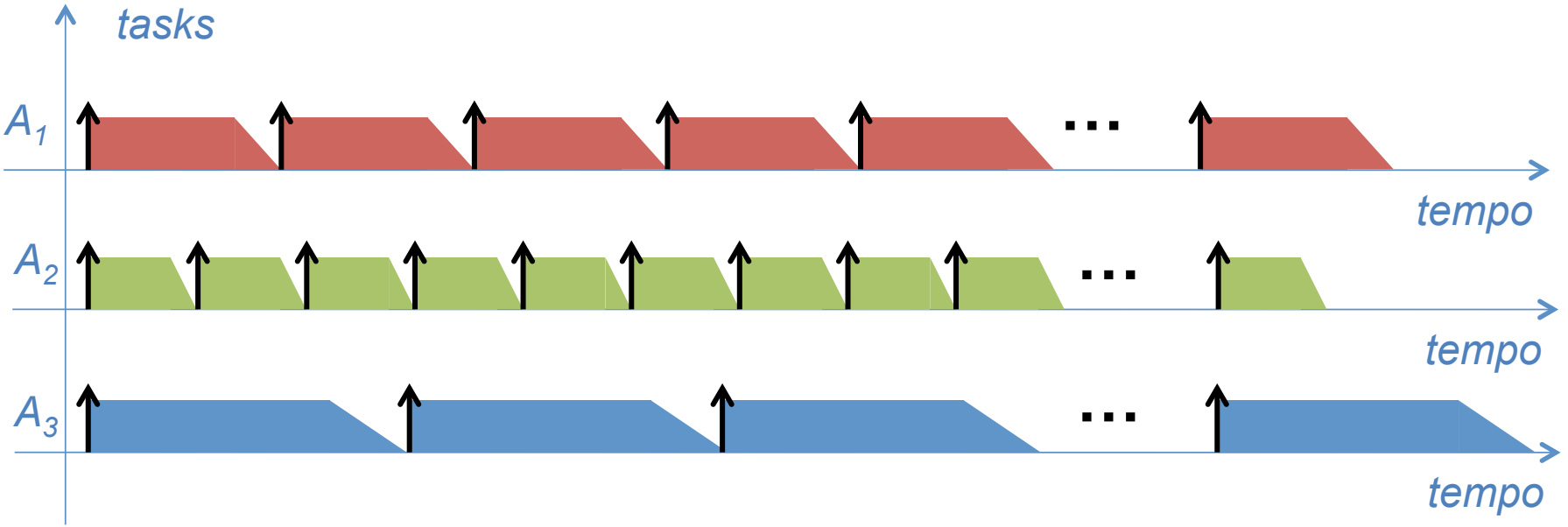
1. La deadline relativa del task A_i coincida con il periodo di attivazione T_i ad ogni esecuzione;
2. Il computation time C_i di ogni task sia costante ad ogni esecuzione;
3. I task non condividano risorse mutuamente esclusive.



TASK PERIODICI

ESEMPIO

Dato un insieme di n = 3 TASK PERIODICI che verifichino le ipotesi 1-3, graficamente avremo:



DEFINIZIONE DEL PROBLEMA

PROBLEMA DELLO SCHEDULING DI TASK PERIODICI

Dato un insieme di n task periodici (A_1, A_2, …, A_n) che verifichino le seguenti ipotesi:

1. La deadline relativa del task A_i coincida con il periodo di attivazione T_i ad ogni esecuzione;
2. Il computation time C_i di ogni task sia costante ad ogni esecuzione;
3. I task non condividano risorse mutuamente esclusive.

Identificare un algoritmo di scheduling **MONOPROCESSORE GUARANTEED**.

OSSERVAZIONI

- Il problema dello scheduling di task periodici è costituito da:

1. **REQUISITI DI SISTEMA**: dati dal numero n di task e dagli n periodi di attivazione, ovvero dalla $(n, T_1, T_2, …, T_n)$;
2. **VINCOLI DI SISTEMA**: dati dagli n valori di computation time $(C_1, C_2, …, C_n)$;

- I primi sono in generale dati fissi, i secondi dipendono molto da come il sistema è stato implementato (ad es. dalla velocità del processore).

FATTORE DI UTILIZZAZIONE

DEFINIZIONE

Dato un problema di scheduling di task periodici, si definisce **FATTORE DI UTILIZZAZIONE** il coefficiente definito nel modo seguente:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

FATTORE DI UTILIZZAZIONE

OSSERVAZIONI

- Il fattore di utilizzazione è un numero **reale non negativo**;
- Il fattore di utilizzazione rappresenta in maniera percentuale il **tempo di utilizzazione della risorsa computazionale**;
- Se il fattore di utilizzazione è maggiore di 1, il problema dello scheduling di task periodici **NON AMMETTE SOLUZIONE.**

ESEMPIO

PROBLEMA

Dato un problema di scheduling costituito da n = 2 task periodici (A_1, A_2) tali che:

- T_1 = 8 time unit (t.u.) e C_1 = 2 t.u.;
- T_2 = 12 t.u. e C_2 = 8 t.u.;

Stabilire se esso è schedulabile.

SOLUZIONE

Il fattore di utilizzazione del problema di scheduling è dato da:

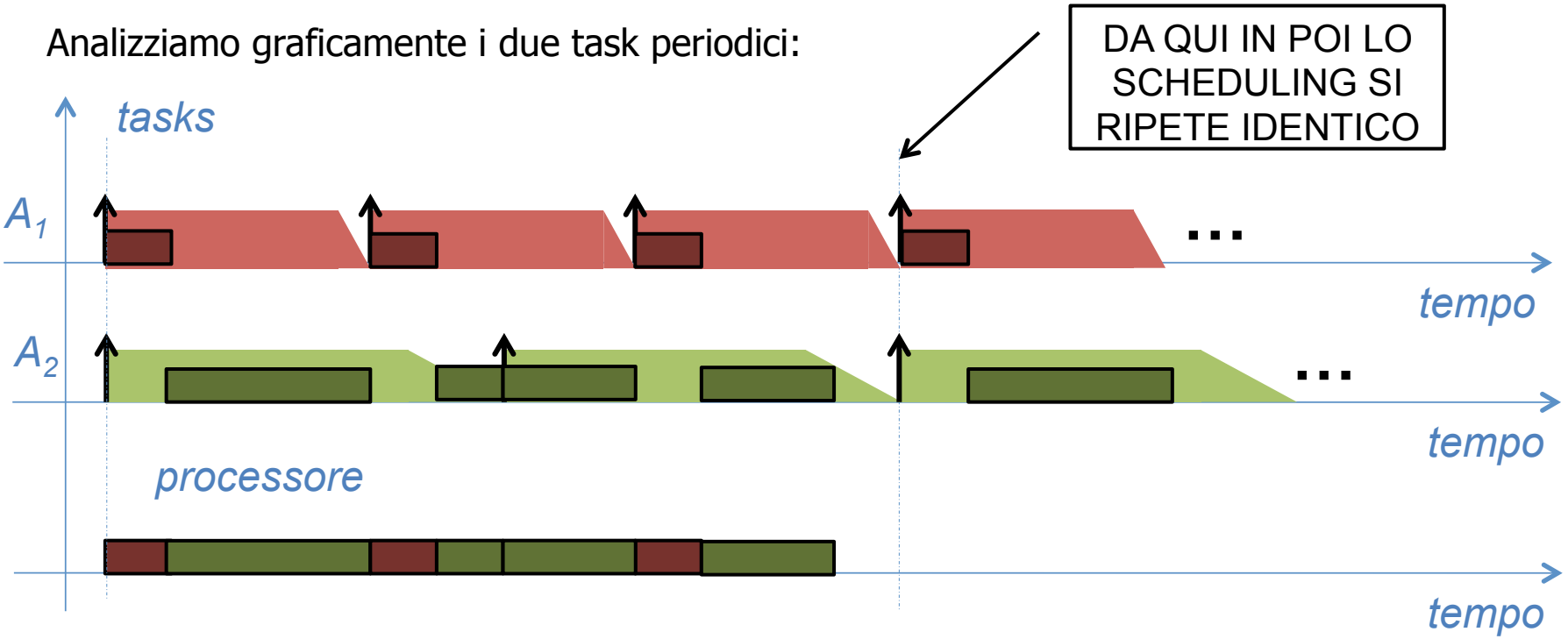
$$U = \sum_{i=1}^{2} \frac{C_i}{T_i} = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{2}{8} + \frac{8}{12} = \frac{6+16}{24} = \frac{11}{12} \approx 0{,}917 < 1$$

Essendo il fattore di utilizzazione minore di 1, si può escludere che esso sia NON SCHEDULABILE. Per essere certi che esso sia schedulabile, però, è necessario individuare almeno un algoritmo di scheduling che sia in grado di risolvere il problema dato.

ESEMPIO cont'd

Prendiamo in considerazione un algoritmo di scheduling PREEMPTIVE che assegna priorità maggiore al task caratterizzato dal periodo di attivazione minore (nel nostro caso, il task A_1).

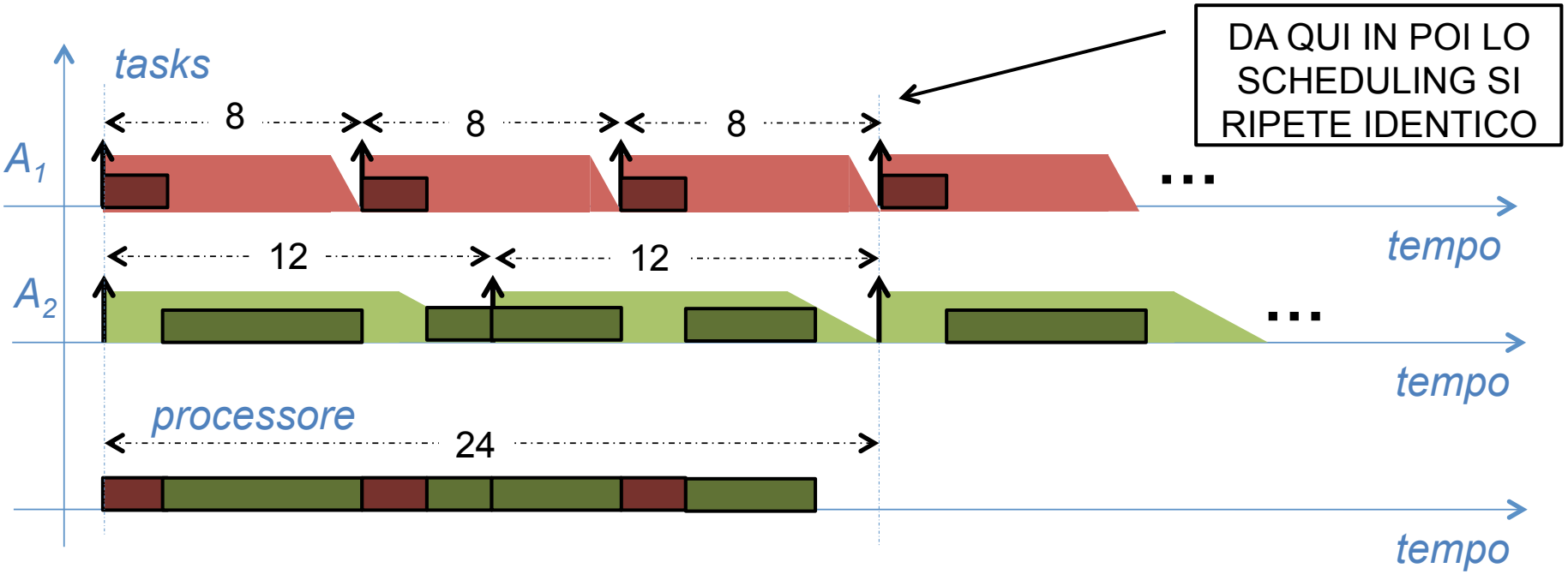
Analizziamo graficamente i due task periodici:



ESEMPIO cont'd

Possiamo notare che:

- Abbiamo trovato un algoritmo di scheduling che **RISOLVE** il problema;
- Lo scheduling si ripete ogni **minimo comune multiplo** dei periodi di attivazione;
- Il processore **NON È COMPLETAMENTE UTILIZZATO**.



FATTORE DI UTILIZZAZIONE MASSIMO

Ci si potrebbe chiedere, **dato l'algoritmo di scheduling** definito nell'esempio, quanto possiamo «**spingere al limite**» **il problema di scheduling di task periodici** prima che l'algoritmo **non sia più soluzione** del problema assegnato.

Scelto un **algoritmo di scheduling F** applicato ad **un problema di scheduling di task periodici** dove siano fissati i REQUISITI e variabili i VINCOLI, ovvero dove:

- Sia fissato il numero n di task periodici;
- Siano fissati i periodi di attivazione T_i degli n task;
- Siano scelti i computation time C_i degli n task in maniera tale che il problema risulti sempre potenzialmente ammissibile (ovvero $U \leq 1$)

è possibile calcolare, **al variare dei computation time C_i degli n task**, il **massimo valore del fattore di utilizzazione** U_{max} oltre il quale la **schedulazione** dell'insieme di task dati **non è più fattibile** con tale algoritmo?

FATTORE DI UTILIZZAZIONE MASSIMO

Dato un **algoritmo di scheduling F** applicato ad un problema di scheduling di task periodici (n, T_1 , T_2 , …, T_n) il **massimo valore del fattore di utilizzazione** U_{max} sarà:

$$U_{max}(F, n, T_1, T_2, \ldots, T_n) = \max_{\substack{C_i \in \mathbb{R}^+ \\ F\ soluzione}} U(n, T_1, T_2, \ldots, T_n) = \max_{\substack{C_i \in \mathbb{R}^+ \\ F\ soluzione}} \sum_{i=1}^{n} \frac{C_i}{T_i}$$

È bene notare che il limite $\mathbf{U_{max}}$ **dipende** sia **dall'algoritmo di scheduling F** che **dall'insieme di task periodici** rispetto a cui viene calcolato (ovvero dai parametri n, T_1 , T_2 , …, T_n).

PROCESSORE COMPLETAMENTE UTILIZZATO

DEFINIZIONE

Dato un algoritmo di scheduling F ed un insieme di task periodici, il processore viene detto **COMPLETAMENTE UTILIZZATO** se la schedulazione è fattibile e se un aumento comunque piccolo di uno dei computation time C_i rende la schedulazione impossibile.

PROPOSIZIONE

Dato un algoritmo di scheduling F ed un insieme di task periodici, se il processore è **COMPLETAMENTE UTILIZZATO** allora $U = U_{max}$.

DIMOSTRAZIONE

Per definizione di massimo fattore di utilizzazione non possiamo avere che $U > U_{max}$.

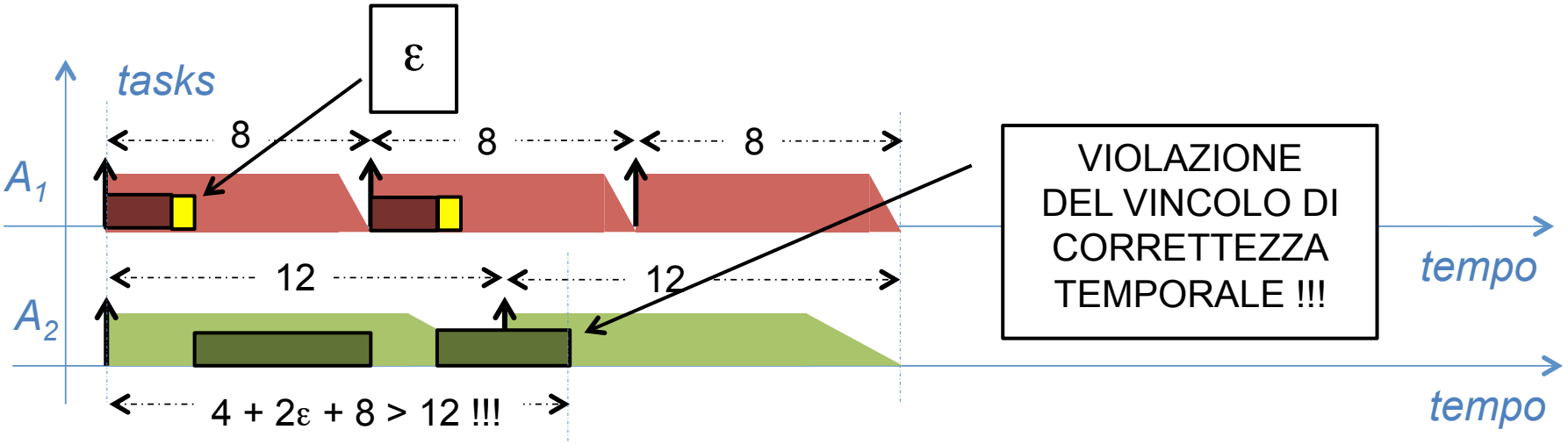
Ipotizziamo per assurdo che $U < U_{max}$. Questo implica che esiste un $\varepsilon>0$ sufficientemente piccolo tale da avere un altro fattore di utilizzazione (U^*) dell'insieme dato di task periodici tale che $U < U^* < U_{max}$ che sia schedulabile dall'algoritmo dato. Ma questo contraddice l'ipotesi che il processore fosse completamente utilizzato!

ESEMPIO cont'd

Se proviamo ad aumentare il fattore di utilizzazione U, ad esempio aumentando di una quantità $\varepsilon>0$ il computation time del task A_1, e scegliendone un valore piccolo a piacere tale da mantenere U < 1:

$C_1 = 2 + \varepsilon$ t.u.;

L'algoritmo non è più in grado di garantire il vincolo di correttezza temporale.

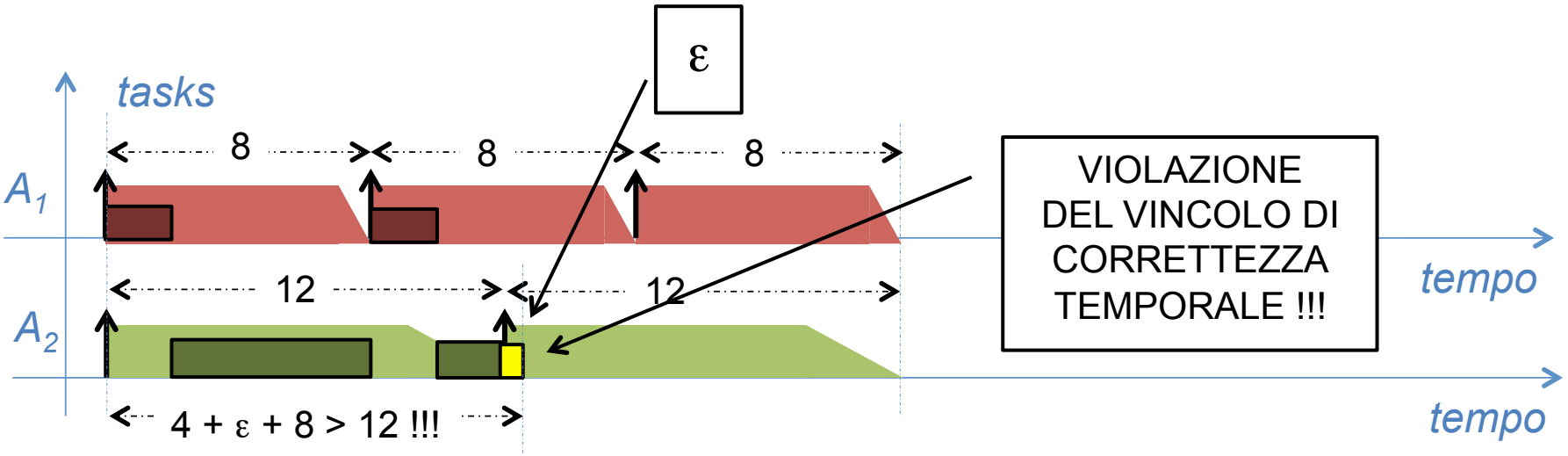


ESEMPIO cont'd

Se proviamo ad aumentare il fattore di utilizzazione U, ad esempio aumentando di una quantità $\varepsilon>0$ il computation time del task A_2, e scegliendone un valore piccolo a piacere tale da mantenere U < 1:

$C_2 = 8 + \varepsilon$ t.u.;

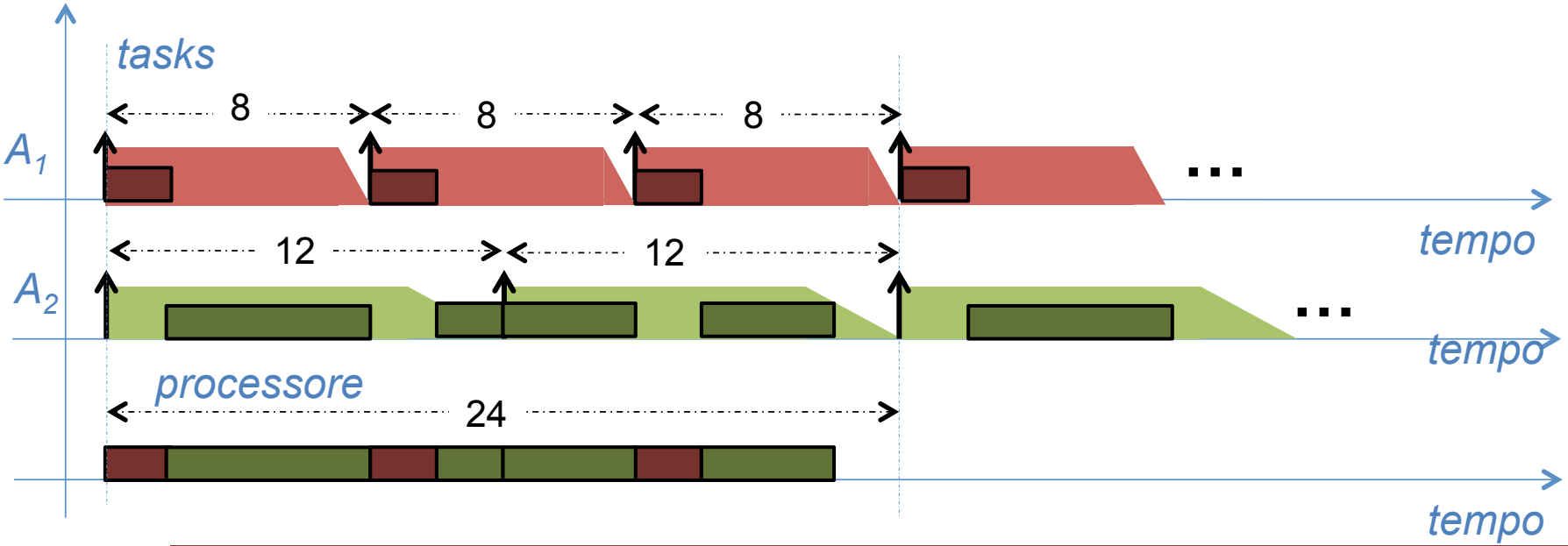
L'algoritmo non è più in grado di garantire il vincolo di correttezza temporale.



ESEMPIO cont'd

Dato il problema di scheduling di task periodici (n=2, $T_1 = 8$, $T_2 = 12$) e l'algoritmo di scheduling F = «PREEMPTIVE che assegna priorità maggiore al task caratterizzato dal periodo di attivazione minore», possiamo affermare che il processore è COMPLETAMENTE UTILIZZATO e quindi:

$$U_{max} = \frac{11}{12} \approx 0{,}917 < 1$$



LIMITE SUPERIORE MINIMO DEL FATTORE DI UTILIZZAZIONE

DEFINIZIONE

Si definisce **LIMITE SUPERIORE MINIMO DEL FATTORE DI UTILIZZAZIONE** $U_{lsm}(F)$ di un algoritmo di scheduling F, il minimo tra tutti i massimi fattori di utilizzazione calcolati per ogni insieme possibile di task periodici (n, T_1 , T_2 , …, T_n).

$$U_{lsm}(F) = \min_{\substack{n \in \mathbb{N} \\ T_i \in \mathbb{R}^+}} \left(\max_{\substack{C_i \in \mathbb{R}^+ \\ F\ soluzione}} \sum_{i=1}^{n} \frac{C_i}{T_i} \right)$$

OSSERVAZIONE

Il parametro $U_{lsm}(F)$ è molto importante in quanto caratterizza la bontà di un algoritmo di scheduling, indipendentemente dal problema di scheduling di task periodici assegnato, in quanto ne **descrive il carico computazionale massimo che può essere sicuramente gestito**.

LIMITE SUPERIORE MINIMO DEL FATTORE DI UTILIZZAZIONE

PROPOSIZIONE (SENZA DIMOSTRAZIONE)

Un insieme di task periodici ($n, T_1, T_2, \ldots, T_n, C_1, C_2, \ldots, C_n$) è sicuramente schedulabile tramite un algoritmo F se risulta:

$$U \leq U_{lsm}(F)$$

BIBLIOGRAFIA

Sezione 2.4 (pagg. 46-53)

TITOLO

Sistemi di automazione industriale

Architetture e controllo

AUTORI

Claudio Bonivento

Luca Gentili

Andrea Paoli

EDITORE

McGraw-Hill