SAPIENZA
UNIVERSITÀ DI ROMA

Algoritmi di scheduling
- Parte 2

Automazione

21/10/2015

Vincenzo Suraci

STRUTTURA DEL NUCLEO TEMATICO

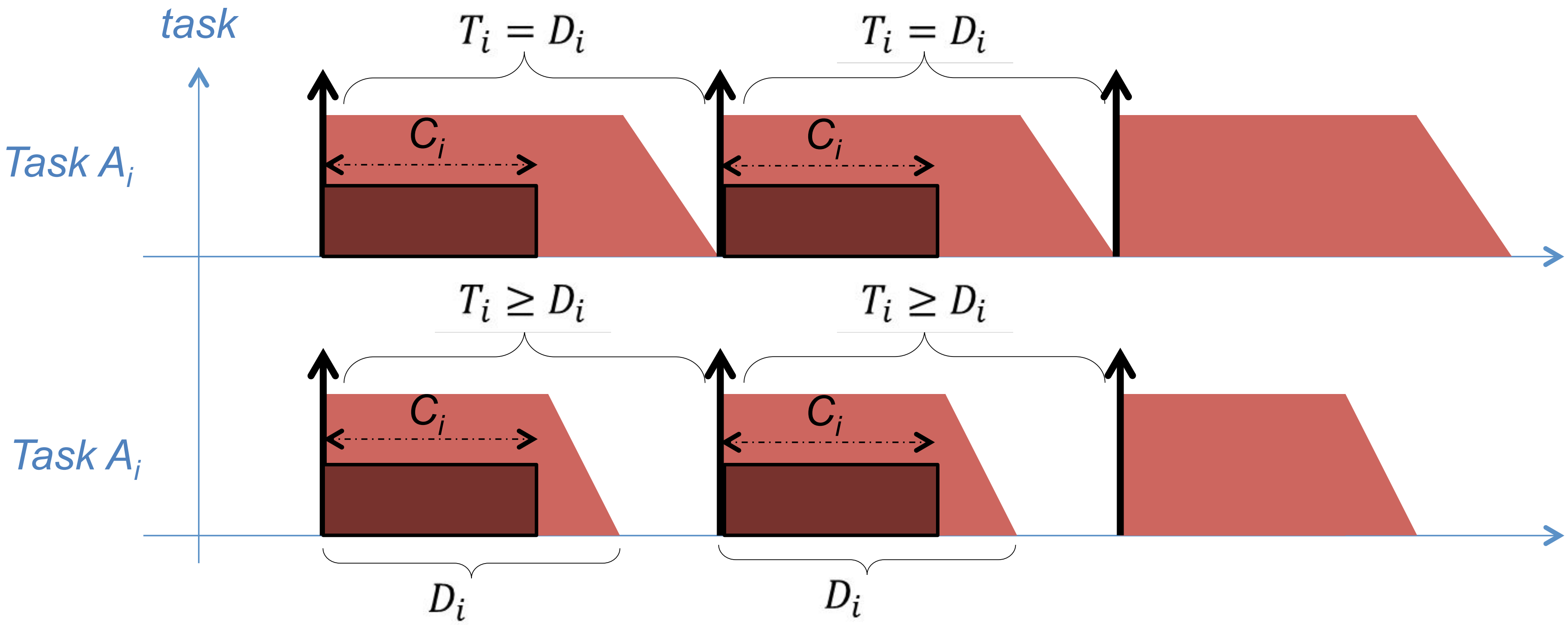
- ALGORITMO DEADLINE MONOTONIC PRIORITY ORDERING (DMPO)
- ALGORITMO TIMELINE SCHEDULING (TS)
- SCHEDULING DI TASK MISTI: PERIODICI E APERIODICI

ALGORTIMO DEADLINE MONOTONIC PRIORITY ORDERING (DMPO)

TASK PERIODICI

Finora abbiamo considerato task periodici in cui il **periodo di attivazione** T_i coincide con la **deadline relativa** D_i.

Ma questo è un caso particolare del caso più generale dei task periodici.

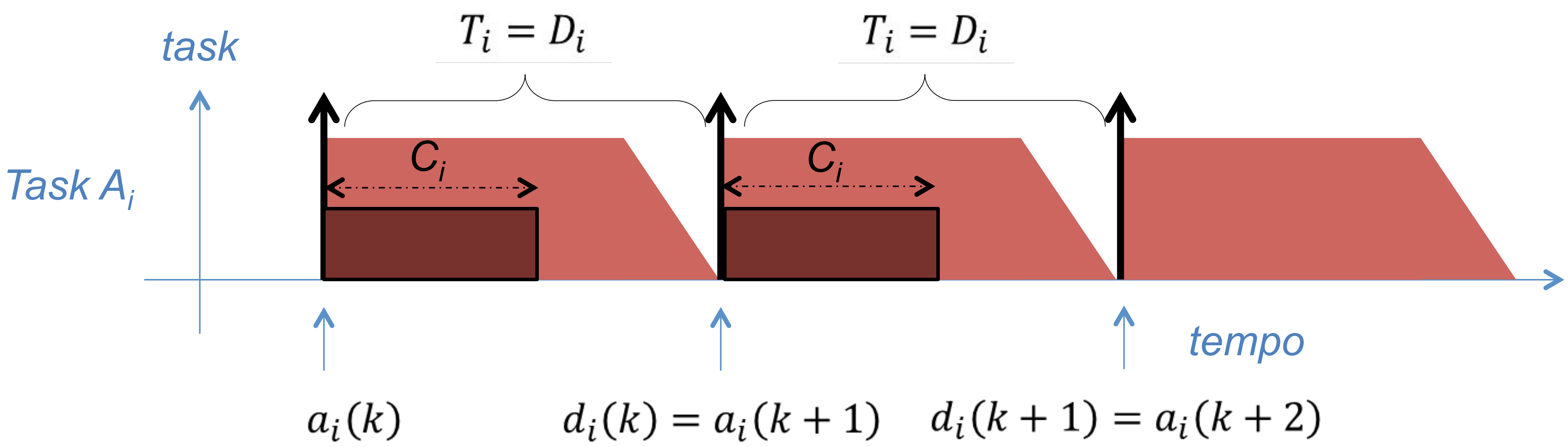


TASK PERIODICI

IPOTESI ORIGINALE

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

1. La deadline relativa $D_i(k)$ del task $A_i(k)$ coincida con il periodo di attivazione T_i ad ogni esecuzione k ($D_i = T_i$)
2. Il computation time C_i di ogni task sia costante ad ogni esecuzione;
3. I task siano indipendenti tra loro e non condividano risorse mutuamente esclusive.

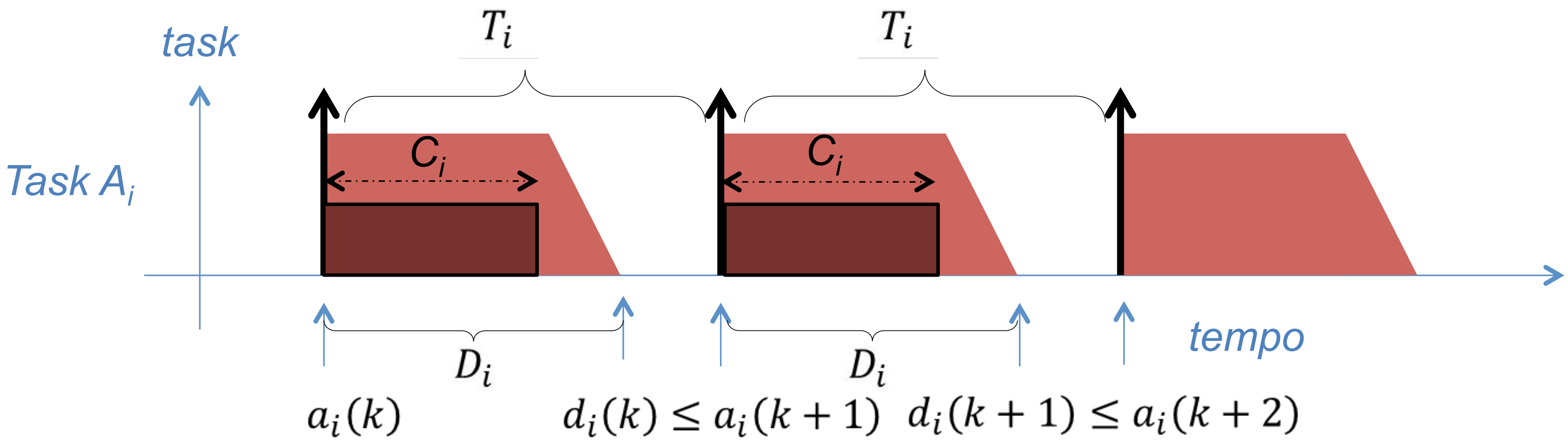


TASK PERIODICI

IPOTESI MODIFICATA

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

1. La deadline relativa $D_i(k)$ del task $A_i(k)$ sia costante e minore o uguale del periodo di attivazione T_i $(D_i \leq T_i)$;
2. Il computation time C_i di ogni task sia costante ad ogni esecuzione;
3. I task siano indipendenti tra loro e non condividano risorse mutuamente esclusive.



TASK PERIODICI

DEFINIZIONE

Dato un insieme di n TASK PERIODICI GENERICI (A_1, A_2, …, A_n) definiamo **REQUISITI** e **VINCOLI** di sistema i seguenti insiemi:

- REQUISITI DI SISTEMA: (n, T_1, T_2, …, T_n, D_1, D_2, …, D_n)
- VINCOLI DI SISTEMA: (C_1, C_2, …, C_n)

DEFINIZIONE

Dato un insieme di n TASK PERIODICI GENERICI (A_1, A_2, …, A_n) definiamo **coefficiente di utilizzazione relativo**:

$$U_{rel} = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

DEADLINE MONOTONIC PRIORITY ORDERING (DMPO)

Siano noti i REQUISITI ed i VINCOLI DI SISTEMA di un problema di scheduling di task periodici generici:

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n, D_1, D_2, \ldots, D_n)$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$

L'algoritmo DMPO è un algoritmo di scheduling **PREEMPTIVE** che assegna a ciascun task una priorità inversamente proporzionale alla deadline relativa D_i .

Dato che al variare del numero e delle deadline relative dei task in ingresso allo scheduler, la configurazione dei task in uscita dallo scheduler può variare, l'algoritmo DMPO è **ON-LINE**.

Dato che la deadline relativa D_i è fissata per ogni task, l'algoritmo DMPO è **STATICO**.

L'algoritmo **DMPO è una generalizzazione dell'algoritmo RMPO**. Si noti infatti che ponendo $T_i = D_i$, si ottiene la definizione dell'algoritmo RMPO.

Proprietà dell'algoritmo DMPO

PROPOSIZIONE 1 (senza dimostrazione)

Se un insieme di task periodici generici NON risulta schedulabile tramite l'algoritmo DMPO, allora **NON ESISTE NESSUN ALTRO ALGORITMO DI SCHEDULING STATICO** che riesca a risolvere lo stesso problema.

PROPOSIZIONE 2

Dato un insieme di task periodici generici, se esiste un algoritmo di scheduling statico che risolve il problema allora esso è schedulabile anche con l'algoritmo DMPO.

PROPOSIZIONE 3 (senza dimostrazione)

Un insieme di n task periodici generici è schedulabile con l'algoritmo DMPO se:

$$\boxed{U_{rel} = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{1/n} - 1\right)}$$

CONDIZIONE SUFFICIENTE SCHEDULABILITÀ DMPO

Proprietà dell'algoritmo DMPO

OSSERVAZIONE

Notiamo come la condizione di sufficienza sia **altamente inefficiente**.

Partiamo dal problema di scheduling di **task periodici generici**:

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n, D_1, D_2, \ldots, D_n)$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$

Il coefficiente di utilizzazione U è minore del coefficiente di utilizzazione relativo U_{rel} infatti:

$$D_i \leq T_i \quad \forall\, i \in \{1,2,\ldots,n\}$$

Da cui deriva che:

$$\frac{1}{T_i} \leq \frac{1}{D_i} \quad \forall\, i \in \{1,2,\ldots,n\}$$

Proprietà dell'algoritmo DMPO

Moltiplicando ambo i membri per il computation time:

$$\frac{C_i}{T_i} \leq \frac{C_i}{D_i} \quad \forall\, i \in \{1,2,\dots,n\}$$

Sommando gli n valori in ambo i membri:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \sum_{i=1}^{n} \frac{C_i}{D_i} = U_{rel}$$

Da cui:

$$U \leq U_{rel}$$

Proprietà dell’algoritmo DMPO

Costruiamo un nuovo problema di scheduling di **task periodici**:

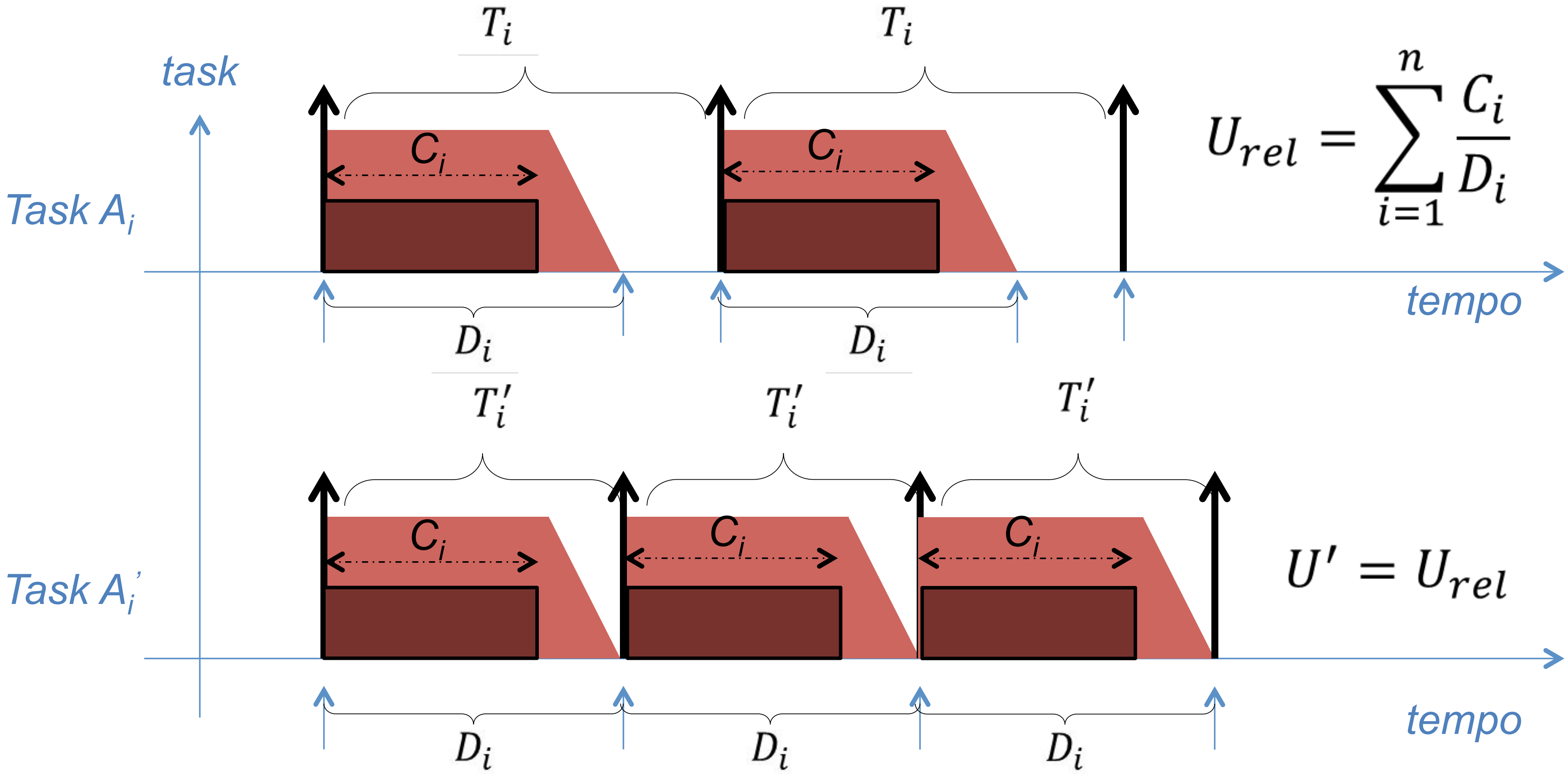
- REQUISITI DI SISTEMA: (n, $T_1' = D_1$, $T_2' = D_2$, …, $T_n' = D_n$)
- VINCOLI DI SISTEMA: (C_1, C_2, …, C_n)

Notiamo che il coefficiente di utilizzazione U’ di questo problema è pari al coefficiente di utilizzazione relativo del primo problema.

$$U' = \sum_{i=1}^{n} \frac{C_i}{T_i'} = \sum_{i=1}^{n} \frac{C_i}{D_i} = U_{rel}$$

Proprietà dell'algoritmo DMPO

Notiamo graficamente come il secondo problema abbia vincoli più stringenti del primo:



$$U_{rel} = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

$$U' = U_{rel}$$

Proprietà dell'algoritmo DMPO

La condizione sufficiente imposta al coefficiente di utilizzazione relativo

$$U_{rel} \leq n\left(2^{1/n} - 1\right)$$

implica che il secondo problema verifichi la condizione sufficiente per la schedulabilità secondo l'algoritmo RMPO:

$$U' = U_{rel} \leq n\left(2^{1/n} - 1\right)$$

Pertanto la schedulabilità tramite RMPO del secondo problema di task periodici con coefficiente di utilizzazione ben maggiore del primo problema di task periodici generici, è indice della inefficienza della condizione sufficiente di schedulabilità tramite DMPO.

$$U \leq U' = U_{rel} \leq n\left(2^{1/n} - 1\right)$$

TEOREMA DI AUDSLEY

TEOREMA DI AUDSLEY (senza dimostrazione)

Dato il problema di scheduling:

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n, D_1, D_2, \ldots, D_n)$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$

Definito l'algoritmo iterativo:

$$R_i^{(0)} = \sum_{k=1}^{i} C_k \qquad I_i^{(0)} = \sum_{k=1}^{i-1} C_k$$

PARTE INTERA SUPERIORE

$$R_i^{(s)} = I_i^{(s)} + C_i \qquad I_i^{(s)} = \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(s-1)}}{T_k} \right\rceil C_k \qquad s \in \mathbb{N}$$

TEOREMA DI AUDSLEY

Allora l'insieme di n task è schedulabile SE E SOLO SE valgono le seguenti due condizioni:

1. $$\boxed{\exists\ \alpha \in \mathbb{N} \mid R_i = R_i^{(\alpha)} = R_i^{(\alpha-1)}}$$

$$\forall i, \qquad 1 \leq i \leq n$$

2. $$\boxed{R_i \leq D_i}$$

La condizione (1.) impone che l'algoritmo iterativo di Audsley converga al valore R_i, la condizione (2.) impone un limite superiore a tale valore.

DMPO vs RMPO

OSSERVAZIONE

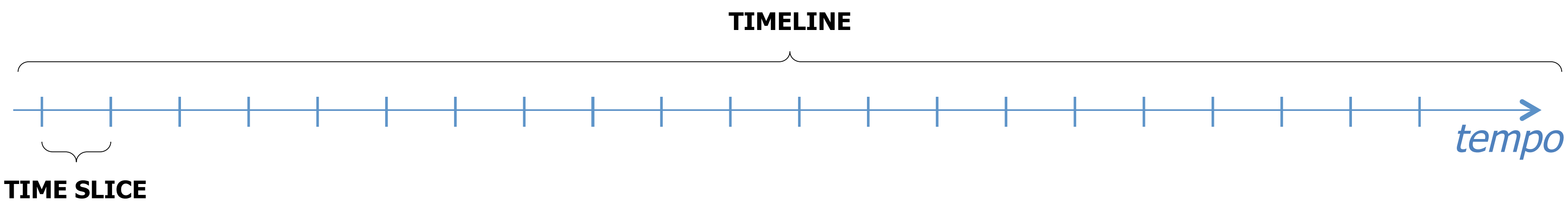
- L'algoritmo **RMPO** può essere utilizzato **esclusivamente per task periodici**, in cui il periodo di attivazione è fissato e noto a priori.

- L'algoritmo **DMPO** può essere utilizzato nello **scheduling di task periodici generici**, in quanto la priorità di scheduling NON dipende dal periodo di occorrenza dei task, ma solo dalle loro deadline relative.

ALGORITMO TIMELINE SCHEDULING (TS)

TIMELINE e TIME SLICES

Un approccio completamente differente al problema dello scheduling di task è quello di **quantizzare la risorsa di calcolo**, suddividendo **l'asse temporale** (**TIMELINE**) in **intervalli di tempo uguali** (**TIME SLICES**), in maniera tale che tutti i task possano essere eseguiti nel rispetto delle loro deadline.



DEFINIZIONE

Dato il problema di scheduling di task periodici:

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n)$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$

Si definiscono:

- **MINOR CYCLE** il **MASSIMO COMUN DIVISORE** dei periodi di attivazione T_i
- **MAJOR CYCLE** il **minimo comune multiplo** dei periodi di attivazione T_i

TIMELINE e TIME SLICES

OSSERVAZIONE

Dato il problema di scheduling di task periodici:

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n)$;
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$;

Si potrebbe obiettare che se $T_i \in \mathbb{R}$ allora non è possibile definire il minimo comune multiplo (mcm) né il massimo comun divisore (MCD) dei valori $T_1, T_2, \ldots, T_n$.

In realtà, la **precisione con cui si misura il tempo** (time unit nano-, micro-, milli-secondi) implica necessariamente che i T_i siano espressi come **multipli interi di time unit**.

Pertanto T_i è un **NUMERO NATURALE POSITIVO** ($T_i \in \mathbb{N}$) ed è quindi pertinente parlare di mcm e MCD dei valori $T_1, T_2, \ldots, T_n$.

ALGORITMO TIMELINE SCHEDULING (TS)

DEFINIZIONE

Dato un problema di scheduling di n di task periodici:

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n)$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$

Data una TIMELINE di un sistema MONOPROCESSORE divisa in TIME SLICES tale che:

- La durata di ogni TIME SLICE è pari al MINOR CYCLE
- Il COMPUTATION TIME di ogni task sia inferiore o uguale al MINOR CYCLE
- Durante ogni TIME SLICE NON si effettua PREEMPTION

L'algoritmo TS è un algoritmo di scheduling **NON PREEMPTIVE** e **OFFLINE** che assegna in maniera arbitraria a ciascun time slice l'intera esecuzione di uno o più task.

L'algoritmo TS è **OFFLINE**, in quanto una volta definita la distribuzione dell'esecuzione dei task in ciascun time slice, lo scheduling è noto a priori.

ESEMPIO

PROBLEMA

Dato un problema di scheduling di n di task periodici:

- REQUISITI DI SISTEMA: (n = 3, T_1= 8 t.u., T_2 = 16 t.u., T_3 = 32 t.u.)
- VINCOLI DI SISTEMA: (C_1= 2 t.u., C_2= 4 t.u., C_3= 6 t.u.)

Mostrare uno schema di timeline scheduling che risolva il problema.

SOLUZIONE

Innanzitutto verifichiamo che il problema non sia inammissibile:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{2}{8} + \frac{4}{16} + \frac{6}{32} = \frac{8+8+6}{32} = \frac{22}{32} = 0{,}6875 < 1$$

Dato che il coefficiente di utilizzazione è minore di 1, il problema non è inammissibile.

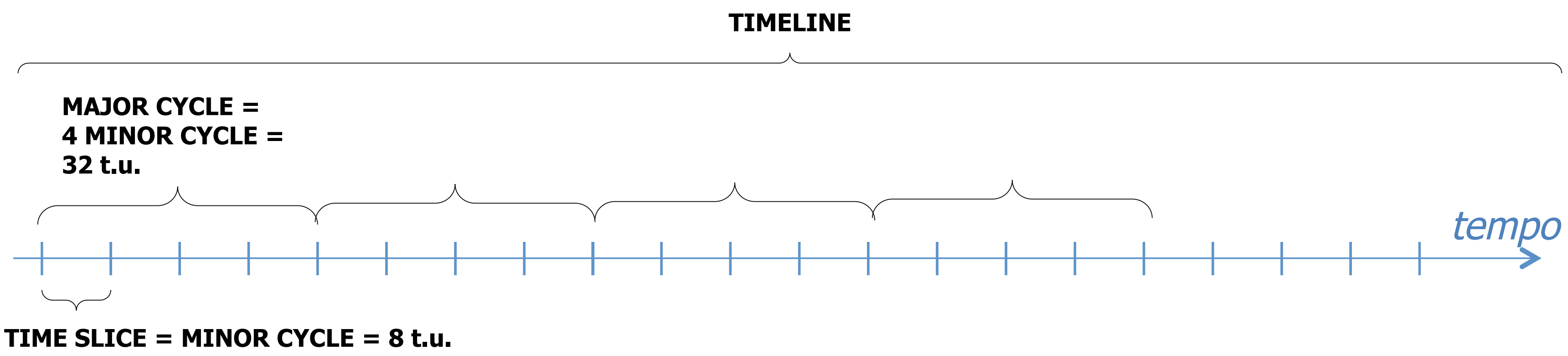
ESEMPIO cont'd

Passiamo quindi a tracciare la TIMELINE e i differenti TIME SLICE che la compongono.

Calcoliamo pertanto il MINOR CYCLE (pari alla durata del TIME SLICE) e il MAJOR CYCLE (che definisce la periodicità dell'algoritmo di TIMELINE SCHEDULING).

$$MINOR\ CYCLE = MCD(8,16,32) = 8\ t.u.$$

$$MAJOR\ CYCLE = mcm(8,16,32) = 32\ t.u.$$

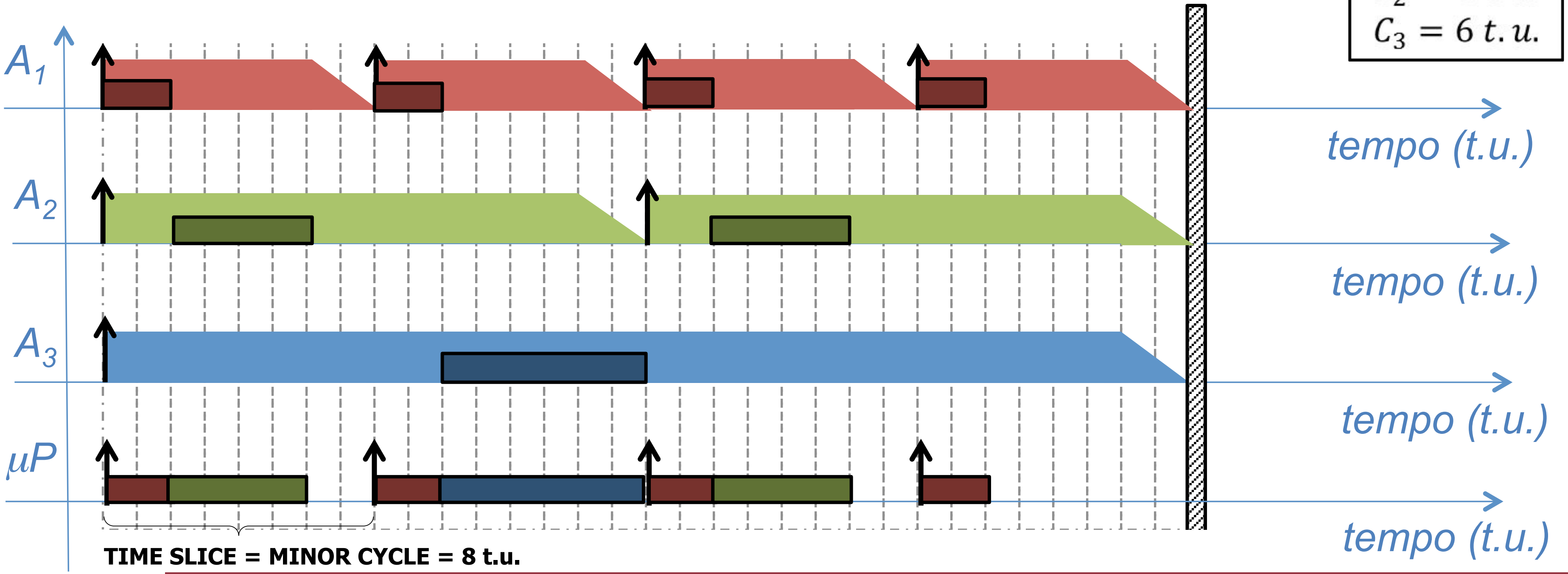


ESEMPIO cont'd

Tracciamo il diagramma temporale dei 3 task periodici ed identifichiamo una possibile soluzione, notando che:

1. Il task A_1 dovrà ripetersi MAJOR CYCLE / T_1 = 4 volte in un MAJOR CYCLE
2. Il task A_2 dovrà ripetersi MAJOR CYCLE / T_2 = 2 volte in un MAJOR CYCLE
3. Il task A_3 dovrà ripetersi MAJOR CYCLE / T_3 = 1 volta in un MAJOR CYCLE

$$C_1 = 2\ t.u.$$
$$C_2 = 4\ t.u.$$
$$C_3 = 6\ t.u.$$

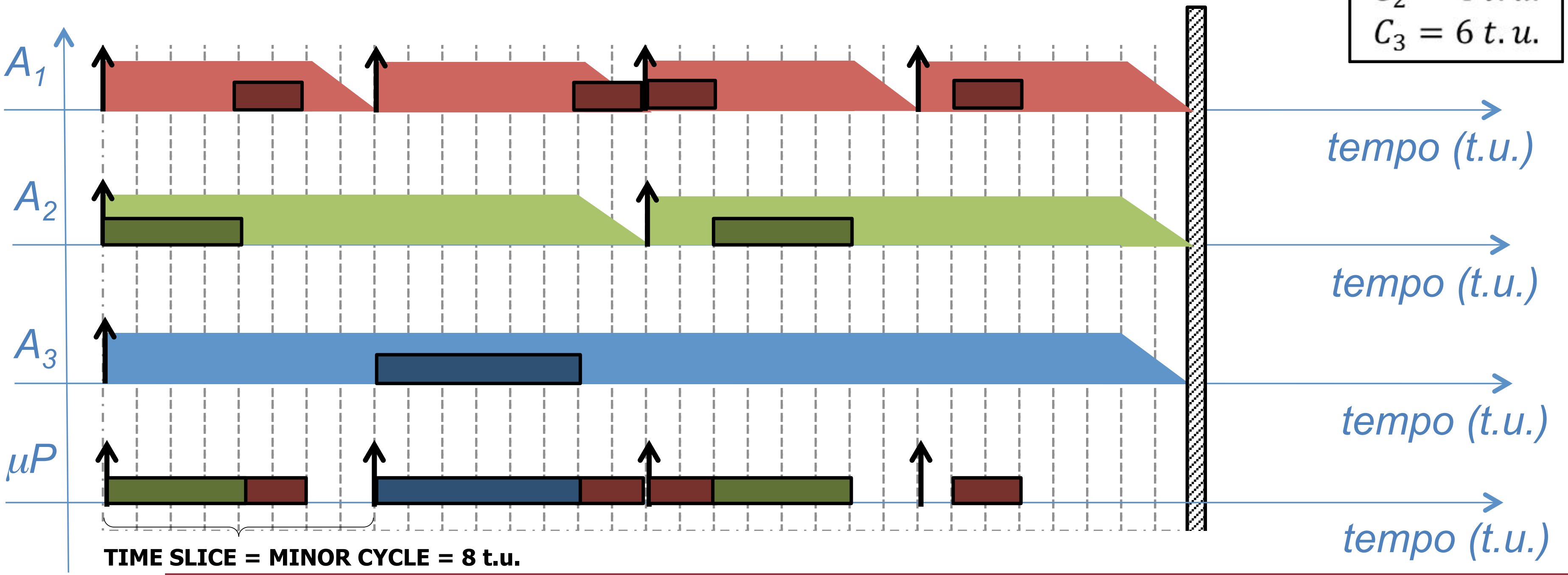


ESEMPIO cont'd

Ovviamente la soluzione NON È UNIVOCA. La condizione SUFFICIENTE affinché un algoritmo di TIMELINE SCHEDULING sia una soluzione è che la somma dei COMPUTATION TIME in ogni TIME SLICE sia inferiore o uguale al MINOR CYCLE.

Ad esempio, un'altra possibile soluzione è:

$$C_1 = 2\ t.u.$$
$$C_2 = 4\ t.u.$$
$$C_3 = 6\ t.u.$$



TIMELINE SCHEDULING - VANTAGGI

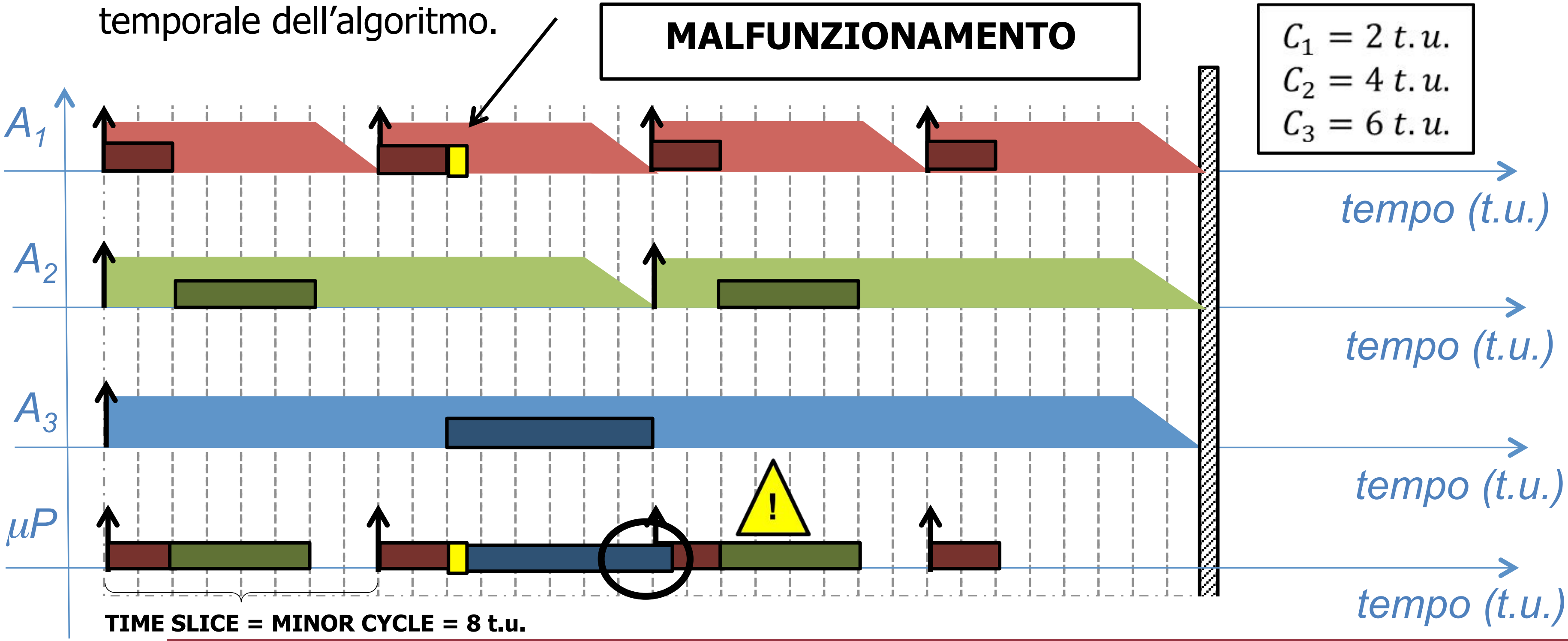
- **SEMPLICITÀ DI IMPLEMENTAZIONE** – Un timer tiene conto dell'inizio di ogni MINOR CYCLE e una allocazione statica ed offline dei task viene ciclicamente eseguita ad ogni MAJOR CYCLE

- **NESSUNA PRE-EMPTION** – Non è necessario memorizzare lo stato di quei task la cui esecuzione viene bloccata a seguito della preemption

- **NESSUNA PRIORITÀ** – Essendo l'algoritmo di TIMELINE SCHEDULING OFF-LINE, non è necessario eseguire calcoli o controlli sulla priorità dei task attivi, semplicemente vengono eseguiti i task in base alla tabella di scheduling valida per quel MINOR CYCLE

TIMELINE SCHEDULING - SVANTAGGI

- **FLESSIBILITÀ** – la modifica anche di un computation time di uno o più dei task periodici comporta, necessariamente, la ristrutturazione dell’algoritmo.
- **ROBUSTEZZA** – essendo un algoritmo OFF-LINE, lo scheduling è noto a priori, pertanto un qualsiasi malfunzionamento in uno dei task, può minare la correttezza temporale dell’algoritmo.



TIMELINE SCHEDULING

OSSERVAZIONI

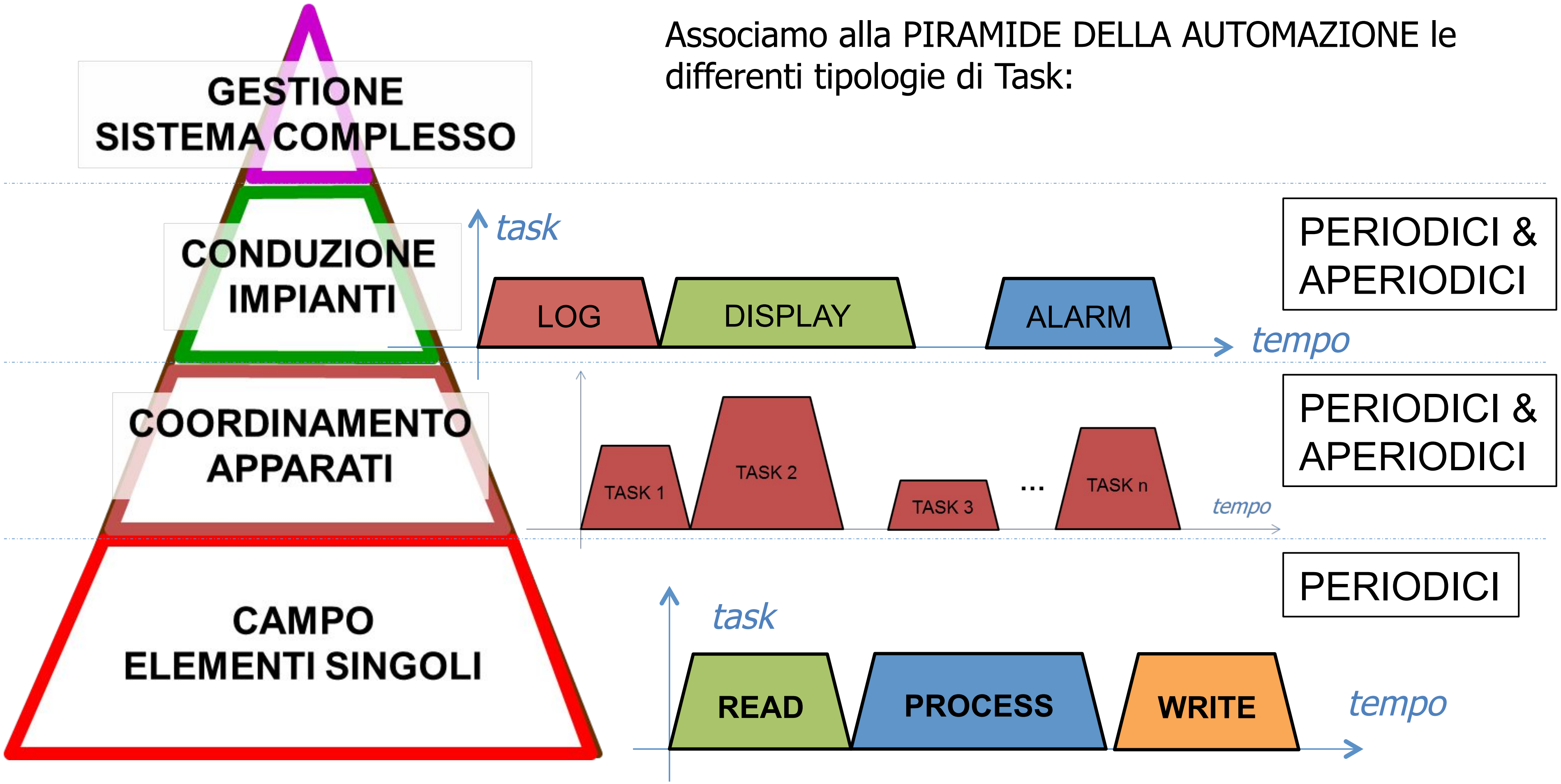
- L'algoritmo di TIMELINE SCHEDULING è conveniente quando i tempi di attivazione e di esecuzione dei task sono **RIGIDAMENTE FISSATI** e **NON SONO SOGGETTI A VARIAZIONI** dovute a situazioni anomale

- L'algoritmo di **TIMELINE SCHEDULING NON GESTISCE TASK APERIODICI**

SCHEDULING DI TASK MISTI: PERIODICI E APERIODICI

TASK PERIODICI E TASK APERIODICI NELLA AUTOMAZIONE

Associamo alla PIRAMIDE DELLA AUTOMAZIONE le differenti tipologie di Task:



TASK PERIODICI E TASK APERIODICI

OSSERVAZIONE

Nei SISTEMI DI AUTOMAZIONE è molto frequente la situazione in cui si debbano gestire **SIMULTANEAMENTE** task **PERIODICI** assieme a task **APERIODICI**.

ESEMPI DI TASK PERIODICI

- Lettura dati dai sensori (polling)
- Invio comandi agli attuatori (enforcement)
- Elaborazione delle azioni di intervento
- Log dei dati su database
- Aggiornamento Human Machine Interface

ESEMPI DI TASK APERIODICI

- Gestione allarmi (anomalie, malfunzionamenti, emergenza, sicurezza, etc.)
- Gestione input utente (ad es. on/off)

TASK MISTI (PERIODICI E APERIODICI)

Non esiste un metodo universale per trattare simultaneamente Task PERIODICI e task APERIODICI (ovvero Task MISTI)

È **necessario definire REQUISITI** ed opportune IPOTESI per poter verificare la schedulabilità o meno di **Task MISTI**.

Il **primo REQUISITO** che bisogna chiarire è se i Task APERIODICI debbano rispettare vincoli **HARD REAL-TIME** o **SOFT REAL-TIME**.

Un **esempio** di task **APERIODICO HARD REAL TIME** è dato dagli **allarmi** legati all'**incolumità del personale**.

Un **esempio** di task **APERIODICO SOFT REAL TIME** è dato dalle interazioni con il display utente (**Human-Machine Interface**).

Nel seguito tratteremo in maniera differenziata i due casi.

TASK MISTI – HARD REAL TIME

IPOTESI PER TASK APERIODICI HARD REAL TIME

1. Supponiamo di conoscere **l'intervallo di occorrenza MINIMO** (T_i^{min}) tra due attivazioni dello stesso Task aperiodico A_i
2. Supponiamo di conoscere il **MASSIMO tempo di computazione** (C_i^{MAX}) di qualsiasi occorrenza del Task aperiodico A_i

DEFINIZIONE

Dato un problema di scheduling di TASK MISTI, composto da

n di task periodici

- REQUISITI DI SISTEMA: $(n, T_1, T_2, \ldots, T_n)$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n)$

ed m task aperiodici che verifichino le ipotesi 1. e 2.

- REQUISITI DI SISTEMA: $(m, T_{n+1}^{min}, T_{n+2}^{min}, \ldots, T_{n+m}^{min})$
- VINCOLI DI SISTEMA: $(C_{n+1}^{MAX}, C_{n+2}^{MAX}, \ldots, C_{n+m}^{MAX})$

Si definisce problema di scheduling EQUIVALENTE il seguente problema di scheduling di TASK PERIODICI:

TASK MISTI – HARD REAL TIME

PROBLEMA EQUIVALENTE

- REQUISITI DI SISTEMA: $(n+m, T_1, T_2, \ldots, T_n, T_{n+1}^{min}, T_{n+2}^{min}, \ldots, T_{n+m}^{min})$
- VINCOLI DI SISTEMA: $(C_1, C_2, \ldots, C_n, C_{n+1}^{MAX}, C_{n+2}^{MAX}, \ldots, C_{n+m}^{MAX})$

OSSERVAZIONI

VANTAGGIO

Il problema equivalente può essere risolto con gli algoritmi di scheduling già visti (RMPO, EDF, DMPO) **garantendo la schedulabilità**.

SVANTAGGIO

Il prezzo da pagare è l'**inefficienza dell'utilizzo effettivo del processore**.

Ogniqualvolta un task aperiodico **tarda ad essere attivato** o **impiega un tempo di computazione minore di quello massimo**, il processore viene schedulato per non eseguire alcuna operazione.

TASK MISTI – SOFT REAL TIME

In AUTOMAZIONE sono molto diffusi i task APERIODICI SOFT REAL TIME.

In questo caso l'algoritmo di scheduling deve GARANTIRE i task HARD REAL TIME (periodici ed aperiodici) e SERVIRE AL MEGLIO (BEST EFFORT) i task SOFT REAL TIME.

Esistono due metodologie di scheduling per task MISTI SOFT e HARD REAL TIME:

1. ALGORITMI DI SERVIZIO IN BACKGROUND
2. ALGORITMI TRAMITE SERVER

BIBLIOGRAFIA

Sezione 2.4 (pagg. 56-61)

TITOLO

Sistemi di automazione industriale

Architetture e controllo

AUTORI

Claudio Bonivento

Luca Gentili

Andrea Paoli

EDITORE

McGraw-Hill