

# Lezione R11

## Real-time su multiprocessore I

Sistemi operativi open-source, embedded e real-time

30 novembre 2017

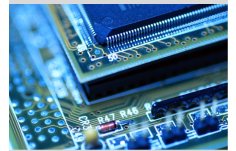
Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica  
Università degli Studi di Roma Tor Vergata

### Di cosa parliamo in questa lezione?

In questa lezione si dà una visione introduttiva del problema della schedulazione real-time in sistemi multiprocessore

- 1 Sistemi multiprocessore
- 2 Effetto Dhall
- 3 Anomalie di schedulazione
- 4 Test e condizioni di schedulabilità
- 5 Schedulazione partizionata



Schema della lezione

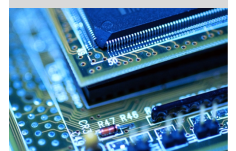
Sistemi  
multiprocessori

Effetto Dhall

Anomalie di  
schedulazione

Schedulabilità

Schedulazione  
partizionata



Schema della lezione

Sistemi  
multiprocessori

Effetto Dhall

Anomalie di  
schedulazione

Schedulabilità

Schedulazione  
partizionata

## Sistemi multiprocessore

Un sistema real-time è detto **multiprocessore** quando è dotato di due o più processori, ciascuno in grado di eseguire job autonomamente

I processori possono essere dello stesso tipo o di tipo diverso

Ad esempio si consideri un sistema costituito da

- Diversi microprocessori multi-core
- Diverse schede di rete
- Diverse schede PCI con controllori DMA

In generale modellando il sistema è necessario specificare:

- il numero  $\mu$  di **tipi di processore**
- il numero  $m_i$  di processori dell' $i$ -esimo tipo ( $1 \leq i \leq \mu$ )
- su quali tipi di processore può eseguire ciascun job

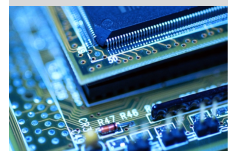
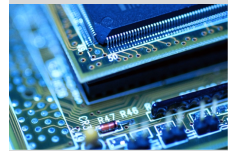
In questa lezione tutti i processori sono dello stesso tipo

## Real-time su sistemi multiprocessori

*Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem.*

*The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.*

C. L. Liu, 1969



## Sistemi statici

Un sistema real-time è detto *statico* quando ciascun job è assegnato ad uno specifico processore

Esistono due varianti di sistemi *statici*:

- L'insieme dei job (o task) nel sistema è predeterminato, e l'assegnazione di ciascun job ad uno specifico processore è effettuata una volta per tutte nella fase di progetto del sistema
- L'insieme dei task nel sistema non è predeterminato, e l'assegnazione del task ad uno specifico processore è effettuata dal sistema operativo durante la fase di creazione del task (*scheduler partizionati*)

In entrambi i casi lo scheduler **non** decide su quale processore sarà eseguito un job appena rilasciato (è già stabilito)

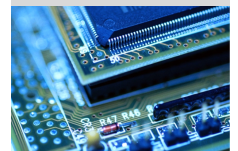
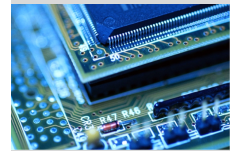
## Sistemi dinamici

Un sistema real-time è detto *dinamico* quando lo scheduler può assegnare dinamicamente un job ad un qualunque processore disponibile

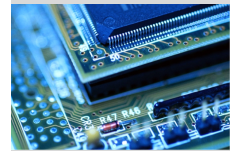
Esistono tre varianti di sistemi *dinamici*:

- Con job *non interrompibili*
- Con job *interrompibili* e *non migrabili*: anche se interrotto, il job deve riprendere l'esecuzione sullo stesso processore in cui era in esecuzione precedentemente
- Con job *interrompibili* e *migrabili*: una volta interrotto, il job può riprendere l'esecuzione su qualunque processore che possa eseguirlo

Un algoritmo di schedulazione per un sistema dinamico è detto *globale* perché stabilisce quale processore eseguirà ciascun job



## Esempio di schedulazione in sistema statico

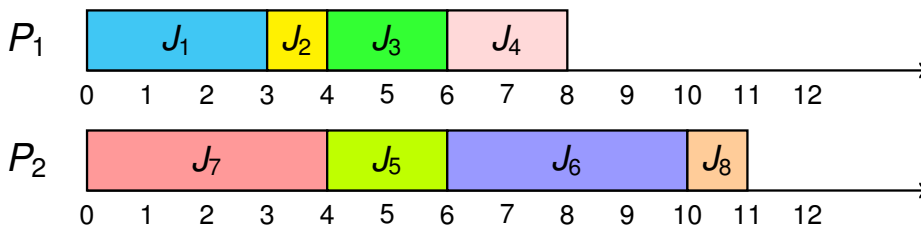
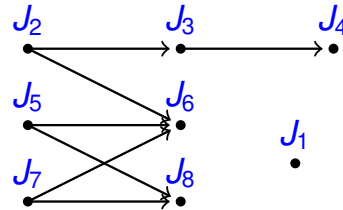


In un sistema **statico** esiste per ogni processore una lista di task o job con le relative priorità

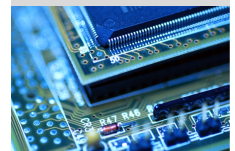
Lista processore  $P_1$ :  $J_1, J_2, J_3, J_4$

Lista processore  $P_2$ :  $J_5, J_6, J_7, J_8$

$i$	1	2	3	4	5	6	7	8
$r_i$	0	0	0	0	4	0	0	0
$e_i$	3	1	2	2	2	4	4	1

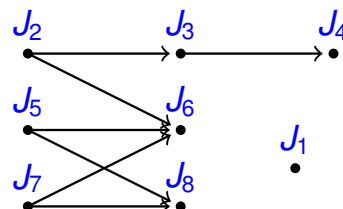


## Esempio di schedulazione in sistema dinamico

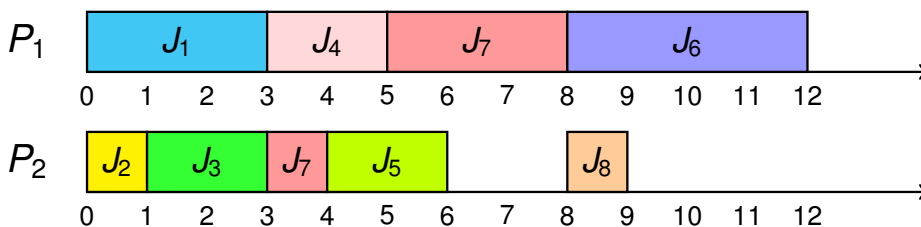


Lista:  $J_1, J_2, \dots, J_8$

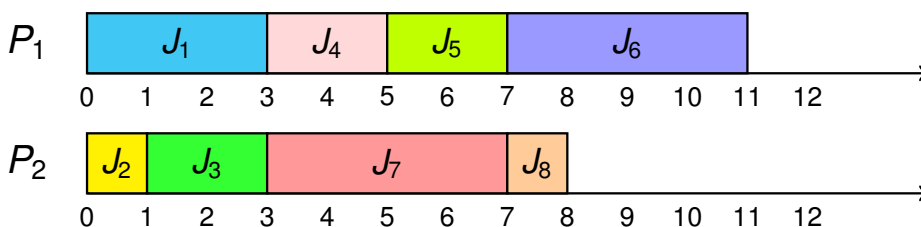
$i$	1	2	3	4	5	6	7	8
$r_i$	0	0	0	0	4	0	0	0
$e_i$	3	1	2	2	2	4	4	1



Job interrompibili e migrabili:



Job non interrompibili:



## Vantaggi dei sistemi statici

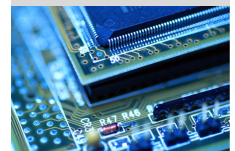
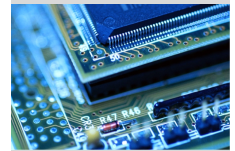
*Quali sono i vantaggi dei sistemi statici?*

- Si può analizzare la schedulabilità su ciascun processore utilizzando i risultati teorici validi per il caso uniprocessore (fondamentale per i sistemi hard real-time!)
- Un job che impiega più tempo di quanto previsto dal suo WCET (*overrun*) può ritardare l'esecuzione dei soli task associati al suo processore
- Poiché i job interrotti riprendono sempre l'esecuzione sullo stesso processore si evitano i costi dovuti alla migrazione del contesto ad un altro processore
- La coda di esecuzione (in cui i job rilasciati aspettano di essere attivati) è relativa al singolo processore ed è quindi più piccola

## Vantaggi dei sistemi dinamici

*Quali sono i vantaggi dei sistemi dinamici?*

- Hanno tipicamente meno cambi di contesto e interruzioni dei job, poiché un job è interrotto solo quando nessun processore è idle
- Se un job esegue per meno tempo di quanto previsto dal suo WCET, il tempo liberato sul processore può essere utilizzato potenzialmente da tutti i task nel sistema
- Se un job impiega più tempo di quanto previsto dal suo WCET (*overrun*), la probabilità che ciò comporti il mancato rispetto di una o più scadenze è minore
- Per ogni task del sistema che è creato a run-time, assegnazione e bilanciamento del carico sono "automatici" e determinati dall'algoritmo di schedulazione **globale**



# Algoritmi di schedulazione multiprocessore

Gli algoritmi di schedulazione **clock-driven** sono in generale utilizzabili senza problemi con i sistemi multiprocessore

Infatti la schedulazione effettiva è generata “off-line” e validata una volta per tutte

Al contrario, non è immediato applicare gli algoritmi **priority-driven** ai sistemi multiprocessore

Diverse problematiche:

- Efficienza degli algoritmi (effetto Dhall)
- Predicibilità del sistema (anomalie di schedulazione)
- Test di schedulabilità (teoremi non più validi)

## Effetto Dhall

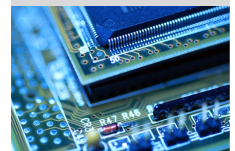
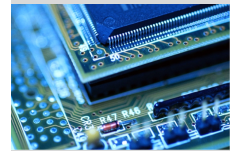
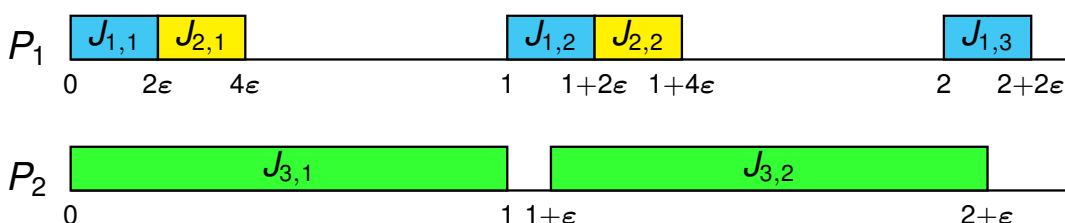
### Teorema (Dhall & Liu, 1978)

Per ogni numero di processori  $m \geq 2$ , esistono insiemi di task con utilizzazione bassa che non sono schedulabili con RM, DM o EDF

Consideriamo  $T_1 = (1, 2\varepsilon)$ ,  $T_2 = (1, 2\varepsilon)$ ,  $\dots$ ,  $T_m = (1, 2\varepsilon)$ ,  
 $T_{m+1} = (1 + \varepsilon, 1)$

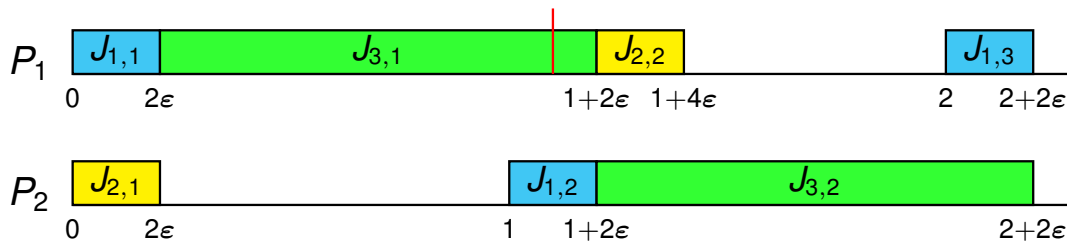
Utilizzazione globale:  $U_g = 2\varepsilon \cdot m + 1/(1 + \varepsilon) \rightarrow 1$  se  $\varepsilon \rightarrow 0$

Schedulazione fattibile,  $m = 2$ :

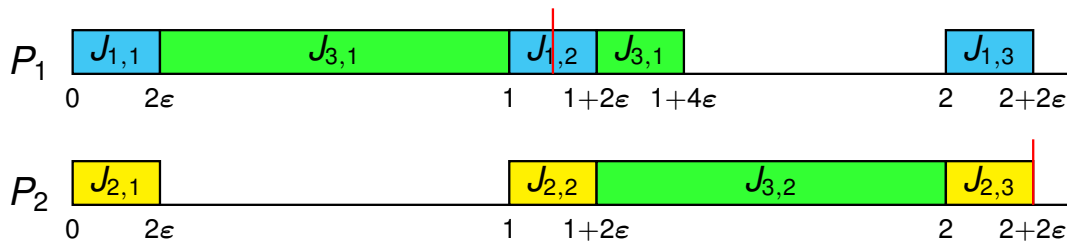


## Effetto Dhall (2)

Schedulazione con EDF,  $m = 2$ :



Schedulazione con RM,  $m = 2$ :



Comportamenti analoghi all'effetto Dhall si verificano solo se almeno uno dei task ha una utilizzazione molto alta (Funk, Goossens & Baruah, 2001)

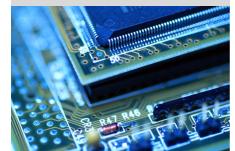
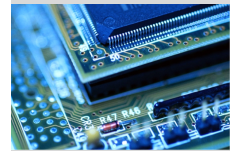
## Anomalie di schedulazione

Si definisce *anomalia di schedulazione* il comportamento di un algoritmo di schedulazione per cui, in presenza di variazioni apparentemente vantaggiose del carico del sistema, si ottiene un peggioramento delle prestazioni

Esempi di variazioni “vantaggiose”:

- Aumento del periodo di un task
- Diminuzione del tempo di esecuzione di un task
- Rimozione di vincoli di precedenza tra i job
- Aumento del numero di processori

Nei sistemi uniprocessore le anomalie di schedulazione possono verificarsi solo nel caso in cui job sono **non interrompibili** e/o **non indipendenti** (Mok, 2000)





# Anomalie di schedulazione in sistemi multiprocessore

Assumiamo che tutti i job siano indipendenti:

Tipo di sistema	Anomalie ?
Statico, job non interrompibili	● Sì (1)
Statico, job interrompibili	● No
Dinamico, job non interrompibili	● Sì (2)
Dinamico, job interrompibili ma non migrabili	● Sì (3)
Dinamico, job interrompibili e migrabili	● Sì (4)

- (1) Mok, 2000 (cfr. esempi in Lezione R5)
- (2) Graham, 1969
- (3) Ha & Liu, 1994
- (4) Andersson & Jonsson, 2000

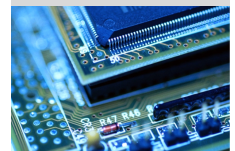
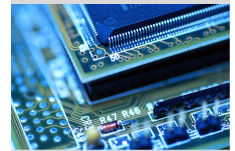
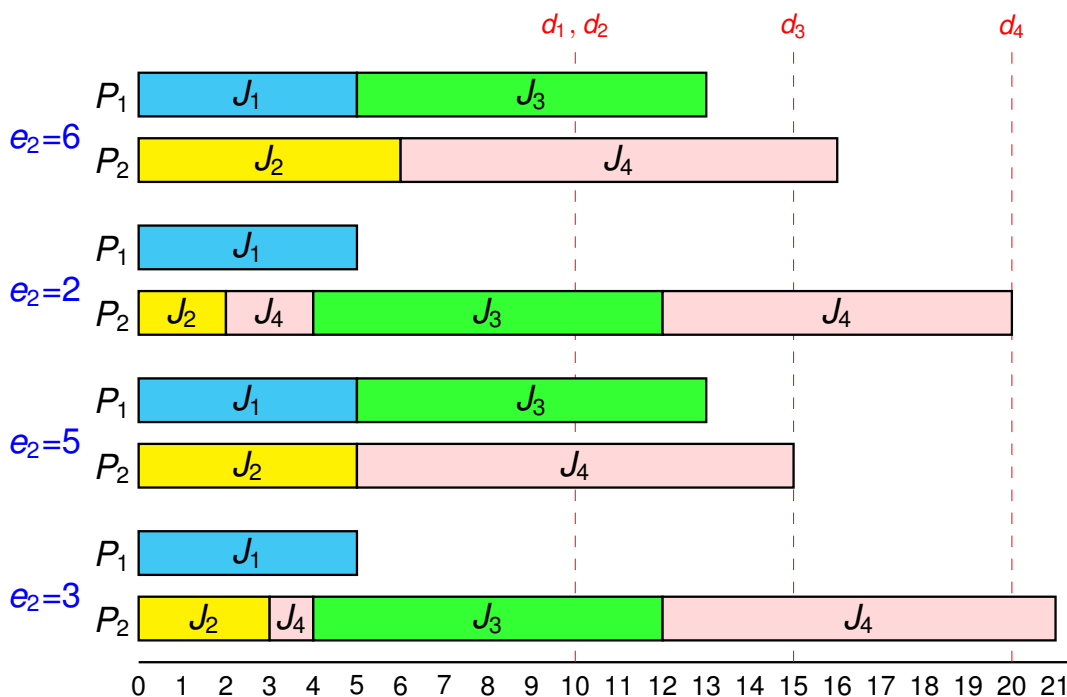
*Perché le anomalie complicano il problema della validazione?*

Se i parametri dei job possono variare, non si può validare il sistema esaminando solo il "caso peggiore", ma è necessario esaminare tutte le combinazioni di parametri

## Anomalie di schedulazione con job non migrabili

Job interrompibili ma non migrabili  
 Indice minore  $\equiv$  priorità maggiore  
 $e_2$  varia da 2 a 6

$i$	1	2	3	4
$r_i$	0	0	4	0
$d_i$	10	10	15	20
$e_i$	5	[2, 6]	8	10



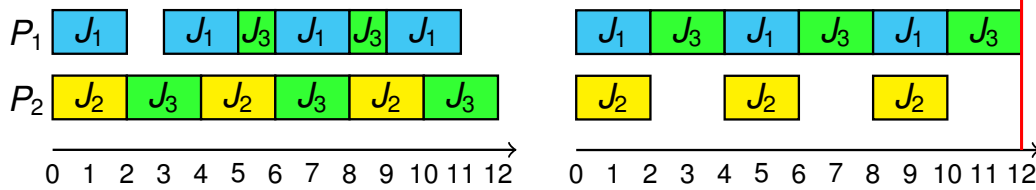


## Anomalie di schedulazione con job migrabili

Aumento del periodo di un task di priorità alta:

$$T_1 = (3, 2), T_2 = (4, 2), \\ T_3 = (12, 8)$$

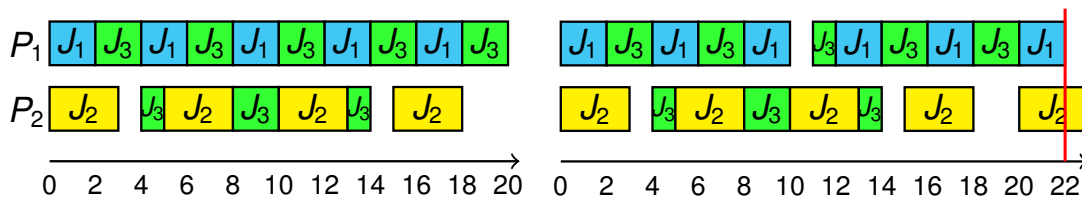
$$T_1 = (4, 2), T_2 = (4, 2), \\ T_3 = (12, 8)$$



Aumento del periodo di un task di priorità bassa:

$$T_1 = (4, 2), T_2 = (5, 3), \\ T_3 = (10, 7)$$

$$T_1 = (4, 2), T_2 = (5, 3), \\ T_3 = (11, 7)$$

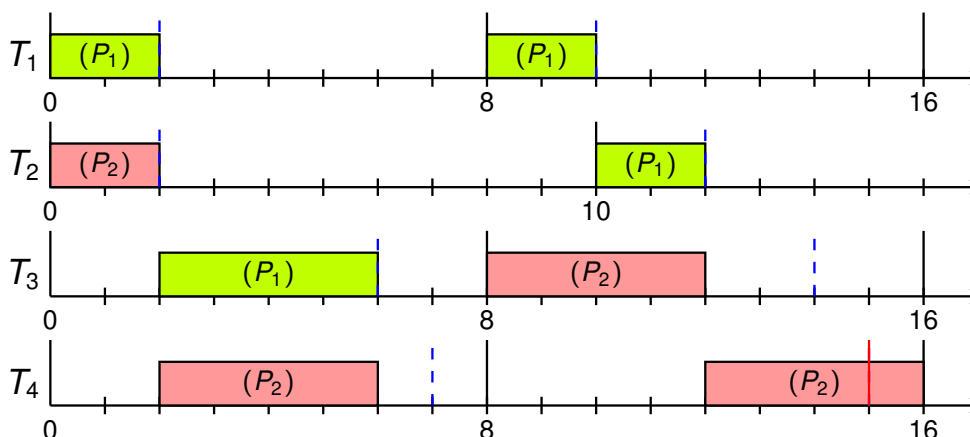


## Istanti critici in schedulazioni globali

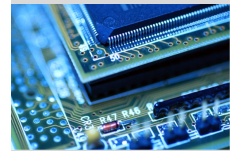
### Teorema (Lauzac, Melhem & Mosse, 1998)

Utilizzando uno scheduler globale a priorità fissa a livello di task (es., DM), l'istante in cui un job di un task  $T_i$  è rilasciato contemporaneamente ai job di tutti i task di priorità superiore  $T_1, \dots, T_{i-1}$  **non** è necessariamente un istante critico di  $T_i$

$$T_1=(8, 2, 2), T_2=(10, 2, 2), T_3=(8, 4, 6), T_4=(8, 4, 7), m=2$$



Il test di schedulabilità non funziona!



## Fattore di utilizzazione per multiprocessore

### Teorema (Oh & Baker, 1998)

Dato un sistema di task periodici con scadenze uguali ai periodi e  $m$  processori, se  $X$  è un qualsiasi algoritmo di schedulazione partizionato con priorità fissa a livello di task:

$$U_X \leq \frac{m+1}{1+2^{1/(m+1)}}$$

### Teorema (Andersson, Baruah & Jonsson, 2001)

Dato un sistema di task periodici con scadenze uguali ai periodi e  $m$  processori, sia  $X$

- un qualsiasi algoritmo di schedulazione partizionato, o
- un qualsiasi algoritmo di schedulazione globale con priorità fissa a livello di job;

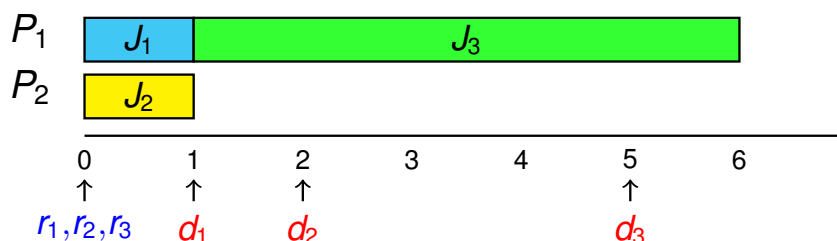
allora per il fattore di utilizzazione di  $X$  si ha:  $U_X \leq \frac{m+1}{2}$

## Schedulazione a priorità fissa su multiprocessore

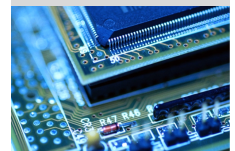
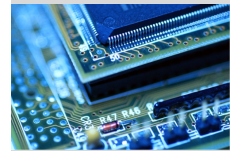
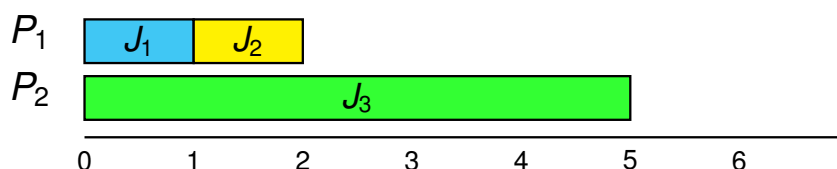
### Corollario (Andersson, Baruah & Jonsson, 2001)

Nessun algoritmo di schedulazione globale con priorità fissa a livello di job è ottimale su multiprocessore

Schedul. EDF di  $T_1 = (1, 1)$ ,  $T_2 = (2, 1)$ ,  $T_3 = (5, 5)$ ,  $m = 2$ :



Eppure una schedulazione fattibile **non EDF** esiste:



## Algoritmi ottimali per multiprocessore

*Possono esistere algoritmi ottimali per multiprocessore? Sì!*

- Alcuni algoritmi di schedulazione dinamica a livello di job hanno fattore di utilizzazione pari a  $m$
- Tuttavia, nessun algoritmo **on-line** (non “chiaroveggente”) è ottimale se gli istanti di rilascio dei job non sono esattamente prefissati (Fisher, 2007)

Una classe di algoritmi ottimali su multiprocessore è derivata dall'algoritmo **Pfair** (Baruah & al., 1996):

- basato sull'idea di schedulazione *fluida*: ogni task progredisce in modo proporzionale alla sua utilizzazione
- tempo diviso in *quanti*: allo scadere di ogni **quanto**, lo scheduler assegna i task ai processori in modo che per ogni task  $T_i$  il lavoro compiuto sia  $\lceil t \cdot e_i/p_i \rceil$  o  $\lfloor t \cdot e_i/p_i \rfloor$

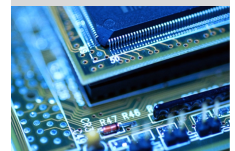
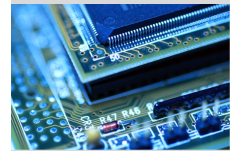
Gli algoritmi dinamici a livello di job sono molto costosi in termini di overhead dello scheduler, quindi non sono adottati

## Schedulazione partizionata

Nei sistemi real-time multiprocessore **statici** l'algoritmo di schedulazione è detto *partizionato*

Consiste di due componenti:

- **Allocazione dei task**: assegnazione di ciascun task ad uno specifico processore
  - questo problema è analogo a *bin packing* ed è **NP hard** (Garey & Johnson, 1979)
- **Problema di priorità**: schedulazione dei task su ciascun processore
  - questo è il problema della schedulazione su sistemi con un singolo processore



## Allocazione dei task

### Formulazione del problema

Dato un sistema di task periodici, partizionare i task in sottoinsiemi tali che ciascun sottoinsieme può essere schedulato in modo fattibile su un singolo processore utilizzando un determinato algoritmo di schedulazione

Un sistema di  $n$  task indipendenti è schedulabile con  $n$  processori (purché ciascun task abbia densità inferiore a uno)

Non è noto alcun algoritmo polinomiale che sia in grado di determinare, dato un sistema di  $n$  task indipendenti, il minimo numero  $m_0$  di processori che permetta di schedularlo

Gli algoritmi di allocazione dei task utilizzabili in pratica trovano soluzioni non ottimali:

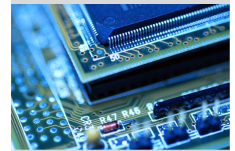
- Non riescono ad associare i task ai processori in modo da sfruttarli nel miglior modo possibile
- Non riescono a determinare schedulazioni fattibili per ogni possibile insieme di task schedulabile

## Allocazione dei task (2)

*Come misurare la "bontà" di un algoritmo di allocazione?*

Tre metriche principali:

- **Rapporto di approssimazione**: è il massimo valore  $m/m_0$ , ove  $m$  è il numero di processori utilizzato dall'algoritmo di allocazione e  $m_0$  è il minimo numero teoricamente necessario, considerando ogni possibile sistema di task
- **Fattore di accelerazione**: quanto è necessario aumentare la velocità di esecuzione degli  $m_0$  processori per schedulare fattibilmente ogni possibile sistema di task con le assegnazioni determinate dall'algoritmo di allocazione
- **Fattore di utilizzazione**: il valore di soglia per cui tutti i sistemi di task con fattore di utilizzazione totale inferiore o uguale sono sempre schedulabili utilizzando l'algoritmo di allocazione dei task



## Algoritmo RMFF

Il più semplice algoritmo per l'allocazione dei task è **RMFF** (**R**ate **M**onotonic **F**irst **F**it, Dhall & Liu, 1978):

- 1 ordina i task per periodi non decrescenti:  $T_1, T_2, \dots$
- 2 ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- 3 cominciando da  $T_1$ , assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **RM**

- $U_{\text{RMFF}} = m \cdot (\sqrt{2} - 1)$  (Oh & Baker, 1998)
- Fattore di approssimazione: **2.23** (Oh & Son, 1993)

*RMFF può essere usato come un algoritmo on-line?*

L'ordinamento dei job richiede la conoscenza di tutti i periodi dei task da schedulare  $\Rightarrow$  usare **RMFF** on-line richiede di riallocare tutti i task quando ne viene creato uno nuovo

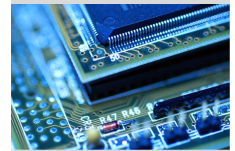
## Algoritmo FFDU

Un altro algoritmo per l'allocazione dei task è **FFDU** (**F**irst **F**it **D**ecreasing **U**tutilization, Davari & Dhall, 1986):

- 1 ordina i task per fattori di utilizzazione decrescenti:  $T_1, T_2, \dots$
- 2 ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- 3 cominciando da  $T_1$ , assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **RM**

- $U_{\text{FFDU}} = m \cdot (\sqrt{2} - 1)$  (Lopez & al., 2003)
- Fattore di approssimazione: **1.67** (Oh & Son, 1995)

Poiché richiede l'ordinamento dei task, **FFDU** è tipicamente utilizzato come algoritmo off-line



## Algoritmo RM-FF

Una variante di **RMFF** è l'algoritmo **RM-FF** (Oh & Son, 1994) che sostanzialmente non effettua l'ordinamento dei task prima della allocazione:

- 1 ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- 2 assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **RM**

- $U_{\text{RM-FF}} = m \cdot (\sqrt{2} - 1)$  (Oh & Baker, 1998)

- Fattore di approssimazione: **2.33** (Oh & Son, 1994)

A differenza di **RMFF**, **RM-FF** è facilmente utilizzabile come algoritmo on-line

## Algoritmo EDF-FF

L'euristica "first fit" accoppiata all'algoritmo di schedulazione **EDF** dà luogo all'algoritmo di allocazione "on-line" **EDF-FF**:

- 1 ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- 2 assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **EDF**

- $U_{\text{EDF-FF}} = \frac{\beta \cdot m + 1}{\beta + 1}, \quad \beta = \left\lfloor 1 / \max_k \frac{e_k}{\rho_k} \right\rfloor$  (Lopez & al., 2000)

- Fattore di approssimazione: **1.7** (Garey & Johnson, 1979)

**EDF-FF** è ottimale tra tutti gli algoritmi partizionati:

$$\beta = 1 \implies U_{\text{EDF-FF}} = (m + 1)/2$$

