Sistemi Operativi: Sistemi realtime

Amos Brocco, Ricercatore, DTI / ISIN

Basato su:
[STA09] “Operating Systems: Internals and Design Principles”, 6/E, William Stallings, Prentice Hall, 2009
[TAN01] “Modern Operating Systems”, 2/E, Andrew S. Tanenbaum, Prentice Hall, 2001
[TAN09] “Modern Operating Systems”, 3/E, Andrew S. Tanenbaum, Prentice Hall, 2009

Cos'è un sistema operativo realtime?

- (RTOS – Real Time Operating System)
- È un sistema operativo multitasking per applicazioni con requisiti real-time
  - In un'applicazione real-time **la correttezza non dipende solo dal risultato logico ma anche dal tempo necessario per ottenere questo risultato**

Cos'è un sistema operativo realtime?

- Un sistema real-time deve essere **prevedibile**:
  - Deve soddisfare dei **criteri di schedulazione** per ogni attività/processo/task da eseguire: se è possibile, il sistema è detto schedulabile
  - Se l'insieme dei processi è dinamico, il sistema deve verificare che i criteri di schedulazione siano sempre soddisfatti

Design dei sistemi real-time

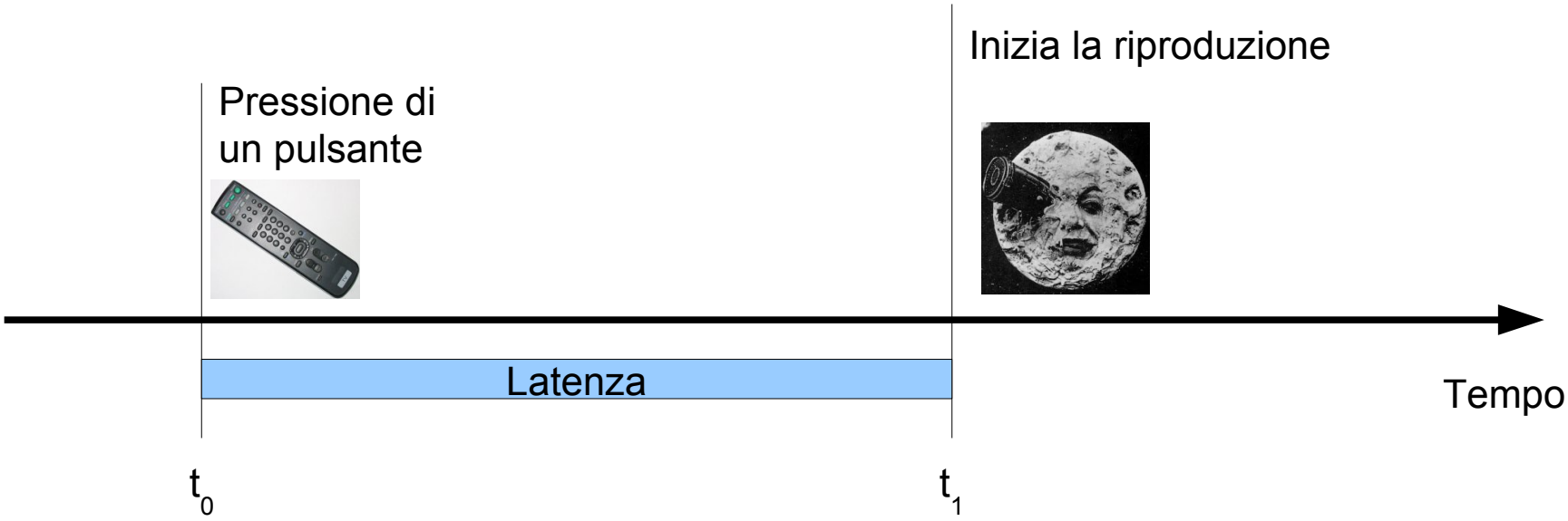
- **A eventi**:
  - Ad ogni evento sono assegnate delle priorità di schedulazione
  - Le attività sono eseguite in risposta a degli eventi
  - Le attività associate a eventi di priorità più alta possono sospendere quelle a priorità più bassa (pre-emption)
- **Time sharing**:
  - Basate su uno scheduler Round-robin
  - ”Comportamento più deterministico”: so esattamente l'ordine con cui verranno eseguite le attività

Attività (task)

- Eseguite in base al tempo (**time triggered**)
  - I task sono eseguiti in un momento determinato o a intervalli regolari (task periodici)
  - Esempio: lettura del valore di un sensore ogni *t* secondi
- Eseguite in risposta ad un evento (**event triggered**)
  - I task sono eseguiti in risposta a un evento (esterno) (task aperiodici)
  - Esempio: l'utente preme un pulsante, una valvola viene aperta

Latenza

- È il tempo che intercorre tra un evento (*stimulus*) e la risposta da parte del software
- Esempio: un riproduttore DVD



Latenza: cause

- **Hardware**:
  - Generazione degli interrupt e propagazione dei segnali attraverso i circuiti
- **Software**:
  - Routine di gestione degli interrupt
  - Algoritmo di schedulazione
  - Context switch
  - Applicazione utente
  - Swapping (se le istruzioni per la gestione dell'evento non sono in memoria)

Prelazione (pre-rilasciabilità, pre-emption)

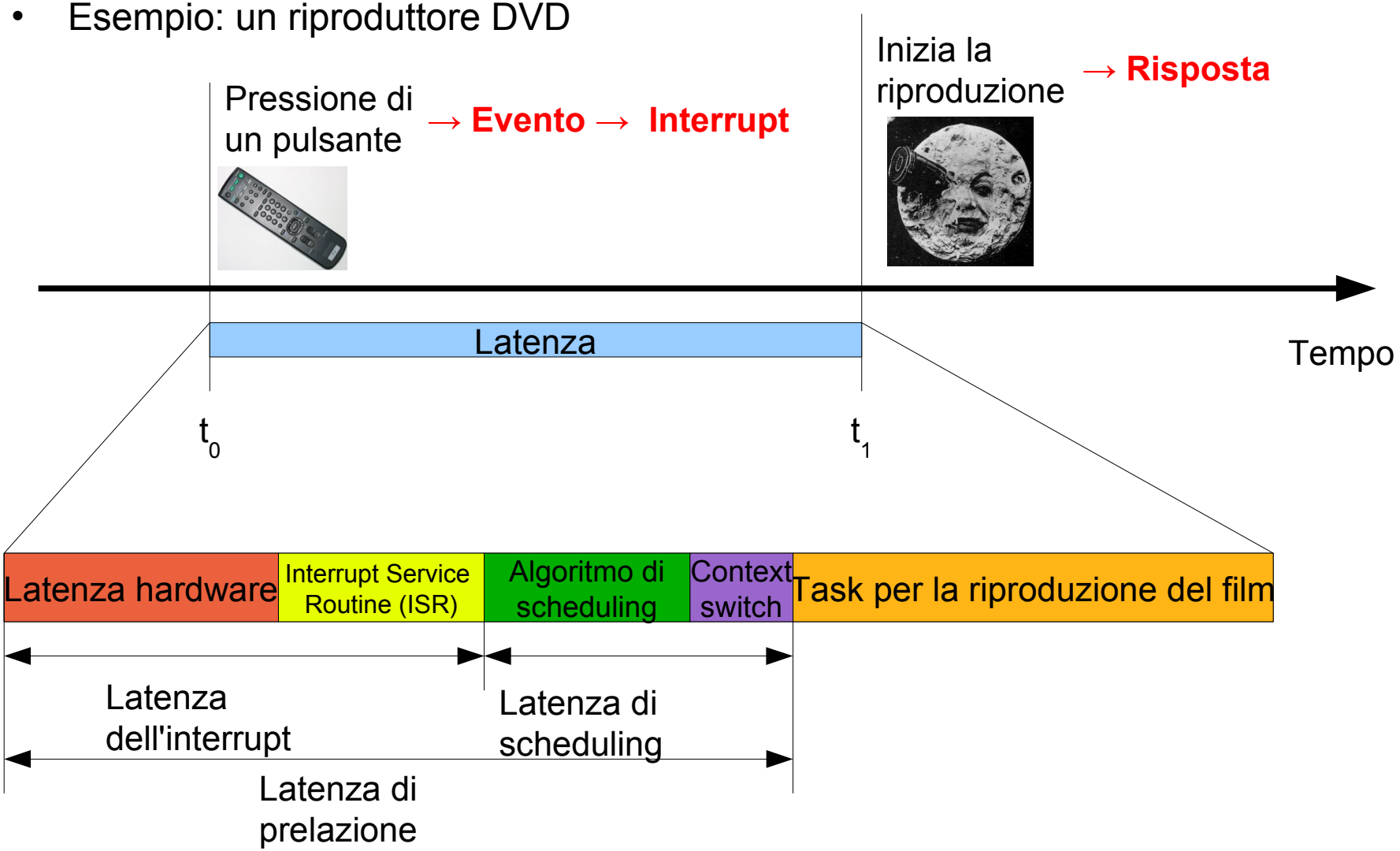
- In un sistema multitasking un altro trask potrebbe essere in esecuzione quando avviene un evento: è tipicamente necessario prelazionare questo task

- La prelazione (**pre-emption**) indica *l'abilità del sistema operativo di sospendere un task attualmente schedulato/in esecuzione per eseguirne un altro a priorità più alta*

- Evento ► Interrupt ► Gestione interrupt ► Pre-emption del task corrente ►Task Switching ► Risposta

Latenza di prelazione (pre-emption latency)

- Esempio: un riproduttore DVD



Tipi di requisiti real-time

- **Hard Real-Time**:
  - I task **devono** essere completati entro una deadline
  - Oltrepassare una deadline non può essere tollerato: se la risposta arriva troppo tardi è inutile e pericoloso per la stabilità del sistema
  - Esempi: sistemi di controllo (autopilota degli aeroplani, controllo di un reattore nucleare,...)

Tipi di requisiti real-time

- **Firm Real-Time**:
  - Oltrepassare una deadline non è pericoloso: se la risposta arriva troppo tardi viene semplicemente ignorata
  - Un task non può continuare l'esecuzione dopo la deadline: se il task non ha finito, il risultato è ignorato
  - Esempio: sistemi multimediali (pipeline di rendering video in real-time: per mantenere la sincronizzazione con l'audio alcuni frame possono essere ignorati)

Tipi di requisiti real-time

- **Soft Real-Time**:
  - I task devono essere completati entro una deadline, ma le eccezioni sono tollerate
  - Perdere una deadline non comporta seri problemi al sistema o conseguenze gravi
  - Esempi: DVD Player (perdere una deadline potrebbe semplicemente causare un degrado nell'immagine)

Schedulazione e schedulabilità

- **Schedulabilità**: condizione per le deadlines sono rispettate per ogni task

- Obiettivo di un RTOS: garantire che le **deadlines** (momento per prima del quale il task deve aver terminato l'esecuzione), garantire la schedulabilità dei task

- Hard real-time:
  - Scheduling statico o dinamico
- Soft real-time:
  - Scheduling con priorità

Scheduling statico

- Il numero di task e l'ordine di schedulazione sono definiti *a priori*
- A runtime, lo scheduling è semplice e richiede poco tempo, ma...
- Il sistema **non è flessibile**:
  - Non è possibile aggiungere o rimuovere task
  - Ogni cambiamento richiede una rivalutazione off-line dell'ordine di scheduling per garantire la schedulabilità

Scheduling dinamico

- I task possono essere aggiunti o rimossi dinamicamente dalla coda di scheduling
- Il sistema deve garantire la validità della coda di esecuzione ad ogni cambiamento (*"il sistema è ancora schedulabile?"*)
- Questo approccio è **flessibile** ma...
  - È difficile garantire la correttezza

Scheduling con priorità

- L'ordine di esecuzione dei task è determinato dalla loro priorità
- Le priorità possono essere modificate a runtime
- Offre delle performance ”best-effort”, ma...
  - L'ordine esatto dello scheduling potrebbe essere imprevedibile

Scheduling periodico

- Tutti i task t_i hanno delle deadline (hard) periodiche
- Tutti i task sono indipendenti e hanno un **tempo di esecuzione** conosciuto $X(t_i)$
- Ogni task è schedulato periodicamente con periodo P_i, e ha un **intervallo di deadline** $D(t_i)$ (ovvero deve completare l'esecuzione in un tempo $D(t_i)$)
- Il tempo **slack** o **laxity** L_i è definito come

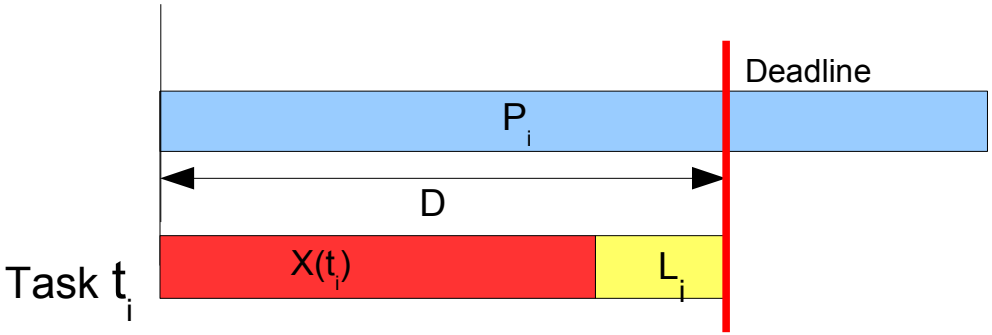
$$L_i = D(t_i) - X(t_i)$$

Scheduling periodico

- **Accumulated utilization**:

$$u = \sum_{i}^{n} X(t_i)/P_i$$

- In un sistema con m processori, abbiamo la schedulabilità se u ≤ m



Rate Monotonic Scheduling (RMS) (1)

- È un esempio di algoritmo di schedulazione statica basata sulle priorità:
  - Ogni task ha una priorità assegnata durante lo sviluppo del sistema (design time), utilizzando l'algorimo rate monotonic (RMA)
- RMA assegna delle priorità fisse ai task periodici in modo da massimizzare la schedulabilità
  - Idea: ”assegnare la priorità ad un task in base al periodo, in modo che **i task con periodo più corto ricevano una priorità più alta**”

Rate Monotonic Scheduling (RMS) (2)

- Consideriamo:
  - Dei task indipendenti e periodici
  - L'intervallo di deadline equivale al periodo ($D_i = P_i$)
  - Il tempo richiesto per l'esecuzione è conosciuto e costante
  - Possiamo ignorare il tempo richiesto per il cambio di contesto

Accumulated utilization per n task

$$u = \sum_{i}^{n} X(t_i)/P_i \leqslant n(2^{1/n} - 1)$$

Condizione di schedulabilità con RMS

Rate Monotonic Scheduling (RMS) (3)

- RMA è un **algoritmo a priorità statiche ottimale**:
  - Se un insieme di task non è schedulabile con RMA, non potrà esserlo con nessun altro algoritmo a priorità statiche

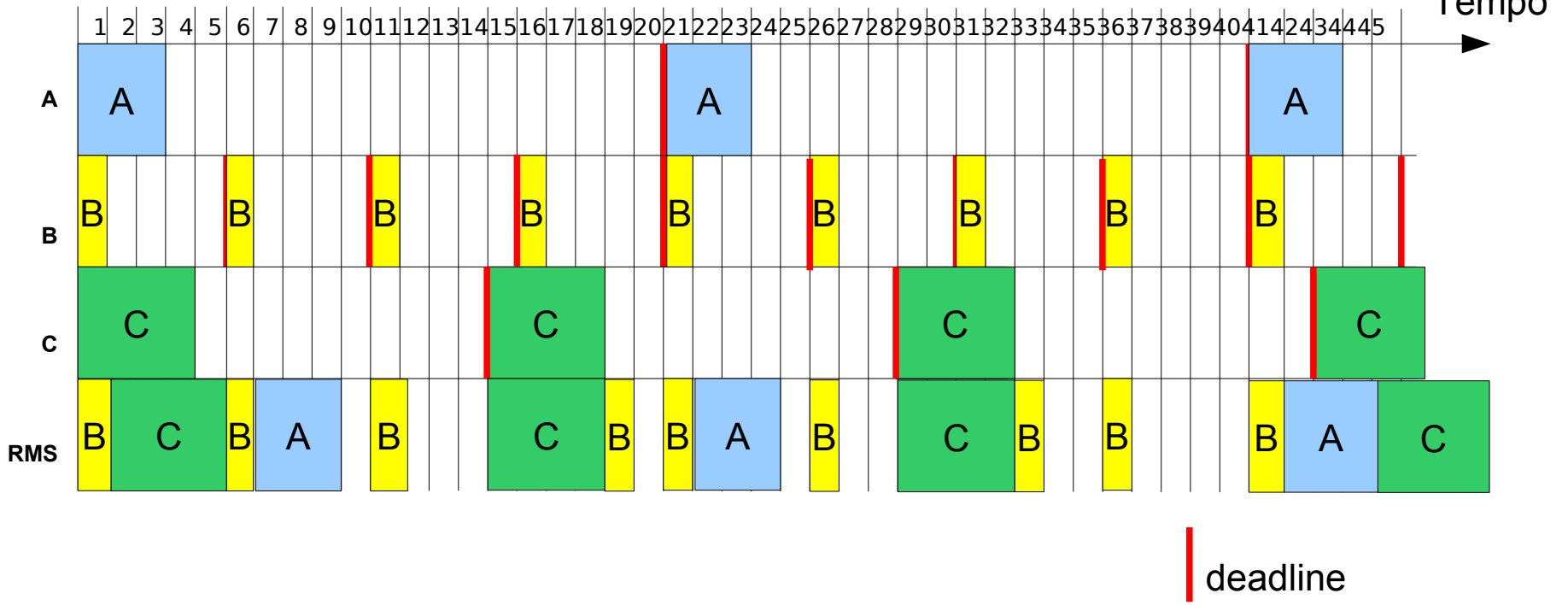
- Peggior caso per 1 CPU, e n task: $W_s = n(2^{n/1} - 1)$
  n $\rightarrow \infty$, $W_s \rightarrow 69\%$

Esempio di scheduling RMS (1)

- 3 task periodici: A (esecuzione 3) ,B (1) ,C (4)
- Periodi: A = 20, B = 5, C = 14
- Priorità: A = 1/20, B = 1/5, C = 1/14 (ordine B,C,A)
- U = 3 / 20 + 1 / 5 + 4 / 14 = ~0.63 = 63%

Il periodo più corto da una priorità più alta

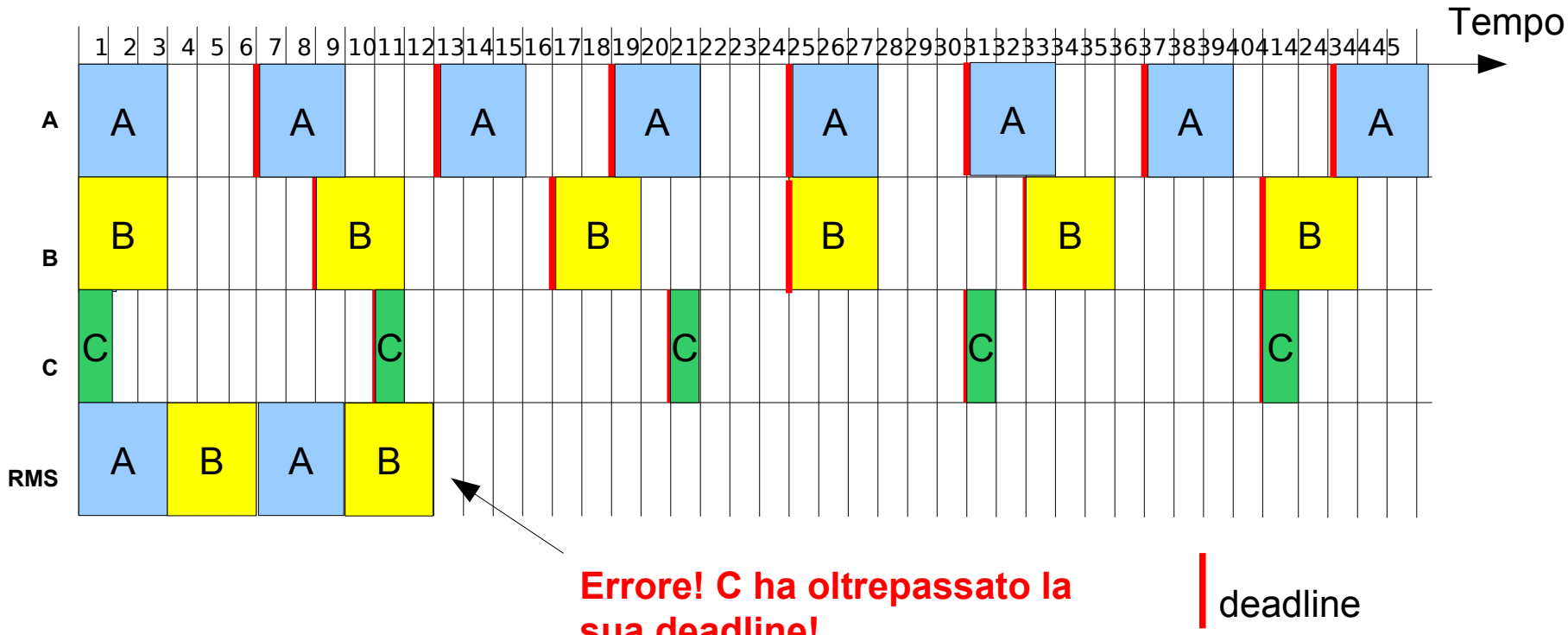
≤ 1 i.e. Schedulabile su 1 CPU,
≤ 0.77 i.e. Schedulabile con RMS
(3 *(2 ^ (1/3) – 1)) = 0.77



Esempio di scheduling RMS (2)

- 3 task periodici: A (3) ,B (**3**) ,C (1)
- Periodi: A = **6**, B = **8**, C = **10**
- Priorità: A = 1/**6**, B = 1/**8**, C = 1/**10** (ordine A,B,C)
- U = 3 / **6** + **3** / 8 + 1 / **10** = ~0.97 = 97%

≤ 1 i.e. Schedulabile su 1 CPU,
> 0.77 i.e. **NON schedulabile con RMS**

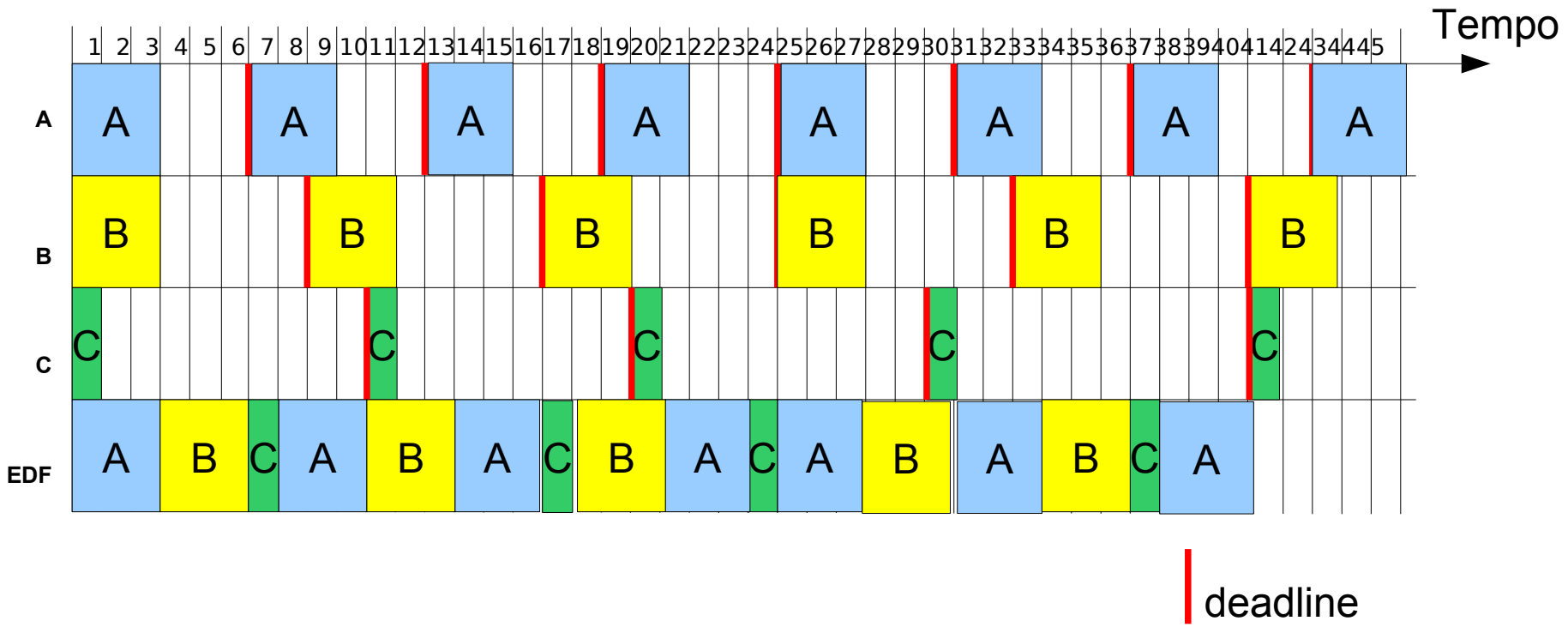


Scheduling Earliest Deadline First (EDF)

- Scheduling con priorità dinamiche
- **Il task che ha la deadline più vicina acquista una priorità più alta**
- Tutte le deadline sono soddisfatte se l'utlizzo della CPU è al massimo 100%
  - Se il sistema è sovraccarico, lo scheduling diventa imprevedibile
- Può essere utilizzato anche per task non-periodici

Esempio di scheduling con EDF

- 3 task periodici: A (3) ,B (3) ,C (1)
- Periodi: A = 6, B = 8, C = 10
- U = 3 / 6 + 3 / 8 + 1 / 10 = ~0.97 = 97% ← ≤ 1 i.e. Schedulabile con 1 CPU



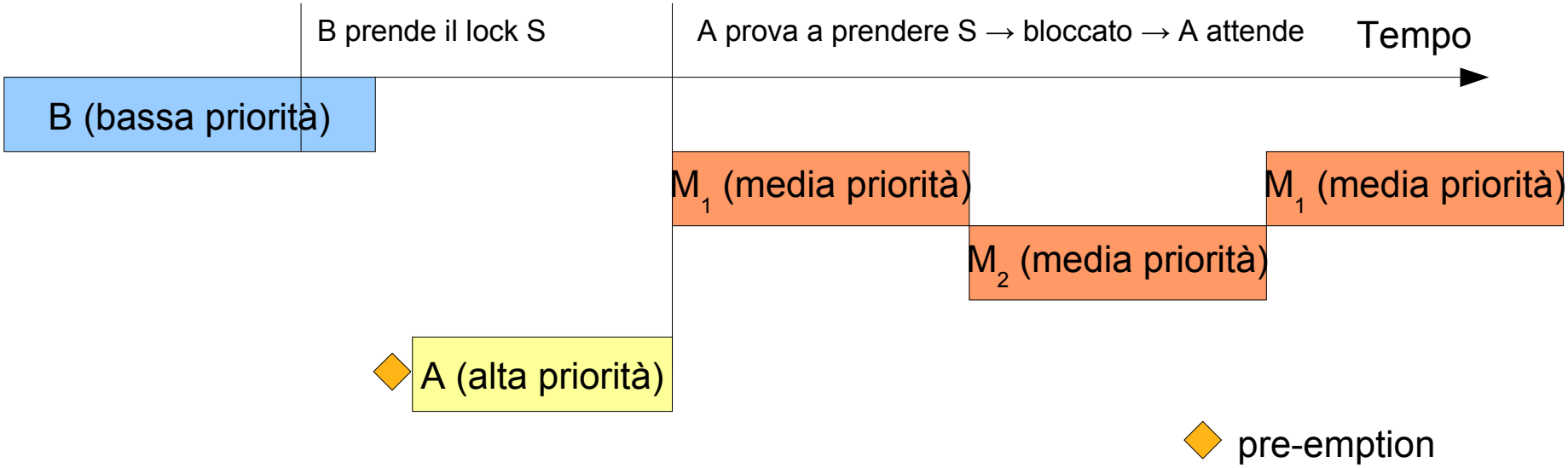
Scheduling Least slack time

- Anche conosciuto come scheduling Least Lexity Time
- Le priorità sono basate sullo slack time dei task: i task che hanno uno slack time più basso hanno una priorità più alta

Il problema dell'inversione della priorità (1)

- Consideriamo questa situazione:
  - Un task a bassa priorità B acquisisce un lock
  - Un task a alta priorità A cerca anche lui di acquisire il lock, ma è messo in attesa
  - Un task a media priorità M è messo in esecuzione dall'altoritmo di scheduling (pre-emption) invece di quello a bassa priorità B
  - Il task a bassa priorità B non verrà più eseguito fintanto che M è pronto, ma così facendo non permetterà mai a A di ottenere il lock e quindi di continuare l'esecuzione

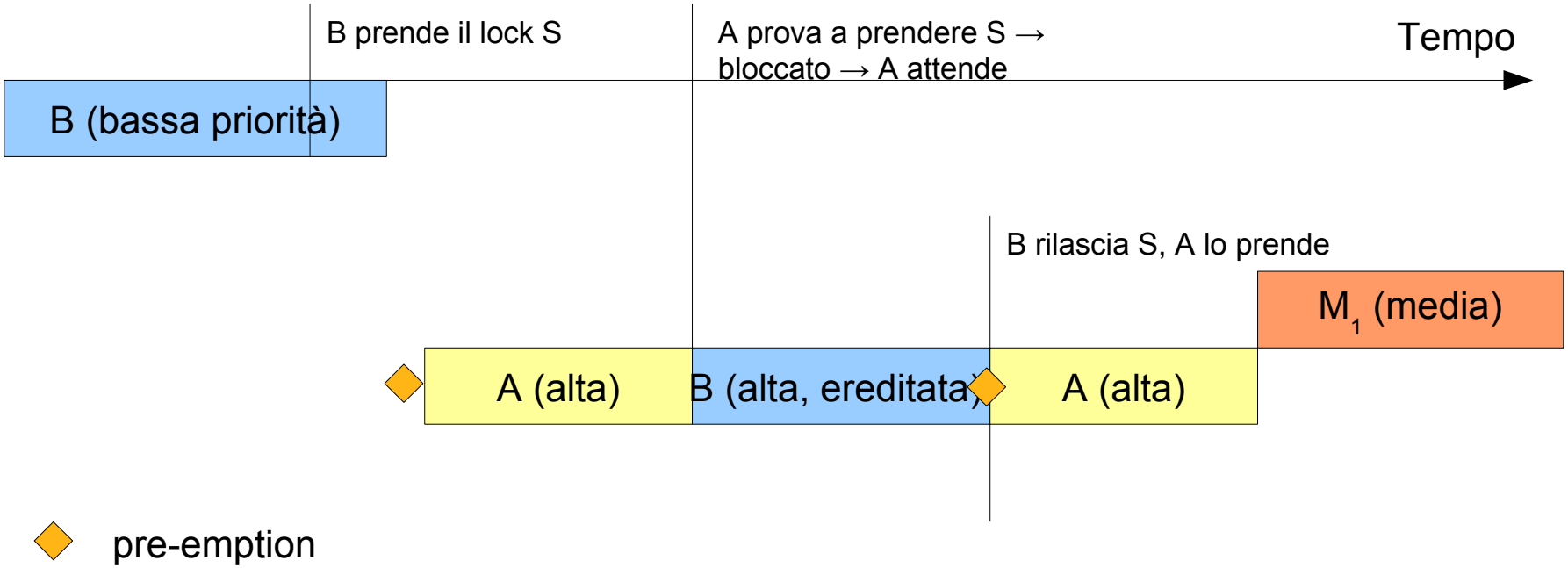
Il problema dell'inversione della priorità (2)



B non viene più schedulato perchè ha una priorità bassa: il lock non sarà mai rilasciato e H perde “il suo vantaggio” non potendo continuare

Il problema dell'inversione della priorità (3)

- Soluzione: **ereditare la priorità**
  - Un processo che acquisisce un semaforo o lock eredita, temporaneamente, la priorità del processo con priorità più alta che è in attesa



Altre problematiche dei RTOS: gestione della memoria

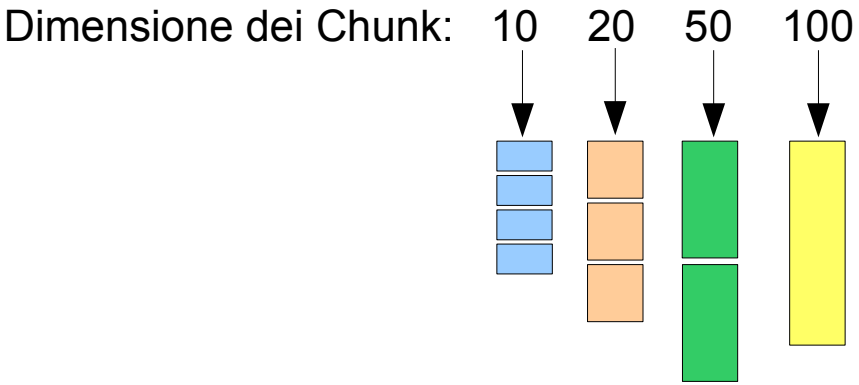
- **Obiettivo:** minimizzare l'overhead della gestione della memoria
- Una gestione della memoria efficiente è critica per un sistema RTOS
  - I RTOS sono tipicamente utilizzati su piattaforme embedded, con memoria limitata
- Punti chiave: velocità di allocazione → evitare la frammentazione

Frammentazione della memoria (1)

- La frammentazione avviene quando un task libera della memoria
- Un sistema di garbage collection o deframmentazione della memoria non è (sempre) possibile
  - Richiede tempo e risorse CPU
  - Non è deterministico (non è possibile sapere quando il GC verrà eseguito, perché dipende dall'ordine di scheduling)

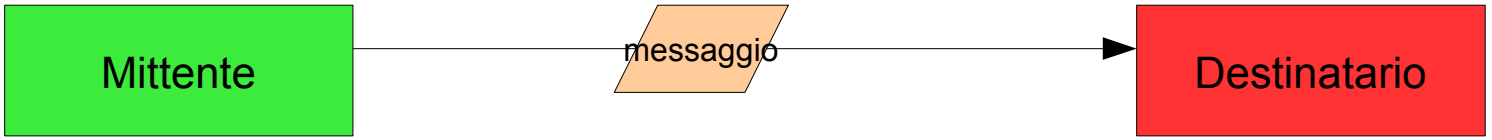
Frammentazione della memoria (2)

- Per evitare la frammentazione i RTOS allocato tipicamente della memoria in blocchi (chunk) di dimensione fissa
- Più dimensioni differenti possono essere offerte:
  - Per ogni dimensione, abbiamo degli insiemi separati di blocchi allocabili
    - Quando un blocco viene liberato lo si rimette nell'insieme corrispondente
  - Il tempo di allocazione e deallocazione è praticamente costante



Comunicazione tra task

- Per evitare i problemi deadlock possiamo utilizzare dei meccanismi di scambio dei messaggi per ottenere la comunicazione tra task (anziché memoria condivisa e deadlock)
  - La maggior parte dei RTOS offre un sistema di message passing di basso livello con prestazioni elevate
  - I messaggi possono essere utilizzati anche per la sincronizzazione, es.
    - Un task A passa a B (via messaggio) un token per l'utilizzo di una risorsa C



Riferimenti

- http://en.wikipedia.org/wiki/Real-time_operating_system
- http://camars.kaist.ac.kr/~maeng/cs310/embedded07/11-Processes-scheduling.pdf
- http://linuxdevices.com/articles/AT4627965573.html
- http://www.dei.unipd.it/corsi/so/no/store/u12aRtos.pdf
- http://www.slideshare.net/dadaista/scheduling/
- Stewart, David and Michael Barr. "Rate Monotonic Scheduling" Embedded Systems Programming, March 2002, pp. 79-80.
- A. S. Tanenbaum, Modern Operating Systems, 2nd Edition, Prentice Hall, 2001.