Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

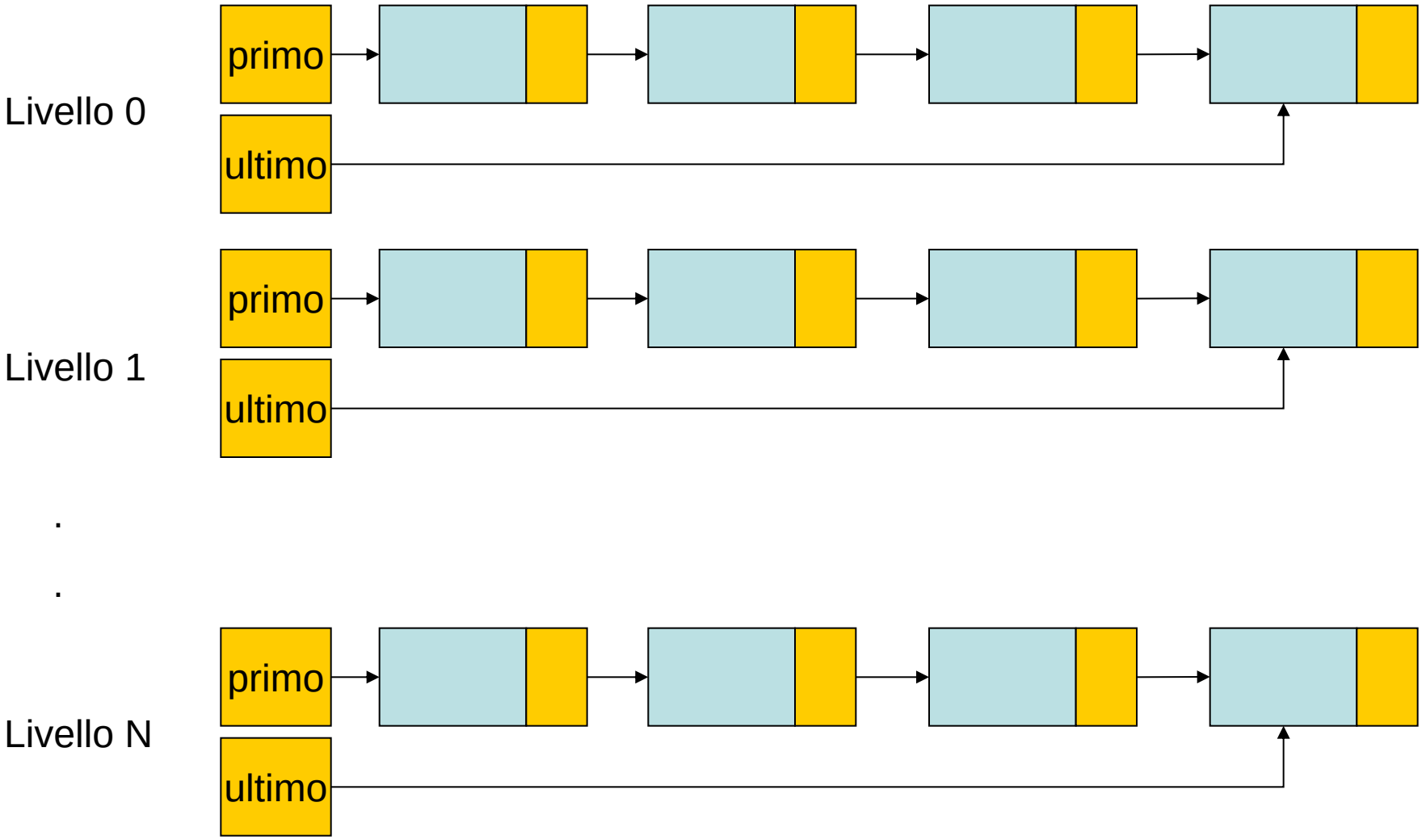
Lezione 8

Martedì 8-11-2016

Algoritmi di scheduling basati sulle priorità

- Assegnano la CPU al processo pronto con priorità più alta. I processi con le stesse priorità si ordinano con politica **FCFS**.
- Le priorità possono essere statiche o dinamiche.
  - **Priorità statiche** se non si modificano durante l'esecuzione dei processi;
  - **Priorità dinamiche** se si modificano durante l'esecuzione dei processi, in base a qualche criterio come ad esempio in base al tempo di utilizzo di cpu.
- La coda di pronto può essere suddivisa in varie code, ciascuna relativa ad un valore di priorità. In questo caso l'algoritmo di scheduling seleziona il processo pronto dalla coda a più alta priorità, applicando all'interno di ogni coda un tipo di algoritmo, ad esempio Round Robin.
- Se non si utilizzano priorità dinamiche si corre il rischio che processi con bassa priorità attendano tempi enormi al limite infiniti (**starvation**).

Scheduling basato sulle priorità

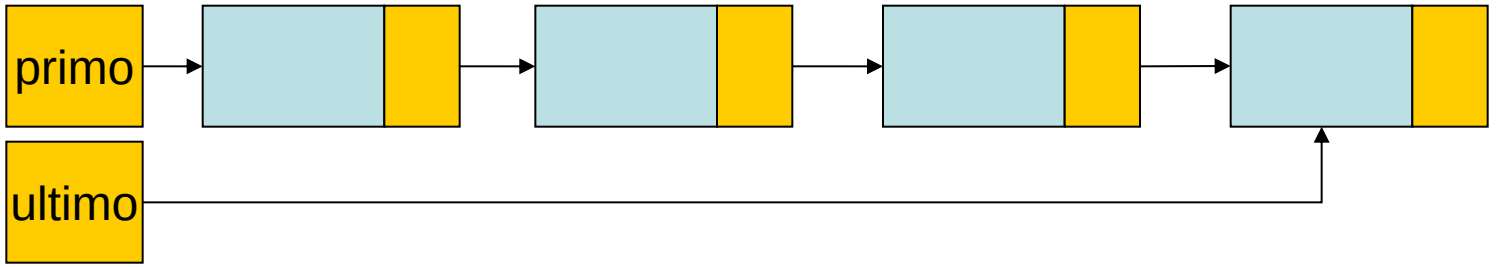


Algoritmi di scheduling a code multiple

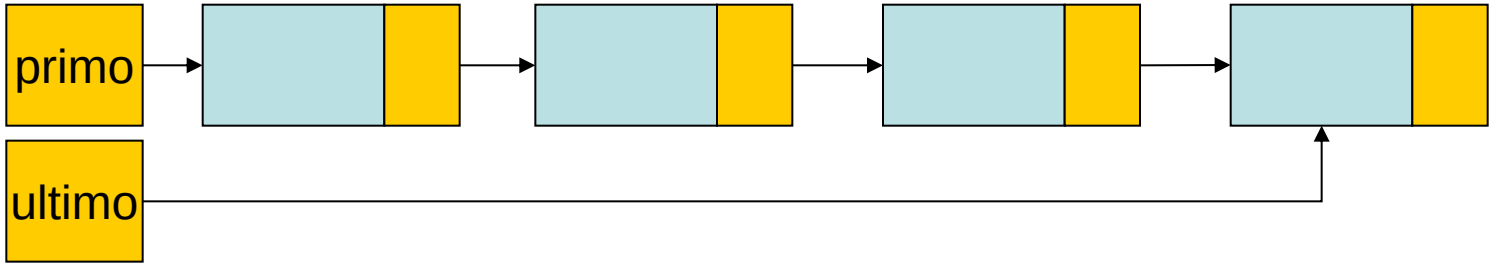
- Nei SO complessi sono in esecuzione processi con caratteristiche diverse.
- Sono presenti processi CPU-bound e IO-bound, oppure processi interattivi e processi tipo “batch” detti rispettivamente processi **foreground** e **background**.
- Pertanto i processi pronti sono inseriti in varie code, ciascuna con diversa priorità. Lo scheduler seleziona per primo un processo dalla coda a più alta priorità.
- Nel caso più semplice sono presenti due code una (livello 0) per i processi interattivi che viene gestita con RR e l’altra (livello 1) a priorità inferiore per i processi background che viene gestita con FCFS.
- Casi più complessi prevedono molte code a diverse priorità in cui i processi possono passare, durante la loro esecuzione da una coda all’altra (multilevel feedback queue)

Scheduling basato sulle code multiple

Livello 0 coda RR con quanto = 20 ms

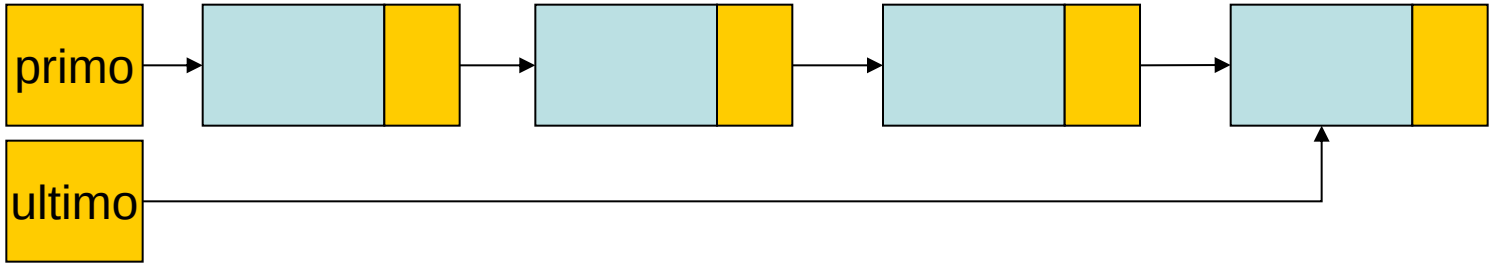


Livello 1 coda RR con quanto = 50 ms



.

. Livello 2 coda FCFS per processi background



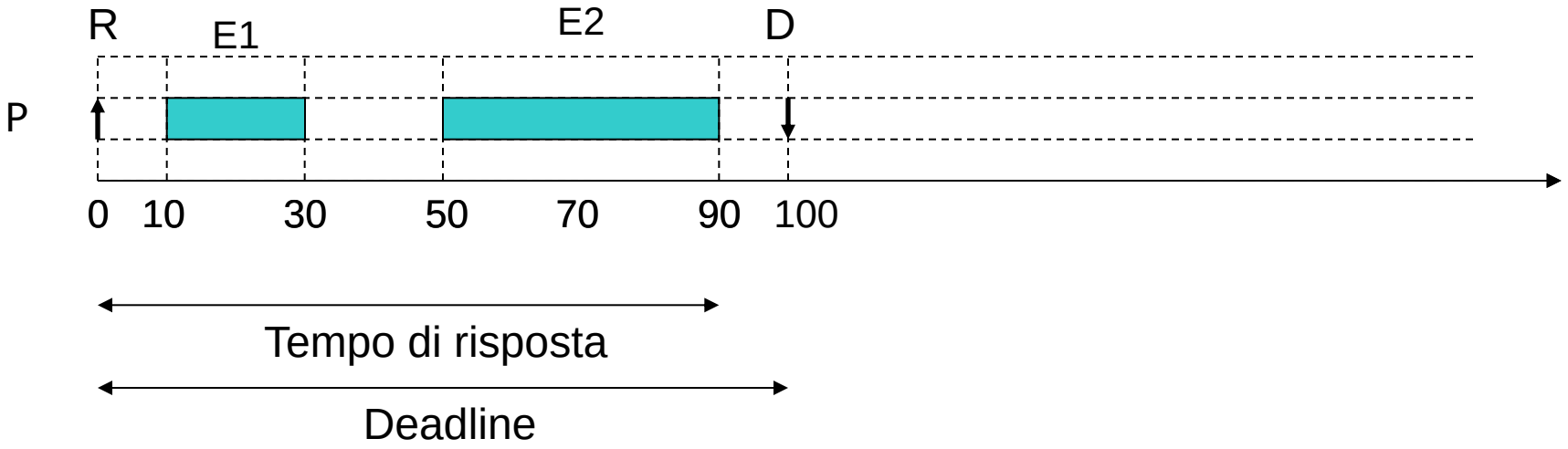
Algoritmi di scheduling real-time

- Generalmente i SO real-time utilizzano algoritmi di scheduling basati sulle priorità. Gli algoritmi possono essere statici o dinamici.
- Gli algoritmi statici assegnano le priorità ai processi in base alla conoscenza di alcuni parametri temporali dei processi noti all'inizio. Al contrario gli algoritmi dinamici cambiano la priorità dei processi durante la loro esecuzione.
- I processi real-time fondamentalmente sono caratterizzati dai seguenti parametri:
  - **istante di richiesta**: l'istante in cui il processo entra nella coda di pronto.
  - **deadline**: istante entro il quale il processo deve essere terminato.
  - **tempo di esecuzione**: tempo di CPU necessario al processo per svolgere il suo lavoro.

Esempio

Per il processo in figura si ha:

- Istante di richiesta **R** = 0;
- tempo di esecuzione **E** = E1 + E2 = 20 + 40 = 60
- deadline **D** = 100
- Tempo di risposta = 90

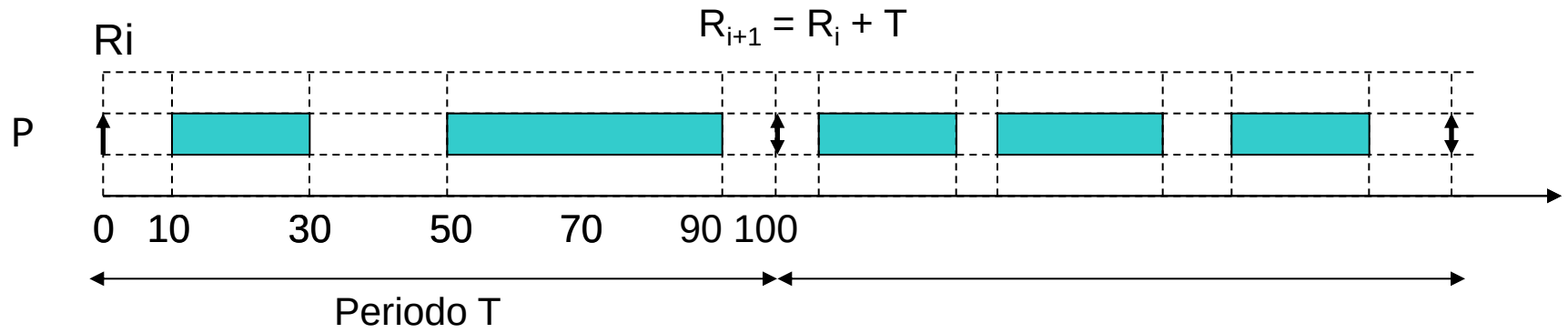


- I processi real-time possono essere **periodici** o **aperiodici**. I processi periodici vengono attivati ciclicamente a periodo costante che dipende dalla grandezza fisica che il processo deve controllare.
- I processi non periodici vengono avviati in situazioni imprevedibili.
- Consideriamo un algoritmo di scheduling per processi periodici.

  In tal caso deve essere:

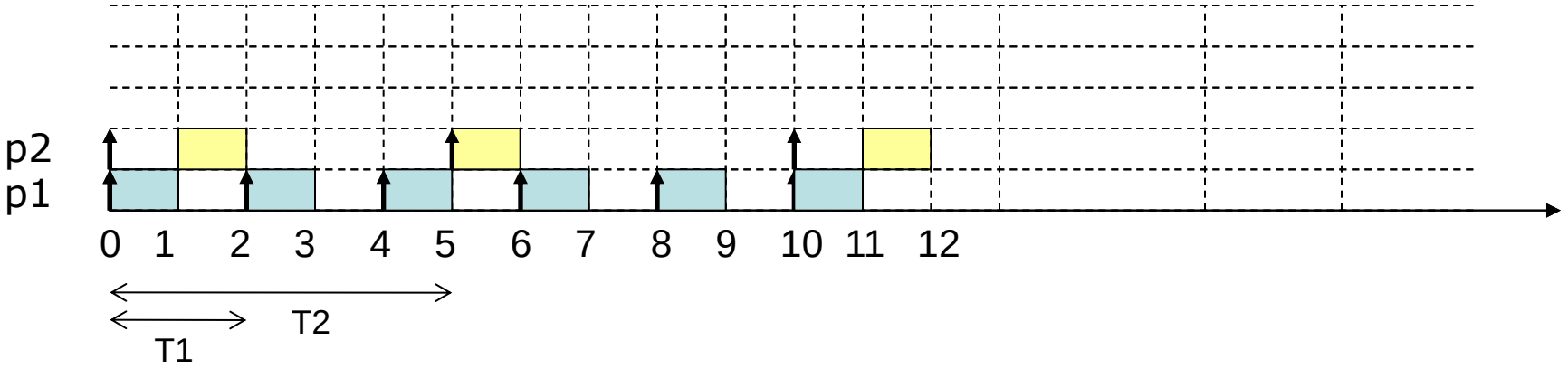
  $R_{i+1} = R_i + T$

  $T >= D$

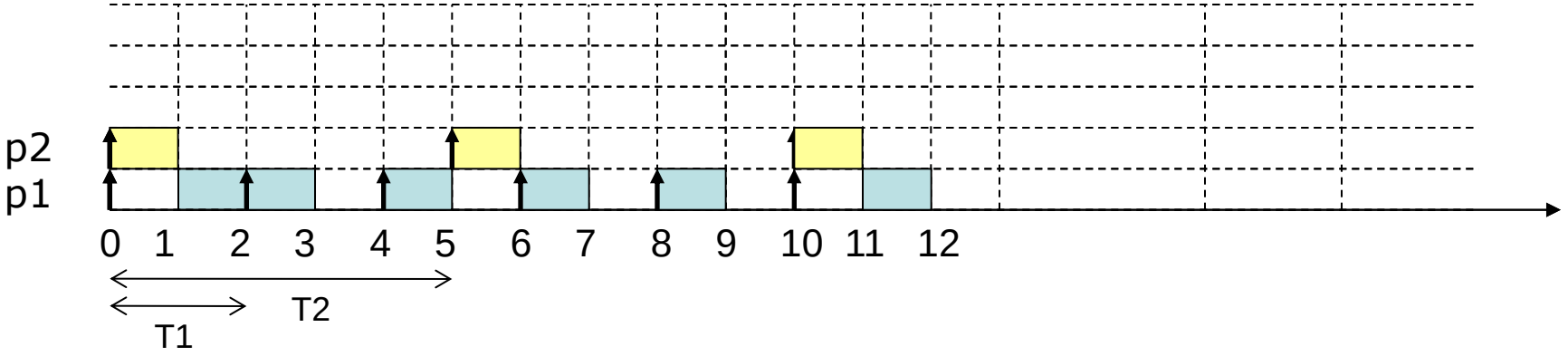


- Dato che in ciascun periodo il tempo di esecuzione di un processo potrebbe cambiare, indichiamo con $\mathbf{E_{max}}$ il tempo massimo di esecuzione di un processo in ciascun periodo **T**.
- $\mathbf{E_{max}}$ e **T** sono tempi noti a priori, imposti dall'applicazione real-time.
- Nei SO RT generalmente si utilizzano algoritmi di scheduling con priorità.
- Il problema è come assegnare la priorità ai processi.
- L'algoritmo ***Rate Monotonic (RM)*** è un algoritmo ottimo, nell'ambito della classe degli algoritmi basati sulle priorità statiche. E' un algoritmo preemptive. Esso assegna la priorità ai processi in base alla durata del loro periodo. In particolare priorità maggiore ai processi che hanno periodo minore.
- Per mostrare la validità di tale criterio, consideriamo un esempio in cui due processi P1 e P2 hanno i seguenti parametri temporali:

P1: T1 = 2; E1 = 1
P2: T2 = 5; E2 = 1



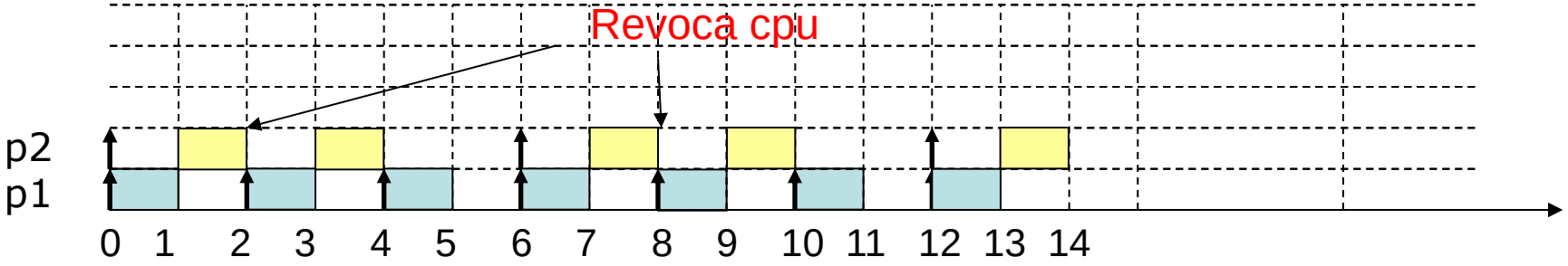
Priorità P1 > priorità P2



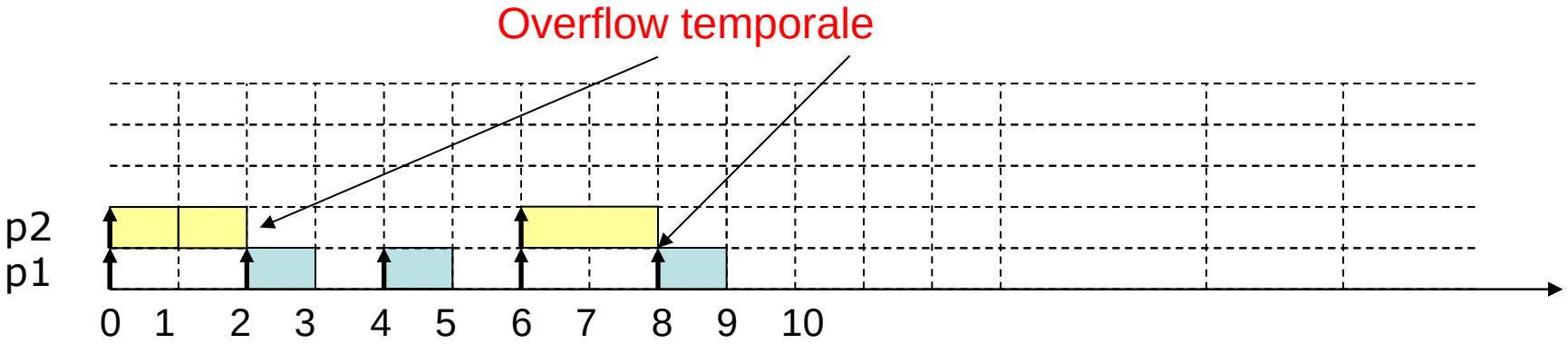
Priorità P2 > priorità P1

Supponiamo ora che sia E2 = 2 T2=6, mentre sia ancora E1 =1 e T1 = 2.

Si ha che nel primo caso i due processi sono ancora schedulabili mentre nel secondo caso no.



Priorità P1 > priorità P2



Priorità P2 > priorità P1

- Tuttavia, anche adottando un criterio ottimo come RM è necessario (ma non sufficiente) che sia verificata la seguente relazione:

  $U = \Sigma (E_i/T_i) <= 1$

  Dove **U** è detto **coefficiente di utilizzazione** della CPU.
- E' stato dimostrato, utilizzando ***Rate Monotonic*** , che affinchè un insieme di **N processi** sia schedulabile è sufficiente che il coefficiente di utilizzazione sia:

  $U <= N(2^{1/N} - 1)$

  Ad esempio per N=4 si ha U=0.76

  per N=50 U=0.698

  per N=100 U=0.695

  per N->infinito U -> ln 2 = 0,69314718055994530941723

Sincronizzazione tra processi/thread

- Come descritto in precedenza, i processi o i thread possono interagire tra loro in due modi: ***cooperazione* e *competizione***.

Cooperazione

- Generalmente la cooperazione tra processi avviene mediante operazioni di sincronizzazione e comunicazione.
- Il modello produttore-consumatore è molto usato per la comunicazione tra processi. In questo paradigma due processi, l'uno detto ***produttore*** produce messaggi e li scrive in un buffer comune e l'altro detto ***consumatore*** legge i messaggi dal buffer e li elabora (consuma).
- Generalmente i processi per cooperare devono sincronizzare le loro attività nel tempo.

Competizione

- La competizione si ha quando i processi richiedono risorse comuni che non possono essere usate contemporaneamente, come ad esempio una struttura dati, un file o un dispositivo.
- Sia per la cooperazione che per la competizione, è necessario che le operazioni eseguite dai processi sulle **risorse comuni** siano effettuate in ***mutua esclusione nel tempo***.

1. L'interazione tra processi si ottiene mediante diversi **strumenti di sincronizzazione** la cui scelta dipende dal tipo di **modello di interazione** tra i processi:

    1. **Modello a memoria comune (ambiente globale)**
    2. **Modello a scambio di messaggi (ambiente locale)**

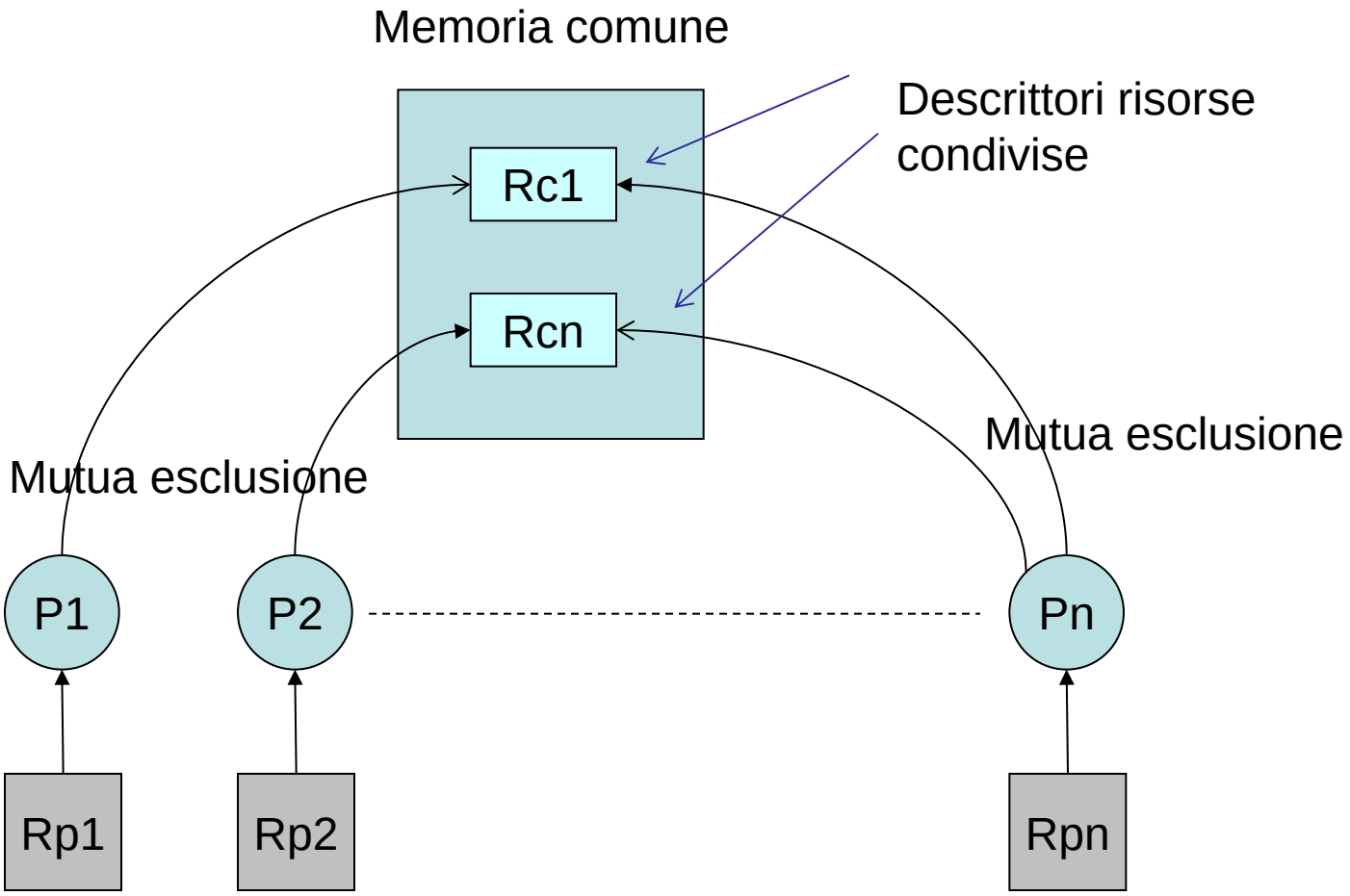
Modello a memoria comune (ambiente globale)

- L'interazione, sia competizione che cooperazione, tra i processi avviene tramite memoria condivisa.
- Generalmente, questo modello è usato in architetture di calcolatori sia con un solo processore che multiprocessore. In quest'ultimo caso i processori sono collegati e condividono un'unica memoria nella quale saranno allocate le risorse comuni.
- Ogni risorsa è rappresentata con una struttura dati che si alloca in un area di memoria condivisa. Ad esempio, un dispositivo di I/O, è rappresentato da un struttura dati detta **descrittore del dispositivo** che rappresenta le sue proprietà e il suo stato.
- Per un processo, una risorsa può essere **privata o condivisa (o globale)**. Nel primo caso il solo processo proprietario può operare su di essa, mentre nel secondo caso è accessibile a più processi.

- Uno strumento di sincronizzazione in questo modello è il **semaforo** e le chiamate di sistema di sincronizzazione **wait** e **signal**.

Indirizzo registro di controllo
Indirizzo registro dati
Indirizzo registro di stato
Semaforo di sincronizzazione **`dato_pronto`**
Contatore num. dati da trasferire **`contatore`**
Indirizzo del buffer **`puntatore`**
Risultato del trasferimento **`stato`**

Esempio di descrittore di dispositivo

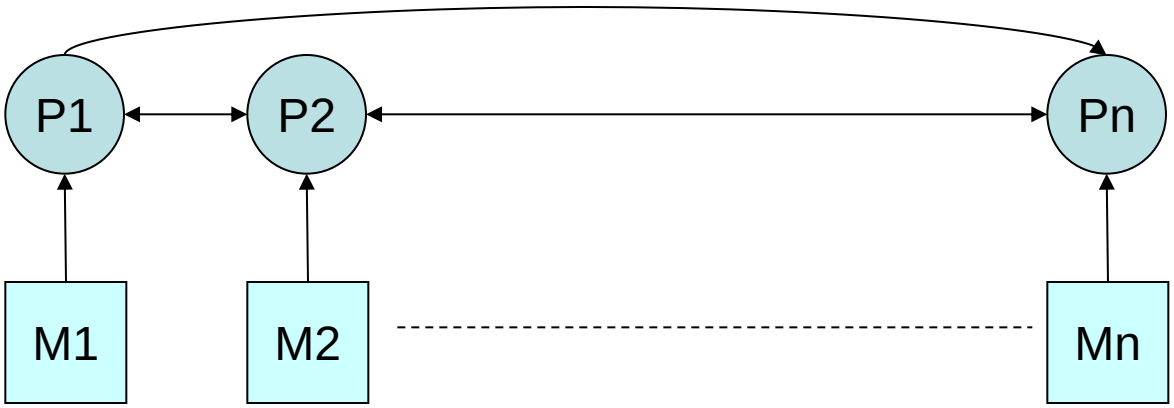


Interazione tra processi in sistemi a memoria comune

Modello ad ambiente locale

- Ogni processo può vedere ed accedere solo alle proprie risorse locali che non sono accessibili agli altri processi. I processi interagiscono esclusivamente tramite **scambio di messaggi**.
- Questo modello è generalmente usato nelle reti di calcolatori, ma può essere anche usato in sistemi mono o multi processore a memoria condivisa. Lo scambio di messaggi avviene sia tramite rete di comunicazione che tramite segmenti di memoria comune.
- Un esempio di strumento di sincronizzazione è dato dalle chiamate di sistema **send** e **receive**.
- Le chiamate send e receive, così come il semaforo con le chiamate wait e signal sono strumenti a basso livello, forniti dal kernel.

Scambio di messaggi



Interazione tra processi in sistemi a memoria locale

Problema della mutua esclusione

- Sia per la cooperazione che per la competizione è necessario che i processi eseguano le operazioni che riguardano le risorse comuni in **mutua esclusione**.
- Con mutua esclusione si intende che le operazioni con le quali i processi accedono alle risorse comuni non si sovrappongano nel tempo.
- Queste operazioni prendono il nome di **sezioni critiche**.

Soluzioni al problema della mutua esclusione

- La soluzione del problema della mutua esclusione consiste nel realizzare un protocollo che i processi devono seguire per interagire correttamente con la risorsa condivisa.
- Un processo, prima di entrare nella sezione critica, dovrà eseguire una serie di operazioni, detta **sezione d'ingresso** (**prologo)**, per assicurarsi l'uso esclusivo della risorsa, nel caso sia libera, oppure ne impediscano l'accesso nel caso sia occupata.
- Al termine dell'esecuzione della sezione critica il processo deve rilasciare la risorsa per consentirne l'allocazione ad altri processi che la richiedono. Per questo dovrà eseguire un'altra serie di operazioni detta **sezione di uscita** (**epilogo)**.

- Un semplice esempio delle operazioni di prologo e di epilogo si ottiene utilizzando una variabile intera condivisa ***occupato*** il cui valore è **1** se la risorsa è occupata o **0** se essa è libera.

Prologo:

```
while (occupato == 1);
occupato = 1;
<sezione critica>
```

Epilogo:

```
occupato = 0;
```

- Affinché la soluzione sia valida è necessario che il SO permetta ai processi di eseguire le istruzioni di **lettura e scrittura** della variabile di controllo condivisa in **modo atomico**.

- Molti processori possiedono istruzioni che consentono di leggere e modificare il contenuto di una locazione in un unico ciclo di memoria. Un esempio è dato dall'istruzione **TSL** (**Test and Set Lock**).
- L'istruzione **TSL R, X** copia il contenuto della locazione di memoria **X** nel registro **R** del processore e viene scritto in X un **valore diverso da 0**.
- Nel caso di sistemi multiprocessore, la CPU che esegue la TSL **blocca il bus di memoria** per impedire che altre CPU accedano alla memoria fino a quando non ha completato l'operazione di TSL.
- La mutua esclusione si ottiene introducendo due funzioni **`lock(x)`** e **`unlock(x)`**:

- **Lock(x):**

```
LOCK:
   TSL R, X    copia il valore di X in R e pone X=1
               (R=X;X=1)
   CMP R,0     verifica se R==0
   JNE LOCK    se R!=0 riesegue il ciclo
   RET         ritorno
```

- **Unlock(x)**

```
  mov x,0      scrive in X il valore 0
  RET          ritorna al chiamante
```

Esempio di due processi

```
P1
Prologo:   lock(x)
           <sezione critica P1>
Epilogo:   unlock(x)

P2
Prologo:   lock(x)
           <sezione critica P2>
Epilogo:   unlock(x)
```

Questa soluzione è caratterizzata da condizioni di **attesa attiva** dei processi. La soluzione è valida quindi per sistemi multiprocessore ed è limitata al caso di sezioni critiche **brevi**.