Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2019-2020

Pietro Frasca

Lezione 14

Giovedì 21-11-2019

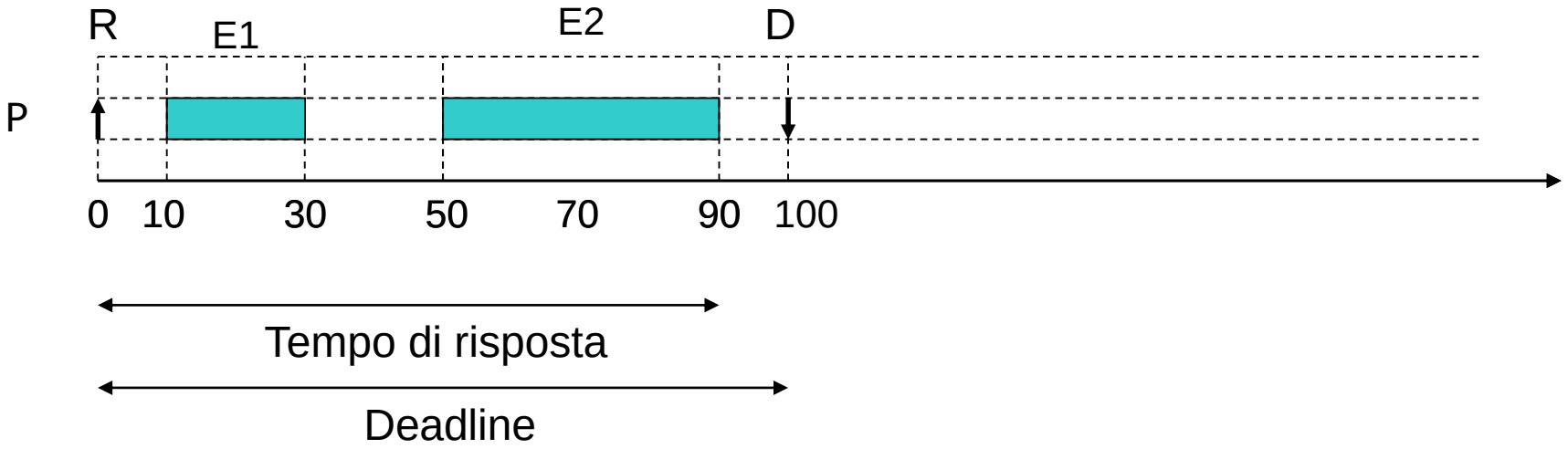
Algoritmi di scheduling real-time

- 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

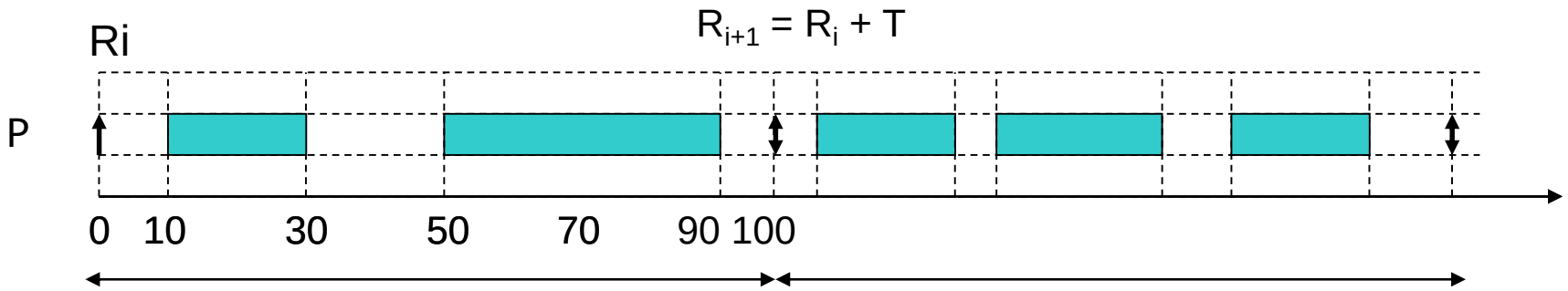


- I processi real-time possono essere **periodici** o **aperiodici**. I processi periodici vengono attivati ciclicamente a periodo costante che dipende dalla grandezza fisica che il processo deve controllare.
- I processi non periodici vengono avviati in situazioni imprevedibili.
- Consideriamo un algoritmo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

  In tal caso deve essere:

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

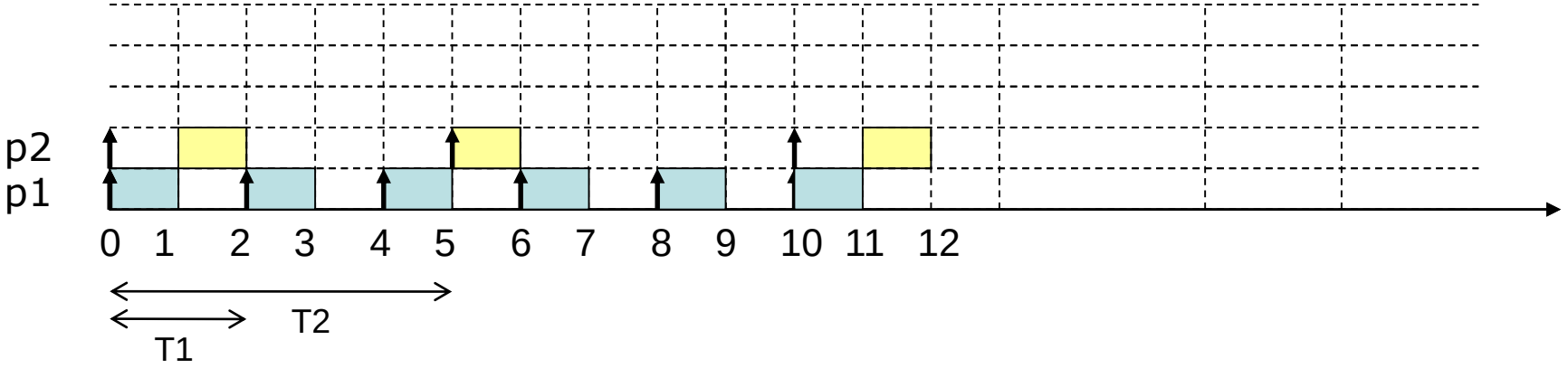
$$T >= D$$



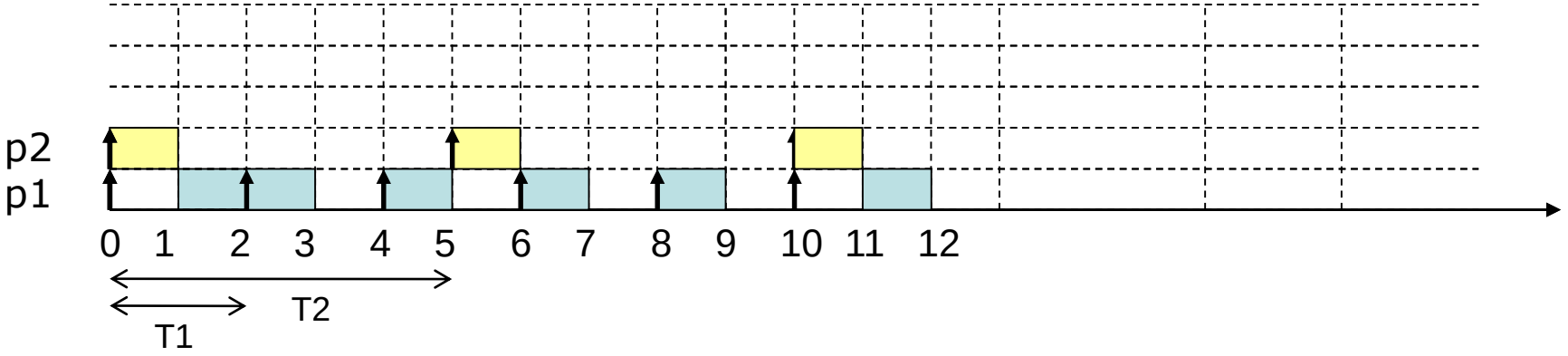
Periodo T

- 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

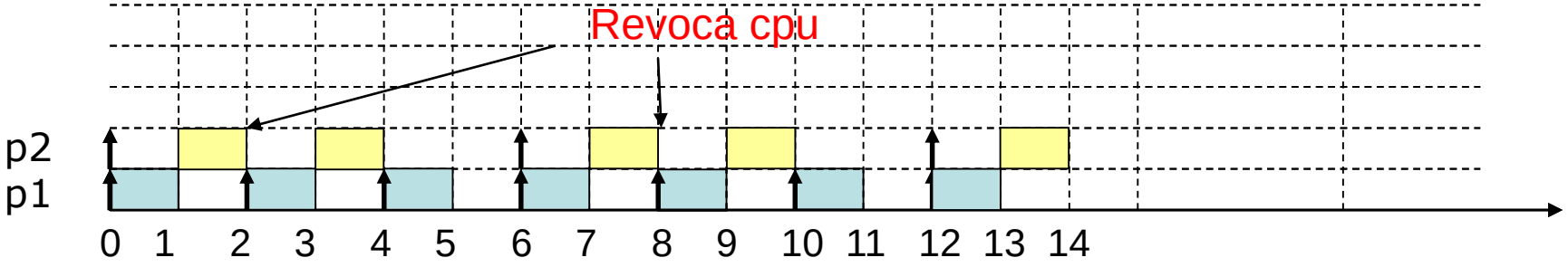


Priorità P2 > priorità P1

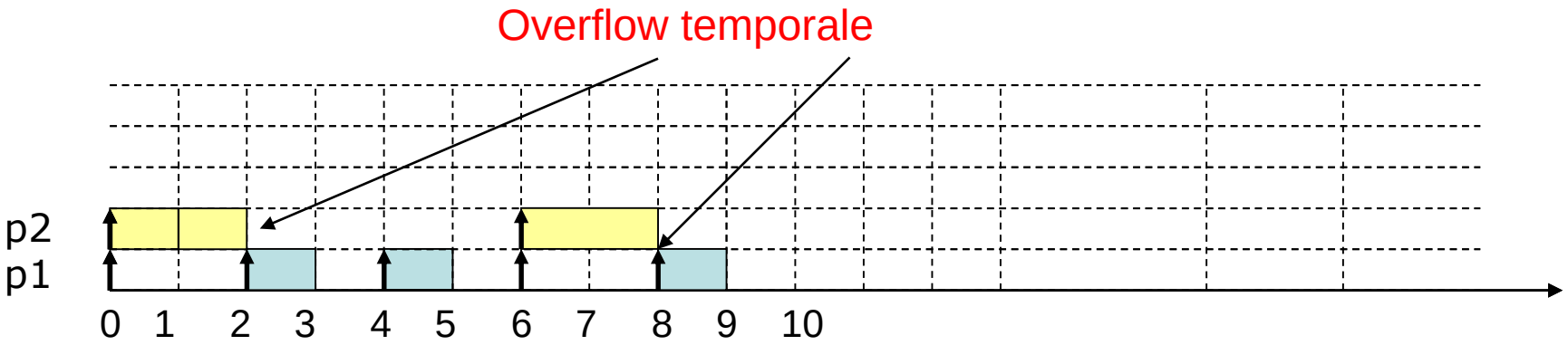
Supponiamo ora che sia E2 = 2 T2=6, mentre sia ancora

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:

  $U = \sum (Ei/Ti) <= 1$

  Dove **U** è detto **coefficiente di utilizzazione** della CPU.
- E' stato dimostrato, utilizzando ***Rate Monotonic***, che affinché 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

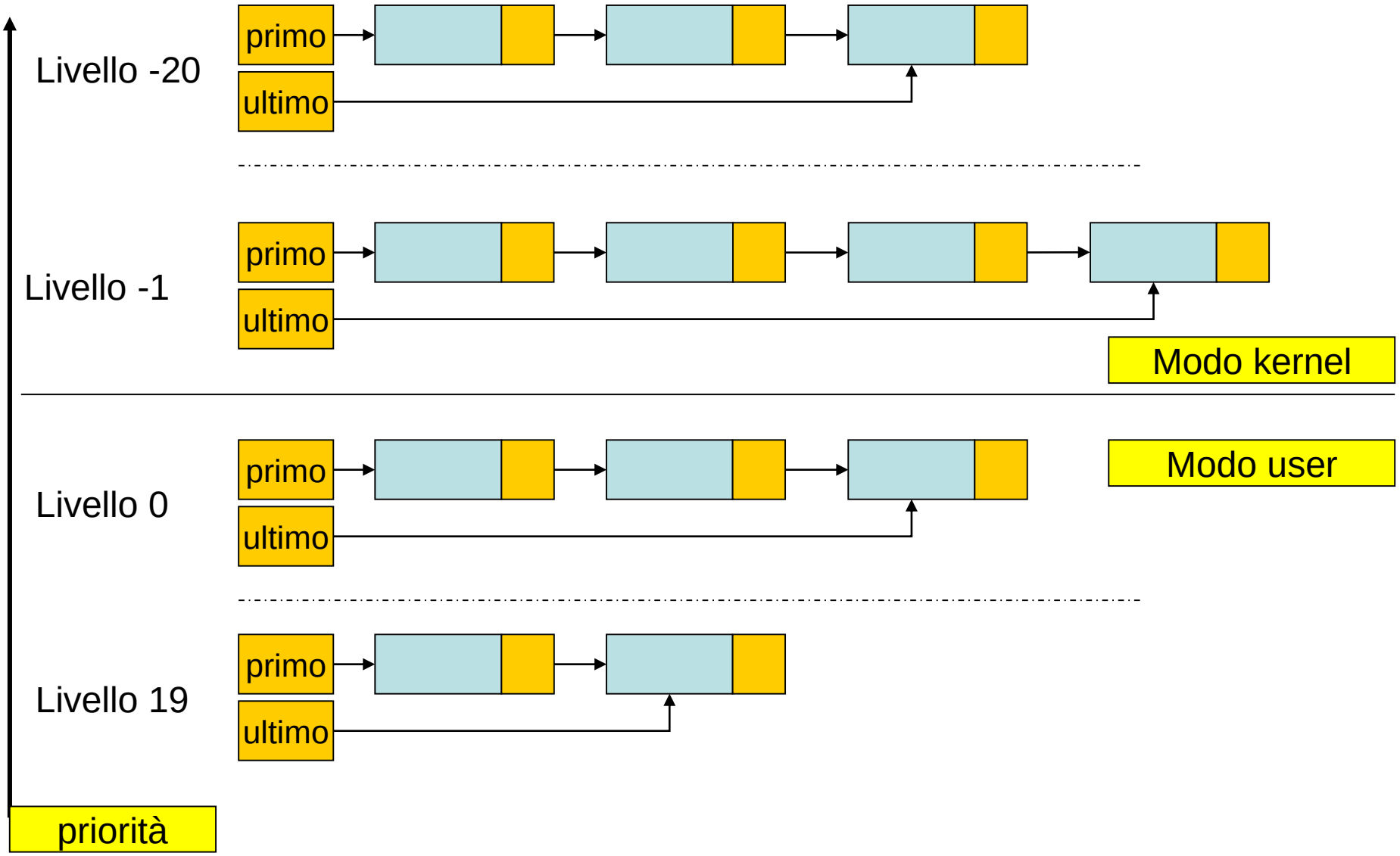
  per N=100 U=0.695

  per N->infinito U -> ln 2 = 0,693147180559453094172

Scheduling in UNIX

- Poiché UNIX è un sistema multiutente e multitasking, l'algoritmo di scheduling della CPU è stato progettato per fornire buoni tempi di risposta ai **processi interattivi**.
- I thread sono generalmente a livello di kernel, pertanto lo scheduler si basa sui thread e non sui processi.
- E' un algoritmo a **due livelli** di scheduler.
- Lo **scheduler a breve termine** sceglie dalla coda dei processi pronti il prossimo processo/thread da eseguire.
- Lo **scheduler a medio termine (swapper)** sposta pagine di processi tra la memoria e il disco (area swap o file di paging) in modo che tutti i processi abbiano la possibilità di essere eseguiti.
- Ogni versione di UNIX ha uno scheduler a breve termine leggermente diverso, ma tutti seguono uno schema di funzionamento basato su code di priorità.

- In Unix le priorità dei processi eseguiti in **modalità utente** sono espresse con valori interi positivi mentre le priorità dei processi eseguiti in **modalità kernel** (che eseguono le chiamate di sistema) sono espresse con valori interi negativi.
- I valori **negativi** rappresentano **priorità maggiori**, rispetto ai valori positivi che hanno priorità minore.
- Lo scheduler a breve termine sceglie un processo dalla coda con priorità più alta. Le code sono gestite in modalità **RR**. Il quanto di tempo assegnato al processo per l'esecuzione dura generalmente **20-100 ms**.
- Un processo è posto in fondo alla coda, quando termina il suo quanto di tempo.



scheduling in unix

- I valori delle **priorità sono dinamici** e sono ricalcolati **ogni secondo** in base ad una relazione che dipende dai seguenti parametri:

  - **valore iniziale (base)**
  - **uso_cpu**
  - **nice**

  priorità = f(base, uso_cpu, nice)

- Ogni processo è poi inserito in una coda, come mostrato nella figura precedente, in base alla nuova priorità.
- **Uso_cpu** rappresenta, l'uso medio della cpu da parte del processo durante gli ultimi secondi precedenti. Questo parametro è un campo del descrittore del processo (PCB).

- L’incremento del valore del parametro **uso_cpu** provoca lo spostamento del processo in una coda a priorità più bassa.
- Il valore di **uso_cpu** varia nel tempo in base a varie strategie usate nelle varie versioni di UNIX.
- Ogni processo ha, inoltre, un valore del parametro **nice** associato. Il valore di base è 0 e l’intervallo di valori possibili è compreso tra –20 e +19. Ad esempio, con il comando **nice** (che utilizza l’omonima chiamata di sistema **nice**), un utente può assegnare ad un proprio processo un valore *nice* compreso tra 0 e 19. Soltanto il **superuser** (root) può assegnare i valori di *nice* compresi tra –1 e –20 ad un processo.
- Esempio:

Carattere per esecuzione in background

nice –n 10 mio_calcolo &

esegue il programma **mio_calcolo** in background assegnando al processo un valore nice pari a 10.

- Per quanto riguarda lo scheduling per le estensioni real-time, lo standard P1003.4 di UNIX, cui aderisce anche Linux, prevedono le seguenti classi di thread:
  - **Real-time FIFO;**
  - **Real time round-roubin;**
  - **Timesharing**
- I thread real-time FIFO hanno la priorità massima. A questi thread può essere revocata la cpu solo da thread della stessa classe con più alta priorità.
- I thread real-time RR sono simili ai real-time FIFO, ma viene loro revocata la CPU allo scadere del proprio quanto di tempo, il cui valore dipende dalla priorità.
- Le due classi di thread sono **soft real-time**.
- I thread real-time hanno livelli di priorità da 0 a 99, dove 0 è il livello di priorità più alto.

- I thread standard, non real-time, hanno livelli di priorità compresi tra 100 e 139. In totale, quindi si hanno 140 livelli di priorità.
- Il quanto di tempo è misurato in numero di scatti di clock. Lo scatto di clock è detto **jiffy** e dura 1 ms.

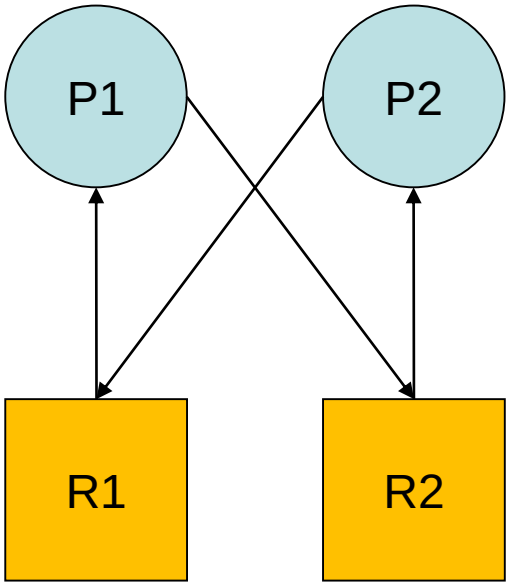
Blocco critico (stallo)

- In un sistema multiprogrammato, durante l'esecuzione, un processo può richiedere risorse condivise per svolgere la sua attività.
- In un determinato istante, la risorsa richiesta potrebbe essere non disponibile perché già allocata in precedenza a un altro processo. In questo caso il processo richiedente passa nello stato di bloccato.
- La situazione di **stallo** (**deadlock**) si può verificare tra due o più processi quando ciascuno dei processi possiede almeno una risorsa e ne richiede altre. Il processo richiedente non può ottenere la risorsa poiché la risorsa richiesta è stata già assegnata ad un altro processo che non la rilascia in quanto è in attesa di un'altra risorsa che è già allocata ad un altro processo ancora.

- Un gruppo di processi è in uno stato di deadlock quando ogni processo è in attesa di un evento che può essere causato solo da un altro processo appartenente al gruppo.
- In un normale funzionamento, un processo può utilizzare una risorsa seguendo la seguente sequenza:

1. **Richiesta**. Il processo richiede la risorsa. Se la richiesta non può essere concessa immediatamente (ad esempio, se la risorsa è utilizzata da un altro processo), allora il processo richiedente deve attendere finché può acquisire la risorsa.
2. **Uso**. Il processo esegue operazioni sulla risorsa
3. **Rilascio**. Il processo rilascia la risorsa.

- La richiesta e il rilascio delle risorse possono essere chiamate di sistema. Esempi di chiamate di sistema per la richiesta e il rilascio sono:
  - open () e close ( ) per file e dispositivi;
  - malloc( ) e free ( ) per la memoria.
- Analogamente, come abbiamo visto , le operazioni di richiesta e di rilascio con i semafori possono essere realizzate mediante le operazioni wait () e signal ().



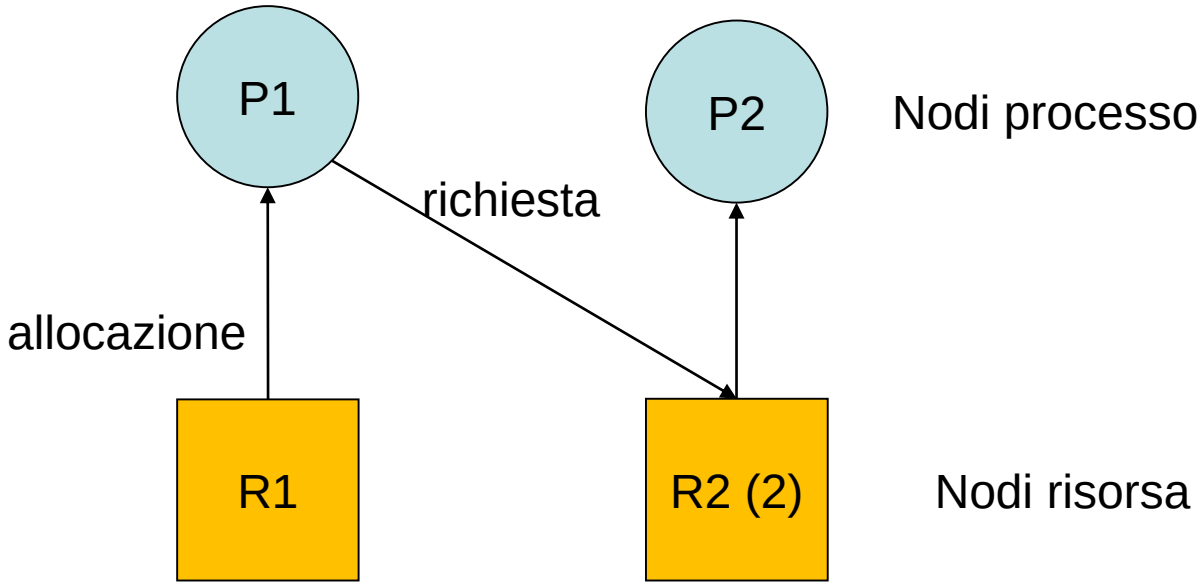
Rappresentazioni dello stato di allocazione delle risorse

- Per stabilire se un certo numero di processi è in stallo è necessario analizzare le informazioni relative alle risorse condivise allocate ai processi e quelle relative alle richieste di risorse in attesa.
- Per rappresentare lo stato di allocazione di un sistema si utilizzano due tipi di rappresentazione:
  - **Modelli basati su grafo**
  - **Modelli basati su matrici**

Modelli basati su grafo

- I deadlock possono essere descritti con un ***grafo orientato***, detto ***grafo di allocazione delle risorse***. Questo grafo è costituito da un insieme di nodi *N* e un insieme di archi *A*.

- L'insieme di nodi *N* è suddiviso in due tipi diversi di nodi: $P = \{P_1, P_2, ..., P_n\}$, l'insieme costituito da tutti i processi attivi nel sistema, e $R = \{R_1, R_2, ..., R_m\}$, l'insieme costituito da tutti i tipi di **risorse condivise** nel sistema.



- Un arco orientato dal processo P_i alla risorsa di tipo R_j, indicato con $P_i \rightarrow R_j$, significa che il processo P_i ha richiesto un'unità di una risorsa della classe R_j ed è ora in attesa per quella risorsa.

- Un arco orientato dal tipo di risorsa R_j verso il processo P_i, espresso con $R_j \rightarrow P_i$, indica che un'unità di risorsa tipo R_j è stata allocata al processo P_i.
- Un arco $P_i \rightarrow R_j$ è detto ***arco di richiesta***; un arco orientato $R_j \rightarrow P_i$ è chiamato ***arco di assegnazione***.
- Graficamente, un processo P_i si rappresenta con un cerchio e ogni tipo di risorsa R_j con un rettangolo o un quadrato.
- Poiché un tipo di risorsa R_j può avere più di un'unità, si indica tale pluralità con un numero intero all'interno del rettangolo.
- Il semplice grafo di allocazione delle risorse mostrato nella figura precedente mostra l'allocazione descritta dai seguenti insiemi: $P = \{P_1, P_2\}$, $R = \{R_1, R_2\}$ e $A = \{ P_1 \rightarrow R_2, R_1 \rightarrow P_1, R_2 \rightarrow P_2\}$.

Modelli basati su matrici

- Con il modello basato su matrici, lo stato di allocazione del sistema è rappresentato dalle seguenti matrici:

	R1	R2	Rm
P1	0	1	2
P2	1	0	3
Pn	0	1	0

Risorse allocate

	R1	R2	Rm
P1	1	0	1
P2	0	1	2
Pn	0	0	1

Risorse richieste

R1	R2	Rm
10	12	9

Risorse totali

R1	R2	Rm
0	3	2

Risorse libere