

Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica
Sistemi operativi e reti

A.A. 2019-2020

Pietro Frasca

Lezione 13

Martedì 19-11-2019

Principali algoritmi di scheduling

- Alcuni algoritmi sono usati sia nei sistemi batch che nei sistemi interattivi.

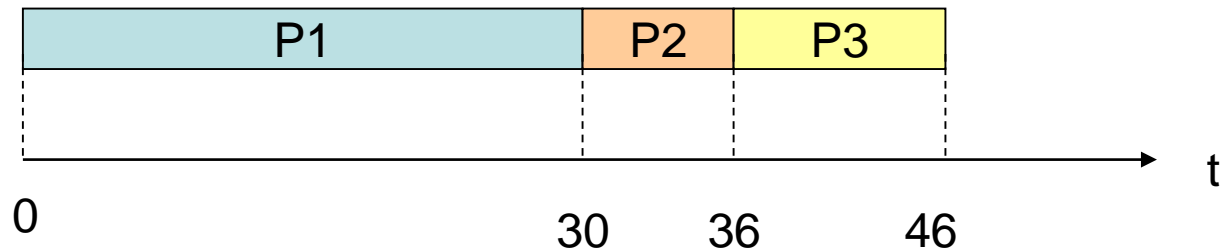
FCFS

- FCFS (First Come First Served) è l'algoritmo più semplice. La CPU è assegnata ai processi seguendo l'ordine con cui essa è stata richiesta, ovvero la cpu è assegnata al processo che è in attesa da più tempo nella coda di pronto.
- Quando un processo acquisisce la CPU, resta in esecuzione fino a quando si blocca volontariamente o termina. Quando un processo in esecuzione si blocca si seleziona il primo processo presente nella coda di pronto. Quando un processo da bloccato ritorna pronto, è accodato nella coda di pronto.
- E' un algoritmo senza diritto di prelazione.

- E' inefficiente nel caso in cui ci sono molti processi che si sospendono frequentemente o nel caso di sistemi interattivi. Il tempo di attesa risulterebbe troppo lungo

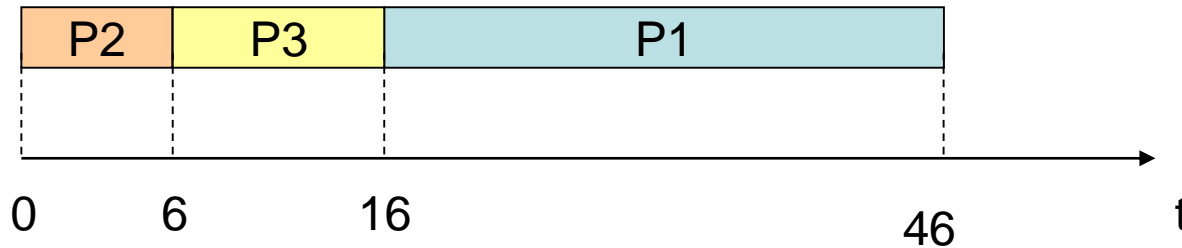
Esempio

Calcoliamo il tempo medio di attesa di tre processi P1, P2 e P3 avviati allo stesso istante e aventi rispettivamente i tempi (in millisecondi) di completamento pari a: $T1=30$, $T2=6$ e $T3=10$. Il diagramma temporale (di Gantt) è il seguente.



$$\text{Tempo}_{\text{attesa medio}} = (0+30+36)/3 = 22 \text{ ms}$$

- Se cambiassimo l'ordine di scheduling nel seguente: {P2, P3, P1} il tempo medio di attesa passerebbe da 22 ms a 7,33 ms.

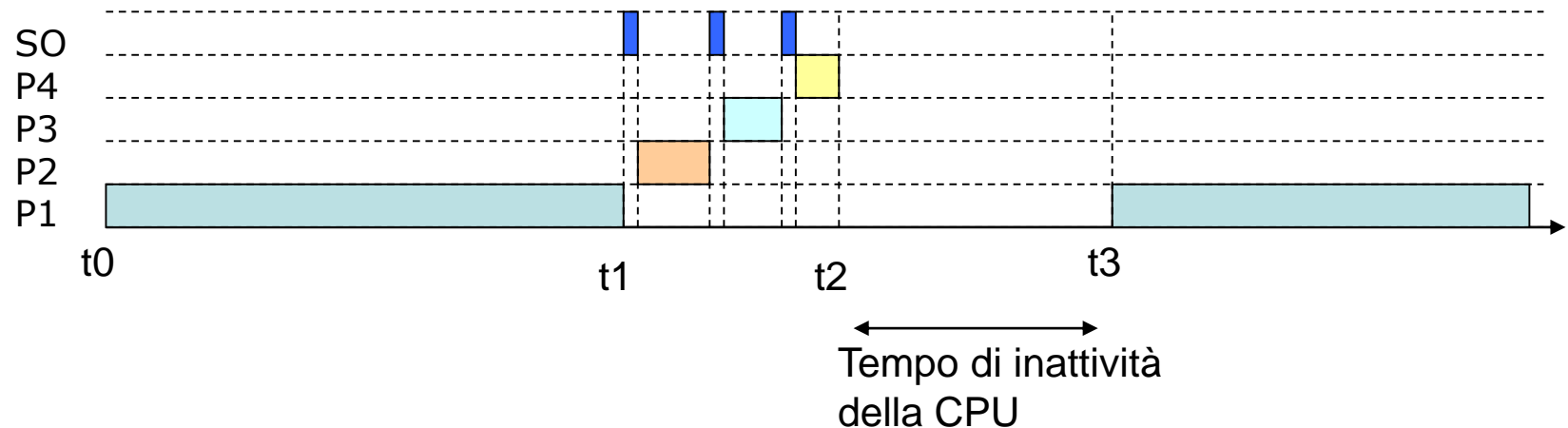


$$\text{Tempo}_{\text{attesa medio}} = (0+6+16)/3 = 7,33 \text{ ms}$$

- Ma il FCFS non consente di cambiare l'ordine dei processi. Quindi nel caso in cui ci siano processi con brevi CPU-burst (processi I/O-bound) in attesa dietro a processi con lunghi CPU-burst (processi CPU-bound), il tempo di attesa sarà alto.
- Inoltre, in considerazione delle prestazioni del FCFS in una situazione dinamica, supponiamo di avere un processo CPU-bound e molti processi I/O-bound e si verifichi il seguente scenario. Il processo CPU-bound ottiene la CPU.

- Durante questo tempo, tutti gli altri processi terminano i loro I/O e si spostano nella coda di pronto, in attesa di ottenere la CPU. Mentre i processi sono in attesa nella coda di pronto, i dispositivi di I/O sono inattivi. Successivamente, il processo CPU-bound termina il suo burst ed è trasferito nella coda di un dispositivo di I/O. Tutti i processi I/O-bound, che hanno brevi burst di CPU, eseguono le loro sequenze d'istruzioni in modo rapido e tornano nelle code di I/O. A questo punto, la CPU rimane inattiva. Il processo CPU-bound sarà poi posto di nuovo nella coda pronto e tornerà in esecuzione. Anche in questo caso, tutti i processi di I/O saranno rimessi nella coda di pronto fino al termine del burst del processo CPU-bound. Vi è un **effetto convoglio**, tutti gli altri processi aspettano che un processo con un lungo CPU-burst lasci la CPU. Questo effetto porta a un utilizzo, sia della CPU che dei dispositivi, inferiore a quanto possa essere se i processi più *brevi* fossero eseguiti prima dei processi più *lunghi*.

- La figura mostra una situazione in cui si verifica l'effetto convoglio. In questo esempio, P1 è un processo CPU-bound; P2, P3 e P4 sono processi I/O-bound. P1 effettua operazioni di I/O nell'intervallo [t1, t3]. Si ha che nell'intervallo [t2, t3] la CPU è inattiva.
- Nella figura sono mostrati anche gli intervalli di tempo in cui lo scheduler va in esecuzione.



Shortest job first (SJF)

- Gli algoritmi SJF, possono essere sia non-preemptive **Shortest Next Process First (SNPF)** sia preemptive **Shortest Remaining Time First (SRTF)**.

Shortest Next Process First (SNPF)

- L'algoritmo SNPF prevede che sia eseguito sempre il processo con il tempo di esecuzione più breve tra quelli pronti.
- Supponiamo ad esempio che si trovino nello stato di pronto i seguenti processi, con la rispettiva durata di esecuzione in millisecondi:

[p1, 10] → [p2, 2] → [p3, 6] → [p4, 4]

- Con SNPF i processi sono eseguiti nel seguente ordine:
 $p2 \rightarrow p4 \rightarrow p3 \rightarrow p1$
- Trascurando il tempo necessario per il cambio di contesto, il processo p2 non attende nulla, perché va subito in esecuzione, p4 attende 2 millisecondi, perché va in esecuzione subito dopo p2, quindi p3 attende 6 millisecondi e p1 ne attende 12. Il tempo di attesa medio è pari a:

$$\text{Tempo}_{\text{attesa medio}} = (0+2+6+12)/4 = 5 \text{ millisecondi}$$

Shortest Remaining Time First (SRTF)

- L'algoritmo **SRTF** è la versione preemptive del precedente.
- Con SRTF, se un nuovo processo, entrante nella coda di pronto, ha una durata minore del tempo restante al processo in esecuzione per portare a terminare il proprio CPU-burst, allora lo scheduler provvede ad effettuare un cambio di contesto e assegna l'uso della CPU al nuovo processo.

- Si può dimostrare che teoricamente l'algoritmo SJF è ottimale, in quanto, selezionando dalla coda di pronto il processo più breve da eseguire, consente di ottenere sempre il valore più basso del tempo di attesa medio.
- Per poterlo applicare, nei sistemi batch, i programmi si avviano fornendo per ciascuno di essi un'informazione relativa alla durata dell'esecuzione. Tale valore, è ricavato dalle precedenti esecuzioni del job.
- Con SRTF lo scheduler, dato che non è possibile stabilire la durata del prossimo cpu-burst, implementa algoritmi di predizione che stimano il prossimo tempo di cpu-burst a partire da quelli precedentemente eseguiti.
- Una stima spesso usata si basa sulla media esponenziale:

$$s_{n+1} = \alpha \cdot T_n + (1 - \alpha) s_n$$

dove T_n è la durata dell'n-esimo CPU-burst, s_{n+1} la durata

prevista per la successiva sequenza e α $[0..1]$ è il peso che deve essere assegnato al passato del processo.

- Espandendo la relazione si può notare come i valori dei singoli intervalli abbiano un peso tanto minore quanto più sono vecchi

:

$$s_{n+1} = \alpha \cdot T_n + (1 - \alpha) \alpha \cdot T_{n-1} + \dots (1-\alpha)^i \alpha \cdot T_{n-i} + \dots (1-\alpha)^{n+1} s_0$$

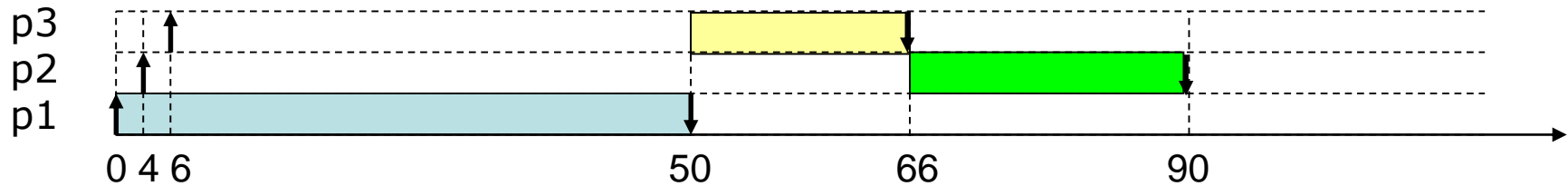
In genere per $\alpha = 0,5$ e $s_0 = 10$ si ha una buona approssimazione.

Esempio

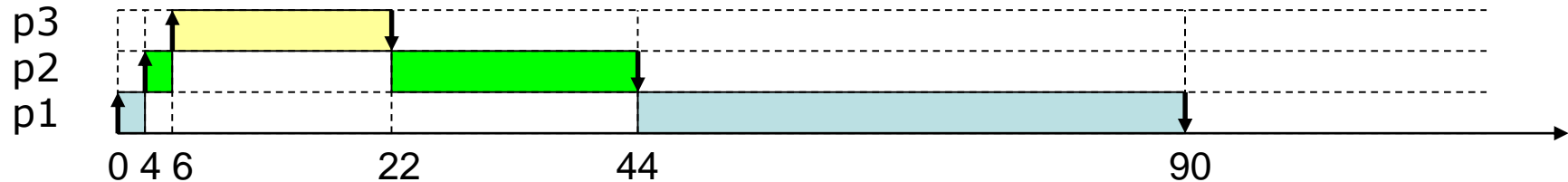
vediamo i diagrammi temporali relativi ai seguenti processi

- P1 [0,50]
- P2 [4,24]
- P3 [6,16]

per gli algoritmi SJF e SRTF



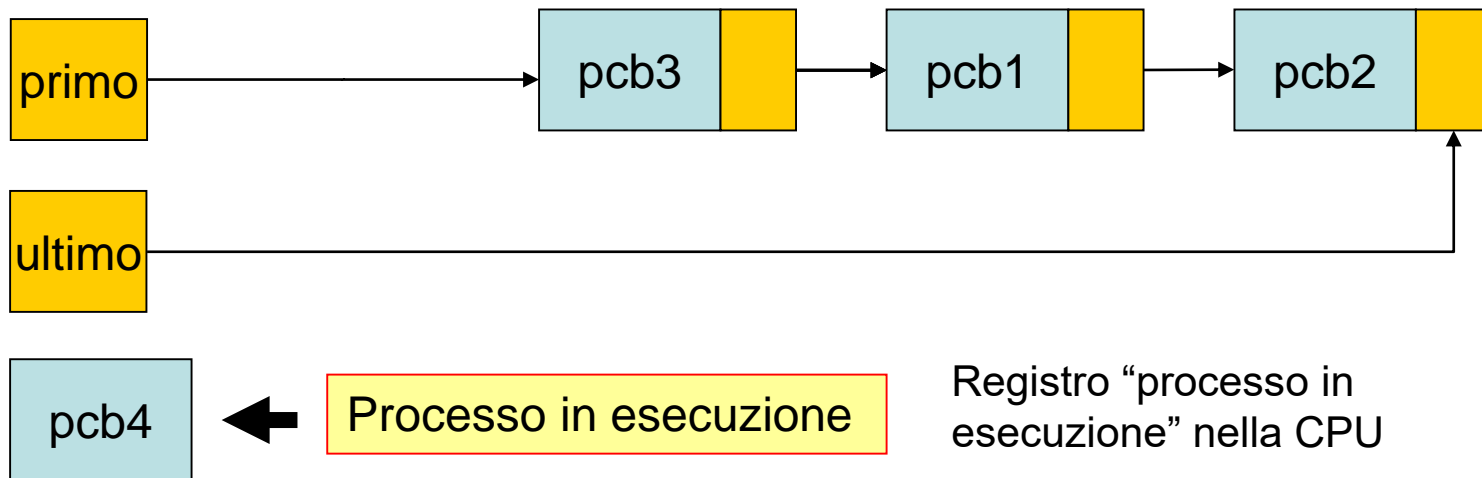
SJF



SRTF

Round Robin

- E' stato realizzato **per i sistemi time-sharing**.
- Consente il prerilascio.
- La coda dei processi pronti è di tipo circolare. La CPU viene assegnata ad ogni processo per un **quanto di tempo**, tipicamente da 10 a 100 millisecondi.
- La coda è gestita in modalità **FIFO**: il processo a cui viene revocata la CPU è inserito in fondo alla coda e il successivo che avrà il controllo della CPU è il primo della coda.

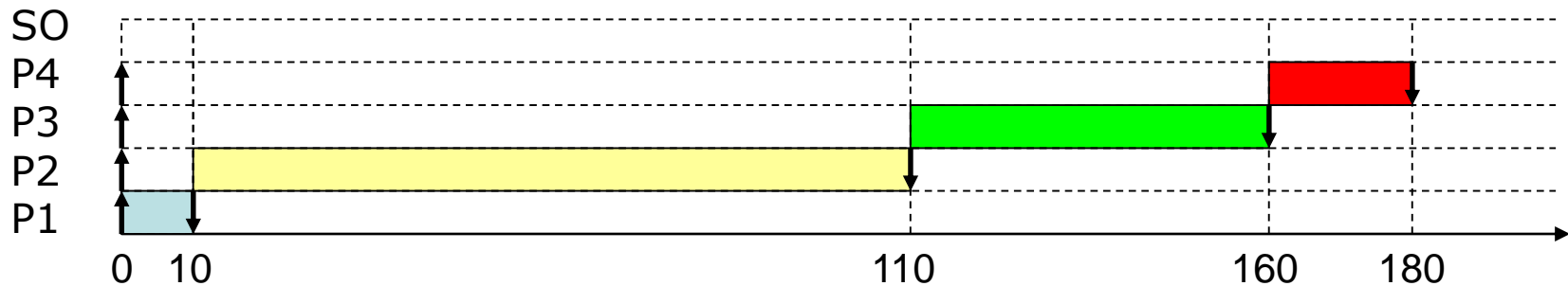


- E' usato nei i sistemi time-sharing in quanto assicura tempi di attesa medi brevi che dipendono principalmente dal valore del quanto di tempo e dal numero medio di processi pronti.
- Il tempo medio di risposta tende a migliorare diminuendo il valore del quanto di tempo, ma se questo assume valori troppo piccoli diventa significativo l'overhead prodotto dalle operazione del cambio di contesto.
- E' necessario che sia:

Tempo_cambio_di_contesto << durata_quanto_di_tempo

Esempio

- Vediamo come **RR** privilegia i processi interattivi rispetto ai CPU-bound. Supponiamo che all'istante T_0 siano presenti i seguenti quattro processi nella coda di pronto. Calcoliamo il tempo medio di attesa e di risposta per quattro processi P1, P2, P3 e P4 aventi rispettivamente i tempi di arrivo e durata di CPU-burst (in millisecondi) pari a: P1 [0,10], P2 [0,100], P3 [0,50], P4 [0,20]. Con **FCFS** si ha:



A1 = 0
A2 = 10
A3 = 110
A4 = 160

R1 = 10
R2 = 110
R3 = 160
R4 = 180

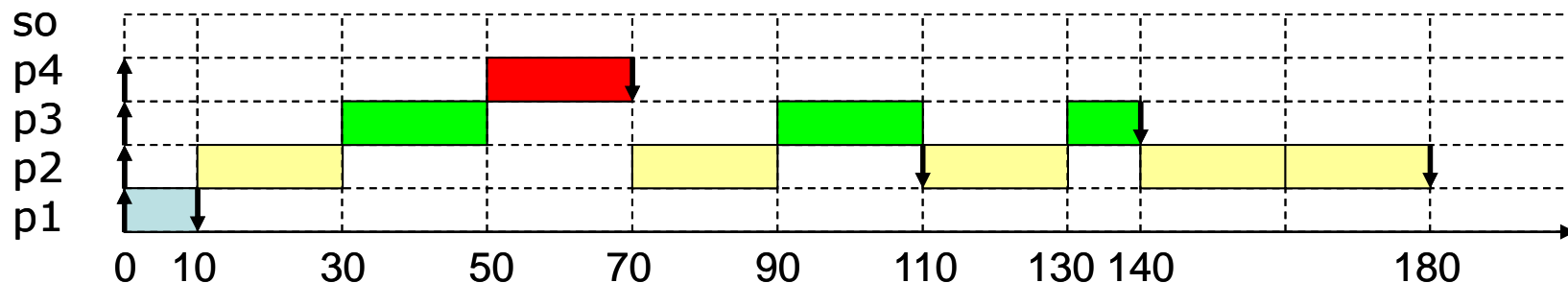
tempo medio di attesa

$$A_m = (0 + 10 + 110 + 160)/4 = 70$$

tempo medio di risposta

$$R_m = (10 + 110 + 160 + 180)/4 = 115$$

- Con **RR** si ha:



$$A1 = 0$$

$$A2 = 10 + 40 + 20 + 10 = 80$$

$$A3 = 30 + 40 + 20 = 90$$

$$A4 = 50$$

$$R1 = 10$$

$$R2 = 180$$

$$R3 = 140$$

$$R4 = 70$$

tempo medio di attesa

$$A_m = (0 + 80 + 90 + 50)/4 = 55$$

tempo medio di risposta

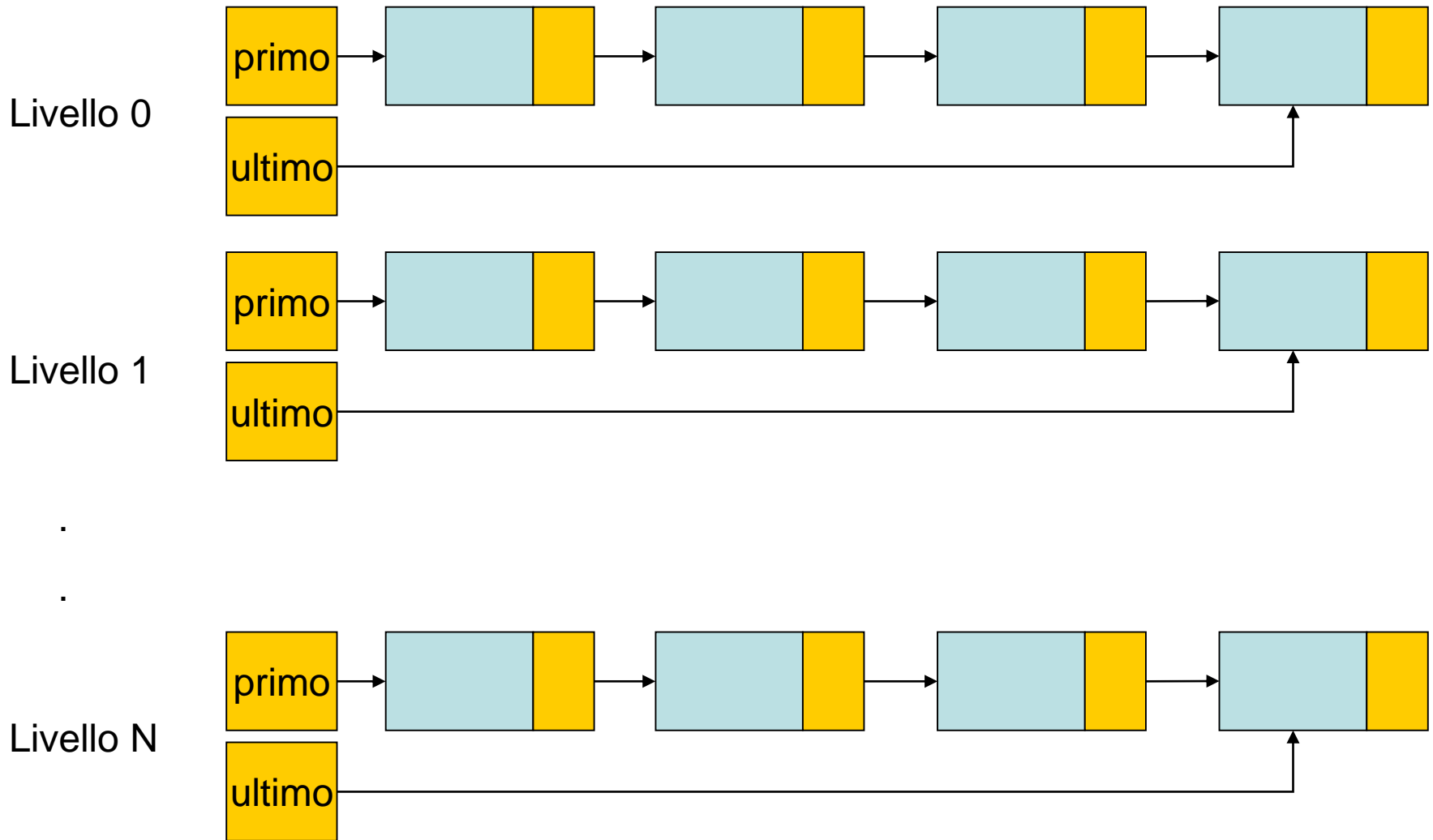
$$R_m = (10 + 180 + 140 + 70)/4 = 100$$

Da questo esempio si vede che RR consente di avere tempi di risposta tanto minori quanto minore è il periodo di tempo di esecuzione richiesto a prescindere dall'ordine in cui i processi sono entrati nella coda di pronto. Nell'esempio si è posto un quanto di tempo $\Delta = 20$ ms.

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

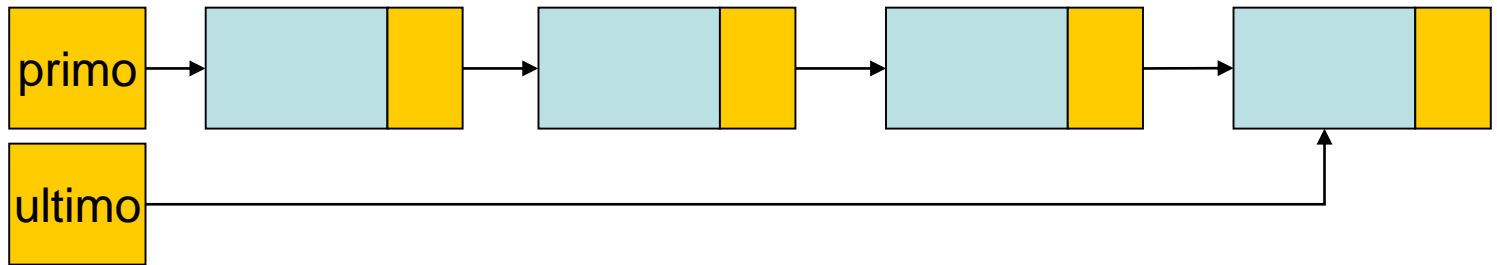


Algoritmi di scheduling a code multiple

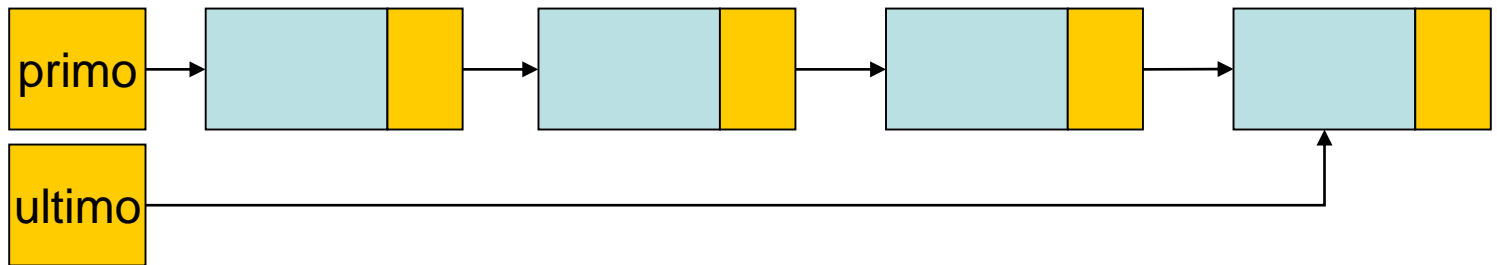
- Nei SO complessi sono in esecuzione processi con caratteristiche diverse.
- Sono presenti processi CPU-bound e IO-bound, oppure processi interattivi e processi tipo “batch” detti rispettivamente processi **foreground** e **background**.
- Pertanto i processi pronti sono inseriti in varie code, ciascuna 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

