Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2017-18

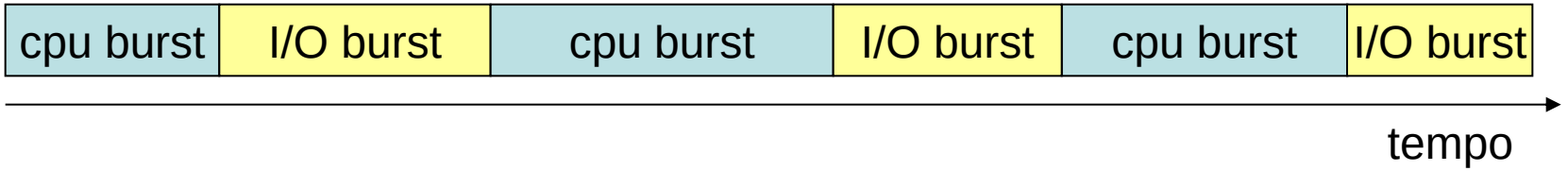
Pietro Frasca

Lezione 14

Martedì 21-11-2017

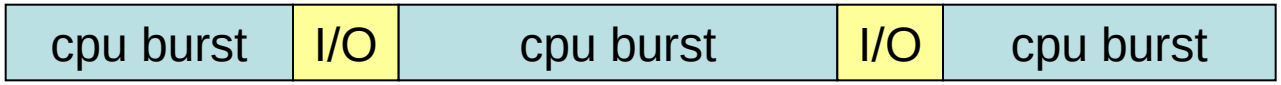
Comportamento dei processi

- Durante la sua attività, un processo generalmente alterna le seguenti fasi**:**
  - **CPU burst**: fase in cui viene impiegata soltanto la CPU senza I/O;
  - **I/O burst:** fase in cui il processo effettua input/output da/verso una risorsa(dispositivo) del sistema.
- Il termine burst (raffica) indica una sequenza di istruzioni.

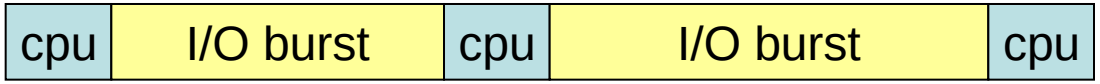


- Quando un processo esegue I/O burst, non utilizza la CPU. In un sistema multiprogrammato, lo scheduler assegna la CPU a un nuovo processo.

- I processi, in base al tipo di istruzioni che eseguono si classificano in:
  - Processi ***CPU-bound (compute-bound)***: se eseguono prevalentemente istruzioni di computazione; CPU burst di *lunga* durata, intervallati da pochi I/O burst di *breve* durata.



  - Processi ***I/O-bound***: se eseguono prevalentemente istruzioni di I/O; CPU burst di *breve* durata, intervallati da I/O burst di *lunga* durata.



- Gli algoritmi di scheduling si possono classificare in due categorie:
  - **con prelazione** (***pre-emptive***): al processo in esecuzione può essere revocata la CPU. Il SO può, in base a determinati criteri, sottrarre ad esso la CPU per assegnarla ad un nuovo processo.
  - **senza prelazione** (***non pre-emptive***): la CPU rimane allocata al processo in esecuzione fino a quando esso si sospende volontariamente (ad esempio, per I/O), o termina.
- Ad esempio, i sistemi a divisione di tempo (*time sharing*) hanno uno scheduling ***pre-emptive.***
- Gli algoritmi **non pre-emptive** sono più semplici, e richiedono minor overhead di sistema in quanto effettuano meno cambi di contesto. D'altra parte sono meno flessibili in quanto offrono una minore gamma di strategie di scheduling. Sono adatti per sistemi batch ma non per sistemi time-sharing.
- Le prime versioni dei sistemi windows (fino alla versione 3.1) utilizzavano algoritmi non pre-emptive. Dalla versione windows 95 fu usato uno scheduler con revoca.
- Linux e Unix adottano algoritmi pre-empitive.

Parametri di scheduling

- Per analizzare e confrontare i diversi algoritmi di scheduling, si considerano i seguenti parametri, i primi tre relativi ai processi e gli altri relativi al sistema:
  - **Tempo di attesa,** è la quantità di tempo che un processo trascorre nella coda di pronto, in attesa della CPU.
  - **Tempo di completamento (Turnaround time),** è l'intervallo di tempo che passa tra l'avvio del processo e il suo completamento.
  - **Tempo di risposta,** è l'intervallo di tempo tra avvio del processo e l'inizio della prima risposta.
  - **Utilizzo della CPU,** esprime la percentuale media di utilizzo della CPU nell'unità di tempo.
  - **Produttività (Throughput rate)** (del sistema), esprime il numero di processi completati nell'unità di tempo.

- Generalmente i parametri che devono essere massimizzati sono:
  - **Utilizzo della CPU (**al massimo: 100%)
  - **Produttività (Throughput)**

  invece, devono essere **minimizzati**:
  - **Tempo medio di completamento (Turnaround)** (sistemi *batch*)
  - **Tempo medio di attesa**
  - **Tempo medio di risposta** (sistemi *interattivi*)

- Non è possibile ottenere tutti gli obiettivi contemporaneamente.

- A seconda del tipo di SO, gli algoritmi di scheduling possono avere **diversi obiettivi**; tipicamente:
  - nei sistemi **batch**:
    - **massimizzare il throughput** e **minimizzare il tempo medio di completamento**
  - nei sistemi **interattivi**:
    - **minimizzare il tempo medio di risposta** dei processi
    - **minimizzare il tempo medio di attesa** dei processi
- Nei sistemi real-time, l'algoritmo con revoca non sempre è necessario dato che i programmi real-time sono progettati per essere eseguiti per brevi periodi di tempo e poi si bloccano.

Principali algoritmi di scheduling

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

FCFS

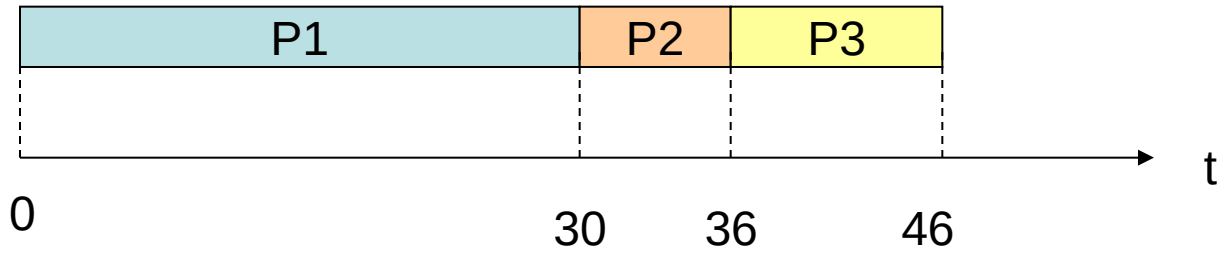
- FCFS (First Came First Served) è l'algoritmo più semplice. La CPU è assegnata ai processi seguendo l'ordine con cui essa è stata richiesta, ovvero la cpu è assegnata al processo che è in attesa da più tempo nella coda di pronto.
- Quando un processo acquisisce la CPU, resta in esecuzione fino a quando 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, esso è accodato nella coda di pronto.

- 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. E’ un algoritmo senza diritto di prelazione.

Esempio

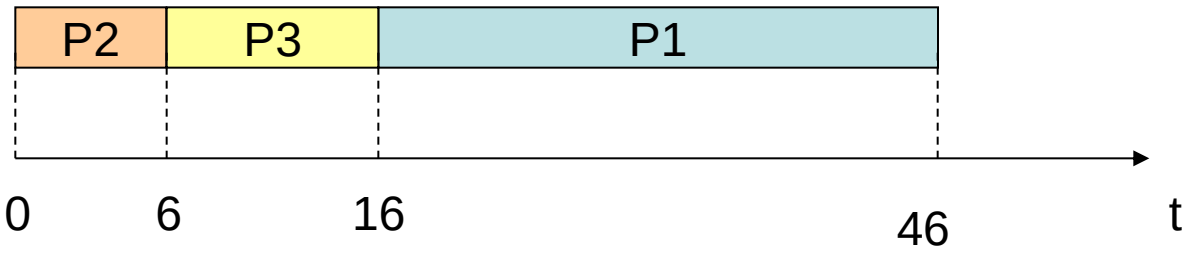
Calcoliamo il tempo medio di attesa di tre processi P1, P2 e P3 avviati allo stesso istante e aventi rispettivamente i tempi (in millisecondi) di completamento pari a: T1=30, T2=6 e T3=10.

Il diagramma temporale (di Gantt) è il seguente.



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

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

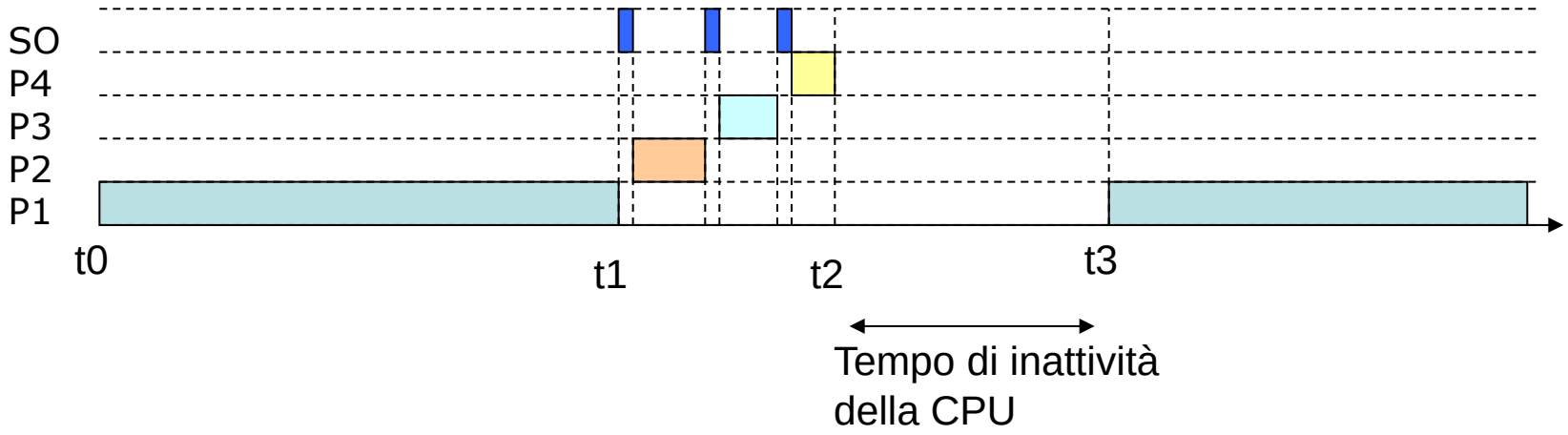


Tempo $_{\text{attesa medio}}$ = (0+6+16)/3 = 7,33

- 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 traferito 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 → p4 → p3 → 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:

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

:

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

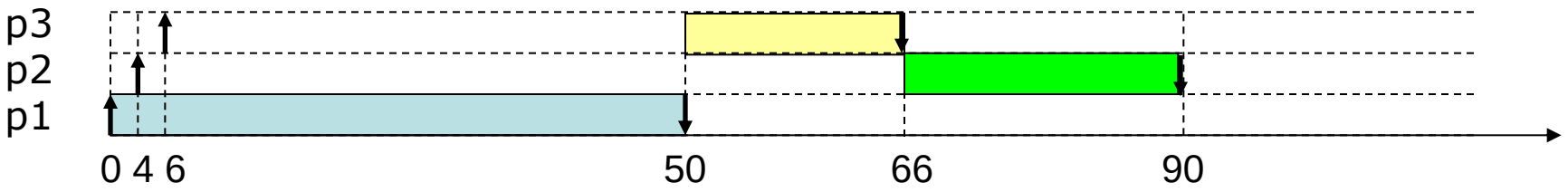
In genere per $\alpha = 0{,}5$ e $s_0 = 10$ si ha una buona approssimazione.

Esempio

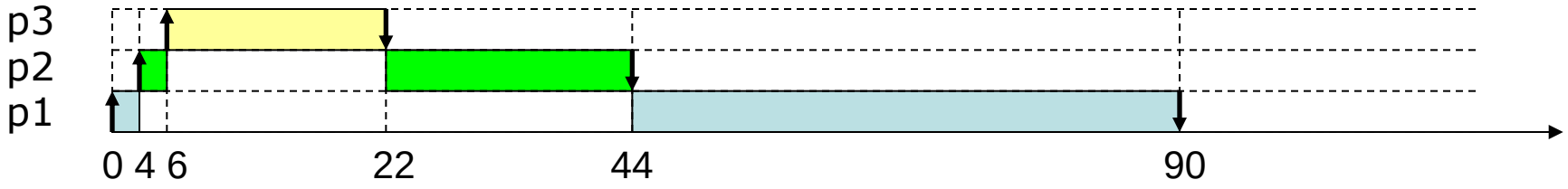
vediamo i diagrammi temporali relativi ai seguenti processi

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

per gli algoritmi SJF e SRTF



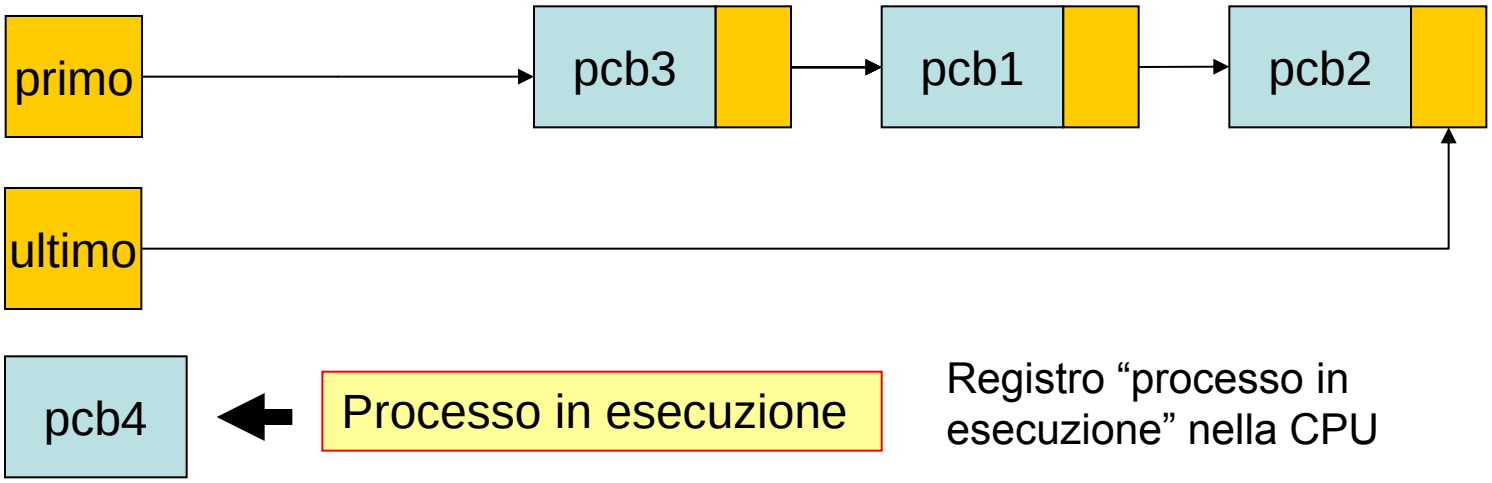
SJF



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.

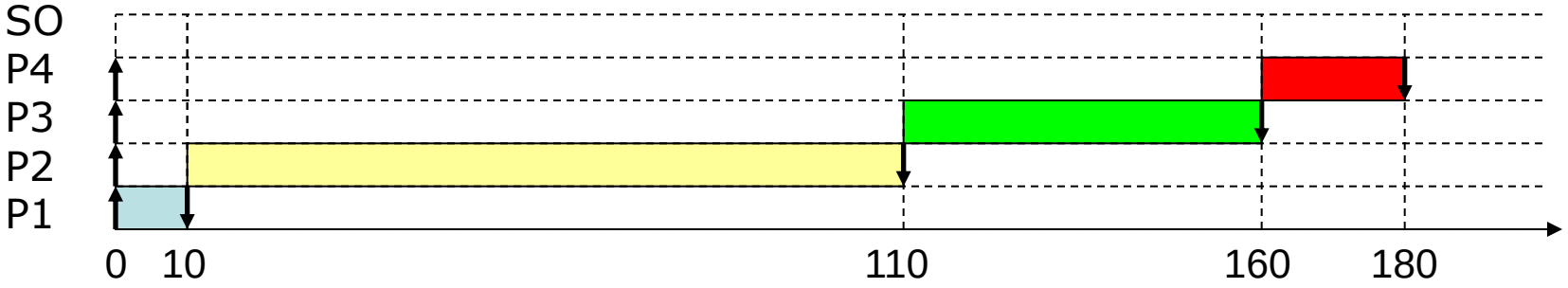


- E’ usato nei i sistemi time-sharing in quanto assicura tempi di attesa medi brevi che dipendono principalmente dal valore del quanto di tempo e dal numero medio di processi pronti.
- Il tempo medio di risposta tende a migliorare diminuendo il valore del quanto di tempo, ma se questo assume valori troppo 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** privilegi 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

tempo medio di attesa

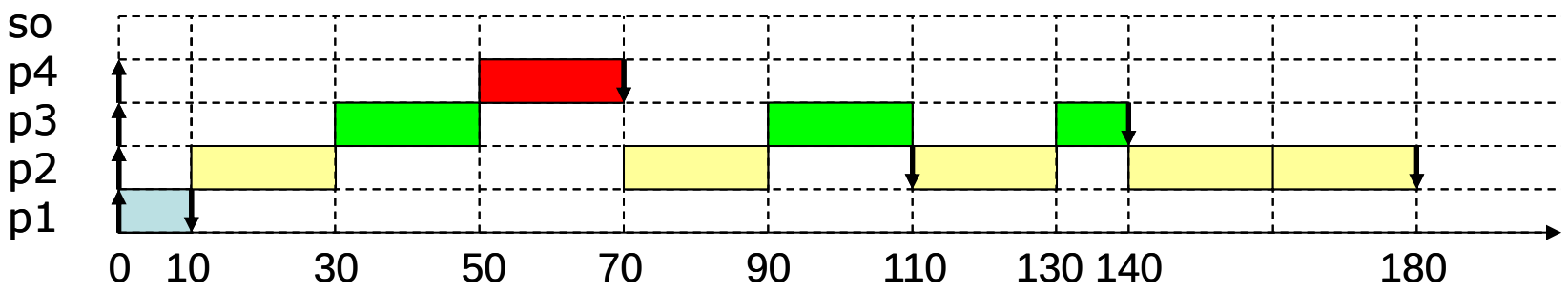
Am = (0 + 10 + 110 + 160)/4 = 70

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

tempo medio di risposta

Rm = (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

Am = (0 + 80 + 90 + 50)/4 = 55

tempo medio di risposta

Rm = (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 **Δ = 20 ms**.