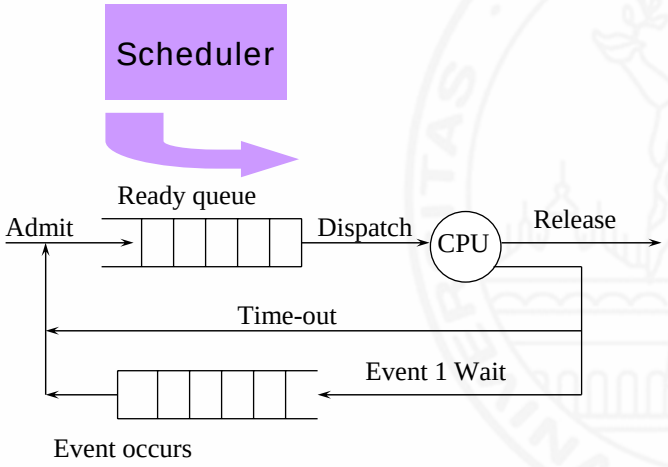
Scheduling

Sistemi Operativi e Distribuiti
A.A. 2004-2005
Bellettini - Maggiorini

Concetti di base

- Il massimo utilizzo della CPU si ottiene mediante la multiprogrammazione
- Ogni processo si alterna su due fasi
  - Uso della CPU
  - I/O
- CPU *burst*
  - Un periodo ininterrotto di utilizzo di
  - Analogamente si parla di I/O *burst*
- Obiettivo: distribuzione dell'utilizzo di CPU

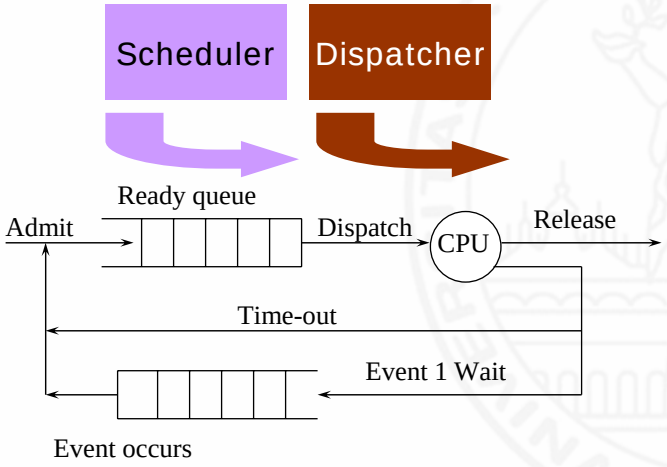
Lo scheduler della CPU



Lo scheduler della CPU

- Opera una selezione tra i vari processi in memoria pronti per l'esecuzione e decide a quale di loro andrà allocata la CPU
- Le decisioni di scheduling della CPU possono essere prese quando un processo:
  1. Cambia stato da running a waiting
  2. Cambia stato da running a ready
  3. Cambia stato da waiting a ready
  4. Termina
- Nei casi 1 e 4 si dice *non-preeemptive (senza prelazione)*
- Nei casi 2 e 3 si dice *preeemptive (con prelazione)*

Il Dispatcher



Il Dispatcher

- Il Dispatcher è quella parte di software del kernel che passa il controllo della CPU al processo selezionato dallo scheduler
- Si occupa di:
  1. Effettuare il context switch
  2. Passare in modalità utente (non privilegiata)
  3. Saltare alla giusta posizione del programma utente per riavviarne l'esecuzione
- *Latenza di Dispatch*
  - è il tempo impiegato dal dispatcher per fermare un processo e avviare l'esecuzione di un altro (tempo di context switch)

Criteri di scheduling

- Utilizzo della CPU
  - tenere il più possibile occupata la CPU (per fare lavoro utile)
- Throughput
  - numero di processi che terminano la loro esecuzione per unità di tempo
- Tempo di turnaround
  - tempo impiegato per eseguire un particolare processo

Criteri di scheduling

- Tempo di attesa
  - somma degli intervalli di tempo passati da un processo nella coda di ready
- Tempo di risposta
  - tempo impiegato da quando la richiesta viene fatta a quando la prima risposta viene prodotta, esclusa l'operazione di output (per gli ambienti time-sharing)

Criteri di ottimizzazione

- Massimizzare l'utilizzo della CPU
- Massimizzare il throughput
- Minimizzare il tempo di turnaround
- Minimizzare il tempo di attesa
- Minimizzare il tempo di risposta

Diagramma Gantt

- SI tratta di un diagramma che rappresenta l'andamento temporale della CPU (chi la stà usando)
  - Sull' asse X il tempo
  - Sull' asse Y i processi
- Quando un certo processo usa la CPU allora il diagramma risulta *occupato* in quel punto

Algoritmo First-Come, First-Served (FCFS)

- Il primo processo a presentarsi allo scheduler sarà quello che per primo si allocherà la CPU
  - Gli altri verranno gestiti secondo la sequenza di arrivo
- Ogni processo che si impossessa della CPU ne mantiene il controllo fino alla fine dell' esecuzione
- È il caso più semplice

FCFS

Supponendo che i seguenti tre processi arrivino contemporaneamente e vengano gestiti nell'ordine : P_1 , P_2 , P_3

Processo	Tempo di arrivo	Tempo di burst
P_1	0	24
P_2	0	3
P_3	0	3

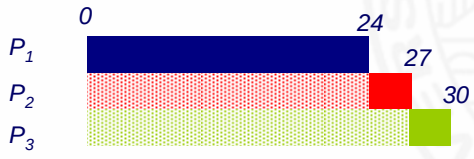
FCFS

Processo	Tempo di arrivo	Tempo di burst
P_1	0	24
P_2	0	3
P_3	0	3

FCFS

Processo	Tempo di arrivo	Tempo di burst
P_1	0	24
P_2	0	3
P_3	0	3

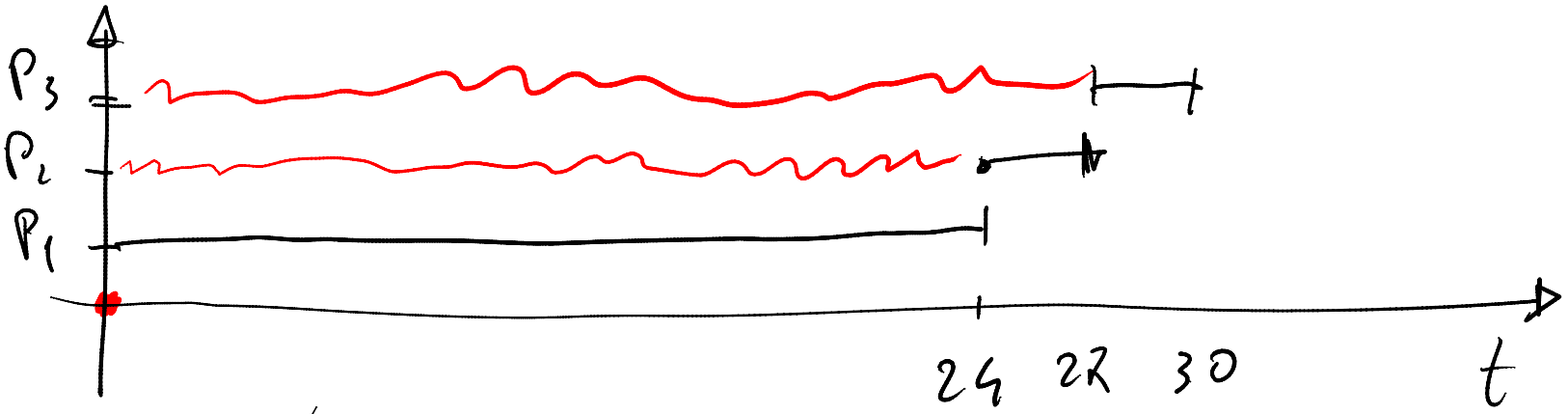
Diagramma di Gantt



- Tempi di attesa: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Tempo medio di attesa: $(0 + 24 + 27)/3 = 17$

P_1	0	24
P_2	0	3
P_3	0	3

$\frac{0+24+27}{3}$



R → ~~P_1~~ P_2 P_3

Dipendenza dall'ordine di arrivo

- Il rendimento del sistema varia a seconda dell' ordine di arrivo dei processi allo scheduler
- Nel caso dell' FCFS abbiamo l'*effetto convoglio*
  - Processi brevi rimangono in coda ad aspettare processi molto lunghi
- Supponiamo che i tre processi vengano gestiti nell' ordine P_2 , P_3 , P_1

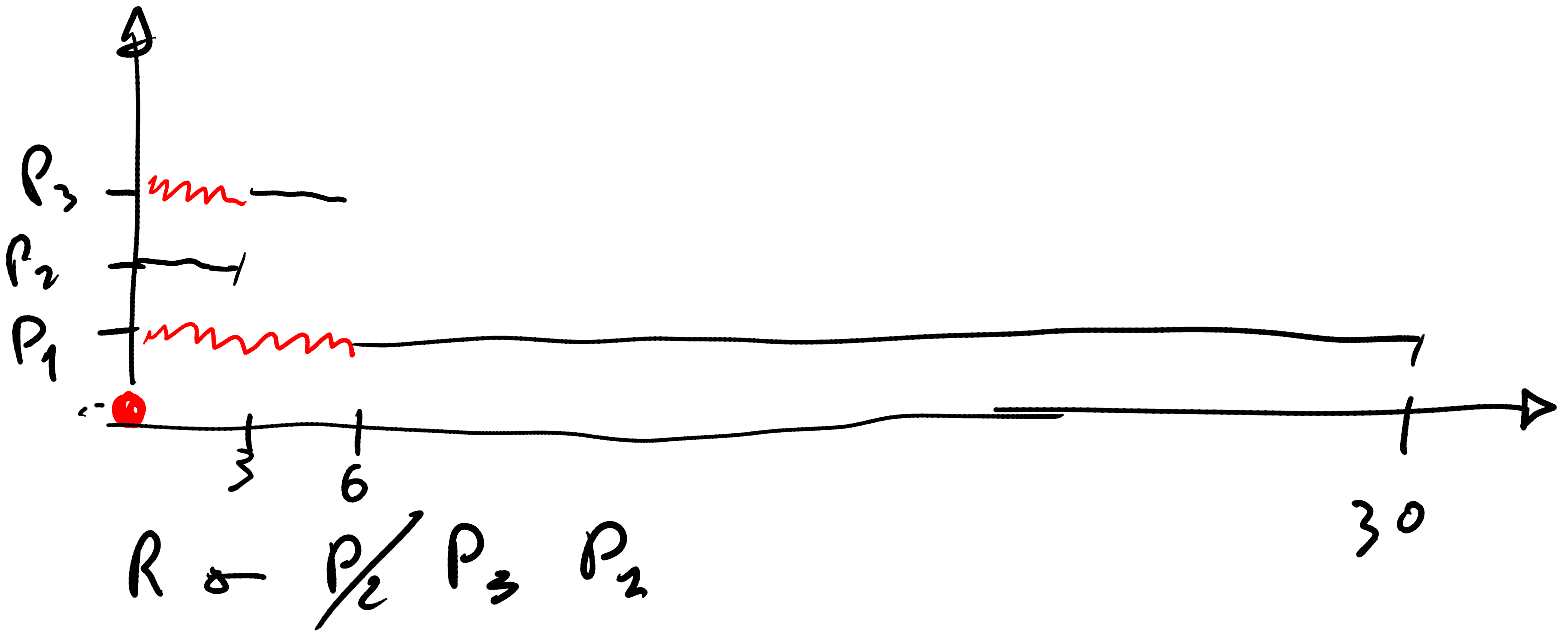
FCFS

(con ordine P_2 , P_3 , P_1)

Processo	Tempo di arrivo	Tempo di burst
P_1	0	24
P_2	0	3
P_3	0	3

P_1	0	24
P_2	0	3
P_3	0	3

$$\frac{6+0+3}{3} = 3$$

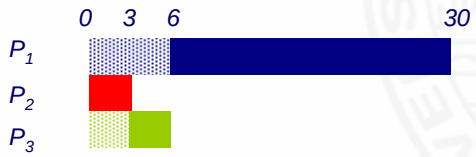


FCFS

(con ordine P_2 , P_3 , P_1)

Processo	Tempo di arrivo	Tempo di burst
P_1	0	24
P_2	0	3
P_3	0	3

Diagramma di Gantt



- Tempi di attesa: $P_1 = 6;\ P_2 = 0;\ P_3 = 3$
- Tempo medio di attesa: $(6 + 0 + 3)/3 = 3$

Algoritmo Shortest-Job-First (SJR)

- Viene tenuto conto del tempo di burst rimanente ad ogni processo
  - Lo scheduler sceglie il processo con il tempo di burst più breve
- Due possibilità:
  1. Nonpreemptive
     - una volta che la CPU è assegnata al processo non può essere prelazionata fino alla fine del suo burst
  2. Preemptive
     - se arriva un nuovo processo con lunghezza di CPU burst inferiore al tempo rimanente al processo in esecuzione, viene prelazionata.
     - Questo schema è chiamato anche *Shortest-Remaining-Time-First* (SRTF)
- SJF è ottimale
  - si ottiene il minimo tempo di attesa medio per un dato insieme di processi.

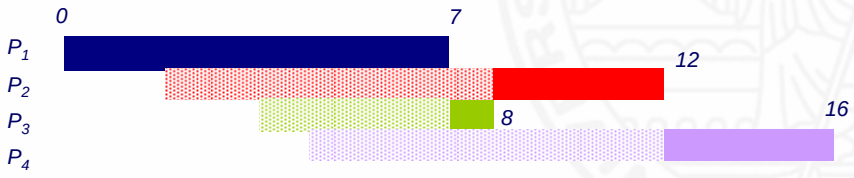
SJF senza prelazione

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

SJF senza prelazione

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

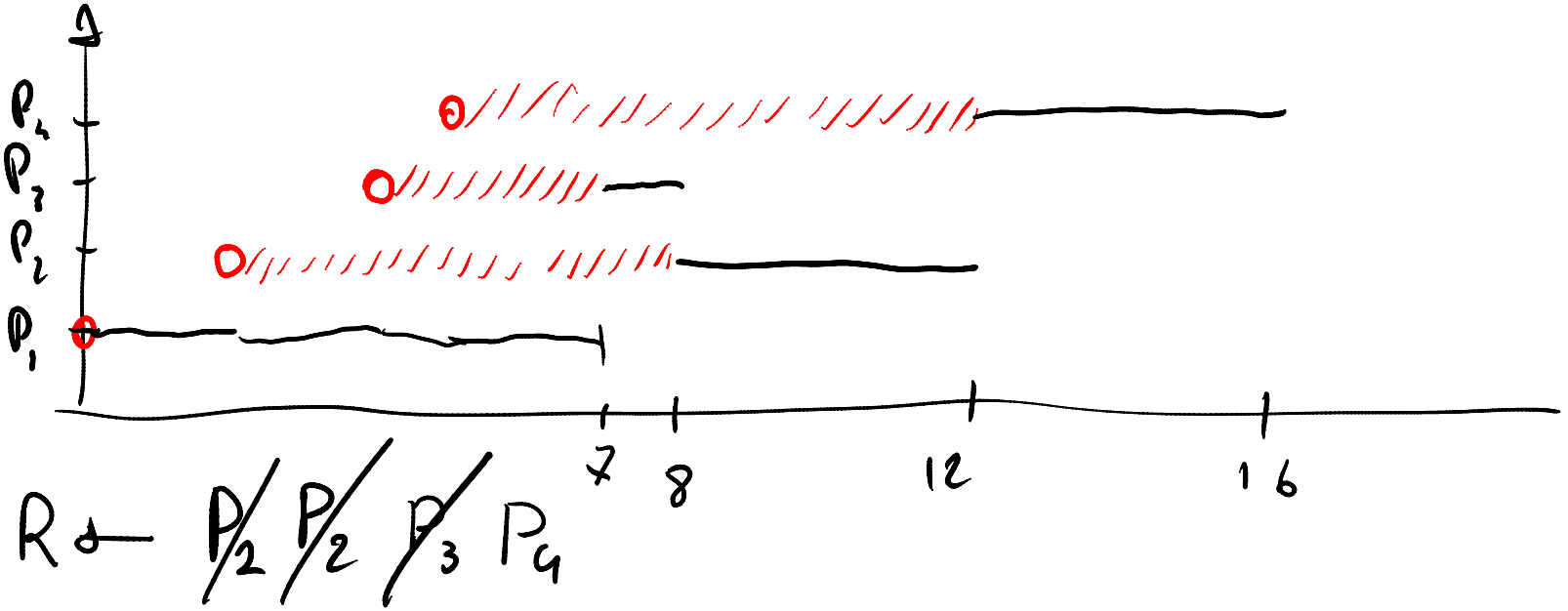
Diagramma di Gantt



Tempo medio di attesa: (0 + 6 + 3 + 7)/4 = 4

P_1	0	~~2~~ 5 ~~3~~ 2
P_2	2	4
P_3	4	1
P_4	5	4

$\frac{0+6+3+7}{4} = 4$



Rd — P_2 P_2 P_3 P_4

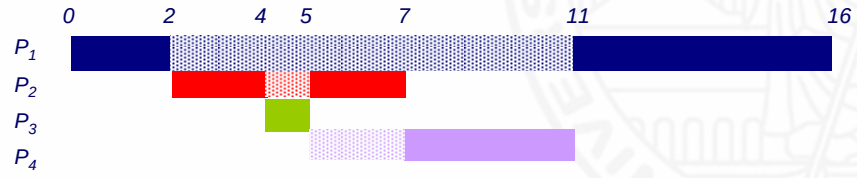
SJF con prelazione

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

SJF con prelazione

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

Diagramma di Gantt



Tempo medio di attesa: (9 + 1 + 0 + 2)/4 = 3

Tempo medio di risposta: (0 + 0 + 0 + 2)/4 = 0.5

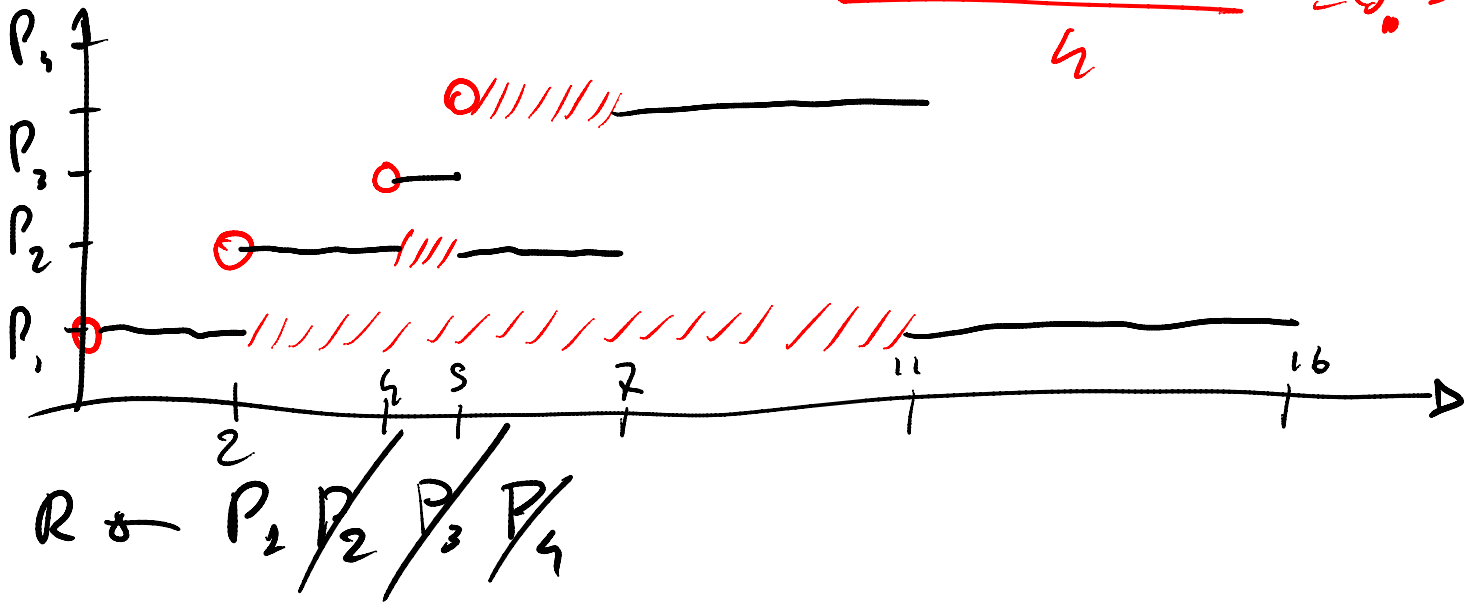
P_1	0	~~7~~ 5
P_2	2	~~4~~ ~~2~~ 0
P_3	4	1
P_4	5	4

ATTESA

$$\frac{9+1+0+2}{4} = 3$$

RISPOSTA

$$\frac{0+0+0+2}{4} = 0.5$$

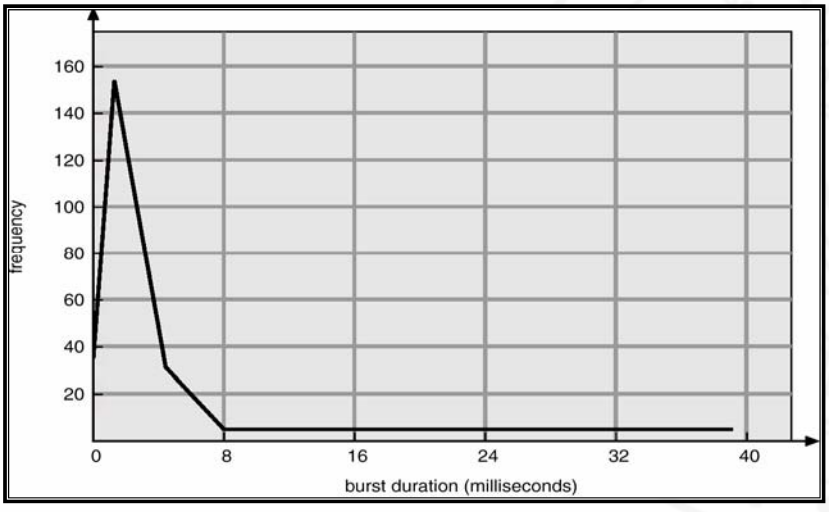


R ← ~~P_1~~ P_2 ~~P_2~~ ~~P_3~~ ~~P_4~~

SJF

- Bello si, ma non implementabile
  - Non c'è modo di sapere per certo quale sarà il burst di CPU di una generica applicazione
- Tutto quello che possiamo fare è tentare di *indovinare* (stimare) se il burst di un processo sarà lungo oppure breve

Istogramma dei tempi di CPU burst



Determinare la lunghezza del prossimo CPU Burst

- La lunghezza si può solo stimare
- Considerando i CPU burst precedenti, si può usare la media esponenziale

1. t_n = lunghezza reale del n^{esimo} CPU burst
2. τ_{n+1} = valore previsto del prossimo CPU burst
3. α, $0 \le \alpha \le 1$
4. Definiamo : $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\tau_n$

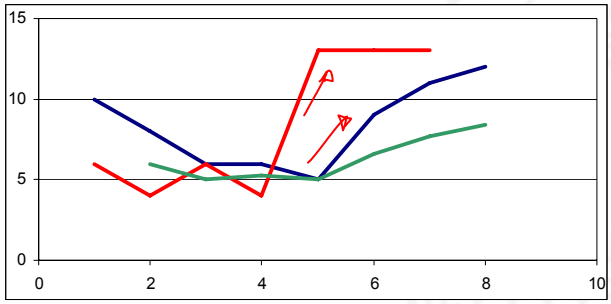
Esempi di media esponenziale

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La storia recente non conta
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Solo l'ultimo effettivo CPU burst conta
- Se espandiamo la formula otteniamo:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + (1-\alpha)^2\,\alpha\, t_{n-2} + \ldots$$
$$+ (1-\alpha)^j\,\alpha\, t_{n-j} + \ldots$$
$$+ (1-\alpha)^n\,\alpha\, t_0 + (1-\alpha)^{n+1}\,\tau_0$$

- Poiché sia α che $(1-\alpha)$ sono inferiori o uguali a 1, ogni termine successivo ha peso inferiore del proprio predecessore

Stima della lunghezza del prossimo Burst



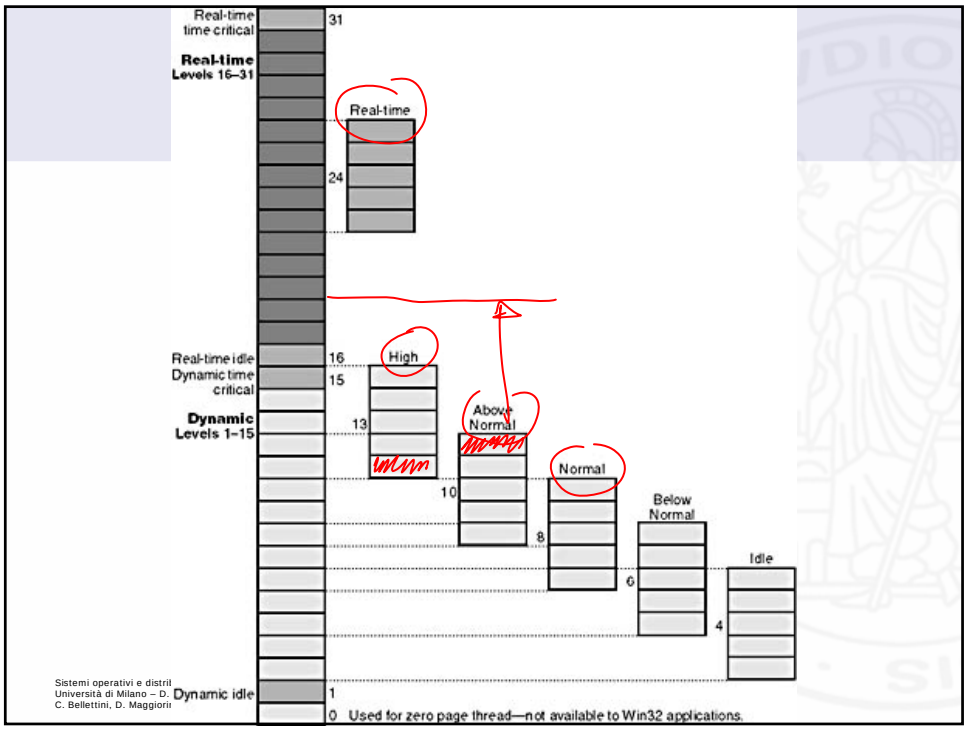
	1	2	3	4	5	6	7	8
Stima	10	8	6	6	5	9	11	12
Burst	6	4	6	4	13	13	13	
Media		6.0	5.0	5.3	5.0	6.6	7.7	8.4

Scheduling con priorità

- A ciascun processo viene associato **staticamente** un valore di priorità (un intero)
- La CPU viene assegnata al processo con la priorità più alta
  - (intero più piccolo ≡ priorità più alta)
  - Anche in questo caso possiamo avere due varianti:
    - Preemptive
    - Nonpreemptive
- SJF è uno scheduling con priorità dove questa è il tempo stimato del prossimo CPU burst
- Problema ≡ *Starvation*
  - i processi a bassa priorità potrebbero non venire mai eseguiti
- Soluzione ≡ *Invecchiamento*
  - con il passare del tempo aumentiamo la priorità dei processi in wait

Priorità in Windows 2000

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	8
above normal	25	14	11	9	7	5
normal	24	12	9	7	5	3
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1



Priority Boosts

- Esistono quattro casi in cui viene incrementato automaticamente il livello di priorità
  1. Quando viene completata una operazione di I/O (l'entità dipende da tipo periferica)
  2. Dopo avere atteso su un evento o semaforo
  3. Quando un Thread che si occupa di una interfaccia si sveglia per qualche attività
  4. Quando un processo pronto è tanto che non viene schedulato
- I *Realtime thread* non sono soggetti a boost
- Non si può superare 15 con il boost

Round Robin (RR)

- Ad ogni processo viene assegnata una piccola quantità di tempo di CPU (*time quantum*), solitamente 10-100 millisecondi
- Allo scadere di questo tempo, il processo viene prelazionato e aggiunto alla fine della coda ready
- Se ci sono n processi nella coda ready e il time quantum è q, ogni processo riceve $1/n$ del tempo di CPU in blocchi di al più q unità di tempo alla volta.
  - Nessun processo aspetta più di $(n-1)q$ unità di tempo.
- Prestazioni
  - q grande $\Rightarrow$ FIFO
  - q piccolo $\Rightarrow$ stallo
    - q deve essere grande rispetto al tempo di context switch, altrimenti l'overhead è troppo alto.

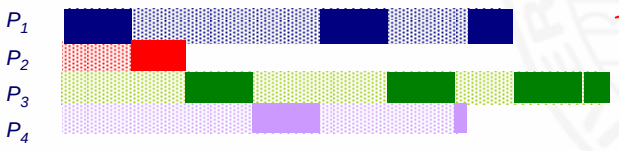
RR (quanto di tempo = 20)

Processo	Tempo di arrivo	Tempo di burst
P_1	0	53
P_2	0	17
P_3	0	68
P_4	0	24

RR (quanto di tempo = 20)

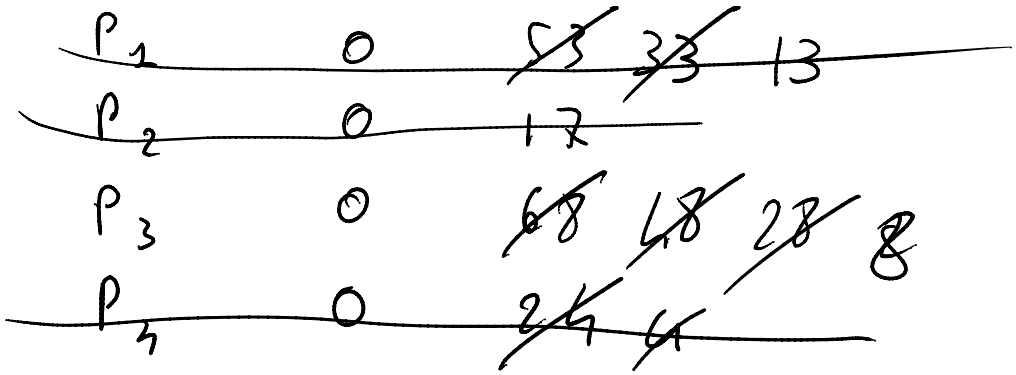
Processo	Tempo di arrivo	Tempo di burst
P_1	0	53
P_2	0	17
P_3	0	68
P_4	0	24

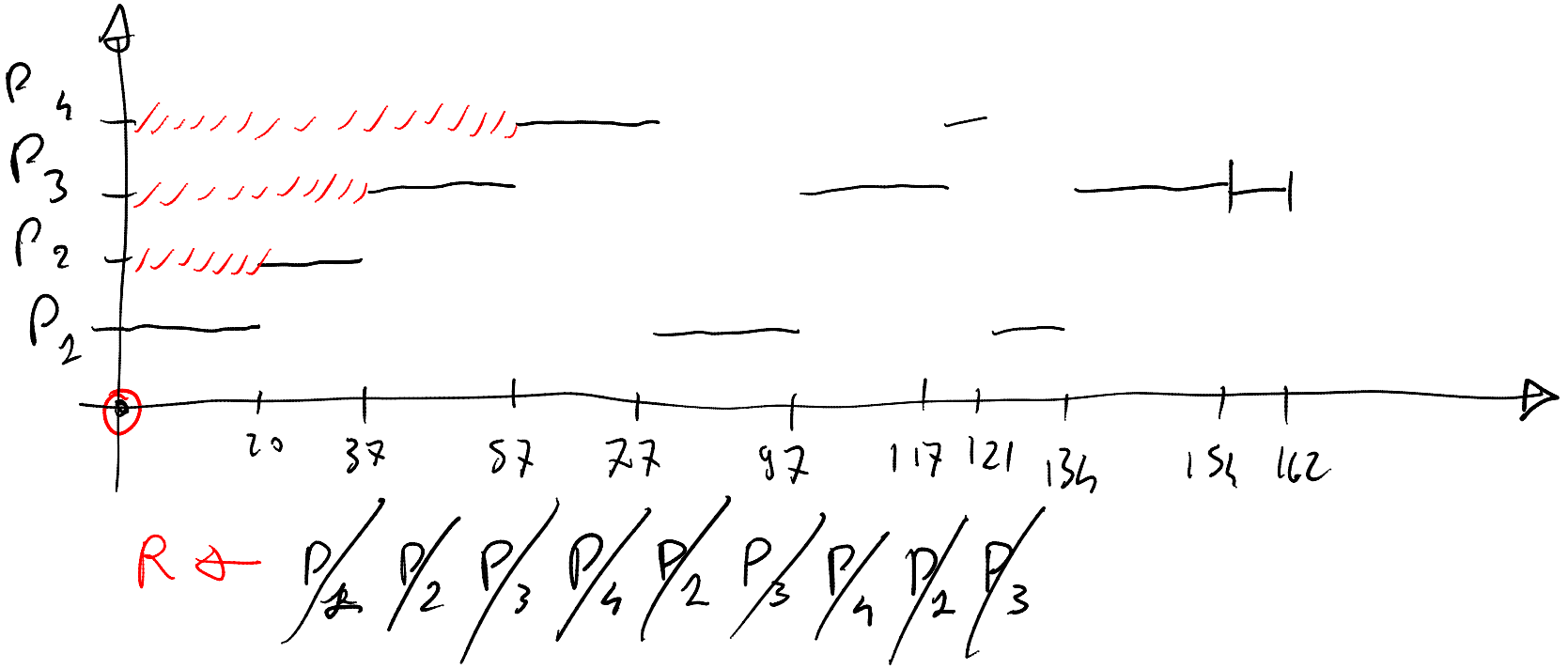
Diagramma di Gantt



0+20+37+57 = 285
4

Mediamente il tempo di tornaround è più alto di SJF, ma otteniamo un miglior tempo di risposta


P1 0 53 33 13
P2 0 17
P3 0 68 48 28 8
P4 0 24 4


P4
P3
P2
P1
0
20
37
57
77
97
117
121
134
154
162
R

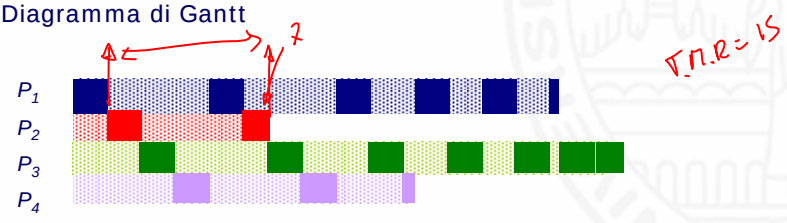
RR (quanto di tempo = 10)

Processo	Tempo di arrivo	Tempo di burst
P_1	0	53
P_2	0	17
P_3	0	68
P_4	0	24

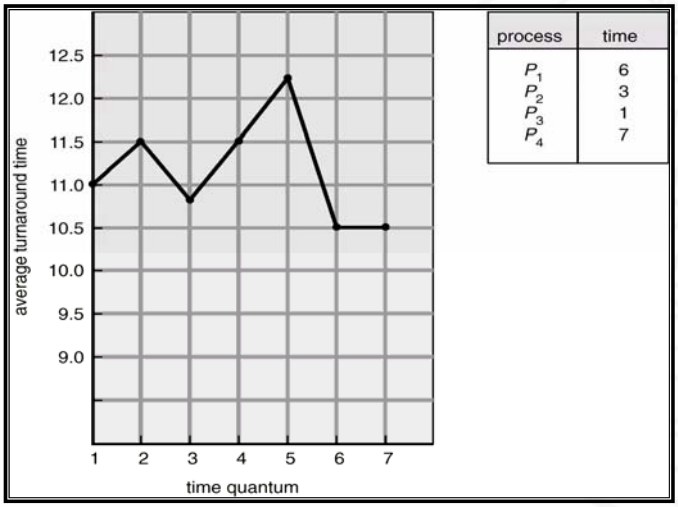
RR (quanto di tempo = 10)

Processo	Tempo di arrivo	Tempo di burst
P_1	0	53
P_2	0	17
P_3	0	68
P_4	0	24

Diagramma di Gantt



Rendimento e quanto di tempo



process	time
P_1	6
P_2	3
P_3	1
P_4	7