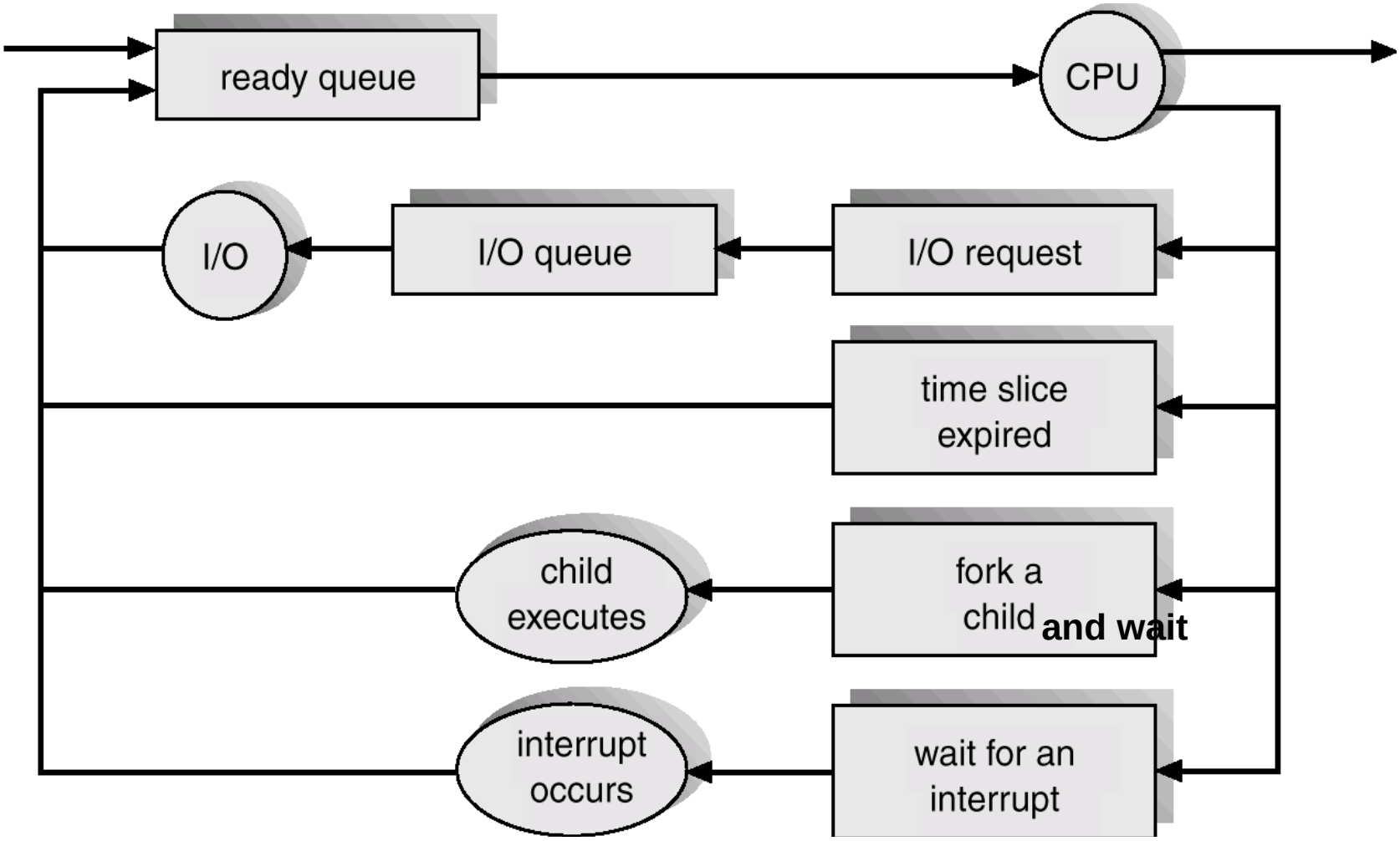
Scheduling

Nel sistema di elaborazione esistono diverse code:

- Coda dei job – tutti i processi di cui è richiesta l'esecuzione
- Coda dei processi ready – insieme dei processi caricati in memoria e pronti ad usare la CPU
- Code dei dispositivi – insieme delle richieste di I/O pendenti per i diversi dispositivi (e relativi processi).

I processi migrano da una coda all'altra nel corso della loro vita

Scheduling processi = selezione del processo da servire tra quelli in coda

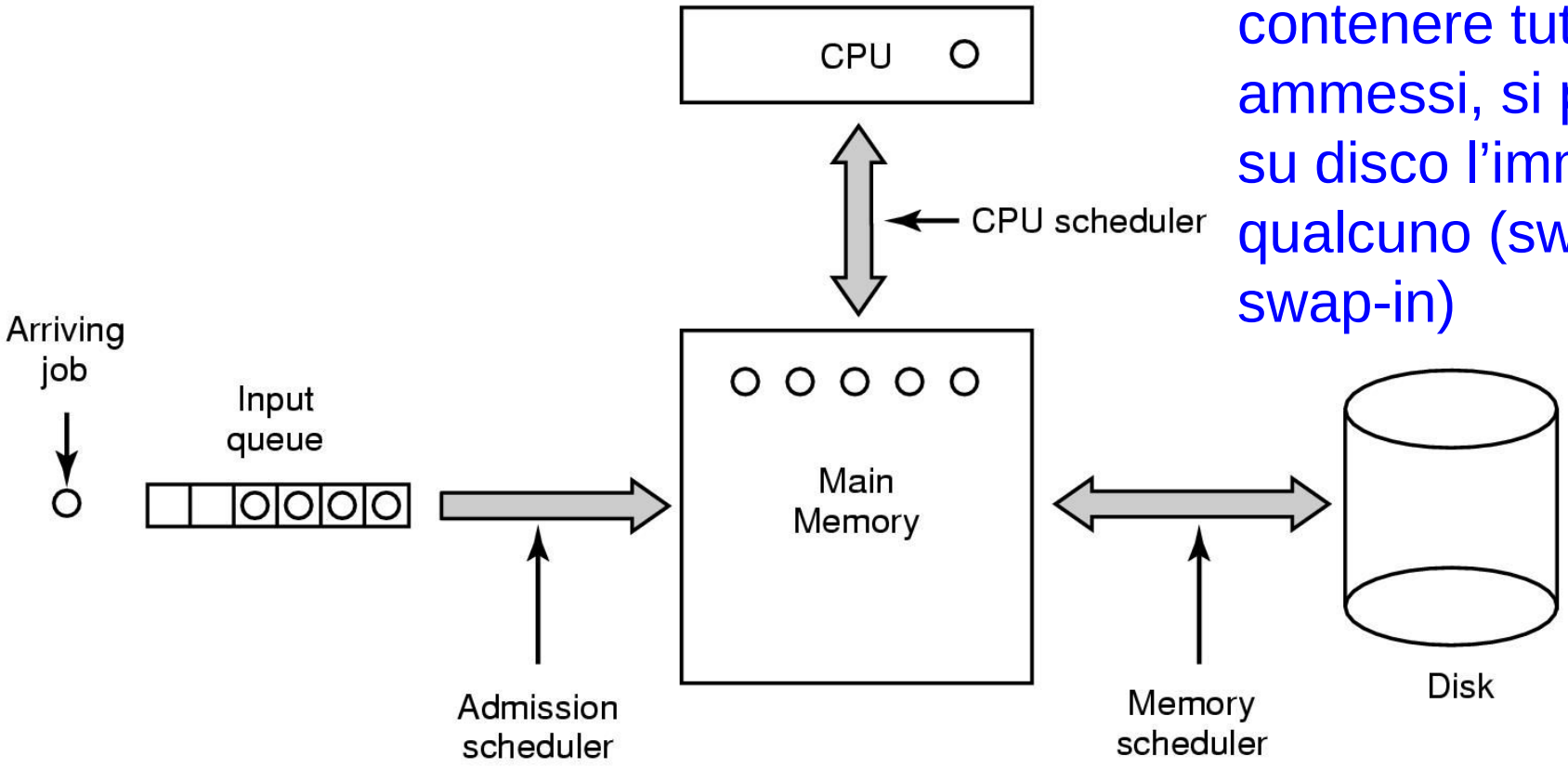


Tipi di Scheduler

- Long-term scheduler (job scheduler) – Decide quali job portare in ready queue (solo sistemi *batch* (a lotti))
- Short-term scheduler (CPU scheduler) – seleziona il prossimo processo a cui assegnare la CPU (deve operare velocemente)
- Medium-term scheduler: regola il livello di multiprogrammazione (tramite swap-in / swap-out) in base alle risorse disponibili, in particolare la memoria

Livelli di Scheduling

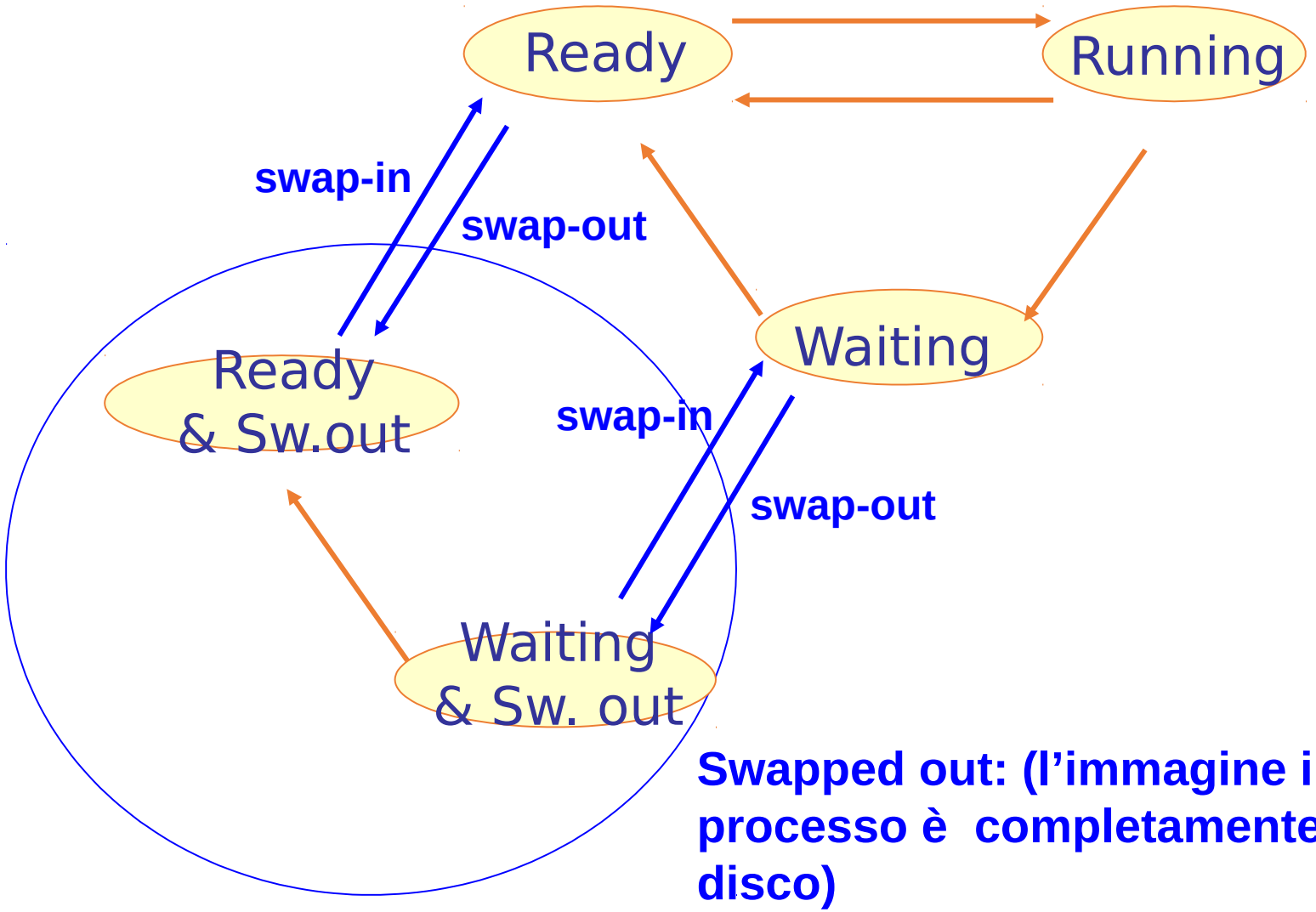
Se la memoria non può contenere tutti i processi ammessi, si parcheggia su disco l'immagine di qualcuno (swap-out / swap-in)



solo in sistemi a lotti (anticamera)

fa parte della gestione memoria: decide *il livello di multiprogrammazione*

Swapping



Swapped out: (l'immagine in memoria del processo è completamente scaricata su disco)

Obiettivi dello scheduling

Generali:

- assegnare il tempo di CPU ai processi in modo “equo” (non necessariamente uguale per tutti; almeno *un po’* di tempo)
- mantenere le risorse del sistema utilizzate

Per sistemi a lotti:

- massimizzare il *throughput* (numero medio di job eseguiti su una scala temporale grande, es. ora o giorno)
- minimizzare il tempo medio di *turnaround* (tempo intercorso fra richiesta di eseguire il job e risultati finali)

Per sistemi interattivi:

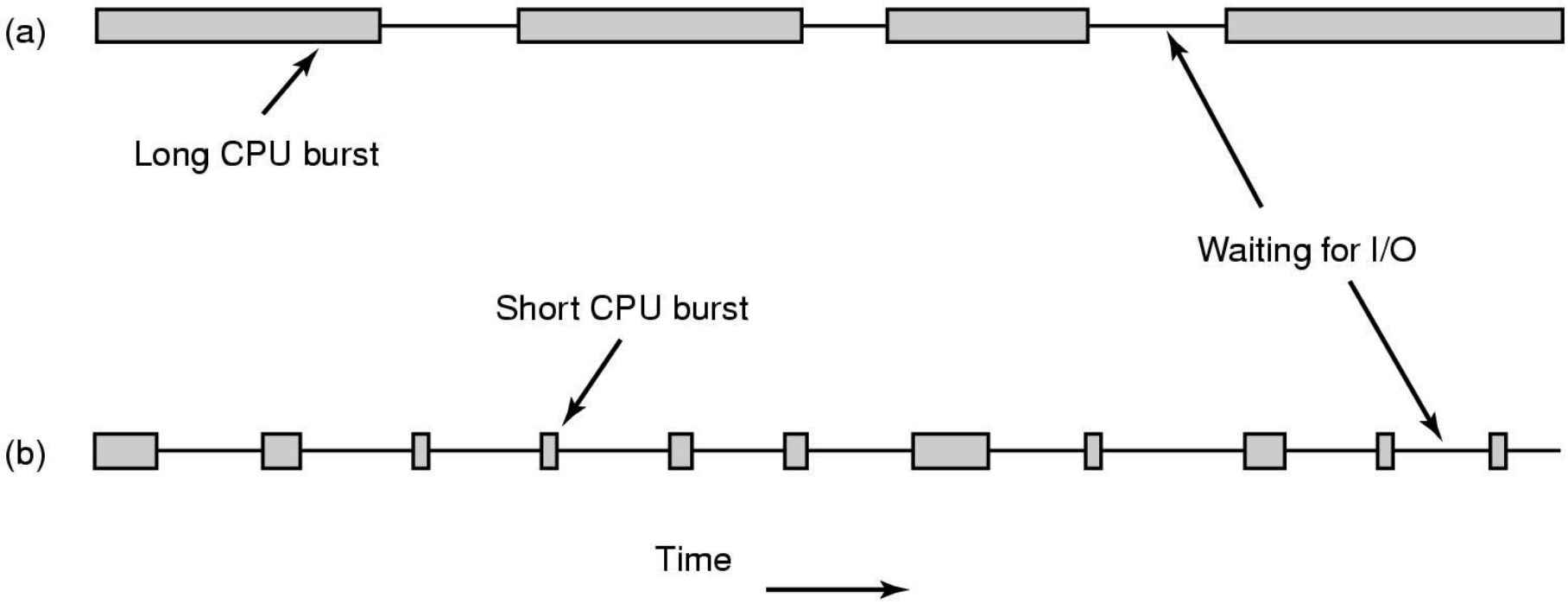
- minimizzare il tempo di risposta (tempo fra richiesta e risposta)
- soddisfare le richieste introducendo attese piccole o comunque in proporzione al tempo effettivo di esecuzione.

Per sistemi real-time:

- rispettare le scadenze, sempre o (versione soft) con poche eccezioni per non degradare la qualità del servizio

Mix di processi da sottoporre allo short-term scheduling

L'uso efficiente delle risorse è possibile sfruttando le diverse caratteristiche dei processi: conviene avere un *mix bilanciato di processi CPU bound(a) e I/O bound(b)*



Scheduling della CPU: quando

La decisione su qual è il prossimo processo da rendere *running* fra quelli pronti (*ready*) *può/deve* avvenire:

- quando il processo *running* termina
- quando il processo *running* effettua un'operazione sospensiva
- quando viene creato un nuovo processo
- quando c'è un'interrupt da disp. I/O (waiting -> ready)
- quando c'è un'interruzione da timer (oppure ogni k timer interrupt)

Deve avvenire nei primi due casi, quando il processo running rilascia volontariamente la CPU
Negli ultimi tre casi, viene chiamato in causa il S.O. che *può* decidere se far intervenire lo scheduler o no. Se si toglie la risorsa CPU al processo *running* si ha *preemption (prelazione)*

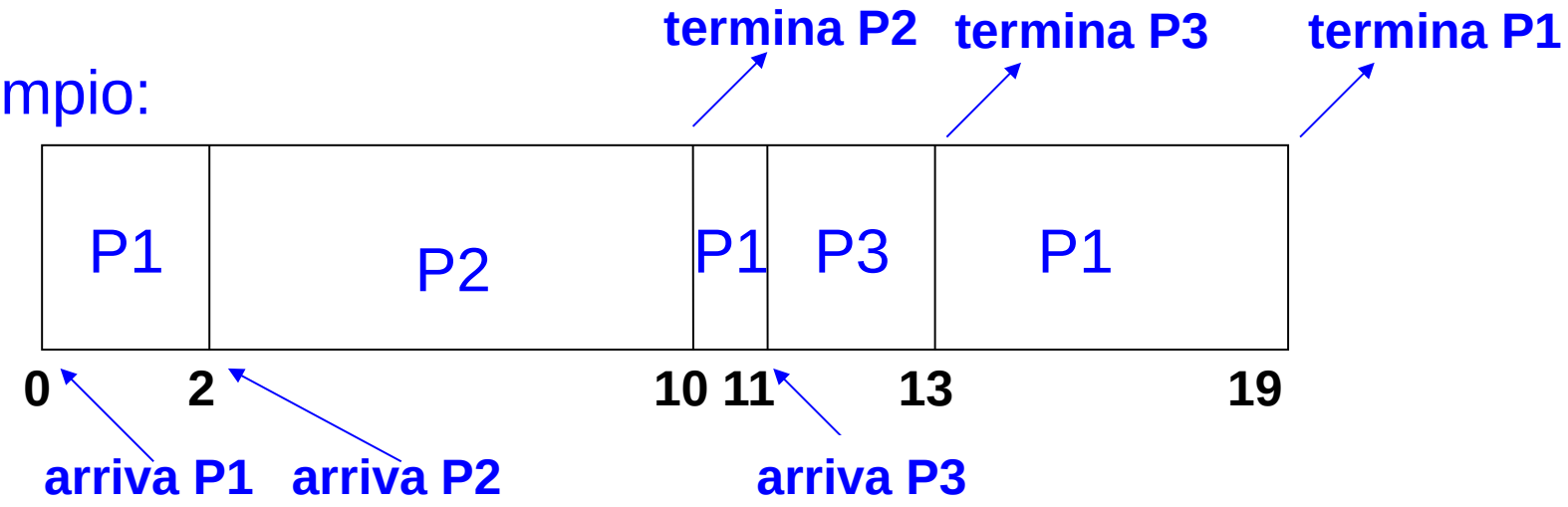
Rappresentazione scelte dello scheduler

Utilizzeremo i diagrammi di Gantt per rappresentare le scelte fatte da un dato algoritmo di scheduling in una data situazione. A partire dal Gantt potremo calcolare:

- il tempo di attesa di ciascun processo (tempo in anticamera o in ready queue)
- il tempo di permanenza di ciascun processo (da arrivo a terminazione)
- il tempo medio di attesa e di permanenza per l'insieme dei processi considerati

Sarà quindi possibile confrontare le prestazioni di ciascun algoritmo sulla base di tali tempi.

Esempio:

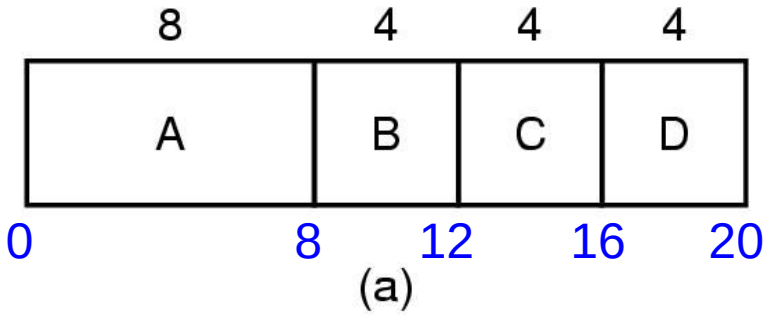


Scheduling per sistemi batch

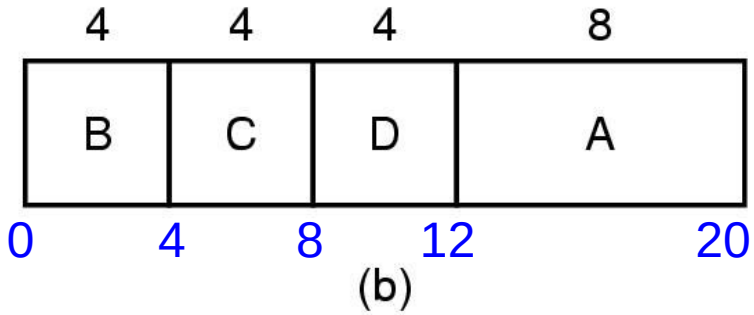
Vengono sottoposti diversi job, e supponiamo vengano eseguiti fino a completamento (ignoriamo la possibilità di usare la CPU per un altro mentre uno fa I/O)

- First-Come First-Served
- Shortest Job First: richiede di conoscere la durata dei job

Nota: entrambi criteri *NON preemptive (senza prelazione)*



FCFS: turnaround medio: 14



SJF: turnaround medio: 11

effetto convoglio (tutti dietro il più "lento" ad essere servito)

Se i job arrivano contemporaneamente, SJF è ottimale

Infatti (in questo caso con 4 job) se x_1 è la durata del primo processo che ha ottenuto la CPU, x_2 quella del secondo, ecc., il tempo medio di turnaround è:

$$(4x_1 + 3x_2 + 2x_3 + x_4) / 4$$

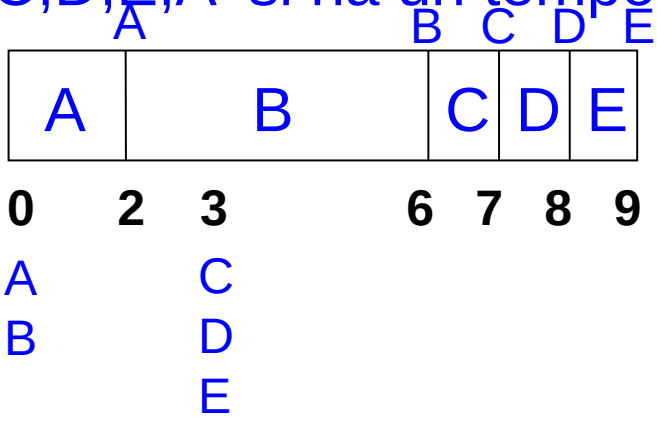
e per minimizzarlo deve essere $x_1 \leq x_2 \leq x_3 \leq x_4$

(ovviamente vale l'analogo per *n* job)

Scheduling per sistemi batch

SJF si può applicare anche se i job non arrivano *contemporaneamente* (si sceglie il più corto tra i job che rimangono) ma non è ottimale

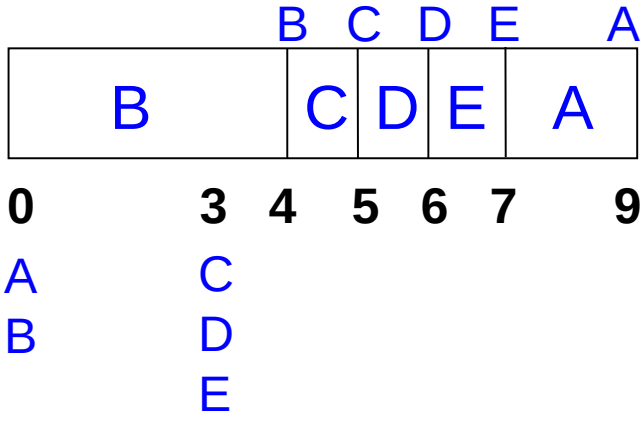
Es. se A, B, C, D, E arrivano all'istante 0,0,3,3,3, rispettivamente e richiedono rispettivamente 2,4,1,1, 1 unità di tempo di computazione, SJF schedulerebbe A,B,C,D,E che non è ottimale: infatti con l'ordine B,C,D,E,A si ha un tempo medio di attesa più basso



Term. - Arrivo

Turnar. - CPU

	Arrivo	CPU	Term.	Turnar.	Attesa
A	0	2	2	2	0
B	0	4	6	6	2
C	3	1	7	4	3
D	3	1	8	5	4
E	3	1	9	6	5
medie				4.6	2.8



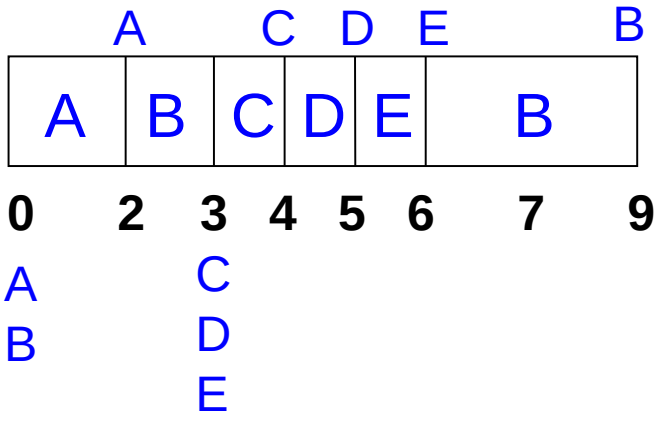
	Arrivo	CPU	Term.	Turnar.	Attesa
A	0	2	9	9	7
B	0	4	4	4	0
C	3	1	5	2	1
D	3	1	6	3	2
E	3	1	7	4	3
medie				4.4	2.6

Scheduling per sistemi batch

Per tener conto dell'arrivo distribuito nel tempo si può usare una versione modificata (preemptive) di SJF:

Shortest Remaining Time Next

Questo algoritmo **è preemptive**: può togliere la CPU a chi è running quando arriva un nuovo job



	Arrivo	CPU	Term.	Turnar.	Attesa
A	0	2	2	2	0
B	0	4	9	9	5
C	3	1	4	1	0
D	3	1	5	2	1
E	3	1	6	3	2
medie				3.4	1.6

Scheduling per sistemi batch

Come si fa a conoscere la durata di un job, per applicare SJF o SRTN nello scheduling a lungo termine?

Nei sistemi batch, quando si sottopone un job si fornisce anche una sovrastima del tempo di elaborazione, che viene usata come tempo limite: se viene superato, il job non viene completato

Un problema di SJF e SRTN: un processo può essere sorpassato da processi più corti e non girare mai (*starvation* = morte per fame), ne riparleremo

Scheduling a breve termine

L'idea di SJF e SRTN può essere applicata anche nello scheduling a breve termine (a chi dare la CPU fra i processi pronti) sulla base della durata stimata del prossimo CPU burst

Gira il processo con il prossimo CPU burst più piccolo (in base alla stima). In questo caso ci serve per avere un tempo medio di risposta piccolo e per anticipare (mediamente) le richieste ai dispositivi

Infatti al termine di questo CPU burst:

- l'utente ottiene presto una risposta e il processo si mette in attesa della reazione, oppure
- il processo sottopone presto (rispetto al caso in cui venga scelto uno con CPU burst più lungo) una operazione al dispositivo; la gestione del dispositivo avrà la richiesta presto e potrà mandarla avanti prima (se scarico) o ottimizzare la gestione delle richieste

Mentre l'utente "pensa" e/o il dispositivo lavora, mando avanti i processi con CPU burst più lunghi

Stima durate CPU burst

Bisogna stimare la durata del prossimo CPU burst, lo si può fare a partire dai dati relativi ai precedenti CPU-burst attraverso una media esponenziale:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n .$$

Dove:

t_n e` la durata effettiva dell'n-esimo CPU burst

α e' compreso tra 0 e 1

τ_{n+1} e` la stima dell'n+1-esimo CPU-burst

τ_0 ha un valore predefinito

Stima durate CPU burst

- α =0:
  - $\tau_{n+1} = \tau_n$
  - La storia non conta (si usa sempre lo stesso valore di default)

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$$

- α =1:
  - $\tau_{n+1} = t_n$
  - Conta solo l'ultimo CPU burst
- $0 < \alpha < 1$: Espandendo la formula si ottiene:

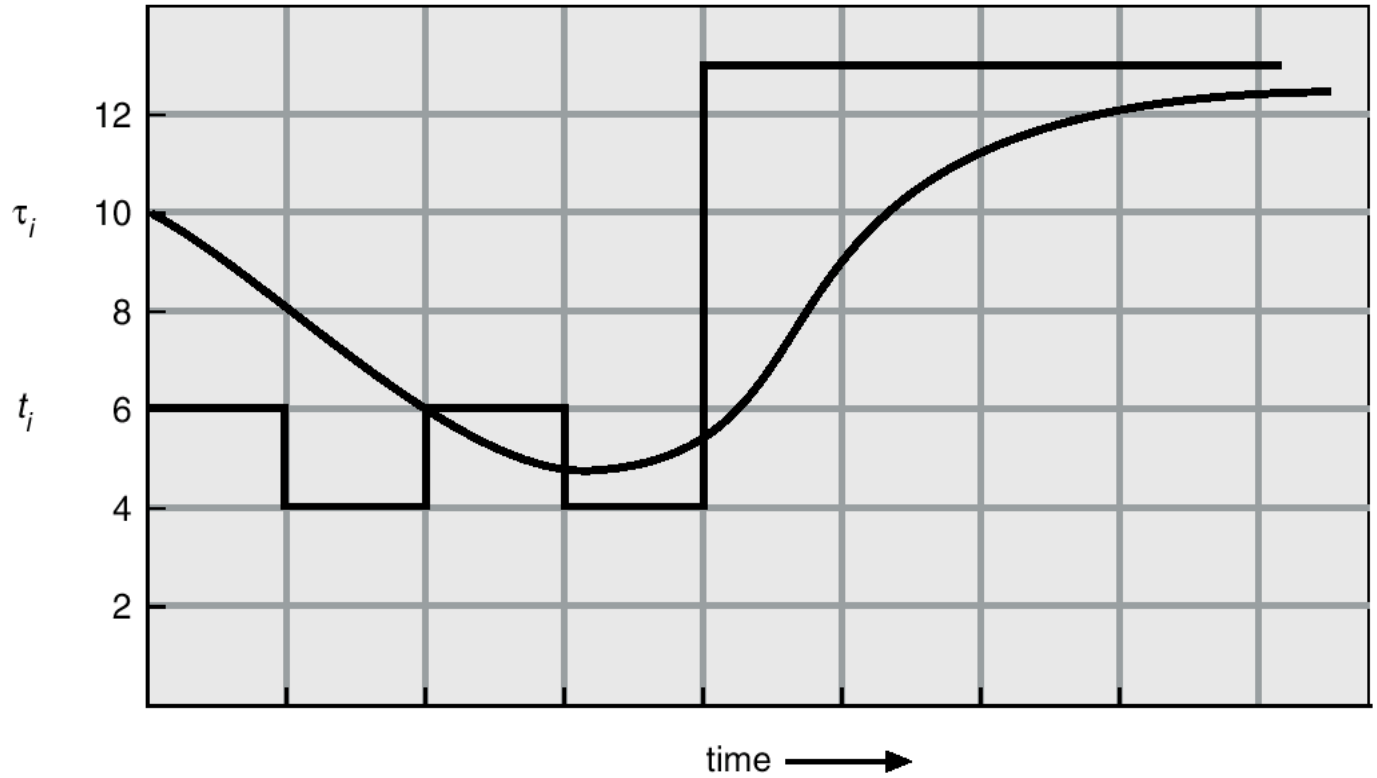
$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \ldots$

$+ (1 - \alpha)^j\, \alpha\, t_{n-j} + \ldots$

$+ (1 - \alpha)^{n+1}\, \tau_0$

Poiché α e $(1 - \alpha)$ sono entrambi ≤ 1, ciascun termine nella formula ha un peso inferiore dei precedenti

Stima durate CPU burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Esempio con α=*1/2*

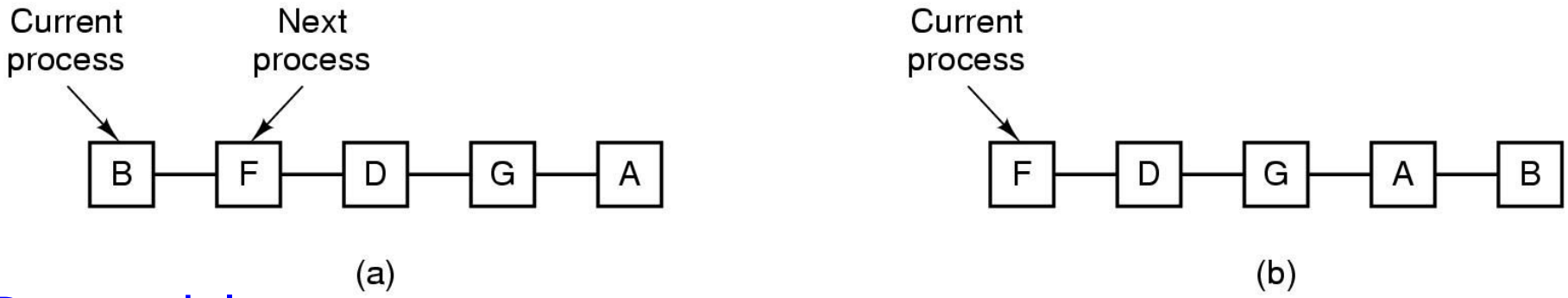
Ogni volta il contributo dei vecchi cpu-burst al calcolo della media si dimezza

Scheduling per sistemi interattivi

Un criterio più semplice per i sistemi interattivi è il

Round Robin:

se il processo corrente è rimasto running per un *quanto* di tempo (k timer interrupt, k≥1), va in fondo alla coda dei processi pronti, viene scelto il primo

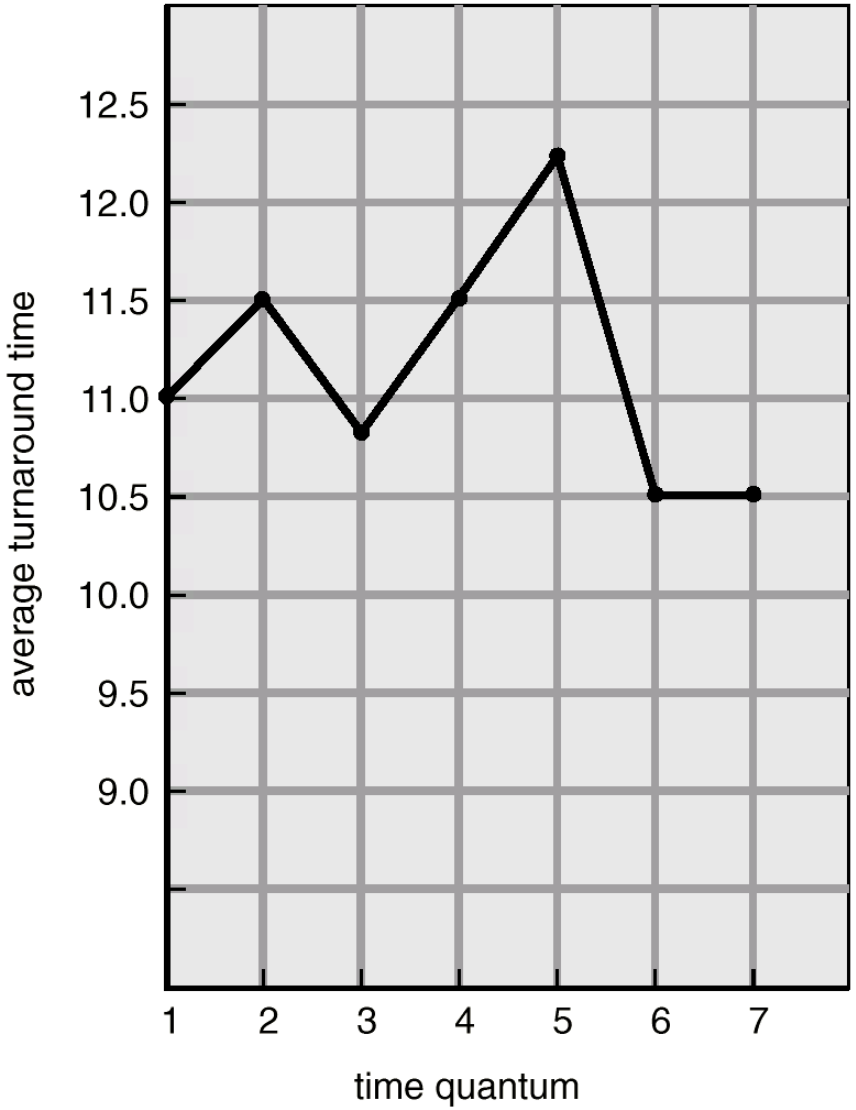


Durata del quanto:

- minimo 10 volte il tempo di context switch (per avere meno del 10% di tempo sprecato nello switch), possibilmente di più

  NB lo switch include anche flush della MMU
- non troppo (es. >>100 msec) per evitare tempi di risposta alti
- tipicamente tra 10-100 msec

Influenza del quanto di tempo sul tempo di permanenza

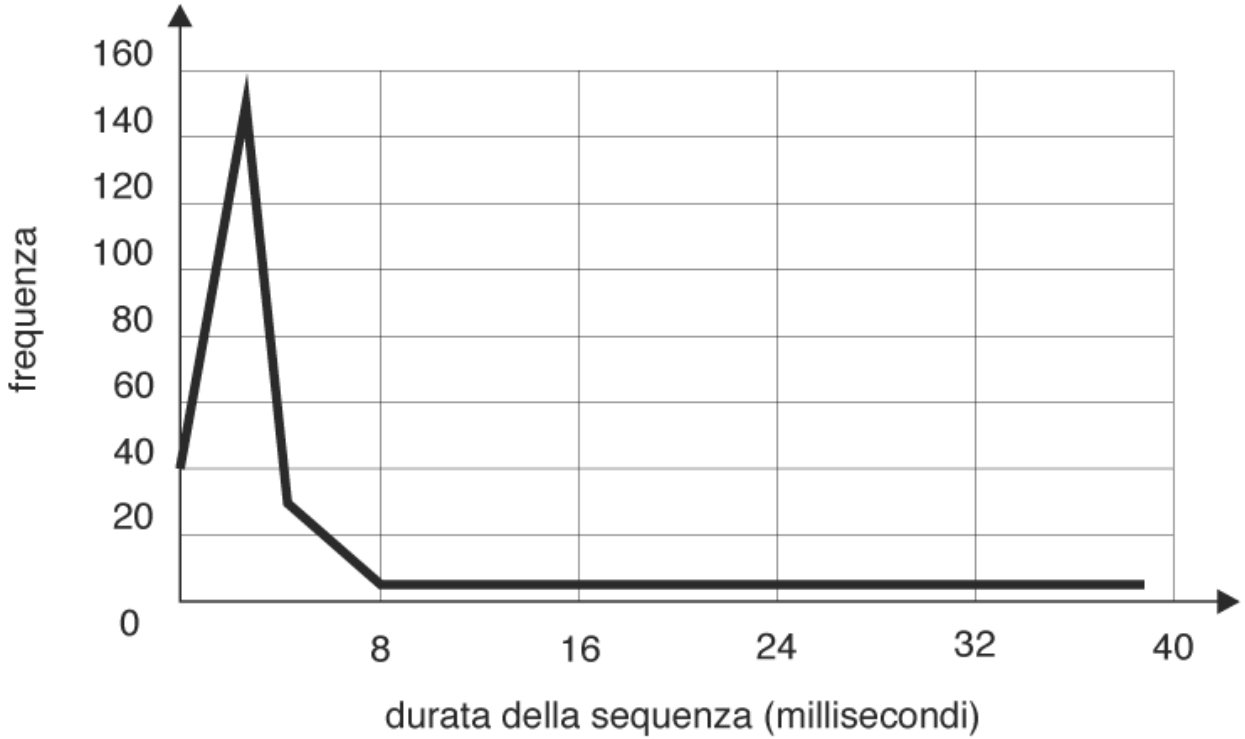


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Un quanto lungo tende a migliorare il tempo di turnaround, ma non c'è una relazione monotona garantita

Aumentando il quanto, RR tende a FCFS (qui coincide per q $\geq$6)

Scheduling round-robin



Misure effettuate su sistemi reali hanno rivelato che la maggior parte dei CPU burst sono piuttosto brevi. Avendo a disposizione un diagramma come quello mostrato in figura si può selezionare un valore di *q* che permetta ad una buona parte (es. l'80%) di CPU burst di completare l'esecuzione all'interno di un quanto. La scelta deve sempre essere bilanciata rispetto al requisito di mantenere un tempo di risposta accettabile.

Scheduling con priorità

Si può assegnare a diversi processi un diverso livello di priorità e fare in modo che la politica di scheduling (con o senza prelazione) scelga sempre il processo con priorità più alta tra quelli ready

- priorità definita in base a parametri interni al SO: caratteristiche del processo misurate o calcolate dal SO
- priorità definita in base a parametri esterni (es. tipo di utente)

Esempi di parametri interni:

- lunghezza CPU burst, quale politica si ottiene?

  SJF (o meglio “Shortest CPU burst” First),
  o SRTN se con prelazione

- Più in generale, processo I/O bound ⇒ priorità alta, per tenere i dispositivi di I/O alimentati e usare la CPU per gli altri processi nel frattempo

  Per sapere quanto è I/O bound, un altro criterio (oltre alla lunghezza assoluta dei CPU burst) è la parte usata degli ultimi quanti di tempo, es.:

  ultimo quanto usato per una frazione f

  ⇒ priorità 1/f

- processo che si risveglia da attesa di input da tastiera
- processo associato alla finestra selezionata

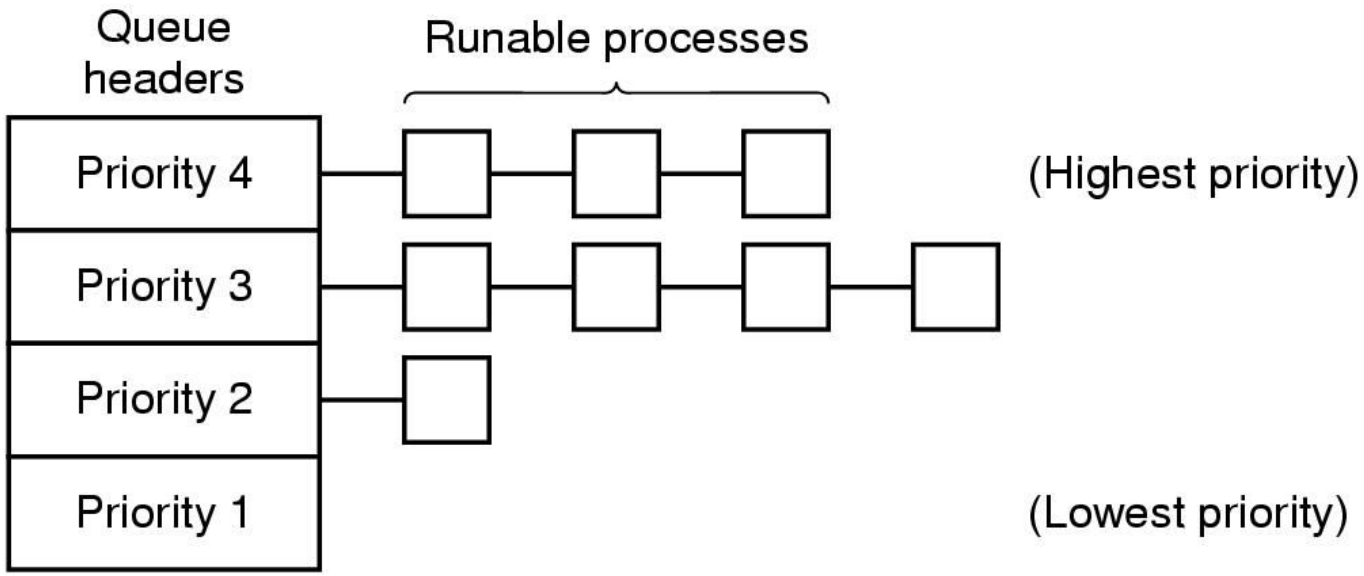
Scheduling con priorità

Problema con le priorità: un processo può avere sempre processi con priorità maggiore e quindi non girare mai (*starvation* = morte per fame)

Soluzione: incrementare la priorità con il passare del tempo dall'ultimo istante in cui il processo ha avuto la CPU (*aging* = invecchiamento)

Scheduling con priorità e code multiple

Può essere organizzato per classi:

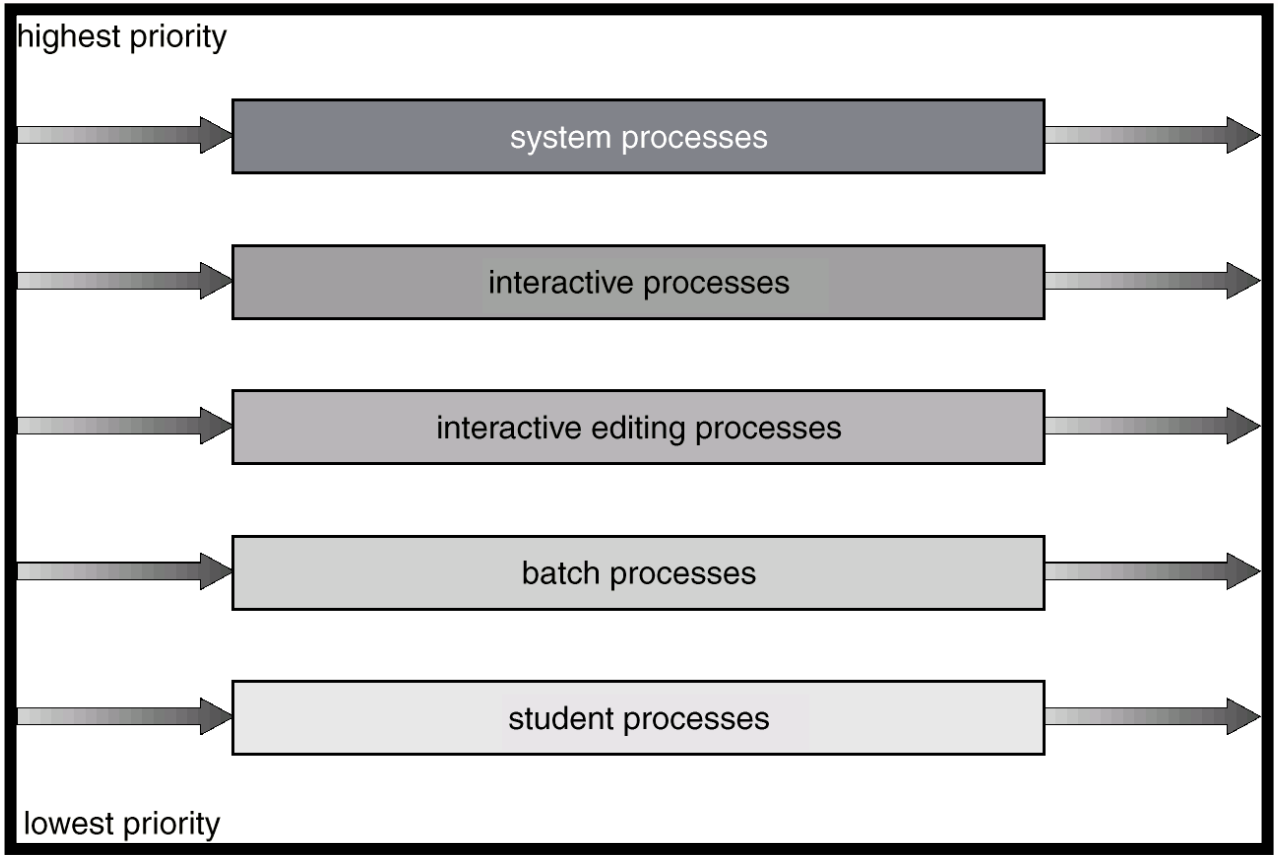


Gira un processo della classe più alta, se esiste

Scheduling uguale o diverso per ciascuna classe, es. Round Robin con q diversi

L'appartenenza alle classi può essere dinamica e in particolare dipendere dal comportamento passato (IO-bound vs. CPU-bound)

Scheduling con priorità e code multiple

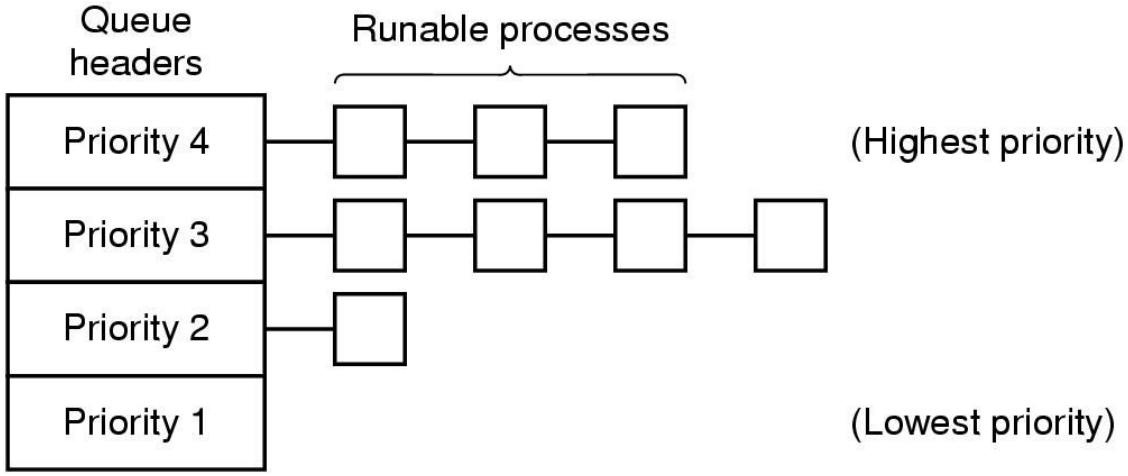


Qui vediamo un esempio di classi di priorità. la classificazione in interattivo o meno può essere automatica

Una variante alla *priorità assoluta* fra code può consistere nell'assegnare *percentuali* di tempo di CPU alle diverse code, gestendo ciascuna coda con un criterio apposito (Round Robin o altro)

Scheduling con priorità e code multiple

Multilevel Feedback Queues: appartenenza dinamica alle classi



Round robin con *q* diversi, più piccoli per le classi “alte”

Un processo inizia in una classe alta (siamo benigni) e:

- Come favorire processi con CPU burst corti senza fare stime per classificarli?

se usa tutto il quanto passa in una più bassa, con *q* maggiore

- Come evitare la *starvation* dei processi che finiscono nelle classi inferiori?

dopo un po’ di tempo (*aging*) in una classe bassa, o nella più bassa, viene riportato in alto

Scheduling a quote

Scheduling a quote: se ci sono n utenti, o n processi, far avere ad ognuno 1/n del tempo di CPU

Oppure quote diverse, a seconda di una priorità

Come realizzarlo (con una certa approssimazione):

dare la CPU a chi è maggiormente in difetto rispetto alla quota dovutagli. Può essere visto come un algoritmo a priorità dinamica, dove il livello di priorità è dato dal rapporto tra la quota spettante e la quota avuta (chi ha avuto la metà del dovuto ha priorità 2, chi ha avuto il doppio ha priorità 0.5).

Le quote si possono usare anche in un algoritmo a priorità con code multiple, assegnando una quota a ciascuna coda.

Scheduling a “lotteria”

Scheduling a lotteria: ogni processo ha un certo numero di biglietti (della lotteria) proporzionale alla percentuale di tempo di CPU al quale ha diritto.

- lo scheduler estrae a caso il “biglietto vincente”: i processi o gli utenti hanno una opportuna frazione di biglietti della lotteria avranno maggiore probabilita’ di essere schedulati
- un processo (per es. un client) può decidere di cedere parte dei suoi biglietti ad un altro processo (per es. il server a cui ha inoltrato una richiesta).

Rispetto alle quote, è più semplice aggiungere un processo

Alcuni esercizi …

First Come, First Served (FCFS)=FIFO

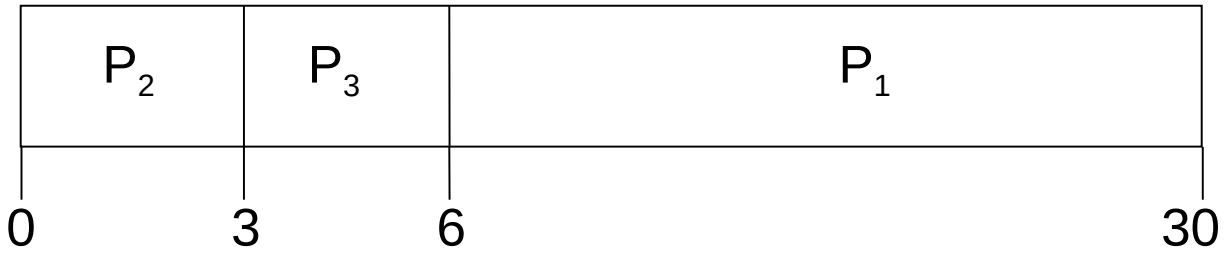
Processo	CPU burst
P_1	24
P_2	3
P_3	3

- Assumendo che l'ordine di arrivo sia: P_1 , P_2 , P_3 il Gantt per la politica FCFS è:

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

SJF Scheduling

Se invece applichiamo la politica SJF:



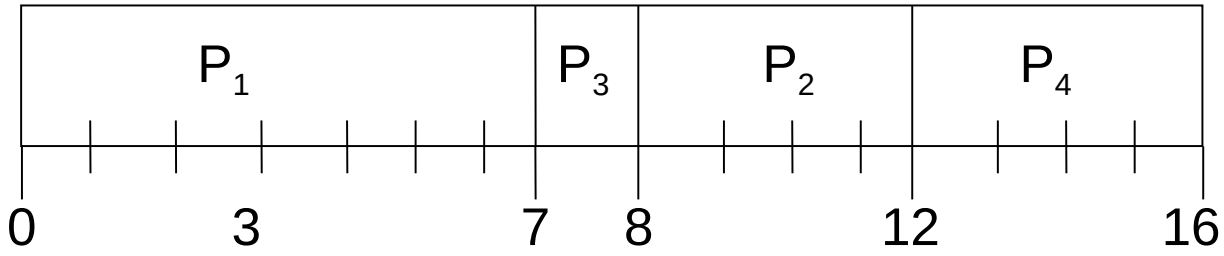
Tempo di attesa per $P_1 = 6; P_2 = 0, P_3 = 3$

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

Ancora SJF (arrivi scaglionati)

Processo	Tempo arr.	CPU burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

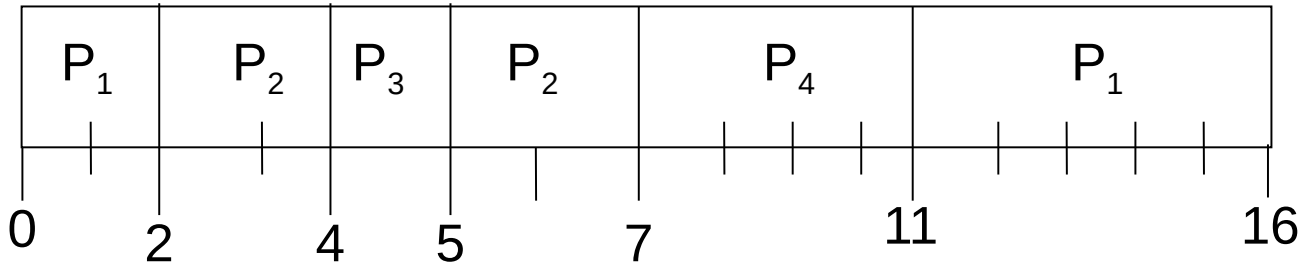


Tempo medio attesa = (0 + 6 + 3 + 7)/4 = 4

SRTN (SJF modificato con prelazione)

Processo	Tempo arr.	CPU burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- Politica SRTN (SJF preemptive)



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

MIGLIORE DEL SJF (Quando gli arrivi sono scaglionati l'algoritmo ottimo e' SRTN).

Esempio Round Robin con q=20

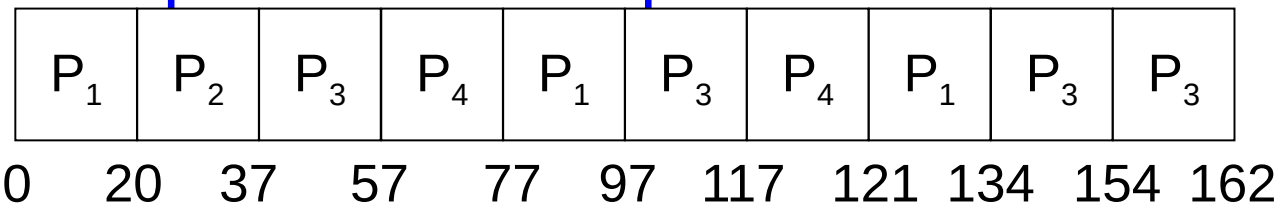
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Il Gantt per il RR con q=20:

Esempio Round Robin con q=20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Il Gantt per il RR con q=20:



- Tempo medio attesa=(81+20+94+97)/4=73
- Fornisce solitamente un tempo medio di attesa peggiore di SJF (38)