

# Sommario

## Concetto di scheduling

## Tipi di scheduling

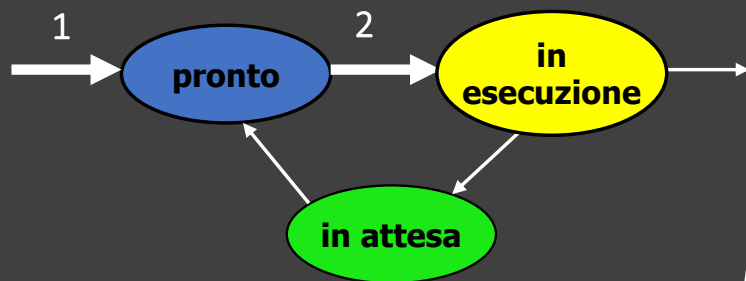
- Lungo termine
- Breve termine (scheduling della CPU)
- Medio termine

## Scheduling della CPU

- Definizioni
- Modello del sistema
- Criteri di scheduling
- Algoritmi di scheduling

Concetto di scheduling

# Scheduling dei processi



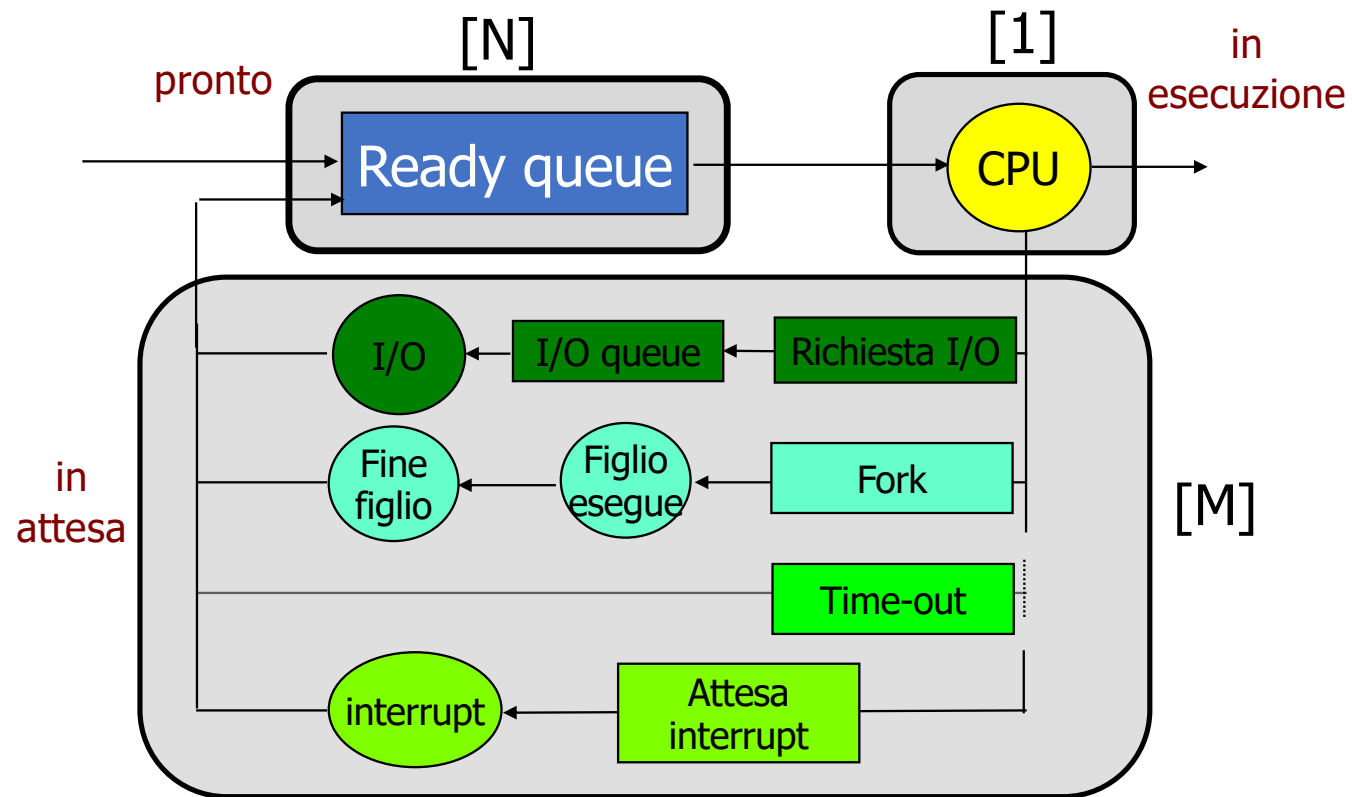
## Scheduling:

- assegnazione di attività nel tempo

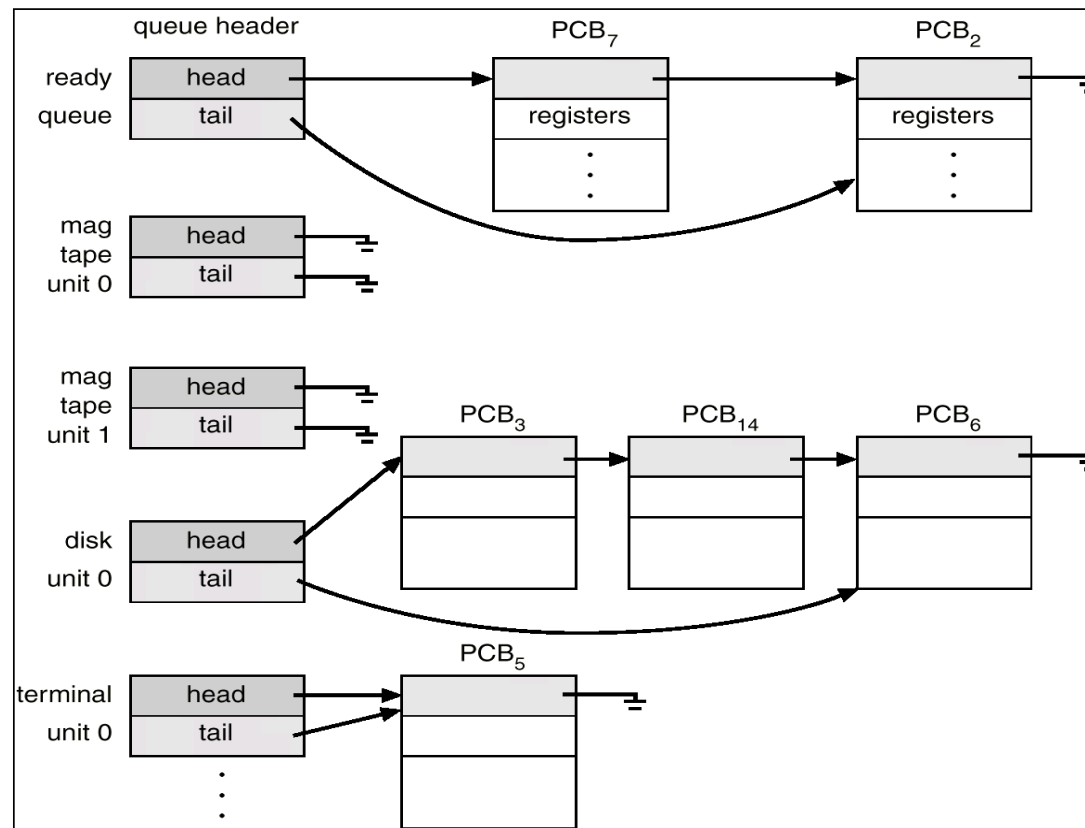
Utilizzo multiprogrammazione impone l'esistenza di una strategia per regolamentare:

- (1) ammissione dei processi nel "sistema" (memoria)
- (2) ammissione dei processi all'esecuzione (CPU)

# Diagramma di accodamento



# Implementazione delle code



# Tipi di scheduling

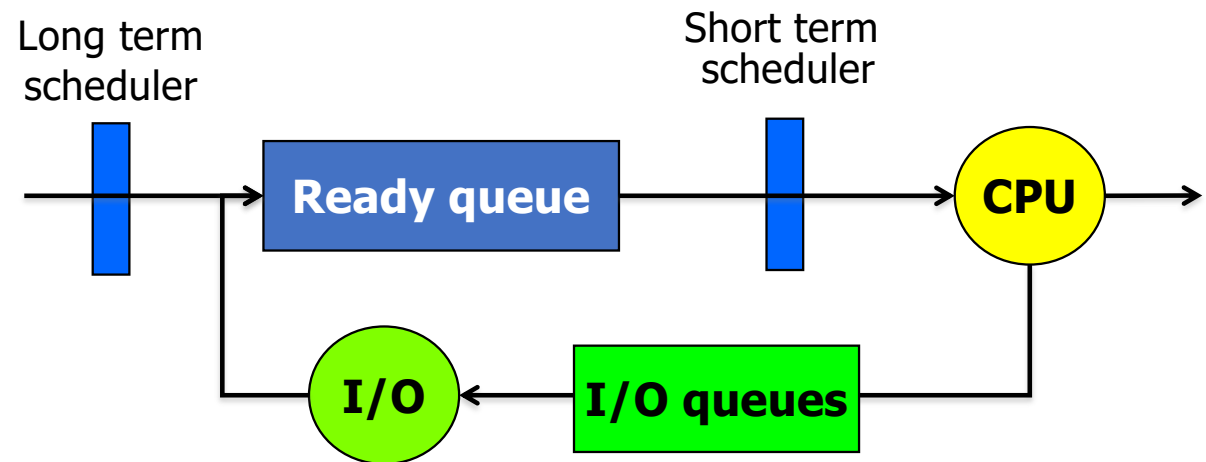
# Tipi di scheduler

## Scheduler a lungo termine (job scheduler)

- Seleziona quali processi devono essere portati dalla memoria alla ready queue

## Scheduler a breve termine (CPU scheduler)

- Seleziona quale processo deve essere eseguito dalla CPU



# Caratteristiche degli scheduler

Scheduler a breve termine è invocato spesso

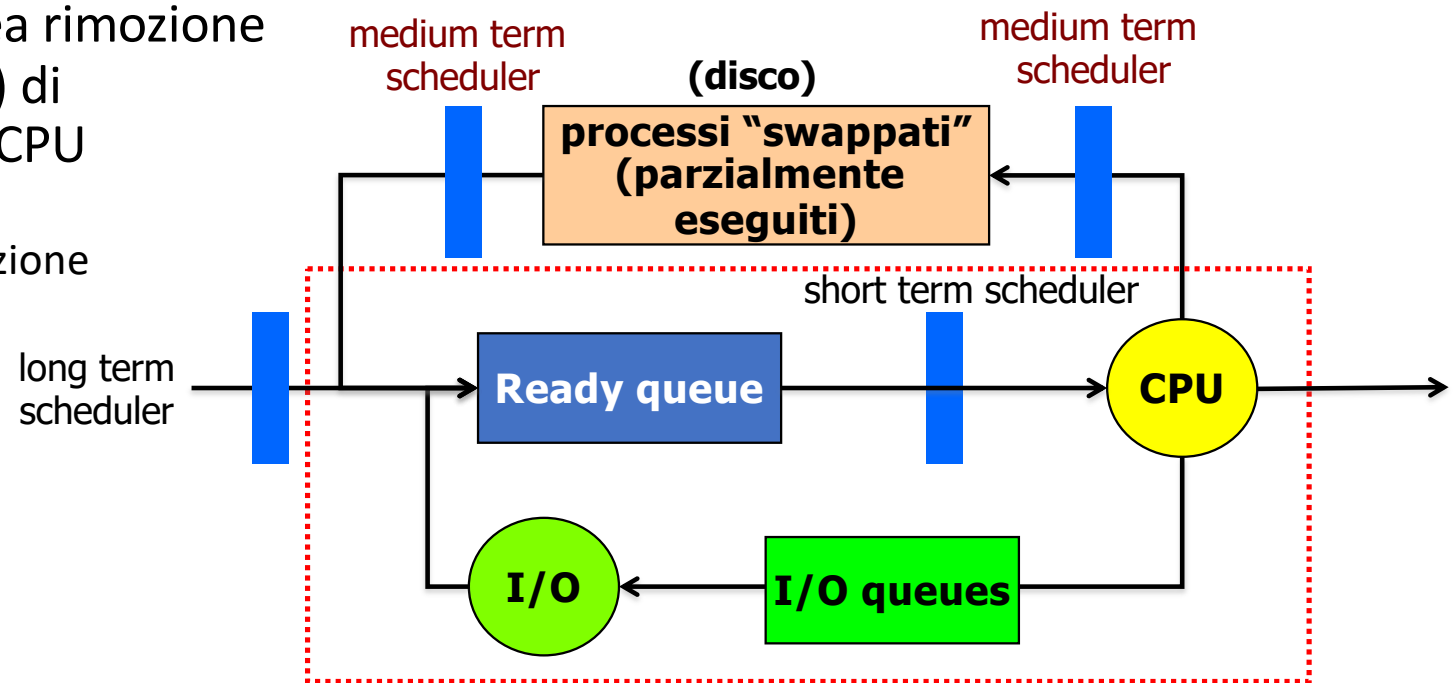
- $O(ms)$   $\Rightarrow$  deve essere veloce
  - Es.: 100 ms per processo, 10 ms per scheduling
  - $10/(110) = 9\%$  del tempo di CPU sprecato per scheduling

Scheduler a lungo termine è invocato più raramente

- $O(ms)$   $\Rightarrow$  può essere lento
- Controlla il grado di multiprogrammazione e il mix di processi
  - I/O-bound
    - molto I/O, molti brevi burst di CPU
  - CPU-bound
    - molti calcoli, pochi lunghi burst di CPU
- Può essere assente
  - usato principalmente in sistemi con risorse limitate

# Scheduling a medio termine

- S.O. con **memoria virtuale** prevedono un livello intermedio di scheduling (a medio termine)
  - Per la momentanea rimozione forzata (*swapping*) di un processo dalla CPU
    - Per ridurre grado multiprogrammazione



# Scheduling della CPU

# CPU Scheduler

Modulo del S.O. che seleziona un processo tra quelli in memoria pronti per l'esecuzione, e gli alloca la CPU

Data la frequenza di invocazione, è una parte critica del S.O.

- Necessità di algoritmi di scheduling

# Dispatcher

Modulo del S.O. che passa il controllo della CPU al processo scelto dallo scheduler

- Switch del contesto
- Passaggio alla modalità user
- Salto alla opportuna locazione nel programma per farlo ripartire

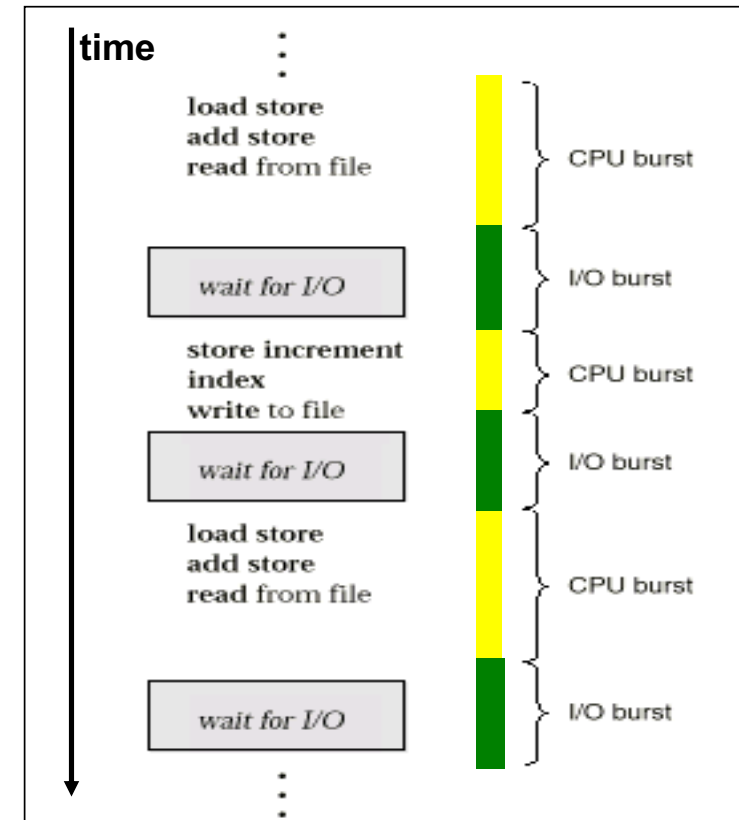
Latenza di dispatch

- Tempo necessario al dispatcher per fermare un processo e farne ripartire un altro
- Deve essere la più bassa possibile

# Modello astratto del sistema

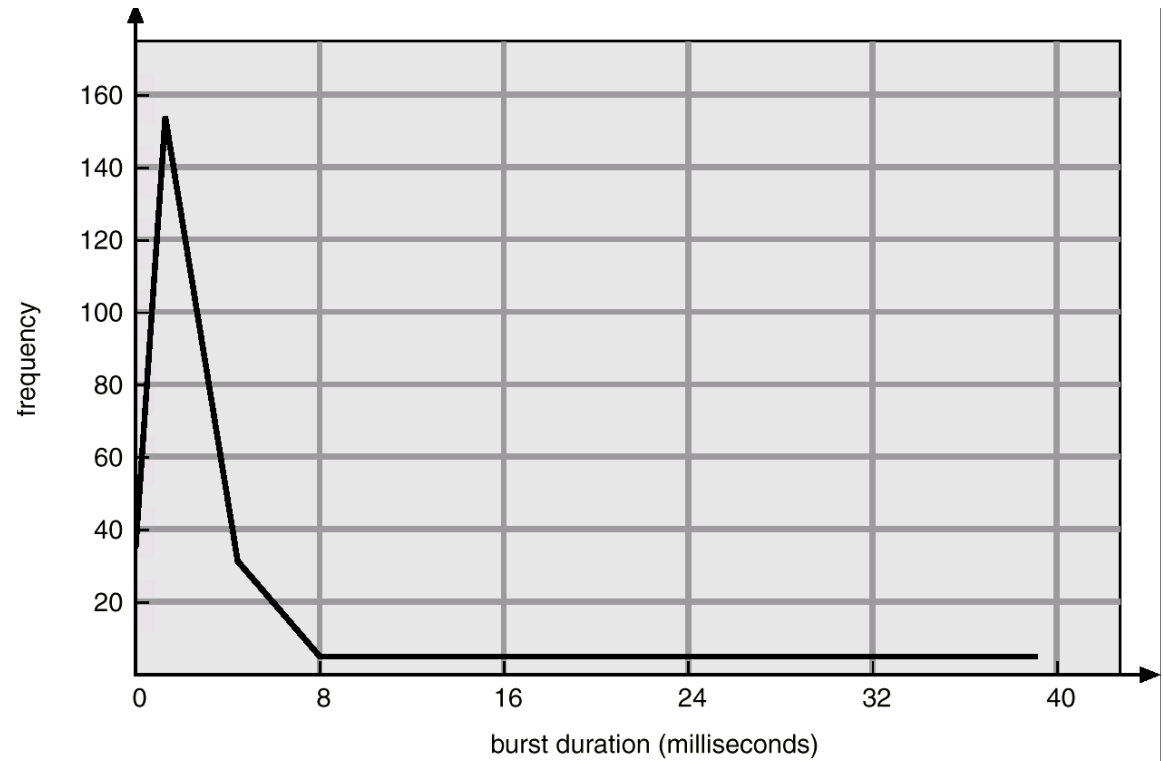
## Modello a cicli di burst CPU-I/O

- Alternanza CPU e I/O burst
  - Burst = sequenza
- L'esecuzione di un processo consiste nell'alternanza ciclica di un burst di CPU e di uno di I/O



# Distribuzione dei CPU burst

- Distribuzione esponenziale
  - Numerosi burst brevi
  - Pochi burst lunghi



# Prelazione (Preemption)

## Prelazione

- rilascio forzato della CPU

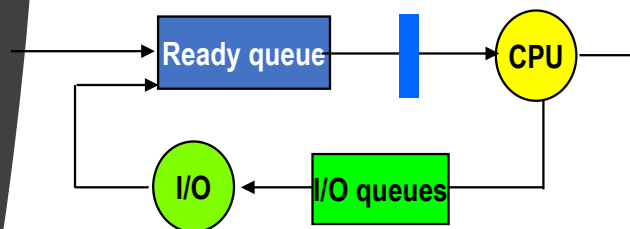
## Scheduling senza prelazione (non-preemptive)

- Processo che detiene la CPU non la rilascia fino al termine del burst

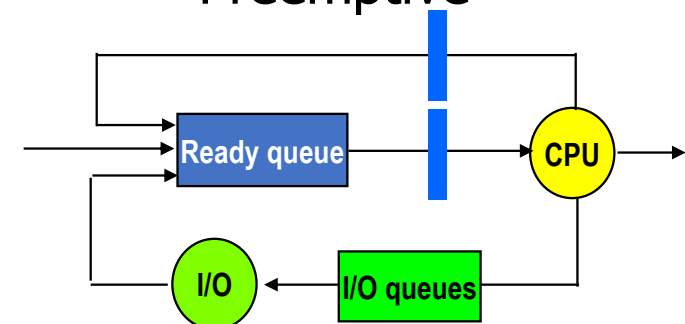
## Scheduling con prelazione (preemptive)

- Processo che detiene la CPU può essere forzato a rilasciarla prima del termine del burst

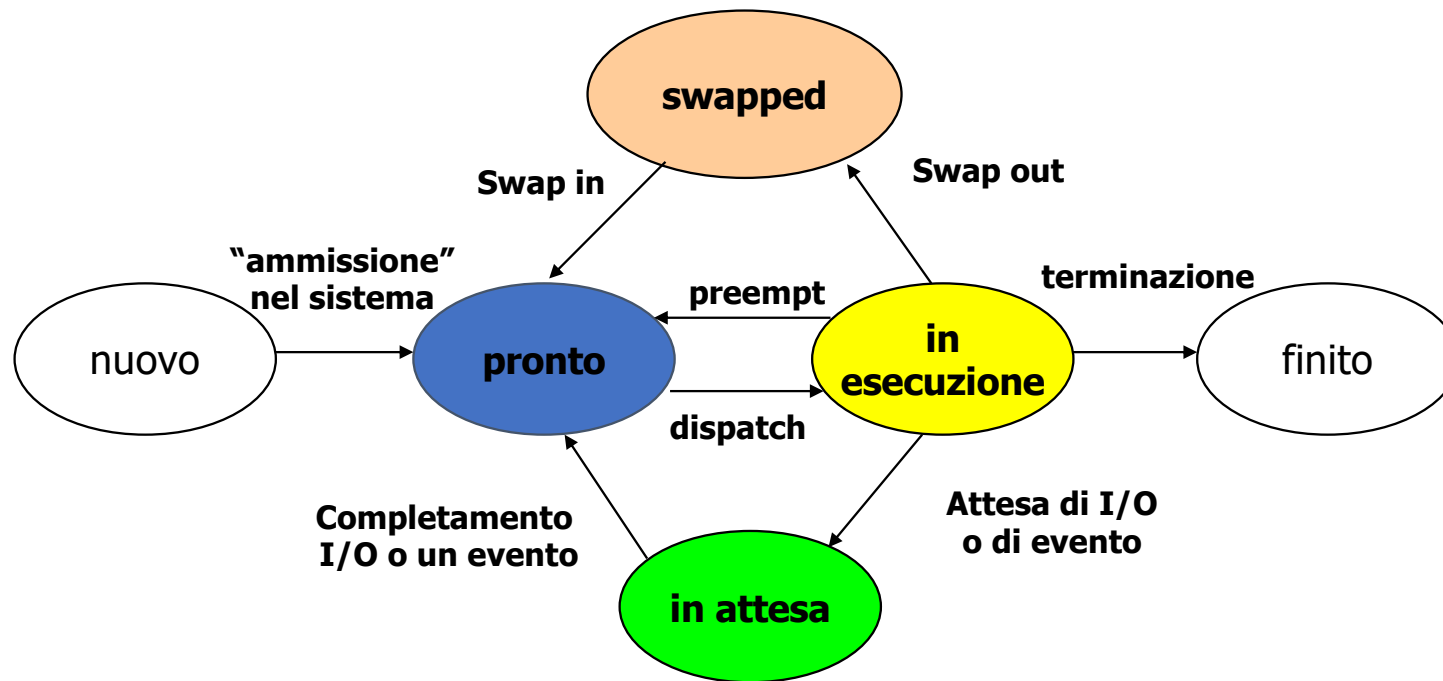
## Non-preemptive



## Preemptive



# Stati di un processo – Schema complessivo



# Metriche di Scheduling

## Utilizzo della CPU

- L'obiettivo è tenere CPU occupata più possibile

## Throughput

- Numero di processi completati per unità di tempo

## Tempo di attesa (*waiting time*, $t_w$ )

- Quantità totale di tempo spesa da un processo nella coda di attesa
- E' influenzato dall'algoritmo di scheduling

# Metriche di Scheduling

## Tempo di completamento (*turnaround*, $t_t$ )

- Tempo necessario ad eseguire un particolare processo dal momento della sottomissione al momento del completamento

## Tempo di risposta (*response time*, $t_r$ )

- Tempo trascorso da quando una richiesta è stata sottoposta al sistema fino alla prima risposta del sistema stesso

# Criteri di ottimizzazione

## Massimizzare

- utilizzo della CPU
- throughput

## Minimizzare

- tempo di turnaround
- tempo di attesa
- tempo di risposta

# Algoritmi di scheduling

# First-Come, First-Served (FCFS)

## Concetto

- Coda dei processi = coda FIFO
- Primo processo arrivato è il primo ad essere servito

## Motivazione

- Implementazione semplice

# FCFS (Esempio)

Processo	Tempo di arrivo	CPU burst
P1	0	24
P2	2	3
P3	4	3

Processo	$T_r$	$T_w$	$T_t$
P1	0	0	24
P2	22	22	25
P3	23	23	26

Tempo	0	24	27	30
P1				
P2				
P3				

- Tempo di attesa medio
  - $Tw\_medio = (0+22+23)/3 = 15$

# FCFS (Esempio)

Processo	Tempo di arrivo	CPU burst
P1	4	24
P2	0	3
P3	2	3

Processo	$T_r$	$T_w$	$T_t$
P1	2	2	26
P2	0	0	3
P3	1	1	4

Tempo	0	3	6	30
P1				
P2				
P3				

- Tempo di attesa medio
  - $T_{w\_medio} = (2+0+1)/3 = 1$  (molto meglio!)

# FCFS - Svantaggio

- Effetto convoglio
  - Processi brevi si accodano ai processi lunghi precedentemente arrivati

## Shortest-Job-First (SJF)

Associa ad ogni processo la lunghezza del prossimo burst di CPU

Il processo con il burst di CPU più breve viene selezionato per l'esecuzione

# Shortest-Job-First (SJF)

Due schemi:

- Non preemptive
- Preemptive
  - Se arriva un nuovo processo con un burst di CPU più breve del tempo che rimane da eseguire al processo in esecuzione, quest'ultimo viene rimosso dalla CPU per fare spazio a quello appena arrivato
  - In questo caso l'algoritmo si chiama Shortest-Remaining-Time-First (SRTF)

SJF è ottimo: minimo tempo medio di attesa

# SJF non preemptive (Esempio)

Processo	Tempo di arrivo	CPU burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Processo	$T_r$	$T_w$	$T_t$
P1	0	0	7
P2	6	6	10
P3	3	3	4
P4	7	7	11

Tempo	0	7	8	12	16
P1					
P2					
P3					
P4					

# SJF preemptive (Esempio)

Processo	Tempo di arrivo	CPU burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Processo	$T_r$	$T_w$	$T_t$
P1	0	9	16
P2	0	1	5
P3	0	0	1
P4	2	2	6

Tempo	0	2	4	5	7	11	16
P1							
P2							
P3							
P4							

# Scheduling a priorità

Viene associata una priorità a ogni processo

CPU allocata al processo con priorità più alta

Opzioni:

- Preemptive
- Non-preemptive

Linux: comando *nice* per cambiare la priorità

Esempio:

- SJF è uno scheduling a priorità  
(priorità =  $1/\text{lunghezza del burst successivo}$ )

# Scheduling a priorità

- Politiche di assegnamento della priorità:
  - Interne al S.O.
    - Limiti di tempo
    - Requisiti di memoria
    - N° file aperti
    - ...
  - Esterne al S.O.
    - Importanza del processo
    - Soldi pagati per l'utilizzo del computer
    - Motivi politici
    - ...

# Scheduling a priorità (Esempio)

Proc.	T. di arrivo	Pr.	CPU burst
P1	1	3	10
P2	0	1	1
P3	2	3	2
P4	0	4	1
P5	1	2	5

Processo	$T_r$	$T_w$	$T_t$
P1	5	5	15
P2	0	0	1
P3	14	14	16
P4	18	18	19
P5	0	0	5

Tempo	0	1	6	16	18	19
P1						
P2						
P3						
P4						
P5						

# Scheduling a priorità

## Problema: *starvation*

- Processi a bassa priorità possono non essere mai eseguiti
  - Caso storico: chiusura IBM 7090 al MIT nel 1973
    - Un processo era in attesa dal 1967!

## Soluzione: invecchiamento (*aging*)

- Aumento della priorità col passare del tempo

# Highest Response Ratio Next (HRRN)

- Algoritmo a priorità non-preemptive
  - Priorità  $R = (t_{\text{attesa}} + t_{\text{burst}}) / t_{\text{burst}} = 1 + t_{\text{attesa}} / t_{\text{burst}}$ 
    - maggiore per valori di R più alti
    - dipende anche dal tempo di attesa (dinamica)
    - ricalcolata:
      - al termine di un processo se nel frattempo ne sono arrivati altri
      - oppure, al termine di un processo
  - Favoriti i processi che:
    - completano in poco tempo (come SJF)
    - hanno atteso molto
  - Supera il “favoritismo” di SJF verso job corti

# HRRN (Esempio)

Proc.	T. di arrivo	CPU burst
P1	1	10
P2	0	2
P3	2	2
P4	2	1
P5	1	5

Processo	$T_r$	$T_w$	$T_t$
P1	9	9	19
P2	0	0	2
P3	6	6	8
P4	5	5	6
P5	1	1	6

Calcolo priorità R (termine processo)				
Proc.	t=0	t=2	t=7	t=8
P1	-	$1+1/10$	$1+6/10$	$1+7/10$
P2	1	-	-	-
P3	-	$1+0/2$	$1+5/2$	$1+6/2$
P4	-	$1+0/1$	$1+5/1$	-
P5	-	$1+1/5$	-	-

Tempo	0	2	7	8	10	19
P1						
P2						
P3						
P4						
P5						

# Round Robin (RR)

## Scheduling basato su time-out

- A ogni processo viene assegnata una piccola parte (*quanto*) del tempo di CPU
  - Valori tipici: 10-100 millisecondi
- Al termine del quanto, il processo è prelazonato e messo nella ready queue
  - La ready queue è una coda circolare

## C $n$ processi nella coda e quanto $q$ :

- ogni processo ottiene  $1/n$  del tempo di CPU in blocchi di  $q$  unità di tempo alla volta
- nessun processo attende più di  $(n-1)q$  unità di tempo

# Round Robin (RR)

- Intrinsecamente preemptive
  - In pratica è un FCFS con prelazione
- Scelta del quanto
  - $q$  grande  $\Rightarrow$  FCFS
  - $q$  piccolo  $\Rightarrow$  Attenzione al context switch
    - $q$  troppo piccolo  $\Rightarrow$  troppo overhead per context switch
    - Meglio avere  $q \gg$  tempo di context switch
  - Valore ragionevole di  $q$ ?
    - Fare in modo che 80% dei burst di CPU siano  $< q$
- Prestazioni
  - Tempo di turnaround maggiore/uguale di SJF
  - Tempo di risposta minore/uguale di SJF

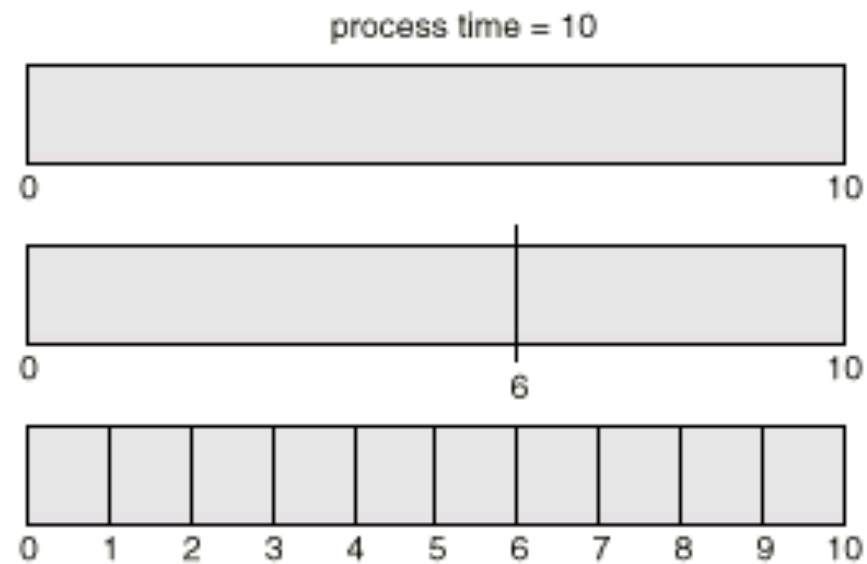
# RR (Esempio)

Proc. ( $q=2$ )	T. di arrivo	CPU burst
P1	0	5
P2	0	1
P3	0	7
P4	0	2

Processo	$T_r$	$T_w$	$T_t$
P1	0	7	12
P2	2	2	3
P3	3	8	15
P4	5	5	7

Tempo ( $q=2$ )	0	2	3	5	7	9	11	12	14	15
In esecuzione	P1	P2	P3	P4	P1	P3	P1	P3	P3	
Nella ready queue	P2	P3	P4	P1	P3	P1	P3			
	P3	P4	P1	P3						
	P4	P1								

Relazione tra  
“quanto” e  
“context  
switch”

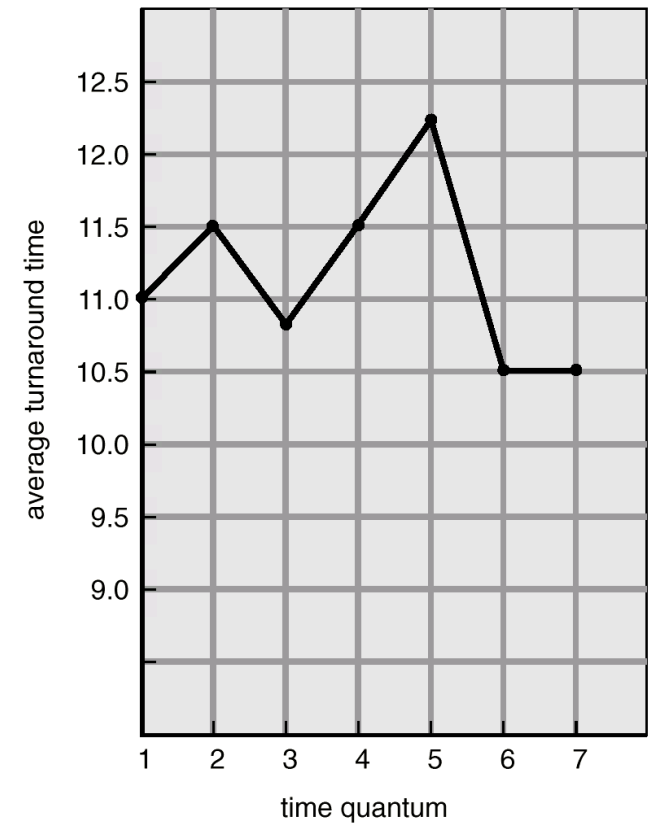


quantum	context switches
12	0
6	1
1	9

## Relazione tra “quanto” e “turnaround”

- $T_{t\_medio} (q=1) = 11$
- Turnaround non decresce sempre all'aumentare del quanto

Pr.	T. di arrivo	CPU burst	$T_t$ ( $q=1$ )
P1	0	6	15
P2	0	3	9
P3	0	1	3
P4	0	7	17



# Code multilivello

Classe di algoritmi in cui la ready queue è partizionata in più code

- Esempio
  - Una coda per job in foreground (interattivi)
  - Una coda per job in background (batch)
  - ...

Ogni coda ha il suo algoritmo di scheduling

- Esempio
  - Job in foreground gestiti con RR
  - Job in background gestiti con FCFS

E' un meccanismo più generale, ma anche più complesso

# Code multilivello

Necessario “scheduling  
tra le code”

## Scheduling a priorità fissa

- Es.: servire prima tutti i job di sistema, poi quelli in foreground, poi quelli in background
- Possibilità di starvation per code a priorità bassa

## Scheduling basato su time slice

- Ogni coda ottiene un quanto del tempo di CPU che può usare per schedulare i suoi processi
- Esempio, 80% per job di foreground con RR, 20% per job di background con FCFS



The diagram illustrates a multi-level code execution model. On the left, a dark gray vertical bar contains a black circle with the text "Code multilivello". To the right of this bar, five horizontal bars represent different process types, arranged from highest to lowest priority. Each bar has a small arrow pointing to its start and a larger arrow pointing to its end. The bars are labeled: "system processes", "interactive processes", "interactive editing processes", "batch processes", and "student processes". The priority is indicated by the text "highest priority" at the top and "lowest priority" at the bottom.

# Code multilivello

highest priority

system processes

interactive processes

interactive editing processes

batch processes

student processes

lowest priority

# Code multilivello con feedback

## Code multilivello classiche

- Un processo viene allocato definitivamente ad una coda

## Code multilivello con feedback (adattative)

- Un processo può spostarsi da una coda all'altra a seconda delle sue caratteristiche
- Usato anche per implementare l'*aging*

## Parametri dello scheduler

- numero delle code
- algoritmi per ogni coda
- criteri per la promozione/degradazione di un processo
- criteri per definire la coda di ingresso di un processo

# Code multilivello con feedback (Esempio)

3 code:

- Coda  $Q_0$ : RR con quanto 8 ms
- Coda  $Q_1$ : RR con quanto 16 ms
- Coda  $Q_2$ : FCFS

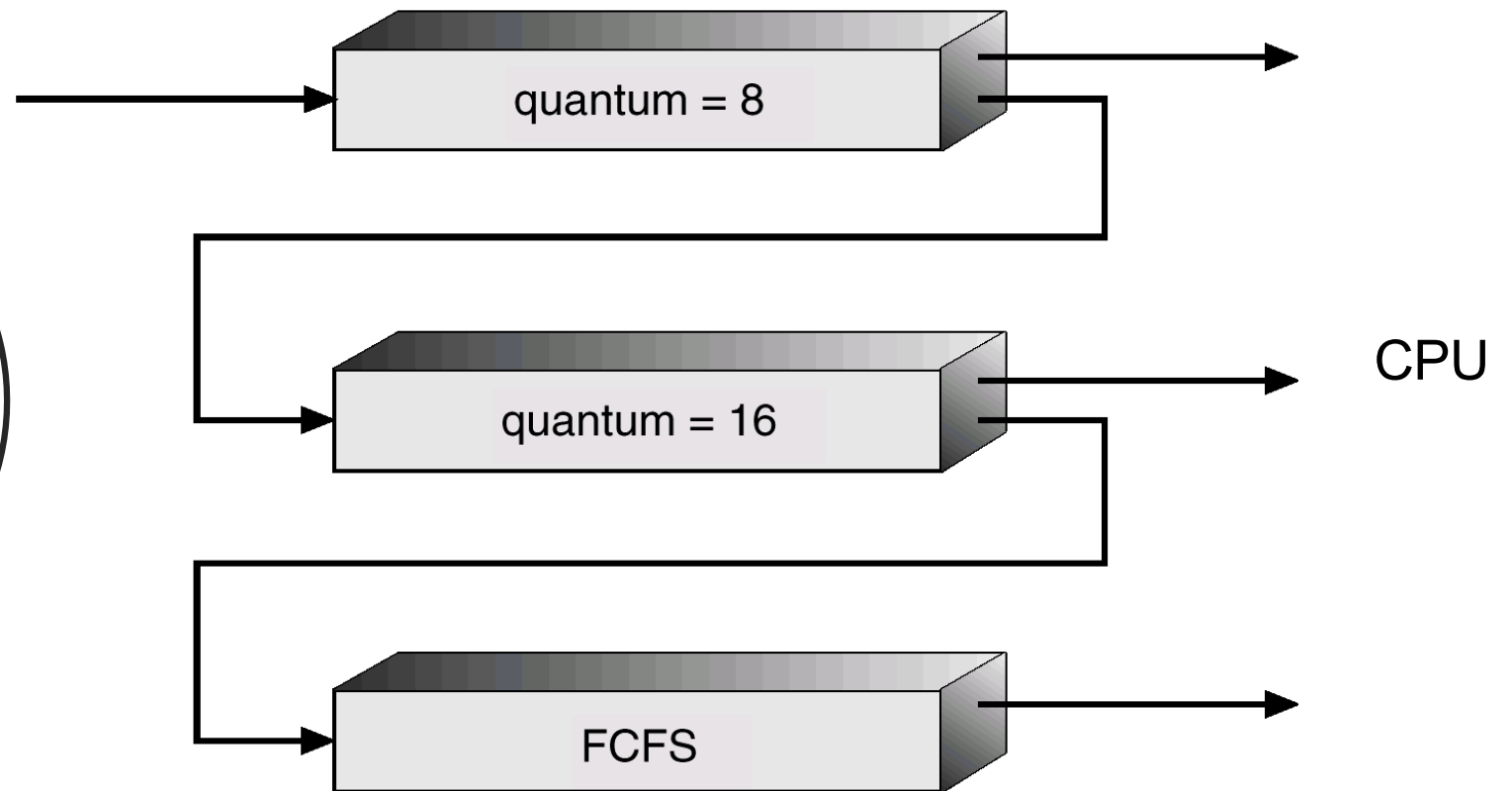
CPU serve nell'ordine  $Q_0, Q_1, Q_2$

- Processi in  $Q_j$  serviti sse  $Q_i$  vuota  $\forall i < j$

Funzionamento:

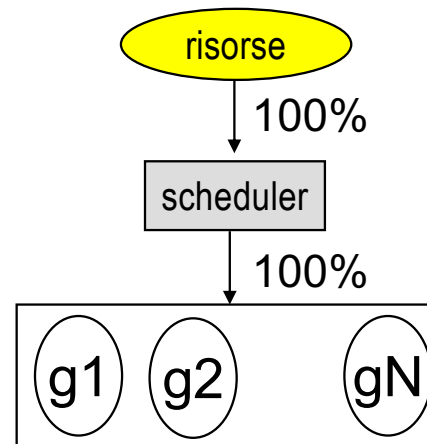
- Un job "nuovo" entra in  $Q_0$ . Quando ottiene la CPU, riceve 8 ms di quanto. Se non finisce entro il quanto, viene prelazionato e degradato alla coda  $Q_1$
- Se  $Q_0$  è vuota, si seleziona un job di  $Q_1$  che riceve 16 ms di quanto. Se non finisce viene prelazionato e messo in  $Q_2$
- Se  $Q_0$  e  $Q_1$  sono vuote, viene selezionato un job in  $Q_2$  con FCFS

Code  
multilivello  
con feedback  
(Esempio)



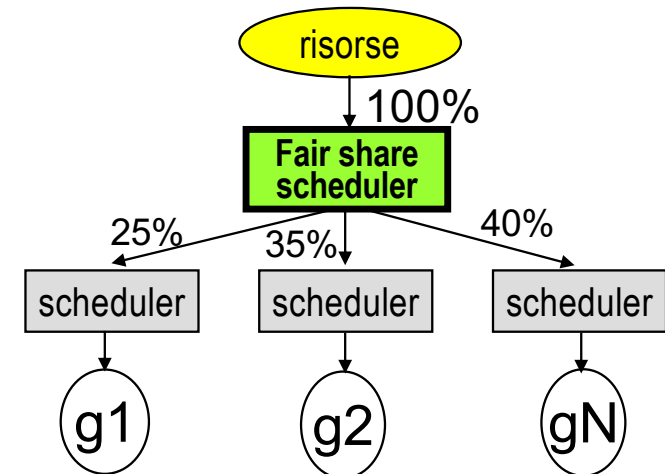
# Scheduling fair share

Le politiche di scheduling precedenti sono orientate al processo, ma un'applicazione può essere composta da più processi



Fair share cerca di fornire equità alle applicazioni (e quindi agli utenti) e non ai singoli processi

Le risorse vengono suddivise NON tra la totalità dei processi ma tra gruppi di processi



# Contesto reale

Obiettivo: minimizzare la complessità

Gli algoritmi reali usano la prelazione e sono spesso basati su RR

Esempio: CPU scheduling in Solaris (Unix di Sun)

- Basato su priorità con *aging*
- Priorità = priorità base + priorità corrente
  - Priorità base =  $[-20 \dots +20]$  (-20=max, +20=min)
  - Priorità corrente =  $0.1 * CPU(5*n)$ 
    - $CPU(t)$  = utilizzo della CPU negli ultimi t secondi
    - n = numero medio di processi pronti all'esecuzione nell'ultimo secondo
  - Concetto: scheduler "dimentica" il 90% dell'utilizzo di CPU degli ultimi 5n secondi
- Idea: favorire processi che hanno usato "poco" la CPU

# Valutazione degli algoritmi

Modello  
deterministico

Modello a reti di  
code

Simulazione

Implementazione

# Modello deterministico (analitico)

Basata sull'algoritmo e su un preciso carico di lavoro

- Ciò che abbiamo fatto negli esempi precedenti!

Definisce le prestazioni di ogni algoritmo per "quello" specifico carico

- Risposte applicabili solo al caso considerato

Di solito usato per illustrare gli algoritmi

Richiede conoscenze troppo specifiche sulla natura dei processi

# Modello a reti di code

Non esiste un preciso gruppo di processi sempre uguali per utilizzare il modello deterministico

Però è possibile determinare le distribuzioni di CPU burst e I/O burst

Il sistema di calcolo è descritto come una rete di server ognuno con la propria coda

Si usano formule matematiche che indicano:

- la probabilità che si verifichi un determinato CPU burst
- la distribuzione dei tempi di arrivo nel sistema dei processi

da cui è possibile ricavare utilizzo, throughput medio, tempi di attesa, ...

# Simulazione

Necessario  
programmare un  
modello del  
sistema

Si utilizzano dati  
statistici o reali

Abbastanza precisa  
ma costosa

# Implementazione

Unico modo assolutamente sicuro per  
valutare un algoritmo di scheduling

Codificarlo

Inserirlo nel S.O.

Vedere come funziona!

# Esercitazione

- Schedulare i processi indicati in tabella con le politiche:
  - FCFS
  - SJF senza prelazione
  - SJF con prelazione
  - RR con quanto = 4
  - RR con quanto = 1
  - HRRN

Proc.	T. di arrivo	CPU burst
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

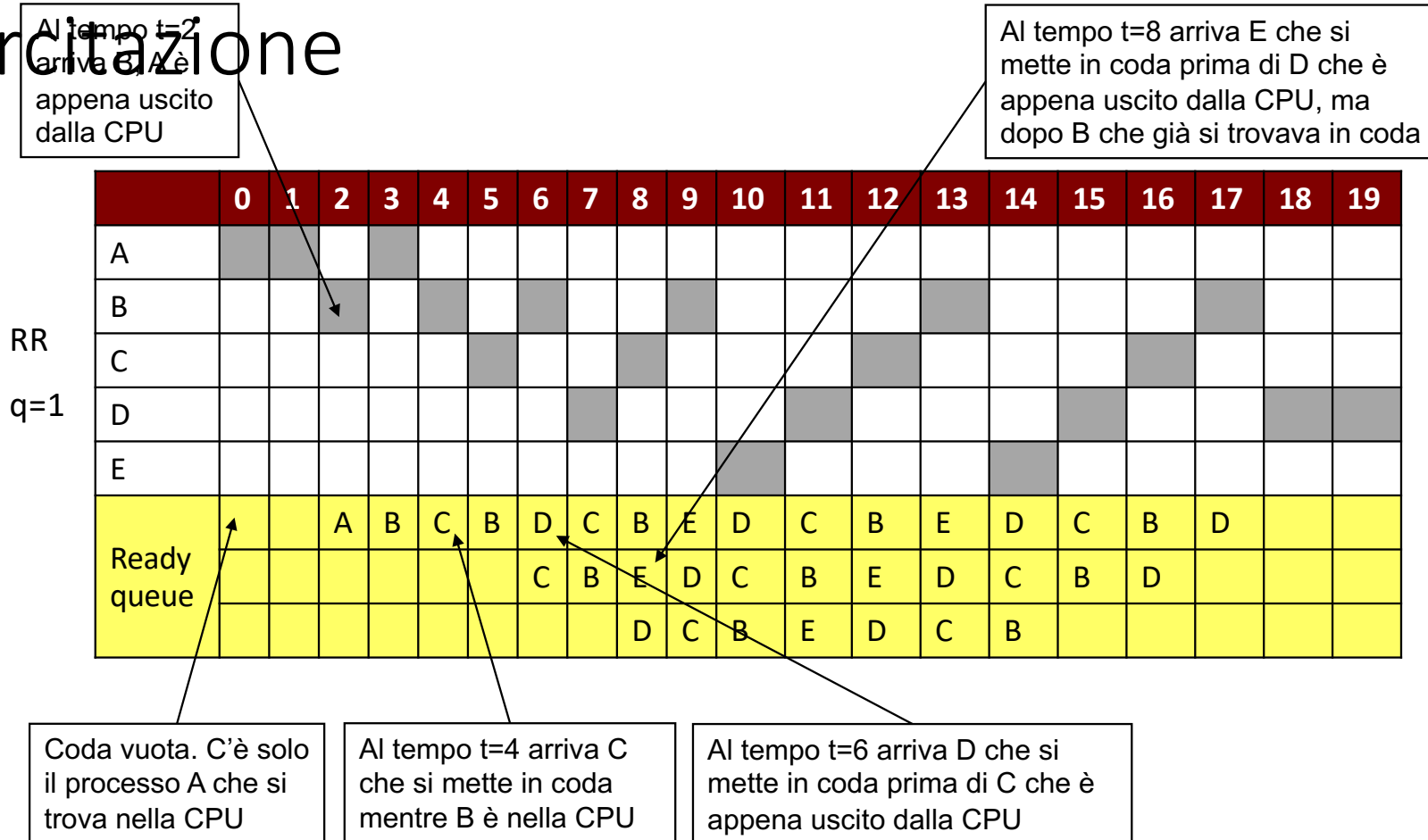
# Esercitazione

[illegible][illegible]

# Esercitazione

[illegible][illegible]

# Esercitazione



# Esercitazione

HRRN

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A																				
B																				
C																				
D																				
E																				

Calcolo priorità R				
Proc.	t=0	T=3	T=9	T=13
A	1	-	-	-
B	-	$1+1/6$	-	-
C	-	-	$1+5/4$	-
D	-	-	$1+3/5$	$1+7/5$
E	-	-	$1+1/2$	$1+5/2$

# Esercitazione

	A	B	C	D	E	Media
FCFS						
t attesa	0	1	5	7	10	4.6
t risposta	0	1	5	7	10	4.6
turnaround	3	7	9	12	12	8.6
SJF (senza prelazione)						
t attesa	0	1	7	9	1	3.6
t risposta	0	1	7	9	1	3.6
turnaround	3	7	11	14	3	7.6
SJF (con prelazione)						
t attesa	0	7	0	9	0	3.2
t risposta	0	1	0	9	0	2.0
turnaround	3	13	4	14	2	7.2

# Esercitazione

	A	B	C	D	E	Media
RR (q=1)						
t attesa	1	10	9	9	5	6.8
t risposta	0	0	1	1	2	0.8
turnaround	4	16	13	14	7	10.8
RR (q=4)						
t attesa	0	9	3	9	9	7.5
t risposta	0	1	3	5	9	3.6
turnaround	3	15	7	14	11	10.0
HRRN						
t attesa	0	1	5	9	5	4.0
t risposta	0	1	5	9	5	4.0
turnaround	3	7	9	14	7	8.0