

Scheduling

Livelli

Algoritmi

Introduzione

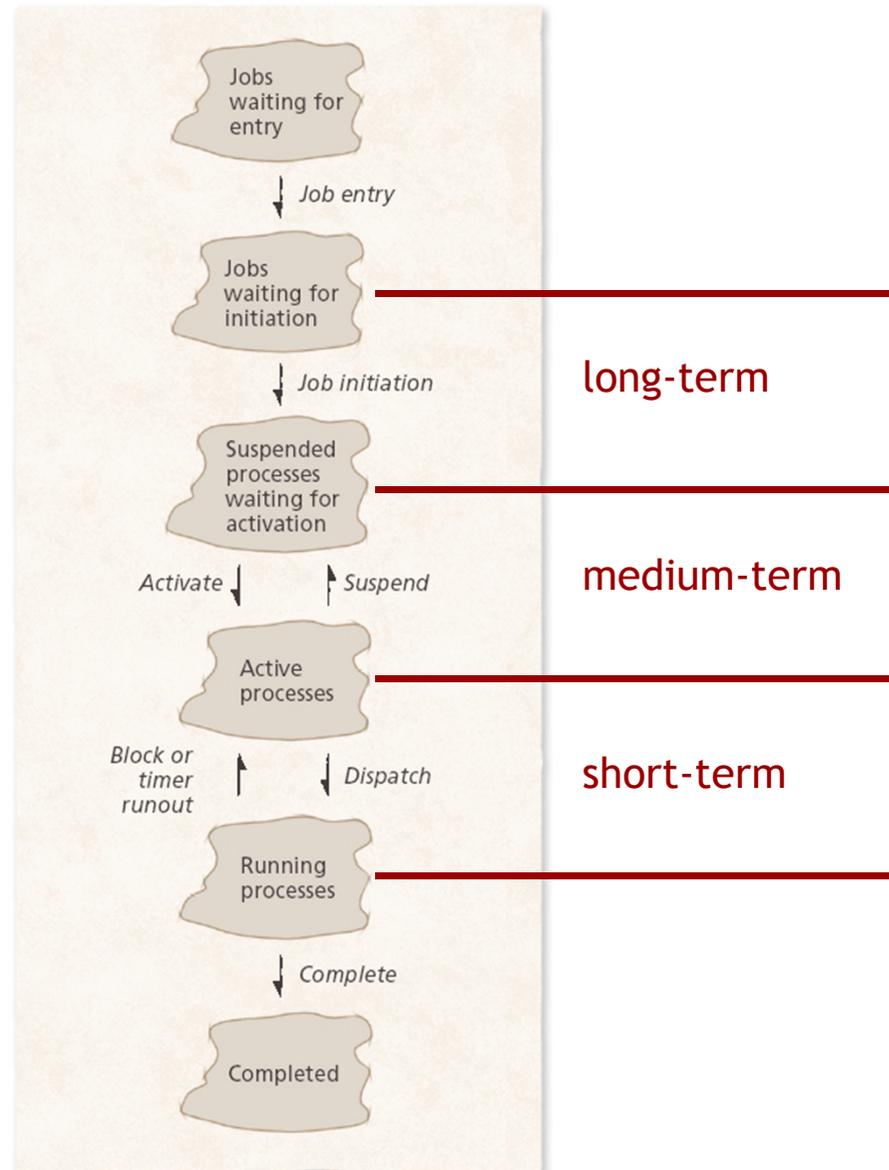
- **Lo scheduling**
 - Ha lo scopo di decidere quale processo eseguire in un dato istante
 - Si realizza mediante un componente specifico del sistema operativo
 - Lo scheduler

- **Possono esistere diversi scheduler con diversi obiettivi**
 - Massimizzare il throughput
 - Minimizzare la latenza
 - Evitare che un processo non venga mai eseguito
 - Completare un processo entro un tempo predefinito
 - Massimizzare la percentuale di utilizzo del processore

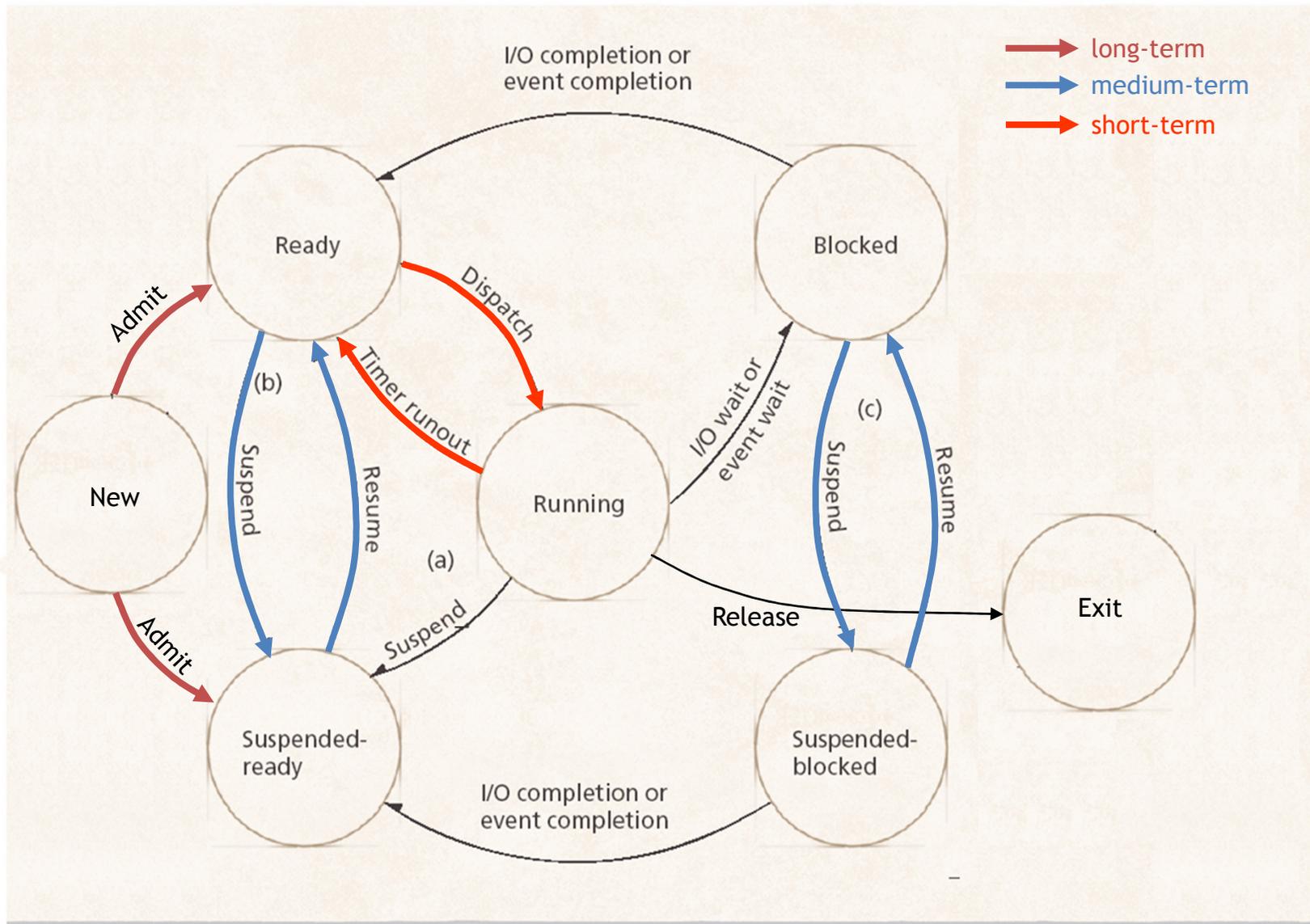
Livelli di scheduling

- **In un sistema operativo si hanno diversi livelli di scheduling**
- **Scheduling di lungo termine (long-term)**
 - Determina quali processi iniziare
 - New \rightarrow Ready, New \rightarrow Ready-Suspend
 - Controlla il numero di processi in un sistema
- **Scheduling di medio termine (medium-term)**
 - Determina quali processi possono competere per il processore
 - Ready \leftrightarrow Ready-Suspend, Blocked \leftrightarrow Blocked-Suspend
 - Reagisce alle fluttuazioni di breve durata nel carico del sistema
- **Scheduling di breve termine (short-term)**
 - Assegna il processore ai processi
 - Ready \leftrightarrow Running
 - Gestisce le priorità

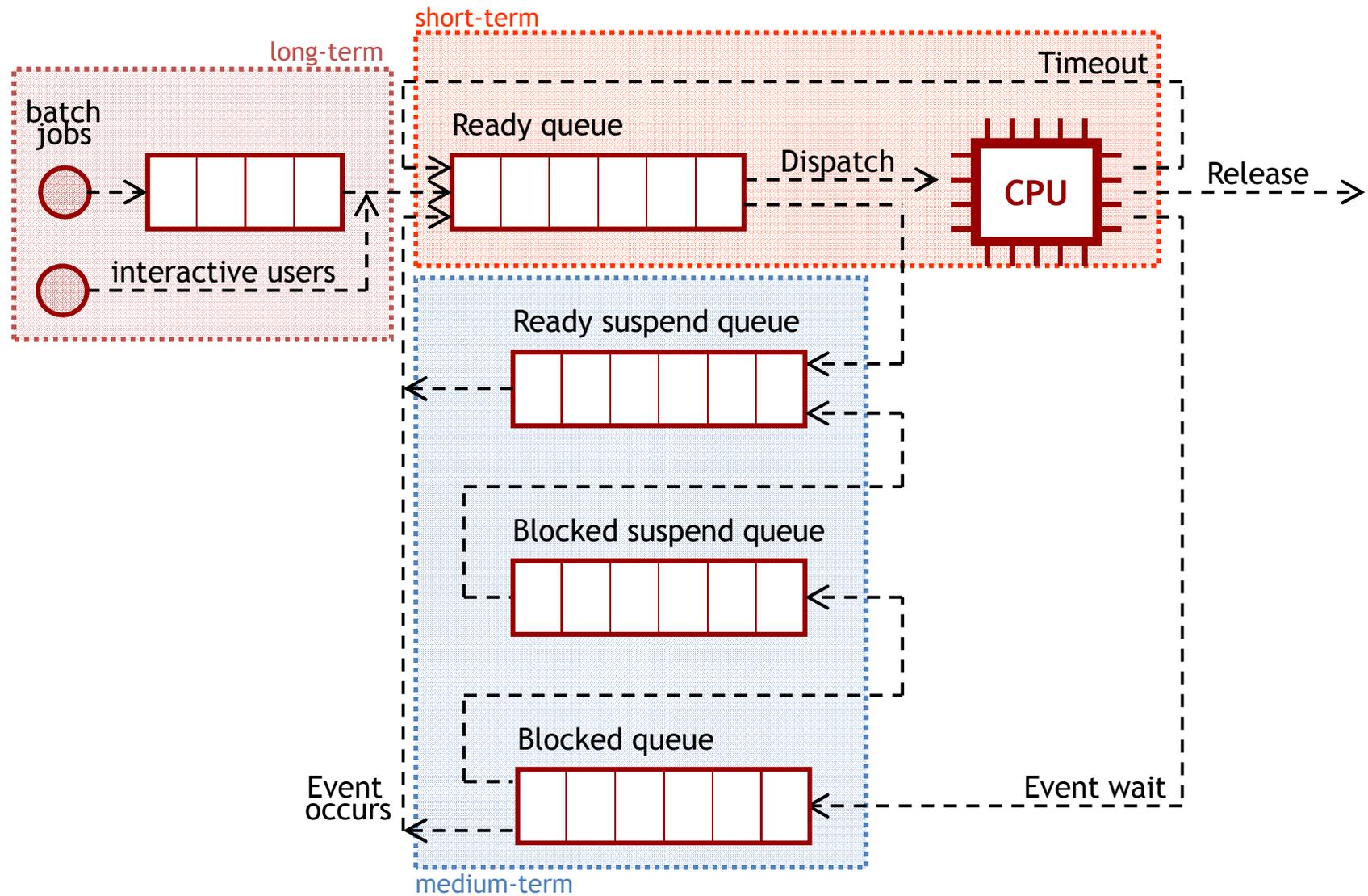
Livelli di scheduling



Livelli di scheduling



Livelli di scheduling



Dispatcher

- **Lo scheduler di breve periodo**
 - È detto dispatcher
 - È quello eseguito più frequentemente
- **Viene invocato quando si verifica un evento**
 - I/O e clock interrupt
 - Chiamata di sistema
 - Segnale
- **Criteri di scheduling di breve termine**
 - User-oriented
 - Minimizzare il “response time”
 - System-oriented
 - Utilizzo efficiente del processore

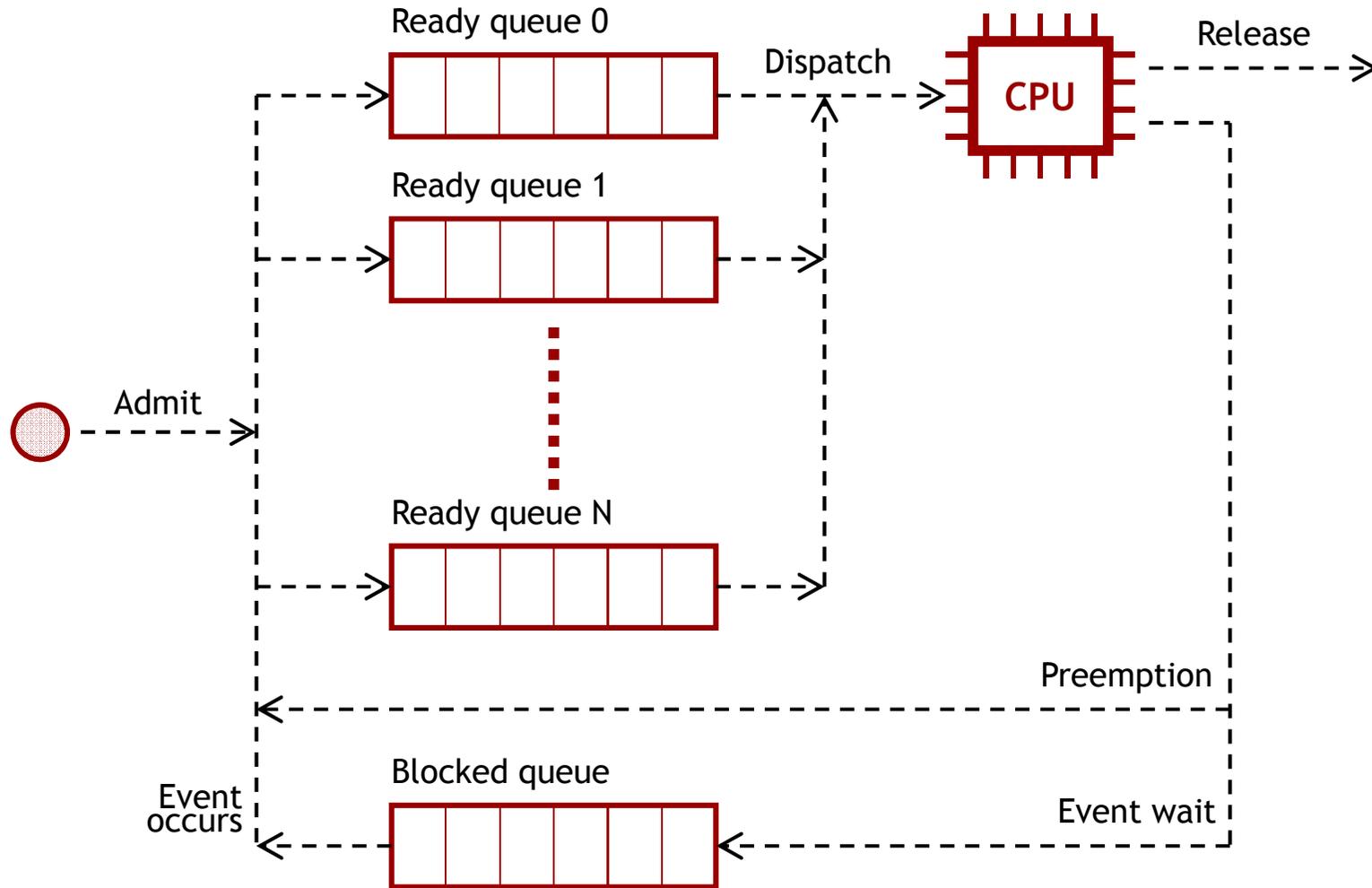
Preemption

- **Le politiche di scheduling e/o i singoli processi possono essere**
 - Preemptive o interrompibili
 - Nonpreemptive o non-interruptibili
- **Preemptive**
 - Possono essere rimossi dal processore cui sono assegnati
 - Possono migliorare il tempo di risposta
 - Sono molto utili in ambienti fortemente interattivi
 - I processi interrotti rimangono in memoria
- **Nonpreemptive**
 - Sono eseguiti fino al completamento o fino a che non cedono il controllo del processore deliberatamente
 - Un processo può bloccare un altro per un tempo imprevedibile e potenzialmente infinito

Priorità

- **Uno scheduler prende decisioni in base alle priorità dei processi**
- **Priorità statica**
 - La priorità assegnata ad un processo alla creazione non cambia
 - Facile da implementare
 - Basso overhead
 - Non permette al sistema di reagire in modo efficiente ai cambiamenti dell'ambiente, per esempio al carico
- **Priorità dinamica**
 - Reattiva ai cambiamenti dell'ambiente
 - Consente una buona interattività
 - Richiede un maggiore overhead rispetto alla soluzione statica
 - Tale aumento è giustificato dalla maggiore responsività

Priorità



Algoritmi

- **Obiettivi dello scheduling**

- Massimizzare il throughput
- Massimizzare il numero di processi interattivi in grado di avere un tempo di risposta accettabile
- Minimizzare l'utilizzo delle risorse
- Evitare che un processo venga ritardato indefinitamente
- Definire e garantire il rispetto delle priorità
- Minimizzare l'overhead

- **Alcuni aspetti sono comuni a tutti gli algoritmi**

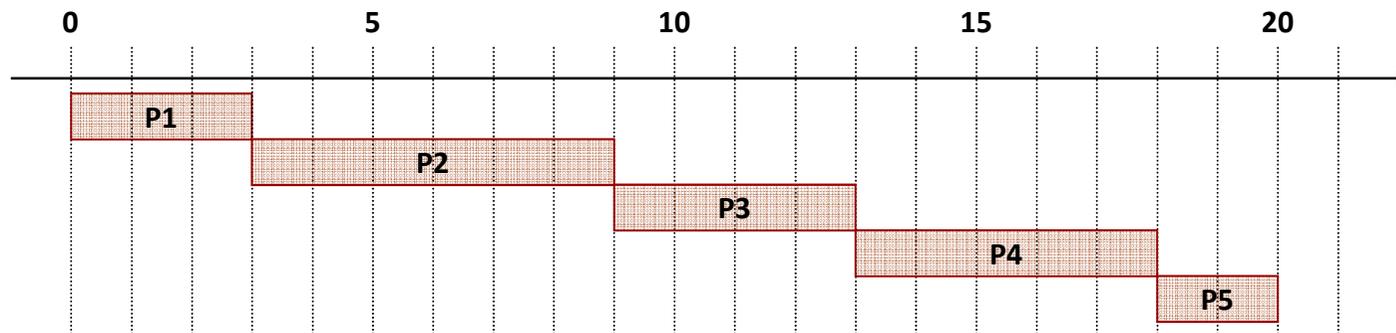
- Fairness
- Predicibilità
- Scalabilità

FIFO o FCFS – First-Come, First-Serve

Caratteristiche

- È lo schema di scheduling nonpreemptive più semplice possibile
- I processi vengono serviti (dispatched) in base all'arrival time

Processo	Arrival time	Service (run) time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2



- Tempi di attesa: P1=0, P2=1, P3=5, P4=7, P5=10
- Tempo medio di attesa: $(0+1+5+7+10)/5=4.6$

FIFO o FCFS – First-Come, First-Serve

▪ Funzionamento

- Ogni nuovo processo viene inserito nella coda dei processi ready
- Quando il processo corrente termina
 - Lo scheduler decide il prossimo processo da eseguire
 - Lo scheduler analizza la coda dei processi ready e sceglie il più vecchio

▪ Limitazioni

- Un processo molto breve potrebbe dover attendere un tempo anche molto lungo prima di essere servito
- Favorisce i processi CPU-bound
- I processi I/O-bound devono attendere la terminazione dei processi CPU-bound

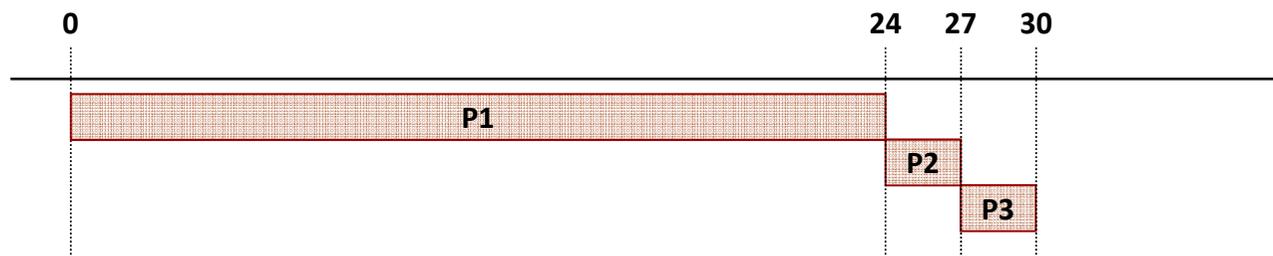
FIFO o FCFS – First-Come, First-Serve

Processi

Processo	Arrival time	Service (run) time
P1	0	24
P2	0	3
P3	0	3

Caso 1

- Ordine di arrivo P1, P2, P3



- Tempi di attesa: P1=0, P2=24, P3=27
- Tempo medio di attesa: $(0+24+27)/3=17$

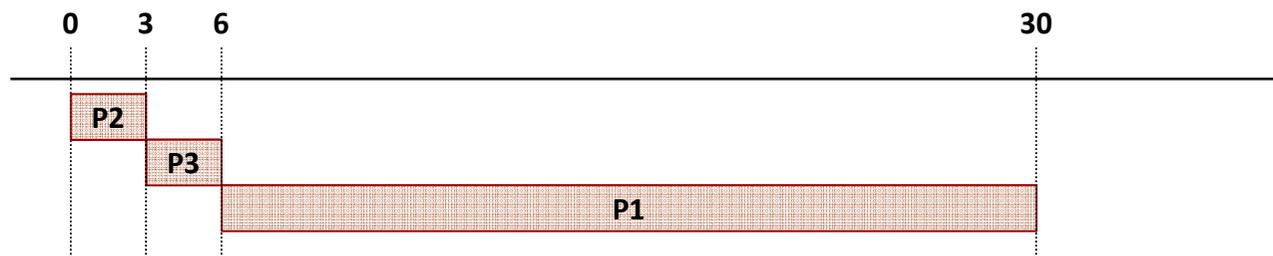
FIFO o FCFS – First-Come, First-Serve

Processi

Processo	Arrival time	Service (run) time
P1	0	24
P2	0	3
P3	0	3

Caso 2

- Ordine di arrivo P2, P3, P1



- Tempi di attesa: $P1=6$, $P2=0$, $P3=3$
- Tempo medio di attesa: $(6+0+3)/3=3$

RR – Round Robin

- **I processi**
 - Vengono serviti secondo un criterio FIFO
 - Possono essere eseguiti solamente per un tempo prefissato
 - Si parla di “quanto di tempo” o “timeslice”
 - Devono poter essere interrotti, quindi si tratta di uno schema preempive

- **Se un processo non termina entro il quanto di tempo assegnato**
 - Il sistema
 - Lo interrompe
 - Lo inserisce al fondo della coda dei processi ready
 - Lo scheduler sceglie il prossimo processo da eseguire

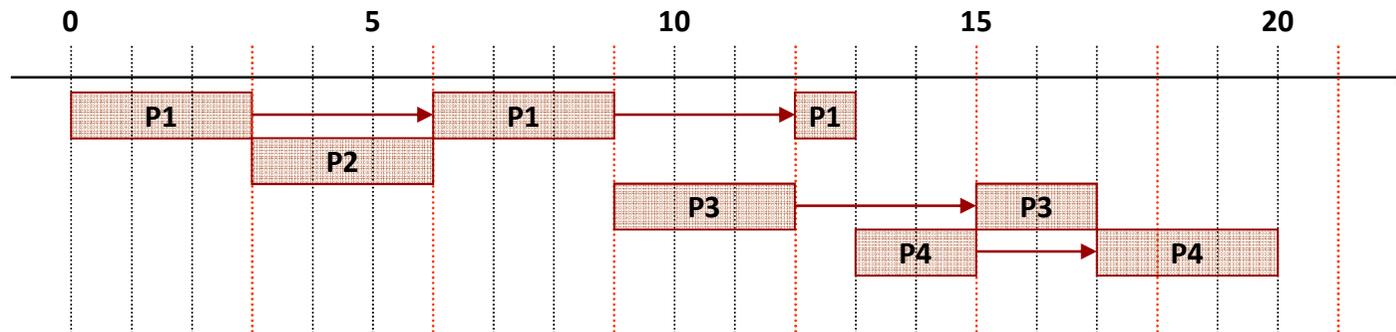
- **Come FIFO/FCFS anche lo scheduling RR in genere non è l’algoritmo di scheduling principale**

RR – Round Robin

- Processi

Processo	Arrival time	Service (run) time
P1	0	7
P2	2	3
P3	5	5
P4	10	5

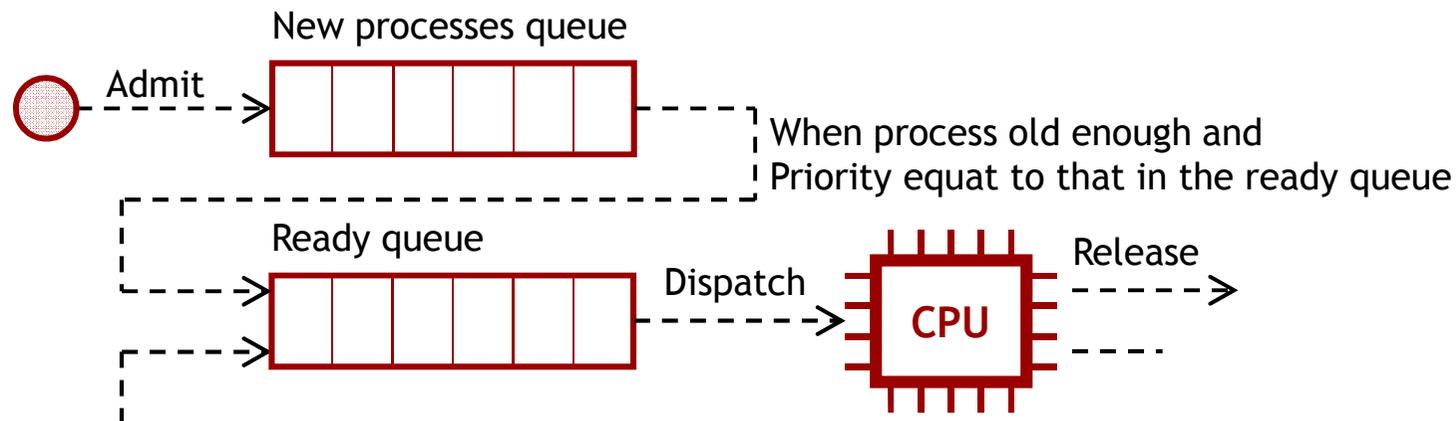
- Quanto di tempo paeri a 3 unità



- Tempi di attesa: $P1=0+3$, $P2=1$, $P3=4+3$, $P4=3+2$
- Tempo medio di attesa: $(3+1+7+5)/4=4$

SRR – Selfish Round Robin

- **Variante rispetto a RR**
 - Ogni processo ha una priorità
 - I processi “invecchiando” aumentano la loro priorità
- **Per gestire questo schema si utilizza una coda aggiuntiva**
 - Un processo entra nella coda dei processi nuovi
 - Li invecchia e la sua priorità cresce
 - Quando è uguale a quella dei processi ready
 - Entra nella coda dei processi ready
 - Diviene soggetto al meccanismo principale di scheduling (RR)



SRR – Selfish Round Robin

- **I processi hanno diverse velocità di invecchiamento**
 - Velocità pari a V_{new} nella coda dei processi nuovi
 - Velocità pari a V_{ready} nella coda dei processi ready
- **Affinchè l’algoritmo funzioni**
 - I processi nella coda dei processi nuovi devono poter entrare nella coda ready per poter essere soggetti a dispatch
 - Deve sussistere la relazione tra le velocità: $V_{new} \geq V_{ready}$
- **La relazione tra le due velocità influenza le caratteristiche e la natura stessa dell’algoritmo SRR**
 - Se V_{new} è poco maggiore di V_{ready} i processi resteranno in attesa per un periodo di tempo breve ma non trascurabile
 - Se V_{new} è molto maggiore di V_{ready} i processi resteranno in attesa per un periodo di tempo molto breve e la politica SRR degenera in RR
 - Se V_{new} è uguale a V_{ready} le due code sono identiche e la politica SRR degenera in FIFO

RR/SRR e Timeslice

- **Quale deve essere la durata del quanto di tempo?**
 - Breve o lunga?
 - Fissa o variabile?
 - Uguale per ogni processo e calcolata ad hoc?
- **Durata**
 - Se la durata cresce
 - Il quanto tende a divenire sufficiente al completamento di un processo
 - Le politiche round robin si trasformano in uno schema FIFO
 - Se la durata decresce troppo
 - Il tempo del context switch rende inefficiente la soluzione
- **È difficile stabile il valore corretto**
 - Linux: da 10ms a 200ms, tipicamente 100ms
 - Processi ad alta priorità ricevono un quanto più lungo
 - Windows XP: 20ms
 - Inferiore se il processo è in secondo piano nella GUI

SJF – Shortest Job First

- **Principio di base**
 - Associa ad ogni processo la lunghezza del prossimo CPU burst
 - Utilizza tali informazioni per schedulare il prossimo processo
 - Sceglie il proceso con il burst più breve
- **Si possono avere due schemi**
 - Nonpreemptive
 - Il processo non può essere interrotto
 - Si attende la fine del CPU-burst corrente
 - Preemptive
 - Un processo può essere interrotto
 - Se arriva un processo il cui tempo è minore del tempo rimanente del proceso corrente, questo viene interrotto e il nuovo processo schedulato
 - Questo schema è noto come SRTF o Shortest-Remaining-Time-First
- **SJF è un algoritmo ottimo**
 - Minimizza il tempo di attesa medio di un insieme di processi

SJF – Shortest Job First

- **Richiede la stima della durata del prossimo burst**
 - Meccanismo adattativo
 - Una stima diversa per ogni processo
 - Basso overhead computazionale
- **Si usa la seguente relazione basata sulla media esponenziale**

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

- **In cui:**

t_n	Durata del burst corrente al tempo n
τ_{n+1}	Durata predetta per il prossimo burst al tempo $n+1$
α	Coefficiente compreso tra 0 ed 1

SJF – Shortest Job First

- Il comportamento dell'algoritmo dipende fortemente da α
- **Con $\alpha = 0$**
 - La relazione diviene: $\tau_{n+1} = \tau_n$
 - La storia del sistema è ininfluente
- **Con $\alpha = 1$**
 - La relazione diviene: $\tau_{n+1} = t_n$
 - Conta solo la durata dell'ultimo burst
- **Con $0 \leq \alpha \leq 1$**
 - Espandendo la relazione si nota che il peso delle vecchie stime (τ_n) tende ad diminuire esponenzialmente nel tempo come $(1-\alpha)^n$

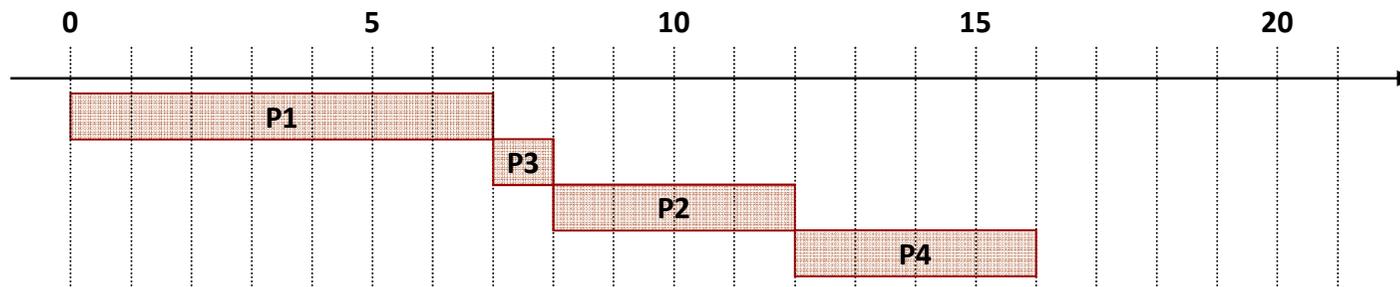
SJF – Shortest Job First

Processi

Processo	Arrival time	Burst time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Caso 1- Nonpreemptive

- La decisione di scheduling viene presa al termine di ogni burst



- Tempi di attesa: P1=0, P2=6, P3=3, P4=7
- Tempo medio di attesa: $(0+6+3+7)/4=4$

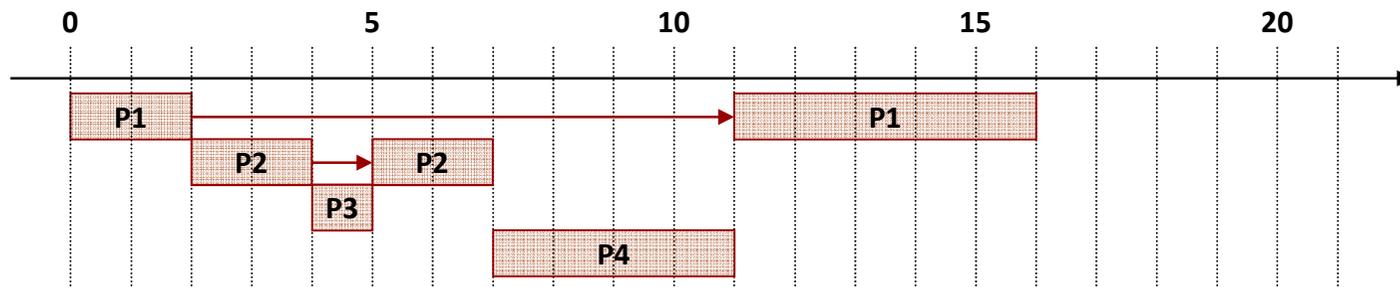
SJF – Shortest Job First

Processi

Processo	Arrival time	Burst time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Caso 1- Nonpreemptive

- La decisione di scheduling viene presa al termine di ogni burst



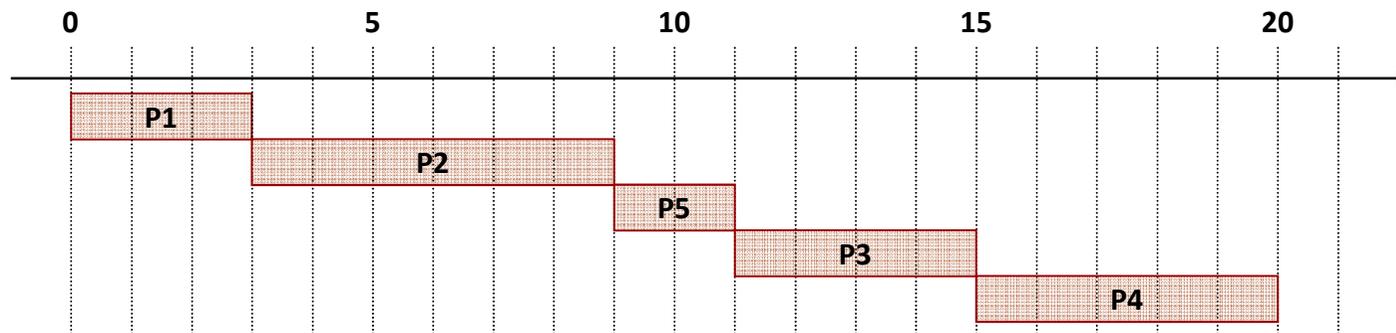
- Tempi di attesa: $P1=0+9$, $P2=0+1$, $P3=0$, $P4=2$
- Tempo medio di attesa: $(9+1+0+2)/4=3$

SPN – Shortest Process Next

Caratteristiche

- Si tratta di uno schema nonpreemptive
- Il processo ready con il runtime stimato minore viene selezionato

Processo	Arrival time	Service (run) time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2



- Tempi di attesa: P1=0, P2=1, P3=7, P4=9, P5=1
- Tempo medio di attesa: $(0+1+7+9+1)/5=3.6$

SPN – Shortest Process Next

- **Vantaggi**

- I processi brevi vengono serviti rapidamente

- **Problemi**

- I processi lunghi potrebbero essere differiti per un tempo anche molto elevato
 - Potenziale problema di starvation
 - Diminuisce la predicibilità per tali processi
- Dipende dalla correttezza della stima del tempo di esecuzione dei vari processi
 - Un errore potrebbe richiedere la terminazione di un processo

HRRN – Highest Response Ratio Next

- **Sviluppato per ovviare ad un limite di SJF/SPN**
 - Tendono a favorire molto i processi brevi rispetto a quelli lunghi
- **Si tratta di un algoritmo nonpreemptive**
 - Al momento del context switch calcola la priorità di ogni processo
 - Esegue il processo selezionato fino al completamento
- **La priorità è calcolata come:**

$$\frac{(T_{attesa} + T_{exec})}{T_{exec}}$$

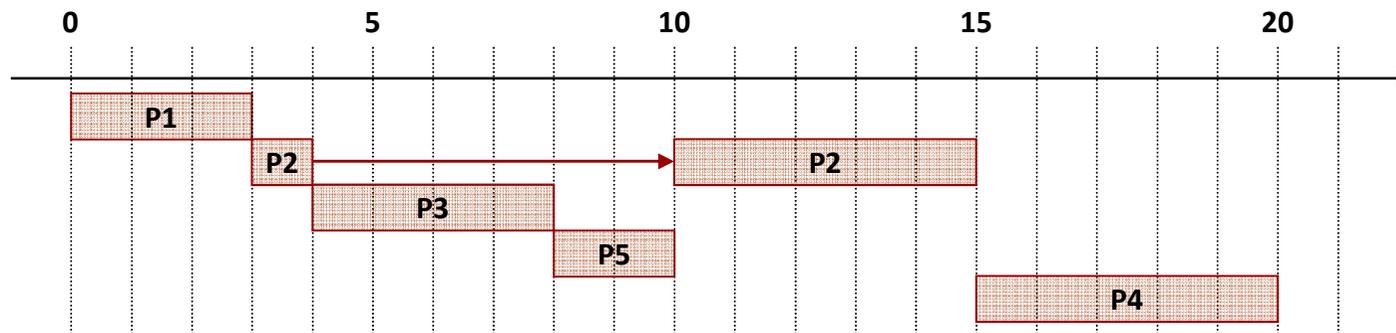
- **Considerazioni**
 - Processi brevi favoriti poiché il tempo di esecuzione appare a denominatore
 - Processi che attendono da molto favoriti poiché il tempo di attesa appare a numeratore

SRT – Shortest Remaining Time

Caratteristiche

- È la versione preemptive dell'algoritmo SPN
- Richiede una stima accurata dei tempi di esecuzione

Processo	Arrival time	Service (run) time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2



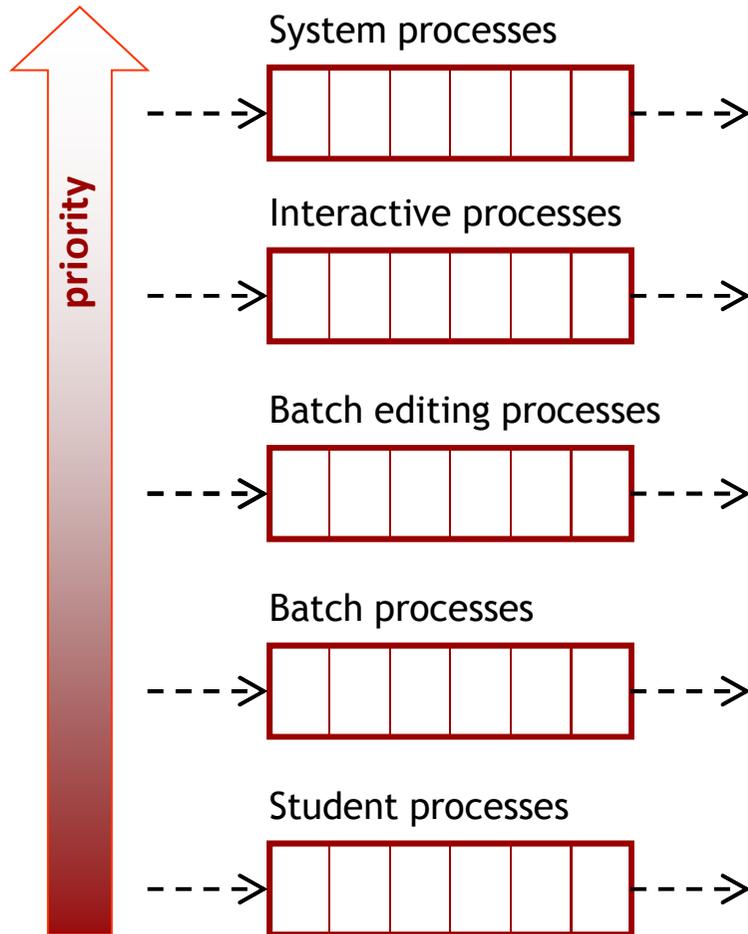
- Tempi di attesa: $P1=0$, $P2=1+6$, $P3=0$, $P4=9$, $P5=0$
- Tempo medio di attesa: $(0+7+0+9+0)/5=3.2$

Multilevel Queues

- **La coda dei processi ready viene suddivisa in**
 - Coda dei processi in foreground (interattivi)
 - Coda dei processi in background (batch)
- **Ogni coda dispone di un appropriato algoritmo di scheduling**
 - Foreground: RR
 - Background: FCFS
- **È necessario effettuare uno scheduling “tra” le code**
 - Fixed priority scheduling
 - Prima tutti i processi in foreground poi i processi in background
 - Possibilità di starvation
 - Time slice
 - Ogni coda dispone di una frazione fissata del tempo di CPU

Multilevel Queues

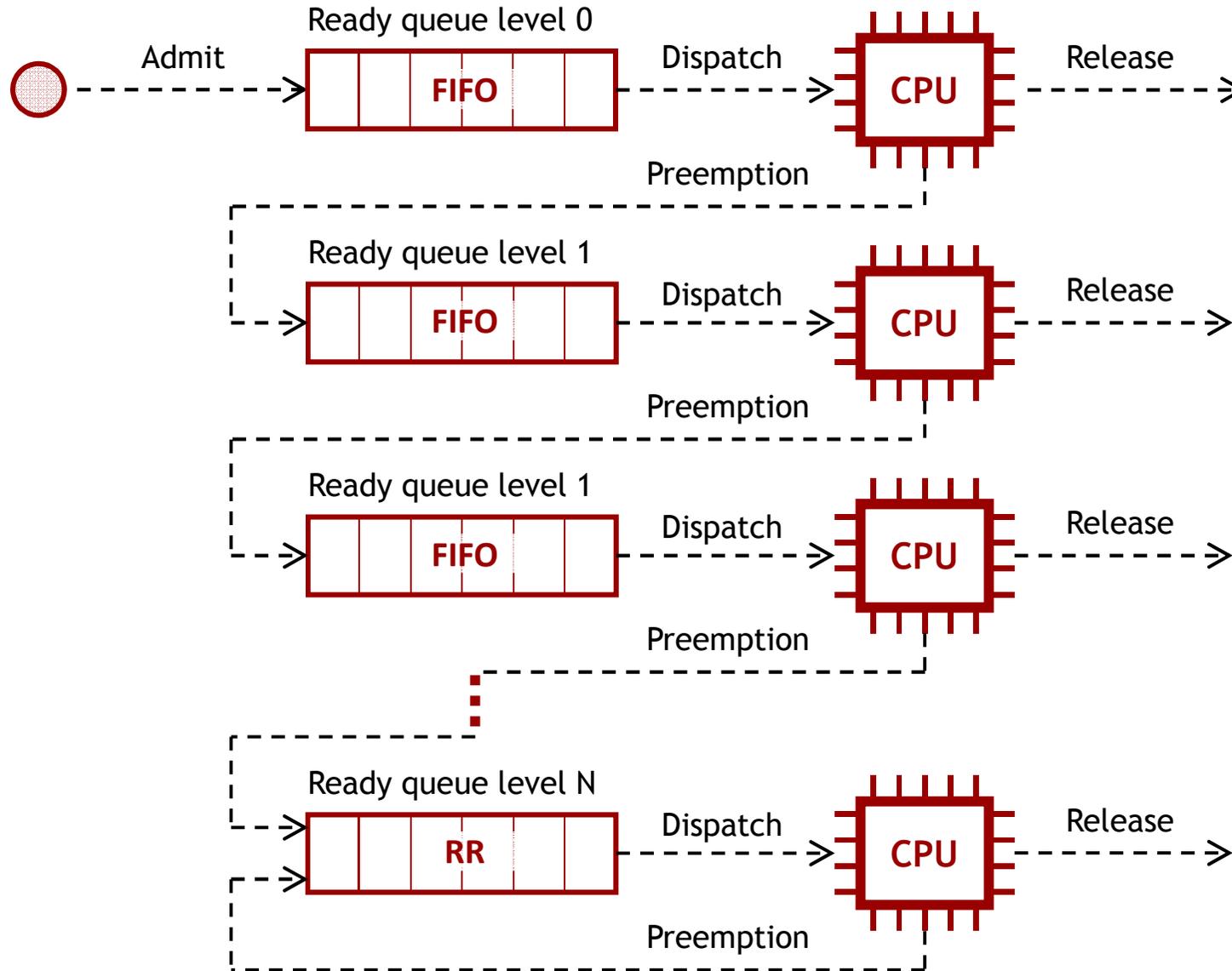
- Il meccanismo può essere facilmente generalizzato a più livelli



Multilevel Feedback Queues

- **Process diversi hanno esigenze differenti**
 - Processi brevi, I/O-bound e interattivi dovrebbero in genere avere la precedenza su processi lunghi, batch e CPU-bound
 - Il tipo di comportamento di un processo non è immediatamente noto allo scheduler ed è difficile prevederlo
- **Si ricorre ad un meccanismo a code multilivello a retroazione**
 - I nuovi processi
 - Entrano nella coda a più alto livello
 - Vengono eseguiti con priorità maggiore dei processi nelle altre code
 - I processi più lunghi scendono verso code a priorità minore
 - I processi brevi e I/O-bound mantengono una priorità maggiore
 - I processi lunghi sono eseguiti quando quelli brevi e I/O-bound terminano
 - Le politiche di scheduling delle diverse code possono essere
 - FIFO
 - RR

Multilevel Feedback Queues



Multilevel Feedback Queues

- **Uno scheduler basato su code multilivello con feedback è definito dai seguenti parametri principali**
 - Numero di code
 - Algoritmi di scheduling per ogni coda
 - Criteri di definizione del livello iniziale di un processo
 - Criteri di upgrade dei processi
 - Criteri di demote dei processi

Multilevel Feedback Queues

Code

- Q0: FIFO, preemptive, 8ms
- Q1: FIFO, preemptive, 16ms
- Q2: RR, preemptive, 32ms

Scheduling

- Nuovo processo entra in Q0
 - Riceve 8ms di CPU
 - Se non termina è interrotto e passa in Q1
- In Q1
 - Riceve 16ms di CPU
 - Se non termina è interrotto e passa in Q2
- In Q2
 - Viene completato in burst di 32ms

