ISLab Intelligent Systems Laboratory

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Sistema Operativo, gestione dei processi e scheduling

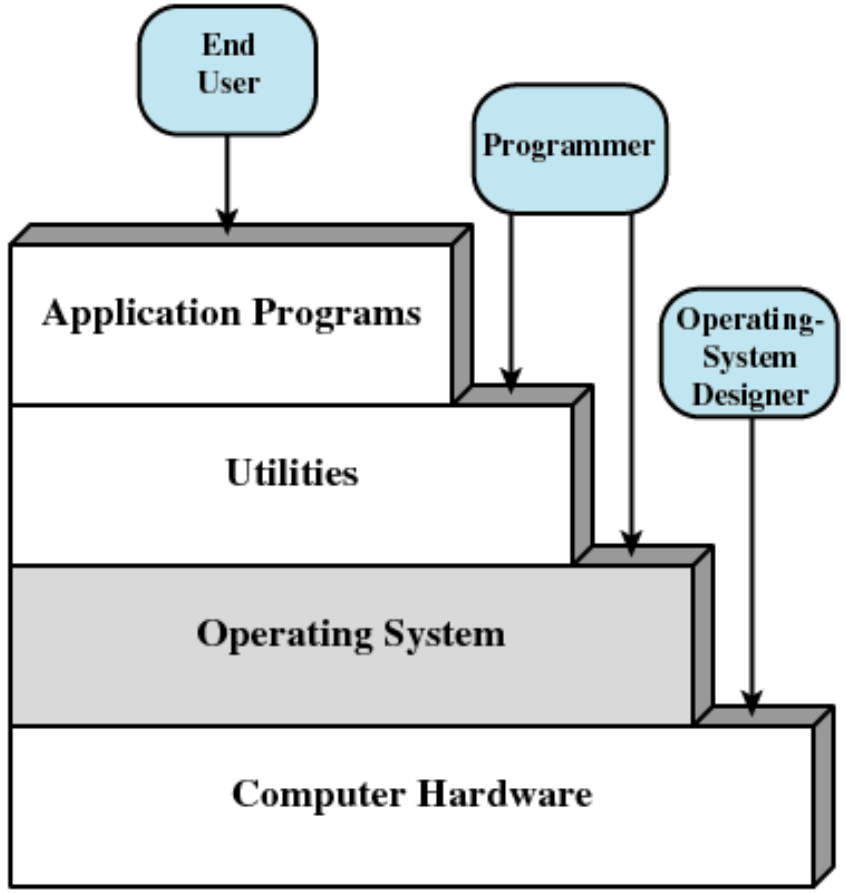
Prof. Giuseppe Pirlo | Prof. Donato Impedovo | Dott. Stefano Galantucci

Architettura degli Elaboratori e Sistemi Operativi @ Corso di Laurea in Informatica

Obiettivi di un Sistema Operativo

- **Convenienza** nell'uso del calcolatore rispetto ai potenziali utenti
- **Efficienza** nell'utilizzo del calcolatore e delle sue parti costitutive
- **Capacità** di evolversi rispetto a evoluzioni hardware, esigenze degli utenti e bug

Sistema Operativo come interfaccia



Il Sistema Operativo:

- Nasconde i dettagli hardware al programmatore
- Fornisce un'**interfaccia** per utilizzare il sistema

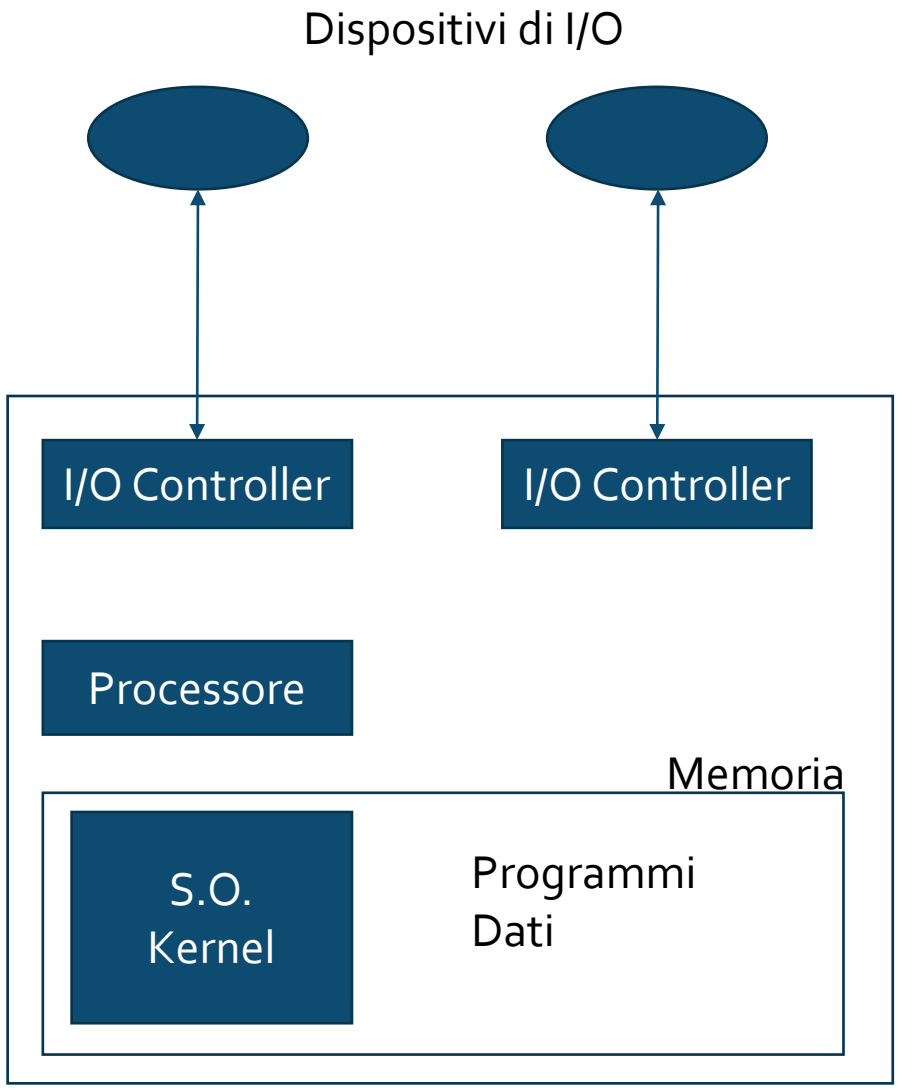
Il Sistema Operativo agisce quindi in maniera **trasparente**

Interfaccia: componente fisico o logico che permette a due o più sistemi elettronici di comunicare e interagire

Servizi offerti dal Sistema Operativo

- **Creazione dei programmi**: compilatore, debugger come utilità offerte al programmatore. Non sono parte del SO ma sono accessibili tramite esso
- **Esecuzione dei programmi**: caricamento in memoria dei programmi, inizializzazione dei dispositividi I/O, ecc.
- **Accesso ai dispositivi di I/O**: l'utente/programmatore ignora il set di istruzioni e i segnali dei dispositivi
- **Accesso controllato ai file**: comprensione del formato, meccanismi di protezione, associazione file indirizzi di memoria
- **Accesso al sistema** (inteso in senso lato)
- **Rilevazione e correzione degli errori** hardware o generati da programmi in esecuzione
- **Contabilità e statistiche d'uso delle risorse**, dei tempi di risposta (fine: migliorare le prestazioni)

Sistema Operativo come gestore delle risorse



Il Sistema Operativo:

- Dirige la CPU nell'utilizzo delle altre risorse del sistema e nella temporizzazione dell'esecuzione dei programmi
- Decide quando un programma in esecuzione può utilizzare una risorsa – il processore stesso è una risorsa

Kernel: Parte del Sistema Operativo risiedente in memoria centrale, contiene le funzioni del SO usate più frequentemente

Batch multiprogrammati

Batch multiprogrammati: sistemi dove è consentita la multiprogrammazione, ovvero l'esecuzione di più programmi in contemporanea

Mono-programmazione

Ad esempio:

- Lettura di un record: 0.0015 sec
- Esecuzione di 100 istruzioni: 0.0001 sec
- Scrittura di un record: 0.0015 sec
- Totale 0.0031 sec

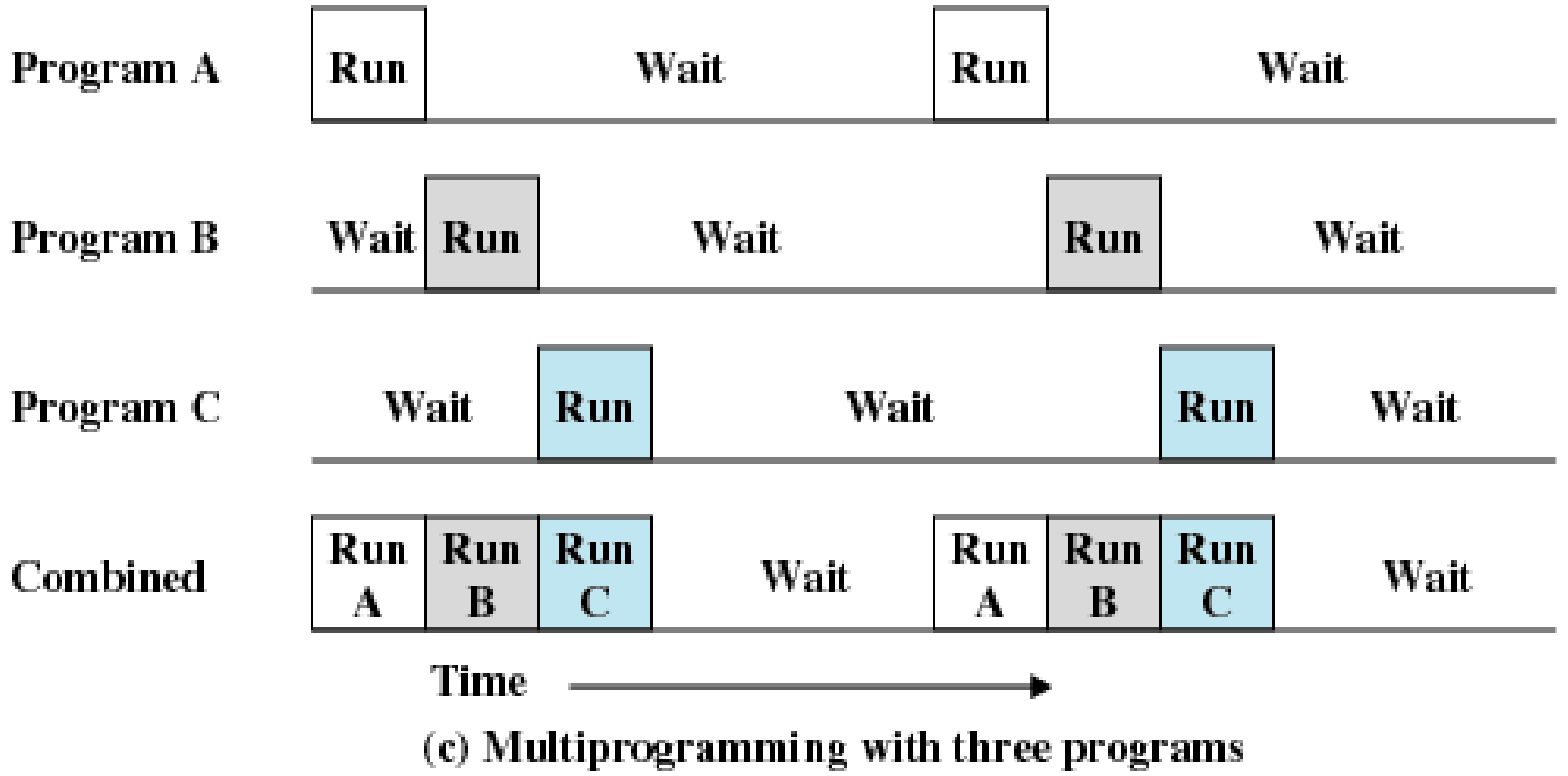
Percentuale di utilizzo della CPU:

$$\frac{0.0001}{0.0031} = 0.032 = 3{,}2\%$$

Multi-programmazione

- Presenza di più programmi in memoria
- Obiettivo: limitare l'inattività del processore, quando un job effettua un'operazione di I/O la CPU può essere impegnata da un altro processo
- Elaborazione seriale dei task

Multi-programmazione



(c) Multiprogramming with three programs

Difficoltà della multi-programmazione:

- Gestione della memoria
- Decidere quale Job mandare in esecuzione (schedulazione)

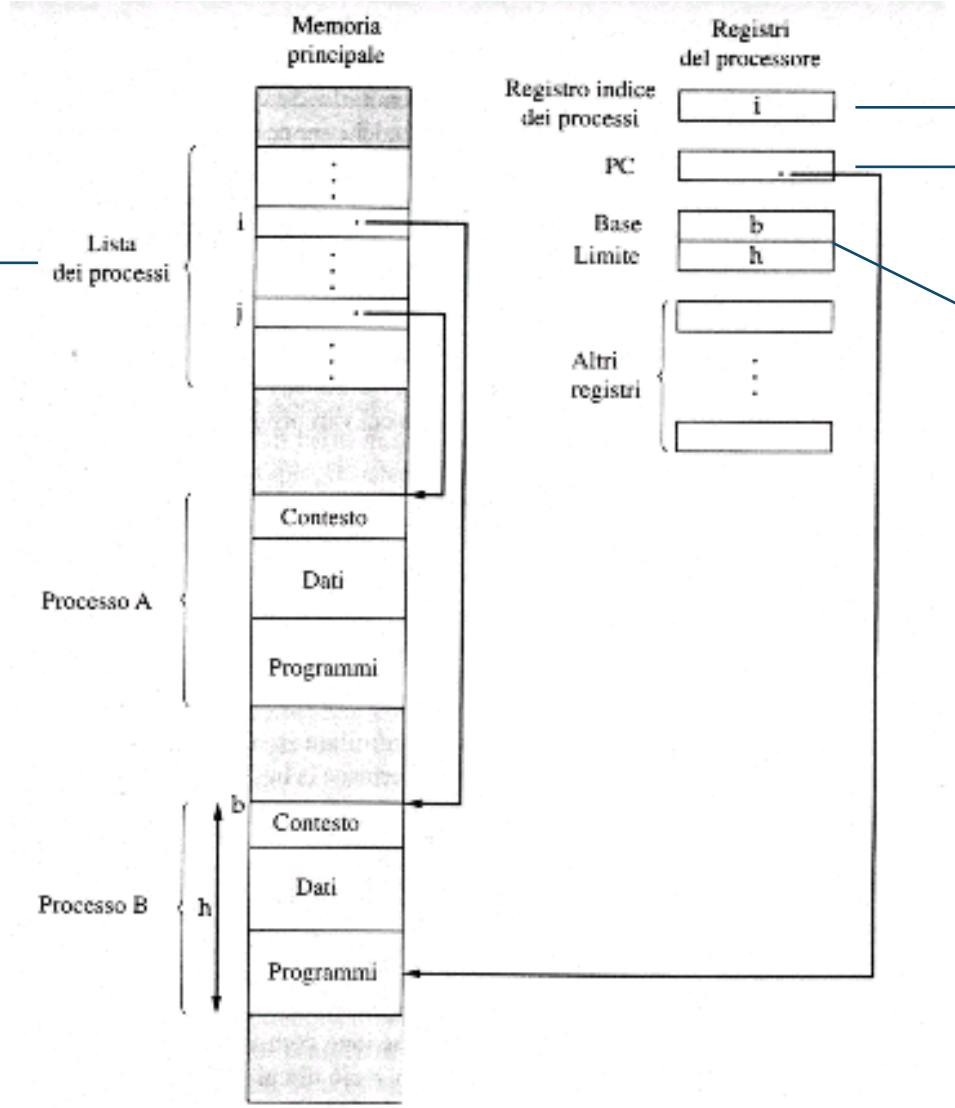
Processo = Job = Task

- Un programma in esecuzione
- L'anima di un programma (!?)
- Un'entità assegnata ad un processore e da essa eseguita
- **Un'attività caratterizzata dall'esecuzione di una sequenza di istruzioni, uno stato corrente e un set di istruzioni di sistema**

Componenti

- Programma
  - Codice eseguibile
- Dati
  - Variabili
  - Spazio di lavoro
  - Buffer
- Contesto di esecuzione (info necessarie al SO per gestire il processo)
  - Contenuto dei registri della CPU
  - Priorità
  - Stato di esecuzione
  - Stato di attesa su un dispositivo di I/O

Implementazione di un processo



Gestita dal SO

Indica il processo in esecuzione

Punta alla istruzione successiva del processo da eseguire

Protezione delle aree di memoria

Gestione della memoria

Il Sistema Operativo deve assolvere 5 compiti:

1. Isolamento dei processi
2. Allocazione e gestione automatica della memoria: la gerarchia delle memorie deve essere trasparente all'utente
3. Supporto alla programmazione modulare: variazione di dimensione dei programmi
4. Protezione e controllo dell'accesso: gestione di aree di memoria condivise tra i processi
5. Memorizzazione a lungo termine

Necessità soddisfatte da:

- **memoria virtuale:** i programmi indirizzano la memoria con riferimenti logici ignorando gli aspetti fisici, quando un programma è in esecuzione solo una sua parte risiede effettivamente in memoria centrale
- **file system:** implementa la memorizzazione a lungo termine

Stati dei processi

Il compito principale di un Sistema Operativo è il controllo dell'esecuzione dei processi

In particolare, è possibile classificare lo stato attuale di un processo mediante, appunto, uno **stato**

Tale classificazione consente di gestire in maniera differente processi in stato differente

Descrizione dei Processi

Il Sistema Operativo necessita di uno strumento per gestire i processi, che tenga traccia di tutte le informazioni disponibili

Esso prende il nome di **Descrittore di Processo**, oppure **Process Control Block (PCB)**

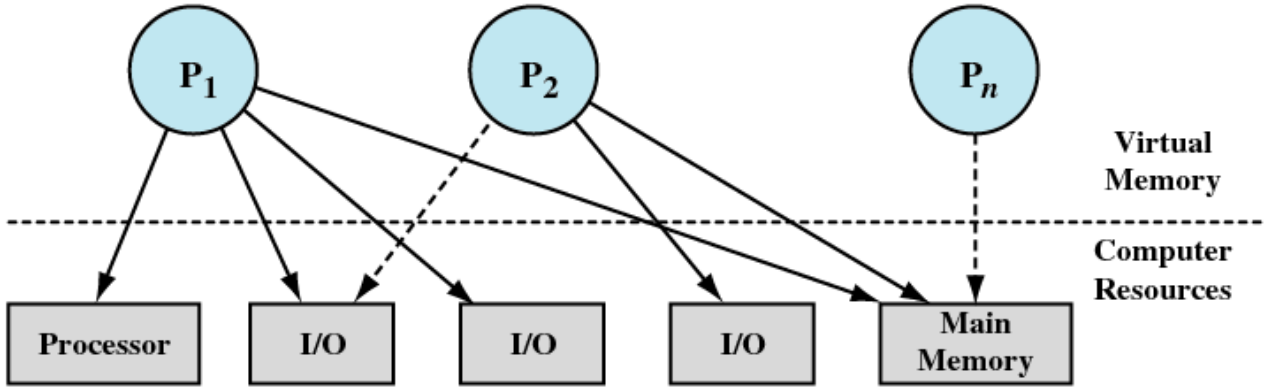


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

Process Control Block – PCB

Identificatore del Processo

- Process IDentification (PID) – Valore numerico

Info sullo stato del processore

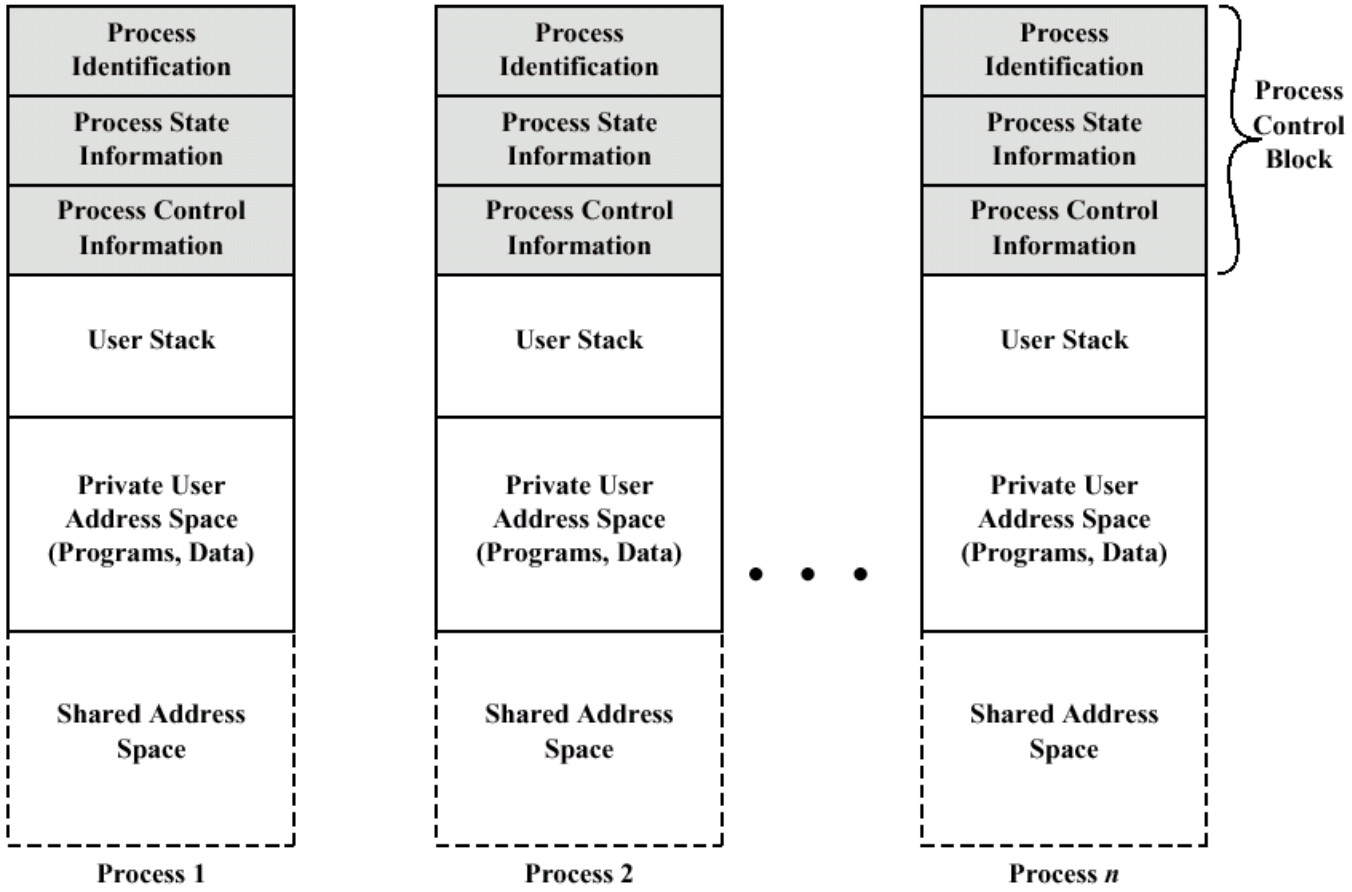
- Registri dati visibili all'utente (i quali dipendo dall'architettura del calcolatore)
- Registri di controllo e di stato
  - Program Counter – indirizzo della prossima istruzione da eseguire
  - Registri di stato – includono i flag per l'abilitazione degli interrupt
  - Registri che contengono codici relativi alla condizione (segno, overflow, etc)
- Puntatori allo stack
  - Usato per procedure e funzioni

Process Control Block – PCB

Informazioni di controllo del processo

- Schedulazione e informazioni di stato
  - Stato del processo (running, ready, etc.)
  - Priorità nelle code di scheduling
  - Informazioni correlate alla schedulazione (tempo di attesa, di esecuzione, etc)
  - Evento del quale è in attesa (se è in attesa)
- Strutturazione dati
  - Puntatori ad altri processi (figli/padre o per l'implementazione di code)
- Comunicazione tra processi
  - Flag, segnali e messaggi per la comunicazione
- Privilegi
  - In relazione all'uso della memoria, dei dispositivi, etc
- Gestione della memoria
  - Limiti di memoria: insieme degli indirizzi accessibili (base, limite)
- Contabilizzazione delle risorse
  - Risorse controllate dal processo (lista dei file aperti, lista dei dispositivi I/O) e la loro storia

Immagine dei processi in memoria



Nell'esempio le immagini occupano locazioni contigue di memoria, in una implementazione reale ciò può non essere vero. Dipende dalla politica di gestione della memoria

Creazione e terminazione dei processi

Eventi che portano alla **creazione** dei processi:

1. Richiesta da terminale (un utente accede al sistema)
2. Il Sistema Operativo genera un processo sulla base della richiesta di un processo utente

   Ad es. stampa – il processo generatore continua la sua esecuzione
3. Un processo utente genera un nuovo processo

   Processo padre – Processo figlio

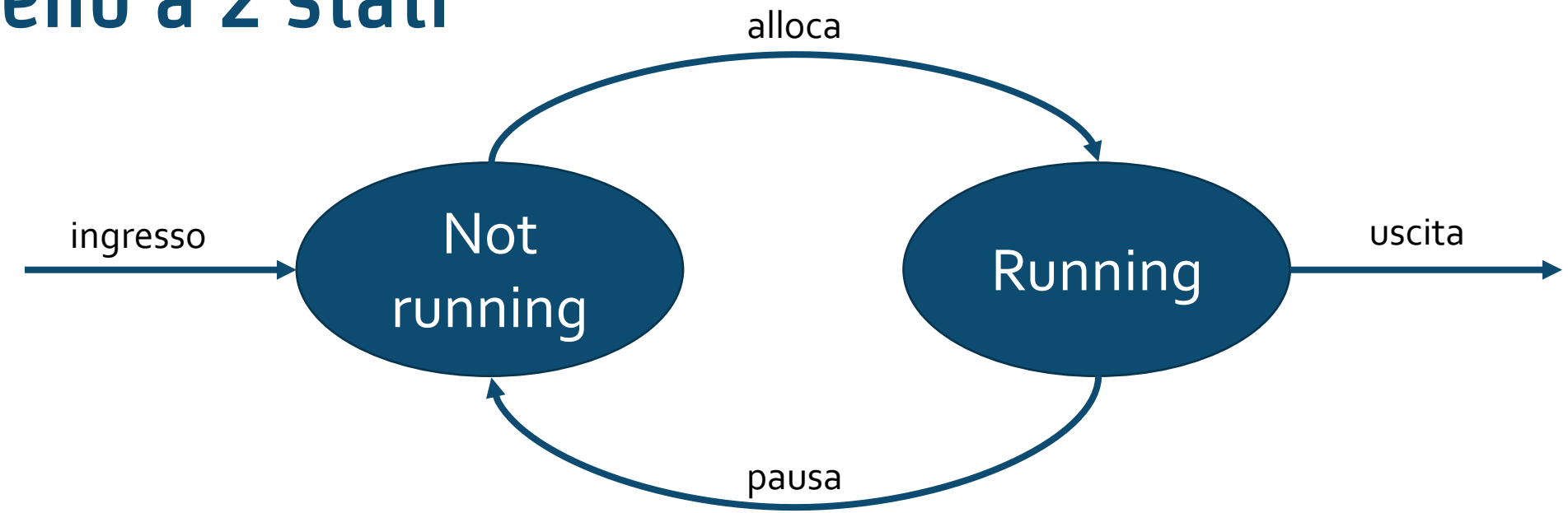
   Es. sfruttare il parallelismo: un processo server genera diverse istanze per gestire diverse richieste

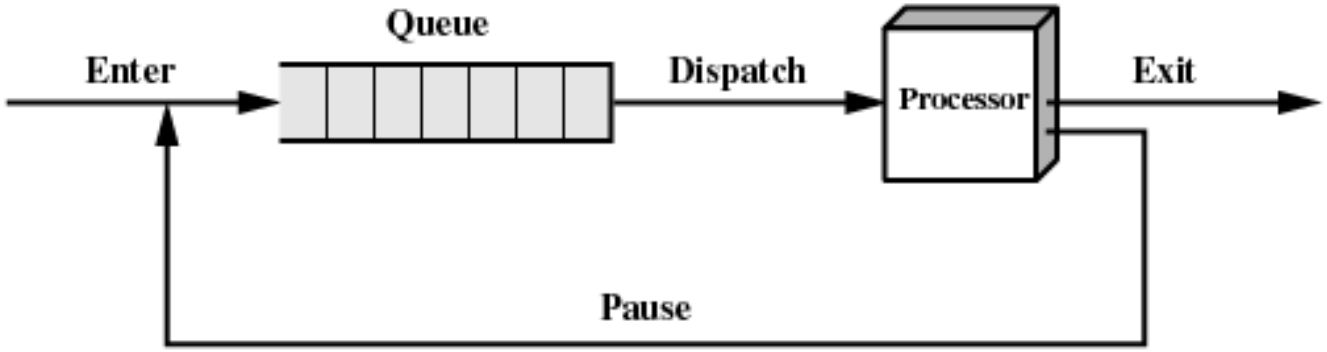
Creazione e terminazione dei processi

Eventi che portano alla **terminazione** dei processi:

1. Terminazione normale (end)
2. Uscita dell'utente dall'applicazione
3. Superamento del tempo massimo
4. Memoria non disponibile
5. Violazione dei limiti di memoria
6. Fallimento di un'operazione (aritmetica – I/O)
7. Terminazione del genitore
8. Richiesta del genitore

Modello a 2 stati





(b) Queuing diagram

Modello a 2 stati

Nel modello a 2 stati, lo stato *not-running* include 2 possibilità:

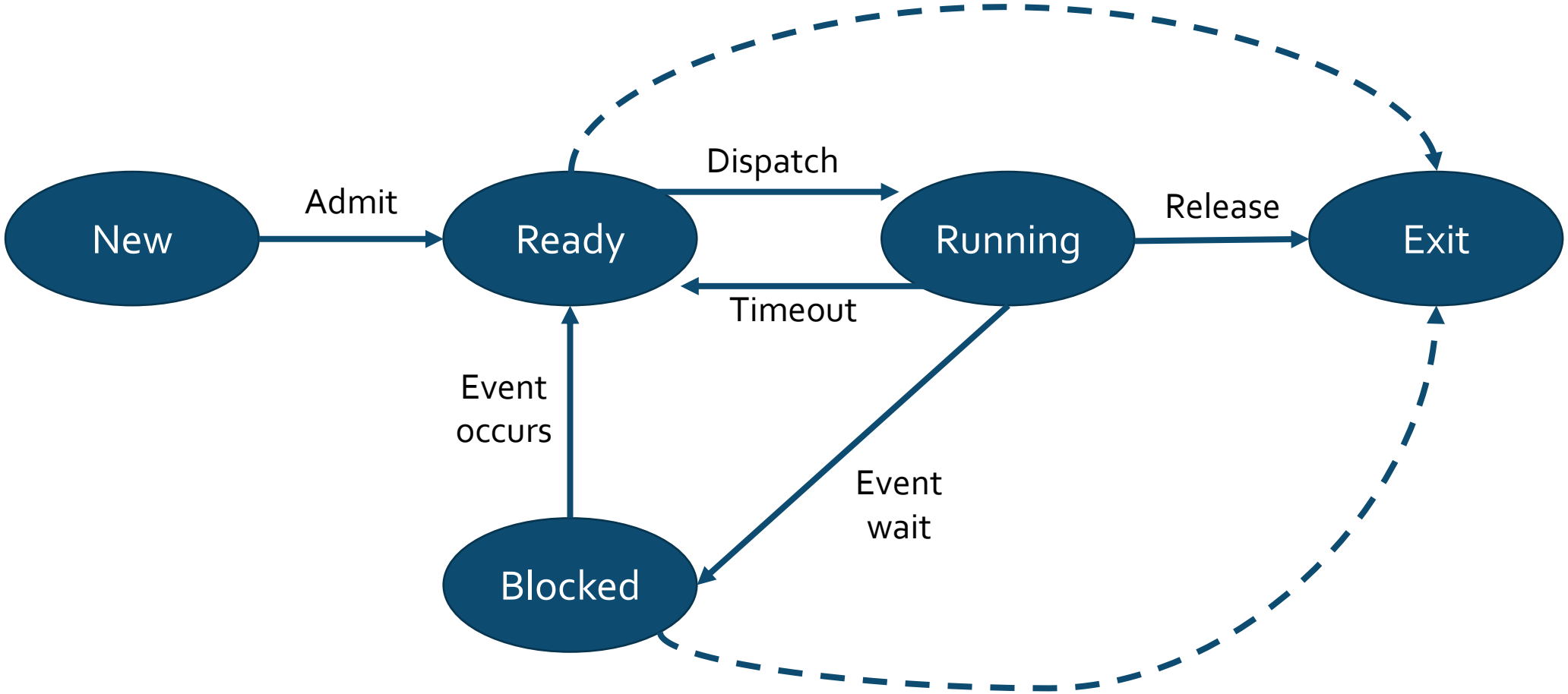
- Il processo è pronto per essere eseguito
- Il processo è in attesa di un evento o di un dispositivo I/O

Il dispatcher (scheduler) non può semplicemente scegliere il processo da più tempo in attesa, poiché esso potrebbe essere in attesa di un trasferimento I/O

Si giunge dunque al modello a 5 stati:

- Vengono introdotti gli stati *New* e *Exit (Terminated)*
- Lo stato di *Not Running* viene diviso negli stati *Ready* (pronto all'esecuzione) e *Blocked* (in attesa di un evento, una risorsa, etc)

Modello a 5 stati



Le transizioni tratteggiate si hanno nel caso in cui un processo genitore termina il processo figlio

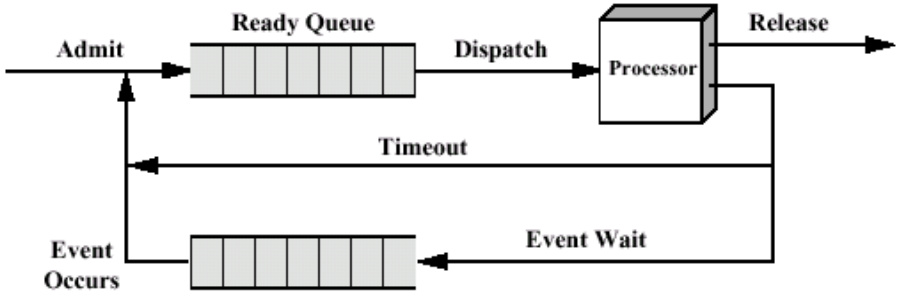
Modello a 5 stati

I nuovi stati New e Exit:

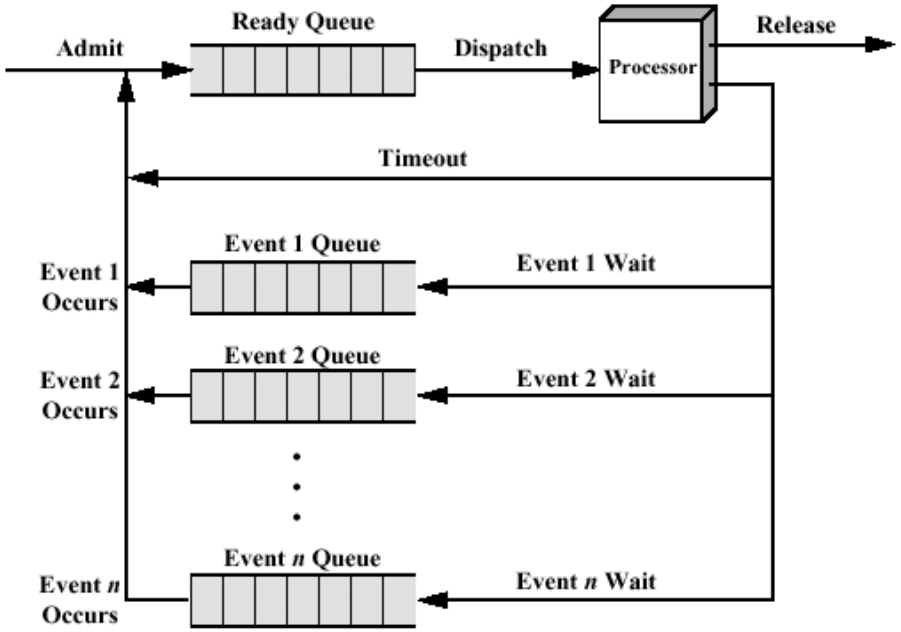
- **New:** SO associa al processo il PID, alloca e costruisce le tabelle per la gestione del processo. NB: il processo non è caricato in memoria

- **Exit:** rilascio delle risorse. Il SO può mantenere alcune info (es. contabilità)

Strategie di accodamento



(a) Single blocked queue



(b) Multiple blocked queues

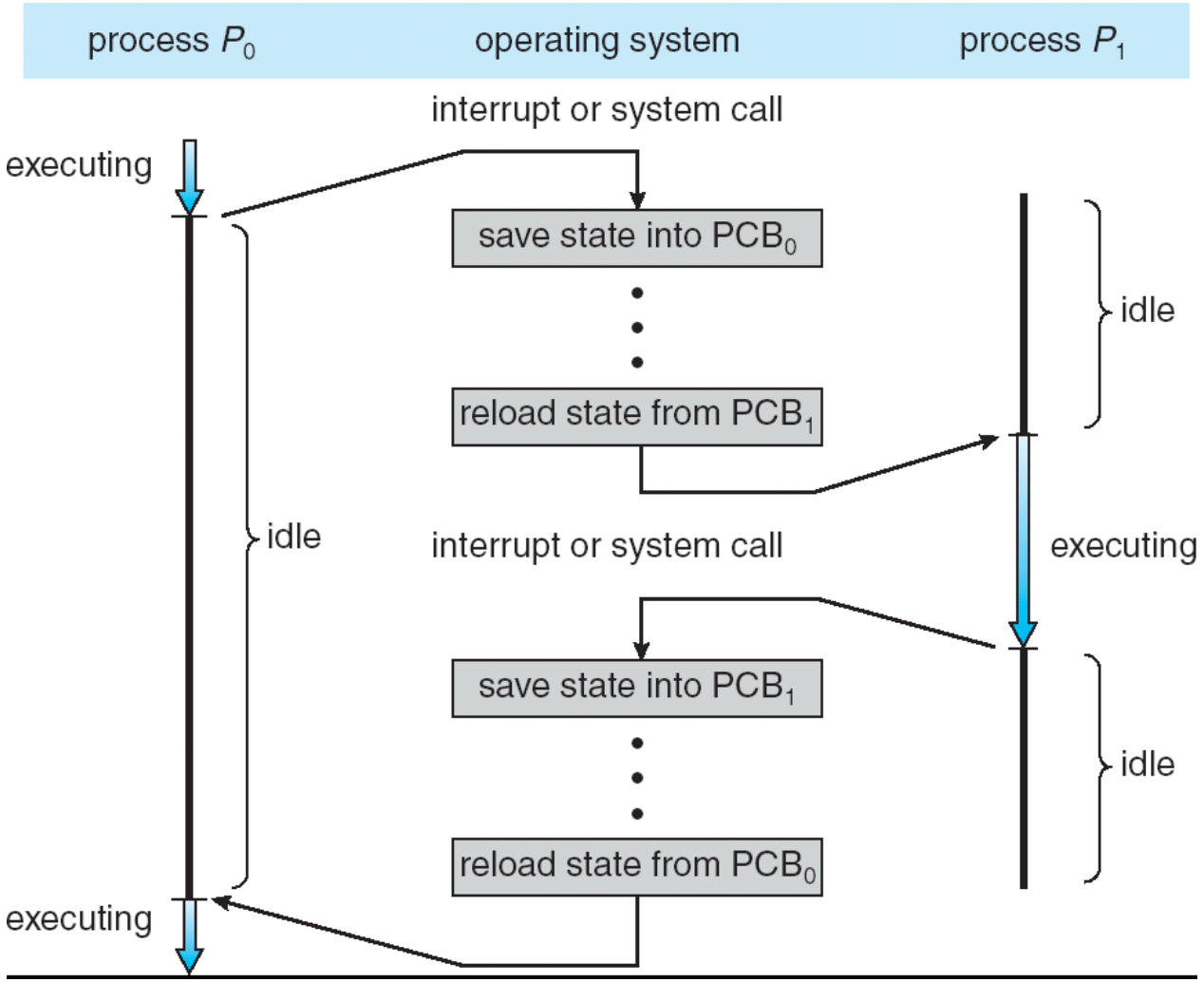
Context Switch

Riguarda il passaggio della CPU ad un nuovo processo

Cause:

- Clock interrupt
  - Il processo ha terminato il tempo a sua disposizione (torna nella coda di ready)
- I/O interrupt
  - Una operazione di I/O è terminata, il SO sposta il processo in attesa di tale evento da blocked a ready e decide se riprendere l'esecuzione del processo precedente
- Memory fault
  - L'indirizzo di memoria generato è sul disco (memoria virtuale): deve essere portato in RAM. Il SO carica il blocco, nel frattempo il processo che ha generato la richiesta è in blocked, al termine del trasferimento andrà in ready
- Trap
  - Errore di esecuzione (il processo potrebbe andare in exit)
- Supervisor call
  - Es. file open, il processo utente va in blocked

Context Switch



Operazioni svolte dal SO in modalità supervisor al momento del cambio di processo in stato di running:

- Salvataggio del contesto del processo che abbandona la CPU (valori dei registri della CPU: pc, psw, reg, ecc.)
- Cambio del valore di stato nel PCB (running -> [ready, blocked, exit])
- Spostamento del PCB in nuova coda (ready o blocked) o deallocare le sue risorse (exit)
- Aggiornamento delle strutture dati gestione memoria (area dello stack)
- Selezione di nuovo processo per lo stato running (dispatcher)
- Aggiornamento del suo stato nel PCB
- Ripristino del contesto

Context Switch

- Context-switch time è overhead; il sistema non svolge nessun compito utile (all'utente)
- Il tempo dipende dalla complessità del SO e dall'hardware

Modalità di esecuzione dei processi

Modalità utente: esecuzione di processi utente

Modalità Sistema o Kernel o Controllo: esecuzione di istruzioni che hanno come scopo

- Gestione dei Processi
  - Creazione e terminazione
  - Schedulazione
  - Cambio di contesto
  - Sincronizzazione
  - PCB
- Gestione della memoria
  - Allocazione
  - Trasferimento disco/RAM e viceversa
  - Gestione della paginazione, segmentazione…
- Gestione I/O
  - Gestione buffer
  - Allocazione canali I/O
- Supporto
  - Gestione interruzioni
  - Contabilità

Creazione dei Processi

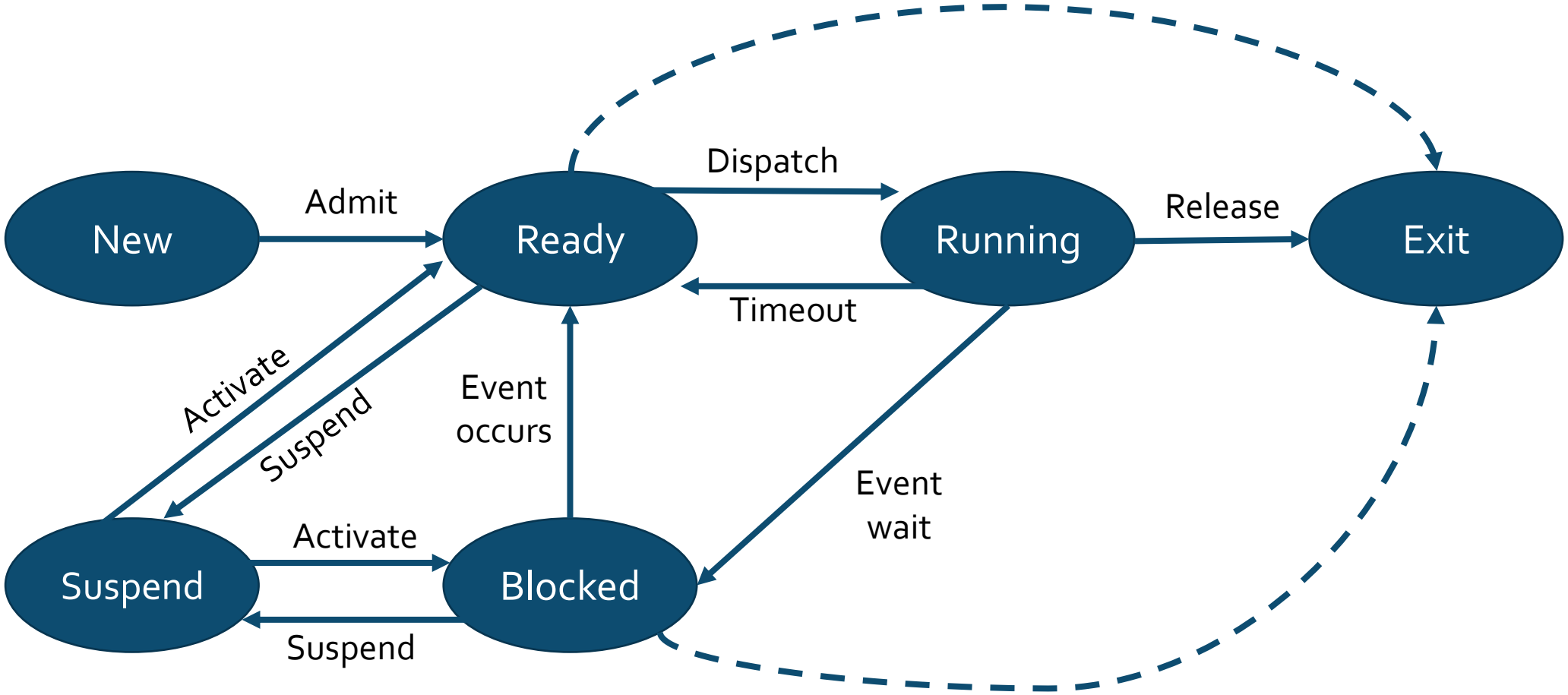
1. Assegnare al processo un PID unico; aggiungere una entry level alla tabella dei processi
2. Allocare lo spazio per il processo e per tutti gli elementi della sua immagine (PCB, User Stack, area di memoria dati e istruzioni, aree condivise)
3. Inizializzazione del PCB
   - Stato del processore = 0
   - PC = prossima istruzione
   - Puntatori allo stack
   - Stato = ready
4. Inserimenti nella coda di ready
5. Estende le strutture al fine della fatturazione o delle statistiche

Evoluzione del modello a 5 stati

- Nonostante la memoria virtuale, un programma per essere eseguito deve essere in RAM
- Con elevata probabilità tutti i processi in memoria restano in attesa di operazioni di I/O

- Processore inattivo (molto più veloce di I/O)
- Soluzioni:
  - Espandere la memoria
    - Costoso
    - Poco efficiente (programmi sempre più grandi)
  - Effettuare lo **swapping**: spostare un processo dalla RAM alla memoria secondaria
    - Introduzione dello stato *suspend*
    - NB: lo swapping è un'ulteriore operazione di I/O, ma in generale è la più rapida tra le operazioni di I/O

Modello a 6 stati



Activate

Modello a 6 stati

Swap out: scaricamento del processo sul disco, ovvero la transizione da blocked a suspended

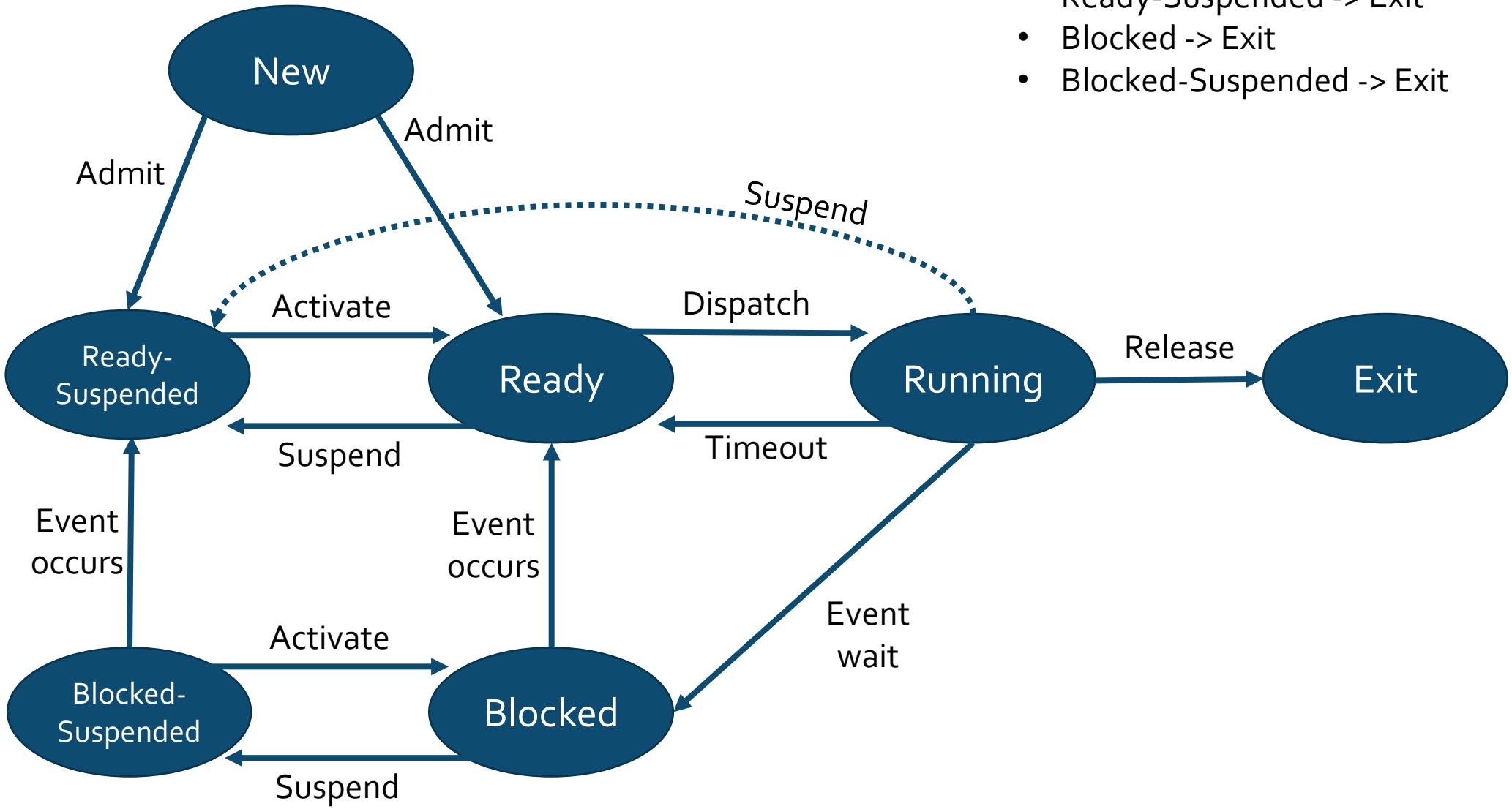
Swap in: operazione inversa allo swap out

Il modello a 6 stati pone un problema analogo a quello a 2 stati: scindiamo quindi lo stato di suspended in *Ready/Suspended* e *Blocked/Suspended*

Ciò consente al Sistema Operativo di scegliere tra i processi in New e in Suspended, per essere portati in Ready

Modello a 7 stati

Sono state omesse le transizioni
- Ready -> Exit
- Ready-Suspended -> Exit
- Blocked -> Exit
- Blocked-Suspended -> Exit



Modello a 7 stati

Esiste uno schema di gestione della memoria noto come **memoria virtuale**, nel quale un processo può trovarsi solo parzialmente in RAM. Quando si fa riferimento a un indirizzo su disco questo viene caricato

Dunque gli stati di sospensione in quel caso sono inutili

La transizione da Ready a Ready-Suspended avviene laddove vi sia la necessità di maggiore memoria per allocare un processo più grande o a maggiore priorità

Schedulazione

ISLab Intelligent Systems Laboratory

Schedulazione

La schedulazione, ossia la scelta dell'ordine di esecuzione dei processi, e la relativa politica di allocazione deve tenere in considerazione i seguenti fattori:

Equità: tutti i processi

- Appartenenti ad una stessa classe
- O con richieste simili
- O stesso costo

Devono avere la stessa possibilità di accesso alla risorsa

Tempo di risposta differenziale: il SO discrimina tra classi che hanno bisogno di risorse diverse e di tempi diversi

- Es. i processi I/O-bound (ovvero con forte uso di I/O) vengono schedulati per primi

Efficienza: massimizzare il throughput (quantità di dati trasmessi), minimizzare il tempo di risposta

Schedulazione

Con scheduling si intende un insieme di tecniche e meccanismi interni del Sistema Operativo che amministrano l'ordine in cui il lavoro viene svolto

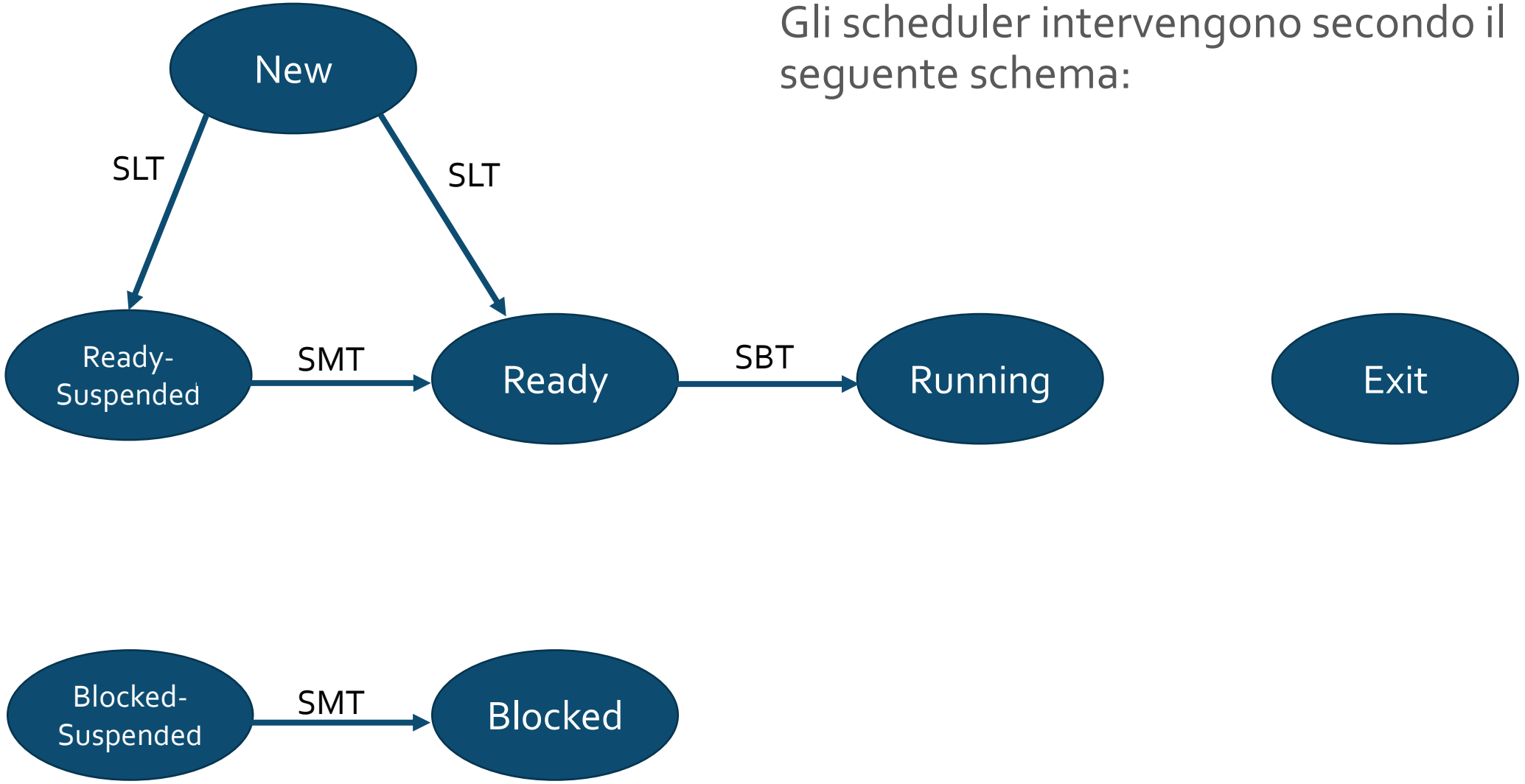
Obiettivo primario dello scheduling è l'ottimizzazione delle prestazioni del sistema

Il Sistema Operativo può prevedere fino a 3 tipi di scheduler:

- Scheduler di lungo termine (SLT)
- Scheduler di medio termine (SMT)
- Scheduler di breve termine (SBT)

Schedulazione

Gli scheduler intervengono secondo il seguente schema:



Scheduler di lungo termine

- Determina quali programmi sono ammessi nel sistema per essere processati (new -> ready, new -> ready/suspended)

- Controlla il grado di multiprogrammazione (new -> ready)
  - Più processi = minor tempo percentuale di esecuzione per ciascuno di questi

- Stime effettuate dal programmatore o dal sistema forniscono informazioni sulle risorse necessarie all'esecuzione, come le dimensioni della memoria, il tempo di esecuzione totale, etc

- Il lavoro dello scheduler di lungo termine si basa quindi sulla stima del comportamento globale dei job

Scheduler di lungo termine

Strategie principali:

1. fornire alla coda dei processi pronti (e quindi allo scheduler di breve termine) gruppi di processi che siano bilanciati tra loro nello sfruttamento della CPU e dell'I/O
2. aumentare il numero di processi provenienti dalla coda batch, quando il carico della CPU diminuisce
3. diminuire (fino anche a bloccare) i lavori provenienti dalla coda batch, quando il carico aumenta e/o i tempi di risposta del sistema diminuiscono

La frequenza di chiamata dell'SLT è bassa e consente di implementare strategie anche complesse di selezione dei lavori e di dimensionamento del carico dei processi da inviare alla coda pronti (ready)

Scheduler di medio termine

Si occupa di gestire la schedulazione delle transizioni

- Ready suspended -> Ready
- Blocked suspended -> Blocked

Si bassa sulla necessità di gestire il livello di multiprogrammazione.

La presenza di molti processi sospesi in memoria riduce la disponibilità per nuovi processi pronti. In questo caso lo scheduler di breve termine è obbligato a scegliere tra i pochi processi pronti…

- utilizza le informazioni del Descrittore di Processo (PCB) per stabilire la richiesta di memoria del processo
- tenta di allocare spazio in memoria centrale
- riposiziona il processo in memoria nella coda dei pronti

Viene attivato

- quando si rende disponibile lo spazio in memoria
- quando l'arrivo di processi pronti scende al di sotto di una soglia specificata

Scheduler di breve termine

Prende anche il nome di **dispatcher**

Si occupa di gestire la transizione Ready -> Run

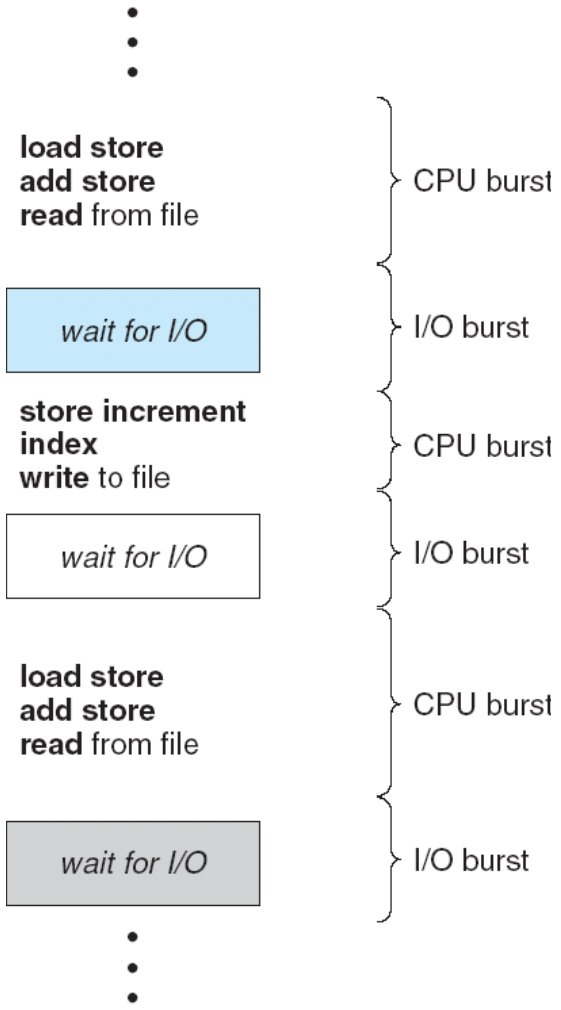
Viene eseguito molto frequentemente

Invocato quando si verifica un evento:

- Clock interrupts
- I/O interrupts
- Chiamate del SO
- Signals

La sua principale strategia è orientata alla massimizzazione delle prestazioni del sistema secondo un specifico insieme di obiettivi

Scheduling della CPU nel Dispatcher



Esecuzione di un processo:

1. Ciclo di elaborazione (CPU)
2. Attesa di completamento di I/O

Lo scheduling della CPU riguarda la distribuzione delle sequenze di elaborazione della CPU

Processo **I/O bound**: processo che presenta molte operazioni di I/O

Processo **CPU bound**: processo che presenta poche operazioni di I/O

Algoritmi di schedulazione

Definiamo anzitutto

- Il **tempo di ricircolo** come il tempo trascorso tra l'avvio di un processo (la sua immissione nel sistema) e la terminazione dello stesso
- Il **tempo di attesa** come il tempo che un processo trascorre in attesa delle risorse a causa di conflitti con altri processi. Si può calcolare come la differenza tra il tempo di ricircolo e il tempo di esecuzione. Sostanzialmente valuta la sorgente di inefficienza, essendo il prezzo da pagare per condividere delle risorse

Vedremo ora gli algoritmi di schedulazione. L'efficienza di tali algoritmi è misurabile utilizzando i tempi sopracitati.

Un buon algoritmo di scheduling cerca di bilanciare l'esecuzione dei processi al meglio, massimizzando l'uso del processore e riducendo i tempi di attesa.

Decision mode

Non preemptive
(non interrompibile)

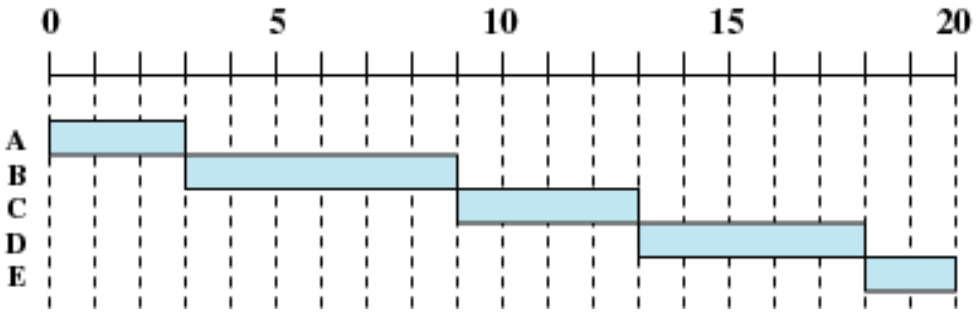
- Un processo in running abbandonerà tale stato solo se termina l'esecuzione o si blocca per un'operazione di I/O

Preemptive
(interrompibile)

- Un processo in running può essere interrotto e spostato in Ready dal Sistema Operativo (ad esempio se giunge un processo «più importante» in ready)

- PRO: Nessun processo può monopolizzare il processore
- CONTRO: Crea problemi dove vi sono processi che condividono dati, richiedono meccanismi di sincronizzazione

Algoritmo First Come First Served – FCFS



Applica il principio della coda:

- Ogni processo entra in coda di Ready
- Quando un processo abbandona lo stato di running si seleziona il processo che da più tempo è in stato di Ready

Favorisce i processi CPU bound. Un processo I/O bound che richiede poco tempo di esecuzione potrebbe attendere molto tempo prima che gli venga assegnata la CPU.

Genera l'effetto convoglio: tutti i processi in coda attendono che un processo CPU-bound termini

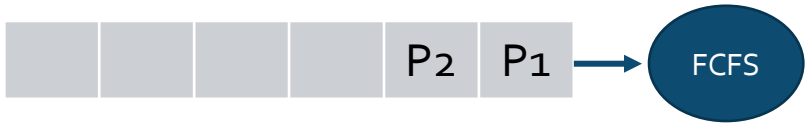
Senza prelazione: basso sfruttamento delle componenti, basso lavoro utile del sistema

Algoritmo First Come First Served – FCFS

Le prestazioni dipendono unicamente dall'ordine di arrivo dei Jobs.

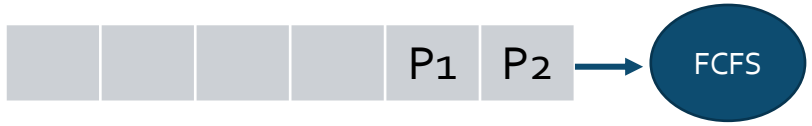
Ad esempio:

- P1 ha un tempo di esecuzione totale di 20
- P2 ha un tempo di esecuzione totale di 2



Tempi di riciclo:
- P1 = 20
- P2 = 22
- Medio = 21

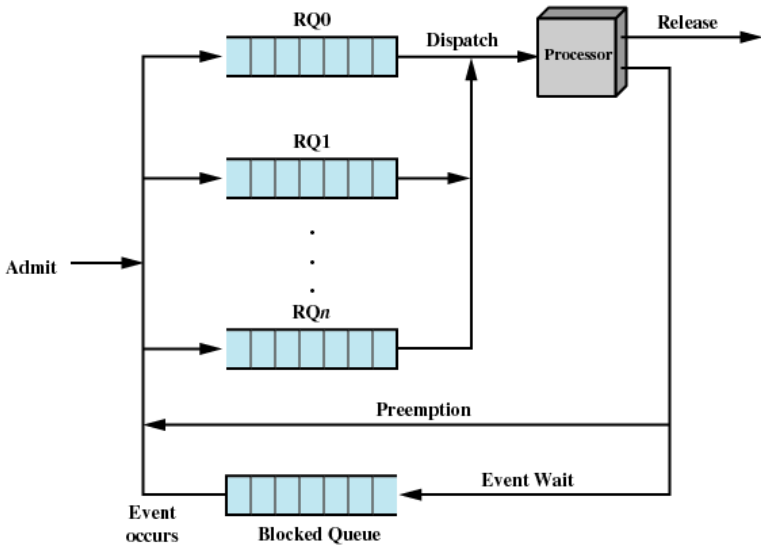
Tempo di attesa:
- P1 = 0
- P2 = 20
- Medio = 10



Tempi di riciclo:
- P1 = 2
- P2 = 22
- Medio = 12

Tempo di attesa:
- P1 = 0
- P2 = 2
- Medio = 1

Algoritmo Event Driven



Schema che ragiona secondo un **valore di priorità** assegnato a ciascun processo: lo scheduler sceglierà sempre il processo pronto con priorità maggiore

La priorità può essere assegnata dall'utente o dal sistema e può essere di tipo statico o dinamico. La priorità dinamica varia in base a:

- Valore iniziale
- Caratteristiche del processo
- Richiesta di risorse
- Comportamento durante l'esecuzione

Tale modello è generalmente applicato nei sistemi dove il tempo di risposta, soprattutto ad eventi esterni, è critico

Algoritmo Event Driven

Il sistemista può influire sull'ordine in cui uno scheduler serve gli eventi esterni modificando le priorità assegnate ai processi

Le prestazioni sono dipendenti da una accurata pianificazione nell'assegnazione delle priorità

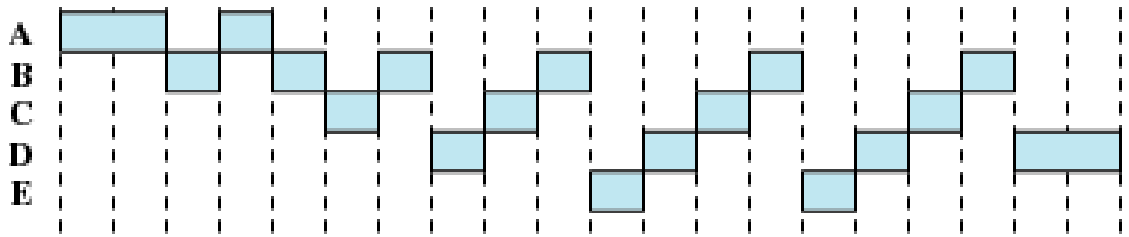
Le priorità sono definite:

- Internamente al SO: utilizzando grandezze misurabili quali l'uso di memoria, file aperti, rapporto tra picchi medi di I/O e di CPU
- Esternamente al SO: rilevanza del processo, criticità

PROBLEMA: Non è in grado di garantire il completamento di un processo in un intervallo di tempo finito dalla sua creazione, in quanto potrebbe essere continuamente sorpassato da processi a priorità più alta. Tale situazione prende il nome di **starvation**

SOLUZIONE: Usare l'**aging** – al passare del tempo in stato di Ready la priorità del processo aumenta

Algoritmo Round Robin – RR



Utilizza come principio il **time slice**, ovvero una preemption basata sul clock (clock interrupt)

Ogni processo utilizza il processore per un dato intervallo di tempo Valori tipici: 10-100msec

Al verificarsi dell'interrupt il processo in esecuzione viene portato nella coda di ready (gestita FIFO – First In First Out)

Con n processi in ready e un time quantum q, ogni processo ottiene $1/n$ del tempo di CPU in frazioni di tempo al più pari a q. Tempo massimo di attesa in ready: $(n-1)q$.

Prestazioni dipendenti dal quanto di tempo:

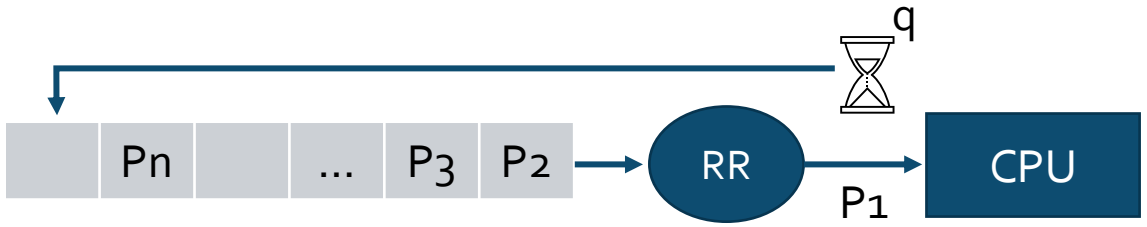
- q troppo grande: degenera in FCFS
- q troppo piccolo: incrementa il numero di context switch, consumando le risorse

Algoritmo Round Robin – RR

La schedulazione Round Robin fornisce una buona condivisione delle risorse del sistema:

- i processi più brevi possono completare l'operazione in un q (buon tempo di risposta)
- i processi più lunghi sono forzati a passare più volte per la coda dei processi pronti (tempo proporzionale alle loro richieste di risorse)
- per i processi interattivi lunghi, se l' esecuzione tra due fasi interattive riesce a completarsi in un q, il tempo di risposta è buono

La realizzazione di uno scheduler RR richiede il supporto di un Timer che invia un'interruzione alla scadenza di ogni q, forzando lo scheduler a sostituire il processo in esecuzione. Il timer viene riazzerato se un processo cede il controllo al Sistema Operativo prima della scadenza del suo q



Algoritmo Highest Response Ratio Next – HRRN

Siano:

- w il tempo speso in coda di Ready (quindi in attesa della disponibilità del processore)
- s il tempo di servizio previsto

Si definisce il Response Ratio come

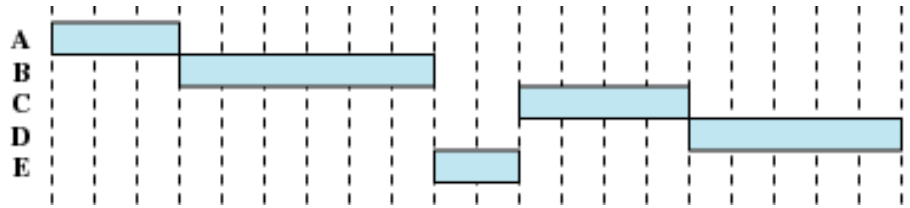
$$RR = \frac{w + s}{s}$$

L'algoritmo Highest Response Ratio Next manda in esecuzione il processo con il più alto valore di RR

Osservazioni:

- Quando un processo entra in coda per la prima volta ha un RR = 1
- Tiene in considerazione l'età del processo, applica quindi un meccanismo di aging (w)

Algoritmo Shortest Process Next – SPN

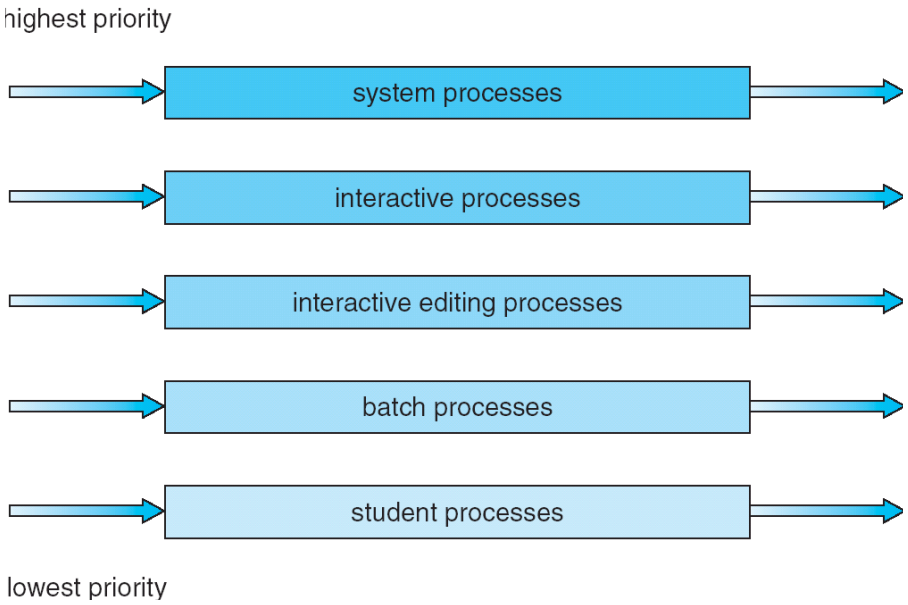


Il processo scelto dalla coda di ready è quello con il più breve tempo di esecuzione stimato (più breve sequenza di operazioni svolte dal processore)

- **PRO**: SPN è ottimale nel senso che fornisce il tempo medio di attesa minimo, per un dato set di processi
- **CONTRO**: Difficile stimare la durata della prossima sequenza di CPU (oltre che oneroso), è possibile la starvation per processi fortemente CPU bound

Ha una versione preemptive: se arriva nuovo processo con una sequenza di CPU minore del tempo necessario per la conclusione della sequenza di CPU del processo attualmente in esecuzione, si ha il prerilascio della CPU a favore del processo appena arrivato. Questo schema è anche noto come **Shortest Remaining Time First (SRTF)** oppure **Shortest Remaining Time Next (SRTN)**

Schedulazione a code multiple



La coda di Ready viene divisa in sotto-code:

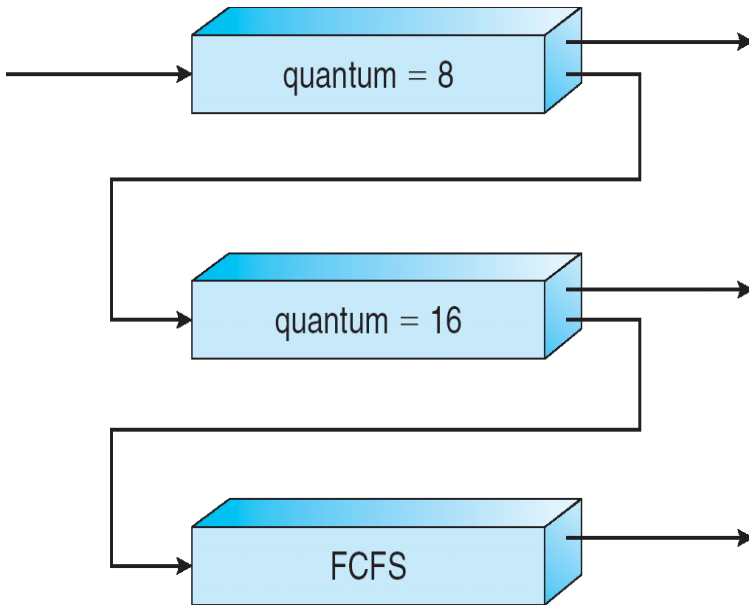
- Foreground (processi interattivi)
- Background (processi batch)

Ogni coda ha un proprio algoritmo di Schedulazione (es. Foreground – RR, Background – FCFS)

Necessita di uno scheduling tra le code:

- A priorità fissa e con prelazione (ad esempio serve prima dalla Foreground e poi dalla Background)
- Time slice – ad ogni coda è associato un certo ammontare di tempo di CPU (ad esempio 80% alla Foreground in RR, 20% alla Background in FCFS)

Schedulazione a code multiple con feedback



Implementa l'aging: un processo può essere spostato da una coda all'altra

Code Multilevel-Feedback definite dai seguenti parametri:

- Numero di code
- Algoritmo di scheduling per ogni coda
- Metodi usati per l'up-grading e il down-grading di ogni processo

Esempio:

Tre code:

- Q_0 – RR con time quantum 8 milliseconds
- Q_1 – RR con ime quantum 16 milliseconds
- Q_2 – FCFS

Scheduling:

- Un nuovo processo entra nella coda Q_0
- Quando ottiene la CPU, la impegna per 8 ms. Se non termina entro gli 8 ms è spostato in Q_1
- Il processo in Q_1 viene nuovamente servito con politica RR e riceve la CPU per ulteriori 16 ms
- Se ancora non termina viene spostato in Q_2 e servito con FCFS