Sistemi Operativi

GESTIONE

DEI

PROCESSI

Processi

- Concetto di Processo
- Scheduling di Processi
- Operazioni su Processi
- Processi Cooperanti
- Concetto di Thread
- Modelli Multithread
- I thread in Java

Concetto di Processo

- L'esecuzione di programmi ha diversi nomi in diversi contesti:
  - Sistemi Batch – **job**
  - Sistemi Time-sharing– **processo** o **task**
- I termini ***job*** e ***processo*** si usano spesso come sinonimi.

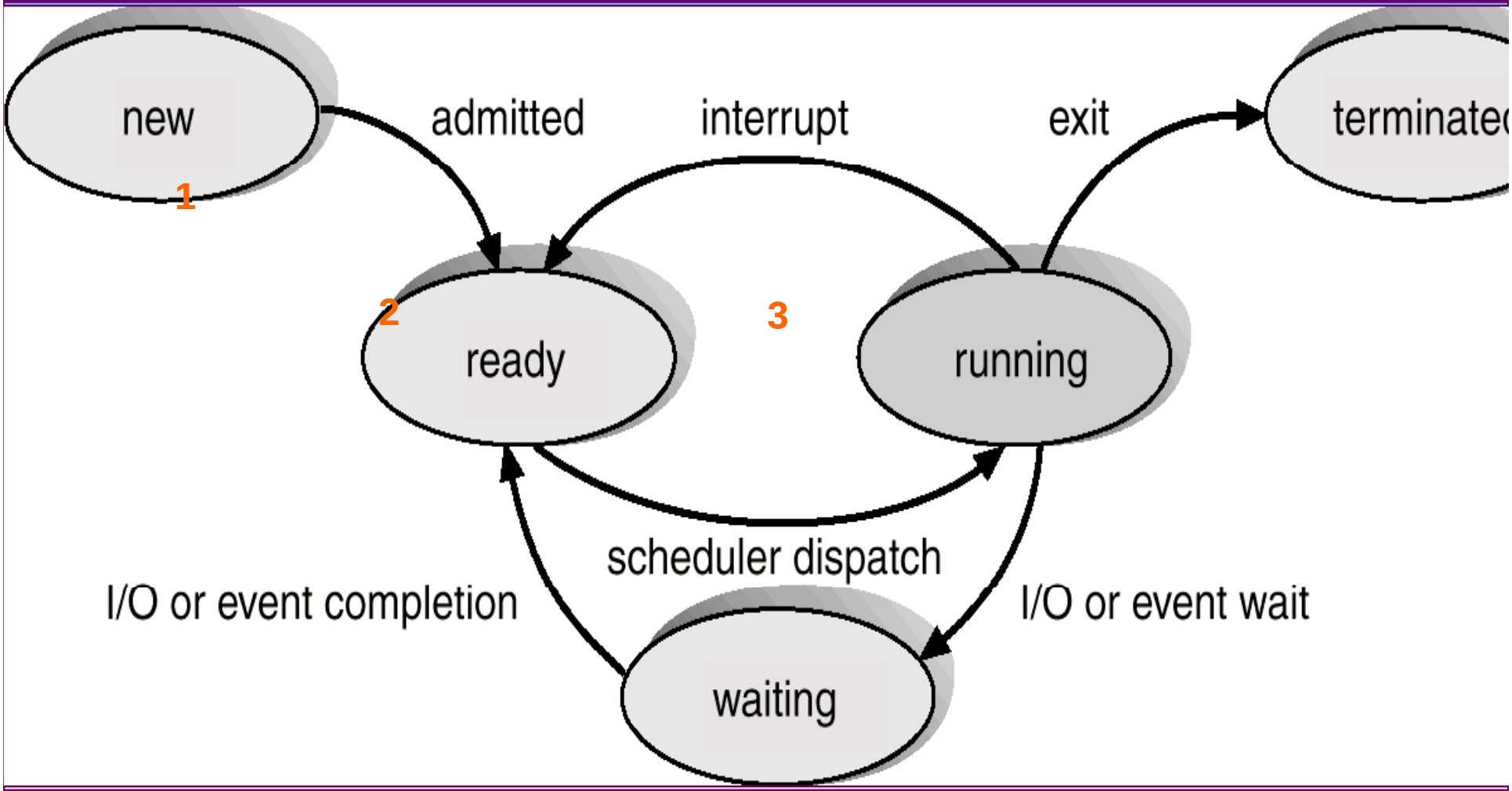
- **Processo** : un programma in esecuzione; l'esecuzione di un singolo processo avviene in maniera sequenziale.

- Un processo include:
  - sezione testo (codice),
  - il program counter,
  - lo stack,
  - la sezione dati.

Stato del Processo

- Durante la sua esecuzione un processo cambia il proprio ***stato*** che può essere:
  - **new**: Il processo viene creato.
  - **running**: Il processo (le sue istruzioni) è in esecuzione.
  - **waiting**: Il processo è in attesa di un dato evento.
  - **ready**: Il processo è pronto per essere eseguito.
  - **terminated**: Il processo ha completato la sua esecuzione.

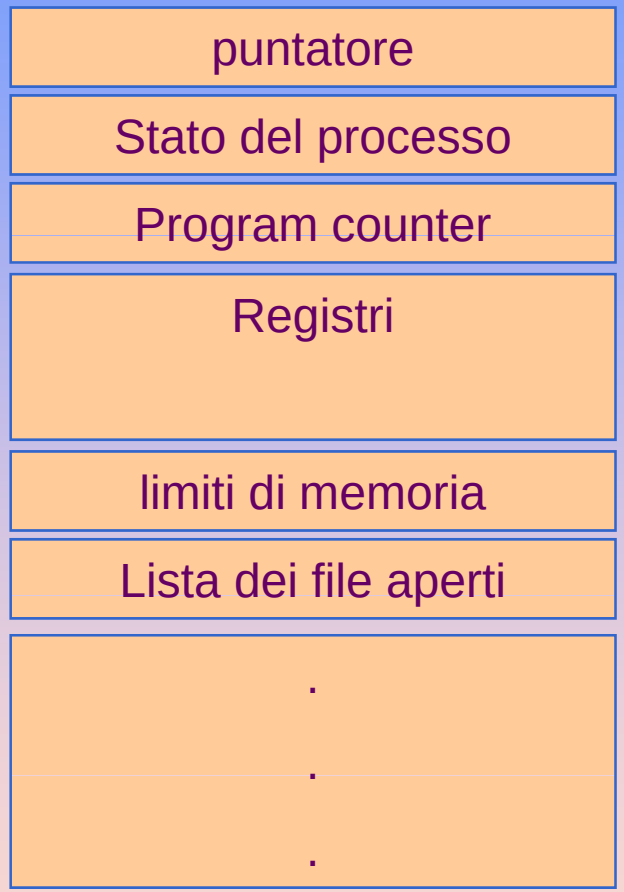
Diagramma di stato di un Processo



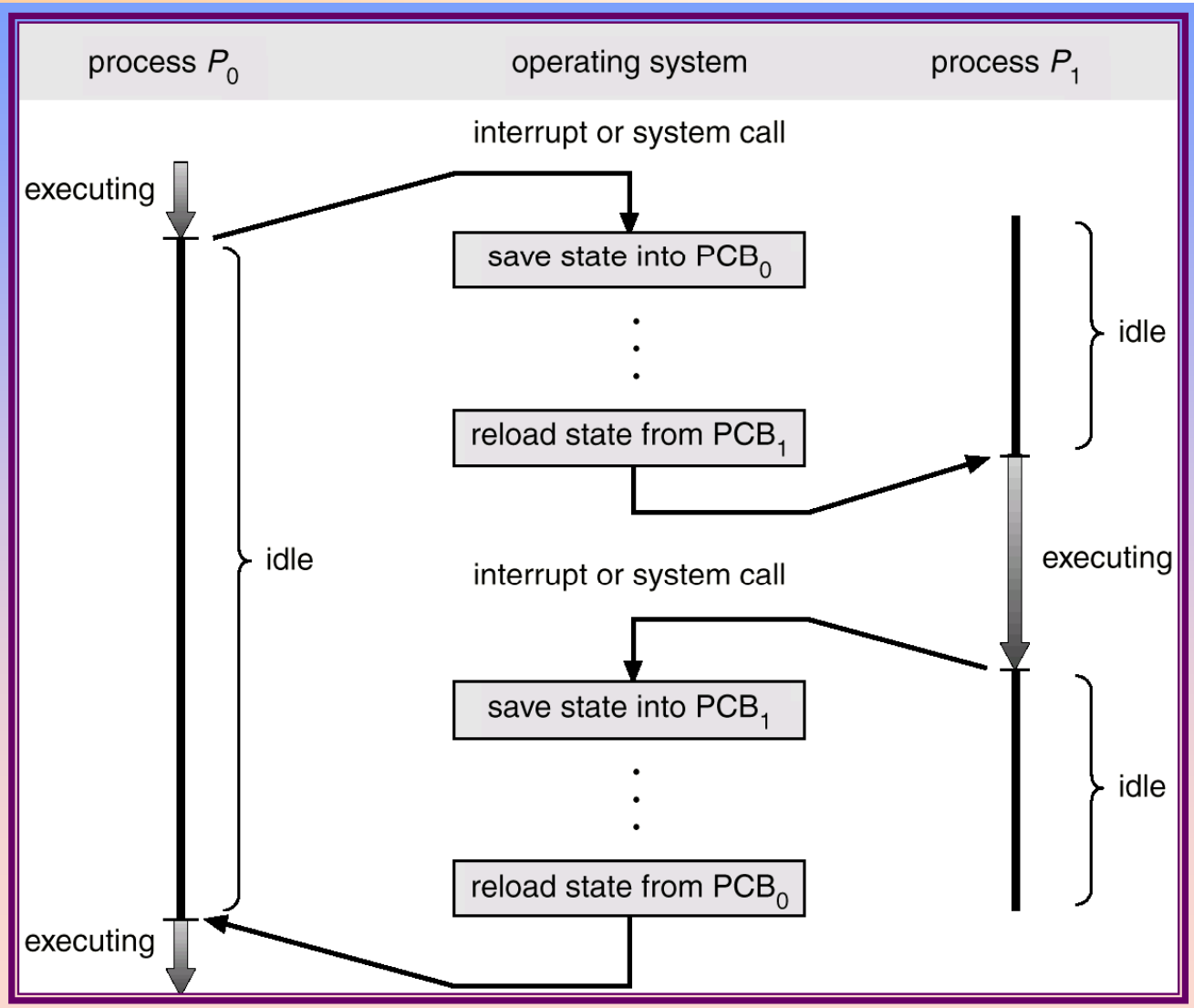
Process Control Block (PCB)

- Il PCB contiene l'informazione associata ad ogni processo:
  - stato del processo
  - program counter
  - registri della CPU
  - info sullo scheduling della CPU
  - informazioni di memory-management
  - informazioni di accounting
  - stato dell'I/O
  - ID del processo
  - ID dell'utente.

Process Control Block (PCB)



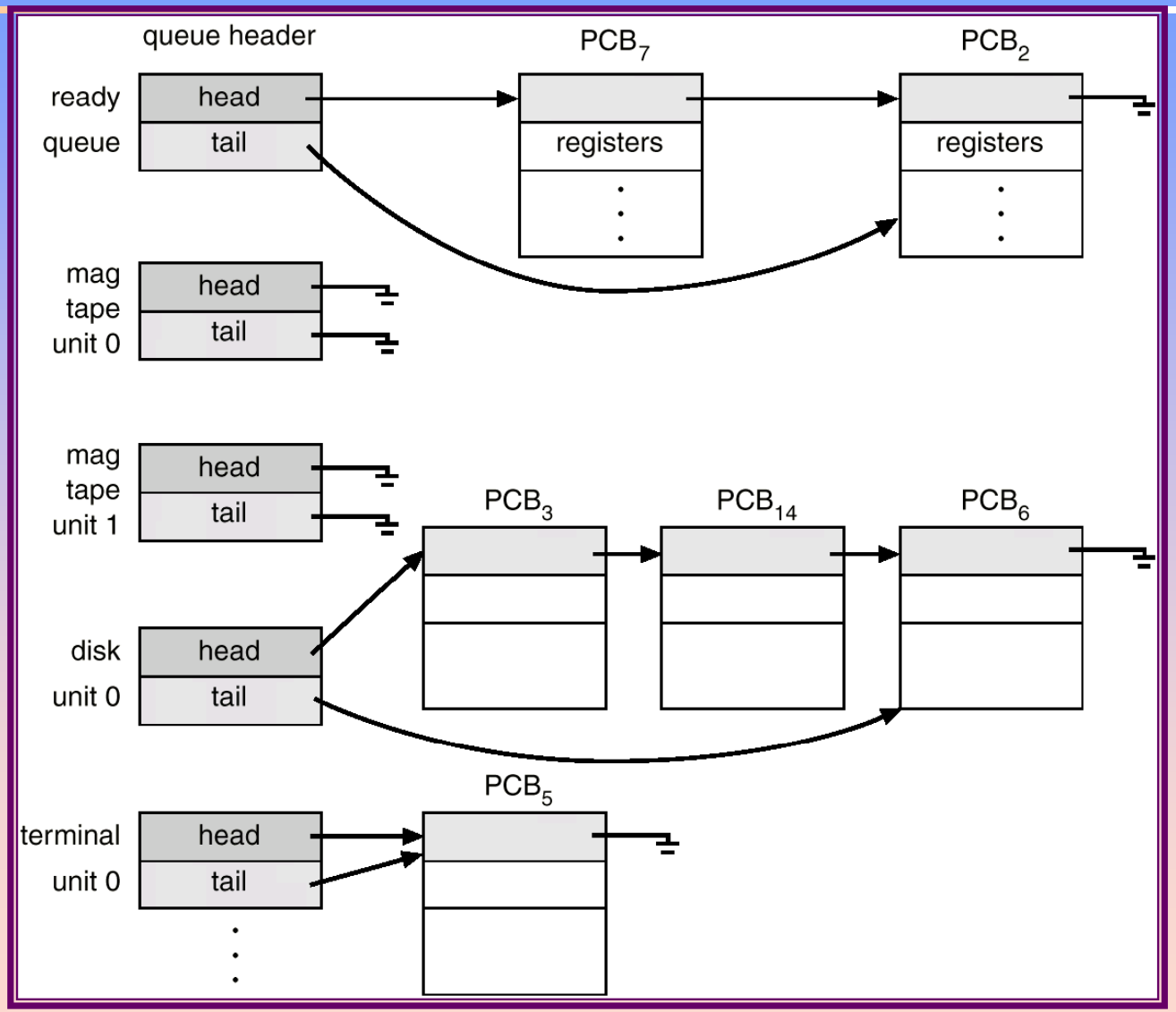
Commutazione della CPU da Processo a Processo



Code di Scheduling

- **Coda dei processi** – l'insieme di tutti i processi nel sistema.
- **Ready queue** – l'insieme dei processi in memoria centrale pronti per essere eseguiti.
- **Coda del dispositivo** – l'insieme dei processi in attesa di usare un dispositivo. (Più code)
- I processi passano da una coda all'altra mentre cambiano stato.

Ready Queue e code dei dispositivi di I/O



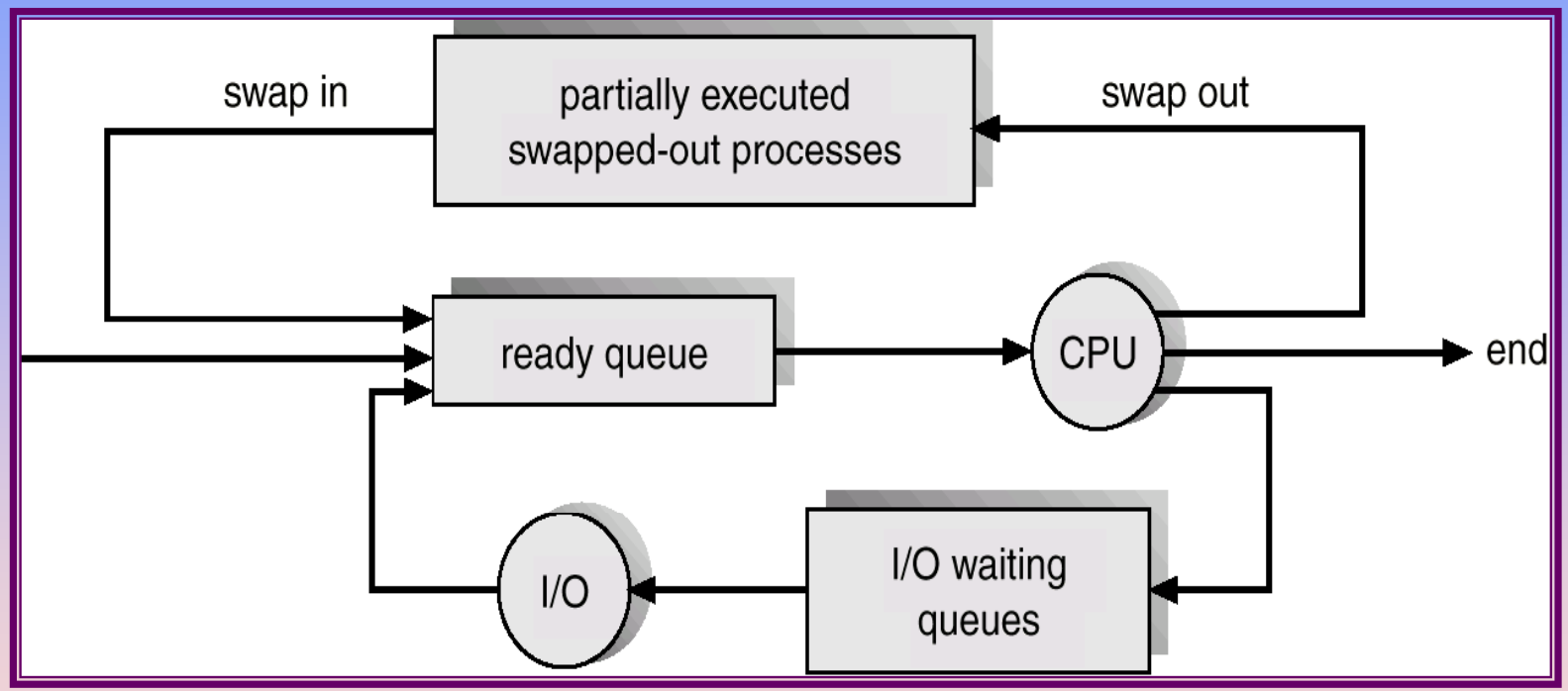
Scheduler

- In un sistema possono esistere più scheduler (es. sistemi batch):
- **Scheduler a lungo termine** (o **job scheduler**) – seleziona i processi da inserire nella *ready queue* (la coda dei processi pronti).
- **Scheduler a breve termine** (o **CPU scheduler**) – seleziona tra i processi pronti quelli che devono essere eseguiti.

Scheduler a medio termine

- In alcuni sistemi time-sharing esiste uno **scheduler a medio termine** che gestisce i processi pronti in memoria centrale. (**swapper**)
- In alcuni casi rimuove i processi dalla memoria (**swap-out**) per riportarli in memoria (**swap-in**) quando sarà possibile.
- Questo migliora l'utilizzo della memoria in caso di una alta richiesta di esecuzione di processi.

Scheduler a medio termine



Schedulers

- Lo scheduler a breve termine è invocato molto frequentemente (millisecondi) $\Rightarrow$ (deve essere veloce).
- Lo scheduler a lungo termine è invocato non molto spesso (secondi, minuti) $\Rightarrow$ (può essere lento).

- Lo scheduler a lungo termine controlla il grado di multi-programmazione (n° dei processi in memoria).

- I processi possono essere classificati come:
  - *processi* I/O-*bound* – basso uso della CPU e elevato uso dell'I/O.
  - *processi CPU-bound* – elevato uso della CPU e basso uso dell'I/O.

Context Switch

- **Context switch**: operazione di passaggio da un processo all'altro da parte della CPU.
- Il tempo impiegato per il context-switch time è un costo: il sistema non effettua lavoro utile per nessun processo utente.
- Il tempo di context switch dipende dal supporto offerto dall'hardware.

Operazioni sui Processi: Creazione

- Un processo qualsiasi può creare altri processi come suoi figli i quali possono creare altri processi, e cosi via.
- Il sistema operativo crea i processi utente come processi figli.
- Possibile condivisione di risorse:
  - processi padri e figli condividono tutte le risorse.
  - Un processo figlio condivide una parte delle risorse del padre.
  - Processi padri e figli non condividono risorse.
- Approcci di esecuzione:
  - Processi padri e figli eseguono concorrentemente.
  - Il padre rimane in attesa della terminazione dei figli.

Processi Indipendenti o Cooperanti

- I *processi indipendenti* non interagiscono con altri processi durante la loro esecuzione.

- *I processi cooperanti* influenzano o possono essere influenzati da altri processi. Il comportamento dipende anche dall'ambiente esterno.

- Vantaggi della cooperazione:
  - Condivisione dell'informazione
  - Velocità di esecuzione
  - Modularità
  - Convenienza.

Processi Cooperanti

- *I processi cooperanti* possono interagire tramite:
  - Scambio esplicito di dati,
  - Sincronizzazione su un particolare evento.
  - Condivisione dell'informazione.
- I sistemi operativi offrono meccanismi per realizzare queste diverse forme di cooperazione.
- Ad esempio:
  - send e receive
  - semafori
  - monitor
  - chiamata di procedura remota
- Alcuni linguaggi offrono anche meccanismi di cooperazione (es: Java).

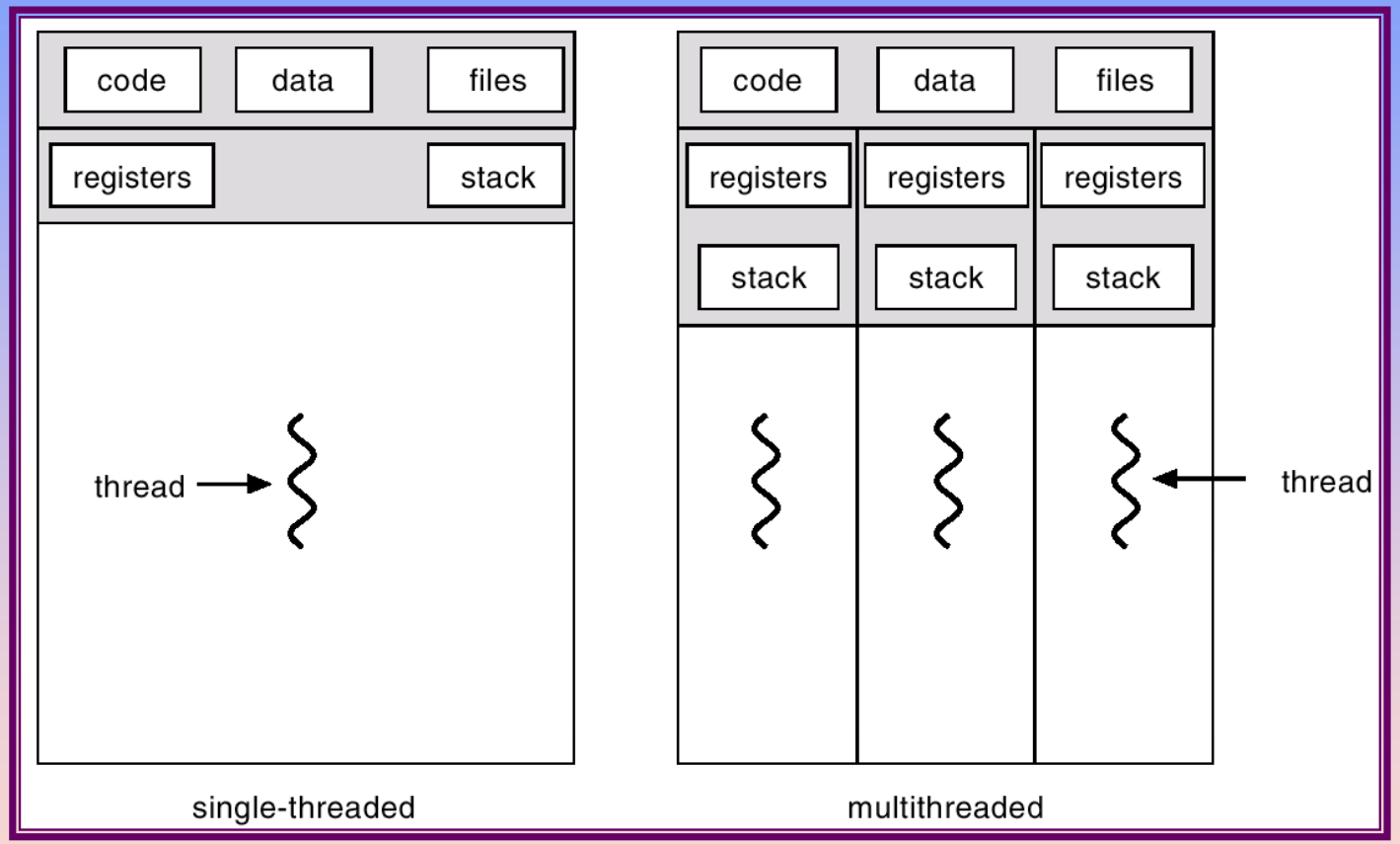
Thread

- Un **thread**, detto anche processo leggero, è una unità di esecuzione che consiste di un program counter, lo stack e un insieme di registi.
- Un thread condivide con altri thread la sezione codice, la sezione dati, e le risorse che servono per la loro esecuzione.
- Un insieme di thread associati prendono il nome di **task**.
- Un **processo** equivale ad un task con un unico thread.
- I thread rendono più efficiente l'esecuzione di attività che condividono lo stesso codice.

Thread

- Il context switch tra thread è molto più veloce.
- Un sistema operativo composto da thread è più efficiente.
- I thread non sono tra loro indipendenti perché condividono codice e dati.
- E’ necessario che le operazioni non generino conflitti tra i diversi thread di un task.

Task con thread singoli e multipli



Benefici

- Velocità di risposta
- Condivisione di risorse
- Economia
- Uso di architettura parallele (multiprocessore)

Thread in Java

- Tutti i programmi Java comprendono almeno un thread. Anche un programma costituito solo dal metodo main viene eseguito come un singolo thread. Inoltre, Java fornisce strumenti che consentono di creare e manipolare thread aggiuntivi nel programma

Esistono due modi per implementare thread in Java:

- Definire una sottoclasse della classe **Thread**.

- Definire una classe che implementa l'interfaccia **Runnable**. Questa modalità è più flessibile, in quanto consente di definire un thread che è sottoclasse di una classe diversa dalla classe Thread.

Definire una sottoclasse della classe Thread

1. Si definisce una nuova classe che estende la classe Thread. La nuova classe deve ridefinire il metodo ***run()*** della classe Thread.

2. Si crea un'istanza della sottoclasse tramite ***new***.

3. Si chiama il metodo ***start()*** sull'istanza creata. Questo determina l'invocazione del metodo run() dell'oggetto, e manda in esecuzione il Thread associato.

Esempio di sottoclasse di Thread

```
class Saluti extends Thread {
  public Saluti(String nome) {
    super(nome);
  }
  public void run() {
    for (int i = 0; i <= 10; i++)
      System.out.println("Ciao da "+getName());
  }
}
public class ThreadTest {
  public static void main(String args[]) {
    Saluti t = new Saluti(“Thread1");
    t.start();
  }
}
```

Definire una classe che implementa Runnable

1. **Si definisce una nuova classe che implementa l'interfaccia Runnable. La nuova classe deve implementare il metodo *run()* dell'interfaccia Runnable.**

2. **Si crea un'istanza della classe tramite *new*.**

3. **Si crea un'istanza della classe Thread, passando al suo costruttore un riferimento all'istanza della nuova classe definita.**

4. **Si chiama il metodo *start()* sull'istanza della classe Thread creata, determinando l'invocazione del metodo run() dell'oggetto Runnable associato.**

Esempio di classe che implementa Runnable

```
class Saluti implements Runnable {
  private String nome;
  public Saluti(String nome) {
    this.nome = nome;
  }
  public void run() {
    for (int i = 0; i <= 10; i++)
      System.out.println("Ciao da "+nome);
  }
}
public class RunnableTest {
  public static void main(String args[]) {
    Saluti s = new Saluti("Thread2");
    Thread t = new Thread (s);
    t.start();
  }
}
```

La classe Thread (1)

Costruttori principali:

Thread (): crea un nuovo oggetto Thread.

Thread (String name): crea un nuovo oggetto Thread con nome **name**.

Thread (Runnable target): crea un nuovo oggetto Thread a partire dall'oggetto **target**.

Thread (Runnable target, String name): crea un nuovo oggetto Thread con nome **name** a partire dall'oggetto **target**.

Metodi principali:

String getName(): restituisce il nome di questo Thread.

void join(): attende fino a quando questo Thread non termina l'esecuzione del proprio metodo run.

La classe Thread (2)

void join(long millis): attende, per un tempo massimo di **millis** millisecondi, fino a quando questo Thread non termina l'esecuzione del proprio metodo run.

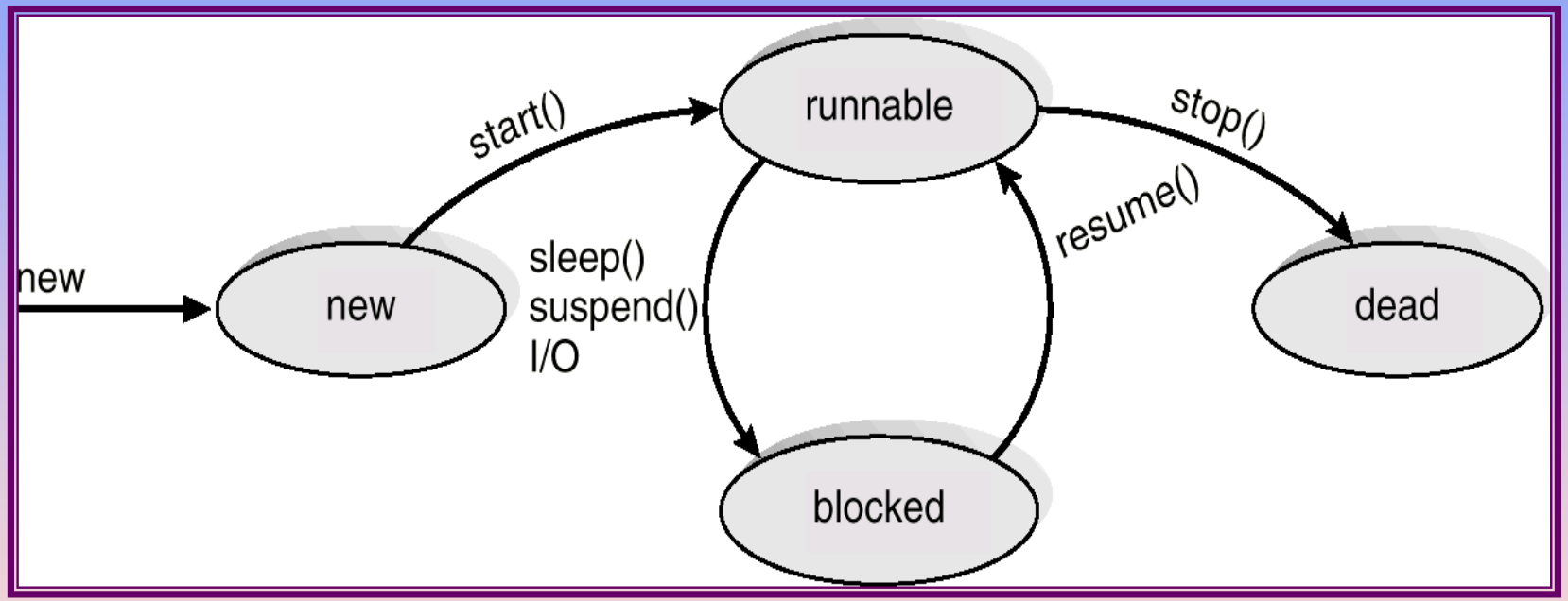
void run(): specifica le operazioni svolte dal Thread. Deve essere ridefinito dalla sottoclasse, altrimenti non effettua alcuna operazione. Se il Thread è stato costruito a partire da un oggetto Runnable, allora verrà invocato il metodo run di tale oggetto.

static void sleep(long millis): determina l'interruzione dell'esecuzione del Thread corrente per un tempo di **millis** millisecondi.

void start(): fa partire l'esecuzione del Thread. Viene invocato il metodo run di questo Thread.

static void yield(): determina l'interruzione temporanea del Thread corrente, e consente ad altri Thread di essere eseguiti.

Digramma di stato di un Thread Java



Domande

- Descrivere le variazioni di stato di un processo che effettua molte operazioni di I/O.
- Cos’è il context switch.
- Come sono gestiti i thread in Java.
- Elencare le differenza principali tra i processi e i thread.
- A cosa serve uno scheduler a medio termine e uno a lungo termine.