Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2017-18

Pietro Frasca

Lezione 4

Giovedì 12-10-2017

Struttura e organizzazione software dei sistemi operativi

- Un sistema operativo deve svolgere molti compiti complessi. Per tale motivo dovrebbe essere progettato in modo tale che, oltre a funzionare correttamente, il suo codice sia facilmente modificabile.
- Ad alto livello, la progettazione del sistema sarà influenzata dalla scelta dell'hardware e dal tipo di sistema: batch, time sharing, real time, singolo utente, multiutente, e così via. Al di là di questo più alto livello di progettazione, i requisiti possono essere molto più difficile da specificare.
- Per la definizione e la progettazione di un sistema operativo si ricorre ai principi generali che sono stati sviluppati nel campo dell'ingegneria del software.
- Nella fase di progettazione è molto importante suddividere le operazioni che il sistema operativo deve svolgere in ***meccanismi*** *(tecniche)* e ***criteri*** *(politiche o strategie).* I meccanismi stabiliscono in che modo deve essere eseguito qualche compito; i criteri, invece, determinano in che modo utilizzare i meccanismi.

- Ad esempio, nei sistemi multiprogrammati il sistema operativo, per commutare la CPU da un processo all'altro, esegue un insieme di operazioni detto *cambio di contesto*, che comprende il salvataggio dei registri della CPU del processo che lascia la CPU ed il caricamento dei registri del nuovo processo che andrà in esecuzione
- Le operazioni di schedulazione della CPU, stabiliscono i criteri con cui assegnare la CPU ad un nuovo processo. Ad esempio la schedulazione potrebbe basarsi su una politica FIFO, su una politica basata sulle priorità che i processi possiedono, o su altri criteri.
- I meccanismi dovrebbero essere progettati in modo tale che siano separati dai criteri. Questo consente, nel caso si cambino i criteri, di mantenere ancora validi i meccanismi.
- Se ad esempio si decidesse di modificare la politica di schedulazione da FIFO ad una politica più complessa, sarebbe ancora possibile utilizzare i meccanismi già esistenti, senza modificarli.

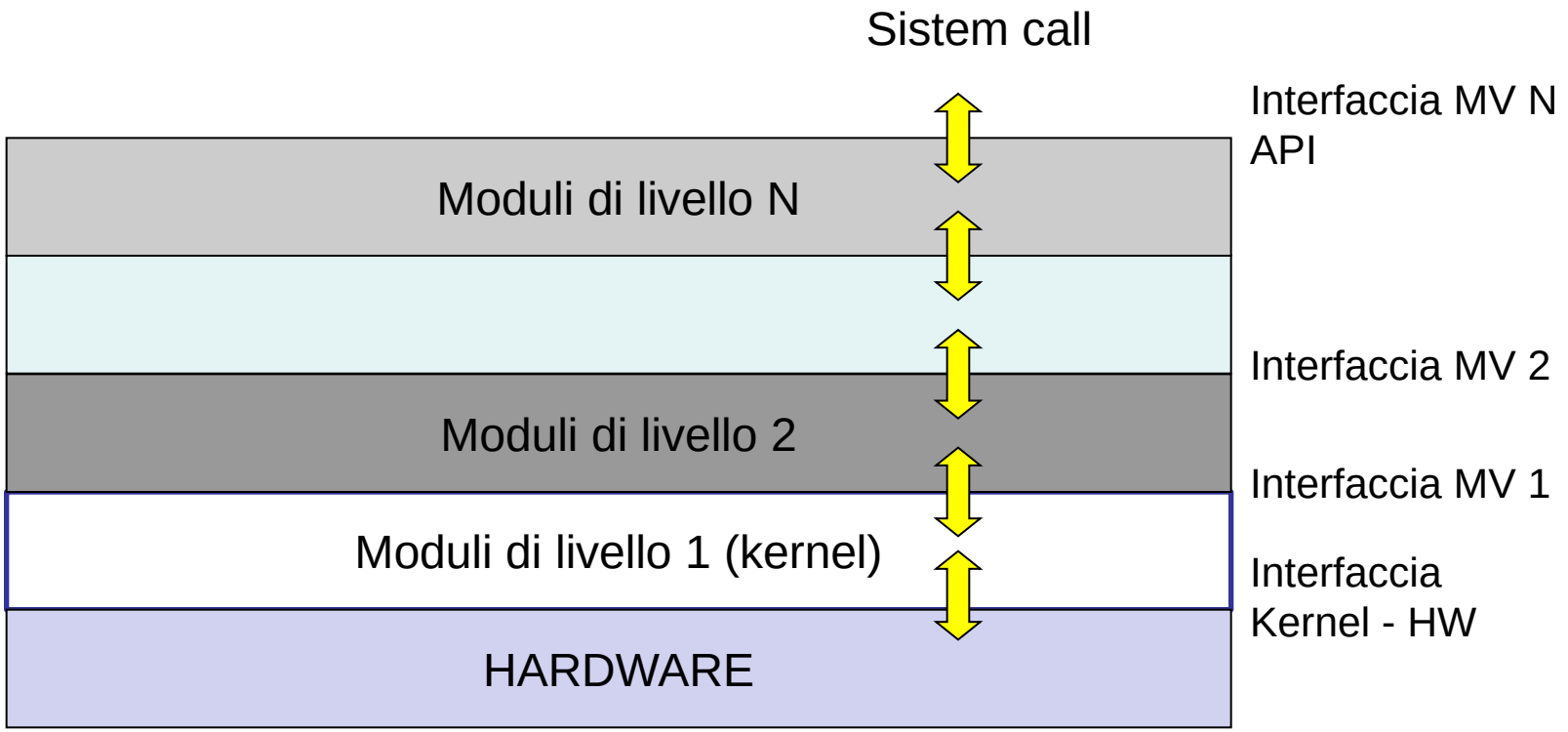
- Una volta progettato il sistema operativo si deve realizzare.
- La scrittura di un sistema operativo dipende fortemente dall'architettura del hardware ed in particolare dal processore o processori utilizzati nel calcolatore.
- Come già descritto, molti processori sono dotati di istruzioni che possono essere eseguite in *modalità privilegiata o in modalità utente*. Questa caratteristica consente di realizzare ed organizzare il software in modo tale che solo il codice del sistema operativo possa eseguire le istruzioni privilegiate, proteggendo, pertanto, le componenti del sistema operativo stesso da un uso improprio o errato da parte dei programmi applicativi.
- Generalmente, i sistemi operativi, si scrivono con linguaggi di alto livello, come ad esempio il C ed il C++, con ristrette parti in linguaggio assembly, per poter accedere ai registri dei dispositivi hardware e realizzare funzioni compatte e veloci.
- I kernel di Linux e Windows sono scritti prevalentemente in C, anche se ci sono alcune piccole parti di codice assembly nello scheduler e nei driver di dispositivi.

Struttura monolitica

- Una semplice organizzazione del software, detta ***struttura monolitica***, consiste nel realizzare un insieme di funzioni ciascuna delle quali implementa un determinato servizio, attivabile tramite una o più chiamate di sistema. Spesso queste funzioni si scrivono in linguaggio assembly, per poter avere la massima velocità di esecuzione e una minore dimensione in termini di occupazione di memoria RAM.
- Questa struttura, piuttosto semplice è stata usata nel sistema operativo Microsoft MS-DOS, un sistema operativo monoutente e mono tasking, scritto per microprocessori Intel 8088, 8086 e 80286, privi di modalità kernel. In assenza di modalità privilegiata, il programmatore può accedere a qualsiasi istruzione del microprocessore e quindi eseguire qualsiasi operazione come, ad esempio, scrivere dati in qualsiasi locazione di memoria, anche se riservata al sistema operativo. E' evidente che, in questi sistemi, un semplice errore di programmazione in un'applicazione può portare al crash del sistema.

Struttura stratificata

- Per la progettazione e lo sviluppo di sistemi più complessi si può ricorrere ai modelli e alle tecniche della programmazione strutturata o meglio ancora alla programmazione ad oggetti. I progettisti organizzano il sistema in un insieme di moduli, strutturandoli in vari livelli.
- Ciascun modulo di un livello utilizza le funzionalità offerte dai moduli di livello sottostante e fornisce a sua volta servizi ai moduli del livello superiore. Nei sistemi stratificati con il termine *kernel* si indica il livello che è a stretto contatto con l'hardware.



Struttura a livelli

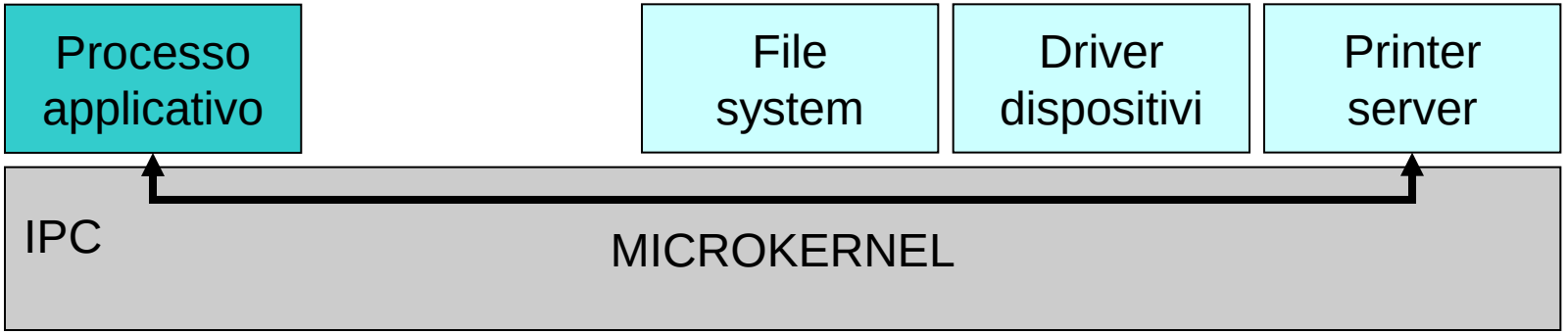
- La tecnica della stratificazione semplifica la fase di progettazione e rende più agevole apportare modifiche e correzioni al codice. Ogni strato può essere modificato, senza apportare cambiamenti ai restanti strati.
- Tuttavia, è richiesta un attenta e complessa analisi per stabilire quanti strati realizzare e scegliere quale funzionalità implementare in ciascun di essi.
- Inoltre, la stratificazione porta ad un funzionamento meno efficiente in termini di velocità di esecuzione e di occupazione di memoria. Ad esempio per eseguire un operazione, un programma applicativo, potrebbe effettuare una chiamata di sistema al livello sottostante, la quale, a sua volta, ne richiama un'altra, e questa un'altra ancora, e così. In altre parole, il programma applicativo per ottenere un servizio potrebbe attendere l'esecuzione di N funzioni di sistema.

- Bisogna tenere anche presente che nel passaggio da uno strato all'altro sono allocate strutture dati e parametri, con conseguente maggiore impegno di memoria. Per tale motivo, attualmente, si progettano sistemi stratificati con un limitato numero di strati.

Struttura a microkernel

- Come già detto, per proteggere le componenti del SO è necessario che il processore sia dotato di istruzioni eseguibili in stato privilegiato per garantire che solo il codice del SO possa girare in stato privilegiato.
- D'altra parte, il fatto che solo il SO possa eseguire istruzioni privilegiate, rende il sistema più difficile da modificare.
- Per rendere più semplice, in particolare, il cambiamento delle politiche di gestione delle risorse è stata pensata la struttura a *microkernel*.

- Per gestire una risorsa sono definite due tipi di componenti del SO: le tecniche (meccanismi) per consentire la gestione della risorsa; Le strategie di gestione (politiche) realizzate utilizzando i precedenti meccanismi.
- Nei sistemi a struttura a microkernel l'insieme dei meccanismi costituisce il microkernel, che è l'unico componente a girare nello stato privilegiato. Tutte le strategie sono implementate in programmi applicativi che girano nella modalità utente (non privilegiata). In tal modo questi programmi sono più facilmente modificabili ed espandibili. Quando un processo applicativo richiede una risorsa, interagisce con il relativo processo server mediante un insieme di chiamate di sistema detto ***IPC (Inter Process Comunication)***, fornite dal microkernel, che consentono la comunicazione tra processi.



La struttura a microkernel produce una perdita di efficienza, poiché per ogni operazione di comunicazione tra processi è necessario l’uso di chiamate di sistema.

Struttura modulare

- Attualmente, probabilmente la migliore tecnica per progettare e realizzare i sistemi operativi complessi si basa sulla programmazione orientata agli eventi e agli oggetti.
- Con tale tecnica si sviluppa il sistema in moduli, ciascuno dei quali svolge un particolare compito.
- Ogni modulo è implementato da un insieme di funzioni descritte da un'*interfaccia* che descrive le funzionalità svolte dal modulo e da un *corpo* che consiste nel codice che implementa le funzioni descritte nell'interfaccia.

Interfaccia Definizione di un insieme di funzioni implementate nel modulo
Corpo Implementazione delle funzionalità (nascoste all'esterno del modulo)

- Il codice del corpo di un modulo è nascosto al resto del sistema e si comunica con esso solo attraverso le funzioni della propria interfaccia. Il kernel si realizza in base ad un numero di componenti fondamentali, ai quali, se richiesto, se ne aggiungono altri dinamicamente durante la procedura di avvio o durante l'esecuzione. La tecnica di caricare dinamicamente moduli è attualmente usata dai sistemi operativi come ad esempio Windows, Linux, Solaris e Max OS X.

Struttura ibrida

- In realtà, pochi sistemi operativi adottano una struttura unica. Invece, generalmente, l'architettura è ibrida, costituita da una combinazione di diverse strutture, in modo da risolvere i problemi di prestazioni, sicurezza e usabilità.
- Ad esempio, Linux e altri sistemi Unix sono monolitici, perché avere il kernel in un unico spazio di indirizzamento offre prestazioni molto efficienti. Tuttavia, sono anche modulari, poiché nuove funzionalità possono essere aggiunte dinamicamente al kernel.
- Windows è in gran parte monolitico, soprattutto per motivi di prestazioni, ma conserva alcuni comportamenti tipici dei sistemi a microkernel, compreso il supporto per sottosistemi come win32 e POSIX, che sono eseguiti come processi in modalità utente. I sistemi Windows hanno anche il supporto per i moduli del kernel caricabili dinamicamente.

Principali componenti del SO

- Da quanto fino ad ora descritto, a grandi linee, possiamo distinguere i SO nelle seguenti classi principali, in base al tipo di applicazioni per cui sono stati progettati:
  - **Sistemi operativi batch** per applicazioni di calcolo intensivo (applicazioni CPU-bound) con l'obiettivo di ottimizzare l'efficienza d'uso delle risorse.
  - **Sistemi operativi time-sharing** per applicazioni interattive (I/O-bound) con l'obiettivo di minimizzare i tempi medi di risposta.
  - **Sistemi operativi real-time** per applicazioni di controllo, con l'obiettivo di rispettare tutti i loro vincoli temporali (deadline).
  - **Sistemi operativi per personal computer** per l'uso di un singolo utente al fine di massimizzare la semplicità d'uso e le prestazioni.
  - **Sistemi operativi distribuiti e di rete** per applicazioni distribuite con l'obiettivo di condividere risorse.

- Pur avendo architetture e obiettivi diversi, le suddette classi di SO hanno molte caratteristiche comuni.
- In particolare esamineremo le componenti di un SO che svolgono le seguenti funzioni fondamentali:.
  - **Gestione dei processi (scheduler)**.
  - **Gestione della memoria**.
  - **Gestione dell'I/O**
  - **Gestione del file system**
  - **Protezione e sicurezza**
  - Comunicazione su rete

Gestione dei processi

- I sistemi operativi multiprogrammati consentono l'esecuzione di più programmi in modo concorrente.
- Un sistema è costituito da un insieme di programmi che condividono nel tempo l'uso del processore o di più processori nel caso di architetture multiprocessore.
- Un compito importante di un sistema operativo è pianificare e fare avanzare l'esecuzione dei programmi utente.
- Tuttavia, per via della sua elevata complessità, un sistema operativo è realizzato in un elevato numero di processi di sistema ciascuno dei quali fornisce ai programmi applicativi determinati servizi. Quando un computer è acceso, si avvia per primo il kernel, e anche se gli utenti del sistema non hanno avviato alcun programma, al termine del caricamento sono presenti decine o centinaia di processi. Si tratta di processi di sistema che insieme al kernel consentono al computer di funzionare.
- Tanto più è complesso un sistema operativo, tanto sono più numerosi i processi di sistema.

- I processi di sistema nei sistemi UNIX sono spesso indicati con il nome *daemon* mentre nei sistemi Windows si identificano con il termine *service (servizi)*.

Concetto di processo

- I termini programma e processo sono spesso usati come sinonimi. Informalmente, finora, si è definito un processo come un programma in esecuzione.
- Il concetto di processo è legato alla multiprogrammazione. In un sistema multitasking molti programmi sono caricati in memoria pronti per essere eseguiti. La CPU passa rapidamente ad eseguire istruzioni di un programma a istruzioni di un altro programmi. Quando la CPU sospende di eseguire le istruzioni di un programma, è necessario salvare il valore dei suoi registri, in modo che il programma interrotto possa, in seguito, riprendere la sua esecuzione come se l'interruzione non fosse avvenuta.

- Pertanto, un processo non si identifica soltanto con il codice di un programma ma comprende anche l'attività del processore rappresentata, istante per istante, dai valori contenuti nei suoi registri.
- Generalmente, un processo include anche una ***sezione dati*** per memorizzare le variabili globali e statiche del processo e lo ***stack***, che contiene i dati temporanei, come ad esempio i parametri e le variabili locali di funzione. Può comprendere anche un ***heap*** nel caso in cui, durante l'esecuzione, il processo allochi memoria dinamicamente.
- Per tener traccia del ciclo di vita di un processo, tutte le sue proprietà sono mantenute in una struttura dati, il PCB (Process Controll Box).
- L'insieme di queste componenti è detta ***immagine*** **del processo**.

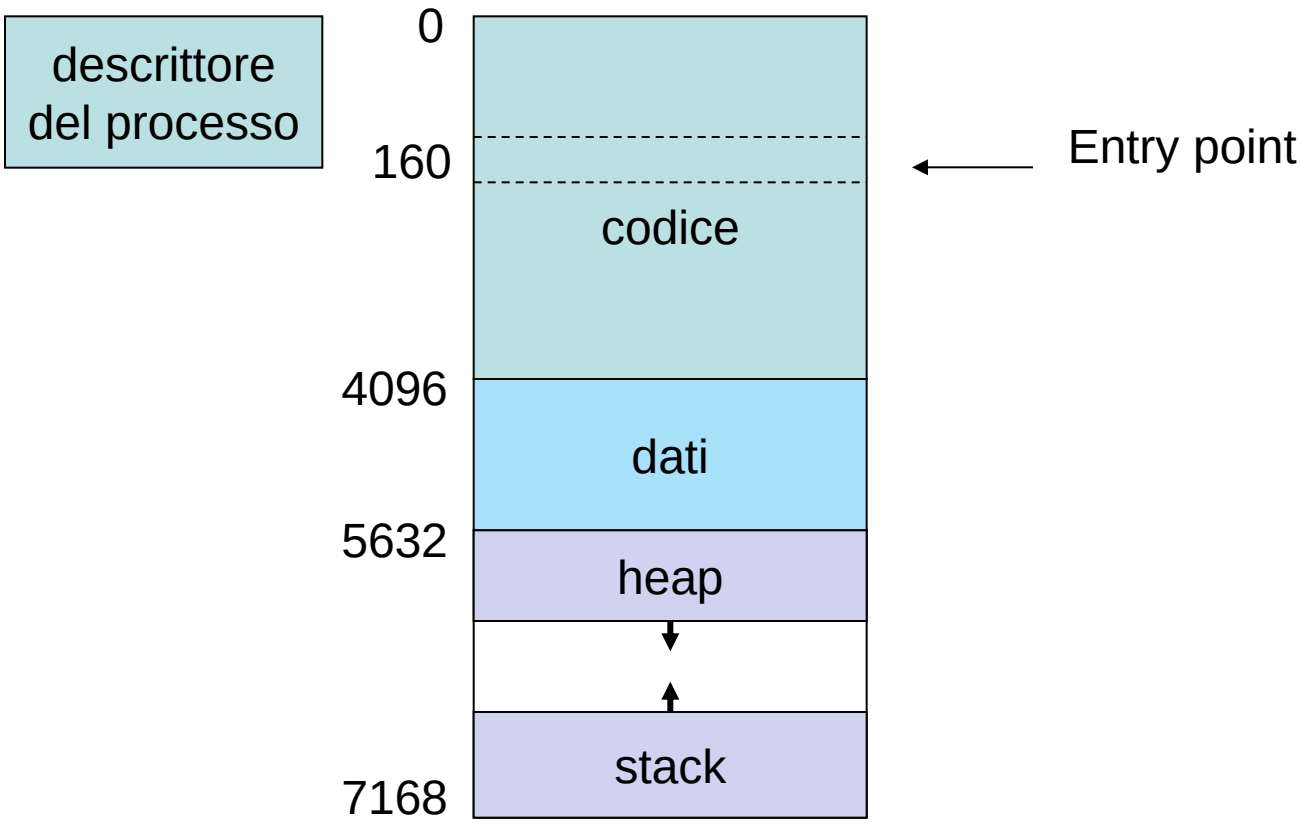


Immagine di un processo

Stati di un processo

- Un processo durante la sua esecuzione, esegue transizioni di stato che sono causate sia dall'esecuzione delle istruzioni del processo stesso, come ad esempio quando esso esegue operazioni di I/O, sia da eventi esterni asincroni con la sua esecuzione, come ad esempio la ricezione di segnali.
- Un processo entra nello ***stato nuovo*** quando è creato, ad esempio quando si avvia un programma.
- Un processo passa dallo stato nuovo allo ***stato di pronto*** dopo che il SO ha verificato che esso può essere effettivamente eseguito (ad esempio il processo ha i diritti di esecuzione).
- Un processo passa dallo stato di **pronto** allo stato di **esecuzione** quando ad esso è assegnato il processore.
- Un processo è nello **stato attivo** quando si trova nello stato di **pronto** o di **esecuzione**.
- Un processo passa dallo **stato di esecuzione** allo **stato di bloccato** quando è in attesa di qualche evento.

- Il processo ritorna nello stato di pronto quando l'evento atteso si è verificato.
- La transizione dallo stato di esecuzione allo stato di pronto è chiamato **prerilascio** (preemption) o **revoca**.
- Il prerilascio può avvenire per vari motivi, come ad esempio, quando un processo ha esaurito il suo quanto di tempo (nei sistemi time-sharing), o è presente nella coda di pronto un altro processo con priorità più alta.
- Il passaggio dallo stato di pronto allo stato di esecuzione è gestito dallo scheduler, un componente del kernel che seleziona un processo cui assegnare la CPU, tra tutti i processi che si trovano nello stato di pronto. Lo **scheduler** ha il compito di garantire che tutti i processi pronti possano avanzare nella loro esecuzione.
- Il processo passa nello stato **terminato** quando ha terminato l'esecuzione del suo programma o quando si è verificato un'eccezione di vario tipo.

Transizioni di stato

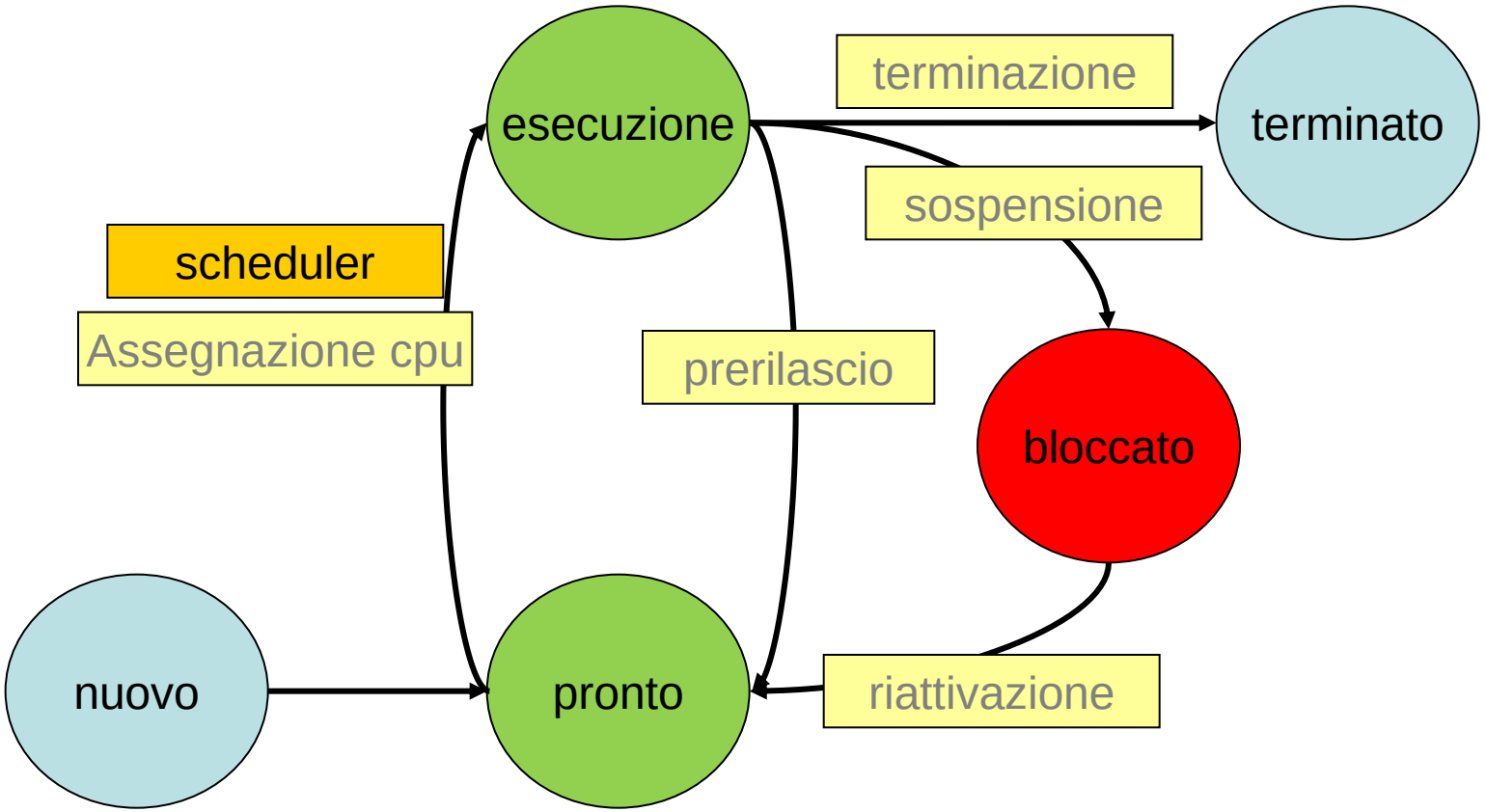


Diagramma di transizione a 5 stati

- In molti sistemi è previsto che un processo sia trasferito dalla memoria principale alla memoria secondaria (disco) in modo da ottenere spazio per altri processi. Tale operazione prende il nome di ***swapping (scambio)*** e lo stato relativo è detto ***swapped***.
- Vedremo un diagramma di transizione di stato con lo stato swapped quando parleremo della gestione della memoria.