

Gestione dei processi

- 2.1** Definizione di processo
- 2.2** Stati di un processo
- 2.3** Rappresentazione dei processi
- 2.4** Cambio di contesto
- 2.5** Scheduling
- 2.6** Operazioni sui processi
- 2.7** Processi leggeri

2.1 Definizione di processo

- Informalmente, il termine processo viene utilizzato per indicare un programma in esecuzione.
- È l'unità di esecuzione all'interno del S.O.
 - processi sequenziali
 - un S.O. multiprogrammato consente l'esecuzione concorrente di più processi

Istanze di un processo

- Più processi possono essere associati allo stesso programma (*istanza*): ciascuno rappresenta l'esecuzione dello stesso codice con dati di ingresso diversi

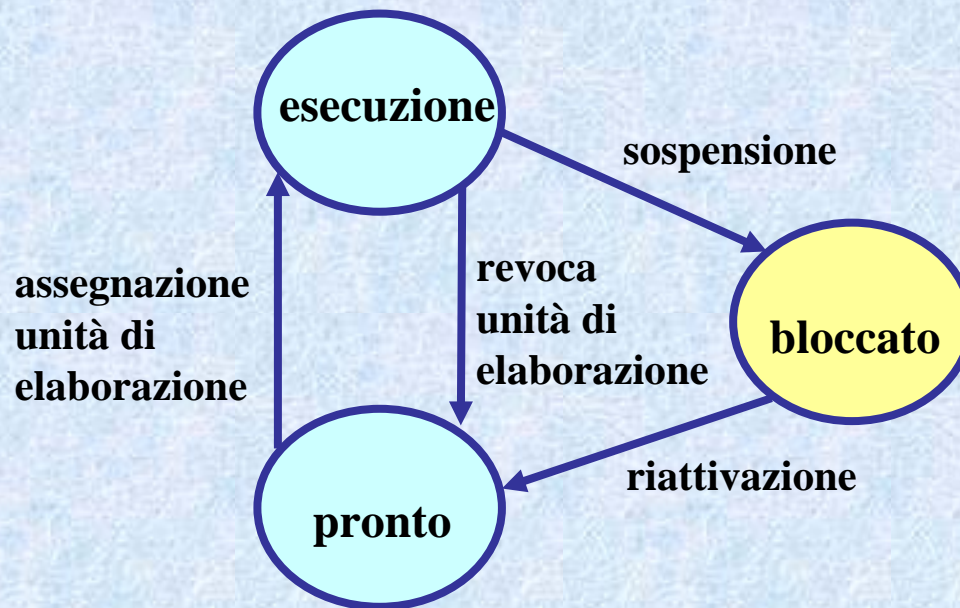
Rappresentazione del processo

- Un processo è rappresentato da:
 - Codice
 - Dati
 - Program Counter
 - Registri
 - Stack
- Ad un processo possono essere associate delle risorse:
 - Memoria
 - Files aperti
 - Dispositivi di I/O

2.2 Stati di un processo



- Se il numero di CPU è minore del numero dei processi (es. sistema monoprocessore), lo stato attivo si suddivide in due stati:
 - **pronto** (in attesa di una CPU)
 - **in esecuzione** (utilizza la CPU)



- Altri stati:

- **nuovo** (creazione di un nuovo processo)
- **terminato** (terminazione del processo)
- **in memoria secondaria**



2.3 Rappresentazione dei processi

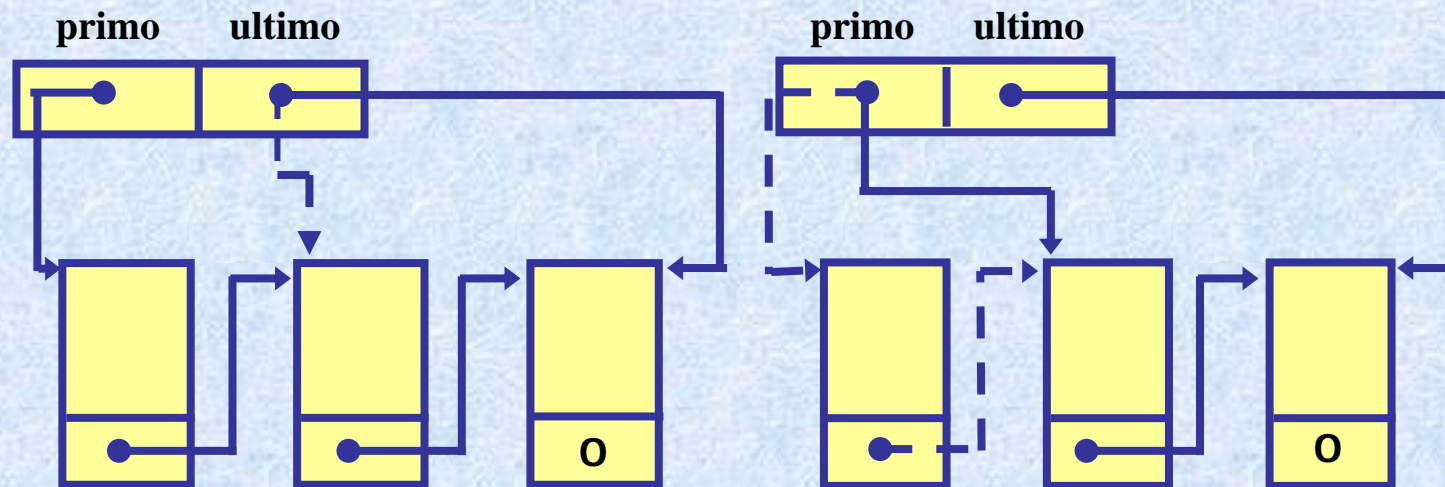
- **Descrittore del processo (Process Control Block, PCB):** struttura dati associata ad ogni processo.
- **Tabella dei processi:** contiene i descrittori di tutti i processi

Descrittore di un processo

- Nome del processo
- Stato del processo
- Contesto del processo
- Modalità di servizio dei processi
 - Priorità, parametri di scheduling
 - Scadenze temporali per sistemi in tempo reale
- Gestione della memoria
- Utilizzo delle risorse
- Processo successivo

Code di processi

- Code di processi pronti
- Code di processi bloccati
- Registro del processo in esecuzione



2.5 Scheduling dei processi

- È l'attività mediante la quale il sistema operativo effettua delle scelte tra i processi, riguardo a:

assegnazione del processore

- revoca del processore

=>> scheduling a breve termine

ma anche (eventualmente) :

- inclusione nell'insieme dei processi schedulabili
- temporanea esclusione da questo insieme

=>> scheduling a lungo e medio termine

Tre diverse attività di scheduling:

– Scheduling a breve termine

- Sceglie tra i processi pronti quello a cui assegnare la CPU
- Interviene quando il processo in esecuzione perde il controllo della CPU:
 - non preemptive scheduling
 - preemptive scheduling

– Scheduling a medio termine (Swapping)

- trasferimento temporaneo in memoria secondaria di processi
- Memoria principale inferiore alla somma delle richieste dei vari processi

– Scheduling a lungo termine

- Sceglie nella memoria secondaria quali programmi caricare in memoria centrale
- Controlla il grado di multiprogrammazione
- È una componente importante dei sistemi batch multiprogrammati

2.6 Operazioni sui processi

- Meccanismi per la gestione dei processi:
 - Creazione
 - Terminazione
 - Interazione tra processi

➔ **Chiamate di sistema**

Generazione di processi

- Processi permanenti (creati)
- Processi temporanei (generati)

Meccanismi per la generazione dei processi:

- Un processo (padre) può richiedere la generazione di nuovi processi (figli)
- *mediante chiamate di sistema*

esempio: nei sistemi interattivi *Login* (processo padre, permanente) genera processi figli per esecuzione di programmi applicativi

=>> gerarchie di processi

Dopo la generazione, il processo padre può passare in stato di attesa della terminazione del figlio, oppure evolversi concorrentemente ad esso

Terminazione di processi

- Un processo termina :
 - Quando esegue una *chiamata di sistema* per la propria terminazione
 - Quando effettua una operazione illecita
 - es. cerca di accedere alla memoria privata di altri processi
 - Quando c'è un errore che non gli permette di proseguire (es. overflow, etc)
- In tutti questi casi il processore ricomincia automaticamente ad eseguire il sistema operativo ad un indirizzo prefissato

Terminazione di processi

- Ogni processo temporaneo:
 - è figlio di un altro processo
 - può essere a sua volta padre di altri processi
- Il sistema operativo mantiene le informazioni relative alle relazioni parentali (nel descrittore)
- Se un processo termina:
 - Deve essere rilevato il suo stato di terminazione
UNIX: lo fa il padre con una chiamata di sistema
 - I figli possono restare in vita dopo la terminazione del padre
chi rileva lo stato di terminazione dei figli?

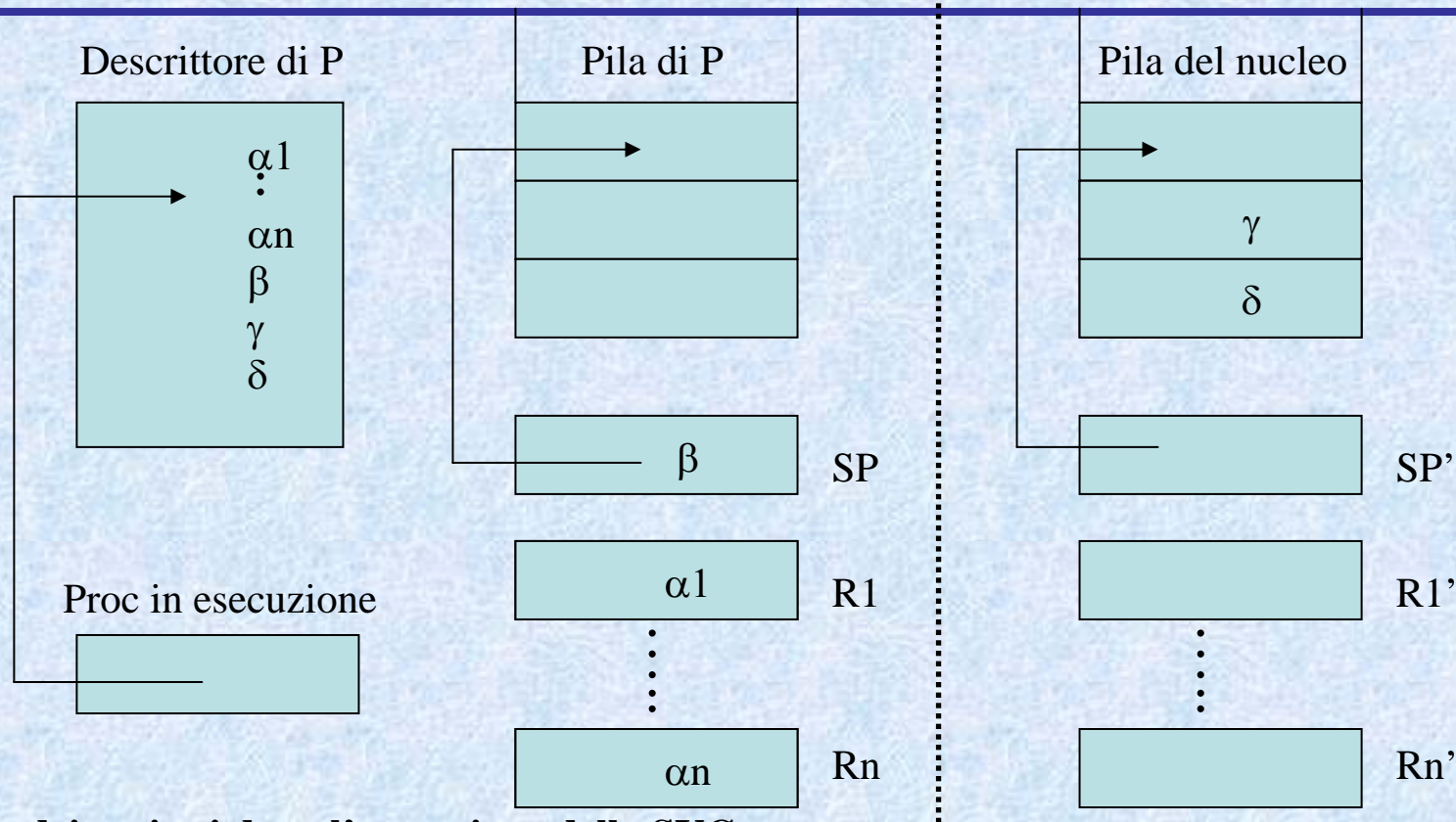
2.4 Cambio di contesto

- L'uso della CPU viene commutato da un processo ad un altro:
 - Salvataggio del contesto del processo in esecuzione nel suo descrittore (**salvataggio stato**)
 - Inserimento del descrittore nella coda dei processi bloccati o pronti
 - Selezione di un altro processo dalla coda dei processi pronti e caricamento del suo nome nel registro processo in esecuzione (**short term scheduling**)
 - Caricamento del contesto del nuovo processo nei registri del processore (**ripristino stato**)

Cambio di contesto

Overhead

- Salvataggio e ripristino dei registri
- Inserimento del descrittore in una coda; selezione ed estrazione del descrittore del processo da mandare in esecuzione (short-term scheduling)
- Invalidazione e ricostituzione della cache
- Operazioni indotte sul gestore della memoria (eventuale swap-out e swap-in, picco di eccezioni di indirizzo e page-faults nei sistemi con paginazione)



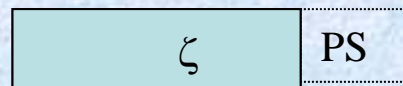
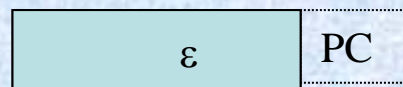
Stato dei registri dopo l'esecuzione della SVC

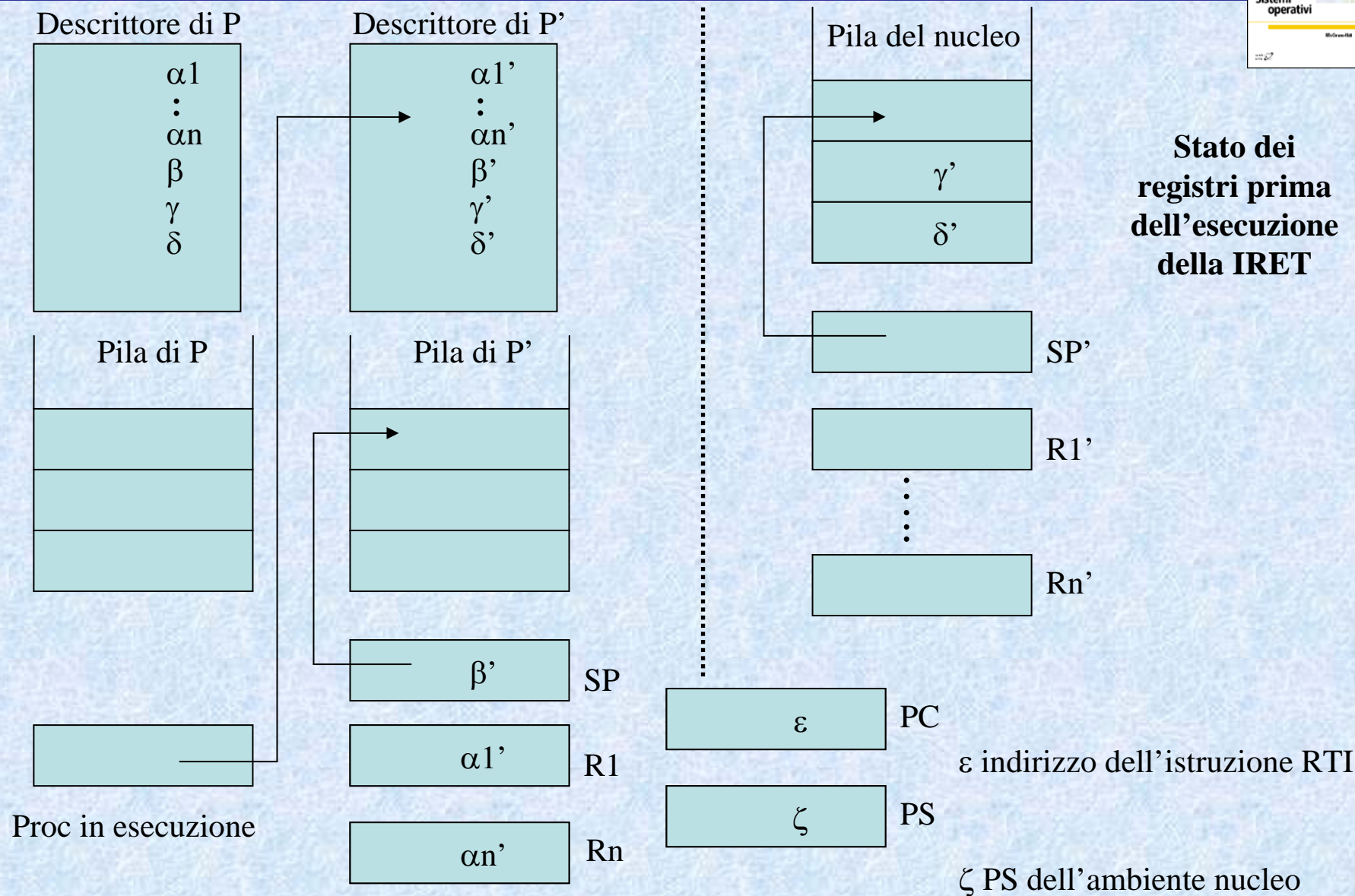
γ PC del proc che ha eseguito SVC

δ PS del proc che ha eseguito SVC

ϵ indirizzo della procedura di servizio

ζ PS dell'ambiente nucleo





Modello a processi

Modello a processi con ambiente globale

- i processi possono condividere dati
(esiste memoria comune)

Modello a processi con ambiente locale

- i processi non possono condividere dati
(non esiste memoria comune)

Modello a processi con ambiente locale

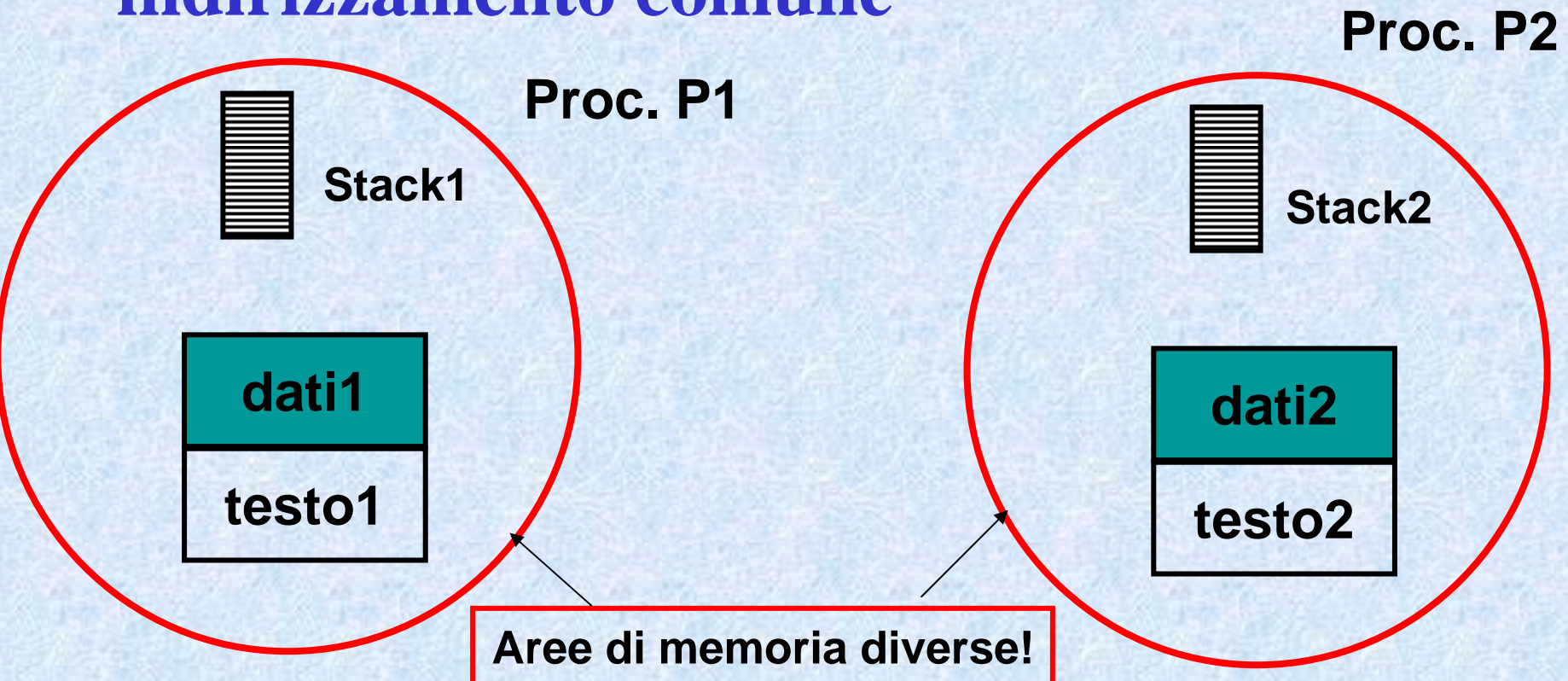
- Nel modello a processi con ambiente locale, ogni processo ha il suo spazio di indirizzamento privato e le sue interazioni avvengono utilizzando i meccanismi di IPC messi a disposizione dal kernel
 - es : send, receive

Questo implica alti costi di interazione:

- costo delle primitive di comunicazione
- costo delle commutazioni di contesto

Modello a processi ad ambiente locale

- Due processi diversi non hanno spazio di indirizzamento comune



Flussi di esecuzione

Una stessa applicazione può contenere più flussi di esecuzione

Esempio: flussi in un editor di testi

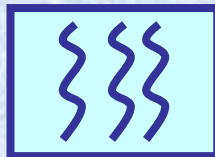
- *Immissione da tastiera*
- *Gestione della struttura dati del testo*
- *Visualizzazione sul terminale*
- *Controllo ortografico*
- *Salvataggio periodico nell'archivio*
- **La realizzazione mediante un processo per ogni flusso comporta frequenti interazioni con conseguenti commutazioni di contesto.**
- **Il tempo di permanenza in esecuzione di un processo può essere breve rispetto a quello necessario per la commutazione di contesto.**

Processi pesanti e processi leggeri

- Se a ogni flusso di esecuzione corrisponde un processo, un processo è al tempo stesso:
 - un elemento che possiede risorse
 - un elemento cui viene assegnata la CPU (unit di schedulazione)
- Separazione dei due aspetti:
 - **Processo pesante (task):** elemento che possiede le risorse
 - **Processo leggero (thread):** elemento cui viene assegnata la CPU

Thread

- Un thread rappresenta un flusso di esecuzione all'interno di un processo pesante
 - I thread di un processo pesante condividono lo stesso spazio di indirizzamento
- **Multithreading:** molteplicità di flussi di esecuzione all'interno di un processo pesante



più thread per processo

Multithreading

Esempio: editore di testi con multithreading

Processo: possiede le risorse

- Memoria
- Archivio (memoria secondaria)
- Terminale

Treads: corrispondono ai flussi di esecuzione

- Immissione da tastiera
- Gestione della struttura dati del testo
- Visualizzazione sul terminale
- Controllo ortografico
- Salvataggio periodico nell'archivio

Thread

- Thread definiti in un processo
- Ne condividono le risorse
- Ogni thread ha il proprio stack ed il proprio descrittore.
- **Processo= (Thread1, Thread2, ..., codice, dati; risorse di memoria, altre risorse)**
- **Thread= (PC, registri, stato, stack, descrittore)**

Thread

Proprietà dei processi:

- Spazi di indirizzamento separati e protetti
- Interazioni implicano chiamate di sistema e cambi di contesto
- Generazione e terminazione complesse: implicano anche l'assegnazione e il rilascio di risorse

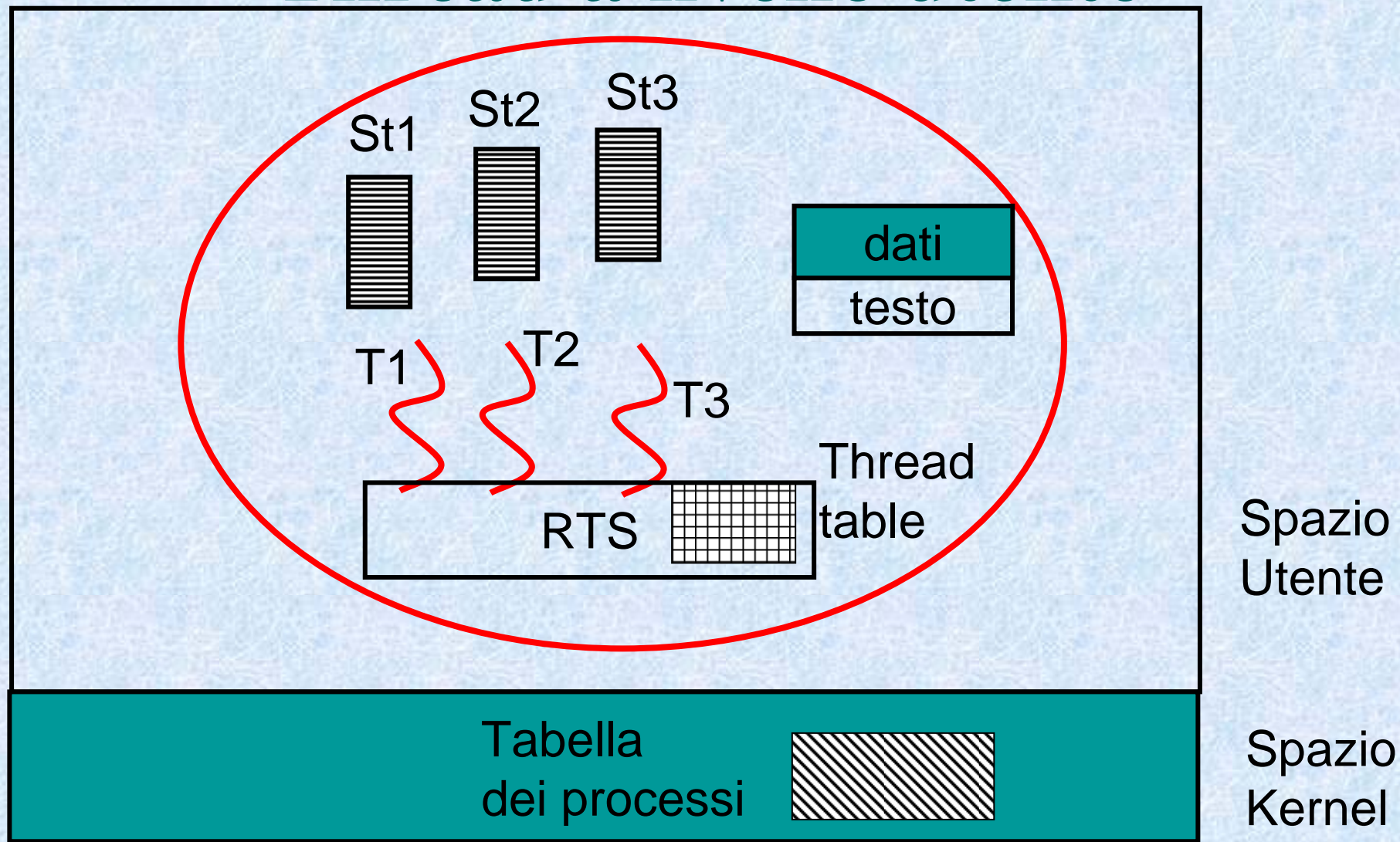
Proprietà dei thread:

- Condividono lo spazio di indirizzamento del processo cui appartengono
- Interazioni con modalità semplificate
- Generazione e terminazione non implicano assegnazione e rilascio di risorse

Thread a livello utente

- *Thread package*
 - libreria realizzata in spazio utente.
- Il passaggio da un thread ad un altro non richiede il supporto del S.O.
 - Lo scheduling dei thread è effettuato dal runtime support della libreria
- Il S.O. gestisce solo i processi

Thread a livello utente



Thread a livello utente

- Le thread table sono strutture private del processo
 - una thread table per ogni processo
- Scheduling dei thread realizzato dal processo
 - =>> le transizioni di stato dei thread **non possono** essere innescate da chiamate di sistema o da interruzioni
 - =>> i thread **devono** rilasciare esplicitamente la CPU per permettere allo scheduler dei thread di eseguire un altro thread (**thread_yield**)
 - =>> se un thread esegue un chiamata di sistema e si blocca in attesa di un servizio **tutto il processo a cui appartiene viene bloccato**
 - =>> non danno alcun vantaggio in sistemi multiprocessore

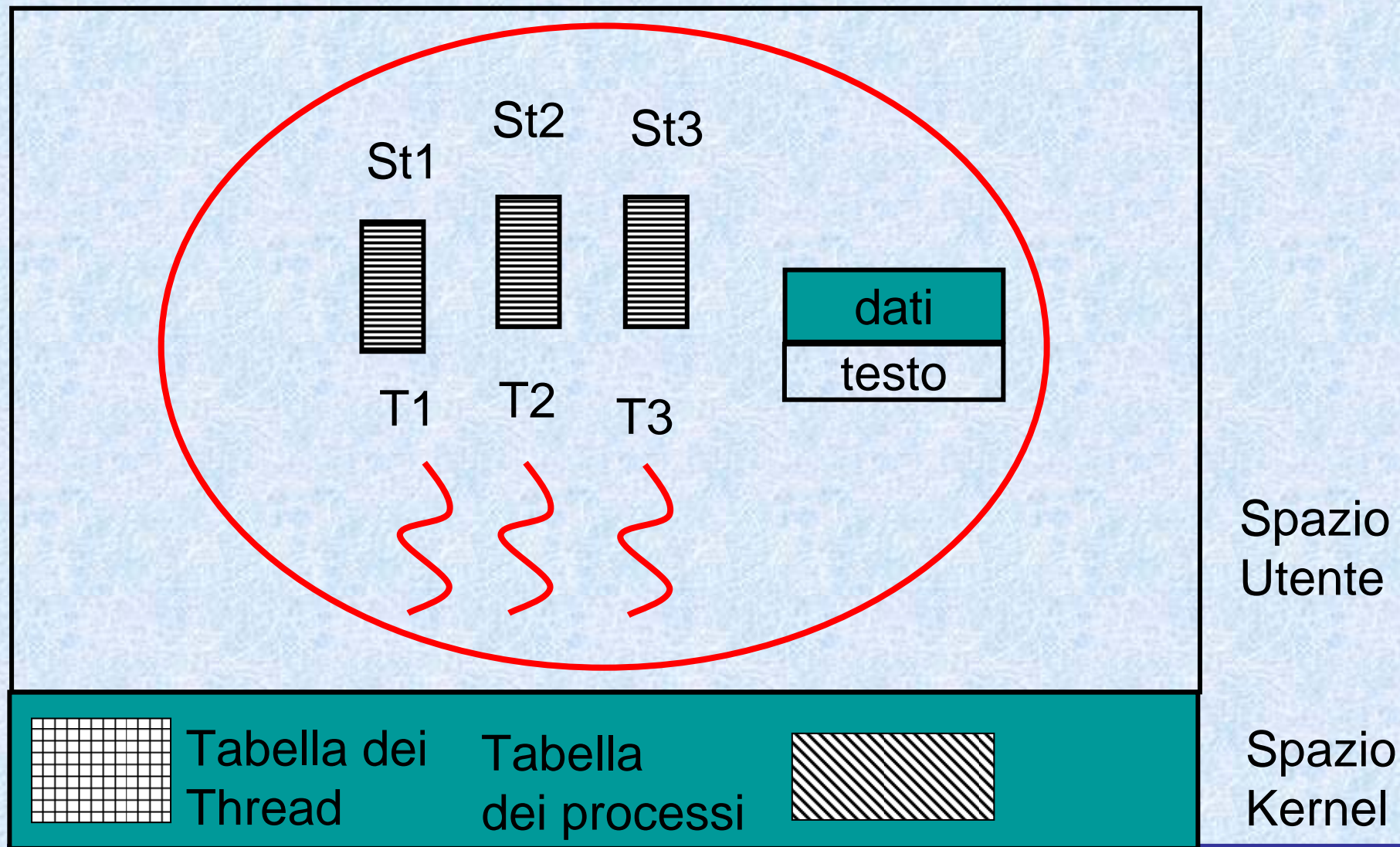
Thread a livello del nucleo

- Il S.O. gestisce direttamente i thread (generazione, terminazione, interazioni, **schedulazione**)

Windows: le unità schedulabili sono i thread

- Possibilità di utilizzare le potenzialità di un sistema multiprocessore

Thread a livello del nucleo



Thread a livello del nucleo

- Thread table unica (nel kernel)
- Le primitive che lavorano sui thread sono system call
 - `thread_create()`, `thread _exit()`, `thread_wait()`...
- Non è necessario che un thread rilasci esplicitamente la CPU
- Le system call possono bloccarsi senza bloccare tutti i thread di quel processo

Realizzazione di thread a livello utente (confrontata con realizzazione a livello di nucleo)

- **creazione e commutazione molto veloce**
- **interazioni inefficienti (interazioni non bloccanti)**
- **possibilità di scheduling “personalizzato”,
dipendente dall'applicazione**
- **realizzabili anche su un SO che supporti solo i
processi**
- **gestione problematica delle system call
bloccanti**
 - **necessari meccanismi non bloccanti**