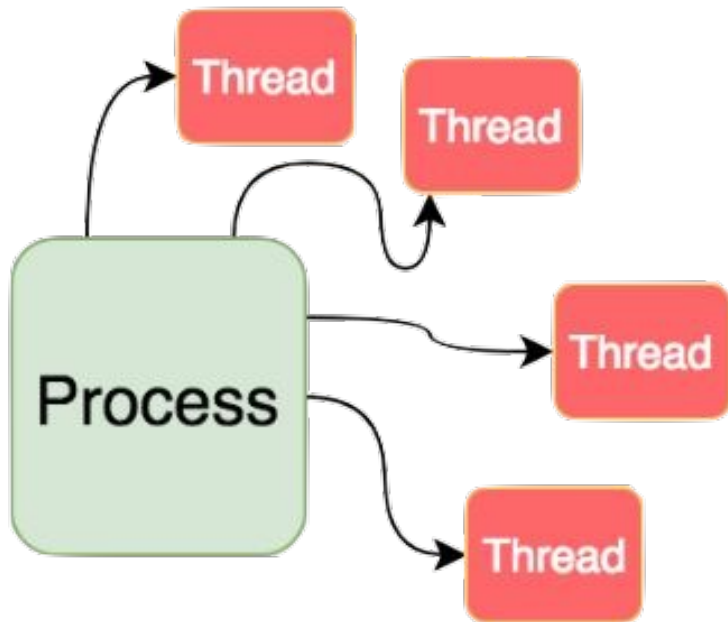


# Sistemi Operativi (M-Z)

# I Thread

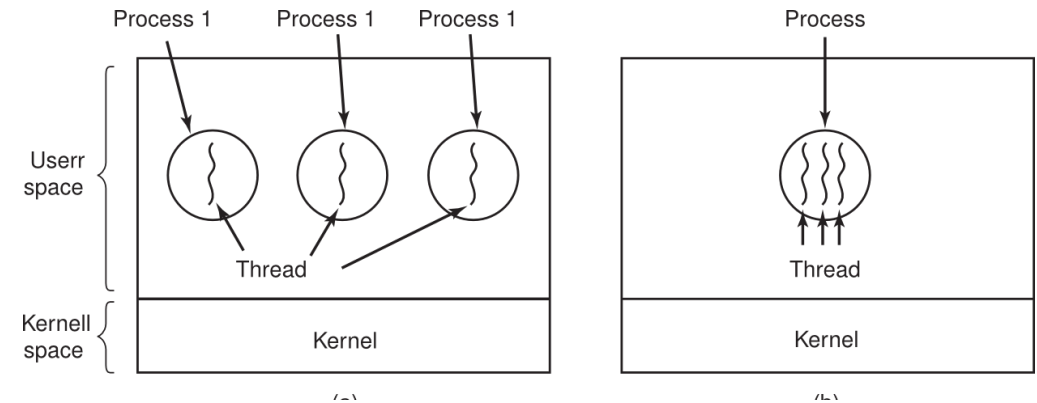


C.d.L. in Informatica  
(laurea triennale)  
A.A. 2023-2024

**Prof. Mario F. Pavone**

Dipartimento di Matematica e Informatica  
Università degli Studi di Catania  
[mario.pavone@unict.it](mailto:mario.pavone@unict.it)

# Thread



- Modello dei processi: entità indipendenti che **raggruppano risorse** e con un **flusso di esecuzione**;
- può essere utile far condividere a più flussi di esecuzione lo stesso spazio di indirizzi: **thread**;
- quando può essere utile?
  - esempi: web-browser, videoscrittura, web-server, ...

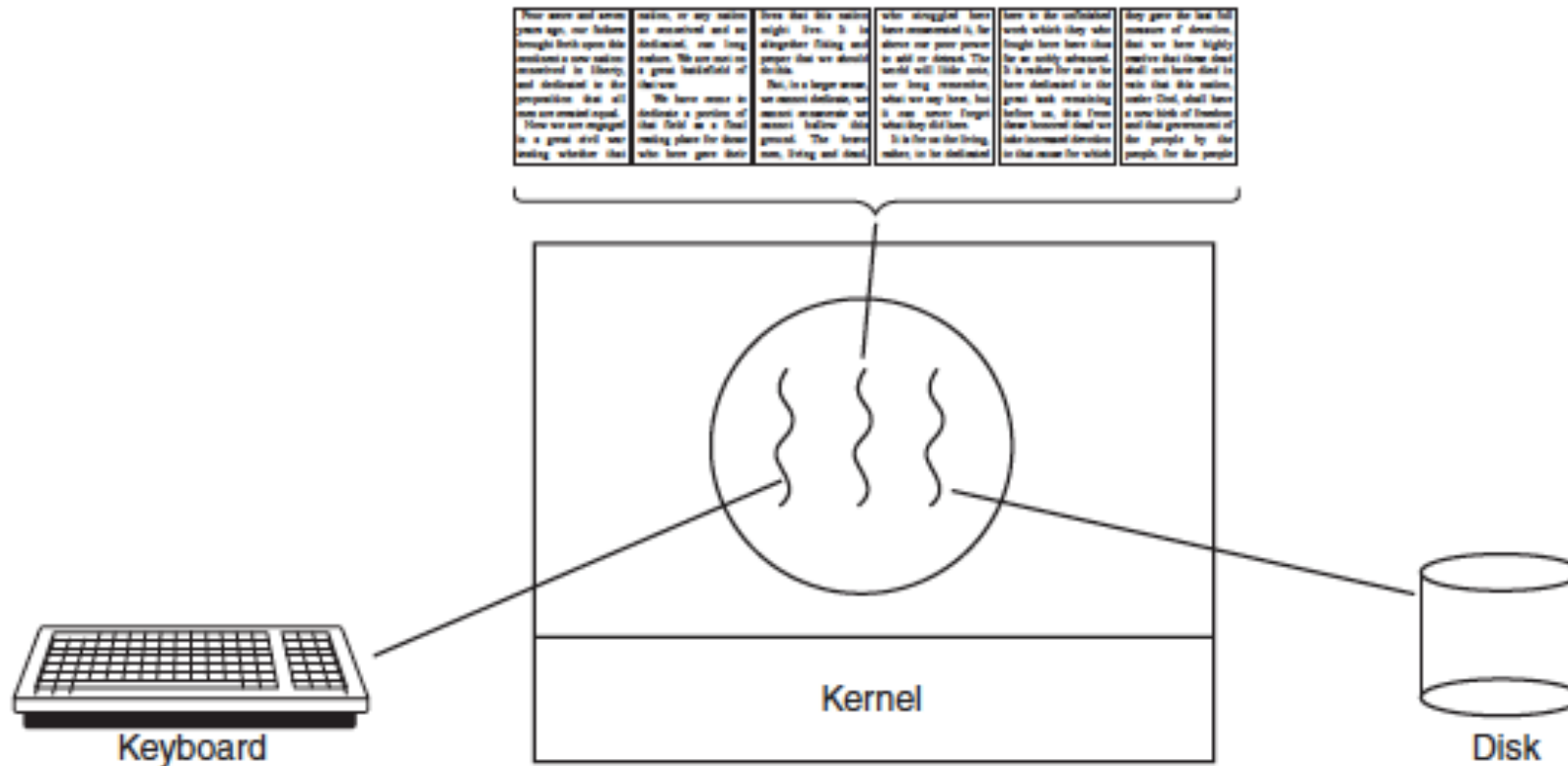


Figure 2-7. A word processor with three threads.

Thread: an example

• WORD PROCESSOR

# Thread: an example

## • WEB SERVER PROCESS

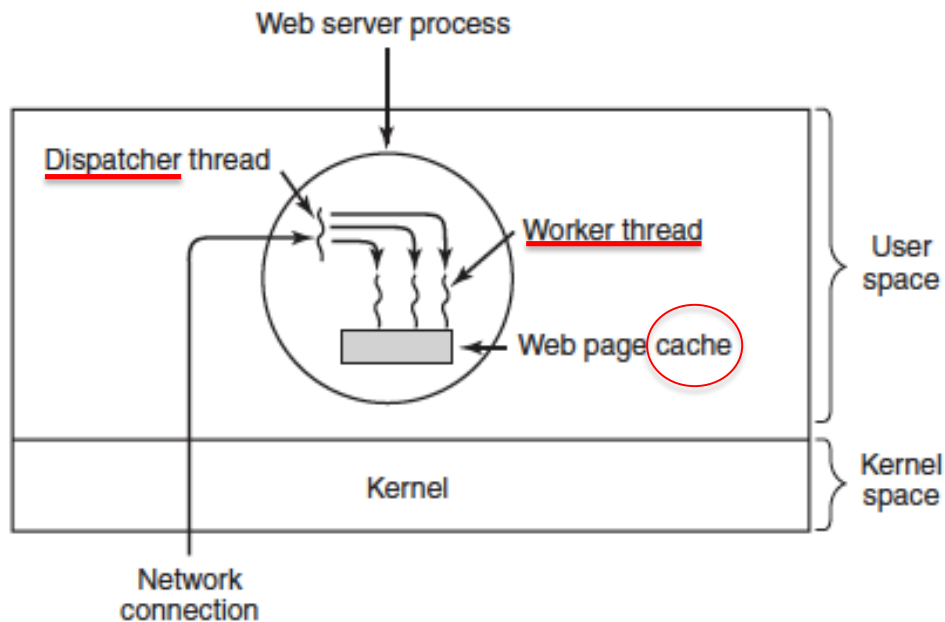
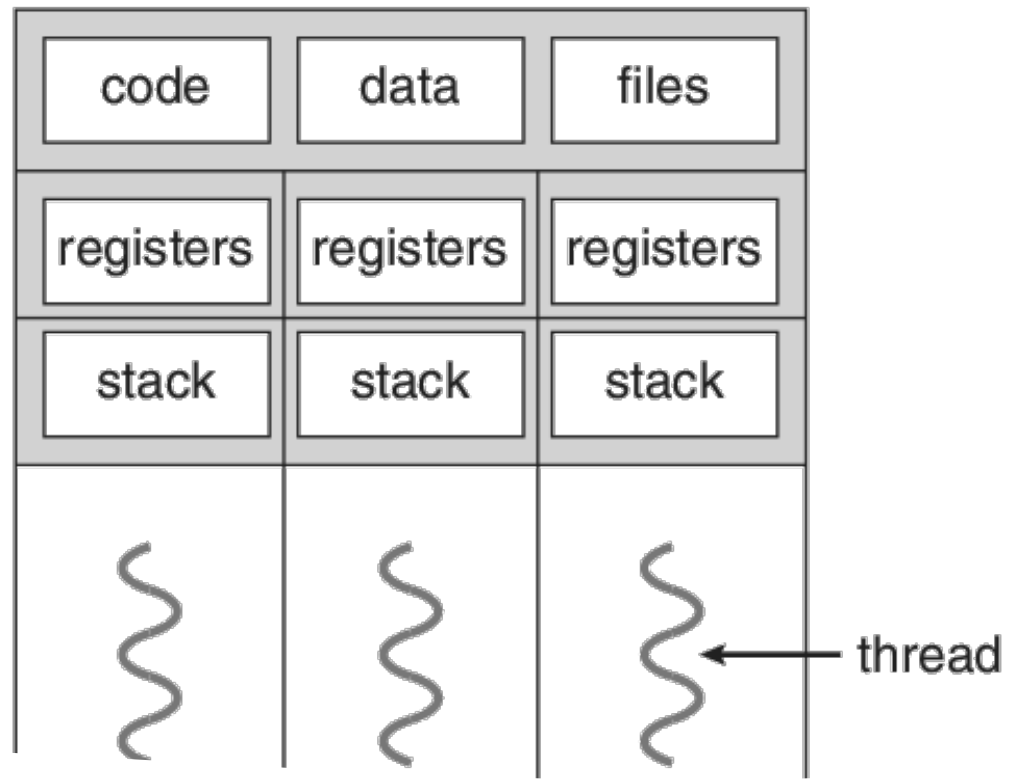
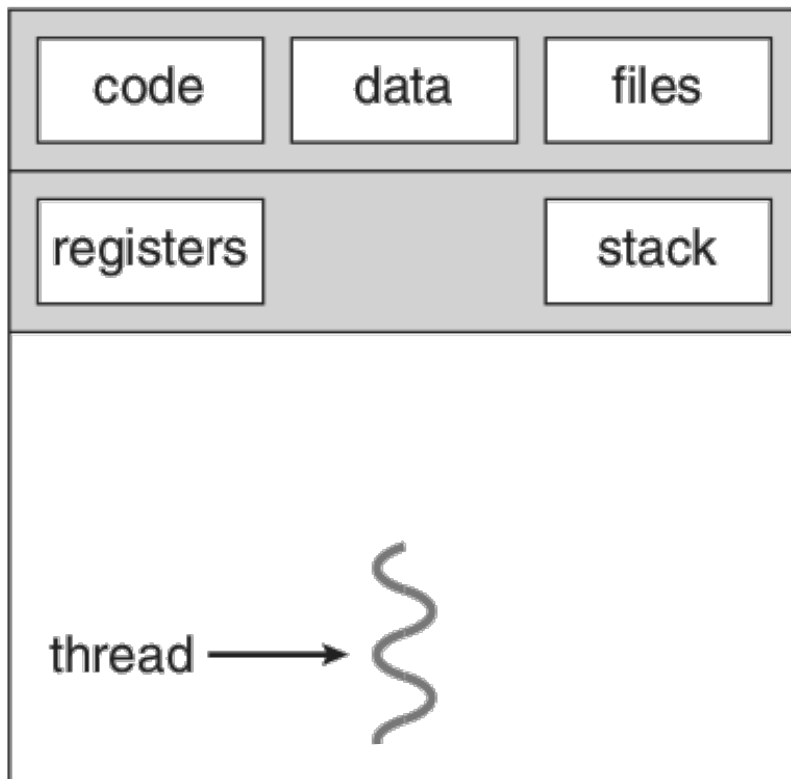


Figure 2-8. A multithreaded Web server.

*NOTA: il dispatcher è un ciclo infinito che acquisisce richieste di lavoro e le passa ad un worker thread...*

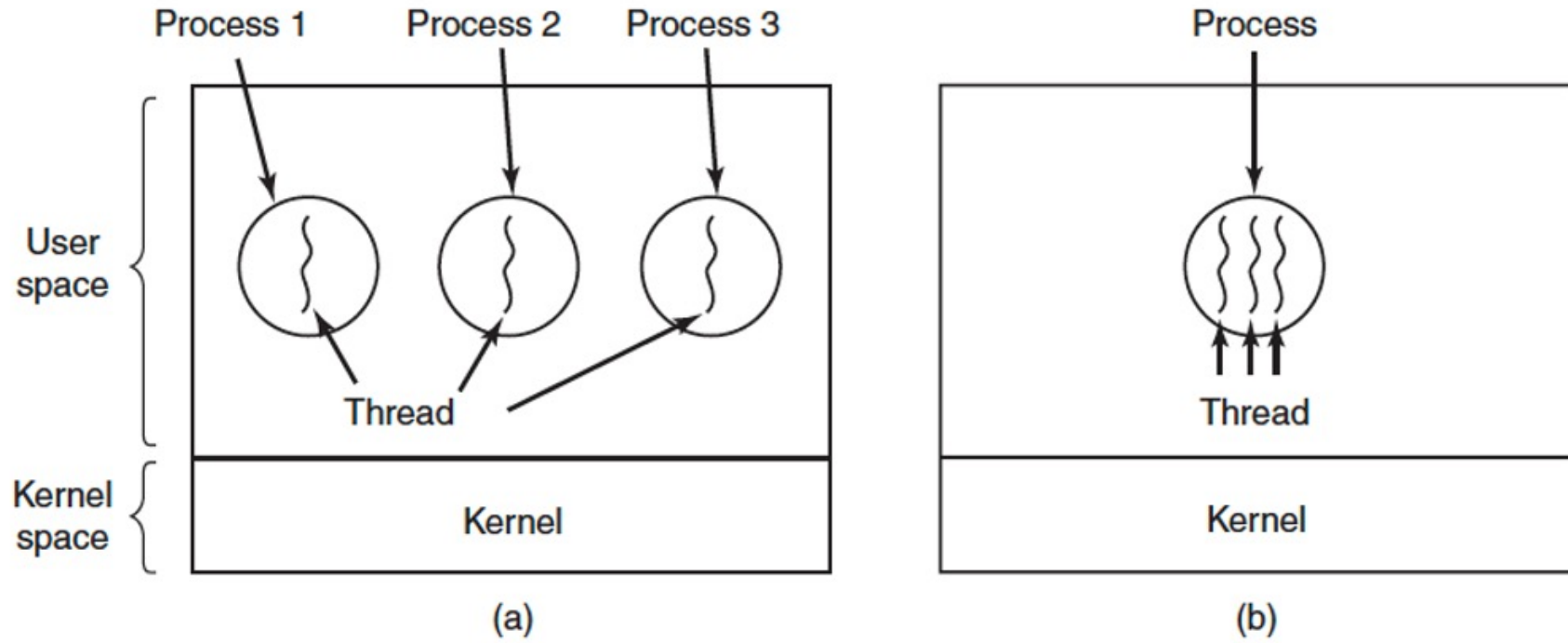
*... anche quello del worker thread è un ciclo infinito*



# Il Modello Thread

- Un thread è **caratterizzato** da:
  - PC, registri, stack, stato;
  - condivide tutto il resto;
    - non protezione di memoria.
- **scheduling** dei thread;
- **cambio di contesto** più veloce;

*Sistemi Multithreading*

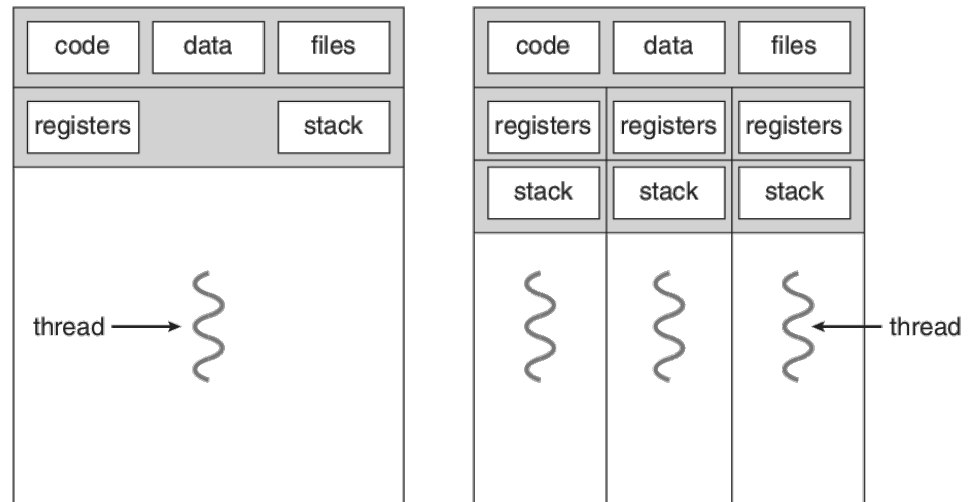


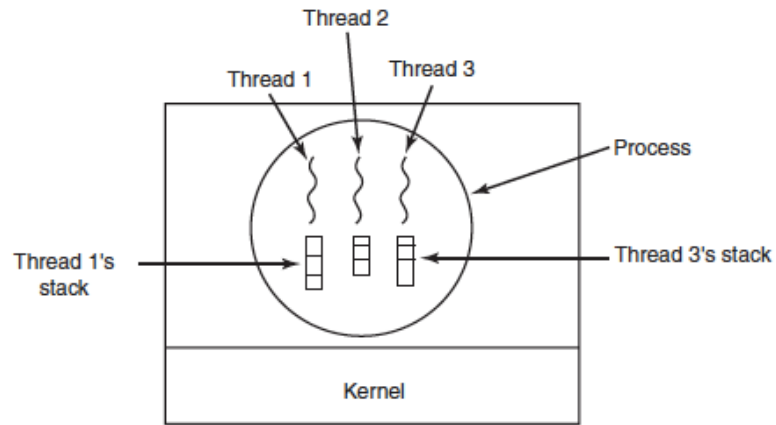
# Thread Vs. Processo

- Fig. (a): Ogni processo lavora in spazi degli indirizzi diversi
- Fig. (b): tutti i thread condividono lo stesso spazio degli indirizzi

# Thread

- Thread diversi nello stesso processo non sono indipendenti
- Condivisione e **No protezione di memoria**
  - un thread può leggere, scrivere o cancellare lo stack di un altro thread
- Stesso user -> Cooperano -> non entrano in conflitto
- Thread apre file => è visibile ad ogni thread nel processo
- Unità di gestione delle risorse => Processo



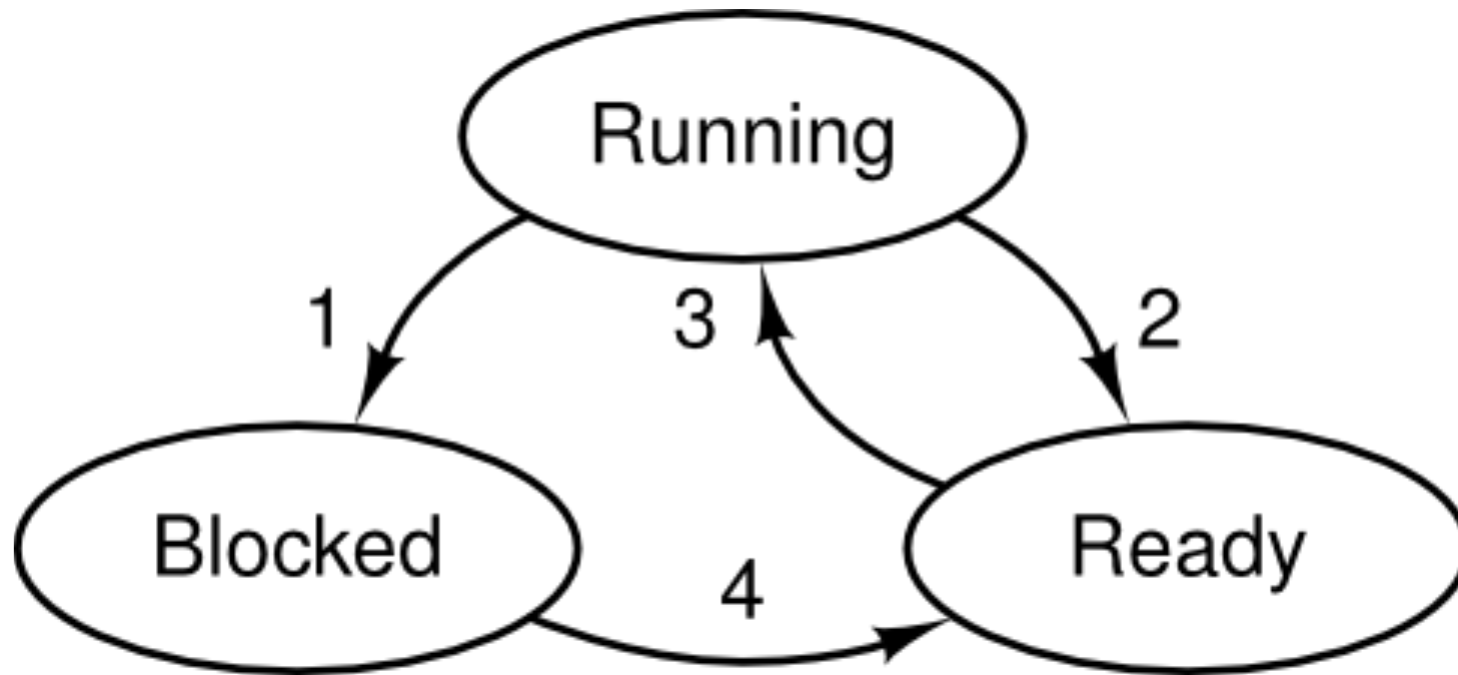


Per-process items	Per-thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

# Thread

- Ogni Thread ha un proprio Stack
- **storia di esecuzione diversa**





# Thread

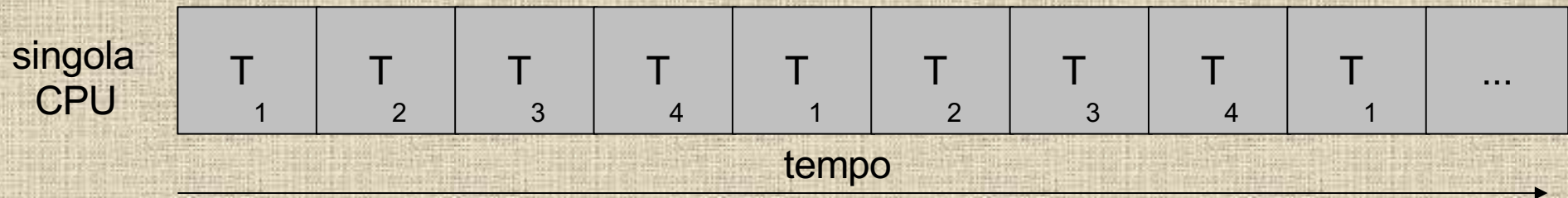
- Anche un thread può trovarsi in uno dei seguenti stati
- Transizioni di stato => analogo a quello dei processi.

# Operazioni sui Thread

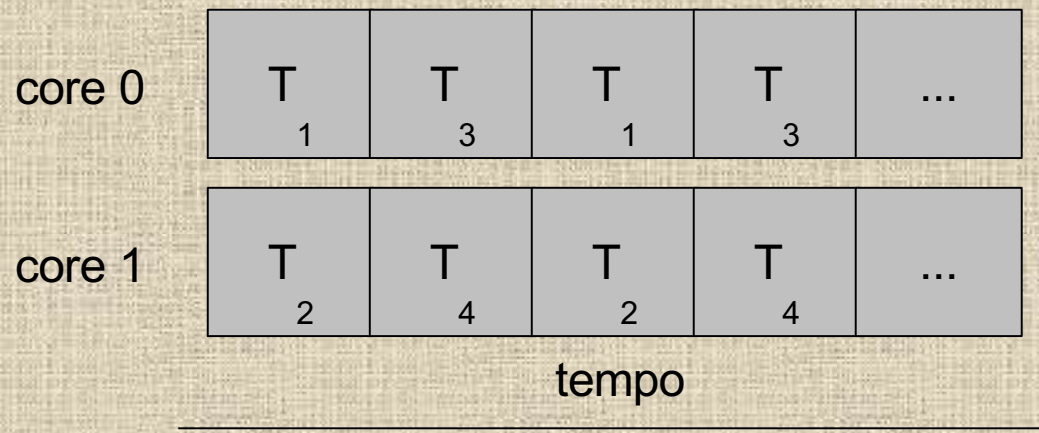
- Nei sistemi Multithreading i processi iniziano con un singolo thread
- **operazioni** tipiche sui thread:
  - **thread\_create**: crea un nuovo thread e specifica la procedura che deve eseguire;
  - **thread\_exit**: il thread chiamante termina;
  - **thread\_join**: un thread si sincronizza con la fine di un altro thread (attende che uno specifico thread termini);
  - **thread\_yield**: il thread chiamante rilascia **volontariamente** la CPU
    - **no clock interrupt**

# Programmazione multicore

- I thread permettono una **migliore scalabilità** con core con hypertreading e soprattutto con sistemi multicore;
- con un sistema single-core abbiamo una esecuzione interleaved;



- su un sistema multi-core abbiamo parallelismo puro.



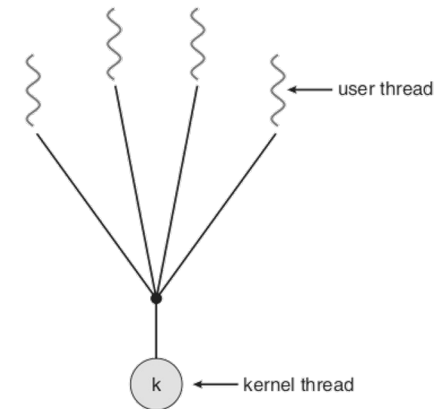
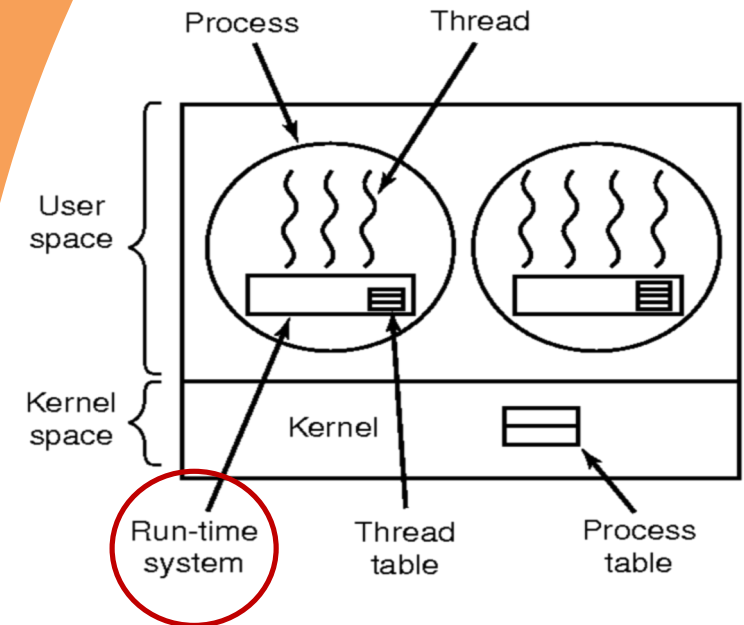


# Programmazione multicore

- Progettare programmi che sfruttino le moderne architetture multicore non è banale;
- **principi base:**
  - **separazione dei task;**
    - trovare aree per attività separate e simultanee
  - **bilanciamento;**
    - assicurarsi che eseguano lo stesso lavoro di uguale valore
  - **suddivisione dei dati;**
    - dividere i dati per essere eseguiti su core separati
  - **dipendenze dei dati;**
    - esecuzione delle attività sia sincronizzata
  - **test e debugging**

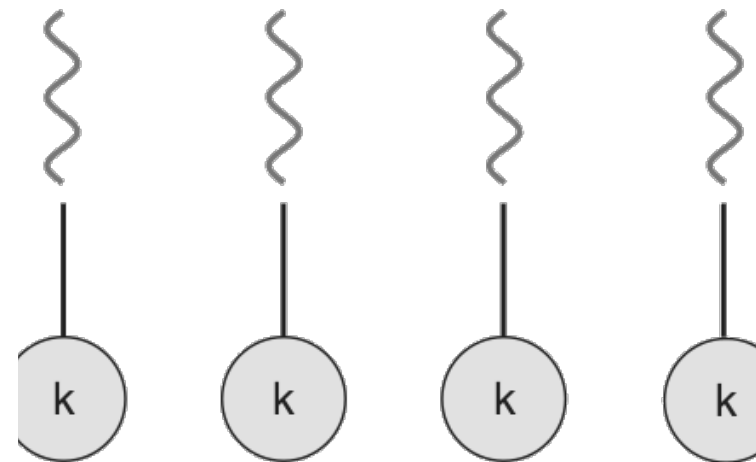
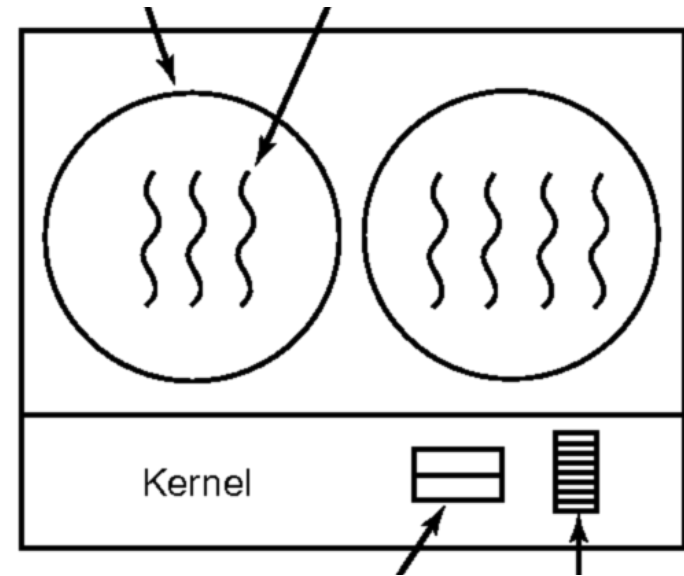
# Thread a livello utente

- Detto anche "modello 1-a-molti";
- utile se non c'è supporto da parte del kernel ai thread;
- una **libreria** che implementa un **sistema run-time** che gestisce una **tabella dei thread** del processo.
- **Pro:**
  - il dispatching non richiede trap nel kernel;
  - scheduling personalizzato;
- **Contro:**
  - chiamate bloccanti (select, page-fault);
  - possibilità di non rilascio della CPU.



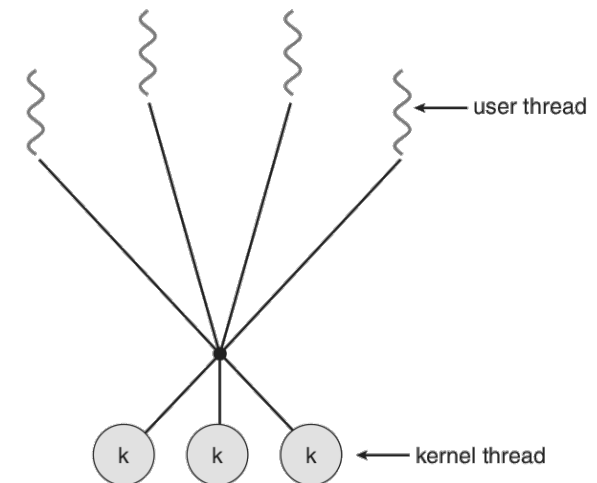
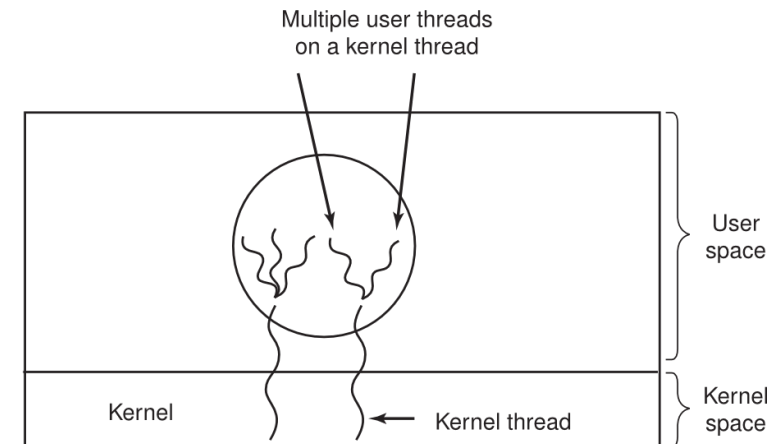
# Thread a livello kernel

- Detto anche **"modello 1-a-1"**;
- richiede il supporto specifico dal kernel (praticamente tutti i moderni SO);
- **unica tabella dei thread** del kernel;
- **Pro:**
  - un thread su chiamata bloccante non intralcia gli altri;
- **Contro:**
  - cambio di contesto più lento (richiede trap);
  - creazione e distruzione più costose (numero di thread kernel tipicamente limitato, possibile riciclo).



# Modello ibrido

- Detto anche "**multi-a-molti**";
- prende il meglio degli altri due;
- prevede un certo numero di **thread del kernel**;
- ognuno di essi viene assegnato ad un certo numero di **thread utente** (eventualmente uno);
- **assegnazione** decisa dal programmatore.



# I thread nei nostri sistemi operativi

- Quasi tutti i sistemi operativi supportano i **thread a livello kernel**;
  - Windows, Linux, Solaris, Mac OS X,...
- Supporto ai **thread utente** attraverso apposite librerie:
  - *green threads* su Solaris;
  - *GNU portable thread* su UNIX;
  - *fiber* su Win32.
- **Librerie di accesso ai thread** (a prescindere dal modello):
  - **Pthreads** di POSIX (Solaris, Linux, Mac OS X, anche Windows);
    - una specifica da implementare sui vari sistemi;
  - threads Win32;
  - thread in Java;
    - wrapper sulle API sottostanti.