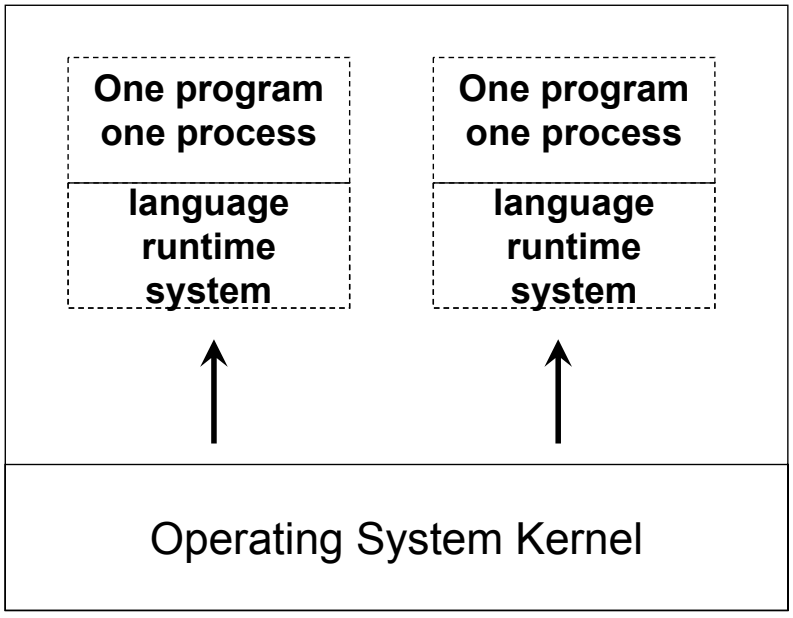
Thread

Alcuni aspetti architetturali

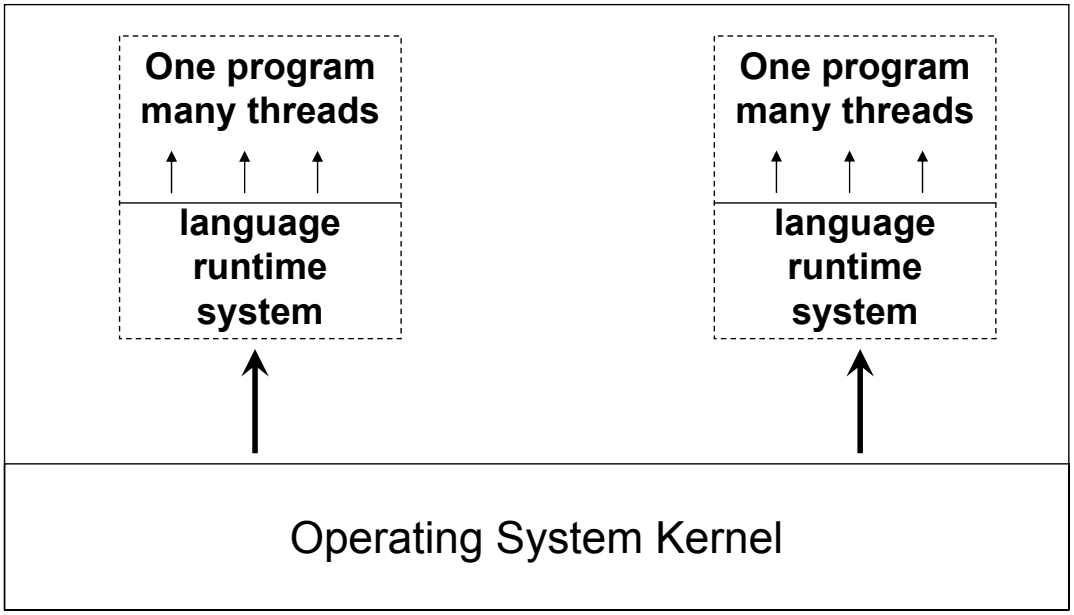
Schemi realizzativi

Modello sequenziale



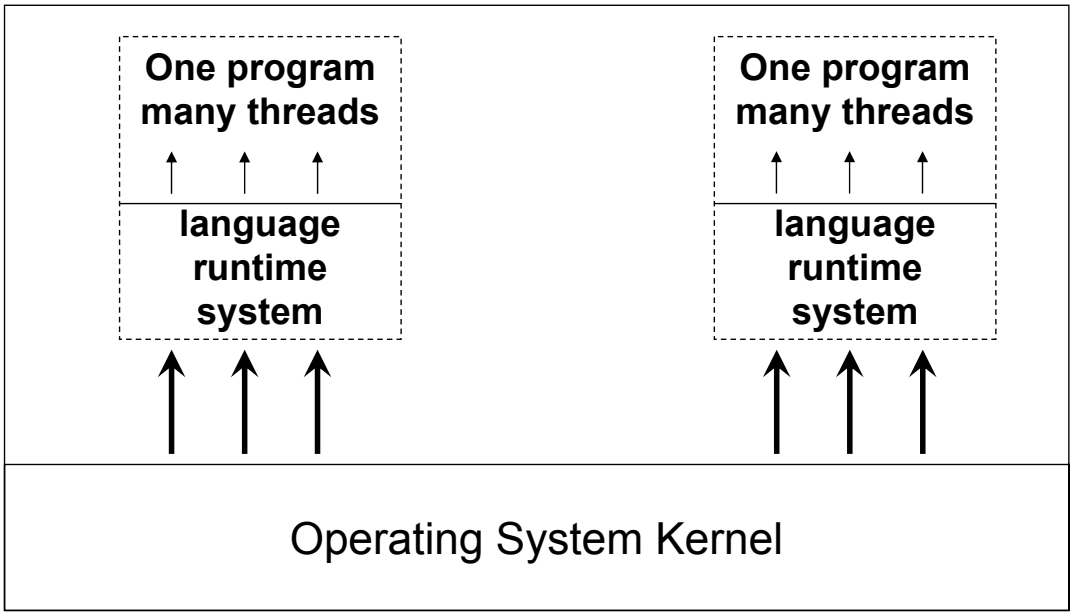
Schemi realizzativi

Linguaggio concorrente senza supporto del SO

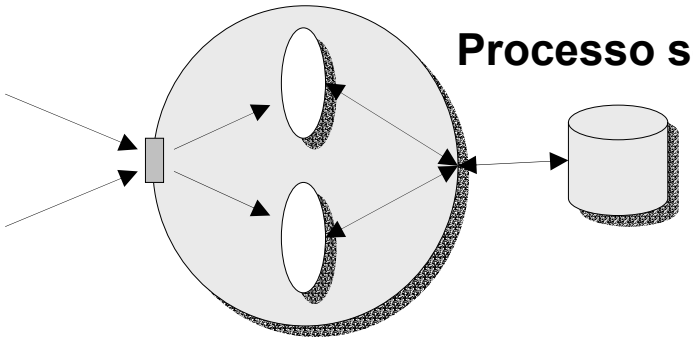


Schemi realizzativi

Linguaggio concorrente con supporto del SO



Throughput: multi-thread (monoprocessore)



gli accessi al disco sono sequenziali

T_c = tempo di calcolo (es., 2 ms);

T_a = tempo di accesso al disco (es., 8 ms)

Elapsed Time = $T_e = T_c + T_a = 2 + 8 = 10$ ms

THROUGHPUT, $\mathcal{T}$ (singolo elaboratore)

Singolo thread	$\mathcal{T}_S = 1/T_e = 100$ richieste/s
Multi-thread (due thread)	$\mathcal{T}_M = 1/T_a = 125$ richieste/s

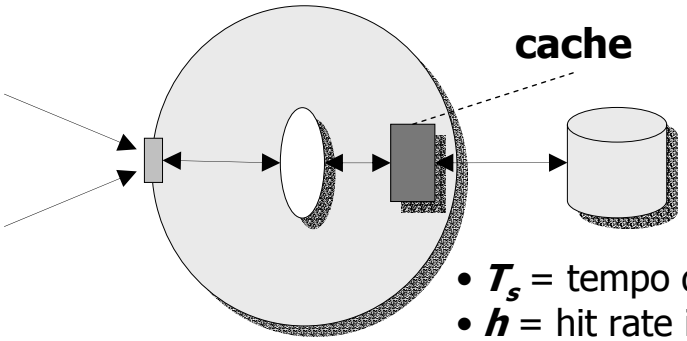
Throughput: multi-thread (monoprocessore)

Speedup

$$\begin{cases} F_e = \dfrac{T_c}{T_e} = \dfrac{2}{10} = 0.2 \\ S_e \to \infty \end{cases} \Rightarrow \boxed{S_o = \frac{1}{1-F_e} = \frac{1}{0.8} = 1.25}$$

- È la parte sequenziale del programma (1-F_e) che fissa le prestazioni
- Non c'è vantaggio (teorico) ad aumentare il grado di parallelismo ($S_e \to \infty$)

Throughput: cache



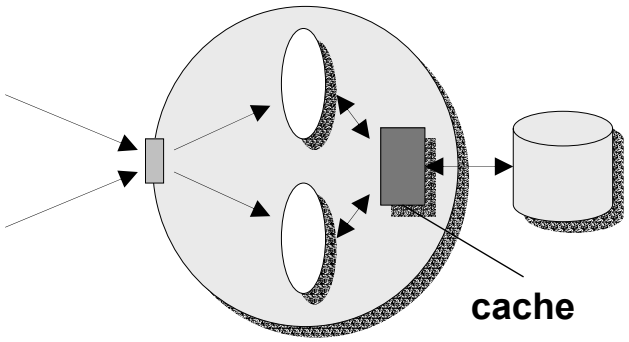
gli accessi al disco sono sequenziali

- T_s = tempo di ricerca in cache (es., 0.5 ms);
- h = hit rate in cache (es., 75%)

$T_e = T_c + T_a = 2 + 8 = 10$ ms

$$\begin{cases} F_e = \dfrac{h \times T_a}{T_e} = \dfrac{0.75 \times 8}{10} = 0.6 \\ S_e = \dfrac{h \times T_a}{T_s} = \dfrac{0.75 \times 8}{0.5} = 12 \end{cases} \Rightarrow S = \frac{1}{(1 - F_e) + \dfrac{F_e}{S_e}} = \frac{1}{0.4 + \dfrac{0.6}{12}} = \frac{1}{0.45} = 2.22$$

Throughput: multi-thread con cache (monoprocessore)



h = hit rate in cache (75%)

T_s = tempo di ricerca in cache (0.5 ms)

Tempo medio di accesso al disco: $T_a' = (1-h)T_a = 2\ ms$

Elapsed time medio: $T_e' = \underbrace{(T_c + T_s)}_{T_c'} + T_a' = T_c' + T_a' = 4.5\ ms$

THROUGHPUT

$\mathcal{T}_S = 1/T_e' \simeq 222$ richieste/s (single-thread)

$\mathcal{T}_M = 1/T_c' = 400$ richieste/s (multi-thread)

Throughput: multi-thread con cache (monoprocessore)

ipotesi semplificativa:
cache e multi-thread sono indipendenti

Speedup$_{cache}$

$$\begin{cases} F_e' = \dfrac{h \times T_a}{T_e} = \dfrac{0.75 \times 8}{10} = 0.6 \\ S_e' = \dfrac{h \times T_a}{T_s} = \dfrac{0.75 \times 8}{0.5} = 12 \end{cases} \Rightarrow S_o' = 2.22$$

Speedup$_{multi}$

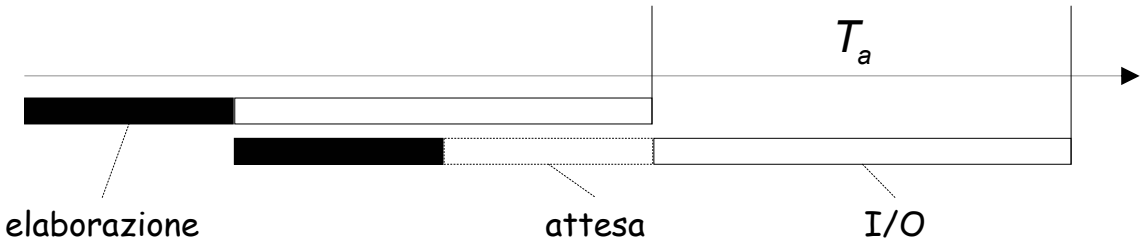
$$\begin{cases} F_e'' = \dfrac{T_c}{T_e'} = \dfrac{2}{4.5} \Rightarrow S_0'' = \dfrac{1}{1 - F_e''} = \dfrac{4.5}{2.5} = 1.8 \\ S_e' \to \infty \end{cases}$$

Speedup$_{totale}$

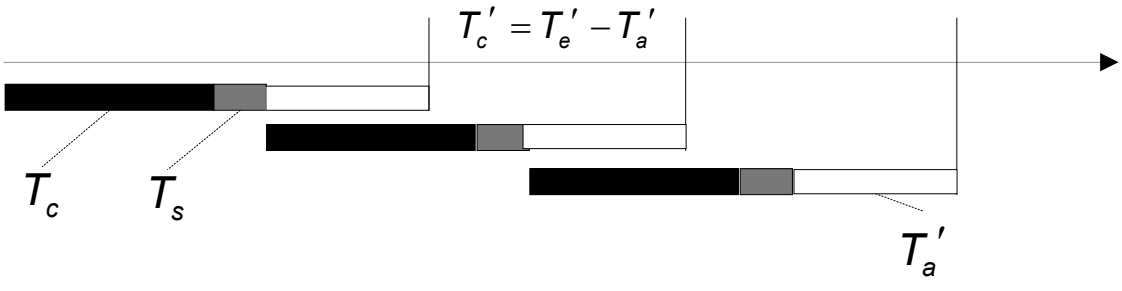
$$S_o = S_o' \times S_0'' = \frac{1}{0.45} \times \frac{4.5}{2.5} = 4$$

Throughput: multi-thread con cache (monoprocessore)

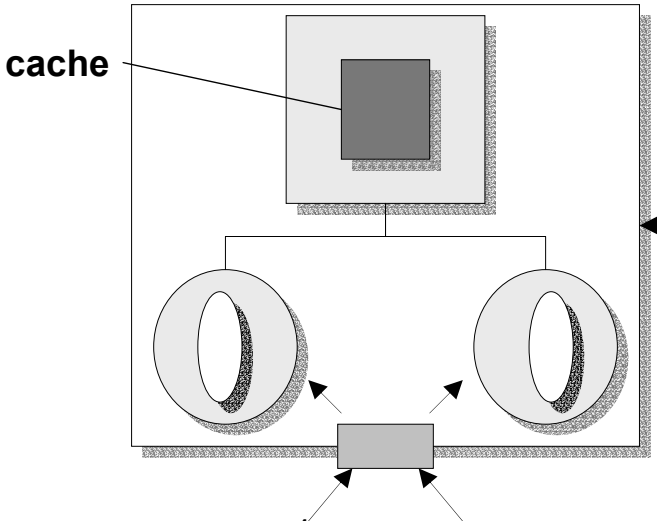
multi-thread senza cache



multi-thread con cache



Throughput (biprocessore)



server host, biprocessore a memoria condivisa

gli accessi al disco sequenziali

$$\mathcal{T}_S = 2 \times \frac{1}{\left(T_a' + T_c + T_s\right)} \simeq 444 \text{ richieste/s}$$

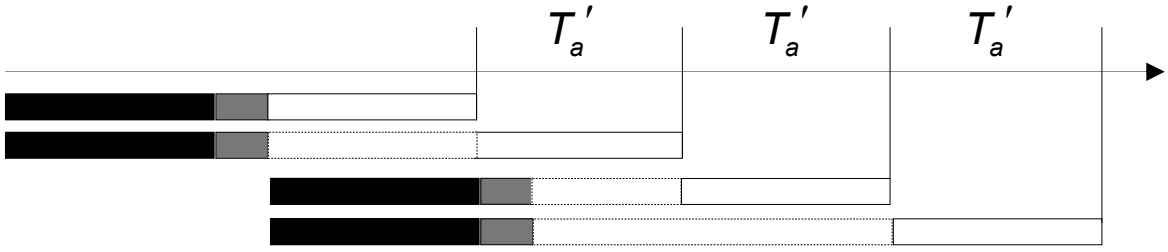
(due thread, un thread per processore)

$$\mathcal{T}_M = \frac{1}{T_a'} = 500 \text{ richieste/s}$$

(due thread per processore)

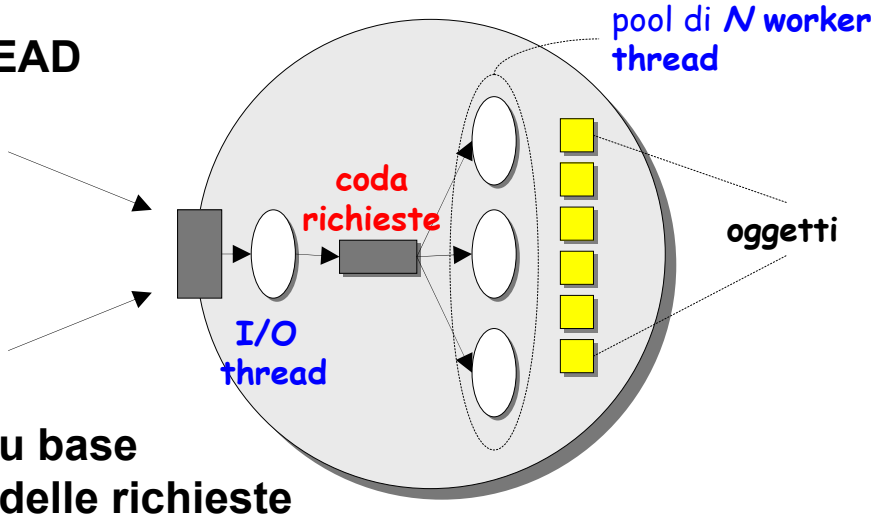
Schemi temporali (biprocessore)

MULTITHREAD CON CACHE (due thread/processore)



Architetture di server multi-thread

POOL DI THREAD



Vantaggi

- **semplicità**
- **gestione su base prioritaria delle richieste**

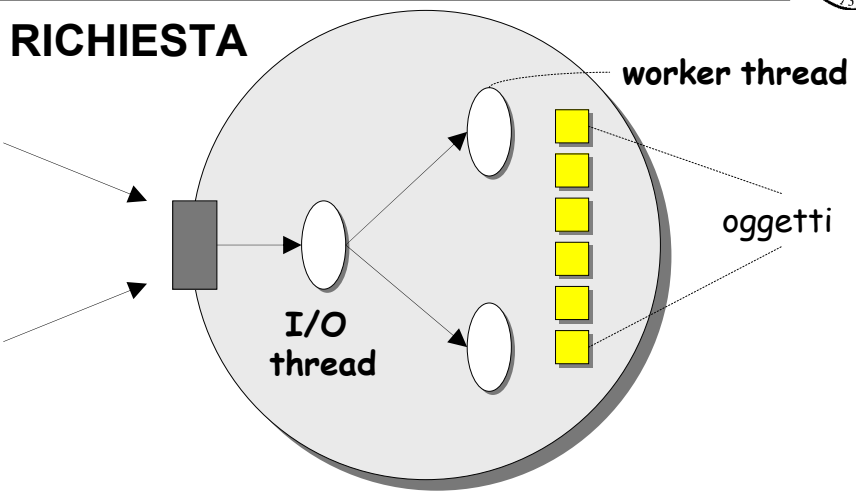
Svantaggi

- **limitata flessibilità dovuto al numero *N* fisso dei thread**
- **overhead dovuto alla *coda richieste***

Architetture di server multi-thread

UN THREAD PER RICHIESTA

- **Si crea un thread per ogni richiesta.**
- **Si distrugge il thread non appena ha servito la richiesta**



Vantaggi

- **non c'è l'overhead della coda richieste**
- **il numero di thread non è limitato**

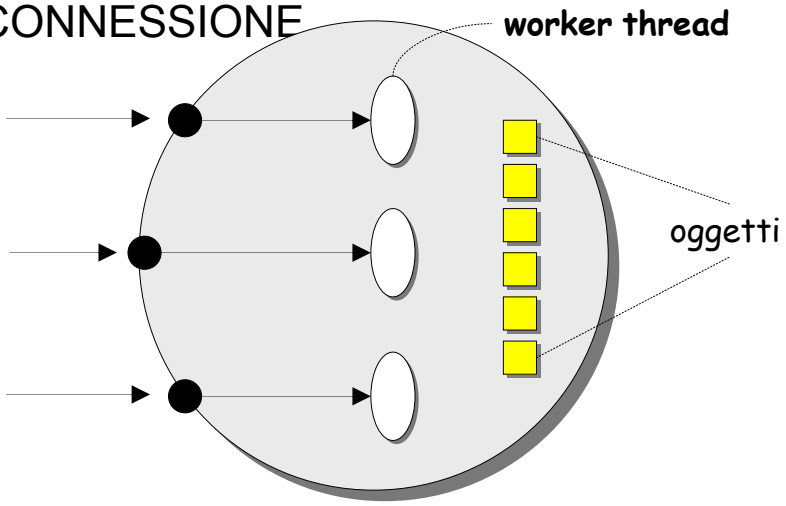
Svantaggi

- **overhead dovuto alla creazione ed alla distruzione dinamica dei thread**

Architetture di server multi-thread

UN THREAD PER CONNESSIONE

Si crea un worker thread quando il cliente stabilisce una connessione e lo si distrugge quando la connessione viene chiusa



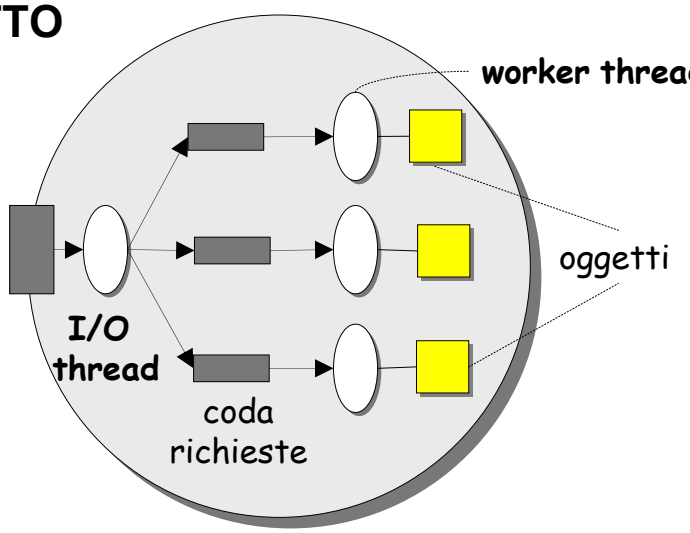
Su una connessione possono arrivare richieste per qualunque oggetto

- **Vantaggi: si riduce l'overhead della creazione/distruzione dinamica dei thread;**
- **Svantaggi: un thread può avere richieste pendenti mentre altri thread sono inattivi**

Architetture di server multi-thread

UN THREAD PER OGGETTO

**Si crea un worker thread per ciascun oggetto.
L'I/O thread inoltra le richieste agli worker thread**



- **Vantaggi: elimina l'overhead della creazione/distruzione dinamica dei thread**
- **Svantaggi: un thread può avere richieste pendenti mentre altri thread sono inattivi**

Thread vs Processi

- Sia i thread sia i processi permettono di incrementare il grado di multiprogrammazione
  - permettono di sovrapporre l'elaborazione all'I/O
  - permettono l'esecuzione concorrente sui multiprocessori
- Tuttavia, rispetto ai processi, i thread
  - facilitano la condivisione delle risorse (memoria condivisa)
  - sono più efficienti da gestire: la creazione ed il context switch sono più efficienti con i thread che con i processi (≅ rapporto 1:10)