# LABORATORIO DI RETI DI CALCOLATORI

## Gestione client multipli: struttura server



## Bibliografia

- slide della docente
- *testo di supporto*: D. Maggiorini, "Introduzione alla programmazione client-server", Pearson Ed., 2009
  - cap.1 (tutto)

# Client vs. server

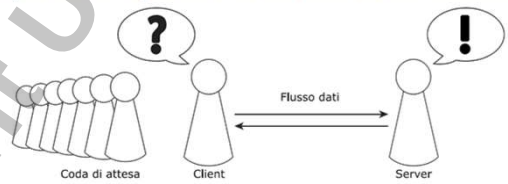
- lato **server**:
  - mette a disposizione risorse (file, computazione, hardware…)
  - sempre in attesa di clienti: indirizzo *"ben noto"*
  - deve: controllare sicurezza (AAA)
    **gestire efficientemente più clienti contemporanei**
  - server farm / mirror

- lato **client**:
  - usa risorse → inizia la comunicazione
  - indirizzo qualsiasi (serve solo per la risposta)
  - se termina inaspettatamente: server deve continuare ad operare



# Server iterativo

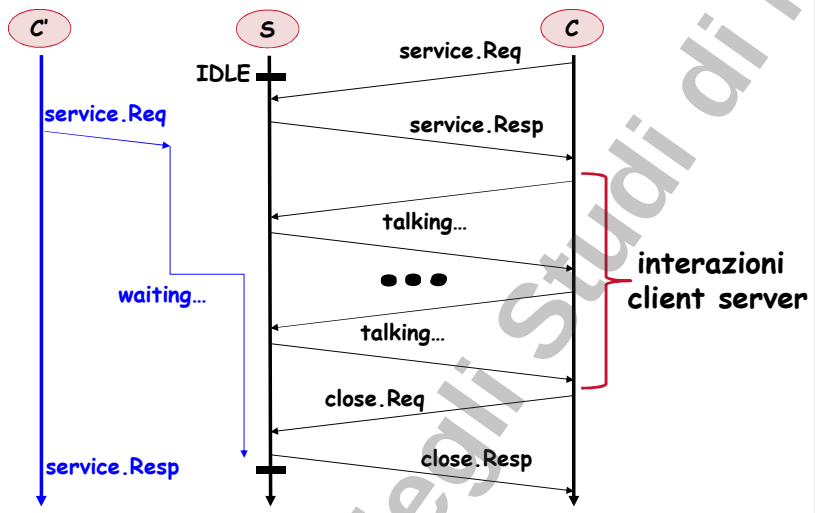
- se il server è libero → il client può entrare in servizio immediatamente
- se il server è occupato →
  (a) c'è spazio in coda e il client può mettersi in attesa
  (b) la coda è piena e il servizio viene rifiutato




es. un risparmiatore e uno sportello bancomat

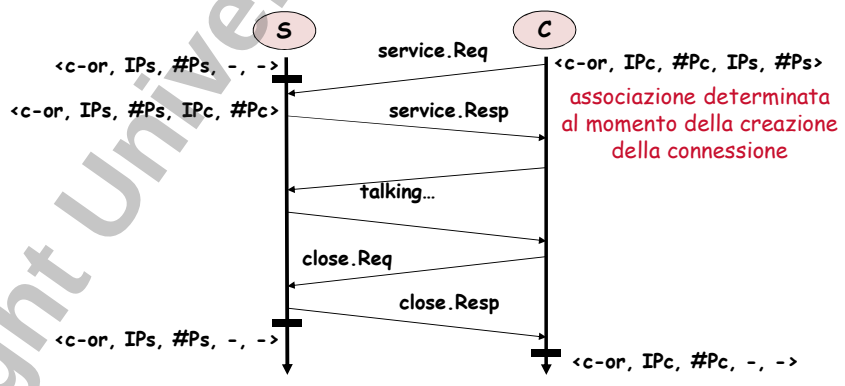
- Problemi
  - gestione coda; lunga attesa (starvation); attacchi Denial of Service (DoS)

# Server iterativo: diagramma temp.






# conn-oriented con server iterativo



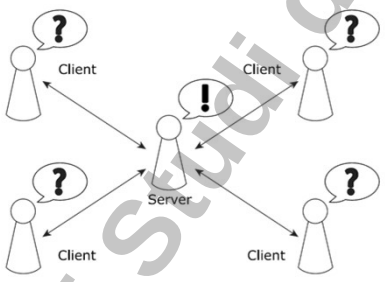

- l'associazione è *completamente definita* per tutta la conversazione: server *non può* associarsi ad altri client
  - in effetti: la socket poi non può essere ri-usata con altri…

# Server concorrente

- La concorrenza dell'erogazione è un **apparente** parallelismo nel mantenere i rapporti (time sharing), non nell'inviare o ricevere dati.

es. la segretaria di un'azienda e gli impiegati che ci lavorano

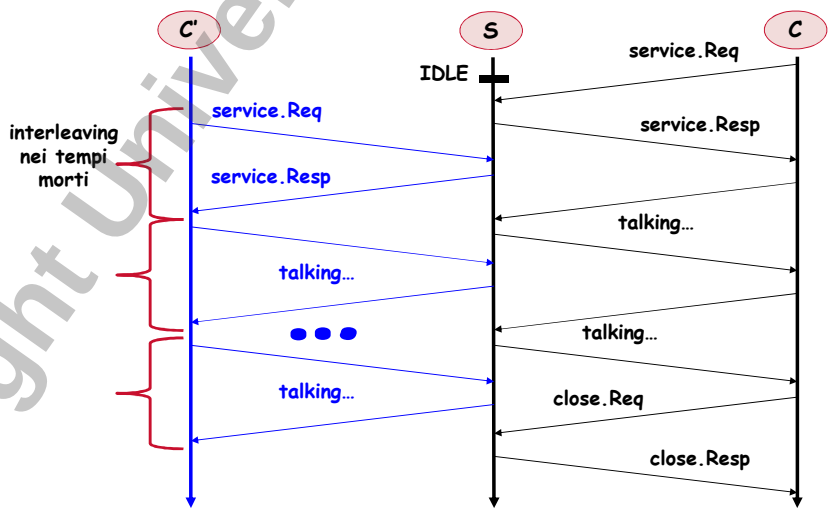
server sfrutta tempi morti di un client (I/O bound) per servirne altri



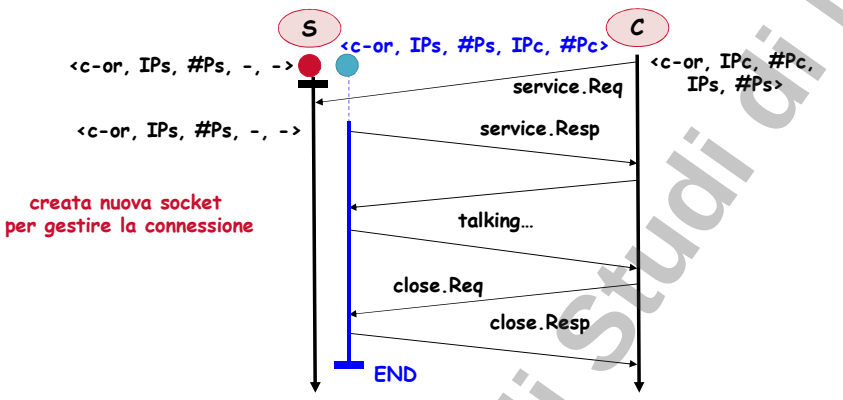

- Problemi
  - gestione difficile: il server deve accentrare su di sé tutti i canali di comunicazione aperti con i client → non *scalabile*
  - e se i canali finiscono?



# Server concorrente: diagr. temp.



# conn-oriented con server concorrente

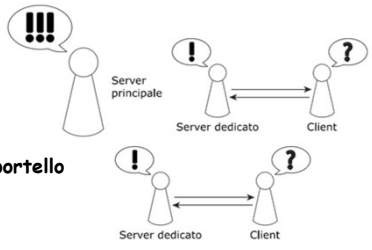



- su socket con associazione parzialmente definita si possono accettare altre richieste di connessione
- server monitora tutti i suoi canali (socket)



# Server multi-processo

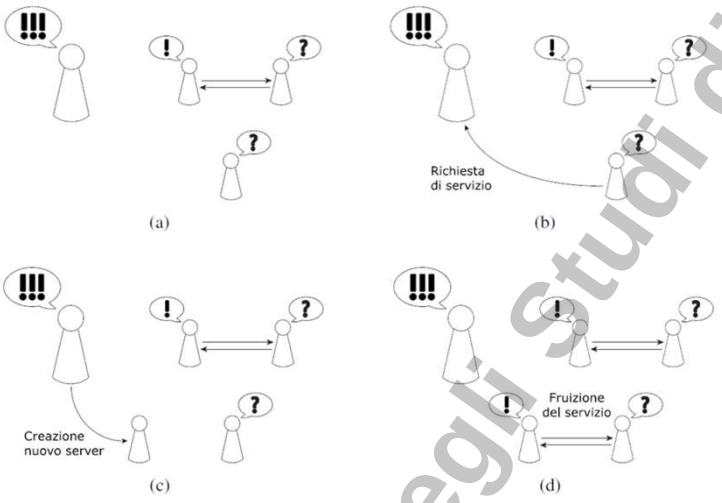
- Più server entrano in gioco contemporaneamente
  - entità distinte e contemporaneamente attive
  - più flussi esecutivi paralleli per la fornitura del servizio (processi / thread)
  - parallelismo reale?
    - scheduling CPU…




es. un correntista e la sua banca
- ogni cliente è delegato ad un diverso sportello

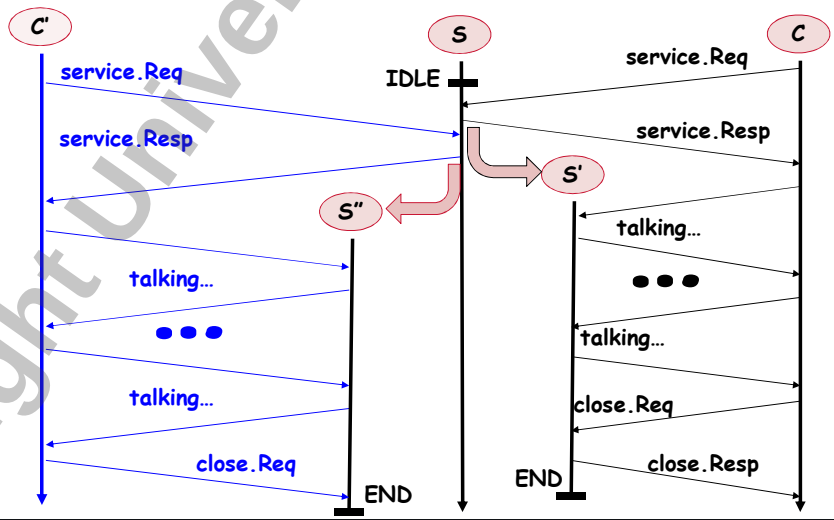
- Problemi
  - gestione limitatezza e carico computazionale di ciascun processo/thread
  - ogni thread gestisce la coda in modo iterativo o concorrente

# Server multi-processo



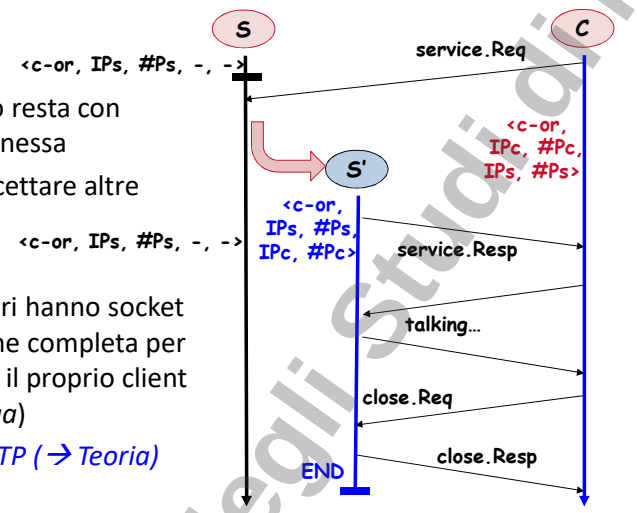



# Server multiprocesso: diagr. temp.



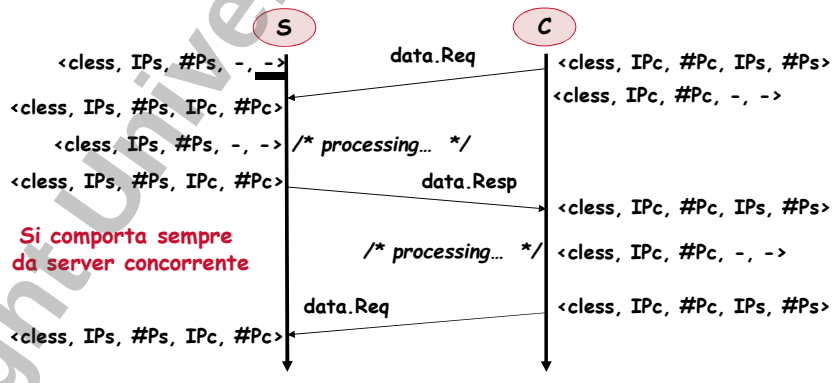
# conn-oriented con server multiproc.

- server primario resta con socket non connessa
- su essa può accettare altre richieste
- server secondari hanno socket con associazione completa per gestire ognuno il proprio client (che *disambigua*)
- *così funziona FTP (→ Teoria)*






# E connectionless?




- l'associazione è *sempre parzialmente definita* tranne nell'istante in cui si verifica l'invio/ricezione di un messaggio
  - possibile interleaving di diverse conversazioni

## server: confronti

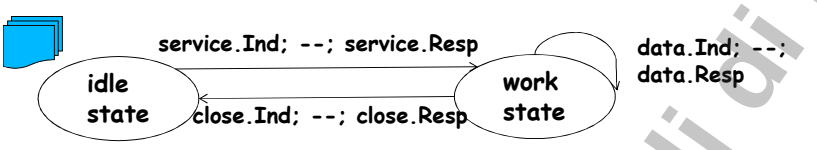
- server iterativo vs. concorrente:
  - se interazione client-server è di solo un messaggio → sono comparabili
  - altrimenti: interleaving tra interazioni con client diversi
- server concorrente vs. multi-processo:
  - concorrente sfrutta tempi morti di un client per servirne altri
  - multi-processo: gestisce parallelamente diversi client in funzione di politica di scheduling
  - processi: hanno spazi di memoria *separati*
  - thread: condividono spazio di indirizzamento
    - meccanismi di gestione accessi concorrenti
  - in C si preferiscono *processi*, in Java si preferiscono *thread*



## Confronto modelli servizio

- cosa succede se interazione con server è di 1 messaggio?
  - server iterativo è molto simile a quello concorrente
  - allora: *scelta server può dipendere da modalità interazione*
- concorrenza di server multi-processo è reale?
  - beh… con una sola CPU… ☺
  - server figli gestiscono unico client, quindi non sono né iterativi né concorrenti!
- server concorrente è il più difficile da implementare
  - in ogni istante può ricevere messaggi di qualunque tipo
  - … si vede negli automi!

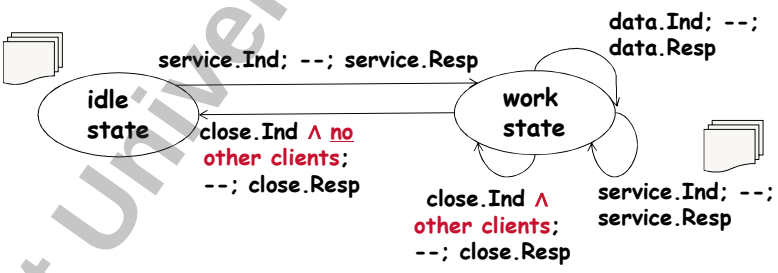
# Server iterativo: ASF




- esce da Idle quando riceve service.Req
- quando vi entra, ne esce immediatamente se esistono altre service.Req in coda
  - cioè in attesa all'interfaccia con livello inferiore
- lo ASF del client è sempre uguale → esercizio!



# Server concorrente: ASF




- serve ancora la coda? beh, è quella con livello inferiore→ SÌ
- abbiamo messo tutte le transizioni possibili?
  - può ricevere service.Req mentre Idle? dipende… ritrasmissioni, messaggi in ritardo… forse sì!
  - **attenzione a gestire anche situazioni "anomale" dovute a caratteristiche livelli inferiori → gestione eccezioni (o segmentation fault!)**