




Problematiche dei server concorrenti



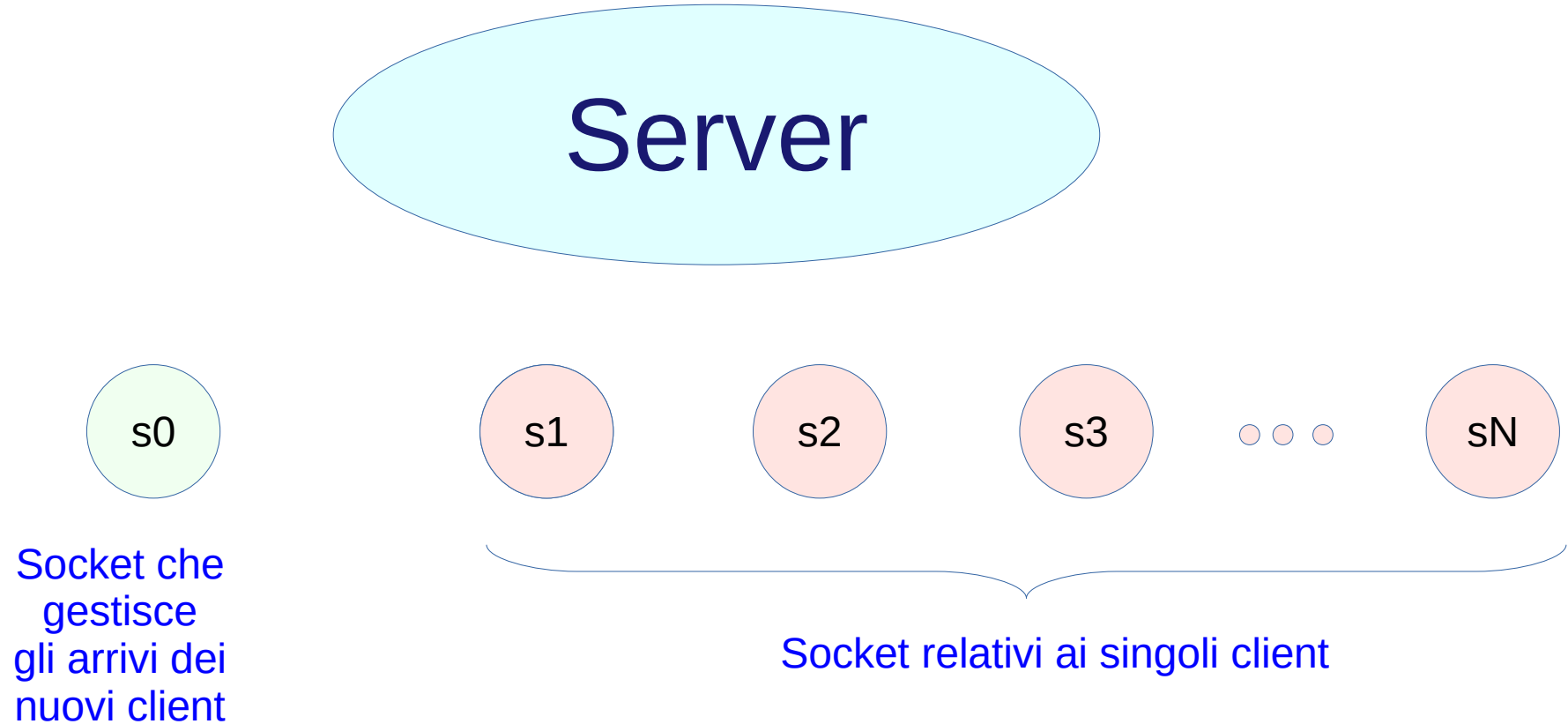
Nel nostri esempi con i server per i numeri primi, la comunicazione per ogni singolo client è la seguente

- il client invia 2 interi (8 byte)
- il server riceve i 2 interi e calcola i primi nell'intervallo
- il server invia al client il numero N di primi trovati (4 byte)
- il server invia N primi (4N byte)
- il client invia la somma dei primi da lui ricevuti

L'esecuzione di `multipclient.py 100 200 -t 10` genera 10 istanze di client che contemporaneamente inviano al server richieste per avere i primi negli intervalli:

[100,200], [1100,1200], ..., [9100,9200]

Server che gestisce connessioni concorrenti





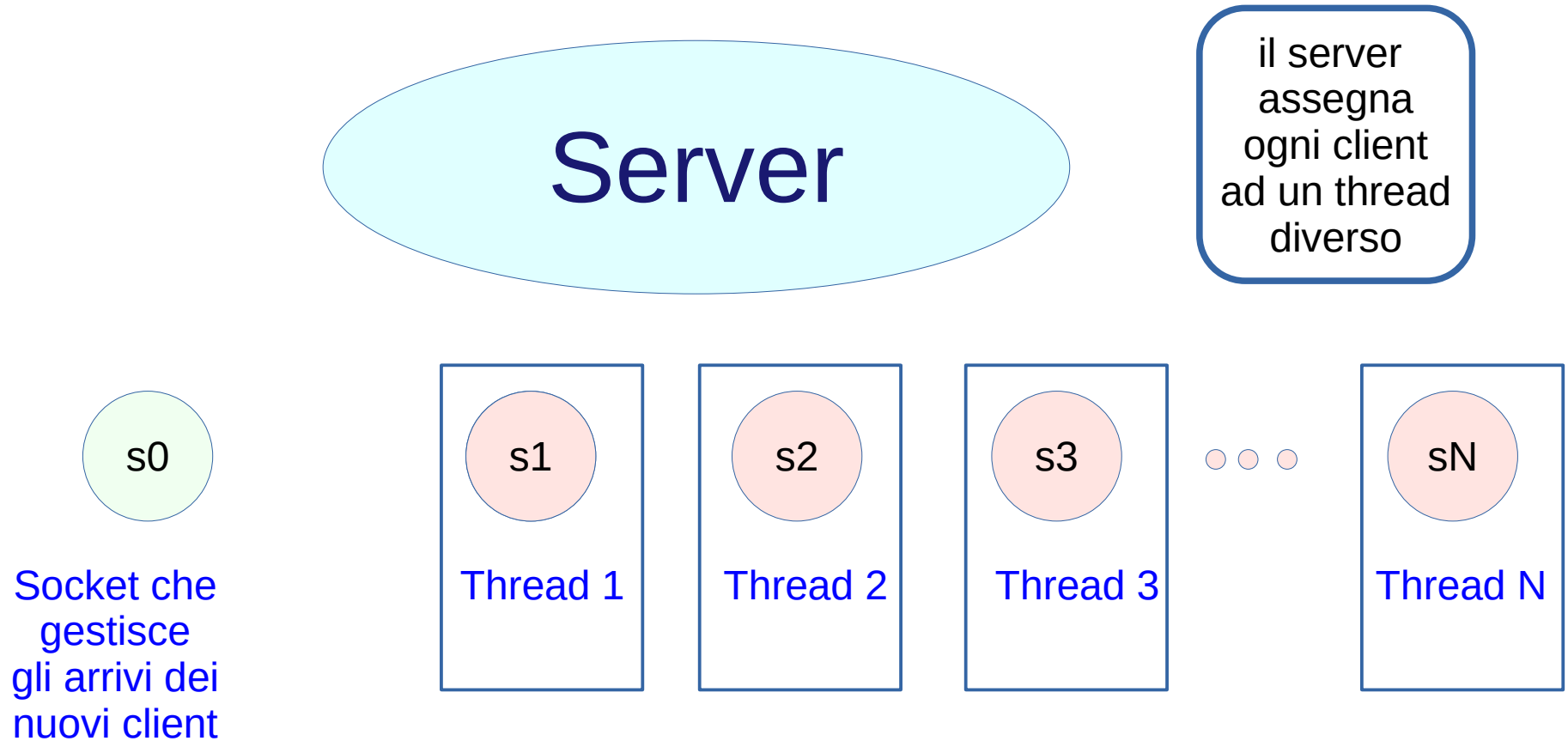
Ricordiamo che le comunicazioni mediante socket possono essere molto lente in quanto inviate attraverso la rete.


E' fondamentale evitare che il server rimanga in attesa di una tale comunicazione “trascurando” altri client che potrebbero essere pronti a inviare/ricevere dati.

Abbiamo due diversi meccanismi per risolvere questo problema:

- “spezziamo” la gestione dei client in tante operazioni R/W e usiamo la `select` per gestire ogni volta quelle che sono pronte a comunicare (`selectserver.py`)
- utilizziamo thread dedicato ad ogni socket (`poolserver.py`)

Struttura usata in `poolserver.py`



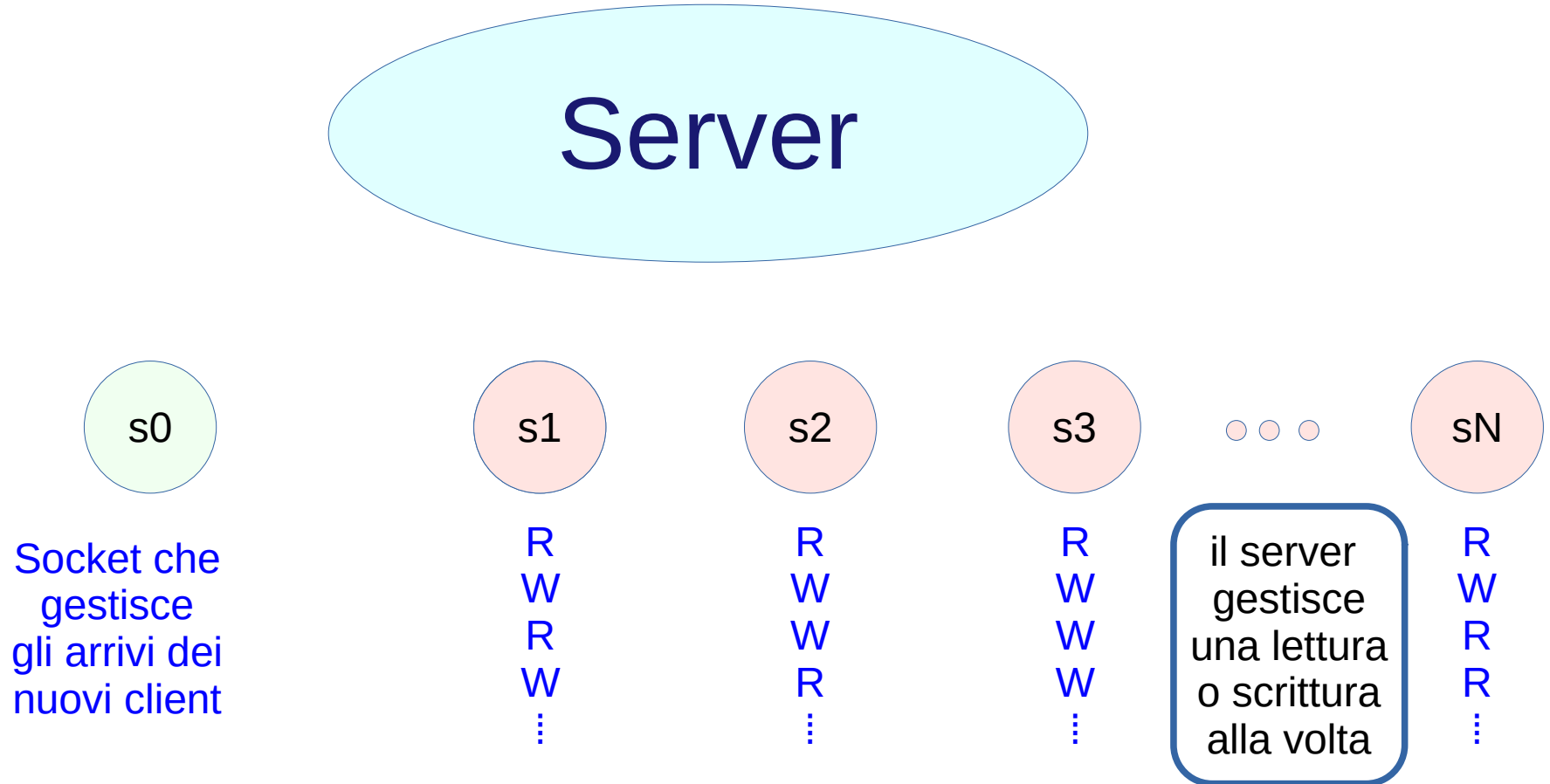


La system call **select** permette al server di gestire più richieste di connessioni contemporaneamente, e quindi più client.

Però quando viene servito un client tutti gli altri vengono ignorati, anche se quel client sta semplicemente attendendo nuovi dati

Per evitare questo “spezziamo” la gestione dei client in tante operazioni R/W e usiamo la **select** per gestire ogni volta quelle che sono pronte

Struttura usata in `selectpserver.py`





L'implementazione di **selectpserver** è basata sul fatto che un client richiede **una** lettura (gli estremi dell'intervallo) e un certo numero di scritture (il numero dei primi e i primi stessi)

Viene utilizzato un dizionario **inuscita** che associa ad ogni socket un elenco di interi ancora da trasmettere

Il server ripete continuamente un ciclo nel quale con la **select** ottiene gli elenchi dei socket da cui posso leggere/scrivere (liste **inready** e **outready**)

Se il socket di un client è in **outready** gli mando **uno** degli interi che gli devo ancora trasmettere (solo uno così non blocco gli altri client)