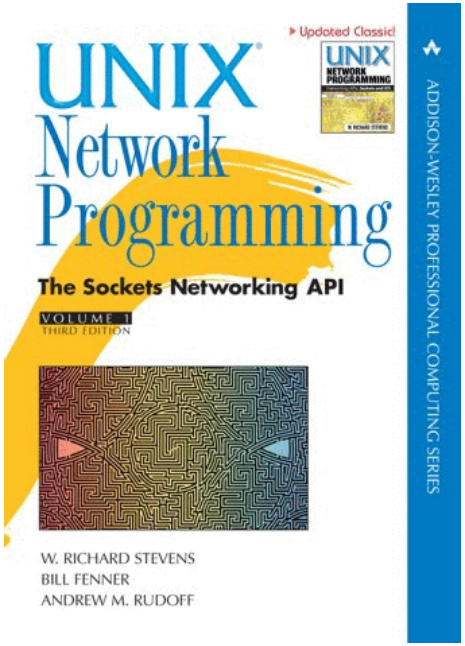
Corso: Reti di Calcolatori - Laboratorio

Docente: Gennaro Oliva

Testo di riferimento

UNIX® Network Programming Volume 1, 3rd ed.: The Sockets Networking API



Applicazioni di rete

- Applicazioni di rete:
  - programmi in esecuzione su macchine differenti che operano in modo indipendente e che possono scambiare informazioni
- Lo scambio di informazioni avviene utilizzando i servizi forniti dal sottosistema di comunicazione

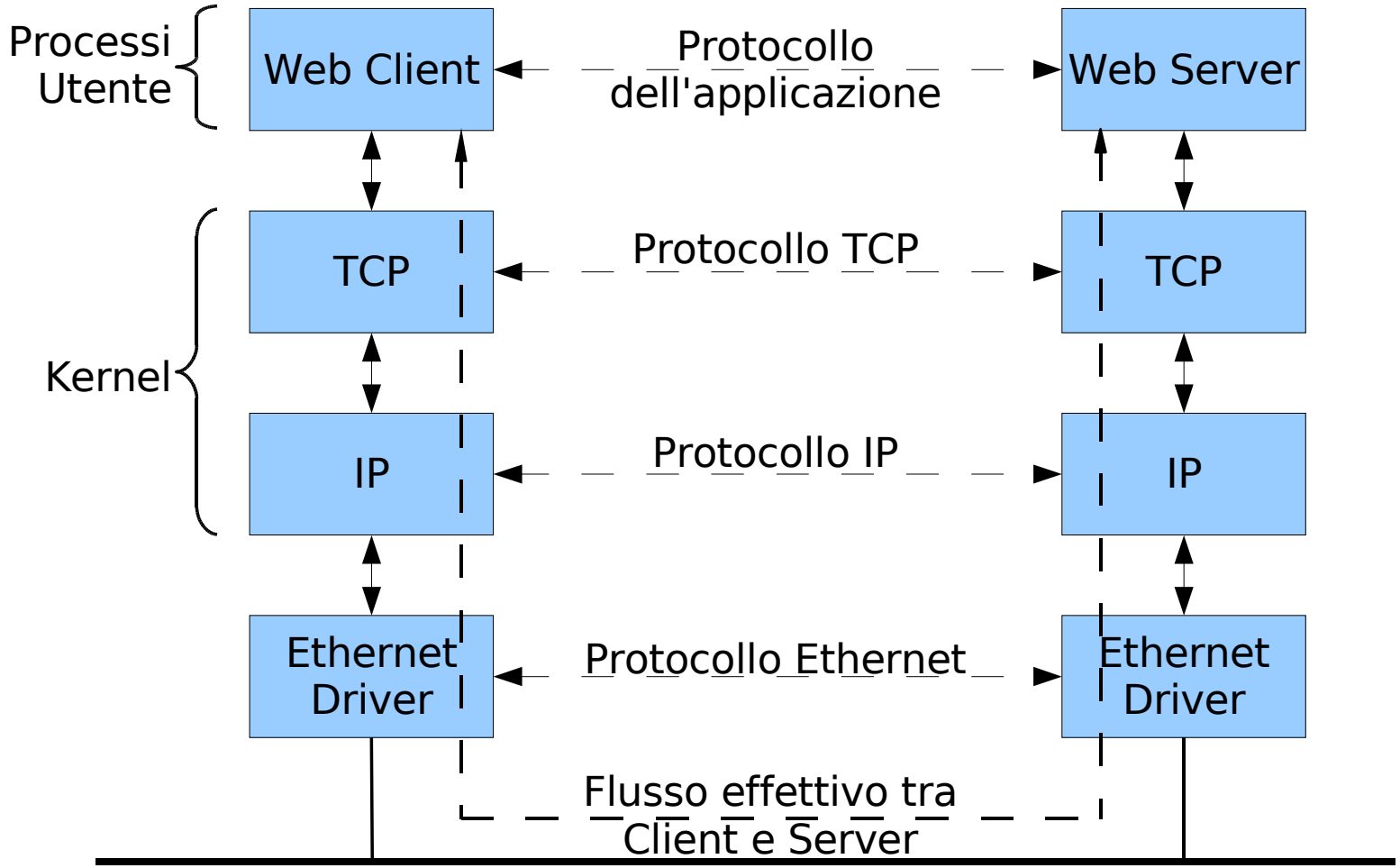
Protocollo

- Le entità conivolte in un'applicazione di rete comunicano secondo un protocollo ben definito:
  - Insieme di regole utilizzate delle entità per scambiarsi informazioni
  - Specifica cosa deve essere comunicato, in che modo e quando

Architettura Protocollare

- Una caratteristica comune dei protocolli di rete è il loro essere strutturati in livelli gerarchici
- Ogni livello utilizza il sottostante
- In questo corso studieremo ed utilizzeremo la Suite di Protocolli Internet

Architettura Protocollare Client-Server



TCP vs UDP

- TCP fornisce un servizio di trasporto affidabile orientato alla connessione
  - TCP consegna alla destinazione una sequenza (stream) di byte identica a quella che il mittente ha prodotto
- UDP Fornisce un servizio di trasporto orientato ai datagrammi non connesso e inaffidabile
  - Non c'e' nessuna garanzia che il pacchetto arrivi a destinazione o che l'ordine dei pacchetti sia rispettato

Modelli di programmazione

- Principali modelli di programmazione per apllicazioni di rete
  - client–server
  - peer–to-peer
  - three-tier

Modello Client-Server

- Nel modello client-server si distinguono due categorie di soggetti
  - I programmi che forniscono un servizio, chiamati server
  - I programmi di utilizzo, detti client che effettuano le richieste
- Un server può (di norma deve) essere in grado di rispondere a più di un client

Modello Client-Server

- Distinguiamo due classi di server:
  - concorrenti
  - iterativi
- Seguono questo modello tutti i servizi fondamentali di internet:
  - pagine web, ftp, telnet, ssh, etc.

Identificazione del Server

- Ogni qual volta un client deve comunicare con un server deve identificarlo univocamente
- Tale identificazione avviene attraverso due livelli di indirizzamento:
  - Il primo determina l'host su cui e' in esecuzione il processo server
  - Il secondo determina il processo server con cui si vuole comunicare

Identificazione del Server

- Ogni qual volta un client deve comunicare con un server deve identificarlo univocamente
- Tale identificazione avviene attraverso due livelli di indirizzamento:
  - Il primo determina l'host su cui e' in esecuzione il processo server
  - Ad ogni host di una rete IP e' associato un indirizzo IP
  - Il secondo determina il processo server con cui si vuole comunicare

Indirizzi IP

- Un Indirizzo IP è un numero che identifica univocamente un dispositivo collegato ad una rete informatica che utilizza lo standard IP (Internet Protocol)
- Gli indirizzi IPv4 sono costituiti da 32 bit (4 byte), e vengono descritti con 4 numeri decimali rappresentati su 1 byte (quindi ogni numero varia tra 0 e 255) separati da un punto

Indirizzi IP

- Un esempio di indirizzo IPv4 è il seguente: 192.167.11.34
- Questa rappresentazione limita lo spazio di indirizzamento a 4,294,967,296 indirizzi univoci possibili
- La rete internet esclude 18.000.000 indirizzi utilizzati per le reti private
- Per ovviare al problema della mancanza di indirizzi IP dovuta alla costante crescita di Internet è stato introdotto l'IPv6

Indirizzi IP

- Per conoscere l'indirizzo ip della vostra postazione usate il comando

  /sbin/ifconfig

Identificazione del server

- Ogni qual volta un client deve comunicare con un server deve identificarlo univocamente
- Tale identificazione avviene attraverso due livelli di indirizzamento:
  - Il primo determina l'host su cui e' in esecuzione il processo server
  - Ad ogni host di una rete IP e' associato un indirizzo IP
  - Il secondo determina il processo server con cui si vuole comunicare
  - Ad ogni applicazione server in esecuzione su un host e' associato un numero di porta

Numeri di Porta

- In un ambiente multitasking più processi in esecuzione su uno stesso host devono poter comunicare mediante lo stesso sottosistema di rete
- E' necessario consentire più connessioni simultaneamente
- Per poter tenere distinte le diverse connessioni su uno stesso host si utilizzano le porte

Numeri di Porta

- Le porte sono interi a 16 bit
  da 0 a 65535
- Da 0 a 1023: porte riservate (ai processi di root)
- Da 5000 a 32768: porte utente
- Altre: porte effimere (per i client, ai quali non interessa scegliere una porta specifica)

Porte Riservate

- Esempi di porte riservate
- 21 ftp (trasferimento file)
- 22 ssh (login remoto sicuro)
- 25 smtp (invio email)
- 80 http (web)
- 143 imap (lettura email)
- Lista ufficiale su: http://www.iana.org/
- La corrispondenza tra nomi simbolici e numeri si trova nel file /etc/services

Identificare una Comunicazione

- La classica implementazione di un server concorrente prevede ad ogni nuova richiesta client venga generato un processo che la gestisce
- Come si distinguono processi server che forniscono uno stesso servizio a client diversi se utilizzano la stessa porta per comunicare e sono in esecuzione sullo stesso host?

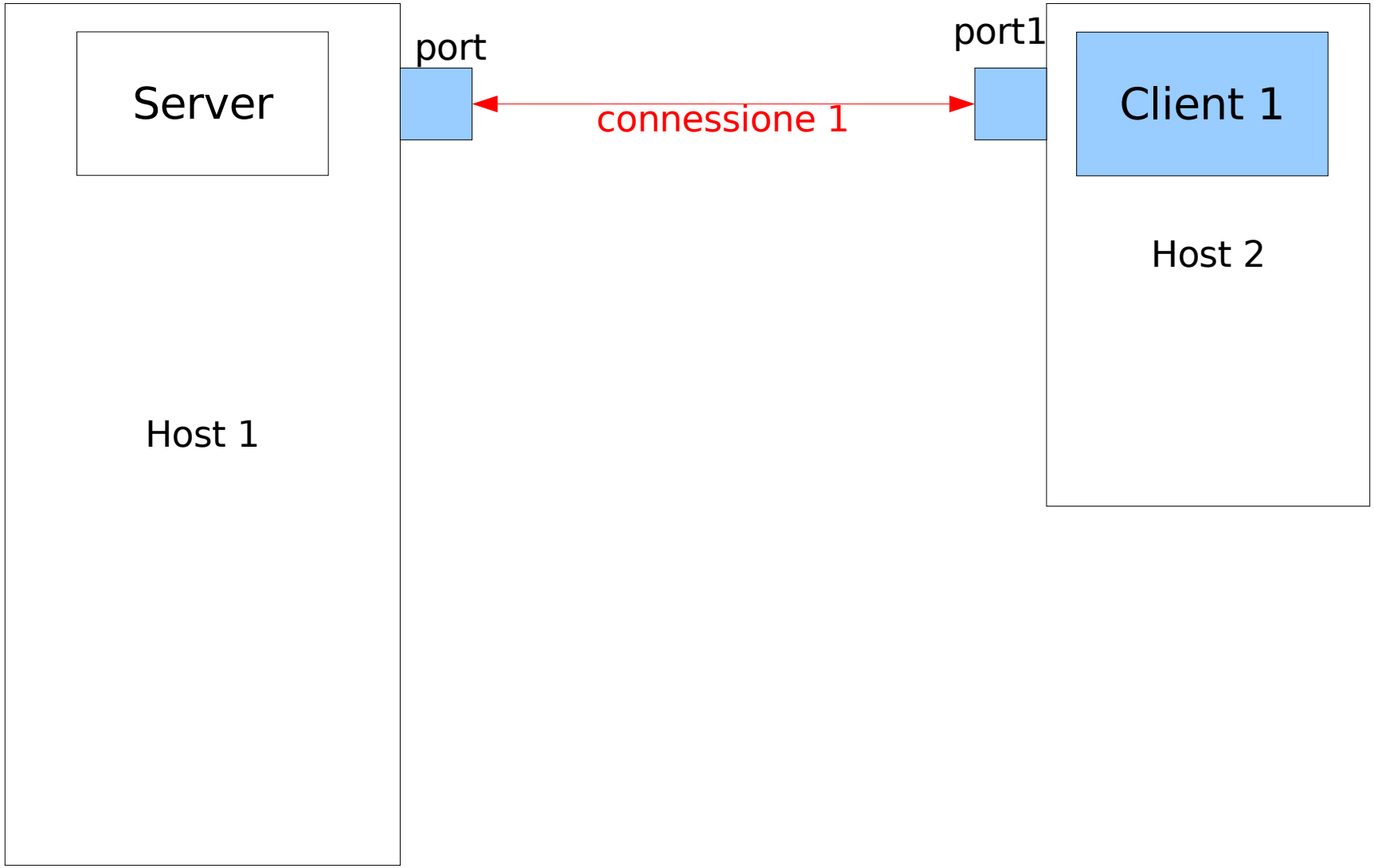
Identificare una Comunicazione

- TCP e UDP usano 4 informazioni per identificare una comunicazione
  - Indirizzo IP del server
  - Numero di porta del servizio lato server
  - Indirizzo IP del client
  - Numero di porta del servizio lato client

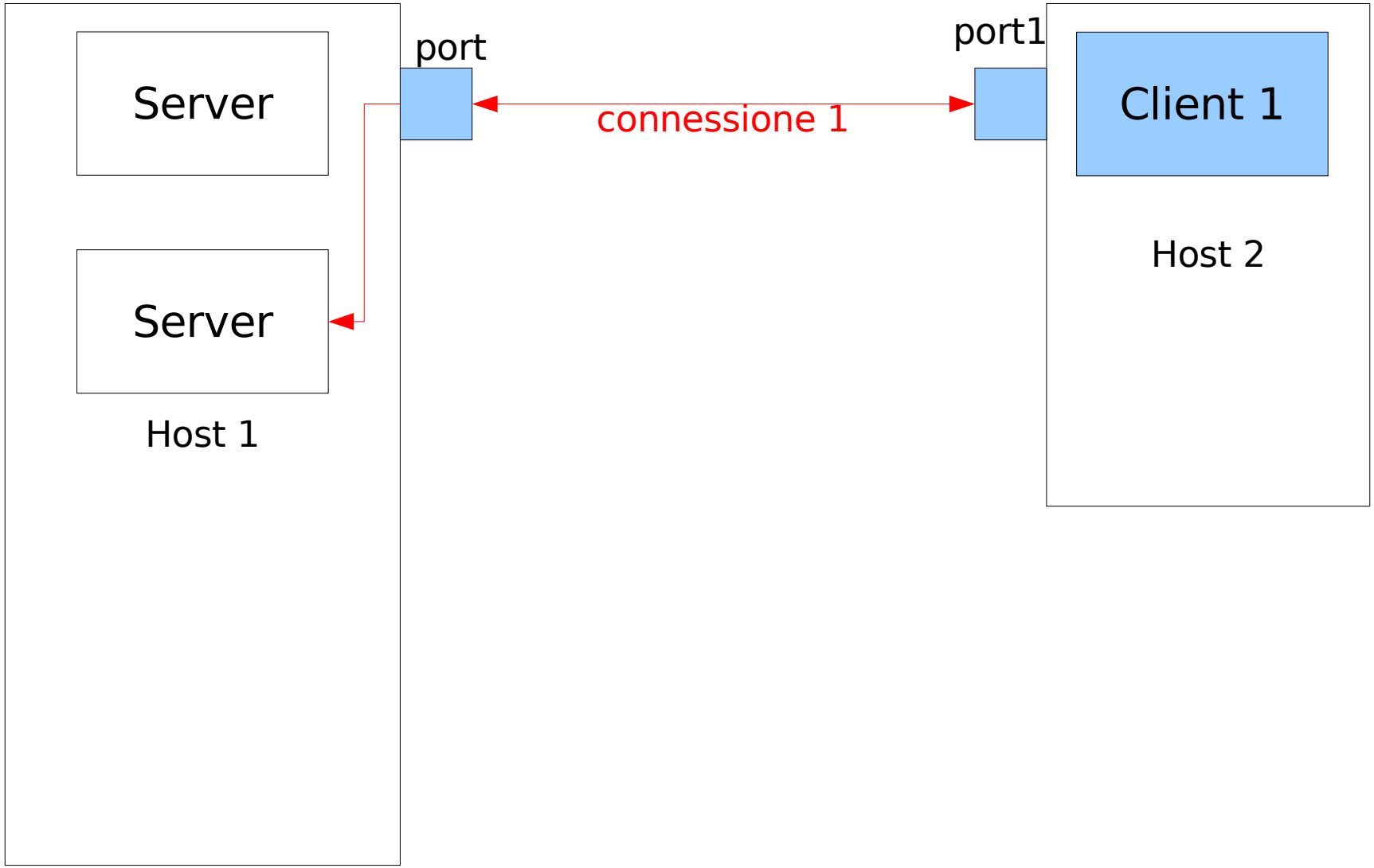
Endpoint

- Un endpoint è una coppia
  - (indirizzo IP, porta)
- Una connessione è una coppia di endpoints
  - (endpoint sorgente,endpoint destinazione)

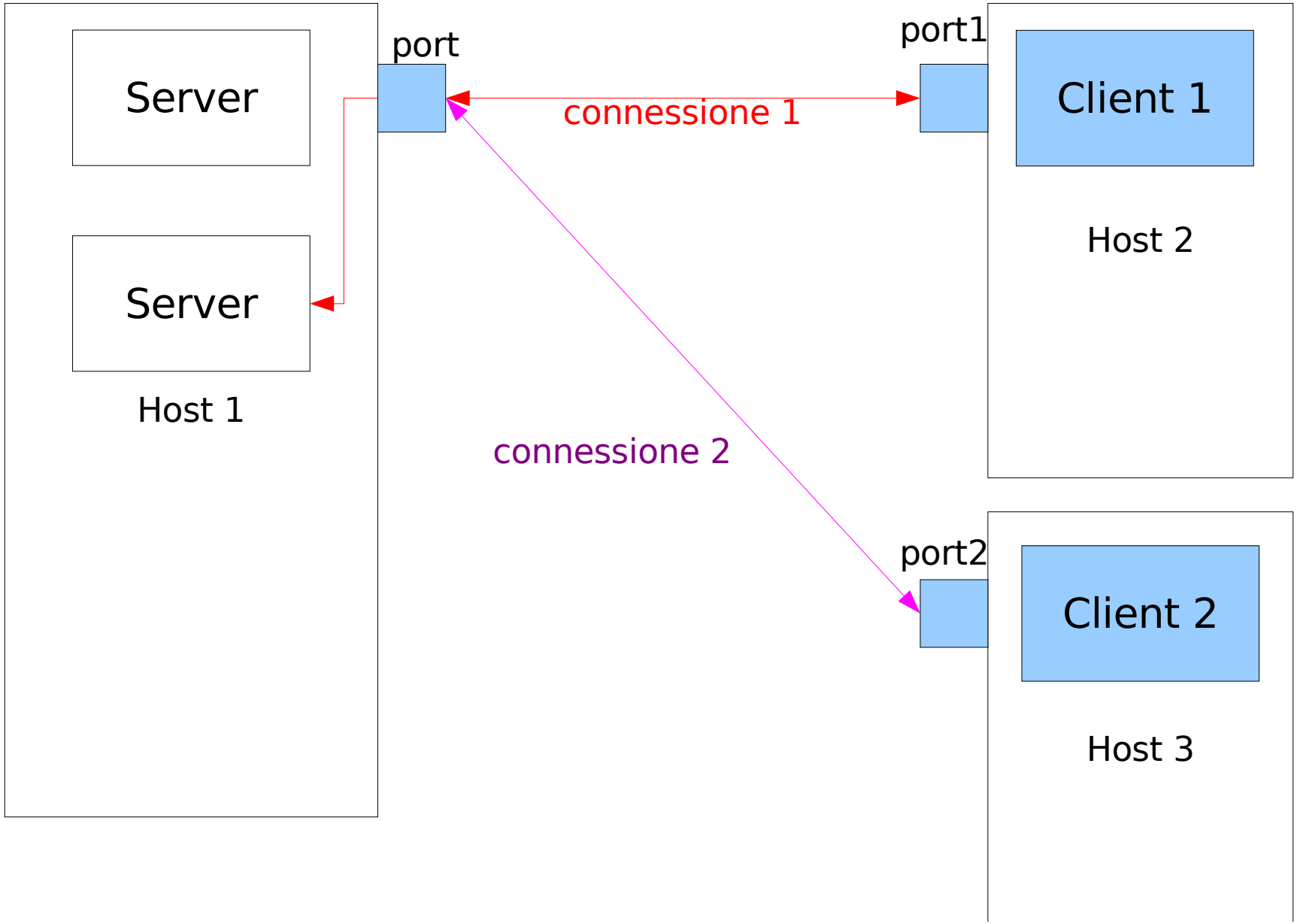
Server Concorrenti



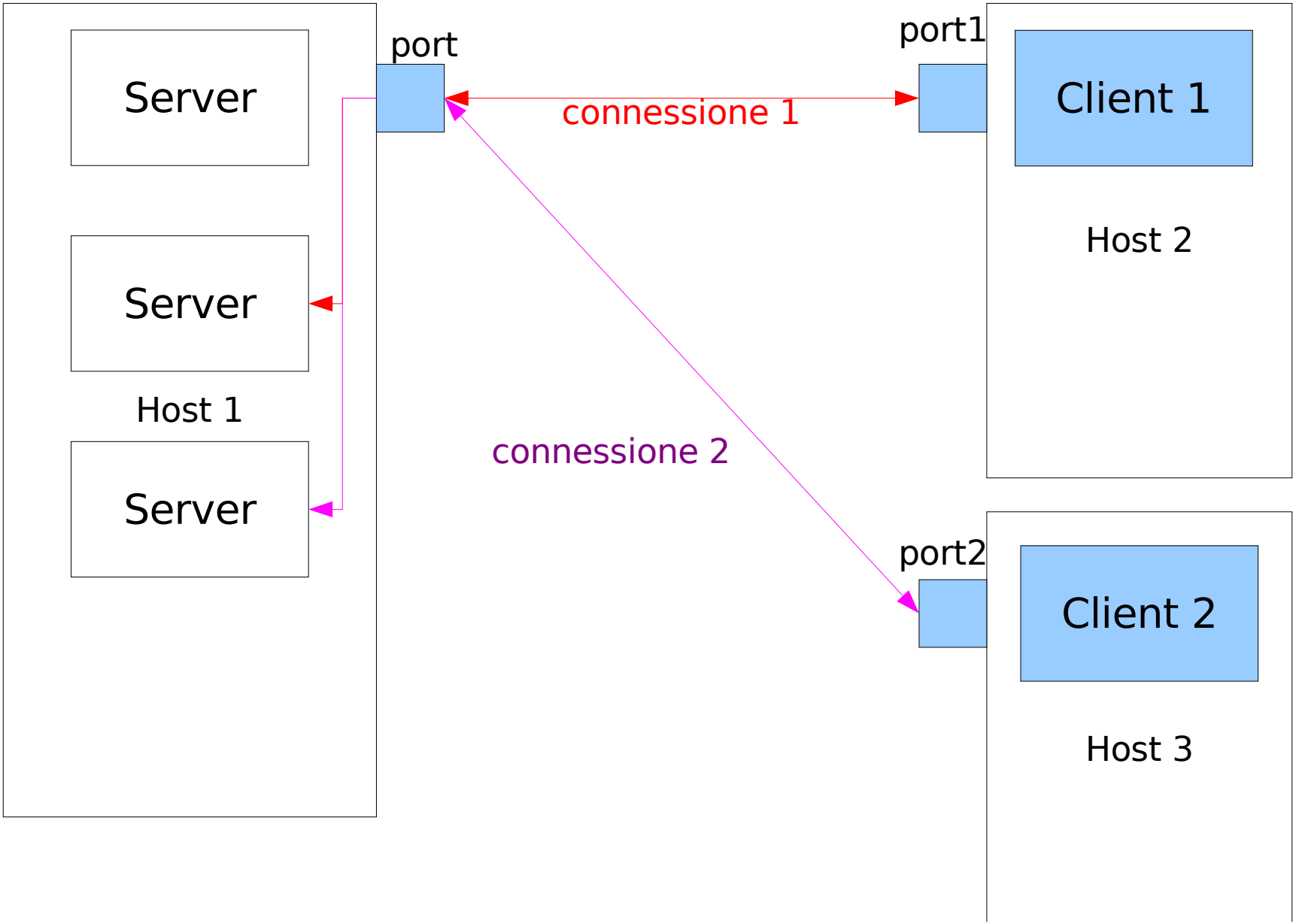
Server Concorrenti



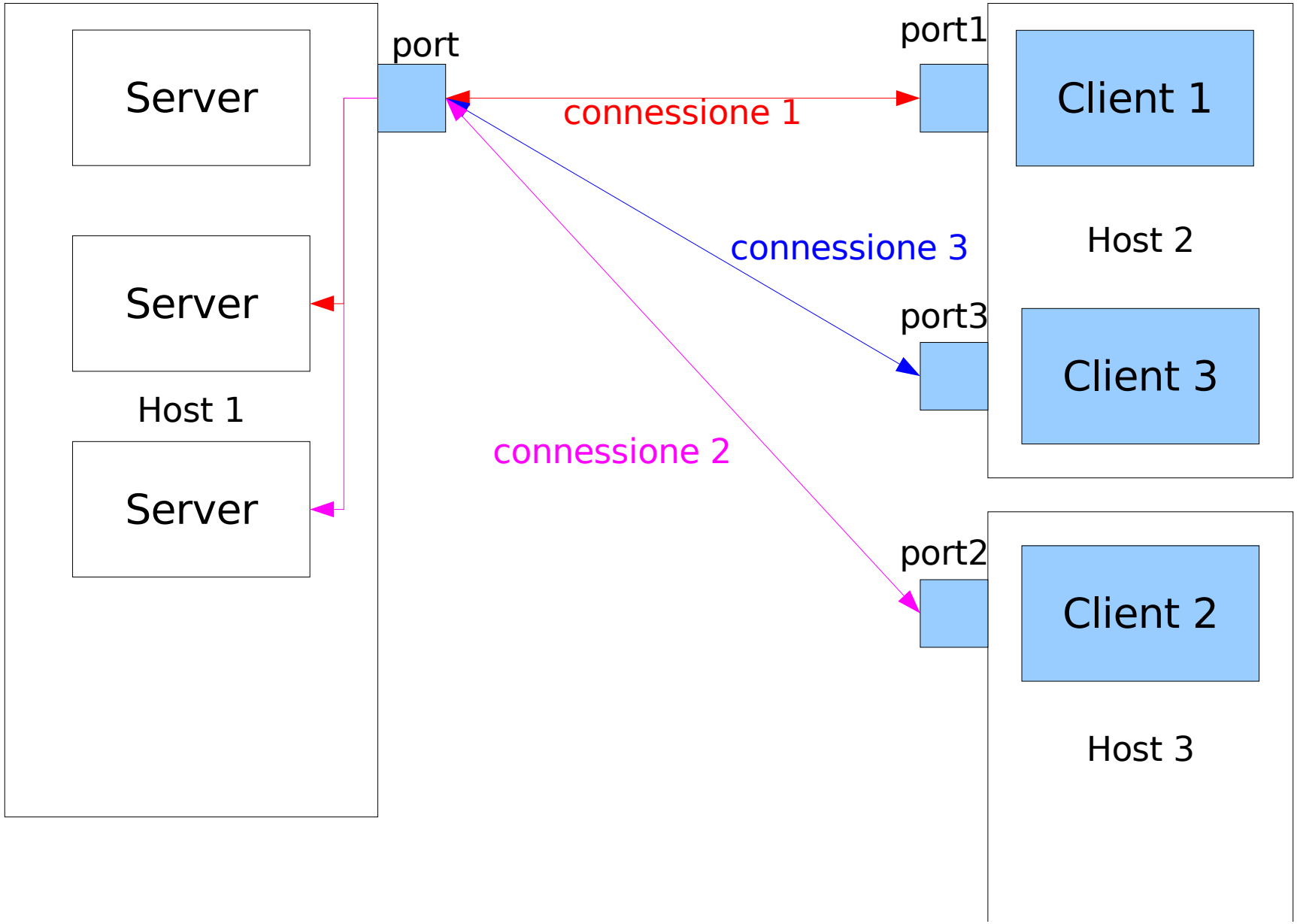
Server Concorrenti



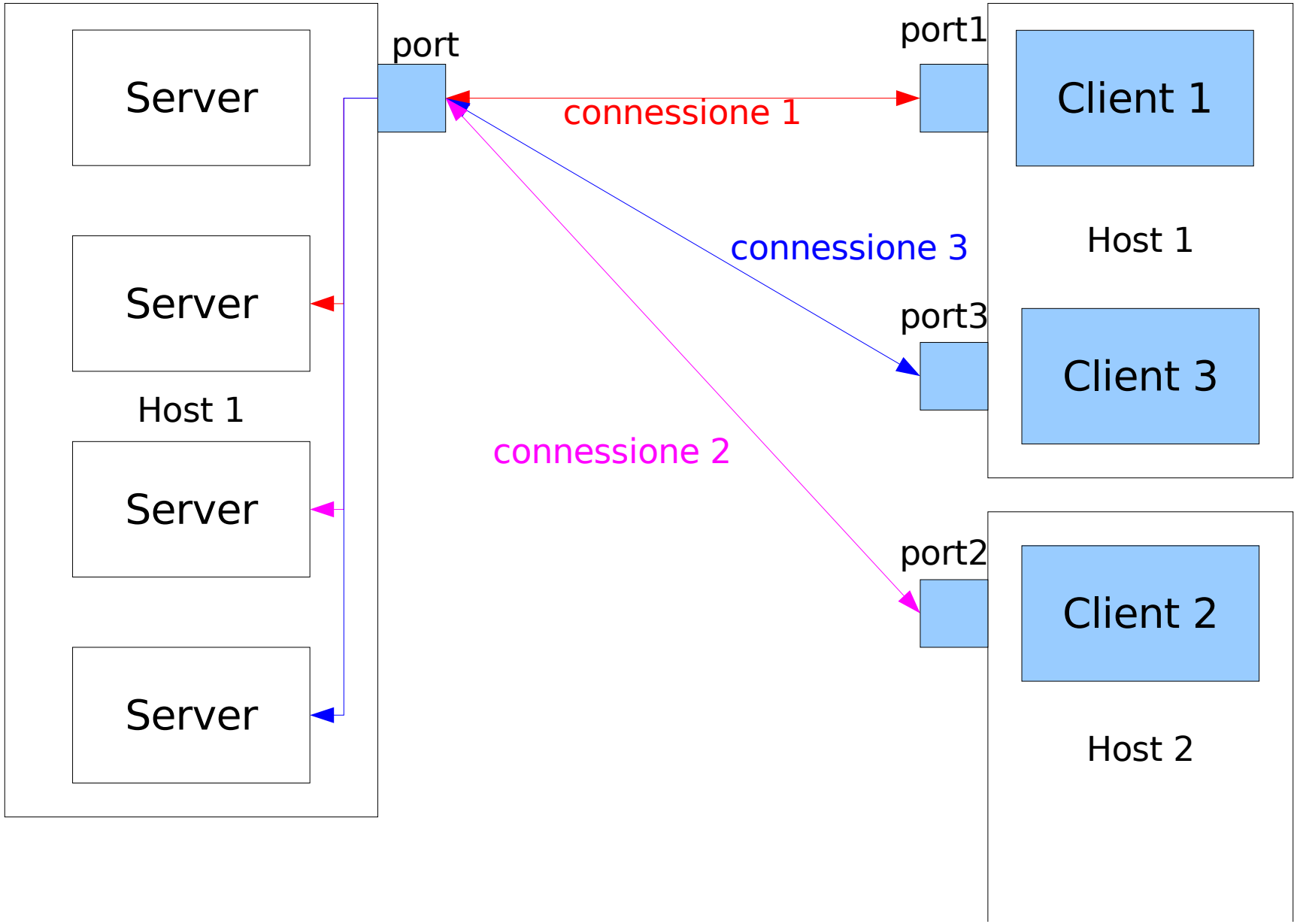
Server Concorrenti



Server Concorrenti



Server Concorrenti



Berkeley Sockets

- I socket di Berkeley sono un'API che definisce una libreria C per comunicazioni inter-processo anche su rete
- Introdotti con la versione 4.2 di BSD Unix (nel 1983)
- Sono lo standard de facto per la realizzazione di applicazion di rete

Struttura di un'applicazione client elementare

- Crea il socket
- Si connette ad un server
- ...
- Chiude il socket

- socket(...)
- connect(...)
- ...
- close(...)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main(int argc, char **argv)
{
  int               sockfd, n;
  char              recvline[1025] ;
  struct sockaddr_in servaddr;
  if (argc != 2) {
    fprintf(stderr,"usage: %s <IPaddress>\n",argv[0]);
    exit (1);
  }
  if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    fprintf(stderr,"socket error\n");
    exit (1);
  }
    servaddr.sin_family = AF_INET;
    servaddr.sin_port   = htons(13);
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
      fprintf(stderr,"inet_pton error for %s\n", argv[1]);
      exit (1);
    }
  if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    fprintf(stderr,"connect error\n");
    exit(1);
  }
  while ( (n = read(sockfd, recvline, 1024)) > 0) {
    recvline[n] = 0;
    if (fputs(recvline, stdout) == EOF) {
      fprintf(stderr,"fputs error\n");
      exit(1);
    }
  }
  if (n < 0) {
    fprintf(stderr,"read error\n");
    exit(1);
  }
  exit(0);
}
```

Socket

int socket(int famiglia, int tipo, int protocollo);

Famiglia	Scopo	man
PF_UNIX,PF_LOCAL	Local communication	unix(7)
PF_INET	IPv4 Internet protocols	ip(7)
PF_INET6	IPv6 Internet protocols	
PF_IPX	IPX - Novell protocols	
PF_NETLINK	Kernel user interface device	netlink(7)
PF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
PF_AX25	Amateur radio AX.25 protocol	
PF_ATMPVC	Access to raw ATM PVCs	
PF_APPLETALK	Appletalk	ddp(7)
PF_PACKET	Low level packet interface	packet(7)

I vari formati sono definiti in <sys/socket.h>

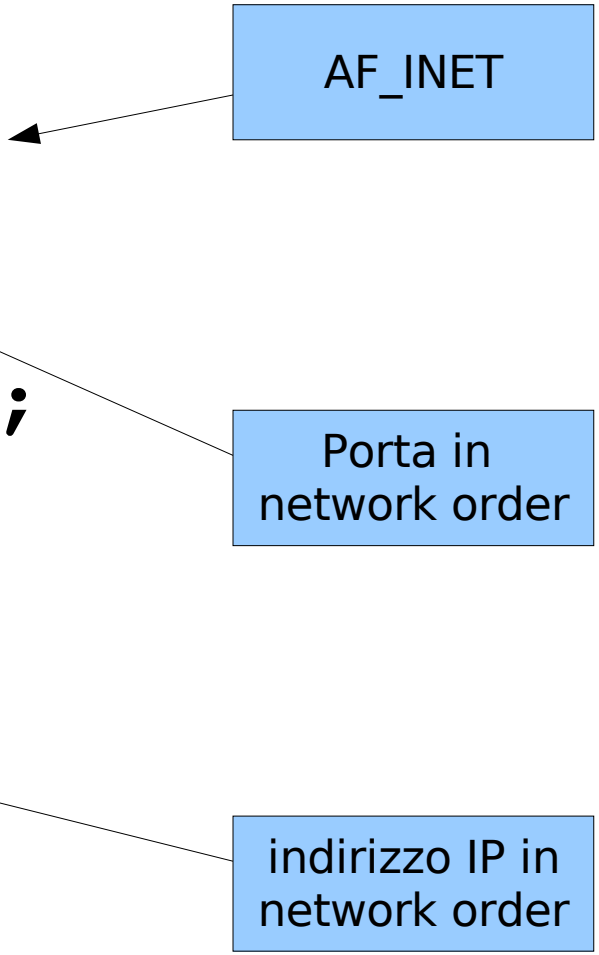
Socket - tipo

int socket(int famiglia, int tipo, int protocollo);

- SOCK_STREAM canale bidirezionale, sequenziale affidabile che opera su connessione. I dati vengono ricevuti e trasmessi come un flusso continuo
- SOCK_DGRAM usato per trasmettere pacchetti di dati di lunghezza massima prefissata (datagram), indirizzati singolarmente senza connessione in maniera non affidabile.
- SOCK_SEQPACKET canale bidirezionale, sequenziale e affidabile che opera su connessione. I dati vengono trasmessi per pacchetti di dimensione massima fissata e vanno letti integralmente
- SOCK_RAW canale di basso livello per accedere ai protocolli di rete e alle varie interfacce. Solitamente non utilizzato dalle applicazioni
- SOCK_RDM canale di trasmissione di dati affidabile in cui non è garantito l'ordine di arrivo dei pacchetti.
- SOCK_PACKET Obsoleto

Indirizzo TCP/IP del server

```
struct sockaddr_in {
sa_family_t sin_family;
u_int16_t sin_port;
struct in_addr sin_addr;
};
struct in_addr {
u_int32_t s_addr;
};
```



Codifica dati

- Come viene memorizzato un dato superiore al byte?
- Si consideri un intero di costituito da 4 byte
- consideriamo 65537, cioe' $2^{16} + 1$, la codifica “Big Endian” (prima il byte piu' significativo):

00000000 00000000 10000000 00000001

- la codifica “Little Endian” (prima il byte meno significativo):

00000001 10000000 00000000 00000000

Codifica dati

- L'utilzzo di una codifica è stabilito dall'architettura del processore
- La suite di protocolli internet utilizza big endian pertanto sono necessarie funzioni di conversione
- conversioni tra unsigned:

- – #include <netinet/in.h>
- – uint32_t htonl(uint32_t x)
- – uint16_t htons(uint16_t x)
- – uint32_t ntohl(uint32_t x)
- – uint16_t ntohs(uint16_t x)

h = Host
n = Network (big endian)
l = Long (4 bytes)
s = Short (2 bytes)

Porta associata al servizio

servaddr.sin_port   = htons(13);

- L'istruzione memorizza nel campo sin_port della struct serveaddr l'intero 13 scritto in network order
- 13 e' la porta su cui risponde il server che stiamo contattando

Conversione dell'IP del server

inet_pton(PF_INET, argv[1], &servaddr.sin_addr)

- Questa funzione converte la stringa passata come secondo argomento in un indirizzo di rete scritto in network order e lo memorizza nella locazione di memoria puntata dal terzo argomento
- Il nostro programma quindi dovra' specificare come argomento l'indirizzo IP del server a cui fare la richiesta
- La funzione restituisce un numero negativo o zero in caso di errore ed un numero positivo in caso di successo

Connect

```
connect(sockfd, (struct sockaddr *) &servaddr,
   sizeof(servaddr))
```

Connette il socket sockfd all'indirizzo serv_addr

- Il terzo argomento e' la dimensione in byte della struttura
- Il cast e' necessario in quanto la funzione puo' essere utilizzata con diversi tipi di socket e quindi con diversi tipi di strutture
- Restituisce 0 (successo) oppure -1 (errore)

Stampa del messaggio

```
while ( (n = read(sockfd, recvline, 1024)) > 0) {
 recvline[n] = 0;
 if (fputs(recvline, stdout) == EOF) {
   fprintf(stderr,"fputs error\n");
   exit(1);
 }
```

Prima di terminare il programma stampa a video il messaggio ricevuto dal server

Per leggere e scrivere su socket si utilizzano le system call read e write