Programmazione e Architetture (Modulo B)

Lezione 21
Socket e programmazione di rete

Luca Manzoni

Cosa sono i socket?

Astrarre la comunicazione di rete

- Una astrazione molto comune per la programmazione di rete è il **socket**
- Un socket astrae la comunicazione fornendo un file descriptor su cui è possibile fare operazioni di read e write…
- …che leggono e scrivono “sulla rete”
- Non è l’unico modo ma è molto diffuso, fu introdotto per la prima volta in 4.3BSD UNIX
- Oggi usabile su sistemi Unix, Linux, macOS, Windows, etc.

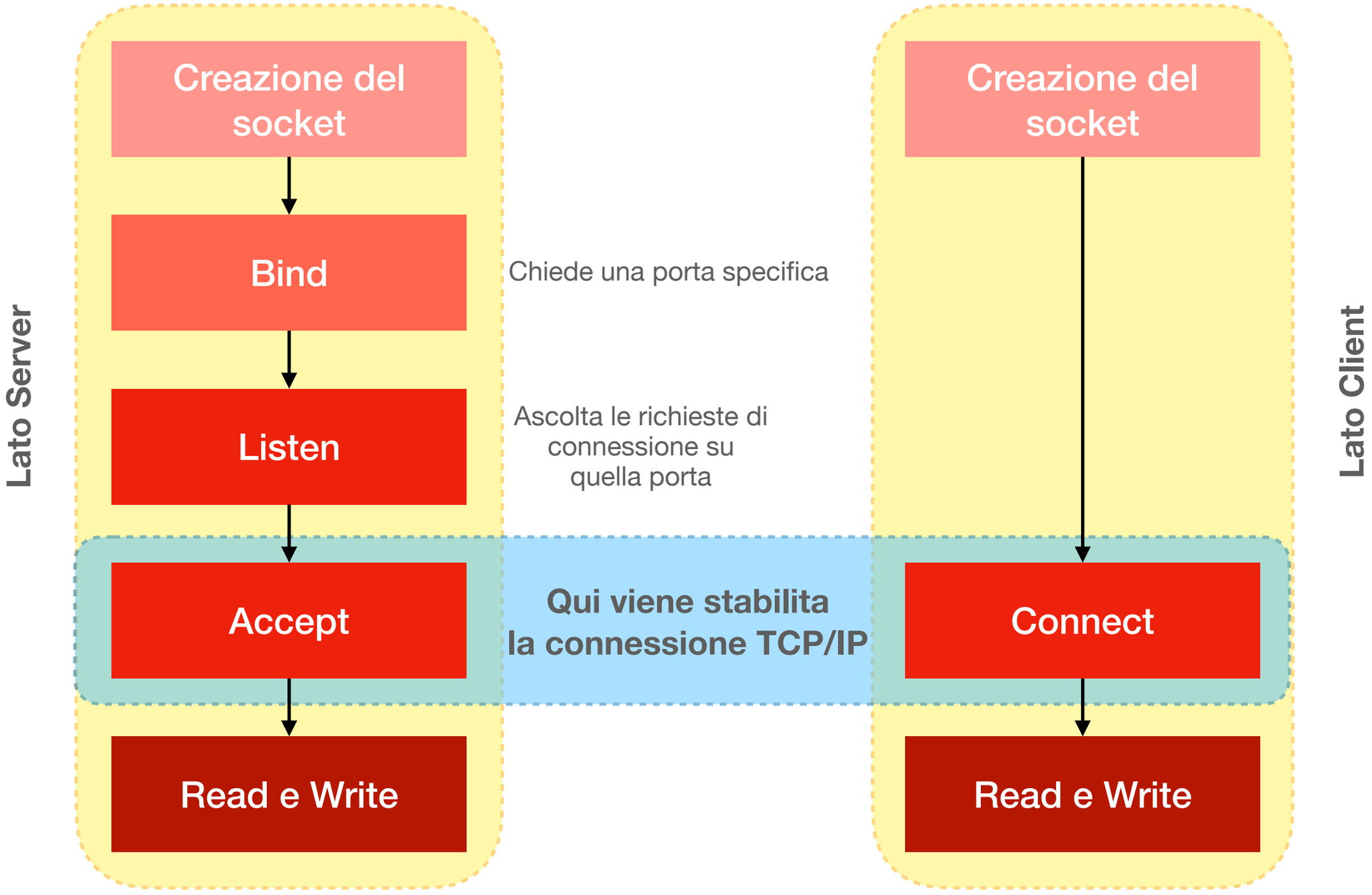
Cosa sono i socket?

Astrarre la comunicazione di rete

- Vi sono due tipi principali di socket:
  - *Stream*. Forniscono un circuito virtuale
  - *Datagram*. Lavorano a livello di singoli pacchetti.
- I socket sono in generale indipendenti dal protocollo, possono usare TCP/IP o UDP/IP ma non sono limitati ai protocolli di Internet
- E.g., famiglie di protocolli per
  comunicazione locale (PF_UNIX/PF_LOCAL)
  comunicazioni via internet (PF_INET)
  e usando IP versione 6 (PF_INET6)

Socket: ciclo di vita

Lato server e client



Creare un socket

In C

```
#include <sys/types.h>
#include <sys/socket.h>
```

PF_INET per IPv4
(a volte AF_INET, sono equivalenti)

```
int fd = socket(family, type, protocol);
```

SOCK_STREAM per TCP
SOCK_DGRAM per UDP

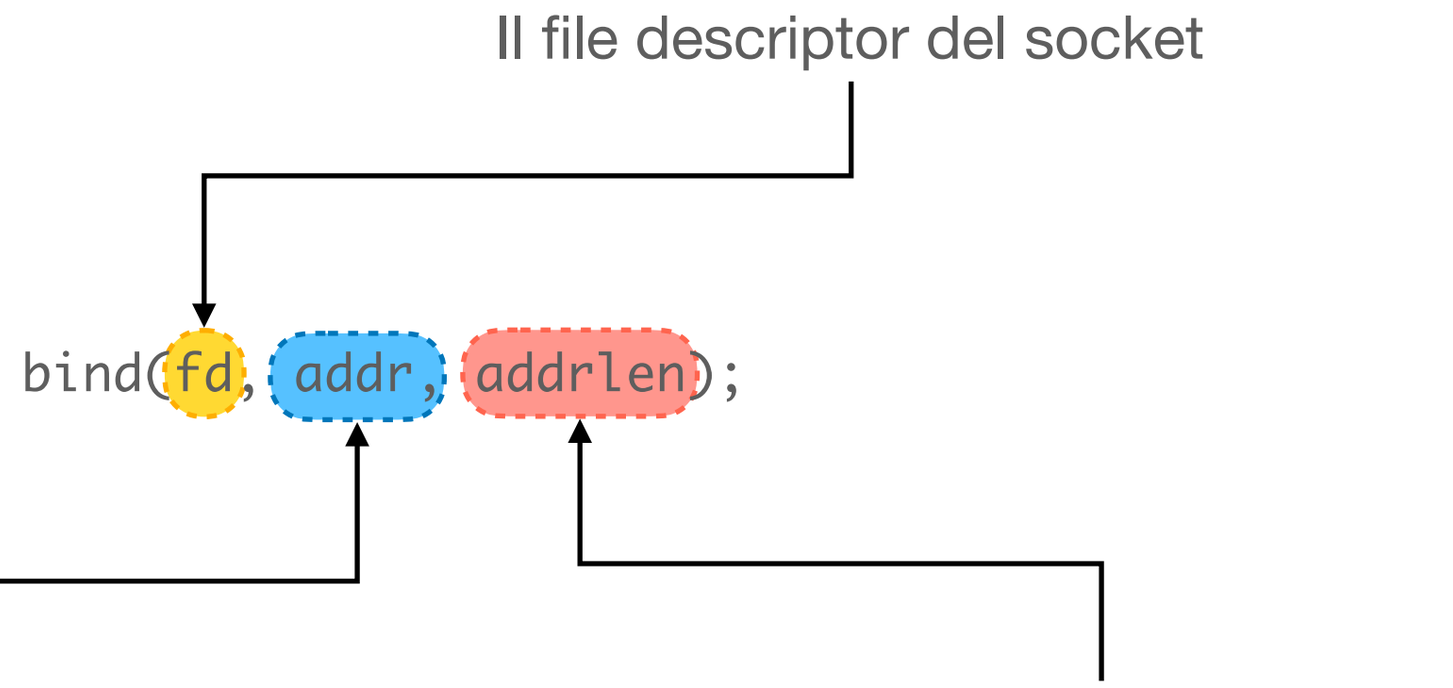
0 se si usa PF_INET

Viene ritornato -1 in caso di errori

Bind

In C (lato server)

Quando viene creato un socket è necessario associargli un indirizzo.
Questo avviene chiamando la funzione *bind*

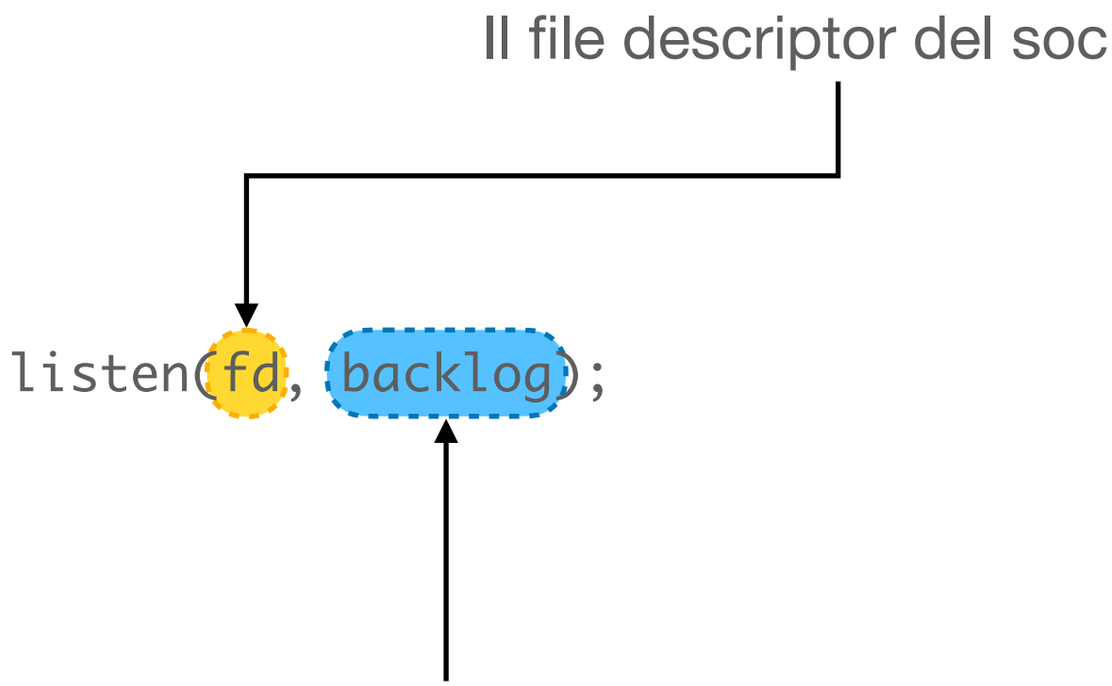


Viene ritornato -1 in caso di errori

Ascoltare per connessioni in ingresso

In C (lato server)

Possiamo ora attendere che vengano stabilite delle connessioni “ascoltando” su un dato socket

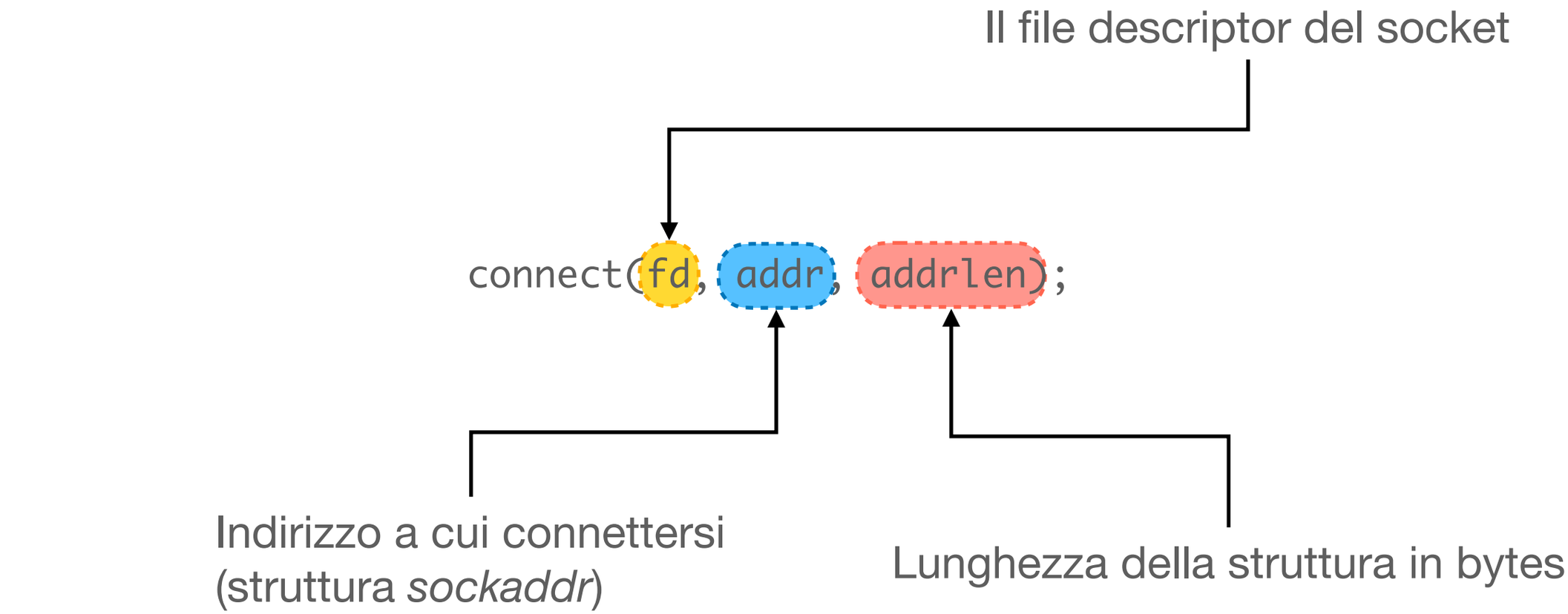


Viene ritornato -1 in caso di errori

Stabilire una connessione

In C (lato client)

Possiamo connetterci a un server che sta ascoltando su un dato indirizzo e porta usando la funzione *connect*.



Viene ritornato -1 in caso di errori

Accettare connessioni

In C (lato server)

Mentre un socket ascolta su una data porta possiamo accettare connessioni in ingresso con *accept*.
La chiamata ad *accept* ritorna quando abbiamo stabilito una connessione.
Viene ritornato un nuovo file descriptor

Il file descriptor del socket
su cui stiamo ascoltando

```
int connfd = accept(fd, (struct sockaddr *) &cliaddr, &cliaddrlen);
```

Puntatore ad una struttura di tipo sockaddr in cui sono salvati i dati del client che si è connesso

Puntatore a un intero in cui salvare la dimensione della struttura *sockaddr*

Viene ritornato -1 in caso di errori

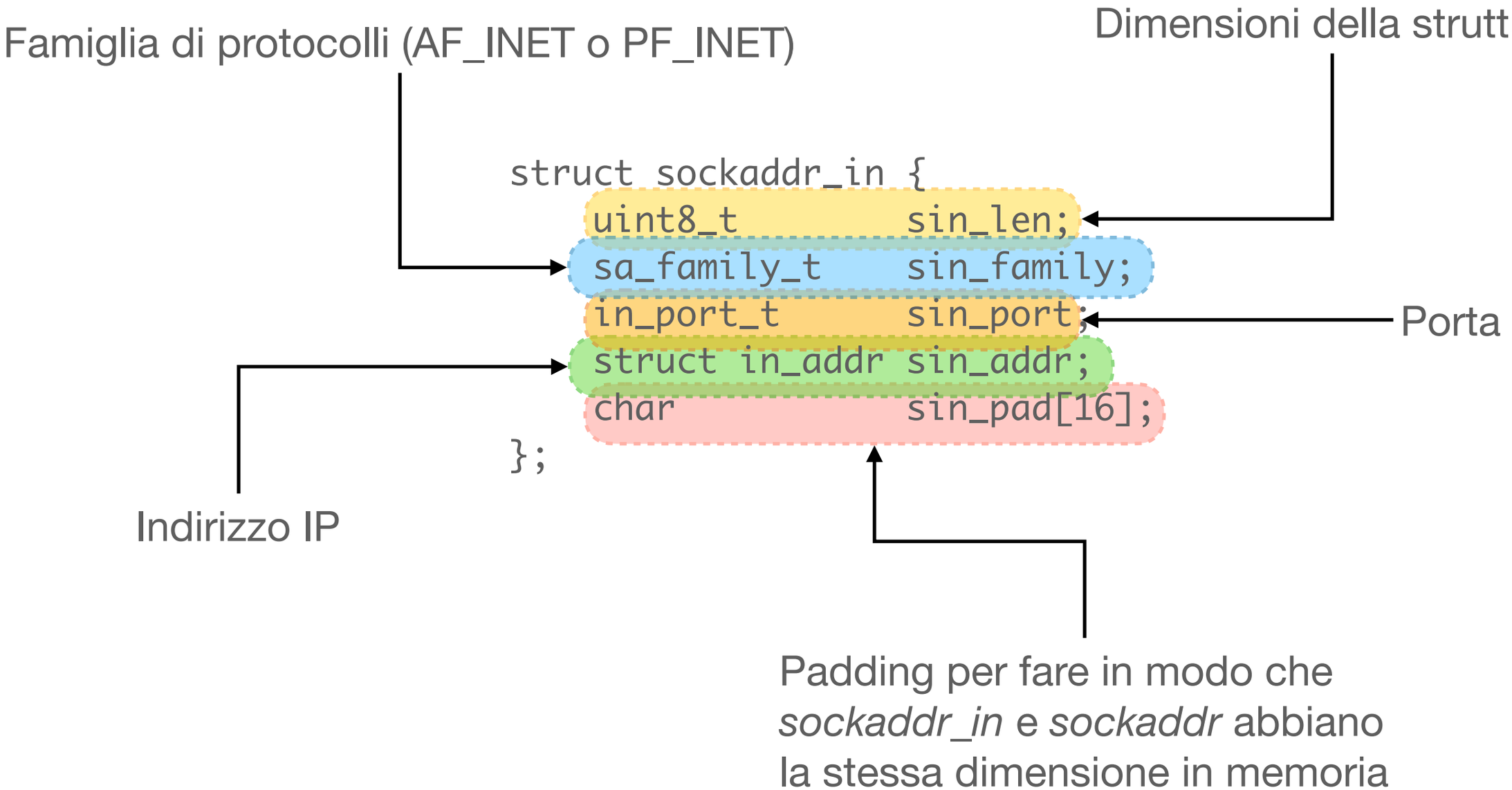
La struttura sockaddr

E, per IPv4, sockaddr_in

- La struttura sockaddr tratta l'indirizzo come una sequenza opaca di byte
- Esistono strutture specifiche per i diversi protocolli che:
  - Forniscono campi specifici (e.g., porte, indirizzi IP, etc)
  - Hanno la stessa dimensione di sockaddr e possiamo fare casting da e verso sockaddr (con le dovute precauzioni)
  - Per IP si usa *sockaddr_in*

Indirizzi IPv4

Usando sockaddr_in



Creazione di un indirizzo

Indirizzo IP e porta

- `inet_pton()` permette di convertire una stringa in notazione dotted decimal
- `inet_pton(AF_INET, “127.0.0.1”, &addr.sin_addr);`
  dove addr è una struttura sockaddr_in
- `htos()` permette di convertire un intero nel numero di porta corrispondente
- `addr.sin_port = htos(80);`
  dove 80 è il numero di porta (ricordate che è di 16 bit, quindi al massimo 65535)

Creazione di un indirizzo

Indirizzo IP e porta

- Sul server non dobbiamo dire a che indirizzo connetterci ma su che indirizzo ascoltare
- Possiamo usare la costante INADDR_ANY per ascoltare su tutte le interfacce disponibili
- Esiste anche la possibilità di usare il DNS per non dover usare direttamente l'IP, non vedremo questa opzione
- È necessario importare <netinet/in.h> e <arpa/inet.h>

Lettura e scrittura

Uso di read e write

- Read e Write funzionano come per le pipe, è necessario specificare:
  - Il file descriptor su cui leggere/scrivere
  - Il buffer in cui ci sono i dati da scrivere/in cui salvare i dati da leggere (ricordate che non sono terminati con null quando li ricevete)
  - La dimensione del buffer di ricezione o il numero di byte da scrivere
- Al termine delle operazioni ricordate di chiamare close() sul file descriptor del socket per liberare le risorse