Writing TCP-based Clients

Chris Brown

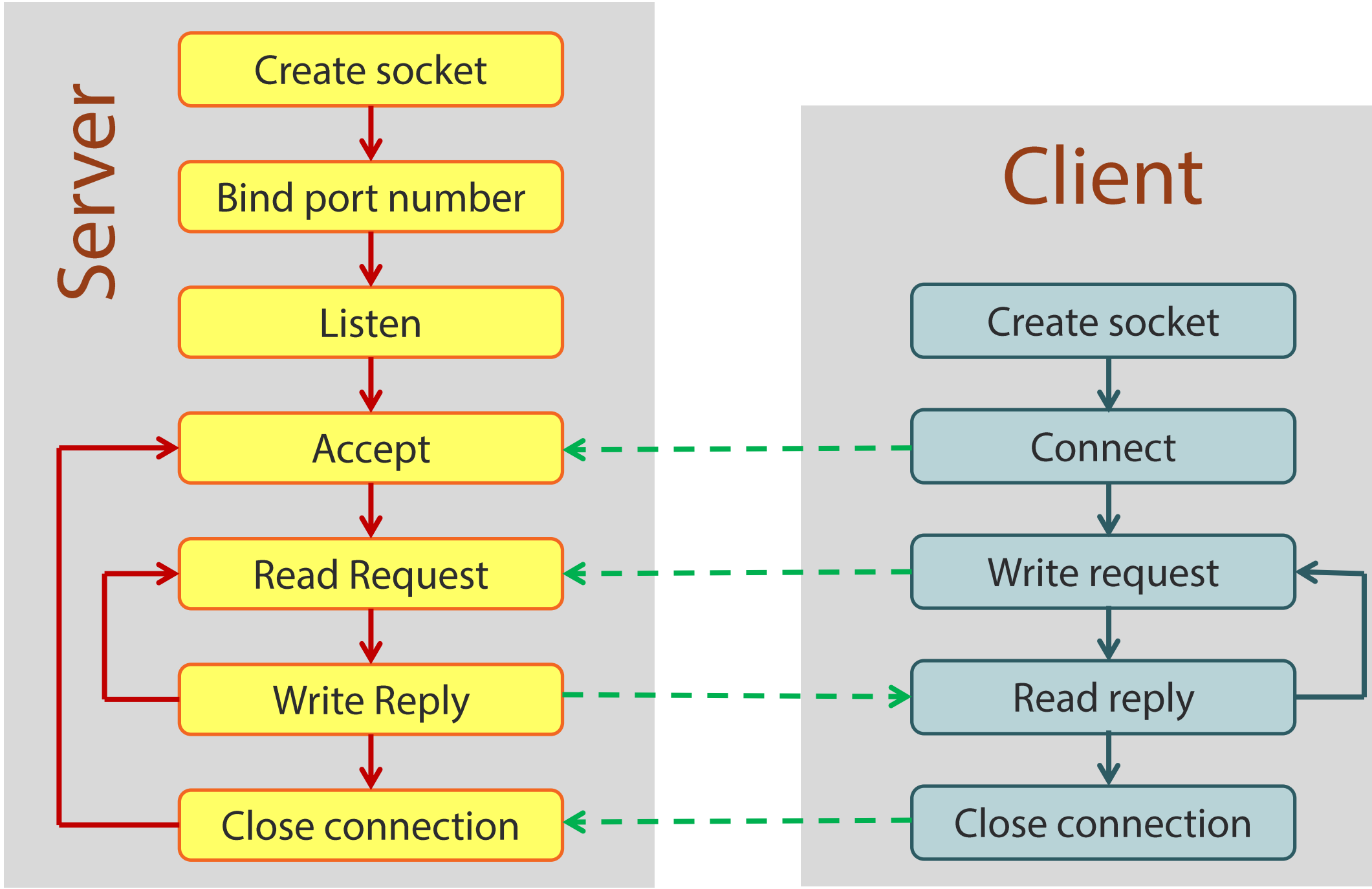
In This Module …

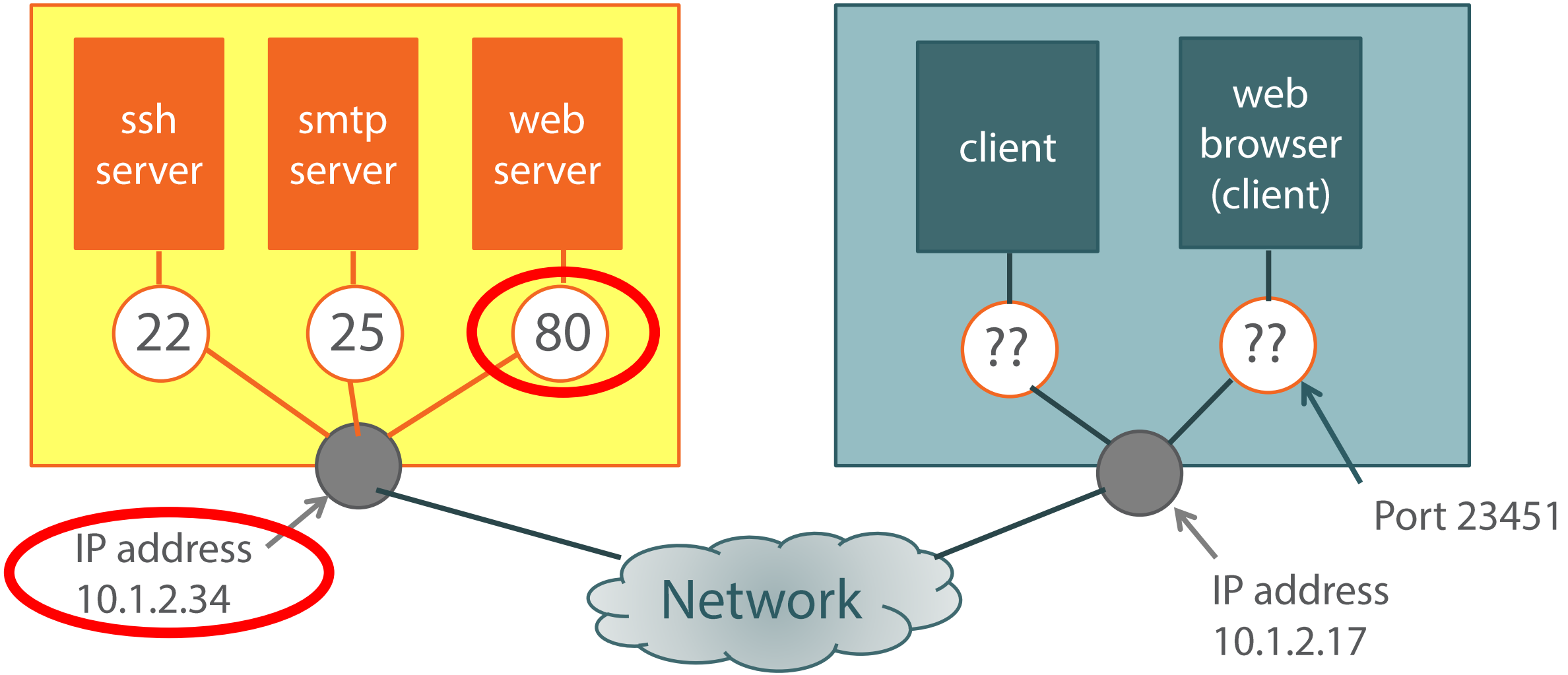
A TCP client for our rot13 server

Finding the service (the old-fashioned way)

Finding the service (the new way)

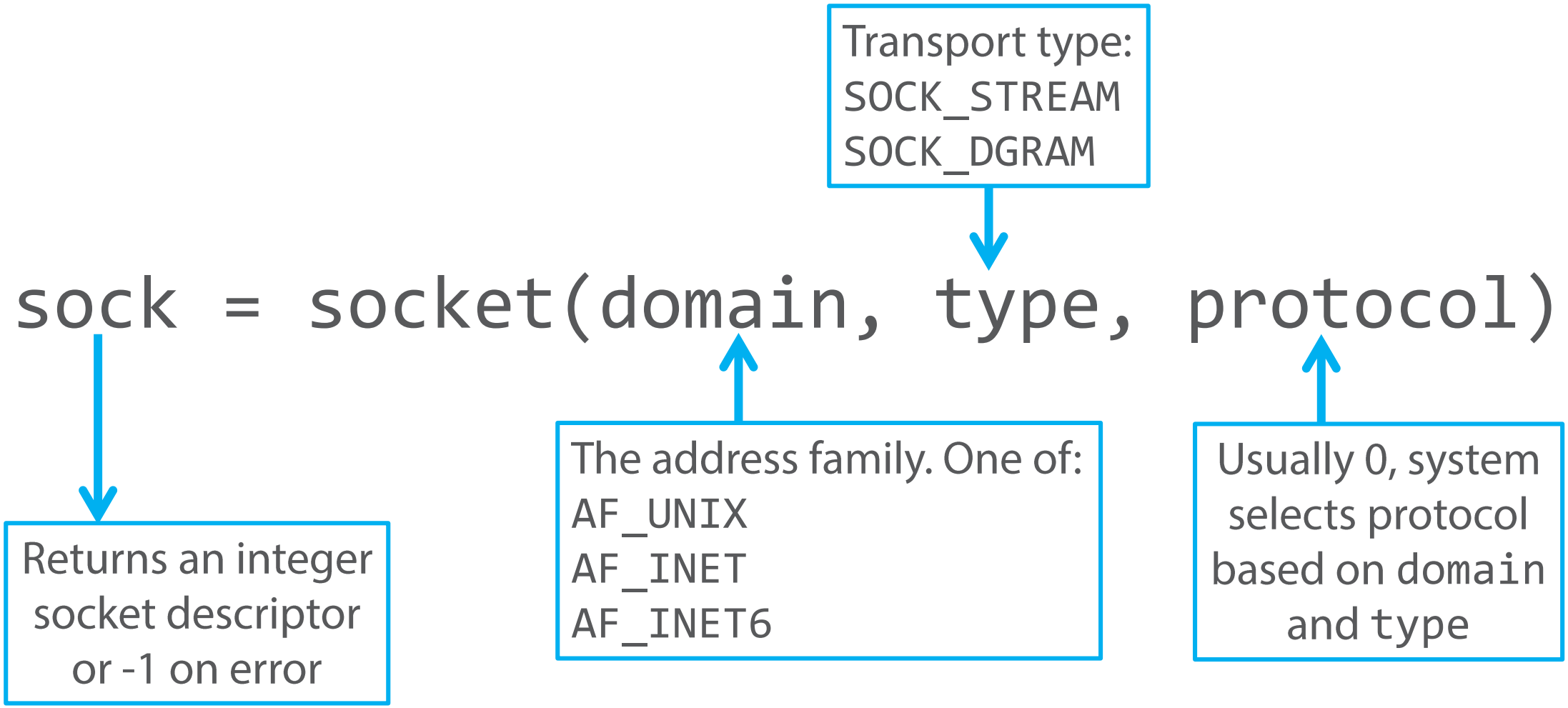
Doing it in Python


Server
Create socket
Bind port number
Listen
Accept
Read Request
Write Reply
Close connection
Client
Create socket
Connect
Write request
Read reply
Close connection

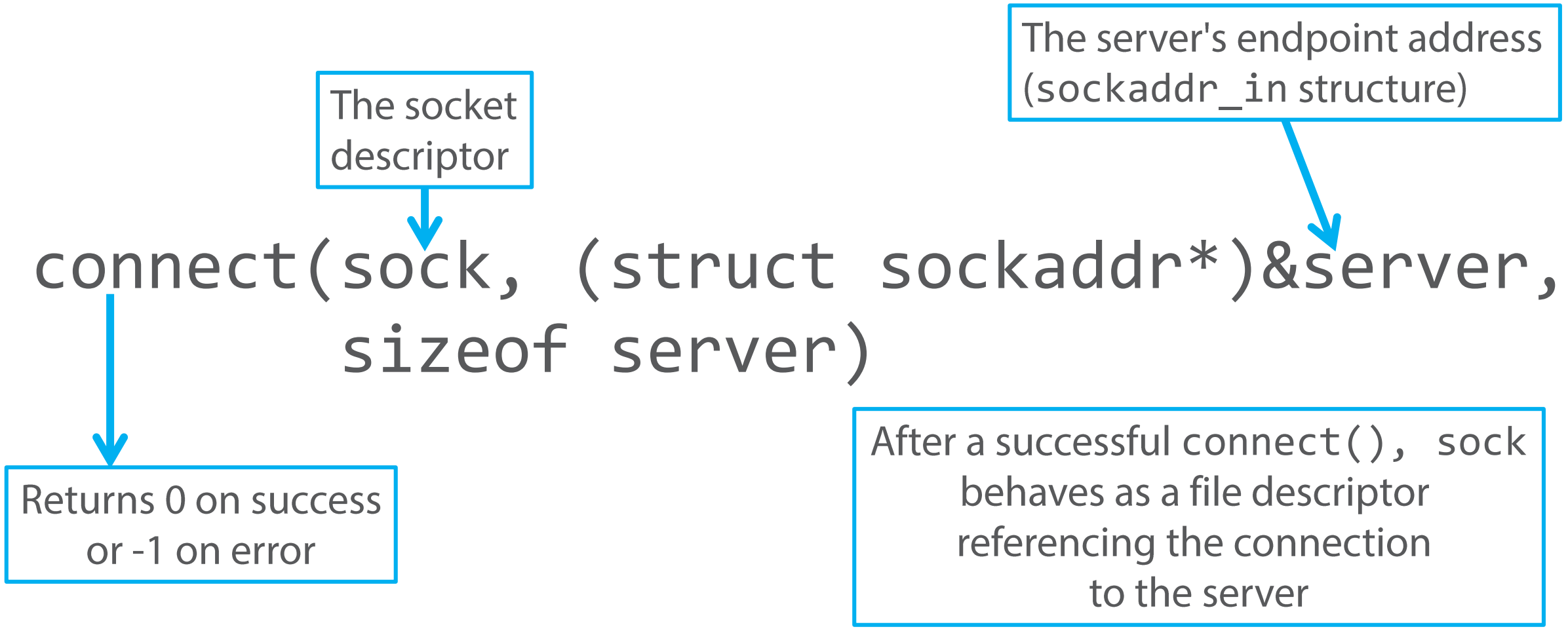


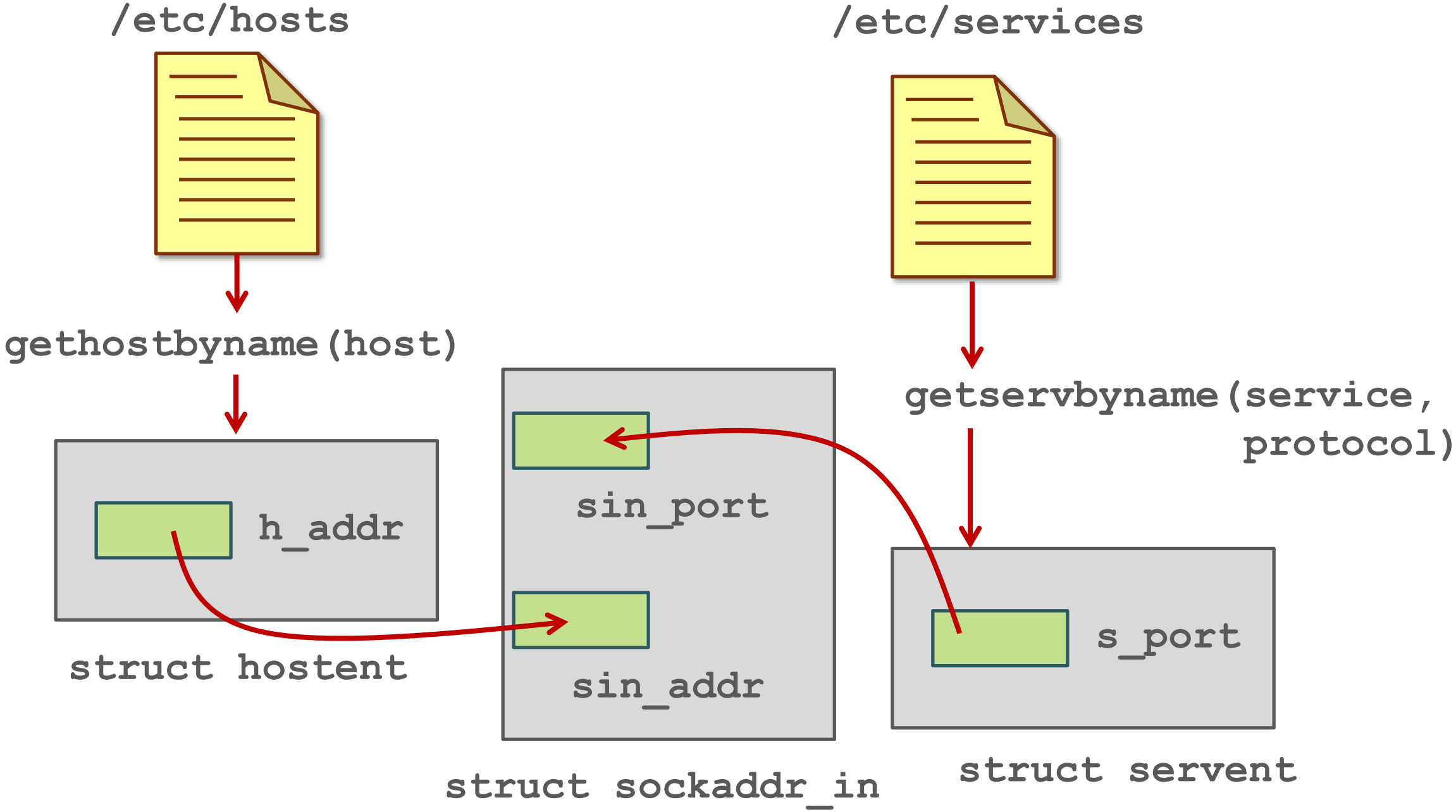
Association: {client IP, client port, server IP, server port} =
{10.1.2.17, 23451, 10.1.2.34, 80}

Creating a Socket



Making the Connection




/etc/hosts
/etc/services
gethostbyname(host)
getservbyname(service,
protocol)
h_addr
sin_port
s_port
sin_addr
struct hostent
struct sockaddr_in
struct servent

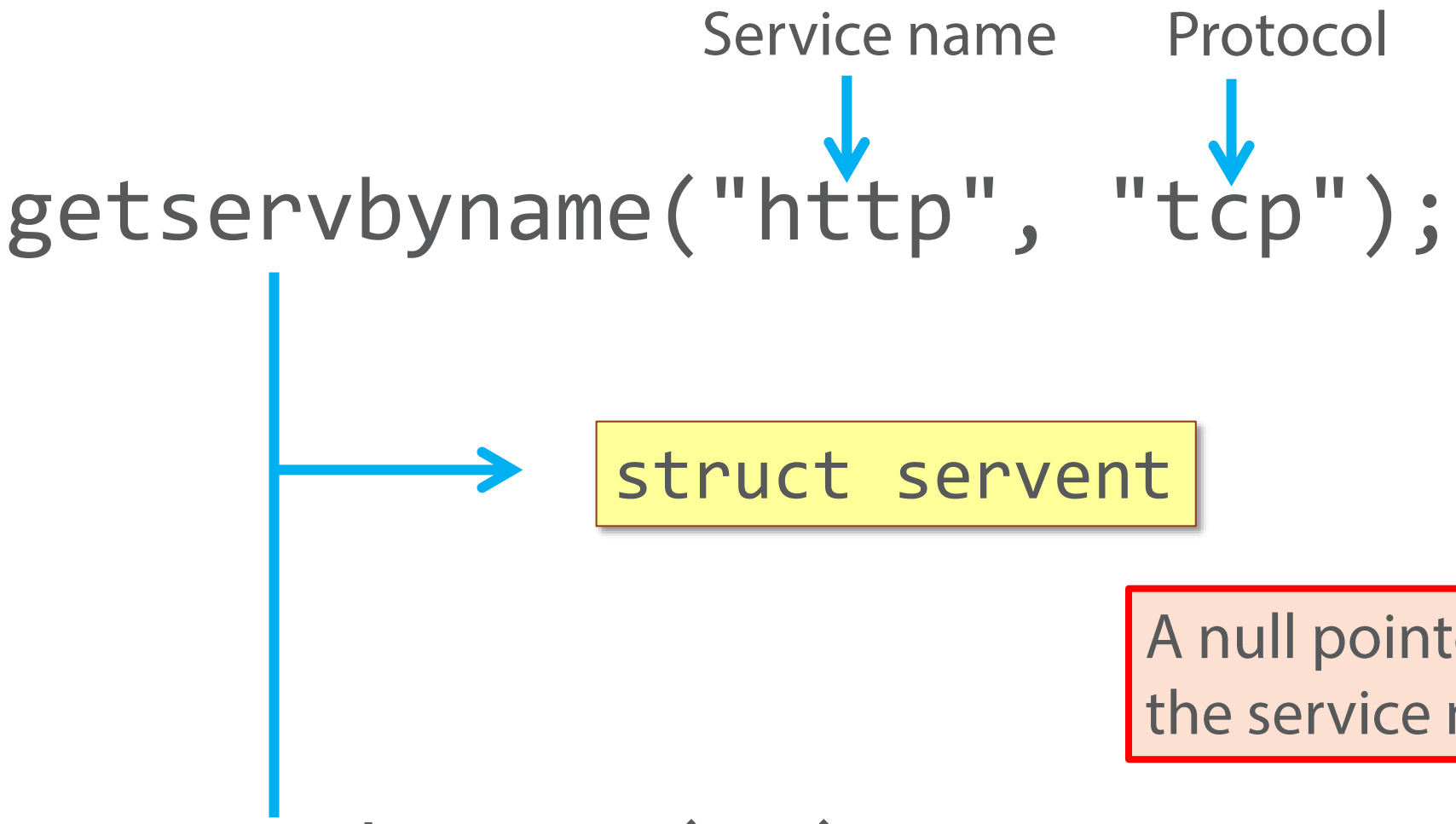
The /etc/services File

```
ftp-data      20/tcp
ftp           21/tcp
ssh           22/tcp
ssh           22/udp
telnet        23/tcp
smtp          25/tcp
domain        53/tcp
domain        53/udp
tftp          69/udp
http          80/tcp
```

Maps service names to port number and protocol

For official port assignments see:
`http://www.iana.org/assignments/port-numbers`

Finding the Service

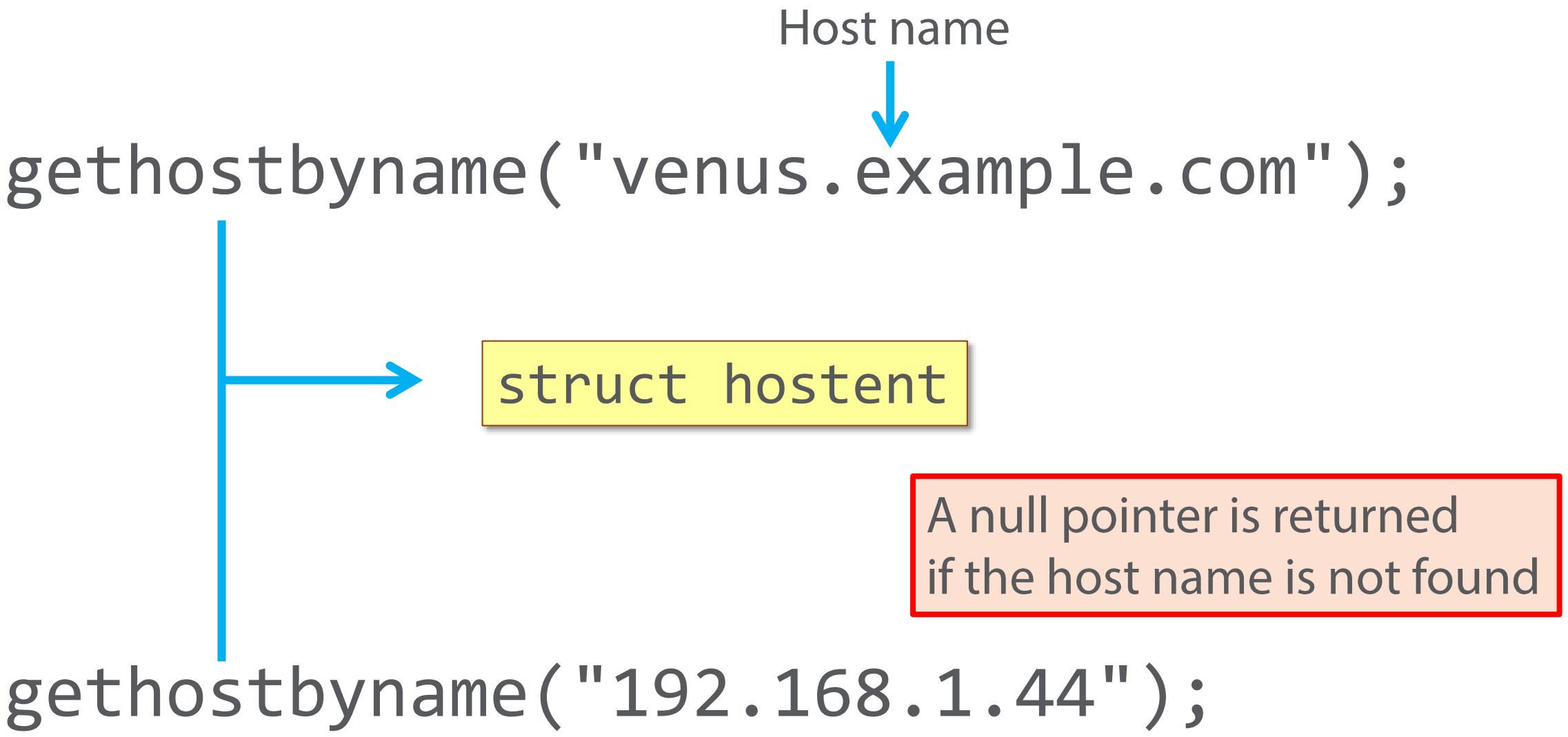


A null pointer is returned if the service name is not found

The servent Structure

```
struct servent {
    char  *s_name;        /* official service name */
    char **s_aliases;     /* alias list */
    int    s_port;        /* port number */
    char  *s_proto;       /* protocol */
}
```

Finding the Host



The hostent Structure

```
struct hostent {
    char  *h_name;        /* official name of host */
    char **h_aliases;     /* alias list */
    int    h_addrtype;    /* host address type */
    int    h_length;      /* length of address */
    char **h_addr_list;   /* list of addresses */
}
/* For backward compatibility: */
#define h_addr h_addr_list[0]
```

"Canonical" name

AF_INET or AF_INET6

Null-terminated array of pointers to IP addresses

Easy way to access the first address in the list

Putting it all Together

```
struct hostent      *host_info;
struct servent      *serv_info;
struct sockaddr_in  server;

host_info = gethostbyname("venus.example.com");
serv_info = getservbyname("http", "tcp");

server.sin_family = AF_INET;
memcpy(&server.sin_addr, host_info->haddr, host_info->h_length);
server.sin_port = serv_info->s_port;
connect(sock, (struct sockaddr*)&server, sizeof server)
```

Demonstration

Old-fashioned TCP client

Protocol Independence

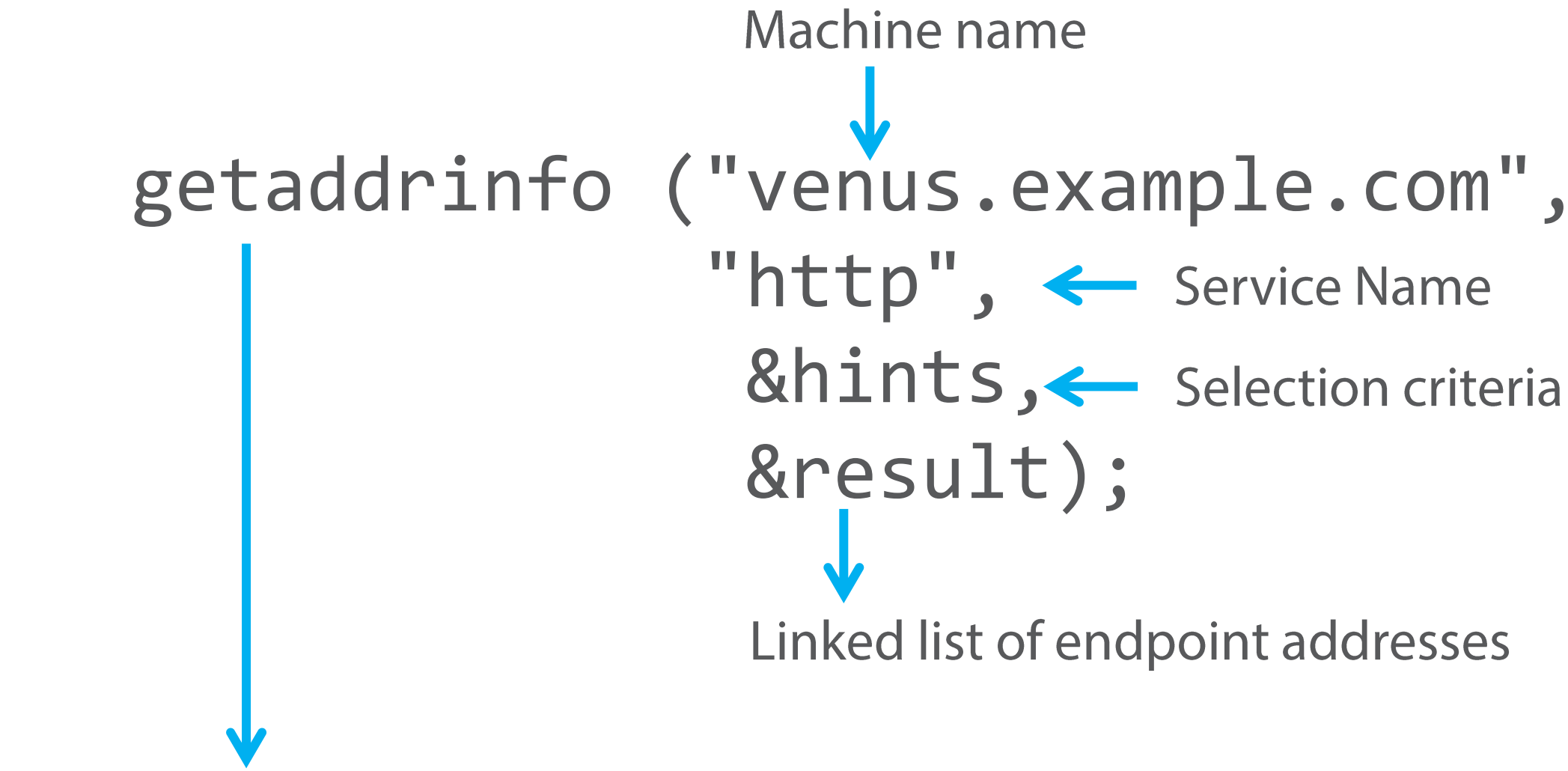
Traditional API (`gethostbyname()`, etc) makes it hard to be "protocol independent"

Growing need for IPv4 / IPv6 operability

`getaddrinfo()` more complex API ... but easier to achieve protocol independence

Also ... traditional API is not suitable for multi-threaded applications

Finding the Service (the Modern Way)



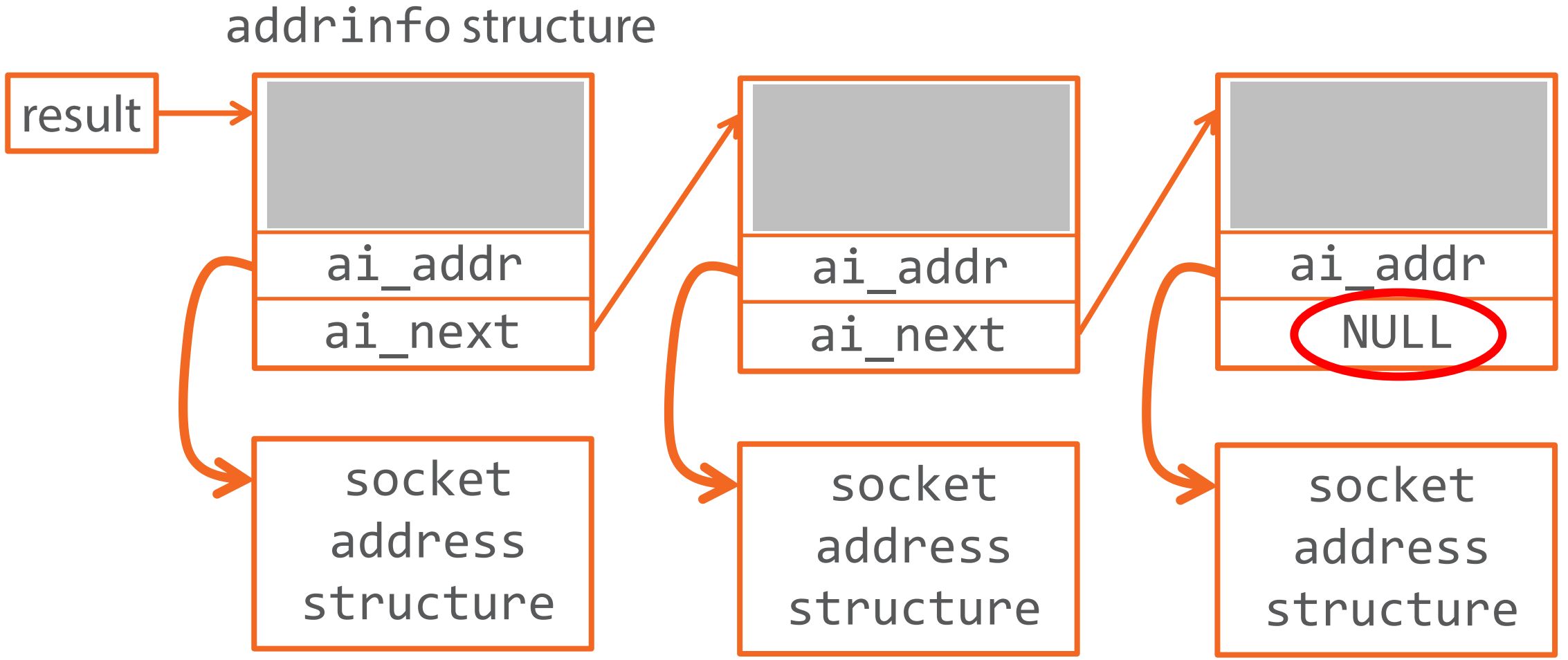
Providing Hints

The `hints` argument of `getaddrinfo()` lets you constrain the types of endpoint addresses you want.

For example, to return only IPV6 UDP endpoints:

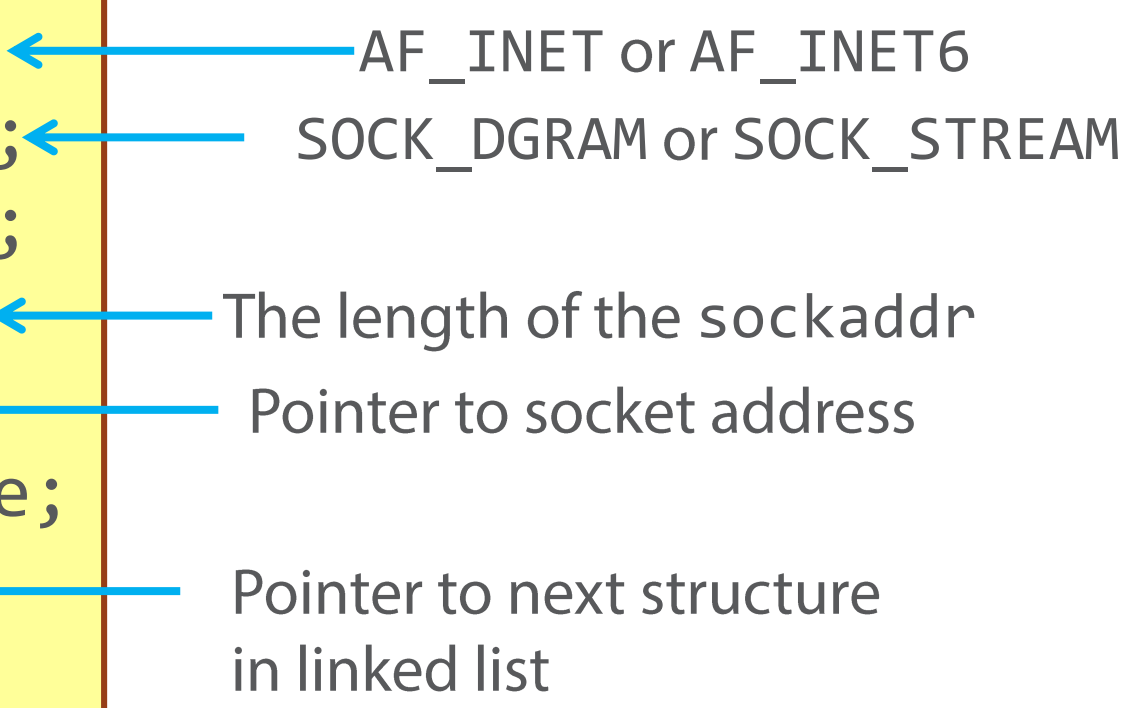
```
struct addrinfo   hints;

memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_INET6;
hints.ai_socktype = SOCK_DGRAM
```

Data Structures



The addrinfo Structure

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    socklen_t        ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
```



Demonstration

Modern TCP client

Doing it in Python

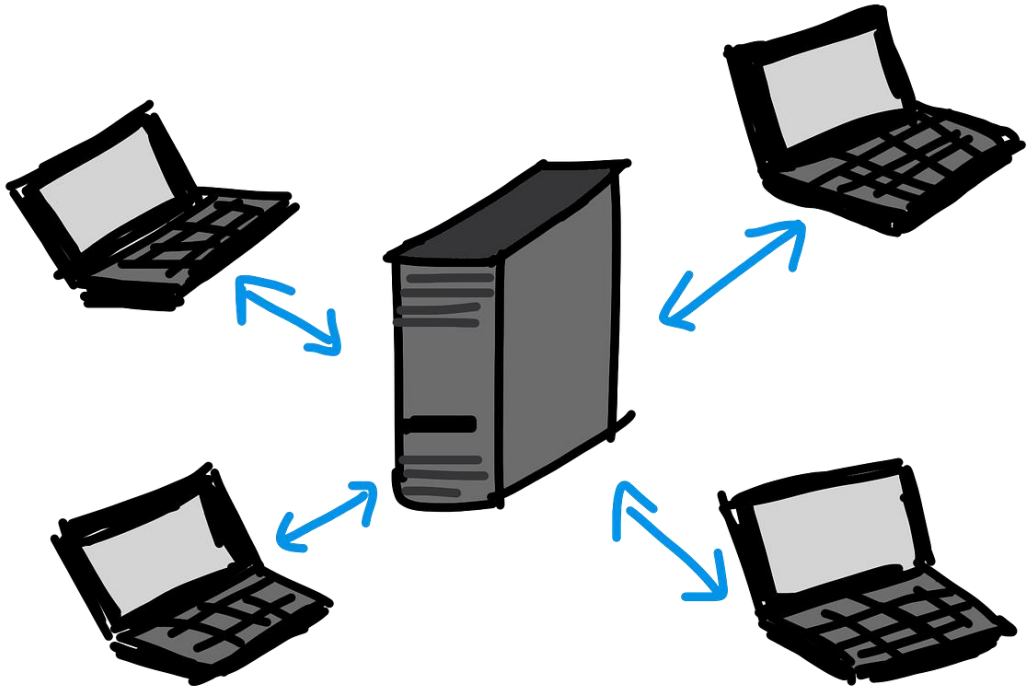

```
#!/usr/bin/python3
# The core of a Python TCP client

from socket import *

s = socket(AF_INET, SOCK_STREAM)
port = getservbyname("rot13")
s.connect(("localhost", port))

s.send(...)
s.recv(...)
```

Must be an integer

Demonstration

Python TCP client

Module Summary



Finding the service

Finding the host

Connecting to the server

Protocol Independence

Doing it in Python

Moving Forward …

Coming up in the next module:

- UDP clients and servers
- TFTP client
- UDP broadcasting