Networking via TCP

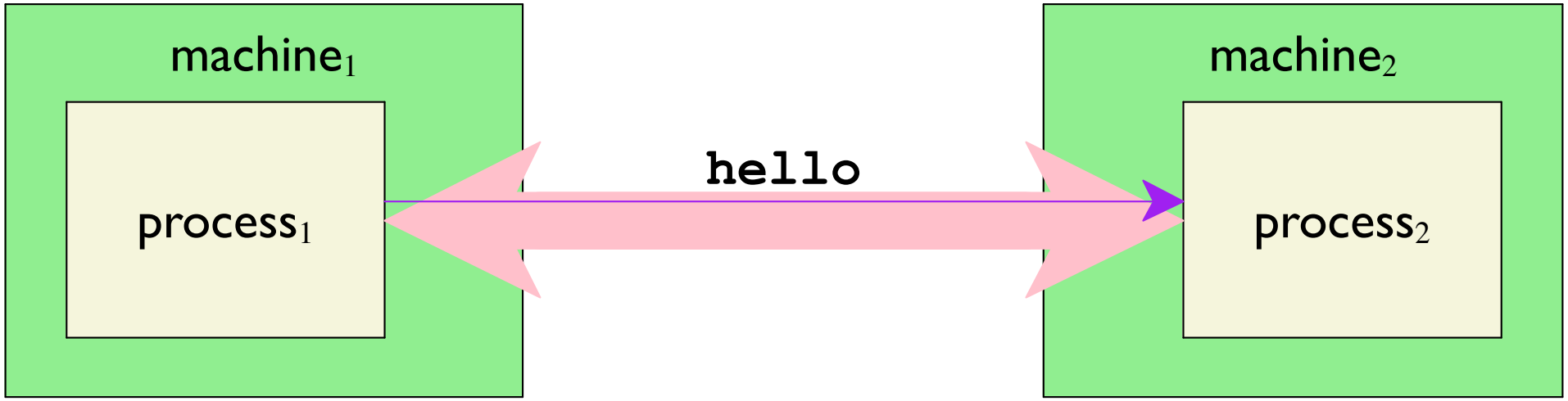
Networking via TCP

Networking via TCP

 = `tcp_open(....);`

 = `tcp_open(....);`

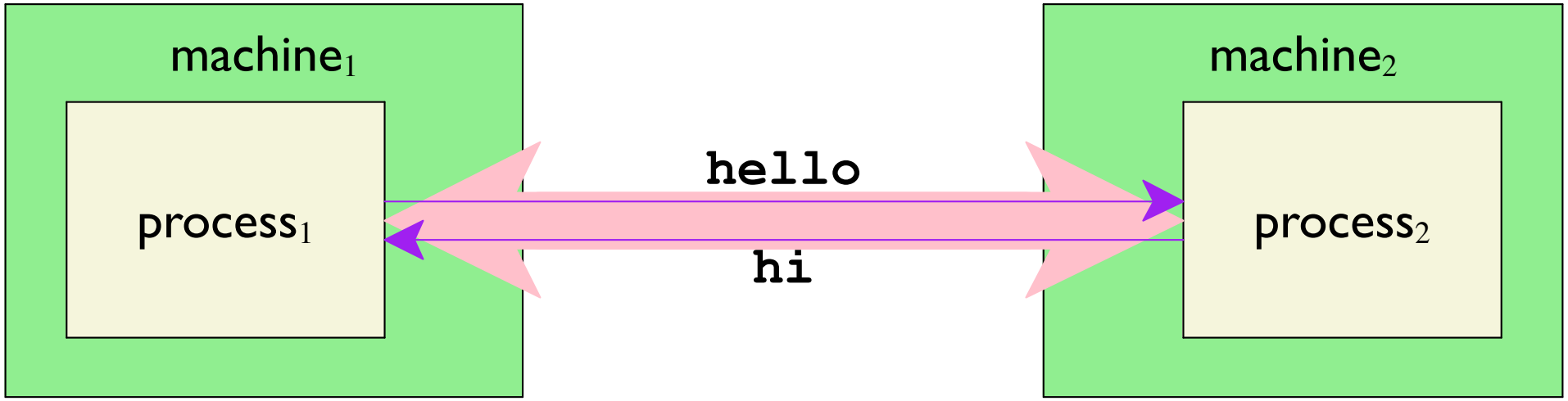
Networking via TCP



↔ = `tcp_open(....);` ↔ = `tcp_open(....);`

`Write(↔, "hello", 5);` `Read(↔, buffer, 5);`

Networking via TCP

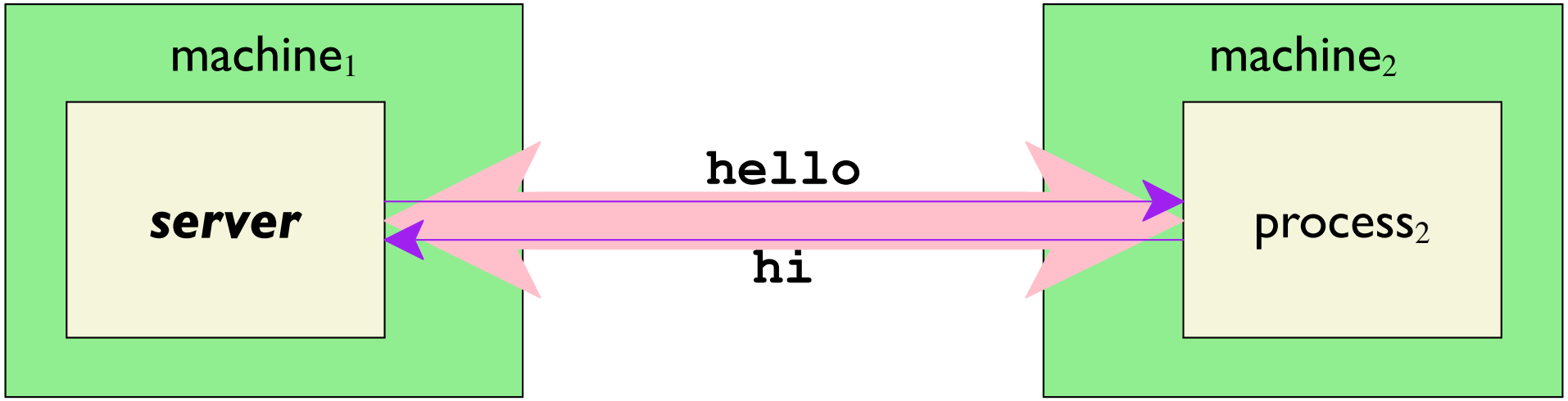


↔ = `tcp_open(....);` ↔ = `tcp_open(....);`

```
Write(↔, "hello", 5);
Read(↔, buffer, 2);
```

```
Read(↔, buffer, 5);
Write(↔, "hi", 2);
```

Networking via TCP

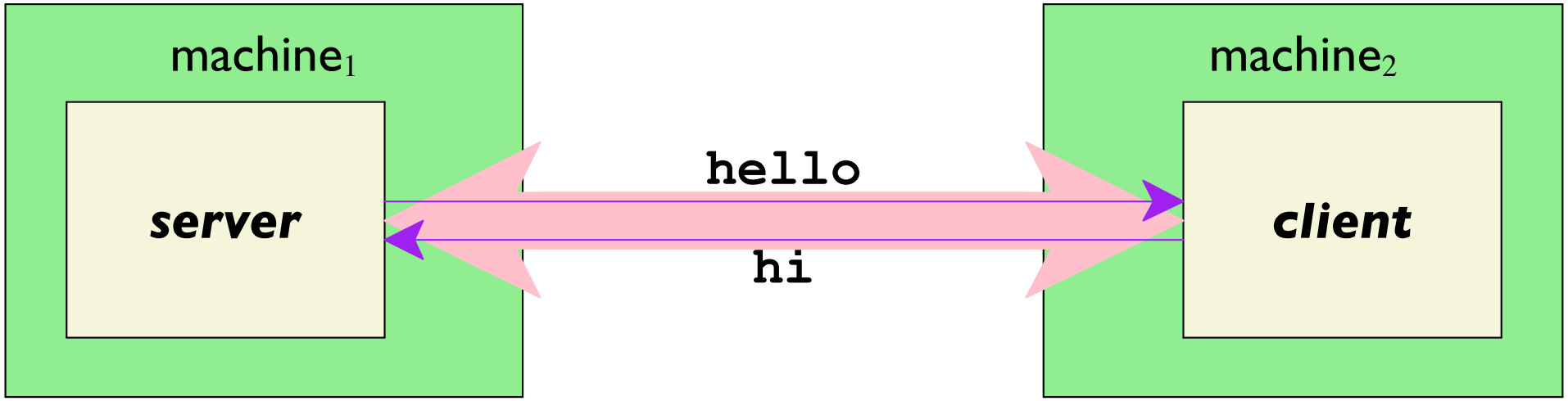



```
lnr = Open_listenfd(....);
    = Accept(lnr, ....);
```

```
Write(    , "hello", 5);
Read(     , buffer, 2);
```

```
Read(     , buffer, 5);
Write(    , "hi", 2);
```

Networking via TCP




```
lnr = Open_listenfd(....);
      = Accept(lnr, ....);
```

```
                                  = Open_clientfd(....
```

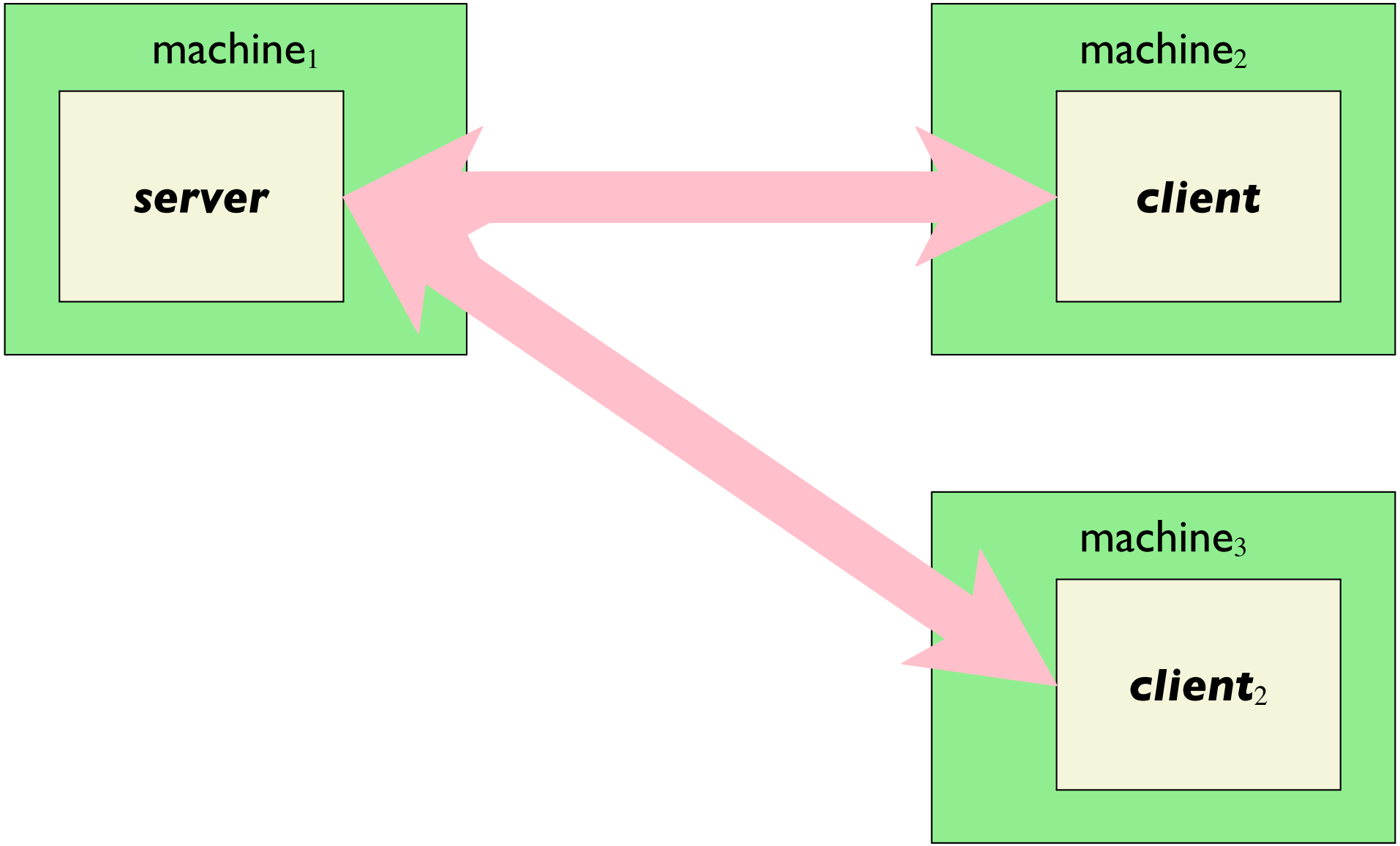
```
Write(      , "hello", 5);
```

```
Read(      , buffer, 2);
```

```
Read(      , buffer, 5);
```

```
Write(      , "hi", 2);
```

Networking via TCP



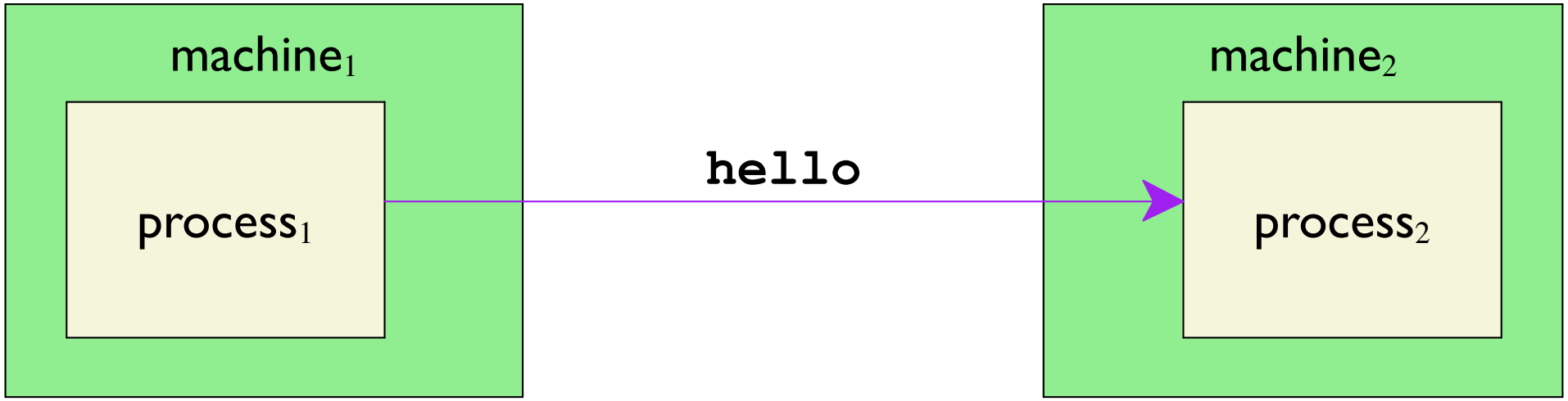
Networking via UDP

Networking via UDP


```
➤ = Socket(....);
Bind(➤, ....);
```

```
◀ = Socket(....);
Bind(◀, ....);
```

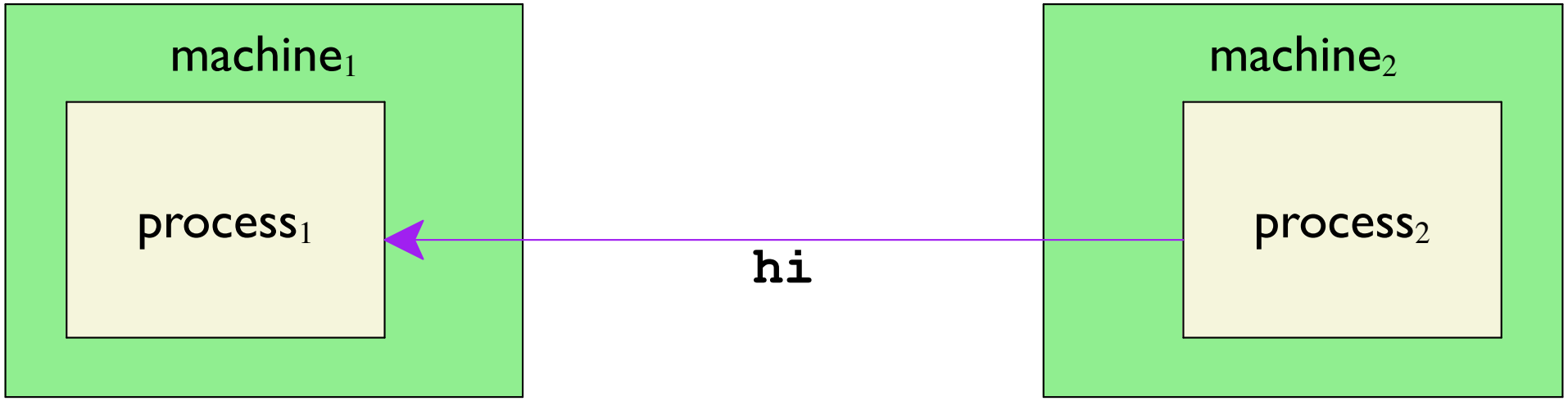
Networking via UDP




```
➤ = Socket(....);
Bind(➤, ....);

Sendto(➤, "hello", 5, ...);
```

```
➤ = Socket(....);
Bind(➤, ....);

Recv(➤, buffer, 5, 0);
```

Networking via UDP




```
➤ = Socket(....);
Bind(➤, ....);

Recv(➤, buffer, 2);
```

```
➤ = Socket(....);
Bind(➤, ....);

Sendto(➤, "hi", 2, ...);
```

TCP vs. UDP

TCP

Connection- and stream-oriented

Reliable

The most widely used networking protocol

UDP

Connectionless and packet-oriented

Best-effort

Minimal structure over next primitive layer

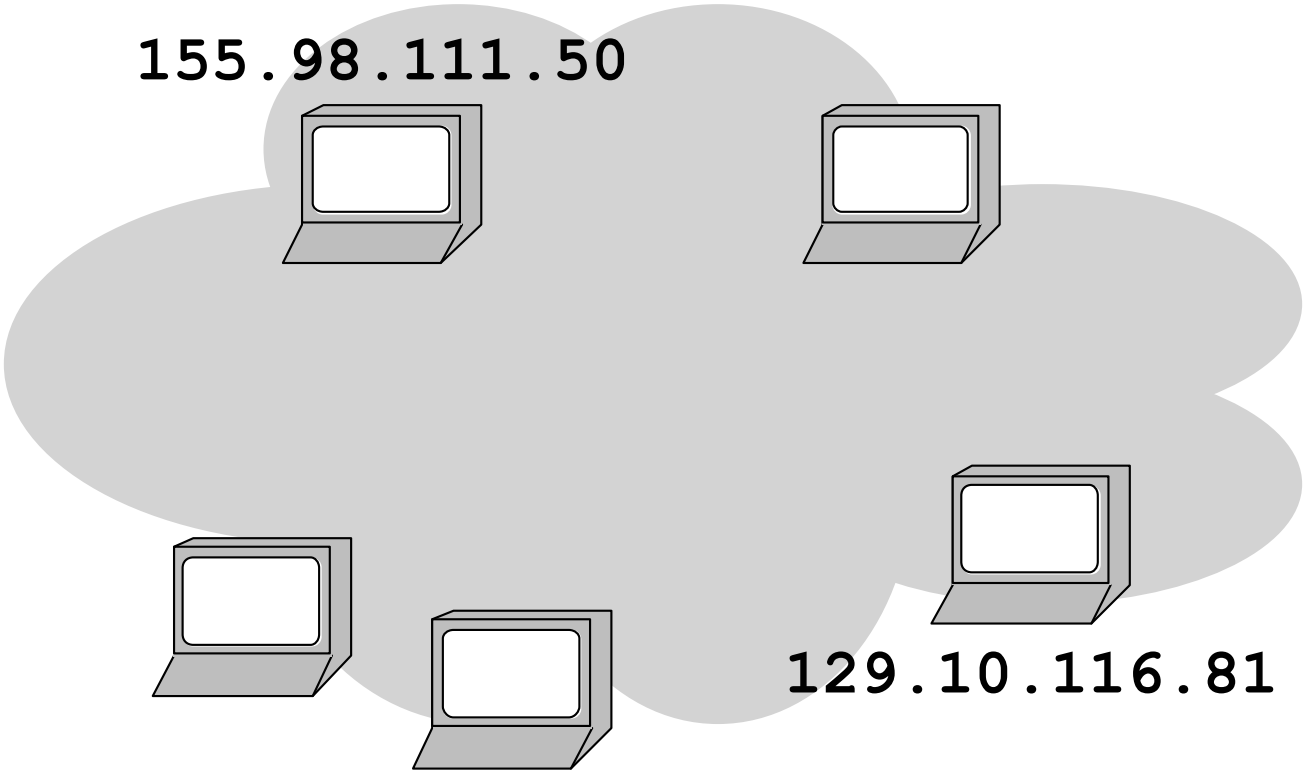
Both built on ***IP***

Finding Hosts on a Network

Using ***IP***, a ***host*** is named by a 32-bit value

More precisely, this is IPv4

Written as dot-separated, unsigned 8-bit values

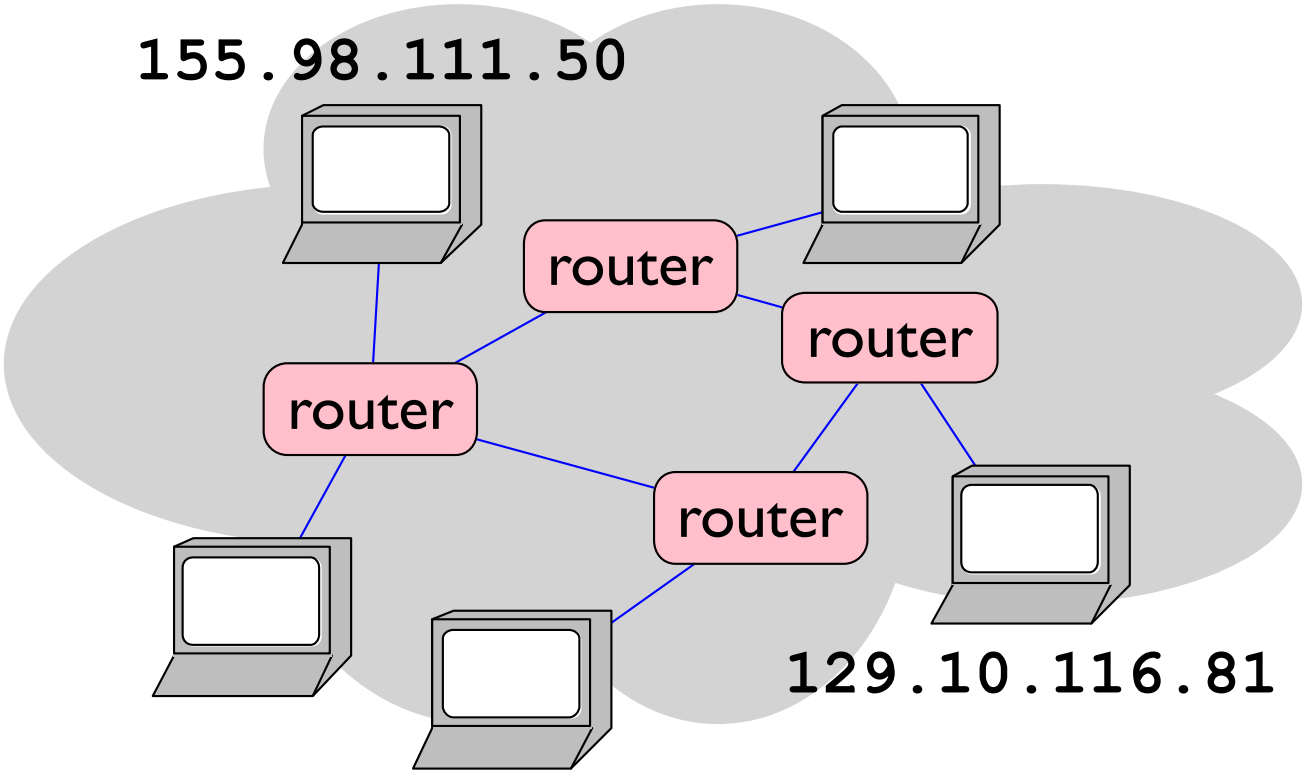


Finding Hosts on a Network

Using ***IP***, a ***host*** is named by a 32-bit value

More precisely, this is IPv4

Written as dot-separated, unsigned 8-bit values

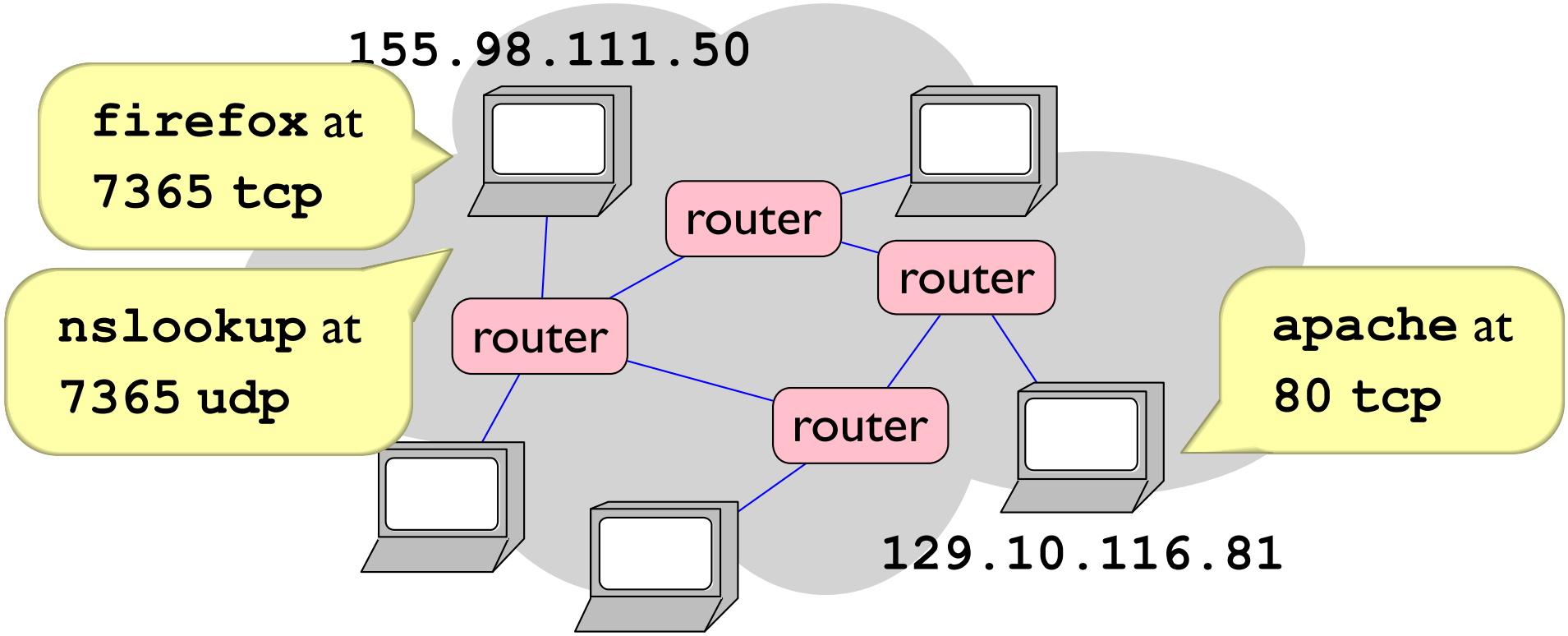


Finding Hosts on a Network

Using ***IP***, a ***host*** is named by a 32-bit value

More precisely, this is IPv4

A ***port*** plus ***protocol*** identifies an endpoint within a host

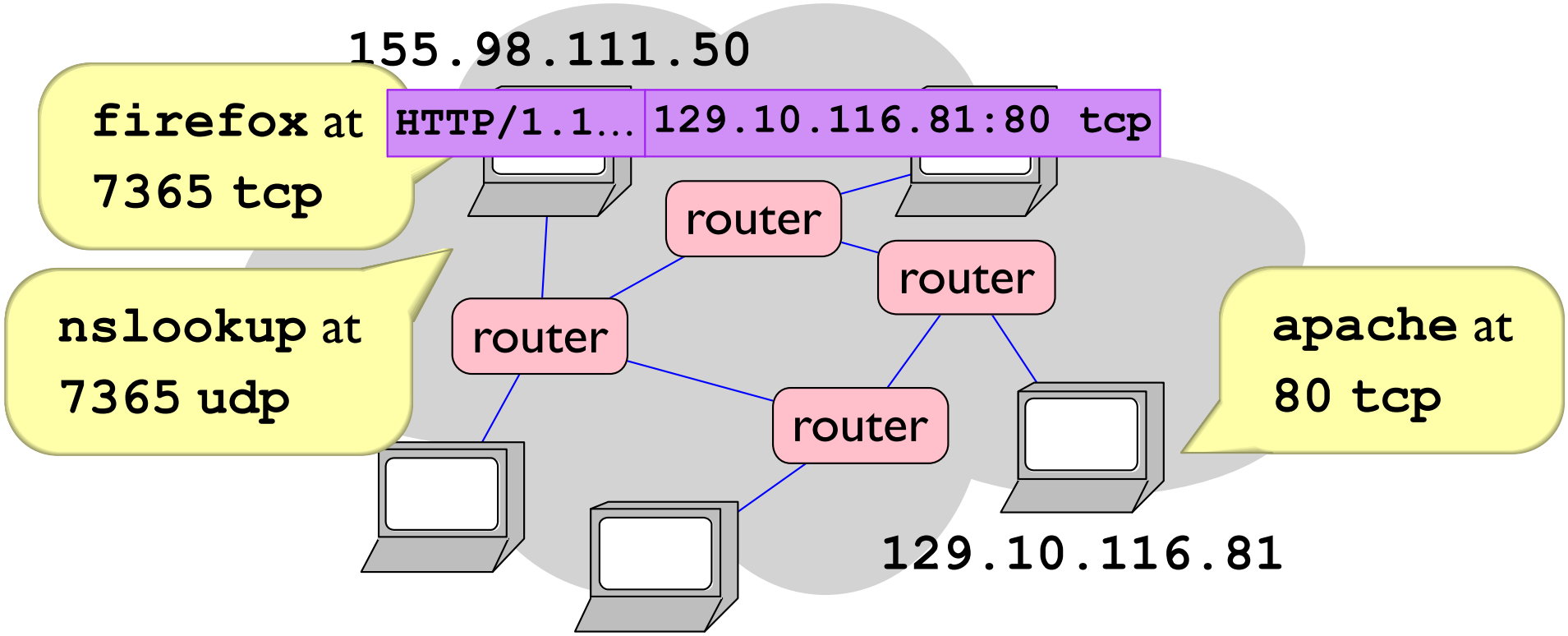


Finding Hosts on a Network

Using ***IP***, a ***host*** is named by a 32-bit value

More precisely, this is IPv4

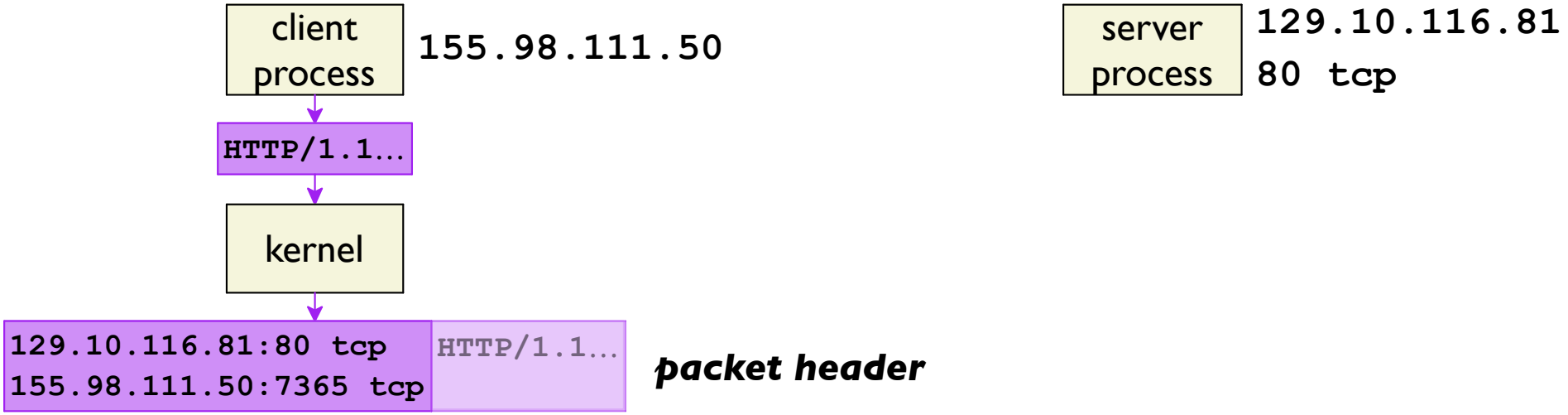
A ***port*** plus ***protocol*** identifies an endpoint within a host



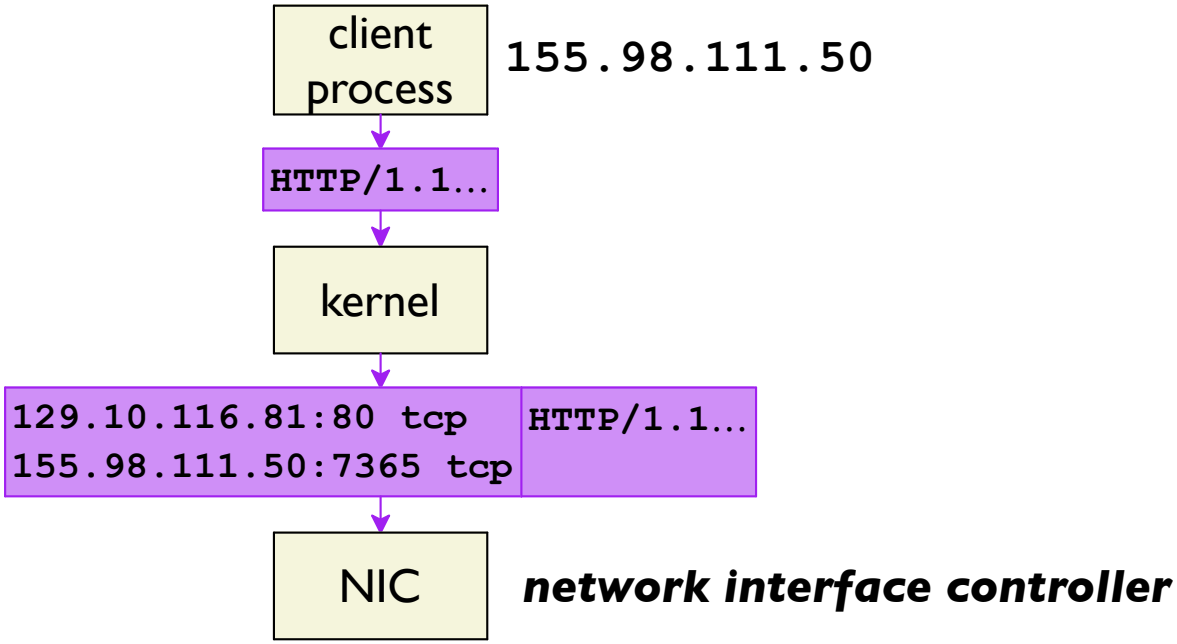
Message Transport

Message Transport

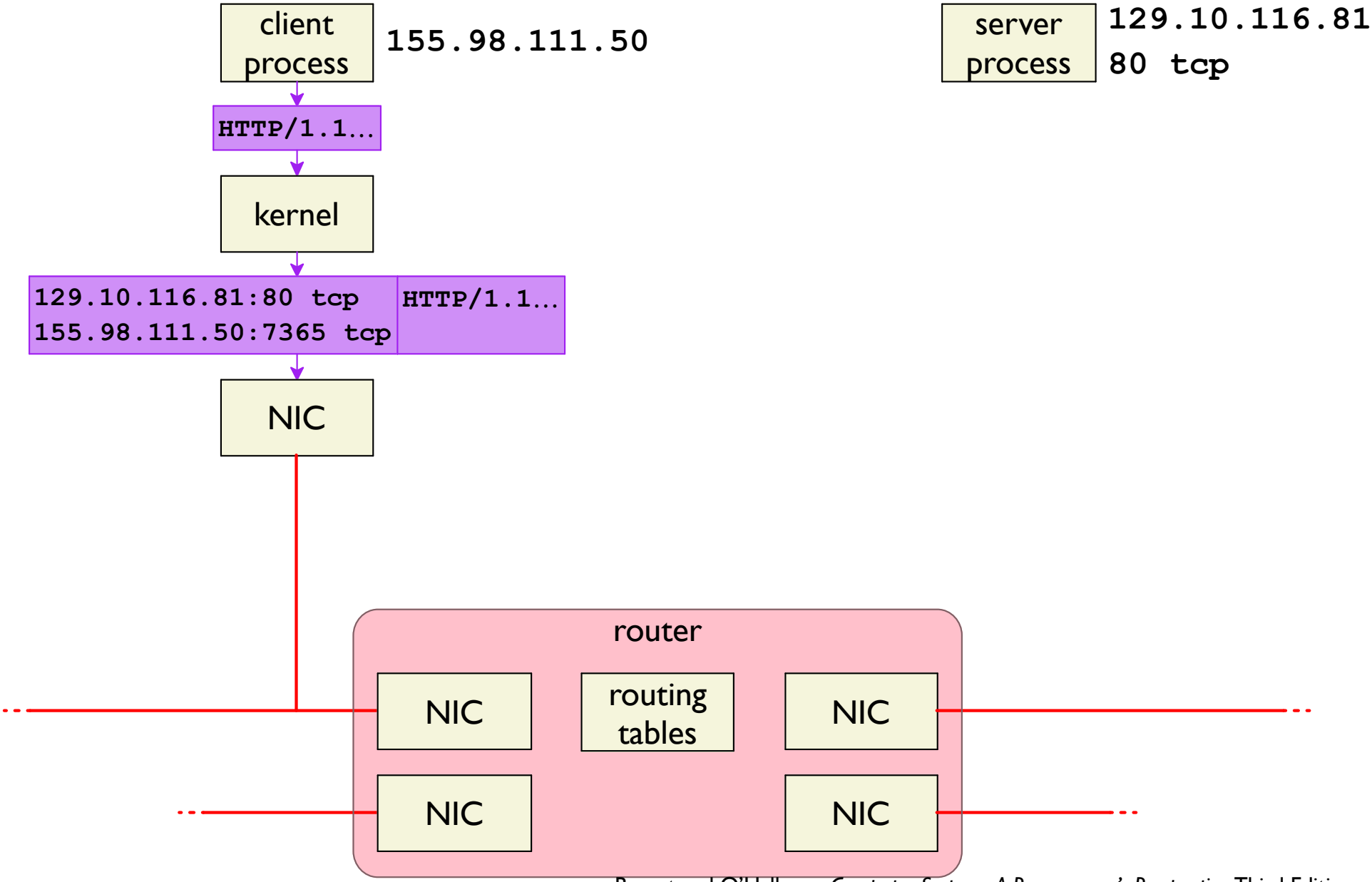
Message Transport



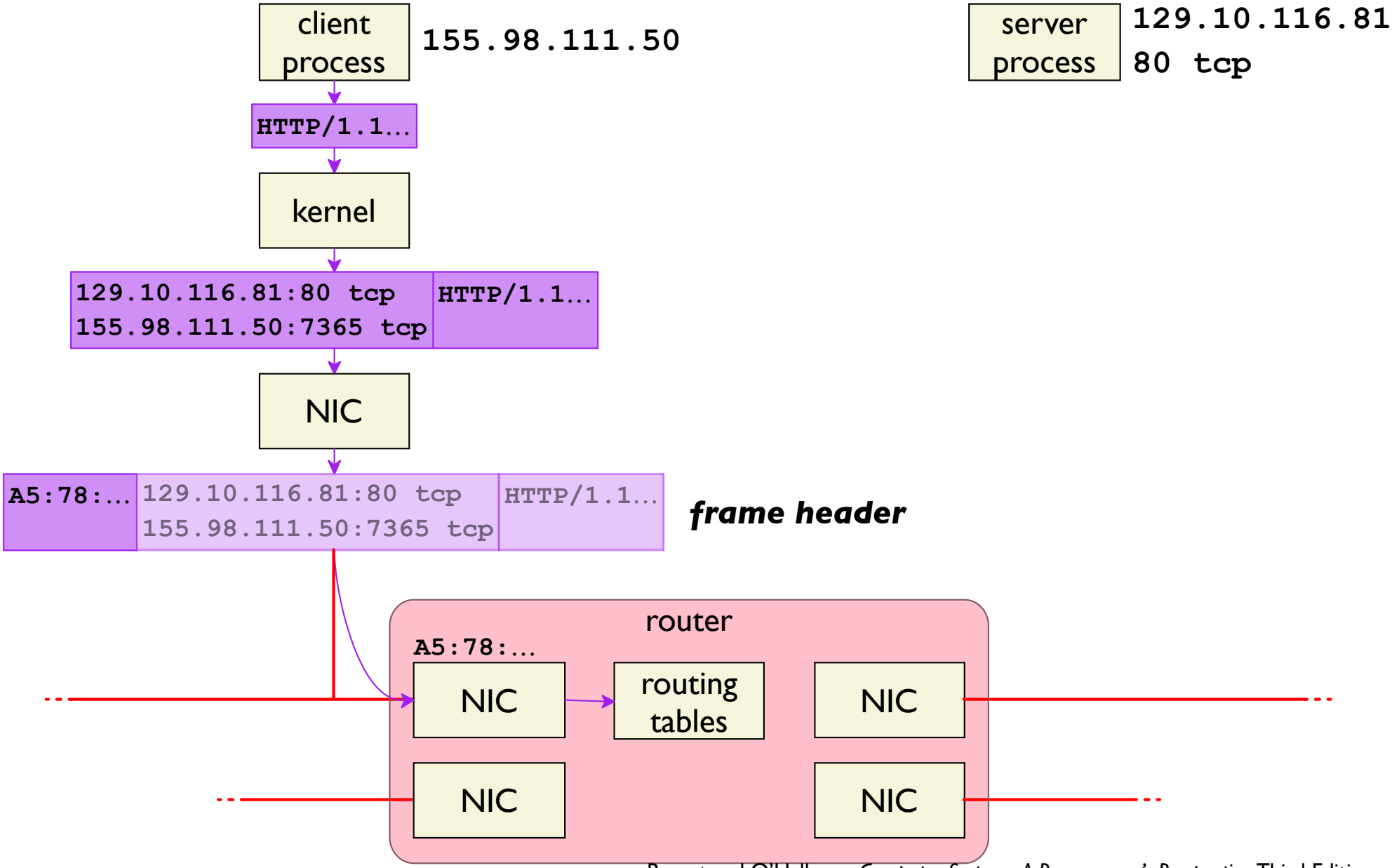
Message Transport



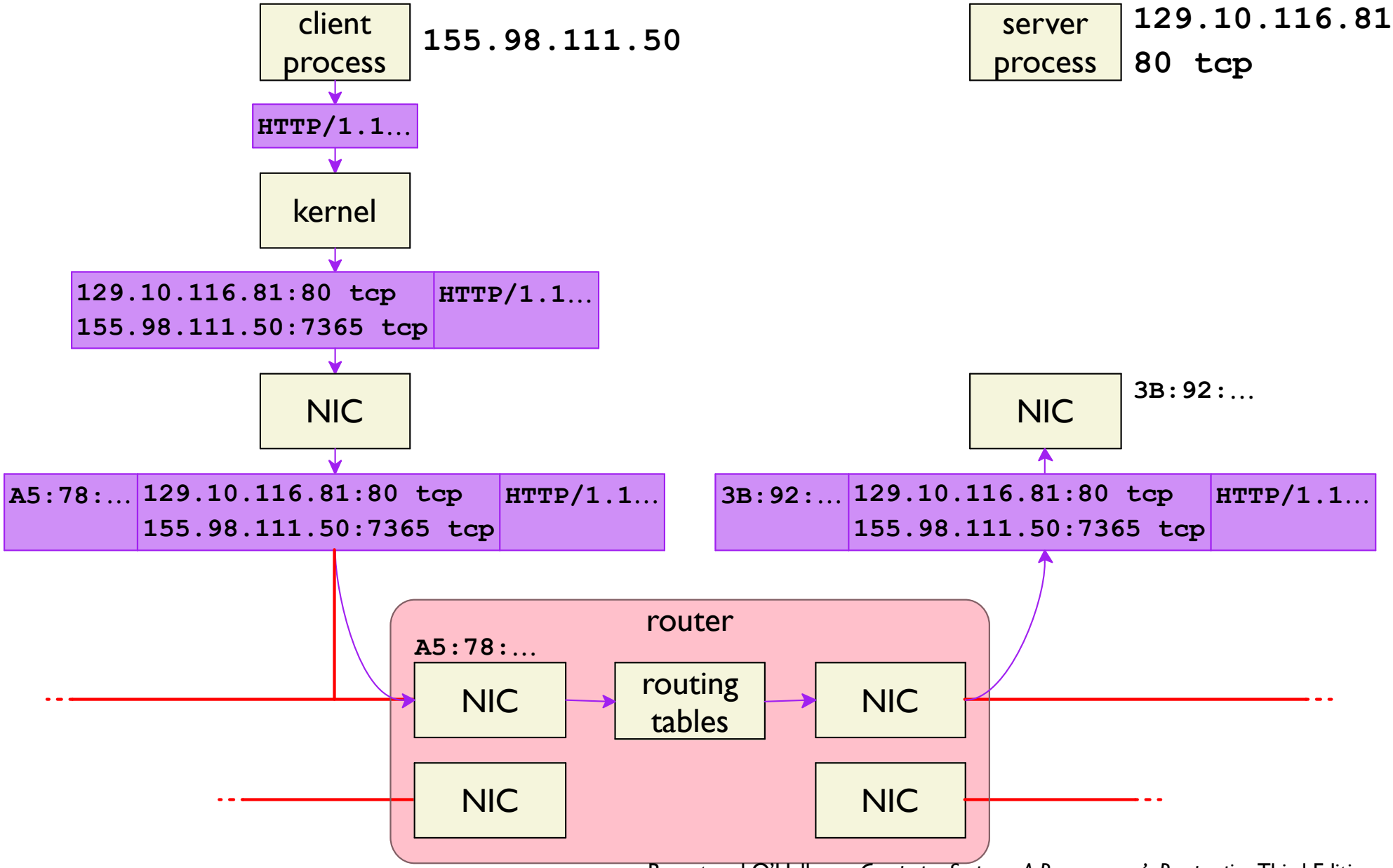
Message Transport



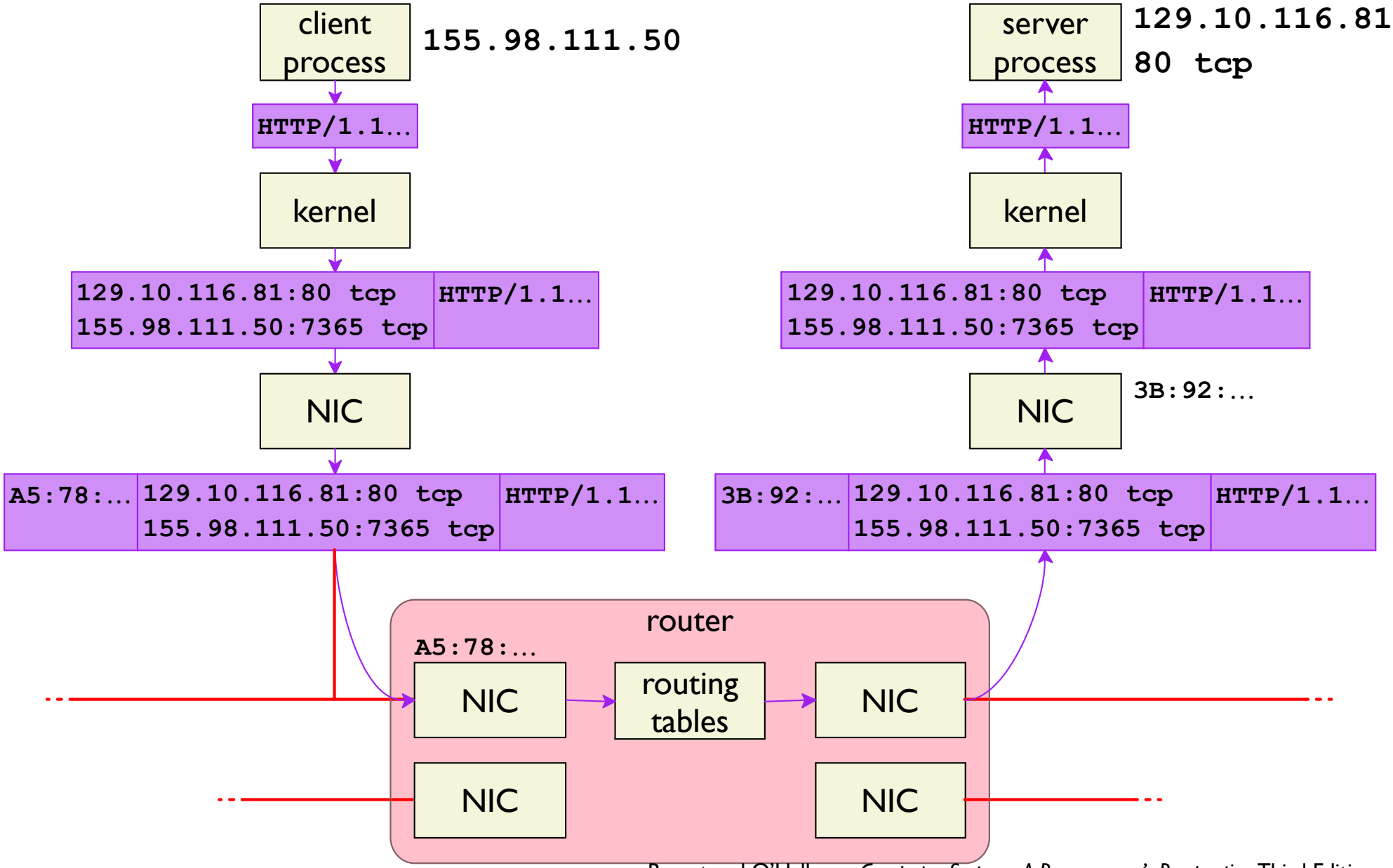
Message Transport



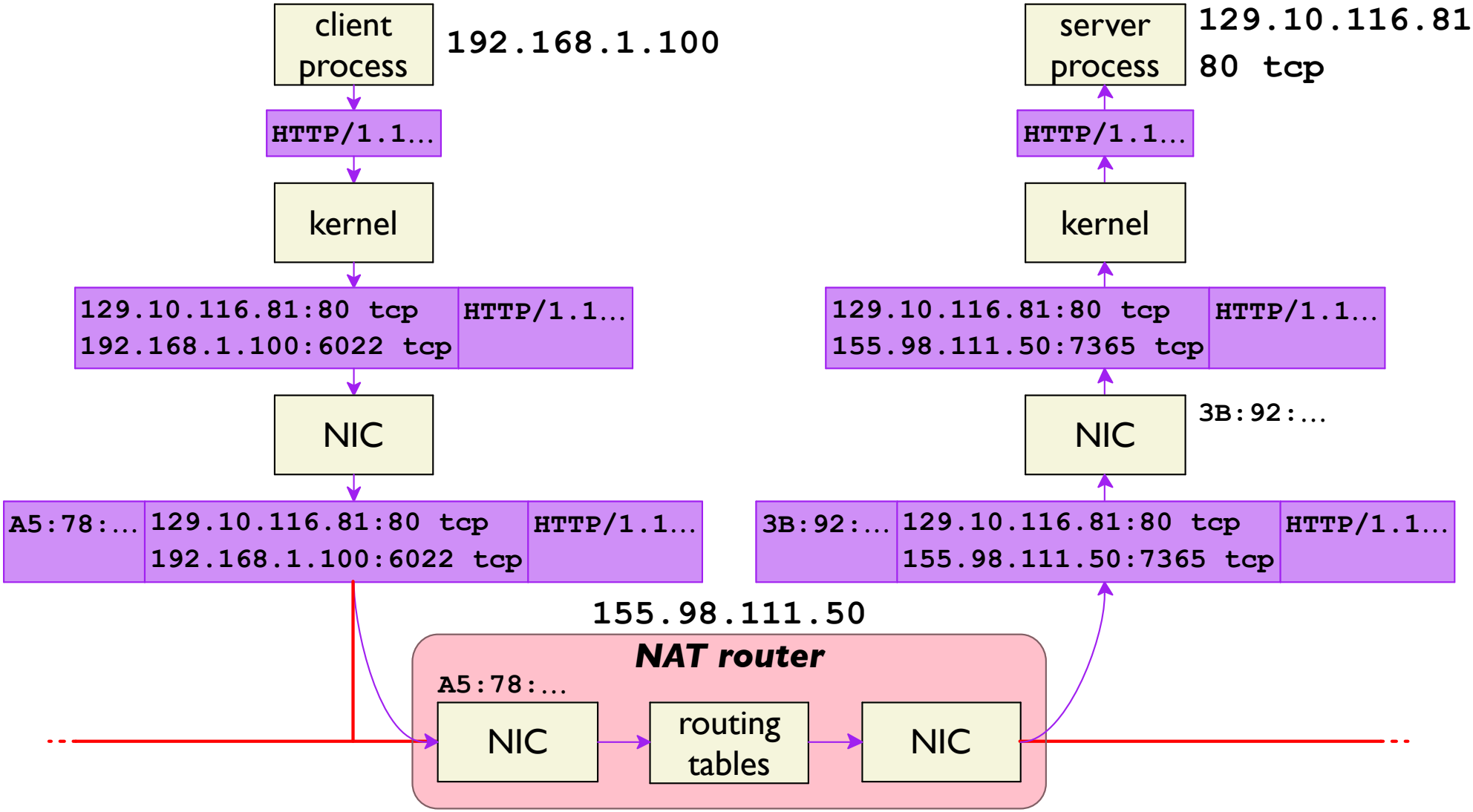
Message Transport



Message Transport



Message Transport



Name Resolution

IP locates hosts by number

e.g., `155.98.111.50`

```
$ ssh 155.98.111.50
```

For many purposes, names are obviously better

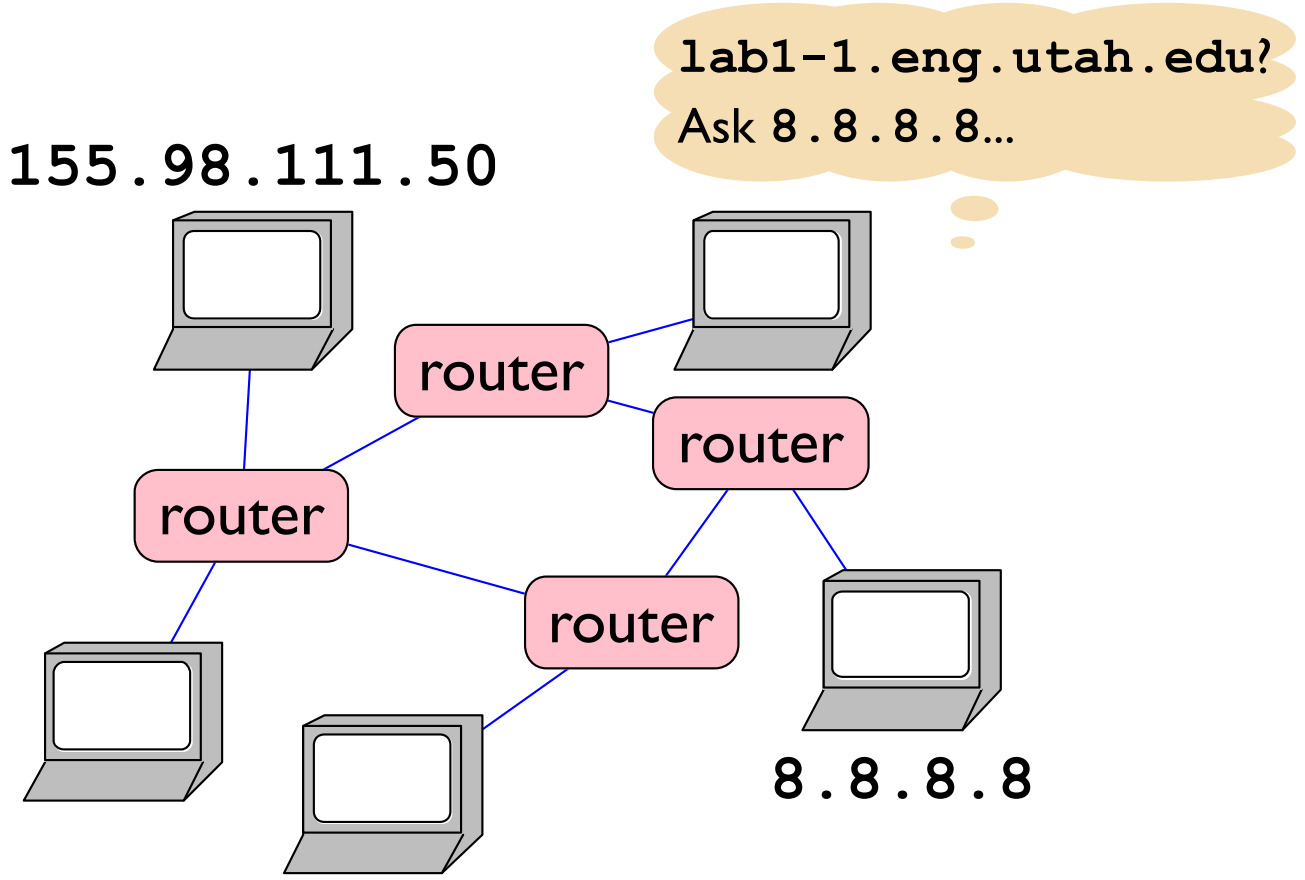
e.g., `lab1-1.eng.utah.edu`

```
$ ssh lab1-1.eng.utah.edu
```

Name Resolution

DNS (Domain Name System) maps names to remote addresses

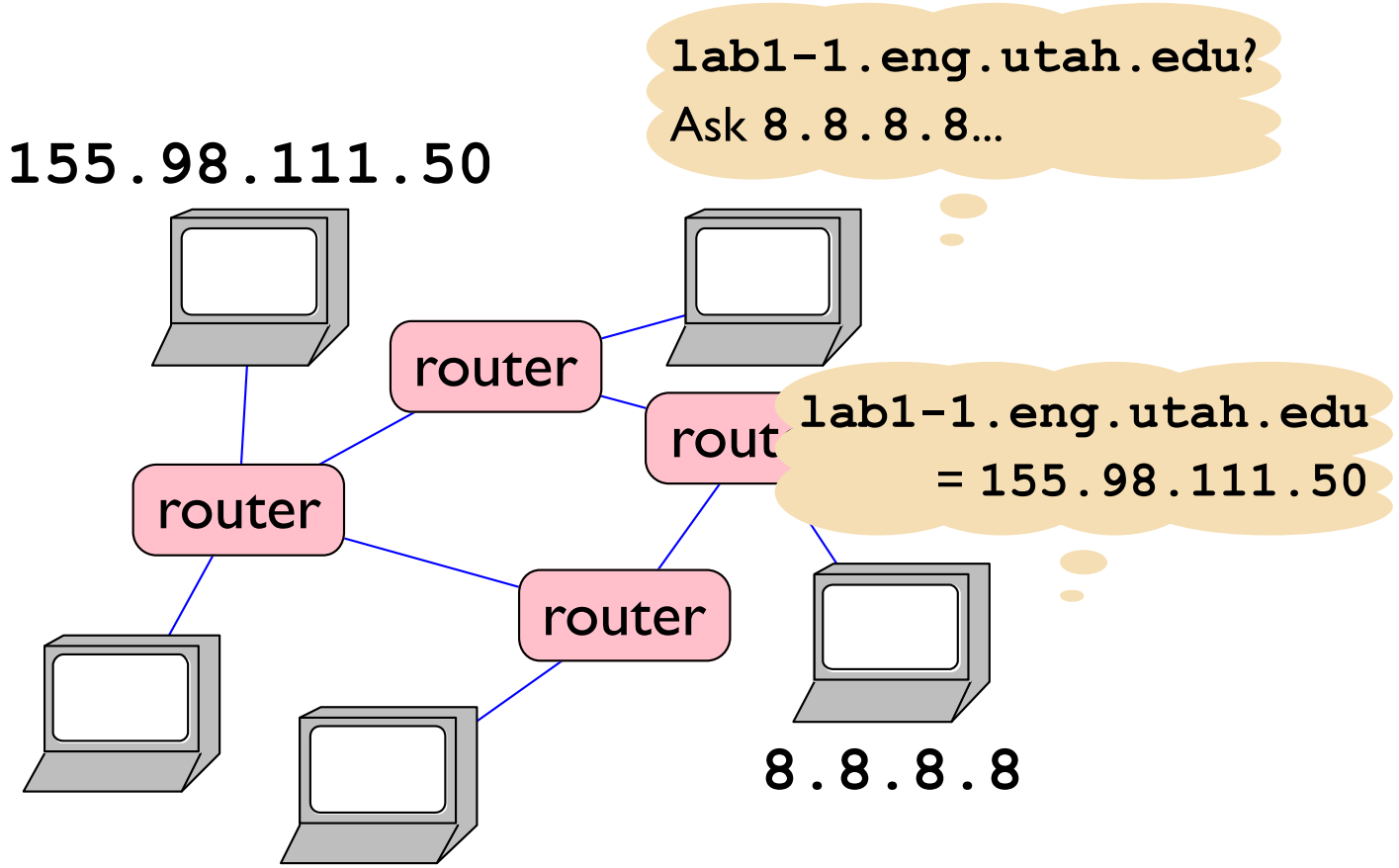
- identify DNS server by address
- DNS server maps names to addresses and vice versa



Name Resolution

DNS (Domain Name System) maps names to remote addresses

- identify DNS server by address
- DNS server maps names to addresses and vice versa



Name Resolution

Multiple names can map to the same address

```
$ ./hostinfo www.eng.utah.edu
155.98.110.30
$ ./hostinfo www.cade.utah.edu
155.98.110.30
```

We'll implement **`hostinfo`**...

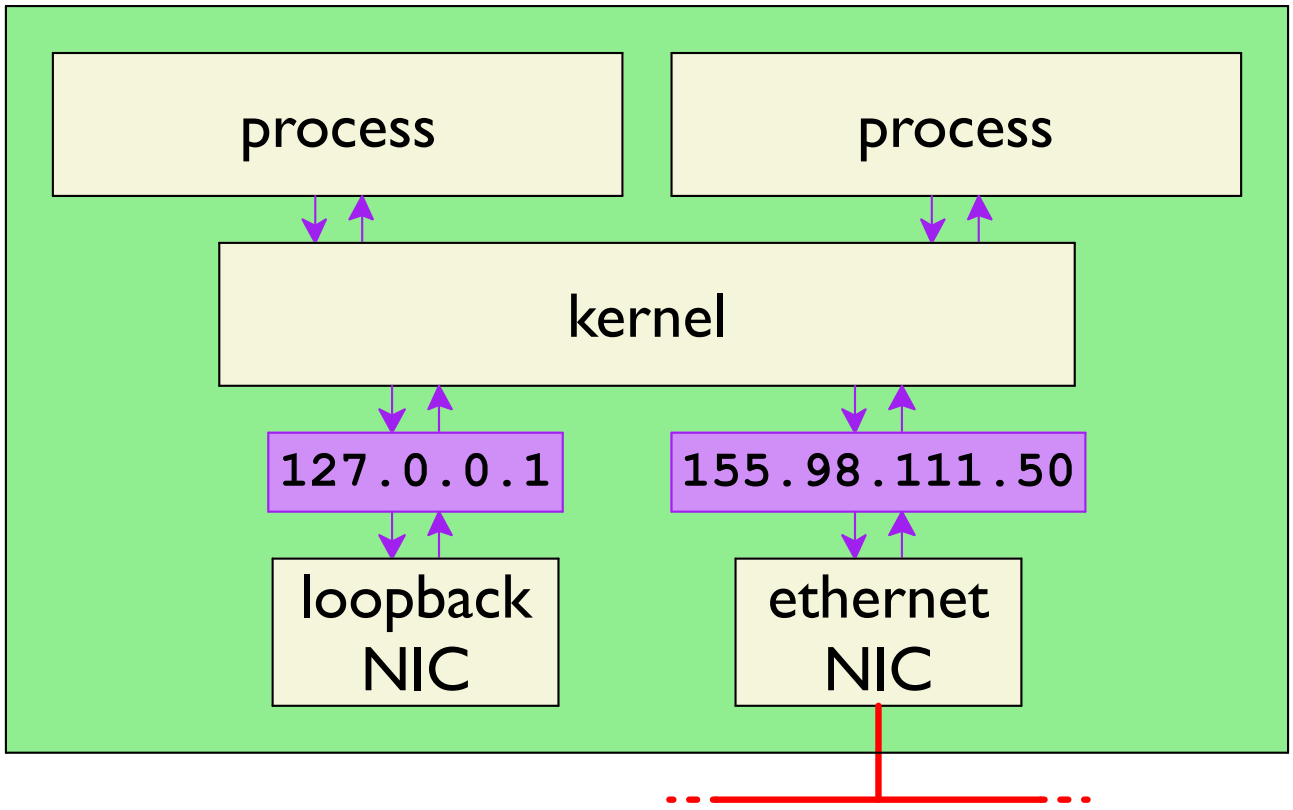
Name Resolution

A single name can map to multiple addresses

```
$ ./hostinfo twitter.com
104.244.42.129
104.244.42.65
104.244.42.193
104.244.42.1
```

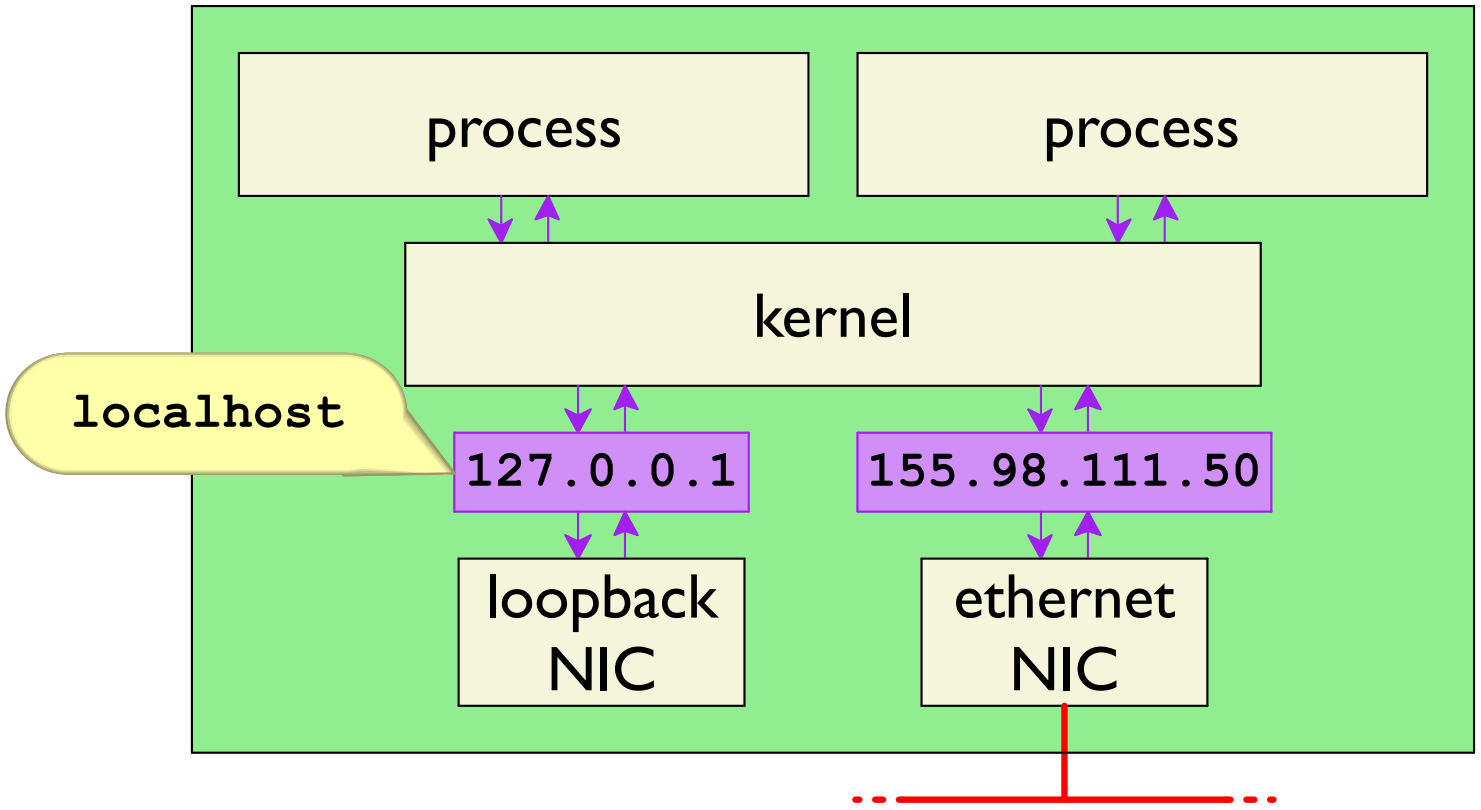
Name Resolution

Naming is not just about finding remote hosts



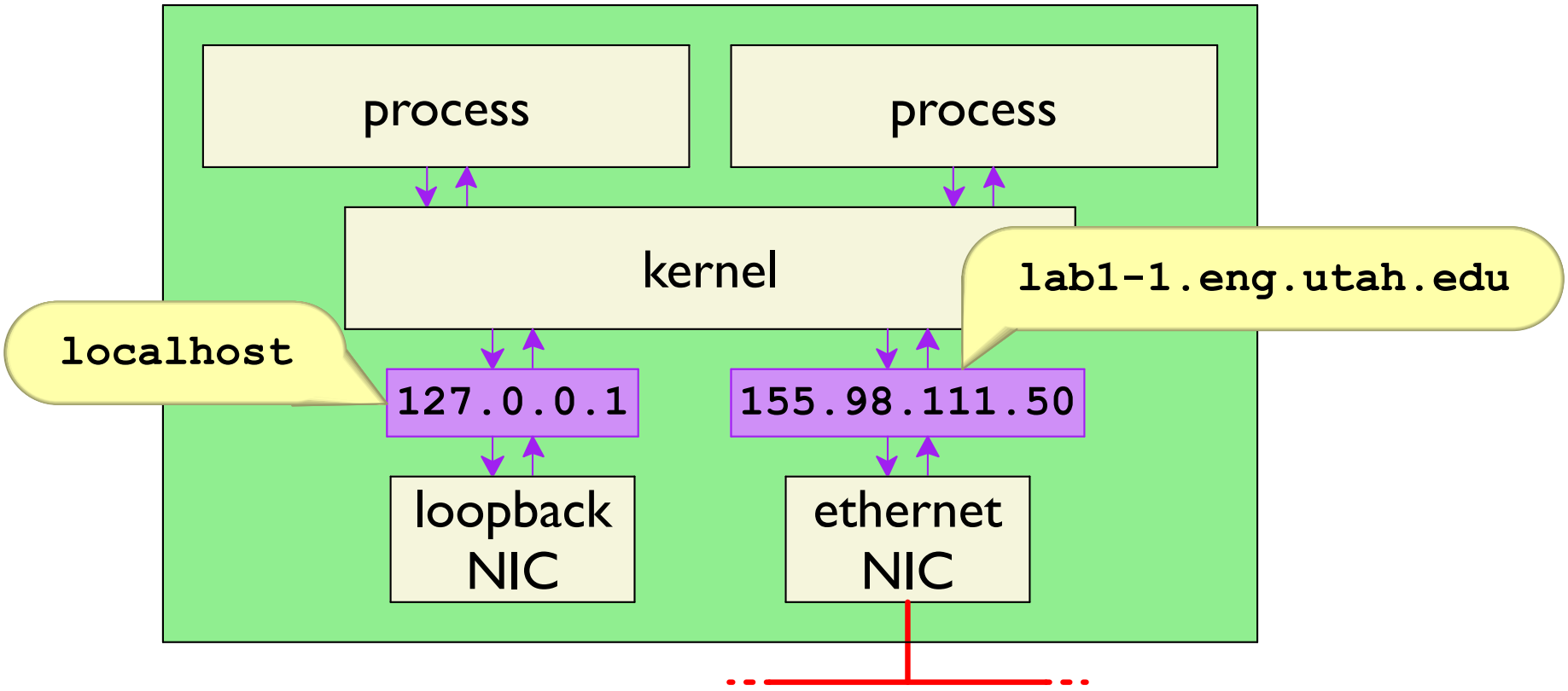
Name Resolution

Naming is not just about finding remote hosts



Name Resolution

Naming is not just about finding remote hosts



Name Resolution

Naming is not just about IPv4

- **`localhost`** via IPv4 = **`127.0.0.1`**
- **`localhost`** via IPv6 = **`::1`**

System calls need to support many protocols

C Library for Name Resolution

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

int getaddrinfo(const char *hostname, const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

Converts a combination of name and port to one or more protocol-specific addresses

- **`hostname`** can be **`NULL`** for "any here"
- **`servname`** is a port number or alias, **`NULL`** for "any"
- **`hints`** can request IPv4, TCP, etc.
- **`res`** is the result: set to a linked list of addresses

C Library for Name Resolution

/usr/include/netdb.h

```
struct addrinfo {
  int ai_flags;
  int ai_family;              /* protocol family */
  int ai_socktype;            /* socket type */
  int ai_protocol;            /* protocol */
  socklen_t ai_addrlen;       /* length of ai_addr */
  struct sockaddr *ai_addr; /* address */
  char *ai_canonname;
  struct addrinfo *ai_next; /* next in list */
};
```

Represents an IPv4 address When **`ai_family`** = **`AF_INET`**

C Library for Name Resolution

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

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, socklen_t hostlen,
                char *serv, socklen_t servlen,
                int flags);
```

The reverse of **`getaddrinfo`**

Set **`flags`** to

`NI_NUMERICHOST | NI_NUMERICSERV`

for numeric address and port

C Library for Name Resolution

hostinfo.c

```
#include "csapp.h"

int main(int argc, char **argv, char **envp) {
  struct addrinfo hints, *addrs, *addr;
  char host[256];

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;       /* Request IPv4 */
  hints.ai_socktype = SOCK_STREAM; /* TCP connection */
  Getaddrinfo(argv[1], NULL, &hints, &addrs);

  for (addr = addrs; addr != NULL; addr = addr->ai_next) {
    Getnameinfo(addr->ai_addr, addr->ai_addrlen,
                host, sizeof(host),
                NULL, 0,
                NI_NUMERICHOST);
    printf("%s\n", host);
  }
  ....
}
```

C Library for Name Resolution

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

void freeaddrinfo(struct addrinfo *ai);
```

Frees options allocated by **`getaddrinfo`**

UDP/IP and TCP/IP

IP is the addressing and packet-transfer layer

`127.0.0.1`

- Packets can get lost
- Packets can get reordered

UDP is a thin layer on IP

- Packets can get lost
- Packets can get reordered

TCP is a substantial layer on IP

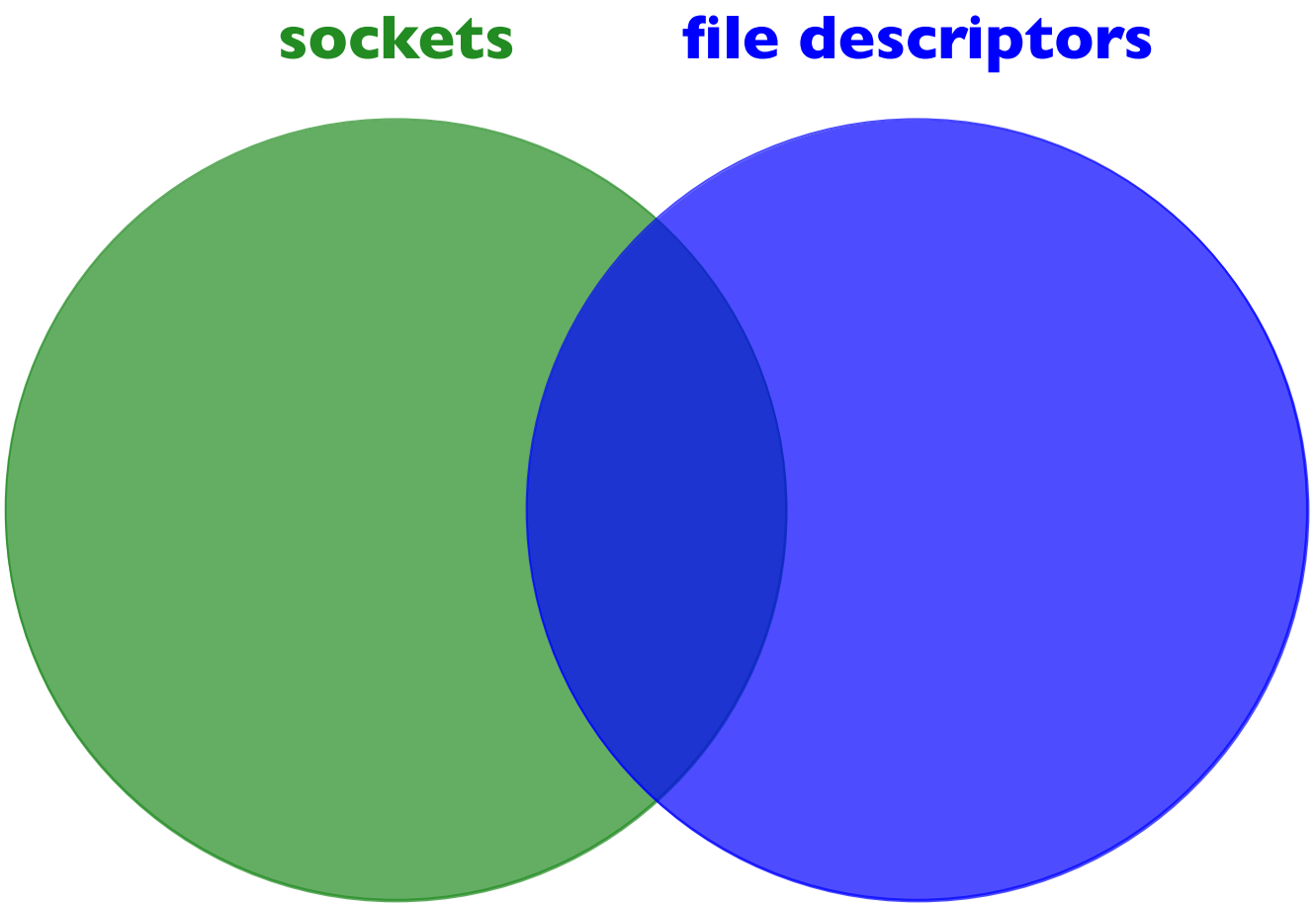
- Mostly hides packet nature behind a stream interface
- Retries as needed to get data sent
- Tags packets with sequence numbers for ordering

Sockets

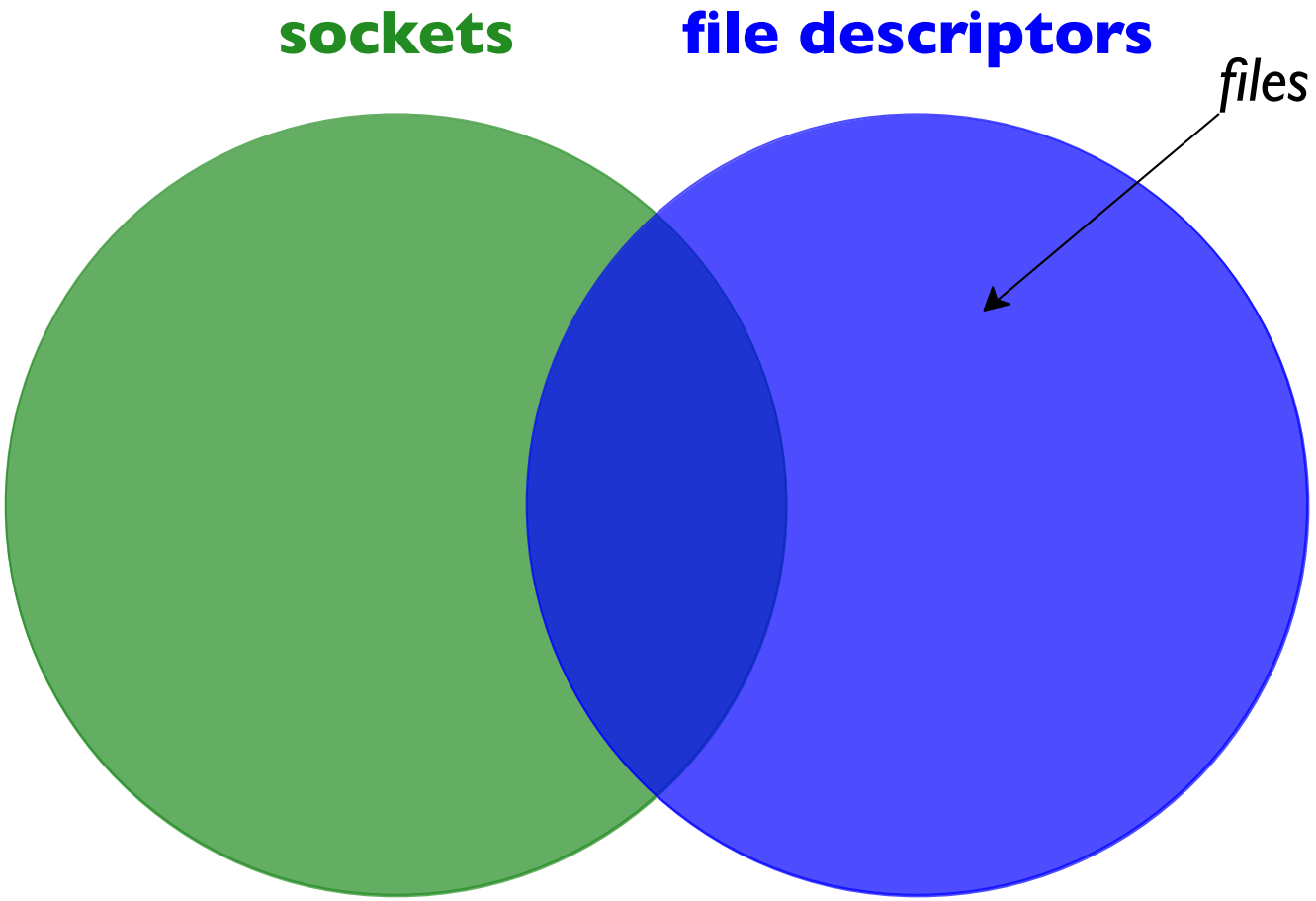
A generic communication is a ***socket***

A socket is represented as an **`int`**

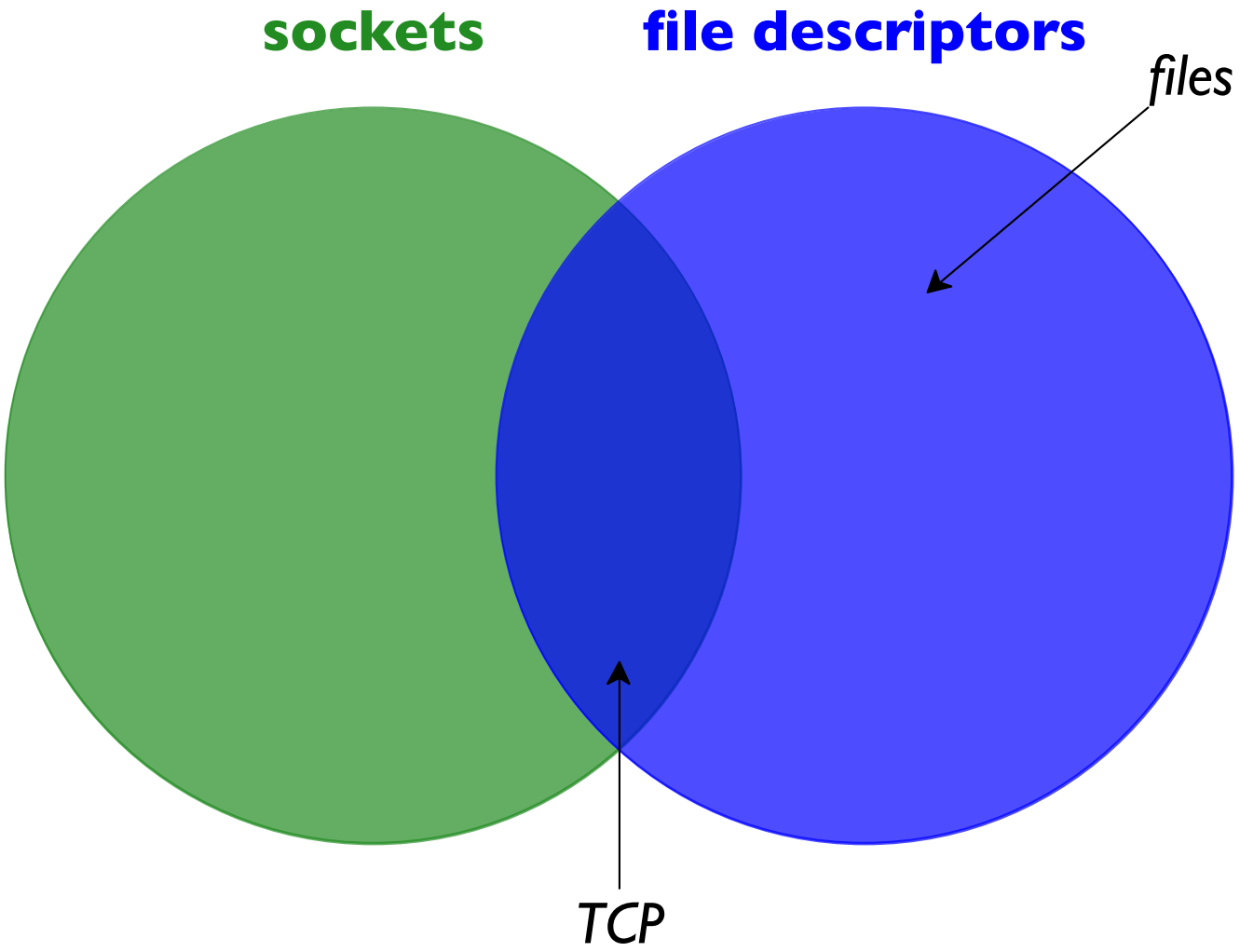
Sockets



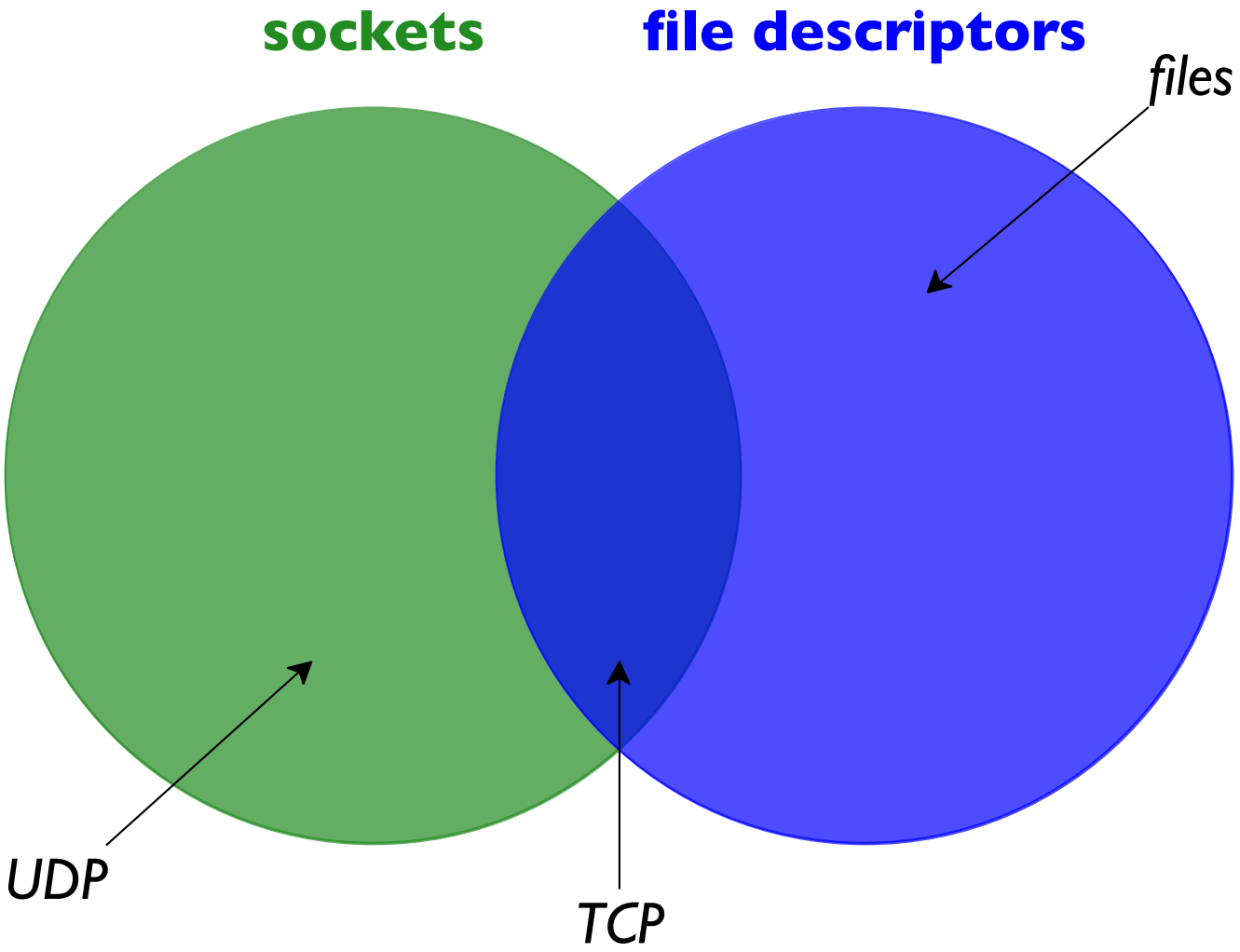
Sockets



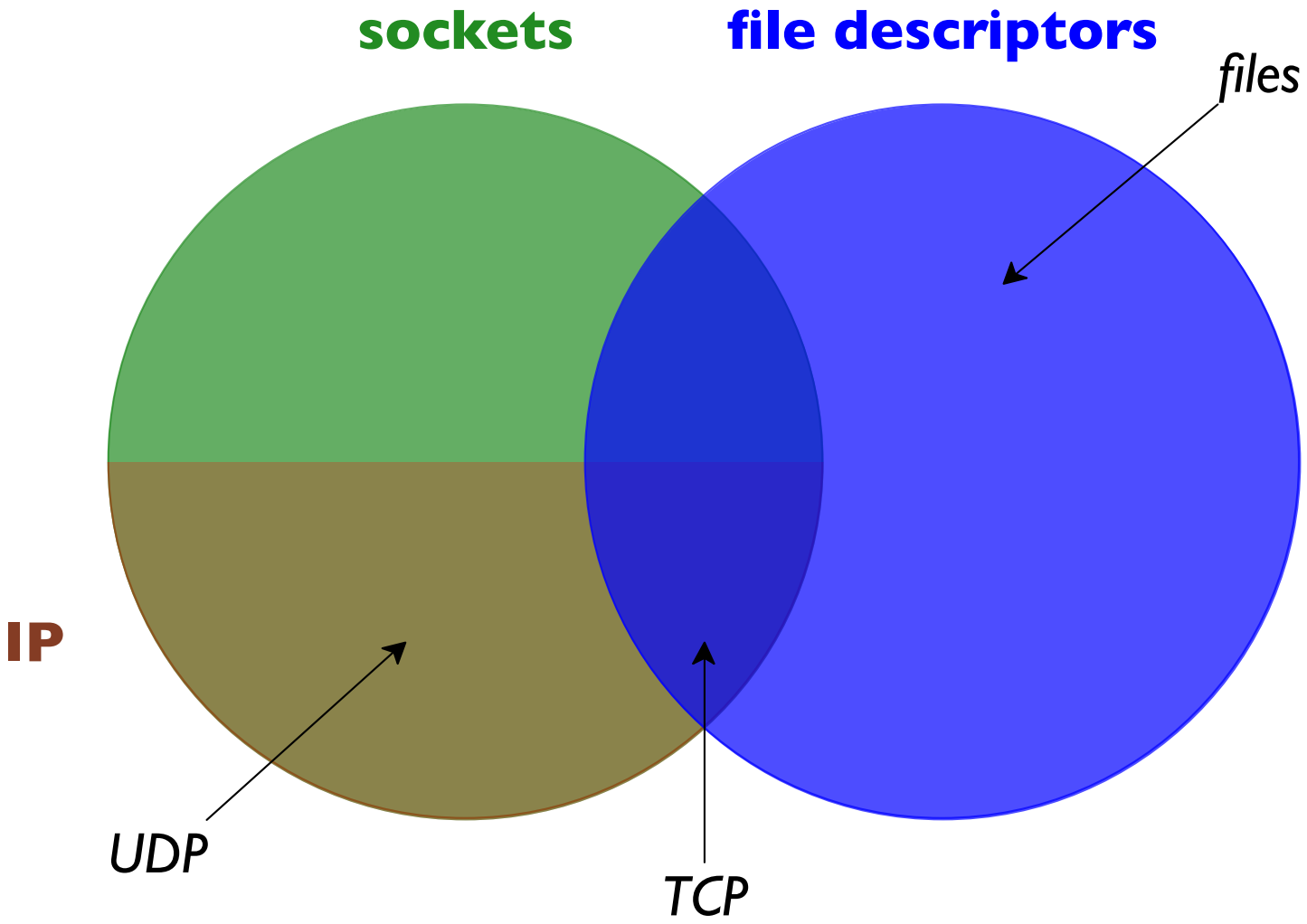
Sockets



Sockets



Sockets



Sockets

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

int socket(int domain, int type, int protocol);
```

Creates a new socket

- **`domain`** is a protocol family; **`PF_INET`** means IPv4
- **`type`** is
  - **`SOCK_DGRAM`** for UDP
  - **`SOCK_STREAM`** for TCP
- **`protocol`** is a kind of subtype

For portable code, get arguments from the result of **`getaddrinfo`**

Binding Sockets

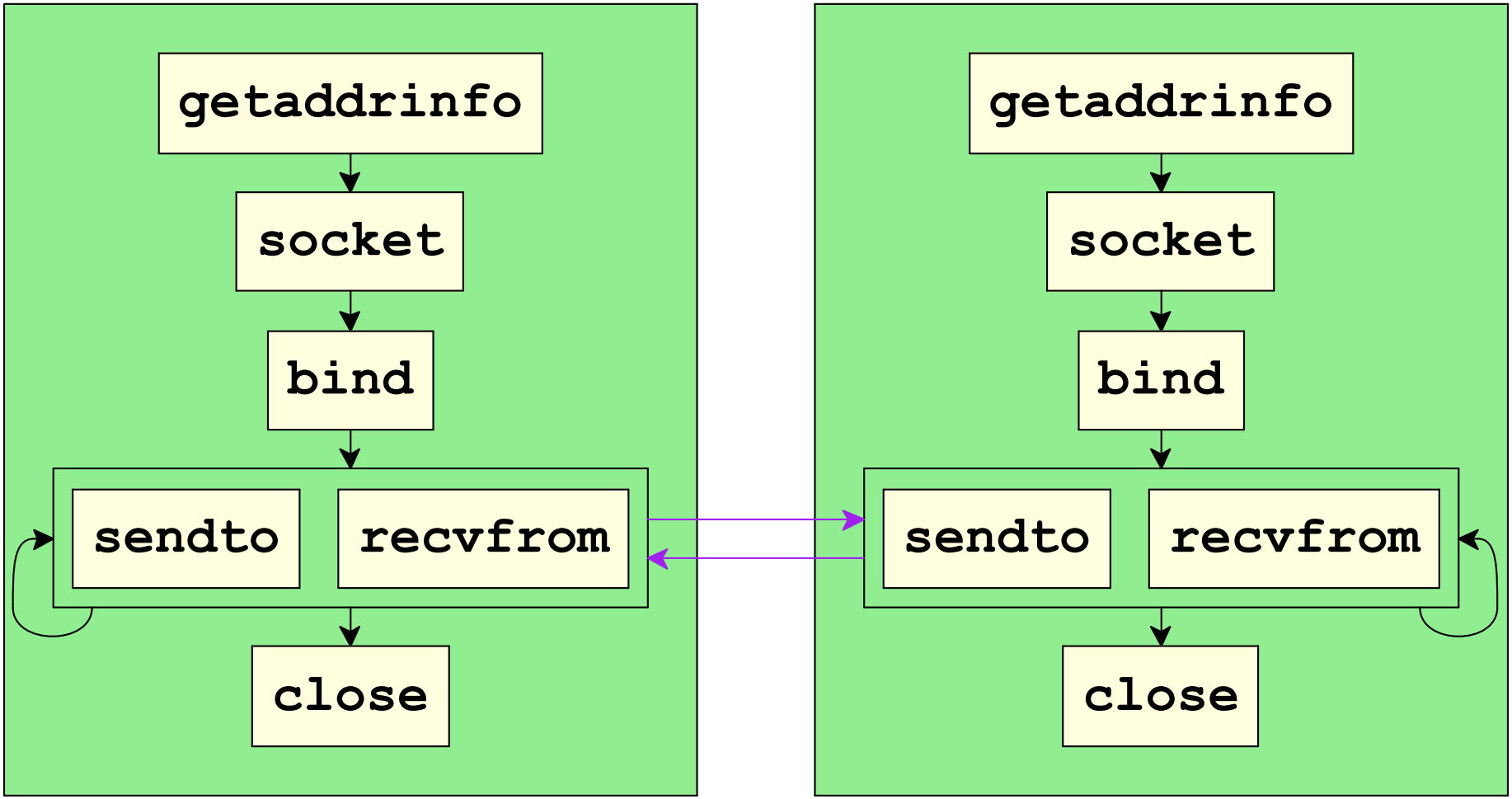
```
#include <sys/socket.h>

int bind(int socket,
         struct sockaddr *addr, socklen_t addr_len);
```

Attaches a socket to a specific address

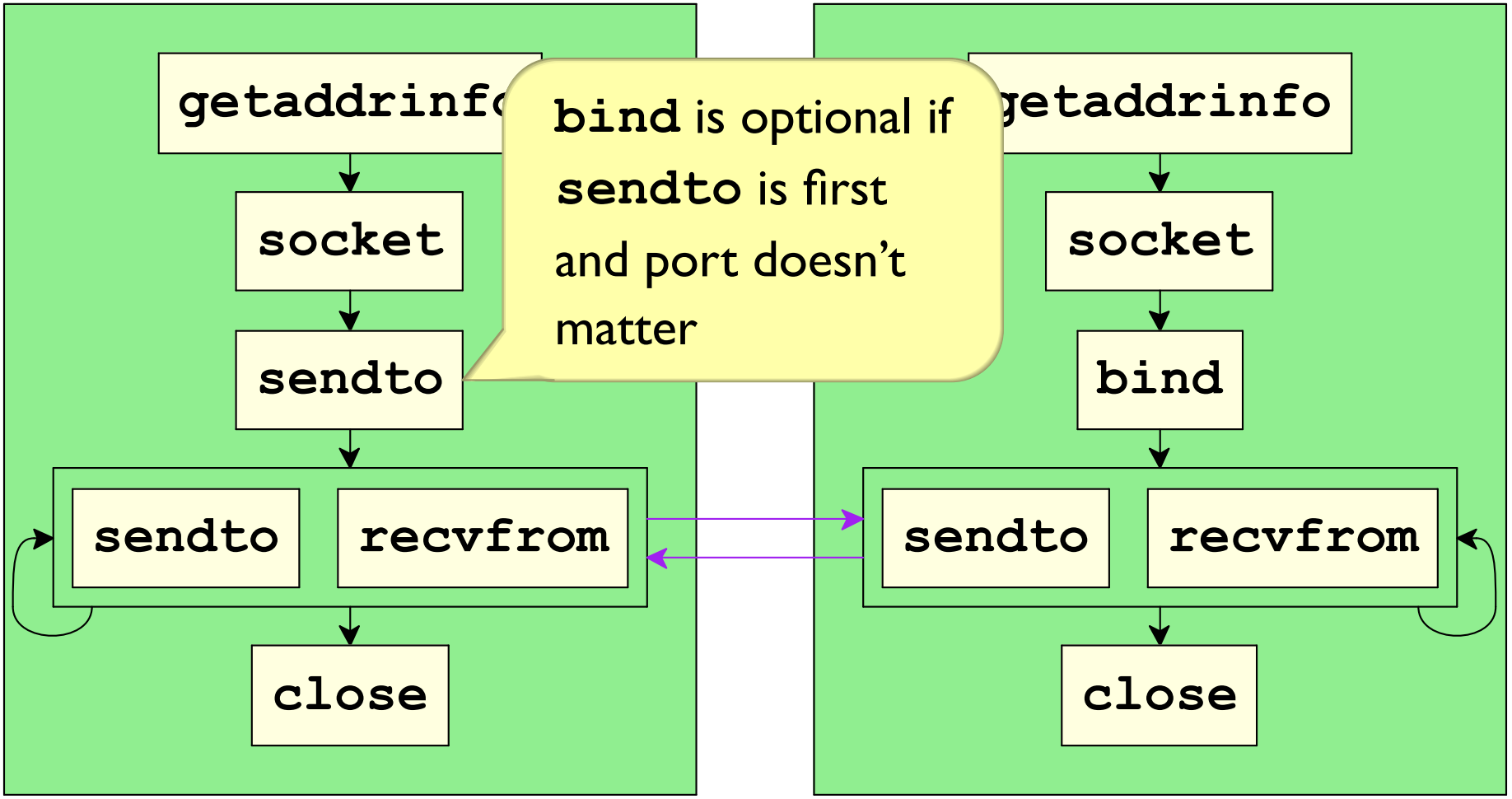
If other processes know the address, they can send a message to the socket

The **`addr`** and **`addr_len`** arguments come from **`getaddrinfo`**

Using UDP

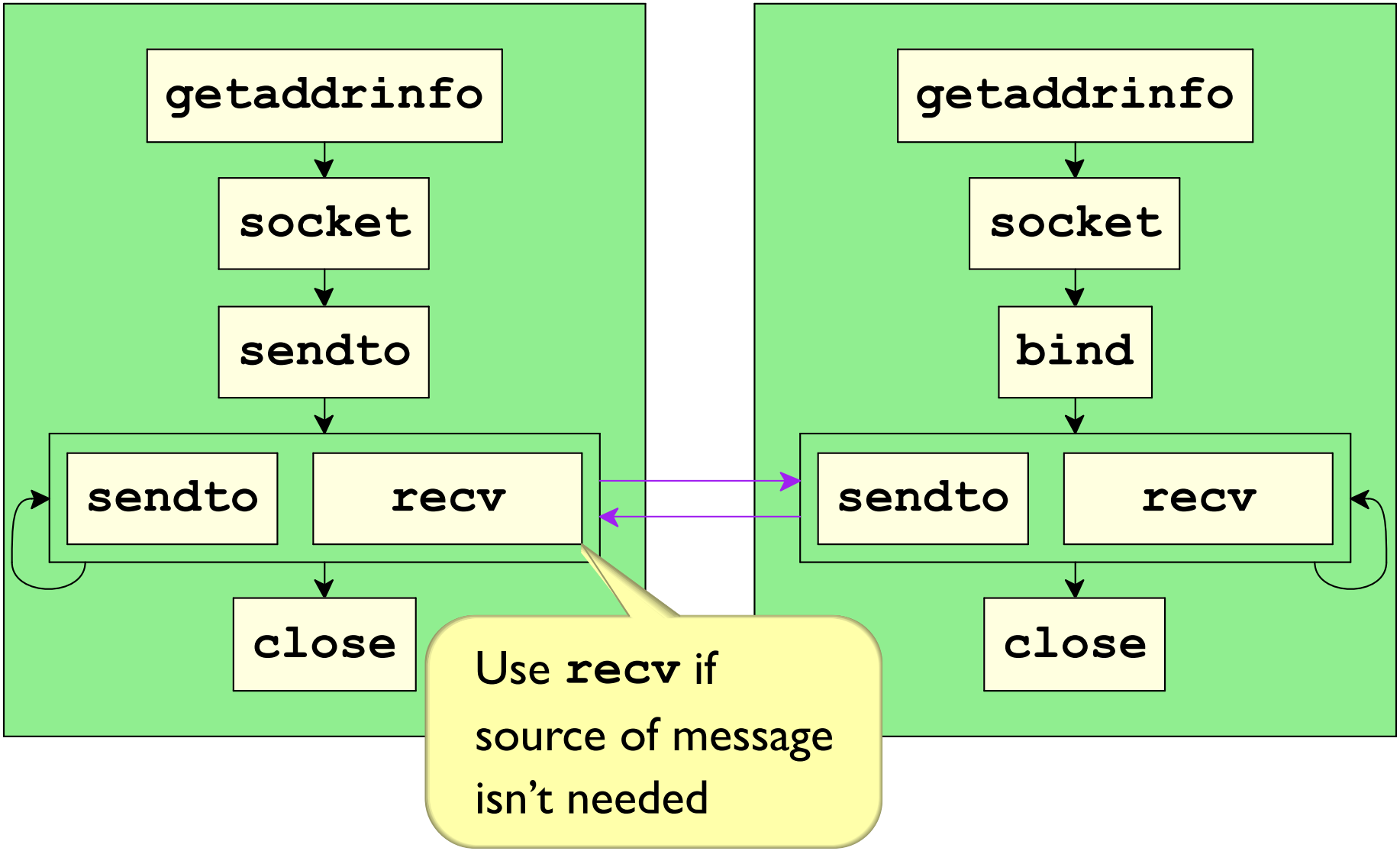

getaddrinfo
socket
bind
sendto
recvfrom
close
getaddrinfo
socket
bind
sendto
recvfrom
close

Using UDP



Automatically selected ports are in the ***ephemeral port*** range

Using UDP


getaddrinfo
socket
sendto
sendto
recv
close
getaddrinfo
socket
bind
sendto
recv
close
Use recv if source of message isn't needed

UDP Server

udp_recv.c

```
#include "csapp.h"

int main(int argc, char **argv) { /* argv[0] == portno */
  struct addrinfo hints, *addrs;
  int s;

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;        /* Request IPv4 */
  hints.ai_socktype = SOCK_DGRAM;   /* Accept UDP connections */
  hints.ai_flags = AI_PASSIVE;      /* ... on any IP address */
  Getaddrinfo(NULL, argv[0], &hints, &addrs);

  s = Socket(addrs->ai_family, addrs->ai_socktype, addrs->ai_protocol);
  Bind(s, addrs->ai_addr, addrs->ai_addrlen);
  Freeaddrinfo(addrs);

  while (1) {
    char buffer[MAXBUF];
    size_t amt;
    amt = Recv(s, buffer, MAXBUF, 0);
    Write(1, buffer, amt);
    Write(1, "\n", 1);
  }

  return 0;
}
```

UDP Client

udp_send.c

```
#include "csapp.h"

int main(int argc, char **argv, char **envp) {
  char *hostname = argv[1], *portno = argv[2];
  struct addrinfo hints, *addrs;
  char host[256], serv[32];
  int s;
  size_t amt;

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;       /* Request IPv4 */
  hints.ai_socktype = SOCK_DGRAM; /* UDP connection */
  Getaddrinfo(hostname, portno, &hints, &addrs);

  Getnameinfo(addrs->ai_addr, addrs->ai_addrlen,
              host, sizeof(host), serv, sizeof(serv),
              NI_NUMERICHOST | NI_NUMERICSERV);
  printf("sending to %s:%s\n", host, serv);

  s = Socket(addrs->ai_family, addrs->ai_socktype, addrs->ai_protocol);
  amt = Sendto(s, argv[3], strlen(argv[3]), 0,
               addrs->ai_addr, addrs->ai_addrlen);
  Freeaddrinfo(addrs);

  return (amt != strlen(argv[3]));
}
```

UDP Server

udp_recv.c

```
#include "csapp.h"

int main(int argc, char **argv) { /* argv[0] == portno */
  struct addrinfo hints, *addrs;
  int s;

  memset(&hints, 0
  hints.ai_family
  hints.ai_socktyp
  hints.ai_flags
  Getaddrinfo(NULL, argv[0], &hints, &addrs);

  s = Socket(addrs->ai_family, addrs->ai_socktype, addrs->ai_protocol);
  Bind(s, addrs->ai_addr, addrs->ai_addrlen);
  Freeaddrinfo(addrs);

  while (1) {
    char buffer[MAXBUF];
    size_t amt;
    amt = Recv(s, buffer, MAXBUF, 0);
    Write(1, buffer, amt);
    Write(1, "\n", 1);
  }

  return 0;
}
```

Using "**localhost**" constrains to a loopback connection

Revised UDP Server

udp_recvfrom.c

```
....
int counter = 0;
....

while (1) {
  char buffer[MAXBUF];
  size_t amt;
  struct sockaddr_in from_addr;
  unsigned int from_len = sizeof(from_addr);

  amt = Recvfrom(s, buffer, MAXBUF, 0,
                 (struct sockaddr *)&from_addr, &from_len);
  Write(1, buffer, amt);
  Write(1, "\n", 1);

  Getnameinfo((struct sockaddr *)&from_addr, from_len,
              host, sizeof(host),
              serv, sizeof(serv),
              NI_NUMERICHOST | NI_NUMERICSERV);
  printf(" from %s:%s [%d]\n", host, serv, ++counter);
}

....
```

Revised UDP Client

udp_from_send.c

```
char *myportno = argv[1];
char *hostname = argv[2];
char *portno = argv[3];
....
Getaddrinfo(NULL, myportno, &hints, &my_addrs);

....

if (argc == 6)
  copies = atoi(argv[5]);
else
  copies = 1;

while (copies--)
  amt = Sendto(s, argv[4], strlen(argv[4]), 0,
               addrs->ai_addr, addrs->ai_addrlen);

....
```