Linux Network Programming

Setting the Scene

Chris Brown

pluralsight

In This Module …

The client/server model:
Which is the client?
Which is the server?

Connection-oriented
vs
Connectionless services

Language choice:

C

Python

Tools:
Development environment
"What's going on?" tools

pluralsight

Assumed Knowledge

C language

Syntax, data types, structures, pointers

Python

Not essential (but see "Python Fundamentals" for a good intro)

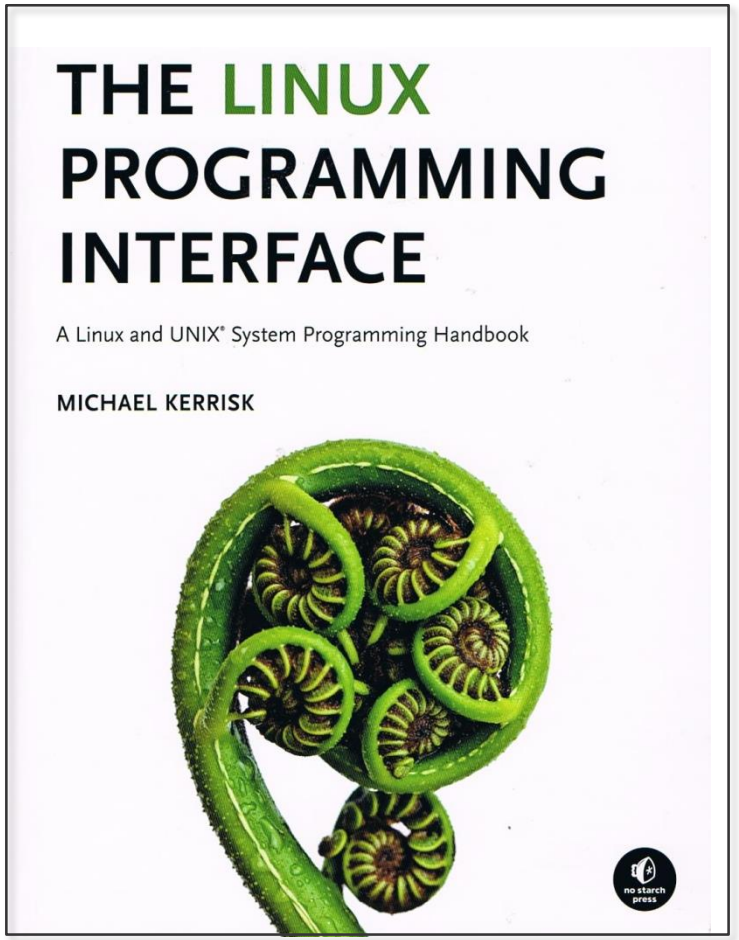
Linux

Knowledge of systems programming

This course is intended as a follow-on to the "Linux Systems Programming" course

pluralsight

Book Recommendation



1500 pages!

Authoritative

Well written

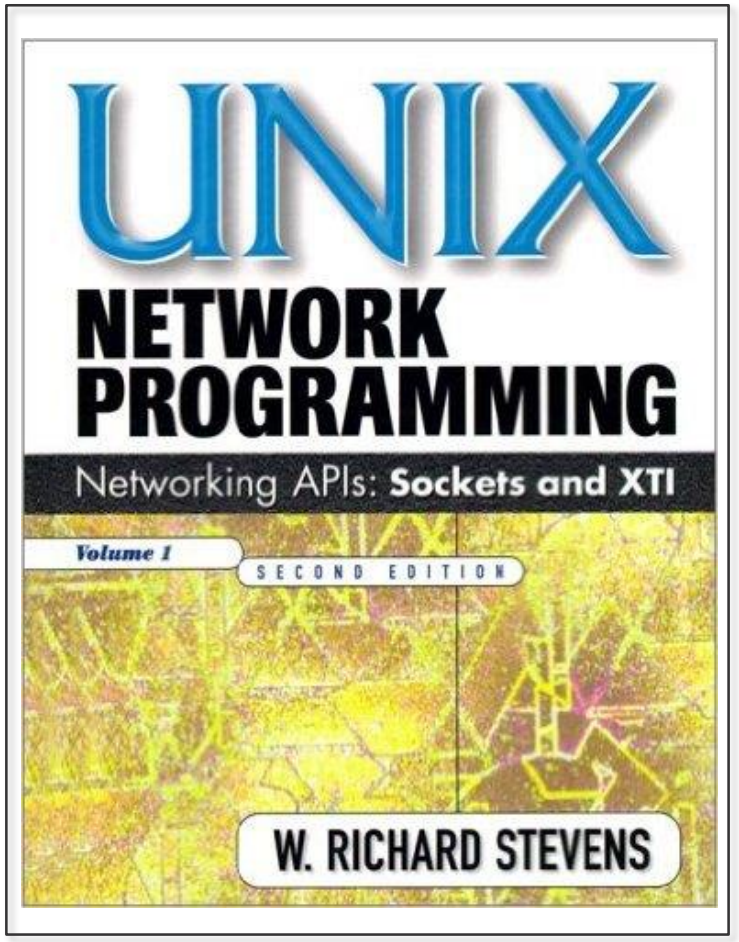
Lots of examples

Excellent!

Chapters 56 - 61

pluralsight

Another Book Recommendation



Older (UNIX) focus
(but 99% relevant to Linux)

Authoritative

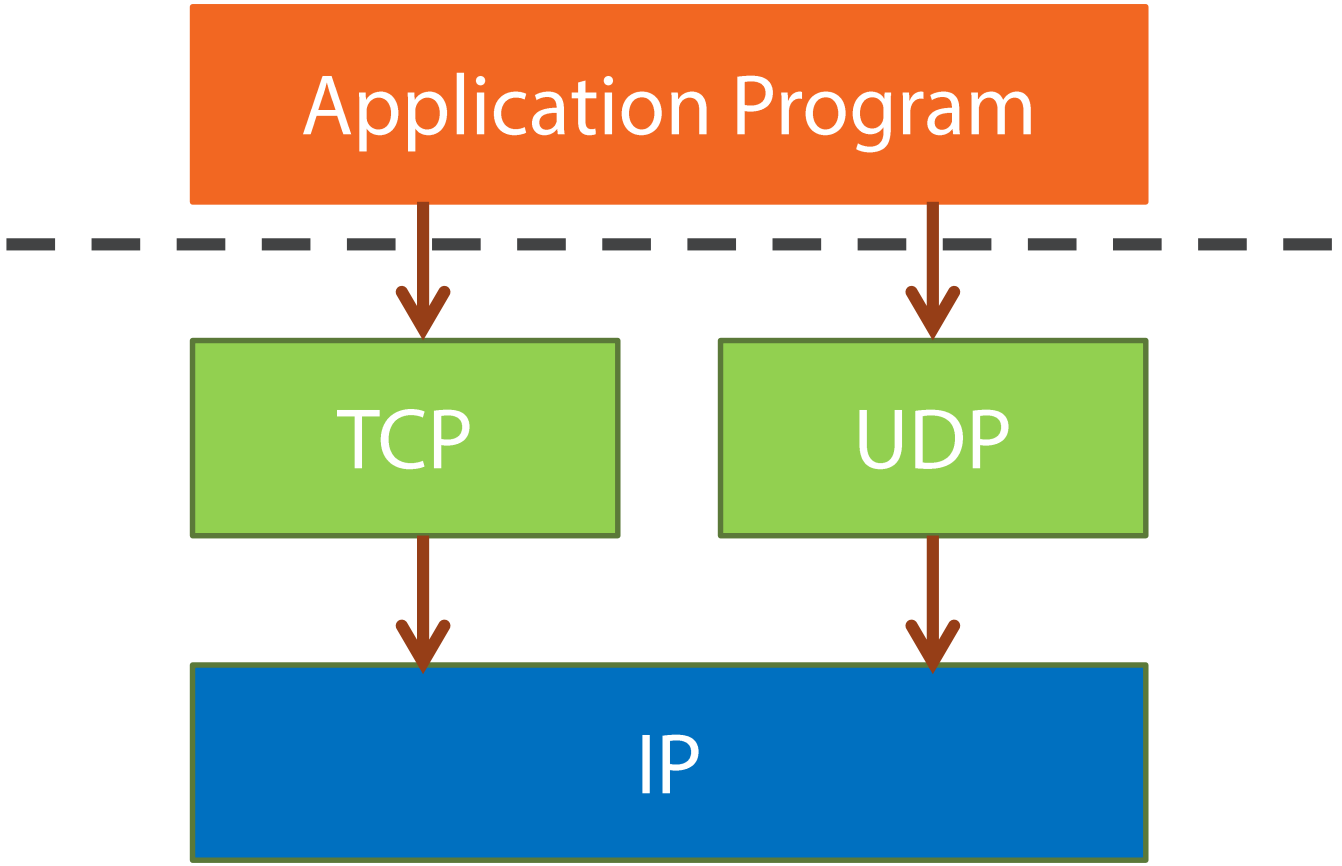
Well written

Lots of examples

Excellent!

pluralsight

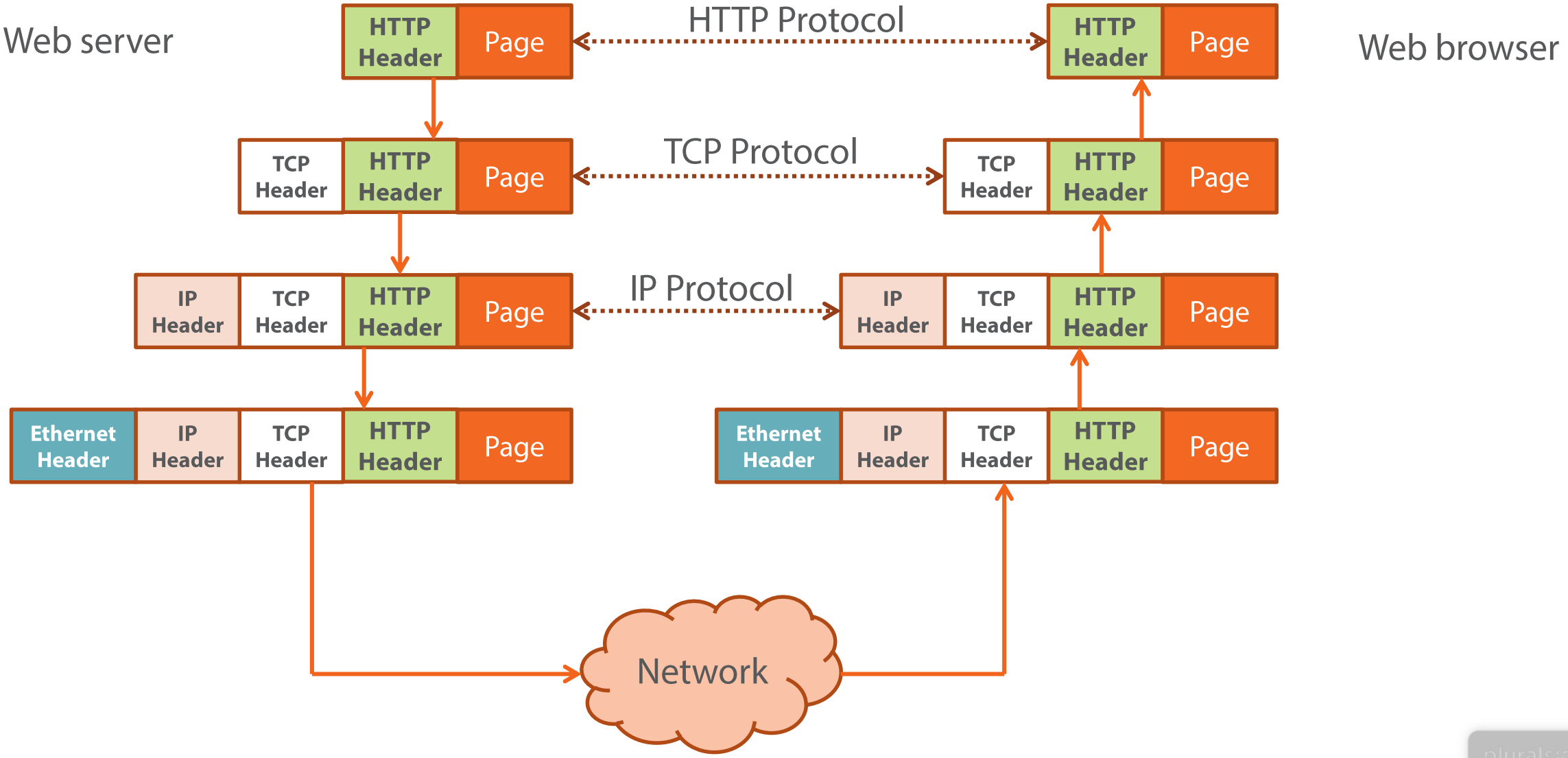
Layers




```
socket()
bind()
listen()
accept()
connect()
```

pluralsight

Encapsulation



The Client/server Relationship

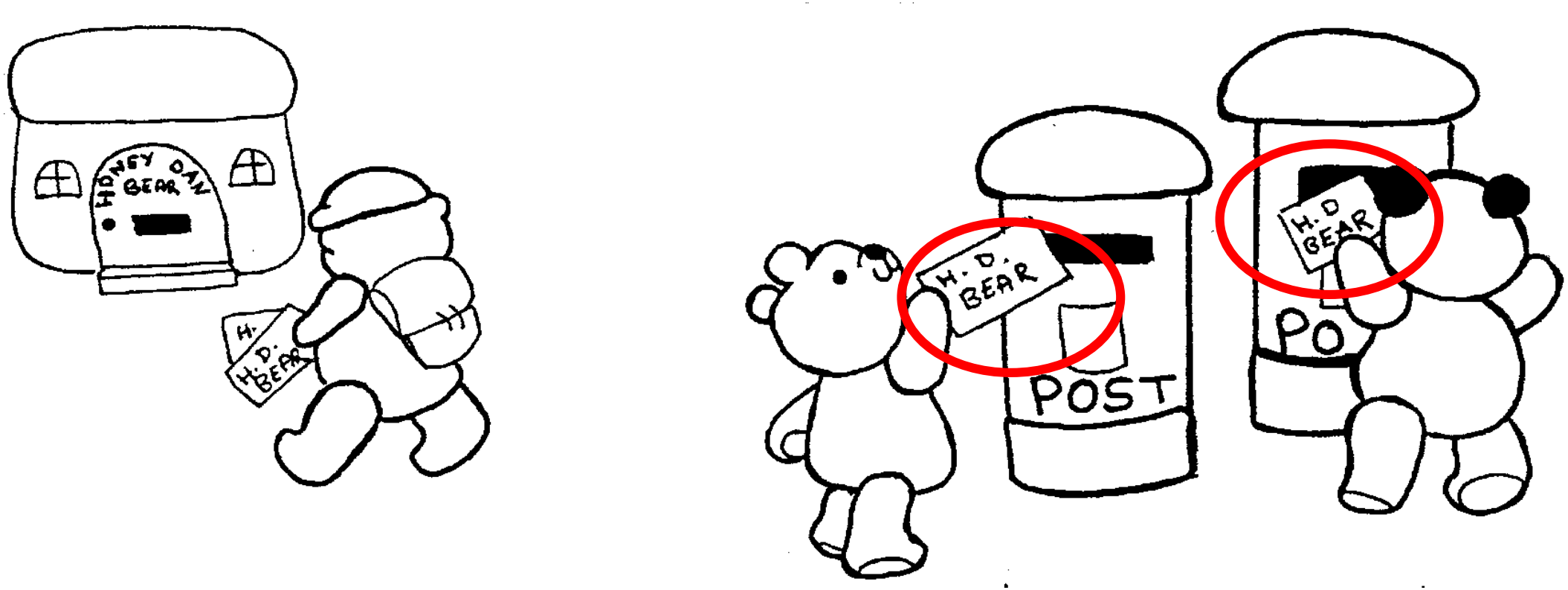
Server offers a service
- Establishes a "communication endpoint"
- Passively waits for business

Client "consumes" a service
- Actively connects to the server

Typically:
- Client is local
- Server may be remote

pluralsight

Connectionless Communication



Each message is individually addressed

pluralsight

Connectionless Communication

No guarantee of delivery

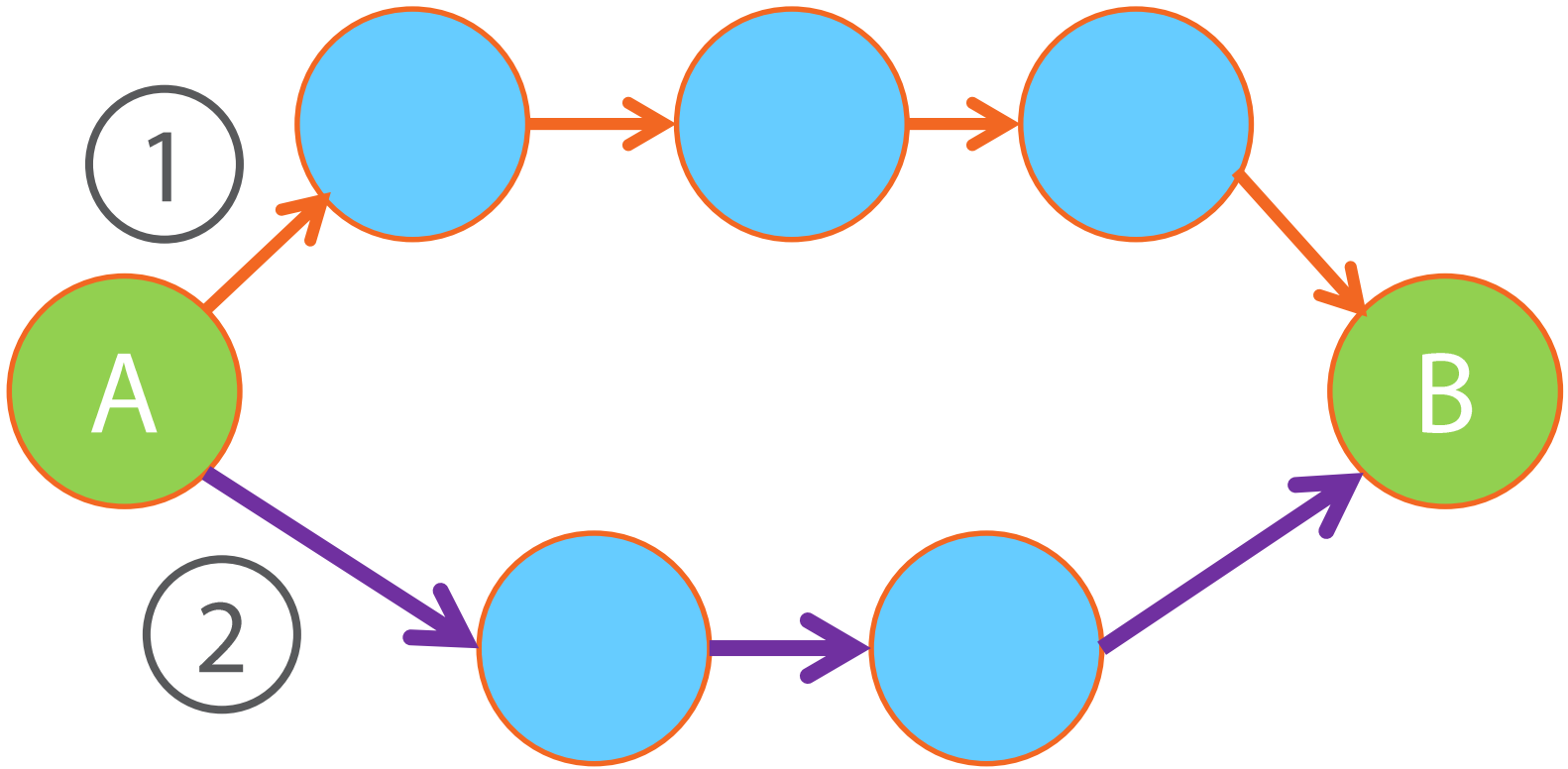
pluralsight

Connectionless Communication

Messages may not arrive in the order they were sent

pluralsight

Mis-ordered Delivery



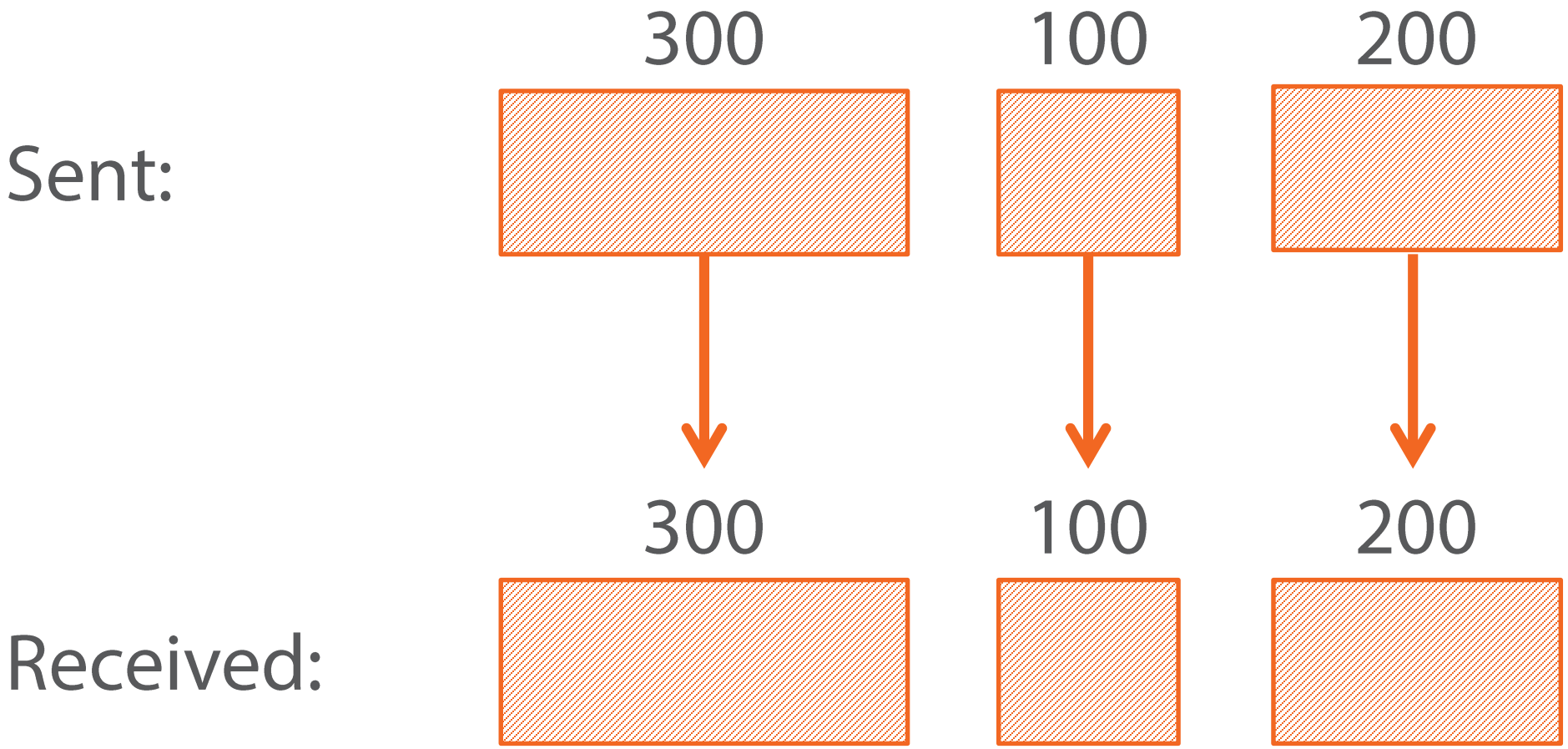
pluralsight

Connectionless Communication

Message boundaries are preserved

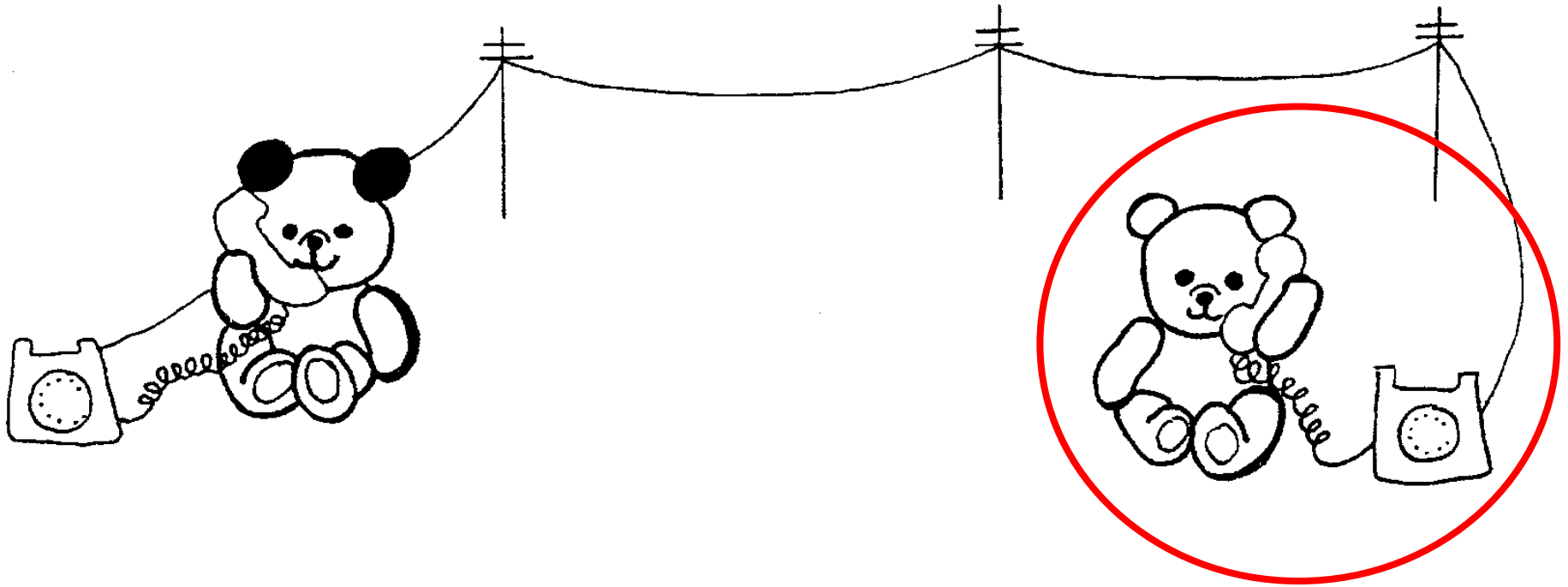
pluralsight

Message Boundaries Are Preserved



pluralsight

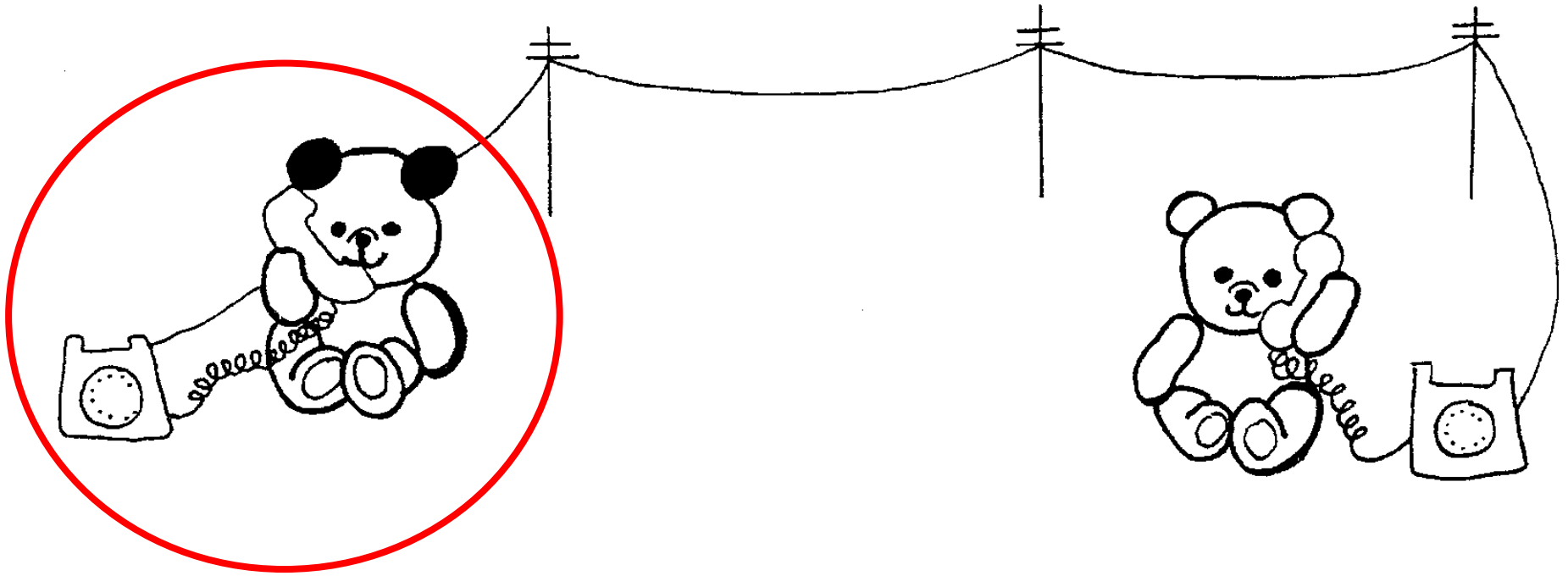
Connection-oriented Communication



Recipient address supplied "up front" when connection established

pluralsight

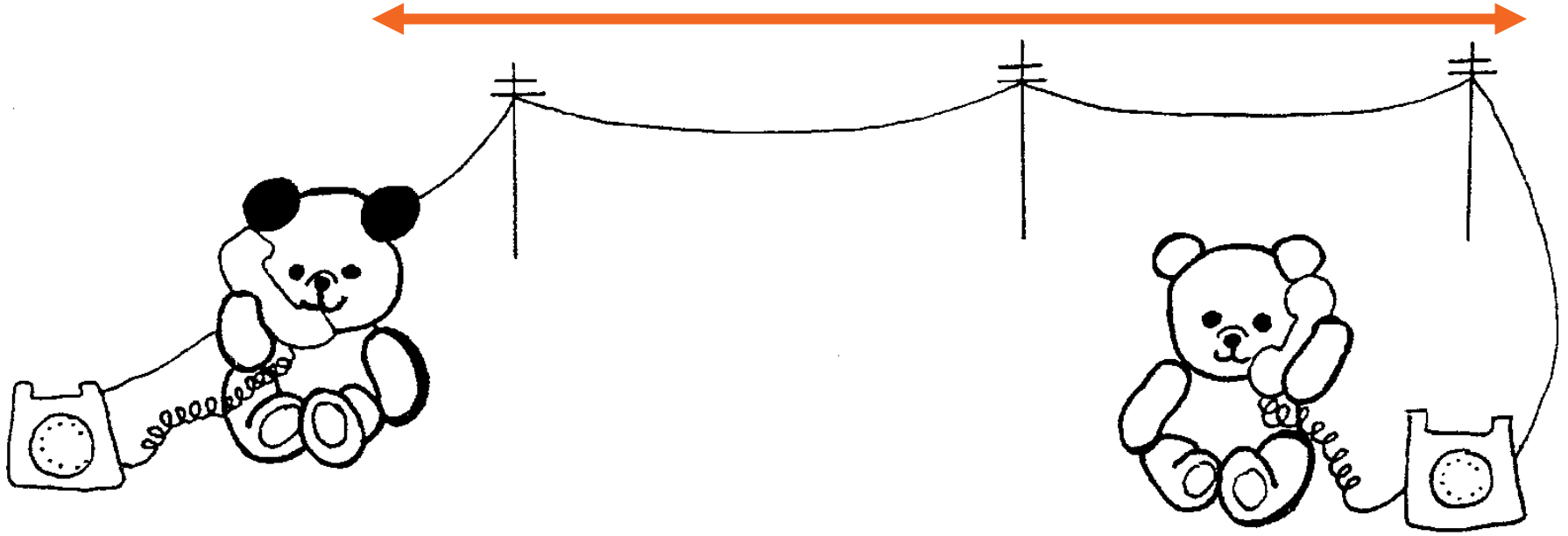
Connection-oriented Communication



Recipient must answer the call ("accept the connection")

pluralsight

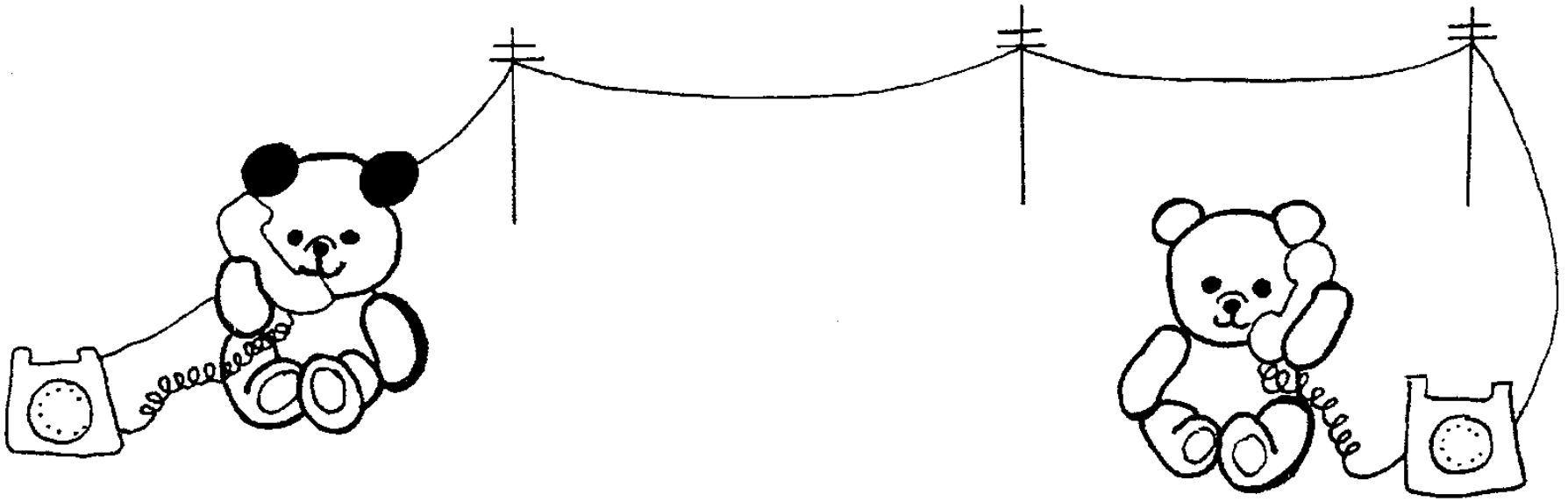
Connection-oriented Communication



Illusion of a copper wire connecting the two ends

pluralsight

Connection-oriented Communication

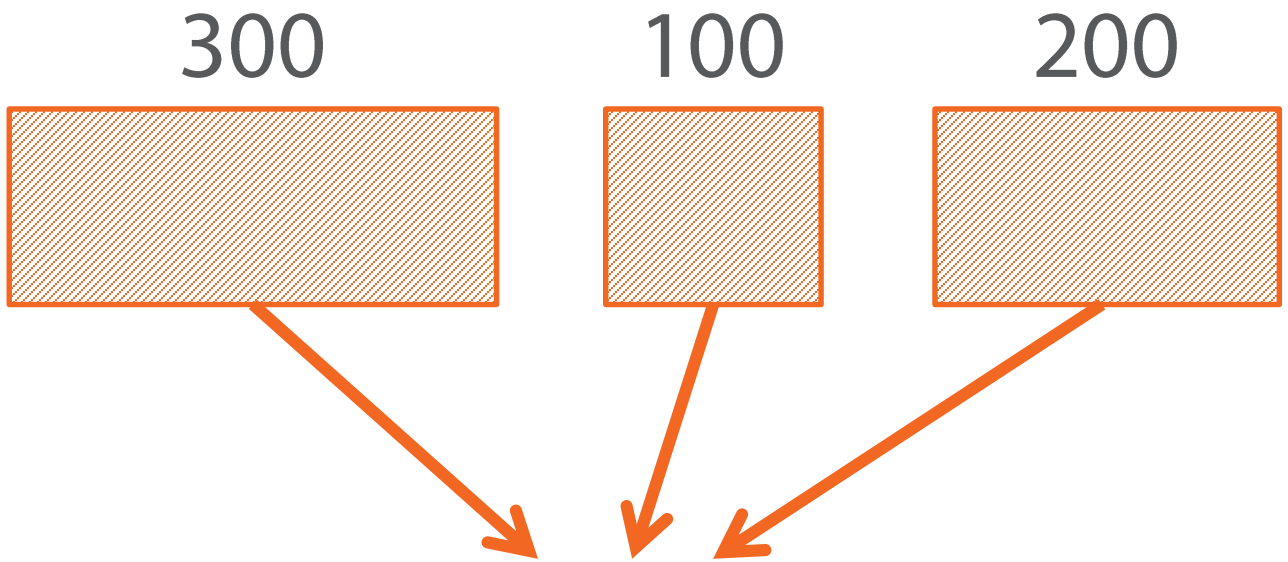


Message boundaries are not preserved

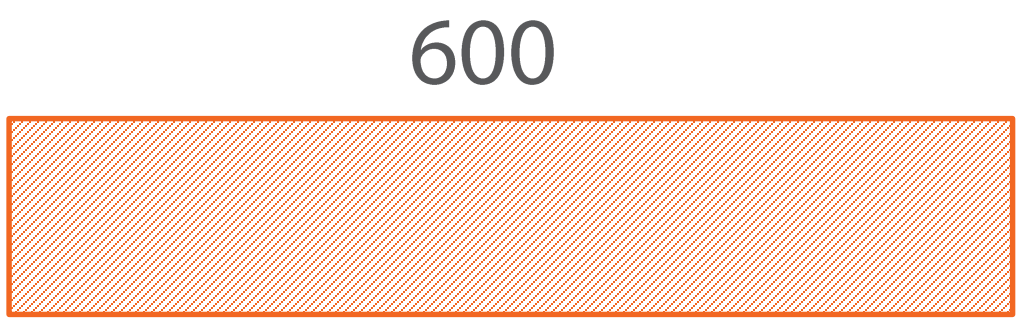
pluralsight

Message Boundaries Are Not Preserved

Sent:

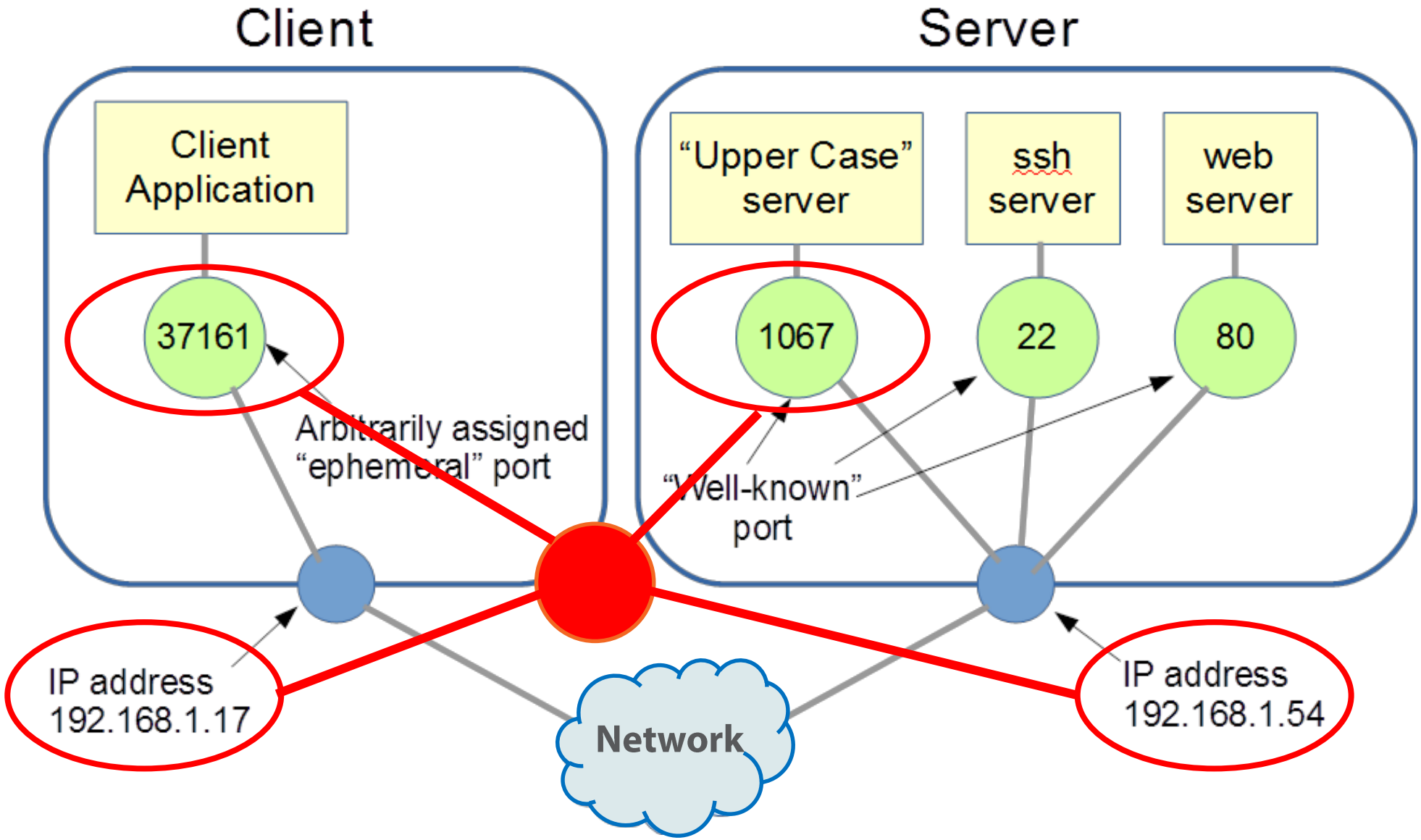


Received:



pluralsight

Client/server Association



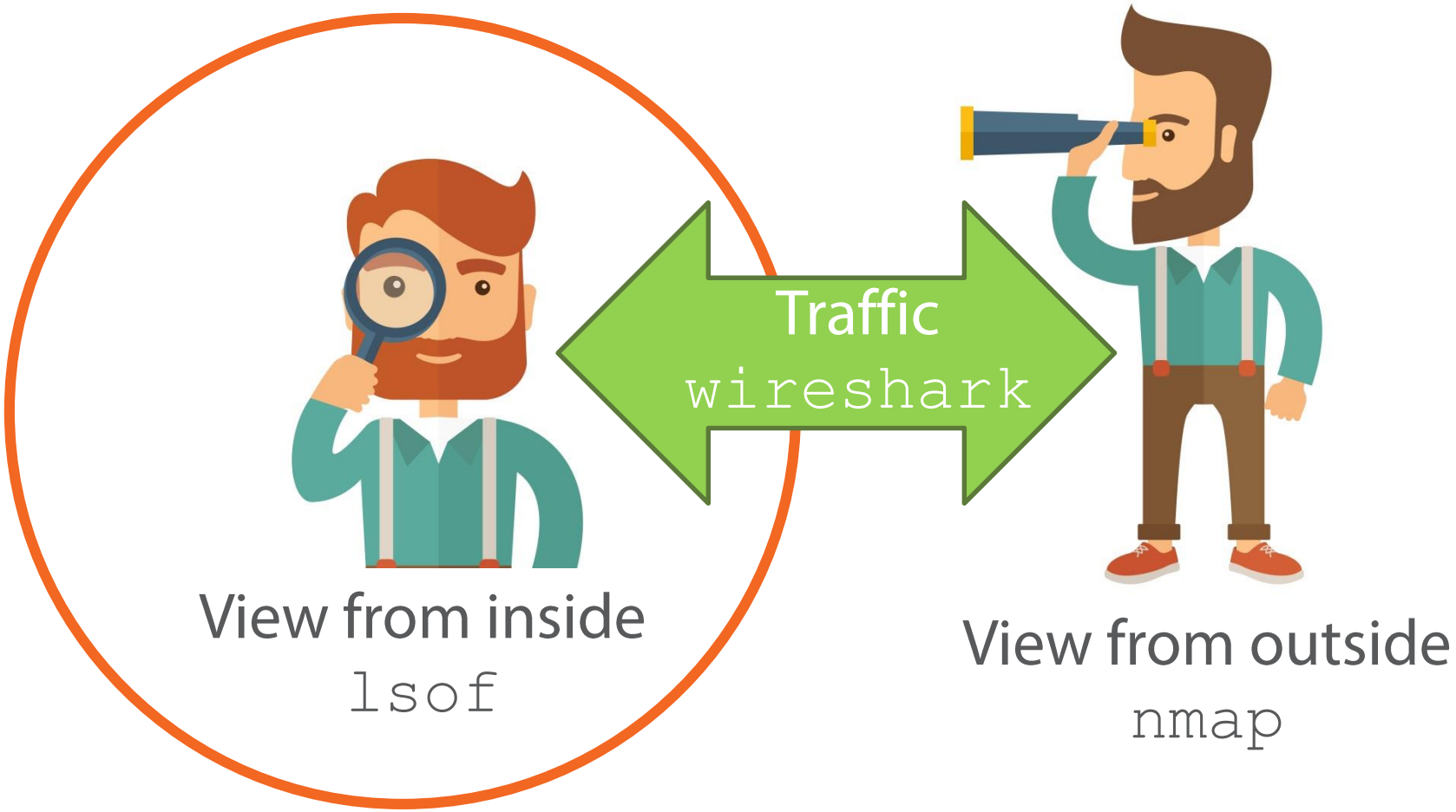
C and Python

- Low-level systems programming
- Dennis Ritchie, 1972
- Statically typed
- Procedural
- Fully compiled

- High-level, multi-purpose
- Guido van Rossum, 1991
- Dynamically typed
- Multi-paradigm including O-O
- Interpreted

pluralsight

Demonstration



pluralsight

Moving Forward…

In this module:

- Clients and servers
- Connectionless service
- Connection-oriented service
- Tools: lsof, nmap, wireshark

Coming up in the next module:

- Writing TCP-based servers

pluralsight

Writing TCP-based Servers

Chris Brown

pluralsight

In This Module …

Client-side and server-side operations

Key data structures

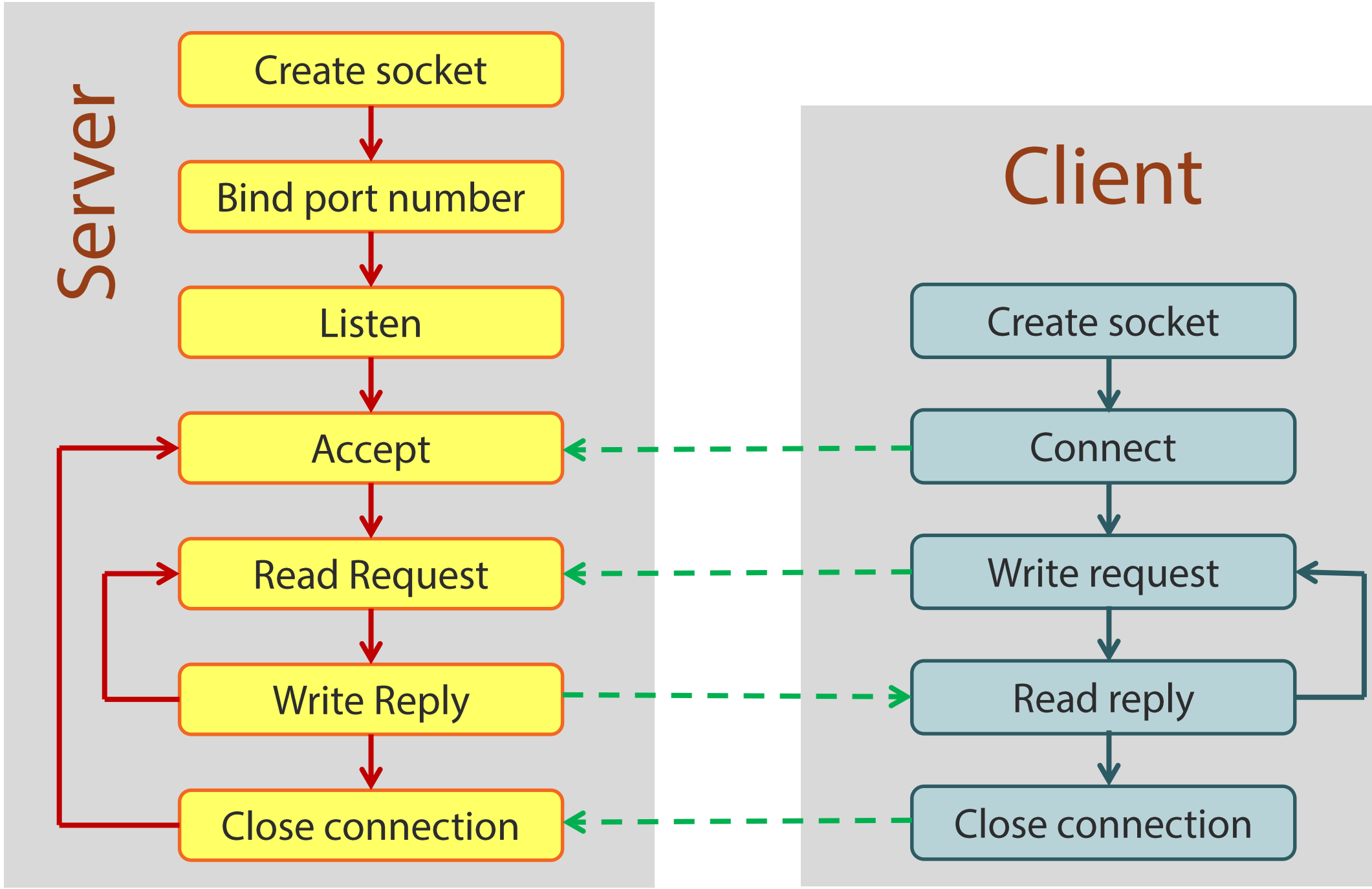
Key sockets system calls

Python as an alternative language

Demonstration:

A 'rot13' server

pluralsight


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

The sockaddr_in Structure

```
struct in_addr {
  inaddr_t  s_addr;
}
```

← 32-bit IP Address

```
struct sockaddr_in {
  sa_family_t    sin_family;
  in_port_t      sin_port;
  struct in_addr sin_addr;
  unsigned char __pad[..];
}
```

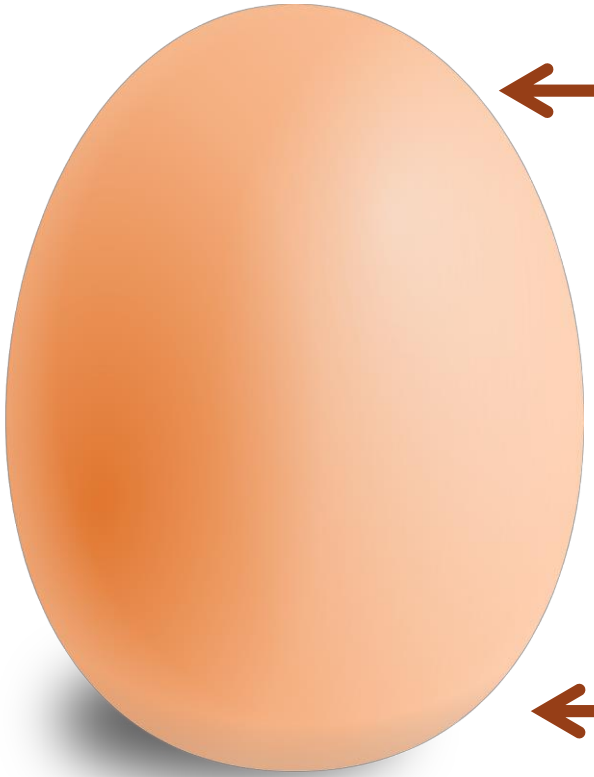
← Address family (AF_INET)

← IPV4 address

← Pad to size of generic sockaddr struct

pluralsight

Little-endian vs. Big-endian



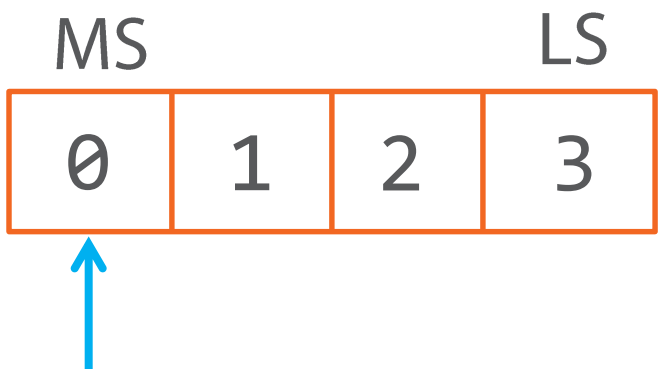
Little-Endian

Big-Endian

pluralsight

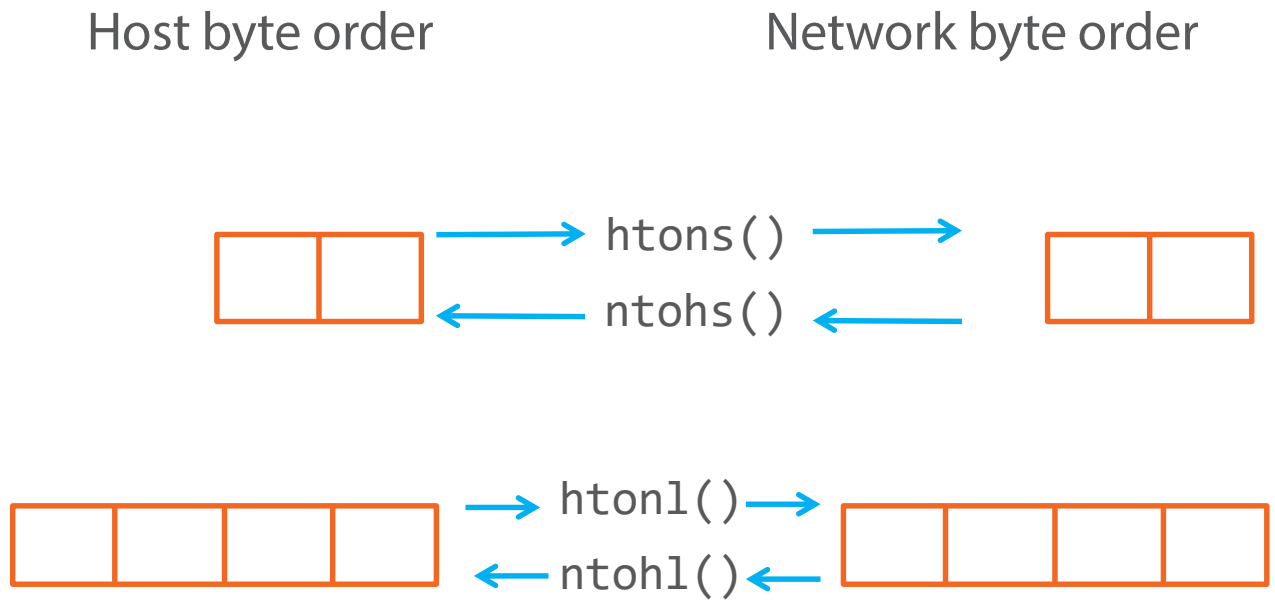
Network Byte Order

Socket addresses must be in network byte order (big-endian)



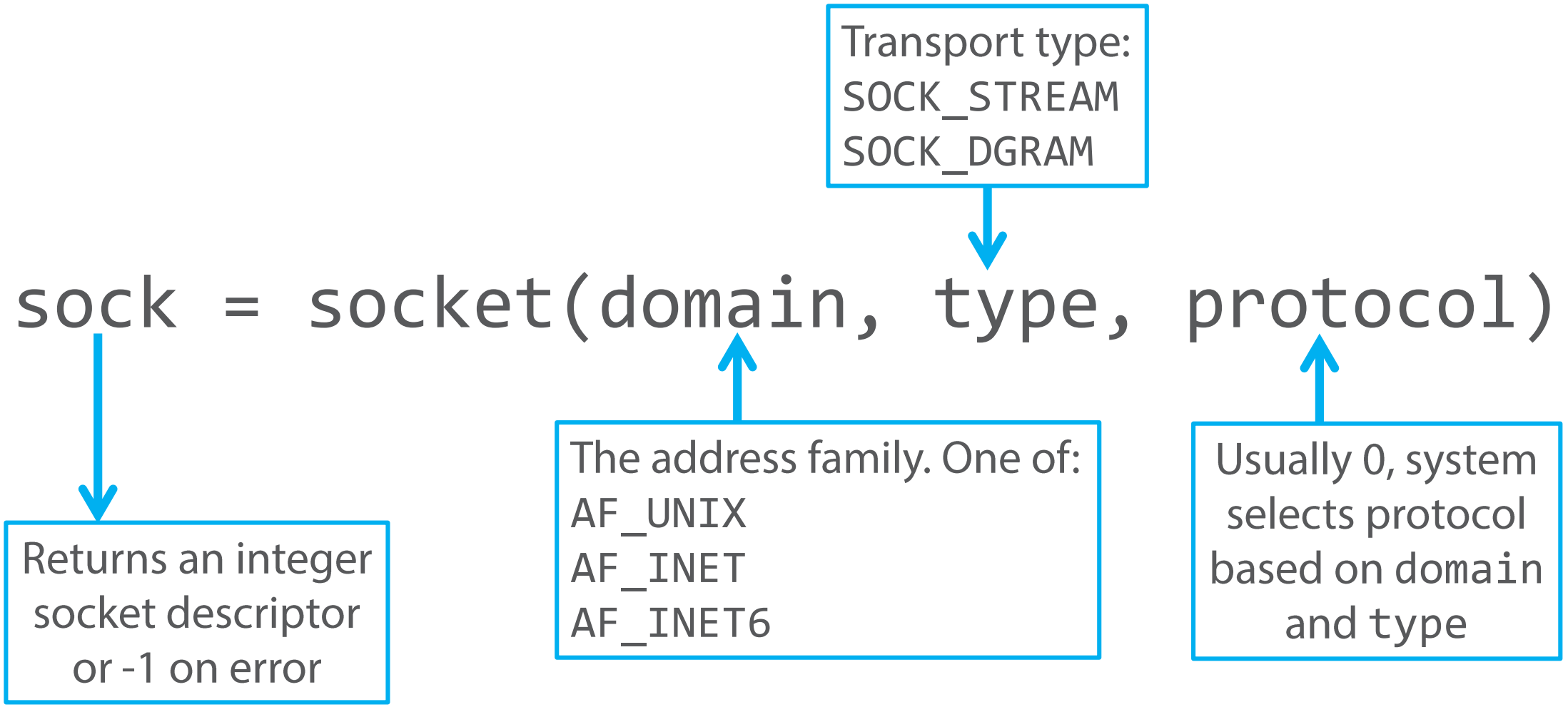
Most significant byte stored at lowest memory address, sent across network first

Macros convert to/from the machine's internal byte order.



pluralsight

Creating a Socket



pluralsight

Setting the Local Address

```
#define SERVER_PORT     1067
struct sockaddr_in      server;

server.sin_family      = AF_INET;
server.sin_addr.saddr = htonl(INADDR_ANY)
server.sin_port        = htons(SERVER_PORT);

bind(sock, (struct sockaddr *)&server, sizeof server);
```

pluralsight

Waiting for Business

```
struct sockaddr_in client;
int fd, client_len;

listen(sock, 5);

client_len = sizeof(client);
fd = accept(sock, (struct sockaddr *)&client, &client_len);
```

Set up a queue for pending connections

Connection descriptor

Rendezvous descriptor

Client's endpoint address returned here.

pluralsight

Talking to the Client

- The connection descriptor returned from `accept()` looks like an open file
  - Supports `read()` and `write()` system calls

```
unsigned char buf[1024];
int count;
while ((count = read(fd, buf, 1024)) > 0)
{
  rot13(buf, count);
  write(fd, buf, count);
}
```

pluralsight

Demonstration: The rot13 Server

pluralsight

Doing It in Python

- Python's `sockets` module exposes the traditional sockets API
- Python language features hide some of the messier stuff
  - Automatic buffer allocation on receiving
  - Implicit buffer length management on sending
  - Optional arguments to methods
  - Passing and returning tuples
  - Exception handling replaces error checking
- A socket is a *class*
  - Methods `bind()`, `listen()`, `accept()` ... expose the API

pluralsight

Doing It in Python


```
s = socket(AF_INET, SOCK_STREAM)
s.bind(('', 1068))
s.listen(5)

while True:
    client,addr = s.accept()
    rot13_service(client)
    client.close()
```

Empty address string means `INADDR_ANY`

Tuple assignment:
(connection descriptor, client endpoint address)

pluralsight

Demonstration: Python Server

pluralsight

Moving Forward …

In this module:

- Sequence of operations
- Key data structures
- Key system calls
- The rot13 server
- Doing it in Python

Coming up in the next module:

- The client side of TCP/IP

pluralsight

Writing TCP-based Clients

Chris Brown

pluralsight

In This Module …

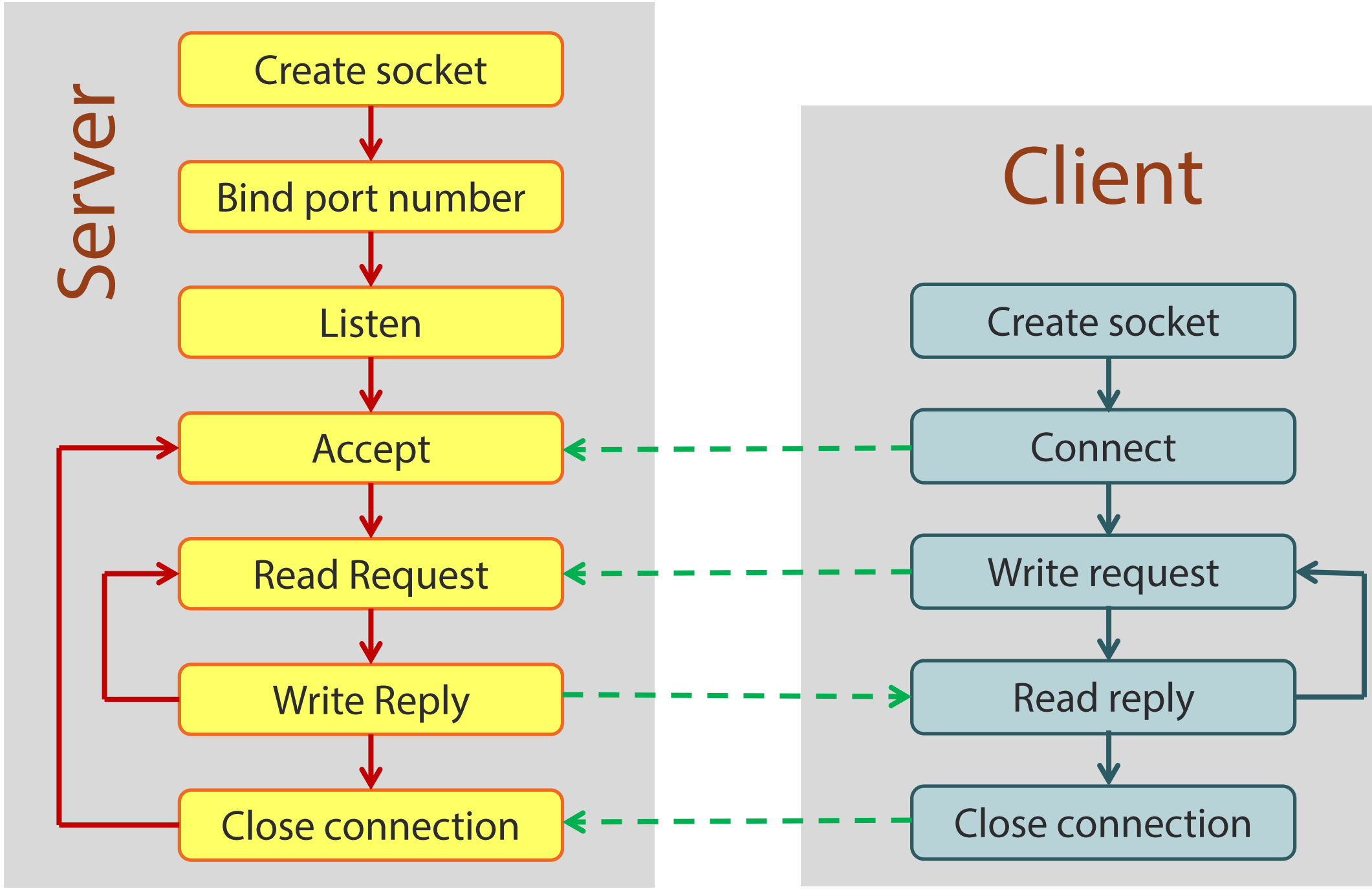
A TCP client for our rot13 server

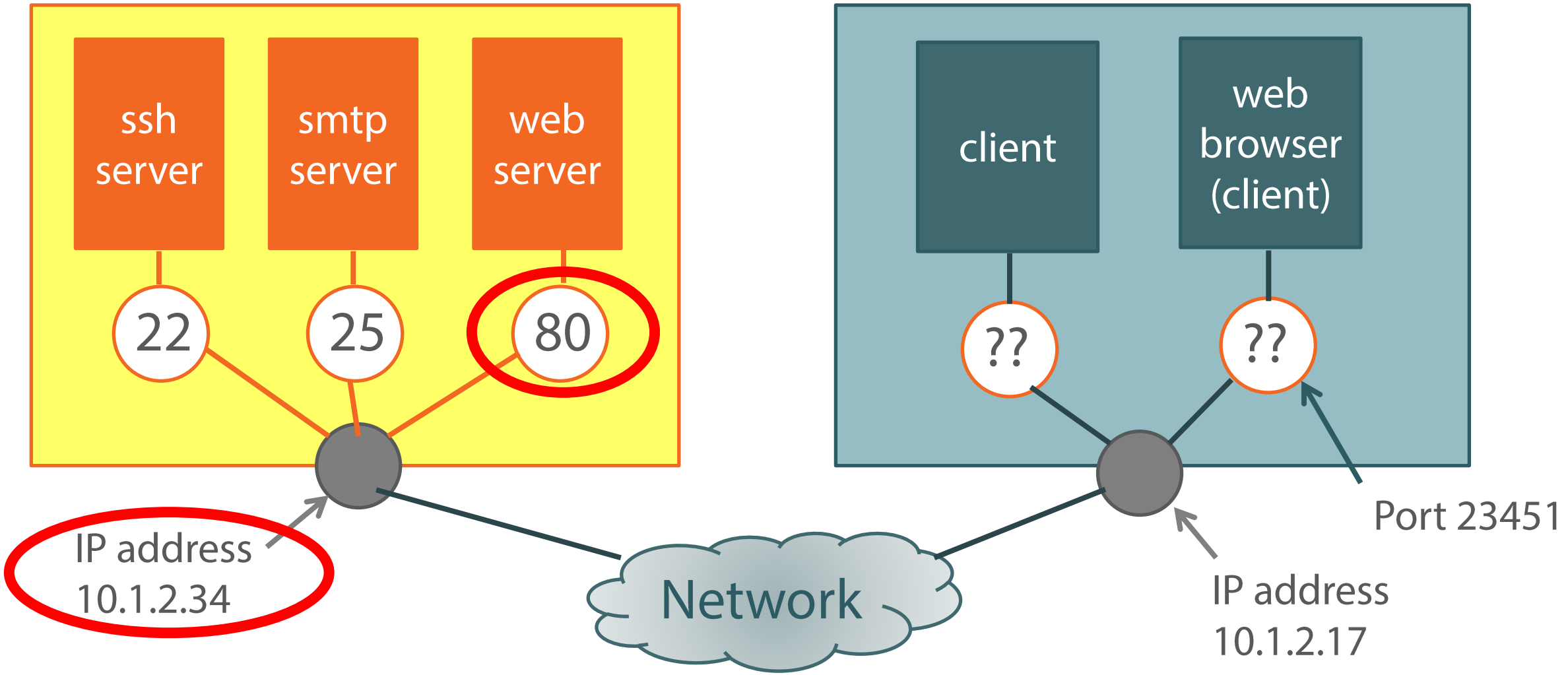
Finding the service (the old-fashioned way)

Finding the service (the new way)

Doing it in Python

pluralsight

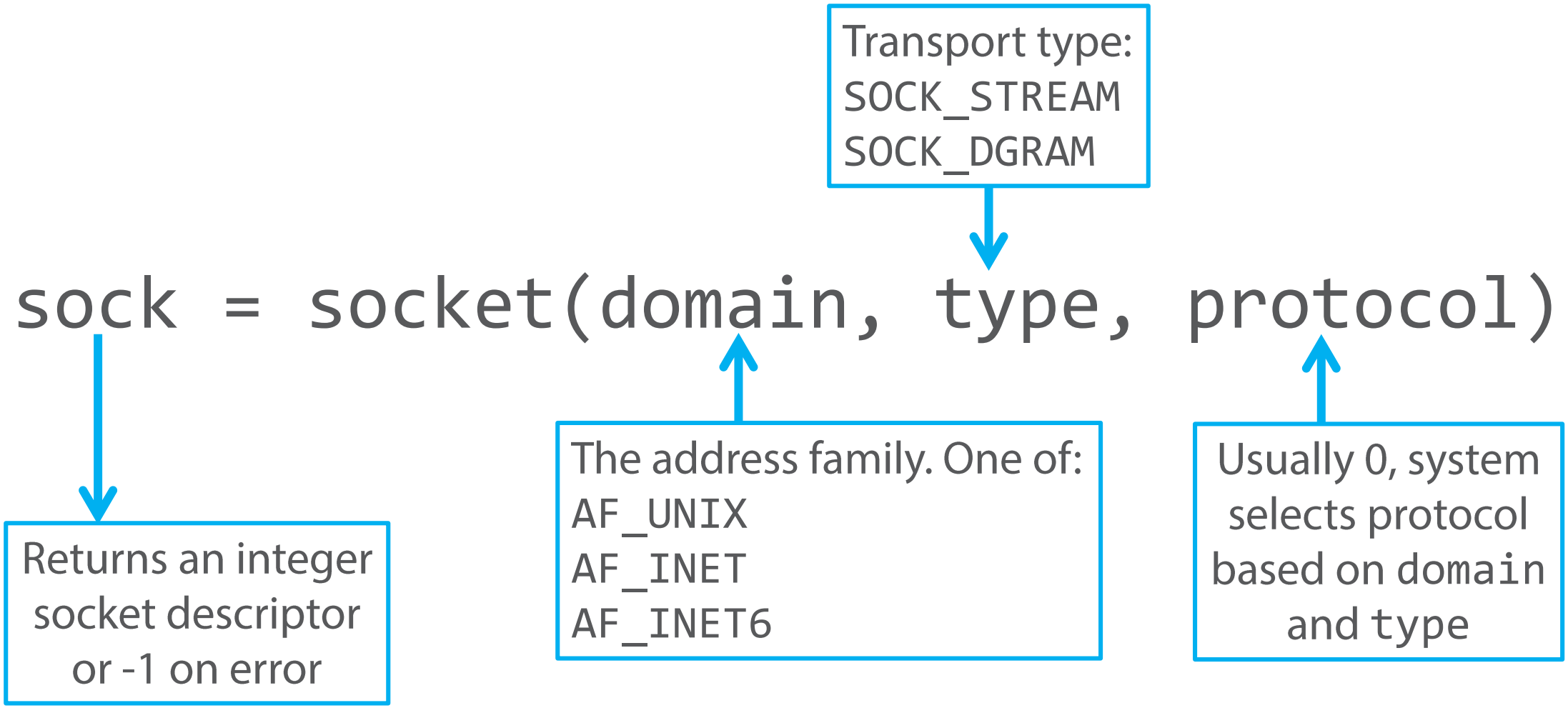

Server
Create socket
Bind port number
Listen
Accept
Read Request
Write Reply
Close connection
Client
Create socket
Connect
Write request
Read reply
Close connection



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

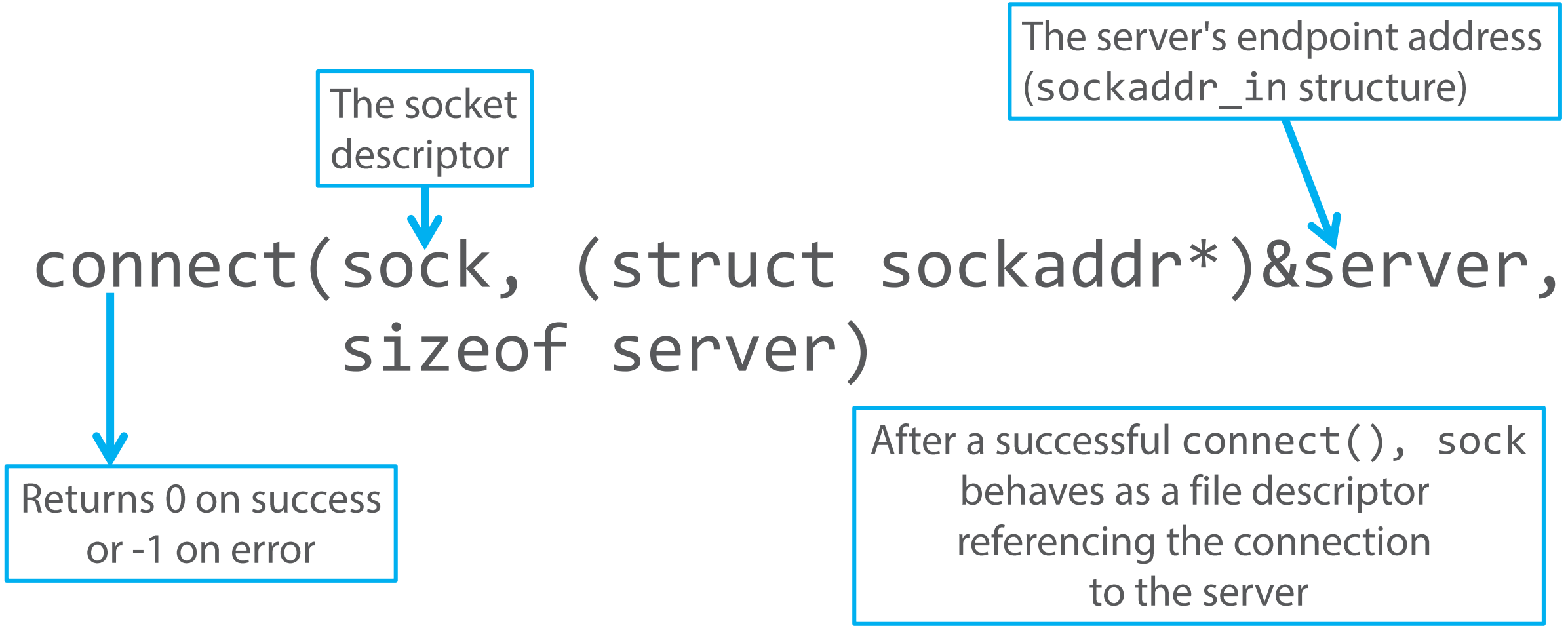
pluralsight

Creating a Socket

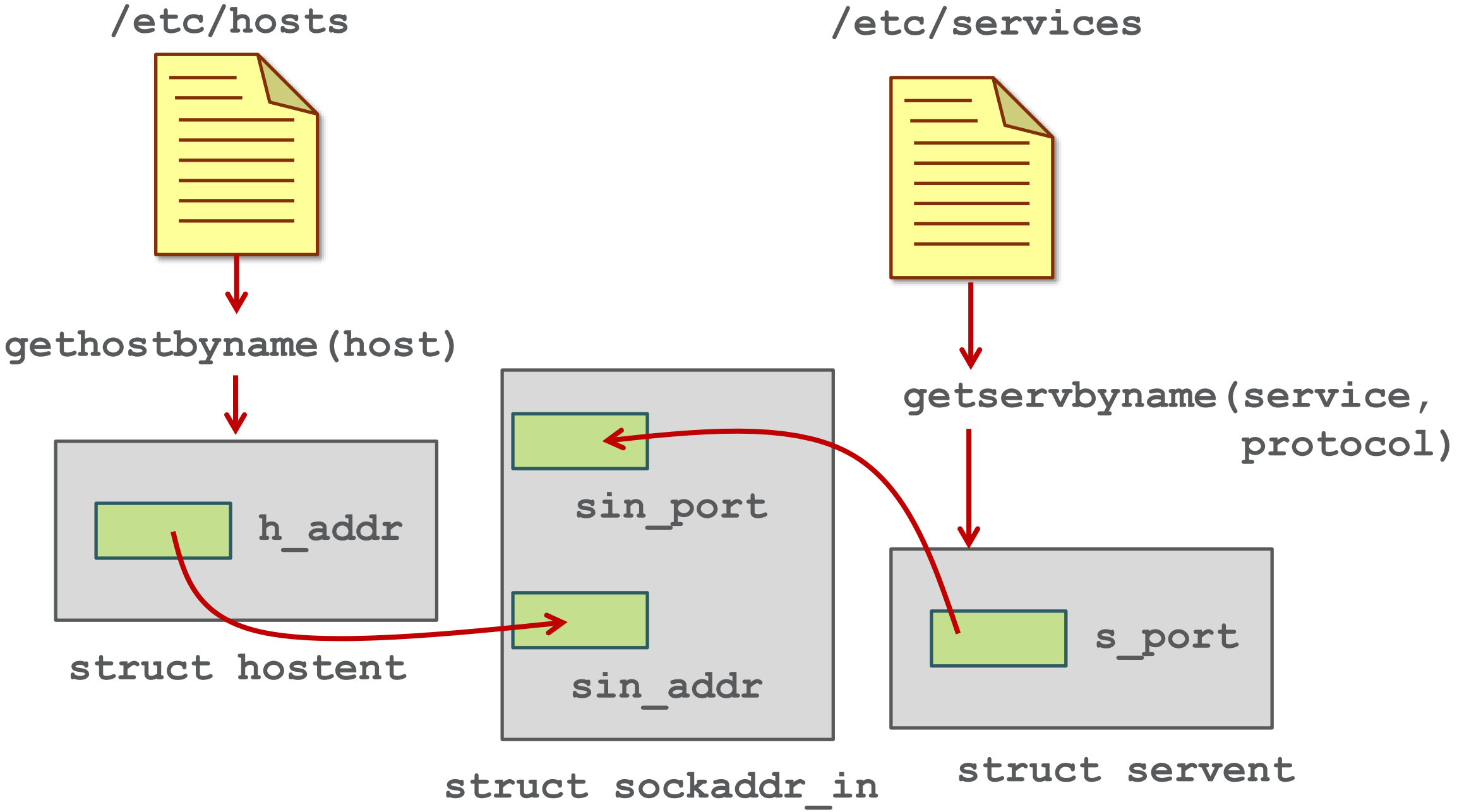


pluralsight

Making the Connection



pluralsight


/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

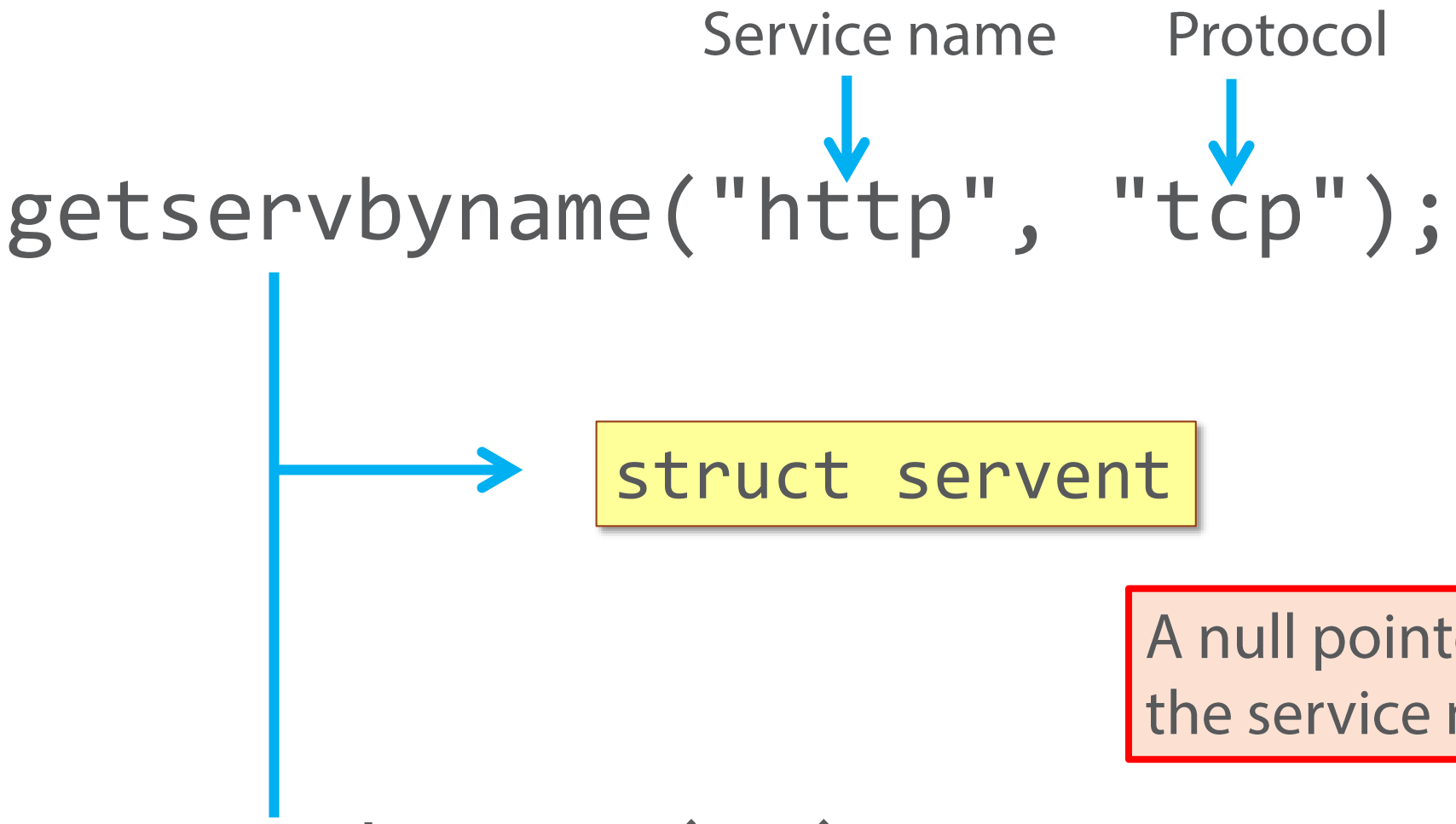
```
ftp-data      20/tcp
ftp           21/tcp
ssh           22/tcp
ssh           22/udp
telnet        23/tcp
smtp          25/tcp
domain        53/tcp
domain        53/udp
tftp          69/udp
http          80/tcp
```

Maps service names to port number and protocol

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

pluralsight

Finding the Service



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

pluralsight

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 */
}
```

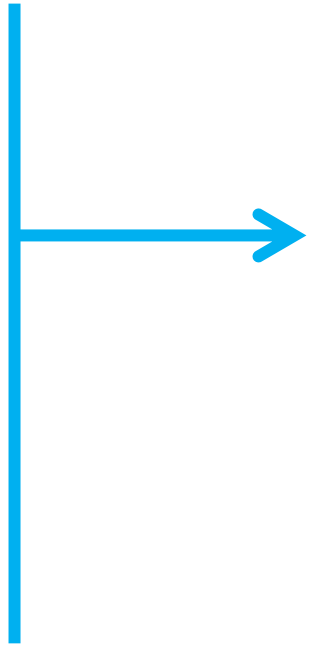
In network byte order

pluralsight

Finding the Host

Host name

```
gethostbyname("venus.example.com");
```



`struct hostent`

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

```
gethostbyname("192.168.1.44");
```

pluralsight

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

pluralsight

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)
```

pluralsight

Demonstration

Old-fashioned TCP client

pluralsight

Protocol Independence

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

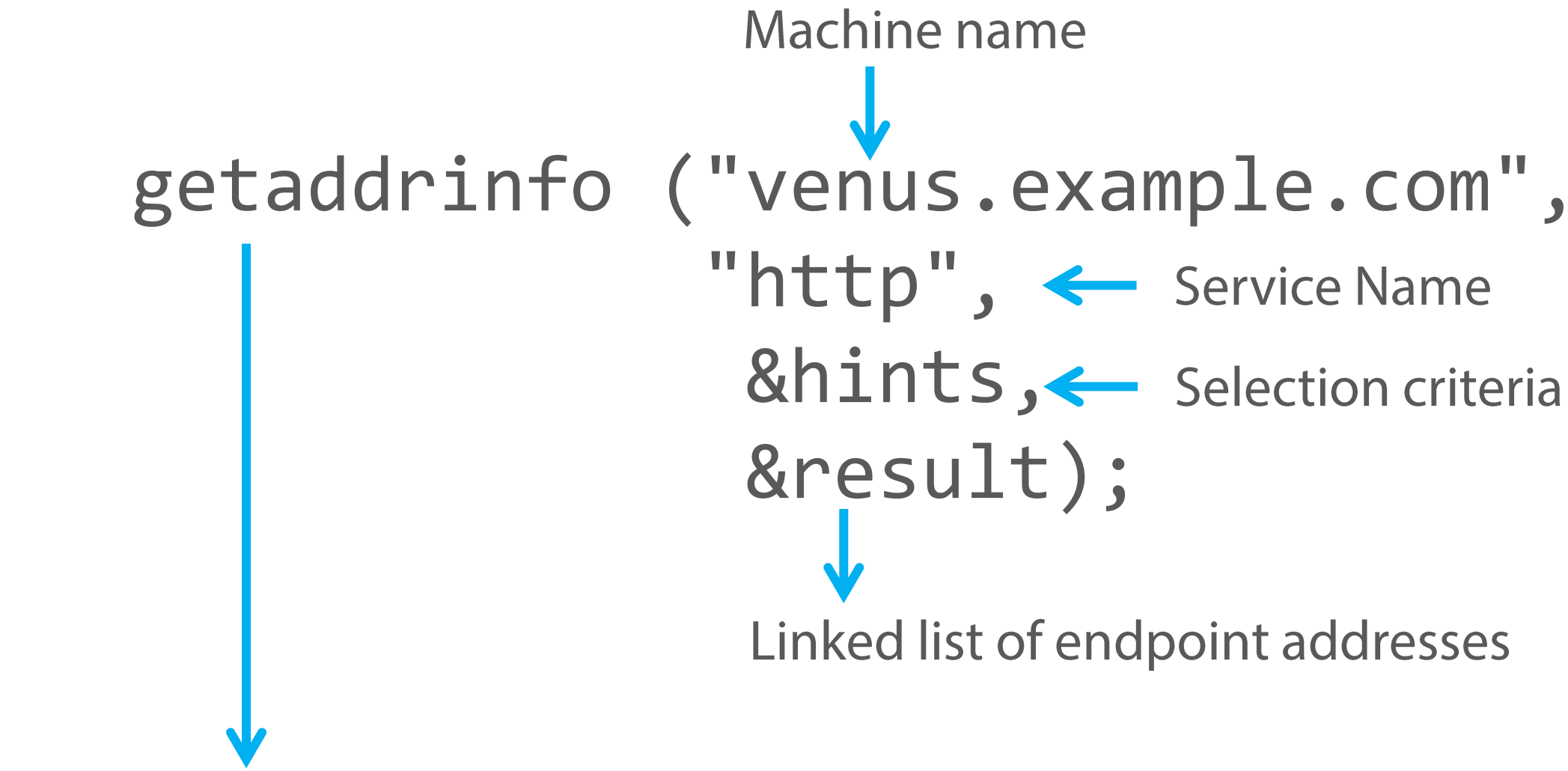
Growing need for IPv4 / IPv6 operability

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

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

pluralsight

Finding the Service (the Modern Way)



pluralsight

Providing Hints

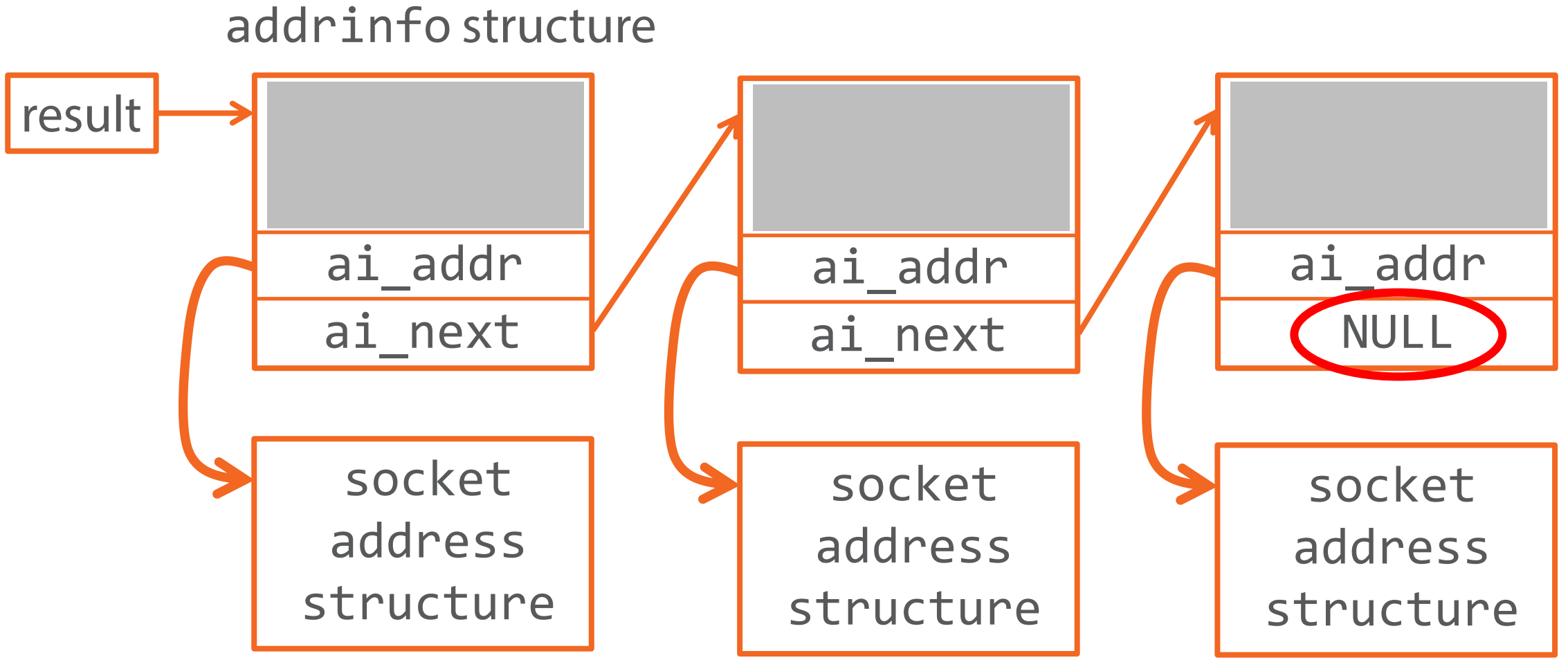
The `hints` argument of `getaddrinfo()` lets you constrain the types of endpoint addresses you want.

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
```

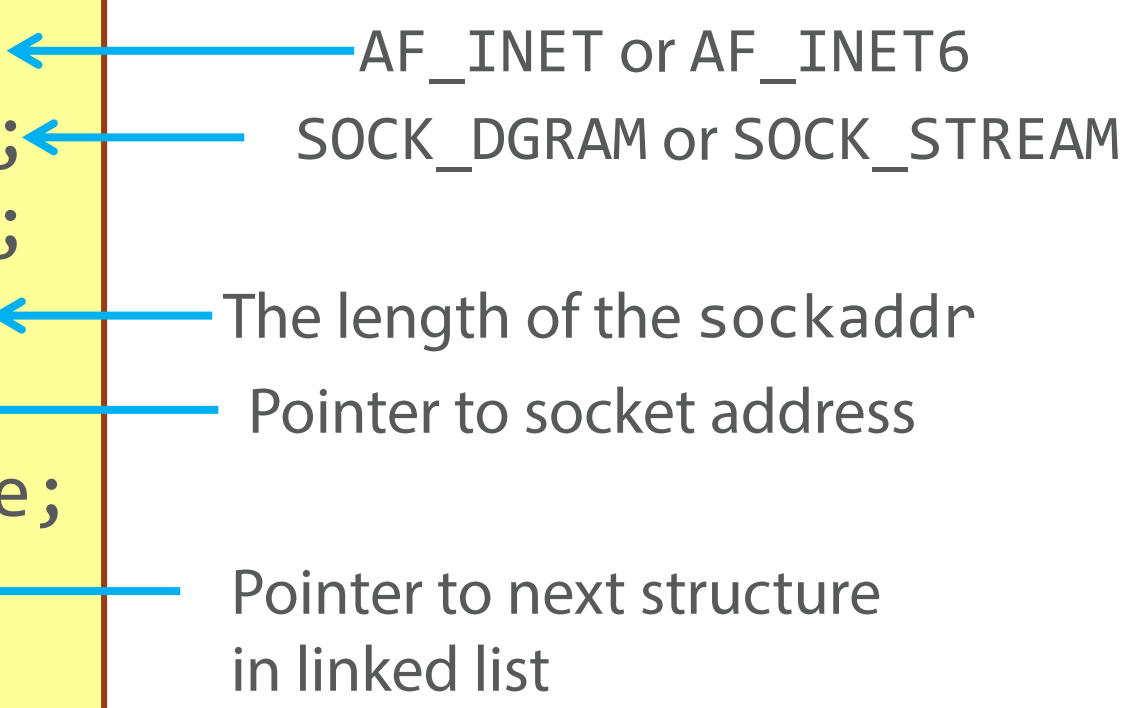
pluralsight

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;
};
```



pluralsight

Demonstration

Modern TCP client

pluralsight

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

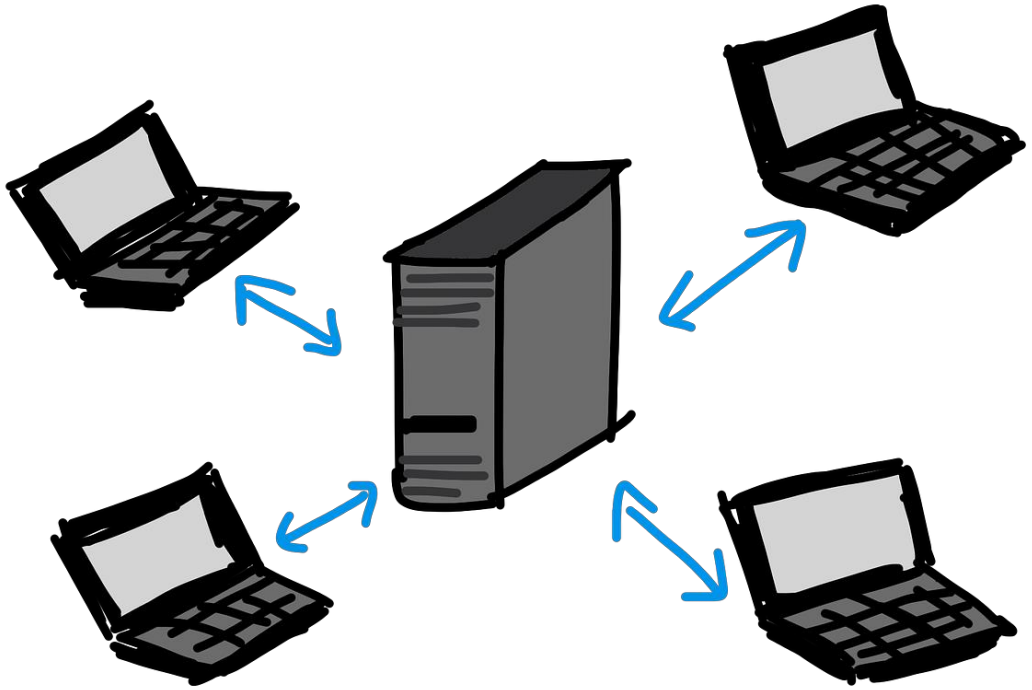
pluralsight

Demonstration

Python TCP client

pluralsight

Module Summary



Finding the service

Finding the host

Connecting to the server

Protocol Independence

Doing it in Python

pluralsight

Moving Forward …

Coming up in the next module:

- UDP clients and servers
- TFTP client
- UDP broadcasting

pluralsight

Writing UDP-based Servers and Clients

Chris Brown

pluralsight

In This Module ...

Client and server-side operations using UDP

UDP sockets API

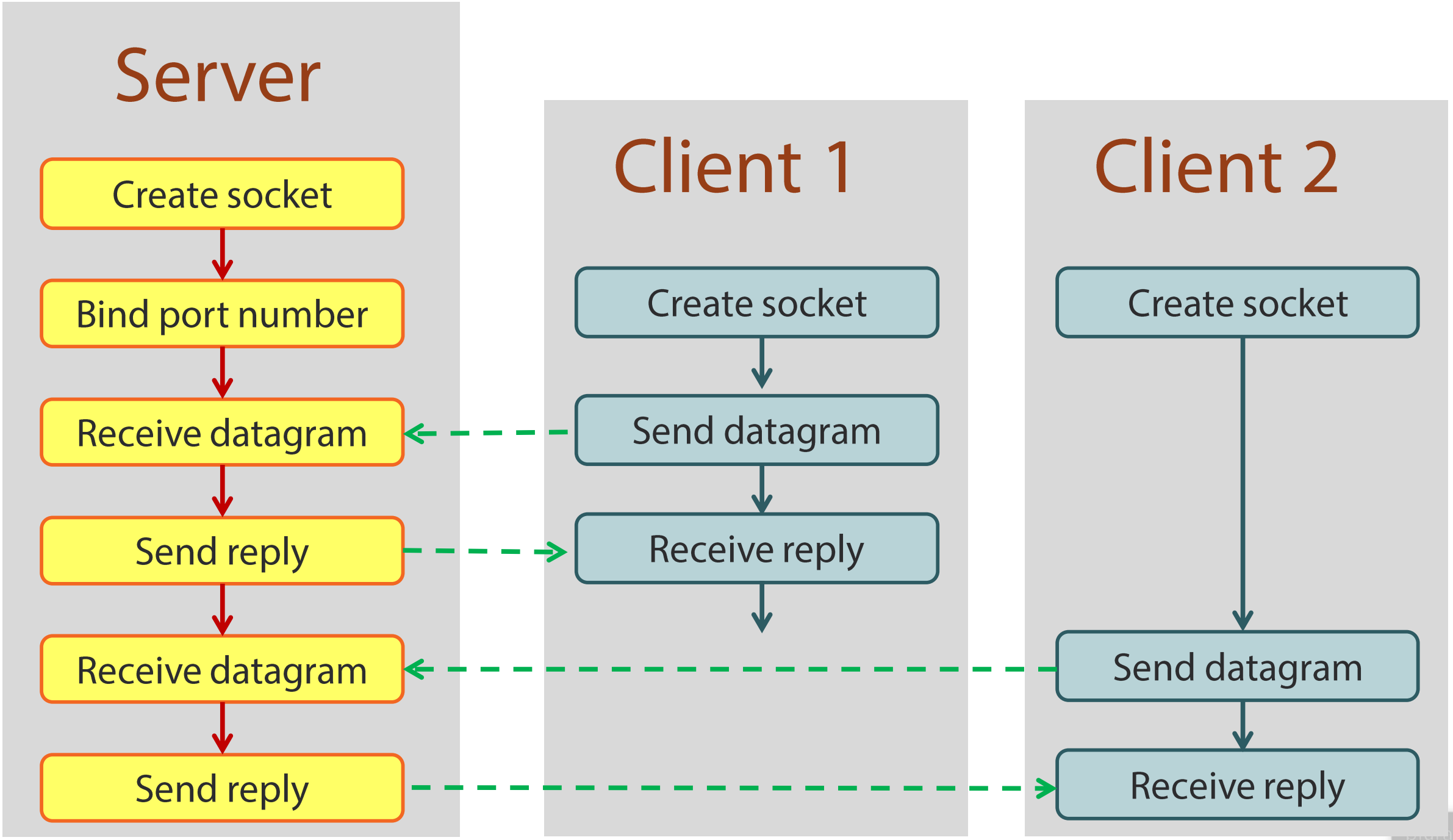
A binary application protocol (TFTP)

Demonstration: rcat – a TFTP client

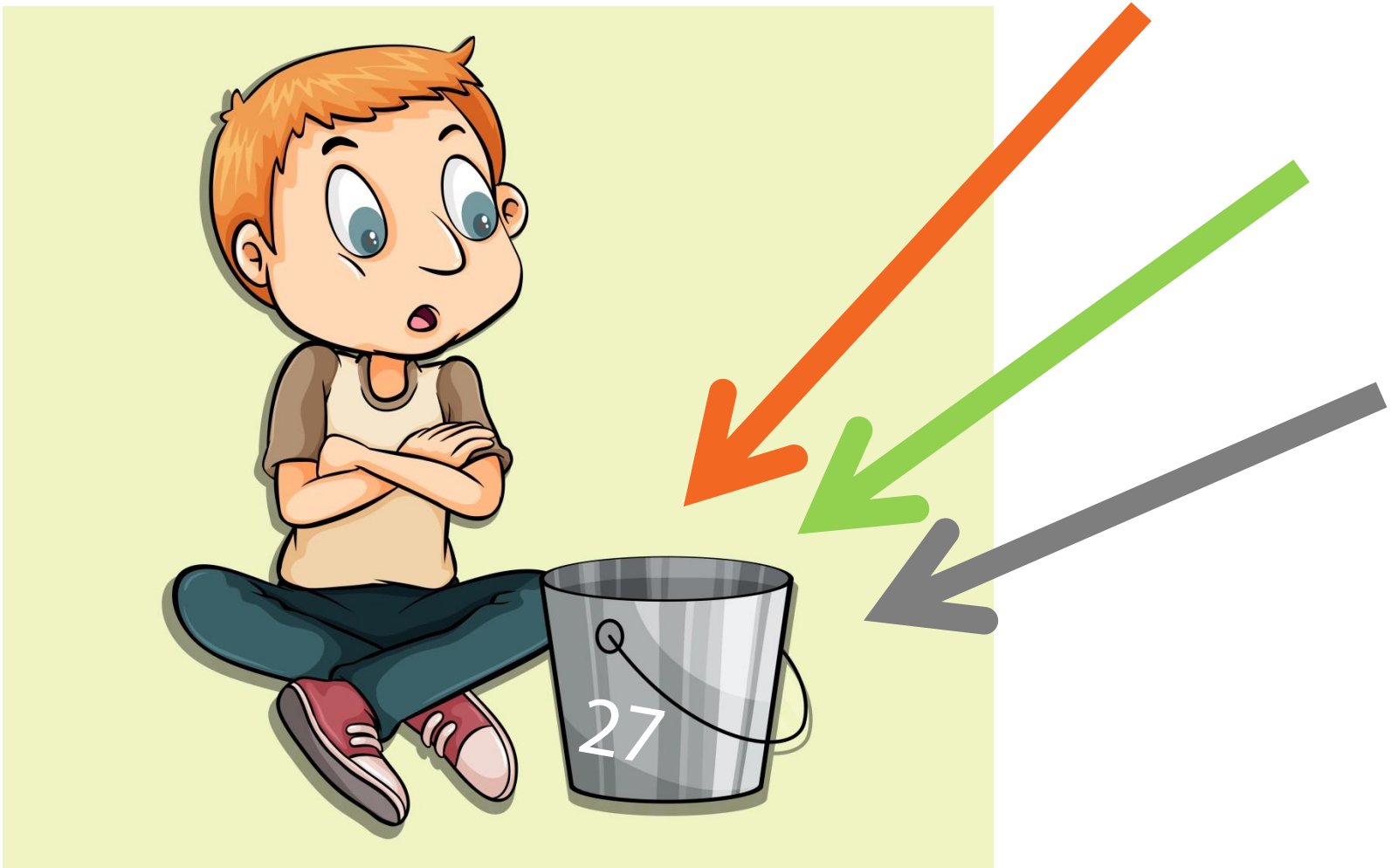
Broadcasting with UDP

Demonstration: A UDP broadcast application

pluralsight


Server
Create socket
Bind port number
Receive datagram
Send reply
Receive datagram
Send reply
Client 1
Create socket
Send datagram
Receive reply
Client 2
Create socket
Send datagram
Receive reply

The Bucket Analogy

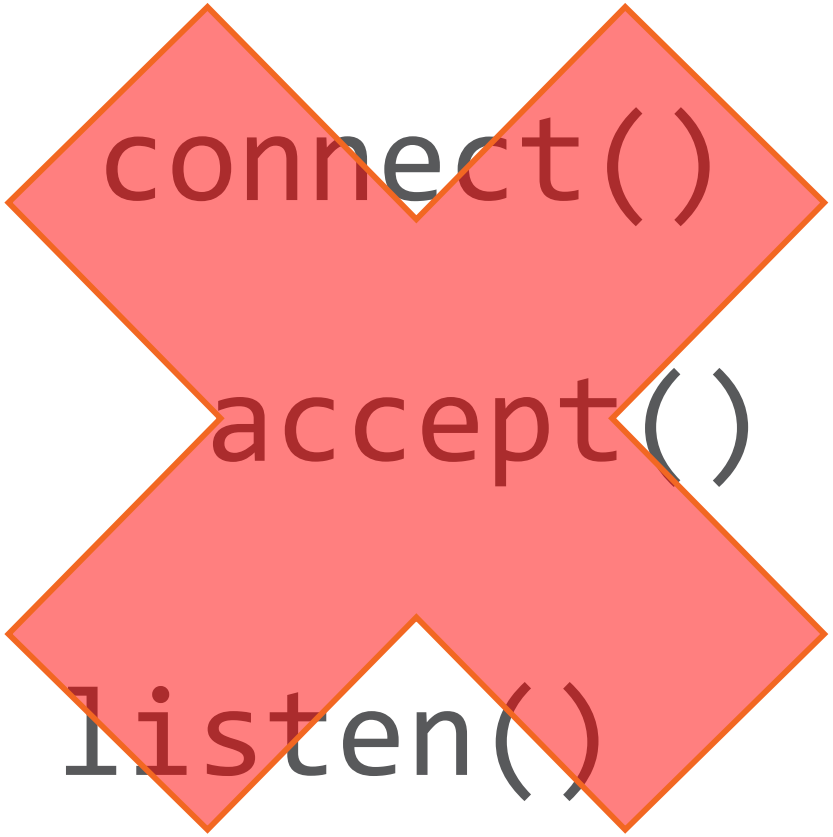


pluralsight

The Postbox Analogy

Each message is individually addressed

pluralsight



struct sockaddr_in{..}

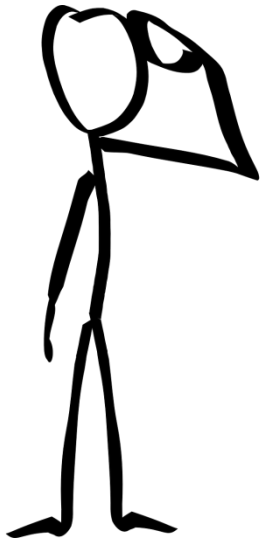
pluralsight

Coping with Unreliability

UDP is "unreliable"
— packets may get dropped
— or may be mis-ordered

Is the underlying network "reliable enough"?

?



Can we tolerate occasional dropped packets?
— Catastrophic failure of application?
— Gradual degradation of performance?

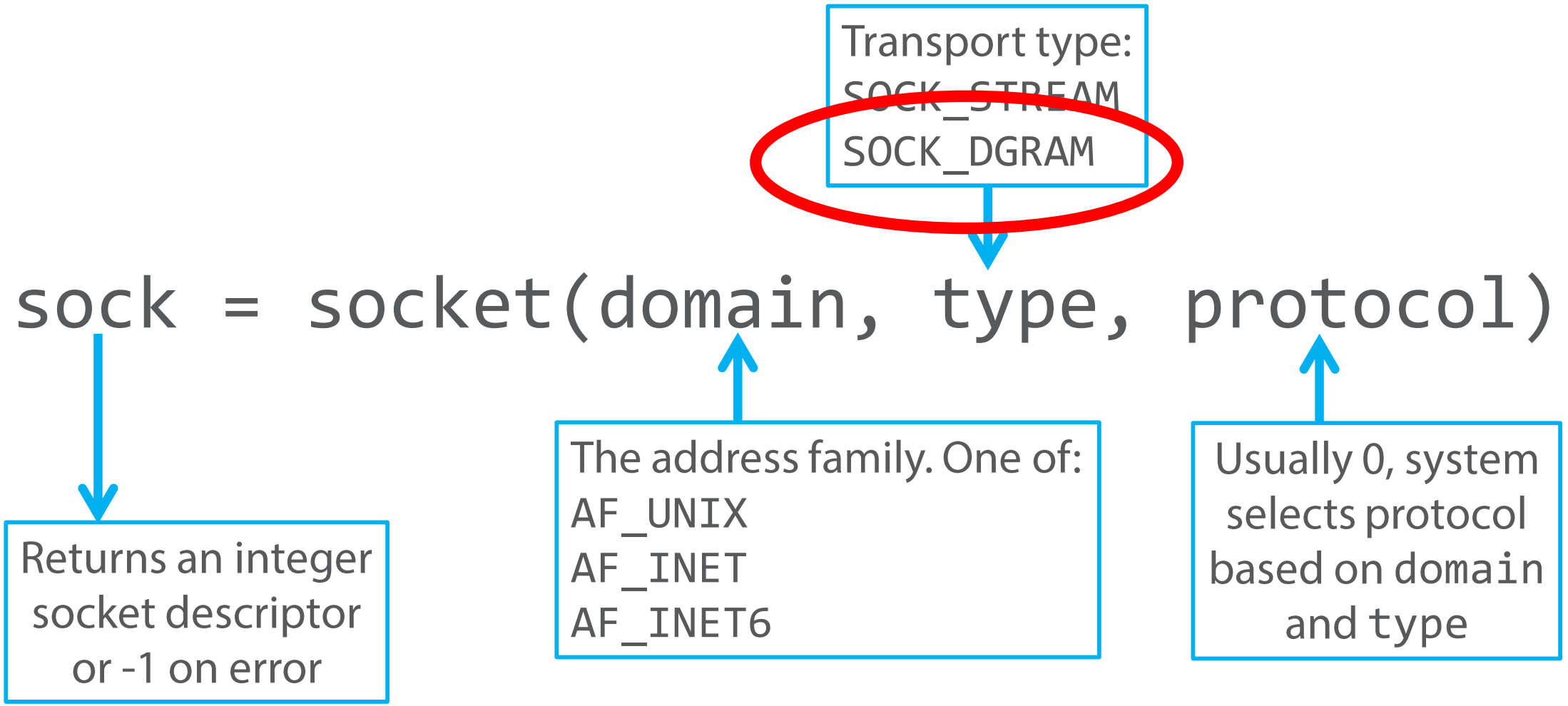
?

Handle acknowledgements, timeouts and re-transmissions within the application protocol

?

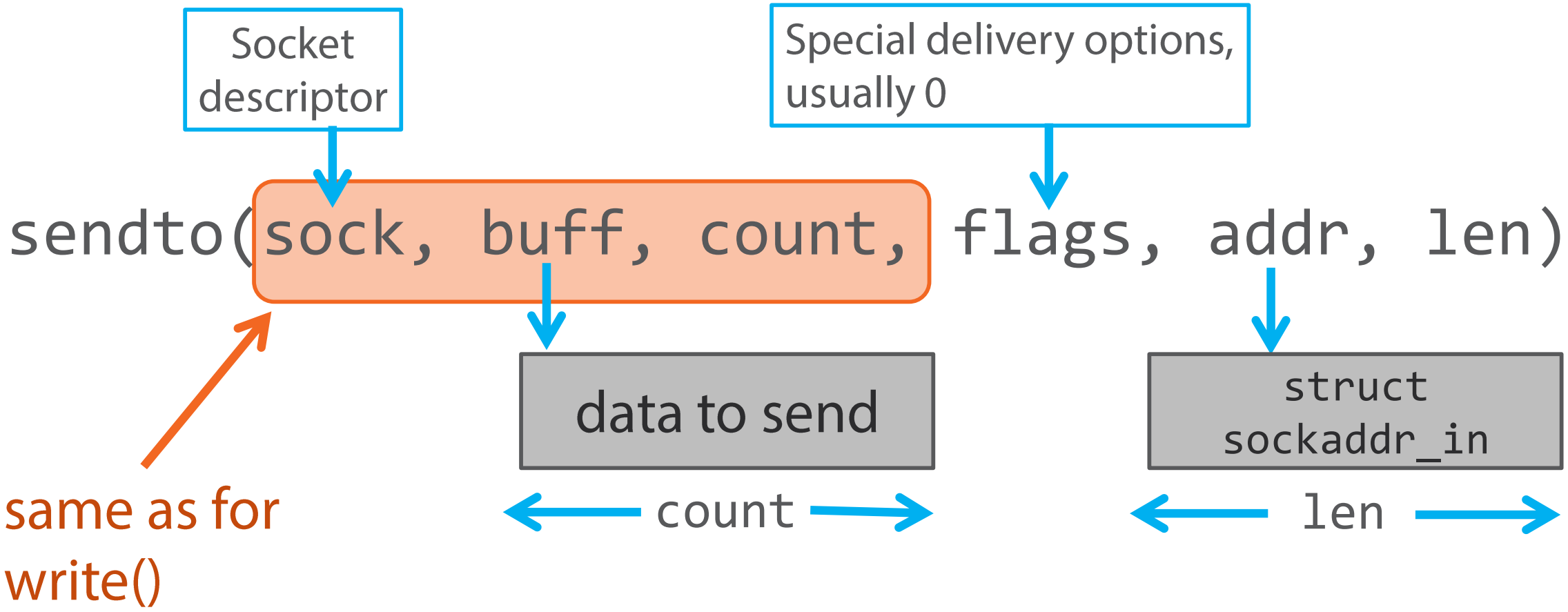
pluralsight

Creating a Socket



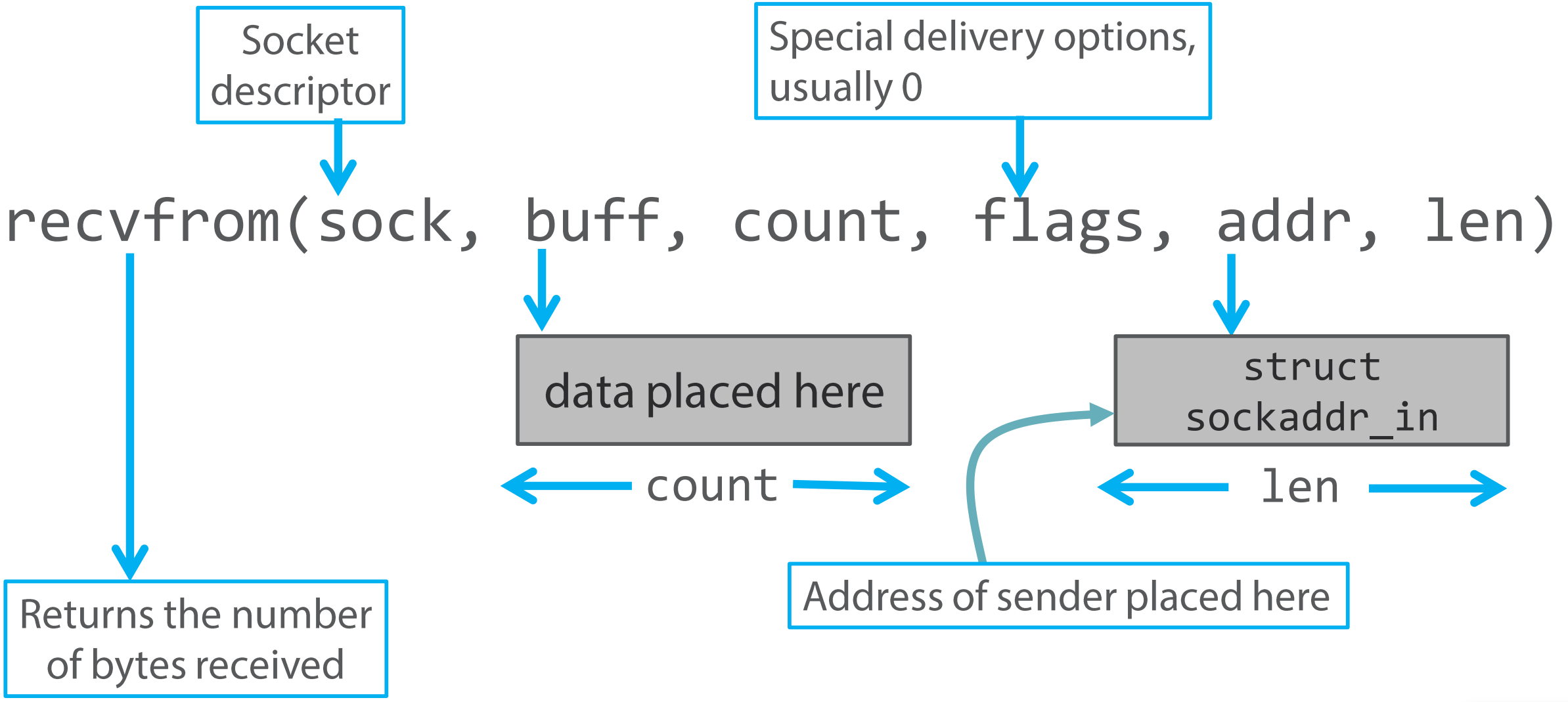
pluralsight

Sending a Datagram



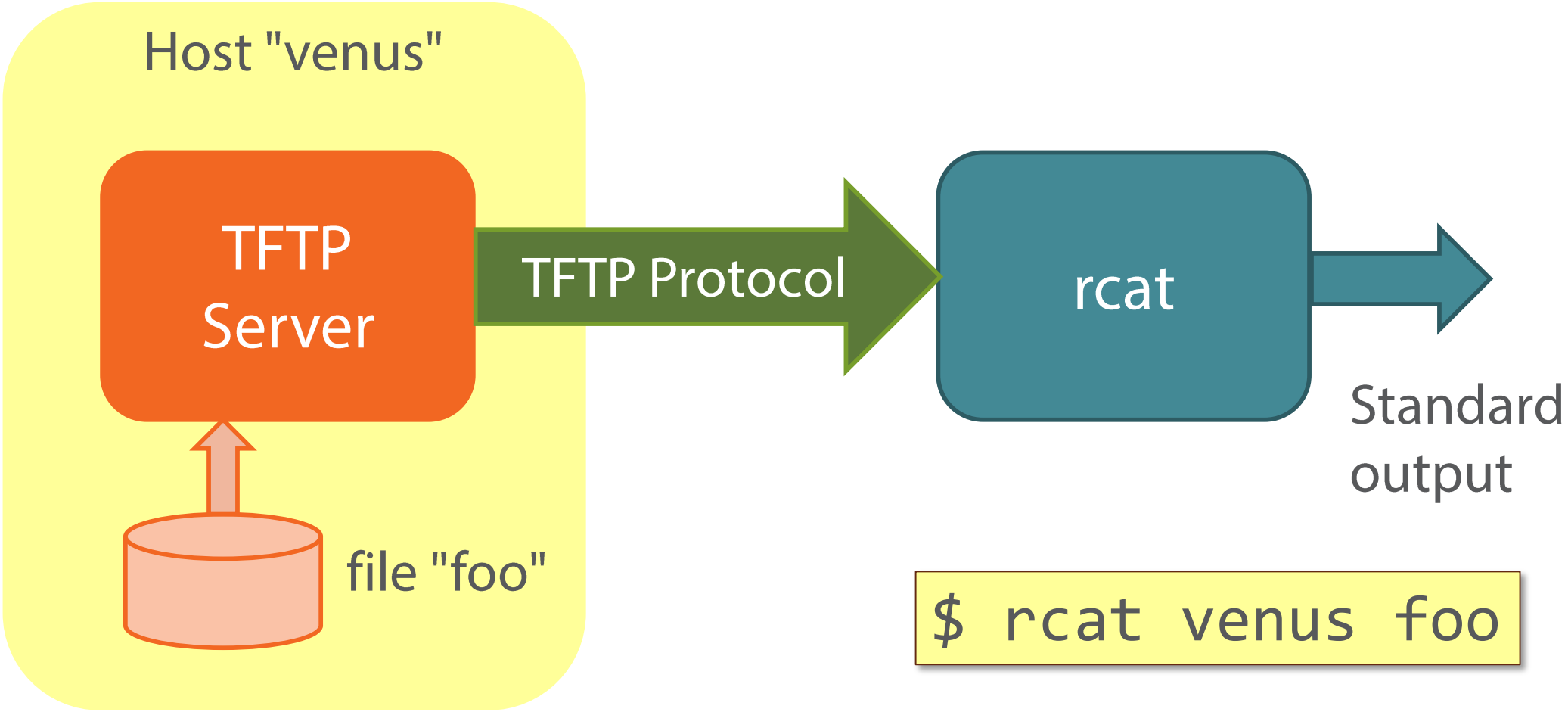
pluralsight

Receiving a Datagram



pluralsight

Our UDP Demonstration Client



pluralsight

Protocols

protocol – a set of rules governing the exchange or transmission of data between devices

Text-based:

- Based on text strings (human readable)
- HTTP, Telnet, SMTP, ...

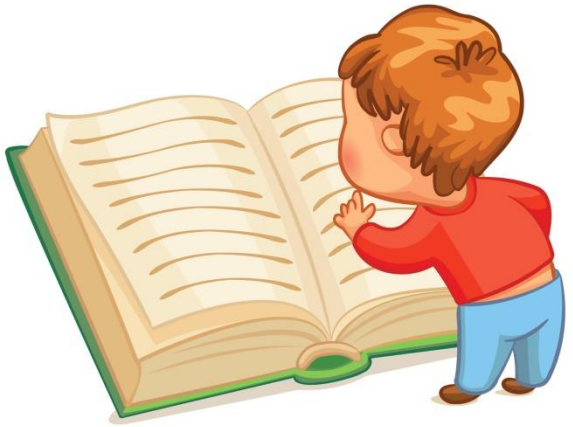
Binary:

- Based on data structures (machine-readable)
- IP, TCP, UDP, TFTP, ...

pluralsight

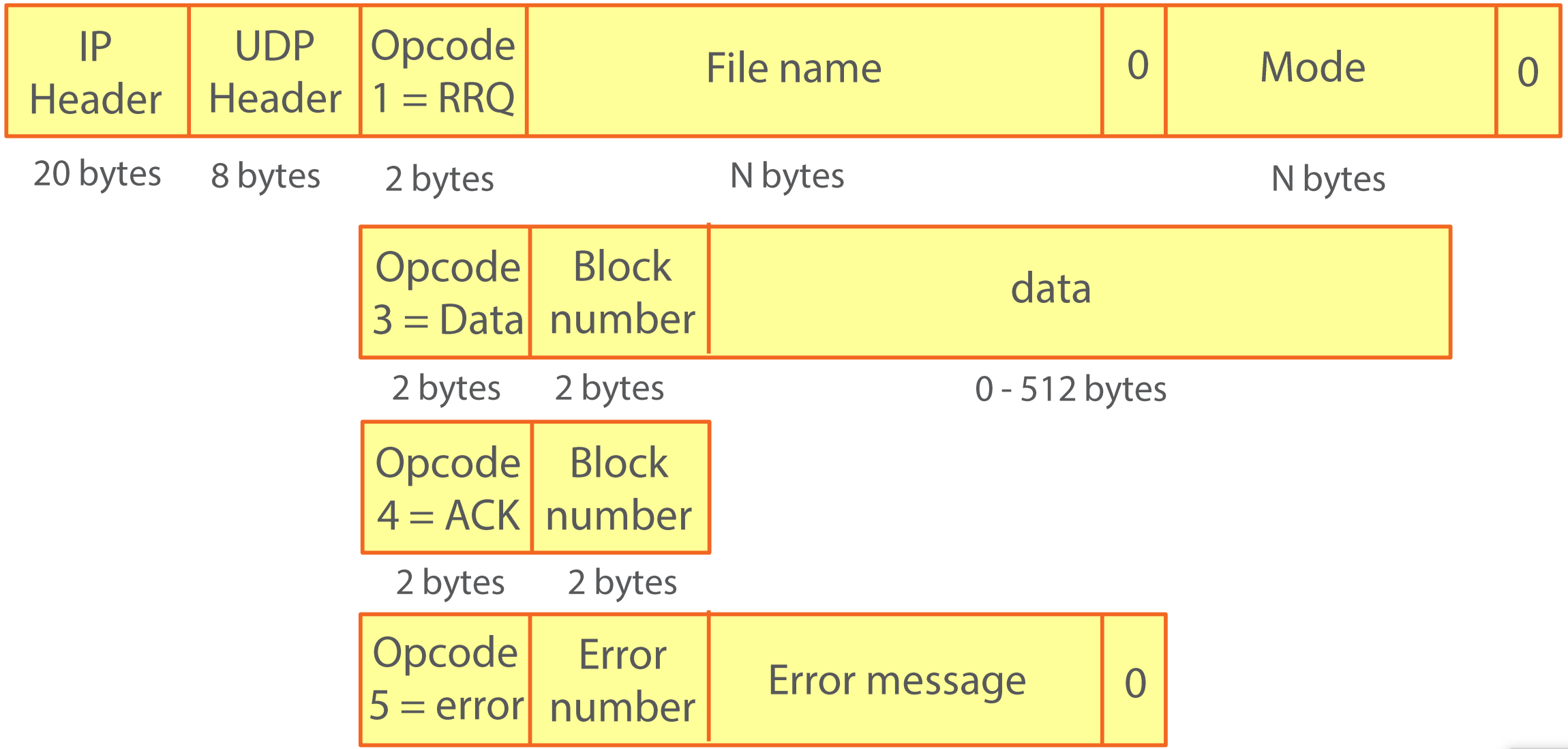
Introducing TFTP

- Trivial File Transfer Protocol
- Simple design
  - Small code footprint
- Useful to download kernel image, etc. to small systems
- No authentication or access control
  - Server often confined to `/tftpboot` directory
- Consult RFC1350 for the full story



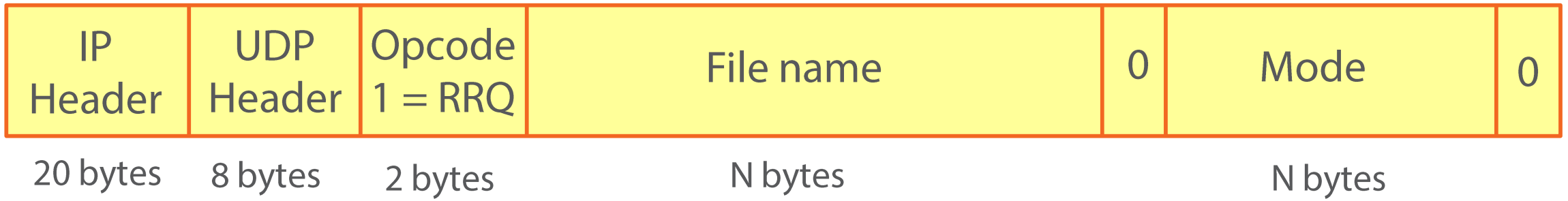
pluralsight

TFTP Packet Formats



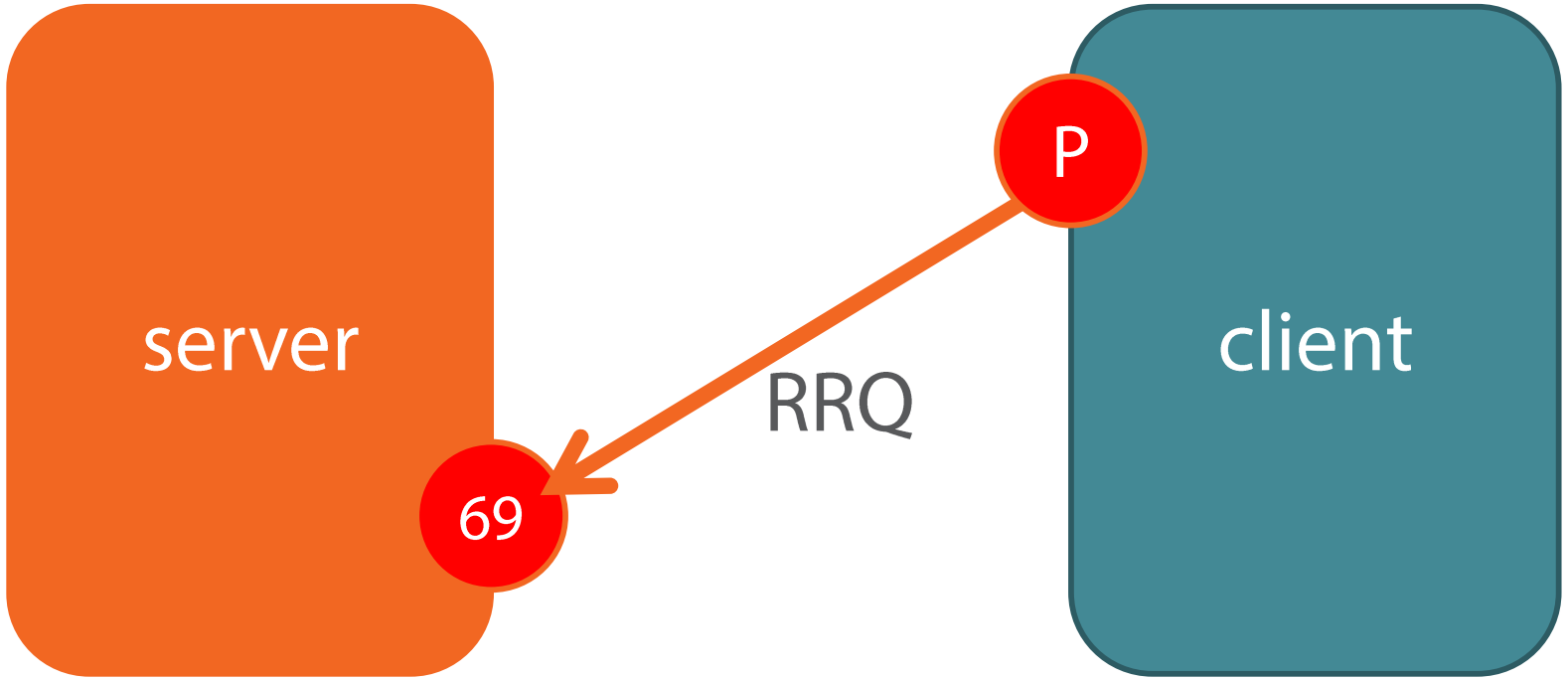
pluralsight

TFTP Packet Formats



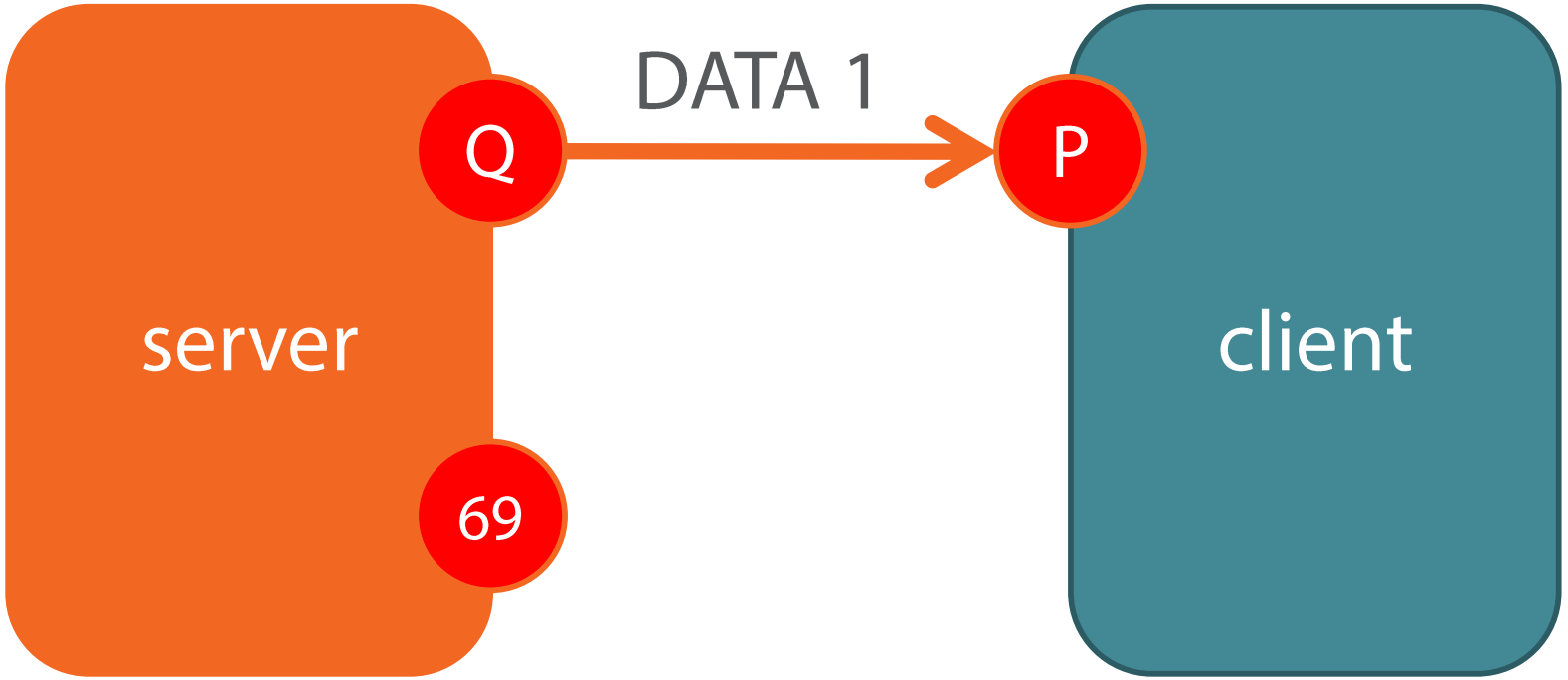
pluralsight

TFTP Operation



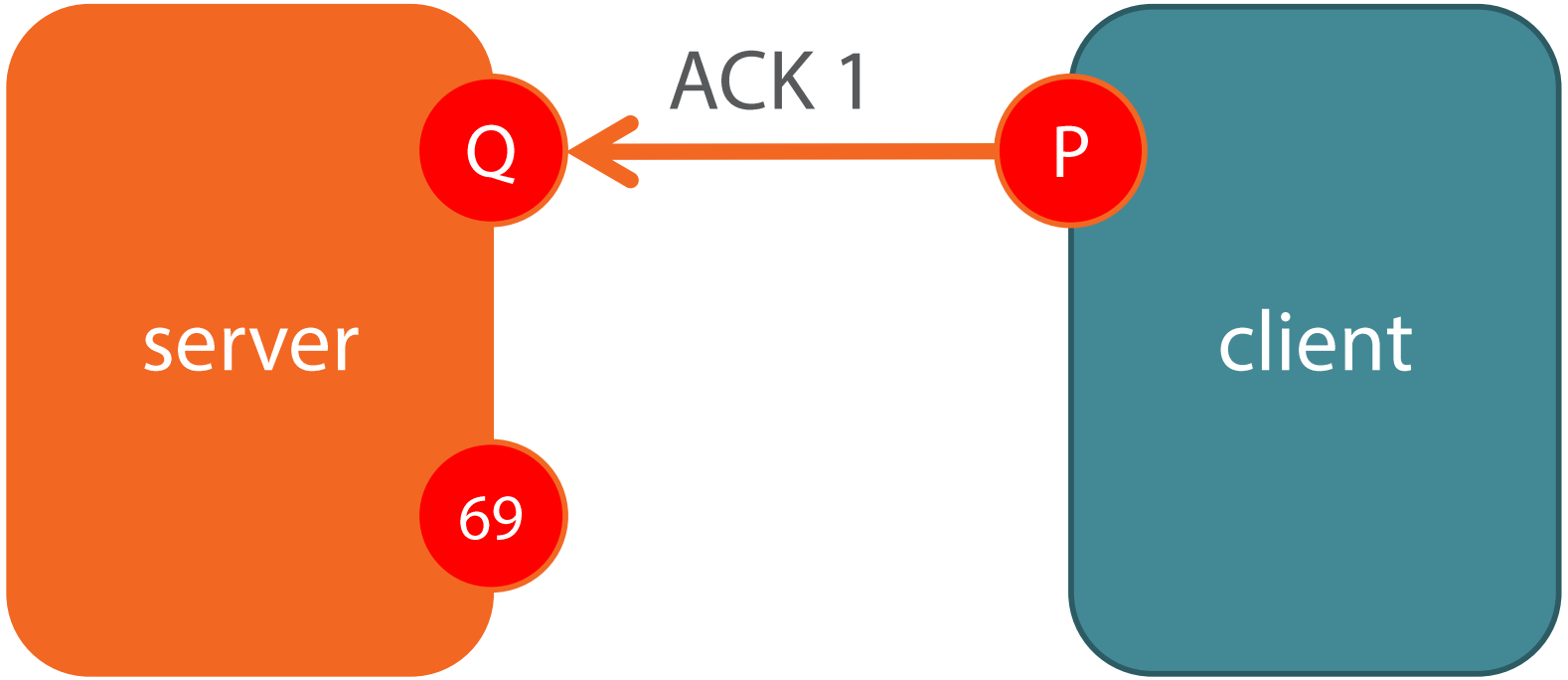
pluralsight

TFTP Operation



pluralsight

TFTP Operation



pluralsight

Demonstration

`rcat` – a TFTP client

UDP Packet Trace

pluralsight

UDP Broadcasting

UDP supports broadcasting

Datagram sent using IP host ID of "all ones"

All recipients must listen on same port

Packet size limited by MTU of underlying network

Does not work through routers

pluralsight

Broadcasting – Core Code

```
int sock, yes = TRUE;
struct sockaddr_in bcast;
char data[100];

sock = socket(AF_INET, SOCK_DGRAM, 0);
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &yes, sizeof yes);

bcast.sin_family = AF_INET;
bcast.sin_addr.s_addr = 0xffffffff;
bcast.sin_port = htons(MY_PORT);

sendto(sock, &data, sizeof data, 0, &bcast, sizeof bcast);
```

pluralsight

A Distributed Update Service

Multiple Participants

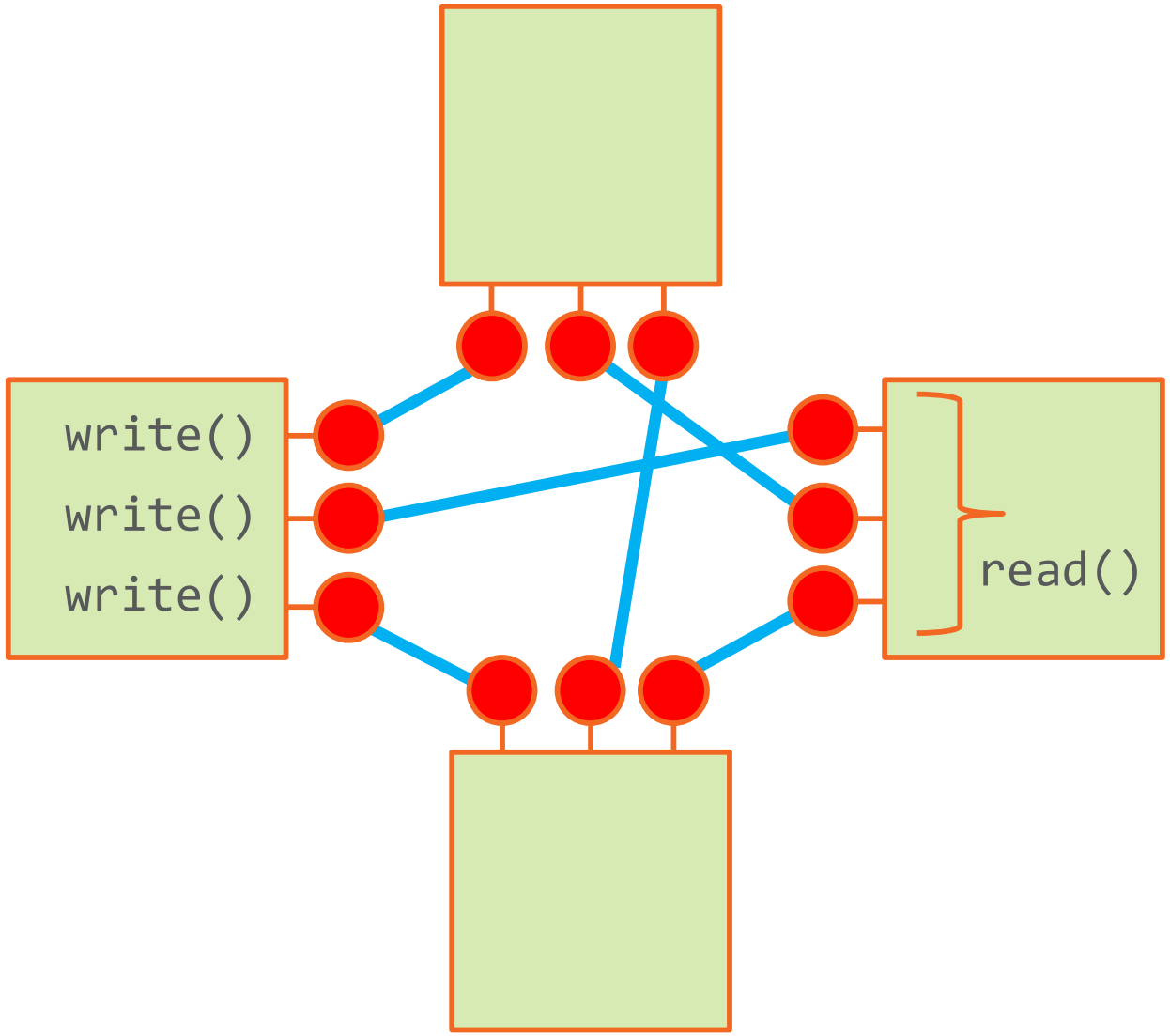
Each generates status information periodically

Each needs to receive status updates from all the others

Each displays the overall status

pluralsight

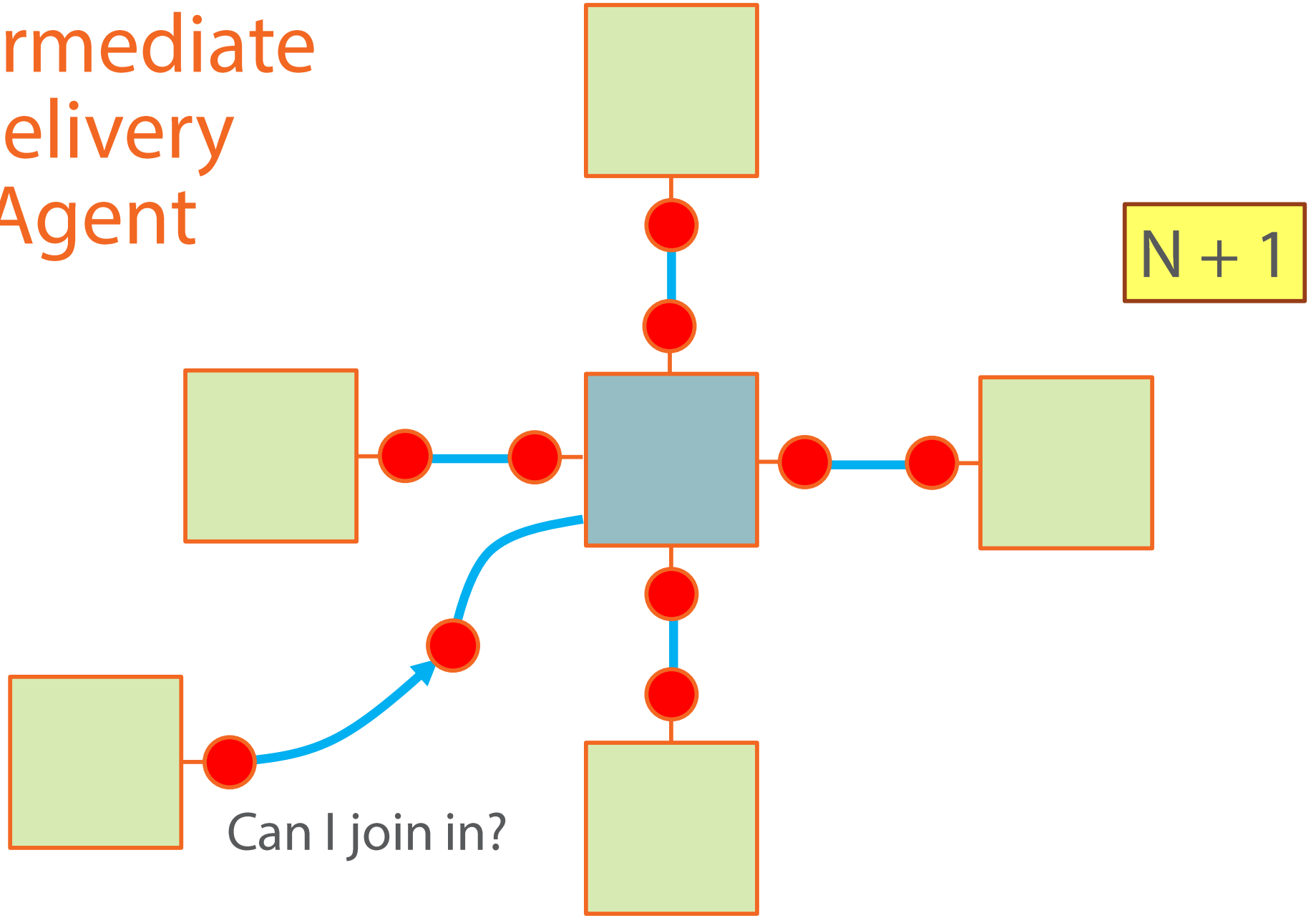
Fully Connected Model



N * (N – 1)

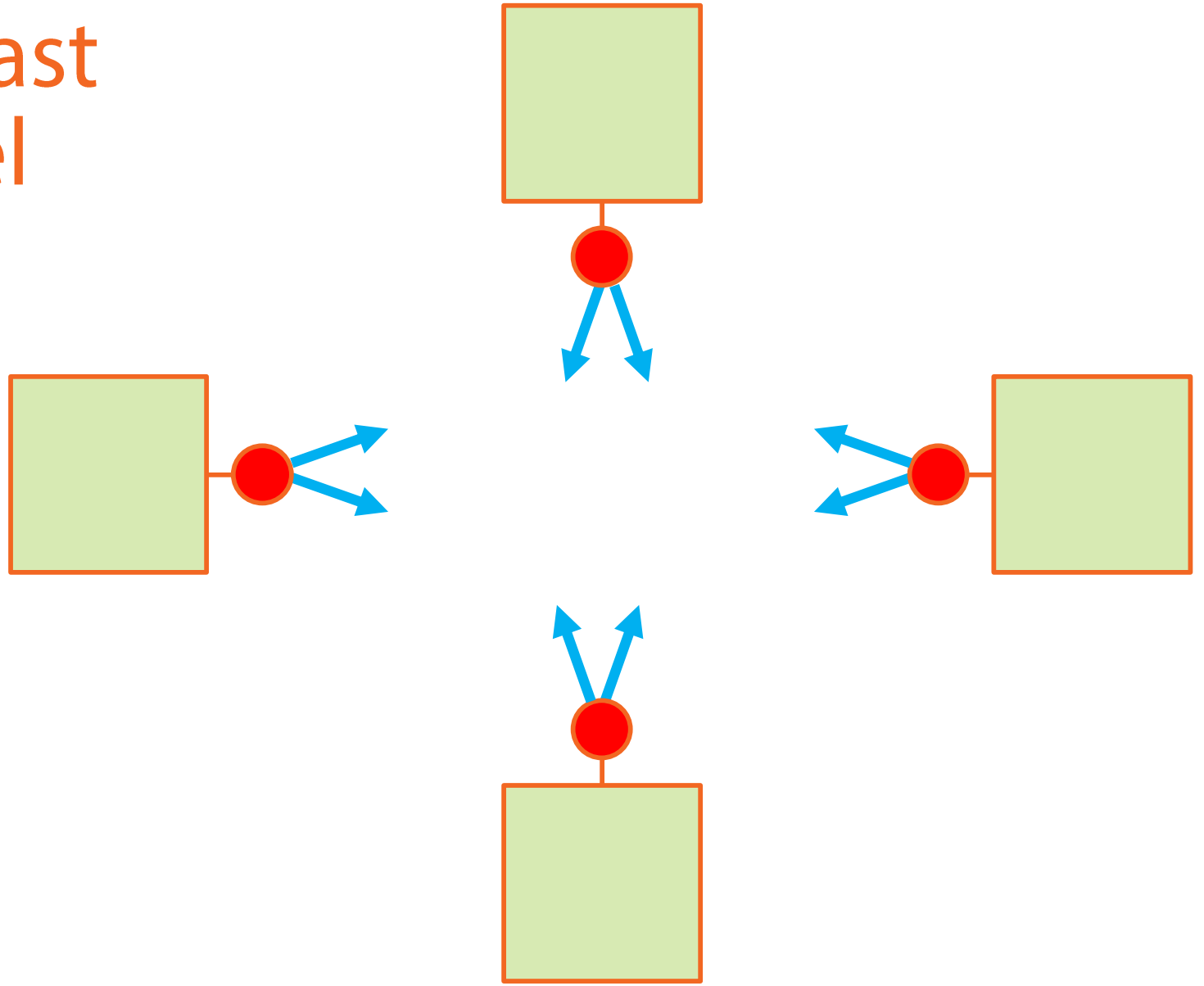
pluralsight

Intermediate Delivery Agent



pluralsight

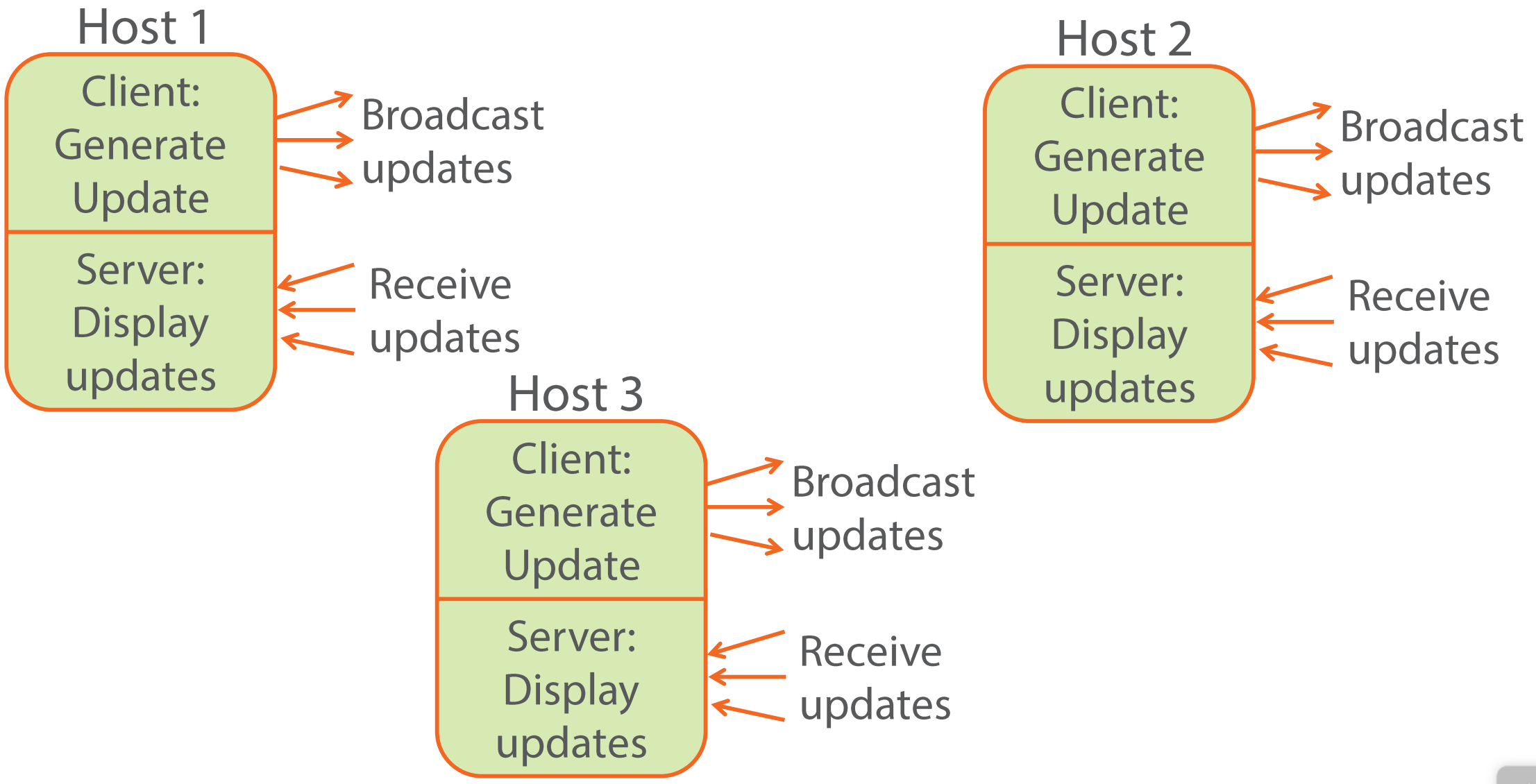
Broadcast Model



N

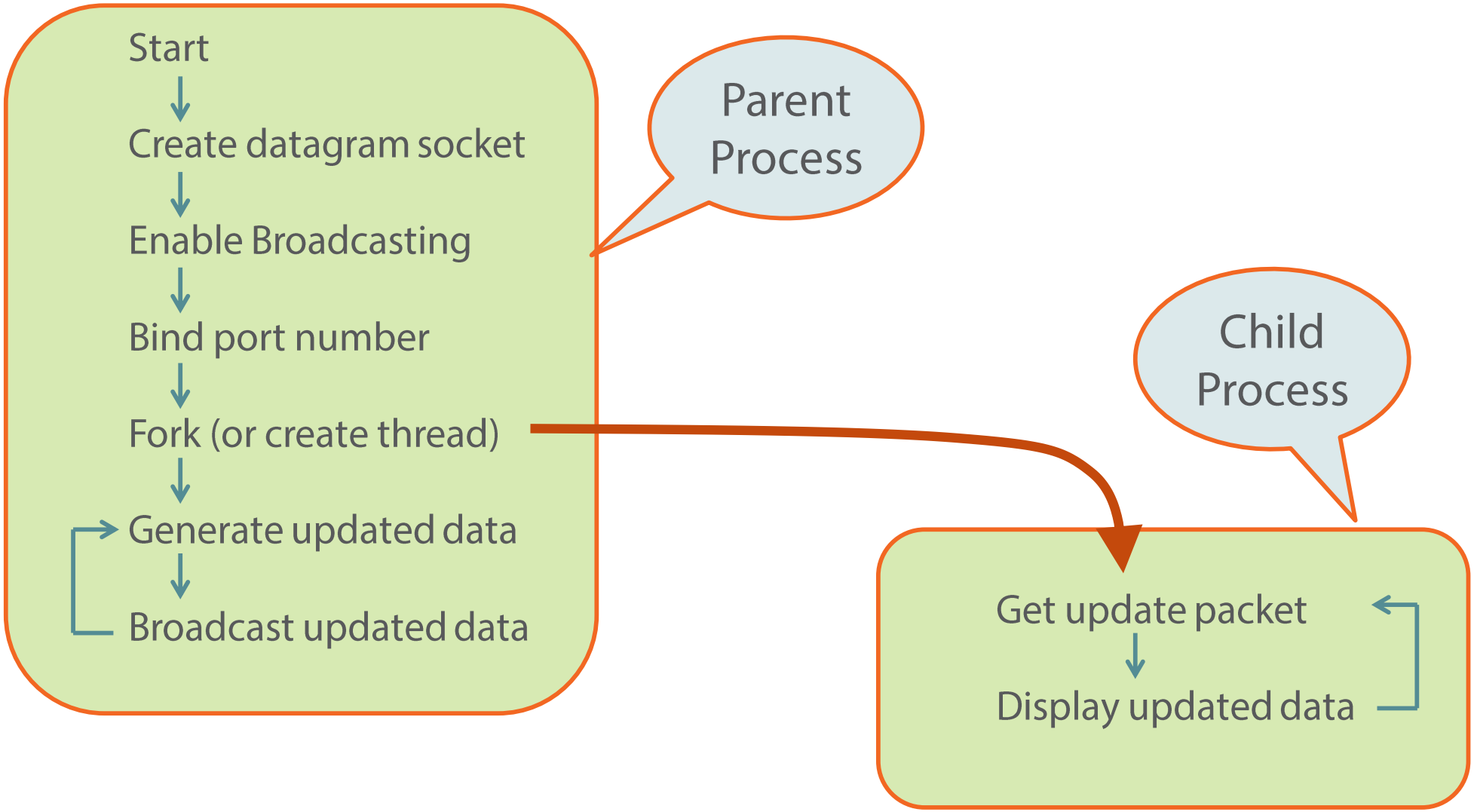
pluralsight

Peer-to-Peer Operation



pluralsight

Program Structure



pluralsight

Demonstration:

A Distributed Update Service

pluralsight

Moving Forward …

In this module:

- UDP sequence of operations
- UDP sockets API
- Text and binary protocols
- The rcat client
- UDP broadcasting
- Distributed update service

Coming up in the next module:

- Concurrent servers and clients

pluralsight

Concurrent Servers and Clients

Chris Brown

pluralsight

In This Module …

The process-per-client concurrent server model

Avoiding zombies

Concurrent servers using `select()`

Maintaining state in single-process servers

pluralsight

The Need for Concurrency

- The TCP-based server we wrote earlier was iterative
  - Completed dialogue with one client before accepting a connection from the next
  - Clients have to wait
  - If the connection queue fills up, connections will be refused
- The servers we will write in this chapter are concurrent
- - Able to conduct dialogues with multiple clients simultaneously

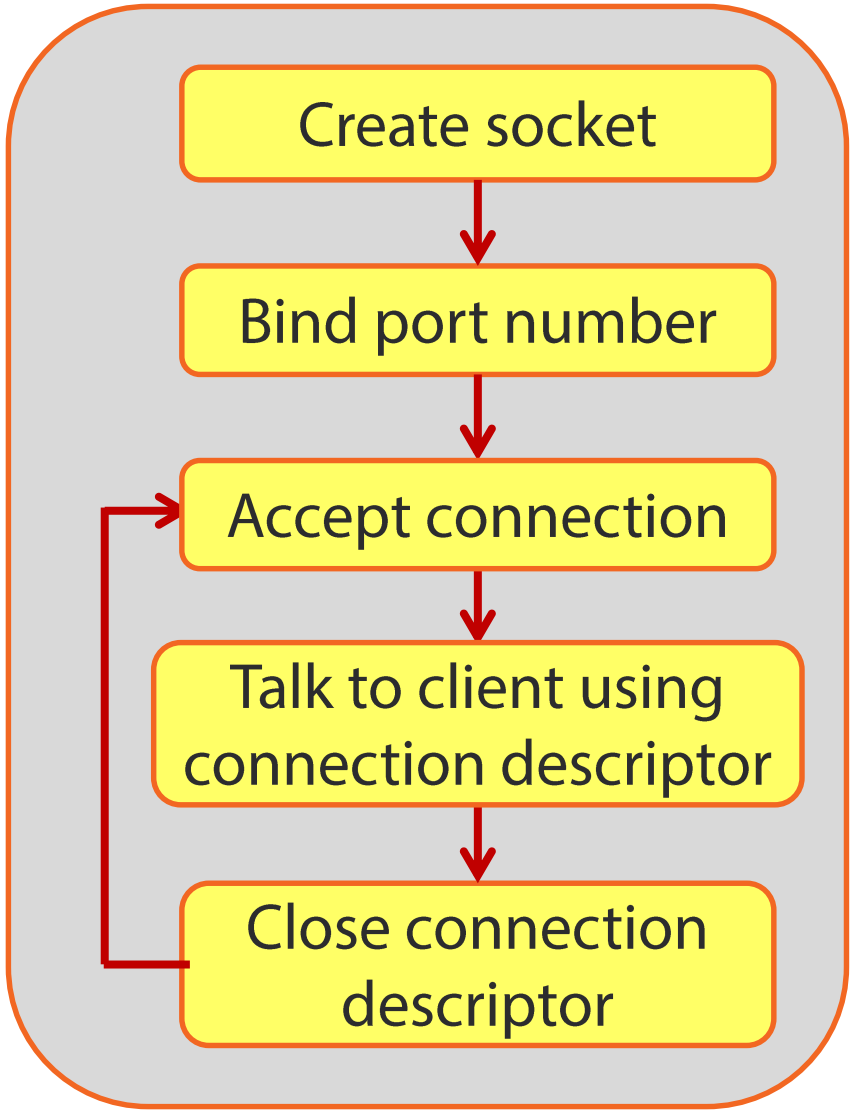
pluralsight

Iterative Server Schema (recap)

```
sock = socket( ... );
bind(sock, ...);
listen(sock, 5);

while(1) {
  fd = accept(sock, ...);
  /* Conduct dialogue with client,
     using fd for input and output
  */
  close(fd);
}
```

pluralsight

Iterative Servers Illustrated



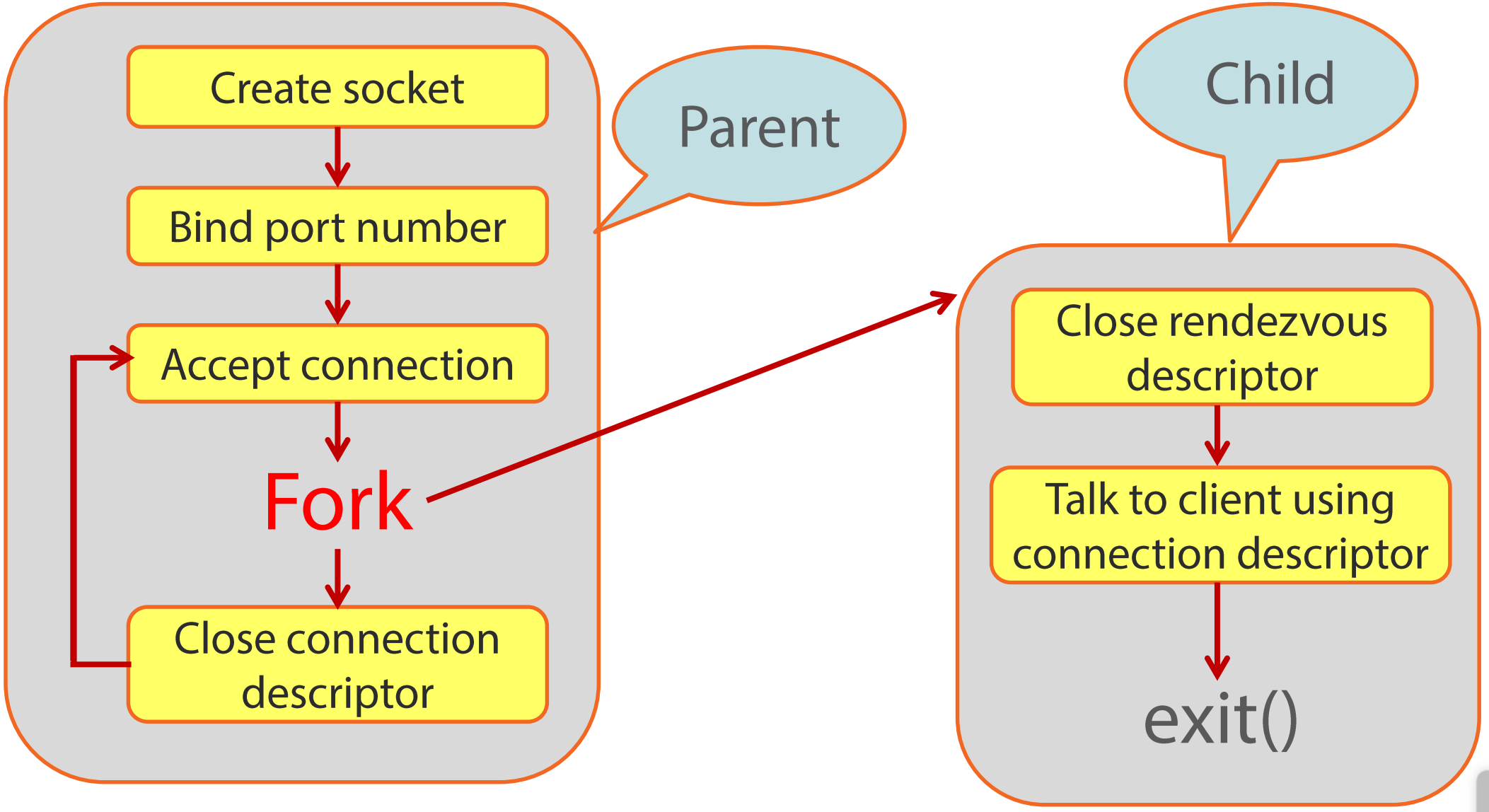
pluralsight

Concurrent Server Schema

```
sock = socket( ... );
bind(sock, ...);
listen(sock, 5);

while(1) {
  fd = accept(sock, ...);
  if (fork() == 0) {
    close(sock);  // Child – process request
    /* Conduct dialogue with client,
       using fd for input and output */
    exit(0);
  } else
      close(fd);  // Parent
}
```

pluralsight

Concurrent Servers Illustrated



pluralsight

Demonstration

Concurrent "hangman" server

pluralsight

The Game of "Hangman"

protection

e?

s?

t?

a?

i?

pluralsight

The Zombie Problem

```
#include <unistd.h>
#include <stdlib.h>

main()
{
  if (fork() == 0) exit(0);
  sleep(1000);
}
```

pluralsight

The Zombie Solution

```
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

main()
{
  signal(SIGCHLD, SIG_IGN);
  if (fork() == 0) exit(0);
  sleep(1000);
}
```

pluralsight

Concurrency Within a Single Process

Single-process concurrent servers

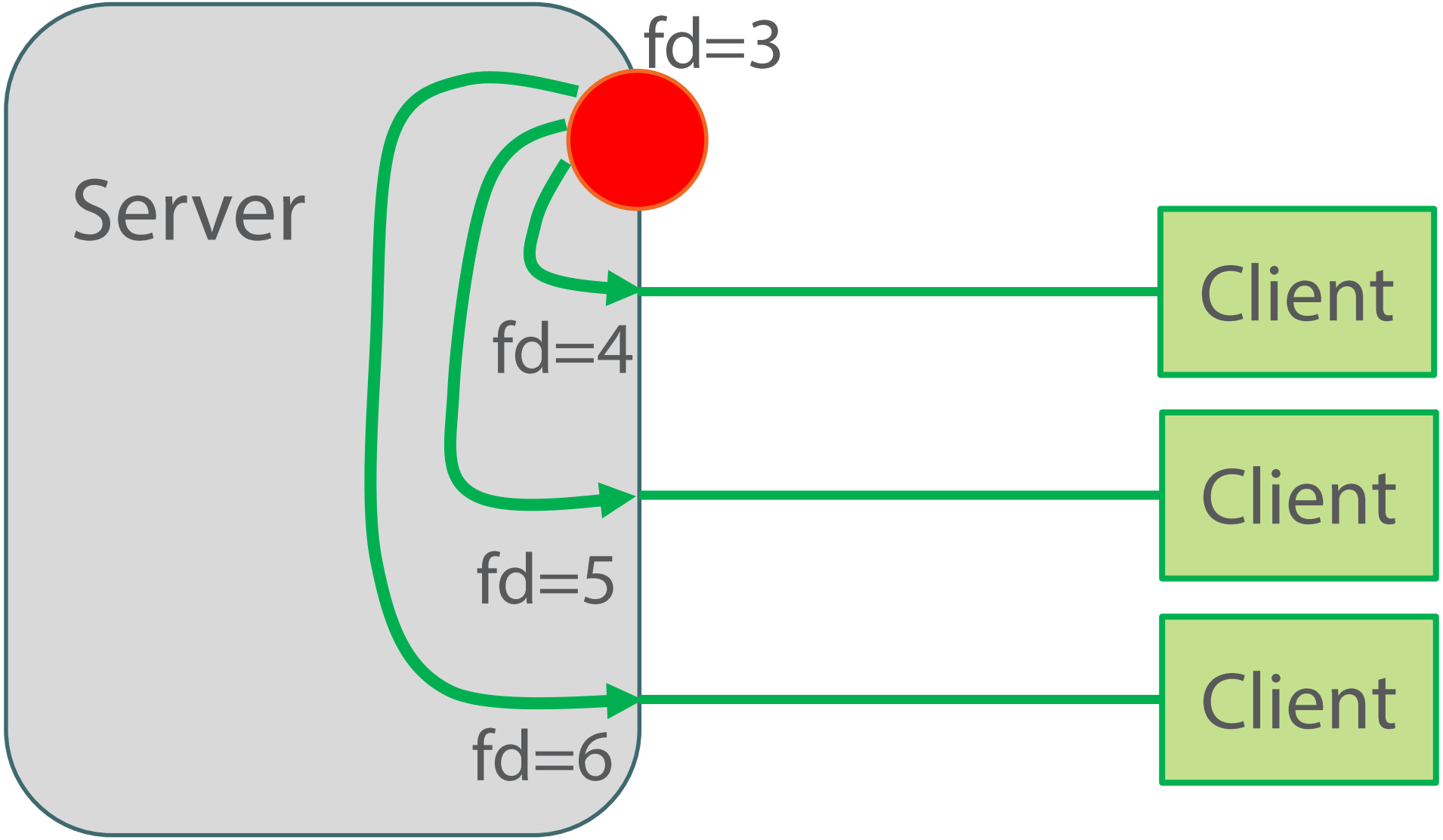
The `select()` call

Maintaining State

Demonstration

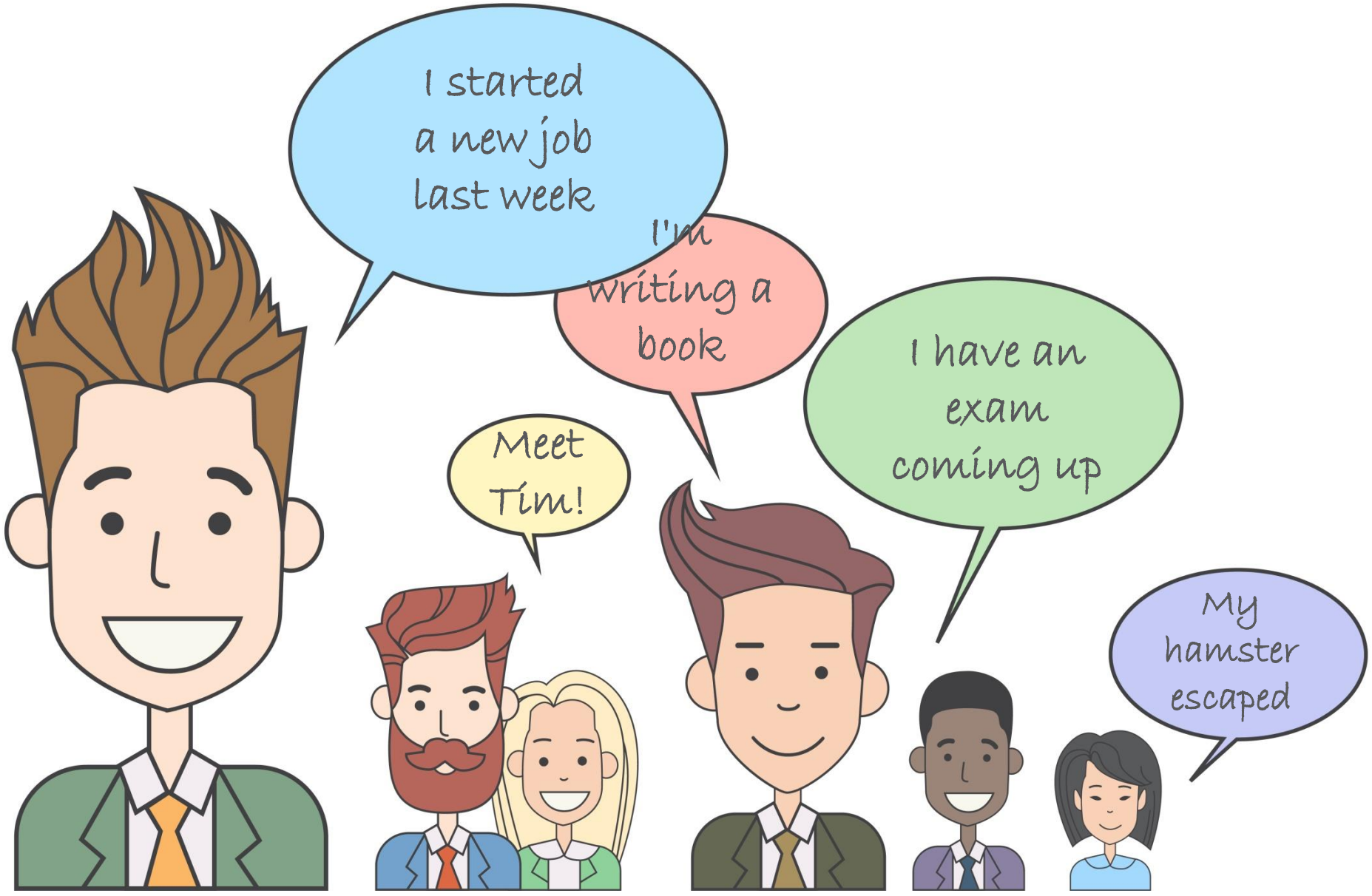
pluralsight

Supporting Multiple Clients



pluralsight

Maintaining Per-Client State



pluralsight

State in the Concurrent Hangman Server

? What per-client state does the hangman server need to store?

—The "target" word
—The word as guessed so far
— How many lives remain

```
prohibit
-r--ibi-
7
```

? How does it store it?

—Each child process has its own instance of the program's variables

pluralsight

State in the rot13 Server

? What per-client state does the *rot13* server need to store?

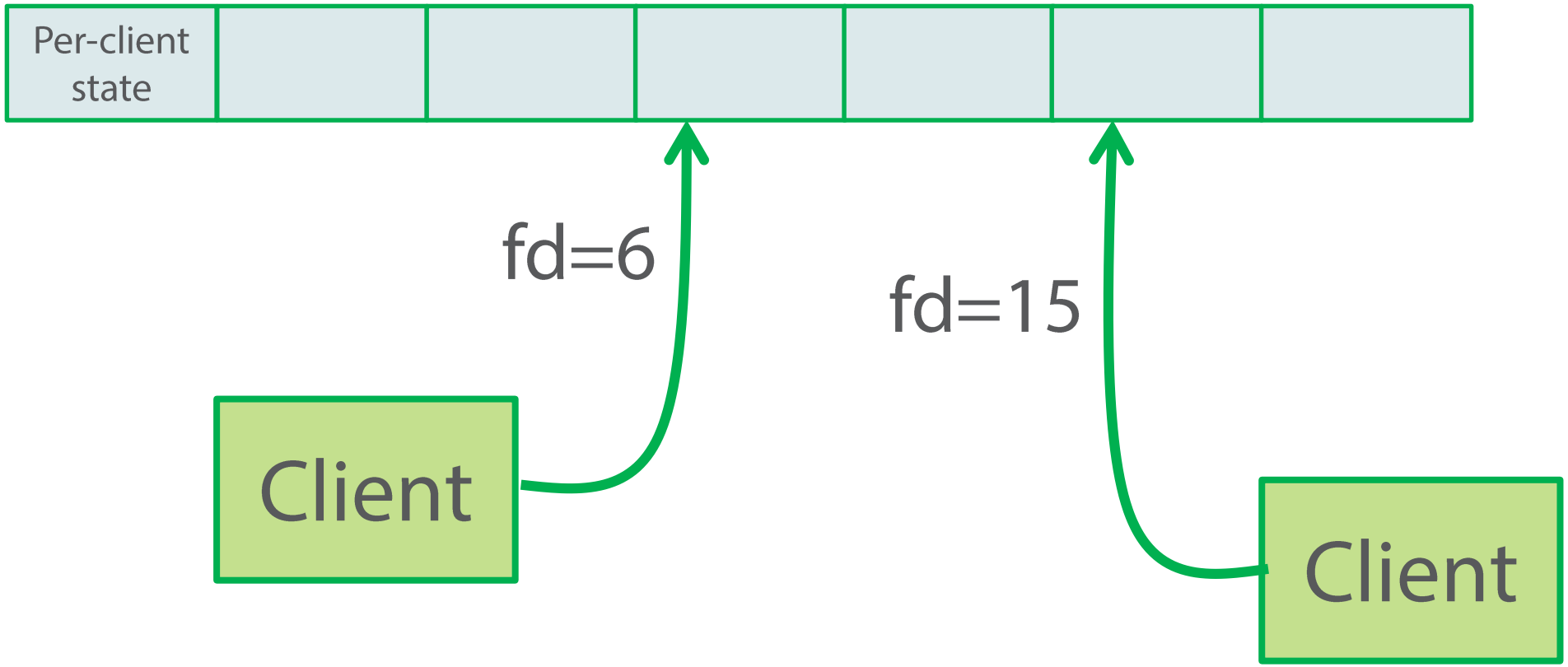
—None!

—Each request is serviced entirely without reference to earlier requests

—Stateless

pluralsight

Maintaining State in Single-Process Servers



pluralsight

Sharing State

pluralsight

Sharing State

Single-process concurrent server

Easy!

— Just use global variables

— No risk of race conditions

Process-per-client concurrent server

Harder

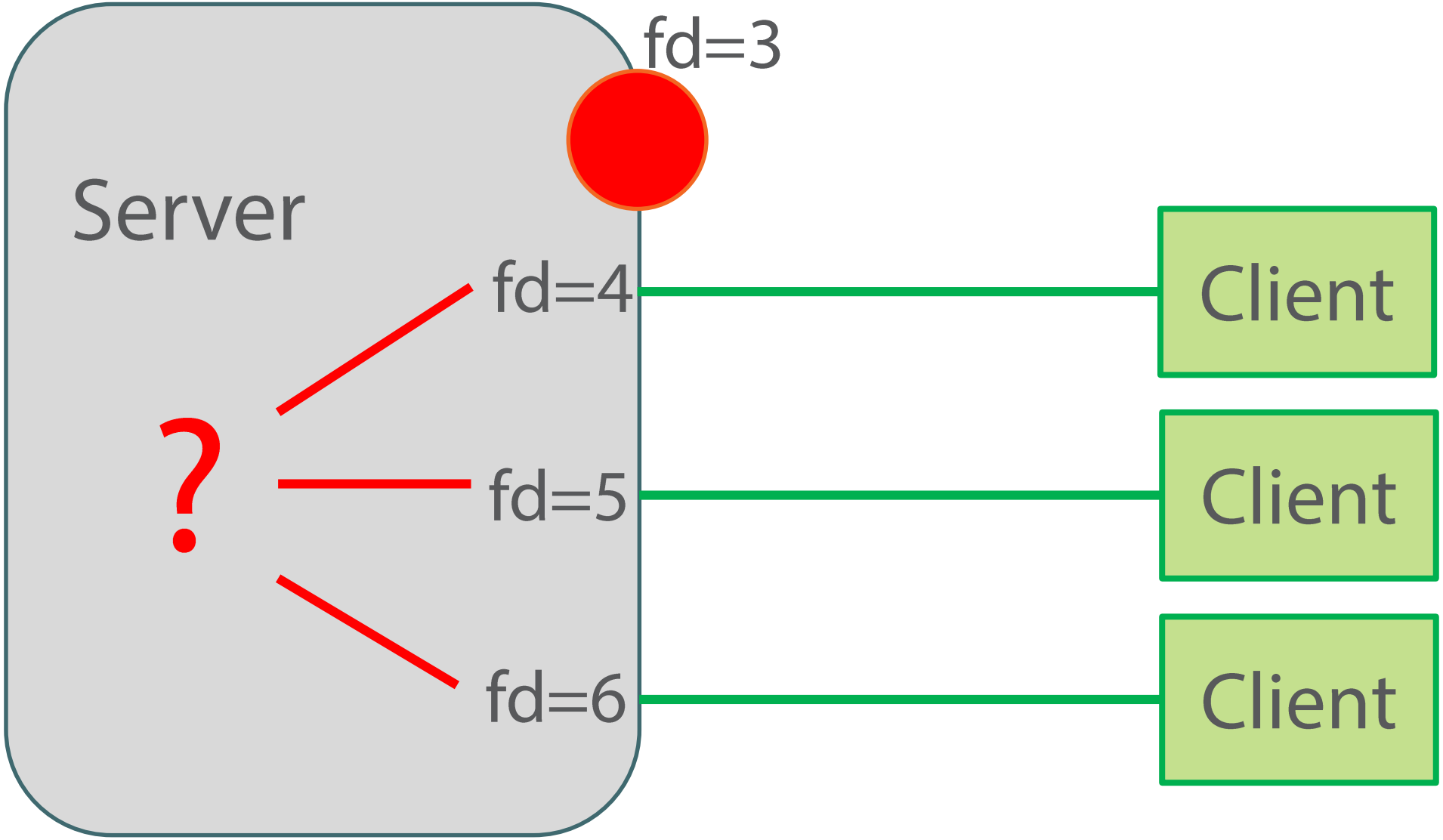
— Create a shared memory segment

— Or use a file or a database

— Need to avoid race conditions

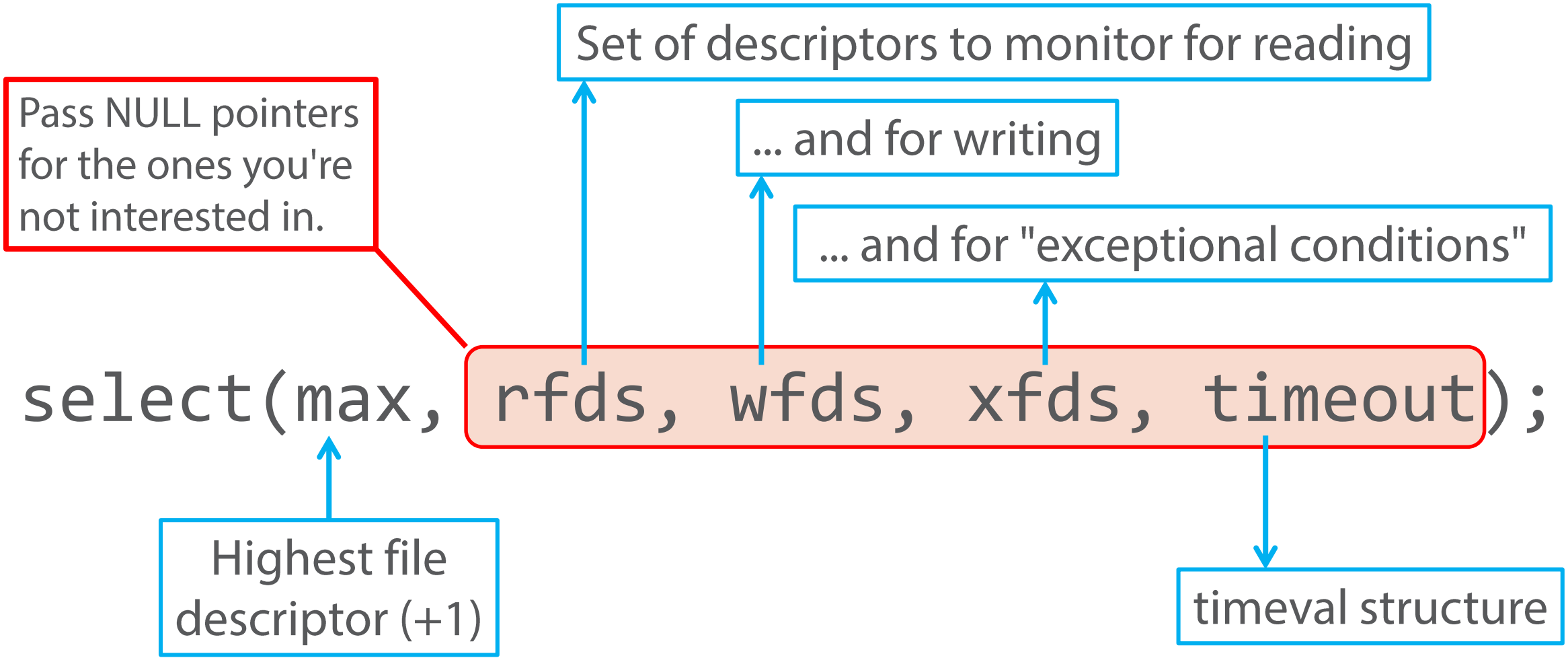
pluralsight

Managing Multiple File Descriptors



pluralsight

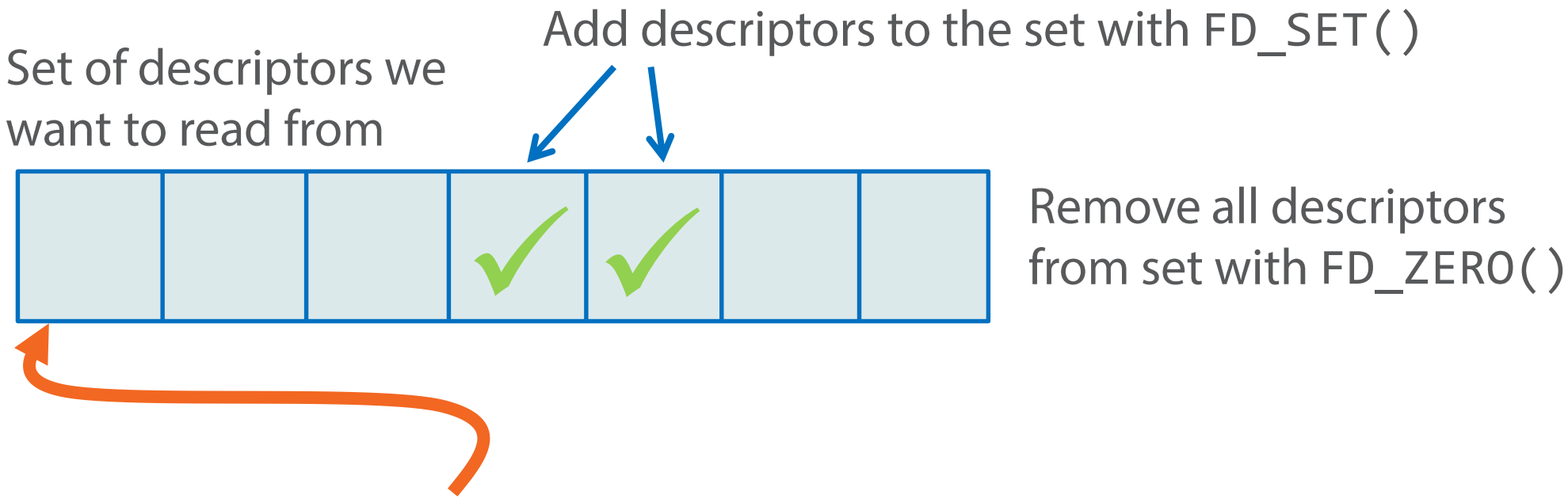
The `select()` System Call



pluralsight

Using Descriptor Sets with `select()`

Before the call ...



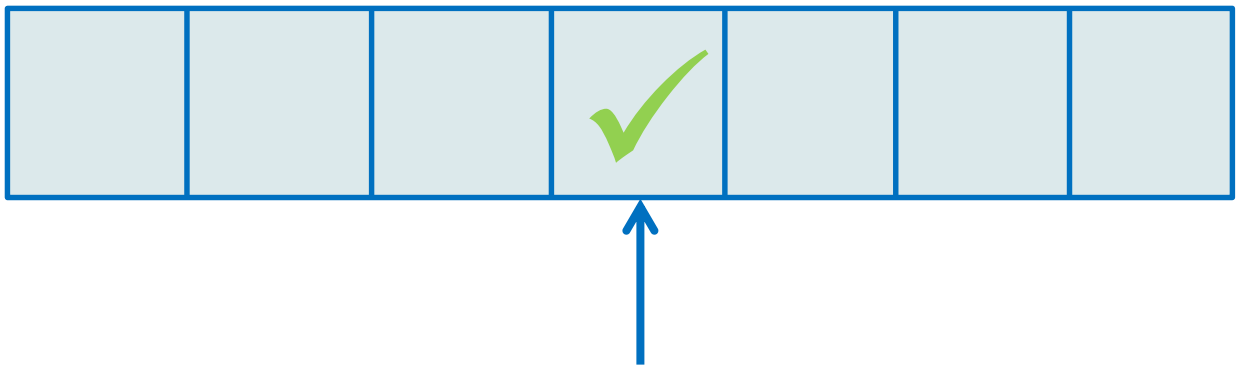

```
select(max, rfds, wfds, xfds, timeout);
```

pluralsight

Using Descriptor Sets with `select()`

After the call ...

Set of descriptors that
are ready for reading



Test if a descriptor is ready with `FD_ISSET()`

pluralsight

Simple `select()` Example

```
fd_set myset;
/* Put descriptors f1 and f2 into myset */
FD_ZERO(&myset);
FD_SET(f1, &myset);
FD_SET(f2; &myset);

/* Wait until f1 or f2 is ready for reading /*
select(16, &myset, NULL, NULL, NULL);
if (FD_ISSET(f1, &myset)) {
    // Read from descriptor f1
}
if (FD_ISSET(f2, &myset)) {
    // Read from descriptor f2
}
```

pluralsight

Demonstration

Concurrent
single-process server

pluralsight

Module Summary

The need for concurrency

Classic process-per-client concurrent servers

State and how to maintain it

Concurrent servers using `select()`

pluralsight

Moving Forward …

Coming up in the next module:

- Threads (compared to processes)
- The "pthreads" API
- Concurrent servers and clients using threads
- How to be "thread-safe"

pluralsight

Multi-threaded Concurrency

Chris Brown

In This Module ...

Thread concepts

The pthreads API

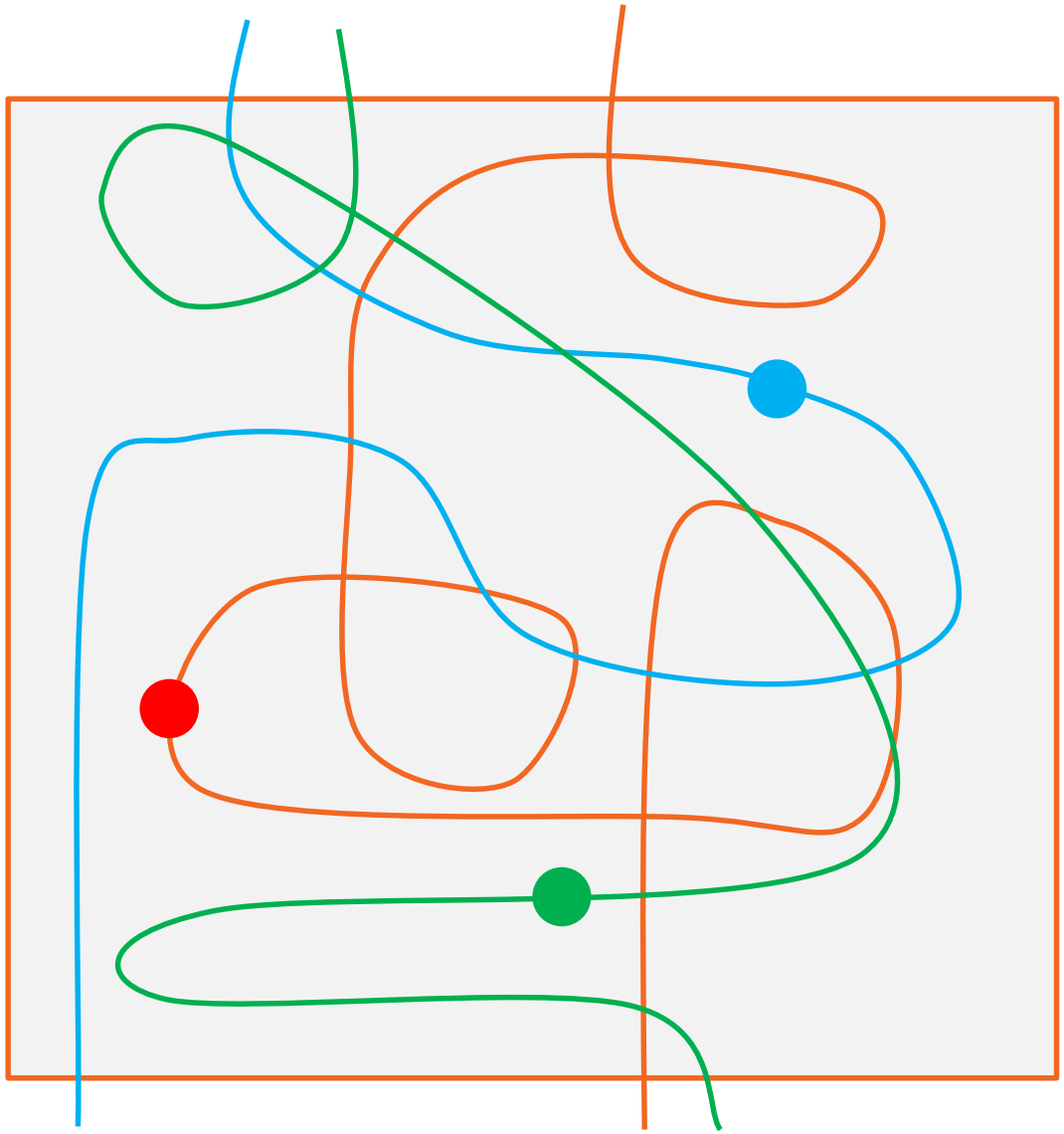
Multi-threaded servers and clients

Processor farms

Demonstration: Counting primes

Writing "thread-safe" code

Multi-threading Illustrated



Reasons for Multi-threading

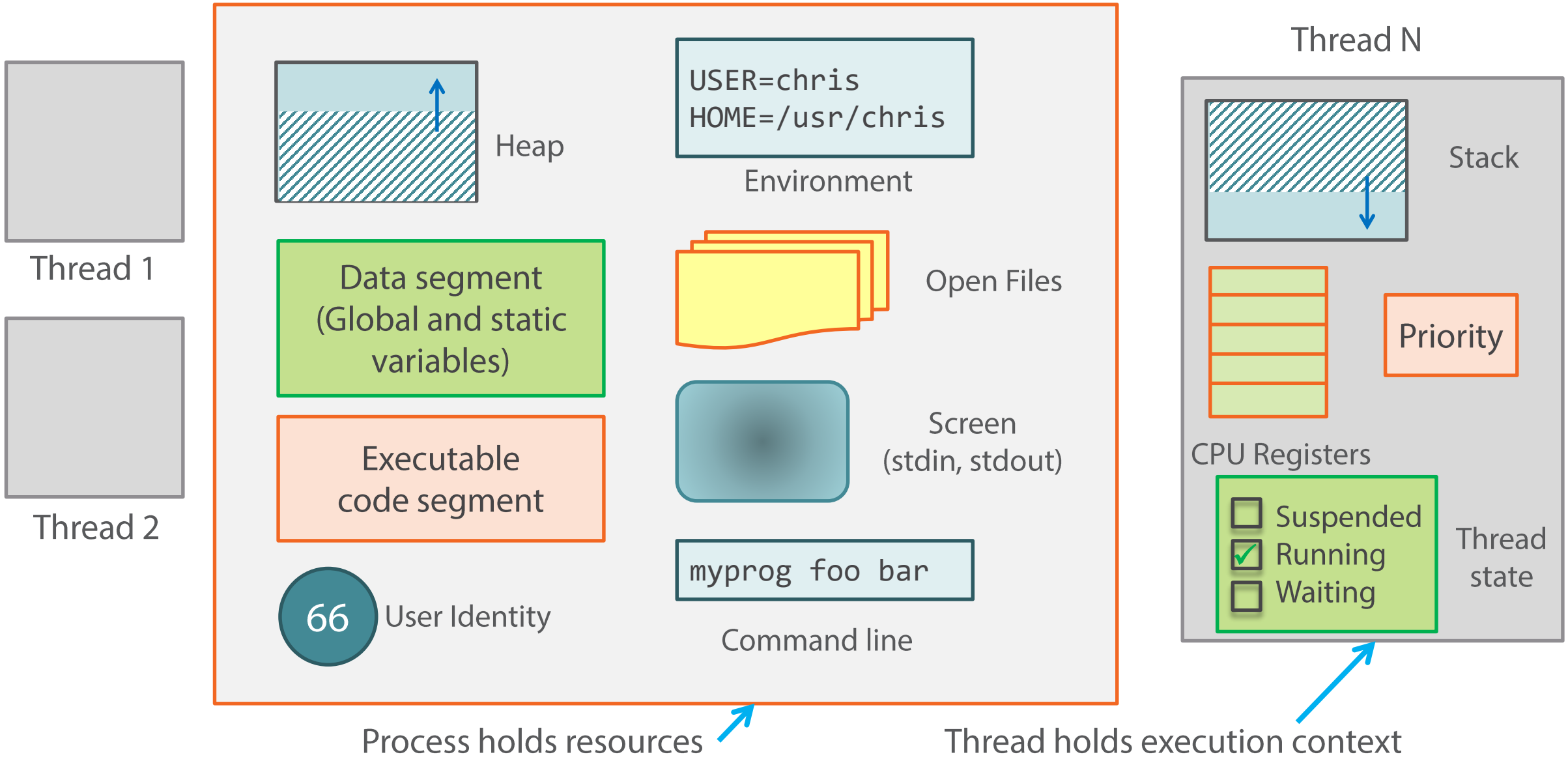
Expressing logical concurrency

Implementing background tasks

Concurrent servers

Exploiting multi-processor hardware

Threads and Processes



Shared and Not Shared

Threads share:

- Code
- Global and static variables
- Open file descriptors

Threads do not share:

- Variables local to functions

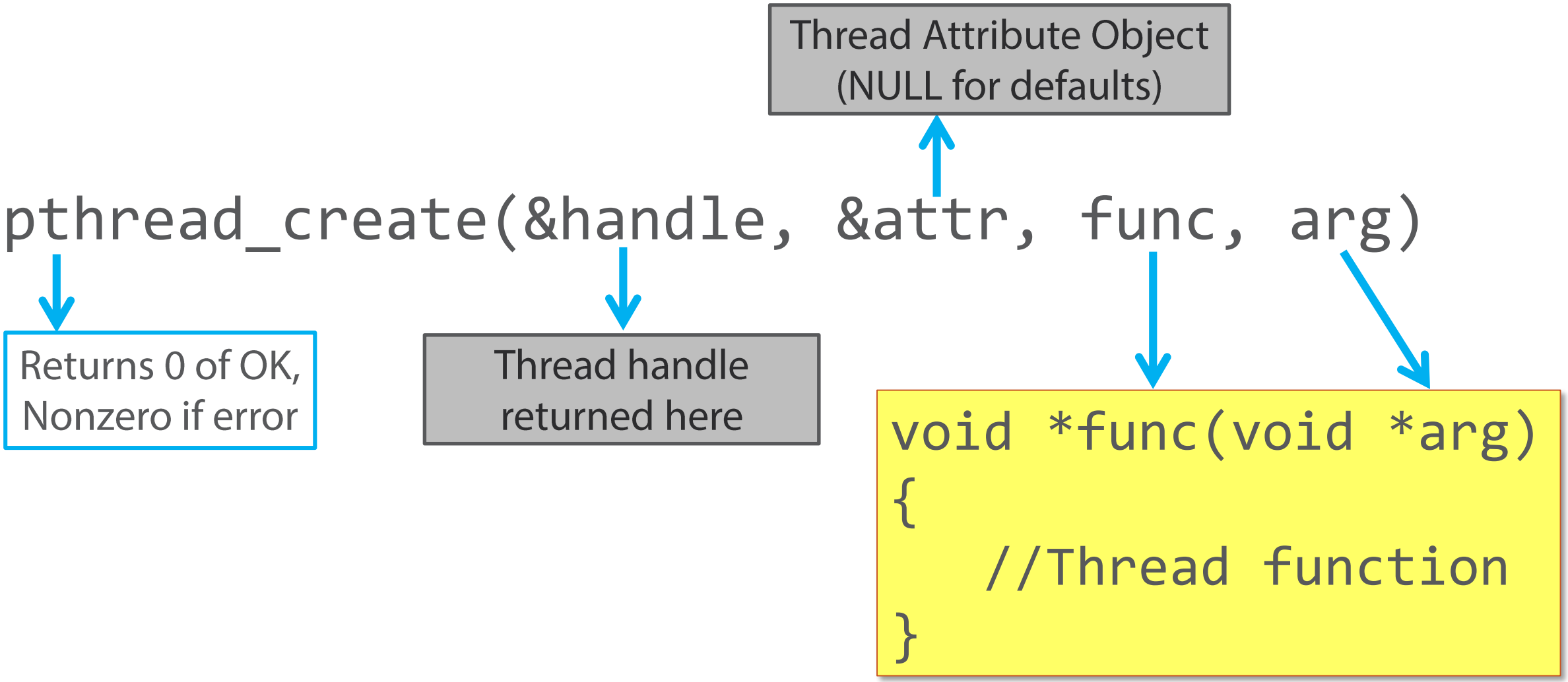
Pthreads

POSIX 1003.1c

A standardised set of C library routines for thread creation and management

— Upwards of 60 functions

Thread Creation



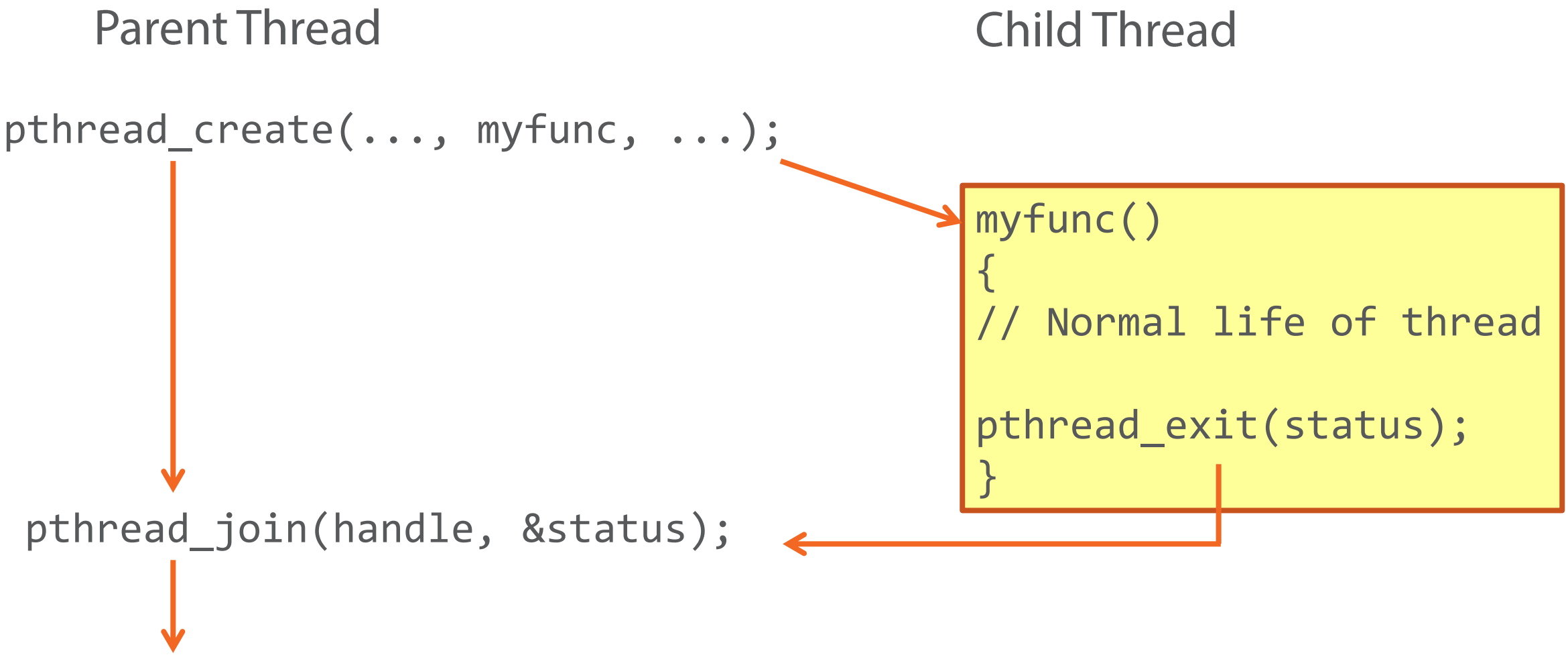
Thread Termination

- A thread can terminate in several ways:
  1. By calling `pthread_exit(exit_status)`
  2. By returning from its top-level function
  3. By some other thread sending a cancellation request:

     `pthread_cancel(handle);`
- Parent can wait for thread to finish and get exit status:
  - `pthread_join(handle, &exit_status);`
- Parent should detach thread if they don't need to join on it:

  `pthread_detach(handle);`

Thread Life Cycle



Demonstration

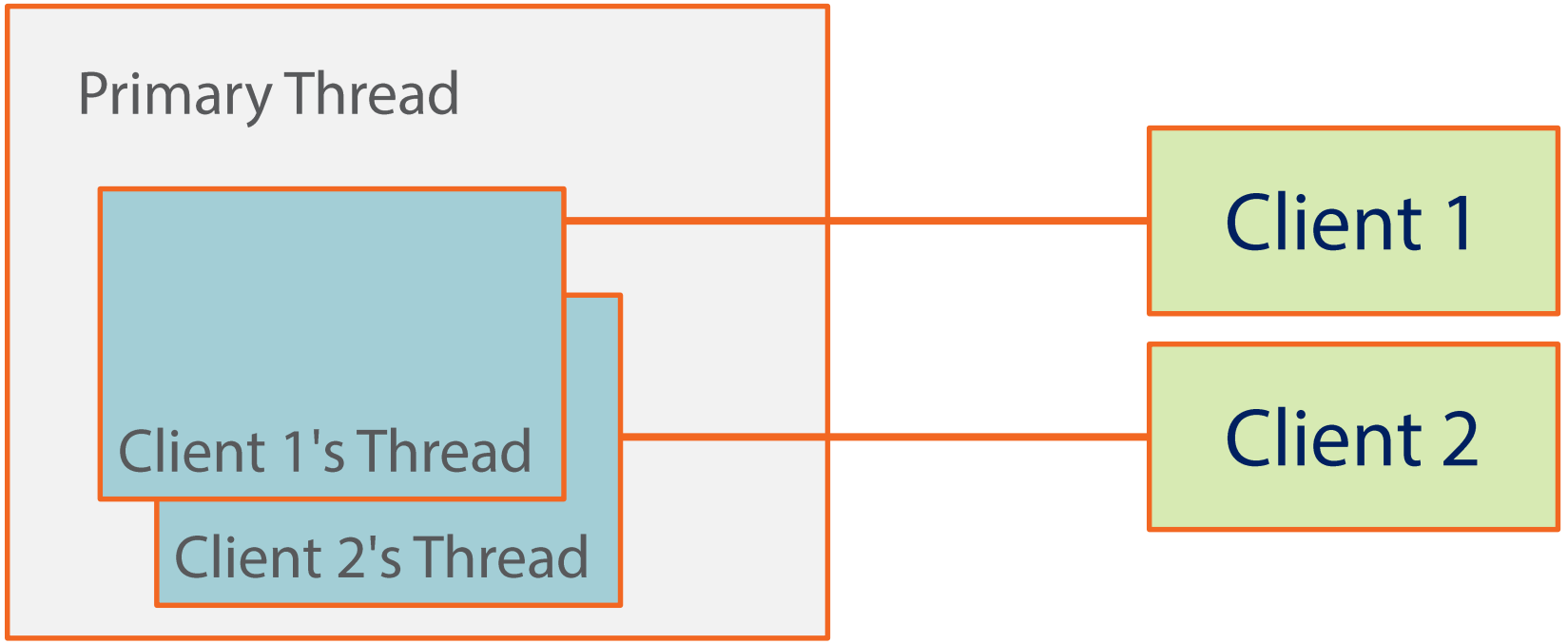
Simple thread example

Thread Example

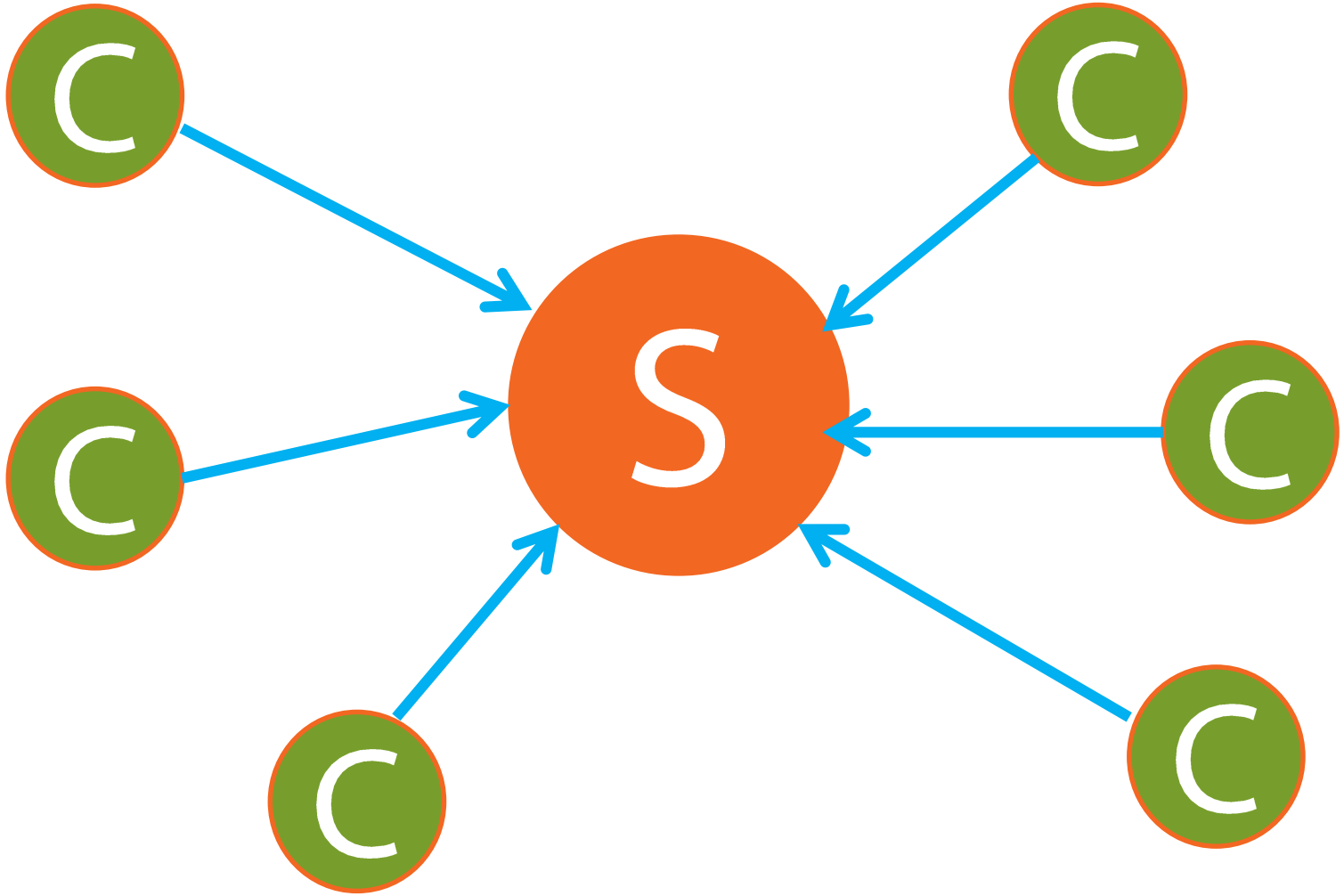
```
#include <pthread.h>
#include <stdio.h>

void *func(void *arg)
{
    printf("child thread says %s\n", (char *)arg);
    pthread_exit((void *)99);
}

int main()
{
    pthread_t handle;
    int exitcode;

    pthread_create(&handle, NULL, func, "hi!");
    printf("primary thread says hello\n");
    pthread_join(handle, (void **)&exitcode);
    printf("exit code %d\n", exitcode);
}
```

Multi-threaded Server

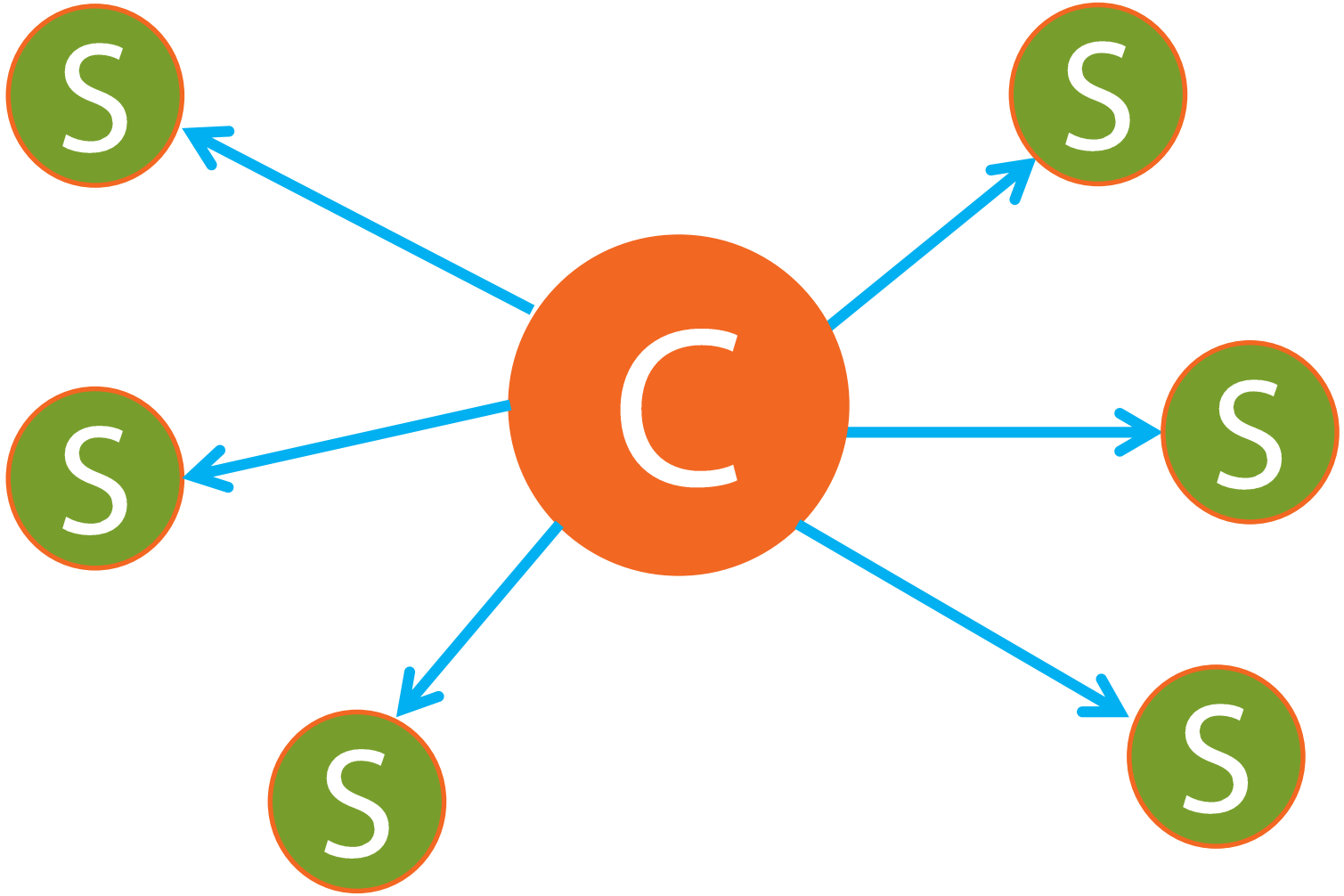
- Thread-per-client is an elegant model for concurrent servers
  - Efficient
  - Easy to keep per-client state (local variables)
  - Easy to share state (global variables)



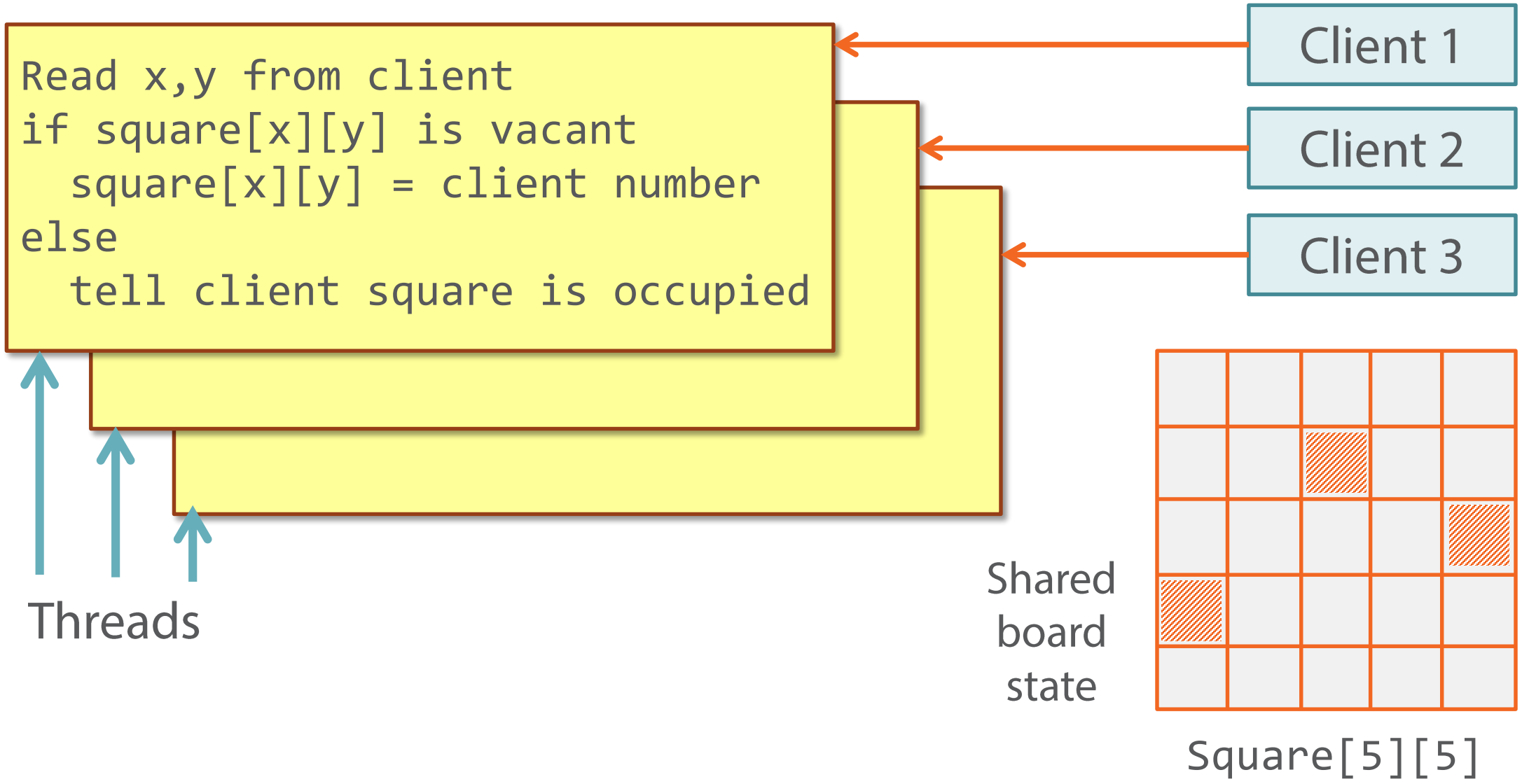
Concurrent Server



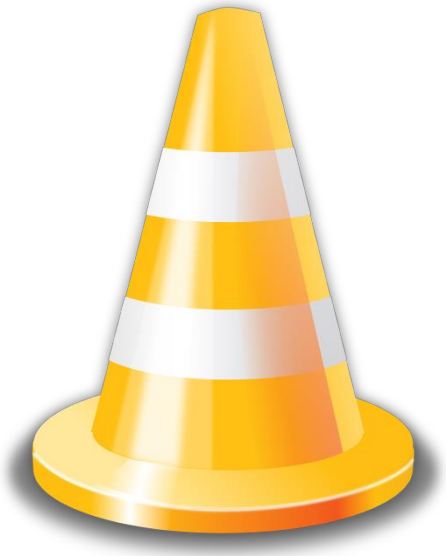
Processor Farm



Sharing Client State



Thread Safety



Problems can arise when multiple asynchronous threads access shared data

Mutual exclusion locks ("mutexes") control entry to code sections that update or access shared state

"Thread-safe" code

Library functions you call must also be thread-safe

Pthread Mutexes

```
#include <pthread.h>

static int shared_data;
pthread_mutex_t mylock = PTHREAD_MUTEX_INITIALIZER;

func( ... )
{
        pthread_mutex_lock(&mylock);
        // Update or access shared_data here
        pthread_mutex_unlock(&mylock);
}
```

Demonstration

Why do we need mutexes?

Processor Farms

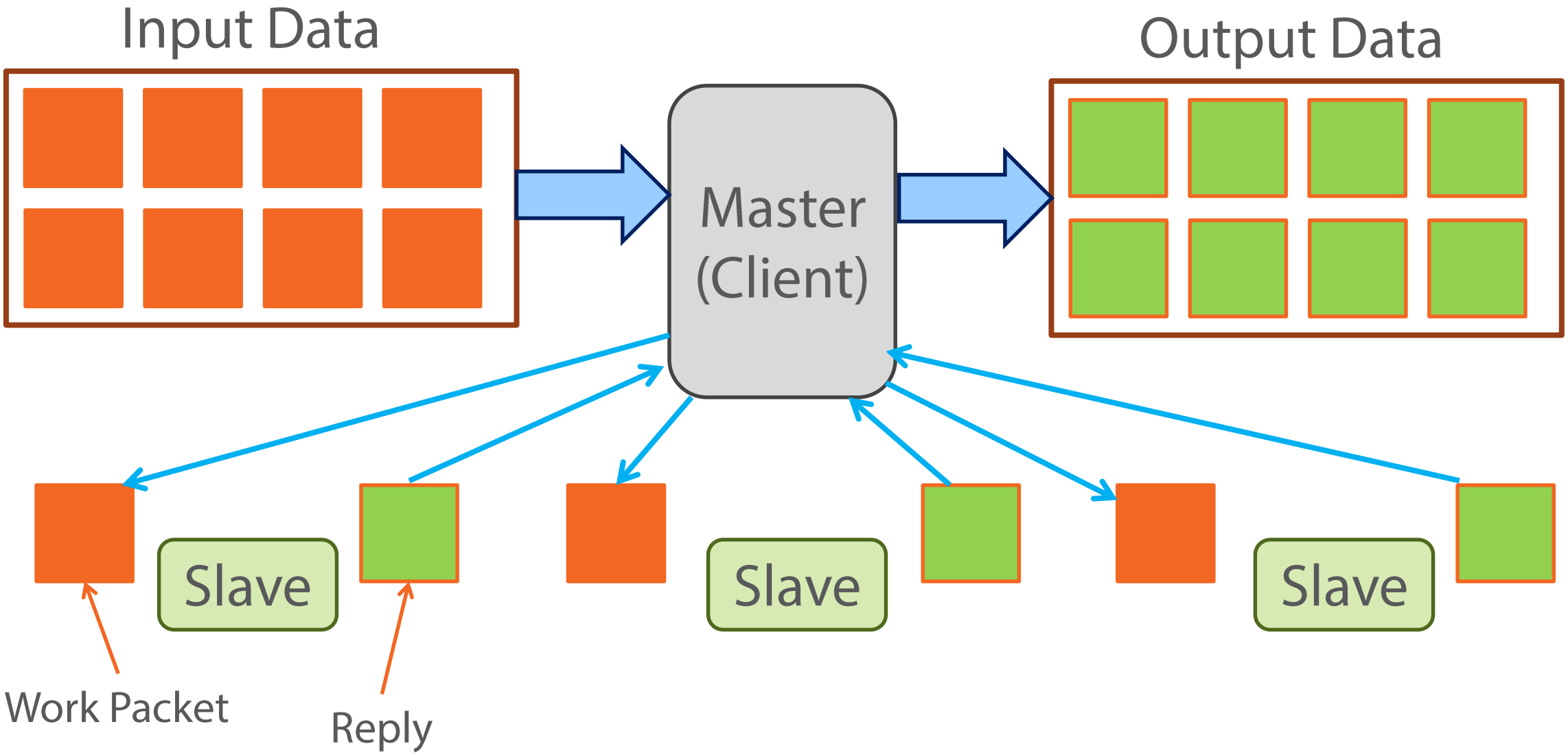
A common model for decomposing computationally-intensive tasks

Input data set broken down into "work packets"
— each processed independently of the others

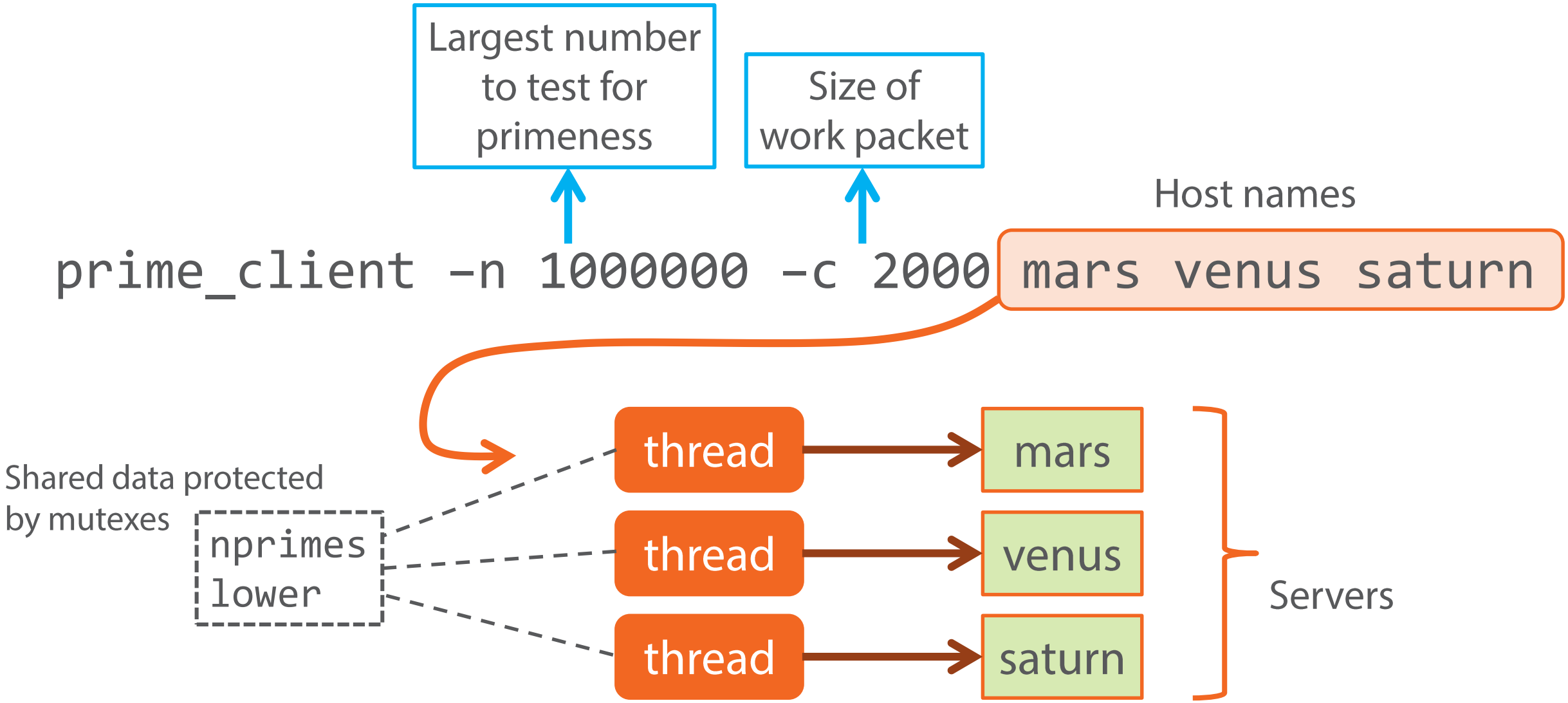
Need ≈ 10x as many work packets as processors
— Automatic load balancing

Goal: Linear speedup

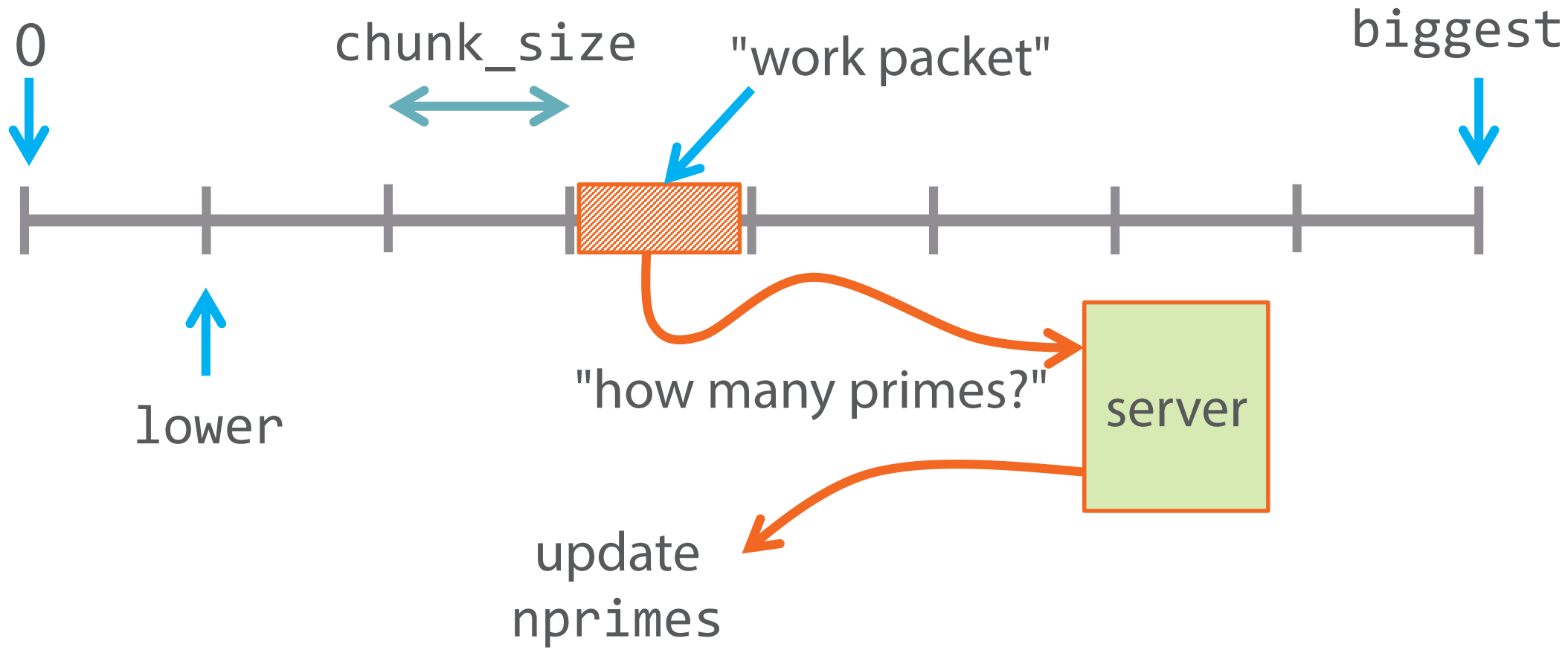
Processor Farm Illustrated



A Processor Farm to Count Prime Numbers



Dividing up the Work



Demonstration

Processor Farm

Module Summary

In this module:

- Threads compared to processes
- The pthreads API
- Thread-safe code: accessing shared data
- The processor farm model
- A processor farm to count prime numbers

Course Summary

Congratulations!

In this course:

- The characteristics of TCP and UDP protocols
- Writing TCP-based servers
- Writing TCP-based clients
- UDP servers and clients
- Writing concurrent servers
- Writing concurrent clients using threads