CS 43: Computer Networks

05:Network Services and Distributed Systems

September 19, 2024

SWARTHMORE COLLEGE

Last class

- Inter-process communication using message passing
- How send and recv buffers work
- Concurrency

Today

- Server side TCP Sockets
- Application-layer communication paradigms:
  - Client-Server
  - Peer-to-peer architecture
- Distributed network applications: Sources of complexity

Where we are

Application: the application (e.g., the Web, Email)

Transport: end-to-end connections, reliability

Network: routing

Link (data-link): framing, error detection

Physical: 1's and 0's/bits across a medium
(copper, the air, fiber)

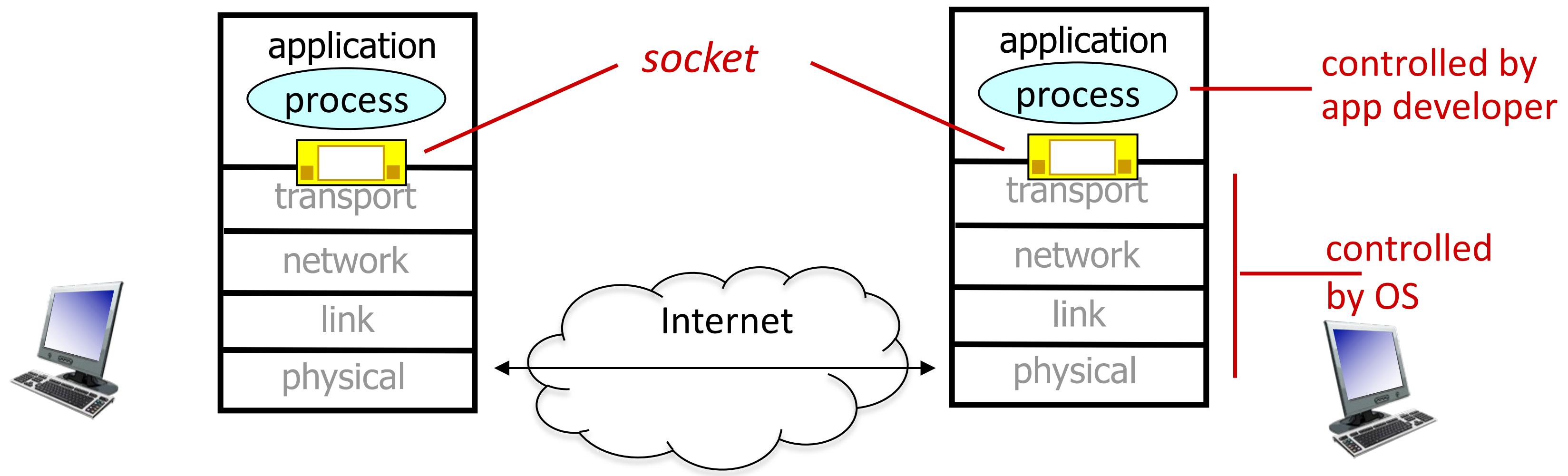
What is a socket?

An abstraction through which an application may send and receive data,

in the same way as a open-file handle allows an application to read and write data to storage.

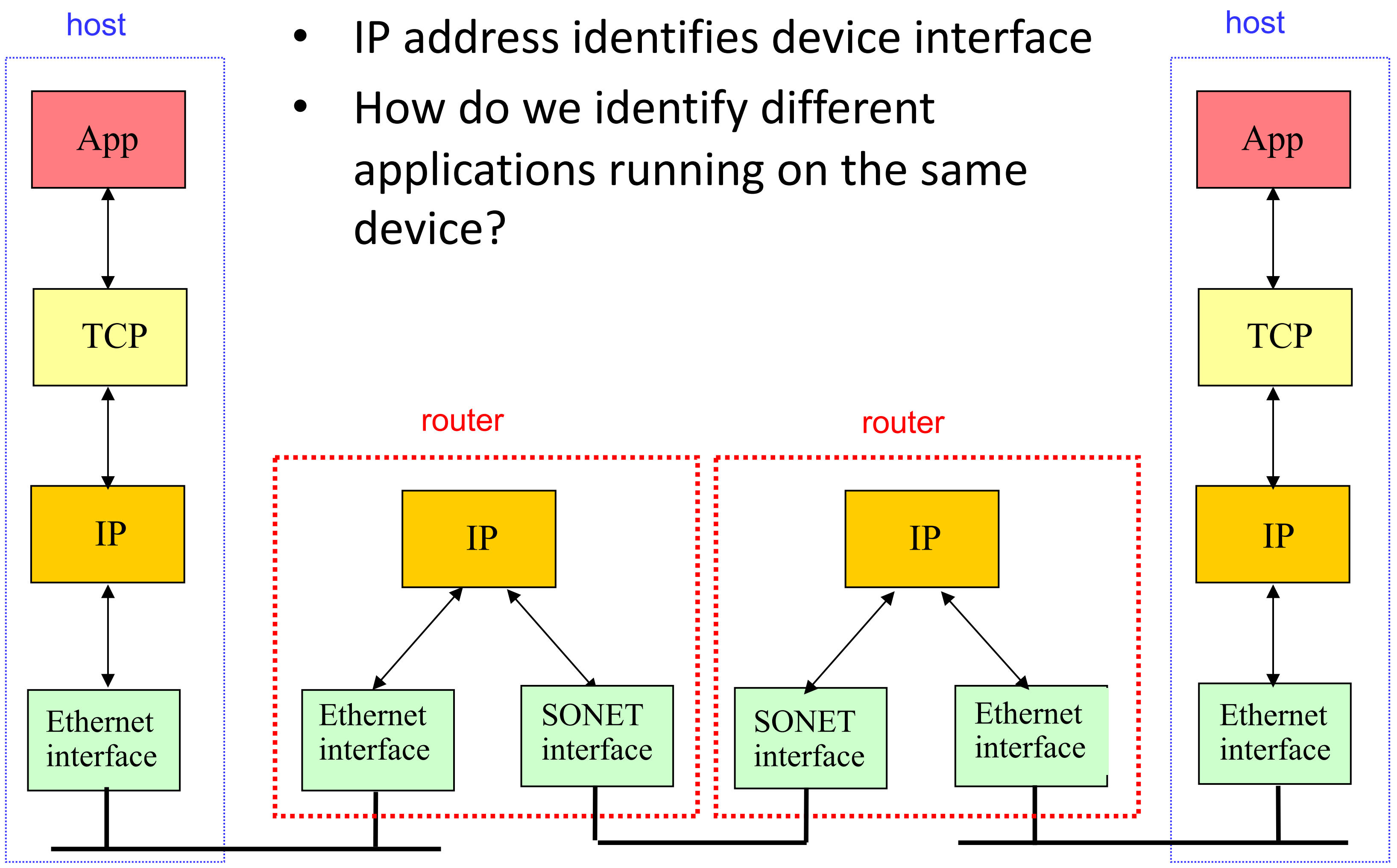
Sockets

- Process sends/receives messages to/from its socket
- Application has a few options, operating system handles the details
  - Choice of transport protocol (TCP, etc.)



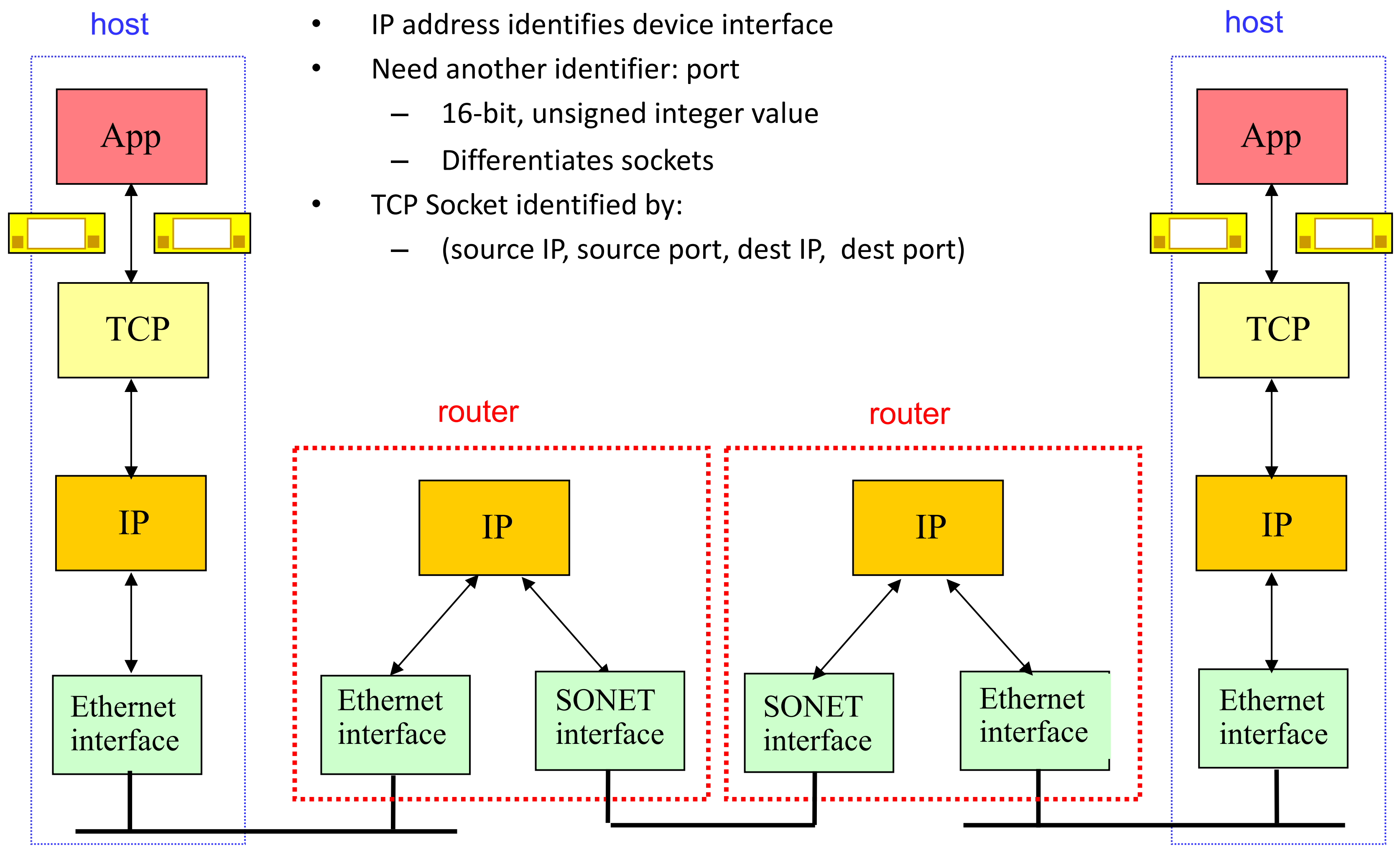
Addressing Sockets

- IP address identifies device interface
- How do we identify different applications running on the same device?



Addressing Sockets

- IP address identifies device interface
- Need another identifier: port
  - 16-bit, unsigned integer value
  - Differentiates sockets
- TCP Socket identified by:
  - (source IP, source port, dest IP, dest port)

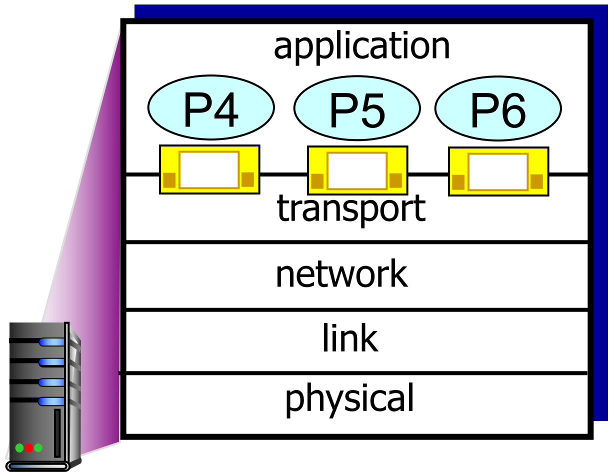


Connection-oriented: example

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- Receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

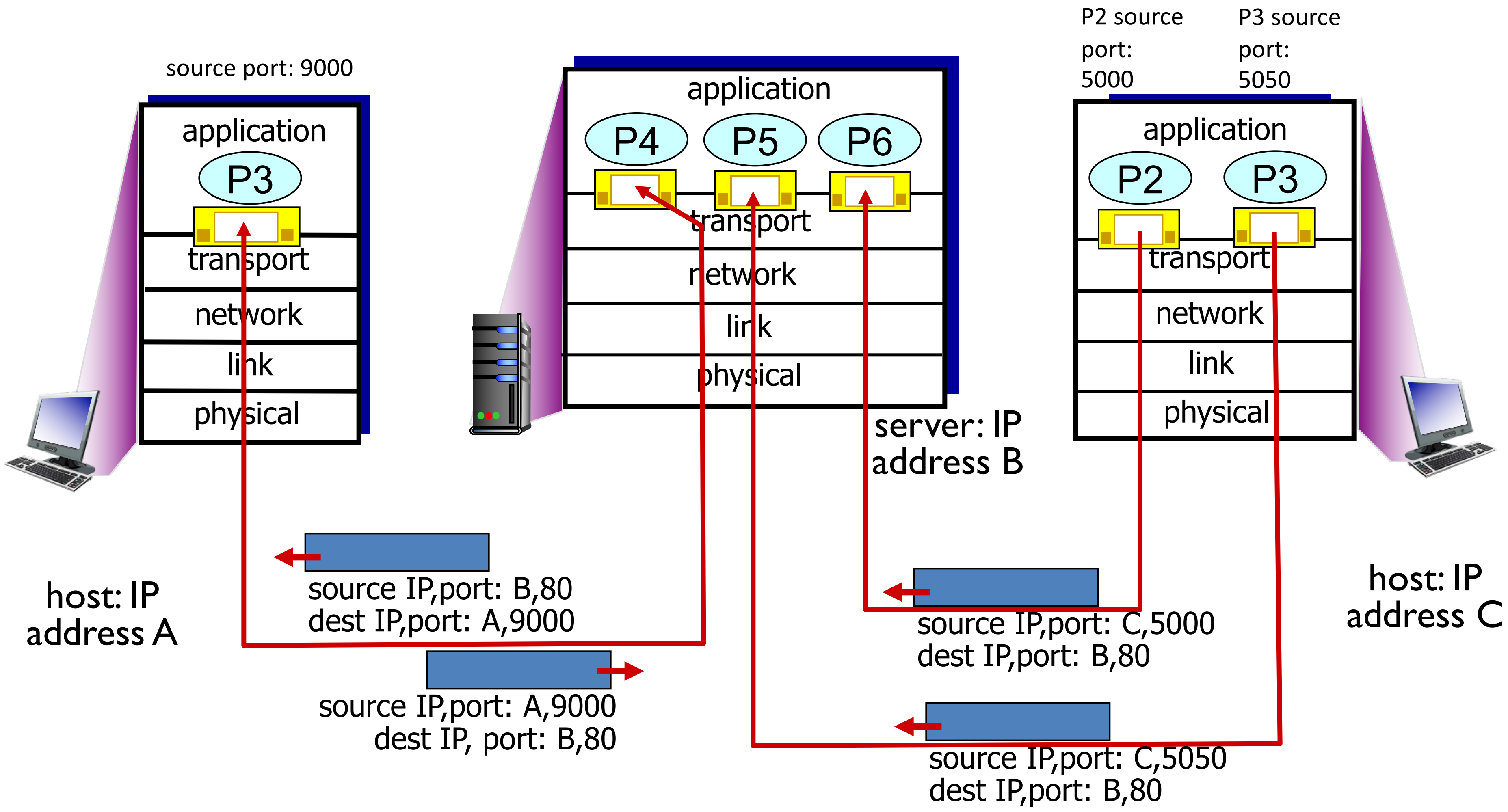
Connection-oriented: HTTP example

A socket is uniquely identified by (source IP, source port, dest IP, dest port)

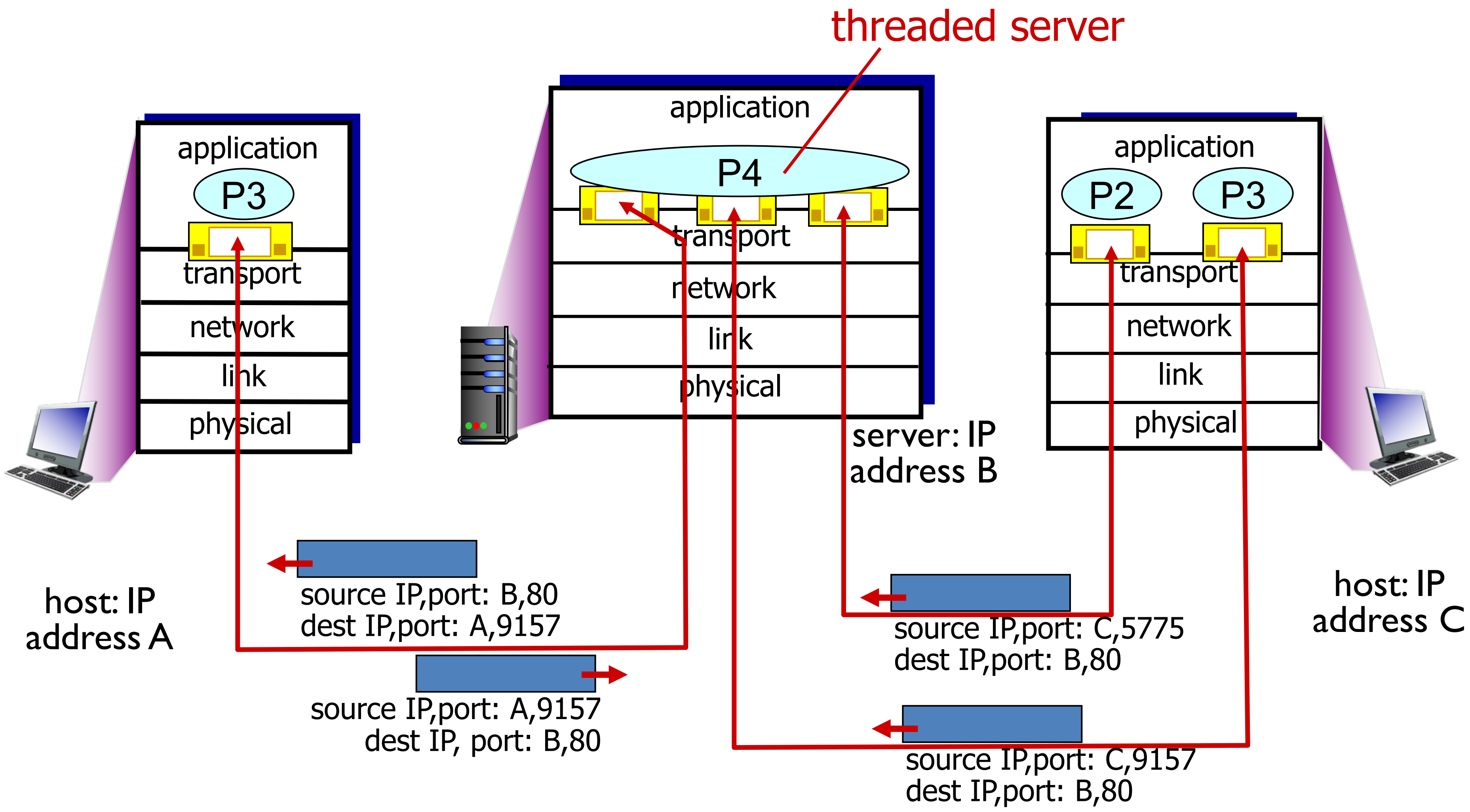


Connection-oriented: HTTP example

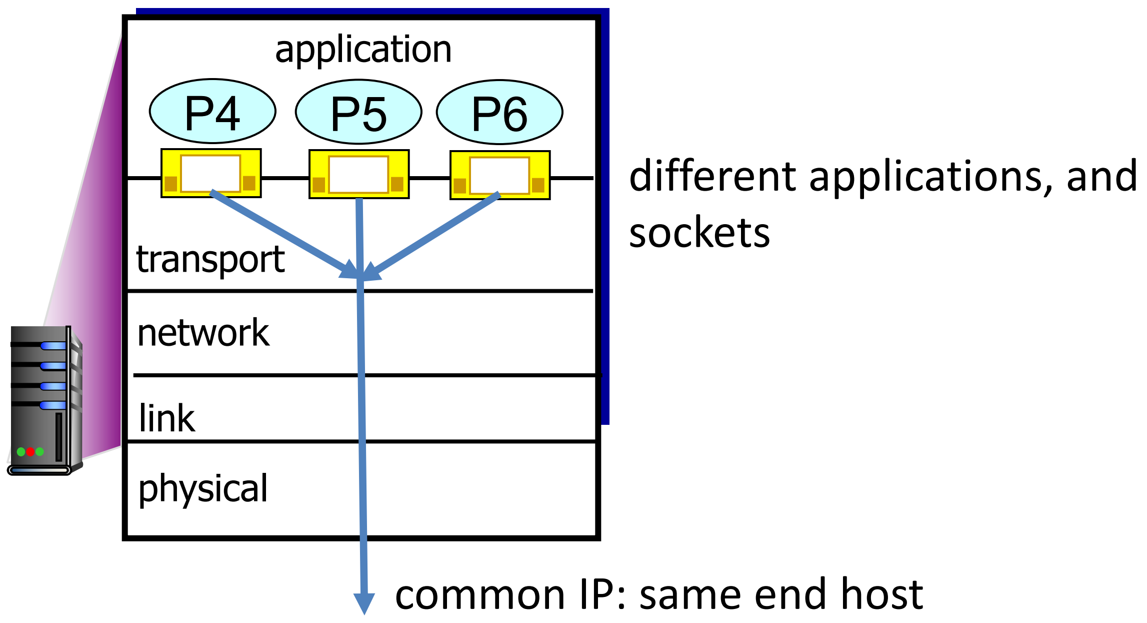
A socket is uniquely identified by (source IP, source port, dest IP, dest port)



Connection-oriented: example



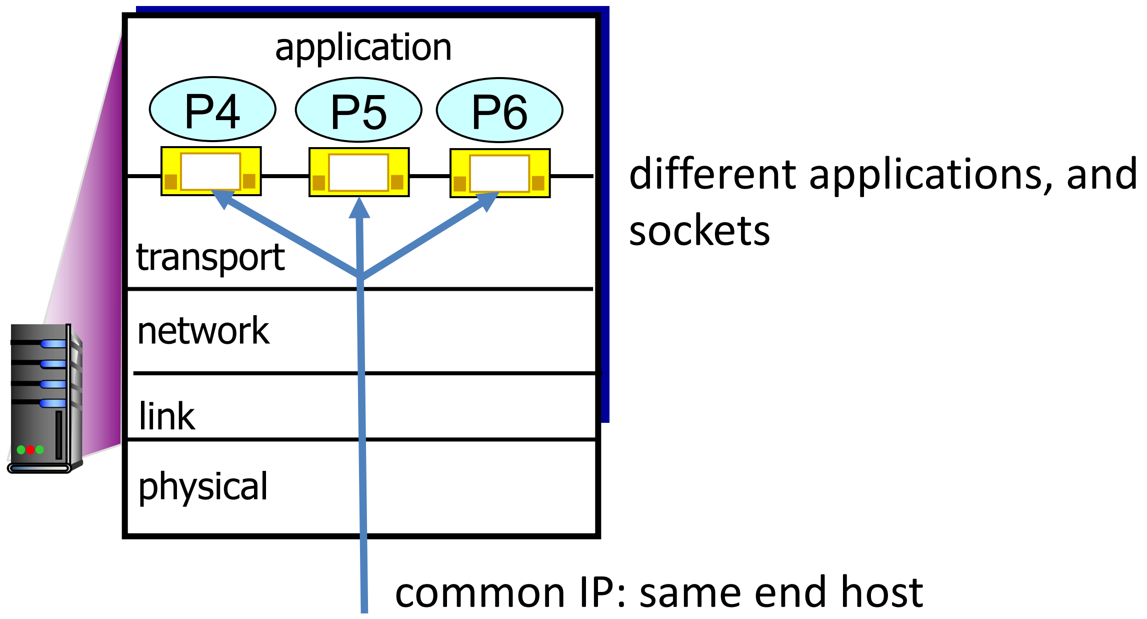
Multiplexing/Demultiplexing



Multiplexing:

– gather data packets from multiple sockets,
– encapsulate each packet with transport header inforation
– pass the packet to the network layer to send it over a shared communication channel.

Multiplexing/Demultiplexing



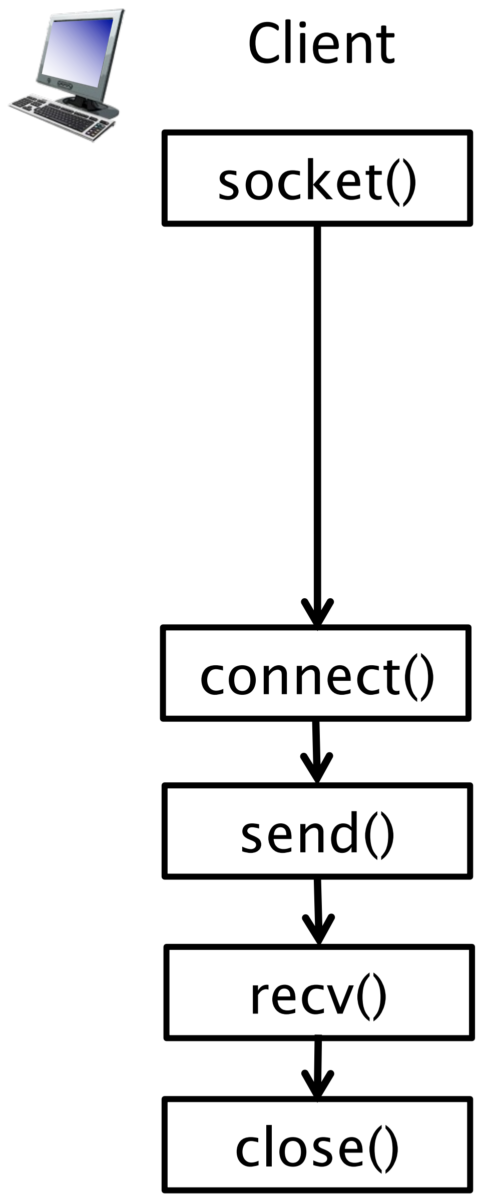
De-Multiplexing:

- examine transport layer header of data packet sent from the network layer
- identify receiving socket
- deliver data to the correct socket for each application

Application Design: Client-Server architecture

- Client:
  - initiates communication
  - must know the address and port of the server
  - active socket
- Server:
  - passively waits for and responds to clients
  - passive socket

TCP Socket Procedures: Client



create a new communication endpoint

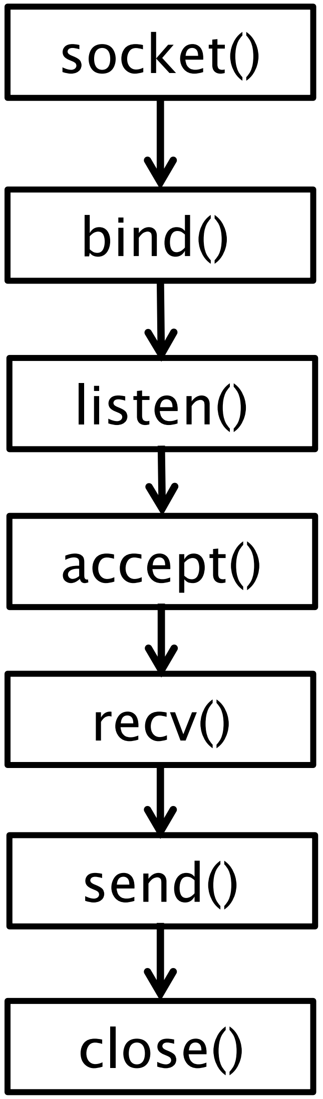
actively attempt to establish a connection

send some data over a connection

receive some data over a connection

release the connection

TCP socket procedures for a web server



socket: create a new communication endpoint

bind: attach a local address to a socket

listen: announce willingness to accept connections

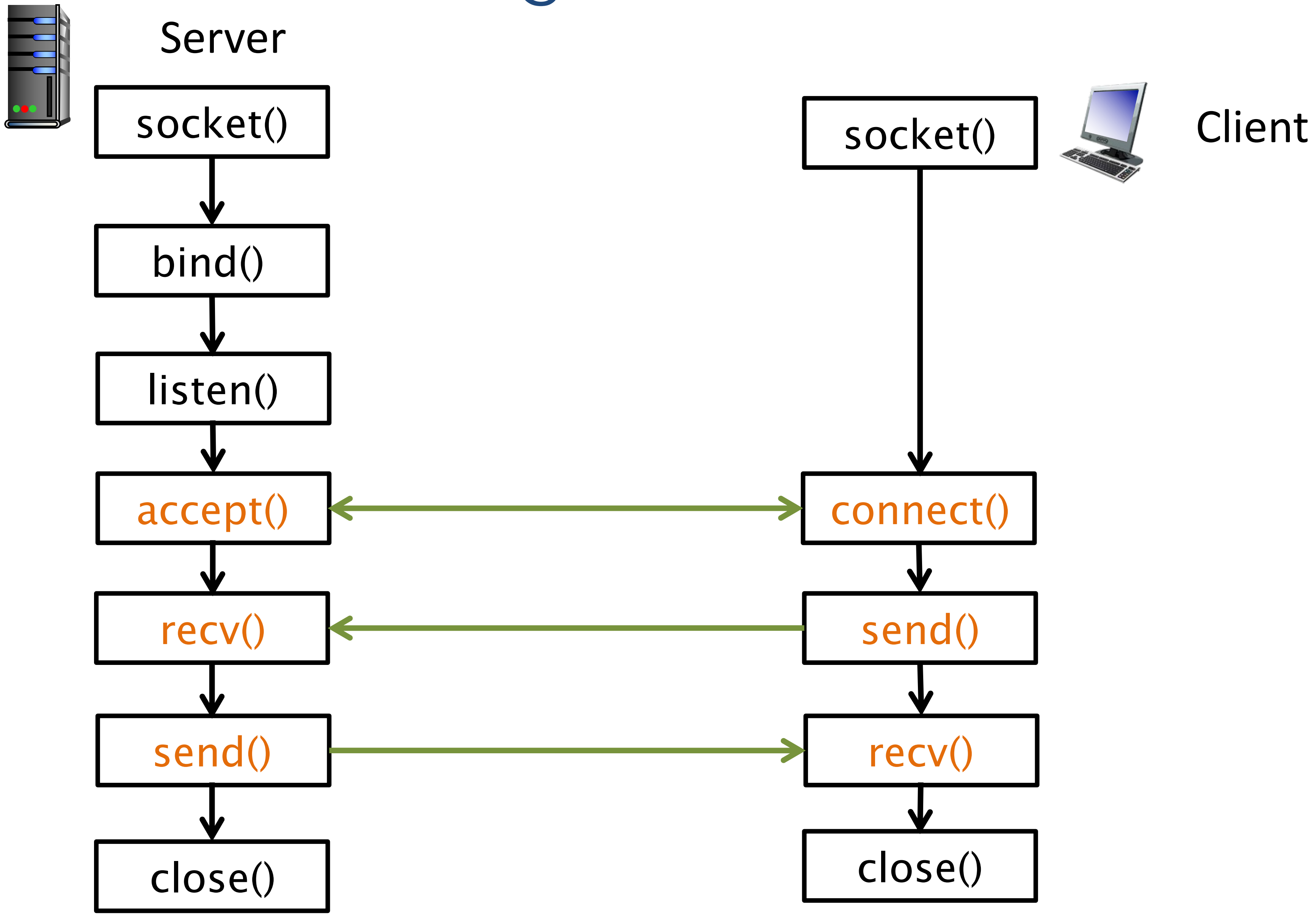
accept: block caller until a connection request arrives

recv: receive some data over a connection

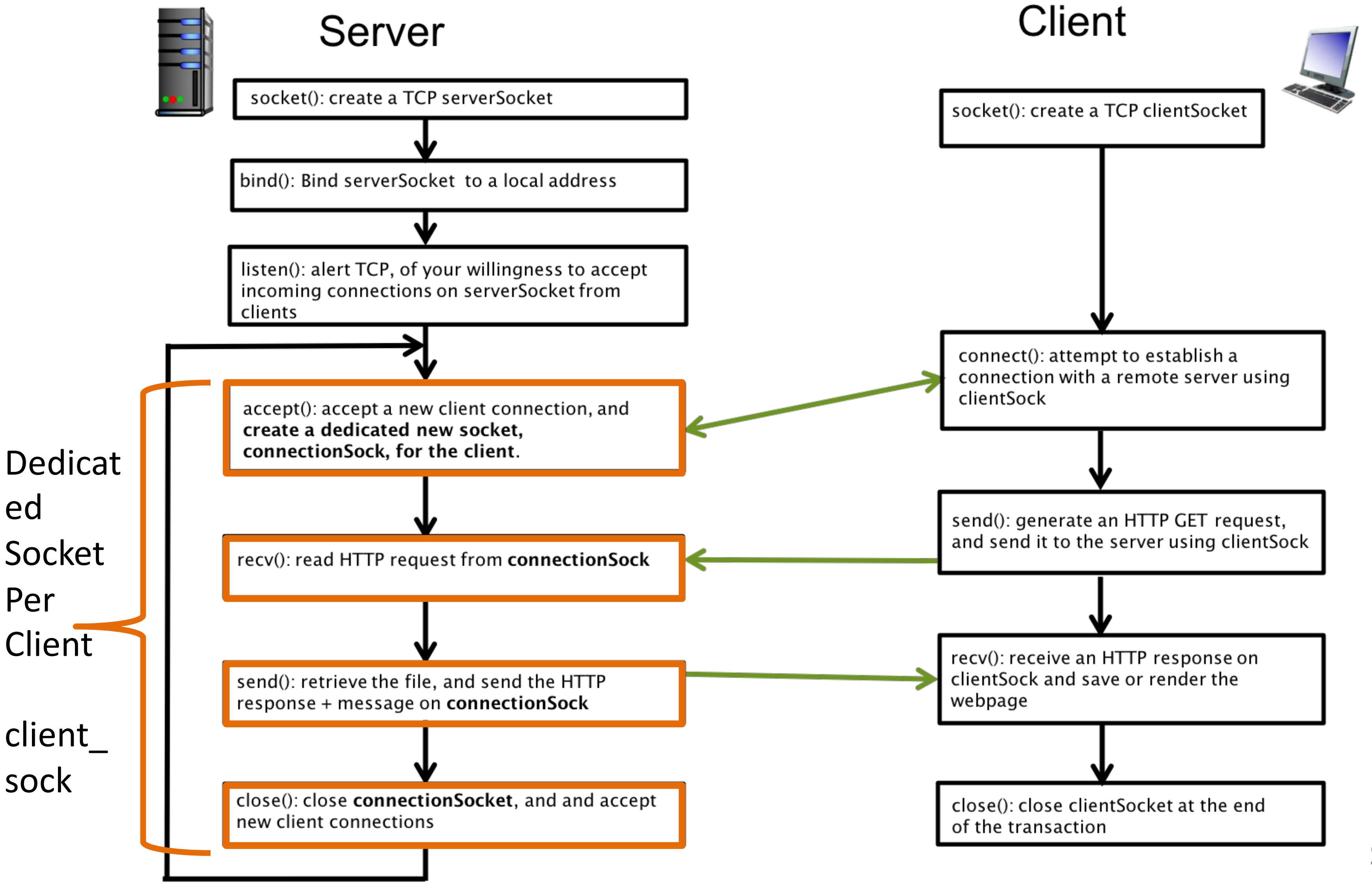
send: send some data over a connection

close: release the connection

Running a Web Server over TCP



Running a Web Server


Server
socket(): create a TCP serverSocket
bind(): Bind serverSocket to a local address
listen(): alert TCP, of your willingness to accept incoming connections on serverSocket from clients
accept(): accept a new client connection, and **create a dedicated new socket, connectionSock, for the client.**
recv(): read HTTP request from **connectionSock**
send(): retrieve the file, and send the HTTP response + message on **connectionSock**
close(): close **connectionSocket**, and and accept new client connections
Client
socket(): create a TCP clientSocket
connect(): attempt to establish a connection with a remote server using clientSock
send(): generate an HTTP GET request, and send it to the server using clientSock
recv(): receive an HTTP response on clientSock and save or render the webpage
close(): close clientSocket at the end of the transaction
Dedicated Socket Per Client
client_sock