CS 43: Computer Networks

05:Network Services and Distributed Systems

September 22, 2020

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)

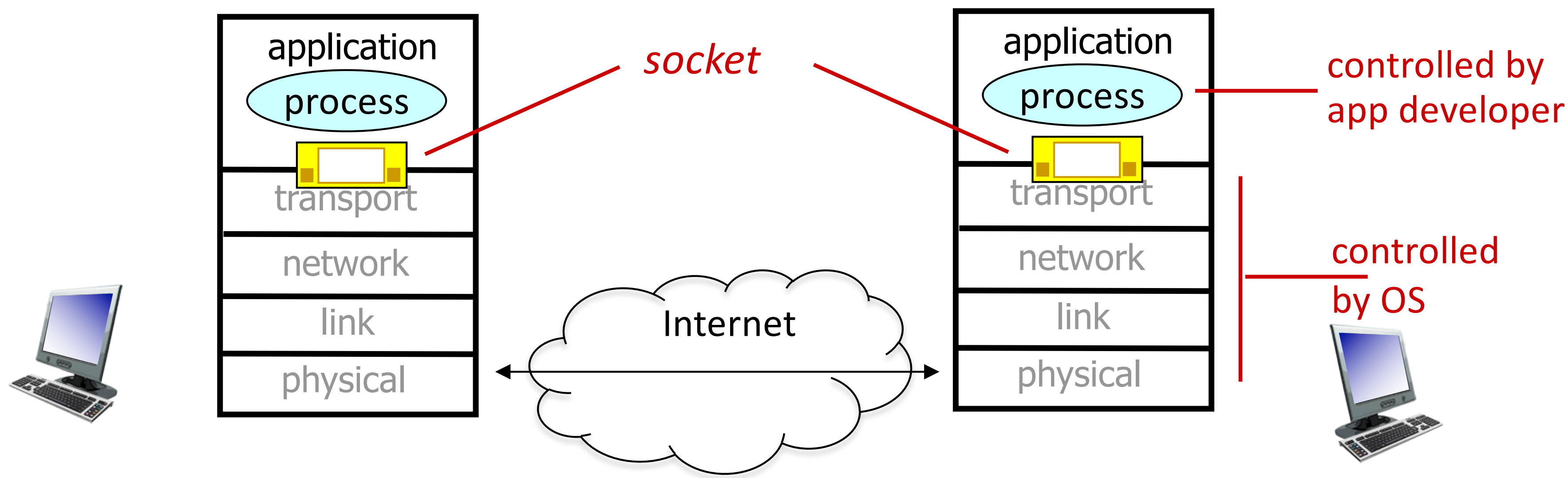
What is a socket?

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.

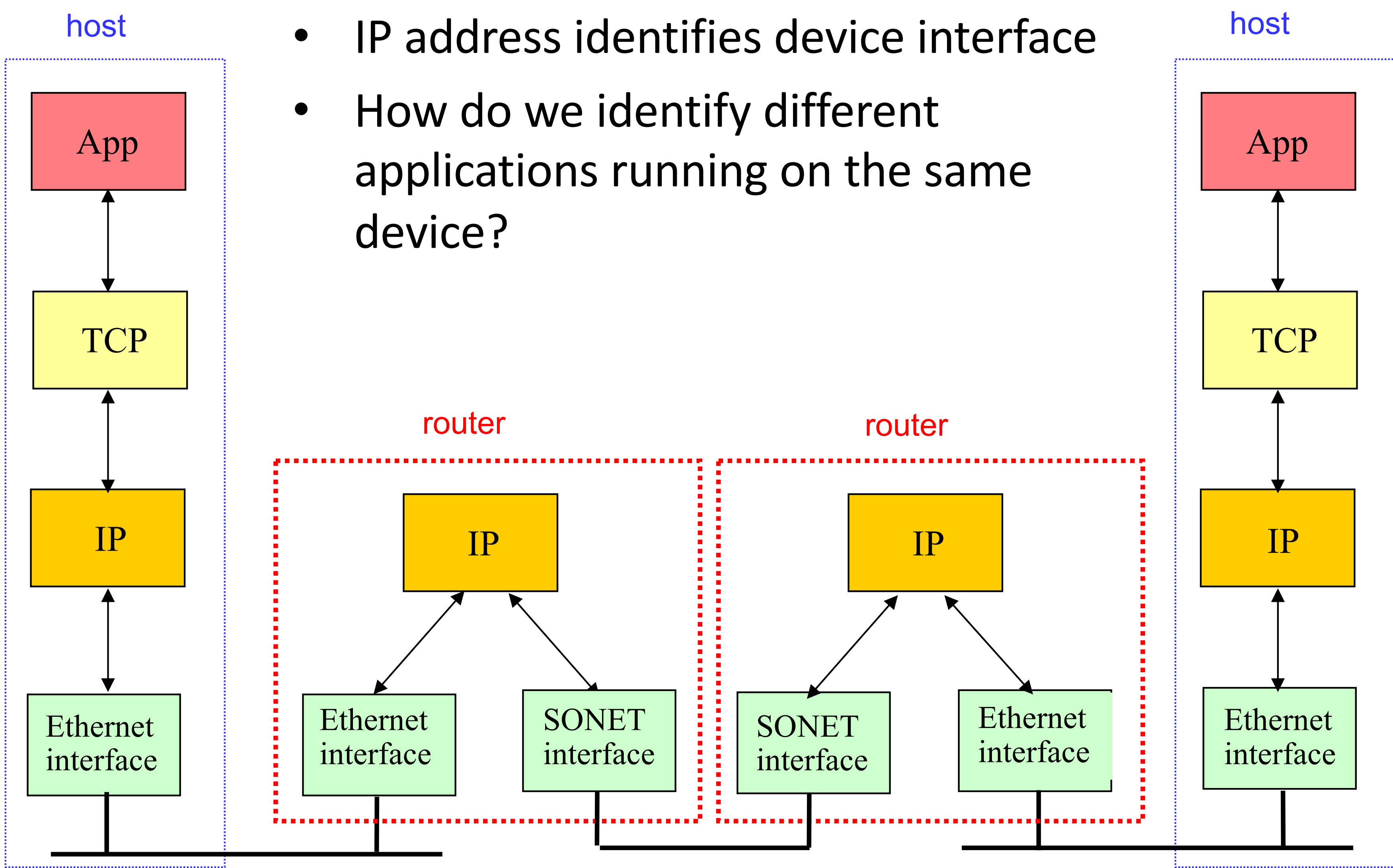
Sockets

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



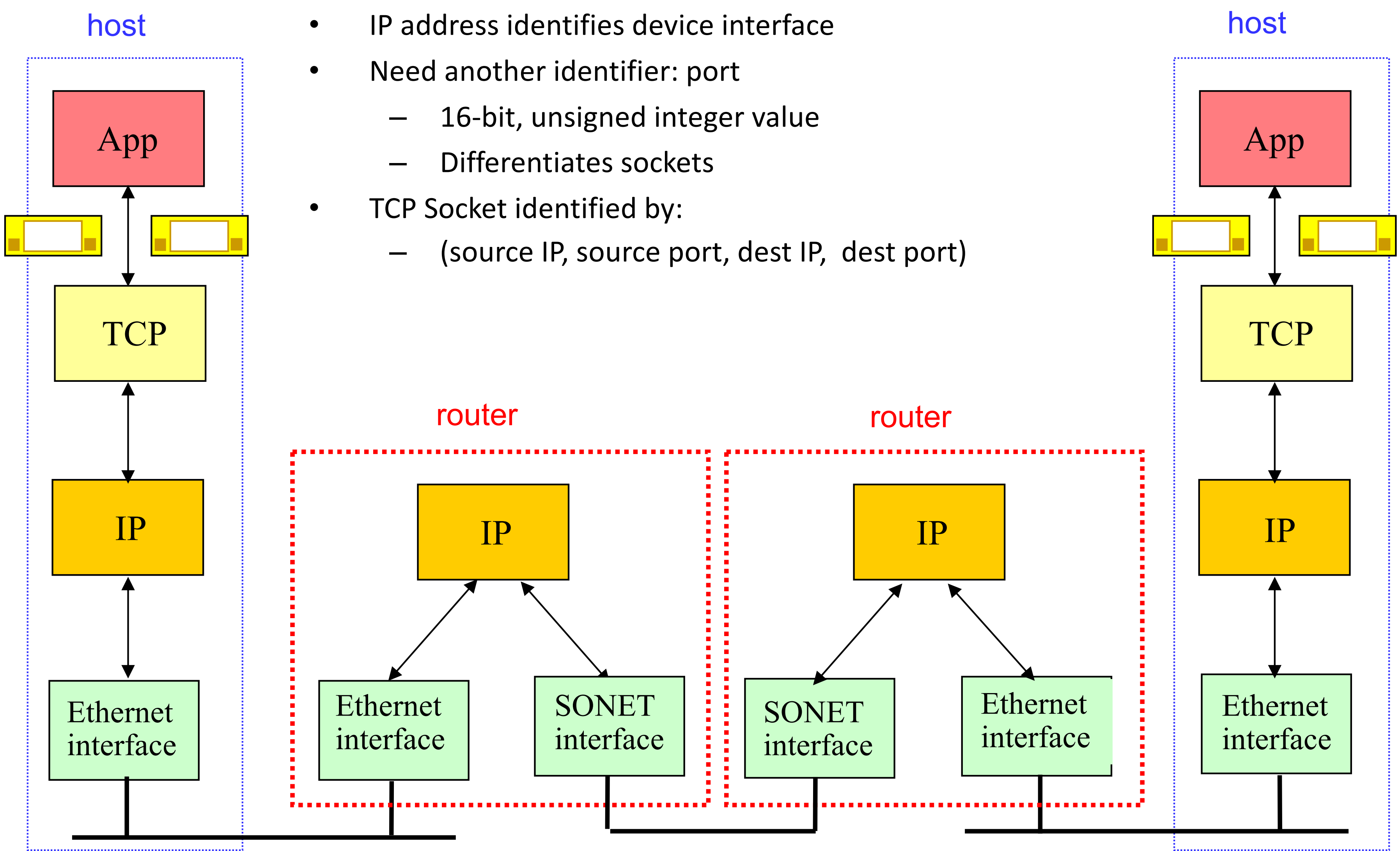
Addressing Sockets

- 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)



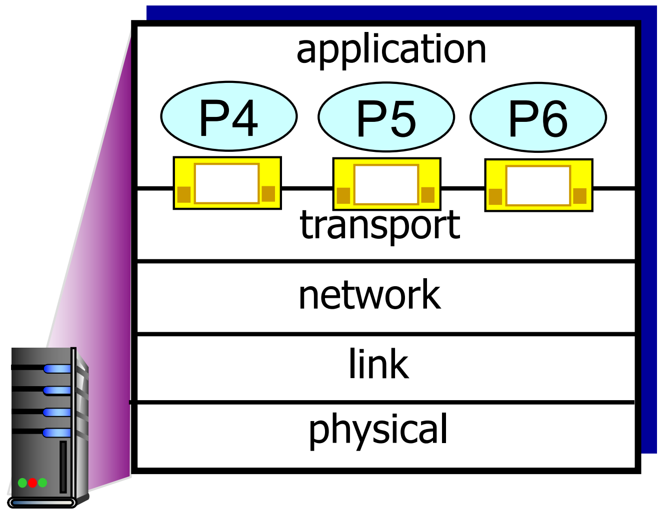
Connection-oriented: example

- 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

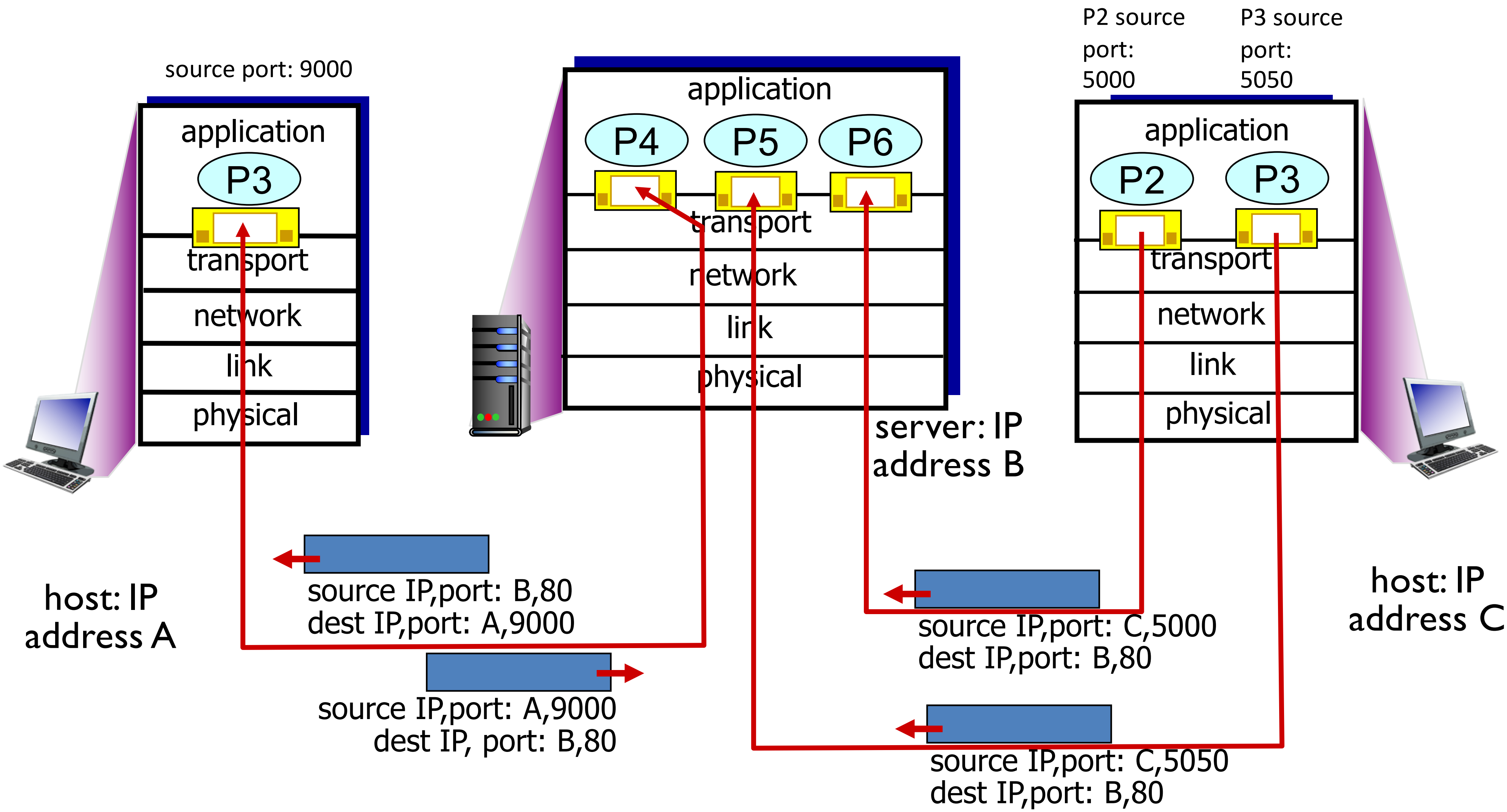
Connection-oriented: HTTP example

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

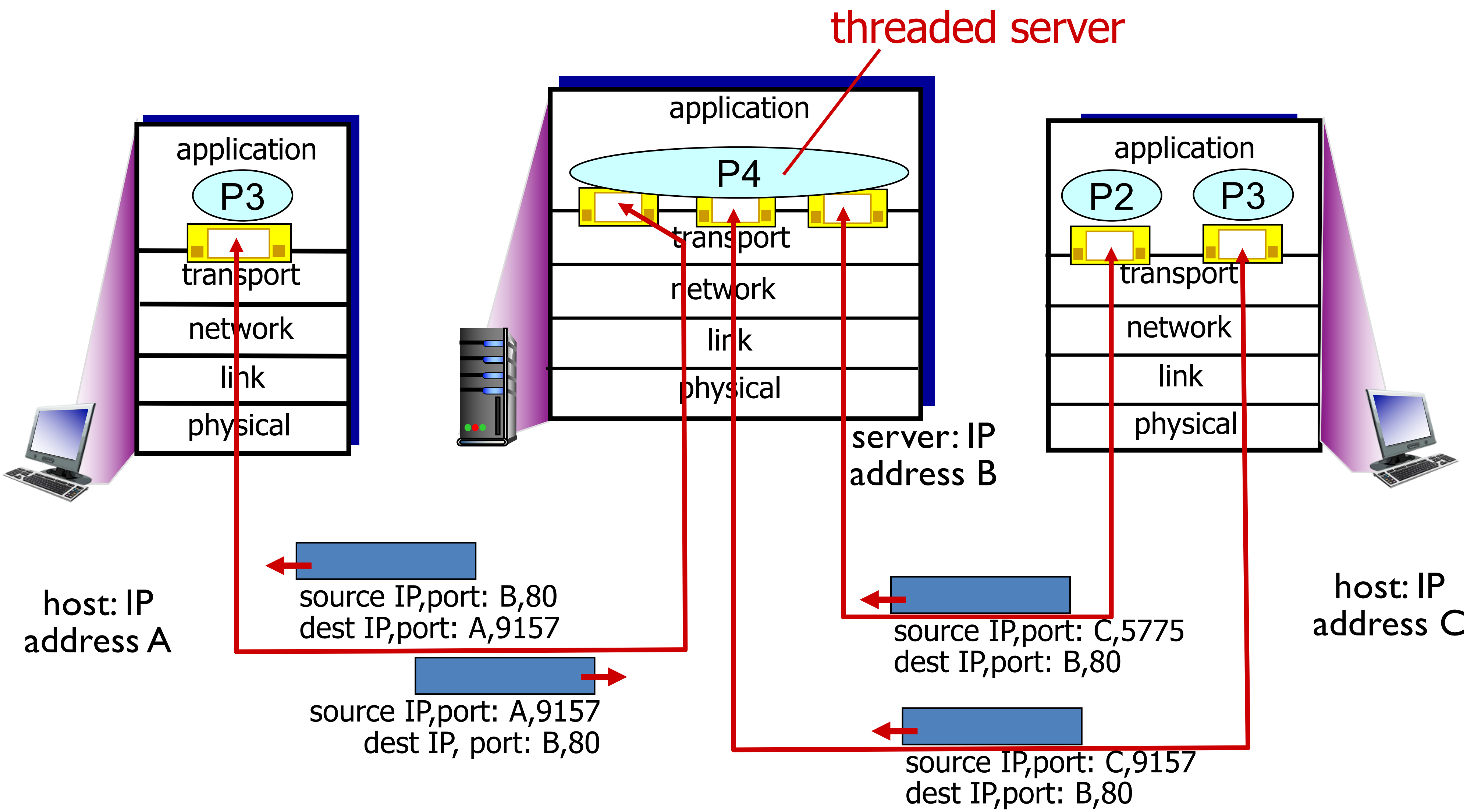


Connection-oriented: HTTP example

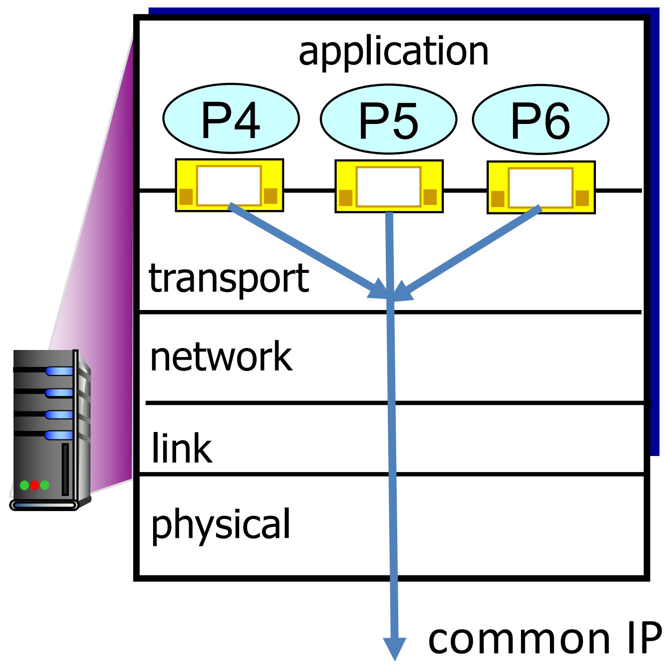
A socket is uniquely identified by (source IP, source port, dest IP, dest port)



Connection-oriented: example



Multiplexing/Demultiplexing



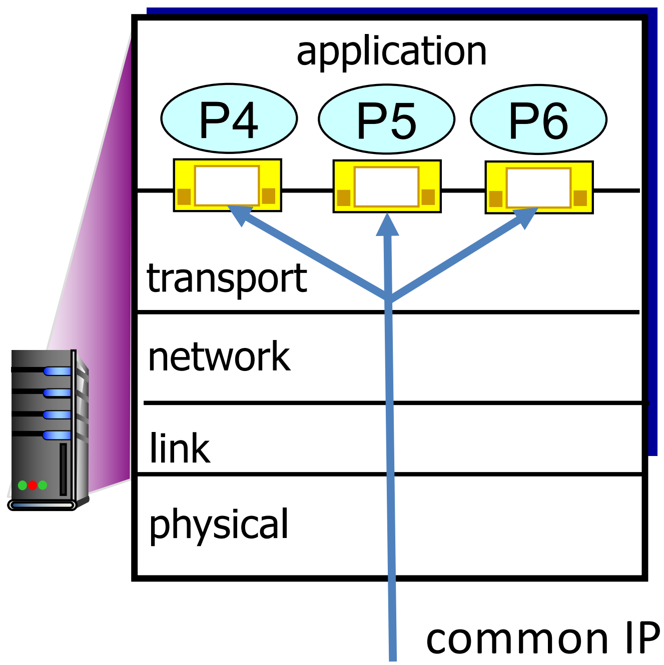
different applications, and sockets

common IP: same end host

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

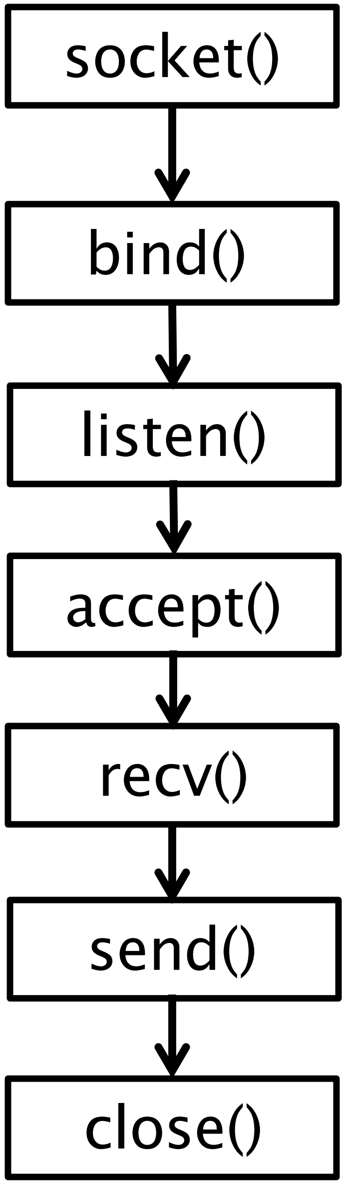
- Client:
  - initiates communication
  - must know the address and port of the server
  - active socket
- Server:
  - passively waits for and responds to clients
  - passive socket

TCP Socket Procedures: Client

Client

socket()	create a new communication endpoint
connect()	actively attempt to establish a connection
send()	send some data over a connection
recv()	receive some data over a connection
close()	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

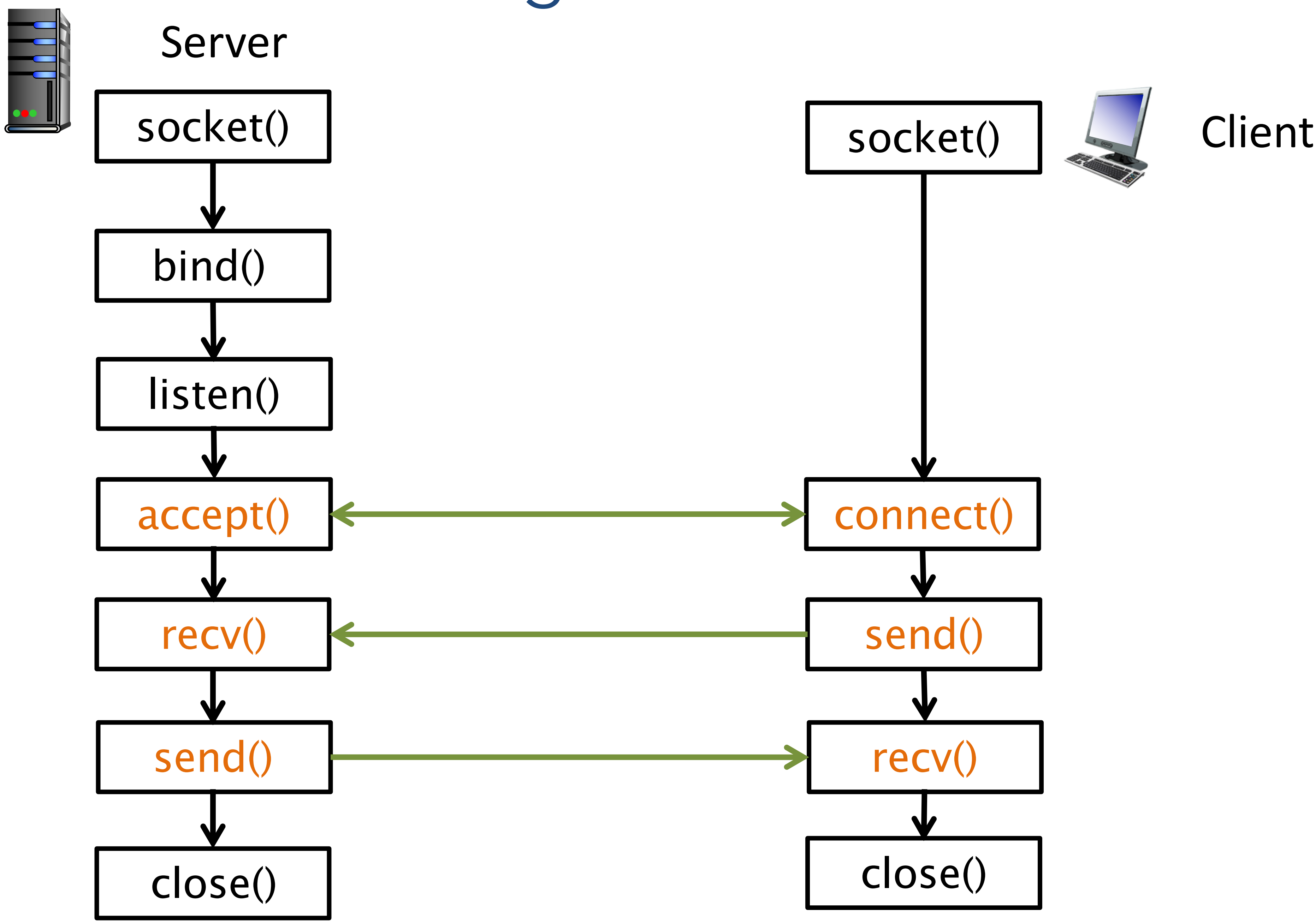
accept: block caller until a connection request arrives

recv: receive some data over a connection

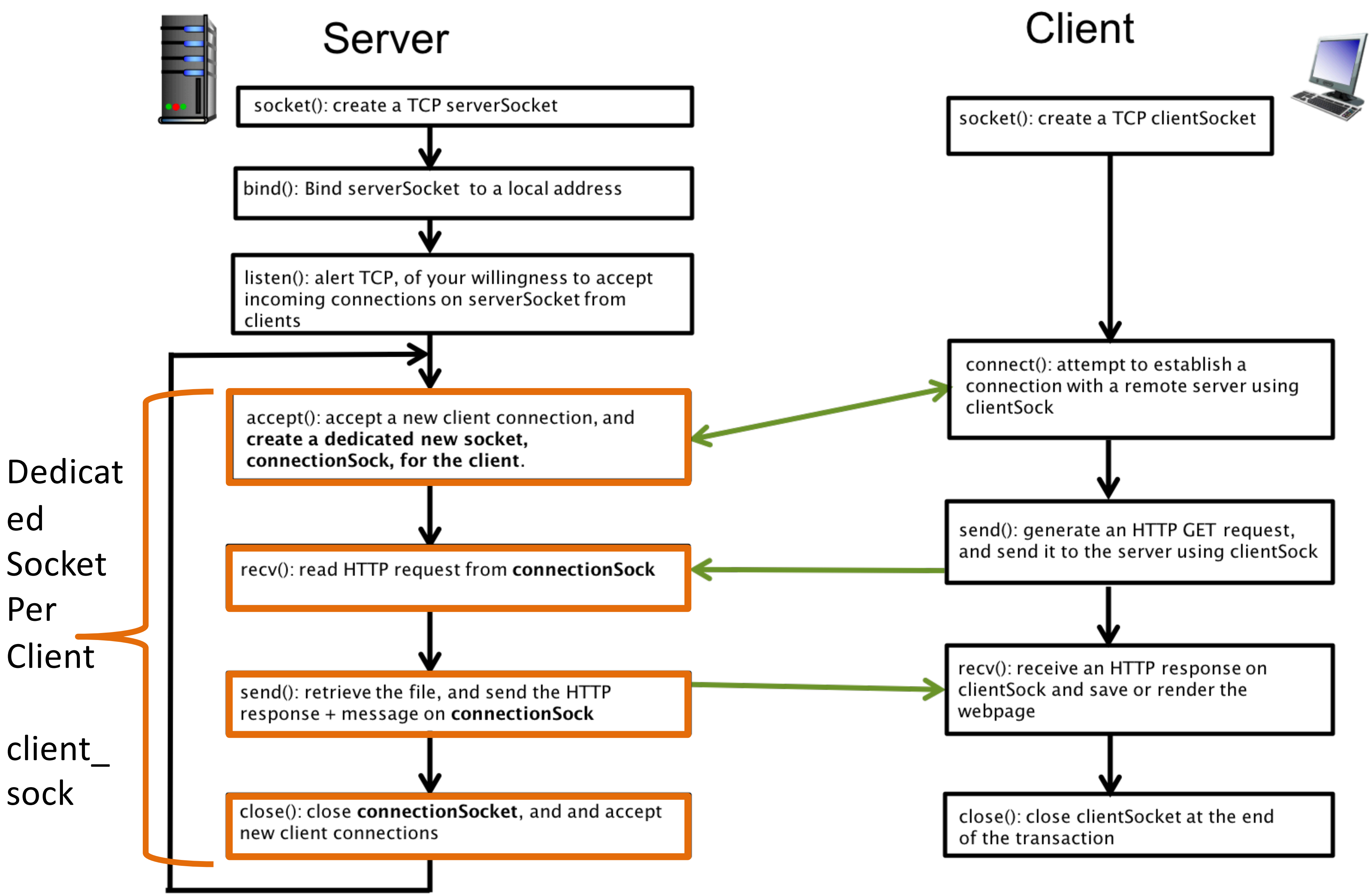
send: send some data over a connection

close: release the connection

Running a Web Server over TCP

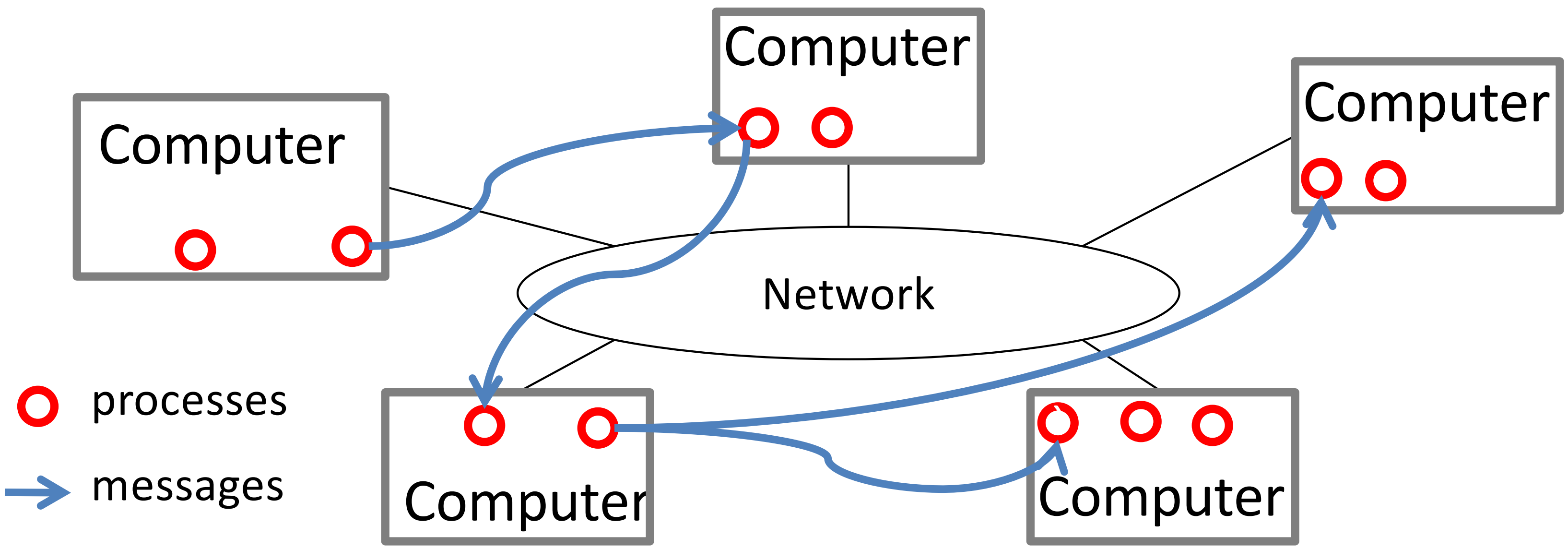


Running a Web Server



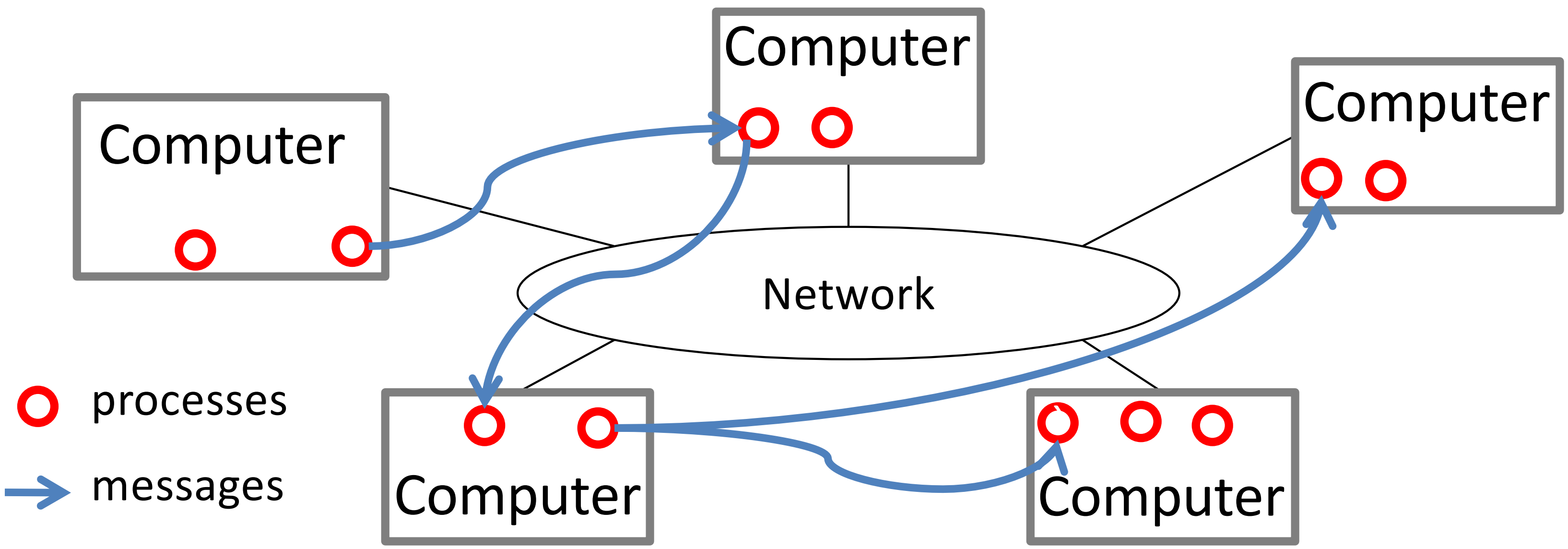
Distributed Network Applications

What is a distributed application?



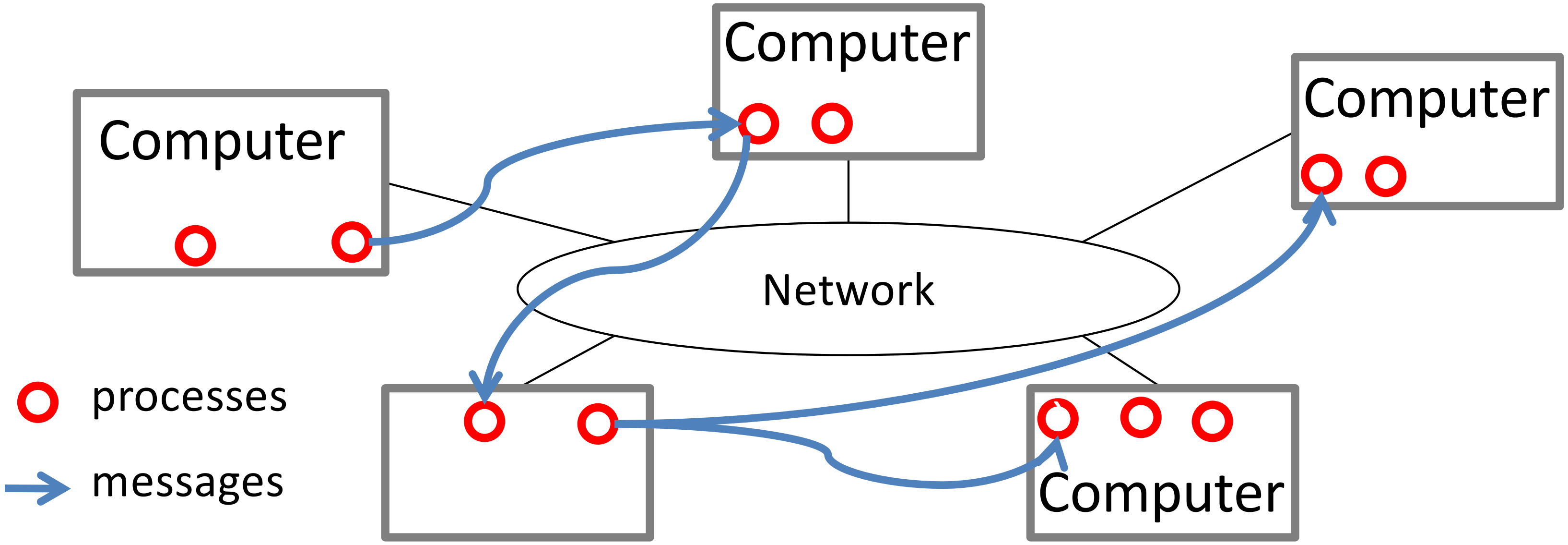
- Cooperating processes in a computer network
- Varying degrees of integration
  - Loose: email, web browsing
  - Medium: chat, Skype, remote execution, remote file systems
  - Tight: process migration, distributed file systems

Distributed Systems: Advantages



- Speed: parallelism, less contention
- Reliability: redundancy, fault tolerance (NSPF)
- Scalability: incremental growth, economy of scale
- Geographic distribution: low latency, reliability

Distributed Systems: Disadvantages



- Fundamental problems of decentralized control
  - State uncertainty: no shared memory or clock
  - Action uncertainty: mutually conflicting decisions
- Distributed algorithms are complex

On a single system…

- You have a number of components
  - CPU
  - Memory
  - Disk
  - Power supply
- If any of these go wrong, you’re (usually) toast.

On multiple systems…

- New classes of failures (**partial failures**).
  - A link might fail
  - One (of many) processes might fail
  - The network might be partitioned

On multiple systems…

- New classes of failures (**partial failures**).
  - A link might fail
  - One (of many) processes might fail
  - The network might be partitioned

Introduces major complexity!

Desirable Properties

- Consistency
  - Nodes agree on the distributed system’s state
- Availability
  - The system is able and willing to process requests
- Partition tolerance
  - The system is robust to network (dis)connectivity

The CAP Theorem

- **C**onsistency
  - Nodes agree on the distributed system's state
- **A**vailability
  - The system is able and willing to process requests
- **P**artition tolerance
  - The system is robust to network (dis)connectivity
- Choose Two
- "CAP prohibits only a tiny part of the design space: perfect availability and consistency in the presence of partitions, which are rare."*

* Brewer, Eric. "CAP twelve years later: How the" rules" have changed." Computer 45.2 (2012): 23-29.

Event Ordering

- It’s very useful if all nodes can agree on the order of events in a distributed system

- For example: Two users trying to update a shared file across two replicas

If two events occur (digitally or in the “real world”), can we always tell which happened first?

A. Yes

B. No

Event Ordering

- It's very useful if all nodes can agree on the order of events in a distributed system

- For example: Two users trying to update a shared file across two replicas

- "Time, Clocks, and the Ordering of Events in a Distributed System" by Leslie Lamport (1978)
  - Establishes causal orderings
  - Cited > 8000 times

Causal Consistency Example

- Suppose we have the following scenario:
  - Sally posts to Facebook, “Bill is missing!”
  - (Bill is at a friend’s house, sees message, calls mom)
  - Sally posts new message, “False alarm, he’s fine”
  - Sally’s friend James posts, “What a relief!”

Causal Consistency Example

- Suppose we have the following scenario:
  - Sally posts to Facebook, “Bill is missing!”
  - Sally’s friend James posts, “What a relief!
- NOT causally consistent:
  - Third user, Henry, sees only:
  - Sally posts to Facebook, “Bill is missing!”
  - Sally’s friend James posts, “What a relief!”

Causal Consistency Example

- Suppose we have the following scenario:
  1. Sally posts to Facebook, “Bill is missing!” (Bill is at a friend’s house, sees message, calls mom)
  2. Sally posts new message, “False alarm, he’s fine”
  3. Sally’s friend James posts, “What a relief!”

- Causally consistent version:
  – Because James had seen Sally’s second post (which caused his response), Henry must also see it prior to seeing James’s.

Summary

- Client-server vs. peer-to-peer models
- Distributed systems are hard to build!
  - Partial failures
  - Ordering of events
- Take CS 87 for more details!