CS 43: Computer Networks
Network Applications

Kevin Webb
Swarthmore College
September 19, 2017

Reading Quiz

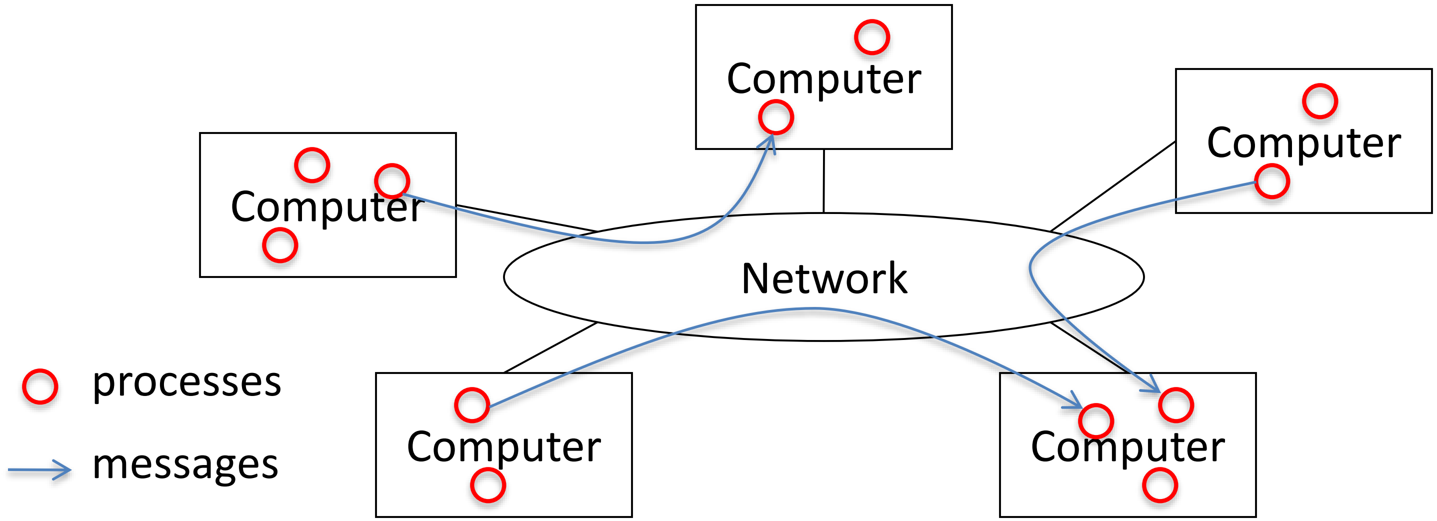
Overview

- Last time: blocking and app structure

- Today: distributed network applications
  - Common models, pros/cons, complexity sources

- Up next:
  - depth into other protocols

What is a distributed application?



- Cooperating processes in a computer network

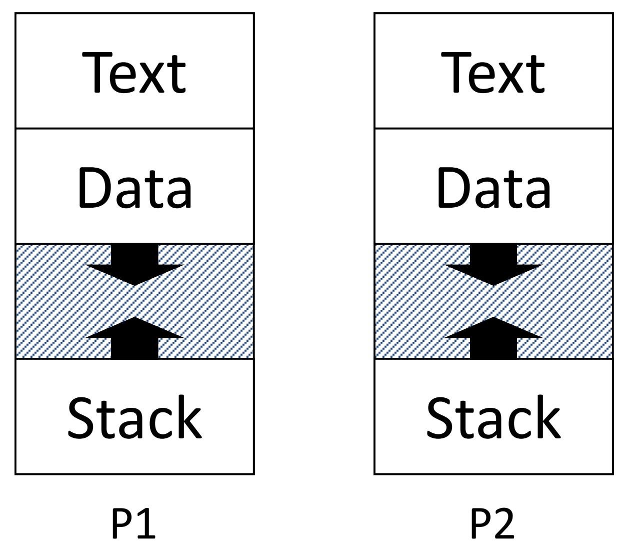
- Varying degrees of integration
  - Loose: email, web browsing
  - Medium: chat, Skype, remote execution, remote file systems
  - Tight: process migration, distributed file systems

Interprocess Communication (IPC)

- In order to cooperate, need to communicate
- Achieved via IPC: interprocess communication
  - ability for a process to communicate with another

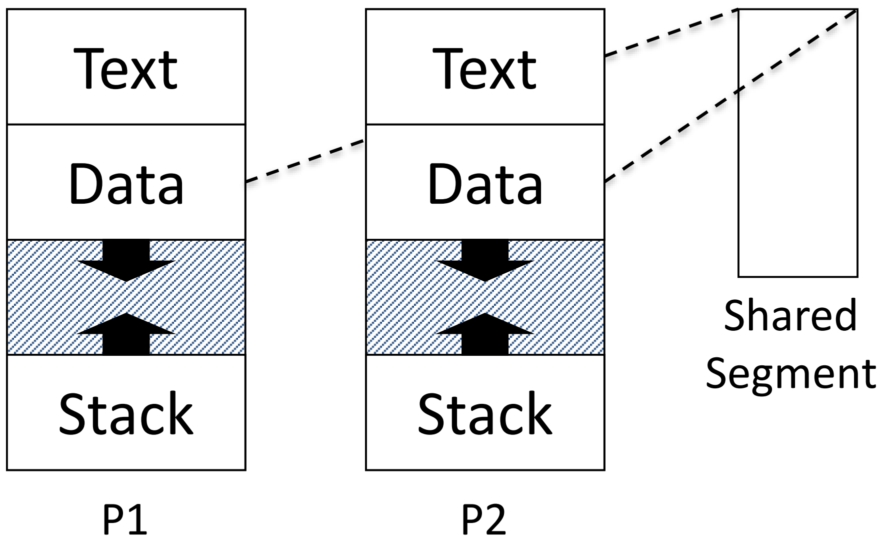
Interprocess Communication (IPC)

- In order to cooperate, need to communicate
- Achieved via IPC: interprocess communication
  - ability for a process to communicate with another
- On a single machine:
  - Shared memory



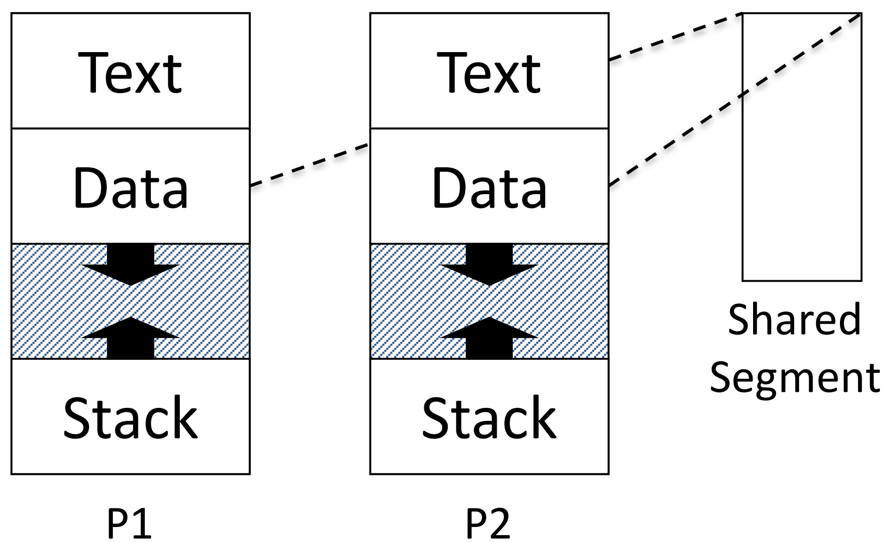
Interprocess Communication (IPC)

- In order to cooperate, need to communicate
- Achieved via IPC: interprocess communication
  - ability for a process to communicate with another
- On a single machine:
  - Shared memory



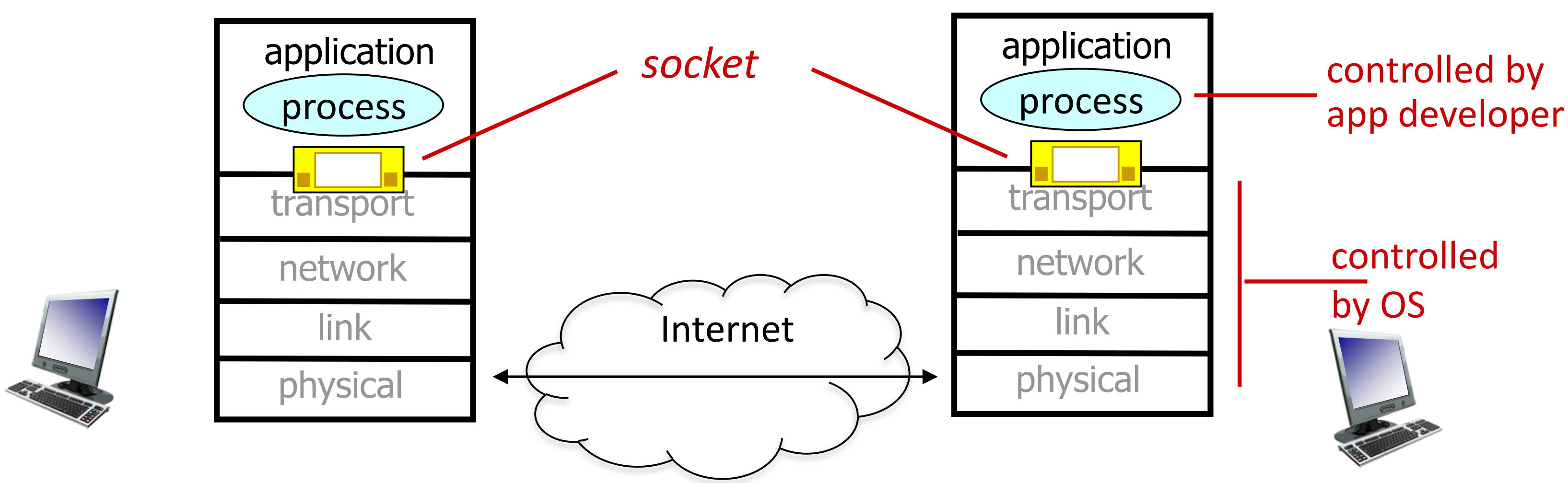
Interprocess Communication (IPC)

- In order to cooperate, need to communicate
- Achieved via IPC: interprocess communication
  - ability for a process to communicate with another
- On a single machine:
  - Shared memory
- Across machines:
  - We need other abstractions (message passing)



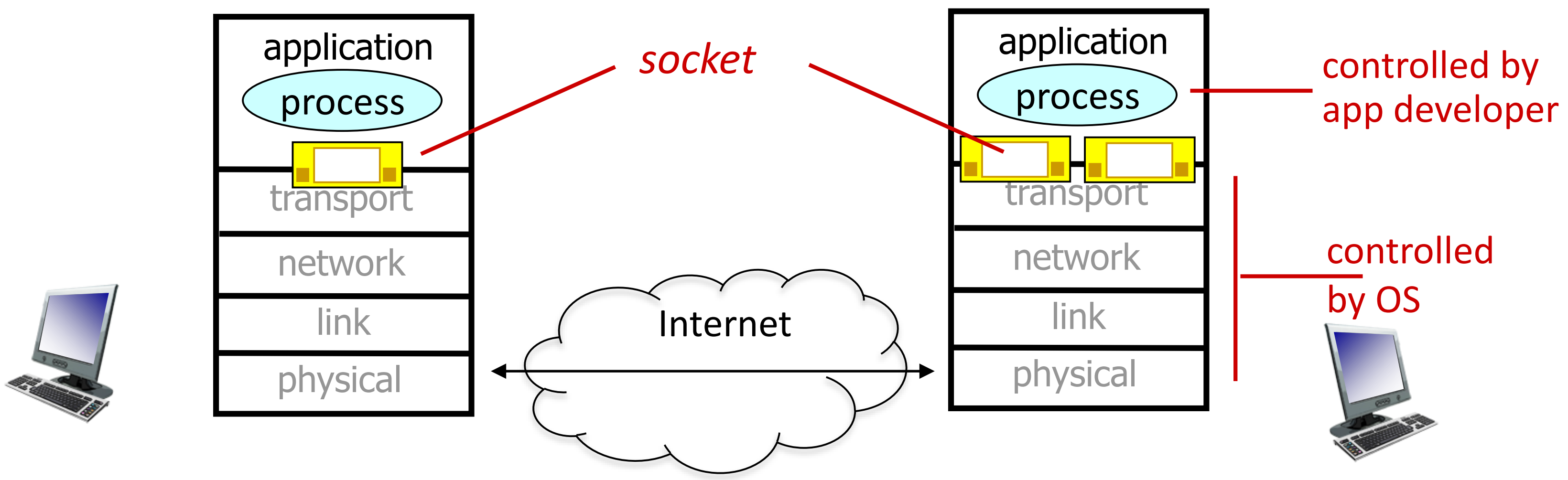
Sockets

- Process sends/receives messages to/from its socket
- Application has a few options, operating system handles the details
  - Choice of transport protocol (TCP, UDP, ICMP, SCTP, etc.)
  - Transport options (TCP: maximum segment size, delayed sends)



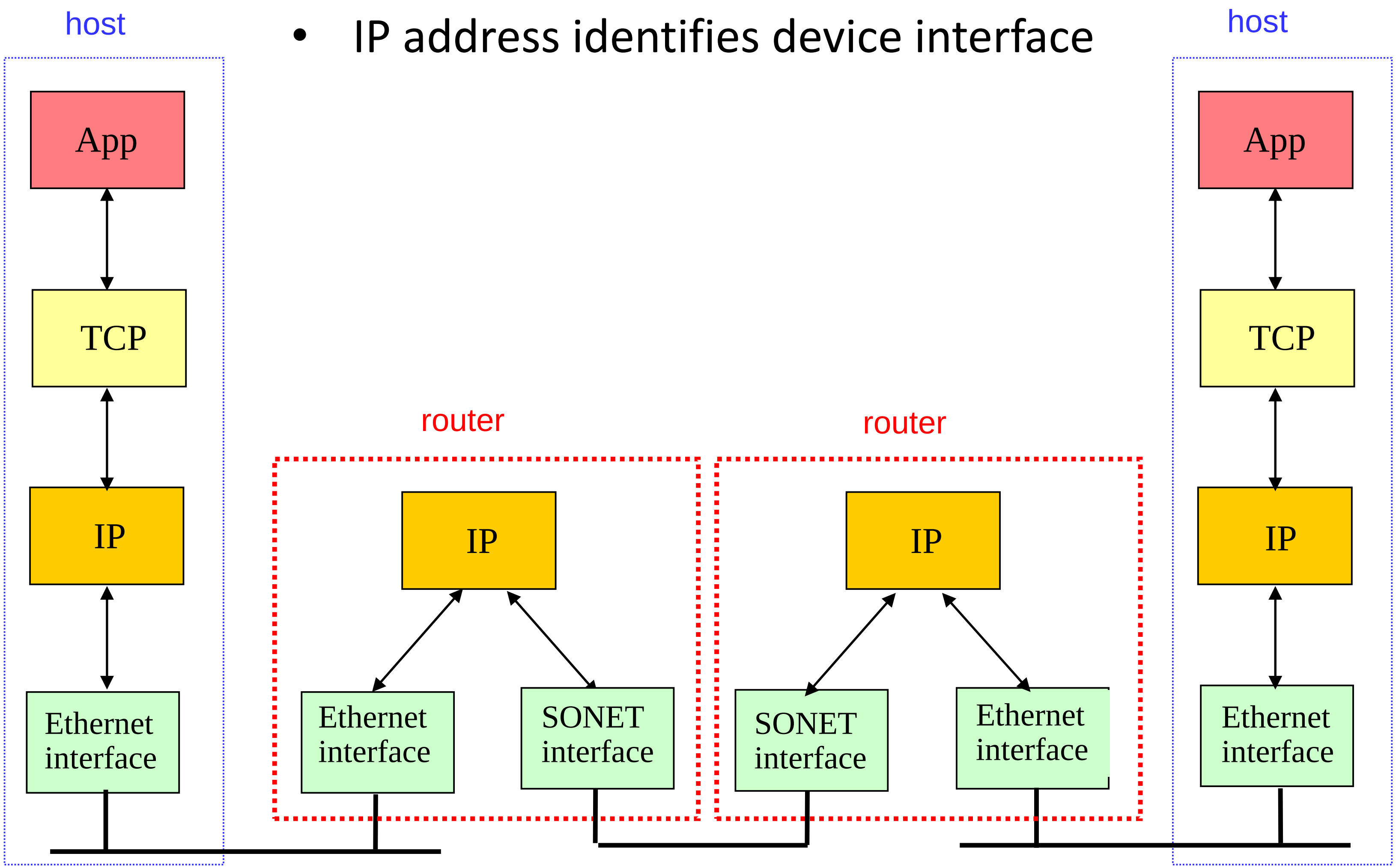
Sockets

- Process sends/receives messages to/from its socket
- Application has a few options, operating system handles the details
  - Choice of transport protocol (TCP, UDP, ICMP, SCTP, etc.)
  - Transport options (TCP: maximum segment size, delayed sends)
- Must identify which socket we're addressing

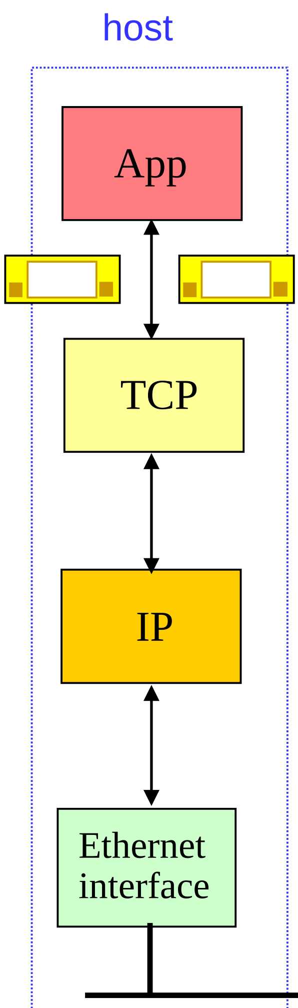


Addressing Sockets

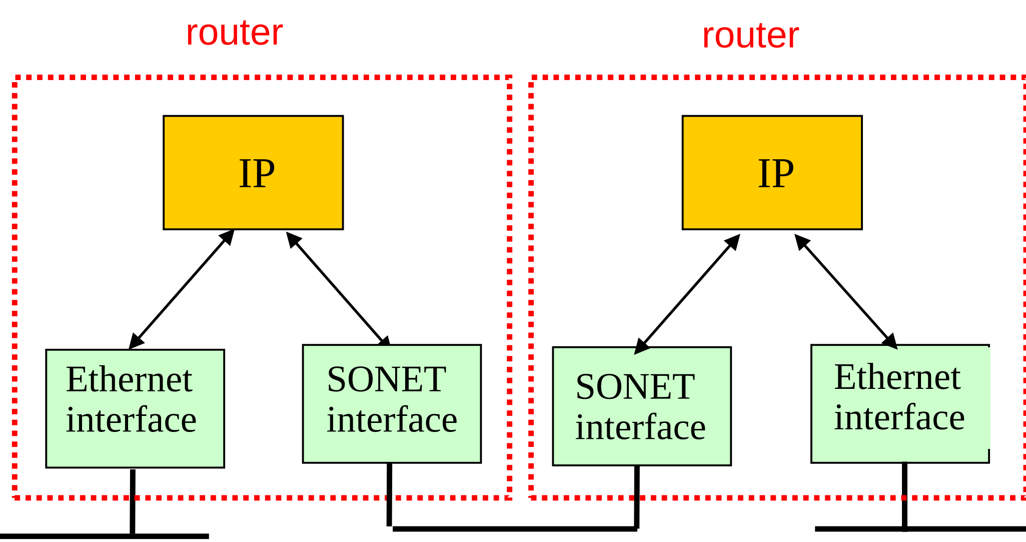
- IP address identifies device interface

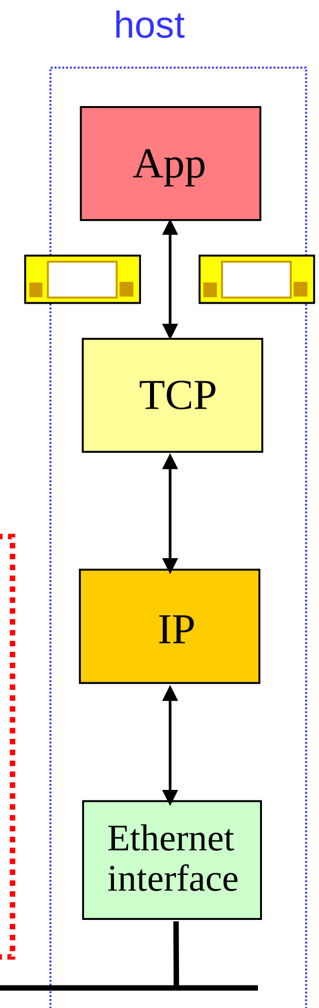


Addressing Sockets

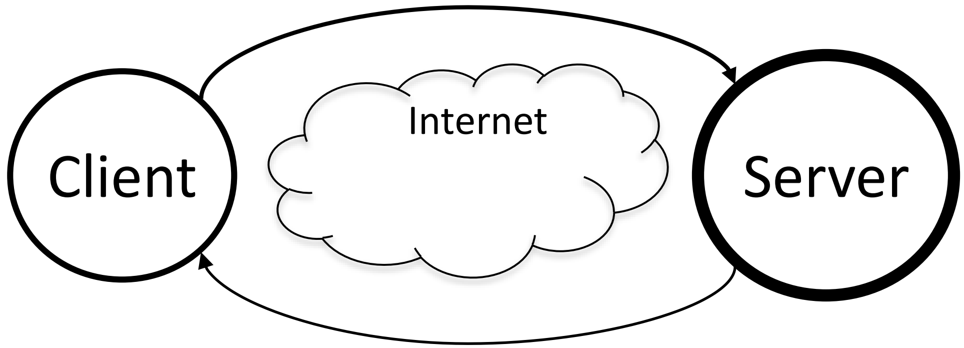


- IP address identifies device interface
- Need another identifier: port
  - 16-bit, unsigned integer value
  - Differentiates sockets





The Client/Server Model



- Client
  - Short-lived process that makes requests
  - “User-side” of application
  - Initiates the communication (often via *connect()*)
- Server
  - Exports well-defined requests/response interface
  - Long-lived process that waits for requests
  - Upon receiving request, carries it out (may spawn processes)

Client versus Server

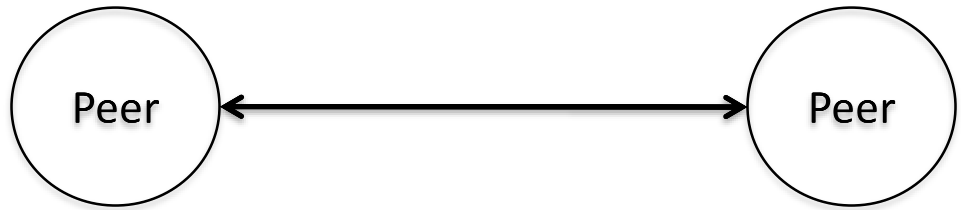
Server:

- always-on host
- permanent (IP) address (rendezvous location)
- static port conventions (http:80, email:25, ssh:22)
- data centers for scaling
- may communicate with other servers to respond

Clients:

- may be intermittently connected
- may have dynamic (IP) addresses
- do not communicate directly with each other

Peer-to-Peer



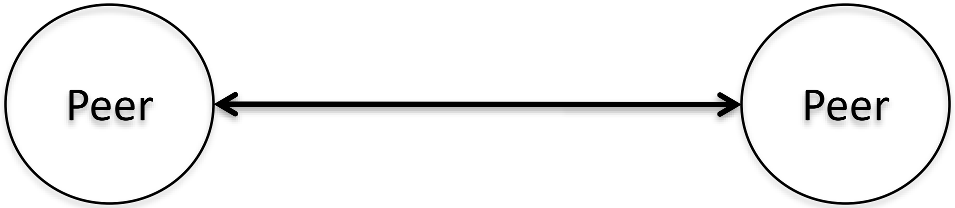
- A peer talks directly with another peer
  - No permanent rendezvous involved
  - Symmetric responsibility (unlike client/server)
- Often used for:
  - File sharing (Napster, BitTorrent)
  - Games
  - “NoSQL” data retrieval
  - In general: “distributed systems”

In a peer-to-peer architecture, are there clients and servers?

A. Yes

B. No

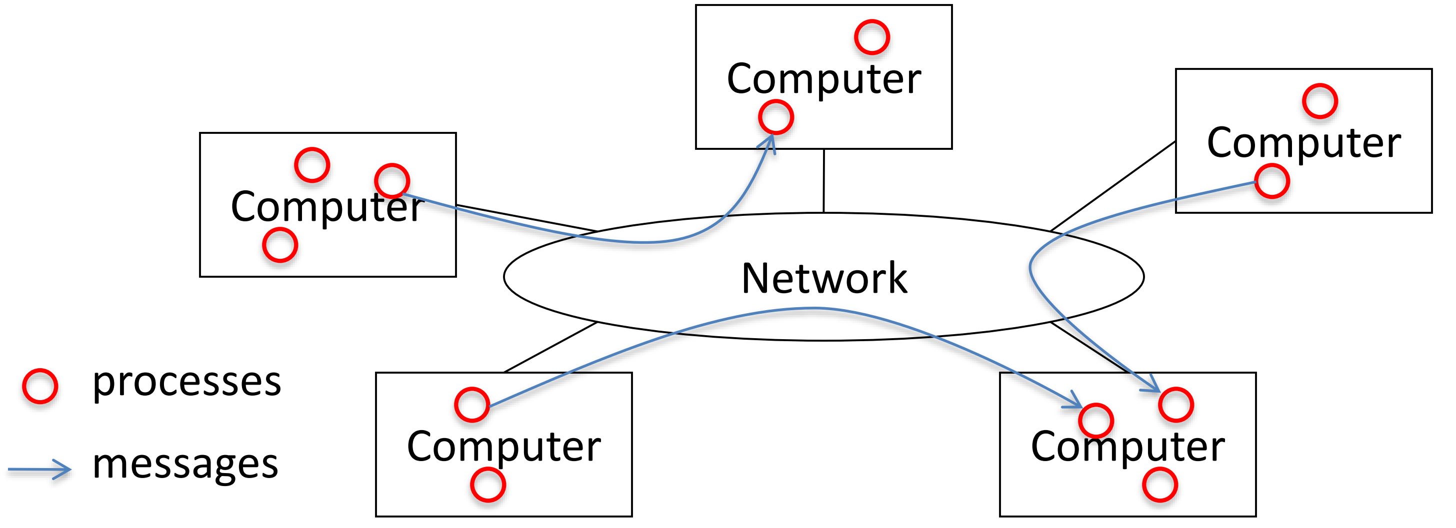
Peer-to-Peer



- (+) Peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands

- (-) Complex management, difficult problems

Advantages

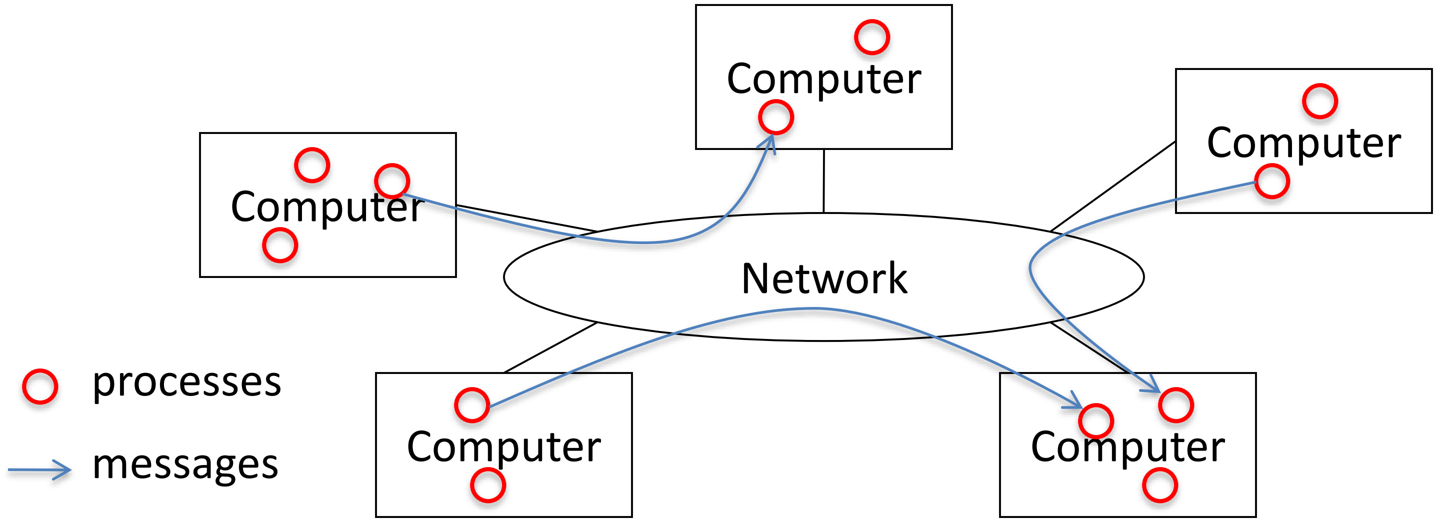


- Speed: parallelism, less contention
- Reliability: redundancy, fault tolerance
- Scalability: incremental growth, economy of scale
- Geographic distribution: low latency, reliability

If one machine can process requests at a rate of X per second, how quickly can two machines process requests?

A. Slower than one machine (<X)
B. The same speed (X)
C. Faster than one machine, but not double (X-2X)
D. Twice as fast (2X)
E. More than twice as fast(>2X)

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!

If a process sends a message, can it tell the difference between a slow link and a delivery failure?

If a process sends a message, can it tell the difference between a slow link and a delivery failure?

A. Yes

B. No

What should we do to handle a partial failure? Under what circumstances, or what types of distributed applications?

A. If one process fails or becomes unreachable, switch to a spare.
B. Pause or shut down the application until all connectivity and processes are available.
C. Allow the application to keep running, even if not all processes can communicate.
D. Handle the failure in some other way.

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.

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

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

A. Yes

B. No

"Relativity of simultaneity"

- Example: observing car crashes
- Exception: causal relationship

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

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!