CS 352
Sockets, App Layer, DNS

Lecture 3

http://www.cs.rutgers.edu/~sn624/352-F22

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Review of concepts

- Switching: Circuit, Message, Packet
- Layering: Modularity

Protocol: Message format and actions

Protocols can be public domain (RFCs) or proprietary.

Today’s lecture

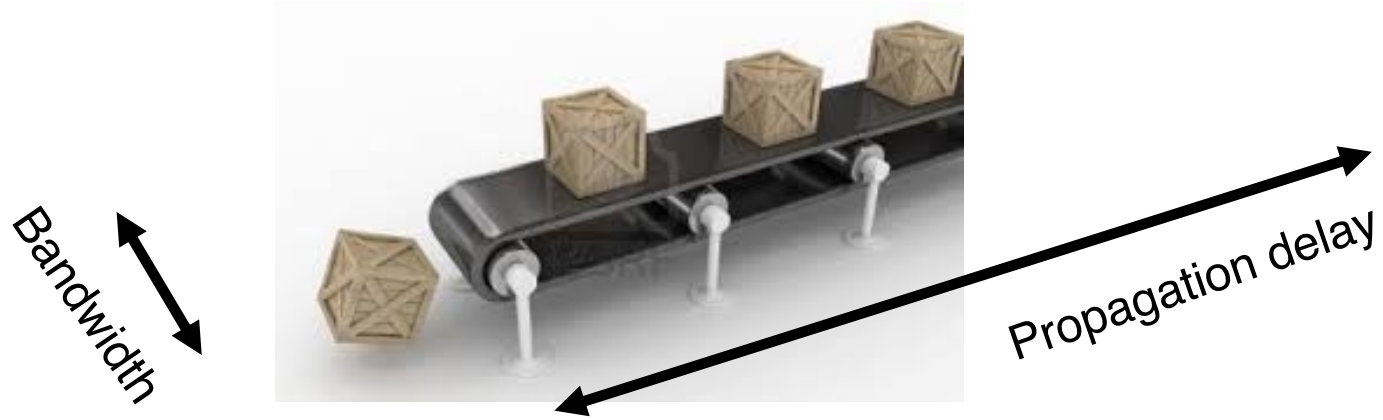
- Understand how to measure the Internet
- Learn how application software accesses the Internet
- Dive into our first app-layer protocol

Measuring Networks (including the Internet)

Some definitions

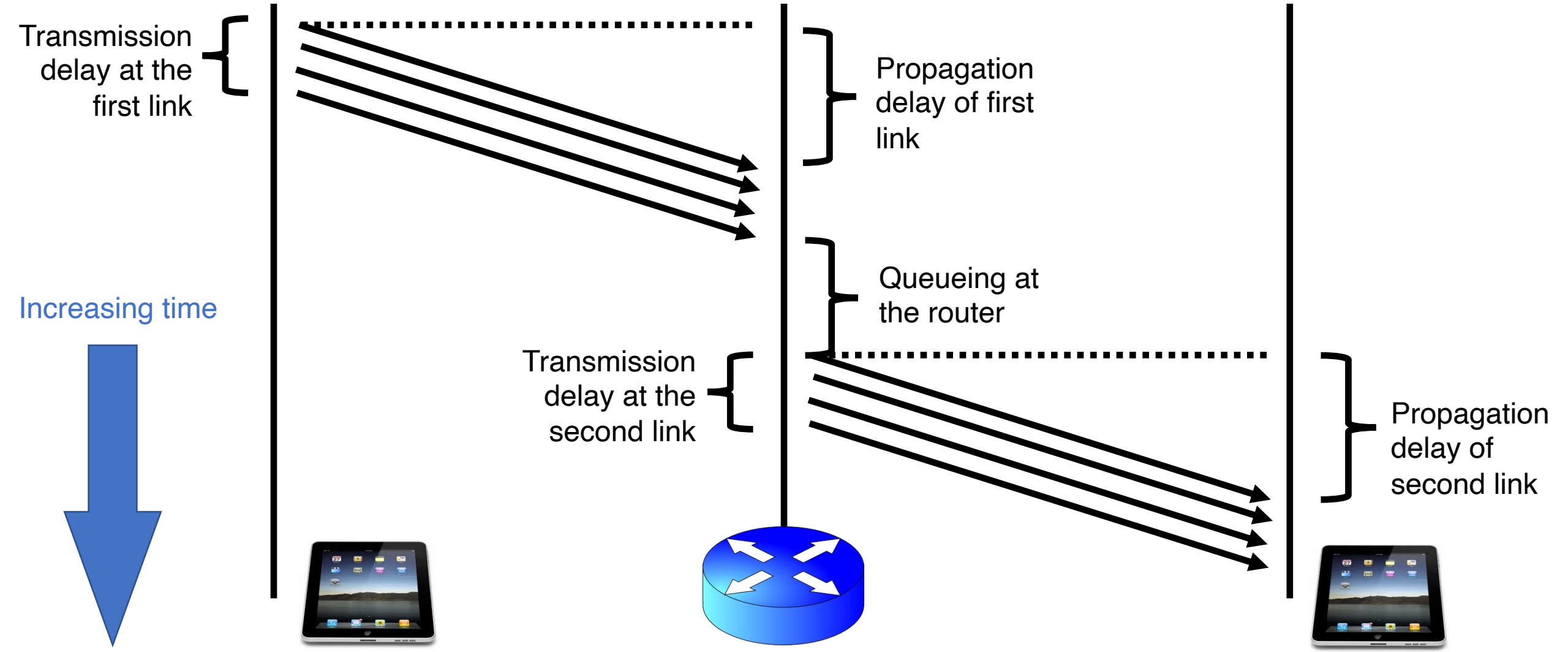
- Packet size: length of a packet (bits or bytes), incl. header and data
- Bandwidth: For a single link, amount of data it can transmit per unit time (bits/second or Bytes/second or packets/second)
- Propagation delay: Time needed to move one bit across (second)
  - Imposed by the communication medium; depends on the link “length”
- Transmission delay: Time from first bit@sender to last bit@sender
  - Determined by link bandwidth and packet size
- Queueing delay: Time that a packet waits for transmission
  - Determined by contention for the link
- Total packet delay: time from first bit@sender to last bit@receiver
  - propagation delay + queueing delay + transmission delay for a single packet

An analogy: Conveyor belt



- Propagation delay = time for first box to travel the length of the belt
- Bandwidth = the number of boxes put on the belt per minute (“rate”)
- Suppose we have N boxes in one shipment
- Shipment transmission time = N / rate
  - The next box is put on the belt (1/rate) minutes after the last
- Total transfer time = transmission time + propagation delay

Visualizing the components of delay



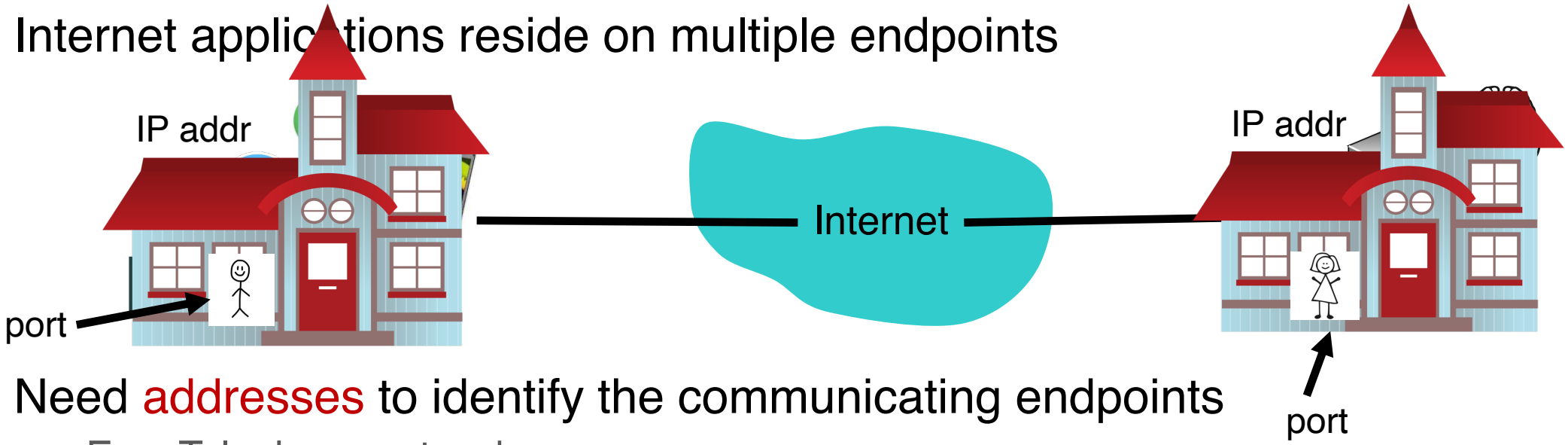
Bandwidth and delay demo

- Throughput (related to bandwidth)
  - `iperf -s` # at the destination
  - `iperf -c <destination>` # at the source,
  - e.g., `iperf -c localhost`
- (total) delay
  - `ping <destination>`
  - e.g., `ping google.com`
- (Don’t just watch; you can try it!)

Application Layer

App-layer communication

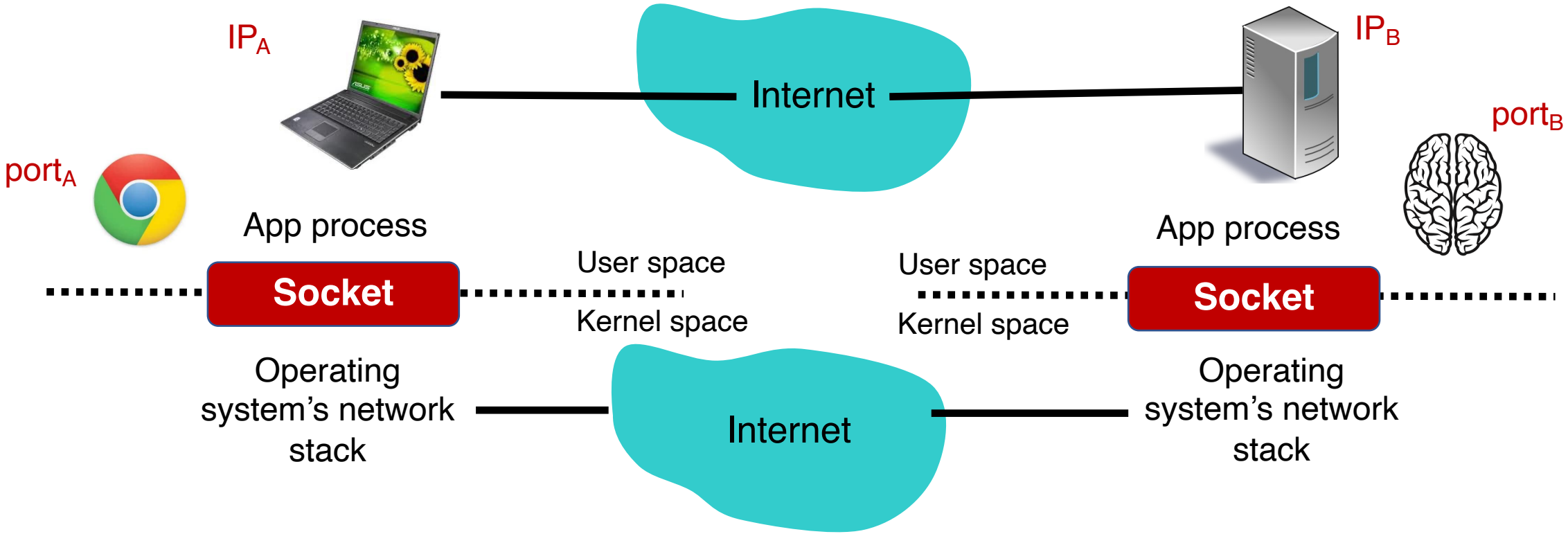
- Internet applications reside on multiple endpoints



- Need addresses to identify the communicating endpoints
  - E.g., Telephone network: xxx-yyy-zzzz
- Internet: Internet Protocol (IP) addresses
  - IPv4 (32 bits) 128.6.24.78
  - IPv6 (128 bits) 2001:4000:A000:C000:6000:B001:412A:8000
- Which app on each endpoint? Port number

How are addresses used?

- Socket: abstraction (API) of the Internet for applications



App-layer connection is a 4-tuple: (IP_A, $port_A$, IP_B, $port_B$)

Socket system calls

Process A

(client)

process

socket

$IP_A + port_A$

`connect(`

IP_B, $port_B$`)`

`send()`

Process B

(server)

process

socket

$IP_B + port_B$

`bind(IPaddr`$_B$`, port`$_B$`)`

`listen()`

`accept()`

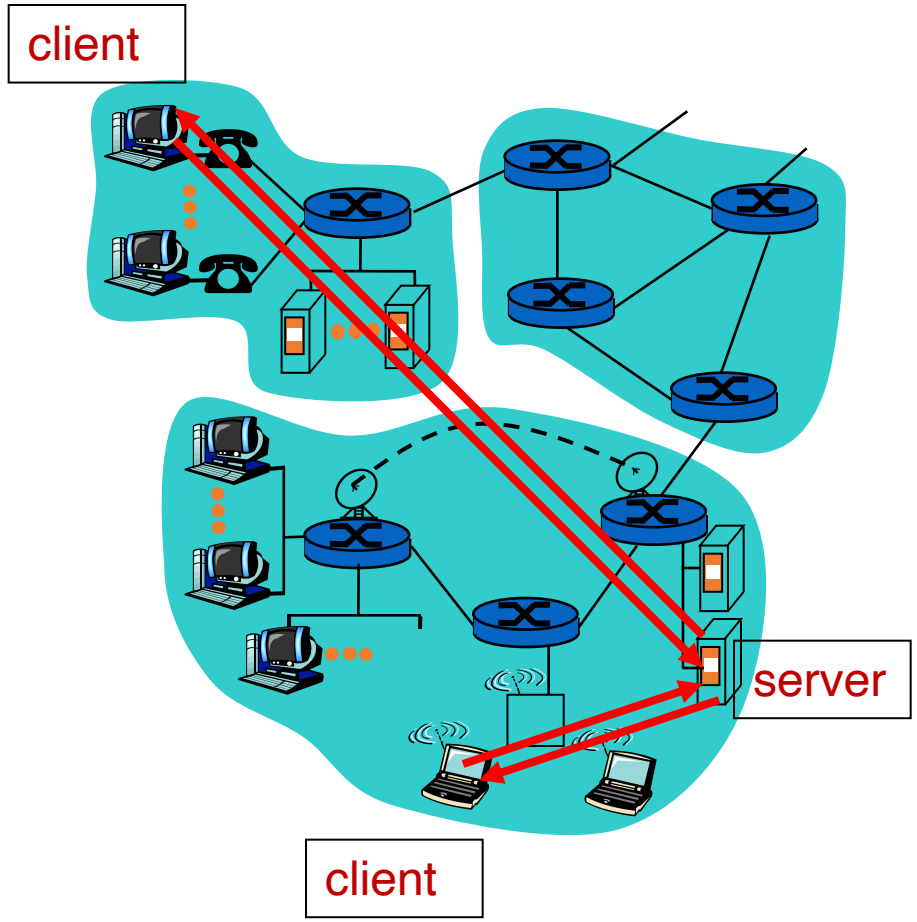
`recv()`

Seeing app-layer connections

- `netstat`
- `ss`

Common Architectures of Applications

Client-server architecture



Server:

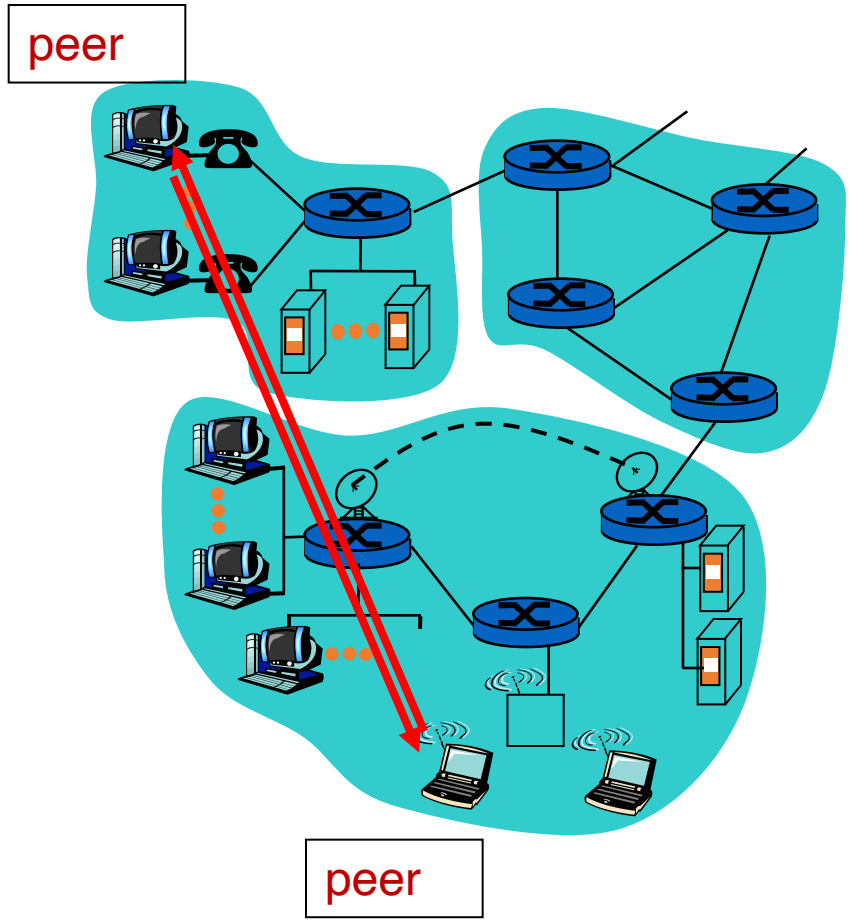
- Always-on endpoint
- Provides a “service” to the world
- Typically, a permanent IP address
- Compute clusters to scale to many users

Clients:

- A “customer” of the server
- May be intermittently connected
- May have dynamic IP addresses
- Typically, do not communicate directly with other clients

- The web and most mobile apps use a client-server architecture

Peer-to-peer (P2P) architecture



- Peers:
  - Intermittently connected hosts
  - Directly talking to each other
- Little to no reliance on always-up servers
  - Examples: BitTorrent
- Today, many applications use a hybrid model
  - Example: (webRTC) Google meet, FB messenger, …

Going forward: A few app-layer protocols

- Domain Name System
- The web
- Mail
- Streaming video

Domain Name System

You have my name. Can you lookup my address?

Domain Name System (DNS)

- Problem statement:
  - Average brain can easily remember 7 digits for a few names
  - On average, IP addresses have 12 digits
  - We need an easier way to remember IP addresses

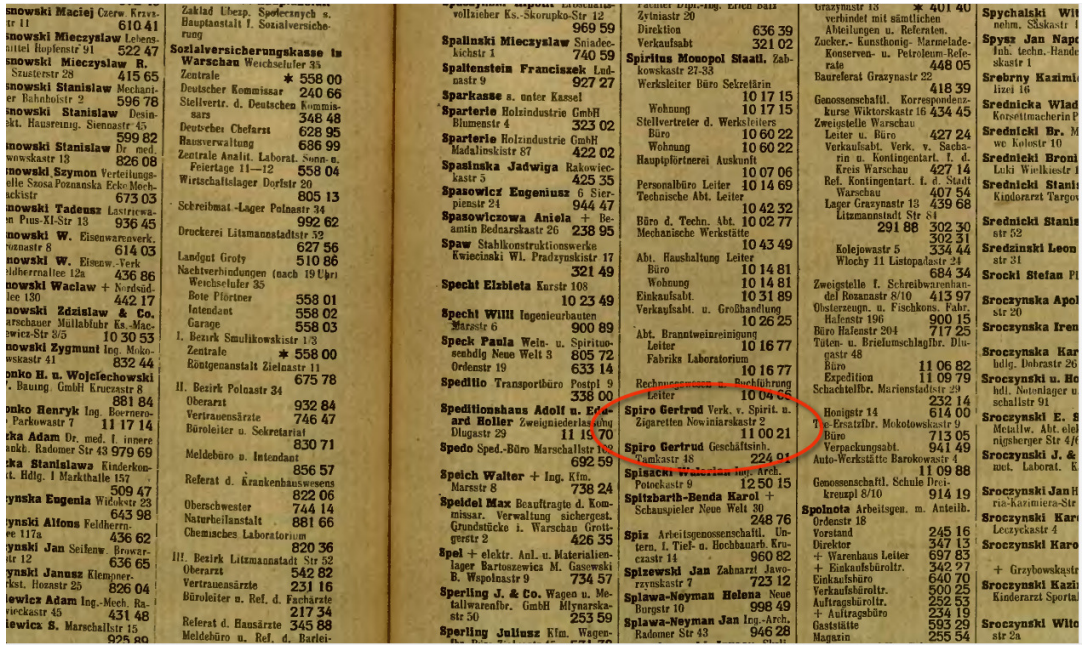
- Solution:
  - Use alphanumeric names to refer to hosts.
  - Called host names or domain names (e.g.: cs.rutgers.edu)
  - We need a directory (address book)
  - A service to map alphanumeric host names to binary IP addresses
  - We call this process Address Resolution

Types of Directories

- Directories map a *name* to an *address*
- Simplistic designs
  - Central directory
  - Ask everyone (e.g., flooding)
  - Tell everyone (e.g., push to a file like /etc/hosts)
- Scalable distributed designs
  - Hierarchical namespace (e.g., Domain Name System (DNS))
  - Flat name space (e.g., Distributed Hash Table)

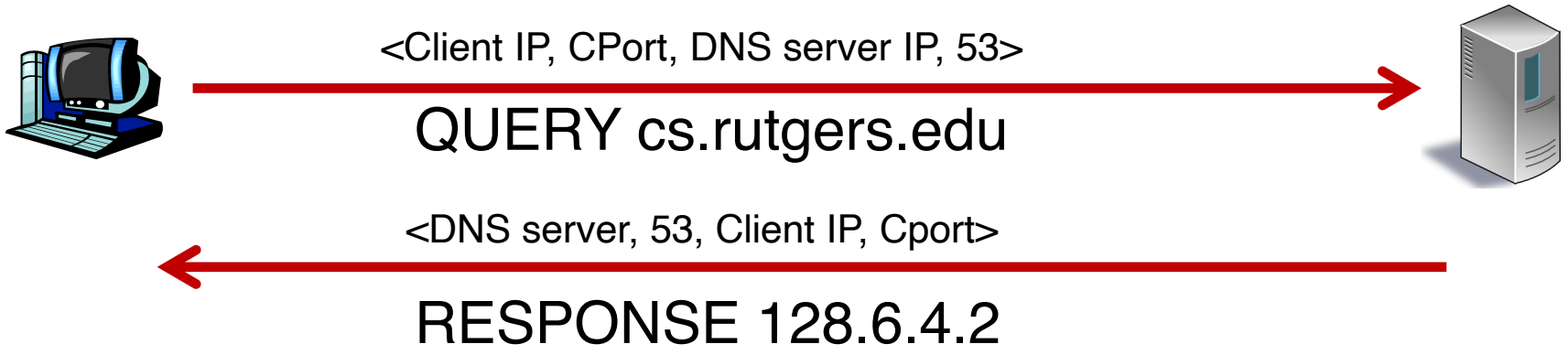
Simple DNS

- What if every endpoint has a local directory?
- `/etc/hosts.txt`
  - How things worked in the early days of the Internet!
- What if endpoints changed addresses? How do you keep this up to date?



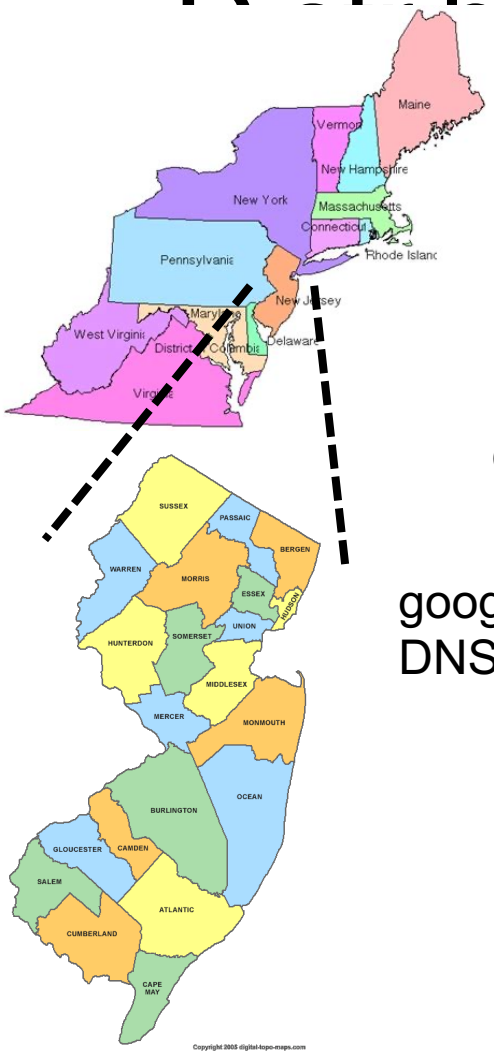
Simple DNS

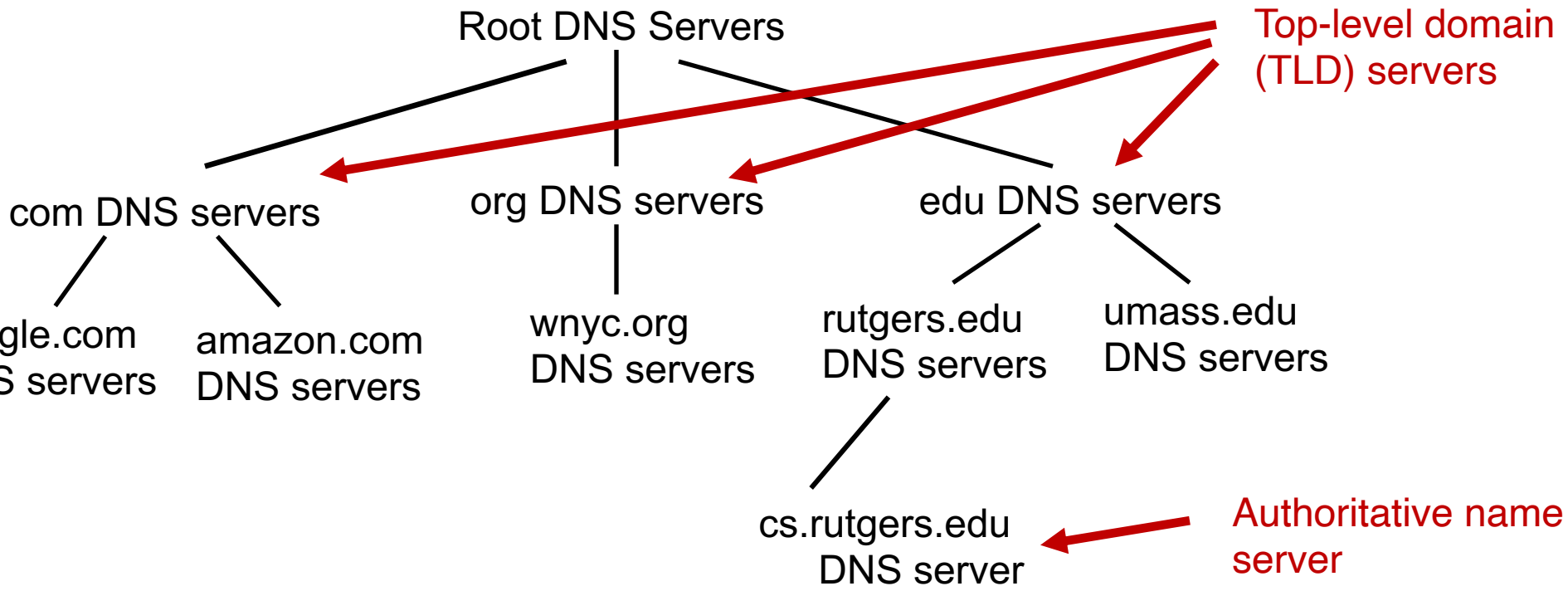
DOMAIN NAME	IP ADDRESS
spotify.com	98.138.253.109
cs.rutgers.edu	128.6.4.2
www.google.com	74.125.225.243
www.princeton.edu	128.112.132.86



- Key idea: Implement a server that looks up a table.
- Will this scale?
  - Every new (changed) host needs to be entered in this table
  - Performance: can the server serve billions of Internet users
  - Failure: what if the server or the database crashes?
  - How to secure this server?

Distributed and hierarchical database





RFC 1034: Distribution through hierarchy enables scaling

DNS Protocol

- Client-server application
- Client connects to (known) port 53 on server
- Assume DNS server IP known
- Two types of messages
  - Queries
  - Responses
- Type of Query (OPCODE)
  - Standard query (0x0)
    - e.g., Request IP address for a given domain name
  - Updates (0x5)
    - Provide a binding of IP address to domain name
- Each type has a common message format that follows the header