# Measurement, App Layer

Lecture 3

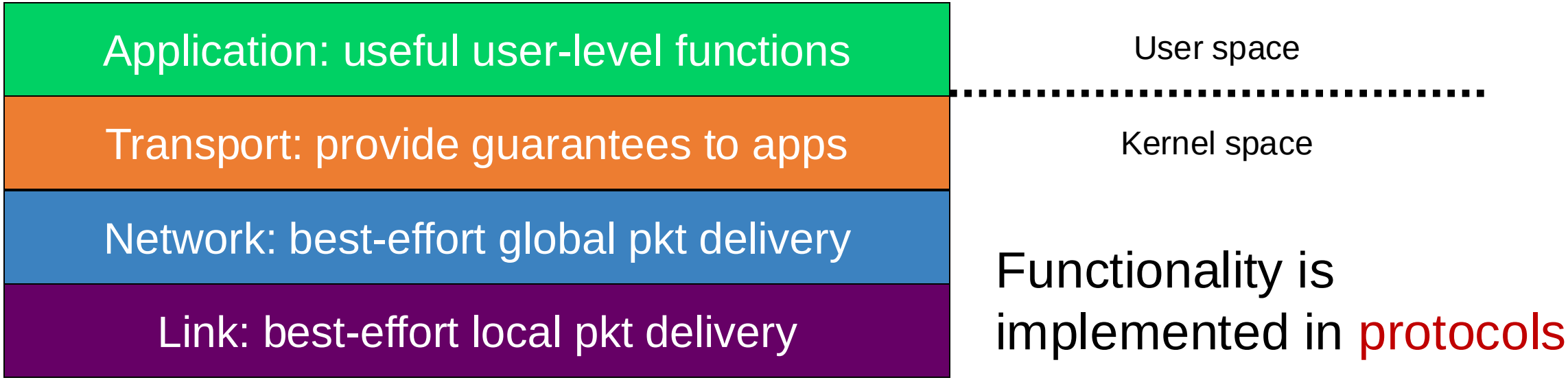
http://www.cs.rutgers.edu/~sn624/352-F24

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

# Review

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




Functionality is implemented in protocols

# Protocols: The “rules” of networking

- Protocols consist of two things

- Message format
  - structure of messages exchanged with an endpoint

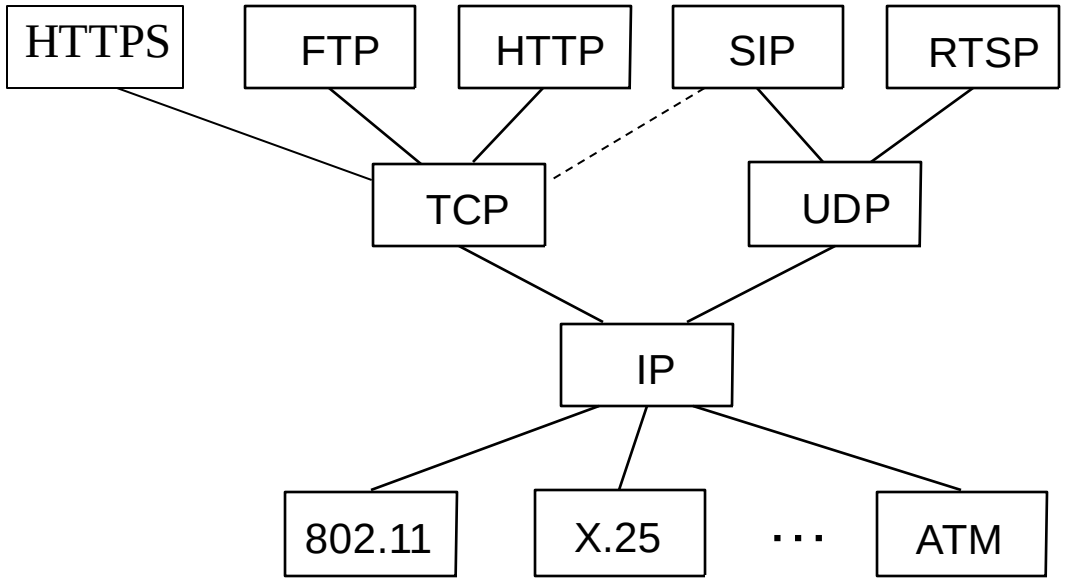
- Actions
  - operations upon receiving, or not receiving, messages

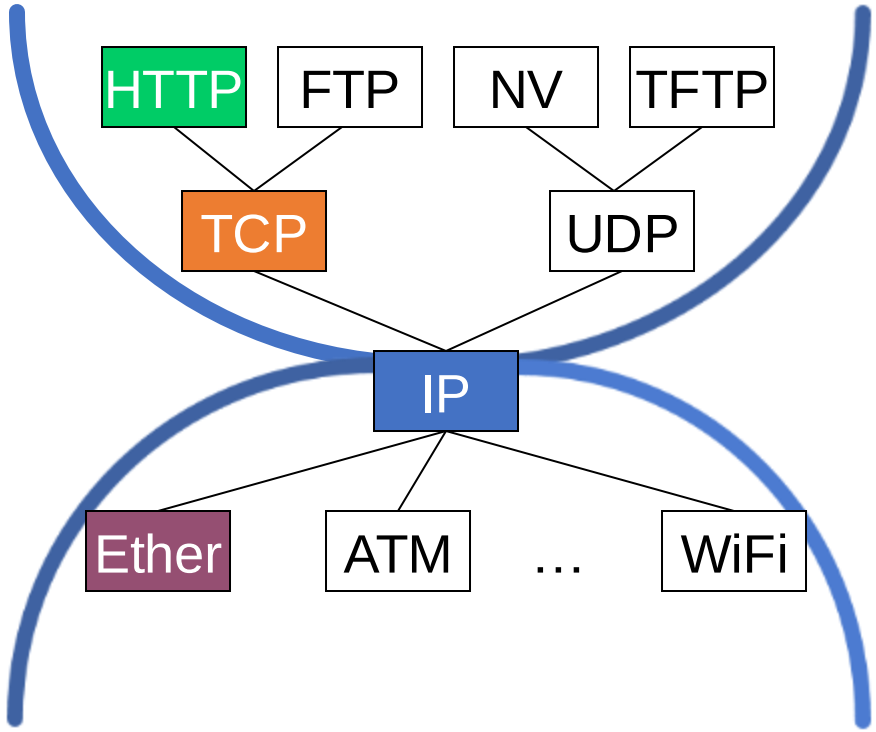
- Example of a Zoom conversation:
  - Message format:  English words and sentences
  - Actions: when a word is heard, say “yes”; when nothing is heard for more than 3 seconds, say “can you hear me?”

# The protocols of the Internet

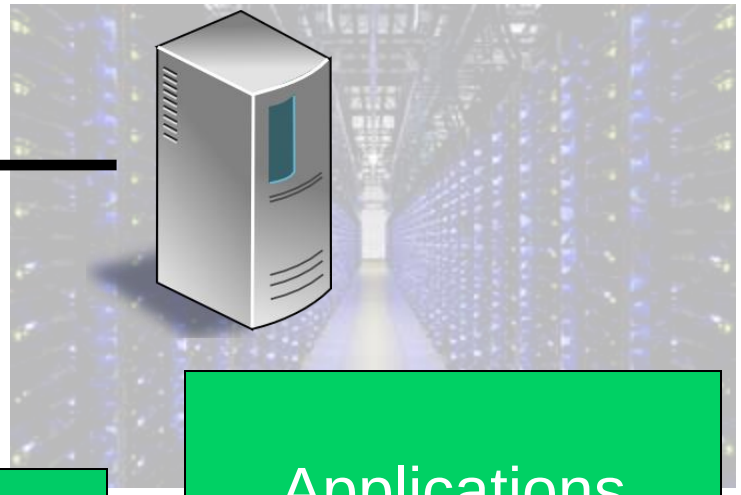
- Standardized by the Internet Engineering Task Force (IETF)
  - through documents called RFCs (“Request For Comments”)

- Layering of protocols



The Internet



Applications

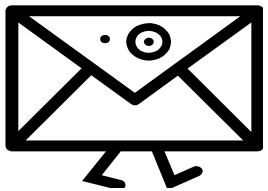
Transport

Network

Link layer

Packet starts as an app message

Packet takes on headers at each layer



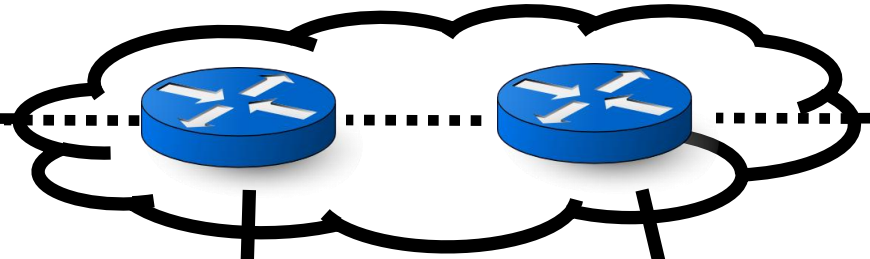
The Internet

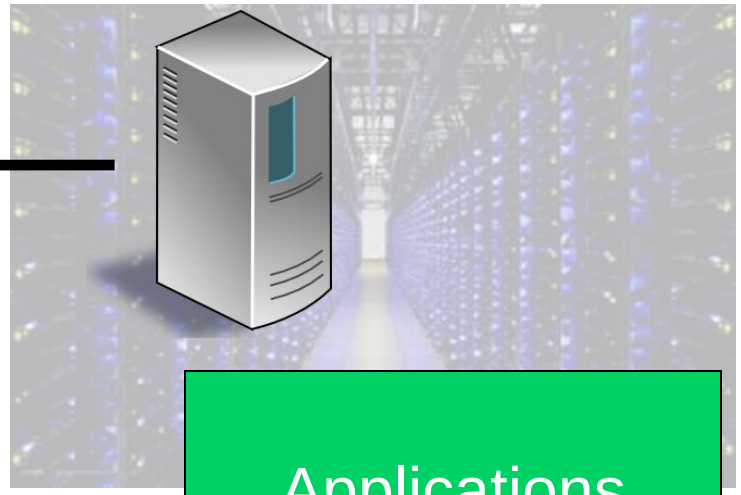
Applications

Transport

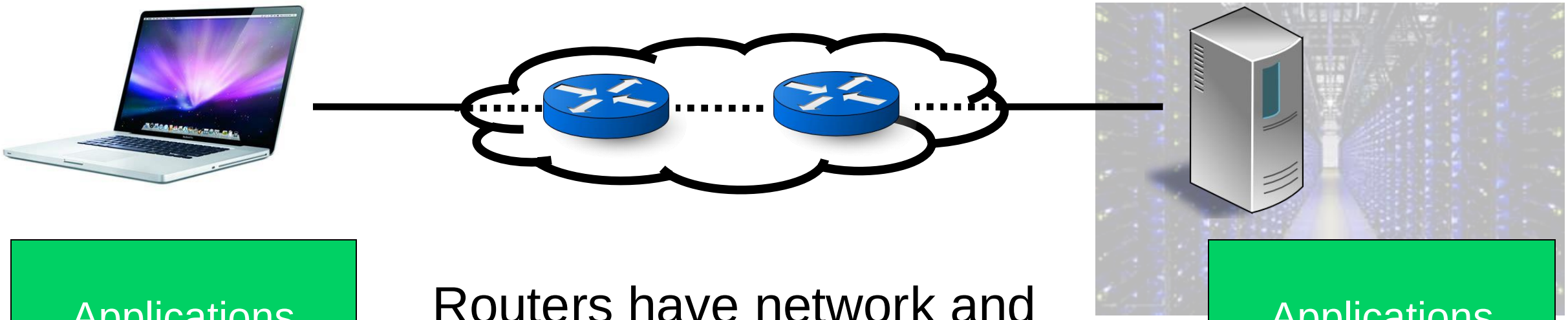
Network

Link layer

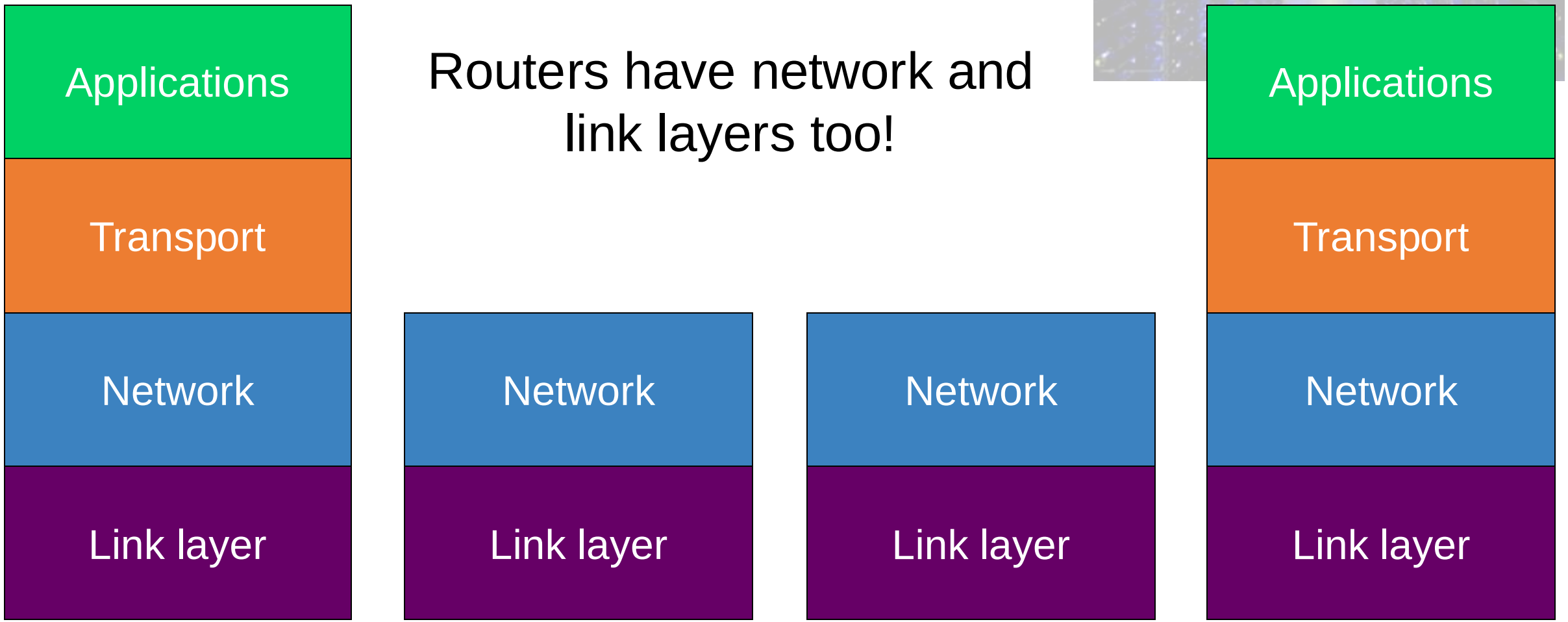




| | | | |
|---|---|---|---|
| Applications | | | Applications |
| Transport | | | Transport |
| Network | Network | Network | Network |
| Link layer | Link layer | Link layer | Link layer |



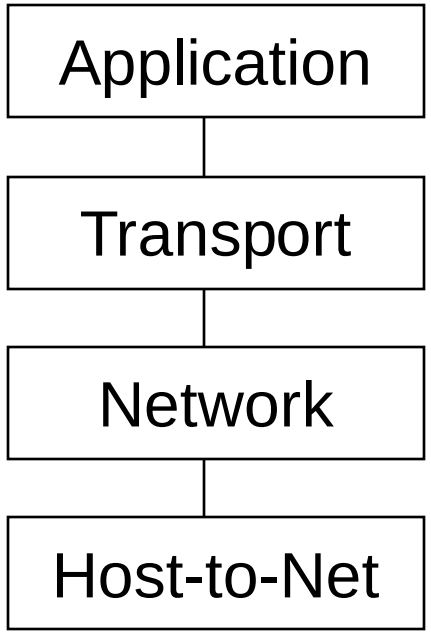
Routers have network and link layers too!

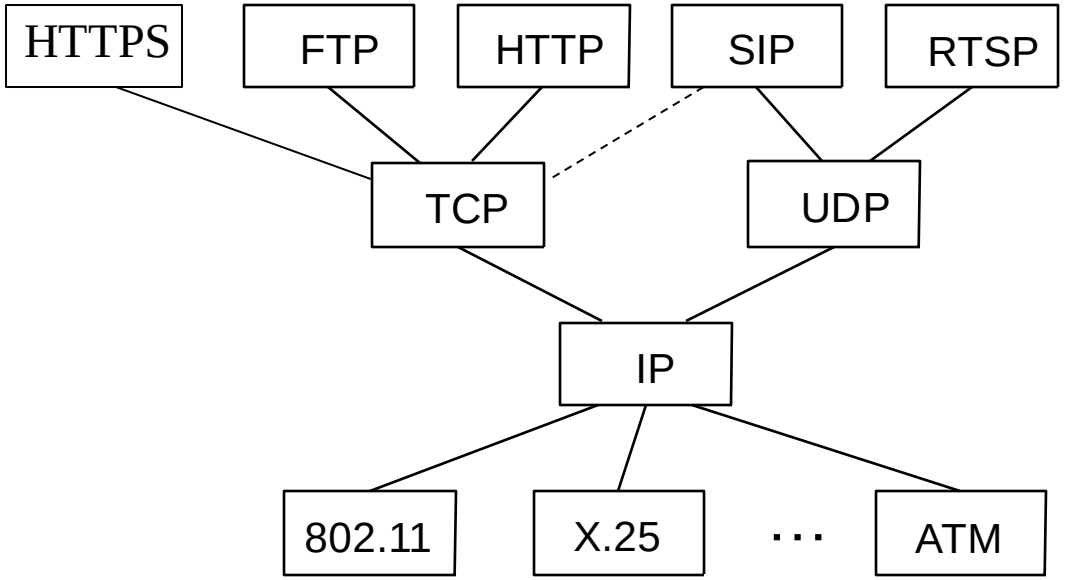


# Layering

- Communication over the Internet is a complex problem.
- Layering simplifies understanding, testing, maintaining
- Easy to improve or replace protocol at one layer without affecting others

# This course has layers




# Measuring the Internet

Speed, by any other name

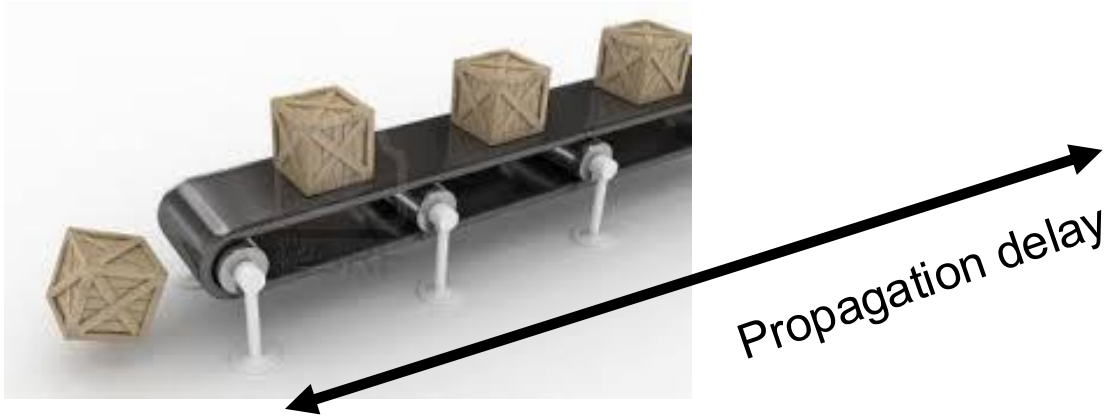
# What exactly do we mean by speed?

- A packet consists of many bits, including header and data
  - Packet size: length of the packet (bits or bytes) incl. header and data

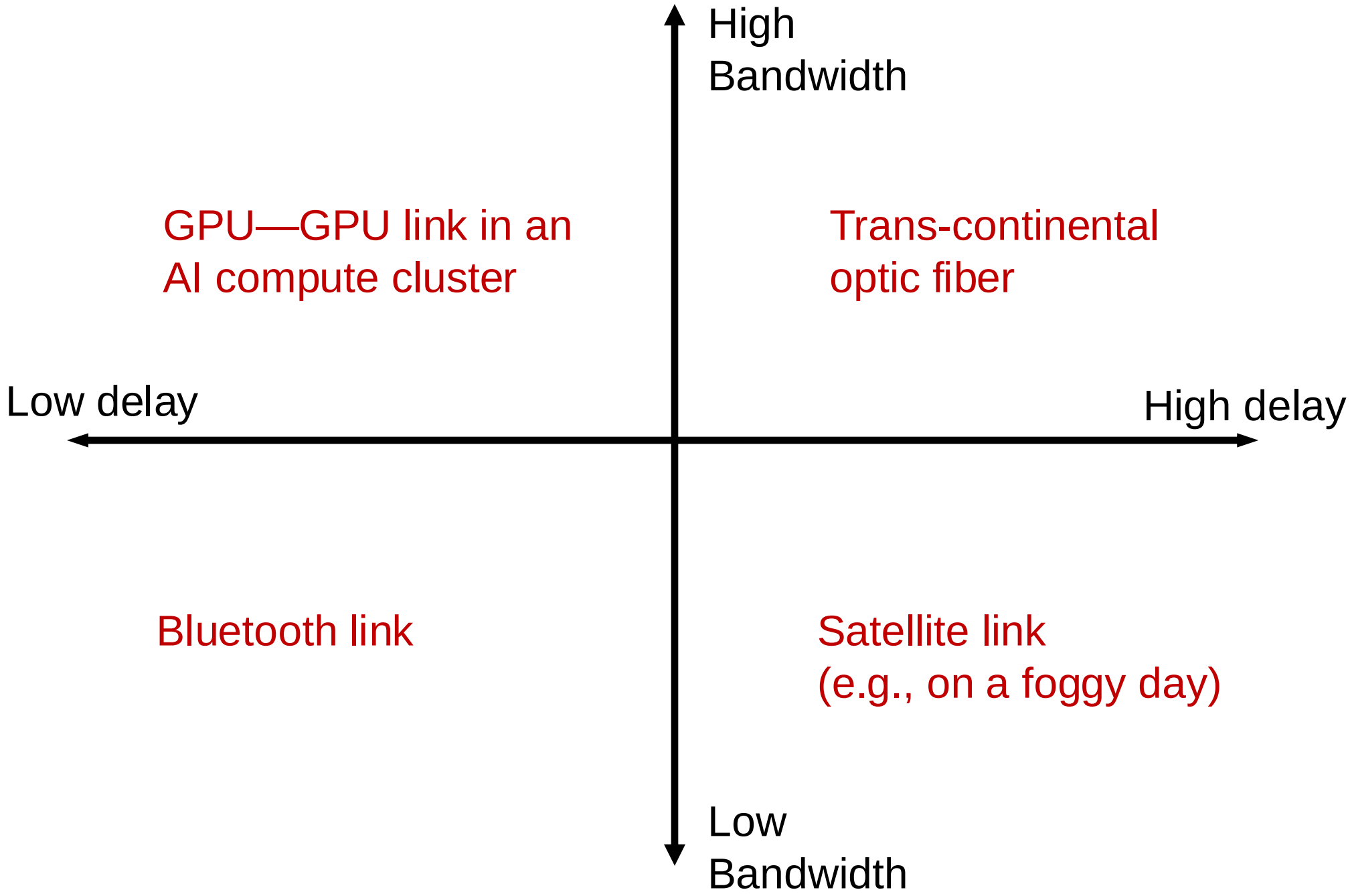
- Bandwidth: For a single link, amount of data it can transmit per unit time (bits/second or Bytes/second or packets/second)

- Total packet delay: time from the first bit@sender to the last bit@receiver

# Bandwidth and delay are related but distinct



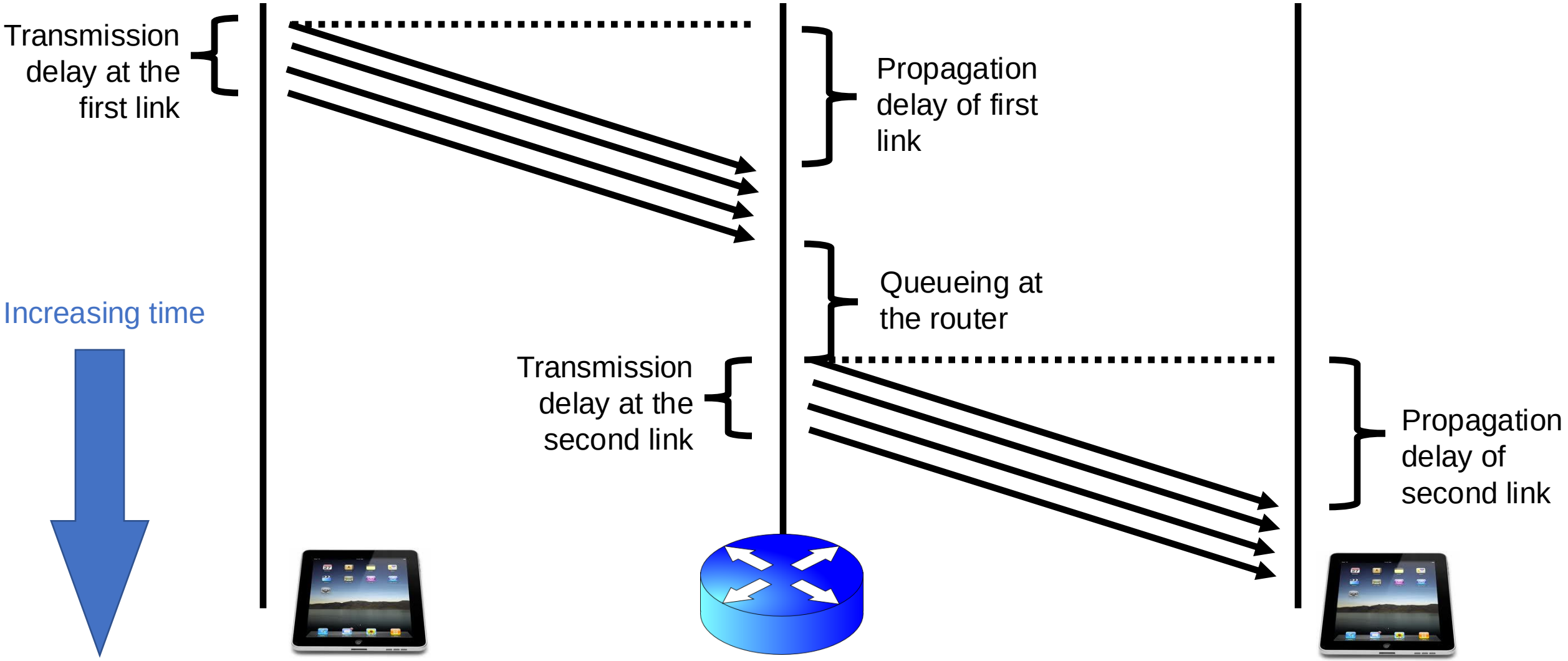

- Total packet delay = time for a box to travel the length of the belt
- Bandwidth = the number of boxes put on the belt per minute (“rate”)


High
Bandwidth
GPU—GPU link in an
AI compute cluster
Trans-continental
optic fiber
Low delay
High delay
Bluetooth link
Satellite link
(e.g., on a foggy day)
Low
Bandwidth

# Total Packet Delay has a few pieces

- 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
  - Packet size / link bandwidth
- Queueing delay: Time that a packet waits for transmission
  - Determined by contention for the link
- Total packet delay = propagation delay + queueing delay + transmission delay for a single packet

# Visualizing the components of delay



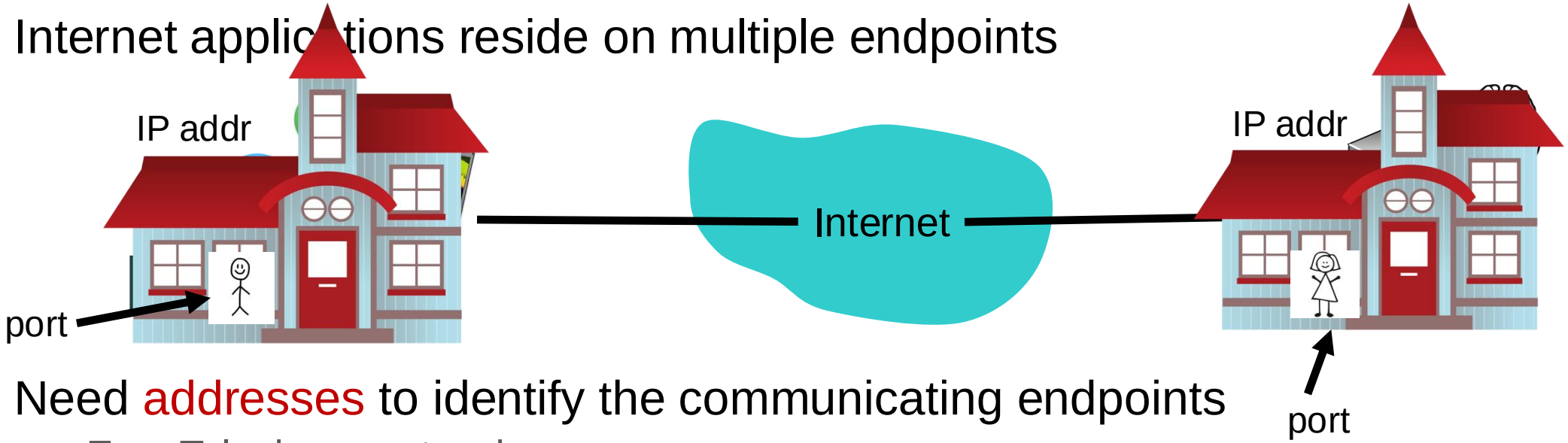
# Bandwidth and delay demo

- 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`
- (you can try it!)

# Application Layer

# App-layer communication

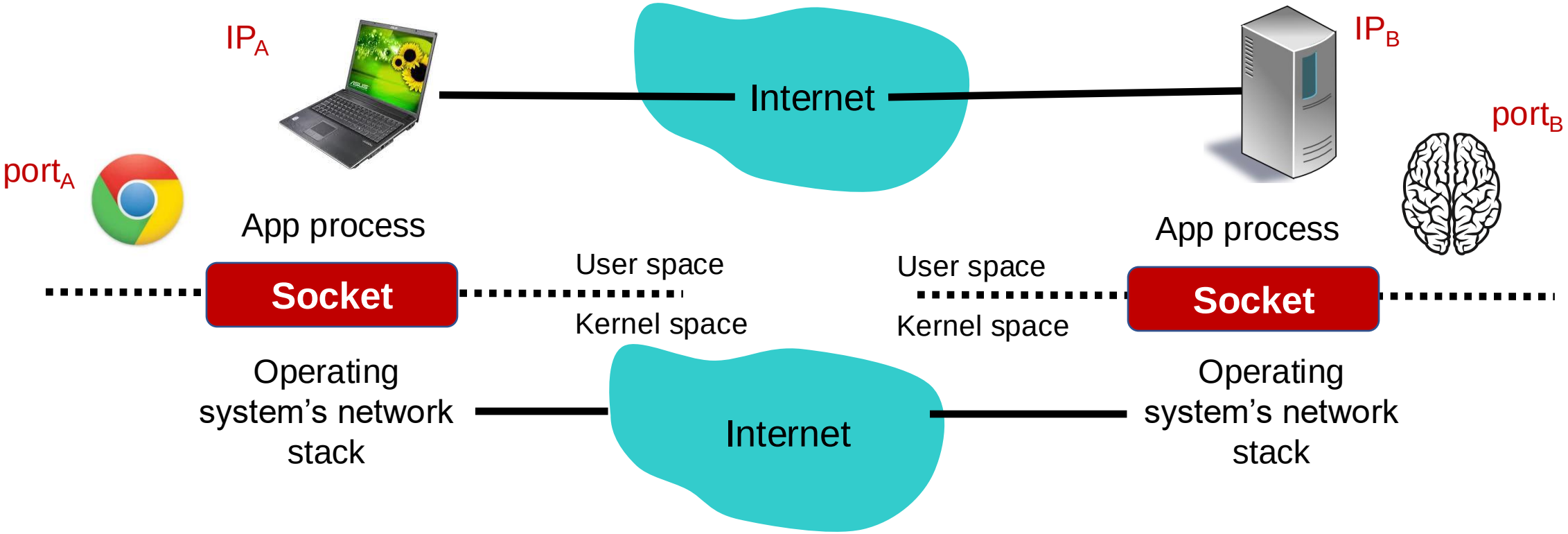
- Internet applications reside on multiple endpoints




- 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()`

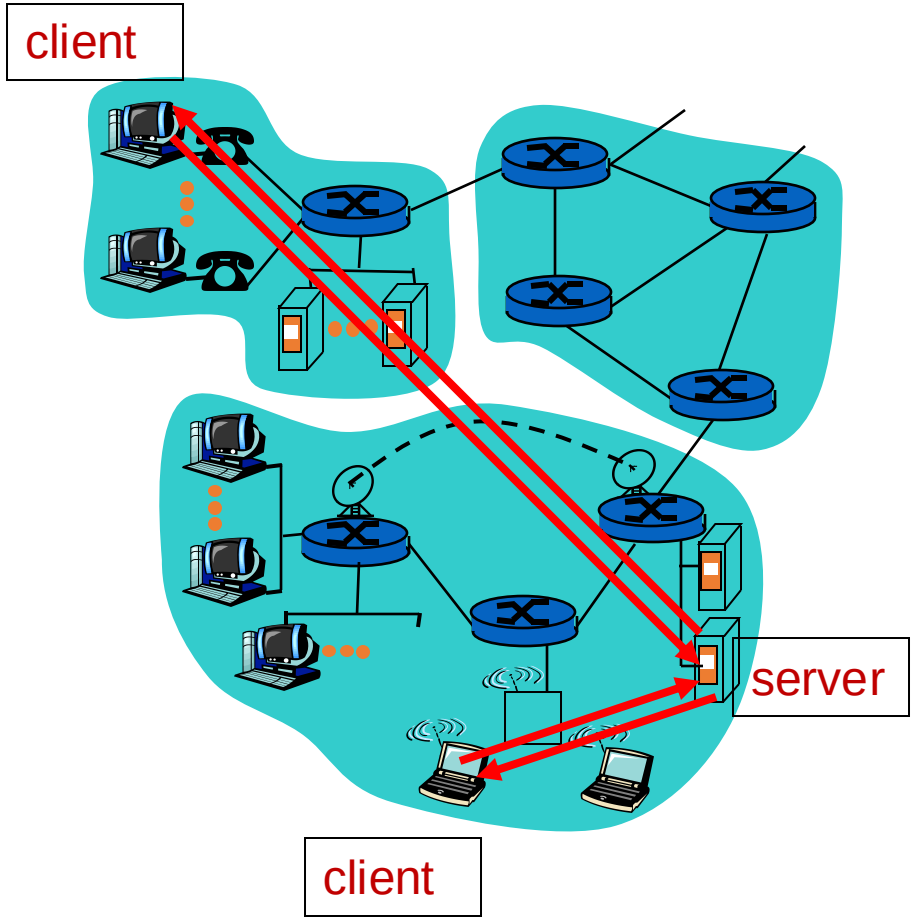
`recv()`

# Seeing app-layer connections

- `netstat`
- `ss`

# Common Architectures of Applications

# Client-server architecture




## Server:

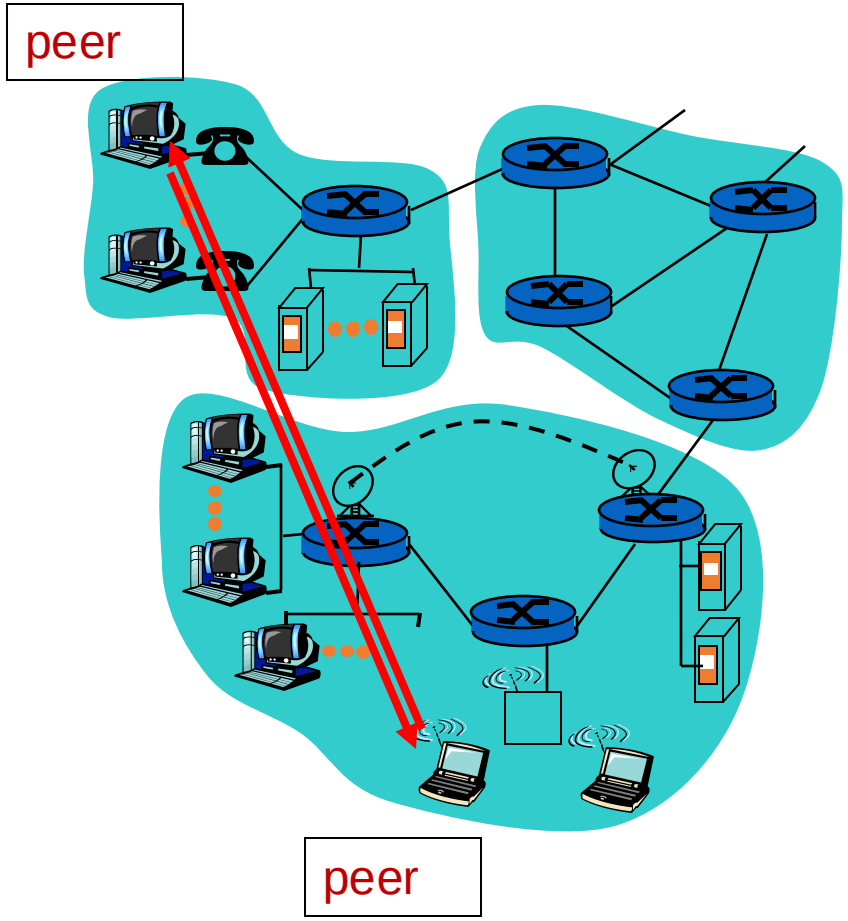
- Always-on endpoint
- Provides a “service” to the world
- Typically, a permanent IP address
- Compute clusters to scale to many users

## 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, Facebook messenger, …

# Going forward: A few app-layer protocols

- Domain Name System
- The web
- Streaming video

# Domain Name System

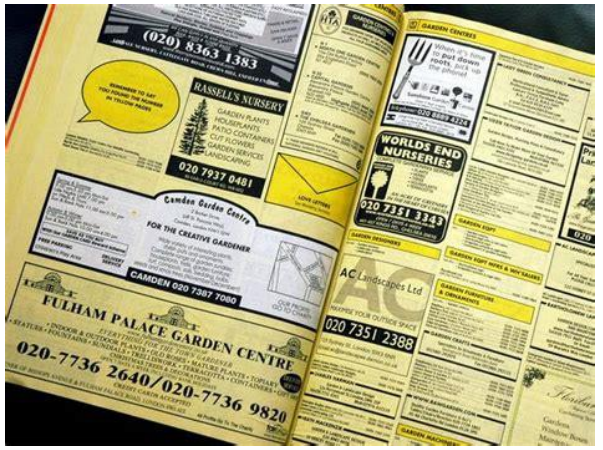
You have my name. Can you lookup my address?

# Domain Name System (DNS)

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

- 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

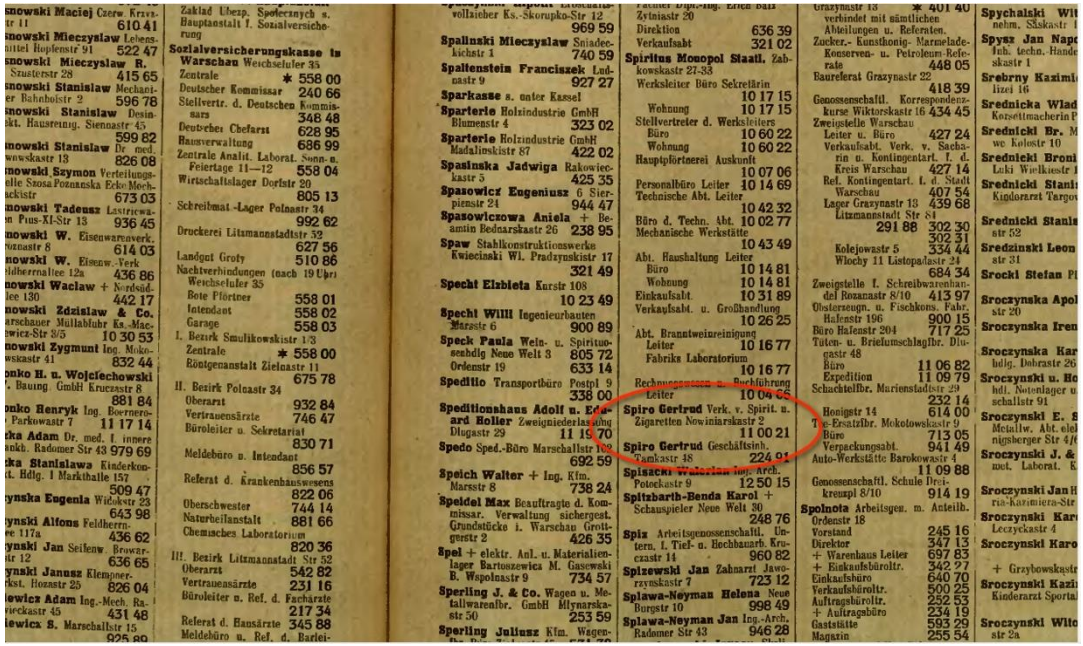
# Types of Directories




- 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)

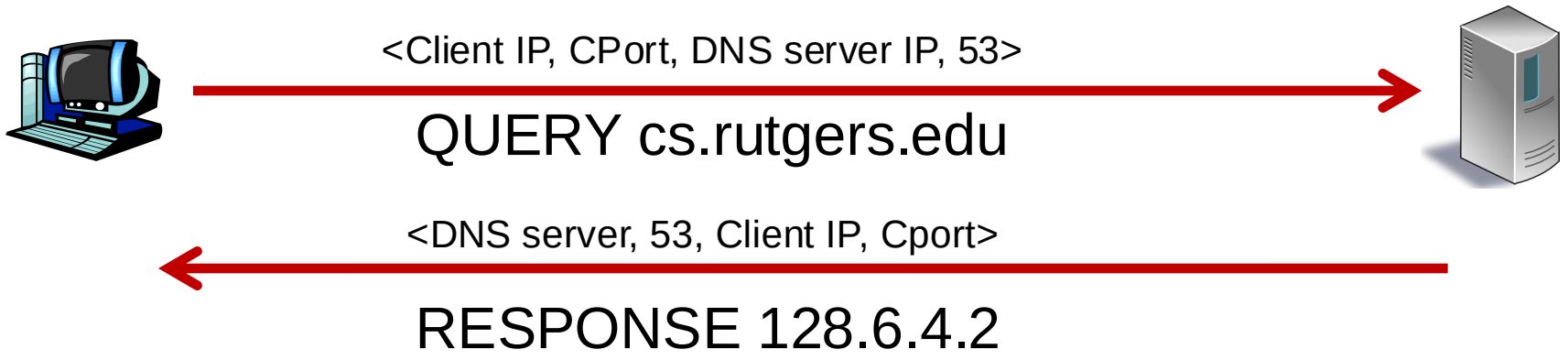
# Simple DNS

- 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?