Internetworking in a day
Day 1 of 2

Carnegie Mellon 15-440 - Distributed Systems

Key Things to Watch For

- Modularity, Layering, and Decomposition: Techniques for dividing the work of building systems; hiding the complexity of components from each other; hiding implementation details to deal with heterogeneity
- Resource sharing and isolation
- Models and assumptions about the environment and components
- and...

Dist. Sys challenges

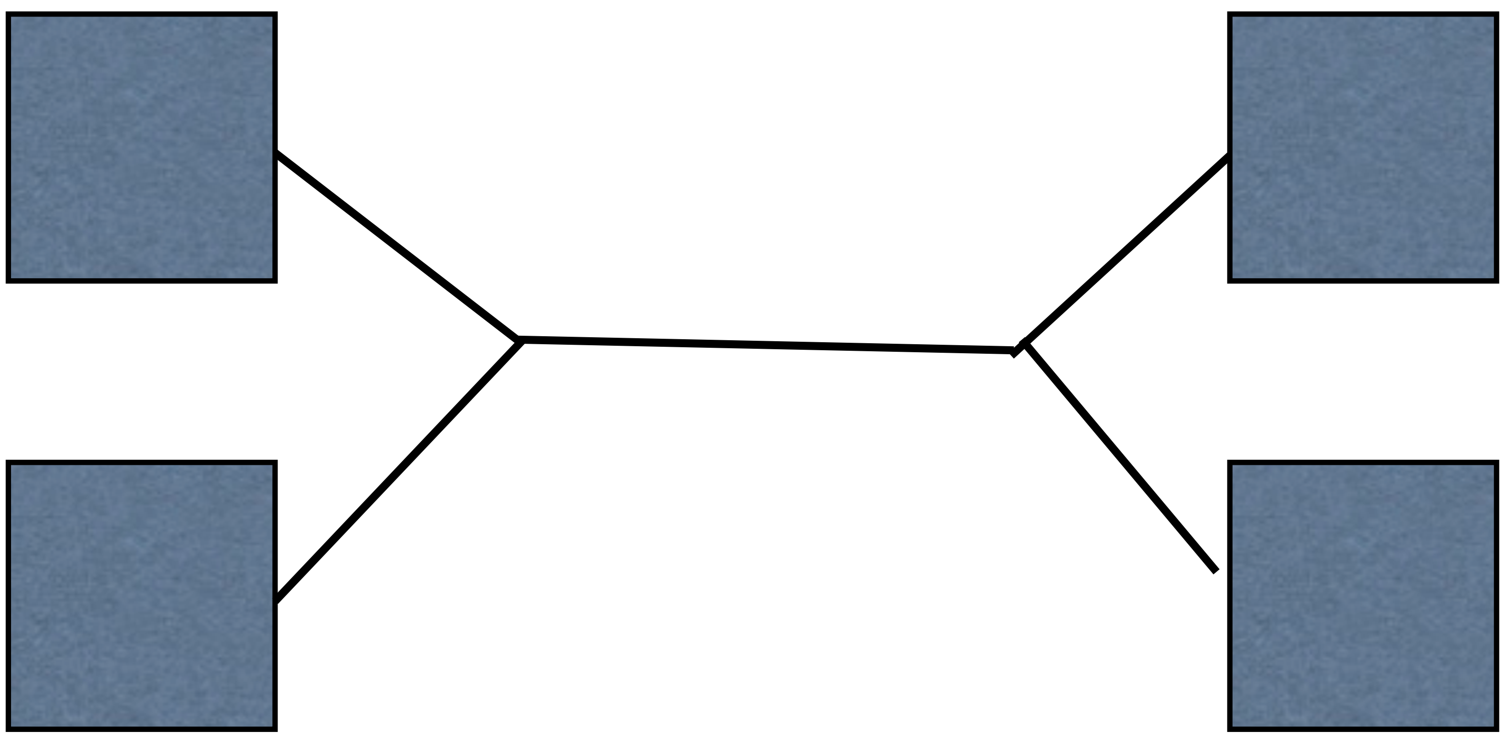
- Heterogeneity (ex: how many different types of devices are there on the Internet?)
- Scale (how big is the Internet?);
- Perhaps: Geography (speed of light is a bummer)
- Security (what a mess is the Internet?)
- Failure Handling (how reliable is software?)
- Concurrency

Networks

- Broadly speaking:
  - Circuit-switched (the phone network of old)
  - Packet-switched (the Internet)

- We're only talking about the latter. What does this mean? It's all about sharing.

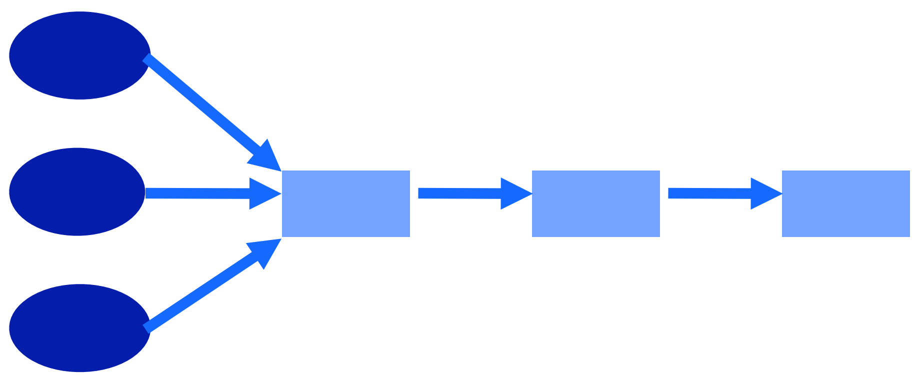
How do 2 nodes share a (wire, other medium?)



Multiplexing!
Talk at different frequencies (TV!)
Take turns -- time (long? circuits! short fixed? TDMA)
Packets (time, but not fixed)

Statistical Multiplexing

- Switches arbitrate between inputs



- Can send from *any* input that's ready
  - » Links never idle when traffic to send
  - » (Efficiency!)

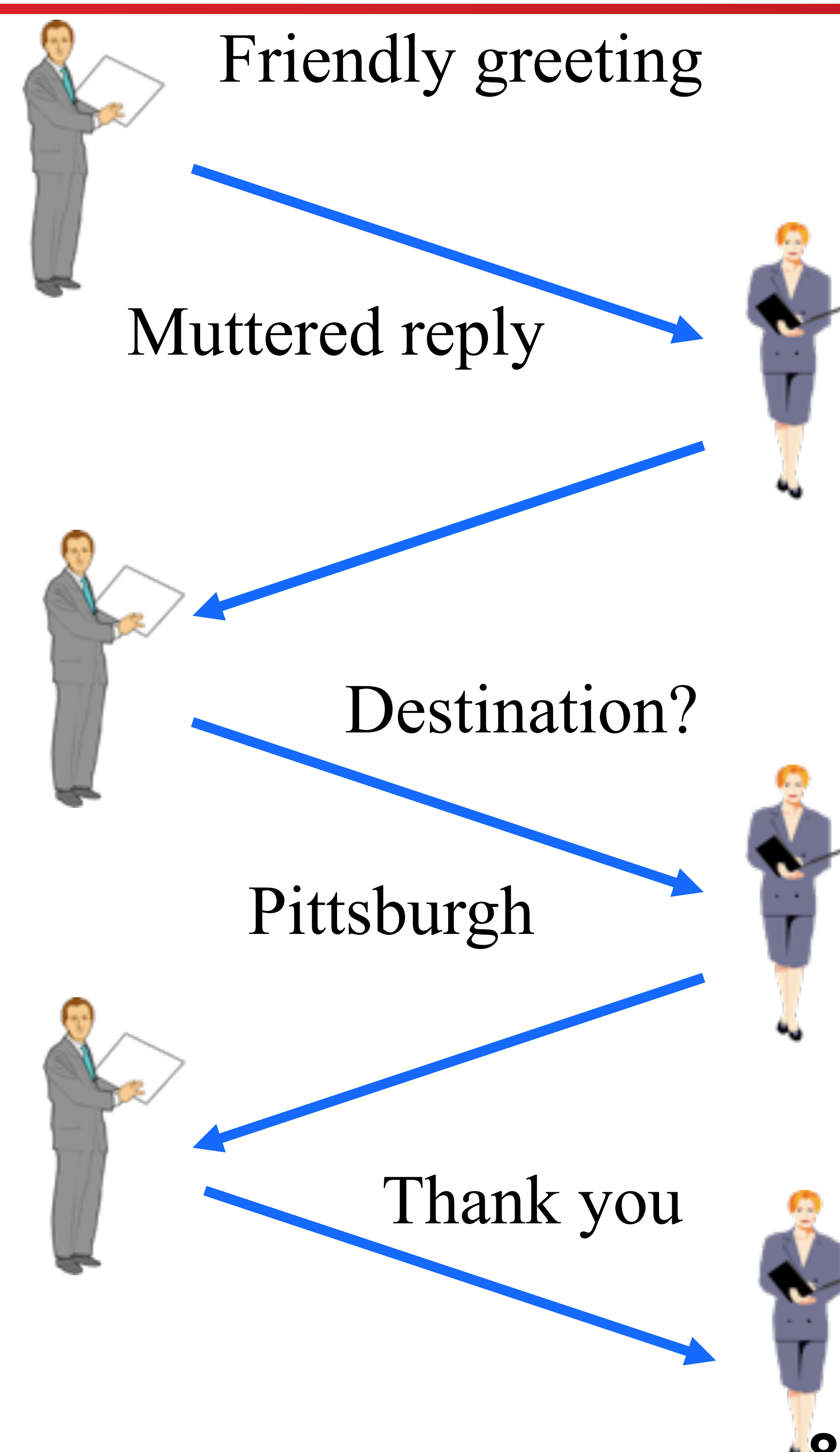
- What networks can we build with these tools?

Internets

- An “internet”: A network of networks
- The Internet: a global collection of over 18,000 individual networks that speak a common protocol (IP) and can talk to each other.
- The history: Uniting a mess of different, incompatible networks
  - Option 1: Protocol translators
  - Option 2: a common protocol

What is a Protocol

- An agreement between parties on how communication should take place.
- Protocols may have to define many aspects of the communication.
- Syntax:
  - » Data encoding, language, etc.
- Semantics:
  - » Error handling, termination, ordering of requests, etc.
- Protocols at hardware, software, *all* levels!
- Example: Buying airline ticket by typing.
- Syntax: English, ascii, lines delimited by "\n"



Interfaces

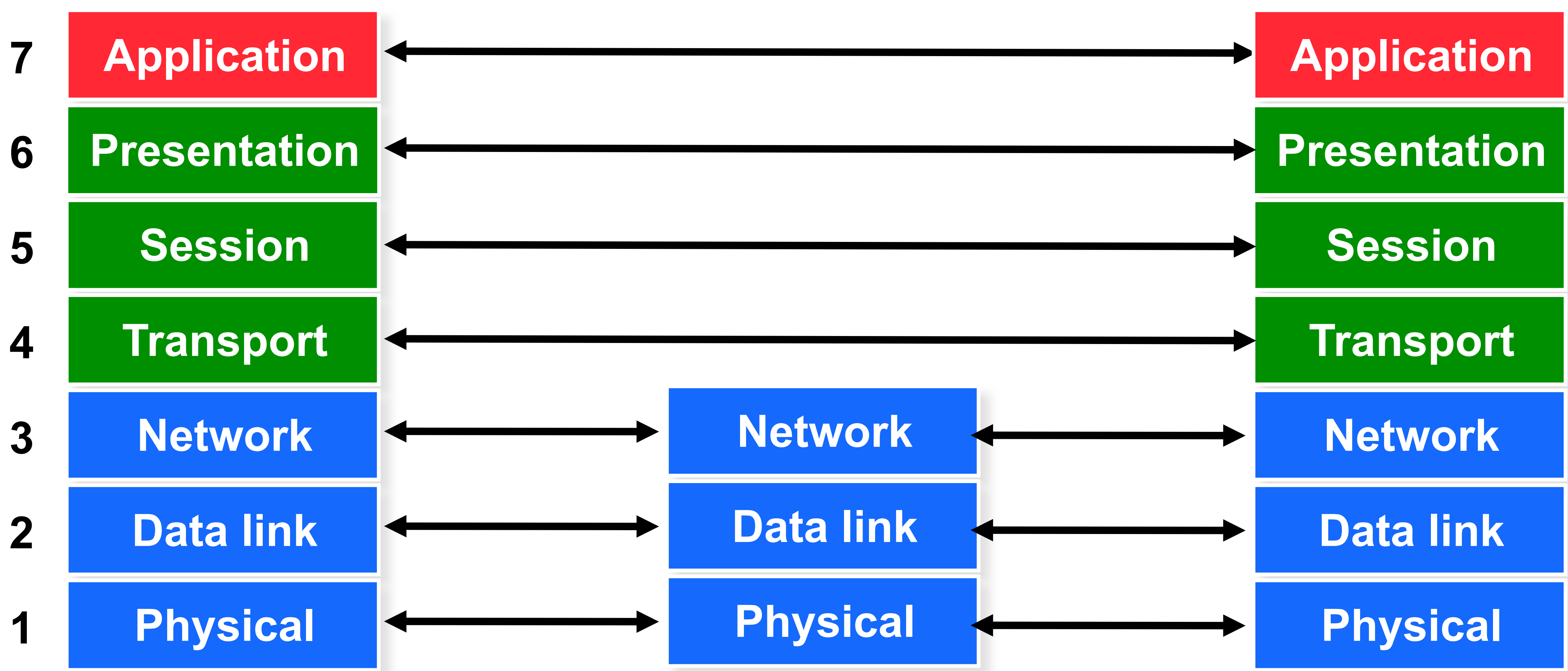
- Each protocol offers an interface to its users, and expects one from the layers on which it builds
  - » Syntax and semantics strike again
    - – Data formats
    - – Interface characteristics, e.g. IP service model
- Protocols build upon each other
  - » Add value
    - – E.g., a reliable protocol running on top of IP
  - » Reuse
    - – E.g., OS provides TCP, so apps don't have to rewrite

Common protocol

- Where to unifiy?
- How can you physically connect machines?
  - Optical, Electrical, Wireless, Carrier Pigeon, ...
  - Hm. Lots of diversity there -- we probably shouldn't define the common layer as a physical one.

A Layered Network Model

The Open Systems Interconnection (OSI) Model.



How do you talk in a medium?

- Once you've established a physical connection ... how do you signal data?
  - 1s and 0s...
  - separating messages...
- The answer to this depends a lot on the characteristics of the physical medium
  - Example: point-to-point optical network with a cable in each direction
  - vs. shared wireless where your transmissions can be overheard by others
- This is the "link" layer, and it's nice to be able to adapt on a per-technology basis

What about applications?

- You could standardize the mail exchange format (it's been done...);
- and then the news format ... and the www format ... and the skype format .. and .. and ... and...
  - Ew! Inhibits deployment of new applications - results in the phone network. Every network would have to understand every application protocol.

Internet Goals

- **Fundamental goal: Effective network interconnection**
- **Goals, *in order of priority:***
  1. **Continue despite loss of networks or gateways**
  2. **Support multiple types of communication service**
  3. **Accommodate a variety of networks**
  4. **Permit distributed management of Internet resources**
  5. **Cost effective**
  6. **Host attachment should be easy**
  7. **Resource accountability**

Principle: Fate Sharing

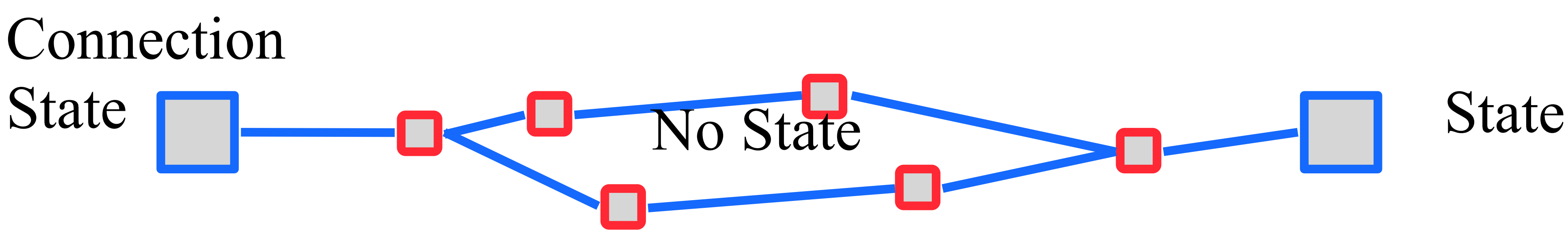
Make ability to communicate depend only on entities strictly needed...

Survivability

- If network disrupted and reconfigured
  - » Communicating entities should not care!
  - » No higher-level state reconfiguration
  - » Ergo, transport interface only knows “working” and “not working.” Not working == complete partition.
- How to achieve such reliability?
  - » Where can communication state be stored?

	Network	Host
Failure handing	Replication	“Fate sharing”
Net Engineering	Tough	Simple
Switches	Maintain state	Stateless
Host trust	Less	More

Fate Sharing



- **Lose state information for an entity if (and only if?) the entity itself is lost.**
- **Examples:**
  - » **OK to lose TCP state if one endpoint crashes**
    - – **NOT okay to lose if an intermediate router reboots**
  - » **Is this still true in today's network?**
    - – **NATs and firewalls**
- **Survivability compromise: Heterogenous network -> less information available to end hosts and Internet level recovery mechanisms**

Soooo...

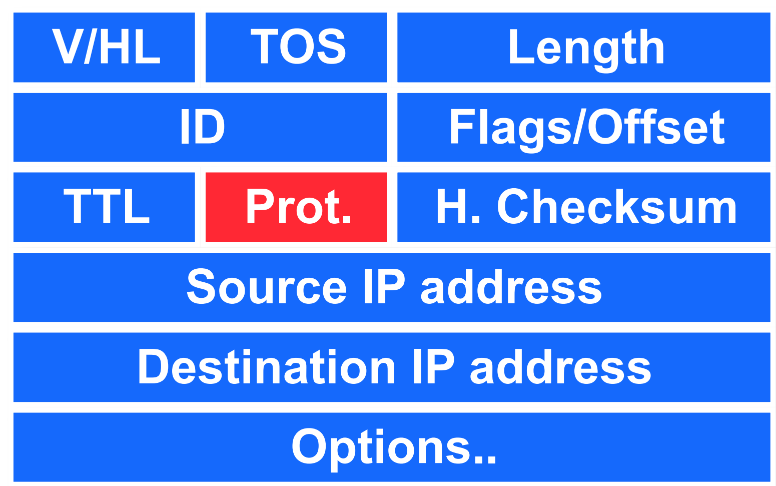
- TCP pushed to endpoints for survivability
  - » Connections can re-establish if intermediate routers crash and reboot. Good!

- Goal: Minimal requirements to interconnect networks
- IP is about the simplest thing.

-

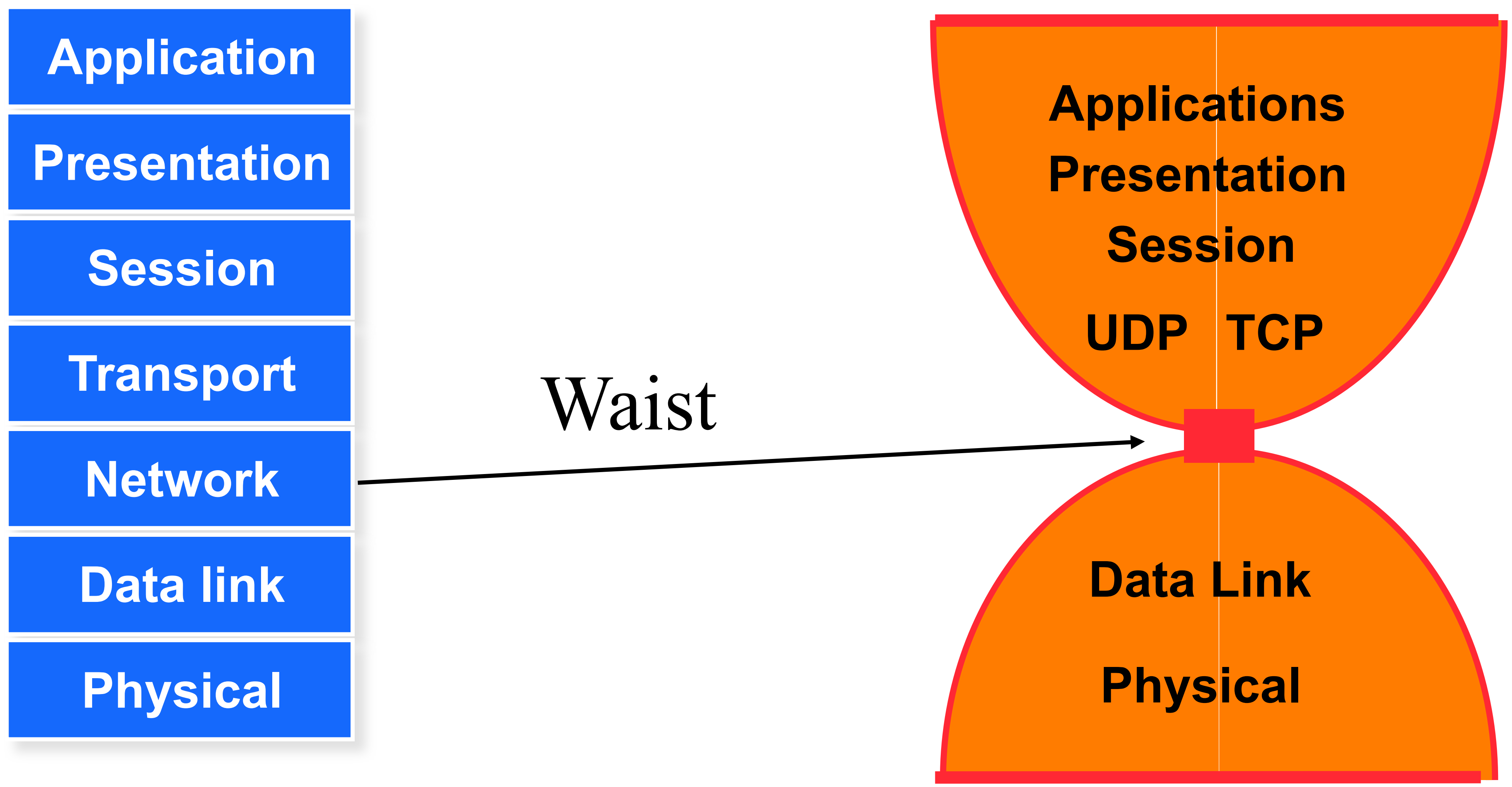
IP packets

- **Full diagram in slides...**



- **Important bits:**
  - » **Version**
  - » **Length**
  - » **Source IP address**
  - » **Dest IP address**
  - » **Protocol ID to demux to next layer up...**

The Internet Protocol Suite



The waist facilitates

Interoperability.

The Hourglass Model

How to combine protocols?

- In networking, often *encapsulate*
- eg IP packets sent in Ethernet packets;
- TCP packets sent in IP packets;
- HTTP request sent as...
  a stream of TCP packets inside IP packets
  inside foo...

- In OS, more likely to build on top of the abstraction, but not necessarily carry headers through

Questions to ask

- What is the interface between components?
  - » IP: send / receive packets
- What are the semantics (promise) made?
  - » IP: "Best-effort" delivery -- we'll try to get your packets there, but we might drop them
  - » or re-order them
  - » or change the contents
  - » or let someone listen to them
- We could just as easily ask these questions about, say, an operating system, or a filesystem. Same principle is at work.
- And to reason about them, we might abstract them into models...

System Models

- **ISO model of the network stack**
  - » **(note differences from TCP/IP model)**
- **Model of a communication channel**
  - » **Latency - how long does it take for the first bit to reach destination**
  - » **Capacity - how many bits/sec can we push through? (Often termed “bandwidth”)**
  - » **Jitter - how much variation in latency?**
  - » **Loss / Reliability - can the channel drop packets?**
  - » **Reordering**

Interaction models

- And can even model how processes communicate
  - "Synchronous" model: upper and lower bounds of time to execute a step of a process
  - Messages received within bounded time
  - Each computer's clock has bounded error from "true" time

- Asynchronous model (the Internet...)
  - steps may take unbounded time or fail
  - unbounded delay and re-ordering
  - no accurate local clock

Failure models

- **Fail-stop:**
  - » **When something goes wrong, the process stops / crashes / etc.**
- **Fail-slow or fail-stutter:**
  - » ***Performance* may vary on failures as well**
- **Byzantine:**
  - » **Anything that can go wrong, will.**
  - » **Including malicious entities taking over your computers and making them do whatever they want.**
- **These models are useful for proving things;**
- **The real world typically has a bit of everything. Deciding *which* model to use is important!**

Model Example: pw cracker

- Project 1: Build a password cracker
- Server --- many clients
- Communication:
  - » Send job
  - » ACK job
  - » do some work
  - » send result to server
  - » (repeat)

- IP communication model:
  - » Messages may be lost, re-ordered, corrupted (we'll ignore corruption, mostly, except for some sanity checking)
- Fail-stop node model:
  - » You don't need to worry about evil participants faking you out.

Protocols and Models

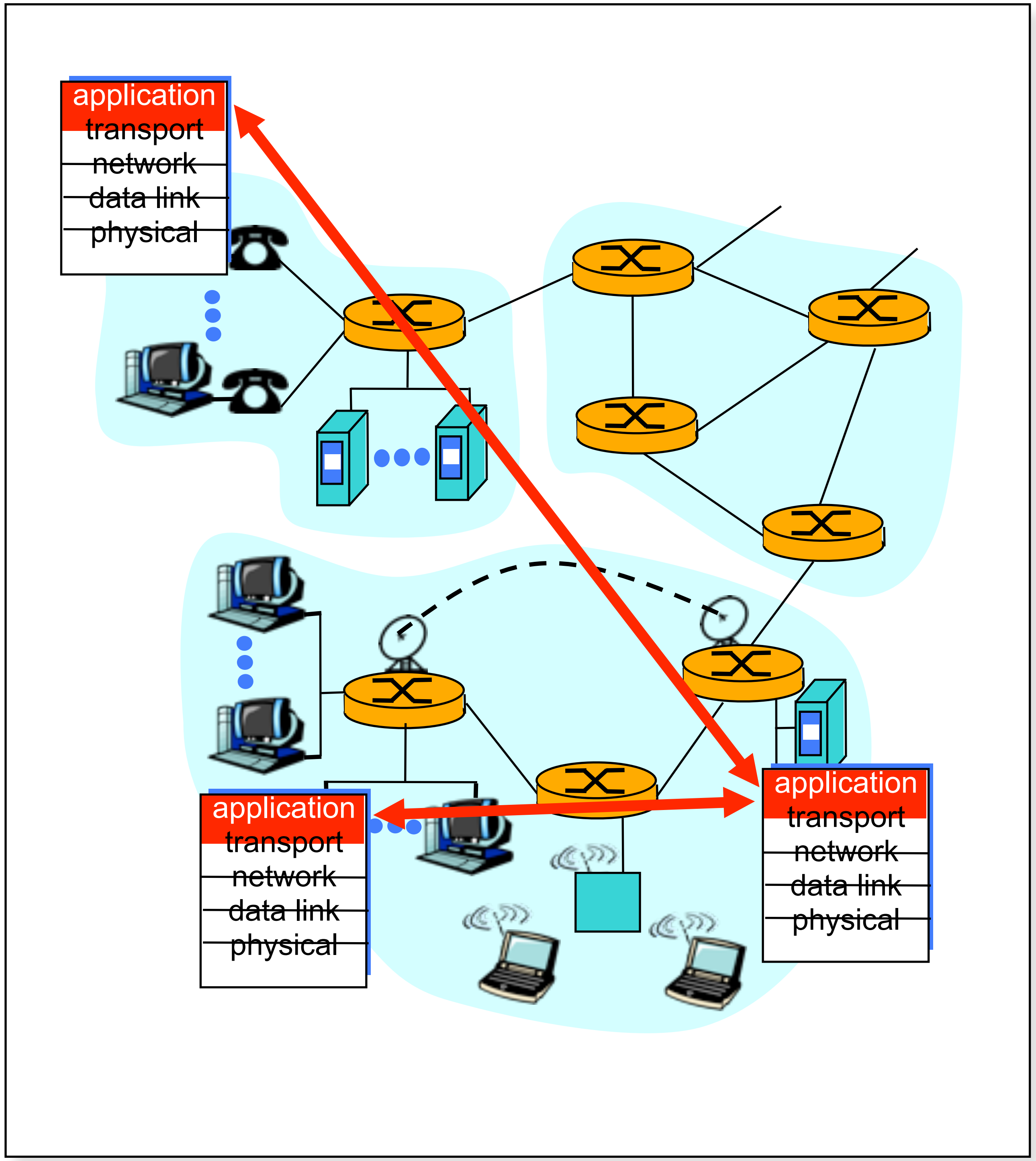
- **An interesting thing...**
  - » **We often build protocols that provide "simpler" models**
  - » **Example: TCP**
  - » **Provides reliable, in-order, mostly no-corruption, stream-oriented communication**
  - » **so that programmers don't have to implement these features in every application**
  - » **But note limitations: TCP can't turn a byzantine failure model into a fail-stop model...**

Designing applications

- Application architecture
  - » Client-server? (vs p2p vs all in one)
  - » Application requirements
- Application level communication
  - » TCP vs. UDP
  - » Addressing
- Application examples (Lecture 4).
  - » ftp, http
  - » End-to-end argument discussion

Applications and Application-Layer Protocols

- **Application: communicating, distributed processes**
  - » **Running in network hosts in "user space"**
  - » **Exchange messages to implement app**
  - » **e.g., email, file transfer, the Web**
- **Application-layer protocols**
  - » **One "piece" of an app**
  - » **Define messages exchanged by apps and actions taken**
  - » **Use services provided by lower layer protocols**
- **Sockets API refresher next week (remember from 213)**



Client-Server Paradigm

Typical network app has two pieces: *client* and *server*

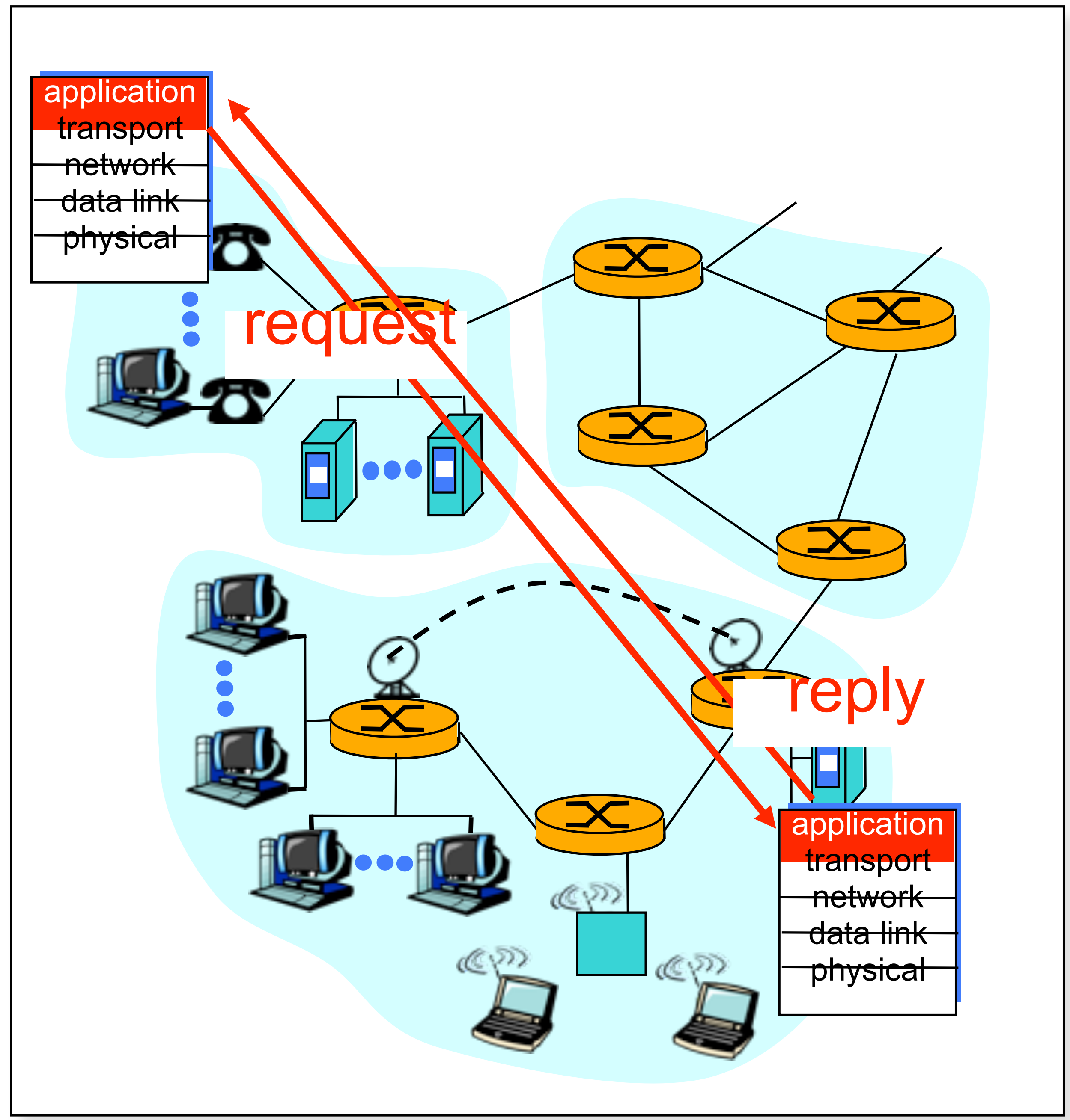
Client:

- **Initiates contact with server ("speaks first")**
- **Typically requests service from server,**
- **For Web, client is implemented in browser; for e-mail, in mail reader**

Server:

- **Provides requested service to client**
- **e.g., Web server sends requested Web page, mail server delivers e-mail**

- **(We'll cover p2p at semester end)**



What Transport Service Does an Application Need?

Data loss

- Some applications (e.g., audio) can tolerate some loss
- Other applications (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- Some applications (e.g., Internet telephony, interactive games) require low delay to be "effective"

Bandwidth

- Some applications (e.g., multimedia) require a minimum amount of bandwidth to be "effective"
- Other applications ("elastic apps") will make use of whatever bandwidth they get

User Datagram Protocol(UDP): An Analogy

UDP	Postal Mail
Single socket to receive messages	Single mailbox to receive letters
No guarantee of delivery	Unreliable ☺
Not necessarily in-order delivery	Not necessarily in-order delivery
Datagram – independent packets	Letters sent independently
Must address each packet	Must address each reply

Example UDP applications

Multimedia, voice over IP

Transmission Control Protocol (TCP): An Analogy

TCP	Telephone Call
Reliable – guarantee delivery	Guaranteed delivery
Byte stream – in-order delivery	In-order delivery
Connection-oriented – single socket per connection	Connection-oriented
Setup connection followed by data transfer	Setup connection followed by conversation

Example TCP applications

Web, Email, Telnet

Transport Service Requirements of Common Applications

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
web documents	no loss	elastic	no
real-time audio/ video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps	yes, 100's msec
financial apps	no loss	elastic	yes and no

- **Interactions between layers are important.**
  - » **persistent HTTP**
  - » **encryption and compression**
  - » **MPEG frame types. Loss & real-time video.**

Server and Client

Server and Client exchange messages over the network through a common Socket API