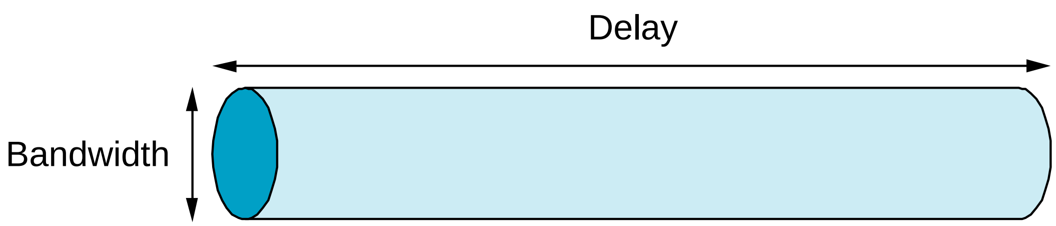
Lecture 2:
Protocols and Layering

CSE 123: Computer Networks
Stefan Savage

UCSDCSE

Last time

- Bandwidth, latency, overhead, message size, error rate
- Bandwidth-delay product



- High-level run through of how Web browsing works

Today

- Circuit Switching vs Packet Switching
- Protocols and Layering

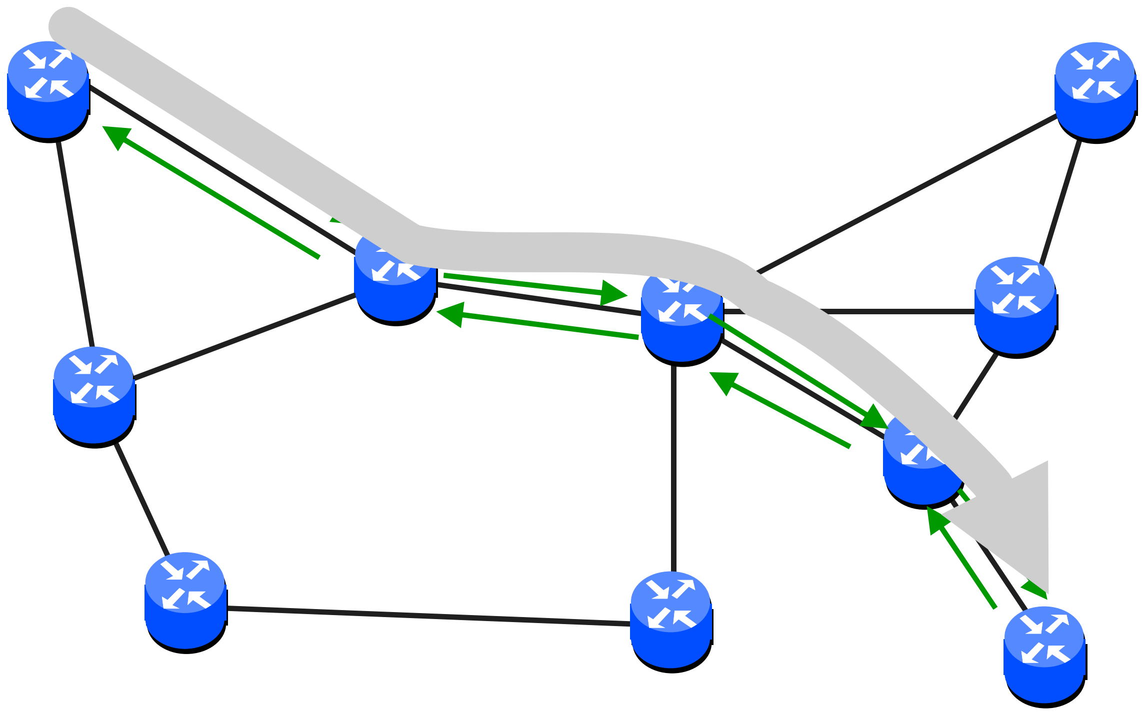
Circuit Switching

- Original phone system: continuous DC circuit from sender to receiver
- Physically switch circuit

- Circuit Switching: same model in digital domain
  - Model: data sent continuously
  - Created a session (e.g., phone call) reserves dedicated bandwidth in series of switches between caller and recipient
  - Guaranteed capacity (in both directions) so long as session up

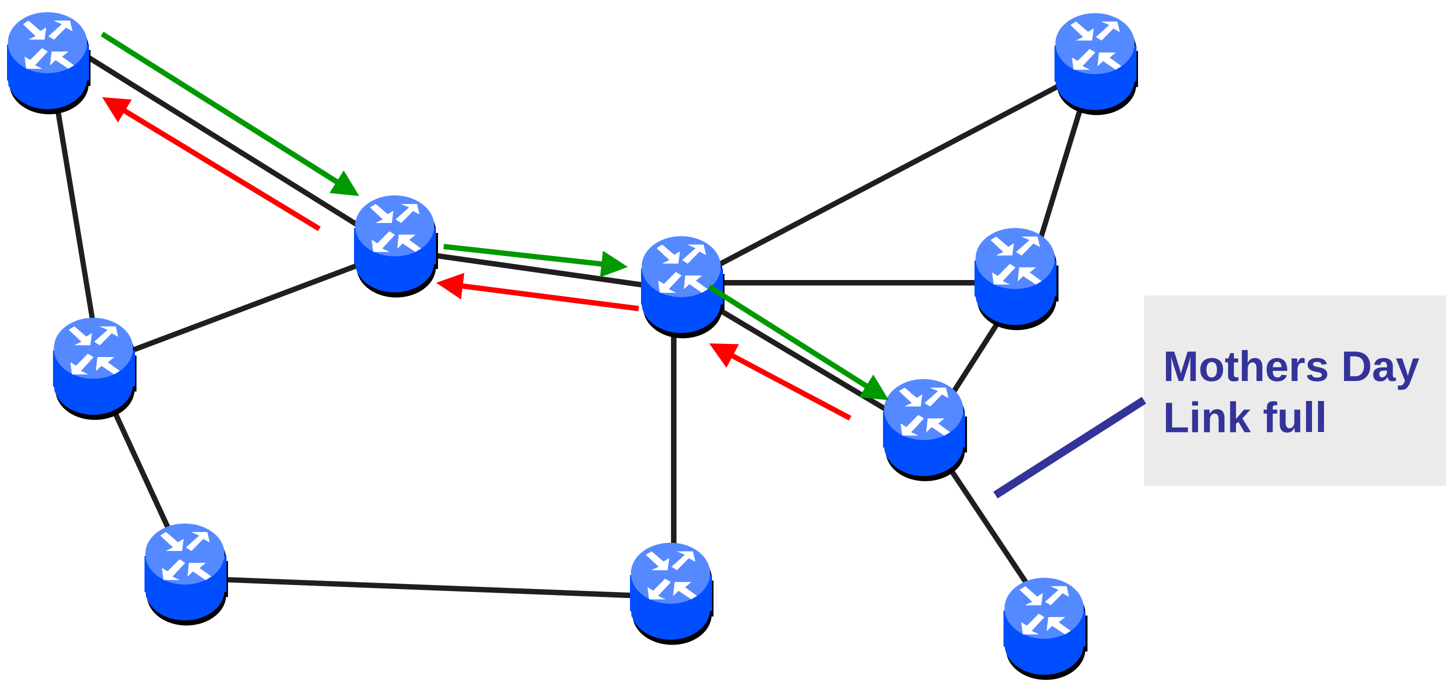
Circuit Switching

- Before sending, must reserve capacity for session
- Success = bandwidth is **guaranteed** for life of session



Circuit Switching

- What if request fails?
- Session refused (e.g., busy signal on phone)



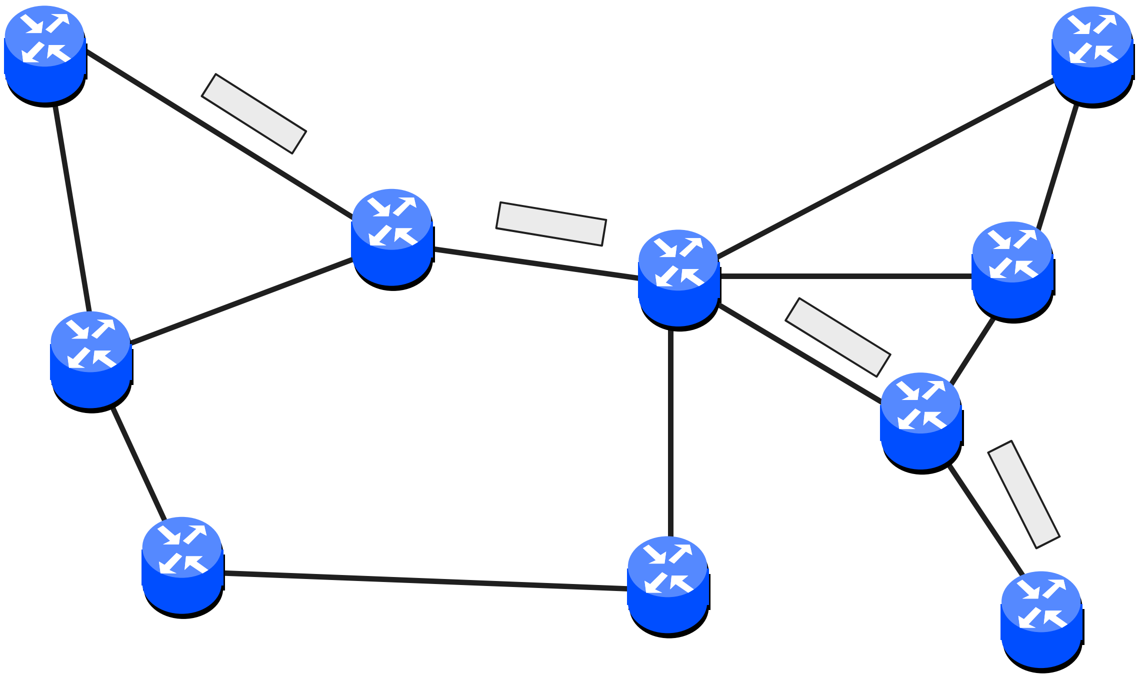
Packet Switching

- Packet Switching
  - Model: data sent opportunistically in small chunks (packets)
  - No session setup; send immediately
  - Each switch must know how to forward along any packet
  - Use queues to buffer bursts of traffic that exceed capacity

- History (mid-60s and 70s):
  - Paul Baran (US), Donald Davies (UK)
  - Kleinrock: queuing theory
  - Bob Taylor, JCR Licklider, Lawrence Roberts et al. ARPAnet

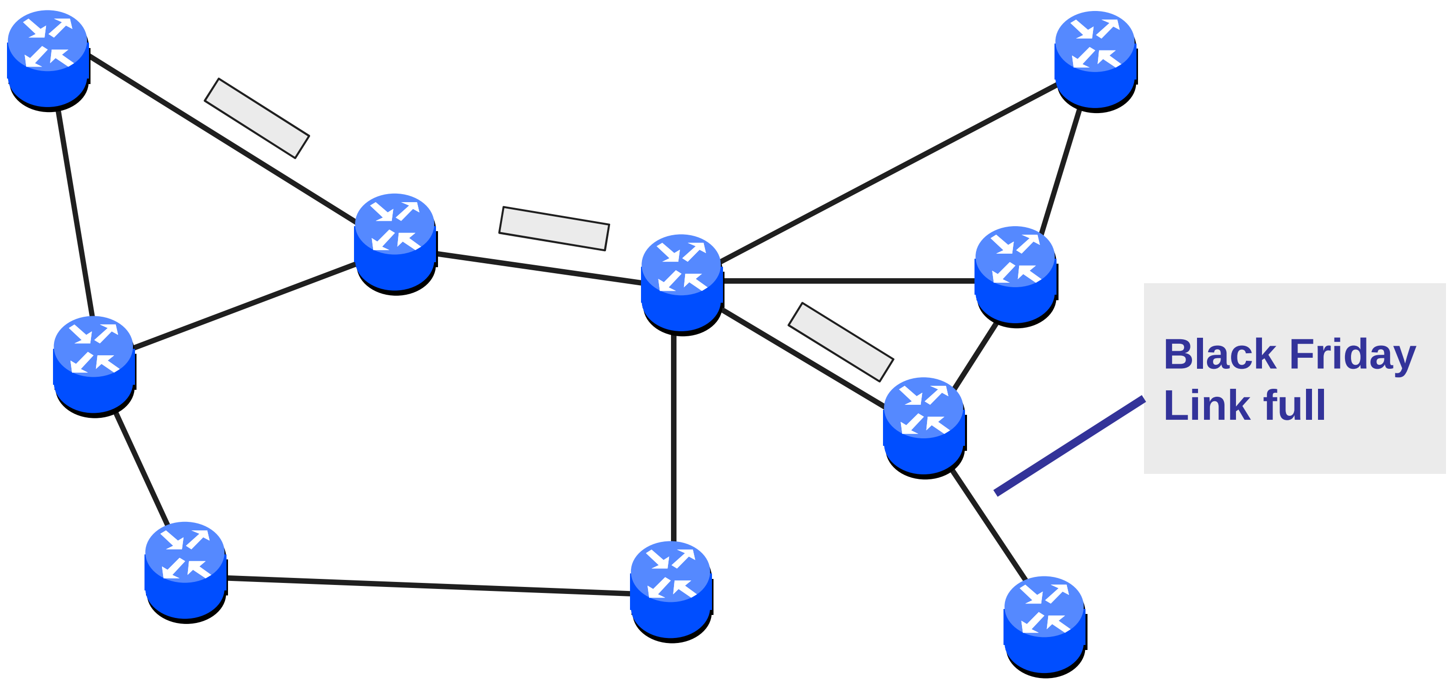
Packet Switching

- Break data into “packets”; send when they are ready
- Try to use bandwidth only when it is needed



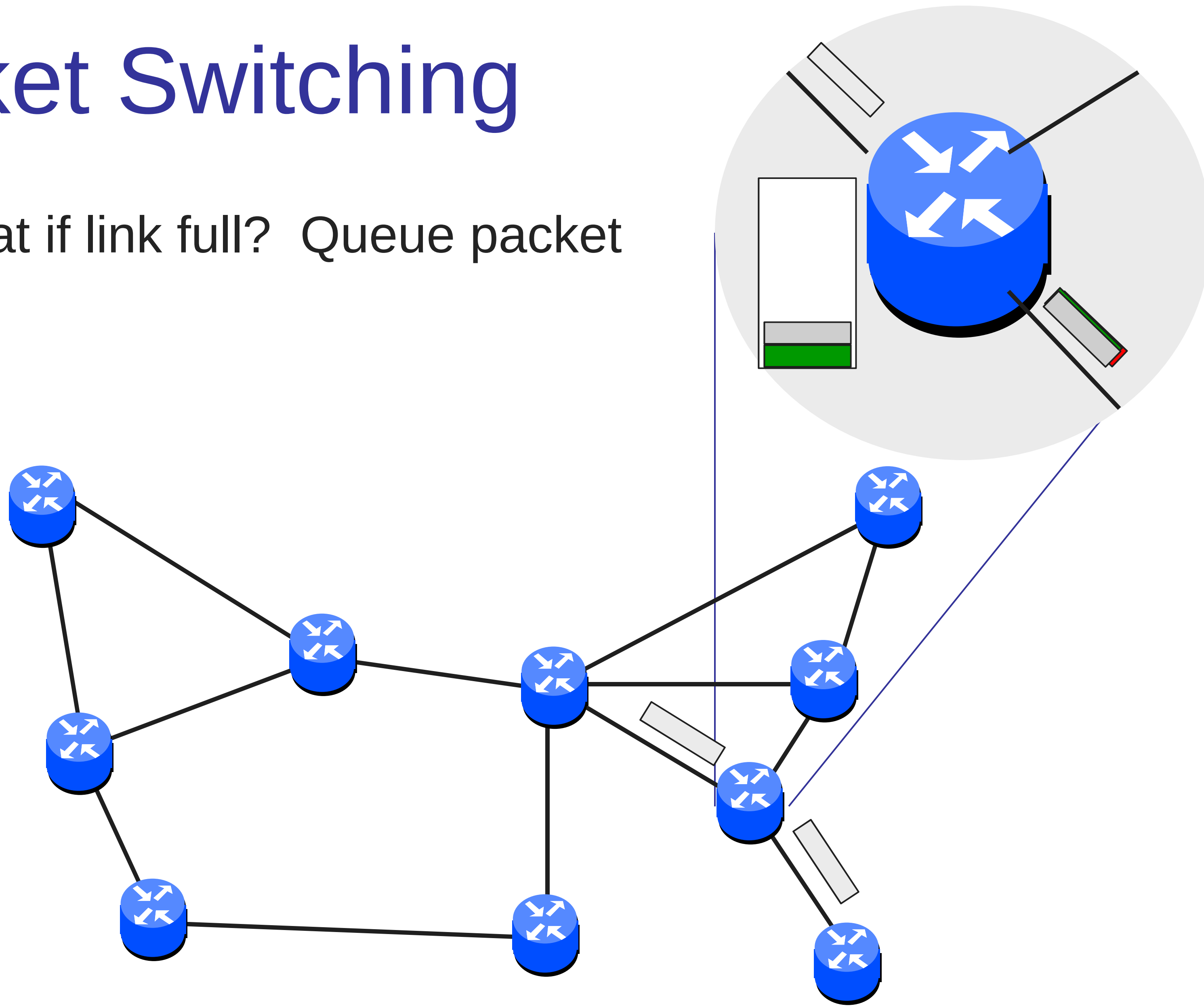
Packet Switching

- What if link full?



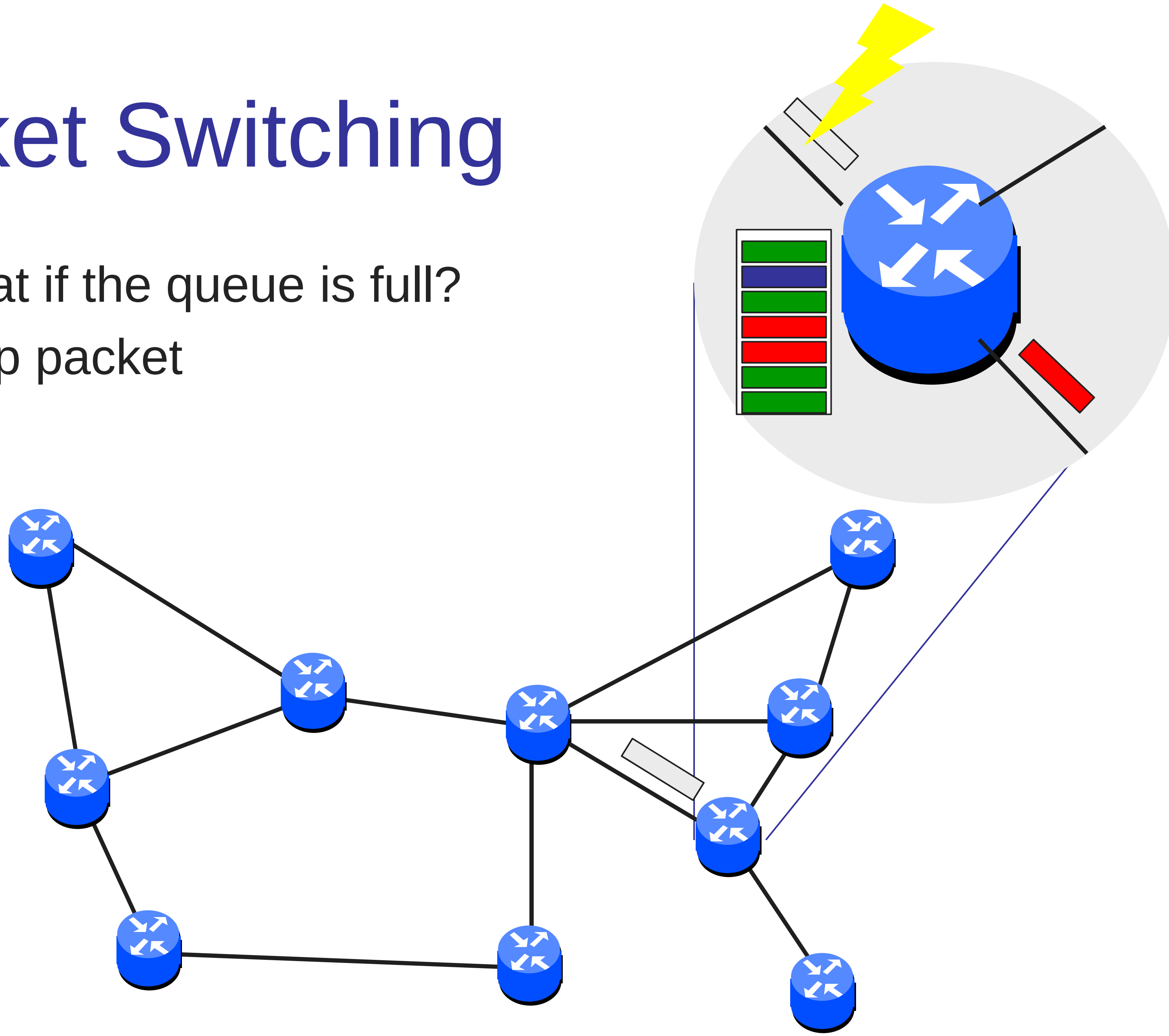
Packet Switching

- What if link full? Queue packet



Packet Switching

- What if the queue is full?
- Drop packet



Pros/cons?

- Circuit switching
  - Pro: If you get a circuit you have guaranteed bandwidth
  - Con: latency to set up circuit (one round-trip time)
  - Con: you may not get a circuit
  - Con: if you get a circuit and don't use it, bandwidth is wasted

- Packet switching
  - Pro: can send immediately (not required latency)
  - Pro: can share bandwidth dynamically among users (**statistical multiplexing**)
  - Con: available bandwidth per user can fluctuate, packets can be dropped

Why do you think the Internet is based on packet switching?

- Internet applications are bursty
  - E-mail only consumes bandwidth when mail send/recvs
  - Web browsing only consumes bandwidth when you visit site

- Packet switching is a much more efficient way to support bursty applications

- We will primarily focus on packet switching in this class since this is most of modern data networking

Protocols and Layering

- What’s a protocol?
- Organizing protocols via *layering*
- Encoding layers in packets
- The OSI & Internet layering models
- The end-to-end argument

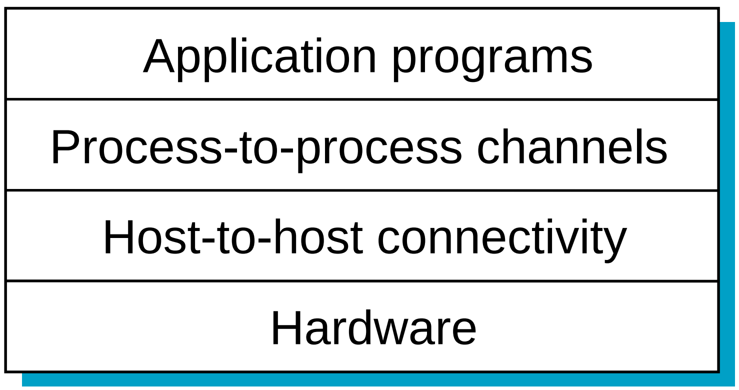
Definitions

- **Service**: A particular networking function (e.g. reliable message delivery)
- **Protocol**: An implementation of a service (e.g. TCP)
- **Interface**: How a protocol is manipulated by applications (e.g. packet format)

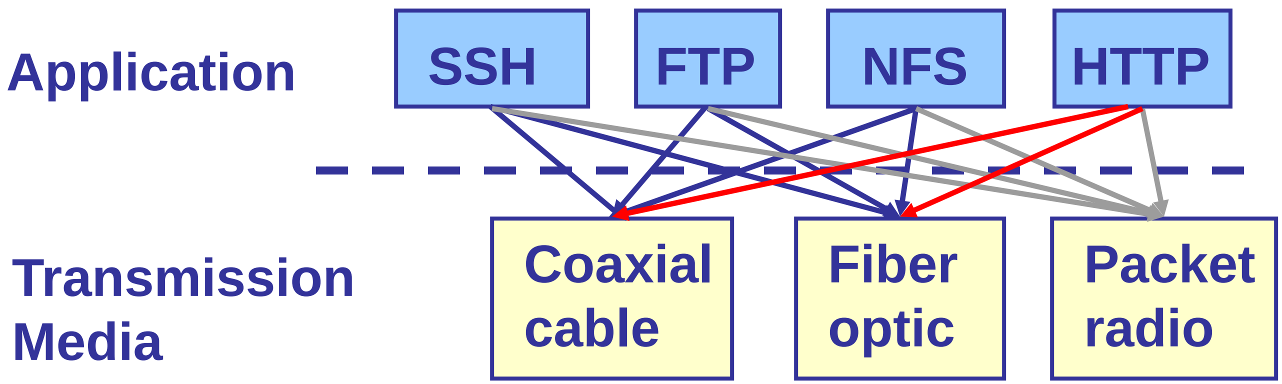
- **Layering**
  - Technique for organizing protocols into an **ordered** series of distinct abstractions
  - The services provided by a layer depend **only** on the services provided by the previous, less abstract, layer

Layered network system

- Start with services offered by hardware
- Add sequence of layers that provide higher (more abstract) level of service
- Example layered network system:



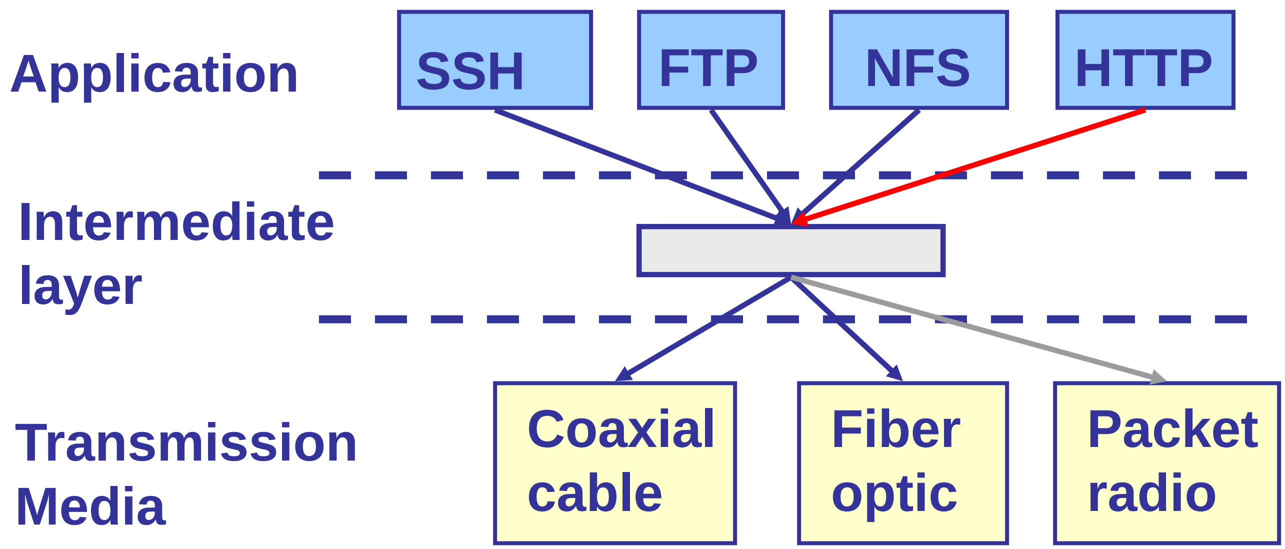
Why layering?



- Example: without layering each new application has to be re-implemented for every network technology

Why layering?

- Solution: introduce an intermediate layer that provides a **unique** abstraction for various network technologies



Benefits of layering

- **Encapsulation**
  - Functionality inside a layer is self-contained; one layer doesn't need to reason about other layers
  - Decomposes problem of building network into more manageable components
- **Modularity**
  - Can replace a layer without impacting other layers
  - Lower layers can be reused by higher layers
    - e.g. TCP and UDP both are layered upon IP

- One obvious drawback
  - Information hiding can produce **inefficient implementations**

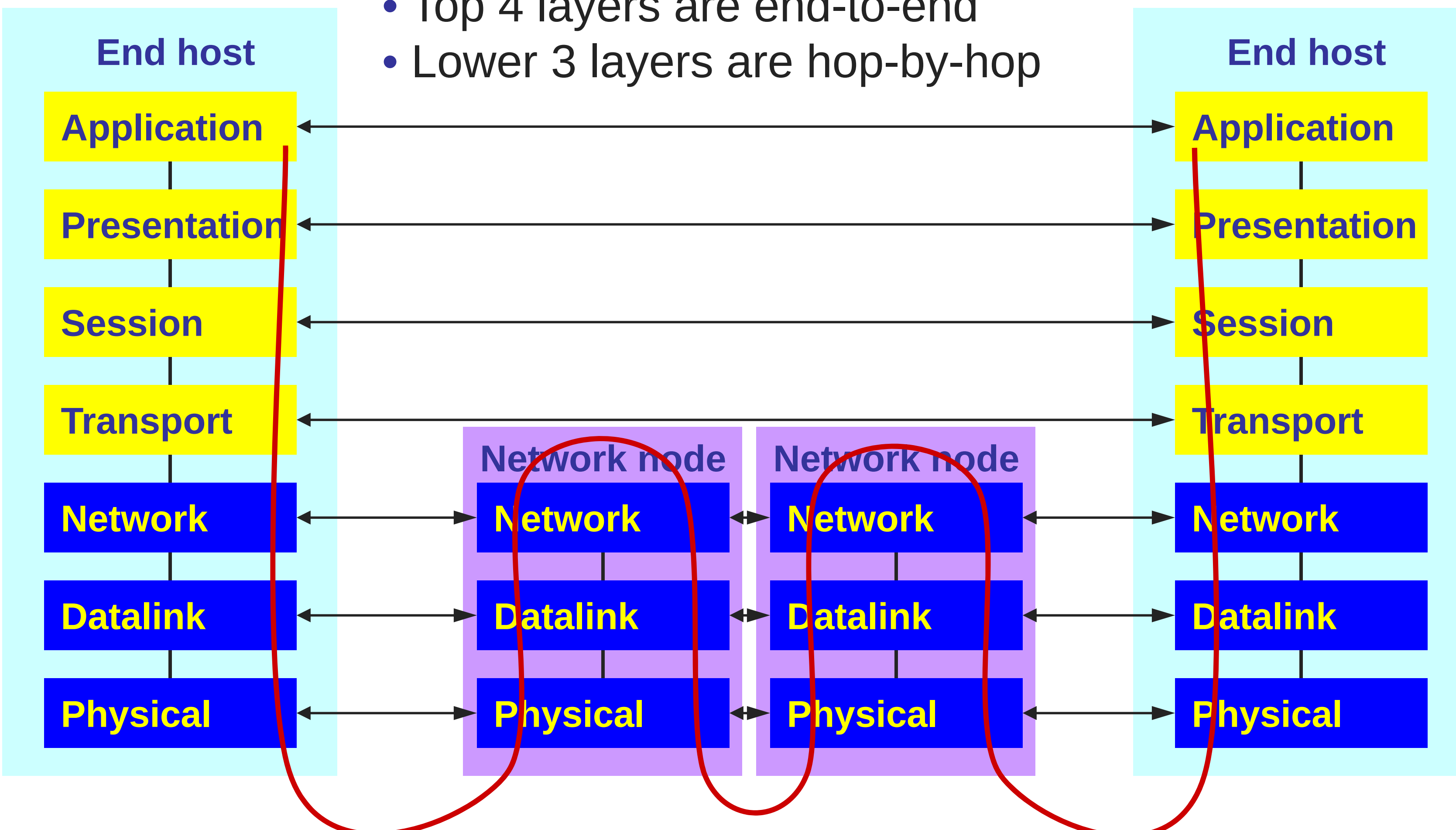
Who decides what goes in the layers?

- ISO OSI network architecture/reference model
  - ISO – International Standard Organization
  - OSI – Open Systems Interconnection
  - Designed by committee in 1978
  - Goal: open standard to support a market for vendors to compete on protocol implementation and design

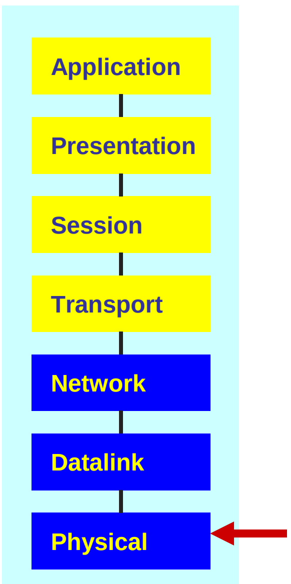
- Internet architecture
  - Backporting of experience from ARPAnet (1969) and TCP/IP protocols (1974)
  - Shares much of OSI design
  - Roughly managed by the Internet Engineering Task Force (IETF)

The OSI layering model

- Top 4 layers are end-to-end
- Lower 3 layers are hop-by-hop



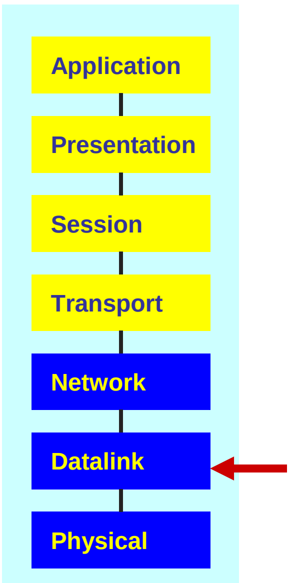
Physical Layer (1)



- **Service**: move the information between two systems connected by a physical link
- **Interface**: specifies how to send a bit
- **Protocol**: coding scheme used to represent a bit, voltage levels, duration of a bit

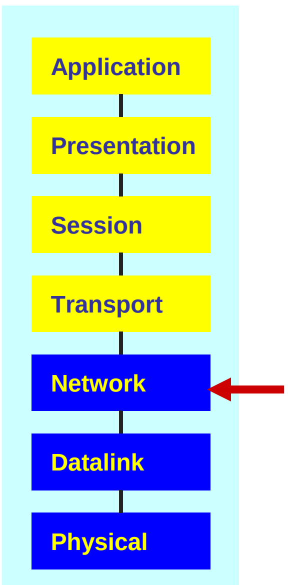
- Examples: coaxial cable, optical fiber links, transmitters, receivers

Datalink Layer (2)



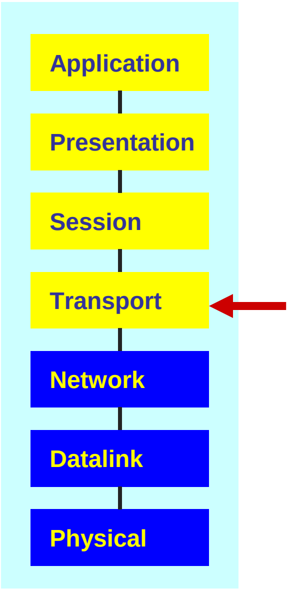
- **Service**:
  - *Framing*: where piece of data begins and ends
  - *Local addressing and delivery*: send data frames between peers attached to the same physical media
  - Others (sometimes):
    - » Shared media access
    - » Reliable transmission (resend missing packets)
- **Interface**: send a data unit (packet) to a machine connected to the *same* physical media
- **Protocol**: MAC addresses, media access control (MAC) implementation
- Examples: Ethernet, 802.11

Network Layer (3)



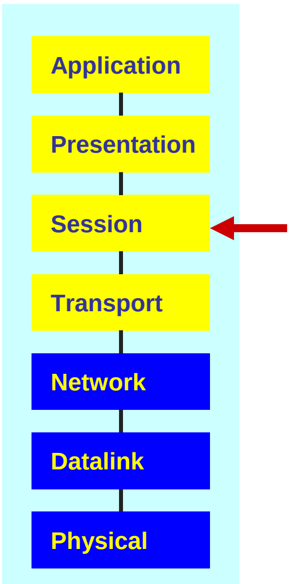
- **Service**:
  - Deliver a packet to specified destination
  - Perform segmentation/reassemble (fragmentation/defragmentation)
  - Sometimes:
    - » Packet scheduling: order packets are sent
    - » Buffer management: what if there are too many packets?
- **Interface**: send a packet to a given destination
- **Protocol**: global unique addresses, construct routing tables, forward packets towards destination
- Examples: Internet Protocol (IP)

Transport Layer (4)



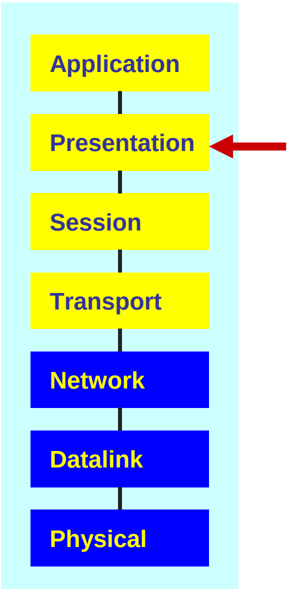
- **Service**:
  - Provide an error-free and flow-controlled end-to-end connection
  - Multiplex multiple transport connections to one network connection
- **Interface**: send a packet to specific destination
- **Protocol**: implement reliability and flow control
- Examples: TCP and UDP

Session Layer (5)



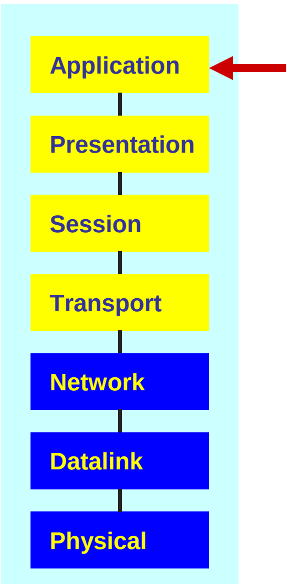
- **Service**
  - Session management
  - Synchronization, e.g., between audio/video streams
- **Interface**: depends on service
- **Protocols**: full duplex connection setup/teardown, restart and checkpointing, inter-session synchronization
- Examples: SMIL

Presentation Layer (6)



- **Service**: data format conversion
- **Interface**: depends on service
- **Protocol**: define data formats, and rules to convert from one format to another
- Examples: NFS’s XDR, various platform independent remote procedure call interfaces

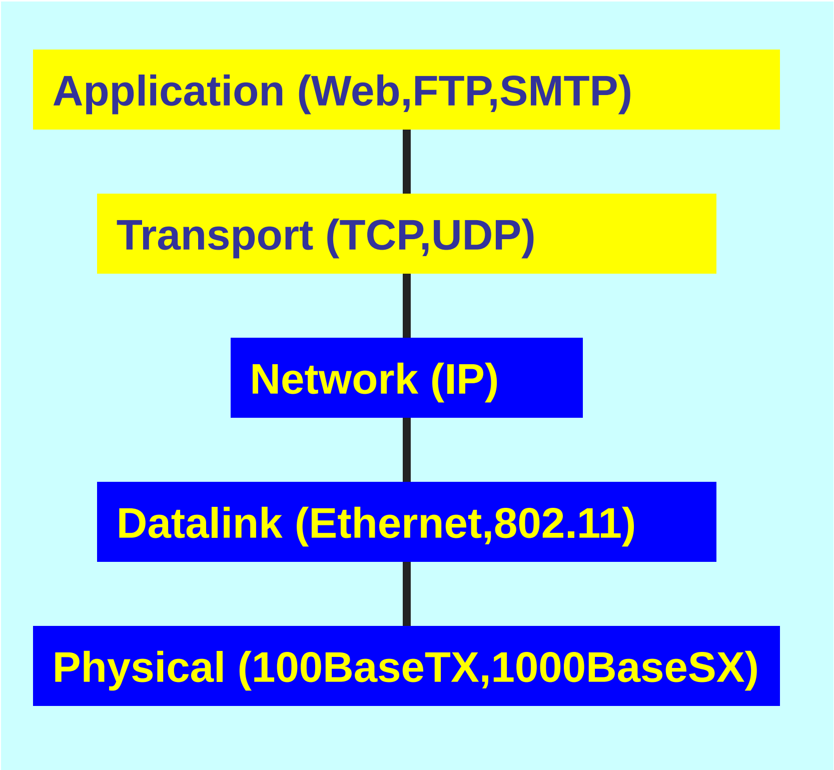
Application Layer (7)



- **Service**: any service provided to the end user
- **Interface**: depends on the application
- **Protocol**: depends on the application

- Examples: SMTP, BitTorrent, SSH, HTTP (Web)

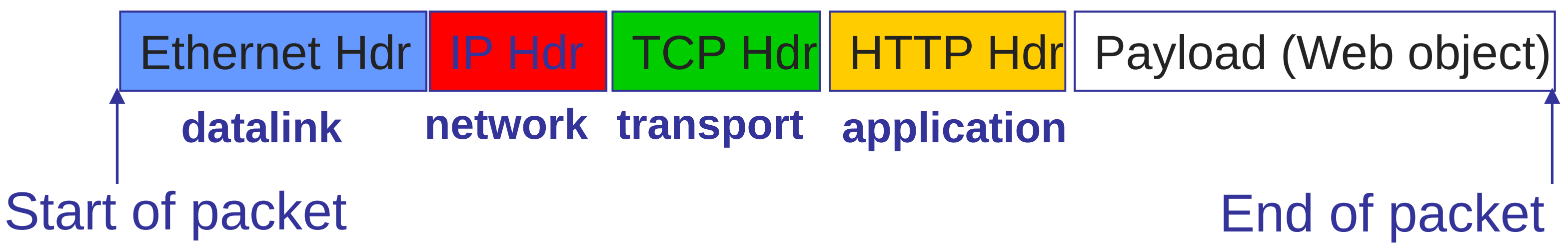
The Internet layering model



- So-called “hourglass” model
  - One network layer protocol
  - More diversity at other layers
- No presentation or session layers
- Standards driven by implementations

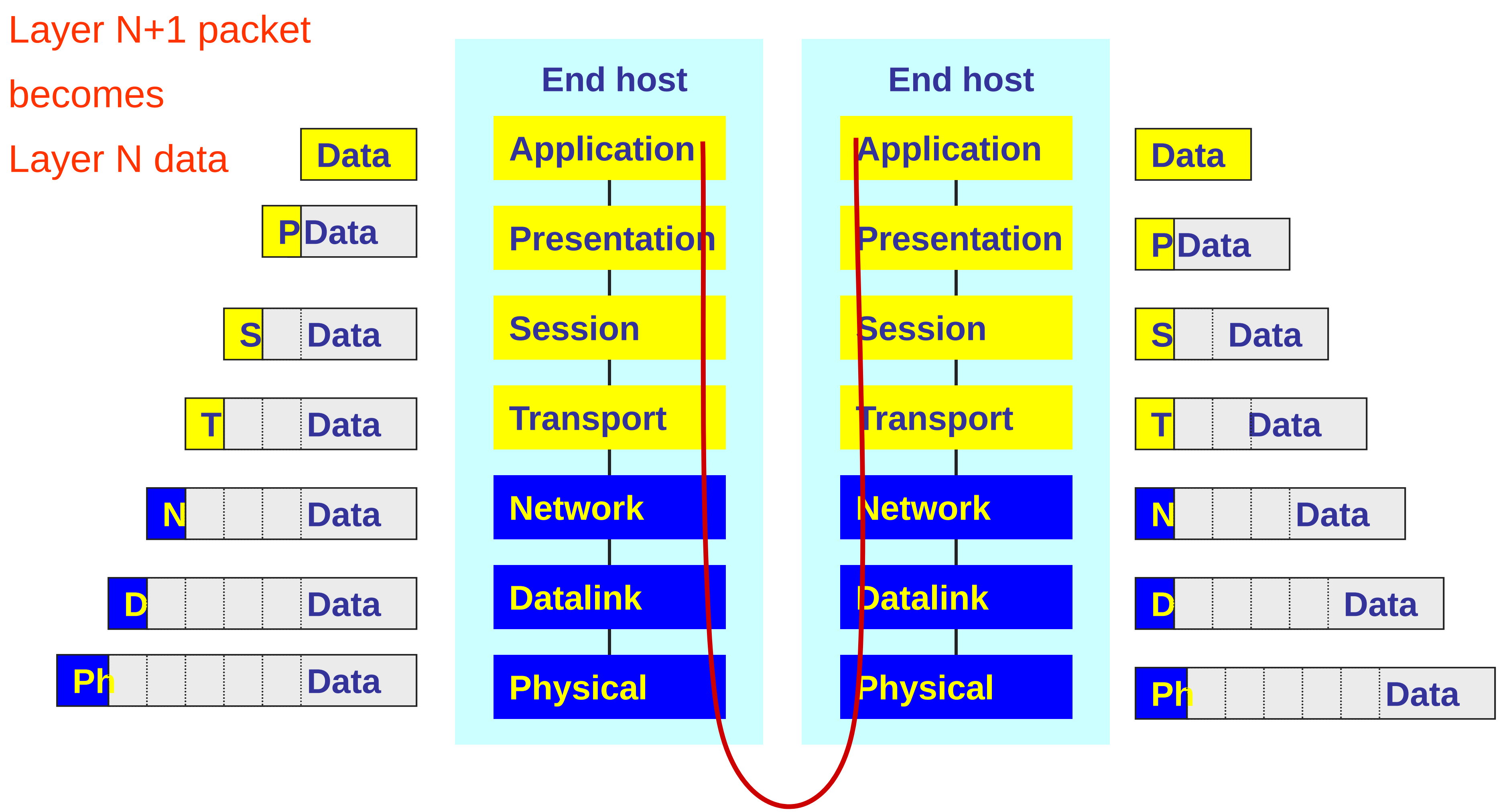
Layer encapsulation via *packet headers*

- Typical Web packet



- Notice that layers add overhead
  - Space (headers), effective bandwidth
  - Time (processing headers, “peeling the onion”), latency

Layer encapsulation via *packet headers*



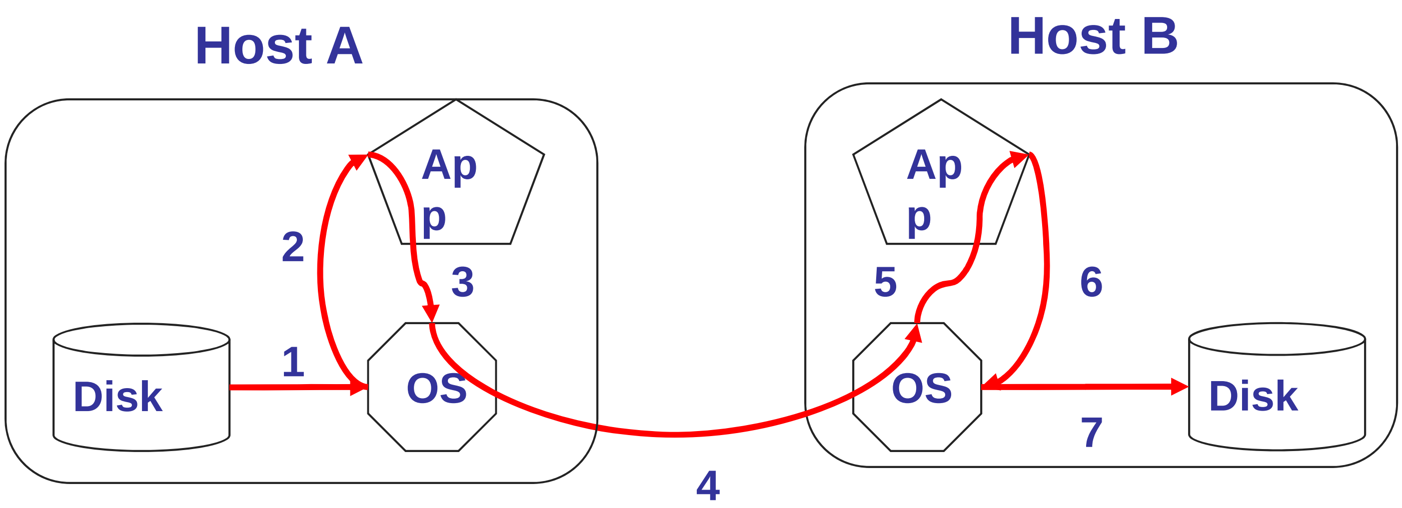
Key Design Decision

- How do you divide functionality across the layers?

- **End-to-end argument [Saltzer84]**
  - Functionality should be implemented at a lower layer iff it can be **correctly** and **completely** implemented there
  - Incomplete versions of a function can be used as a performance enhancement, but not for correctness

- Early, and still relevant, example
  - ARPAnet provided reliable link transfers between switches
  - Packets could still get corrupted on host-switch link, or inside of the switches
  - Hence, still need reliability at higher layers

Example:
Reliable File Transfer



- Where can data be corrupted/lost?
- Where to check if data has been corrupted/lost?
- Is there any value in lower-layer reliability?

Violating the E2E argument for performance optimization

- Functionality at lower layer can enhance performance
  - Not required for correct operation
  - Can be required for reasonably efficient operation

- Example: transport layer is responsisble for reliability, but should we add reliability to the datalink layer too?
  - N hops (average hops on Internet route = 15 hops)
  - Prob (corrupted/lost packet per link) = p
  - Prob (packet corrupted/lost end to end)
    - If p = 0.0001% -> Prob(e2e loss) = 0.0015%
    - If p = 1% -> Prob(e2e loss) = 15%
  - Reasonable to implement additional reliability in the datalink layer for the 2nd case

Optimization trade-offs

- Higher layers have more semantic information about service needs
  - (e.g. video: bits comprising MPEG I frames are much more important that bits in MPEG B frames)
- Lower layers have more information about true capabilities
  - (e.g. packet size, bandwidth, error rate)
- This tension is the subject of countless papers…

- Layer interactions:
  - Where to put compression vs encryption?

Summary: layering

- Key technique to implement communication protocols; provides
  - Modularity
  - Abstraction
  - Reuse
- Key design decision: what functionality to put in each layer?

- For next time: read Chap 2-2.2
- We're going to cover signaling, coding and clock recovery…