CS4450

Computer Networks:
Architecture and Protocols

Lecture 4
- Packet Delays
- How the Internet works
- Three Architectural Principles

Spring 2018

Rachit Agarwal

Context for and Goals of Today's Lecture

- **Context:**
  - Today's lecture is going to be one of the hardest lectures
  - If you understand everything
    - There is something wrong!
- **Goals:**
  - How does the Internet work?
    - An end-to-end view
  - Three Principles

But, as usual, lets start with:

what we learnt last lecture

Recap: Challenges with Circuit switching (reservation)

- Handling failures
- Resource underutilization
- Blocked connections
- Connection set up overheads
- Per-connection state in switches (scalability problem)

Recap: Solution: Packet switching

- **Break data into smaller pieces**
  - **Packets!**

- **Transmit the packets without any reservations**
  - And, hope for the best

Recap: Packet switching summary

- **Goods:**
  - Easier to handle failures
  - No resource underutilization
    - A source can send more if others don't use resources
  - No blocked connection problem
  - No per-connection state
  - No set-up cost
- **Not-so-goods:**
  - Unpredictable performance
  - High latency
  - Packet header overhead

Recap: Deep dive into one link: packet delay/latency

- **Consists of six components**
  - Link properties:
    - Transmission delay
    - Propagation delay
  - OS internals:
    - Processing delay
    - Queueing delay
  - Traffic matrix and switch internals:
    - Processing delay
    - Queueing delay
- First, consider transmission, propagation delays
- Queueing delay and processing delays later in the course

Recap: Transmission and propagation delay

- **Transmission delay:**
  - Time taken to push all the bits of a packet into a link
  - **= Packet size / Link bandwidth**

- **Propagation delay:**
  - Time taken to move one bit from one end of the link to other
  - **= Link length / Speed of light**

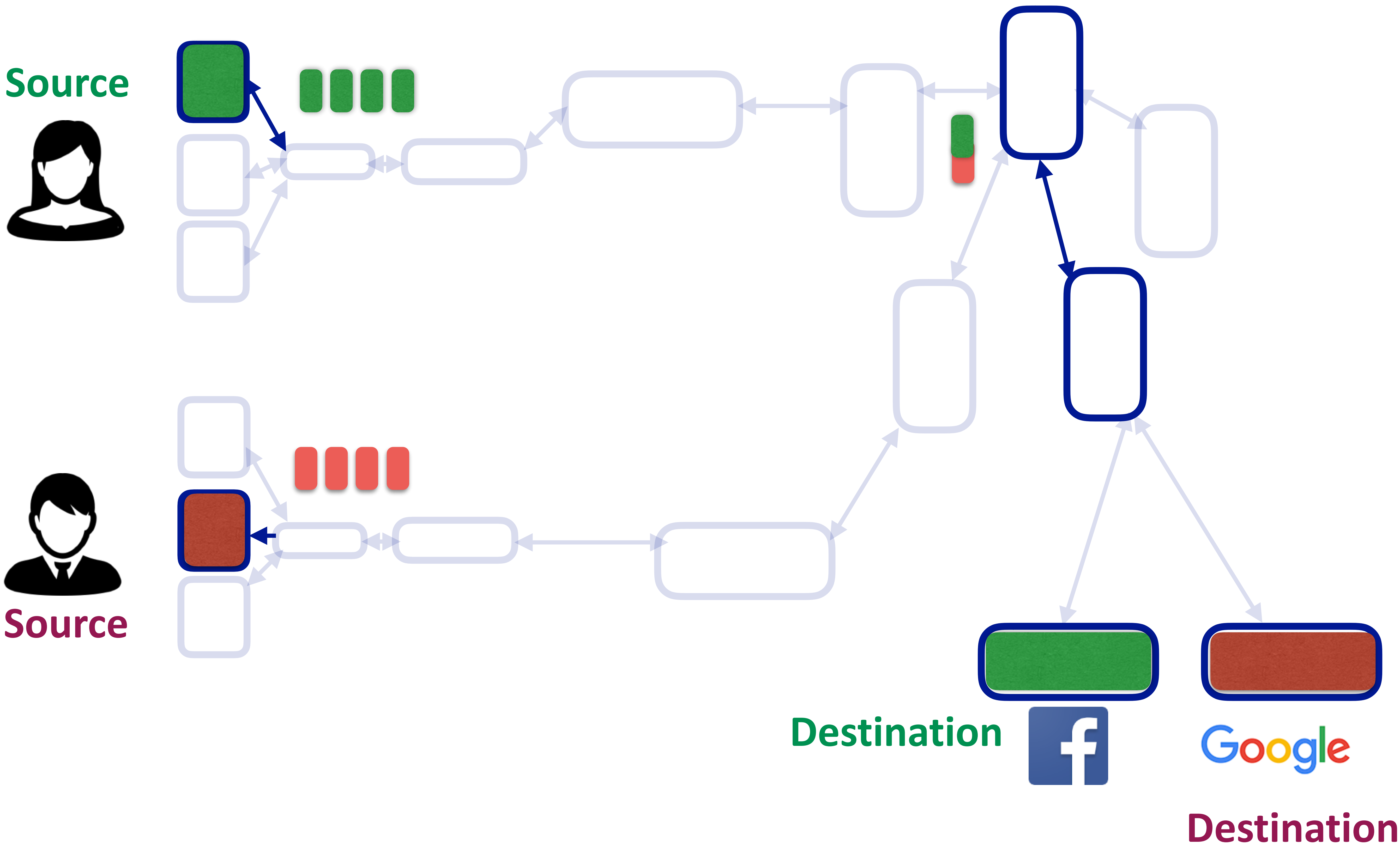
Questions?

Today's lecture

1. Dive into end-to-end: from source to destination
2. First look into switches: routing, queueing, forwarding
3. First look into network stack: sockets, ports, "the stack"
4. **Second look into the stack: layers**
5. Why layering?

First look into end-to-end

End-to-end: what mechanisms do we need?



Four fundamental problems!

- **Locating the destination**: Naming, addressing
- **Finding a path to the destination**: Routing
- **Sending data to the destination:** Forwarding
- **Reliability:** Failure handling

Four fundamental problems!

Naming, Routing, Forwarding, Reliability

- Each is motivated by a clear need
- The solutions are not always clean or deep
- **But if you keep in mind what the problem is**
    - You’ll be able to understand the solutions
    - When the right time comes :-)

Fundamental problem #1: Host Names and Addresses

- Network **Address: where host is located**
  - Requires an address for the destination host
    - can be multiple headers
- Network **Name: which host it is**
  - why?
- **When you move server to new building**
  - Name doesn’t change
  - Address does change
- **Same thing with your own name and address!**

- **Remember the analogy: human names, addresses, post office, letters**

Names versus addresses

- **Consider when you access a web page**
  - Insert URL into browser (eg, www.cornell.edu)
  - Packets sent to web site (reliably)
  - Packet reach application on destination host

- **How do you get to the website?**
  - URL is user-level name (eg, www.cornell.edu)
  - Network needs address (eg, where is www.cornell.edu)?

- Must map names to addresses
  - Just like we use an address book to map human names to addresses

Mapping Names to Addresses

- On the Internet, we only name hosts (sort of)
  - URLs are based on the name of the host containing the content (that is, www.cornell.edu names a host)

- Before you can send packets to www.cornell.edu, you must resolve names into the host's address

- Done by the **Domain Name System (DNS)**

The source knows the name;

Maps that name to an address using DNS!

Questions?

Fundamental problem #2

Routing to destination

- Given destination address, how does each switch/router know where to send the packet so that the packet reaches its destination

- When a packet arrives at a router
    - a **routing table** determines which outgoing link the packet is sent on

Routing protocols (conceptually)

- Distributed algorithm that runs between routers
  - Distributed means no single router has “full” view of the network
  - Exchange of messages to gather “enough” information …
- … about the network topology
- Compute paths through that topology
- Store forwarding information in each router
  - If packet is destined for X, send out link l1
  - If packet is destined for Y, send out link l2
  - Can packets going to different destinations sent out to same port?
- **We call this a routing table**

Questions?

Fundamental problem #3

Queueing and Forwarding of packets at switches/routers

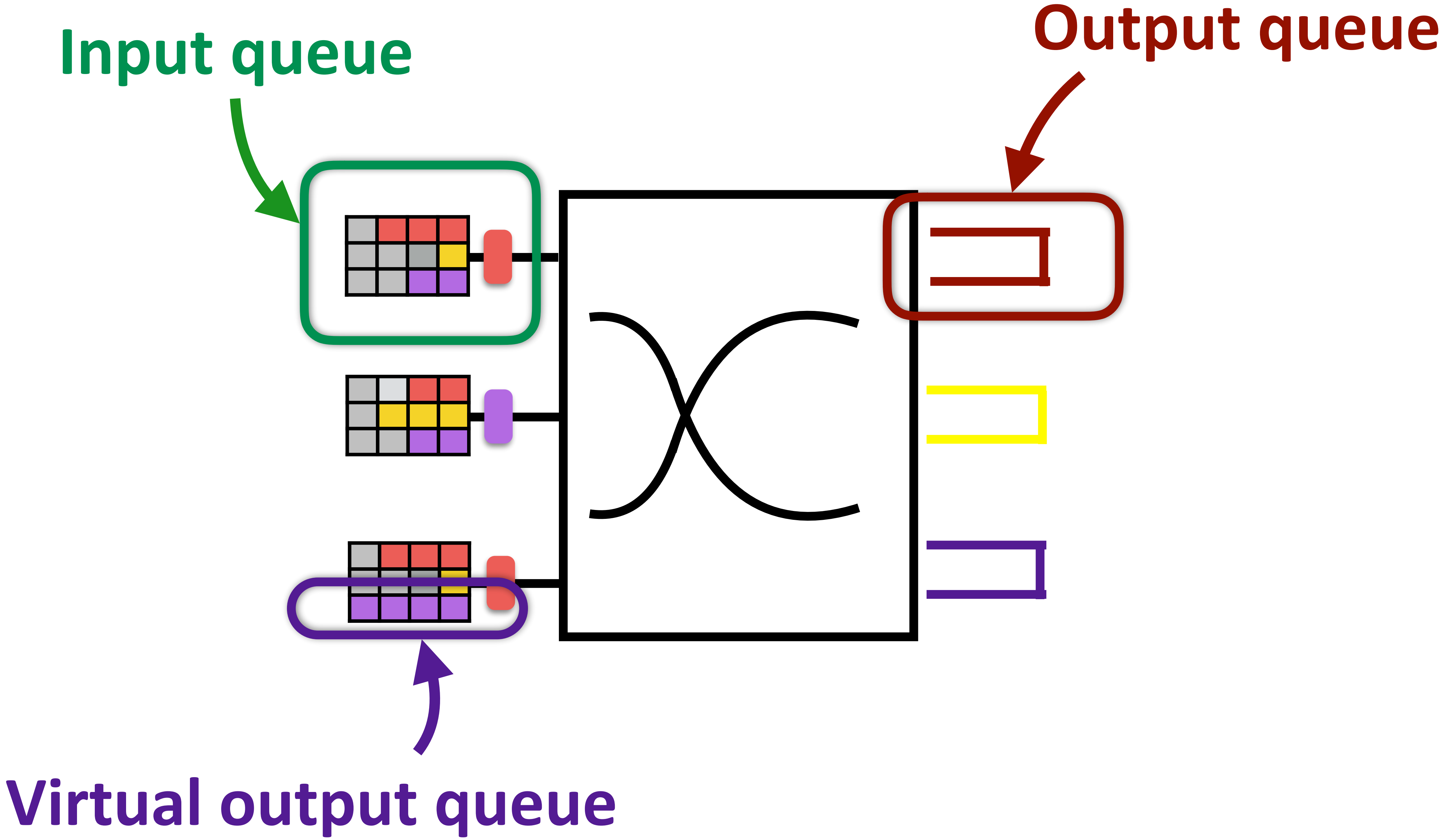
- **Queueing:** When a packet arrives, store it in “input queues”
    - Each incoming queue divided into multiple virtual output queues
    - One virtual output queue per outgoing link
    - When a packet arrives:
        - Look up its destination’s address (how?)
        - Find the link on which the packet will be forwarded (how?)
        - Store the packet in corresponding virtual output queue

- **Forwarding:** When the outgoing link free
    - Pick a packet from the corresponding virtual output queue
    - forward the packet!

What must packets carry to enable forwarding?

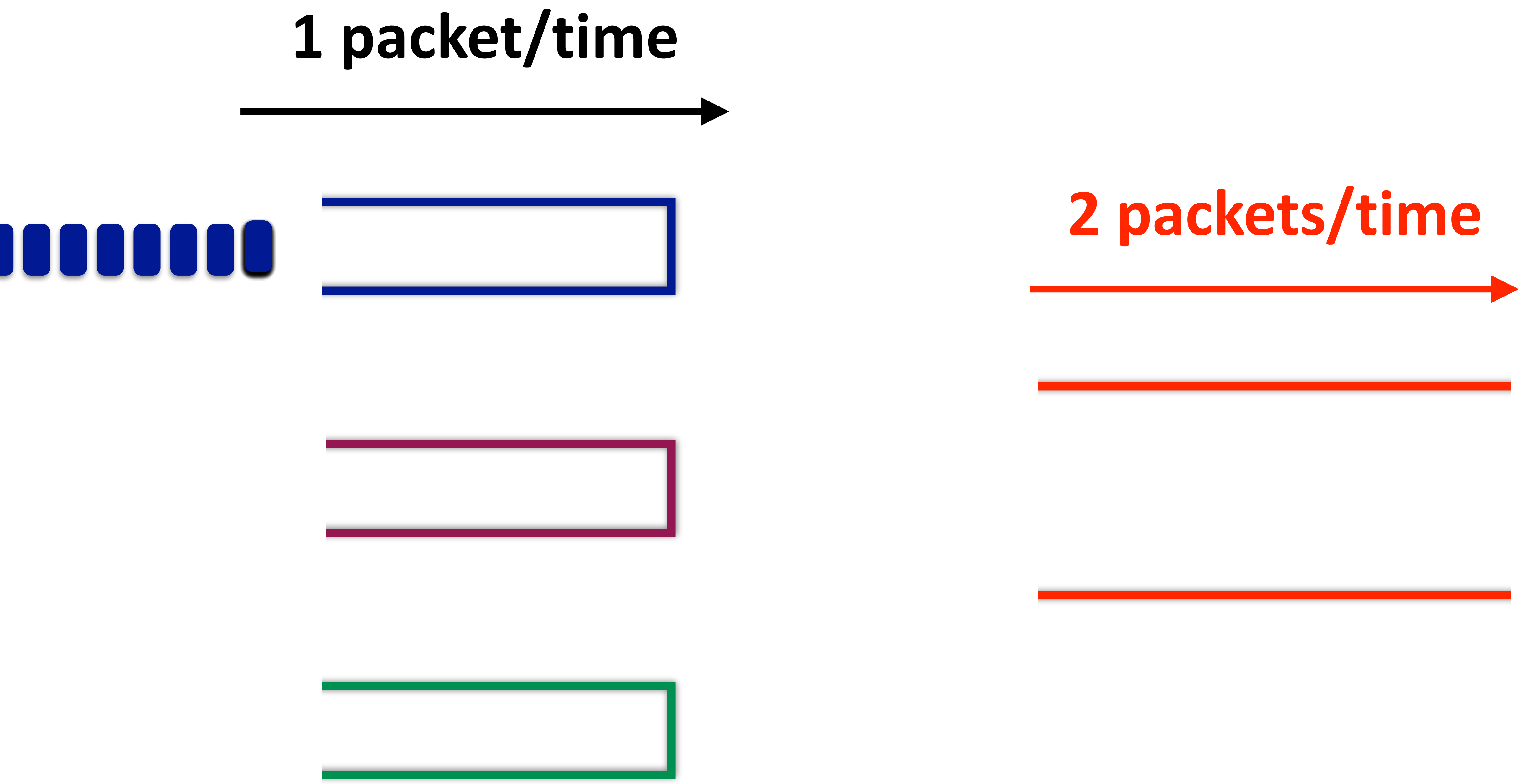
- **Packets must describe where it should be sent**
  - Requires an address for the destination
- **Packets must describe where its coming from**
  - For handling failures, etc.
  - Requires an address for the source
- **Packets must carry data**
  - can be bits in a file, image, whatever

What does a switch/router look like

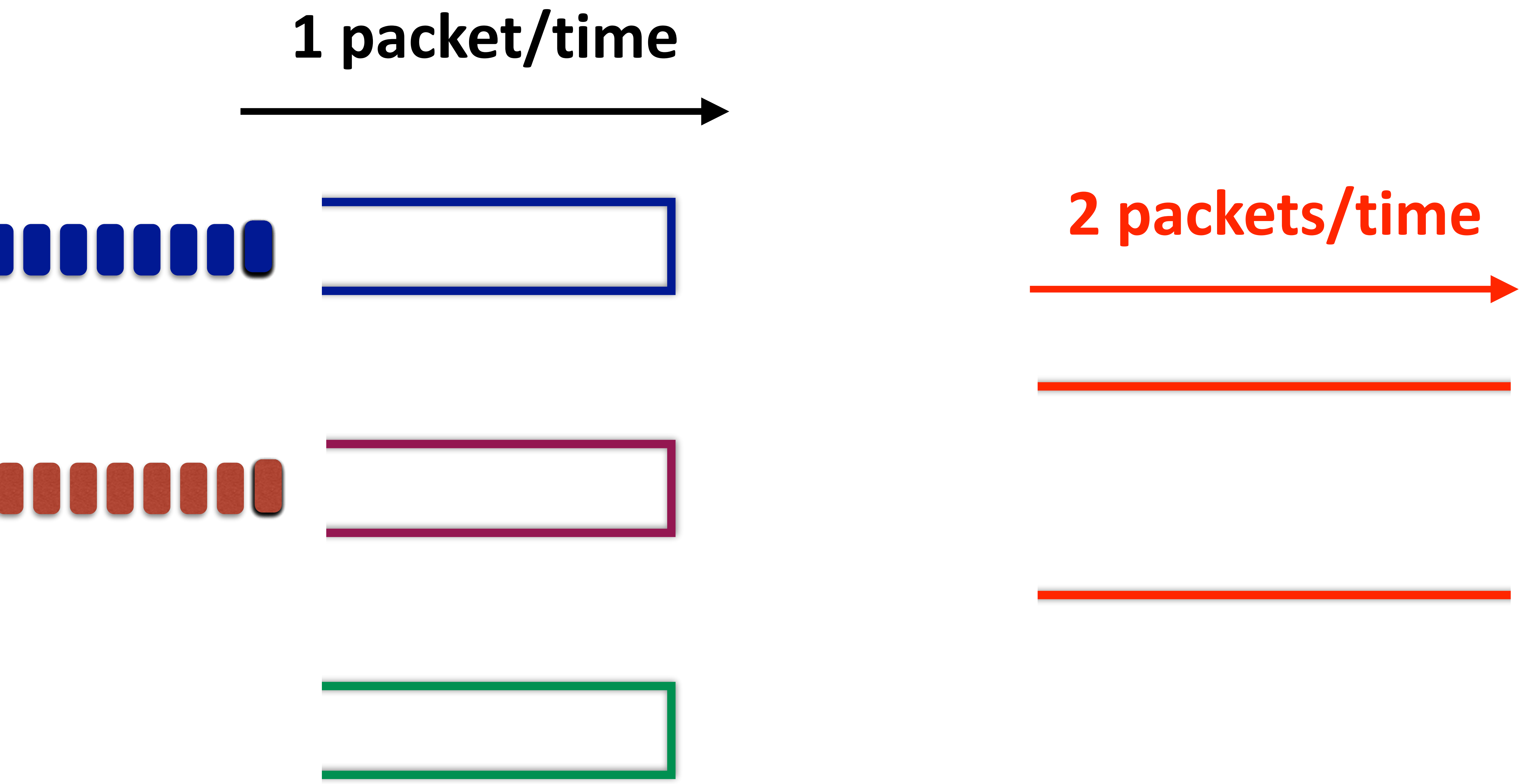


- **Each <u>input queue</u> could send packets to each output queue <u>at full rate</u>**
  - That is, a switch architecture is heavily parallelized
  - Can always focus on a single outgoing queue for design/analysis

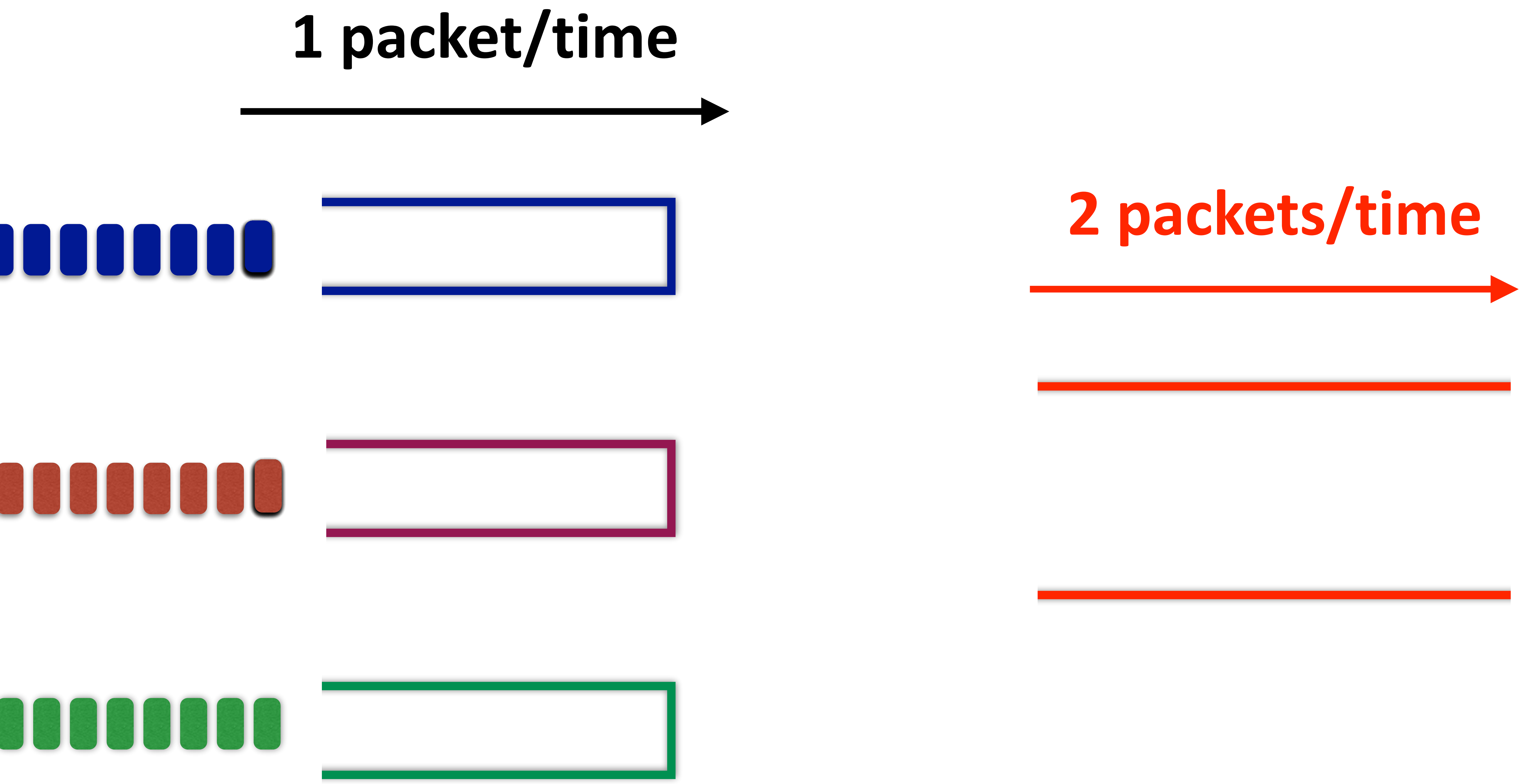
Queueing and processing delay: Case I (low load)



Queueing and processing delay: Case II (balanced load)



Queueing and processing delay: Case II (high load)



Queueing and processing delay

- **Processing delay**
  - Easy; each switch/router needs to decide where to put packet
  - Requires checking header, etc.
- **Queueing delay**
  - Depends on network load
  - As load increases, queueing delay increases
- **In an extreme case, increase in network load**
  - results in packet drops

Questions?

Fundamental problem #4

How do you deliver packets reliable?

- Packets can be dropped along the way
  - Buffers in router can overflow
  - Routers can crash while buffering packets
  - Links can garble packets

- How do you make sure packets arrive safely on an unreliable network?
  - Or, at least, know if they are delivered?
  - Want no false positives, and high change of success

Two questions about reliability

- **Who is responsible for this? (architecture)**
    - Network?
    - Host?
- **How is it implemented? (engineering)**
- We will consider both perspectives

Questions?

Finishing our story

- We now have the address of the web site
- And, a route/path to the destination
- And, mechanisms in place to forward the packets at each switch/router
- In a reliable manner
  - So, we can send packets from source to destination
  - Are we done?

- When a packet arrives at a host, what does the host do with it?
  - To which process (application) should the packet be sent?

- If the packet header only has the destination address, how does the host know where to deliver packet?
  - There may be multiple applications on that destination

And while we are finishing our story ….

- Who puts the source address, source port, destination address, destination port in the packet header?

The final piece in the game: End-host stack

Of Sockets and Ports

- When a process wants access to the network, it opens a socket, which is associated with a port

- **Socket:** an OS mechanism that connects processes to the networking stack

- **Port:** number that identifies that particular socket

- The port number is used by the OS to direct incoming packets

Implications for Packet Header

- **Packet Header must include:**
  - Destination address (used by network)
  - Destination port (used by network stack)
  - And?
  - Source address (used by network)
  - Source port (used by network stack)

- When a packet arrives at the destination host, packet is delivered to the socket associated with the destination port

- More details later

Separation of concerns

- **Network:** Deliver packets from host to host (based on address)
- **Network stack (OS):** Deliver packets to appropriate socket (based on port)
- **Applications:**
    - Send and receive packets
    - Understand content of packet bodies

Secret of the Internet's success is getting these and other abstractions right

Who cares?

- **Why is separation of concerns important?**
  - Separation of concerns ~ Modularity
- If each component’s task well-defined, one can focus design on that task
  - And replace it with any other implementation that does that task
  - Without changing anything else

What is Modularity

- Modularity is nothing more than decomposing programs/systems into smaller units.
  - **A clean "separation of concerns"**

- Plays a crucial role in computer science…

- … and networking

Modularity in Computer Science

“Modularity based on abstraction is the way to get things done”

- - Barbara Liskov

Computer System Modularity

- **Partition system into modules**
  - Each module has well defined interface
- **Interfaces give flexibility in implementation**
  - Changes have limited scope
- **Examples**
  - Libraries encapsulating set of functionalities
  - Programming language abstracts away CPU
- **The trick is to find the *right* modularity**
  - The interfaces should be long-lasting
  - If interfaces are changing often, modularity is wrong

Network System Modularity

- The need for modularity still applies
  - **And is even more important!** (***why?***)

- Network implementations not just distributed across many lines of code
  - Normal modularity “organizes” that code

- Networking is distributed across many machines
  - Hosts
  - Routers

Network Modularity Decisions

- How to break system into modules?
  - Classic decomposition into tasks
- Where are modules implemented?
  - Hosts?
  - Routers?
  - Both?
- Where is state stored?
  - Hosts?
  - Routers?
  - Both?

Leads to three design principles

- How to break system into modules
  - **Layering**

- Where are modules implemented
  - **End-to-End Principle**

- Where is state stored?
  - **Fate-Sharing**

Layering

Breakdown into tasks

- Bits on wire
- Packets on wire
- Deliver packets to hosts across local network
- Deliver packets to host across networks
- Deliver packets reliably, to correct process
- Do something with the data

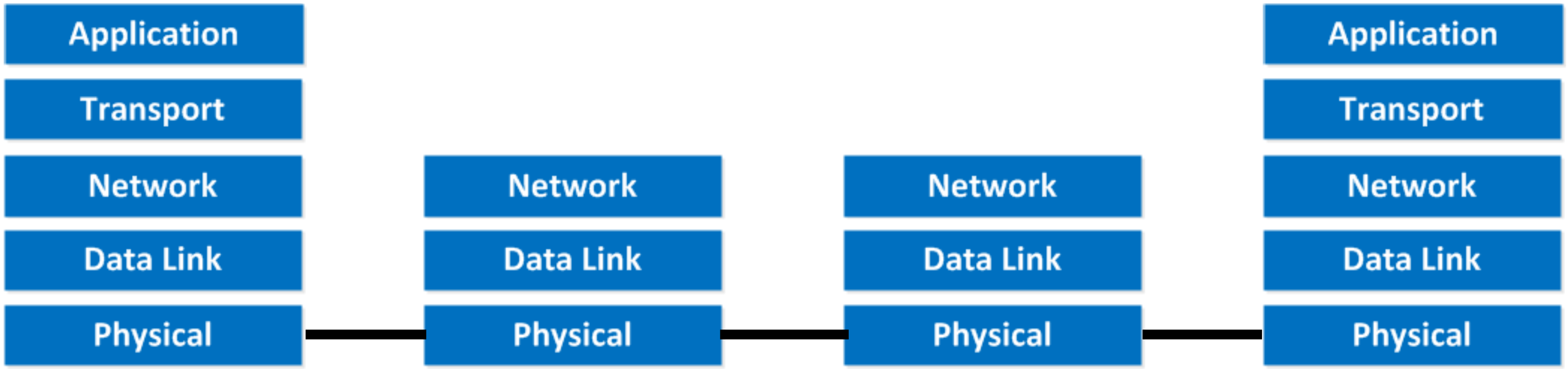
Resulting Modules (Layers)

- **Bits on wire (Physical)**
- Packets on wire
- **Deliver packets to hosts across local network (Datalink)**
- **Deliver packets to host across networks (Network)**
- **Deliver packets reliably, to correct process (Transport)**
- **Do something with the data (Application)**

Five Layers (Top - Down)

- **Application:** Providing network support for apps
- **Transport (L4):** (Reliable) end-to-end delivery
- **Network (L3):** Global best-effort delivery
- **Datalink (L2):** Local best-effort delivery
- **Physical:** Bits on wire

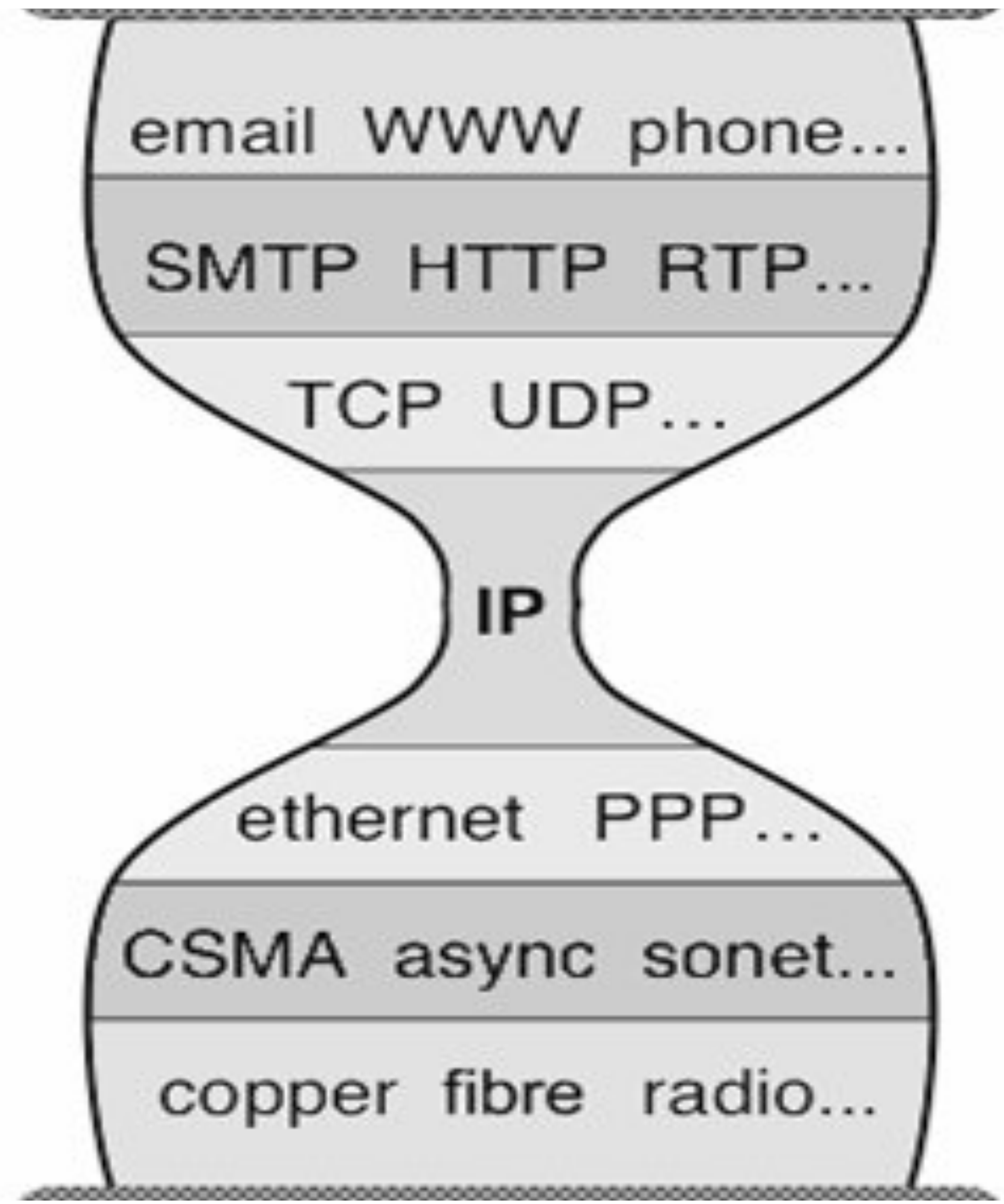
Layering



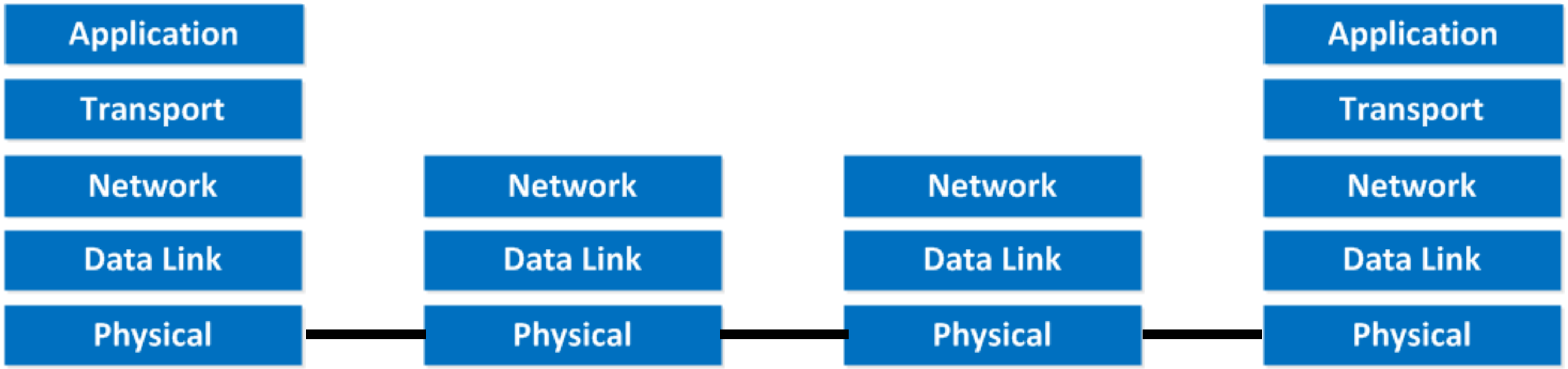
- **A kind of modularity**
  - Functionality separated into layers
  - Layer n **interfaces with only layer n-1**
    - Hides complexity of surrounding layers
    - Evolution of “modules”
  - (IP) Connectivity becomes a commodity

Three Observations

- Each layer:
  - Depends on the layer below
  - Supports layer above
  - Independent of others
- Multiple versions in layer
  - Interfaces differ somewhat
  - Components pick which lower-level protocol to use
- But only one IP layer
  - Unifying protocol



Layering and Innovation



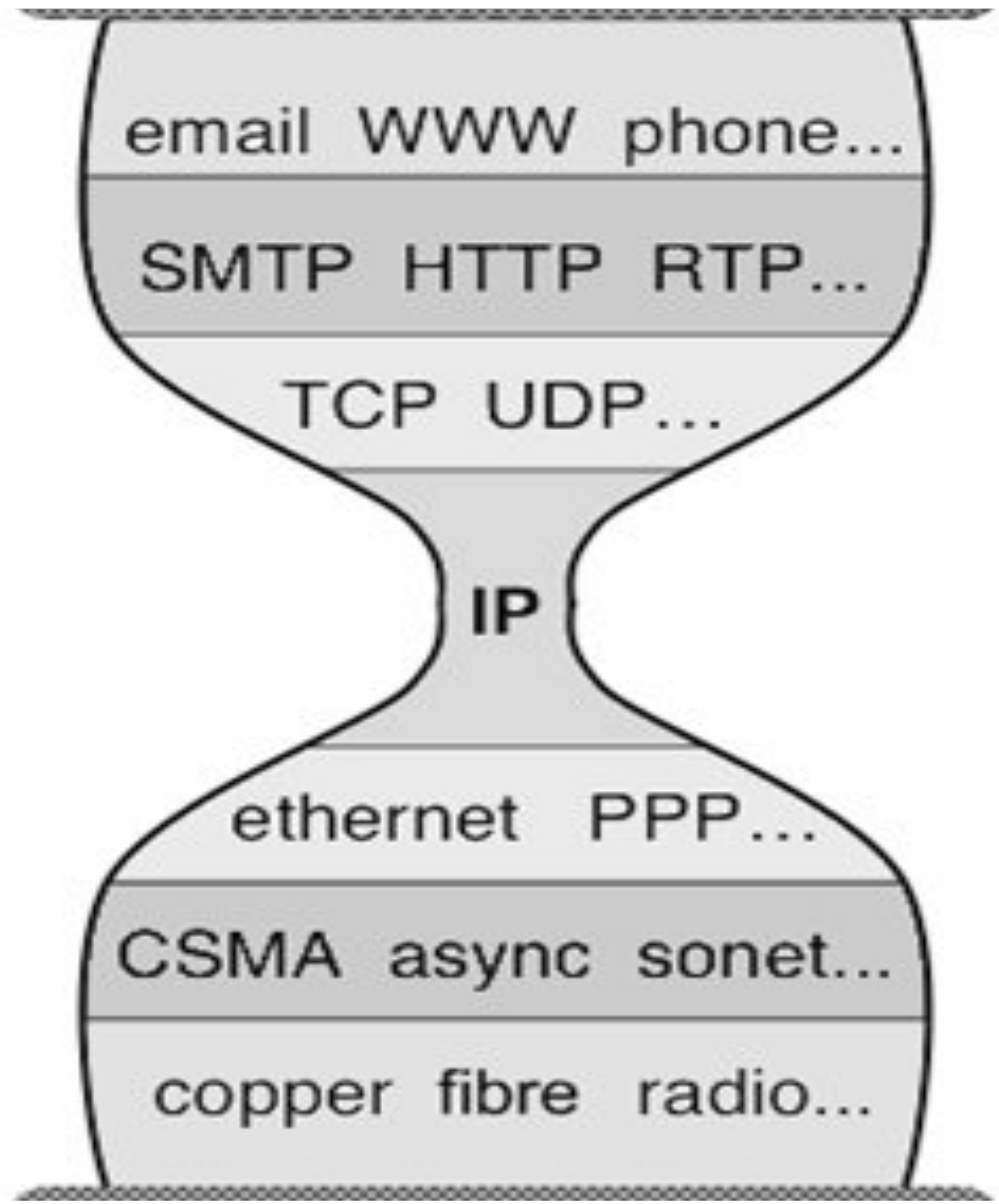
Layering "modularized" the Internet architecture with **flexible open interfaces** which helped spur innovation

Layering crucial to Internet's success

- Innovation at most levels:
  - Applications (lots)
  - Transport (few)
  - Datalink (few)
  - Physical (lots)

- Innovation proceeded largely in parallel
  - **Payoff of modularity!**

- Pursued by very different communities
  - Like systems and chip designers



Questions?

Three Internet Design Principles

- How to break system into modules?
  - Layering
- Where are modules implemented?
  - **End-to-End Principle**
- Where is state stored?
  - Fate-Sharing

Distributing Layers across Network

- Layers are simple if only on a simple machine
  - Just stack of modules interacting with those above/below
- But we need to implement layers across machines
  - Hosts
  - Routers (Switches)
- What gets implemented where?

What gets implemented on Host?

- Bits arrive on wire, must make it up to application

- Therefore, all layers must exist at host!

What gets implemented on Router?

- Bits arrive on wire
  - Physical layer necessary

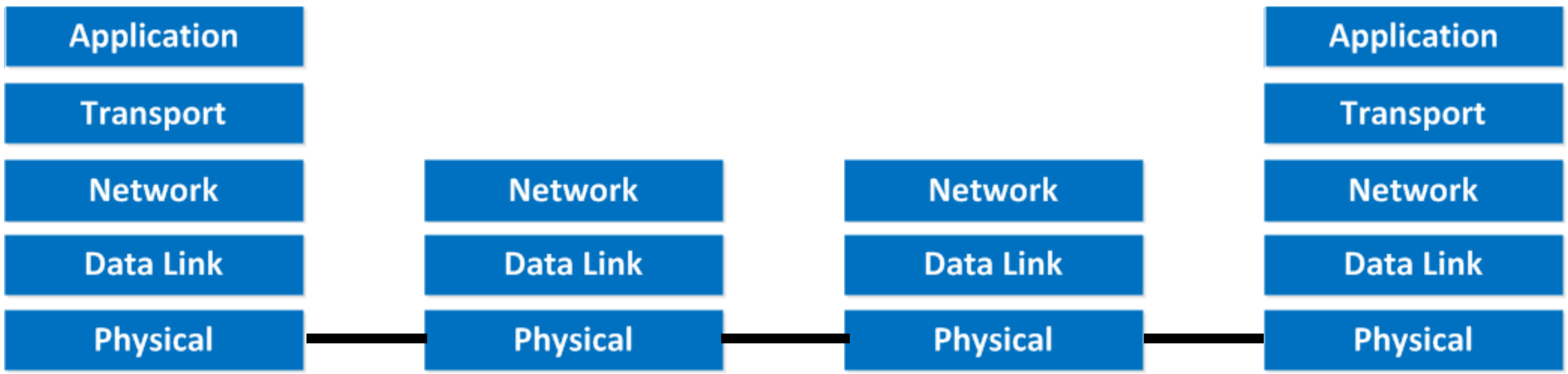
- Packets must be delivered to next hop
  - Datalink layer necessary

- Routers participate in global delivery
  - Network layer necessary

- Routers do not support reliable delivery
  - Transport layer (and above) <u>***not***</u> supported

Simple Diagram

- Lower three layers implemented everywhere
- Top two layers only implemented at hosts



But why implemented this way?

- Layering doesn't tell you **what services each layer should provide**
- What is an effective division of responsibility between various layers?

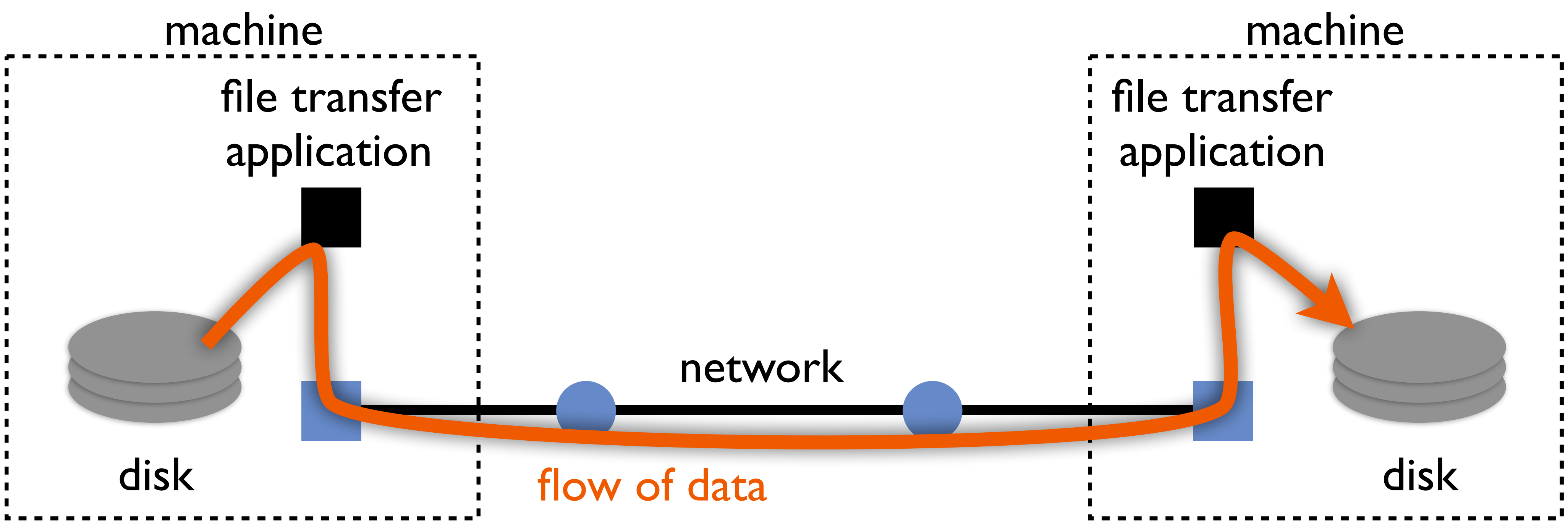
End-to-end Principle

If a function can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system,

then providing that function as a feature of the communication system itself is not possible.

Sometimes providing an incomplete version of that function as a feature of the communication system itself may be useful as a performance enhancement.

End-to-end Principle: an example



Suppose the link layer is reliable. Does that ensure reliable data transfer?

Suppose the network layer is reliable. Does that ensure reliable data transfer?

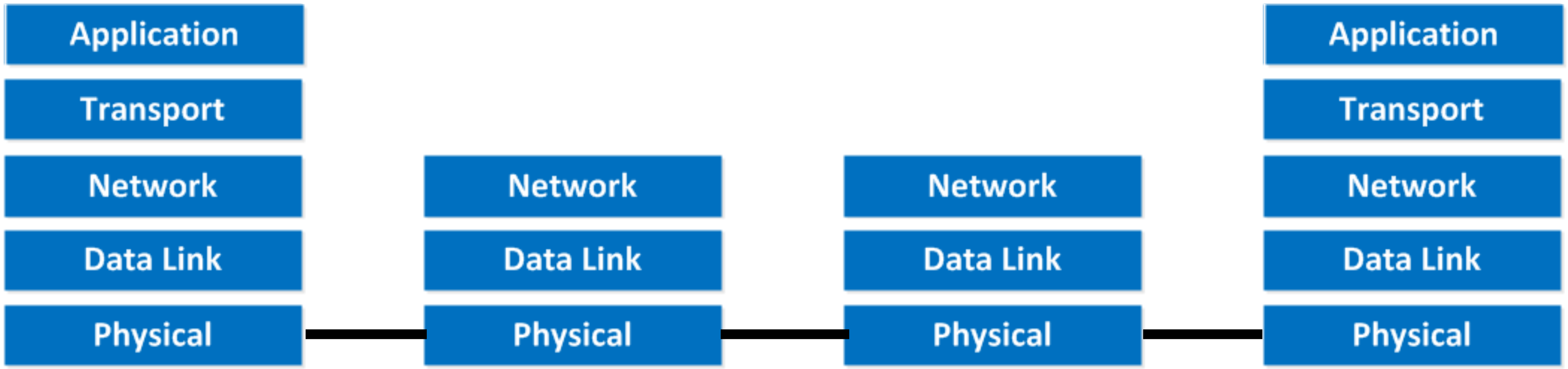
End-to-end Principle (Interpretation)

Assume the condition (IF) holds. Then,

- **End-to-end implementation**
  - Correct
  - Generalized, and simplifies lower layers
- **In-network implementation**
  - Insufficient
  - May help — or hurt — performance

Examples? Contradictions?

End-to-end Principle (Interpretation)



What does the end mean?

Group Exercise 4

Where shall we implement the following?

- Failure avoidance?
- Failure reaction?
- Routing?
    - Topology discovery?
    - Path Selection?
- Security?
- Network management?
- Resource management?

Summary

- Where to implement functionality is complicated
  - No right or wrong answer

- But everyone agrees that reliability does not belong in the network

- Multicast is a good test case

Questions?

Three Internet Design Principles

- How to break system into modules?
  - Layering

- Where are modules implemented?
  - End-to-End Principle

- **Where is the state stored?**
  - **Fate-sharing**

Fate-Sharing

- Note that E2E principle relied on "fate-sharing"
  - Invariants only break when endpoints themselves break
  - Minimize the dependence on other network elements

- This should dictate placement of storage

General Principle: Fate-Sharing

- When storing state in a distributed system, colocate it with entities that rely on that state

- Only way failure can cause loss of the critical state is if the entity that cares about it also fails …
  - … in which case it doesn't matter

- Often argues for keeping network state at end hosts rather than inside routers
  - E.g., packet-switching rather than circuit-switching

Decisions and their Principles

- How to break system into modules
  - **Dictated by layering**
- Where modules are implemented
  - **Dictated by End-to-End Principle**
- Where state is stored
  - **Dictated by Fate Sharing**

Today's lecture

- The Internet is a huge, complicated system
- One can study the parts in isolation
  - Routing
  - Ports, sockets
  - Network stack
  - …
- But the pieces all fit together in a particular way
- Today was quick overview of how pieces fit…
  - Don't worry if you didn't understand much of it
  - **You probably absorbed more than you realize**