CS4450

Computer Networks:
Architecture and Protocols

Lecture 4
- Three Architectural Principles

Rachit Agarwal

Announcements

- Please fill out the poll for deciding the office hours by tonight.

Quick recap from last lecture

Recap: four fundamental problems!

- **Locating the destination**: Naming, addressing
  - Mapping of names to addresses using **Domain Name System**

- **Finding a path to the destination**: Routing
  - Distributed **routing algorithms** compute and store **routing tables**

- **Sending data to the destination:** Forwarding
  - **Input queues, virtual output queues, output queues**
  - Enablers: Packet header (address), and routing table (outgoing link)

- **Reliability:** Failure handling
  - Not much discussion, but **the** question: **hosts or networks?**

Recap: the final piece in the story — Host network stack

Of Sockets and Ports

- When a process wants access to the network, it opens a **socket**
    - an OS mechanism that connects processes to the network stack
- Each socket is associated with a **Port**
    - The port number is a unique number that identifies the socket
    - Used by the OS to direct incoming packets

Recap: Implications for Packet Header

- **Packet Header must include:**
  - Source and destination address (used by network)
  - Source and destination 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

Recap: the end-to-end story

- Application opens a **socket** that allows it to connect to the **network stack**
- Maps **name** of the web site to its **address** using **DNS**
- The network stack at the source embeds the address and **port** for both the source and the destination in **packet header**
- Each **router** constructs a **routing table** using a distributed algorithm
- Each router uses destination address in the packet header to look up the **outgoing link** in the routing table
    - And when the link is free, forwards the packet
- When a packet arrives the destination:
    - The network stack at the destination uses the port to forward the packet to the right application

Recap: Separation of concerns

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

Questions?

Context for Today's Lecture

- So far, we have discussed several high-level concepts
  - Network sharing
  - End-to-end working of the Internet
  - Addressing, Routing, Switch/Router functionality, etc.

- And, have dived deep into several topics:
  - Circuit switching and packet switching (especially the "why")
  - Delays (transmission, propagation)

- **You know more about computer networks than you may realize!**

- **Today:** **Lay the foundation for rest of the course**

Goals for Today's Lecture

- **Three architectural principles:**
  - Layering
  - End-to-end principle
  - Fate Sharing principle
- **We will come back to these over and over again**
  - Almost every lecture in the semester
- **Before we start, let me outrightly admit ….**
  - First time I learnt these, I said — what the @#$% ….
  - … there are easier ways to torture students!
  - **Now, these have become the guiding principles of my career!**

The key to Internet's success: Separation of concerns

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

The key to Internet's success: Separation of concerns

- **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

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 <u>distributed across many machines</u>
  - Hosts
  - Routers

“Thinking” Network System Modularity

- Applications deal with data
- End-host network **stacks** move data from applications to the fabric
- Network **fabric** delivers data between **network stacks**
- **Network (stack + fabric)** delivers data **between applications**
- What is the **interface** between applications and network stacks?
  - **Sockets**
- What is the **interface** between network stacks and network fabric?
  - **Packet headers**
- The right way to think about sockets and packets

Three Architectural Principles

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 end-to-end functionality into tasks

- Bits on wire
- Packets on wire
- Deliver packets between hosts in a “local” network (eg, within Cornell)
- Routing & forwarding packets across networks (eg, from Cornell to UIUC)
- Deliver data reliably between processes (applications)
- Do something with the data

Breakdown end-to-end functionality into tasks

- **Bits on wire**
- Packets on wire
- **Deliver packets between hosts in a local network**
- **Routing and forwarding (packets) across networks**
- **Deliver data reliably between processes**
- Do something with the data

Resulting Modules (Layers)

- **Bits on wire (Physical)**
- Packets on wire
- **Deliver packets between hosts in a local network (Datalink)**
- **Routing and forwarding (packets) across networks (Network)**
- **Deliver data reliably between processes (Transport)**
- Do something with the data (Application)

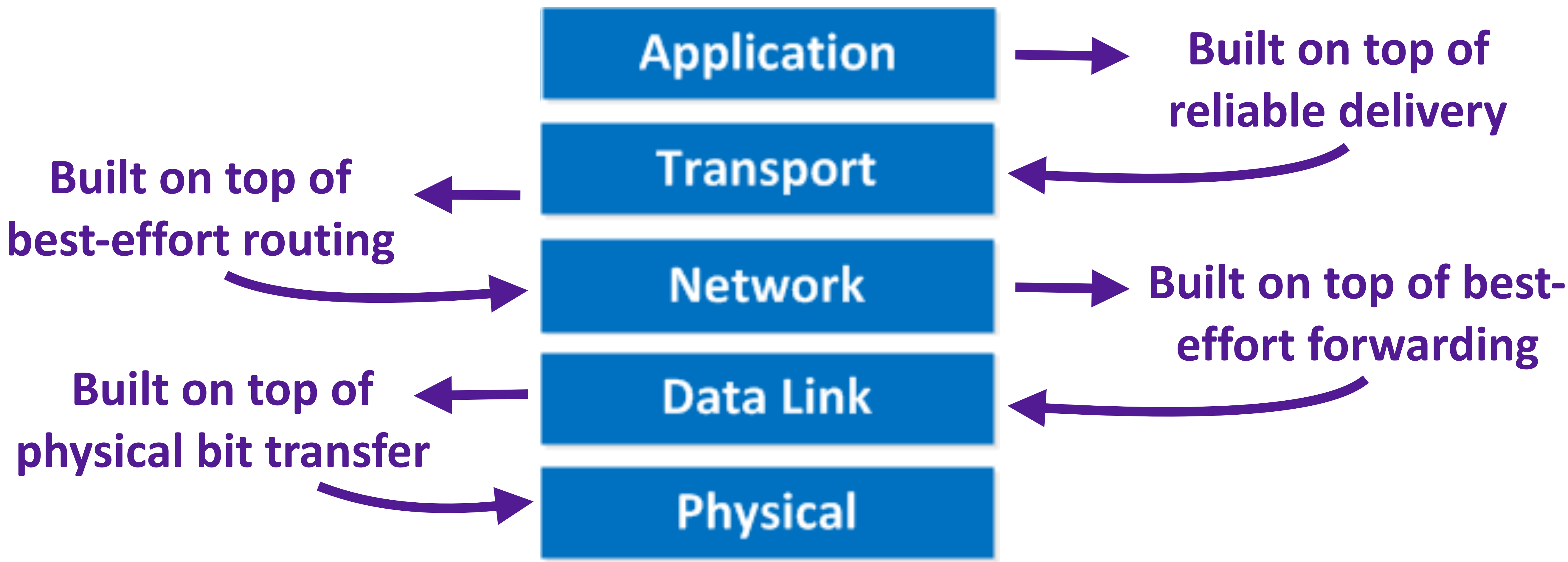
Resulting Modules (Layers)

- **Bits on wire (Physical, Layer1)**
- Packets on wire
- **Deliver packets to hosts across local network (Datalink, Layer2)**
- **Routing and forwarding (packets) across networks (Network, Layer3)**
- **Deliver data reliably between processes (Transport, Layer4)**
- 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):** Routing and forwarding across networks
- **Datalink (L2):** Forwarding within a local network
- **Physical (L1):** Bits on wire

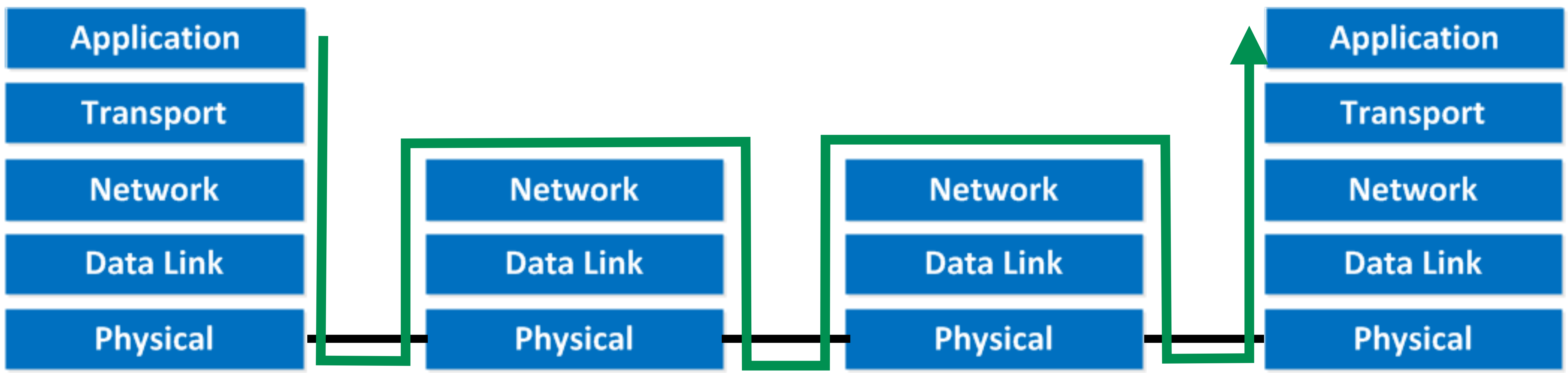
Layering



- **A kind of modularity**
  - Functionality separated into layers
  - Layer n **interfaces with only layer n-1 and layer n+1**
    - Hides complexity of surrounding layers

An end-to-end view of the layers

- Application: Providing network support for apps
- **Transport (L4):** (Reliable) end-to-end delivery
- **Network (L3):** Routing and forwarding across networks
- **Datalink (L2):** Forwarding within a local network
- **Physical (L1):** Bits on wire



Why does the packet go all the way to network layer at each hop?

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 single 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? And why?

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

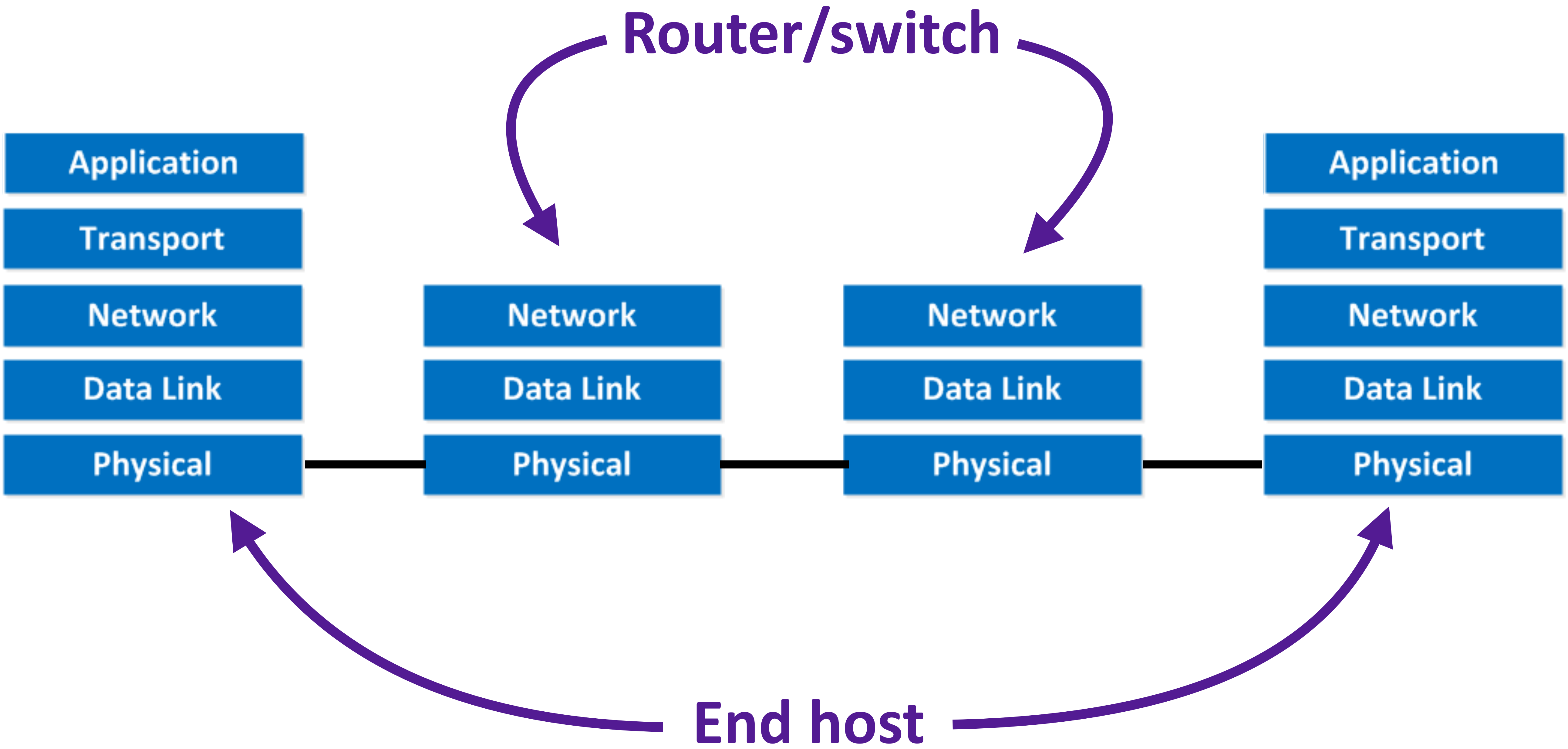
- Packets must be forwarded to next router/switch
  - Datalink layer necessary

- Routers participate in global delivery
  - Network layer necessary

- **Routers do not support reliable delivery**
  - Transport layer (and above) ***<u>not</u>*** supported
  - The question we want to answer: **Why?**

Visualizing what gets implemented where

- 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?

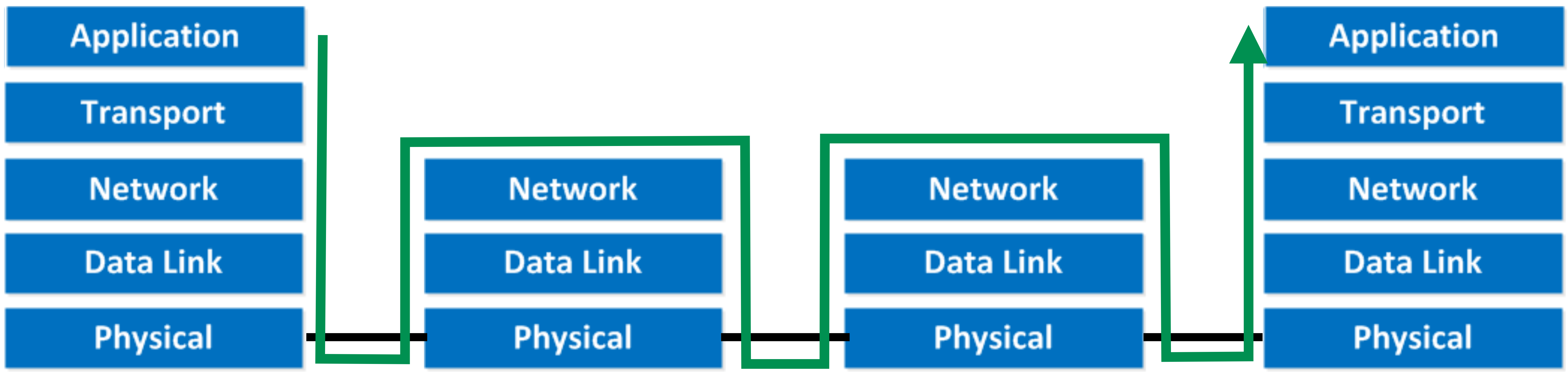
End-to-end Principle

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 each link layer transmission is reliable**
  - Does that ensure end-to-end (application-to-application) reliability?
- **Suppose network layer is reliable**
  - Does that ensure end-to-end (application-to-application) reliability?

End-to-end Principle: lets read again

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.

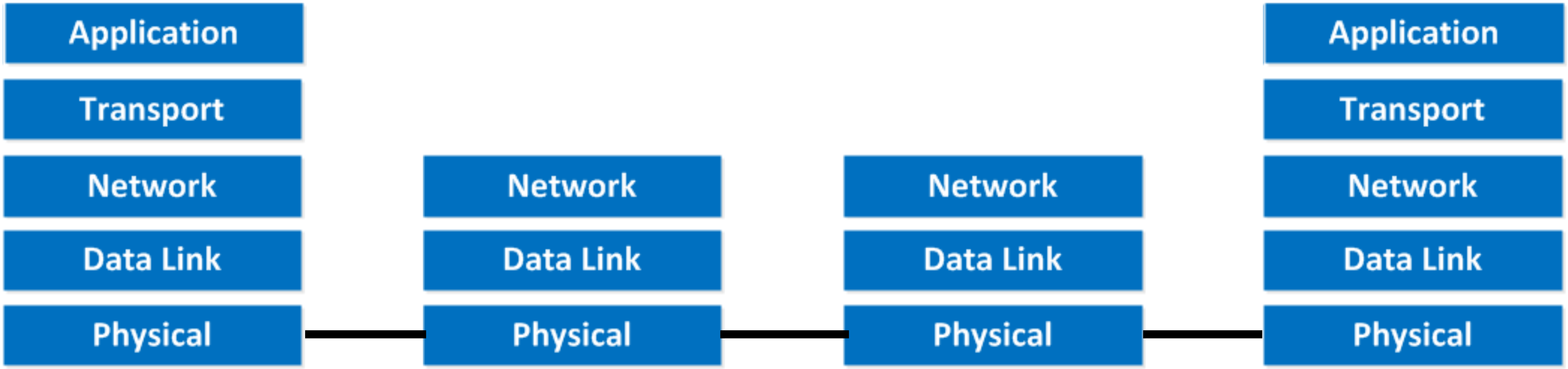
Sometimes providing an incomplete version of that function as a feature of the communication system itself may be useful as a performance enhancement.

End-to-end Principle (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

End-to-end Principle (Interpretation)



What does the end mean?

End-to-end Principle (Three things to know)

- **Everyone knows what it is**
  - So, you must!
- **Everyone believes it**
  - So, you must!
- **Nobody knows what it means**
  - We are all doomed anyways.

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 the end-to-end principle relied on “fate-sharing”**
  - Invariants only break when endpoints themselves break
  - Minimize the dependence on other network elements

- This should dictate placement of state

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

Questions?

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**