Taj Mahal, Agra, India

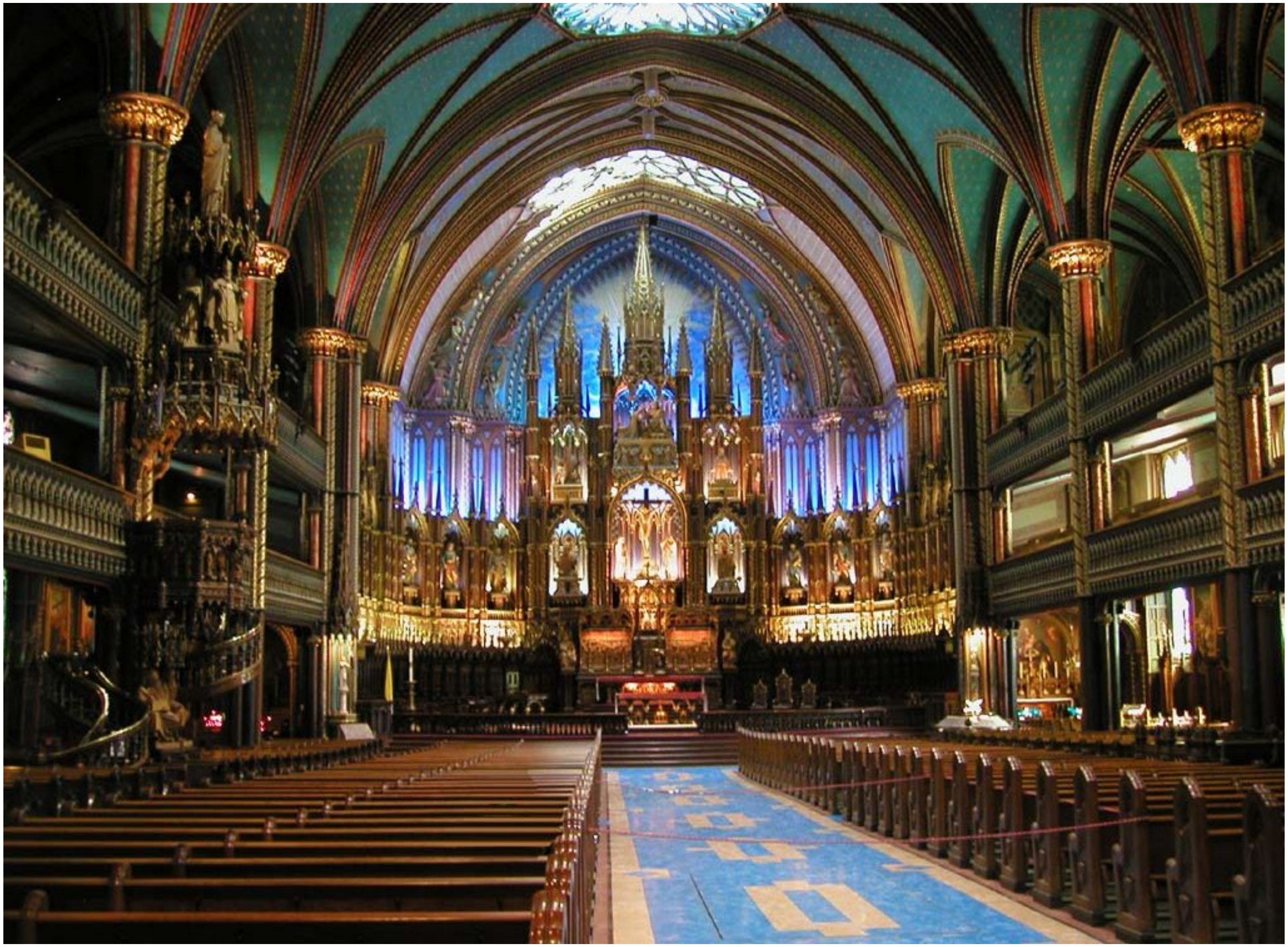
Temple of Heaven, Beijing, China

Opera House, Sydney, Australia

Parthenon, Athens, Greece

Burj al Arab Hotel, Dubai, UAE

Fallingwater, Mill Run, USA



Notre Dame Cathedral, Paris, France

Stata Hall, MIT, Cambridge, USA

COMP/ELEC 429/556
Introduction to Computer Networks

Internet architecture

Some slides used with permissions from Edward W. Knightly, T. S. Eugene Ng, Ion Stoica, Hui Zhang

Architecture: Organizing Network Functionality

- Goals: Functional, flexible, elegant/beautiful

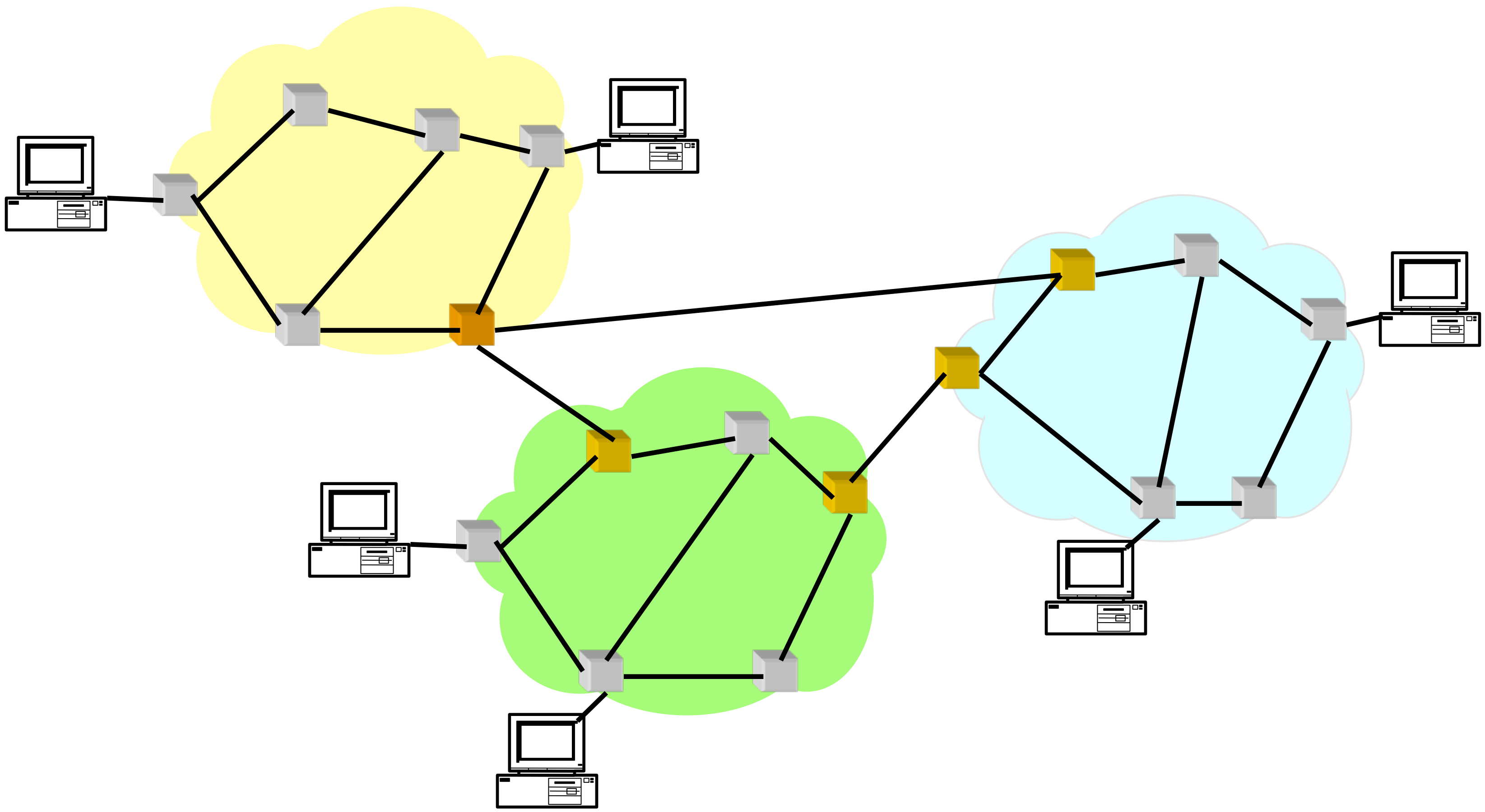
- Many kinds of networking functionality
  - e.g., encoding, framing, routing, addressing, reliability, etc.
- Many different network styles and technologies
  - circuit-switched vs packet-switched, etc.
  - wireless vs wired, electrical vs optical, etc.
- Many different applications
  - dropbox, web, voice, video, etc.
- A network has many nodes (routers, switches, hosts)

- Network architecture
  - On which node(s) should each functionality be placed?
  - How should functionalities on a node be organized?

<u>Central question:</u>
<u>On which node(s) should functionalities be placed?</u>

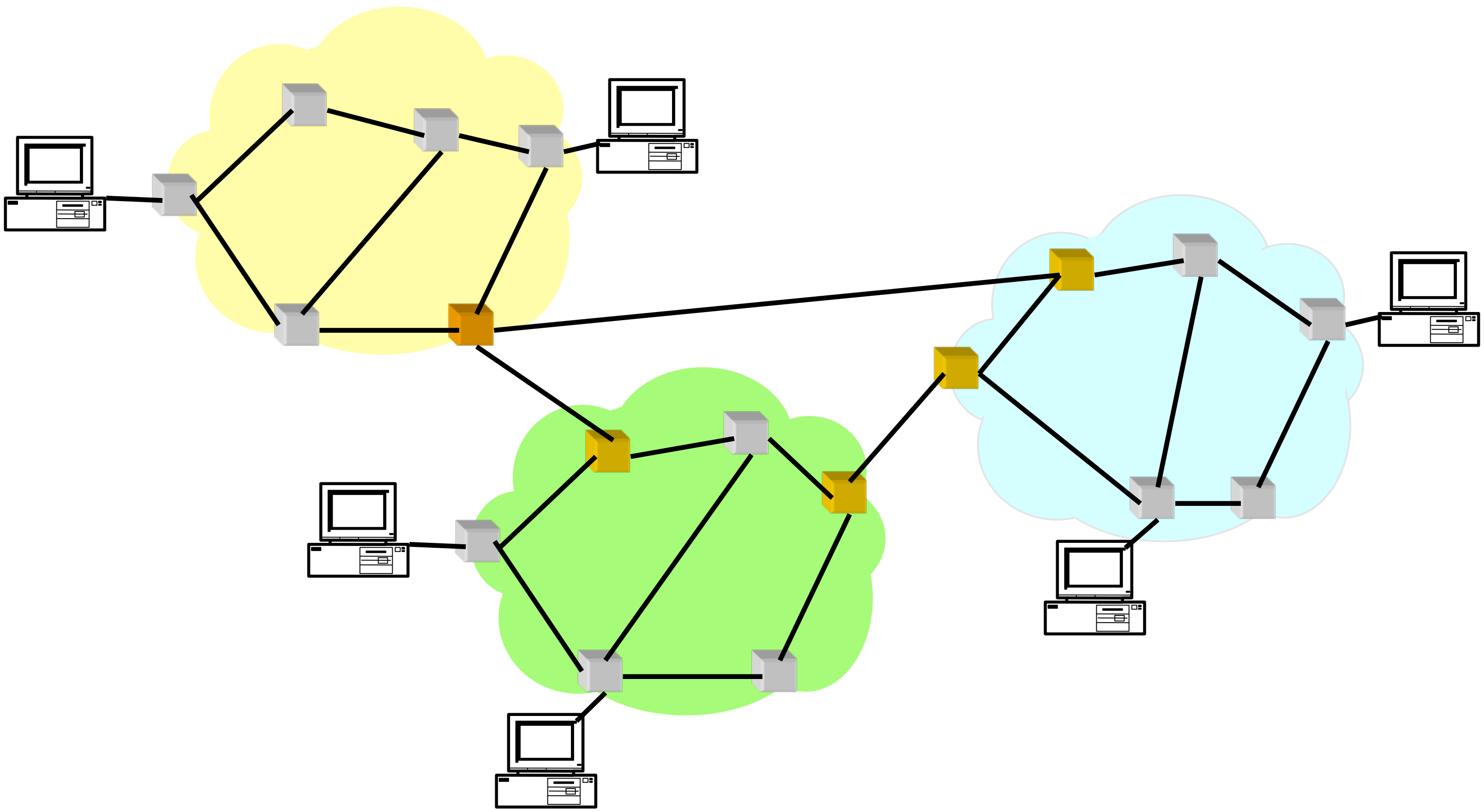
Example: Addressing

Which node(s) should be able to interpret network addresses in packets?



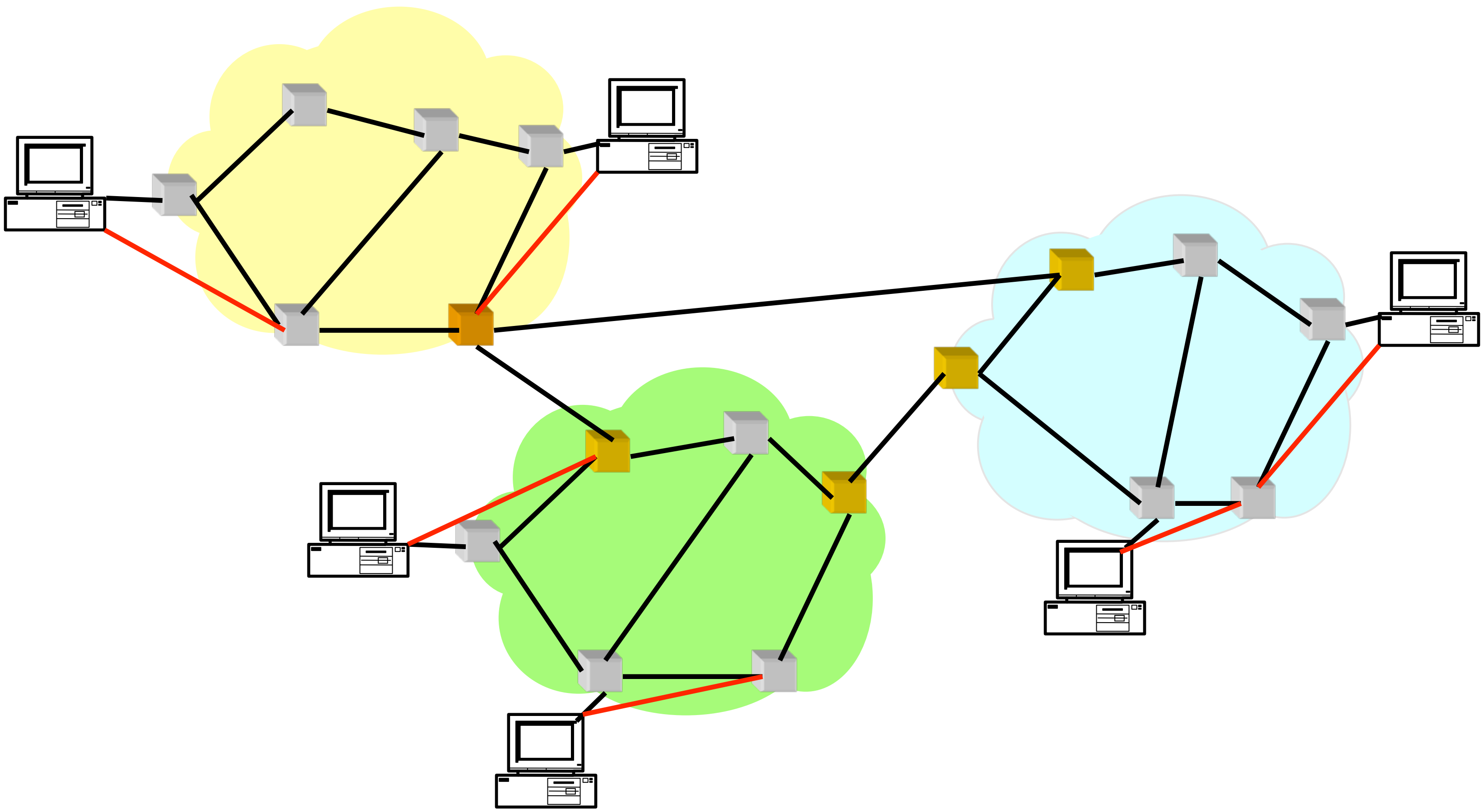
Example: Routing

Which node(s) should make routing decisions?

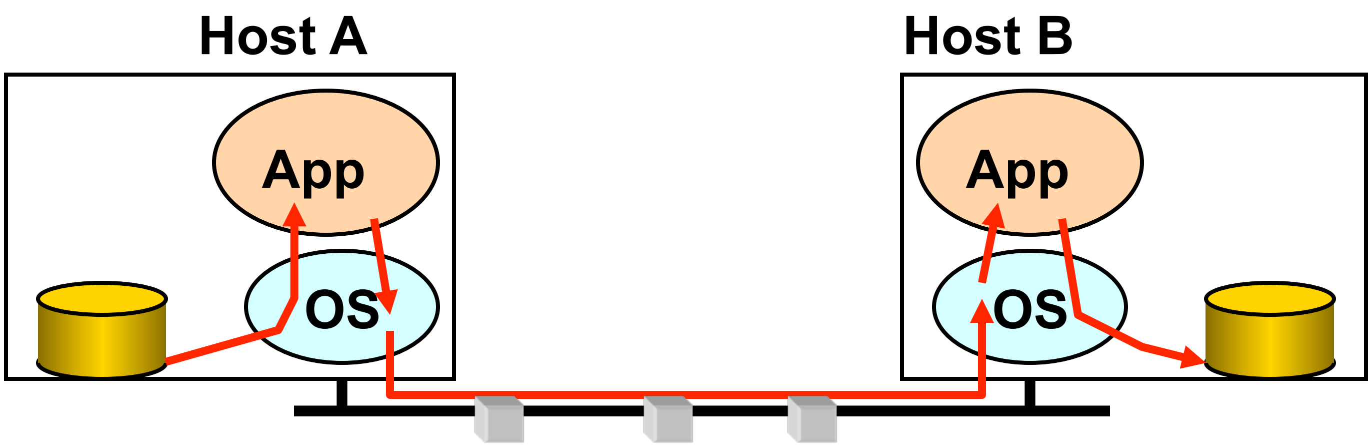


Example: Routing

Which node(s) should make routing decisions?



Example: Reliable File Transfer



- Idea 1: put reliability function in the network; i.e. make network reliable
- Idea 2: put reliability function in the hosts; i.e. implement correctness check at application and retry if failed

Example (cont'd)

- Idea 1 not complete
  - Bugs can exist in OS code!
  - Data on disk can be corrupted during write
  - The receiver has to do the check anyway!
- Idea 2 is complete
  - Full functionality can be entirely implemented at application with no need for guaranteed reliability from other components

Placing Functionality

- The most influential paper about placing functionality is "End-to-End Arguments in System Design" by Saltzer, Reed, and Clark

Basic Observations

- There are many levels in a distributed system: application, OS, network, etc.
- Some applications have end-to-end performance requirements
  - reliability, security, etc.
- Implementing these in levels below the applications is very hard
  - every step along the way must be fail-proof
  - if not fail-proof, application's implementation complexity is not reduced
  - increases lower levels' complexity
  - imposes delay and overhead on all applications, even if they don't have such requirements
- The applications:
  - can satisfy the requirement
  - can't depend on the lower levels

Conservative Interpretation of the End-to-End Argument

- Don’t implement a function in a low level unless it can be completely implemented in that level
  - Unless you can relieve the burden from applications, then don’t bother

Radical Interpretation

- Don't implement anything in a low level that can be implemented correctly by the applications
- Make the low level absolutely minimal
  - ignore performance issues

Moderate Interpretation

- Think twice before implementing functionality in the network
- If hosts can implement functionality correctly, implement it at a lower level only as a performance enhancement
- But do so only if it does not impose burden on applications that do not require that functionality

How to Organize Functionalities on Each Node?

Software Modularity

Break system into modules:

- Well-defined interfaces gives flexibility
  - can change implementation of modules
  - can extend functionality of system by adding new modules
- Interfaces hide information
  - allows for flexibility
  - but can hurt performance

Network Modularity

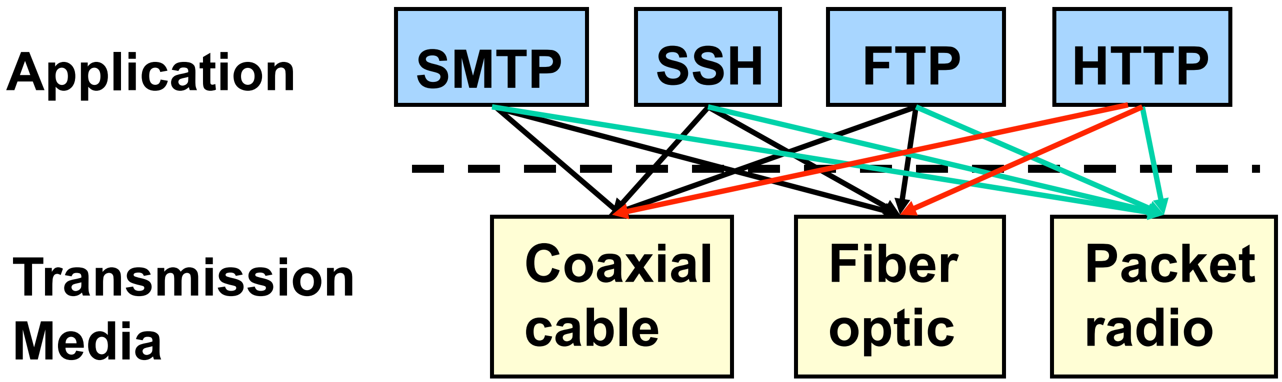
Like software modularity, but with a twist:

- Implementation distributed (e.g. across routers and hosts)
- Must decide both:
  - how to break system into modules
  - where modules are implemented

Layering

- Layering is a particular form of modularization
- The system is broken into a vertical hierarchy of logically distinct entities (layers)
- The service provided by one layer is based solely on the service provided by layer below
- Rigid structure: easy reuse, performance may suffer

A Naïve Architecture

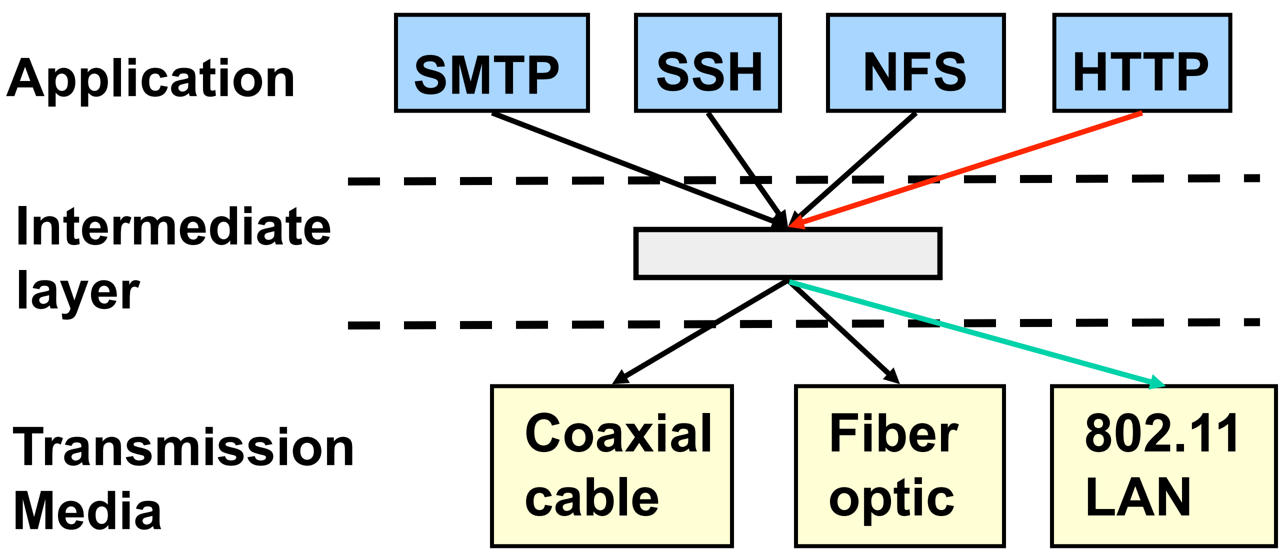


- new application has to interface to all existing media
  - adding new application requires O(m) work, m = number of media
- new media requires all existing applications be modified
  - adding new media requires O(a) work, a = number of applications
- total work in system O(ma) → eventually too much work to add apps/media

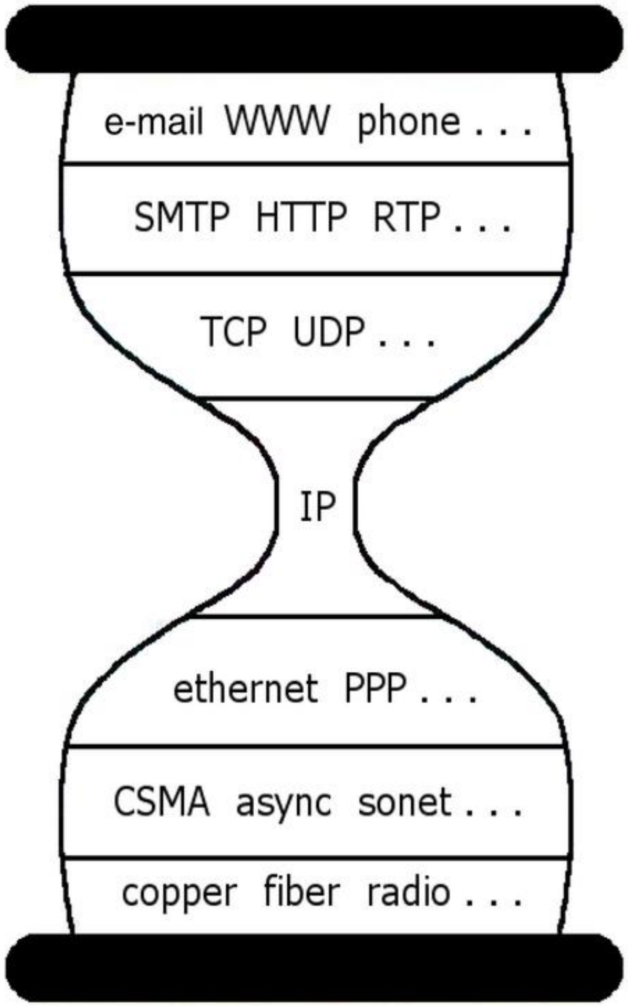
Solution: Indirection

- Solution: introduce an intermediate layer that provides a single abstraction for various network technologies
  - O(1) work to add app/media

vs



Take home point: The Internet Hourglass



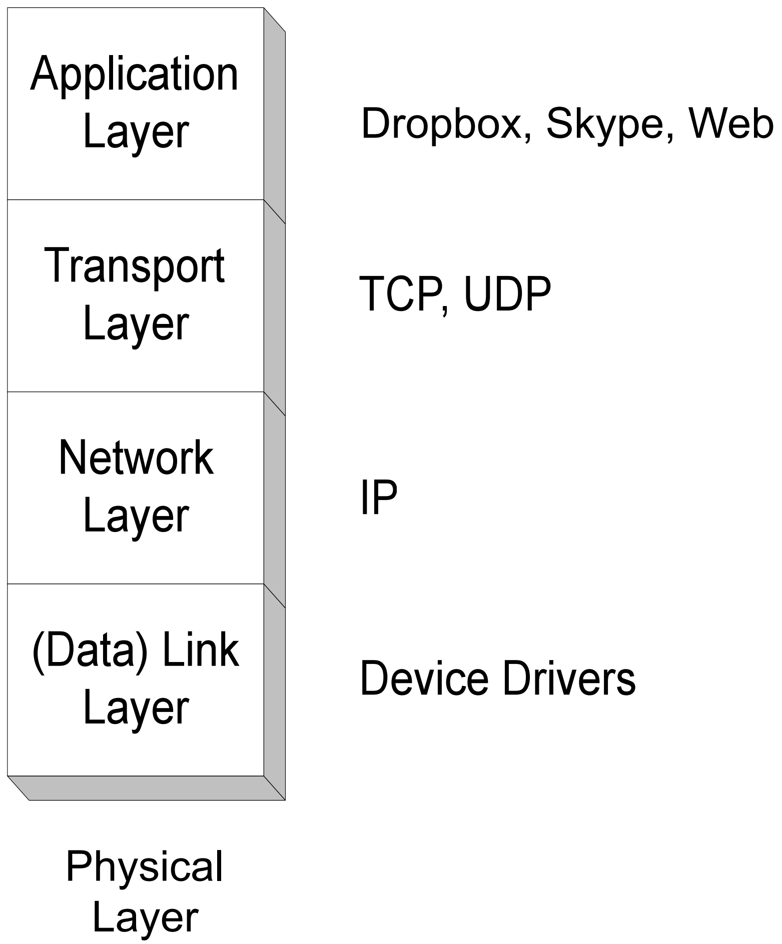
Implications of Hourglass

A single Internet layer module:

- Allows all networks to interoperate
  – all networks technologies that support IP can exchange packets
- Allows all applications to function on all networks
  – all applications that can run on IP can use any network
- Simultaneous developments above and below IP

Internet Protocol Architecture

- The TCP/IP protocol suite is the basis for the networks that we call the Internet.
- The TCP/IP suite has four layers: Application, Transport, Network, and (Data) Link Layer.



Terminology

- Service – says what a layer does
  - Ethernet: unreliable subnet unicast/multicast/broadcast datagram service
  - IP: unreliable end-to-end unicast datagram service
  - TCP: reliable end-to-end bi-directional byte stream service
- Service Interface – says how to access the service
  - E.g. UNIX socket interface
- Protocol – says how the service is implemented
  - a set of rules and formats that govern the communication between two peers

Physical Layer (1)

- **Service**: move 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

Datalink Layer (2)

- **Service**:
  - framing (attach frame separators)
  - send data frames between peers
  - others:
    - arbitrate the access to common physical media
    - per-hop reliable transmission
    - per-hop flow control

- **Interface**: send a data unit (packet) to a machine connected to the same physical media
- **Protocol**: layer addresses, implement Medium Access Control (MAC) (e.g., IEEE802.3, IEEE802.11, CSMA/CD)…

Network Layer (3)

- **Service**:
  - deliver a packet to specified network destination
  - perform segmentation/reassemble
  - others will be discussed
- **Interface**: send a packet to a specified destination
- **Protocol**: define global unique addresses; construct routing tables (e.g IPv4, IPv6)

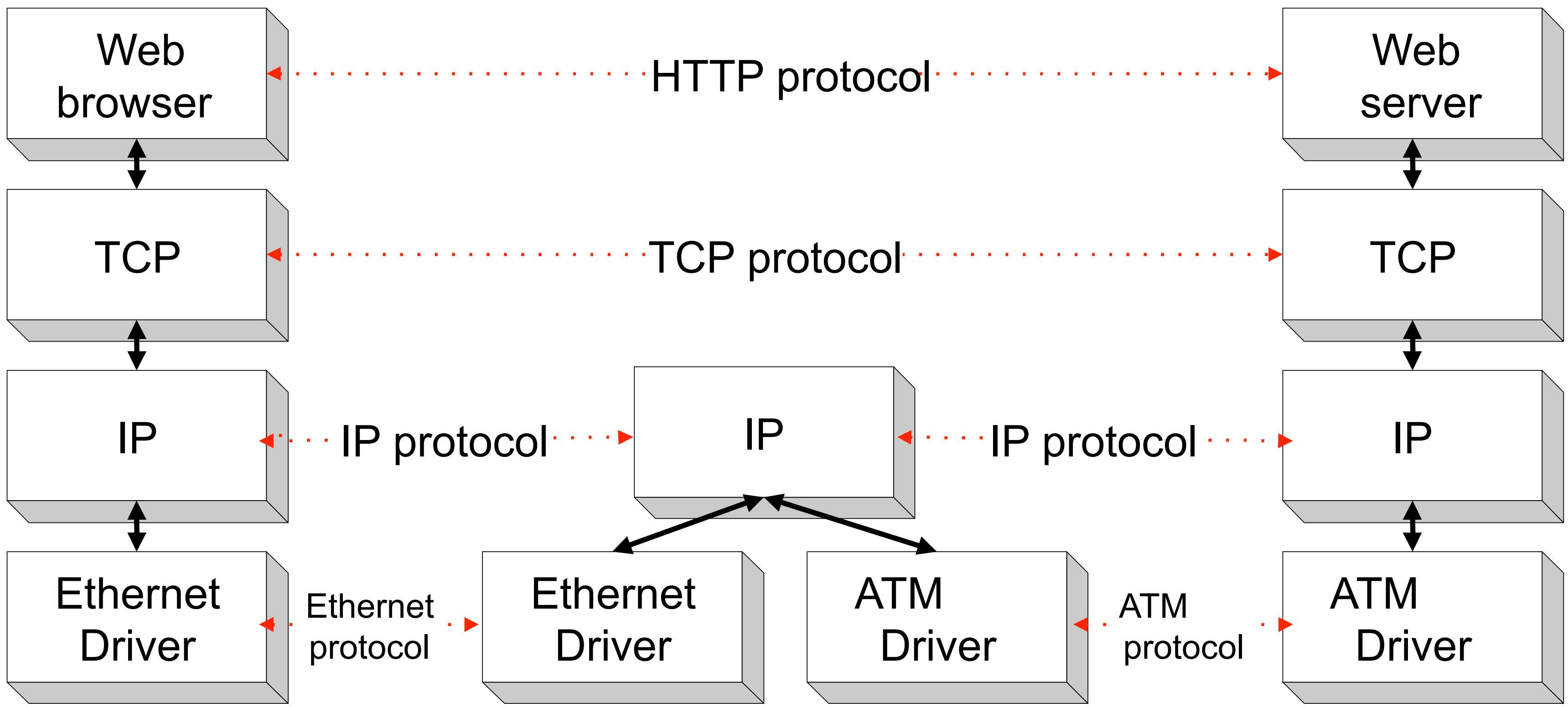
Transport Layer (4)

- **Service**:
  - Multiplexing/demultiplexing
  - optional: error-free and flow-controlled delivery
- **Interface**: send message to specific destination
- **Protocol**: implements reliability and flow control
- Examples: TCP and UDP

Application Layer (7)

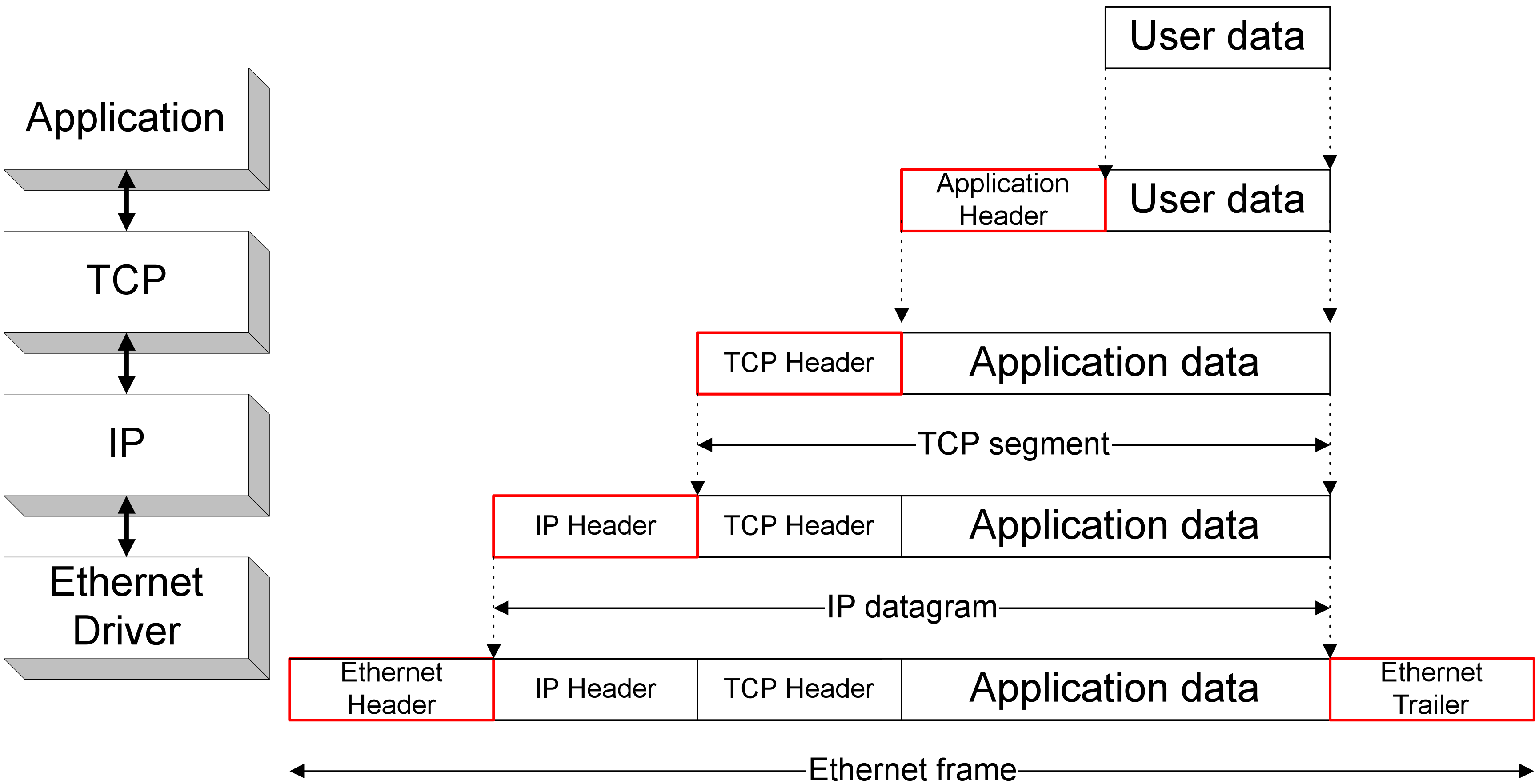
- **Service**: any service provided to the end user
- **Interface**: depends on the application
- **Protocol**: depends on the application
- Examples: Dropbox, Skype, Web

Internet Protocol Architecture



Encapsulation

- As data is moving down the protocol stack, each protocol is adding layer-specific control information.



Reality

- Layering and E2E Principle regularly violated:
  - Firewall
  - Network address translation
  - Transparent web cache
- Battle between architectural purity and commercial pressures