

# Networks Introduction

CSE 333 Autumn 2021

**Instructor:** Chris Thachuk

**Teaching Assistants:**

Arpad (John) Depaszthory

Ian Hsiao

Logan Gnanapragasam

Mengqi (Frank) Chen

Angela Xu

Khang Vinh Phan

Maruchi Kim

Cosmo Wang

# Administrivia

- ❖ Rest of the quarter:
  - Topics: Networking, Concurrency, Processes
  - Homework 3: file search shell
  - Homework 4: file search shell over the network
  
- ❖ Exercise 12 is due next Monday (Nov. 15)
- ❖ Exercise 13 released on Wednesday
- ❖ Only 3 exercises after that:
  - TCP sender, TCP listener, concurrency
  
- ❖ Homework 3 is due Nov. 24

# Lecture Outline

- ❖ **C++ Conversions**
- ❖ Introduction to Networks

# Implicit Conversion

- ❖ The compiler tries to infer some kinds of conversions
  - When types are not equal and you don't specify an explicit cast, the compiler looks for an acceptable implicit conversion

```
void bar(std::string x);

void foo() {
    int x = 5.7;    // conversion, float -> int
    char c = x;    // conversion, int -> char
    bar("hi");     // conversion, (const char*) -> string
}
```

# Sneaky Implicit Conversions

- ❖ (`const char*`) to `string` conversion?
  - If a class has a constructor with a single parameter, the compiler will exploit it to perform implicit conversions
  - At most, one user-defined implicit conversion will happen
    - Can do `int` → `Foo`, but not `int` → `Foo` → `Baz`

```
class Foo {
public:
    Foo(int xi) : x(xi) { }
    int x;
};

int Bar(Foo f) {
    return f.x;
}

int main(int argc, char** argv) {
    return Bar(5); // equivalent to return Bar(Foo(5));
}
```

Constructor implicitly invoked

# Avoiding Sneaky Implicit

- ❖ Declare one-argument constructors as `explicit` if you want to disable them from being used as an implicit conversion path
  - Usually a good idea

```
class Foo {
public:
    explicit Foo(int xi) : x(xi) { }
    int x;
};

int Bar(Foo f) {
    return f.x;
}

int main(int argc, char** argv) {
    return Bar(5); // compiler error    No longer allowed
}
```

# Lecture Outline

- ❖ C++ Conversions
- ❖ Introduction to Networks
  - Layers upon layers upon layers...

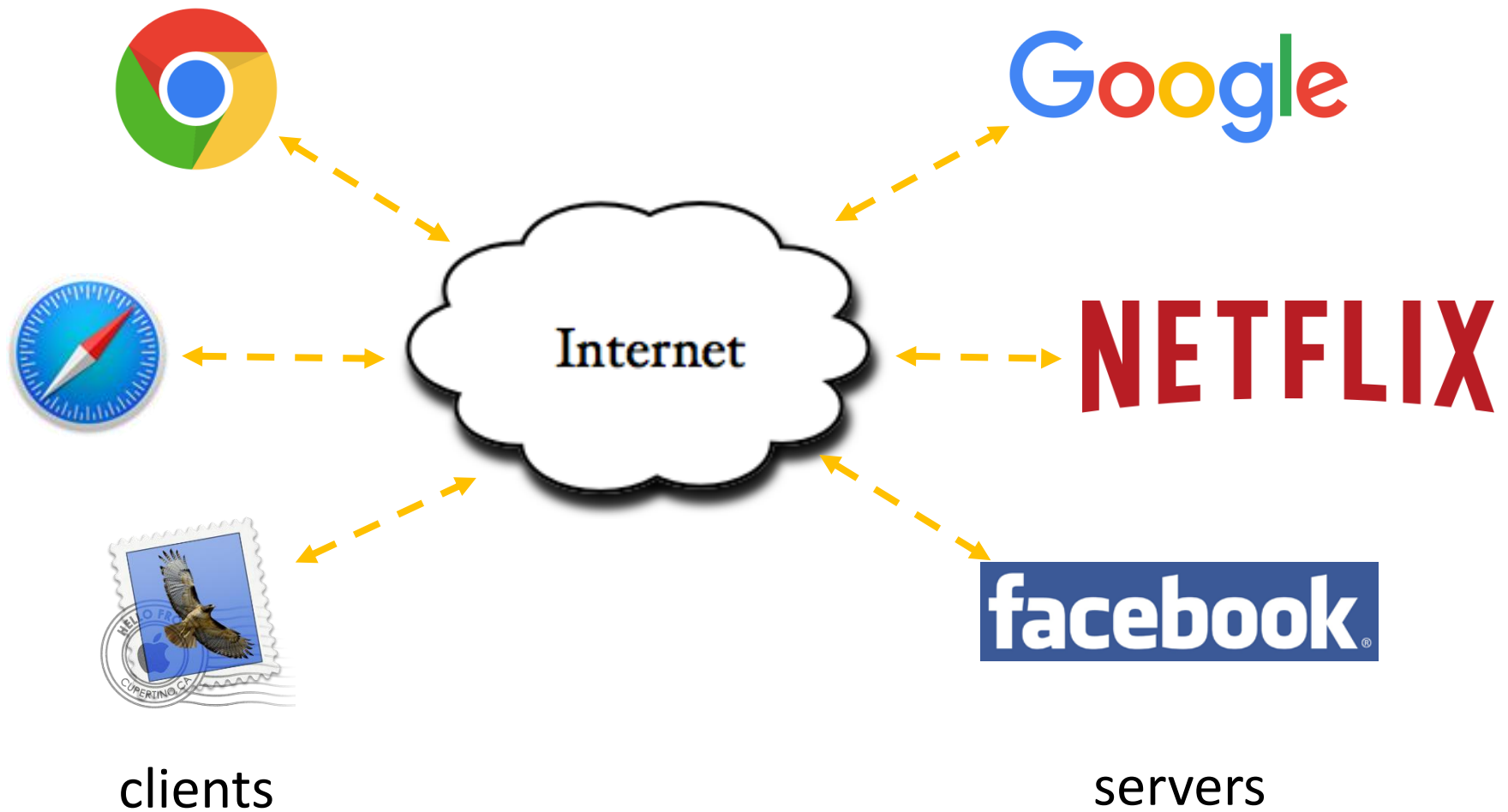


more awesome pictures at [THEMETAPICTURE.COM](http://THEMETAPICTURE.COM)

# Today's Goals

- ❖ Networking is a very common programming feature
  - You will likely have to create a program that will read/write over the network at some point in your career
- ❖ We want to give you a basic, high-level understanding of how networks work before you use them
  - Lecture will be more “story-like;” we will purposefully skip over most of the details, but hopefully you will learn something new about the Internet today!
  - Take CSE 461 if you want to know more about the implementations of networks (the course is pretty cool 😊)
- ❖ Let's also examine “the network” as a *system*
  - Inputs? Outputs? Robustness? Efficiency? Customers?

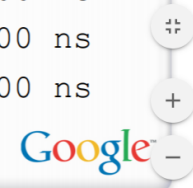
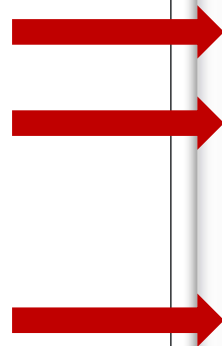
# Networks From 10,000 ft



# “Network” Latency is Highly Variable

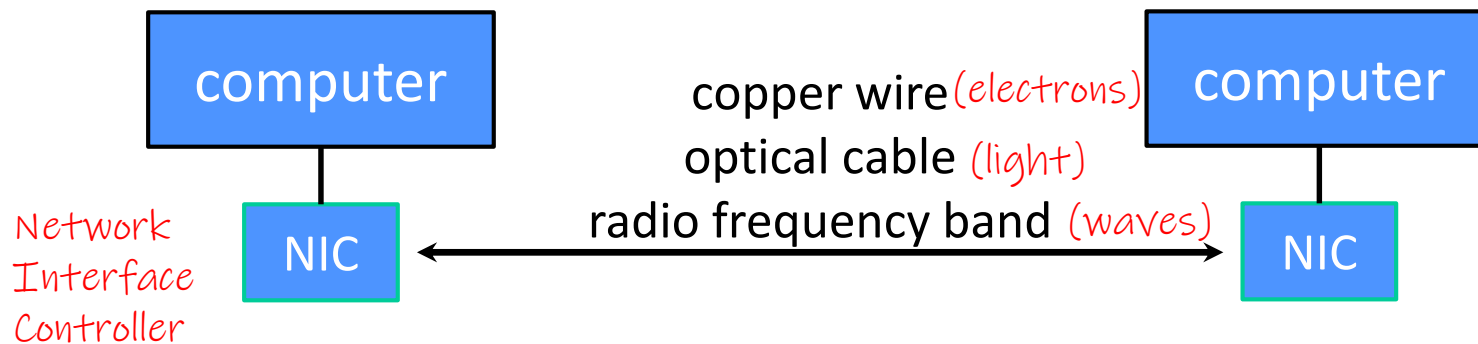
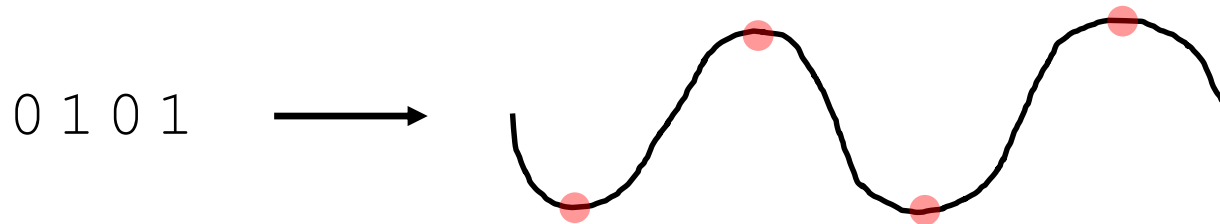
- ❖ Jeff Dean’s “Numbers Everyone Should Know” (LADIS ‘09)

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns



# The Physical Layer

- ❖ Individual bits are modulated onto a wire or transmitted over radio
  - Physical layer specifies how bits are encoded at a signal level
  - Many choices, *e.g.*, encode “1” as +1v, “0” as -0v; or “0”=+1v, “1”=-1v, ...



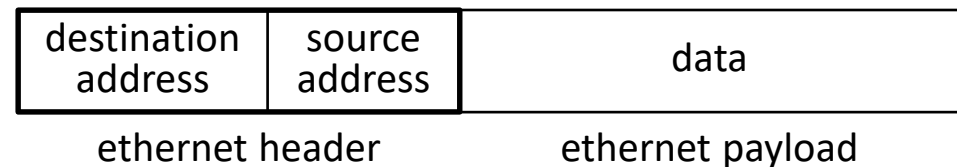
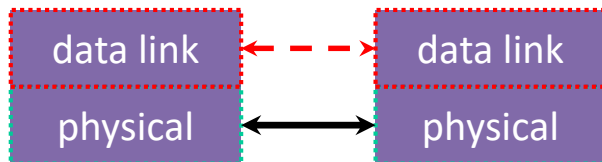
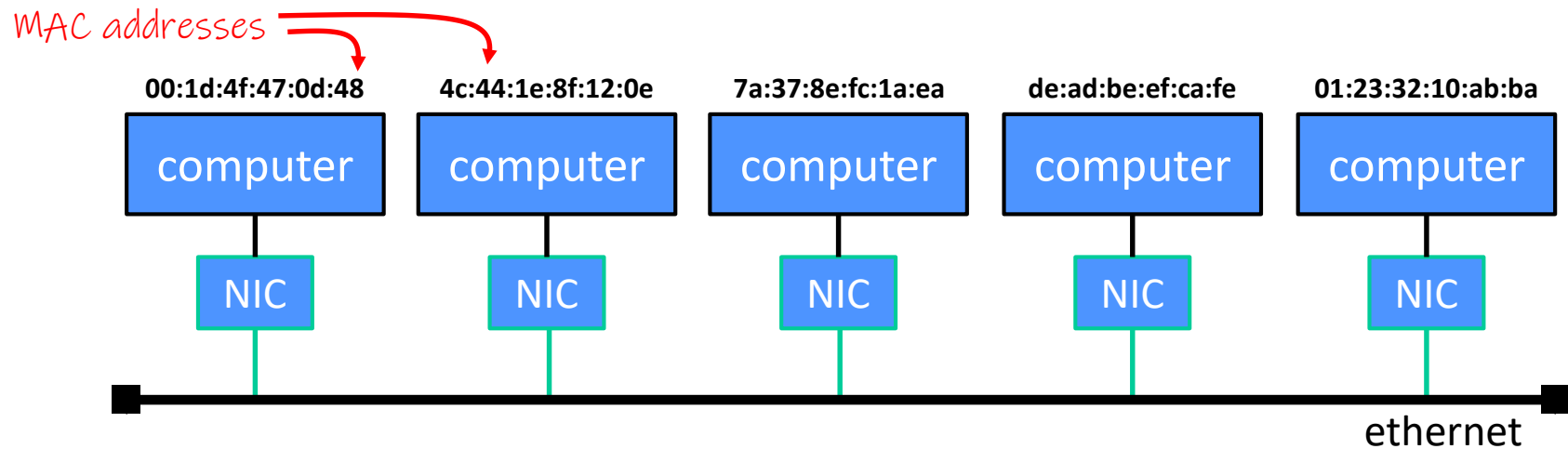
# Materials Matter – Latency

- ❖ Fiber optic cables are lower-latency and higher-bandwidth than traditional copper wiring
  - Much of the internet's "long haul" data is transmitted on these
  - (signal attenuation is much better too)
- ❖ Is it faster to send 1 person from UW to ...
  - Downtown Seattle?
  - Downtown Ballard?

*Not just distance, but also speed limit & number of lanes*

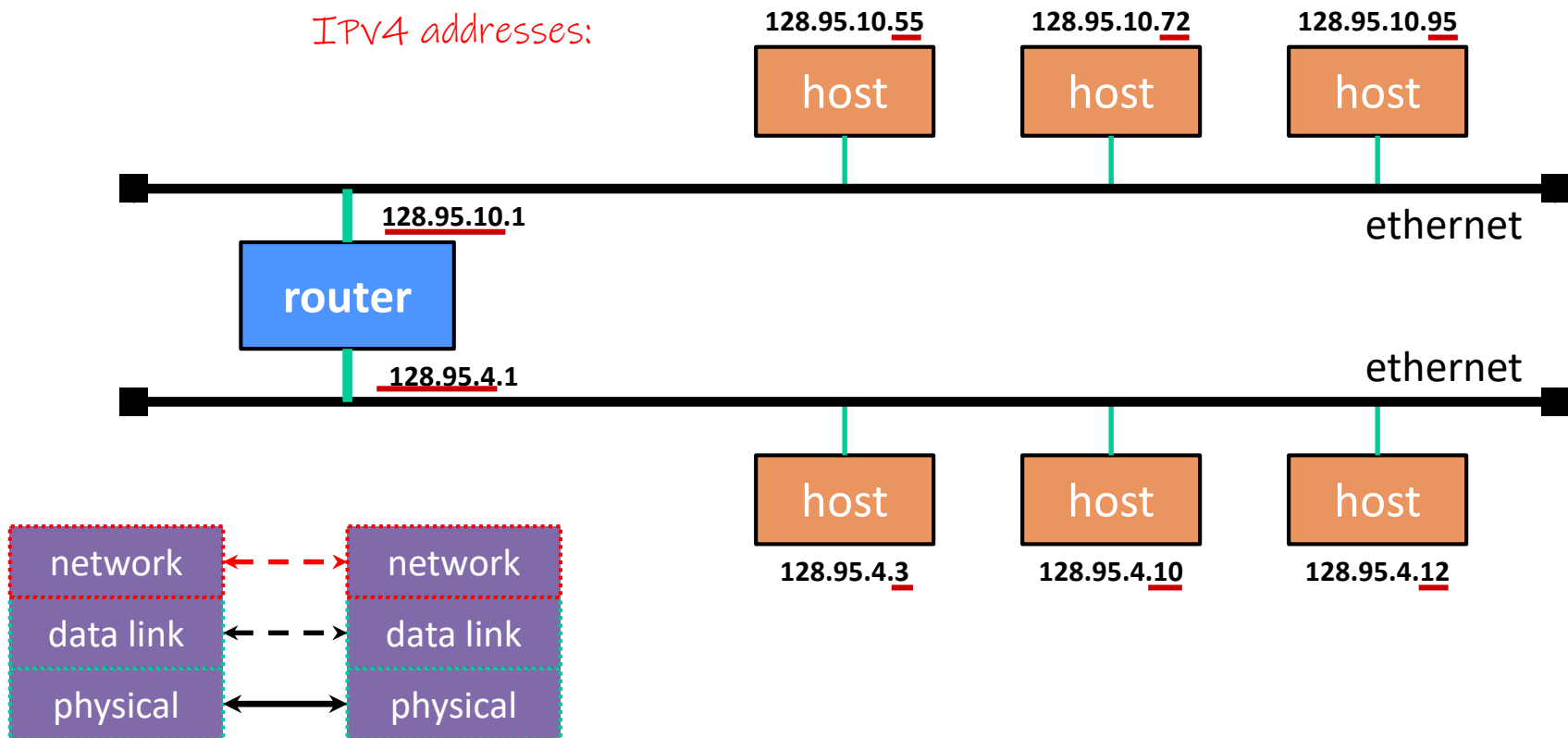
# The Data Link Layer

- ❖ Multiple computers on a LAN contend for the network medium
  - Media access control (MAC) specifies how computers cooperate
  - Link layer also specifies how bits are “packetized” and network interface controllers (NICs) are addressed



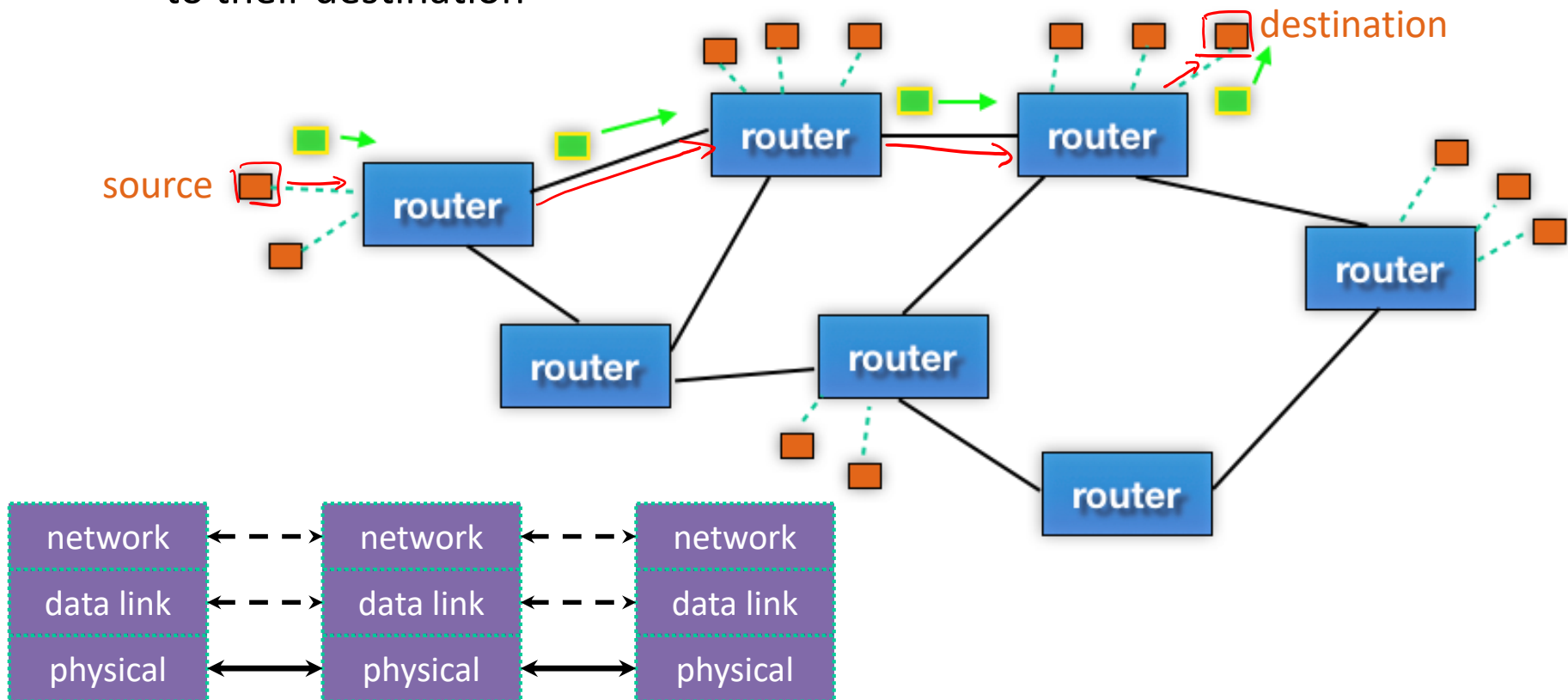
# The Network Layer (IP)

- ❖ Internet Protocol (IP) routes packets across multiple networks
  - Every computer has a unique IP address
  - Individual networks are connected by routers that span networks



# The Network Layer (IP)

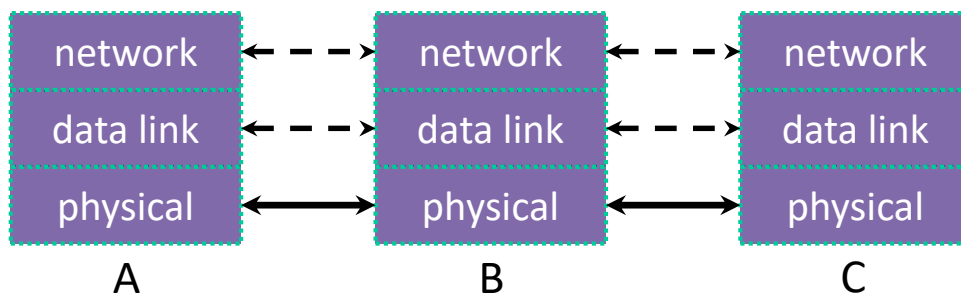
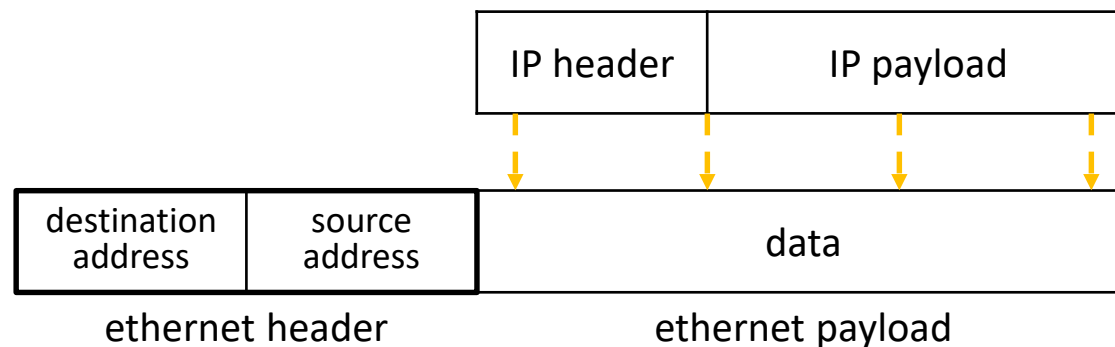
- ❖ There are protocols to:
  - Let a host map an IP to MAC address on the same network
  - Let a router learn about other routers to get IP packets one step closer to their destination



# The Network Layer (IP)

## ❖ Packet encapsulation:

- An IP packet is encapsulated as the payload of an Ethernet frame
- As IP packets traverse networks, routers pull out the IP packet from an Ethernet frame and plunk it into a new one on the next network



# Distance Matters – Latency

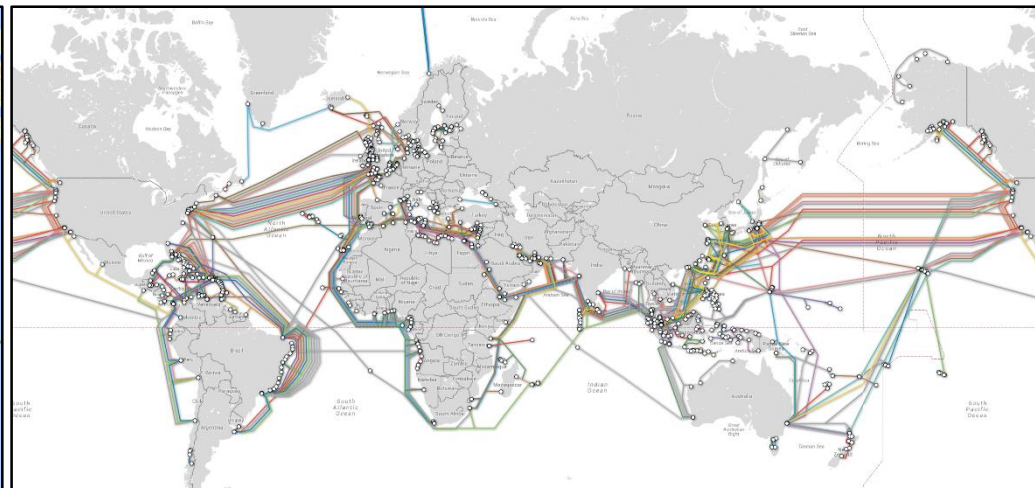
- ❖ Distances within a single datacenter are smaller than distances across continents
- ❖ Even within a datacenter, distances can sometimes matter



123Net Data Center, Wikimedia

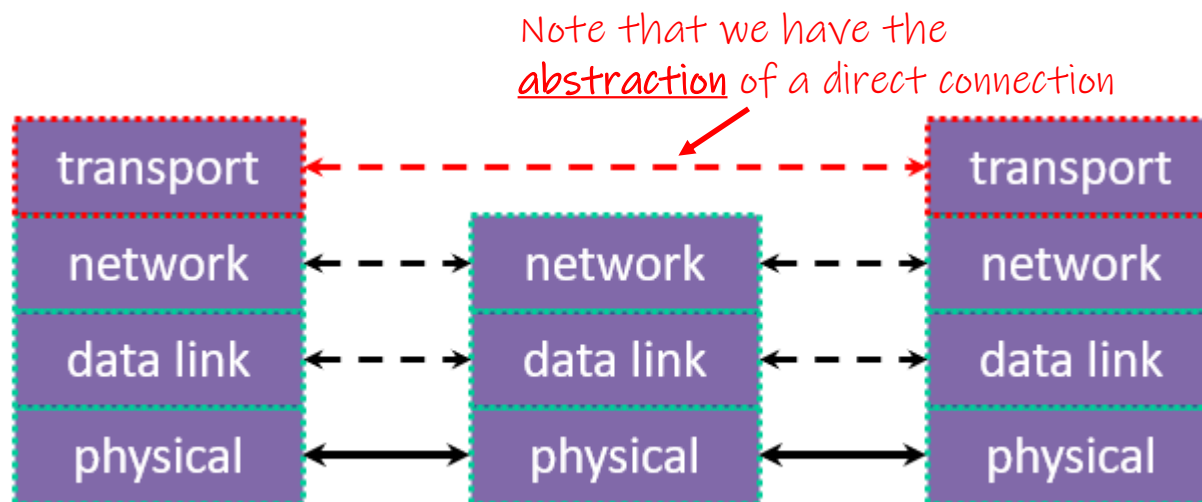
# Topology Matters – Latency and Reliability

- ❖ Some places are surprisingly well- or poorly-connected to “backbone” infrastructure like fiber optic cables
- ❖ Unintuitive topology can create interesting failures
  - *e.g.*, 2006 7.0-magnitude Hengchun Earthquake disrupted communications to Singapore, Philippines, Thailand, China, etc. for a month



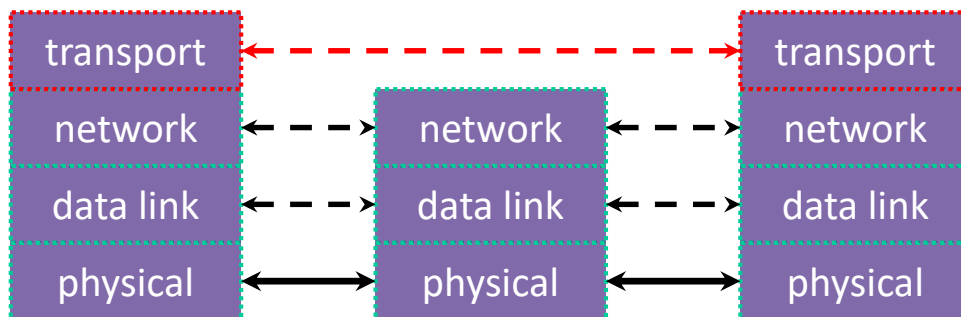
# The Transport Layer

- ❖ Provides an interface to treat the network as a *data stream*
- ❖ Provides different protocols to interface between source and destination:
  - e.g., Transmission Control Protocol (TCP), User Datagram Protocol (UDP)
  - These protocols still work with packets, but manages their order, reliability, multiple applications using the network...



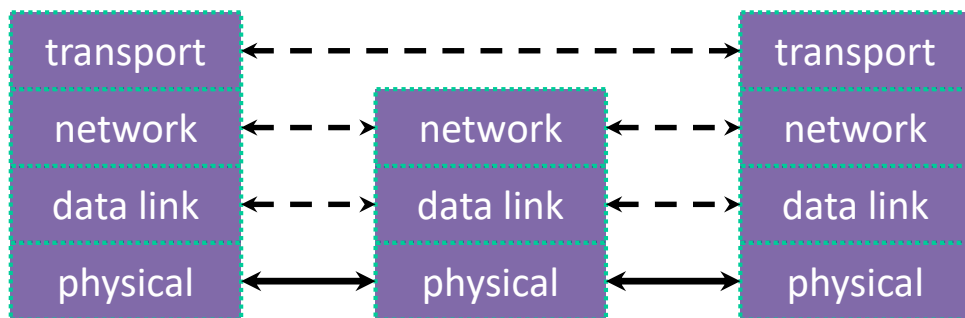
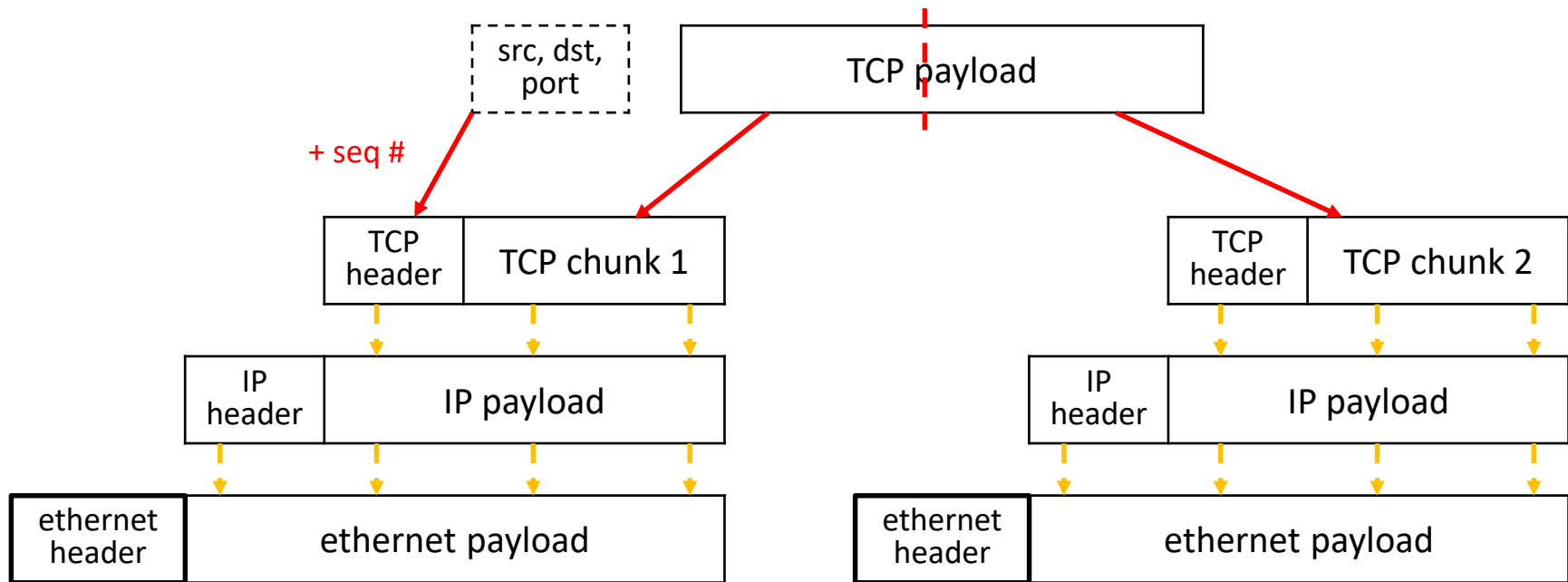
# The Transport Layer (TCP)

- ❖ Transmission Control Protocol (TCP):
  - Provides applications with reliable, ordered, congestion-controlled byte streams
    - Sends stream data as multiple IP packets (differentiated by sequence numbers) and retransmits them as necessary
    - When receiving, puts packets back in order and detects missing packets
  - A single host (IP address) can have up to  $2^{16} = 65,535$  “ports”
    - Kind of like an apartment number at a postal address (your applications are the residents who get mail sent to an apt. #)



# The Transport Layer (TCP)

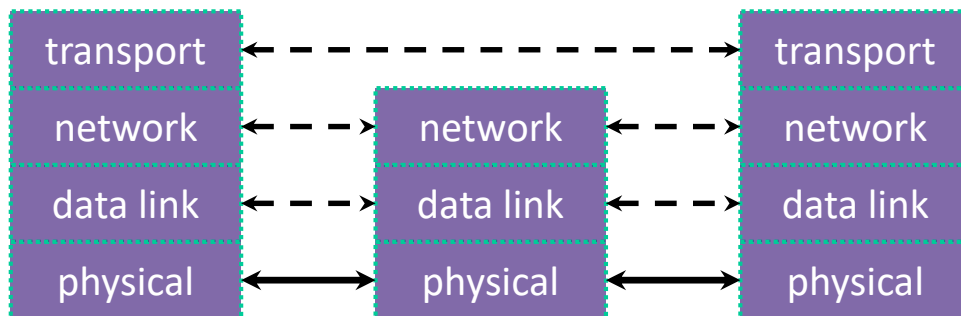
- ❖ Packet encapsulation – one more nested layer!



# The Transport Layer (TCP)

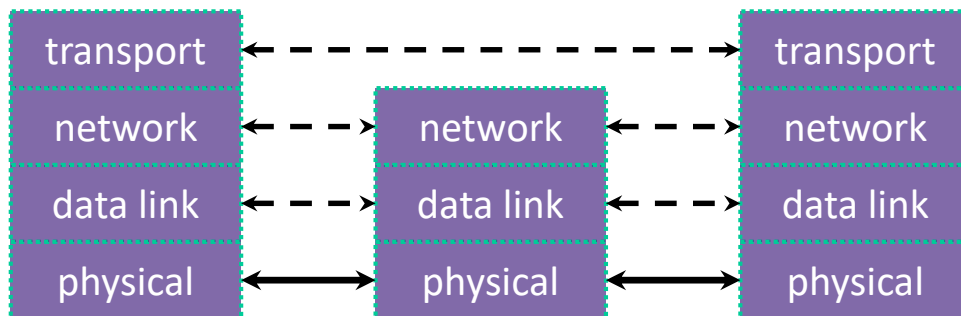
- ❖ Applications use OS services to establish TCP streams:
  - The “Berkeley sockets” API
    - A set of OS system calls *(part of POSIX for Linux)*
  - Clients **connect** () to a server IP address + application port number
  - Servers **listen** () for and **accept** () client connections
  - Clients and servers **read** () and **write** () data to each other

*same as  
for file I/O*



# The Transport Layer (UDP)

- ❖ User Datagram Protocol (UDP):
  - Provides applications with *unreliable* packet delivery (ok for things like video streaming)
  - UDP is a really thin, simple layer on top of IP
    - Datagrams still are fragmented into multiple IP packets

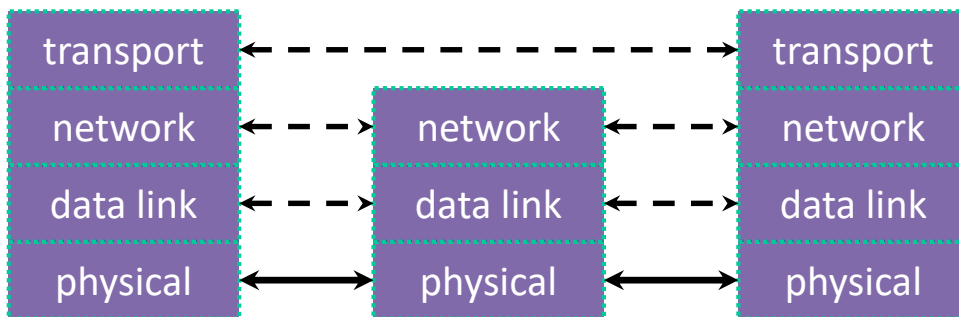


# The Transport Layer

**TCP:**

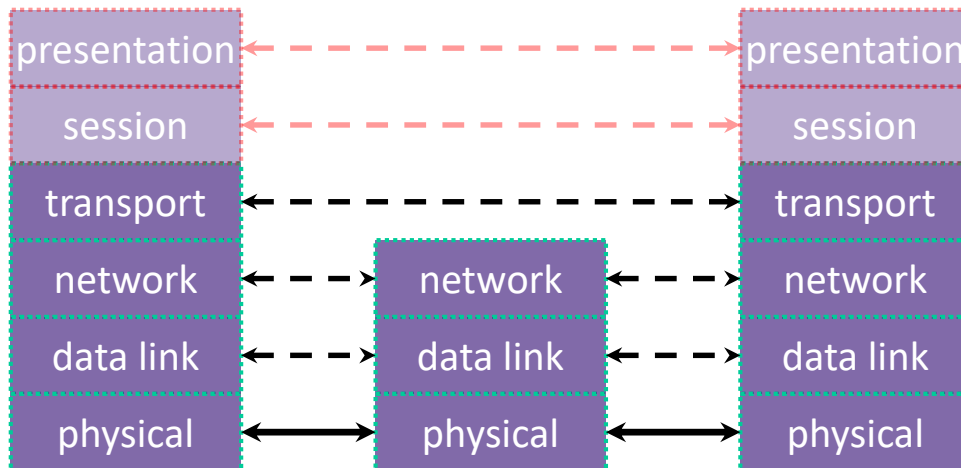


**UDP:**



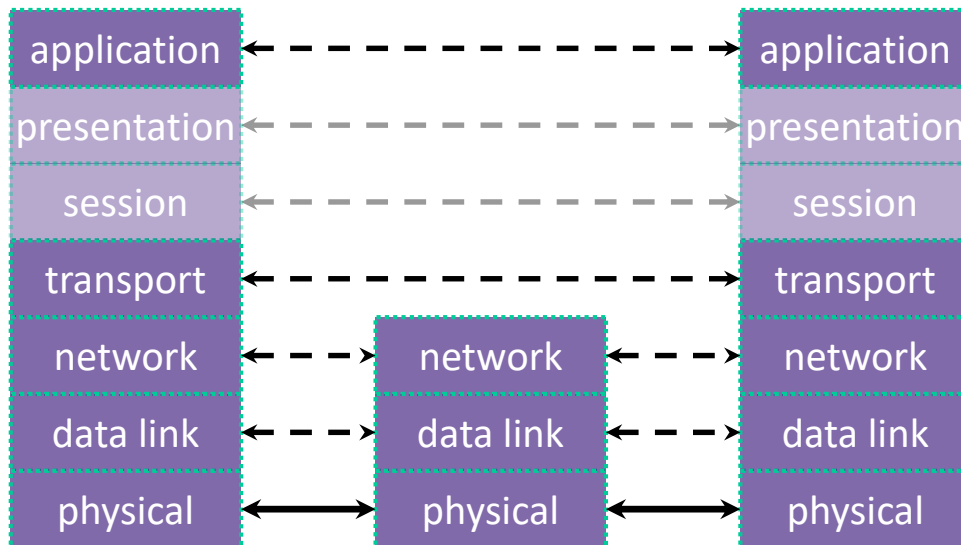
# The (Mostly Missing) Layers 5 & 6

- ❖ Layer 5: Session Layer
  - Supposedly handles establishing and terminating application sessions
  - Remote Procedure Call (RPC) kind of fits in here
- ❖ Layer 6: Presentation Layer
  - Supposedly maps application-specific data units into a more network-neutral representation
  - Encryption (SSL) kind of fits in here



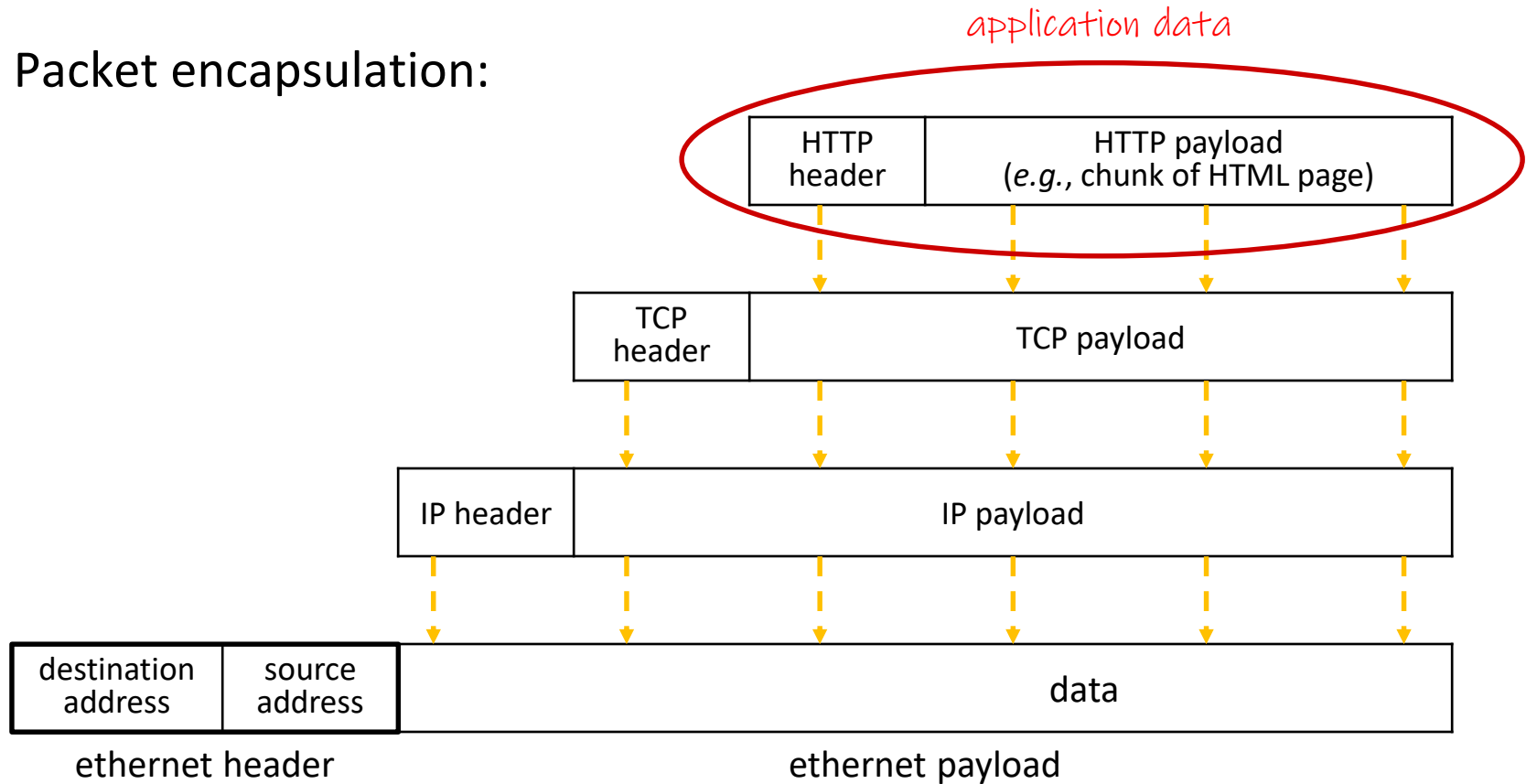
# The Application Layer

- ❖ Application protocols
  - The format and meaning of messages between application entities
  - *e.g.*, HTTP is an application-level protocol that dictates how web browsers and web servers communicate
    - HTTP is implemented *on top of* TCP streams



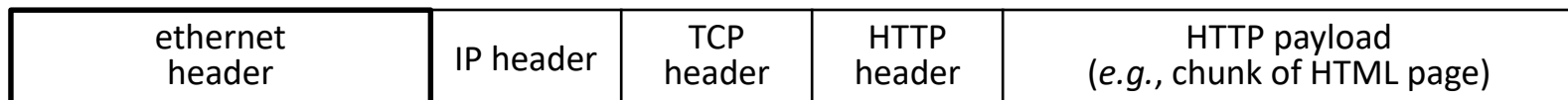
# The Application Layer

- ❖ Packet encapsulation:



# The Application Layer

- ❖ Packet encapsulation:



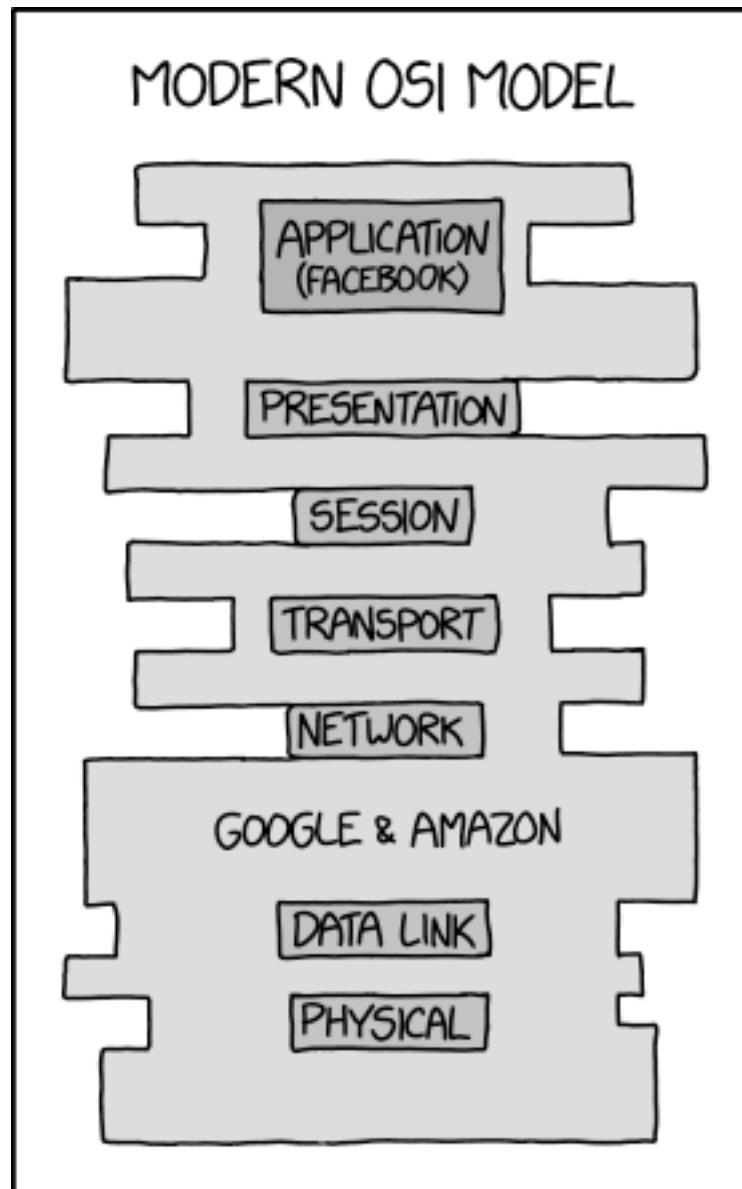
# The Application Layer

- ❖ Popular application-level protocols:
  - **DNS:** translates a domain name (*e.g.*, [www.google.com](http://www.google.com)) into one or more IP addresses (*e.g.*, 74.125.197.106)
    - Domain Name System
    - An hierarchy of DNS servers cooperate to do this
  - **HTTP:** web protocols
    - Hypertext Transfer Protocol
  - **SMTP, IMAP, POP:** mail delivery and access protocols
    - Secure Mail Transfer Protocol, Internet Message Access Protocol, Post Office Protocol
  - **SSH:** secure remote login protocol
    - Secure Shell
  - **bittorrent:** peer-to-peer, swarming file sharing protocol

# netcat demo (if time)

- ❖ netcat (`nc`) is “a computer networking utility for reading from and writing to network connections using TCP or UDP”
  - <https://en.wikipedia.org/wiki/Netcat>
  - Listen on port: `nc -l <port>`
  - Connect: `nc <IPaddr> <port>`
    - Local host: `127.0.0.1`

# In Other Words...



<https://xkcd.com/2105/>