NETWORK PROTOCOLS
IP, UDP, AND TCP

George Porter
Week 1
Fall 2020

UC San Diego

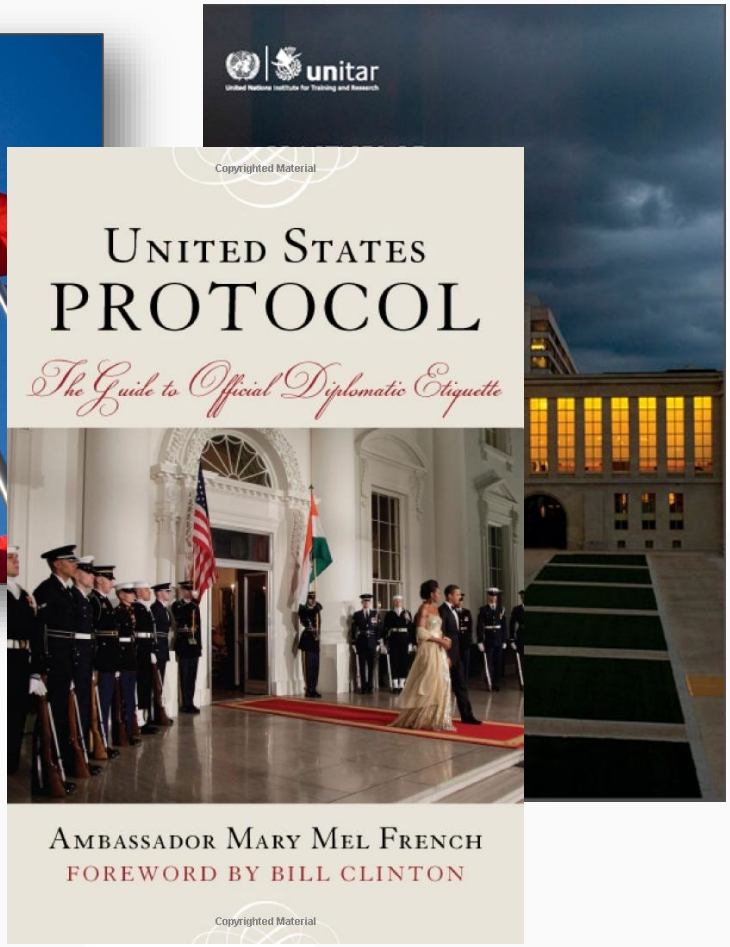
ATTRIBUTION

- These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
- These slides incorporate material from:
  - Alex C. Snoeren, UC San Diego
  - Michael Freedman and Kyle Jamieson, Princeton University
  - Internet Society
  - Computer Networking: A Top Down Approach

UC San Diego

WHAT ARE PROTOCOLS?

Image from public domain

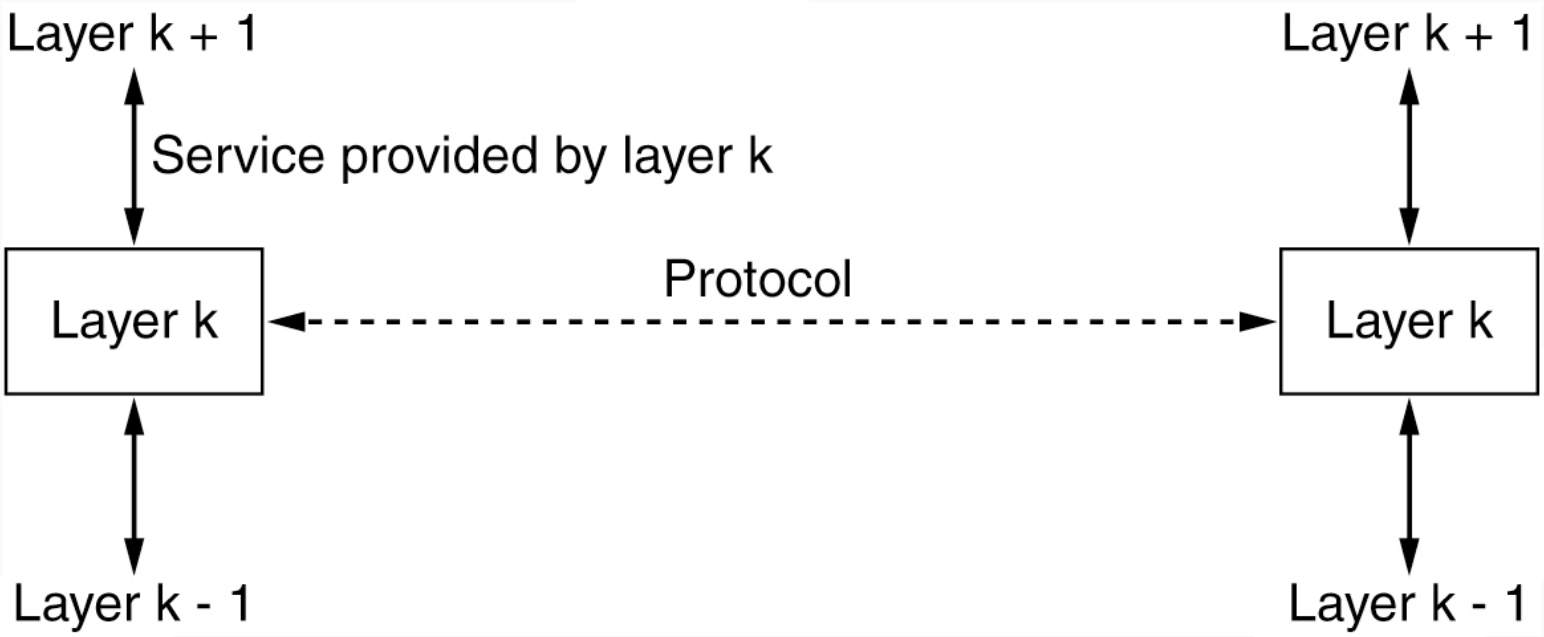


- Explicit and implicit conventions for how to communicate
  - Not for what is communicated
- Enables heterogeneous architectures, languages, OSes, byte ordering, …

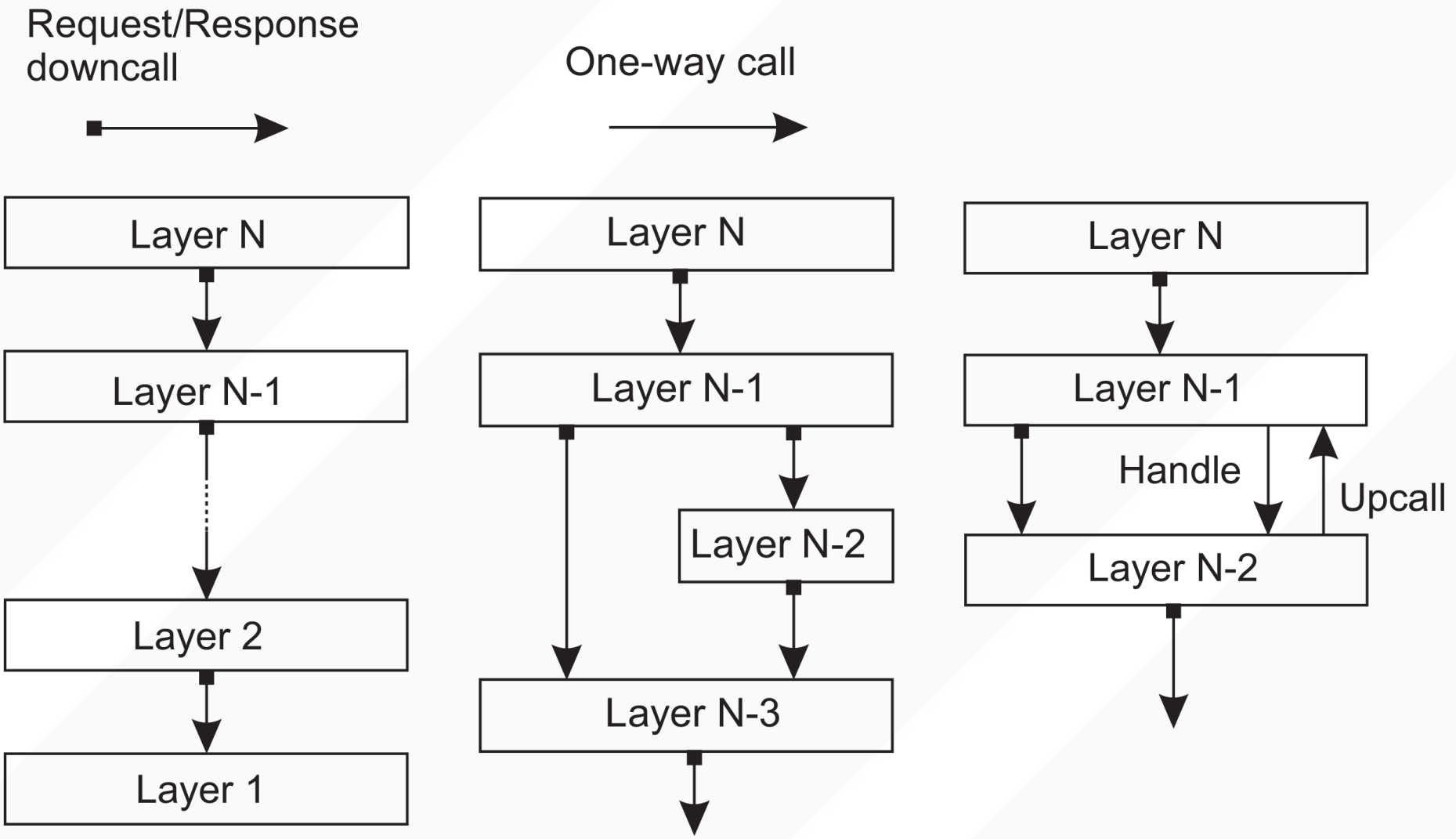
PROTOCOL LAYERING

- Sub-divide responsibilities
  - Each layer relies on services from layer below
  - Each layer exports services to layer above
- Interface between layers defines interaction
  - Hides implementation details (encapsulation)
  - Layers can change without disturbing other layers (modularity)
- Interface among peers in a layer is a protocol
  - If peers speak same protocol, they can interoperate

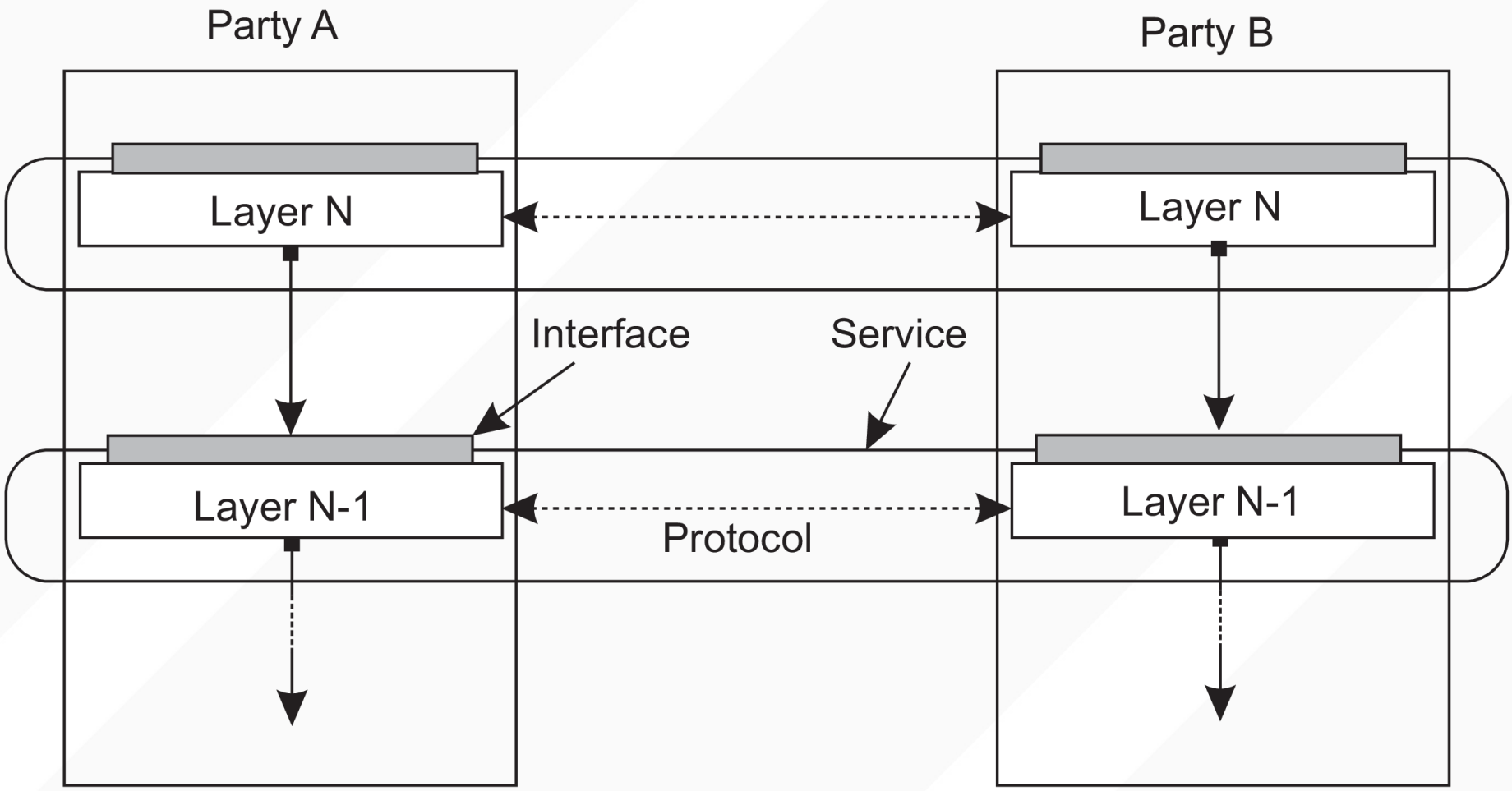
LAYERING: A DETAILED LOOK



COMPOSING LAYERS



SERVICE AND PROTOCOL INTERFACES



WHERE DO PROTOCOLS COME FROM?

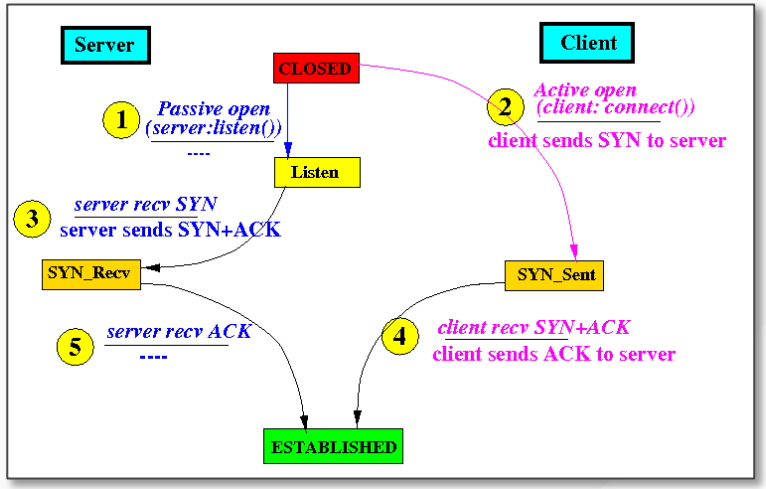
- Standards bodies
  - IETF: Internet Engineering Task Force
  - ISO: International Standards Organization
- Community efforts
  - “Request for comments”
  - Bitcoin
- Corporations/industry
  - RealAudio™, Call of Duty multiplayer, Skype

HOW ARE PROTOCOLS SPECIFIED?

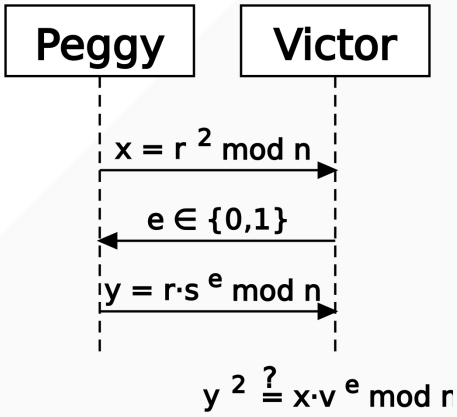
Prose/BNF

```
3.2.  HEADER FIELD DEFINITIONS

     These rules show a field meta-syntax, without regard for the
particular  type  or internal syntax.  Their purpose is to permit
detection of fields; also, they present to  higher-level  parsers
an image of each field as fitting on one line.

field       =  field-name ":" [ field-body ] CRLF

field-name  =  1*<any CHAR, excluding CTLs, SPACE, and ":">

field-body  =  field-body-contents
               [CRLF LWSP-char field-body]

field-body-contents =
              <the ASCII characters making up the field-body, as
               defined in the following sections, and consisting
               of combinations of atom, quoted-string, and
               specials tokens, or else consisting of texts>
```

State transition diagrams

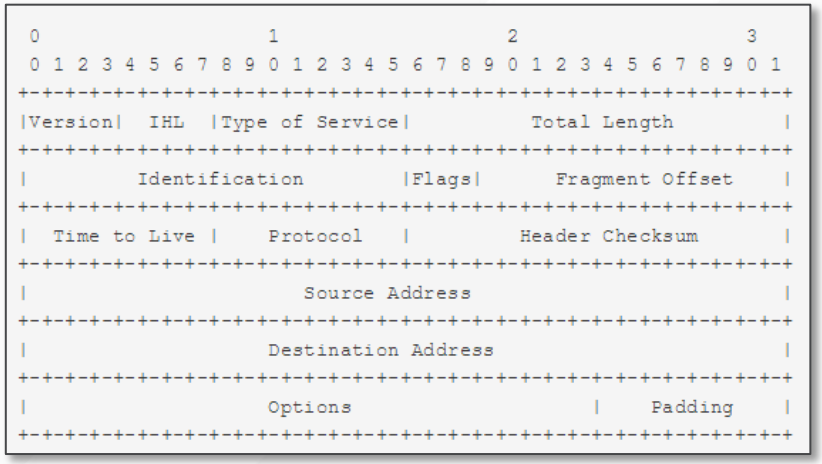


Message Sequence Diagram



By Stefan Birkner, cc-by-sa-2.5,2.0,1.0

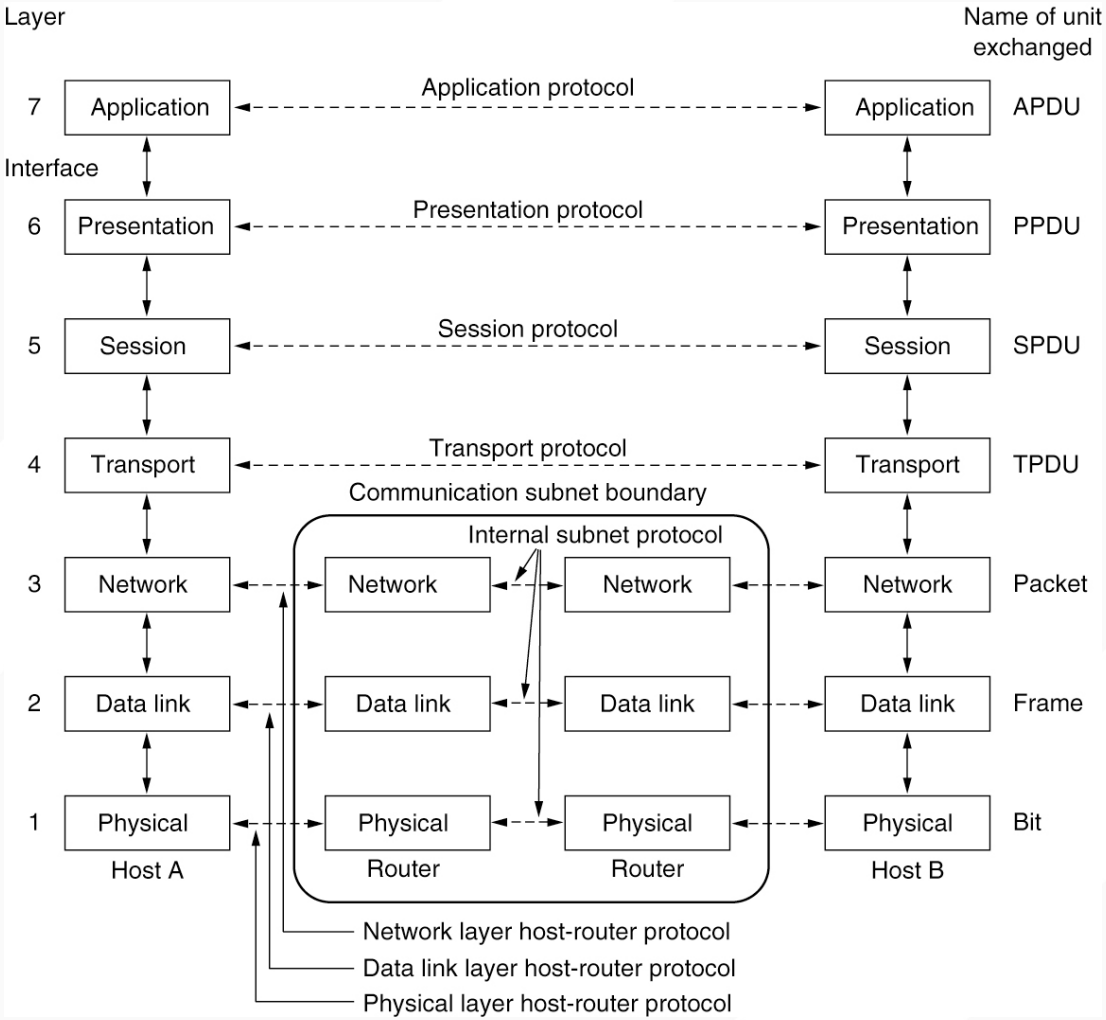
Packet formats



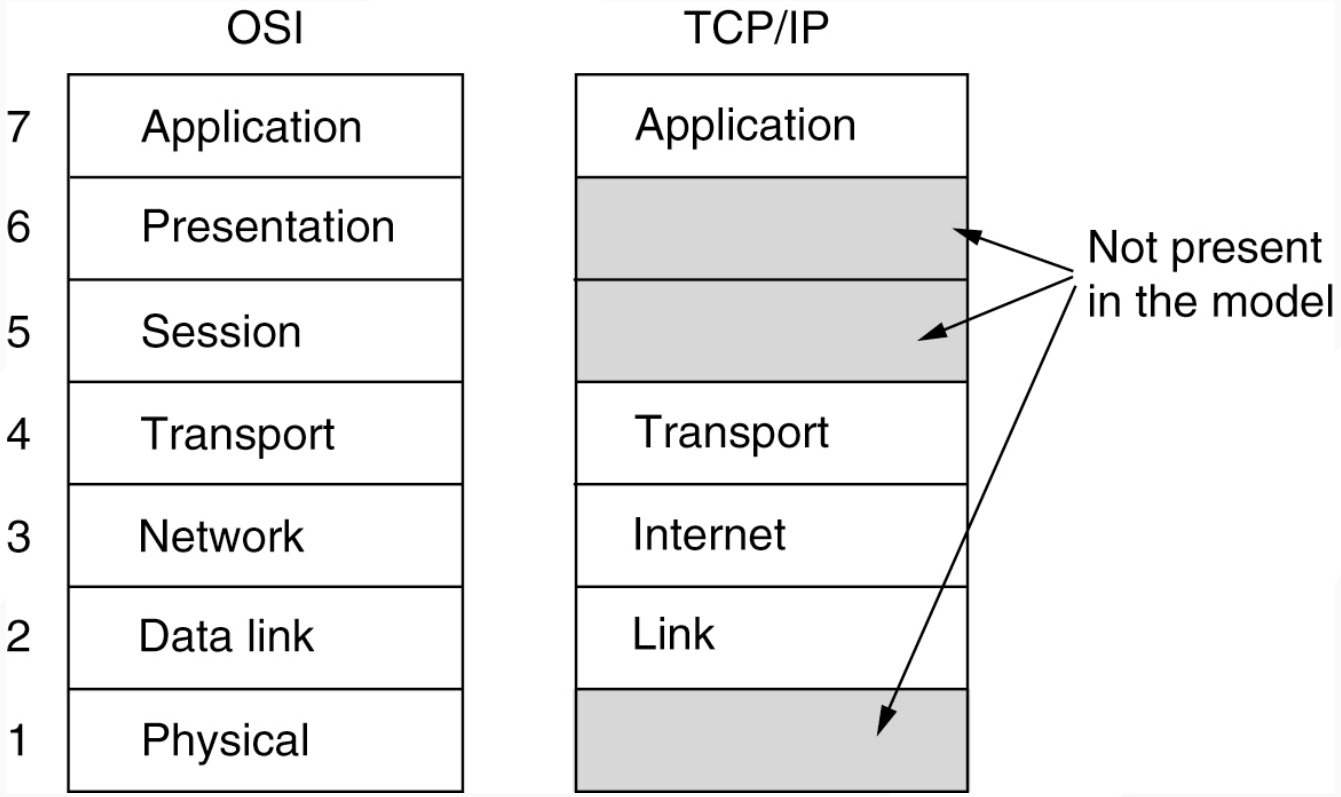
THE “OSI 7 LAYER” PROTOCOL STACK

- Principles for the seven layers
  - Layers created for different abstractions
  - Each layer performs well-defined function
  - Function of layer chosen with definition of international standard protocols in mind
  - Minimize information flow across interfaces between boundaries
  - Number of layers should be optimum
- Three concepts central to the OSI model:
  - Services
  - Interfaces
  - Protocols

OSI 7-LAYER STACK



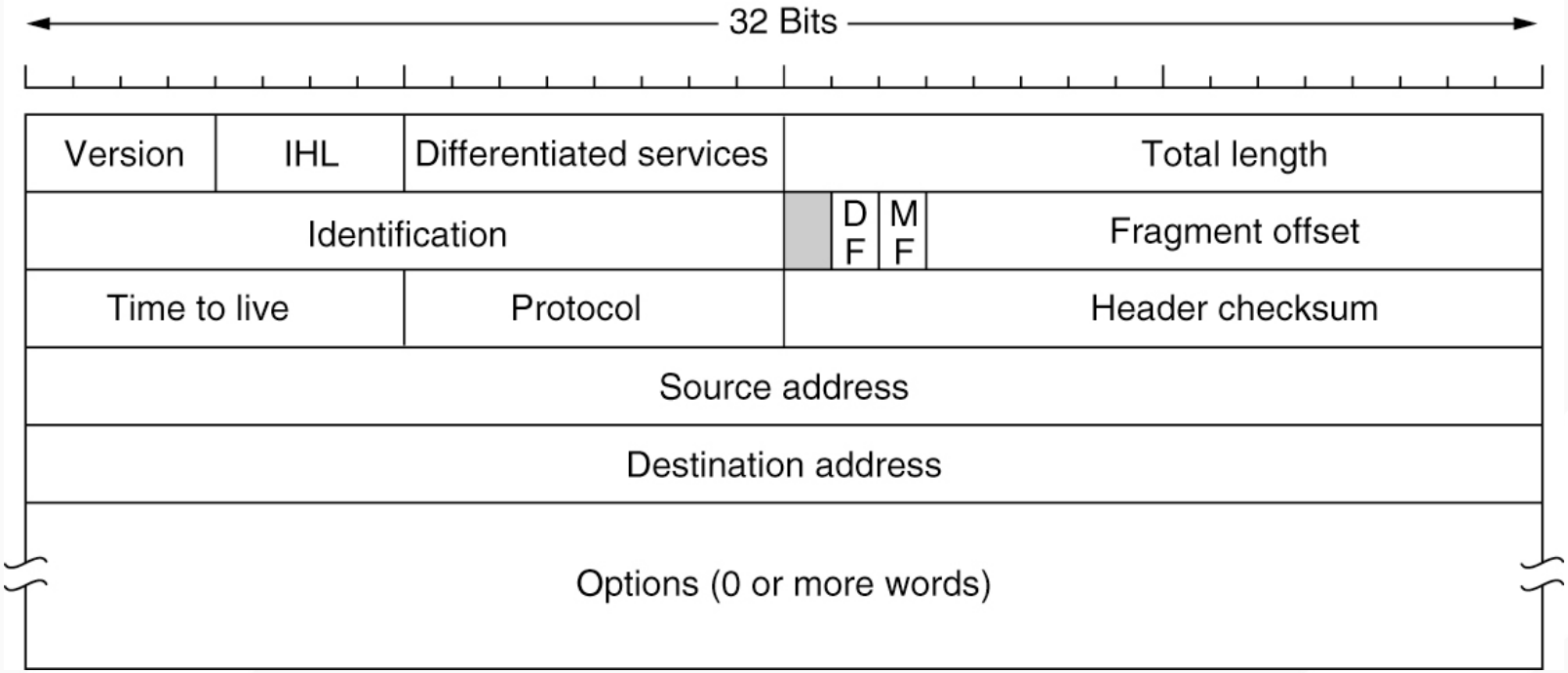
WHAT ACTUALLY IS DEPLOYED: THE TCP/IP MODEL



LINK AND IP LAYERS

- The Link Layer
  - Lowest layer in the model
  - Specific to the link technology (e.g. wireless LAN, wired Ethernet, Cable Modem, Fiber optic cable, etc…)
- The Internet Layer (IP)
  - Permits hosts to inject packets into any network and have them travel independently to the destination
  - Defines an official packet format and protocol called IP (Internet Protocol)
  - Defines a companion protocol called ICMP (Internet Control Message Protocol) that helps IP function

IP HEADER EXAMPLE



- IP protocol delivers packets with ***best effort*** reliability to destination
  - Along the way, packets can be **dropped/lost**, **their relative ordering can change**, and **they can become corrupted**

TRANSPORT AND APPLICATION LAYERS

- The Transport Layer
  - Process-to-process communication
  - Uses two end-to-end transport protocols
    - TCP (Transmission Control Protocol)
    - UDP (User Datagram Protocol)
- The Application Layer
  - Contains all the higher-level protocols
    - Web, streaming video, Zoom, Minecraft, etc…

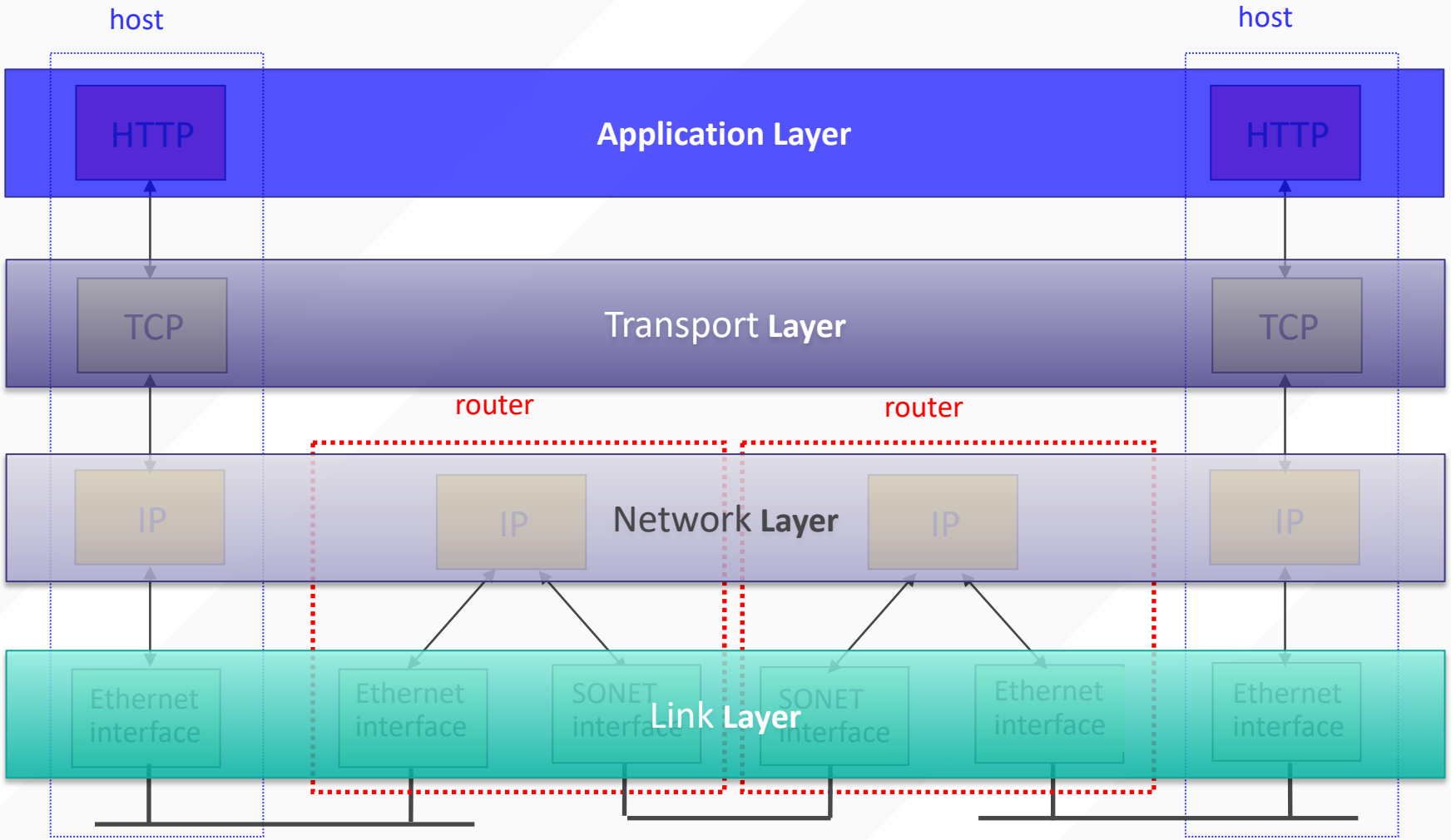
UNIGRAM DATA PROTOCOL (UDP)

- Offers applications same best-effort delivery model that IP provides
  - Except ensures that packets aren't corrupted
  - But can still be lost and arrive out of order
  - Send(p1); Send(p2); Send(p3)
    - Yet arrival order is p3 then p1 then p2
    - Or even: p3 then p1 (and p2 never arrives)

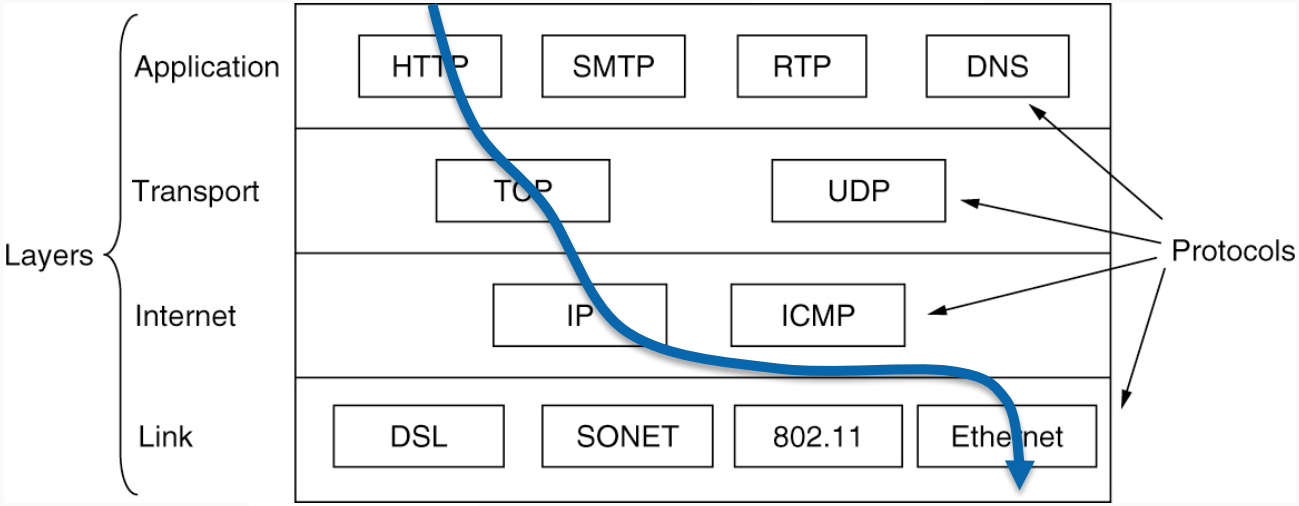
TRANSMISSION CONTROL PROTOCOL (TCP)

- Remember 1 GB ~ 715,000 packets?
  - With IP best effort delivery model, your application would have to keep track of each packet, whether it got there, did it get lost, did some get reordered??
- TCP offers *infinite bytestream* abstraction
  - If you put *N* bytes into TCP connection as sender, those *N* bytes will arrive to the destination in order, without loss, and without corruption
    - Compelling abstraction for higher-level applications such as web, video, music, …
    - How does TCP do that? Topic for CSE 123/222a!

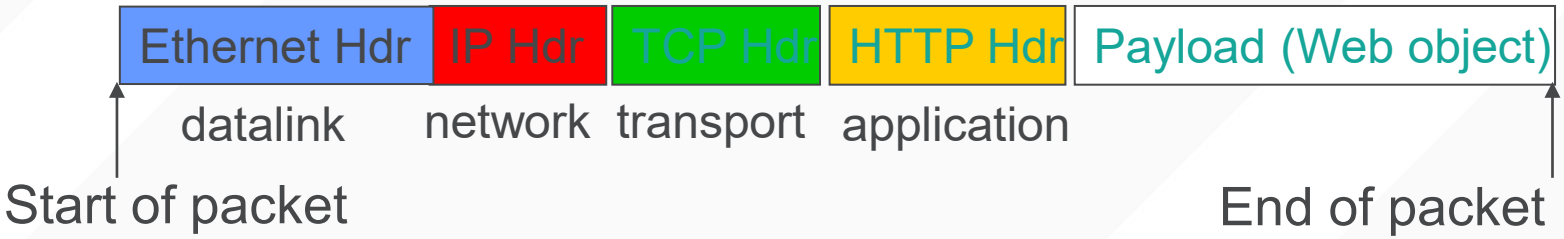
TCP/IP PROTOCOL STACK



ENCAPSULATION VIA HEADERS

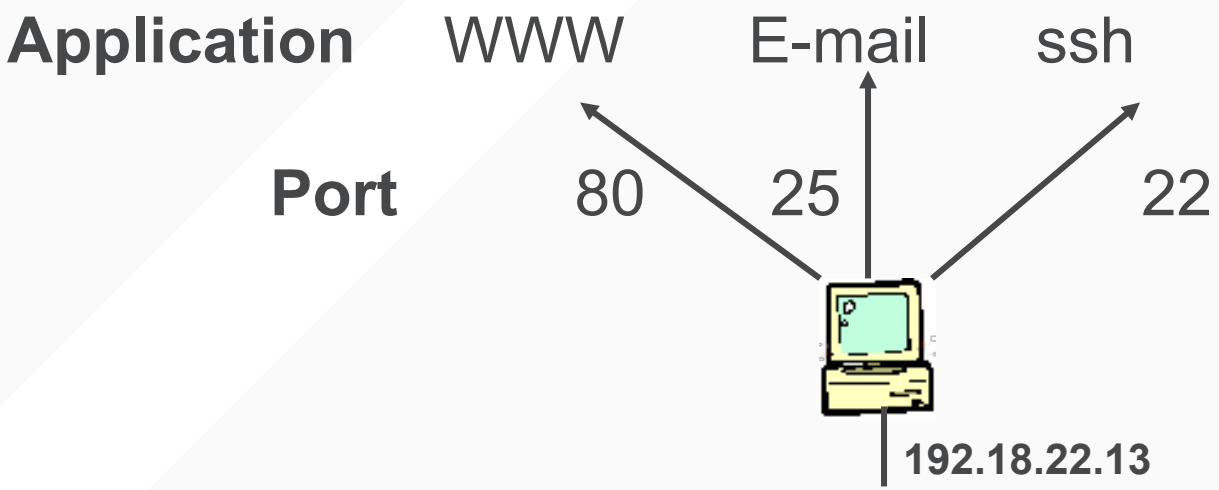


- Typical web packet:



- Notice that layers add overhead
  - Space (headers), effective bandwidth
  - Time (processing headers), latency

TCP AND UDP PORTS



- IP addresses identify hosts
  - But a host has many applications!
- Ports (16-bit identifier) identify a process

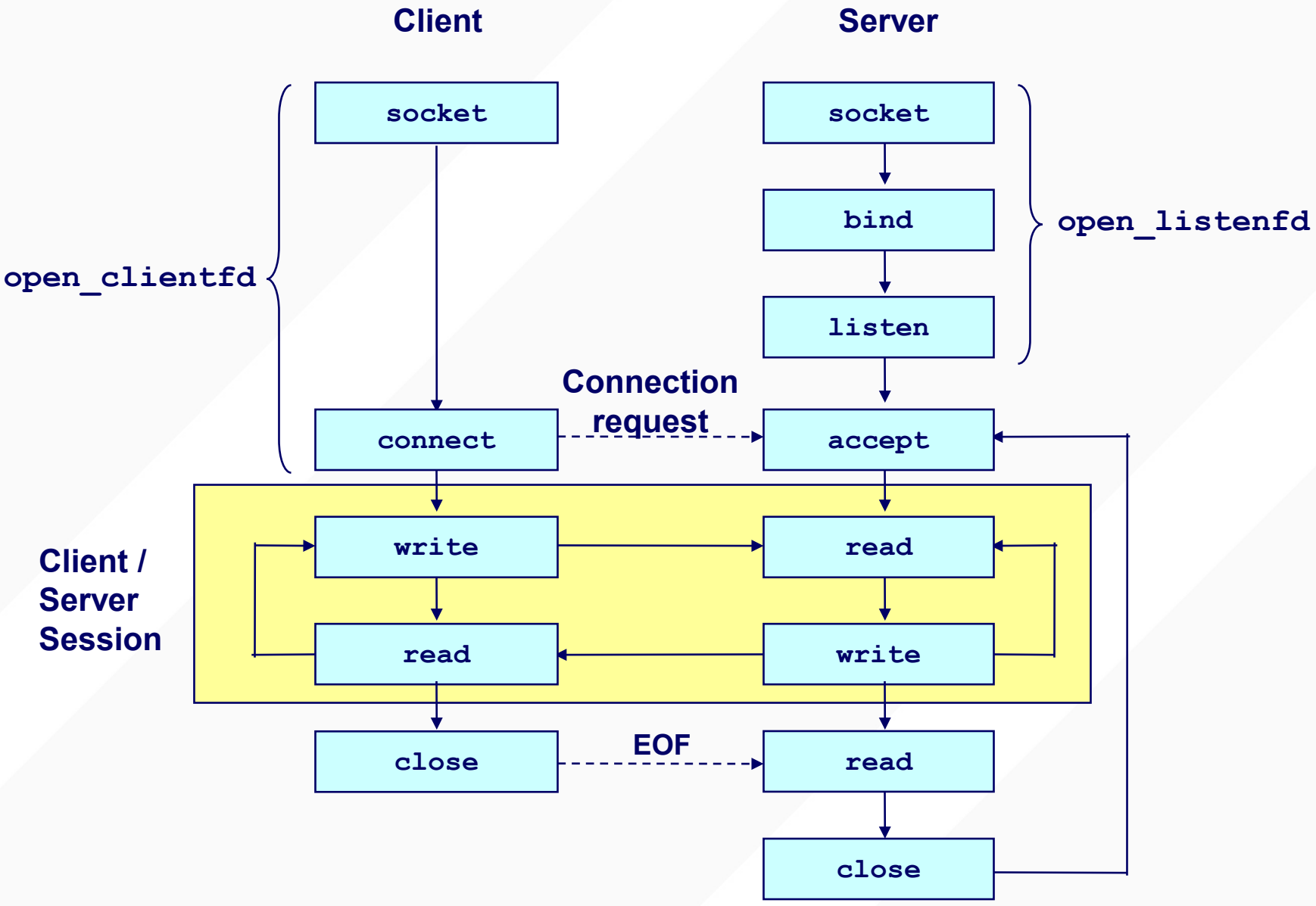
COMMON "WELL KNOWN" TCP PORTS

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

BERKELEY SOCKETS API

- What is a socket?
  - The point where a local application attaches to the network
  - An interface between an application and the transport protocol
  - An application creates the socket
- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket
- Sockets are where your program send and receive data

CLIENT AND SERVER SOCKETS (SYSTEM CALLS)



TCP EXAMPLE IN GO

```
package main

import (
	"flag"
	"log"
	"net"
	"strconv"
)

func main() {
	port := flag.Int("port", 3333, "Port to accept connections on.")
	host := flag.String("host", "127.0.0.1", "Host or IP to bind to")
	flag.Parse()

	l, err := net.Listen("tcp", *host+":"+strconv.Itoa(*port))
	if err != nil {
		log.Panicln(err)
	}
	log.Println("Listening at '"+*host+"' on port", strconv.Itoa(*port))
	defer l.Close()

	for {
		conn, err := l.Accept()
		if err != nil {
			log.Panicln(err)
		}

		go handleRequest(conn)
	}
}
```

Code courtesy of @golergka

TCP EXAMPLE IN GO (CON'T)

```
func handleRequest(conn net.Conn) {
	log.Println("Accepted new connection.")
	defer conn.Close()
	defer log.Println("Closed connection.")

	for {
		buf := make([]byte, 1024)
		size, err := conn.Read(buf)
		if err != nil {
			return
		}
		data := buf[:size]
		log.Println("Read new data from connection", data)
		conn.Write(data)
	}
}
```

Code courtesy of @golergka

UC San Diego