School of Computer Science

University of St.Gallen

"From insight to impact"

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBA ACCREDITED

Fundamentals of Computer Science for Business Studies

Week 6: Computer Networks and Distributed Systems

Prof. Dr. Stephan Aier
stephan.aier@unisg.ch

Prof. Dr. Simon Mayer
simon.mayer@unisg.ch

Prof. Dr. Siegfried Handschuh
siegfried.handschuh@uniusg.ch

Quiz 05 (graded)

On Canvas

12.17-12.37 (max. 15. minutes)

Course Outline

Date (L/GE)	Topic (L/GE)	Exercise/E	Coaching/CO	Who
L01: 22.09.2021 **GE01:** 24.09.2021	Intro, Course Overview, Computing Basics	Setting up Python environments	*no coaching*	**L/GE:** Simon Mayer, Stephan Aier
L02: 29.09.2021 **AU02:** 01.10.2021	Programming I Quiz 1: Programming Basics, ungraded	Assignment 1: Programming Basics	27.09.2021	**L/GE:** Stephan Aier
L03: 06.10.2021 **AU03:** 08.10.2021	Programming II Quiz 2: Programming I, ungraded	Assignment 2: Programming I	04.10.2021	**L/GE:** Stephan Aier
L04: 13.10.2021 **GE04:** 15.10.2021	Programming III Quiz 3: Programming II, **graded**	Assignment 3: Programming II	11.10.2021	**L/GE:** Stephan Aier
L05: 20.10.2021 **GE05:** 22.10.2021	Programming IV Quiz 4: Programming III, ungraded	Assignment 4: Programming III	18.10.2021	**L/GE:** Stephan Aier
L06: 27.10.2021 **GE06:** 29.10.2021	Distributed Systems Quiz 5: Programming IV, **graded**	Assignment 5: Programming IV	25.10.2021	**L/GE:** Simon Mayer
--	*semester break*			
L07: 17.11.2021 **GE07:** 19.11.2021	Databases, Structured Data, SQL Quiz 6: Distributed Systems, **graded**	Assignment 6: Distributed Systems	15.11.2021	**L/GE:** Stephan Aier
L08: 24.11.2021 **GE08:** 26.11.2021	Data Science I: Intro Quiz 7: Databases, SQL, **graded**	Assignment 7: Databases, SQL	22.11.2021	**L/GE:** Stephan Aier
L09: 01.12.2021 **GE09:** 03.12.2021	Data Science II: Data Wrangling Quiz 8: Data Science I, ungraded	Assignment 8: Data Science I	29.11.2021	**L/GE:** Stephan Aier
L10: 08.12.2021 **GE10:** 10.12.2021	Data Science III: Data Visualization Quiz 9: Data Science II, **graded**	Assignment 9: Data Science II	06.12.2021	**L/GE:** Stephan Aier
L11: 15.12.2021 **GE12:** 17.12.2021	Data Science IV: Machine Learning Quiz 10: Data Science III, ungraded	Assignment 10: Data Science III	13.12.2021	**L/GE:** Siegfried Handschuh
L12: 20.12.2021 **L/GE12:** 22.12.2021	Data Science V: Machine Learning, Wrap up Quiz 11: Data Science IV, **graded** (22.12.2021)	*no exercise*	20.12.2021	**L/GE:** Siegfried Handschuh

Computer Networks and Distributed Systems

This lecture introduces **basics of distributed systems**, i.e. systems that are based on networks and internetworks between machines. The goal is for you to get a **broad technical understanding** of the main mechanisms at work in today's **Internet and on the World Wide Web**, and of **how to use these mechanisms from a program.**

This block's topics include:

- Computer Networks Basics and History
- The Layered Networking Stack
- The Internet and the Web
- Definition and Usage of Web APIs

Parts of this Lecture are based on:

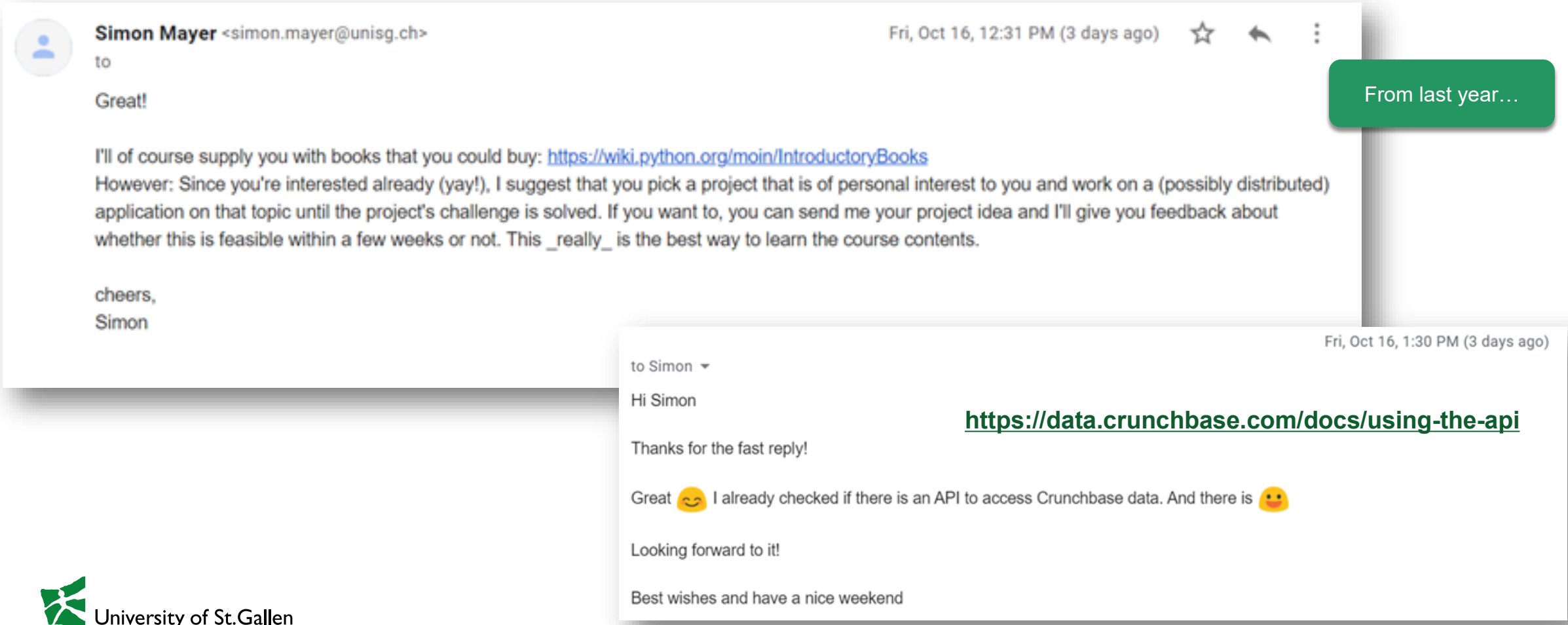
S. Mayer and F. Michahelles: *IoT: Foundations and Applications*, UC Berkeley
S. Mayer: *Introduction to Computer Systems and Networks*, University of St. Gallen
B. Christian and T. Griffiths: *Algorithms to Live By*

Central Learning Objective

Understand how **information** from a computer makes its way within **networks** to the cloud and back, and how it interacts with **application logic**.
And implement some of this ☺

Why all of this?

Because it enables you to do **fascinating things in many different domains**!



This Week's Menu

Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

Guided Exercise

Briefing: Security
Stock Exchange

I'm showing demos (available on Canvas) **locally on my machine**. You can execute them as well if you want.

1. Install a **python interpreter.** This is available and described in https://www.python.org/downloads/
2. Put code into **.py files** and then run it from your **Terminal** (e.g., PowerShell on Win or Terminal on Mac)
3. Download FCS-BWL-12-ComputerNetworksDistributedSystems-Demos.zip from Canvas

Remember Encapsulation?

Week 5: Programming 4

This week's goals and learning objectives

- The goal of this week's lectures is to theoretically and practically introduce you to the basic concepts of **object orientation**:
  - **Object** and **Class**
  - **Method** and **Attribute**
  - **Inheritance**
  - **Polymorphism**
  - **Encapsulation**

… and their implementation in Python (**object-oriented programming**)

After this week, you should
- know
- be able to understand and
- **write programs** utilizing

5

Basics

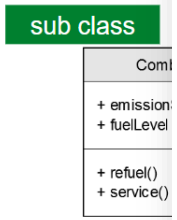
Encapsulation

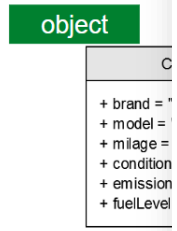
- Often it is **not** a good idea to just **manipulate** an object's attribute value **directly**
  → CarOfMax.milage -= 20000

- Instead, it is a better idea to call a method on this object. The **method takes care of manipulating** attributes appropriately
  → CarOfMax.drive(300)
    → CarOfMax.milage += 300
    → CarOfMax.condition -= (300 / 1000)

  → CarOfMax.drive(-20000)
    → you cannot drive negative distances / distance must be >= 0

- Furthermore, for a caller of a method it is important to know
  - How to call the method (arguments)
  - What effect/return to expect
- It is **NOT / SHOULD NOT BE important to know HOW the effect is achieved**
  - E.g., CarOfMax.service()





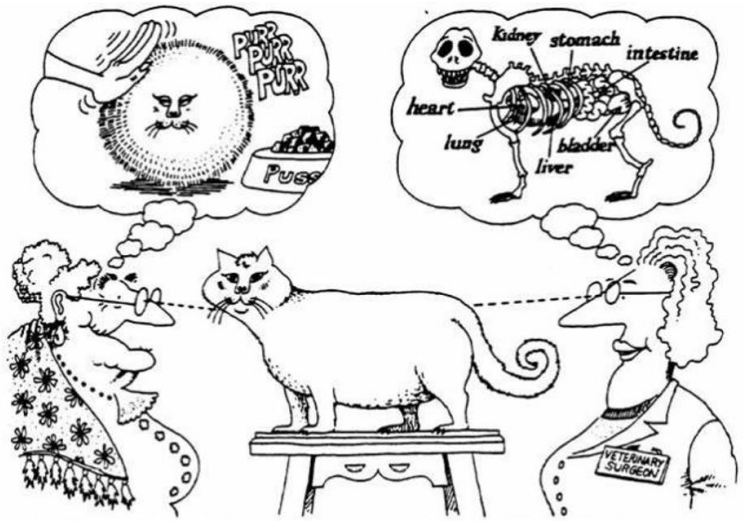
© Oct-21, SCS-HSG

Basics

Encapsulation

User perspective
needs to know inputs, outputs

Black box model



Designer perspective
needs to know the inner components and dependencies

White box model

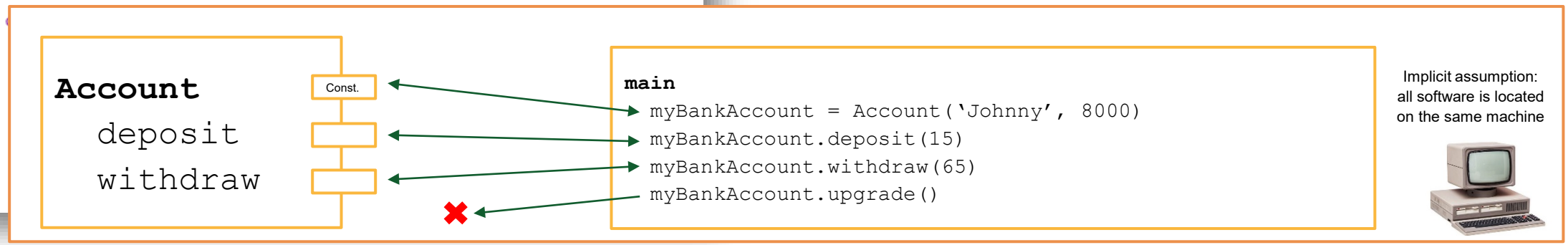
[Booth 1994]

© Mar-21, SCS-HSG 13

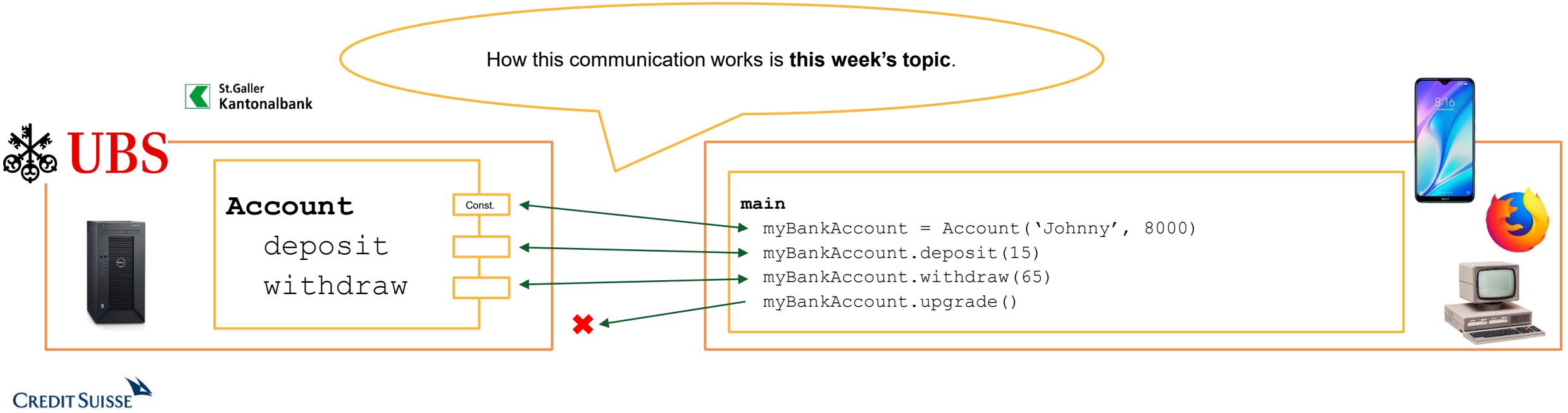
Another Example

```
class Account:
    """Account class for maintaining a bank account balance."""

    def __init__(self, name, balance):
        """Initialize an Account object."""

        # if balance is less than 0.00, raise an exception
        if balance < Decimal('0.00'):
            raise ValueError('Initial balance must be >= to 0.00.')

        print('Initializing: ', name, balance)
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        """Deposit money to the account."""

        # if amount is less than 0.00, raise an exception
        if amount < Decimal('0.00'):
            raise ValueError('amount must be positive.')

        print('Depositing ', amount)
        self.balance = self.balance - amount
```

- The `Account` class is **encapsulated** and account instances are accessed through **controlled interfaces**.
- Method **implementations** are **private** to the class and thus hidden from clients (**main**, in this case)
- This **information hiding** allows us to **separate** the **implementation** of `Account` from the **usage** of `Account` instances.
- This **protects other parts** of the code from modifications in case the implementation of `Account` changes! Thereby, it allows for software systems to **scale**!



It's the core OOP concepts again – but distributed!



- The `Account` **application** is **encapsulated** and **clients** access it through **controlled interfaces**.
- This **protects client applications** of the application from modifications in case the implementation of the `Account` application changes, as long as the **API remains stable**.
- We say that the interaction between client and application (on the server) happens **through an Application Programming Interface**, or "*API*"

So what's an API then?

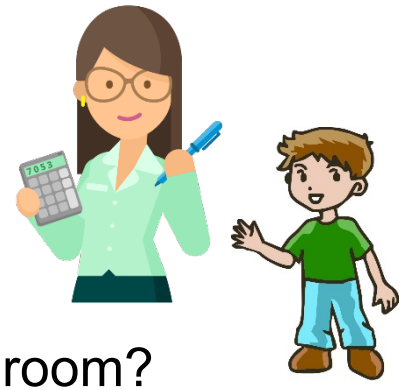
Wikipedia: "In computer programming, an **application programming interface** (**API**) is a set of subroutine definitions, communication protocols, and tools for building software."

Also Wikipedia: "In building applications, an API **simplifies programming by abstracting** the underlying implementation and only exposing objects or actions the developer needs."

RedHat, Inc.: "An API is a set of **definitions and protocols for building and integrating application software**. (…) APIs let your product or service communicate with other products and services **without having to know how they're implemented**. This can simplify app development, saving time and money. When you're designing new tools and products—or managing existing ones—APIs give you flexibility; simplify design, administration, and use; and provide **opportunities for innovation**."

https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Application Programming Interfaces (APIs)



Imagine “interfacing with” (e.g., entering) a **room**…

…how do YOU access that room?

…how do YOUR KIDS access that room?

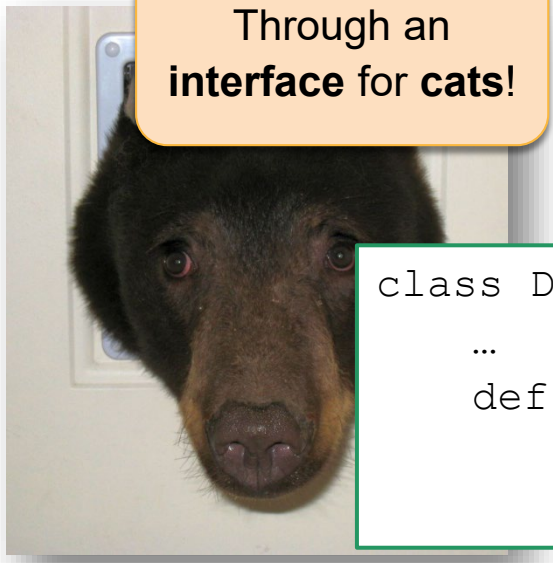
…how does YOUR CAT access that room?

…how does YOUR ROBOT access that room?

Through an **interface** for **grown-ups**!

Through an **interface** for **kids**!

Through an **interface** for **cats**!



Through an **interface** for **programs**!

```
class Door:
    …
    def openDoor():
        …
        # Application code to open the door
```

Application Programming Interfaces (APIs)

```
class Door:
    …
    def openDoor():
        …
        # Application code to open the door
```

Through an **interface** for **programs**!

e.g., the program that runs on this robot…

```
doCleanRoom();

door13 = Door('13')
door13.openDoor();

enterRoom(door13);
doCleanRoom();
```

Web + API = Web API

```
class Door:
    …
    def openDoor():
        …
        # Application code to open the door
```

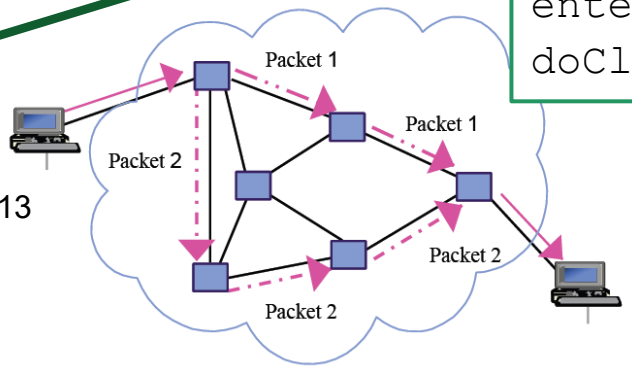
Through an **interface** for **programs**!

http://mybuilding.org/door13

Web Server

e.g., the program that runs on this robot…

```
doCleanRoom();

// Send a Web request to the door
PUT /door13 HTTP/1.1
Host: mybuilding.org
{ state : 'open' }

enterRoom();
doCleanRoom();
```



Web APIs massively lower the usage barrier for clients to use services (and devices ☺)

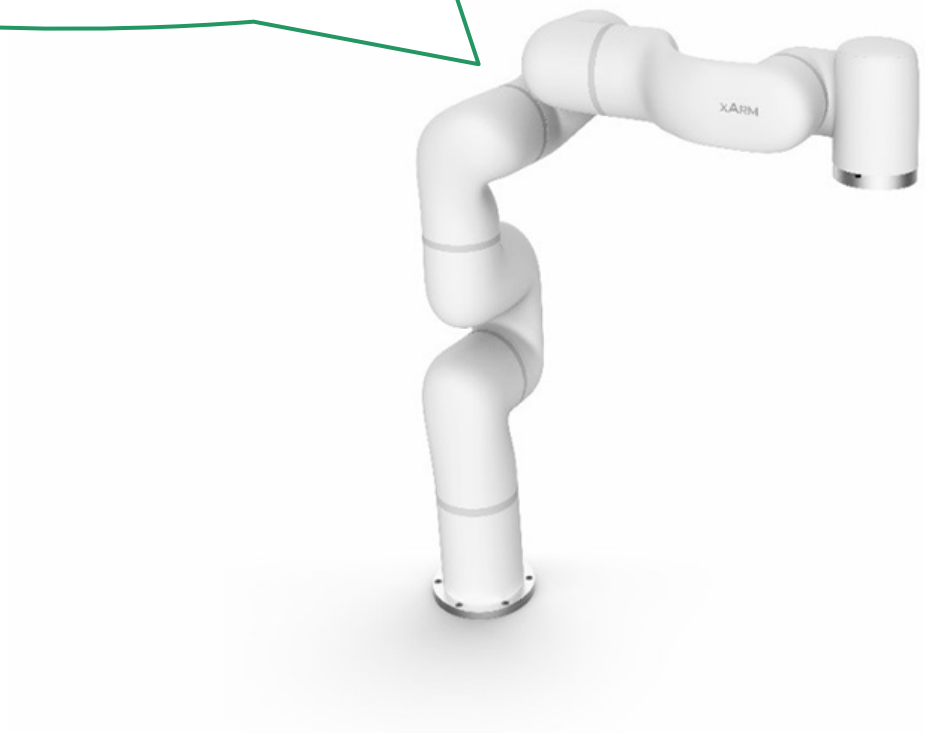
Cherrybot: https://api.interactions.ics.unisg.ch/cherrybot

Notes

- User only gets a **single entry point**
- By **navigating** the hypermedia structure, user can **control a robot**
- To accomplish this, it is sufficient to be a «**Web User**». We train this for years…
  - You don’t need to be a robot control expert
  - You don’t need to be an embedded systems designer
  - You don’t need to be a networking person
  - You don’t need to be an API designer
  - You don’t need to be a human-computer interaction expert

Web APIs massively lower the usage barrier for clients to use services (and devices ☺)

So the «**client**» is the Web browser in this case (or a python script ;-)) – but what does the «**server**» look like?



Simple Dynamic HTTP Server Example

We have a python script that reads out the **current system time** and prints it on the screen

```
from datetime import datetime

def getCurrentDateTime():
        today = datetime.now()
        return str(today)
```

https://docs.python.org/3/library/datetime.html

We want to serve that functionality over the Web…

https://docs.python.org/3/library/http.server.html

If you can do this, this means that you can serve ***any* functionality that you can express in Python** over the Web (including, for instance, a robot controller ☺)

Simple Dynamic HTTP Server Example

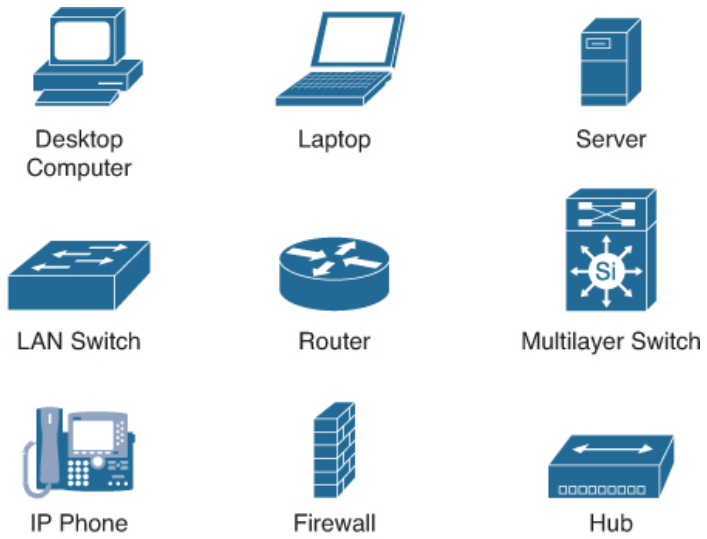
The current system time is now being **served within a Web resource** by this python **Web server**.

If my machine is on the **Internet** and the network is properly configured, this Web server is **publicly reachable** through my machine's **IP address** and on the **port** the server is running on. You can do this "traditionally" by putting a server into your basement or "modern" by using some service…

To **facilitate the usage** of this Web resource by users, one would now **register a URI that points to the IP address of the Web server** and make this URL **discoverable** by users (e.g., through a search engine like Google).

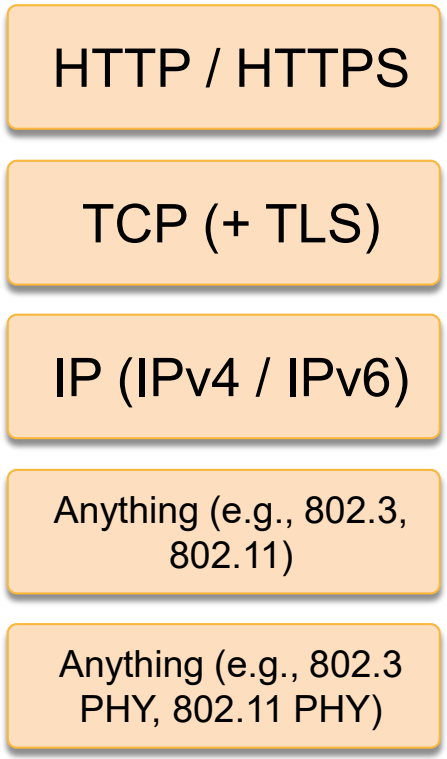
By the end of today, you should be able to explain all of this.

Break: Movie Screening ☺



Different networking devices operate on different layers of the stack

- **Repeater**: Physical and Data Link
- **Switch**: Physical and Data Link
- **Router**: Physical, Data Link and Network



Let's see how IP packets travel from your PC to a Web server

Warriors of the Net

(remember that this is a few decades old…)

https://www.youtube.com/watch?v=n7mtJ3ZV6xM

http://chittagongit.com/icon/network-devices-icon-5.html

Any Questions / Comments / Doubts / Concerns?

This Week's Menu

Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

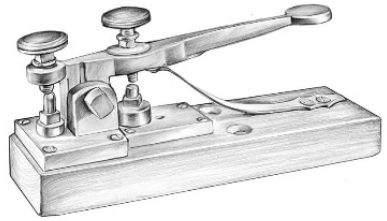
Guided Exercise

Briefing: Security
Stock Exchange

I'm showing all demos **locally on my machine**. You can do this as well if you want.

1. Install a **python interpreter.** This is available and described in https://www.python.org/downloads/
2. Put code into **.py files** and then run it from your **Terminal** (e.g., PowerShell on Win or Terminal on Mac)
3. Download FCS-BWL-12-ComputerNetworksDistributedSystems-Demos.zip from Canvas

Brief History of (Computer) Networks



May 24, **1844**: Samuel F. B. Morse (at the U.S. Supreme Court) to Alfred Vail (in Baltimore)

- "WHAT HATH GOD WROUGHT" – opening of the world's first telegraph line
- Even earlier (in 1833) accomplished by Carl Friedrich Gauss and Wilhelm Weber (but *not commercialized*)

March 10, **1876**: Alexander G. Bell to Thomas A. Watson

- "Mr. Watson, come here; I want to see you" – limits of "Tele"-communication

April 3, **1973**: Martin Cooper (Motorola) to Joel Engel (AT&T)

- "Joel, I'm calling you from a cellular phone. A real cellular phone: a handheld, portable, real cellular phone."

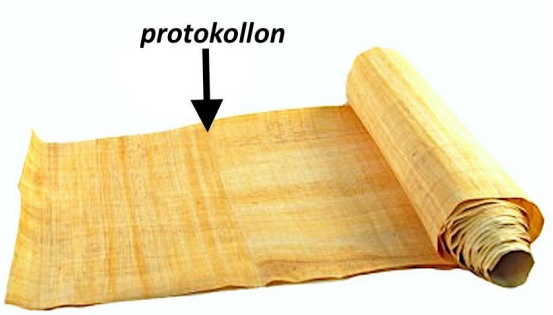
October 29, **1969**: Charley Kline (UCLA) to Bill Duval (Stanford Research Institute)

- "lo" (receiving machine crashed after first two characters of "login") – first transmission via the ARPANET

December 3, **1992**: Neil Papworth (Sema Telecom) to Richard Jarvis (Vodafone)

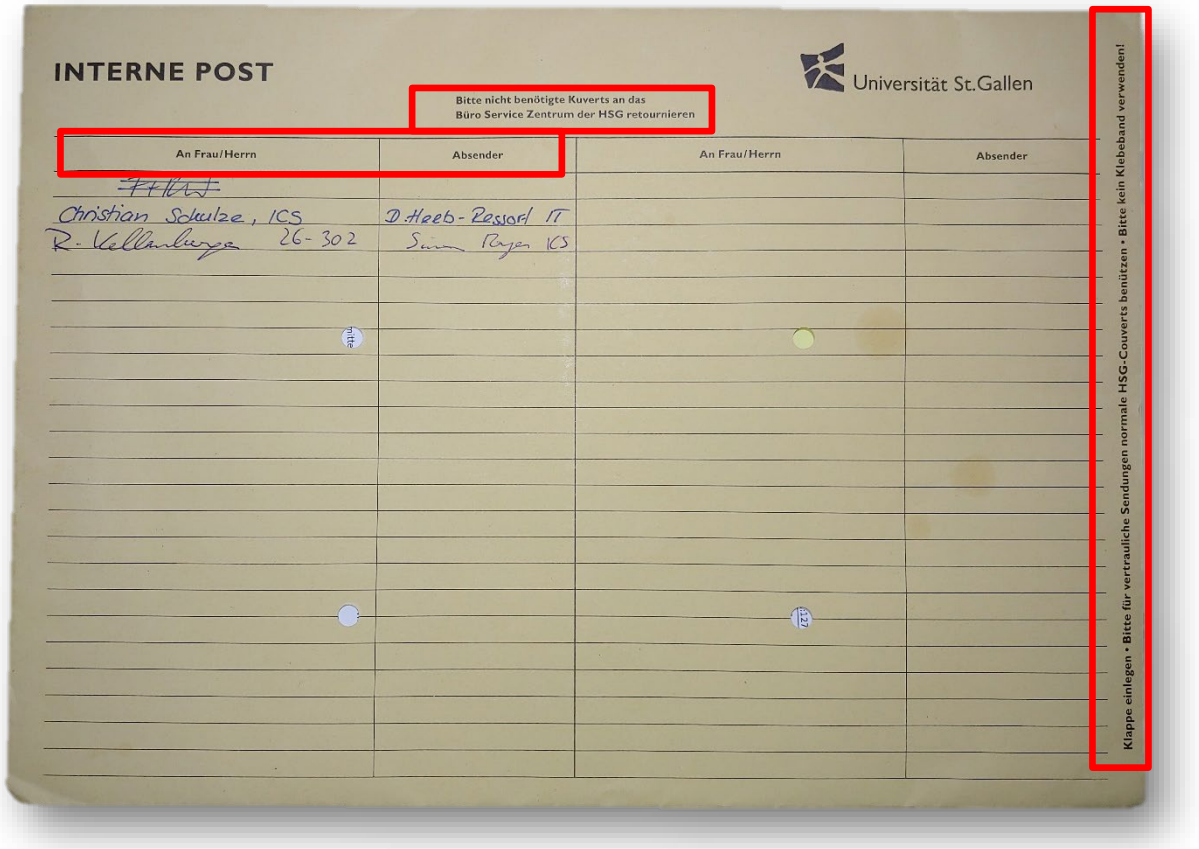
- "Merry Christmas." – first text message

Communication Protocols

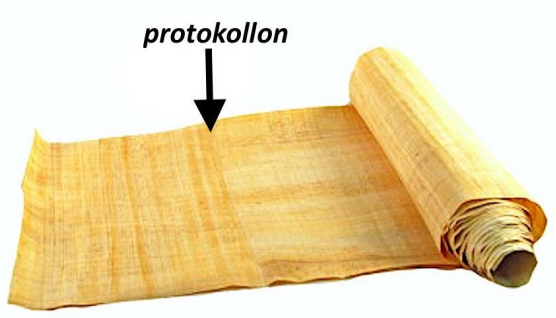


Protocols are at the **foundation** of **human** communication and of **machine** communication as well

- Proto-kollon: “First Glue” (outer page of a book or manuscript)
- Shared **convention of procedures and expectations**



Communication Protocols



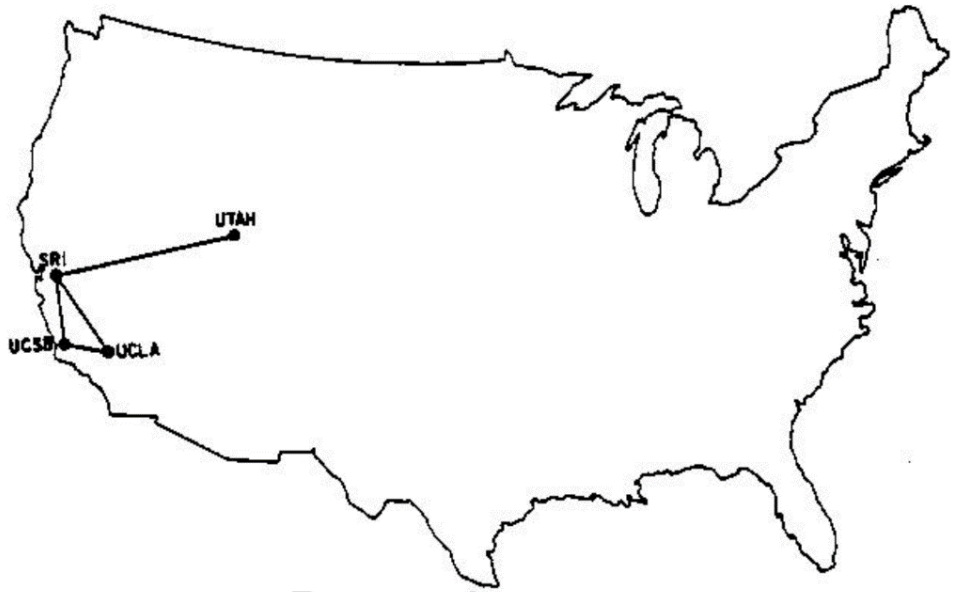
Protocols are at the **foundation** of **human** communication and of **machine** communication as well

- Proto-kollon: “First Glue” (outer page of a book or manuscript)
- Shared **convention of procedures and expectations**
- People: handshakes, hellos, etiquette, politesse, social norms
- **Machines: not much difference!**

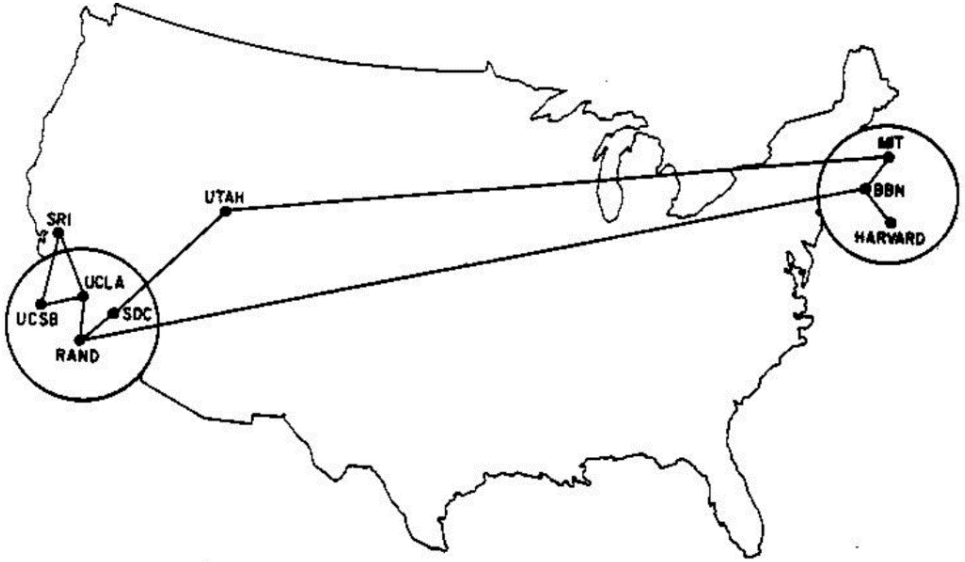
Telegraph, telephone, text message: **new conduits** for **person-to-person** communication

Internet: **computers become communication endpoints**, responsible for solving their own communication issues. Machine-to-machine communication challenges **mimic our own**…

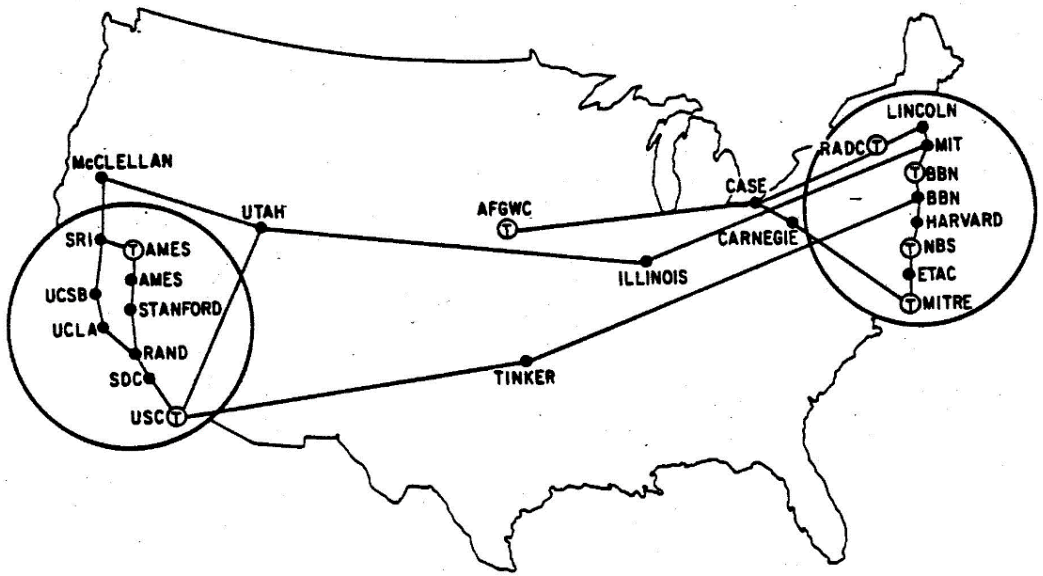
- “One after the other, please!” → Medium access control (**“Token Ring” - Standard**)
- “Come again?” → Carrier sensing and collision detection! (**“CSMA/CD” - Standard**)
- “What’s their number?” → Addressing! (**“Internet Protocol” - Specification**)
- “Slow down, please!” → Flow control! (**“TCP” - Specification**)



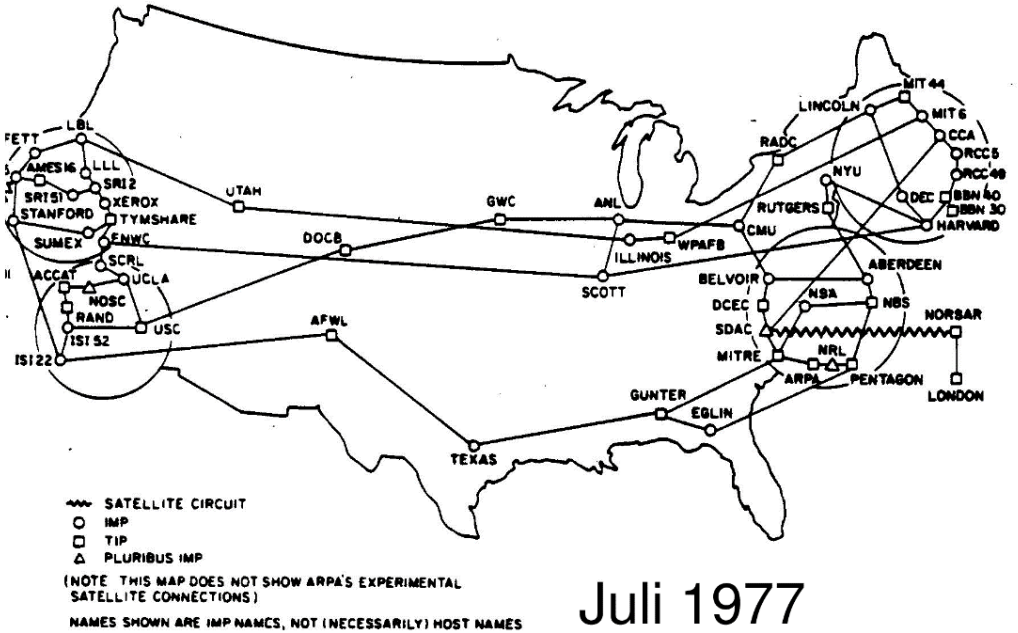
Dezember 1969



Juni 1970



März 1972



Juli 1977

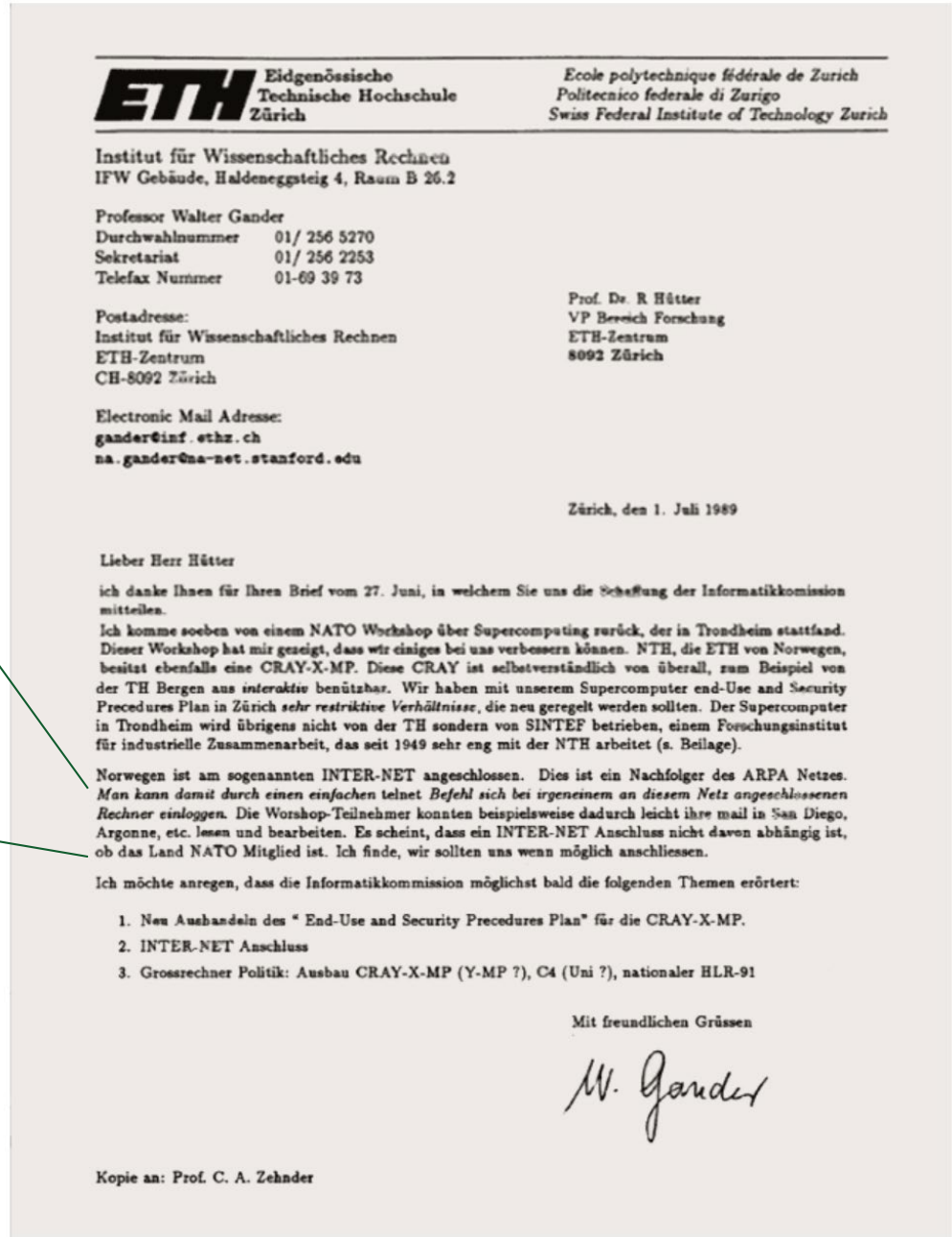
Uni

The Internet

"The workshop participants could, for example, easily read and edit their mail in San Diego, Argonne, etc. It seems that an INTER-NET connection is not dependent on whether the country is a NATO member. I think we should connect to it if possible."

Letter from **Prof. Walter Gander** (ETH Institute of Scientific Computing) to **Prof. Ralf Hütter** (ETH Vice President Research). Copy to **Prof. Carl Zehnder** (Founder of the Swiss Computer Science Association).

1 July 1989



ETH Eidgenössische Technische Hochschule Zürich
École polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für Wissenschaftliches Rechnen
IFW Gebäude, Haldeneggsteig 4, Raum B 26.2

Professor Walter Gander
Durchwahlnummer 01/ 256 5270
Sekretariat 01/ 256 2253
Telefax Nummer 01-69 39 73

Postadresse:
Institut für Wissenschaftliches Rechnen
ETH-Zentrum
CH-8092 Zürich

Electronic Mail Adresse:
gander@inf.ethz.ch
na.gander@na-net.stanford.edu

Prof. Dr. R Hütter
VP Bereich Forschung
ETH-Zentrum
8092 Zürich

Zürich, den 1. Juli 1989

Lieber Herr Hütter

ich danke Ihnen für Ihren Brief vom 27. Juni, in welchem Sie uns die Schaffung der Informatikkomission mitteilen.
Ich komme soeben von einem NATO Workshop über Supercomputing zurück, der in Trondheim stattfand. Dieser Workshop hat mir gezeigt, dass wir einiges bei uns verbessern können. NTH, die ETH von Norwegen, besitzt ebenfalls eine CRAY-X-MP. Diese CRAY ist selbstverständlich von überall, zum Beispiel von der TH Bergen aus *interaktiv* benützbar. Wir haben mit unserem Supercomputer end-Use and Security Precedures Plan in Zürich *sehr restriktive Verhältnisse*, die neu geregelt werden sollten. Der Supercomputer in Trondheim wird übrigens nicht von der TH sondern von SINTEF betrieben, einem Forschungsinstitut für industrielle Zusammenarbeit, das seit 1949 sehr eng mit der NTH arbeitet (s. Beilage).

Norwegen ist am sogenannten INTER-NET angeschlossen. Dies ist ein Nachfolger des ARPA Netzes. *Man kann damit durch einen einfachen* telnet *Befehl sich bei irgeneinem an diesem Netz angeschlossenen Rechner einloggen.* Die Worshop-Teilnehmer konnten beispielsweise dadurch leicht ihre mail in San Diego, Argonne, etc. lesen und bearbeiten. Es scheint, dass ein INTER-NET Anschluss nicht davon abhängig ist, ob das Land NATO Mitglied ist. Ich finde, wir sollten uns wenn möglich anschliessen.

Ich möchte anregen, dass die Informatikkommission möglichst bald die folgenden Themen erörtert:

1. Neu Aushandeln des " End-Use and Security Precedures Plan" für die CRAY-X-MP.
2. INTER-NET Anschluss
3. Grossrechner Politik: Ausbau CRAY-X-MP (Y-MP ?), C4 (Uni ?), nationaler HLR-91

Mit freundlichen Grüssen

M. Gander

Kopie an: Prof. C. A. Zehnder

The Internet: Interconnecting Networks

The Internet according to Wikipedia: "…the global system of interconnected computer networks that use **the Internet protocol suite (TCP/IP)** to link billions of devices worldwide."

The Internet was invented specifically for **interconnecting networks**

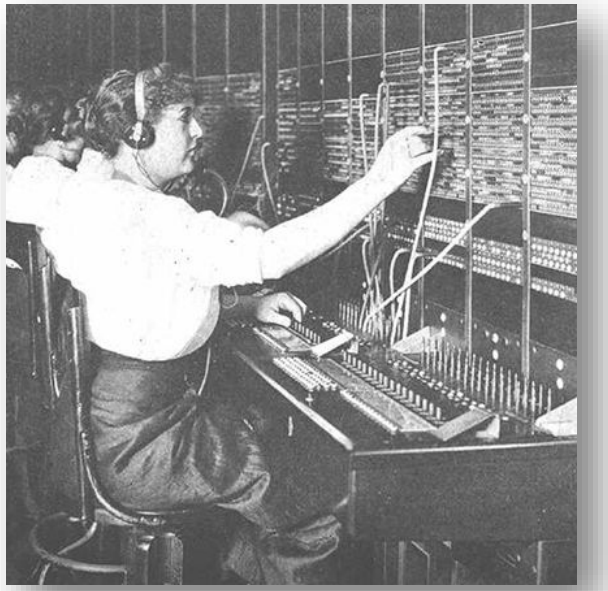
- Started with the ARPANET in the late 1960s
- The Internet then focused on **protocols** and not on networks
- Over time, other protocols faded away in favor of **TCP/IP's packet switching** approach
- The Internet's biggest push was the **advent of the Web** in the early 1990s – the Web made the Internet popular

Internet Protocol: *Who to talk to? How to get there?*

Transmission Control Protocol: *How to talk? How to manage an exchange?*

Both are based on **Packet Switching**

From Circuit Switching to Packet Switching



(Traditional) phone calls are based on **circuit switching**

- **Dedicated** channel between sender and receiver
- **Constant bandwidth** between parties in both directions
- Makes a lot of sense for human interaction!

However: computers **don't talk in the same way as humans do** (non-continuous & bursty)

- Computers **don't talk** most of the time
- But when they talk, they want **immediate** medium access, and they **talk a lot**
- Implication: we **cannot afford a dedicated channel.** It would be **underutilized most of the time!**

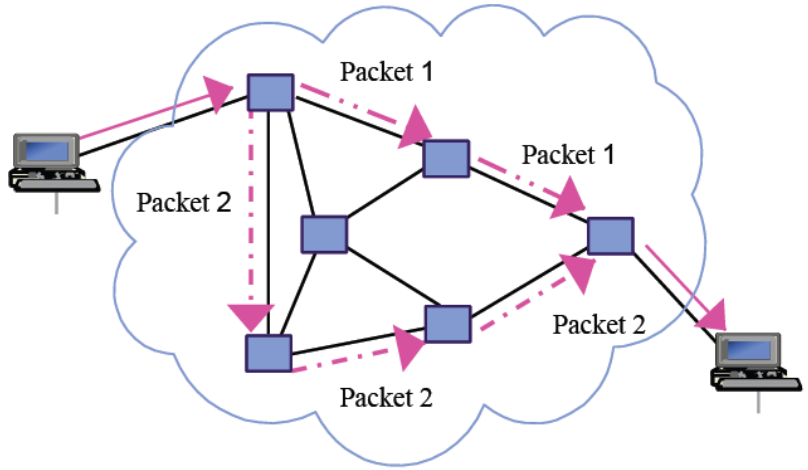
Example: Your phone doesn't do anything. Then **suddenly** it starts an **update download**.

1970s Circuit Switching (AT&T)

- **35 seconds** to set up a call, **minimum charge for 3min of connection (i.e. of dedicated circuit)**
- Desire to send 100 milliseconds of data per connection

TCP's packet switching can thus be regarded as the **evolution of mail** (rather than of the telephone)

Packet Switching



Messages are split into chunks called **packets**

Each packet is sent on its way **independently**

A “connection” on the Internet is an **illusion** of the communicating parties, like a “connection” in the postal mail system. However, correspondences still go back and forth…

Motivation: Much better use of **bandwidth** + much more **robustness**

- Bandwidth: communication by machines vs. communication by people!
- Robustness: enable military communications to survive a nuclear attack!
  - Circuit Switching: connection fails if **any link is disrupted** (reliability **decreases** as the network grows!)
  - Packet Switching: “connection” fails if **all links between two parties are disrupted** (reliability **increases** as the network grows!)

In packet switching, phone wires are just means to an end! In fact, endpoints operate **agnostically** over any number of diverse media, i.e. **any carrier can be used for packet switching**, in principle

(example: Avian Carriers, https://tools.ietf.org/html/rfc1149)

Any Questions / Comments / Doubts / Concerns?

This Week's Menu

Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

Guided Exercise

Briefing: Security
Stock Exchange

Layering: Separation of Concerns

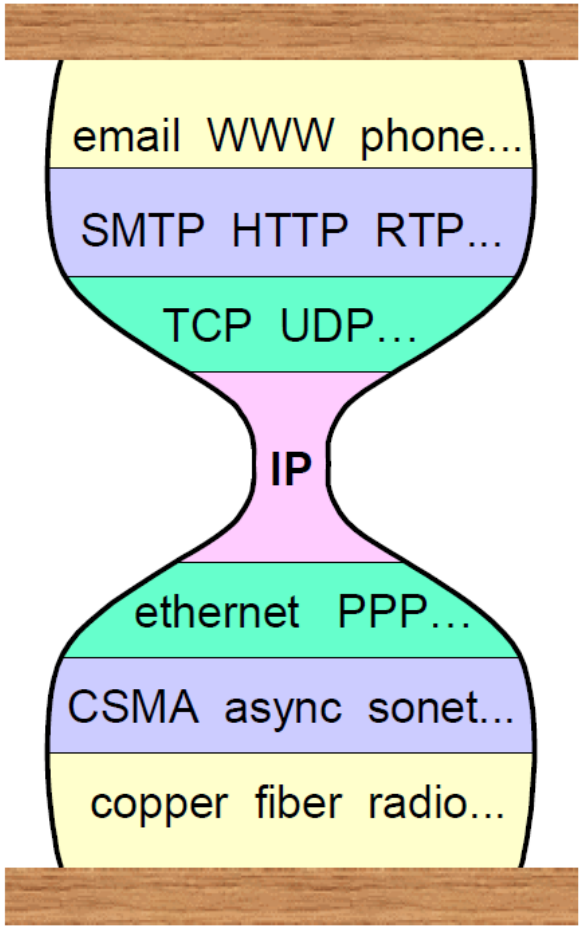
"Networking" circumscribes a large **set of issues**…

- What information should go on the carrier?
- Where does it go?
- How is it secured?
- How do we detect/recover from transmission failures?
- Can we avoid collisions on the carrier?
- What transmission frequencies do we use?

In computer networks, these (and more) issues are tackled by assigning them to **different layers in the networking stack**

Thanks to this so-called "separation of concerns"

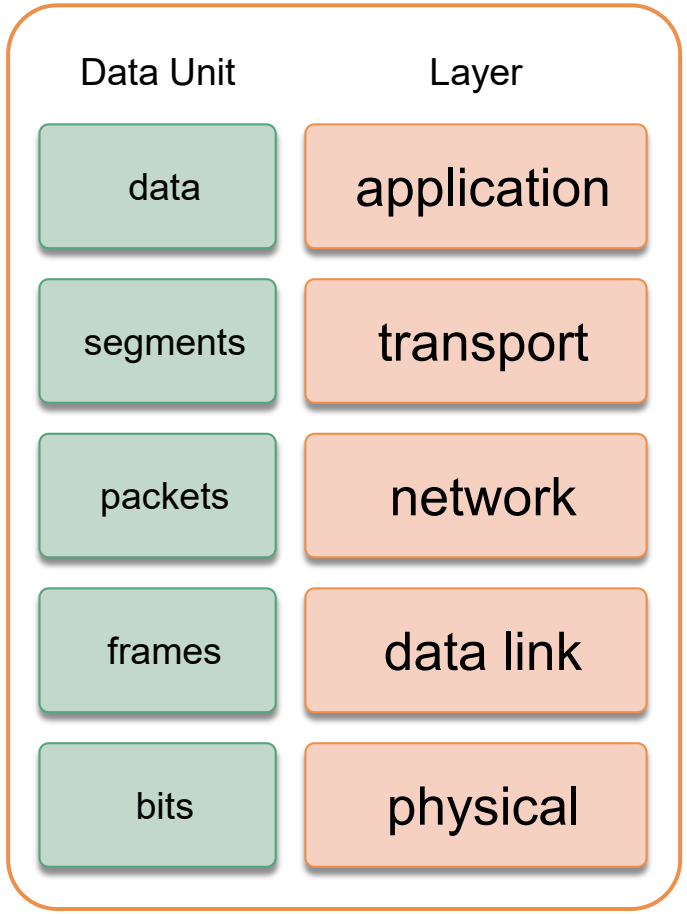
- End-to-end solutions become easier to create
- Parts of solutions may become **reusable**
- Parts of solutions may become **replaceable**
- However, **cross-layer optimization is undermined** by strict SoC



Overview

What is the information **about** and **why** does the interaction happen?

(not part of this course ;-))



What should endpoints do with the payload to get **application-relevant information**?

How do the endpoints of a communication **manage the connection**?

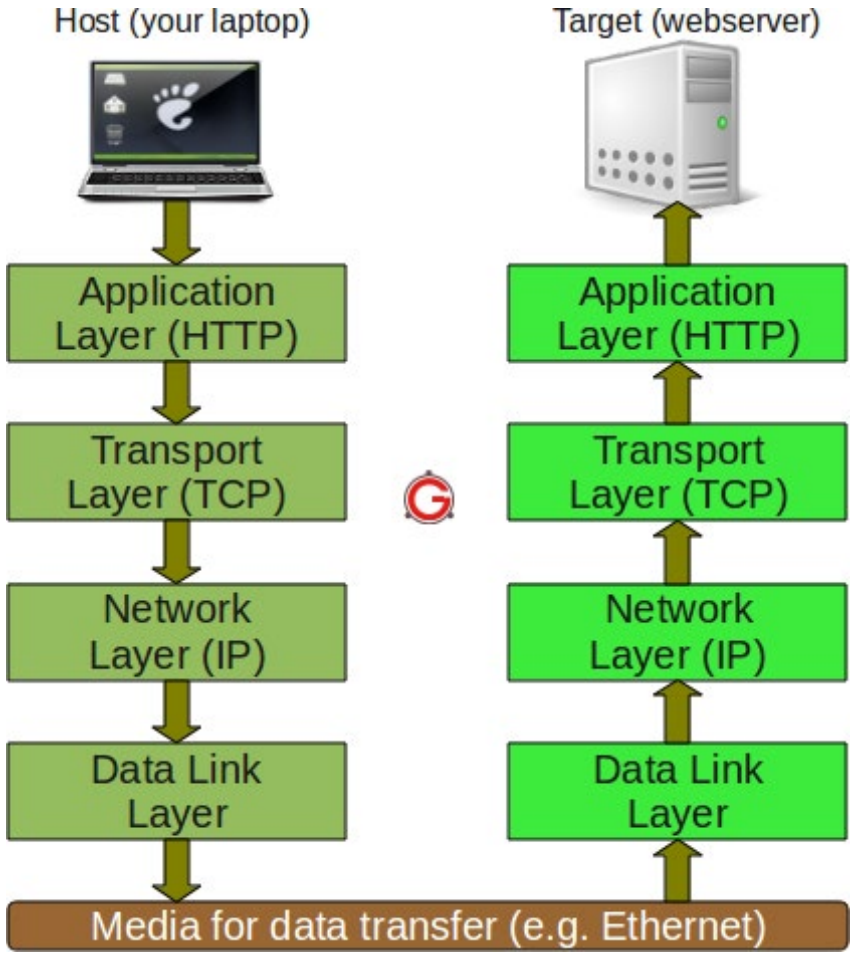
What happens when we leave the immediately connected carrier? **Where** does our data go?

How do multiple senders/receivers **manage the (shared) carrier**?

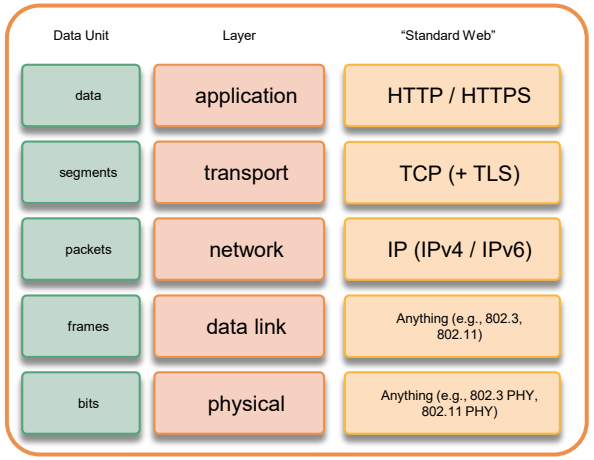
What's our **carrier** and how do we **transmit data** over it?

Bits, Frames, Packets, Segments

Data Unit	Layer
data	application
segments	transport
packets	network
frames	data link
bits	physical



Bits, Frames, Packets, Segments



Let's have a look at a **network monitor**!

Bits, Frames, Packets, Segments

You computer accessed a Web resource on a remote server. This is the **server's response**.

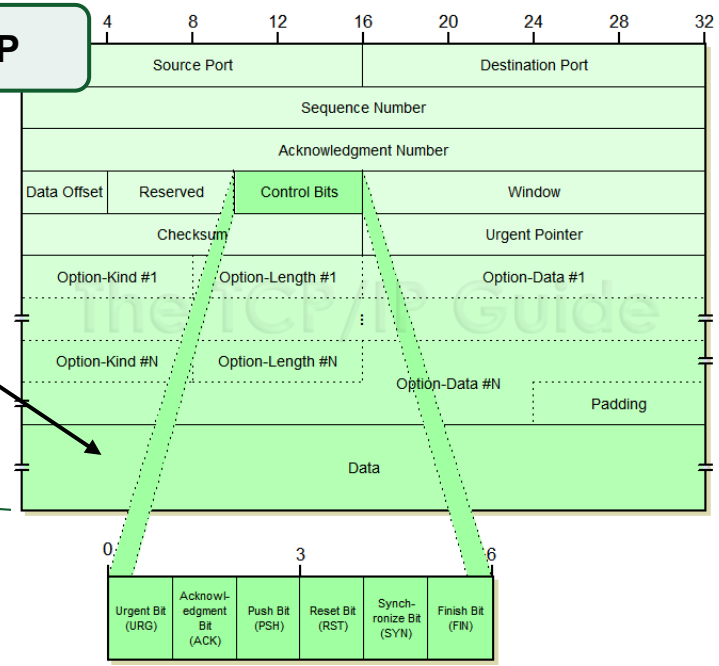
Data Unit	Layer	"Standard Web"
data	application	HTTP / HTTPS
segments	transport	TCP (+ TLS)
packets	network	IP (IPv4 / IPv6)
frames	data link	Anything (e.g., 802.3, 802.11)
bits	physical	Anything (e.g., 802.3 PHY, 802.11 PHY)

Application payload from a Webserver, e.g. HTML: `<html><body><h1>Hello, World</h1></body></html>`

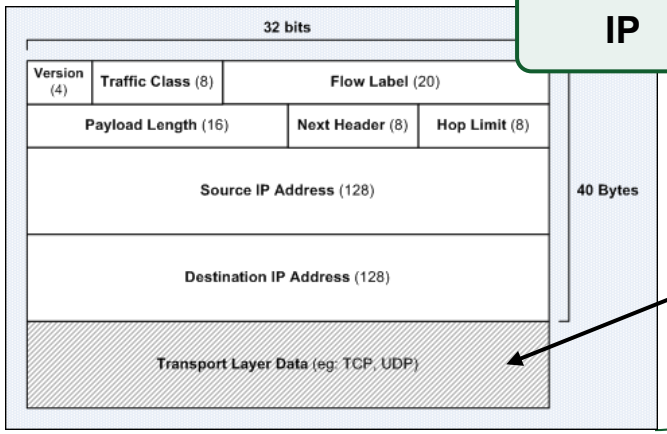
HTTP

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
 <body>
  <h1>Hello World!</h1>
 </body>
</html>
```

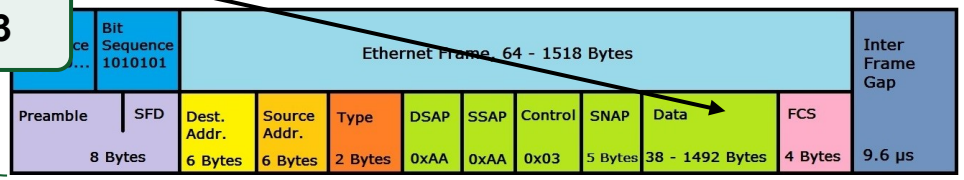
TCP



IP



802.3



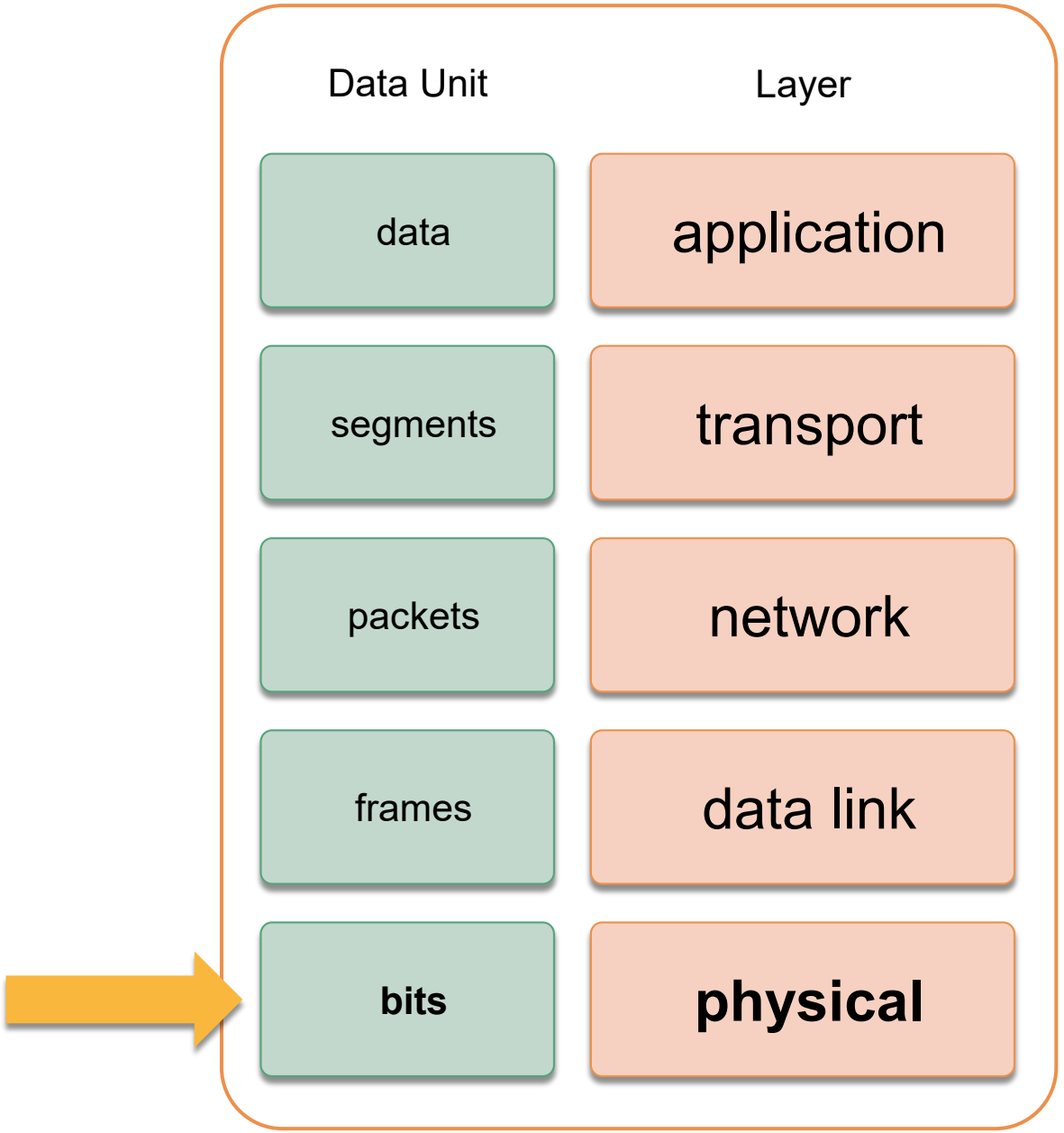
1000Base-SX

"1000BASE-SX is a fiber optic Gigabit Ethernet standard for operation over multi-mode fiber using a 770 to 860 nanometer, near infrared light wavelength."

This Week's Menu

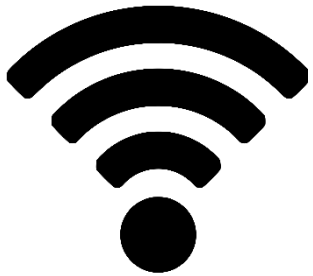
Lecture

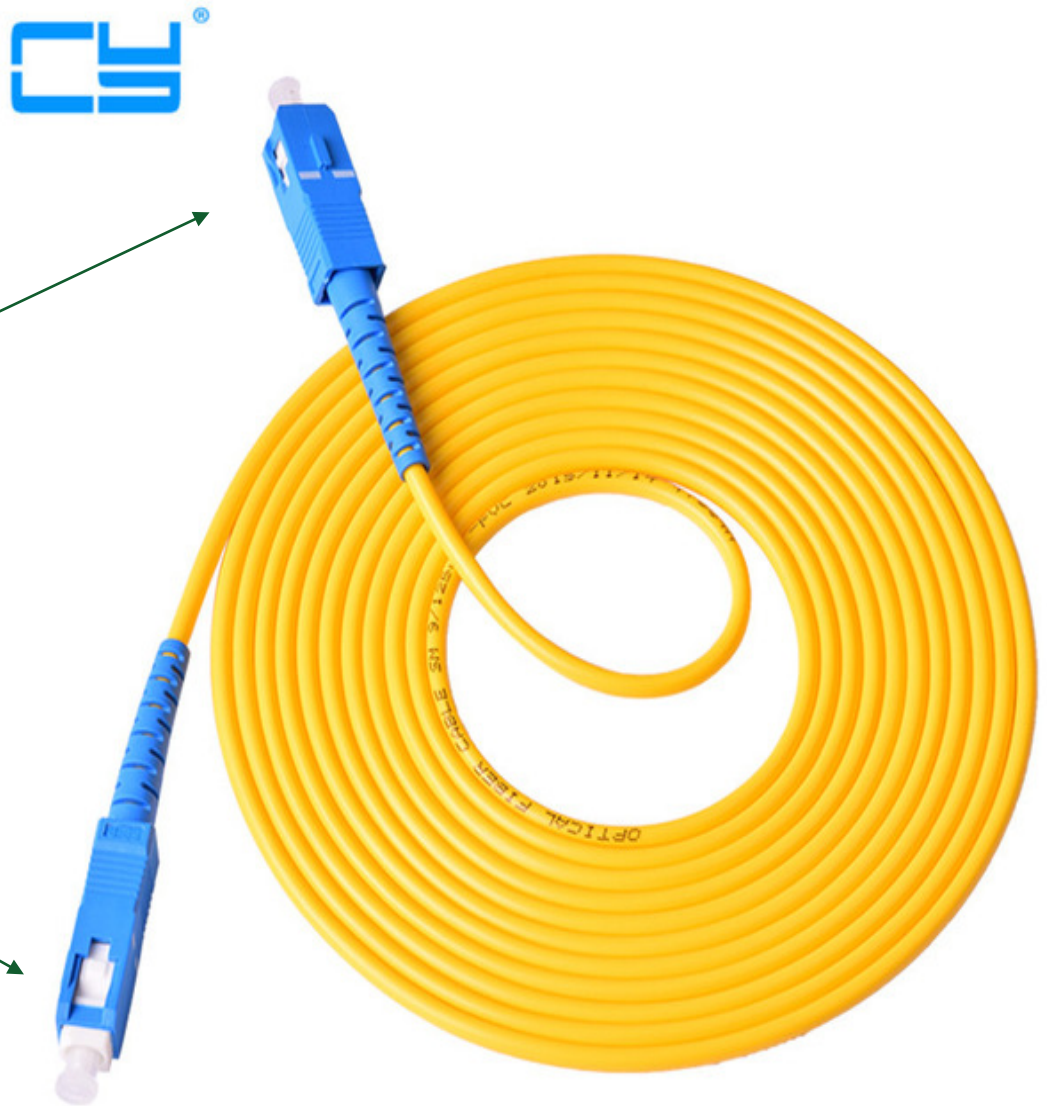
Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web



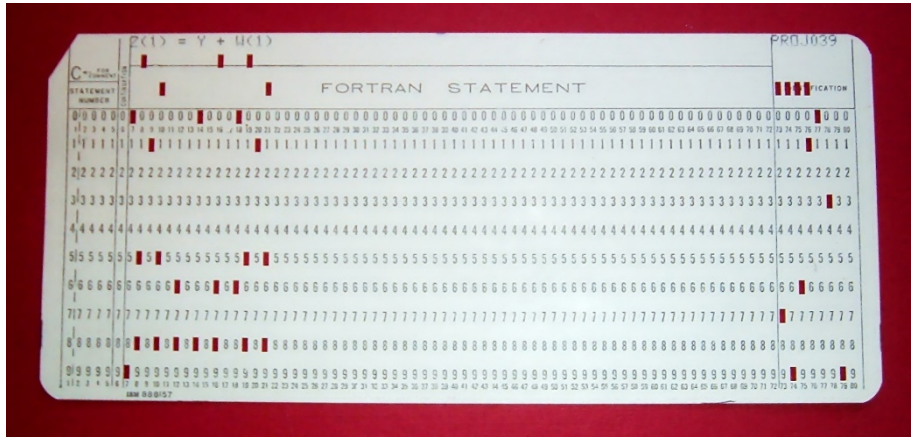
Our Goal

Get **one signal**
from here to here





Remember: Bits?



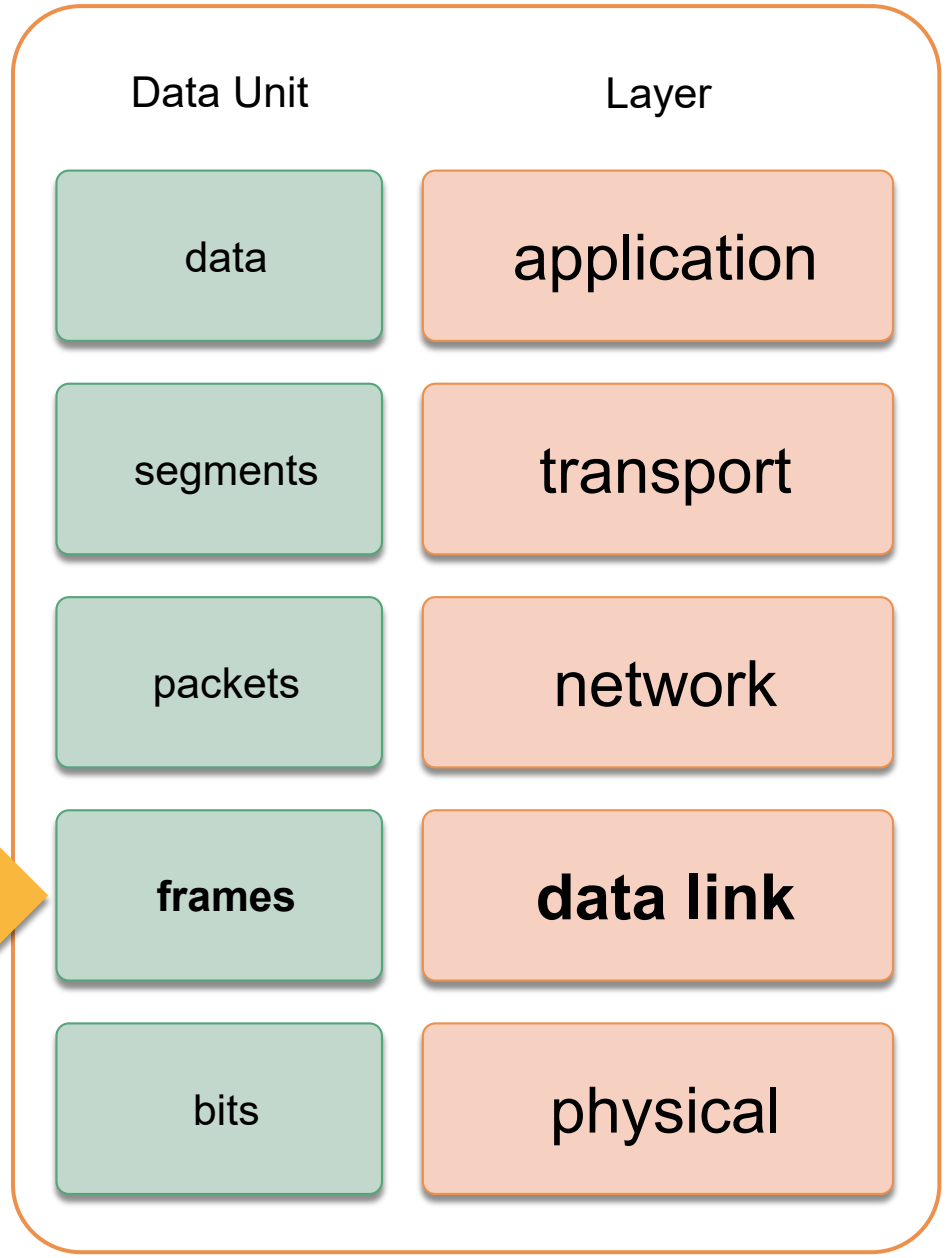
Why ***binary*** digits?

- Basic representation of the **logical values** *True* and *False*
- Many easily feasible **physical representations**:
  - Punched cards: Hole vs. No Hole
  - Electrical switches: On vs. Off
  - Random Access Memory: High Electrical Charge vs. Low Electrical Charge
  - Computer Hard Disks: Polarity of Magnetization
  - Solid State Drive: High Electrical Charge vs. Low Electrical Charge
  - **Ethernet Cables:** **High Voltage vs. Low Voltage**
  - **Fibre Optics:** **Light vs. No Light**
  - **Wireless Transmission:** **High Frequency vs. Low Frequency**
  - Credit Cards: Polarity of Magnetization
  - Bar Codes: Thick Line vs. Narrow Line
  - Station Clocks: Light vs. No Light

This Week's Menu

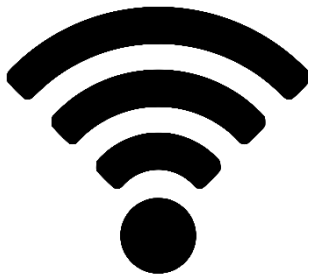
Lecture

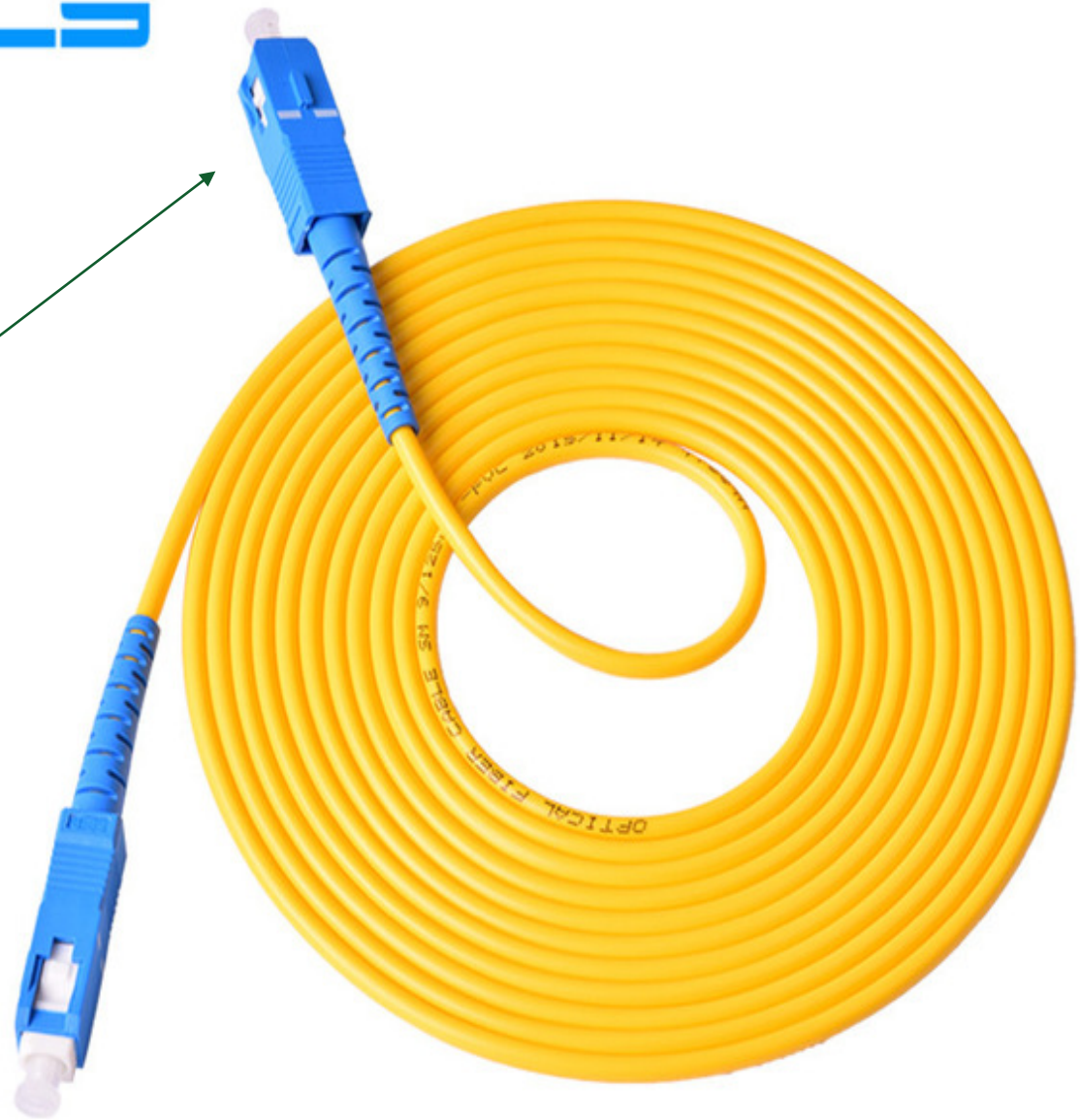
Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web



Our Goal

Get **multiple signals** from here
to here at the same time





First, we need to **identify** devices!

MAC Addresses are **unique identifiers** that are **statically assigned** to device network interfaces.
Thus, they are "**Burned-in Addresses**"

```
Example: 00:1B:44:11:3A:B7
```

What's the problem with statically assigned unique identifiers?

- According to E. Snowden, the US NSA **tracks movements of people by monitoring MAC addresses**
- MAC addresses are randomized during network scanning by current versions of iOS, Android, Windows, Linux
- However, this might not be enough…

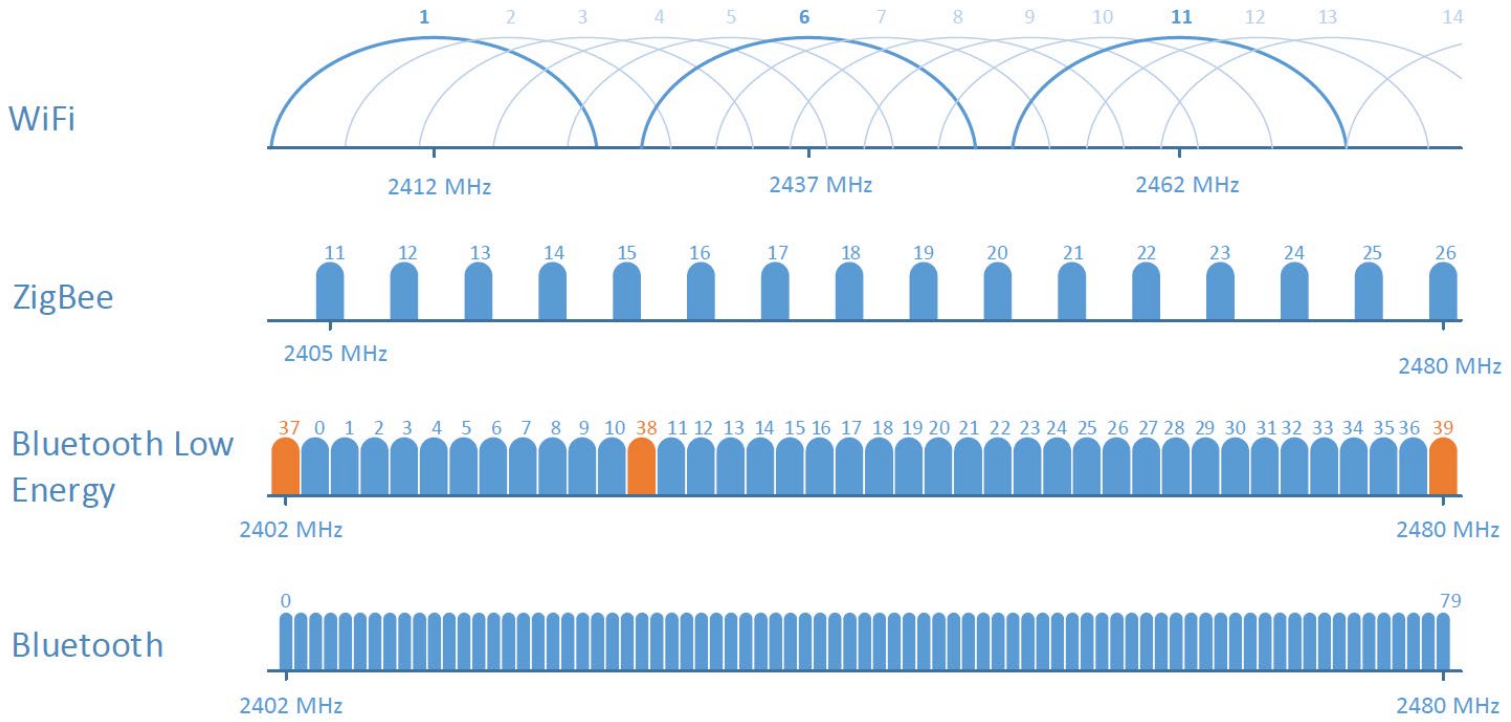
https://hal.archives-ouvertes.fr/tel-01659783/

Let's see what equipment the university uses.
(`tracert www.google.com; arp -a`)

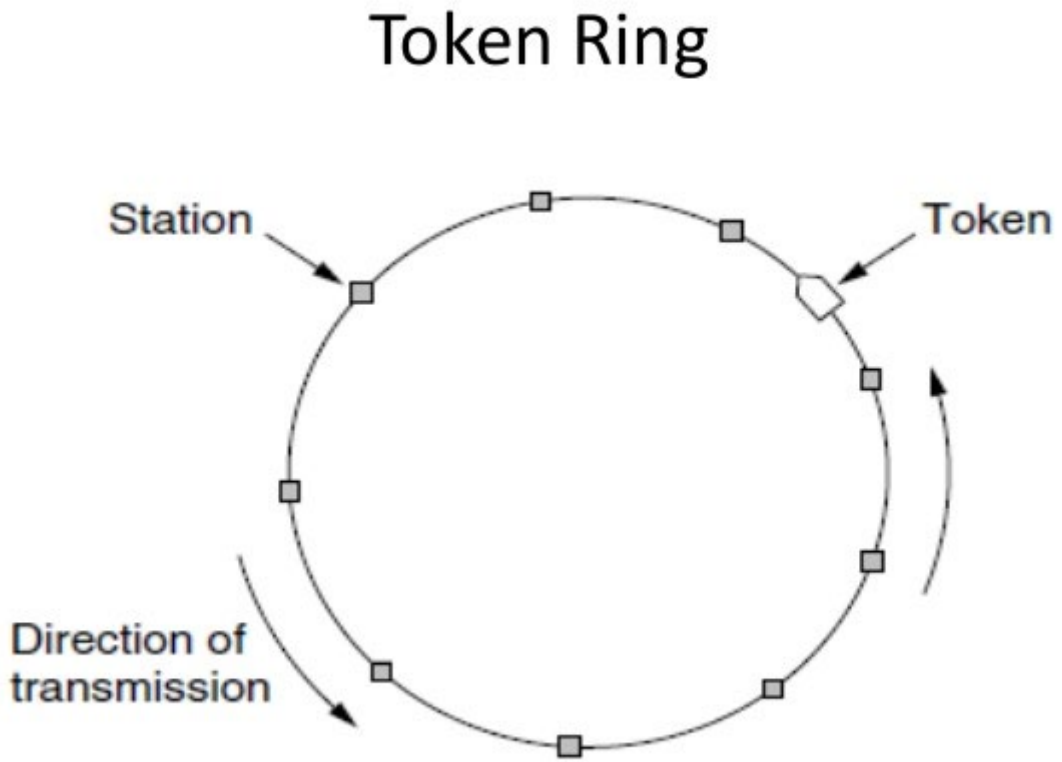
Now, what if they **talk at the same time**?

Devices that are connected to a cable (or that share a wireless carrier) **need to coordinate** when they talk! Otherwise, there would be too much crosstalk, or «**interference**»

Illustration: https://www.youtube.com/watch?v=tIbF97kpoIg



Example: Token Ring



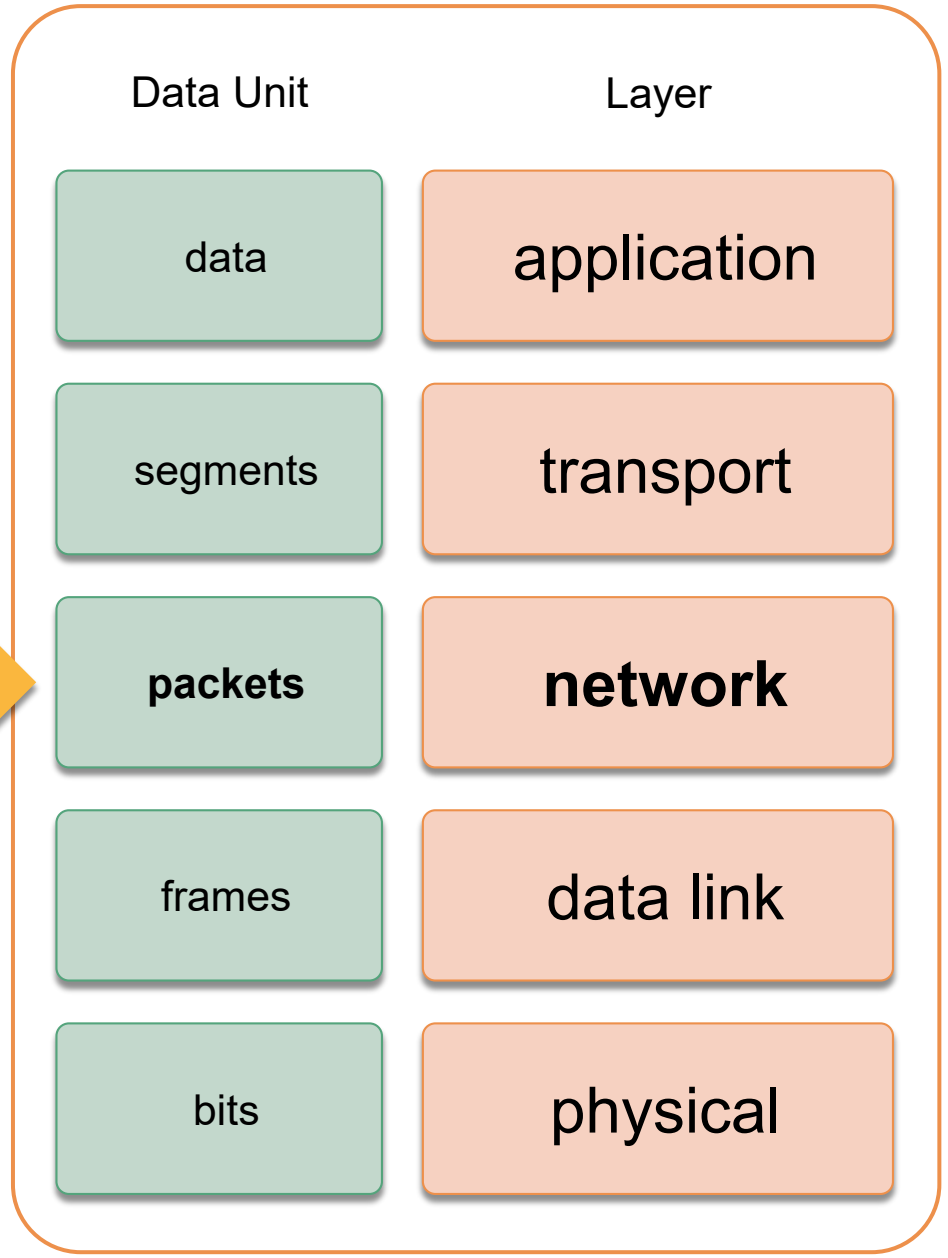
https://www.slideshare.net/hemangkothari/lecture-5-collision-free-protocols

Any Questions / Comments / Doubts / Concerns?

This Week's Menu

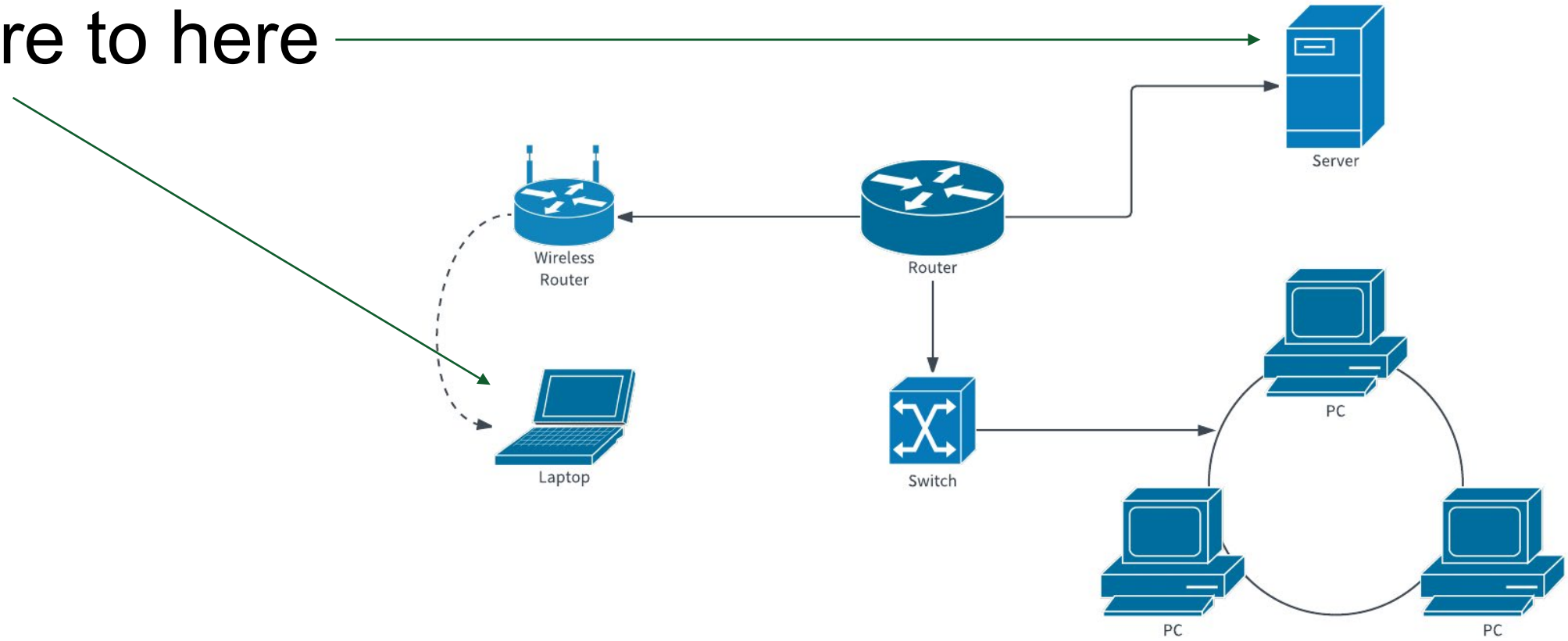
Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

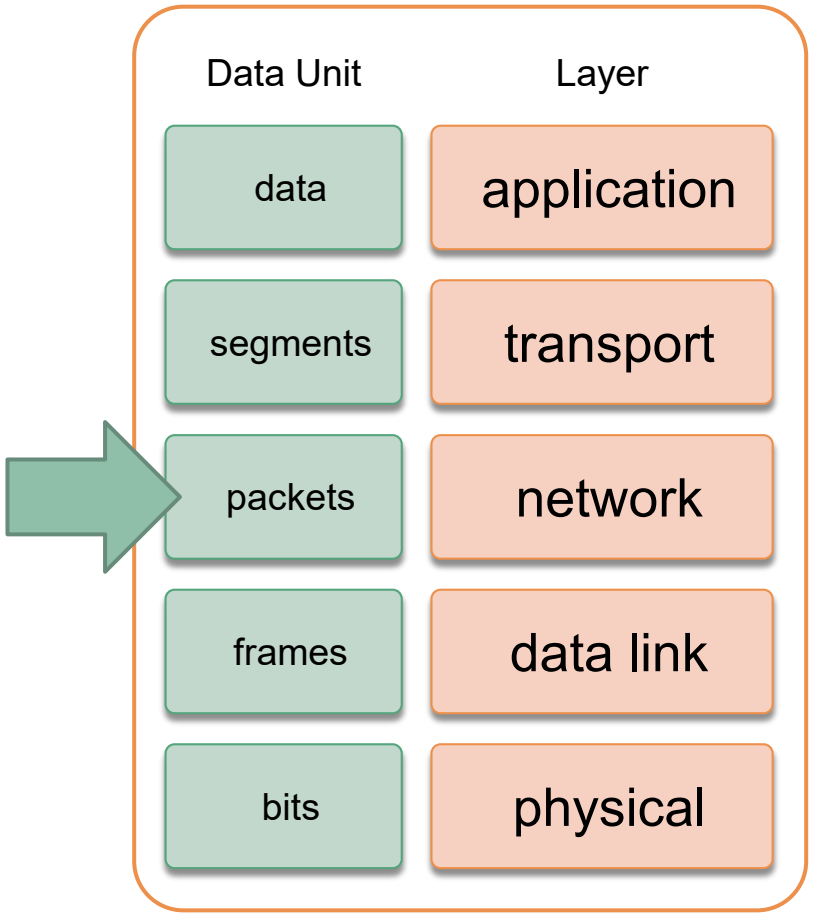


Our Goal

Get **frames** from here to here



The Networking Stack: Network



Allows **addressing** at the network level

Allows **packet routing** across networks

For the Internet (and almost all other networked systems), this means **IP addresses**

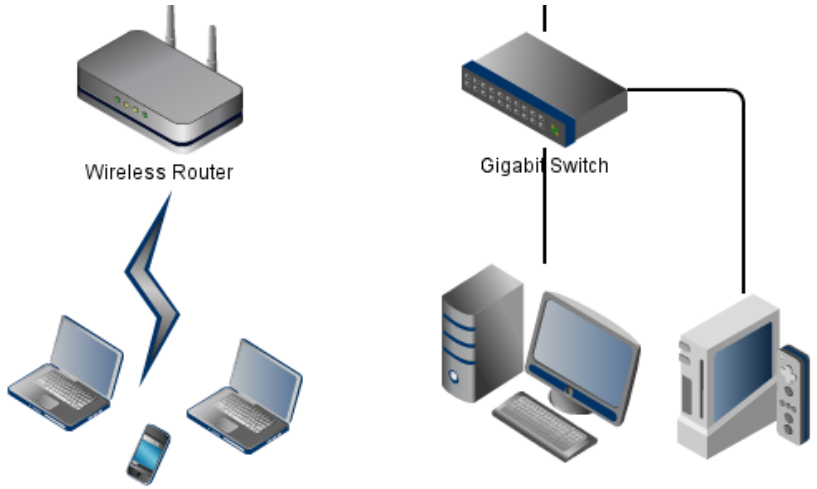
- IPv4 is the version that has powered the Internet so far
- IPv6 is phased in to be able to handle IoT-scale growth

(Note that packets might be dropped or duplicated)

The Internet Protocol

Purpose of IP: Relay frames **across network boundaries**

IP routing enables **internetworking**, and thereby **establishes the Internet**. The **routing** of IP packets to their destination is solely based on the IP address of the destination.



Demo (Networking Infrastructure and Traceroute)

Commands: traceroute (Unix), tracert (Win)

Example: tracert www.google.com

https://www.submarinecablemap.com/

https://askubuntu.com/questions/21673/hybrid-wireless-network-repeating

IP Addresses

32-bit IPv4 Addresses -> 2^32 = 4'294'967'296 host addresses

128-bit IPv6 Addresses -> 2^128 = 340'282'366'920'938'000'000'000'000'000'000'000 host addresses (i.e. we can address all atoms in the known universe)

Numerical labels that are assigned to **connected** devices

Dynamically assigned (in most cases), e.g. via a protocol called DHCP

IPv4 Addresses: 32 bits

- Typical IPv4 Local Area Network address: `192.168.0.1`
- Managed by the Internet Assigned Numbers Authority (IANA) and five regional Internet registries (RIRs)
  - How many addresses do we have in this space?
  - **Top-level exhaustion** happened on 31 January 2011, and all the five regional Internet registries (RIRs) have exhausted all non-reserved address blocks

IPv6 Addresses: 128 bits

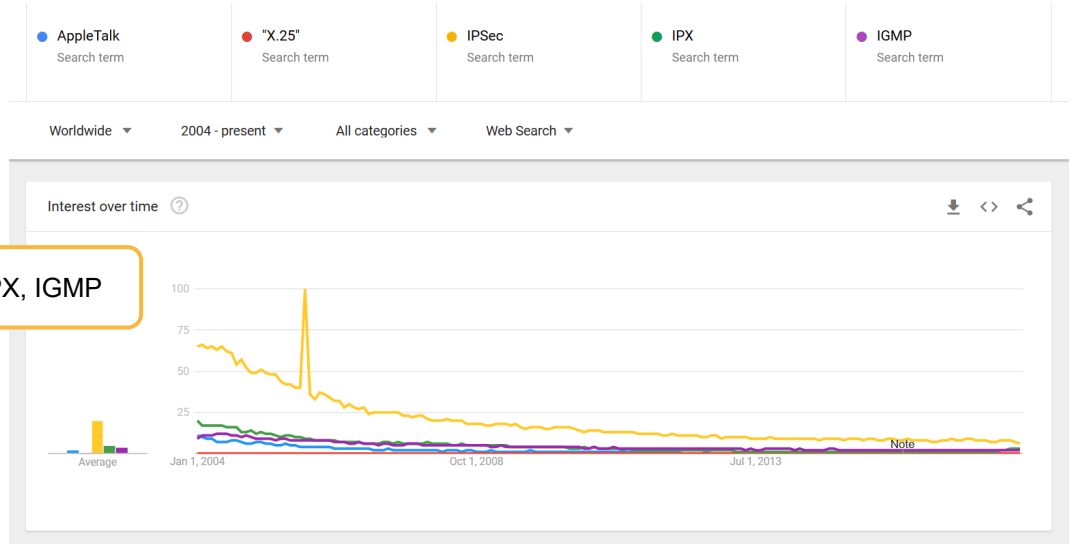
- Typical IPv6 address: `2001:0db8:ac10:fe01:0000:0000:0000:0000`
- These are the **future** (and presence, at about 50% penetration) of networking

April Fool's Joke 1994: https://tools.ietf.org/html/rfc1606

The Internet and its Protocols

ICMP: Not for data exchange…
IPsec: Security layer on top of IP
IGMP: *Replaced* by IPv6 Group Management
Xerox IPX: *Replaced* by IP (1998)
AppleTalk: *Replaced* by IP (1990-2009) [-> Bonjour]
ITU-T X.25 PLP: *Replaced* by IP (~2012; still used in niche applications)

3. Network layer [hide]
IP (IPv4 · IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk · X.25 PLP



Google Interest over time in AppleTalk, X.25, IPSec, IPX, IGMP

OSI model
by layer

7. Application layer [hide]
NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · DHCP · Netconf · *more....*

6. Presentation layer [hide]
MIME · XDR

5. Session layer [hide]
Named pipe · NetBIOS · SAP · PPTP · RTP · SOCKS · SPDY

4. Transport layer [hide]
TCP · UDP · SCTP · DCCP · SPX

3. Network layer [hide]
IP (IPv4 · IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk · X.25 PLP

2. Data link layer [hide]
ATM · ARP · IS-IS · SDLC · HDLC · CSLIP · SLIP · GFP · PLIP · IEEE 802.2 · LLC · MAC · L2TP · IEEE 802.3 · Frame Relay · ITU-T G.hn DLL · PPP · X.25 LAPB · Q.921 LAPD · Q.922 LAPF

1. Physical layer [hide]
EIA/TIA-232 · EIA/TIA-449 · ITU-T V-Series · I.430 · I.431 · PDH · SONET/SDH · PON · OTN · DSL · IEEE 802.3 · IEEE 802.11 · IEEE 802.15 · IEEE 802.16 · IEEE 1394 · ITU-T G.hn PHY · USB · Bluetooth · RS-232 · RS-449

V · T · E

IP: The Narrow Waist of Networking

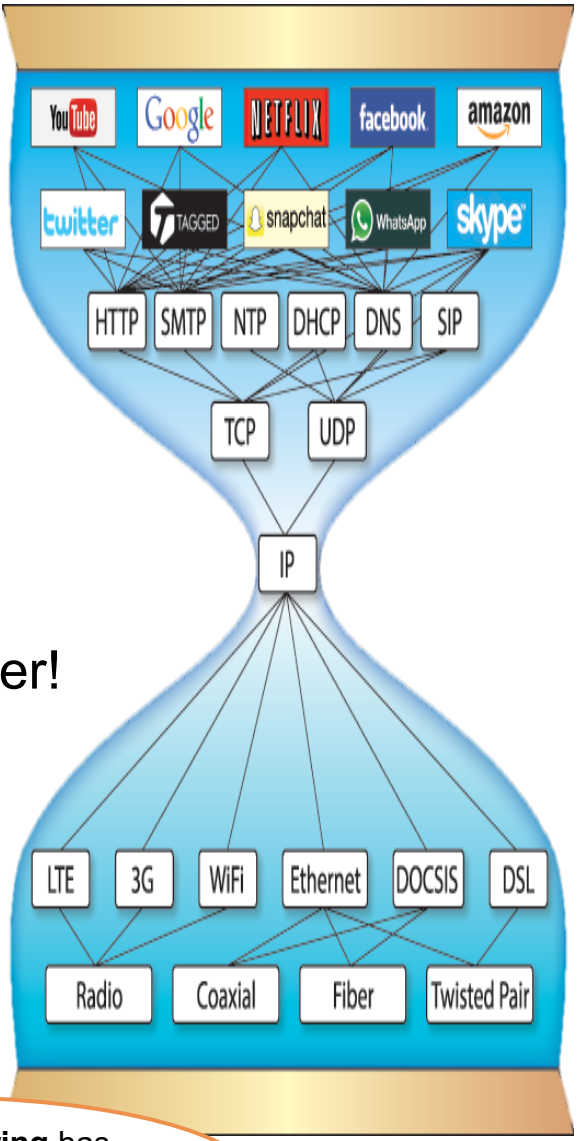
ICMP: Not for data exchange…
IPsec: Security layer on top of IP
IGMP: *Replaced* by IPv6 Group Management
Xerox IPX: *Replaced* by IP (1998)
AppleTalk: *Replaced* by IP (1990-2009) [-> Bonjour]
ITU-T X.25 PLP: *Replaced* by IP (~2012; still used in niche applications)

"*IP over everything, everything over IP*" is the **biggest network effect** ever!

- Fundamental goal: interconnect **heterogeneous** systems
- Fundamental requirement: **global** addressing scheme

Drawback?

- Adding functionality in the network layer is extremely difficult!
- This is exemplified in the **tedious decade-long process** of switching from IPv4 to IPv6



In general, **layering** has worked out very well for computer networks…

A Bridge Built to Sway When the Earth Shakes

The new eastern span of the Bay Bridge in San Francisco. The existing eastern bridge, far left, will be torn down after the new bridge is complete. Mika O. Grondahl/The New York Times

By Henry Fountain

Feb. 6, 2012

SAN FRANCISCO — Venture deep inside the new skyway of the San Francisco-Oakland Bay Bridge, and it becomes clear that the bridge's engineers have planned for the long term.

https://www.nytimes.com/2012/02/07/science/to-survive-a-quake-new-bay-bridge-span-will-offer-least-resistance.html

This is effectively what we are doing today with IPv4 / IPv6 Dual-Stack Configurations

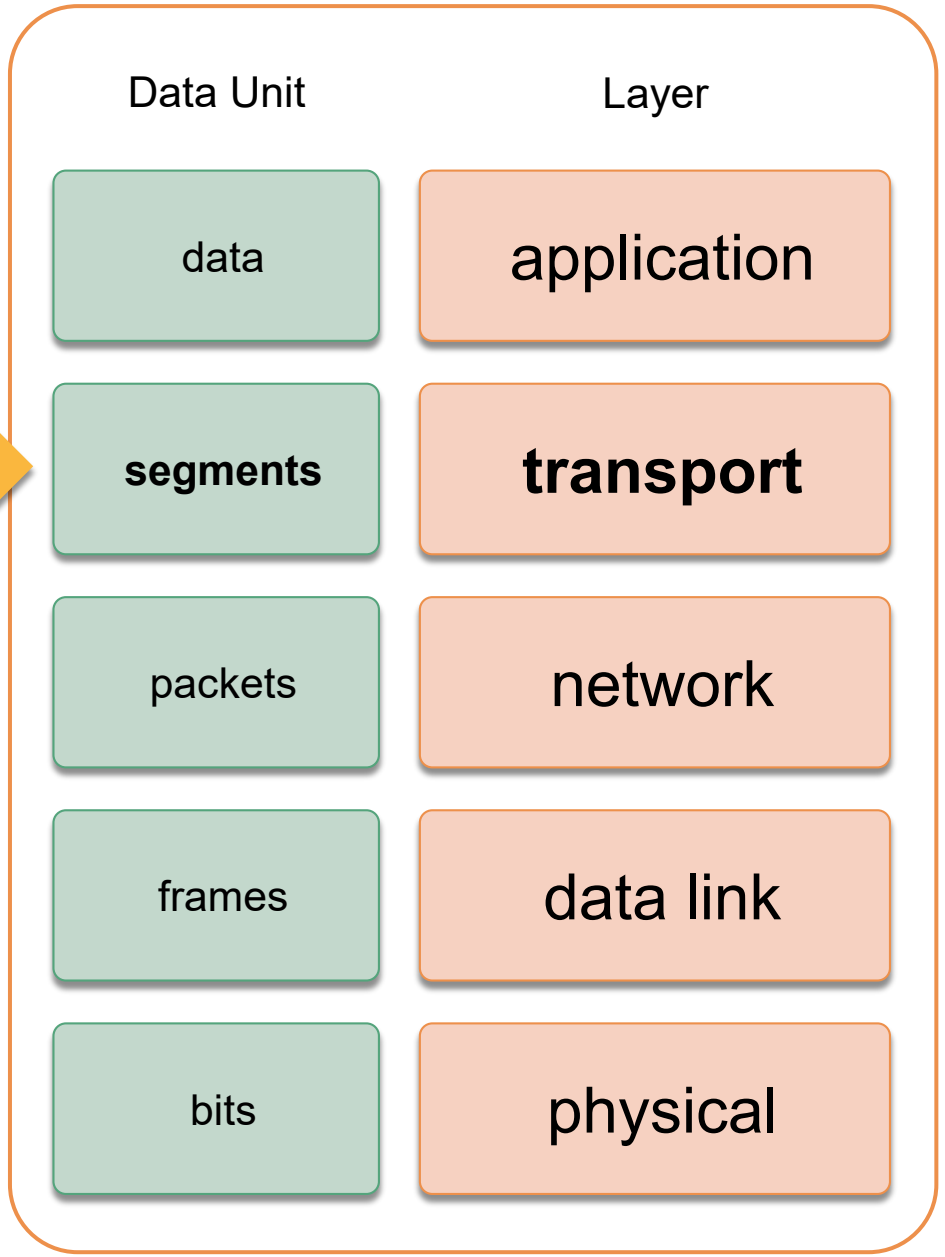
https://en.wikipedia.org/wiki/IPv6#Dual-stack_IP_implementation

Any Questions / Comments / Doubts / Concerns?

This Week's Menu

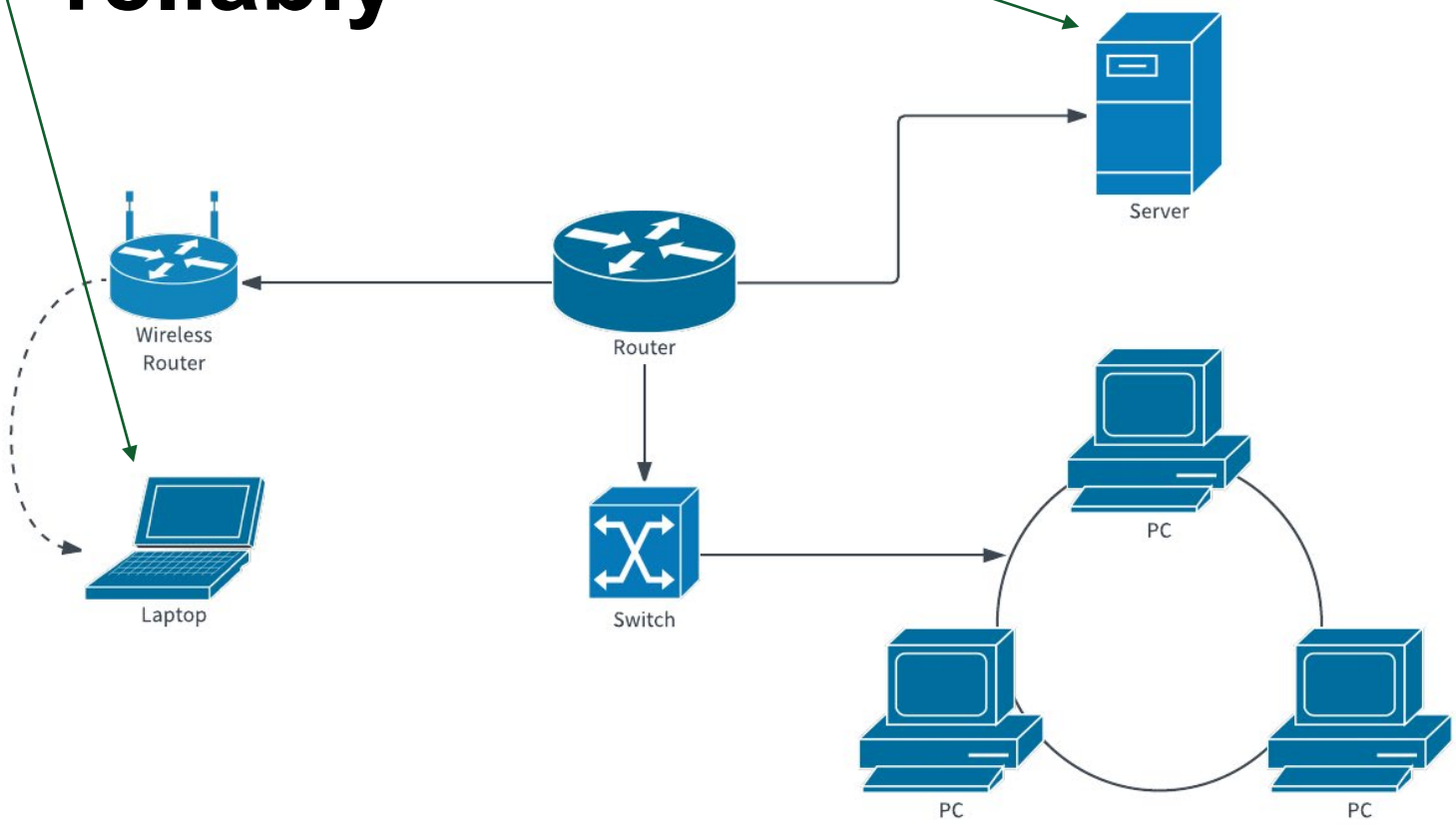
Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web



Our Goal

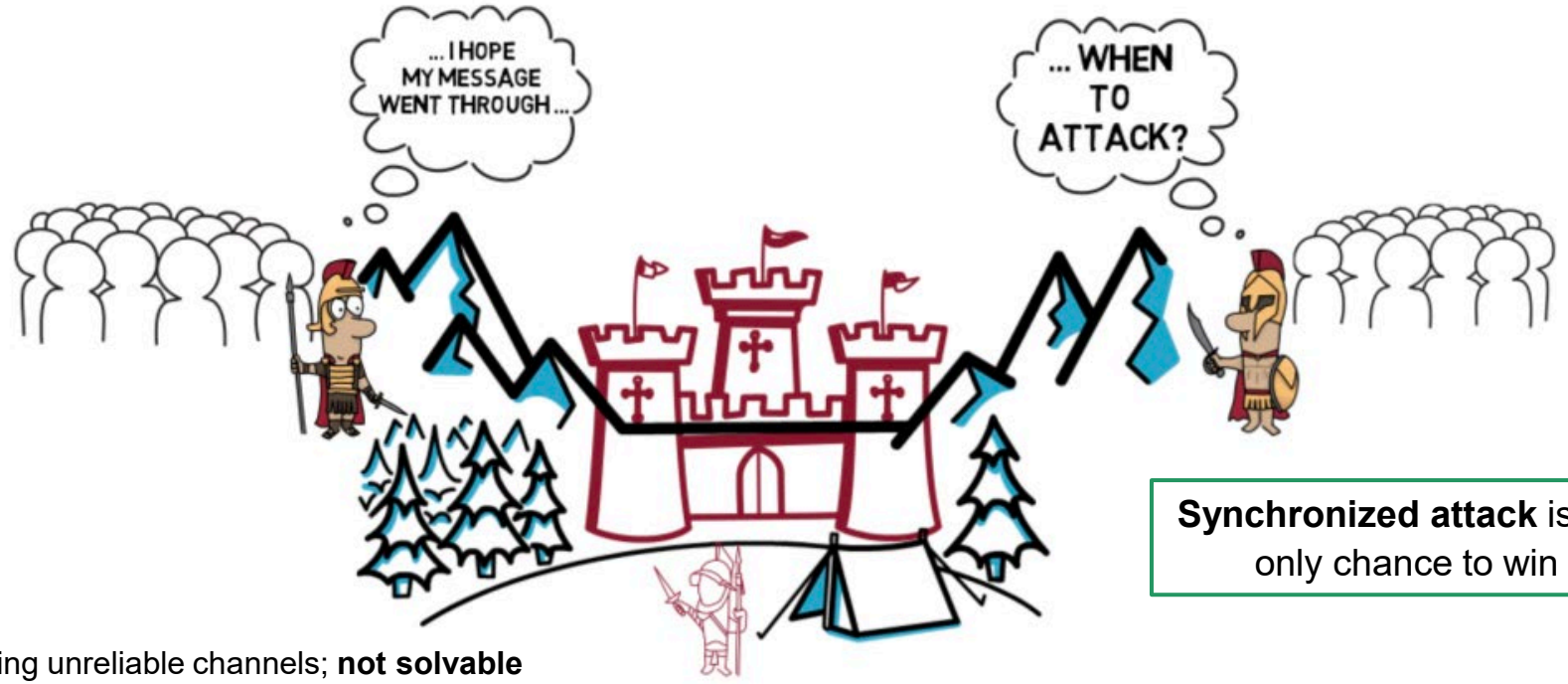
Get packets from here to here, **reliably**



Sending Packets over Unreliable Channels

Simple (?) Question: **How can two parties be sure that they have reached agreement over an unreliable communications link?**

The channel is **unreliable**, i.e. messages **might get lost**.
Messages **cannot be read or tampered with** though (let's say they are "encrypted")



Synchronized attack is the only chance to win

Classic computer science problem regarding unreliable channels; **not solvable**

https://www.youtube.com/watch?time_continue=31&v=s8Wbt0b8bwY

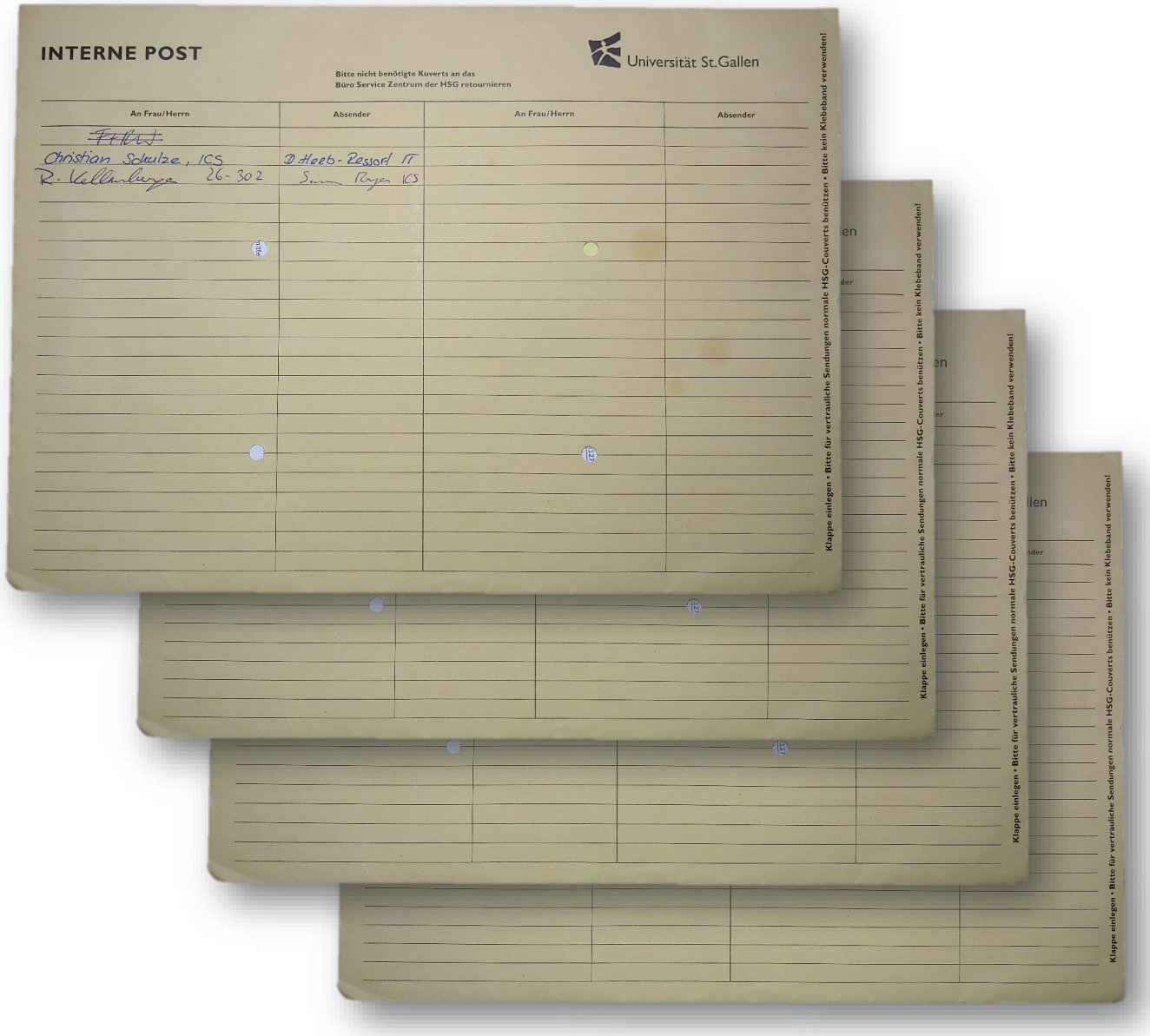
How to add enough reliability?

We send four envelopes on their way separately.

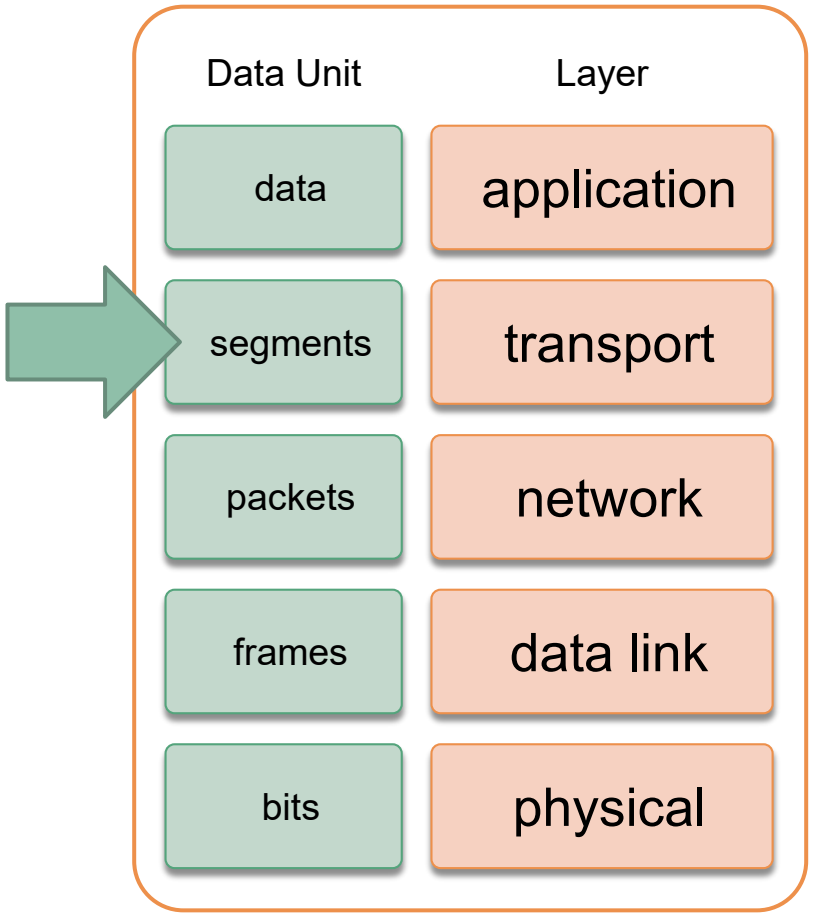
To **increase the reliability of the channel**, we could put **sequence numbers** on the envelopes and the receiver could re-request all missing envelopes.

Sequence numbers also help with **in-order delivery**!

And if an envelope is **received twice**, the receiver can simply **discard** it.



The Networking Stack: Reliability



TCP: More reliable connections (but more overhead)

- Flow-controlled (avoiding congestion)
- No data lost or duplicated (using serial numbers in data packets)

```
# Python: create a socket for TCP over IP
myTCPSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

UDP: Unreliable connections (but less overhead)

- No connection setup phase -> start transmitting data immediately
- Lightweight packet format -> little overhead
- Configurable checksum-based error correction -> little overhead
- Protocol of choice for loss-tolerant settings (e.g., videoconferencing)
- Interesting for IoT settings with resource-constrained nodes…

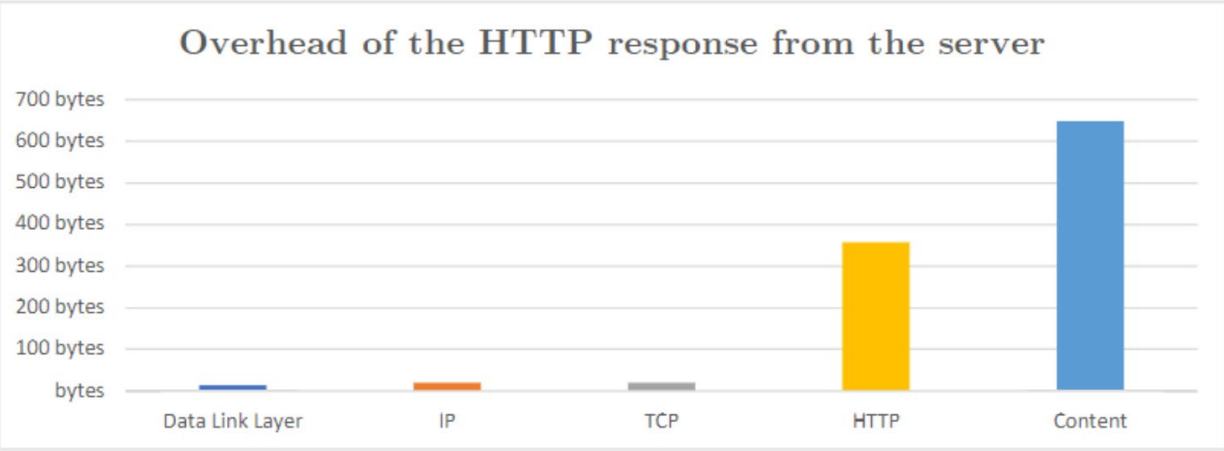
```
# Python: create a socket for UDP over IP
myUDPSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

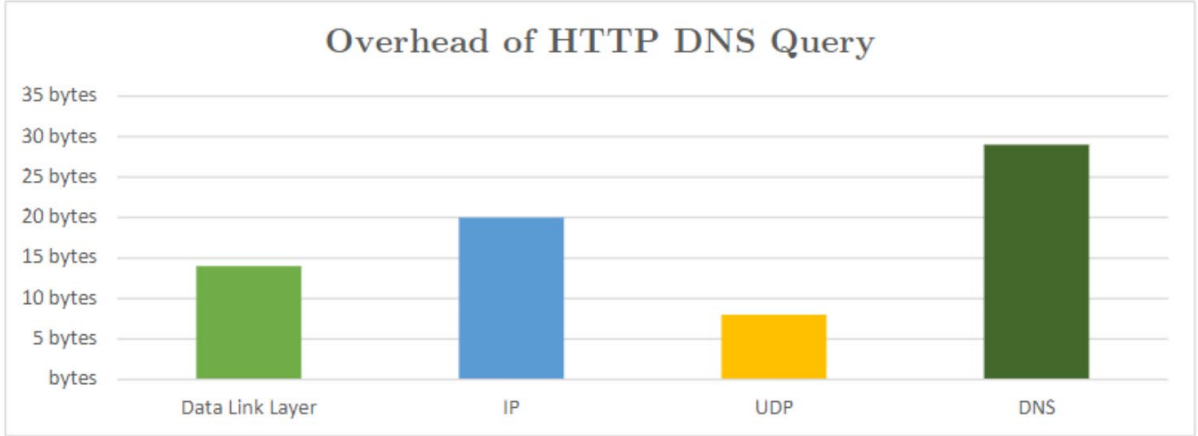
Protocol Overhead…

	Amount of bytes used	Ratio of Protocol Overhead to payload
Data Link Layer	14 bytes	7:324
IP	20 bytes	5:162
TCP	20 bytes	
HTTP	357 bytes	119:216
Content	648 bytes	1:1
Total	1,059 bytes	

150% overhead…

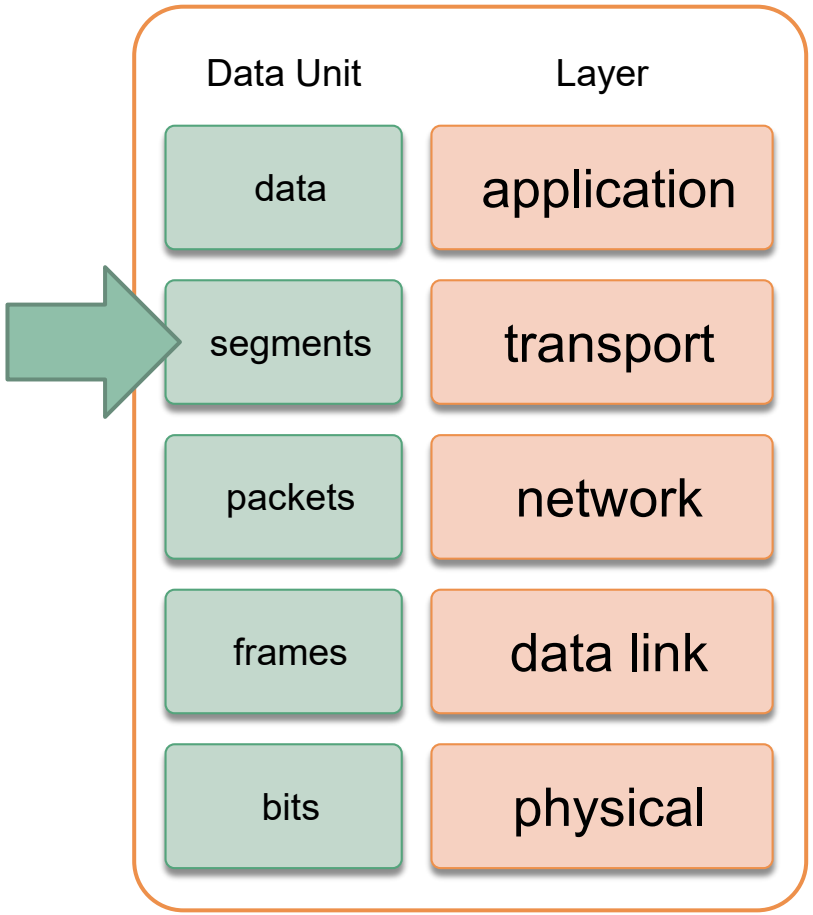
	Amount of bytes used	Ratio of Protocol Overhead to payload
Data Link Layer	14 bytes	14:29
IP	20 bytes	20:29
UDP	8 bytes	8:29
DNS	29 bytes	1:1
Total	71 bytes	





Source: ICSN Students, HS20

The Networking Stack: Ports



The network layer establishes connectivity on a **machine-to-machine** level, typically using IP addresses.

The transport layer (both TCP and UDP) establishes connectivity **between processes.** To do this, it uses **ports.**

Analogy: An **IP address** identifies a **house** (i.e., the entire machine) while the **port number** identifies **individual rooms** (i.e., processes on this machine)

(Note that the application doesn't know what to do with received segments)

TCP and UDP Jokes

Out-of-order delivery

"Hi, would you like to hear a UDP joke?."

"Yes, I would like to hear a UDP joke."

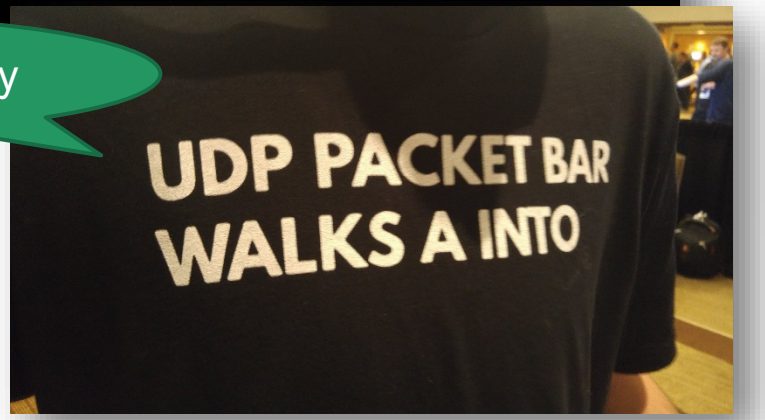
"To get to the other side."

...

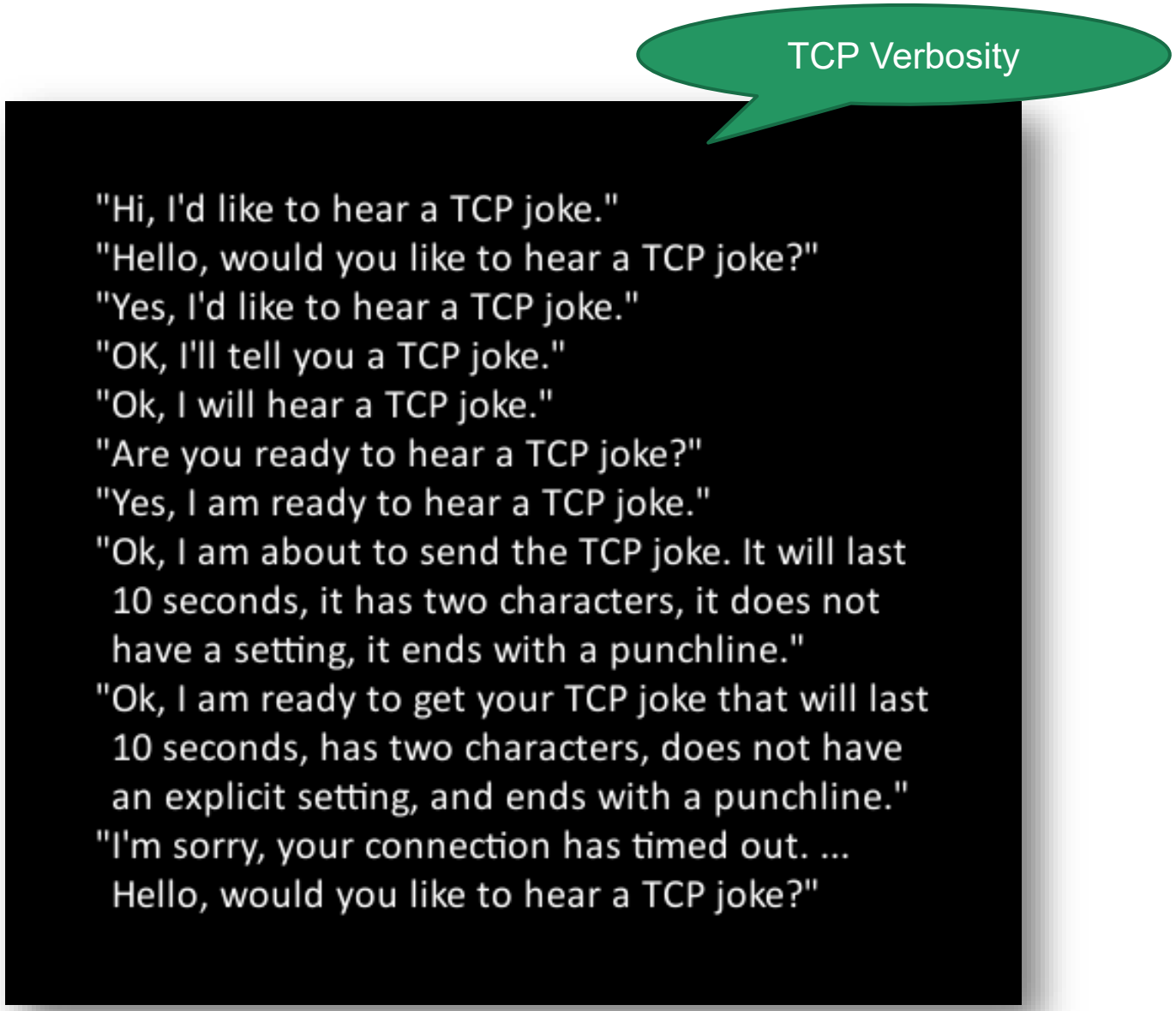
"Why did the chicken cross the road?"

Unreliable delivery

Out-of-order delivery



TCP and UDP Jokes

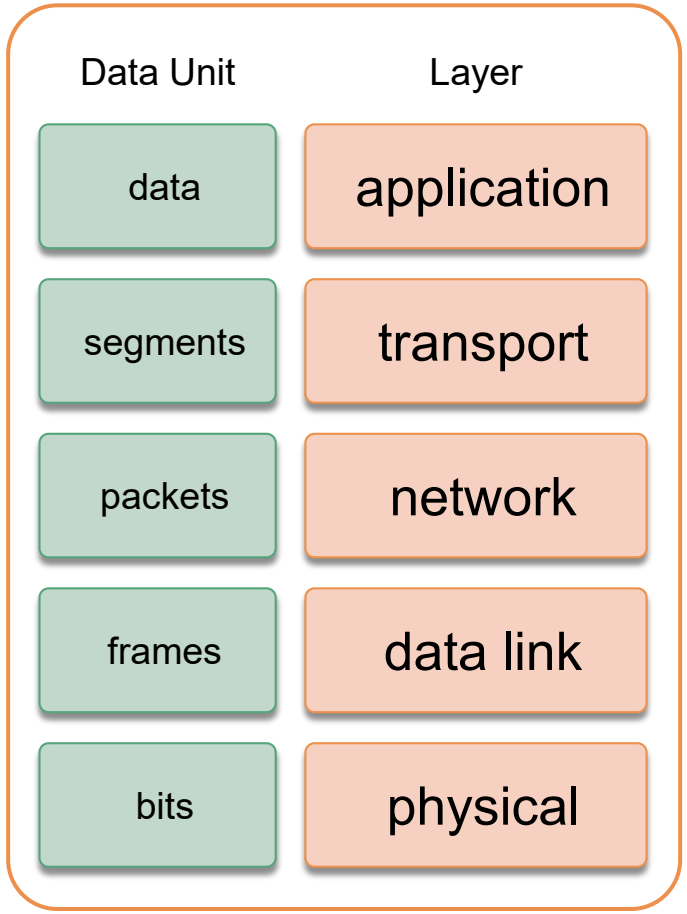


Any Questions / Comments / Doubts / Concerns?

Overview

What is the information **about** and **why** does the interaction happen?

(not part of this course ;-))



What should endpoints do with the payload to get **application-relevant information**?

How do the endpoints of a communication **manage the connection**?

What happens when we leave the immediately connected carrier? **Where** does our data go?

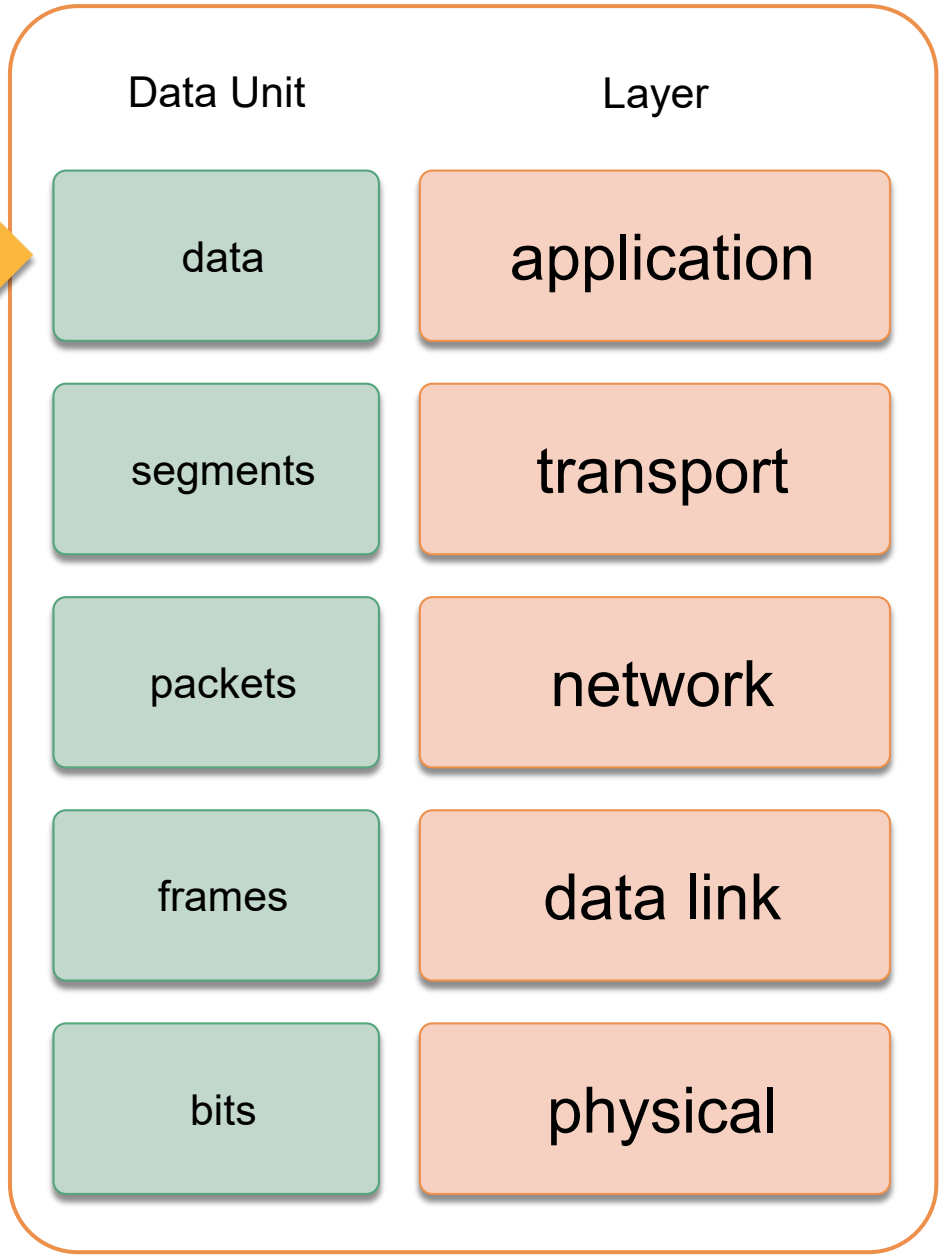
How do multiple senders/receivers **manage the (shared) carrier**?

What's our **carrier** and how do we **transmit data** over it?

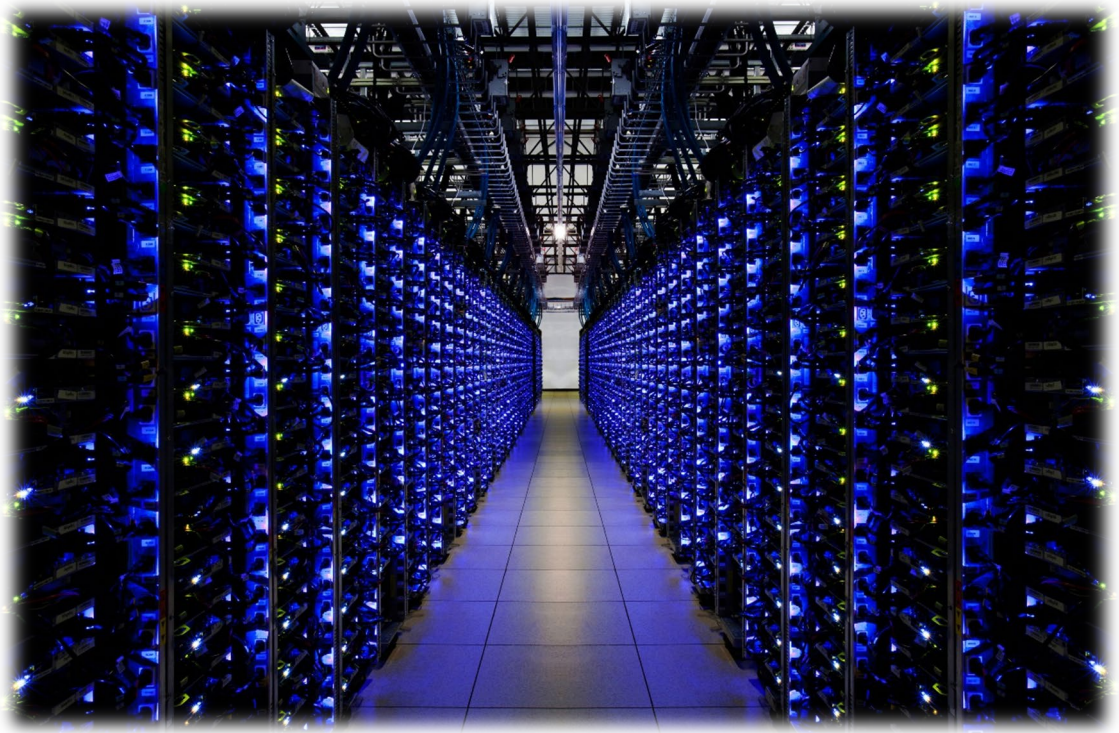
This Week's Menu

Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

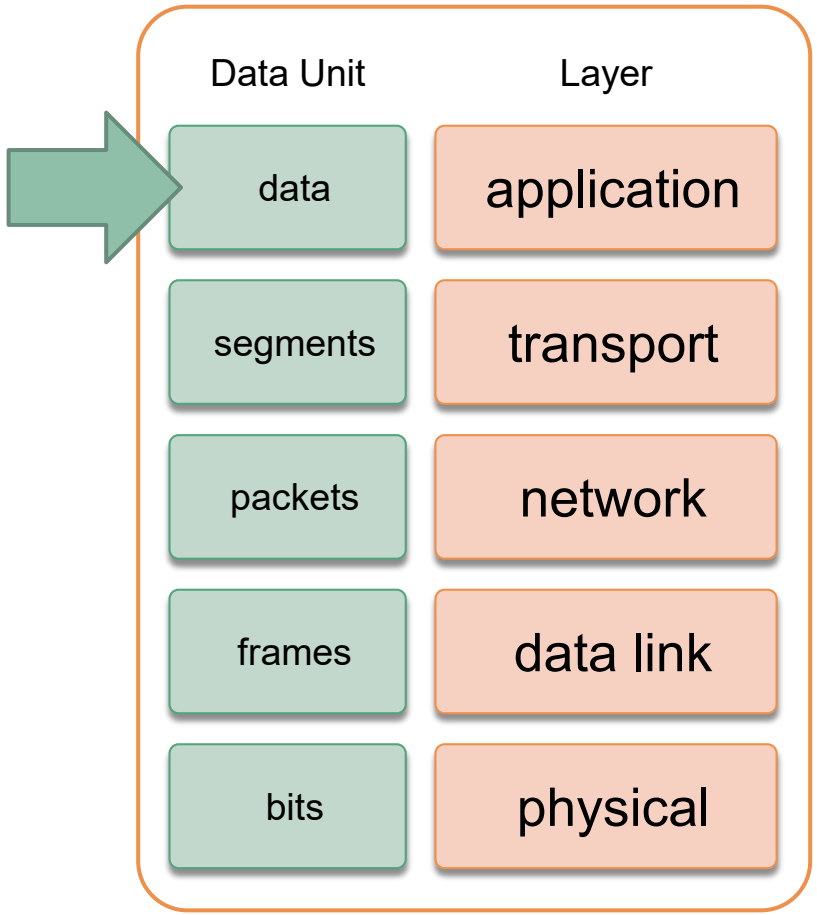


Our Goal



Get these to **interact meaningfully**

The Networking Stack: Application



Applications are what is is built **on top of** a layered networking stack. They communicate with each other through application-layer protocols.

Example: a **Web browser application** communicating with a **Web server application** (using the HTTP or HTTPS protocol)

Example: an **email client** communicating with an **email server** (using the SMTP protocol)

The Domain Name System: Naming vs. Addressing

IP addresses **depend on the network topology**

- Reorganizing a network may **change** all IP addresses
- Identifying important hosts should **not** be address-based

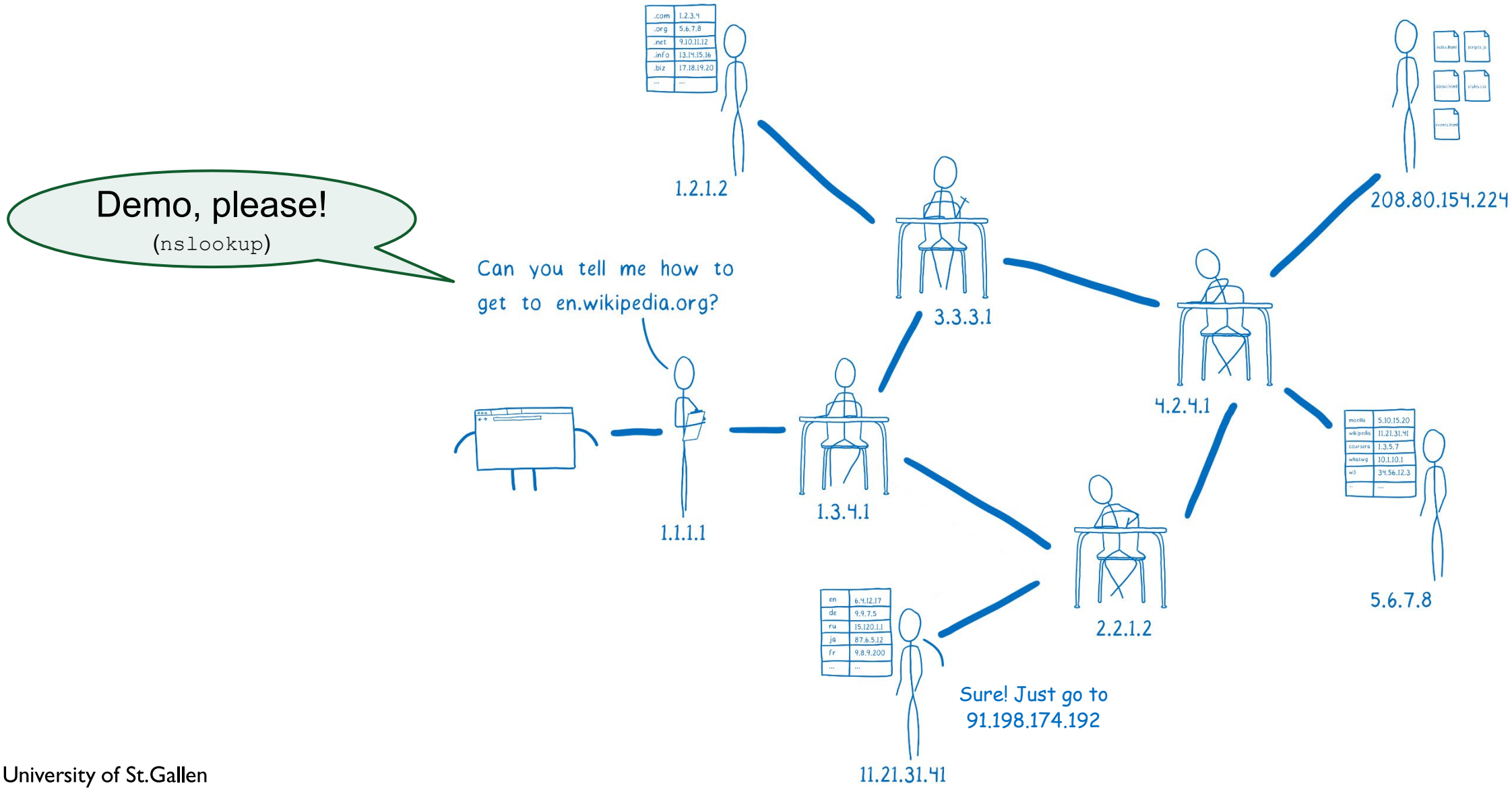
Names are supposed to be **more stable** than addresses

- A name is an **abstract identification** of something
- Network services should use names instead of addresses

Before using a named service, a **mapping** has to be performed

- The **Domain Name System (DNS)** provides this service
- In *www.unisg.ch, ch* is the Top-Level Domain (TLD)
- TLDs are either generic (gTLD, e.g. *com*) or country code (ccTLD, e.g. *ch*)

The Domain Name System and the World Wide Web



The Hypertext Transfer Protocol (HTTP)

```
GET /path/to/resource   HTTP/1.1
Host: example.com
Accept: text/html, *
User-Agent: HTTPTool/1.1
[blank line here]
```

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
 <body>
  <h1>Hello World!</h1>
 </body>
</html>
```

HTTP is a *text-based* protocol

- The connection is used to transmit **text messages**
- The messages can contain **entities** that are encoded in a variety of formats
  - Textual: *text/html, text/xml, application/json, text/csv*, etc.
  - Non-Textual: *image/png, image/jpg*
  - Application-Specific: *application/vnd.openxmlformats-officedocument.presentationml.presentation*, etc.
- **HTTP messages** are human-readable (not all *entities*, though)
- HTTP needs a **reliable** connection – it therefore builds on top of **TCP**
- **Advantages** of being text-based: Understandable & Debuggable! Basic HTTP operations can be carried out by hand!
- **Disadvantage** of being text-based: Not very efficient…

The client sends a (textual) request to a resource

- The **HTTP method** indicates the action to be performed on the resource
- The request also includes information about what **type of response** the client will accept

e.g., create a new blog post

In response to the request, a server will **return** a (textual) **response**, **modify** the resource and/or **create** a new resource

e.g., return wikipedia page.

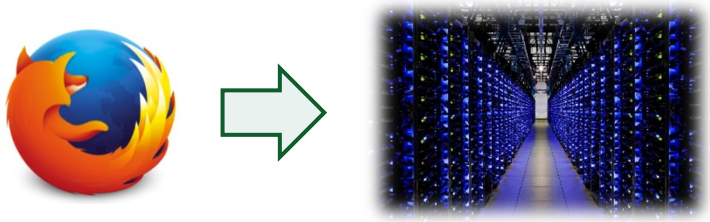
e.g., put something in your shopping basket.

HTTP

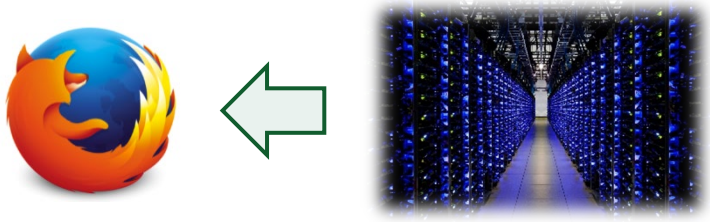
We can do this manually as well ☺

(`telnet`)

(remember to set localecho using CTRL+])



```
GET /path/file.html HTTP/1.1
Host: example.com
Accept: text/html, *
User-Agent: HTTPTool/1.1
[blank line here]
```



```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
 <body>
  <h1>Hello World!</h1>
 </body>
</html>
```

HTTP Requests

The URI **identifies the resource** to which the request should be applied

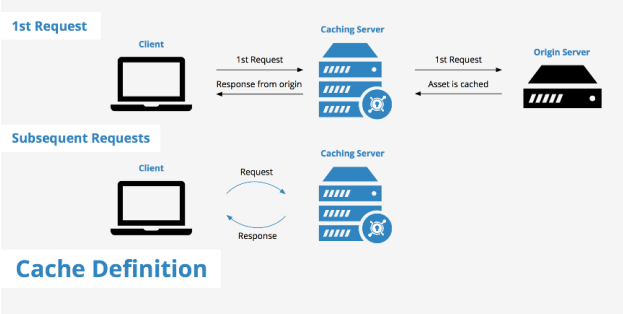
- It may contain **query information:** https://www.google.com/?gws_rd=ssl#q=HTTP

The **HTTP method name** identifies the operation that is applied to the resource

- HTTP's most relevant methods are GET, POST, PUT, and DELETE (but there are more)
- HTTP methods have the **same semantics with every resource** (this is called a “Uniform Interface”)

HTTP **GET**: **Retrieve** a representation of the addressed resource

- Maybe implemented by reading a file, processing a file, running code,…
- GET is **safe**: it **does not modify the Web resource representation**
  - Safe methods can be **cached**
  - Safe methods can be **pre-fetched**
  - **This might sound small, but it is one of the main reasons why the Web can reach Internet scale!**



https://www.keycdn.com/support/cache-definition-explanation

HTTP Headers and Response Codes

HTTP **defines** header fields that contain **information about the request**

- Request headers*: Accept*, *User-Agent*, Connection properties
- Response headers: Content metadata, lifetime, etc.
- Servers/Clients **must ignore** unknown fields – why?
  - This makes the **protocol extensible**
  - (Useful) additions can enter the protocol **over time**
  - **Standard system design principle: "Be conservative in what you send, and liberal in what you accept"**

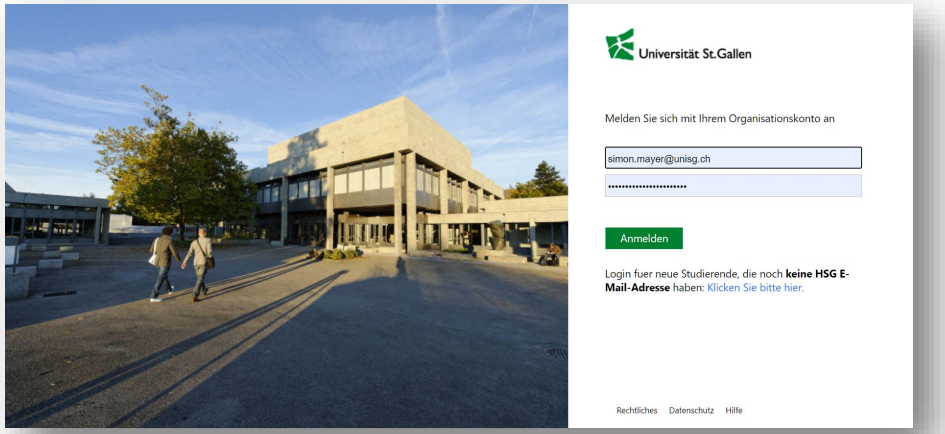
The server's response to acting on a request: Numeric status code + Text

- 2** for variations of OK
- 3** for redirections
- 4** are client side problems
- 5** are server side problems

Playing with HTTP / Web APIs: useful applications such as *Postman*

https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP Access Control



HTTP servers can **deny access** to resources

- *401 Unauthorized* means the resource is **access controlled**
- *403 Forbidden* means the resource is **inaccessible**
- *405 Method Not Allowed* signals a request is using an **impermissible request method**

Usually, in response to a 401, a user is **redirected to a login page** or the browser **displays a "login" window**, collects the user's credentials, and re-submits the request

Two different approaches to unauthorized access are possible

- **Repeat** the HTTP request with the proper authentication credentials
- **Redirect** to a authentication resource for authentication (OAuth): "Authentication-as-a-Service"

https://www.youtube.com/watch?v=tFYrq3d54Dc

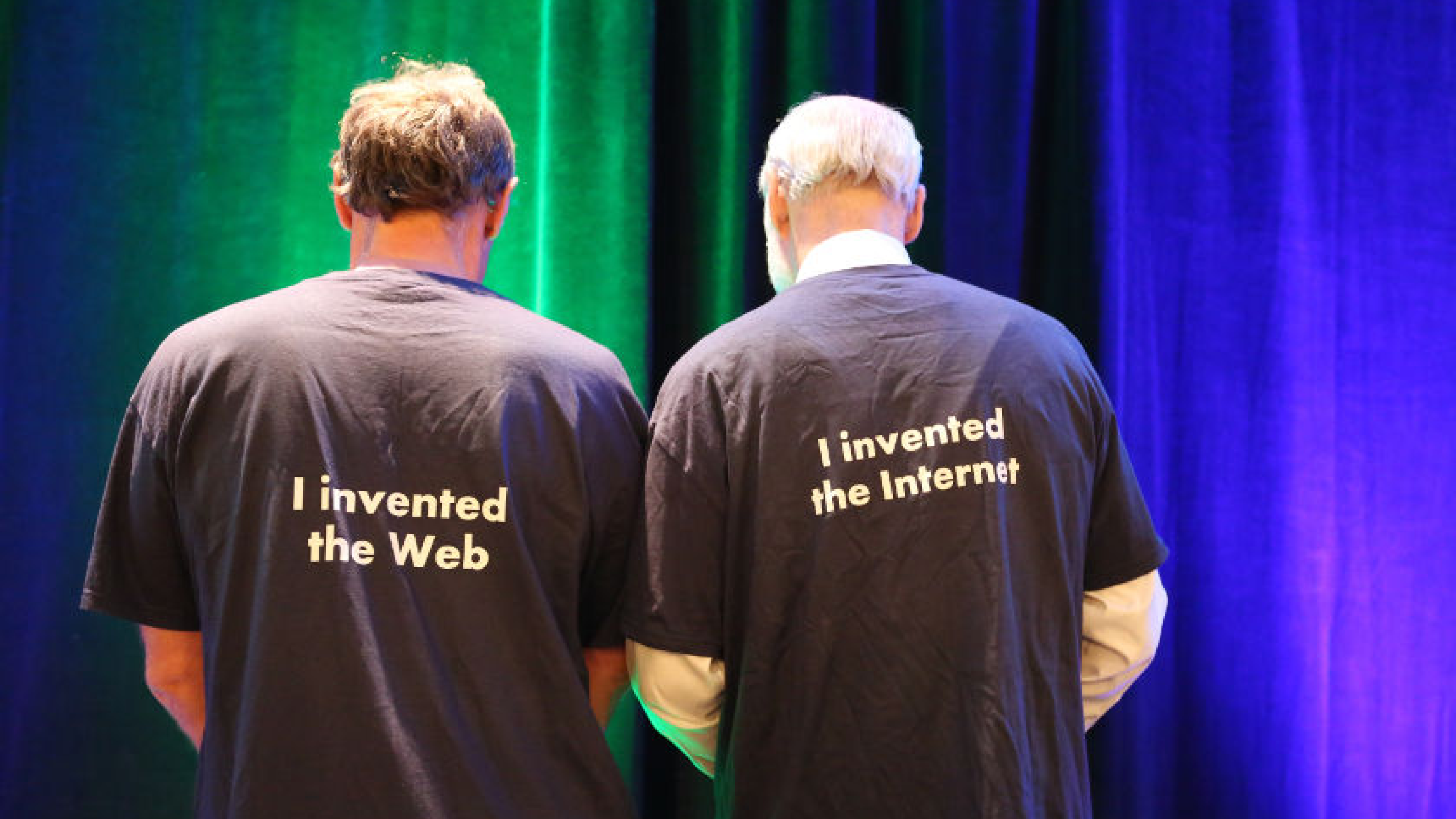
This Week's Menu

Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

Guided Exercise

Briefing: Security
Stock Exchange


I invented
the Web
I invented
the Internet


I didn't invent
the Internet
I did not
invent the Web

Internet

From Wikipedia, the free encyclopedia

This article is about the worldwide computer network. For other uses, see Internet (disambiguation).

Not to be confused with the World Wide Web.

World Wide Web

From Wikipedia, the free encyclopedia

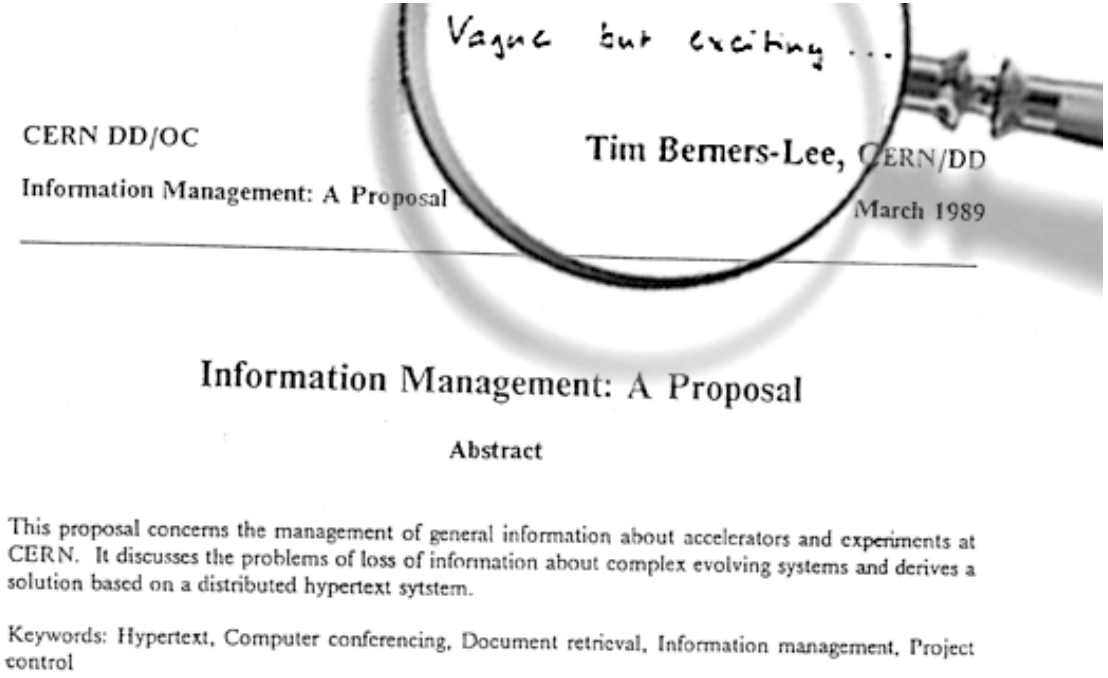
"WWW" and "The Web" redirect here. For other uses of WWW, see WWW (disambiguation). For other uses of web, see Web (disambiguation).

For the first web software, see WorldWideWeb.

Not to be confused with the Internet.

The **World Wide Web** made the Internet popular…

1989: Tim Berners-Lee submits a proposal for an information management system at CERN to his boss, Mike Sendall (http://info.cern.ch/Proposal.html)


Vague but exciting ...

CERN DD/OC

Tim Berners-Lee, CERN/DD

Information Management: A Proposal

March 1989

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

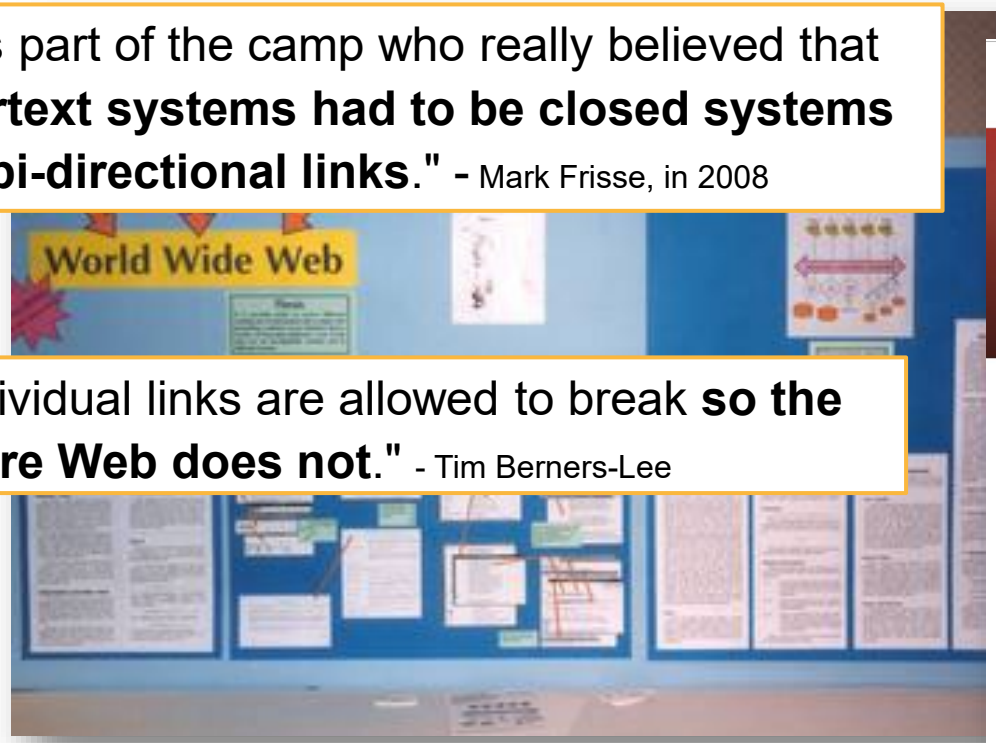
Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control

1990: Berners-Lee defines the terms **URI** (then UDI), **HTTP**, and **HTML**, and develops the first components and tools for the Web (browser, editor, server, line-mode browser)

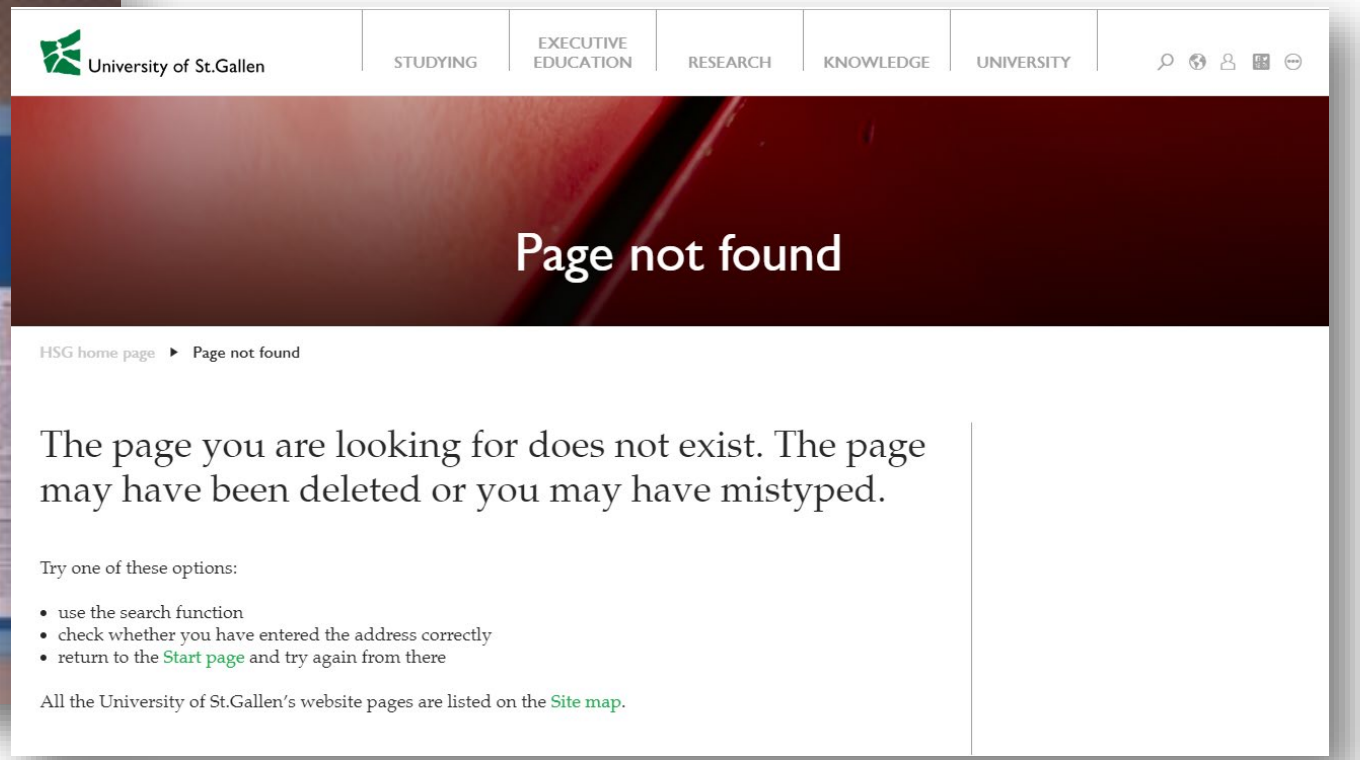
The **World Wide Web** made the Internet popular…

1991: Tim Berners-Lee submits a paper on the Web to the *Hypertext '91 Conference*.
The paper is **rejected as an oral presentation** and only accepted as a **poster** contribution…

"I was part of the camp who really believed that **hypertext systems had to be closed systems with bi-directional links**." - Mark Frisse, in 2008



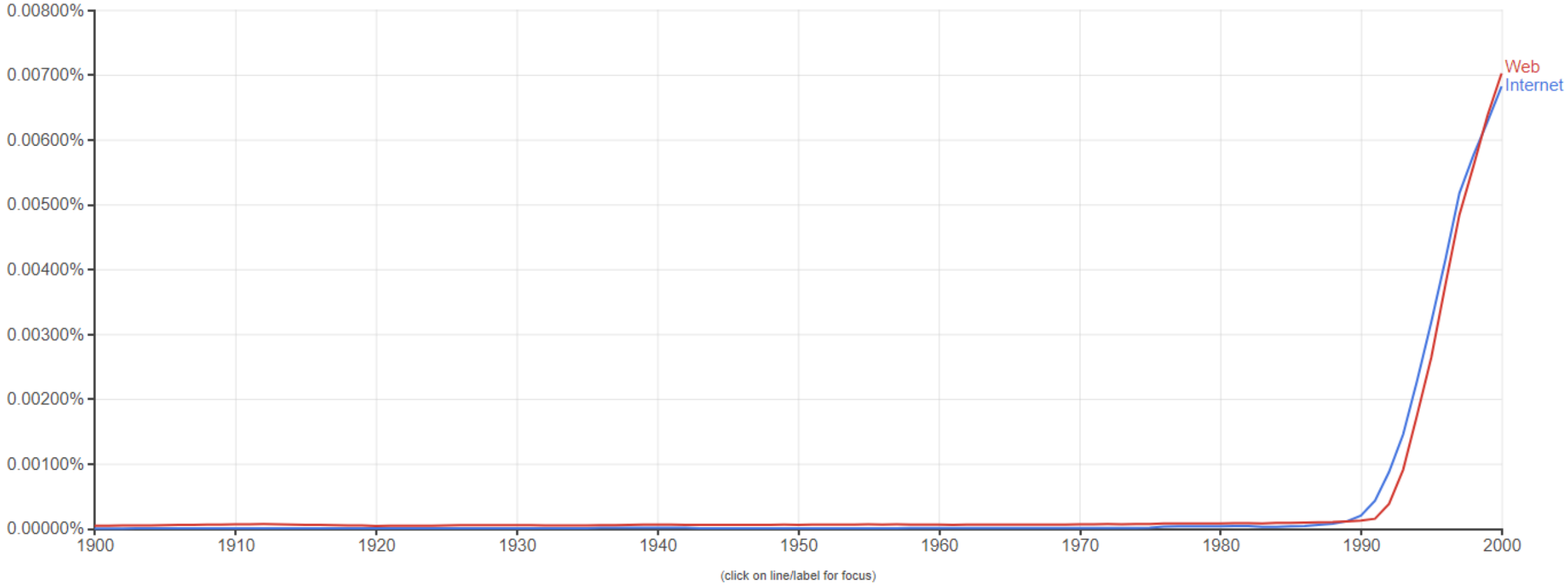
"Individual links are allowed to break **so the entire Web does not**." - Tim Berners-Lee



Google Books Ngram Viewer

Graph these comma-separated phrases: Internet,Web case-insensitive

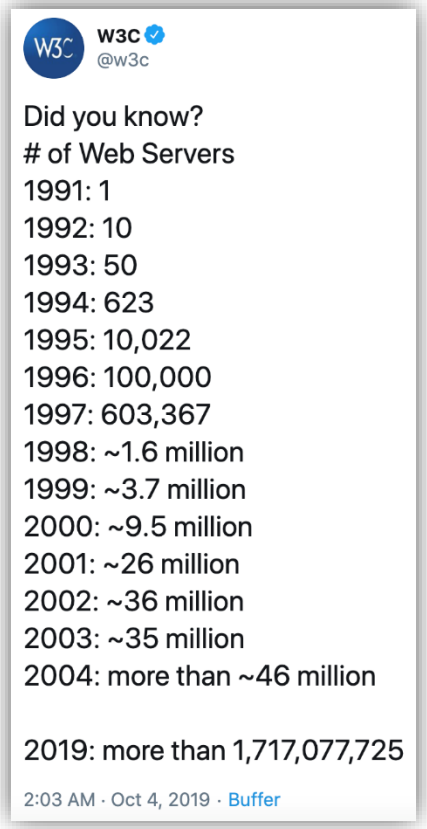
between 1900 and 2000 from the corpus English with smoothing of 3 . Search lots of books



The **World Wide Web** is an information system where **resources** are identified by **Uniform Resource Identifiers** (URIs) which are accessible over the **Internet**, and may be interlinked by **hyperlinks**.

It's main application-layer protocol is **HTTP**.

The Web in Numbers



In 2016:

In 2020:

> The Google Search index contains hundreds of billions of webpages and is well over 100,000,000 gigabytes in size. It's like the index in the back of a book — with an entry for every word seen on every webpage we index. When we index a webpage, we add it to the entries for all of the words it contains.

Google Search, "How Search Works"

Nobody can state ***precisely*** how large the Web is …
but it is certainly an **Internet-scale** system.

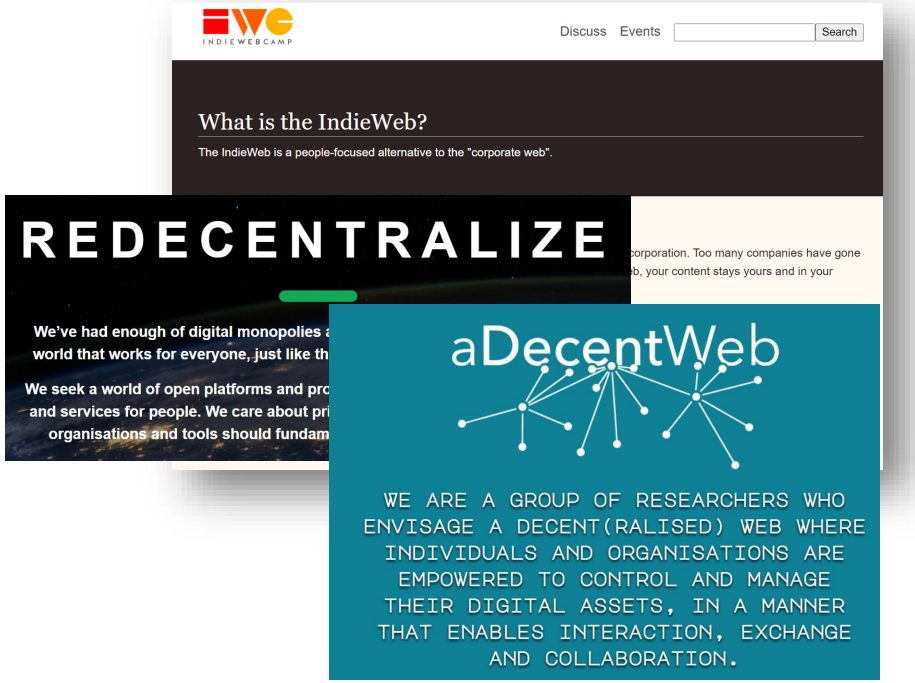
https://www.google.com/search/howsearchworks/crawling-indexing/

Selected Threads to the **Open Web**

Note that «open» neither implies «insecure» nor «uncontrolled»

Dominant online services (e.g., Facebook) replacing the **open** Web

Platforms try to **own your data**, but are **not themselves interoperable** (GDPR undermines this)




WIPO MAGAZINE

European Court of Justice: hyperlinks to unauthorized content may infringe copyright

December 2016

By **Tobias Cohen Jehoram**, Partner, De Brauw, Blackstone, Westbroek, Amsterdam, Netherlands

* This article was first published at: *www.debrauw.com/newsletter/ecj-hyperlinks-unauthorised-content-may-infringe-copyright*

Hyperlinking has become more risky. In its long-awaited judgment in Sanoma/GS Media, delivered on September 8, 2016, the European Court of Justice (ECJ) clarified that the posting of a hyperlink to unauthorized content protected by copyright and hosted by a third party may result in copyright infringement. This will be the case if the person posting the link knows or should have known that the hyperlink

Putting it all Together…

The server's response will contain a **representation of the resource** at https://apps.unisg.ch in a format that can be rendered by the client (in this case: HTML).

This is a **URI**. It is also a **URL**
The URI's *scheme* is **https**
Its *path* is **apps.unisg.ch**

https://apps.unisg.ch/

DNS can resolve this hostname
It resolves to the *IPv4 address* **130.82.40.87**
It does not resolve to an *IPv6 address* (currently)

The hostname is **apps.unisg.ch**
Its DNS *leaf domain* is **apps**
Its DNS *second-level domain* is **unisg**
Its DNS *top-level domain* is **ch**

When we access this URL through a Web browser, the browser will formulate an **HTTP GET** request, get the server's IP address through **DNS**, wrap the HTTP into a **TCP segment** (at the https default port of 443), wrap this packet into an **IP packet** with the proper address information and put it on the **carrier** (properly packaged for that carrier, of course).

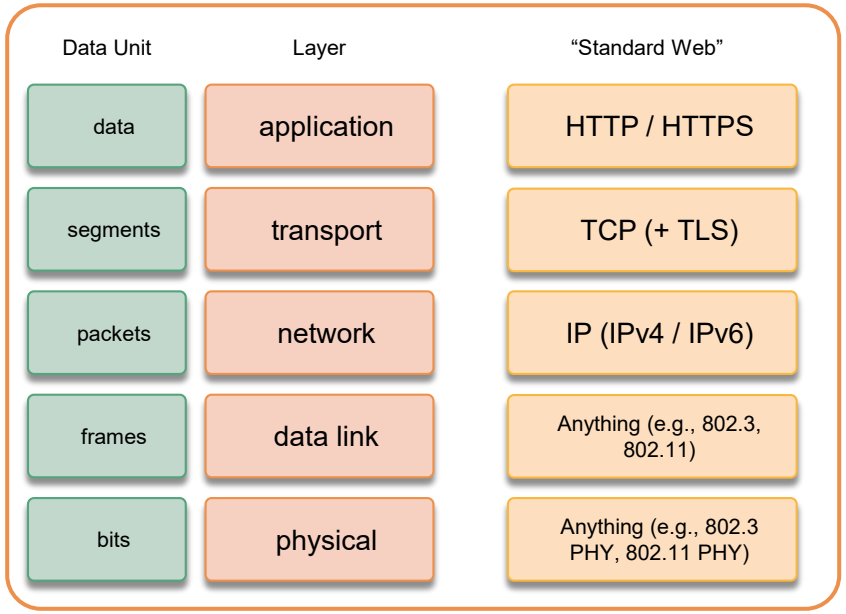
Conclusion: Networking and the World Wide Web

The **Internet** is a network of networks that all use the **Internet Protocol**

- **IP** provides addresses for Internet hosts and forms a narrow waist of the networking stack
- **TCP** is used for reliable, in-order communication and flow control on top of IP
- **UDP** is used for unreliable best-effort communication – it also builds on IP
- **DNS** assigns stable (host)names to IP addresses
- The **Web** is one of the main applications of the Internet, its main protocol at the moment is HTTP(S)

Next Assignment: **Practice**!

Lots of working with **Web APIs.**
Creation of **Web mashups.**



This Week's Menu

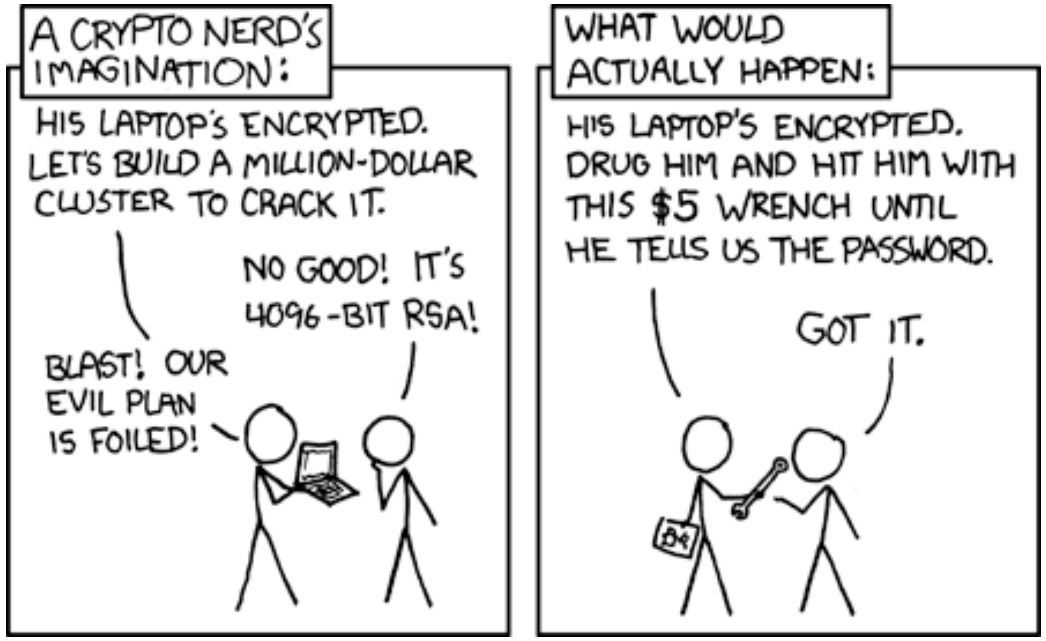
Lecture

Motivation
Computer Networks Basics
The Networking Stack
The World Wide Web

Guided Exercise

Briefing: Security
Stock Exchange

HTTPS and TLS



TLS (Transport Layer Security) is a cryptographic protocol that provides **data privacy** and **data integrity** as well as **authentication** between computer applications

- TLS runs "on top of" a reliable transport (e.g., TCP), so it's treated as a **transport-layer protocol**
- Applications must actively initiate TLS, so it's sometimes classified as an **application-layer protocol**

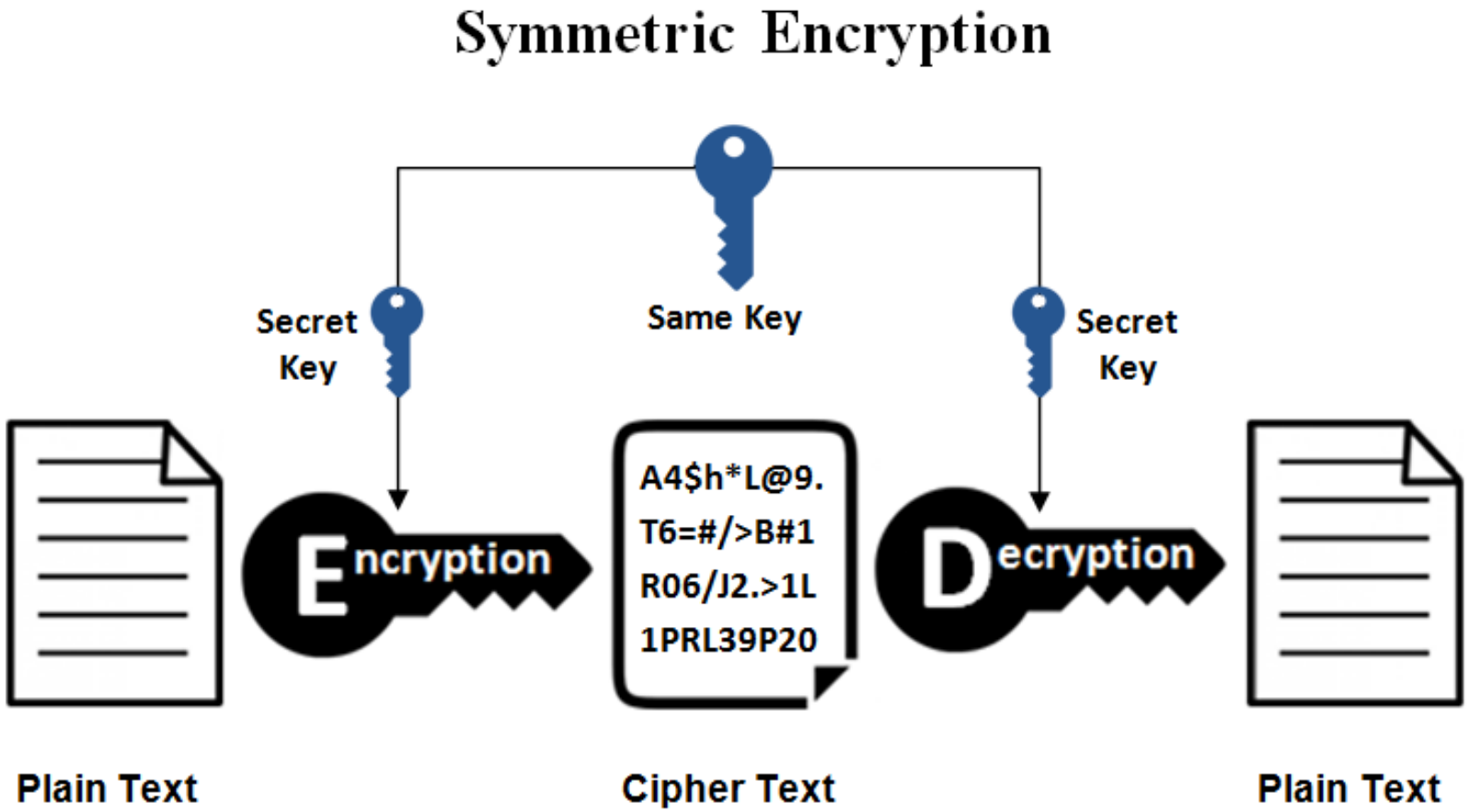
Website protocol support (Apr. 2021)

Protocol version	Website support[67]	Security[67][68]
SSL 2.0	0.5%	Insecure
SSL 3.0	3.4%	Insecure[69]
TLS 1.0	46.9%	Deprecated[8][9][10]
TLS 1.1	52.3%	Deprecated[8][9][10]
TLS 1.2	99.4%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	44.8%	Secure

https://en.wikipedia.org/wiki/Transport_Layer_Security

Encryption Primer…

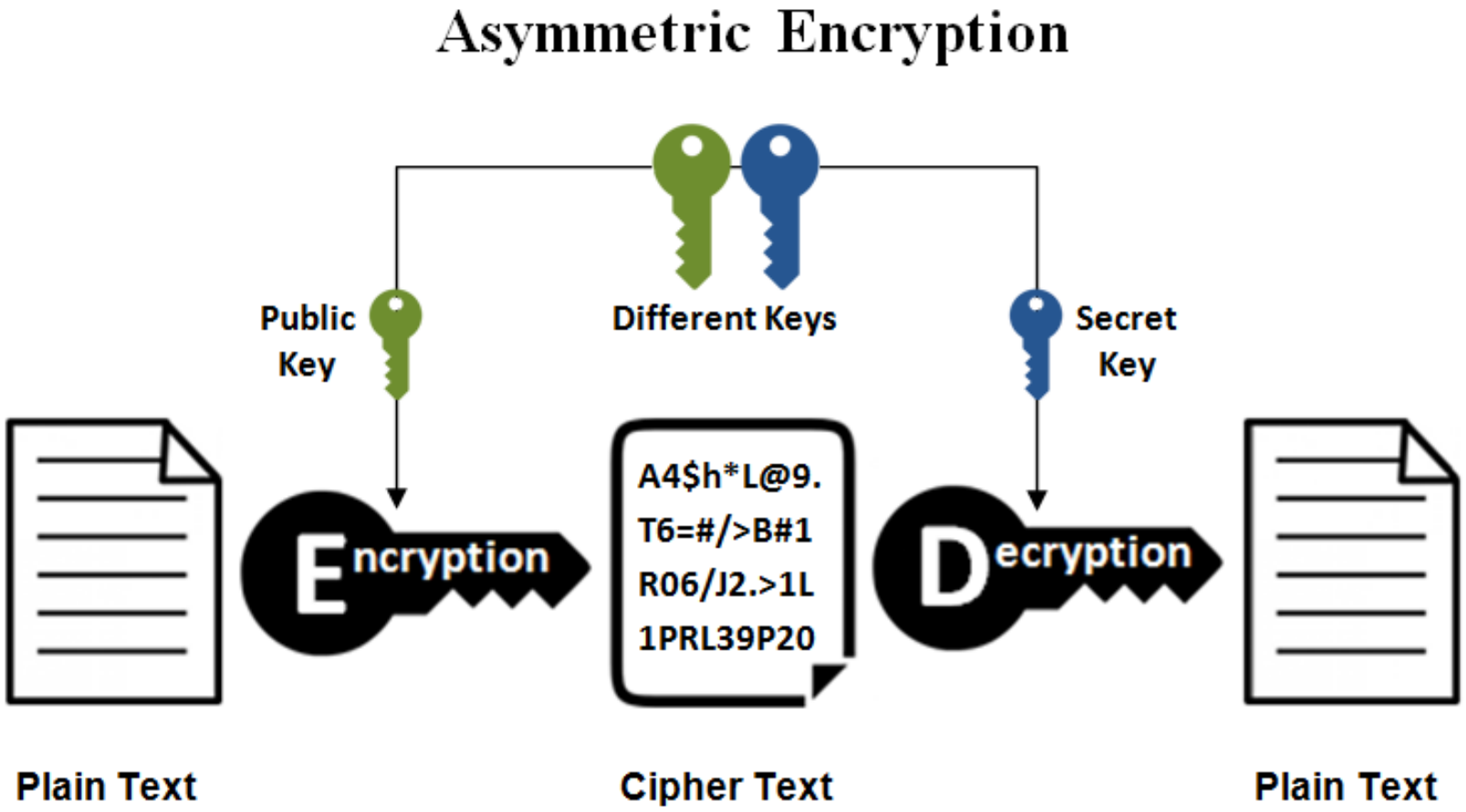
Advantage: Fast! Simple!
Disadvantage: Requires shared key!
Well-known Algorithms: AES, DES



https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

Encryption Primer…

Advantages: No shared secret! Usually also enables digital signatures!
Disadvantages: Slow! How to know that a public key is authentic?
Well-known Algorithms: RSA, DSA, Elliptic curve techniques



https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

Let's end with «Magic»: How to Establish Shared Secrets?

Wouldn't it be great if we could…

…use a "slow" method to **derive a shared secret**

…then use this shared secret for **"fast" symmetric encryption**

Excellent: https://www.youtube.com/watch?v=YEBfamv-_do from 2:08

https://networklessons.com/uncategorized/diffie-hellman-key-exchange-explained/

TLS Features

Here is (quality-controlled) background material about this:
https://en.wikipedia.org/wiki/Public_key_infrastructure

Setup

- The communicating parties use **symmetric encryption** to encrypt the transmitted data
- The **encryption keys** are **generated from a shared secret** and are unique for each connection
- The shared secret, encryption algorithm, and keys are **negotiated** at the start of each session (“TLS Handshake”), based on protocols such as DH
- Each entity has a **private and a public key**; these are mathematically coupled to get the following properties
- Each entity **advertises its public key** and **keeps the private key private**

Main Feature #1: Secure Connections

- The encrypting entity **encrypts messages** with the **receiver’s public key**
- The receiver **decrypts messages** with its **own private key**

Main Feature #2: Authenticated Identities (usually: optional for the client but required for the server)

- The authenticating entity **signs messages** with its **private** key
- The authenticating entity also **advertises** its **public** key

Do you see any problem here?

- How do we know that the advertised public keys are correct?
- If we don’t, so-called **machine-in-the-middle attacks** become feasible!
- Solution: The authenticity of the public keys are certified by a trusted third party called a **certification authority**

HTTPS and TLS

Data Unit	Layer
data	application
segments	transport
packets	network
frames	data link
bits	physical

HTTPS (“HTTP-over-TLS”) = Security on the **transport layer!**

– The entire request **above transport is encrypted** but nothing at or below the transport layer

So what does that mean **in practice**? When I send an HTTPS request, …

– …can an eavesdropper **modify the request** sent over HTTPS? **Why?**
– …can he see the **request contents**? **Yes!**
– …can he see the **request URL**? **Yes!**
– …can he see the **host IP address**
– …can he see the **host port number**? **Yes!**
– …can he see the **HTTP request method**?
– …can he see the **amount of data** transferred? **Yes!**
– …can he see the **duration** of the interaction with the host? **Yes!**

Any Questions / Comments / Doubts / Concerns?