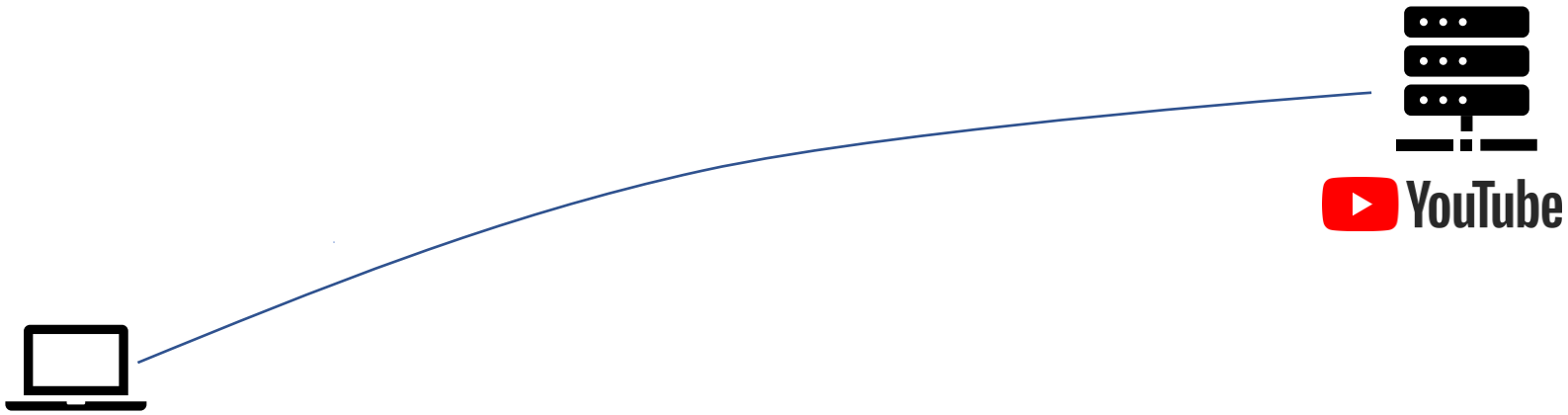
Problem / Overview

Course: Networking Fundamentals
Module: Application

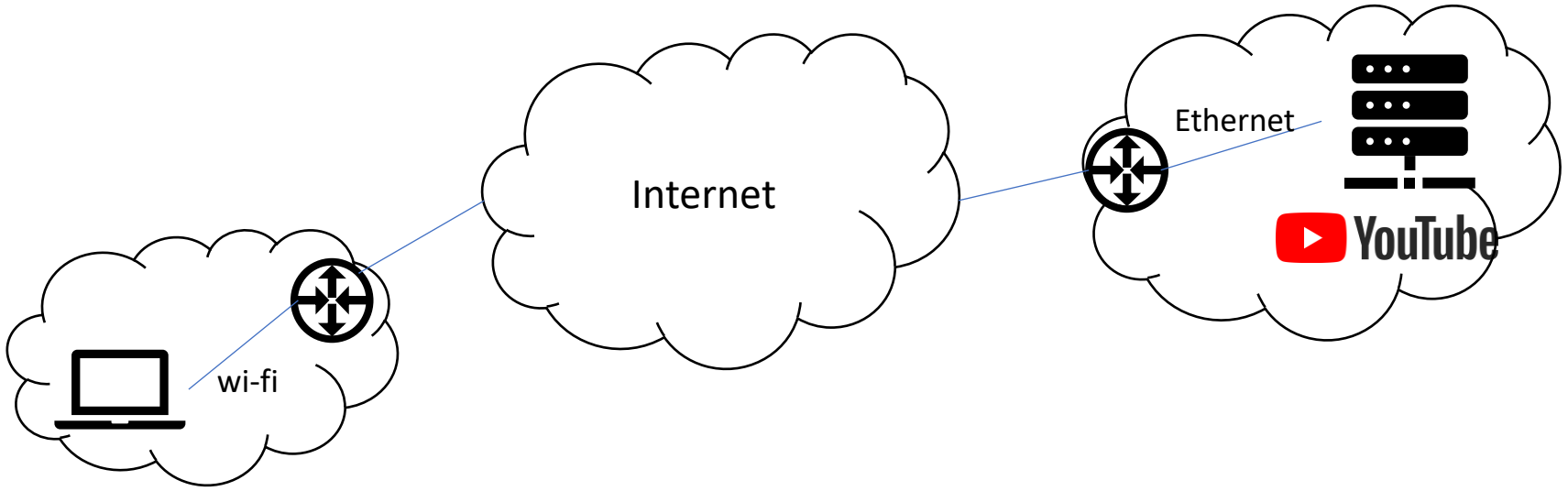
University of Colorado **Boulder**

Visiting Youtube (or any other service)



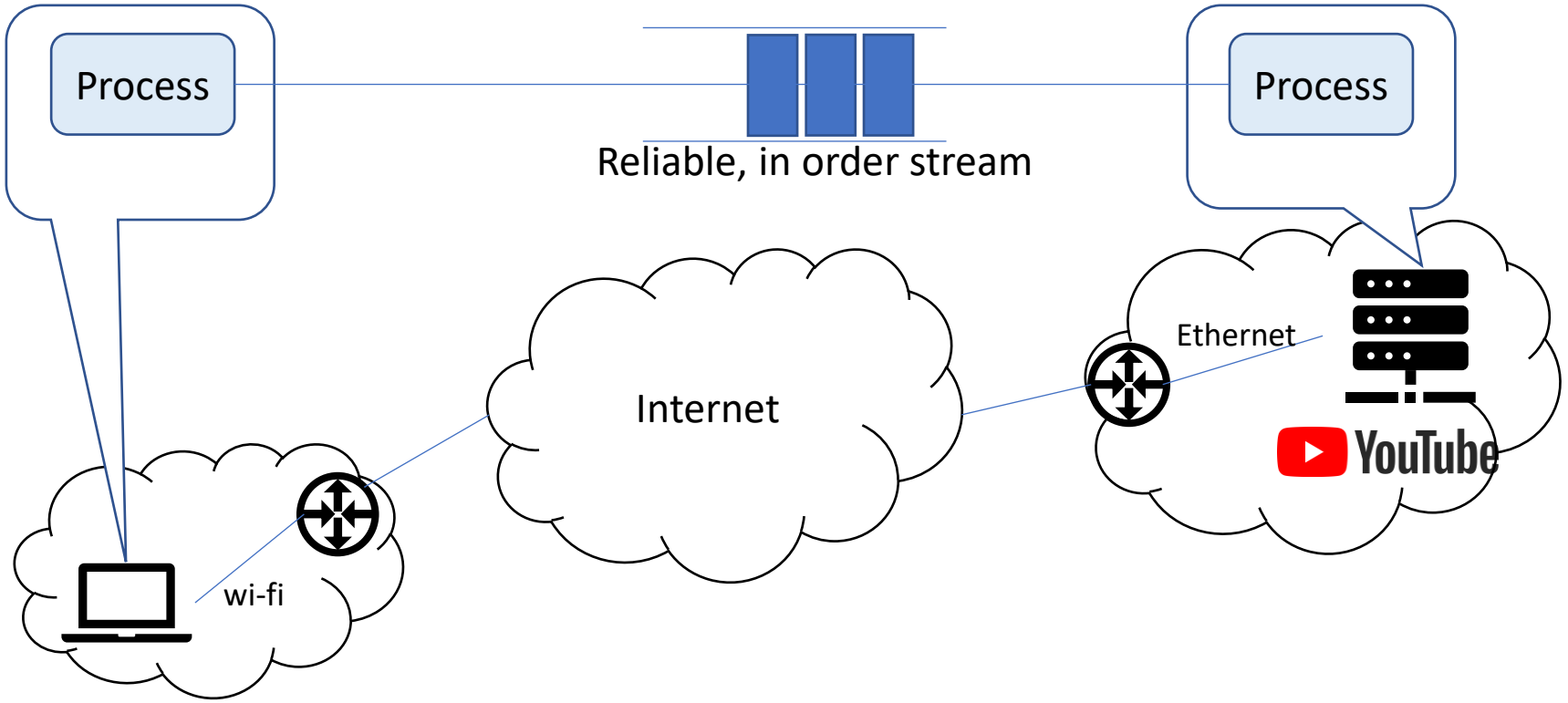
Link and Network Layer

- Link layer enables device to talk on a local network (e.g., to a gateway)
- Network layer enables global scalability, and interconnecting different networks (wi-fi, Ethernet)

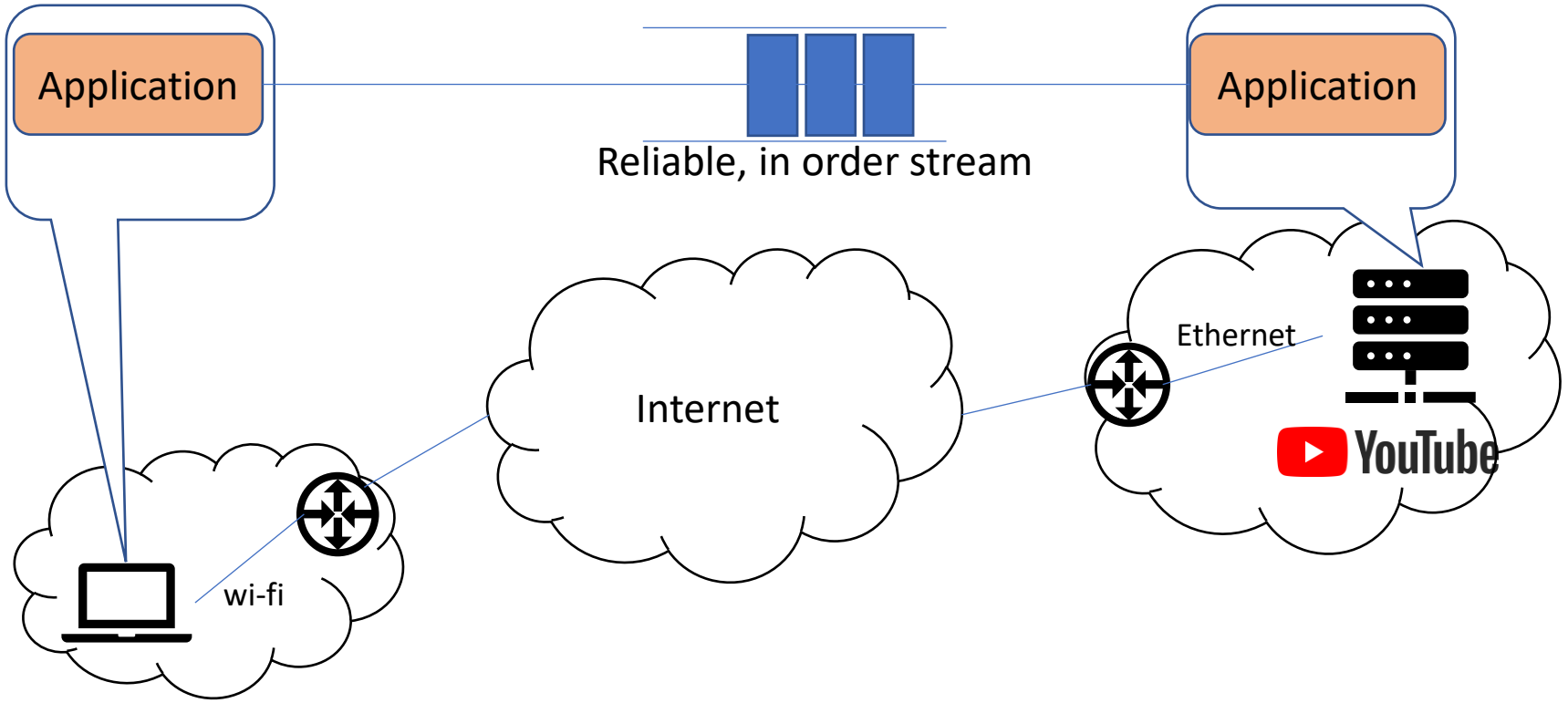


Transport Layer

- Transport layer enables process-to-process, and reliable in-order stream

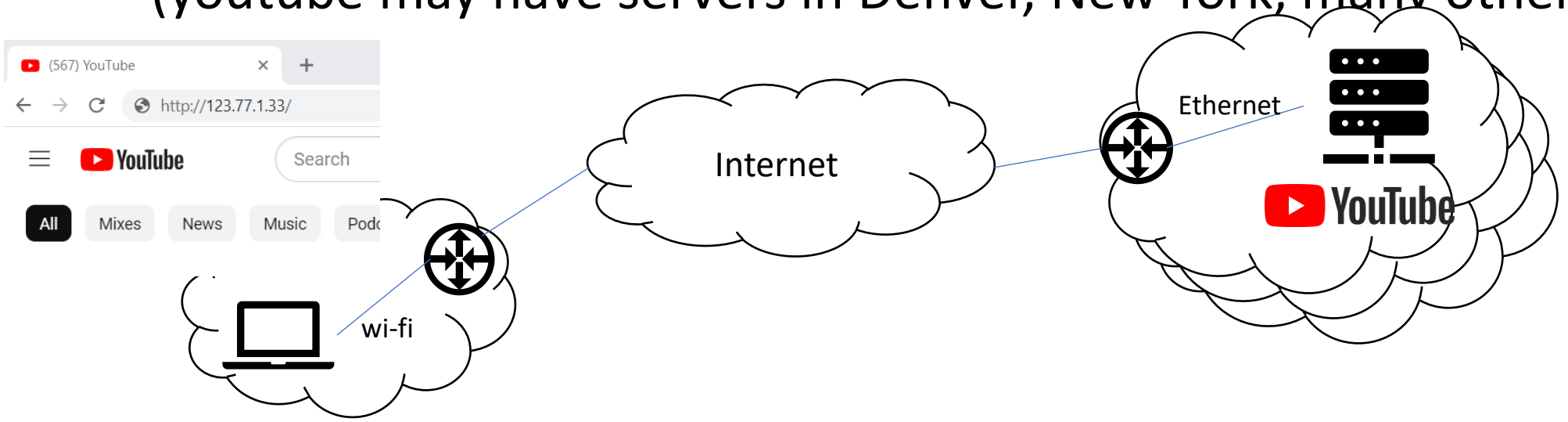


Application Layer



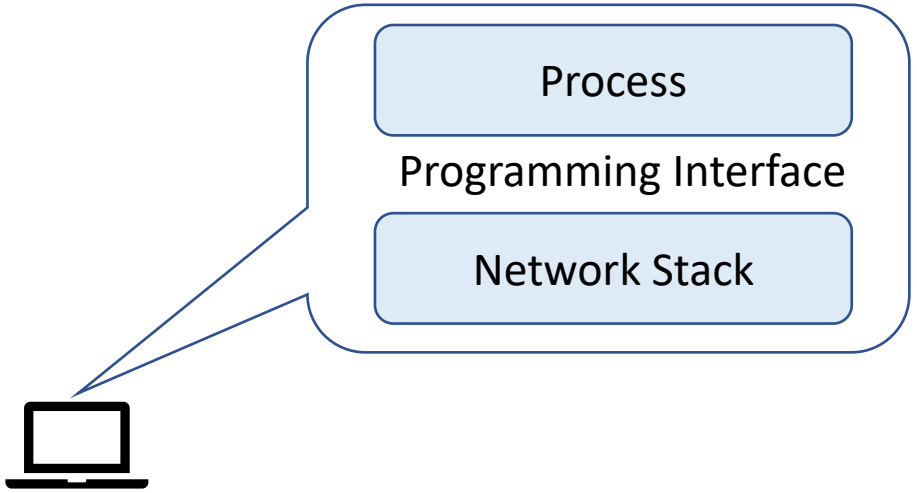
Problem 1: Addressing

- IP address isn't human readable
  (want to go to youtube.com)
- IP address represents single machine
  (youtube may have servers in Denver, New York, many others)



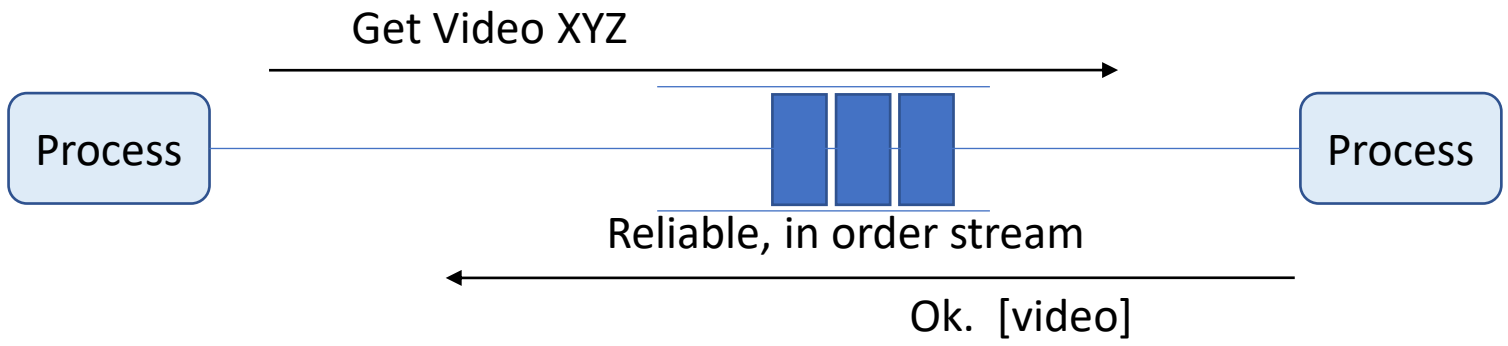
Problem 2: Programming

- Link, Network, and Transport layer protocols are concepts, and the operating system network stack is their implementation
- How do we write programs to interface to this network stack?



Problem 3: Application-level protocol

- How should the processes communicate what they want (e.g., with youtube – I can watch a video, or I can upload a video)
- How do processes encode data



University of Colorado **Boulder**

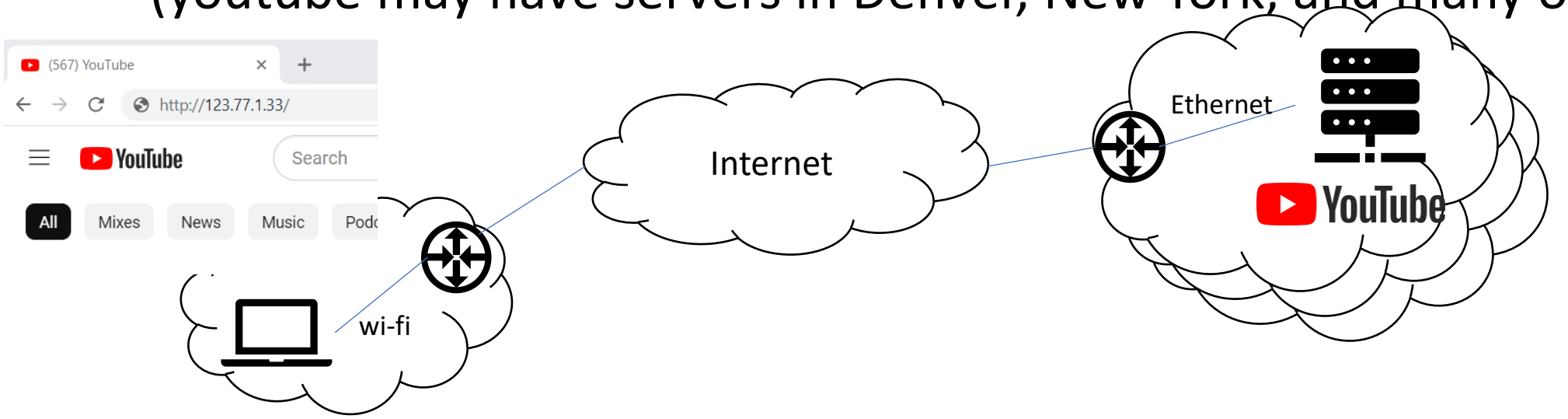
Domain Name System

Course: Networking Fundamentals
Module: Application

University of Colorado **Boulder**

Problem 1: Addressing

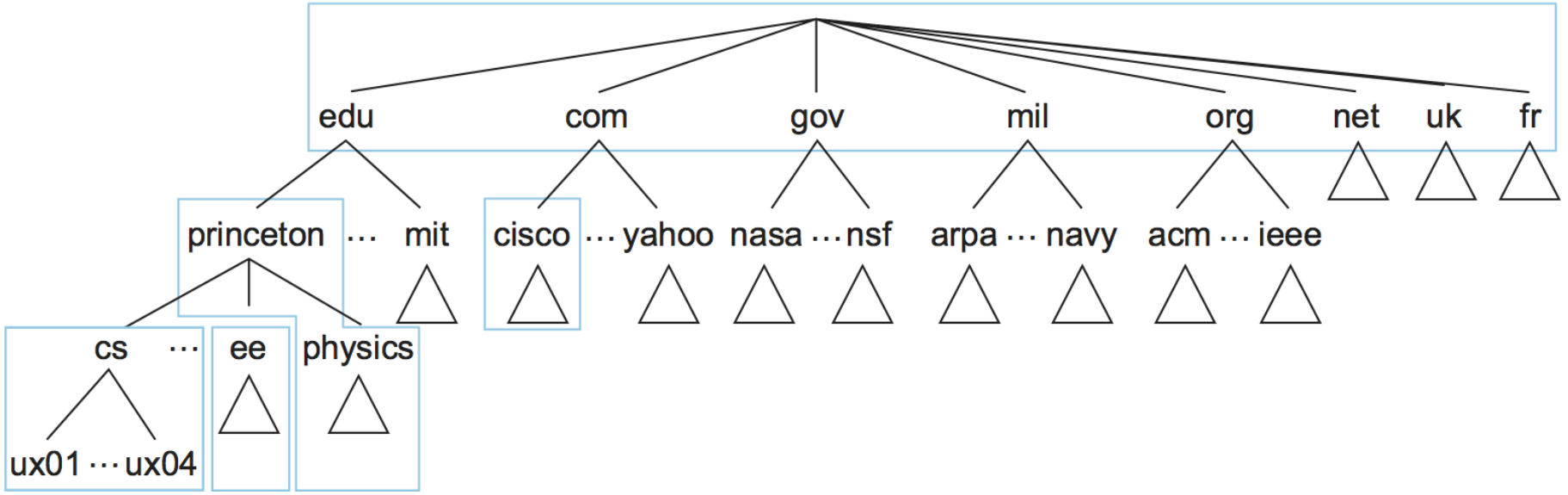
- IP address isn't human readable
  (want to go to youtube.com)
- IP address represents single machine
  (youtube may have servers in Denver, New York, and many others)



Addressing at Different Layers

	Host Name	IP Address	MAC Address
Example	www.cs.colorado.edu	128.138.7.156	00-15-C5-49-04-A9
Size	Hierarchical, human readable, variable length	Hierarchical, machine readable, 32 bits (in IPv4)	Flat, machine readable, 48 bits
Read by	Humans, hosts	IP routers	Switches in LAN
Allocation, top-level	Domain, assigned by registrar (e.g., for .edu)	Variable-length prefixes, assigned by ICANN, RIR, or ISP	Fixed-sized blocks, assigned by IEEE to vendors (e.g., Dell)
Allocation, low-level	Host name, local administrator	Interface, by admin or DHCP	Interface, by vendor

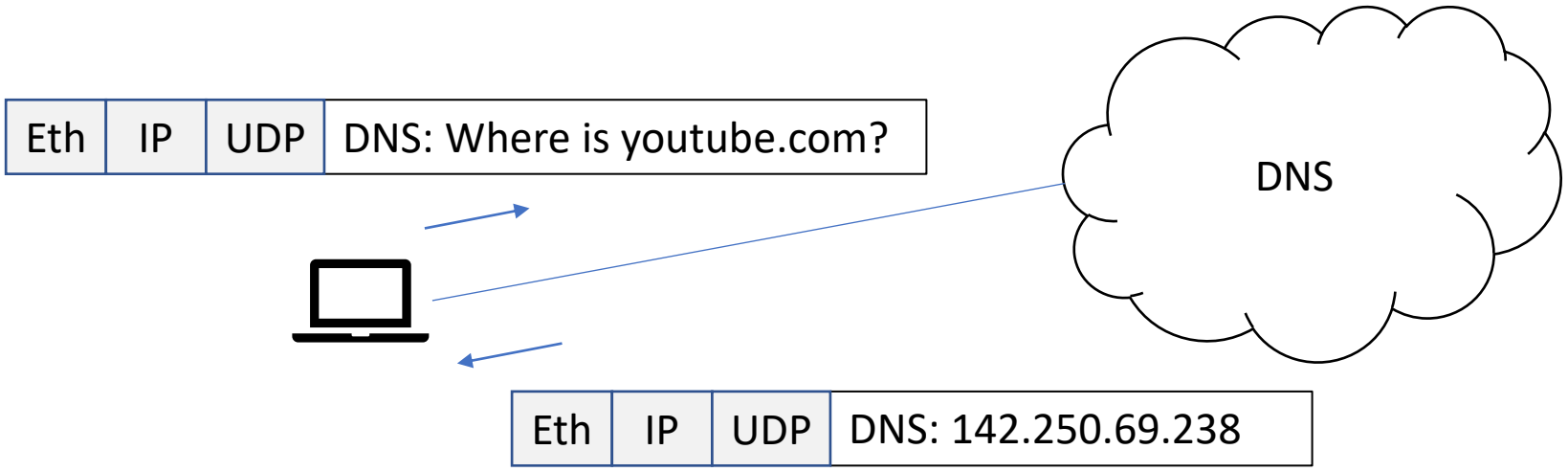
Hierarchical Naming



Pic: https://book.systemsapproach.org/applications/infrastructure.html

Domain Name System

- Distributed database of mappings between hostnames and IP addresses
- Application layer protocol that runs on top of UDP



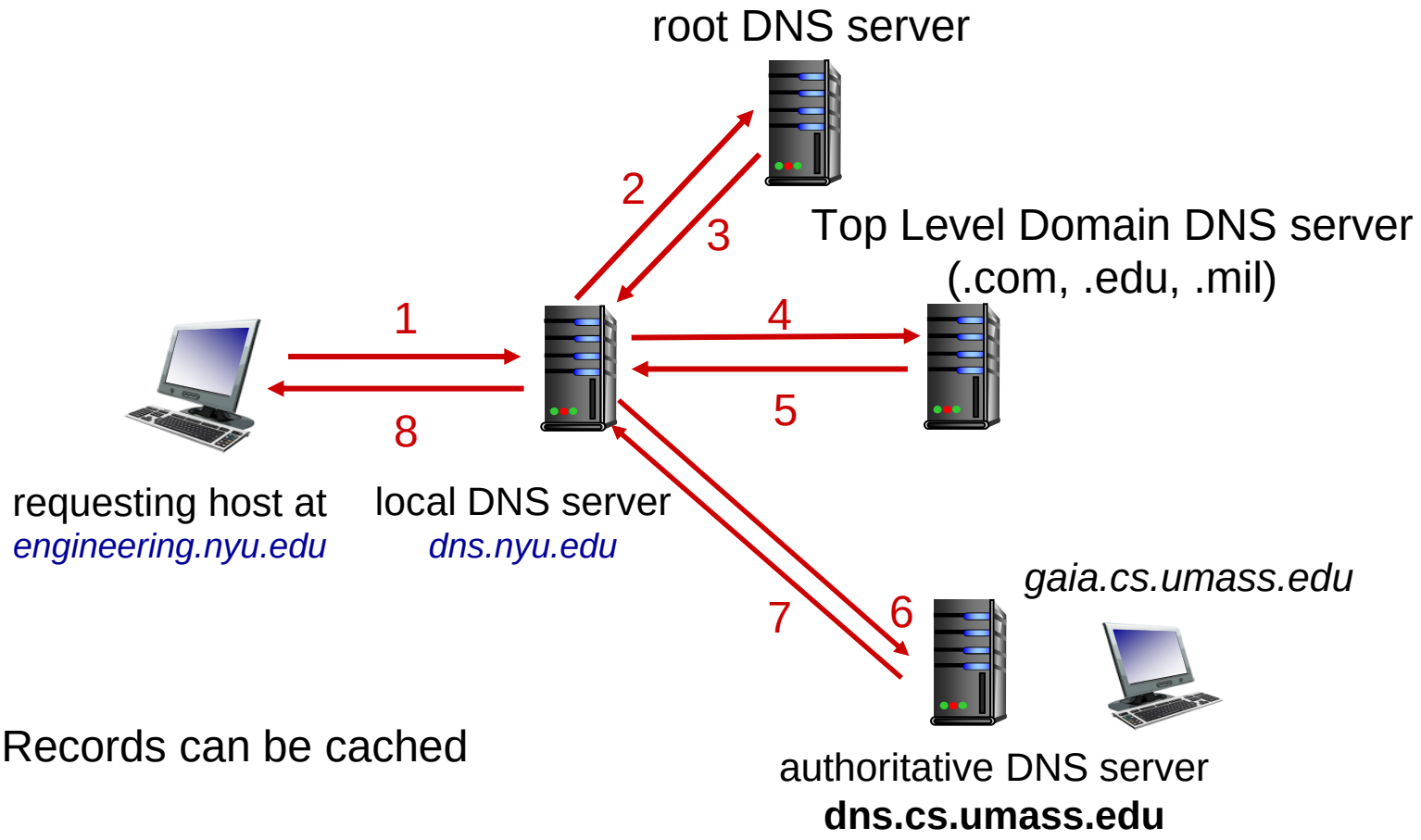
DNS Records

DNS: distributed database storing resource records (RR)

RR format: (`name, value, type, ttl`)

- Type A
  - Name – hostname
  - Value – IP address
- Type NS
  - Name – domain
  - Value – hostname of authoritative name server for this domain

Name Servers



Records can be cached

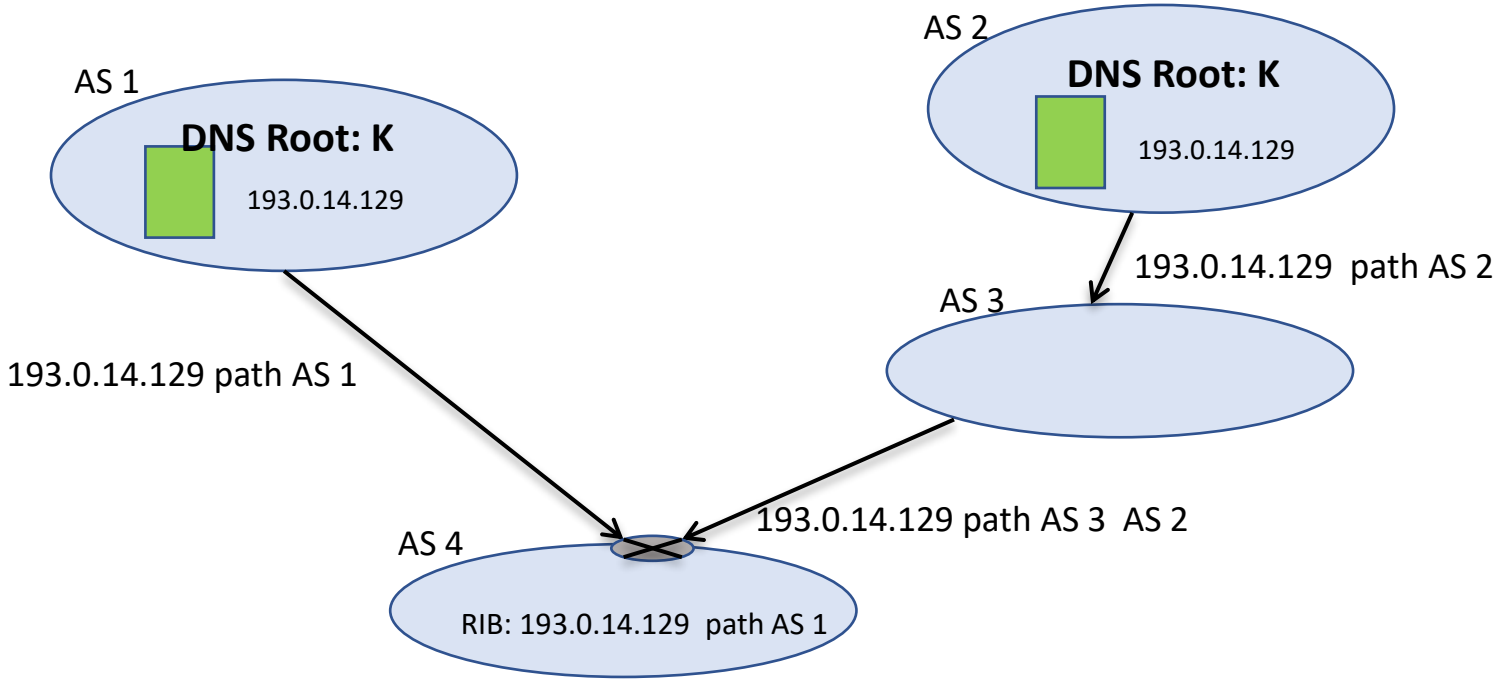
Pic from: https://gaia.cs.umass.edu/kurose_ross

Root Servers

- DNS is critical infrastructure for the Internet / Web
- Root servers are critical within DNS
- Known set of 13 root servers (IP addresses are configured in resolvers)
- Each root server is replicated for redundancy

Any Cast

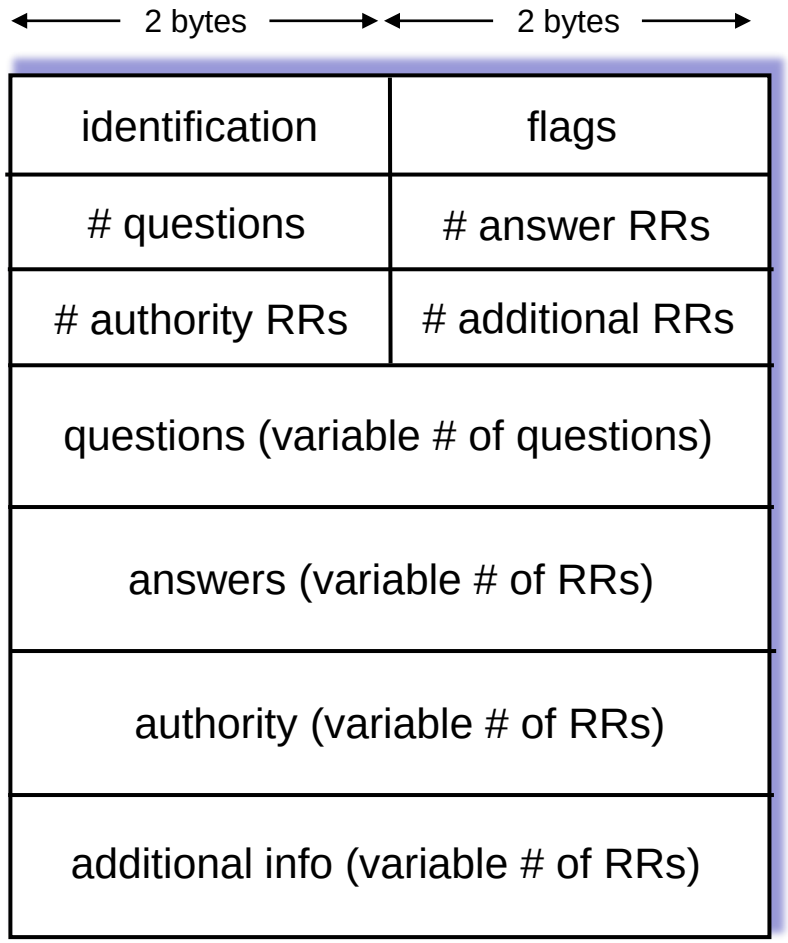
- Advertise a single IP prefix from multiple distinct locations.
- Hosts trying to reach that prefix will (hopefully) go to the closest one



Message Format

Query and Reply has same format

- Identification – reply will match query
- Flags – tell if query or reply, etc.
- Questions, answers, authority, additional



Questions

- Contains a query of the hostname being looked up

Resource record (RR) fields

Field	Description	Length (octets)
NAME	Name of the requested resource	Variable
TYPE	Type of RR (A, AAAA, MX, TXT, etc.)	2
CLASS	Class code	2

```
⊟ Queries
  ⊟ www.google.com: type A, class IN
      Name: www.google.com
      Type: A (Host address)
      Class: IN (0x0001)
```


Pic: https://en.wikipedia.org/wiki/Domain_Name_System

Pic: https://www.catchpoint.com/blog/how-dns-works

Answers

- Responses to queries if resolver has the answer

Resource record (RR) fields

Field	Description	Length (octets)
NAME	Name of the node to which this record pertains	Variable
TYPE	Type of RR in numeric form (e.g., 15 for MX RRs)	2
CLASS	Class code	2
TTL	Count of seconds that the RR stays valid (The maximum is $2^{31}-1$, which is about 68 years)	4
RDLENGTH	Length of RDATA field (specified in octets)	2
RDATA	Additional RR-specific data	Variable, as per RDLENGTH

```
⊟ Answers
  ⊟ www.google.com: type A, class IN, addr 74.125.131.147
      Name: www.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 5 minutes
      Data length: 4
      Addr: 74.125.131.147 (74.125.131.147)
  ⊞ www.google.com: type A, class IN, addr 74.125.131.103
  ⊞ www.google.com: type A, class IN, addr 74.125.131.104
  ⊞ www.google.com: type A, class IN, addr 74.125.131.106
  ⊞ www.google.com: type A, class IN, addr 74.125.131.99
  ⊞ www.google.com: type A, class IN, addr 74.125.131.105
```


Pic: https://en.wikipedia.org/wiki/Domain_Name_System

Pic: https://www.catchpoint.com/blog/how-dns-works

Authority

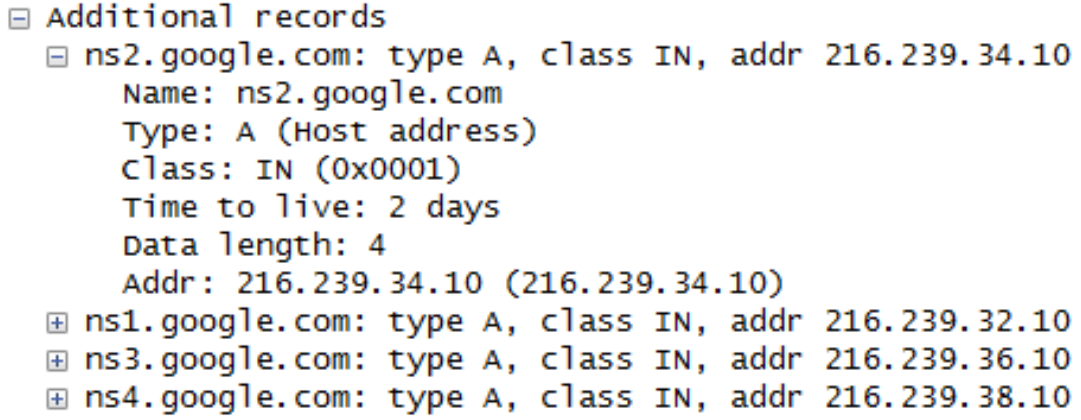
- If resolver doesn't know the answer, it responds with a name server that it knows is authoritative for the name or part of the name

```
⊟ Domain Name System (response)
    [Request In: 3]
    [Time: 0.014981000 seconds]
    Transaction ID: 0xccf9
  ⊞ Flags: 0x8000 Standard query response, No error
    Questions: 1
    Answer RRs: 0
    Authority RRs: 4
    Additional RRs: 4
  ⊞ Queries
  ⊟ Authoritative nameservers
    ⊟ google.com: type NS, class IN, ns ns2.google.com
        Name: google.com
        Type: NS (Authoritative name server)
        Class: IN (0x0001)
        Time to live: 2 days
        Data length: 6
        Name Server: ns2.google.com
    ⊞ google.com: type NS, class IN, ns ns1.google.com
    ⊞ google.com: type NS, class IN, ns ns3.google.com
    ⊞ google.com: type NS, class IN, ns ns4.google.com
  ⊞ Additional records
```


Pic: https://www.catchpoint.com/blog/how-dns-works

Additional

- To help prevent further lookups, if the resolver can resolve the address of the nameserver in the authority record, it includes as an additional record



Pic: https://www.catchpoint.com/blog/how-dns-works

University of Colorado **Boulder**

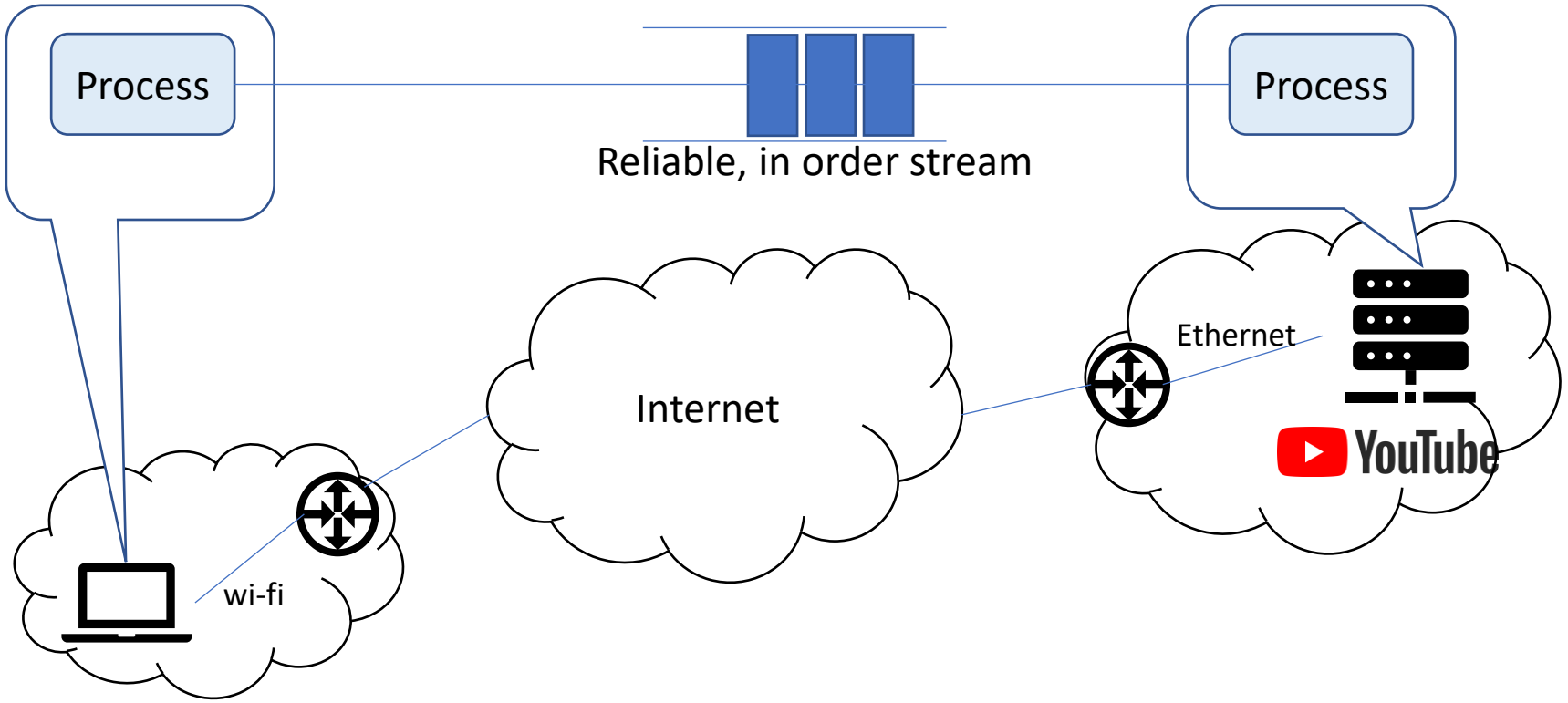
Socket Programming

Course: Networking Fundamentals
Module: Application

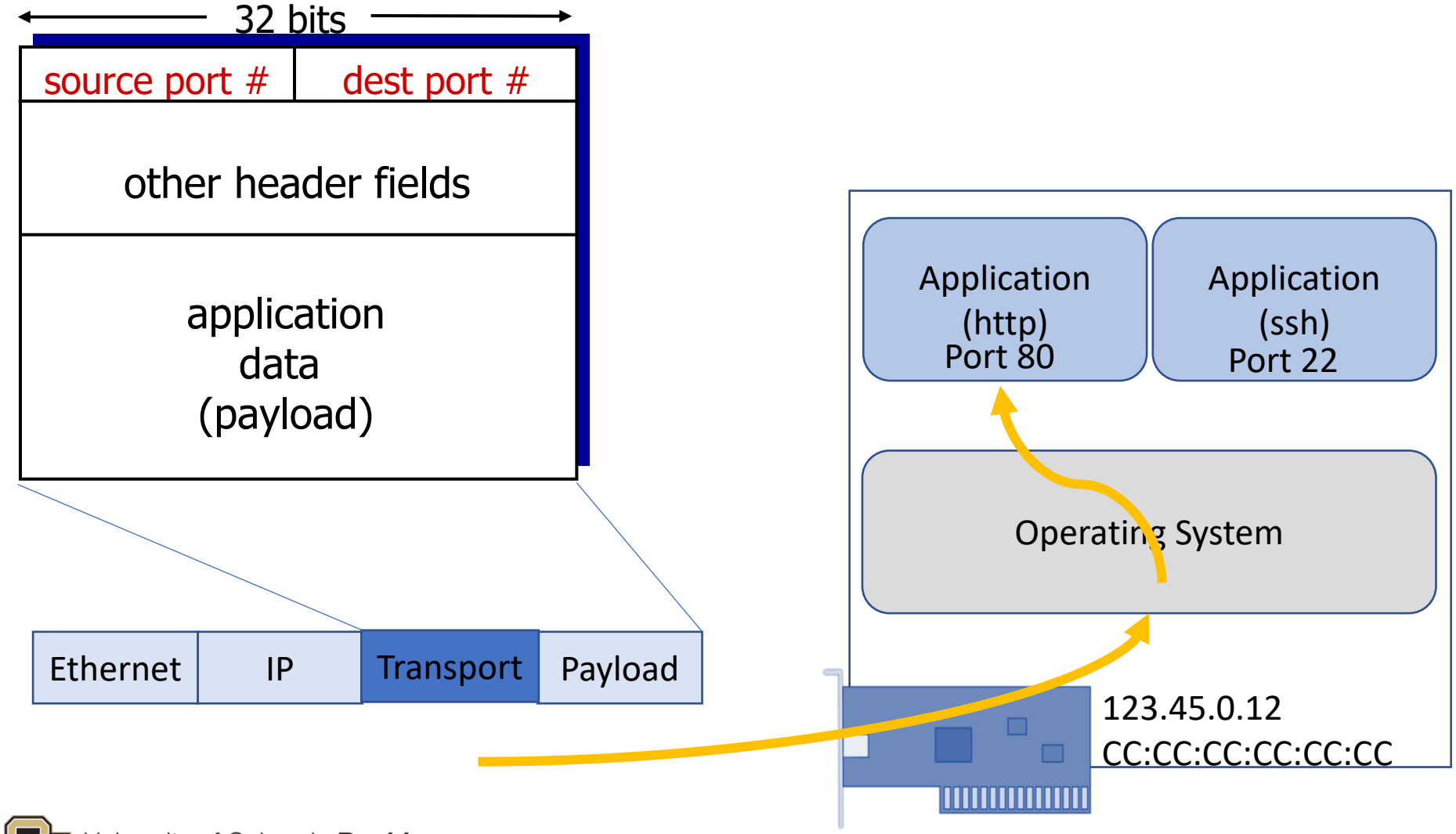
University of Colorado **Boulder**

Transport Layer

- Transport layer enables process-to-process, and reliable in-order stream

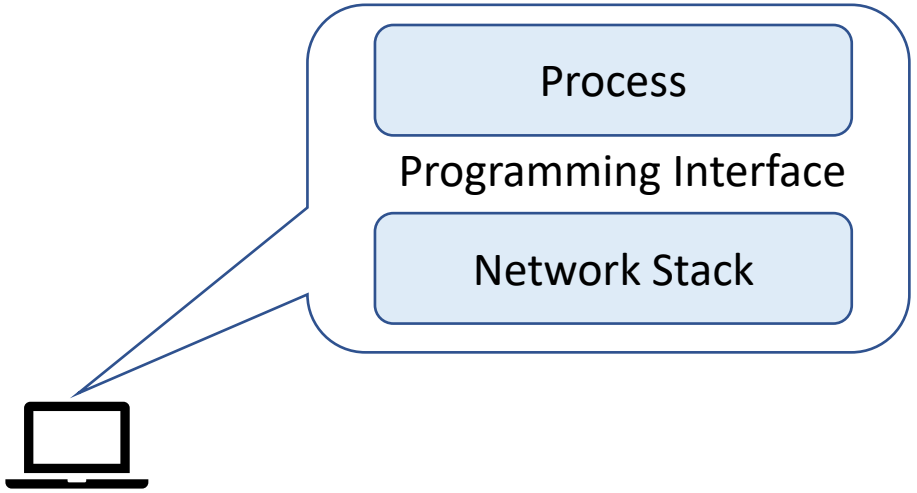


Transport Addressing - Ports



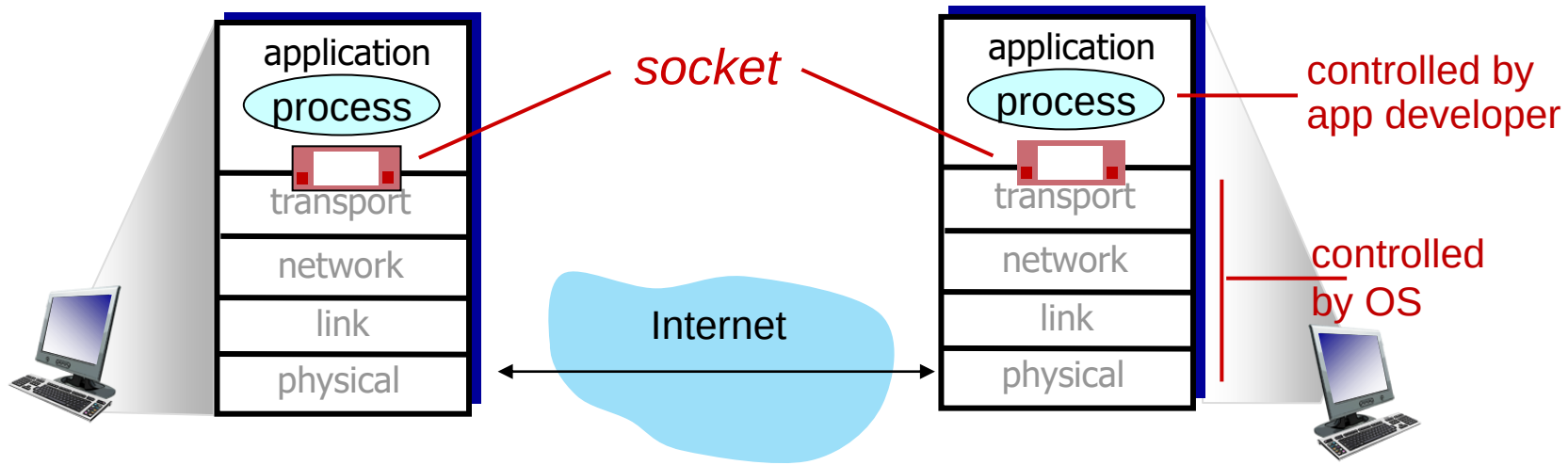
Problem 2: Programming

- Link, Network, and Transport layer protocols are concepts
  Operating system network stack is their implementation
- How do we write programs to interface to this network stack?

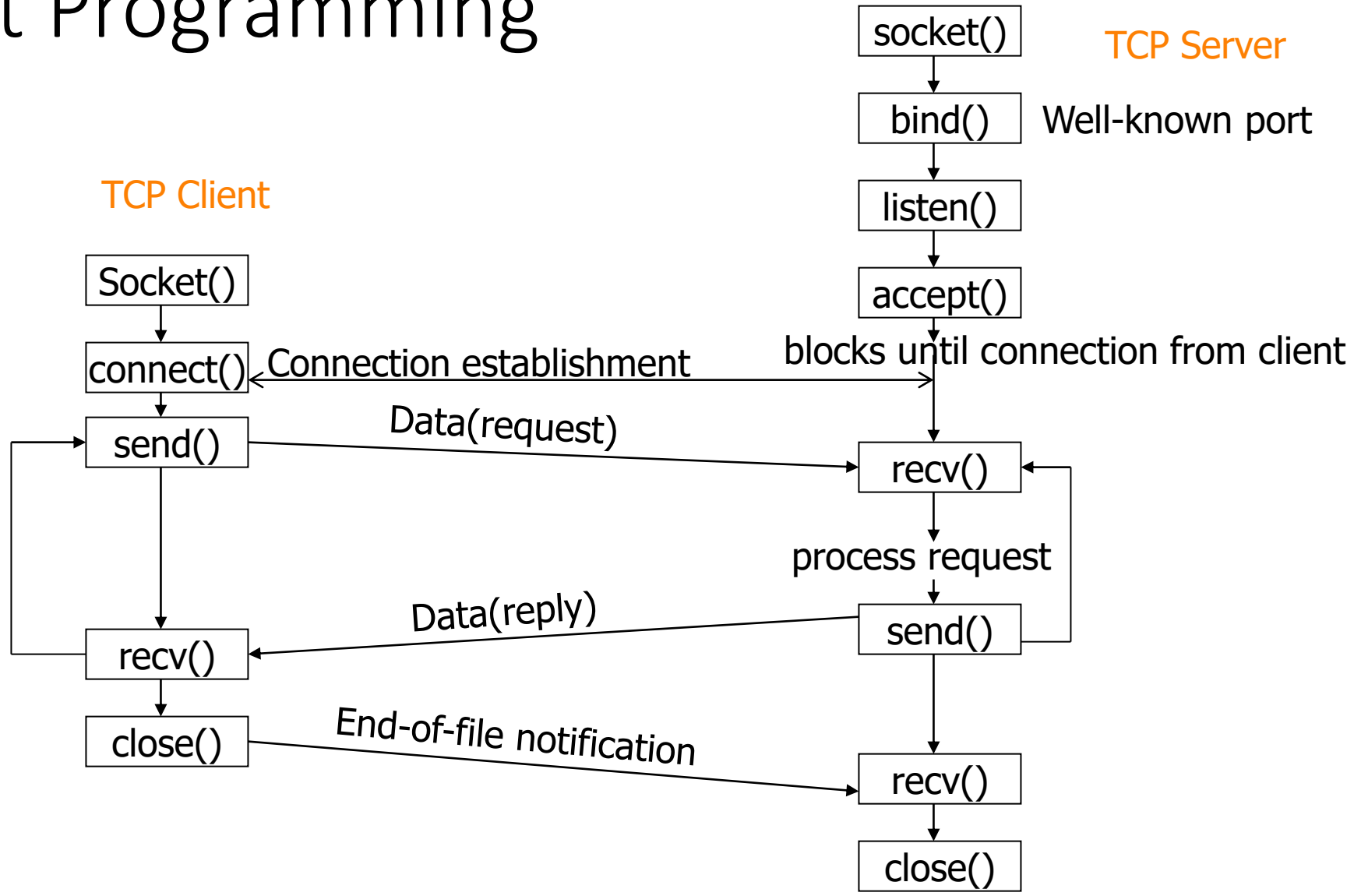


Berkeley Sockets

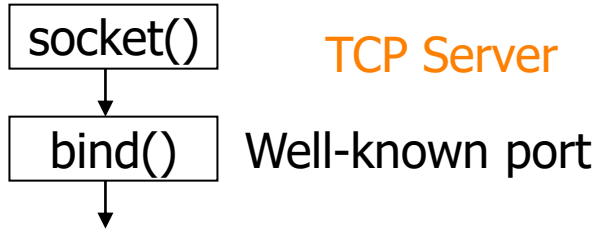
- Originated in 1983 and has been the standard since
- A socket is an abstract representation for the local endpoint of a network communication path.
- App. puts data into socket, other app. gets data from the socket.



Socket Programming



Socket Programming



socket() - OS will create a socket and return a handle (file descriptor)

```
int socket(int domain, int type, int protocol);
```

PF_INET

PF_INET6

SOCK_STREAM (TCP)

SOCK_DGRAM (UDP)

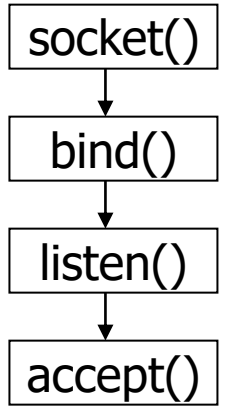
bind() – tells the OS what address to use

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

IP address,
port number

https://beej.us/guide/bgnet/html/

Socket Programming



TCP Server

Well-known port

listen() – notifies OS the willingness to accept incoming connections on this socket

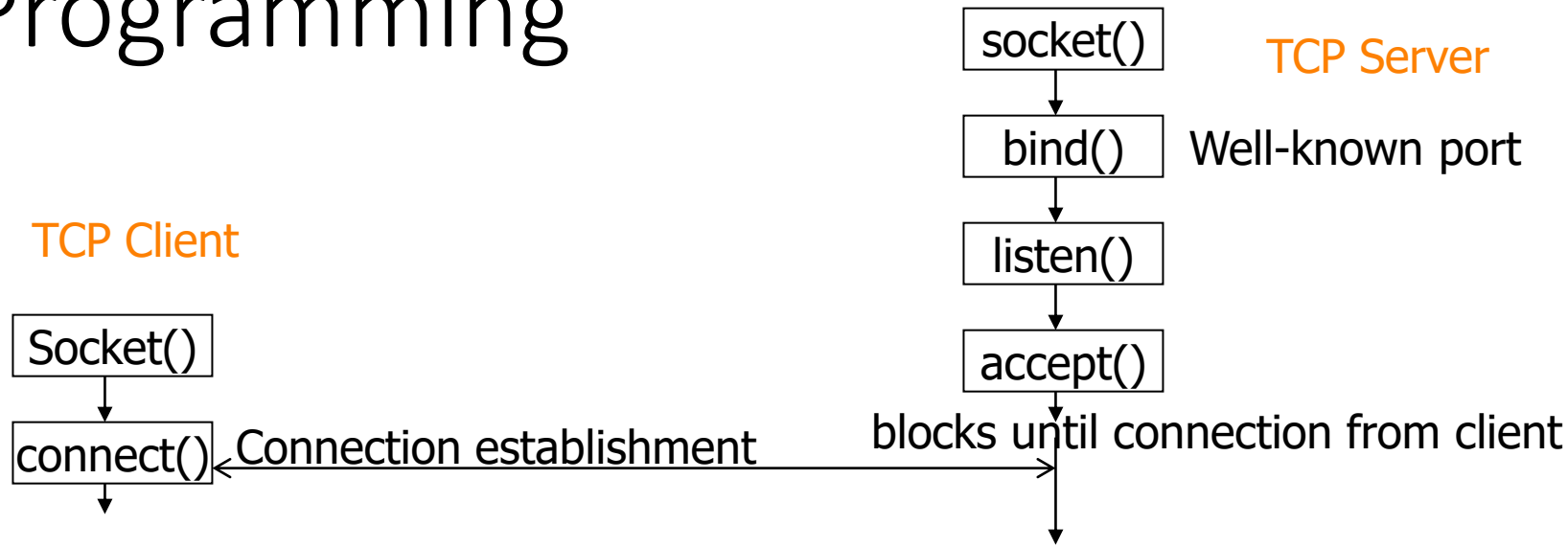
```
int listen(int sockfd, int backlog);
```

accept() – blocks waiting for connections. Sets the address of the incoming connection.

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```


https://beej.us/guide/bgnet/html/

Socket Programming



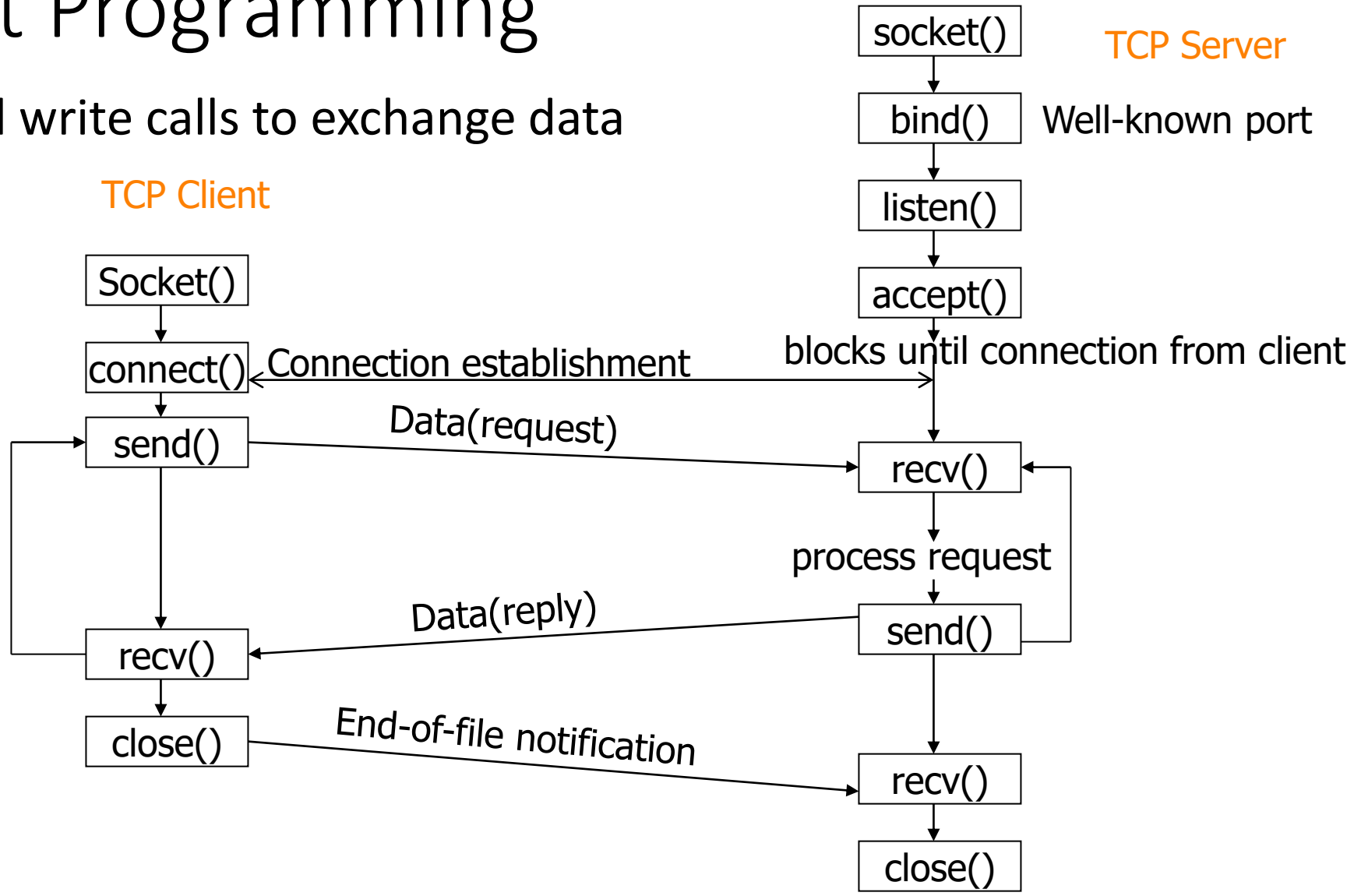
connect() – on the client side, tell OS to initiate connection to a specific address.

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```


https://beej.us/guide/bgnet/html/

Socket Programming

- Then read and write calls to exchange data



Full Server in Python

```
def server_program():
    # get the hostname
    host = socket.gethostname()
    port = 5000  # initiate port no above 1024

    server_socket = socket.socket()  # get instance
    # look closely. The bind() function takes tuple as argument
    server_socket.bind((host, port))  # bind host address and port together

    # configure how many client the server can listen simultaneously
    server_socket.listen(2)
    conn, address = server_socket.accept()  # accept new connection
    print("Connection from: " + str(address))
    while True:
        # receive data stream. it won't accept data packet greater than 1024 bytes
        data = conn.recv(1024).decode()
        if not data:
            # if data is not received break
            break
        print("from connected user: " + str(data))
        data = input(' -> ')
        conn.send(data.encode())  # send data to the client

    conn.close()  # close the connection
```

https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client

Full Client in Python

```
def client_program():
    host = socket.gethostname()  # as both code is running on same pc
    port = 5000  # socket server port number

    client_socket = socket.socket()  # instantiate
    client_socket.connect((host, port))  # connect to the server

    message = input(" -> ")  # take input

    while message.lower().strip() != 'bye':
        client_socket.send(message.encode())  # send message
        data = client_socket.recv(1024).decode()  # receive response

        print('Received from server: ' + data)  # show in terminal

        message = input(" -> ")  # again take input

    client_socket.close()  # close the connection
```

https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client

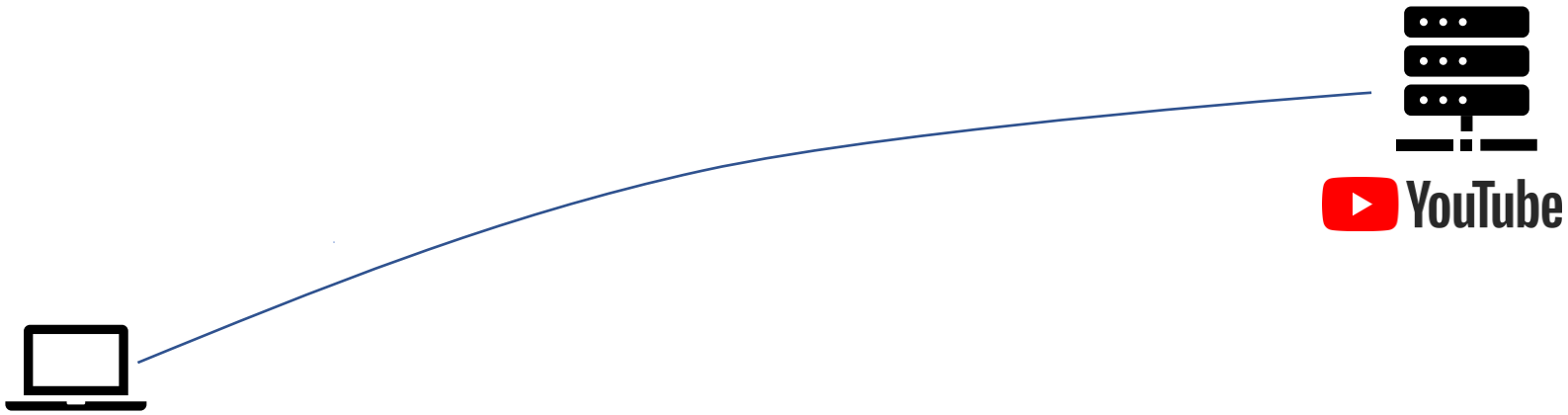
University of Colorado **Boulder**

Application Protocol: HTTP

Course: Networking Fundamentals
Module: Application

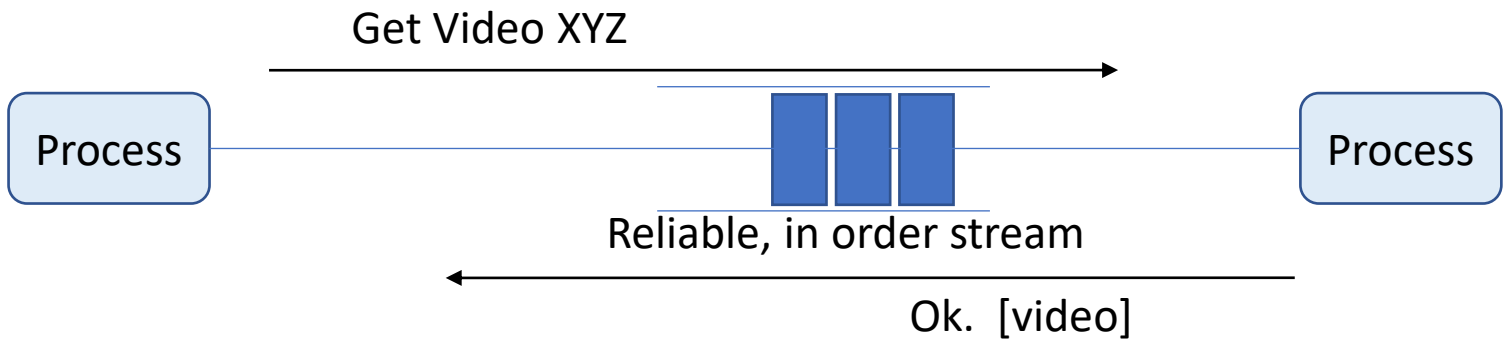
University of Colorado **Boulder**

Visiting Youtube (or any other service)



Problem 3: Application-level protocol

- How should the processes communicate what they want (e.g., with youtube – I can watch a video, or I can upload a video)
- How do processes encode data

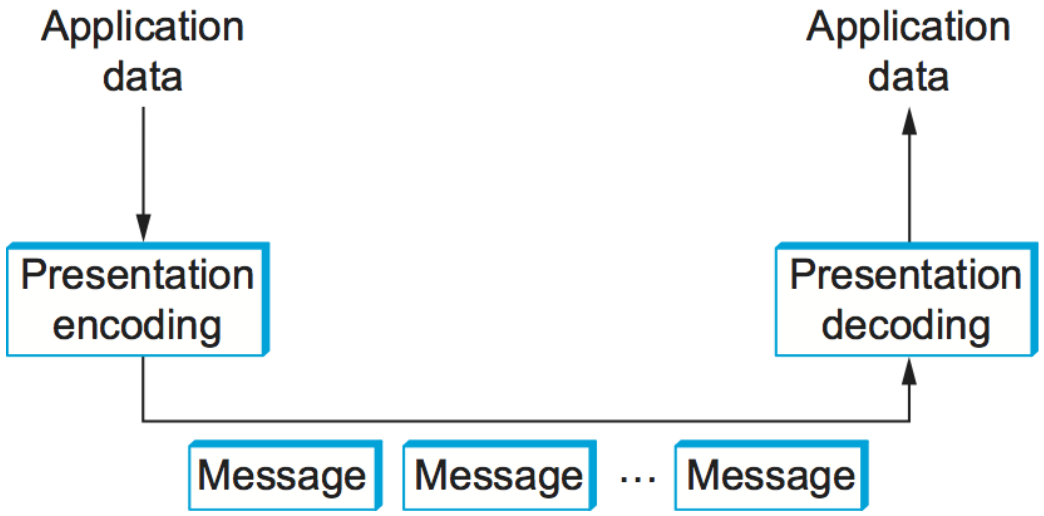


An Application Layer Protocol Defines

- Message syntax – what fields, how fields are delineated
- Types of messages exchanged – e.g., request, response
- Message semantics – meaning of information in fields

Message Syntax

- Requirement: Receiver needs to be able to extract the same message as what the transmitter sent
- Consideration: goal of approach – debuggability, bandwidth, processing

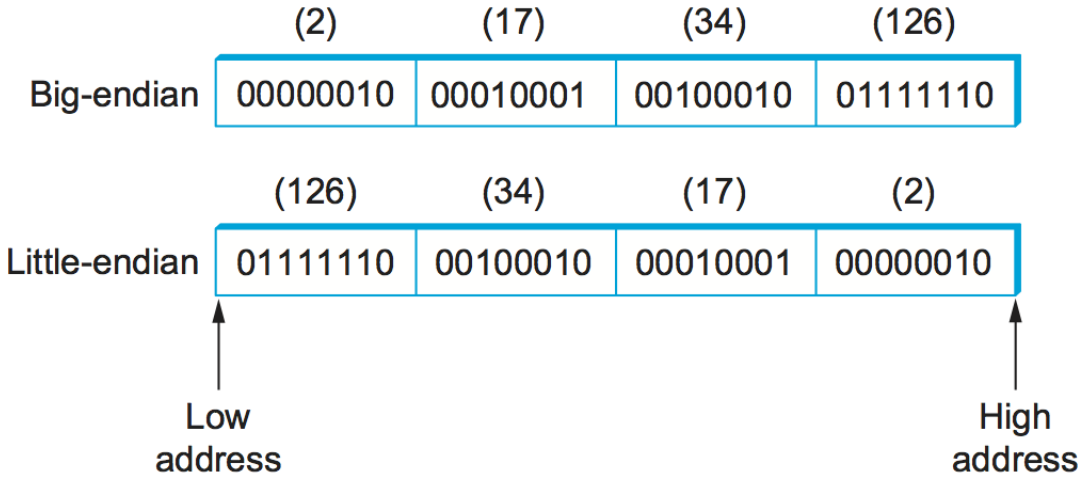


Next: Taxonomy:
- Data Types
- Conversion Strategy
- Tagging

https://book.systemsapproach.org/data/presentation.html

Data Types

- Base Types
  - E.g., Integer
  - Concerns - Number of bits? Order of bytes?
- Flat types
  - E.g., array or structure
  - Concerns - Padding? Length?
- Complex types
  - E.g., tree or linked list (has pointers)
  - Concerns – must serialize / flatten



https://book.systemsapproach.org/data/presentation.html

Conversion Strategy

- Canonical Intermediate Form
  - Sender converts its internal representation to some agreed upon format
  - Receiver converts received data into its internal representation
- Receiver Makes Right
  - Sender transmits data in its internal representation
  - Includes information about representation
  - Receiver converts from the sender's representation to its own representation (if needed)
  - Works well if assumption is a homogenous infrastructure

Tagging

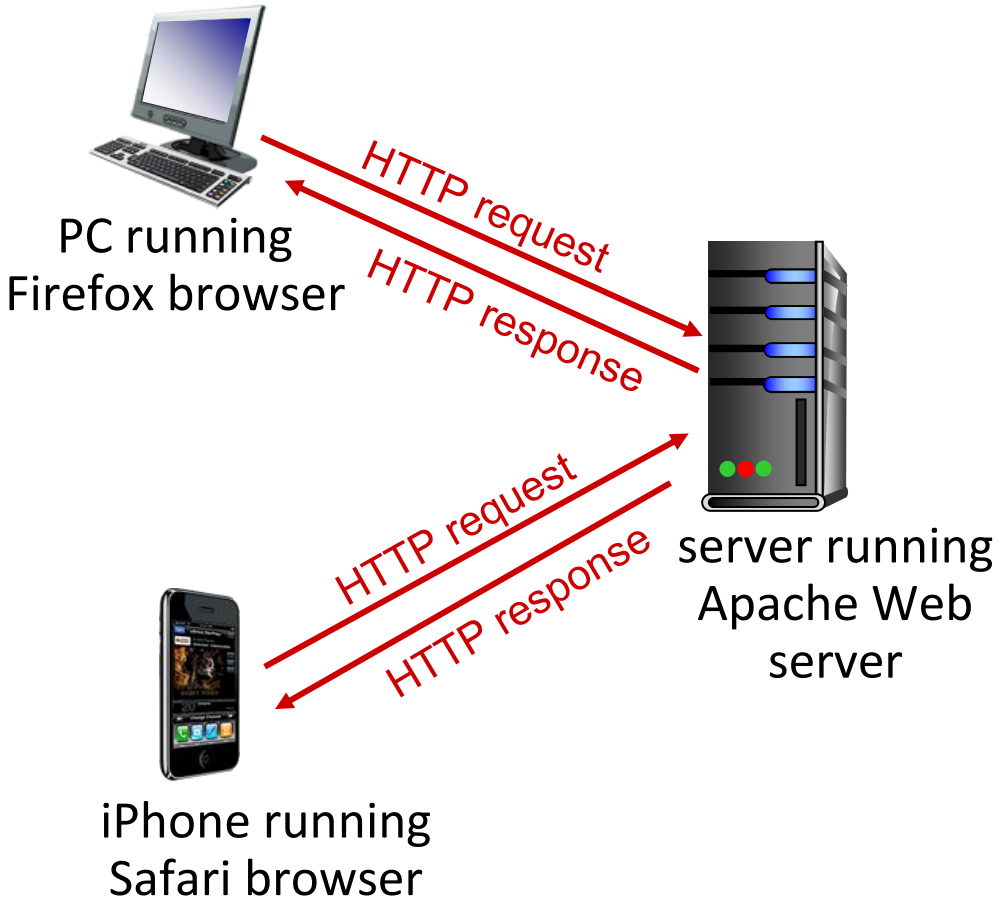
- Untagged
  - Agree on type, length, and location of data
- Tagged
  - Include in message tags about the data (e.g., type and length)
  - We'll discuss JSON and Protobufs

Example Protocol - HTTP

- Web's application-layer protocol
- Client / Server exchange Web Objects
  - Html, jpeg, audio file
- Stateless – server doesn't retain information about past client requests

`http://www.example.com/blah/blah.gif`

proto host name path



HTTP Message Format

START_LINE <CRLF>

MESSAGE_HEADER <CRLF>

<CRLF>

MESSAGE_BODY <CRLF>

Request

```
GET /online/ HTTP/1.1
Host: majesticsublimesilversecret.neverssl.com
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 07 Jul 2023 20:16:29 GMT
Server: Apache/2.4.57 ()
Upgrade: h2,h2c
Connection: Upgrade, Keep-Alive
Last-Modified: Wed, 29 Jun 2022 00:23:22 GMT
ETag: "8be-5e28b29291e10-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1173
Keep-Alive: timeout=5, max=100
Content-Type: text/html; charset=UTF-8

<html>
        <head>
                <title>NeverSSL - helping you

                <style>
                body {
                        font-family: Montserra
                        font-size: 16x;
                        color: #444444;
                        margin: 0;
```

HTTP Message types

Operation	Description
OPTIONS	Request information about available options
GET	Retrieve document identified in URL
HEAD	Retrieve metainformation about document identified in URL
POST	Give information (e.g., annotation) to server
PUT	Store document under specified URL
DELETE	Delete specified URL
TRACE	Loopback request message
CONNECT	For use by proxies

HTTP data format examples

HTML – hypertext markup language

Good for describing appearance (Webpage)

Content-Type: text/html

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

JSON – JavaScript Object Notation

Good for passing info (APIs)

Content-Type: application/json

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```


https://www.w3schools.com/

Quick Overview of JSON

- Data is in name : value pairs
  - Key – String
  - Value – string, number, object, array, boolean, null
- Curly braces hold objects (list of name:value pairs)
- Square brackets hold array (list of values)

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Example JSON


```
{
"TopQBs":[
        {"rank":1, "player":{"first":"Jalen", "last":"Hurts"}},
        {"rank":2, "player":{"first":"Daniel", "last":"Jones"}},
        {"rank":3, "player":{"first":"Dak", "last":"Prescott"}}  ],
"src ":"Eric"
}
```

Example – top level object name:value pairs


```
{
"TopQBs":[
        {"rank":1, "player":{"first":"Jalen", "last":"Hurts"}},
        {"rank":2, "player":{"first":"Daniel", "last":"Jones"}},
        {"rank":3, "player":{"first":"Dak", "last":"Prescott"}}  ],
"src":"Eric"
}
```

Example JSON – Array Element

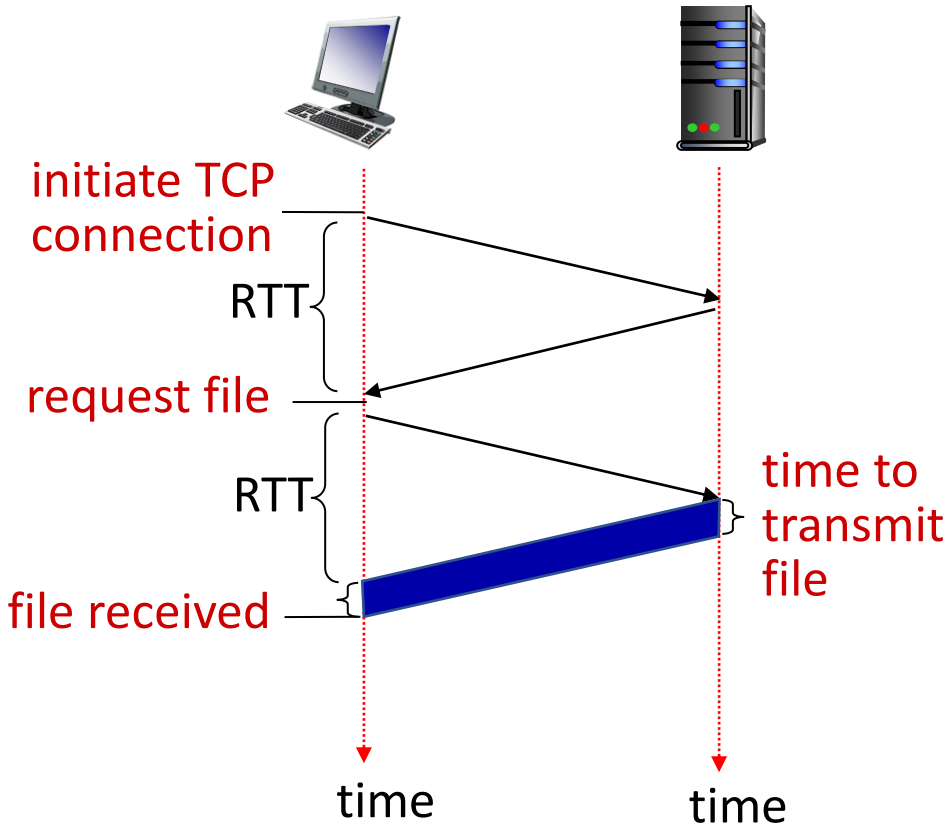

```
{
"TopQBs":[
        {"rank":1, "player":{"first":"Jalen", "last":"Hurts"}},
        {"rank":2, "player":{"first":"Daniel", "last":"Jones"}},
        {"rank":3, "player":{"first":"Dak", "last":"Prescott"}}  ],
"src":"Eric"
}
```

Example JSON – Array Item Object


```
{
"TopQBs":[
        {"rank":1, "player":{"first":"Jalen", "last":"Hurts"}},
        {"rank":2, "player":{"first":"Daniel", "last":"Jones"}},
        {"rank":3, "player":{"first":"Dak", "last":"Prescott"}}  ],
"src":"Eric"
}
```

HTTP versions

Version	
1.0	First major version
1.1	Persistent Connections
2.0	Multiplexed Requests
3.0	HTTP over QUIC (instead of TCP)



University of Colorado **Boulder**

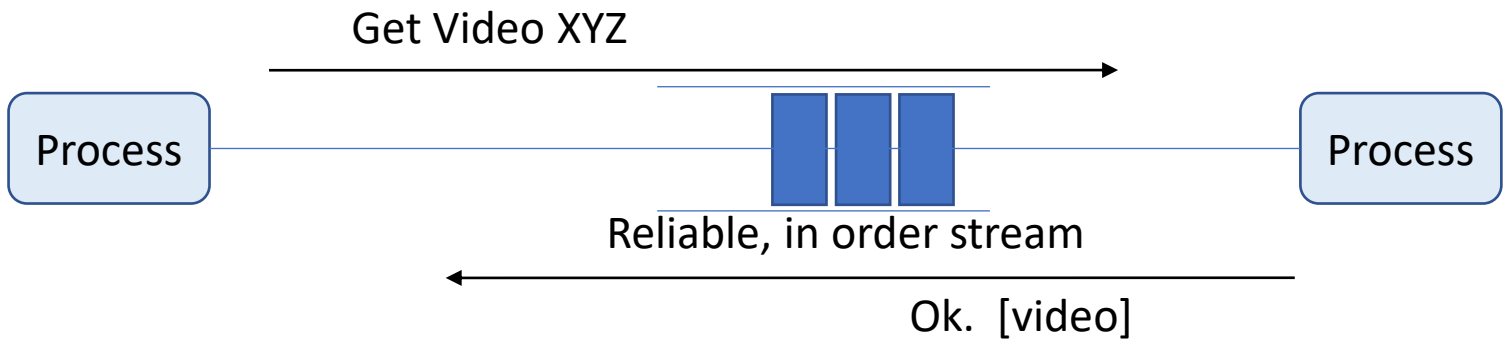
Application Protocol: gRPC

Course: Networking Fundamentals
Module: Application

University of Colorado **Boulder**

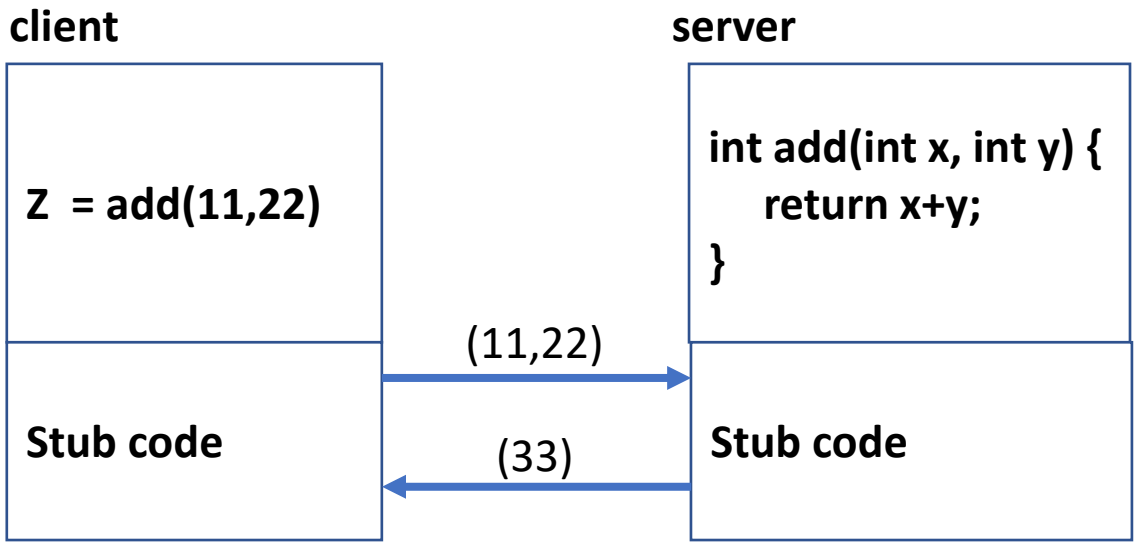
Problem 3: Application-level protocol

- What should the processes communicate what they want (e.g., with youtube – I can watch a video, or I can upload a video)
- How do processes encode data

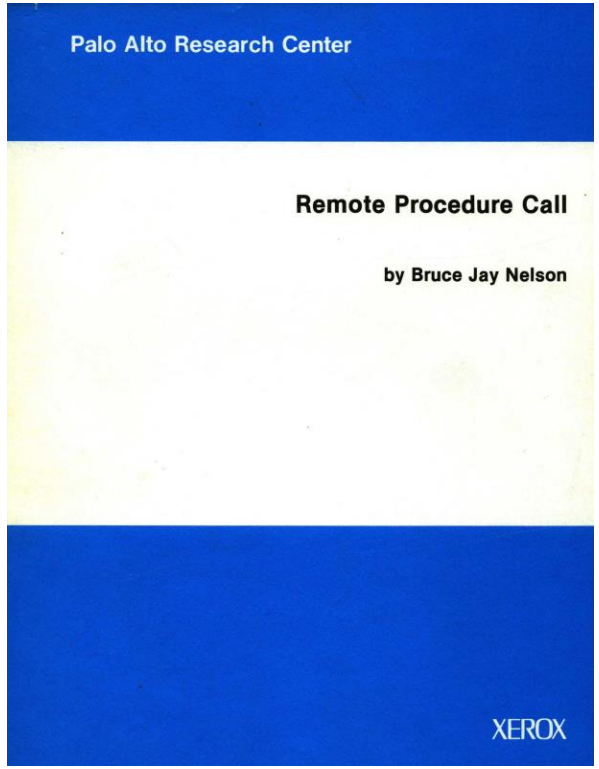


Remote Procedure Call (RPC) - 1981

- Extends the idea of local procedure call to a remote server
- Defined framework for data encoding, and communication protocol

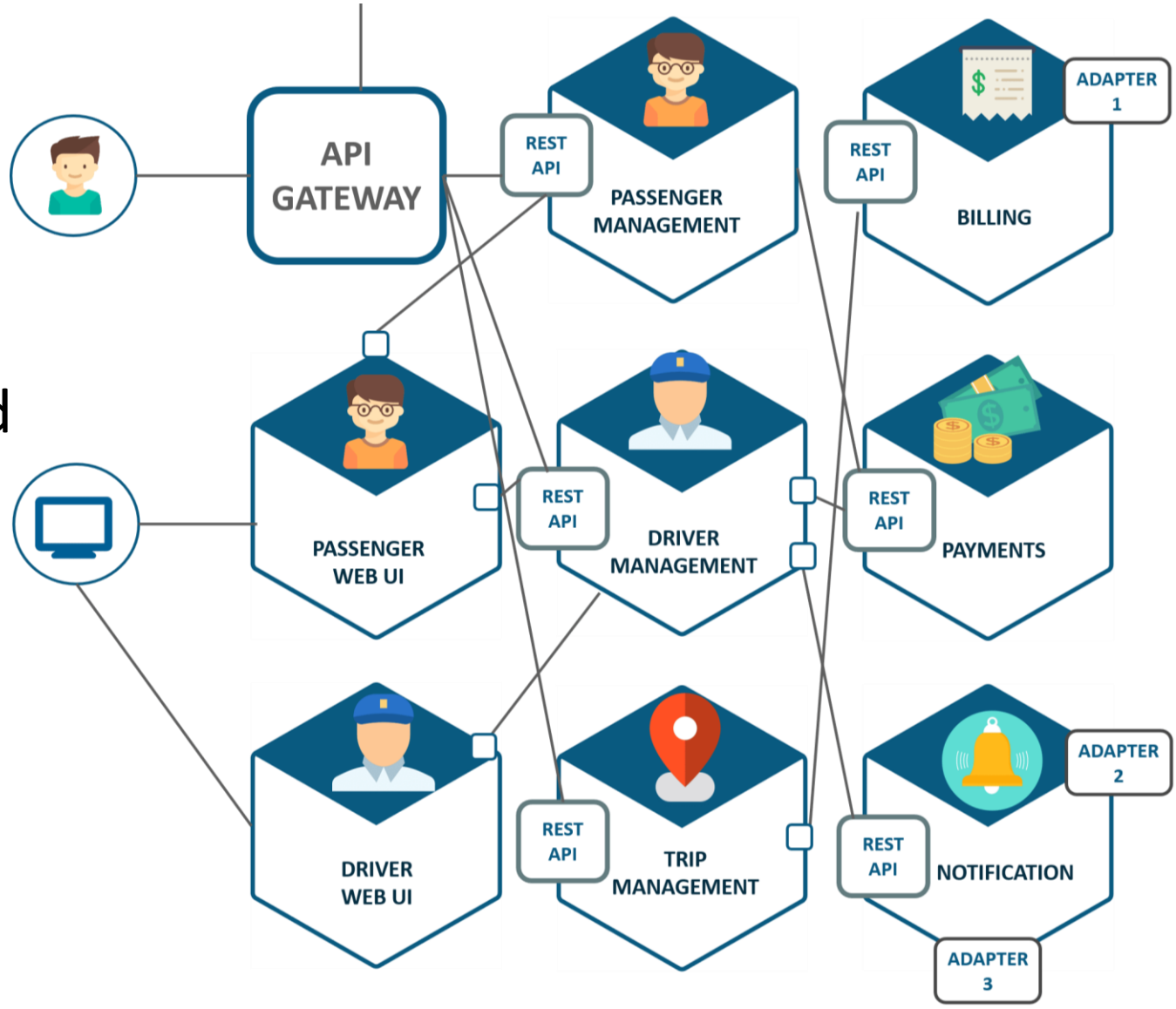


1981



Useful in Modern Web Services

- Services are partitioned into smaller services, each with APIs
- e.g., Ride share service on right
- Could be REST (with JSON), but could also use gRPC as the protocol



gRPC – Modern Implementation

- Based on protobufs – binary format to serialize data
- Leverages HTTP 2 (persistent and pipelined messages)

JSON: 55 bytes

```
{
    "age": 35,
    "first_name": "Stephane",
    "last_name": "Maarek"
}
```

Same in Protocol Buffers: 20 bytes

```
message Person {
    int32 age = 1;
    string first_name = 2;
    string last_name = 3;
}
```

https://medium.com/@yangli907/grpc-learning-part-1-cdcf59e52707

Workflow

.proto

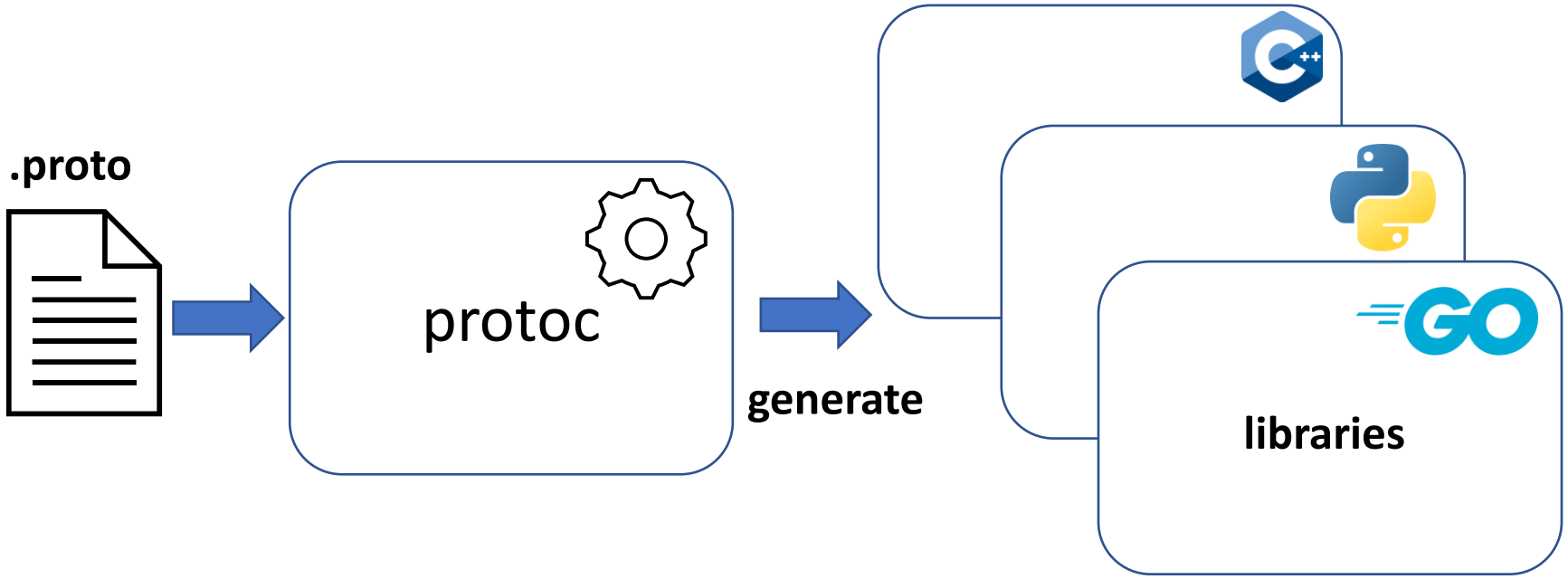
Create a .proto file

It specifies the structure of the data and defines the functions

```
service Echo {
 rpc echo(Message) returns (Message) {}
}

message Message{
 string message = 1;
}
```

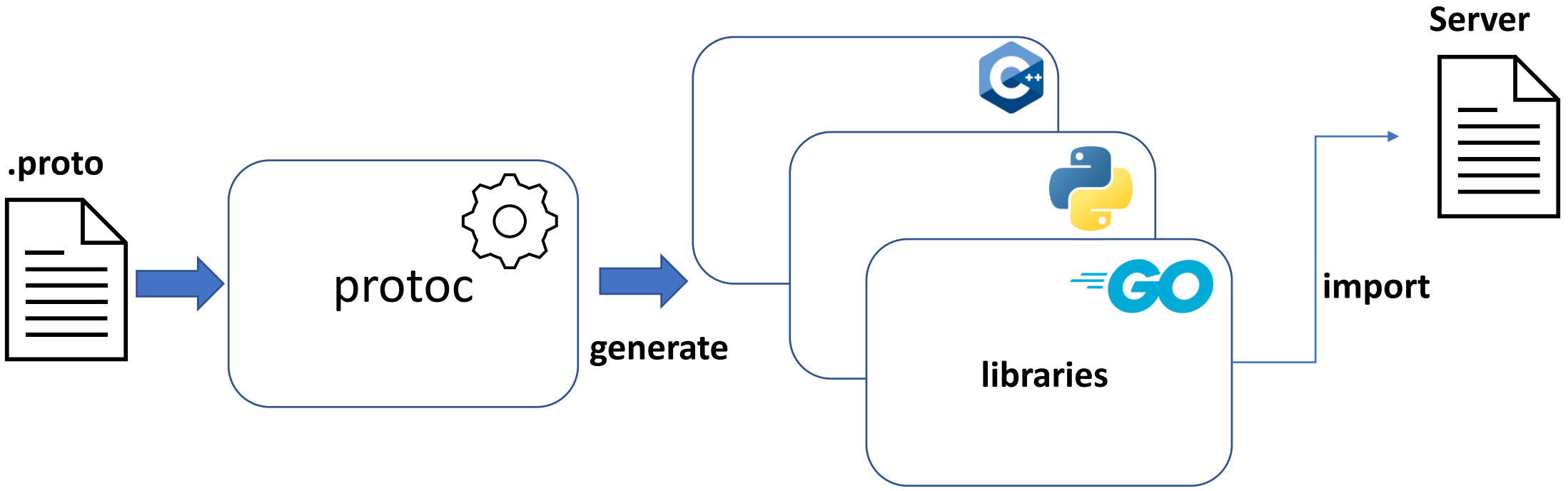
Workflow



Compiler generates libraries of the stubs (serializing/deserializing data)

Supports a number of different languages

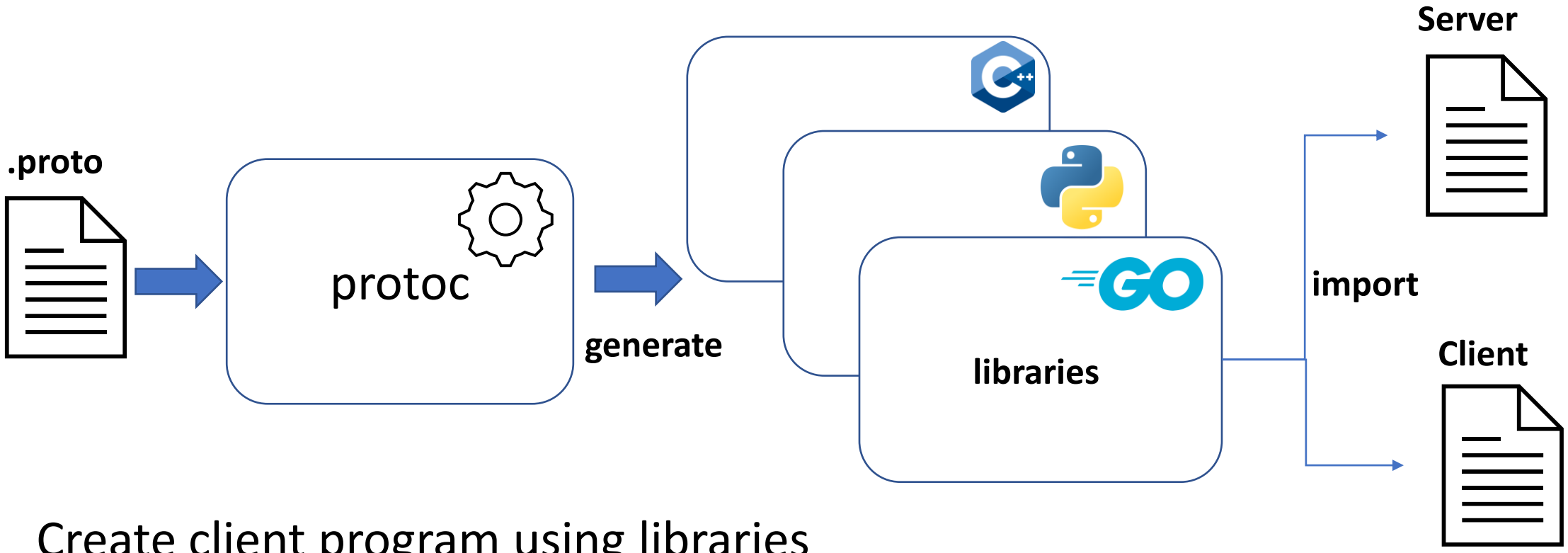
Workflow



Create server program using libraries

Define the implementation of the functions specified in the .proto

Workflow

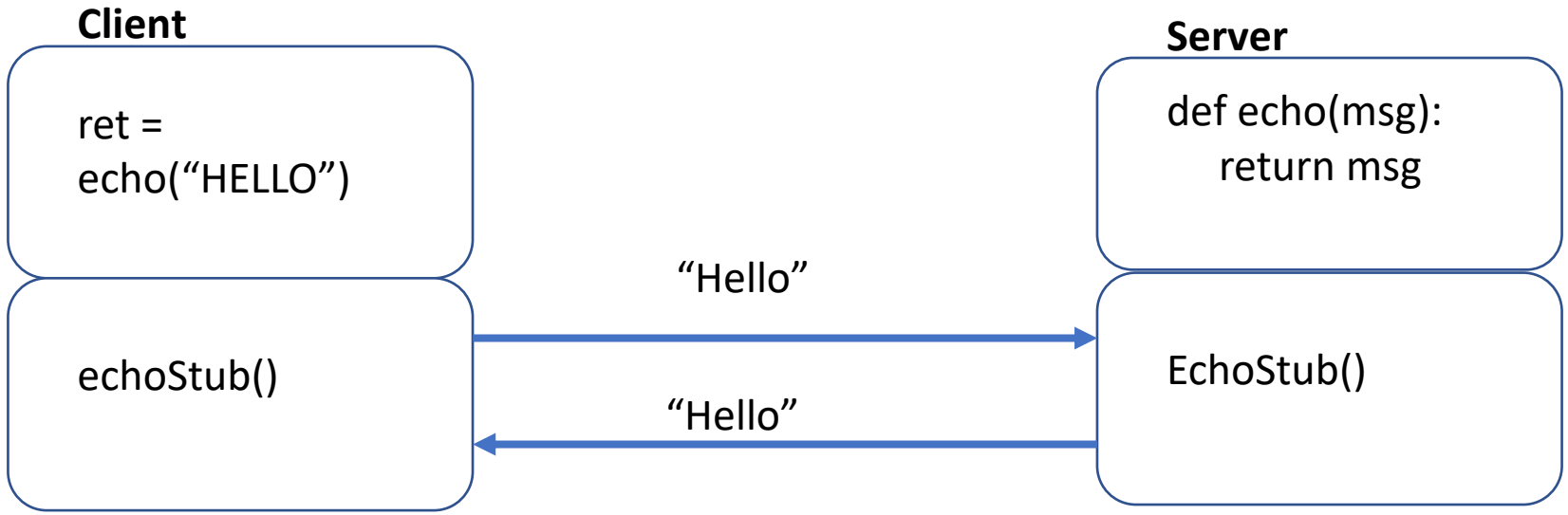


Create client program using libraries

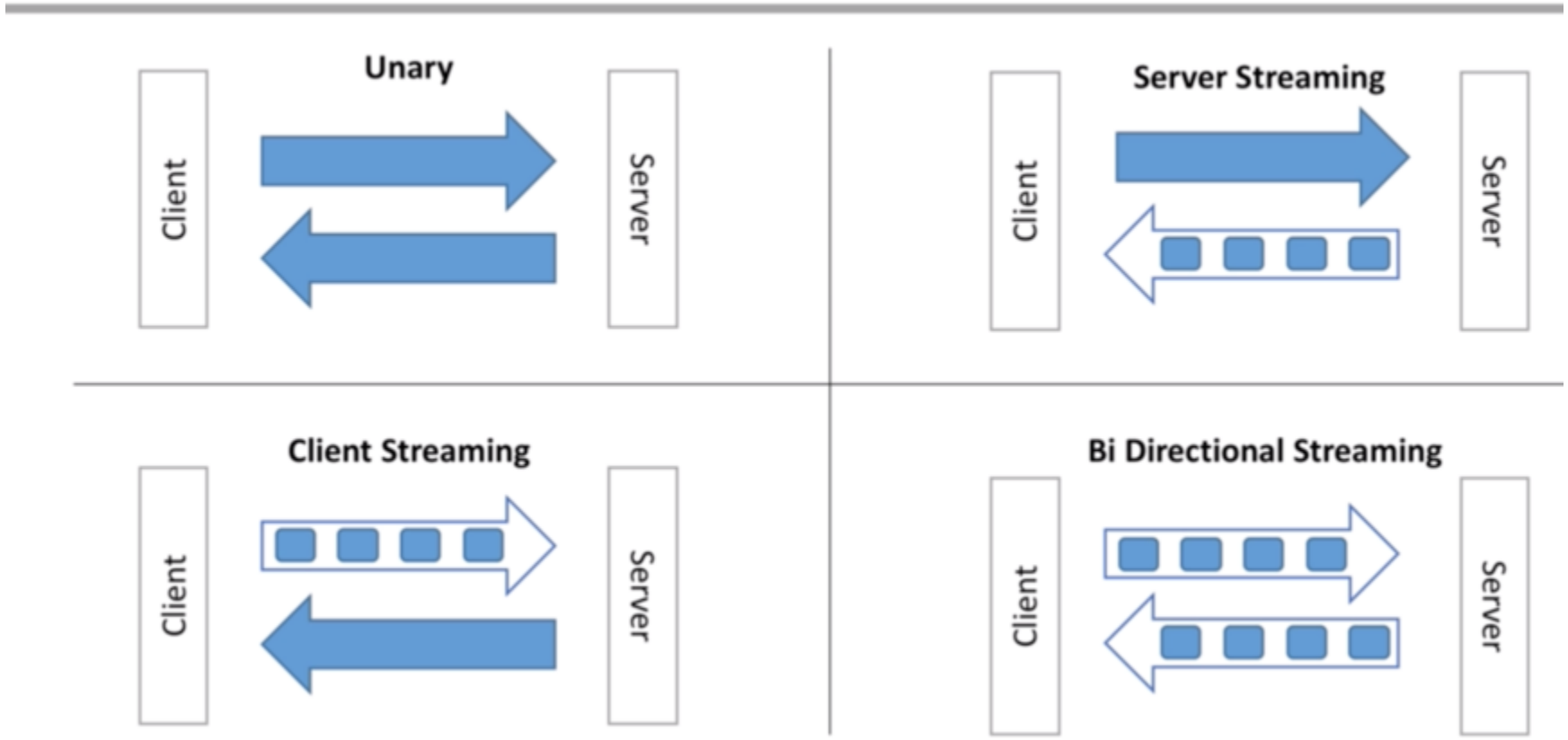
Make calls to the functions specified in the .proto

Walkthrough Example – Echo Client / Server

- Wrote: simple.proto, simple_server.py, simple_client.py
- Called: python -m grpc_tools.protoc --proto_path=. ./simple.proto --python_out=. --grpc_python_out=.
- Generated: simple_pb2.py, simple_pb2_grpc.py



4 Types of APIs in gRPC



https://medium.com/@yangli907/grpc-learning-part-1-cdcf59e52707

More Complex Example with Annotated Code

- Wrote: moderate.proto, moderate_server.py, moderate_client.py
- Called: python -m grpc_tools.protoc --proto_path=. ./moderate.proto --python_out=. --grpc_python_out=.
- Generated: moderate_pb2.py, moderate_pb2_grpc.py

- python3 ./moderate_server.py
- python3 ./moderate_client.py

University of Colorado **Boulder**