DASH, Transport Intro

Lecture 9

http://www.cs.rutgers.edu/~sn624/352-S22

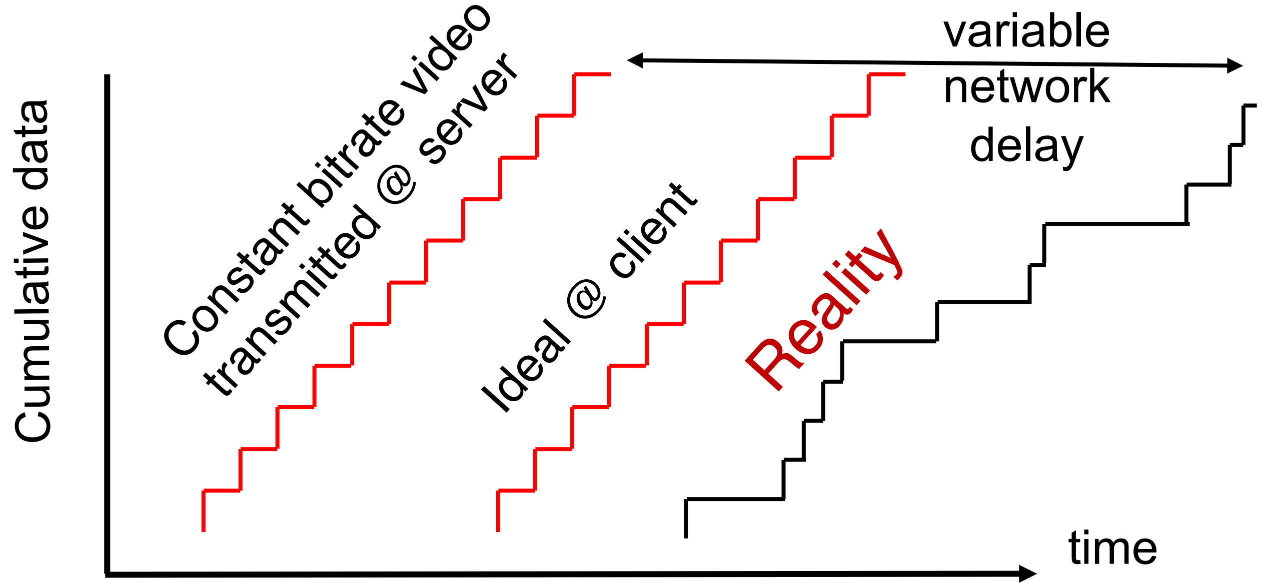
Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Quick recap of concepts

Video Bitrate

Bits played out per second (can vary over video's lifetime)



Buffer at the client to hold frames initially until playout delay t_p

Choosing t_p is hard! Don't know buffer fill rate apriori

Adaptive bit-rate selection

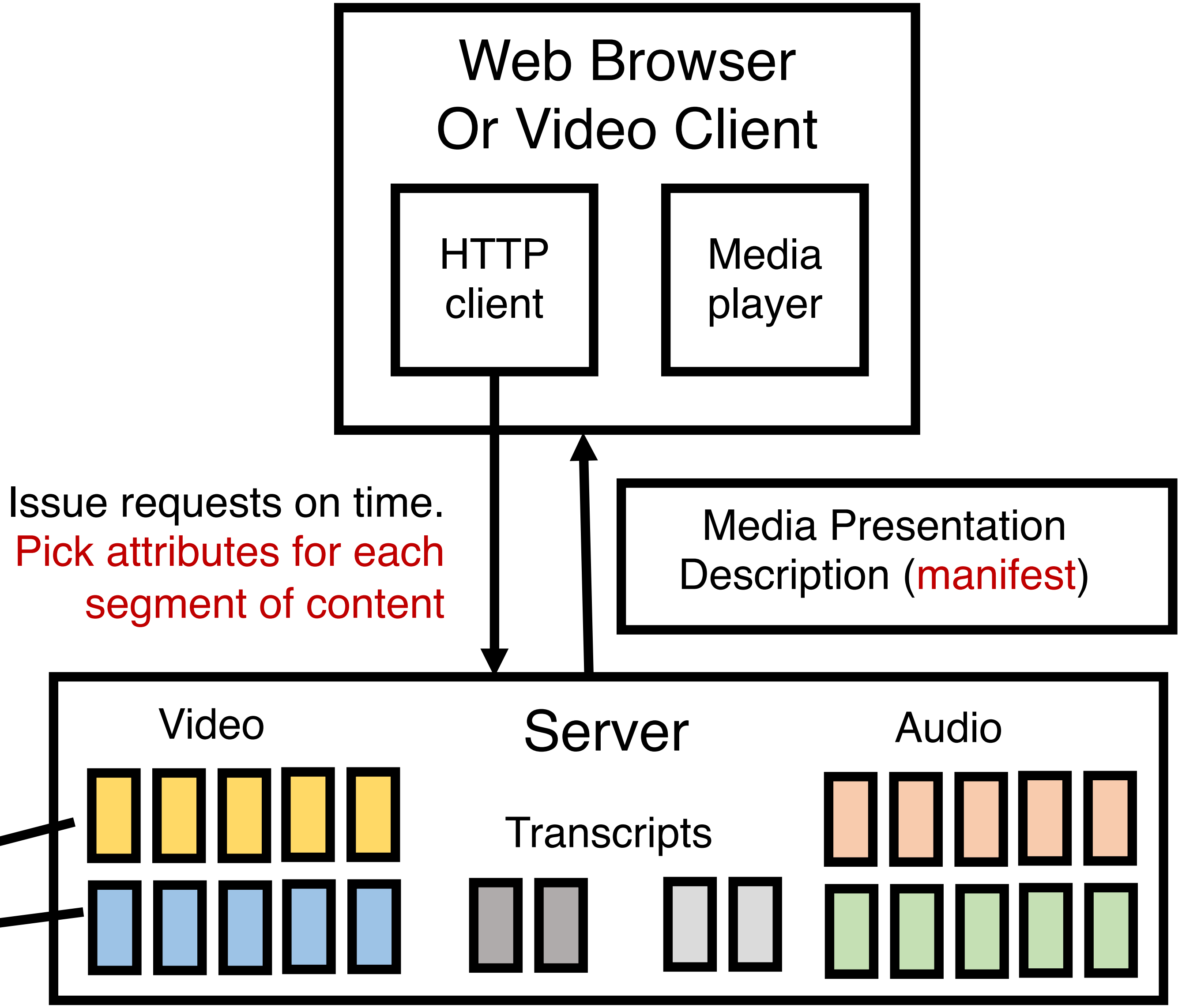
Dynamic Adaptive Streaming over HTTP (DASH)

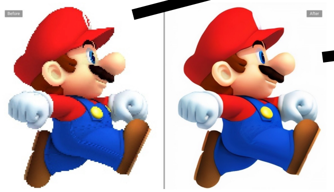
Streaming multimedia with DASH

- Dynamic Adaptive Streaming over HTTP
  - Used by Netflix and most popular video streaming services
- Adaptive: Perform video bit rate adaptation
  - It can be done on the client, or the server (with client feedback)
- Dynamic: Retrieve a single video from multiple sources
- The DASH video server is just a standard HTTP server
  - Provides video/audio content in multiple formats and encodings
- Leverage existing web-based infrastructure
  - DNS
  - CDNs!

DASH: Key ideas

- Content (video, audio, transcript, etc.) divided into segments (time)
- Algorithms to determine and request varying attributes (e.g., bitrate, language) for each segment
- Goal: ensure good quality of service, match user prefs, etc.





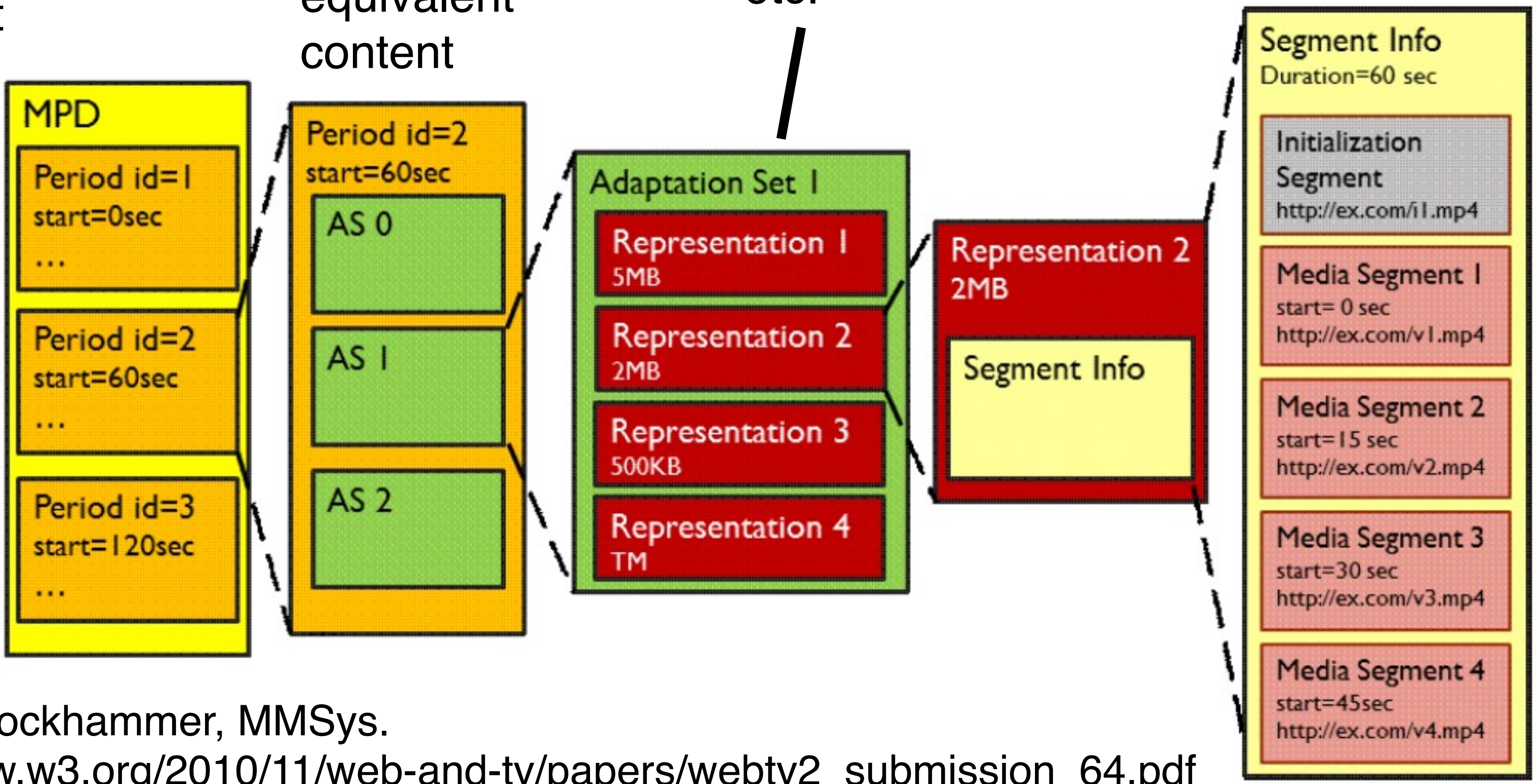
What does the manifest contain?

Periods: Durations of content

Adaptation set: functionally equivalent content

Representations: codecs, bit rates, etc.

Functionally equivalent: RSes of given AS
Functionally different: different ASes



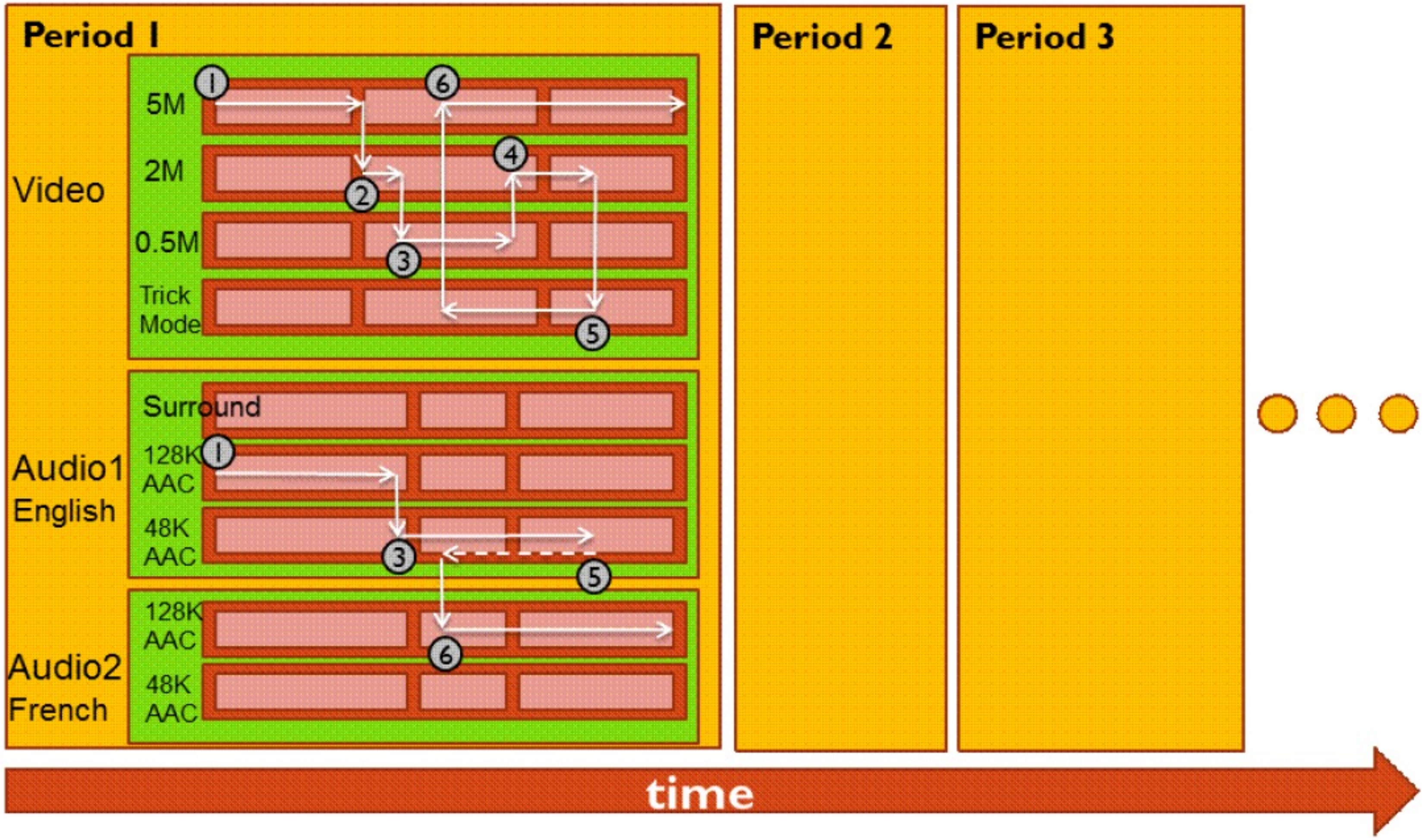
Multiple segments per representation

URL available for each segment

Byte ranges per segment (HTTP header for a range request)

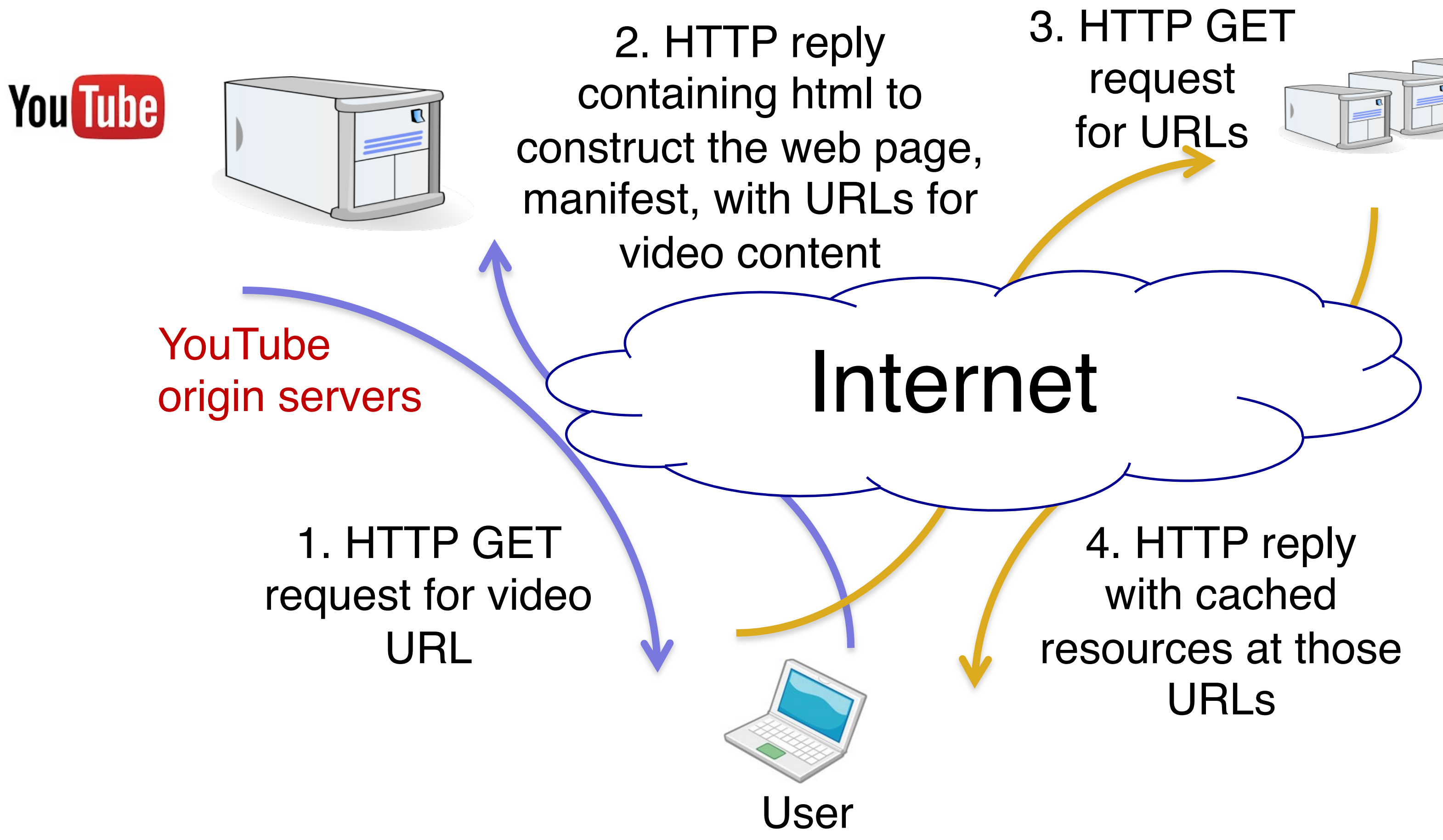
Source: Stockhammer, MMSys.
https://www.w3.org/2010/11/web-and-tv/papers/webtv2_submission_64.pdf

Dynamic changes in stream quality



Dynamic changes in stream location

- Just an HTTP request for an HTTP object



CDN DNS points user to best CDN server

Subtle: DNS granularity is per (sub)domain. Content from different (sub)domains can go to different CDN servers or origin

DASH reference player

- https://reference.dashif.org/dash.js/latest/samples/dash-if-reference-player/index.html

DASH Summary

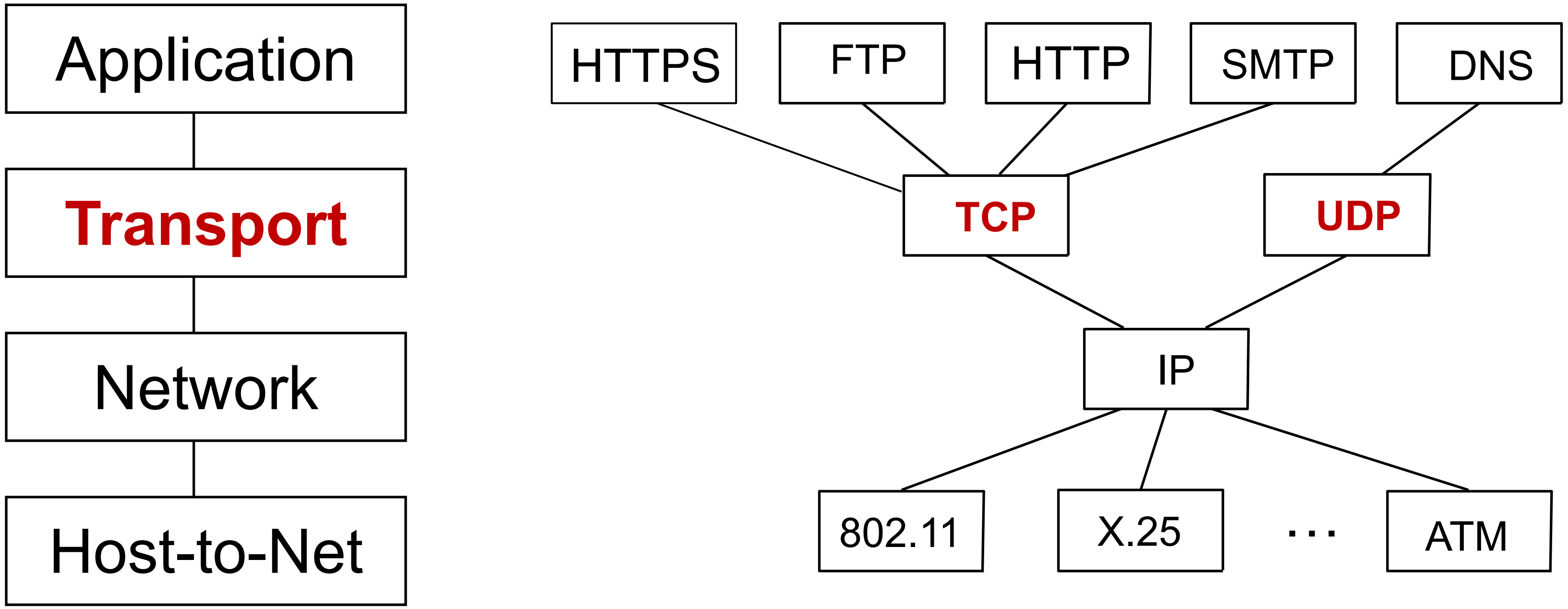
- Piggyback video on HTTP: widely used
- Enables independent HTTP requests per segment
  - Choose dynamic quality & preferences over time
  - Independent HTTP byte ranges
- Works well with CDNs
  - Fetch segments from locations other than the origin server
  - Fetch different segments from possibly different locations
- More resources on DASH
  - https://www.w3.org/2010/11/web-and-tv/papers/webtv2_submission_64.pdf
  - https://www.youtube.com/watch?v=xgowGnH5kUE

Application Layer: Wrap-up

- Name resolution, the web, mail, video
- Protocols built over the `socket()` abstraction
- Simple designs go a long way
    - Plain text protocols, header-based evolution, …
- Infrastructure for functionality, performance, …
    - Mail servers, CDNs, proxies, …
- Fit your apps to run on browsers: run almost anywhere (e.g. video)
- Apps are ultimately what users and most engineers care about
- BUT: if you don’t understand what’s under the hood, you risk bad design and poor performance for your Internet-facing applications

Transport

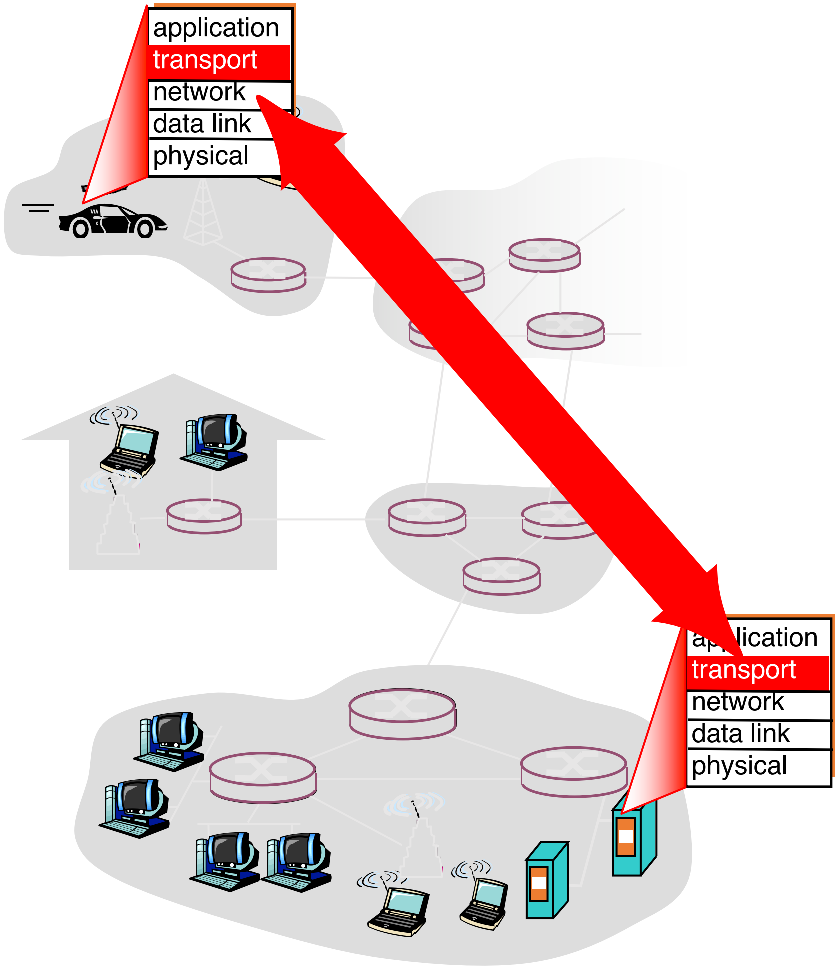
Transport


Application
Transport
Network
Host-to-Net
HTTPS
FTP
HTTP
SMTP
DNS
TCP
UDP
IP
802.11
X.25
...
ATM


Tp layer

Transport services and protocols

- Provide a communication abstraction between application processes
- Transport protocols run @ endpoints
  - send side: transport breaks app messages into segments, passes to network layer
  - recv side: reassembles segments into messages, passes to app layer
- Multiple transport protocols available to apps
  - Very popular in the Internet: TCP and UDP



Transport vs. network layer

- **Network layer:** abstraction to communicate between endpoints. Network layer provides best effort packet delivery to a remote endpoint.

- **Transport layer:** communication abstraction between processes. Delivers packets to the process.

Household analogy:

3 kids sending letters to 3 kids

- endpoints = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Alice and Bob who de/mux to in-house siblings
- network-layer protocol = postal service

Alice

Bob

Identifying a single conversation

- Application connections are identified by 4-tuple:

- Source IP address
- Source port
- Destination IP address
- Destination port

- In this analogy,

- Source address: the address of the first house
- Source port: name of a kid in the first house
- Destination address: the address of the second house
- Destination port: name of a kid in the second house

Demultiplexing Packets

Two popular transports

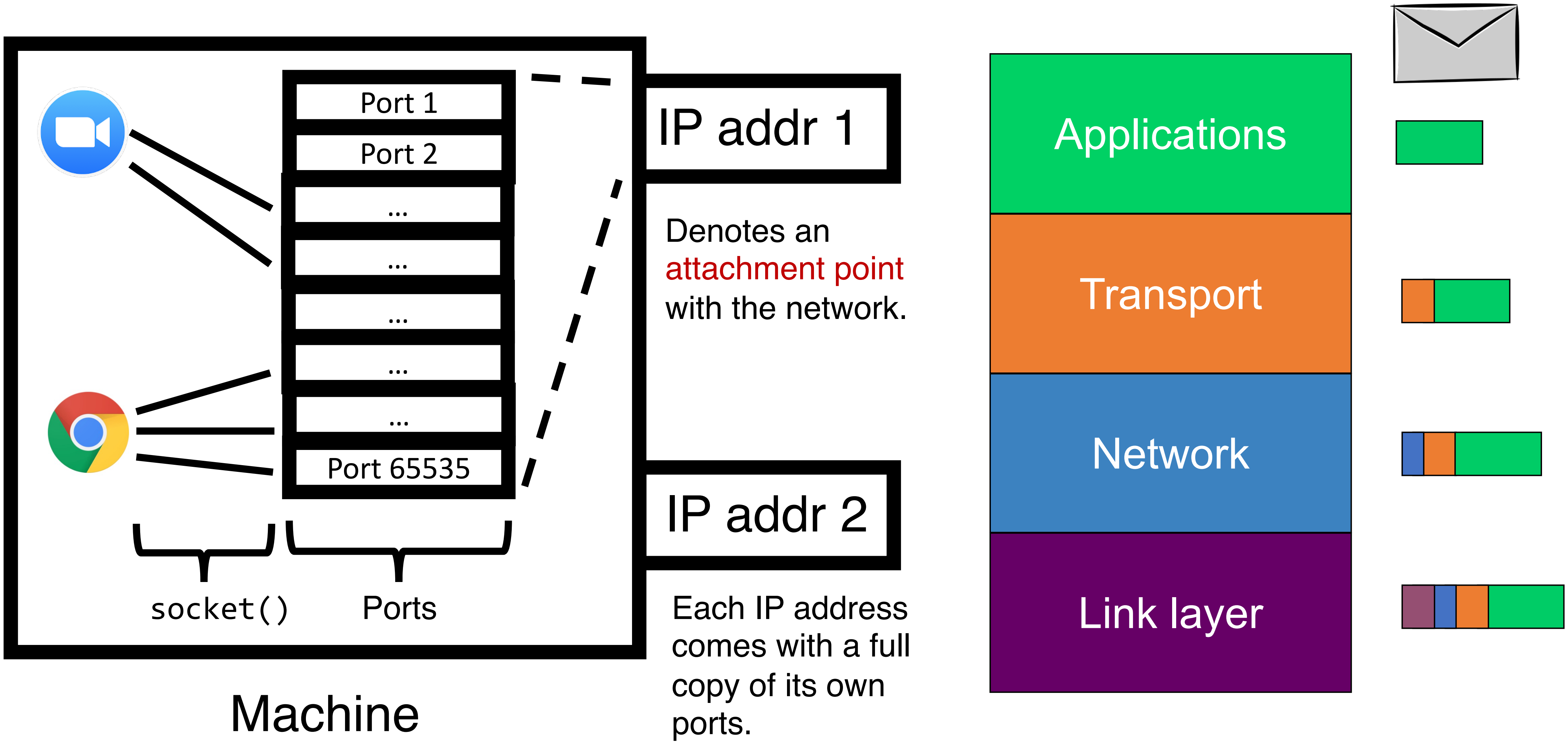
Transmission Control Protocol (TCP)

- Connection-based: the application remembers the other process talking to it.
- Suitable for longer-term, contextual data transfers, like HTTP, file transfers, etc.
- Guarantees: reliability, ordering, congestion control

User Datagram Protocol (UDP)

- Connectionless: app doesn't remember the last process or source that talked to it.
- Suitable for single req/resp flows, like DNS.
- Guarantees: basic error detection

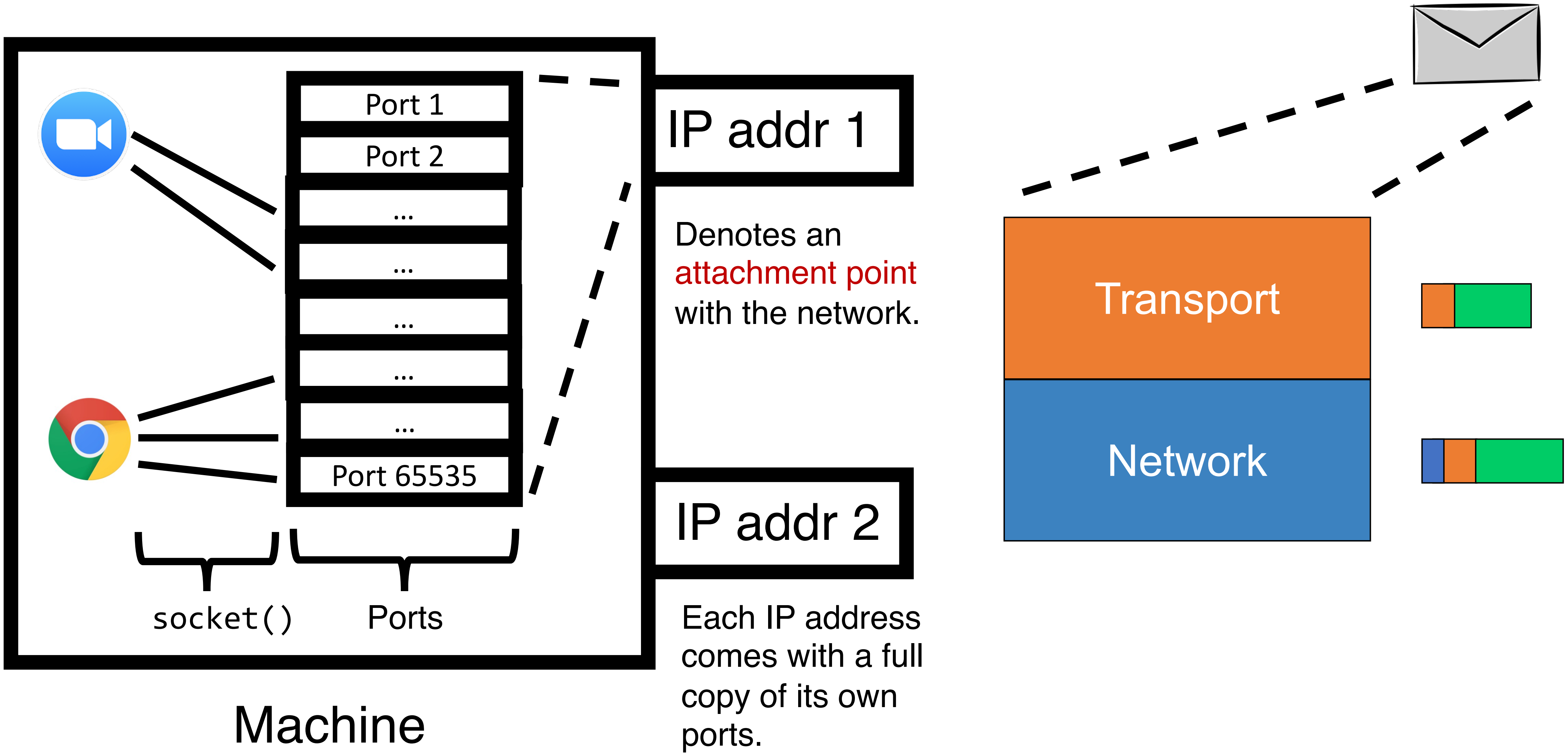
Demultiplexing



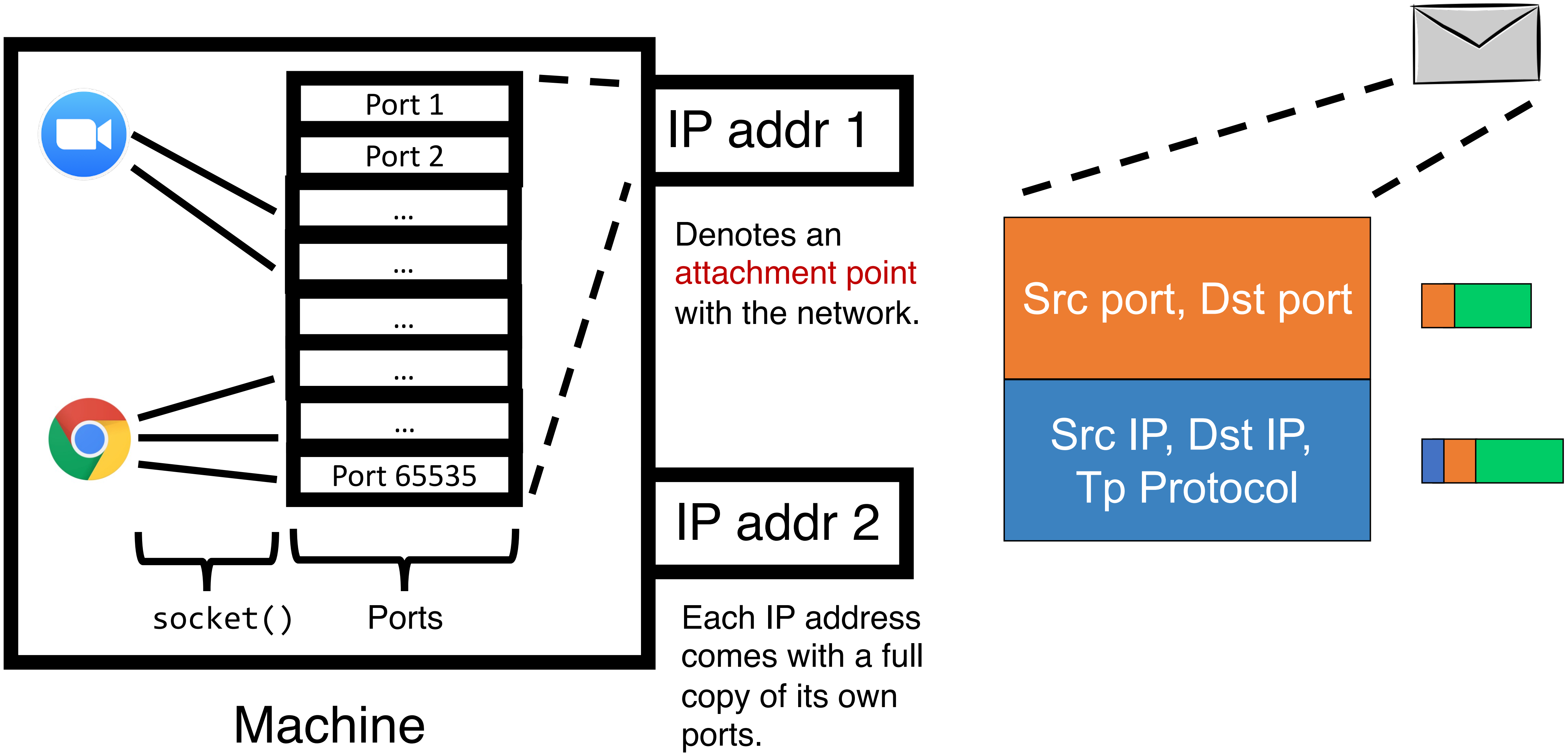
Denotes an attachment point with the network.

Each IP address comes with a full copy of its own ports.

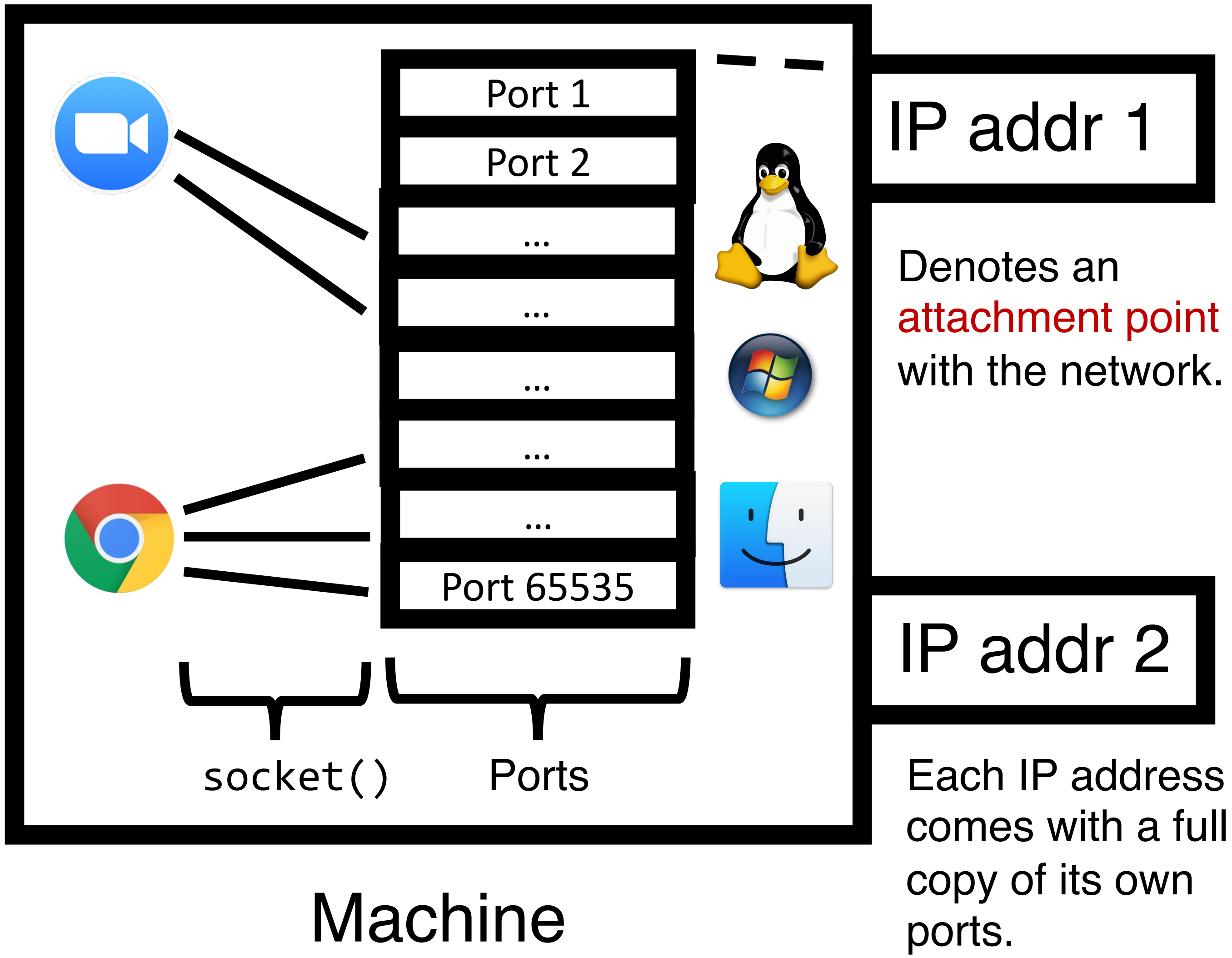
Demultiplexing



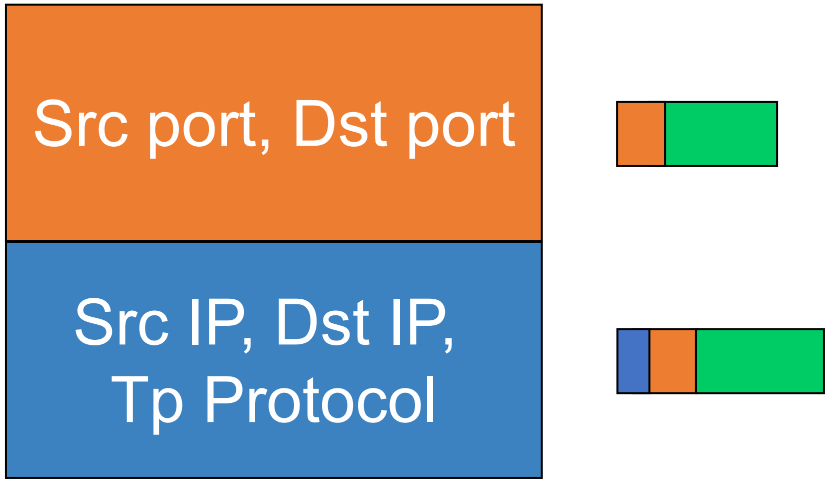
Demultiplexing



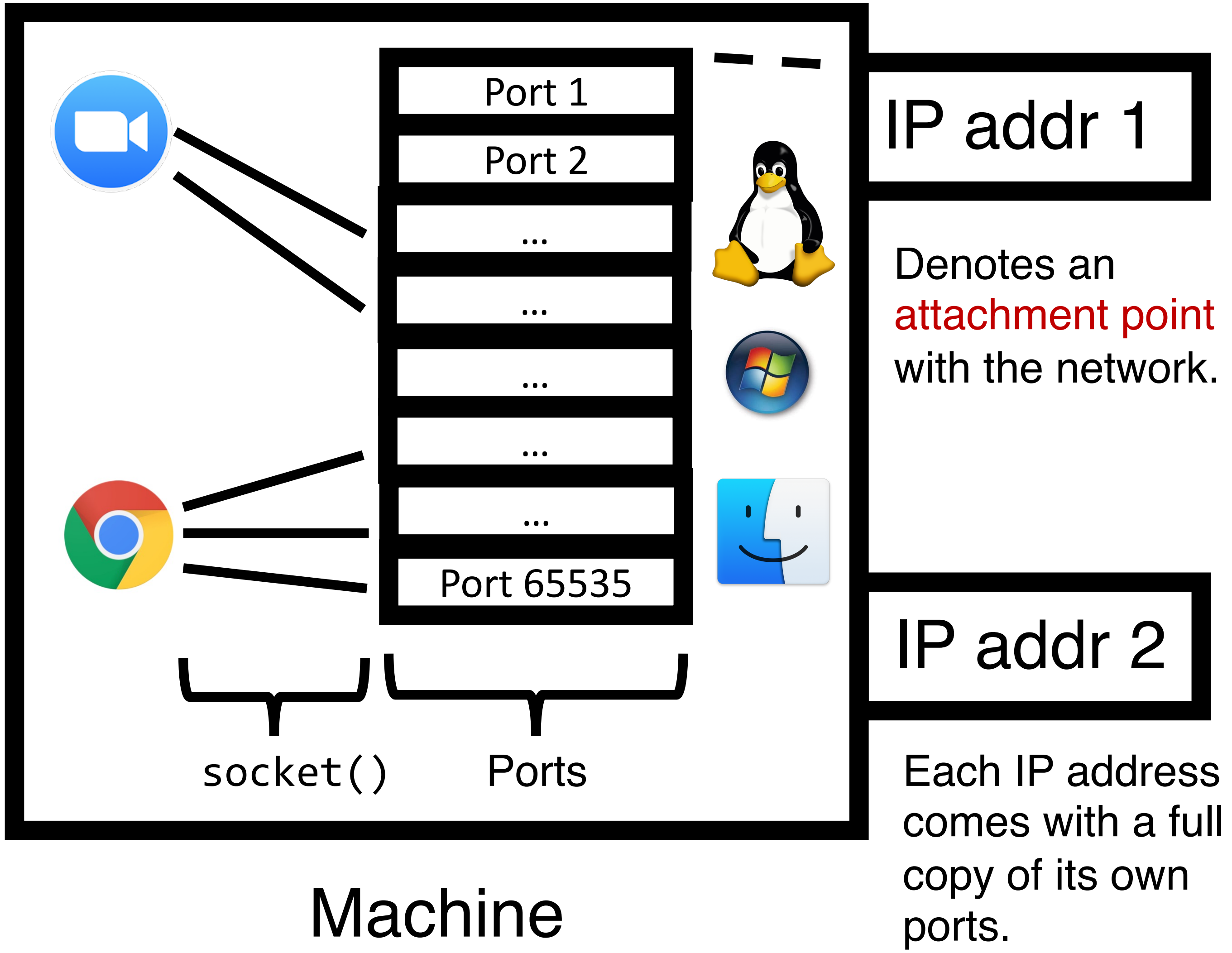
Demultiplexing



Connection lookup: The operating system does a lookup using these data to determine the right socket and app.



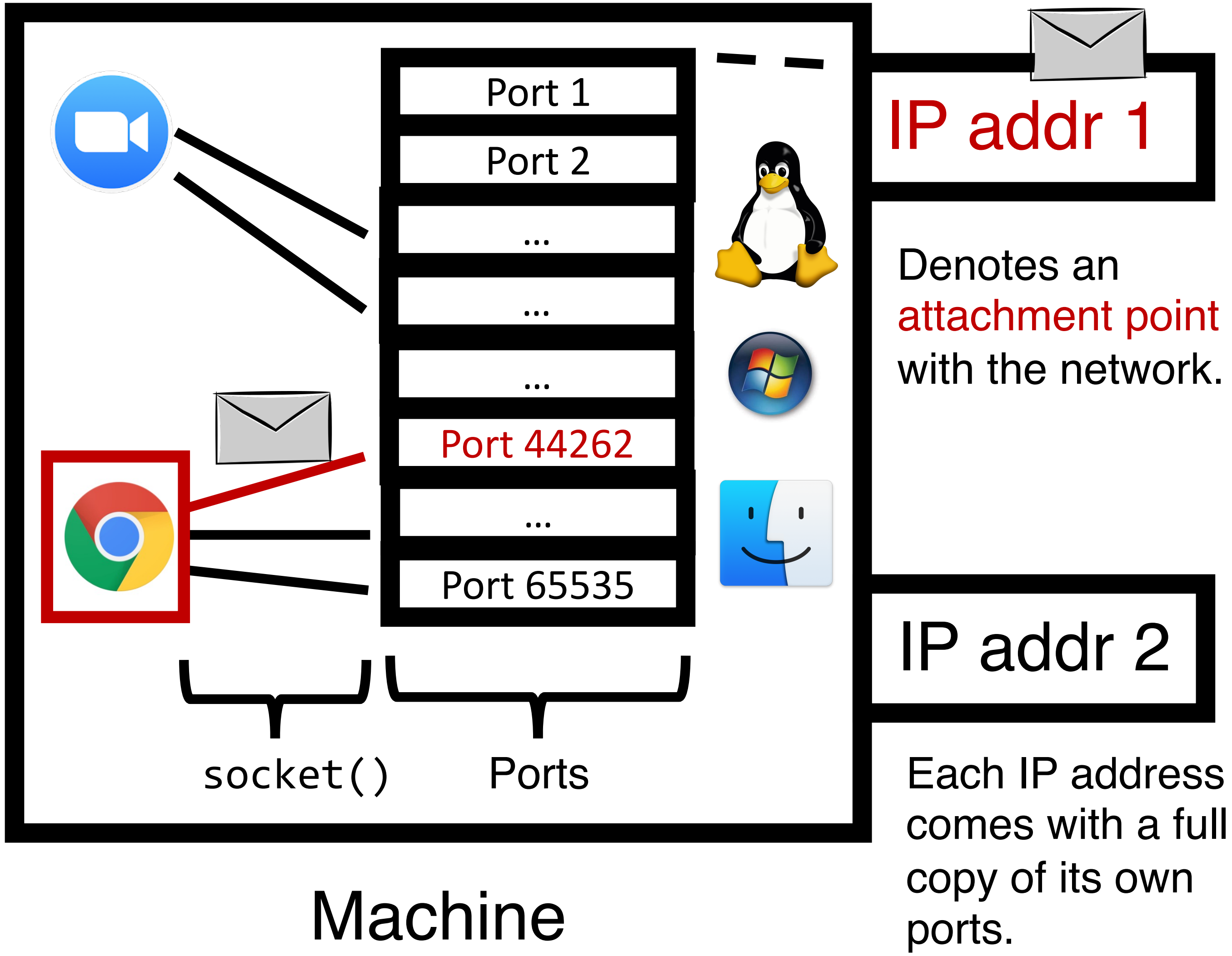
Demultiplexing



Connection lookup: The operating system does a lookup using these data to determine the right socket and app.

TCP sockets:
(src IP, dst IP, src port, dst port)
➔
Socket ID

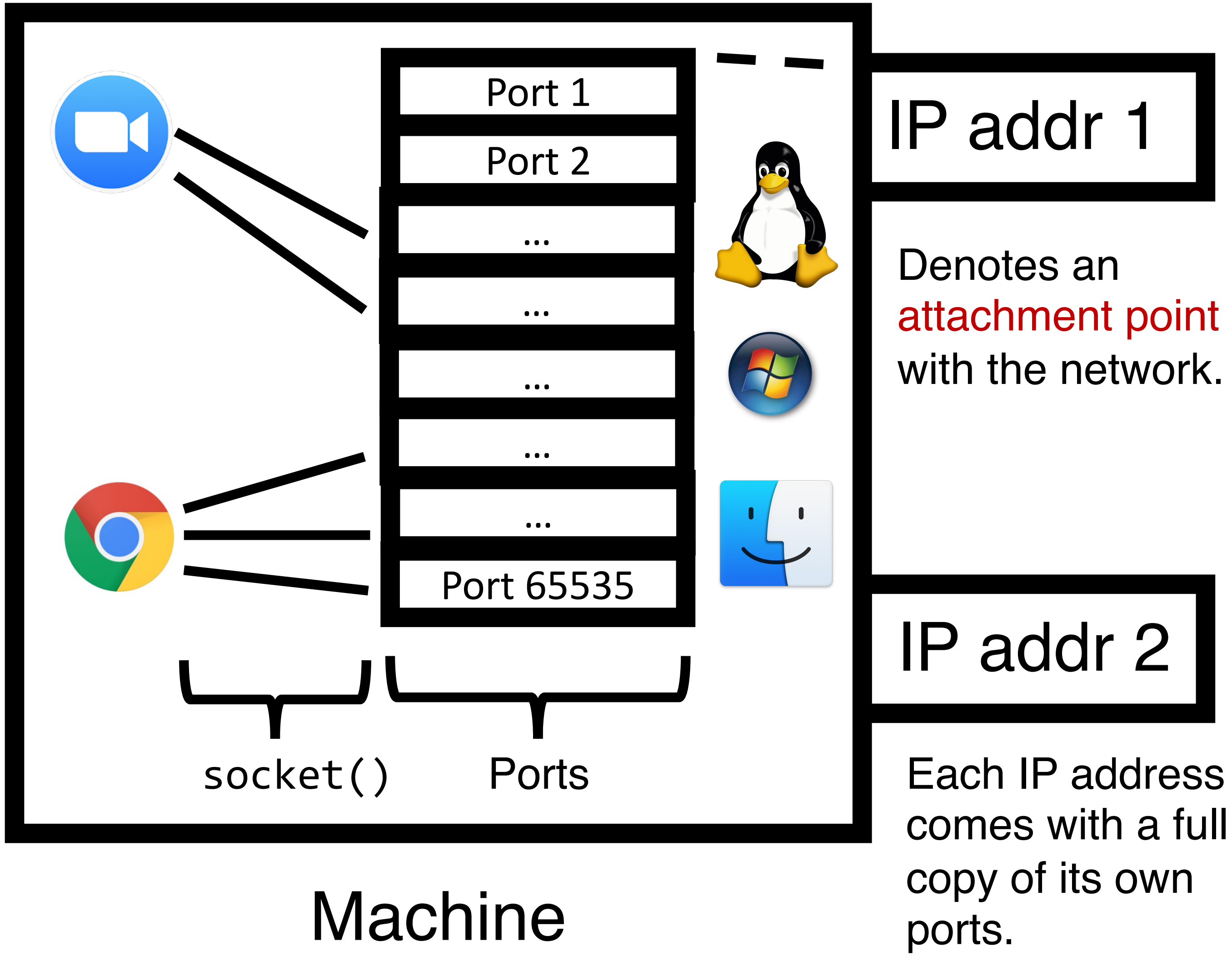
Demultiplexing



Connection lookup: The operating system does a lookup using these data to determine the right socket and app.

TCP sockets:
(src IP, dst IP, src port, dst port)
➔
Socket ID

Demultiplexing



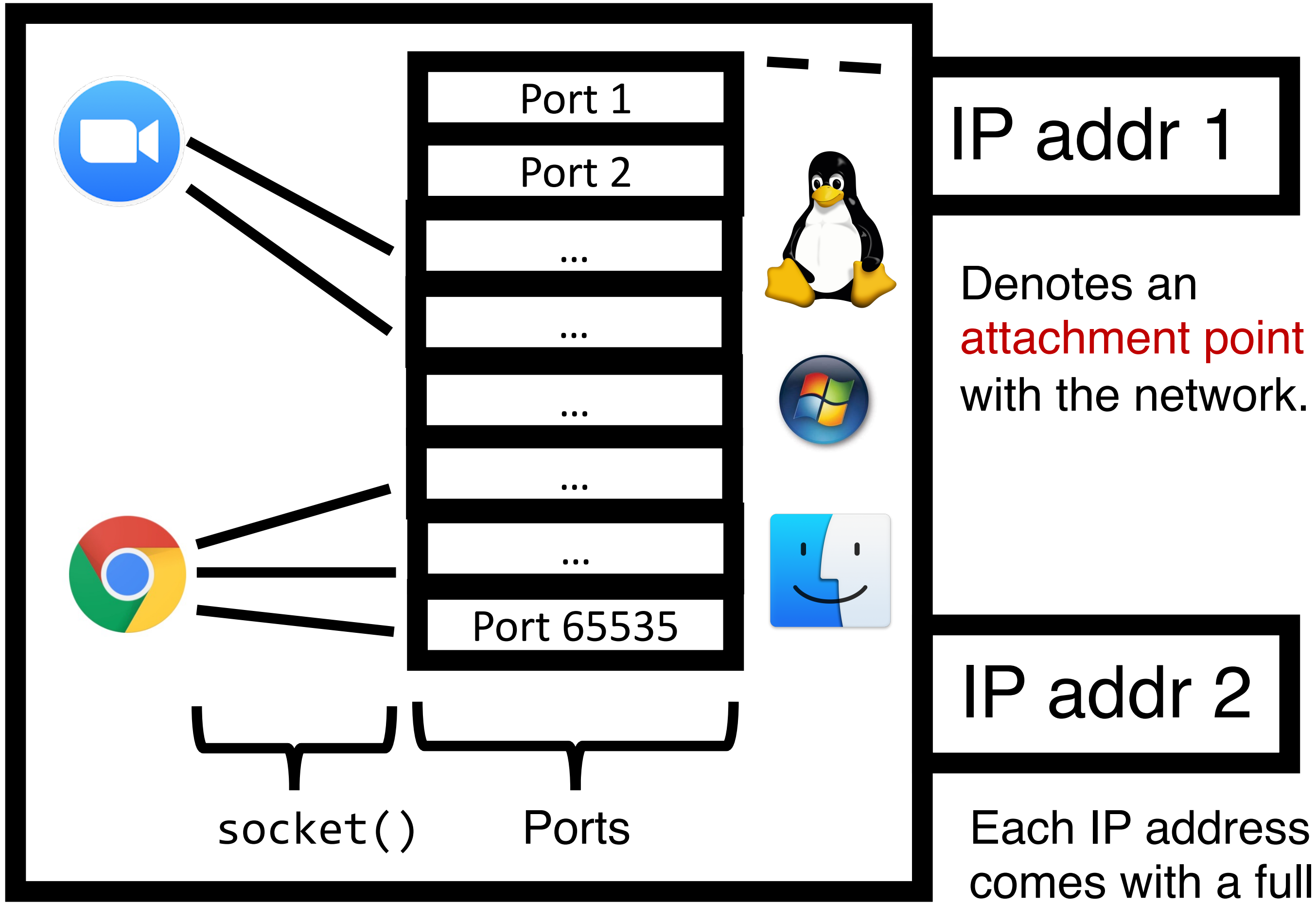
Connection lookup: The operating system does a lookup using these data to determine the right socket and app.

TCP sockets:
(src IP, dst IP, src port, dst port)
➔
Socket ID

UDP sockets:
(dst IP, dst port)
➔
Socket ID

Connectionless: the socket is shared across all sources!

Demultiplexing



Denotes an attachment point with the network.

Each IP address comes with a full copy of its own ports.

Machine

Connection lookup: The operating system does a lookup using these data to determine the right socket and app.

TCP sockets** Some caveats!

(src IP, dst IP, src port, dst port)

➔

Socket ID

UDP sockets:

(dst IP, dst port)

➔

Socket ID

Connectionless: the socket is shared across all sources!

TCP sockets of different types

Listening (bound but unconnected)

```
# On server side
ss = socket(AF_INET, SOCK_STREAM)
ss.bind(serv_ip, serv_port)
ss.listen() # no accept() yet
```

Connected (Established)

```
# On server side
csockid, addr = ss.accept()

# On client side
cs.connect(serv_ip, serv_port)
```

(src IP, dst IP, src port, dst port)

➔

Socket (`csockid NOT ss`)

TCP sockets of different types

Listening (bound but unconnected)

```
# On server side
ss = socket(AF_INET, SOCK_STREAM)
ss.bind(serv_ip, serv_port)
ss.listen() # no accept() yet
```

(dst IP, dst port)

➔

Socket (ss)

Enables new connections to be demultiplexed correctly

Connected (Established)

```
# On server side
csockid, addr = ss.accept()
```

`accept()` creates a new socket with the 4-tuple (established) mapping

```
# On client side
cs.connect(serv_ip, serv_port)
```

(src IP, dst IP, src port, dst port)

➔

Socket (csockid NOT ss)

Enables existing connections to be demultiplexed correctly

TCP demultiplexing

- When a TCP packet comes in, the operating system:

- Looks up table of existing connections using 4-tuple
  - If success, send to corresponding (established) socket

- If fail (no table entry), look up table of listening connections using just (dst IP, dst port)
  - If success, send to corresponding (listening) socket

- If fail again (no table entry), send error to client
  - Connection refused

UDP demultiplexing

- When a UDP packet comes in, the operating system:

- Looks up table of listening UDP sockets using (dst IP, dst port)
  - If success, send packet to corresponding socket
  - There are no “established” UDP sockets

- If fail (no table entry), send error to client
  - Port unreachable

Listing sockets and connections

- List all sockets with `ss`
- Create and observe UDP sockets with `iperf`
- Observe a TCP listening socket with `iperf` (or your own server!)