Transport: Demultiplexing

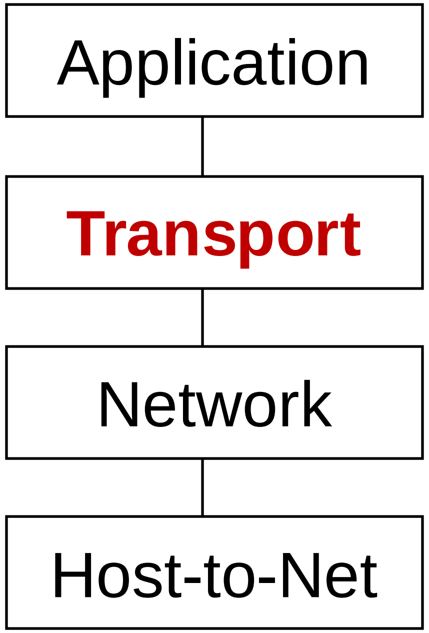
Lecture 11

http://www.cs.rutgers.edu/~sn624/352-F24

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Transport

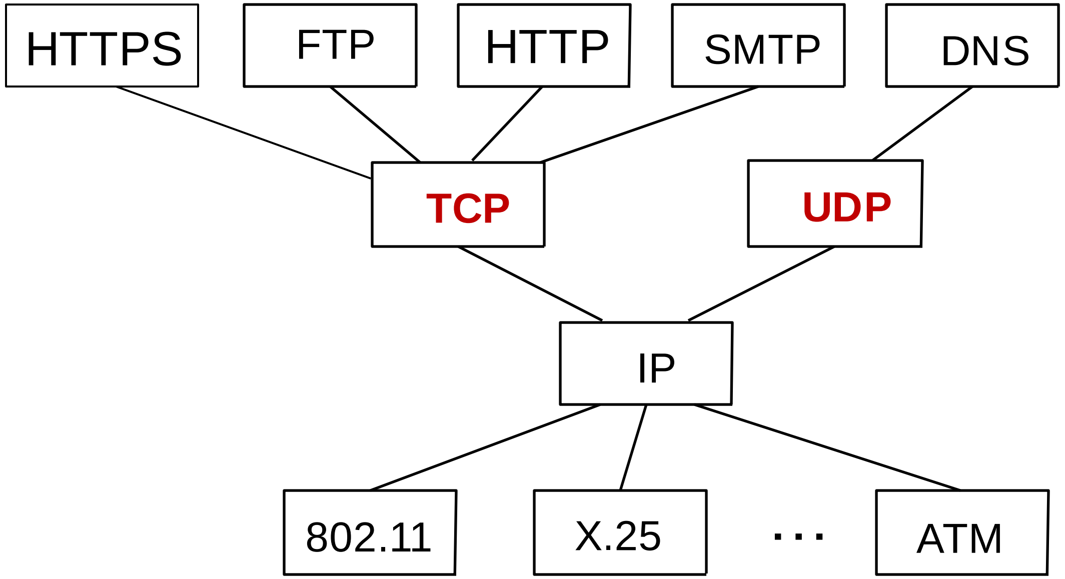


Application

Transport

Network

Host-to-Net



HTTPS FTP HTTP SMTP DNS

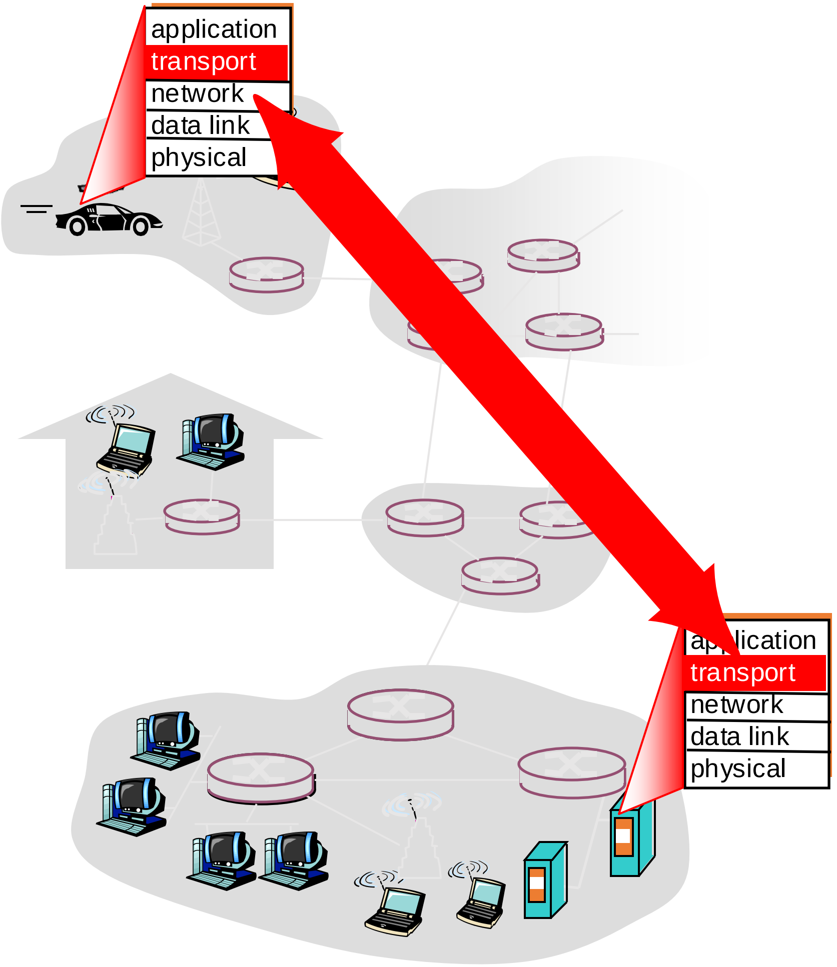
TCP **UDP**

IP

802.11 X.25 … ATM

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

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

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

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

Demultiplexing

(Not always 4-tuple)

- 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

Two popular transports

Transmission Control Protocol (TCP)

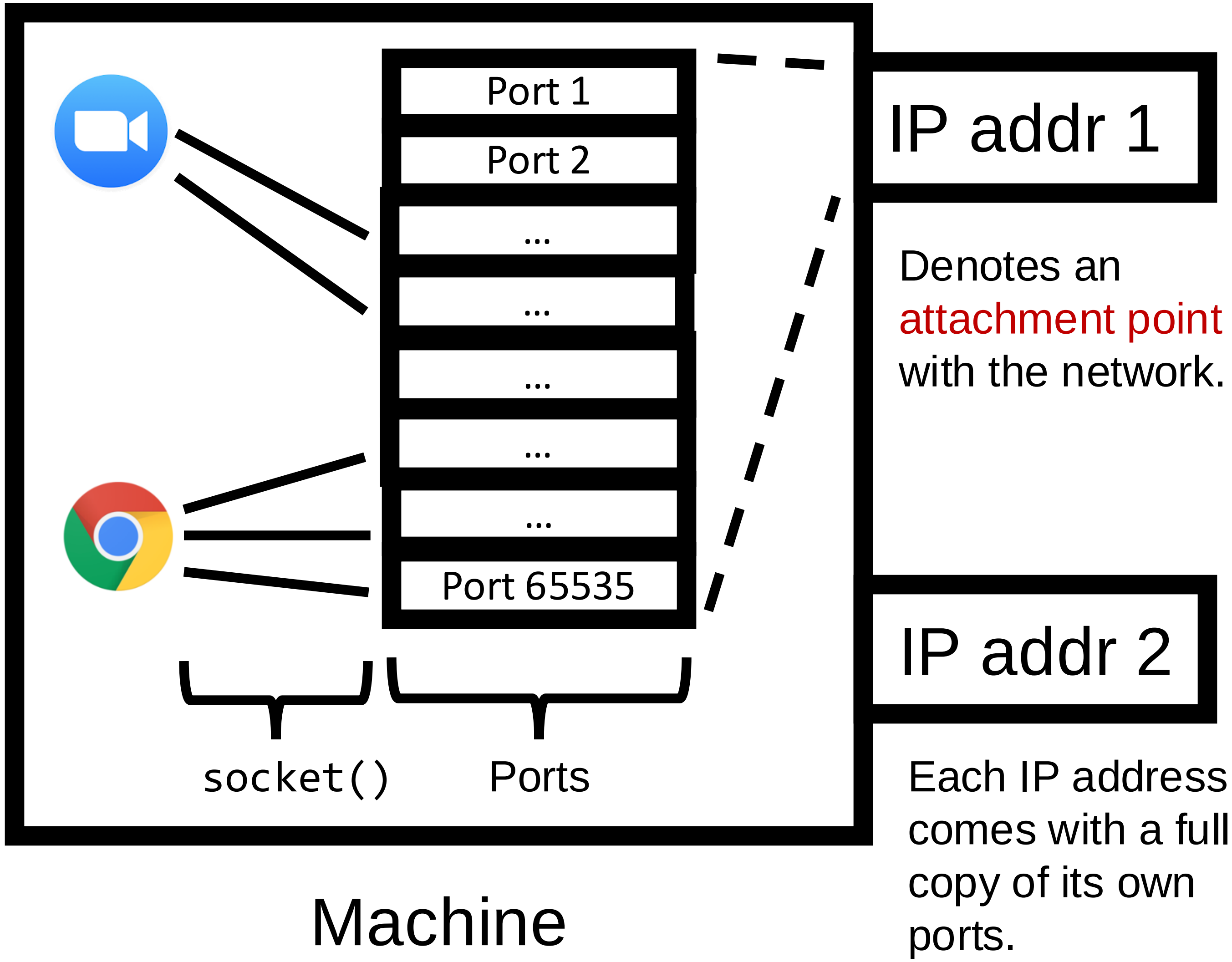
- Connection-based: the application remembers the other process talking to it.
- Suitable for longer-term, contextual data transfers, like HTTP, e-mail, etc.
- Guarantees: reliability, ordering, congestion control

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

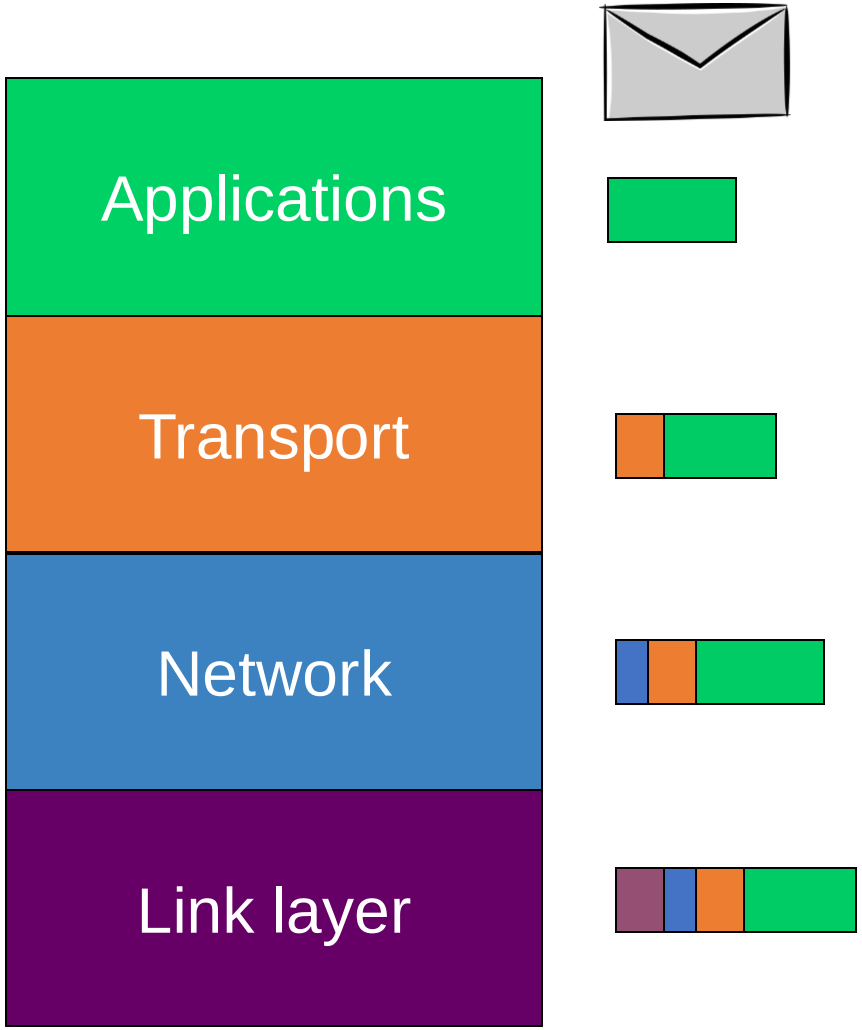
Demultiplexing Packets

Demultiplexing

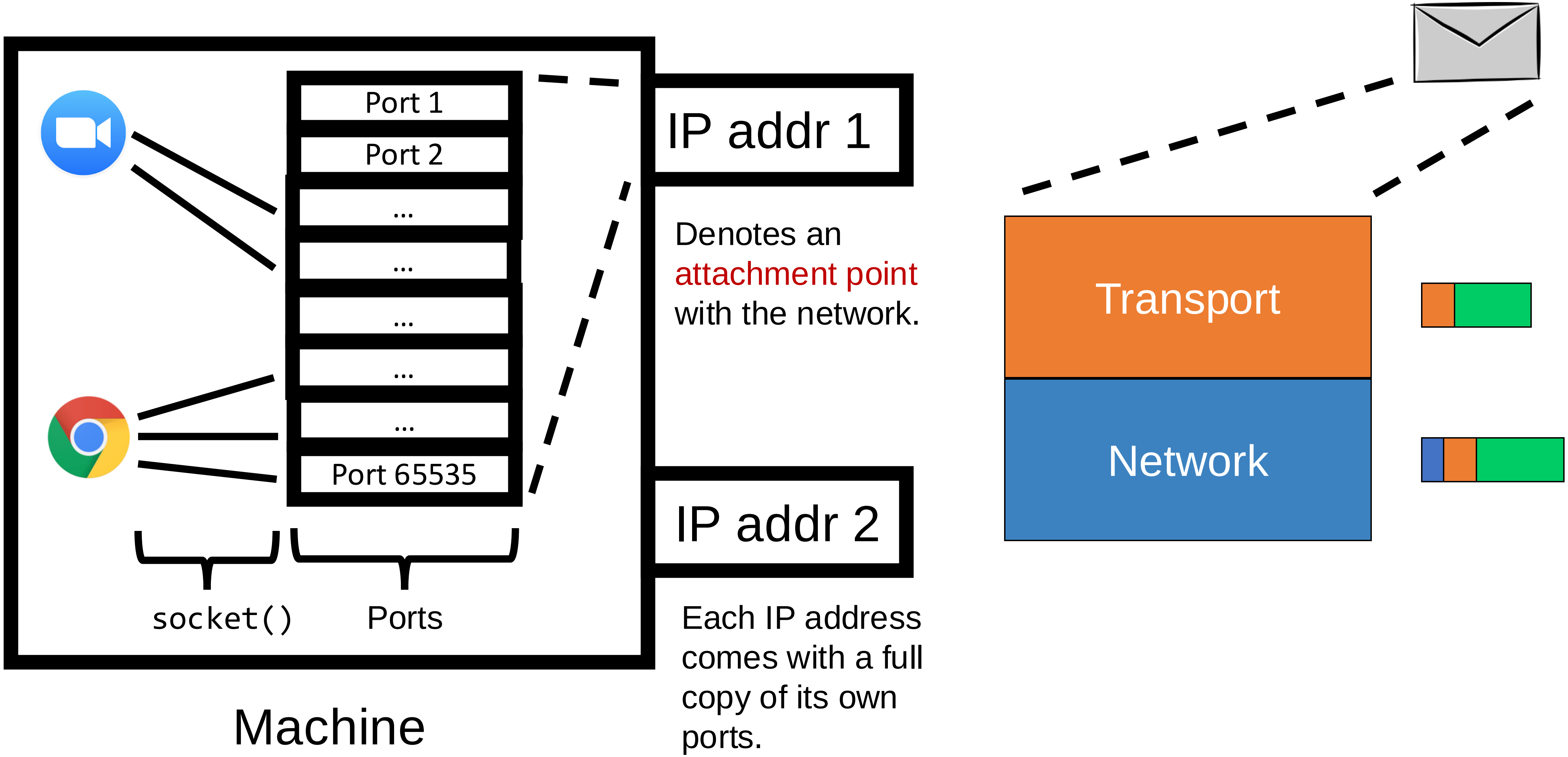


Denotes an attachment point with the network.

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



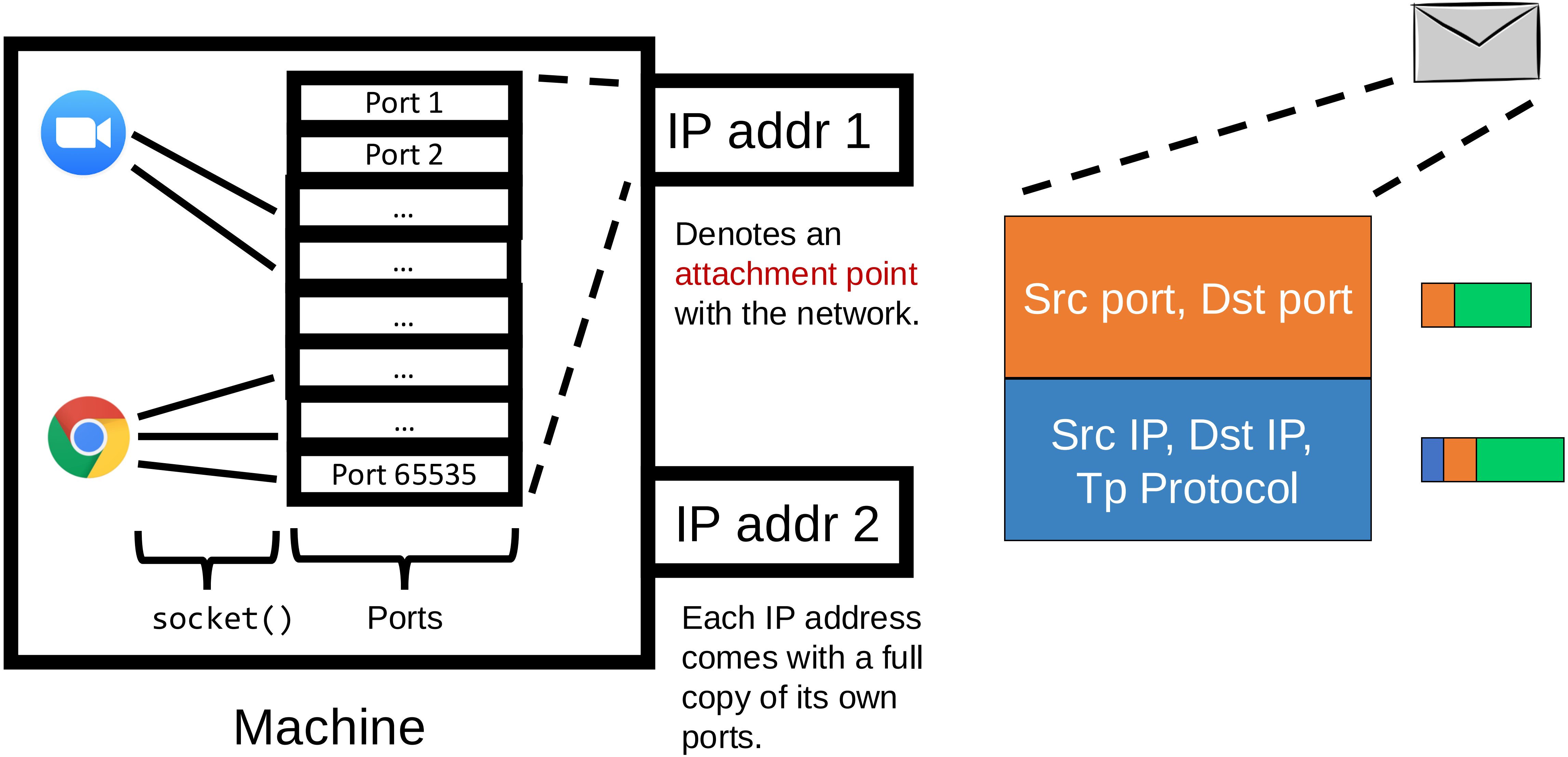
Demultiplexing



Denotes an attachment point with the network.

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

Demultiplexing



Port 1

Port 2

...

...

...

...

...

Port 65535

socket()

Ports

Machine

IP addr 1

Denotes an attachment point with the network.

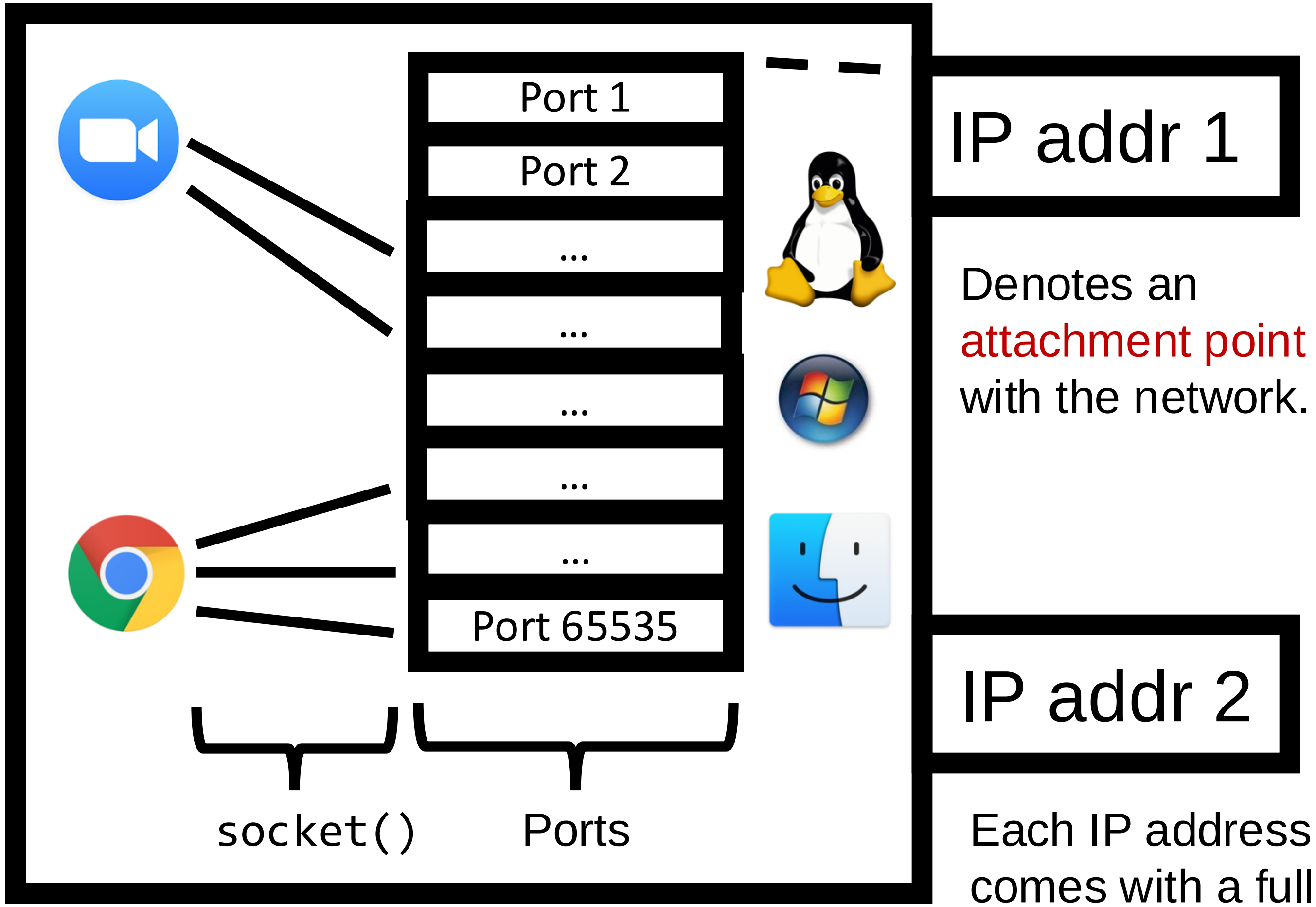
IP addr 2

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

Src port, Dst port

Src IP, Dst IP, Tp Protocol

Demultiplexing



Port 1

Port 2

…

…

…

…

…

Port 65535

socket()

Ports

IP addr 1

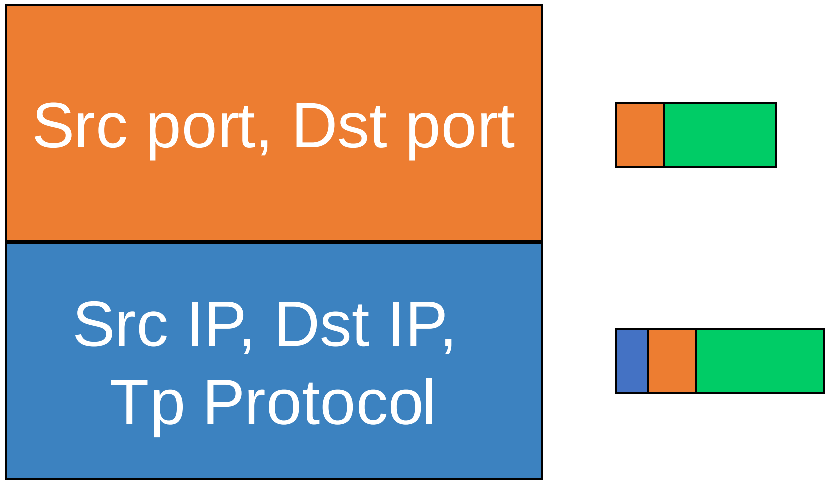
IP addr 2

Denotes an attachment point with the network.

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

Machine

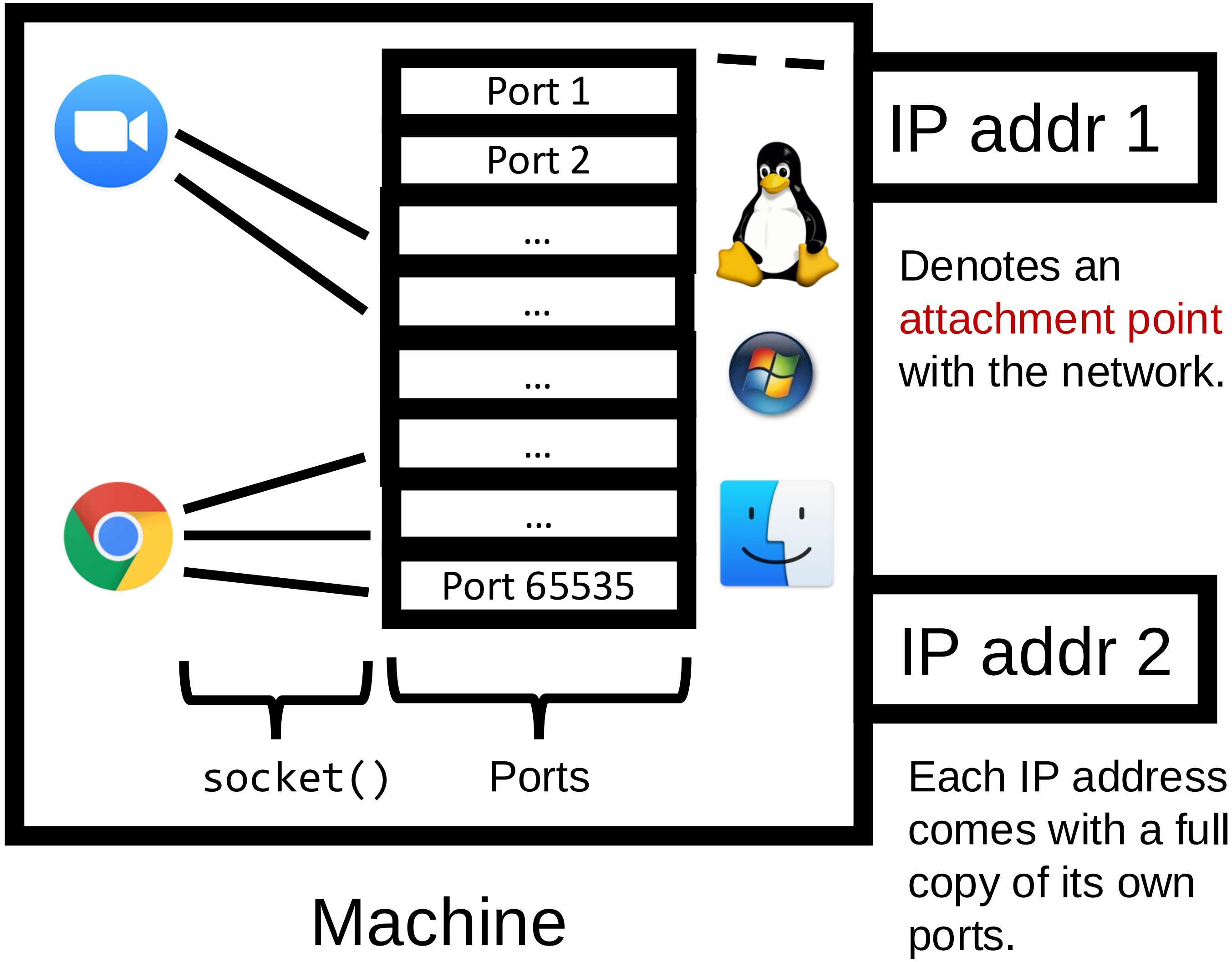
Connection lookup: The operating system does a lookup using these data to determine the right socket and app.



Src port, Dst port

Src IP, Dst IP, Tp Protocol

Demultiplexing



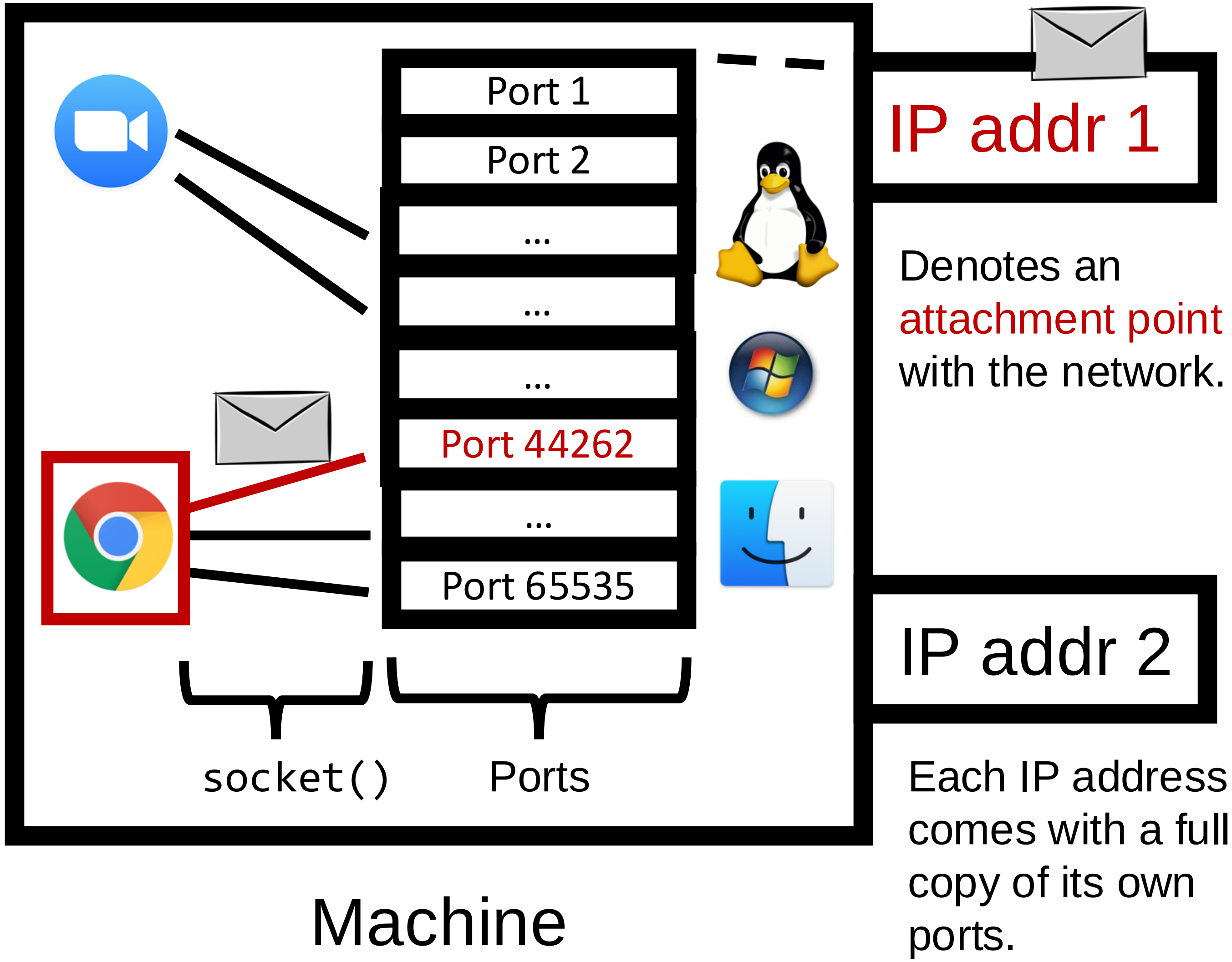
Denotes an attachment point with the network.

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

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



Denotes an attachment point with the network.

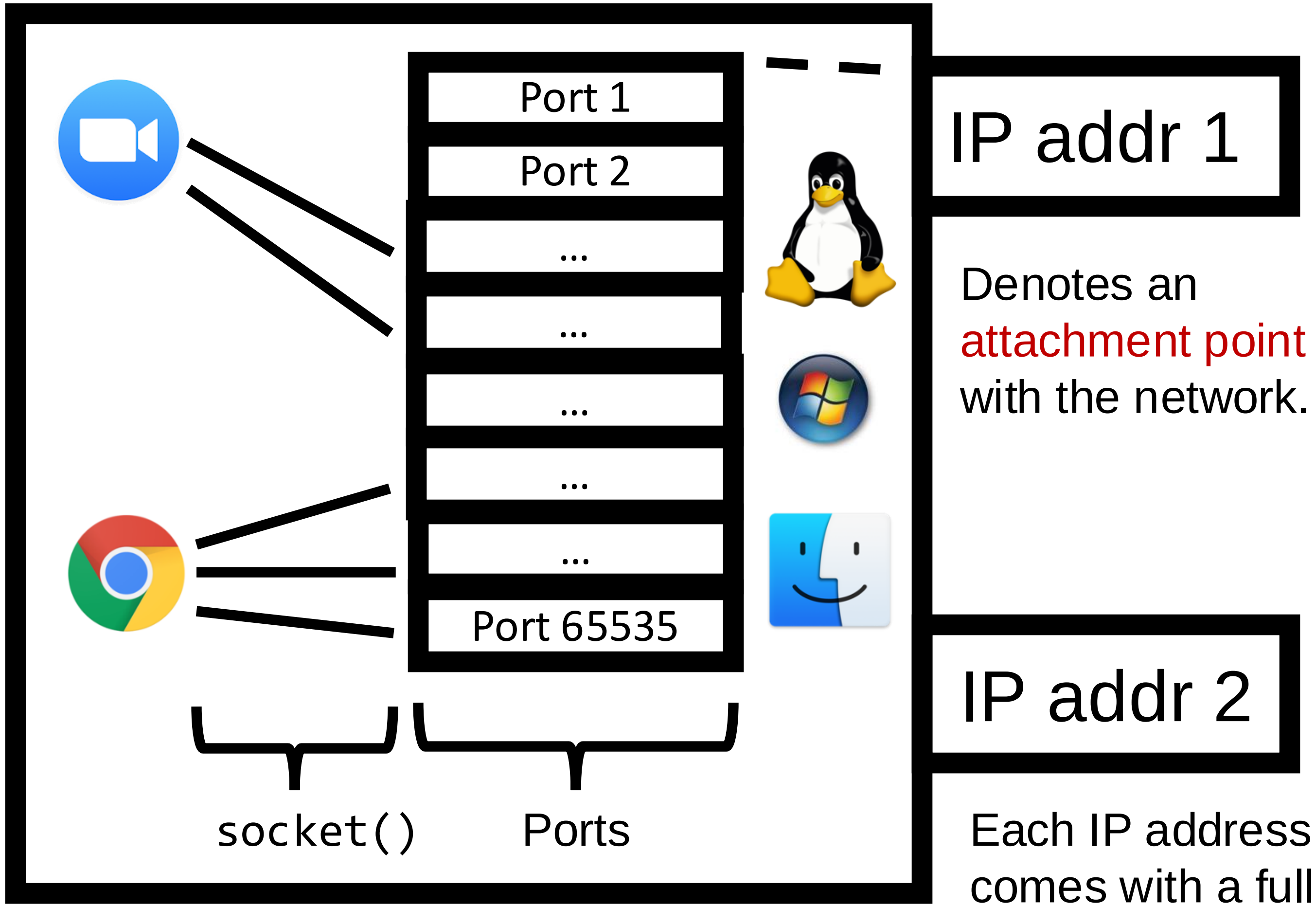
Each IP address comes with a full copy of its own ports.

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

(Our familiar 4-tuple)

Demultiplexing



Machine

Denotes an attachment point with the network.

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

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

(Our familiar 4-tuple)

UDP sockets:

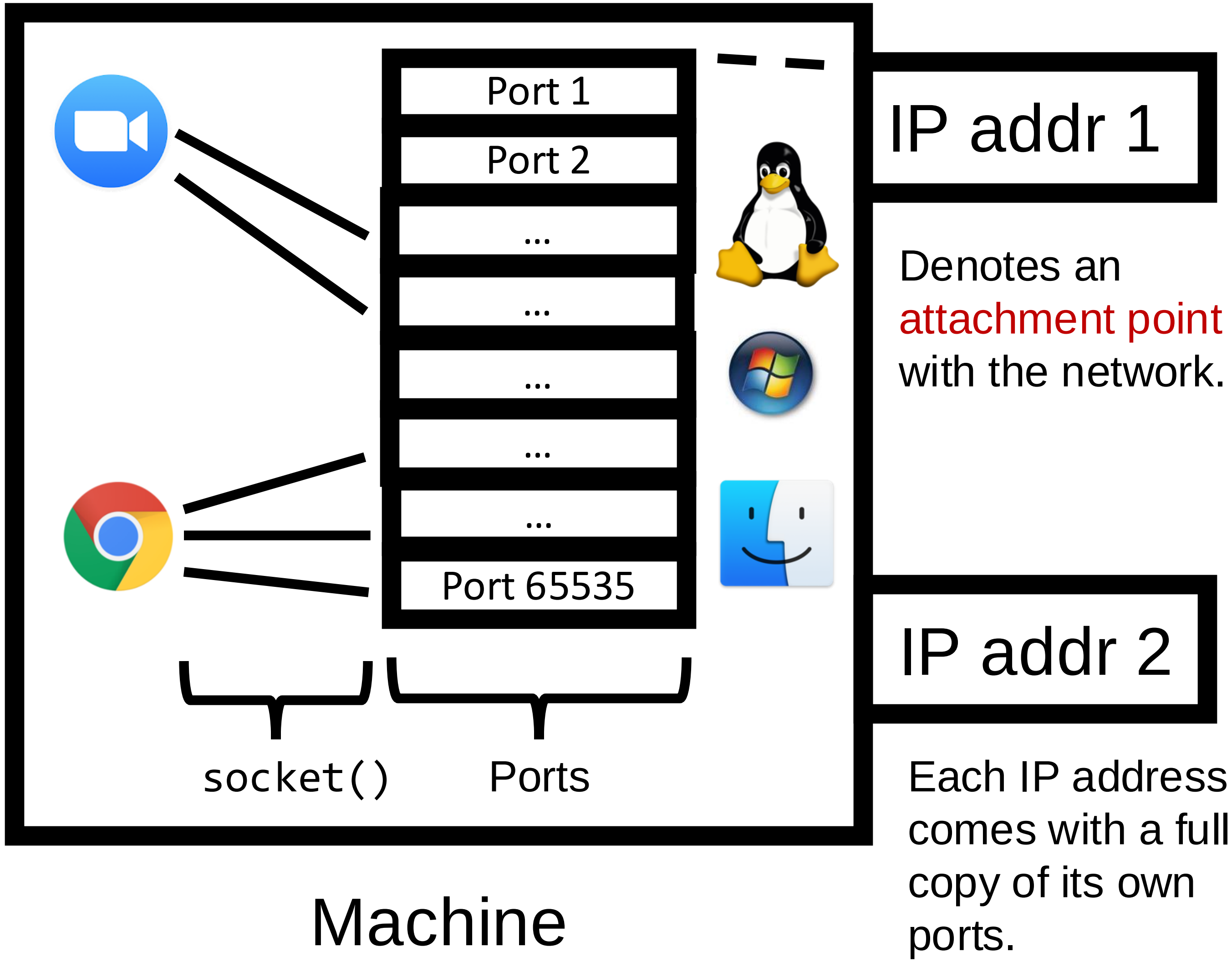
(dst IP, dst port)

➔

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.

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

TCP sockets** More cases!
(src IP, dst IP, src port, dst port)
➔
Socket ID

(Our familiar 4-tuple)

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()

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

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

(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 established 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
  - Add an entry for established connection in the established table (next packet from the established connection will demultiplex correctly)

- If lookup failed in the listening table (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; they're all "unconnected"

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