CS 352
Transport: Intro

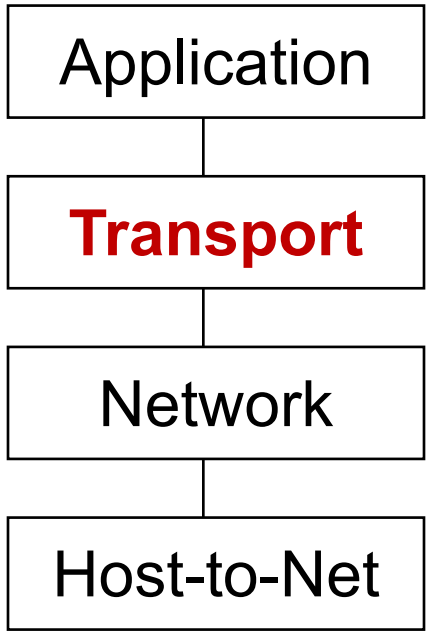
CS 352, Lecture 7.1

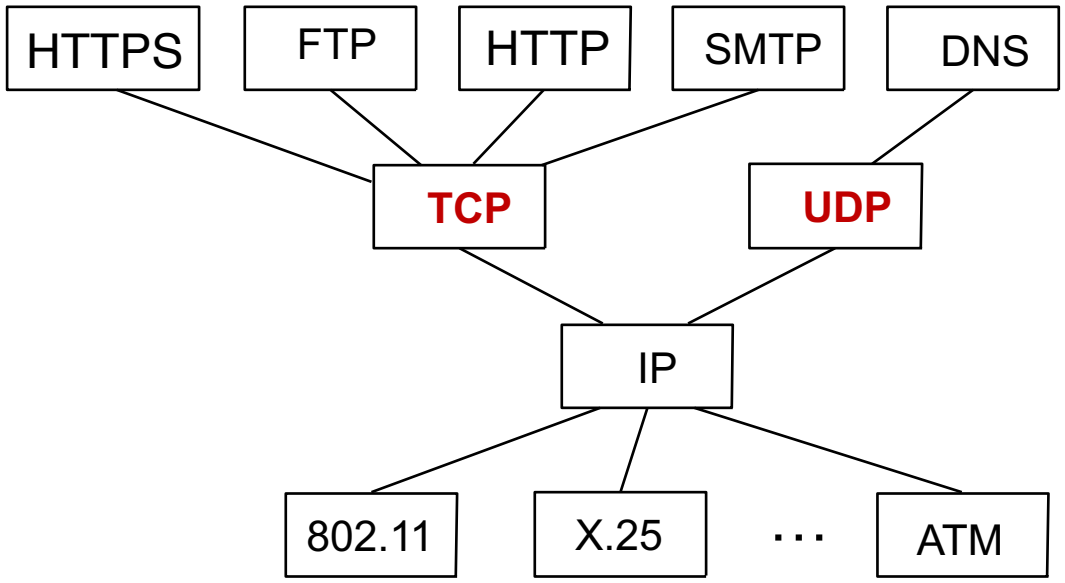
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

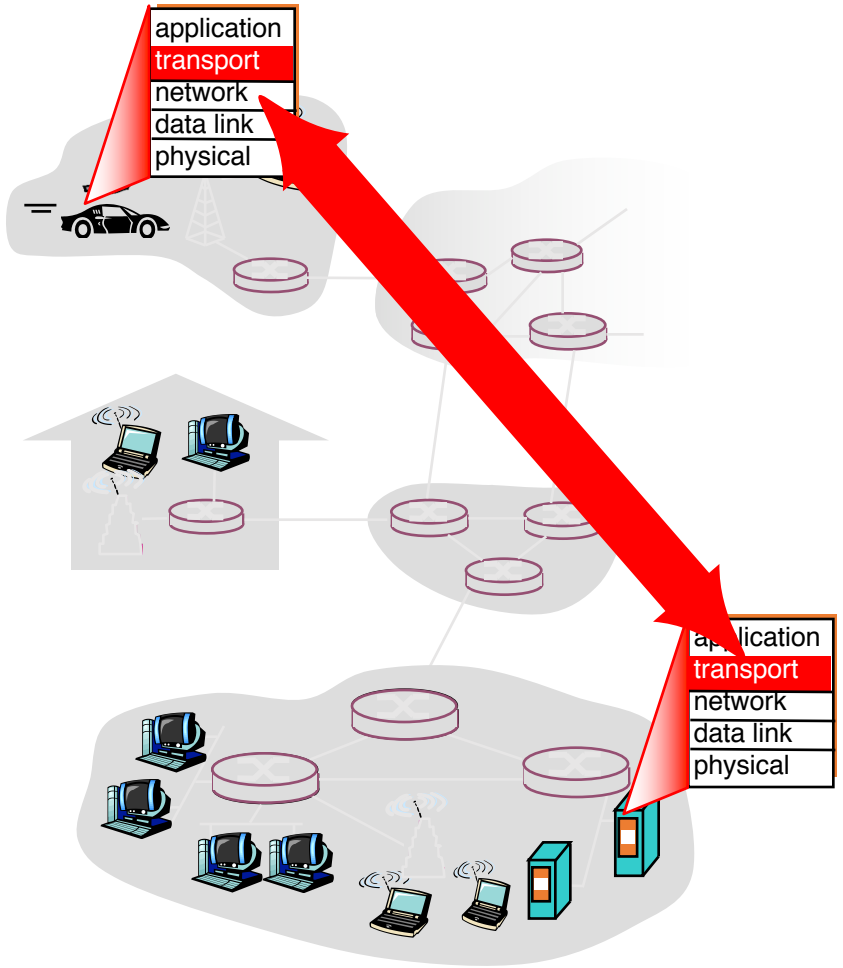
Transport


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

- 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:

12 kids sending letters to 12 kids

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

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

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.

Hotel analogy:

Hotel residents order food to their rooms from a restaurant using a delivery service.

- processes = residents of rooms and restaurant chefs
- app messages = food packages
- endpoint = the hotel / restaurant
- transport protocol = local hotel staff who bring the food to the different rooms
- network-layer protocol = food delivery service

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 restaurant
- Source port: the chef preparing the specific order
- Destination address: the address of the hotel
- Destination port: room number in the hotel

CS 352
Demultiplexing Packets

CS 352, Lecture 7.2

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

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Two popular transports

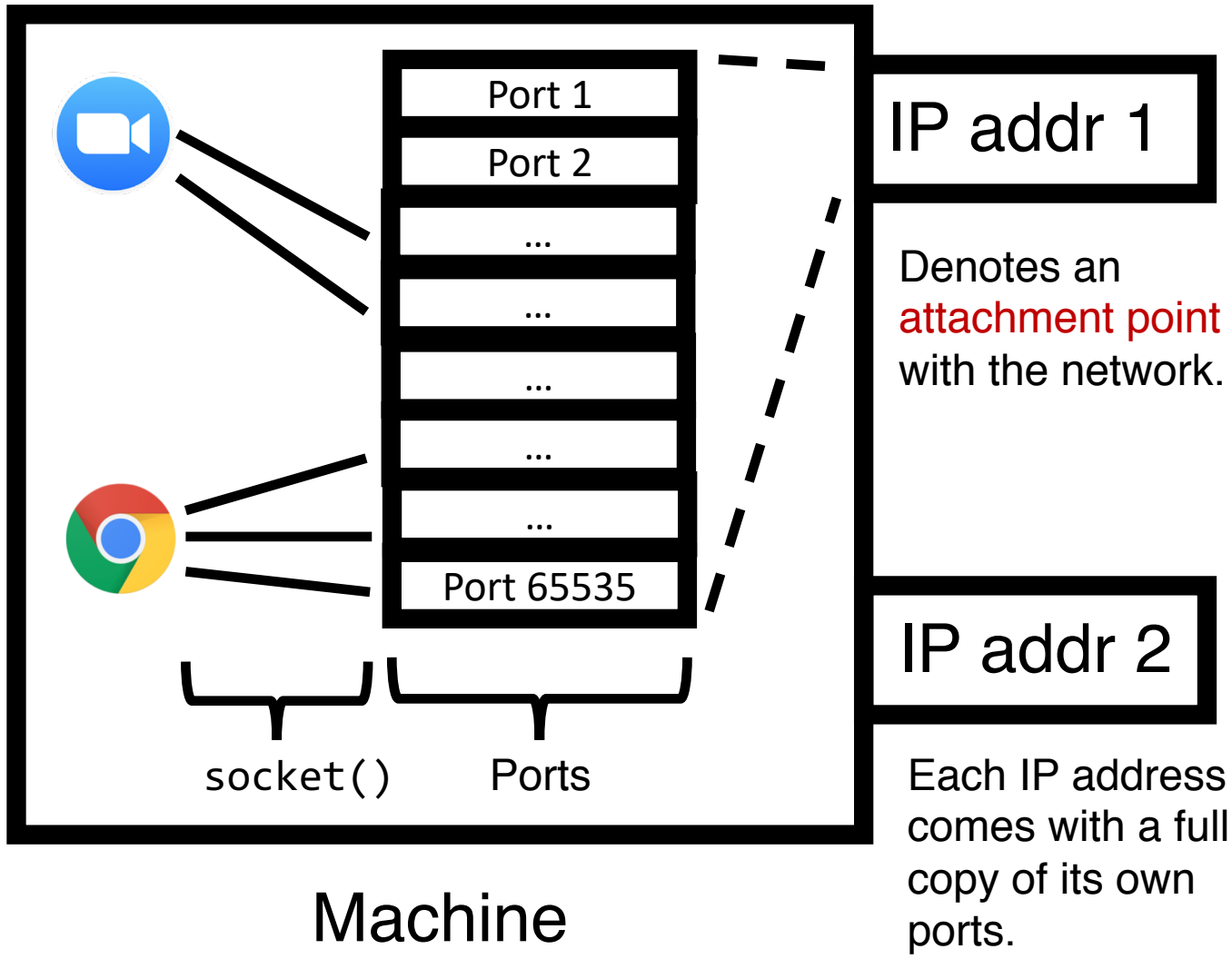
Transmission Control Protocol (TCP)

- 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)

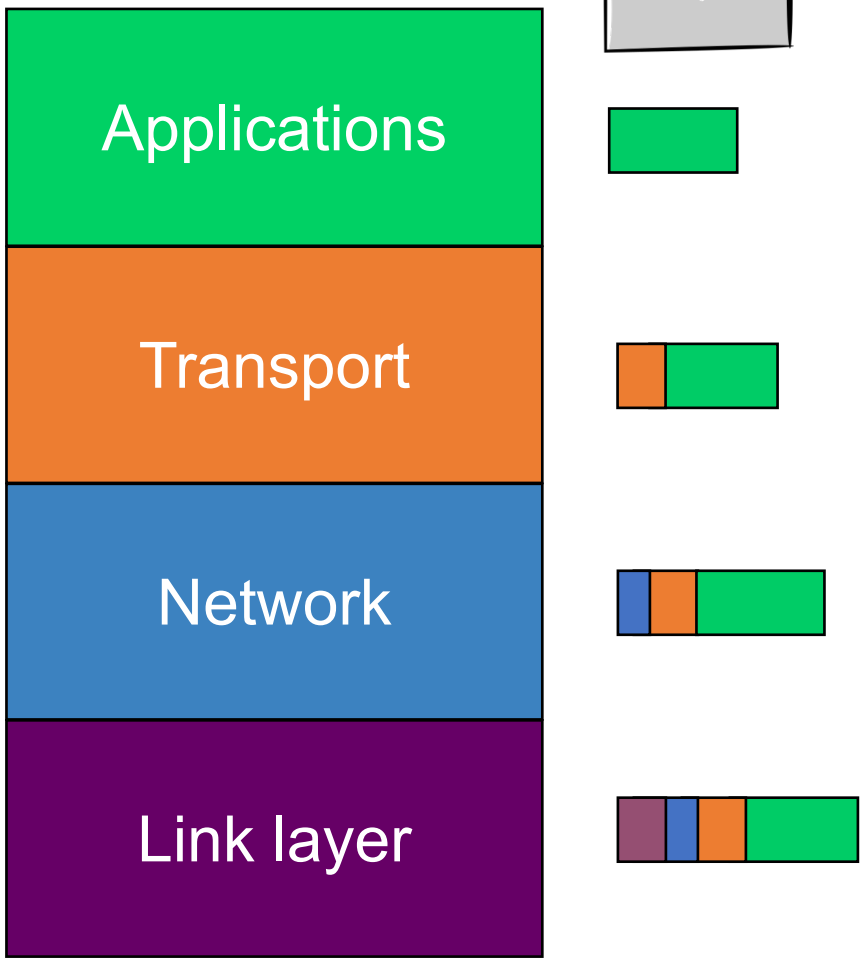
- Connectionless: app doesn't remember the last process or source that talked to it.
- Suitable for single req/resp flows, like DNS.
- Guarantees: basic error detection

Demultiplexing

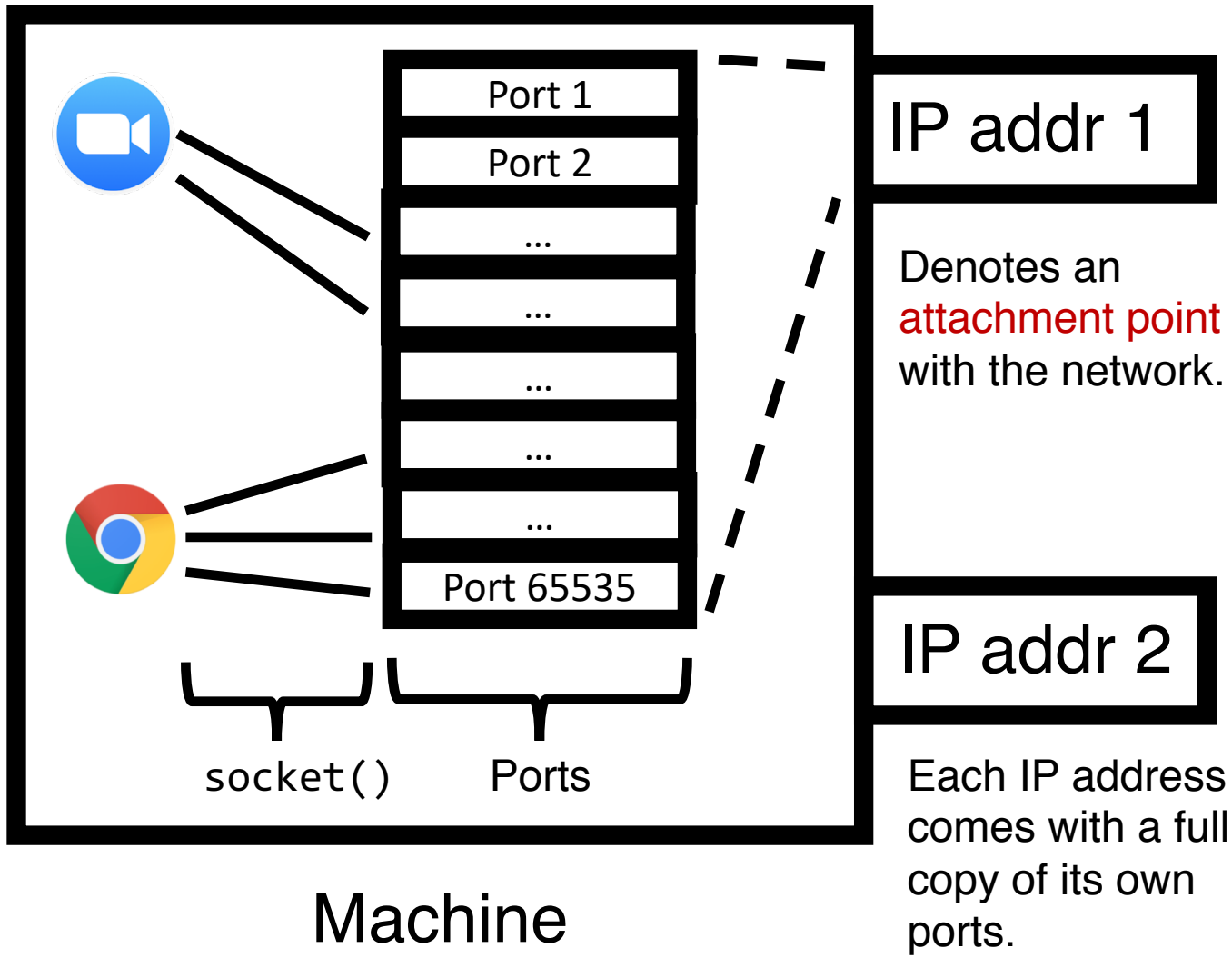


Denotes an attachment point with the network.

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

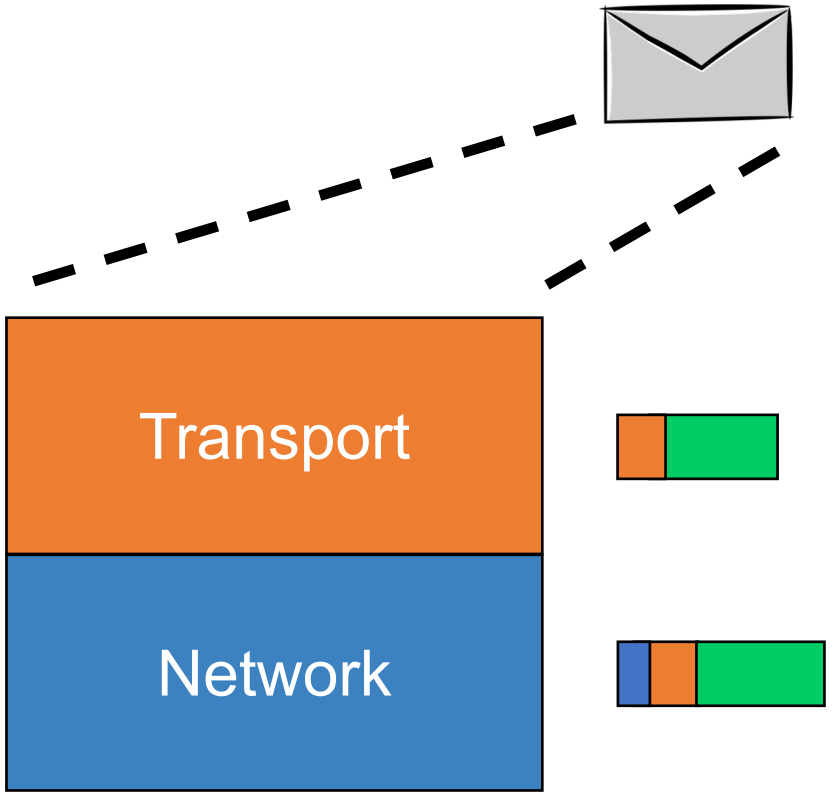


Demultiplexing

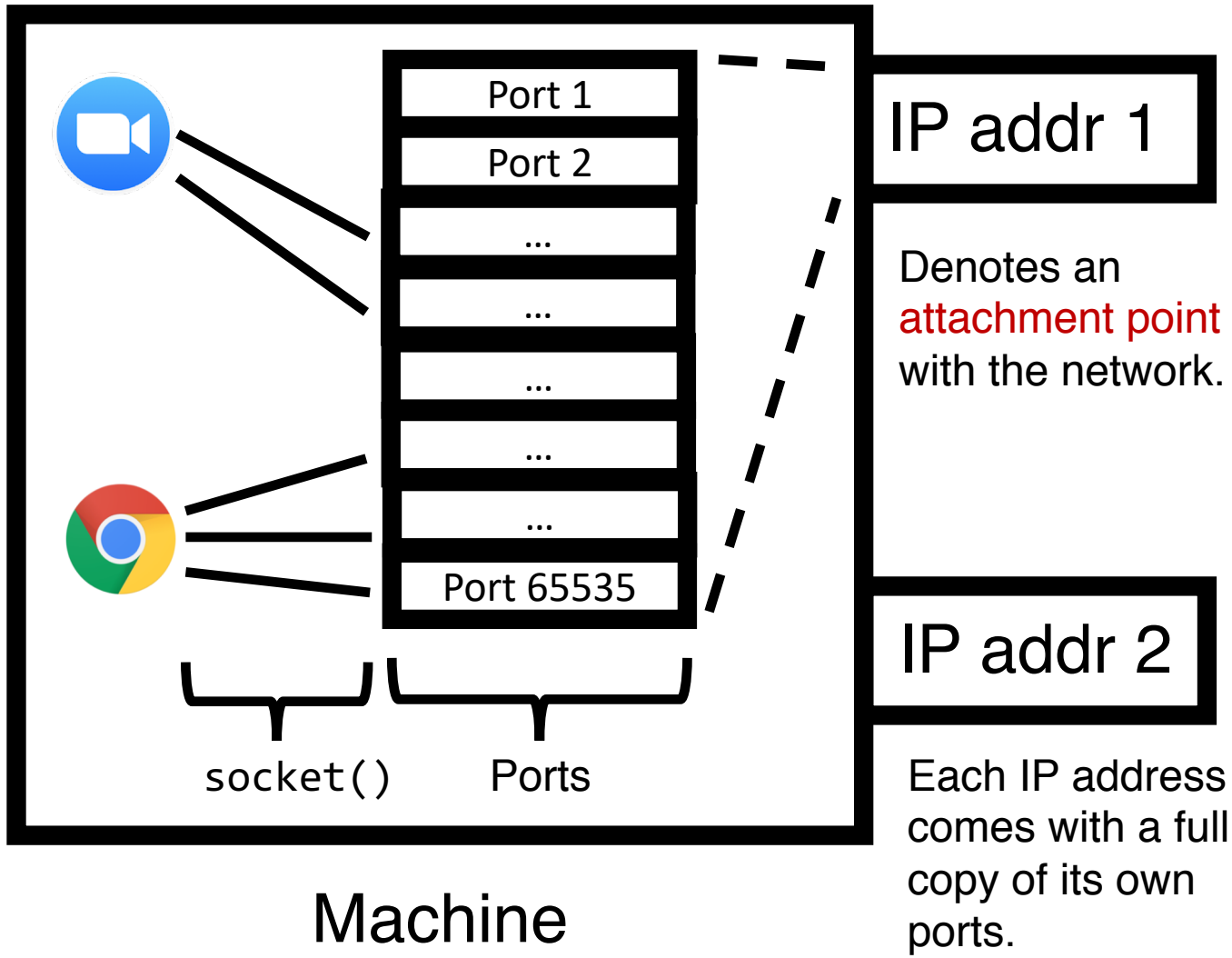


Denotes an attachment point with the network.

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

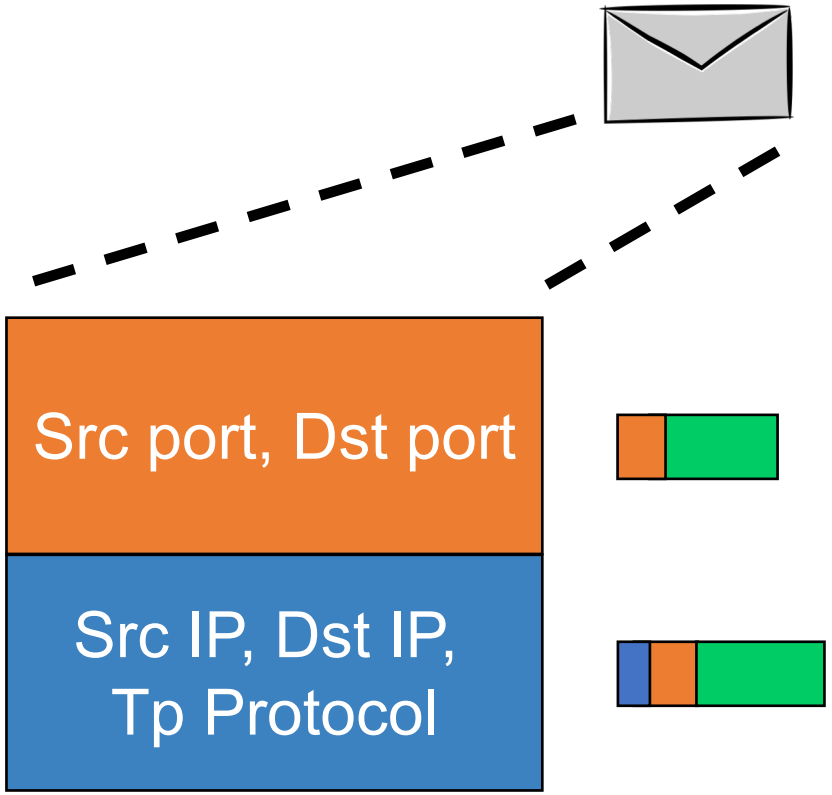


Demultiplexing

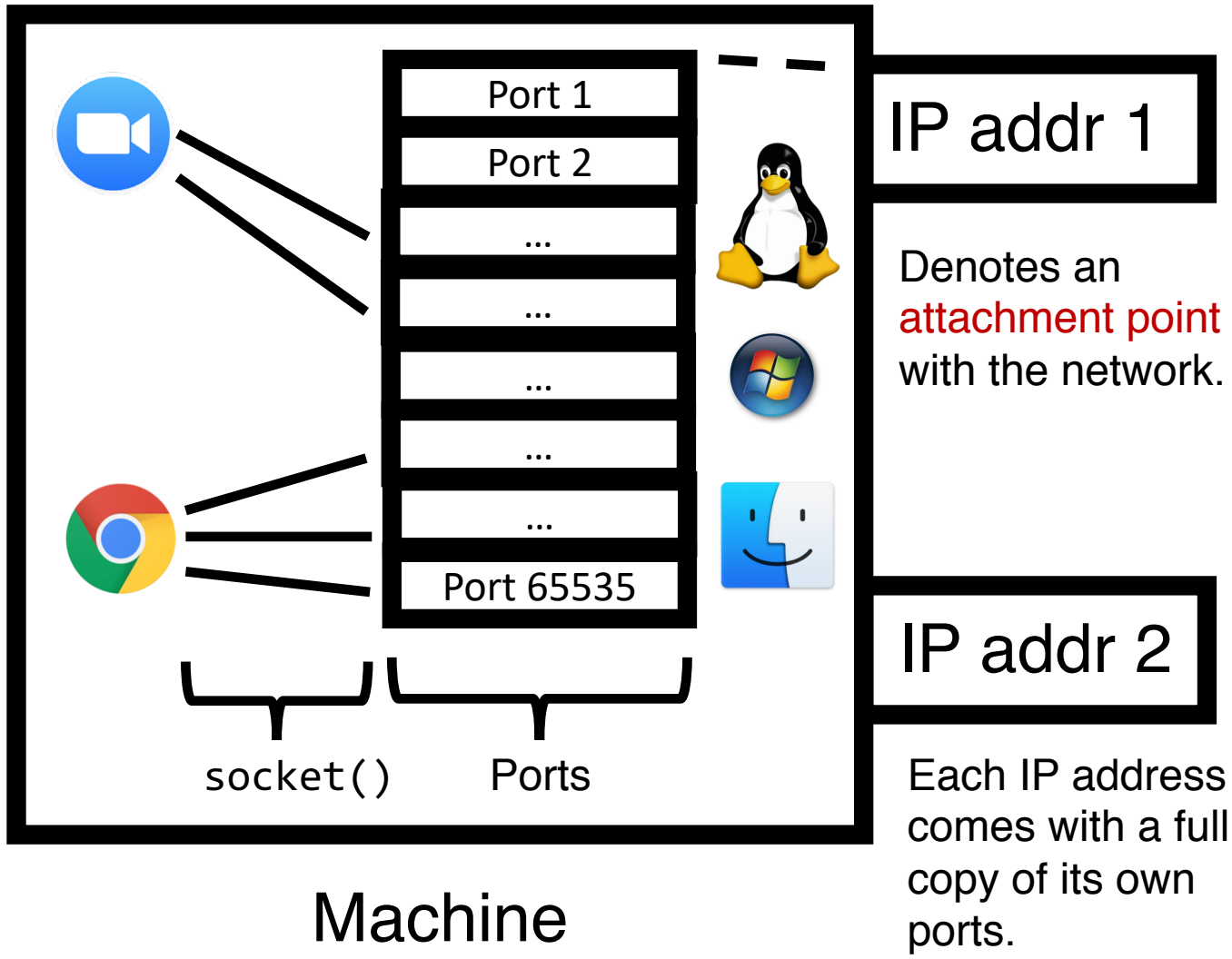


Denotes an attachment point with the network.

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



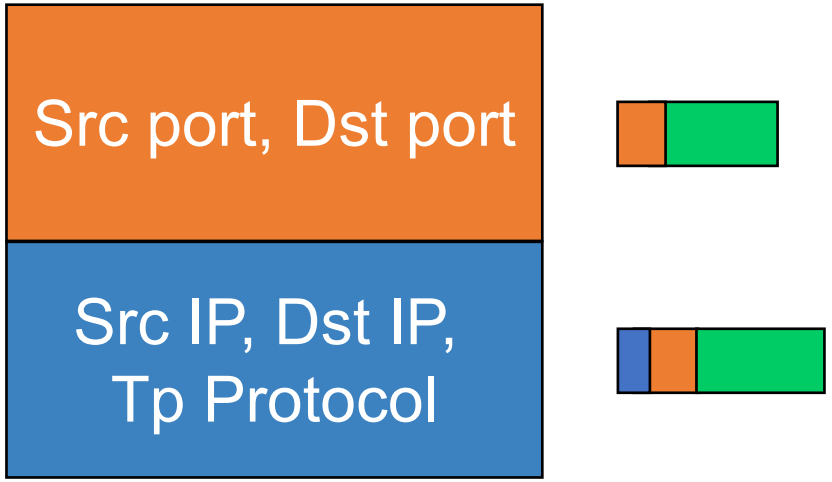
Demultiplexing



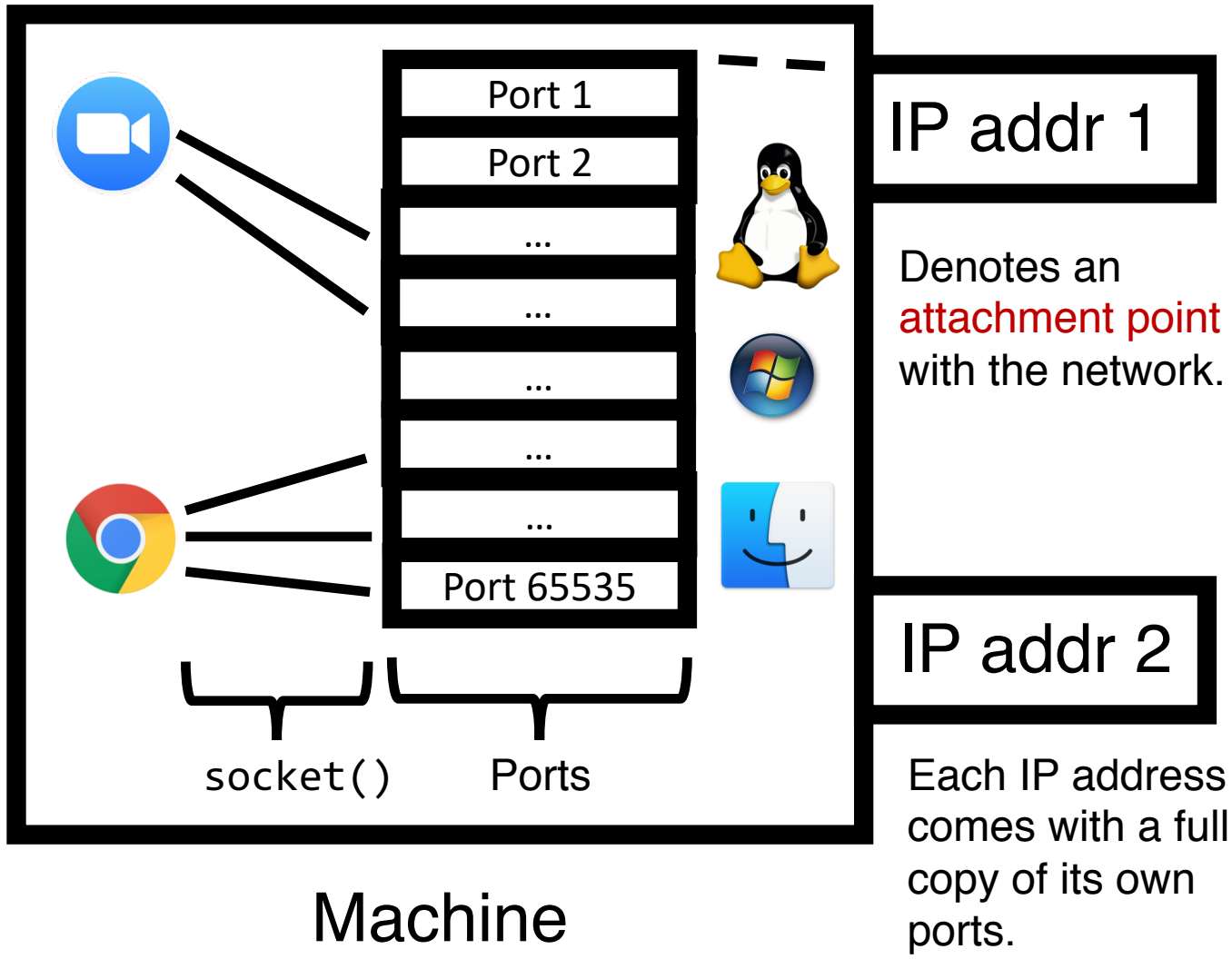
Denotes an attachment point with the network.

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.



Demultiplexing



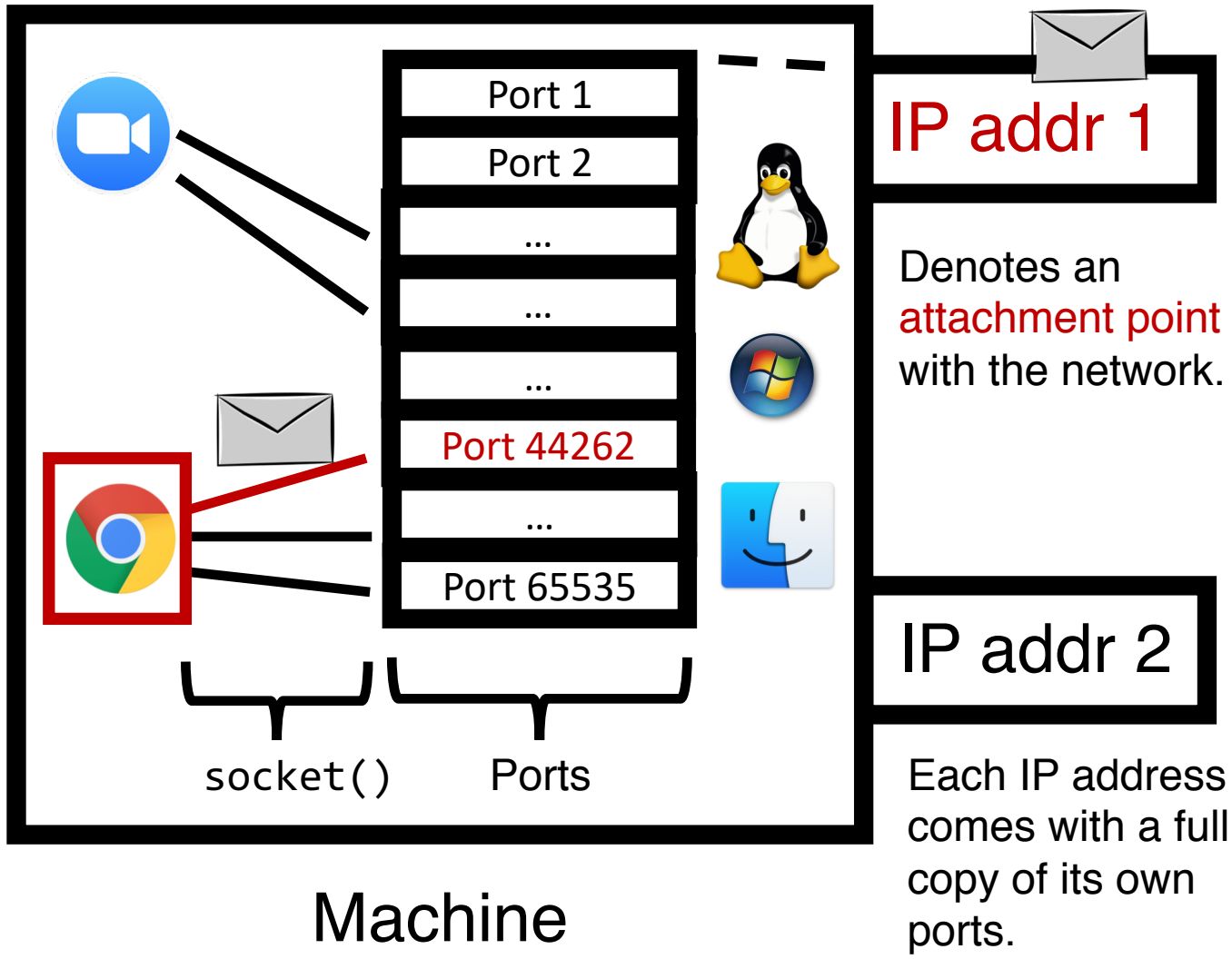
Denotes an attachment point with the network.

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

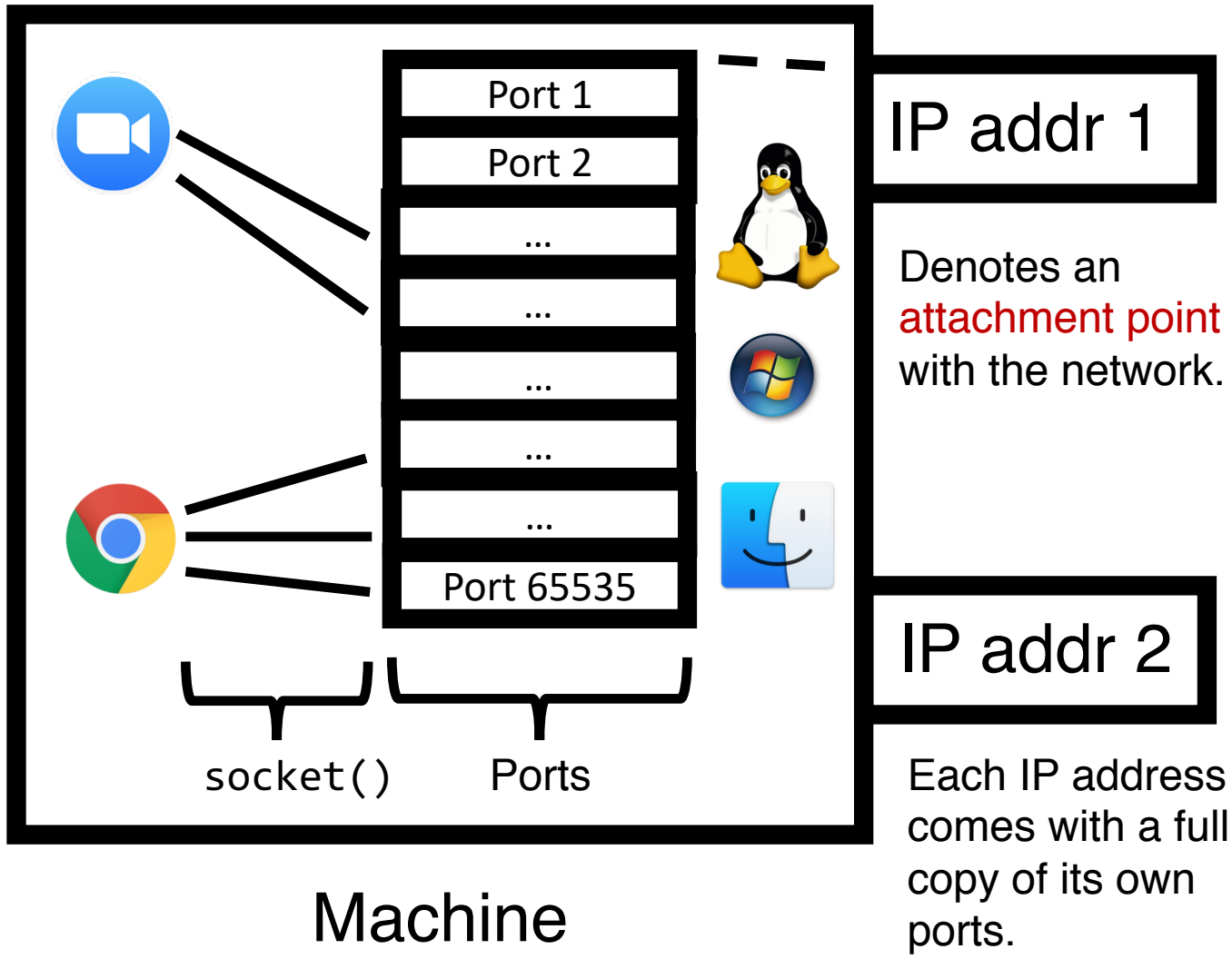
Demultiplexing



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.

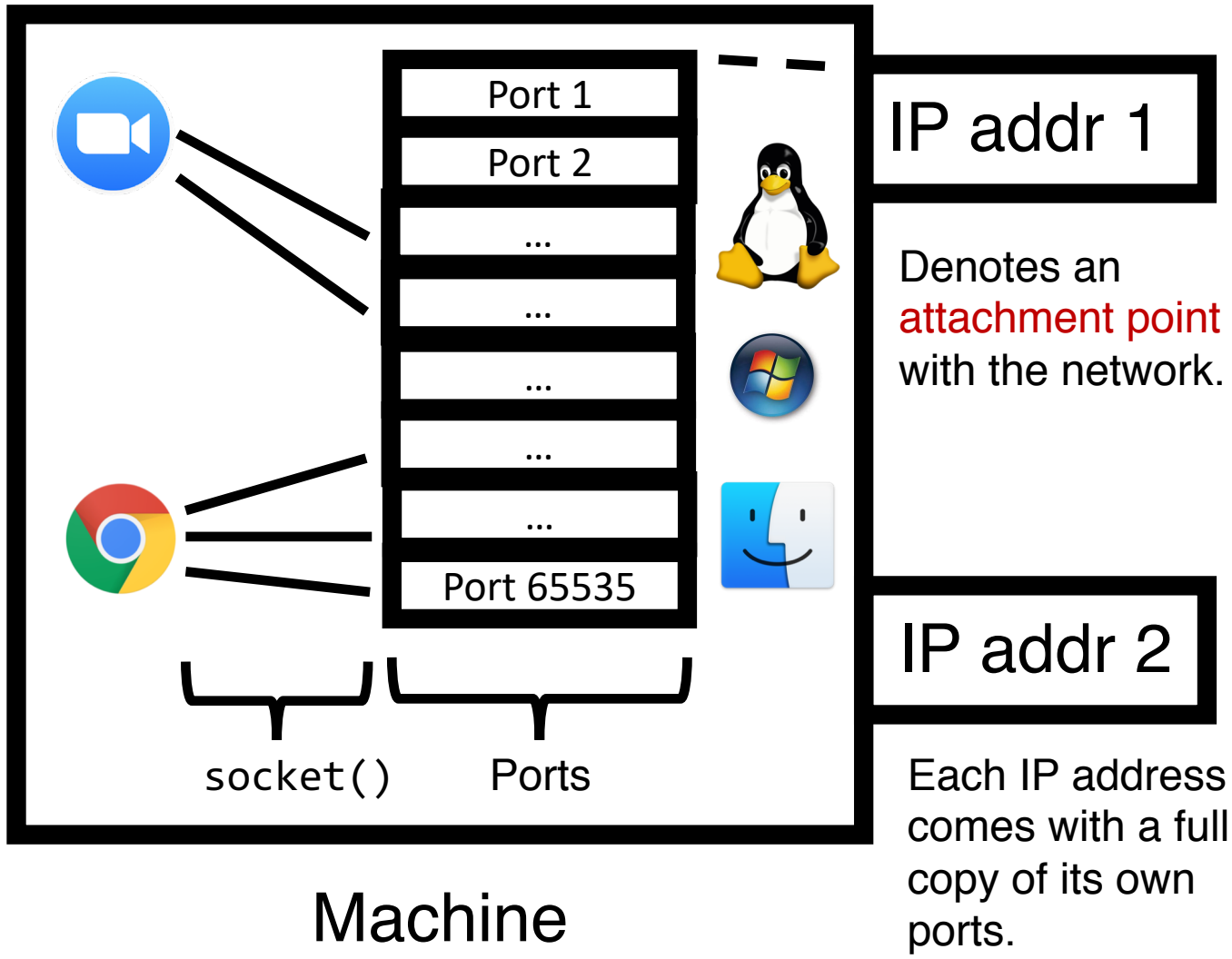
Connection lookup: The operating system does a lookup using these data to determine the right socket and app.

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

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

Connectionless: the socket is common 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** 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()

# 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

Listing sockets and connections

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