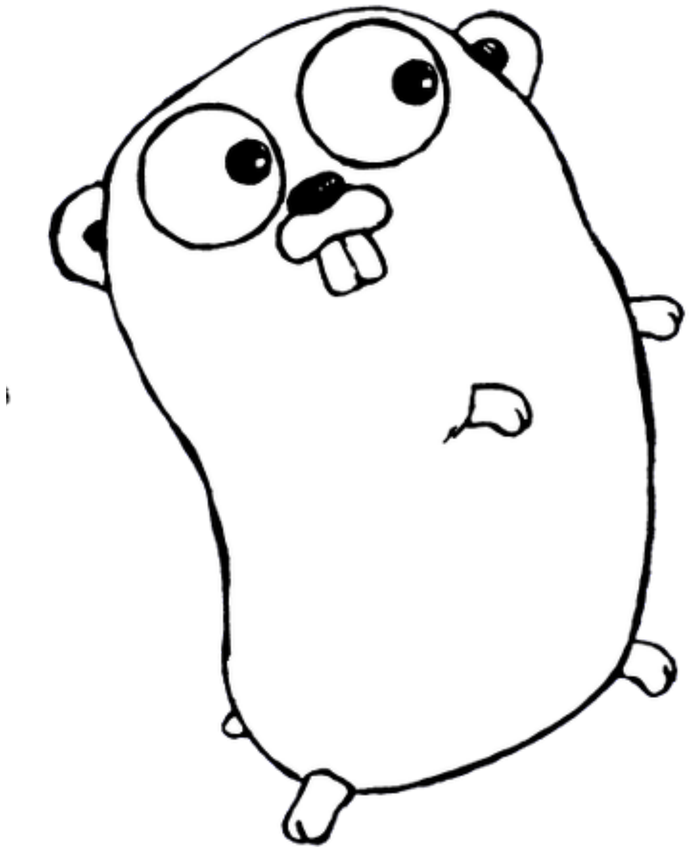
Network Principles for *Programmers*



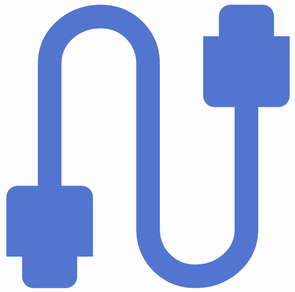
TheUnknownThing

Layers

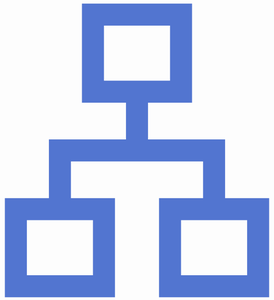
Layer			Protocol data unit (PDU)	Function
Host layers	7	Application	Data	High-level protocols such as for resource sharing or remote file access, e.g. HTTP.
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet, Datagram	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

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

Try Another View



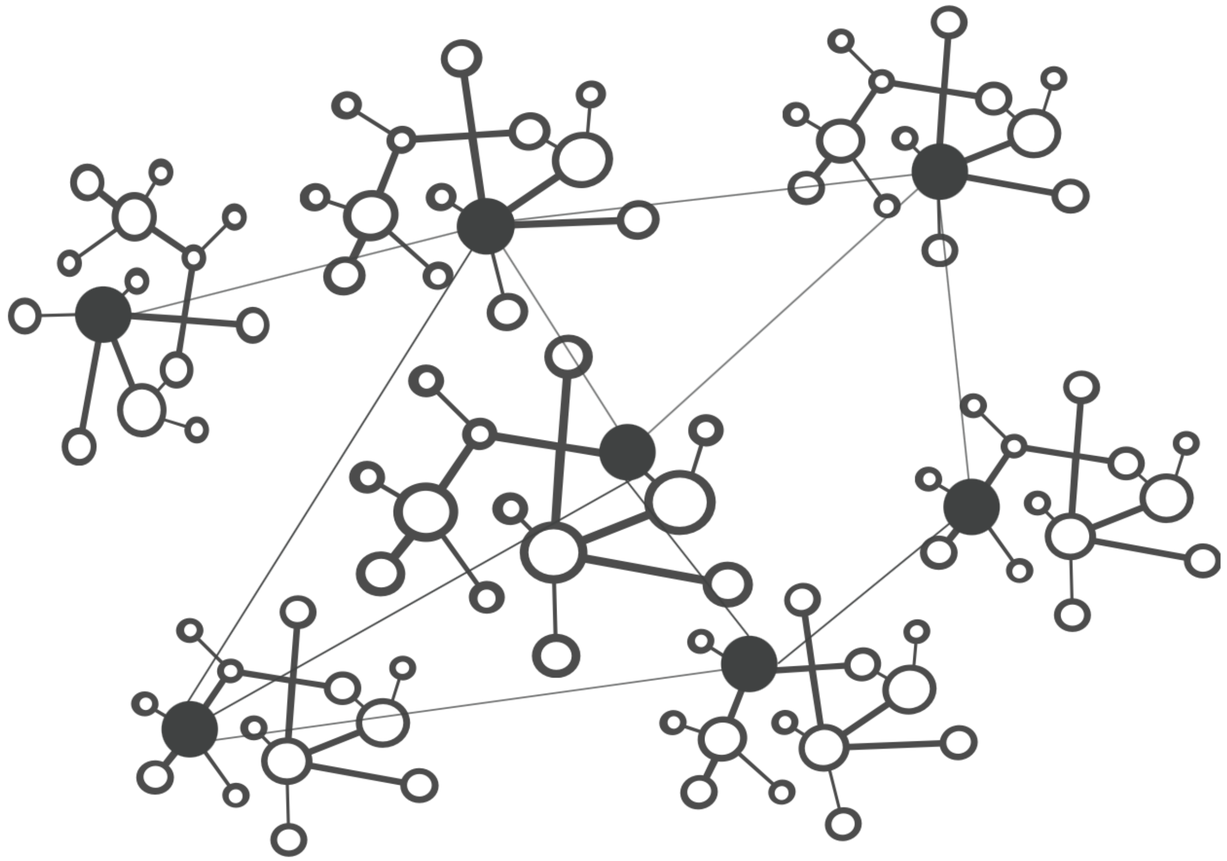
1



2

3

The Internet
A Network of Networks



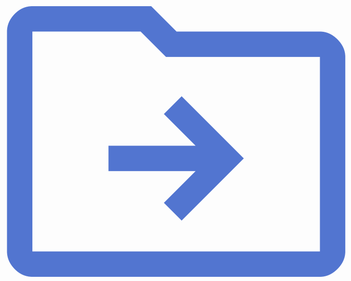
The Internet is…

Federated

End to End

Best Effort

Layers (CONT)



Session — Cookie; Auth Token; …

Presentation — JSON; HTML; …

4

7

5 & 6

Takeaways

Layer			Protocol data unit (PDU)	Function
Host layers	7	Application	Data	High-level protocols such as for resource sharing or remote file access, e.g. HTTP.
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet, Datagram	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

example.com ?

The user wants to access example.com!
But what's example.com?

Here's a packet to the Chrome browser.
But how to send this to Chrome browser?

Here's some traffic from a Linux machine.
But where is it? Where to send?

Someone is accessing my website.
What content do they need?

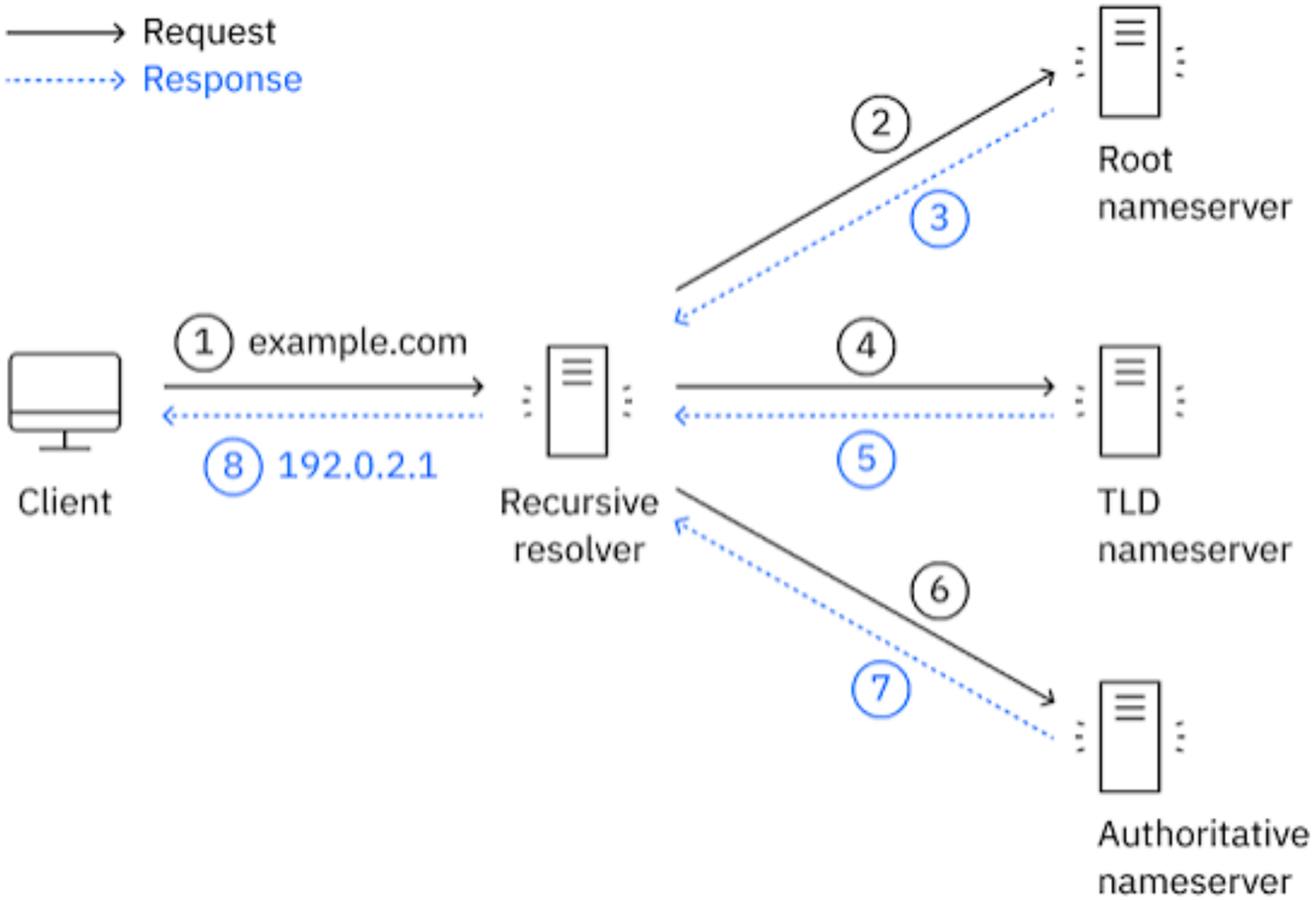
1. Who do I want to talk to?
2. How do my bytes get there?
3. How does the receiver know what I am talking about?
4. What if the network loses, reorders, or splits my data?

1. **Who do I want to talk to?**
2. How do my bytes get there?
3. How does the receiver know what I am talking about?
4. What if the network loses, reorders, or splits my data?

Communication needs ***Names***

The user wants to access example.com!
But what's example.com?

DNS: I'll tell you what's example.com


Request
Response
Client
1 example.com
8 192.0.2.1
Recursive resolver
2
3
Root nameserver
4
5
TLD nameserver
6
7
Authoritative nameserver

Experiment Time!

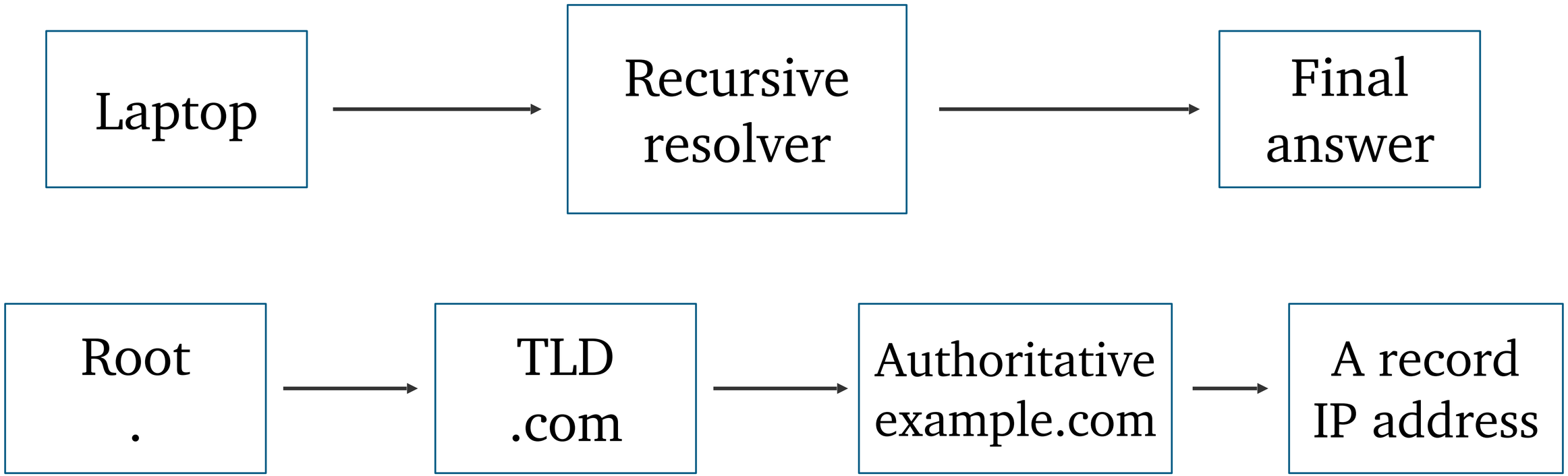
Run them in your WSL / macOS / Linux shell, please.
NO CMD / PowerShell.

```
nslookup example.com

dig example.com

curl -v http://example.com/

dig +trace example.com A
```

If there's no such command, install them:

```
apt install dnsutils
```

Recursive from the client's point of view



Root does not know every website

Ask root: what is example.com?

```
dig @a.root-servers.net example.com A +norecurse
```

Root usually replies:

```
com.   IN   NS   a.gtld-servers.net.
com.   IN   NS   b.gtld-servers.net.
...
```

Translation: “I do not know the final IP. Go ask .com.”

Authoritative server gives the answer

Ask the owner of example.com:

```
dig @a.iana-servers.net example.com A +norecurse
```

```
example.com.   300   IN   A   93.184.216.34
```

This server is authoritative for the zone, so it can give the final record.

Cache: DNS is not traced every time

If every lookup started from root, the Internet would be slow.

Resolvers cache answers for a limited time.

```
example.com.   300   IN   A   93.184.216.34
               ^
               TTL: cache this answer for about 300 seconds
```

The user wants to access example.com!
But what's example.com?

✅

Here's a packet to the Chrome browser.
But how to send this to Chrome browser?

Here's some traffic from a Linux machine.
But where is it? Where to send?

Someone is accessing my website.
What content do they need?

Here’s a packet to the Chrome browser.
But how to send this to Chrome browser?

I would look up something called “*Chrome*” in the task list?
Okay now your Chrome has opened 100 tabs…

Port: I’ll tell you where to send.

Port answers the OS question

A machine runs many programs.

The destination IP only identifies the machine.

The port helps the OS find the right network endpoint.

```
127.0.0.1:9330
          ^
          port
```

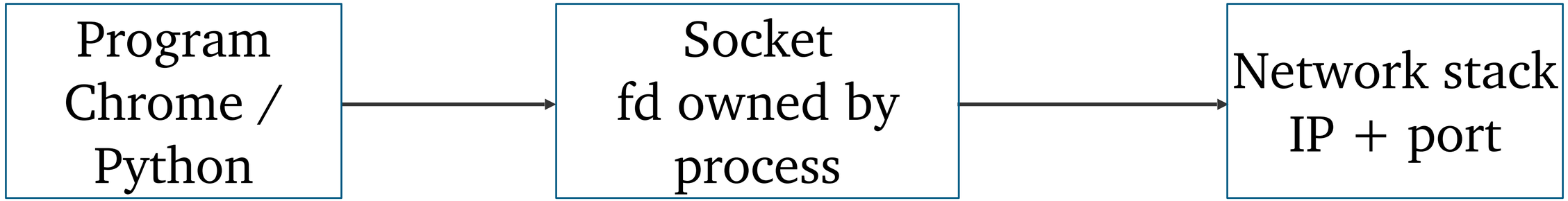
SSH	22
HTTP	80
HTTPS	443
DNS	53

Chrome send the packet for me.
But how should I send the packet?

Socket: Abstraction for sending things over network.

Socket is the bridge to the program

A socket is an OS object representing one communication endpoint.



Socket is the bridge to the program

```
server = socket()
server.bind(("0.0.0.0", 9330))
server.listen()

conn, addr = server.accept()
```

客服热线：

- 我弄了个客服电话号码
- 接线员上班

- 每当有人来了，就有专门的接线员去处理

Server-side sockets

```
server = socket()
server.bind(("0.0.0.0", 9330))
server.listen()

conn, addr = server.accept()
```

listen socket

waits for new clients on port 9330

connected socket

talks to one specific client

To learn more about the 2 sockets: https://theunknownth.ing/blog/socket/

Experiment: Find the Socket Owner

```
# Terminal 1
python3 -m http.server 9330

# Terminal 2
curl http://127.0.0.1:9330/

# Terminal 3
lsof -nP -iTCP:9330
```

Look for:

- COMMAND: which program?
- PID: which process?
- FD: which socket handle?
- LISTEN / ESTABLISHED: what state?

The user wants to access example.com!
But what's example.com?

✅

Here's a packet to the Chrome browser.
But how to send this to Chrome browser?

✅

Here's some traffic from a Linux machine.
But where is it? Where to send?

Someone is accessing my website.
What content do they need?

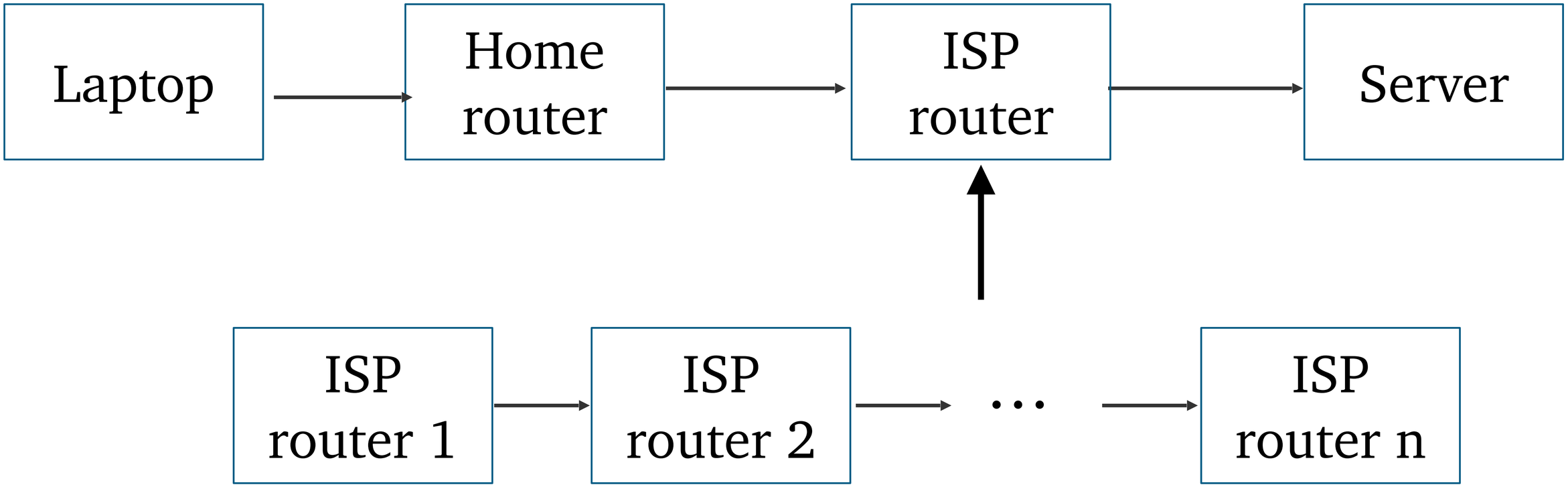
1. Who do I want to talk to?
2. **How do my bytes get there?**
3. How does the receiver know what I am talking about?
4. What if the network loses, reorders, or splits my data?

Communication needs ***Delivery***

Communication needs ***Local Delivery***

Communication needs ***Global Delivery***

Each hop chooses only the next hop



The routing table is the local map

```
# Linux
ip route get 1.1.1.1

# macOS
route -n get 1.1.1.1
```

```
1.1.1.1 via 192.168.1.1 dev wlan0 src 192.168.1.23
```

Translation: to reach 1.1.1.1, send the packet to 192.168.1.1 first.

So How that Table is Produced?

In case you are curious about it, check out my blog!

https://theunknownth.ing/blog/internet-protocol/#routing

And my previous talk of “*How computer network route your packets*”

Local delivery uses MAC addresses

Did you still remember Layer 2?

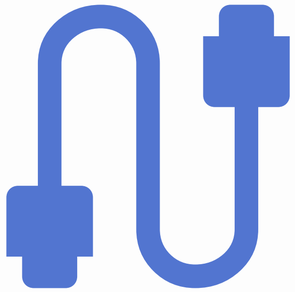
Different Layers speak different languages

IP is an “Layer 3 language”. Network switch don’t speak this!

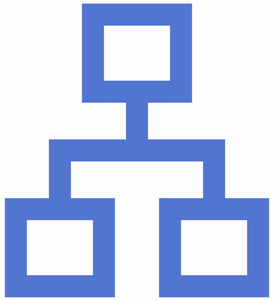
Okay so why we don’t speak the same language?

The Chicken and Egg Paradox

Try Another View



1



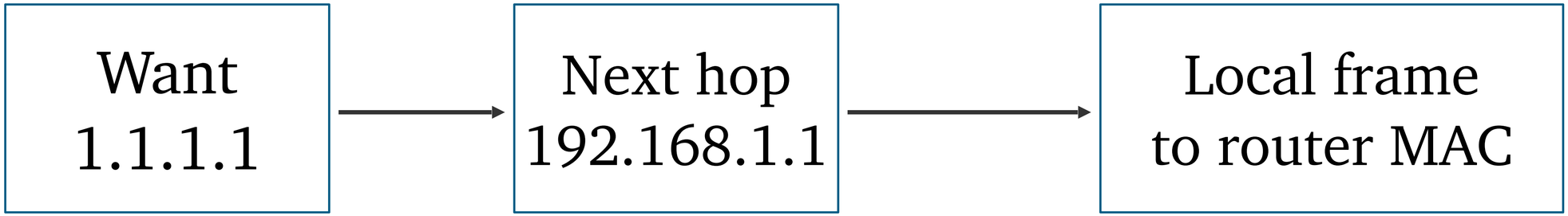
2

3

Local delivery uses MAC addresses

IP tells us the destination across networks.

MAC tells us the next device on this local link.




```
arp -a
# or
ip neigh
```

Traceroute: watch the next hops

It discovers routers by sending packets with increasing TTL. Try them in your terminal!

```
# Linux / macOS
traceroute example.com

# Windows equivalent
tracert example.com
```

```
# Some better alternative: nexttrace
# Installation:
curl -sL https://nxtrace.org/nt | bash
# Usage:
nexttrace example.com
```

Delivery is best effort

It does *not* promise:

- delivery
- order
- no duplicates
- constant latency

Higher layers decide what guarantees they need.

Recap: The Internet is…

Federated

End to End

Best Effort

Checkpoint: delivery

DNS gives an IP.

The routing table chooses a next hop.

Routers repeat this decision until the packet reaches the destination network.

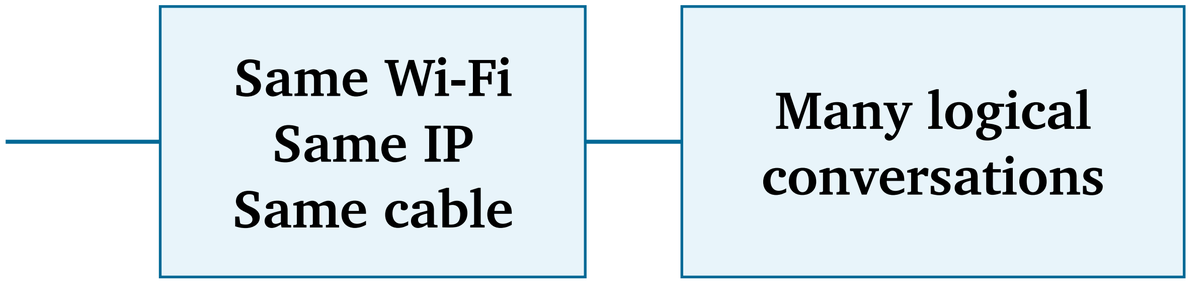
Delivery is hop-by-hop.

Communication needs ***Multiplexing***

Communication needs Multiplexing

One network is shared by many conversations.

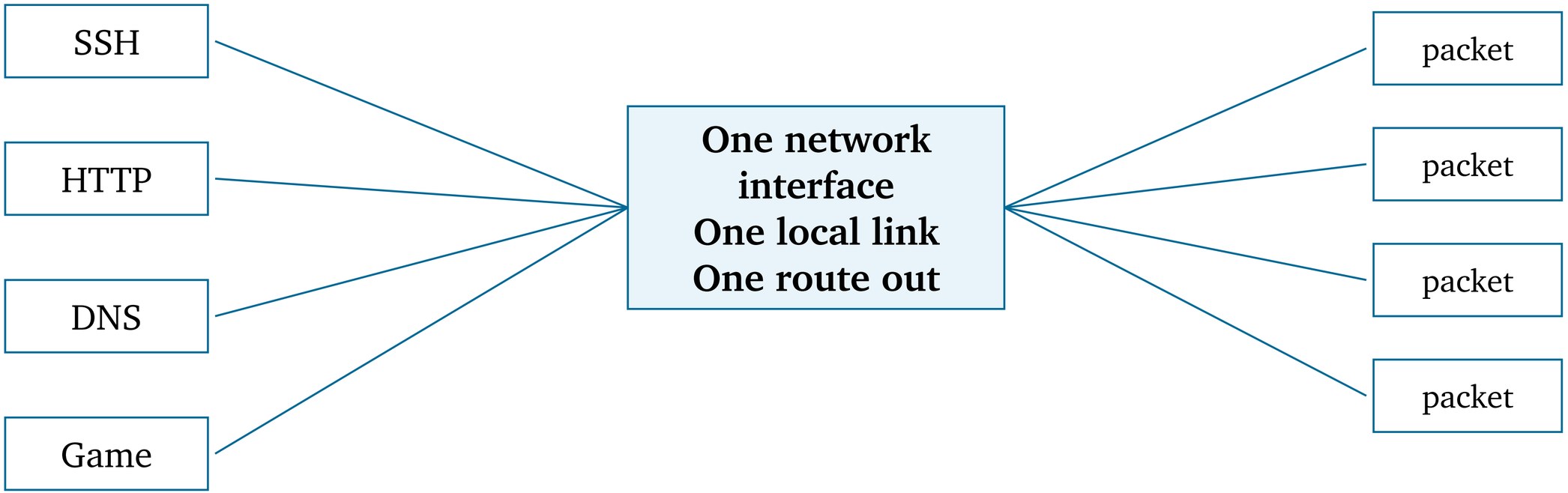
Chrome tab #1
Chrome tab #2
VSCode update
SSH session
WeChat / Discord / game



Question: how does the network avoid mixing them together?

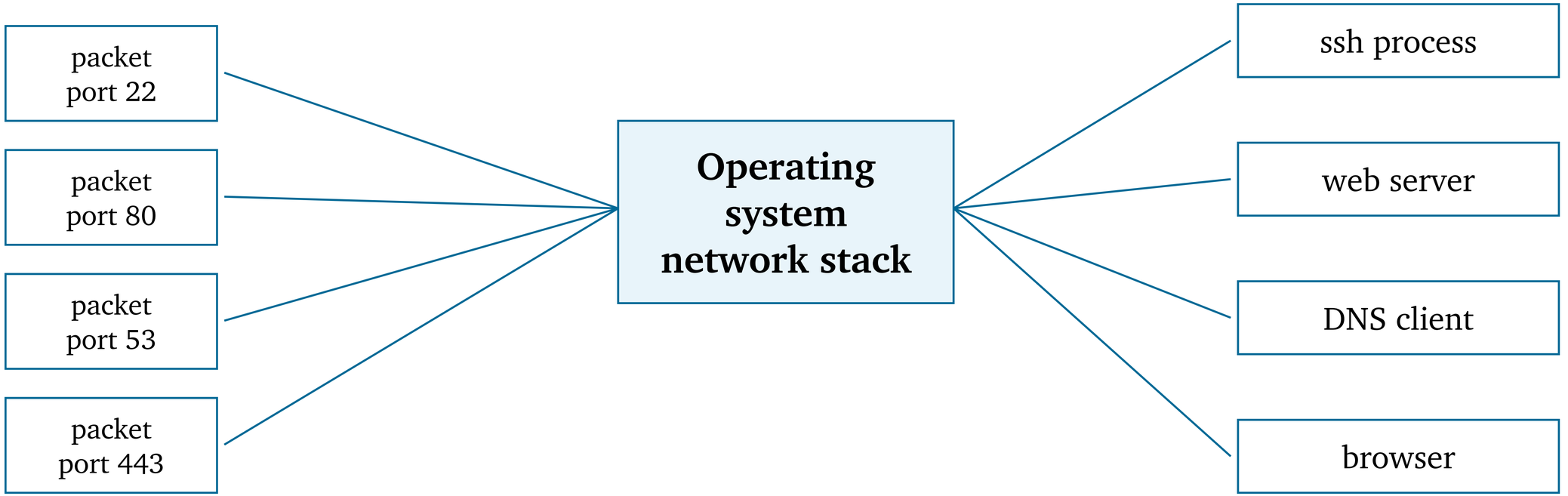
Many conversations share one path

Multiplexing means combining many logical streams onto one shared resource.



Multiplexing and Demultiplexing

Demultiplexing means using ***headers*** to decide which receiver should get each packet.

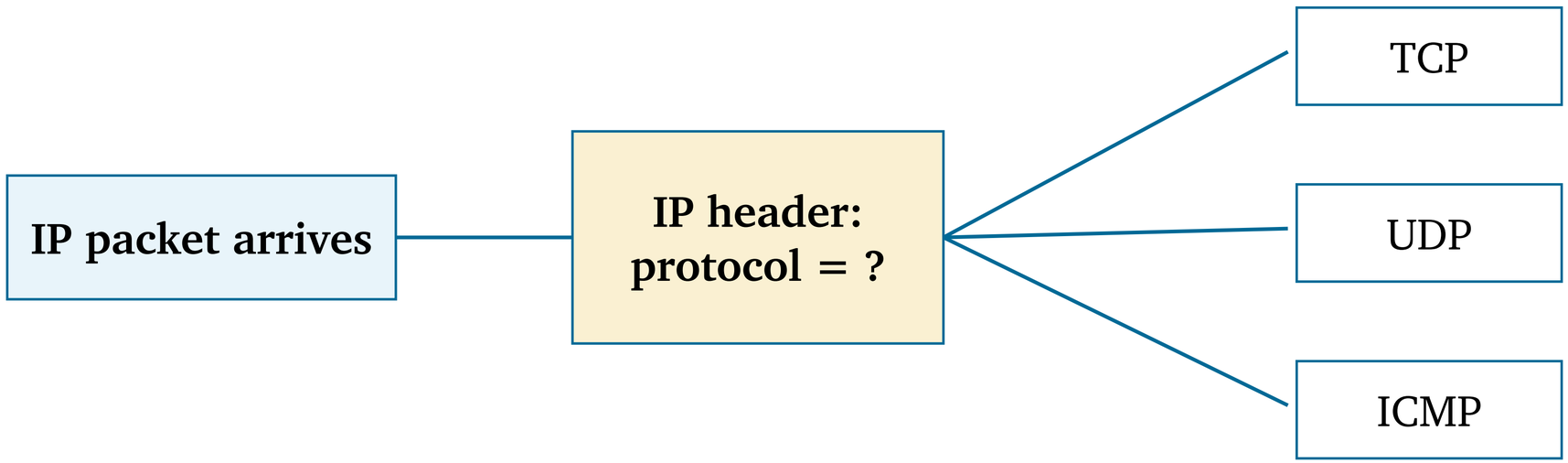


матрёшка

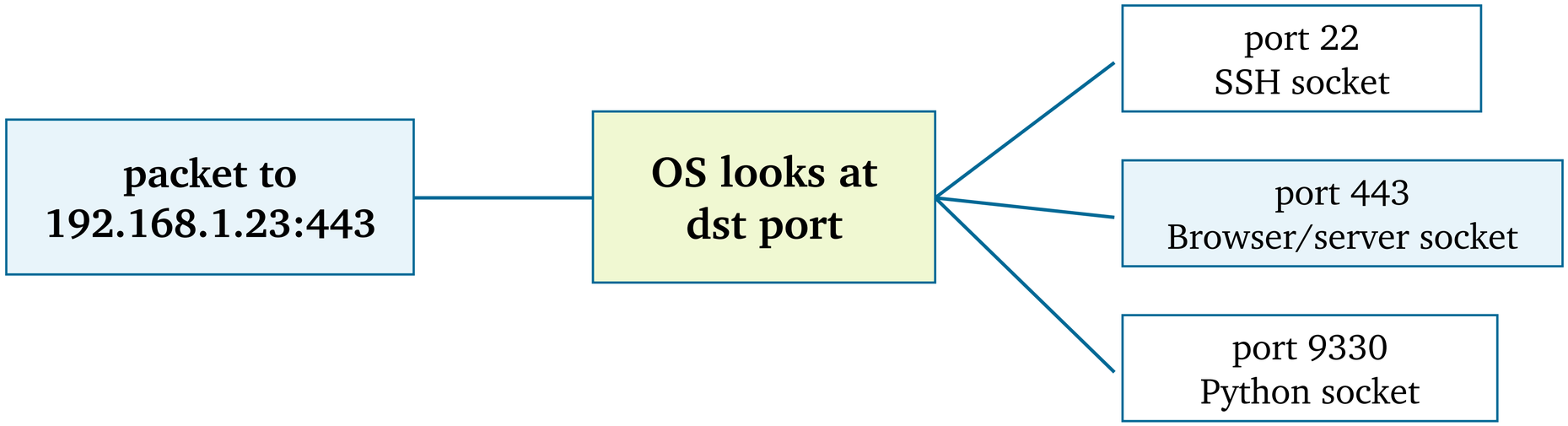
Each layer wrap the packet and adds a small label for identification.

Ethernet header Who on this local link?	IP header Which host?	TCP/UDP header Which socket?	HTTP data Which website / request?

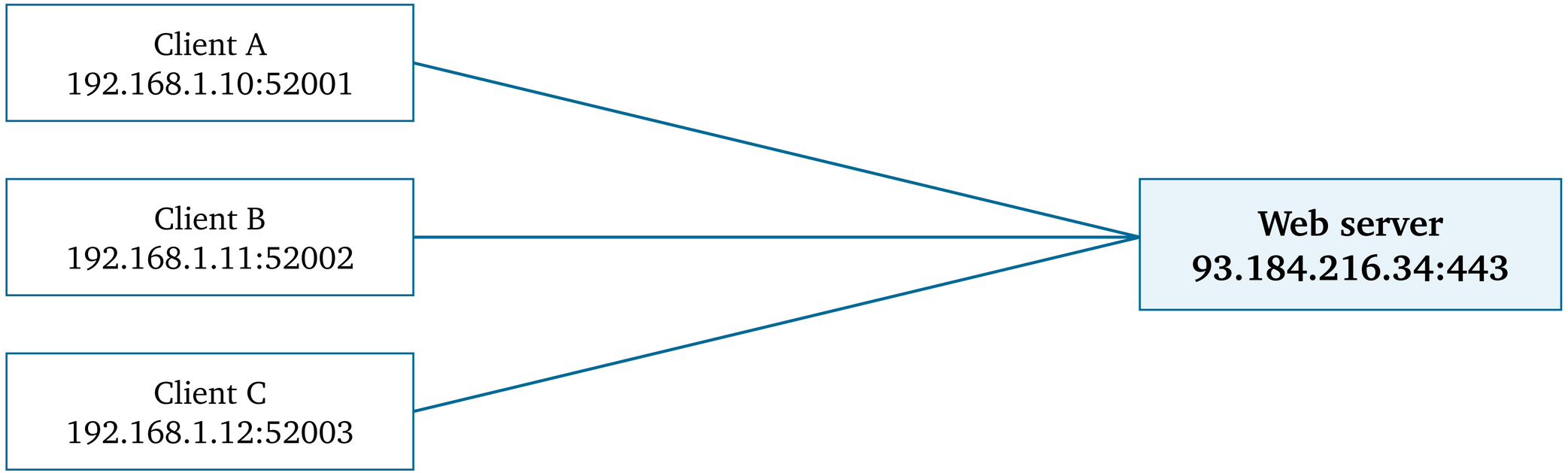
Layer 3 demux: IP protocol number



Layer 4 demux: ports choose the socket



Layer 7 demux: Different Client IP and Port



A TCP connection has a full name

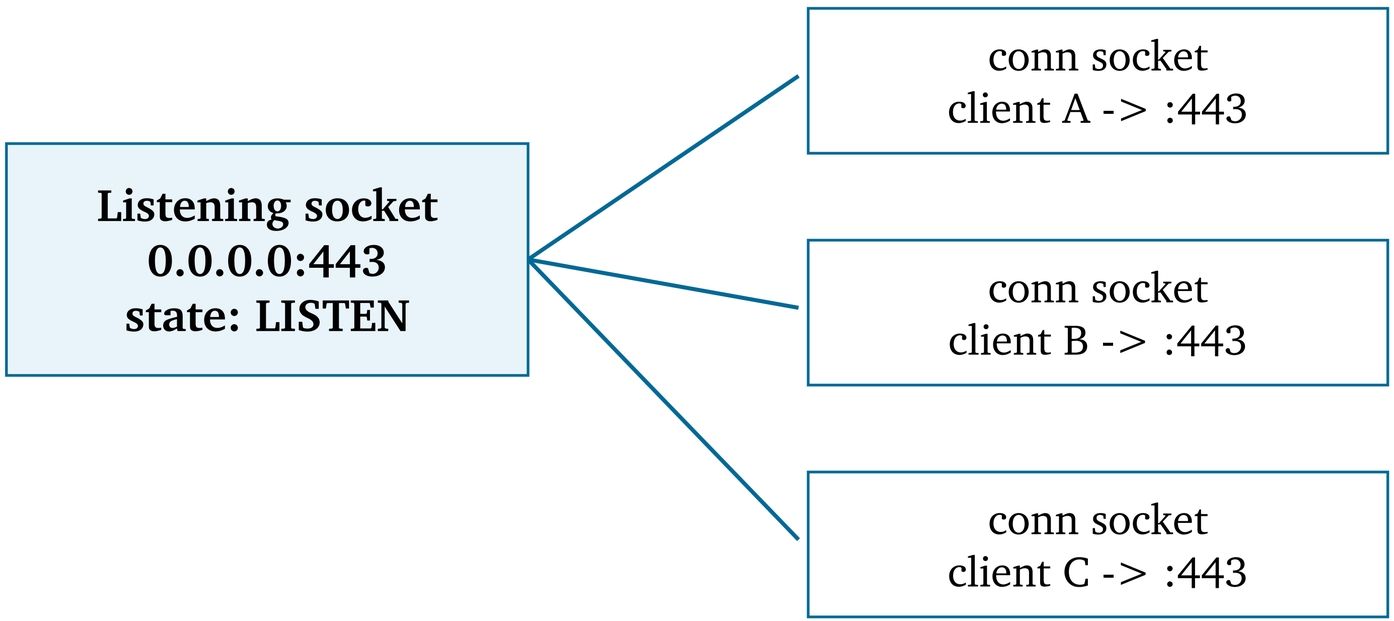
A TCP connection is not identified by the server port alone.

```
TCP connection name =
source IP + source port + destination IP + destination port
```

```
Example: 192.168.1.10:52001 → 93.184.216.34:443
```

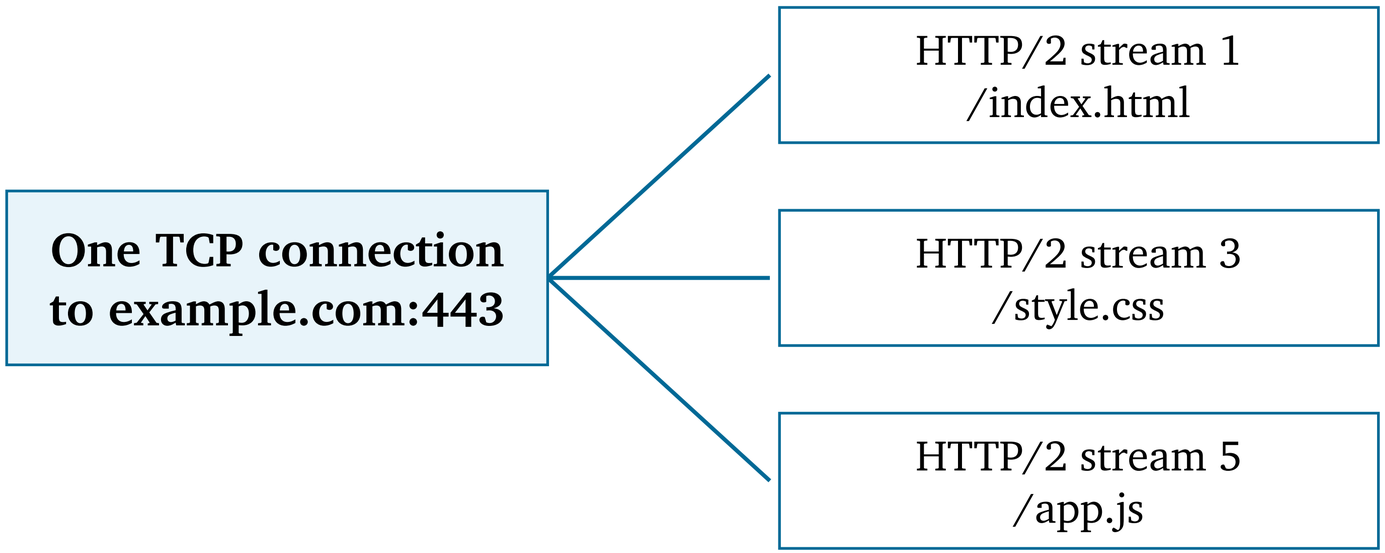
This is often called the 4-tuple.

Multiplexing (1)



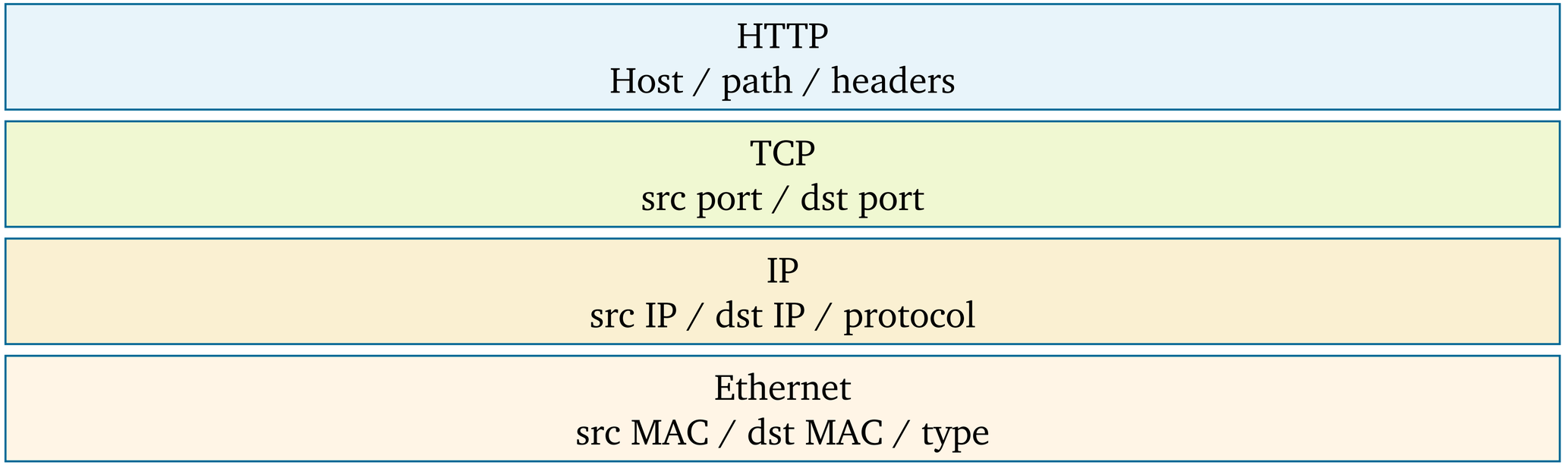

```
server = socket()
server.bind(("0.0.0.0", 443))
server.listen()

conn, addr = server.accept()
```

Multiplexing (2)



Same TCP connection.
Multiple application streams.

Multiplexing Across Layers



Sending: encapsulate. Receiving: decapsulate and demultiplex.

The user wants to access example.com!
But what's example.com?

✅

Here's a packet to the Chrome browser.
But how to send this to Chrome browser?

✅

Here's some traffic from a Linux machine.
But where is it? Where to send?

✅

Someone is accessing my website.
What content do they need?

1. Who do I want to talk to?
2. How do my bytes get there?
3. **How does the receiver know what I am talking about?**
4. What if the network loses, reorders, or splits my data?

Communication needs ***Agreement***

Bytes do not explain themselves

The receiver gets bytes.

But bytes need rules to become meaning.

```
47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Protocol = shared agreement about how to interpret bytes.

HTTP is a human-readable agreement

```
GET /index.html HTTP/1.1
Host: example.com
User-Agent: curl/8.0
Accept: */*
```

- Request line: what method and path?
- Headers: extra metadata.
- Blank line: headers are finished.
- Body: optional data after the headers.

Message boundaries are important

HTTP needs rules to know where a message ends.

```
HTTP/1.1 200 OK
Content-Length: 12

Hello World!
```

Without agreement, bytes are just bytes.

Session meaning lives above packets

HTTP is stateless by default.

A website remembers you using application-level names.

```
Cookie: session_id=abc123
Authorization: Bearer ...
```

This is another layer of naming: user/session identity.

Experiment Time: Speak HTTP

```
# Terminal 1
nc example.com 80
```

- What happens if you omit the Host header?
- What happens if you request a different path?
- Where does the HTTP response start and end?

Checkpoint: agreement

- A protocol tells the program what the bytes mean.
- HTTP uses request lines, headers, blank lines, and bodies.
- Session cookies and tokens are application-level names.

Communication needs ***Trust***

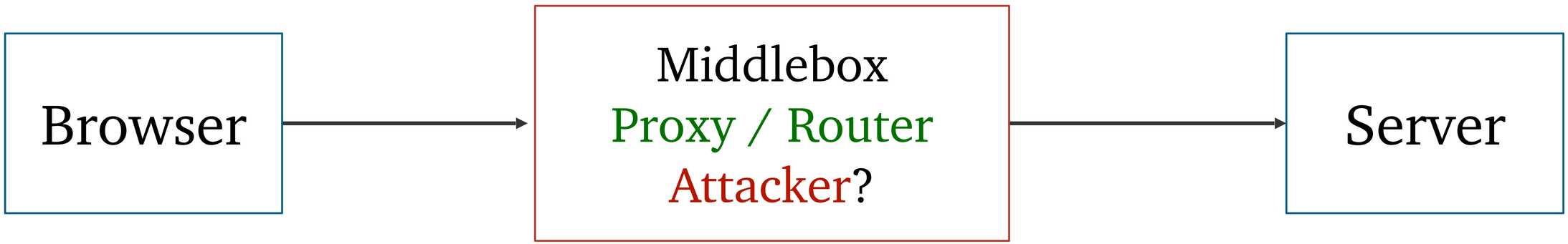
A Dangerous Question

So far, we know how to reach an address and open a connection.
But a dangerous question remains:

“Am I really talking to who I think I am?”

Successful delivery does not imply trust.

A middleman may be useful, or dangerous



HTTP is easy to understand, but...

```
GET / HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: text/html

<h1>Hello</h1>
```

That is great for learning. **It is terrible for trust.**

Trust has three separate meanings

Confidentiality Can others read it?	Integrity Has it been changed?	Authentication Who am I talking to?

Trust != Encryption

HTTPS is HTTP inside a TLS tunnel

TLS adds *encryption*, *integrity checks*, and *server authentication*.

Certificates answer a different question

DNS
Where should I connect?

Certificate
Who are you?

Browser checks:

- certificate is valid for example.com
- certificate is signed by a trusted CA
- certificate is not expired

```
# Check the CA yourself!
ls /etc/ssl/certs
```

MITM against HTTPS needs fake trust

Normal HTTPS:

HTTPS MITM after installing MITM CA:

The browser accepts the fake certificate only if it trusts the MITM CA.

Experiment Time: what should you observe?

```
# HTTP: easy to inspect
curl -v http://example.com/

# HTTPS: certificate and TLS appear
curl -v https://example.com/
```

Checkpoint: trust

- HTTP is readable and modifiable by a middleman.
- HTTPS is HTTP over TLS.
- TLS protects confidentiality, integrity, and authentication.
- DNS tells where to connect; certificates help prove who answered.
- MITM is powerful only when it can sit in the path and gain trust.

The user wants to access example.com!
But what's example.com?

✅

Here's a packet to the Chrome browser.
But how to send this to Chrome browser?

✅

Here's some traffic from a Linux machine.
But where is it? Where to send?

✅

Someone is accessing my website.
What content do they need?

✅

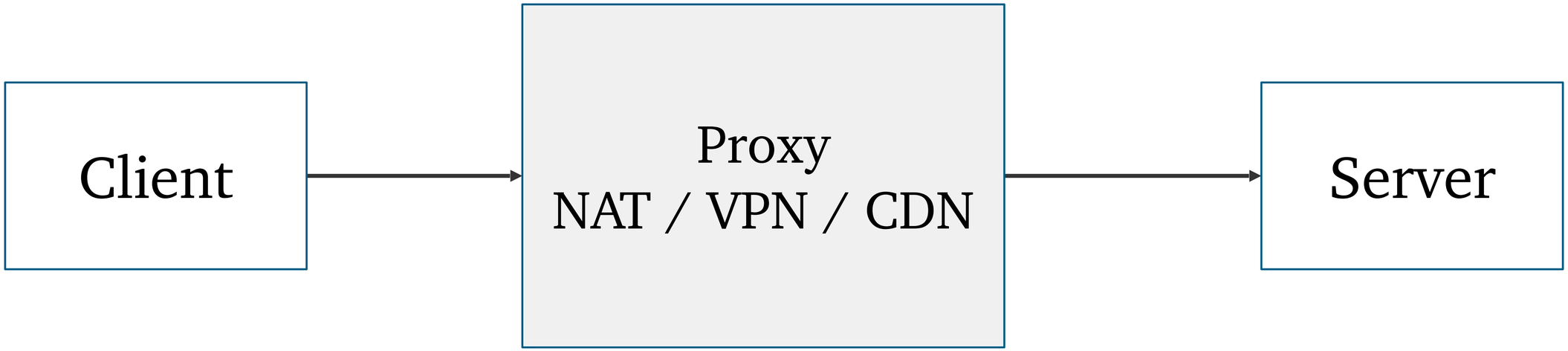
1. Who do I want to talk to?
2. How do my bytes get there?
3. How does the receiver know what I am talking about?
4. **What if the network loses, reorders, or splits my data?**

Communication needs ***Indirection***

Direct communication is the simplest story

But many real networks are not this simple.

Indirect communication



The client talks to the middlebox. The middlebox talks to the server.

A proxy splits conversation into 2

```
Without proxy:
  laptop:11451  →  example.com:80

With SOCKS5 proxy:
  laptop:11451  →  proxy:1080
  proxy:19198   →  example.com:80
```

The server sees the proxy as the peer.

Indirection changes who each side believes it is talking to.

SOCKS5: “connect for me”

The client does not ask the proxy for a web page.

It asks the proxy to open a TCP connection.

```
Client → SOCKS5 proxy:
  Please connect to example.com:80

Proxy → example.com:
  opens a TCP connection
```

Okay... But am I Safe? (Recap)

Normal HTTPS:

HTTPS MITM after installing MITM CA:

Okay... But am I Safe?

Normal Connection:

After applying a SOCKS5 Proxy:

Are they the same? NO.

HTTPS MITM after installing MITM CA:

After applying a SOCKS5 Proxy:

Experiment Time

You'll probably need to test your SOCKS5 proxy by this!

```
# Direct connection
curl -v http://example.com/

# Through SOCKS5
curl -v --socks5 127.0.0.1:1080 http://example.com/
```

Indirection appears everywhere

- NAT: many devices share one public IP.
- VPN: your traffic exits somewhere else.
- CDN: a nearby server answers for a global website.
- Load balancer: one public name maps to many backend servers.
- Proxy: policy, caching, logging, filtering, or reachability.

Indirection and trust are connected

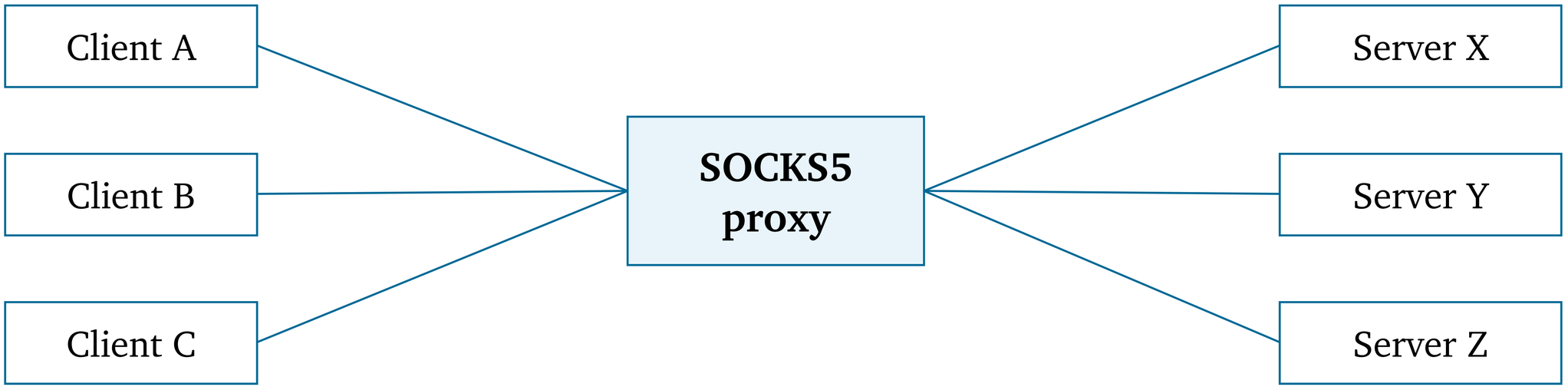
A middlebox may be intentional:

- SOCKS5 proxy
- company proxy
- CDN edge

Or it may be hostile:

- MITM attacker
- malicious Wi-Fi access point
- fake DNS / fake certificate

Multiplexing also explains proxies



Checkpoint: indirection

- Sometimes your peer is not the final destination.
- A proxy usually turns one end-to-end connection into two connections.
- SOCKS5 is a protocol between client the proxy server to relay bytes.
- Indirection helps with reachability, policy, caching, and load balancing.

SOCKS5 lab: build controlled indirection.

Communication needs ***Reliability Choices***

Recap: The Internet is…

Federated

End to End

Best Effort

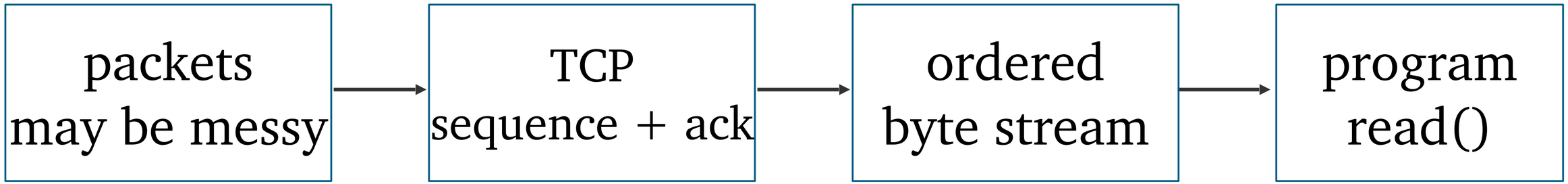
We may need several attempts

The network may:

- drop a packet
- deliver packets out of order
- duplicate a packet
- split data into multiple packets

The application usually does not want to handle all of this!

TCP Provides a Reliable Byte Stream



TCP hides many packet-level problems from the application.

Reliability is built from small mechanisms

- Sequence numbers: where does this byte belong?
- Acknowledgements: what has arrived?
- Retransmission: send again if probably lost.
- Flow control: do not overwhelm the receiver.
- Congestion control: do not overwhelm the network.

You do not need the full algorithm today. But if you are interested, check out how ***BBR*** (developed by Google) is implemented!

Meaning of Preserve Byte Order

Program A may call:

```
send("hello")
send("world")
```

Program B may receive:

```
recv() → "helloworld"
# or
recv() → "hel"
recv() → "loworld"
```

UDP makes a different choice

UDP gives **datagrams**.

It *preserves message boundaries*, but gives fewer guarantees.

- No connection setup.
- No built-in retransmission.
- No built-in ordering.
- Useful when the application wants control.

TCP or UDP is a design choice

TCP
web pages
SSH
file transfer

UDP
DNS
video calls
games

Reliable byte stream vs **lightweight datagrams.**

Experiment Time: TCP Stream

```
import socket

s = socket.socket()
s.connect(("example.com", 80))
s.sendall(b"GET / HTTP/1.1\r\nHost: example.com\r\n\r\n")

while True:
    data = s.recv(10)
    if not data: break
    print(repr(data))
```

Checkpoint: reliability choices

- IP is best effort.
- TCP builds a reliable ordered byte stream.
- TCP does not preserve application message boundaries.
- UDP preserves datagram boundaries but gives fewer guarantees.
- Applications choose the abstraction they need.

The four questions again

- Who do I want to talk to? Names: domain, IP, port, socket.
- How do my bytes get there? Delivery: routing, next hop, MAC.
- How does the receiver know what I mean? Agreement: HTTP, headers, cookies.
- Can I trust the peer? Trust: HTTPS, TLS, certificates, MITM.
- What if the network is messy? Reliability choices: TCP or UDP.
- What if I talk through someone else? Indirection: SOCKS5, proxy, NAT, CDN.

Final Takeaways

- Networking is ***practical***: protocols exist because people needed them.
- The same problem appears at many layers.
- ***Names***, ***delivery***, ***agreement***, and ***reliability*** are the core concepts to answer some questions.
- Experiments are more useful than memorizing the OSI table!
- Get your hands dirty, and have fun 🤩!