

[544] Networking and RPCs

Tyler Caraza-Harter

Outline

Networks

Internets and "The Internet"

Transport Protocols

Application Protocols

Network Interface Controllers and MAC Addresses



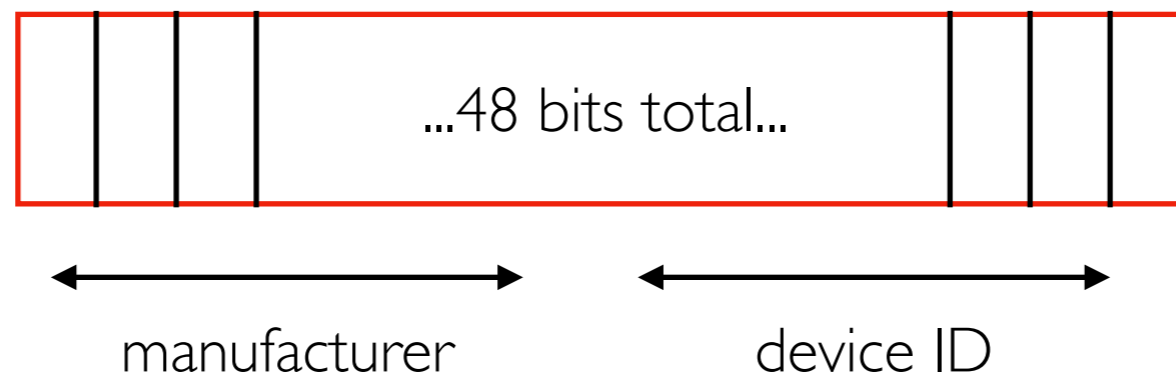
NICs can connect a computer to different physical mediums, such as:

- Ethernet (wired)
- Wi-Fi (wireless)

Every NIC in the world has a unique MAC (media access control) address

- 28 trillion possible addrs
- some devices randomly change their MAC addr for privacy

MAC address:



ifconfig (Interface Config)

interface

MAC address

```
trh@instance-1:~$ ifconfig
ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
    inet 10.128.0.36 netmask 255.255.255.255 broadcast 0.0.0.0
    inet6 fe80::4001:aff:fe80:24 prefixlen 64 scopeid 0x20<link>
    ether 42:01:0a:80:00:24 txqueuelen 1000 (Ethernet)
    RX packets 2332795 bytes 6456770667 (6.4 GB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1994116 bytes 718670305 (718.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 407321 bytes 417582056 (417.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 407321 bytes 417582056 (417.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

`sudo apt install net-tools`

Virtual Interfaces

```
trh@instance-1:~$ ifconfig
ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
    inet 10.128.0.36 netmask 255.255.255.255 broadcast 0.0.0.0
    inet6 fe80::4001:aff:fe80:24 prefixlen 64 scopeid 0x20<link>
    ether 42:01:0a:80:00:24 txqueuelen 1000 (Ethernet)
    RX packets 2332795 bytes 6456770667 (6.4 GB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1994116 bytes 718670305 (718.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 407321 bytes 417582056 (417.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 407321 bytes 417582056 (417.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

loopback (lo) device a virtual (not actual hardware) interface connecting to a mini network containing just your computer

Google Console: Adding Interfaces (NICs)

Create Instance > Advanced Options > Networking

Network interfaces

Network interface is permanent

default default (10.128.0.0/20)	▼
other-net subnet (10.0.0.0/24)	▼
ADD NETWORK INTERFACE	

<input type="checkbox"/>	<input checked="" type="checkbox"/>	instance-2	us-central1-a	10.128.0.37 (nic0)	34.29.220.248 (nic0)
				10.0.0.2 (nic1)	35.202.74.234 (nic1)

Google Console: Adding Interfaces

Create Instance > Advanced Options > Networking

Network interfaces 

```
trh@instance-2:~$ ifconfig
```

```
ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
  inet 10.128.0.37 netmask 255.255.255.255 broadcast 0.0.0.0
  inet6 fe80::4001:aff:fe80:25 prefixlen 64 scopeid 0x20<link>
  ether 42:01:0a:80:00:25 txqueuelen 1000 (Ethernet)
  RX packets 637 bytes 546000 (546.0 KB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 612 bytes 97265 (97.2 KB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
ens5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
  inet 10.0.0.2 netmask 255.255.255.255 broadcast 0.0.0.0
  inet6 fe80::4001:aff:fe00:2 prefixlen 64 scopeid 0x20<link>
  ether 42:01:0a:00:00:02 txqueuelen 1000 (Ethernet)
  RX packets 51 bytes 9955 (9.9 KB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 61 bytes 6834 (6.8 KB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

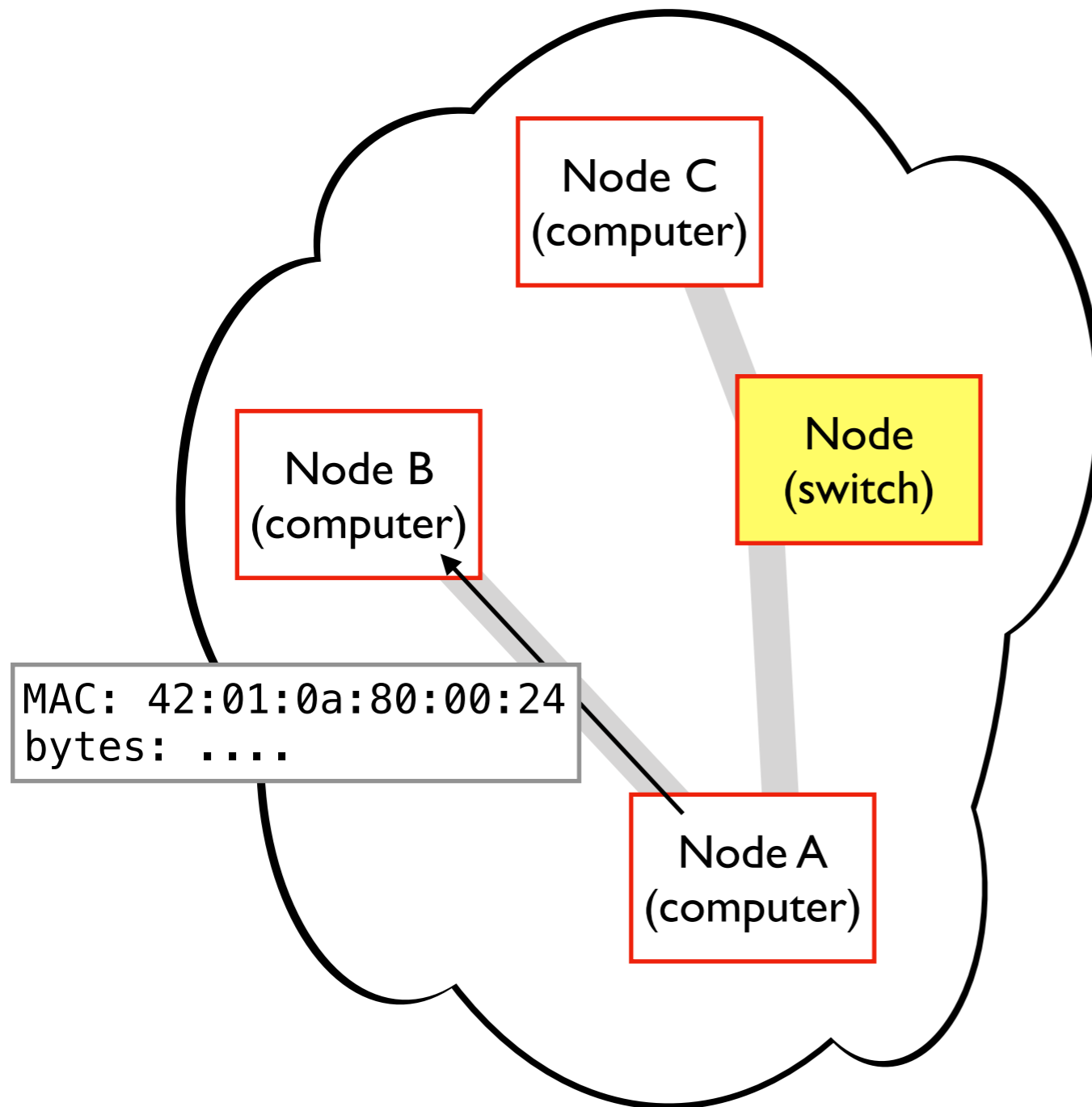
```
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
  inet 127.0.0.1 netmask 255.0.0.0
  inet6 ::1 prefixlen 128 scopeid 0x10<host>
  loop txqueuelen 1000 (Local Loopback)
  RX packets 120 bytes 13534 (13.5 KB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 120 bytes 13534 (13.5 KB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

7	34.29.220.248 (nic0) 35.202.74.234 (nic1)
---	--

Networks

A **network** has **nodes** that send bytes to other nodes **by MAC address**

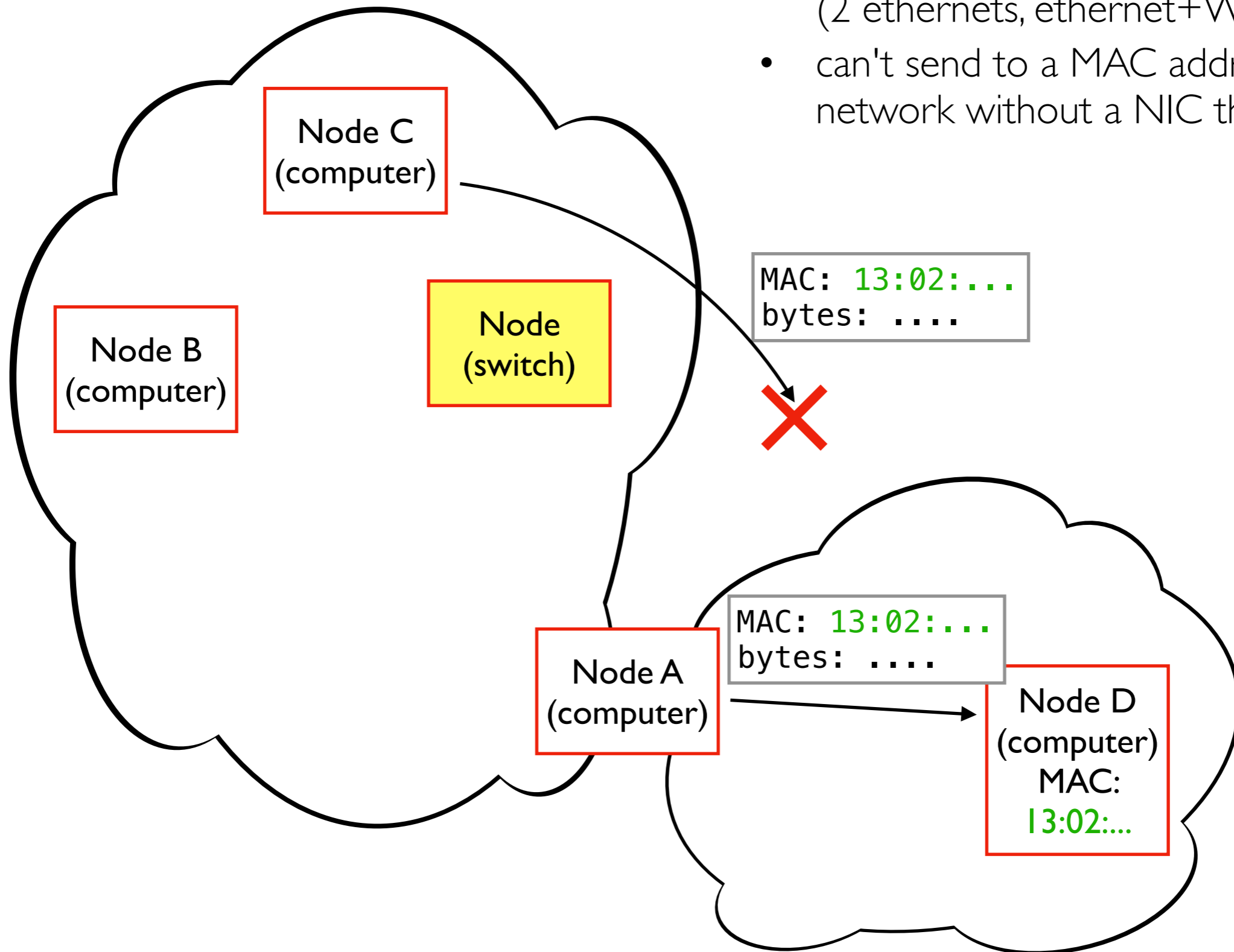
- **nodes**: computer, switch, etc
- direct, or **forwarded by switches**
- whole network uses same physical tech (Wi-Fi, Ethernet, etc)



Networks

Computers can have multiple NICs

- can be on multiple networks (2 ethernets, ethernet+Wi-Fi, etc)
- can't send to a MAC addr in another network without a NIC there



Outline

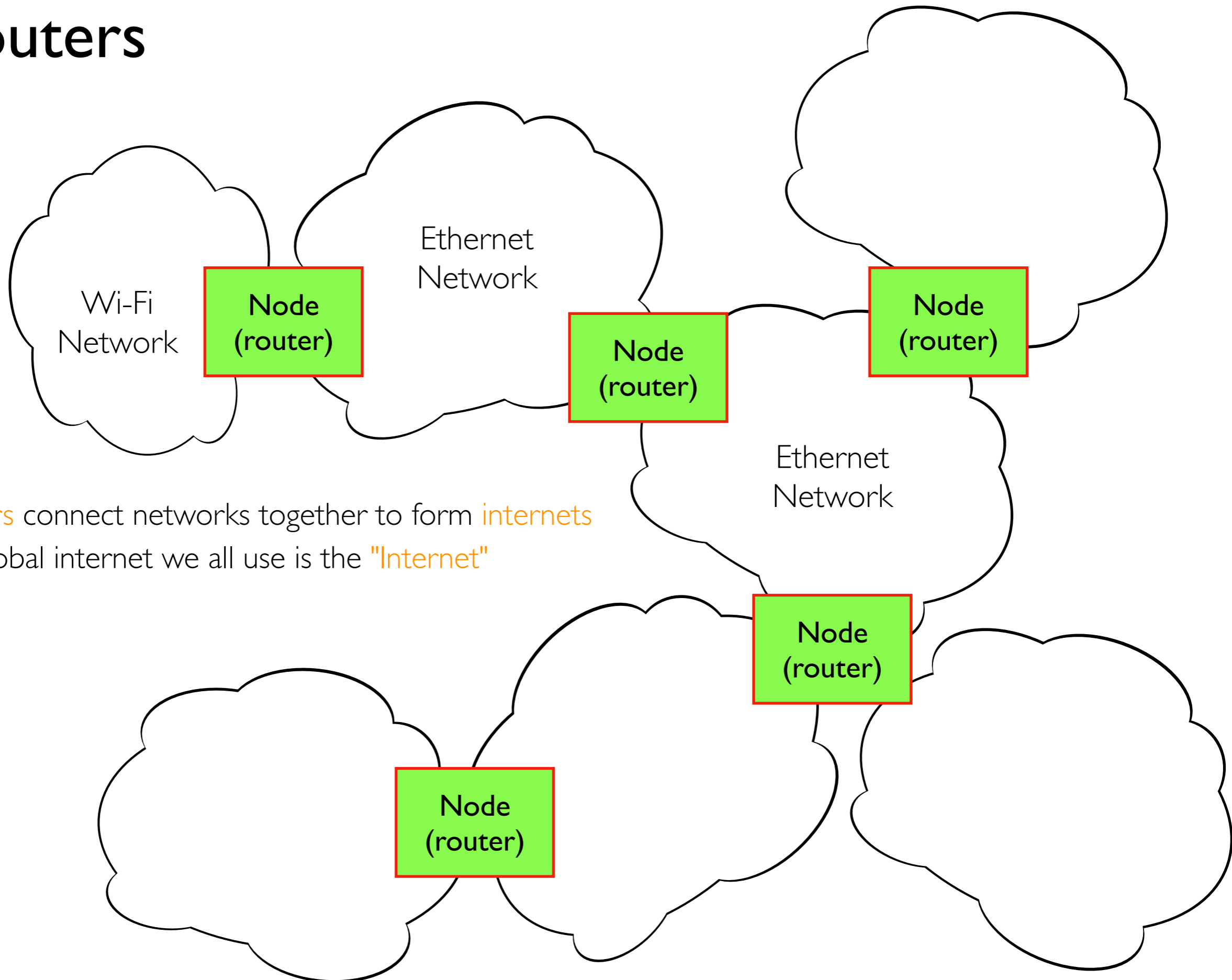
Networks

Internets and "The Internet"

Transport Protocols

Application Protocols

Routers

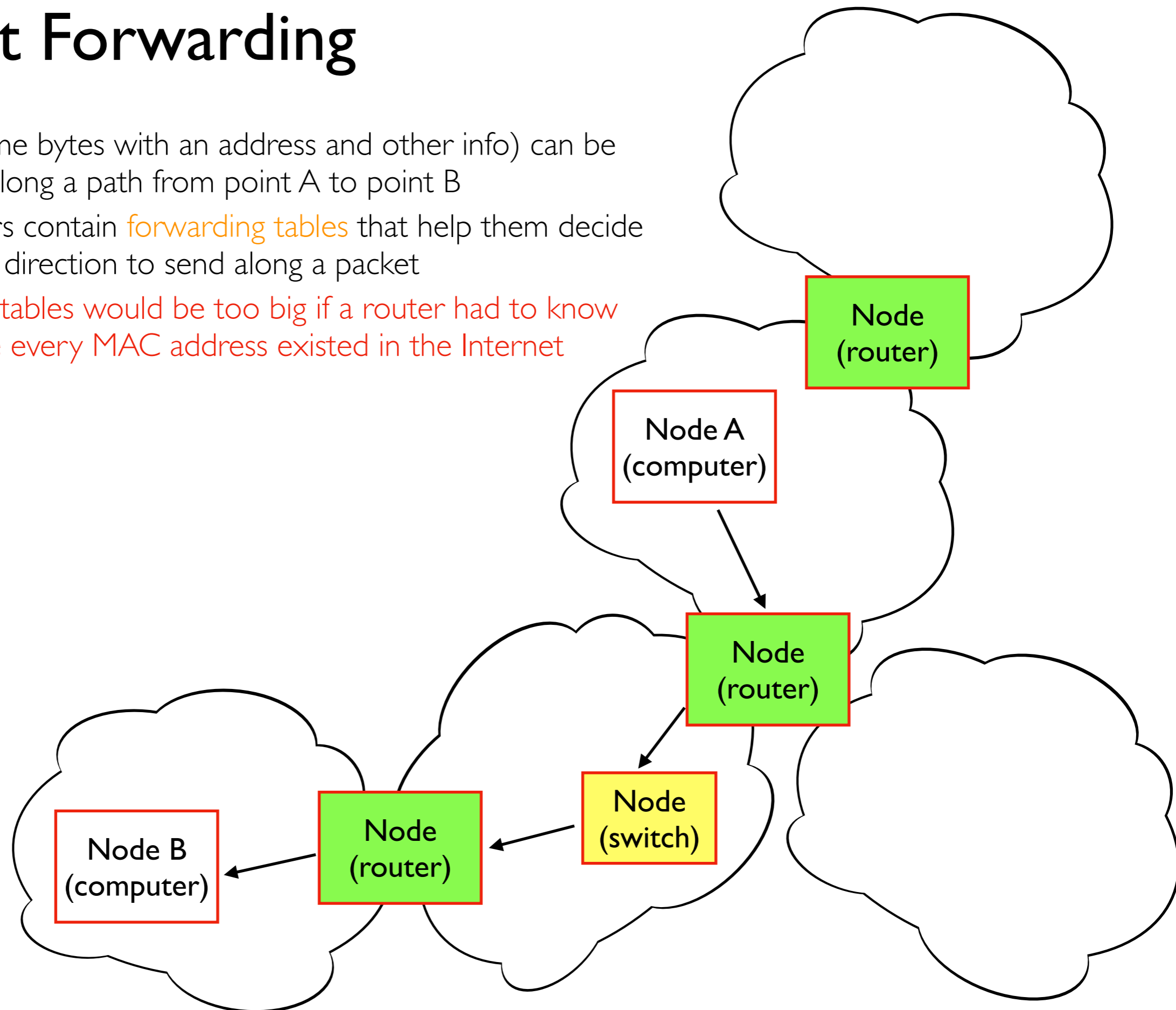


- routers connect networks together to form internets
- the global internet we all use is the "Internet"

Packet Forwarding

Packets (some bytes with an address and other info) can be forwarded along a path from point A to point B

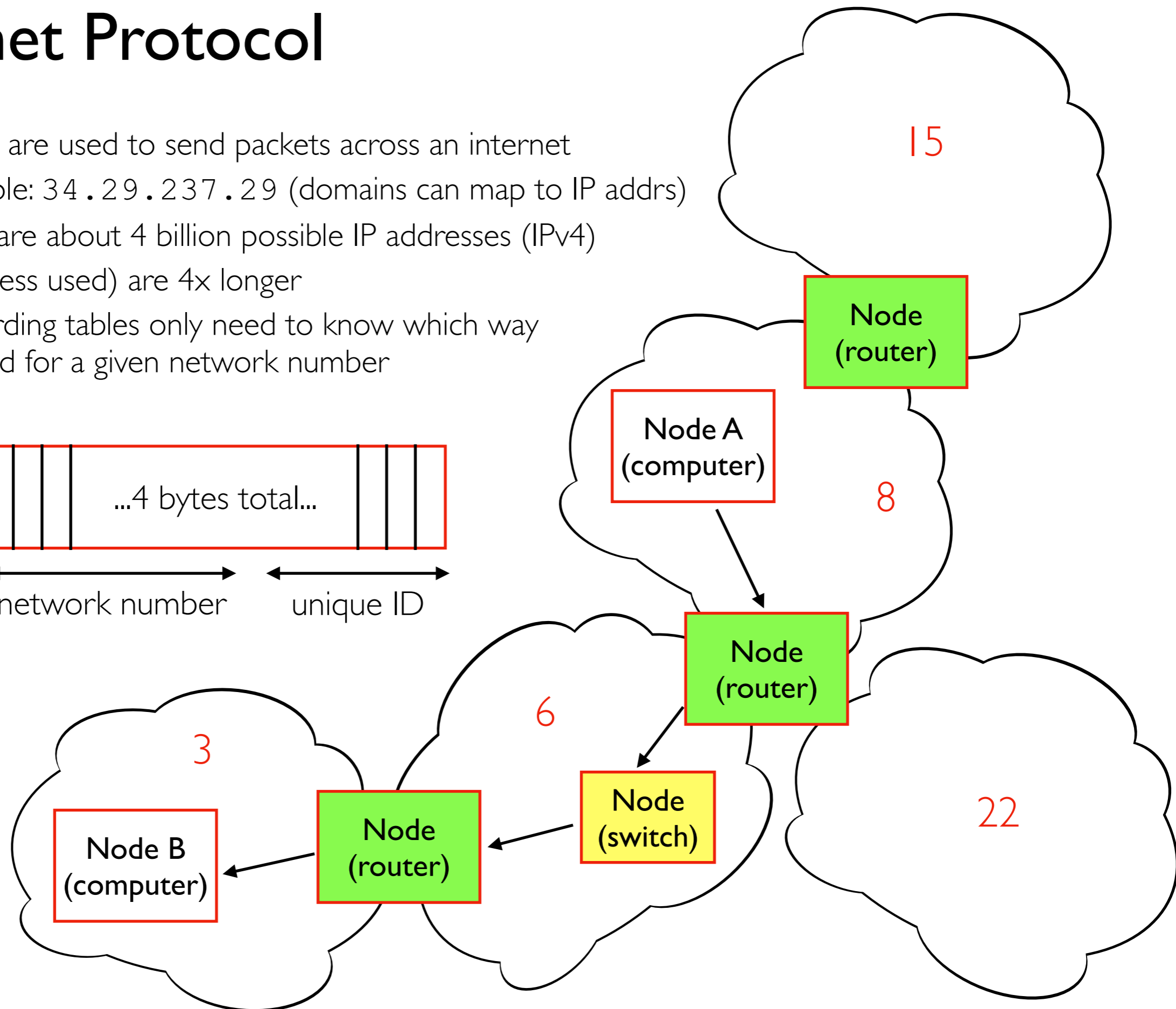
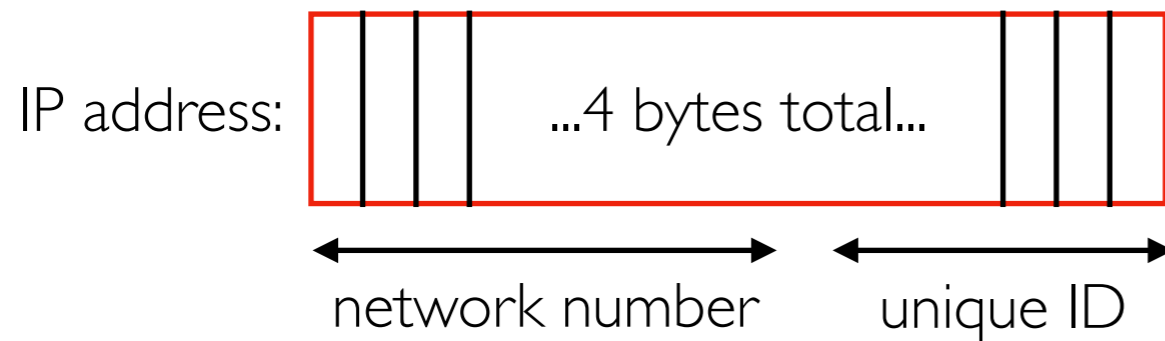
- routers contain **forwarding tables** that help them decide which direction to send along a packet
- those tables would be too big if a router had to know where every MAC address existed in the Internet



Internet Protocol

IP addresses are used to send packets across an internet

- example: 34 . 29 . 237 . 29 (domains can map to IP addrs)
- there are about 4 billion possible IP addresses (IPv4)
- IPv6 (less used) are 4x longer
- forwarding tables only need to know which way to send for a given network number



Listening on an Interface

```
trh@instance-2:~$ ifconfig
ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
    inet 10.0.1.2 netmask 255.255.255.255 broadcast 0.0.0.0
    inet6 fe80::...<link>
    ether 42:00:00:00:00:00
    ...
```

```
python3 -m http.server --bind 10.0.1.2
```

```
ens5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1430
    inet 10.0.3.2 netmask 255.255.255.255 broadcast 10.0.3.2
    inet6 fe80::...<link>
    ether 42:00:00:00:00:00
    ...
```

```
python3 -m http.server --bind 10.0.3.2
```

```
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1
    loop txqu...
    ...
```

```
python3 -m http.server --bind 127.0.0.1
```

```
all of them: python3 -m http.server --bind 0.0.0.0
```

Private Networks

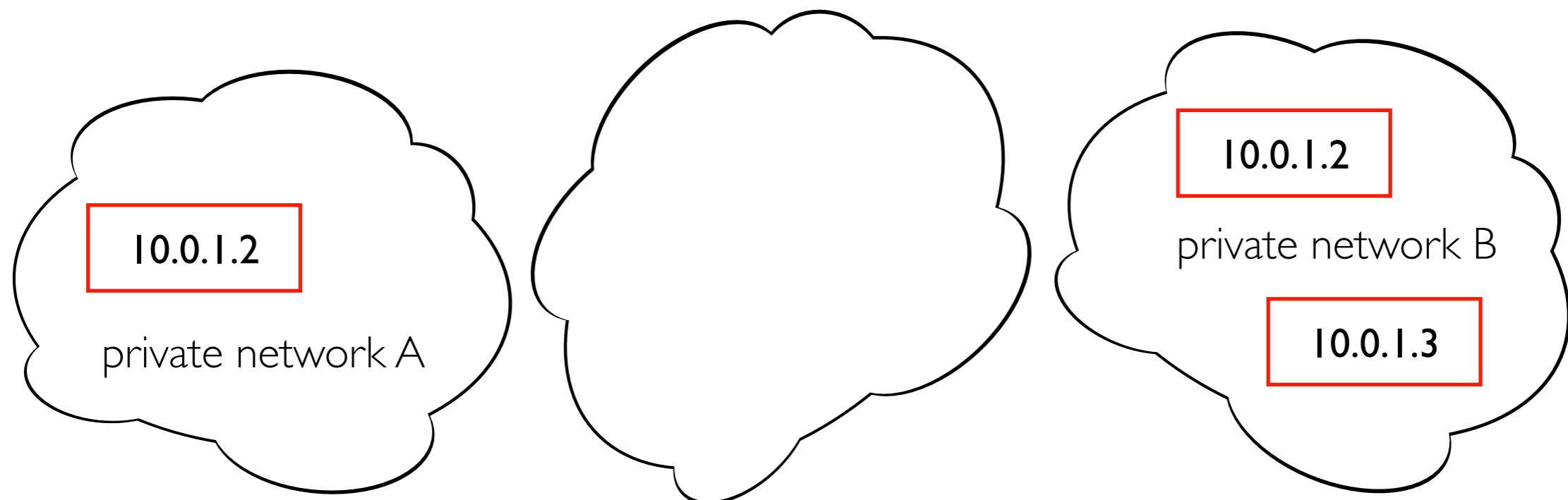
Challenges

- we don't have enough IPv4 addresses
- we don't want every machine to be able to receive packets from anywhere

Private ranges:

- 192.168.0.0 to 192.168.255.255
- 172.16.0.0 to 172.31.255.255
- 10.0.0.0 to 10.255.255.255

Private networks allow duplicates and unreachable machines



Private Networks

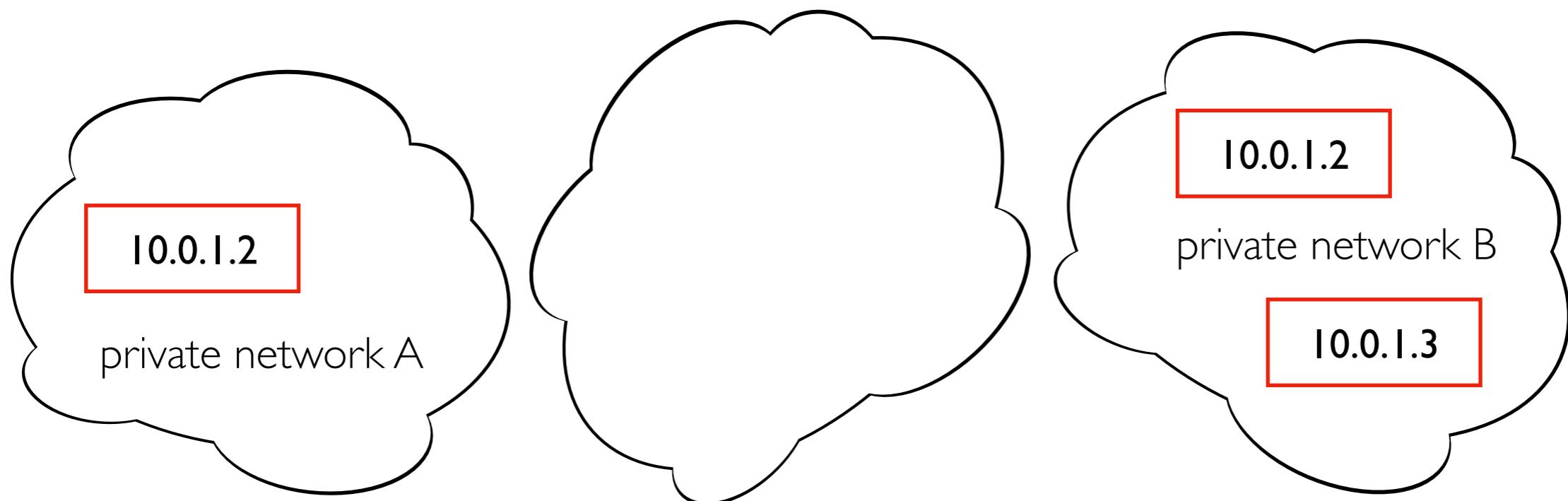
```
trh@instance-2:~$ ifconfig
ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1460
    inet 10.0.1.2 python3 -m http.server --bind 10.0.1.2
    inet6 fe80::40
    ether 42:01:0a:00:01:02 txqueuelen 1000 (Ethernet)
    ...
```

Private ranges:

- 192.168.0.0 to 192.168.255.255
- 172.16.0.0 to 172.31.255.255
- 10.0.0.0 to 10.255.255.255

`http://10.0.1.2:...`
won't work for webbrowsers!

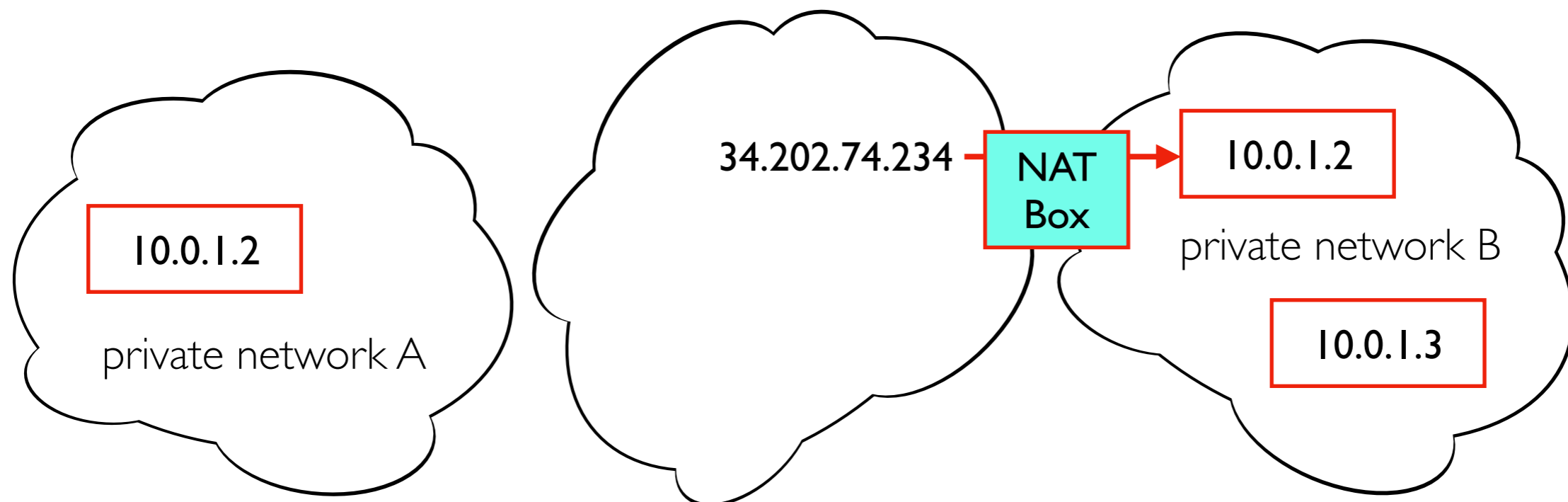
Private networks allow duplicates and unreachable machines



Network Address Translation

Google Console (view NAT config)

<input type="checkbox"/>	Status	Name ↑	Internal IP	External IP
<input type="checkbox"/>	✓	instance-1	10.128.0.36 (nic0)	34.29.237.29 (nic0)
<input type="checkbox"/>	✓	instance-2	10.0.1.2 (nic0) 10.0.3.2 (nic1)	35.202.74.234 (nic0) 34.29.220.248 (nic1)



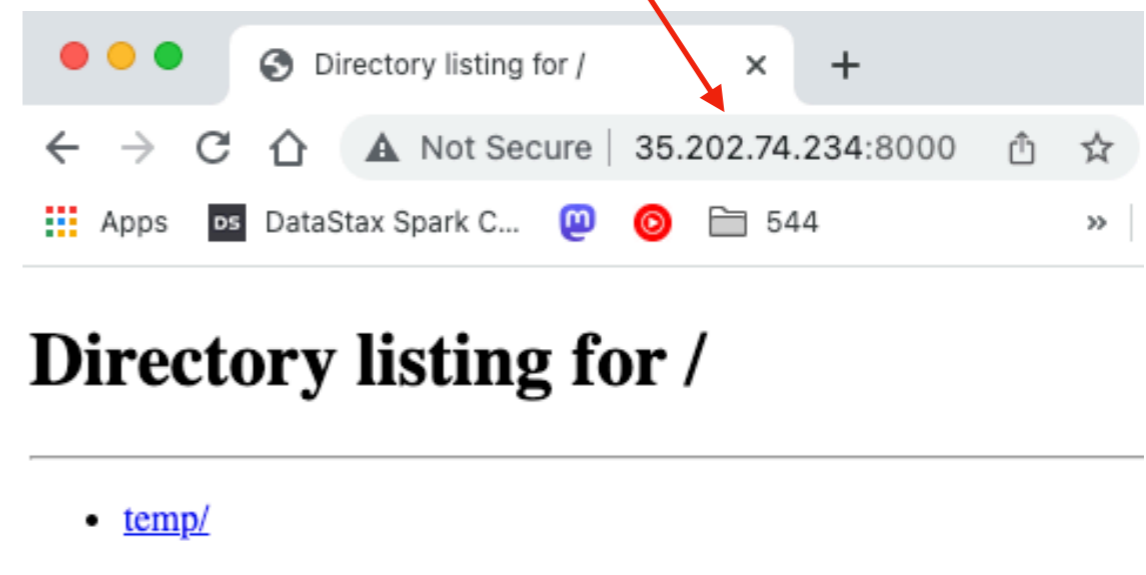
Network Address Translation

Google Console (view NAT config)

<input type="checkbox"/>	Status	Name ↑	Internal IP	External IP
<input type="checkbox"/>	✓	instance-1	10.128.0.36 (nic0)	34.29.237.29 (nic0)
<input type="checkbox"/>	✓	instance-2	10.0.1.2 (nic0) 10.0.3.2 (nic1)	35.202.74.234 (nic0) 34.29.220.248 (nic1)

By default, the external IPs are "ephemeral" (change upon reboot). You can rent "static" IPs that don't change.

Browser



Server

```
trh@instance-2:~/temp$ python3 -m http.server --bind 10.0.1.2
Serving HTTP on 10.0.1.2 port 8000 (http://10.0.1.2:8000/) ...
72.33.0.184 - - [10/Feb/2023 21:12:53] "GET / HTTP/1.1" 200 -
...
```

Outline

Networks

Internets and "The Internet"

Transport Protocols

Application Protocols

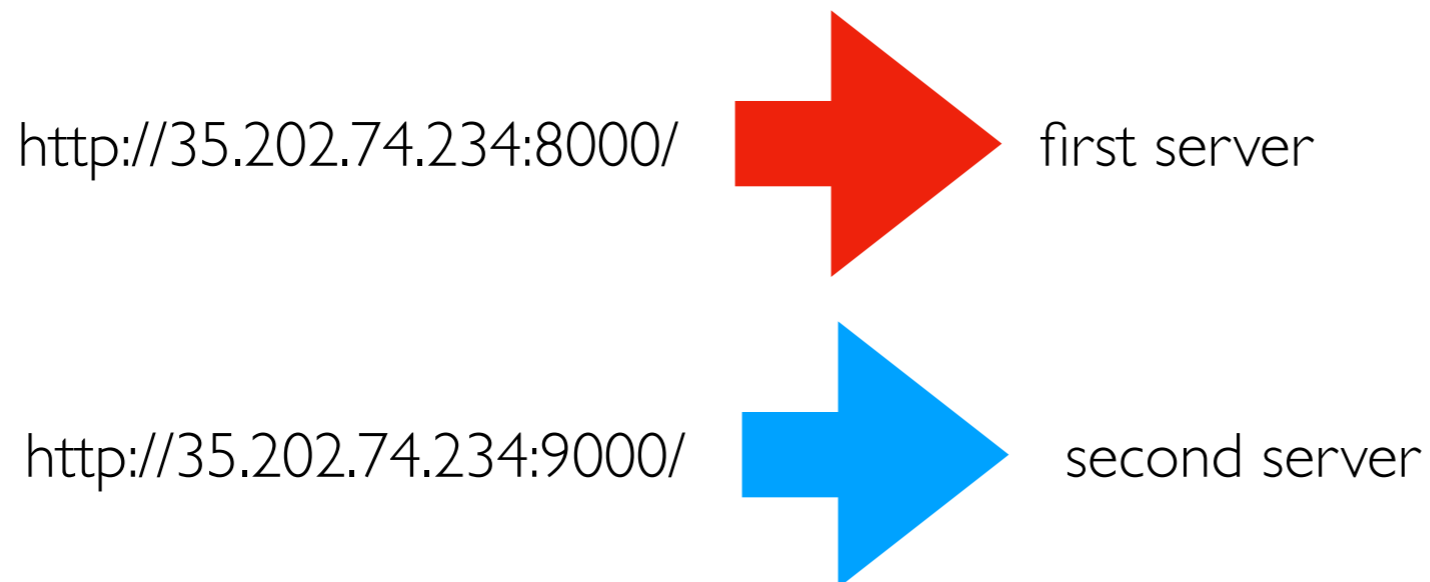
Port Numbers

Computers might be running multiple processes using the network

- IP address => which NIC?
- Port number => which process?

```
trh@instance-2:~$ python3 -m http.server --directory=A --bind 10.0.1.2 8000 &  
[1] 13502  
Serving HTTP on 10.0.1.2 port 8000 (http://10.0.1.2:8000/) ...
```

```
trh@instance-2:~$ python3 -m http.server --directory=B --bind 10.0.1.2 9000 &  
[2] 13503  
Serving HTTP on 10.0.1.2 port 9000 (http://10.0.1.2:9000/) ...
```



TopHat

42:01:0a:80:00:25 is an example of what?

Transport Protocols

Most common

- **UDP** (User Datagram Protocol)
- **TCP** (Transmission Control Protocol)

BOTH build on IP networking and BOTH provide **port numbers**

```
trh@instance-2:~/temp$ sudo lsof -i tcp -P
COMMAND      PID NODE NAME
...
sshd          863 TCP *:22 (LISTEN)
sshd          863 TCP *:22 (LISTEN)
...
python3      13607 TCP instance-2...internal:8000 (LISTEN)
python3      13608 TCP instance-2...internal:9000 (LISTEN)
```

Reliability: UDP vs. TCP

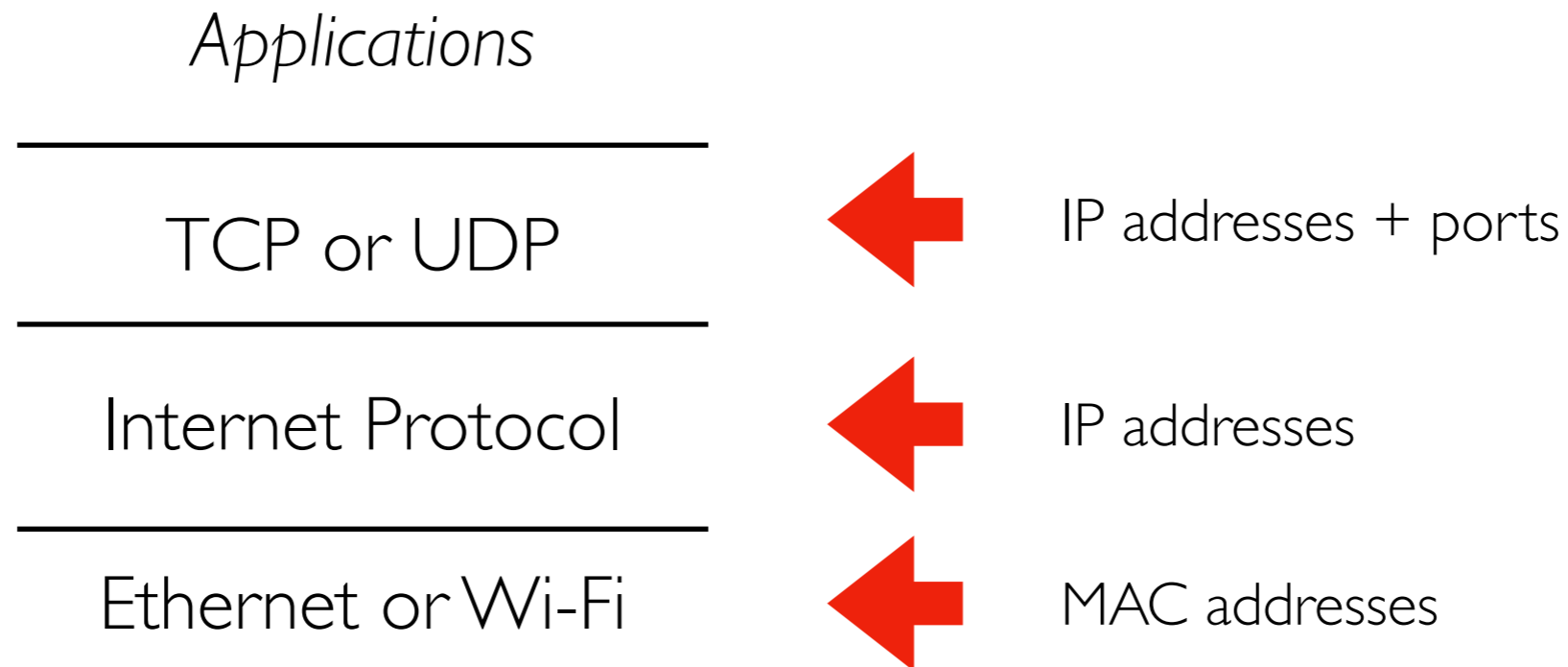
Packets may be

- dropped
- reordered
- split

TCP saves+reassembles packets in order to provide original message (when possible).
For packet drops, it retries. We'll mostly use TCP.

UDP doesn't do this extra work. Why ever use UDP?

Network Stack: Common Implementations



Network applications (like most complex systems) are not built as one single system. Layers are built upon other layers to provide additional functionality.

Outline

Networks

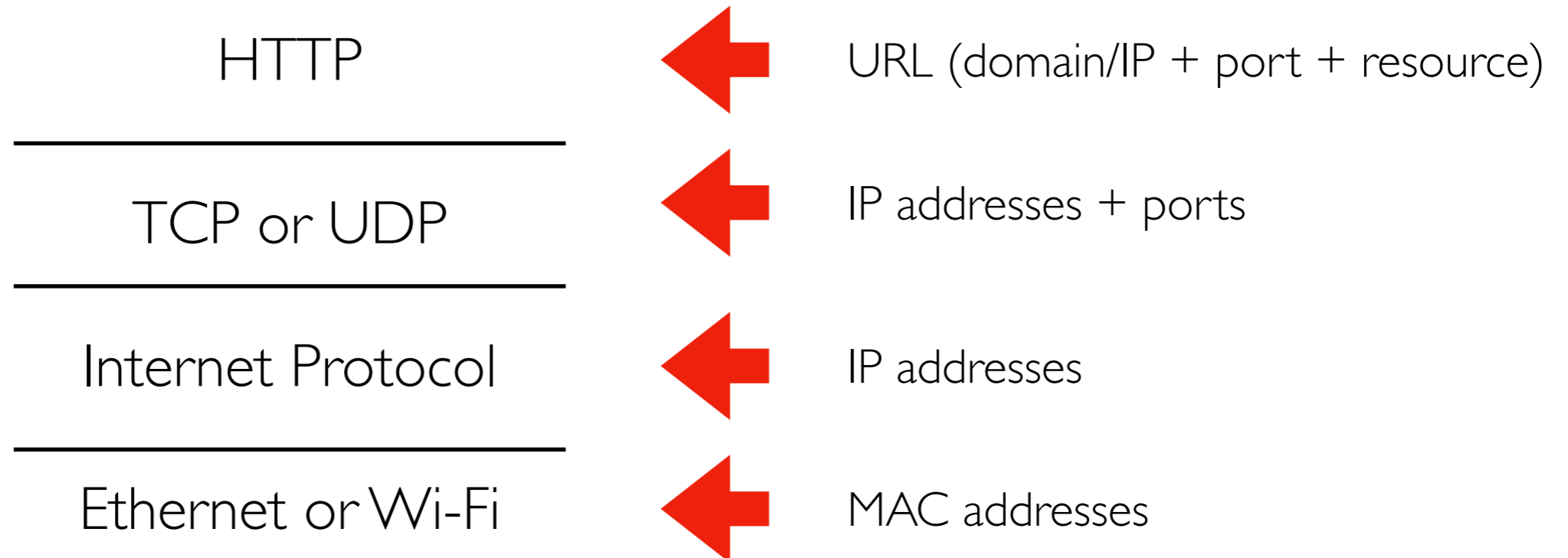
Internets and "The Internet"

Transport Protocols

Application Protocols

- **HTTP**
- gRPC

HTTP (Hypertext Transfer Protocol)



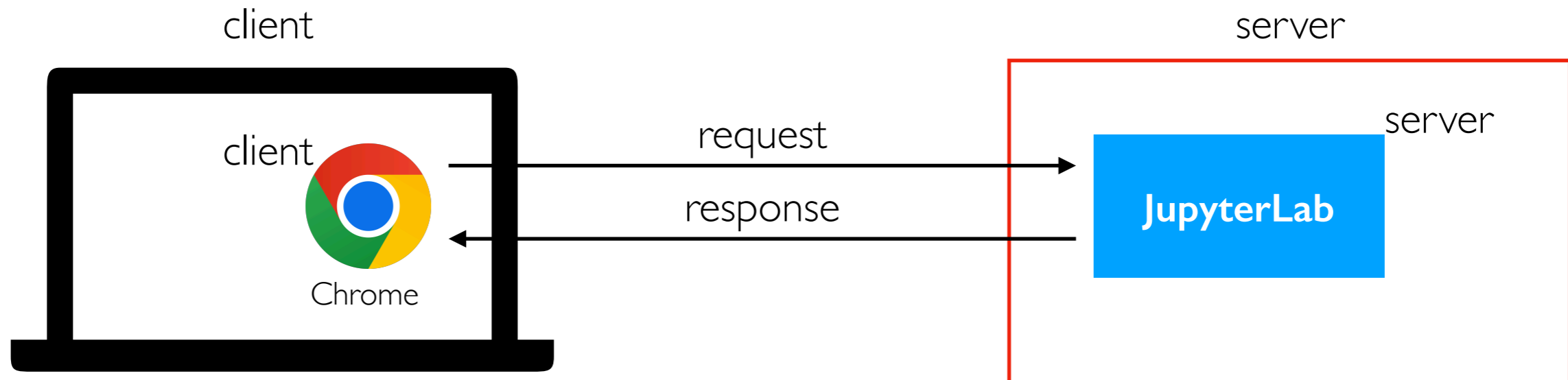
`https://tyler.caraza-harter.com:443/cs544/s23/schedule.html`

domain name
(mapped to an IP)

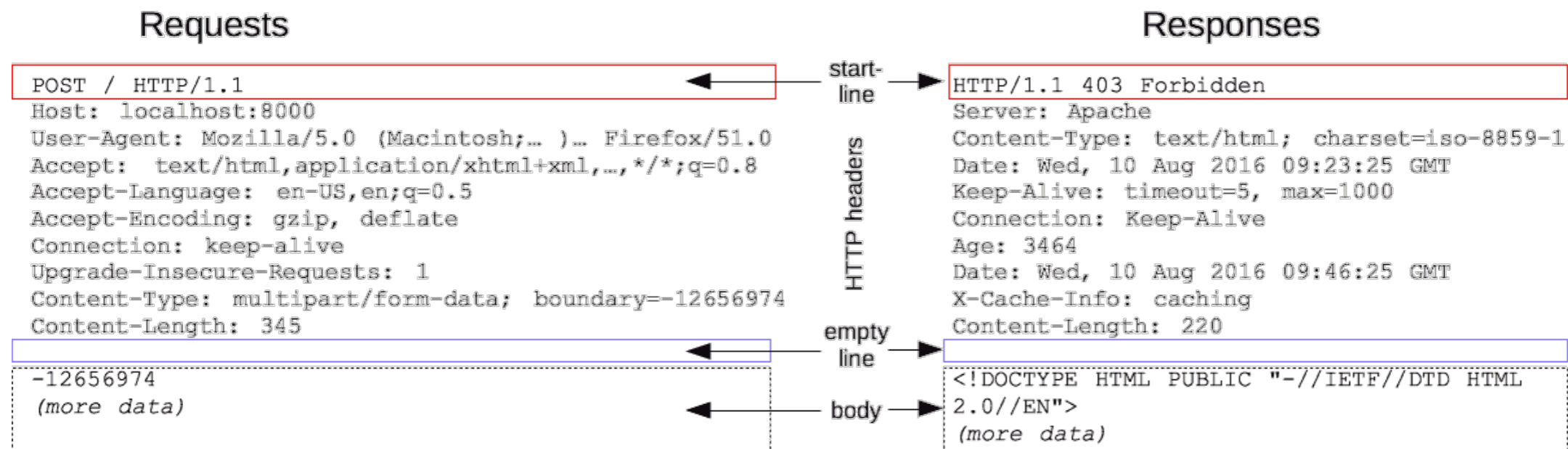
port
(443 is default
for https)

resource

HTTP Messages Between Clients and Servers



Parts: method, resource, status code, headers, body



<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

HTTP Methods (types of messages)

Types of request

- **POST**: create a new resource (request+response have body)
- **PUT**: update a resource (request+response have body, usually)
- **GET**: fetch a resource (response has body)
- **DELETE**: delete a resource ()
- others...

Canvas API example:

GET <https://canvas.wisc.edu/api/v1/conversations>
(see all Canvas conversations in JSON format)

POST <https://canvas.wisc.edu/api/v1/conversations>
(create new Canvas conversation)

<https://canvas.instructure.com/doc/api/conversations.html>

Outline

Networks

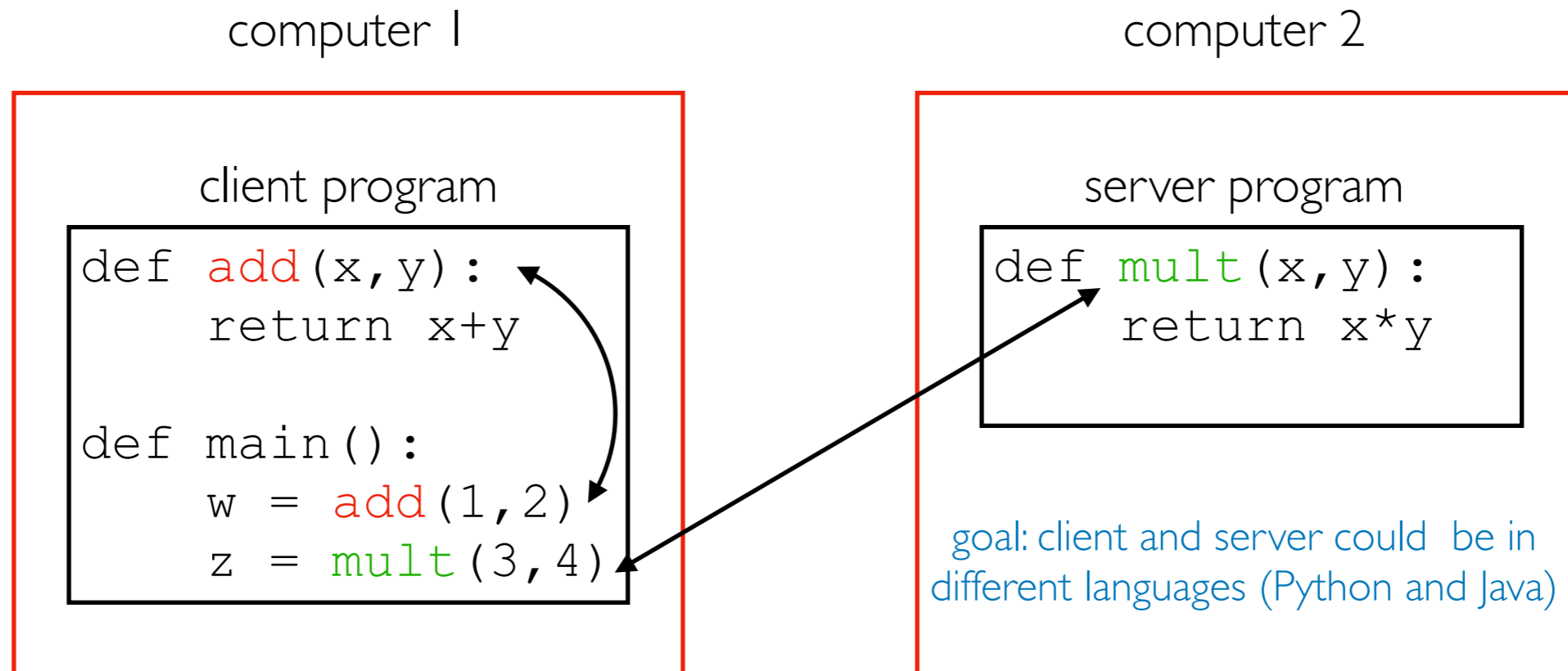
Internets and "The Internet"

Transport Protocols

Application Protocols

- HTTP
- **gRPC**

Remote Procedure Calls (RPCs)



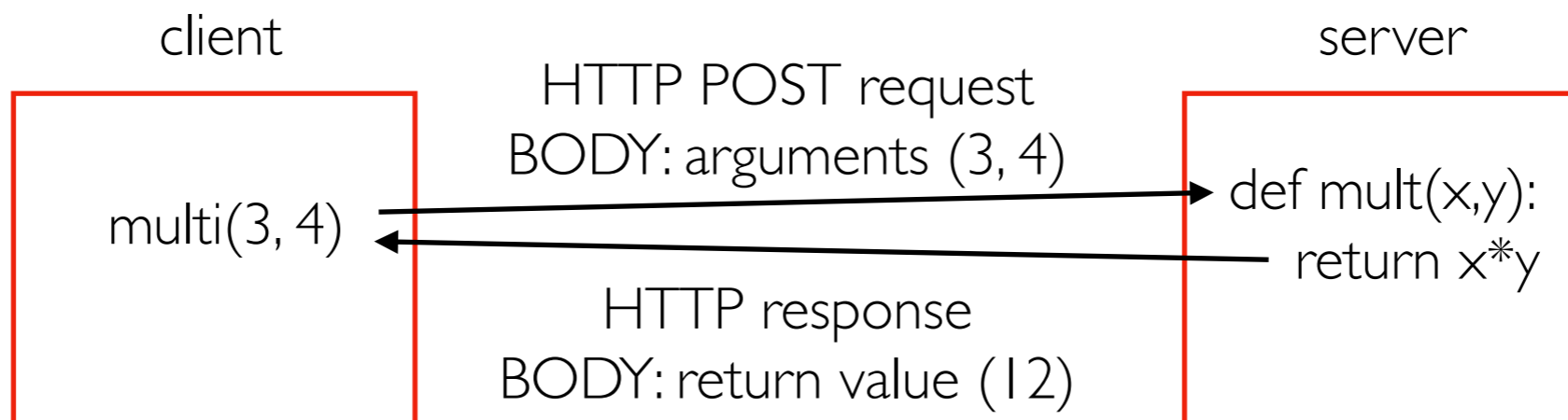
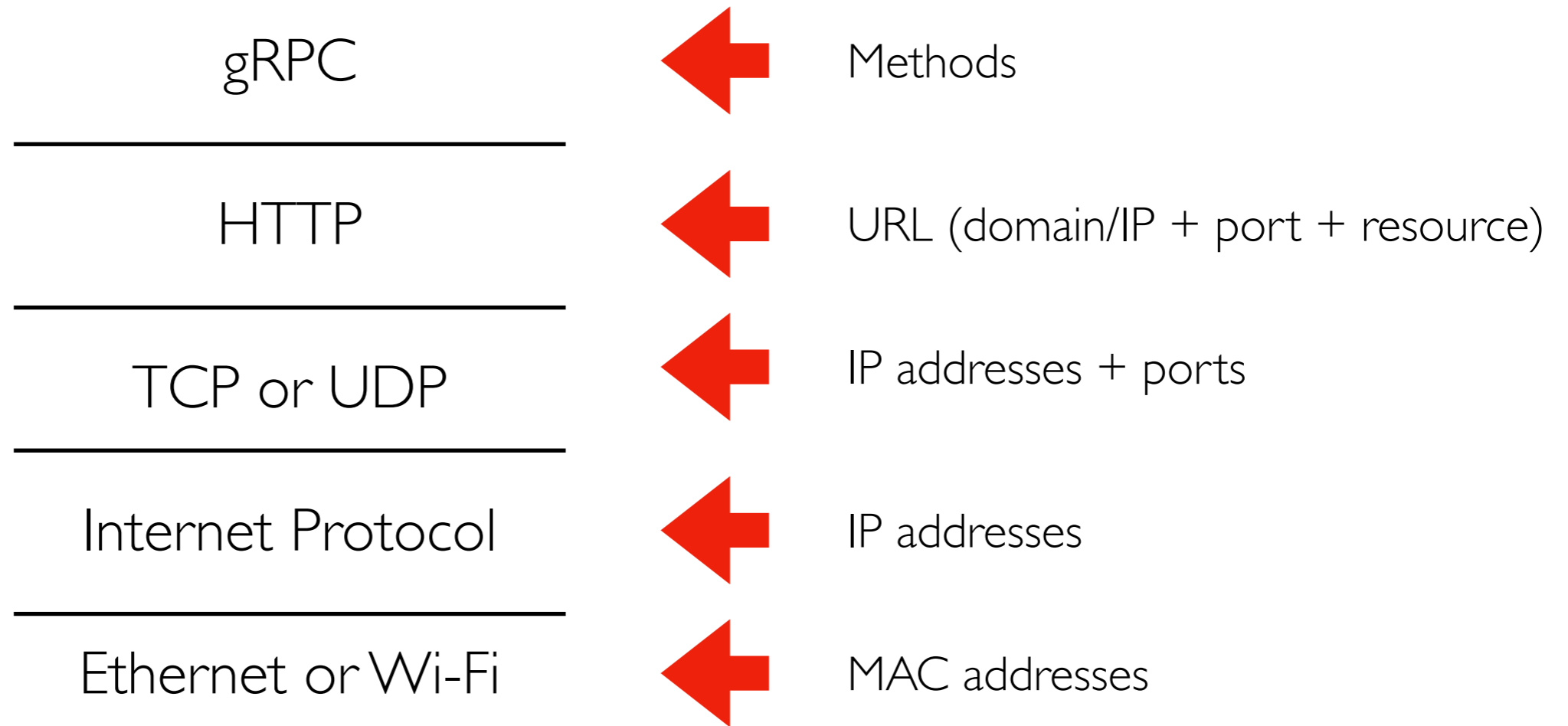
procedure = function

- **main** calling **add** is a regular procedure call
- **main** call **mult** is a remote procedure call

There are MANY tools to do RPCs

- Thrift (developed at Facebook)
- gRPC (developed at Google) -- this semester

gRPC builds on HTTP



Serialization/deserialization (Protobufs)

How do we represent arguments and return values as bytes in a request/response body?

Serialization: various types (ints, strs, lists, etc) to **bytes** ("wire format")

Deserialization: **bytes** to various types

Challenge 1: every language has different types and we want cross-languages calls

gRPC uses Google's **Protocol Buffers** provide a uniform type system.

Challenge 2: different hardware orders bytes differently

cpu A int32:

byte 1	byte 2	byte 3	byte 4
--------	--------	--------	--------

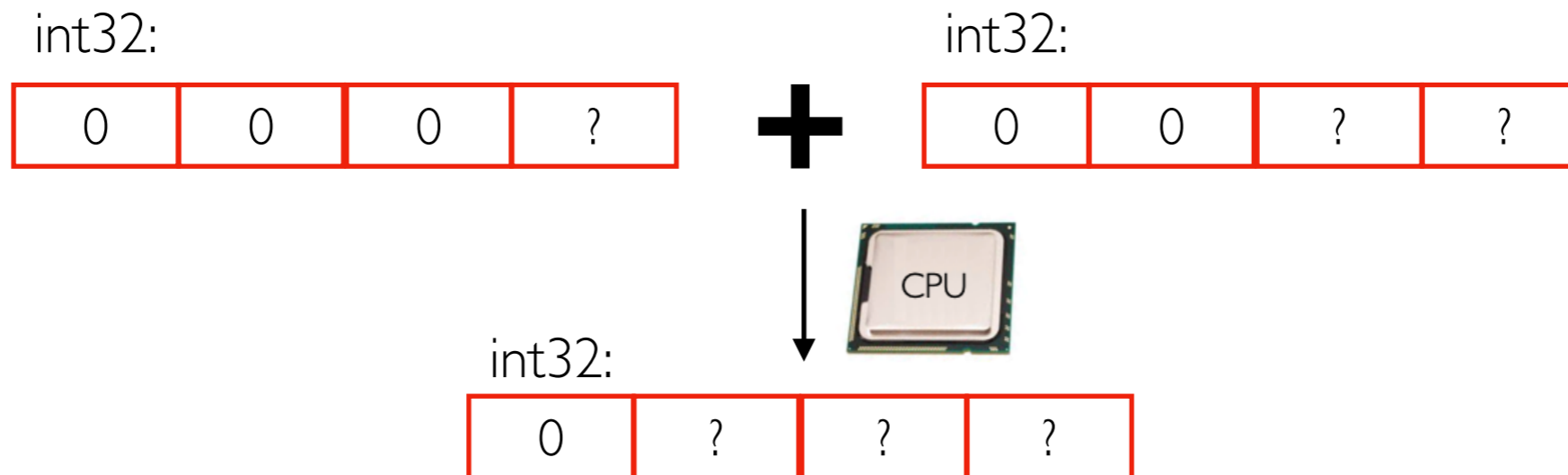
cpu B int32:

byte 4	byte 3	byte 2	byte 1
--------	--------	--------	--------

.proto	C++	Java	Python
double	double	double	float
float	float	float	float
int32	int32	int	int
int64	int64	long	int
uint32	uint32	int	int
uint64	uint64	long	int
sint32	int32	int	int
sint64	int64	long	int
bool	bool	boolean	bool
string	string	String	str
bytes	string	ByteStri	bytes

<https://protobuf.dev/programming-guides/proto/>

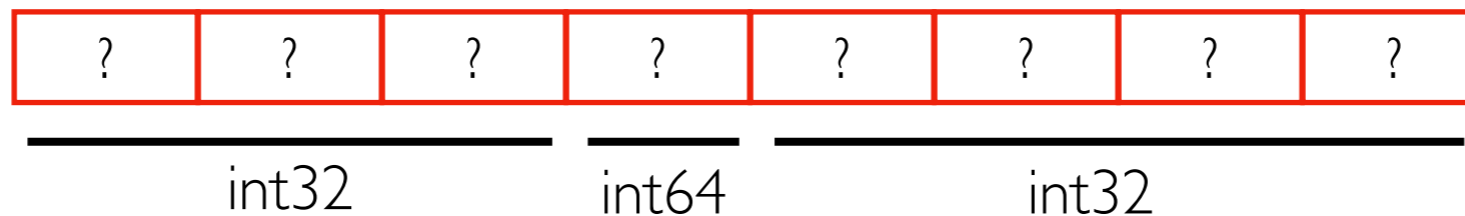
Variable-Length Encoding



For computational efficiency, `int32`'s use 4 bytes during computation.

For space efficiency, smaller numbers in `int32`s use fewer bytes (4 bytes is max). This reduces network traffic.

Example nums in a protobuf:



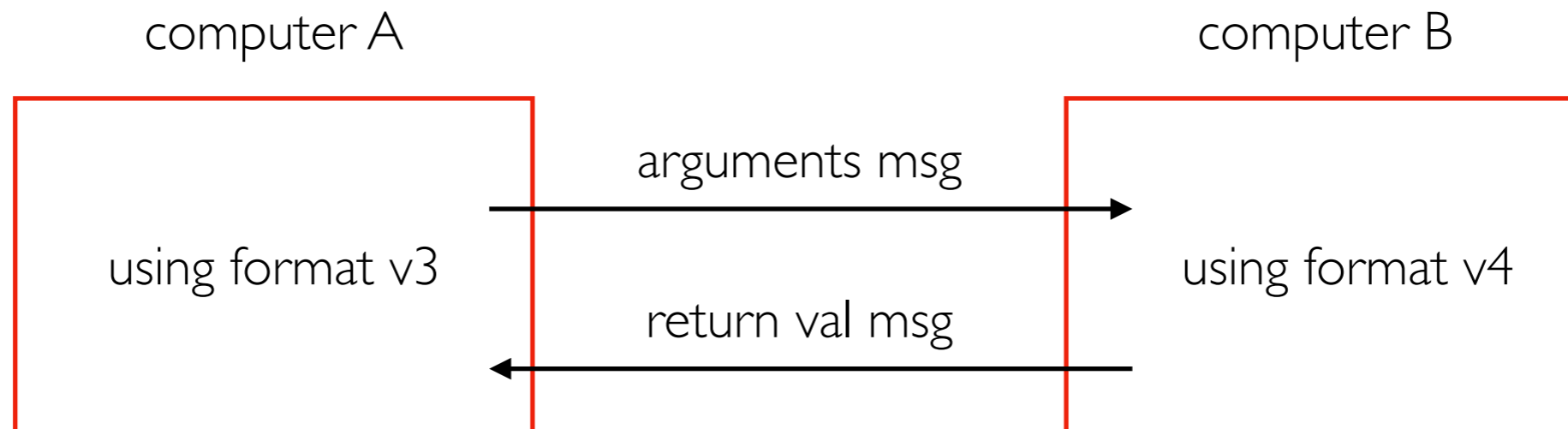
Demos...

Forwards+Backwards Compatibility

Problem: message formats change over time, and it's not possible to magically update all software at the same instant. Clients+servers using different versions may try to communicate.

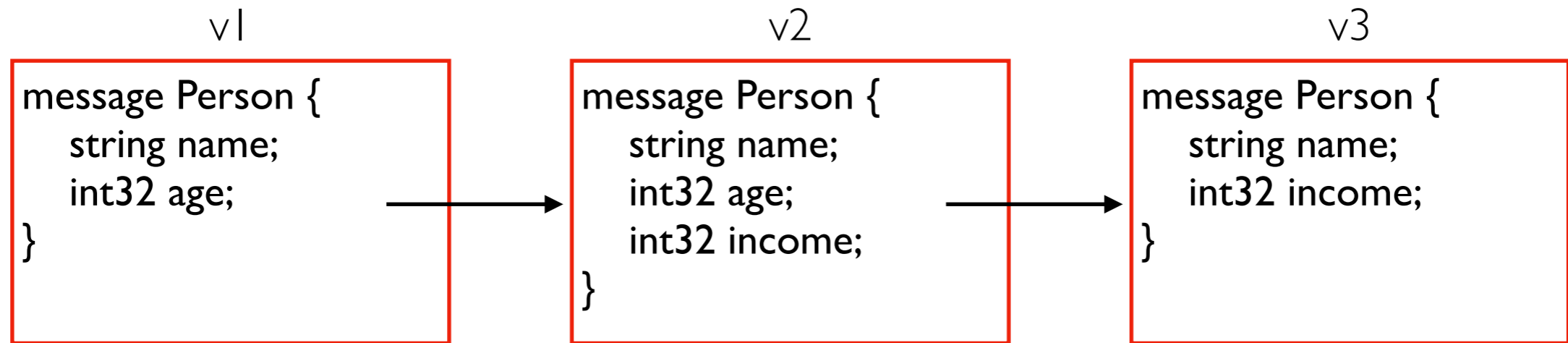
Compatibility

- **backward**: a newer program can read messages in an older format
- **forward**: an older program can read messages in a newer format



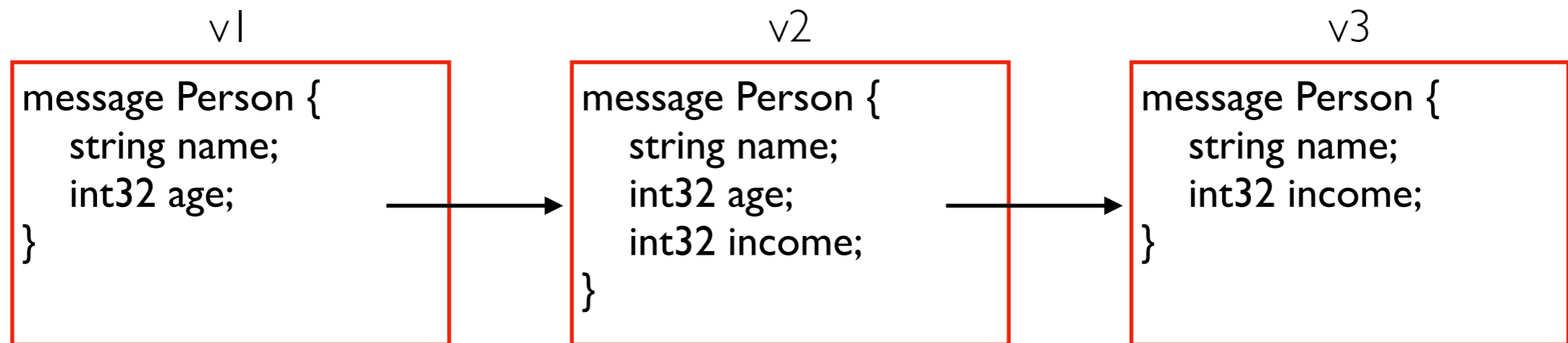
B needs to be backwards compatible to use the arguments and A needs to be forward compatible to use the return value.

Schema Evolution



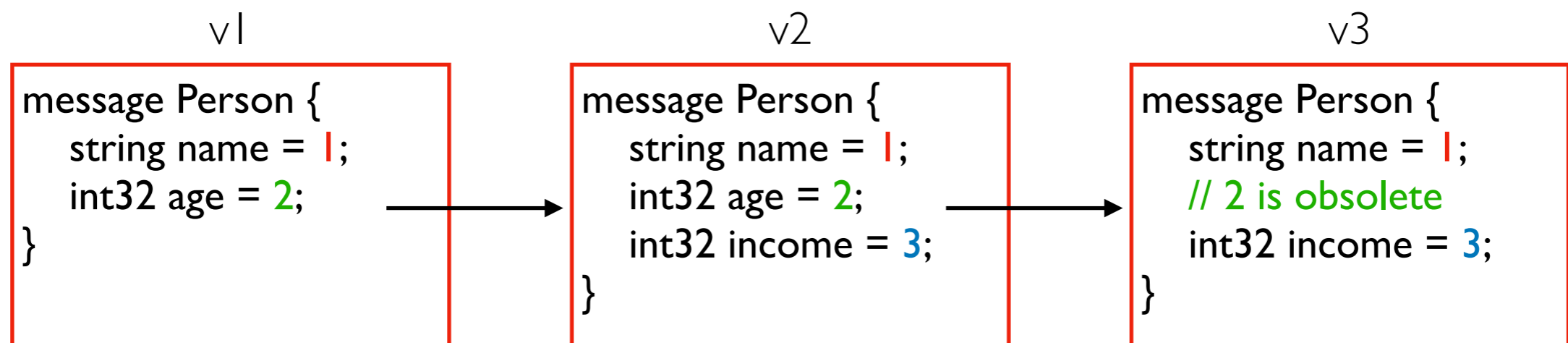
Problem: if a v3 server is sent a message from a v1 client, how will the server interpret the age number? Column/field names not sent in each message (for efficiency).

Schema Evolution

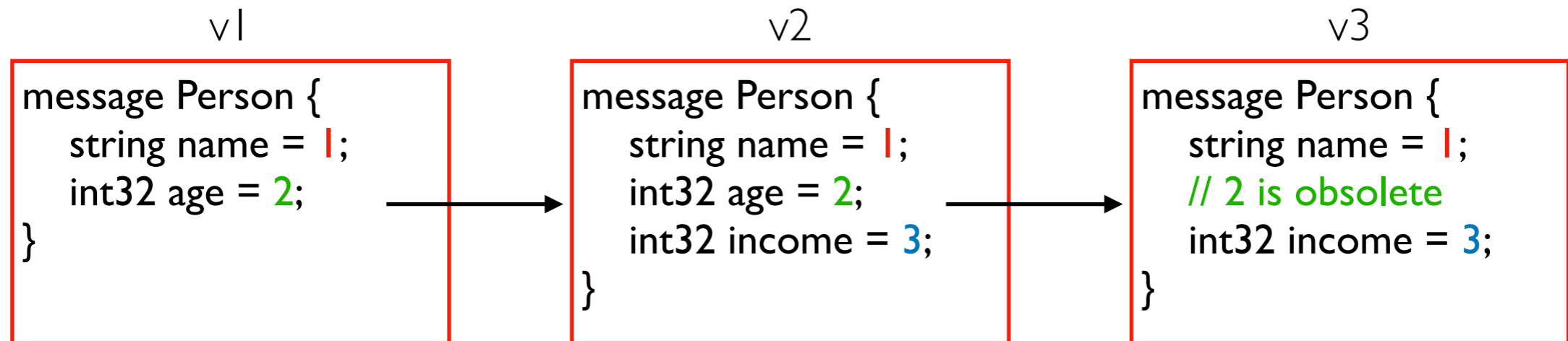


Problem: if a v3 server is sent a message from a v1 client, how will the server interpret the age number? Column/field names not sent in each message (for efficiency).

Solution: number fields and never reuse numbers:



Dropped/default values



v1 client is sending to v3 server.

- **drop** age (server doesn't want it)
- use **default** income (client didn't send it)

Defaults (from <https://protobuf.dev/programming-guides/proto3/#default>)

- string: ""
- bools: False
- numeric: 0

Conclusion

