Computer Organization and Networks

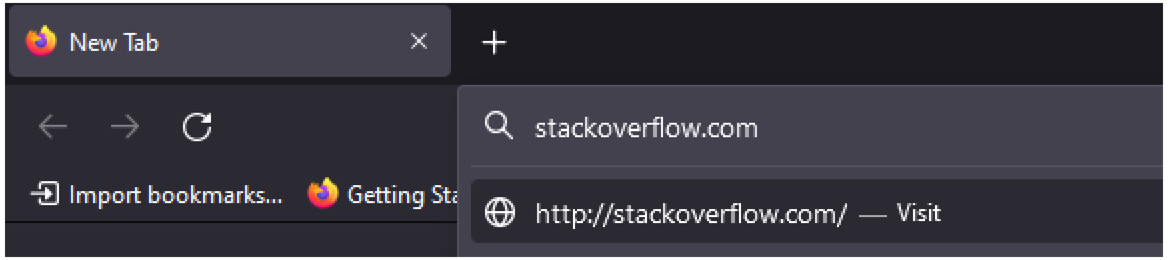
Chapter 6: Networking I

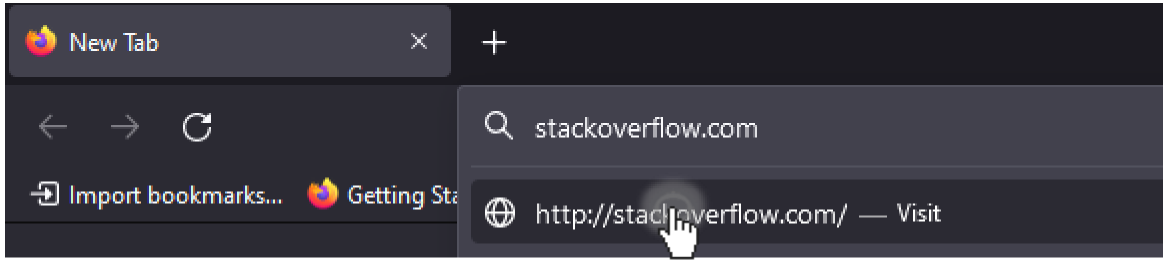
Winter 2023/2024

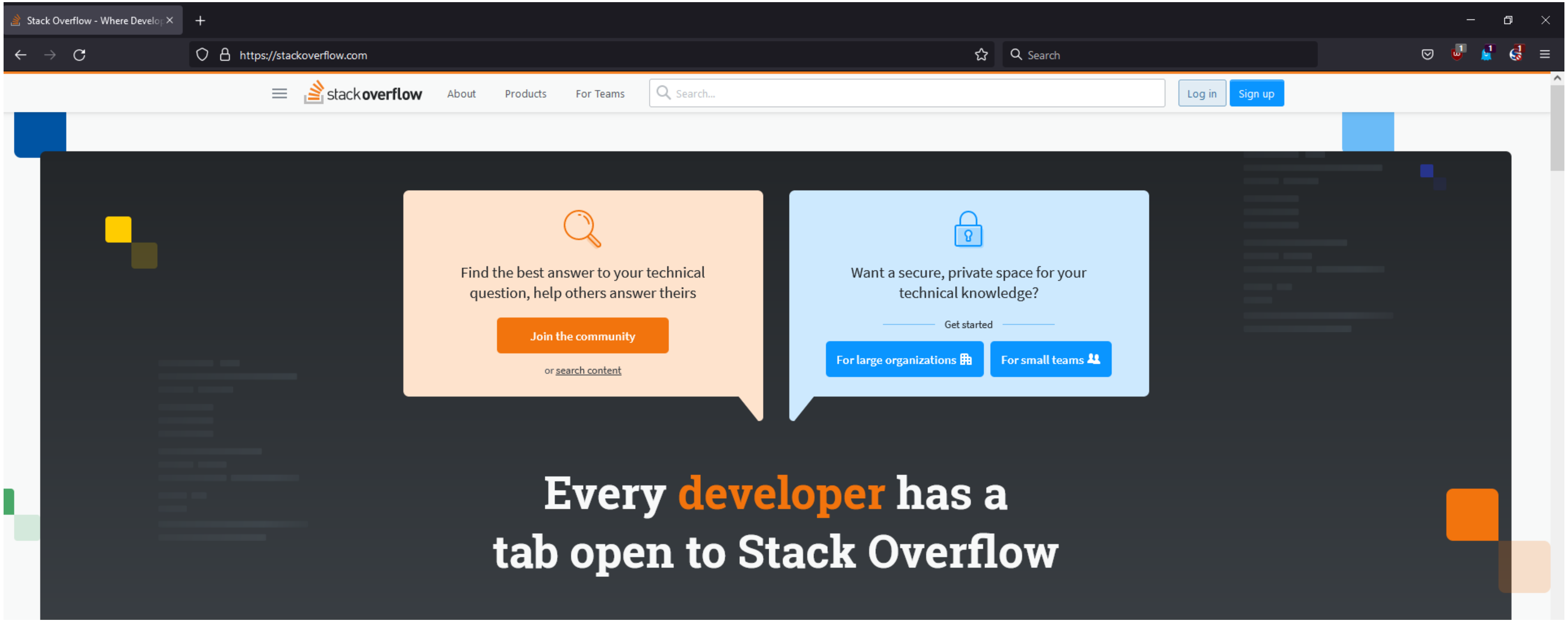
Jakob Heher, www.iaik.tugraz.at
he/his

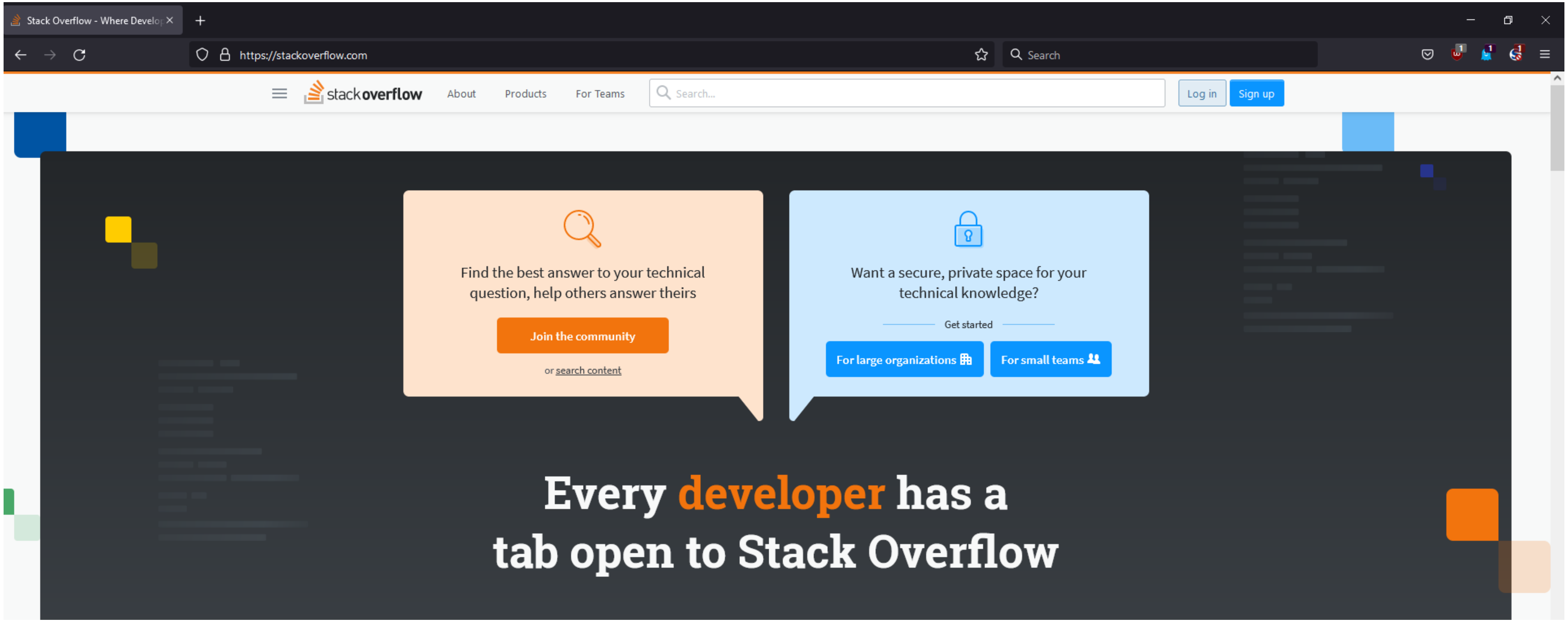
Motivation

- You’ve built a CPU
  - Now let’s make it talk to others

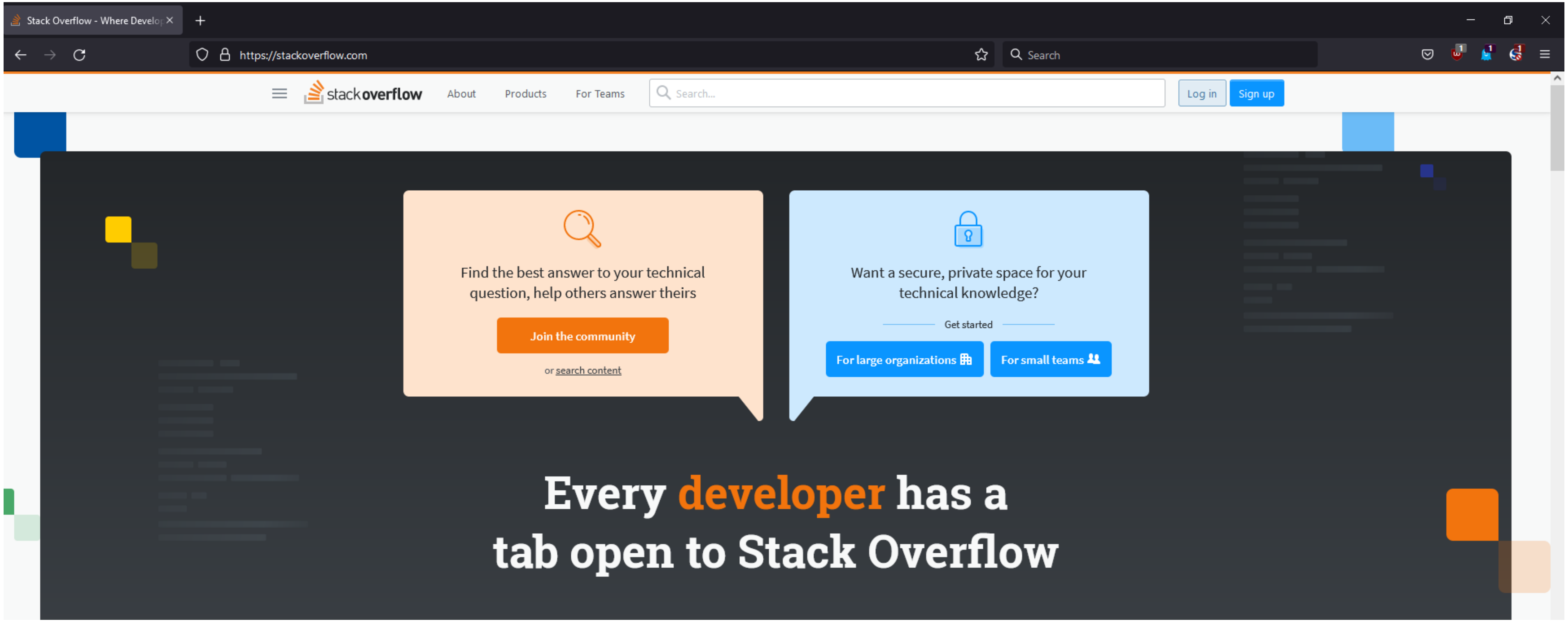

New Tab
stackoverflow.com
Import bookmarks...
Getting Sta
http://stackoverflow.com/ — Visit


New Tab
stackoverflow.com
Import bookmarks...
http://stackoverflow.com/ — Visit

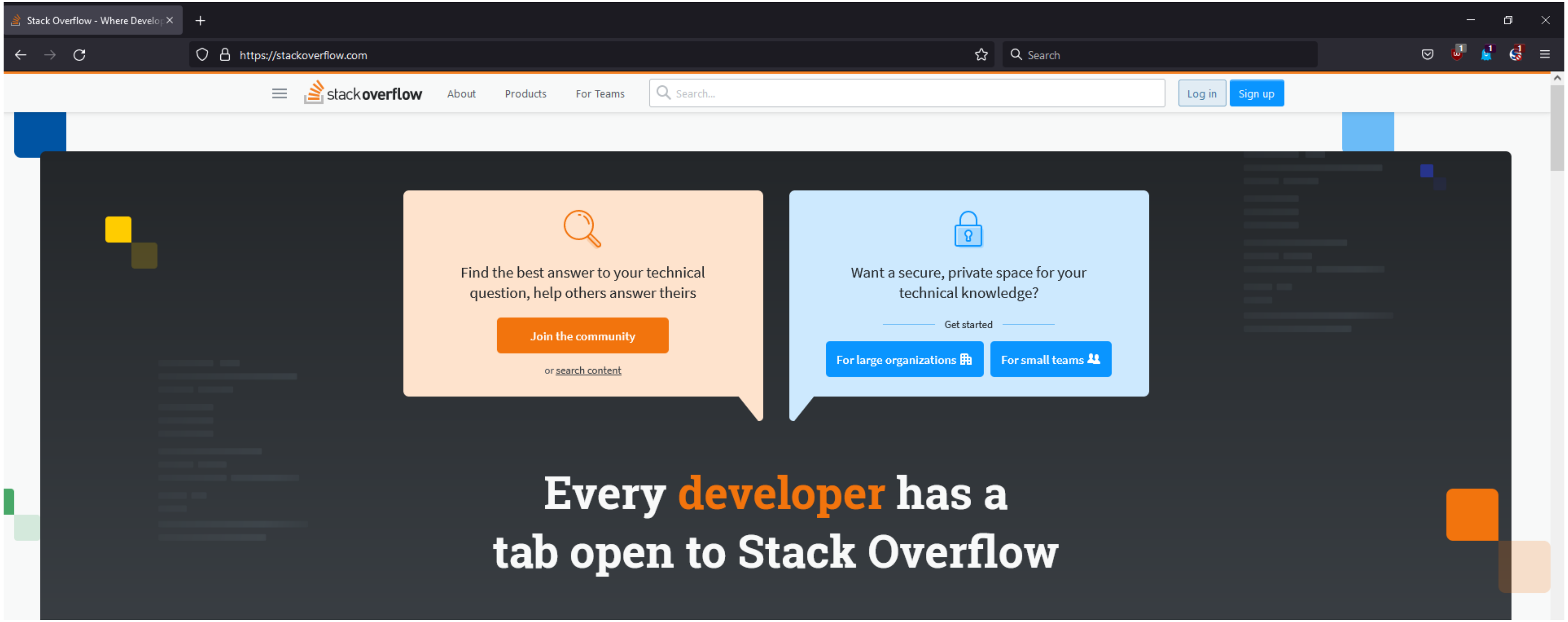

Stack Overflow - Where Develop
https://stackoverflow.com
Search
stackoverflow
About
Products
For Teams
Search...
Log in
Sign up
Find the best answer to your technical question, help others answer theirs
Join the community
or search content
Want a secure, private space for your technical knowledge?
Get started
For large organizations
For small teams
Every developer has a tab open to Stack Overflow



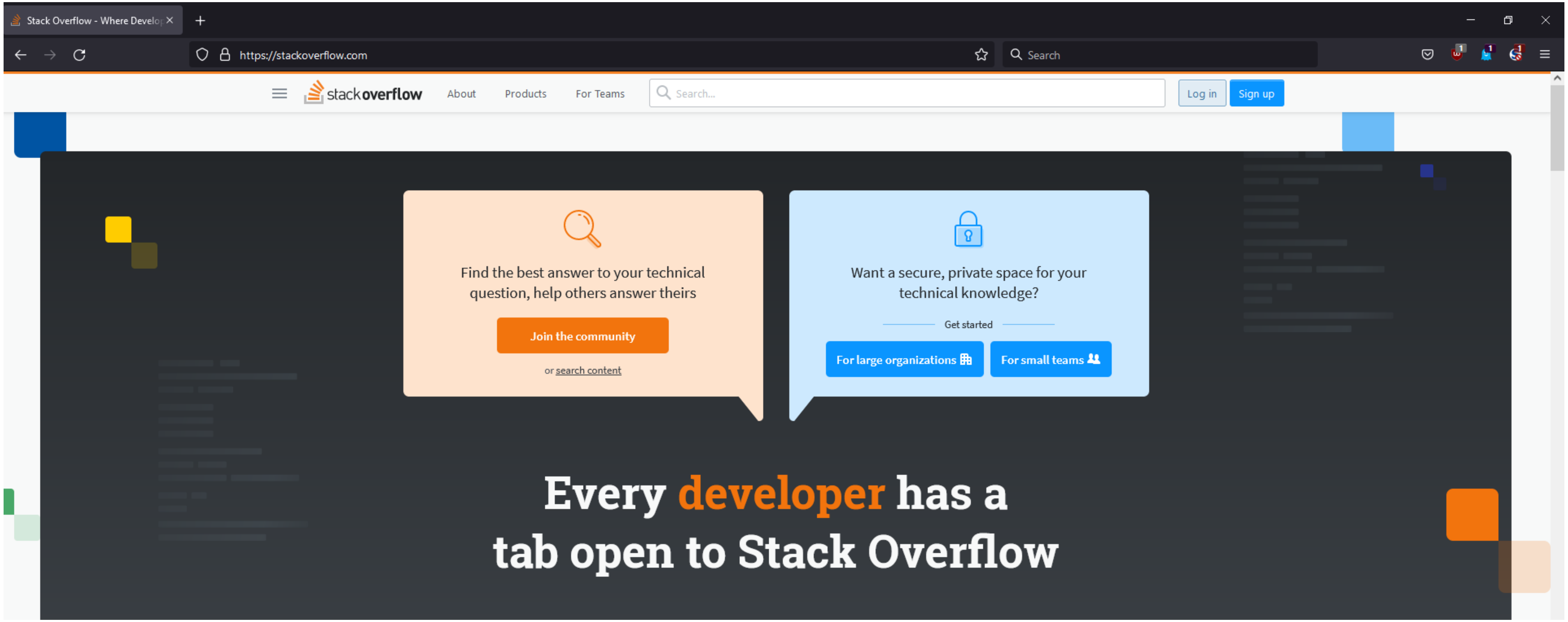
- OK, what did we just do?



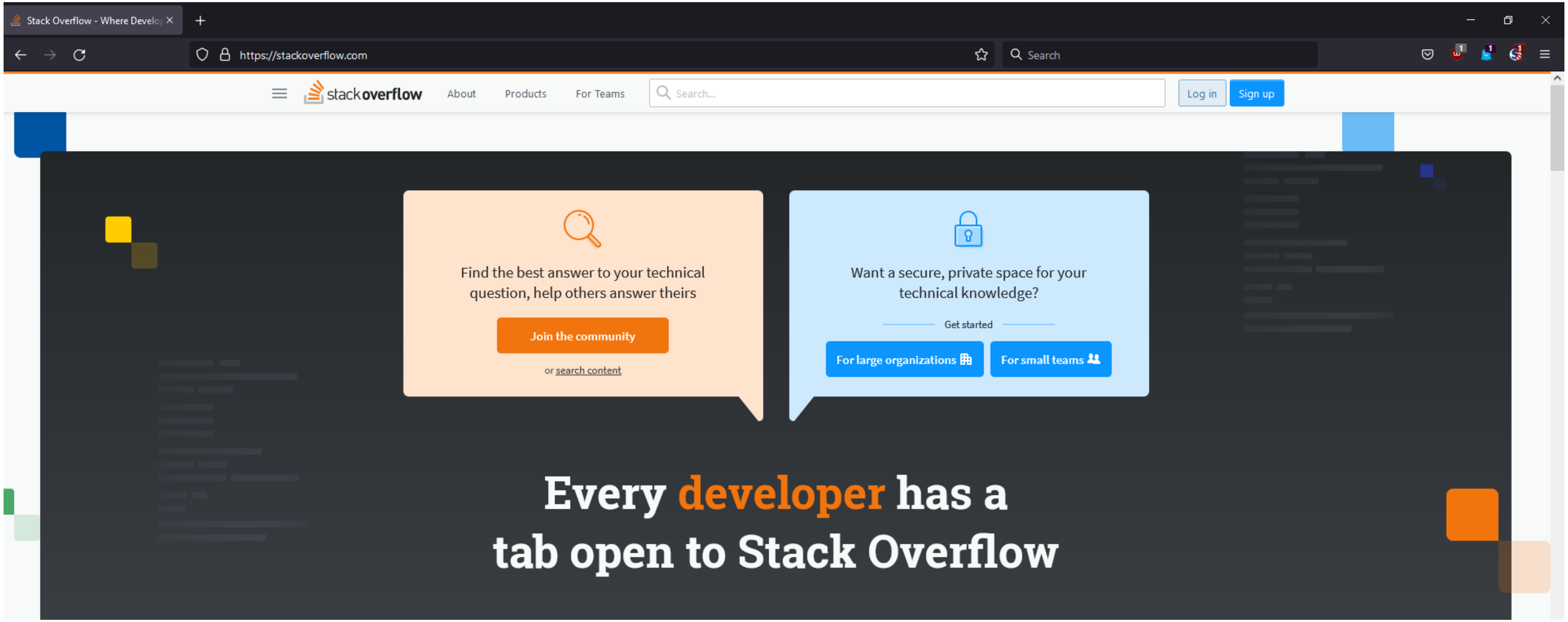
- OK, what did our browser just do?



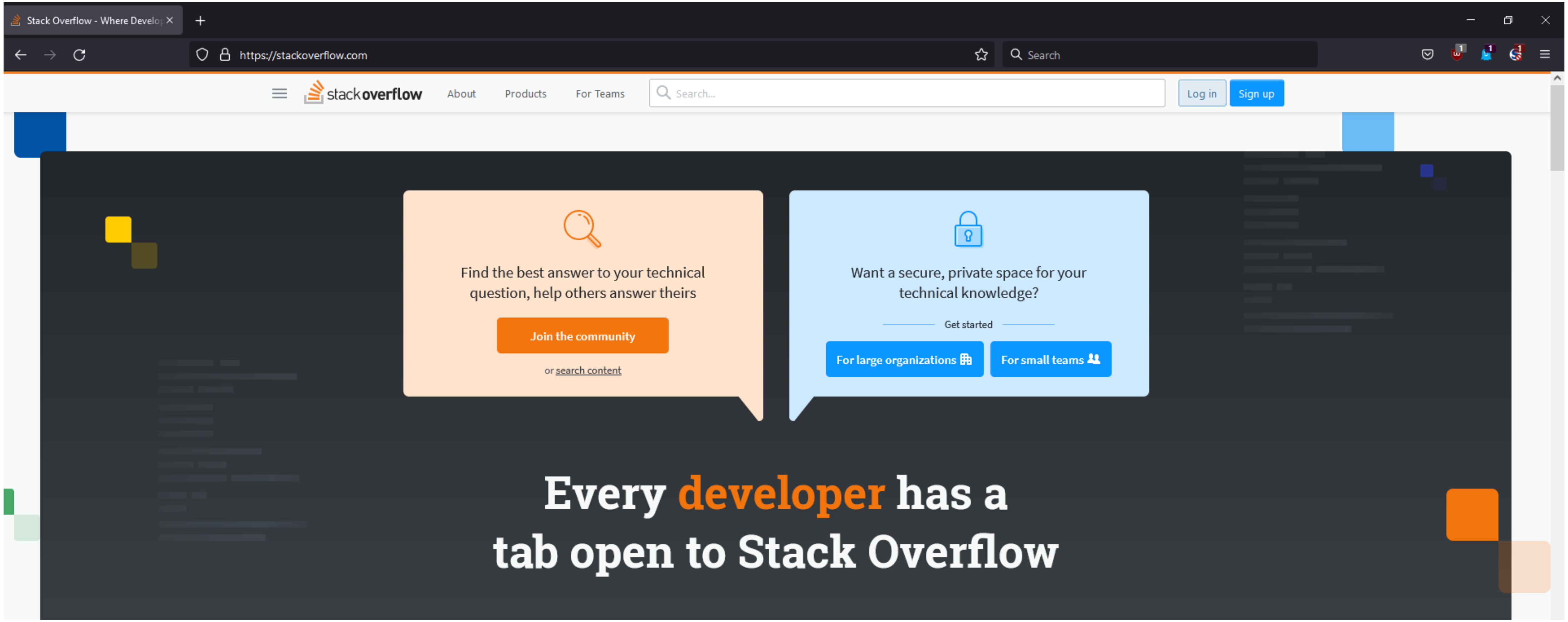
- OK, what did our OS just do?



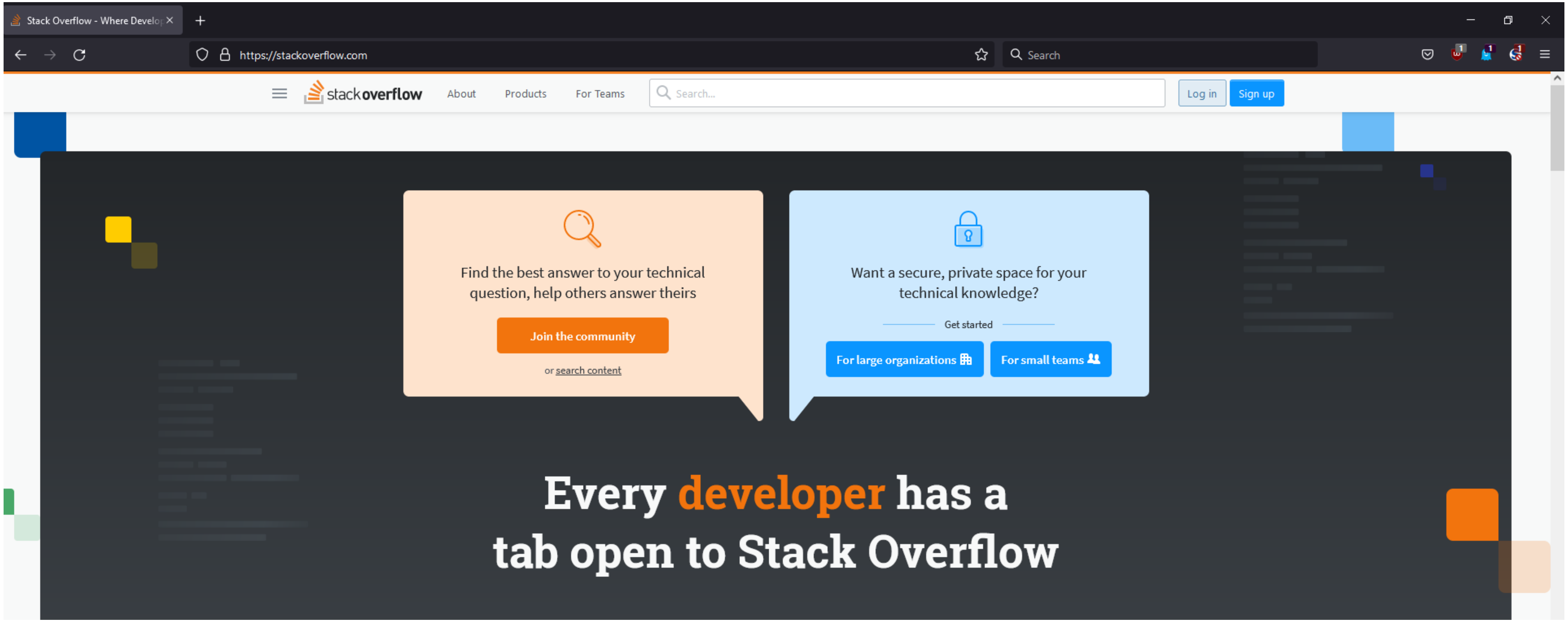
- OK, what did our network card just do?



- OK, what did our router just do?



- OK, what did the StackOverflow server just do?



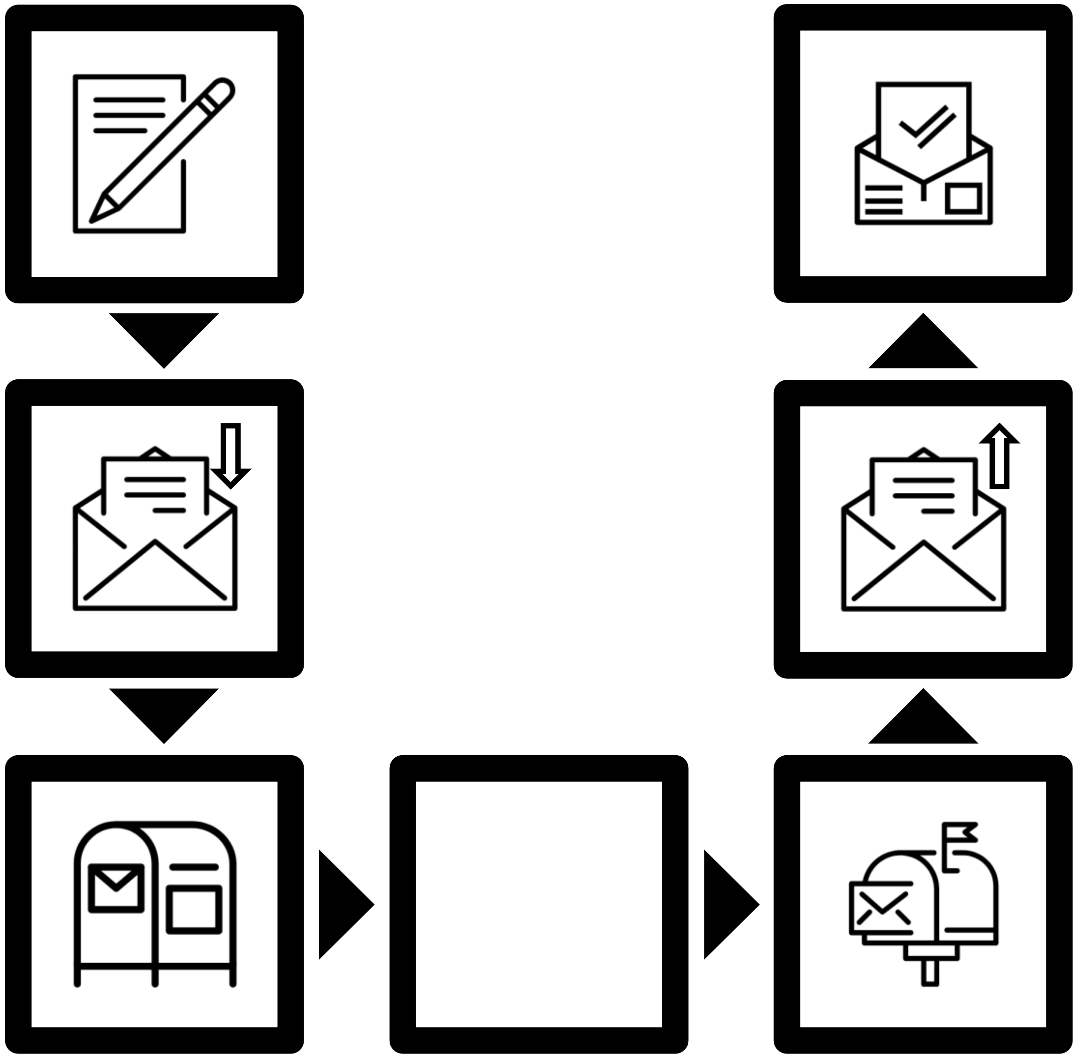
- By the time we’re done here, you’ll know!

Layers & Abstraction

- How do you send a postcard?

Layers & Abstraction

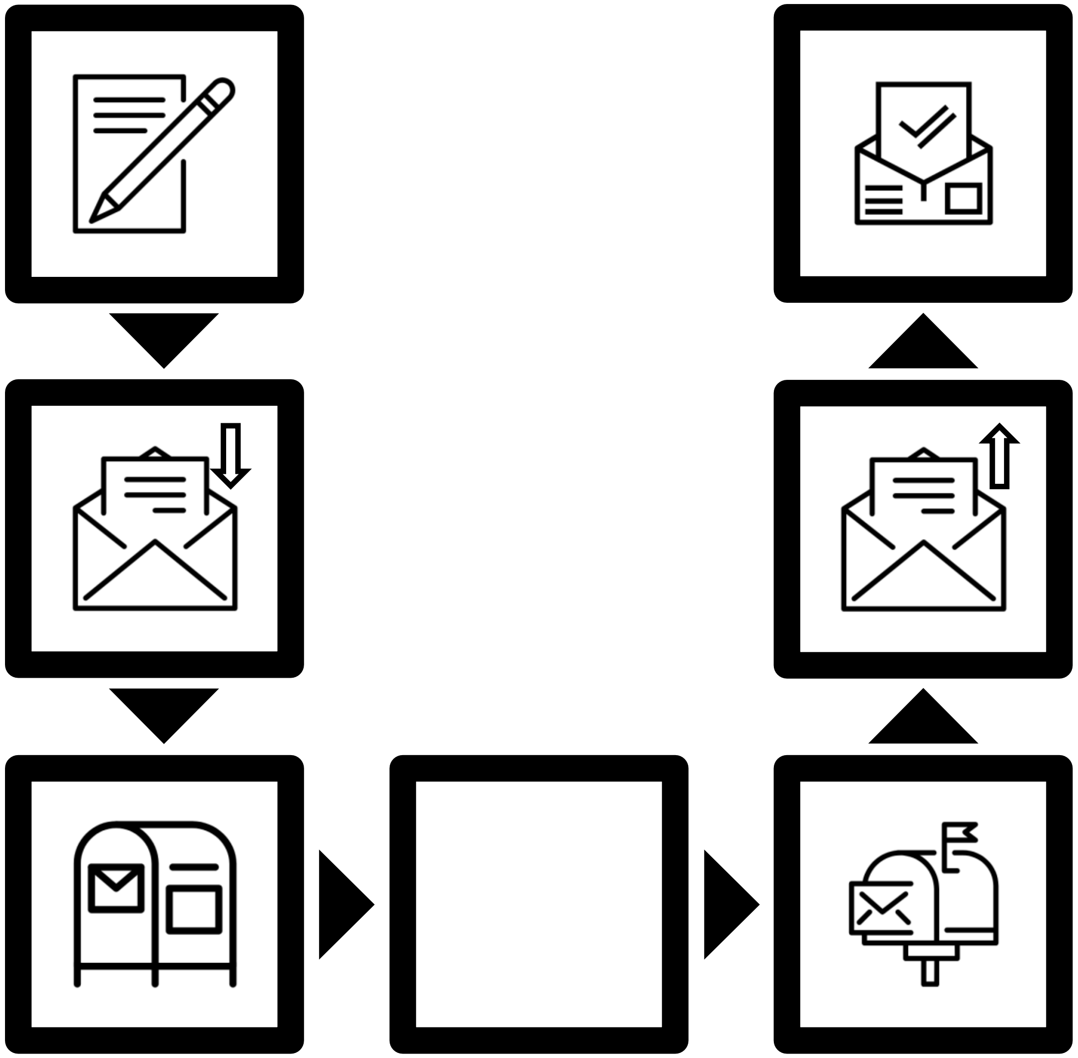
- How do you send a postcard?
  1. Write postcard
  2. Put postcard in envelope
  3. Mail envelope to recipient
  4. Recipient receives envelope
  5. Recipient opens envelope
  6. Recipient reads postcard



Layers & Abstraction

- How do you send a postcard?
  1. Write postcard
  2. Put postcard in envelope
  3. **<u>Mail envelope to recipient</u>**
  4. **<u>Recipient receives envelope</u>**
  5. Recipient opens envelope
  6. Recipient reads postcard

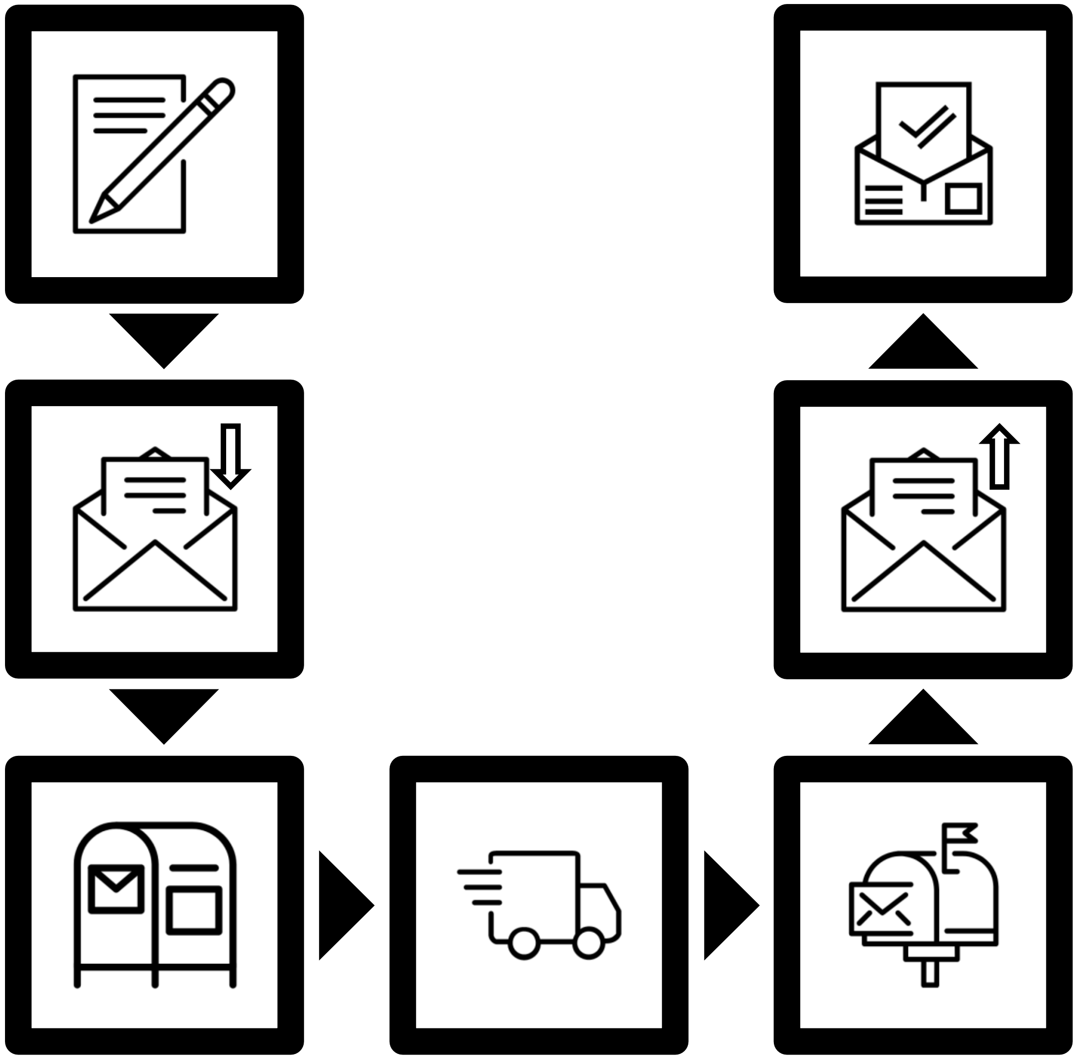
- How does the envelope get there?



Layers & Abstraction

- How do you send a postcard?
    1. Write postcard
    2. Put postcard in envelope
    3. **<u>Mail envelope to recipient</u>**
    4. **<u>Recipient receives envelope</u>**
    5. Recipient opens envelope
    6. Recipient reads postcard

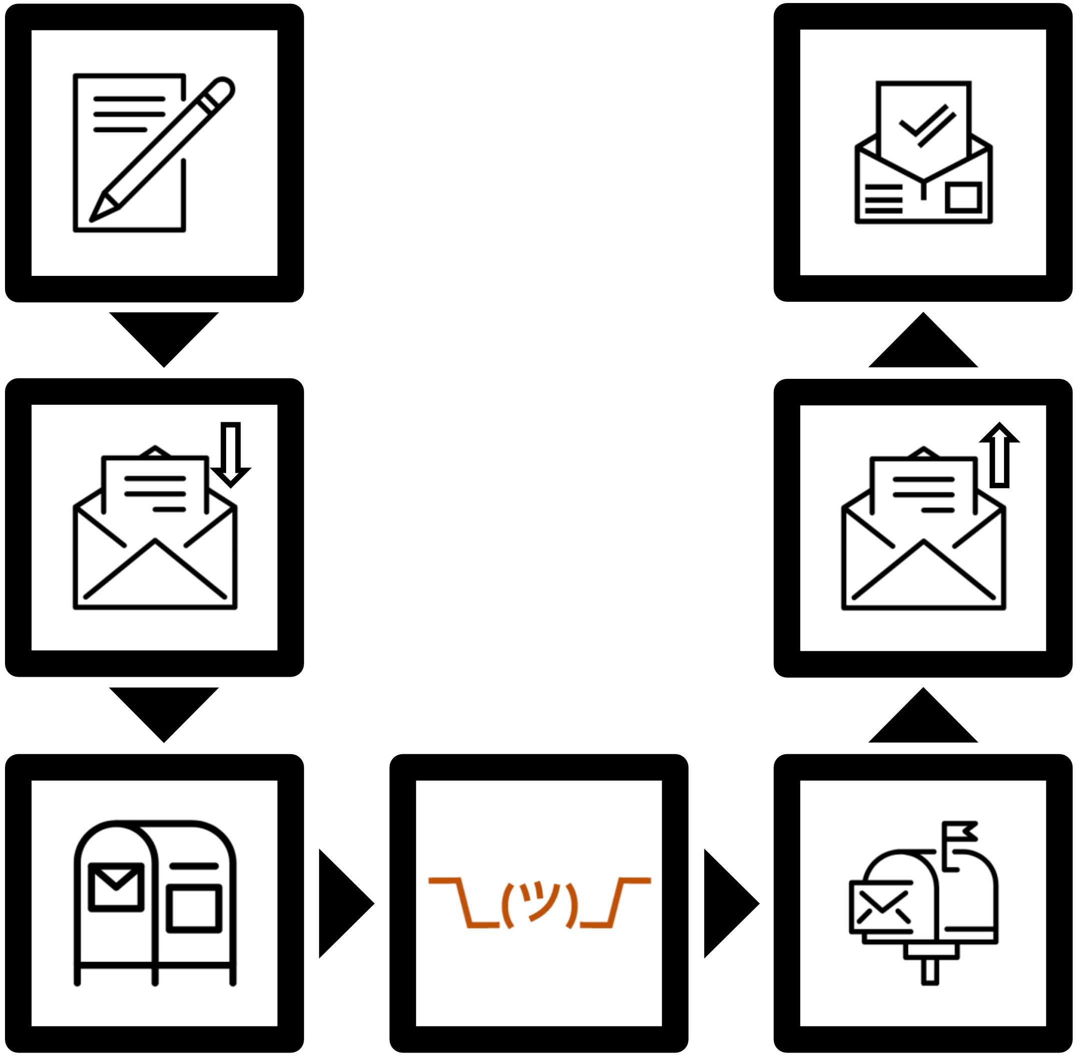
- How does the envelope get there?



Layers & Abstraction

- How do you send a postcard?
  1. Write postcard
  2. Put postcard in envelope
  3. **<u>Mail envelope to recipient</u>**
  4. **<u>Recipient receives envelope</u>**
  5. Recipient opens envelope
  6. Recipient reads postcard

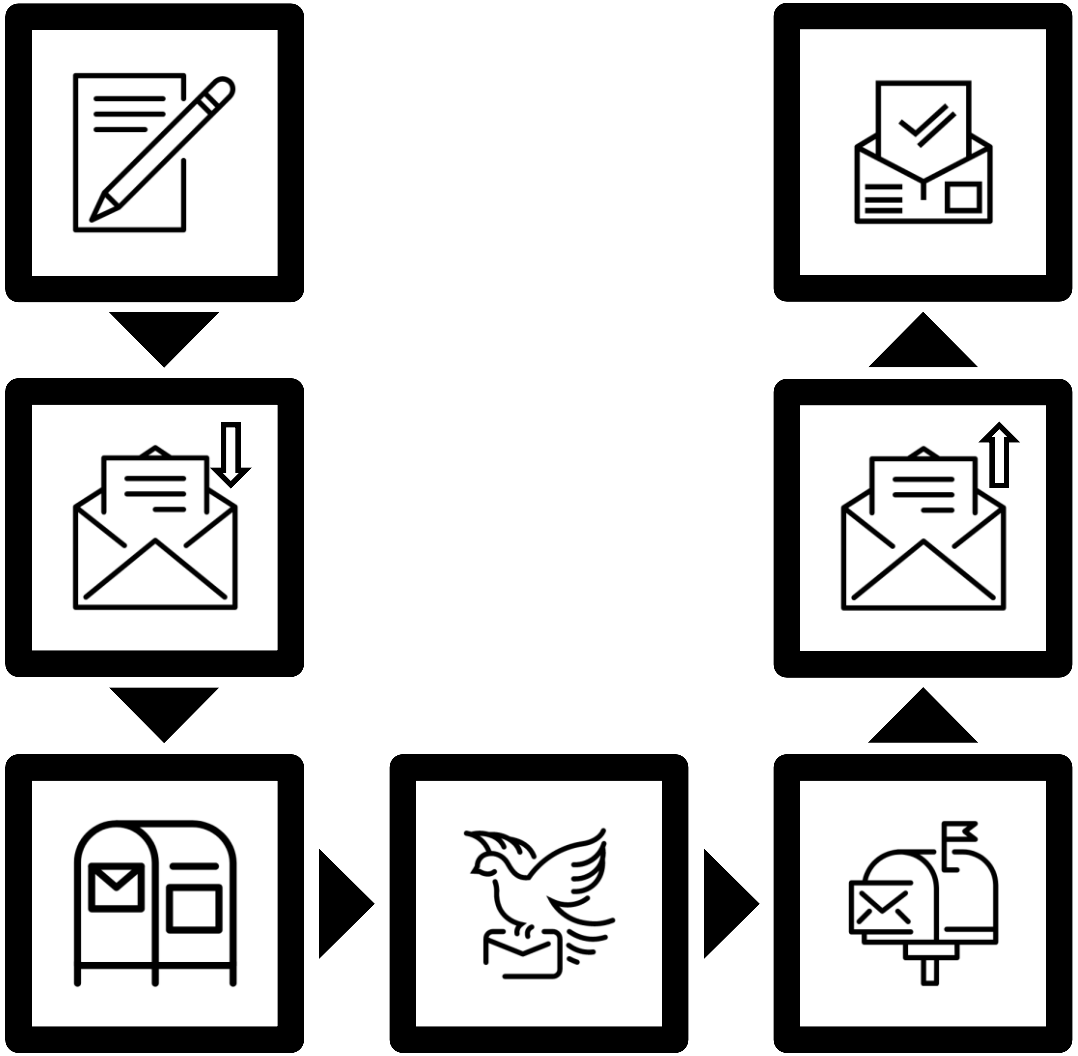
- How does the envelope get there?
  - We don’t care!



Layers & Abstraction

- How do you send a postcard?
  1. Write postcard
  2. Put postcard in envelope
  3. **Mail envelope to recipient**
  4. **Recipient receives envelope**
  5. Recipient opens envelope
  6. Recipient reads postcard

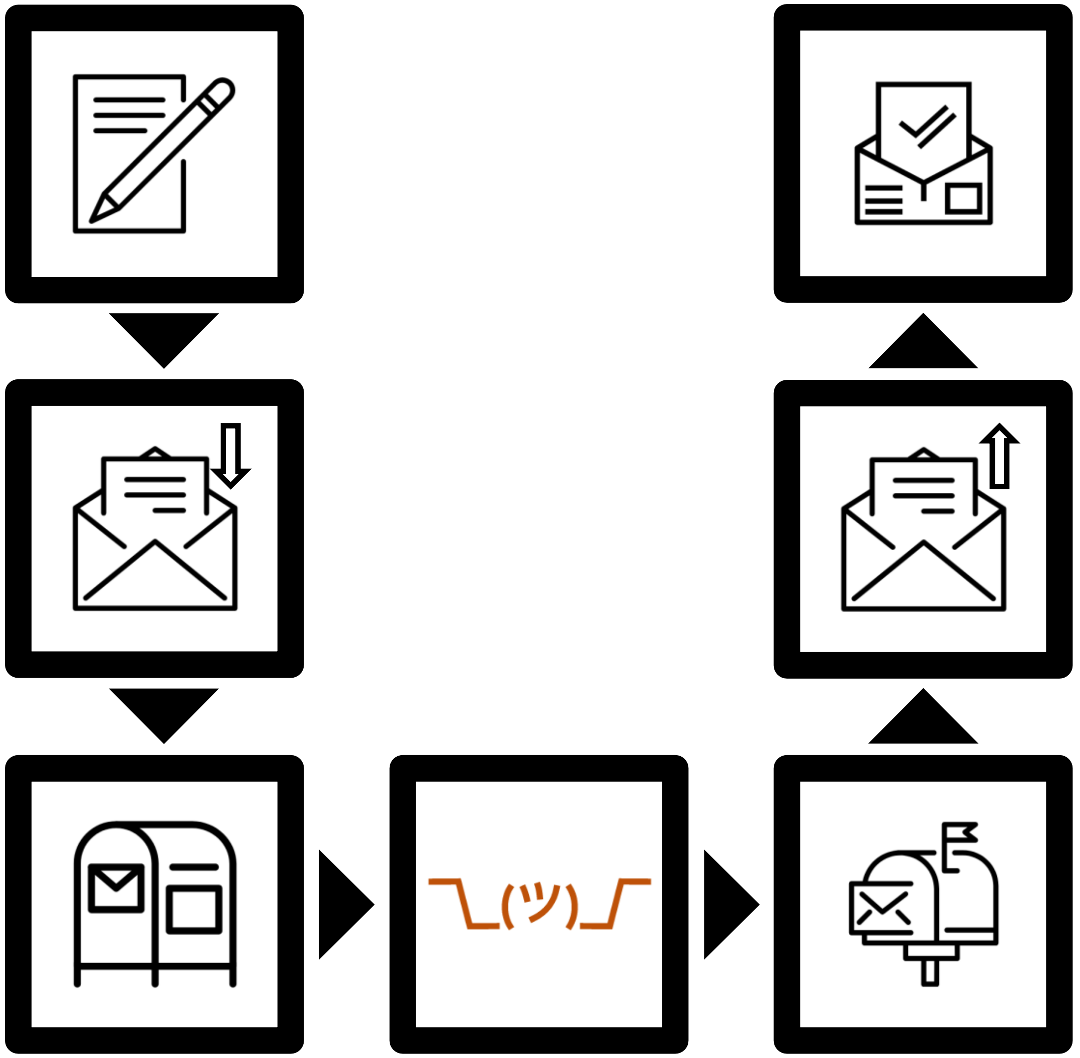
- Use a homing pigeon instead?



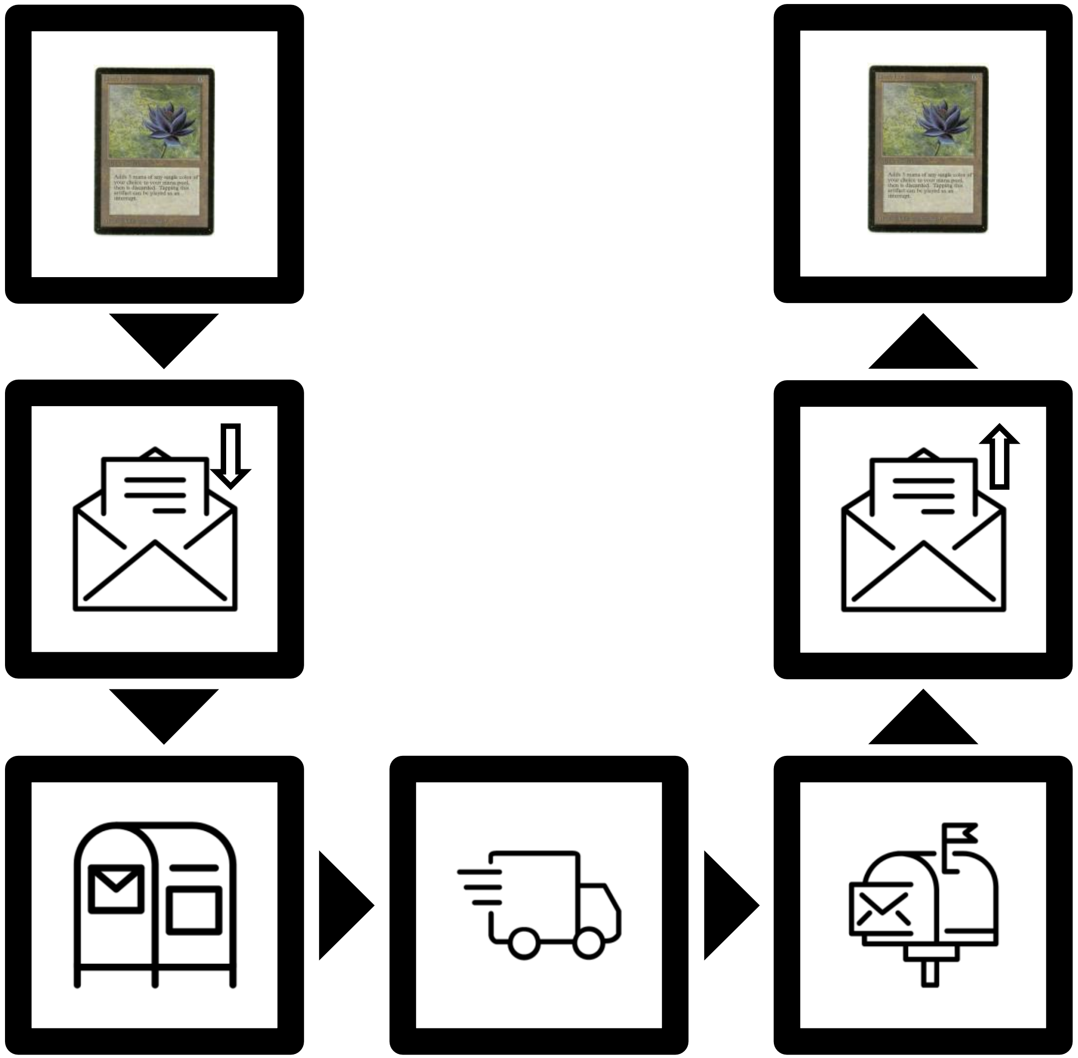
Layers & Abstraction

- How do you send a postcard?
  1. Write postcard
  2. Put postcard in envelope
  3. **Mail envelope to recipient**
  4. **Recipient receives envelope**
  5. Recipient opens envelope
  6. Recipient reads postcard

- Use a homing pigeon instead?
  - We don't care!

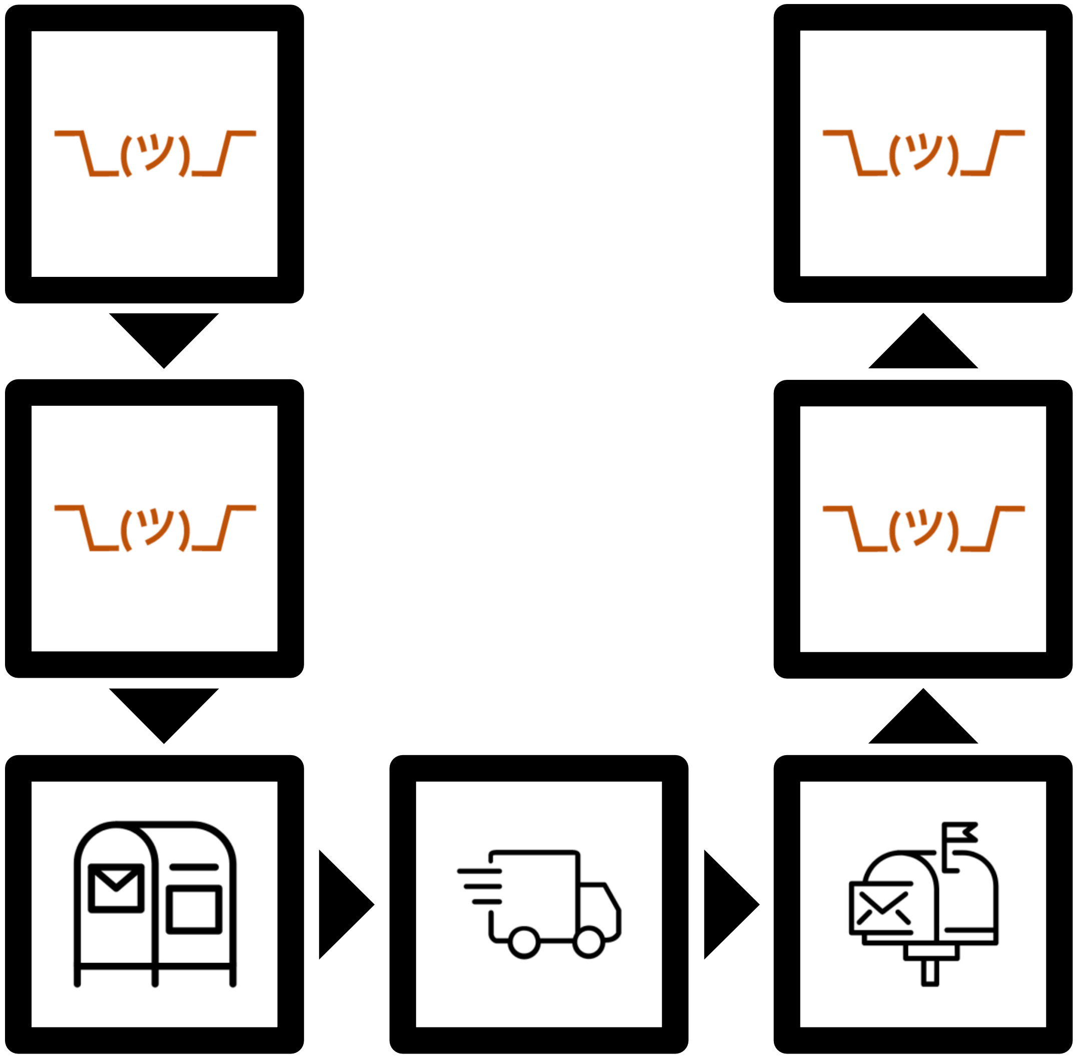


Layers & Abstraction



- Mail a trading card instead?

Layers & Abstraction



- Mail a trading card instead?
  - The post office doesn’t care!

Layers & Abstraction

- Division of responsibility
  - I don’t need to care how my envelope gets there
    - Transporting it is the post office’s job
  - The post office only needs to care about envelopes
    - Securing the something *inside* an envelope is my job

- No need to constantly re-invent the wheel!

Layers & Abstraction

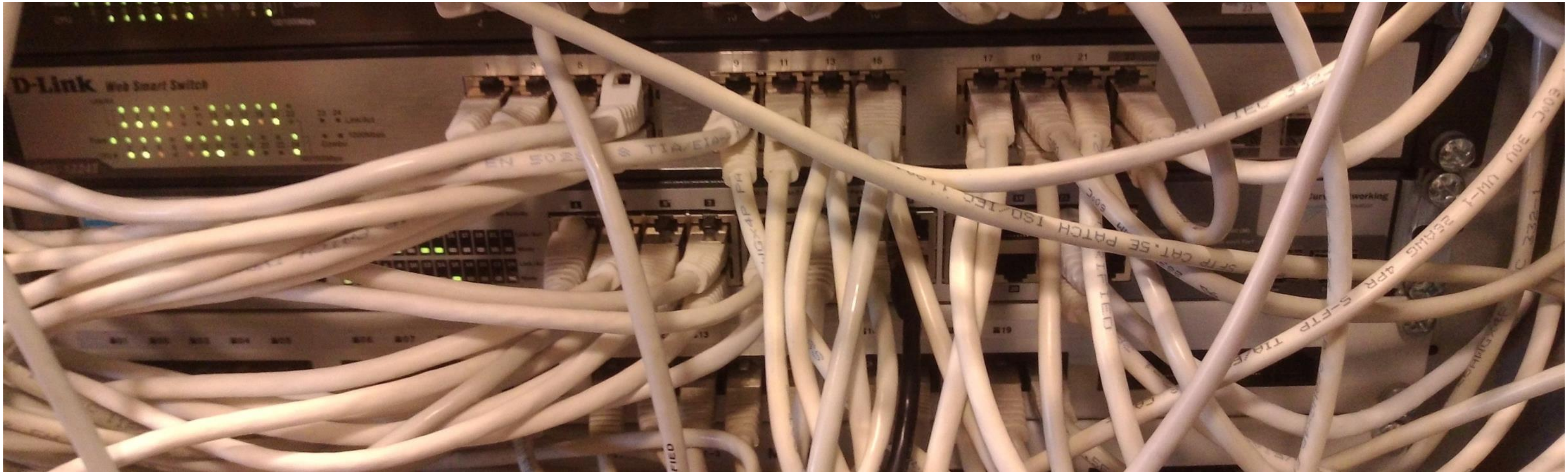
- Networking equivalent: **Layers**
- 1980/90s: competing models & protocol suites
  - TCP/IP, OSI, …
- Modern internet uses the TCP/IP model
  - So that's what we'll talk about!
  - Less powerful than OSI, but more flexible

The TCP/IP model

- Link layer
  - Send a chunk of data to a **directly connected** computer
- Internet layer
  - Route a chunk of data to a **remote** computer along a series of direct links
- Transport layer
  - Transmit a **structured** bit stream across the internet
- Application layer
  - **Offer services** without having to worry about details

Layers & Abstraction

- Abstraction:
  - Keeps complexity manageable
  - May introduce inefficiencies
  - Introduces rigidity
- Real-world protocols are not *fully* isolated from one another
  - Designers will consider properties of other layers’ common protocols



The Link Layer

The Link Layer

- Computers A, B, C, etc. are all “connected” to each other
- **Goal:** Send data from A to C

- Properties of the medium:
  - If you speak, can “everyone” hear you?

Shared Medium

Switched Medium

Addressing

An Ethernet frame, common on the modern Internet

- An *address* identifies a destination
  - Shared Medium: recipients can recognize messages
  - Switched Medium: we know where to send messages

- **MAC address**: 48-bit identifier
  - Used in: Ethernet, Wi-Fi, Bluetooth, …
  - Should be locally unique

- **Broadcast address**: `FF:FF:FF:FF:FF:FF`
  - Will be sent to all connected hosts

Image: Public Domain

The Link Layer

- Computers A, B, C, etc. are all “connected” to each other
- **Goal:** Send data from A to C

- Properties of the medium:
  - If you speak, can “everyone” hear you? (shared or switched medium)
  - Can you send and receive at the same time? (“half-duplex” vs “full-duplex”)
  - Can you send and listen at the same time? (collision detection)

- Concerns:
  - Was the data distorted over the “wire”? (integrity)

Checksums

An Ethernet frame, common on the modern Internet

- A *checksum* is…
  - … a fixed-size value
  - … calculated based on arbitrary data

- It allows us to detect errors!
  - Each message is sent with its correct checksum
  - If random bits get flipped, the checksum is no longer correct!

- This doesn’t help against an intelligent attacker!

Image: Public Domain

Example: Wi-Fi (IEEE 802.11)

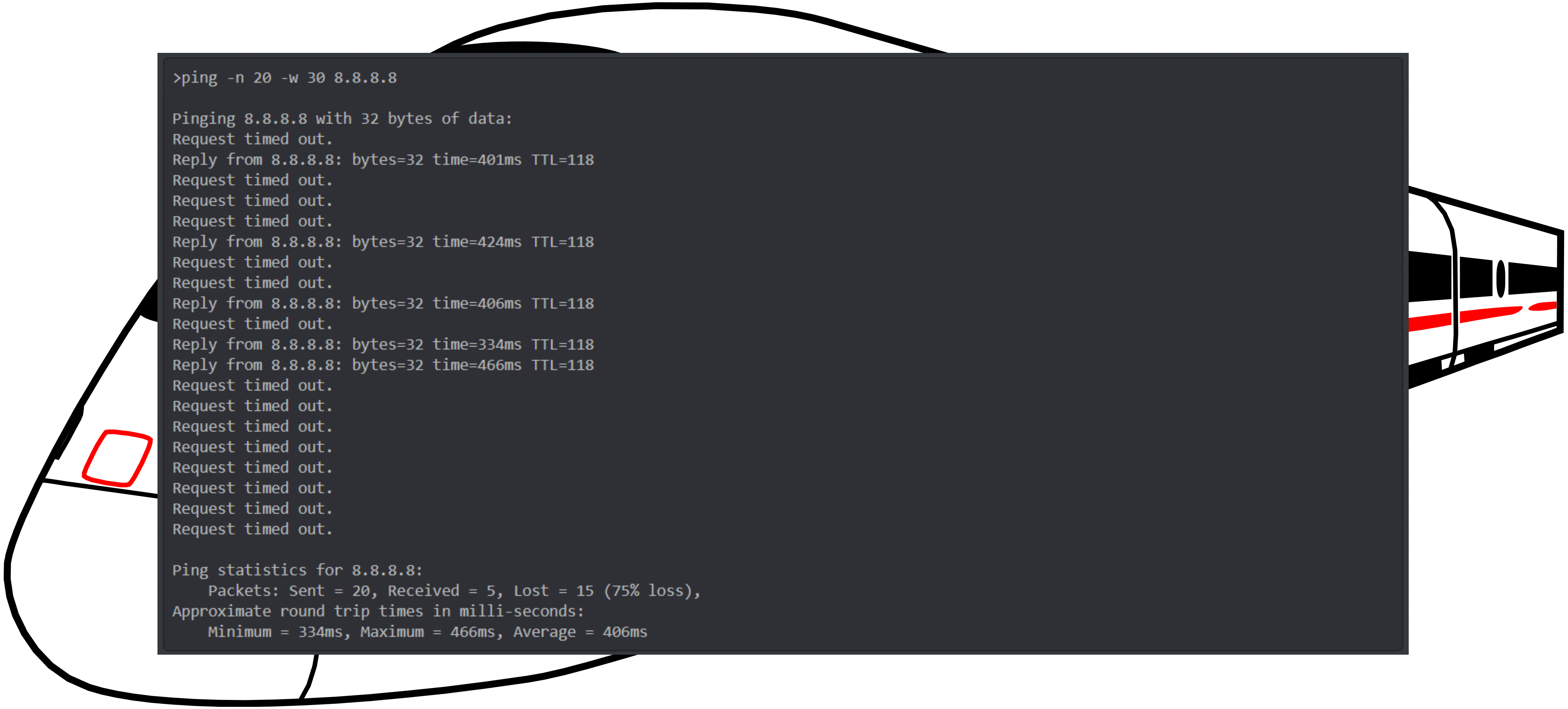
Image: Public Domain

- Shared medium: Wireless radio
- Central access point
  - Nodes communicate via the AP
- Not full-duplex
  - If two nodes send at the same time, the signals are garbled
- No direct collision detection
  - If a node is sending, it cannot listen for transmissions at the same time
- Data is acknowledged
  - Collision -> no acknowledgment -> Data re-sent

Example: Wi-Fi (IEEE 802.11)

Example: Wi-Fi (IEEE 802.11)

Example: Wi-Fi (IEEE 802.11)

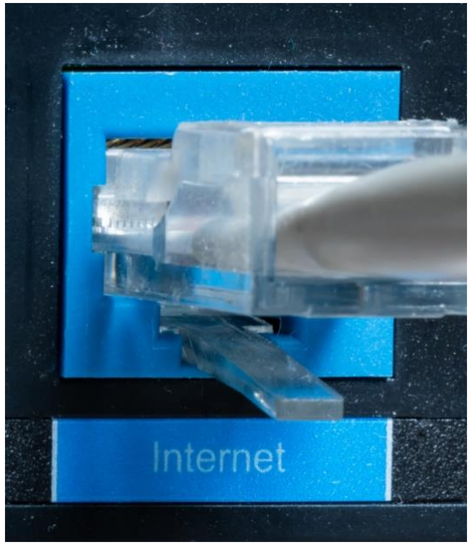


```
>ping -n 20 -w 30 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Request timed out.
Reply from 8.8.8.8: bytes=32 time=401ms TTL=118
Request timed out.
Request timed out.
Request timed out.
Reply from 8.8.8.8: bytes=32 time=424ms TTL=118
Request timed out.
Request timed out.
Reply from 8.8.8.8: bytes=32 time=406ms TTL=118
Request timed out.
Reply from 8.8.8.8: bytes=32 time=334ms TTL=118
Reply from 8.8.8.8: bytes=32 time=466ms TTL=118
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 8.8.8.8:
    Packets: Sent = 20, Received = 5, Lost = 15 (75% loss),
Approximate round trip times in milli-seconds:
    Minimum = 334ms, Maximum = 466ms, Average = 406ms
```

Example: Ethernet (IEEE 802.3)



- Star-shaped structure
  - Clients directly connected to one or more *switches*
  - Hardware failure only disconnects that client
- Full-duplex (in modern networks)
  - No collisions possible
- Switched medium (mostly, in modern networks)
  - We'll talk details in a bit

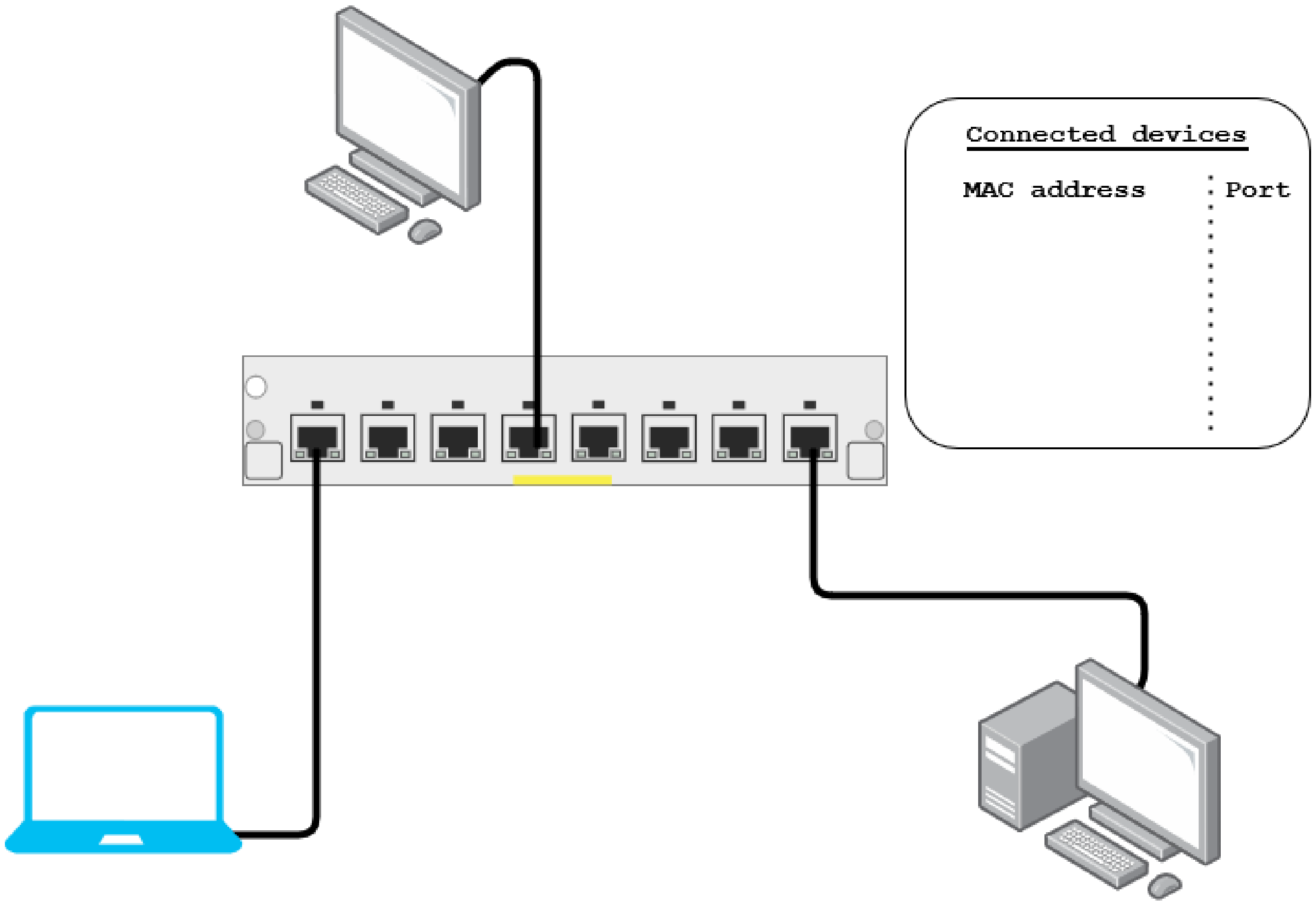
Ye Olde Ethernette

- Once upon a time, Ethernet was a shared medium…
  - At first, it used a single coaxial cable…
    - Physically connecting all the hosts!
  - Later, it used *Ethernet hubs* that emulated this…
    - Simply re-broadcast any received signal to all ports
- We interconnect hundreds of computers
  - Only one can talk at a time?

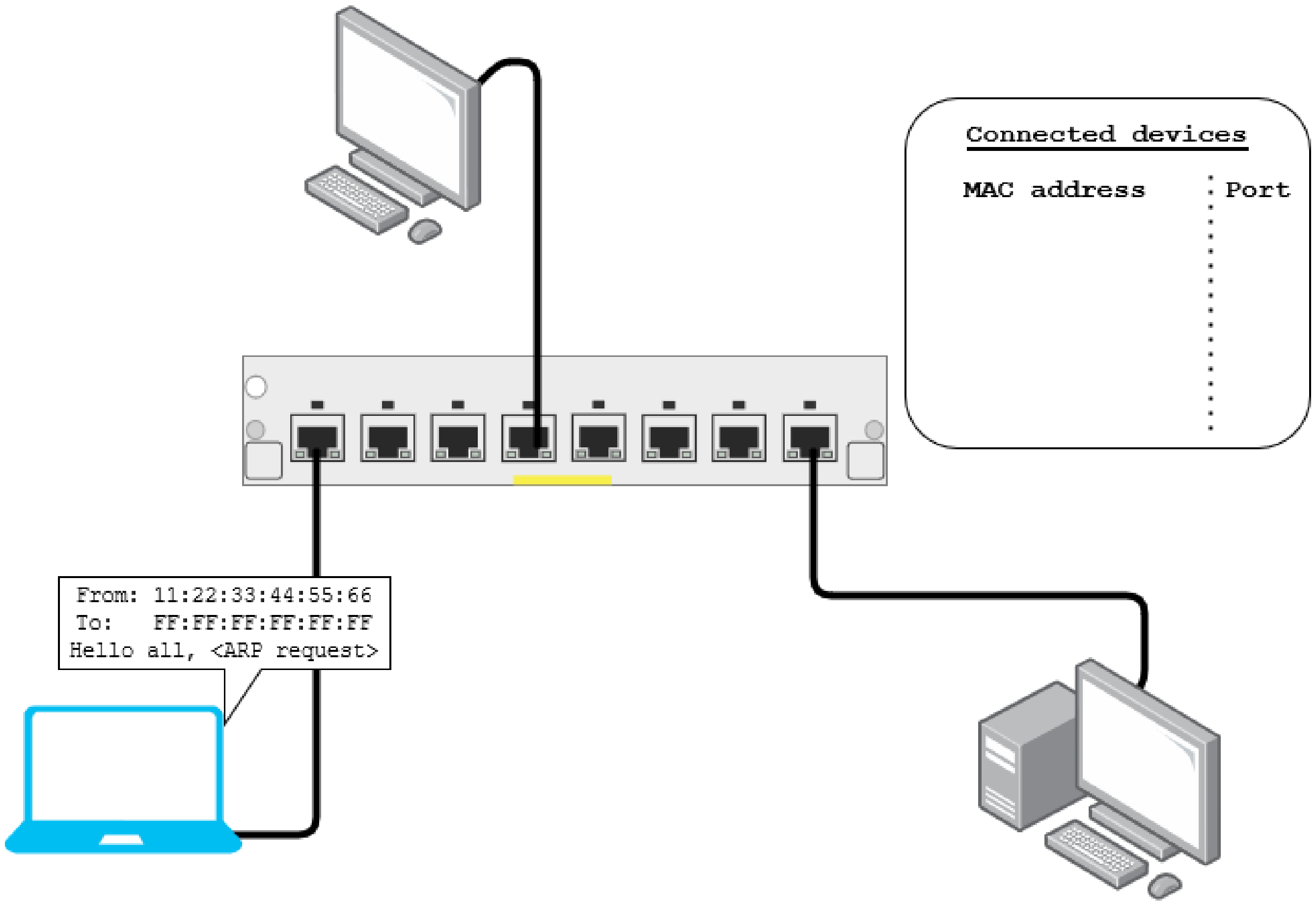
Ethernet: Switching

- Ethernet *switches* understand Link Layer data
    - Read source/destination MAC addresses
- Record source addresses to build map address <-> port
- Only forward packets to the appropriate port
    - Minimize wasted bandwidth
    - No collisions possible!

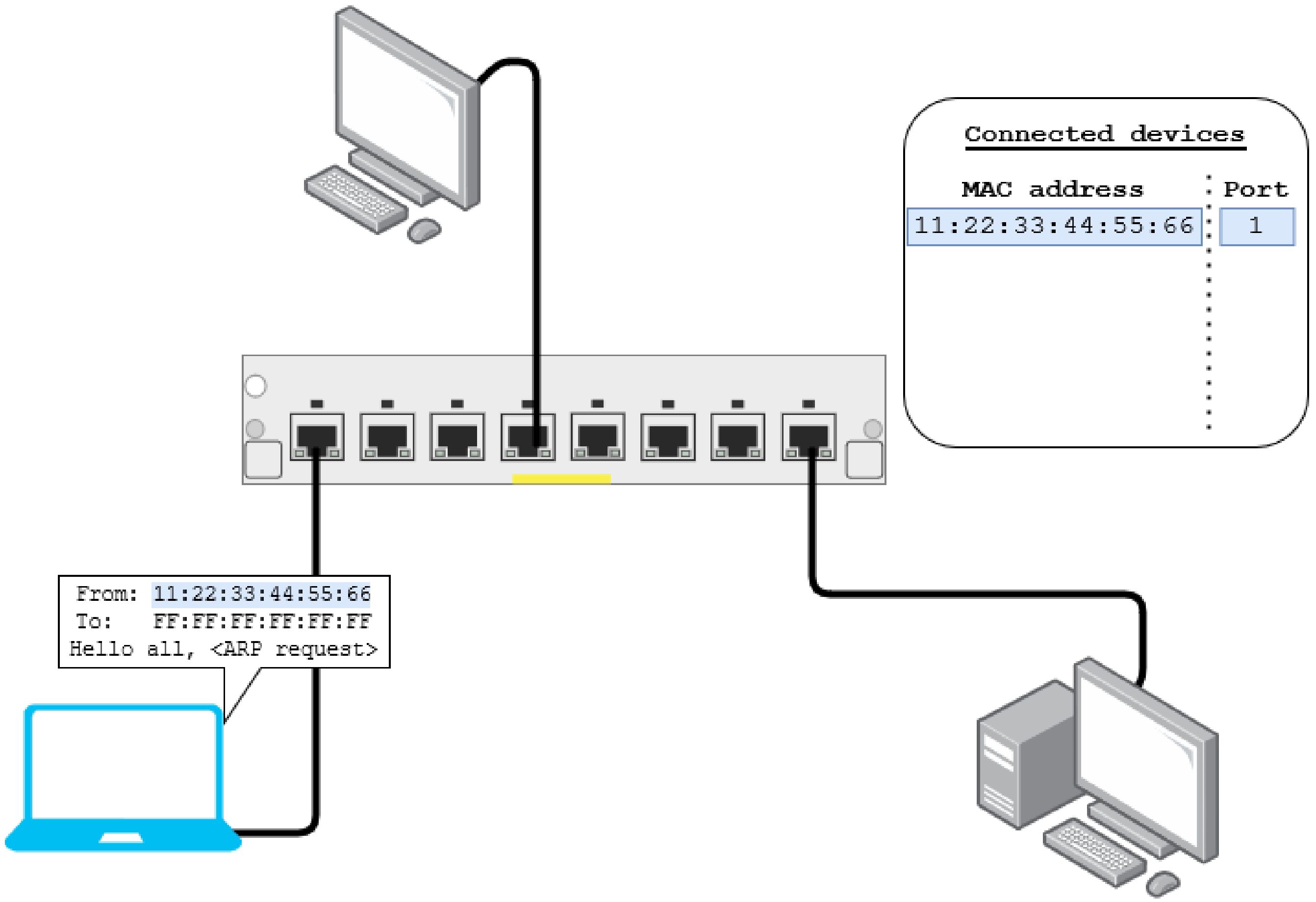
Ethernet: Switching



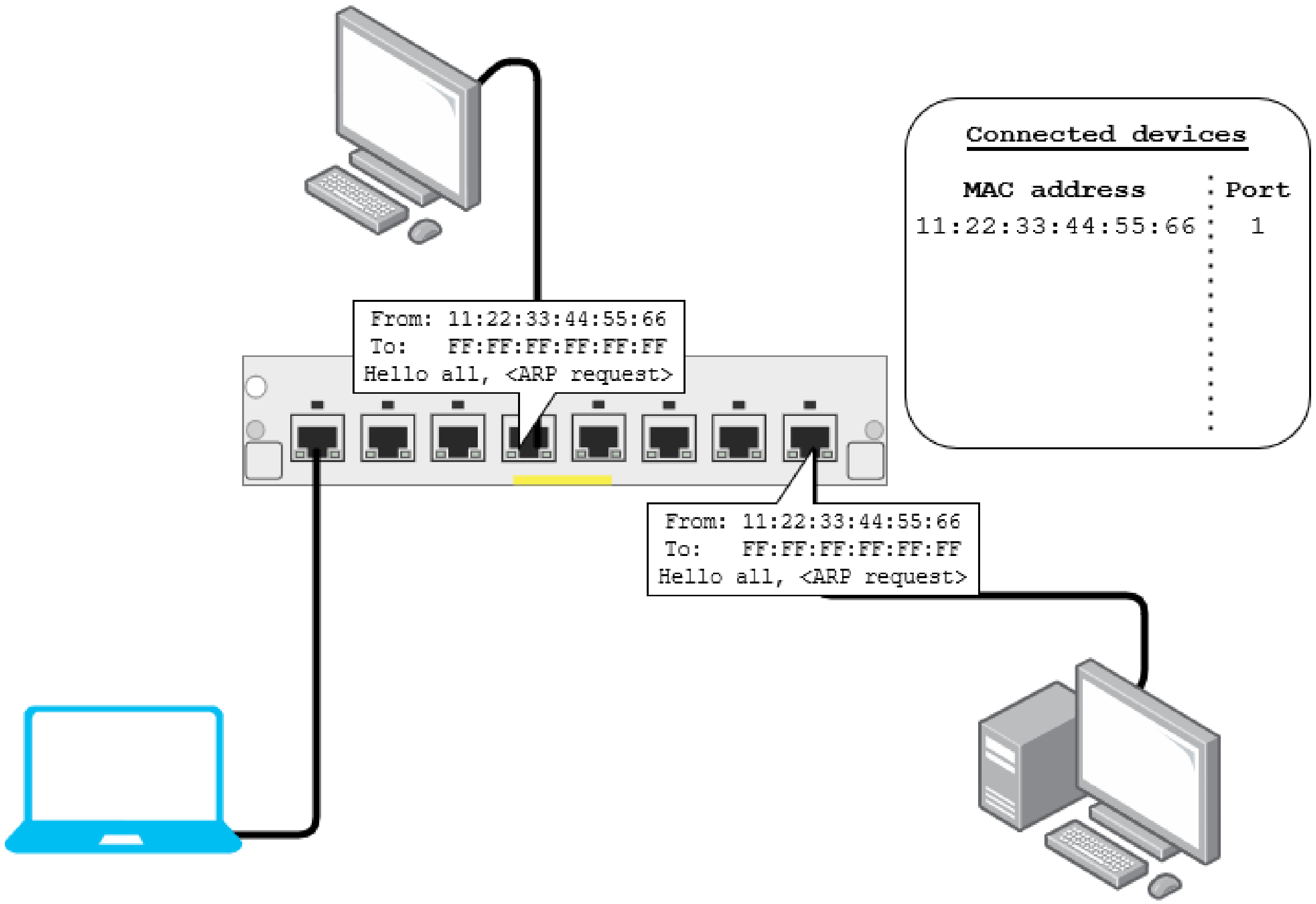
Ethernet: Switching



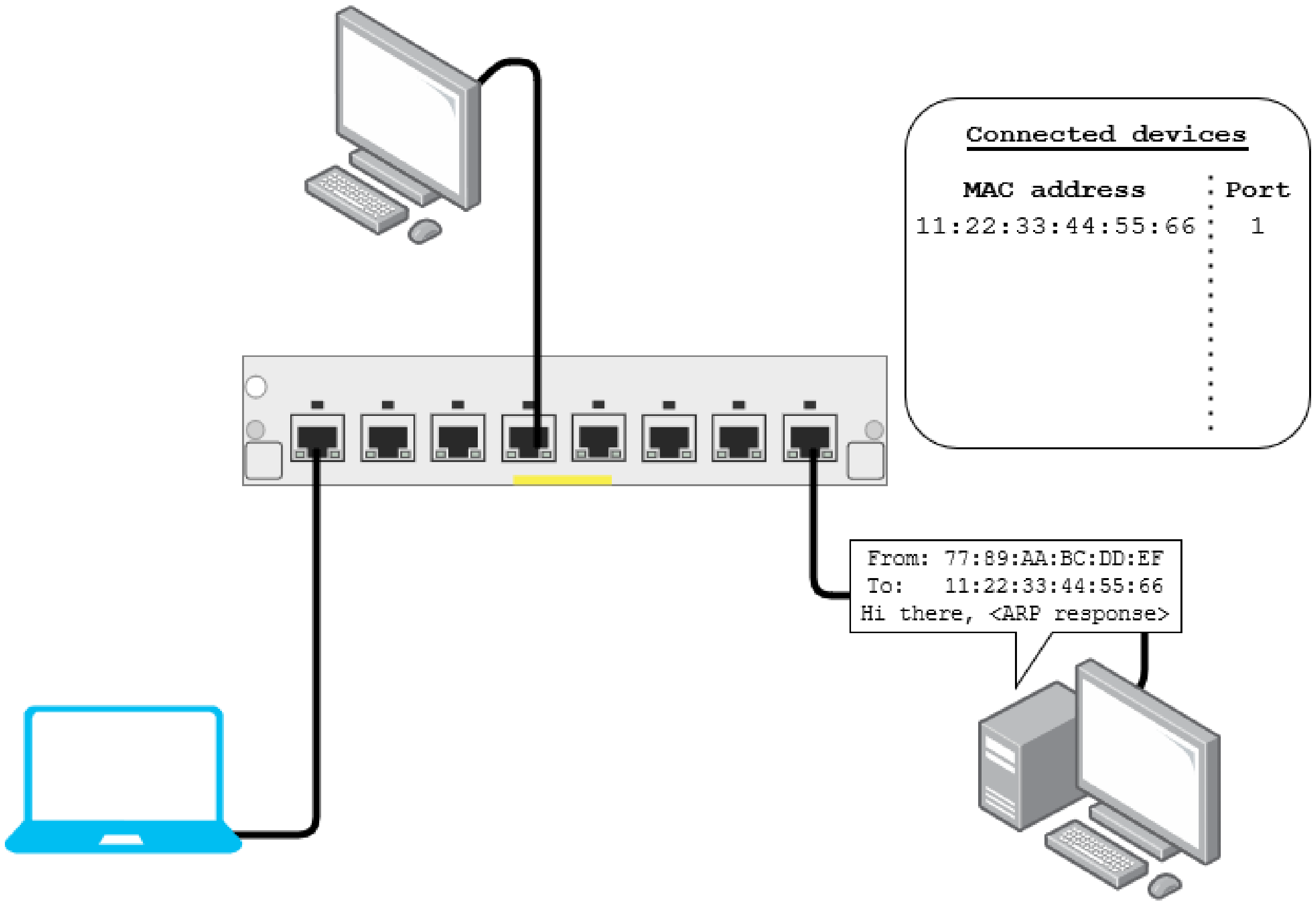
Ethernet: Switching



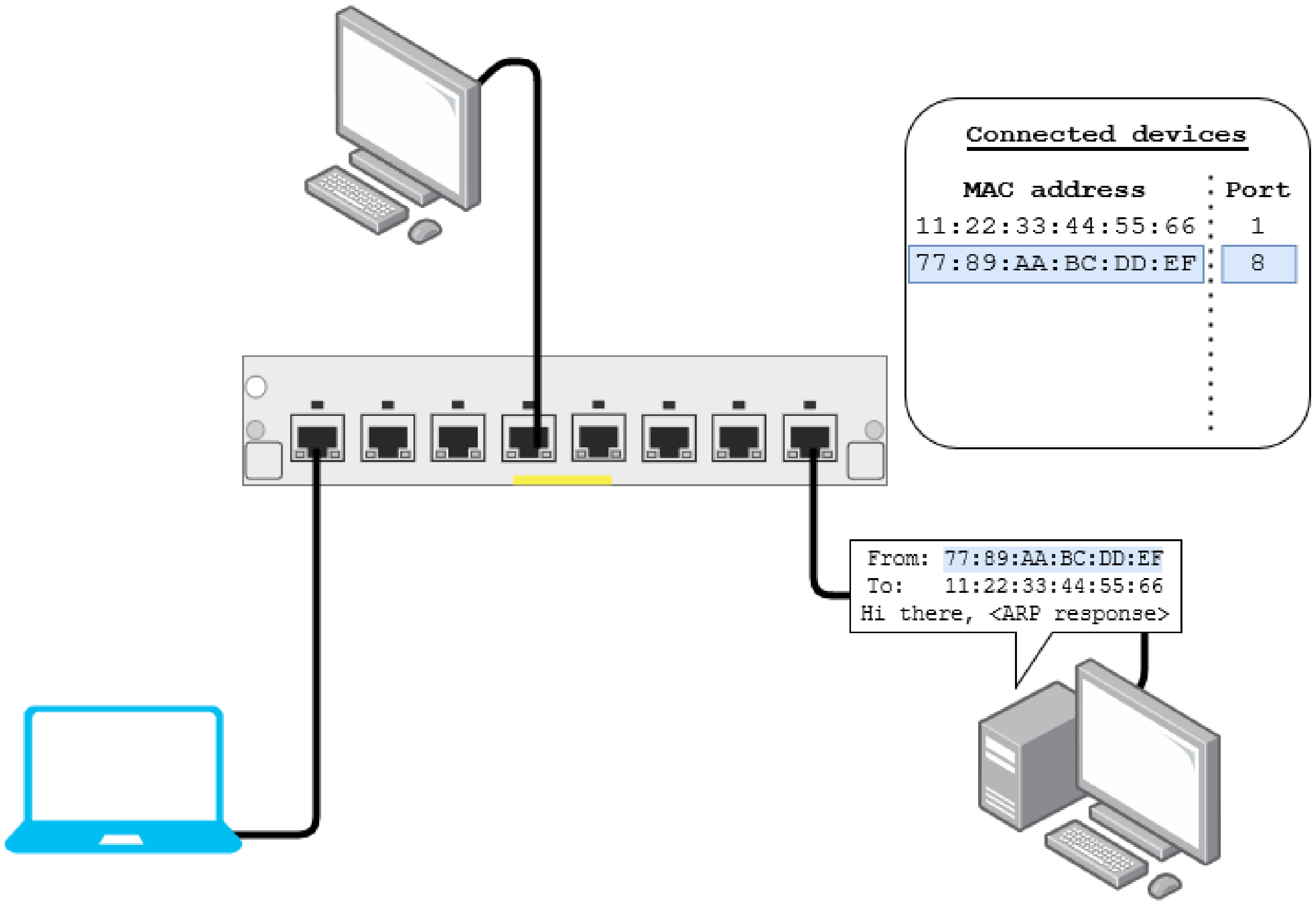
Ethernet: Switching



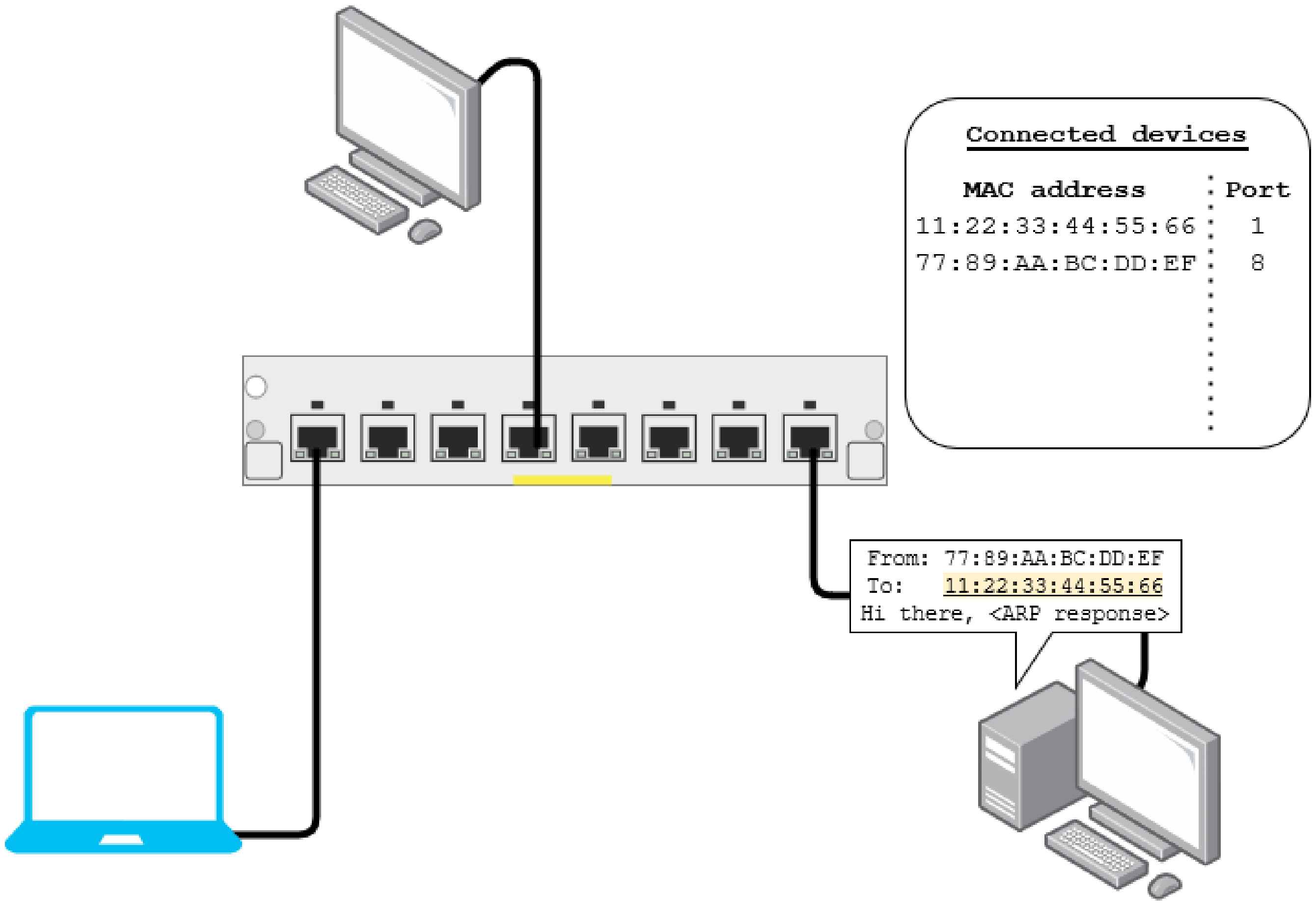
Ethernet: Switching



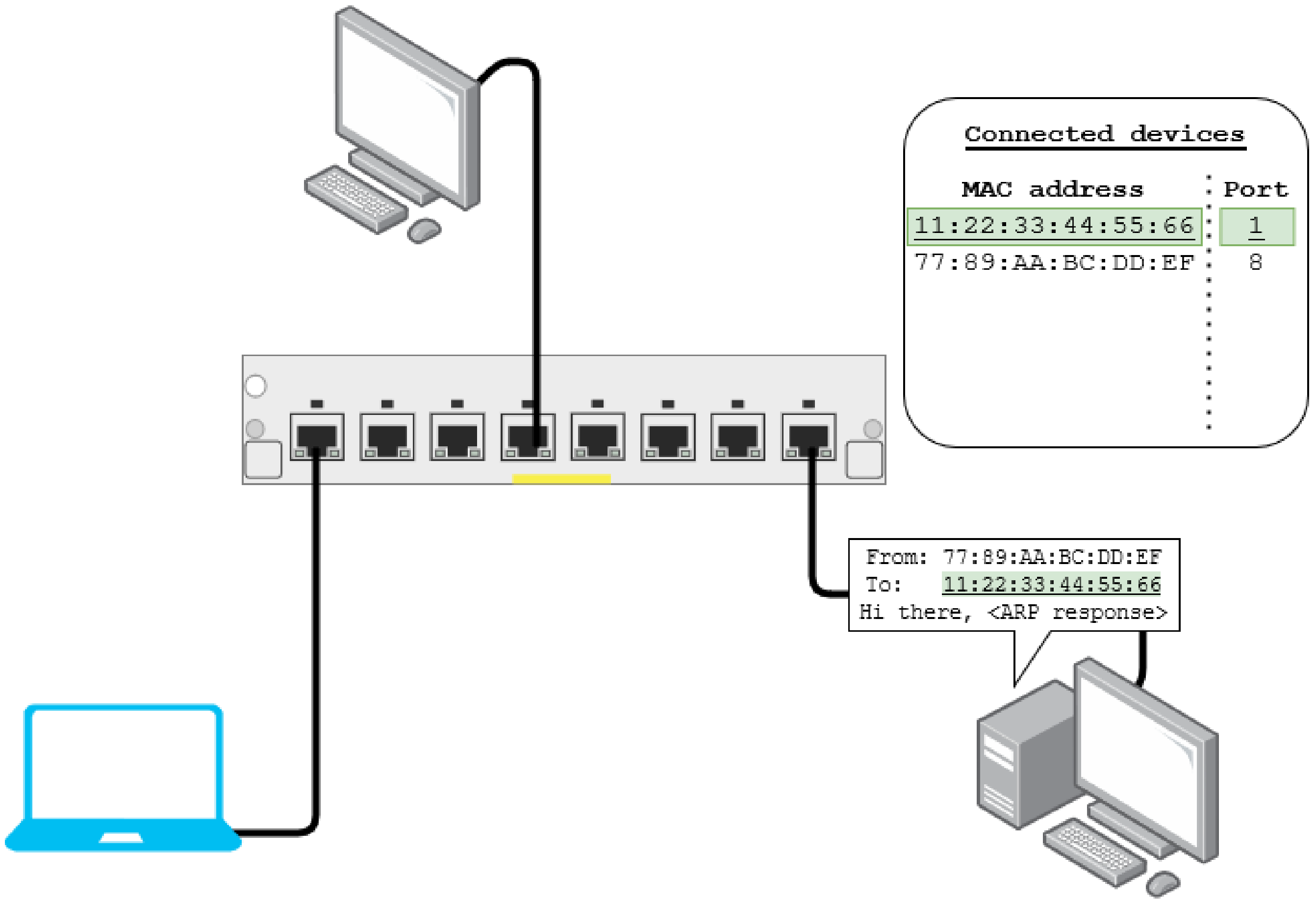
Ethernet: Switching



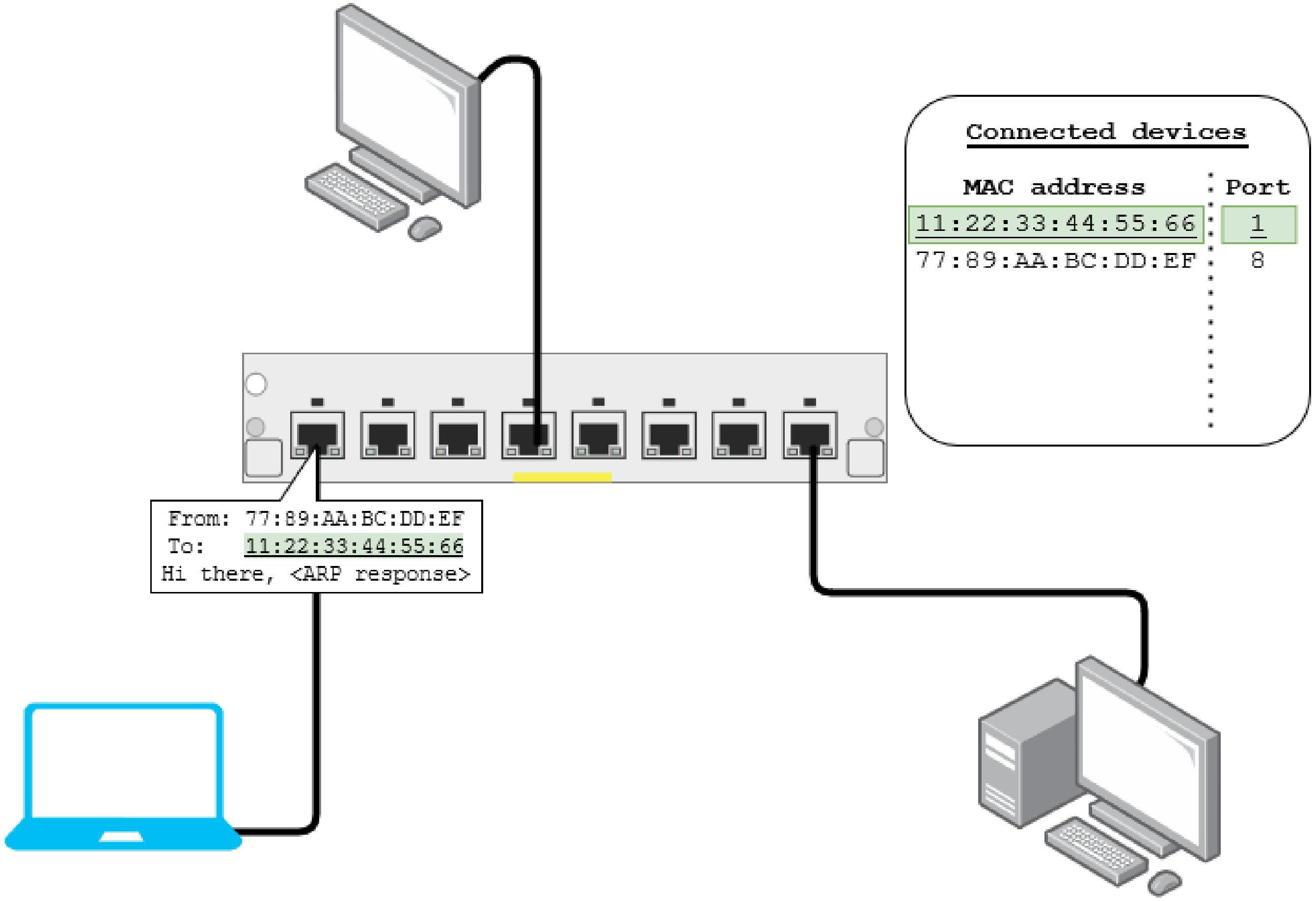
Ethernet: Switching



Ethernet: Switching

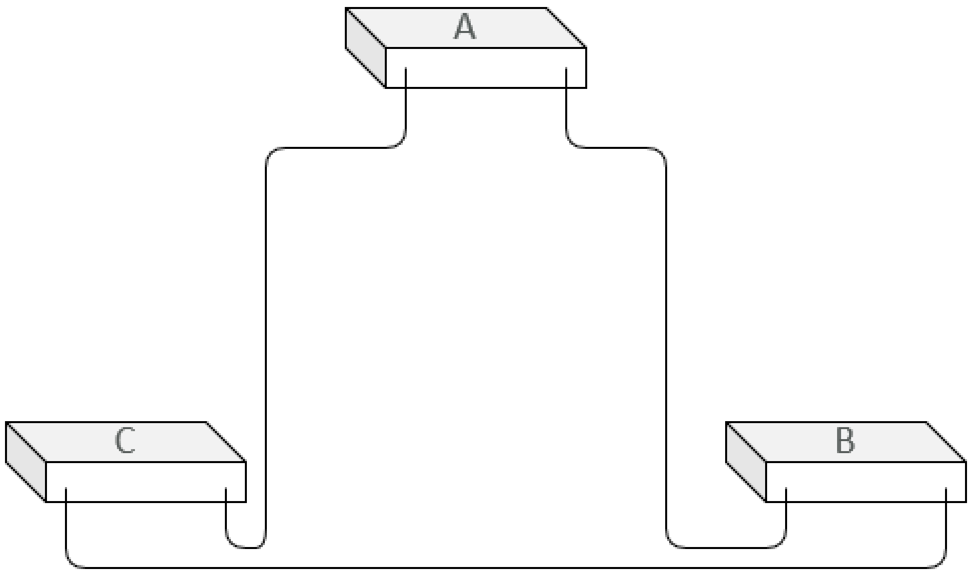


Ethernet: Switching



Ethernet: Switching Loops

- Multiple switches can be interconnected to form one big network
- **Problem:** switching loops
  - Why is it a problem? Broadcasts!
  - If a broadcast frame reaches this topology, it will multiply endlessly
- **Solution:** don't build switching loops!
  - However, they are useful for redundancy
- **S**panning **T**ree **P**rotocol
  - Supported by professional switches
  - Automatically disables redundant links until needed



Ethernet: V-LAN

- **V**irtual **LAN**s
  - Partition switch ports into different logical networks
  - Devices on different networks cannot send packets to each other
  - Broadcast packets are only broadcast to the device's VLAN
- Benefits
  - Partitioned networks
  - No re-wiring required
  - Configured in software
- Downsides
  - Configured in software

The Network Layer

The Network Layer

- Computers A and B are connected to different physical networks
- There is some way to get from A’s network to B’s network
- **Goal:** Send data from A to B

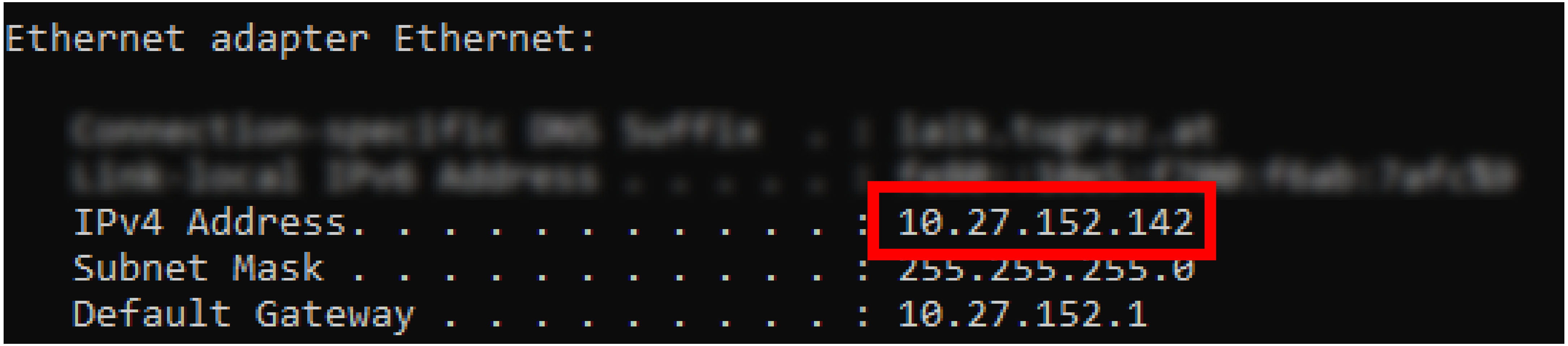
- Concerns:
  - How does the data get from A to B? (routing)
  - What if the data is too large for a certain path? (fragmentation)

IPv4

- **I**nternet **P**rotocol, **v**ersion **4**
- Foundation of today's internet

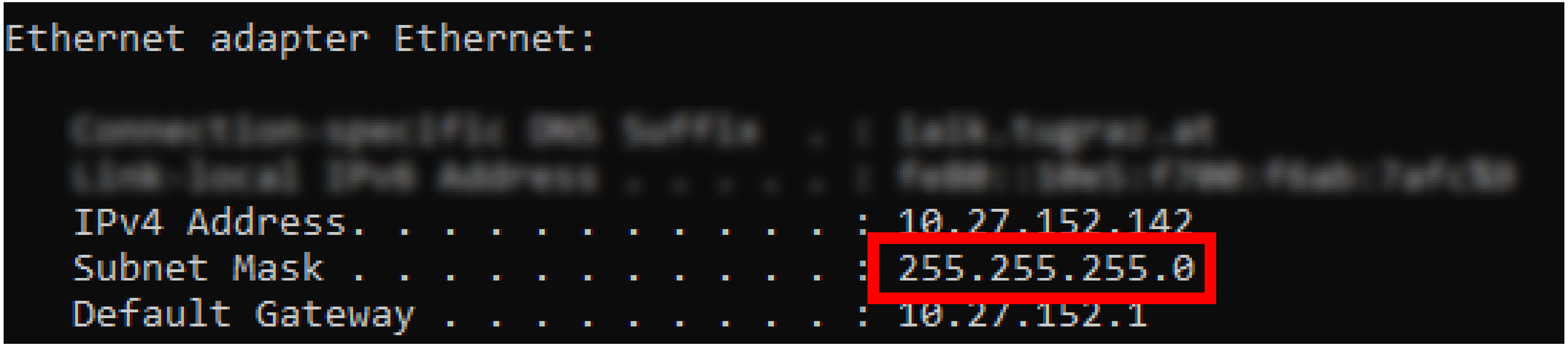
- Used in almost every network-enabled device

IPv4 addressing



- 32-bit address
  - Notation: bytes’ decimal value (0-255)
  - **`10.27.152.142`** is the same as **`0a 1b 98 8e`**
- Each participating network card has a single IPv4 address

IPv4 addressing



- 32-bit subnet mask
  - All ones, followed by all zeros
  - Splits address into *network prefix* and *host number*
  - Alternate notation: just specify number of ones
    - **`255.255.255.0`** is the same as **`/24`**

IPv4 addressing

Network prefix				Host number
10	. 27	. 152	.	142
00001010	.00011011	.10011000	.	10001110
11111111	**.11111111**	**.11111111**	.	00000000
255	. 255	. 255	.	0

IPv4 addressing

- All hosts with the same network prefix form a *subnet*
- Hosts within the same subnet can communicate directly
  - They're in the same Link Layer network!

- Two addresses per subnet have special meaning
  - Host number all zeros ≙ network identifier
    - `10.27.152.142/24` is part of the `10.27.152.0/24` network
  - Host number all ones ≙ broadcast address
    - `10.27.152.255/24` is the broadcast address for the `10.27.152.0/24` network

IPv4 addressing

- Subnet masks do not need to be full bytes
  - **`255.255.255.240`** (28 bits network prefix, 4 bits host number ≙ **`/28`**)
  - **`192.168.13.80/28`** can have up to 14 host addresses
    - Network address: **`192.168.13.80`** (**`80 ≙ 01010000`**)
    - First host address: **`192.168.13.81`** (**`81 ≙ 01010001`**)
    - Last host address: **`192.168.13.94`** (**`94 ≙ 01011110`**)
- Not every broadcast address ends with **`.255`**!
  - What is the broadcast address for **`192.168.195.0/28`**?
- Not every address that ends with **`.255`** is a broadcast address!
  - **`10.5.0.255/16`** is the 255th host in the **`10.5.0.0/16`** subnet

IPv4 addressing

- Need addresses for your home?
  - Private address space that anyone can use:
    - `10.0.0.0/8` (i.e., `10.0.0.0` to `10.255.255.255`)
    - `172.16.0.0/12` (i.e., `172.16.0.0` to `172.31.255.255`)
    - `192.168.0.0/16` (i.e., `192.168.0.0` to `192.168.255.255`)
  - Not globally unique
    - Won't work over the internet!
- Never configured an IP address before?
  - Your ISP modem likely does this for you!
  - **D**ynamic **H**ost **C**onfiguration **P**rotocol
  - Enabled by default on modern devices

IPv4 addressing

- Destination address in my subnet?
  - Talk to it using Data Link Layer

- …talk to it using Data Link Layer?
  - We only have an IP address
  - At the Data Link Layer, we need a MAC address

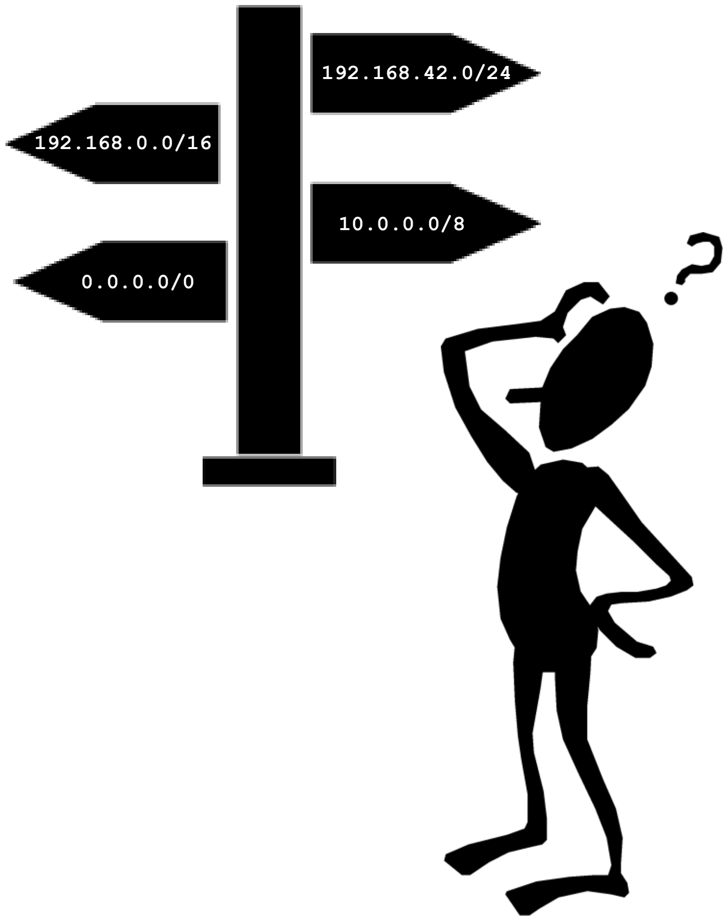
IPv4 addressing

- Destination address in my subnet?
    - Talk to it using Data Link Layer

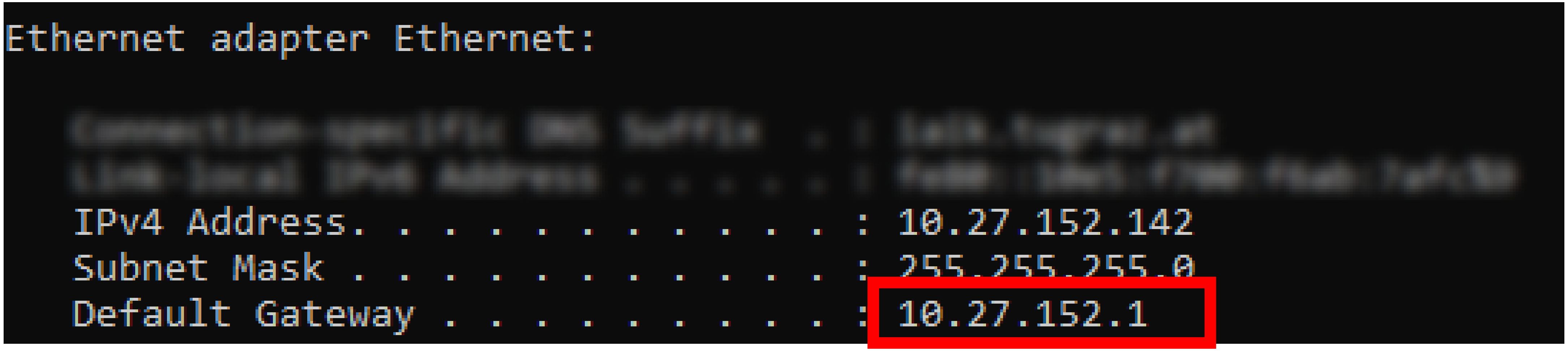
- **A**ddress **R**esolution **P**rotocol
    - Ethernet frames with type `0x0806`
    - Very simple stateless protocol
        - Request MAC for given IP (Ethernet broadcast)
        - Target responds (Ethernet unicast), now we know its MAC address
    - Heavily cached to avoid lots of broadcasting

IPv4 routing

- Destination address in my subnet?
  - Talk to it using Data Link Layer
- Destination address *not* in my subnet?
  - Check *routing table*
  - Maps destination address to *next hop*
    - Move packet in “the right direction”
  - Send packet to next hop using Data Link Layer
  - Eventually it gets there

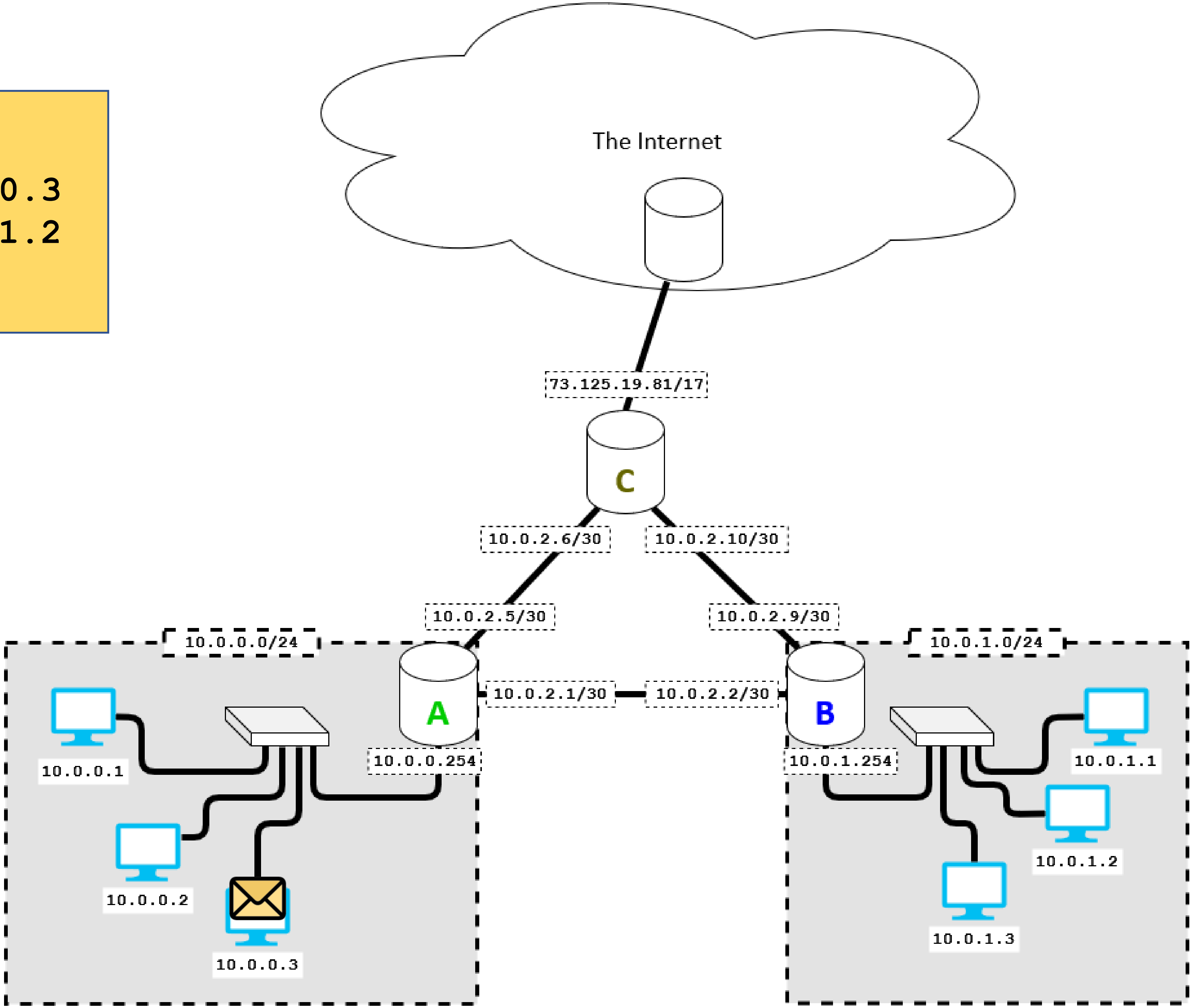


IPv4 routing

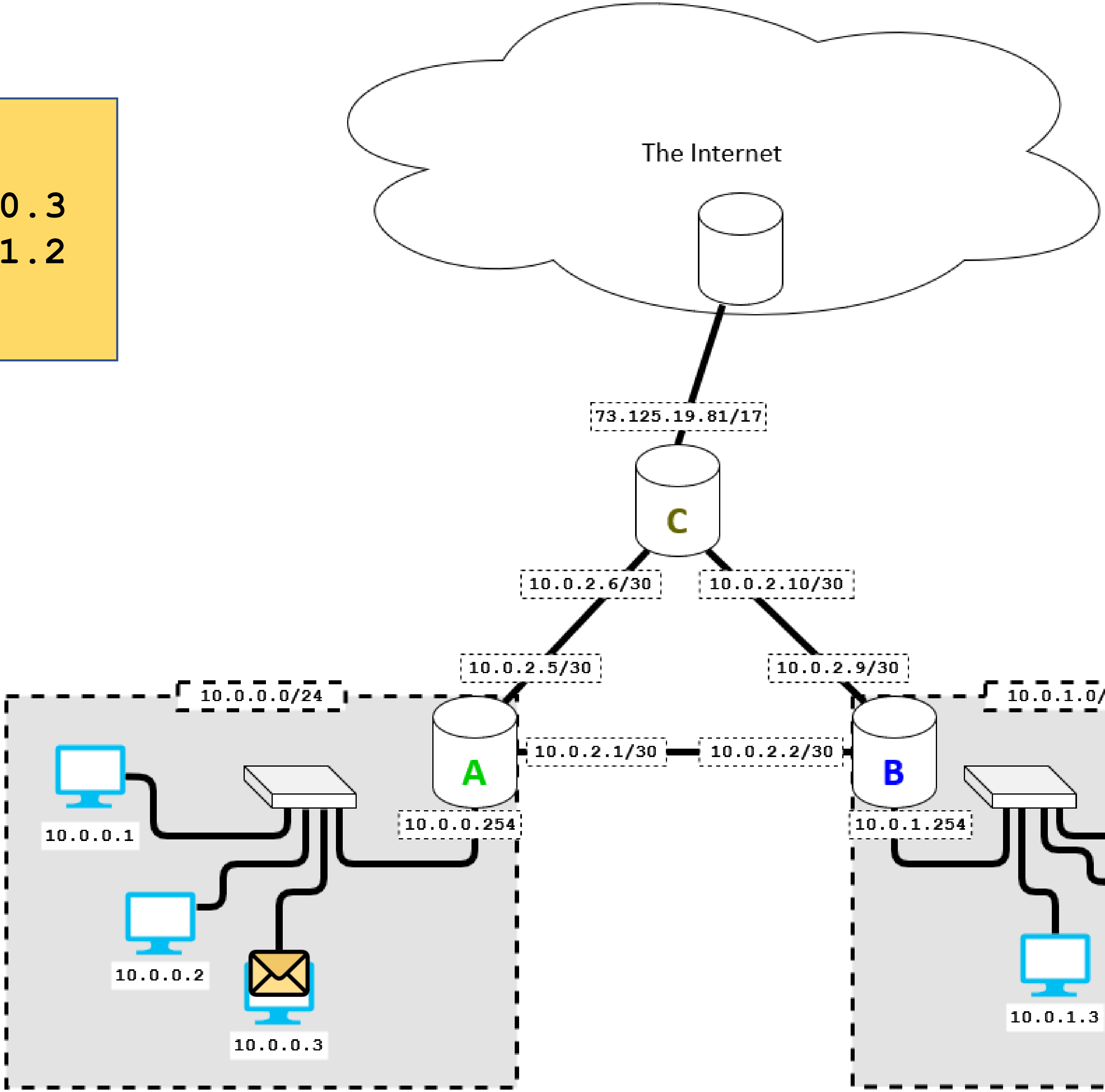


- Most host computers only have one entry in their routing table
  - Send any non-subnet data to this *router*
    - At home, this is usually your ISP modem!
  - The router will figure out where to pass the packet to

From: **10.0.0.3**
To: **10.0.1.2**



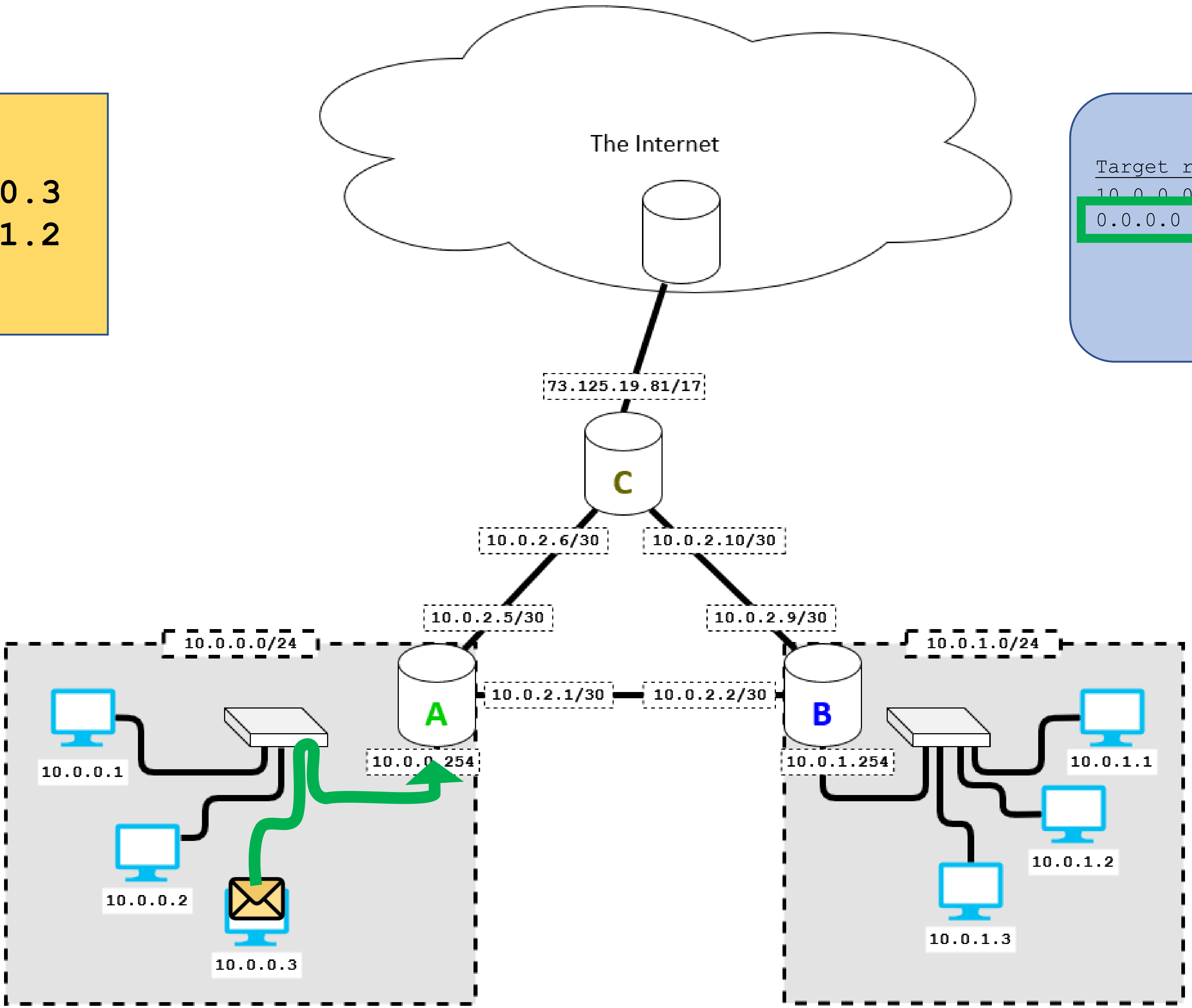
From: **10.0.0.3**
To: **10.0.1.2**



Routing Table

Target range		Next hop
10.0.0.0	/24	n/a
0.0.0.0	/ 0	10.0.0.254

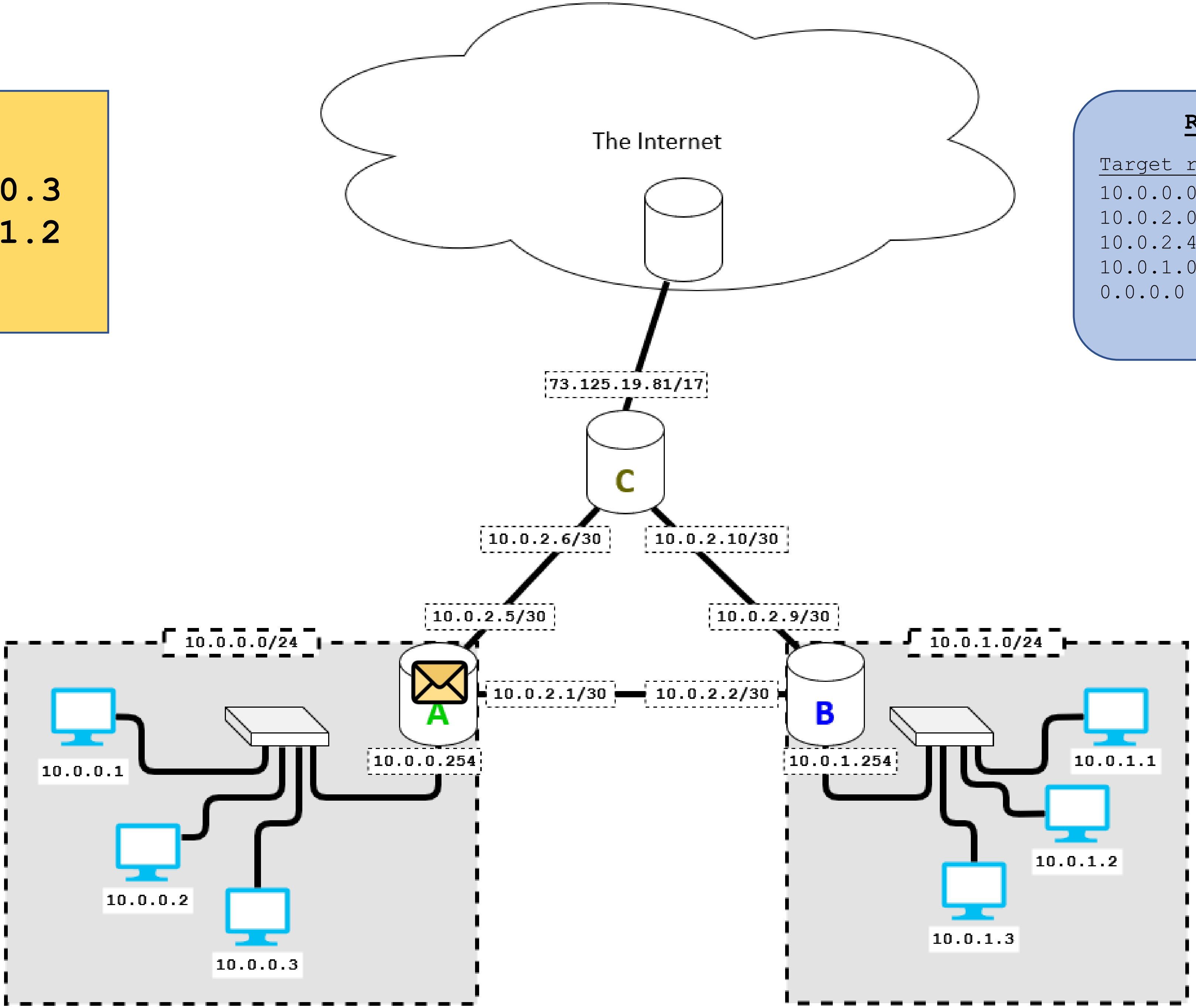
From: **10.0.0.3**
To: **10.0.1.2**



Routing Table

Target range		Next hop
10.0.0.0	/24	n/a
0.0.0.0	/ 0	10.0.0.254

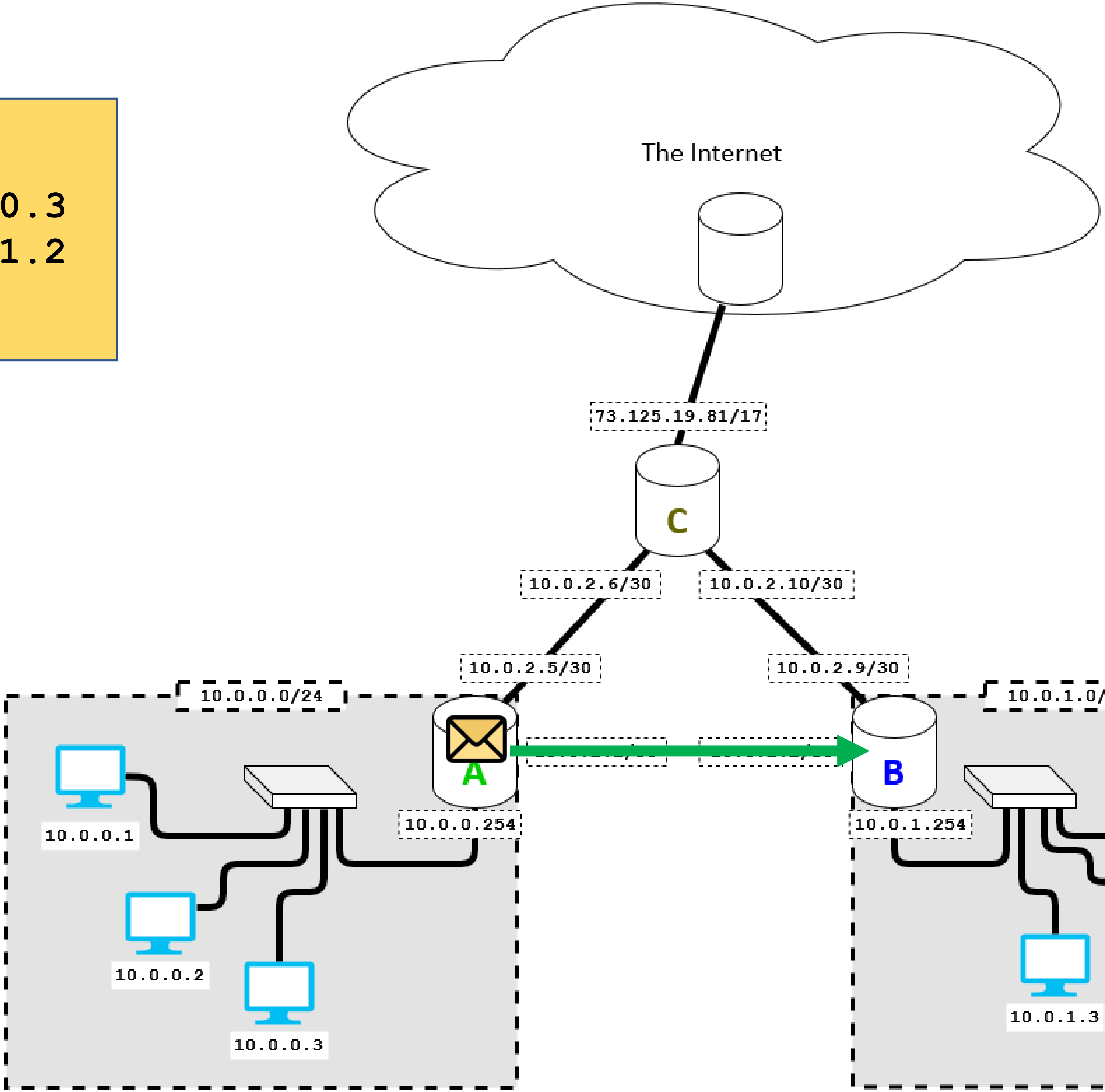
From: **10.0.0.3**
To: **10.0.1.2**



Routing Table on A

Target range		Next hop
10.0.0.0	/24	n/a
10.0.2.0	/30	n/a
10.0.2.4	/30	n/a
10.0.1.0	/24	10.0.2.2
0.0.0.0	/ 0	10.0.2.6

From: **10.0.0.3**
To: **10.0.1.2**



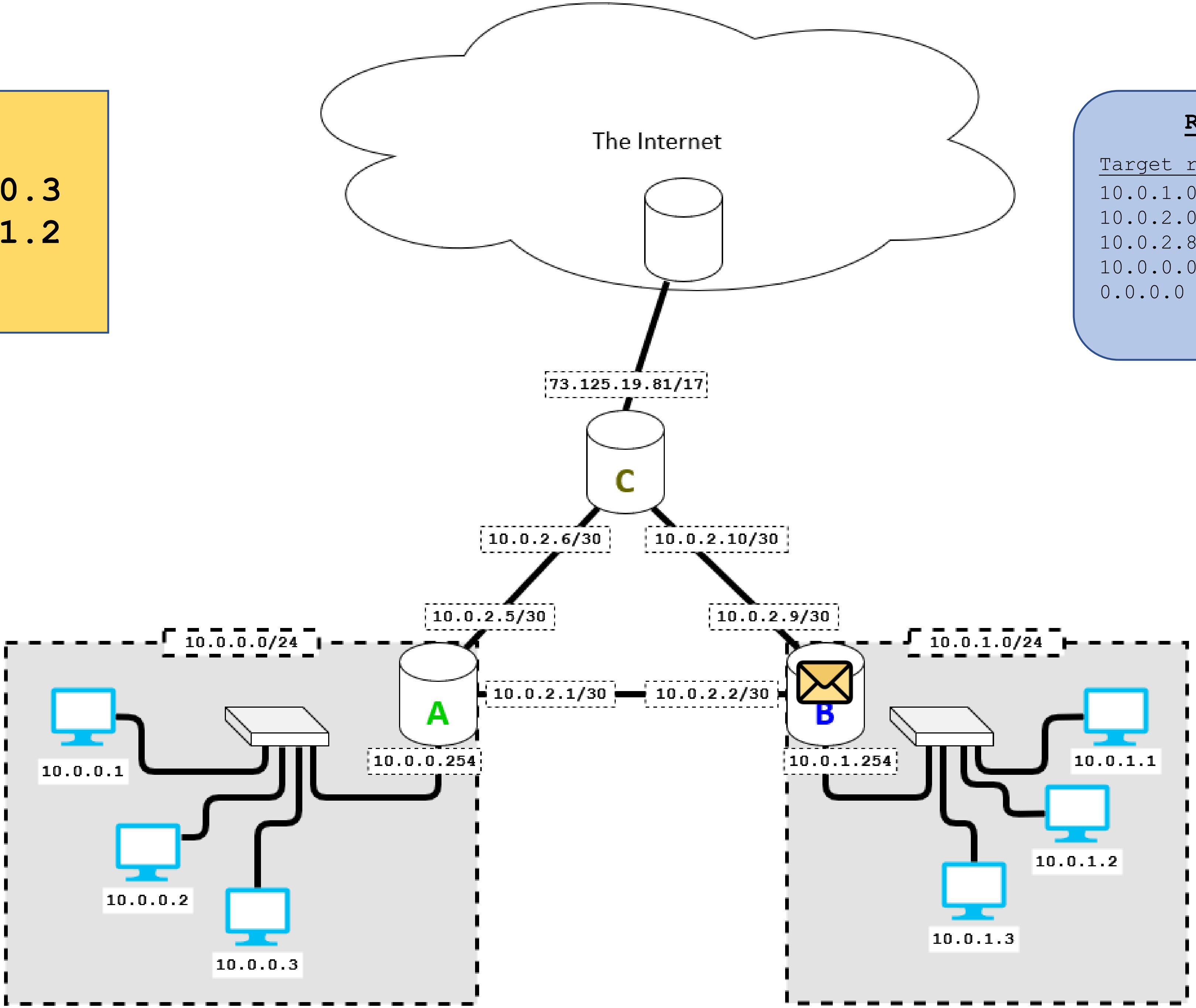
Routing Table on A

Target range		Next hop
10.0.0.0	/24	n/a
10.0.2.0	/30	n/a
10.0.2.4	/30	n/a
10.0.1.0	/24	10.0.2.2
0.0.0.0	/ 0	10.0.2.6

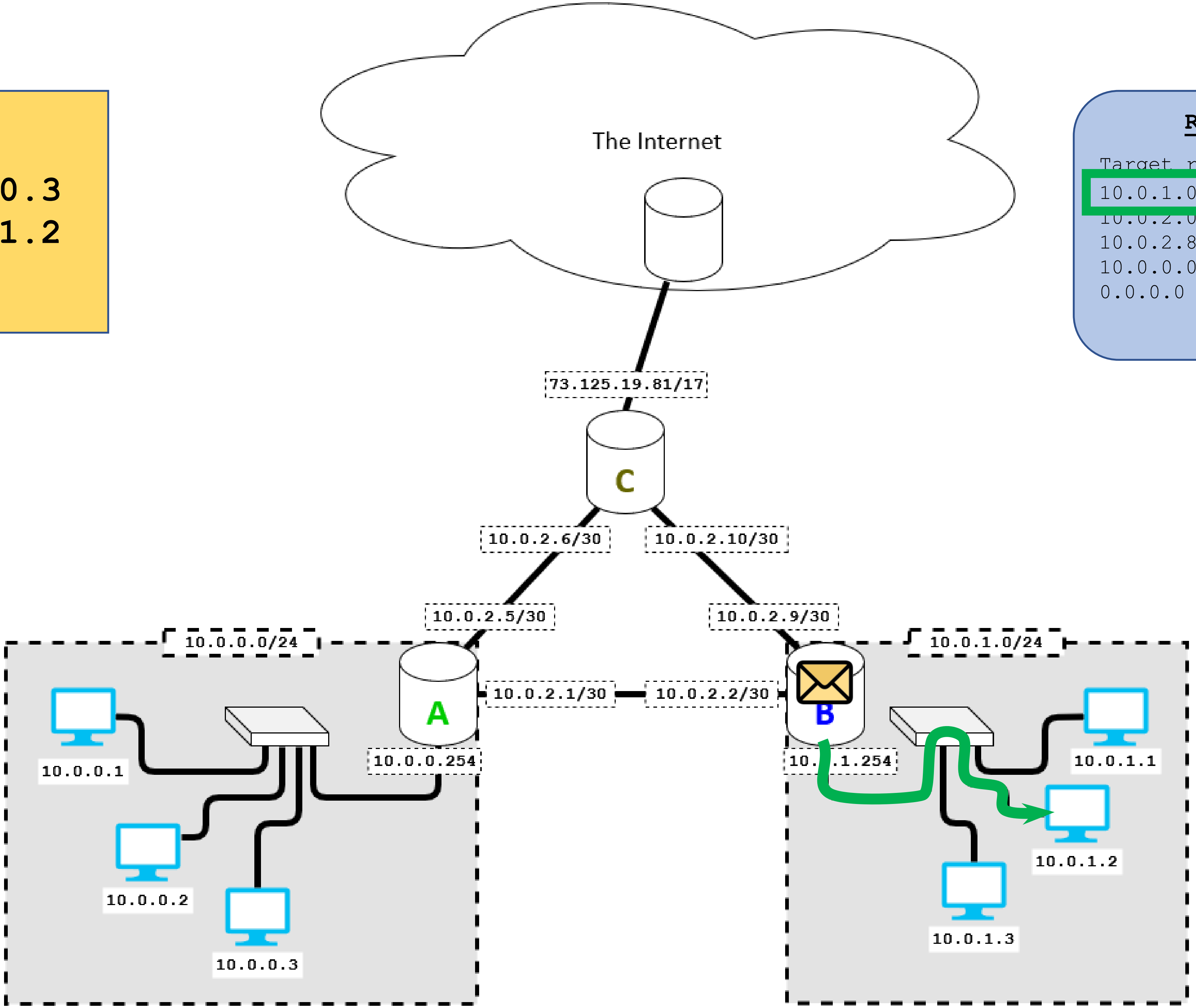
From: **10.0.0.3**
To: **10.0.1.2**

Routing Table on B

Target range		Next hop
10.0.1.0	/24	n/a
10.0.2.0	/30	n/a
10.0.2.8	/30	n/a
10.0.0.0	/24	10.0.2.1
0.0.0.0	/ 0	10.0.2.10



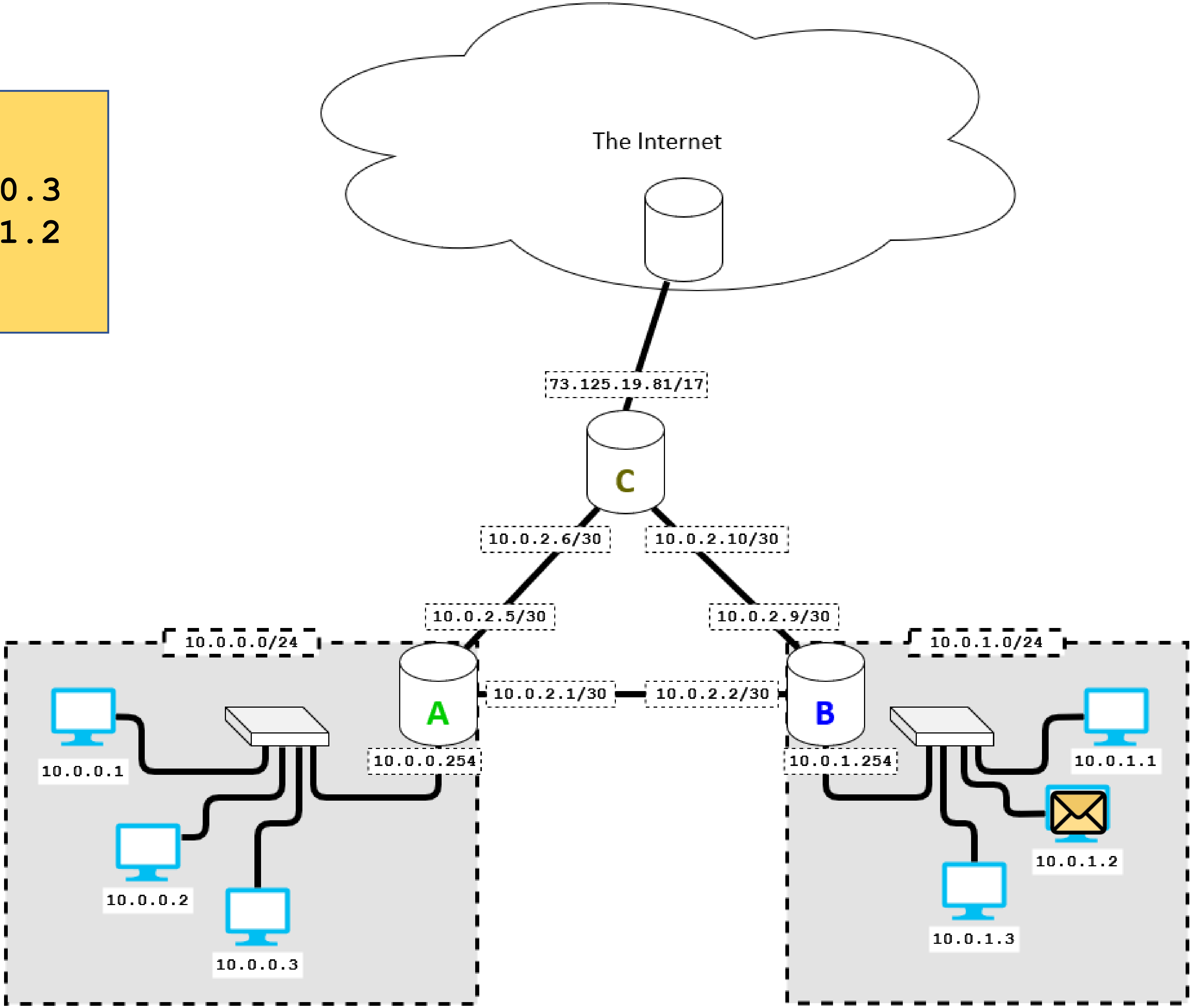
From: **10.0.0.3**
To: **10.0.1.2**



Routing Table on B

Target range		Next hop
10.0.1.0	/24	n/a
10.0.2.0	/30	n/a
10.0.2.8	/30	n/a
10.0.0.0	/24	10.0.2.1
0.0.0.0	/ 0	10.0.2.10

```
From: 10.0.0.3
To:   10.0.1.2
```



IPv4

- You can try this at home!
  - See your IP addresses:
    - **`ip addr`** or **`ifconfig`** (Linux, Mac), **`ipconfig`** (Windows)
  - See your routing table:
    - **`ip route`** or **`netstat –rn`** (Linux , Mac), **`route print`** (Windows)
  - Watch a packet over the internet:
    - **`traceroute`** (Linux , Mac), **`tracert`** (Windows)

```
Tracing route to stackoverflow.com [151.101.193.69]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  10.27.152.1
  2    <1 ms    <1 ms    <1 ms  129.27.200.161
  3     *        *        *     Request timed out.
  4     1 ms     1 ms     1 ms  graz1.aco.net [193.171.21.41]
  5     5 ms     5 ms     5 ms  aconet-ias-aconet-gw.vie.at.geant.net [83.97.88.2]
  6     6 ms    11 ms     8 ms  aconet-ias-geant-gw.vie.at.geant.net [83.97.88.1]
  7     5 ms     5 ms     5 ms  193.203.0.65
  8     5 ms     5 ms     4 ms  151.101.193.69
```

IPv4 packet overview

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Version</td><td colspan="4">Header Length</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td>8</td><td>64</td><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td>12</td><td>96</td><td colspan="32">Source IP Address</td></tr>
<tr><td>16</td><td>128</td><td colspan="32">Destination IP Address</td></tr>
</table>

- Version: always **`0100`** (version 4)
- Twin “Length” fields
  - Length of just the header
    - Optional header extensions may make it longer!
  - Length of this packet

IPv4 packet overview

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0																																
4	32																																
8	64	Time To Live																Header Checksum															
12	96																																
16	128																																

- Safeguards
  - *Header Checksum* protects header integrity
    - guards against header corruption on lower layer
  - *Time To Live* limits how far a packet can travel
    - after 256 hops, the packet is dropped
    - guards against routing issues (loops etc.)

IPv4 packet overview

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Header Length				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset												
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															

- *Fragmentation* happens if a packet is too large for a given connection
  - Packet is split into two or more packets
  - Recipient re-assembles the fragments
- Fragments are routed as separate packets
  - Might take different routes, arrive out-of-order, etc.

IPv4 packet overview

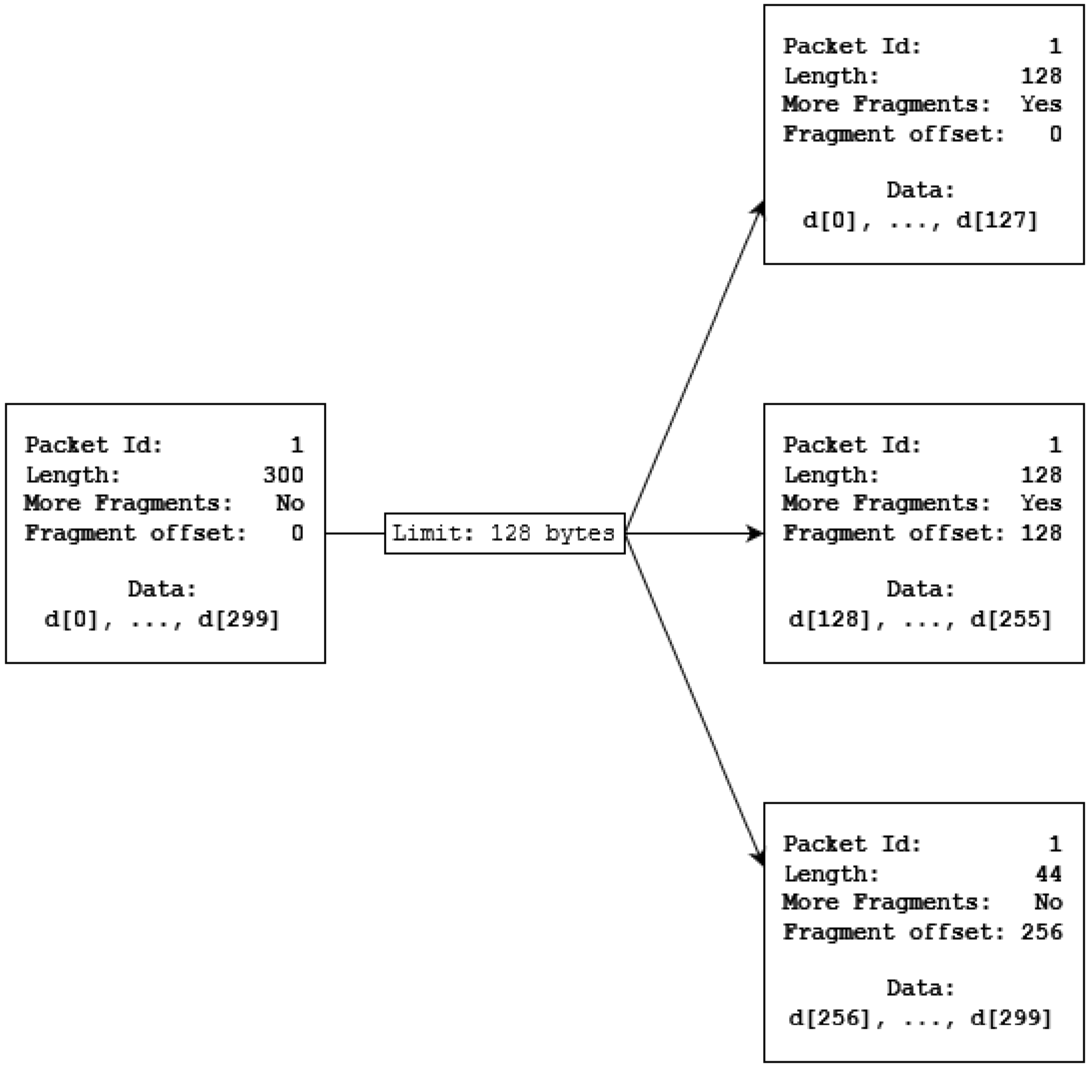
<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Version</td><td colspan="4">Header Length</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td>8</td><td>64</td><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td>12</td><td>96</td><td colspan="32">Source IP Address</td></tr>
<tr><td>16</td><td>128</td><td colspan="32">Destination IP Address</td></tr>
</table>

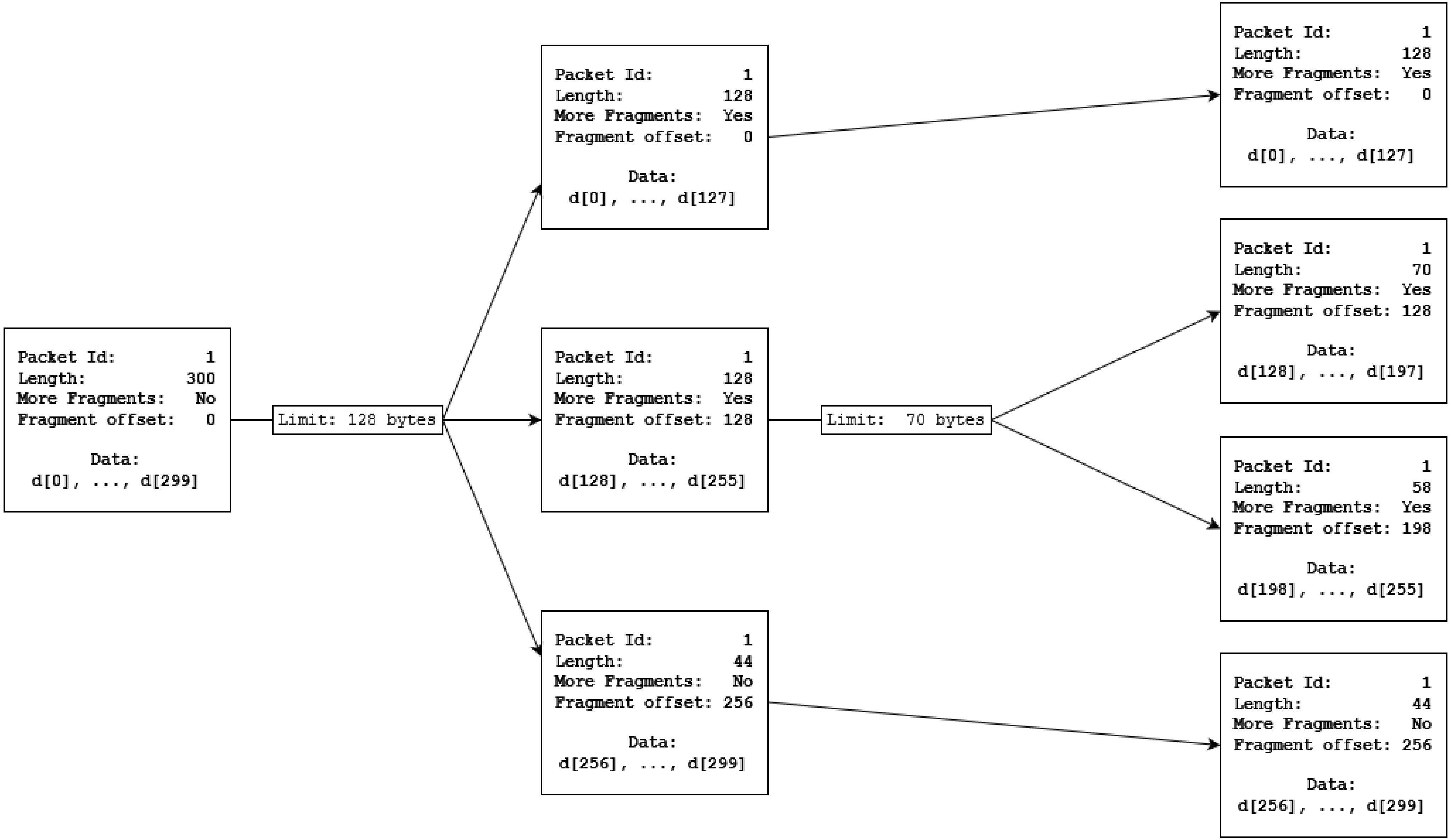
- *Identification* is the same across all fragments
- *Flags*: whether this is *not* the last packet (***M**ore **F**ragments* flag)
- *Fragment offset*: this fragment's position within the original message

IPv4 fragmentation

```
Packet Id:           1
Length:            300
More Fragments:     No
Fragment offset:     0

       Data:
 d[0], ..., d[299]
```


Packet Id: 1
Length: 300
More Fragments: No
Fragment offset: 0
Data:
d[0], ..., d[299]
Limit: 128 bytes
Packet Id: 1
Length: 128
More Fragments: Yes
Fragment offset: 0
Data:
d[0], ..., d[127]
Packet Id: 1
Length: 128
More Fragments: Yes
Fragment offset: 128
Data:
d[128], ..., d[255]
Packet Id: 1
Length: 44
More Fragments: No
Fragment offset: 256
Data:
d[256], ..., d[299]


Packet Id: 1
Length: 300
More Fragments: No
Fragment offset: 0
Data:
d[0], ..., d[299]
Limit: 128 bytes
Packet Id: 1
Length: 128
More Fragments: Yes
Fragment offset: 0
Data:
d[0], ..., d[127]
Packet Id: 1
Length: 128
More Fragments: Yes
Fragment offset: 128
Data:
d[128], ..., d[255]
Packet Id: 1
Length: 44
More Fragments: No
Fragment offset: 256
Data:
d[256], ..., d[299]
Limit: 70 bytes
Packet Id: 1
Length: 128
More Fragments: Yes
Fragment offset: 0
Data:
d[0], ..., d[127]
Packet Id: 1
Length: 70
More Fragments: Yes
Fragment offset: 128
Data:
d[128], ..., d[197]
Packet Id: 1
Length: 58
More Fragments: Yes
Fragment offset: 198
Data:
d[198], ..., d[255]
Packet Id: 1
Length: 44
More Fragments: No
Fragment offset: 256
Data:
d[256], ..., d[299]

IPv4 fragmentation – Issues

- 16-bit packet ID is insufficient for high transmission rates
  - 16 bit packet ID ≙ 65536 packets “in flight”
  - No acknowledgments ⇒ ID can’t be reused until TTL expires
  - 65536 packets ÷ 128 seconds = 512 packets per second

- Also: other issues
  - We’ll talk details later
  - (We need to understand transport layer concepts first 😉)

Further reading: https://datatracker.ietf.org/doc/html/rfc8900

IPv4 fragmentation – Alternatives

- Path MTU discovery
  - Detect the largest packet size that can be sent unfragmented

- How: it’s complicated
  - **D**on’t **F**ragment flag in IP header + trial & error
    - Problem: failure notifications might not arrive
  - More sophisticated trial & error at higher layers
    - Problem: need to re-invent this wheel for every transport layer protocol
    - Not every transport layer protocol is able to fragment data!

Further reading: https://datatracker.ietf.org/doc/html/rfc8900