Computer Organization and Networks

Chapter 8: Networking II

Winter 2021/2022

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

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																															

- 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	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																															

- 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

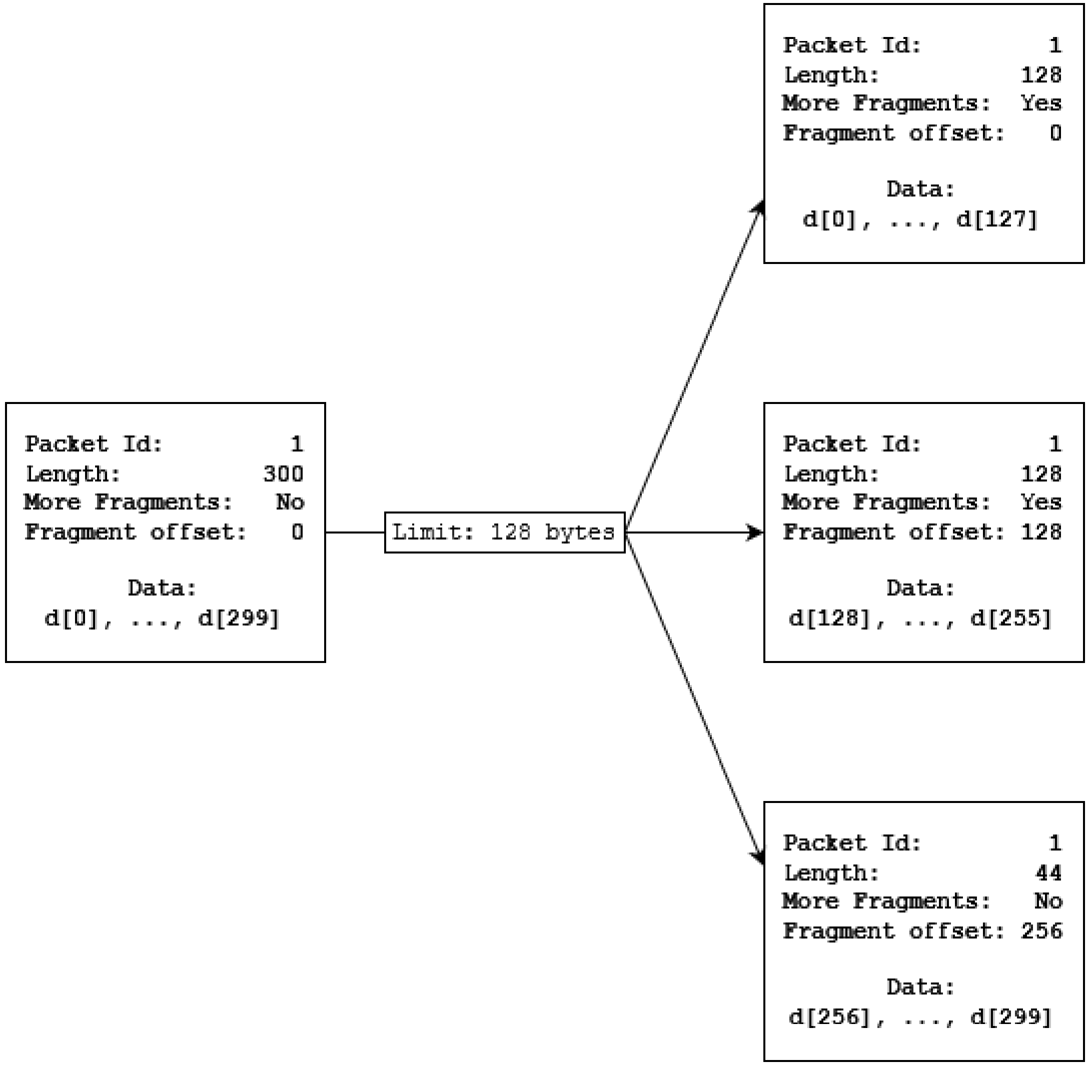
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Header Length				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset												
8	64	Time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

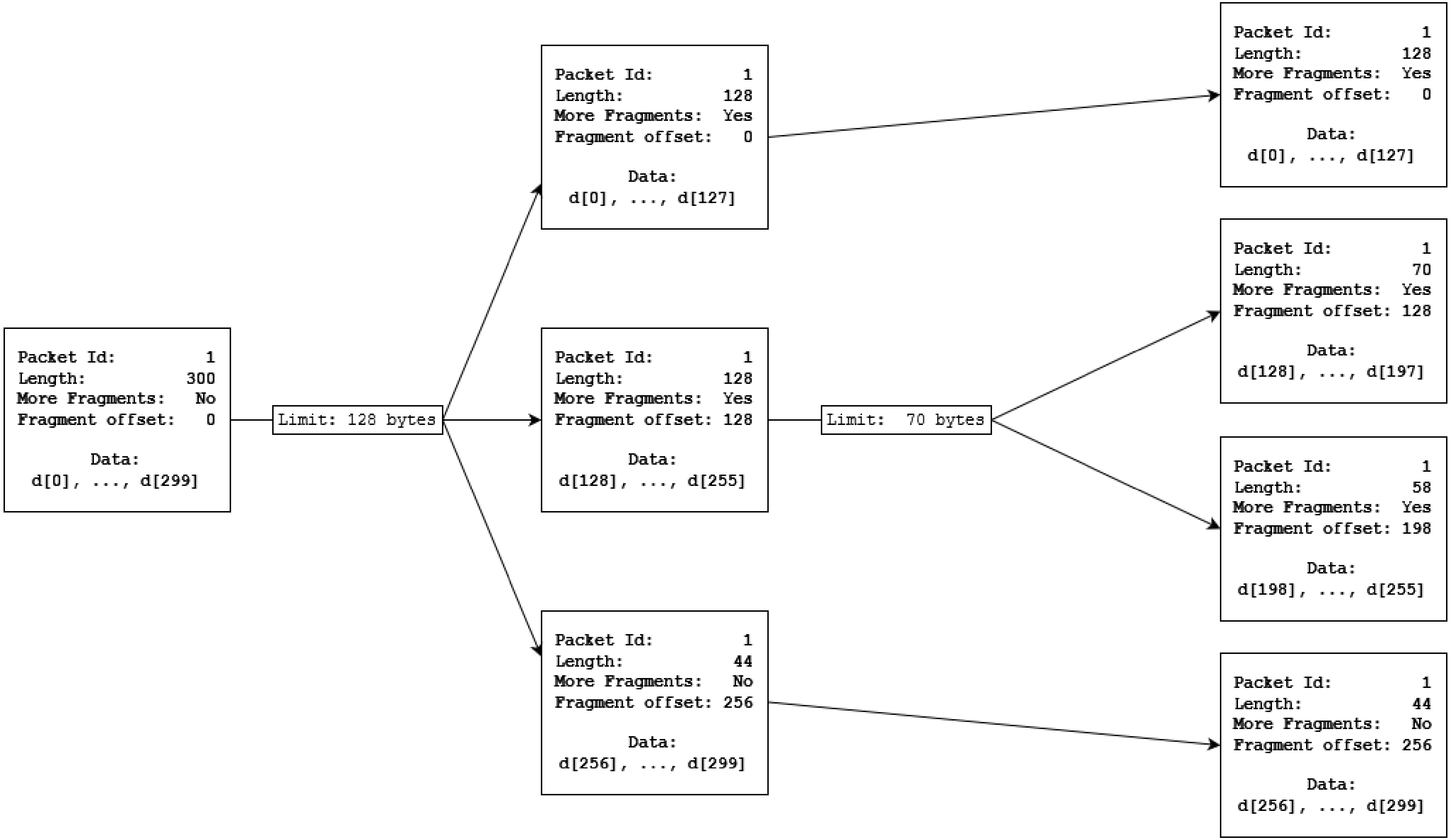
- *Identification* is the same across all fragments
- *Flags*: whether this is <u>*not*</u> the last packet (***M****ore* ***F****ragments* flag)
- *Fragment offset*: this fragment's position within the original message

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 address space

- IPv4 addresses are 32 bits long
  - How many different IP addresses can exist?

IPv4 address exhaustion

- IPv4 addresses are 32 bits long
  - There can be at most 2^{32} different IPv4 addresses
  - 2^{32} = 4 billion, 294 million, 967 thousand, two hundred and ninety-six
  - Global population ≈ 7.9 billion (September 2021)
- How many devices do you own that use IPv4?
  - Your home PC
  - Your phone
  - Your ISP router (twice!)
  - Laptops? Game consoles? Cars? Fridges? Doorbells?

IPv4 address exhaustion

https://www.arin.net/vault/announcements/2015/20150924.html

ARIN IPv4 Free Pool Reaches Zero

Posted: Thursday, 24 September 2015

On 24 September 2015, ARIN issued the final IPv4 addresses in its free pool. ARIN will

https://www.ripe.net/publications/news/about-ripe-ncc-and-ripe/the-ripe-ncc-has-run-out-of-ipv4-addresses

The RIPE NCC has run out of IPv4 Addresses

Today, at 15:35 (UTC+1) on 25 November 2019, we made our final /22 IPv4 allocation from the last remaining addresses in our available pool. We have now run out of IPv4 addresses.

https://www.lacnic.net/4848/2/lacnic/ipv4-exhaustion:-lacnic-has-assigned-the-last-remaining-address-block

IPv4 Exhaustion: LACNIC Has Assigned the Last Remaining Address Block

19 August 2020

The Latin American and Caribbean Internet Address Registry (LACNIC) announces that the last available IPv4 address block has been reserved.

IPv4 address exhaustion

- The internet is out of IPv4 addresses…
- Somehow, your new phone still works?

- There are ways around address exhaustion
  - We'll talk about this later!

IPv6

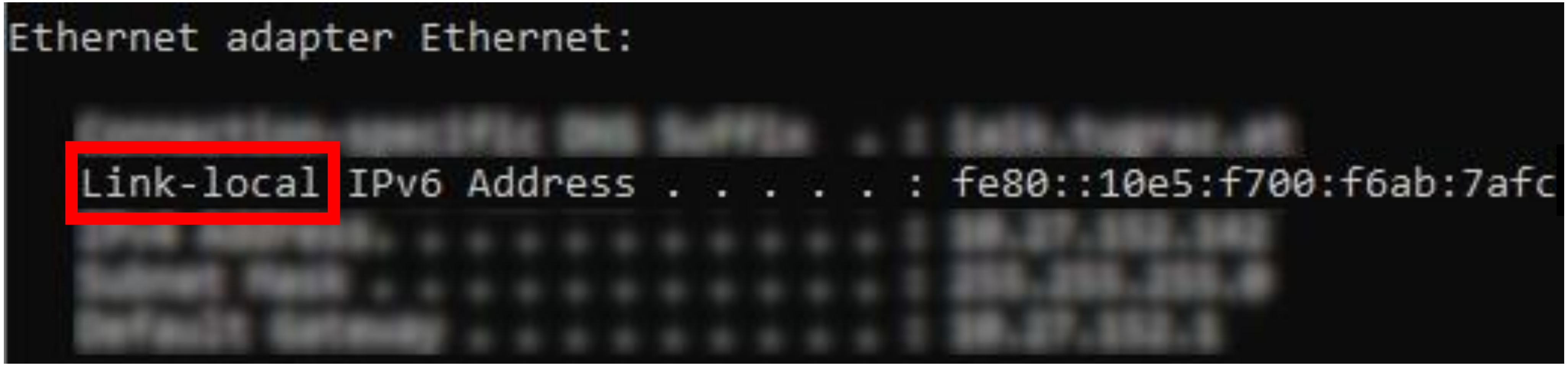
- **I**nternet **P**rotocol, **v**ersion **6**
- Successor to IPv4

- Not natively interoperable with IPv4
  - IPv4-only devices cannot communicate with IPv6-only devices
  - Most modern devices implement *both* IPv4 and IPv6
  - Eventually, IPv4 will be phased out…

IPv6 addressing

- 128-bit address
  - Notation: 16-bit hexadecimal blocks separated by colons (`:`)
  - Zero blocks can be omitted using double colon (`::`)
  - **`fe80::10e5:f700:f6ab:7afc`** is the same as
    `fe 80 00 00 00 00 00 00 10 e5 f7 00 f6 ab 7a fc`

IPv6 addressing



- 64-bit *network prefix*, 64-bit *interface identifier*
- A single interface (e.g.: a network card) may have multiple addresses
  - Addresses share the *interface identifier*
- Addresses have a *scope* in which they are valid

IPv6 scoping

- Global addresses
  - Valid in any network connected to the internet
  - May be routed on the public internet
- Unique-local addresses (in **`fc00::/7`**)
  - Only valid within “the local network(s)”
    - Example: valid within all the classroom networks of a school
  - Routed between those networks, but *not* on the public internet
- Link-local addresses (in **`fe80::/64`**)
  - Only valid within the Link Layer network

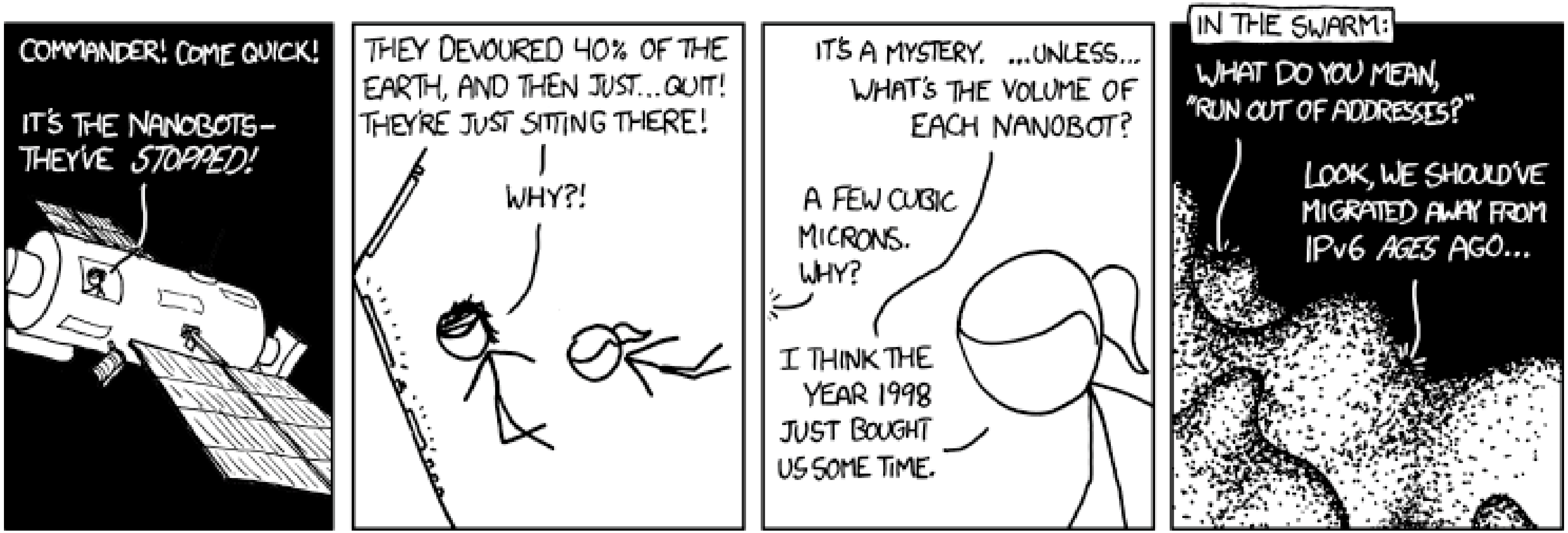
IPv6 packet overview

<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><th>0</th><th>0</th><td colspan="4">Version</td><td colspan="8">Traffic class</td><td colspan="20">Flow label</td></tr>
<tr><th>4</th><th>32</th><td colspan="16">Payload length</td><td colspan="8">Next header</td><td colspan="8">Hop limit</td></tr>
<tr><th>8</th><th>64</th><td colspan="32" rowspan="4">Source address</td></tr>
<tr><th>12</th><th>96</th></tr>
<tr><th>16</th><th>128</th></tr>
<tr><th>20</th><th>160</th></tr>
<tr><th>24</th><th>192</th><td colspan="32" rowspan="4">Destination address</td></tr>
<tr><th>28</th><th>224</th></tr>
<tr><th>32</th><th>256</th></tr>
<tr><th>36</th><th>288</th></tr>
</table>

- Similar fields to IPv4 packets
  - Version is always `0110` (version 6)
  - Length, Source and Destination fields
  - Optional extension header blocks
- Header checksum removed
  - Relies on Link Layer to provide error detection
- Fragmentation (mostly) removed
  - No fragmentation by routers
  - Fragmentation by hosts only as an extension
    - Application Layer is expected to perform fragmentation

IPv6 recap

- Successor to IPv4
  - “Permanent” solution to IP address exhaustion
    - We’ll talk about IPv4 workarounds in a bit!
  - Some protocol-level improvements
  - Not interoperable with IPv4
- Supported by most modern end-user devices
  - Server-side support is… still lacking [https://ipv6.watch]
- 128-bit addresses (64-bit network part, 64-bit interface identifier)
  - 2^{64} networks, each consisting of 2^{64} hosts



xkcd #865 “Nanobots” | Randall Munroe | *https://xkcd.com/*

Image: own work, derived from Public Domain

The Transport Layer

The Transport Layer

- Computers A and B are capable of sending data to each other
- **Goal:** Allow multiple applications to communicate reliably

- Concerns:
  - How to distinguish which application data is meant for? (multiplexing)
  - What if data is lost on the lower layers? (reliability)
  - How much data can the network handle? (congestion control)
  - How much data can the receiver handle? (flow control)

The Transport Layer

- The internet has two widely-used protocols at the Transport Layer:
  - **T**ransmission **C**ontrol **P**rotocol
    - Focused on reliable delivery
    - Connection-based
  - **U**ser **D**atagram **P**rotocol
    - Focused on speed
    - Connectionless

The Transport Layer: Ports

- Concept used for both TCP and UDP
- Source and destination identified by *port number*
  - 16 bits (65536 available ports)
  - TCP and UDP ports are *separate*
    - The protocols implement the same idea, but each only cares about its own ports…
- Common notation: Port number after IP address
  - **`127.0.0.1:8000`** is port 8000 at host 127.0.0.1
  - **`[::1]:8000`** is port 8000 at host ::1

UDP

- Fire-and-forget transmission
  - Real-time applications
- Data may never arrive, may arrive out of order, …
  - Data loss must be tolerable for the upper-layer application
- Extremely simple and straightforward

UDP datagram header

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	Source port																Destination port															
4	32	Length																Checksum															

TCP

- Highly reliable transmission of a byte stream
  - Acknowledgments and re-transmission
  - Guaranteed to maintain data ordering
- Non-trivial protocol overhead
  - Still better than re-inventing the wheel if you need it!

TCP

- TCP connections have two sides: *server* and *client*

- *Server* listens on a specific port
  - Server port is fixed for all connections
- *Client* connects to that port on the server
  - Client uses a “random” ephemeral port, different for each connection
  - See for yourself: **`netstat –onb`** (Win) or **`netstat –tnap`** (Linux, Mac)

- Connections are uniquely identified by *client IP* + *client port*

The Transport Layer: Ports 2

- Two applications can’t use the same port number
- Client needs to know which port number to connect to

- Port numbers are standardized by IANA
  - 0–1023: *well-known* ports
    - Examples: 22 (SSH), 80 (HTTP), 123 (NTP), 194 (IRC), 443 (HTTPS), …
  - 1024–49151: *registered* ports
    - Most server applications will use this range (even unregistered ones…)
  - 49152–65535: *dynamic* ports
    - Most OS will use this range for ephemeral (client) ports

TCP packet overview

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	[illegible]																															
8	64	[illegible]																															
12	96	[illegible]																															
16	128	Checksum																[illegible]															

- Source + destination ports allow identification of connection
- Checksum over entire header + data

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			[illegible]	[illegible]	[illegible]	[illegible]	ACK	[illegible]	[illegible]	[illegible]	[illegible]	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															

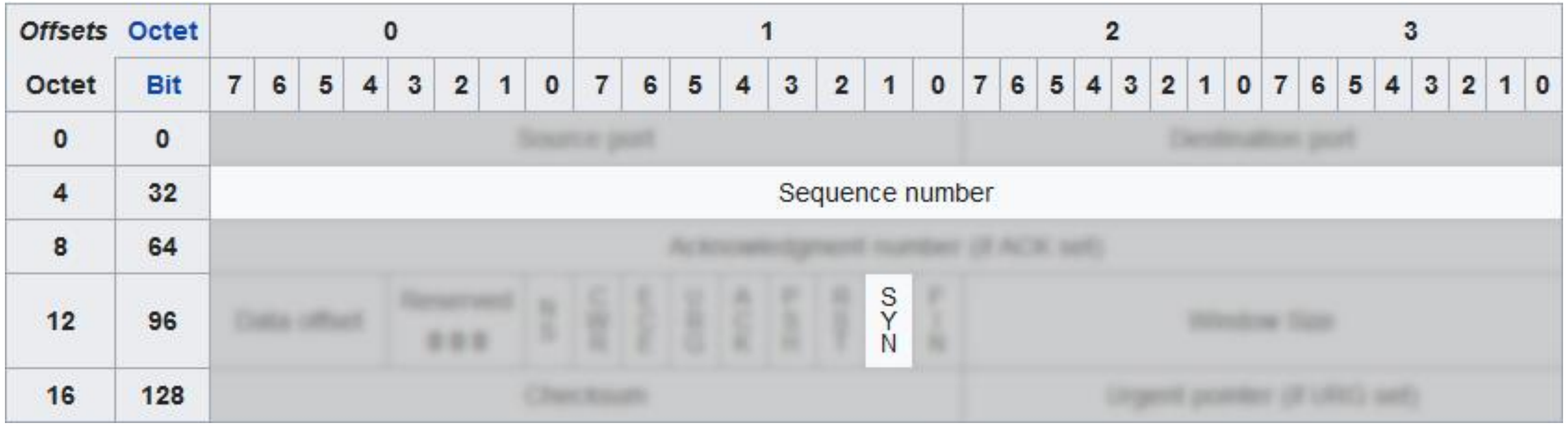
- TCP maintains a *sequence number* across the entire connection
  - Separate number for each end's packets
- Receipt of contiguous data confirmed via *acknowledgment number*
  - Acknowledgement number := next expected sequence number
- This allows ordering of data and re-sending of lost packets!

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	[illegible]																[illegible]															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	[illegible]				[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	ACK	[illegible]	[illegible]	SYN	[illegible]	[illegible]															
16	128	[illegible]																[illegible]															

- Connection establishment: Three-way handshake
  - Client -> Server: **`SYN`**
  - Server -> Client: **`SYN + ACK`**
  - Client -> Server: **`ACK`**

TCP data ordering



- Client -> Server: **`SYN`**
  - Sequence number: **`seq_c`**, chosen randomly

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	[illegible]																[illegible]															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	[illegible]				[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	ACK	[illegible]	[illegible]	SYN	[illegible]	[illegible]															
16	128	[illegible]																[illegible]															

- Server -> Client: **`SYN + ACK`**
  - Sequence number: **`seq_s`**, chosen randomly
  - Acknowledgement: **`seq_c+1`**

TCP data ordering

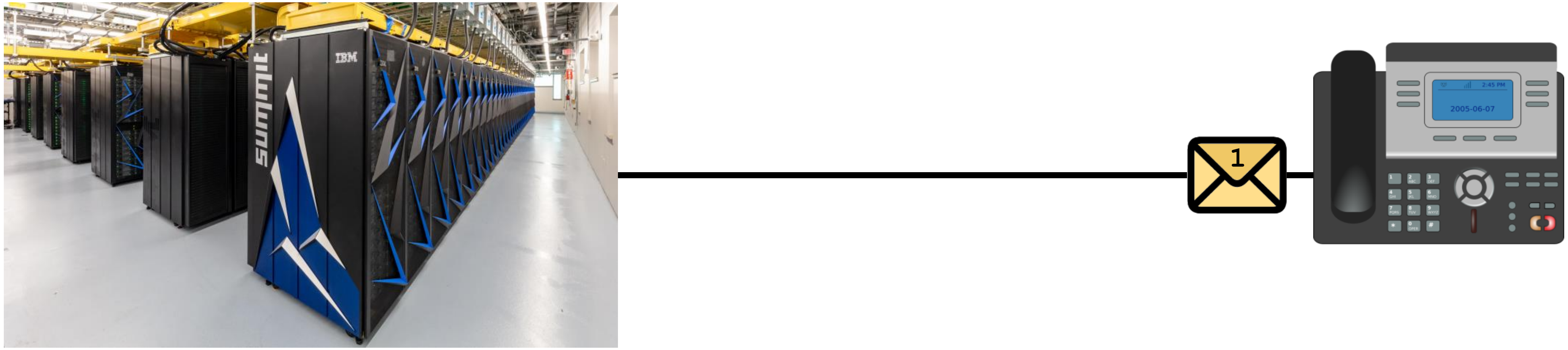
Offsets	Octet	0	1	2	3
Octet	Bit	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0	0				
4	32	Sequence number			
8	64	Acknowledgment number (if ACK set)			
12	96		ACK		
16	128				

- Client -> Server: **`ACK`**
  - Sequence number: **`seq_c+1`**
  - Acknowledgement: **`seq_s+1`**

- Now both sides know that the other side has their sequence number
  - Ready to communicate!

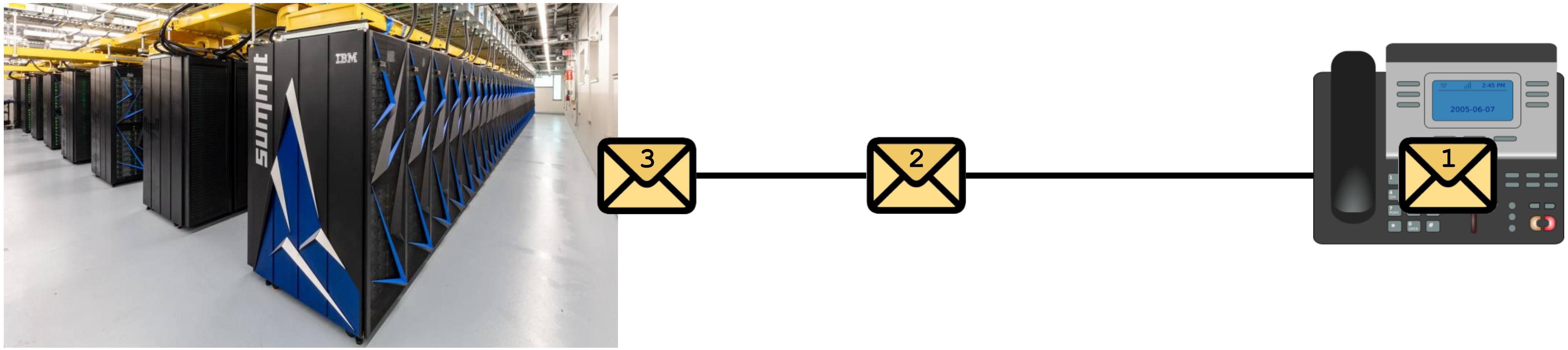
TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



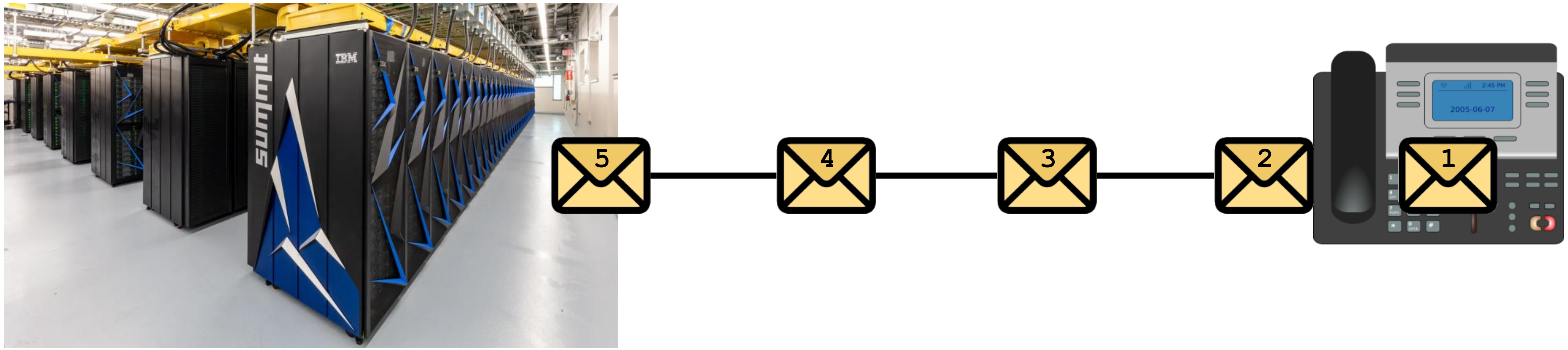
TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



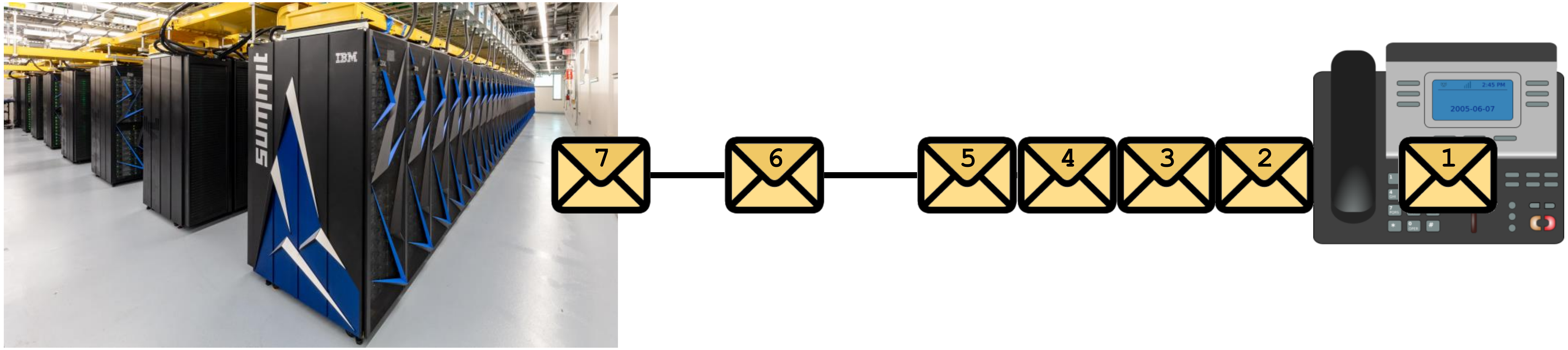
TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



- The desk phone doesn’t stand a chance to keep up!

TCP flow control

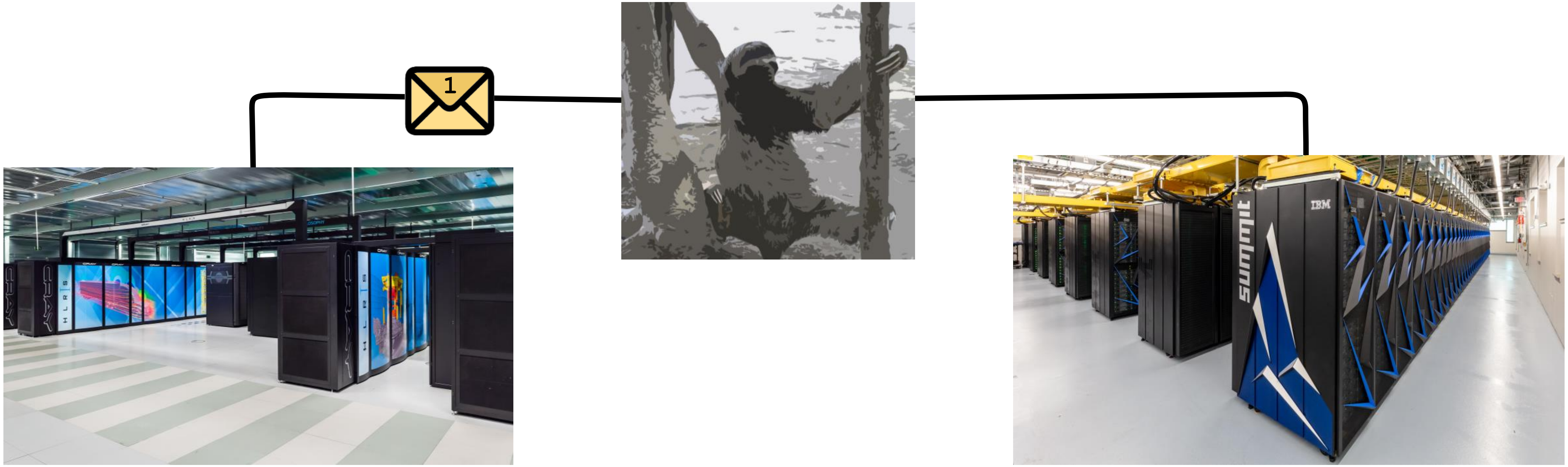
<table>
<tr><td>Offsets</td><td>Octet</td><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td>Octet</td><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td colspan="16"></td><td colspan="16"></td></tr>
<tr><td>4</td><td>32</td><td colspan="32"></td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4"></td><td colspan="3"></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16"></td><td colspan="16"></td></tr>
</table>

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link

- *Window size* indicates how much more data the host can handle
- The other end must throttle its transmission rate to accommodate
  - Window size is relative to the last ACK'd packet

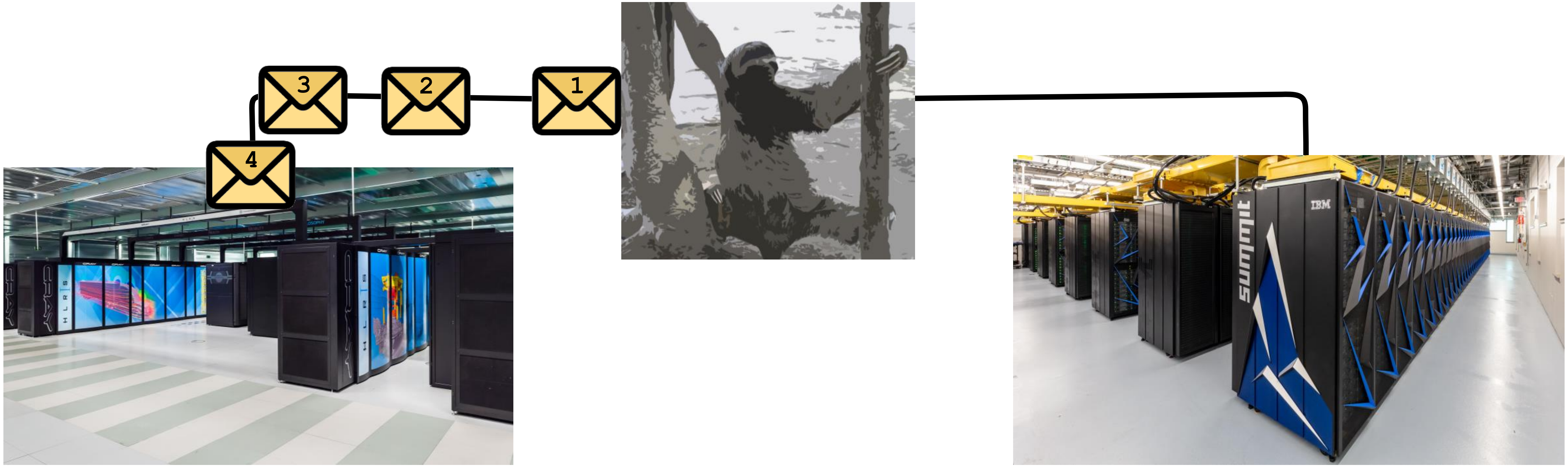
TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
  - Keep in mind: the "dial-up connection" could be some intermediate network



TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
  - Keep in mind: the “dial-up connection” could be some intermediate network

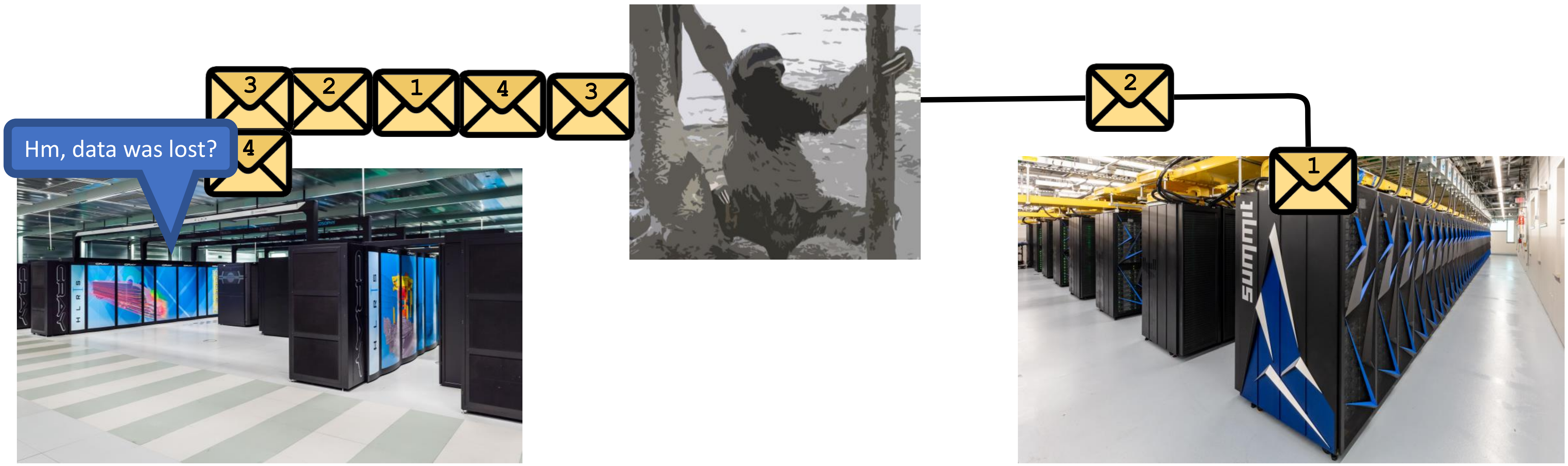


TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
  - Keep in mind: the “dial-up connection” could be some intermediate network

TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
  - Keep in mind: the “dial-up connection” could be some intermediate network



TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
  - Keep in mind: the “dial-up connection” could be some intermediate network
- If you just keep shoving data…
  - … it will get stuck in a queue somewhere …
    - ... so you think it was lost and send it again …
      - … now your queue is twice the size …
        - … and nothing useful gets done.
- How do we avoid that?

TCP congestion control

- Each side throttles its data transmission rate independently
  - No cooperation required
  - Different OS have different algorithms
- Basic concept:
  - Start at a relatively slow rate, then increase speed until data gets lost
  - Once data is lost, assume we overloaded the connection and slow down again
- Details differ from OS to OS

Transport Layer recap

- Two main protocols: TCP and UDP
  - TCP: highly reliable, but comes with overhead
  - UDP: low overhead, but no reliability guarantees
- *Port numbers* identify target application
  - By convention, low port numbers (0–1023) are reserved for specific services
  - 1024–49151 are used by other servers
  - 49152–65535 are used for ephemeral ports

TCP recap

- *Client* establishes connection to *Server*
  - Server lists on a pre-agreed port
  - Client uses a “random” port (49152–65535)
- *Sequence numbers* and *acknowledgement numbers*
  - Client and server have separate counters
  - Acknowledgement of received data using the other side’s counter
  - Re-ordering and re-sending if necessary

TCP recap

- ***Flow Control*** protects the ***recipient***
  - Recipient advertises its capacity
  - Sender has to abide by it
- ***Congestion Control*** protects the ***network***
  - Transmission rate is gradually increased
    - Throttled back if packet loss is detected
  - Each side handles this independently
  - Details differ from OS to OS

The Network Layer

(again?)

Recap: IPv4 address exhaustion

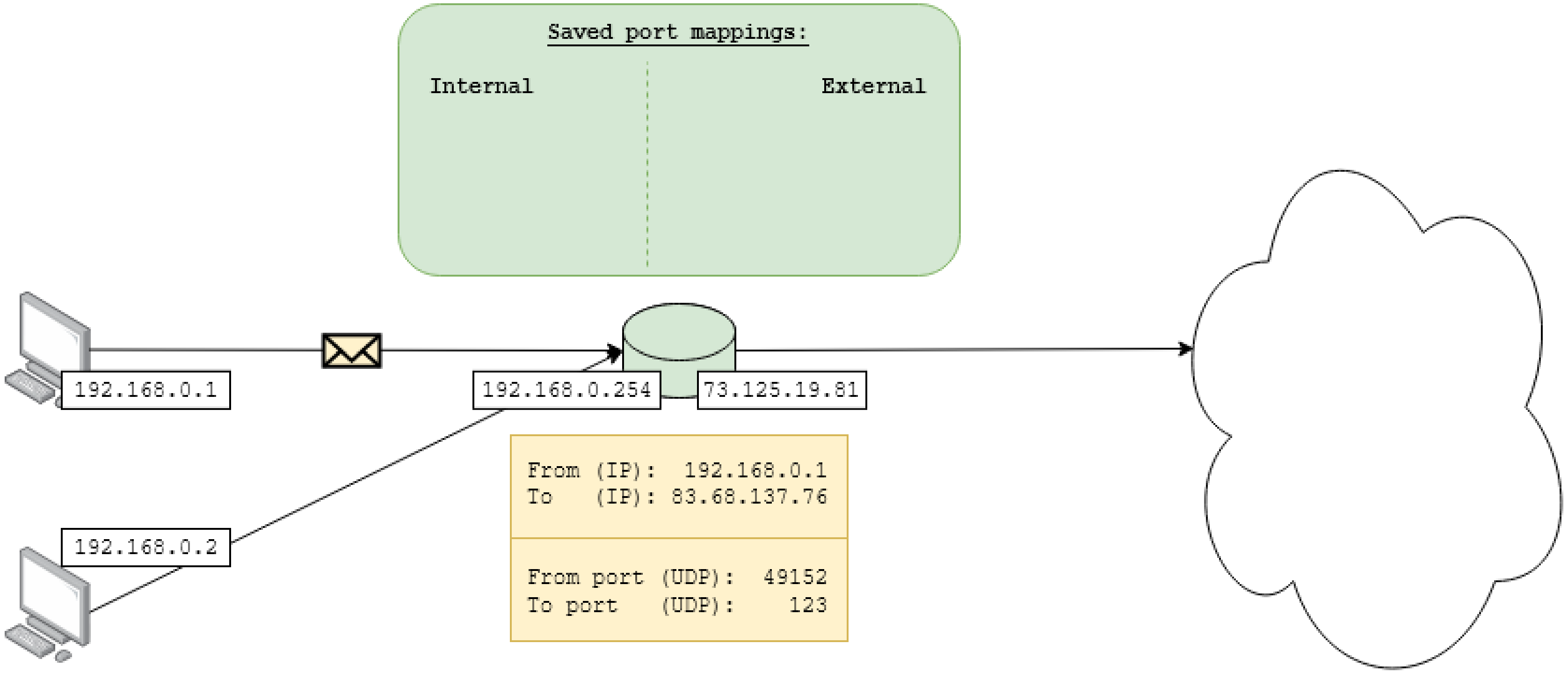
- IPv4 addresses are 32 bits long
  - 2^{32} is about 4 billion
- Every Internet-enabled device needs an address to communicate
  - There are a lot of devices
- The internet is (mostly) out of IPv4 addresses!

Port Address Translation

- “Hide” an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
- Also known as:
  - “PAT”
  - Network Address Translation (“NAT”)
  - Network Address and Port Translation (“NAPT”)
  - NAT overloading
  - IP masquerading

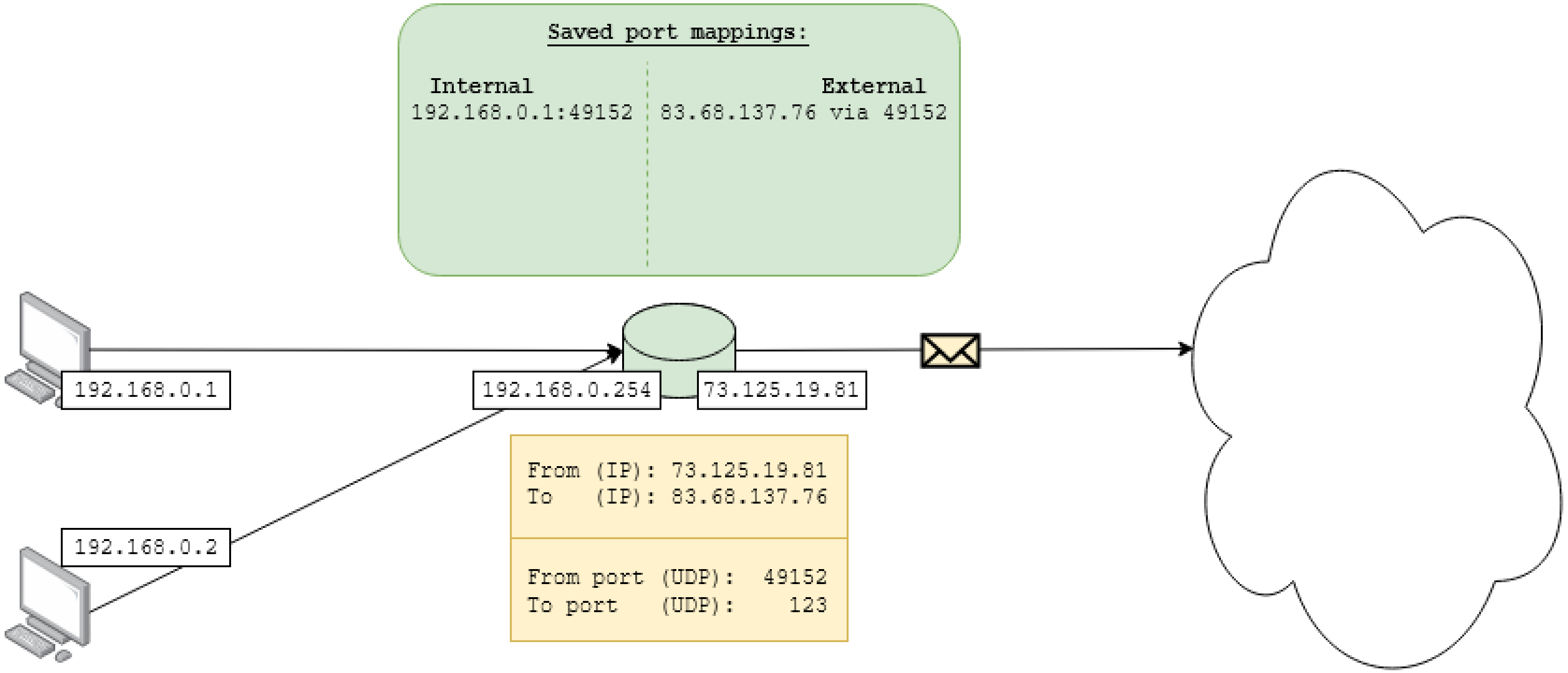
Port Address Translation

- “Hide” an entire private network behind a single public IP
  - Rewrite IP packets at the boundary



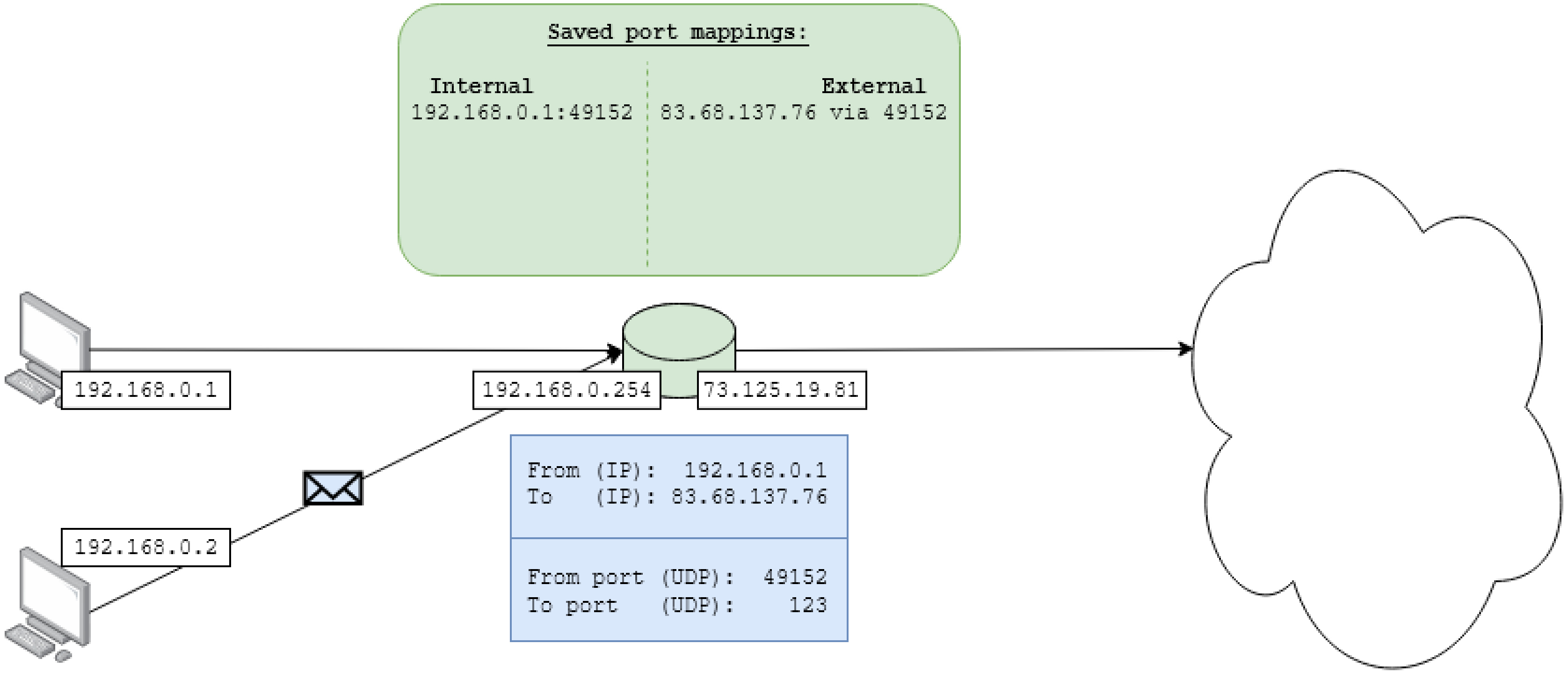
Port Address Translation

- "Hide" an entire private network behind a single public IP
  - Rewrite IP packets at the boundary



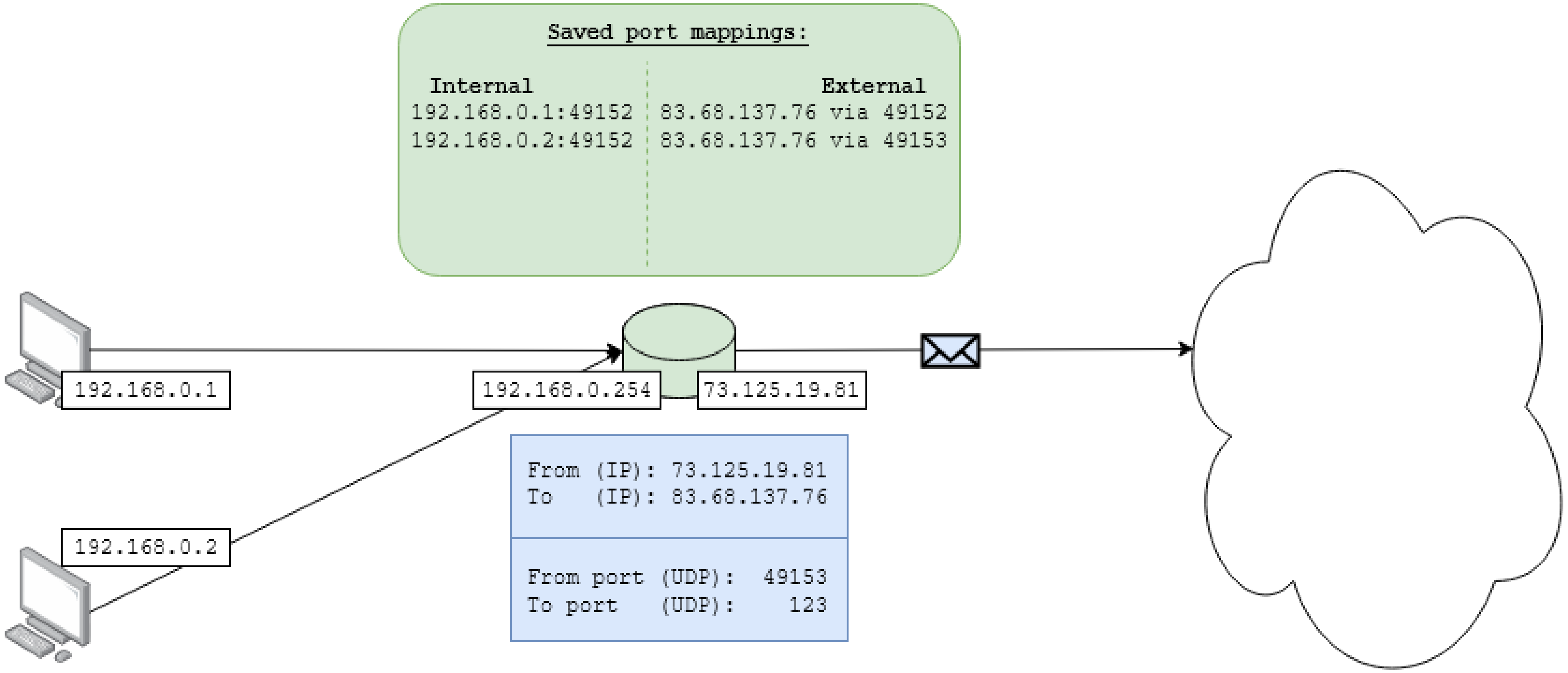
Port Address Translation

- "Hide" an entire private network behind a single public IP
  - Rewrite IP packets at the boundary



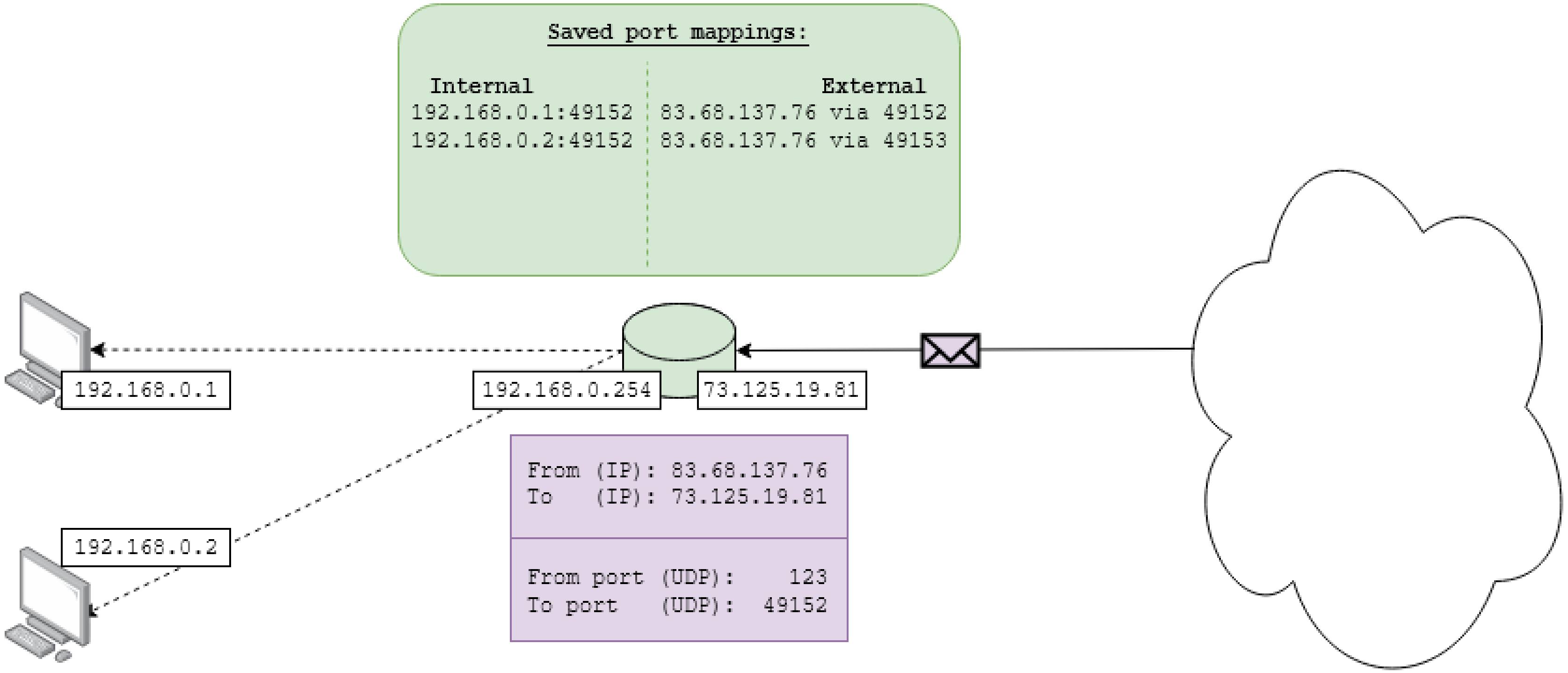
Port Address Translation

- “Hide” an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
  - Rewrite TCP/UDP ports to disambiguate



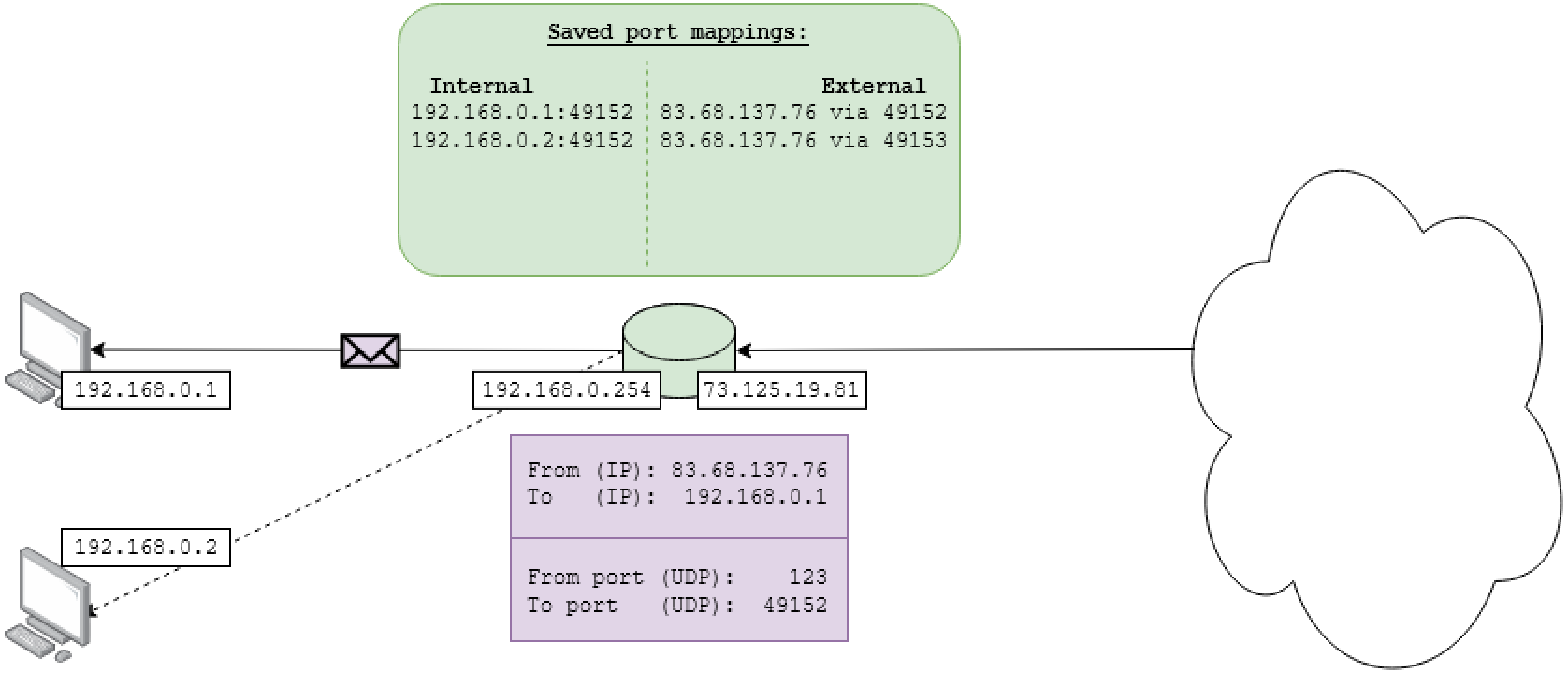
Port Address Translation

- "Hide" an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
  - Rewrite TCP/UDP ports to disambiguate



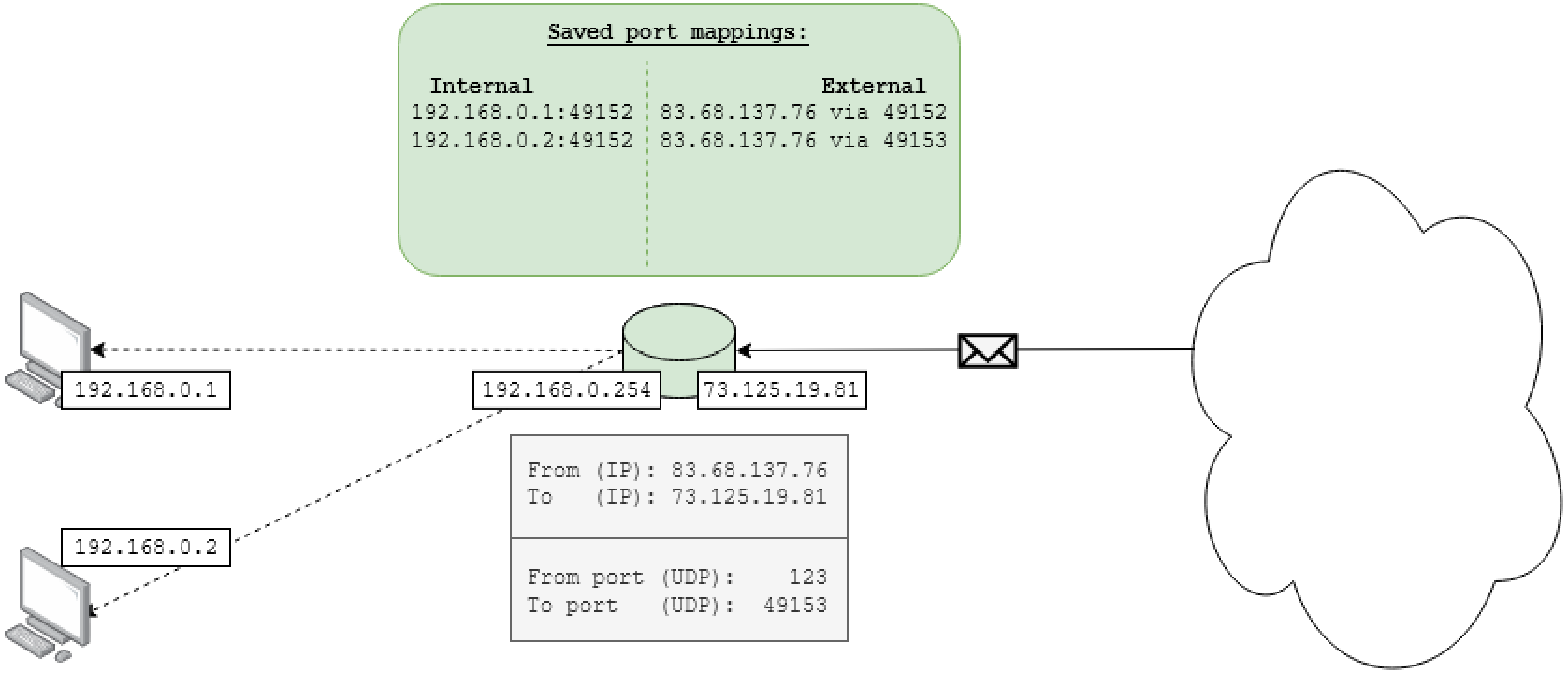
Port Address Translation

- “Hide” an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
  - Rewrite TCP/UDP ports to disambiguate



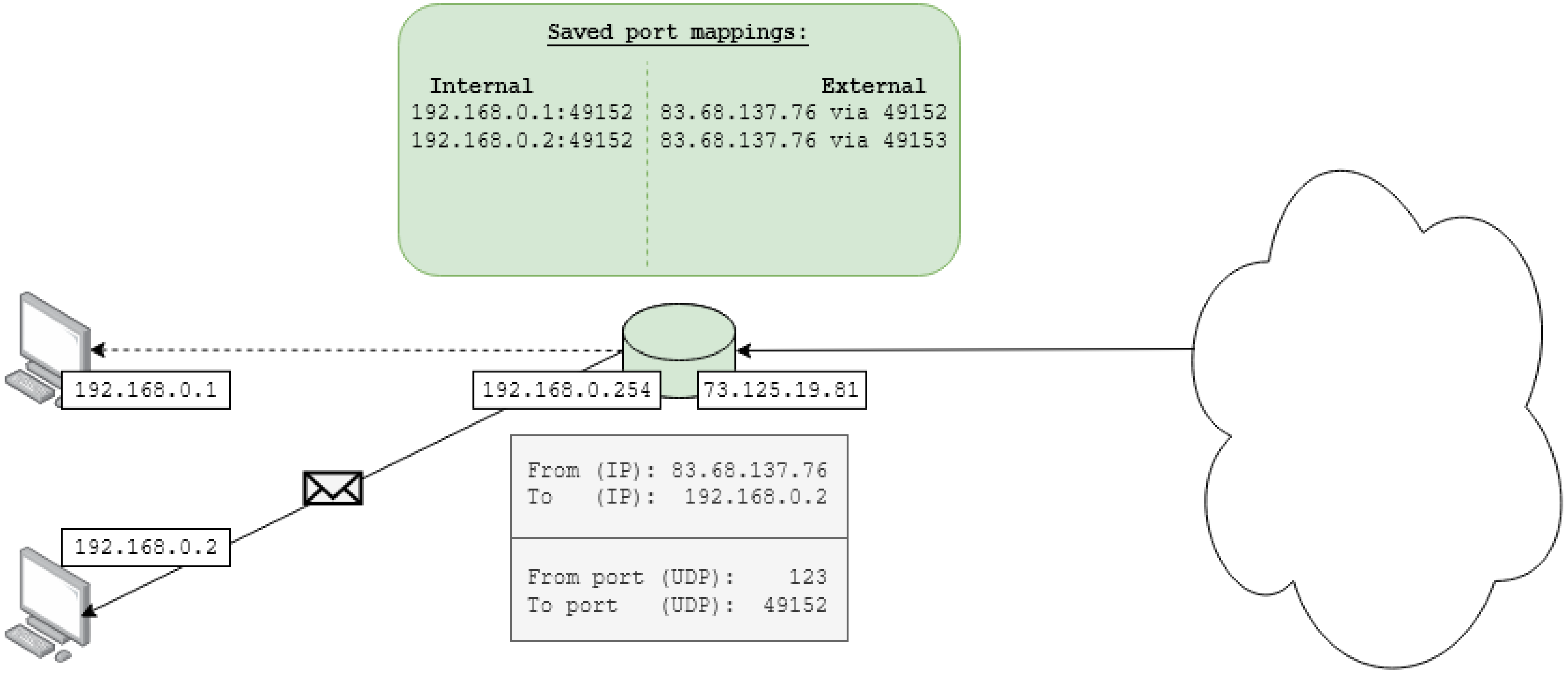
Port Address Translation

- "Hide" an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
  - Rewrite TCP/UDP ports to disambiguate



Port Address Translation

- “Hide” an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
  - Rewrite TCP/UDP ports to disambiguate



Port Address Translation

- "Hide" an entire private network behind a single public IP
  - Rewrite IP packets at the boundary
  - Rewrite TCP/UDP ports to disambiguate
- Transparent if a client "inside" connects to a server "outside"
  - The reverse will not work (by default)
- You can have PAT networks nested within PAT networks
  - Entire ISPs can connect all their clients using one publicly-routable IP address!
- Your home ISP router almost definitely does this!
  - Compare your **`ipconfig`**/**`ifconfig`** address with "what's my ip" (google)