CSCI 466: Networks

Transport Layer (Multiplexing, Error Checking)

Reese Pearsall

Fall 2024

*All images are stolen from the internet

Announcements

PA1 Due **Sunday** @ 11:59 PM

- Files must be pushed to a PA1 folder on your GitHub Repo
- Video demo is required
- Submit your repo link to D2L when finished

- Adding `time.sleep()` after your `.send` calls can resolve timing issues

Quiz on Friday

Put a README in head of your folders. Provide the video demo link, group members, and instructions for how to run

[redacted]-csci466 / PA1 /

[redacted]

Name	Last commit message
..	
README.md	Update README.md
client.py	Edits to client.py and server.py
server.py	Edits to client.py and server.py

README.md

Demo Video: [redacted]

Group Members:

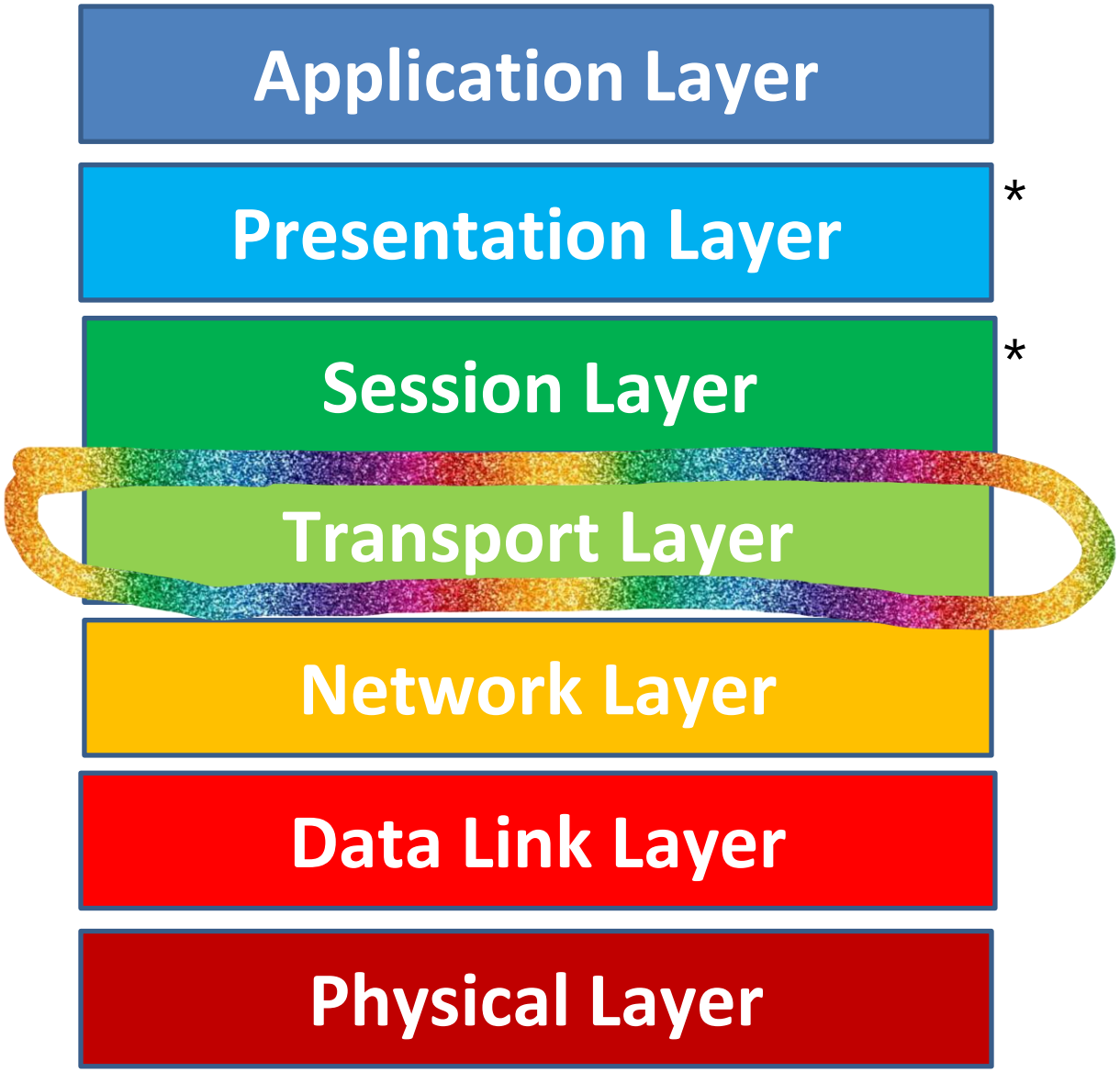
- [redacted]

How to run:

1. Run server.py via the command line and give it an extra instruction for the port number (python server.py [PORT NUMBER])
2. Run client.py via the command line and give it an extra instruction for the port number (python client.py [PORT NUMBER])
3. Play Battleship from the terminal running client.py by entering coordinates.

UDP Sockets

OSI Model



Application Layer

Messages from Network Applications

Physical Layer

Bits being transmitted over a copper wire

**In the textbook, they condense it to a 5-layer model, but 7 layers is what is most used*

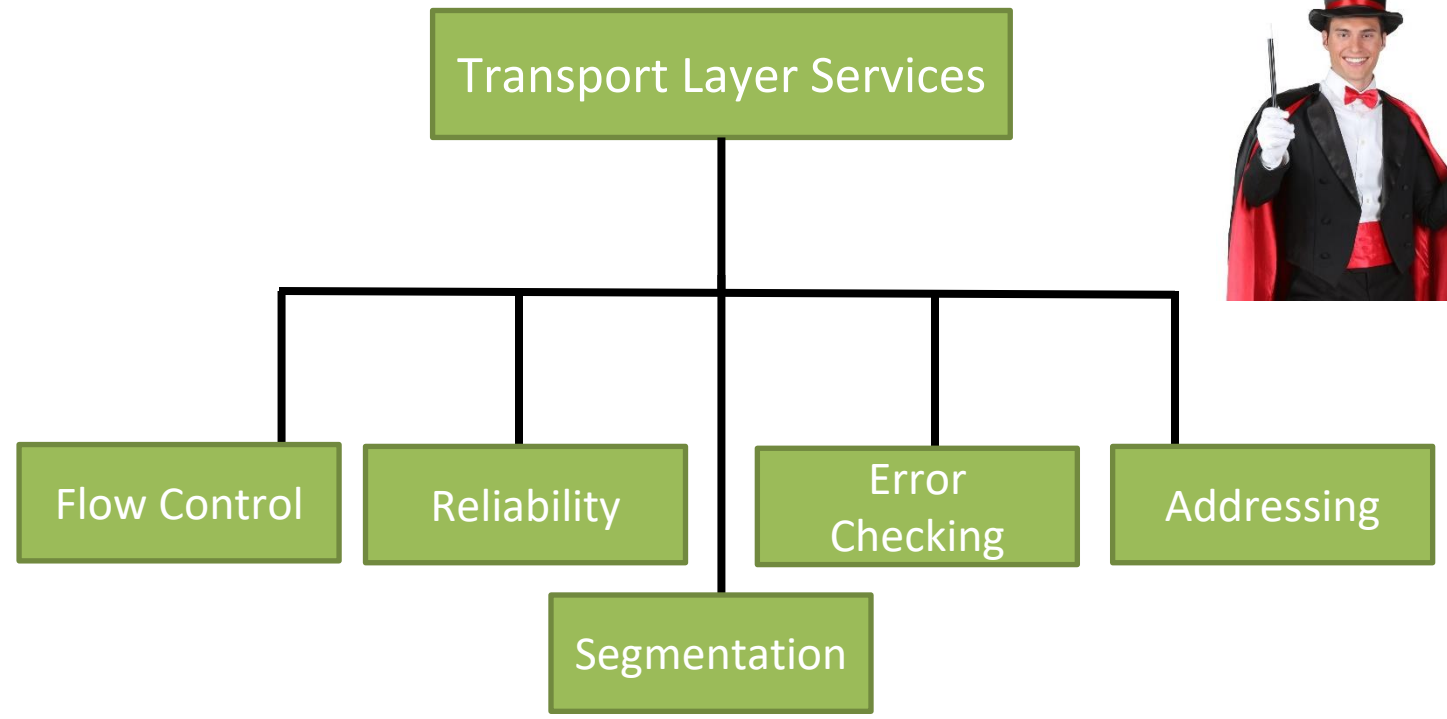
Transport Layer

Provides *logical* end-to-end communication between application **processes**

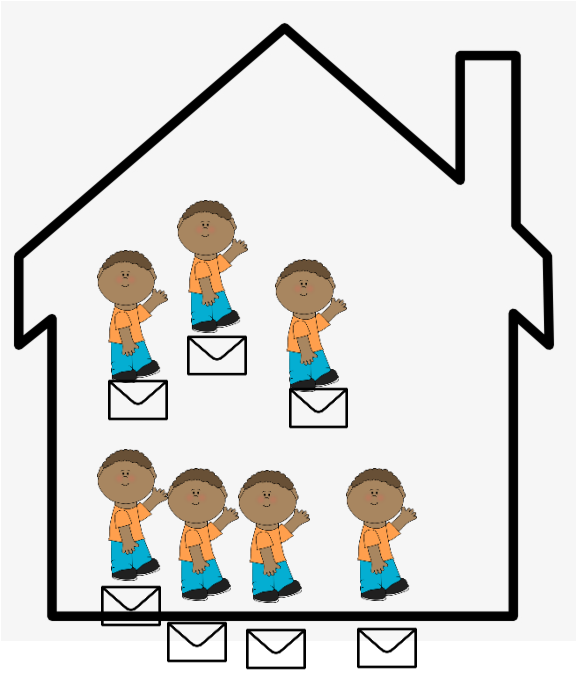
(Network layer handles delivery between **hosts**)

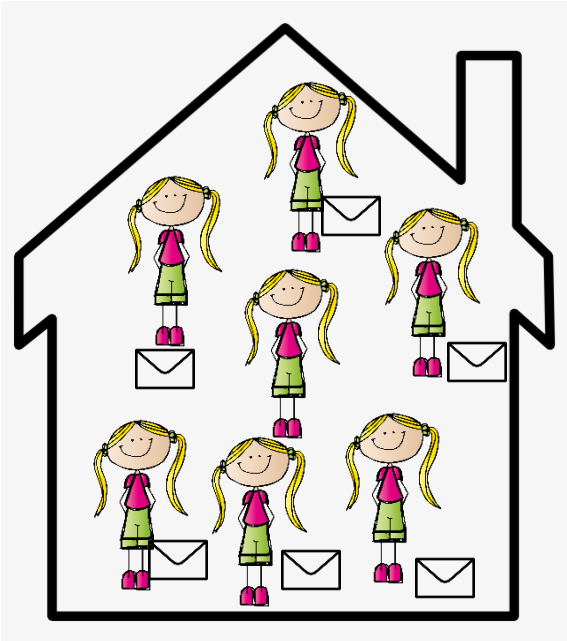
Important Services

- Flow Control
- Reliability
- Segmentation
- Error Checking
- Addressing

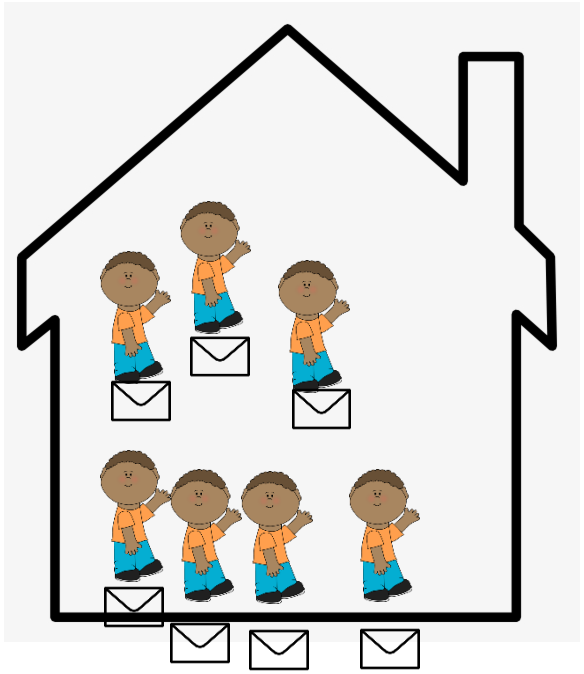


Transport Layer



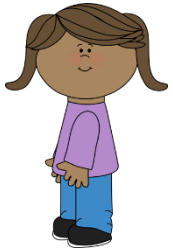


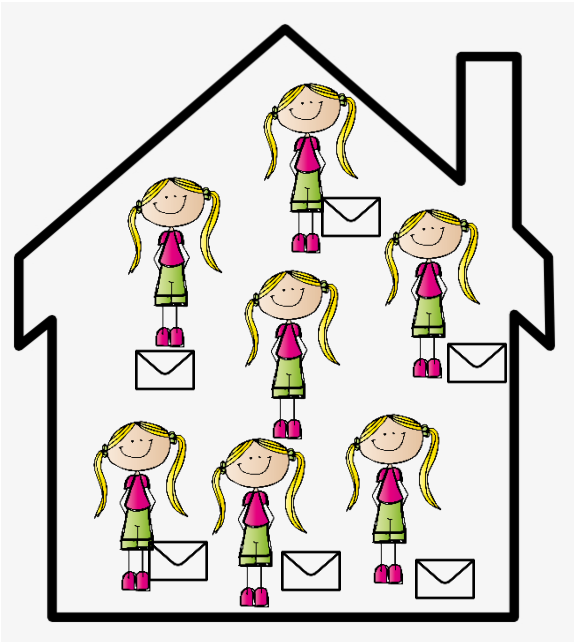
Transport Layer



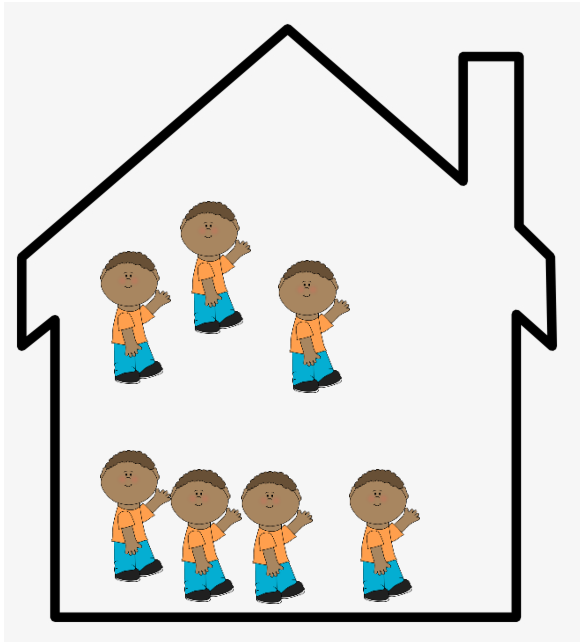
Bill

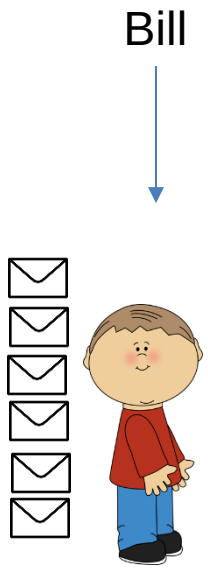
Ann

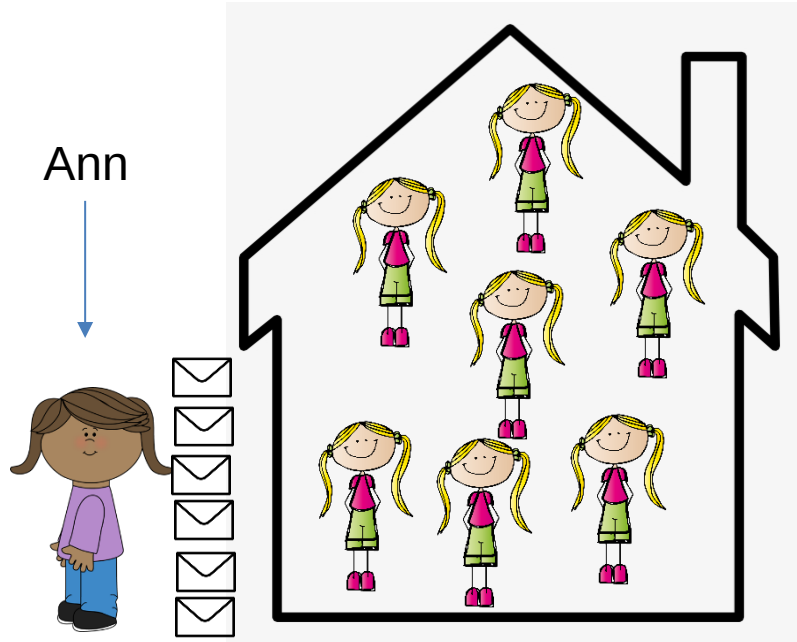




Transport Layer

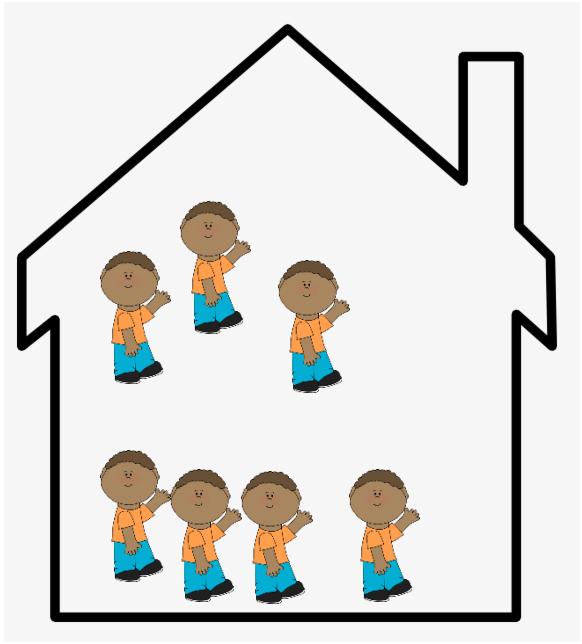


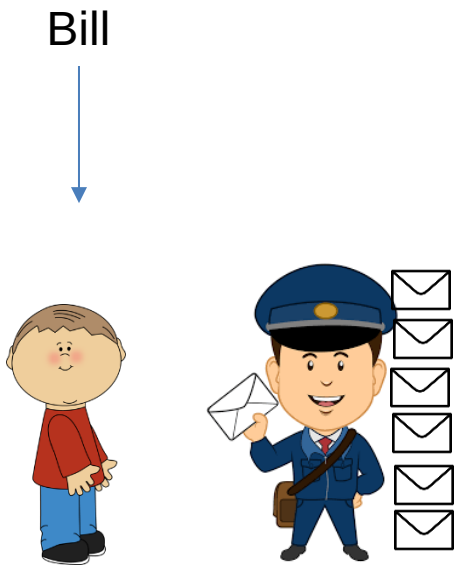


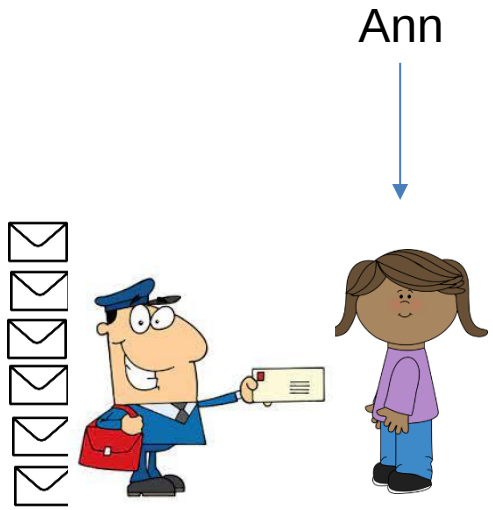


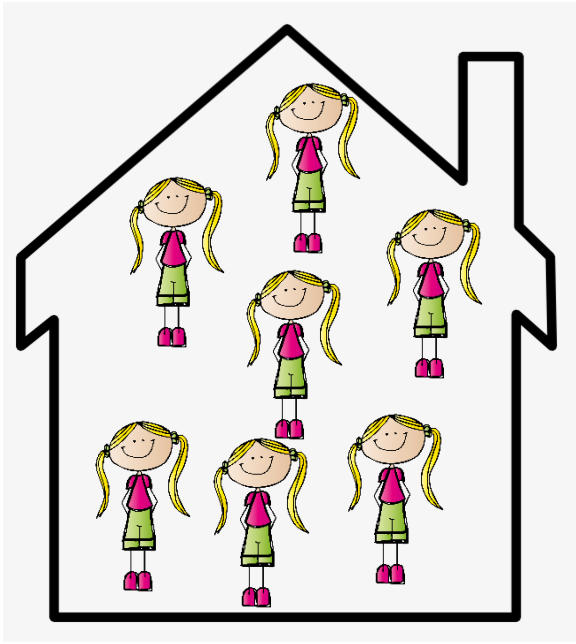
Bill and Ann are responsible for collecting their siblings mail…

Transport Layer





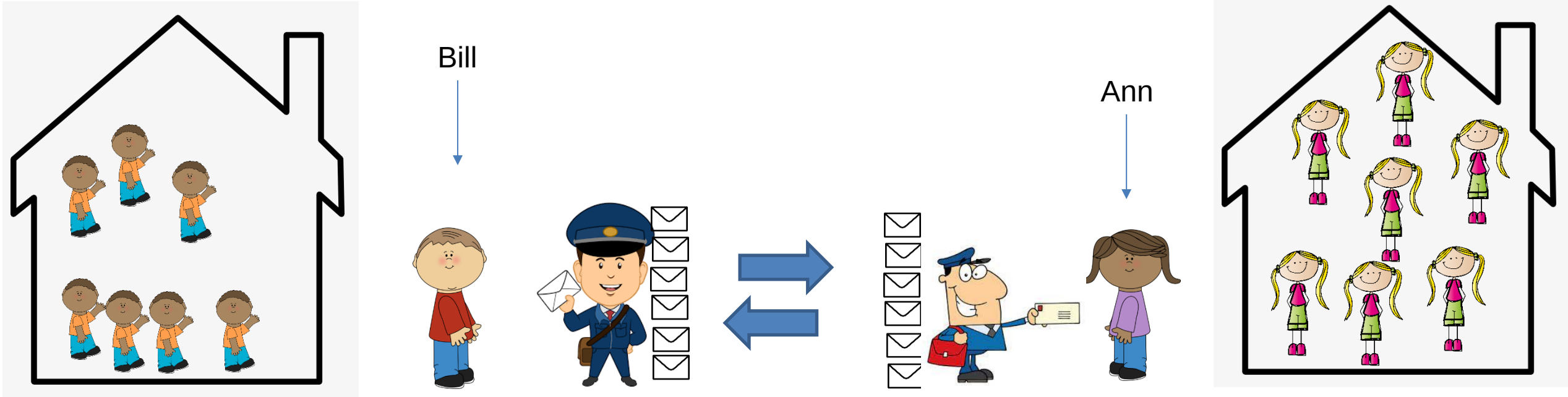




Bill and Ann are responsible for collecting their siblings mail…

And delivering it to the postal service worker

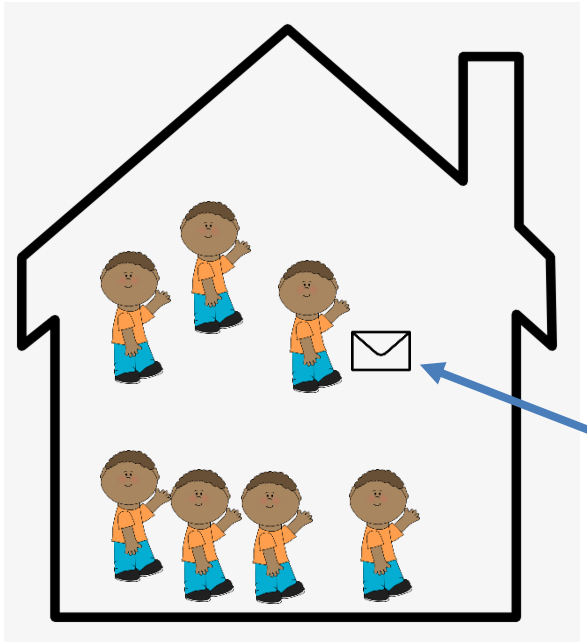
Transport Layer



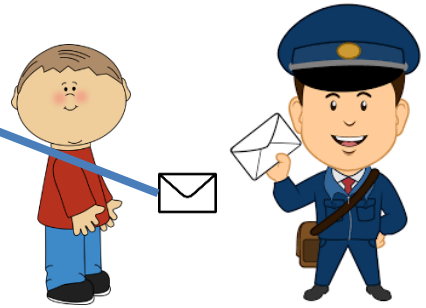
Bill and Ann are responsible for collecting their siblings mail…

And delivering it to the postal service worker

Transport Layer

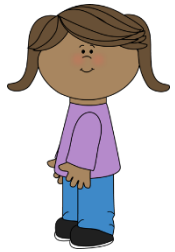


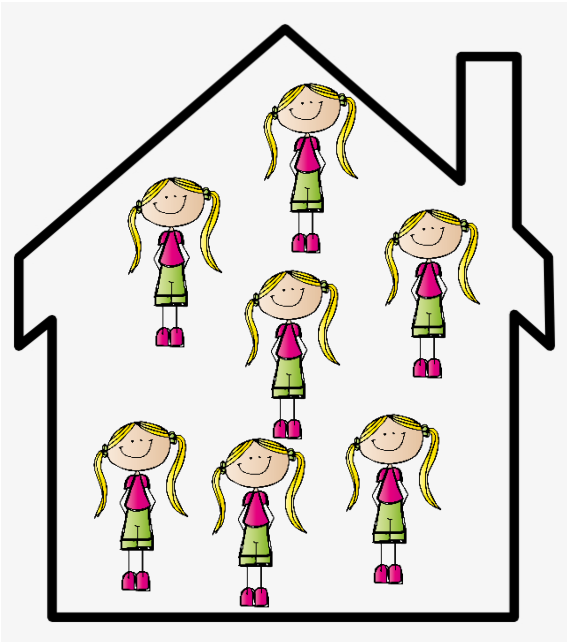
Bill



When new mail arrives, Bill and Ann receive it from the postman and distribute it to the correct cousin

Ann



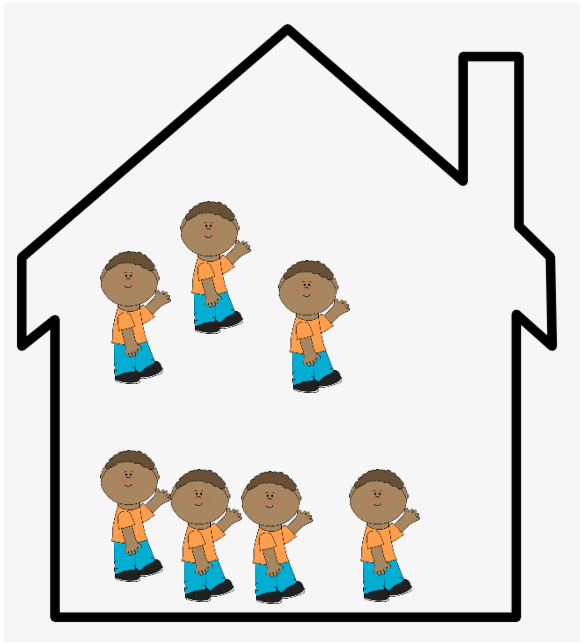


Bill and Ann are responsible for collecting their siblings mail…

And delivering it to the postal service worker

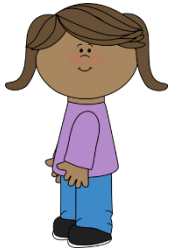
Transport Layer

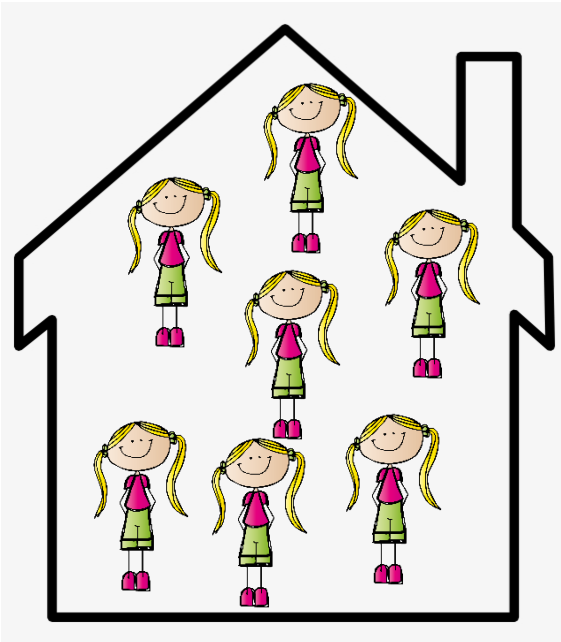
Letters in envelopes = Application messages



Bill

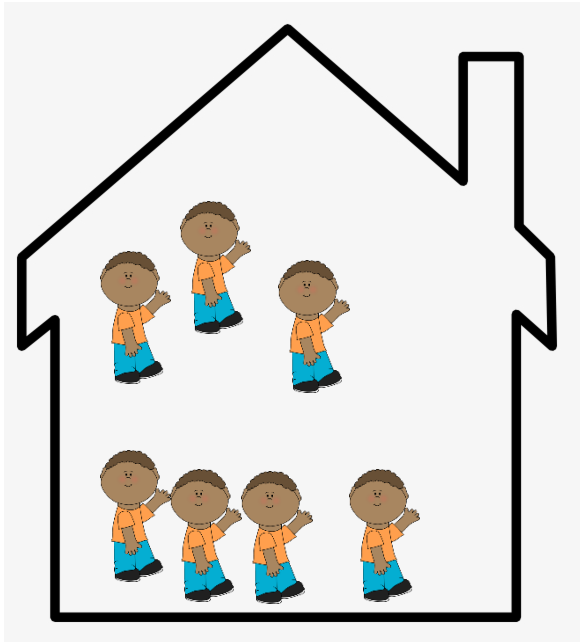
Ann





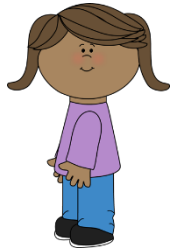
Transport Layer

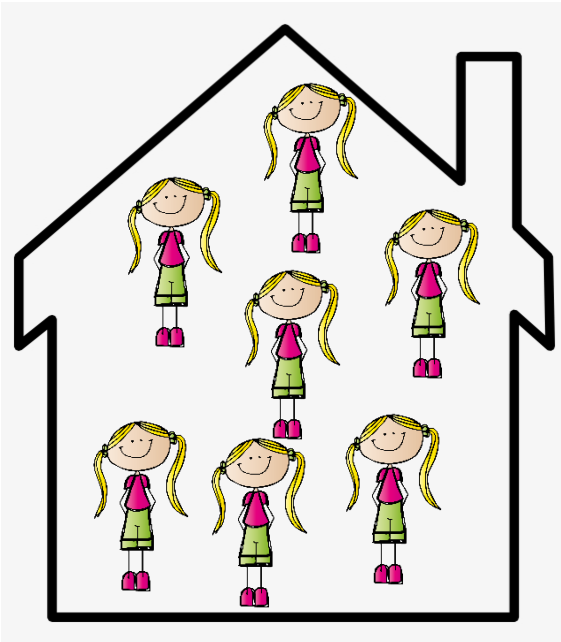
Letters in envelopes = Application messages
Cousins = Processes



Bill

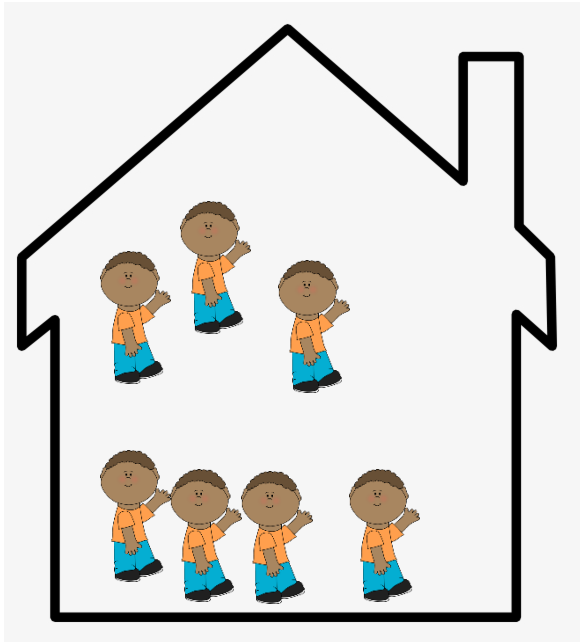
Ann





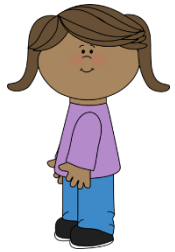
Transport Layer

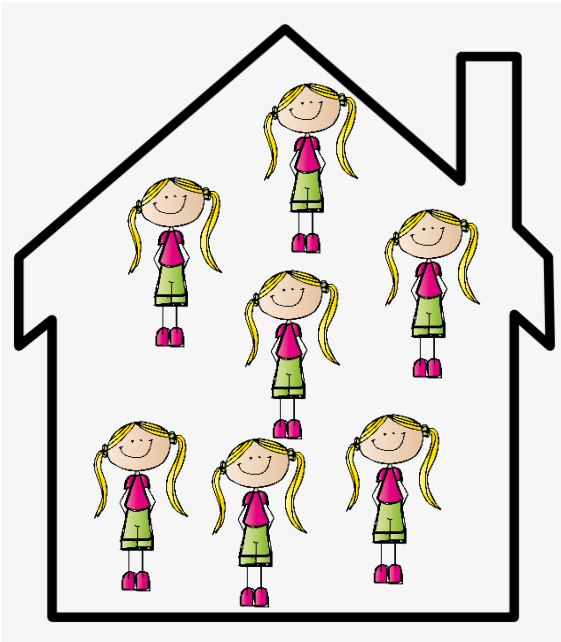
Letters in envelopes = Application messages
Cousins = Processes
Houses = Hosts/End systems



Bill

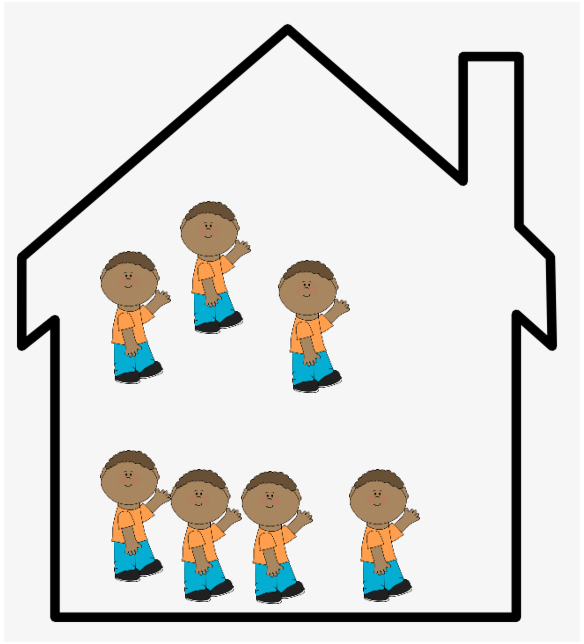
Ann





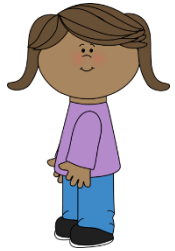
Transport Layer

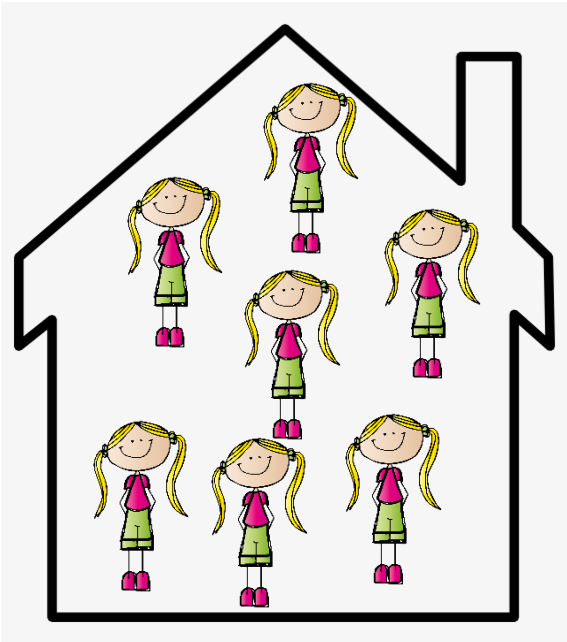
Letters in envelopes = Application messages
Cousins = Processes
Houses = Hosts/End systems
Ann and Bill = Transport Layer Protocol



Bill

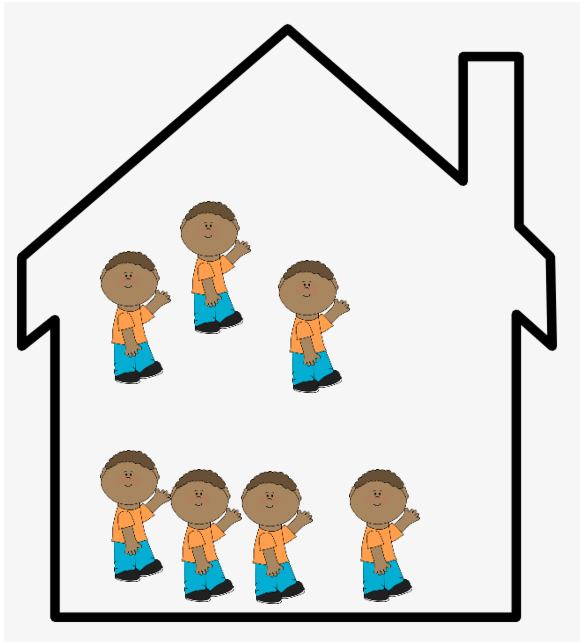
Ann





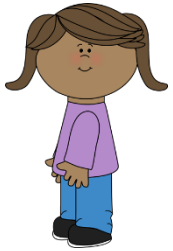
Transport Layer

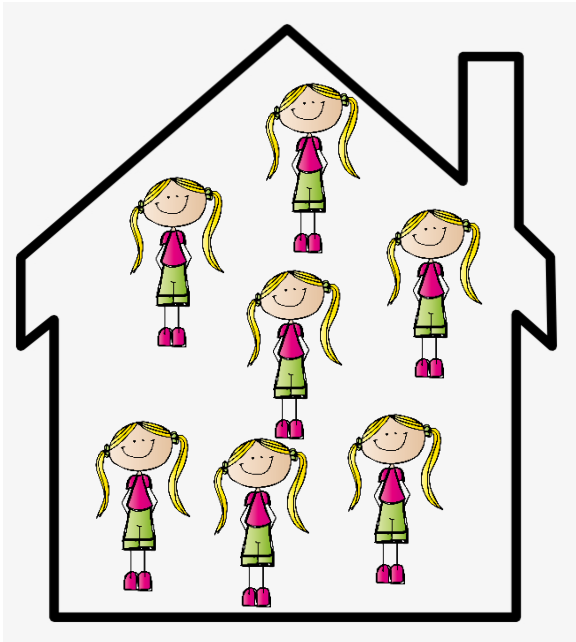
Letters in envelopes = Application messages
Cousins = Processes
Houses = Hosts/End systems
Ann and Bill = Transport Layer Protocol
Postal Service = Network Layer/Network core



Bill

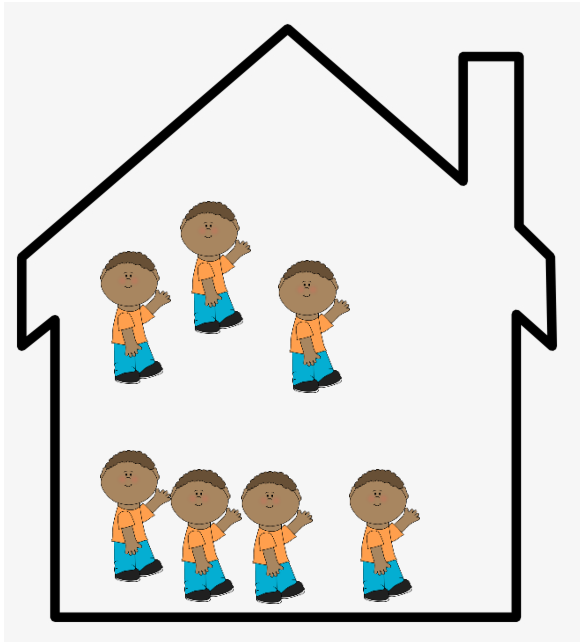
Ann



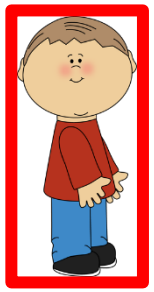


Transport Layer

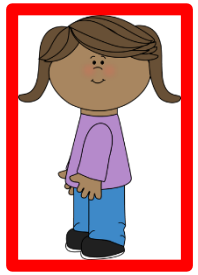
Letters in envelopes = Application messages
Cousins = Processes
Houses = Hosts/End systems
Ann and Bill = Transport Layer Protocol
Postal Service = Network Layer/Network core

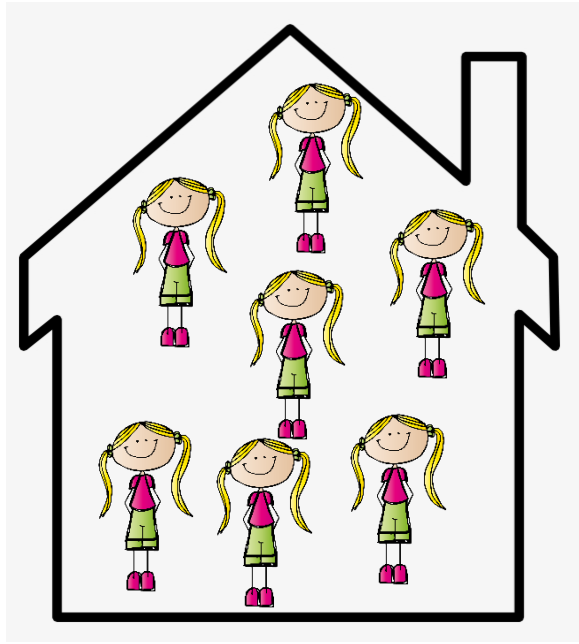


Bill



Ann

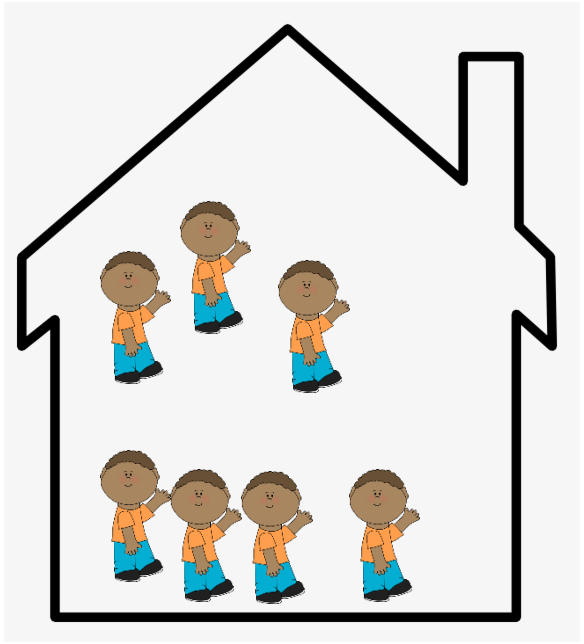




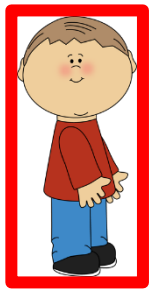
From the perspective of the cousins, Bill and Ann are the postal service

Transport Layer

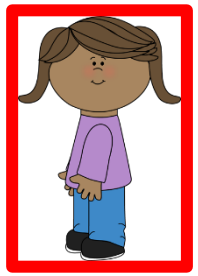
Letters in envelopes = Application messages
Cousins = Processes
Houses = Hosts/End systems
Ann and Bill = Transport Layer Protocol
Postal Service = Network Layer/Network core

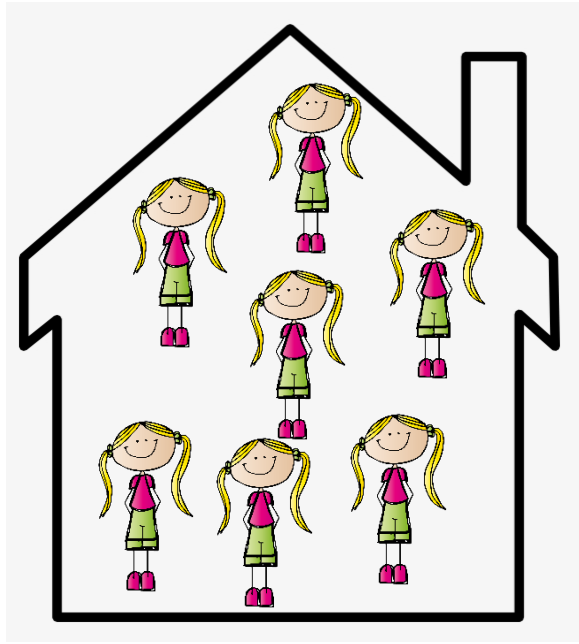


Bill



Ann





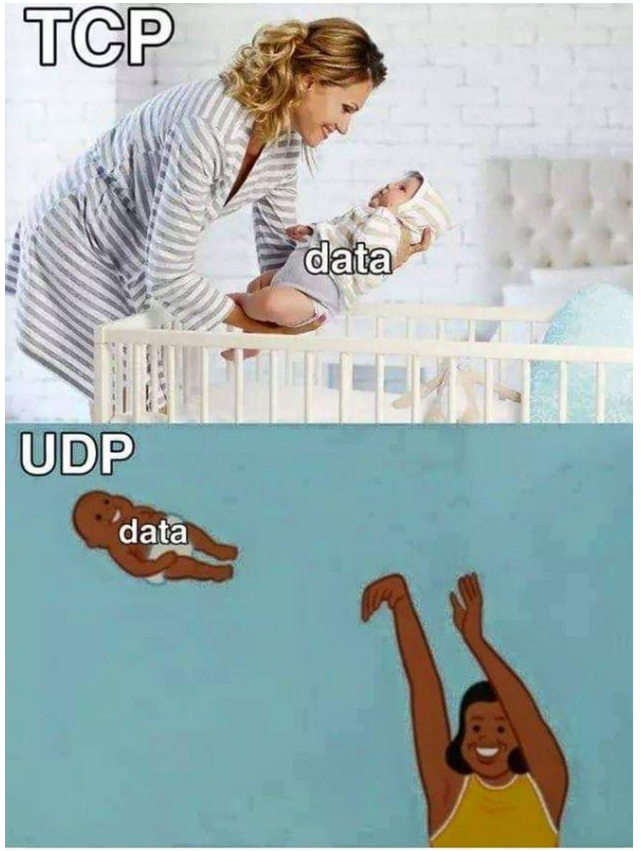
From the perspective of the cousins, Bill and Ann are the postal service

What if Bill and Ann are sick?

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocol

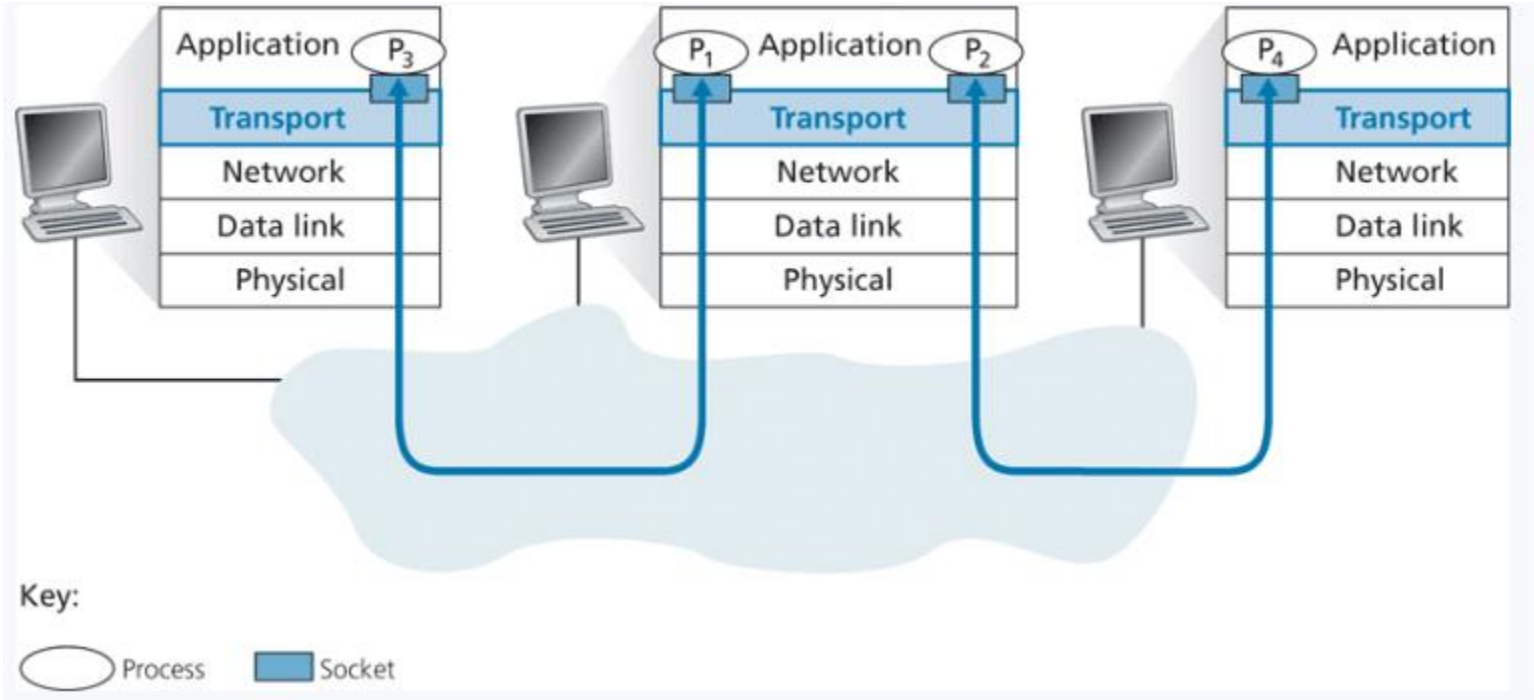
Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Secure remote terminal access	SSH	TCP
Web	HTTP, HTTP/3	TCP (for HTTP), UDP (for HTTP/3)
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	DASH	TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Name translation	DNS	Typically UDP



Transport Layer

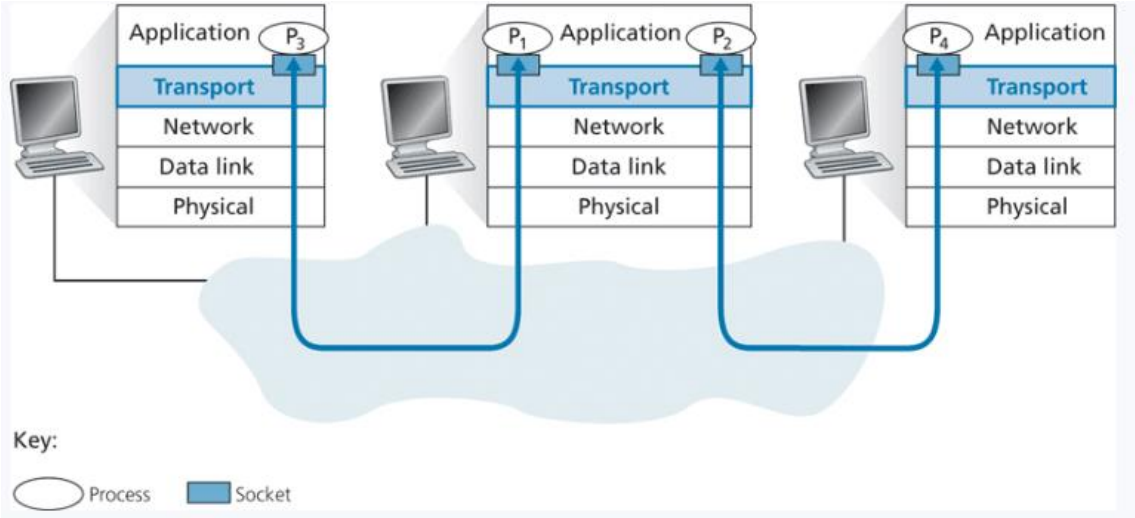
TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocol

Transport layer delivers to **sockets**

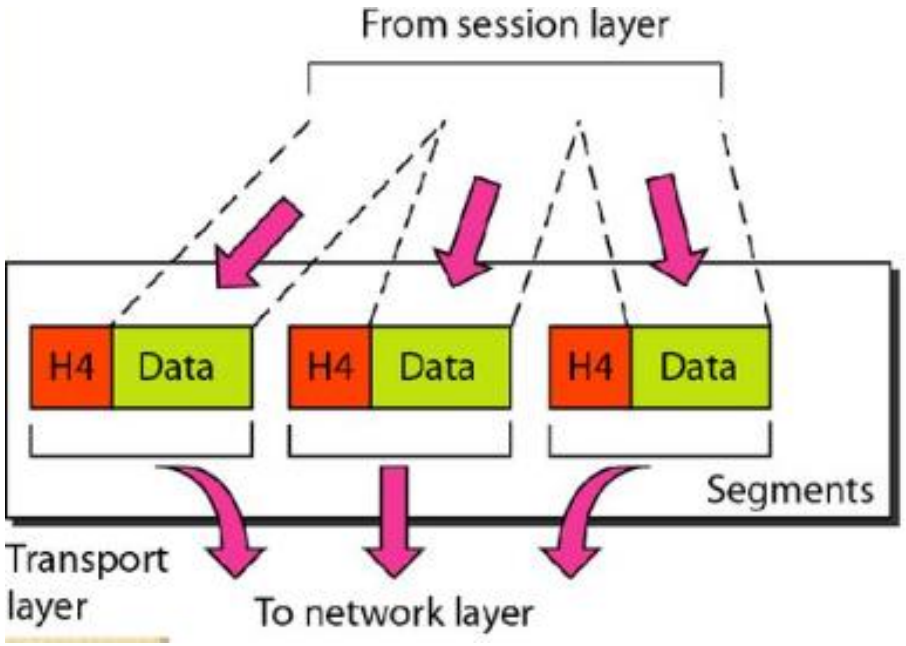


Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl



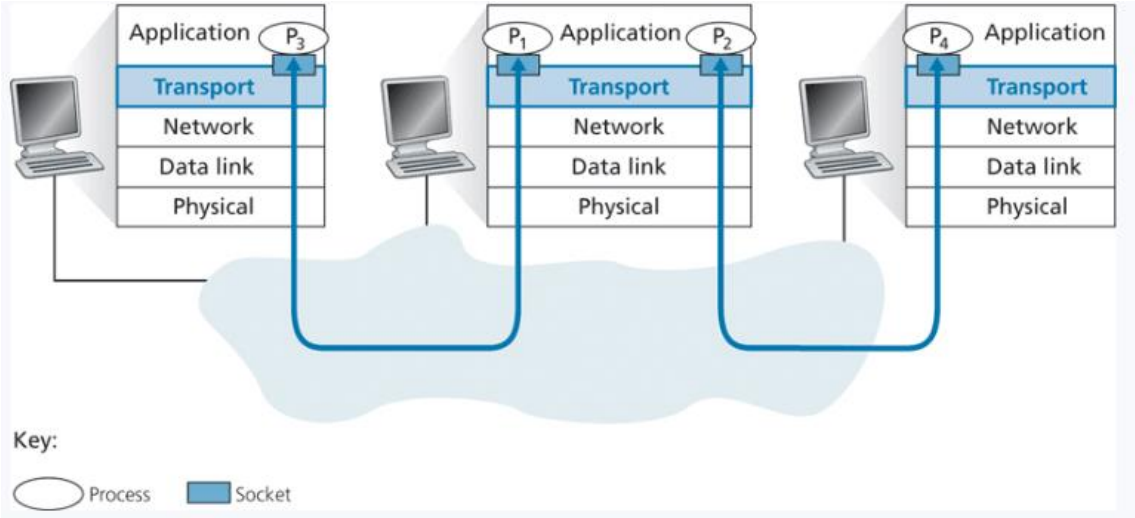
Transport layer delivers to **sockets**



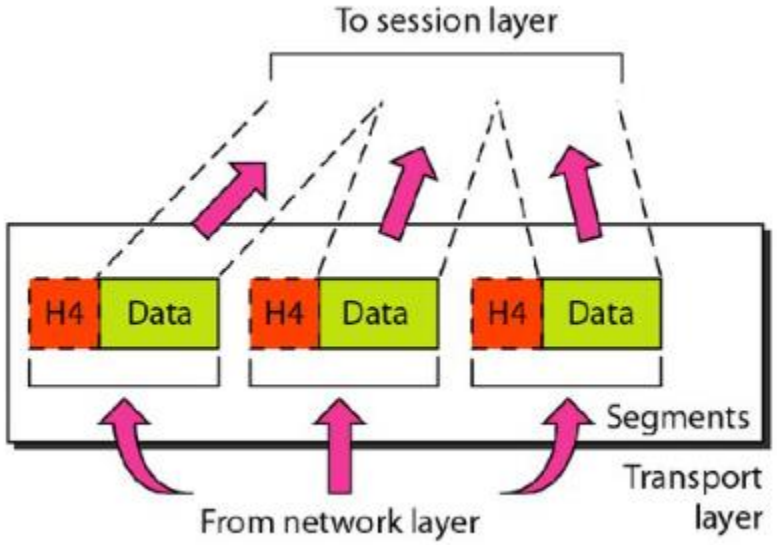
Messages from the Application Layer/Session Layer are split into smaller chunks called **segments**, and passed into the network layer

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl



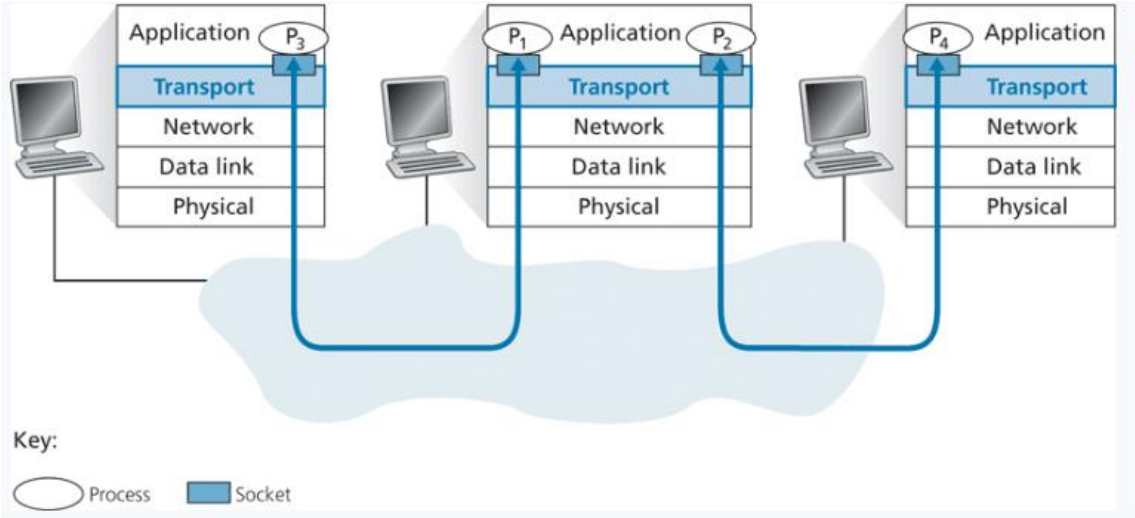
Transport layer delivers to **sockets**



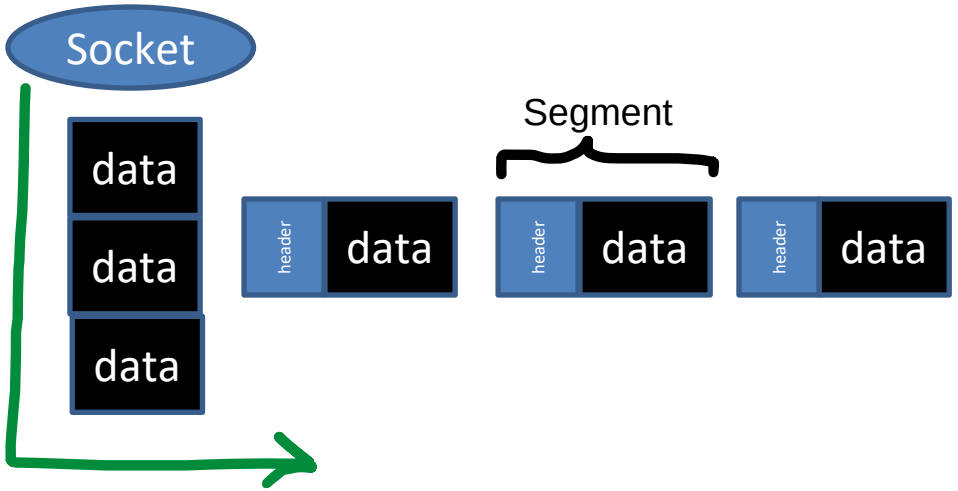
Messages from the network layer arrive as segments. Transport layer must reassemble to send it to the correct process

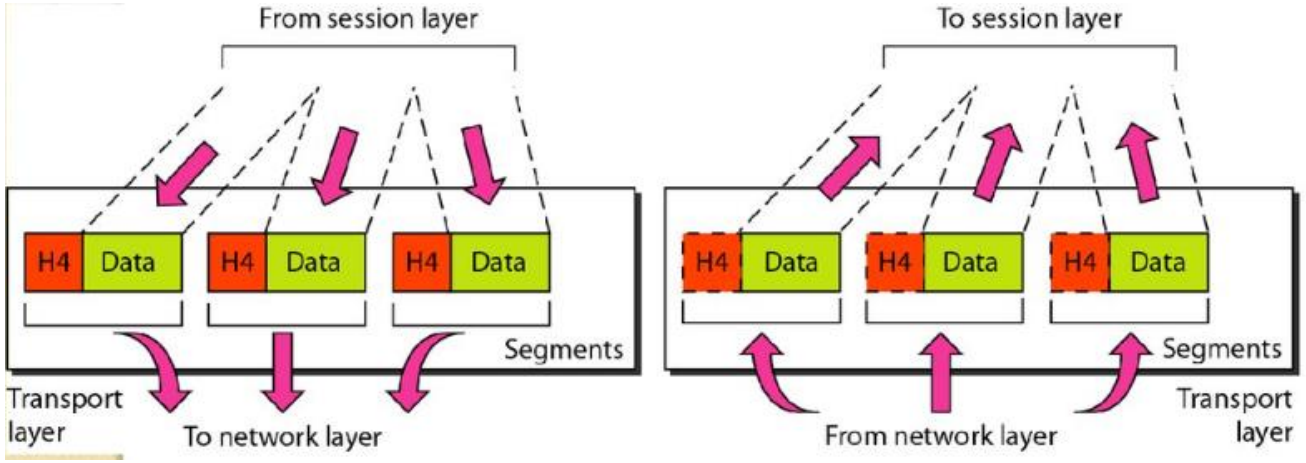
Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocol



Multiplexing is the process of gathering chunks from sockets, encapsulating chunks with header information, and passing the segment into the network layer

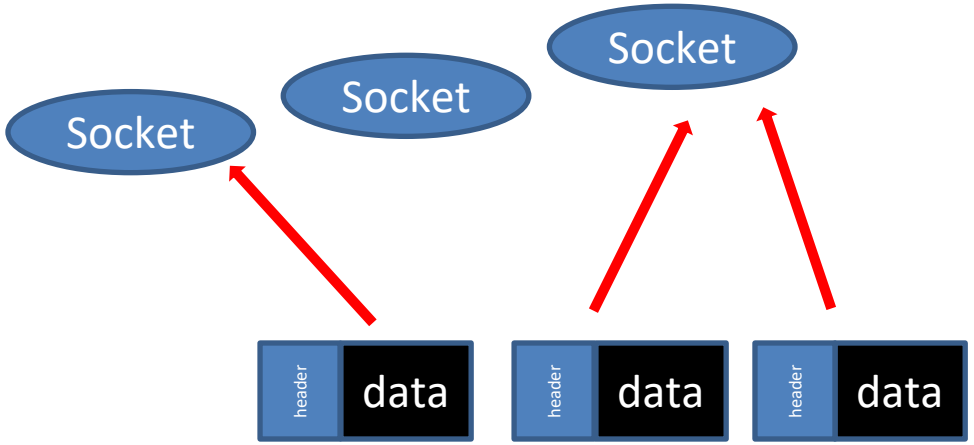


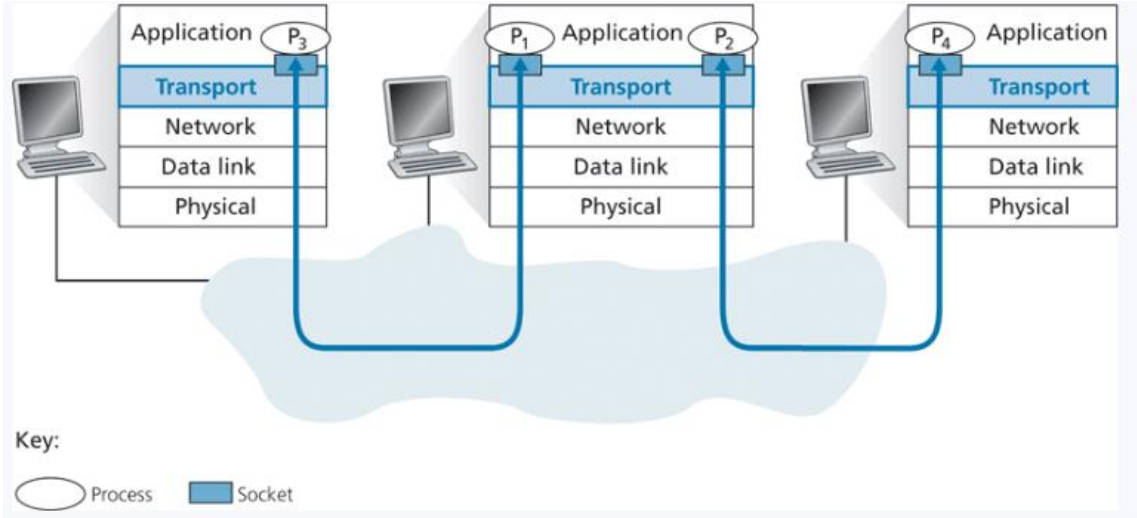


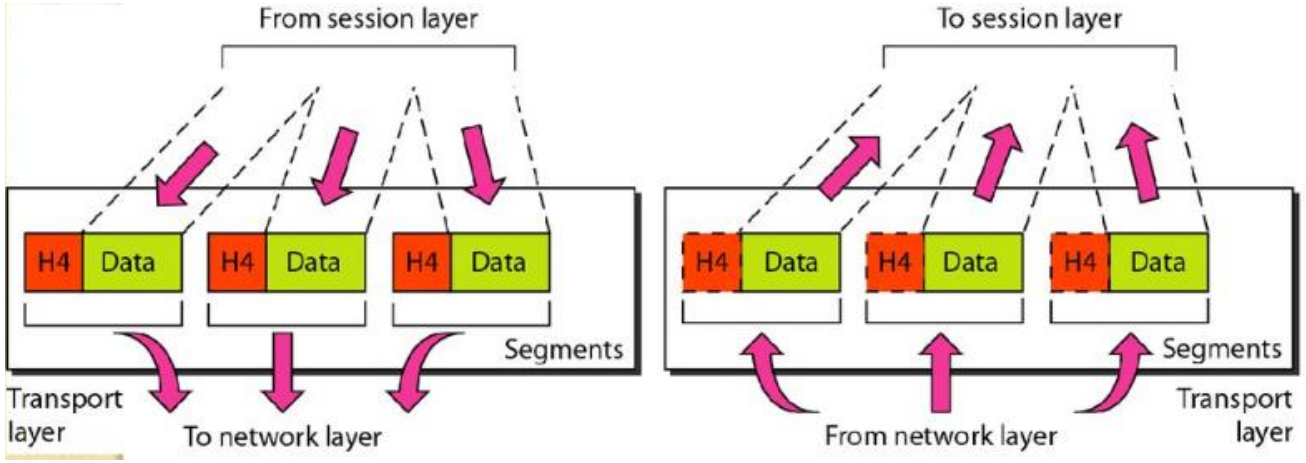
Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl

Demultiplexing is the receiving segments from the transport layer and delivering the segment to the correct socket.

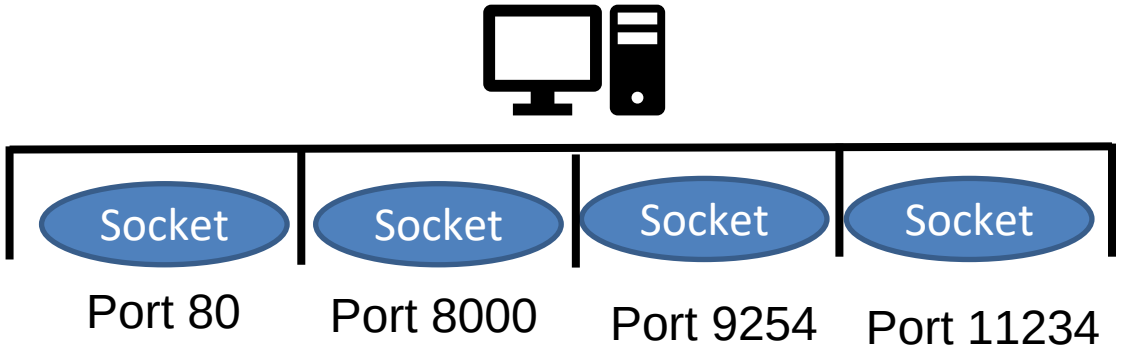


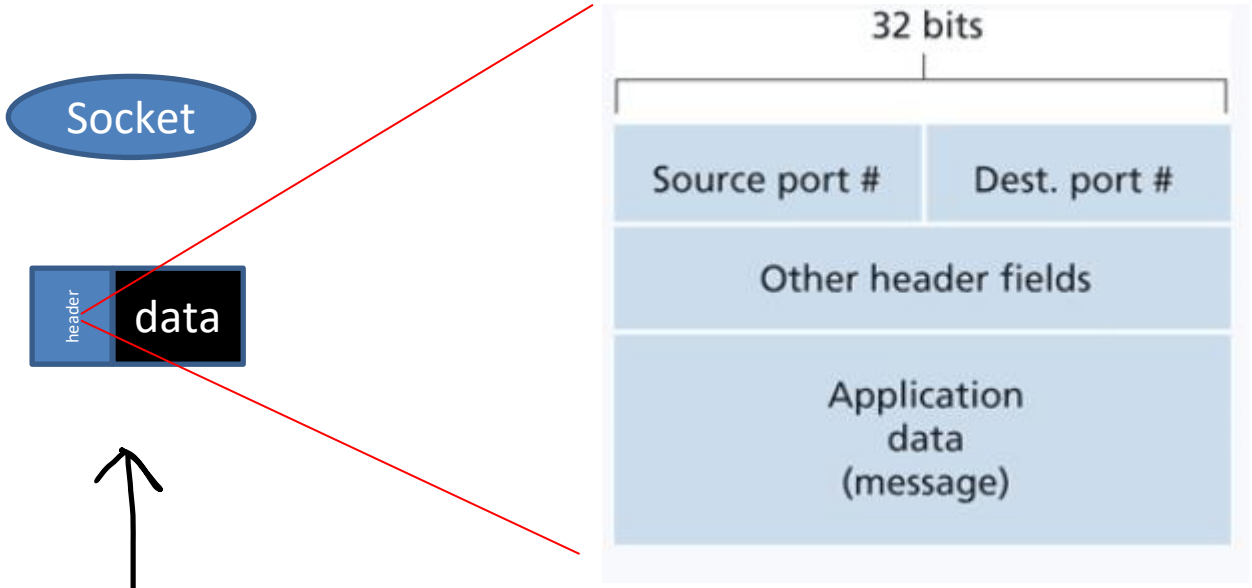




Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl





A **port** is a 16-bit number that is an entrance/exit point of a machine

An active port is associated with a specific process or service

Network Layer

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl

Port	Request type
7	ECHO
20	FTP -- Data
21	FTP -- Control
22	SSH Remote Login Protocol
23	Telnet
25	Simple Mail Transfer Protocol (SMTP)
37	Time
53	Domain Name System (DNS)
69	Trivial File Transfer Protocol (TFTP)
79	Finger
80	HTTP
110	POP3
115	Simple File Transfer Protocol (SFTP)
137	NetBIOS Name Service
139	NetBIOS Datagram Service
143	Interim Mail Access Protocol (IMAP)
156	SQL Server
161	SNMP
194	Internet Relay Chat (IRC)
389	Lightweight Directory Access Protocol (LDAP)
443	HTTPS
445	Microsoft-DS
458	Apple QuickTime
546	DHCP Client
547	DHCP Server

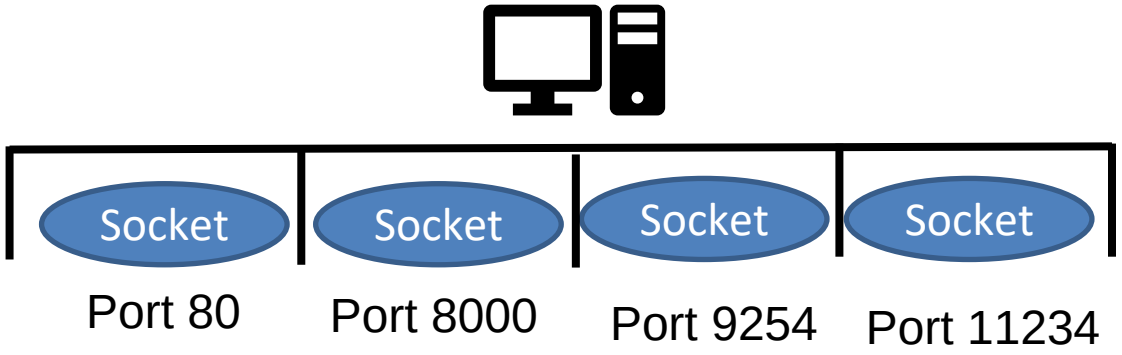
Ones you should probably remember

- HTTP: 80
- HTTPS: 443
- DNS: 53
- SSH: 22
- FTP: 20/21
- SMTP: 25

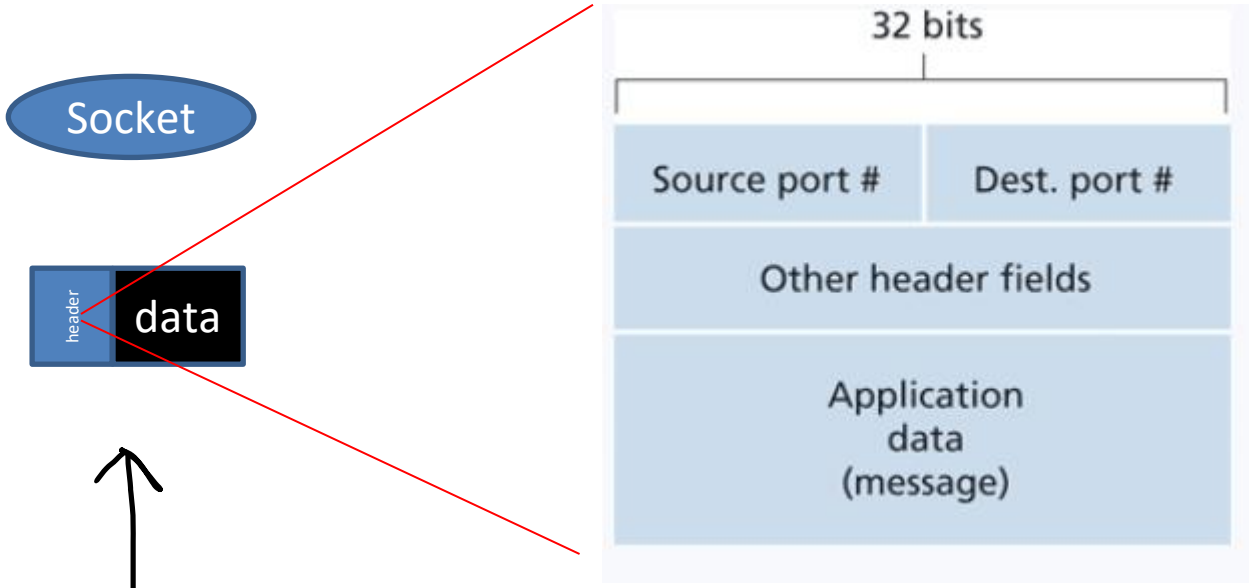
You can request to register a service under a specific port number through the Internet Assigned Numbers Authority (**IANA**)

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocol



(Demultiplexing)

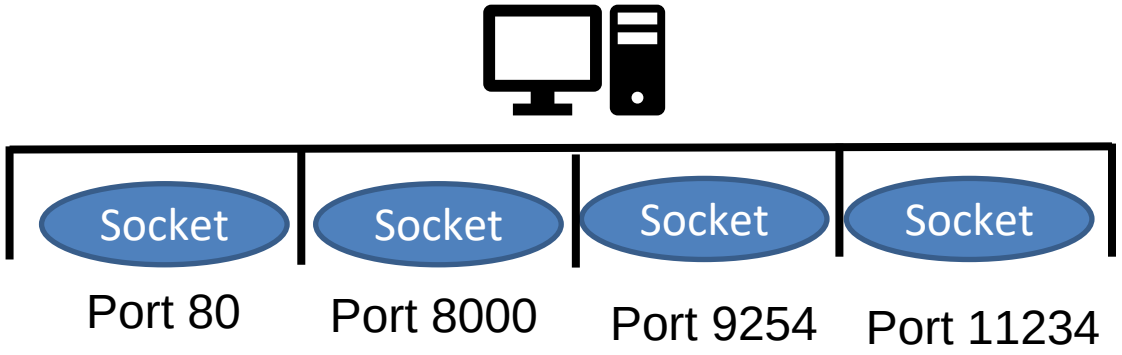


TCP/UDP Segment

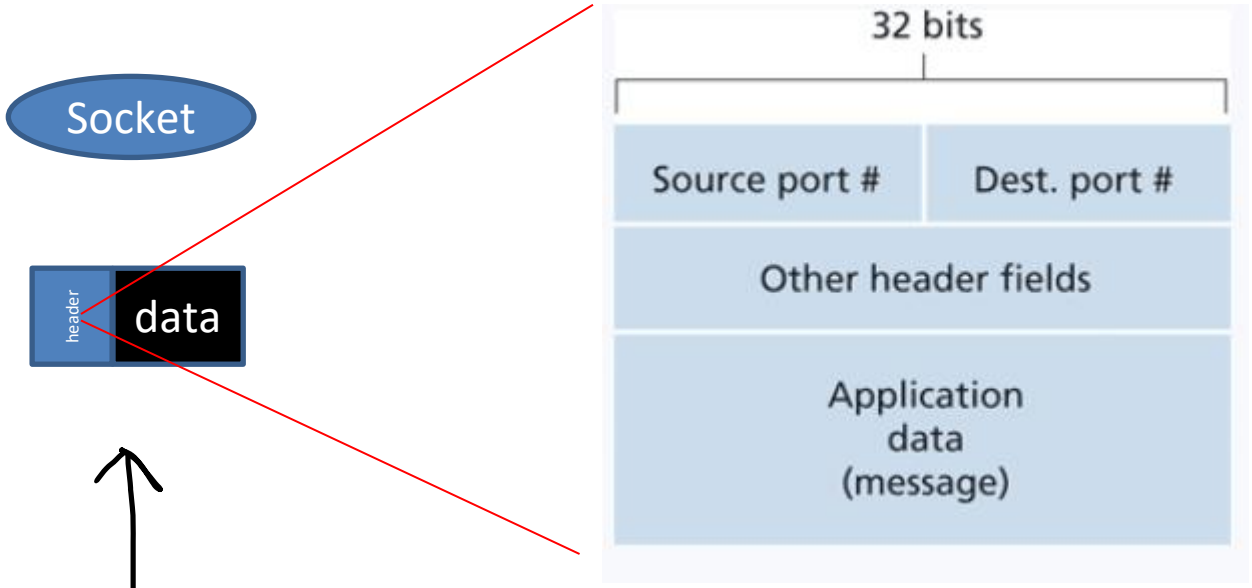
Network Layer

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocol



(Demultiplexing)

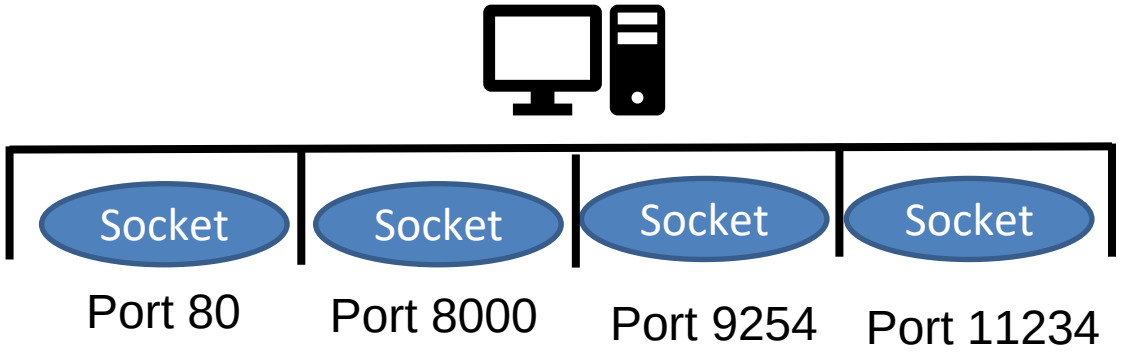


TCP/UDP Segment

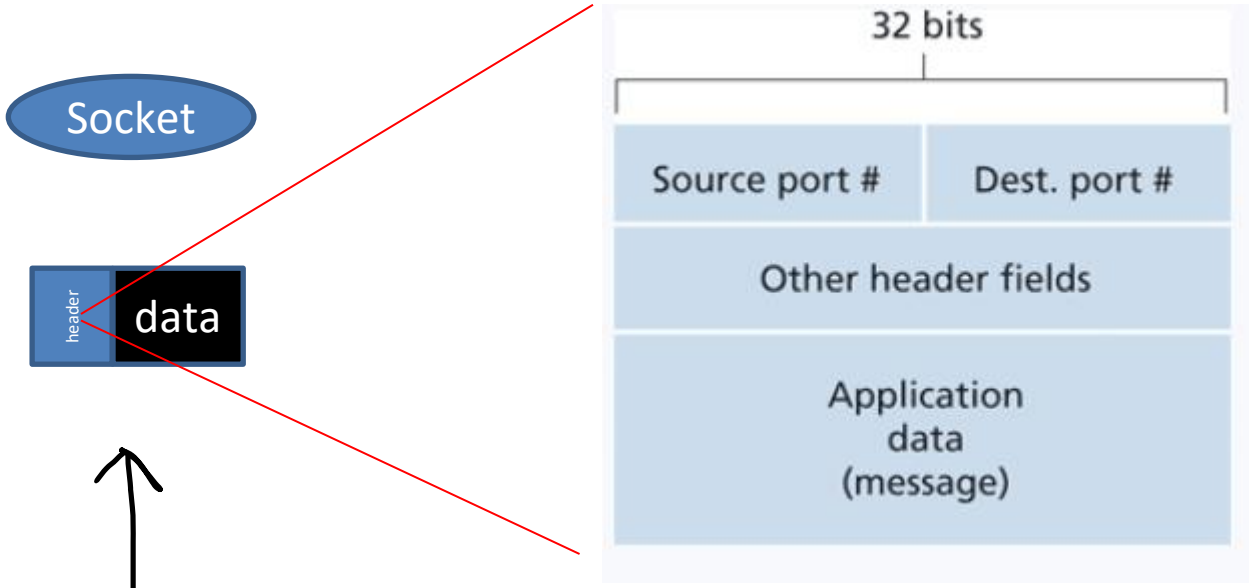
Network Layer

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocol



(Demultiplexing)

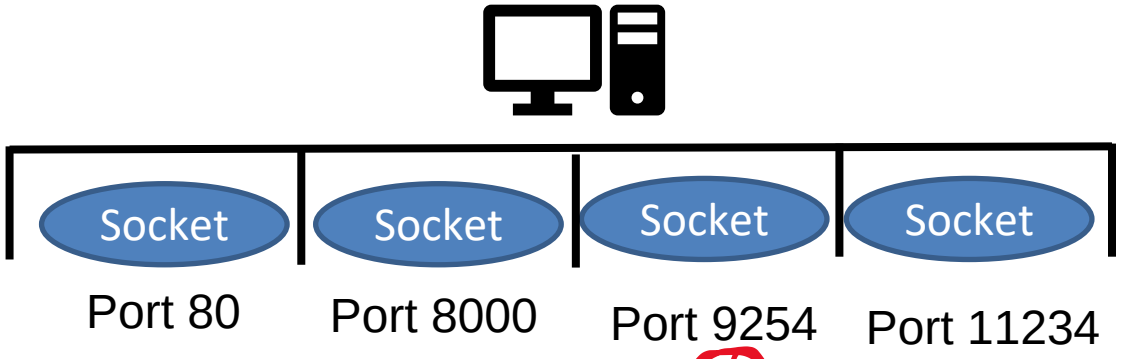


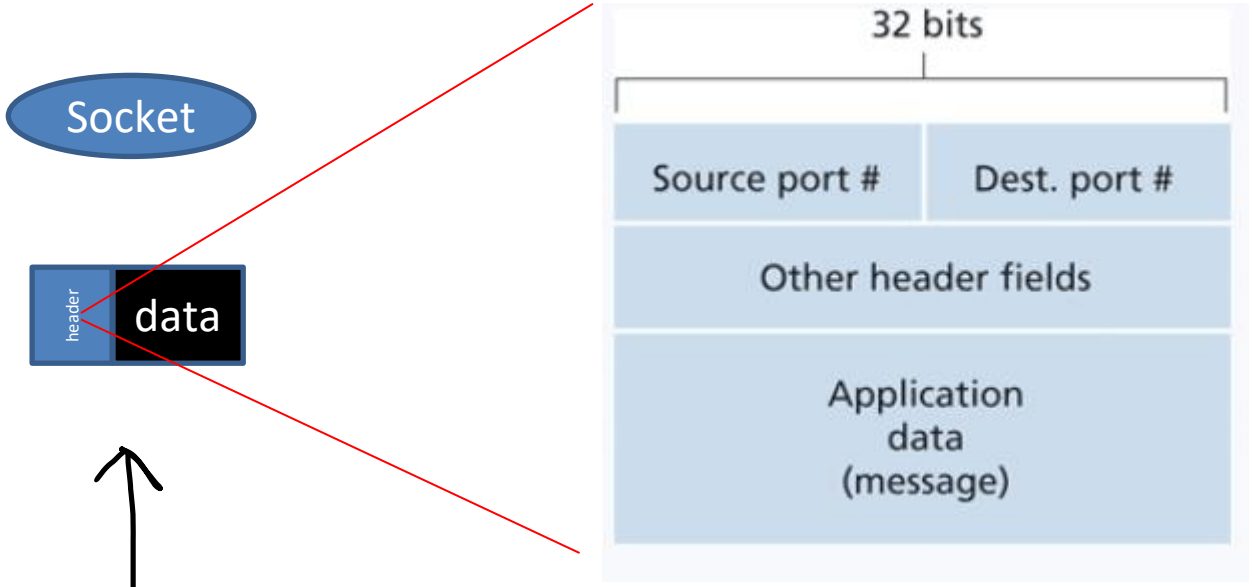
TCP/UDP Segment

Network Layer

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl



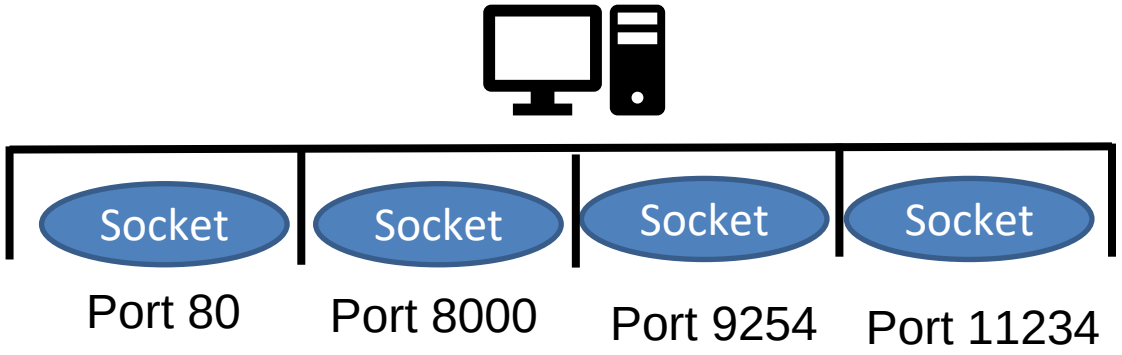


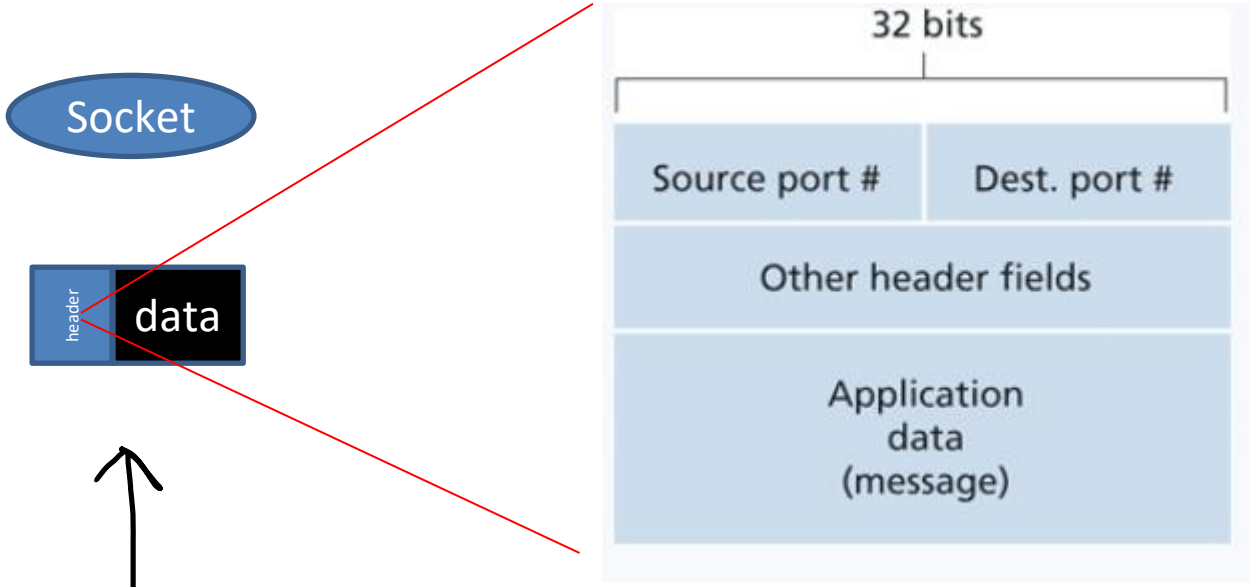
TCP/UDP Segment

Network Layer

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl





A **port** is a 16-bit number that is an entrance/exit point of a machine

An active port is associated with a specific process or service

When developing a new application, we need to assigned the application a port number (greater than 1024)

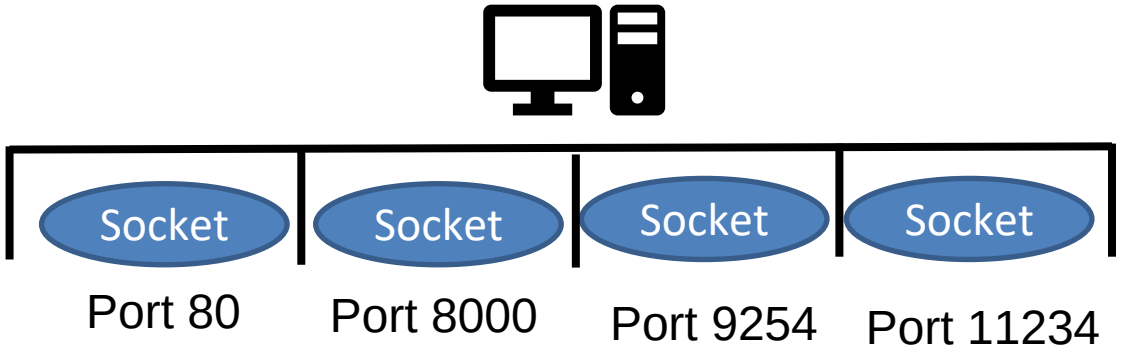
Network Layer

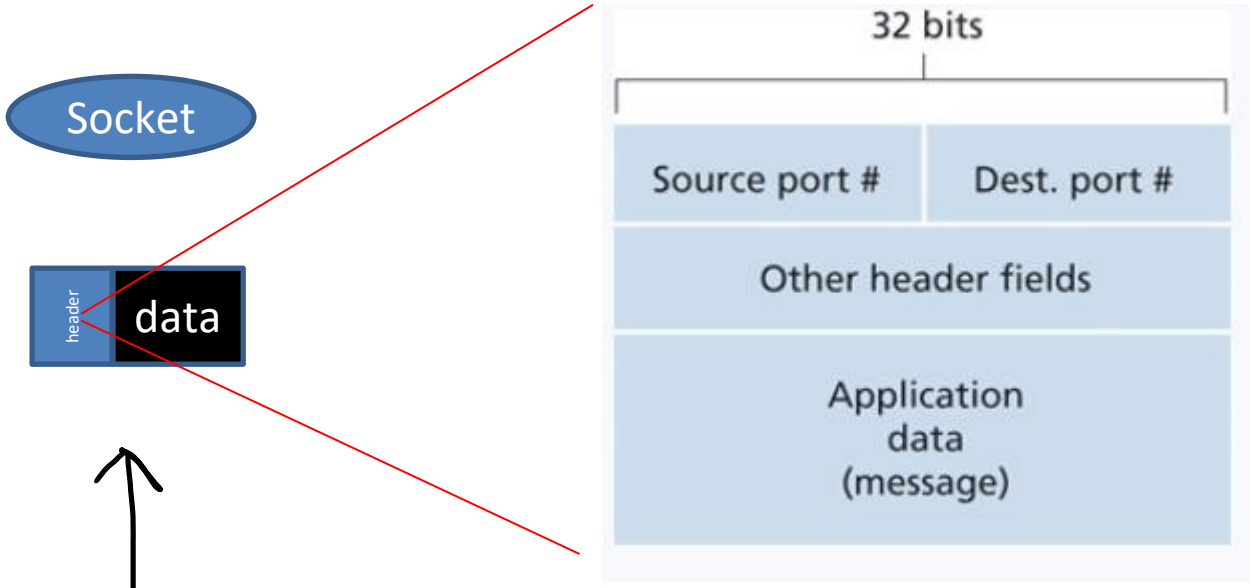
```
clientSocket.bind(('', 19157))
```

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

Transport Layer

TCP – Reliable, Connection-oriented transport layer protocol
UDP – Unreliable, connectionless transport layer protocl





A **port** is a 16-bit number that is an entrance/exit point of a machine

An active port is associated with a specific process or service

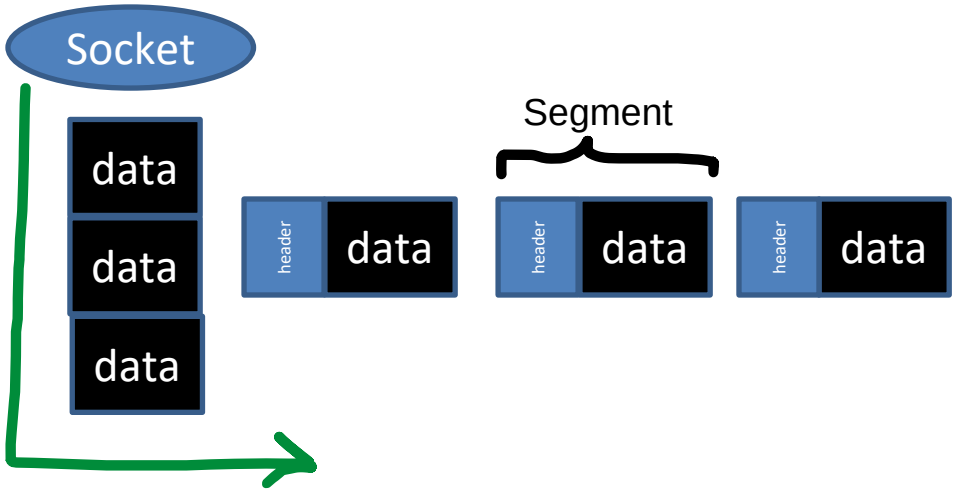
Why does UDP need a source port # ?

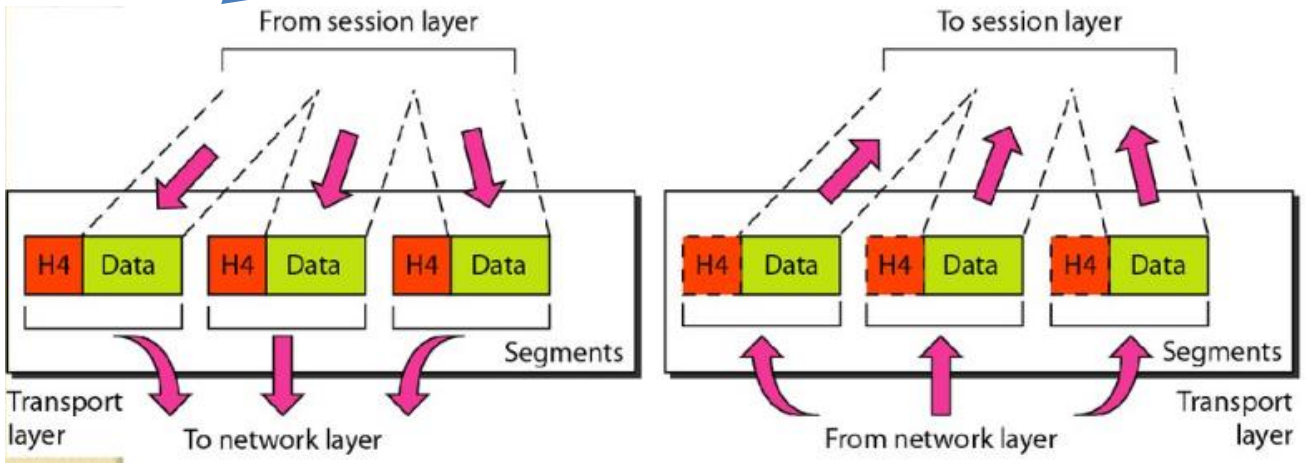
Network Layer

Transport Layer

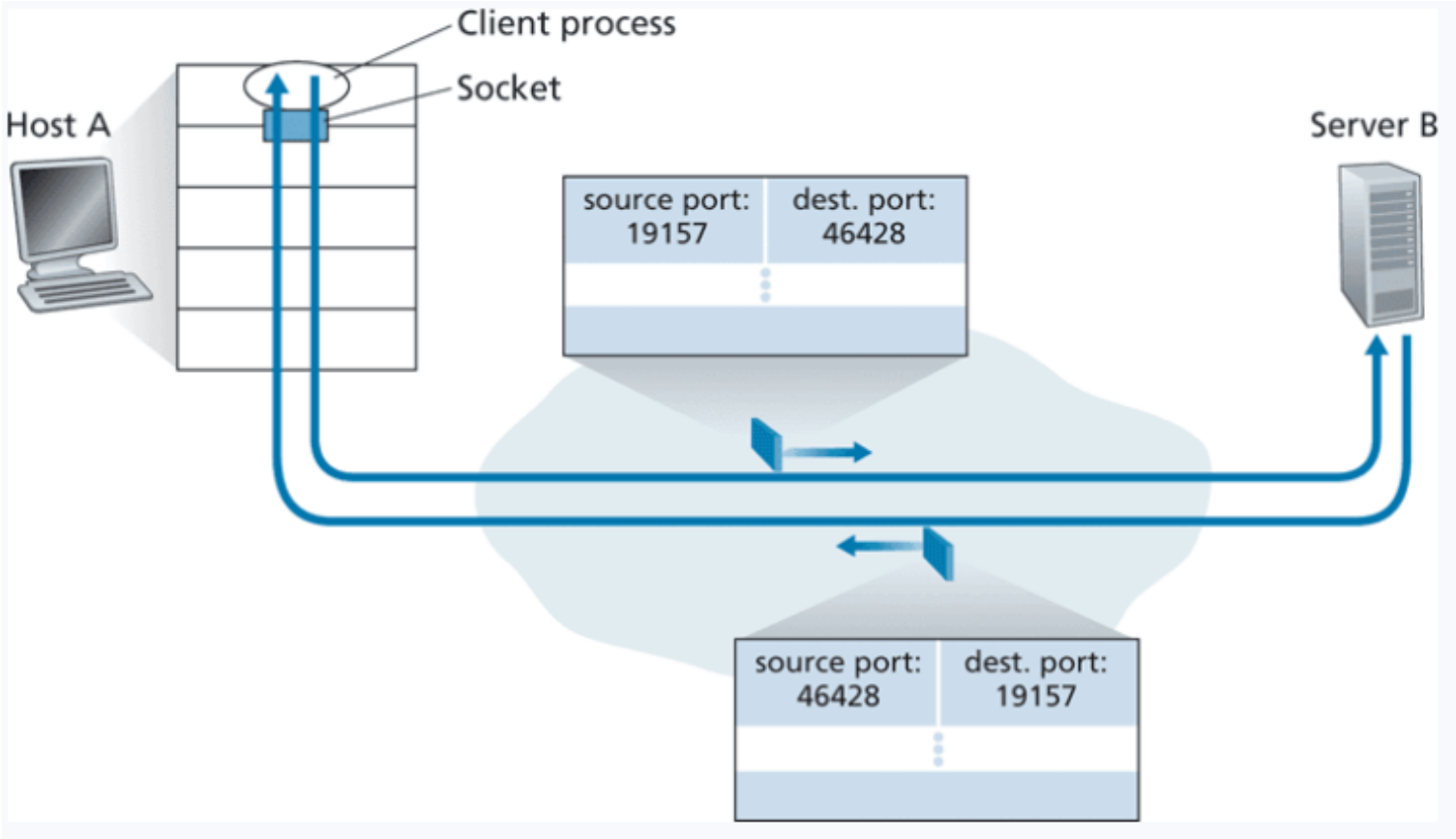
Multiplexing is the process of gathering chunks from sockets, encapsulating chunks with header information, and passing the segment into the network layer

Demultiplexing is the receiving segments from the transport layer and delivering the segment to the correct socket.





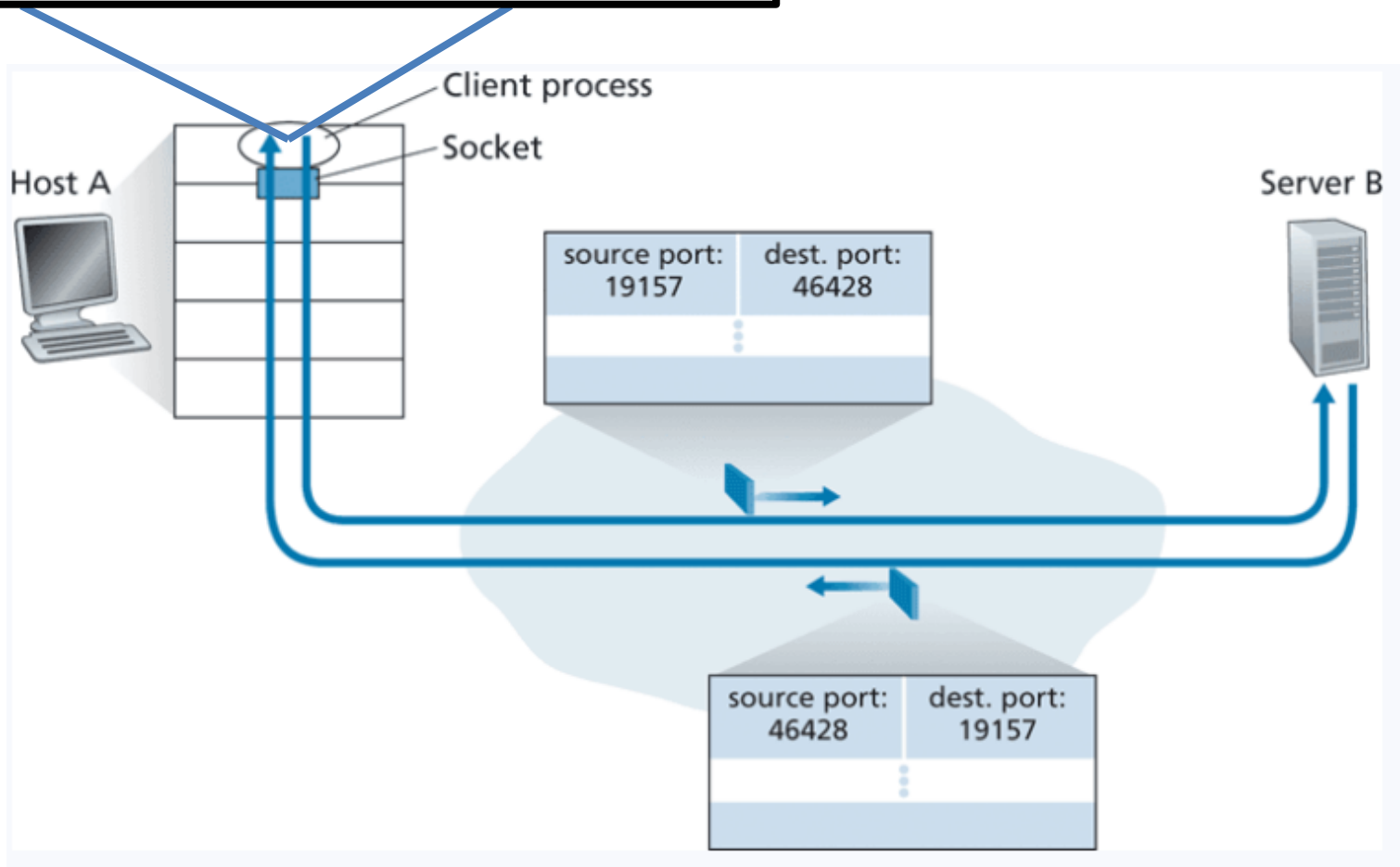
Transport Layer



Transport Layer

UDP sockets are identified by a two-tuple

(destination IP address, destination port number)

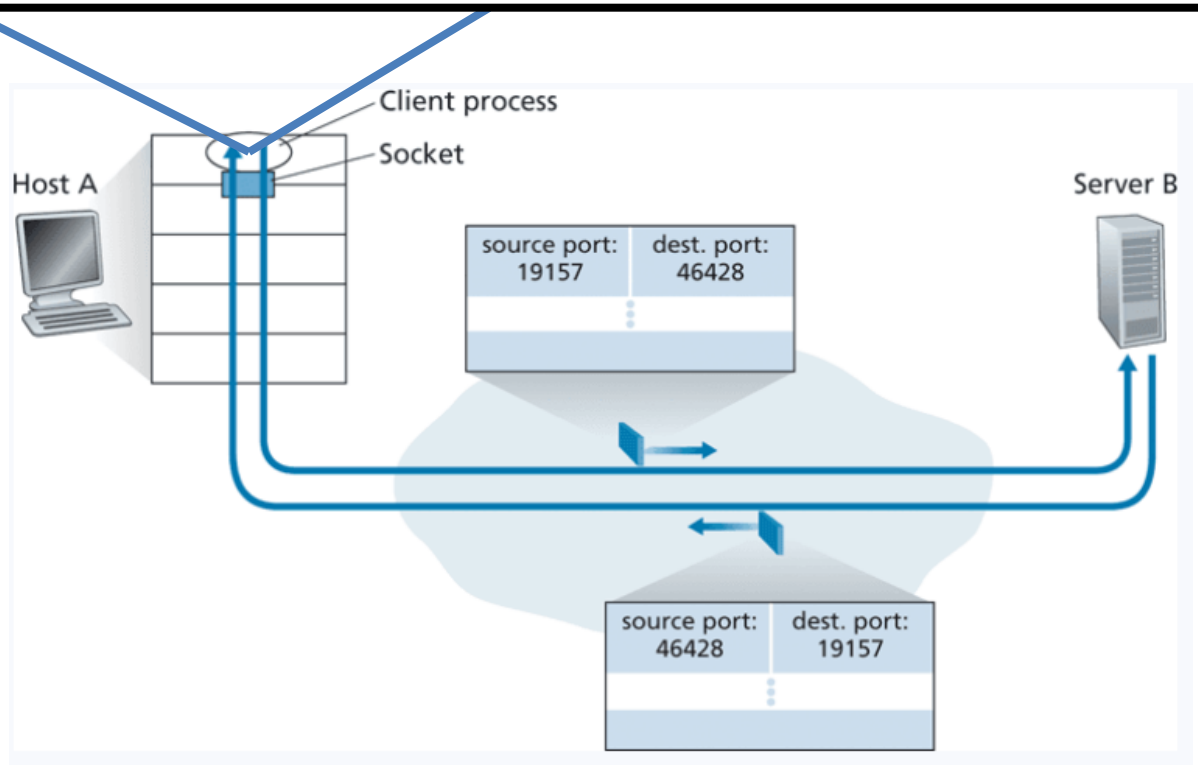


Transport Layer

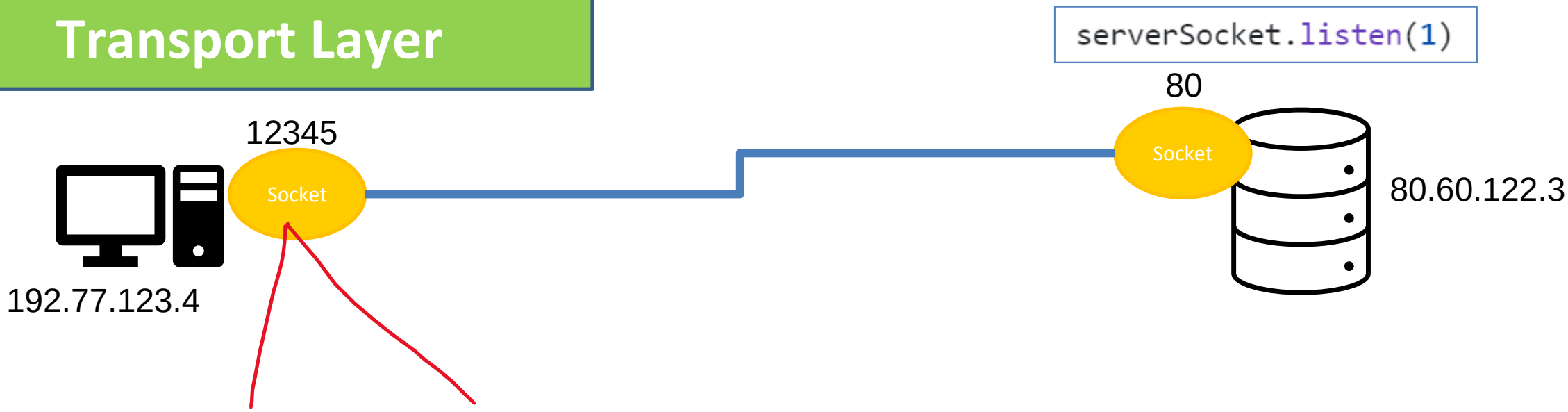
TCP sockets are identified by a four-tuple

A host will demultiplex segments using all of these values

(source IP address, source port number, destination IP address, destination port number)

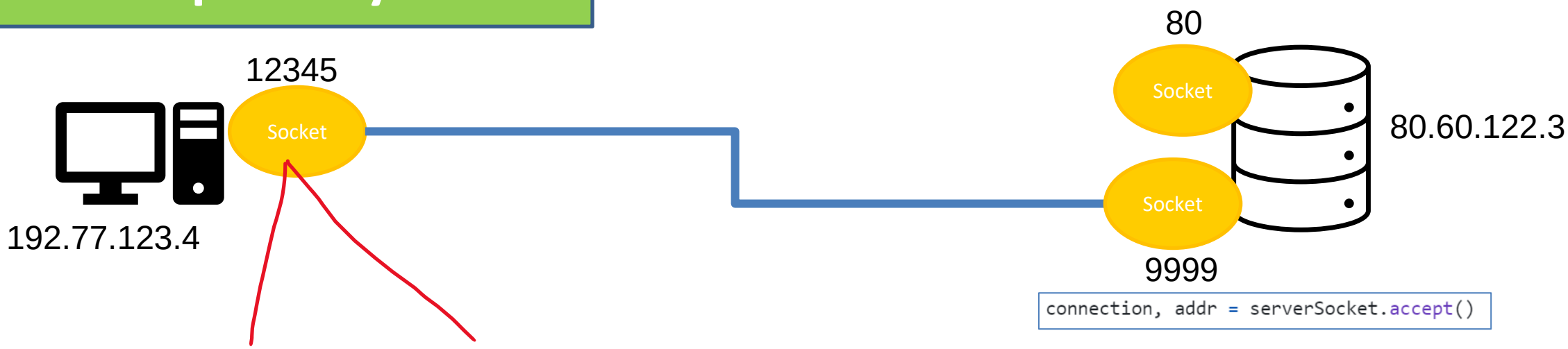


Transport Layer



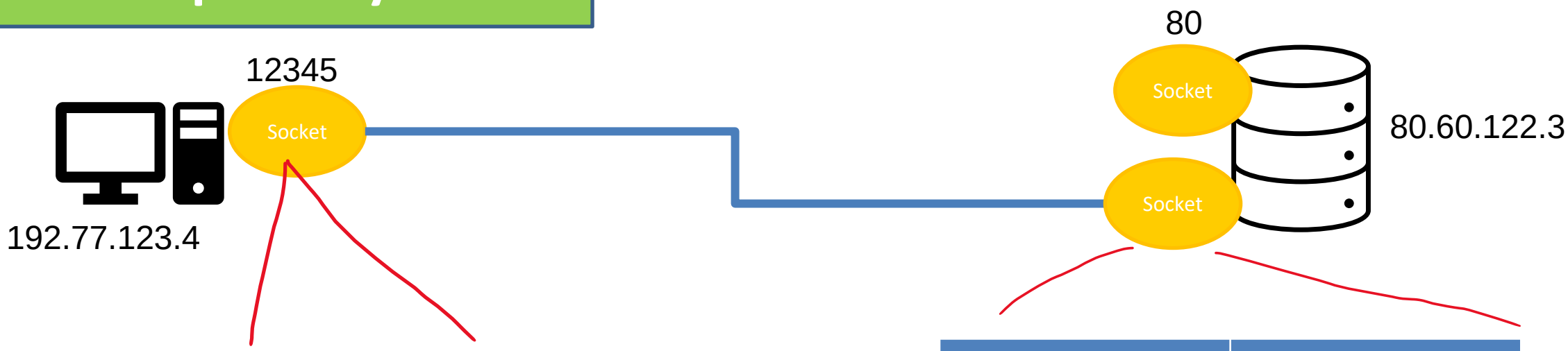
Name	Value
Src IP	192.77.123.4
Src Port	12345
Dst IP	80.60.122.3
Dst Port	80

Transport Layer



Name	Value
Src IP	192.77.123.4
Src Port	12345
Dst IP	80.60.122.3
Dst Port	80

Transport Layer

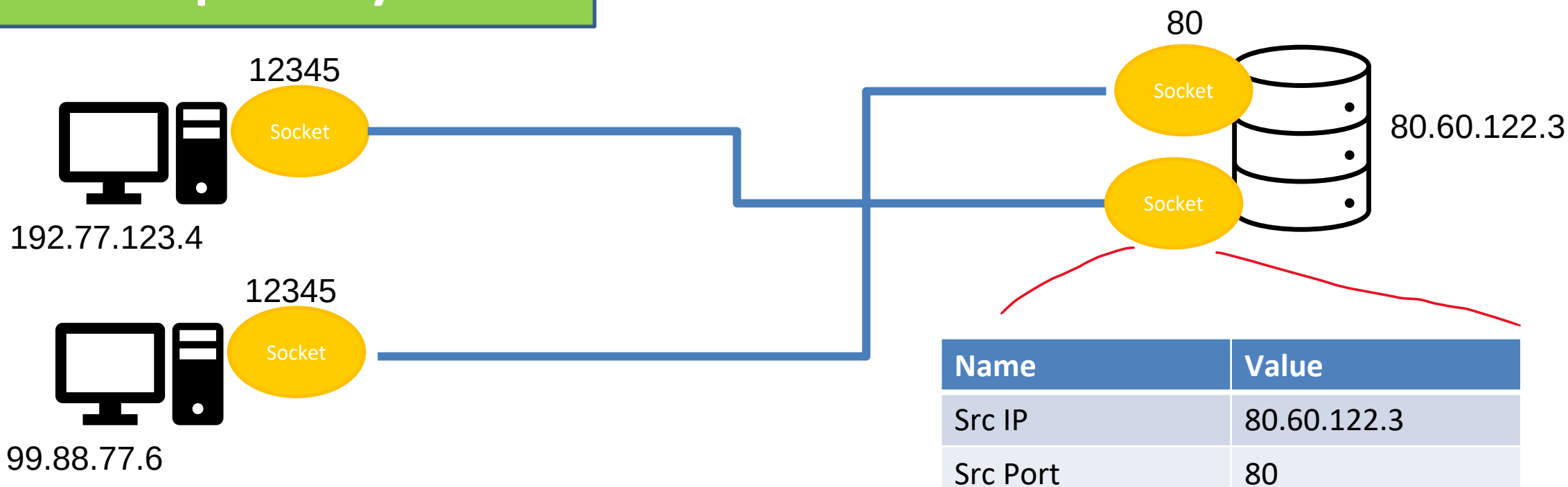


Name	Value
Src IP	192.77.123.4
Src Port	12345
Dst IP	80.60.122.3
Dst Port	80

Name	Value
Src IP	80.60.122.3
Src Port	80
Dst IP	192.77.123.4
Dst Port	12345

New socket, but isn't binded to a specific port like the listening socket

Transport Layer

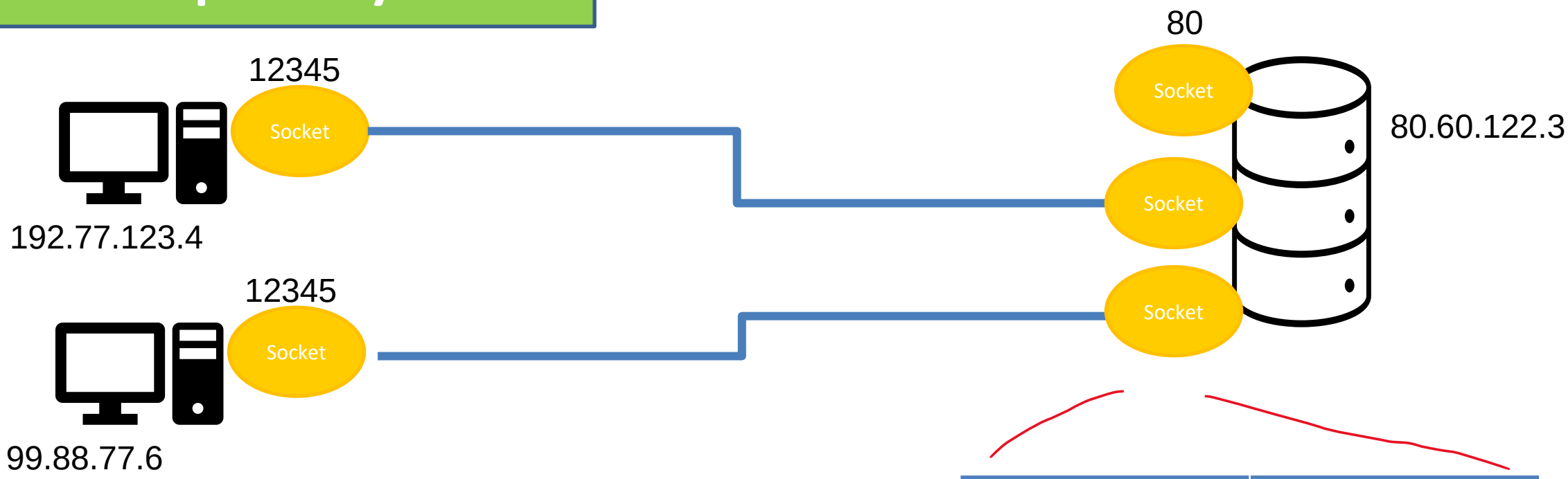


Name	Value
Src IP	99.88.77.6
Src Port	12345
Dst IP	80.60.122.3
Dst Port	80

Name	Value
Src IP	80.60.122.3
Src Port	80
Dst IP	192.77.123.4
Dst Port	12345

New socket, but isn't binded to a specific port like the listening socket

Transport Layer



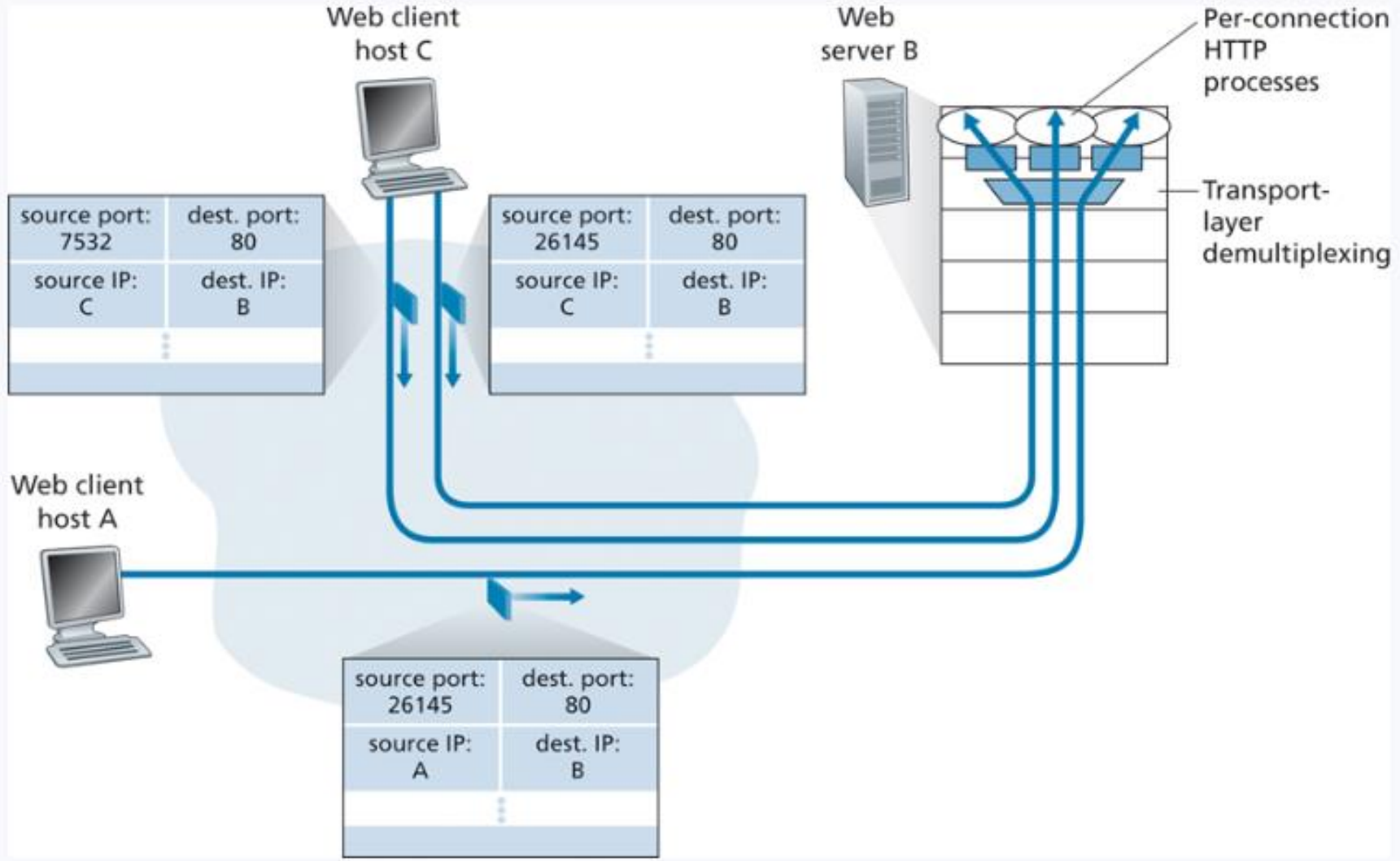
Name	Value
Src IP	99.88.77.6
Src Port	12345
Dst IP	80.60.122.3
Dst Port	80

Name	Value
Src IP	80.60.122.3
Src Port	80
Dst IP	99.88.77.6
Dst Port	12345

New socket, but isn't binded to a specific port like the listening socket

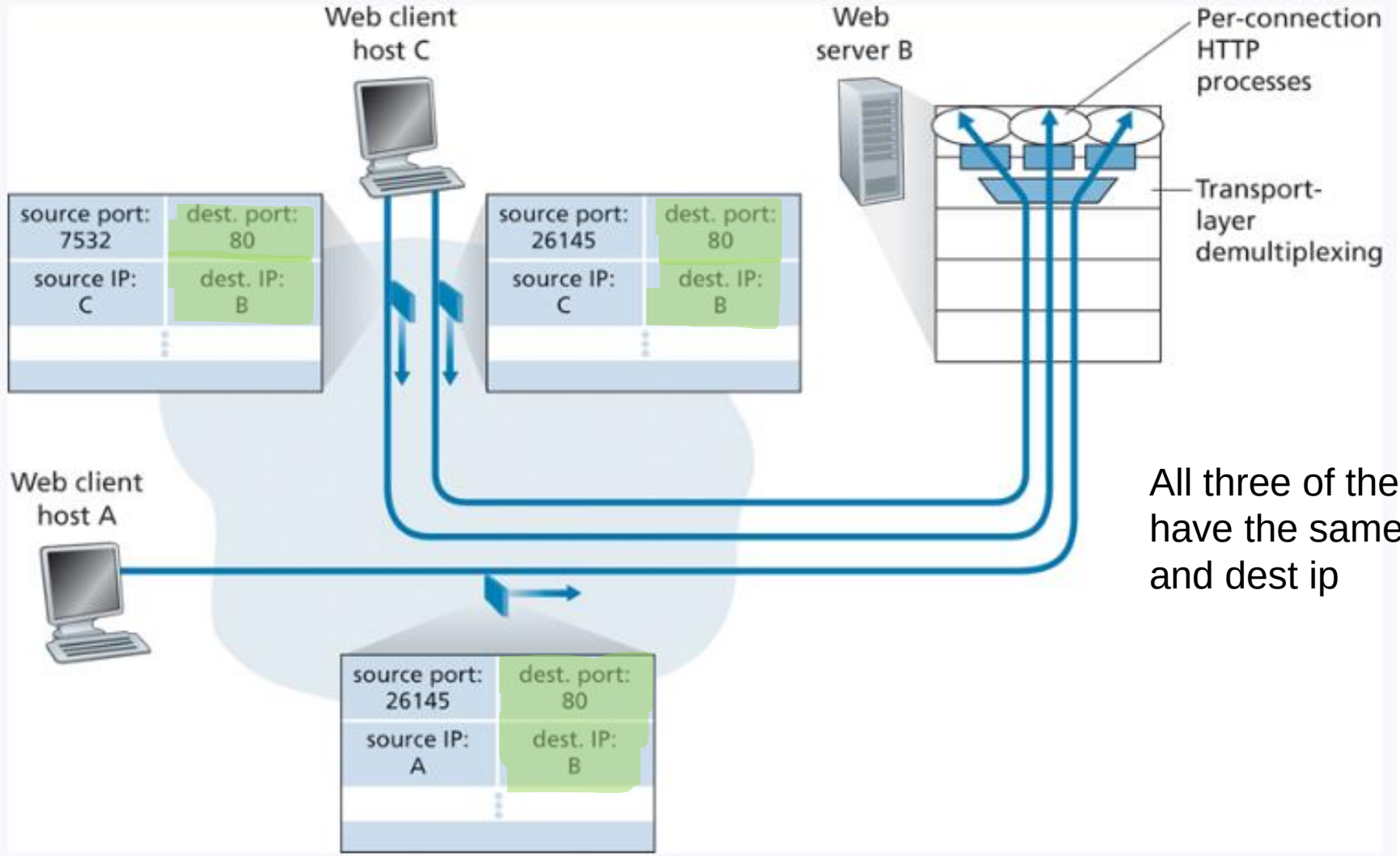
Transport Layer

Now that we are using four values, we can support simultaneous connections



Transport Layer

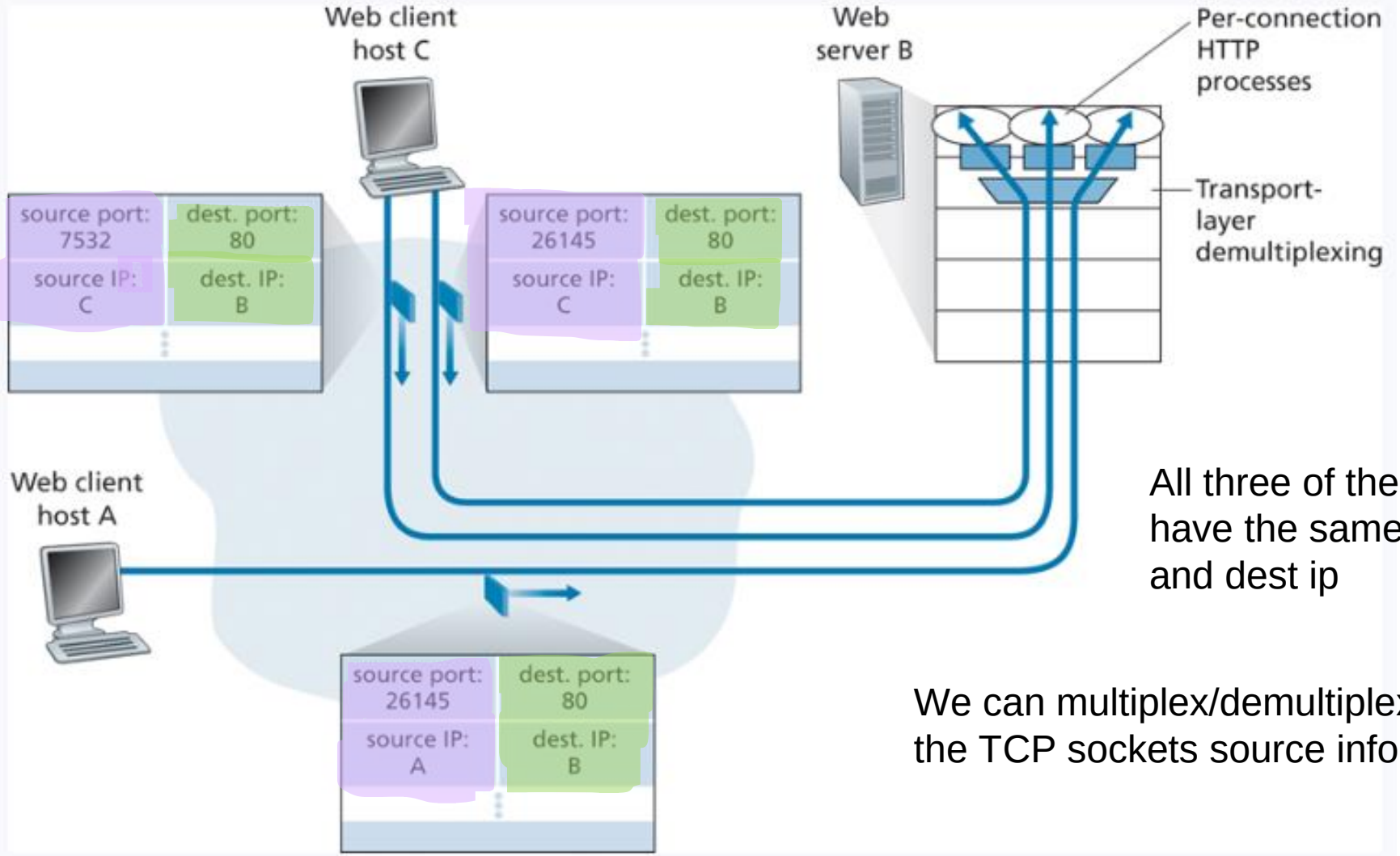
Now that we are using four values, we can support simultaneous connections



All three of these packets have the same dest port and dest ip

Transport Layer

Now that we are using four values, we can support simultaneous connections



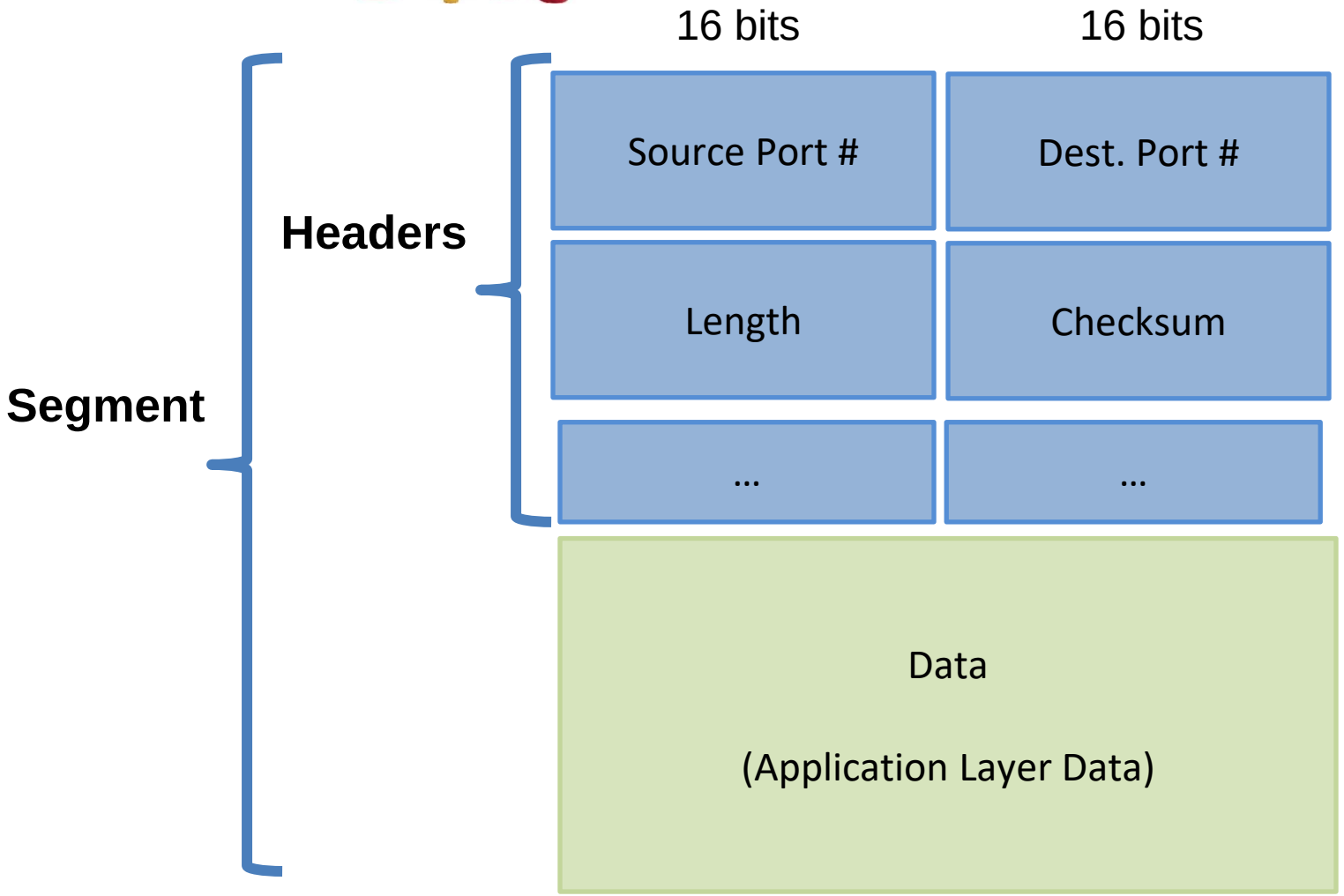
All three of these packets have the same dest port and dest ip

We can multiplex/demultiplex using the TCP sockets source information

Port Scanning with nmap

Transport Layer

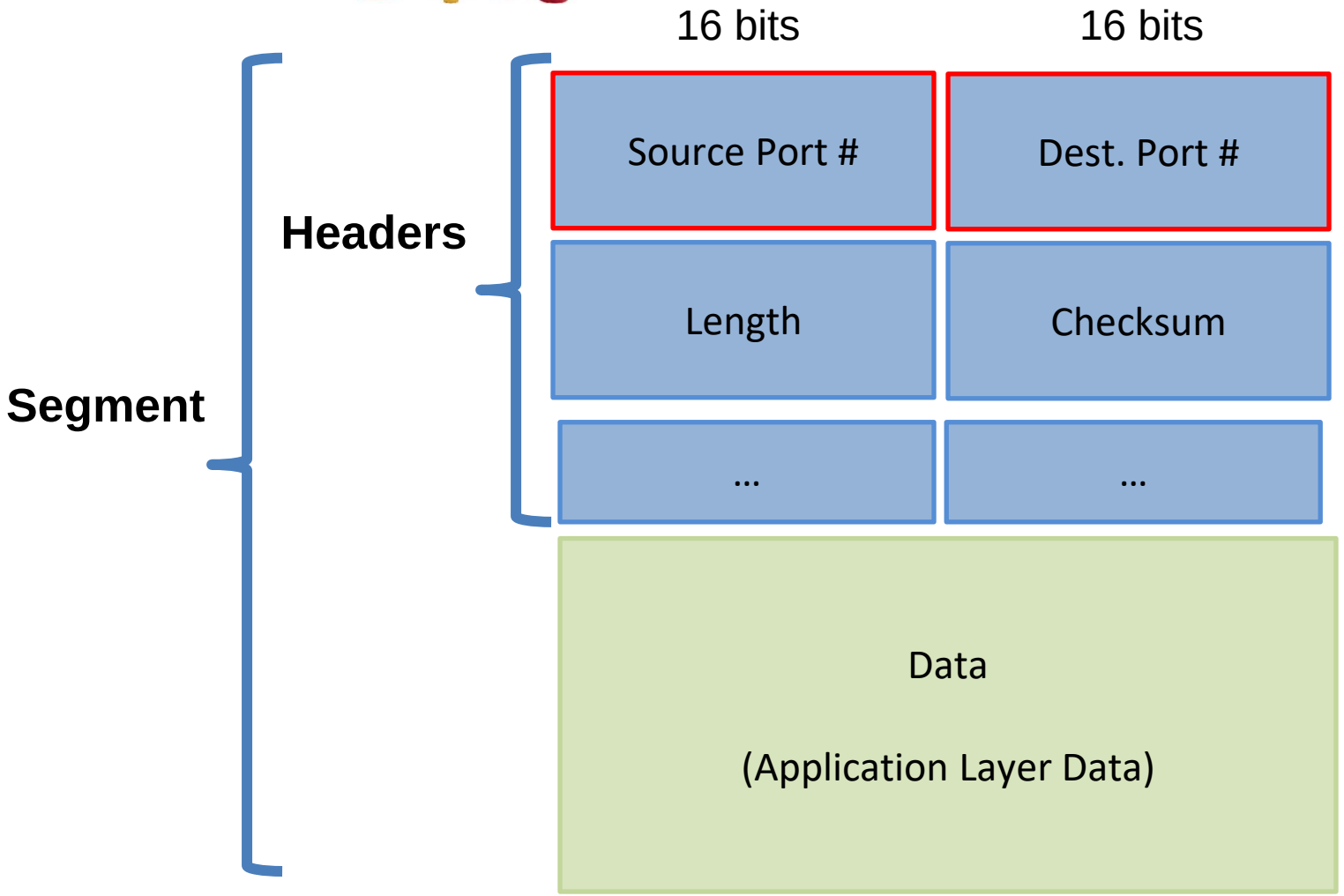
Our application layer message (HTTP request, DNS Query, FTP data) gets split into chunks, and each chunk is encapsulated in a **transport layer header**



Transport Layer

Our application layer message (HTTP request, DNS Query, FTP data) gets split into chunks, and each chunk is encapsulated in a **transport layer header**

Source port and Dest Port are attached to our packet

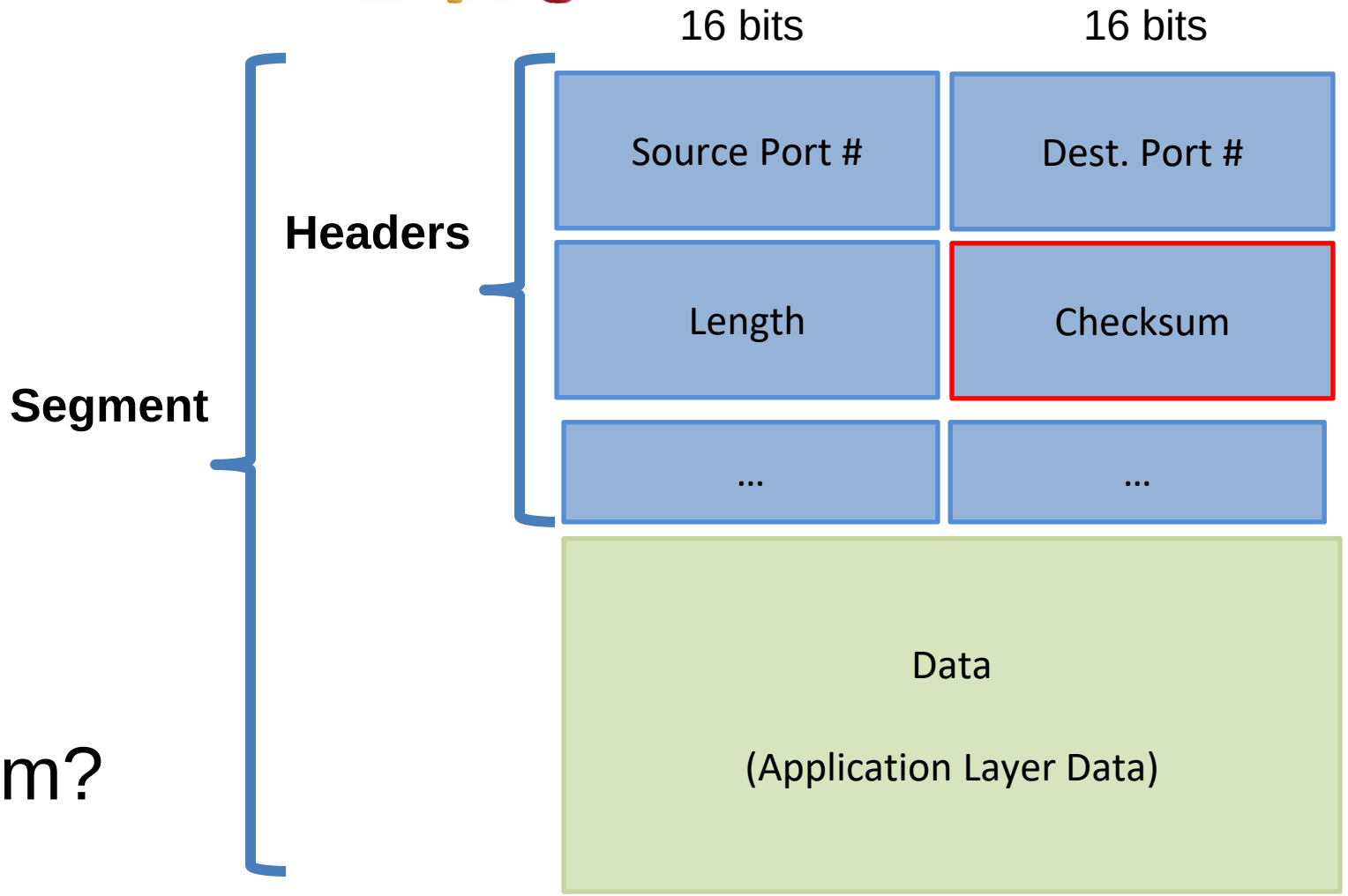


Transport Layer

Our application layer message (HTTP request, DNS Query, FTP data) gets split into chunks, and each chunk is encapsulated in a **transport layer header**

Source port and Dest Port are attached to our packet

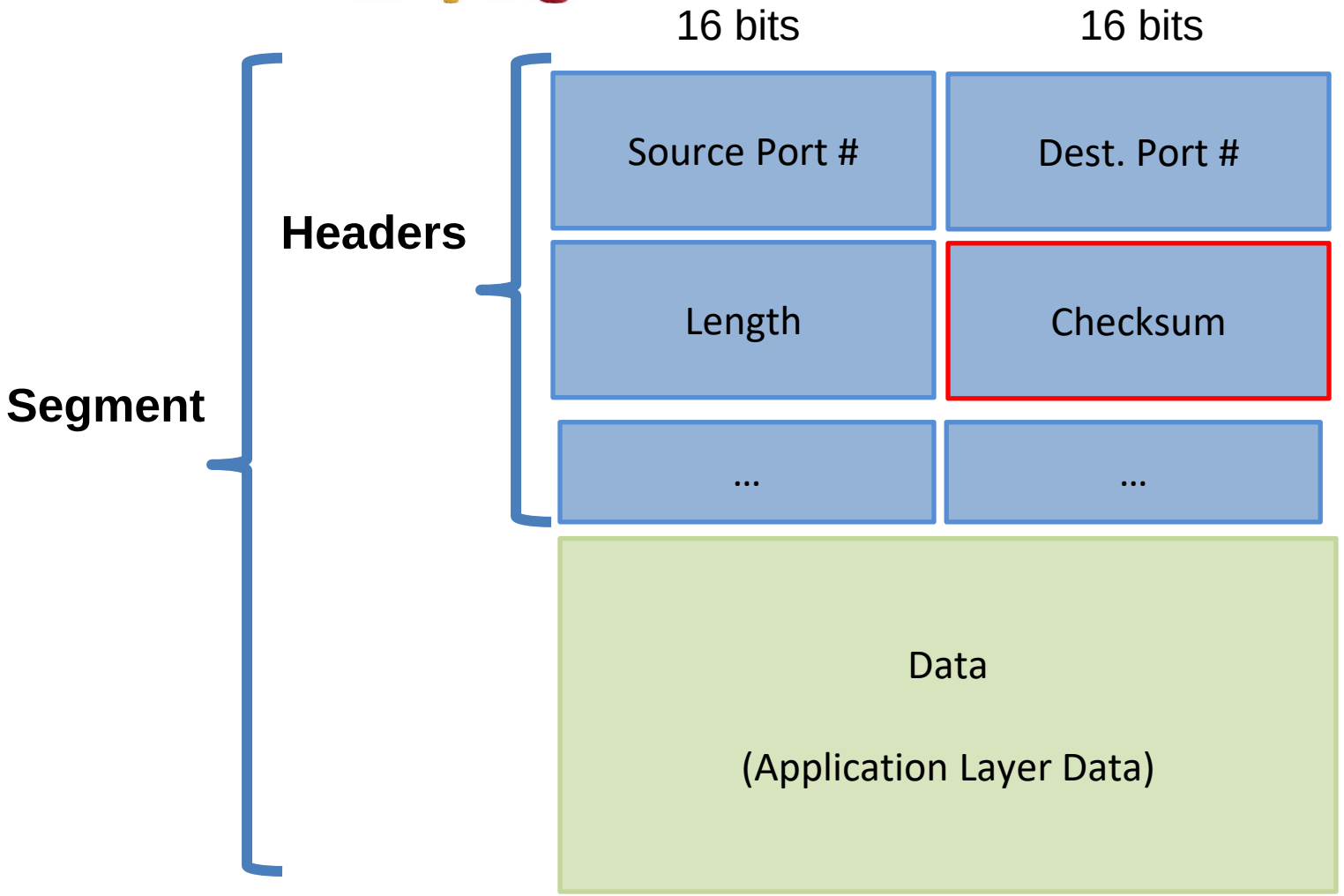
What is the checksum?



Transport Layer

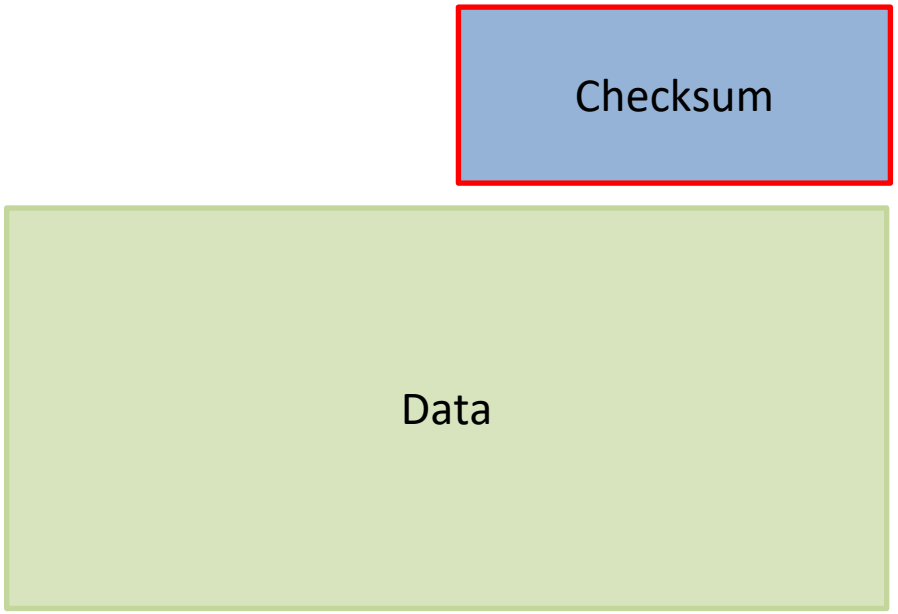
Important Services

- Flow Control
- Reliability
- Segmentation
- Error Checking
- Addressing



Transport Layer

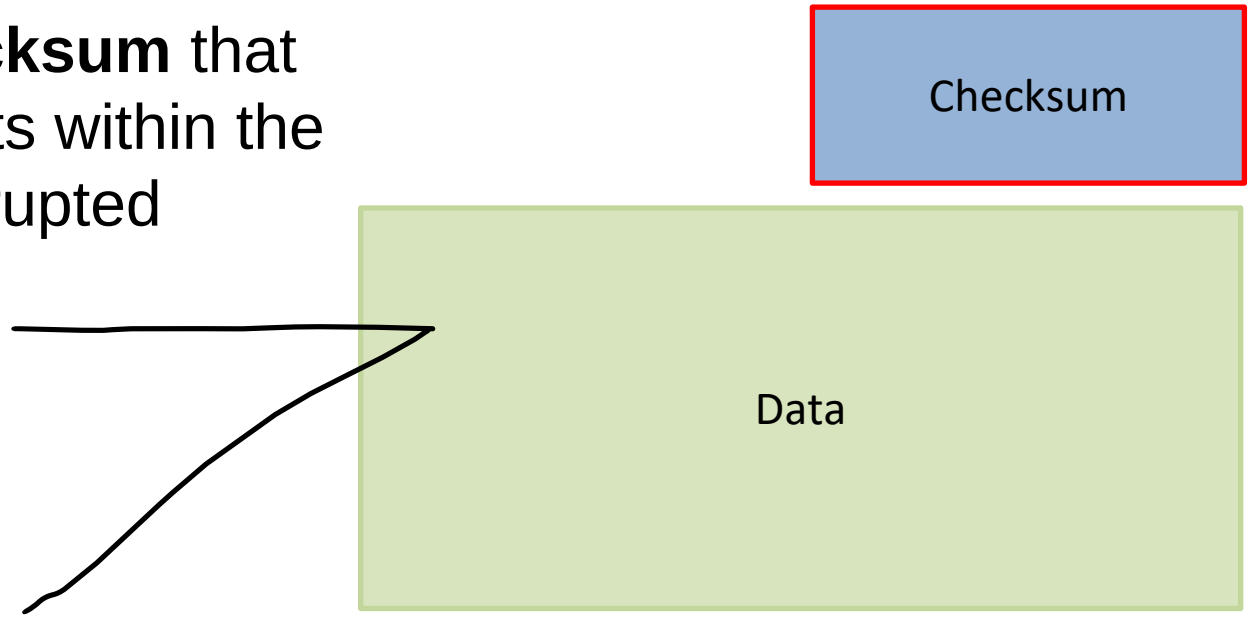
Transport layer provides a **checksum** that is used to determine whether bits within the segment have been altered/corrupted



Transport Layer

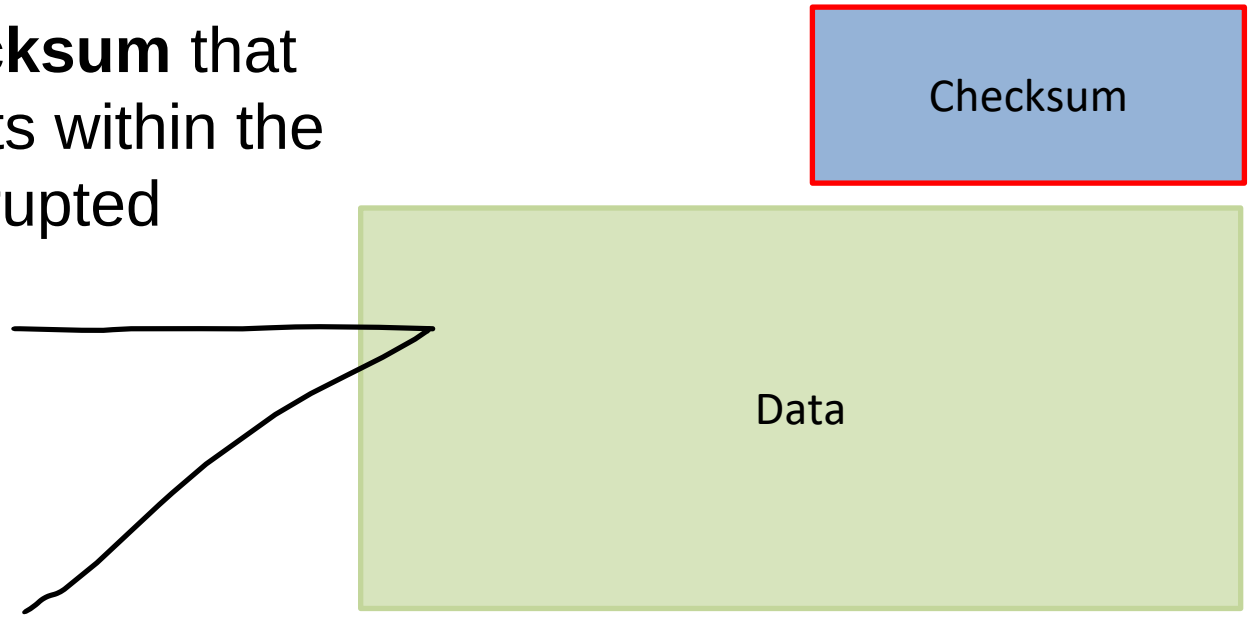
Transport layer provides a **checksum** that is used to determine whether bits within the segment have been altered/corrupted

0110011001100000
0101010101010101
1000111100001100



Transport Layer

Transport layer provides a **checksum** that is used to determine whether bits within the segment have been altered/corrupted



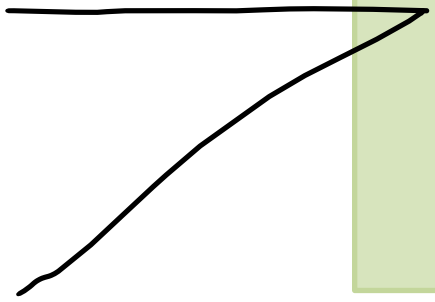
0110011001100000
0101010101010101
1000111100001100

Checksum is used a bit-level operation to see if any data has been corrutped

Transport Layer

Split up data into 16-bit “words”

0110011001100000
0101010101010101
1000111100001100



Checksum

0110011001100000
0101010101010101
1000111100001100

Checksum is used a bit-level operation to see if any data has been corrutped

Transport Layer

Split up data into 16-bit “words”
Sum up all words

```
  1
  0110011001100000
+ 0101010101010101
------------------
  1011101110110101
```

Checksum

```
 1
  0110011001100000
 2
  0101010101010101
 3
  1000111100001100
```

Checksum is used a bit-level operation to see if any data has been corrutped

Transport Layer

Split up data into 16-bit “words”
Sum up all words

```
  1
  0110011001100000
+ 0101010101010101
------------------
  1011101110110101
+ 1000111100001100
------------------
```

Checksum

```
 1
  0110011001100000
 2
  0101010101010101
 3
  1000111100001100
```

Checksum is used a bit-level operation to see if any data has been corrutped

Transport Layer

Split up data into 16-bit “words”
Sum up all words

```
  1
  0110011001100000
+ 0101010101010101
------------------
  1011101110110101
+ 1000111100001100
------------------
  0100101011000010
```

Checksum

```
 1
 0110011001100000
2
 0101010101010101
3
 1000111100001100
```

Checksum is used a bit-level operation to see if any data has been corrutped

Transport Layer

Split up data into 16-bit “words”
Sum up all words
Compute the ones compliment

Checksum

1 0110011001100000
2 0101010101010101
3 1000111100001100

```
  1 0110011001100000
+ 2 0101010101010101
  ----------------
  1011101110110101
+ 3 1000111100001100
  ----------------
  0100101011000010
```

Checksum is used a bit-level operation to see if any data has been corrutped

Transport Layer

Split up data into 16-bit “words”
Sum up all words
Compute the ones compliment

```
  1
  0110011001100000
+ 0101010101010101
------------------
  1011101110110101
+ 1000111100001100
------------------
  0100101011000010
+                1
------------------
  1011010100111101
```

Checksum

```
 1
  0110011001100000
2 0101010101010101
3 1000111100001100
```

This is our checksum value!

Transport Layer

Split up data into 16-bit “words”
Sum up all words
Compute the ones compliment

```
  1
  0110011001100000
+ 0101010101010101
  ----------------
  1011101110110101
+ 1000111100001100
  ----------------
  0100101011000010
+ 1
  ----------------
  1011010100111101
```

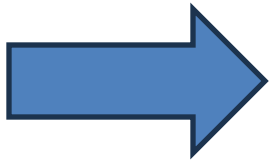
1011010100111101

```
1 0110011001100000
2 0101010101010101
3 1000111100001100
```

This is our checksum value!

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

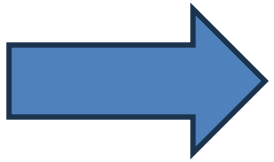


Send to receiver

1 0110011001100000
2 0101010101010101
3 1000111100001100

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…



Send to receiver

1 0110011001100000
2 0101010101010101
3 1000111100001100

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

```
  1
+ 0110011001100000
  2
  0101010101010101
  ----------------
  1011101110110101
+ 3
  1000111100001100
  ----------------
  0100101011000010
+ C
  1011010100111101
  ----------------
```

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

```
  1 0110011001100000
+ 2 0101010101010101
  ----------------
    1011101110110101
+ 3 1000111100001100
  ----------------
    0100101011000010
+ C 1011010100111101
  ----------------

    1111111111111111
```

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

```
  1
  0110011001100000
+ 2
  0101010101010101
  ----------------
  1011101110110101
+ 3
  1000111100001100
  ----------------
  0100010011000010
+ C
  1011010100111101
  ----------------
  1111111111111111
```

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

```
  1 0110011001100000
+ 2 0101010101010101
  ----------------
  1011101110110101
+ 3 1000111100001100
  ----------------
  0100101011000010
+ C 1011010100111101
  ----------------

  1111111111111111
```

If the result of this operation is **not** all 1s, then data **must** have been corrupted in the segment

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

```
  1 0110011001100000
+ 2 0101010101010101
  ----------------
    1011101110110101
+ 3 1000111100001100
  ----------------
    0100101011000010
+ C 1011010100111101
  ----------------

  X 1111111111111111
```

If the result of this operation is **not** all 1s, then data **must** have been corrupted in the segment

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

```
  1 0110011001100000
+ 2 0101010101010101
  ------------------
    1011101110110101
+ 3 1000111100001100
  ------------------
    0100101011000010
+ C 1011010100111101
  ------------------

  X 1111111111111111
```

The receiver will do that for each packet, if the checksum is invalid, it may ask for a retransmission

Transport Layer

Source Port #	Dest. Port #
Length	1011010100111101
…	…

1 0110011001100000
2 0101010101010101
3 1000111100001100

Send to receiver

When the receiver gets the segment, it will compute the sum of all four 16-bit words (including the checksum)

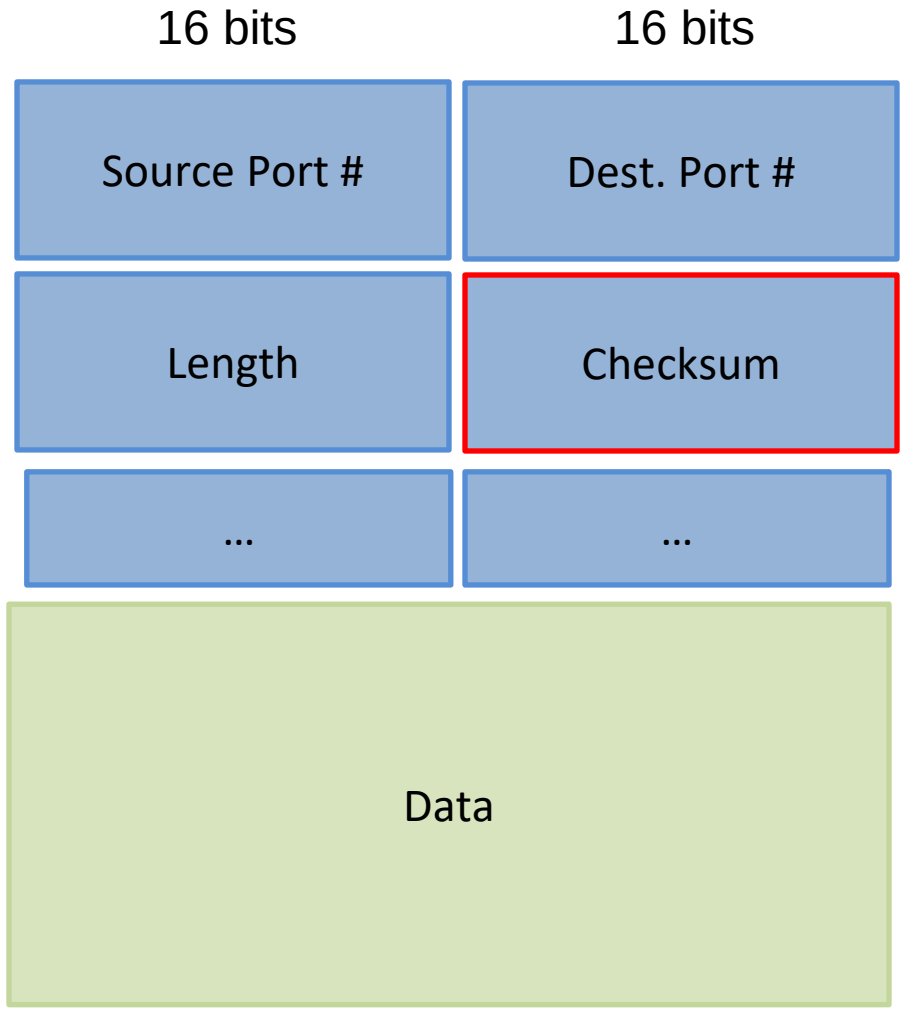
```
  1 0110011001100000
+ 2 0101010101010101
  ------------------
    1011101110110101
+ 3 1000111100001100
  ------------------
    0100101011000010
+ C 1011010100111101
  ------------------

  X 1111111111111111
```

The receiver will do that for each packet, if the checksum is invalid, it may ask for a retransmission

Transport Layer

Why do error checking here?

Protocols at other layers (link layer) can also do error checking

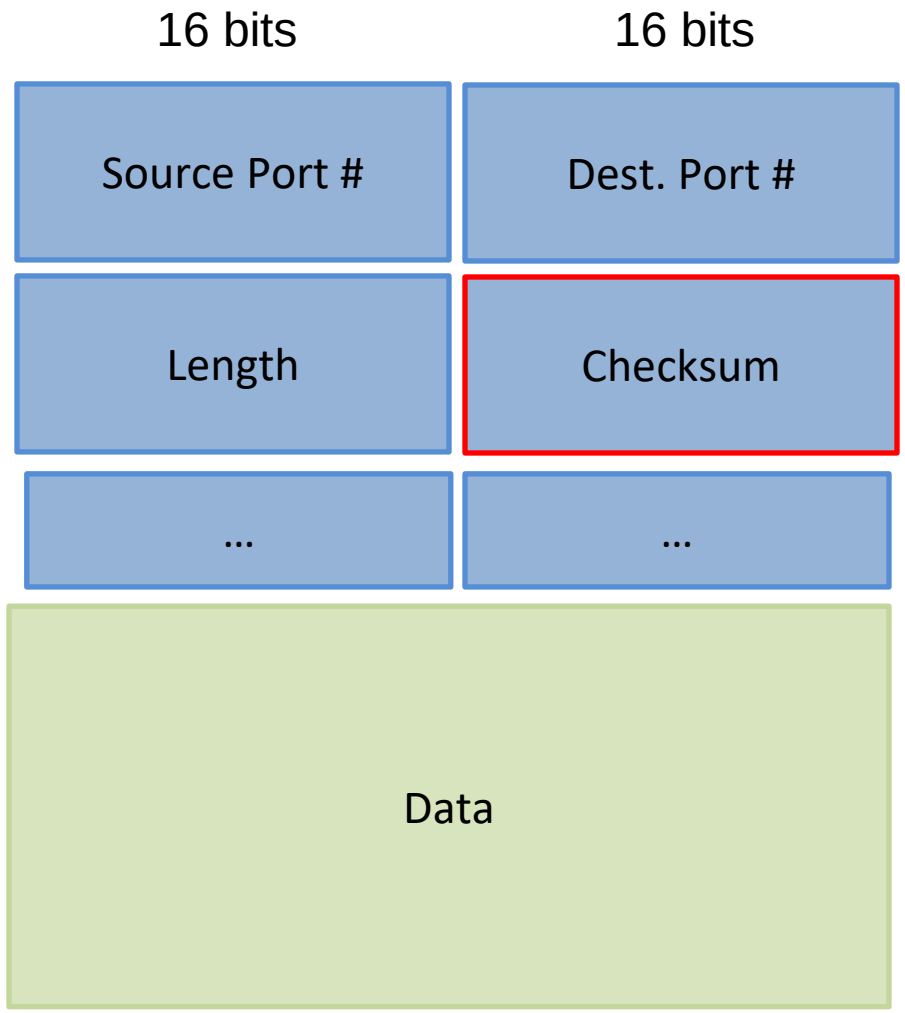


Transport Layer

Why do error checking here?

Protocols at other layers (link layer) can also do error checking

However, there is no guarantee that our packet of information will use these protocols



Transport Layer

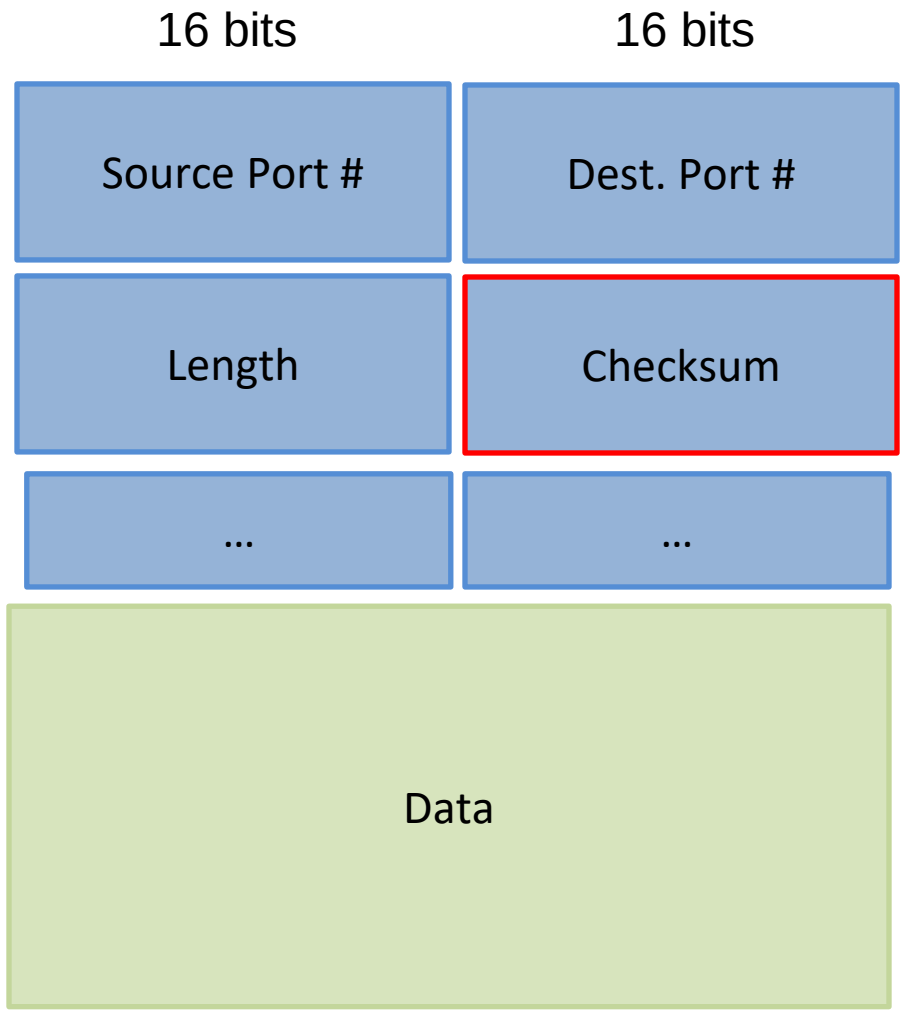
Why do error checking here?

Protocols at other layers (link layer) can also do error checking

However, there is no guarantee that our packet of information will use these protocols

End-to-end principle states that since certain functionality such as error detection, must be implemented on an end-end bases

Functionality places at the lower levels may be redundant or of little value when compared to the cost of providing them at a higher level



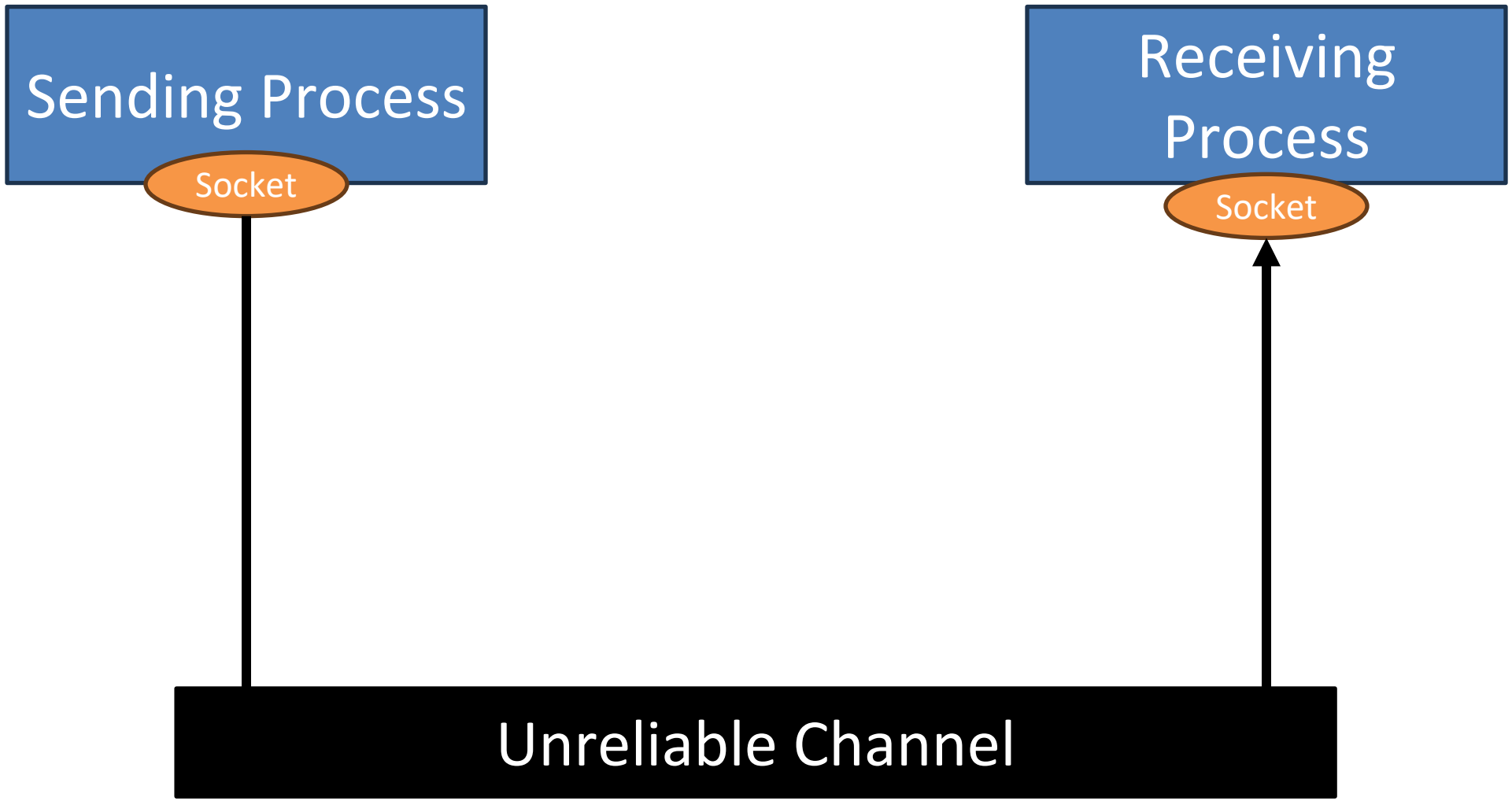
Transport Layer

Important Services

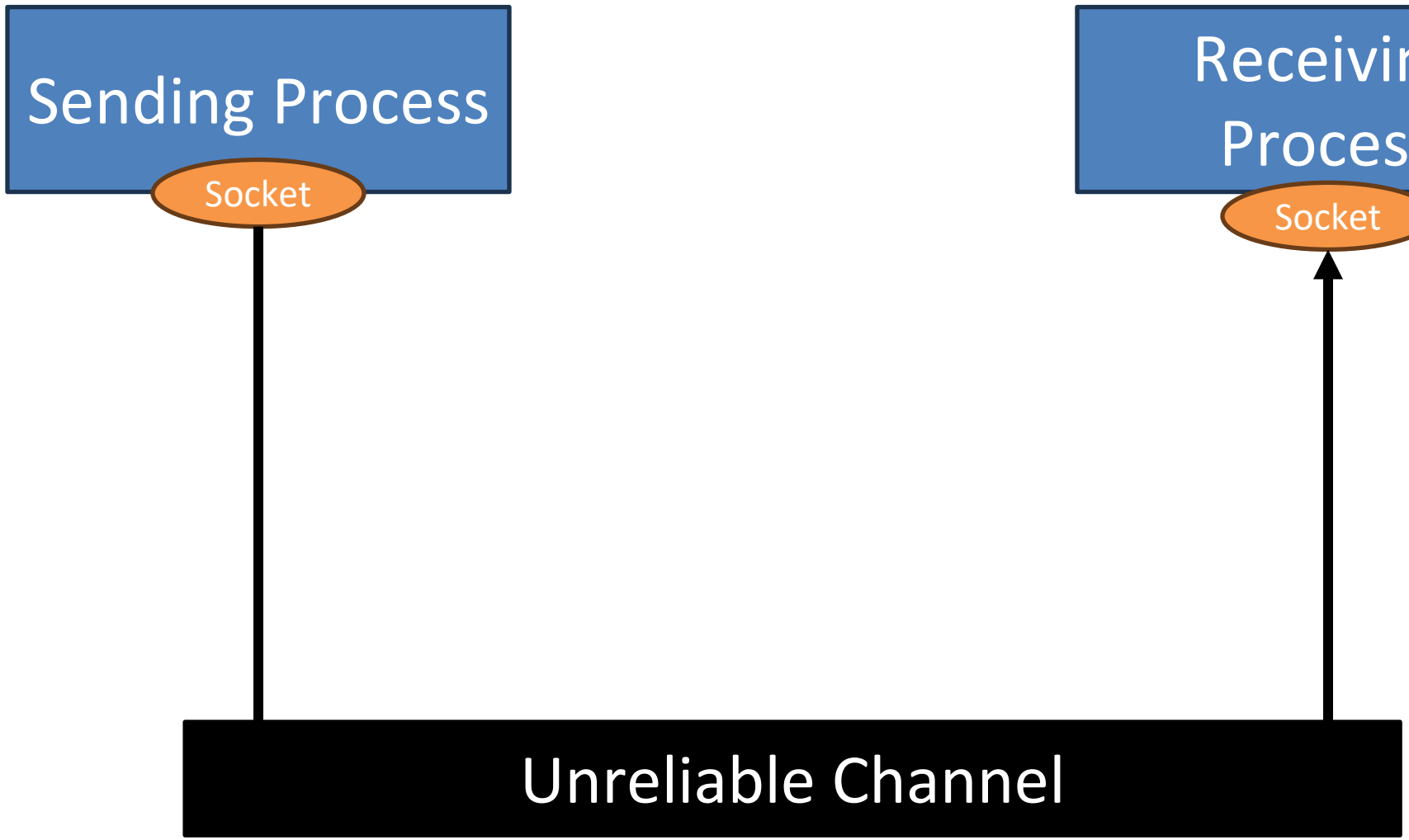
- Flow Control
- Reliability
- Segmentation
- Error Checking
- Addressing

Transport Layer

Transport Layer



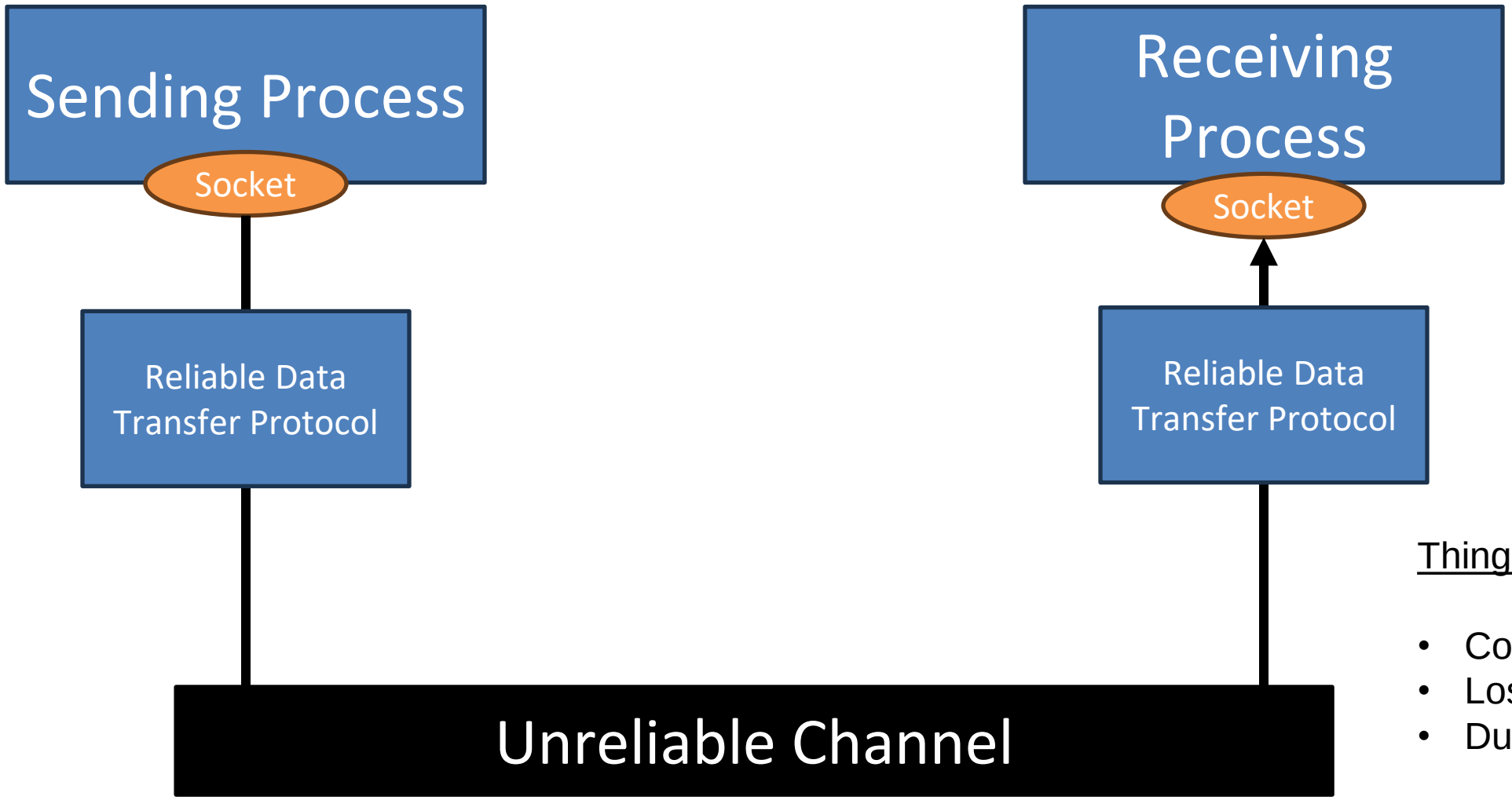
Transport Layer



Things that may happen

- Corruption
- Loss of packets
- Duplicate Delivery

Transport Layer



Things that may happen

- Corruption
- Loss of packets
- Duplicate Delivery

Transport Layer