

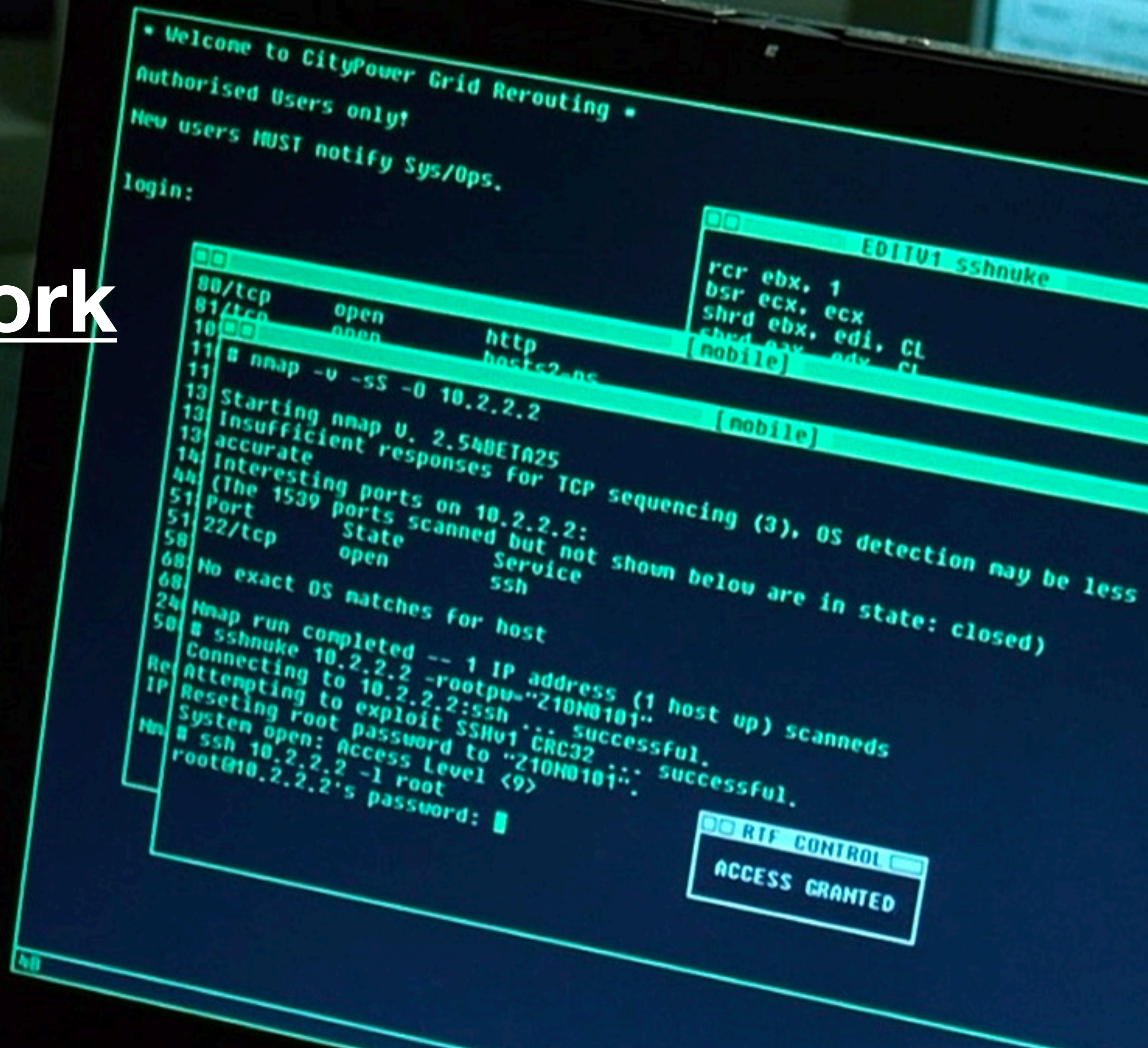
# Computer Network

## Security

ECE 4112/6612

CS 4262/6262

Prof. Frank Li





# Logistics

HW1 regrade requests should be made by **next Tuesday (Sept 26)**

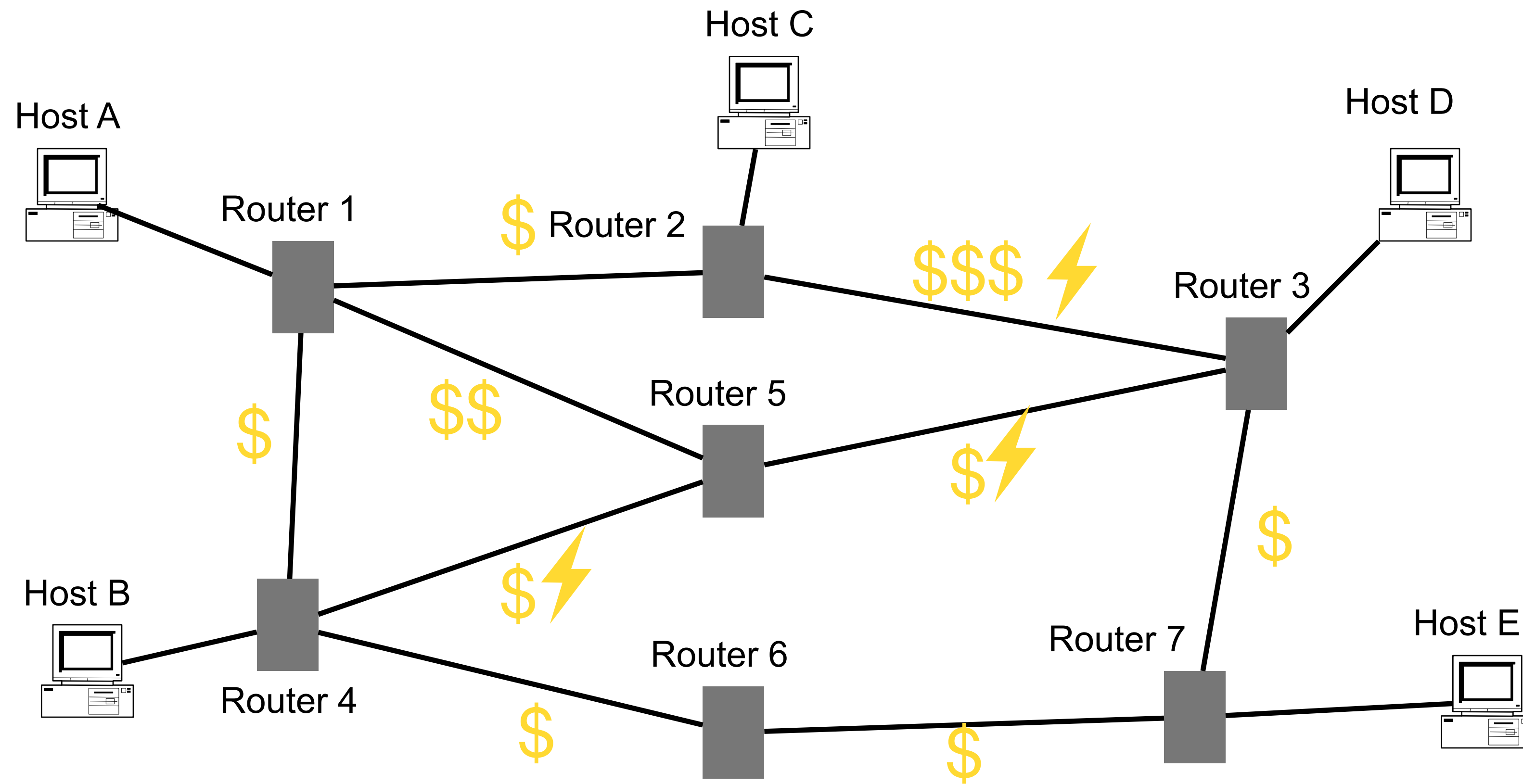
Quiz 1 done! Being graded now (regrades will be available afterwards)

Project proposals due next Friday!

# Finishing up with Routing

# Routing

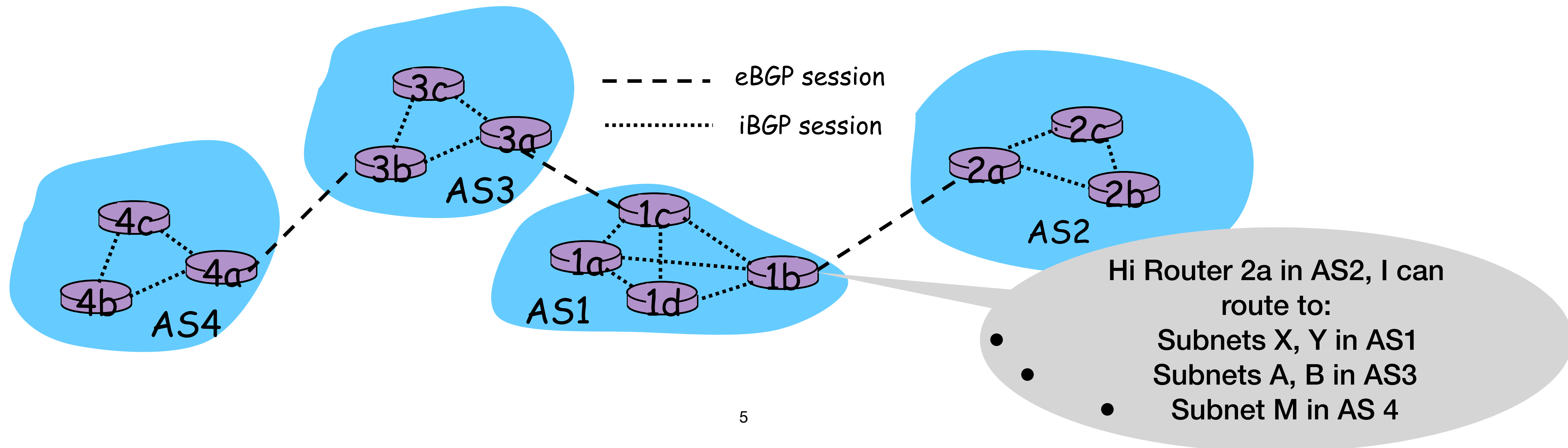
Host A wants to communicate with Host D. How do you figure out which route to take?



# Routing via BGP

**Border Gateway Protocol (BGP):** Allows networks, called Autonomous Systems (AS), to advertise reachability and route attribute information (e.g., which subnets is an AS willing to deliver traffic to). *Note: BGP is a TCP protocol (so application layer), but routing is a network layer task.*

- Within a network/AS, iBGP is run between routers
- Between networks/ASes, eBGP is run between border routers (BGP peers)



# BGP Route Selection

A router may learn multiple routes to the same subnet/prefix. How to decide which route to select? Simplified decision process (in order of preference):

1. Route preference value (e.g. peering vs \$\$\$)
2. Shortest AS-PATH
3. Closest NEXT-HOP (e.g., round-trip time)
4. Further tie breakers

When routing to a specific IP address, a router finds the most specific subnet/prefix it can route to based on *longest prefix match*, and uses the selected route for that prefix.

Example: Destination = 108.0.1.2

Routing Table:

| Prefix       | Route AS-PATH  |
|--------------|----------------|
| 108.0.0.0/8  | AS2, AS17      |
| 108.0.1.0/24 | AS3, AS18, AS4 |

# BGP Security Issue

An AS can announce (either maliciously or by accident) invalid/incorrect routes, potentially redirecting traffic to a destination. This is called BGP hijacking.

- Requires other ASes to accept the announced route, but this can often happen by announcing a very specific route (e.g., long prefix).

# BGP Hijacking

RYAN SINGEL

SECURITY 02.25.2008 10:37 AM

## Pakistan's Accidental YouTube Re-Routing Exposes Trust Flaw in Net

The Pakistani government ordered ISPs to censor YouTube to prevent Pakistanis from seeing a trailer to an anti-Islamic film by Dutch politician Geert Wilders. YouTube has since removed the clip for violating its terms of service, but a screenshot of the film, available via Google, shows a crude drawing of a pig defecating with the word Allah underneath it.

Pakistan Telecom complied by changing the **BGP** entry for YouTube -- essentially updating its local internet address book for where YouTube's section of the internet is. The idea was to direct its internet users to a page that said YouTube was blocked.

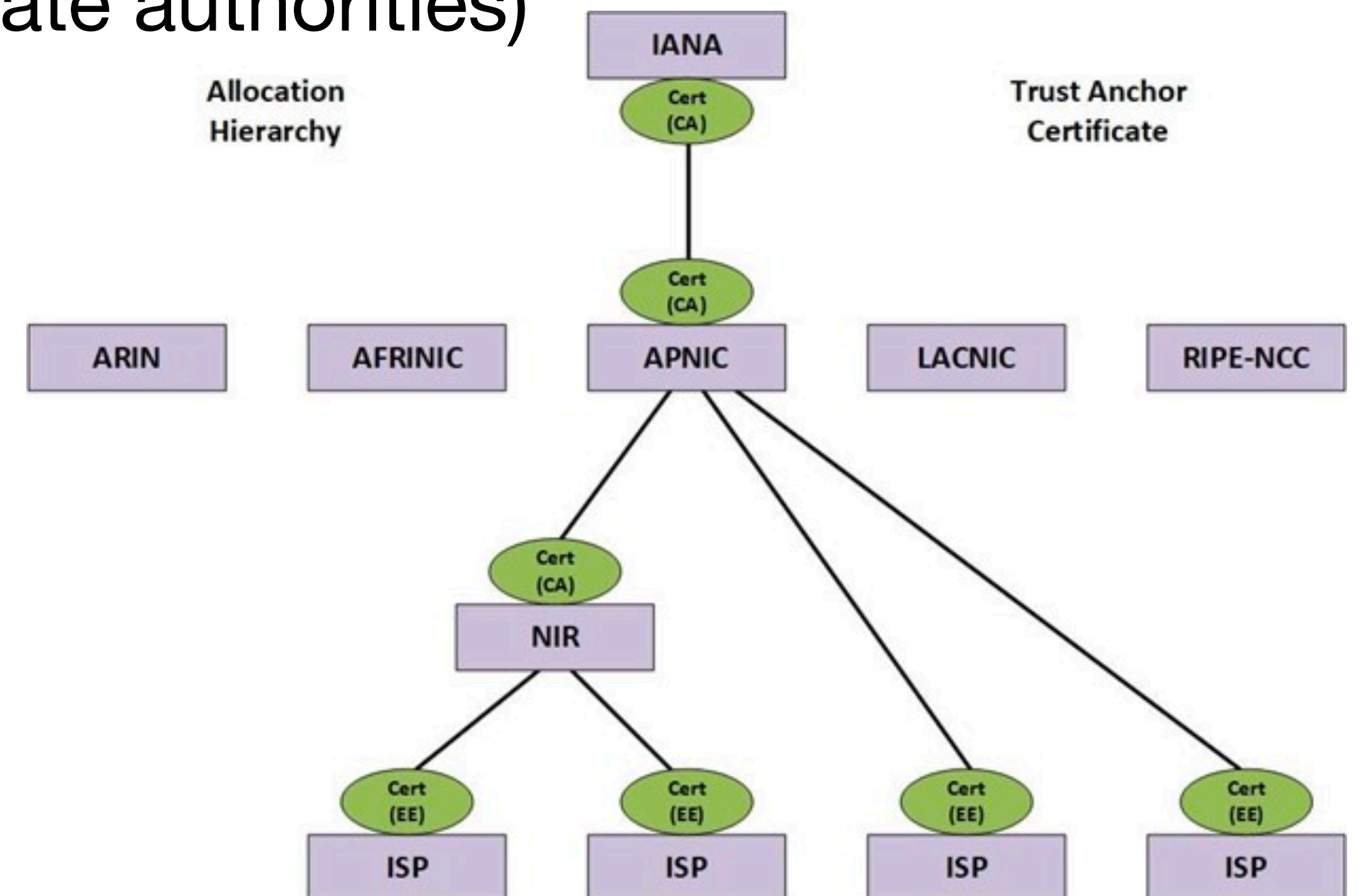
Unfortunately, the ISP announced the new route to upstream providers. The upstream providers didn't verify the new route but accepted it and then passed it along, cascading the bad address around the net, until most everyone using the net on Sunday would have been directed to the Pakistani's network block. The blunder not only took down YouTube, but also choked the Pakistani ISP, which was quickly deluged with millions of requests for talking cat videos.



# BGP Defense

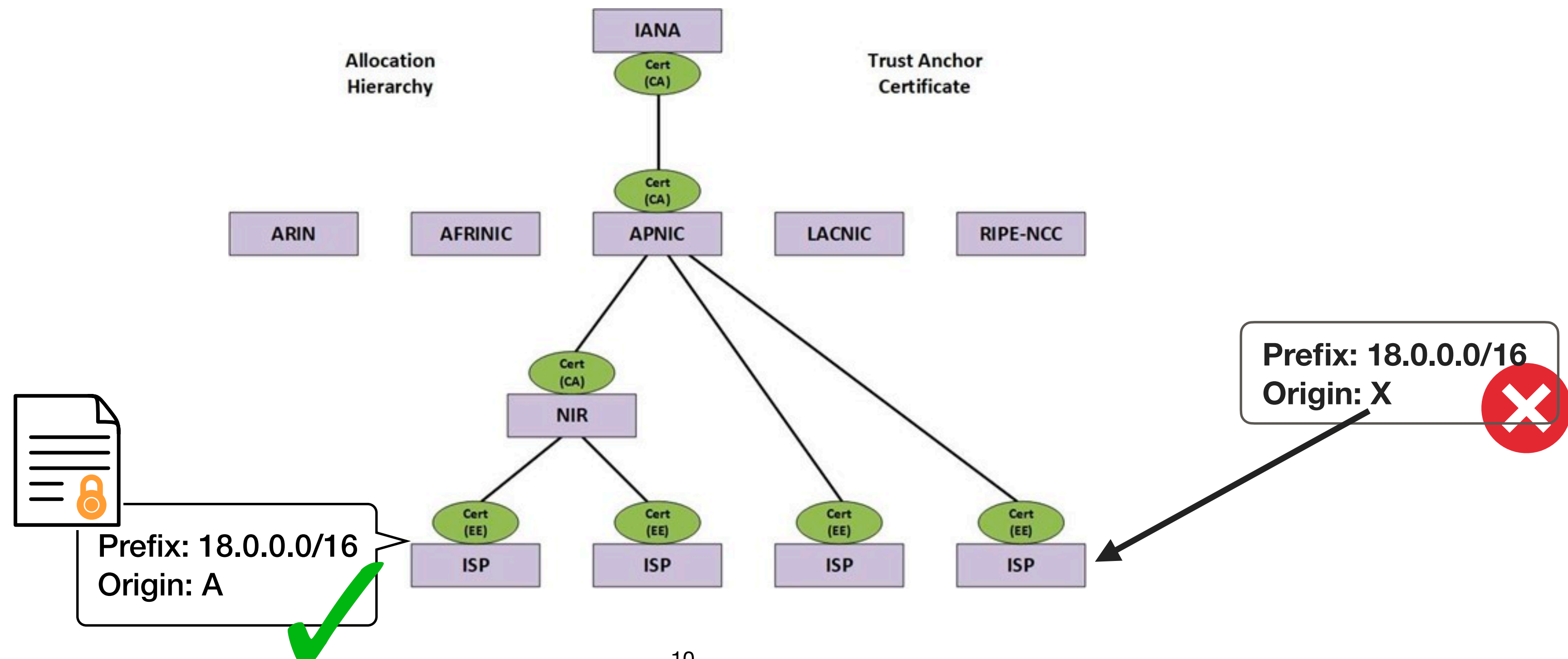
Internet Assigned Numbers Authority (IANA) and the 5 Regional Internet Registries (RIR) decide which IP addresses belong to each AS.

**Resource PKI (RPKI):** With RPKI, IANA + RIRs + ASes have public-key pairs. RIRs sign certificates for ASes binding AS-owned subnets/prefixes to the AS public keys (i.e., IANA + RIRs = root certificate authorities)



# BGP Defense

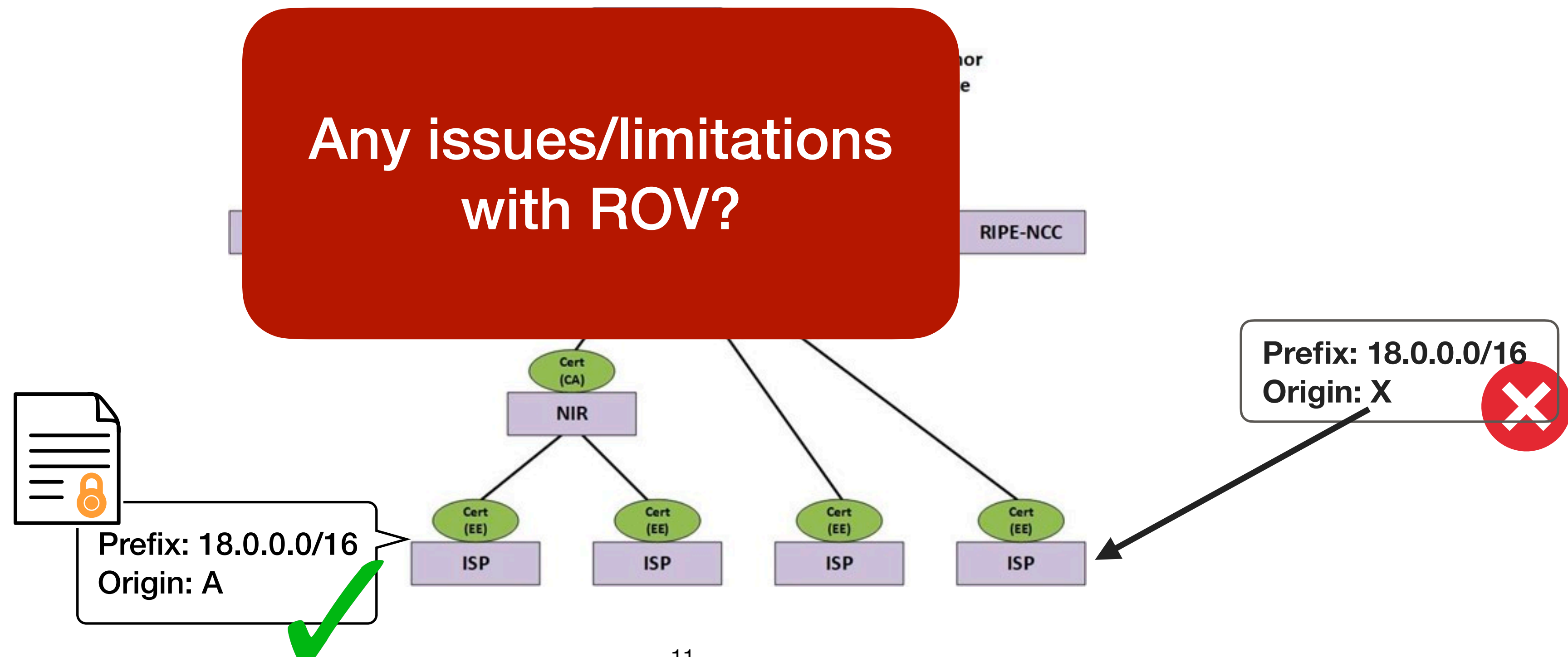
**Route Origin Validation (ROV):** ASes can share Route Origin Authorizations (ROAs), signed records of which ASes can originate a route for a prefix. BGP routers can validate that a BGP announcement has a valid origin (i.e., the origin AS in the AS-PATH is allowed to announce the prefix).





# BGP Defense

**Route Origin Validation (ROV):** ASes can share Route Origin Authorizations (ROAs), signed records of which ASes can originate a route for a prefix. BGP routers can validate that a BGP announcement has a valid origin (i.e., the origin AS in the AS-PATH is allowed to announce the prefix).




# BGP Defense

ROV validates origin AS can announce a BGP route (i.e., it "owns" the IP space being announced).

**BUT:** the rest of the BGP route is not validated.

Ex: Say AS19 owns IP prefix X, with ROV

AS-Path in benign BGP route for X: AS 24 -> AS 35 -> AS 19 

AS-Path in attacker's BGP route for X: **AS 666** -> AS 35 -> AS 19 



MITM



# BGP Defense

**BGPsec:** Each AS in the AS-PATH of a BGP route announcement provides a signature indicating that they announced the route to the next AS. This provides integrity over the whole AS-PATH, rather than just the origin AS.

AS-PATH: AS 24 -> AS 35 -> AS 19

BGPsec AS-PATH: AS 24 -> AS 35 -> AS 19

This route sent to AS 35  
(Signed by AS 19)

This route sent to AS 24  
(Signed by AS 35)

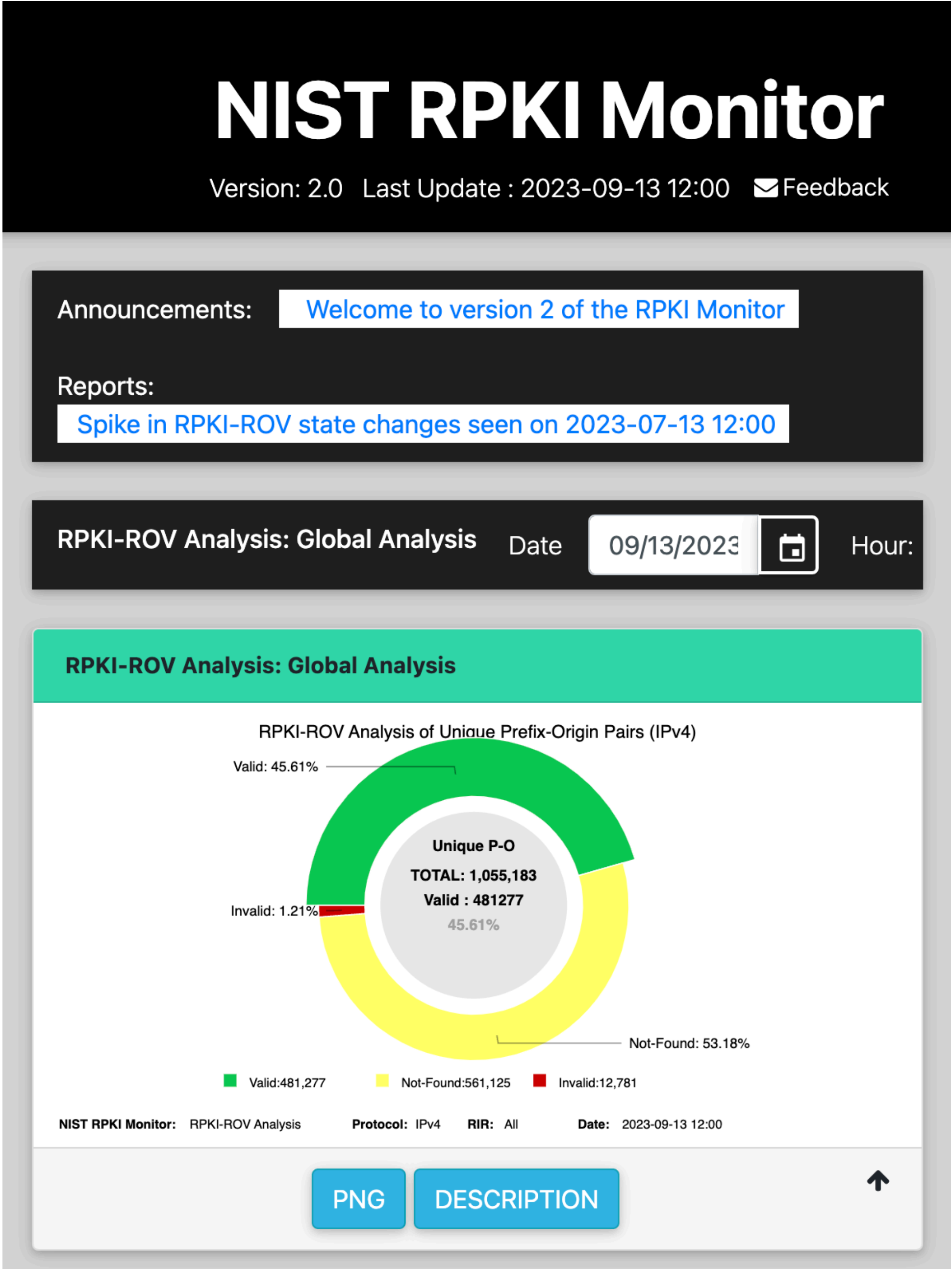
Attacker AS-Path: **AS 666** -> AS 35 -> AS 19

This route sent to AS 35  
(Signed by AS 19)

NO SIGNED ROUTE  
BY AS 35

# BGP Defense

**ROV:** Adopted by ~45% of routed prefixes





# BGP Defense

**BGPsec:** Very little adoption :(

Why?

- Only works if entire AS path uses it. If not all ASes use it, either need to
  - 1) not accept/use their routes (not good...)
  - 2) do not enforce BGPsec (then little value in adopting...)
- Also, doesn't prevent "route leaks". ROV validates origin AS announcements, but neither ROV or BGPsec prevent intermediate AS from propagating a route incorrectly.
- BGPsec entails performance overhead

BGP security is growing in importance, so likely will see developments soon!

# Today: Transport Layer Security



# Best Effort Delivery is Lame!

Last time: IP protocol + routing helps us send a packet from host A to host B.

IP packets are individual packets, with no guarantee of delivery, and no way to associate IP packets with specific applications on a host



# Transport Layer Protocols

**Goal:** Provide end-to-end communication between applications on two remote hosts.

**TCP (Transmission Control Protocol):** establish a reliable, ordered, and error-checked connection between applications on two hosts

- Heavyweight (requires a 3-way handshake to initiate)

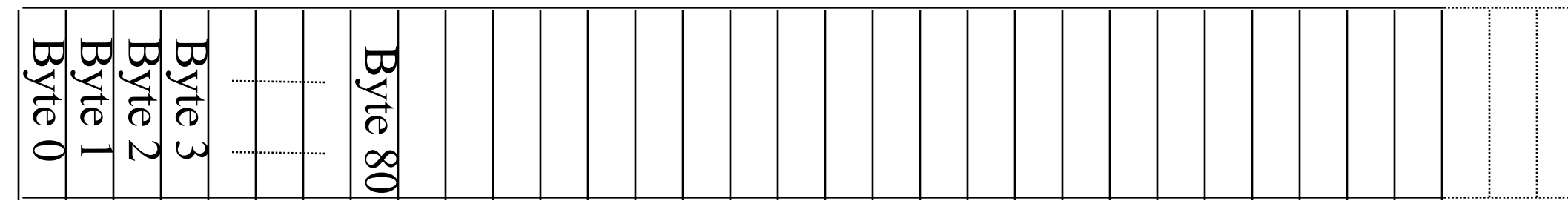
**UDP (User Datagram Protocol):** establish connectionless communication between applications on two hosts (fire and forget)

- No guarantee of delivery, ordering, or duplicate protection.
- Lightweight, good for time-sensitive or light request/response applications

To distinguish applications at either endpoints, both TCP and UDP use *port numbers* (16-bit value). Most server application protocols have standard port numbers (e.g., HTTP is port 80).

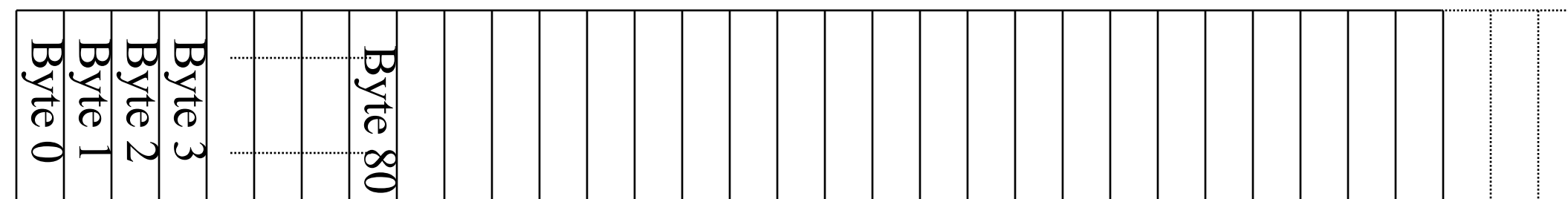
# TCP: Bytestream Abstraction

Process A on host H1



Processes don't ever see packet boundaries, lost or corrupted packets, retransmissions, etc.

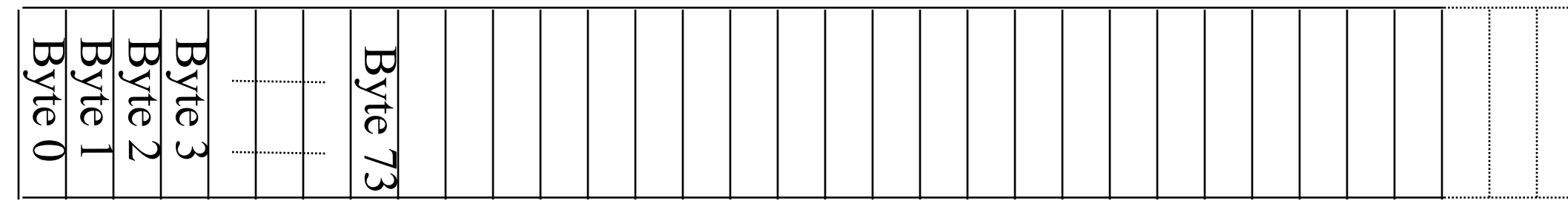
Process B  
on host H2





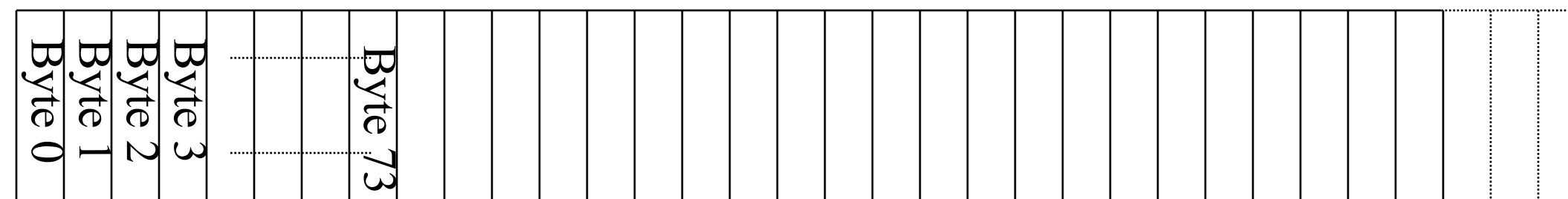
# TCP: Bidirectional Bytestream Abstraction

Process B on host H2

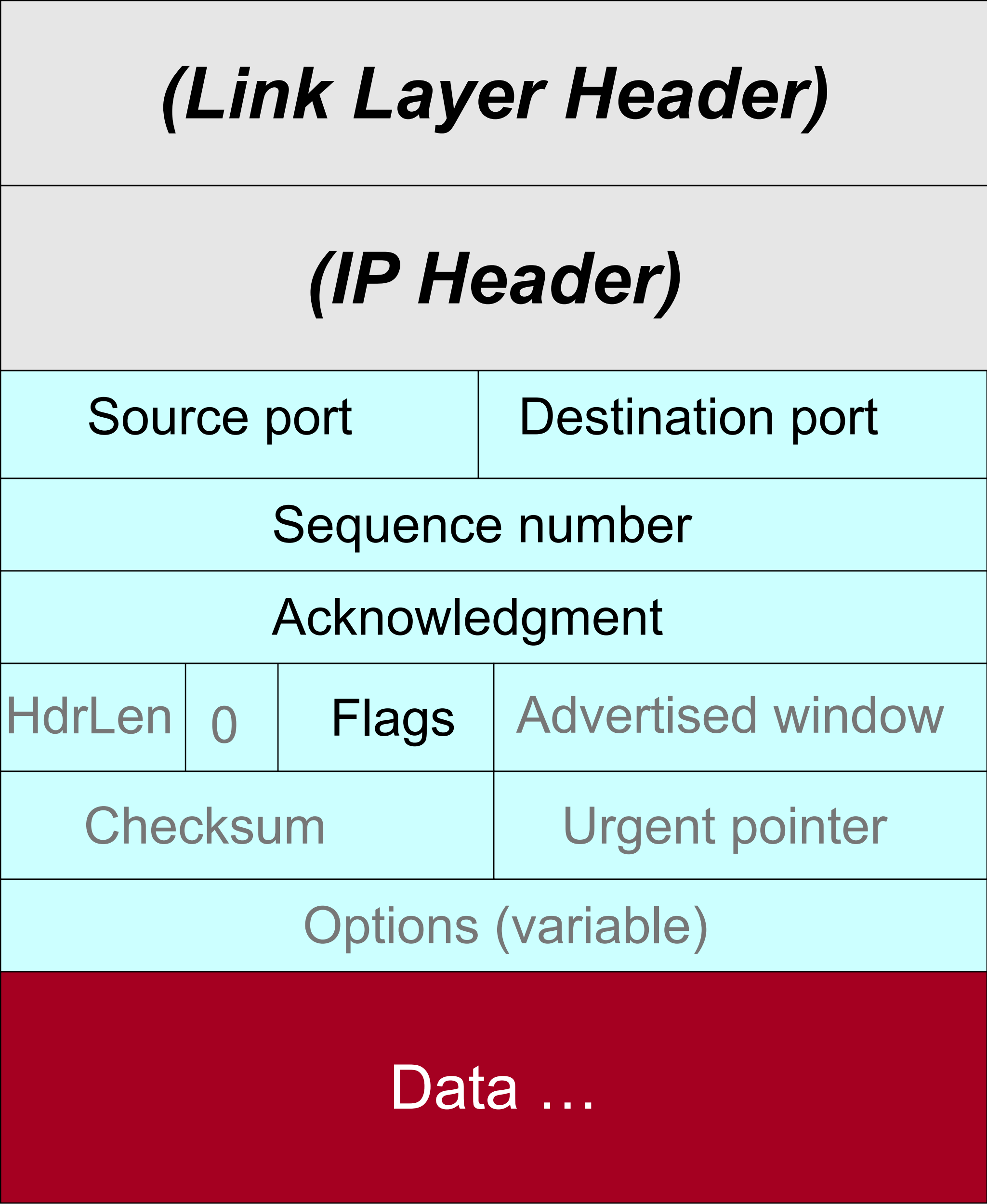


There are two separate **bytestreams**, one in each direction

Process A  
on host H1

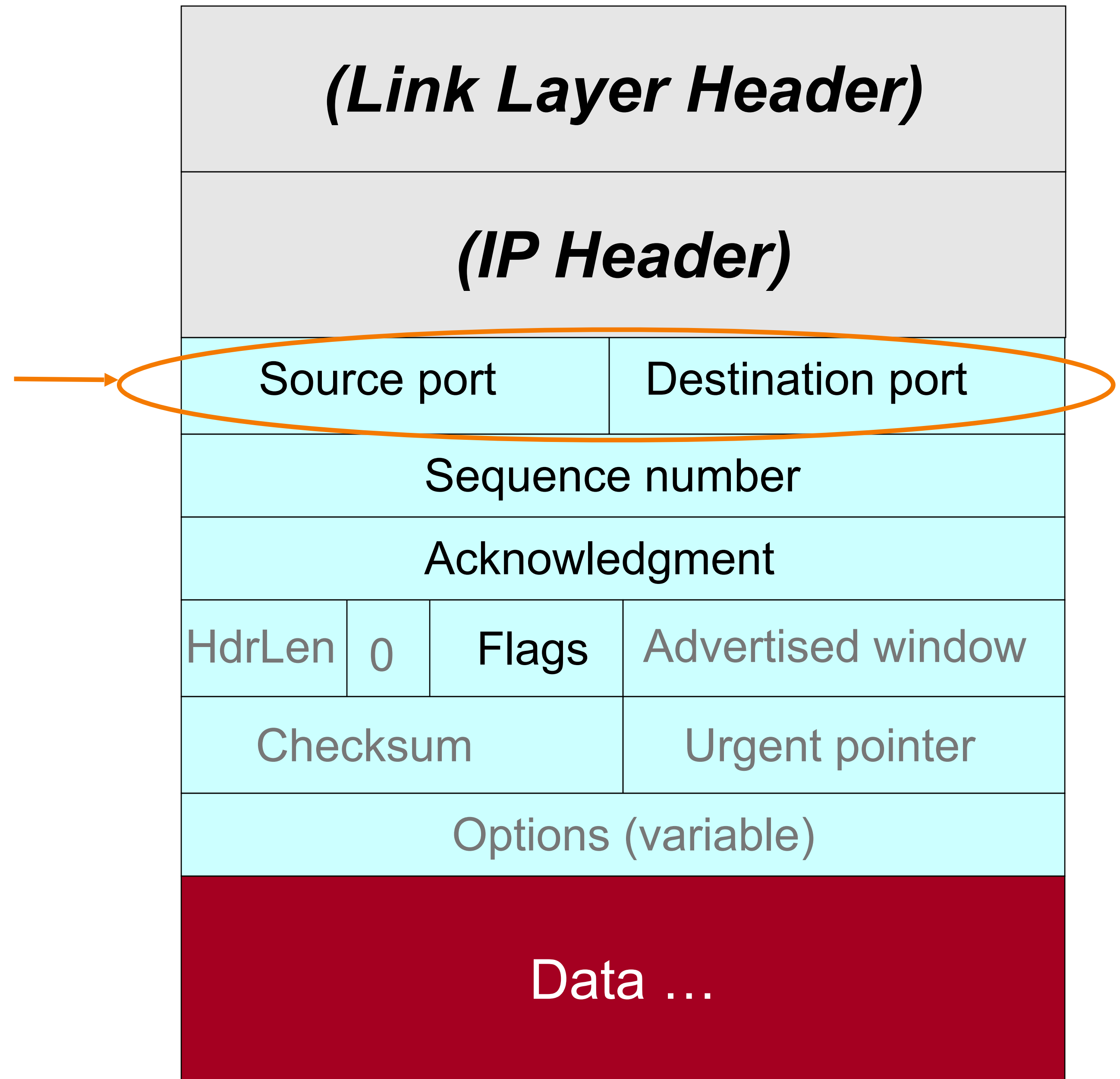


# TCP Header



# TCP Header

*Ports are associated with OS processes*



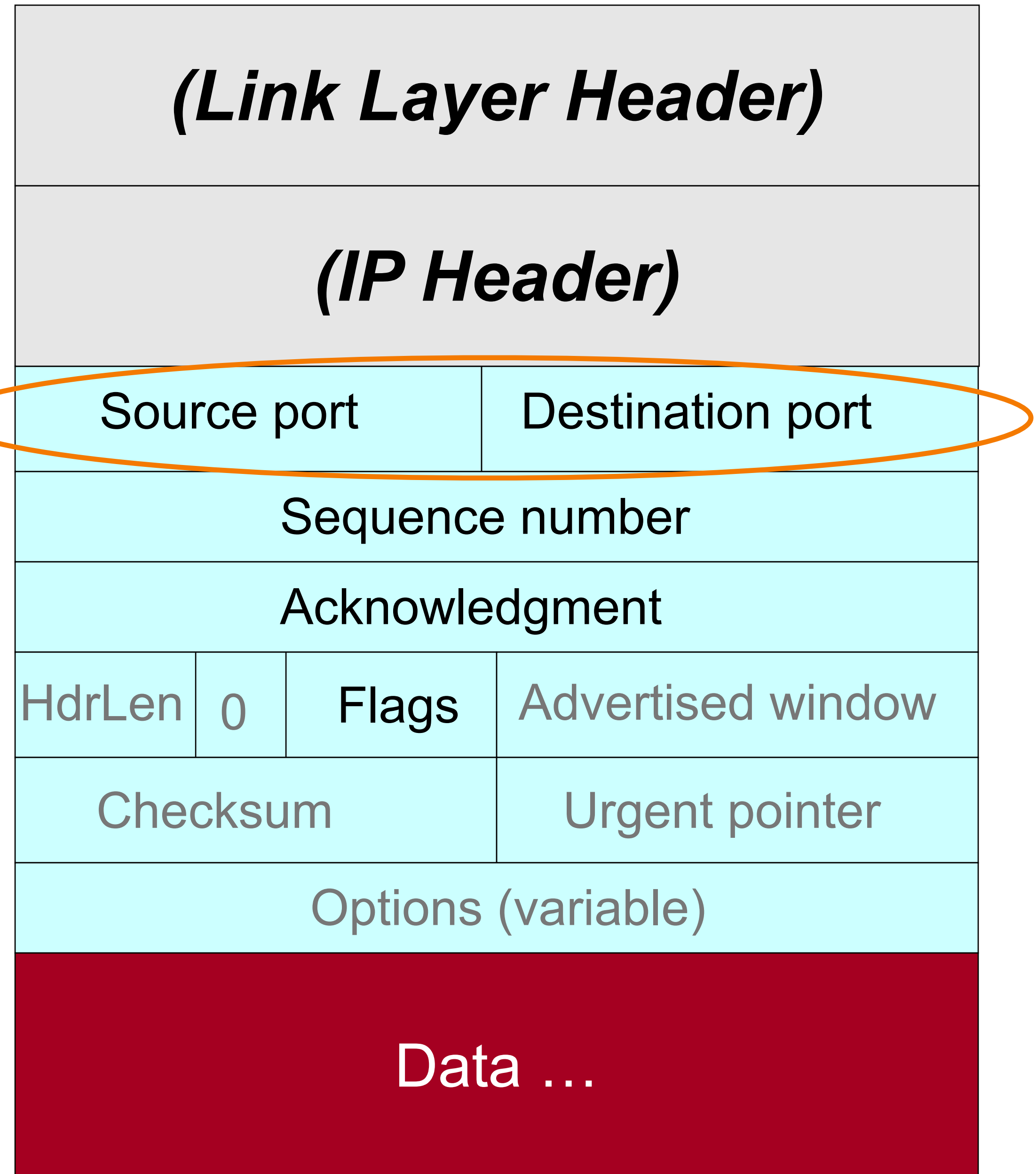


# TCP Header

Ports are associated  
with OS processes

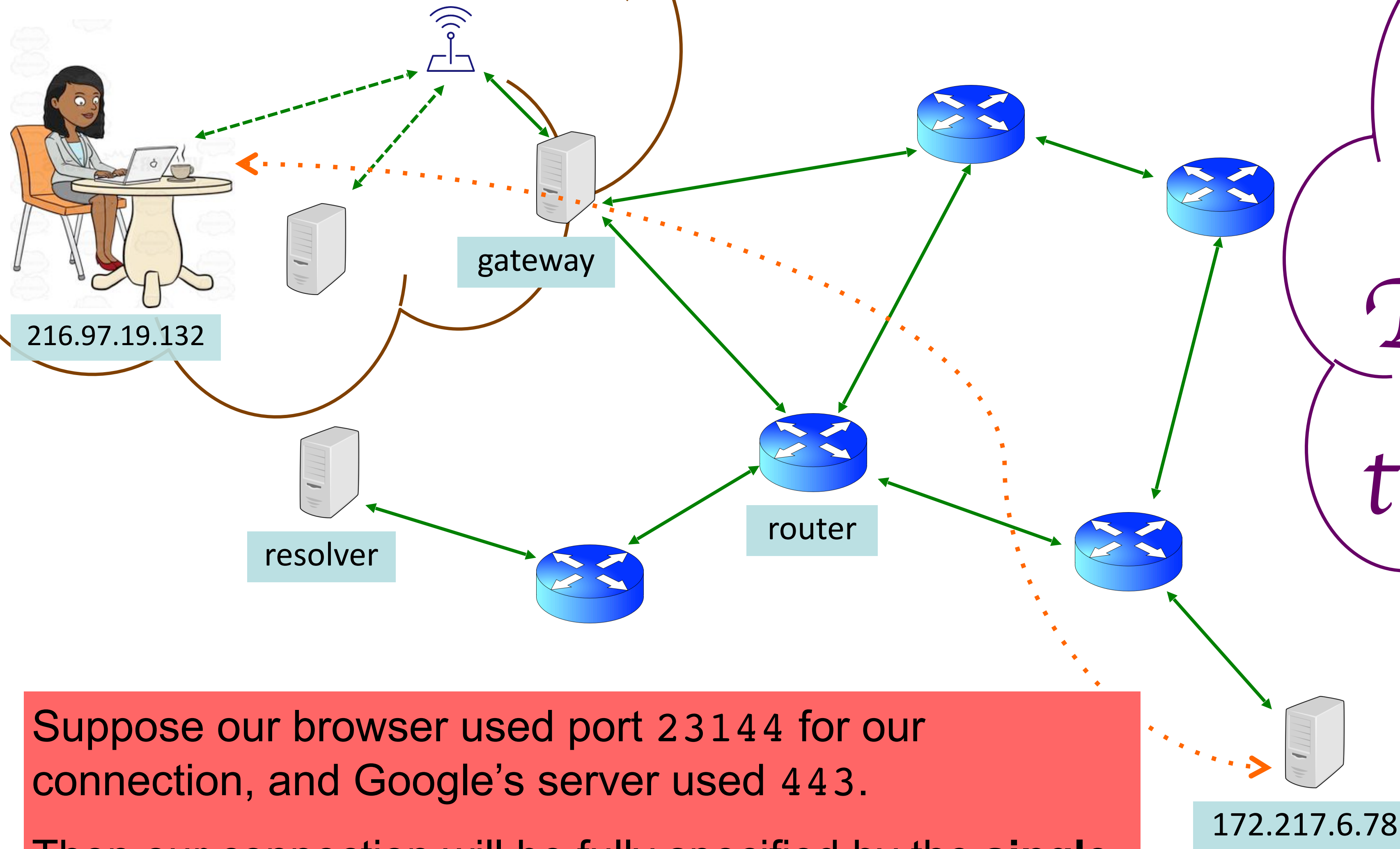
IP source & destination addresses  
plus TCP source and destination  
ports uniquely identifies a  
(bidirectional) TCP connection:

*(src IP, src port, dst IP, dst port)*



*Coffee Shop*

Connect to google.com server



*The Rest of the Internet*

Suppose our browser used port 23144 for our connection, and Google's server used 443.

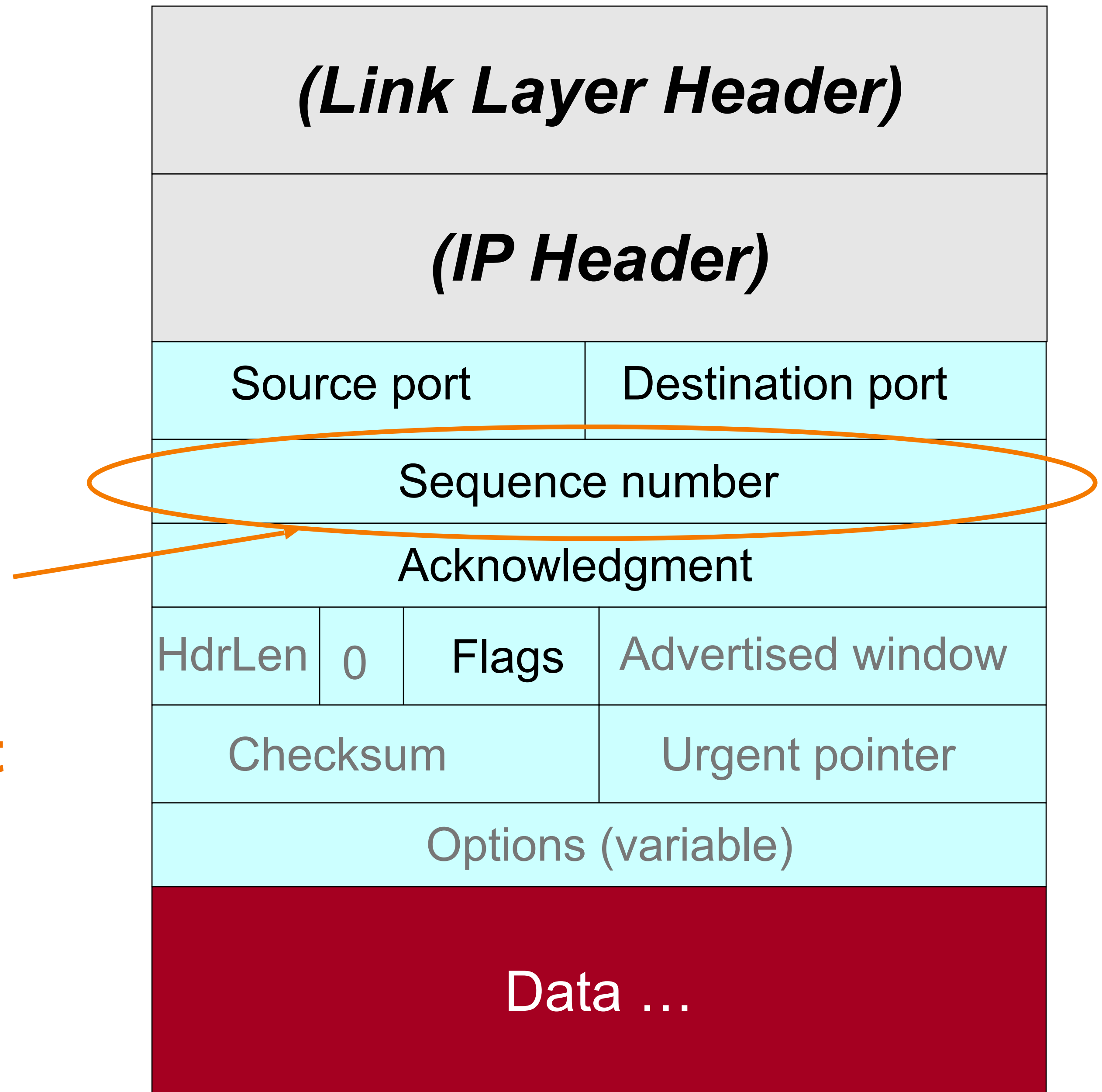
Then our connection will be fully specified by the **single** tuple  $\langle 216.97.19.132, 23144, 172.217.6.78, 443 \rangle$

# TCP Header

Starting sequence number (byte offset into the TCP bystream) of data carried in this packet (data from this host, me).

Initial seq # is picked when TCP connection is setup, **doesn't start at 0**.

Note: B/c TCP is bidirectional, each host tracks its own seq #.



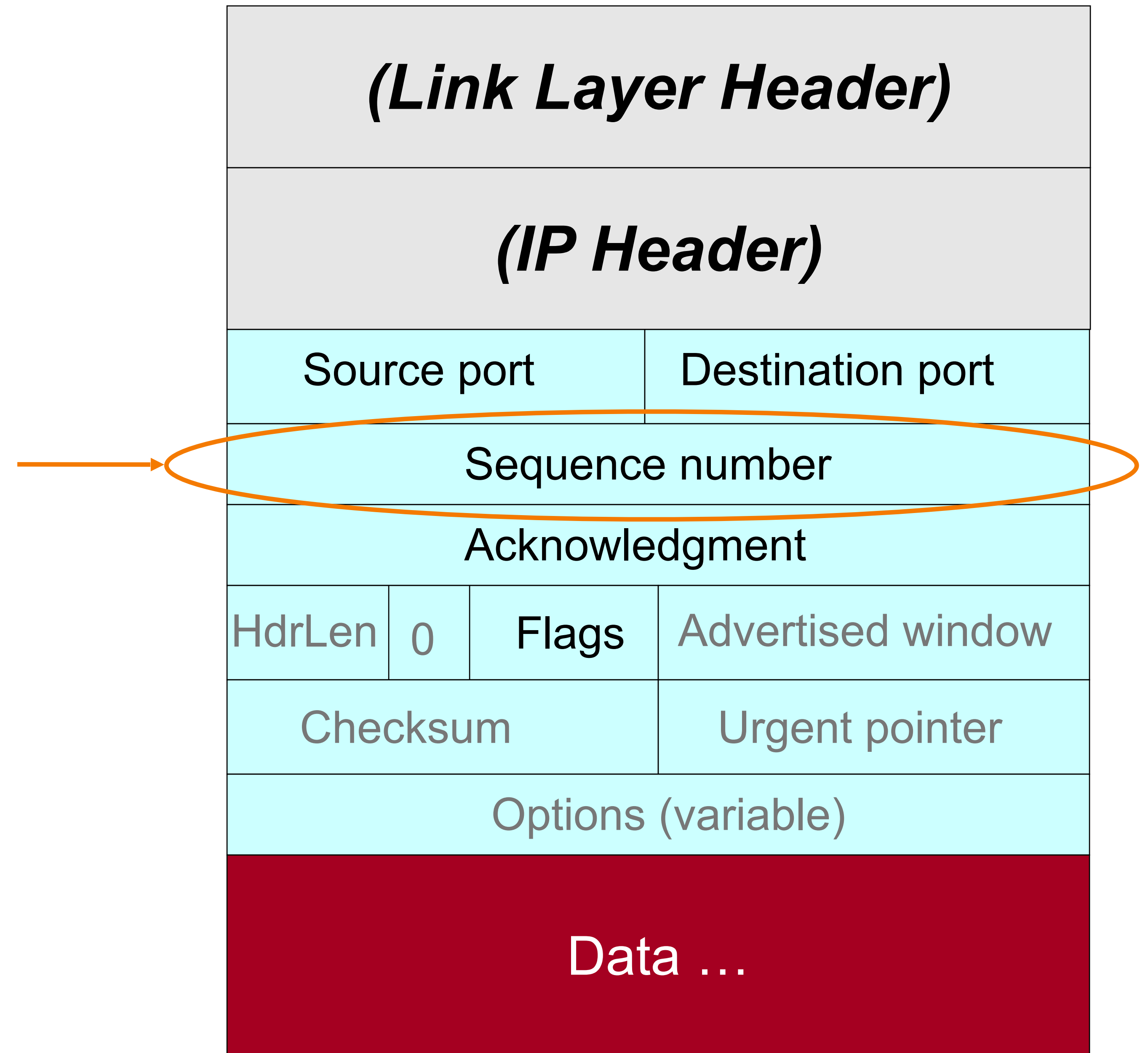


# TCP Header

Example:

My initial Seq # = 1024

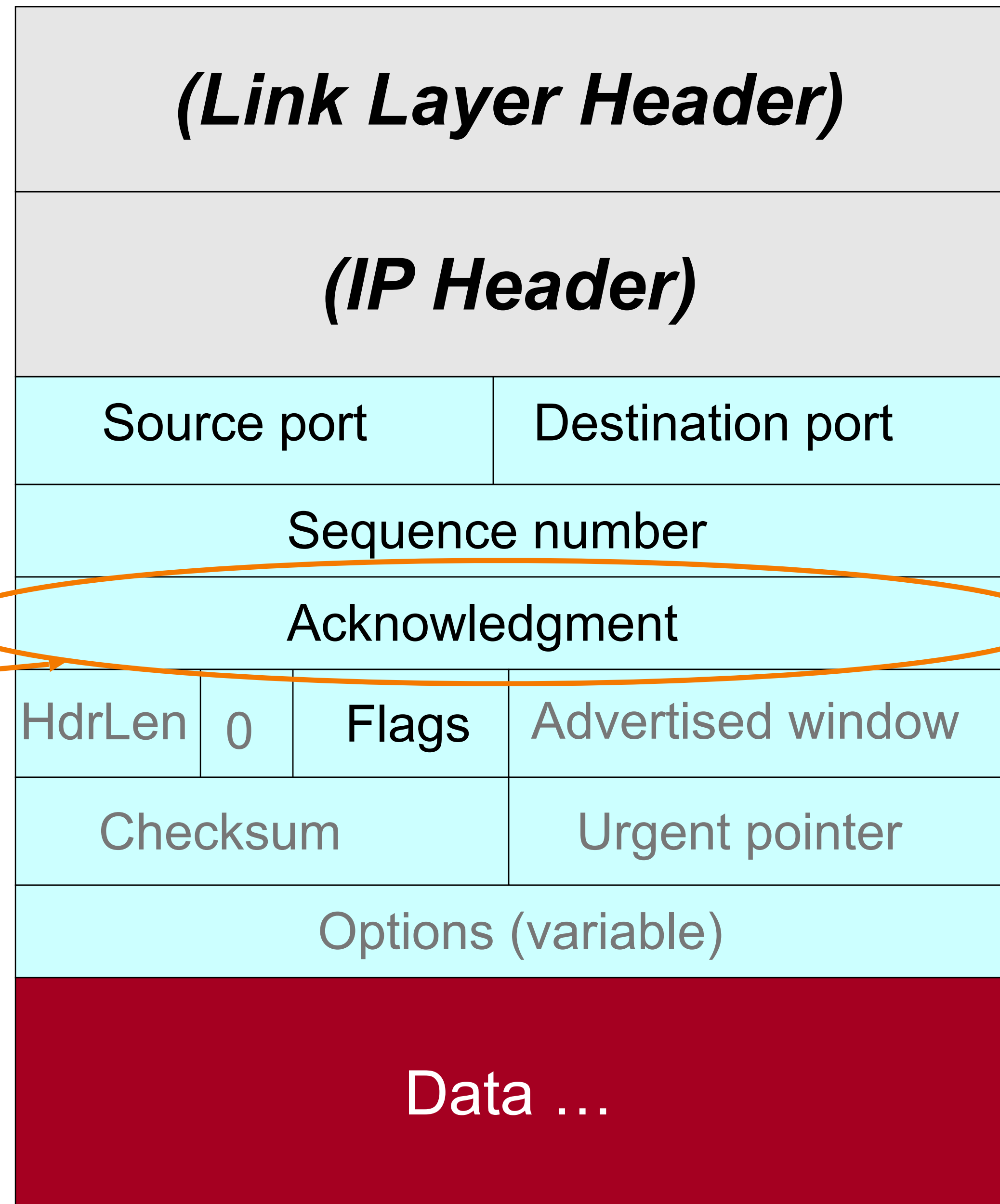
This packet's Seq # 1224: I'm about to send you the 200th byte in this bytestream



# TCP Header

Tells the other TCP host (you) what's the next seq # this host (me) expects to receive, based on what seq #'s this host (me) has received in order so far.

If the other TCP host just sent this host N bytes in a packet with seq # S, then this host will send a TCP ack # = S+N



# Sequence + Acknowledgement Numbers

Host A

ISN (initial sequence number)

Sequence number from A  
= 1<sup>st</sup> byte of data  
= ISN + bytes sent so far

TCP  
HDR

TCP Data

TCP  
HDR

TCP Data

Host B

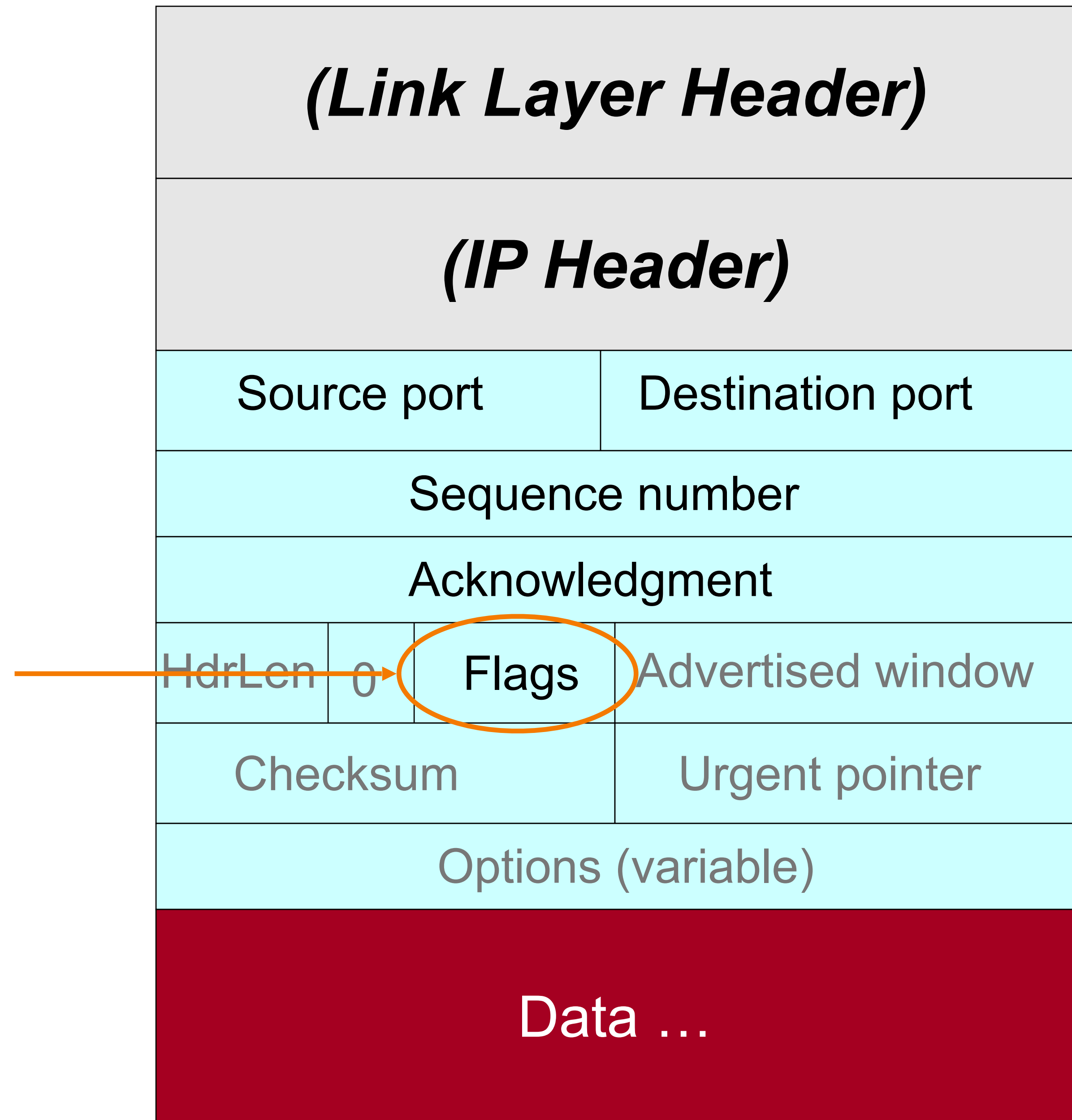
ACK number from B  
= next expected  
byte



# TCP Header

What does this TCP packet do?

- Acknowledging data (ACK packet has ACK flag = 1)
- Setting up TCP connection (SYN flag = 1)
- Closing this TCP connection (FIN = 1 or RST = 1)



# Establishing a TCP Connection

A

B



- *Three-way handshake* to establish connection

# Establishing a TCP Connection

A

B

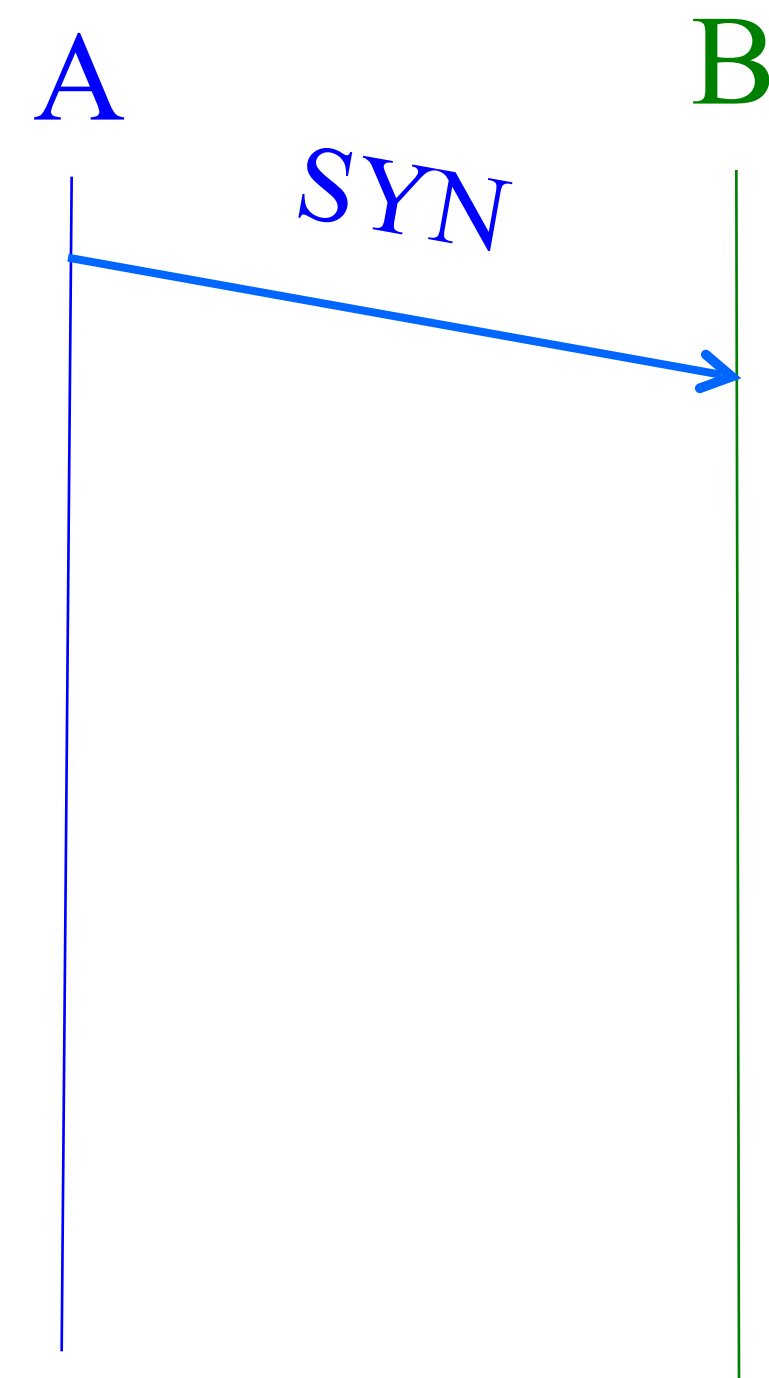
Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on a clock)

- *Three-way handshake* to establish connection



# Establishing a TCP Connection

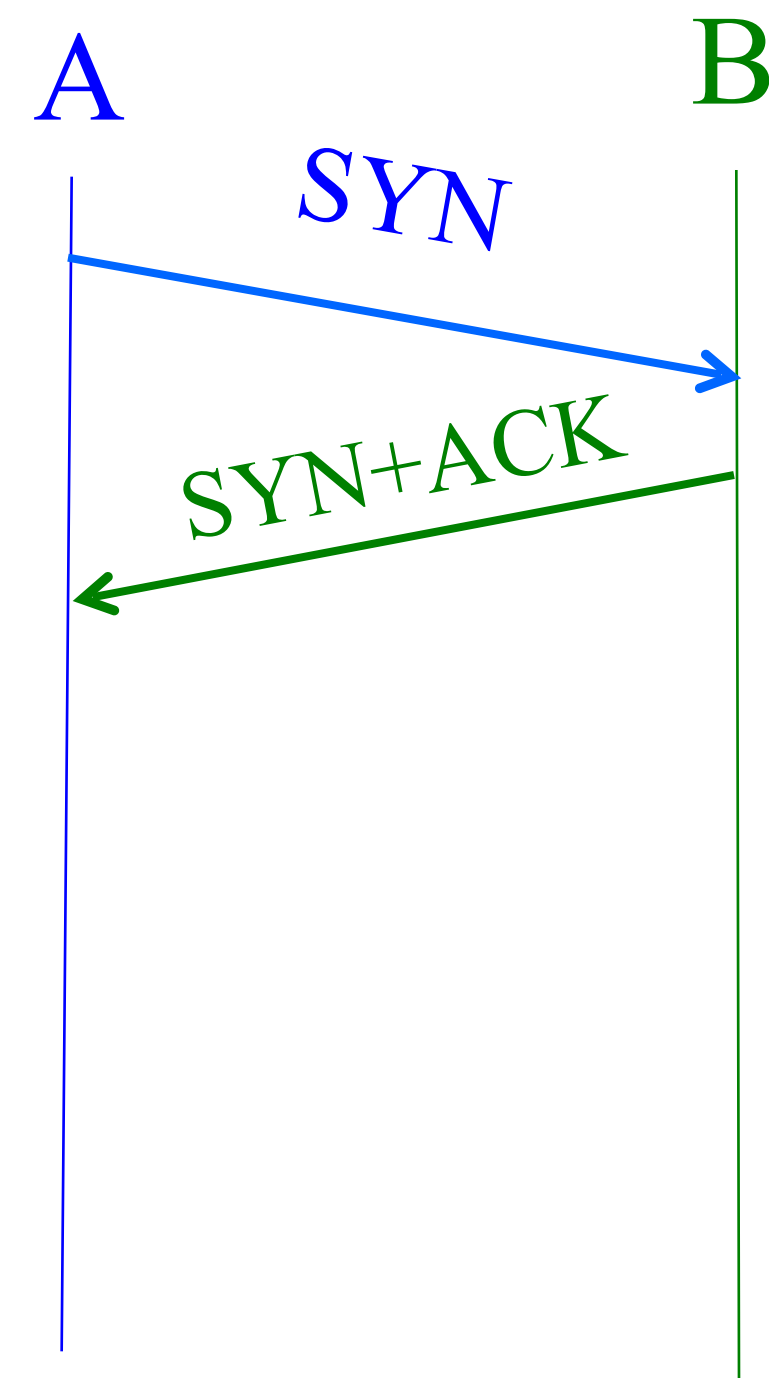


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on a clock)

- *Three-way handshake* to establish connection
  - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B

# Establishing a TCP Connection

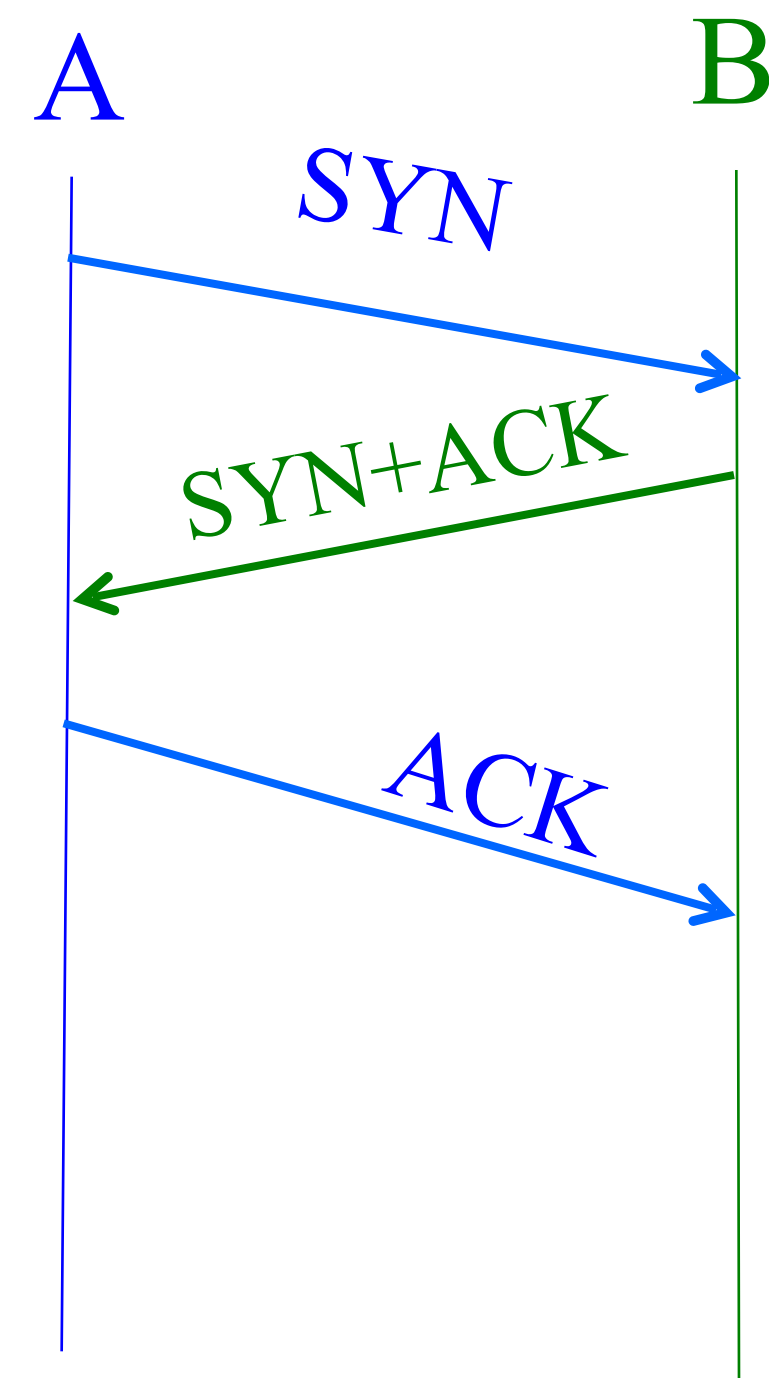


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on a clock)

- *Three-way handshake* to establish connection
  - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (**SYN+ACK**)

# Establishing a TCP Connection

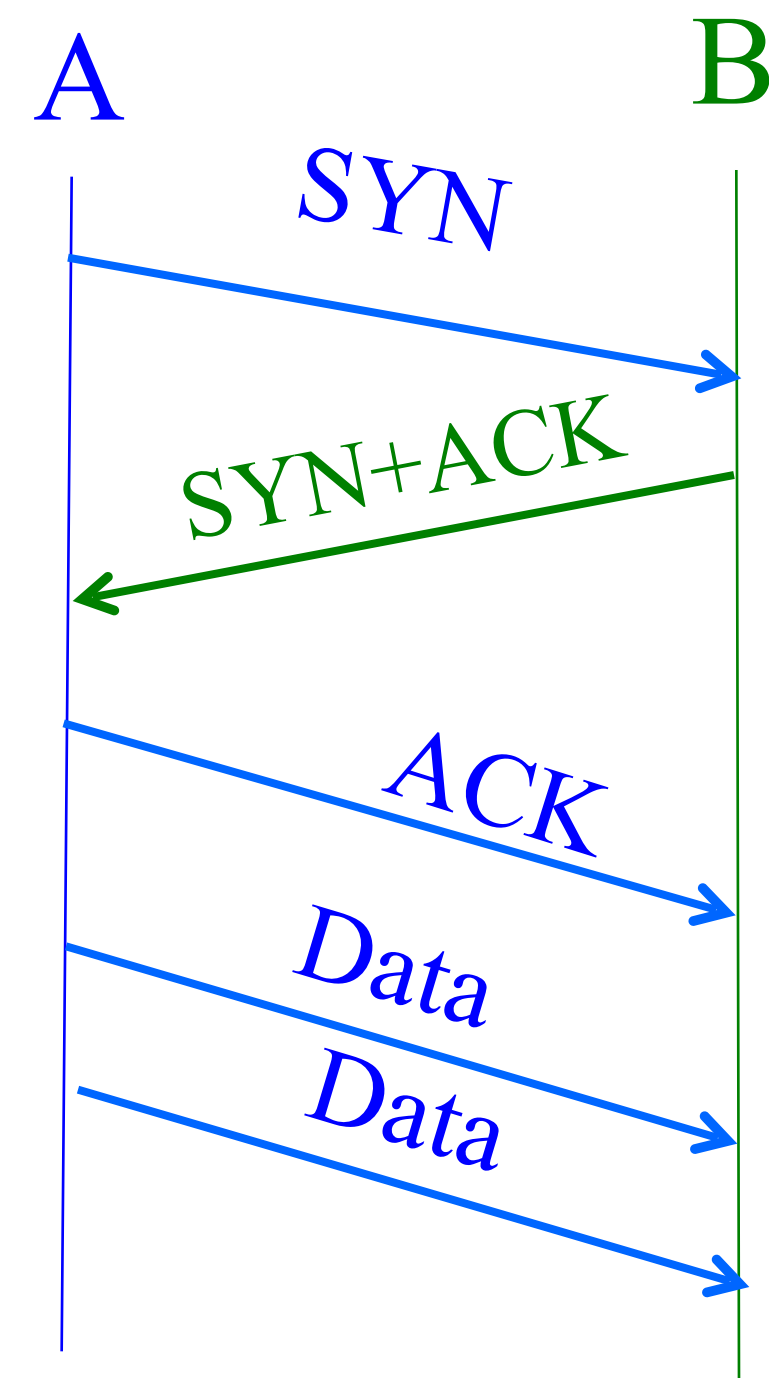


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on a clock)

- *Three-way handshake* to establish connection
  - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (**SYN+ACK**)
  - Host A sends an **ACK** to acknowledge the SYN+ACK

# Establishing a TCP Connection



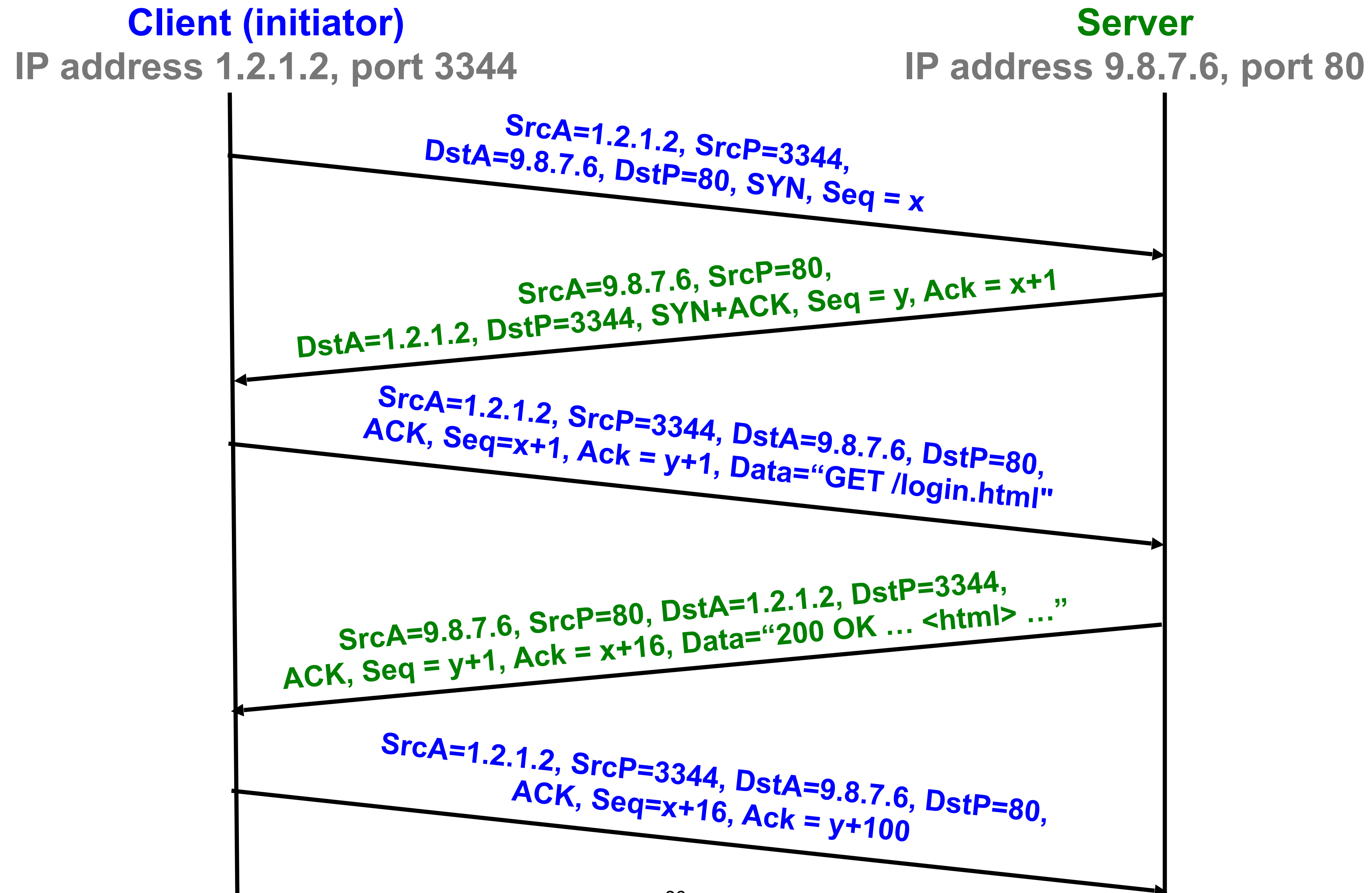
Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on a clock)

- *Three-way handshake* to establish connection
  - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (**SYN+ACK**)
  - Host A sends an **ACK** to acknowledge the SYN+ACK



# TCP Setup + Data Exchange



# TCP Reliability

Received data is only acknowledged in order.

- Sender sends 3 packets with seq #s of 100, 200, and 300 (each packet data is length 100).
- Receiver only receives packets with seq # 100 and 300. Receiver only sends packets with ACK # = 200 (next expected byte).
- Note, if receiver did receive all 3 packets close together, it can send 1 packet acknowledging all in-order data received so far (ACK # = 400).
- Selective ACKs (SACK): Optimization over traditional TCP, where receiver can indicate all seq #s received so far, so sender can retransmit only the missing segments

If sent data is not acknowledged with a certain timeout period, the sender will retransmit the data.

- Timeout period is variable and dynamically picked to reduce congestion in the network

# TCP Threats: Connection Disruption

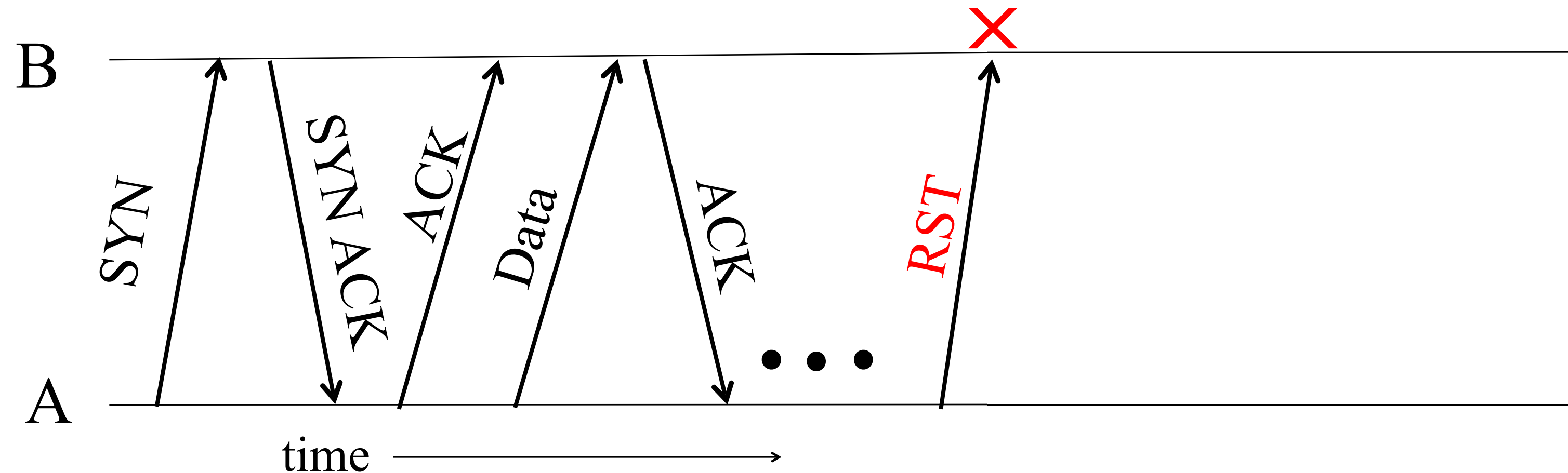
Normally, TCP finishes (“closes”) a connection by each side sending a FIN control message.

- Reliably delivered, since other side must ack

But: if a TCP endpoint finds unable to continue (process dies; info from other “peer” is inconsistent), it abruptly **terminates** by sending a RST control message

- Unilateral
- Takes effect immediately (no ack needed)
- Only accepted by peer if has correct\* sequence number

# Abrupt Termination



- A sends a TCP packet with RESET (**RST**) flag to B
  - E.g., because app. process on A **crashed**
- Assuming that the sequence numbers in the **RST** fit with what B expects, **That's It:**
  - B's user-level process receives: ECONNRESET
  - No further communication on connection is possible



# TCP Threats: Connection Disruption

Normally, TCP finishes (“closes”) a connection by each side sending a FIN control message.

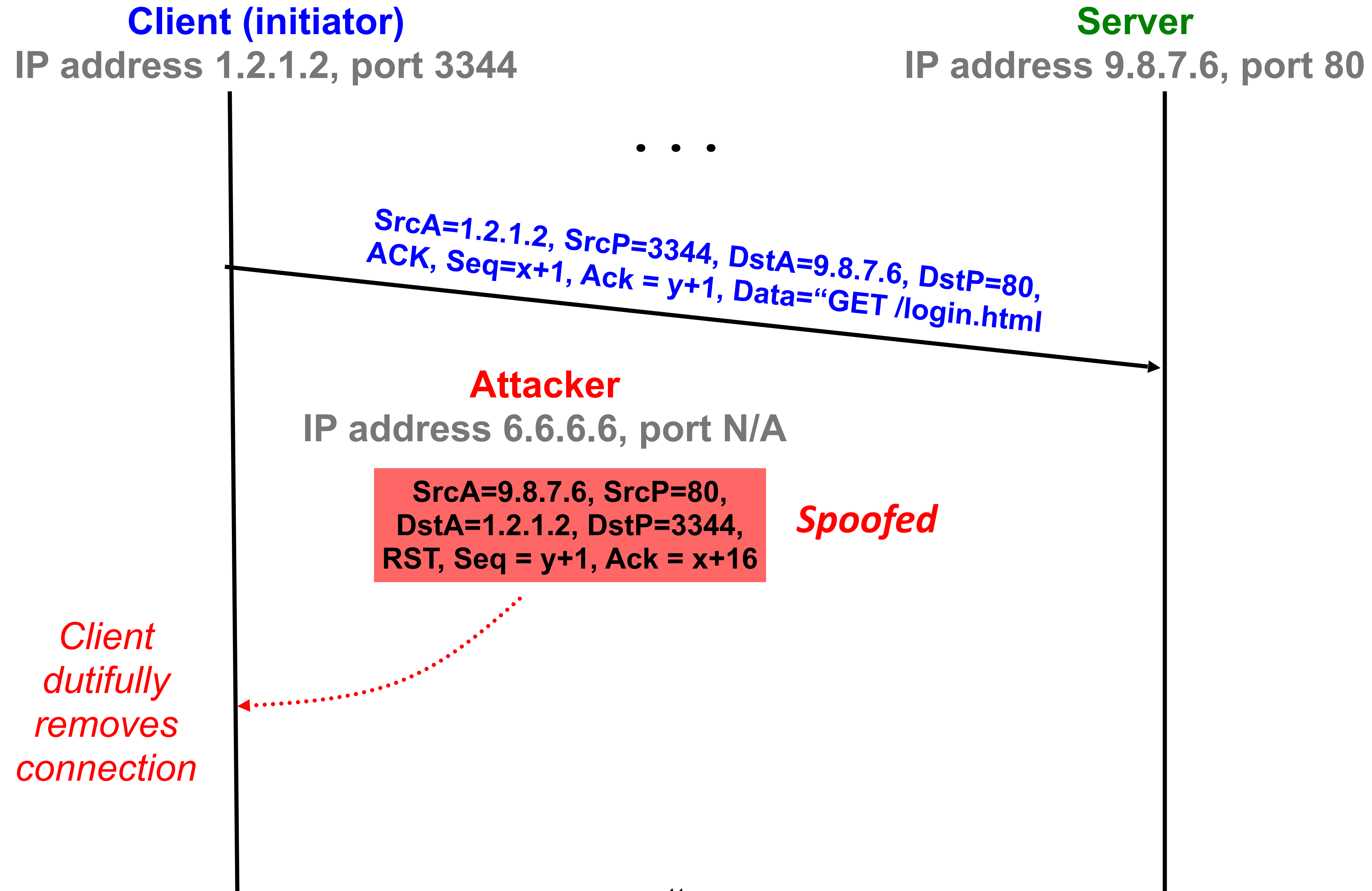
- Reliably delivered, since other side must ack

But: if a TCP endpoint finds unable to continue (process dies; info from other “peer” is inconsistent), it abruptly **terminates** by sending a RST control message

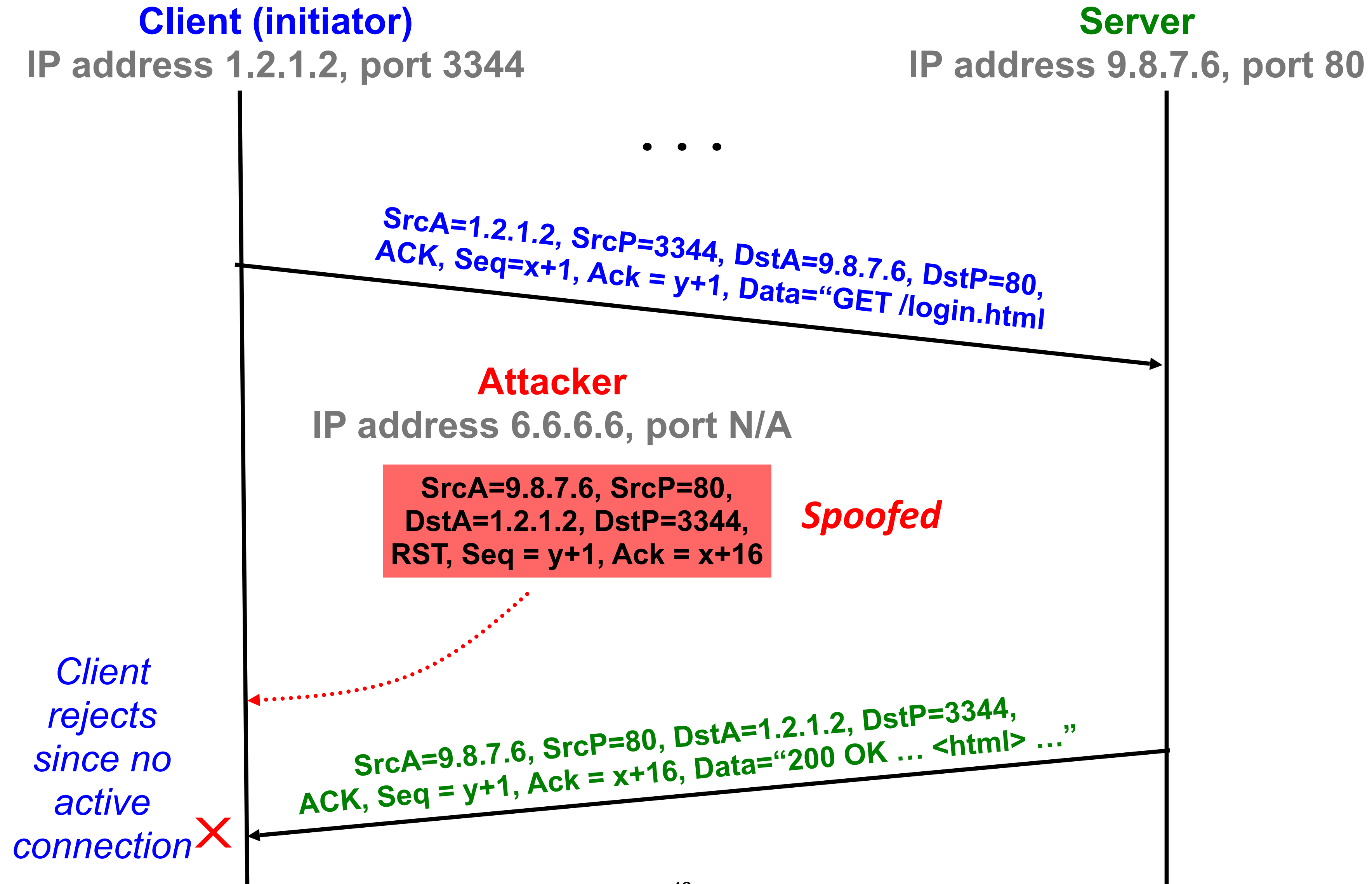
- Unilateral
- Takes effect immediately (no ack needed)
- Only accepted by peer if has correct\* sequence number

So if attacker knows/can guess the **ports & sequence** numbers, can disrupt a TCP connection with a spoofed packet. (Could be MITM or sniffing attacker)

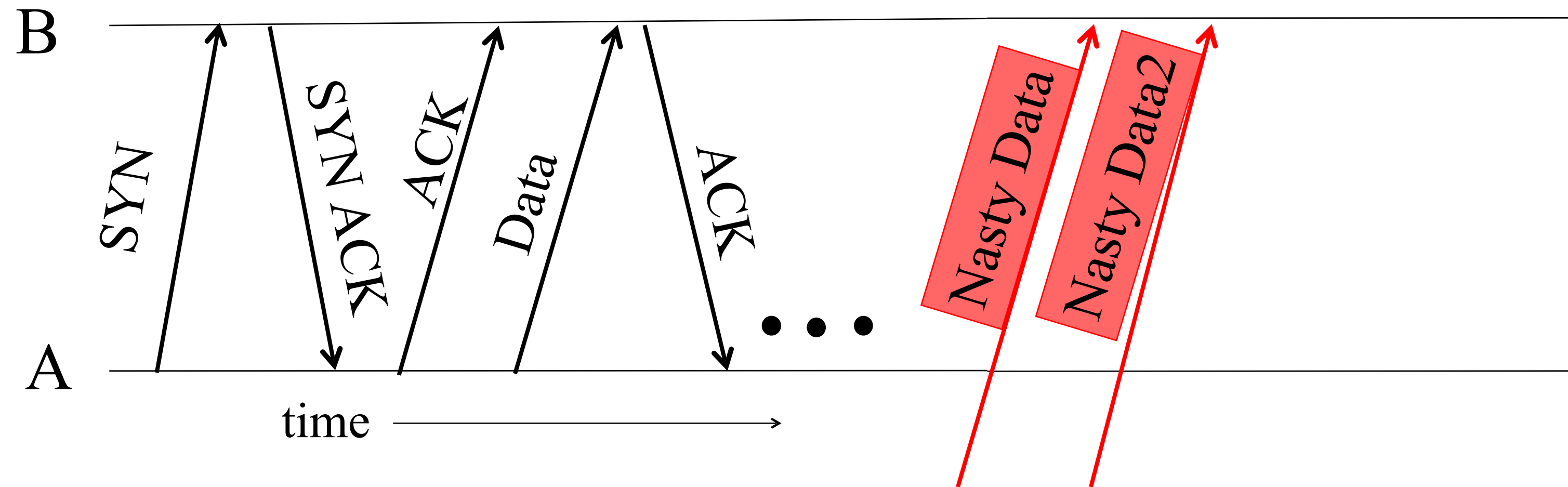
# TCP RST Injection



# TCP RST Injection



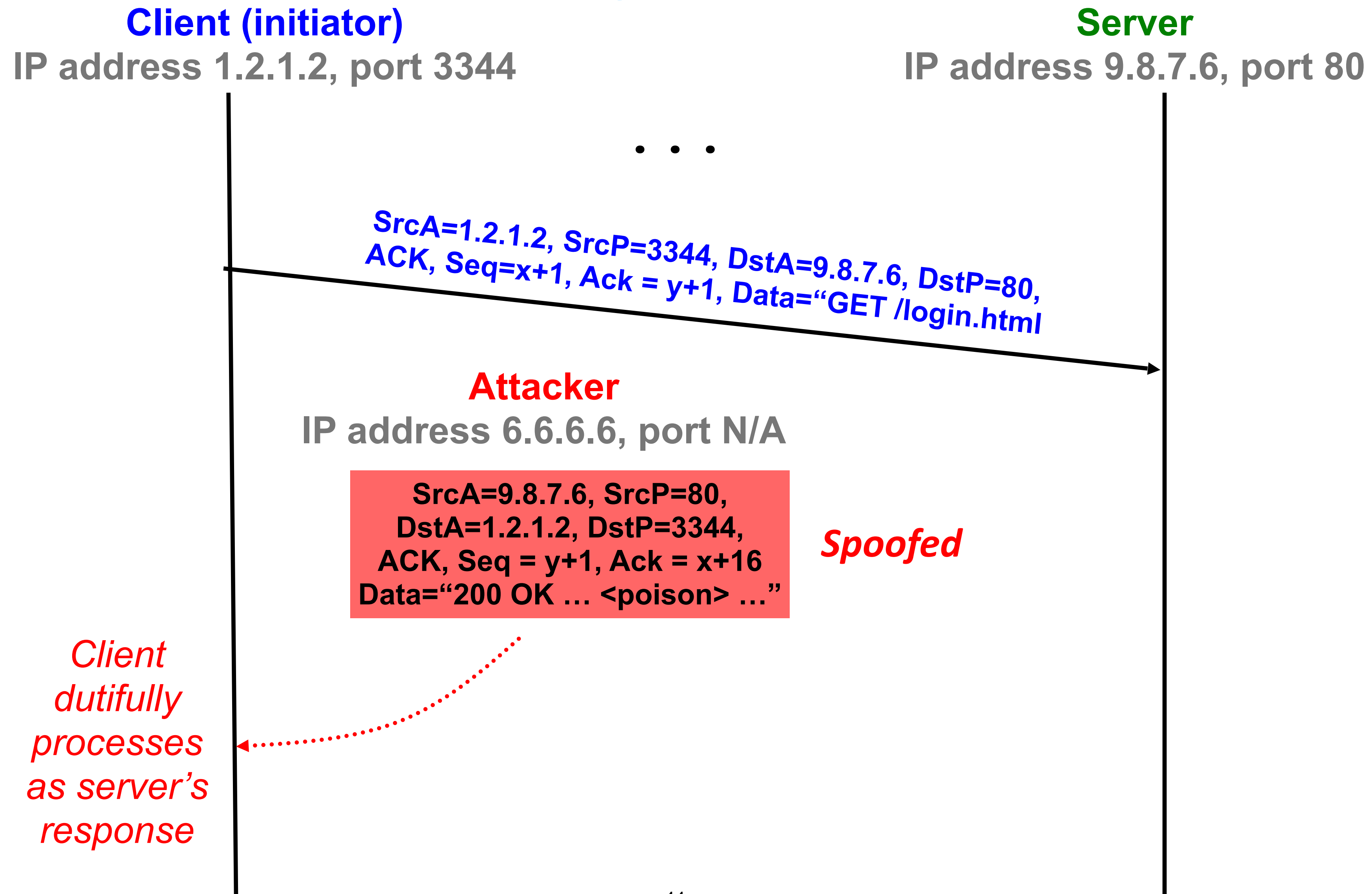
# TCP Threats: Data Injection



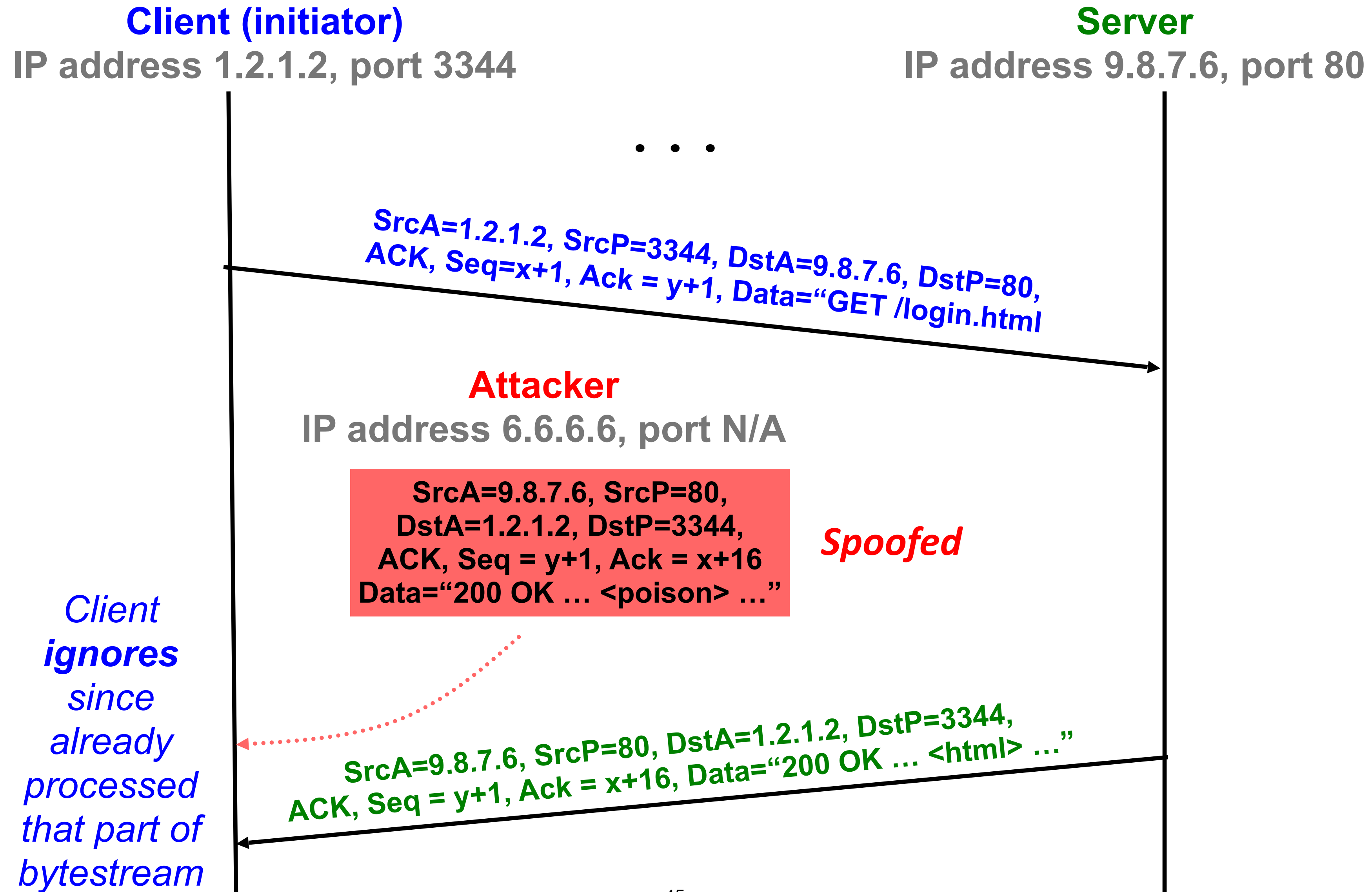
- What about **inserting data** rather than disrupting a connection?
  - Again, all that's required is attacker knows correct ports, seq. numbers
  - Receiver B is *none the wiser!*
- Termed TCP **connection hijacking** (or “*session hijacking*”)
  - A general means to take over an already-established connection!
- **We are toast if an attacker can see our TCP traffic!**
  - Because then they immediately know the **port & sequence numbers**



# TCP Threats: Data Injection



# TCP Threats: Data Injection



# TCP Threats: Blind Spoofing

- Is it possible for an attacker to inject into a TCP connection even if they **can't** see our traffic?
- **YES**: if somehow they can **infer** or **guess** the port and sequence numbers
- Original specifications said to pick seq #s based on clock. If so, attacker may be able to infer the numbers.
  - How? Attacker makes a legitimate connection with a server, and observes the initial seq #. Can potentially learn what subsequent initial seq #s will be.
  - Defense? Randomize the initial seq #.

# TCP Threat Summary

TCP 4-tuple (source and destination IPs + ports) define a TCP connection, and seq #s indicate how far into the bytestream the connection is (going in one direction).

If an attacker can learn these values, they can spoof TCP packets that will be accepted, allowing for:

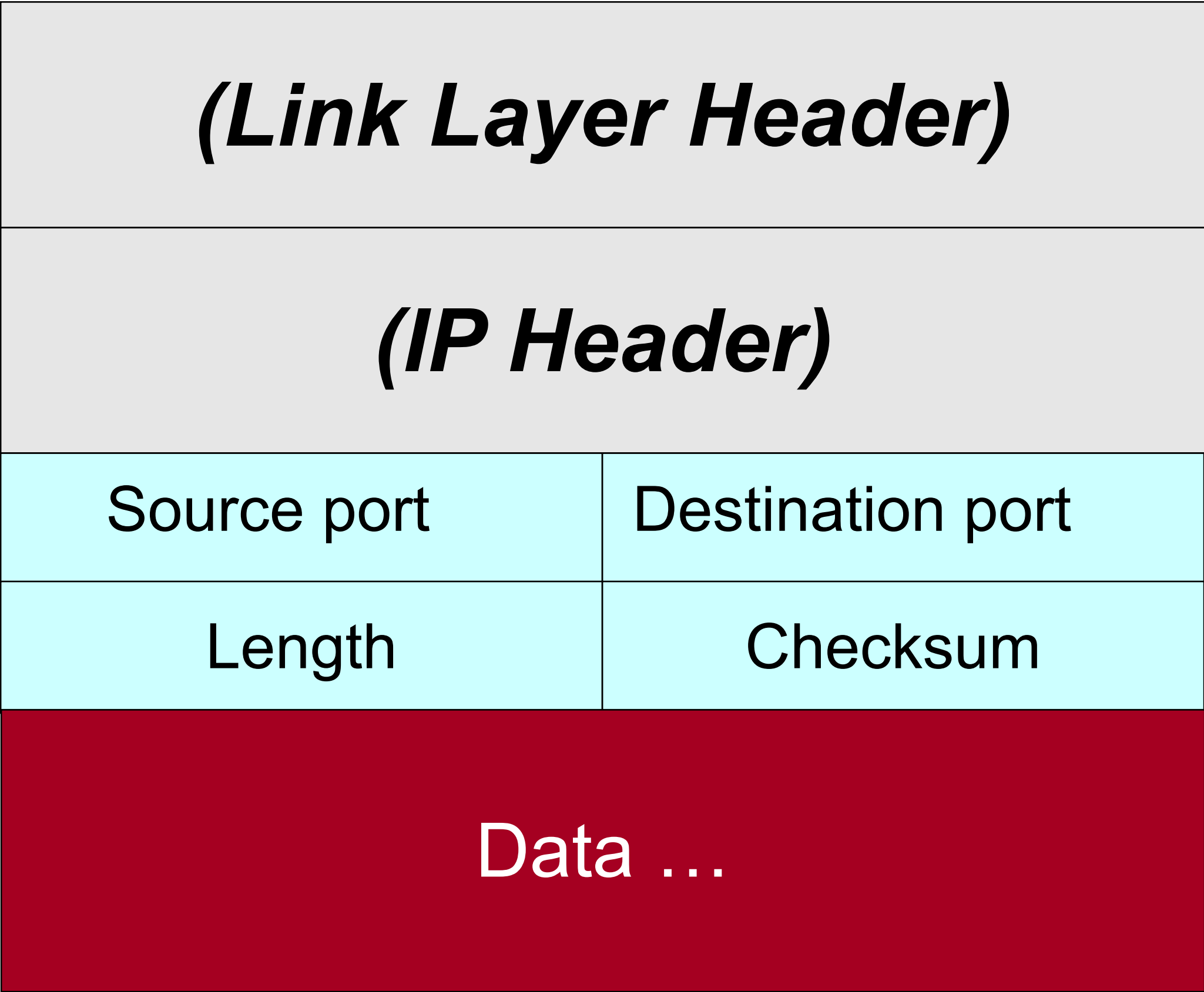
- Connection disruption via RSTs
- Inject fake data into the TCP bytestream

These attacks have been used in practice, particularly for Internet censorship and denial-of-service attacks.

Defenses are limited (would need to rely on some crypto, typically done with TLS at the application level, but also could use IPSec)



# UDP Header



# UDP Security Issues

Since UDP is connectionless and unreliable:

- Easy to terminate UDP-based communication without detection.
- Easier to inject packets into UDP-based communication
- Easy to modify or reorder UDP packets
- Attacker can send UDP packets from spoofed IP addresses

Implications of UDP security are noticeable at the application layer (coming up in next lectures)

