Network Attacks, Con't

CS 161: Computer Security

Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula,
David Fifield, Mia Gil Epner, David Hahn, Warren He,
Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic,
Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

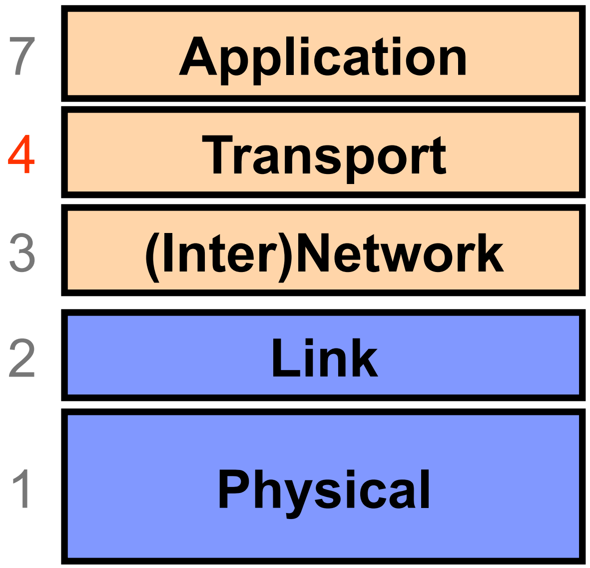
March 14, 2017

The Transport Layer: TCP

"Best Effort" is Lame! What to do?

- It's the job of our Transport (layer 4) protocols to build data delivery services that our apps need out of IP's modest layer-3 service

Layer 4: Transport Layer



End-to-end communication between processes

Different services provided:

TCP = reliable *byte stream*

UDP = unreliable *datagrams*

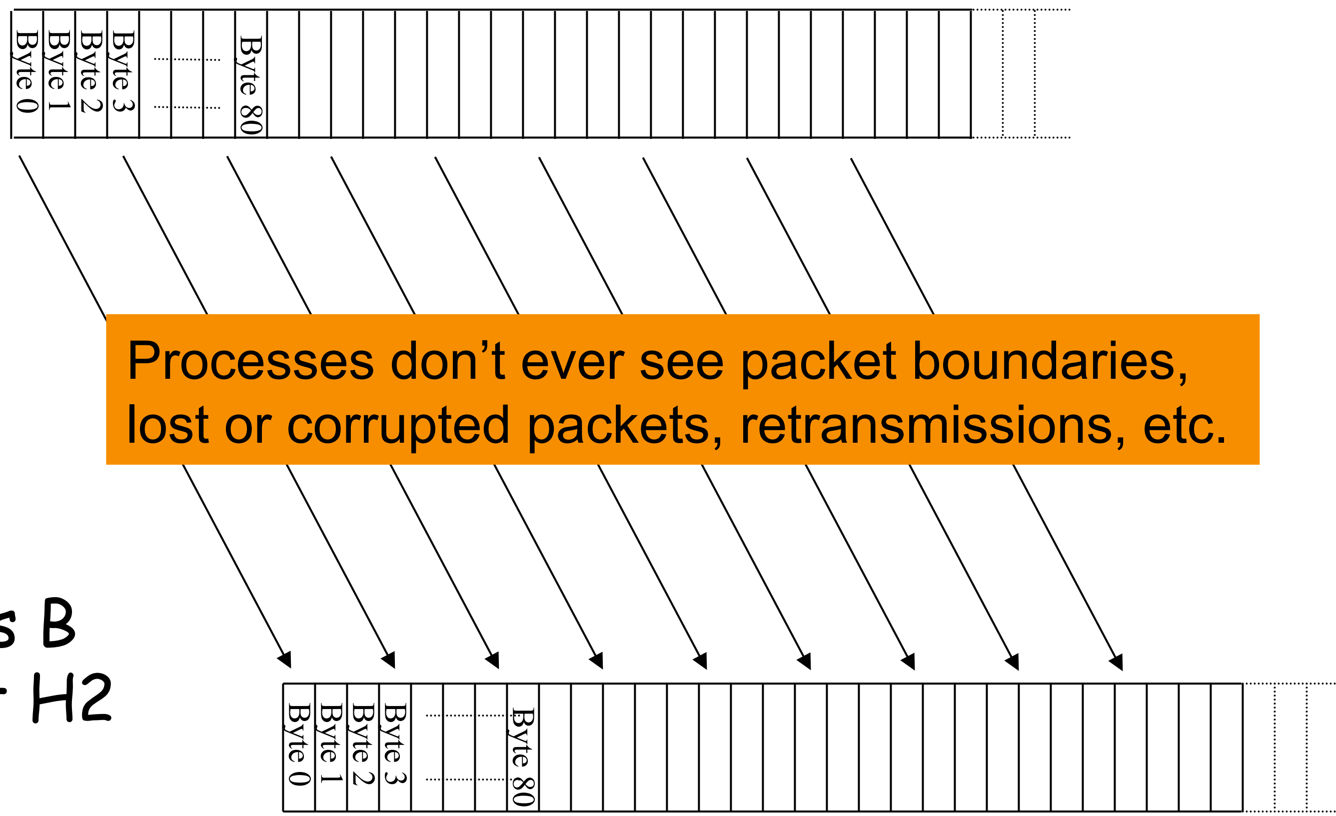
(Datagram = single packet message)

"Best Effort" is Lame! What to do?

- It's the job of our Transport (layer 4) protocols to build data delivery services that our apps need out of IP's modest layer-3 service
- #1 workhorse: TCP (Transmission Control Protocol)
- Service provided by TCP:
  - Connection oriented (explicit set-up / tear-down)
    - End hosts (processes) can have multiple concurrent long-lived communication
  - **Reliable**, in-order, *byte-stream* delivery
    - Robust detection & retransmission of lost data

TCP “Bytestream” Service

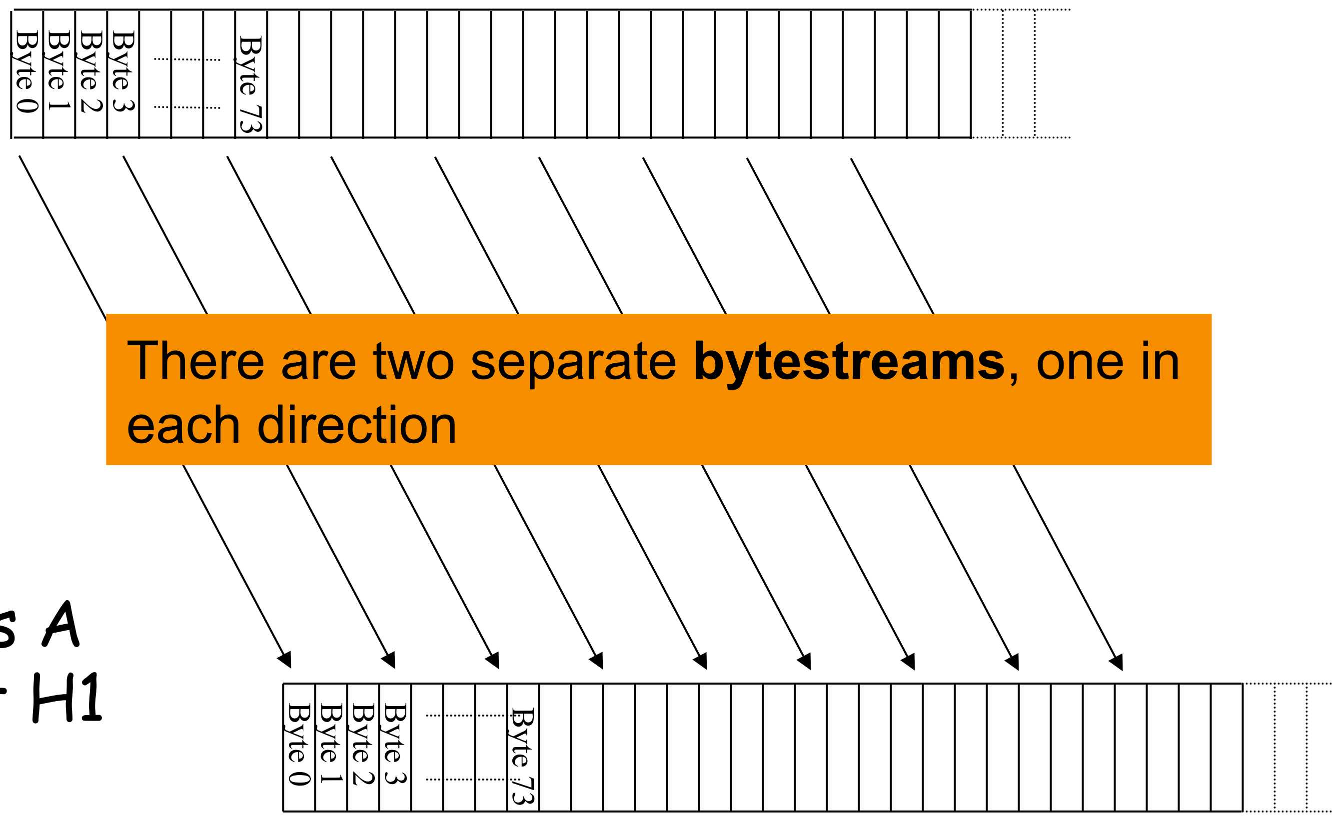
Process A on host H1



Process B
on host H2

Bidirectional communication:

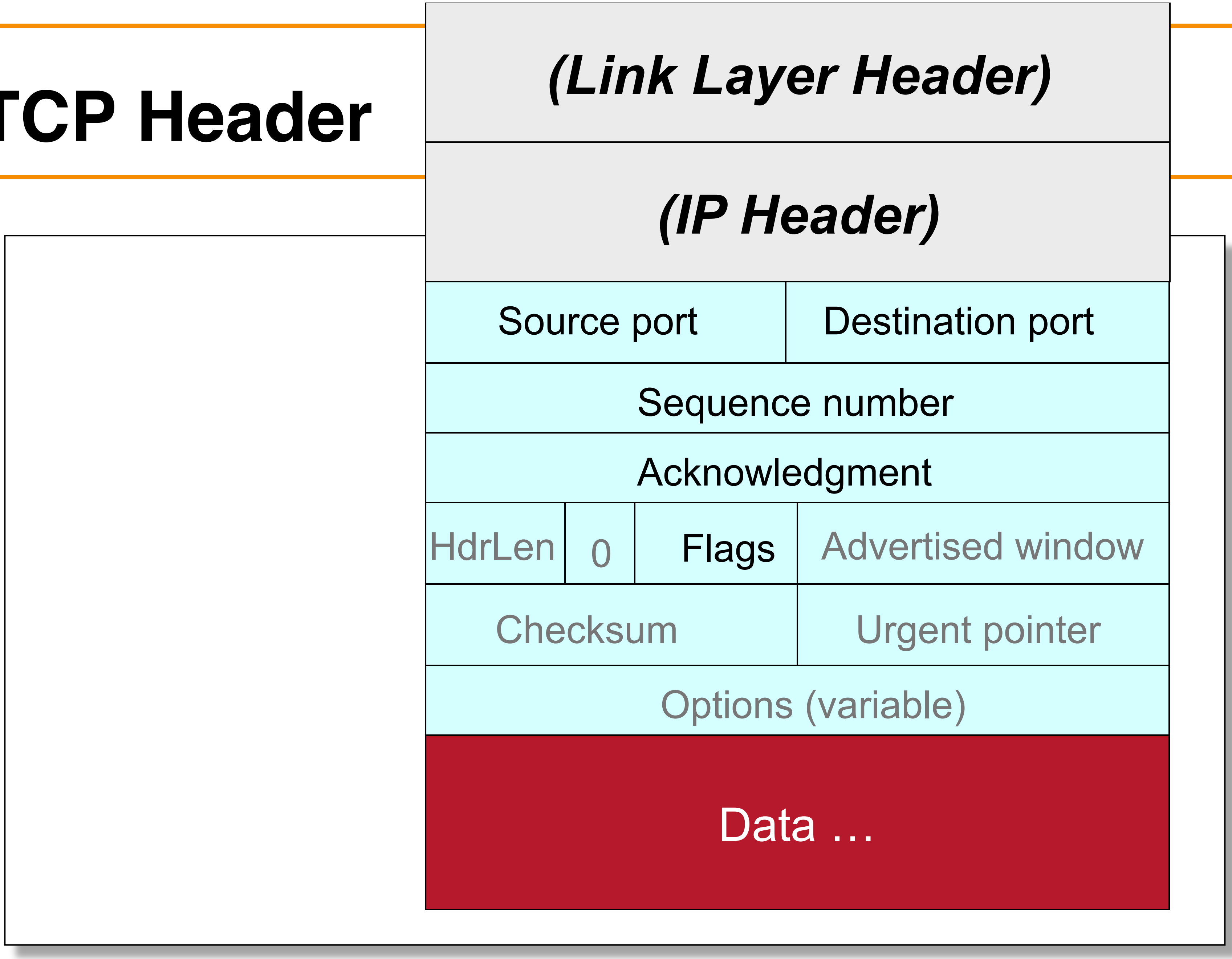
Process B on host H2



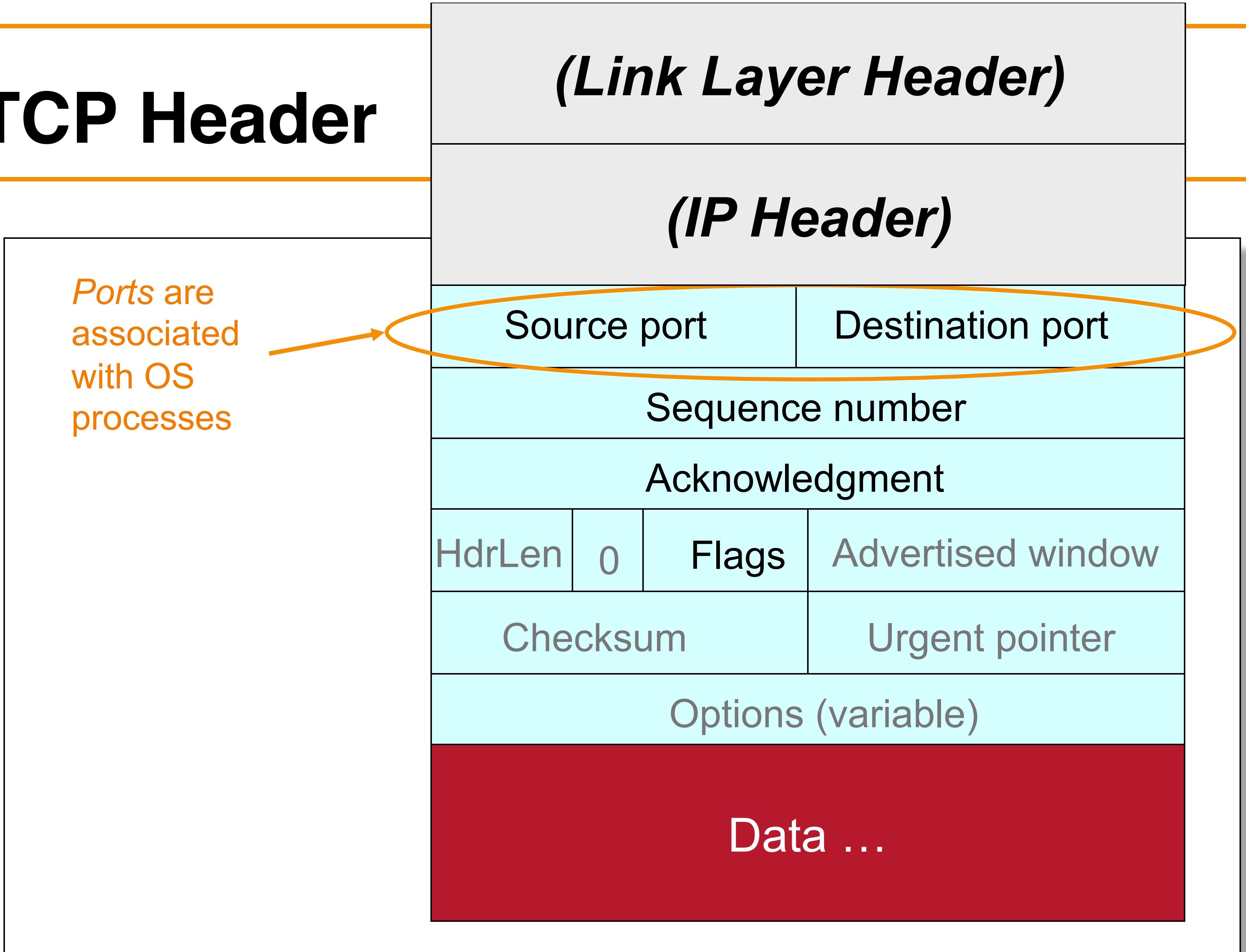
There are two separate **bytestreams**, one in each direction

Process A
on host H1

TCP Header


(Link Layer Header)
(IP Header)
Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data …

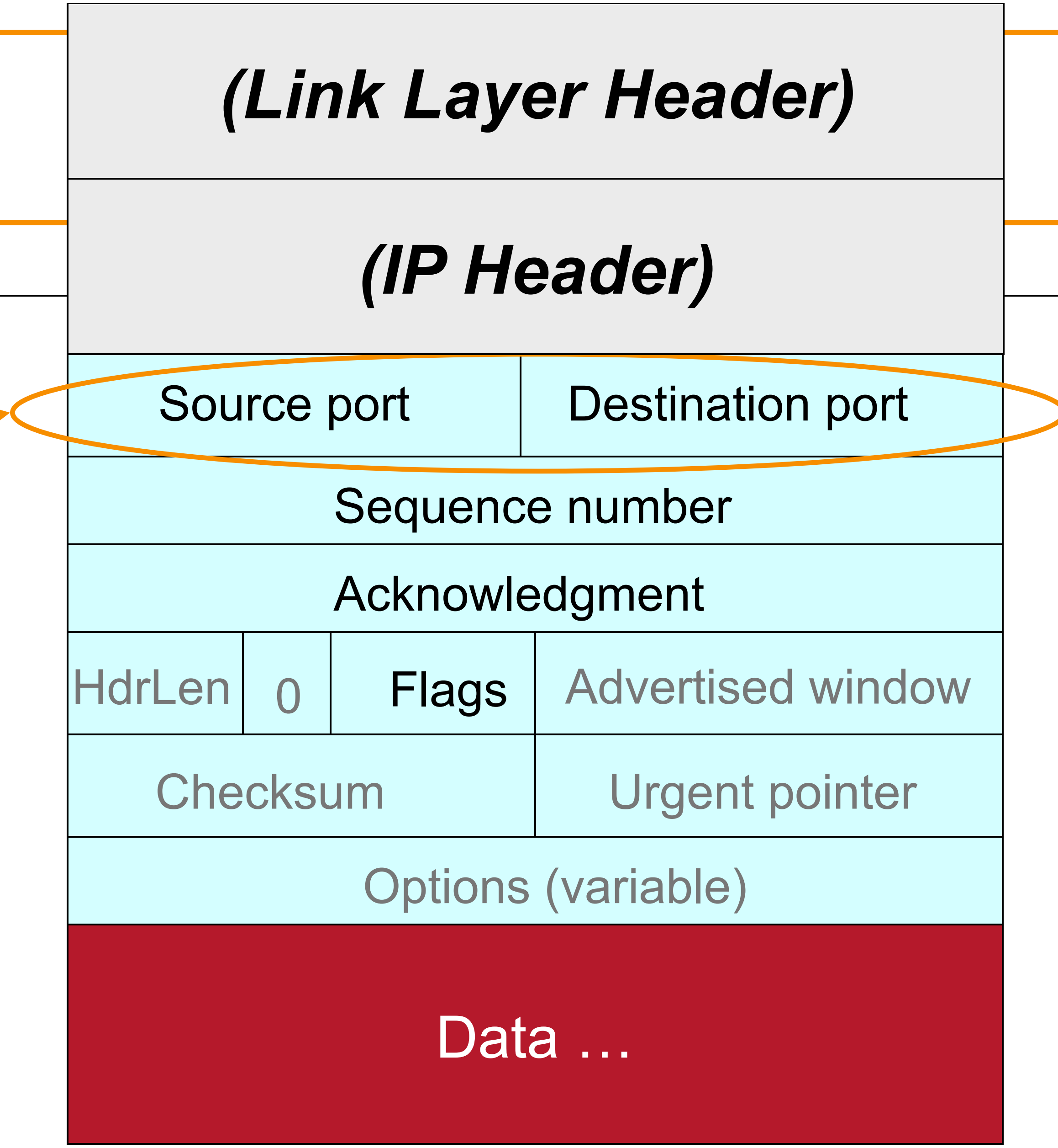
TCP Header



Ports are associated with OS processes
(Link Layer Header)
(IP Header)
Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data …

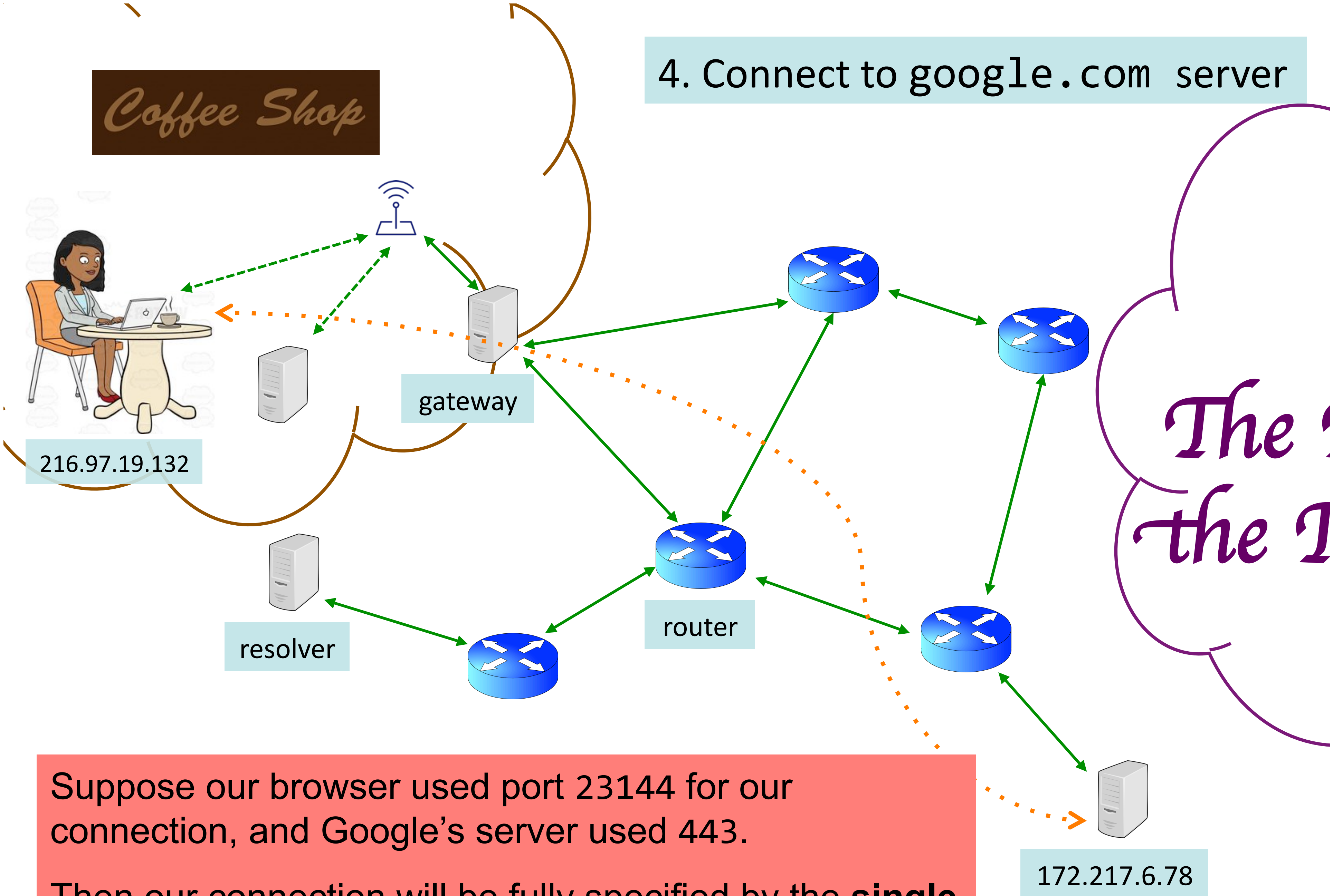
TCP Header

Ports are associated with OS processes



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

4. Connect to `google.com` server



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

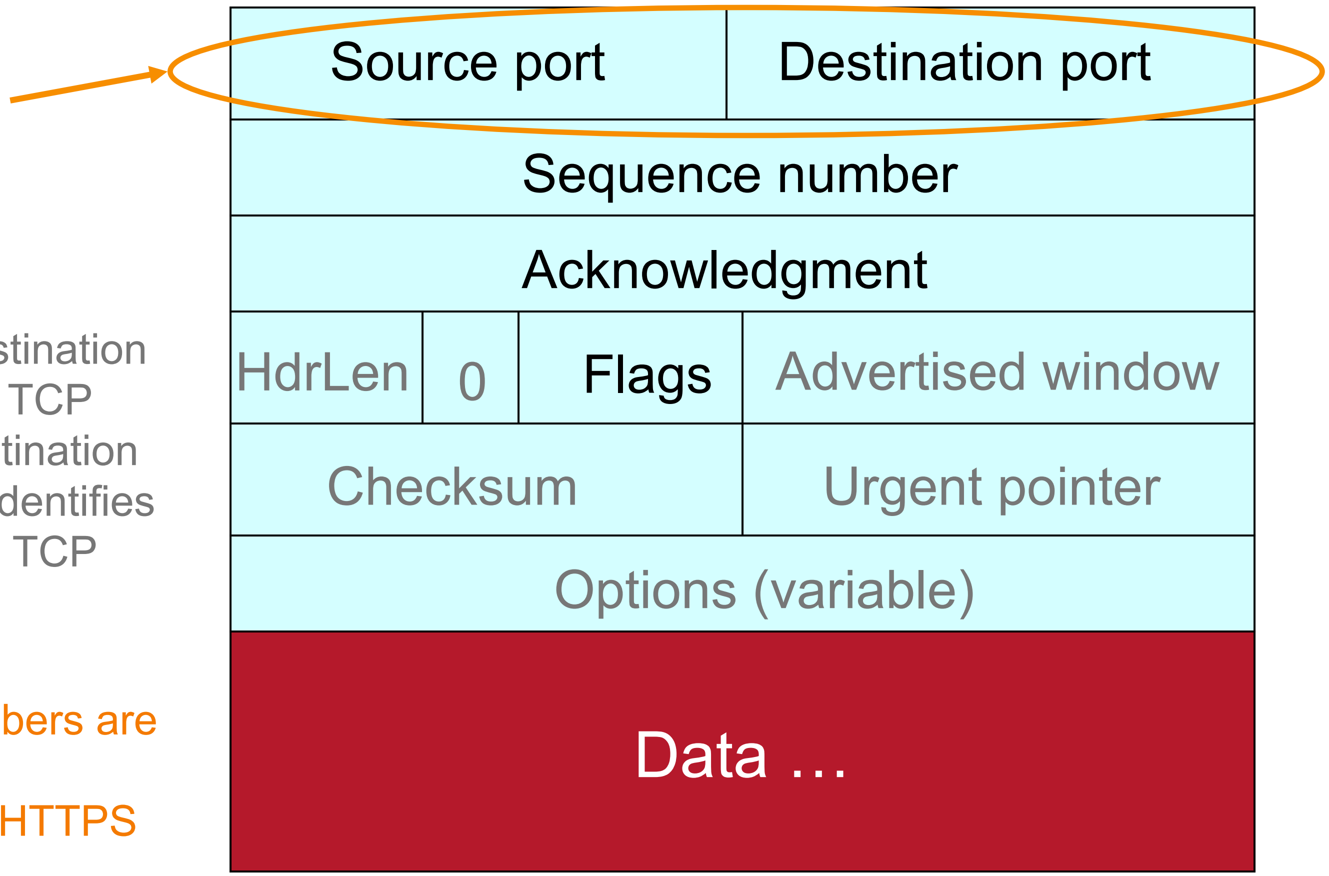
Then our connection will be fully specified by the **single** tuple <`216.97.19.132`, `23144`, `172.217.6.78`, `443`>

TCP Header

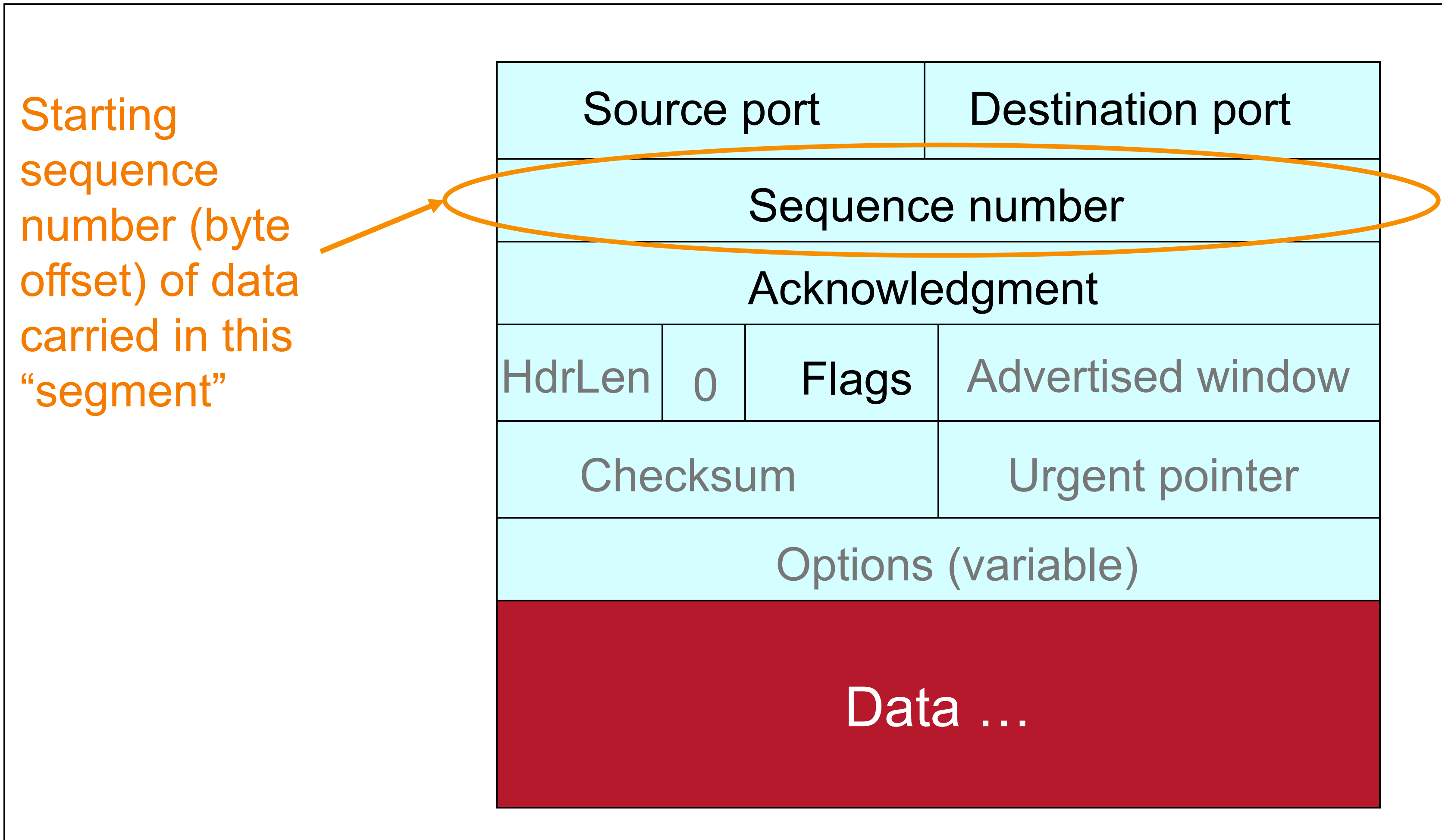
Ports are associated with OS processes

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

Some port numbers are "well known"
e.g. port 443 = HTTPS



TCP Header

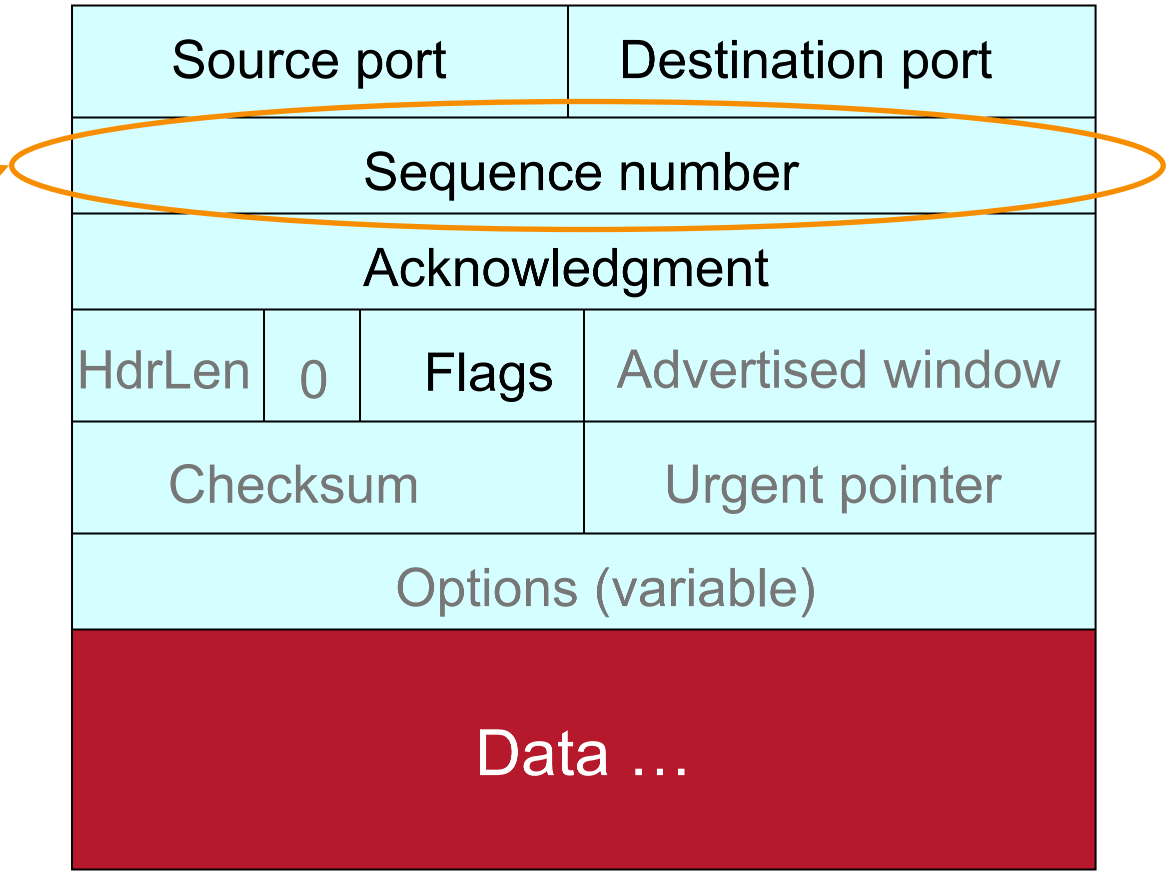


Starting sequence number (byte offset) of data carried in this "segment"
Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data …

TCP Header

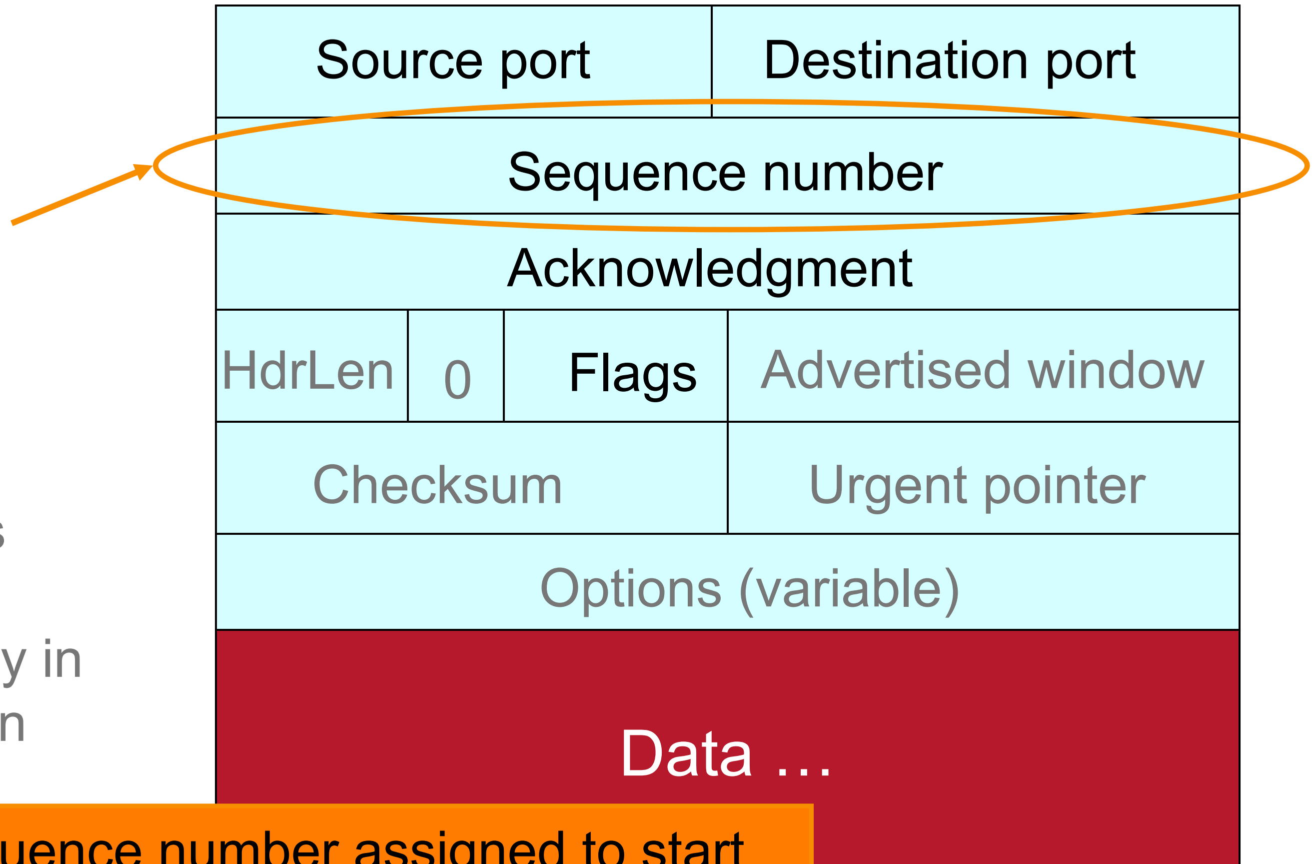
Starting sequence number (byte offset) of data carried in this “segment”

Byte streams numbered independently in each direction



TCP Header

Starting sequence number (byte offset) of data carried in this “segment”



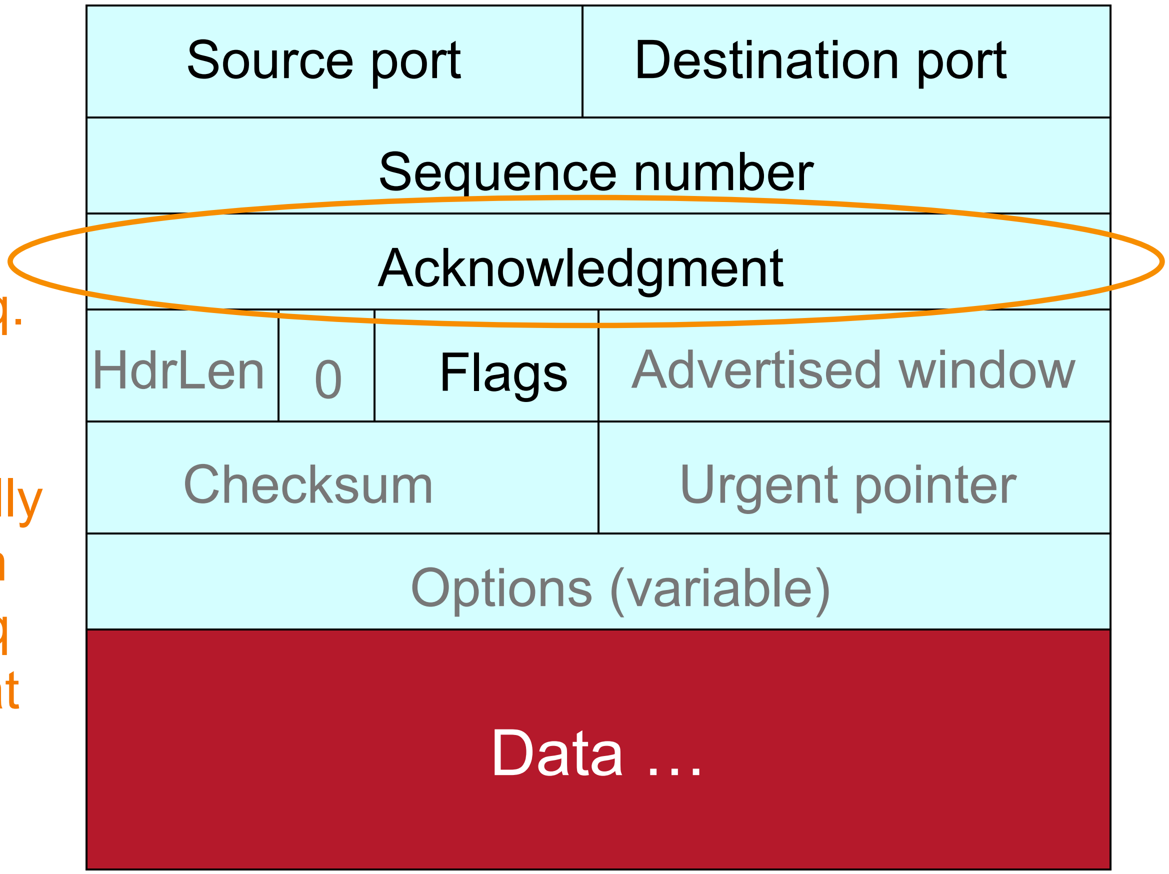
Byte streams numbered independently in each direction

Sequence number assigned to start of byte stream is picked when connection begins; **doesn’t** start at 0

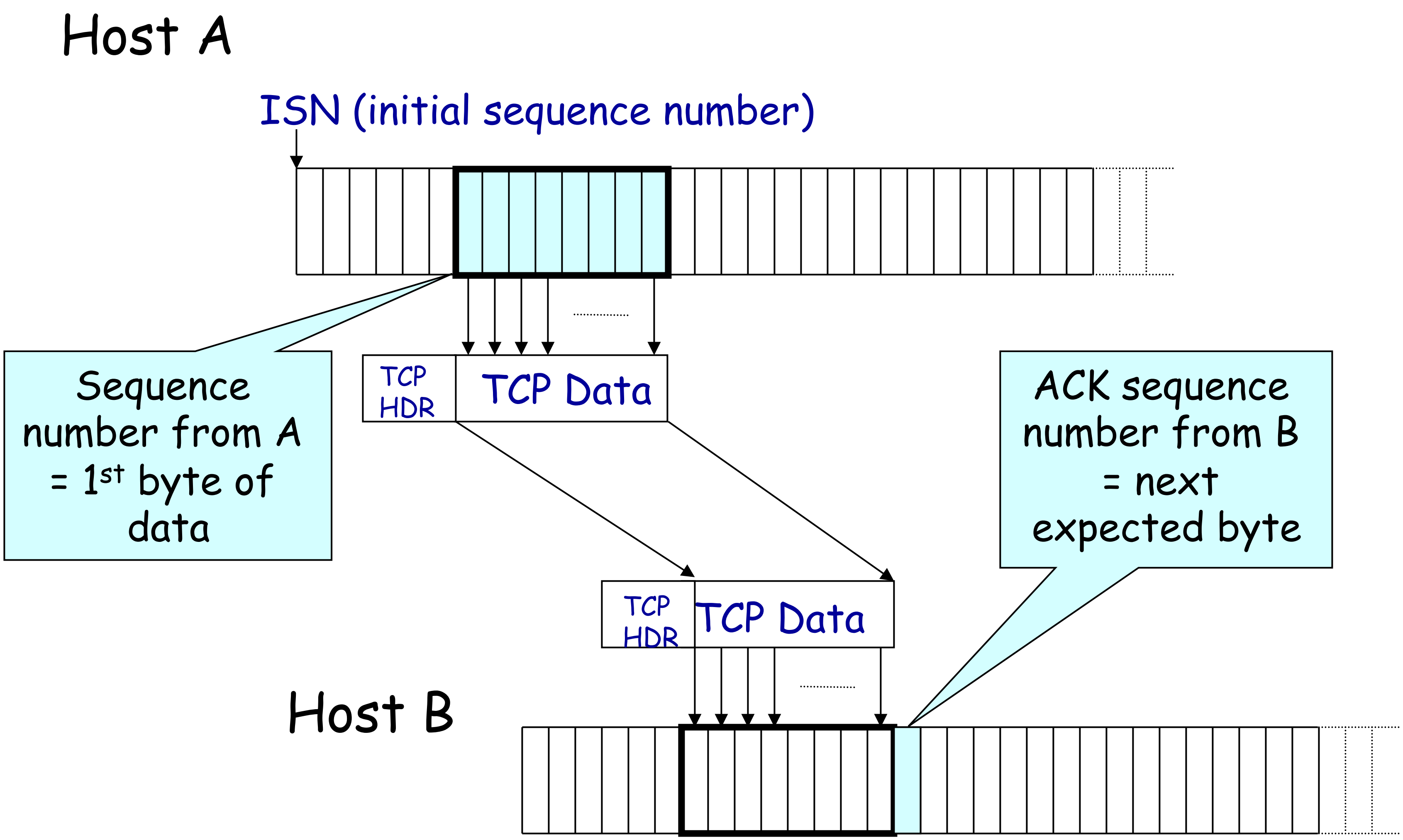
TCP Header

Acknowledgment gives seq # **just beyond** highest seq. received **in order**.

If sender successfully sends **N** bytestream bytes starting at seq **S** then "ack" for that will be **S+N**.



Sequence Numbers

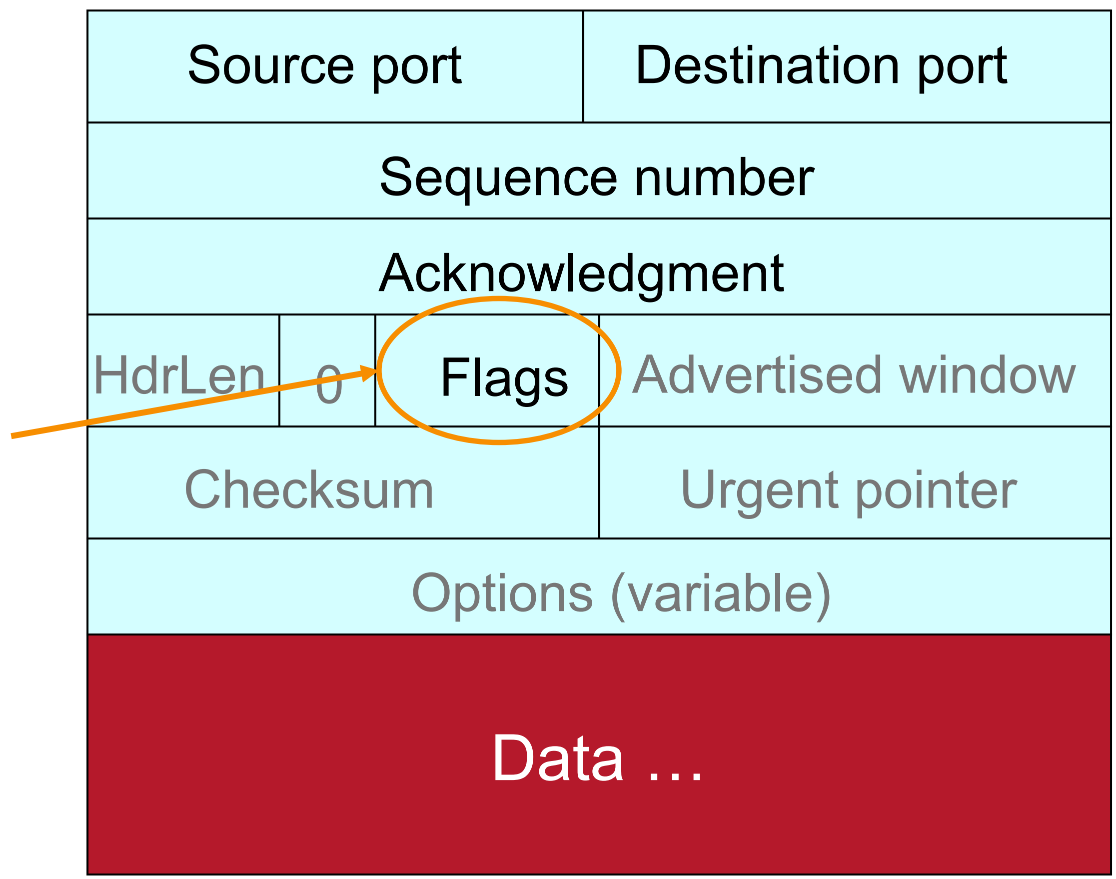


TCP Header

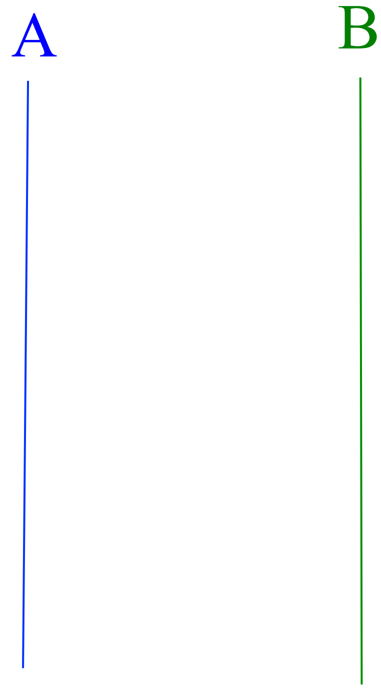
Uses include:

acknowledging data (“**ACK**”)

setting up (“**SYN**”) and closing connections (“**FIN**” and “**RST**”)

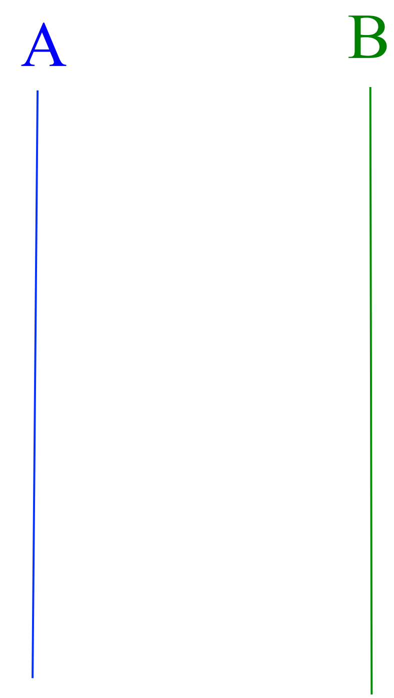


Establishing a TCP Connection



- *Three-way handshake* to establish connection

Establishing a TCP Connection

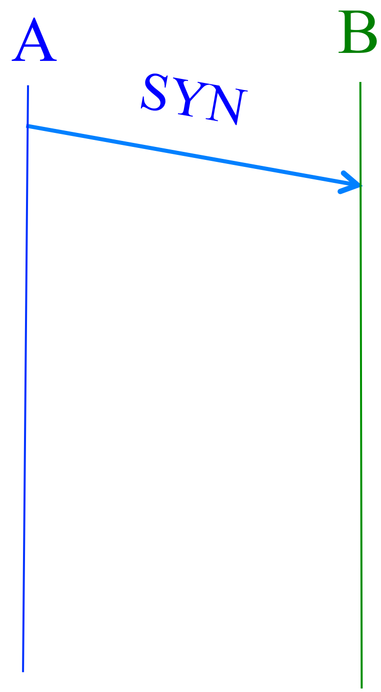


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on a clock)

- *Three-way handshake* to establish connection

Establishing a TCP Connection

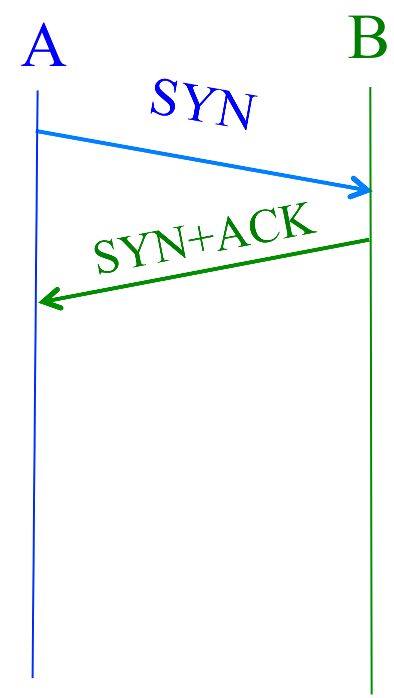


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(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

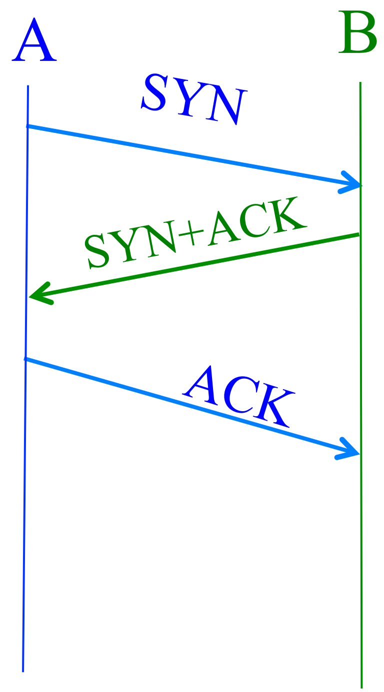


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(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

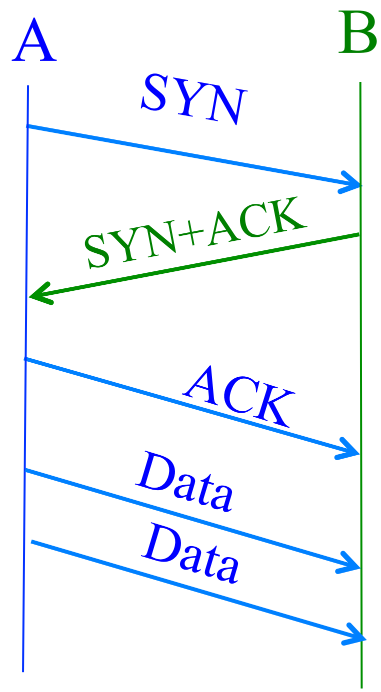


Each host tells its *Initial Sequence Number* (ISN) to the other host.

(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

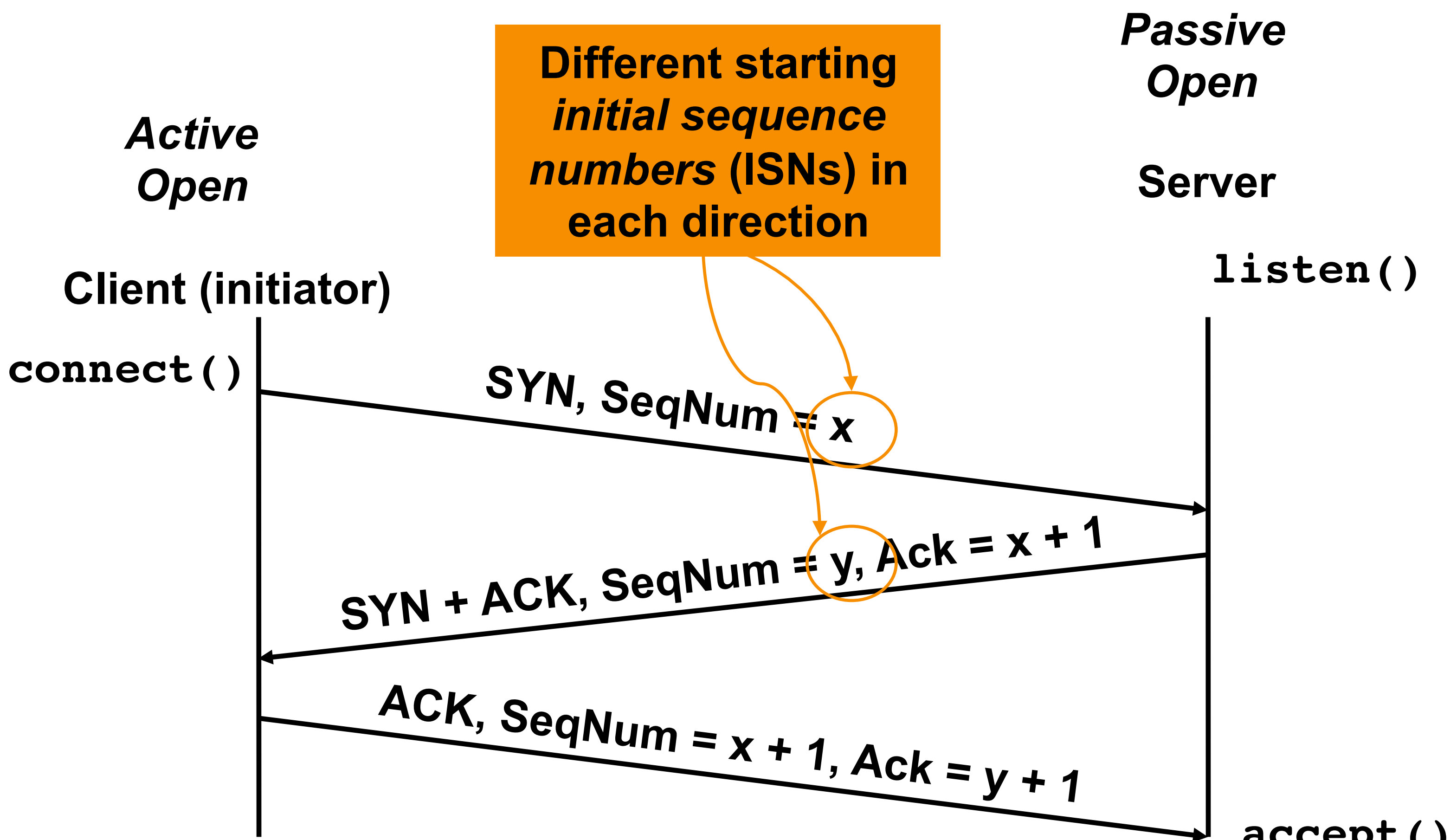


Each host tells its *Initial Sequence Number* (ISN) to the other host.

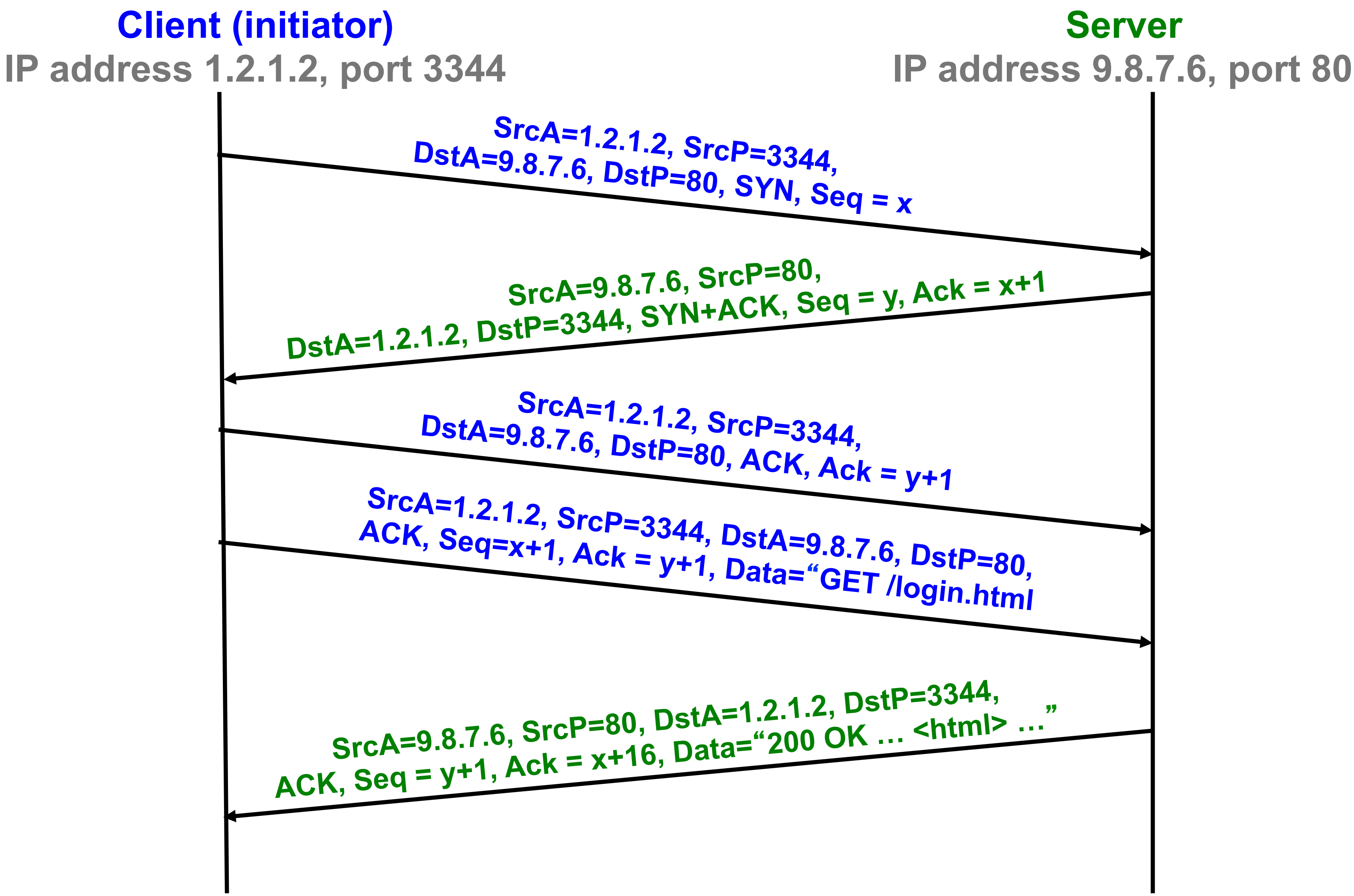
(Spec says to pick based on a clock)

- *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

Timing Diagram: 3-Way Handshaking

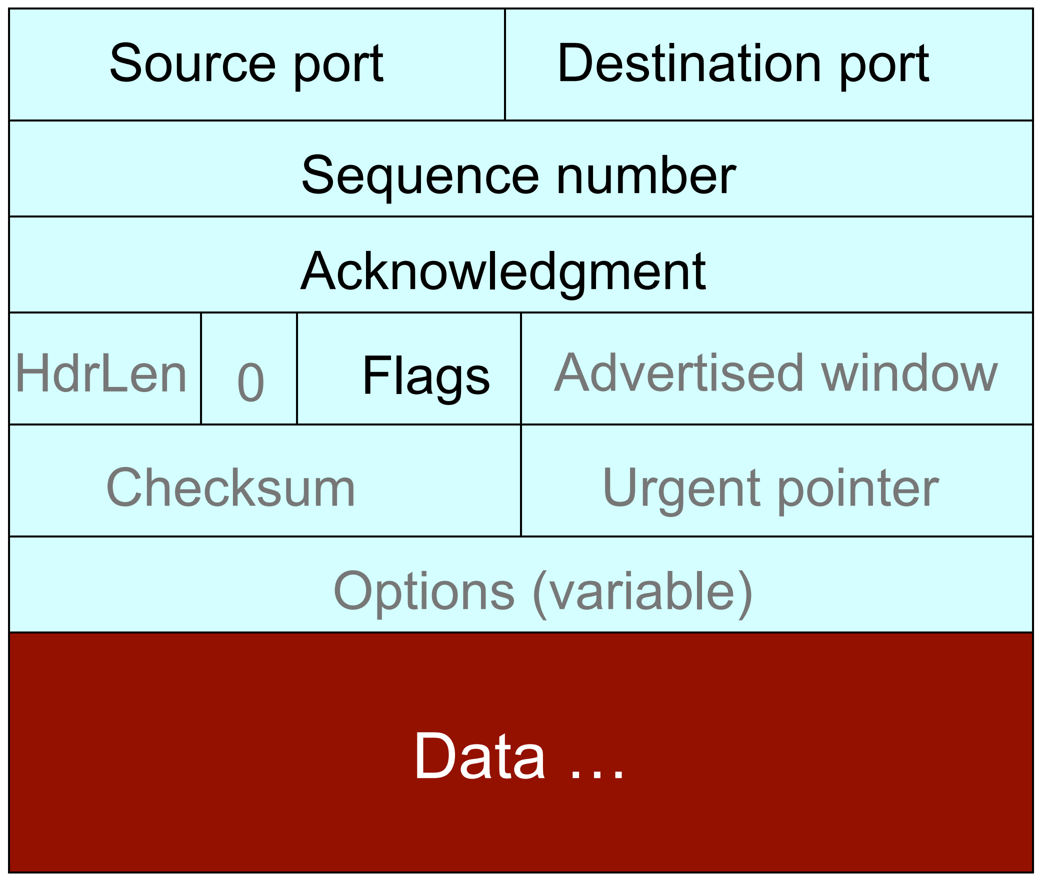


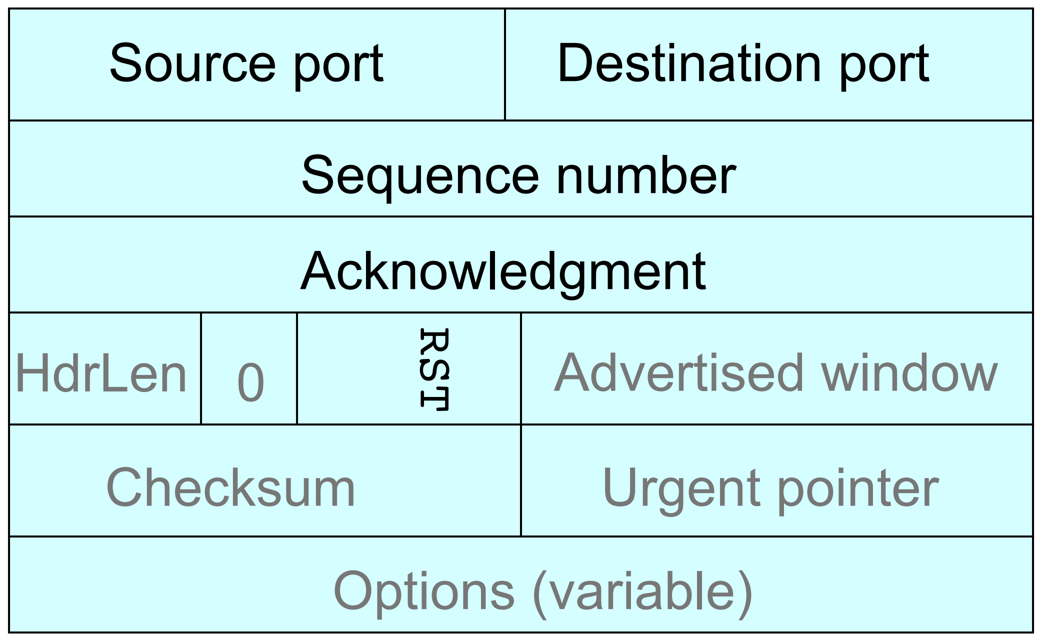
TCP Conn. Setup & Data Exchange



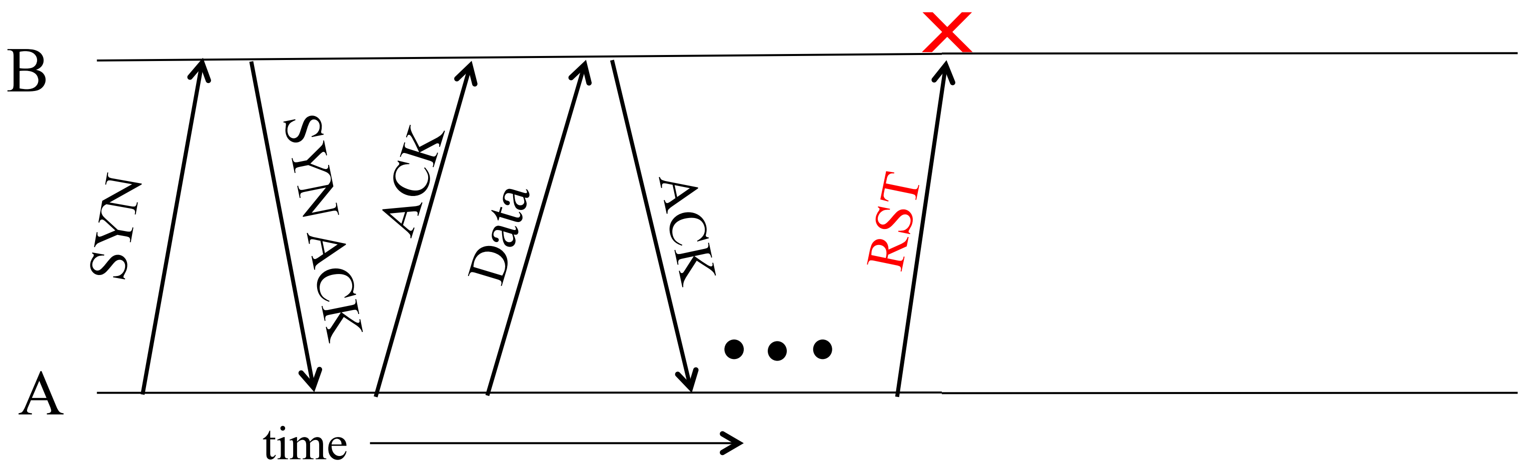
TCP Threat: Disruption

- 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


Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data …


Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
RST
Advertised window
Checksum
Urgent pointer
Options (variable)

Abrupt Termination

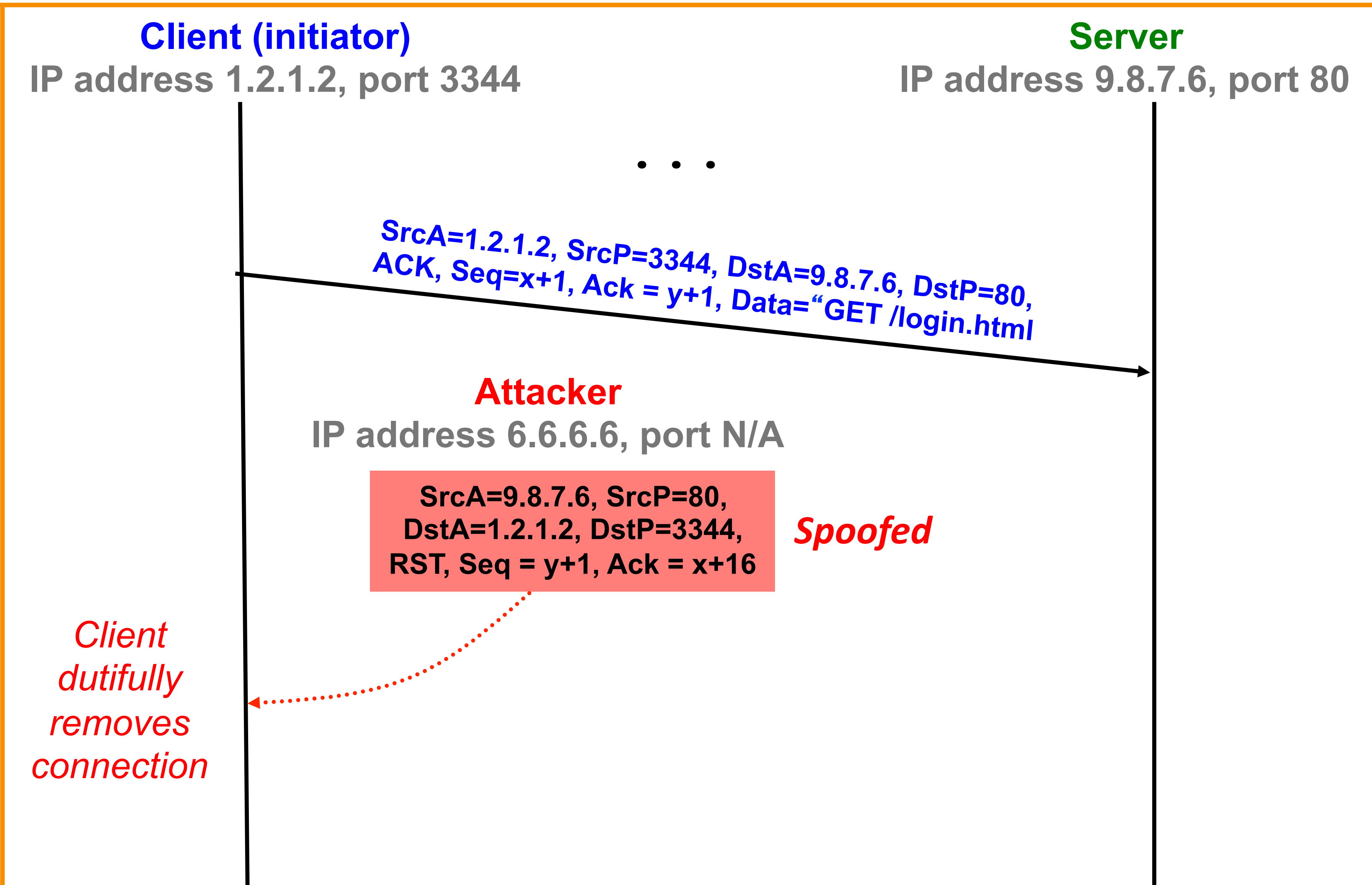


- A sends a TCP packet with RESET (**RST**) flag to B
  - E.g., because app. process on A crashed
  - (Could instead be that B sends a RST to A)
- Assuming that the sequence numbers in the **RST** fit with what B expects, That's It:
  - B's user-level process receives: `ECONNRESET`
  - No further communication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

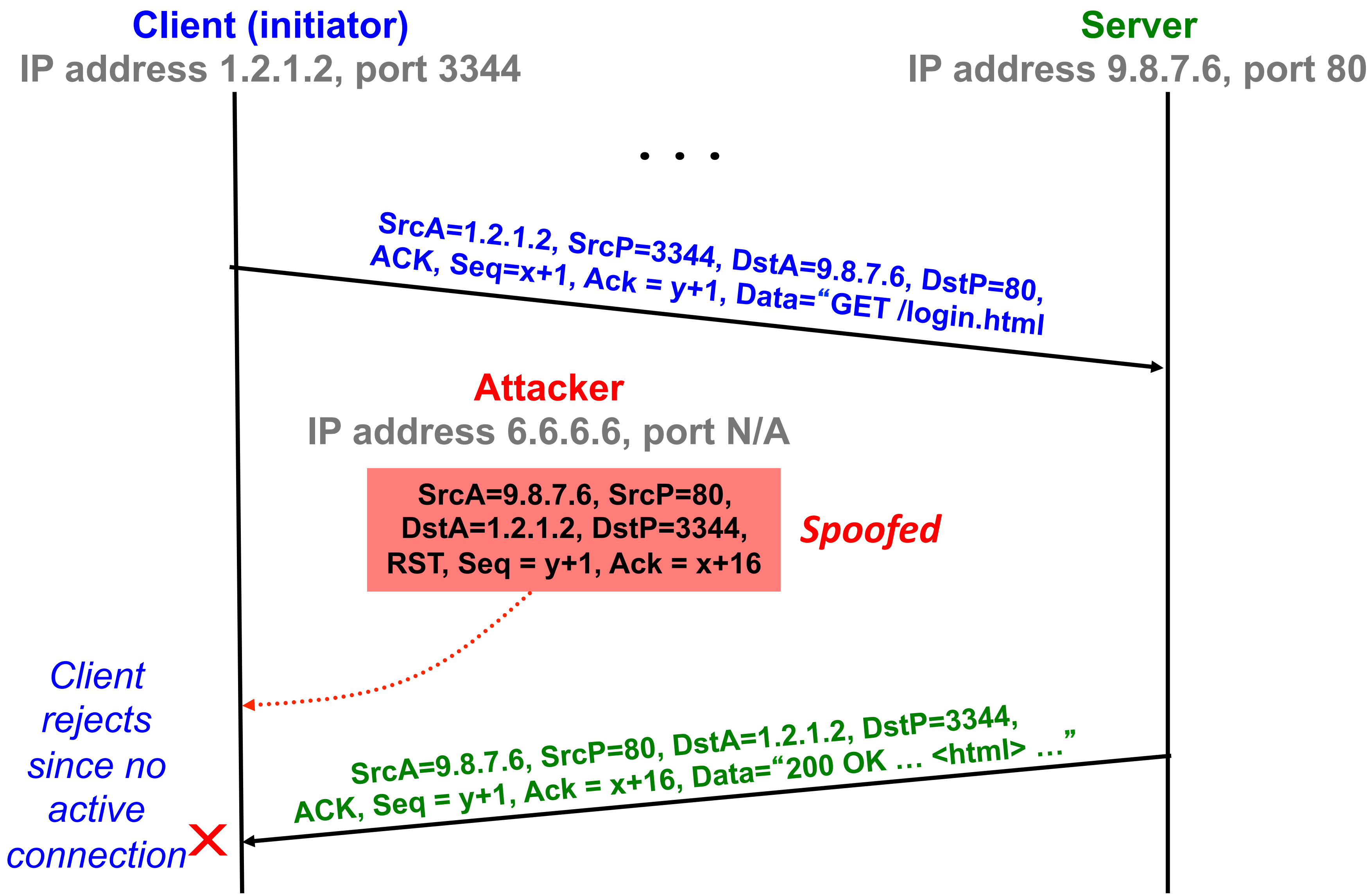
TCP Threat: Disruption

- 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 ports & sequence numbers, can disrupt any TCP connection

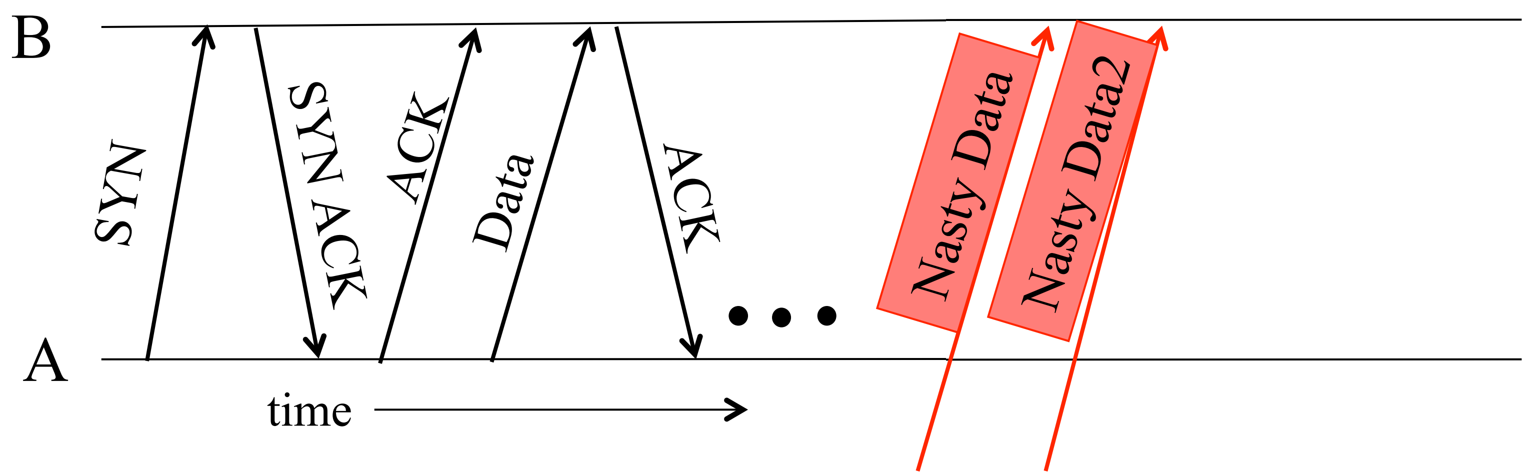
TCP *RST Injection*



TCP *RST Injection*

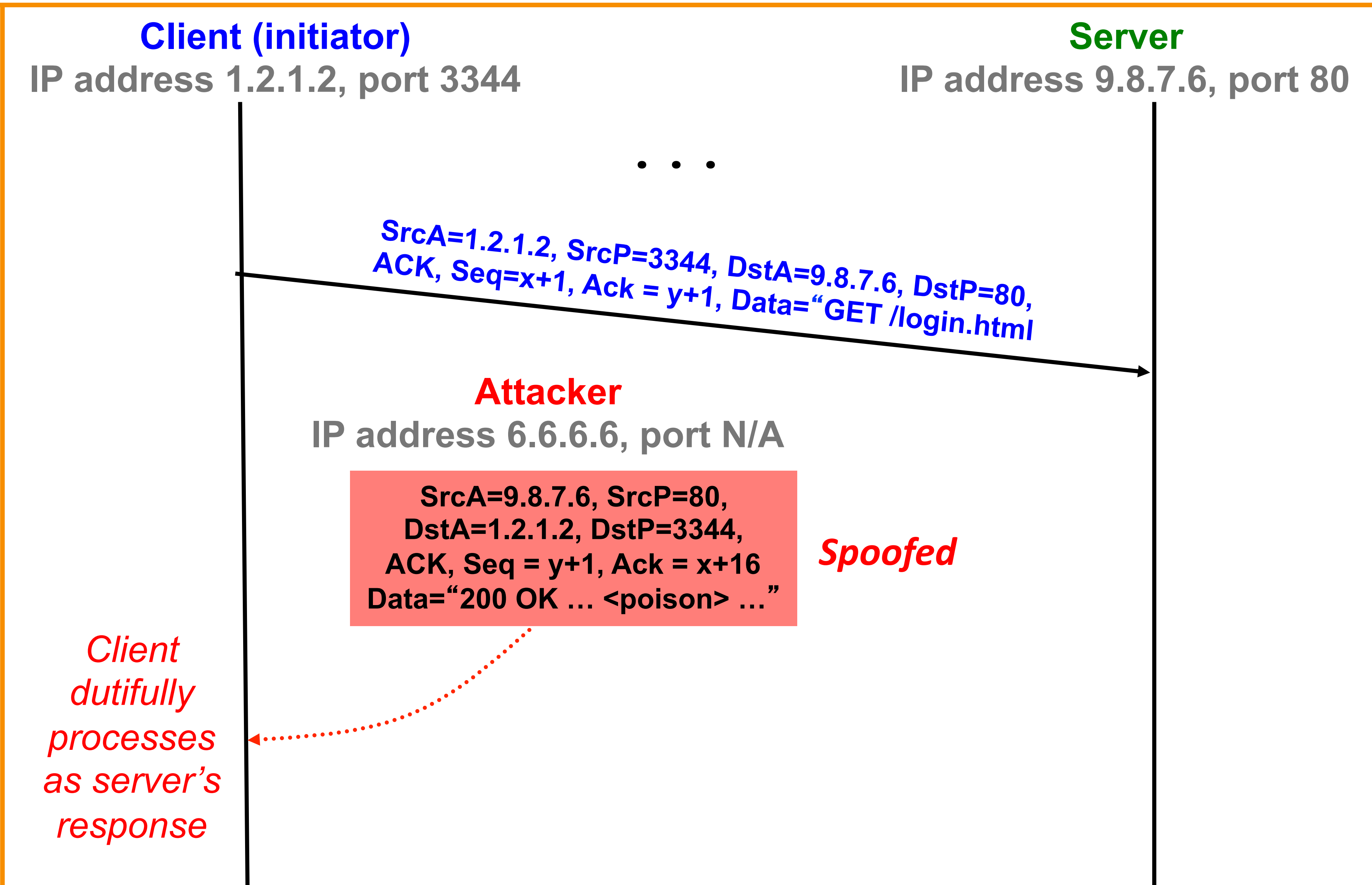


TCP Threat: Data Injection

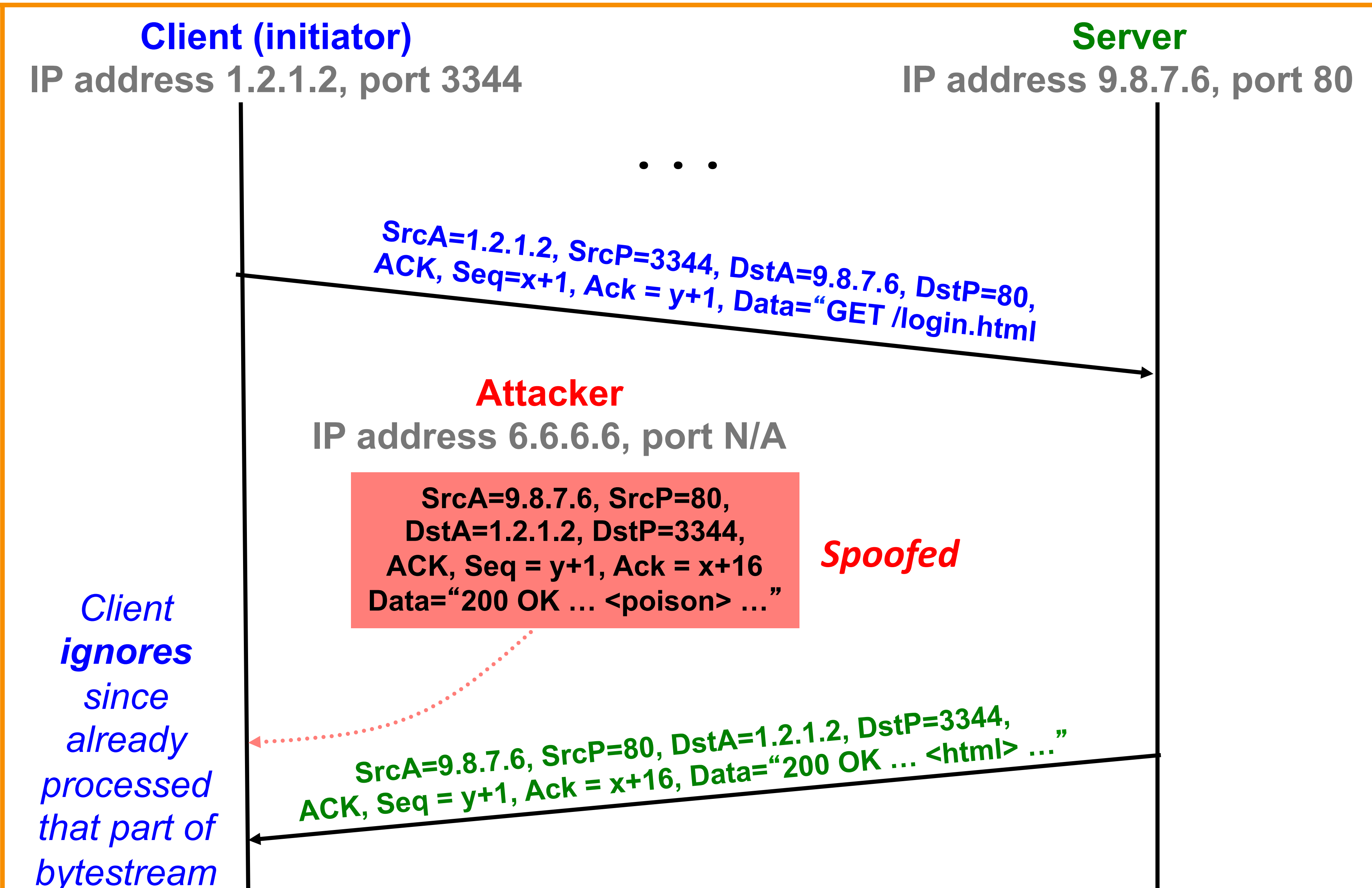


- What about inserting data rather than disrupting a connection?
  - Again, all that's required is attacker knows correct ports, seq. numbers
  - Receiver B is *none the wiser!*
- Termed TCP connection hijacking (or "*session hijacking*")
  - A general means to take over an already-established connection!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

TCP *Data Injection*



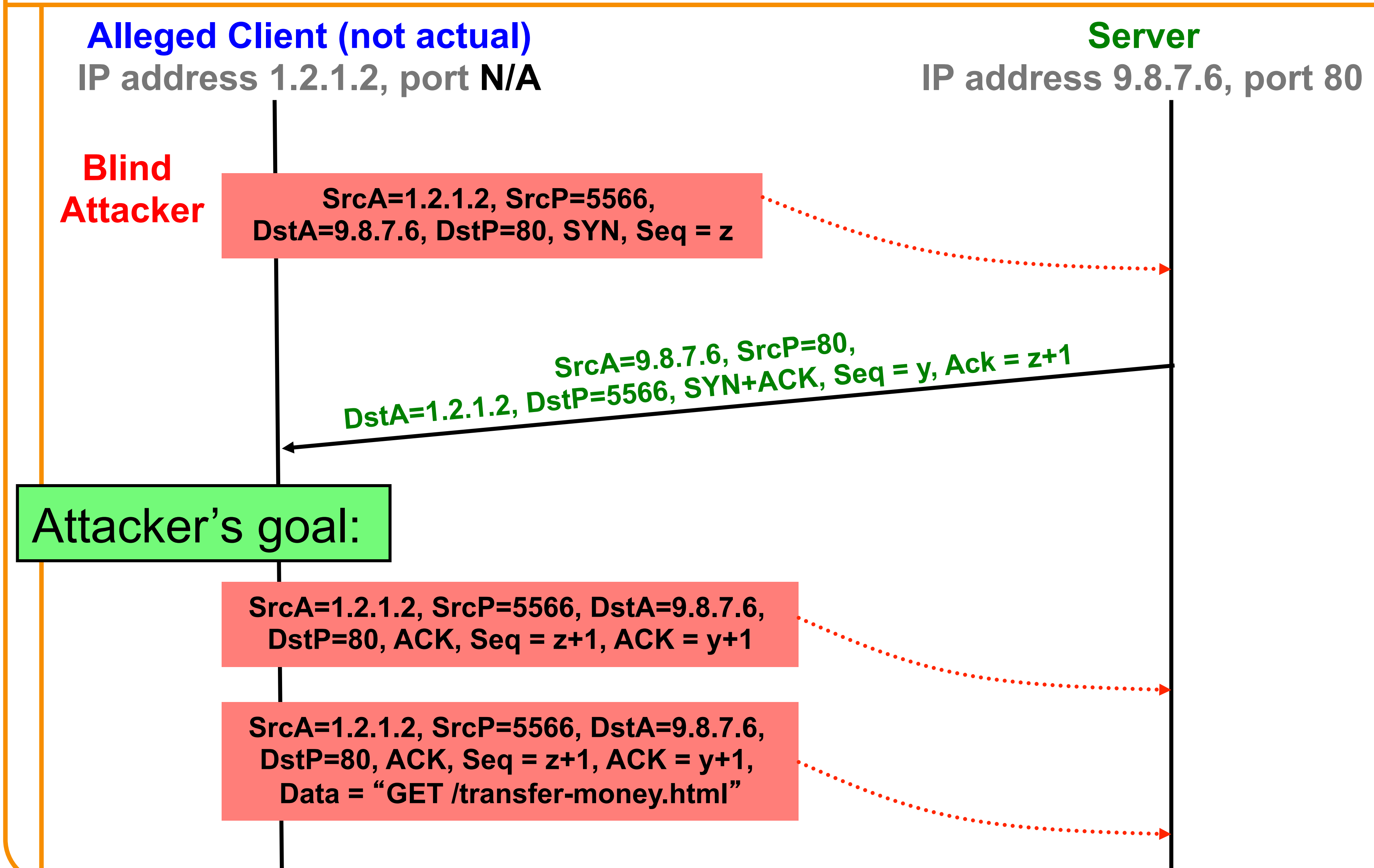
TCP *Data Injection*



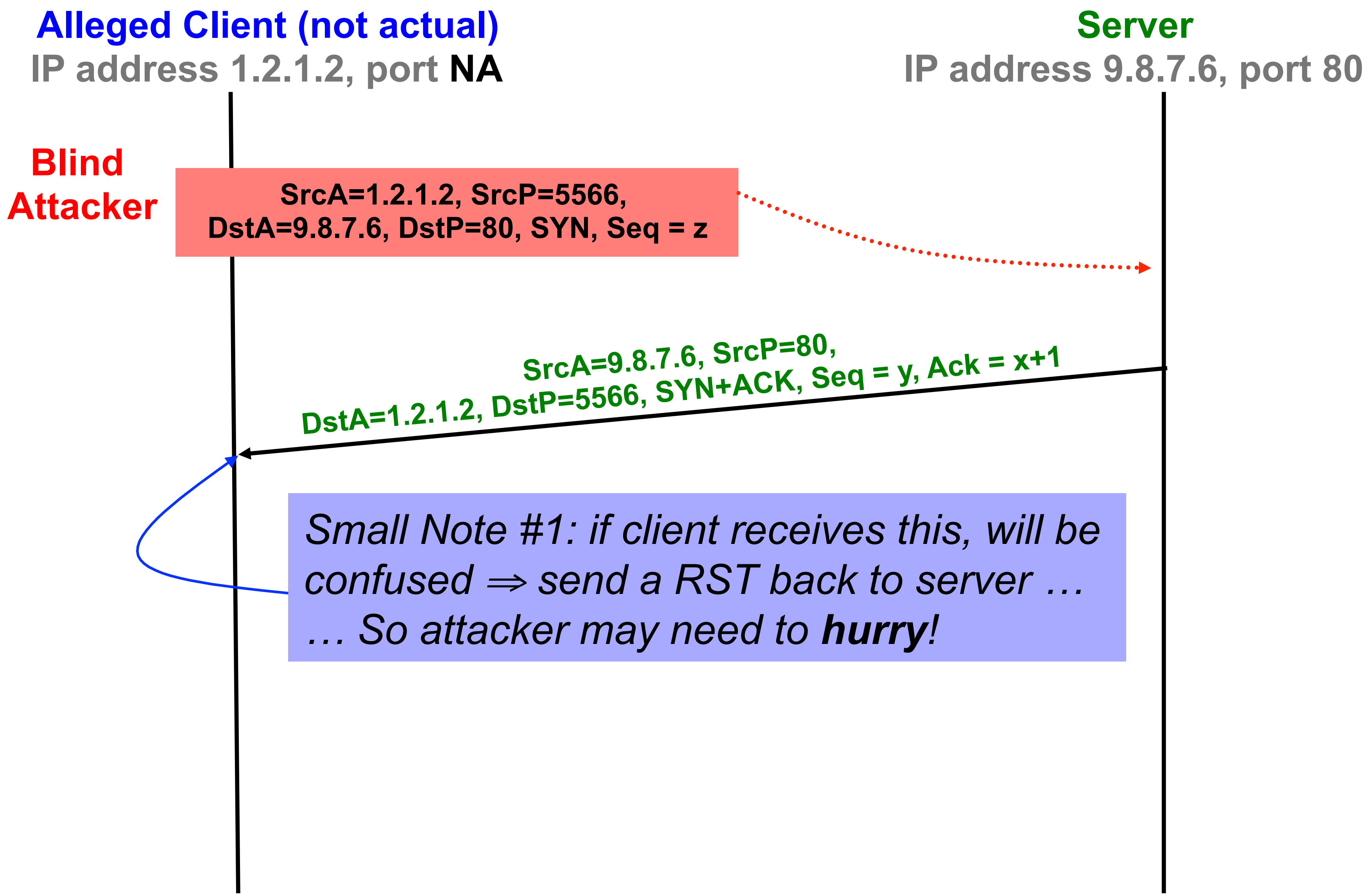
TCP Threat: Blind Spoofing

- 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
- Let's look at a simpler related attack where the goal of the attacker is to create a fake connection, rather than inject into a real one
  - Why?
  - Perhaps to leverage a server's trust of a given client as identified by its IP address
  - Perhaps to frame a given client so the attacker's actions during the connections can't be traced back to the attacker

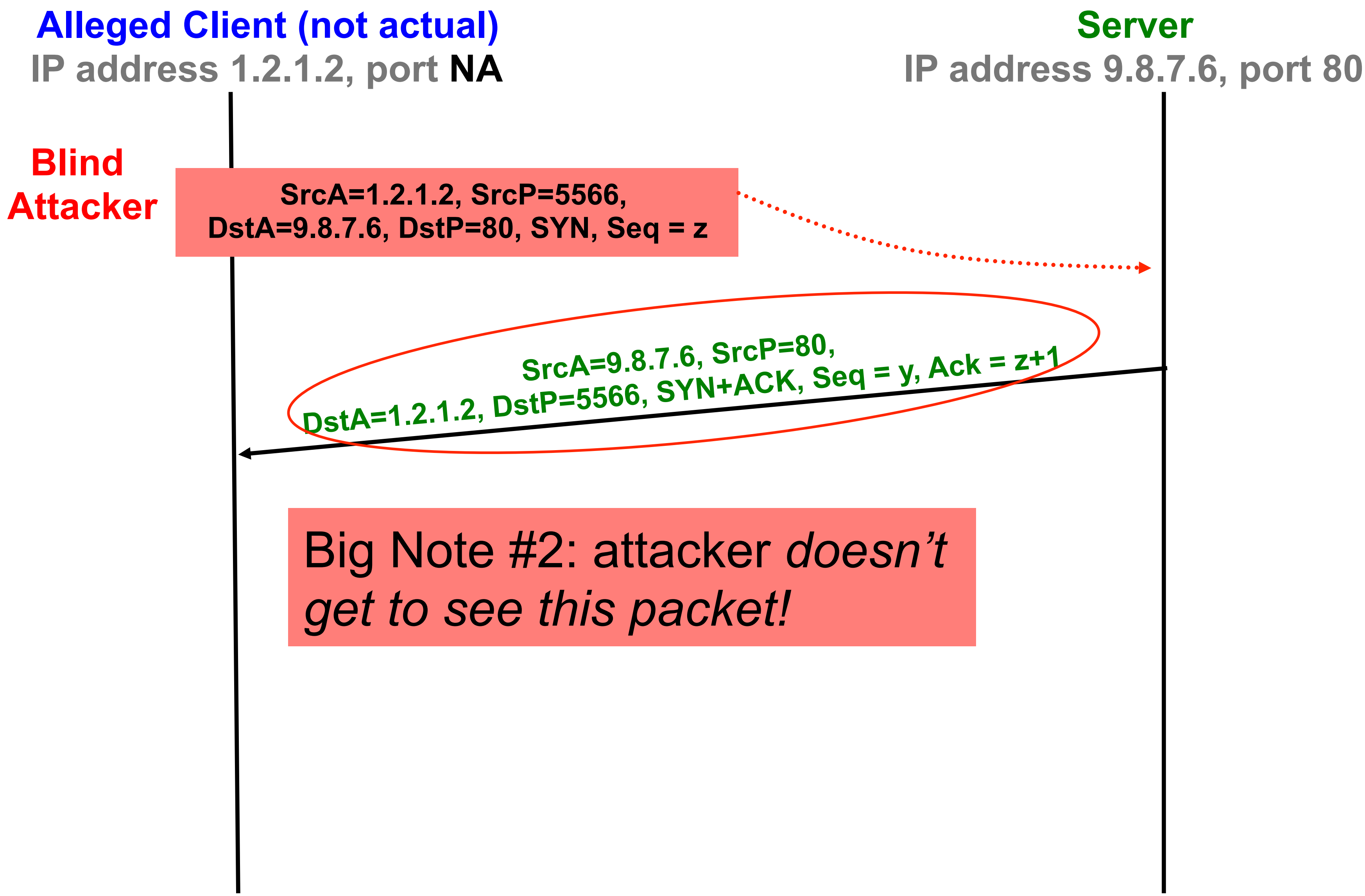
Spoofing an Entire TCP Connection



Spoofing an Entire TCP Connection

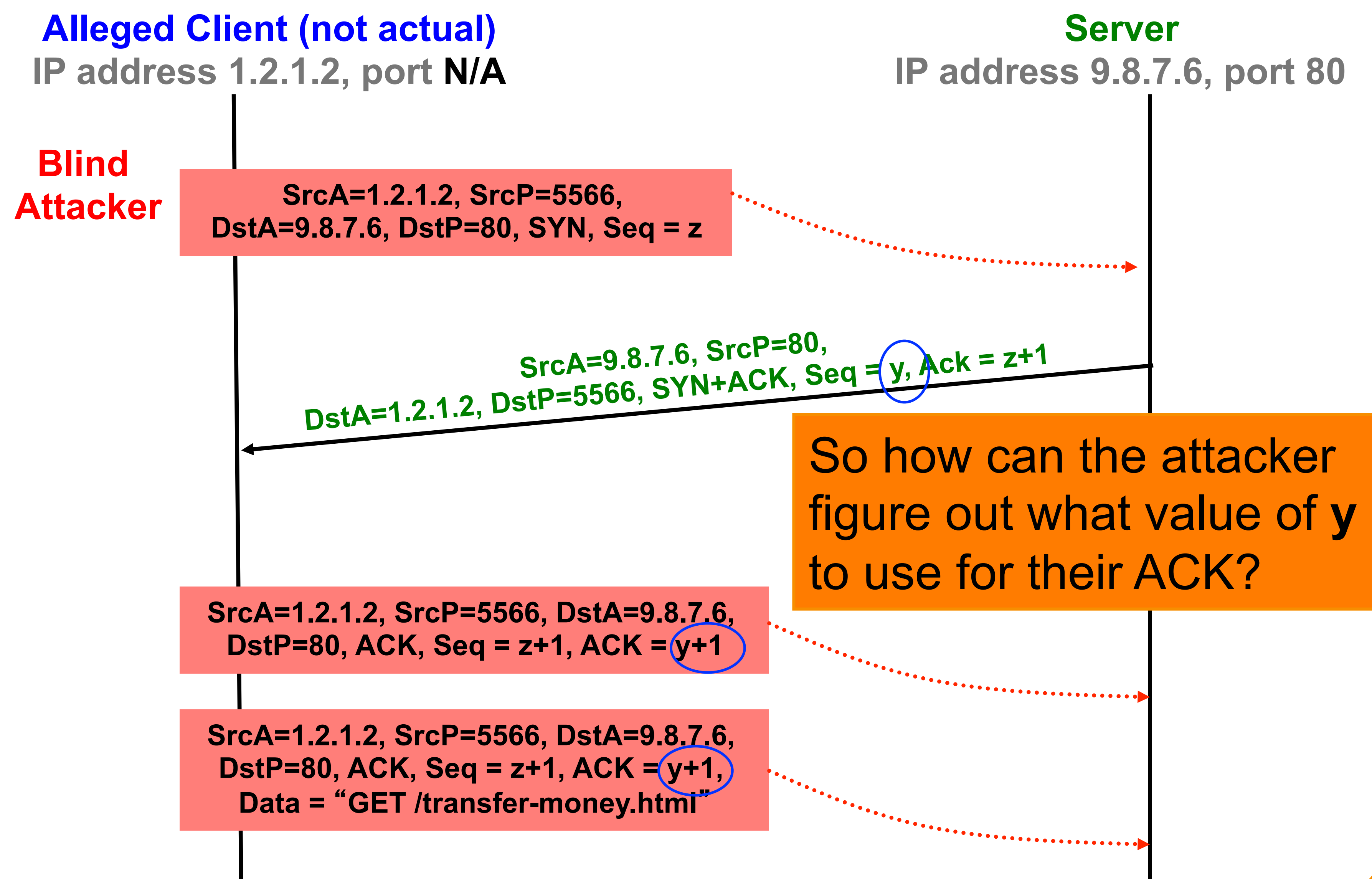


Spoofing an Entire TCP Connection

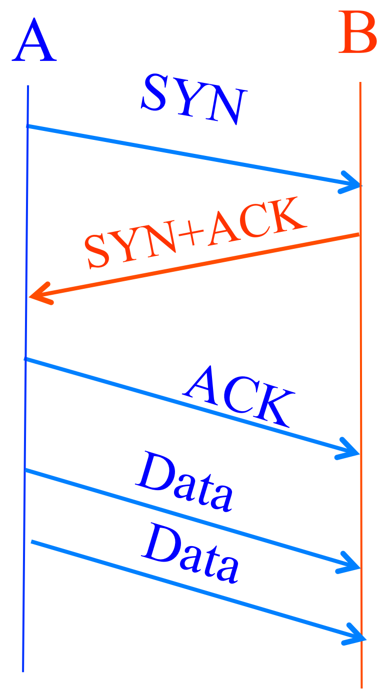


Big Note #2: attacker *doesn't get to see this packet!*

Spoofing an Entire TCP Connection



Reminder: Establishing a TCP Connection



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

(Spec says to pick based on a clock)

Hmm, any way for the attacker to know ***this?***

How Do We Fix This?

Use a (Pseudo)-*Random* ISN

Sure - make a non-spoofed connection *first*, and see what server used for ISN y then!

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully **terminate** by forging a RST packet
  - **Inject** (*spoof*) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - *Remains a major threat today*

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully **terminate** by forging a RST packet
  - **Inject** (*spoof*) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - *Remains a major threat today*
- An attacker who can predict the ISN chosen by a server can "blind spoof" a connection to the server
  - Makes it appear that host ABC has connected, and has sent data of the attacker's choosing, when in fact it hasn't
  - *Undermines any security based on trusting ABC's IP address*
  - Allows attacker to "frame" ABC or otherwise avoid detection
  - **Fixed** (mostly) today by choosing **random** ISNs

5 Minute Break

Questions Before We Proceed?

DNS: Operation & Threats

Host Names vs. IP addresses

- Host names
  - Examples: `www.cnn.com` and `bbc.co.uk`
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provide little (if any) information about location
- IP addresses
  - Examples: `64.236.16.20` and `212.58.224.131`
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location

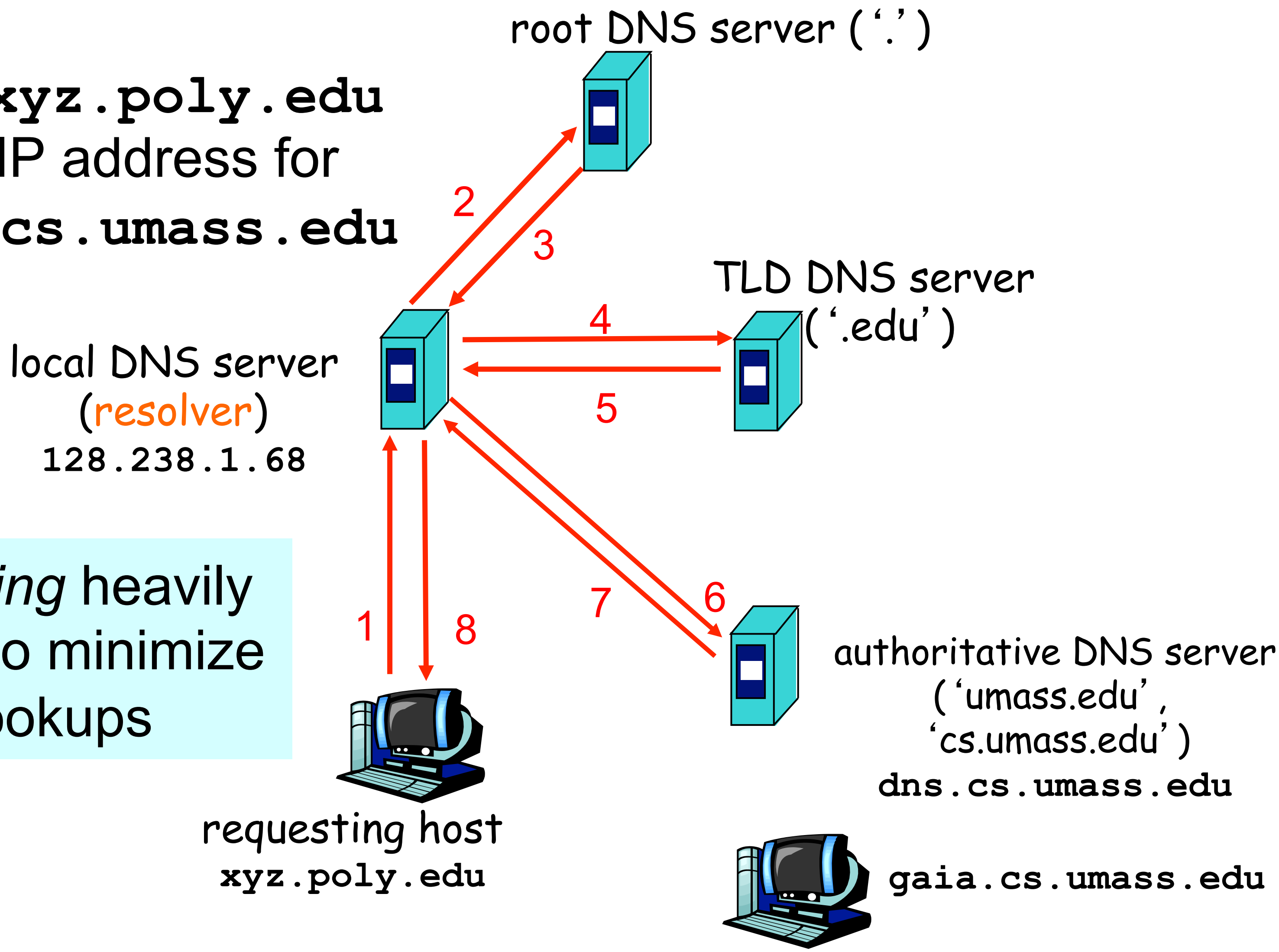
Mapping Names to Addresses

- Domain Name System (DNS)
  - Hierarchical name space divided into sub-trees ("*zones*")
    - *E.g. .edu, .berkeley.edu, .eecs.berkeley.edu*
  - Zones distributed over collection of DNS *name servers*
- Hierarchy of DNS servers
  - Root (hardwired into other servers)
  - Top-level domain (TLD) servers
    - E.g. .com, .org, .net, .uk, .biz
  - "Authoritative" DNS servers (e.g. for *facebook.com*)
- End systems configured with IP address of a *resolver* to contact for their lookups

DNS Lookups via a *Resolver*

Host at **`xyz.poly.edu`** wants IP address for **`gaia.cs.umass.edu`**

Caching heavily used to minimize lookups



DNS Threats

- DNS: path-critical for just about everything we do
  - Maps hostnames ⇔ IP addresses
  - Design only **scales** if we can minimize lookup traffic
    - #1 way to do so: caching
    - #2 way to do so: return not only answers to queries, but additional info that will likely be needed shortly
- What if attacker eavesdrops on our DNS queries?
  - Simple to then redirect us w/ **spoofed** misinformation
- Consider attackers who *can't* eavesdrop - but still aim to manipulate us via *how the protocol functions*
- Directly interacting w/ DNS: **`dig`** program on Unix
  - Allows querying of DNS system
  - Dumps each field in DNS responses

dig eecs.mit.edu A

Use Unix "`dig`" utility to look up IP address ("`A`") for hostname `eecs.mit.edu` via DNS

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

This is `dig` identifying its version and the query it is attempting to look up

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Status values returned from the remote name server queried by `dig`

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           2160

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Including a 16-bit **transaction identifier** that enables the DNS client (`dig`, in this case) to match up the reply with its original request

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                                        0NS.mit.edu.
mit.edu.                                        RAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

The name server echoes back the question that it is answering as the first part of its reply

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode
;; flags: qr rd ra; QU                                   IONAL: 3
```

> "**Answer**" tells us the IP address associated with `eecs.mit.edu` is `18.62.1.6` and we can *cache* the result for 21,600 seconds

```
;; QUESTION SECTION:
;eecs.mit.edu.                IN      A

;; ANSWER SECTION:
eecs.mit.edu.         21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.              11088   IN      NS      BITSY.mit.edu.
mit.edu.              11088   IN      NS      W20NS.mit.edu.
mit.edu.              11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.       126738  IN      A       18.71.0.151
BITSY.mit.edu.        166408  IN      A       18.72.0.3
W20NS.mit.edu.        126738  IN      A       18.70.0.160
```