Computer Network Security

ECE 4112/6612

CS 4262/6262

Prof. Frank Li

Logistics

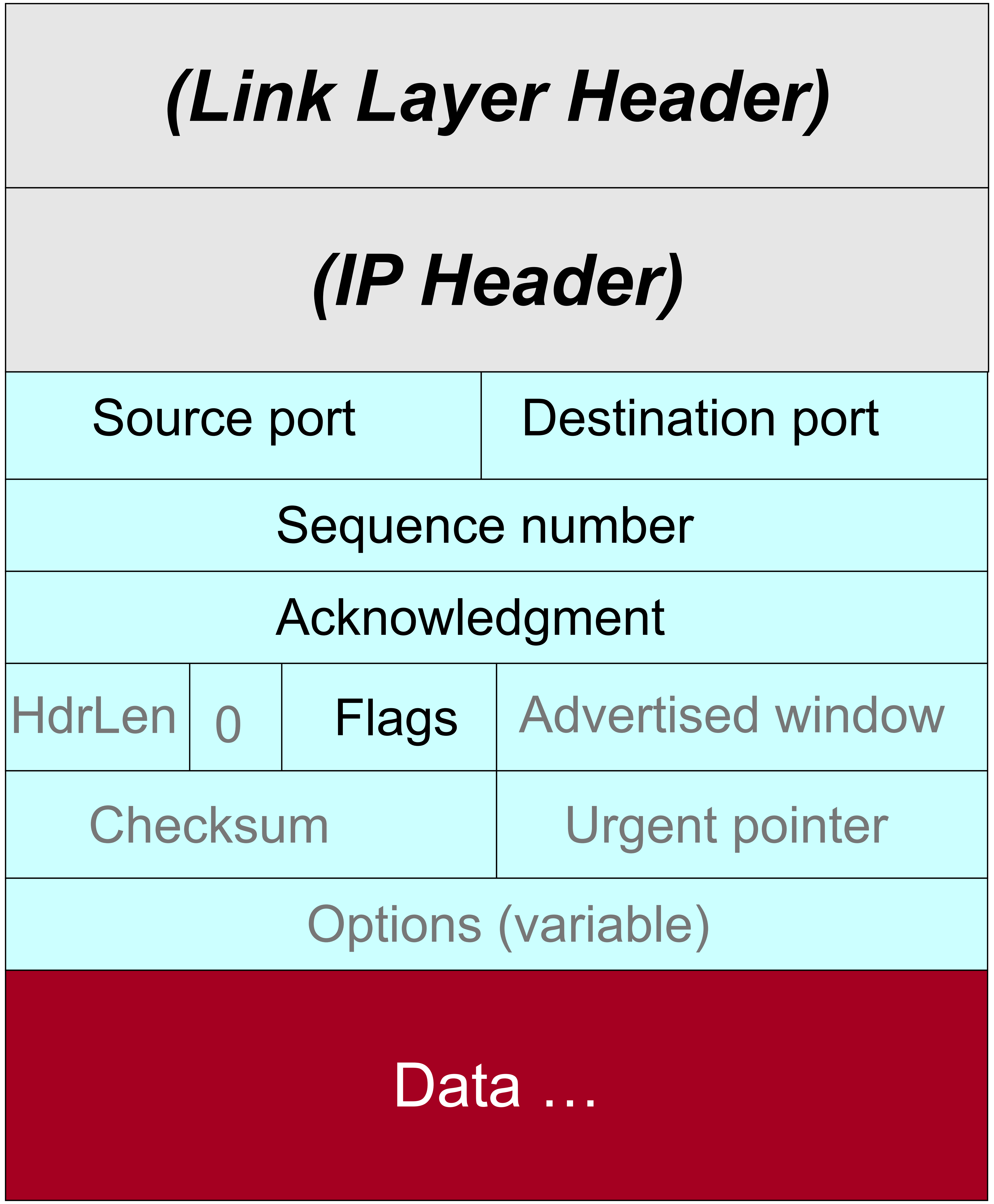
HW1 regrade requests close **tonight** midnight.

Quiz 1 scores to be released tomorrow. Regrade requests open until next **Tuesday, Oct 3, midnight.**

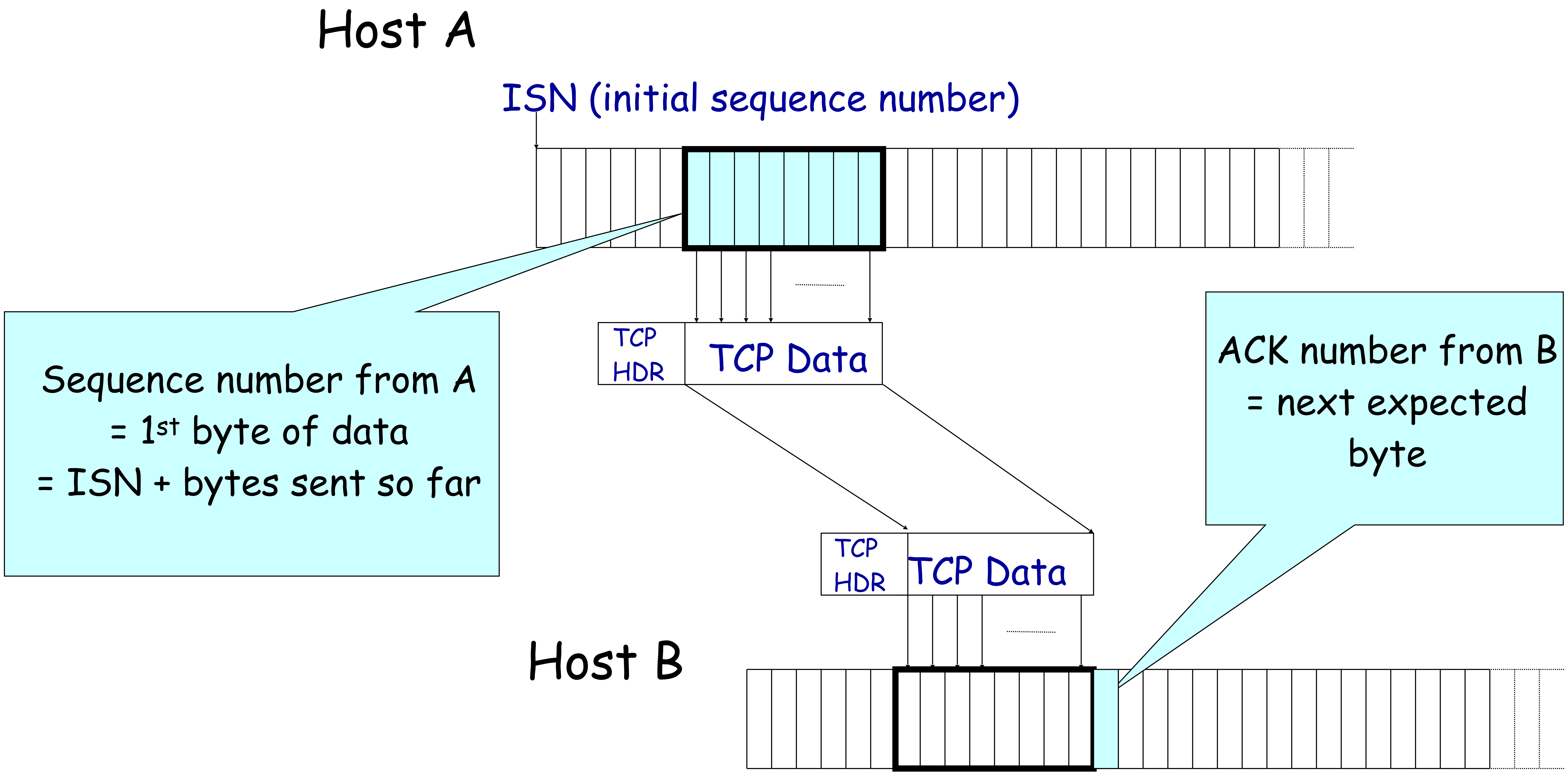
Project proposals due this **Friday, midnight**.

Wrapping up from last time:
TCP + UDP

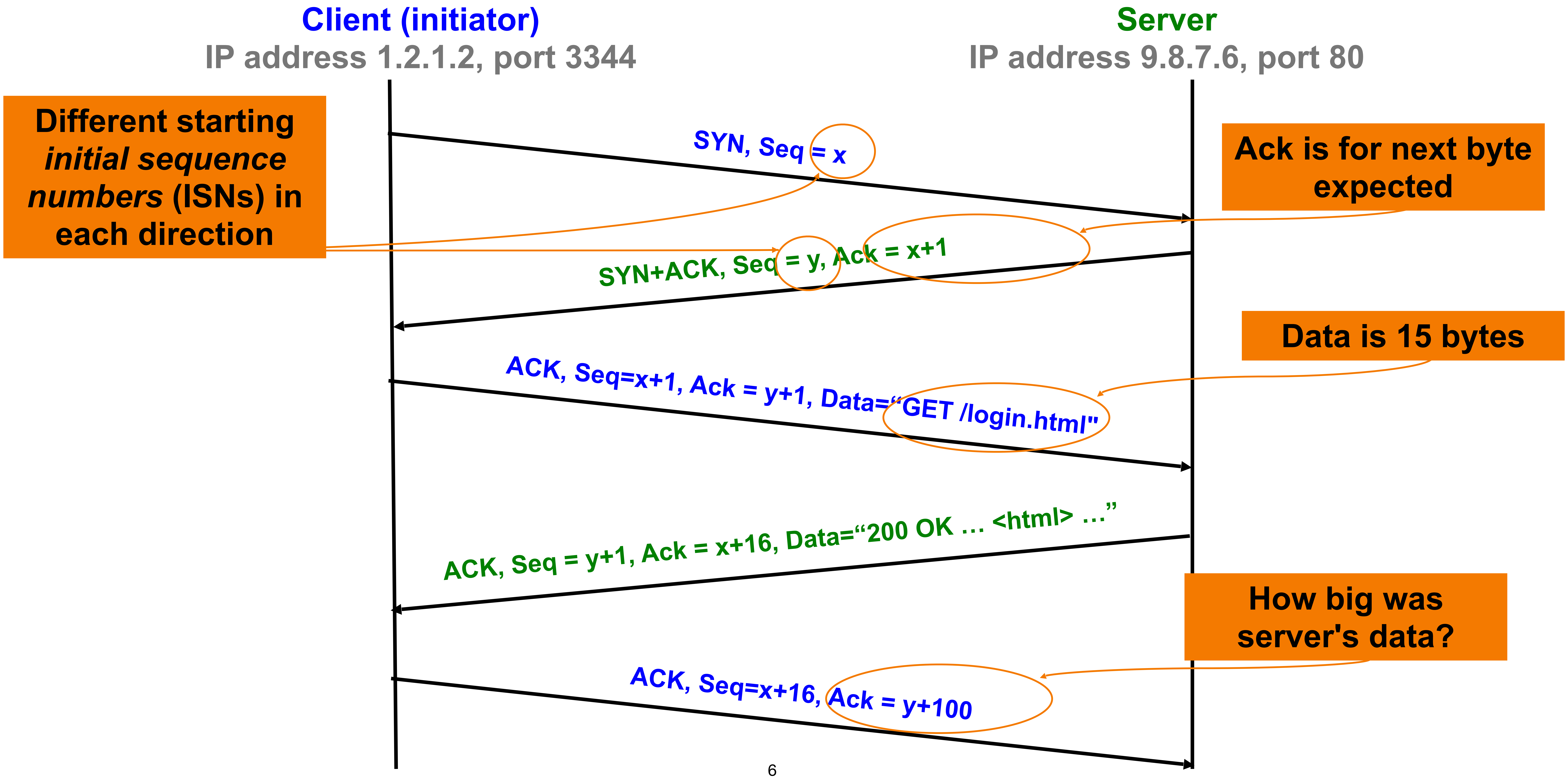
TCP Header


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

Sequence + Acknowledgement Numbers



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 acknowleding 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

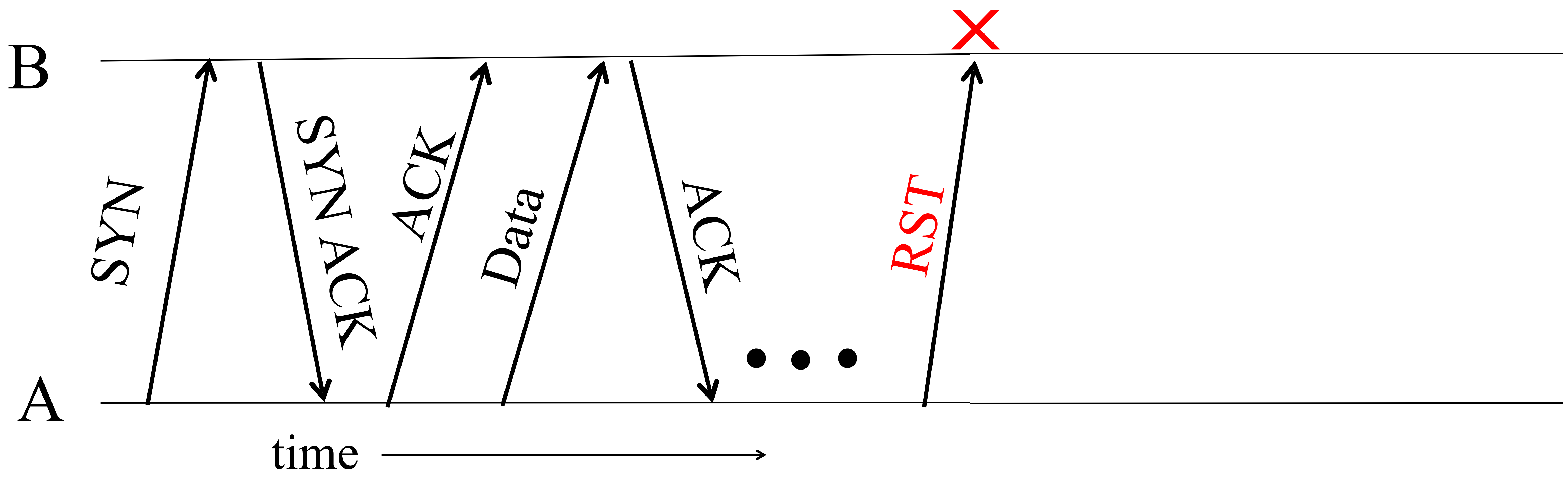
Normally, TCP finishes (“closes”) a connection by each side sending a FIN control message.

- 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.

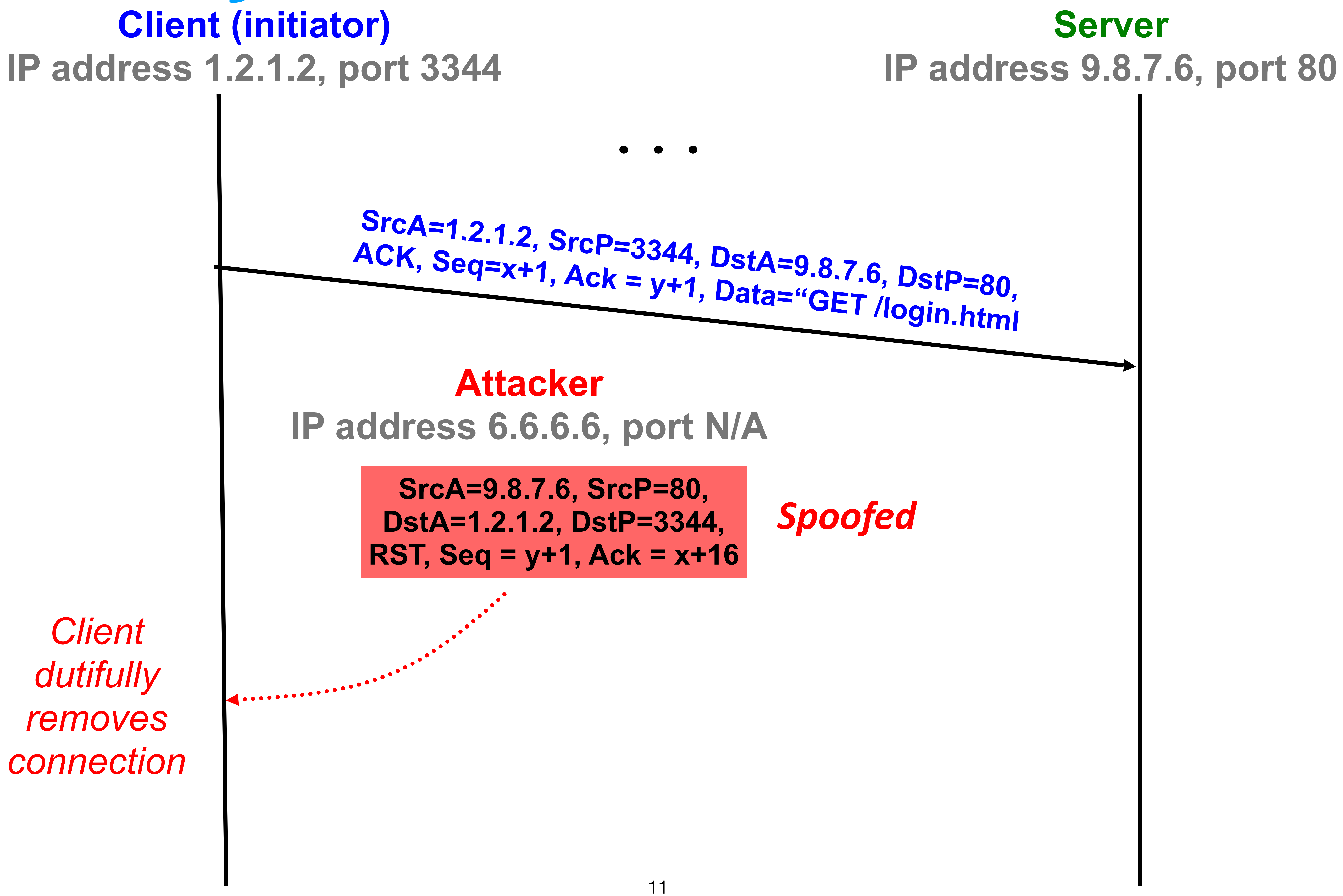
- Reliably delivered, since other side must ack

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

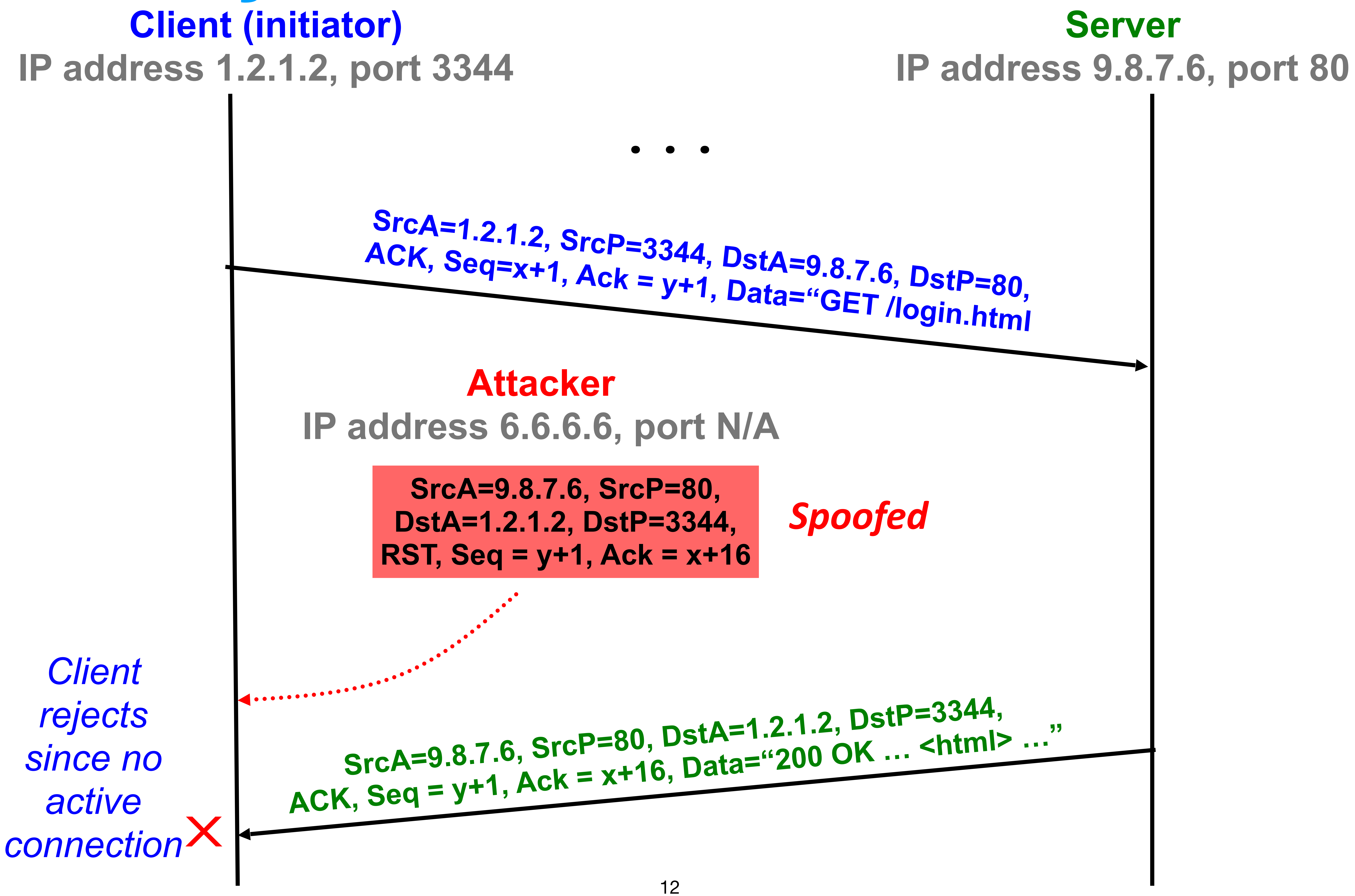
- Unilateral
- Takes effect immediately (no ack needed)
- Only accepted by peer if has correct* sequence number

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)

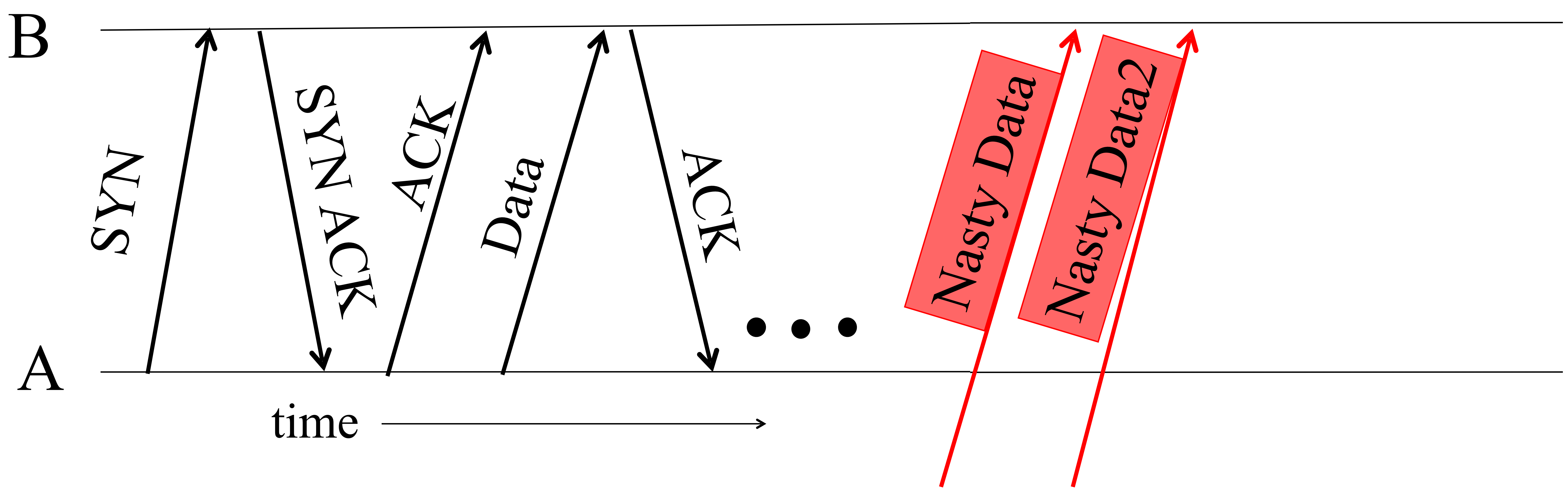
TCP RST Injection



TCP RST Injection

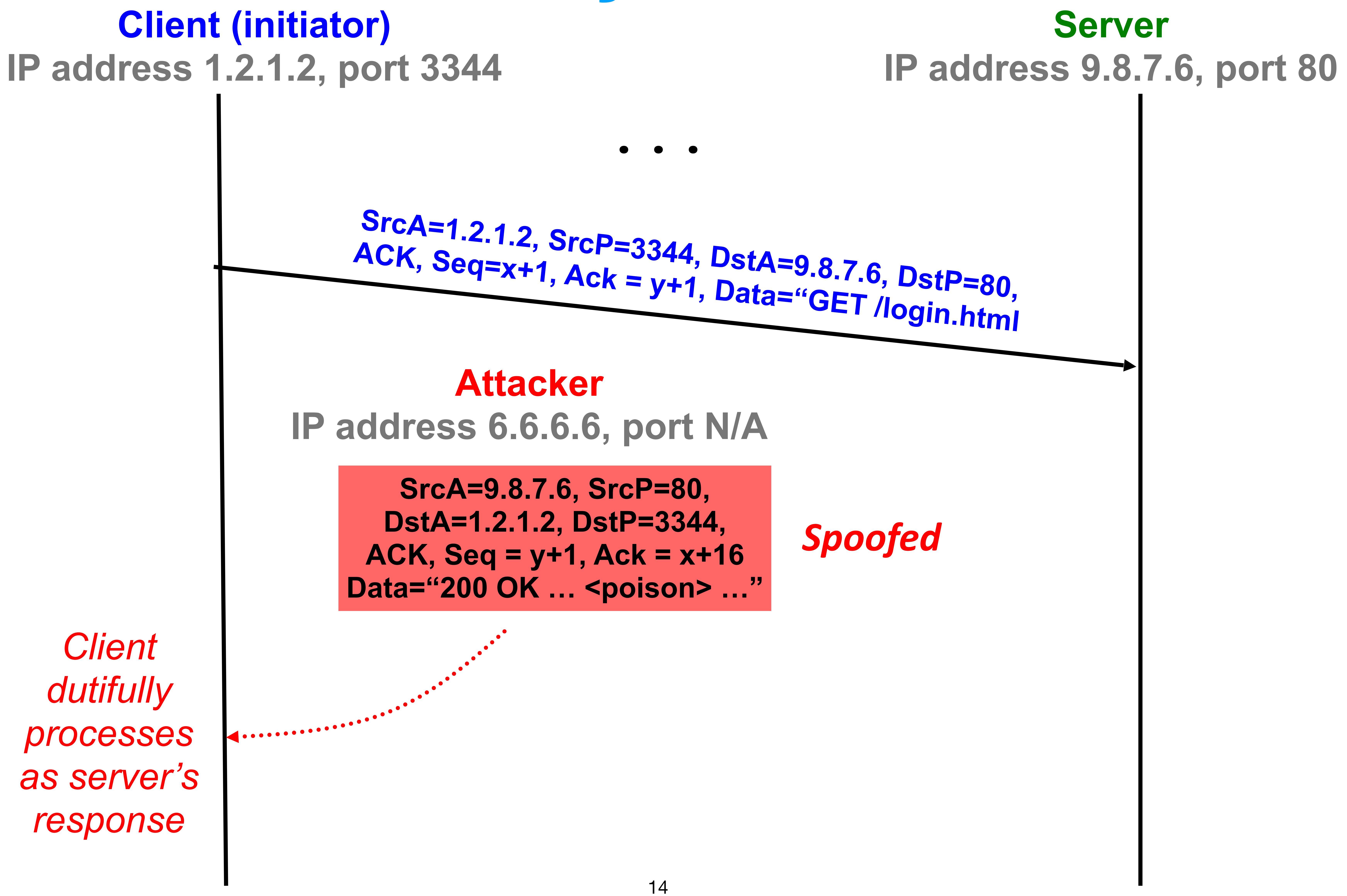


TCP Threats: Data Injection

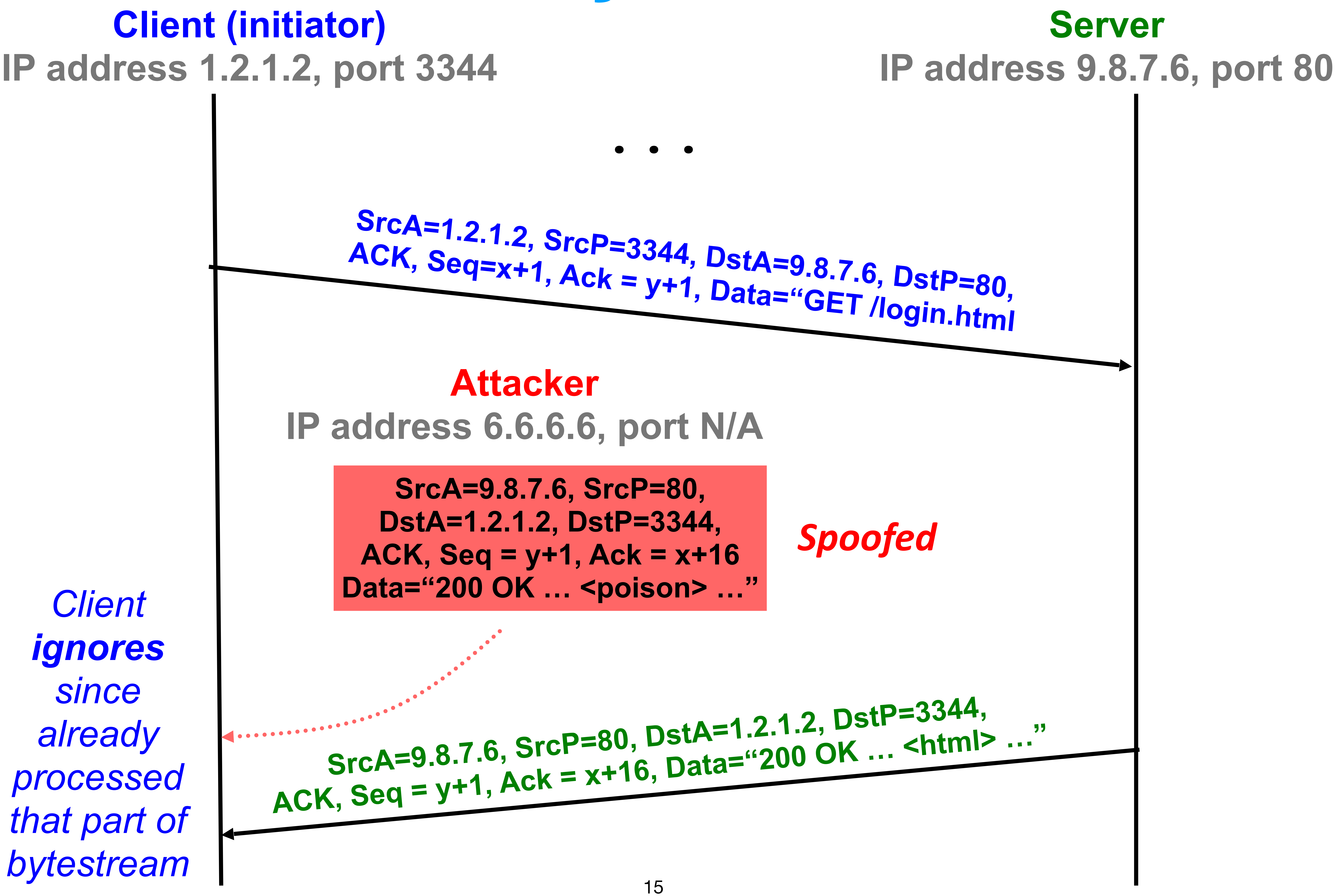


- What about inserting data rather than disrupting a connection?
  - Again, all that's required is attacker knows correct ports, seq. numbers
  - Receiver B is *none the wiser!*
- Termed TCP connection hijacking (or "*session hijacking*")
  - A general means to take over an already-established connection!
- **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 Threats: SYN Flood

Each time you send a SYN packet to a server, it needs to use reserve some computational resources to set up the connection (e.g., record the TCP 4-tuple and the connection seq numbers).

What happens if an attacker sends a ton of SYN packets to the server with spoofed IP sources? Cause the server to eventually stop accepting new connections, triggering a denial of service attack, without being able to block the IP source address.

TCP Threats: SYN Flood

Defense?

SYN Cookies: Server encodes the connection information in the SYN-ACK packet's sequence #, so you don't need to store any state for this connection *unless* you get back an ACK packet (protecting against the spoofing).

- Seq # = 5-bit timestamp T || 3-bit connection encoding || 24-bit hash of 4-tuple + T, so H(src IP, src port, dst IP, dst port, T)
- In the ACK packet (3rd packet of handshake), ack # - 1 is the initial seq #. Can check that the timestamp is recent enough and that the hash is valid. If so, complete TCP connection setup.
- Note: doesn't work if SYN flood targets network bandwidth rather than server resources.

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).

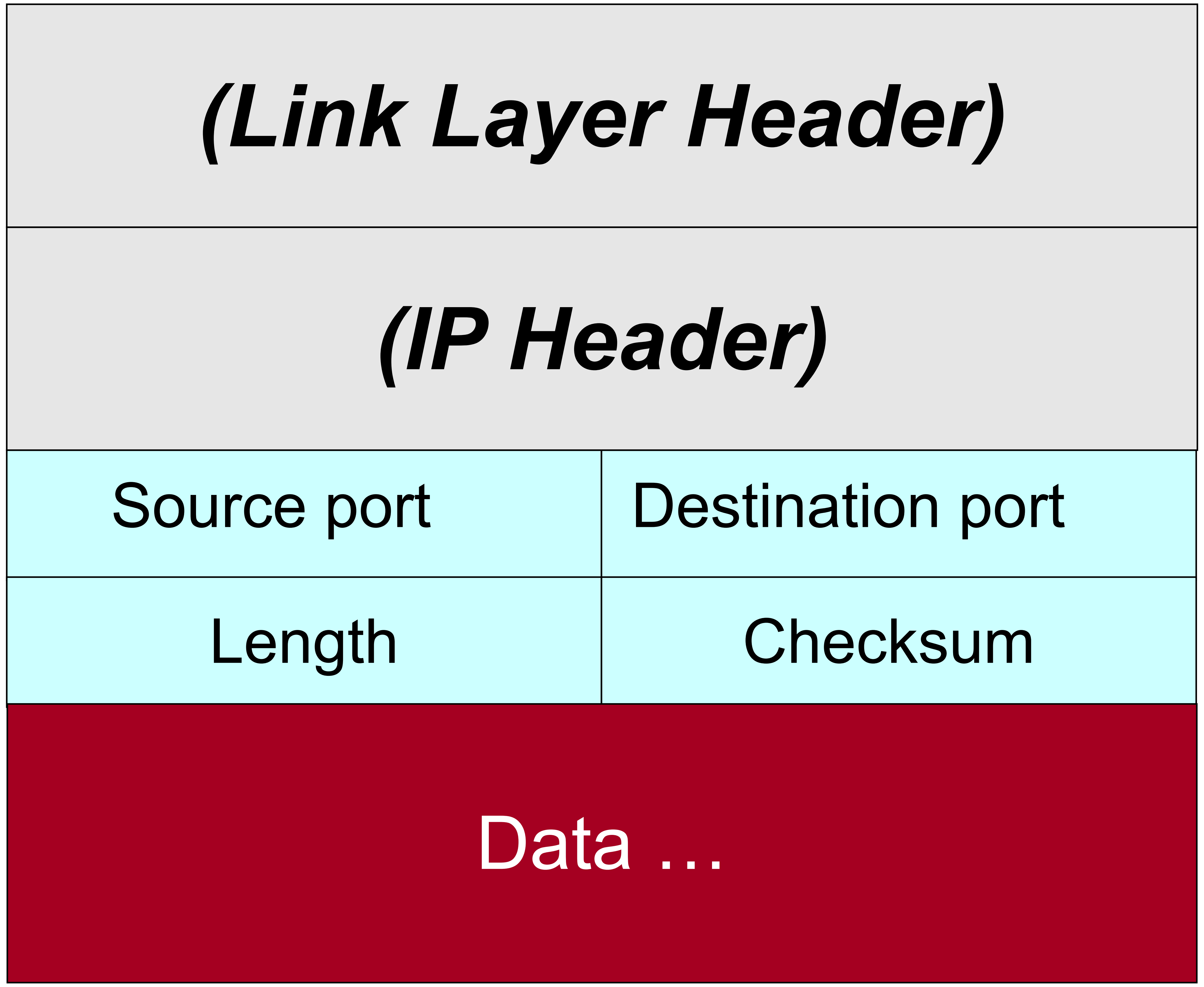
If an attacker can learn these values, they can spoof TCP packets that will be accepted, allowing for:

- 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

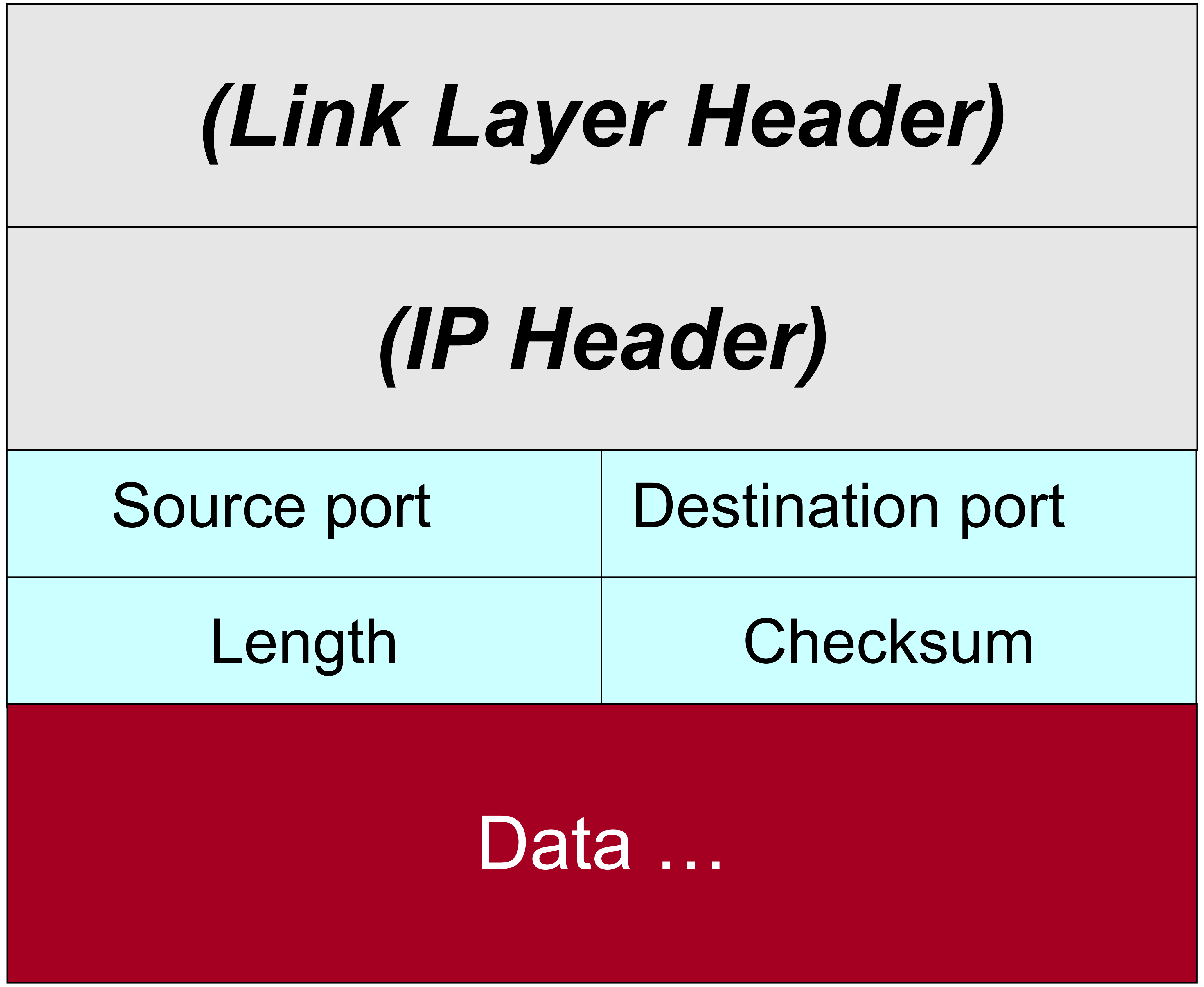

(Link Layer Header)
(IP Header)
Source port
Destination port
Length
Checksum
Data …

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 next)



DNS Security

Slides borrowed from Manos Antonakakis and Vern Paxson

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP addresses - used for addressing packets
- "name", e.g., www.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:

- distributed database implemented in hierarchy of many name servers
- application-layer protocol host, routers, name servers to communicate to resolve names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol

DNS

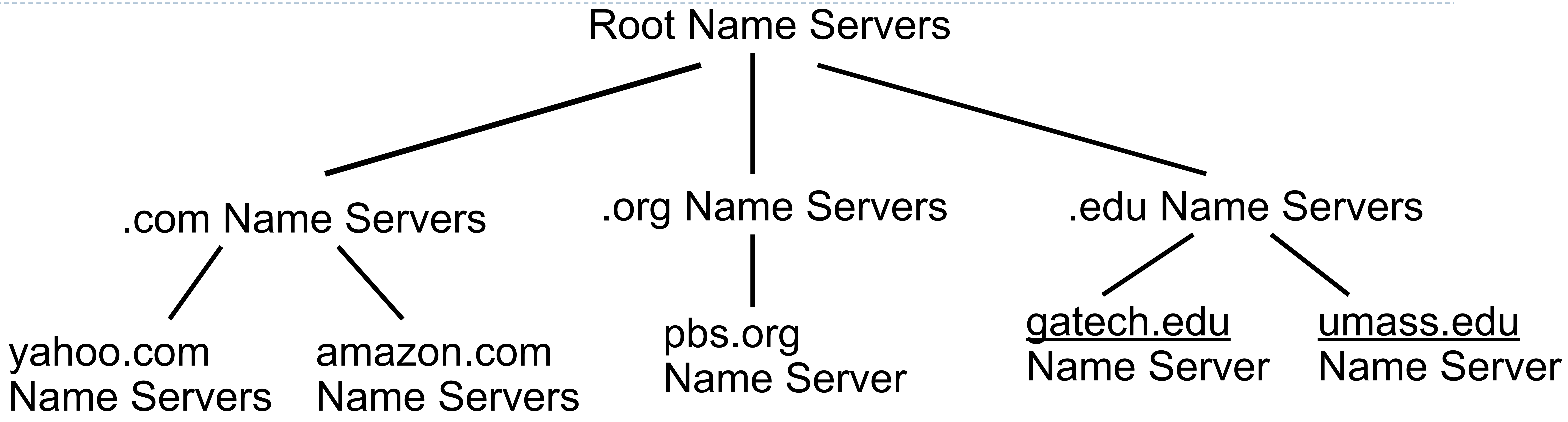
DNS services

- Hostname to IP address translation
- Host aliasing
  - Canonical and alias names
- Mail server aliasing
- Load distribution
  - Replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance
- centralization doesn't scale

Distributed, Hierarchical Database



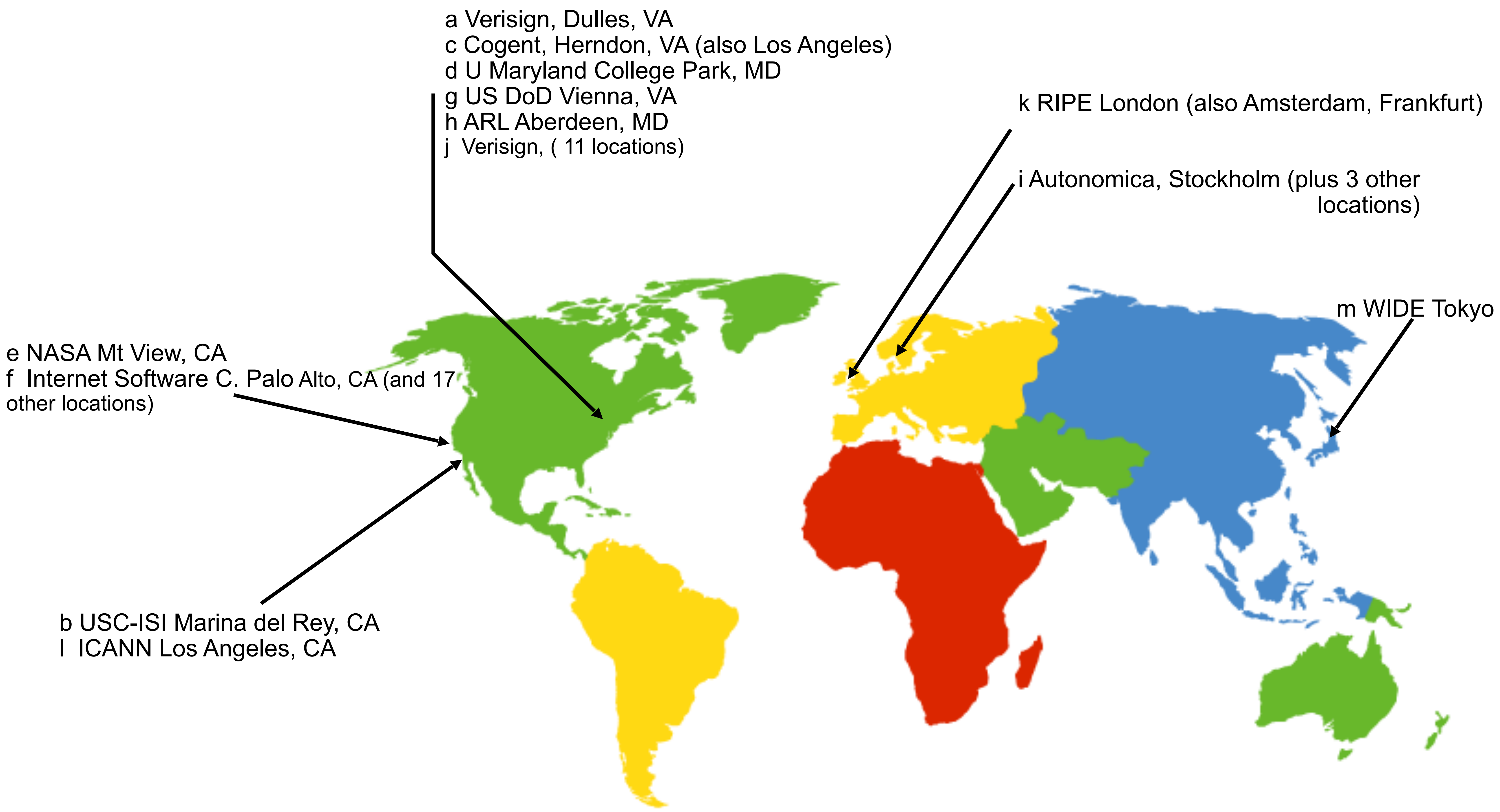
Client wants IP for www.amazon.com; 1st approx:

- Client queries a root name server to find "com" name server
- Client queries "com" name server to get amazon.com name server
- Client queries amazon.com name server to get IP address for www.amazon.com

Types of DNS Servers

- Root Name Servers
  - There are 13 DNS root name servers known to every resolver
  - More than only 13 machines in the root name server system
  - Multiple copies of these 13 name servers exist around the world
- Top-level domain (TLD) Name Servers
  - Responsible for ”top level” of domains (e.g., com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp)
- Authoritative DNS Name Server
  - DNS servers providing the final mapping for a record (e.g., A record mapping hostname to IP address)
- Recursive Resolvers
  - Acts as middleman between client and a DNS name server
  - Caches results from queries for performance

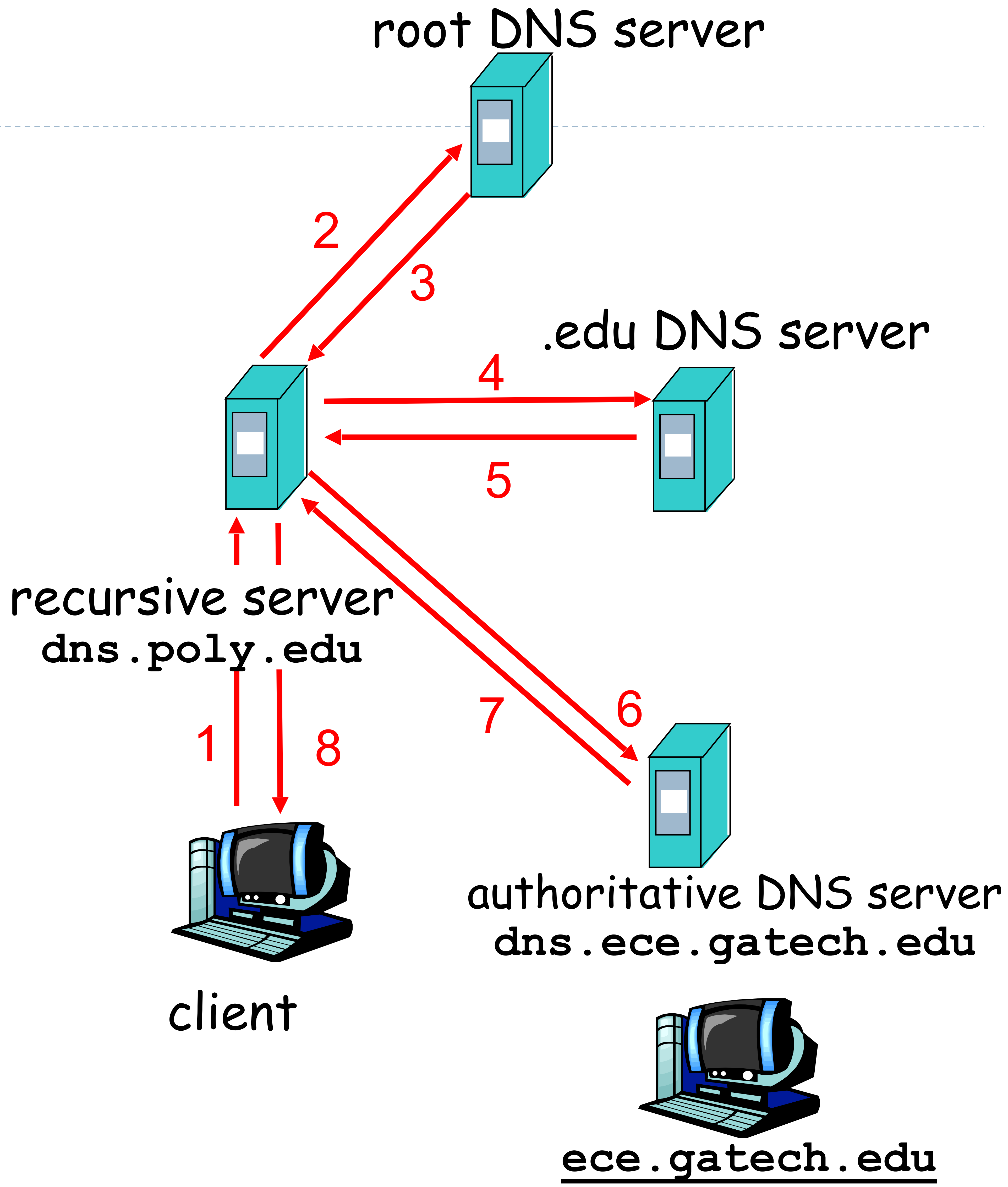
DNS: Root name servers



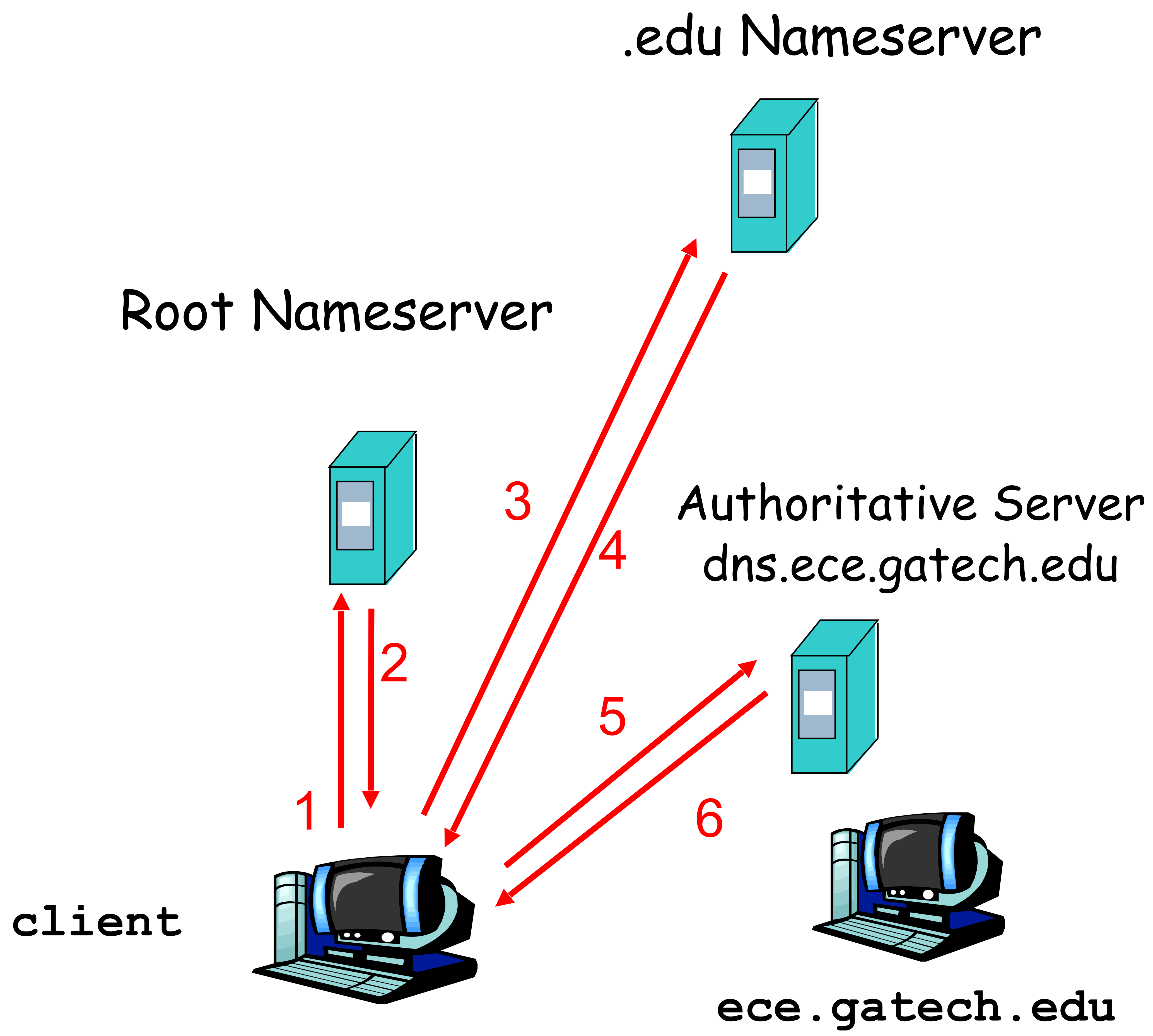
13 root name servers worldwide

Recursive Query

- Host at ex.poly.edu wants IP address for ece.gatech.edu
- Host sends a "**recursion-requested**" query request to dns.poly.edu.
- Local DNS server does a "**recursive**" search. This requires contacting several other DNS servers before the final answer is given to host.



Iterative Query



DNS records

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

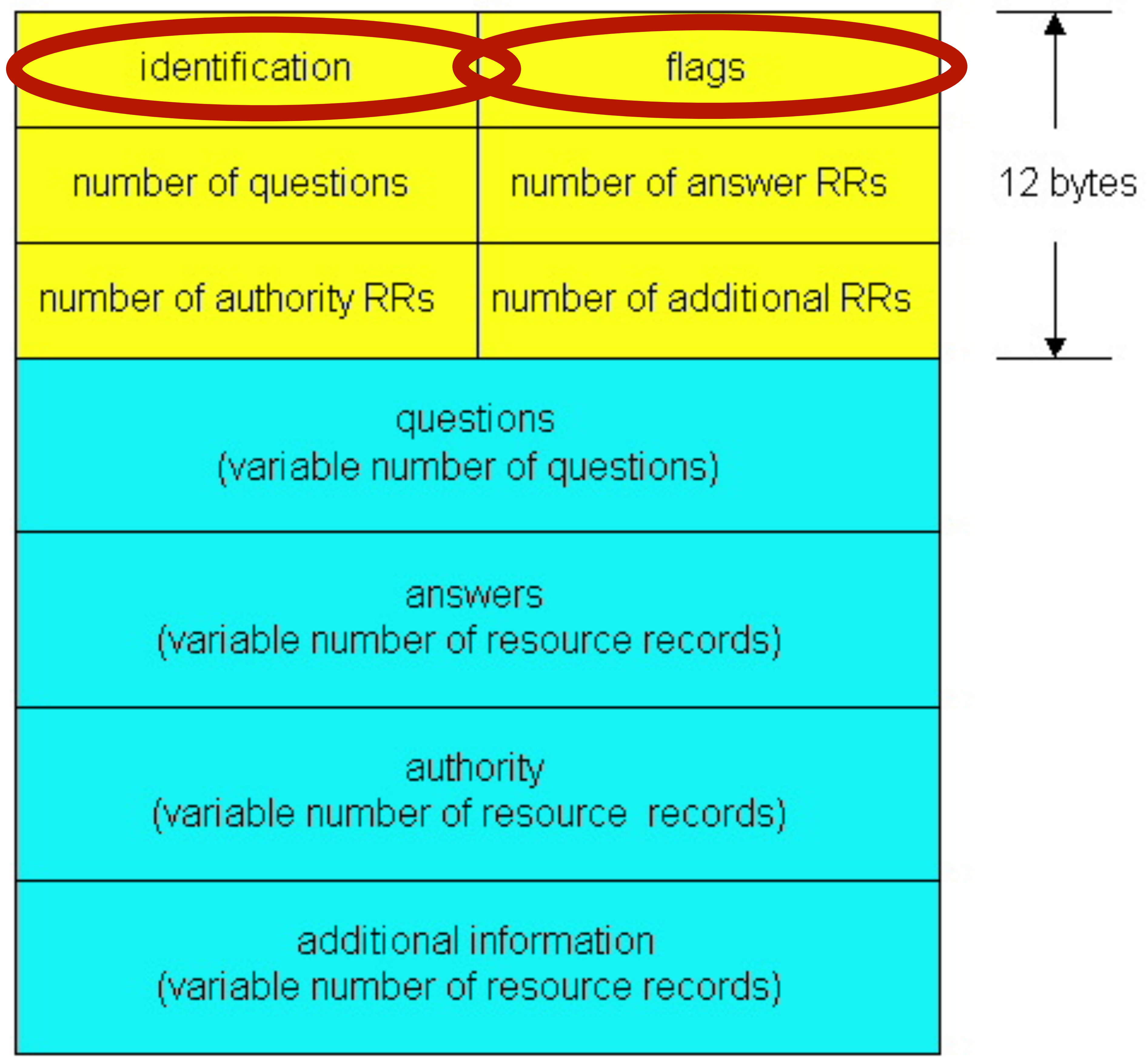
- Type=A (AAAA for IP6)
  - `name` is hostname
  - `value` is IP address
- Type=NS
  - `name` is domain (e.g. foo.com)
  - `value` is hostname of authoritative name server for this domain
- Type=CNAME
  - `name` is alias name for some "canonical" (the real) name
    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - `value` is canonical name
- Type=MX
  - `value` is name of mailserver associated with `name`

DNS protocol, messages

DNS protocol: UDP (port 53) query and reply messages, both with same message format

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



Inserting records into DNS

- Example: just created startup "Network Utopia"
- Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- At authoritative server, can configure Type A record for www.networkuptopia.com and Type MX record for networkutopia.com

DNS: caching and updating records

- once (any) name server learns mapping, it caches mapping
  - cache entries timeout (disappear) after some time. Sometimes based on ttl but *not always*
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited

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.
mit.edu.

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

In general, a single *Resource Record* (RR) like this includes, left-to-right, a DNS name, a *time-to-live*, a family (`IN` for our purposes - ignore), a type (`A` here, which stands for “Address”), and an associated value

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

;; 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
```

“**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

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APl
;; global options: +cm
;; Got answer:
;; ->>HEADER<<- opcode
;; flags: qr rd ra; QU                                                    3

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

;; 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
```

"**Authority**" tells us the *name servers* responsible for the answer. Each RR gives the *hostname* of a different name server ("`NS`") for names in `mit.edu.` We should cache each record for 11,088 seconds.

If the "**Answer**" had been empty, then the resolver's next step would be to send the original query to one of these name servers.

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.

;; AUTHORITY SECT
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
```

“**Additional**” provides extra information to save us from making separate lookups for it, or helps with bootstrapping.

Here, it tells us the IP addresses for the hostnames of the name servers. We add these to our cache.

DNS Security Threats

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.

;; ANSWER SECTION:
eecs.mit.edu.            21

;; 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
```

What if the `mit.edu` name server is untrustworthy? Could its operator steal, say, all of our web surfing to Facebook?

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.

;; ANSWER SECTION:
eecs.mit.edu.           21                                  .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
```

Let's look at a flaw in the original DNS design (since fixed)

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.

;; 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.                 30      IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com         30      IN      A       18.6.6.6
BITSY.mit.edu.           166408  IN      A       18.72.0.3
W20NS.mit.edu.           126738  IN      A       18.70.0.160
```

What could happen if the `mit.edu` name server returns the following to us instead?

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.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com        30      IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

We'd dutifully store in our cache a mapping of `www.facebook.com` to an IP address under MIT's control. (It could have been any IP address they wanted, not just one of theirs.)

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.

;; AUTHORITY SECTION:
mit.edu.              11088       IN      NS      BITSY.mit.edu.
mit.edu.              11088       IN      NS      W20NS.mit.edu.
mit.edu.              30          IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com      30          IN      A       18.6.6.6
BITSY.mit.edu.        166408      IN      A       18.72.0.3
W20NS.mit.edu.        126738      IN      A       18.70.0.160
```

In this case they chose to make the mapping *disappear* after 30 seconds. They could have made it persist for weeks, or disappear even quicker.

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.

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                                                    edu.
mit.edu.             11088    IN      NS      W20NS.mit.edu.
mit.edu.             30       IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com     30       IN      A       18.6.6.6
BITSY.mit.edu.       166408   IN      A       18.72.0.3
W20NS.mit.edu.       126738   IN      A       18.70.0.160
```

Next time one of our clients starts to connect to `www.facebook.com`, it will ask our resolver for the corresponding IP address. The resolver will find the answer in its cache and return **18.6.6.6** 😟

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.

;; AUTHORITY SECTION:
mit.edu.                 11088  IN      NS      BITSY.mit.edu.
mit.edu.                 11088  IN      NS      W20NS.mit.edu.
mit.edu.                 30     IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com         30     IN      A       18.6.6.6
BITSY.mit.edu.           166408 IN      A       18.72.0.3
W20NS.mit.edu.           126738 IN      A       18.70.0.160
```

How do we fix such ***cache poisoning***?

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-AP
;; global options: +c
;; Got answer:
;; ->>HEADER<<- opcod
;; flags: qr rd ra; Q

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

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                 11088   IN      NS      BITSY.mit.edu.
mit.edu.                 11088   IN      NS      W20NS.mit.edu.
mit.edu.                 30      IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com         30      IN      A       18.6.6.6
BITSY.mit.edu.           166408  IN      A       18.72.0.3
W20NS.mit.edu.           126738  IN      A       18.70.0.160
```

Don't accept **Additional** records unless they're for the domain of the name server we queried

E.g., contacting a name server for `mit.edu` ⇒ only accept additional records from `*.mit.edu`

No extra risk in accepting these since server could return them to us directly in an **Answer** anyway.

dig eecs.mit.edu A

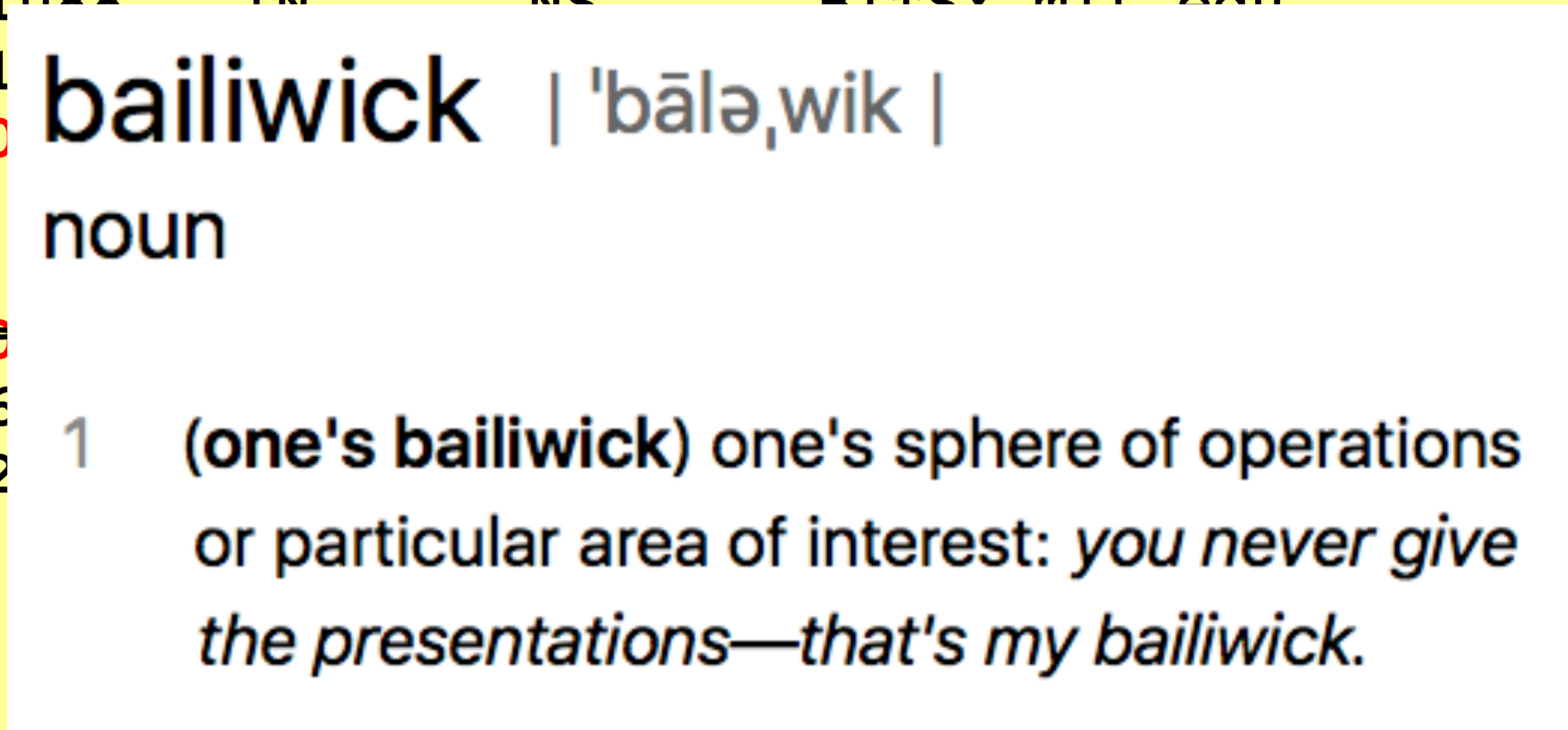
```
; ; <<>> DiG 9.6.0-AP
;; global options: +c
;; Got answer:
;; ->>HEADER<<- opcod
;; flags: qr rd ra; Q

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

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                 11088     IN        NS        BITSY.mit.edu.
mit.edu.                 11
mit.edu.                 30

;; ADDITIONAL SECTION:
www.facebook.com         30
BITSY.mit.edu.           16
W20NS.mit.edu.           12
```

Don't accept **Additional** records unless they're for the domain of the name server we queried

E.g., contacting a name server for `mit.edu` ⇒ only accept additional records from `*.mit.edu`

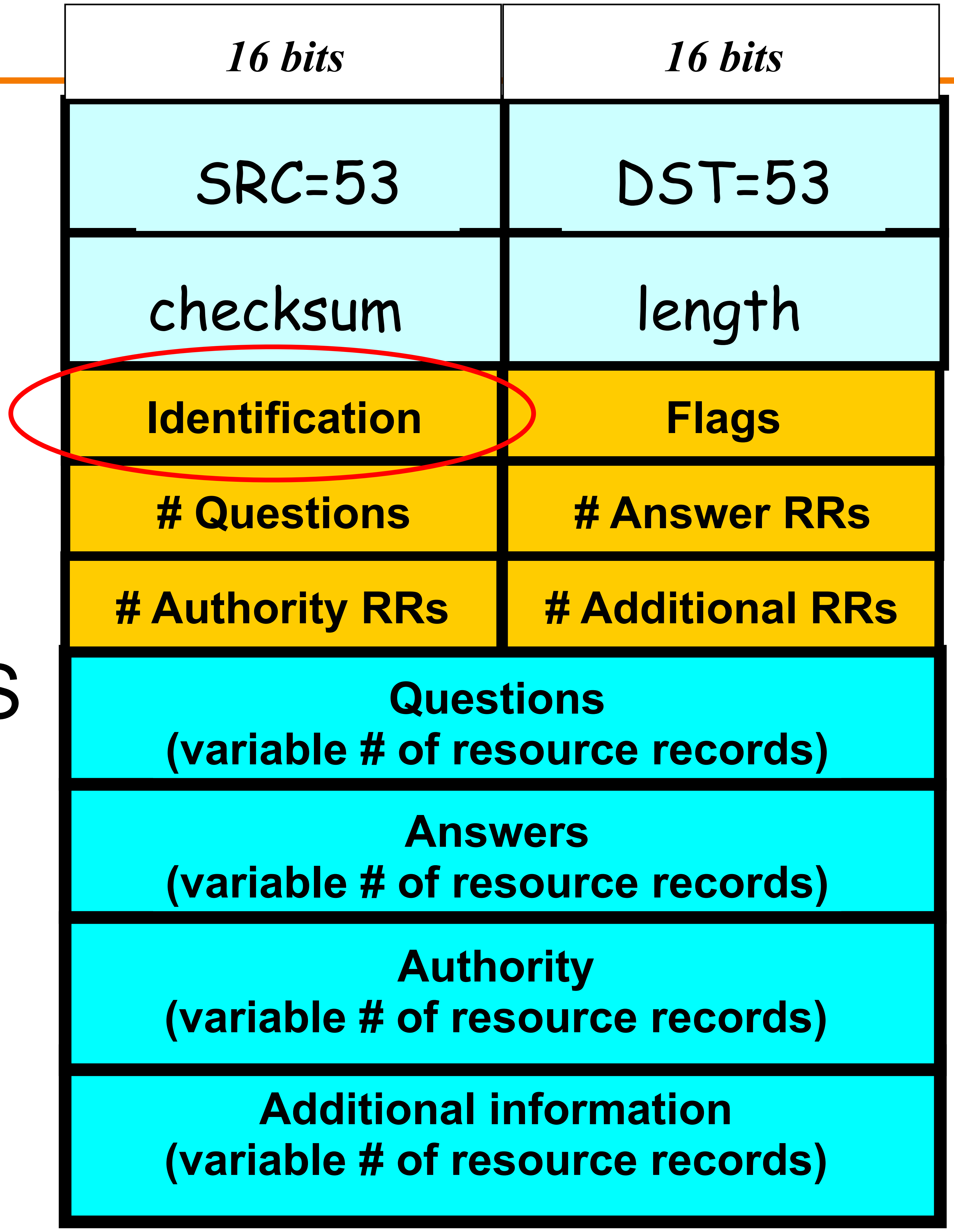
No extra risk in accepting these since server could return them to us directly in an **Answer** anyway.

This is called "bailiwick checking".



DNS Threats: Spoofing

- If an attacker observes the DNS identification number, they can spoof a DNS response to a victim, giving them false DNS answers (e.g., respond to a DNS query for `mail.google.com` with an A record pointing to the attacker server)
- Has been used in real attacks, including censorship.



DNS Threats: Spoofing

What about *blind spoofing*?

- Say we look up `mail.google.com`; how can an **off-path** attacker feed us a bogus `A` answer before the legitimate server replies?

- How can such a **remote** attacker even know we are looking up `mail.google.com`?

Suppose, e.g., we visit a web page under their control:

...<img src="http://mail.google.com" …> ...

16 bits	*16 bits*
SRC=53	DST=53
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

DNS Threats: Spoofing

What about *blind spoofing*?

- Say we look up `mail.google.com`; how can an **off-path** attacker feed us a bogus `A` answer before the legitima
- How ca even k `mail.`

This HTML snippet causes our browser to try to fetch an image from `mail.google.com`. To do that, our browser first has to look up the IP address associated with that name.

Suppose, e.g., we visit a web page under their control:

`...<img src="http://mail.google.com" …> ...`

16 bits	16 bits
SRC=53	DST=53
checksum	length
Identification	Flags
# Questions	# Answer RRs
# Authority RRs	# Additional RRs
estions resource records)	
nswers resource records)	
thority resource records)	
al information (variable # of resource records)	

DNS Blind Spoofing, con't

Fix?

Once they know we're looking it up, they just have to guess the Identification field, and reply before legit server.

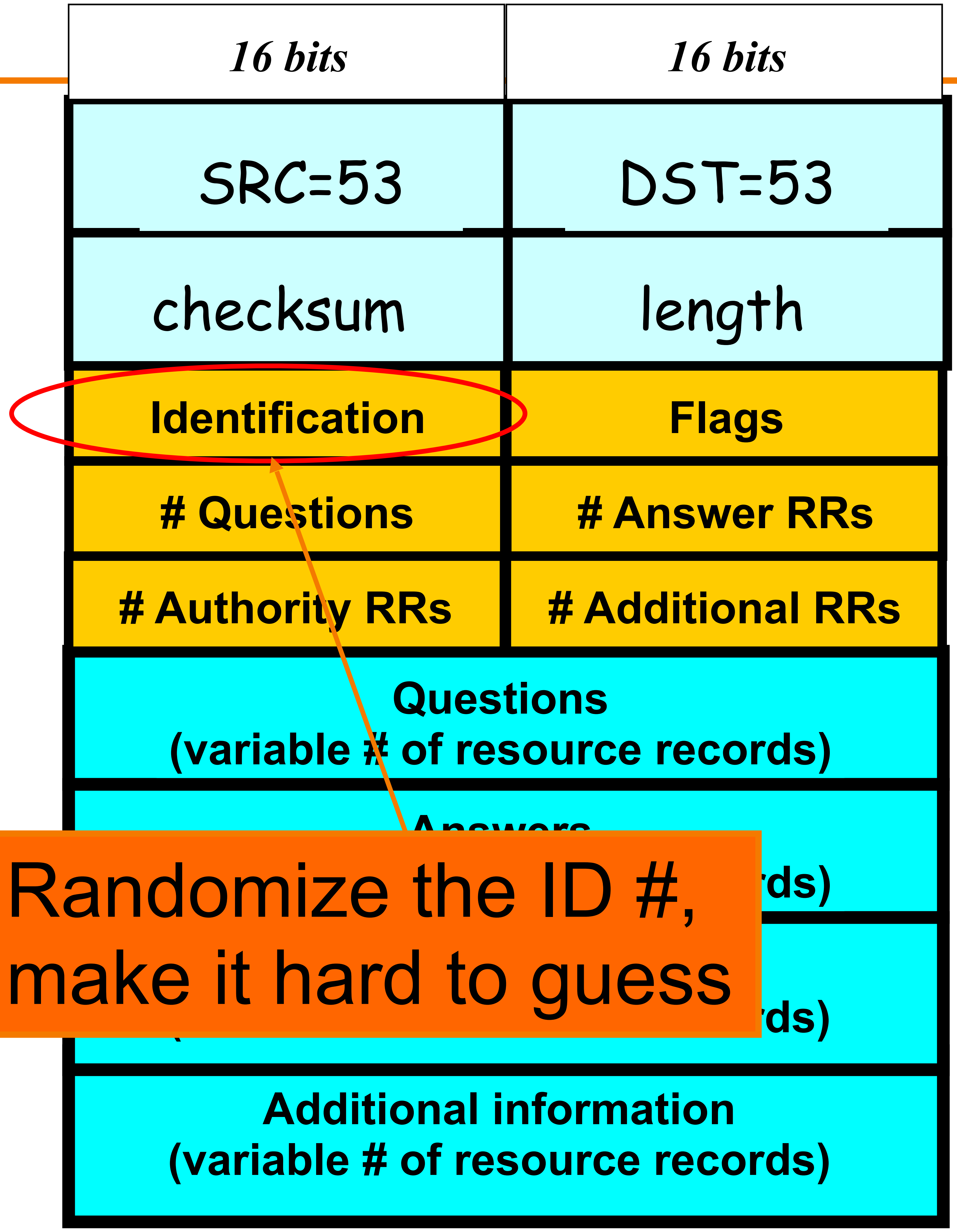
How hard is that?

Originally, identification field incremented by 1 for each request. How does attacker guess it?

(Assuming attacker controls their domain's name server)

<img src="http://attacker.com" ...>They observe ID k here

<img src="http://mail.google.com" ...>So this will be k+1



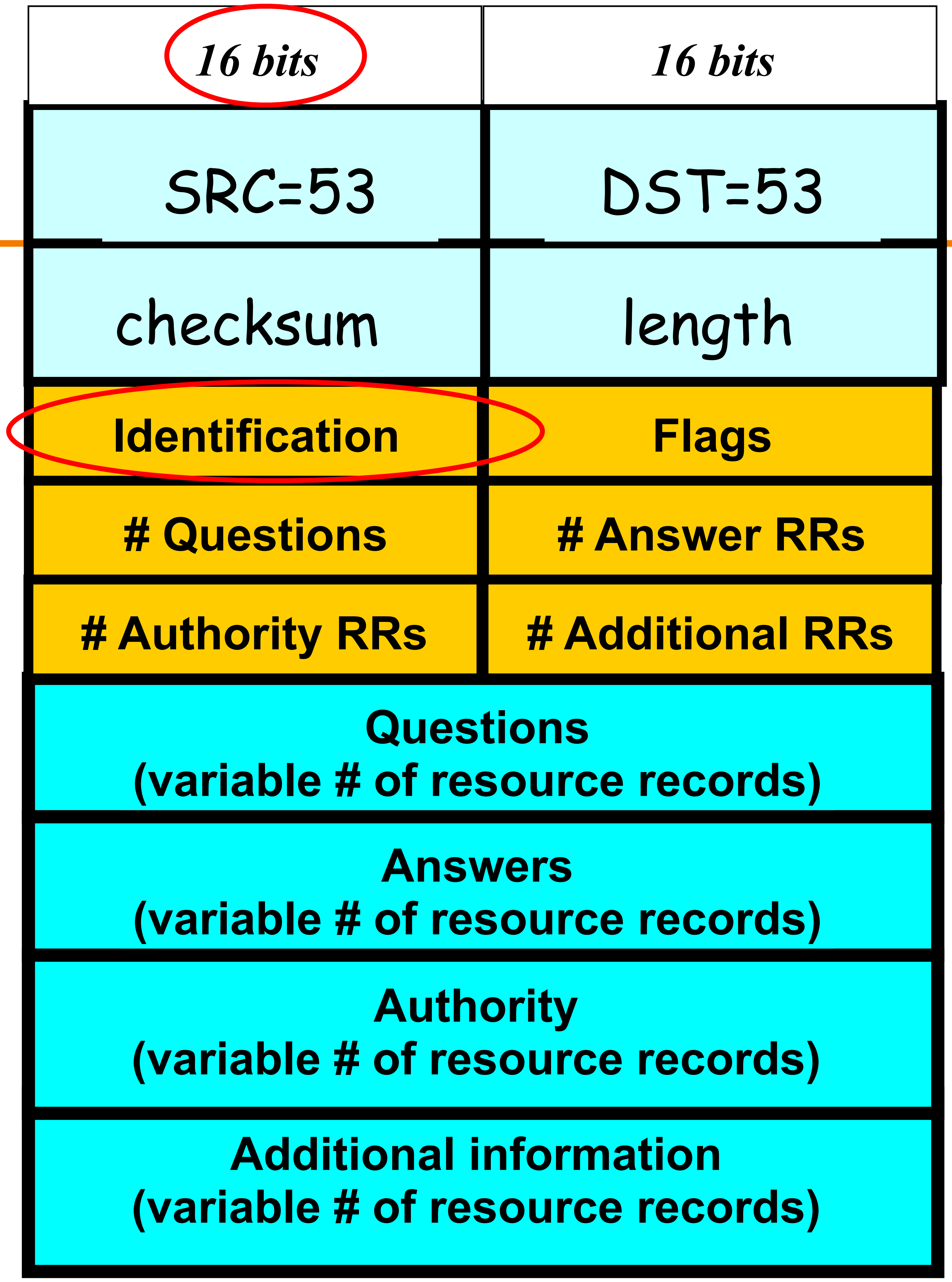
Randomize the ID #, make it hard to guess

DNS Blind Spoofing, con't

Once we randomize the Identification, attacker has a 1/65536 chance of guessing it correctly.
Are we pretty much safe?

Attacker can send *lots* of replies, not just one …

However: once a reply from legit server arrives (with correct Identification), it's **cached** and no more opportunity to poison it. Victim is inoculated!



Unless attacker can send 1000s of replies before legit arrives, we're likely safe - phew! ?

DNS Blind Spoofing (Kaminsky 2008)

- Two key ideas:
  - Spoof uses **Additional** field (rather than **Answer**)
  - Attacker can get around caching of legit replies by generating a series of *different* name lookups:

```
<img src="http://random1.google.com" …>
<img src="http://random2.google.com" …>
<img src="http://random3.google.com" …>
                   . . .
<img src="http://randomN.google.com" …>
```

Kaminsky Blind Spoofing, con't

For each lookup of *randomk*.google.com, attacker **spoofs** a bunch of records like this, each with a different Identifier

```
;; QUESTION SECTION:
;randomk.google.com.            IN      A

;; ANSWER SECTION:
randomk.google.com      21600   IN      A       doesn't matter

;; AUTHORITY SECTION:
google.com.             11088   IN      NS      mail.google.com

;; ADDITIONAL SECTION:
mail.google.com         126738  IN      A       6.6.6.6
```

Once they win the race, not only have they poisoned mail.google.com …

Kaminsky Blind Spoofing, con't

For each lookup of *randomk*.google.com, attacker **spoofs** a bunch of records like this, each with a different Identifier

```
;; QUESTION SECTION:
;randomk.google.com.            IN      A

;; ANSWER SECTION:
randomk.google.com      21600   IN      A       doesn't matter

;; AUTHORITY SECTION:
google.com.             11088   IN      NS      mail.google.com

;; ADDITIONAL SECTION:
mail.google.com         126738  IN      A       6.6.6.6
```

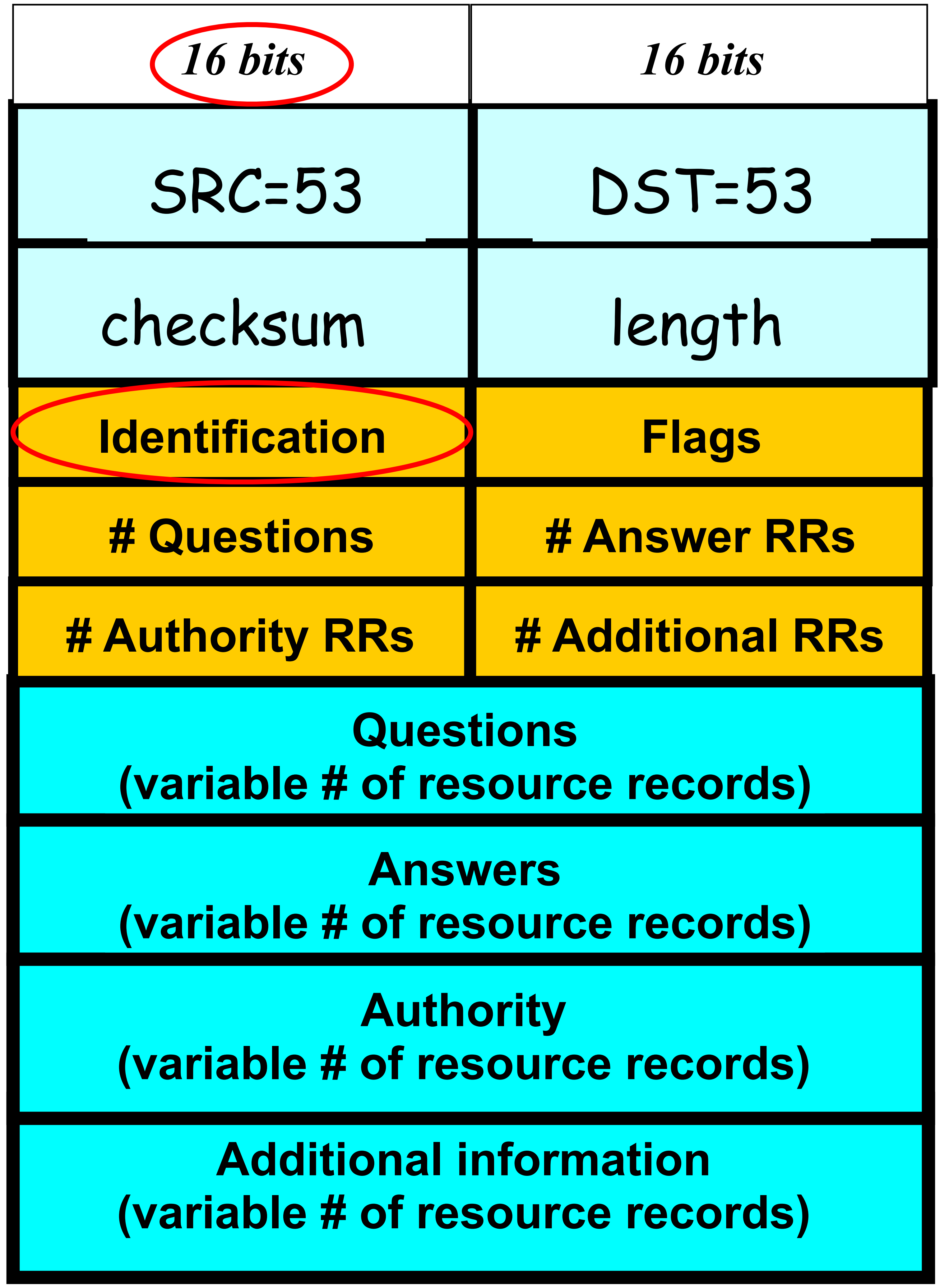
Once they win the race, not only have they poisoned mail.google.com … but also the cached NS record for google.com's name server - so any **future** *X*.google.com lookups *go through the attacker's machine*

Defending Against Blind Spoofing

Central problem: all that tells a client they should accept a response is that it matches the Identification field.

With only 16 bits, it lacks sufficient entropy: even if truly random, the *search space* an attacker must *brute force* is too small.

Where can we get more entropy?



Defending Against Blind Spoofing

Central problem: all that tells a client they should accept a response is that it matches the Identification field.

With only 16 bits, it lacks sufficient entropy: even if truly random, the *search space* an attacker must *brute force* is too small.

Where can we get more entropy? (*Without* requiring a protocol change.)

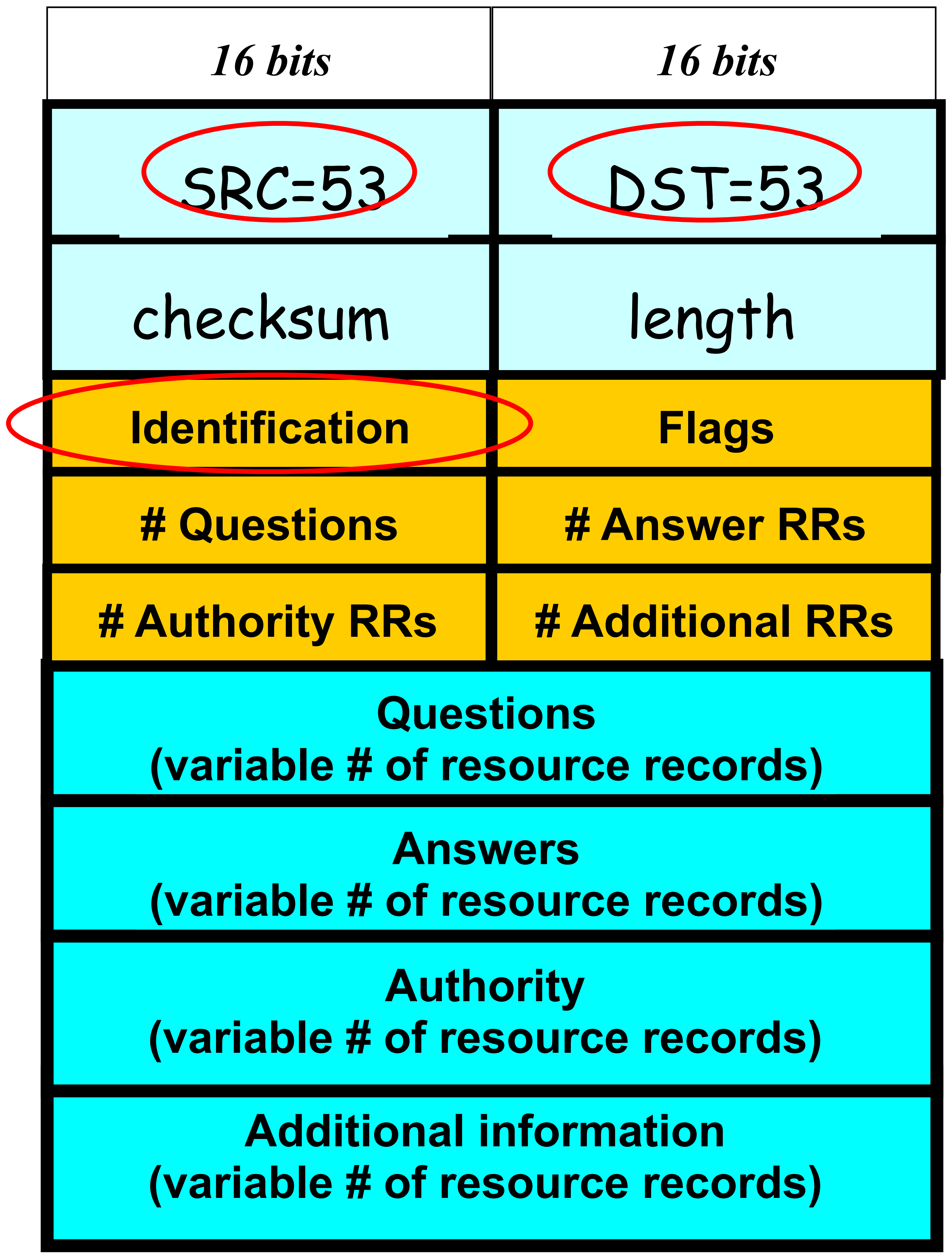
16 bits	16 bits
SRC=53	DST=53
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Defending Against Blind Spoofing

Total *entropy*: 16 bits

For requestor to receive DNS reply, needs both correct Identification and correct ports.

On a request, DST port = 53. SRC port usually also 53 - but not fundamental, just convenient.



Defending Against Blind Spoofing

Total *entropy*: ? bits

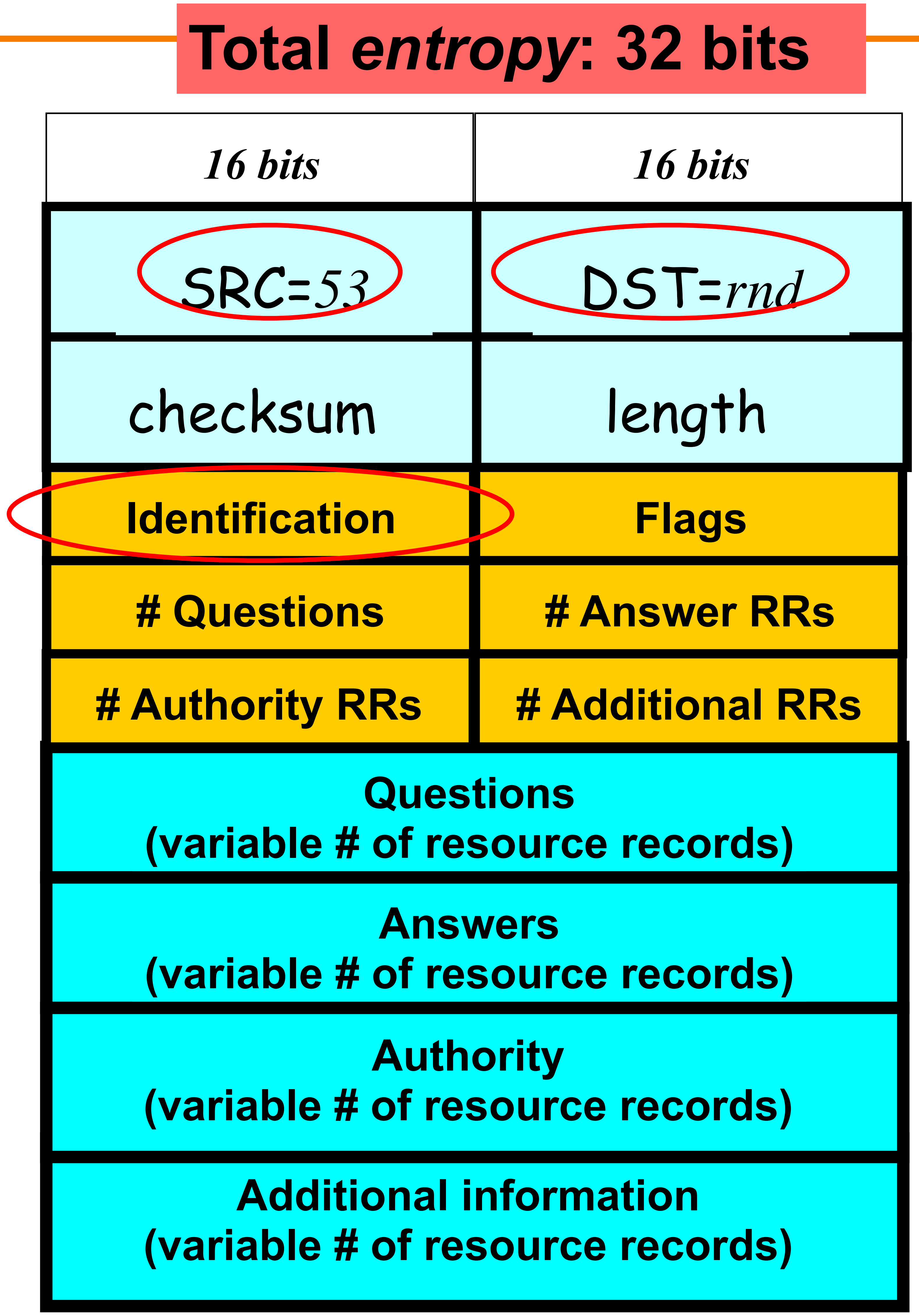
“Fix”: client uses random source port ⇒ attacker doesn’t know correct dest. port to use in reply

16 bits	16 bits
SRC=53	DST=rnd
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Defending Against Blind Spoofing

“Fix”: client uses random source port ⇒ attacker doesn’t know correct dest. port to use in reply

32 bits of entropy makes it orders of magnitude harder for attacker to guess all the necessary fields and dupe victim into accepting spoof response.



Defending Against Blind Spoofing

"Fix": client uses random source port ⇒ attacker doesn't know correct dest. port to use in reply

32 bits of entropy makes it orders of magnitude harder for attacker to guess all the necessary fields and dupe victim into accepting spoof response.

This is what primarily "secures" DNS against blind spoofing today. (Note: not all resolvers have implemented random source ports!)

Total *entropy*: 32 bits

16 bits	*16 bits*
SRC=53	DST=*rnd*
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Summary of DNS Poisoning Issues

- DNS threats highlight:
  - Attackers can attack opportunistically rather than eavesdropping
    - Cache poisoning only required victim to look up some name under attacker’s control (*has been* ***fixed***)
  - Attackers can often manipulate victims into vulnerable activity
    - E.g., `IMG SRC` in web page to force DNS lookups
  - Crucial for identifiers associated with communication to have sufficient entropy (= a lot of bits of unpredictability)
  - “**Attacks only get better**”: threats that appears technically remote can become practical due to unforeseen cleverness