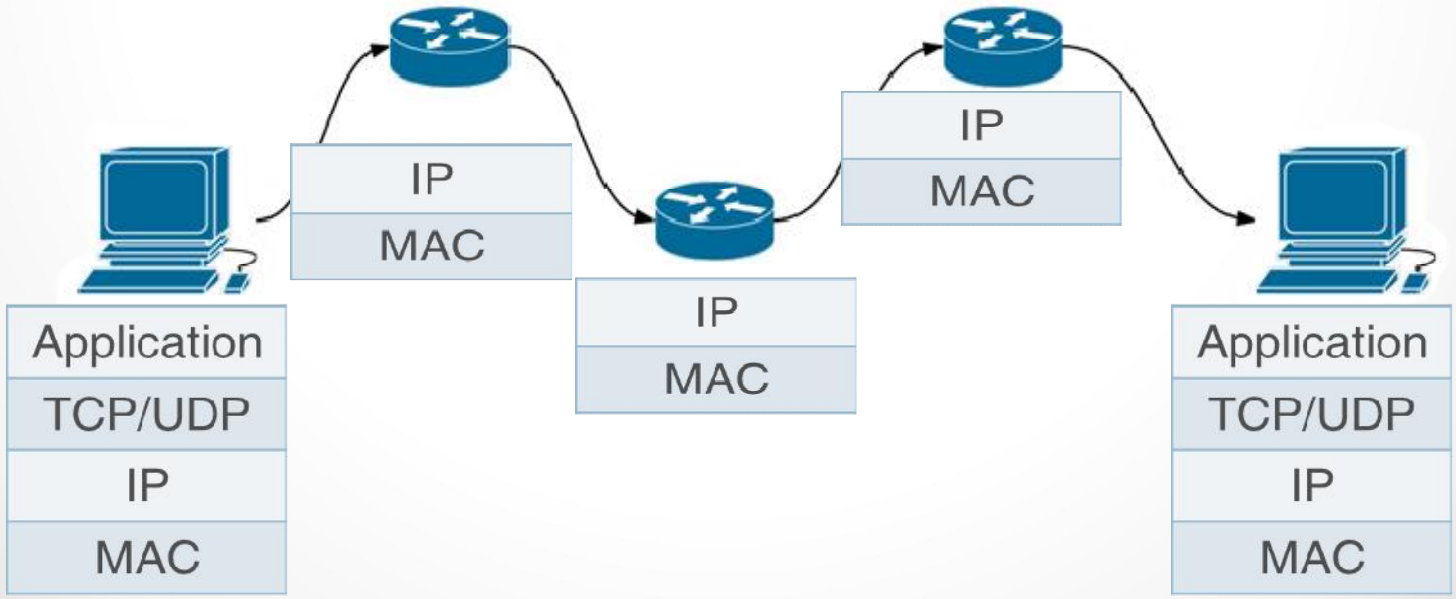
Packet Filtering & Firewalls

Stateless Packet Filtering

- Assume We can classify a "good" packet and/or a "bad packet"
- Each rule can examine that single packet against a good or bad signature/pattern, and accordingly – the packet either passes on or is dropped silently / dropped "loudly" / logged
- We run our set of black-list (deny) rules first, followed by our white-list (allow) rules. The first rule that matches takes precedence (no need to examine the rules)



Stateless rules

- Source & Destination IP addresses
- Transport protocol – TCP / UDP
- Source & Destination Ports
- TCP & UDP fields (i.e.: TCP Flags)
- Packet direction (from X interface to Y interface, internal network->world vs. world->internal network)
- Usually not used but possible –
  - Inspecting deeper layers
  - Pattern matching on packet data (e.g.: shellcode signature)

Stateful Packet Filtering

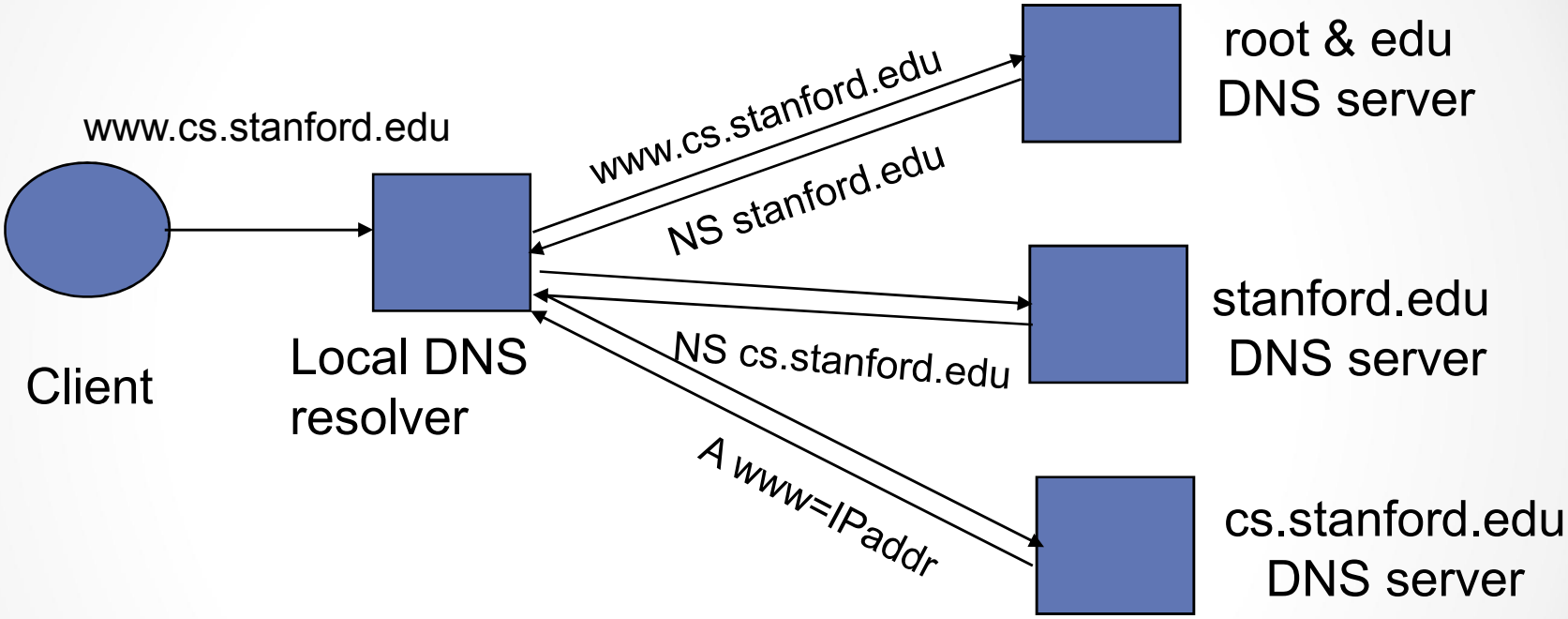
- To enable faster and more advanced filtering, the FW stores a list of active (and allowed) connections, and associated metadata
- If a packet matches a known connection – accept it
- If it doesn't, we'll run through our list of rules to see if the connection is valid

Complex Protocols

- Some protocols are complicated –
  - VoIP & P2P protocols may need to accept incoming connections
  - FTP uses one port for control and another (incoming) port for bulk data
  - TFTP uses random ports for both endpoints
  - Etc.
- We need to inspect (and modify) the application layer to perform proper packet filtering (i.e.: allow valid connections to enable these protocols)

DNS & DNS Attacks

DNS Lookup Example



DNS record types (partial list):

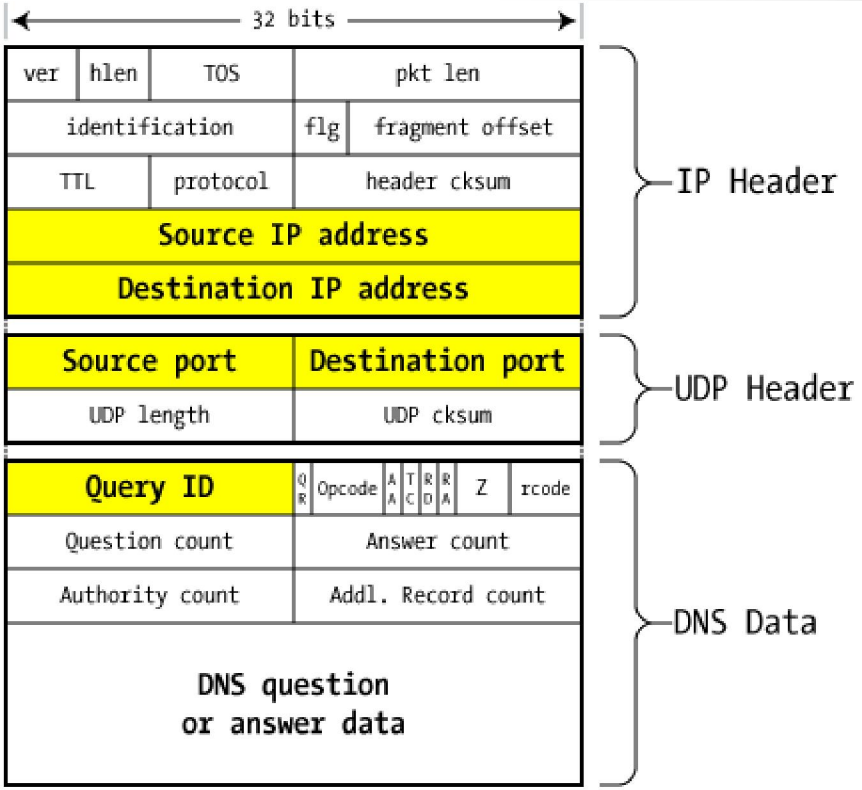
- NS: name server (points to other server)
- A: address record (contains IP address)
- MX: address in charge of handling email
- TXT: generic text (e.g. used to distribute site public keys (DKIM))

Caching

- DNS responses are cached at each level
  - Quick response for repeated translations
  - Useful for finding servers as well as addresses
    - NS records for domains
- DNS negative queries are cached
  - Save time for nonexistent sites, e.g. misspelling
- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record

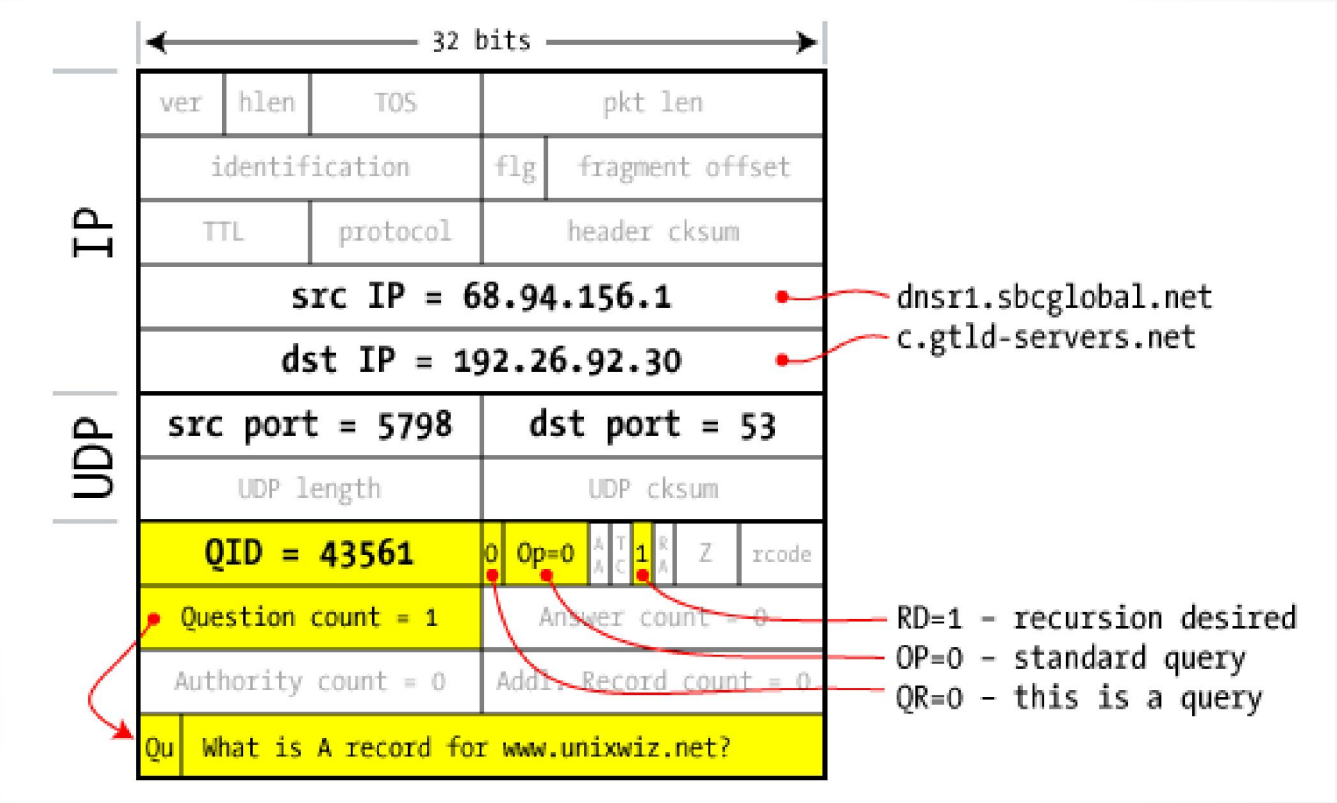
DNS Packet

- Query ID:
  - 16 bit random value
  - Links response to query



(from Steve Friedl)

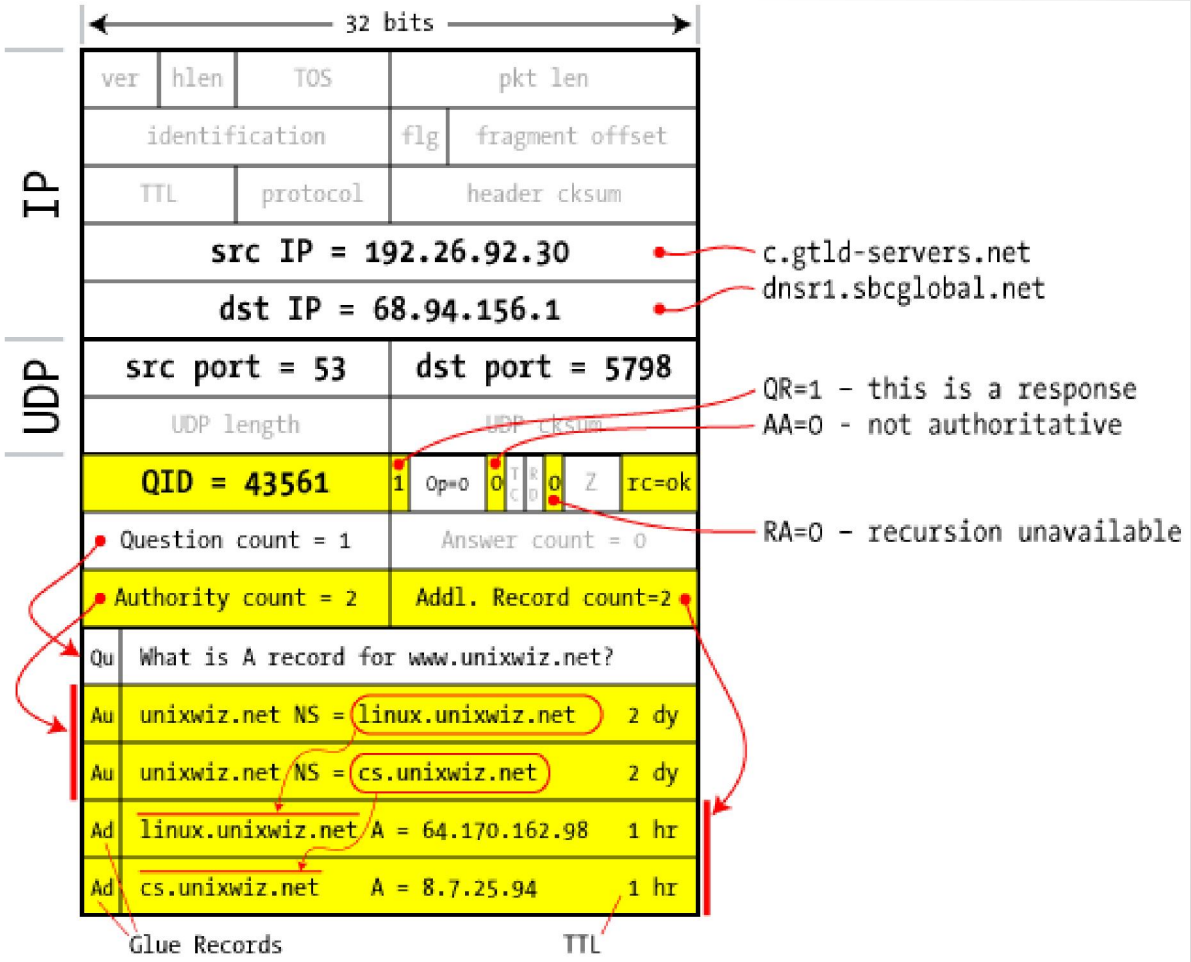
Resolver to NS request



Response to resolver

Response contains IP addr of next NS server (called “glue”)

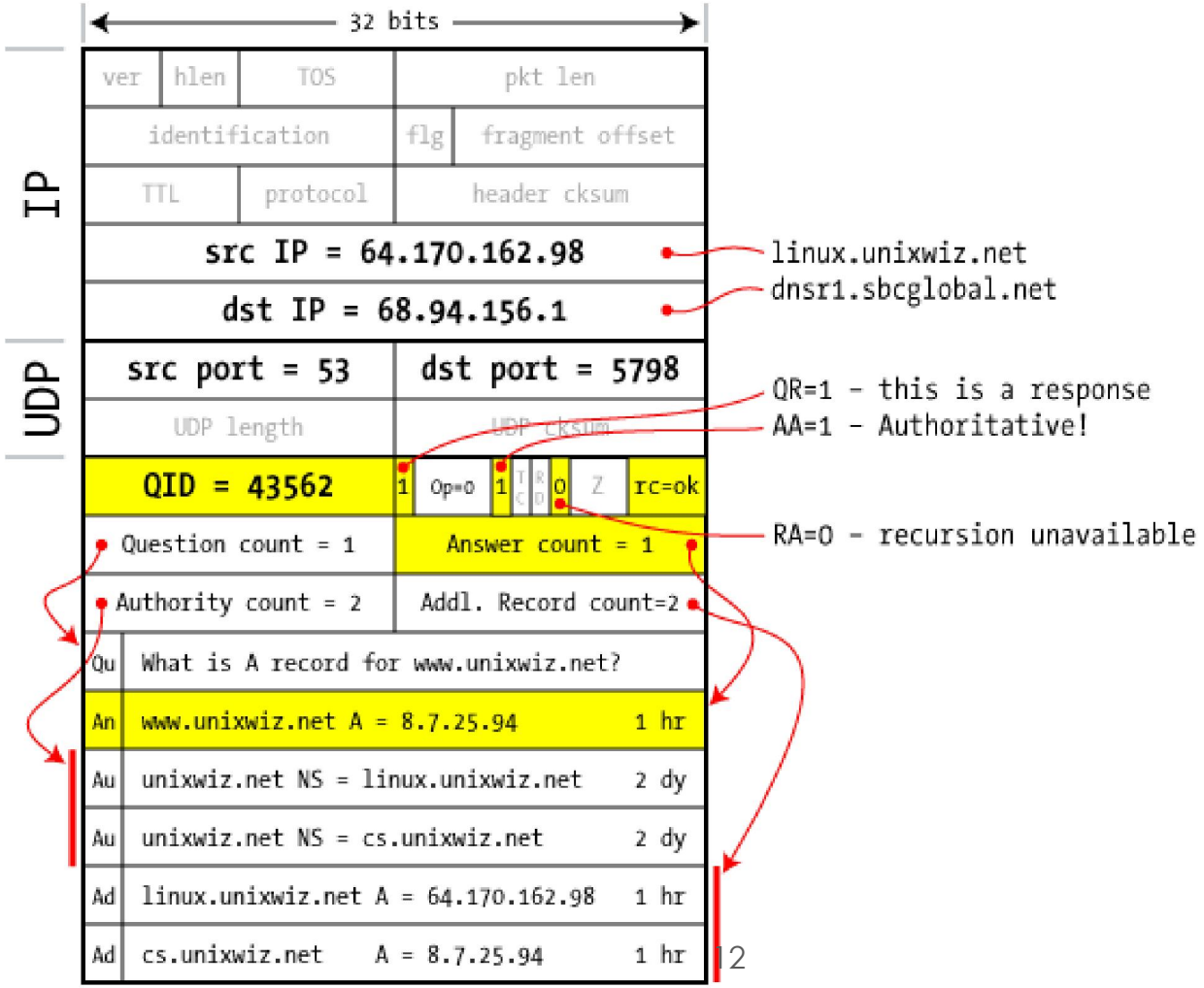
Response ignored if unrecognized QueryID



Authoritative response to resolver

bailiwick checking:
response is cached if it is within the same domain of query (i.e. **a.com** cannot set NS for **b.com**)

final answer →

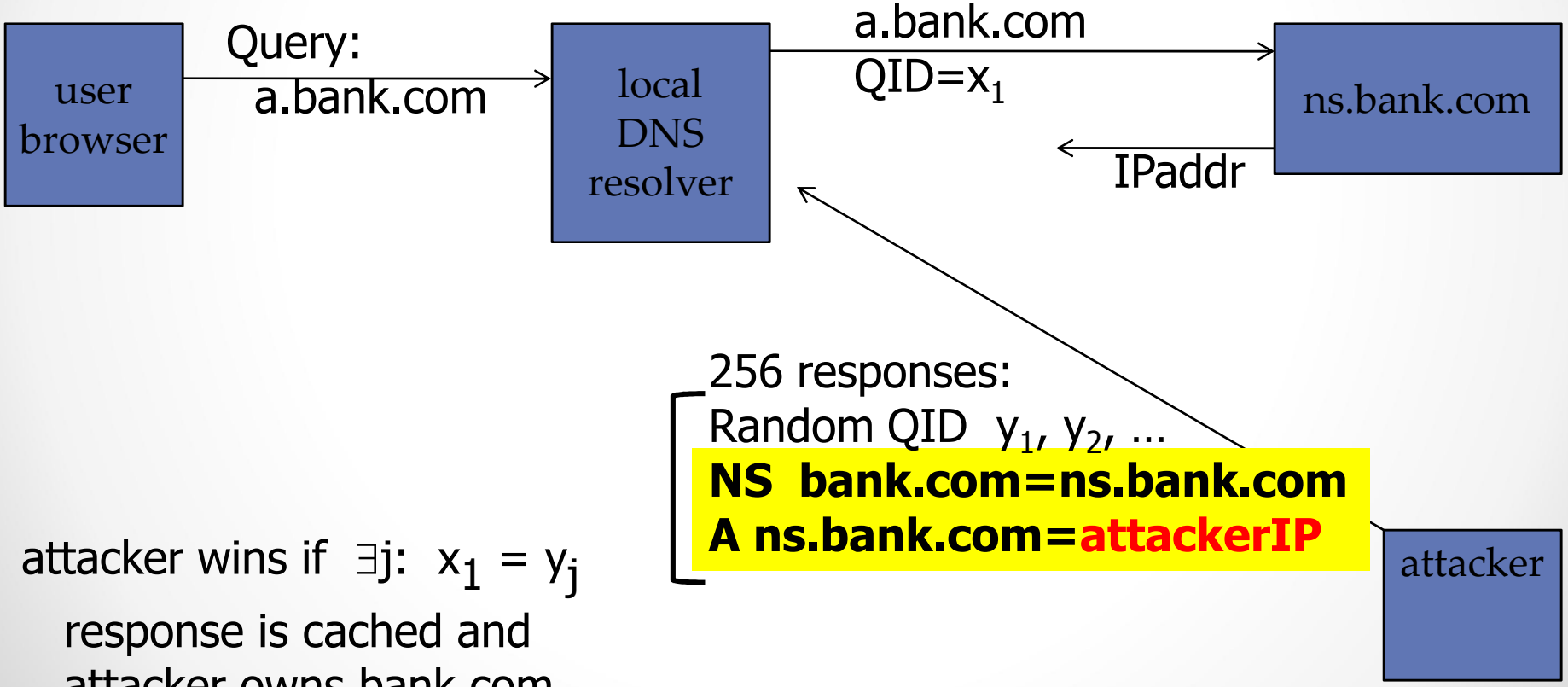


Basic DNS Vulnerabilities

- Users/hosts trust the host-address mapping provided by DNS:
  - Used as basis for many security policies:
    Browser same origin policy, URL address bar
- Obvious problems
  - Interception of requests or compromise of DNS servers can result in incorrect or malicious responses
    - e.g.: malicious access point in a Cafe
  - Solution – authenticated requests/responses
    - Provided by DNSSec
    - Unfortunately, DNSSec is not yet commonly implemented

DNS cache poisoning (a la Kaminsky' 08)

- Victim machine visits attacker's web site, downloads Javascript

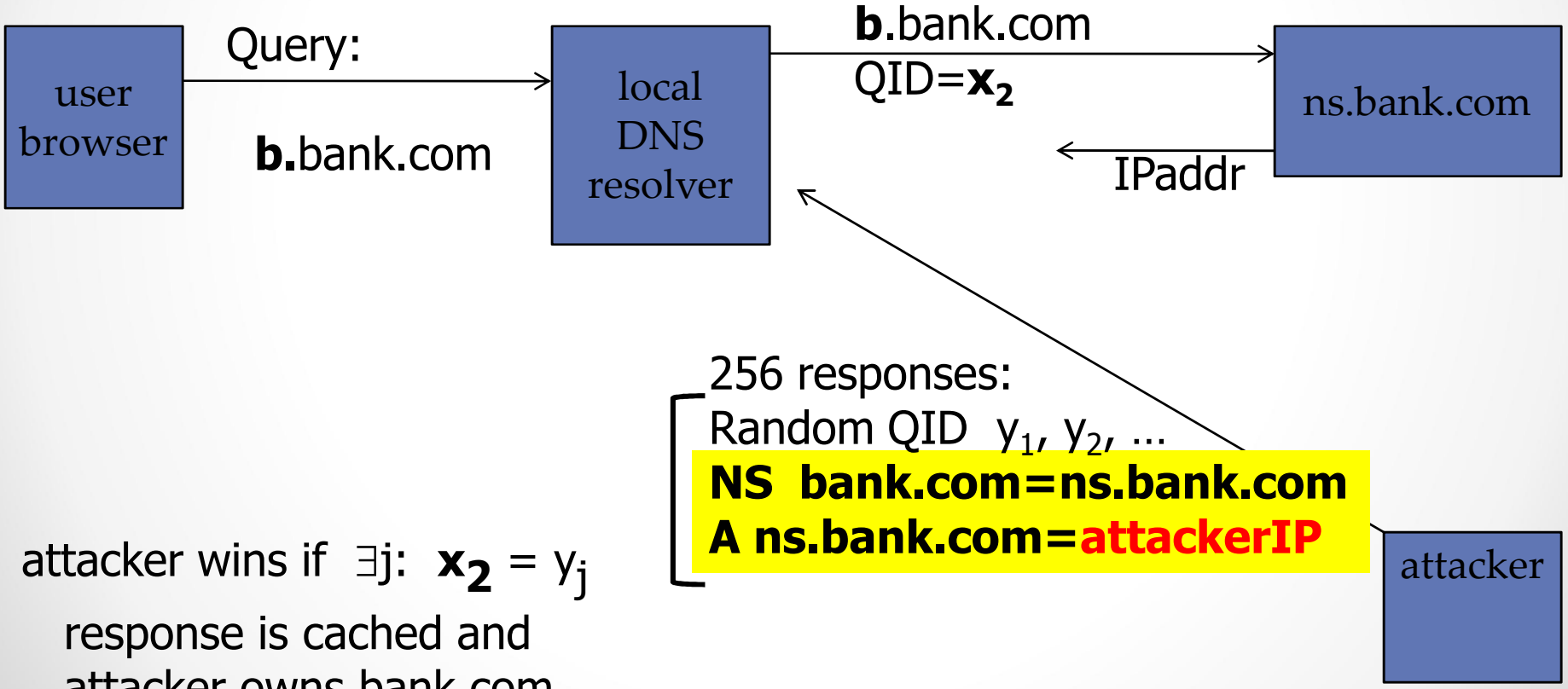


attacker wins if $\exists j$: $x_1 = y_j$

response is cached and attacker owns bank.com

If at first you don't succeed …

- Victim machine visits attacker's web site, downloads Javascript



attacker wins if $\exists j$: $x_2 = y_j$

response is cached and attacker owns bank.com

- success after ≈ 256 tries (few minutes)

Defenses

- Increase Query ID size. How?
- Randomize src port, additional 11 bits
  - Now attack takes several hours

- Ask every DNS query twice:
  - Attacker has to guess QueryID correctly twice (32 bits)
  - But the DNS system cannot handle the load

Questions?