DNS Deep Dive

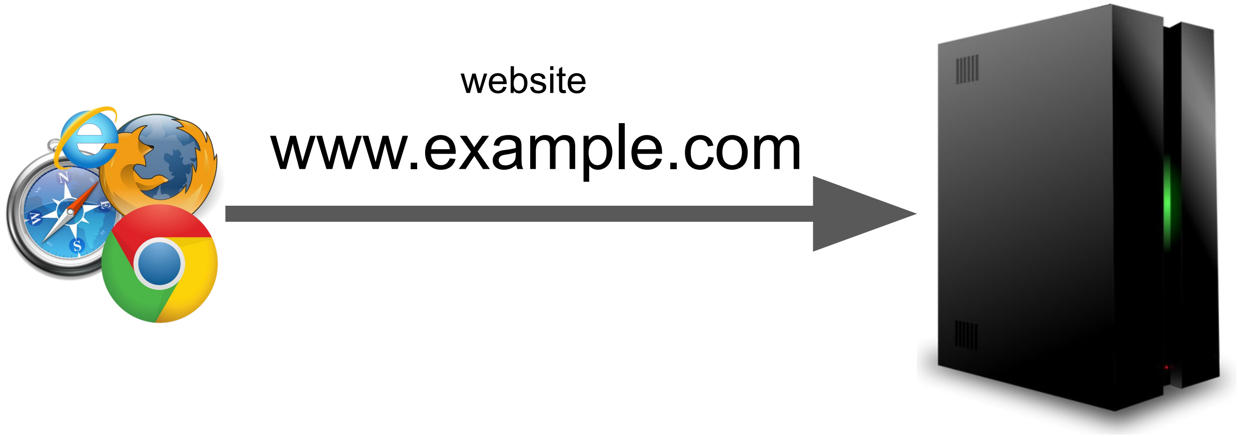
Wes Hardaker <hardaker@isi.edu>
Geoff Houston <gih@apnic.net>
João Damas <joao@apnic.net>

Note Well

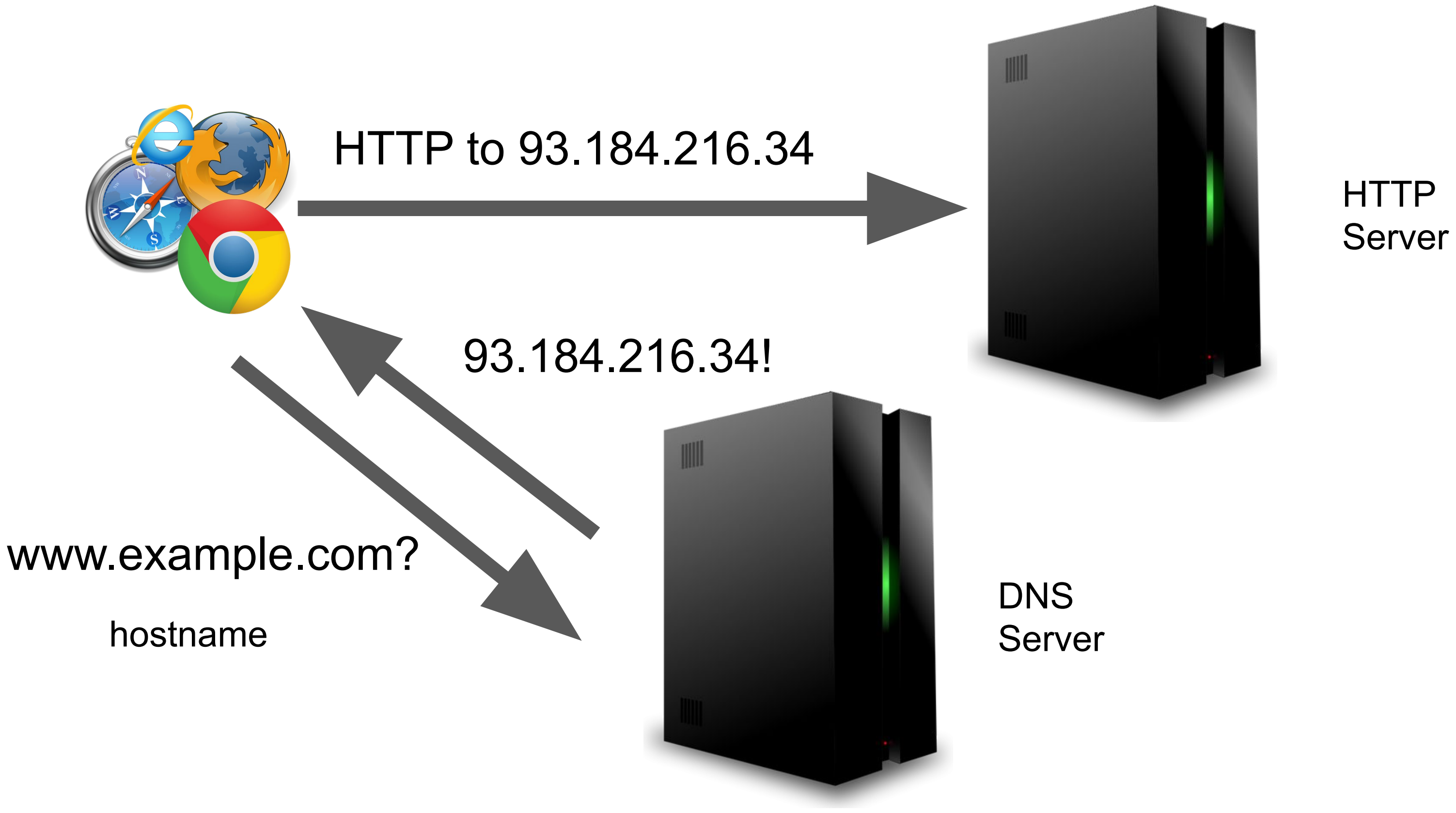
Overview

- Beyond the DNS basics
  - The underlying DNS distributed database model
  - DNS tree navigation basics
  - DNS Packet Evolution -- Some of the sharp / unusual edges of the protocol
  - Resource Record Types
- Resilience of the system
- DNS Software and APIs
- To be continued at IETF109?

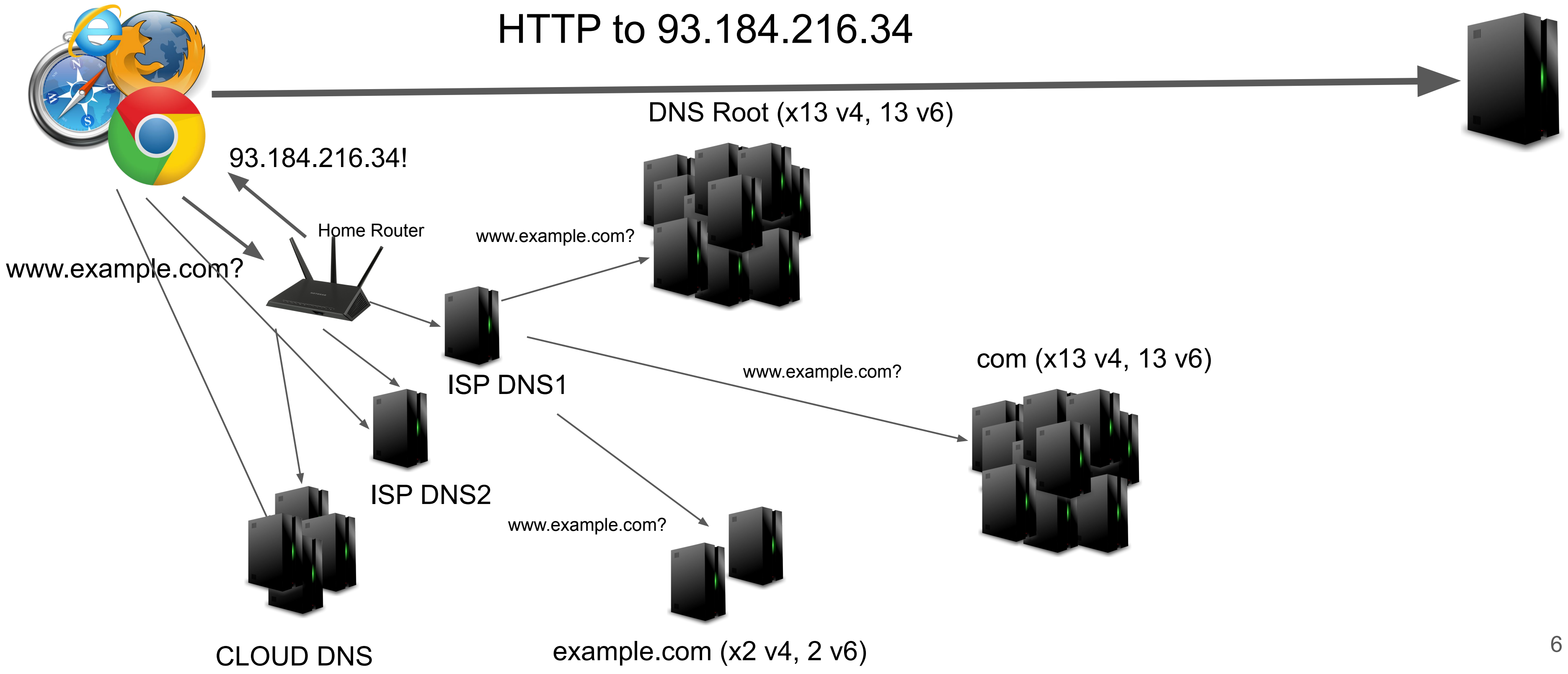
DNS as the novice Internet user sees it



DNS as the Techy Internet user sees it



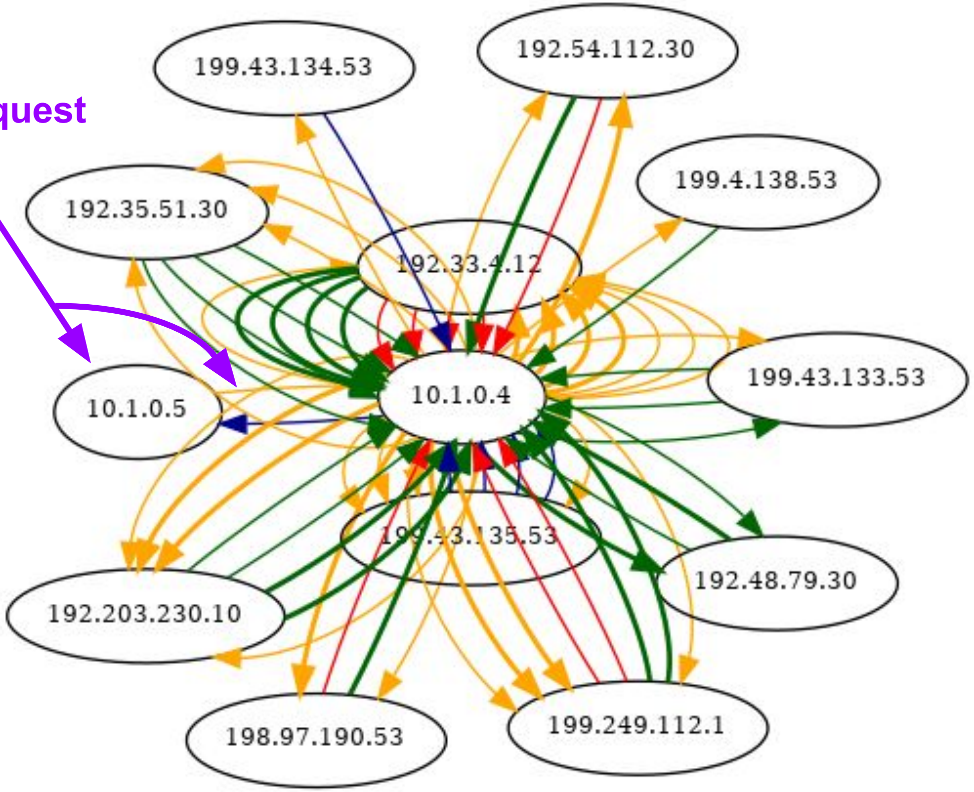
DNS is Much Much More Complex



The example.com web page

You make a single request

- Each line is a DNS request
- The center node is an ISP resolver

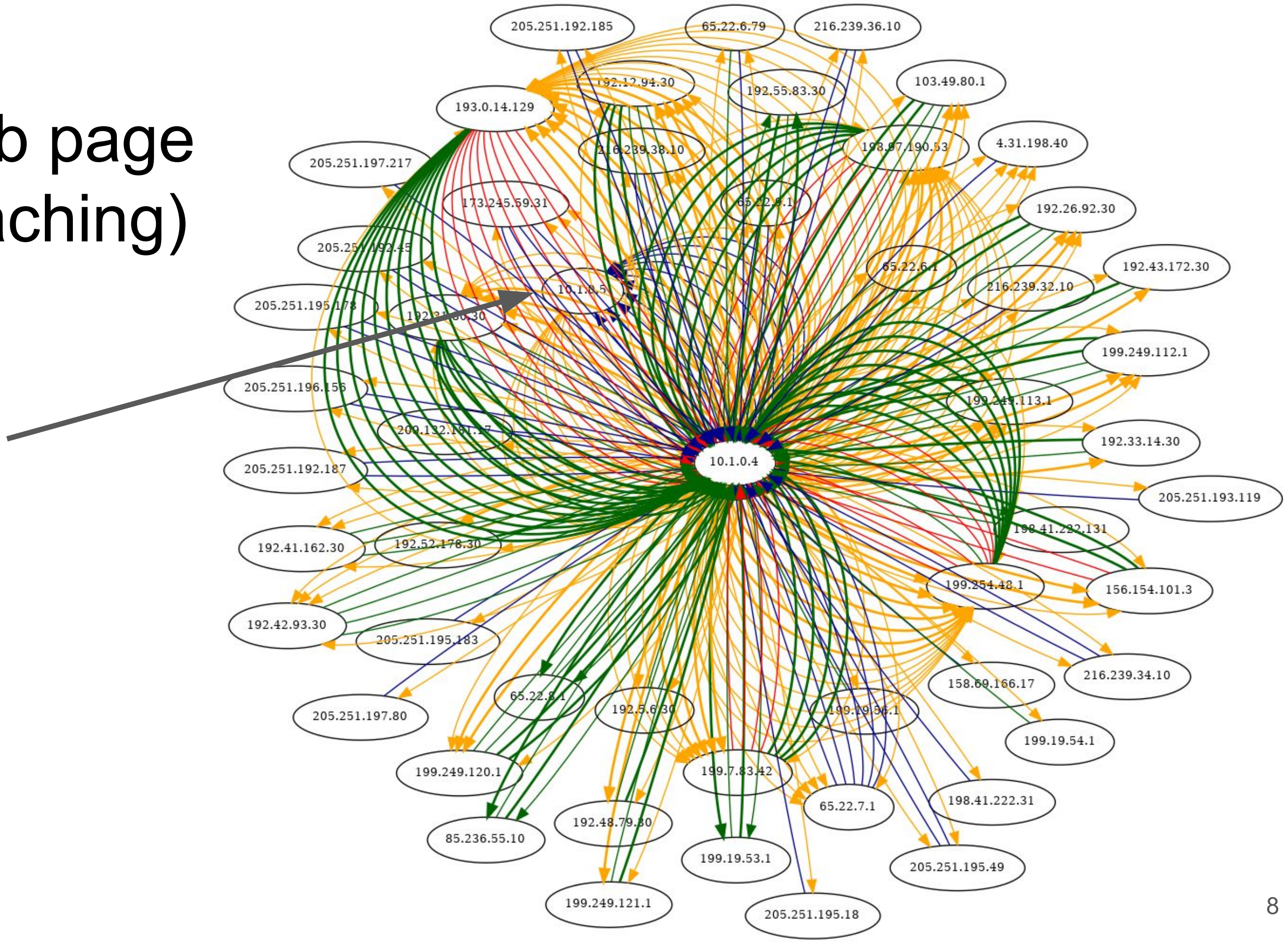


Query
Authoratative/DNSSEC

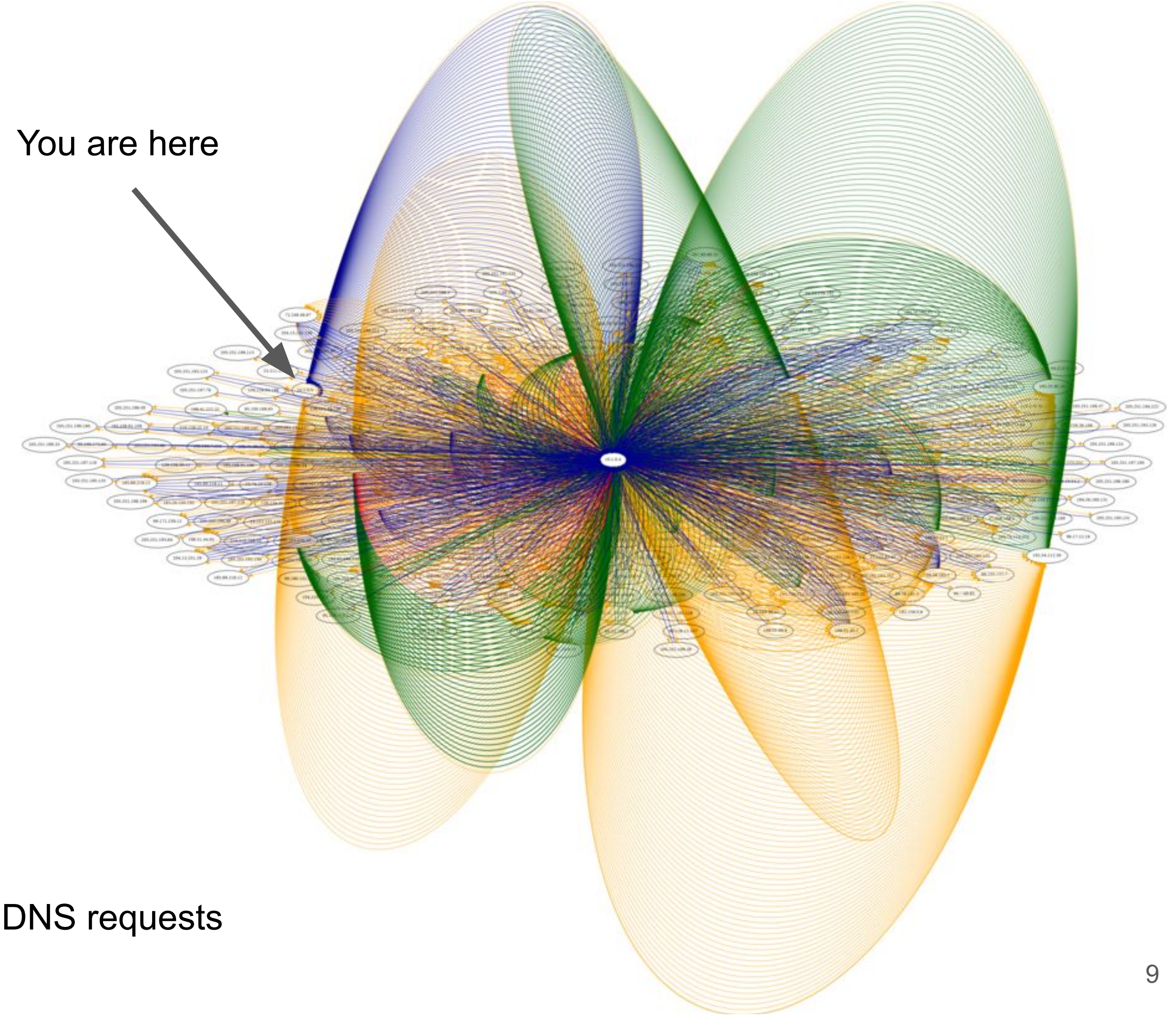
Truncated
Response

ietf.org web page (without caching)

You are here



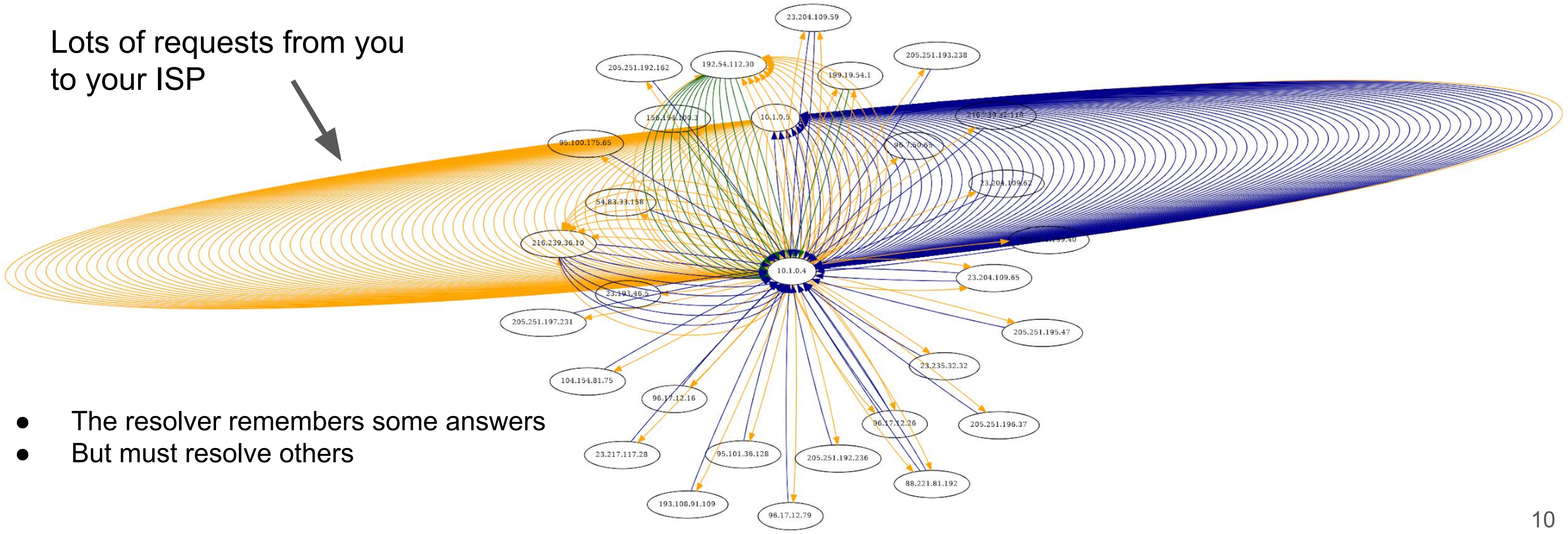
webmd.com (without caching)



TL;DR: Web pages generate many DNS requests

Webmd.com - after DNS caching

Lots of requests from you
to your ISP



- The resolver remembers some answers
- But must resolve others

The Underlying Distributed Model of the DNS

DNS was created as a replacement for /etc/hosts

Distributed system to replace static information

Back in **my day:**

```
127.0.0.1      localhost localhost.localdomain
::1            localhost localhost.localdomain
93.184.216.34  www.example.com
```

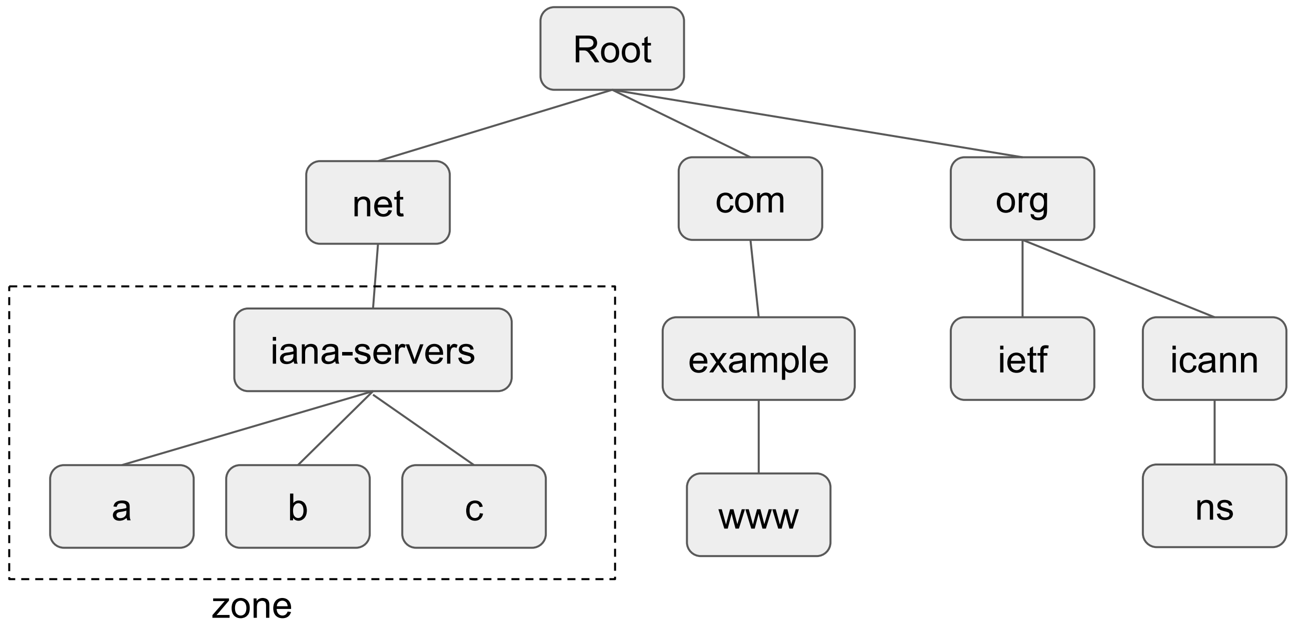
is all we needed.

The DNS ‘tree’ RFC103{4,5}

The Root (aka “.”)

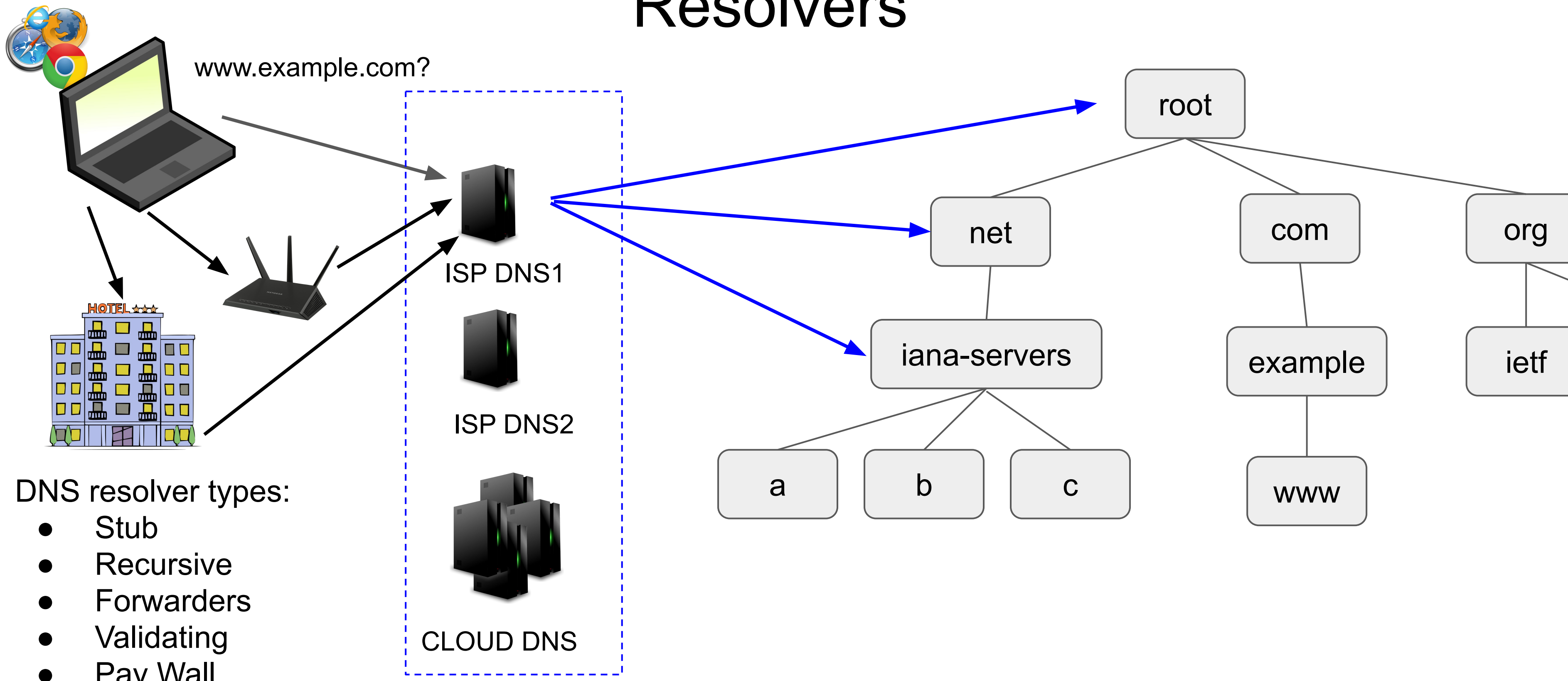
Top Level Domains (TLDs)

Second Level Domains (SLDs)



IMPORTANT: name server records in .net (13), .com (13), and .org (6) are not shown in these slides

Resolvers



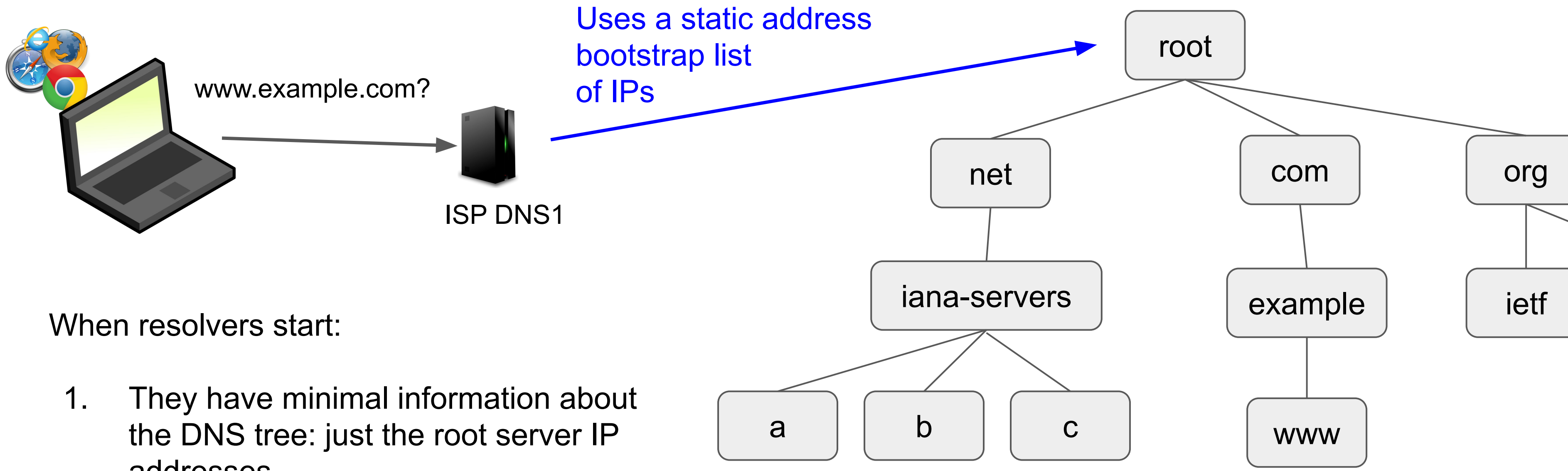
DNS resolver types:

- Stub
- Recursive
- Forwarders
- Validating
- Pay Wall

(to be described later)

Resolvers query the tree to find your answer

Priming Queries -- Bootstrapping Resolvers



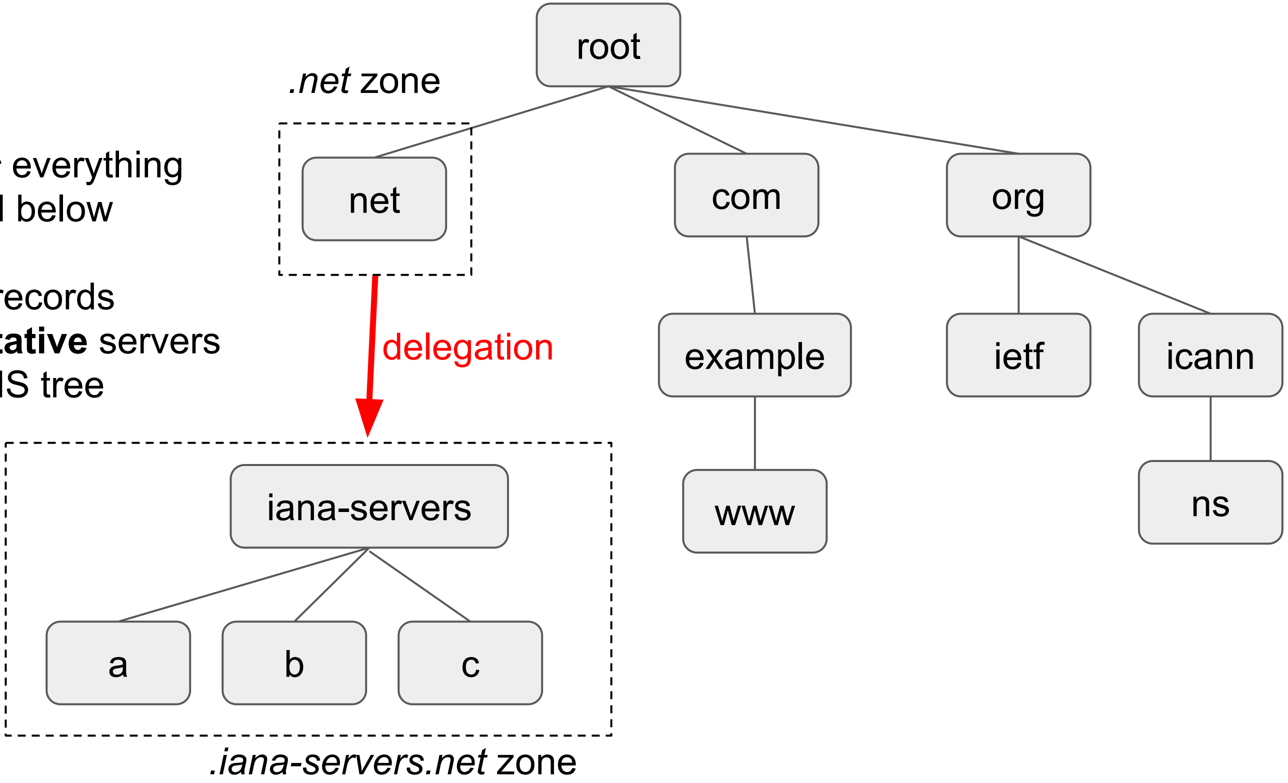
When resolvers start:

1. They have minimal information about the DNS tree: just the root server IP addresses.
2. The first thing they do is query them to ensure their hard-coded list is still correct

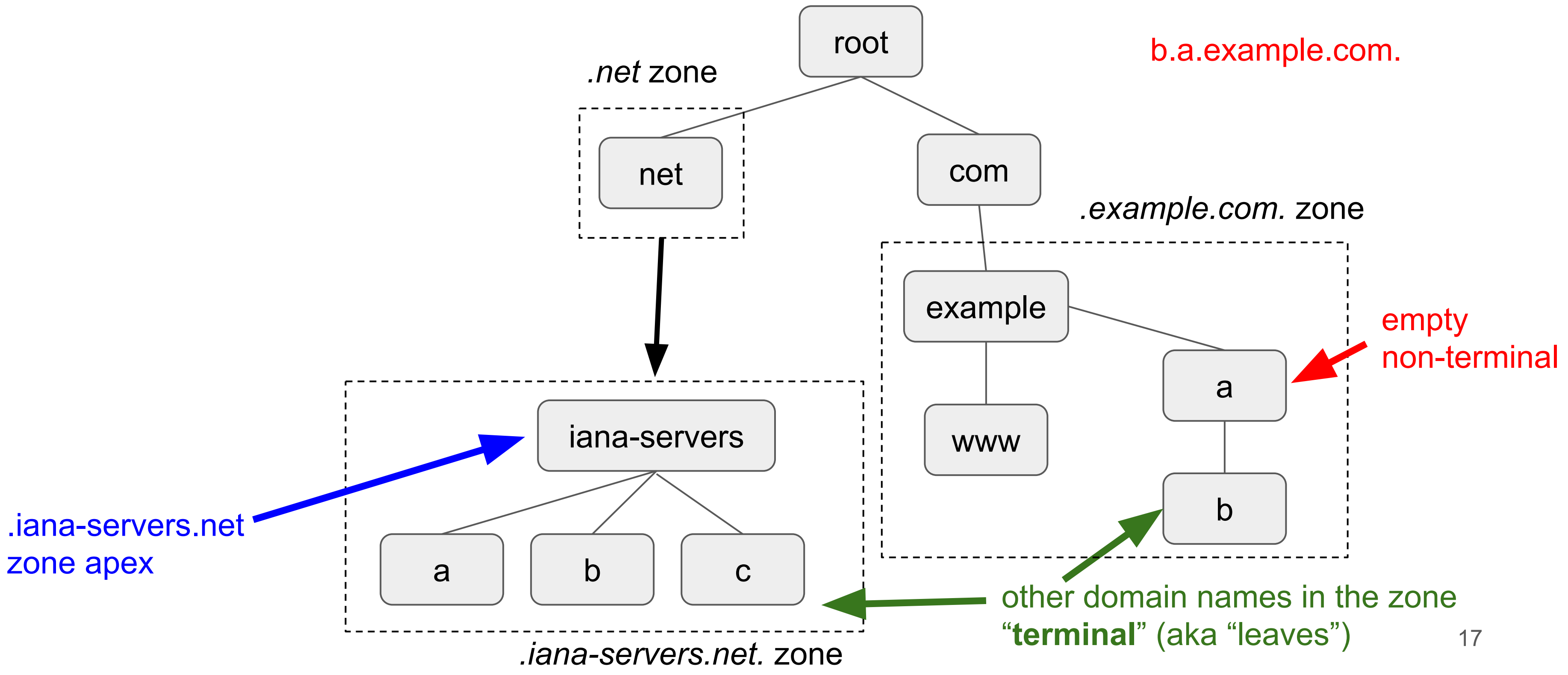
This is called a “priming query”

The DNS is a **distributed** protocol via **delegations**

The **.net** zone *delegates* everything in **.iana-servers.net** and below to **.iana-servers.net** using *nameserver* (***NS***) records that point to the **authoritative** servers for that portion of the DNS tree



Some DNS Terminology

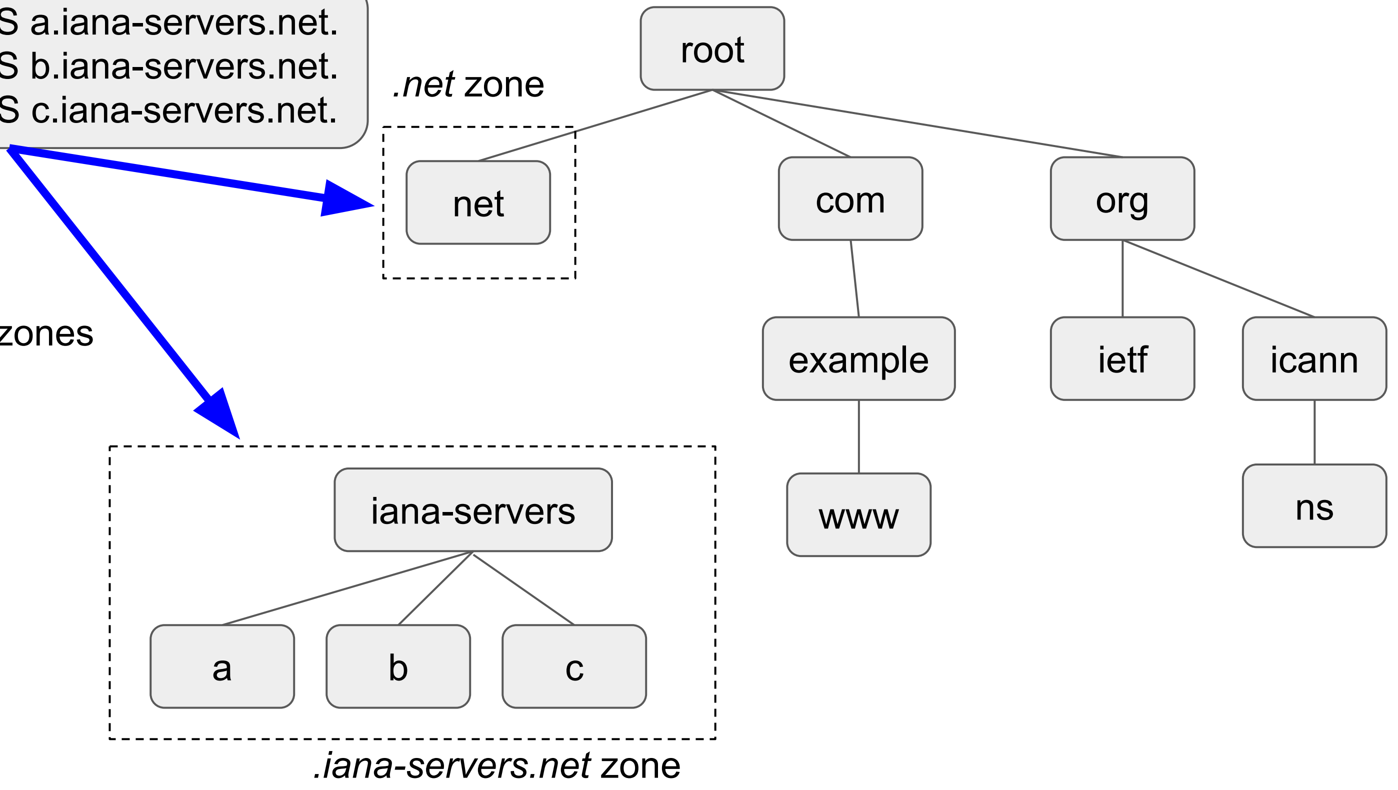


Duplicate records needed in parent/child zones

iana-servers.net. NS a.iana-servers.net.
iana-servers.net. NS b.iana-servers.net.
iana-servers.net. NS c.iana-servers.net.

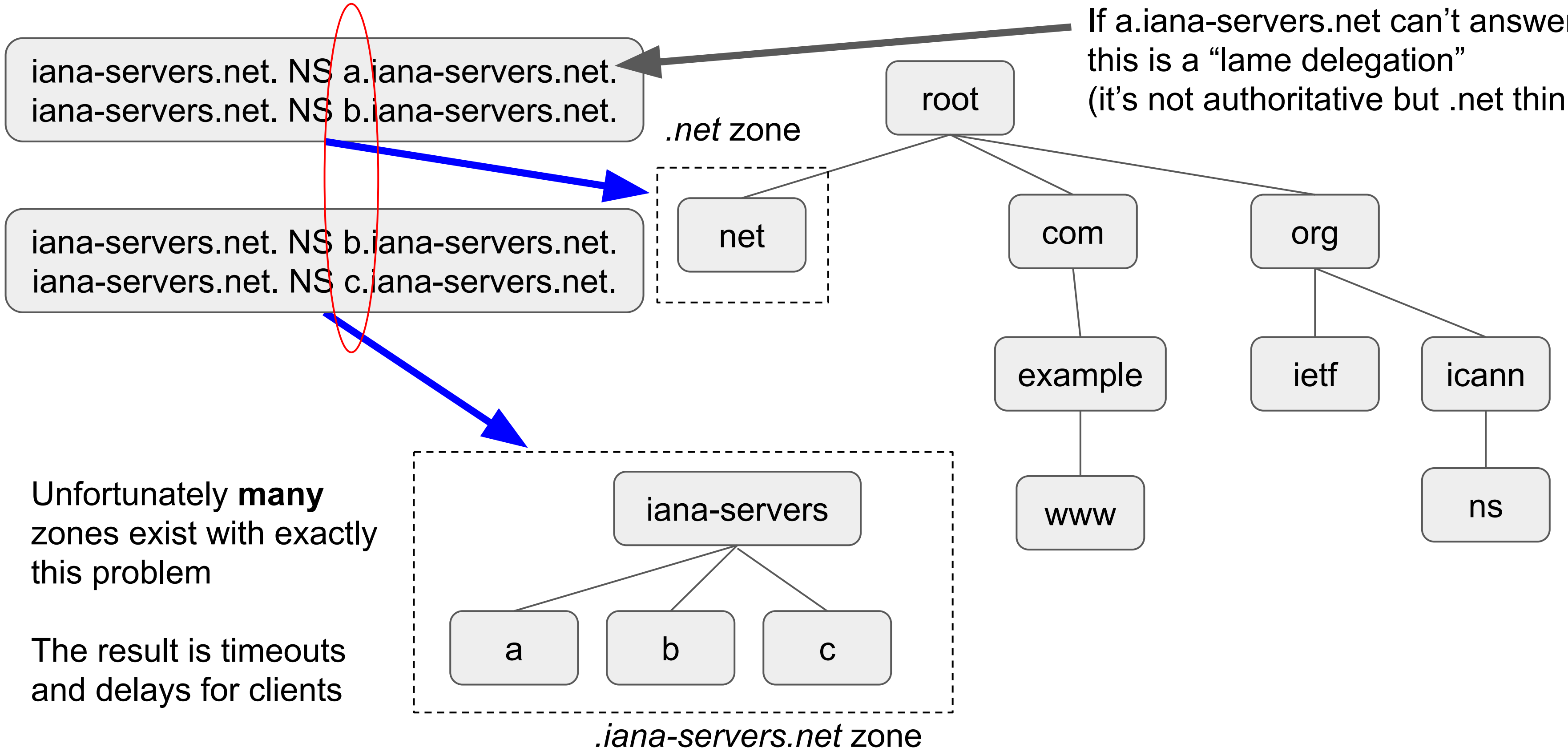
Should be in both zones

The child is the authoritative source!



Does this work? -- Yes but actually not well

If a.iana-servers.net can’t answer,
this is a “lame delegation”
(it’s not authoritative but .net thinks it is)



```
iana-servers.net. NS a.iana-servers.net.
iana-servers.net. NS b.iana-servers.net.
```

```
iana-servers.net. NS b.iana-servers.net.
iana-servers.net. NS c.iana-servers.net.
```

Unfortunately **many** zones exist with exactly this problem

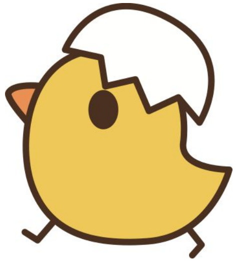
The result is timeouts and delays for clients

Trees that refer to the Forest

- Let's query *.com*'s servers about **example.com**:

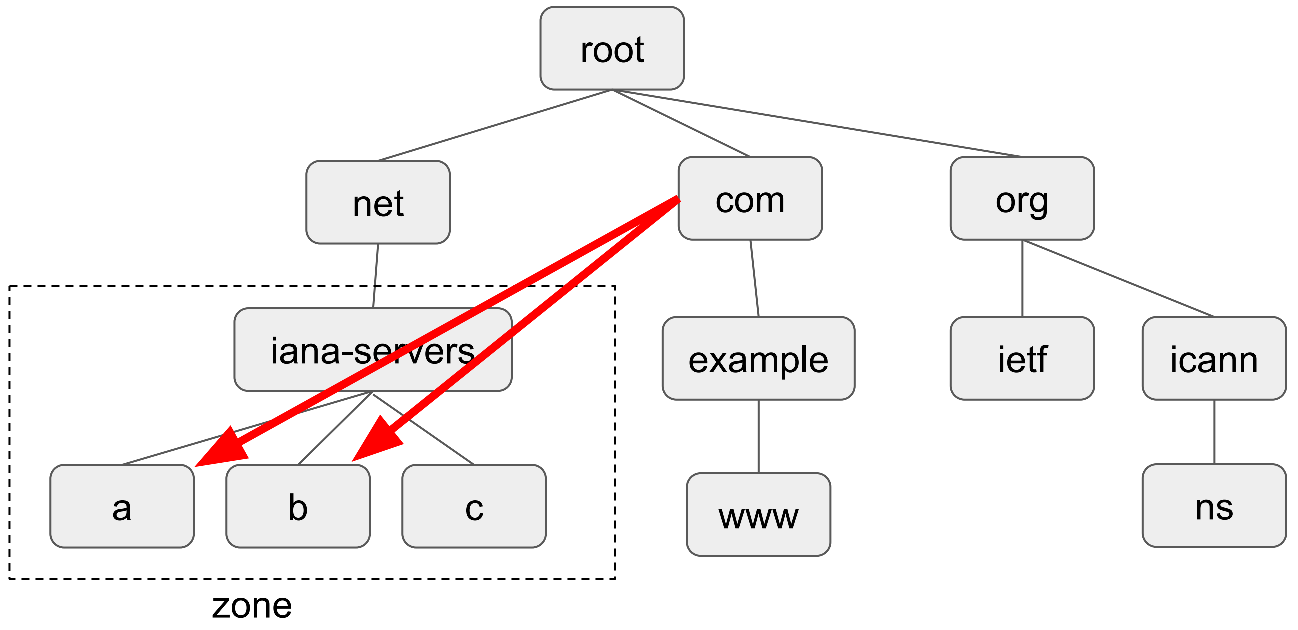
```
# dig @a.gtld-servers.net. www.example.com A
;; AUTHORITY SECTION:
example.com.      172800 IN NS a.iana-servers.net.
example.com.      172800 IN NS b.iana-servers.net.
```

2 day TTL



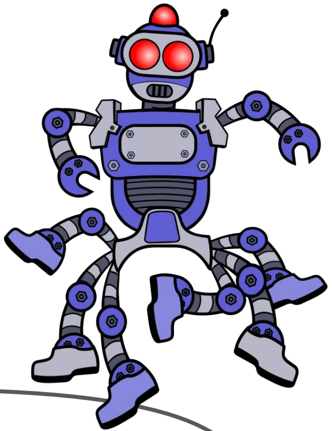
- The answer: *.com* doesn't know where *www.example.com* is
- But it does know where to send you next: **to IANA-SERVERS.NET**
- ***But where is IANA-SERVERS.NET???***
    - *(here we go again)*

Finding Authoritative Servers -- Pictorially



If you **ask *.com*** where ***www.example.com*** is, they tell you to go ask a **completely different part of the tree**

Tricky Tree Grafting -- AKA, what is glue?



```
# dig @c.gtld-servers.net. iana-servers.net ns    (asking .net)
;; ANSWER SECTION:
iana-servers.net.          956  IN   NS   a.iana-servers.net.
iana-servers.net.          956  IN   NS   ns.icann.org.
iana-servers.net.          956  IN   NS   c.iana-servers.net.
iana-servers.net.          956  IN   NS   b.iana-servers.net.
```

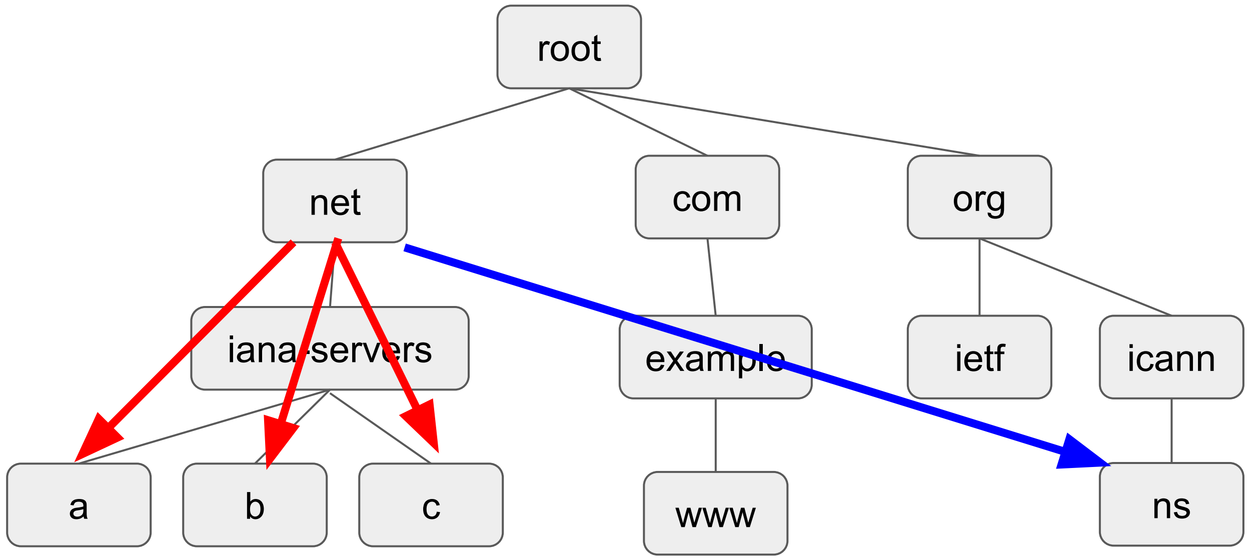
Glue!

How do I talk to *a.iana-servers.net* if it's *inside iana-servers.net itself??*

```
;; ADDITIONAL SECTION:
a.iana-servers.net.   956  IN   AAAA     2001:500:8f::53
b.iana-servers.net.   956  IN   AAAA     2001:500:8d::53
...
```

(note the random ordering of the answer section)

Including Glue



- *.net’s nameservers* knows where **the authoratative source for iana-servers.net is**
- “In-balliwick” name servers are within the zone itself
  - But {a,b,c}.iana-servers.net Must have glue records!
- “Out-of-balliwick” servers are external
  - *ns.icann.org* is out-of-balliwick for *iana-servers.net*

DNS Packet Evolution

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                      NAME                     /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

DNS - A very very simple protocol

- DNS packets ship resource records around
- All ***Resource Records*** are composed of a triplet
  - A Query Name "www.example.com" (aka a "domain name")
  - A Query Type AAAA = IPv6 address
  - A Query Class IN = Internet *(aka, almost the only value used)*
- Resource Record Sets
  - ALL matching combinations are an atomic unit
  - You can't ask for "just 2"
  - They are **not ordered**
- Response Records also contain
  - A *"Time To Live"*
  - Response Data

DNS Packet Components

- Header
  - Transaction ID
  - Flags
  - Number of records in each section
- DNS Resource Record Sections
  - Question
  - Answer
  - Authoritative
  - Additional

Why are multiple questions a problem?
- Do you wait for all authoritative answers?
- What if one authoritative answer has an error and another doesn't?
- What if there are two different errors?
- ...

RFC1035:
[This] section contains QDCOUNT (usually 1) entries

DNS Packet Sections

- Question
  - Where the (single) question goes
  - Repeated in a response
- Answer
  - The answer to the question
- Authoritative
  - What DNS server is the “true” source for the answers
- Additional
  - Anything else you might want to know
    - But shouldn’t trust!
  - E.G., Glue

What happens when DNS things go wrong?

The DNS packet headers contain an “response code” (RCODE) field, yay!

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
```

Drat, it’s only **4 bits…** There are way more than 16 problems

Let’s get creative about the RCODE problem

What if….

Now bear with me….

What if….

We stuck the extra bits somewhere else?

And thus, the “**OPT**” (pseudo-) resource record was created

EDNS0’s “OPT” record -- more bits! RFC2671

- An **“extend” pseudo resource** record to add to the **additional section**
- DNS servers only respond with one if the **client indicates support**
- **Required** to support some protocol modifications (e.g. DNSSEC)
- Reuses the Resource Record byte format, but **changes many fields**
- **Features:**
  - Total RCODE size becomes 4 + 8 = 12 bits
  - Supports additional protocol flags
  - Adds application level max message size / PMTU type discovery
  - Adds support for additional DNS extensions
- Used for other extensions:
  - Client Subnet in DNS Queries (RFC7871)
  - Extended errors (RFC-TBD)
  - ...

OPT Resource Record Field Reusage

RR Field	New Meaning
NAME	Must be empty
TYPE	OPT(41) (16 bits)
CLASS	UDP Payload Size (16 bits) -- max response accepted
TTL (32 bits)	Extended RCODE (8 bits), version (8 bits = 0) and Flags (16 bits)
RDLEN	Data length (same)
RDATA	Atribute (16-bit)/value (variable length) pairs

Truncation

What happens when a response is too big?

- Greater than the client said it could handle in the OPT/UDP Payload Size

A few things:

- The Truncation bit (TC) is set
- Resource records are removed from the response to make it fit. Maybe.
  - Some try to remove unimportant items (the additional section goes first)
  - Some servers drop everything and just expect clients to use TCP
  - Response Rate Limiting (RRL) -- a DDoS defense -- triggers the TC bit due to query frequency
- Clients need to come back over TCP to get the full answer
  - Sometimes clients come back and sometimes they don't if they got the answer they wanted

Ok, but what if you need MOAR errors, text, etc...

What if….

Now bear with me….

What if….

We stuck the extra bits somewhere else?

A soon to be RFC: extended errors!Another OPT

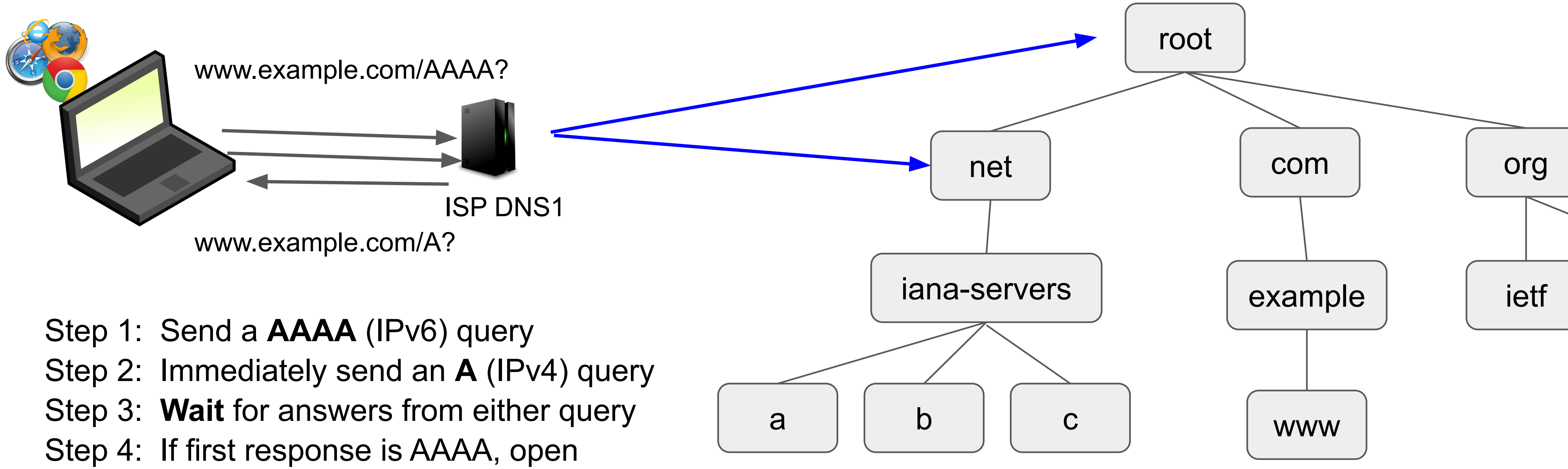
(it's errors all the way down)

DNS Resource Record Types

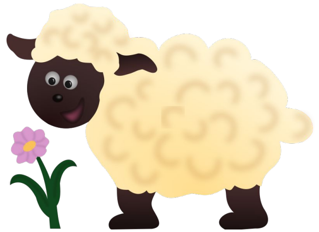
Resource Record Types

Type	Content
A	IPv4 Address
AAAA	IPv6 Address
SOA	Zone information at the APEX
TXT	Free-form text blob

IPv4/IPv6 Deployment: Happy Eyeballs (RFC8305)



Step 1: Send a **AAAA** (IPv6) query
Step 2: Immediately send an **A** (IPv4) query
Step 3: **Wait** for answers from either query
Step 4: If first response is AAAA, open connection. If first response is A, wait a bit (50ms) for a AAAA and then give up and open an IPv4 connection with sadness.
Step 5: **Profit** from your dual-stack deployment!



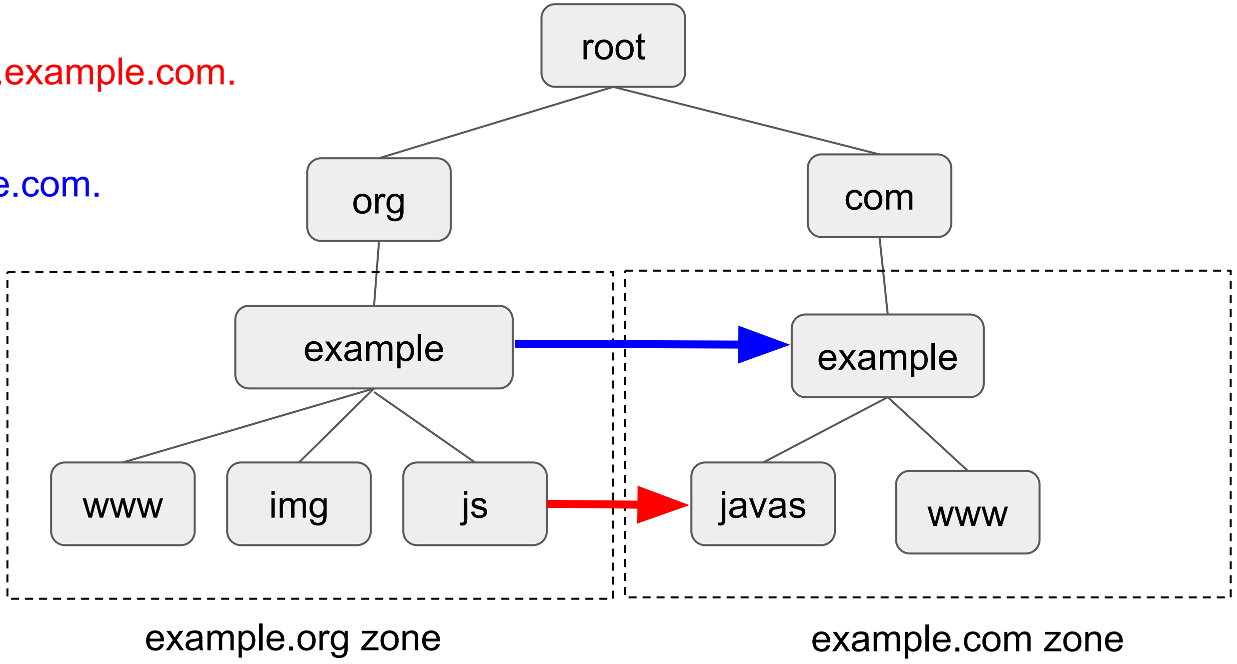
CNAMEs and DNAMEs

js.example.org. 3600 IN CNAME javas.example.com.
CNAMEs cannot occur at the apex

example.org. 3600 IN DNAME example.com.

CNAMEs are aliases for
other tree elements
(can be in the same zone or
in another)

DNAMEs are aliases for
zones themselves

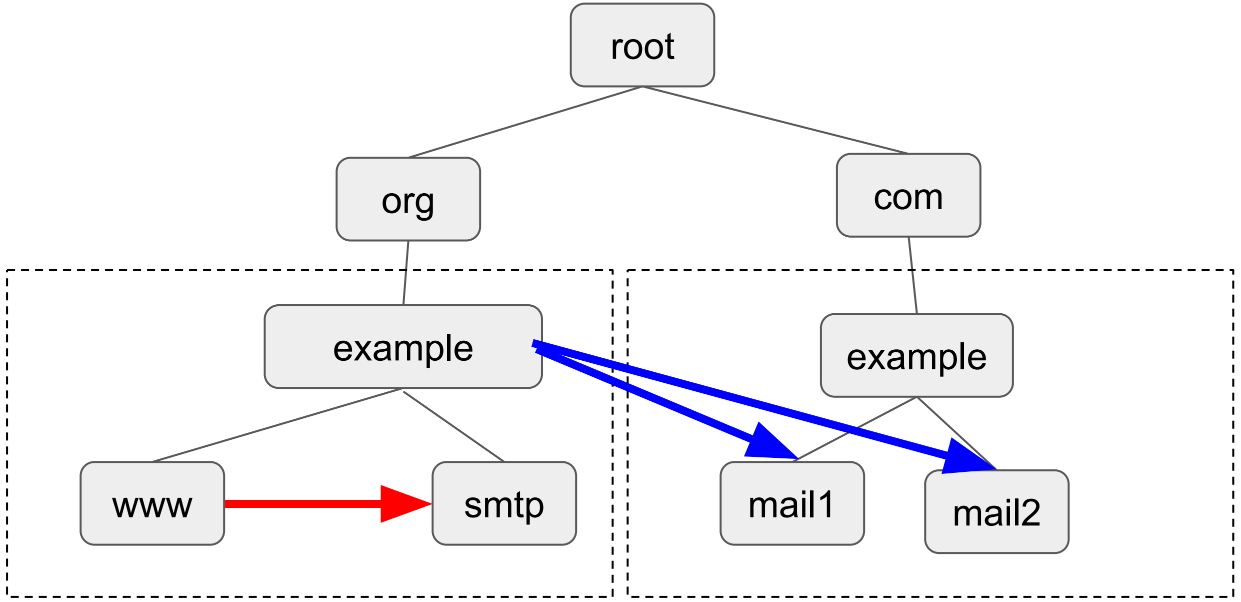


IMPORTANT: CNAMEs MUST exist alone at a name (minus DNSSEC entries)
IMPORTANT: CNAMEs point to ALL records at the other name (A, AAAA, NS, MX, etc)

MX Records

Mail Exchange (MX) records

- Where should e-mail for a domain-name be sent?
- Prioritized contact list



```
www.example.org.     3600 IN AAAA    2606:2800:220:1:248:1893:25c8:1946
www.example.org.     3600 IN MX      5 smtp.example.org.

example.org.         3600 IN AAAA        93.184.216.34
example.org.         3600 IN MX          10 mail1.example.com.
example.org.         3600 IN MX          20 mail2.example.com.
```

Outsourcing mail service is very common

Wildcards (RFC4592)

- Generating responses for missing data
  - Left most label must be a “*” (and only a “*”)
  - Matches any label that doesn’t already exist
    - Including sub-labels under it
  - Causes a nameserver to **synthesize and answer**
  - **Please read RFC4592**! Good examples therein.
- Example records:

```
*.example.com.      3600 IN MX  10 mail.example.com
host1.example.com.  3600 IN A   192.0.2.1
```

- Reponses:

```
    host1.example.com/MX    MATCHES
    host2.example.com/MX    MATCHES
    host1.example.com/A     DOESN’T MATCH (returns 192.0.2.1)
    host2.example.com/A     DOESN’T MATCH (returns NXDOMAIN)
```

Underbar labels: “_foo” (RFC855{2,3})

- For a long time people kept putting TXT records at the APEX
  - SPF
  - DKIM
  - DOMAINKEY
  - DNS ownership verification (google, facebook, docusign, …)
  - …
- The “right” solution was to use a new RRTYPE rather than TXT
  - But this was slower to deploy
- The new solution: use TXT and RRTYPE records at “_” prefixes

_spf.example.com.	IN TXT	- The right “new” for SPF
_domainkey.example.com.	IN TXT	- DKIM key publishing
_25._tcp.mail.example.com.	IN TLSA	- DANE for secured SMTP (RFC7672)
_imaps._tcp.example.com.	IN SRV	- Service host discovery

Summary: DNS is a global distributed identifier DB

Yes, but how does this all scale so well?

I have no idea

Let’s ask Geoff

Extended Errors RFC -- in the RFC editor’s queue

- **SERVFAIL** error is the standard “I couldn’t” response
  - Operators are clueless as to why
  - e.g. most types of DNSSEC validation failures triggers this
- Extended error **adds context** for SERVFAIL (and others)
- With **optional text** providing greater debugging detail

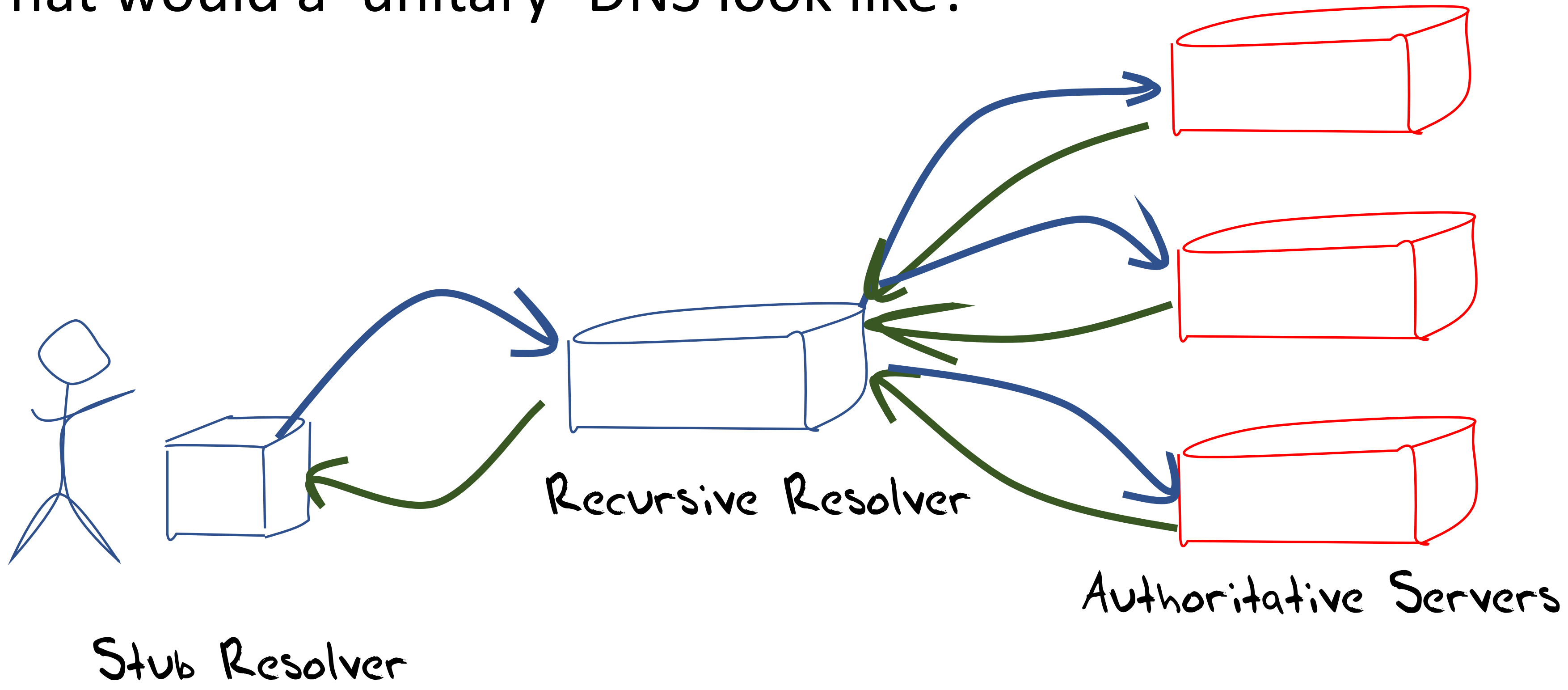
DNS Deep Dive, IETF 108

Resilience

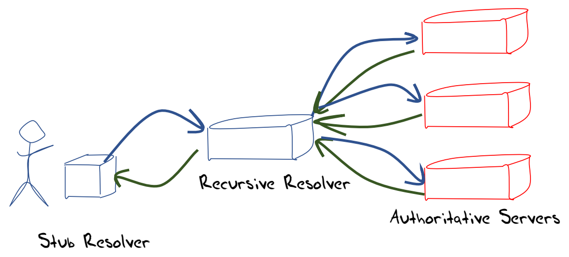
Resilience and Replication

One and Only One?

What would a ‘unitary’ DNS look like?



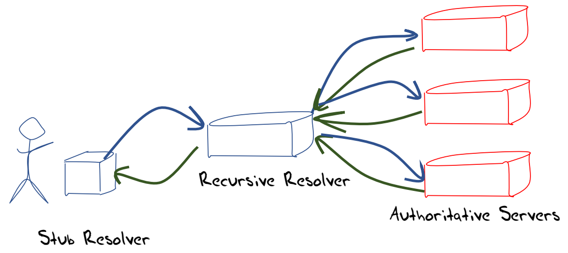
One and Only One?



What would a 'unitary' DNS look like?

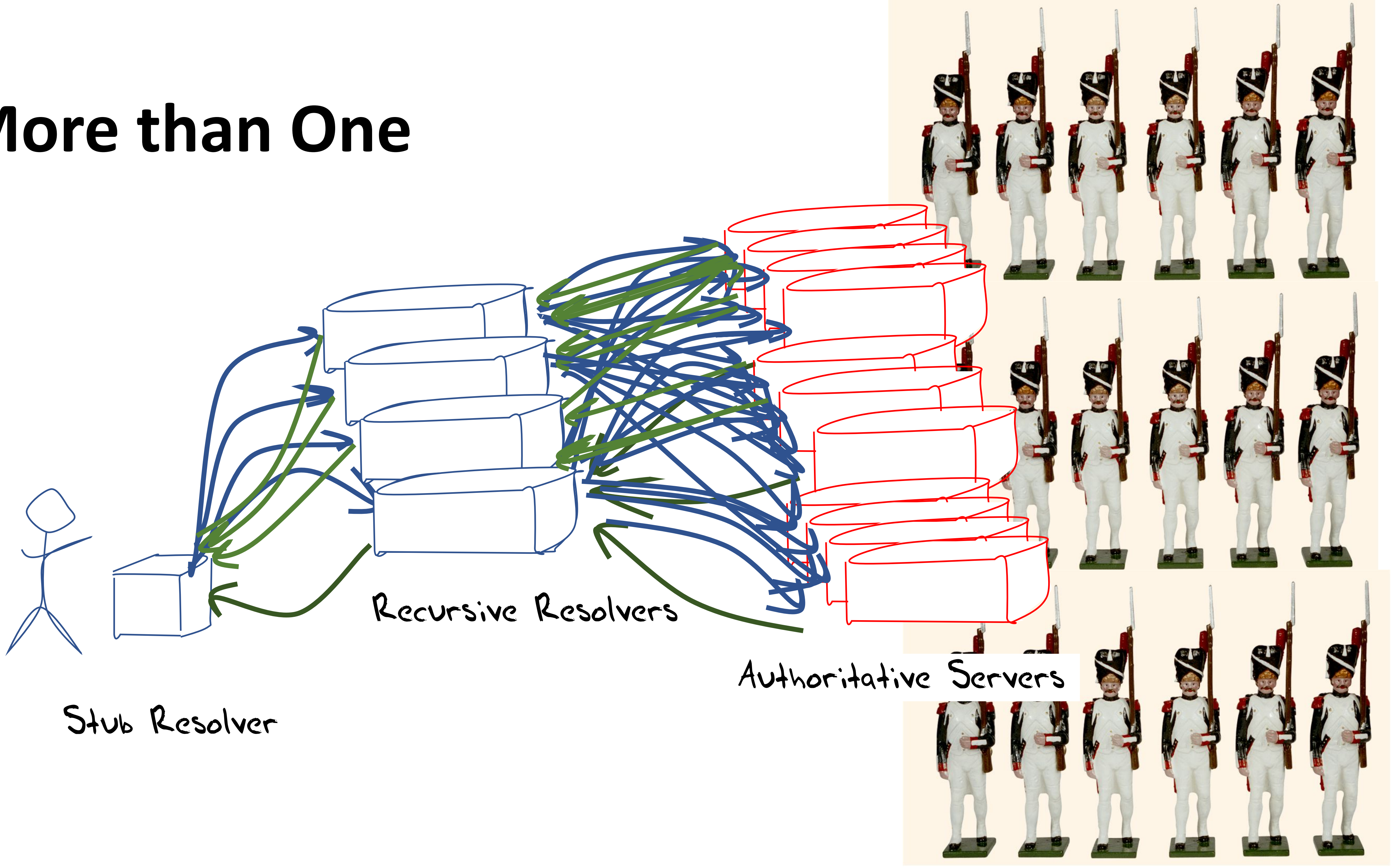
- Each stub resolver has a single address for a recursive resolver
- The stub resolver uses a single query for each name to be resolved
- Each domain is served by a single authoritative server with a single address
- Each recursive resolver uses a single query model

One and Only One = Fail!



- Obviously this model has multiple single points of failure
  - Recursive resolver failure
  - Authoritative server failure
  - UDP packet loss
- And each of these SPOFs becomes a point of vulnerability for hostile attack

More than One



More than One

Resolv.conf

- Stub resolvers are typically configured with up to three recursive resolvers
  - You can add more but…
- In a dual stack world each of these resolvers has both IPv4 and IPv6 addresses and queries are made in parallel via both protocols (1)
- The order is usually important (2), and subsequent resolvers are only queried if there is a UDP timeout from prior queries (3)

1. Unless the "single-request" resolv.conf option is enabled

2. Some measurement of behaviour of Chromium showed a round-robin pattern of use. This round-robin behaviour can be selected on some systems with the resolv.conf option "rotate"

3. Or the resolver receives a response with response code RCODE 2, (SRVFAIL)

More than One

Authoritative Servers

- DNS domains are typically configured with more than one authoritative server
- The root zone uses 13 different server names each with IPv4 and IPv6 addresses
- Between 2 and 4 authoritative servers seems to be common practice these days
- For most applications 5 or more services is probably excessive, and 1 is regarded as too few (1)
- The server list MAY round robin depending on the authoritative server code

1. A single unicast server is probably not enough, and a single anycast address with a set of servers using the same address may still not be enough to be resilient! Diversity is the key here, not mindless replication

Resilience in Transport – not!

UDP

- All bets are off!
- You have no idea if the query or the response was lost
- The UDP sender uses a timeout to determine a “no answer” condition
- The timeout value depends on context:
    - Stub resolvers are observed to use timeout values between 1 and 5 secs (1)
    - Recursive resolvers are seen to use use values between ~300ms and 1 sec
- Should you ask the same server again? Or maybe change the address/protocol? Or maybe ask a different server?
- When to just give up? The number of timeouts is typically limited (2)

1. resolv.conf sets the timeout to 5 seconds by default, with a minimum of 1 and a maximum of 30 configurable in resolv.conf
2. resolv.conf sets the number of attempts to 2 by default with a maximum of 5 configurable in resolv.conf

Resilience in Transport

TCP

- If you can reach the server then you can reasonably expect a response
    - Or an error code or some sort
- Asking the same server again is kinda pointless, particularly if you expected a different answer
    - But sometimes different recursives hide behind the same name or even the same protocol and protocol address
- Asking a different server might be helpful if you didn’t like some first answers (SERVFAIL or possibly REFUSED) but generally it’s not all that useful

Resilience in Transport++

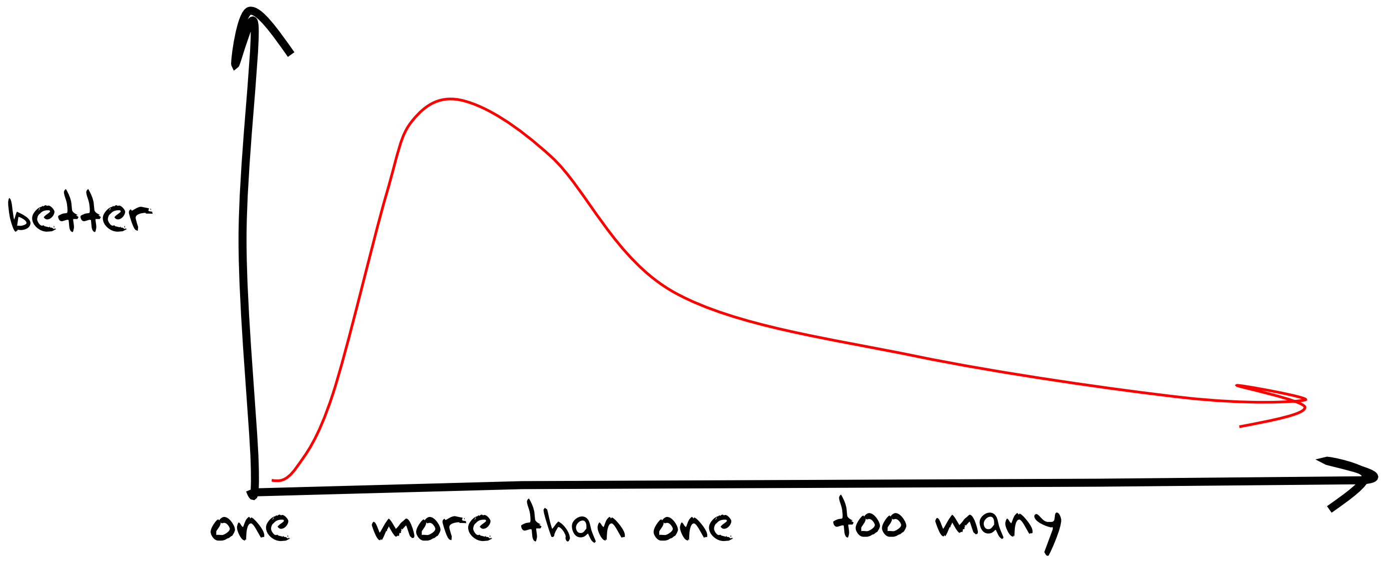
- DNS over TLS over TCP (DoT)
- DNS over HTTPS over TLS over TCP (DoH)
- DNS over QUIC over encrypted TCP-like transport (DoQ)

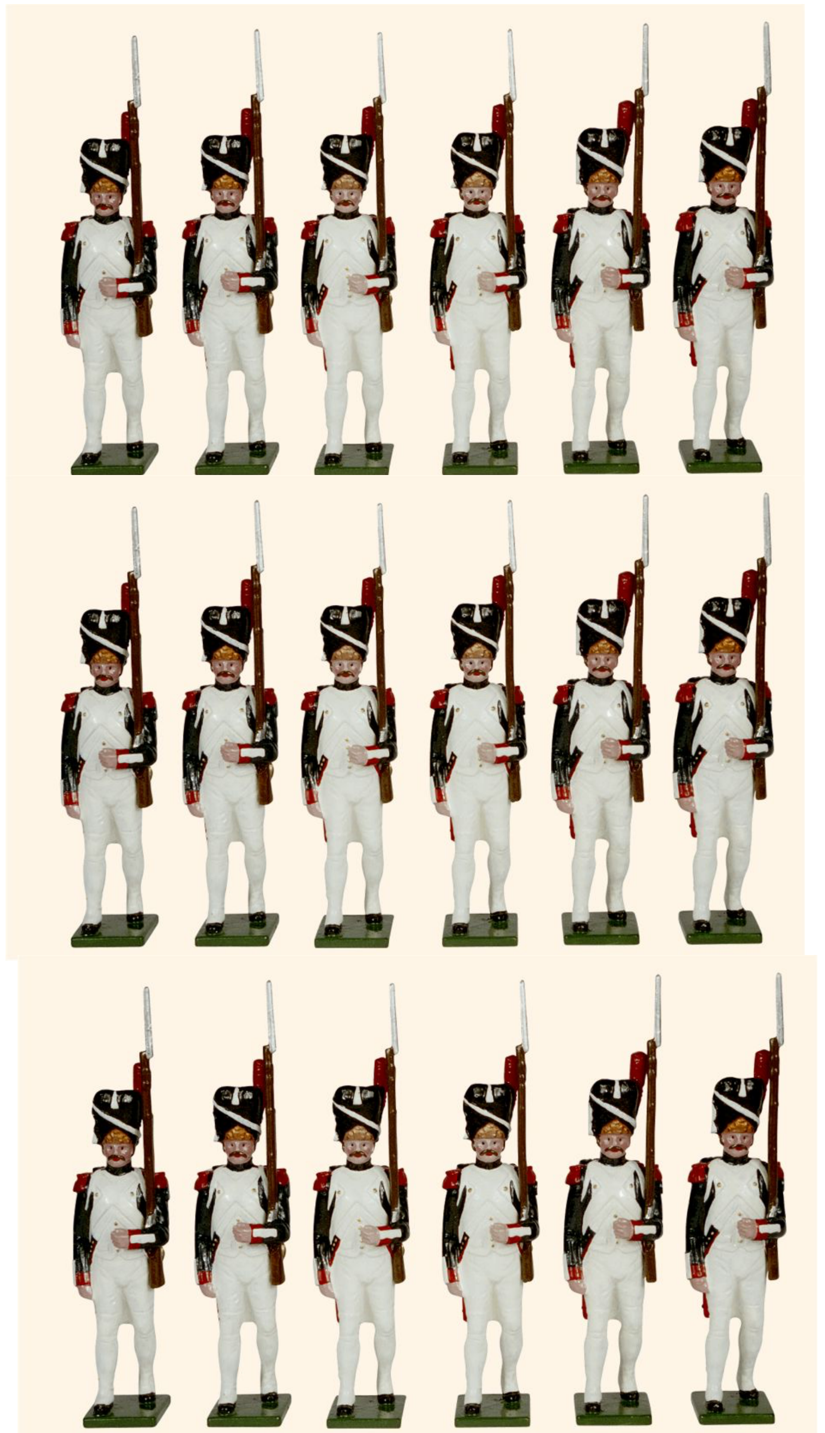
Yes (1)

(1) Session time is finite, this topic is not! Best left to another session with more time!

If more is better…

When is “more” too much?





Pick your answer!

- Replication has its limits as a means of improving service resilience
  - To avoid flooding clients typically use a serial query approach
  - Clients typically limit the number of queries and the overall elapsed time and then declare failure
- Adding more resolvers to resolv.conf, adding more name servers and adding more addresses and protocols to each service name has an initial benefit, but quickly degenerates in terms of added resilience
- How can we improve resilience?

Resilience Engineering

Replication is a coarse response to resilience - can we do this better?

Caching!

- DNS queries have a strong component of self similarity, and caching allows such queries to be answered by the caching resolver and not passed onward into the DNS name resolution infrastructure
    - “Here’s an answer I received earlier that matches your question”
- Without caching in recursive resolvers the DNS as we know it would probably collapse
- Caching is a balance between “freshness” and ”effectiveness”
    - The DNS resolution infrastructure tends to prefer longer cache times over freshness to improve resolution performance
    - Name publishers have an interest in preferring shorter cache times to allow rapid dissemination of changes to the zone contents

Caches

Who caches?

- stub resolvers
- recursive resolvers
- forwarding resolvers
- applications (typically browsers)

Who doesn’t cache?

- caching is optional

Caches

What is cached?

- resource records (not entire responses)

What should not be cached?

- resource records learned through additional sections in responses

Corner Cases:

- Responses with EDNS(0) Client Subnet (1) are cached with the Client Subnet value attached
- NS records from Child vs NS records from Parent
- resource records that fail local DNSSEC validation
  - cache them but only serve from cache if the query has the CD bit set
  - not clear what to do if the query has no EDNS(0) extensions
- DNSSEC NSEC records, which can be used to respond to a range of qnames. This exploits the property that the total name space is far larger than the occupied name space. So the "gaps" between the list of alphabetically sorted (2) names encompass a vastly greater pool of names than the occupied names, so NSEC caching stores these "gaps" to increase the leverage of caches

(1) Which should never have happened

(2) NSEC3 works too, but now we sort the hashes of names, not the names (3)

(3) NSSEC3 should never have happened

Cache TTLs

TTL determines the cache lifetime

- Well not really – TTLs **SUGGEST** a cache lifetime (in seconds) (1)
- The resolver may shorten or lengthen the cache time
- Popular caching resolvers have a “Max TTL” setting to cap TTL values
- There does not appear to be a universal minimum TTL above 0
    - And “0” is equivalent to “do not cache”
- And caches are finite-sized, so entries may be removed prior to TTL expiry under high load

(1) The resolver may (should) treat the value as a timer, but its just a suggestion, so it may play whatever sgames it wants with the TTL value!

Resilience Engineering

Replication is a coarse response to resilience - can we do this better?

Anycast

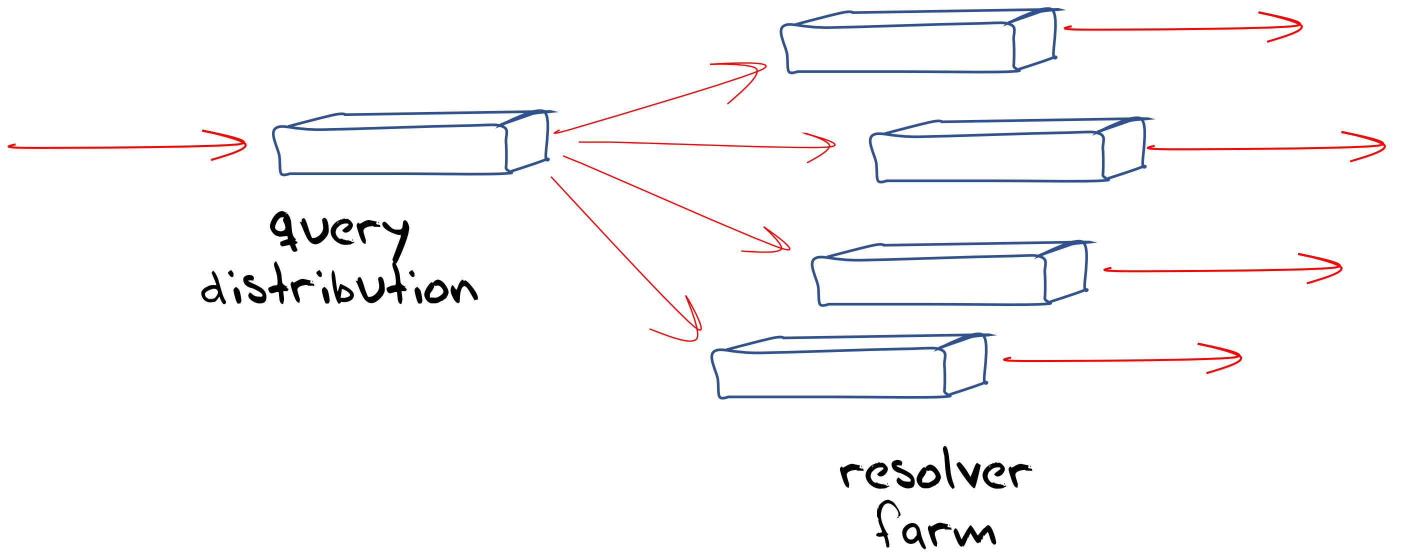
- Use the routing system to determine “closest” instance
- Service instances can be added or removed seamlessly
- Relieves the client of the overheads of serial enumeration
- Anycast Recusive Resolvers
  - Multiple recursive resolver service points that all respond to the same address
- Anycast Authoritative Servers
  - Multiple auth servers all on the same address

Resilience Engineering

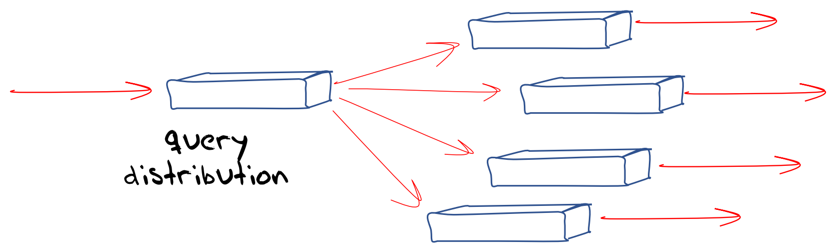
Replication is a coarse response to resilience - can we do this better?

Parallel Resolution!

- Increase resolver capacity though the use of resolver "farms"
  - Common front end / query distributor
  - Collection of resolver engines to form the back end



Resilience Engineering



Resolver "farms":

- How to distribute queries across the back end systems?
  - IP 5-tuple hash (possibly router-based)
  - IP source hash
  - DNS Qname hash
  - Hot caching (unbalanced engine load)
- Cache coherency in "farms"
  - Qname hashing is effective for positive caches, but not NSEC caches
  - IP source hashing is less efficient than 5-tuple hashing, but is coherent for caching
  - Shared caches across the resolver farm can create cache coherency irrespective of the query distribution mechanism, but are challenging to implement efficiently

On to part 3…

DNS Deep Dive, IETF 108

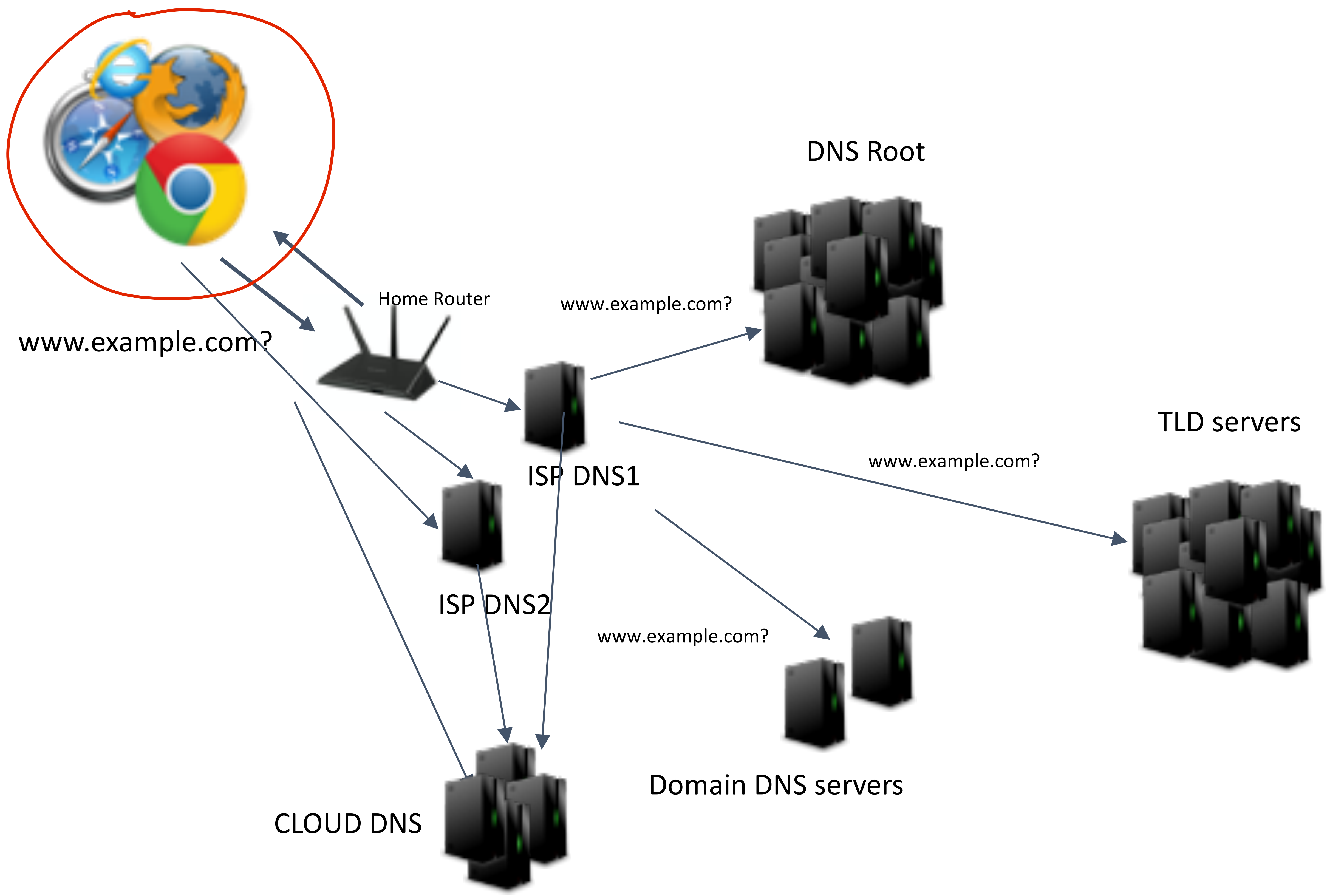
Part 1, section 3, Software

João Damas, APNIC

DNS Software

- Stub resolver - what ships with every OS, different for every OS
- Forwarder - most commonly found in home routers
- Recursive resolver software - the hardest part of DNS software
- Authoritative servers - the source of the data
- All-in-one

Stub resolvers (API)



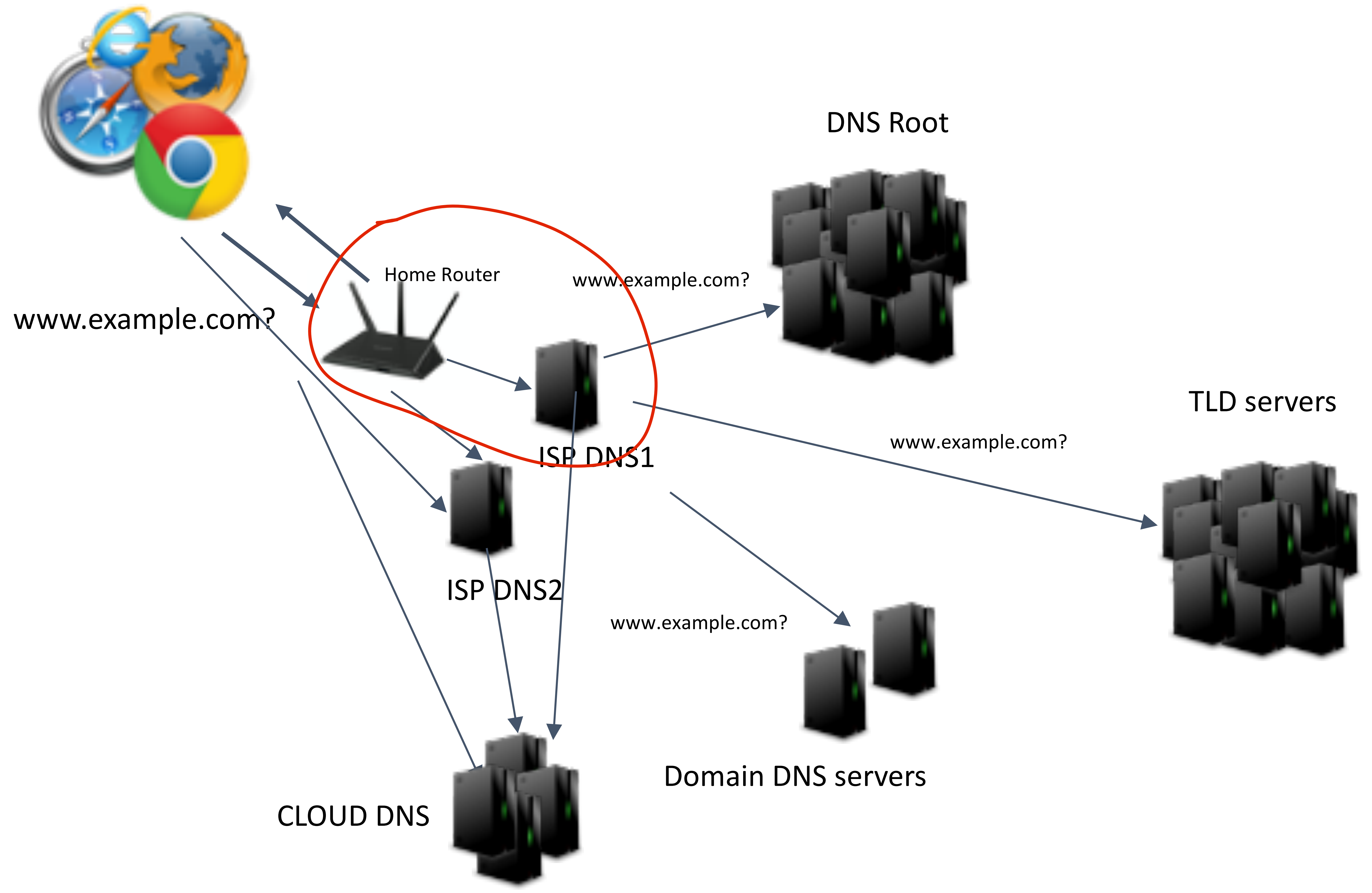
Stub resolvers (API)

- Initially just a part of the BSD Unix API
- gethostbyaddr(), gethostbyname(), gethostent(), and sethostent() functions appeared in 4.2BSD [1983, at the time of RFC 882/883]
- Still the most frequently used calls when working on Unix
- Windows has this and its own
- macOS/iOS also wrap these in more abstract calls/services
- More modern implementations like getDNS and its stub implementation, stubby, and language bindings implement client side features such as DNSSEC validation
- Linux has recently seen the introduction of systemd resolved.service
- Some apps now have their own stub (typically web browsers or “things” that use web “engines”)
- Some may include a host (or application) level cache

More on APIs

- As things evolve we may see other APIs or API-like forms for DNS transactions
    - e.g. DNS Over HTTP, is currently a way of transporting DNS over HTTPS but if you look at it with web app developer eyes, it is pretty much on the way to an API

Forwarders


DNS Root
Home Router
www.example.com?
www.example.com?
TLD servers
ISP DNS1
www.example.com?
ISP DNS2
www.example.com?
Domain DNS servers
CLOUD DNS

Forwarders

- Forwarders fall in between stubs and recursive resolvers, both from a network topology and a functionality point of view.
- They have some features typical of recursive resolvers such as a local cache but do not perform the recursive lookups against authoritative servers, they “forward” the queries they get to a full-service recursive resolver that performs the heavy duty work.

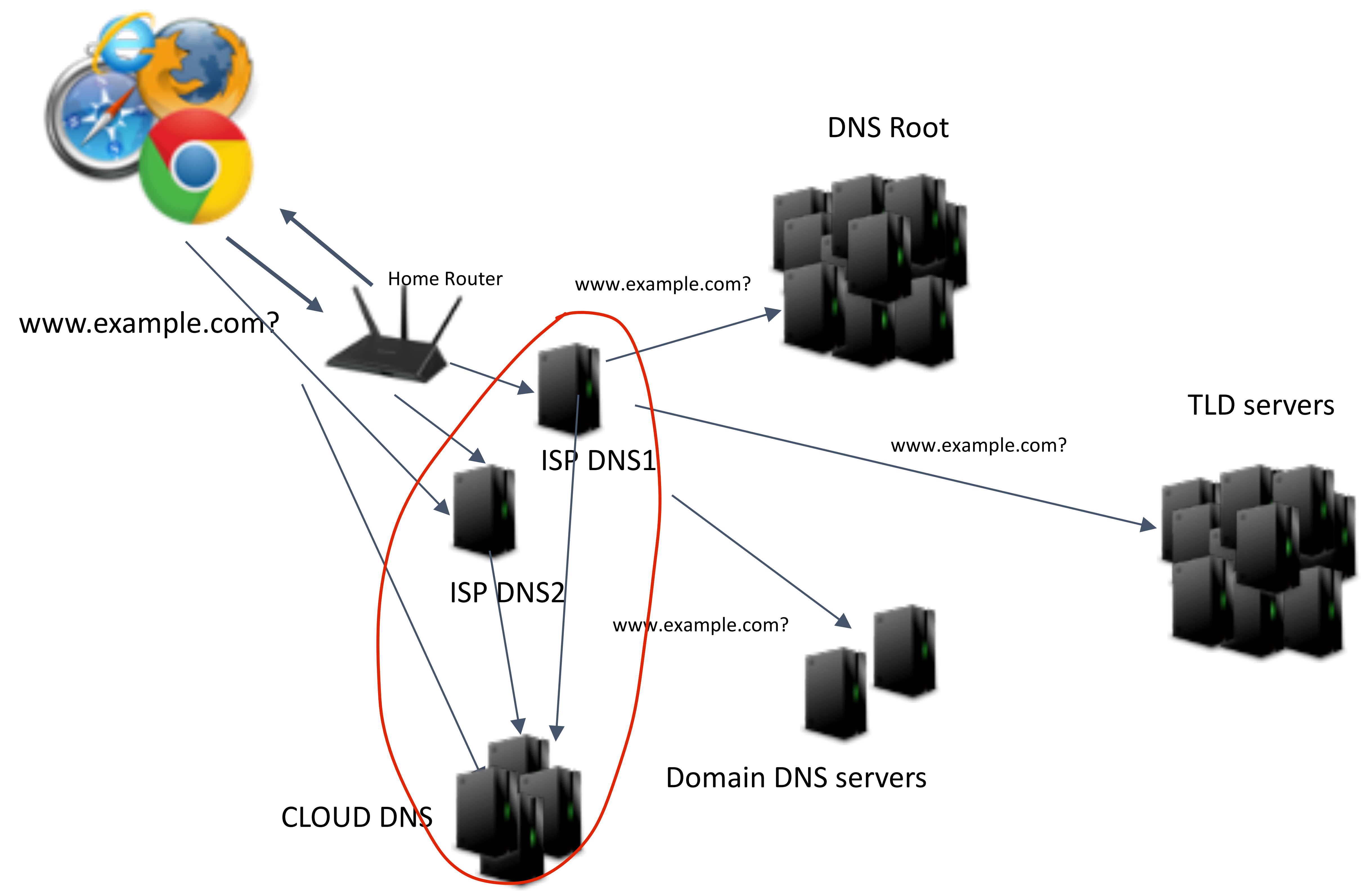
Forwarder implementations

- dnsmasq
    - by far the most common version of DNS forwarder, with various versions, of very diverse vintage, in use in the wild
    - Included as part of heaps of home router software images as it also includes a DHCP server
    - Includes added support for DNSSEC validation since v2.69 (2014)

Forwarder implementations

- BIND, MaraDNS, other recursive servers
    - Configurable as a forwarders if you wish
- These days it is common to see ISPs use full resolver being used with sizeable caches and forwarding queries to open DNS resolvers such as the OpenDNS or the quads (1,8,9)

Recursive resolvers



Recursive resolvers

- Even though today installing and running a local (to your device) recursive server is not a difficult task (at least for non-mobile devices), most people just use what is provided by the network configuration (DHCP, etc) and don’t bother with actual software deployment.
- ISPs and enterprises tend to use a mix of Open Source and proprietary software in their services.
- An increasing trend is the use of “Public Open Resolvers” where users or network administrators change their network configuration so that applications send queries to well-known servers on the Internet
  - These may be running proprietary implementations (e.g. Google, OpenDNS), or be based on DNS recursive software (e.g. Quad1, based on Knot Recursive, Quad9 based on BIND 9, etc)

Recursive resolver

- Some servers (FLOSS or not) extend DNS service to include internal or externally provided policies as part of domain blocking orders, security policies, censorship, etc
- A lot of these feeds are proprietary, at least one option has been put to the IETF (DNS RPZ)

Recursive resolver implementations

Open source

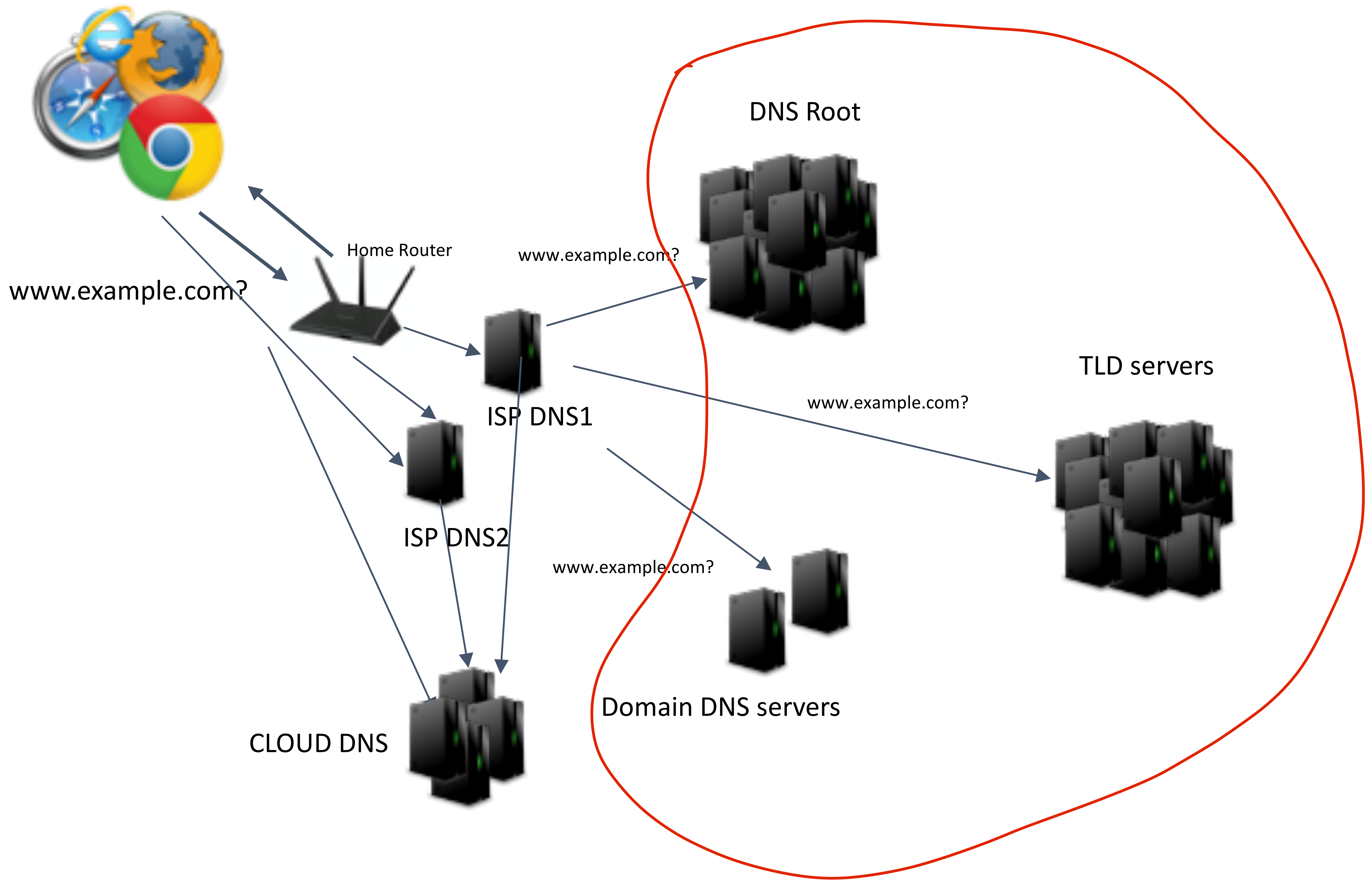
- Traditionally most DNS software has been Open Source software
    - BIND 9
    - Unbound
    - Knot resolver
    - PowerDNS Resolver

Recursive resolver implementations

Closed source

- Microsoft DNS server
- Nominum/Akamai
- Xerocole/Akamai
- Secure64
- Infoblox (mostly a front end on top of BIND, with extensions)

Authoritative servers



Authoritative servers

- In principle these are straightforward: Load zone file and answer requests
- Life is never so simple, especially in such a long-lived protocol
- Years of evolution have led to heaps of configuration options and tunable behaviour
- Split-horizon, ACLs, “Geolocation” and general DNS protocol featuritis
- Storage backends in SQL DBs, etc

Authoritative servers

Open source

- Just like for resolvers most DNS software has been Open Source software
  - BIND 9
  - NSD
  - Knot DNS
  - PowerDNS Authoritative server

Authoritative servers

Closed source

- Akamai (ex-nominum)
- Secure64
- Microsoft DNS: mostly relevant in networks where Active Directory is used as it supports the required (extended) GSS-TSIG mechanism, based on RFC 2078
- Infoblox

Additional bits and pieces

DNS “load balancers”

- Loosely defined devices or applications that aim to present a front towards the network for a series of not-directly accesible backend servers
  - Embedded in traditional load balancer devices (e.g. F5)
    - Most feel like afterthoughts (oh, #!$, we need to support DNS)
  - Open source: dnsdist (sweet!) For the DNS by the DNS
  - Routing techniques (e.g. ECMP) that rely on non-DNS software

Additional bits and pieces

DNS Interceptors

- Because DNS is your gateway to the net, it is frequently used as a control point
    - Captive portals (e.g. hotel networks)
    - Censorship
    - “optimisation”
- All of these are available as part of DNS servers or as extension modules