History of the Domain Name System

- Once upon a time...
  - Computers were very expensive
  - Computers were very large and noisy
  - Computers didn't talk to each other
- The early "internet" was very small
  - Fewer than 100 hosts

The first DNS was a distributed collection of bits of paper stuck to operators' terminals

From sticky notes to `HOSTS.TXT`

- Notes didn't scale beyond ~100 hosts
- Everybody buying computers: lots of churn
- Replaced with `HOSTS.TXT` distributed over FTP
- Worked for a while but not for long
    - The file became impractically large
    - Exponential bandwidth requirements
    - Remember, this was decades before `rsync`

The sticky notes probably worked better !

Finally: the Domain Name System

- Comparatively simple distributed system
  - ...as distributed systems go...
- Described in RFC 882 and RFC 883 in 1983

```
+-----------------------------------------------+
|                                               |
|             *****  WARNING  *****             |
|                                               |
| This RFC contains format specifications which |
| are preliminary and are included for purposes |
| of explanation only.  Do not attempt to use   |
| this information for actual implementations.  |
|                                               |
+-----------------------------------------------+
```

- Updated in RFC 1034 and RFC 1035 in 1987

The DNS protocol

- Asynchronous protocol
- Very simple packet format
- Usually stateless (UDP)
  - Lower latency
- Aggressive caching
  - Responses specify their caching objectives (time to live)
  - Servers respond to queries with additional information

Aggressive caching in DNS

- **How fresh is your data?**
- **TTL values decrement and expire**
- **Try asking for A record repeatedly:**


```
# dig www.yahoo.com
```

Query time?

TTL?

Generic distributed database of stuff

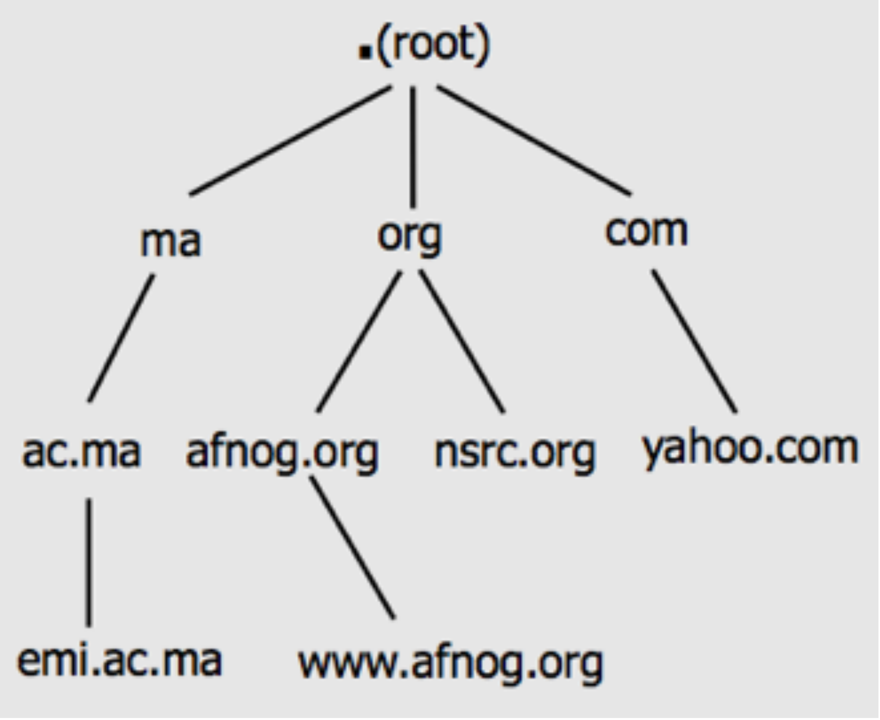
- Theoretically the DNS indexes internet resources
  - IP addresses of hosts
  - Where to send email
  - Which is the webserver
- No real limit to what can be put in the DNS
  - Geographical information (GIS)
  - Public keys (DKIM)
  - Leap seconds! (Yes, really…)

If you can imagine it, someone has put it in the DNS!

Databases have indexes (or indices)

- Data is indexed by domain names
  - A domain name is a sequence of labels
  - Labels are separated by dots (".") and form a tree
- Domain names are case insensitive ASCII
  - Internationalised domain names hacked on recently
  - "Punycode" encoding turns anything into ASCII
  - With many interesting and surprising restrictions

Hierarchical database



- DNS represented as a tree
  - Root (“.”) at the top, domain names as leaves underneath
  - Engineers are not botanists!
- Administration is shared
- Authority is delegated
- No single entity in charge

Operation of the protocol

- Servers respond to queries
- Clients **recursively** query servers
- Responses are **cached** everywhere

Recursion? Fundamental algorithm:

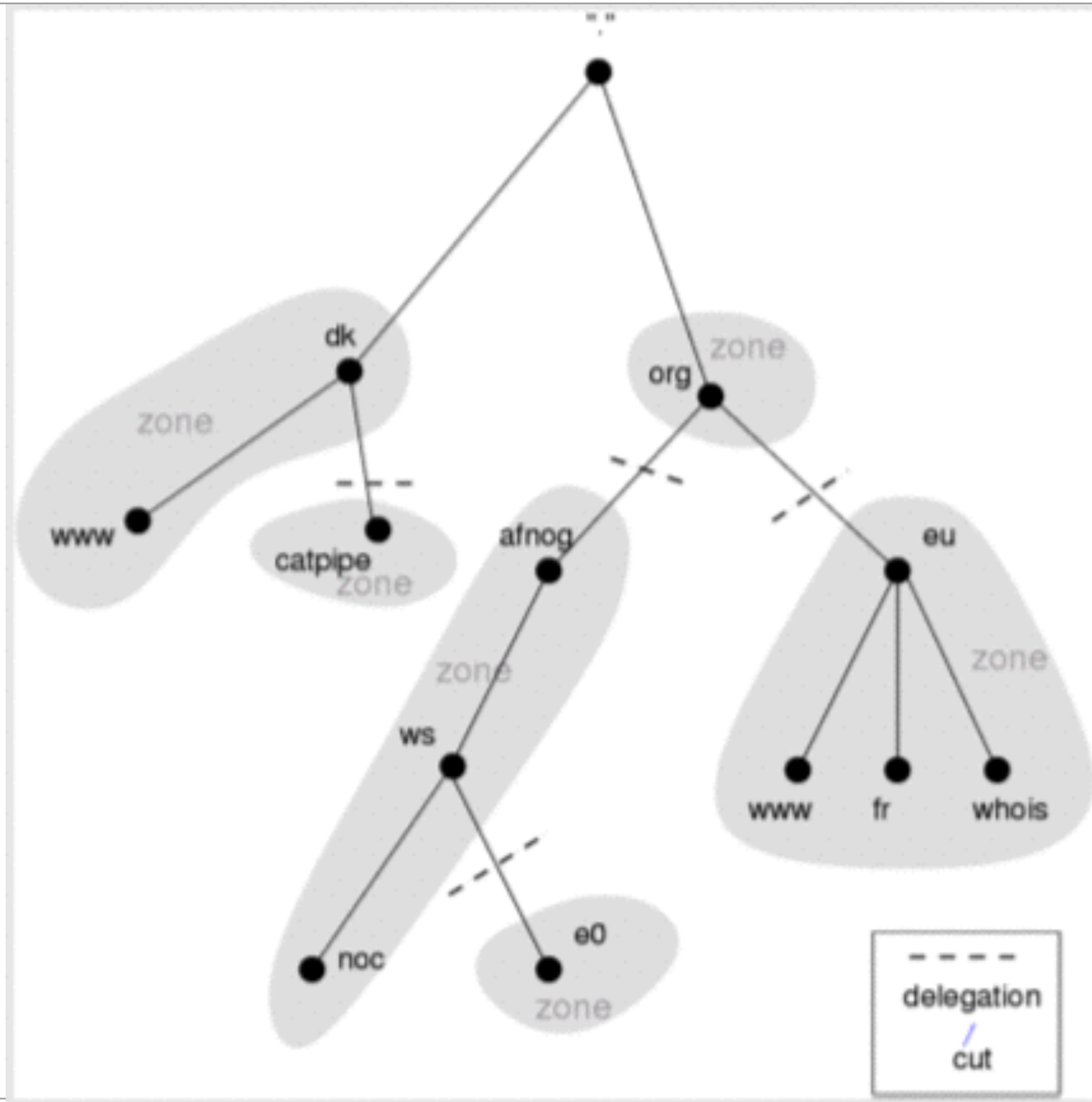
Keep asking the same question until you get a reply

or until you get bored waiting

Root and TLDs

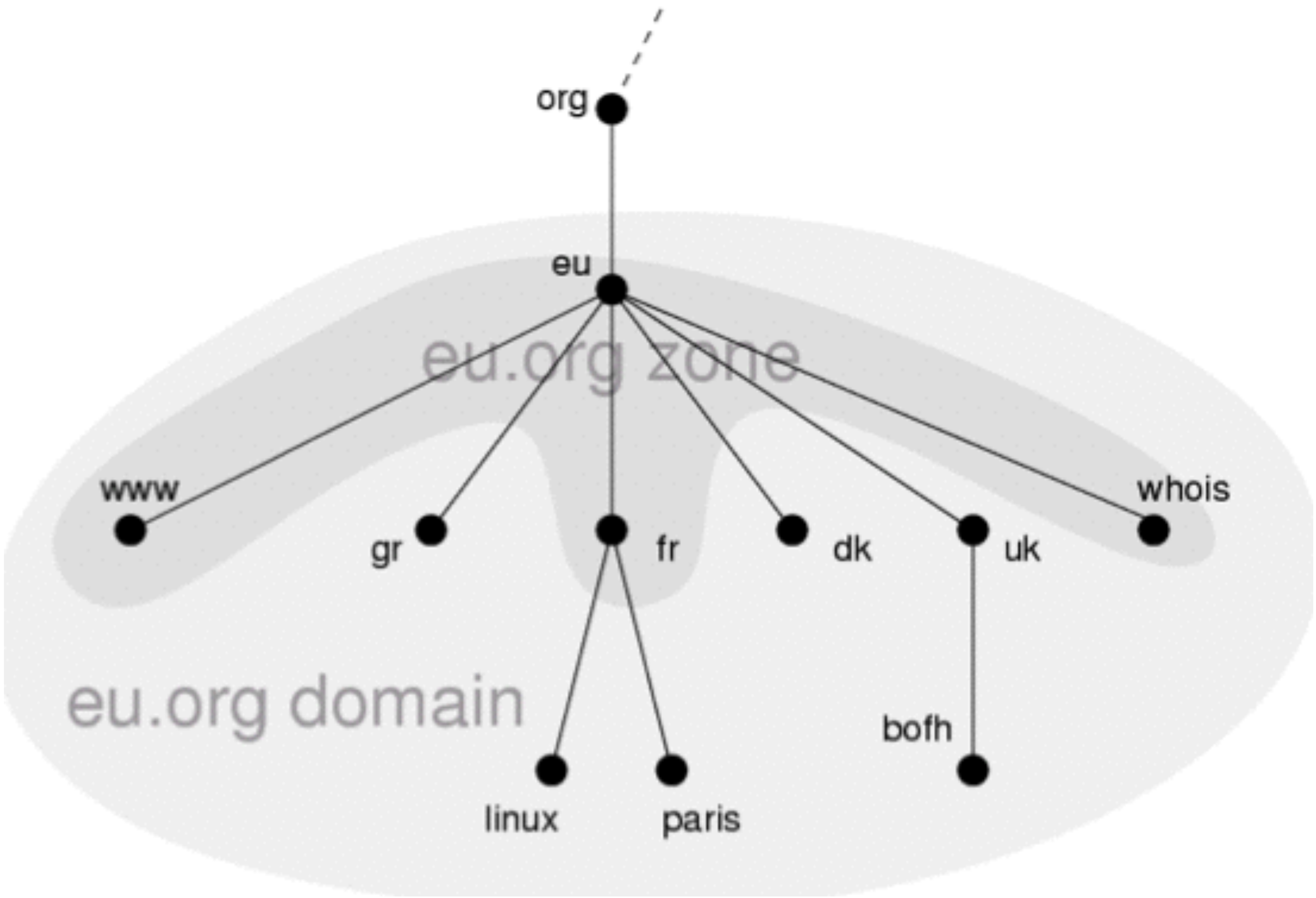
- Top to bottom approach
  - 13 root servers
  - “Empty label” covers the “.” zone
- Top level domains
  - GTLD: Generic Top-Level Domain (.com, .net, .org, etc)
  - ccTLD: Country-Code Top-Level Domain (.it, .nl, .ch, etc)
  - New TLDs (.tourism, .newyork, .museum, etc…)
  - IDN: Internationalised Domain Names (.ایران .MOCKBA)

Delegation


"."
dk
zone
www
catpipe
zone
org
zone
afnog
zone
ws
noc
e0
zone
eu
zone
www
fr
whois
delegation
/
cut

Delegation: domains and zones

- **Domain: entire subtree**
- **Zone: part of domain administered by an entity**



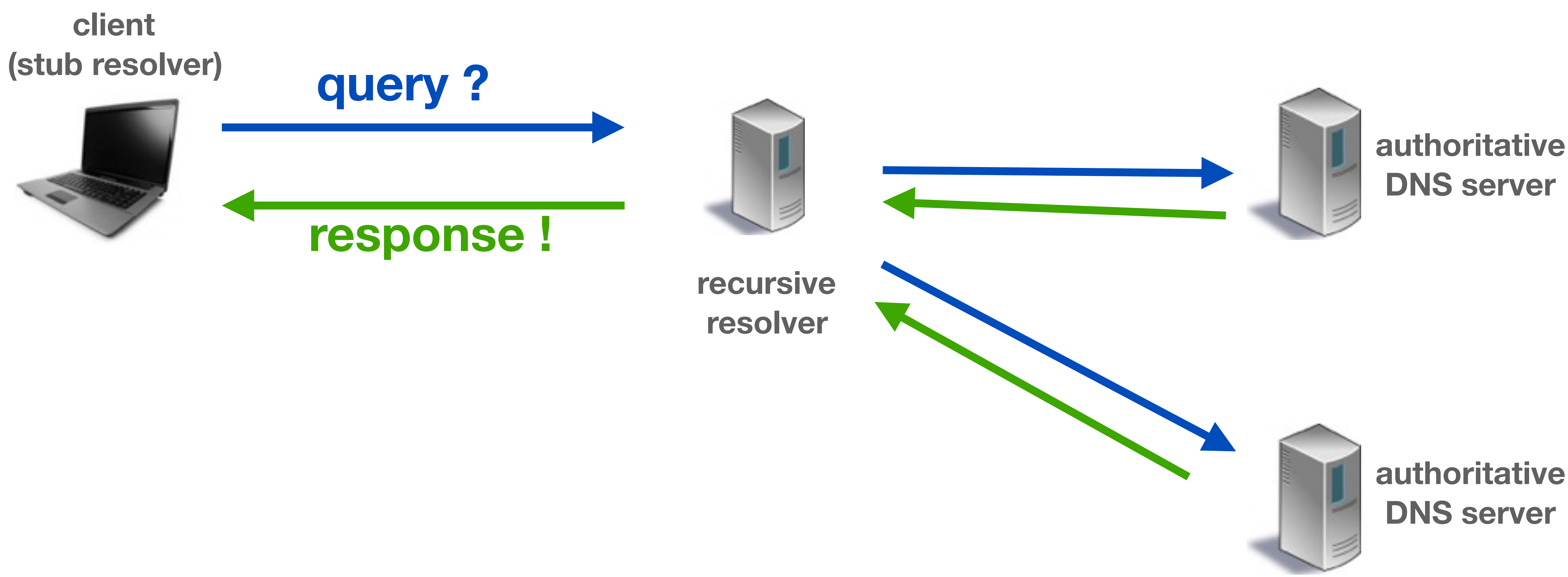
Actors in a DNS play

Clients configure a recursive resolver `/etc/resolv.conf`,

Recursive resolvers find answers on behalf of clients. They often have a large cache.

They query the DNS from the root down until they find answers.

Authoritative servers reply authoritatively to queries.



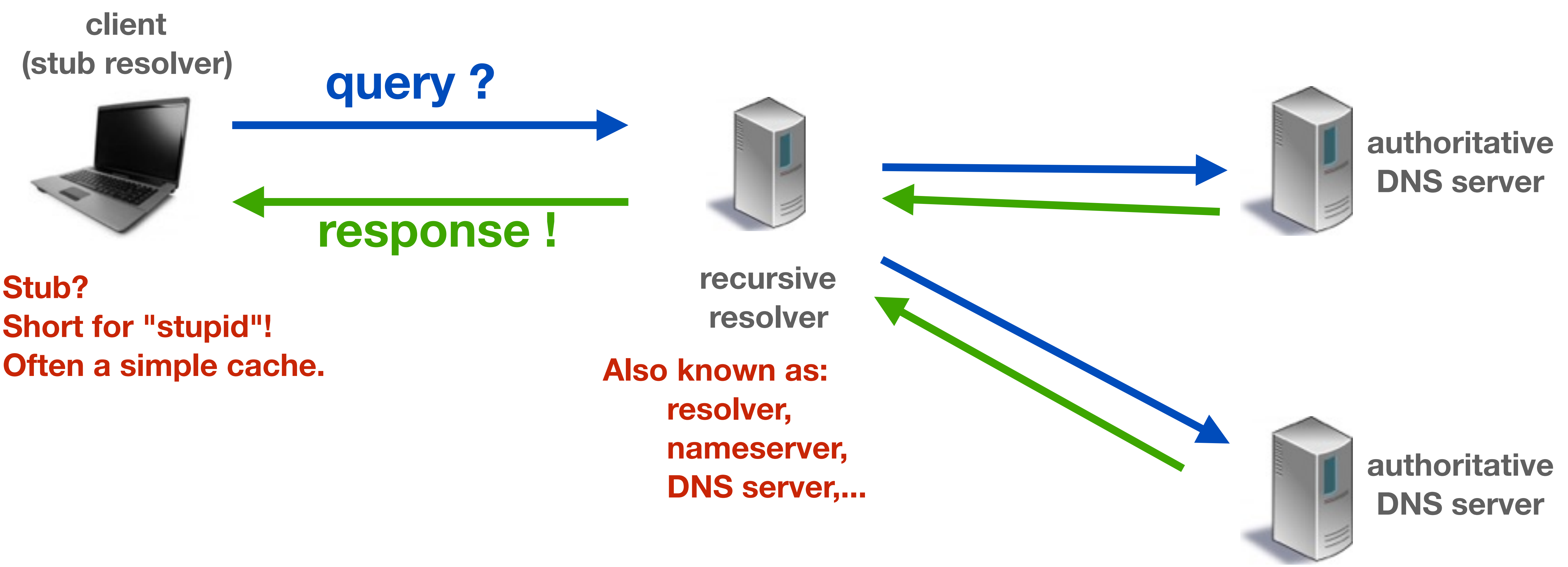
Actors in a DNS play

Clients configure a recursive resolver `/etc/resolv.conf`,

Recursive resolvers find answers on behalf of clients. They often have a large cache.

They query the DNS from the root down until they find answers.

Authoritative servers reply authoritatively to queries.

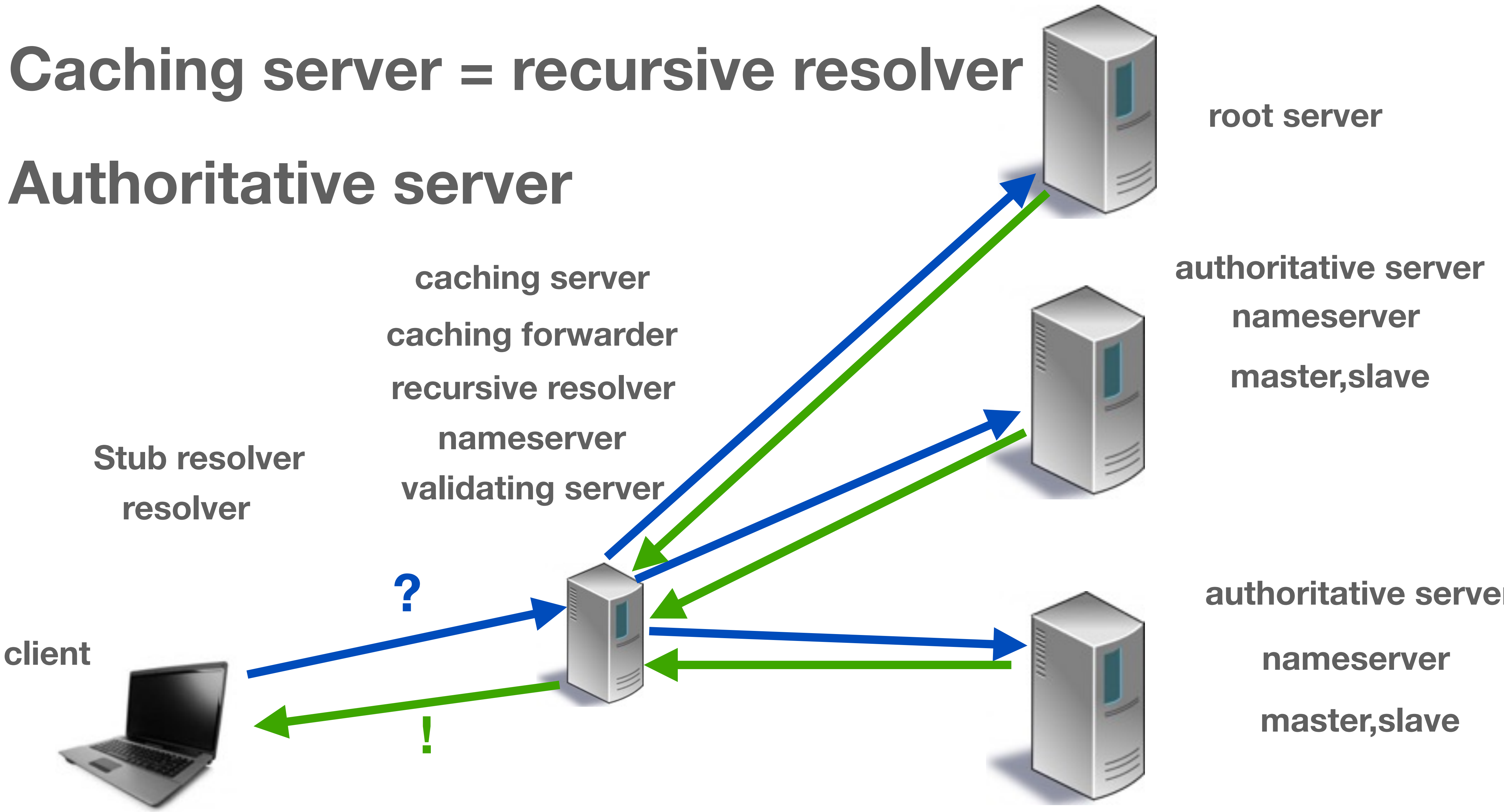


Stub?
Short for "stupid"!
Often a simple cache.

Also known as:
resolver,
nameserver,
DNS server,...

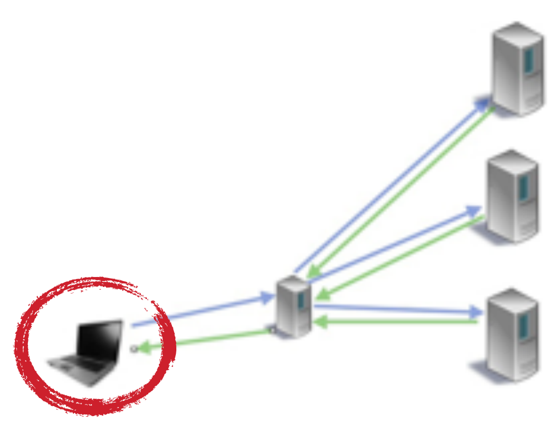
Terminology

- Stub resolver
- Caching server = recursive resolver
- Authoritative server



Actors in a DNS play — clients

- **Client** (web browser, email …) uses OS's **stub resolver** to find **recursive resolver's** IP address



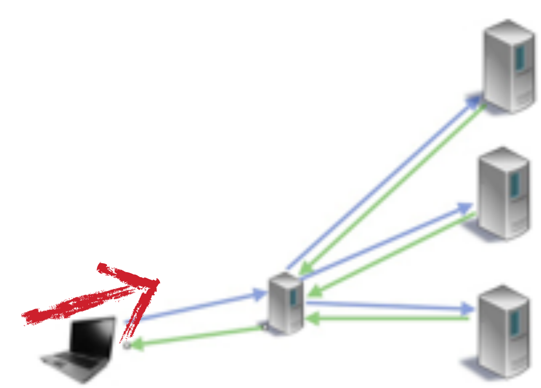
Actors in a DNS play — clients

- How does the **stub resolver** know which **recursive resolver** to query?
- UNIX: look in `/etc/resolv.conf`
  - look for:

    `nameserver a.b.c.d`

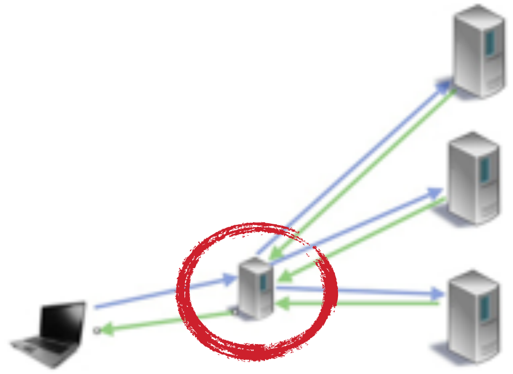
    or

    `nameserver 2001:db8:30a::53`
  - IP of DNS server



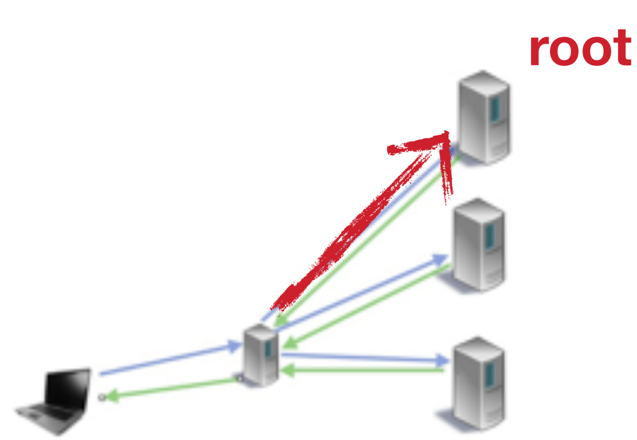
Actors in a DNS play — resolvers

- Queried by **stub resolvers** to resolve names
- They query the **authoritative servers** for the answer and serve it back
- Results are cached based on the Time To Live (TTL) in the zone
- A popular recursive resolver is 8.8.8.8



Actors in a DNS play — root servers

- List of 13 well-known root servers included with resolvers by default (compiled-in or configured)
- If you can reach one root server, you can get the latest list of root servers by querying "."
- Many resolvers choose to cache "." locally

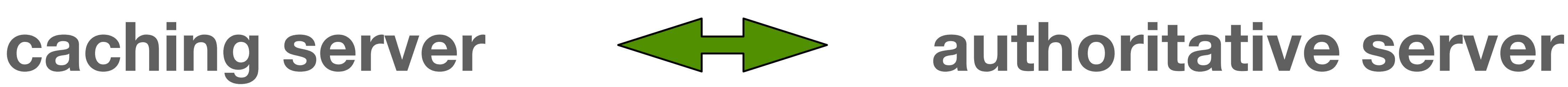


Actors in a DNS play — authoritative

- Records are in its zone file
  - Type A, AAAA, MX, CNAME, etc.
- Only answer queries for data under their authority
  - (Only if they have internal copy of the data)
- If can't answer, it points to authority
  - but doesn't query recursively.

Caching vs Authoritative

- If query repeated: query time will be lower
- Answers **cached** by recursive resolver
- TTL of answer: max time it can be cached



Caching (Recursive) Server

- Forward queries of clients
- Cache answers for later

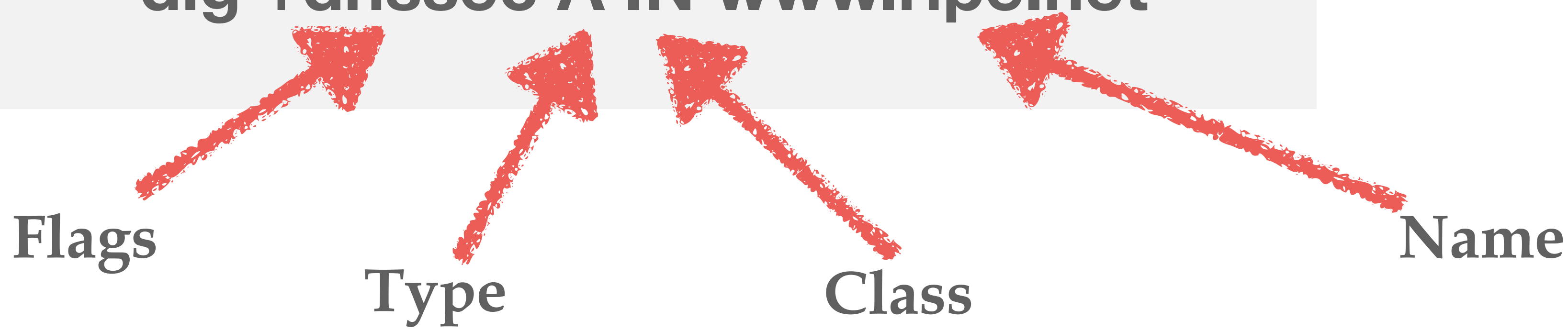
- **Caching** and **authoritative** server can be the same software.
- Better to use separate machines.

Queries, responses and flags

DNS Query

- **Every DNS Query consists of:**
    - qname: a domain name (i.e. www.ripe.net)
    - qtype: A, AAAA, MX, CNAME, PTR, SRV, TXT, NS
    - qclass: IN   (only one used today)
    - Flags: QR, RD, EDNS Opt, DO, AD, etc.

```
dig +dnssec A IN www.ripe.net
```



The most common DNS queries

- Look up a host's address by its name

  "Forward DNS"

```
trouble.is has address 88.198.44.60
trouble.is has IPv6 address 2a01:4f8:a0:10e6::1:1
```

- Look up a host's name by its address

  "Reverse DNS"

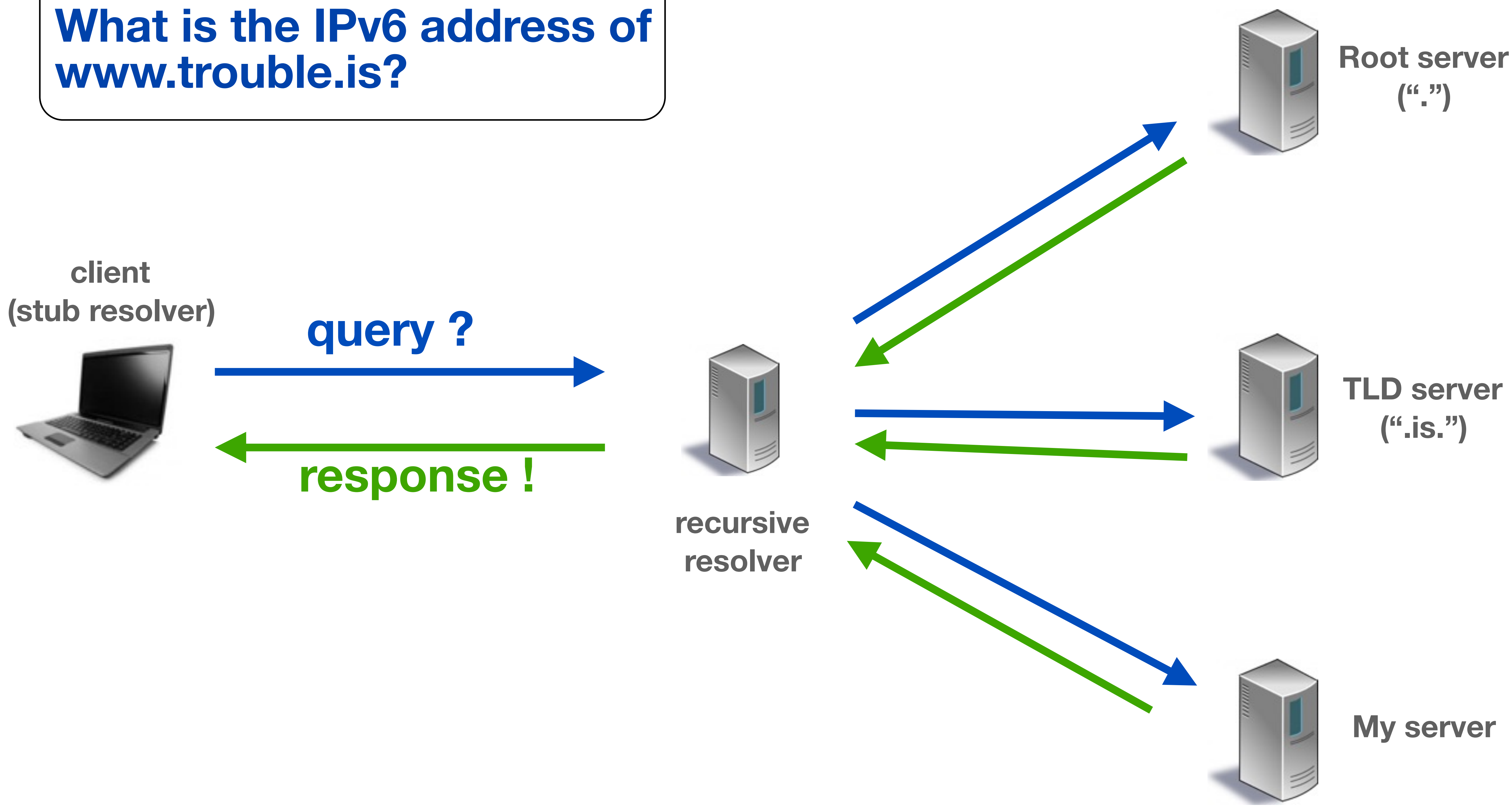
```
60.44.198.88.in-addr.arpa domain name pointer rincewind.trouble.is.
1.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.6.e.0.1.0.a.0.0.8.f.4.0.1.0.a.2.ip6.arpa
    domain name pointer rincewind.trouble.is.
```

Flags

- `qr` query response A
- `rd` recursion desired A Q
- `ra` recursion available A
- `aa` authoritative answer A

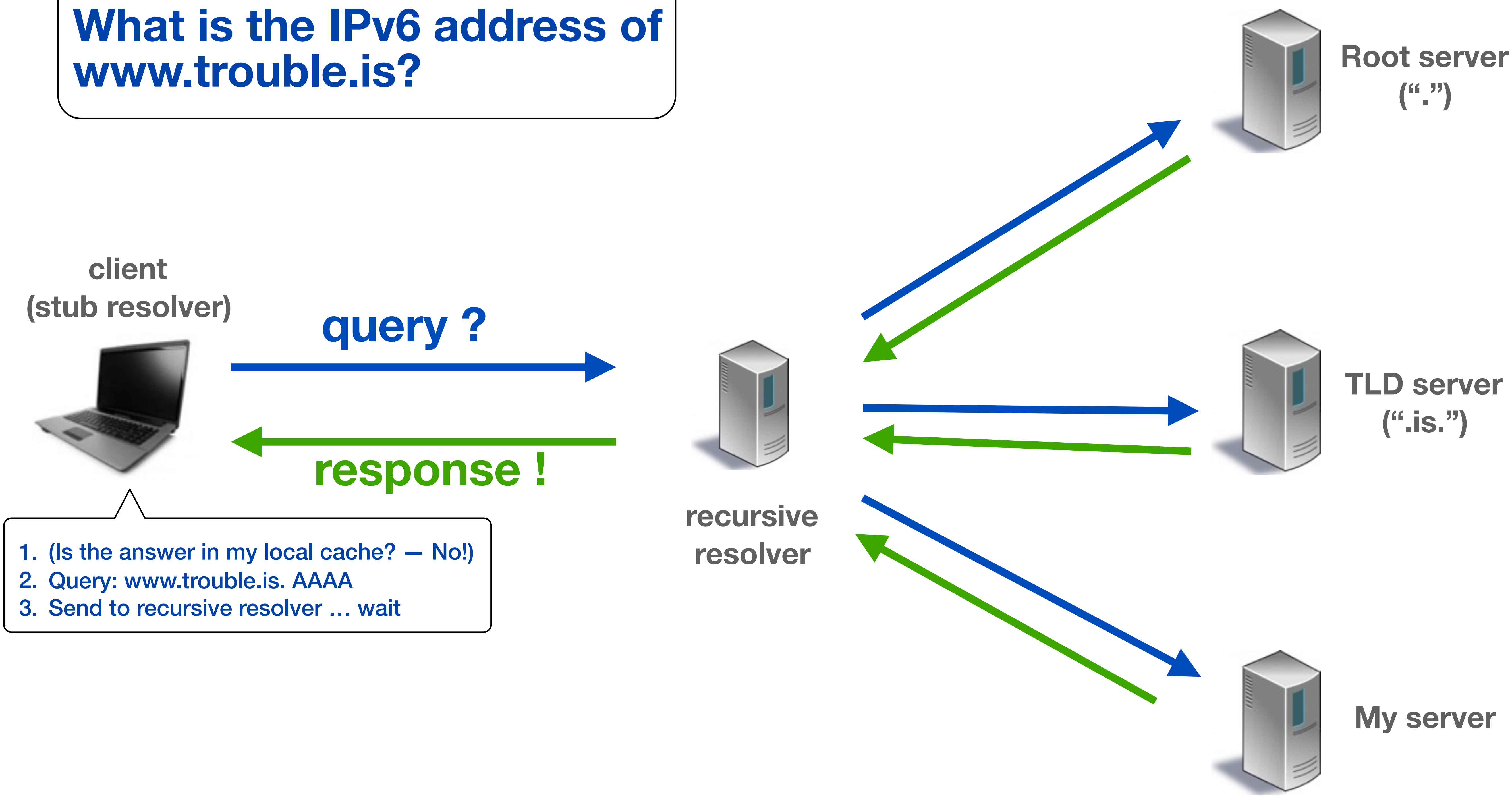
Example DNS query

What is the IPv6 address of www.trouble.is?



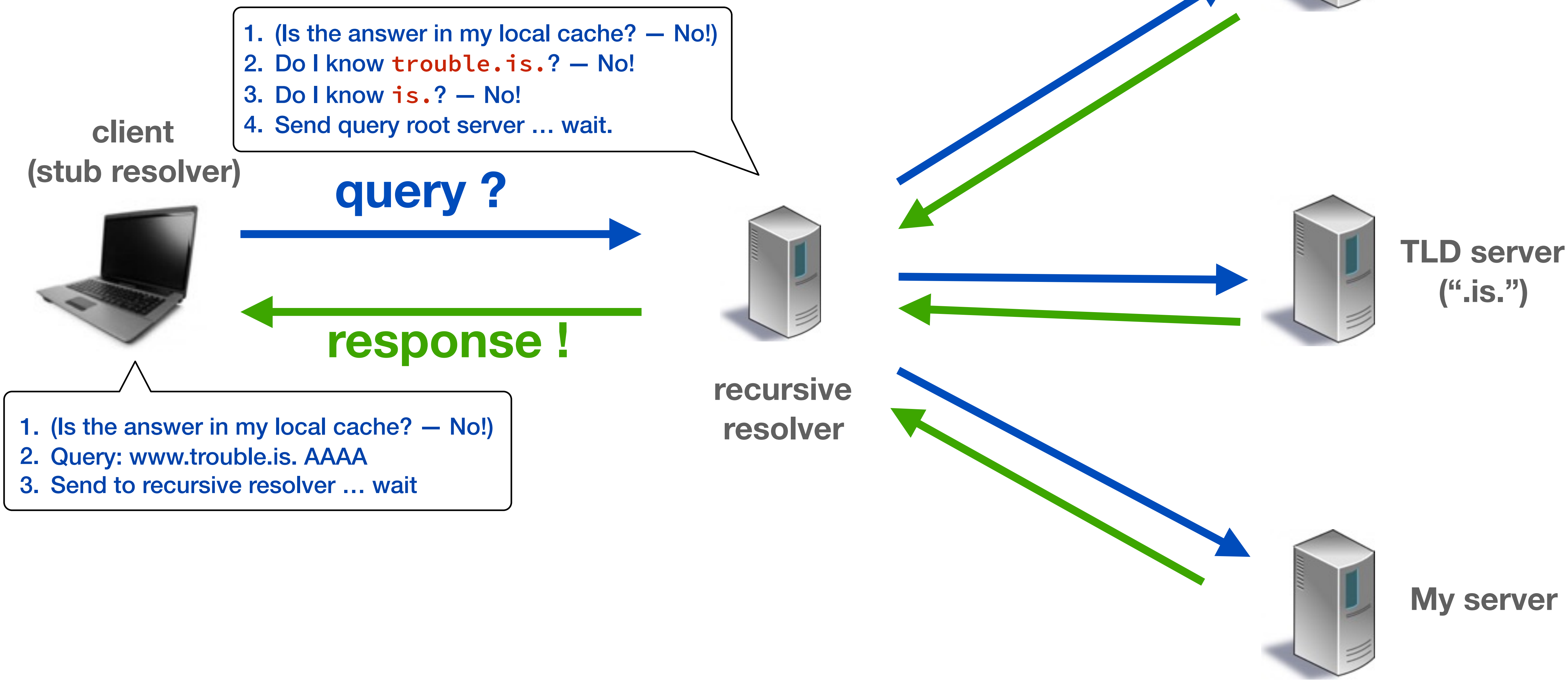
Example DNS query

What is the IPv6 address of www.trouble.is?

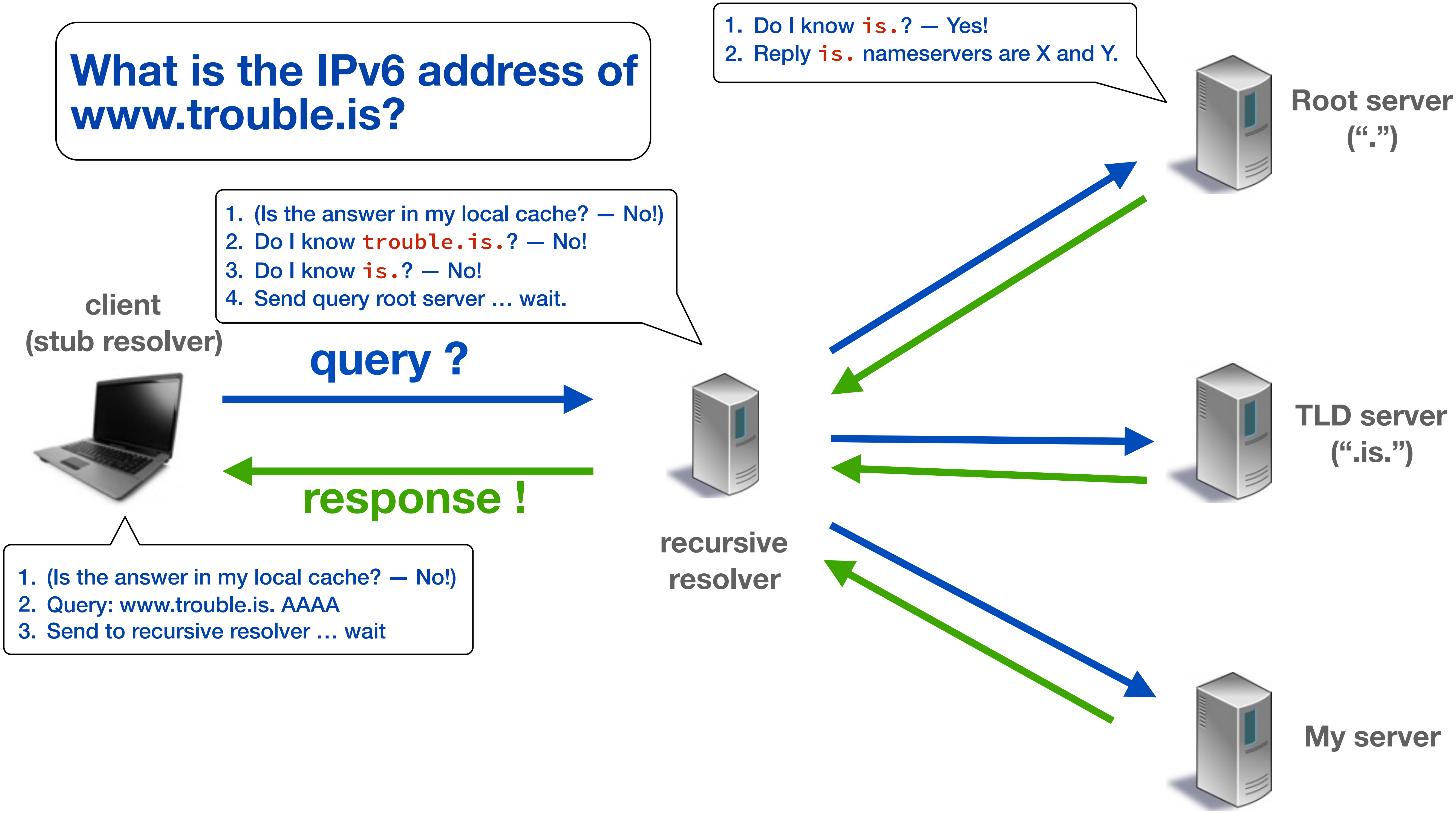


Example DNS query

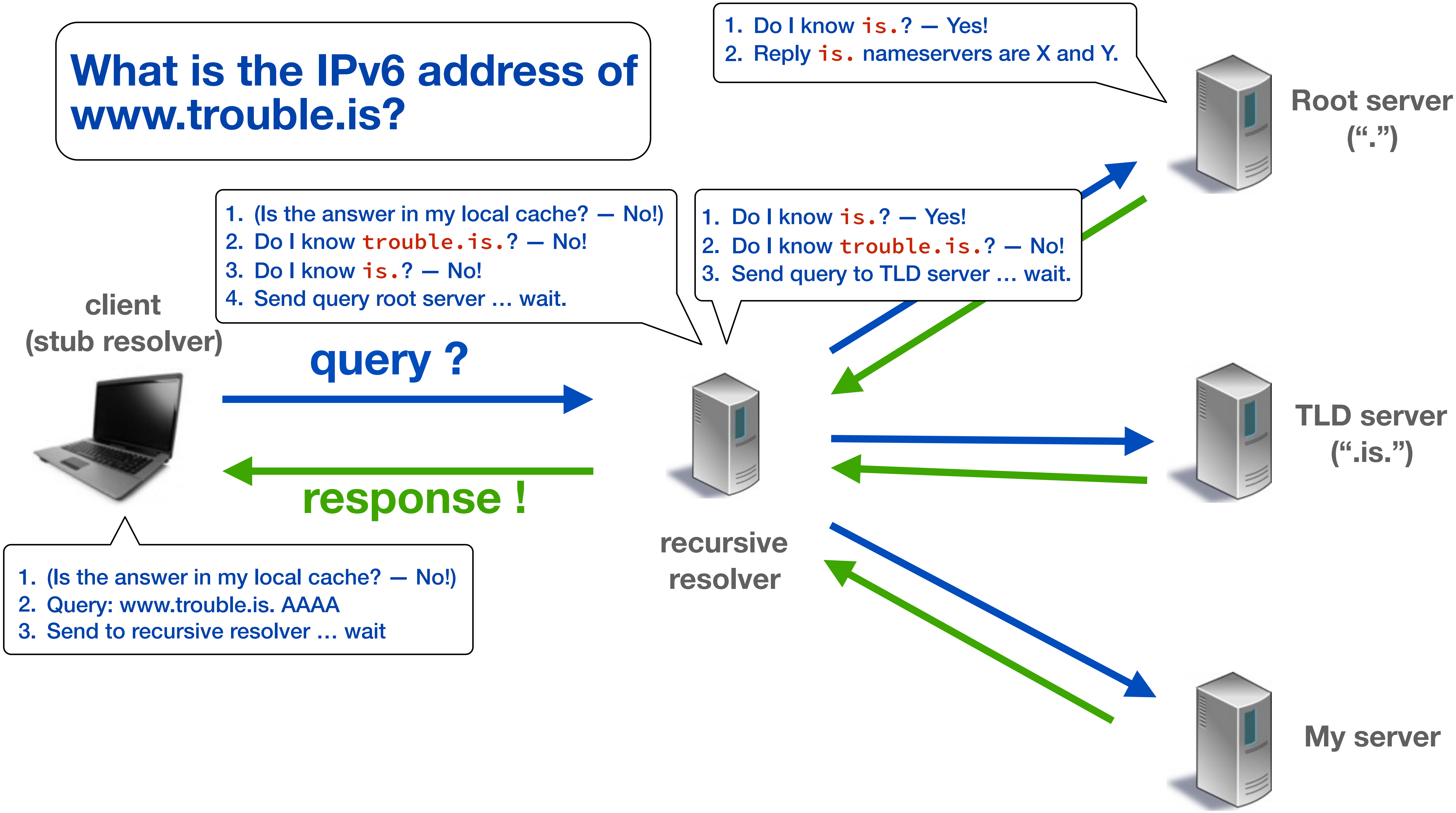
What is the IPv6 address of www.trouble.is?



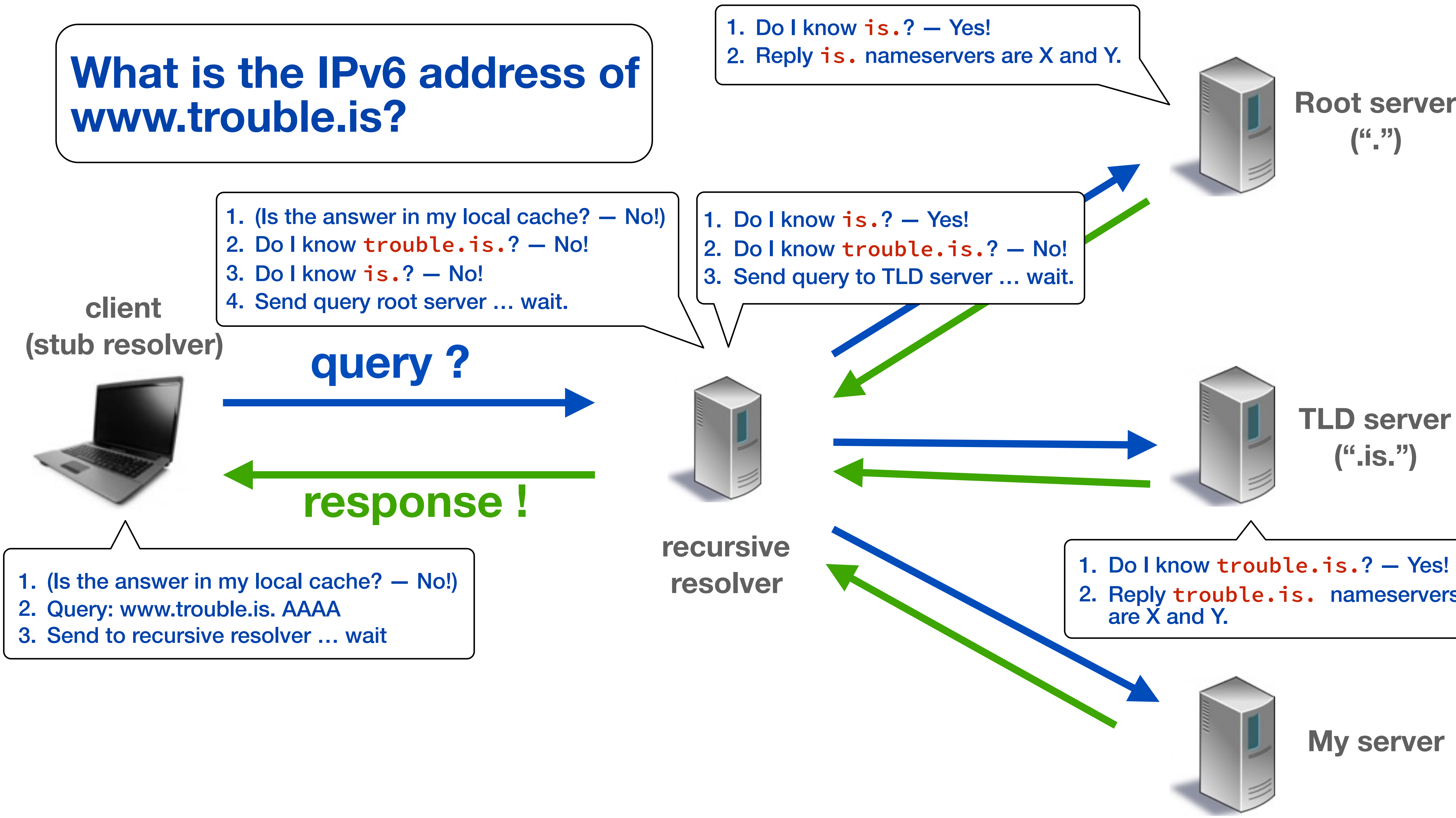
Example DNS query



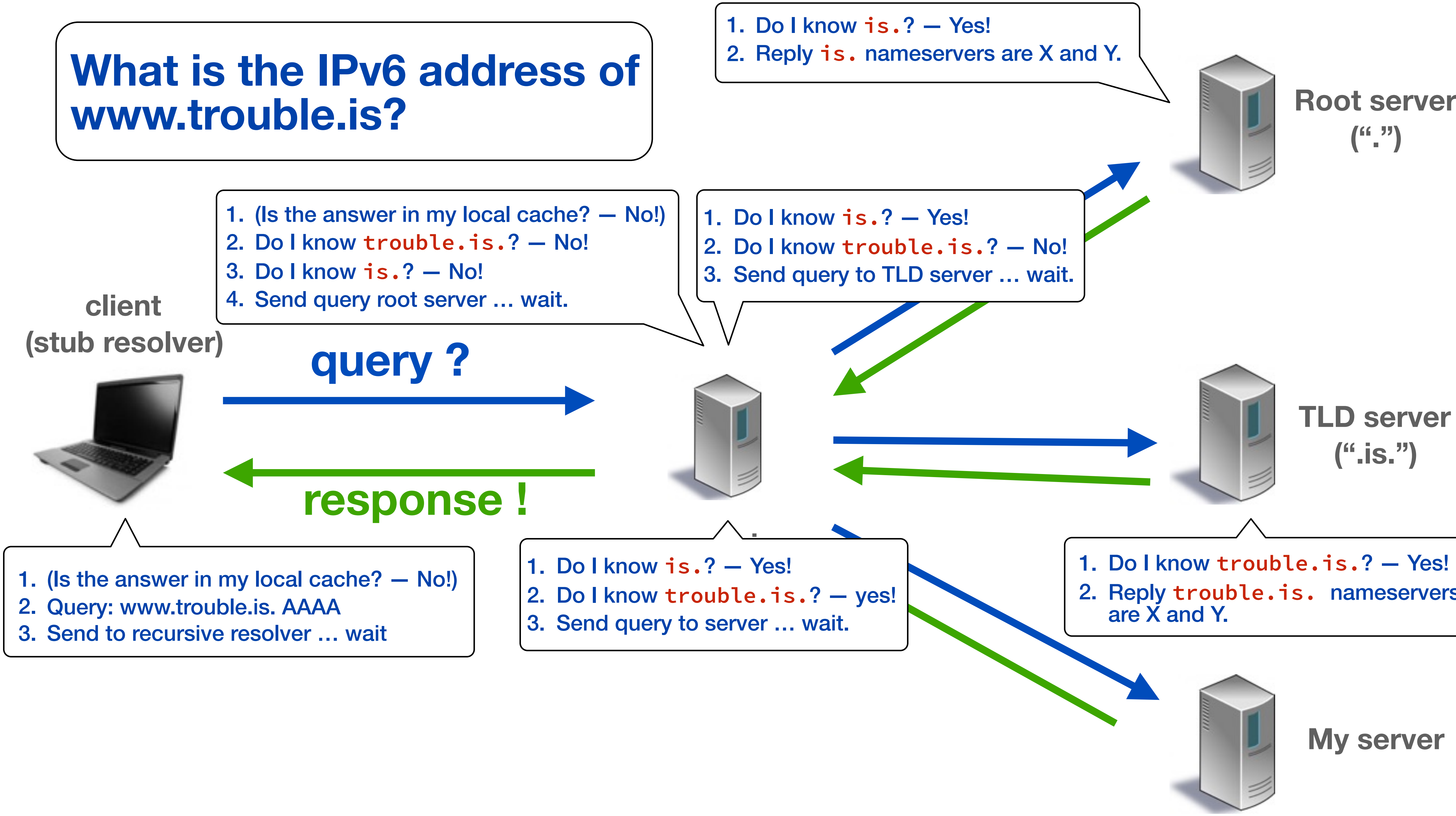
Example DNS query



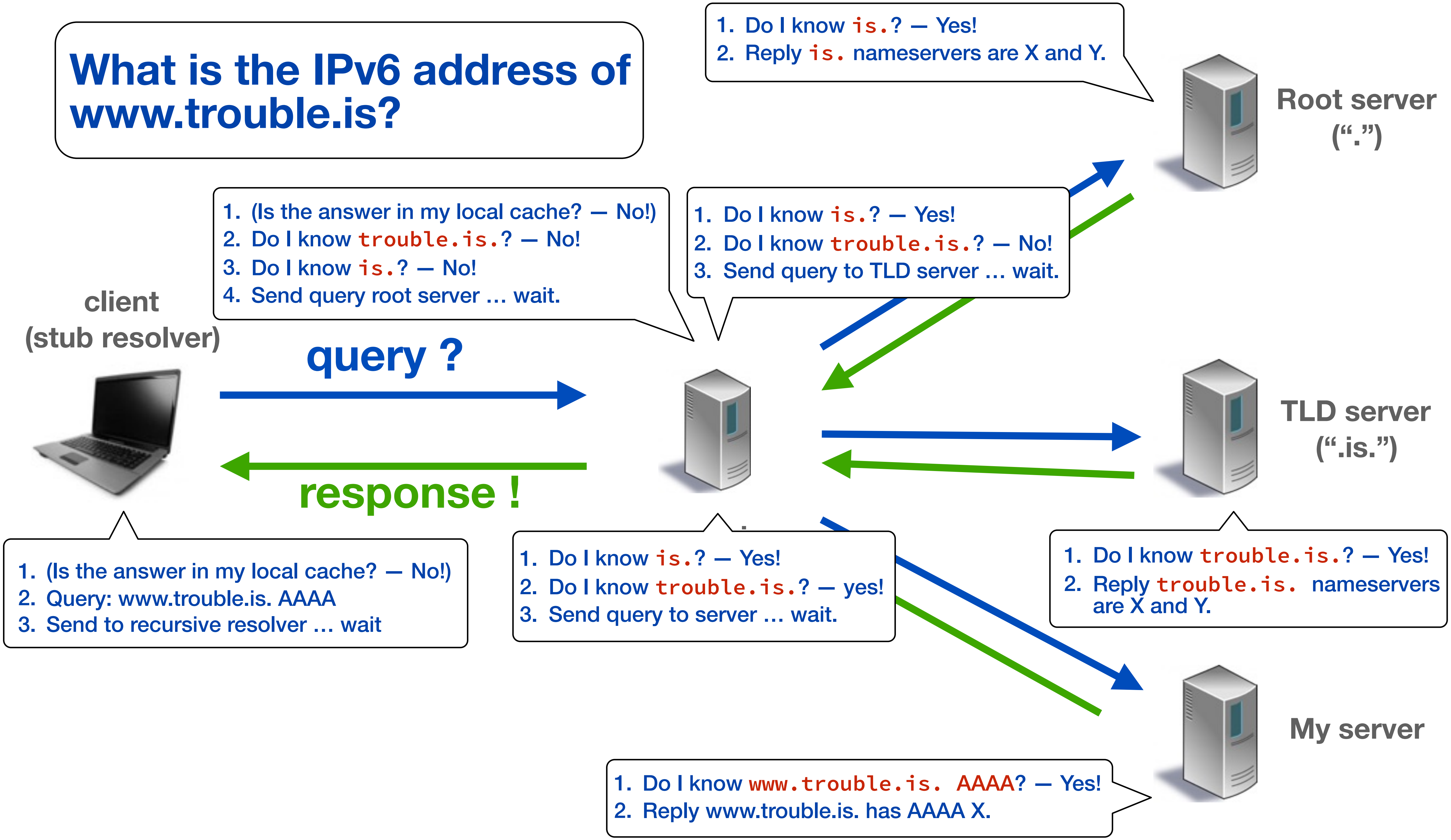
Example DNS query



Example DNS query



Example DNS query



“dig” to Get More Details


```
ns# dig @147.28.0.39 www.nsrc.org. a

; <<>> DiG 9.3.2 <<>> @147.28.0.39 www.nsrc.org
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4620
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 4,
ADDITIONAL: 2

;; QUESTION SECTION:
;www.nsrc.org.                  IN      A

;; ANSWER SECTION:
www.nsrc.org.           14400   IN      A       128.223.162.29

;; AUTHORITY SECTION:
nsrc.org.               14400   IN      NS      rip.psg.com.
nsrc.org.               14400   IN      NS      arizona.edu.

;; ADDITIONAL SECTION:
rip.psg.com.            77044   IN      A       147.28.0.39
arizona.edu.             2301   IN      A       128.196.128.233

;; Query time: 708 msec
;; SERVER: 147.28.0.39#53(147.28.0.39)
;; WHEN: Wed May 10 15:05:55 2007
;; MSG SIZE  rcvd: 128
```

“dig” to Get More Details


```
ns# dig @147.28.0.39 www.nsrc.org. a

; <<>> DiG 9.3.2 <<>> @147.28.0.39 www.nsrc.org
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4620
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 4,
ADDITIONAL: 2

;; QUESTION SECTION:
;www.nsrc.org.                  IN      A

;; ANSWER SECTION:
www.nsrc.org.           14400   IN      A       128.223.162.29

;; AUTHORITY SECTION:
nsrc.org.               14400   IN      NS      rip.psg.com.
nsrc.org.               14400   IN      NS      arizona.edu.

;; ADDITIONAL SECTION:
rip.psg.com.            77044   IN      A       147.28.0.39
arizona.edu.             2301   IN      A       128.196.128.233

;; Query time: 708 msec
;; SERVER: 147.28.0.39#53(147.28.0.39)
;; WHEN: Wed May 10 15:05:55 2007
;; MSG SIZE  rcvd: 128
```

glue

“dig” to Get More Details

@ recursive

```
ns# dig @147.28.0.39 www.nsrc.org. a

; <<>> DiG 9.3.2 <<>> @147.28.0.39 www.nsrc.org
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4620
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 4,
ADDITIONAL: 2

;; QUESTION SECTION:
;www.nsrc.org.                  IN      A

;; ANSWER SECTION:
www.nsrc.org.           14400   IN      A       128.223.162.29

;; AUTHORITY SECTION:
nsrc.org.               14400   IN      NS      rip.psg.com.
nsrc.org.               14400   IN      NS      arizona.edu.

;; ADDITIONAL SECTION:
rip.psg.com.            77044   IN      A       147.28.0.39
arizona.edu.             2301   IN      A       128.196.128.233

;; Query time: 708 msec
;; SERVER: 147.28.0.39#53(147.28.0.39)
;; WHEN: Wed May 10 15:05:55 2007
;; MSG SIZE  rcvd: 128
```

glue

“dig” to Get More Details (cont.)


```
noc# dig www.afrinic.net any

; <<>> DiG 9.4.2 <<>> any www.afrinic.net
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36019
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 6, ADDITIONAL: 10

;; QUESTION SECTION:
;www.afrinic.net.          IN   ANY

;; ANSWER SECTION:
www.afrinic.net. 477       IN   AAAA     2001:42d0::200:80:1
www.afrinic.net. 65423     IN   A        196.216.2.1

;; AUTHORITY SECTION:
afrinic.net.     65324     IN   NS       sec1.apnic.net.
afrinic.net.     65324     IN   NS       sec3.apnic.net.
afrinic.net.     65324     IN   NS       ns1.afrinic.net.
afrinic.net.     65324     IN   NS       tinnie.arin.net.
afrinic.net.     65324     IN   NS       ns.lacnic.net.
afrinic.net.     65324     IN   NS       ns-sec.ripe.net.

;; ADDITIONAL SECTION:
ns.lacnic.net.   151715    IN   A        200.160.0.7
ns.lacnic.net.   65315     IN   AAAA     2001:12ff::7
ns-sec.ripe.net. 136865    IN   A        193.0.0.196
ns-sec.ripe.net. 136865    IN   AAAA     2001:610:240:0:53::4
ns1.afrinic.net. 65315     IN   A        196.216.2.1
tinnie.arin.net. 151715    IN   A        168.143.101.18
sec1.apnic.net.  151715    IN   A        202.12.29.59
sec1.apnic.net.  151715    IN   AAAA     2001:dc0:2001:a:4608::59
sec3.apnic.net.  151715    IN   A        202.12.28.140
sec3.apnic.net.  151715    IN   AAAA     2001:dc0:1:0:4777::140

;; Query time: 1 msec
;; SERVER: 196.200.218.1#53(196.200.218.1)
;; WHEN: Tue May 27 08:48:13 2008
;; MSG SIZE  rcvd: 423
```

“dig” to Get More Details (cont.)


```
noc# dig www.afrinic.net any

; <<>> DiG 9.4.2 <<>> any www.afrinic.net
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36019
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 6, ADDITIONAL: 10

;; QUESTION SECTION:
;www.afrinic.net.          IN   ANY

;; ANSWER SECTION:
www.afrinic.net. 477       IN   AAAA     2001:42d0::200:80:1
www.afrinic.net. 65423     IN   A        196.216.2.1

;; AUTHORITY SECTION:
afrinic.net.     65324     IN   NS       sec1.apnic.net.
afrinic.net.     65324     IN   NS       sec3.apnic.net.
afrinic.net.     65324     IN   NS       ns1.afrinic.net.
afrinic.net.     65324     IN   NS       tinnie.arin.net.
afrinic.net.     65324     IN   NS       ns.lacnic.net.
afrinic.net.     65324     IN   NS       ns-sec.ripe.net.

;; ADDITIONAL SECTION:
ns.lacnic.net.   151715    IN   A        200.160.0.7
ns.lacnic.net.   65315     IN   AAAA     2001:12ff::7
ns-sec.ripe.net. 136865    IN   A        193.0.0.196
ns-sec.ripe.net. 136865    IN   AAAA     2001:610:240:0:53::4
ns1.afrinic.net. 65315     IN   A        196.216.2.1
tinnie.arin.net. 151715    IN   A        168.143.101.18
sec1.apnic.net.  151715    IN   A        202.12.29.59
sec1.apnic.net.  151715    IN   AAAA     2001:dc0:2001:a:4608::59
sec3.apnic.net.  151715    IN   A        202.12.28.140
sec3.apnic.net.  151715    IN   AAAA     2001:dc0:1:0:4777::140

;; Query time: 1 msec
;; SERVER: 196.200.218.1#53(196.200.218.1)
;; WHEN: Tue May 27 08:48:13 2008
;; MSG SIZE  rcvd: 423
```

glue

Record Types

- NS: NameServer
- A, AAAA: IPv4, IPv6 address
- CNAME: Canonical name (alias
- MX: Mail Exchanger
- PTR Reverse info (IP to host)
- SRV: Service (host + port nr)
- SOA: Start of Authority

NS Record

- Name Server record
- Delegates a DNS subtree from parent
  - Lists the authoritative servers for the zone
- Appears in both parent and child zones
- rdata contains hostname of the DNS server

```
ripe.net. 1093 IN NS pri.authdns.ripe.net.
```

IPv4 address

octets

193.0.24.3

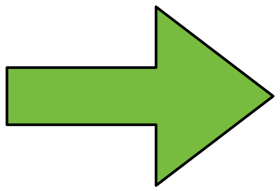
The numbers in every octet can go from **0** to **255**

255 is the maximum possible number in any octet!

IPv4 address

- 32 bit binary numbers
- 2^{32} different combinations =more than $4x10^9$
- divided into 4 octets
- each octet (8 bits) converted to decimal number for human readability

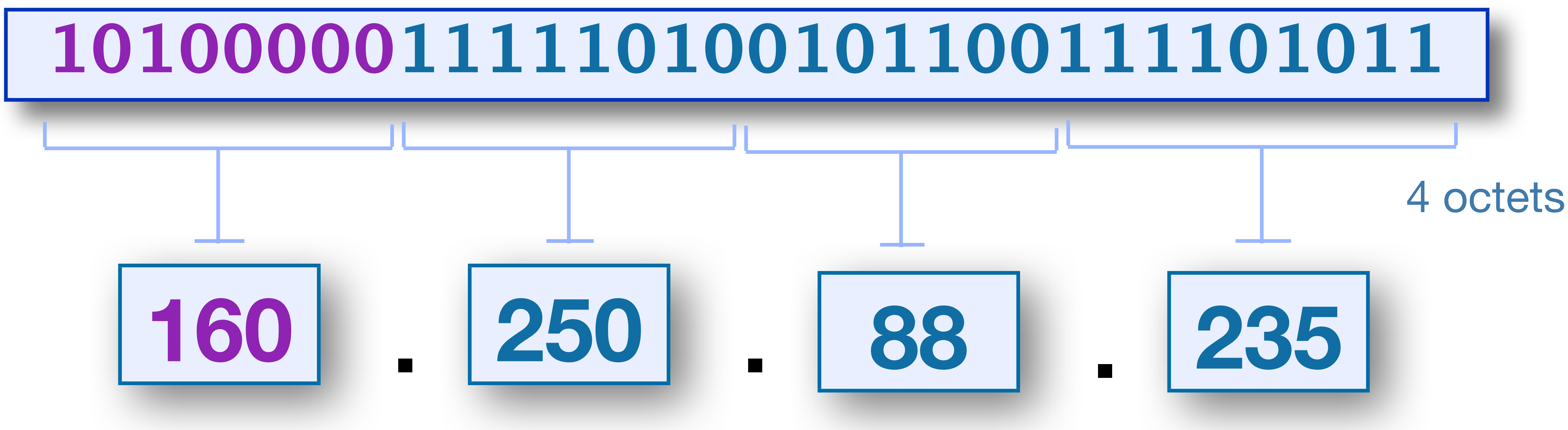
Replacing



Binary	Decimal
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
0000 0100	4
0000 0101	5
0000 0110	6
0000 0111	7
. . .	
1111 1000	248
1111 1001	249
1111 1010	250
1111 1011	251
1111 1100	252
1111 1101	253
1111 1110	254
1111 1111	255

An IPv4 Address

32 bits binary



160.250.88.235 decimal

A Record

- IPv4 Address Record
- rdata contains an IPv4 address

```
www.ripe.net. IN A 193.0.6.139
```

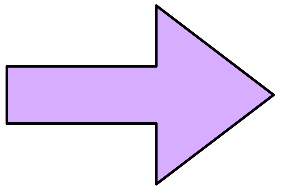
IPv6 Addresses

- 128 bit binary numbers
- (instead of IPv4's 32 bit binary numbers)
- 2^{128} different combinations = more than $3x10^{38}$
- converted to hexadecimal for human readability
- binary numbers are easily converted into hexadecimal
    - much easier than binary into decimal

IPv6 in Hexadecimal Notation

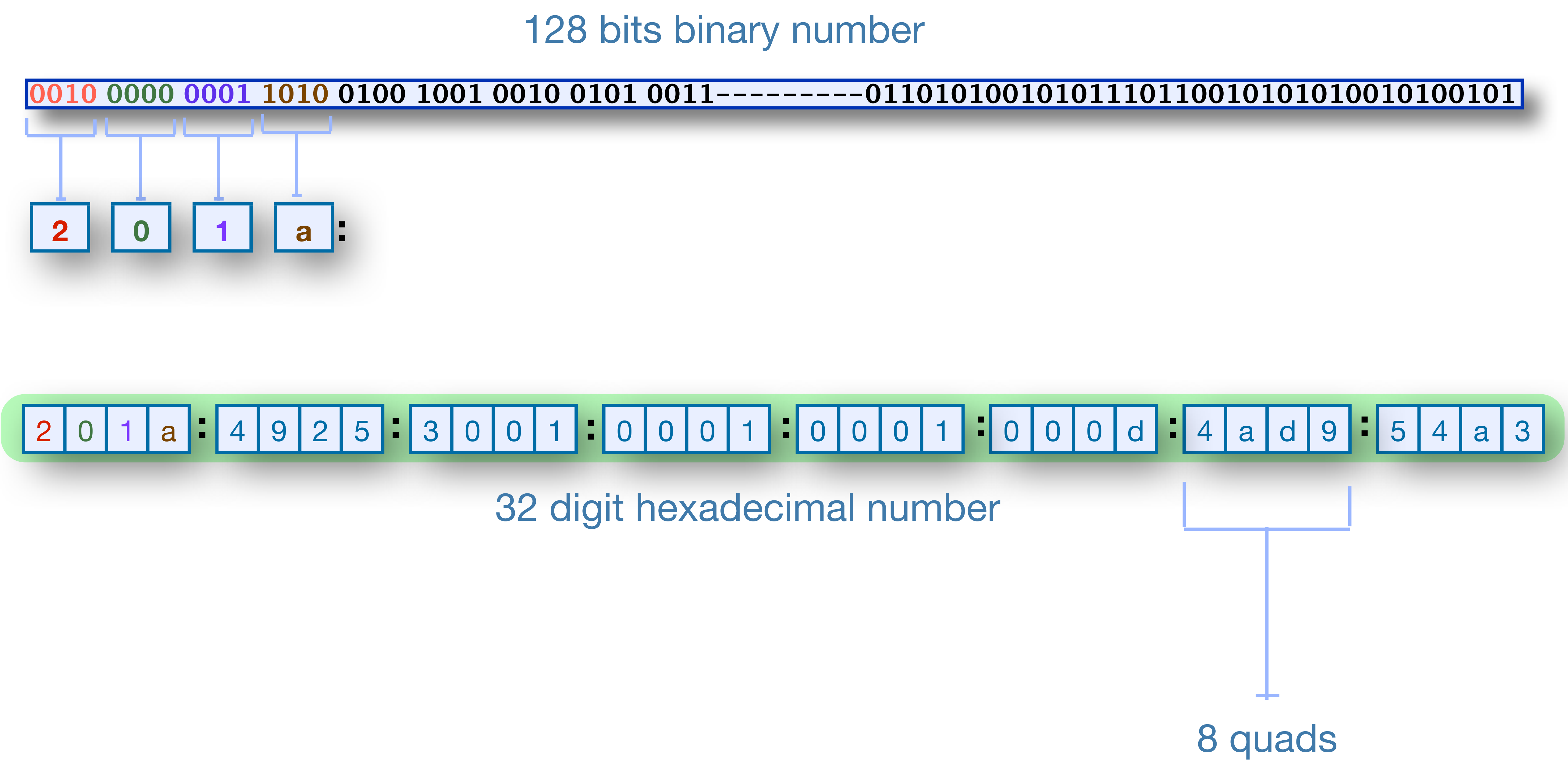
- 128 bit binary number
- replace each set of 4 bits → with hexadecimal number
- = 32 hexadecimal numbers
  - grouped into sets of 4 hex numbers, divided by ':'
- eg 2001:db8:0000:1234:0000:123e:1456:0234
  - shortened to 2001:db8:0:1234::123e:1456:234

Replacing

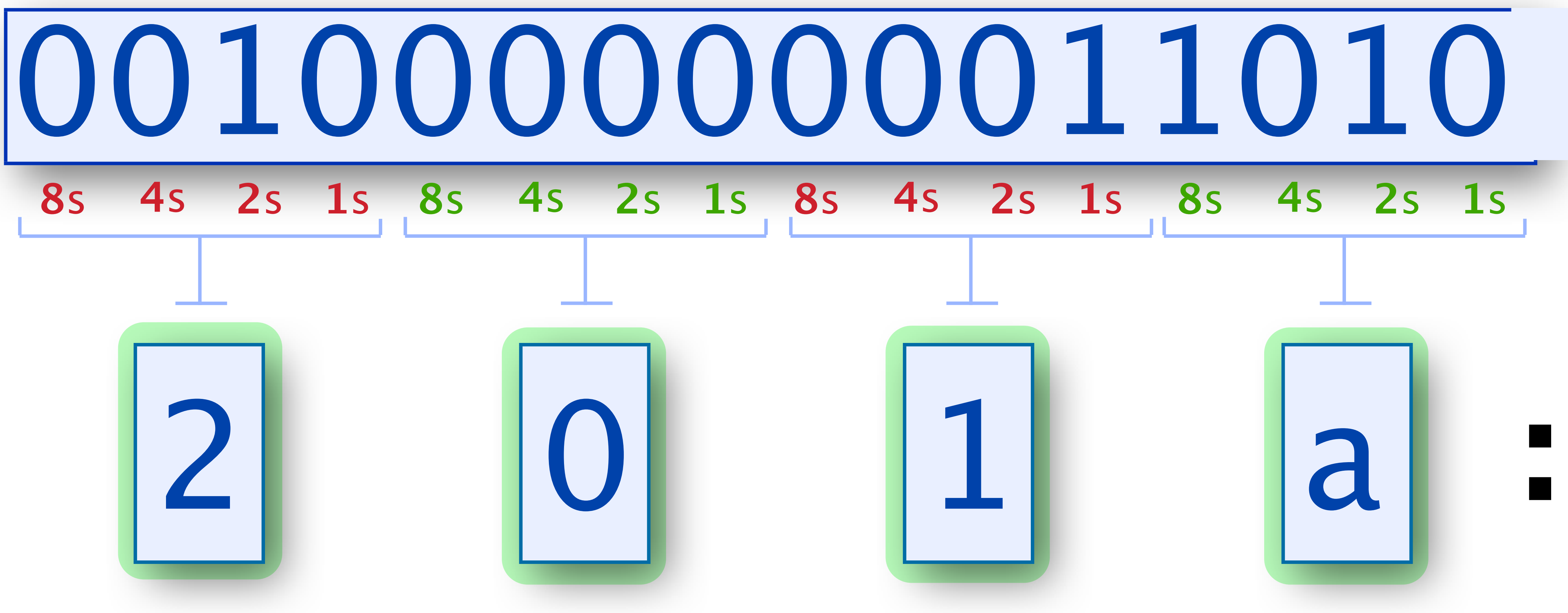


Binary	Hexadecimal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	a
1011	b
1100	c
1101	d
1110	e
1111	f

An IPv6 Address



IPv6 Address: First 16 Bits in Detail



Replace each set of 4 bits with corresponding hexadecimal number

Address Notation

2001:0db8:003e:ef11:0000:0000:c100:004d

Address Notation

2001:0db8:003e:ef11:0000:0000:c100:004d

2001:db8:3e:ef11:0:0:c100:4d

Address Notation

2001:0db8:003e:ef11:0000:0000:c100:004d

2001:db8:3e:ef11:0:0:c100:4d

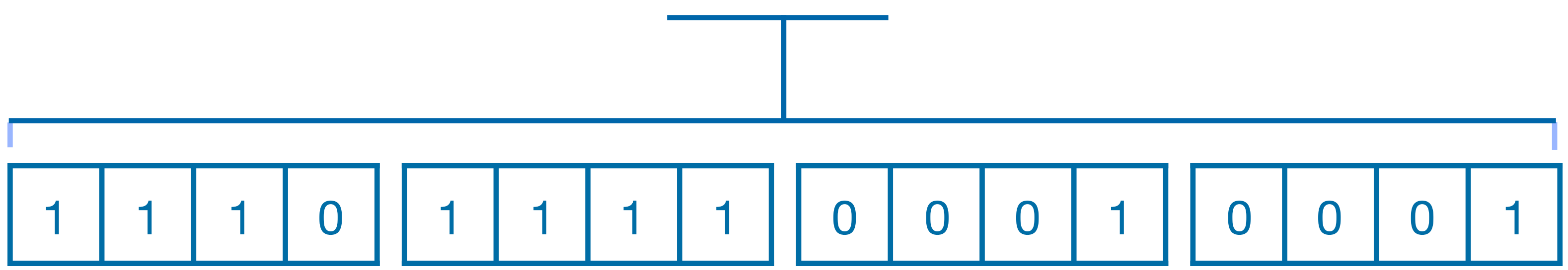
2001:db8:3e:ef11::c100:4d

Address Notation

2001:0db8:003e:ef11:0000:0000:c100:004d

2001:db8:3e:ef11:0:0:c100:4d

2001:db8:3e:ef11::c100:4d



AAAA Record

- IPv6 Address Record
- rdata contains an IPv6 address

```
www.ripe.net. IN AAAA 2001:67c:2e8:22::c100:68b
```

CNAME Record

- An “alias”
  - maps one name to another (regardless of type)
- rdata contains the mapped domain name (“canonical name”)

```
website.ripe.net. IN CNAME www.ripe.net.
```

MX Record

- Mail Exchanger: defines the host receiving mail
- rdata consists of a preference field and the hostname of the mail receiver
- Lower preference = higher priority

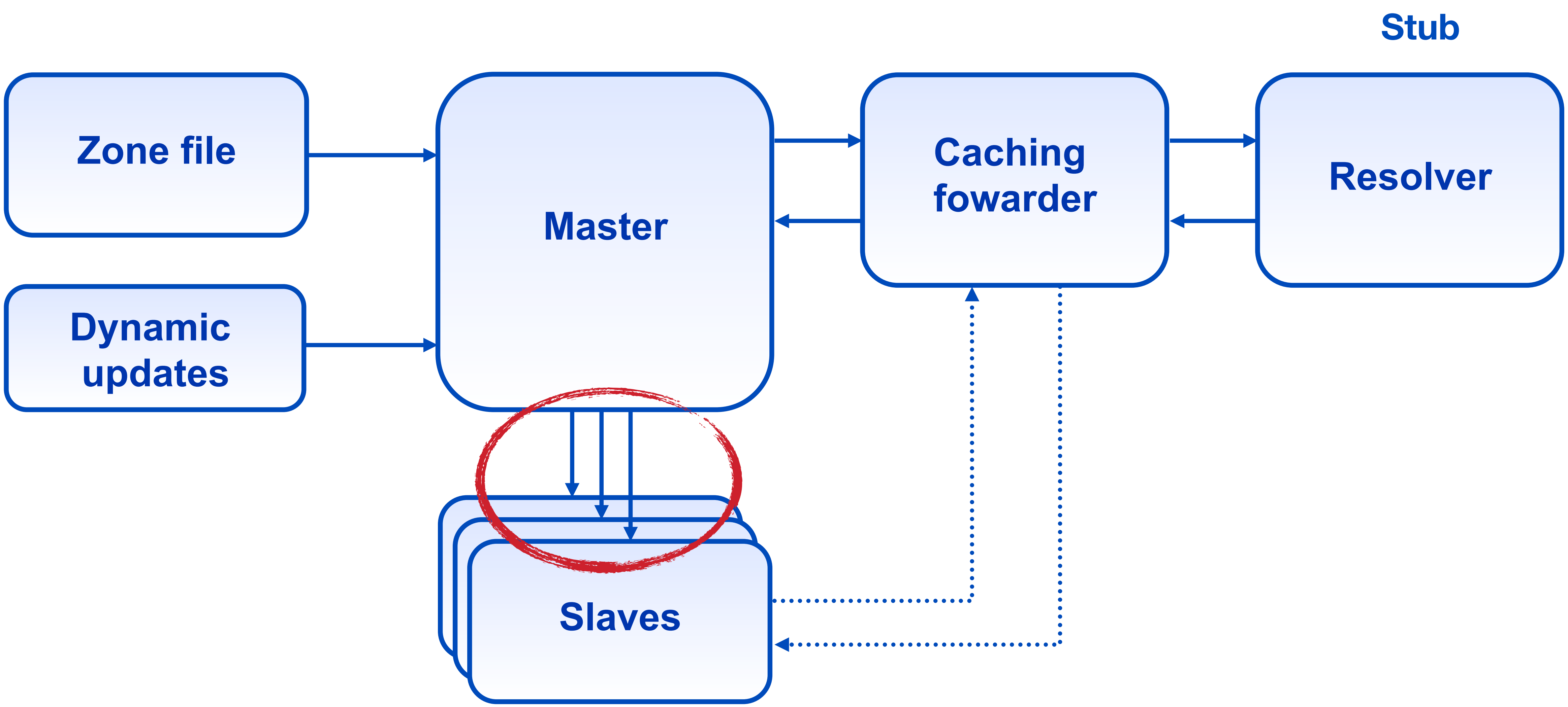
```
ripe.net. IN MX 10 mail1.ripe.net.
ripe.net. IN MX 20 mail2.ripe.net.
```

SRV Record

- Service record: "generic" description of a service
- rdata consists of a **preference** field, a **weight** field and a **host** providing the service
- Lower preference = higher priority
- Higher weight = more often used

```
_kerberos._tcp IN SRV 10 5 88 krb1
_kerberos._tcp IN SRV 10 10 88 krb2
_kerberos._tcp IN SRV 20 5 88 krb3
```

Zone Data Synchronisation



SOA Record

- Defines the start of a new zone
  - also, important parameters for the zone
- Always appears at the beginning of the zone
- Serial number should be incremented on zone content updates

```
ripe.net. IN SOA pri.authdns.ripe.net. dns.ripe.net. 1399456383 3600
600 864000 300
```

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                    200702270 ; serial
                    28800       ; refresh
                    14400       ; retry
                    3600000   ; expire
                    86400       ; neg ttl
...
```

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

SOA
name of master server

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                        200702270 ; serial
                        28800       ; refresh
                        14400       ; retry
                        3600000   ; expire
                        86400       ; neg ttl
...
```

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

SOA
name of master server

RP
(responsible person's email)
Replace first "." by "@"

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                         200702270 ; serial
                         28800         ; refresh
                         14400         ; retry
                         3600000   ; expire
                         86400         ; neg ttl
...
```

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

SOA
name of master server

RP
(responsible person's email)
Replace first "." by "@"

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                        200702270 ; serial
                        28800       ; refresh
                        14400       ; retry
                        3600000   ; expire
                        86400       ; neg ttl
...
```

For replication between namservers

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                     200702270 ; serial
                     28800       ; refresh
                     14400       ; retry
                     3600000   ; expire
                     86400       ; neg ttl
...
```

SOA name of master server

RP (responsible person's email) Replace first "." by "@"

For replication between namservers

How often slave server checks master for new data

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                     200702270 ; serial
                     28800       ; refresh
                     14400       ; retry
                     3600000   ; expire
                     86400       ; neg ttl
...
```

SOA name of master server (→ ns.<domain>.)

RP (responsible person's email) Replace first "." by "@" (→ root.<domain>.)

For replication between namservers (→ serial)

How often slave server checks master for new data (→ refresh)

If refresh fails, how often retry (→ retry)

Authority: Who Owns This Data

- Query the SOA (Start of Authority) for a domain:

SOA
name of master server

RP
(responsible person's email)
Replace first "." by "@"

```
# dig SOA <domain>
...
;; AUTHORITY SECTION:
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
                    200702270 ; serial
                    28800       ; refresh
                    14400       ; retry
                    3600000   ; expire
                    86400       ; neg ttl
...
```

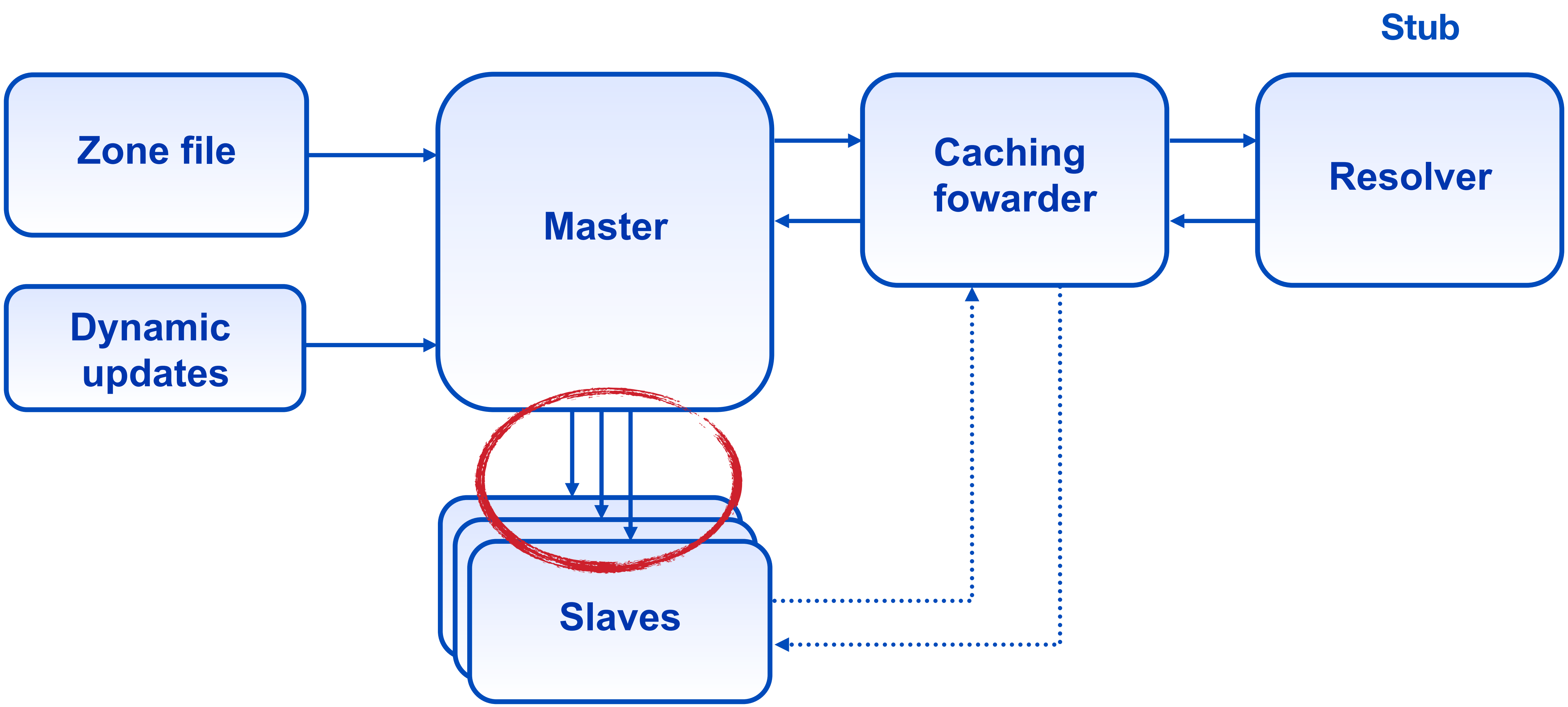
For replication between namservers

How often slave server checks master for new data

If refresh fails, how often retry

If master failed to answer for this long, don't hand out this data to clients

Zone Data Synchronisation



Zone Data Synchronisation

- Data can be synchronised in many ways
- Full zone transfers: AXFR
  - slaves periodically check if new data is present (SOA refresh interval)
- Incremental zone transfers: IXFR
  - slaves are notified when new data is loaded

- Zone transfers can be authenticated using TSIG (Transaction SIGnatures)

Software overview

Why have multiple implementations

- Software **will** have bugs
  - BIND certainly has a ***fairly suspect*** record
  - Paul Vixie: "***most CERT advisories for a single author***"
- Diversity is good in any gene pool
- Competition is good in any market

DNS server implementations

- BIND
- NSD
- knot
- unbound
- djbdns
- PowerDNS
- Microsoft
- Others...

DNS server implementations

- BIND
- NSD
- knot
- unbound
- djbdns
- PowerDNS
- Microsoft
- Others...

Reference implementation – hugely popular! (BIND)

Widely used in many places (NSD, knot, unbound)

Many loyal users but nowhere near as many as the above (djbdns, PowerDNS)

Occasionally encountered in the wild (Microsoft)
Essentially extinct outside labs (Others...)

DNS server implementations

- We will focus only on BIND, NSD and unbound
- Zone files 100% compatible!
  - Well … more than 90% at least
  - Modulo *really* obscure constructs
- BIND is dual personality: authoritative / recursive
- NSD is authoritative-only
- Unbound is recursive-only

DNS server implementations

- BIND included by default in many systems
- unbound included with FreeBSD 10+
- NSD and unbound included with OpenBSD
- NSD and unbound included with NetBSD

All implementations we talk about are trivial to install on any reasonable operating system.

BIND

- Reference implementation
  - Authoritative and Recursive personalities
- Originally written at UCB in the early 1980s
- Actively maintained by ISC
- Three complete rewrites: BIND4, BIND8, BIND9
- BIND10, Bundy, (Kea)

NSD

- Authoritative-only implementation
- Written by NLNet Labs early 2000s
- Actively maintained by NLNet Labs and contribs
- Focus on performance, *code hygiene*, security
- Aims for data compatibility with BIND

Unbound

- **Recursive-only implementation**
- **Written by NLNet Labs in early 2000s**
- **Actively maintained by NLNet Labs and contribs**
- **Focus on performance, *code hygiene*, security**
- **Strong focus on DNSSEC from outset**

Tools and libraries

- BIND comes with a set of "host tools"
  - dig, dnssec-keygen
- LDNS is a *lightweight* library implementation
  - Used by unbound
  - Used by drill (dig reimplementation)
  - Easy to use for automated checks, etc
- doc, nslint, ...
  - Simple tools to make your life easier

Helpful online services

- https://dnsviz.net/
  - Debug many obscure DNSSEC issues
- http://www.zonecut.net/dns/
  - Pretty delegation graphs
  - Debug "standard" (no-DNSSEC) delegations

Conclusion

- DNS = vast subject
- caching & recursion
- Several servers for the same data
    - you don’t always talk to the same one
- Practise!

?

DAY TWO

Overview day 2

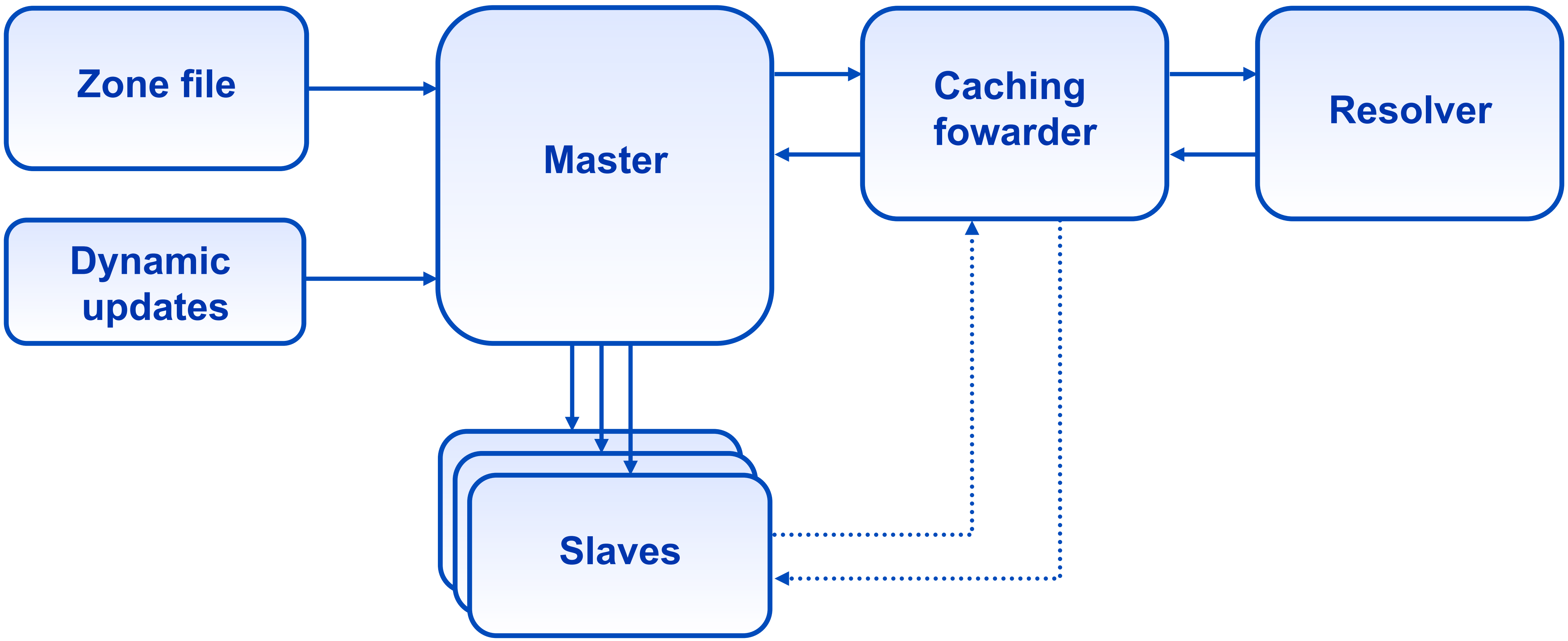
- Configuring authoritative DNS servers
    - Writing and analysing zonefiles
    - Delegating authority
    - Debugging common zonefile problems
- A very brief introduction to cryptography
- Configuring secondary DNS servers
    - Setting up TSIG to secure zone transfers
    - Debugging common zone transfer issues
- Configuring recursive DNS servers

Overview day 2

- Configuring authoritative DNS servers
  - Writing and analysing zonefiles
  - Delegating authority
  - Debugging common zonefile problems
- A very brief introduction to cryptography
- Configuring secondary DNS servers
  - Setting up TSIG to secure zone transfers
  - Debugging common zone transfer issues
- Configuring recursive DNS servers

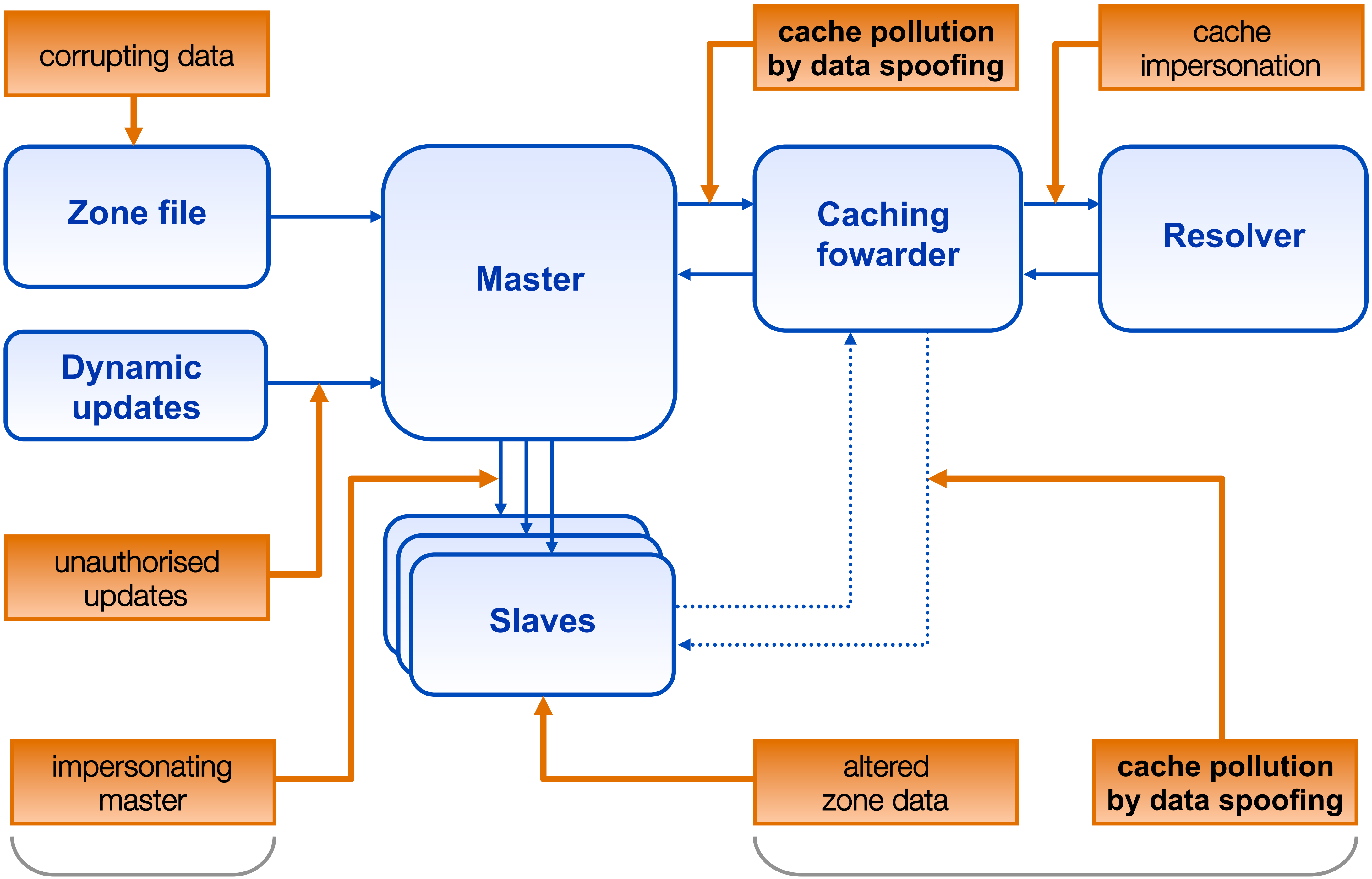
By the end of day 2, you can get a job as a DNS admin!

DNS Data Flow

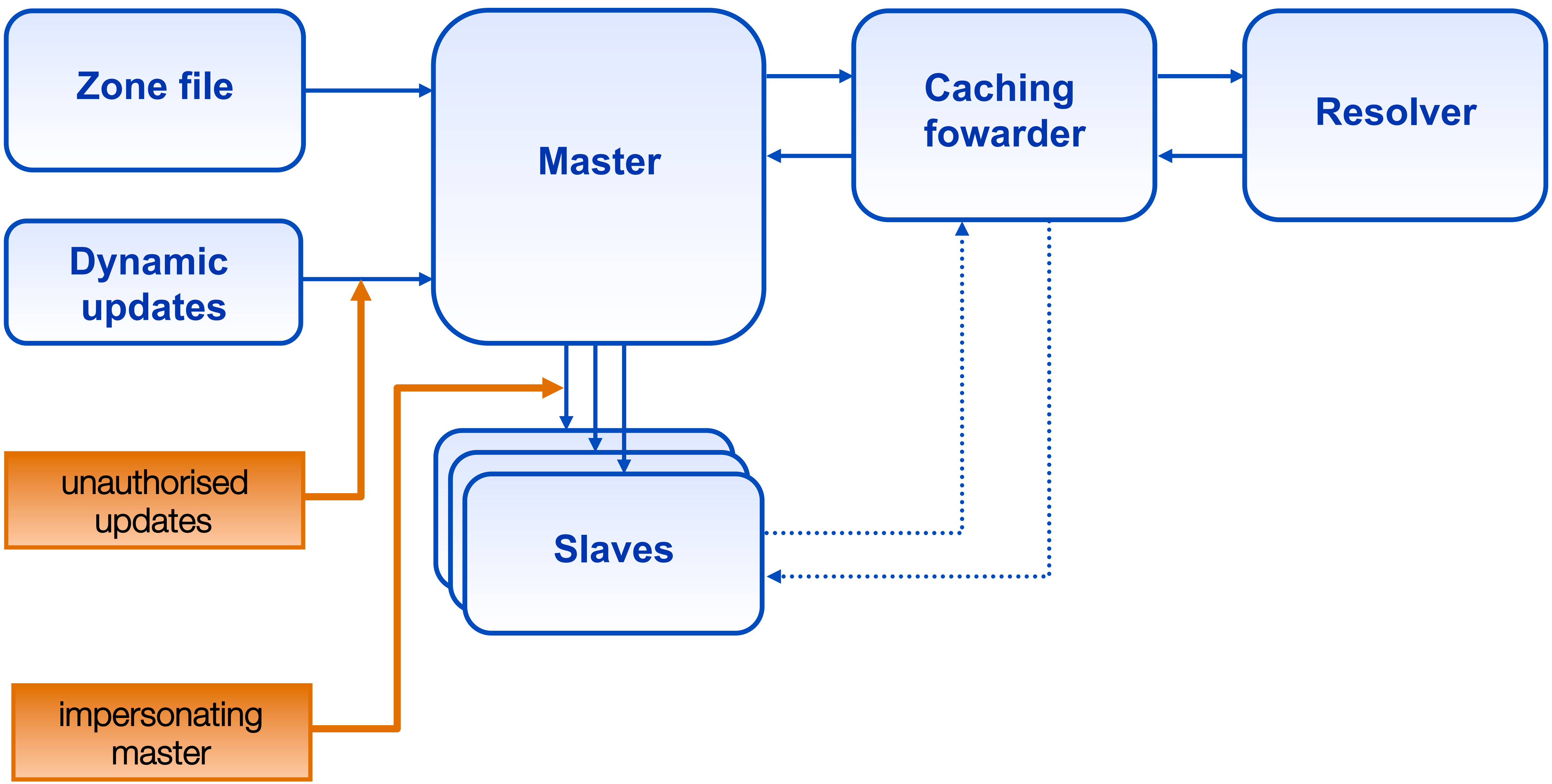


DNS Vulnerabilities

DNS Vulnerabilities

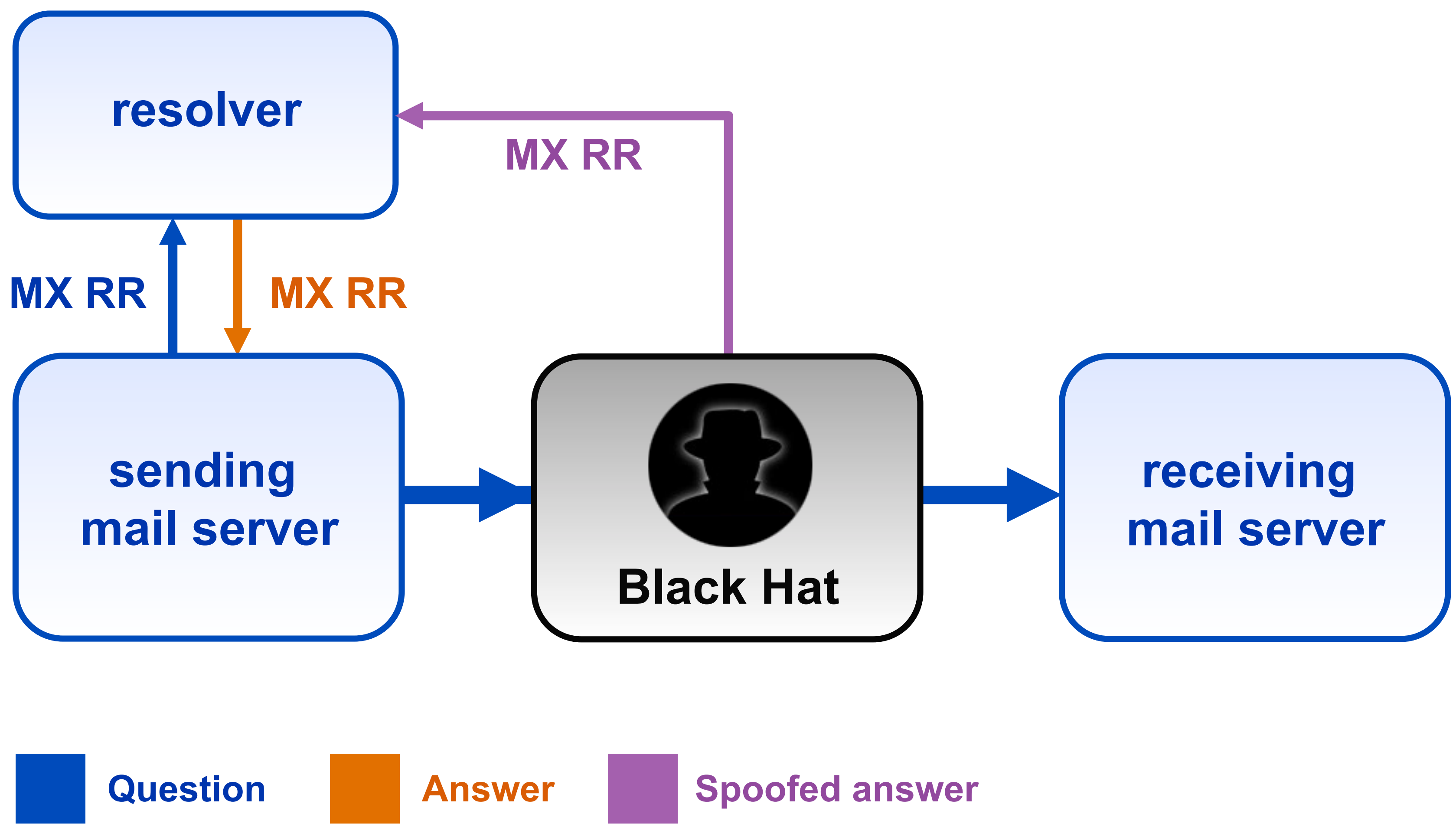


TSIG Protected Vulnerabilities



DNS Exploit Example

- Mail goes to the server in the MX resource record
- Path only visible in the email headers



Introduction to Crytography

Cryptography

- A way to encrypt or hash some content
    - Make it “secure” and/or verifiable
- Intent is not always to hide the message
    - For DNSSEC, goal is to verify the content
- Different methods and keys

Hashes

- Turns a string into a different series of characters
- Fixed length

Hashes

- **Turns a string into a different series of characters**
- **Fixed length**

SHA256 ("This is the DNSSEC Course")
a8feb4dd098d86d1ea326e4c7178ad5dcbacacabb4df421
c0f4bbe04f28077a2

Hashes

- Turns a string into a different series of characters
- Fixed length

SHA256 (“This is the DNSSEC Course”)
a8feb4dd098d86d1ea326e4c7178ad5dcbacacabb4df421
c0f4bbe04f28077a2

SHA256 (“This is the DNSSEC Course for LIRs”)
74fda40946cb6bc835b3322bc0b0a6643aca1ce38af4f88c
a114edec859bec68

Public Key Cryptography

- Most commonly used cryptographic system
- Can guarantee security and authentication
  - Via encryption
  - Via signatures
  - Or both

Public and Private Keys

- Key-pair
  - One private
  - One public
- Content encrypted with one key, can only be decrypted with the other one
  - A public key can “open” content encrypted with the private key, and viceversa

Digital signatures

- If we combine hashes and public key encryption, we obtain a digital signature

- We generate a hash, then encrypt it with a key
  - We check the authenticity by decrypting it, hashing the message again and comparing with the hash received

- If the hashes match, nobody tampered with the message

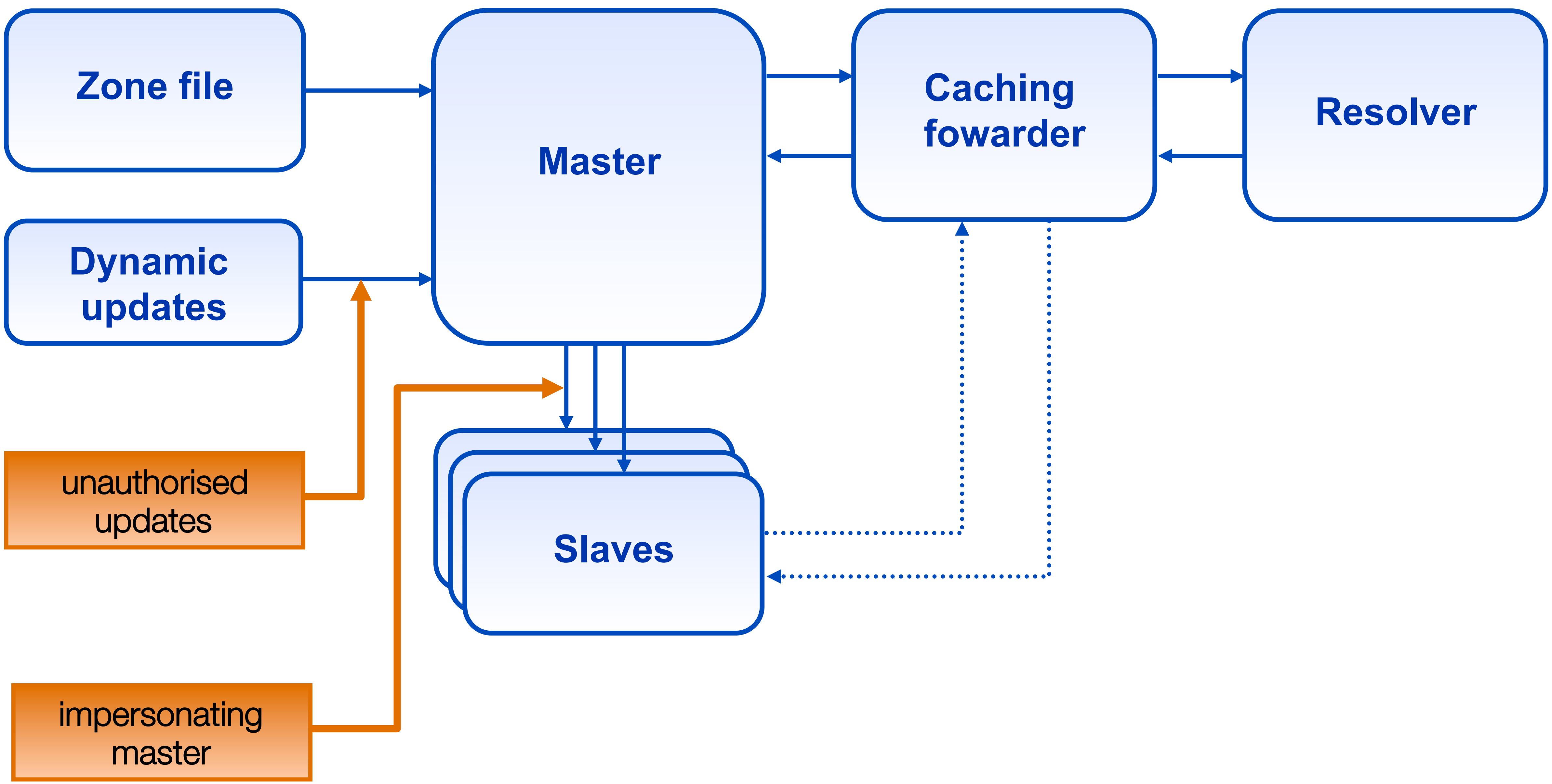
?

TSIG:
Securing Host-to-Host Communication

Transaction Signatures

- Data is secured for clients, but not for transfers to slaves
- Transaction Signatures (TSIG) can ensure data integrity for zone transfers
- Not part of DNSSEC
  - But they complement it

TSIG Protected Vulnerabilities



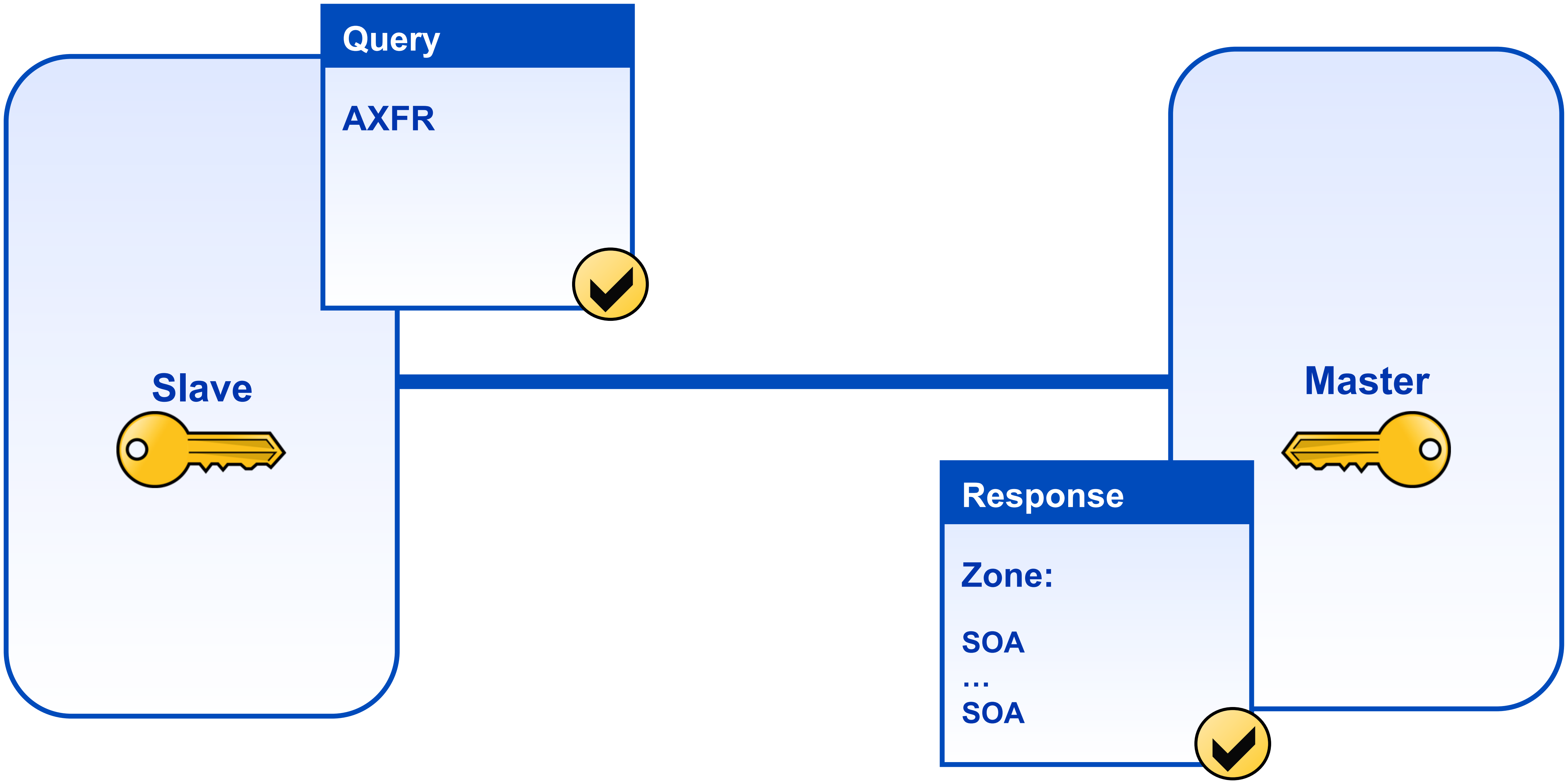
Transaction Signatures

- An attacker could easily read the data in your zone transfers and modify it
  - They happen in cleartext
- Symmetric encryption with shared keys is used

Transaction Signature: TSIG

- TSIG (RFC 2845)
  - Authorising dynamic updates and zone transfers
  - Authentication of caching forwarders
  - Independent from other features of DNSSEC
- One-way hash function
  - DNS question or answer and timestamp
- Traffic signed with “shared secret” key
- Used in configuration, NOT in zone file

TSIG Summary



TSIG for Zone Transfers

1. Generate secret
2. Communicate secret
3. Configure servers
4. Test

Generate TSIG Secret

- dnssec-keygen -a <alg> -b <bits> -n <type> [options] <keyname>

- algorithm: HMAC-SHA256
- ‘-r /dev/urandom’ might be needed
- Bits: 256
- type:host
- Name: unique identifier
  - Suggested: master-slave.zone.name.

- TSIG secret can be generated differently
  - base64 encoding

Master Server: named.conf

- "Key" statement to configure key

```
 key ”me-friend." {
algorithm hmac-sha256;
  secret “nEfRX9jxOmzsby8VKRgDWEJorhyNbjt1ebbPn7lyQtE=";
};
```

- "allow-transfer" indicates which keys are allowed
  - can be combined with IP based restrictions

```
 zone ”example.net" {
         type master;
         file "zones/example.net.";
         allow-transfer { key me-friend.; };
         notify yes;
 };
```

Slave Servers: named.conf

- "key" statement to configure the key

```
key ″me-friend." {
  algorithm hmac-sha256;
  secret “nEfRX9jxOmzsby8VKRgDWEJorhyNbjt1ebbPn7lyQtE=";
  };
```

- "server" statement to indicate key used
  - zone configuration doesn’t change on slave server

```
server 192.168.10.1 {
        keys {me-friend.; };
  };
```

Importance of the Time Stamp

- TSIG/SIG(0) signs a complete DNS request / response with time stamp
  - To prevent replay attacks
  - Currently hardcoded at five minutes

- Operational problems when comparing times
  - Make sure your local time zone is properly defined
  - `date -u` will give UTC time, easy to compare between the two systems
- Use NTP synchronisation!

?

Introduction to DNSSEC

Basic DNS problems

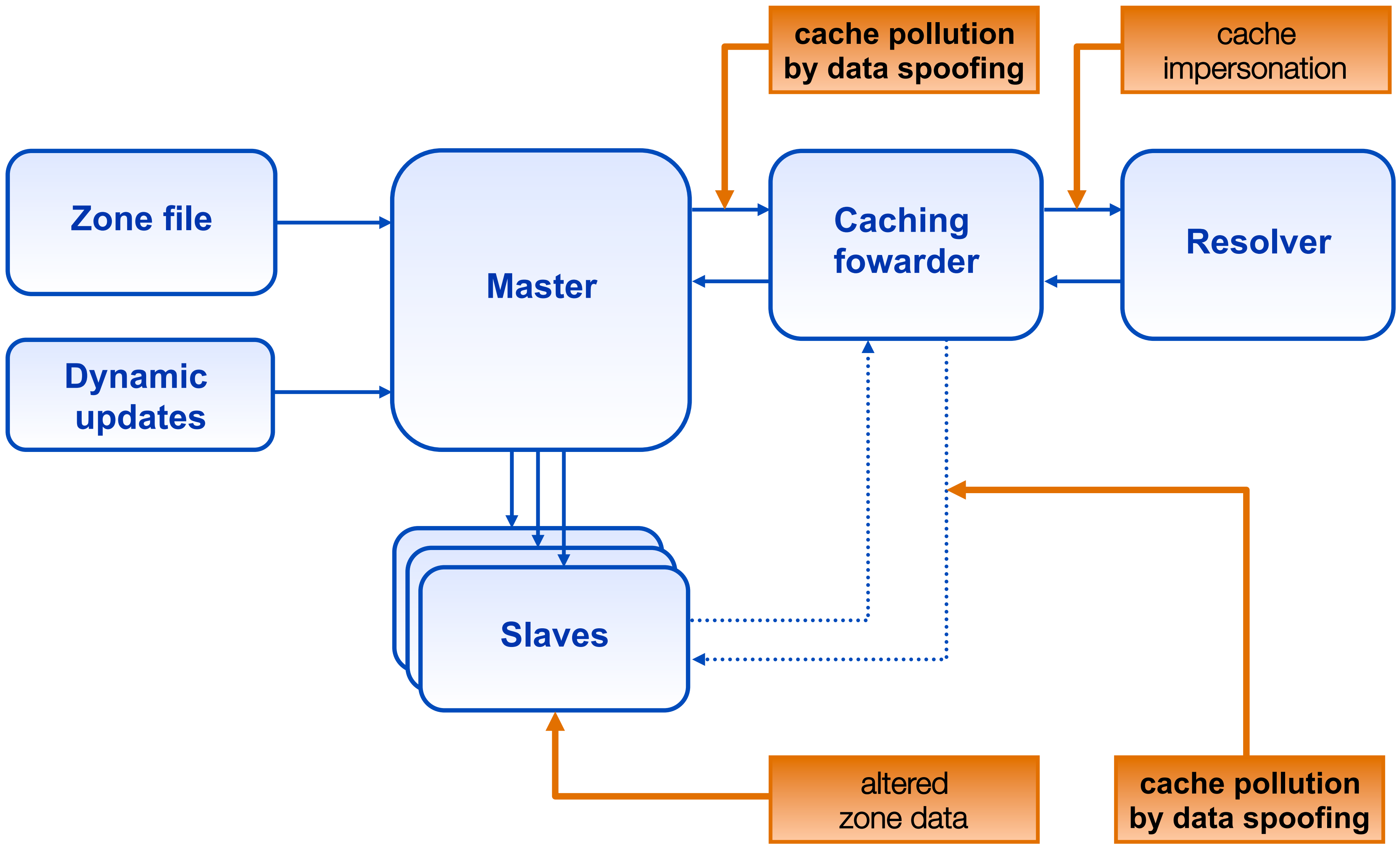
- DNS is plain text
- Simple UDP, no sessions
- A tree structure with delegations
  - Each entity is responsible for a limited part of it
  - We have to blindly trust the other entities
- Resolvers are prone to different kinds of attacks, hijacks and mistakes

- Trust is needed

DNSSEC

- DNS Security Extensions
- RFC4033
- Adds layers on top of DNS to make it verifiable
    - Adds new record types
    - Adds PKI
- A chain of trust is created to validate the data

DNSSec Protected Vulnerabilities



DNSSec Hypersummary

- Data authenticity and integrity by signing the Resource Records Sets with private key
- Public DNSKEYs used to verify the RRSIGs
- Children sign their zones with their private key
  - Authenticity of child's key established by the parent signing hash of child's key (DS)
- Repeat for parent …
  - …and grandparent
- Ideal case: one public DNSKEY distributed

DNSSec Summary

ripe.net.

www.ripe.net	IN 900 **A 193.0.0.214**
www.ripe.net	IN 900 **RRSIG A** ... **26523 ripe.net.** ...
ripe.net	IN 3600 **DNSKEY** 256 3 5 ...
ripe.net	IN 3600 **RRSIG DNSKEY** ... **26523 ripe.net.** ...

net.

ripe.net	IN 3600 **DS 26523** 5 1 ...
ripe.net	IN 3600 **RRSIG DS** .... **573 net. ...**

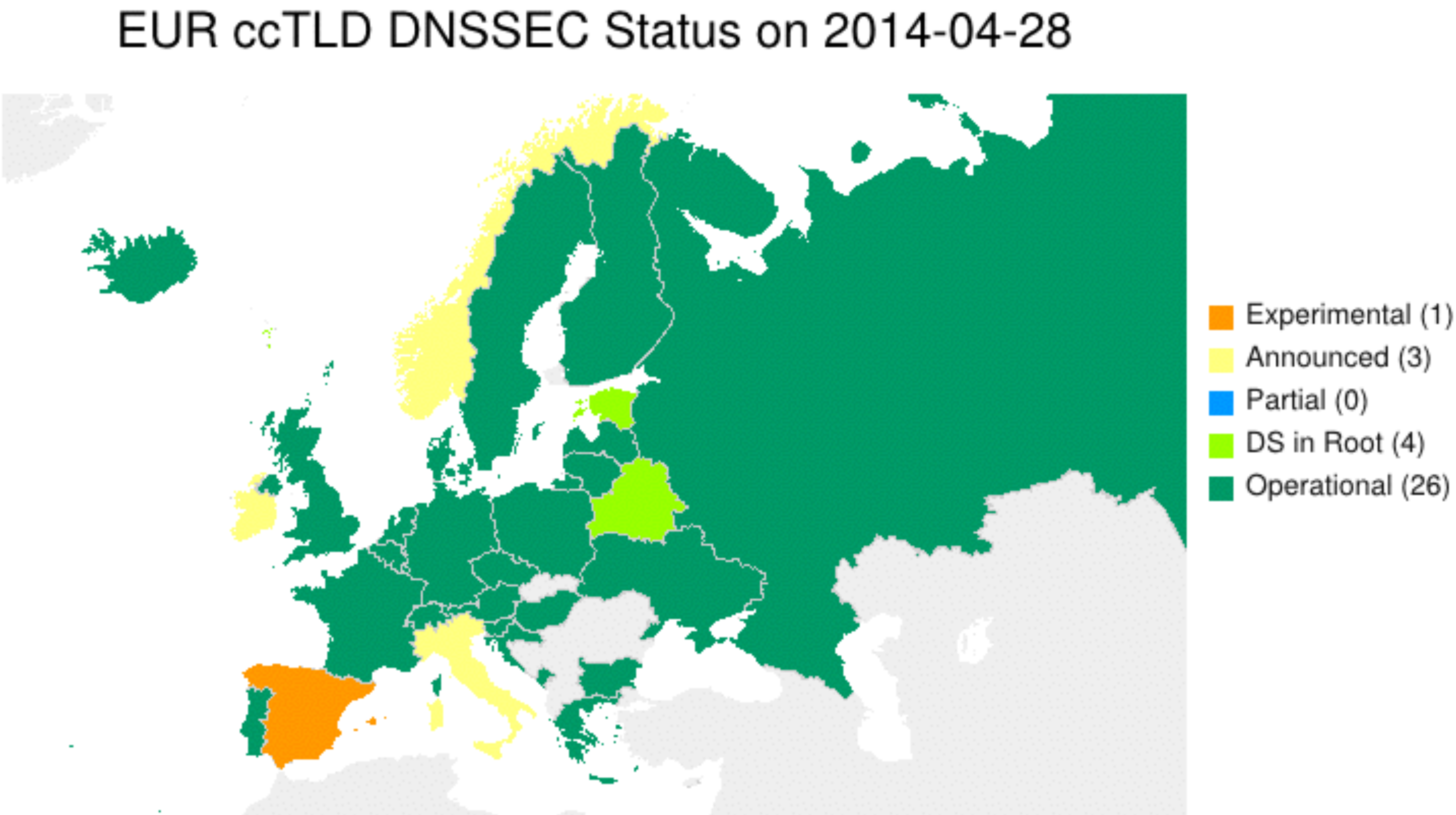
Locally Configured Verifier (named.conf)

trusted-keys { “ripe.net." 256 3 5 “..."; };

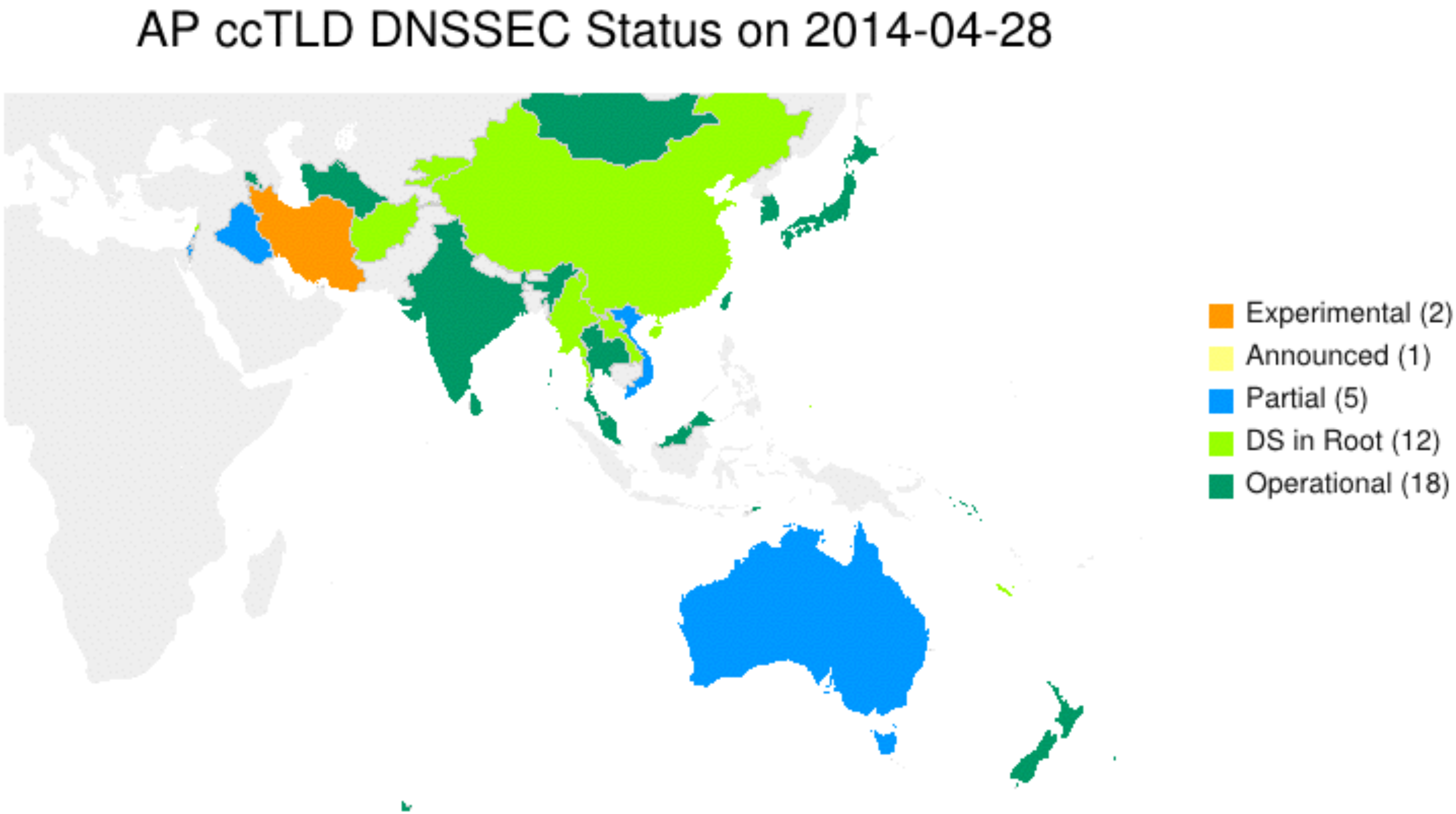
Security Status of Data (RFC 4035)

- Secure
  - Resolver can build chain of signed DNSKEY and DS RRs from trusted anchor to RRset
- Insecure
  - Resolver knows it has no chain of signed DNSKEY and DS RRs from any trusted starting point to RRset
- Bogus
  - Resolver thinks it can build a chain of trust but it is unable to do so
  - May indicate attack or configuration error or data corruption
- Indeterminate
  - Resolver cannot determine whether the RRset should be signed

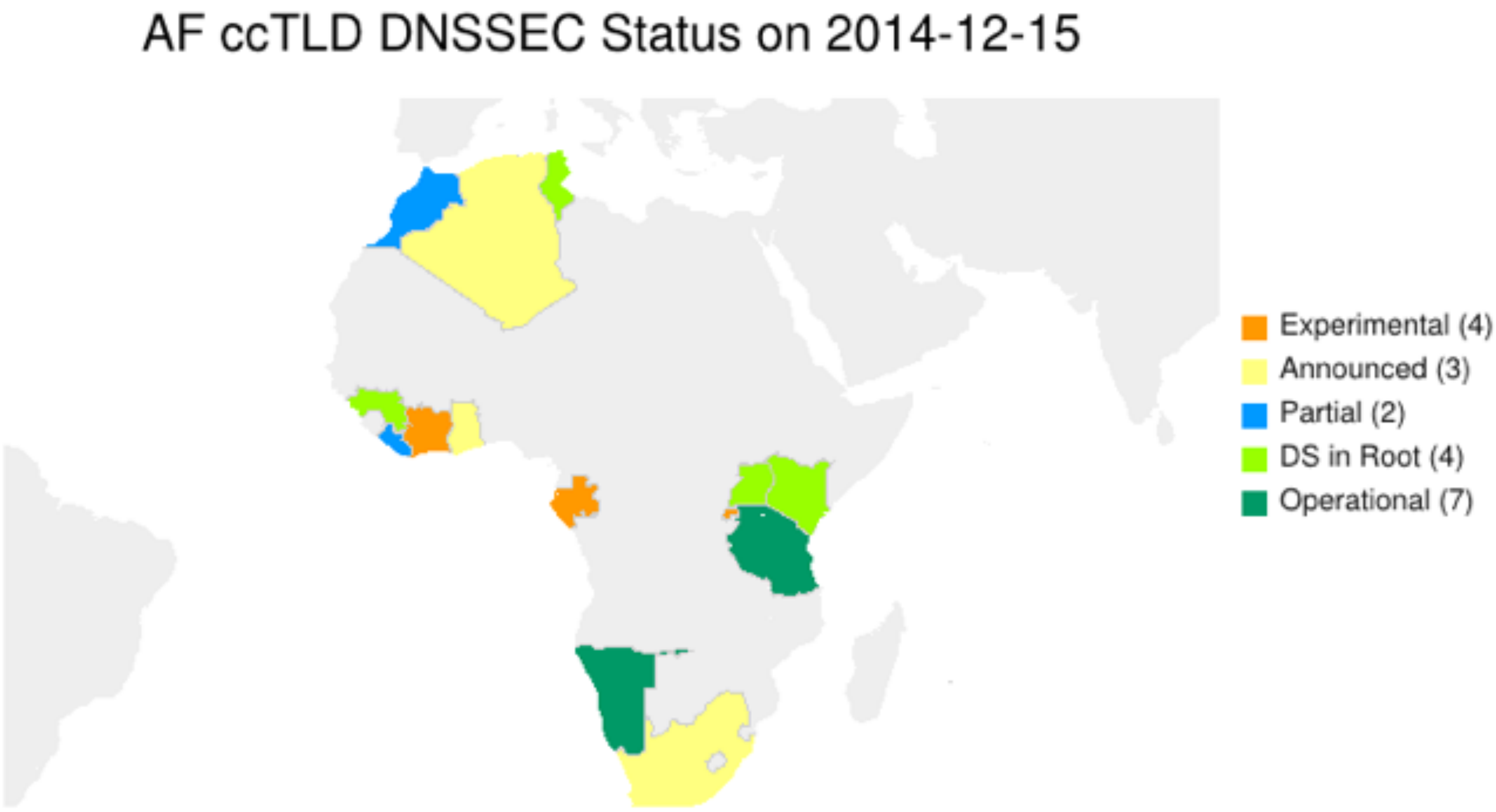
DNSSEC Adoption in Europe



DNSSEC Adoption in Asia



DNSSEC adoption in africa



?

DNSSec:
New Resource Records in DNS

RRs and RRSets

- Resource Record:
  - name TTL class type rdata
  - `www.ripe.net. 7200 IN A 192.168.10.3`

- RRset: RRs with same name, class and type:
  - `www.ripe.net. 7200 IN A 192.168.10.3`
  - `www.ripe.net. 7200 IN A 10.0.0.3`
  - `www.ripe.net. 7200 IN A 172.25.215.2`

- RRSets are signed, not the individual RRs

New Resource Records

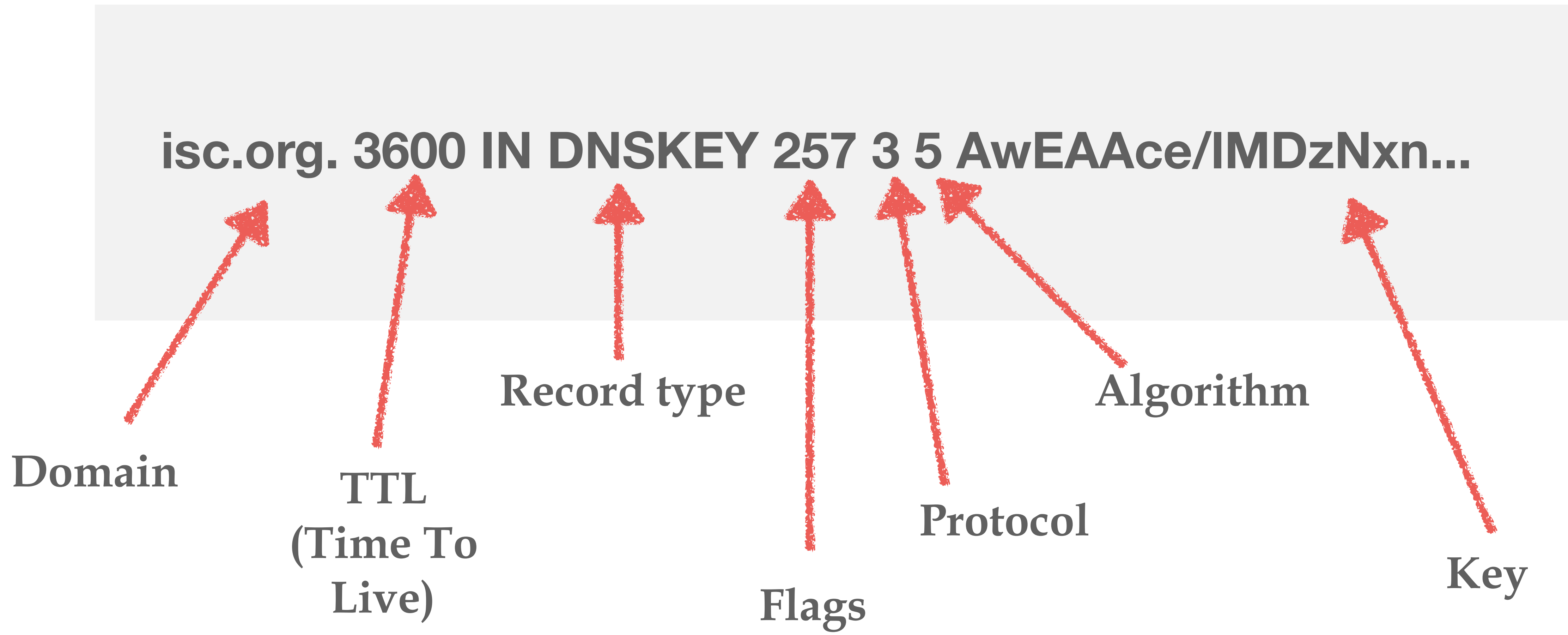
- Three Public key crypto related RRs
  - **RRSIG** Signature over RRset made using private key
  - **DNSKEY** Public key, needed for verifying a RRSIG
  - **DS** Delegation Signer; ‘Pointer’ for building chains of authentication
- One RR for internal consistency
  - **NSEC** Indicates which name is the next one in the zone and which typecodes are available for the current name
  - authenticated non-existence of data

Overview

- **DNSKEY:** Contains zone public key
- **RRSIG:** Contains the key signature
- **DS:** Delegation Signer
- **NSEC:** Points to the next name in the zone
- **NSEC3:** Enhanced version of NSEC

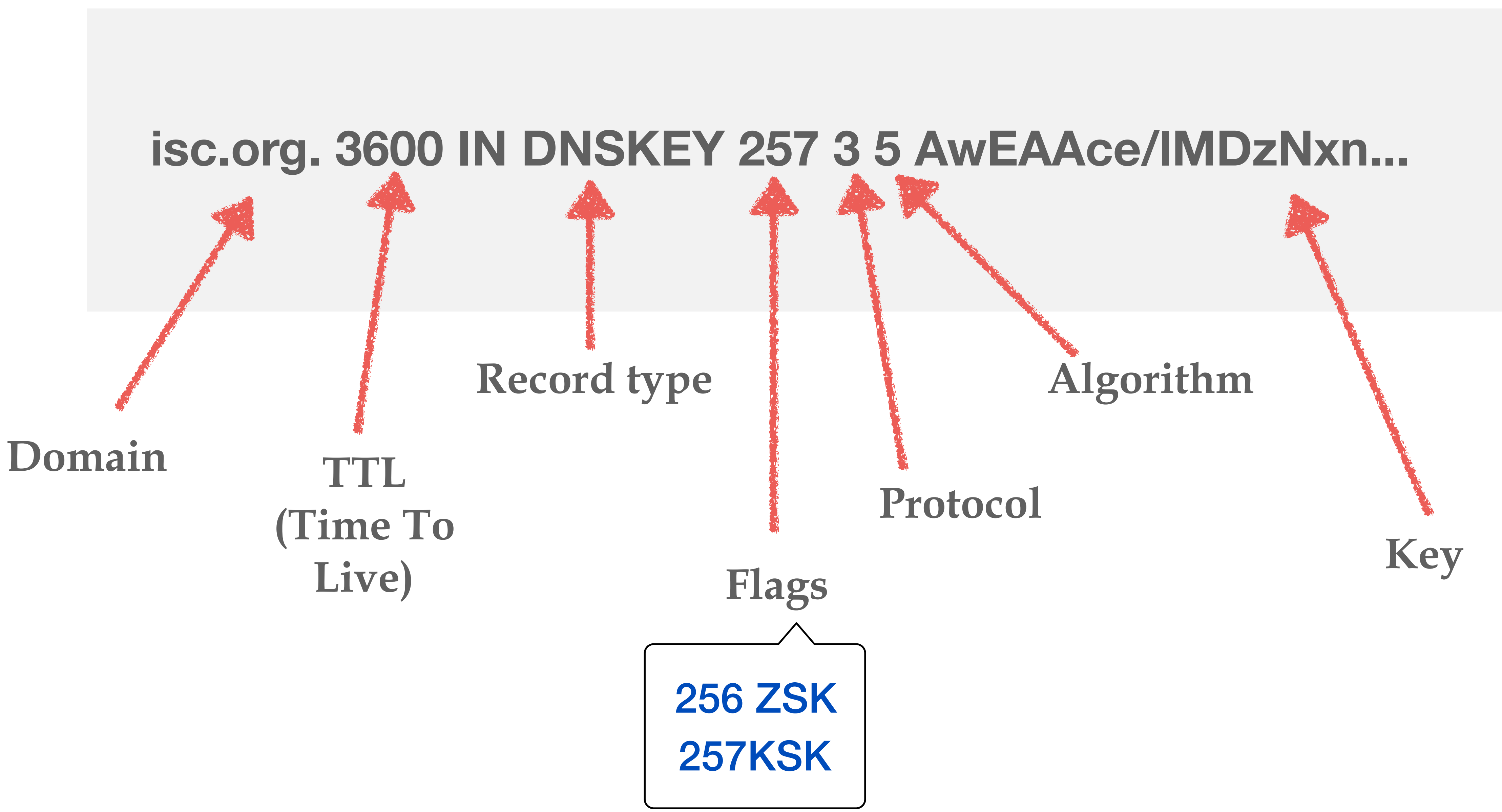
DNSKEY

- Contains Zone's public key(s)

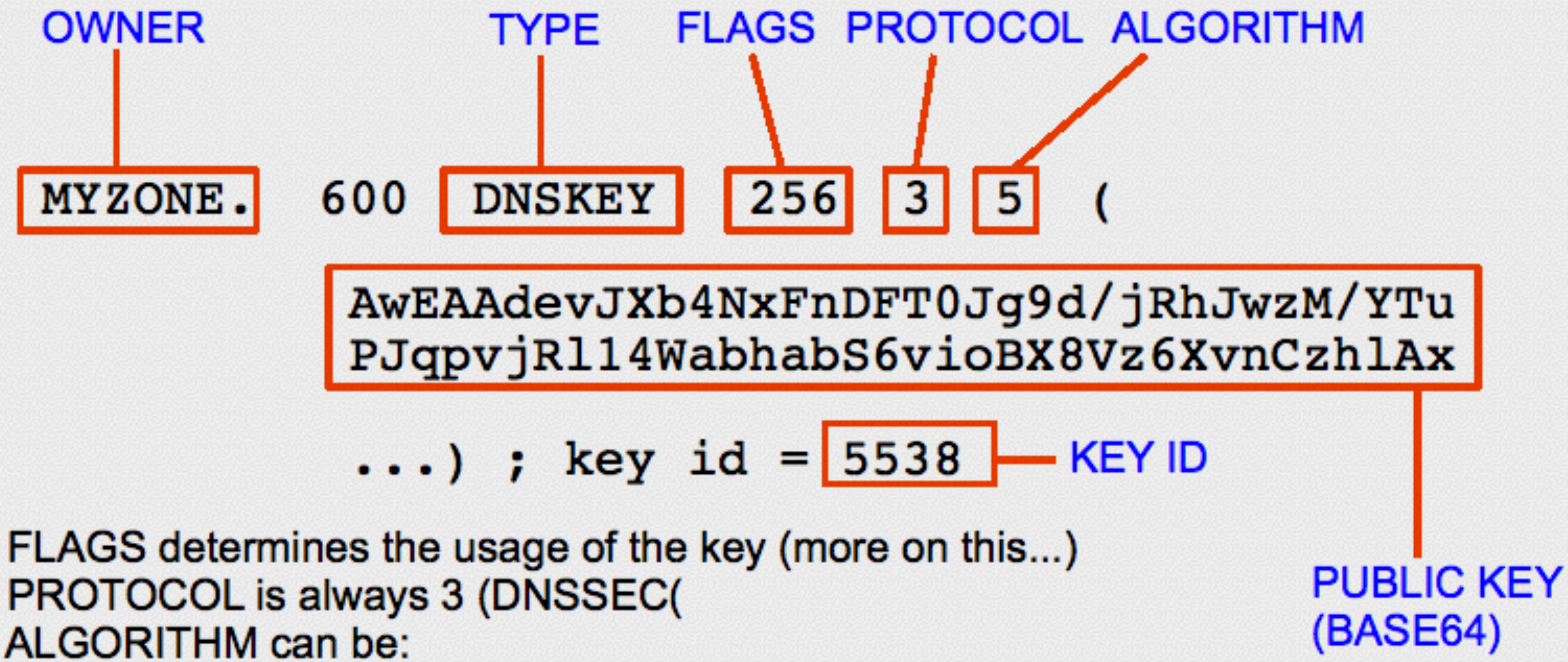


DNSKEY

- Contains Zone's public key(s)



DNSKEY (cont.)

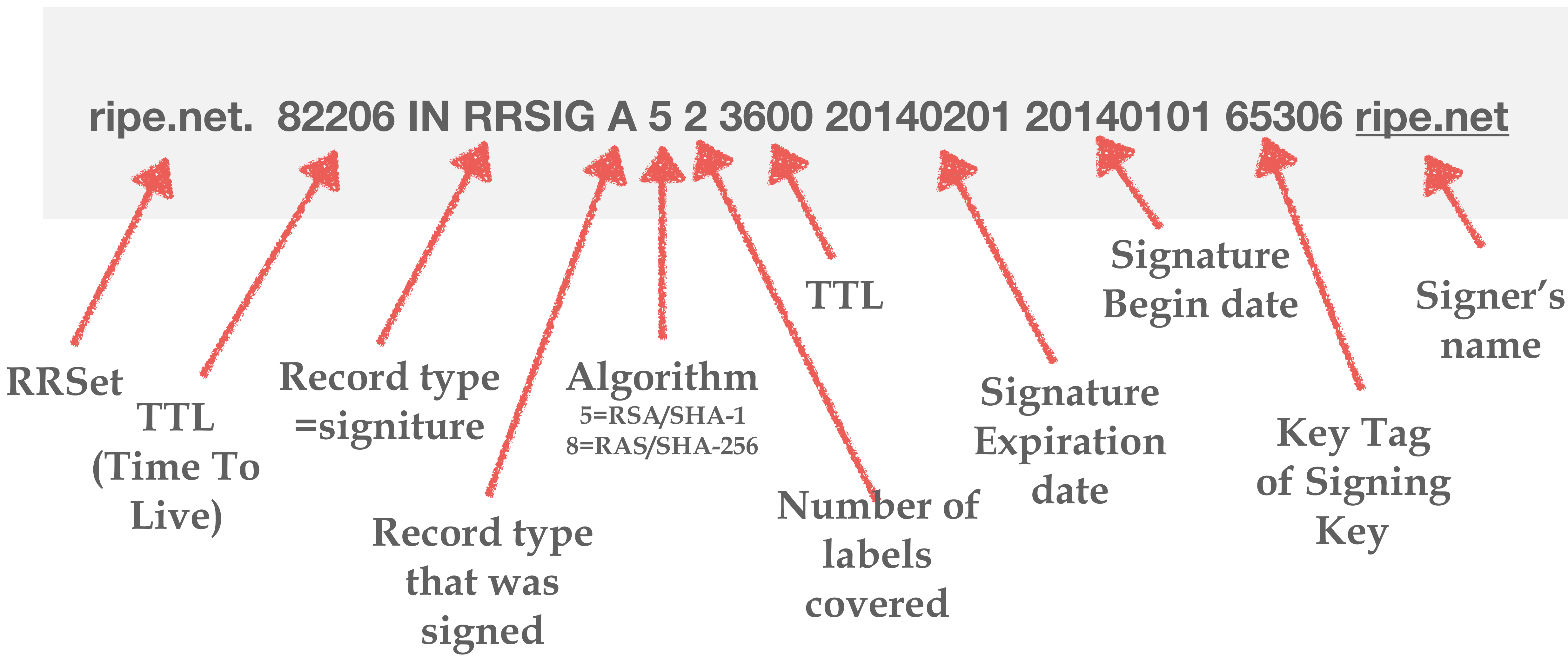


- FLAGS determines the usage of the key (more on this...)
- PROTOCOL is always 3 (DNSSEC(
- ALGORITHM can be:

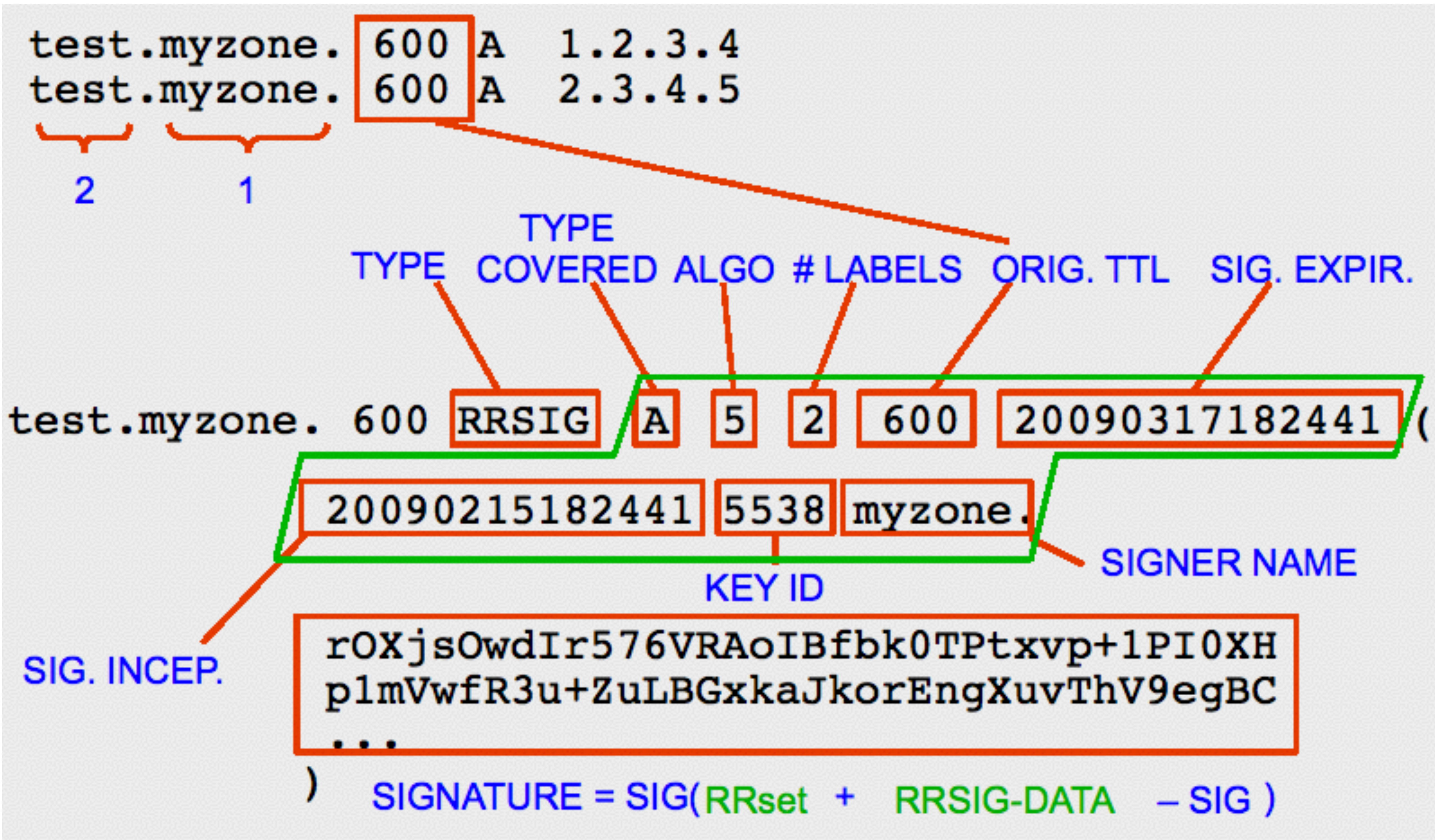
0 – reserved
1 – RSA/MD5 (deprecated)
2 – Diffie/Hellman
3 – DSA/SHA-1 (optional)
4 – reserved
5 – RSA/SHA-1 (mandatory in validator)
8 – RSA/SHA-256

RRSIG

- Resource Record SIGnature
- Digital signature of a set of records
    - Can be A, AAAA, MX, SOA



RRSIG (cont.)


test.myzone. 600 A 1.2.3.4
test.myzone. 600 A 2.3.4.5
2
1
TYPE
TYPE COVERED
ALGO
LABELS
ORIG. TTL
SIG. EXPIR.
test.myzone. 600 RRSIG A 5 2 600 20090317182441 (
20090215182441 5538 myzone.
SIGNER NAME
KEY ID
SIG. INCEP.
rOXjsOwdIr576VRAoIBfbk0TPtxvp+1PI0XH
p1mVwfR3u+ZuLBGxkaJkorEngXuvThV9egBC
...
)
SIGNATURE = SIG(RRset + RRSIG-DATA – SIG)

Delegation Signer

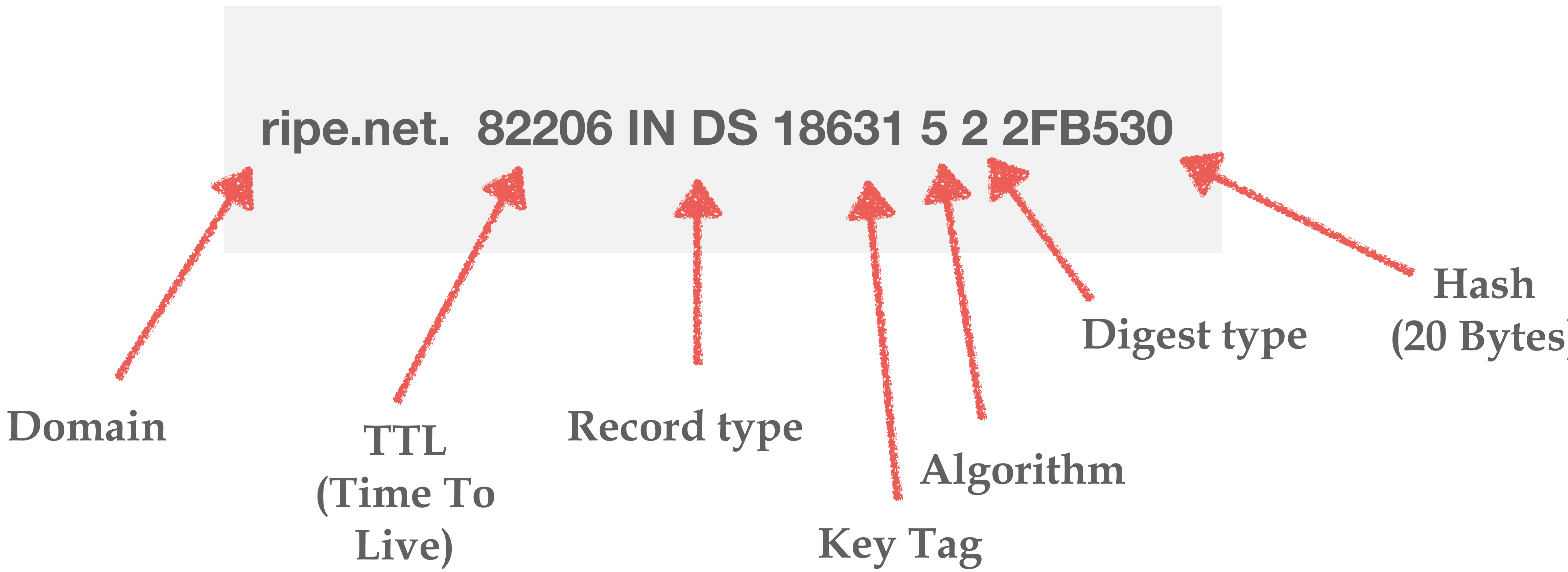
- The child's DNSKEY* is -> hashed
  - hash function
- The hash of the key is signed by the parent's DNSKEY
  - and included in the parents zone file
- Repeat
- Chain of trust

Delegation Signer (DS)

- Delegation Signer (DS) RR indicates that:
  - delegated zone is digitally signed
  - indicated key is used for the delegated zone
- Parent is authoratitive for the DS of the child's zone
  - Not for the NS record delegating the child's zone!
  - DS should not be in the child's zone

DS

- Delegation Signer
- Contains hash of the (KSK) DNSKEY
- To be published in the parent zone of DNS chain



NSEC

- “Next SECure” record
- When the answer is a non existent record, NSEC provides the closest record in the set that would resolve, in alphabetical order
- Side Effect: allows discovery of zone contents

NSEC internals

Query for:

troubleshoot.dnssec-course.net

↓

Non-existent

↓

Next record:

www.dnssec-course.net

NSEC RDATA

- Points to the next domain name in the zone
  - also lists what are all the existing RRs for “name”
  - NSEC record for last name “wraps around” to first name in zone
- N*32 bit type bit map
- Used for authenticated denial-of-existence of data
  - authenticated non-existence of TYPEs and labels
- Example:
- www.ripe.net. 3600 IN NSEC ripe.net. A RRSIG NSEC

NSEC Records

- Proof of non-existence
- If server's response is NXDOMAIN:
  - One/more NSEC RRs show: name or wildcard expansion does not exist
- If server's response is NOERROR:
  - And empty answer section
  - The NSEC proves QTYPE did not exist
- More than one NSEC may be needed
  - Wildcards
- NSEC records generated by tools
- Tools also order the zone

NSEC Walk

- NSEC records allow for zone enumeration
- Privacy not a requirement
- Zone enumeration is a deployment barrier

NSEC3

- Same as NSEC
- But hashes all names to avoid zone discovery
- Hashed names are ordered

```
DRVR6JA3E4VO5UIPOFAO5OEEVV2U4T1K.dnssec-course.net. 3600 IN
NSEC3 1 0 10 03F92714 GJPS66MS4J1N6TIIJ4CL58TS9GQ2KRJ0 A RRSIG
```

?

Setting up a Secure Zone

Step-by-step

Securing a Zone: Step 1

- Generate keypair
  - Include public key (DNSKEY) in zone file
  - dnssec-keygen tool comes with BIND

Generating Keys

- dnssec-keygen to generate keys

```
dnssec-keygen -a alg -b bits -n type [options] name
```

- algorithm: RSASHA1
- Bitsize: depends on key function & paranoia level
- type:zone
- Name: zone you want to sign

- ‘-r /dev/urandom’ might be needed

Creating Keys


```
$dnssec-keygen -a RSASHA1 -b 1024 -n zone example.net.
   kexample.net.+005+20704
$
```

- **2 files are created:**
  - **Kexample.net.+005+20704.key**
    - contains the public key
    - should go into the zone file
  - **Kexample.net.+005+20704.private**
    - contains the private key

Signing a Zone: Step 2

- Sign your zone. It will:

- Sort the zone
- Insert:
  - **NSEC** records
  - **RRSIG** records (signature over each RRset)
  - **DS** records (optional)
- Generate key-set and ds-set files

Signing the Zone


```
dnssec-signzone [options] zonefile [ZSK's]
```

- If zonefile name is not zone name:
  - use –o <origin> option
- Signed zonefile is called “zonefilename.signed”
- Keyset & DS-set files are created as a bonus…
  - ready to go to parent

Securing a Zone: Step 3

- Publish signed zone
- Signed zone is regular zonefile format
  - With extra resource records
- Make sure all your servers are DNSSEC capable!

Securing a Zone: Step 4

- Configure forwarding resolver
- Test
- DNSSEC verification only done in resolver!

Verifying Resolving Name Server

- To verify the content of a zone:
  - Get the public (key signing) key and check that this key belongs to the zone owner

- Configure the keys you trust as secure entry points in named.conf

```
trusted-keys {
        "example.net." 256 3 1 "AQ...QQ==";
        };
```

Securing a Zone: Step 5

- Distribute your public key (DNSKEY)
  - To parent zone (key-set or ds-set can be used)
  - To everyone that wants/needs you as SEP
- Make sure to inform everyone of key rollovers!

?

Delegating Signing Authority

Chains of Trust

Using the DNS to Distribute Keys

- Secured islands make key distribution problematic

- Distributing keys through DNS:
  - Use one trusted key to establish authenticity of other keys
  - Building chains of trust from the root down
  - Parents need to sign the keys of their children

- Only the root key needed in ideal world
- Parents always delegate security to child

Key Problem

- Interaction with parent administratively expensive
  - Should only be done when needed
  - Bigger keys are better
- Signing zones should be fast
  - Memory restrictions
  - Space and time concerns
  - Smaller keys with short lifetimes are better

Key Functions

- Large keys are more secure
  - Can be used longer ✓
  - Large signatures => large zonefiles ✗
  - Signing and verifying computationally expensive ✗

- Small keys are fast
  - Small signatures ✓
  - Signing and verifying less expensive ✓
  - Short lifetime ✗

Key solution: More Than One Key

- RRsets are signed, not RRs
- DS points to specific key
  - Signature from that key over DNSKEY RRset transfers trust to all keys in DNSKEY RRset
- Key that DS points to only signs DNSKEY RRset
  - Key Signing Key (KSK)
- Other keys in DNSKEY RRset sign entire zone
  - Zone Signing Key (ZSK)

Key types

- Two types:
  - ZSK - Zone Signing Key
  - KSK - Key Signing Key
- Each zone has a private and public key pair of both the ZSK and the KSK

Zone Signing Key - ZSK

- Used to sign a zone
- Can be lower strength than the KSK
- No need to coordinate with parent zone if you want to change it

Key Signing Key - KSK

- Only signs the Resource Record Set containing DNSKEYs for a zone
- Used as the trust anchor
- Needs to be specified in the parent zone using DS (Delegation Signature) records

How it works

- ZSK signs the DNS data
- KSK signs the ZSK

- An attacker can only see one signature
  - Not enough data to perform brute force

- Keys can expire and be replaced
  - Makes it even more difficult to compute them

Initial Key Exchange

- **Child needs to:**
  - **Send key signing keyset to parent**
- **Parent needs to:**
  - **Check childs zone**
    - for DNSKEY & RRSIGs
  - **Verify if key can be trusted**
  - **Generate DS RR**

Walking the Chain of Trust

Locally Configured

```
              Trusted Key . 8907
```

(root) .

```
.             DNSKEY (…) 5TQ3s… (8907) ; KSK
              DNSKEY (…) lasE5… (2983) ; ZSK

              RRSIG  DNSKEY (…)  8907 .  69Hw9…

net.          DS   7834 3 1ab15…
              RRSIG   DS (…) . 2983
```

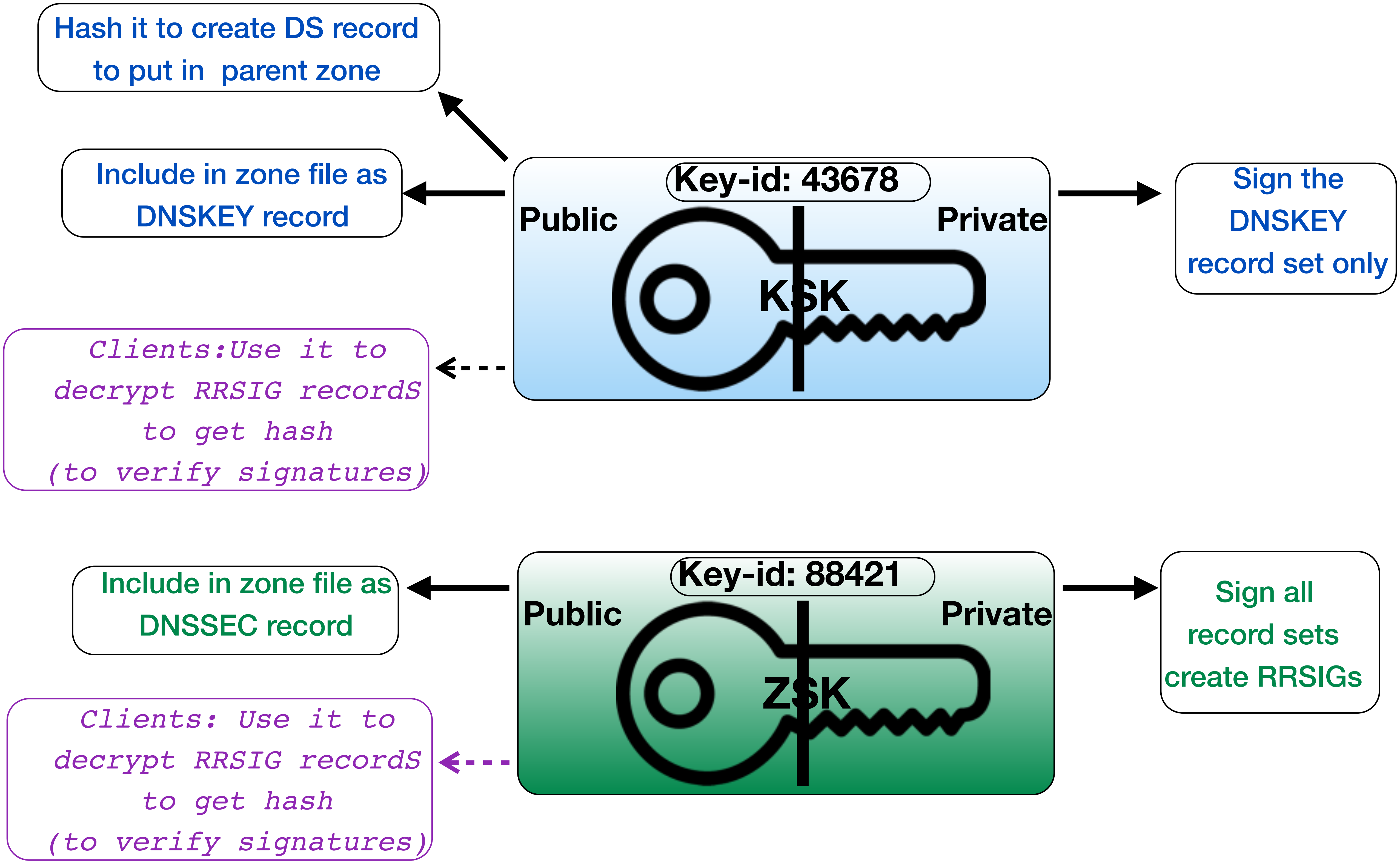
net.

```
net.          DNSKEY (…) q3dEw… (7834) ; KSK
              DNSKEY (…) 5TQ3s… (5612) ; ZSK

              RRSIG  DNSKEY (…)  7834 net.  cMas…

ripe.net.     DS   4252 3 1ab15…
              RRSIG  DS (…) net. 5612
```

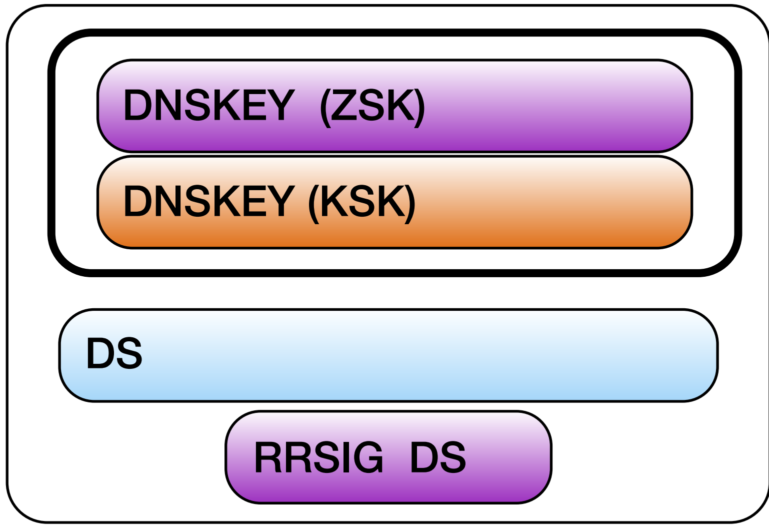
ripe.net.

```
ripe.net.     DNSKEY (…) rwx002…  (4252) ; KSK
              DNSKEY (…) sovP42…  (1111) ; ZSK

              RRSIG  DNSKEY (…) 4252 ripe.net.  5t...

www.ripe.net. A 193.0.0.202
              RRSIG  A  (…)  1111 ripe.net.  a3...
```

Keys


Hash it to create DS record to put in parent zone
Include in zone file as DNSKEY record
Key-id: 43678
Public
Private
KSK
Sign the DNSKEY record set only
Clients:Use it to decrypt RRSIG records to get hash (to verify signatures)
Include in zone file as DNSSEC record
Key-id: 88421
Public
Private
ZSK
Sign all record sets create RRSIGs
Clients: Use it to decrypt RRSIG records to get hash (to verify signatures)

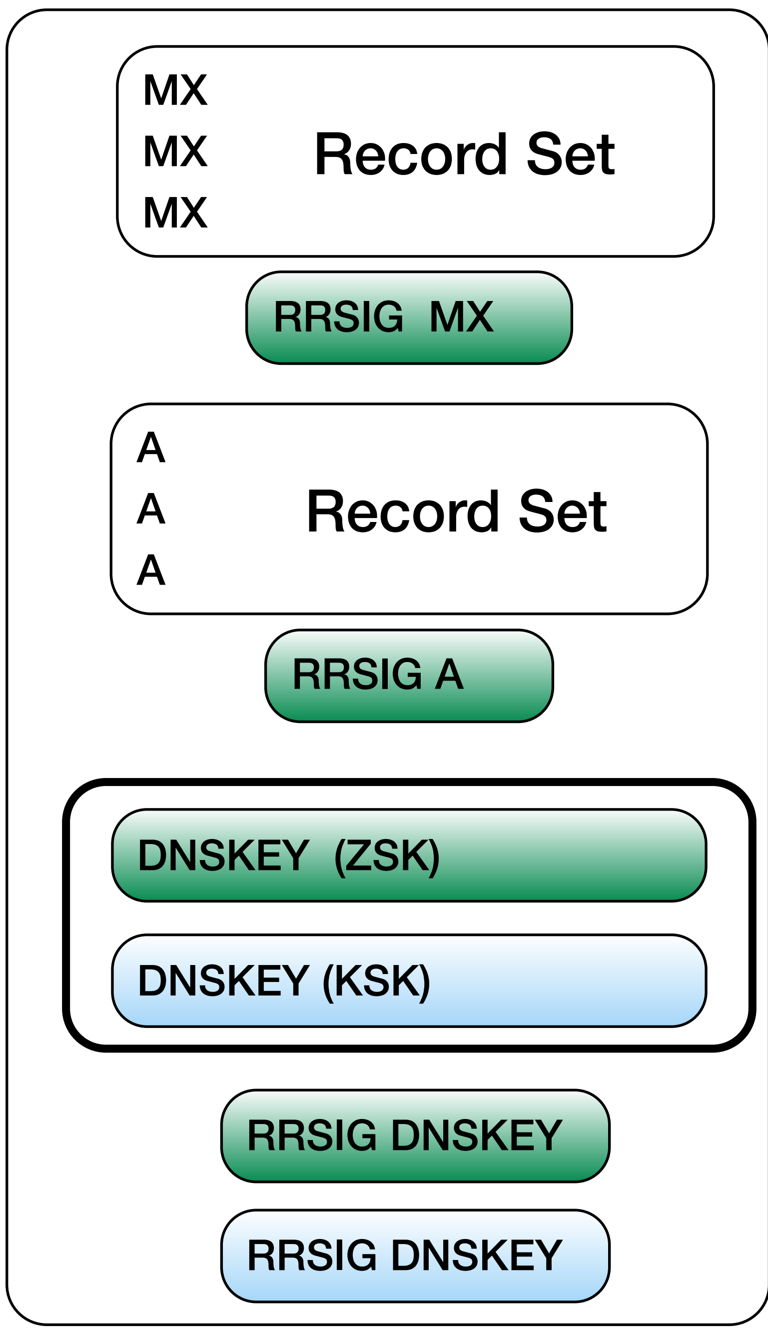
PARENT



Hash of child's (public) KSK

signed by Parent's (private) ZSK

CHILD



signed by (private) ZSK

signed by (private) ZSK

signed by (private) ZSK

signed by (private) KSK

Chain of Trust Verification, Summary

- Data in zone can be trusted if signed by a Zone-Signing-Key
- Zone-Signing-Keys can be trusted if signed by a Key-Signing-Key
- Key-Signing-Key can be trusted if pointed to by trusted DS record
- DS record can be trusted
  - if signed by the parents Zone-Signing-Key

    or
  - DS or DNSKEY records can be trusted if exchanged out-of-band and locally stored (Secure entry point)

Validation

- Performed through DS Records
- They hold a hash of the KSK used for the zone
- Needs to be entered at the tree-level above the one we want to validate

```
ripe.net.  82206 IN DS 18631 5 2
2FB530D881814008C209479B18FFA4B5342E93DC89ABB7DDF8D319EA
686D4385
```

DNS Lookaside Validation

- Generally called DLV
- Transition method while all the TLDs implement DNSSEC
- To be used when registrar/NIC does not support adding DS records
- Run by ISC using dlv.isc.org
- Use is not encouraged anymore

?

Demo

?

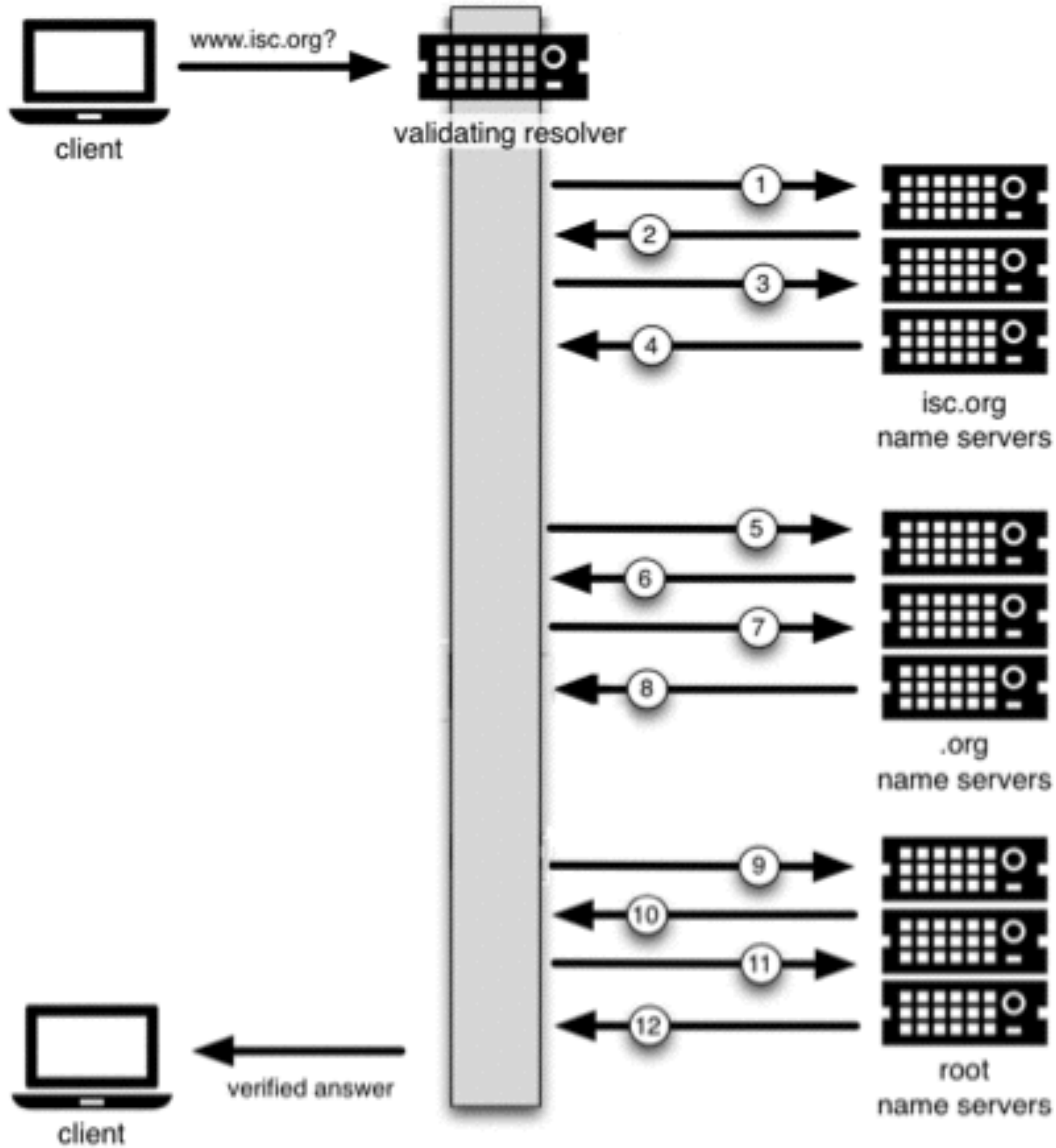
DNS Query with DNSSec Step-by-Step

Query and Validation

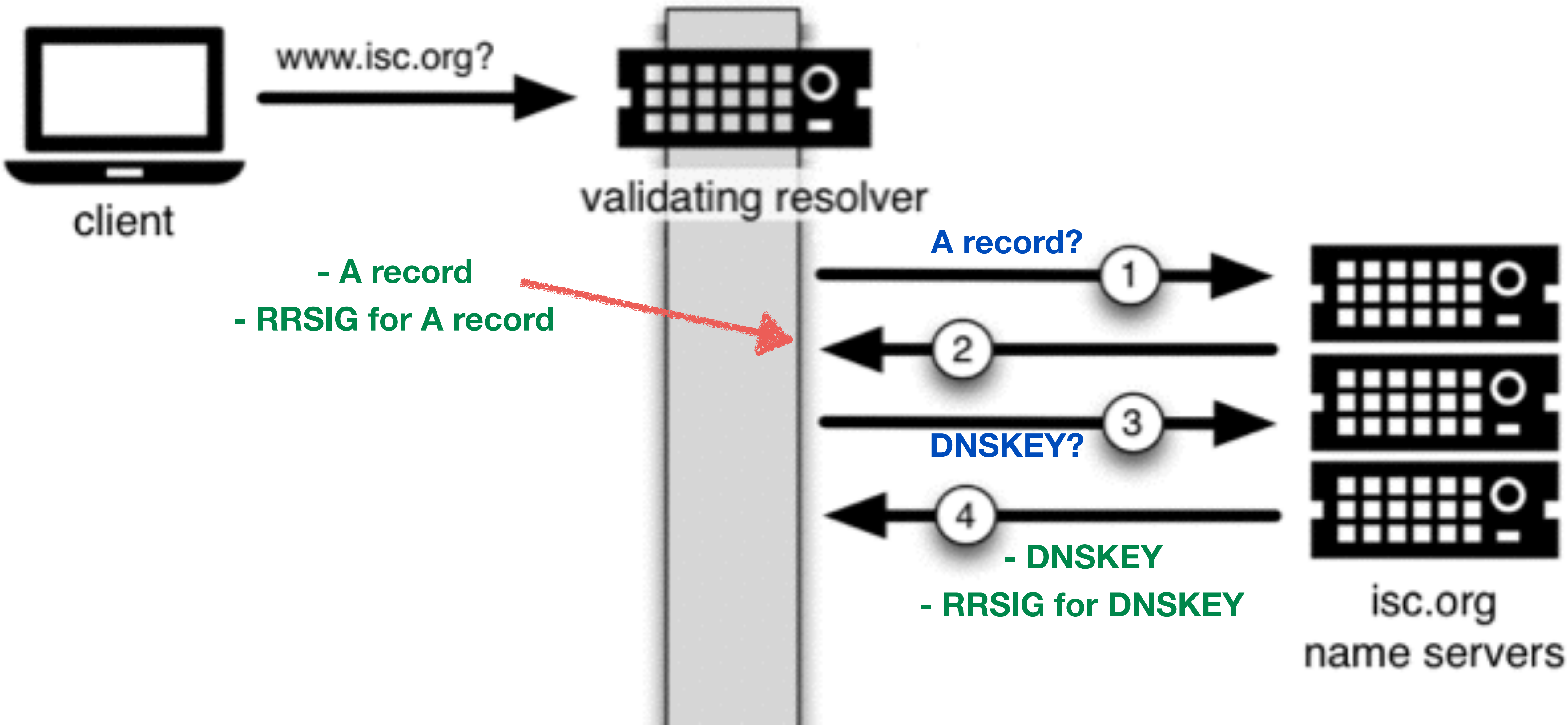
- **DNS Query goes from top down**
  - .
  - .org
  - isc.org
  - foo.isc.org

- **DNSSEC Validation goes bottom up**
  - .
  - .org
  - isc.org
  - foo.isc.org

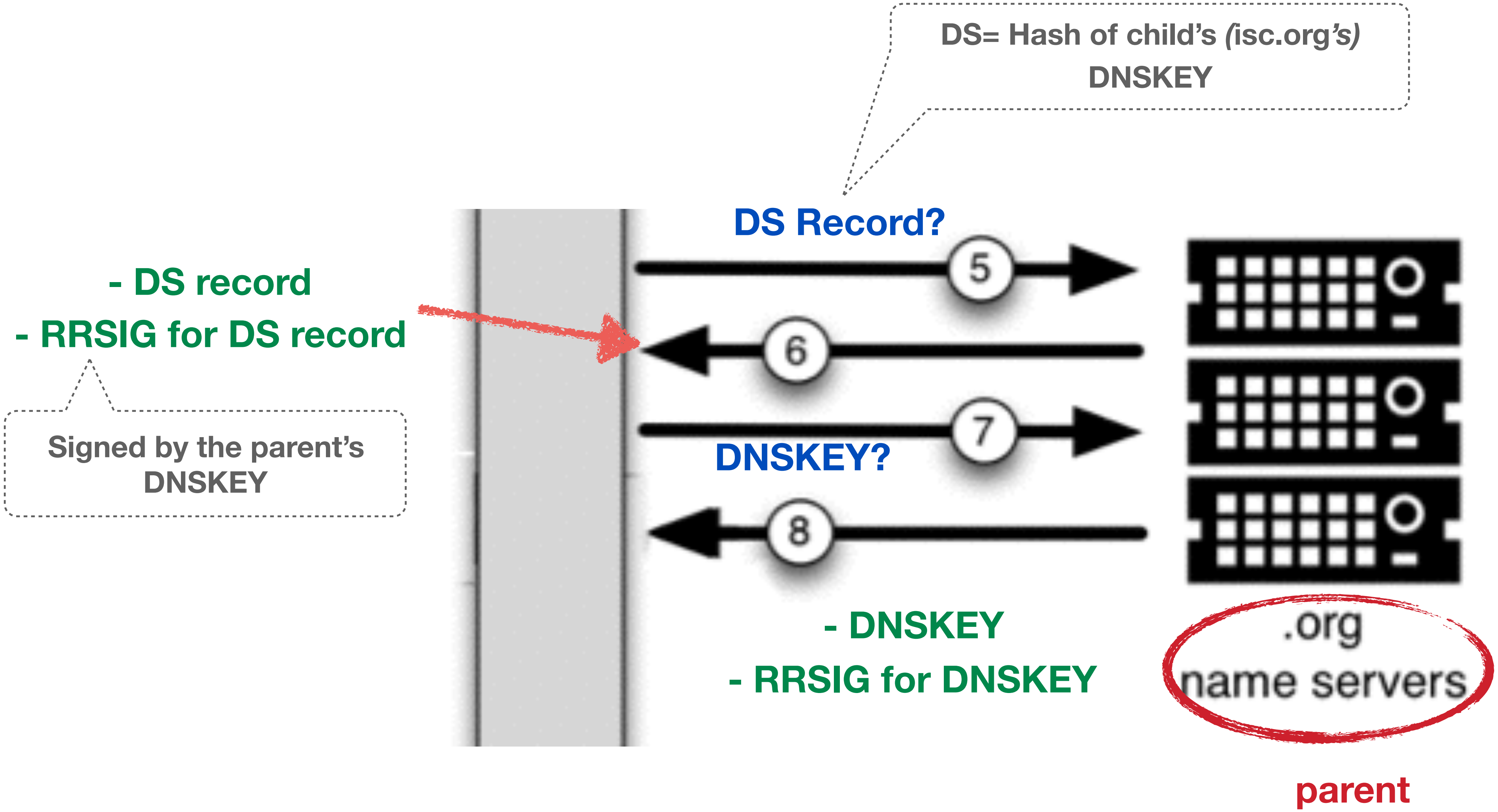
DNSEC Validation in 12 steps



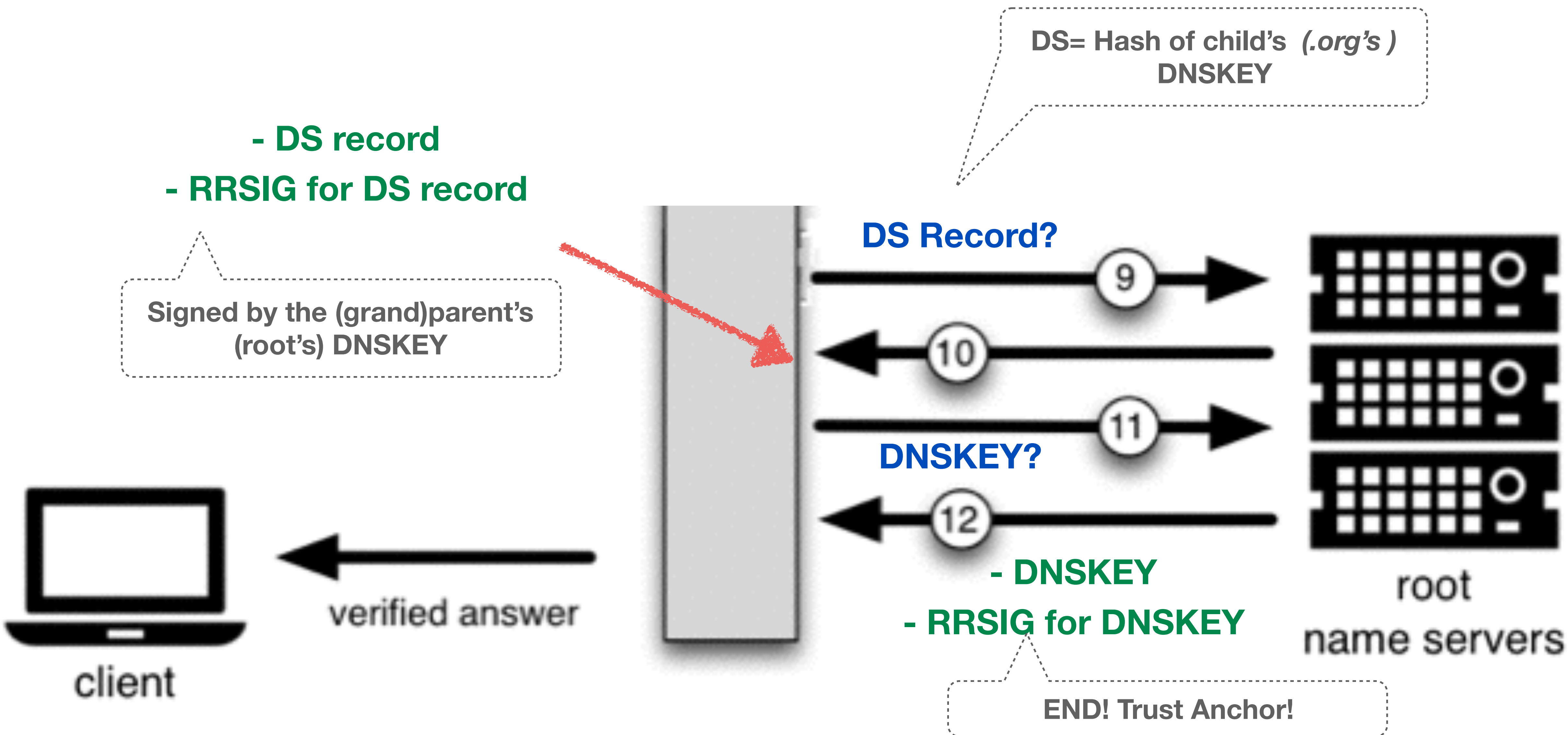
Part one: Querying the child



Part two: Querying the Parent



Querying the root



?

Flags and Scenarios

Flags

- Classical DNS
  - `qr` query response (A)
  - `rd` recursion desired (A) (Q)
  - `ra` recursion available (A)
  - `aa` authoritative answer (A)
- DNSSEC
  - `ad` authenticated data (A)
  - `cd` checking disabled (A) (Q)
  - `do` DNSSEC okay (i.e.: "do" DNSSEC) (A) (Q)

dig

- DNS diagnostic
- Can simulate DNS queries and more
- flags:
    - +dnssec
    - +cdflag
    - +multiline

drill

- **Reimplementation of dig based on LDNS**
- **Can do everything dig can ... sometimes faster!**
- **Friendlier implementation of flags**
  - DO ... do
  - RD ... rd
  - Upper case: option on
  - Lower case: option off

dig

- `dig www.isc.org`
- `dig www.isc.org A`
- `dig @192.168.1.7 www.isc.org A`

same

which recursive server to use

- **DNSSec validated answers?**
  - depends whether Server and Recursive Resolver configured for DNSSec

dig example

If DNSSec is disabled:

No DNSSEC validation on recursive resolver

DNSSEC enabled on server

Q

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20416
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```

A

recursion was desired

recursion was available

dig example

If DNSSec is disabled:

No DNSSEC validation on recursive resolver

DNSSEC enabled on server

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20416
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```

Q

A

recursion was desired

recursion was available

dig example

If DNSSec is disabled: (whole answer)

No DNSSEC validation on recursive resolver

DNSSEC enabled on server

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20416
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isc.org.       IN   A

;; ANSWER SECTION:
www.isc.org.     60   IN   A 149.20.64.69
```

dig example

If DNSSec is disabled: (whole answer)

No DNSSEC validation on recursive resolver

DNSSEC enabled on server

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20416
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isc.org.      IN   A

;; ANSWER SECTION:
www.isc.org.     60   IN   A 149.20.64.69
```

dig example

If DNSSec is enabled:

DNSSEC validation on recursive resolver

DNSSEC enabled on server

show DNSSec data (RRSIG)

sets the DO flag

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline

; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32472
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
```

Authenticated (DNSSec validated) data

“I am showing you the DNSSec records (RRSIG)”

How would the flags and answers be different without the +dnnsec flag?

dig example

If DNSSec is enabled: (whole answer)

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline

; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32472
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.isc.org.    IN A

;; ANSWER SECTION:
www.isc.org.     4 IN A  149.20.64.69
www.isc.org.     4 IN RRSIG A 5 3 60 (
        20141029233238 20140929233238 4521 isc.org.
        DX5BaGVd4KzU2AIH911Kar/UmdmkARyPhJVLr0oyPZaq
        5zoobGqFI4efvzL0mcpncuUg3BSU5Q48WdBu92xinMdb
        E75zl+adgEBOsFgFQR/zqM3myt/8SngWm4+TQ3XFh9eN
        jqExHZZuZ268Ntlxqgf9OmKRRv8X8YigaPShuyU= )

;; Query time: 3 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 03 16:40:04 CST 2014
;; MSG SIZE  rcvd: 223
```

shows RRSIG record

dig example 2

- Let’s use dig to examine a domain with “broken” DNSSec
- Validation NOT enabled on recursive server

dig example 2 (cont.)

No DNSSEC validation on recursive server

DNSSEC broken on server

```
$ dig @192.168.1.7 www.dnssec-failed.org. A

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28878
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.dnssec-failed.org.    IN  A

;; ANSWER SECTION:
www.dnssec-failed.org.  7200  IN  A 68.87.109.242
www.dnssec-failed.org.  7200  IN  A 69.252.193.191

;; Query time: 955 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 17 07:42:50 CST 2014
;; MSG SIZE  rcvd: 82
```

dig example 2 (cont.)

No DNSSEC validation on recursive server

DNSSEC broken on server

```
$ dig @192.168.1.7 www.dnssec-failed.org. A

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28878
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.dnssec-failed.org.    IN   A

;; ANSWER SECTION:
www.dnssec-failed.org.   7200   IN   A 68.87.109.242
www.dnssec-failed.org.   7200   IN   A 69.252.193.191

;; Query time: 955 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 17 07:42:50 CST 2014
;; MSG SIZE  rcvd: 82
```

dig example 2 (cont.)

DNSSEC validation on recursive server

DNSSEC broken on server

```
$ dig @192.168.1.7 www.dnssec-failed.org. A

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 46592
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.dnssec-failed.org.    IN   A

;; Query time: 2435 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 17 07:44:56 CST 2014
;; MSG SIZE  rcvd: 50
```

dig example 2 (cont.)

DNSSEC validation on recursive server

DNSSEC broken on server

```
$ dig @192.168.1.7 www.dnssec-failed.org. A

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 46592
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.dnssec-failed.org.    IN   A

;; Query time: 2435 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 17 07:44:56 CST 2014
;; MSG SIZE  rcvd: 50
```

dig example 2 (diagnostics)

- All DNSSec validation failures -> “SERVFAIL”
  - how do I know failure because of validation?
  - +cd flag!
  - “checking disabled”

dig example 2 (diagnostics)

DNSSEC validation on recursive server

DNSSEC broken on server

CHECKING DISABLED

```
$ dig @192.168.1.7 www.isc.org. A +cd

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.isc.org. A +cd
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33590
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isc.org.       IN   A

;; ANSWER SECTION:
www.isc.org.     30   IN   A 149.20.64.69
```

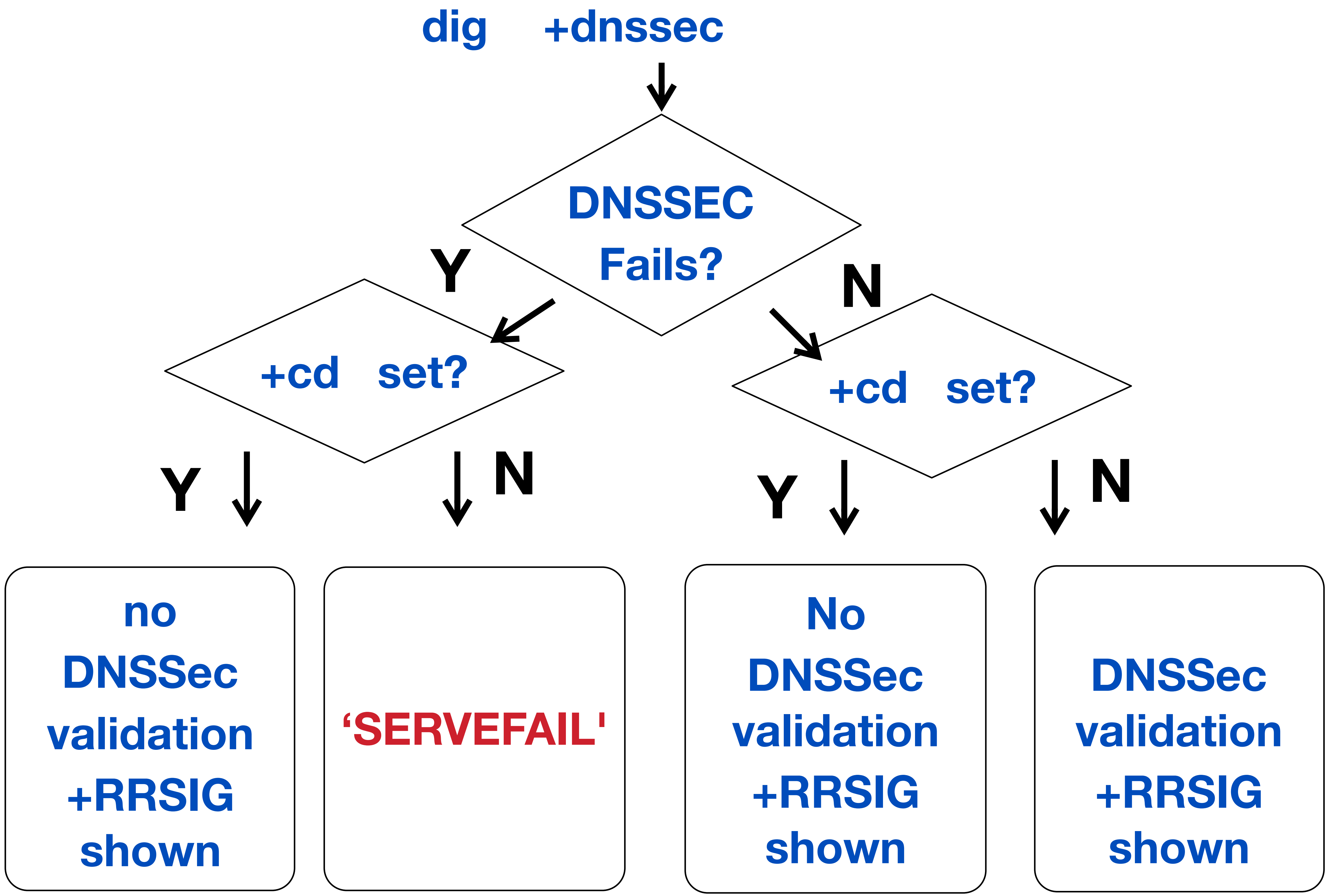
dig example 2 (diagnostics)

DNSSEC validation on recursive server

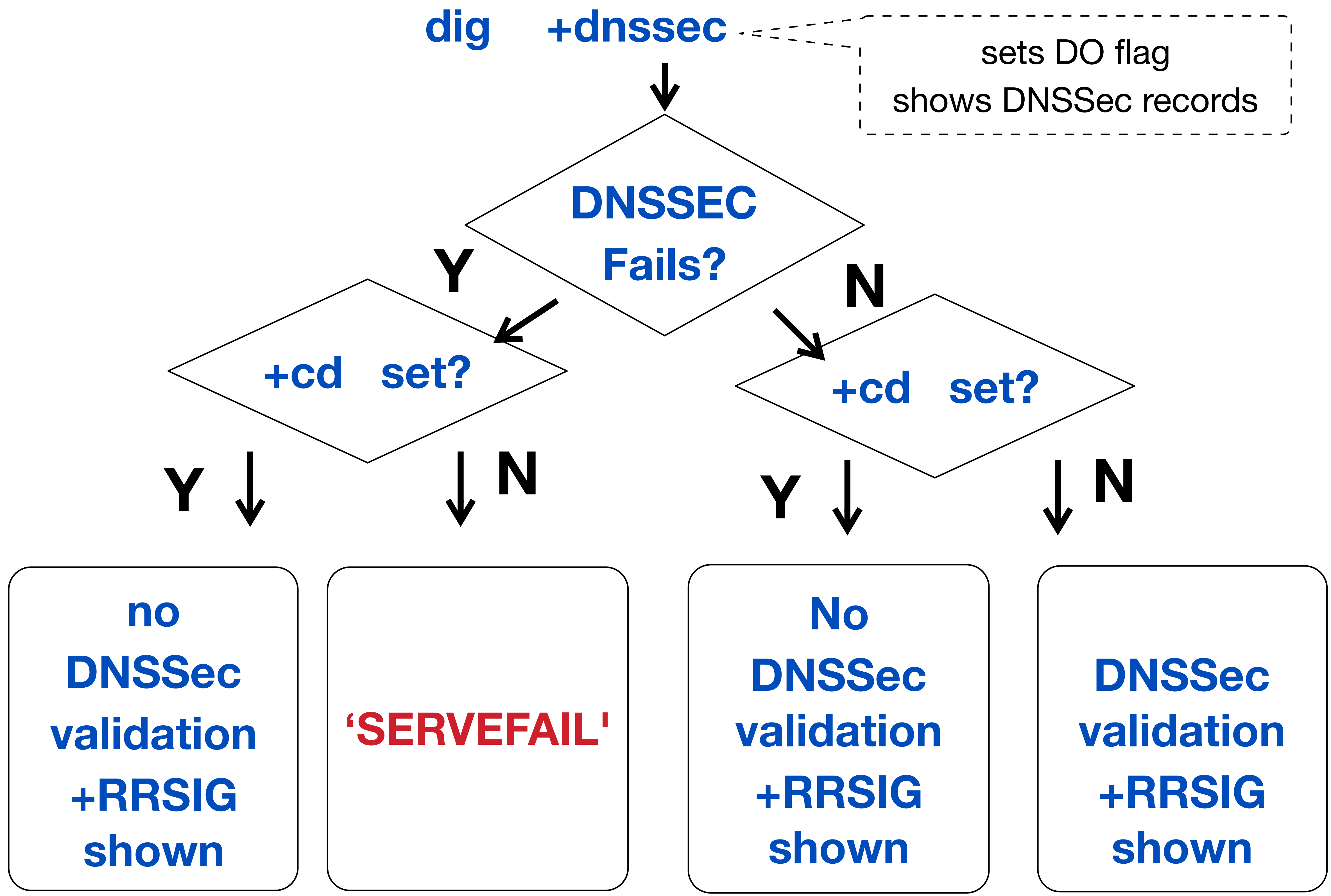
DNSSEC broken on server

CHECKING DISABLED

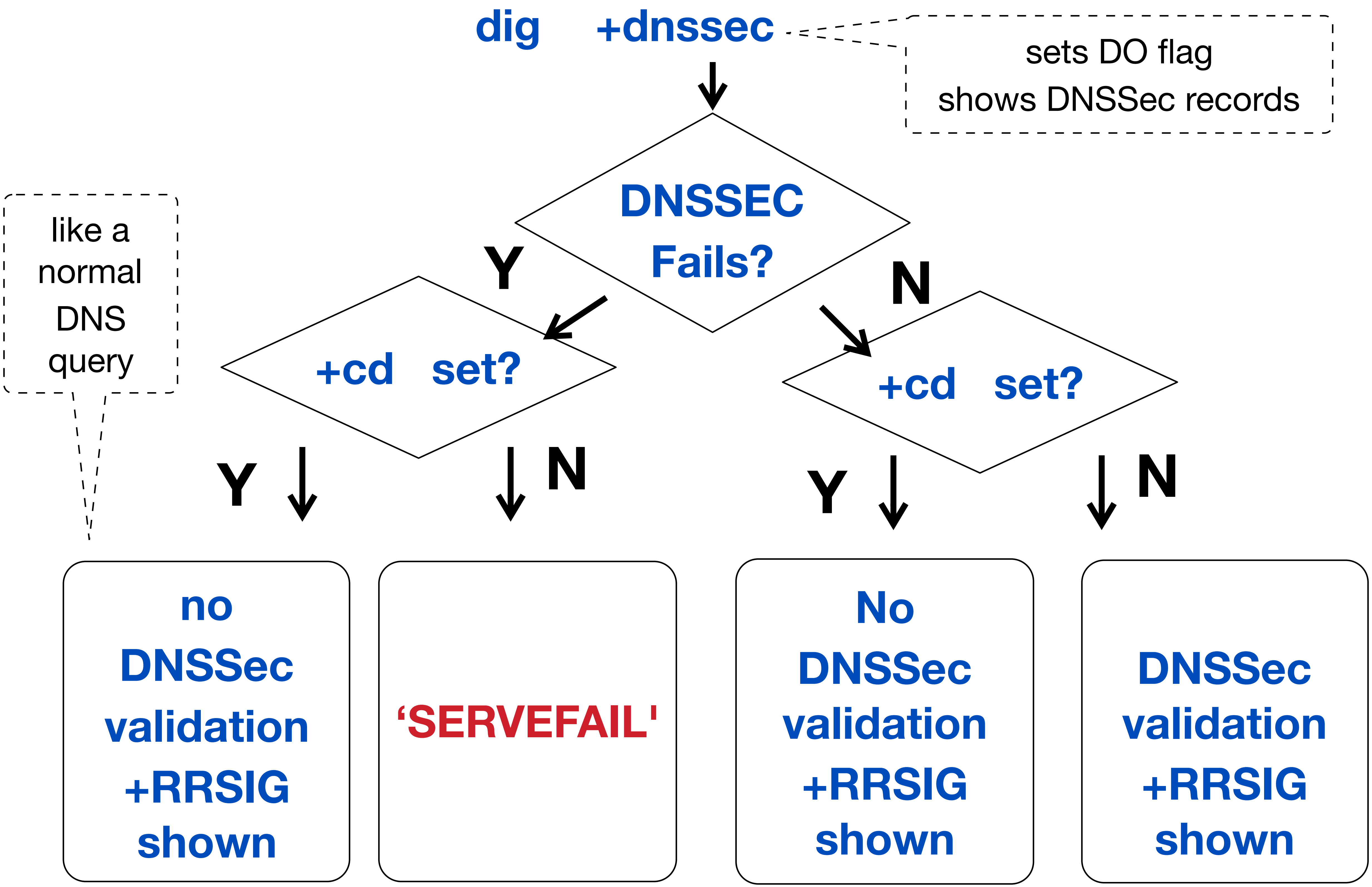
```
$ dig @192.168.1.7 www.isc.org. A +cd

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.isc.org. A +cd
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33590
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isc.org.        IN   A

;; ANSWER SECTION:
www.isc.org.     30   IN   A 149.20.64.69
```



dig +dnssec
DNSSEC Fails?
Y
N
+cd set?
+cd set?
Y
N
Y
N
no DNSSec validation +RRSIG shown
'SERVEFAIL'
No DNSSec validation +RRSIG shown
DNSSec validation +RRSIG shown


dig +dnssec
sets DO flag
shows DNSSec records
DNSSEC
Fails?
Y
N
+cd set?
+cd set?
Y
N
Y
N
no
DNSSec
validation
+RRSIG
shown
'SERVEFAIL'
No
DNSSec
validation
+RRSIG
shown
DNSSec
validation
+RRSIG
shown

+dnssec


dig +dnssec
sets DO flag
shows DNSSec records
DNSSEC Fails?
Y
N
+cd set?
+cd set?
like a normal DNS query
Y
N
Y
N
no DNSSec validation +RRSIG shown
'SERVEFAIL'
No DNSSec validation +RRSIG shown
DNSSec validation +RRSIG shown

?

Setting up a Recursive Resolver

(Signing Step-by-Step)

Setting up a Recursive Resolver

- in named.conf:

```
options {
     dnssec-validation auto;
};
```

done!

- options:
  - **yes** (trust anchor manually configured)
  - **no** (validation disabled= simple DNS resolver)
  - **auto** (default trust anchor in BIND = root)

yes vs auto for Recursive Resolver

- yes: ‘trusted-keys’ in named.conf
- auto: BIND keeps keys up-to-date automatically

?

Setting up an Authoritative Server

Setting up an Authoritative Server

1. Generate Keys
2. Reconfigure BIND

    - add to named.conf info about keys

3. Reload named
4. Upload DS to parent zone
5. Check :

    - keys in zone
    - signatures in zone
    - DS in parent's zone

1. Generate keys

- in /etc/bind/keys/example.com:

Directory where keys are stored

```
# mkdir -p /etc/bind/keys/example.com
# cd /etc/bind/keys/example.com
# dnssec-keygen -a RSASHA256 -b 1024 example.com
Generating key pair...++++++ .............++++++
Kexample.com.+008+17694
# dnssec-keygen -a RSASHA256 -b 2048 -f KSK example.com
Generating key pair.......................+++ .......
Kexample.com.+008+06817
```

ZSK key

KSK key

Algorithm

Number of bits

1. Generate keys (cont.)

- 4 files in /etc/bind/keys/example.com:
  - Kexample.com.+008+06817.key
  - Kexample.com.+008+06817.private
  - Kexample.com.+008+17694.key
  - Kexample.com.+008+17694.private
- looking inside the key file you can tell if ZSK or KSK

1 Generate keys (cont.)


```
# cat Kexample.com.+008+06817.key
; This is a key-signing key, keyid 6817, for example.com.
; Created: 20141120094612 (Thu Nov 20 17:46:12 2014)
; Publish: 20141120094612 (Thu Nov 20 17:46:12 2014)
; Activate: 20141120094612 (Thu Nov 20 17:46:12 2014)
example.com. IN DNSKEY 257 3 8 AwEAAcWDps...lM3NRn/G/R
# cat Kexample.com.+008+17694.key
; This is a zone-signing key, keyid 17694, for example.com.
; Created: 20141120094536 (Thu Nov 20 17:45:36 2014)
; Publish: 20141120094536 (Thu Nov 20 17:45:36 2014)
; Activate: 20141120094536 (Thu Nov 20 17:45:36 2014)
example.com. IN DNSKEY 256 3 8 AwEAAcjGaU...zuu55If5
```

2. Reconfigure BIND

- Add extra lines to 'named.conf' file
  - /etc/bind/named.conf

```
options {
     directory "/etc/bind";
     recursion no;
     minimal-responses yes;
};

zone "example.com" IN {
     type master;
     file "db/example.com.db";
     key-directory "keys/example.com";
     inline-signing yes;
     auto-dnssec maintain;
};
```

created a subfolder 'example.com" for that zone's keys

where named should look for the public and private DNSSec key files

BIND keeps unsigned zone and creates signed zone

next slide

2. Reconfigure BIND (cont)

- auto-dnssec …
  - **off** default. Key management manually
  - **allow** allows uploading keys and resigning the zone when user runs rndc-sign [zone-name]
  - **maintain** same as “allow” +automatically adjusts the keys on schedule (key’s timing metadata)

3. Reload


```
# rndc reload
server reload successful
```

4.Upload DS to parent zone

- Before uploading, make sure newly signed zone has propagated to all your name servers.
- Format to upload depends on parent server

use "dnssec-dnsfromkey" tool

1. DS Record Format: `example.com.IN DS 6817 8 1 59194A835ACD78D25D538D5F35CA043A8F3F4446`
2. DNSKEY Format: `example.com.172800 IN DNSKEY 257 3 8 (AwEAAcjGaU...zuu55If5) ;key id =06817`
3. Trusted Key Format: `"example.com." 257 3 8 "AwEAAcjGaU...zuu55If5";`

Zone file: before and after DNSSec

- See handout
  - BIND DNSSec Guide pages 25-27
  - http://users.isc.org/~jreed/dnssec-guide/dnssec-guide.html

New files created by signing


```
# cd /etc/bind/db
# ls
example.com.db  example.com.db.jbk  example.com.db.signed  example.com.db.signed.jnl
```

original zone file — transient file — **signed zone file** — journal file

5. Checking if your zone is signed correctly

a. Look for DNSKEYs in zone

b. Look for signatures (RRSIG) in zone

c. Check the parent

d. External testing tools

5.a. Look for DNSKEYs in your Zone


```
$ dig @192.168.1.13 example.com. DNSKEY +multiline +noall +answer

; <<>> DiG 9.10.1 <<>> @192.168.1.13 example.com. DNSKEY +multiline +noall +answer
; (1 server found)
;; global options: +cmd
example.com.     300 IN DNSKEY 256 3 8 (
        AwEAAclob7q+ccvDwaTVuMMZddGIynWyMwaZlhFrU6cC
        0qknWoPpkq0gIwTrYf3DJY+eIKPVHxrM+o2AoRIVhubG
        jfvlbT5wTYrawZstS84ejCQ+ehA+8DxKyeWUEzW0ZMBe
        OhyeG0cuQVK/p6Z1E096JLu0DjgbabLspequkw4M+HT7
        ) ; ZSK; alg = RSASHA256; key id = 57009
example.com.     300 IN DNSKEY 257 3 8 (
        AwEAAdQ2ctHx8VmryndiOgpcnXPdj3NwxMeUvAre6uYI
        5KELlFJUghTHrz+/CzEc8CXG8wwQ4ZvAey0FGV2nJAFC
        ENMxoRiCz0oSiQQxryNhACd3RnE2/D7G+Shw1OM6w53E
        wUJ/lsgu5UevSxFC+eA3fKeL3TWR44PH4iJQp9QmfW5v
        7qG8Sic/HQvBGBdOGfFtHAl0a4jDPBi57imS4YsHcUYD
        9bsWmhYWSHJKZ66+JnTiMS0nQM69YwBF43QfDKurs5R6
        qPUDiBlaMCzSxmlaBU6fsI1Mu/yIU8wlewy26a42rUTU
        rPBC3Oa/zf9VQ8kpUrMZgJ7LEAA4xmR+qwWDh6U=
        ) ; KSK; alg = RSASHA256; key id = 28267
```

ZSK key

kSK key

+multiline shows extra info about the keys

5.b. Look for Signatures in Your Zone

```
$ dig @192.168.1.13 example.com. SOA +dnssec +multiline

; <<>> DiG 9.10.1 <<>> @192.168.1.13 example.com. SOA +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31466
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;example.com.   IN SOA

;; ANSWER SECTION:
example.com.    300 IN SOA ns1.example.com. dnsadmin.example.com. (
        2014102111 ; serial
        10800      ; refresh (3 hours)
        1080       ; retry (18 minutes)
        2419200    ; expire (4 weeks)
        900        ; minimum (15 minutes)
        )
example.com.    300 IN RRSIG SOA 8 2 300 (
        20141121122105 20141022112105 57009 example.com.
        NqPGNLkUs40Lg/qq7Fv+bgyCwVB4s9PsHQOK6p9ZWWk3
        36z2Qz2WjM+Q19SlVBAPux9jijvcRcjGb6KREuxER9uX
        wdVeiGx9a4X+PaO3qTqdkiXuGS2XkK1kBm1CgwhVHTYV
        /nxVPsskU4/mpeUoFVjMnT49JkVJmgck63esPFU= )
```

with DNSSec every record set…

…must be signed and have its RRSIG

5.c. Check the parent

it will look up the DS record in the parent ".com" zone

```
$ dig example.com. DS

; <<>> DiG 9.10.1 <<>> example.com. DS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49949
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.com.       IN  DS

;; ANSWER SECTION:
example.com.   61179 IN  DS   28267 8 1 66D47CE4B4F551BE5EDA43AC5F3109E8C98E2FAE
example.com.   61179 IN  DS   28267 8 2 71 ↩
    D9335416B7132519190A95685E18CBF478DCF4CA98867062777938F8FEAB89
```

always 2 types of hashes

5.d. External Testing Tools

1. Verisign Labs DNSSEC Debugger: http://dnssec-debugger.verisignlabs.com/
2. DNSViz: http://dnsviz.net/
3. Sec Spider: http://secspider.cs.ucla.edu/

?

Key Roll-Over

Why roll keys?

- Limit effects of key compromise
- Administrative policy on key lifetime
- Enable quick replacement of keys in the event of compromise

Key rollover considerations

- Keeping the chain of trust intact
- Administrative overhead
- Interaction with parent zone operators

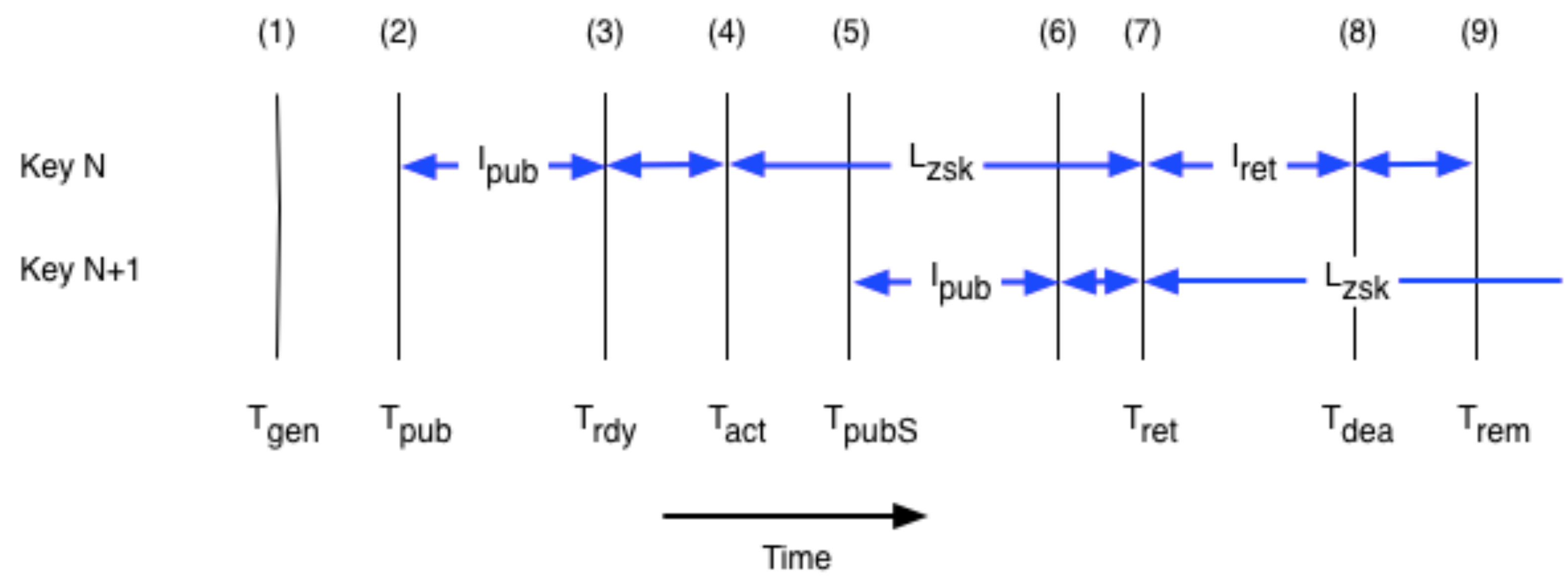
Methods of rolling keys

- Easier than it looks!
- RFC 6781
- `draft-ietf-dnsop-dnssec-key-timing-06`

ZSK rollover: pre-publication

- New key is introduced into the DNSKEY RRset
- After enough time to ensure that any cached DNSKEY RRsets contain both keys, the zone is signed using the new key and the old signatures are removed
- When all signatures created with the old key have expired from caches, the old key is removed

ZSK rollover: pre-publication

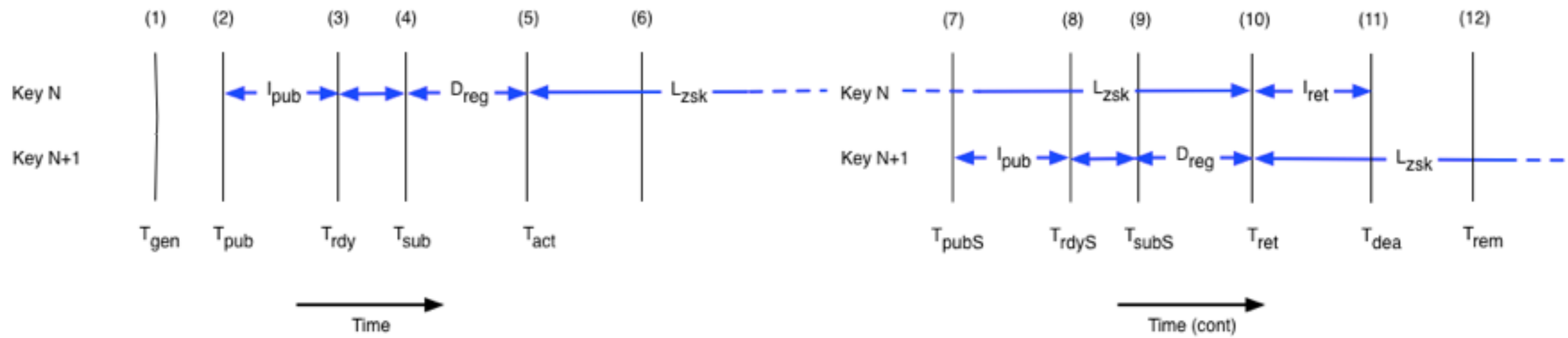


- First key: Ipub = Dprp + min(TTLsoa, SOAmin)
- Future keys: Ipub = Dprp + TTLkey
- TpubS <= Tact + Lzsk - Ipub
- Iret = Dsgn + Dprp + TTLsig

KSK rollover: double signature

- New key is introduced and used to sign the zone
- The old key and signatures are retained
- Once all caches are aware of the new DNSKEY and RRSIGs created with it, the old DNSKEY and RRSIGs can be removed from the zone
- At that point the DS records can simply be replaced in the parent zone

KSK rollover: double signature



- Ipub = Dprp + TTLkey
- TpubS <= Tact + Lksk - Dreg - Ipub
- Iret = DprpP + TTLds

Manual key rollover

- High administrative burden
- Only feasible if keys have a very long lifetime
- Possible tool: `zonesigner` (evil Perl script)

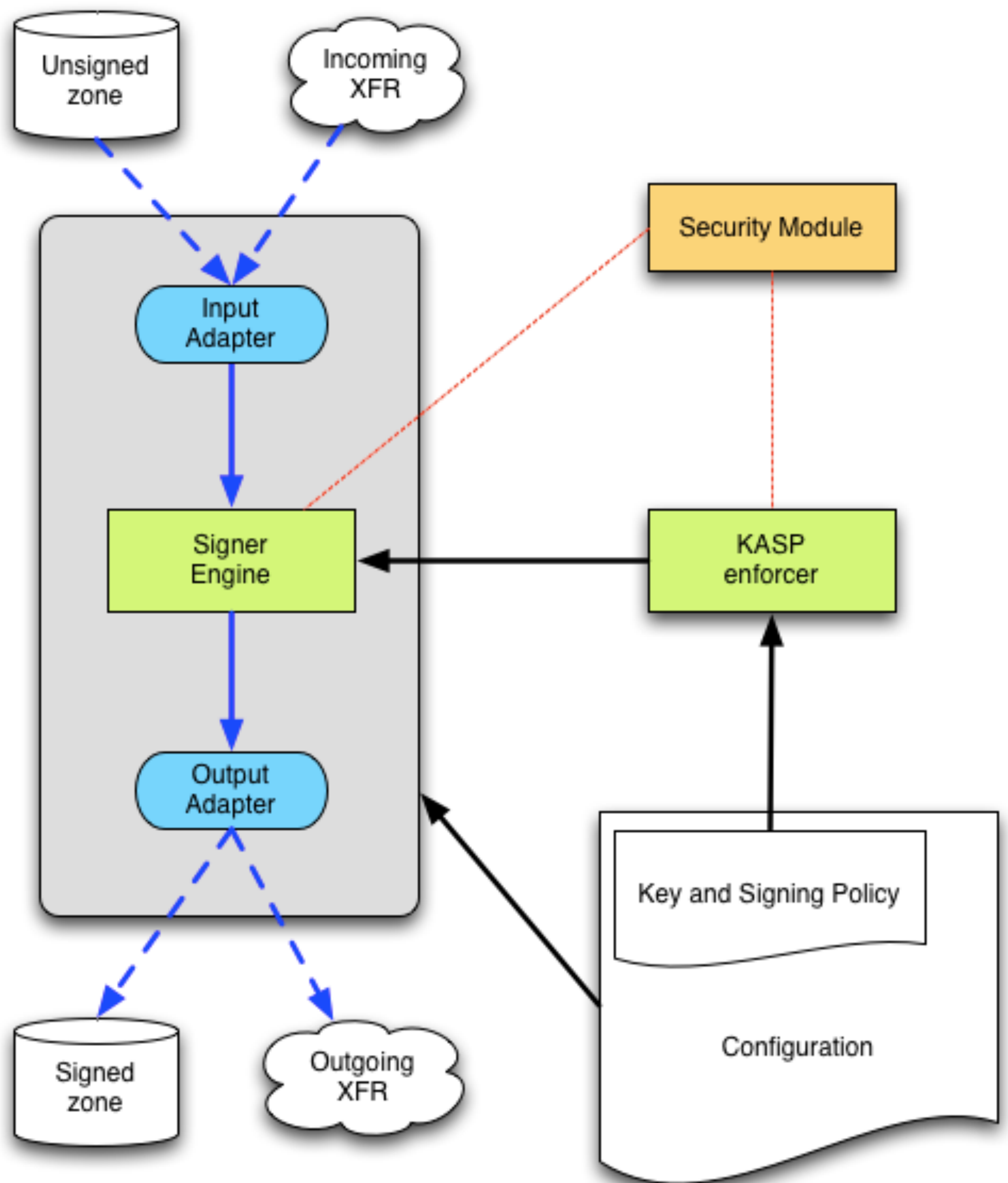
Automatic key rollover with BIND

- Introduced in BIND 9.7
- Affectionately known as "DNSSEC for Humans"
- dnssec-signzone –S example.com
- New managed-keys statement
- Explained in the BIND DNSSEC Guide PDF

OpenDNSSEC: rollovers and more!

- Designed as a "bump in the wire" solution
- Advanced key management options
- Rich description of key management policies

Overview of OpenDNSSEC



?

DAY FOUR

Overview Day Four

1. Recap and Lab exercises

Recap and Lab Exercises

Recap and Lab Exercises

- Recap and lab exercises

Useful DNSSec Guides

- http://users.isc.org/~jreed/dnssec-guide/dnssec-guide.html
- http://www.nlnetlabs.nl/publications/dnssec_howto/
- https://www.michaelwlucas.com/nonfiction/dnssec-mastery

Appendix:
Quizzes

Quiz 1

Question 1

What are some other names for "recursive nameserver"?

Question 1

What are some other names for "recursive nameserver"?

- Resolving cache
- Caching nameserver
- Recursive resolver
- nameserver
- DNS server
- validating resolver
- ...

Question 2

How would you look up the PTR record for 172.19.20.1?

Question 2

How would you look up the PTR record for 172.19.20.1?

1. Flip the address over: 1.20.19.172
2. Stick it in front of in-addr.arpa: 1.20.19.172.in-addr.arpa
3. Ask dig to query it: dig 1.20.19.172.in-addr.arpa ptr
4. (shortcut! dig -x 172.19.20.1)

Question 3

dk
news.dk
gov.dk

Admin of dk is administering gov.dk too.

Q0: How many domains and zones do you see?
Q1: Name the zones
Q2: Name domains
Q3: for which domain(s) are NS records used in dk?

3. Answers

dk
news.dk
gov.dk

Admin of dk is administering gov.dk too.

Q0: How many domains and zones do you see?
3 domains and 2 zones
Q1: Name the zones
dk (including gov.dk), news.dk
Q2: Name domains
dk, news.dk, go.dk
Q3: for which domain(s) are NS record used in dk
news.dk

Question 4

What do we understand by "negative time to live"?

Question 4

What do we understand by "negative time to live"?

The time before we forget we were told that something isn't there

Question 5

(Bonus points) Why do you think the negative TTL is not in each resource?

Question 5

(Bonus points) Why do you think the negative TTL is not in each resource?

Which resource? We've been told the resource does not exist!

Question 6

Q1: Did a recursive resolver or an authoritative server answer?
Q2: How many different ways can you tell?
Q3: Can you speculate about the 'distance' to the server?
Q4: What did we ask for?
Q5: Did the server just answer the question?
Q6: How long are we allowed to cache the answer?
Q7: Did we ask for recursion? Did recursion occur?

```
[992] (philip@rincewind)~% drill @ibid.nixsys.be trouble.is aaaa
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 38713
;; flags: qr aa rd ; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 0
;; QUESTION SECTION:
;; trouble.is.  IN      AAAA

;; ANSWER SECTION:
trouble.is.     86400   IN      AAAA    2a01:4f8:a0:10e6::1:1

;; AUTHORITY SECTION:
trouble.is.     86400   IN      NS      ibid.nixsys.be.
trouble.is.     86400   IN      NS      soapstone.yuri.org.uk.
trouble.is.     86400   IN      NS      panda.droso.dk.

;; ADDITIONAL SECTION:

;; Query time: 3 msec
;; SERVER: 148.251.159.32
;; WHEN: Sat Mar 28 10:57:46 2015
;; MSG SIZE  rcvd: 147
```

Question 7

Q1: Did a recursive resolver or an authoritative server answer?
Q2: How many different ways can you tell?
Q3: Can you speculate about the 'distance' to the server?
Q4: What did we ask for?
Q5: Did the server just answer the question?
Q6: How long are we allowed to cache the answer?
Q7: Did we ask for recursion? Did recursion occur?

```
[993] (philip@rincewind)~% drill foobar.trouble.is TXT
;; ->>HEADER<<- opcode: QUERY, rcode: NXDOMAIN, id: 13484
;; flags: qr rd ra ; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION:
;; foobar.trouble.is.   IN      TXT

;; ANSWER SECTION:

;; AUTHORITY SECTION:
trouble.is.     1210    IN      SOA     ibid.nixsys.be. hostmonster.nixsys.be. 2015032800 3600 900 604800 3600

;; ADDITIONAL SECTION:

;; Query time: 10 msec
;; SERVER: 127.0.0.1
;; WHEN: Sat Mar 28 11:04:50 2015
;; MSG SIZE  rcvd: 97
```

Quiz day 2

Question 1

- **What are some reasons for DNS running on UDP instead of TCP? Under what circumstances does DNS use TCP?**

Question 1

- What are some reasons for DNS running on UDP instead of TCP? Under what circumstances does DNS use TCP?

- DNS is usually asynchronous and stateless
- TCP would be very high overhead for the common case of a short query and a warm cache
- Ideally DNS lookups should have low latency and not get in the way

Question 2

- What do we mean by "recursion" in the context of DNS?
- How does DNS work? (in a nutshell)

Question 2

- What do we mean by "recursion" in the context of DNS?
- How does DNS work? (in a nutshell)
- Recursing through the DNS means walking the tree from the root to the leaves
- DNS is fundamentally about asking increasingly specific specific questions as you walk the tree

Question 3

- What are some of the ways you can tell whether a query was answered by a recursive resolver or an authoritative nameserver

Question 3

- What are some of the ways you can tell whether a query was answered by a recursive resolver or an authoritative nameserver

- Symmetry of flags (rd / ra)
- Presence of the aa flag
- Nice round numbers (repeating queries)
- Absence of authority section

Question 4

- What is special about the CNAME record?

Question 4

- **What is special about the CNAME record?**

It cannot coexist with any other resource record

Question 5

- Is status: NXDOMAIN an error? Why or why not?

Question 5

- Is status: NXDOMAIN an error? Why or why not?

- NXDOMAIN is “denial of existence”
- It is not an error!

Question 6

- What does "NOERROR, zero answers" tell us?

Question 6

- What does "NOERROR, zero answers" tell us?

- The “domain name” exists
- But no records of the given type exist

Question 7

- What is EDNS0? Why do we need it?
- How does it work?

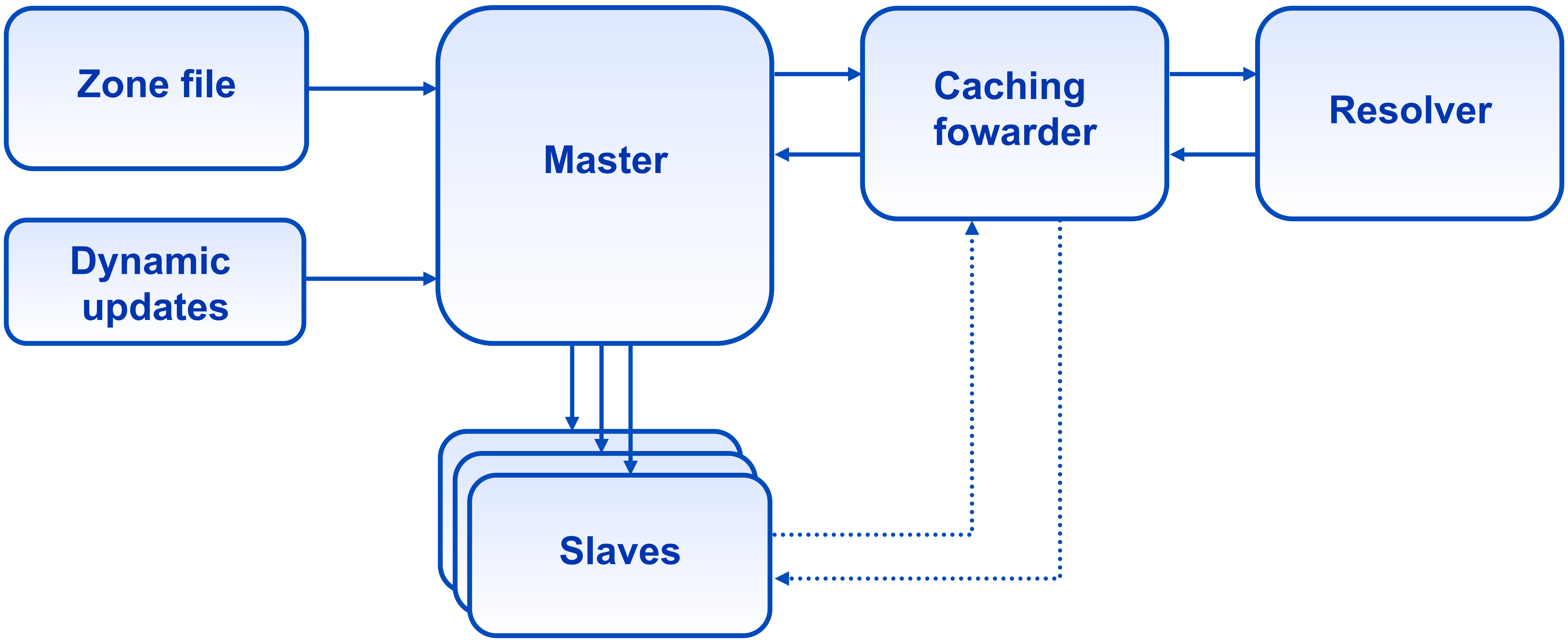
Question 7

- What is EDNS0? Why do we need it?
- How does it work?

- EDNS0 allows >512 byte replies over UDP
- We need it for DNSSEC and often for IPv6
- It uses a pseudo-header in the reply packet

Quiz 2

1.What does TSIG Protect?



Question 2

- How does the master know which key to use for host to host communication?

Question 3

- How are the keys of the master and the slave related to each other?

Question 4

- How is the zone transfer secured?

Question 5

- TSIG uses hashing to secure zone transfer.
- What is hashed exactly?

Question 6

- In which different ways can you transfer zone files between master and slaves?

Question 7

- Full zone transfer. How does the slave know how often it should check the master?

Question 8

- Full zone transfer. How long will the slave wait before it retries copying the zone from the master after the first try failed.

Question 9

What is the reverse domain for this block?

- 64.102.24.0/23

Question 10

What is the reverse domain for this block?

- **2001:db8::/32**

Question 11

- **What record is being queried here?**
- **dig www.ripe.net.net**

Quiz 3

Question 3

- How do you protect messages in master-slave communication from being manipulated?

Question 3

- How do you protect messages in master-slave communication from being manipulated?

- TSIG

Question 4

- What does a TSIG signature cover in a query?

Question 4

- What does a TSIG signature cover in a query?

- Query (type, flags, qname,...)
- Key name (!)
- Timestamp

Question 5

- What kinds of evil does TSIG *not* protect against?

Question 5

- What kinds of evil does TSIG *not* protect against?

- Mangling of zonefiles by rogue employees or others with (accidental or intentional) access to zone file data
- Caching resolver impersonation
- Many others!

Question 6

- Is TSIG part of DNSSEC?

Question 6

- Is TSIG part of DNSSEC?

- Not really
- But often used in combination

Quiz 4

Question 1

- What is the Key Signing Key used for?

Question 1

- What is the Key Signing Key used for?
- Signing the Zone Signing Key
- Enables one to rotate the ZSK frequently without having to interact with parent zones every time

Question 2

- What is the Zone Signing Key used for?

Question 2

- What is the Zone Signing Key used for?
- Signing RRSets in zones (creating RRSIG records)

Question 3

- How do you verify a signature?

Question 3

- How do you verify a signature?
- Query the RRSET
- Query the RRSIG
- Query the DNSKEY
- Recursively follow the DS chain of trust

Question 4

- How does a parent assert that a resolver can trust a child zone?

Question 4

- How does a parent assert that a resolver can trust a child zone?
- Parent zones contain DS records provided by child zone operators
- DS records are hashes of DNSKEY public keys

Question 5

- Imagine you are a recursive resolver.
- You just queried a child'd DS record for a certain zone from the parent's zone.
- **How do you verify, step by step, that the child's zone's DNSKEY can be trusted?**
- Assume that you can trust the zone's parent.

Question 5

- Imagine you are a recursive resolver.
- You just queried a child'd DS record for a certain zone from the parent's zone.
- **How do you verify, step by step, that the child's zone's DNSKEY can be trusted?**
- Assume that you can trust the zone's parent.

- You need the Child's public KSK
- You need the signature on the DS record by the parent's private ZSK
- Decrypt the RRSIG on the DS record using the parent's public ZSK
- Hash the child's KSK
- Compare it to the DS recordß

Question 6

- Why do we need 2 DNSKEYs?
- What do they do?

Question 7

- Can you give alternative names for “recursive server?”

- What does it do?

Question 8

- You configure the “DNS Server address” for your laptops.
- How did we call “DNS Server” in this workshop?

RIPE NCC

Question 9

- Can you give alternative names for the “recursive resolver?”

Question 10

- What does the stub resolver do?

Question 11

- Sitting at your laptop, how can you be aware if your recursive resolver is configured for DNSSEC ?

The End!

Тамом Край Y Diwedd

Fí

النهاية Соңы Վերջ Finis

Liðugt

Ende Finvezh Кінець

Konec Kraj Ënn Fund پایان

Lõpp Beigas Vége Son An Críoch Крај

הסוף

Fine Endir Sfârşit Fin Τέλος

Einde

Конец

Slut Slutt

დასასრული Pabaiga

Fim Amaia Loppu Tmiem Koniec