



# DNS Session 2: DNS cache operation and DNS debugging

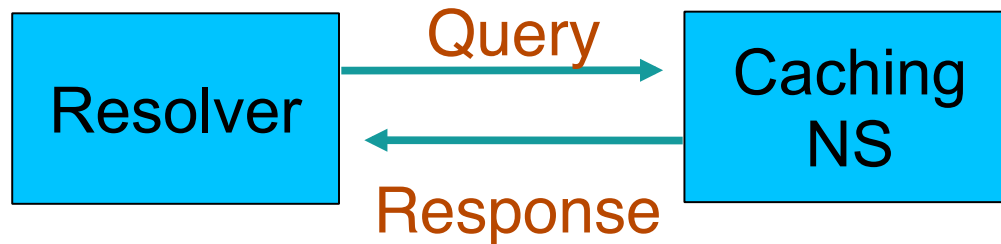


These materials are licensed under the Creative Commons *Attribution-Noncommercial 3.0 Unported* license (<http://creativecommons.org/licenses/by-nc/3.0/>)

# DNS Cache Operation

# How caching NS works (1)

If we've dealt with this query before recently, answer is already in the cache - easy!



# What if the answer is not in the cache?

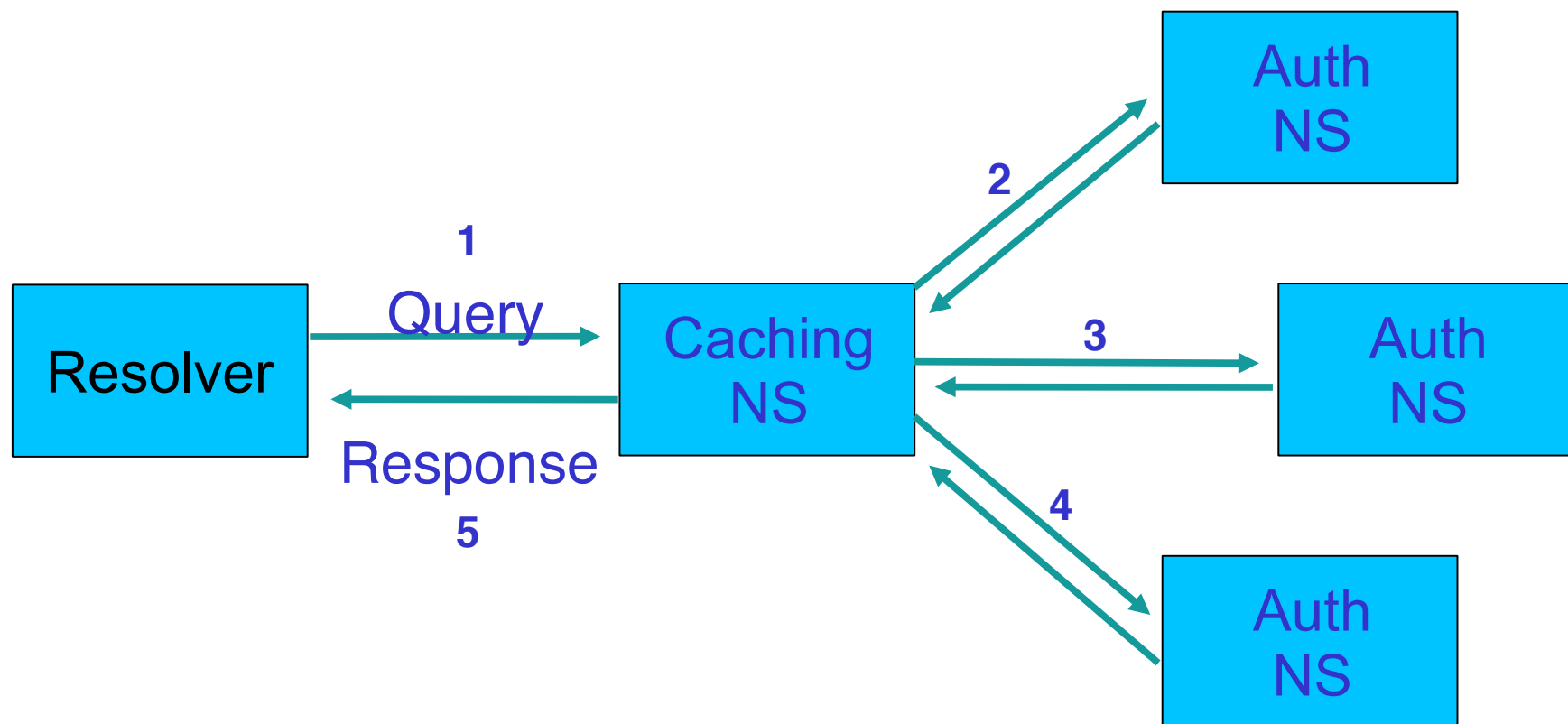
DNS is a distributed database: parts of the tree (called "zones") are held in different servers

They are called "authoritative" for their particular part of the tree

It is the job of a caching nameserver to locate the right authoritative nameserver and get back the result

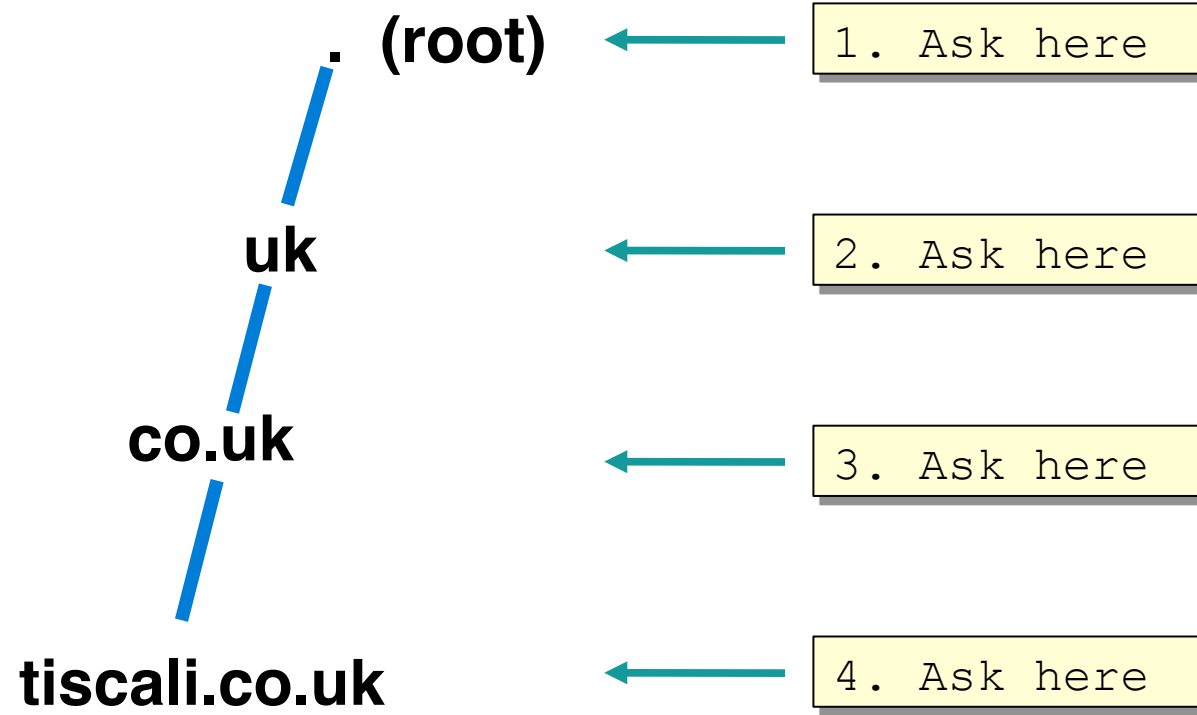
It may have to ask other nameservers first to locate the one it needs

# How caching NS works (2)



# How does it know which authoritative nameserver to ask?

It follows the hierarchical tree structure  
e.g. to query "www.tiscali.co.uk"



# **Intermediate nameservers return "NS" resource records**

"I don't have the answer, but try these other nameservers instead"

Called a REFERRAL

Moves you down the tree by one or more levels

# Eventually this process will either:

Find an authoritative nameserver which knows the answer (positive or negative)

Not find any working nameserver:

*SERVFAIL*

End up at a faulty nameserver - either cannot answer and no further delegation, or wrong answer!

Note: the caching nameserver may happen also to be an authoritative nameserver for a particular query. In that case it will answer immediately without asking anywhere else. We will see later why it's a better idea to have separate machines for caching and authoritative nameservers



# How does this process start?

Each caching nameserver has a list of root

```
/etc/bind/named.conf
```

```
include "/etc/bind/named.conf.options";  
include "/etc/bind/named.conf.local";  
include "/etc/bind/named.conf.default-zones";
```

```
/etc/bind/named.conf.default-zones
```

```
zone "." {  
    type hint;  
    file "/etc/bind/db.root";  
};
```

```
/etc/bind/db.root
```

```
.           3600000   IN      NS      A.ROOT-SERVERS.NET.  
A.ROOT-SERVERS.NET. 3600000   A       198.41.0.4  
A.ROOT-SERVERS.NET. 3600000   AAAA    2001:503:BA3E::2:30  
;  
; FORMERLY NS1.ISI.EDU  
;  
.           3600000   NS      B.ROOT-SERVERS.NET.  
B.ROOT-SERVERS.NET. 3600000   A       192.228.79.201  
..  
..
```

# Where did named.root come from?

`ftp://ftp.internic.net/domain/named.cache`

Worth checking every 6 months or so for updates

# Demonstration

```
dig +trace www.tiscali.co.uk.
```

Instead of sending the query to the cache,  
"dig +trace" traverses the tree from the  
root and displays the responses it gets

`dig +trace` is a bind 9 feature

useful as a demo but not for debugging

# **Distributed systems have many points of failure!**

So each zone has two or more authoritative nameservers for resilience

They are all equivalent and can be tried in any order

Trying stops as soon as one gives an answer

Also helps share the load

The root servers are very busy

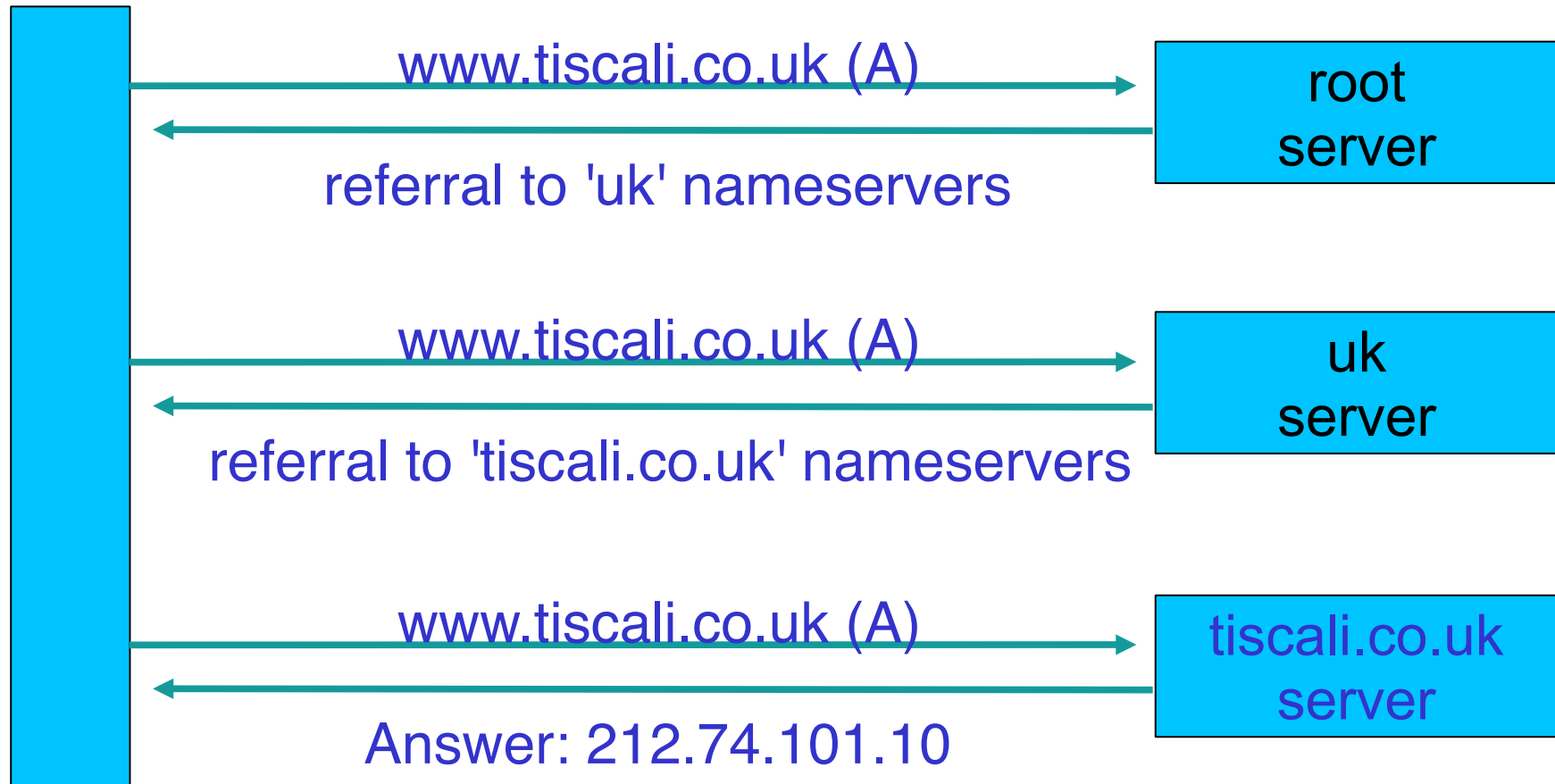
There are currently 13 of them (each of which is a large cluster)

# Caching reduces the load on auth nameservers

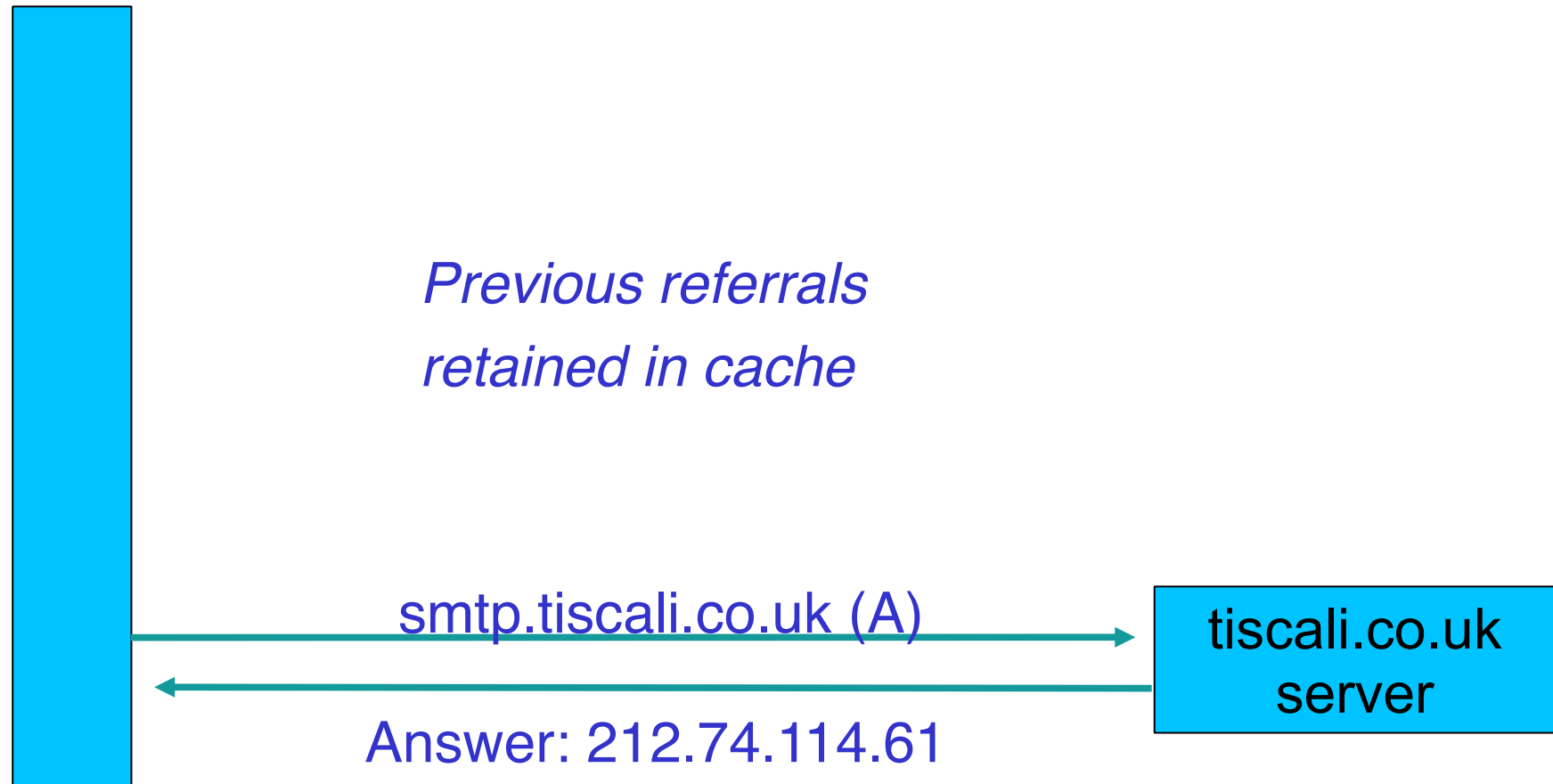
Especially important at the higher levels: root servers, GTLD servers (.com, .net ...) and ccTLDs

All intermediate information is cached as well as the final answer - so NS records from REFERRALS are cached too

# Example 1: www.tiscali.co.uk (on an empty cache)



# Example 2: smtp.tiscali.co.uk (after previous example)



# **Caches can be a problem if data becomes stale**

If caches hold data for too long, they may give out the wrong answers if the authoritative data changes

If caches hold data for too little time, it means increased work for the authoritative servers



# **The owner of an auth server controls how their data is cached**

Each resource record has a "Time To Live" (TTL) which says how long it can be kept in cache

The SOA record says how long a negative answer can be cached (i.e. the non-existence of a resource record)

Note: the cache owner has no control - but they wouldn't want it anyway

# A compromise policy

Set a fairly long TTL - 1 or 2 days

When you know you are about to make a change, reduce the TTL down to 10 minutes

Wait 1 or 2 days BEFORE making the change

After the change, put the TTL back up again

# Any questions?

?

# DNS Debugging

# **What sort of problems might occur when resolving names in DNS?**

Remember that following referrals is in general a multi-step process

Remember the caching

# **(1) One authoritative server is down or unreachable**

Not a problem: timeout and try the next authoritative server

Remember that there are multiple authoritative servers for a zone, so the referral returns multiple NS records

## **(2) \*ALL\* authoritative servers are down or unreachable!**

This is bad; query cannot complete

Make sure all nameservers not on the same subnet (switch/router failure)

Make sure all nameservers not in the same building (power failure)

Make sure all nameservers not even on the same Internet backbone (failure of upstream link)

For more detail read RFC 2182

### **(3) Referral to a nameserver which is not authoritative for this zone**

Bad error. Called "Lame Delegation"

Query cannot proceed - server can give neither the right answer nor the right delegation

Typical error: NS record for a zone points to a caching nameserver which has not been set up as authoritative for that zone

Or: syntax error in zone file means that nameserver software ignores it



## **(4) Inconsistencies between authoritative servers**

If auth servers don't have the same information then you will get different information depending on which one you picked (random)

Because of caching, these problems can be very hard to debug. Problem is intermittent.

## **(5) Inconsistencies in delegations**

NS records in the delegation do not match  
NS records in the zone file (we will write  
zone files later)

Problem: if the two sets aren't the same,  
then which is right?

Leads to unpredictable behaviour

Caches could use one set or the other, or the  
union of both

## **(6) Mixing caching and authoritative nameservers**

Consider when caching nameserver contains an old zone file, but customer has transferred their DNS somewhere else

Caching nameserver responds immediately with the old information, even though NS records point at a different ISP's authoritative nameservers which hold the right information!

This is a very strong reason for having separate machines for authoritative and caching NS

Another reason is that an authoritative-only NS has a fixed memory usage

## **(7) Inappropriate choice of parameters**

e.g. TTL set either far too short or far too long

# **These problems are not the fault of the caching server!**

They all originate from bad configuration of the AUTHORITATIVE name servers

Many of these mistakes are easy to make but difficult to debug, especially because of caching

Running a caching server is easy; running authoritative nameservice properly requires great attention to detail

# How to debug these problems?

We must bypass caching

We must try *\*all\** N servers for a zone (a caching nameserver stops after one)

We must bypass recursion to test all the intermediate referrals

"dig +norec" is your friend

```
dig +norec @1.2.3.4 foo.bar. a
```



Server to query

Domain

Query type

# How to interpret responses (1)

Look for "status: NOERROR"

"flags ... aa" means this is an authoritative answer (i.e. not cached)

"ANSWER SECTION" gives the answer

If you get back just NS records: it's a referral

```
;; ANSWER SECTION  
foo.bar.      3600    IN      A       1.2.3.4
```



Domain name

TTL

Answer

# How to interpret responses (2)

*"status: NXDOMAIN"*

OK, negative (the domain does not exist). You should get back an SOA

*"status: NOERROR"* with zero RRs

OK, negative (domain exists but no RRs of the type requested). Should get back an SOA

Other status may indicate an error

Look also for *Connection Refused* (DNS server is not running or doesn't accept queries from your IP address) or *Timeout* (no answer)



# How to debug a domain using "dig +norec" (1)

1. Start at any root server: [a-m].root-

```
dig +norec @a.root-servers.net. www.tiscali.co.uk. a
```

*Remember the trailing dots!*

1. For a referral, note the NS records returned
2. Repeat the query for *\*all\** NS records
3. Go back to step 2, until you have got the final answers to the query

# How to debug a domain using "dig +norec" (2)

1. Check all the results from a group of authoritative nameservers are consistent with each other
2. Check all the final answers have "flags: aa"
3. Note that the NS records point to names, not IP addresses. So now check every NS record seen maps to the correct IP address using the same process!!

# How to debug a domain using "dig +norec" (3)

Tedious, requires patience and accuracy, but it pays off

Learn this first before playing with more automated tools

Such as:

<http://www.squish.net/dnscheck/>

<http://www.zonecheck.fr/>

These tools all have limitations, none is perfect

# Practical

***Worked examples***

# Building your own caching nameserver

Most common software is “BIND” (Berkeley Internet Name Domain) from ISC, [www.isc.org](http://www.isc.org)

There are other options, e.g. NSD, [www.nlnetlabs.nl](http://www.nlnetlabs.nl)

Most UNIX/Linux distributions have a package for bind and will configure it as a cache.

Ubuntu: `apt-get install bind9`

RedHat/Fedora/CentOS: `yum -y install bind`

FreeBSD: in the base system

Question: what sort of hardware would you choose when building a DNS cache?

# Improving the configuration

Limit client access to your own IP addresses only

No reason for other people on the Internet to be using your cache resources

Make cache authoritative for queries which should not go to the Internet

localhost → A 127.0.0.1

1.0.0.127.in-addr.arpa → PTR localhost

RFC 1918 addresses (10/8, 172.16/12, 192.168/16)

Gives quicker response and saves sending unnecessary queries to the Internet

# Access control

/etc/bind/named.conf.options

```
acl ternet {
    127.0.0.1;
    10.10.0.0/24;
};

Options {
    directory "/var/cache/bind";

    forwarders {
        10.10.0.254;
    };

    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { any; };
    recursion yes;
    allow-recursion { ternet; };
listen-on { any; };

};
```

# localhost -> 127.0.0.1

```
/etc/bind/named.conf
```

```
include "/etc/bind/named.conf.options";  
include "/etc/bind/named.conf.local";  
include "/etc/bind/named.conf.default-zones";
```

```
/etc/bind/named.conf.default-zones
```

```
zone "localhost" {  
    type master;  
    file "/etc/bind/db.local";  
};
```

```
/etc/bind/db.local
```

```
$TTL      604800  
@      IN      SOA localhost. root.localhost. (  
                2      ; Serial  
                604800 ; Refresh  
                86400  ; Retry  
                2419200 ; Expire  
                604800 ) ; Negative Cache TTL  
;  
@      IN      NS   localhost.  
@      IN      A    127.0.0.1  
@      IN      AAAA  ::1
```



# 127.0.0.1 -> localhost

```
/etc/bind/named.conf
```

```
include "/etc/bind/named.conf.options";  
include "/etc/bind/named.conf.local";  
include "/etc/bind/named.conf.default-zones";
```

```
/etc/bind/named.conf.default-zones
```

```
zone "127.in-addr.arpa" {  
    type master;  
    file "/etc/bind/db.127";  
};
```

```
/etc/bind/db.127
```

```
$TTL      604800  
@      IN      SOA localhost. root.localhost. (  
                                1      ; Serial  
                                604800 ; Refresh  
                                86400  ; Retry  
                                2419200 ; Expire  
                                604800 ) ; Negative Cache TTL  
;  
@      IN      NS   localhost.  
1.0.0  IN      PTR  localhost.
```

# rfc 1918 zones

/etc/bind/named.conf

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

/etc/bind/named.conf.local

```
// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";
```

/etc/bind/zones.rfc1918

```
zone "10.in-addr.arpa"      { type master; file "/etc/bind/db.empty"; };

zone "16.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
zone "17.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
zone "18.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
zone "19.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
...
...
...
zone "29.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
zone "30.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
zone "31.172.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };

zone "168.192.in-addr.arpa" { type master; file "/etc/bind/db.empty"; };
```

# Managing a caching nameserver

```
service bind9 start
```

```
rndc status
```

```
rndc reload
```

**After config changes; causes less disruption than restarting the daemon**

```
rndc dumpdb
```

**dumps current cache contents to**  
`/var/cache/bind/named_dump.db`

```
rndc flush
```

**Destroys the cache contents; don't do on a live system!**

# Absolutely critical!

```
tail /var/log/syslog
```

after any nameserver changes and reload/  
restart

A syntax error may result in a nameserver  
which is running, but not in the way you  
wanted

bind is very fussy about syntax

Beware } and ;

Within a zone file, comments start with  
semicolon (;) NOT hash (#)

# Practical

Build a caching nameserver

Examine its operation