BIND 9

(Part 4 - Managing dynamic DNS zones)

Carsten Strotmann and the ISC Team

Welcome

Welcome to part four of our BIND 9 webinar series

In this Webinar

- Dynamic zones vs. static zones
- Provisioning, updating dynamic zones
- NOTIFY and incremental zone transfer (IXFR)
- Hands-On with dynamic zones

The evolution of DNS zone file management

In the beginning

- In the early years of DNS (~ 1988-1994) most networks were static
  - large multi user systems that don't change often
  - maybe one or two changes per month on a large network
  - zone file were static as well, managed by admins with text editors

Client-Server computing

- From 1994 onwards, more and more PC style machines were added to the networks
  - Networks became more dynamic
  - *BOOTP* and later *DHCP* were developed to assign IP addresses dynamically
  - DNS had to keep up with the dynamic nature of the networks
- Dynamic Updates were added to DNS
  - The DHCP server or clients could update the name-to-IP-Address binding in DNS

Dynamic DNS updates, NOTIFY and IXFR

- With dynamic DNS updates, a zone file can change very frequently
  - maybe multiple times in a minute
- The old way of synchronizing authoritative DNS servers with the SOA *REFRESH* value is too slow in these scenarios
  - NOTIFY was invented to trigger almost immediate DNS zone transfers
- Now that zone transfers could occur very often, sending the whole zone (AXFR = all records zone transfer) each time was overhead
  - Incremental Zone Transfer (IXFR) was invented to only send the changes since the last transfer

Service-Registration and -Discovery

- From year 2000 onwards, applications started to use DNS for their services
  - Service Discovery with SRV records (Windows Active Directory, FreeIPA)
  - Wide-Area DNS Service Discovery (DNS-SD)
  - Authentication Token (Let's Encrypt)
  - Container Discovery (Cloud-Services)
  - DNS Server configuration management (BIND 9 Catalog Zones)
- Integrated Network Management Systems (DDI) started to manage DNS through dynamic updates

BIND 9

Dynamic Updates (Plus NOTIFY & IXFR)

DNSIND

- DNSIND is shorthand for a group of extensions to the original DNS protocol.
  - I is for Incremental Zone Transfer (RFC 1995)
  - N is for NOTIFY (RFC 1996)
  - D is for Dynamic Update (RFC 2136)
- Together, the DNSIND extensions allow DNS to handle dynamic networks.
  - RFC compliant dynamic DNS updates are independent and different from dynamic DNS update schemes used to update the address records for devices where the IP address is changed dynamically by the internet provider (DynDNS etc).

Benefits of dynamic zones

- Dynamic Updates can be sent over the network, there is no need to login to the primary authoritative DNS server
- The dynamic DNS update tools (such as `nsupdate`) check the syntax before sending the update
  - this prevents erroneous resource records entering the zone file
- The SOA serial number is automatically incremented on every dynamic change
- In an DNSSEC secured zone, the change will be immediately signed
- DNS zone changes can be easily scripted
- External tool such as Let's Encrypt certbot can make changes to the zone
- The update function can be limited to certain domain names and record types

Dynamic DNS updates

- A dynamic update allows a node to change the contents of a zone.
  - DHCP servers, for example, can add `A`, `AAAA`, and `PTR` records to reflect the leases they give out.
  - Clients can update their DNS information on the authoritative DNS server
  - IP Address Management Systems can update the DNS data based on their inventory database
  - DNS management web front-ends can use dynamic updates as an API to make changes to the DNS data

Dynamic DNS updates

- A dynamic update can:
  - Add resource records to a zone
  - Delete records from a zone:
    - One record
    - All records of a certain type with one domain name (an RRset)
    - All records with one domain name
- There is no *modify* operation in dynamic updates
  - to modify DNS zone data, delete the old data and add the new data inside one atomic transaction

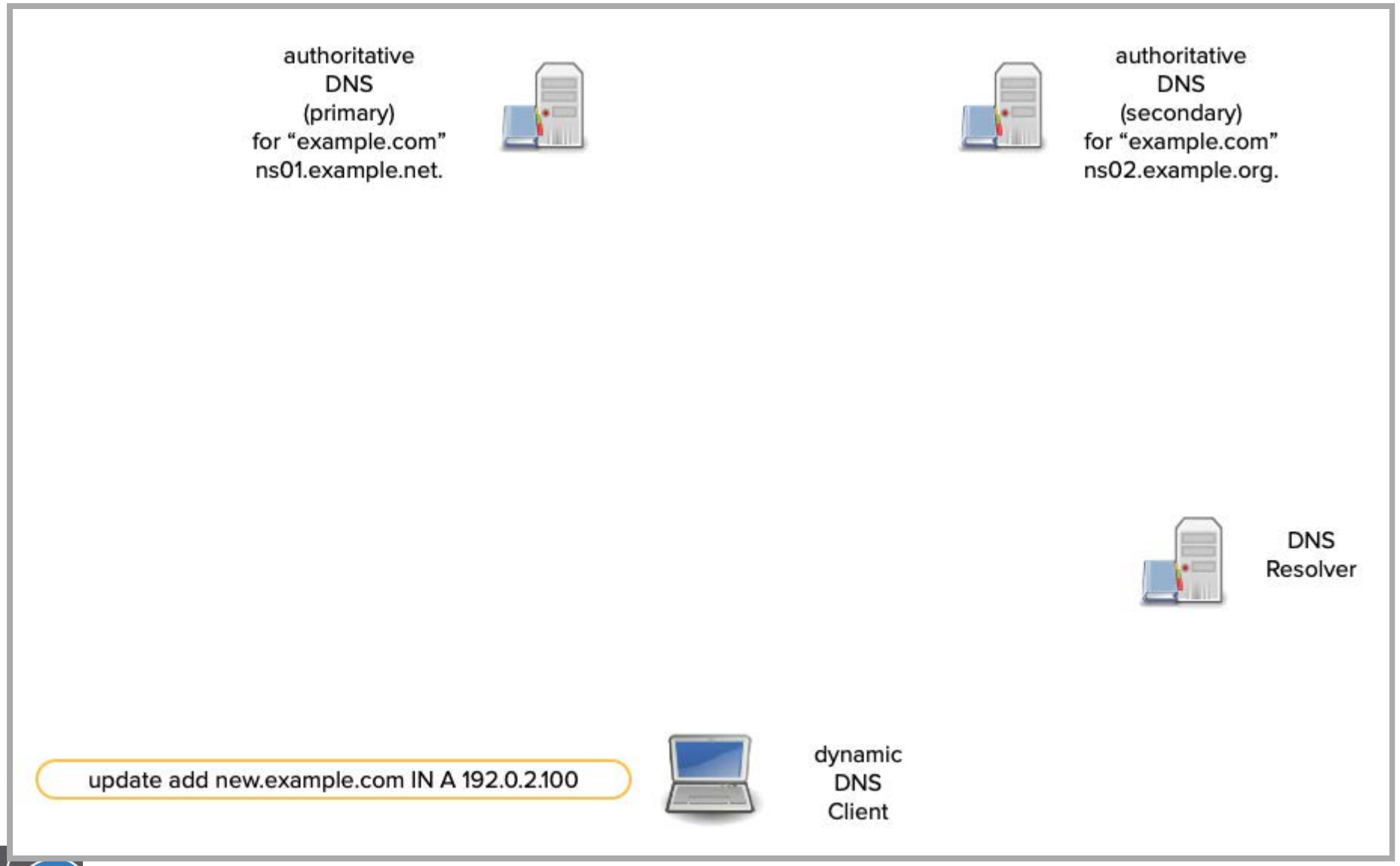
Dynamic DNS updates security

- A server can be configured to only accept updates from senders with the proper PSK (Pre-Shared TSIG Key) or from specific IP addresses.
- A sender can define update prerequisites, such as the existence/non-existence of:
  - A particular record
  - Resource records with a specific domain name and type

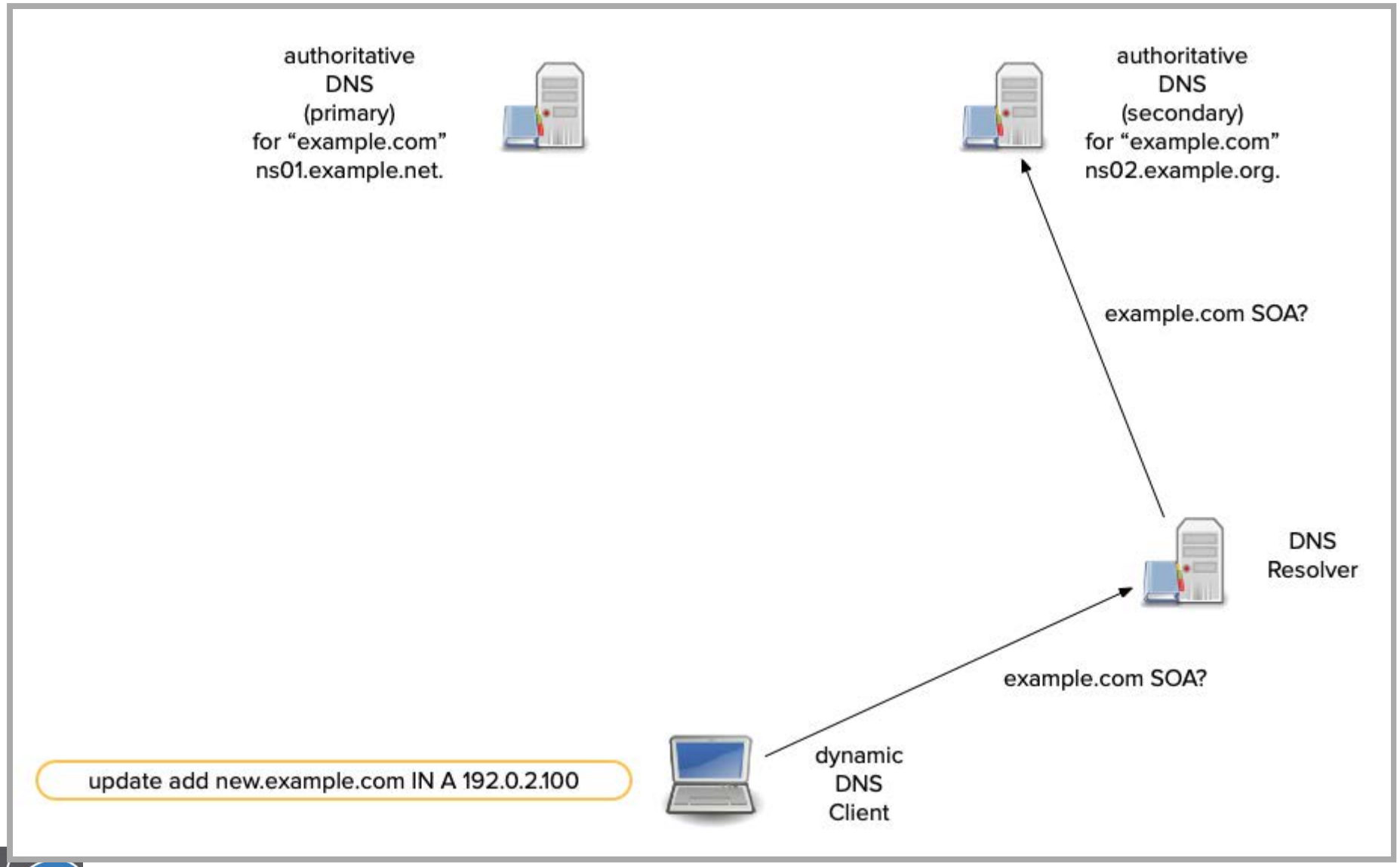
Dynamic DNS updates - how it works

- A sender first tries to transmit an update to a zone's primary server.
  - Most dynamic DNS update tools compare the SOA's `mname` field to the NS RRset.
  - If the `mname` matches an NS RR, the update is sent there.
  - Otherwise the update is sent to one of the domain names in the NS RRset (from the sender's perspective, a secondary).

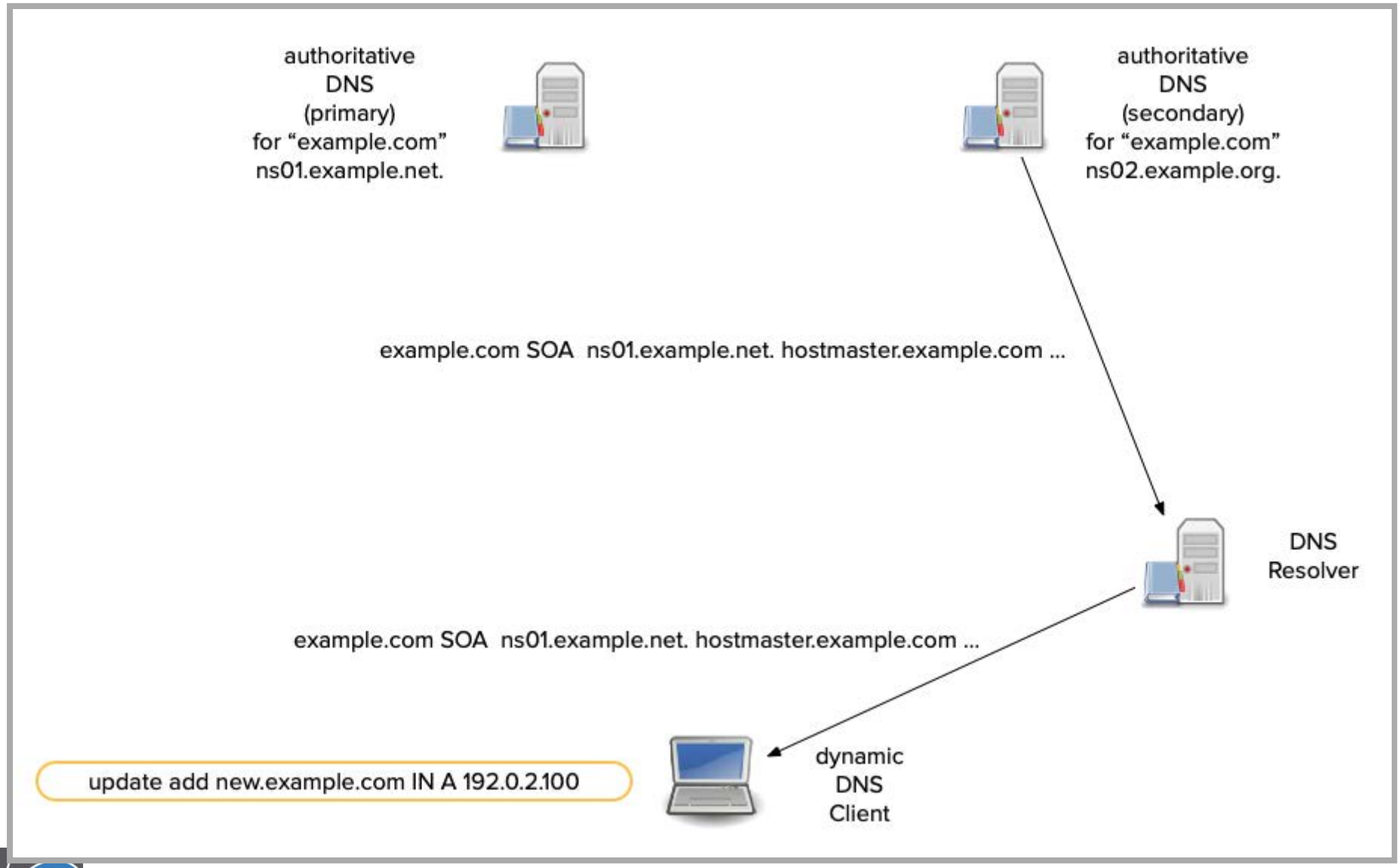
Dynamic DNS updates - how it works



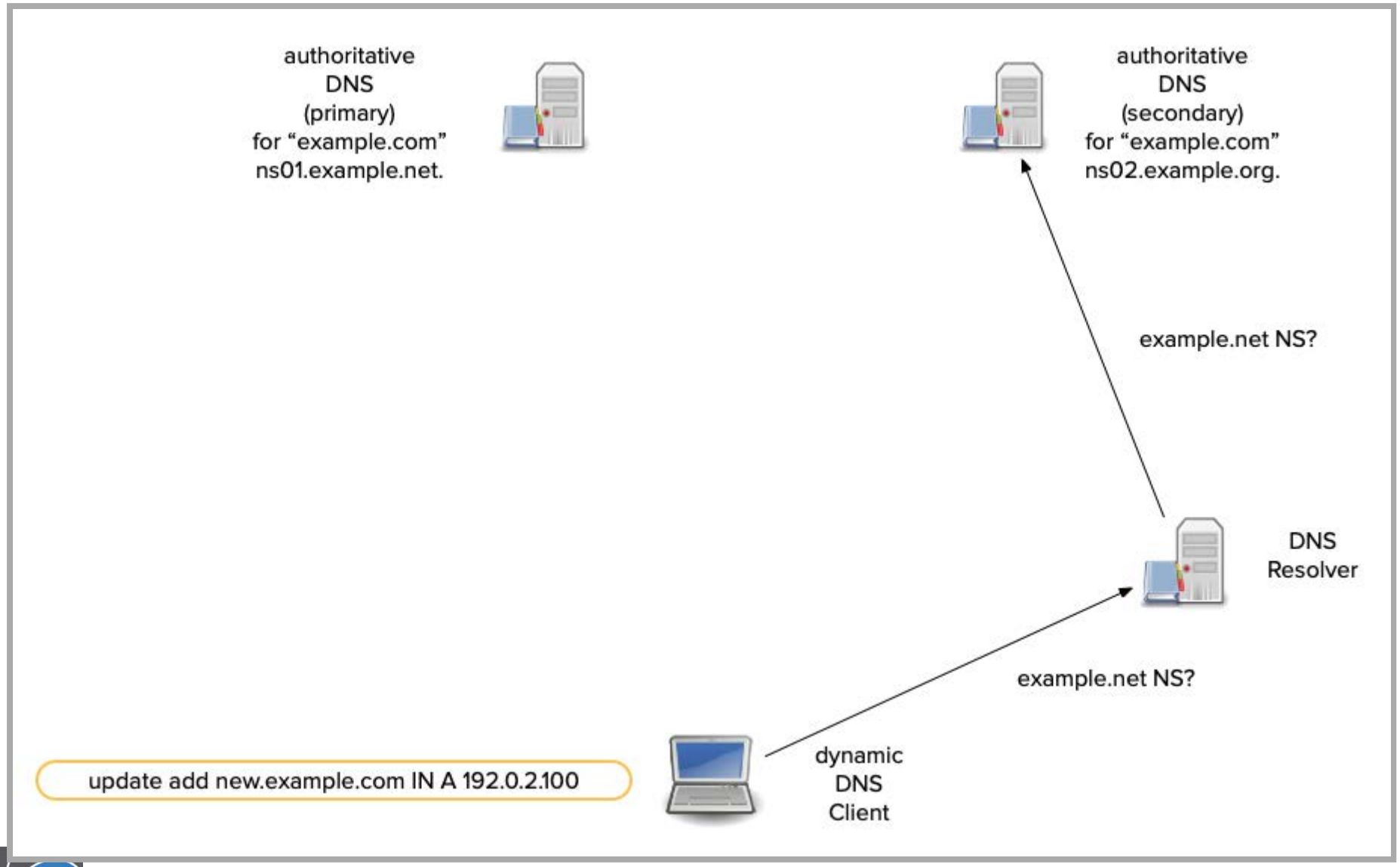
Dynamic DNS updates - how it works



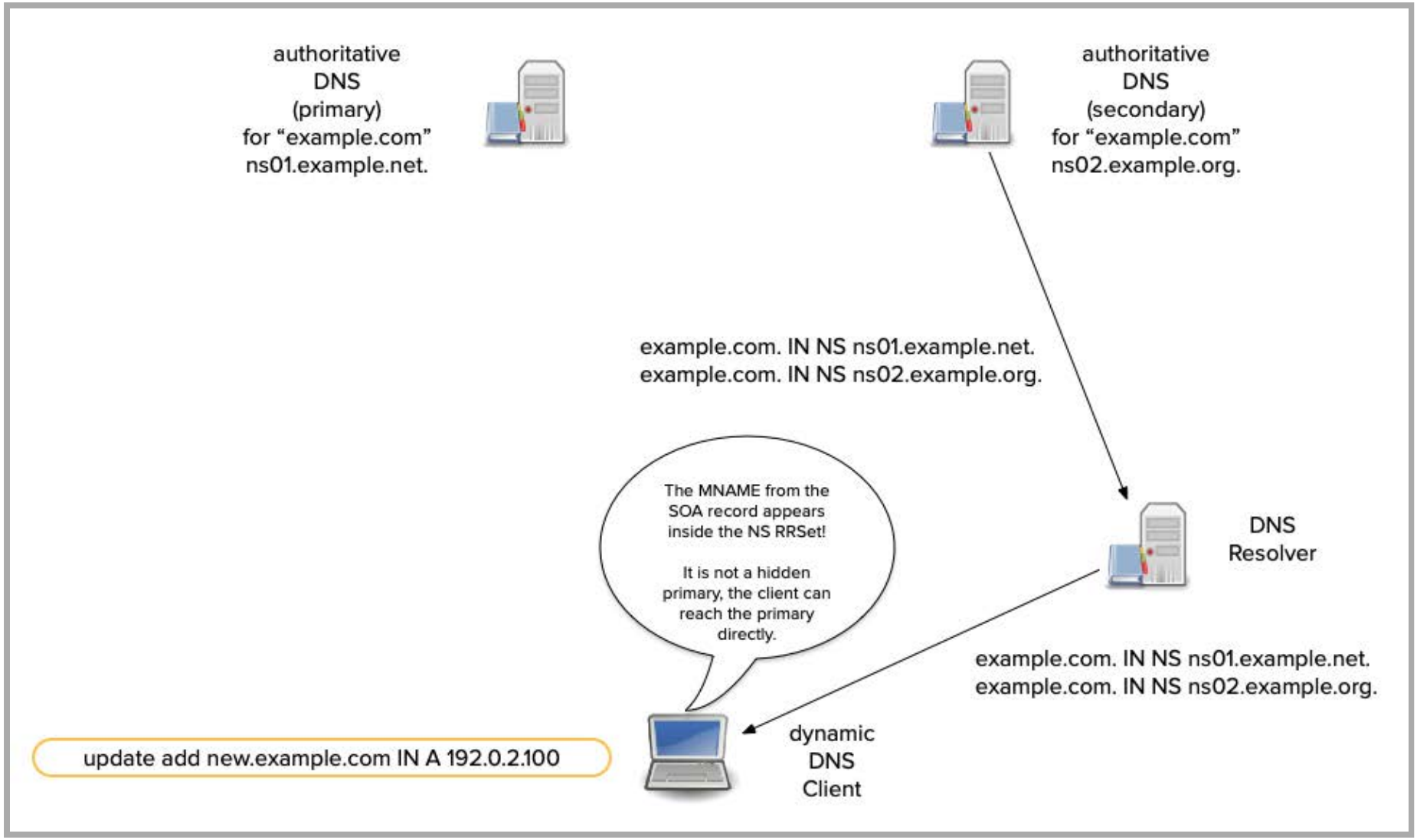
Dynamic DNS updates - how it works



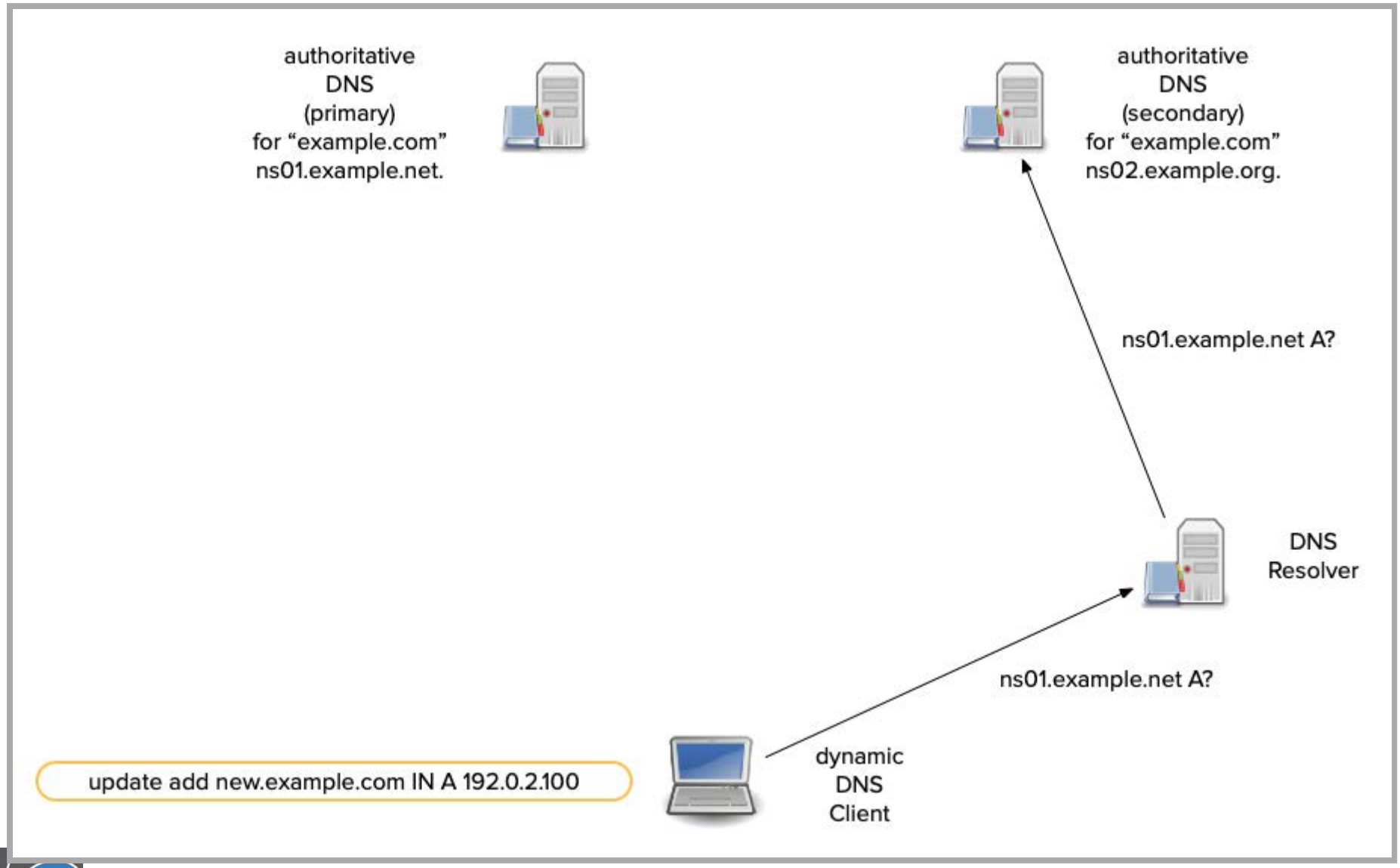
Dynamic DNS updates - how it works



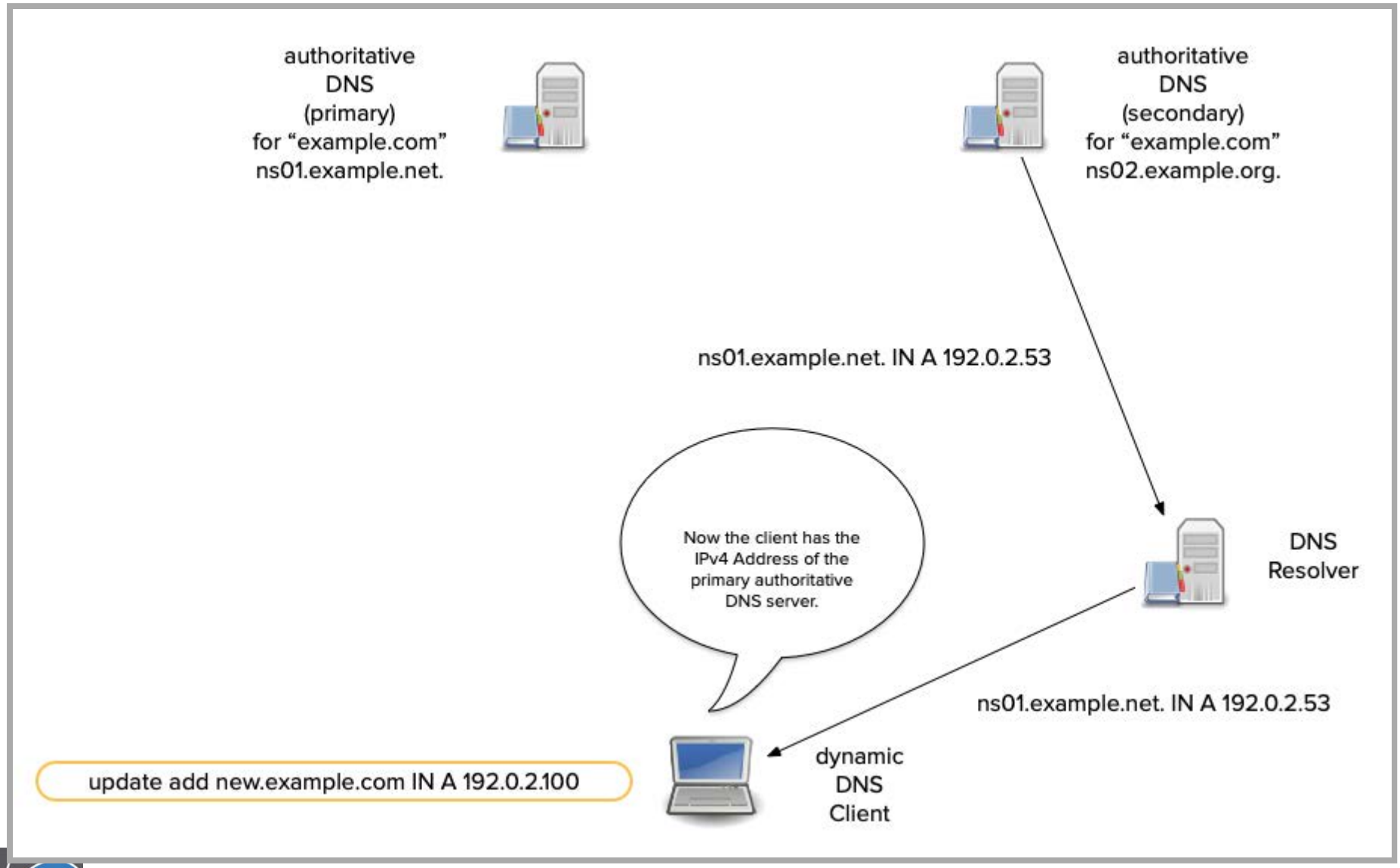
Dynamic DNS updates - how it works



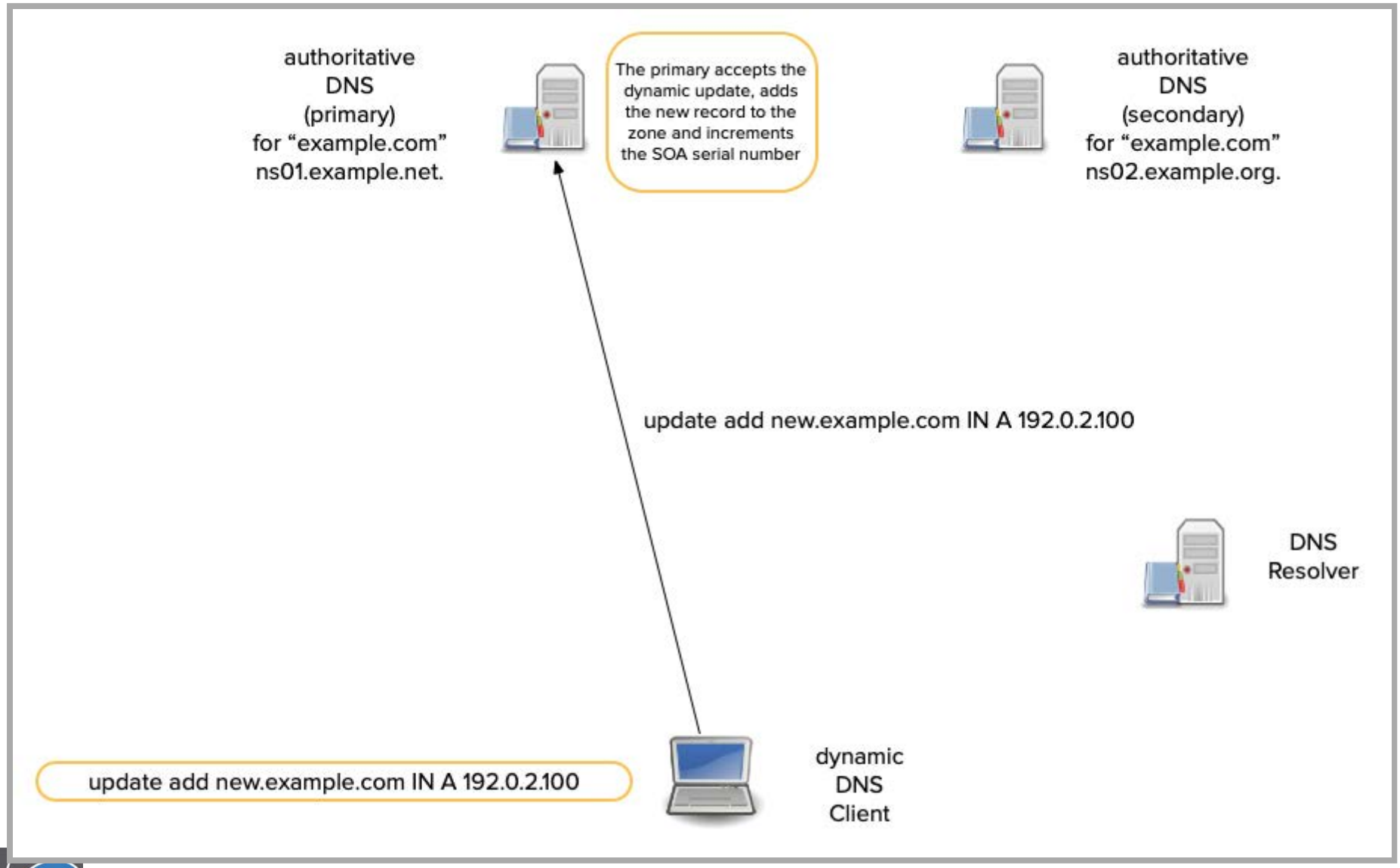
Dynamic DNS updates - how it works



Dynamic DNS updates - how it works



Dynamic DNS updates - how it works



Dynamic update security with ACLs

- On a primary authoritative server, the configuration statement `allow-update {};` in the `named.conf` configuration file enables DDNS.
  - It also limits the updates to specific keys or addresses.
  - Using a symmetric key (TSIG) is more secure, but IP addresses can be provided instead.
    - We will cover securing DDNS with TSIG keys in the upcoming 2nd part of this webinar

```
zone "example.com" {
        type primary;
        file "example.com";
        allow-update { 192.0.2.68; };
};
```

Dynamic Zones: SOA Serial

Updating the SOA serial on dynamic updates

- A dynamic update changes the zone content
  - The SOA serial number must be incremented on zone changes
    - the secondary authoritative DNS servers use the SOA serial to detect updates and to initiate a zone transfer

Updating the SOA serial on dynamic updates

- There are three update options for the SOA in dynamic zones:
  - `serial-update-method ( date | increment | unixtime );`
    - `increment`: simple bump (the default).
    - `date`: YYYYMMDDnn. nn starts at zero and increments.
    - `unixtime`: standard unixtime in seconds.
- The statement can be global in `options` block or `view` block, or specific to a `zone` block.

BIND 9

The Journal file

The Journal-File for dynamic DNS updates

- To avoid constant rewriting of the zone file, a BIND primary instead writes changes to a log file:
  - the log is known as a journal and is not plain-text.
  - the journal's file name is the zone's, appended with `.jnl`.
  - the name and the max size of a journal are configurable.

```
zone "example.com" {
   type primary;
   file "example.com";
   allow-update { key dhcp-server; };
   journal "/fast-disk/example.com.jnl"
   max-journal-size 200M;
};
```

The Journal File

- The journal allows the server to recover the most recent version of the zone after a crash.
  - After loading the zone file, the server loads the journal.
  - A journal can be viewed with: `named-journalprint <journalfile>`
- Each line records the addition or deletion of one DNS resource record.

```
% named-journalprint zone02.dane.onl.jnl
del zone02.dane.onl.    30      IN      SOA     server02.dane.onl. hm.zone02.dane.onl. 2016011308 7200
del me.zone02.dane.onl. 30      IN      TXT     "my birthday!  go me!"
add zone02.dane.onl.    30      IN      SOA     server02.dane.onl. hm.zone02.dane.onl. 2016013100 7200
add me.zone02.dane.onl. 30      IN      TXT     "hi there"
```

Rewriting the zone file after dynamic DNS updates

- Periodically, an authoritative server writes the zone from memory to the zone file.
  - BIND writes 15 minutes after receiving an update.
  - When the primary writes the file, the format changes dramatically: the order is changed, comments are gone, and variable directives are gone.

DDNS Zone File Editing

- Manual changes to a dynamic zone file are lost when the zone file is written!
  - The changes are never used in query responses.
- Manual changes can be made by freezing a zone, however **freezing disables DDNS**!
  - Freezing should generally be avoided.
  - Use `nsupdate` to make changes to a dynamic zone.
  - At the end of the webinar we show a tool that allows editing dynamic zone files

DDNS: Freezing and Thawing

- `rndc freeze <zone>`
  - Freezing without specifying a zone, freezes all zones.
  - The journal can be deleted when the zone is frozen.
    - It contains information for IXFR, and deletion is not recommended.
- `rndc thaw <zone>`
  - A NOTIFY is sent when a zone is thawed.

Recovering from accidental edits of a dynamic zone file

- Once a dynamic zone file has been changed manually (or by a script) without using the dynamic DNS update function, BIND 9 will refuse to load the file
- There might be changes received via dynamic DNS updates while the file has been changed, creating a change conflict
- To recover:
  - Option 1: stop the BIND 9 DNS server, remove the journal file for the defective zone, restart the DNS server
  - Option 2: freeze the zone, remove the journal file for the defective zone, thaw the zone

BIND 9

Using nsupdate

The nsupdate tool to send dynamic updates to a DNS server

- `nsupdate` is a BIND application for sending dynamic changes to a server.
  - It works both interactively and non-interactively.
  - In both modes, commands are one-per-line.

How to use nsupdate

- The three main commands are:
  - `update add`
  - `update delete`
  - `send`
- `update` prepares RRs changes, `send` executes them.

nsupdate usage modes

- Non-interactively, commands come from a file or STDIN.

```
$ nsupdate nsupdate.input.file

$ nsupdate < nsupdate.input.file
```

- Interactive mode:

```
$ nsupdate
>
```

Reviewing nsupdate changes before sending

- The interactive command shown allows changes to be reviewed before being sent.
  - Here no updates have been prepared.

```
$ nsupdate
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:      0
;; flags:; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
>
```

Adding new zone content with nsupdate

- `update add` prepares a resource record to be added to a zone.
  - All fields, except the Class, must be provided.
  - TTL can be provided in a separate statement.

```
> update add a.example.com IN A 192.0.2.1
ttl 'IN': not a valid number
> update add a.example.com 3600 IN A 192.0.2.1
> ttl 3600
> update add a.example.com AAAA 2001:db8:0:deaf::1
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:      0
;; flags:; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
;; UPDATE SECTION:
a.example.com.          3600    IN      A       192.0.2.1
a.example.com.          3600    IN      AAAA    2001:db8:0:deaf::1
> send
```

Removing zone content with nsupdate

- `update delete` prepares resource records to be deleted.
  - Class is optional, and TTL, if provided, is ignored.
  - Providing all fields except TTL and Class deletes one specific resource record.

```
> update delete b.example.com. 1800 A 192.0.2.99
> update delete example.net. MX 15 mailserver.example.net.
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:      0
;; flags:; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
;; UPDATE SECTION:
b.example.com.          0       NONE    A       192.0.2.99
example.net.                0   NONE    MX      15 mailserver.example.net.
```

Removing zone content with nsupdate

- `update delete`: Providing the RTYPE without RDATA, deletes a resource record set (RRset).

```
> update delete b.example.com. AAAA
> update delete c.example.com IN SRV
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:      0
;; flags:; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
;; UPDATE SECTION:
b.example.com.          0       NONE    A       192.0.2.99
example.net.              0     NONE    MX      15 mailserver.example.net.
b.example.com.          0       ANY     AAAA
c.example.com.          0       ANY     SRV
>
```

Removing zone content with nsupdate

- `update delete`: Omitting RTYPE and RDATA deletes all resource record sets for that domain name.

```
> update delete d.example.com.
> show
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:      0
;; flags:; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
;; UPDATE SECTION:
b.example.com.          0       NONE    A       192.0.2.99
example.net.        0   NONE    MX      15 mailserver.example.net.
b.example.com.          0       ANY     AAAA
c.example.com.          0       ANY     SRV
d.example.com.          0       ANY     ANY
```

The "update" key word

- Since BIND 9.9.0, the keyword `update` is optional.

```
> delete  www.example.com. A
> add www.example.com. 600 A 192.0.2.80
> add www.example.com. 600 A 192.0.2.88
```

Sending the update to the server

- send - Transmit the prepared updates to the authoritative server.
  - **WARNING - hitting ENTER is the same as send!**

```
> send
```

Displaying the servers response

- `answer` - shows the results received from the server after a `send`.

```
> send
update failed: REFUSED
> answer
Answer:
;; ->>HEADER<<- opcode: UPDATE, status: REFUSED, id:  13117
;; flags: qr; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
>
```

Defining pre-requisites for an update

- `prereq nxdomain <domain name>` - requires that "domain name" doesn't exist (no resource records of any type).

```
> prereq nxdomain www.example.com
> update add www.example.com 3600 CNAME example.com.
> send
```

Checking if a domain name already exists

- `prereq yxdomain <domain name>` - requires that "domain name" does exist.

```
> prereq yxdomain www.example.com
> update add web.example.com 3600 CNAME www.example.com.
> send
```

Send updates to a specific server

- `server <servername or IP> [port]` - specifies the authoritative server that will receive the update.
  - If not set, `nsupdate` uses `MNAME` from the SOA RR and port 53.
  - Generally, updates should be sent to the primary authoritative server.

```
> server ns0.example.com
```

Specify the local IP address for sending updates

- `local <IP address> [port]` - specifies the outgoing IP address. This is useful for multi-homed machines, especially when IPv6 is used, and when the server limits updates to specific IPs.

```
> local 2001:db8:100:0:ff:aa01:42:12:feaa
```

Send updates to a specific zone

- `zone <zonename>` - specify the zone to be changed.
  - This is necessary when the update is an NS or glue resource record to be made in the parent zone.

```
$ nsupdate
> zone example.com.
> update add ns1.subdomain.example.com. 3600 IN A 192.0.2.53
> update add ns2.subdomain.example.com. 3600 IN A 203.0.113.53
> send
```

DNS NOTIFY

What is DNS NOTIFY?

- NOTIFY allows a primary authoritative server to inform its secondaries when a zone changes.
  - The secondary accepts NOTIFY as a "pop" of the refresh timer.
- Therefore secondary servers typically transfer a zone far more frequently.
  - For large zones, this generates a huge amount of traffic.
  - For a very large zone, a transfer might not complete before the next dynamic update and the next NOTIFY!

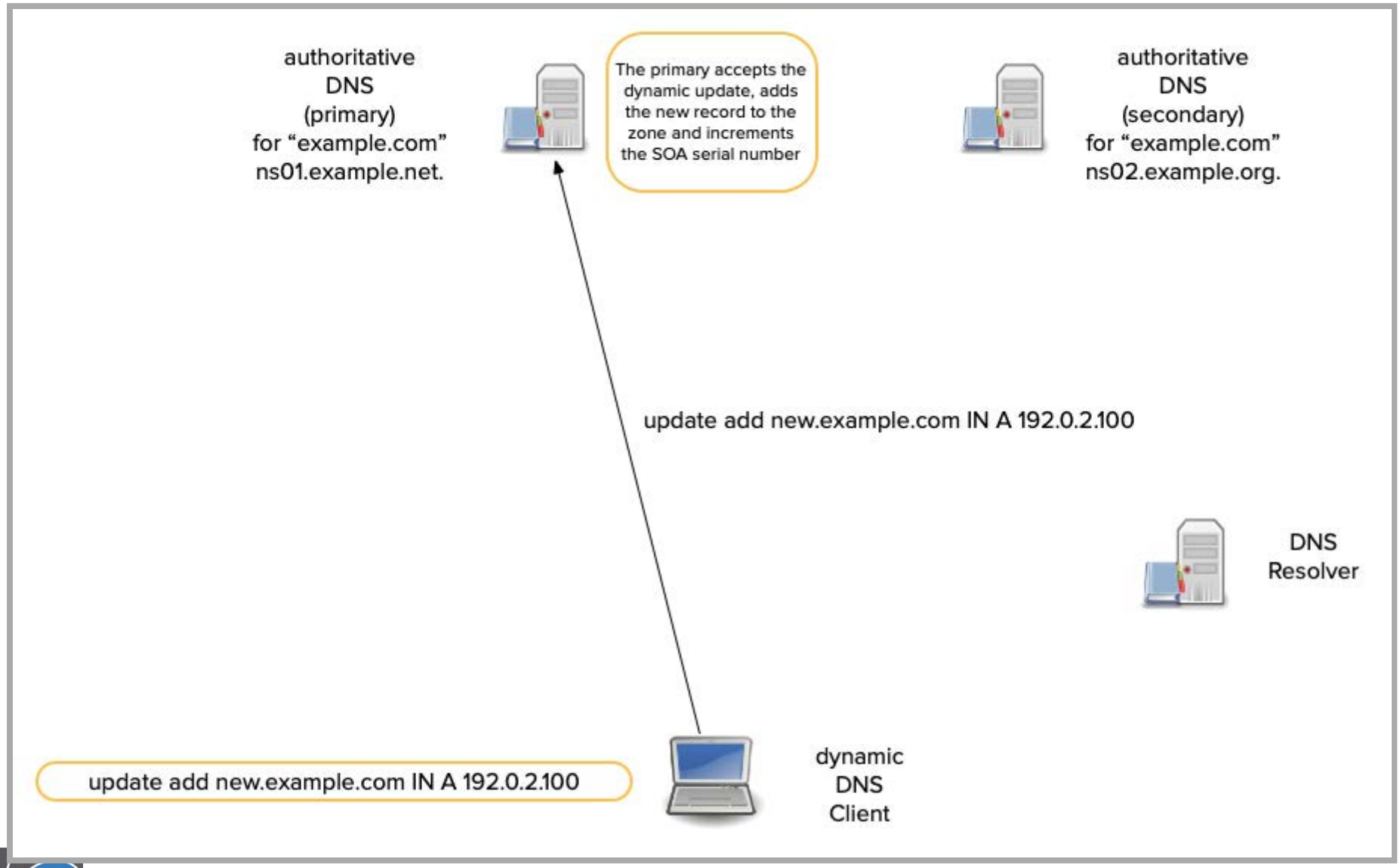
What is DNS NOTIFY?

- Each update to a zone can increment the serial number thereby triggering a NOTIFY.
- A server with the primary zone determines which servers are secondaries by looking at the zone's NS RRset.
  - A name in the NS RRset that matches the `MNAME`, is not sent a NOTIFY.

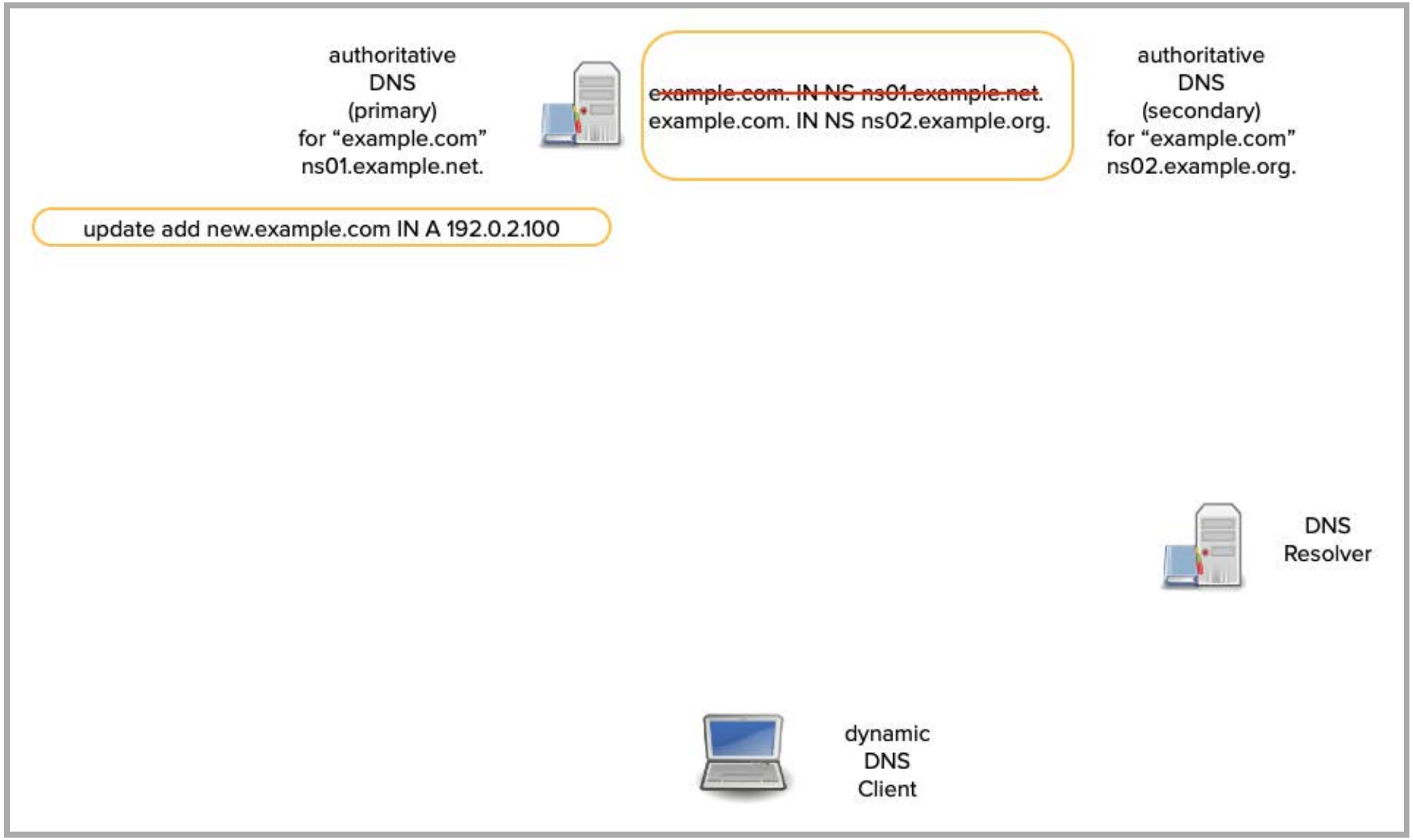
How does DNS notify work?

- When a secondary receives a NOTIFY, it checks:
  - that it is for a zone for which it is authoritative.
  - that it came from a primary server for the zone.
  - that the SOA serial has increased.
- If the NOTIFY is accepted, the secondary immediately schedules a refresh of the zone.
- The secondary sends a NOTIFY to the other secondary servers.

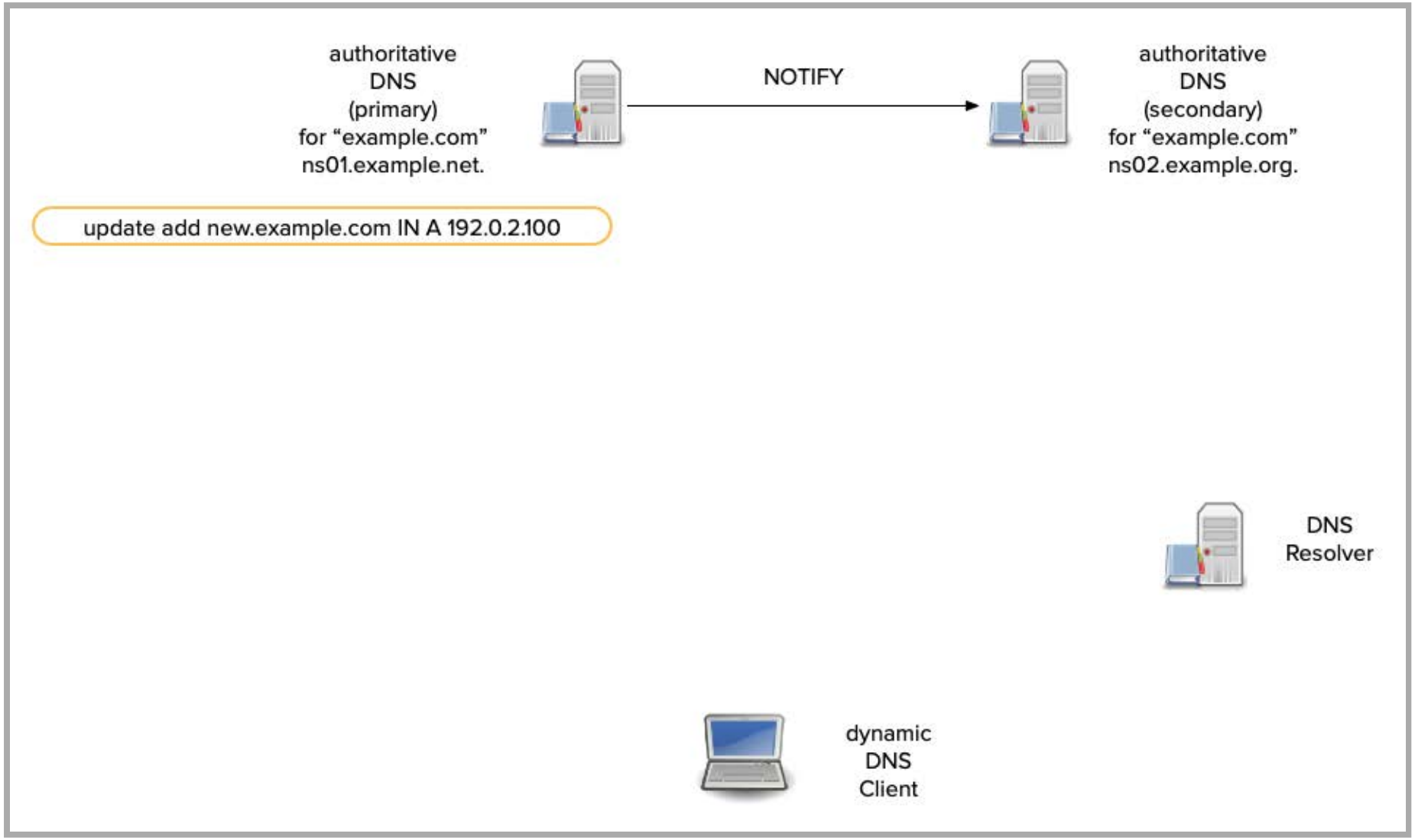
How does DNS NOTIFY work?



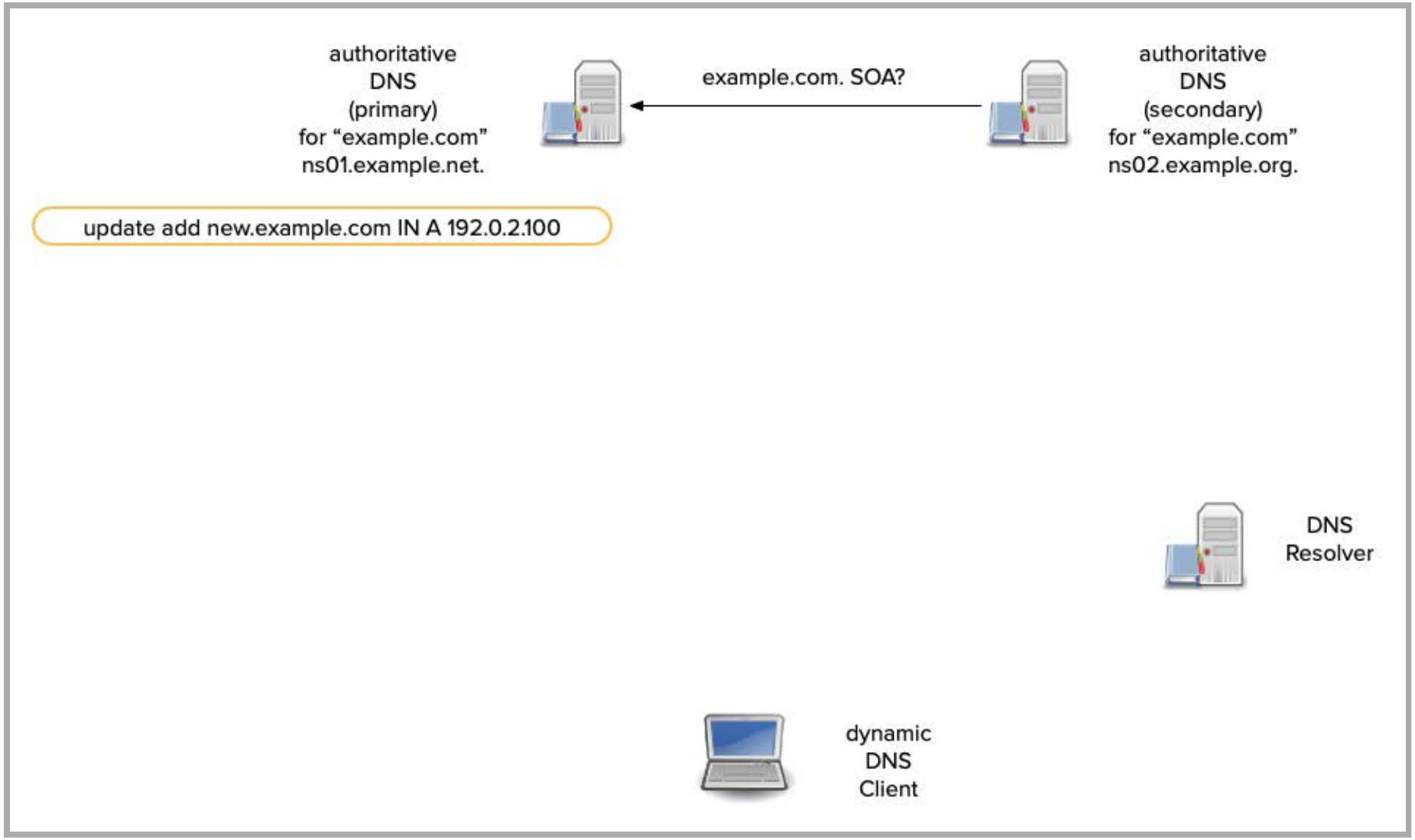
How does DNS NOTIFY work?



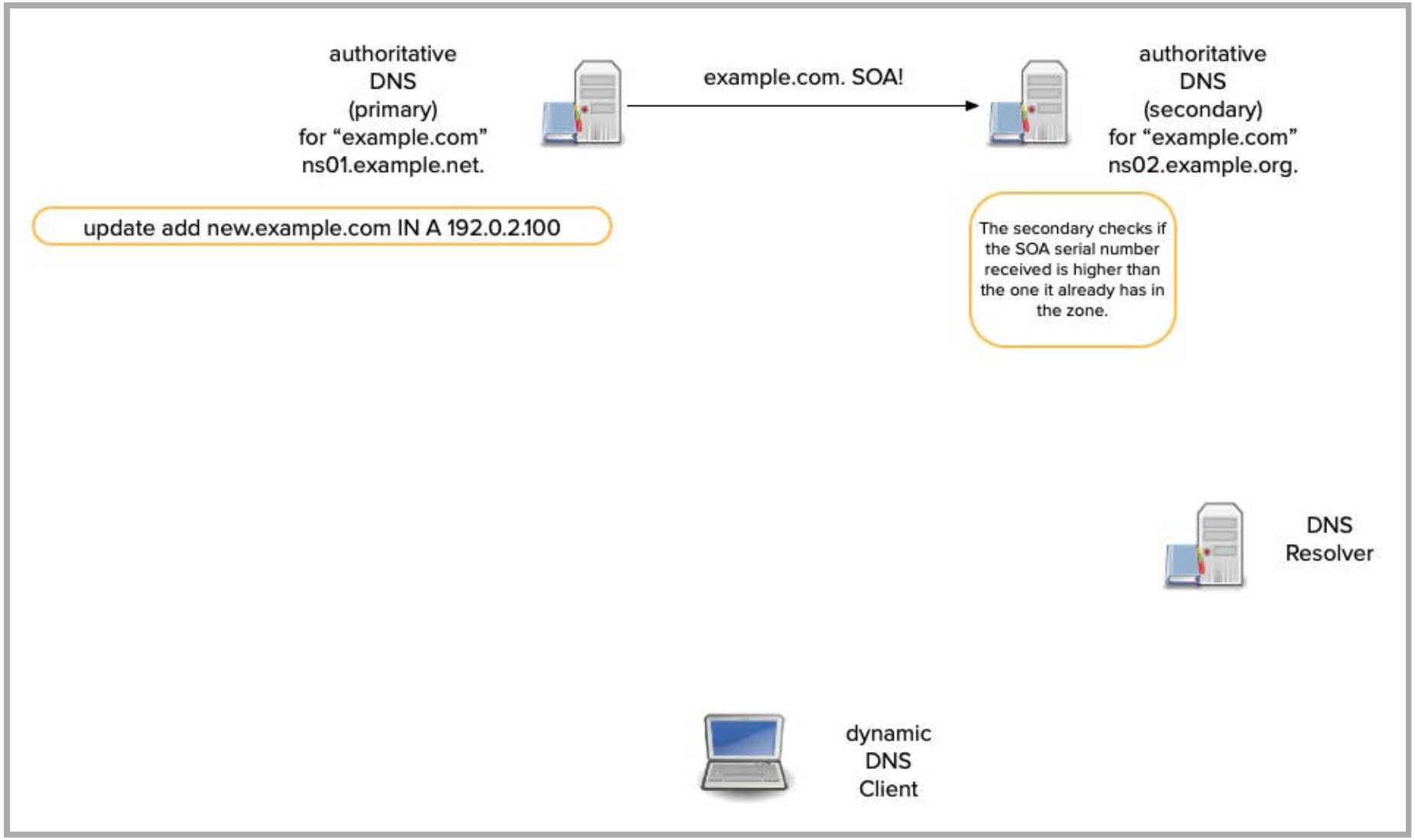
How does DNS NOTIFY work?



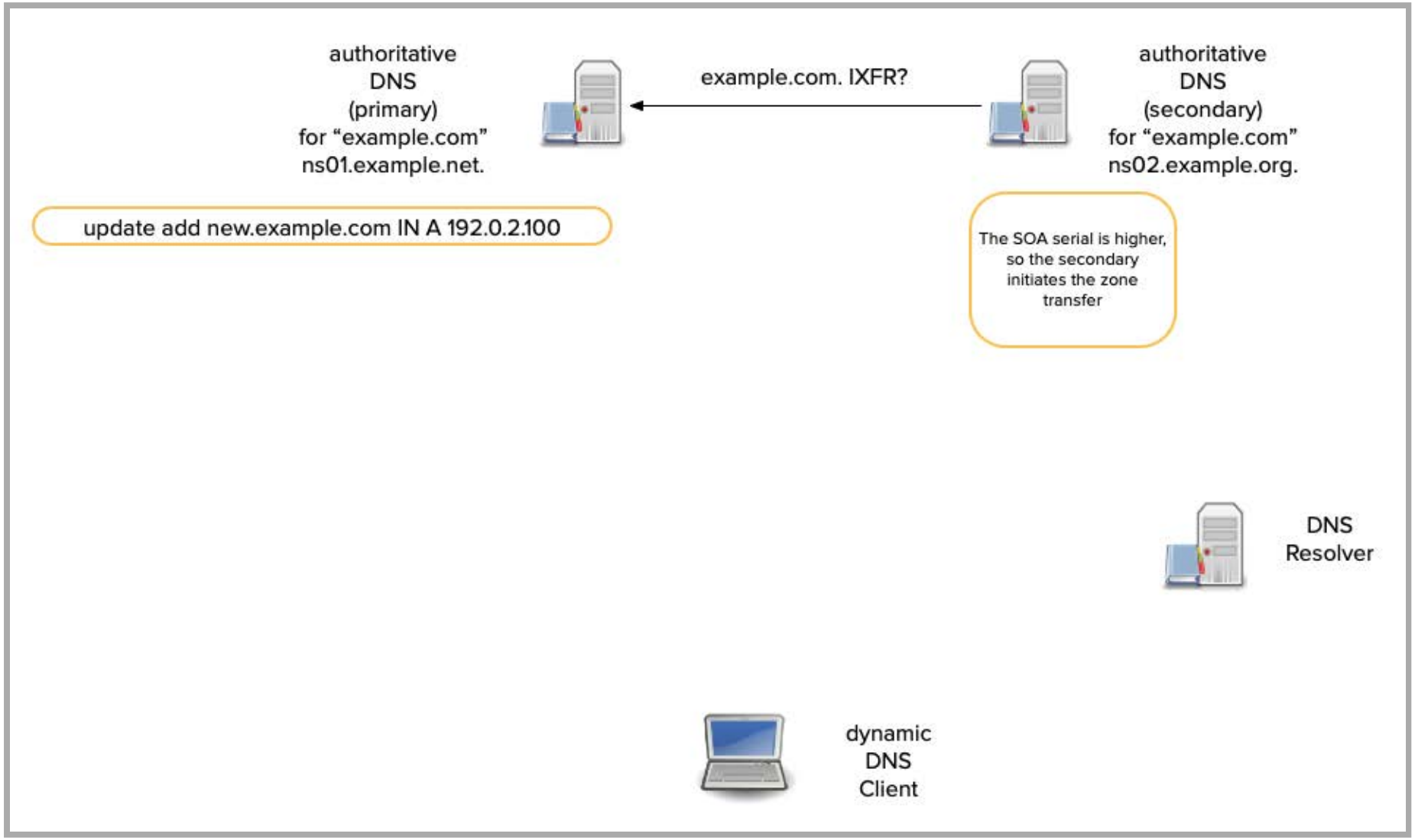
How does DNS NOTIFY work?



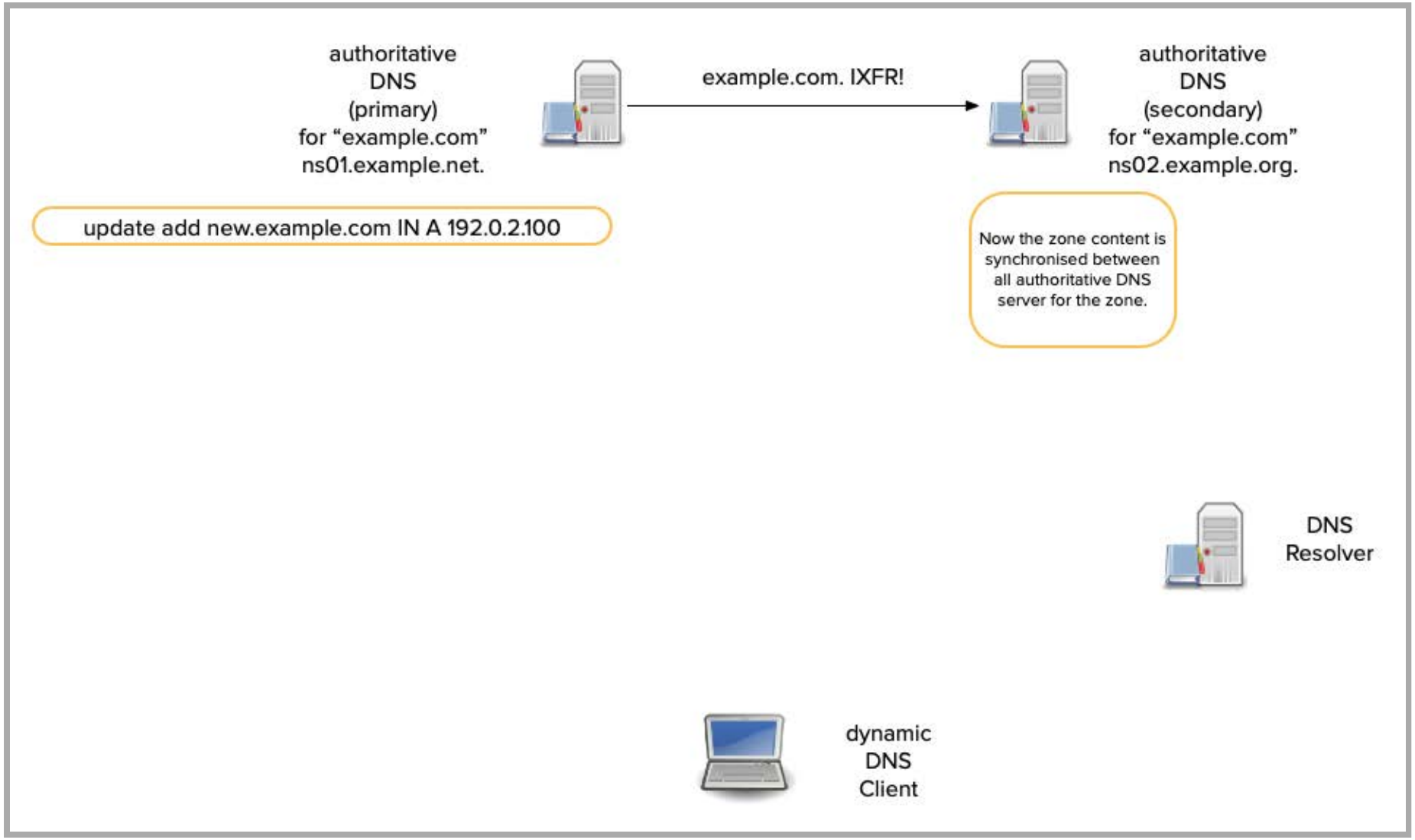
How does DNS NOTIFY work?



How does DNS NOTIFY work?



How does DNS NOTIFY work?



Configuring NOTIFY

- For primaries and secondaries, `NOTIFY` is on by default.
- For a secondary not used as a downstream primary, it can be disabled:

```
options {
  NOTIFY no;
};
```

- To disable `NOTIFY` for a particular zone:

```
zone "example.com";
        type secondary;
        masters { 192.0.2.110; };
        file "bak.example.com";
        NOTIFY no;
};
```

Extra targets for NOTIFY

- Adding an address to the `NOTIFY` list for all zones:

```
options {
      also-NOTIFY { 192.0.2.120; };
};
```

- Adding an address to the NOTIFY list for a particular zone:

```
zone "example.com" {
       type primary;
       file "example.com";
       also-NOTIFY { 192.0.2.120; };
};
```

Explicit NOTIFY

- To NOTIFY only name servers listed in `also-NOTIFY`:

```
options {
        also-NOTIFY { 192.0.2.120; };
        NOTIFY explicit;
};
```

- To NOTIFY only name servers listed in `also-NOTIFY`:

```
zone "example.com" {
        type primary;
        file "example.com";
        also-NOTIFY { 192.0.2.120; };
        NOTIFY explicit;
};
```

Incremental Zone Transfer (IXFR)

What is incremental zone transfer

- An Incremental Zone Transfer (IXFR) allows a client to query just the changes since a previous version.
  - This generally reduces the size (and duration) of the transfer considerably.
  - Like AXFRs (full zone transfer), IXFRs are sent to authoritative servers.
  - BIND secondaries and primary servers request and provide IXFRs by default.

What is incremental zone transfer

- A server querying an IXFR sends a previous serial number for the zone.
  - For a secondary, this is its current serial number.
  - Secondary: "send me changes since version 14284171."
- The queried server responds with all changes since the IXFR's serial number.

IXFR example

- The next examples show BIND's response to an IXFR after the following update.
  - RFC 1995 allows several formats for the IXFR response

```
$ dig +short soa dane.onl
authoritative.dane.onl. hostmaster.dane.onl. 2013081127 7200 1800 3542400 30

% nsupdate -l
> update add zzz.dane.onl 30 TXT "Zees"
> send
> quit
```

IXFR example

- Query with a known previous serial number.
  - IXFR responses always start with the current (newest) SOA RR. This is the same as AXFR.
  - IXFR responses always end with the current (newest) SOA RR. This is the same as AXFR.

```
$ dig +noall +answer dane.onl ixfr=2013081127 @::1 | grep -Ev 'RRSIG|NSEC'
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081128
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081127
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081128
zzz.dane.onl.           30      IN      TXT     "Zees"
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081128
```

- The second RR is always a SOA to delete. It is followed by additional RRs to delete (none in this case.)
  - This section continues until another SOA RR.

IXFR example

- After a section of deletes, comes a sections of adds that always begins with a SOA.
  - In this case there are two additions.

```
$ dig +noall +answer dane.onl ixfr=2013081127 @::1 | grep -Ev 'RRSIG|NSEC'
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081128
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081127
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081128


zzz.dane.onl.           30      IN      TXT     "Zees"
dane.onl.               40      IN      SOA     authoritative.dane.onl. hostmaster.dane.onl. 2013081128
```

IXFR example

- The following example shows how BIND responds to an IXFR covering several versions.
- Example: A zone is at version 2. Not including SOA changes:
  - One RR is added. Now version 3.
  - One RR is deleted. Two added. Now version 4.
  - Two RR deleted. Now version 5.
  - Only the SOA is changed. Now version 6.

```
dig @<Auth-Server> <Domain-Name>  ixfr=2

FQDN [...]  SOA 6 7200 3600 5184000 30
FQDN [...]  SOA 2 [...]
FQDN [...]  SOA 3 [...]
FQDN [...]
FQDN [...]  SOA 3 [...]
FQDN [...]
FQDN [...]  SOA 4 [...]
FQDN [...]
FQDN [...]
FQDN [...]  SOA 4 [...]
FQDN [...]
FQDN [...]
FQDN [...]  SOA 5 [...]
FQDN [...]  SOA 5 [...]
FQDN [...]  SOA 6 [...]
FQDN [...]  SOA 6 7200 3600 5184000 30
```

The journal and IXFR

- BIND authoritative servers use the journal to identify the changes necessary to respond to an IXFR.
  - A non-dynamic zone, doesn't have a journal.
  - Use `ixfr-from-differences yes;` in a zone stanza to create a journal for a non-dynamic zone.

Editing dynamic zones with an editor

The dynamic update dilemma

- Many DNS administrators love editing zone files with their favorite editor
  - some have elaborate scripts and/or macros to help with DNS zone edits
- **but**: dynamic zones cannot be changed with an editor, it will break the dynamic update setup

nsdiff to the rescue

- Tony Finch has created a set of Perl tools that give you the best of both worlds
  - dynamic zones that you can edit with your favorite text editor!
  - Automatically-maintained DNSSEC records are stripped from the zone before it is passed to the editor, and you do not need to manually adjust the SOA serial number
- `nsdiff` / `nspatch` / `nsvi` Homepage: https://dotat.at/prog/nsdiff/

How nsvi works

1. `nsvi` first loads the zone content from the primary authoritative DNS server of the zone
2. it strips the DNSSEC records (DNSKEY, RRSIG, NSEC or NSEC3) that should not be changed manually
3. It launches the configured editor (or `vi`) on the zone content
4. Once the zone has been saved and the editor has been exited, `nsvi` will check the zone with `named-checkzone`
5. If there are no errors, it will calculate the differences between the old and the new zone, translates this differences into commands for `nsupdate` and sends dynamic updates to the primary DNS server

Additional Information

- RFC 1995 Incremental Zone Transfer in DNS
  https://datatracker.ietf.org/doc/html/rfc1995
- RFC 1996 Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)
  https://datatracker.ietf.org/doc/html/rfc1996
- RFC 2136 Dynamic Updates in the Domain Name System (DNS UPDATE)
  https://datatracker.ietf.org/doc/html/rfc2136

Upcoming Webinars

- June 16: Session 5. Dynamic zones, pt2 - Advanced topics
  - TSIG security
  - ACLs for dynamic zones with *update-policy* and the *grant* keyword
  - Combining dynamic updates and *Catalog Zones*

Questions and Answers

Hands-On

- We have prepared a VM machine for every participant
- This time the sessions build upon each other and needs to be done in order
- find the instructions at https://webinar.defaultroutes.de/webinar/04-ddns-workshop.html