This time

Digging into

Networking Protocols

Naming

DNS & DHCP

Naming

- IP addresses allow global connectivity
- But they’re pretty useless for humans!
  - Can’t be expected to pick their own IP address
  - Can’t be expected to remember another’s IP address
- **DHCP** : Setting IP addresses
- **DNS** : Mapping a memorable name to a routable IP address

DHCP

Dynamic Host Configuration Protocol

New host

DHCP server

DHCP

Dynamic Host Configuration Protocol

New host

DHCP server

Doesn’t have an
IP address yet
(can’t set src addr)

DHCP

Dynamic Host Configuration Protocol

New host | DHCP server

Doesn’t have an
IP address yet
(can’t set src addr)

Doesn’t know *who*
to ask for one

DHCP

Dynamic Host Configuration Protocol

New host

DHCP server

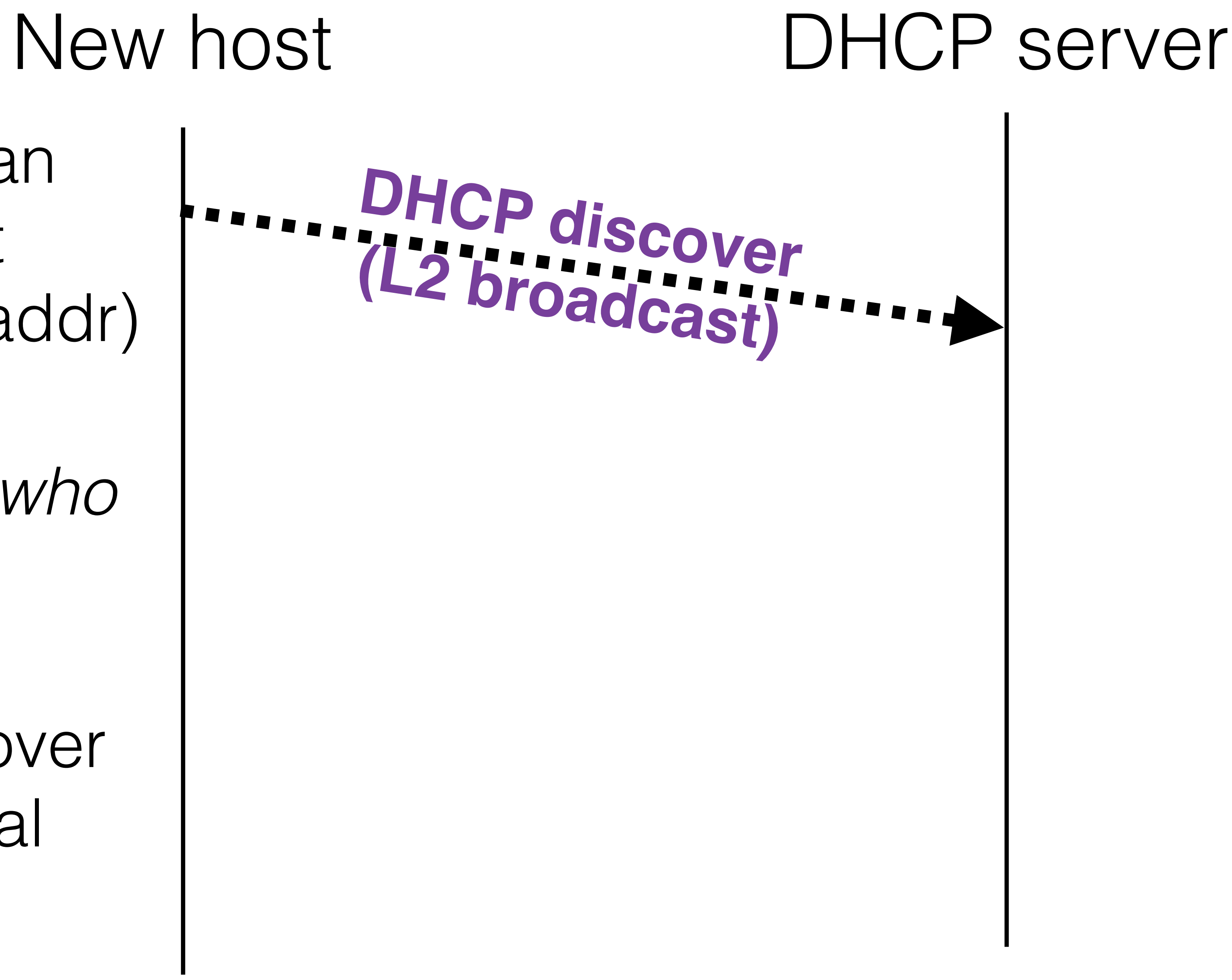
Doesn't have an IP address yet (can't set src addr)

Doesn't know *who* to ask for one

Solution: Discover one on the local subnet

DHCP

Dynamic Host Configuration Protocol



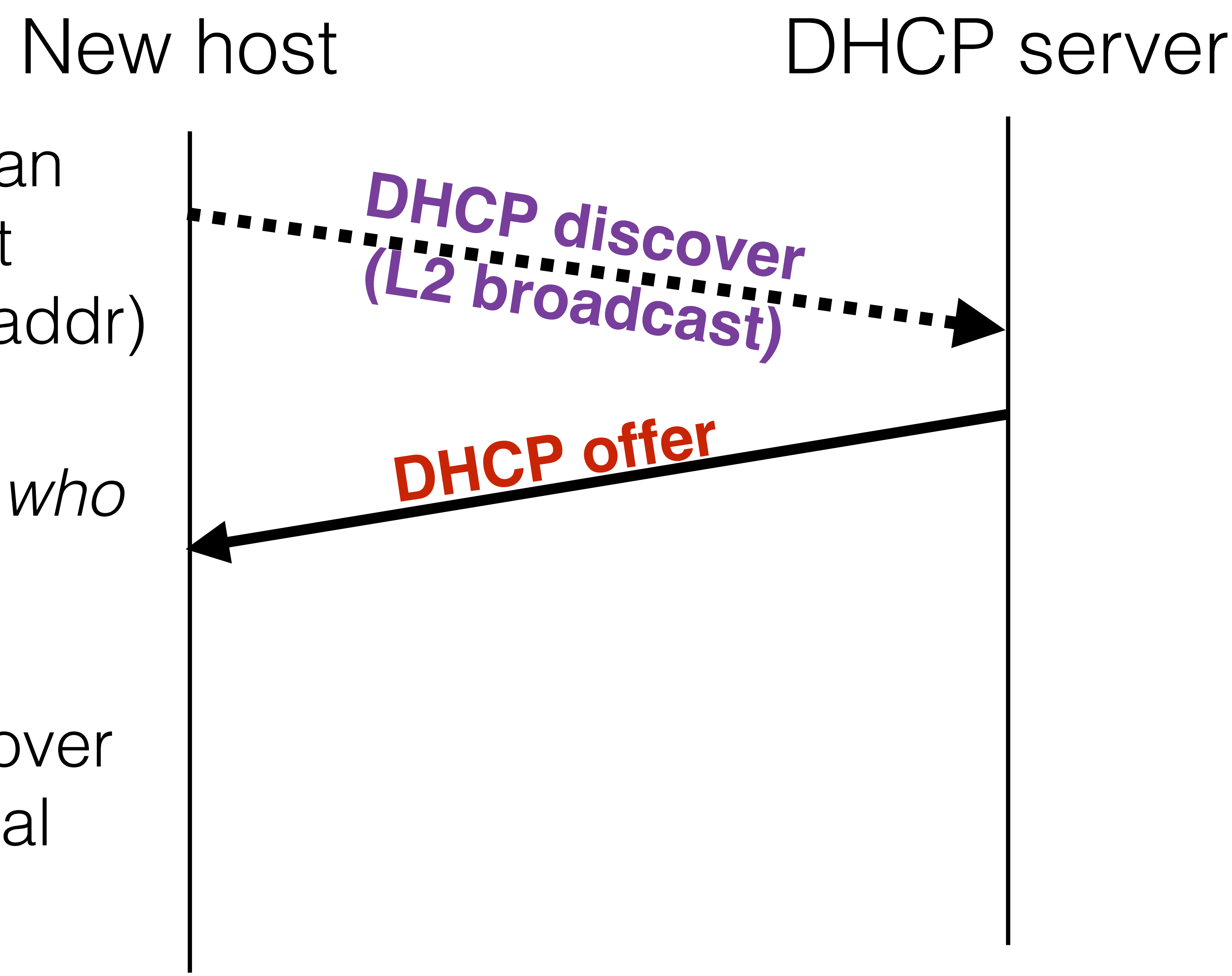
Doesn't have an IP address yet (can't set src addr)

Doesn't know *who* to ask for one

Solution: Discover one on the local subnet

DHCP

Dynamic Host Configuration Protocol



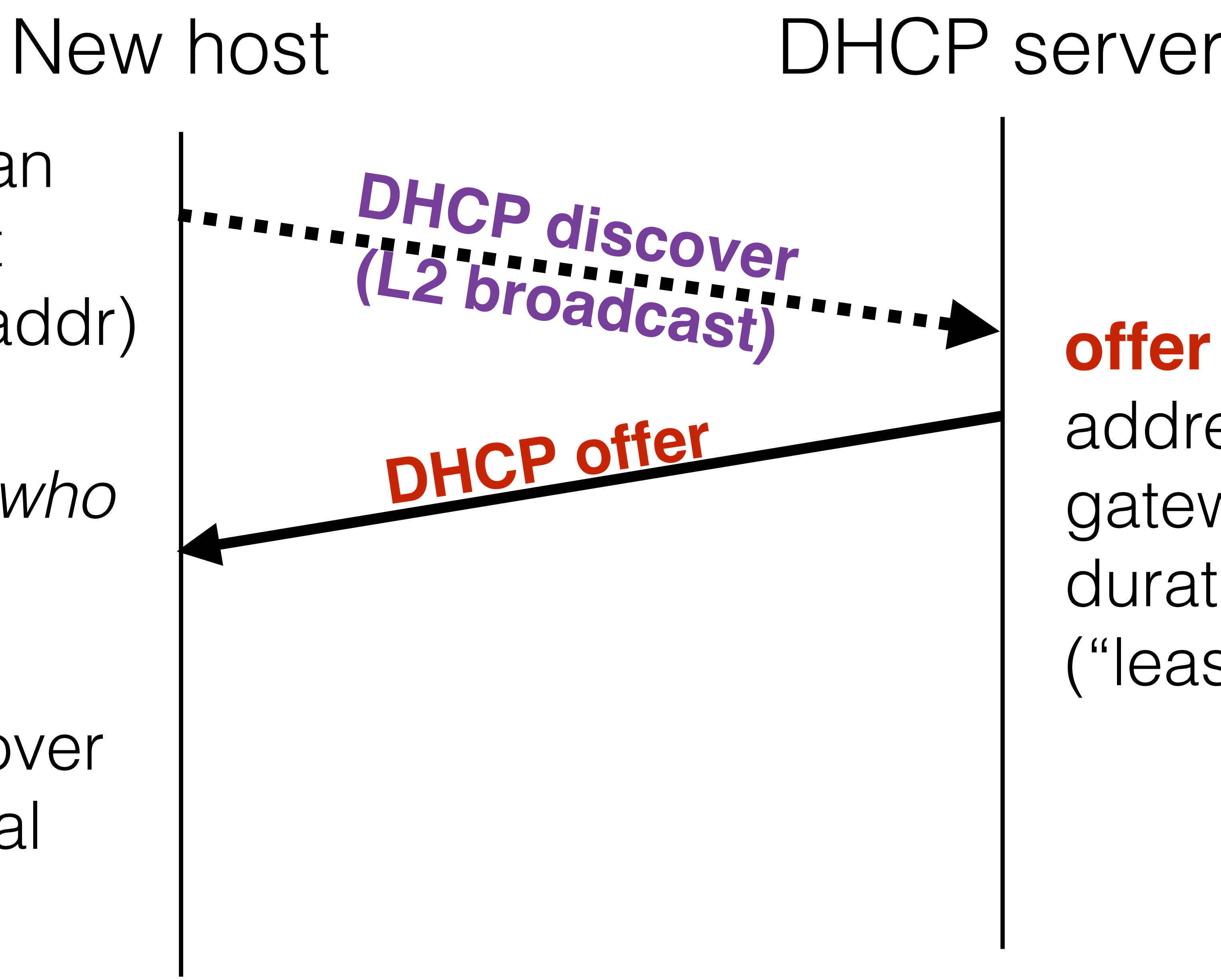
Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know *who* to ask for one

Solution: Discover one on the local subnet

DHCP

Dynamic Host Configuration Protocol



Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know *who* to ask for one

Solution: Discover one on the local subnet

offer includes: IP address, DNS server, gateway router, and duration of this offer (“lease” time)

DHCP

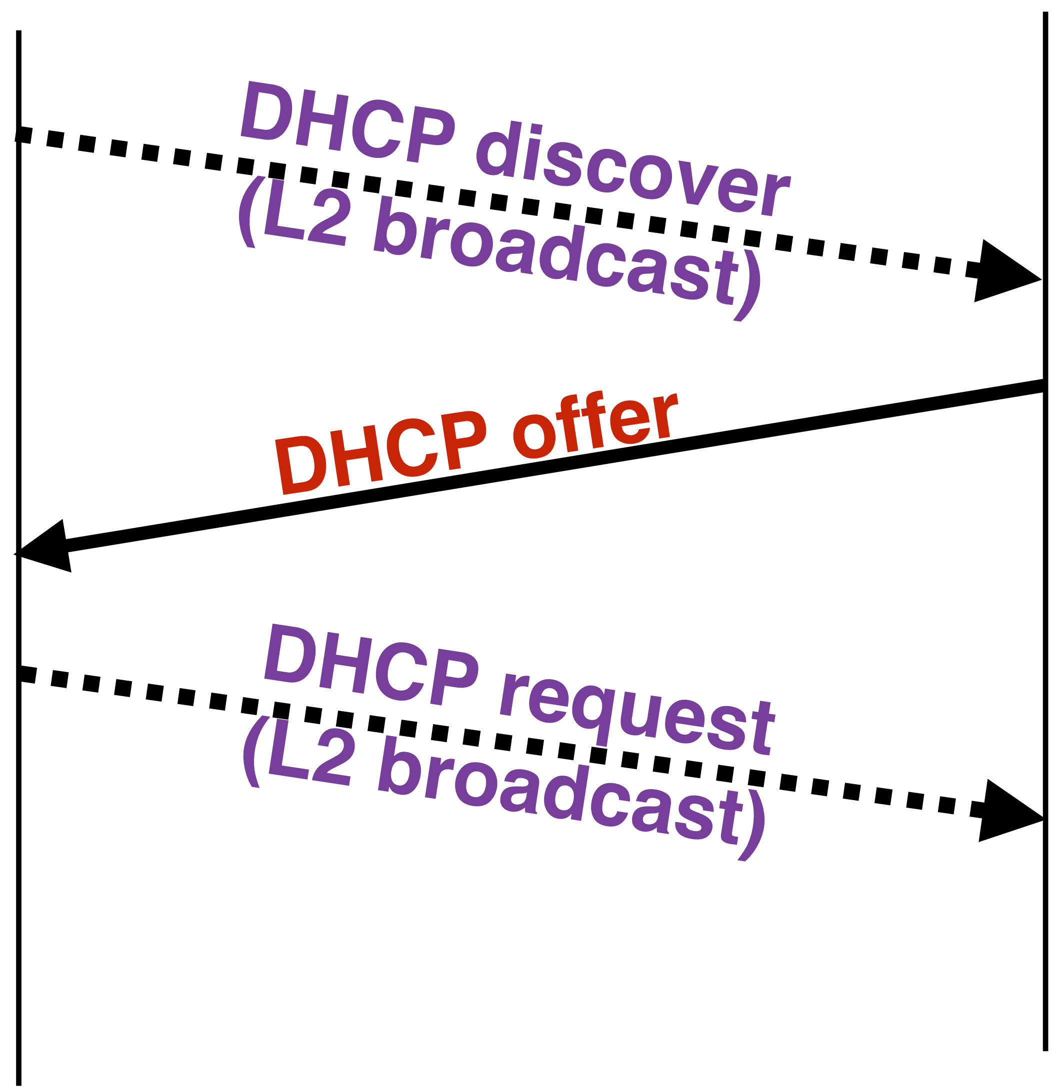
Dynamic Host Configuration Protocol

New host DHCP server

Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know *who* to ask for one

Solution: Discover one on the local subnet



offer includes: IP address, DNS server, gateway router, and duration of this offer (“lease” time)

DHCP

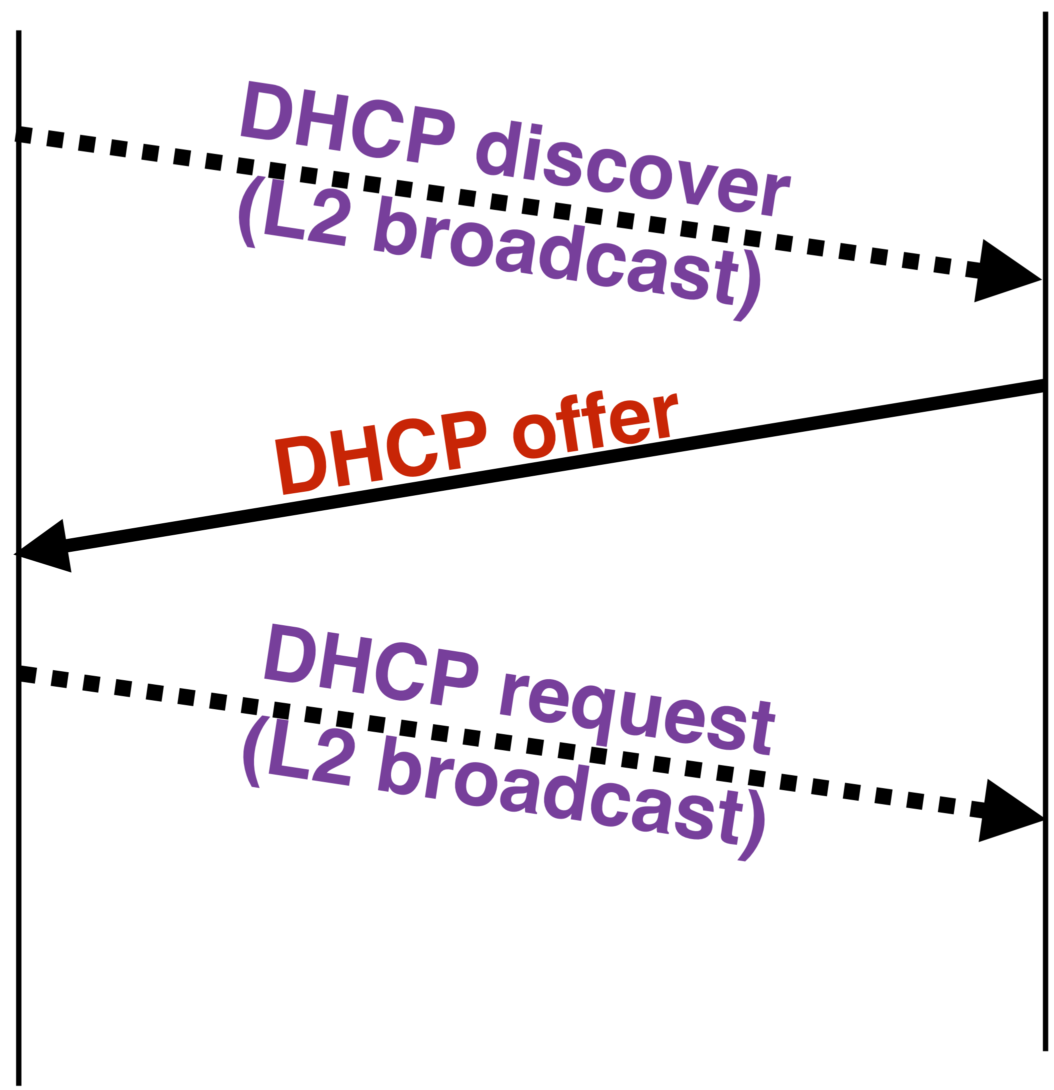
Dynamic Host Configuration Protocol

New host DHCP server

Doesn't have an IP address yet (can't set src addr)

Doesn't know *who* to ask for one

Solution: Discover one on the local subnet



offer includes: IP address, DNS server, gateway router, and duration of this offer ("lease" time)

request asks for the offered IP address

DHCP

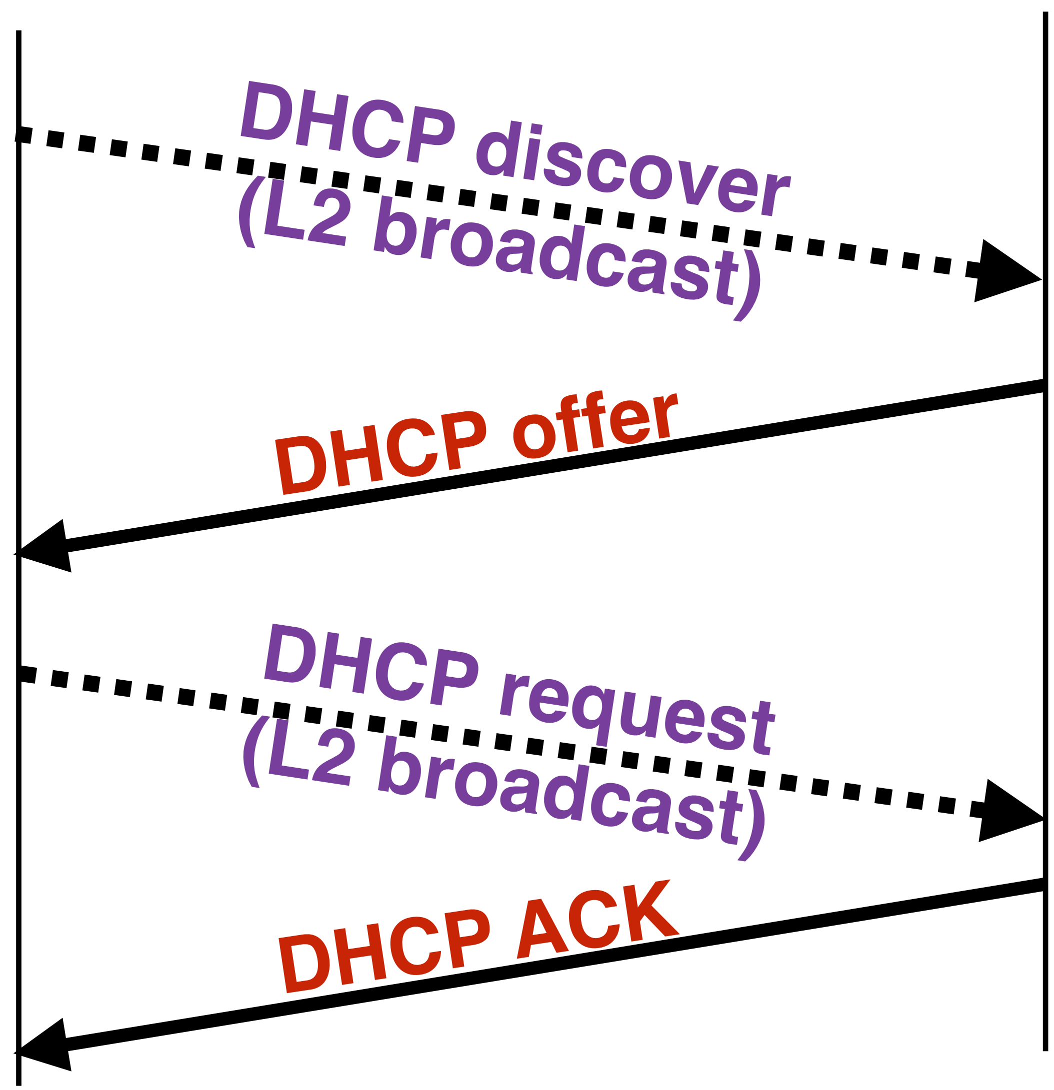
Dynamic Host Configuration Protocol

New host DHCP server

Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know *who* to ask for one

Solution: Discover one on the local subnet



offer includes: IP address, DNS server, gateway router, and duration of this offer (“lease” time)

request asks for the offered IP address

DHCP attacks

- Requests are broadcast: attackers on the same subnet can hear new host's request

- Race the *actual* DHCP server to replace:
    - DNS server
        - Redirect any of a host's lookups ("what IP address should I use when trying to connect to google.com?") to a machine of the attacker's choice
    - Gateway
        - The gateway is where the host sends all of its outgoing traffic (so that the host doesn't have to figure out routes himself)
        - Modify the gateway to intercept all of a user's traffic
        - Then relay it to the gateway (MITM)
        - How could the user detect this?

Hostnames & IP addresses

```
gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
```

Hostnames & IP addresses

```
gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
```

Hostnames & IP addresses

```
gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
```

Hostnames & IP addresses

```
gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
```

google.com is easy to remember, but not routable

74.125.228.65 is routable

Name resolution:
The process of mapping from one to the other

Terminology

- www.cs.umd.edu = "**domain name**"
    - www.cs.umd.edu is a "subdomain" of cs.umd.edu

- Domain names can map to a set of IP addresses

```
gold:~ dml$ dig google.com

; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35815
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 0, ADDITIONAL: 0

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

;; ANSWER SECTION:
google.com.       105  IN   A    74.125.228.70
google.com.       105  IN   A    74.125.228.66
google.com.       105  IN   A    74.125.228.64
google.com.       105  IN   A    74.125.228.69
google.com.       105  IN   A    74.125.228.78
google.com.       105  IN   A    74.125.228.73
google.com.       105  IN   A    74.125.228.68
google.com.       105  IN   A    74.125.228.65
google.com.       105  IN   A    74.125.228.72
```

We'll understand this more in a bit; for now, note that google.com is mapped to many IP addresses

Terminology

- www.cs.umd.edu = "**domain name**"
  - www.cs.umd.edu is a "subdomain" of cs.umd.edu

- Domain names can map to a set of IP addresses

```
gold:~ dml$ dig google.com

; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35815
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 0, ADDITIONAL: 0

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

;; ANSWER SECTION:
google.com.       105  IN    A    74.125.228.70
google.com.       105  IN    A    74.125.228.66
google.com.       105  IN    A    74.125.228.64
google.com.       105  IN    A    74.125.228.69
google.com.       105  IN    A    74.125.228.78
google.com.       105  IN    A    74.125.228.73
google.com.       105  IN    A    74.125.228.68
google.com.       105  IN    A    74.125.228.65
google.com.       105  IN    A    74.125.228.72
```

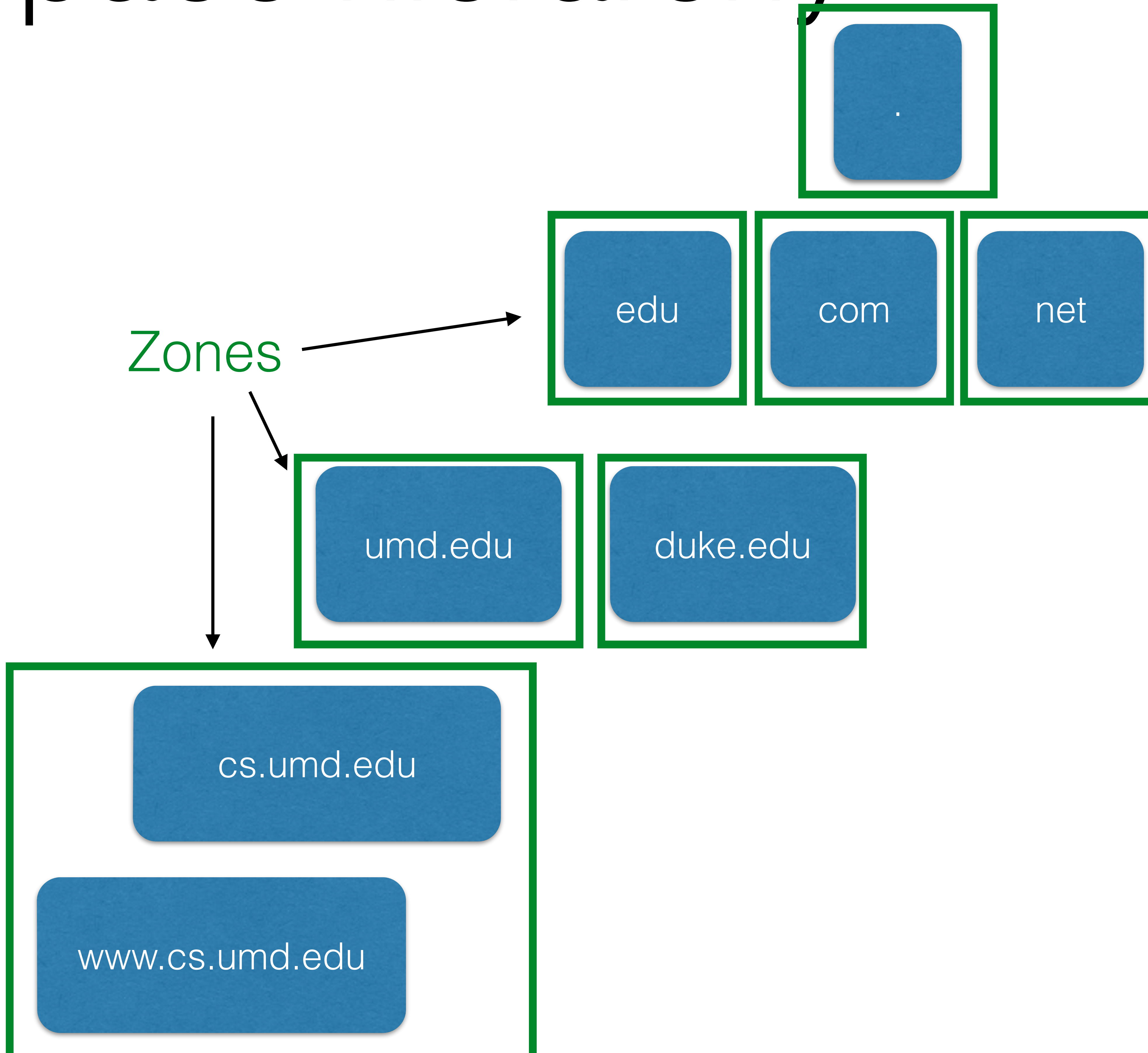
We'll understand this more in a bit; for now, note that google.com is mapped to many IP addresses

Terminology

- "**zone**" = a portion of the DNS namespace, divided up for administrative reasons
  - Think of it like a collection of hostname/IP address pairs that happen to be lumped together
    - www.google.com, mail.google.com, dev.google.com, …

- Subdomains do not need to be in the same zone
  - Allows the owner of one zone (umd.edu) to delegate responsibility to another (cs.umd.edu)

Namespace hierarchy



Terminology

- “**Nameserver**” = A piece of code that answers queries of the form “What is the IP address for foo.bar.com?”
  - Every zone must run ≥2 nameservers
  - Several very common nameserver implementations: BIND, PowerDNS (more popular in Europe)

- “**Authoritative nameserver**”:
  - Every zone has to maintain a file that maps IP addresses and hostnames (“www.cs.umd.edu is 128.8.127.3”)
  - One of the name servers in the zone has the *master* copy of this file. It is the authority on the mapping.

Terminology

- “**Resolver**” - while name servers *answer* queries, resolvers *ask* queries.

- Every OS has a resolver. Typically small and pretty dumb. All it typically does it forward the query to a local…

- “**Recursive nameserver**” - a nameserver which will do the heavy lifting, issuing queries on behalf of the client resolver until an authoritative answer returns.

- Prevalence
  - There is almost always a *local* (private) recursive name server
  - But very rare for name servers to support recursive queries otherwise

Terminology

- "**Record**" (or "resource record") = usually think of it as a mapping between hostname and IP address

- But more generally, it can map virtually anything to virtually anything

- Many record types:
  - (**A**)ddress records (IP <-> hostname)
  - Mail server (**MX**, mail exchanger)
  - SOA (start of authority, to delineate different zones)
  - Others for DNSSEC to be able to share keys

- Records are the unit of information

Terminology

Nameservers within a zone must be able to give:

- **Authoritative answers (A)** for hostnames in that zone
  - The umd.edu zone's nameservers must be able to tell us what the IP address for umd.edu is

"A" record: umd.edu = 54.84.241.99

54.84.241.99 is a valid IP address for umd.edu

Terminology

Nameservers within a zone must be able to give:

- **Authoritative answers (A)** for hostnames in that zone
  - The umd.edu zone's nameservers must be able to tell us what the IP address for umd.edu is

"A" record: umd.edu = 54.84.241.99

54.84.241.99 is a valid IP address for umd.edu

- Pointers to **name servers (NS)** who host zones in its subdomains
  - The umd.edu zone's nameservers must be able to tell us what the name and IP address of the cs.umd.edu zone's nameservers

"NS" record: cs.umd.edu = ipa01.cs.umd.edu.

Ask ipa01.cs.umd.edu for all cs.umd.edu subdomains

DNS

Domain Name Service at a very high level

Requesting host

What is an IP address for cs.umd.edu?

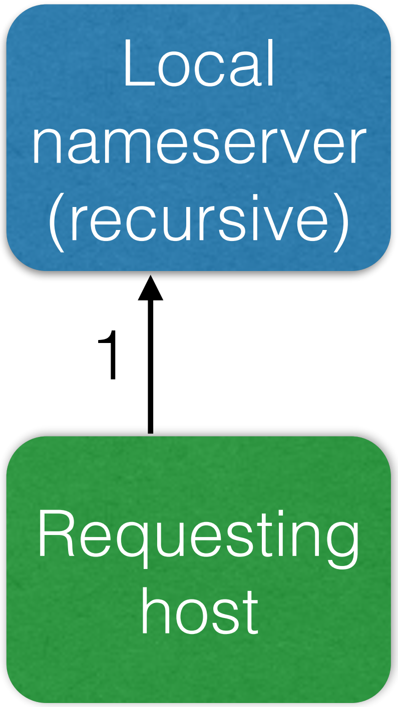
DNS

Domain Name Service at a very high level

What is an IP address for cs.umd.edu?

DNS

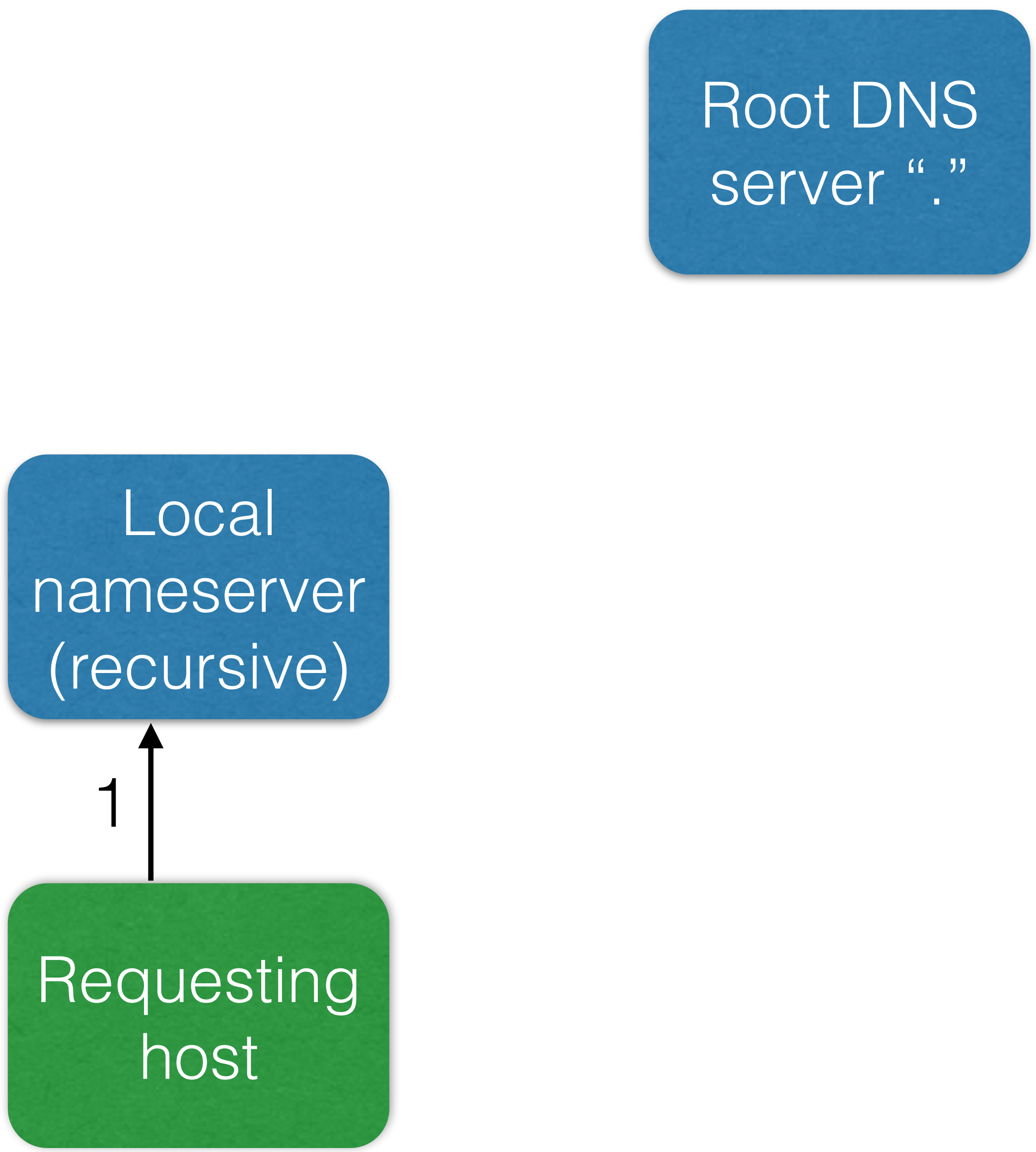
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

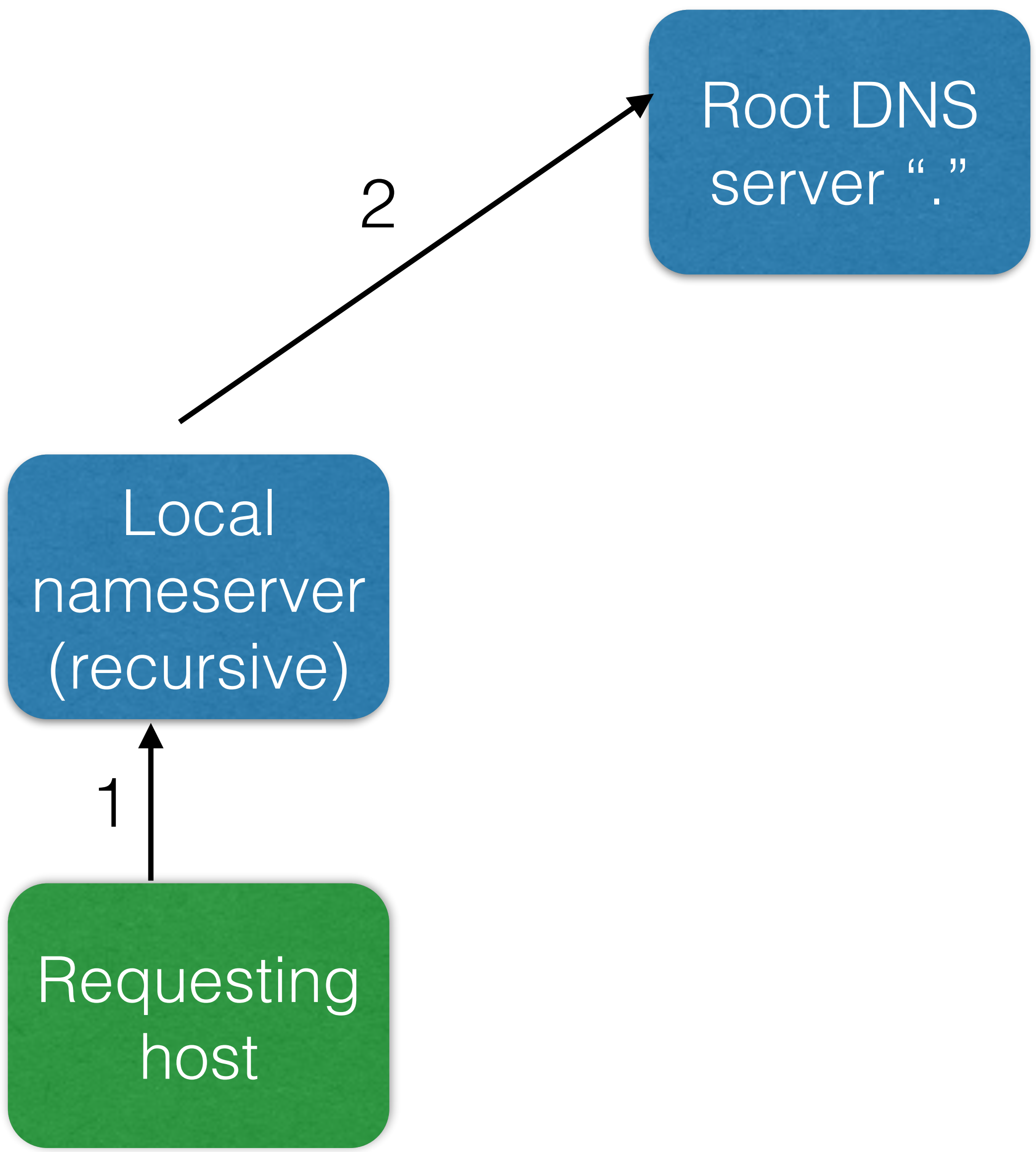
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

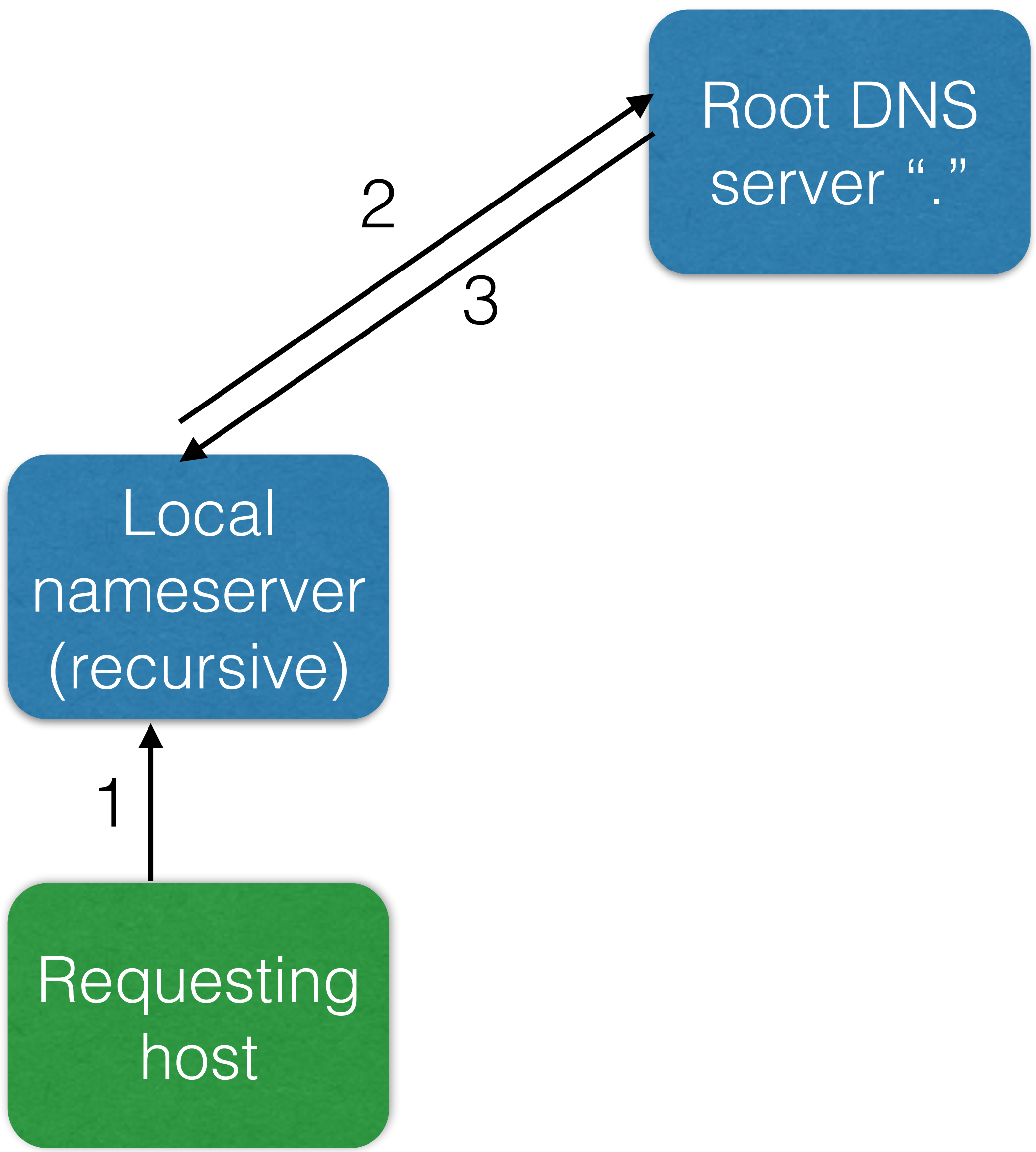
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

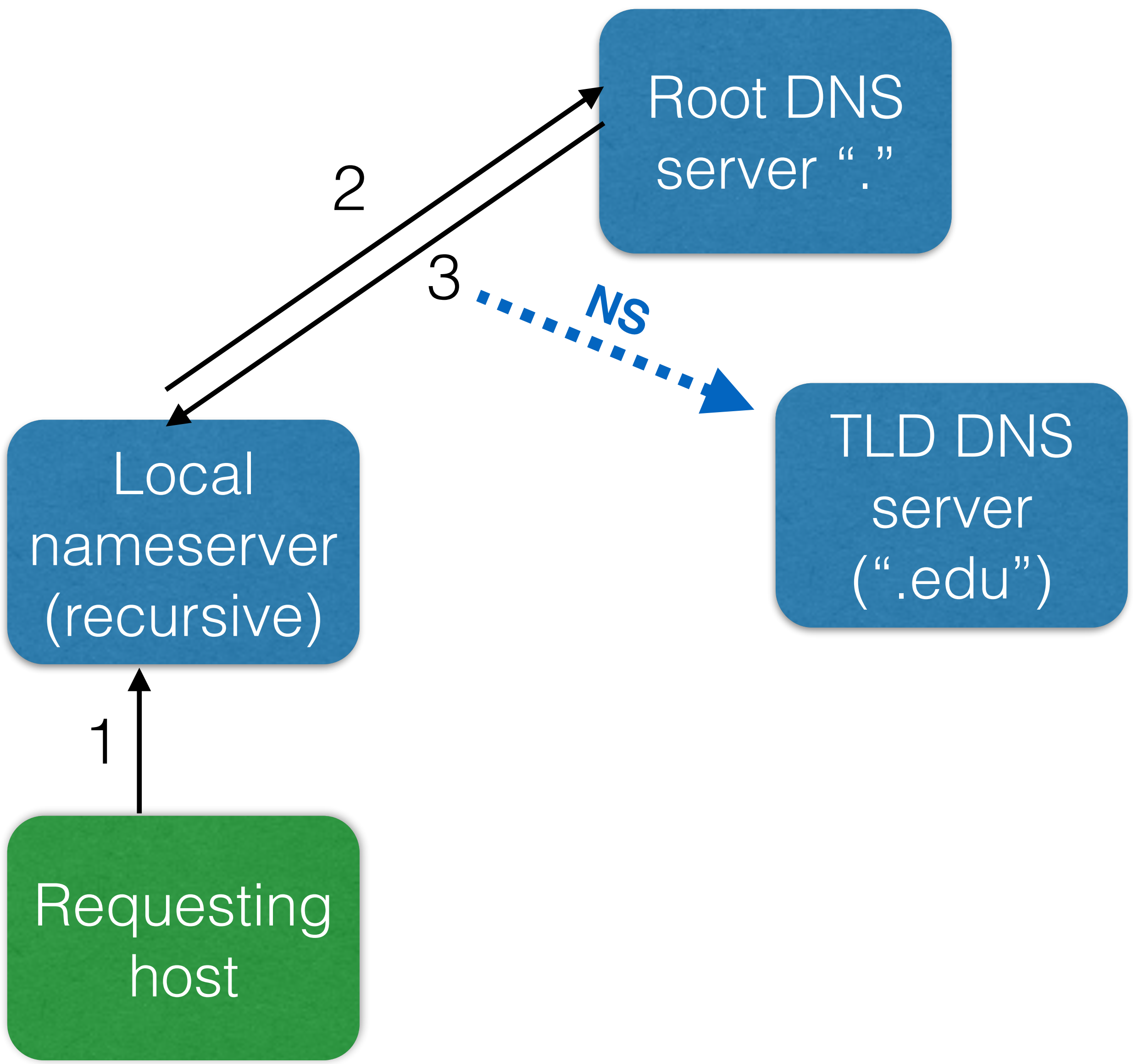
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

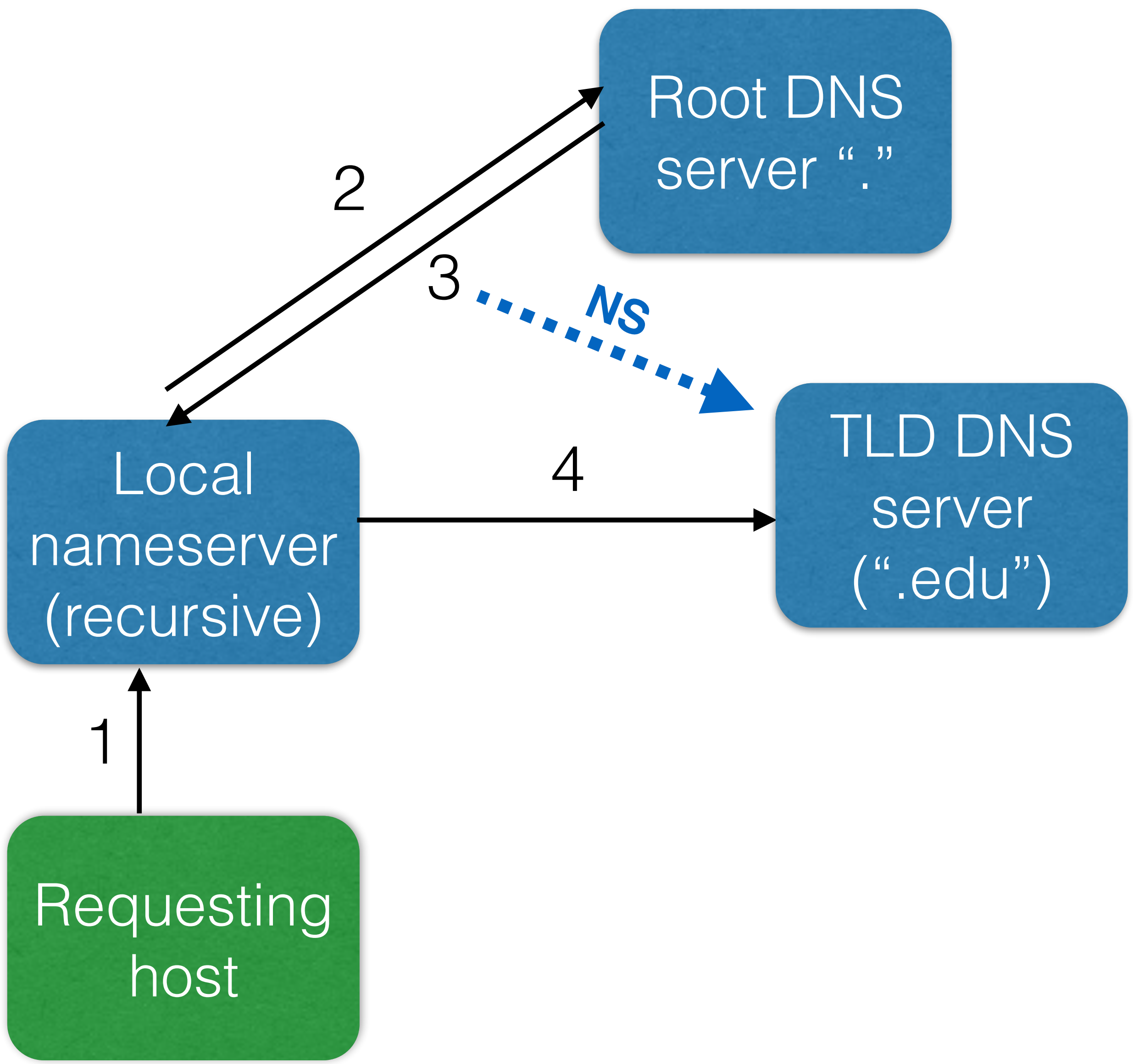
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

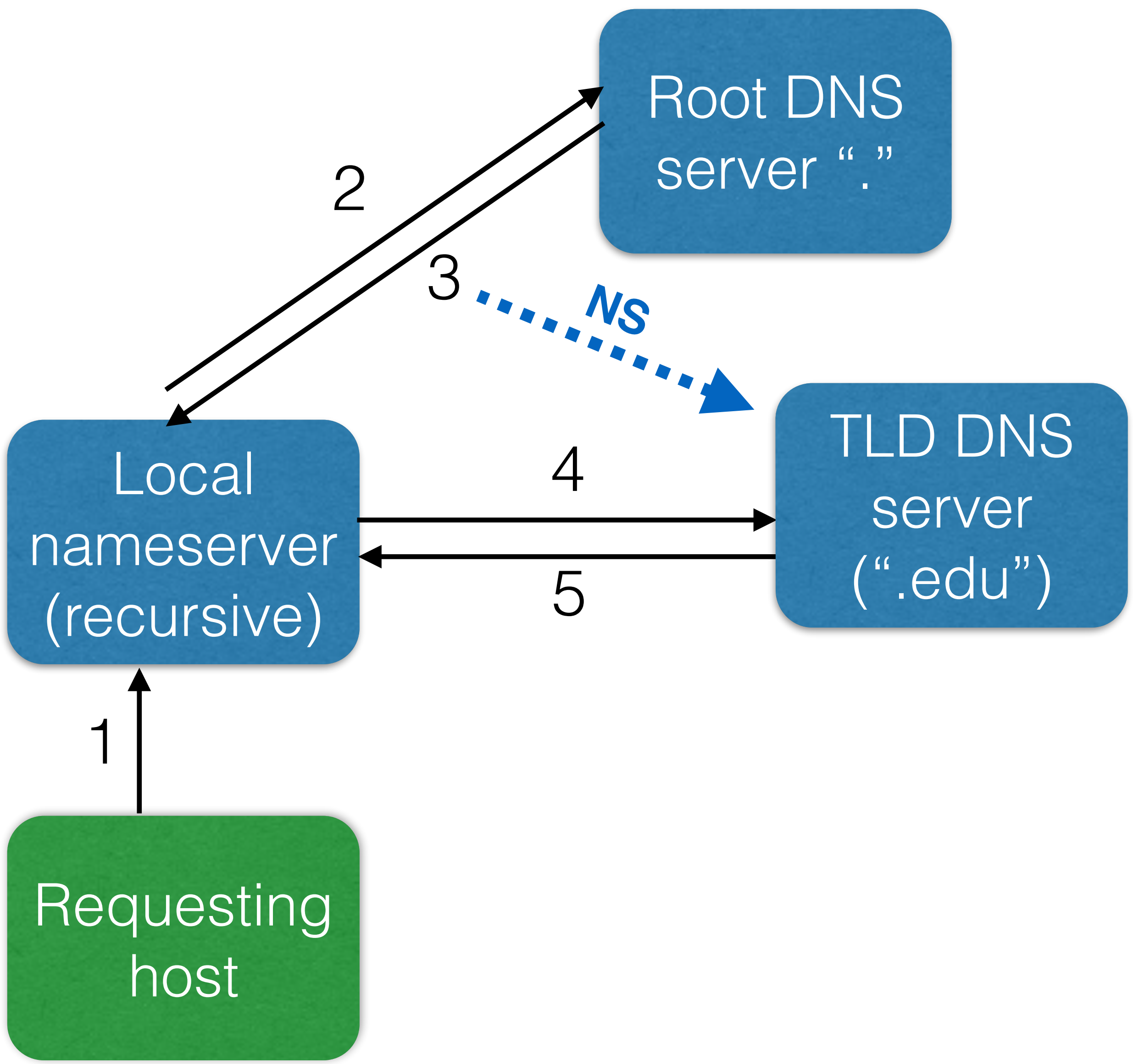
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

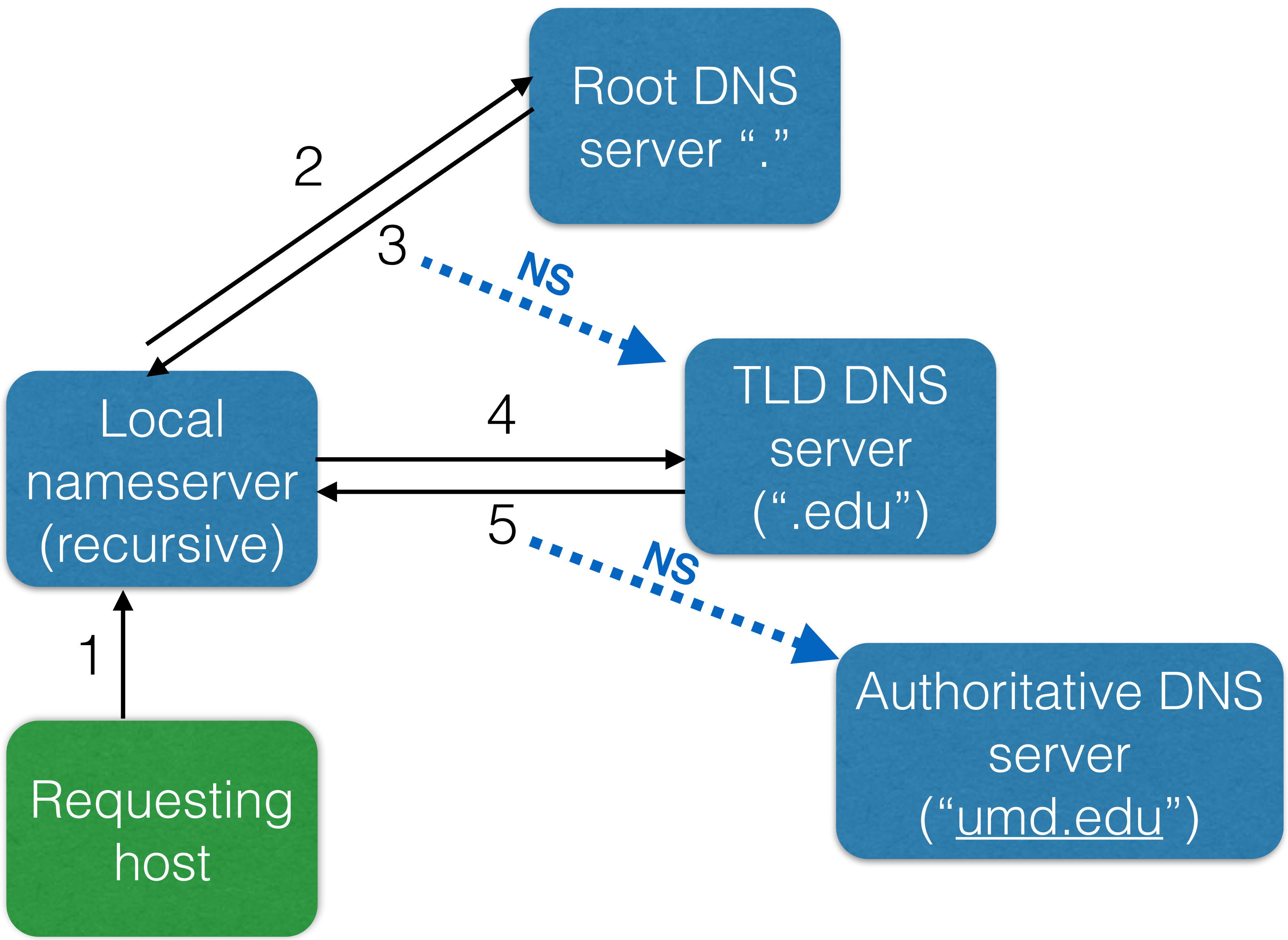
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

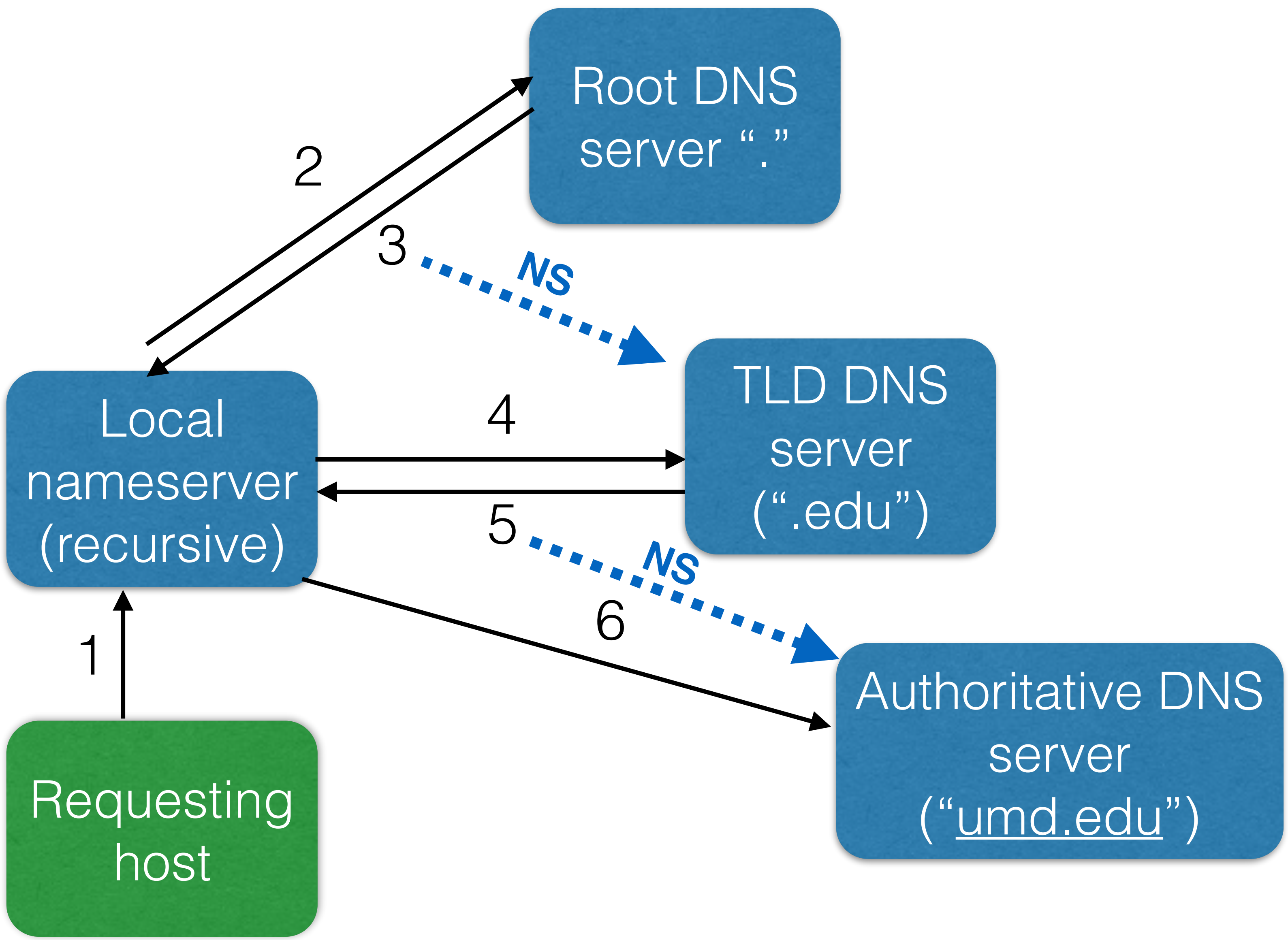
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

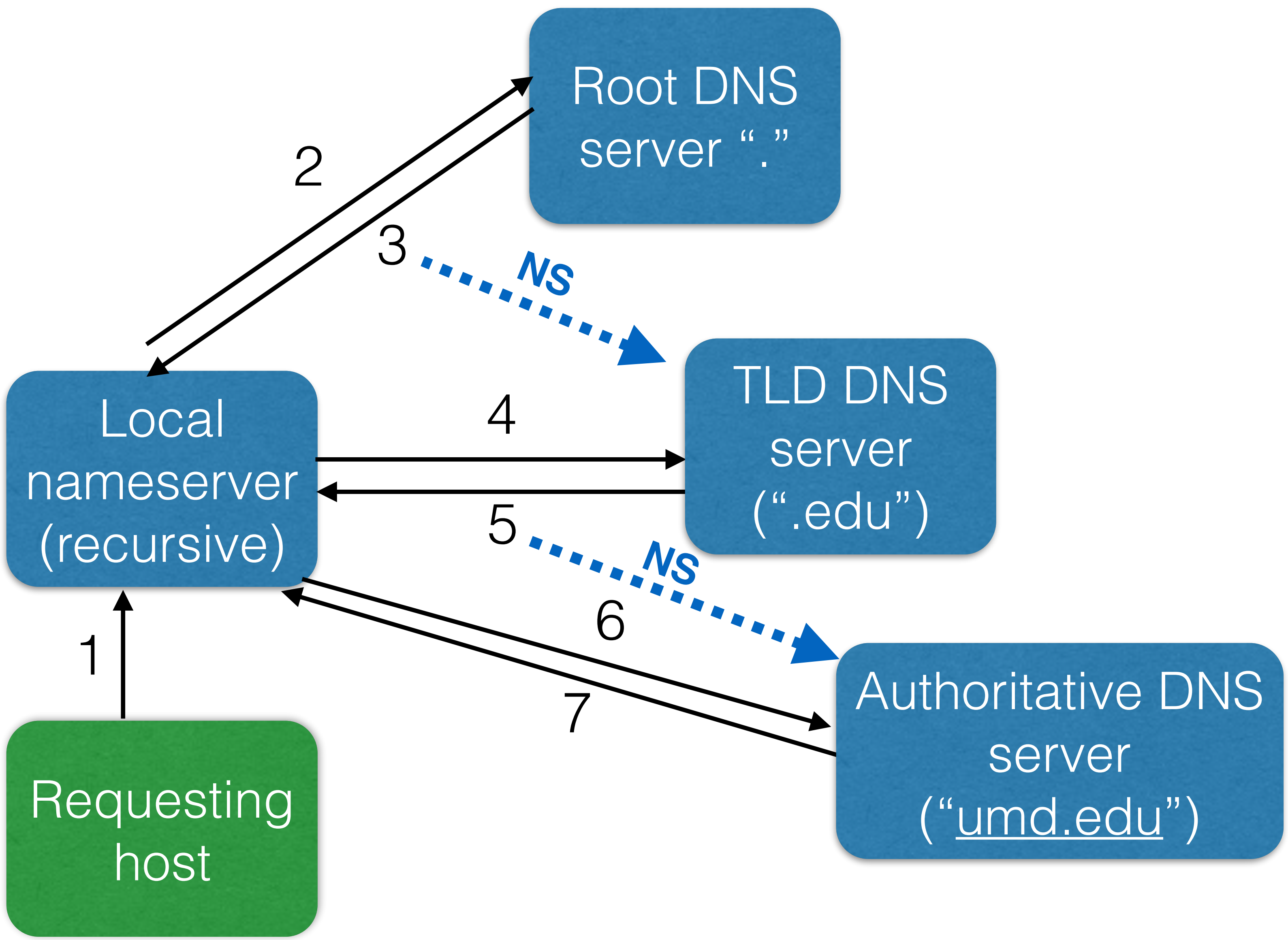
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

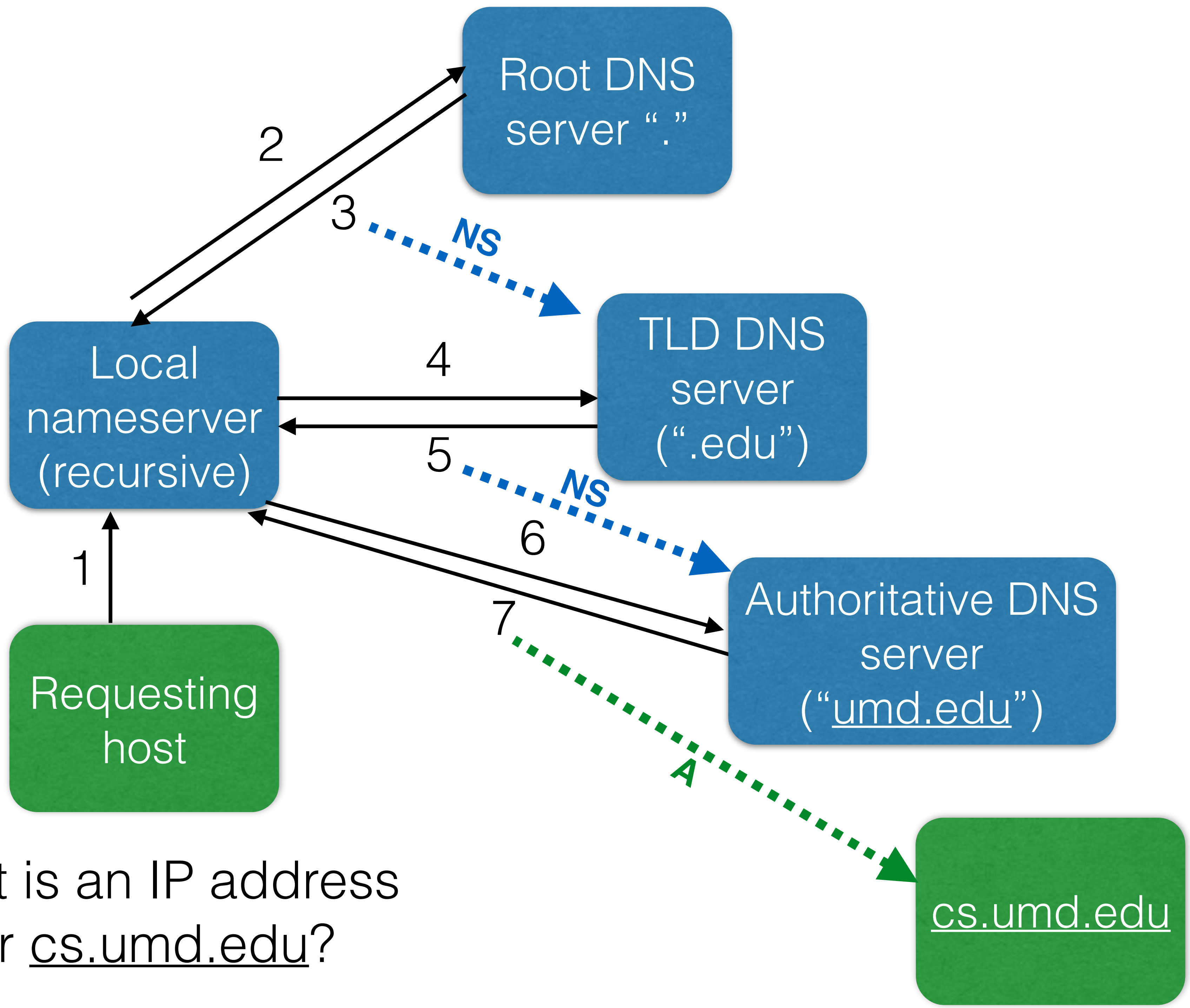
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

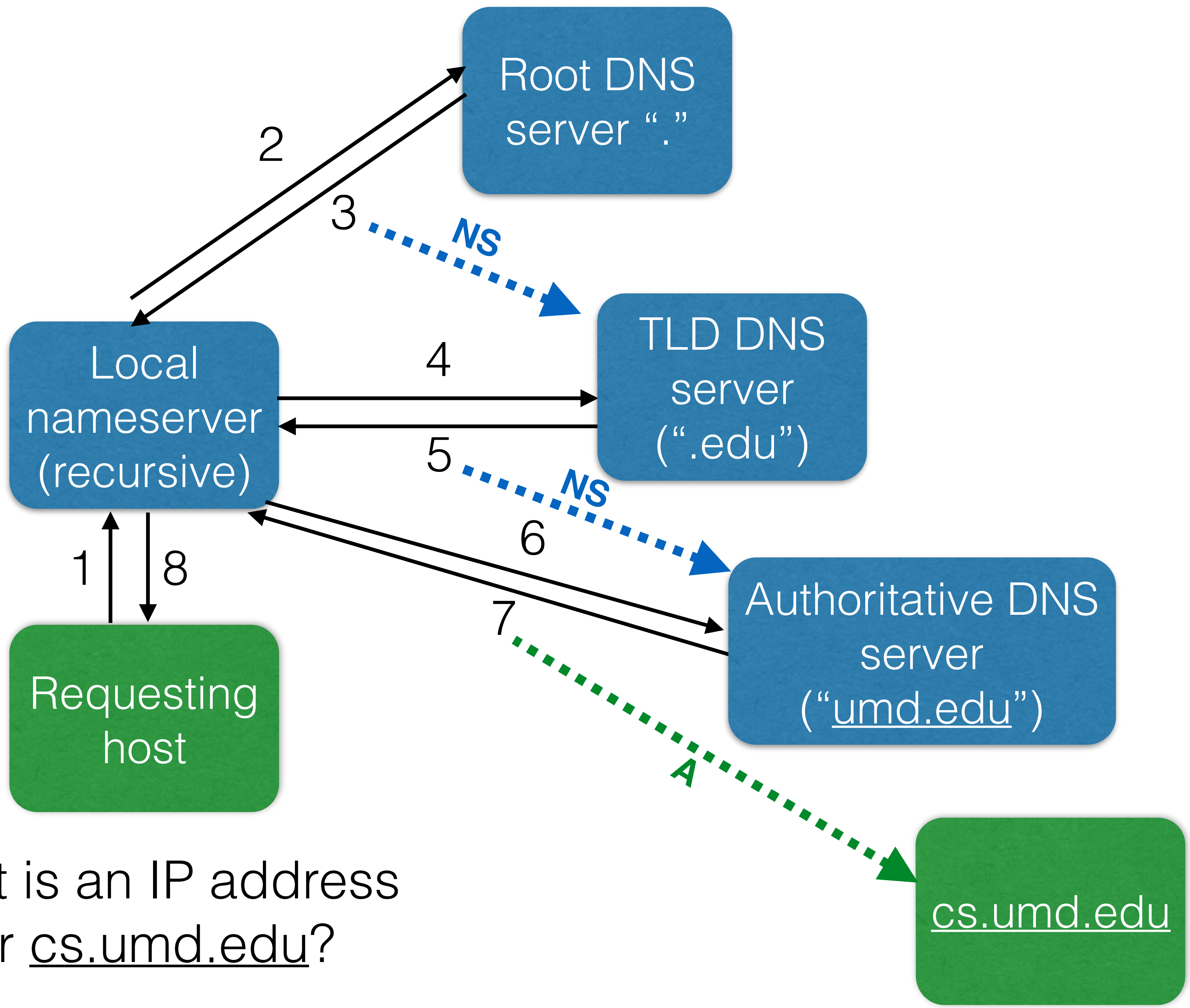
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

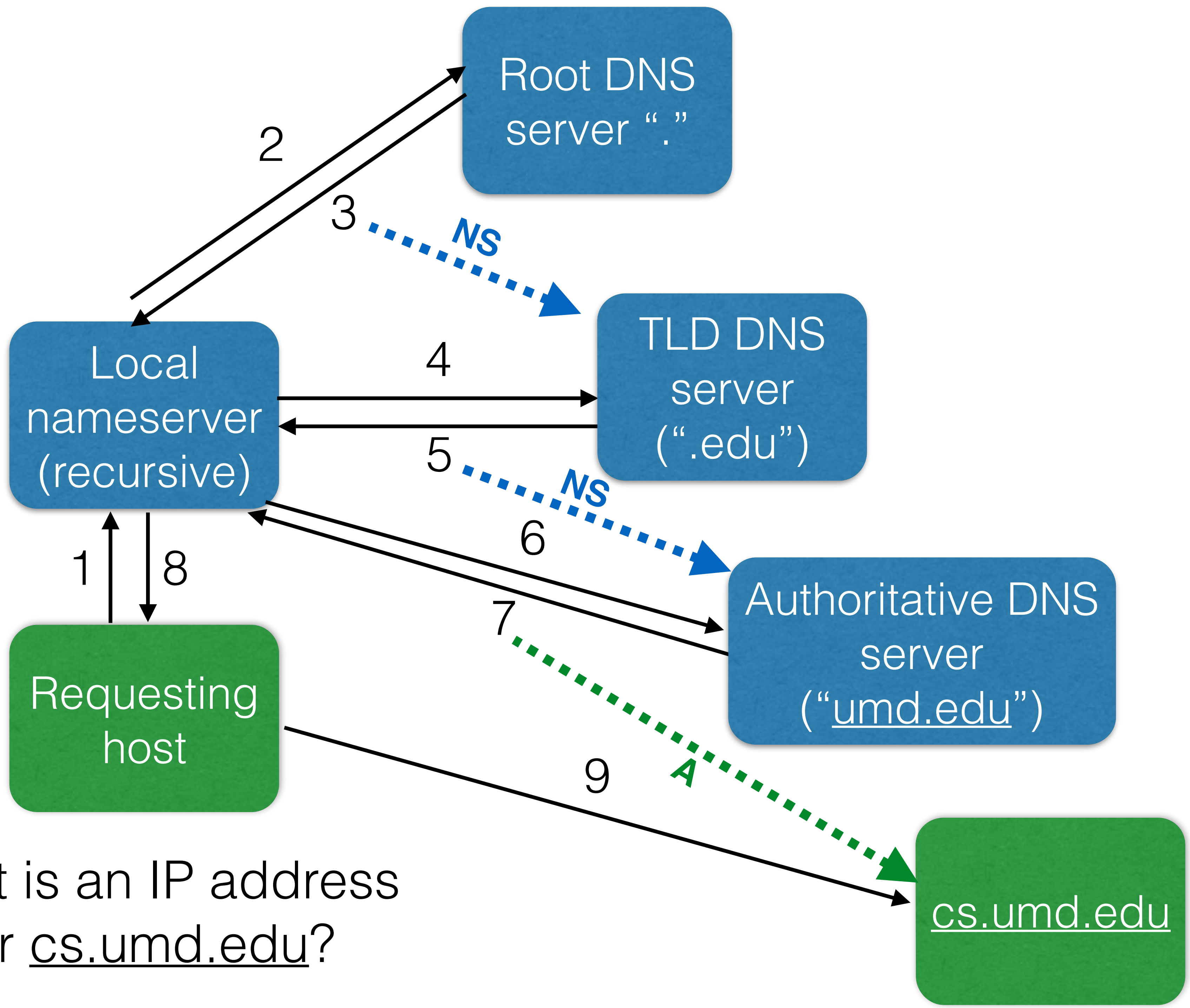
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

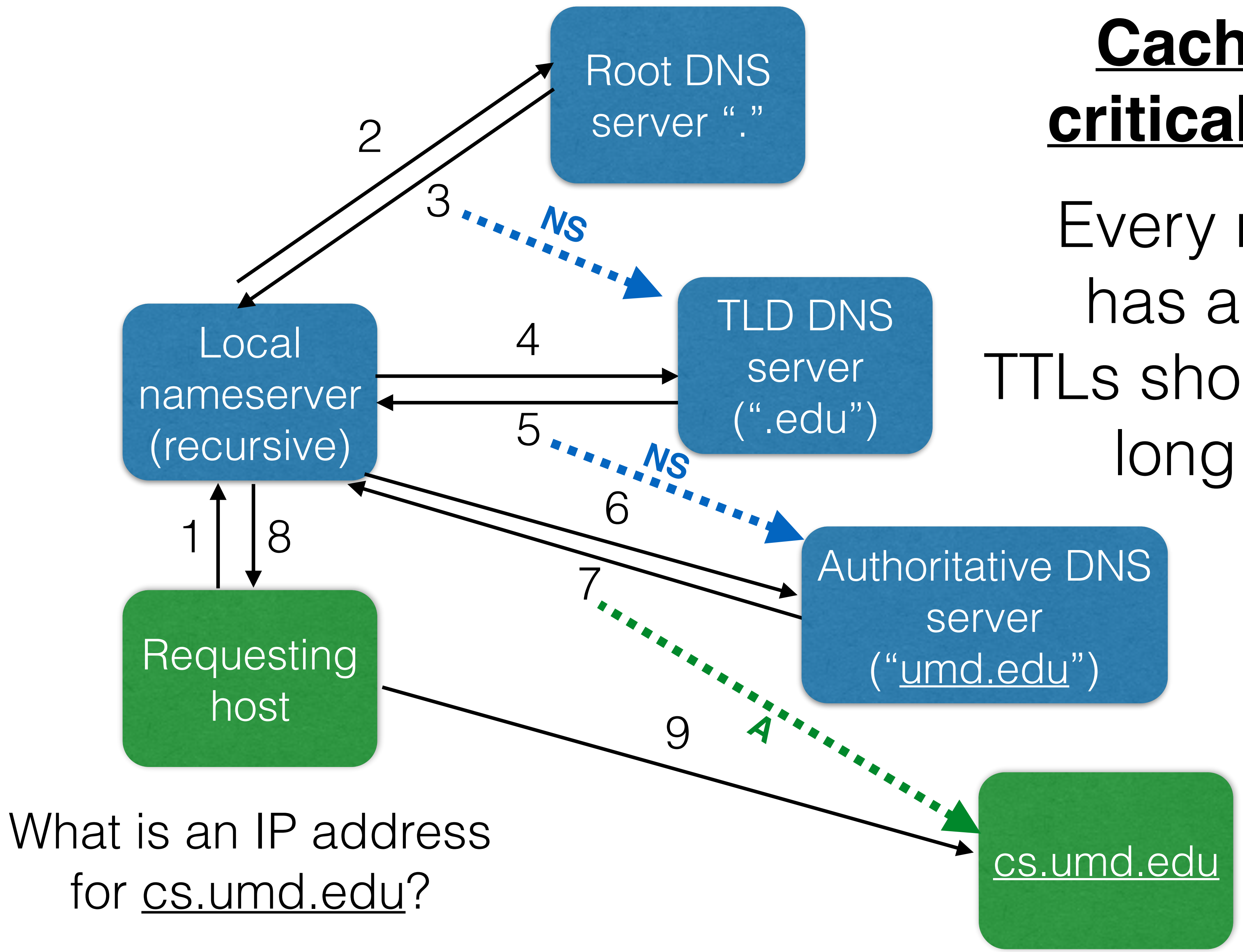
Domain Name Service at a very high level



What is an IP address for cs.umd.edu?

DNS

Domain Name Service at a very high level



Caching responses is critical to DNS's success

Every response (3,5,7,8) has a time-to-live (TTL). TTLs should be reasonably long (days), but some are minutes.

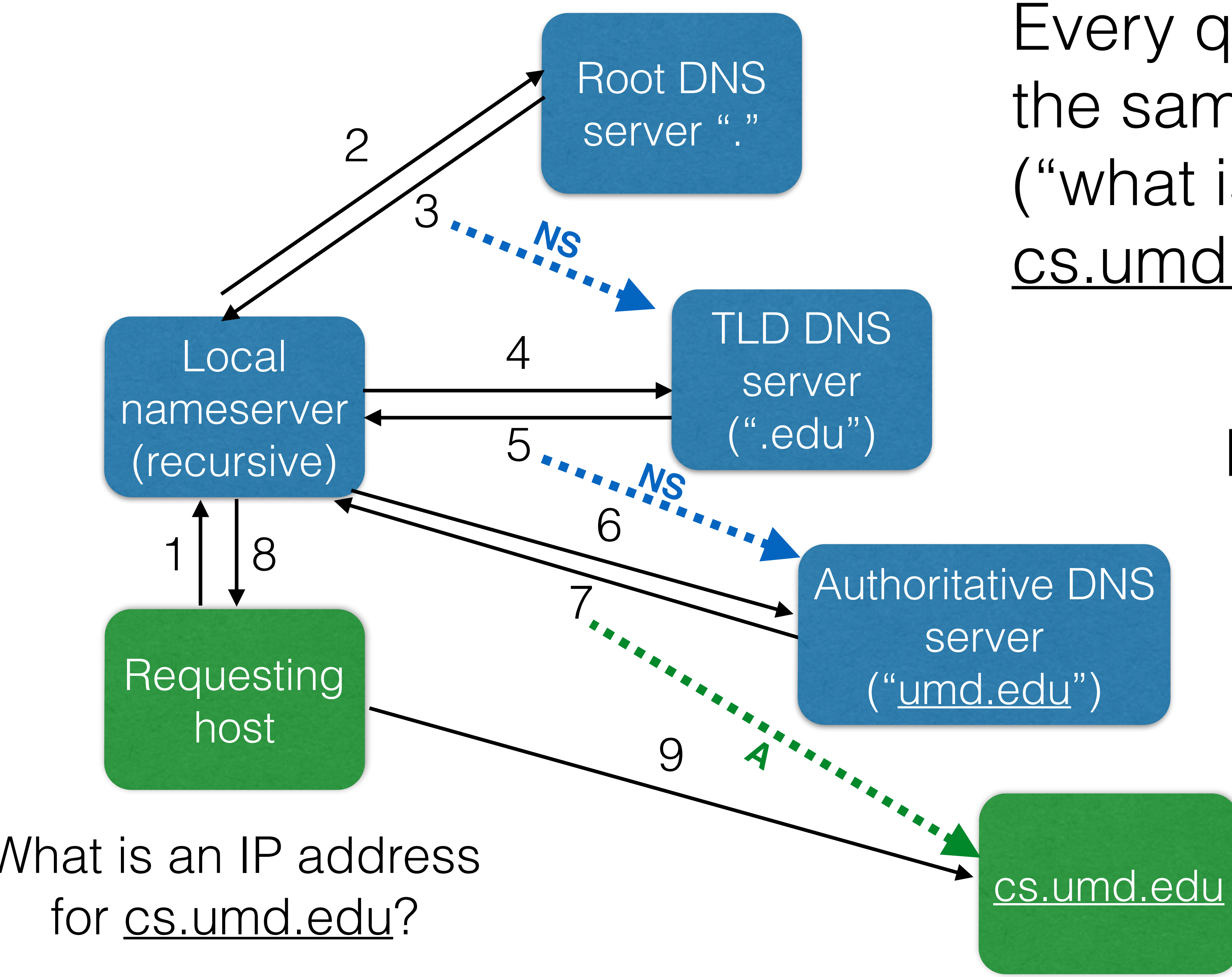
How do they know these IP addresses?

- Local DNS server: host learned this via DHCP
- A parent knows its children: part of the registration process
- Root nameserver: *hardcoded* into the local DNS server (and every DNS server)
  - 13 root servers (logically): A-root, B-root, …, M-root
  - These IP addresses change *very* infrequently
  - **UMD runs D-root.**
    - IP address changed beginning of 2013!!
    - For the most part, the change-over went alright, but Lots of weird things happened — ask me some time.

Caching

- Central to DNS's success
- Also central to attacks
- "Cache poisoning": filling a victim's cache with false information

Queries



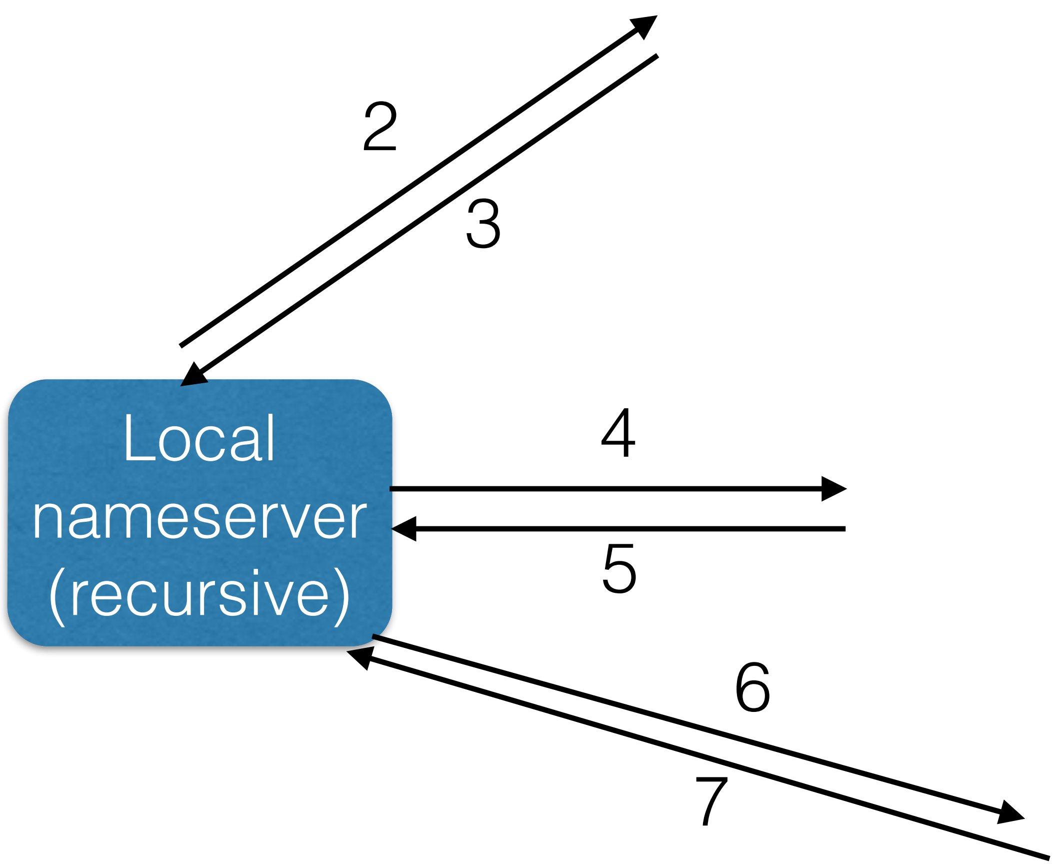
Every query (2,4,6) has the same request in it ("what is the IP address for cs.umd.edu?")

But **different**:
- dst IP (port = 53)
- query ID

What's in a response?

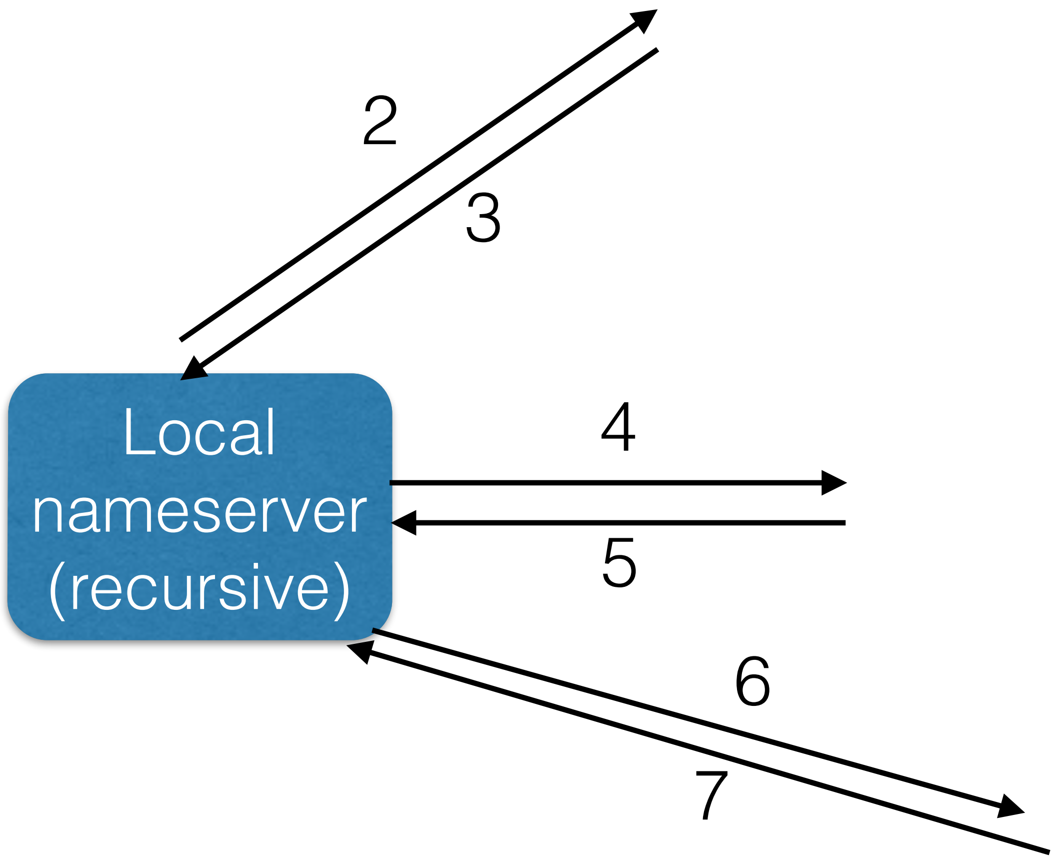
- Many things, but for the attacks we're concerned with…
- A record: gives "the authoritative response for the IP address of this hostname"
- NS record: describes "this is the name of the nameserver who should know more about how to answer this query than I do"
  - Often also contains "glue" records (IP addresses of those name servers to avoid chicken and egg problems)
  - Resolver will generally cache all of this information

Query IDs



- The local resolver has a lot of incoming/outgoing queries at any point in time.
- To determine which response maps to which queries, it uses a *query ID*
- Query ID: 16-bit field in the DNS header
  - Requester sets it to whatever it wants
  - Responder must provide the same value in its response

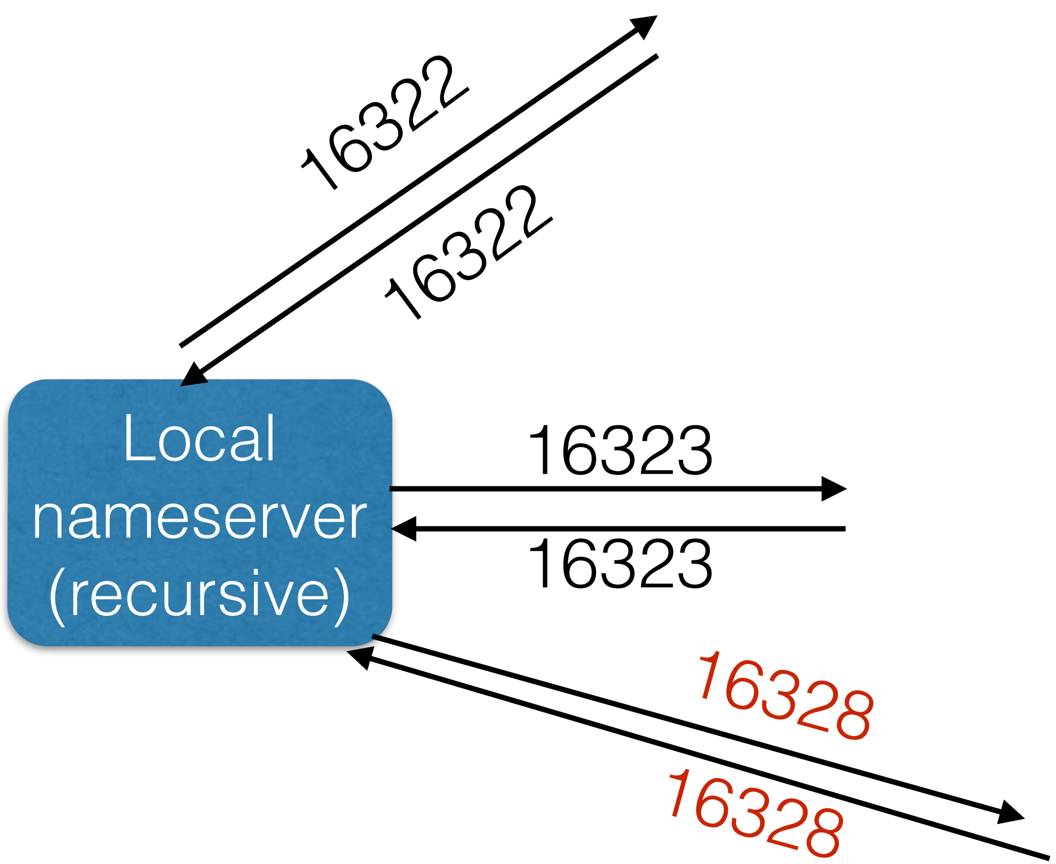
Query IDs



- The local resolver has a lot of incoming/outgoing queries at any point in time.
- To determine which response maps to which queries, it uses a *query ID*
- Query ID: 16-bit field in the DNS header
    - Requester sets it to whatever it wants
    - Responder must provide the same value in its response

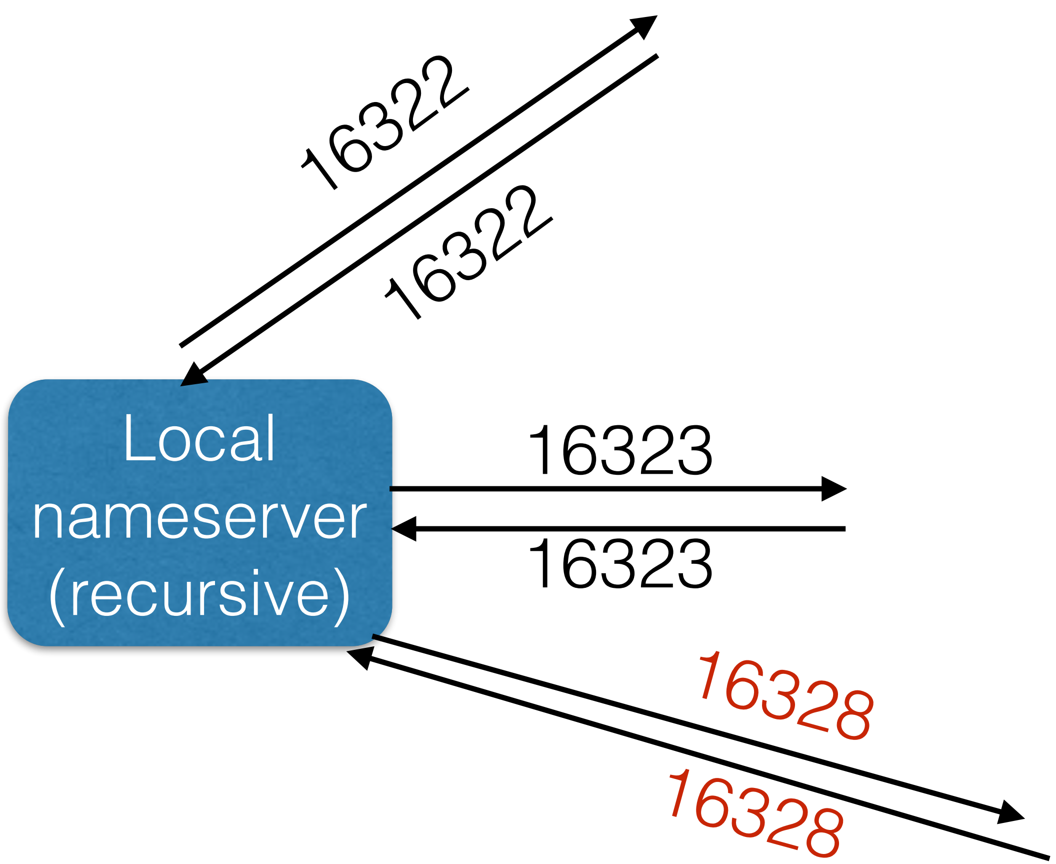
How would you implement query IDs at a resolver?

Query IDs used to increment



- Global query ID value
- Map outstanding query ID to local state of who to respond to (the client)
- Basically:
  new Packet(queryID++)

Query IDs used to increment

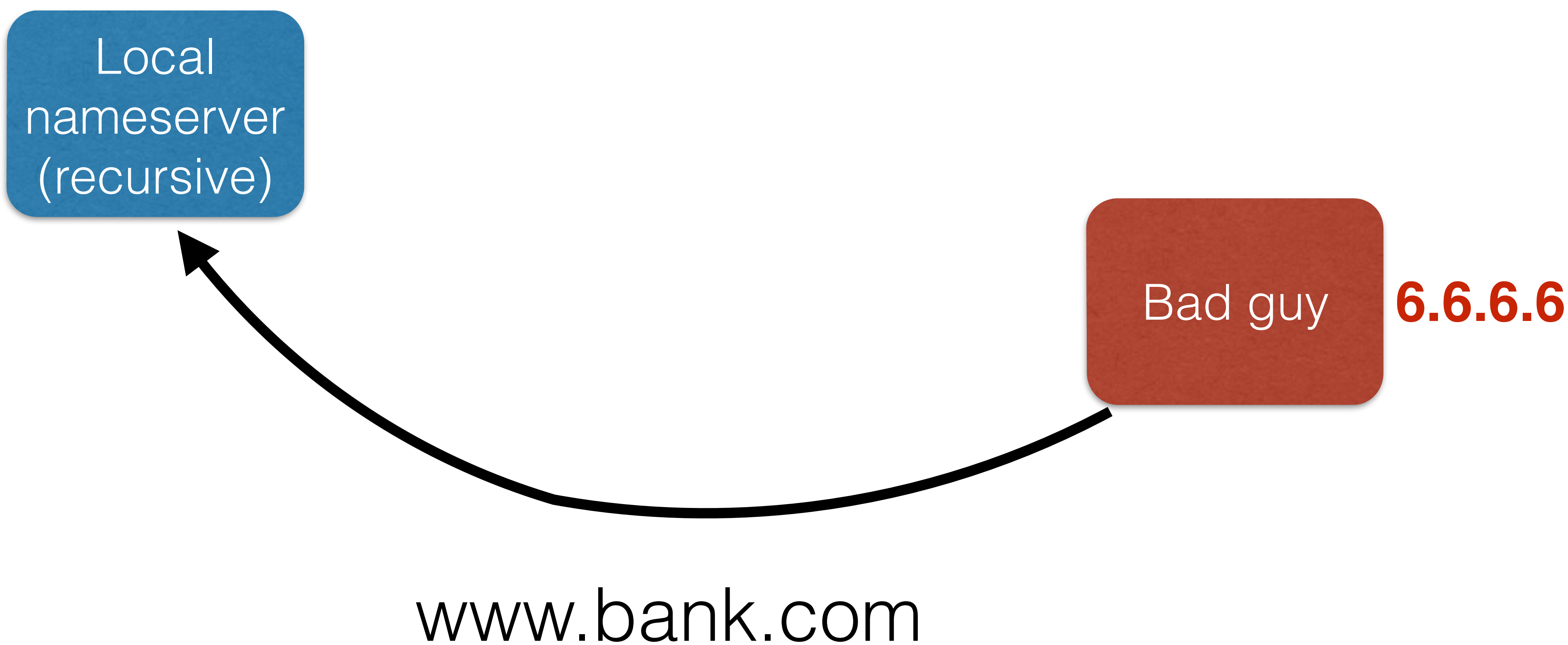


- Global query ID value
- Map outstanding query ID to local state of who to respond to (the client)
- Basically:
  new Packet(queryID++)

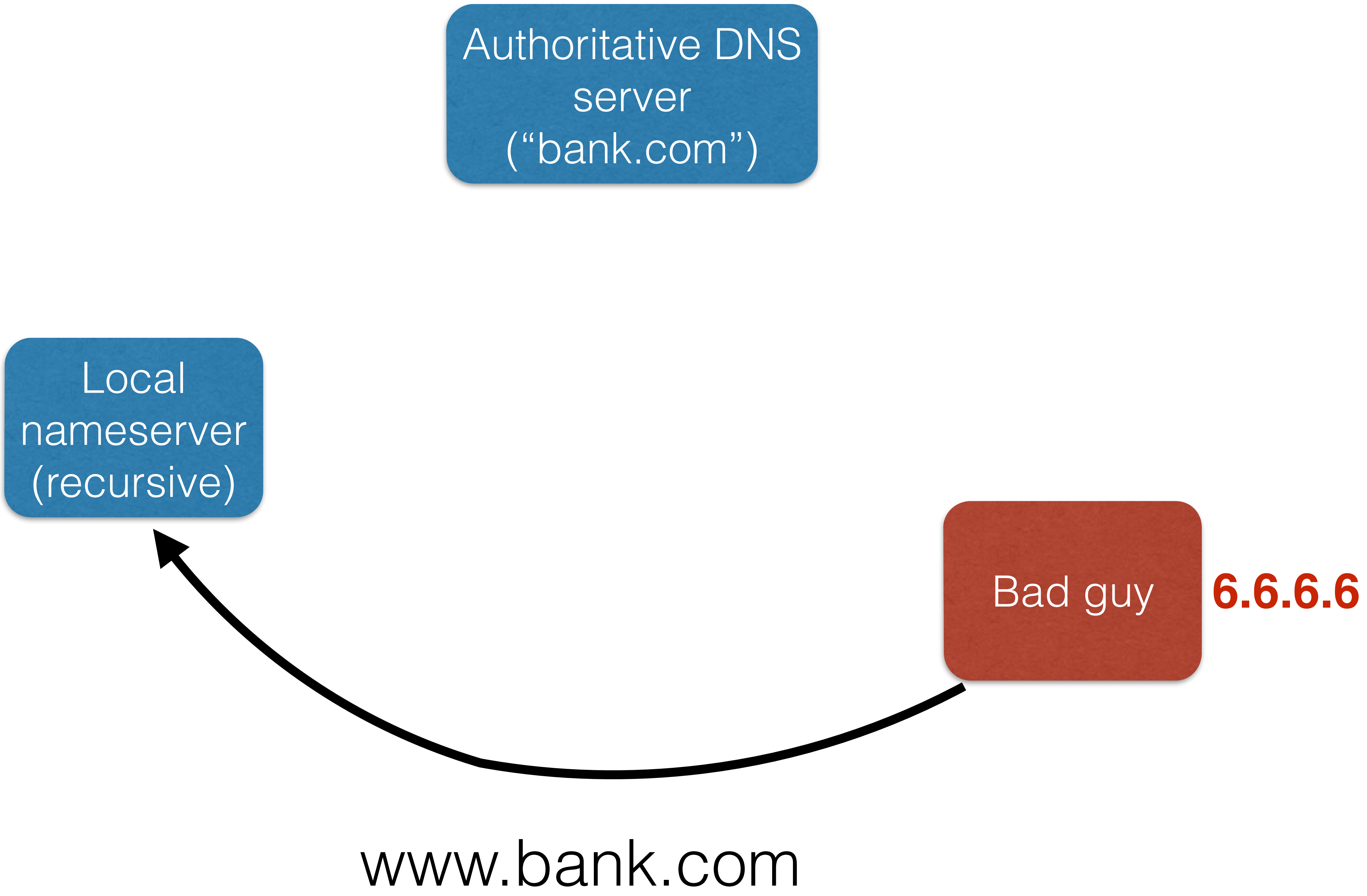
How would you attack this?

Cache poisoning

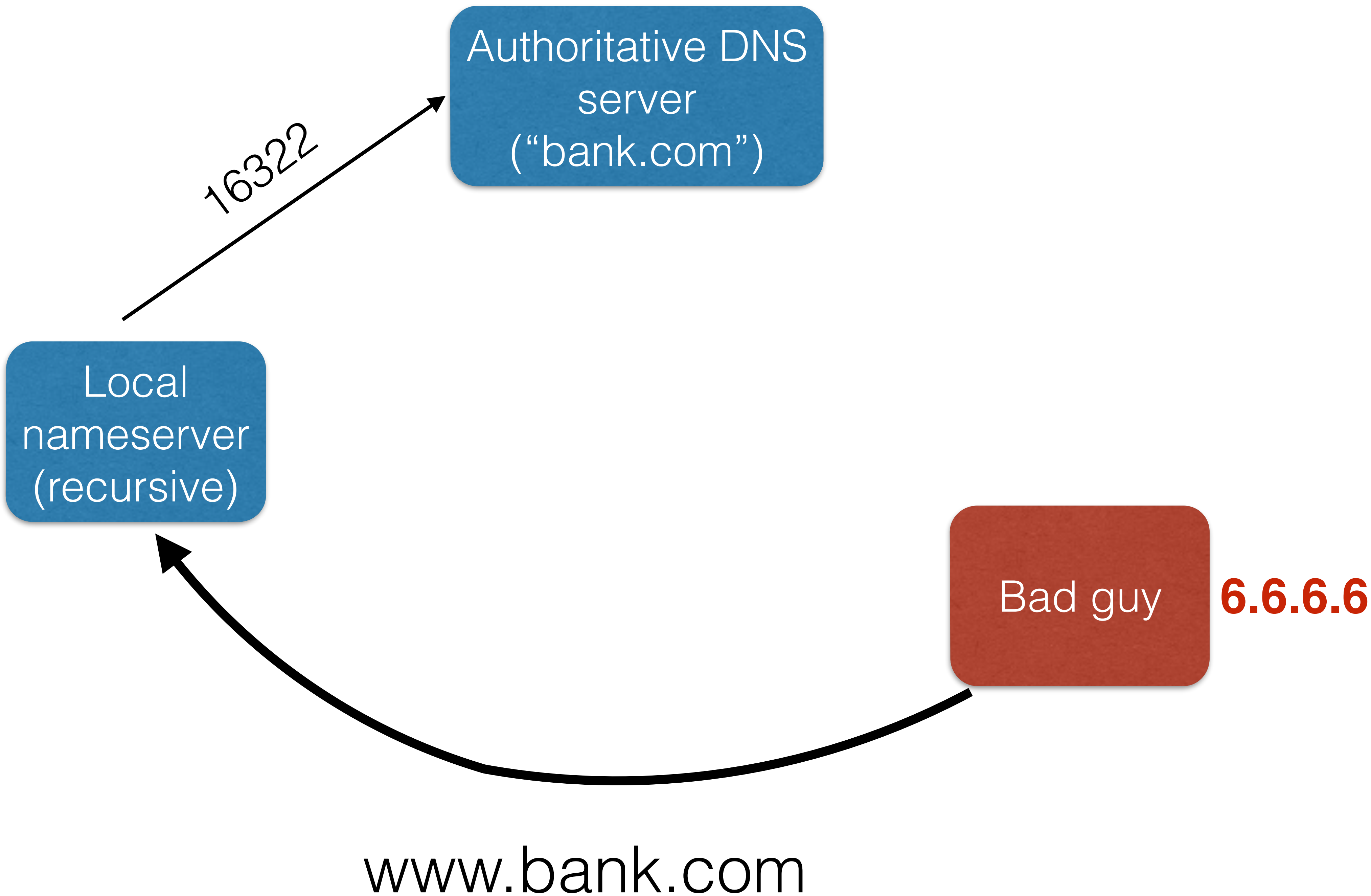
Cache poisoning



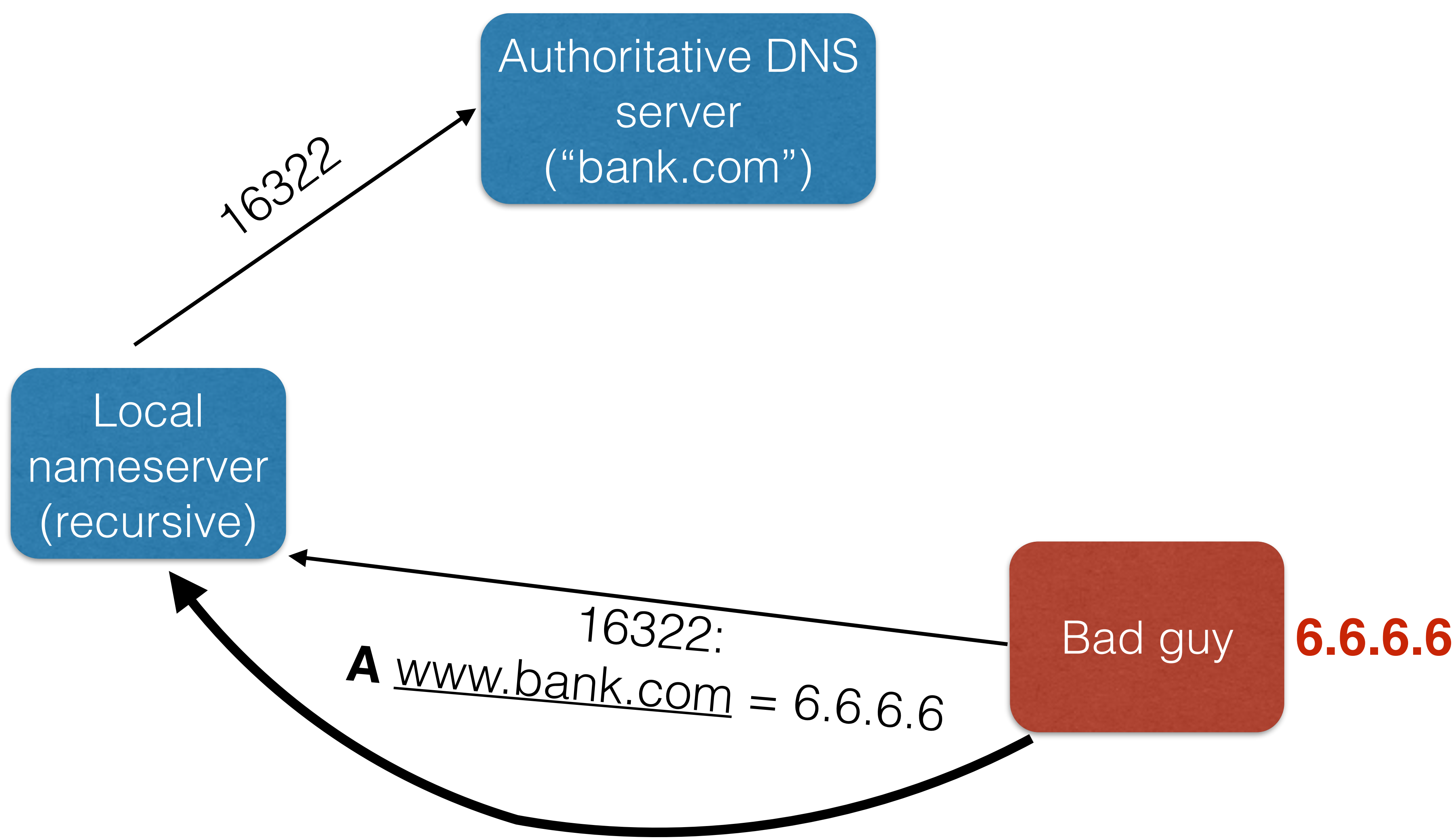
Cache poisoning



Cache poisoning

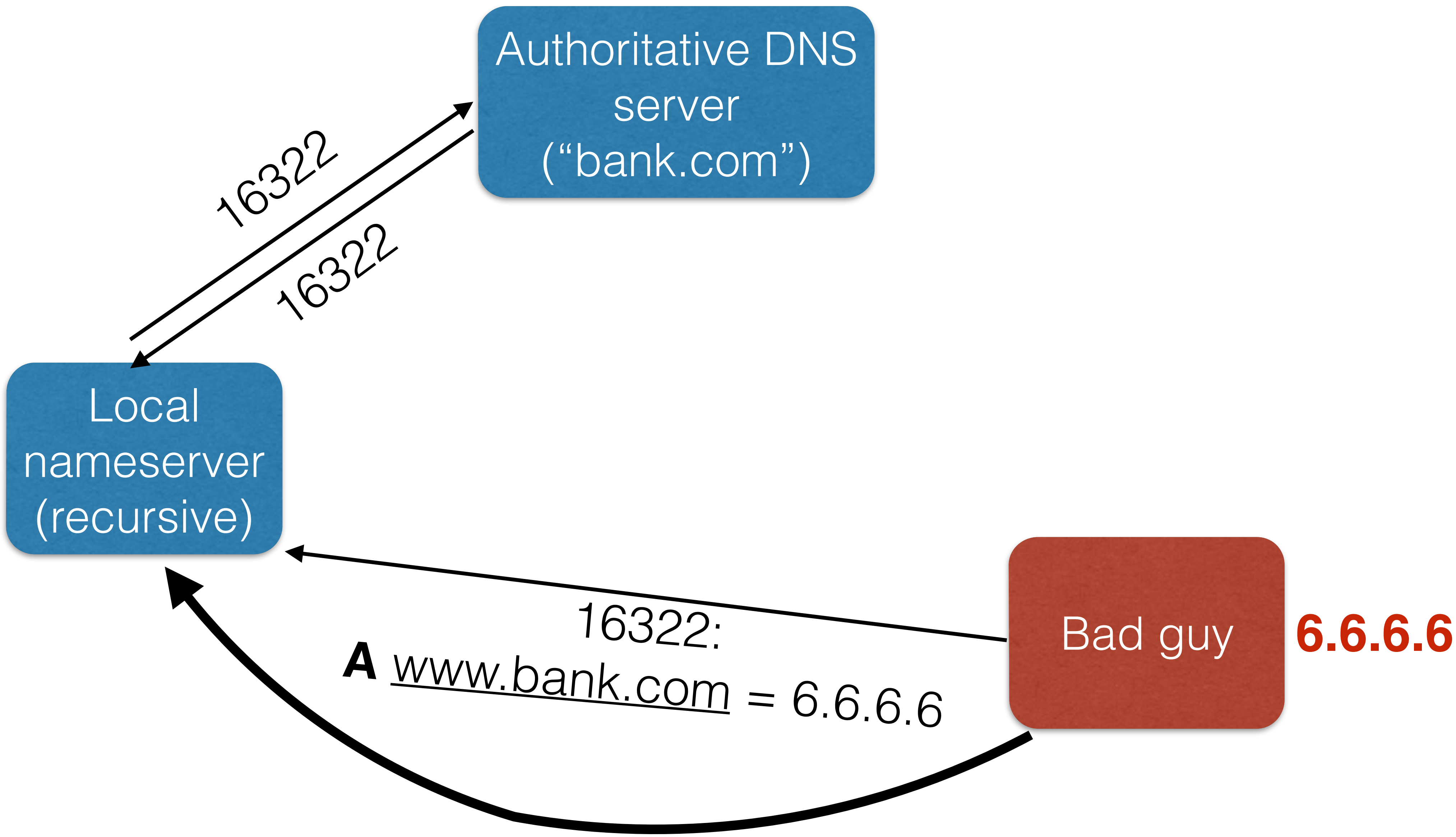


Cache poisoning



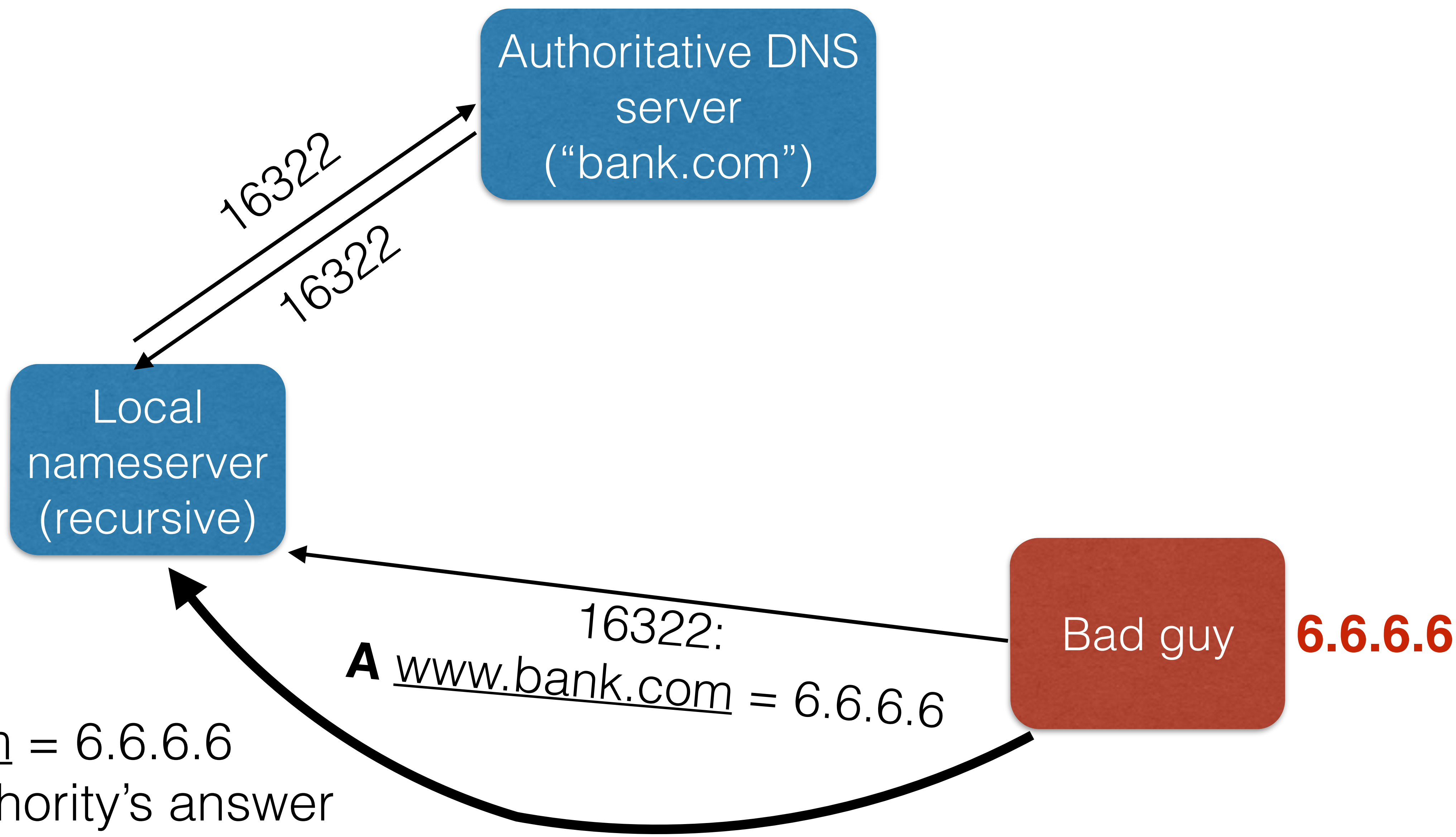
www.bank.com

Cache poisoning



www.bank.com

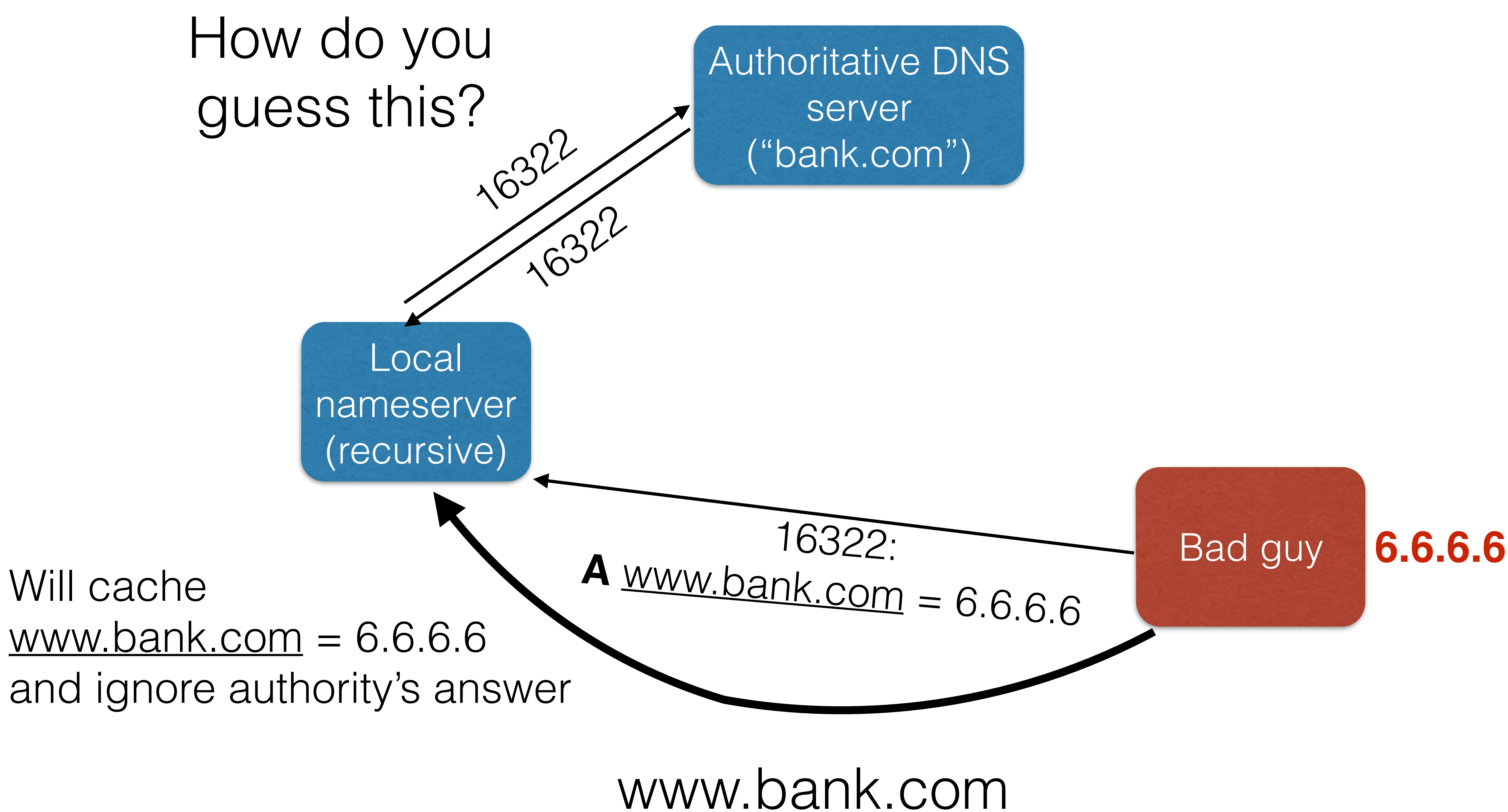
Cache poisoning



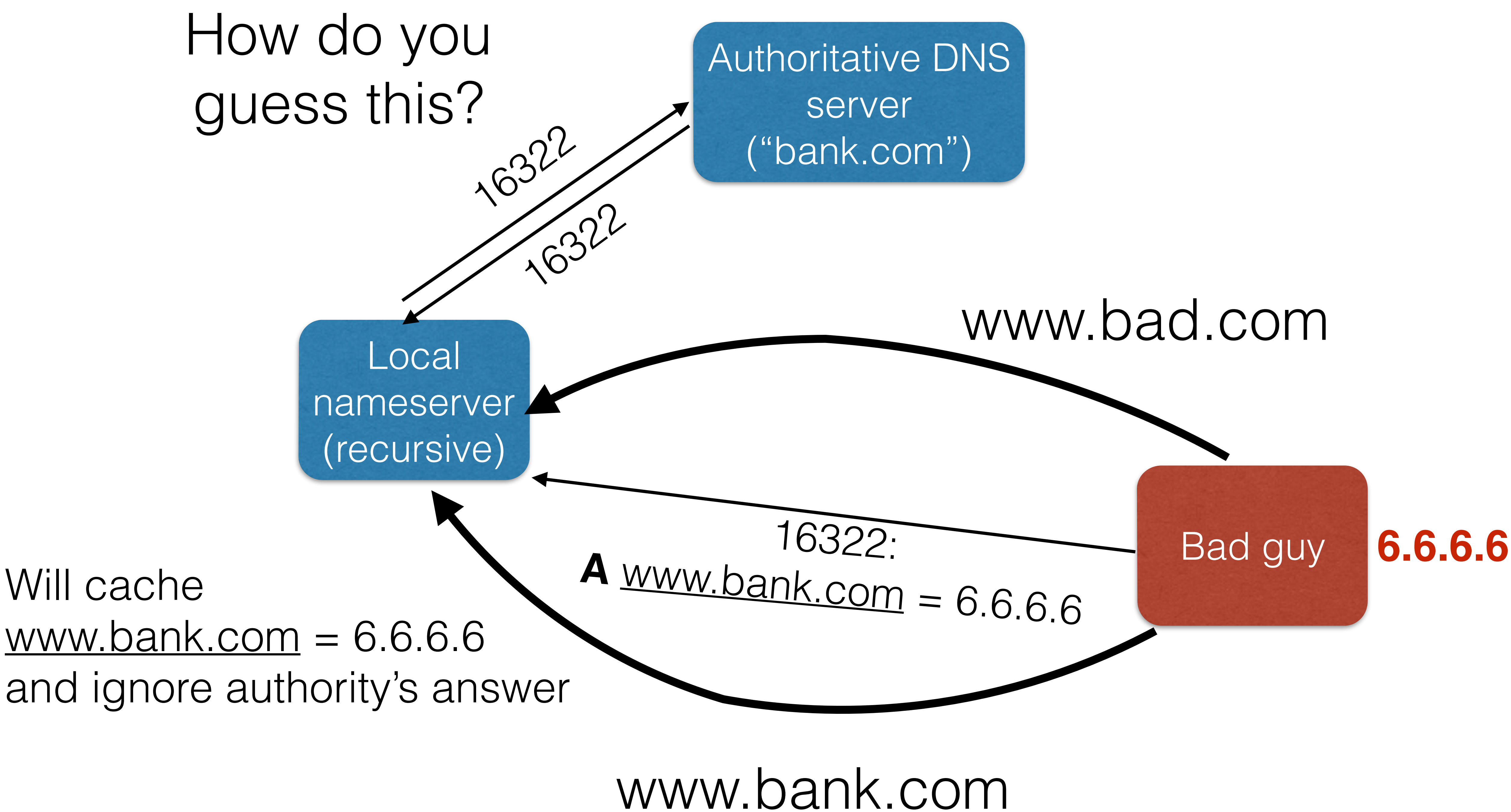
Will cache
www.bank.com = 6.6.6.6
and ignore authority's answer

www.bank.com

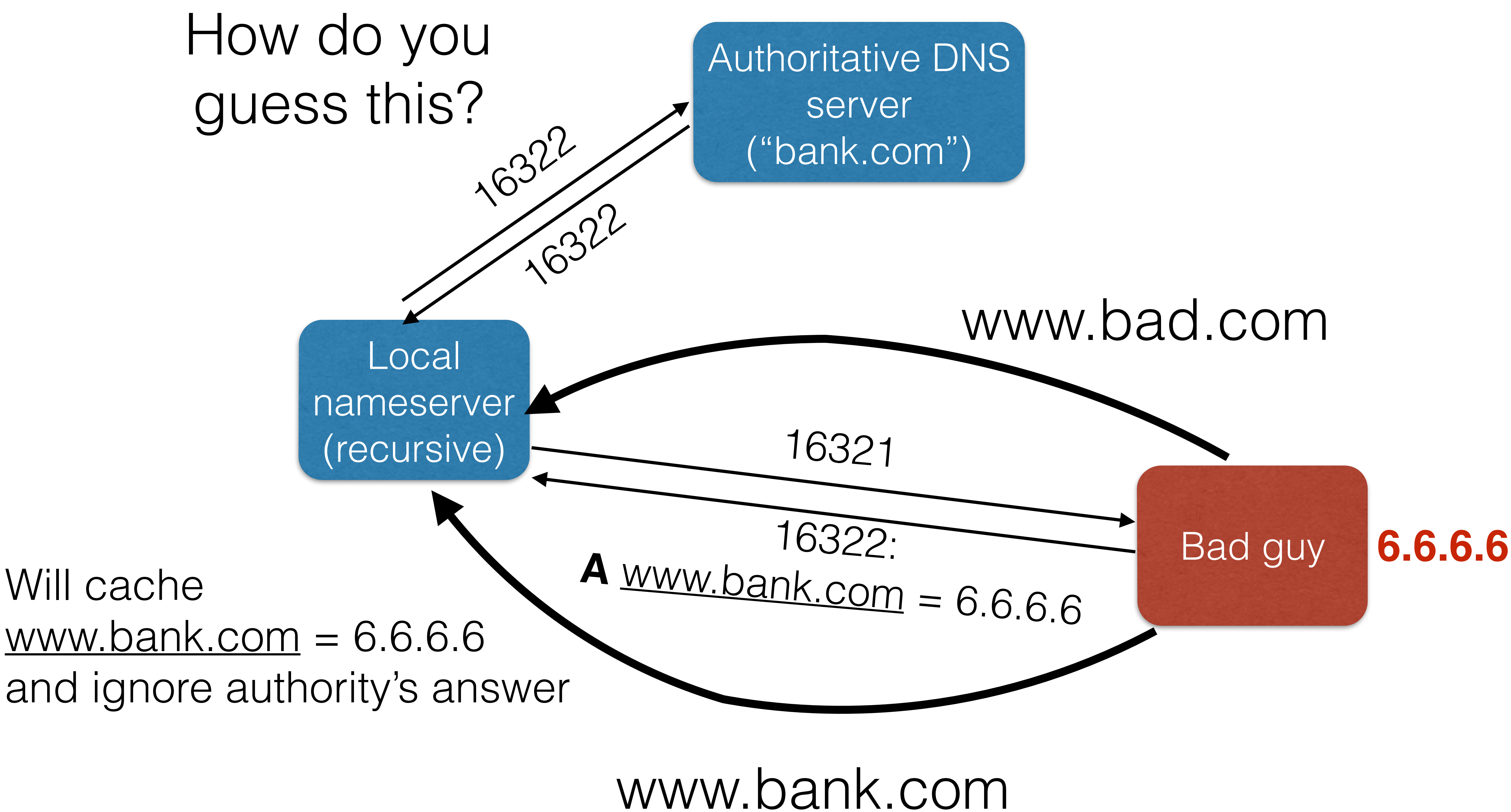
Cache poisoning



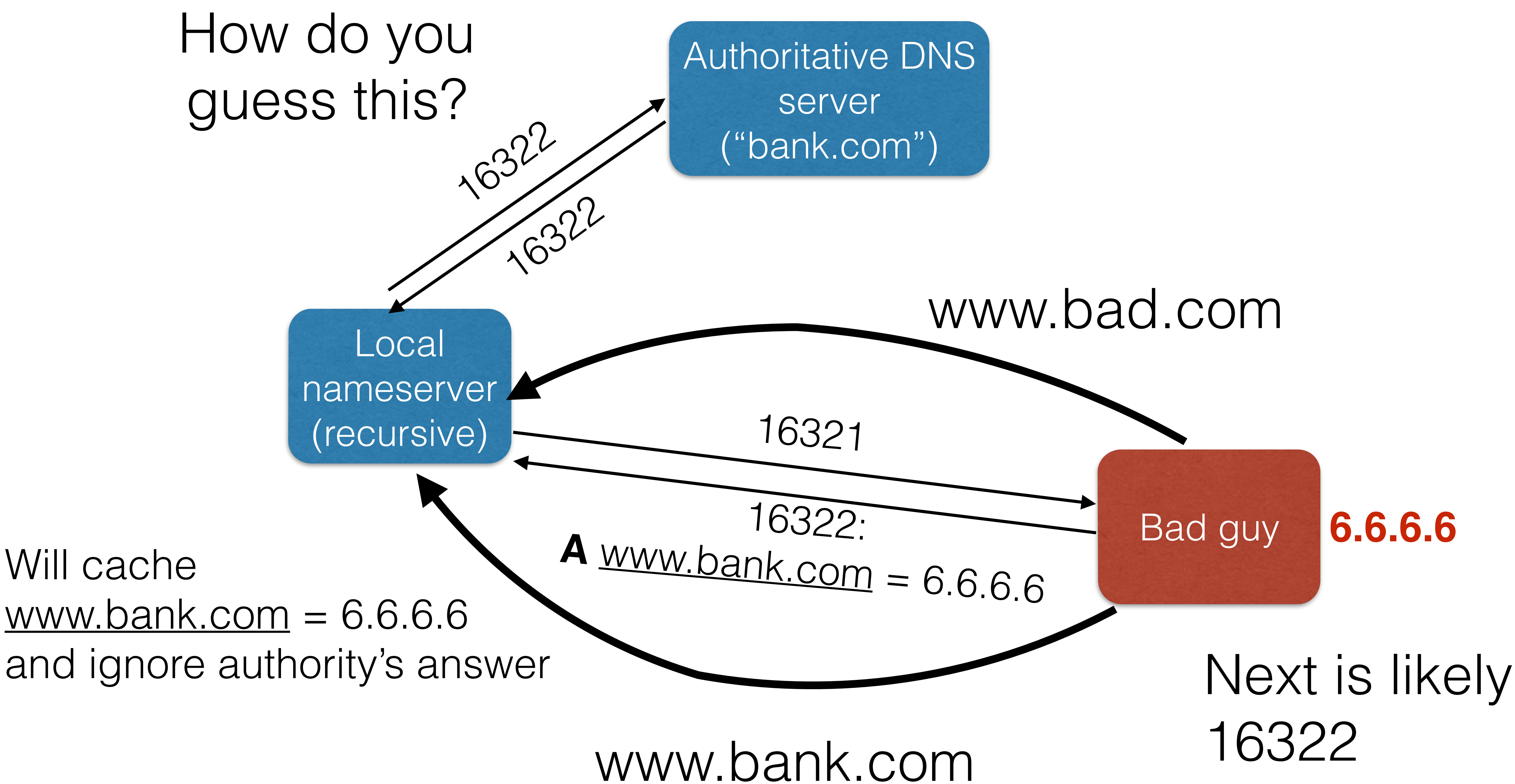
Cache poisoning



Cache poisoning



Cache poisoning

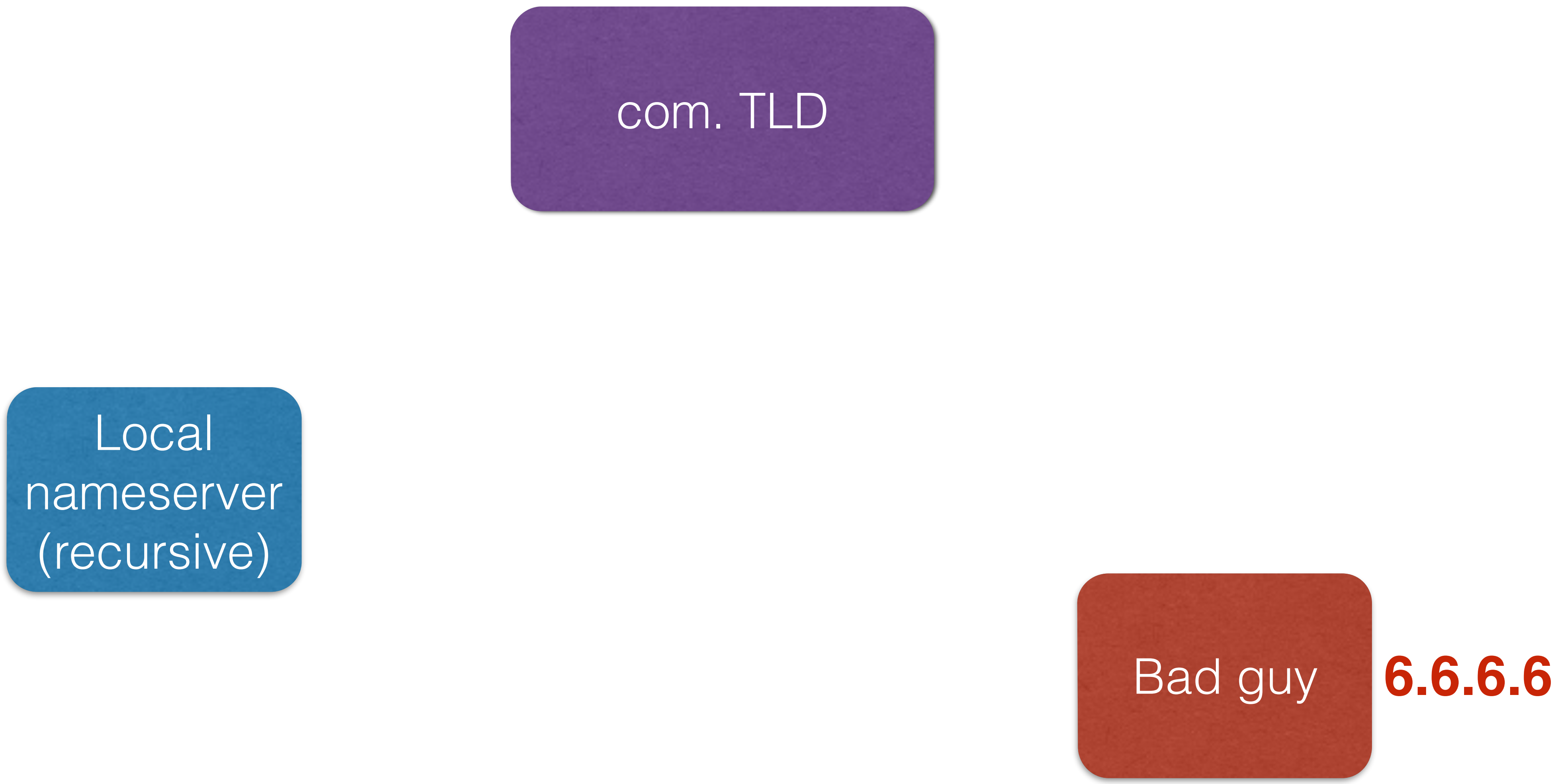


Details of getting the attack to work

- Must guess query ID: ask for it, and go from there
    - Partial fix: randomize query IDs
    - Problem: small space
    - Attack: issue a Lot of query IDs
- Must guess source port number
    - Typically constant for a given server (often always 53)
- The answer must not already be in the cache
    - It will avoid issuing a query in the first place

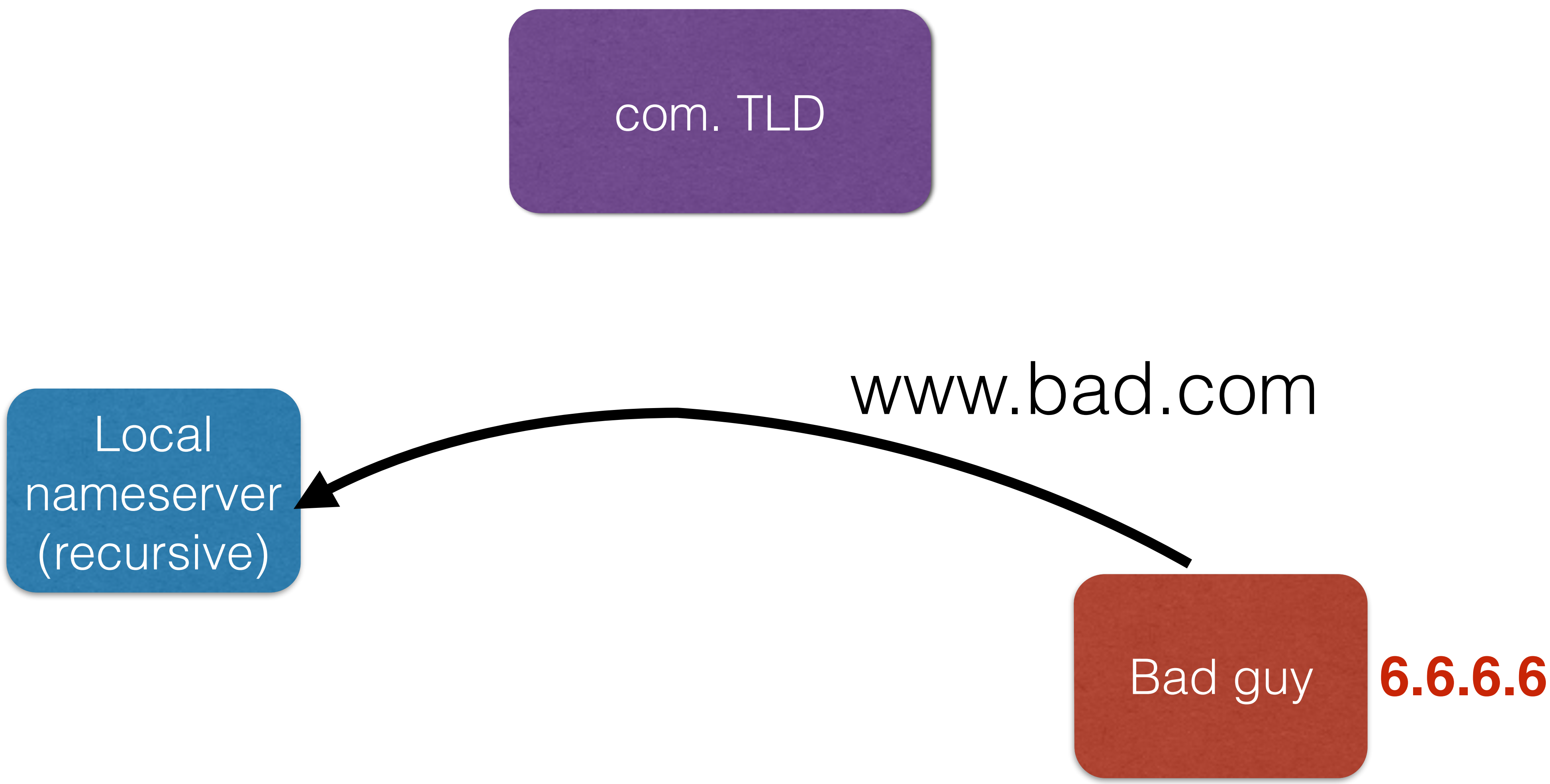
Cache poisoning

Can we do more harm than a single record?



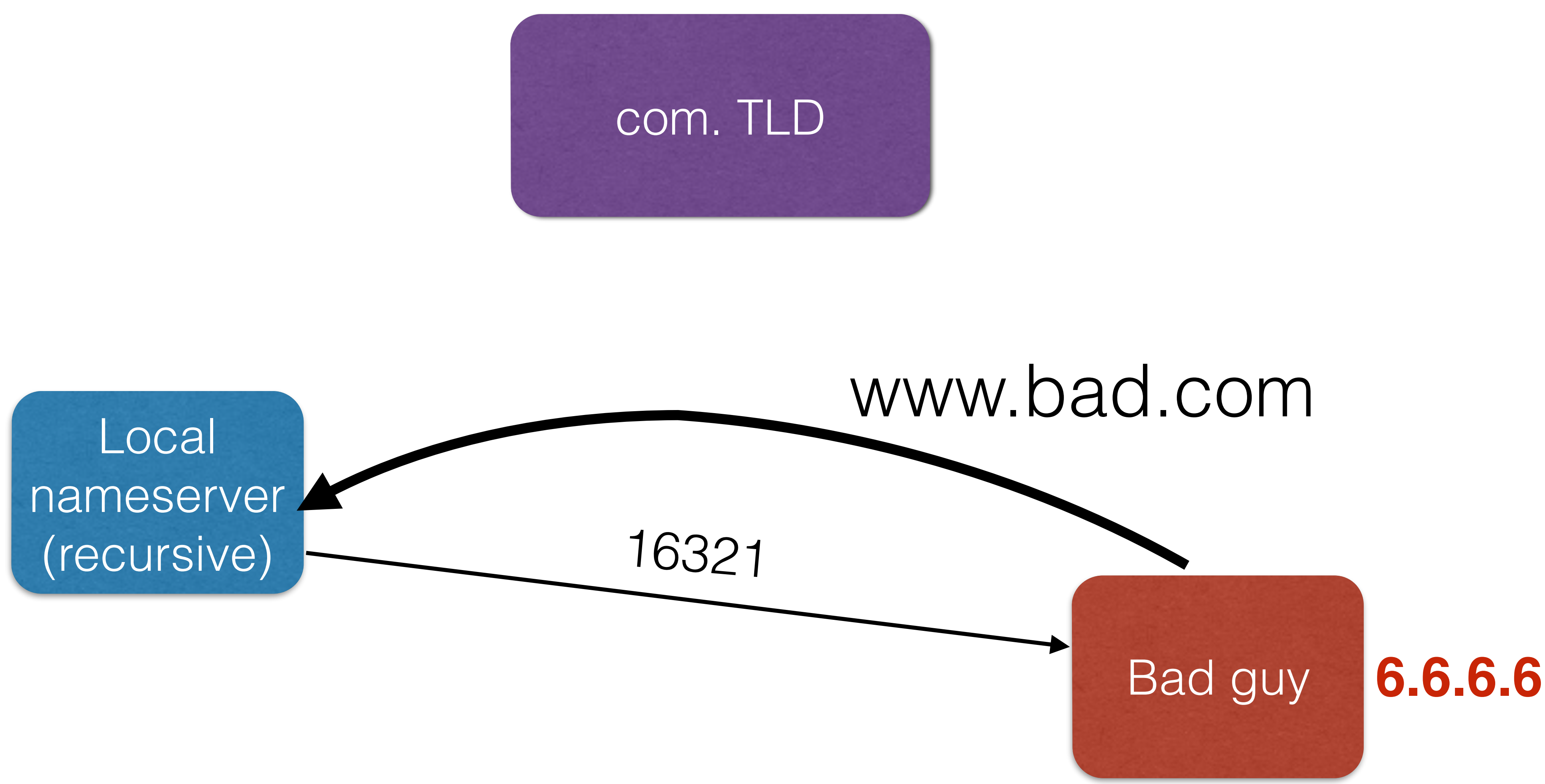
Cache poisoning

Can we do more harm than a single record?



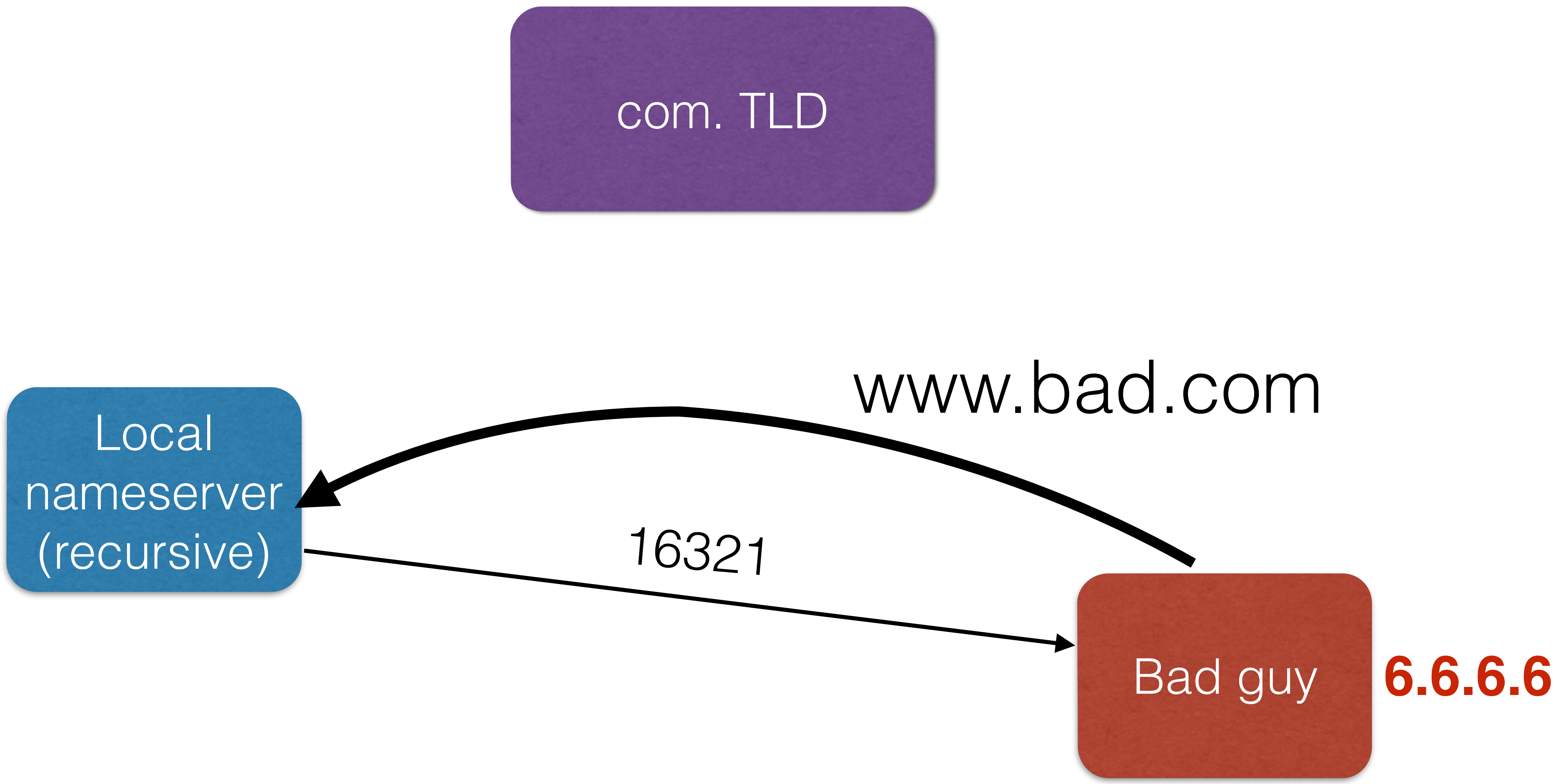
Cache poisoning

Can we do more harm than a single record?



Cache poisoning

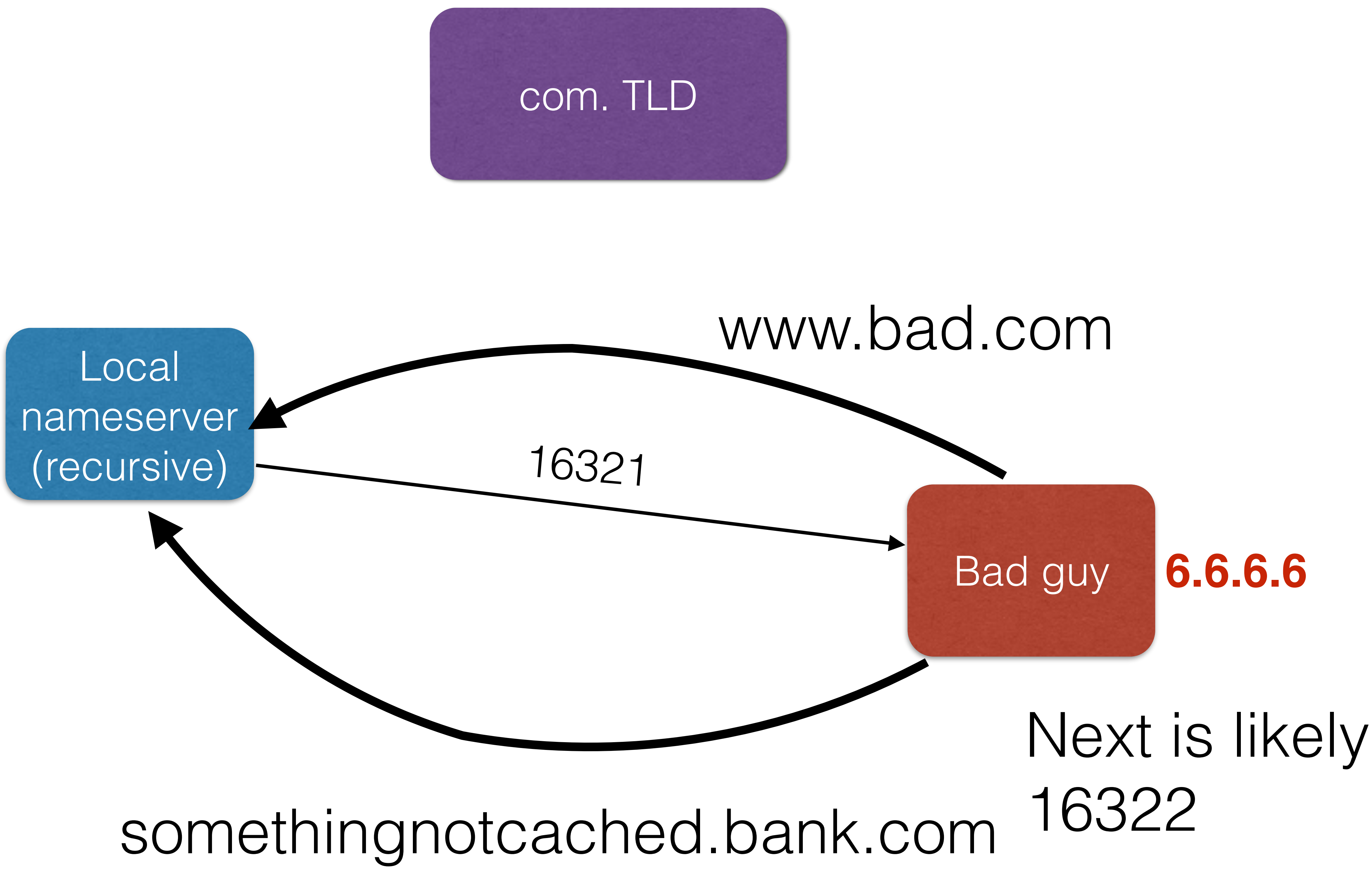
Can we do more harm than a single record?



Next is likely 16322

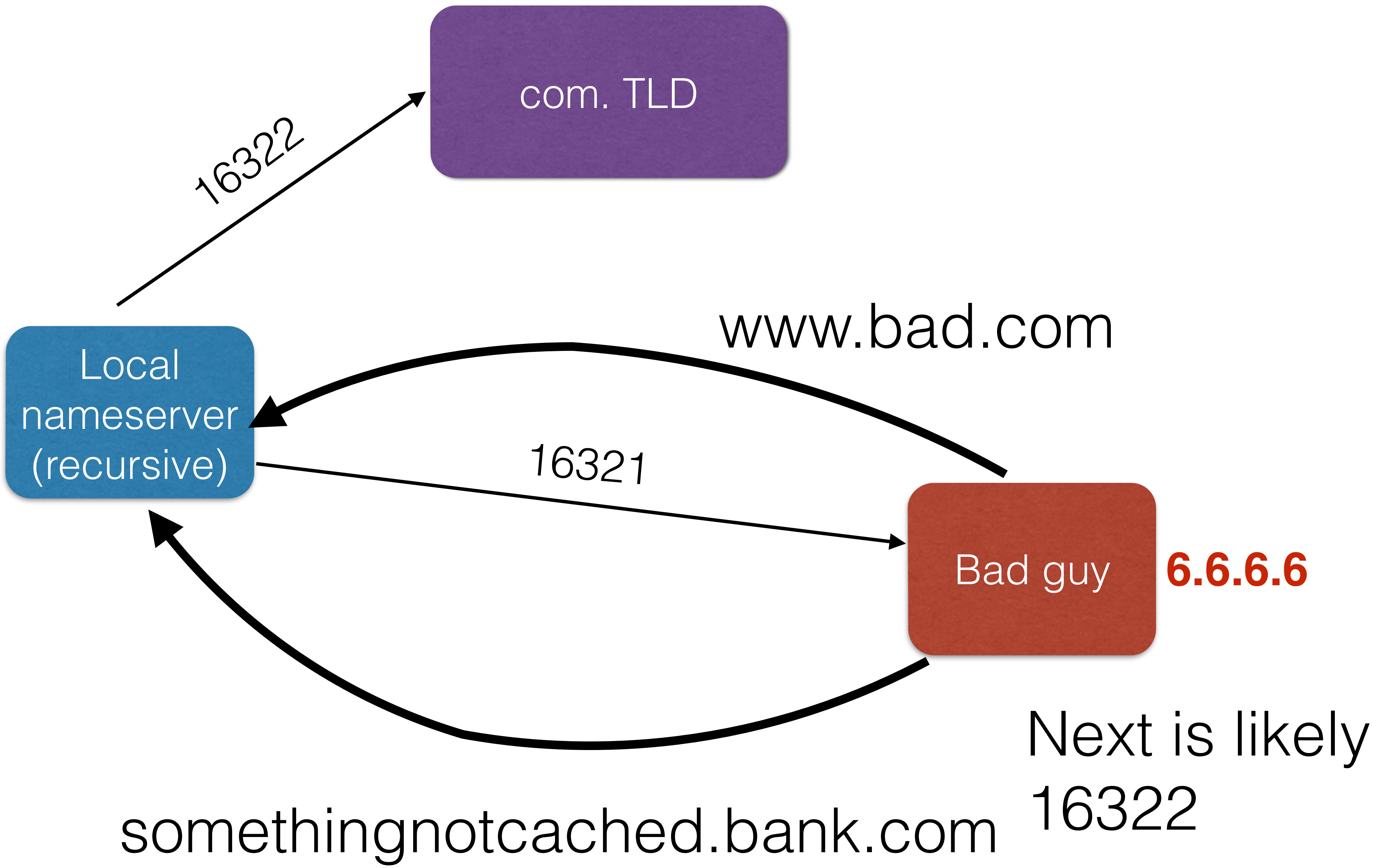
Cache poisoning

Can we do more harm than a single record?



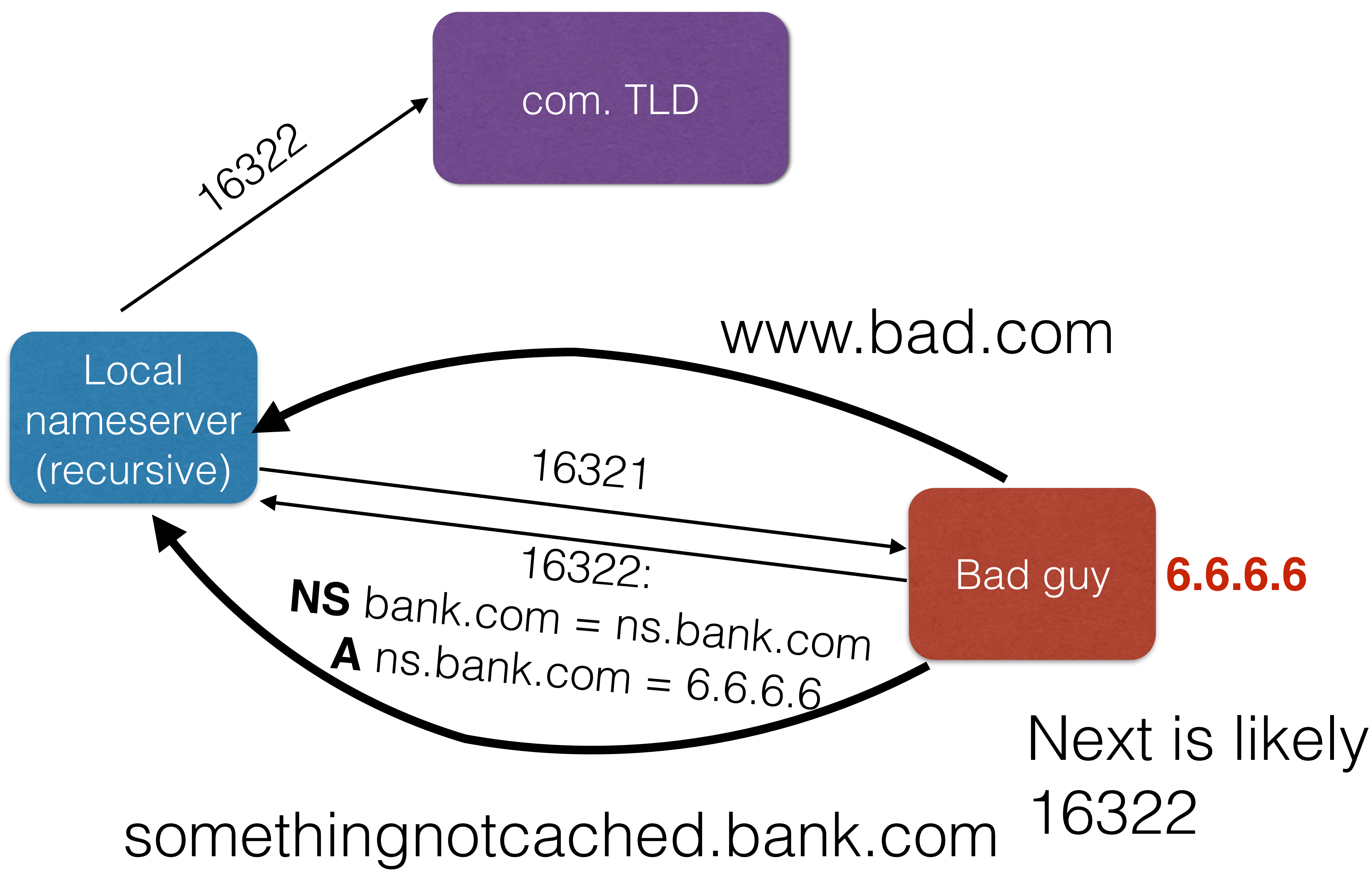
Cache poisoning

Can we do more harm than a single record?



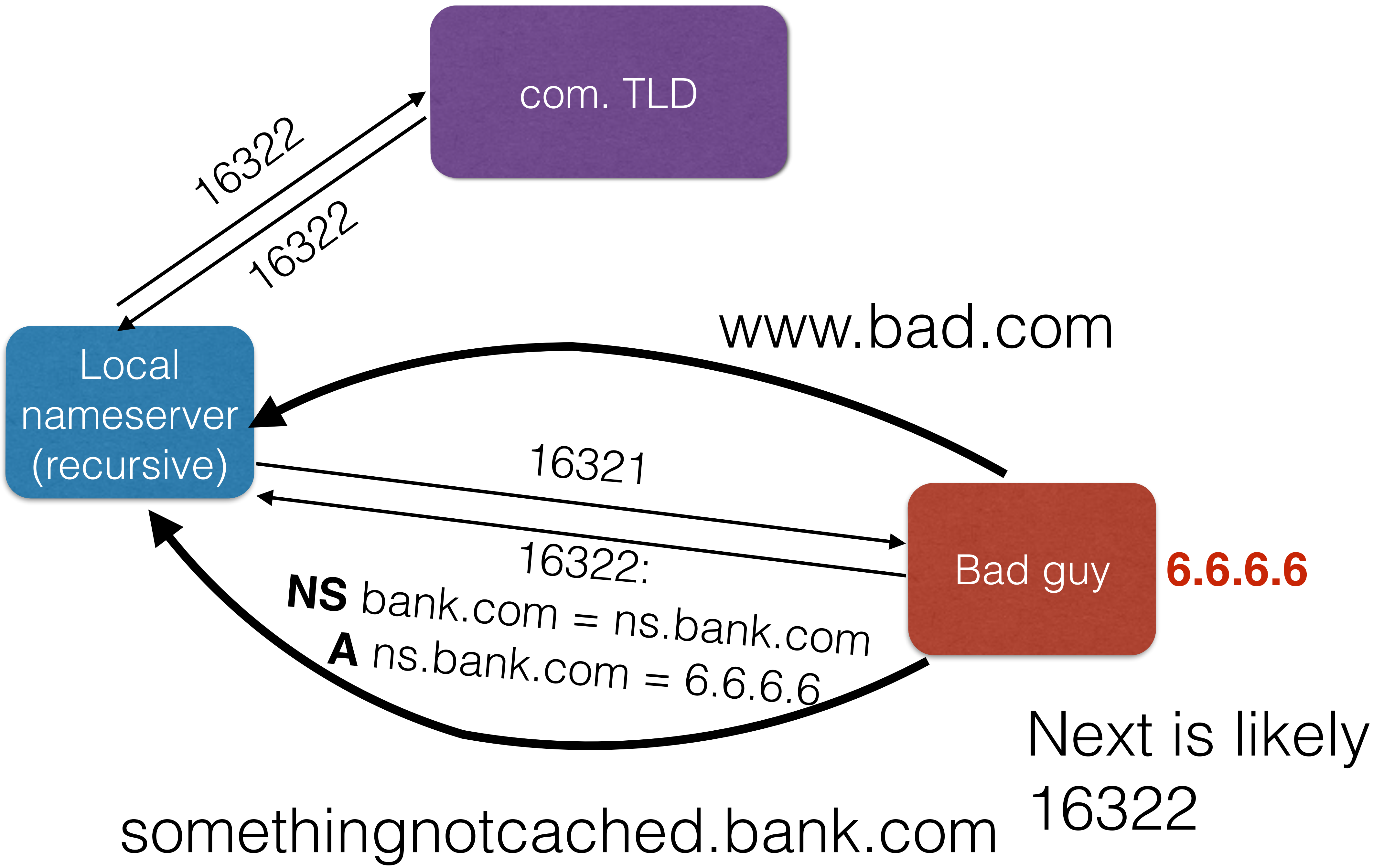
Cache poisoning

Can we do more harm than a single record?



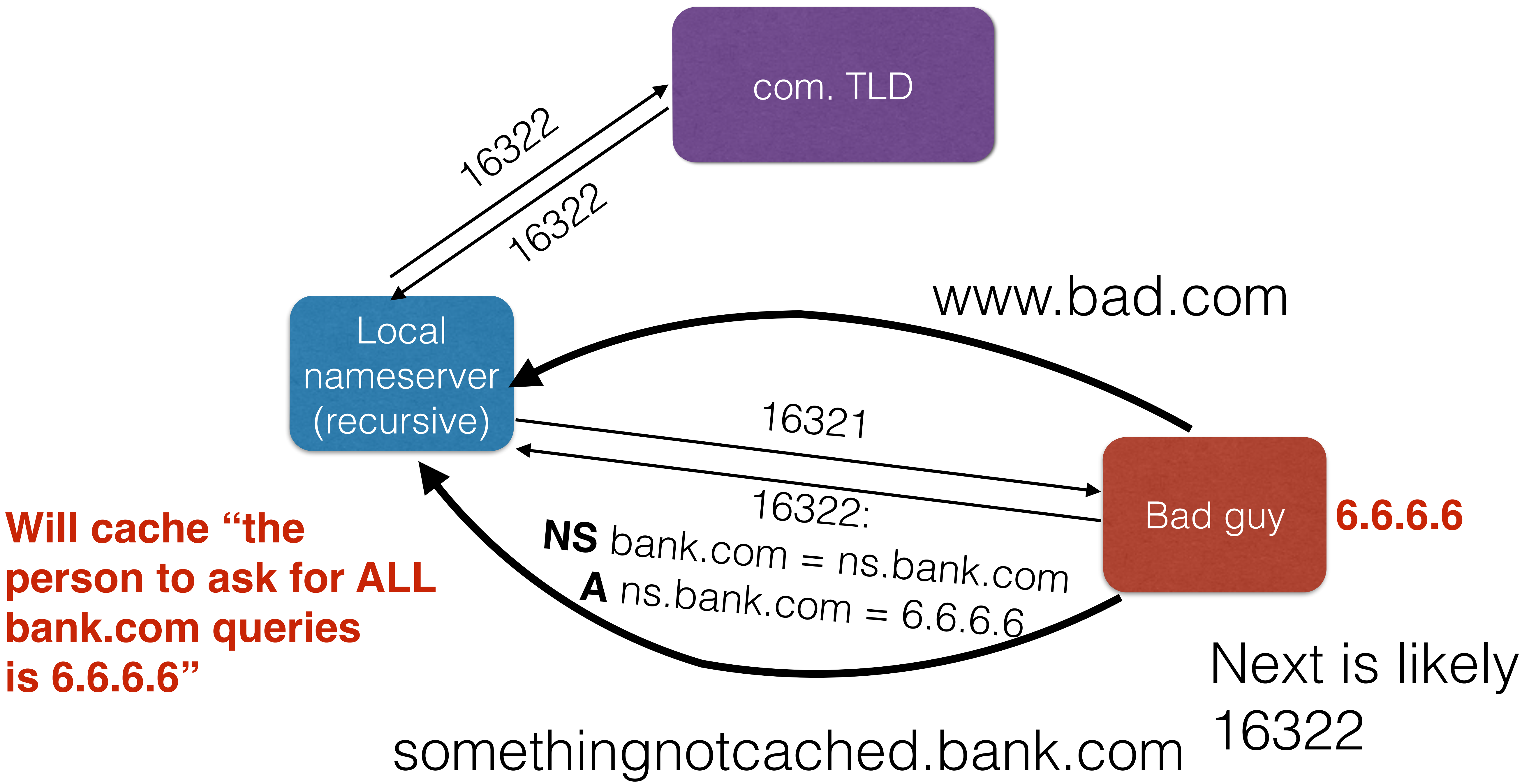
Cache poisoning

Can we do more harm than a single record?



Cache poisoning

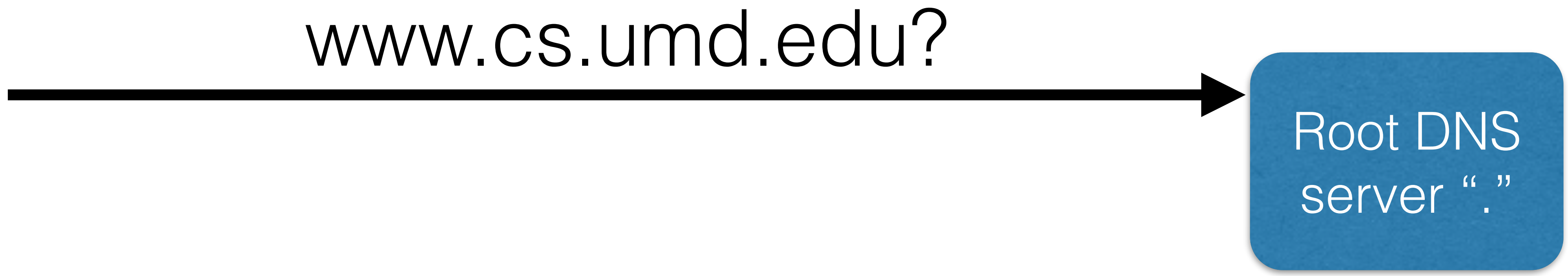
Can we do more harm than a single record?



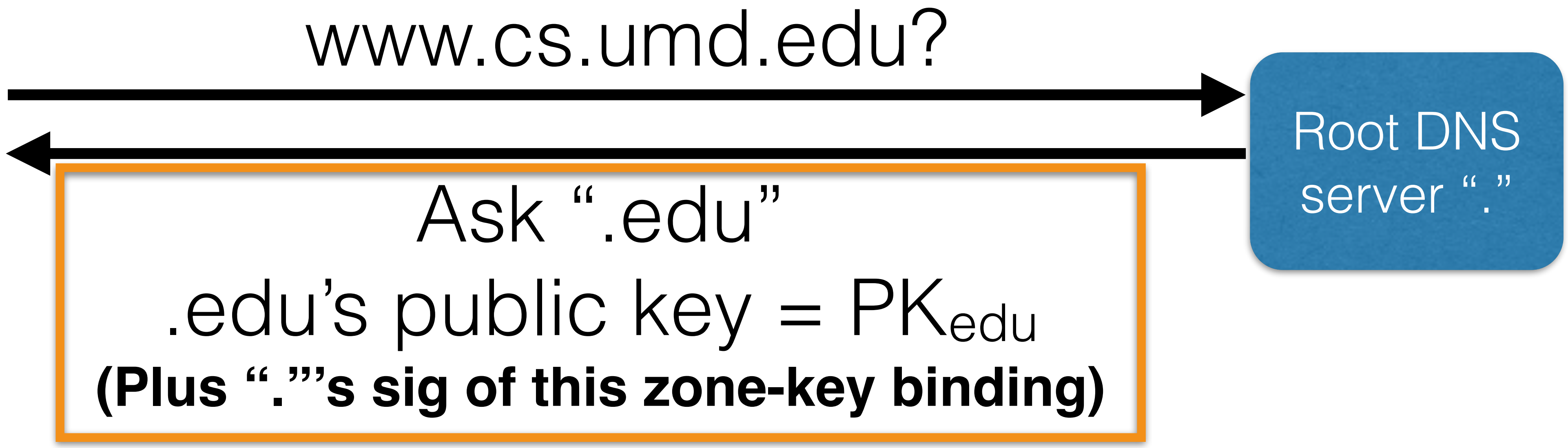
Solutions?

- Randomizing query ID?
  - Not sufficient alone: only 16 bits of entropy
- Randomize source port, as well
  - There’s no reason for it stay constant
  - Gets us another 16 bits of entropy
- DNSSEC?

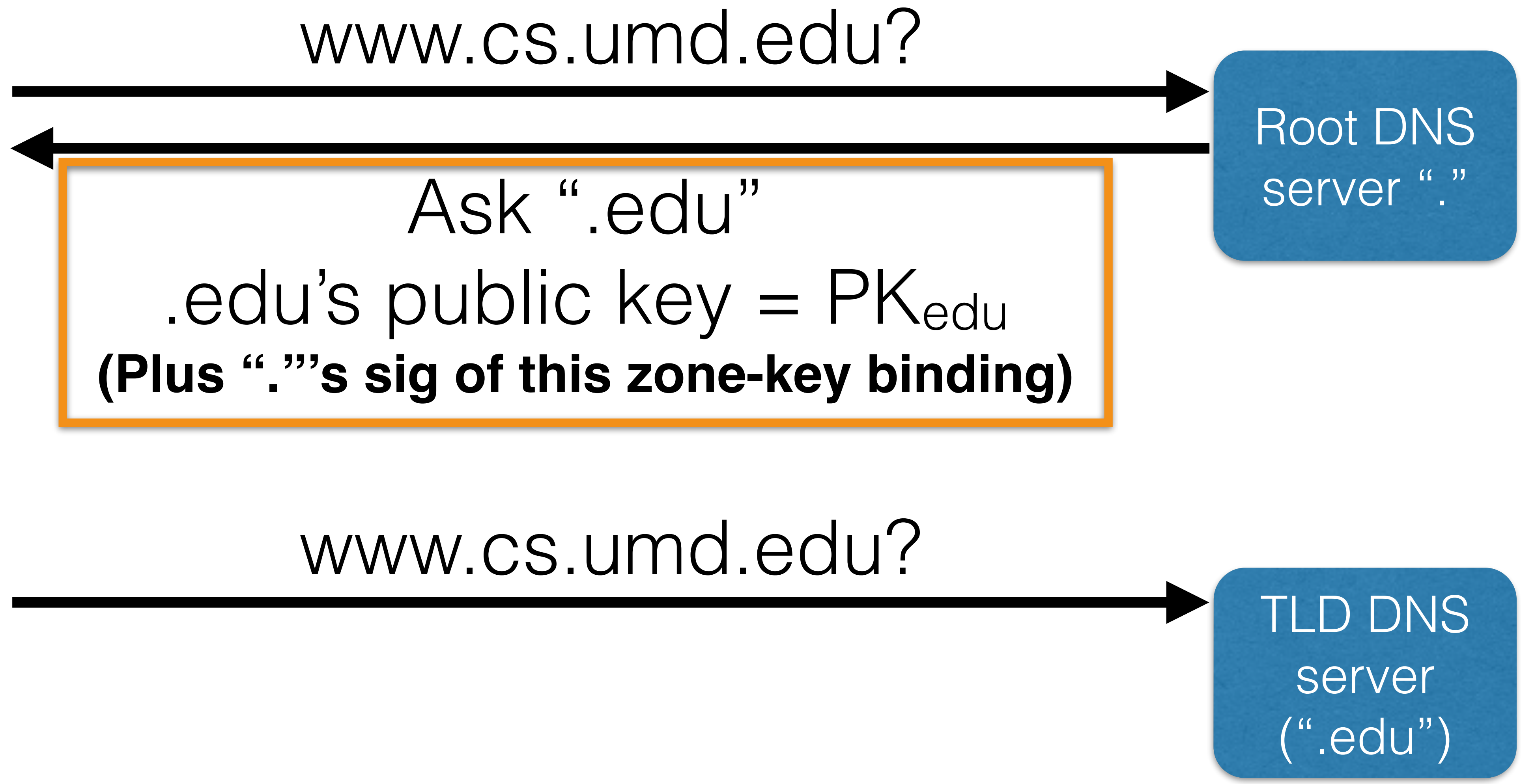
DNSSEC



DNSSEC

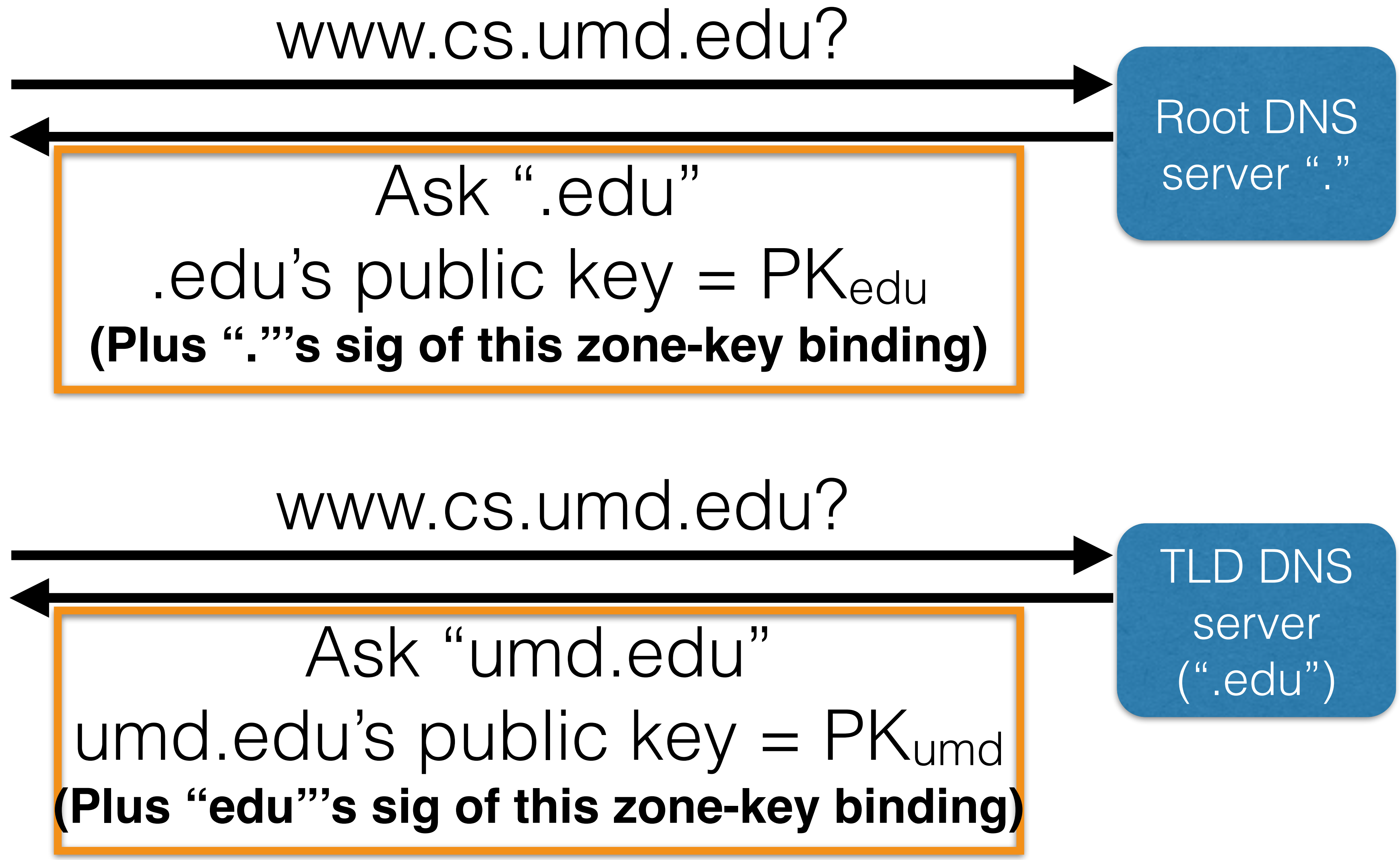

www.cs.umd.edu?
Root DNS server "."
Ask ".edu"
.edu's public key = PK_{edu}
(Plus "."'s sig of this zone-key binding)

DNSSEC

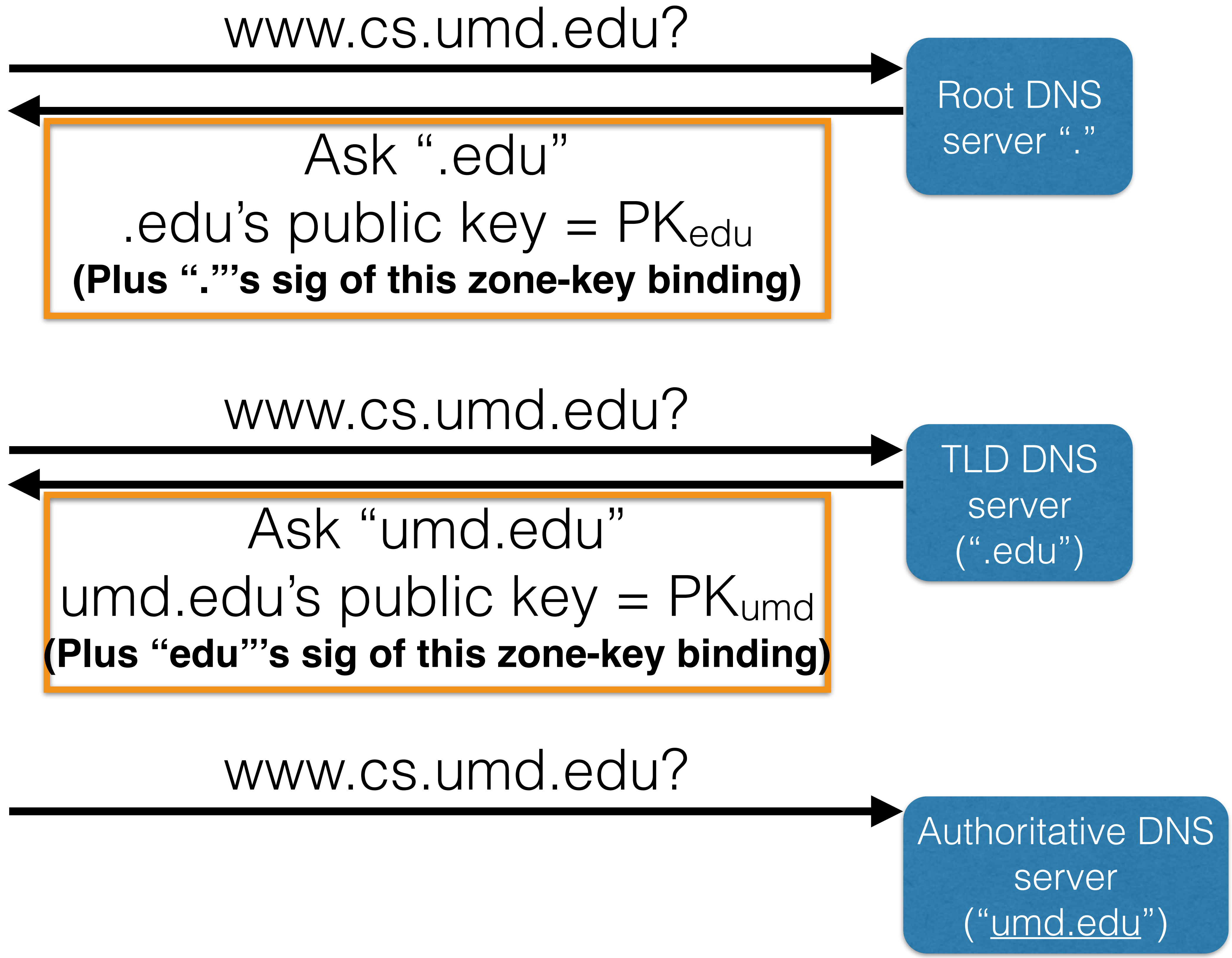


www.cs.umd.edu?
Root DNS server "."
Ask ".edu"
.edu's public key = PK_{edu}
(Plus "."'s sig of this zone-key binding)
www.cs.umd.edu?
TLD DNS server (".edu")

DNSSEC


www.cs.umd.edu?
Root DNS server “.”
Ask “.edu”
.edu’s public key = PK_{edu}
(Plus “.”’s sig of this zone-key binding)
www.cs.umd.edu?
TLD DNS server (“.edu”)
Ask “umd.edu”
umd.edu’s public key = PK_{umd}
(Plus “edu”’s sig of this zone-key binding)

DNSSEC



www.cs.umd.edu?
Root DNS server "."
Ask ".edu"
.edu's public key = PKedu
(Plus ".""s sig of this zone-key binding)
www.cs.umd.edu?
TLD DNS server (".edu")
Ask "umd.edu"
umd.edu's public key = PKumd
(Plus "edu""s sig of this zone-key binding)
www.cs.umd.edu?
Authoritative DNS server ("umd.edu")

DNSSEC

www.cs.umd.edu?

Root DNS server "."

Ask ".edu"

.edu's public key = PK_{edu}

(Plus "."'s sig of this zone-key binding)

www.cs.umd.edu?

TLD DNS server (".edu")

Ask "umd.edu"

umd.edu's public key = PK_{umd}

(Plus "edu"'s sig of this zone-key binding)

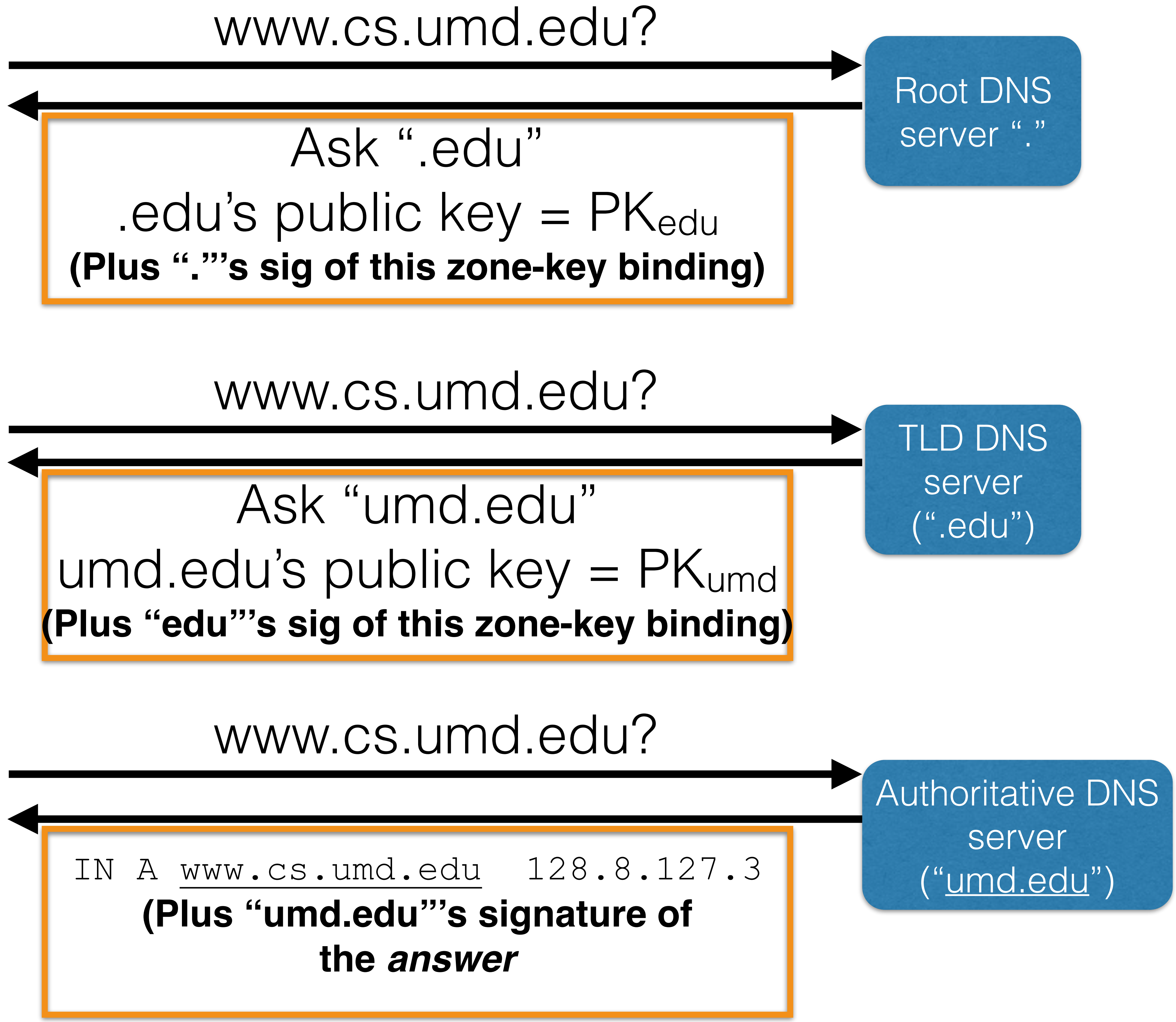
www.cs.umd.edu?

Authoritative DNS server ("umd.edu")

`IN A www.cs.umd.edu 128.8.127.3`

(Plus "umd.edu"'s signature of the *answer*

DNSSEC



Only the authoritative answer is signed

Properties of DNSSEC

- If everyone has deployed it, and if you know the root's keys, then prevents spoofed responses
  - Very similar to PKIs in this sense
- But unlike PKIs, we still want authenticity despite the fact that not everyone has deployed DNSSEC
  - What if someone replies back without DNSSEC?
  - Ignore = secure but you can't connect to a lot of hosts
  - Accept = can connect but insecure
- Back to our notion of incremental deployment
  - DNSSEC is not all that useful incrementally