Domain Name System

Cloud Security

CSE4059

Why Names, Not Addresses

- Everything last session assumed you already knew the address you wanted
  - Address? IP address such as 1.2.3.4

- But you have almost certainly never typed an IP address into a browser
  - You type *www.hanyang.ac.kr* → a human-readable name!

- Why names?
  - IP addresses are hard to remember

- That translation layer is the **Domain Name System (DNS)**

- It runs before every connection you make
  - Makes it a rather interesting thing to attack

Domain Name System (DNS)

- A distributed database that stores records that map domain names to values (e.g., IP address)
  - Ex) provides IP address for a domain name
  - *www.hanyang.ac.kr* → 166.104.177.20

- The address lookup is the one that runs before essentially every connection

- Various information can be stored
  - Which IP address is mapped to this domain?
  - Which email servers accept email for this domain?
  - What security-related data is associated with this domain?

DNS Hierarchy

Names are a "tree" (read right to left)

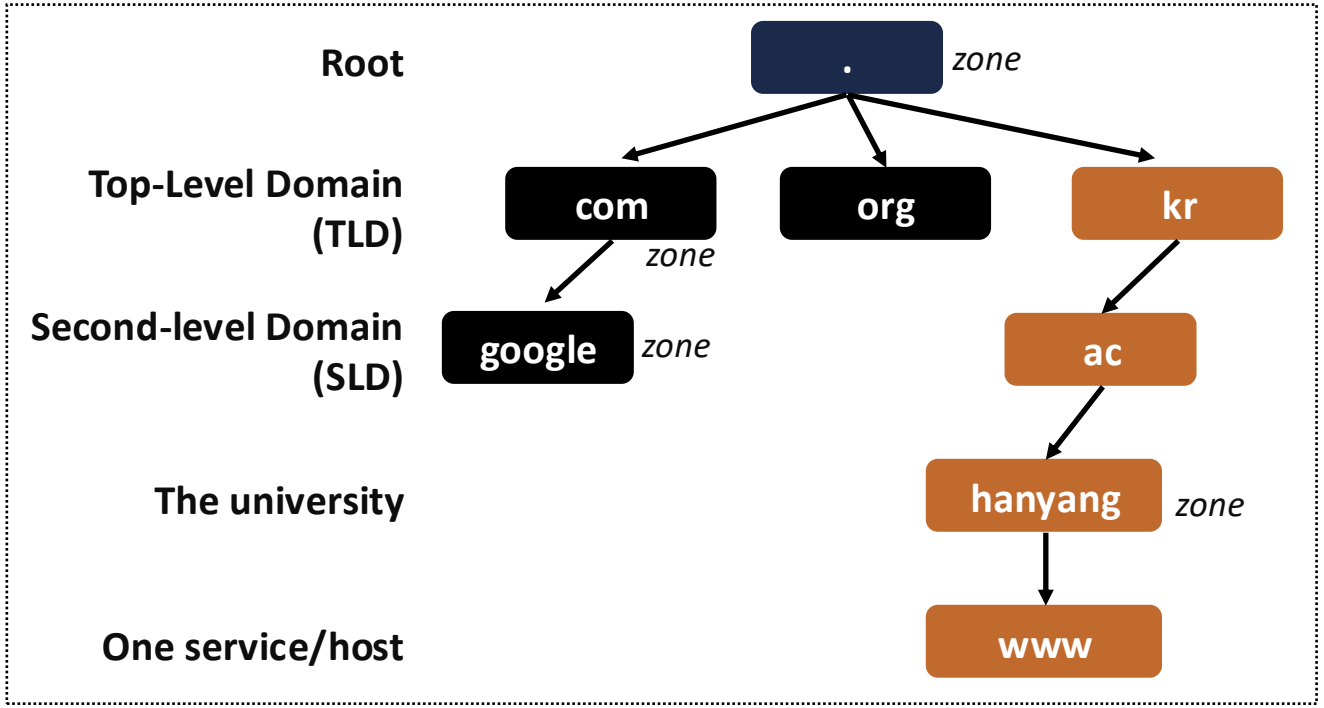
www.hanyang.ac.kr.

kr: a top-level domain (here, a country)

ac.kr: academic institutions in Korea

hanyang.ac.kr: this university

www.hanyang.ac.kr: one service in it



- Each zone can create names only under the part of the tree they control
  - What makes every name globally unique
- Each level knows only who is responsible for the level below it
  - No single DNS server that holds every name

Zone? a portion of the DNS namespace served by authoritative nameservers

DNS Root

- A root server does not know the address of *www.hanyang.ac.kr*
- What it knows is
  - "for anything ending in *.kr* (TLD), ask these servers"

- Each level knows only one thing
  - Who is responsible for the level below it
  - Security perspective: there is no server you could compromise to get every name

- The resolution walk is not a search through one database
  - It is a series of referrals
  - Ex) asking for directions from strangers, each of whom knows only the next street

Top-Level Domains (TLDs)

- Generic TLD (gTLD)
  - .com, .net, .org, .edu, …
- New generic TLD (gTLD)
  - Added later
  - .xyz, .shop, .site, .app, …
- Country-code TLD (ccTLD)
  - Used or reserved for a country
  - .kr, .uk, .us, .au, …
- More than 1,500 TLDs in the world

Why Not One Big Database?

- The obvious design is one big database
  - One server, every name in the world, everyone asks it

- How many (second-level) domains are in the world?
  - Around 400 million

- Why not centralized?
  1. Single point of failure: it goes down, and the entire internet stops working
  2. Traffic volume: every connection made anywhere on earth would query one machine
  3. Single point of update: one organization would control every name change on the internet

Distribution and Delegation

- Authority is delegated down the tree
  - The root delegates *.kr* to whoever administers Korean domains
  - They delegate *ac.kr* to whoever administers Korean academic domains
  - They delegate *hanyang.ac.kr* to this university, which runs its own servers for it

- Each organization operates the part of the tree it owns
  - Also, answers questions about it

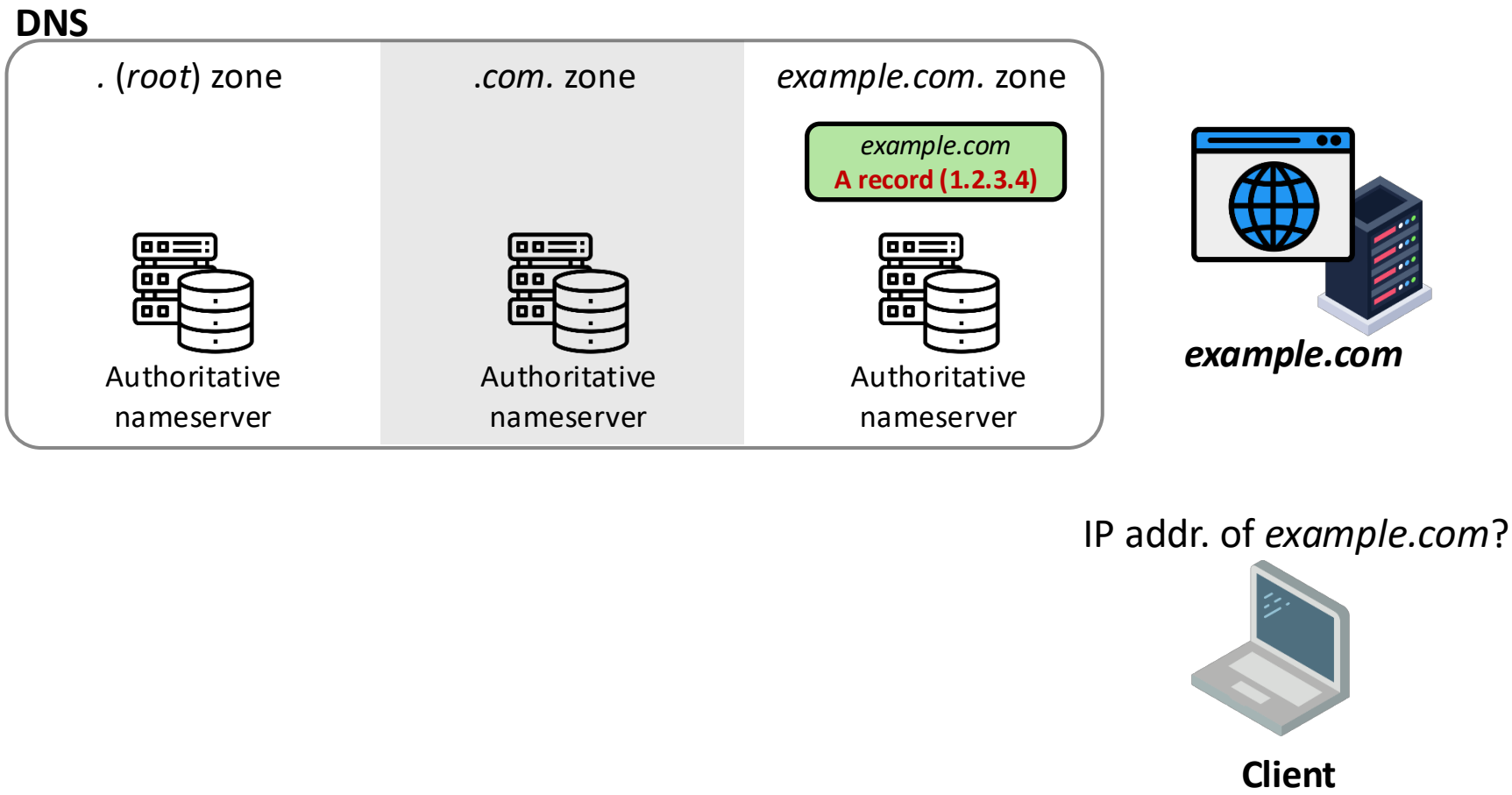
- Nobody holds the whole map
  - Not one server anywhere has a list of every name
  - No single failure point, load spread across thousands of organizations, and each organization controls its own names

DNS Resolution

Entity	What it does
Stub resolver	Sends a DNS query to the local resolver and receives the final answer
Local resolver	Finds the answer on behalf of the client and caches it Usually, managed by your network provider
Authoritative Nameserver	Holds authoritative DNS data for a zone
- Root zone	Refers the resolver to the appropriate TLD nameservers
- TLD zone	Refers the resolver to the authoritative nameservers for the domain
- Domain zone	Provides the requested DNS records

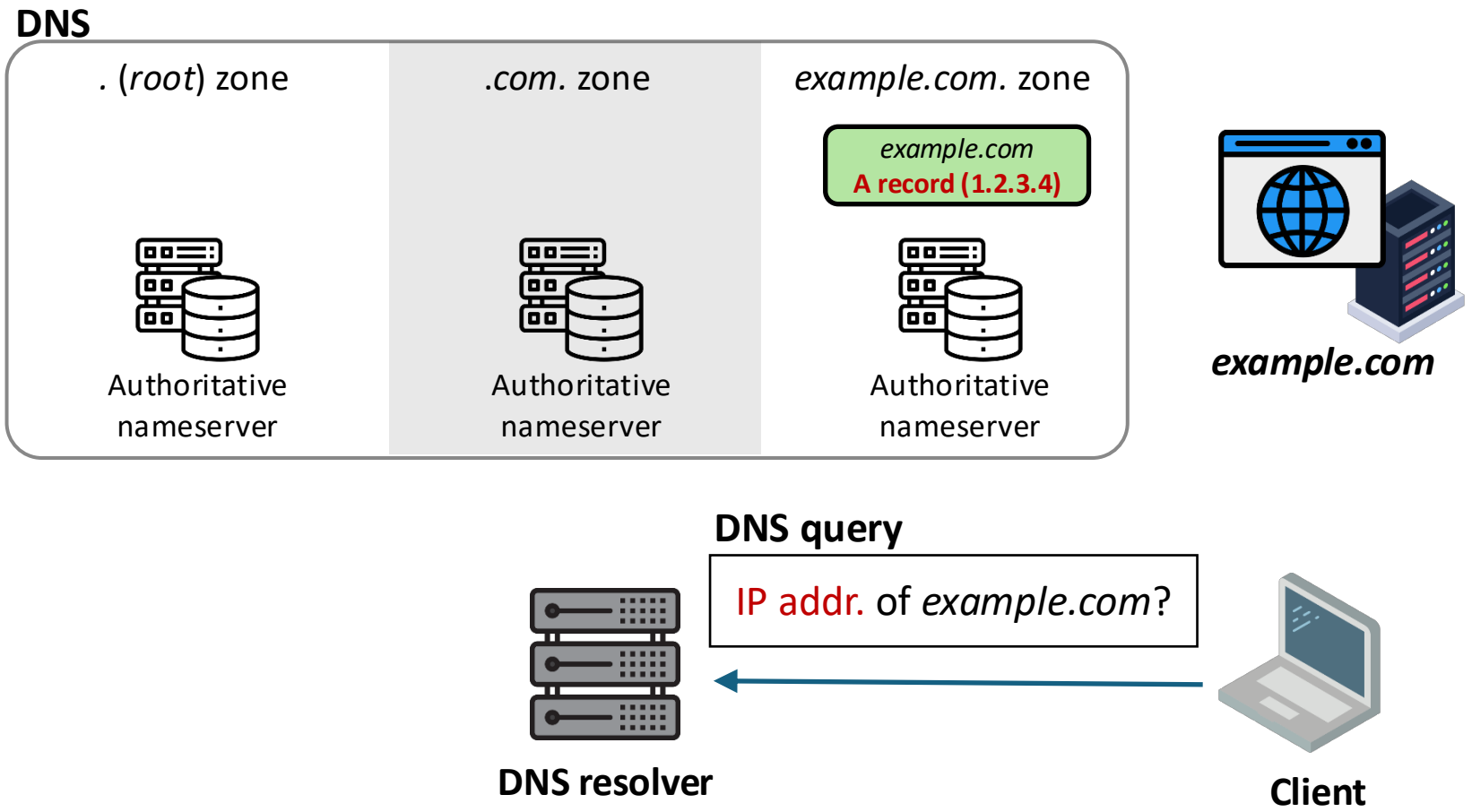
DNS Resolution

Example – DNS Resolution



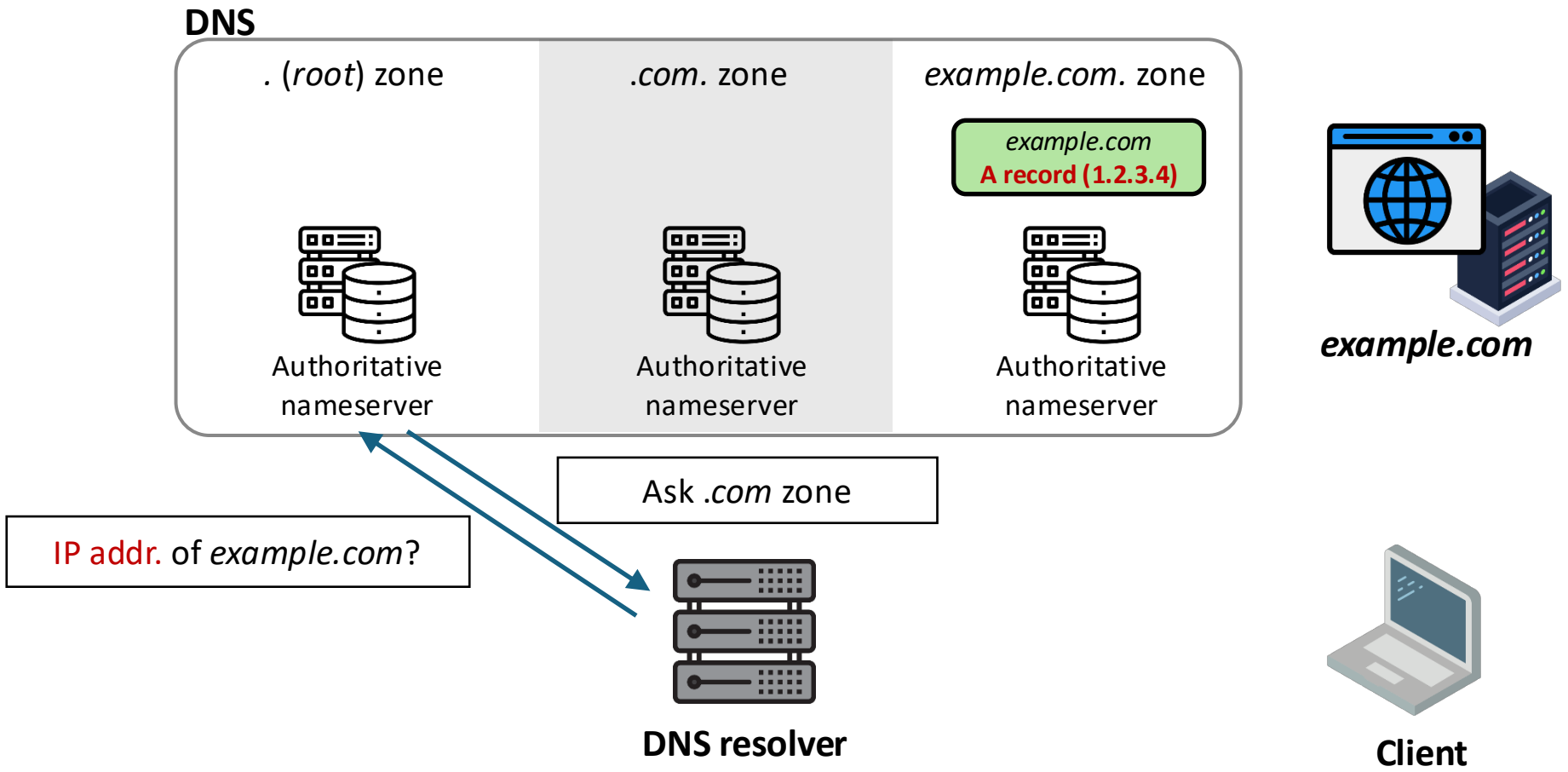
DNS Resolution

Example – DNS Resolution



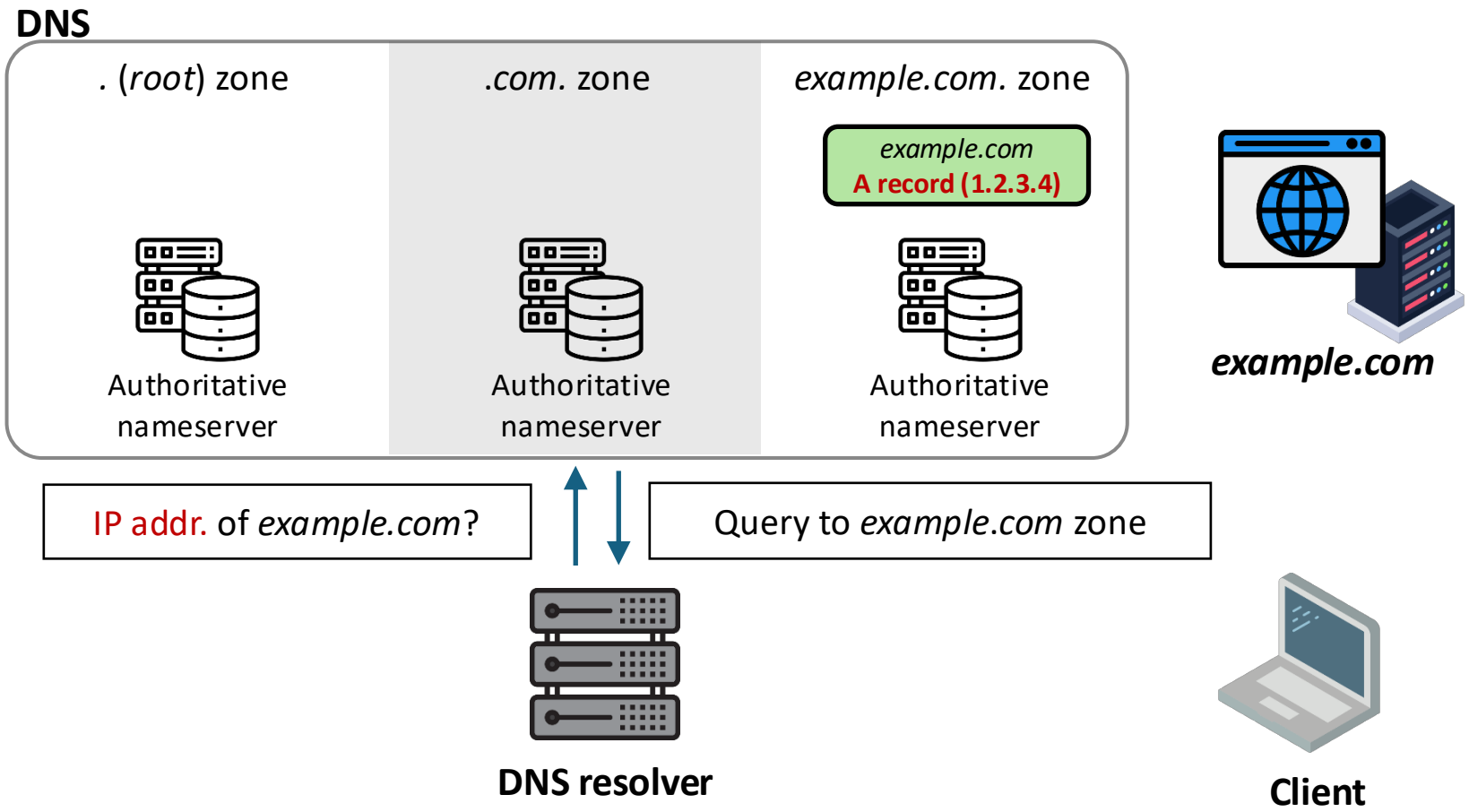
DNS Resolution

Example – DNS Resolution



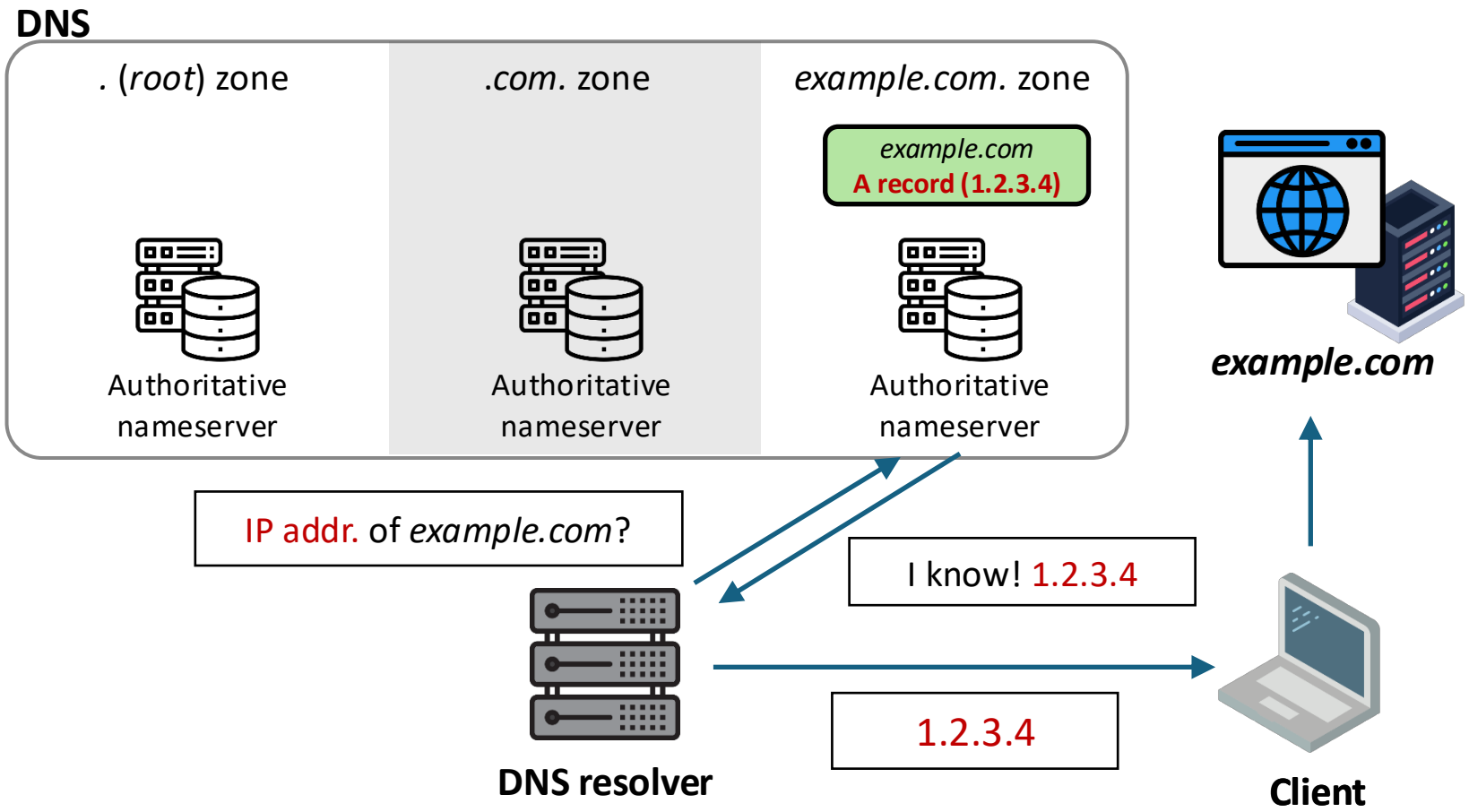
DNS Resolution

Example – DNS Resolution



DNS Resolution

Example – DNS Resolution



DNS Record

- DNS record format
  - NAME: which name this is about
  - TYPE: which kind of information this is
  - CLASS: essentially always "Internet"
  - TTL: how long it may be cached
  - DATA: the actual answer, whose format depends on the TYPE

- A DNS lookup is: "*give me all records of TYPE X for NAME Y*"
  - Records of the same name and type form a *set*

```
QUESTION:
    www.hanyang.ac.kr        IN      A

ANSWER:
    www.hanyang.ac.kr  300   IN      A       166.104.177.20

    (name)           (TTL) (class) (type)  (data)
```

DNS Record Types

Type	Maps a name to	Example
A	An IPv4 address	*google.com* → 142.250.12.102
AAAA	An IPv6 address	*google.com* → 2607:f8b0:4023:402::66
NS	The authoritative servers for a zone	*google.com* → *ns1.google.com*
MX	The email servers for a domain	*gmail.com* → *smtp.google.com*

⋮

more than 90 record types

DNS Resolver

- Why not have your own machine do the DNS lookup itself?

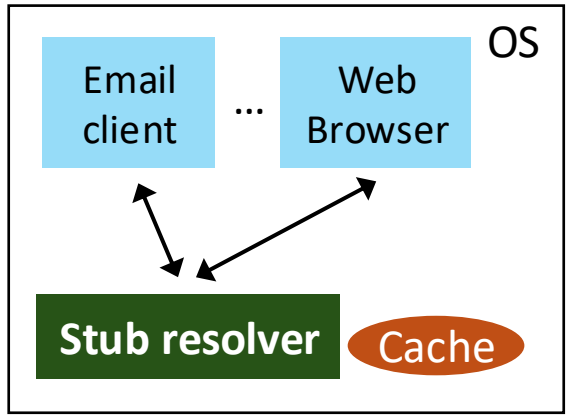
- If every machine do the whole walk, every time
  - The root and TLD servers would receive vastly more traffic

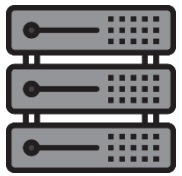
- With a shared local resolver, one lookup serves everybody on the same network (*caching*)
  - The first person to ask pays the full cost
  - Everyone after them gets an instant answer

- Security consequence
  - The local resolver answers for *many* people
  - Whoever controls it or corrupts what it stores → affects all of them

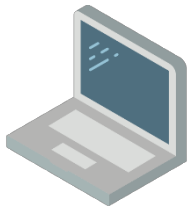
DNS Resolver – Stub and Local Resolver

- **Stub resolver**
  - Not interact with the zone hierarchy
  - Sends basic queries to recursive resolvers
  - May cache answers
- **Local resolver**
  - Performs name lookup on behalf of client
  - Usually located on the local network (ex. managed by internet service providers)
  - Caches DNS responses





Local resolver



Client

DNS Resolver – Open/Public Resolver

- A DNS resolver that is willing to provide recursive DNS lookups for anyone
  - Google DNS (8.8.8.8)
  - Cloudflare (1.1.1.1)
  - Quad9 DNS (9.9.9.9)
  - NextDNS
  - CleanBrowsing
  - …

Why a Resolver Is an Attractive Target

- The facts we know
  - A resolver caches DNS records
  - A resolver answers queries for *many* clients
- The ratio is what makes it interesting
  - One target, many victims
- Attackers can
  - Poison one client → affects that client
  - Poison a shared resolver cache → can affect many clients until the cached record expires

How Long Can The Answer Be Reused? TTL

- Every DNS record carries a Time To Live (TTL)
  - A number of seconds
- It is set by whoever is authoritative for the record (e.g., a domain owner)
  - It is their statement of how long it may be reused
- A resolver keeps the answer for that long, then discards it and asks again
- The TTL is a deliberate trade-off between agility and load
  - Short TTL (say 60s): changes take effect quickly, but far more queries reach the authoritative servers
  - Long TTL (say 24h): very efficient, but a change takes up to a day to be seen by everyone

An Example: TTL

Time	What happens	What the resolver holds
12:00:00	Student A queries the name Resolver performs a DNS lookup, gets the answer, TTL = 300	The answer, good until 12:05:00
12:00:30	Student B asks the same name	Served from cache
12:03:00	The record's owner (e.g., domain owner) changes the address at the authoritative nameserver	Still the old answer; It has no reason to query again yet
12:04:59	Student C queries	Still the old answer
12:05:01	Student D queries	The cache entry has expired Resolver performs a new lookup, gets the fresh answer

- Every party behaved correctly, but some students still received a stale answer

Consequence 1: Changes Propagate Slowly, and Unevenly

- You move a service to a new address and update the record
- Who sees the change, and when?

- Resolvers that have no cached record
  - They fetch the new answer immediately
- Resolvers holding a cached record
  - They continue using it until its TTL expires
  - There is normally no way to invalidate that cached answer before expiration
  - For a period, different clients may receive different answers

- When migrating a service
  - Lower the TTL in advance,
  - Wait for old cached records to expire,
  - Make the change,
  - Then increase the TTL again

Consequence 2: A Wrong Answer Persists

- If a resolver caches a *wrong* answer, it serves that wrong answer to everyone using it
  - Until the TTL expires

- One bad entry, in one resolver, affects every client on that network
  - For a duration the record's TTL controls

- What if the wrong answer got there *on purpose*?

  → DNS security

Non-Existent Domain and Negative Caching

- Ask for a name that does not exist and you do not get silence
  - NXDOMAIN response: "*non-existent domain*"
- A negative answer is cached too
- Why?
  - Without negative caching, repeated queries for the same nonexistent name would repeatedly trigger DNS resolution
- Consequence: after you create a new DNS name, some resolvers may still return NXDOMAIN
  - They may have cached the earlier negative answer
  - They will retry only after the negative cache entry expires

DNS Port

- DNS uses port 53
  - Both UDP and TCP

- Most ordinary DNS queries use UDP
  - One small query, usually, one small response
  - A TCP handshake would cost three messages before the question was even asked
  - Setting up a connection would cost more than the thing you wanted to send!

- If no response arrives, the resolver can retry

- TCP is also used when needed
  - Ex) when a UDP response is truncated because it is too large

Transaction ID

- DNS needs a way to match each response to the corresponding query
  - A resolver may have many questions outstanding at once
- Every query carries a transaction ID
  - A number the resolver picks, which the answer must echo back
- When a response arrives, the resolver checks whether the transaction ID matches
- If the ID does not match, the response is rejected

A DNS Answer Carries No Cryptographic Proof

- When a DNS answer arrives, what does a resolver check?
  - Did it came from the server it queried?
  - Did it arrive at the source port it used?
  - Does the transaction ID match?
  - Does the question match the query?

- In traditional DNS, these checks provide no cryptographic proof
  - No signature. No proof of origin. No integrity check

- An attacker who can forge the expected response may potentially satisfy these checks
  - A forged response may be accepted if it matches the outstanding query
  - If cached, the false answer may then be served to many clients until its TTL expires

- This is a limitation of the original DNS design.. DNS was designed in 1983
  - We will later see how DNSSEC adds cryptographic authentication

Summary

- DNS is a distributed database that stores records that map domain names to values (e.g., IP address)
  - Nobody holds the whole map
  - Each level knows only who is responsible below it
  - DNS resolution walks root → TLD → authoritative
- TTL says how long an answer may be cached and reused
- One resolver serves many clients
  - A wrong answer there affects all of them at once
- A DNS answer carries no proof of where it came from