CSE589 · Naming

DNS
Names, and the machine that resolves them

The Domain Name System translates domain names into IP addresses. This lecture covers what it stores, how that data is distributed, how a single lookup works, and how a new domain is published.

Yaxiong Xie

Why this matters

You have never typed the address of a website

chatgpt.com

and every other service you use, the same way

google.com youtube.com amazon.com github.com buffalo.edu

the machine you reach is identified by a 32-bit number

and you have never seen it

Every service you use is reached by a name.

Why this matters

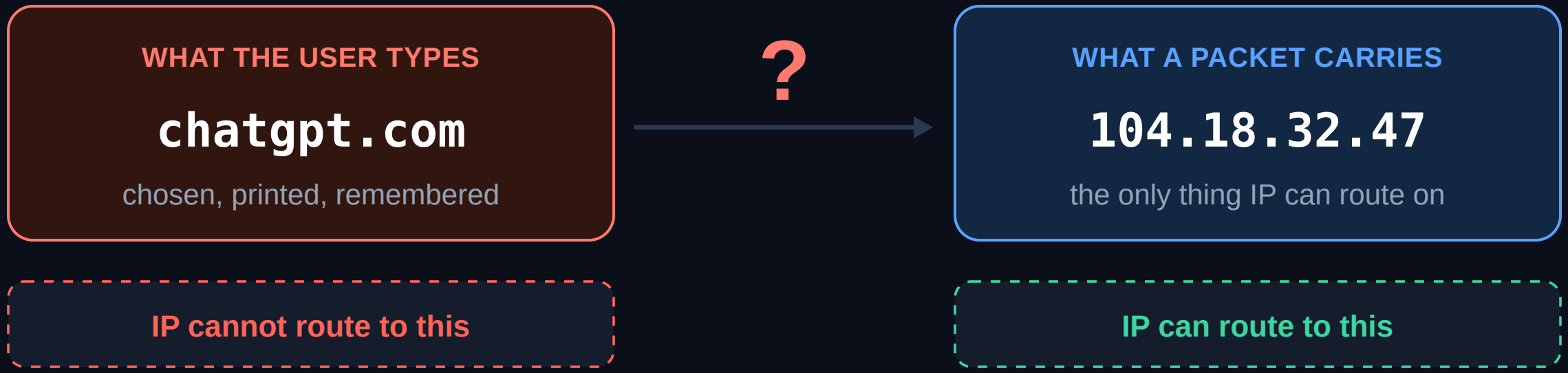
A name people remember is worth paying for

ai.com	$70,000,000	2025
voice.com	$30,000,000	2019
360.com	$17,000,000	2015
chat.com	$15,500,000	2023 · now OpenAI's
NFTs.com	$15,000,000	2022
rocket.com	$14,000,000	2024

a domain sale transfers a name and nothing else — no hardware changes hands

These are the highest publicly reported prices paid for a single name.

A packet can only be addressed with a number



something has to turn the left side into the right side

before a single packet leaves your machine

That translation is the Domain Name System.

Why do we need domain names at all?

01

Why names exist

Being easy to remember is the obvious reason, and a shared text file would satisfy it. Five more reasons follow, and the last of them is what forces a distributed protocol.

Reason 1

Memorability is the obvious reason, and the weakest one

IP address	domain name
104.18.32.47	chatgpt.com
142.251.150.119	www.google.com
140.82.113.4	github.com
128.205.201.56	buffalo.edu

the same four services you just saw, identified two ways

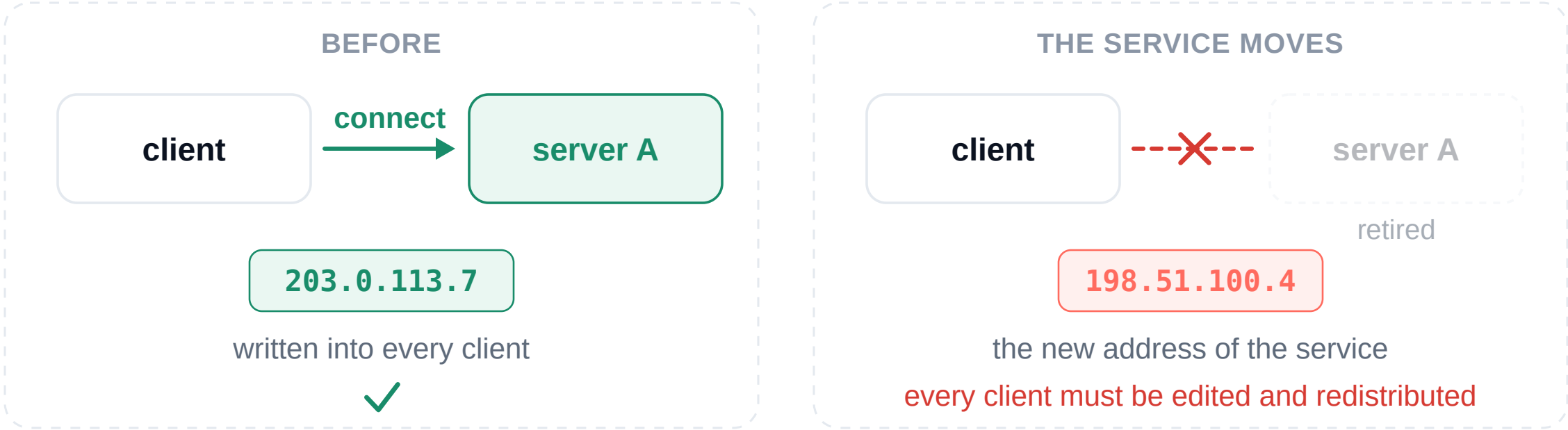
Everything on the last three slides is explained by this one reason, and it is a real one. But it only asks for a lookup table with two columns. It does not ask for a distributed protocol, and it is the first of six.

So why is DNS not just a shared file? Five more reasons follow.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

Reason 2

An IP address identifies a location, and locations change



An IP address describes where a machine currently sits in the routing system. It changes when the service moves to different hardware, a different provider or a different network. Clients should not have to be updated when that happens.

The name stays constant while the address is allowed to change.

Reason 3

One service can run on many machines

ONE USER REQUEST

google.com

the same name

ONE SERVICE, MANY MACHINES

142.251.32.110

142.251.40.238

142.250.65.174

172.217.12.206

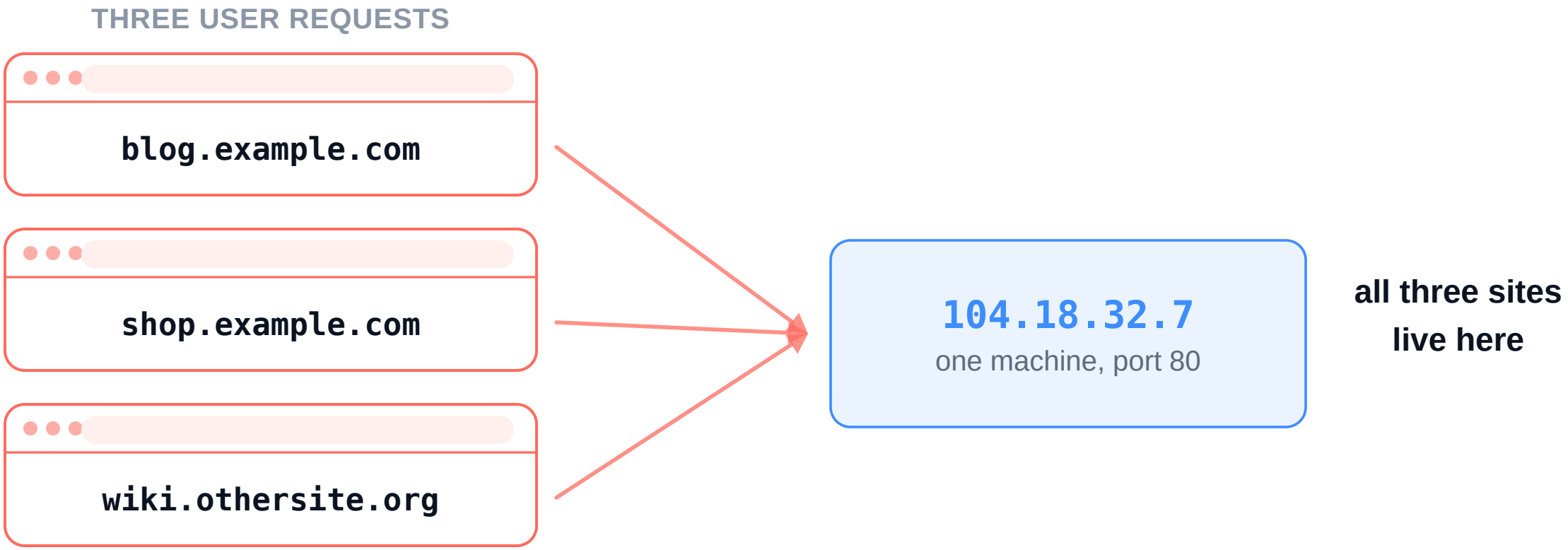
142.251.35.174

any one of these is a correct answer to that request

One name must be able to map to many addresses.

Reason 4

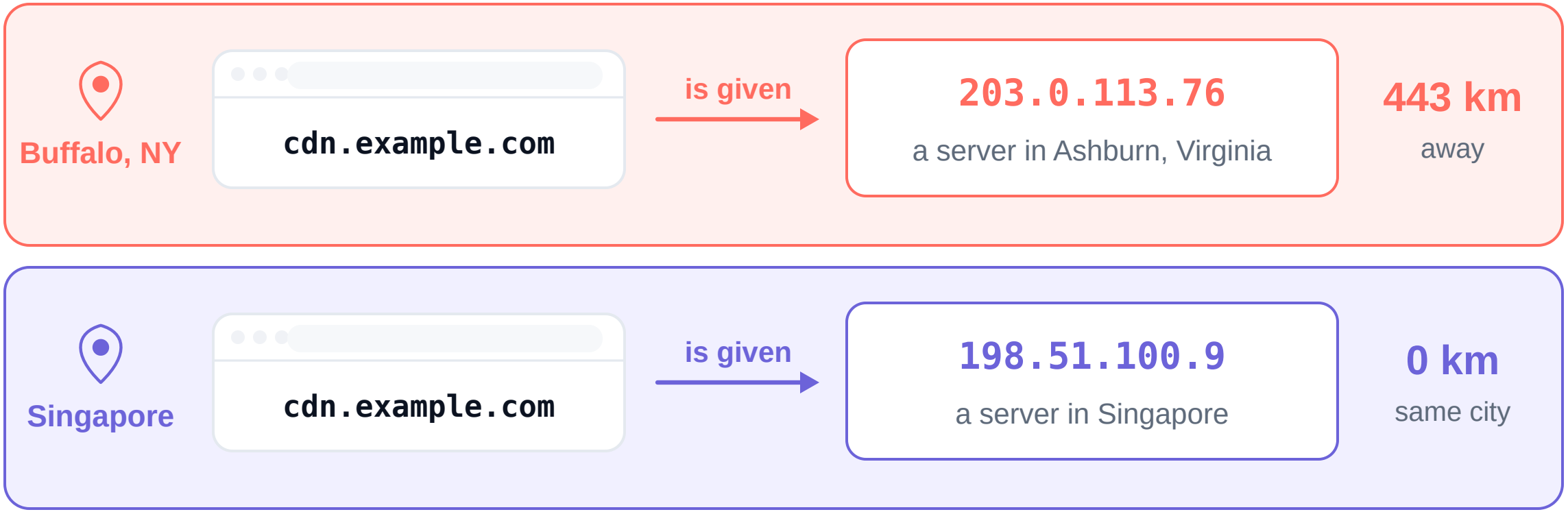
One machine can serve many different names



The mapping between names and addresses is many-to-many.

Reason 5

The best address depends on where the client is



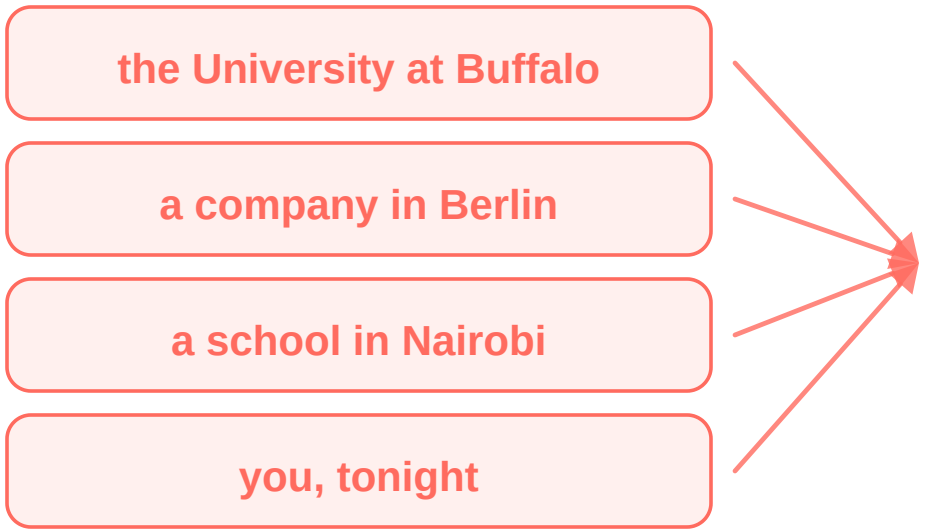
had Singapore been given the Ashburn address, every packet would travel 15,500 km

The same name, asked from two places, has two correct answers.

Reason 6

With one table, one organisation decides every name

`you want to create lab7.cse.buffalo.edu`



ONE ORGANISATION

owns the table

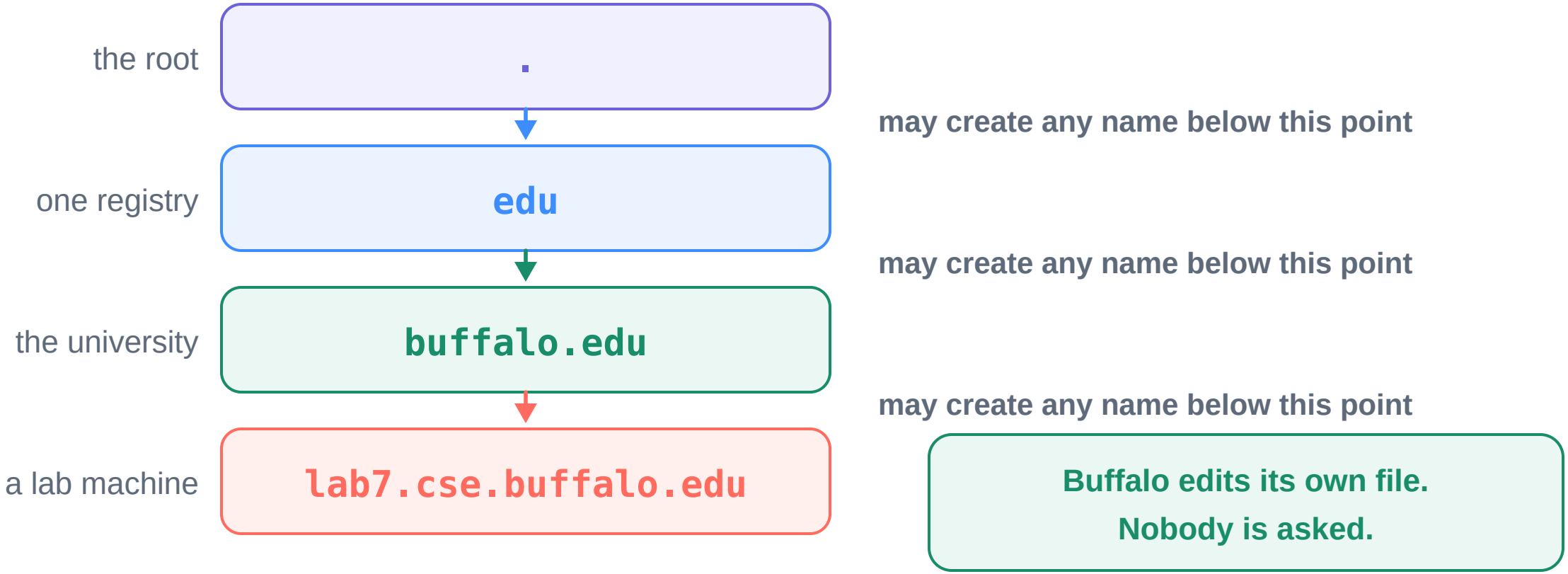
and edits every row

- it approves every new name on Earth
- it decides which names may exist
- nothing is added without it

no amount of copying changes who holds the pen

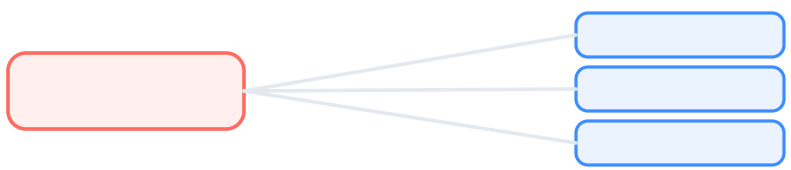
One file means one gatekeeper for the whole Internet.

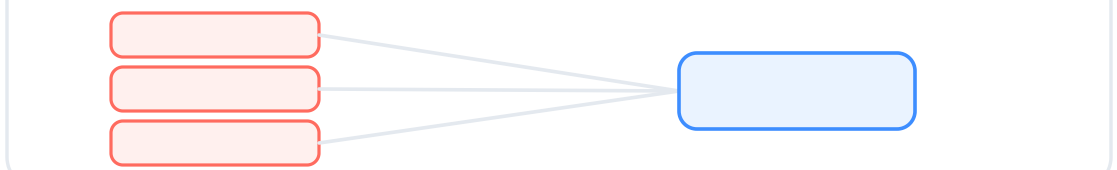
Delegation hands naming authority down the tree

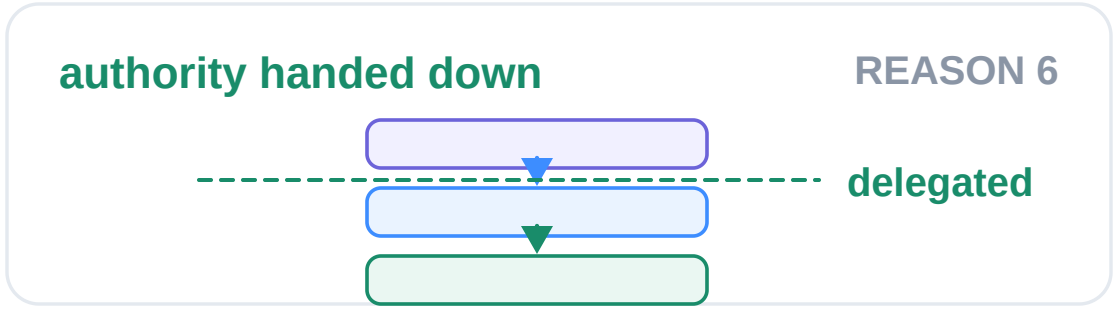


Finding which of those files holds a name is what needs a protocol.

All six reasons ask for one thing: a mapping you can look up

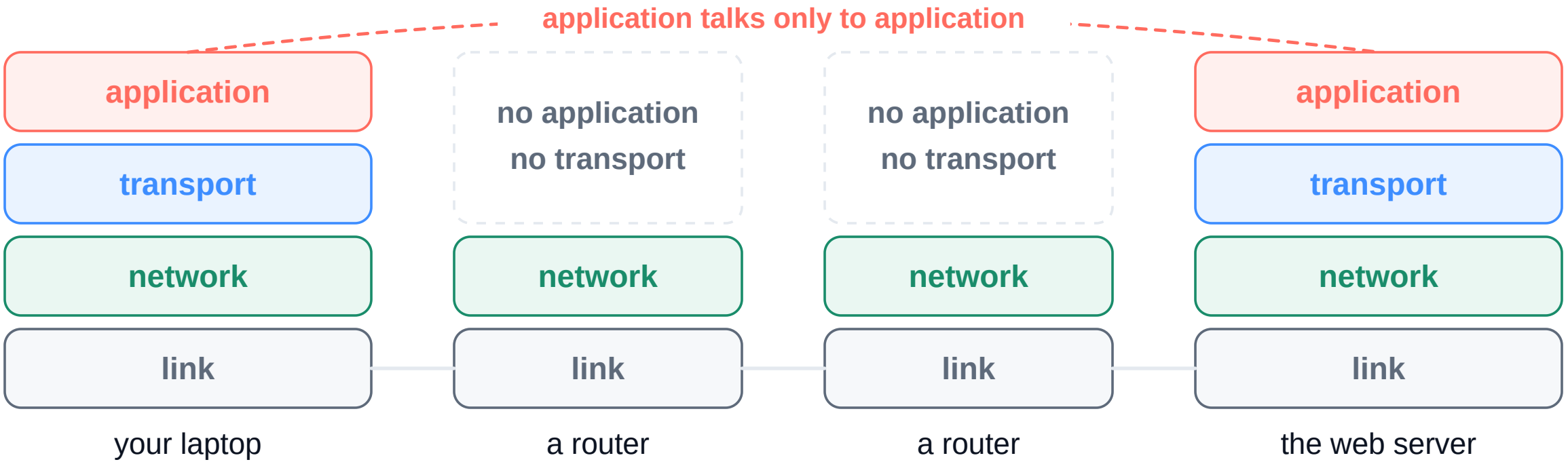






DNS is that mapping, and these four properties are its job description.

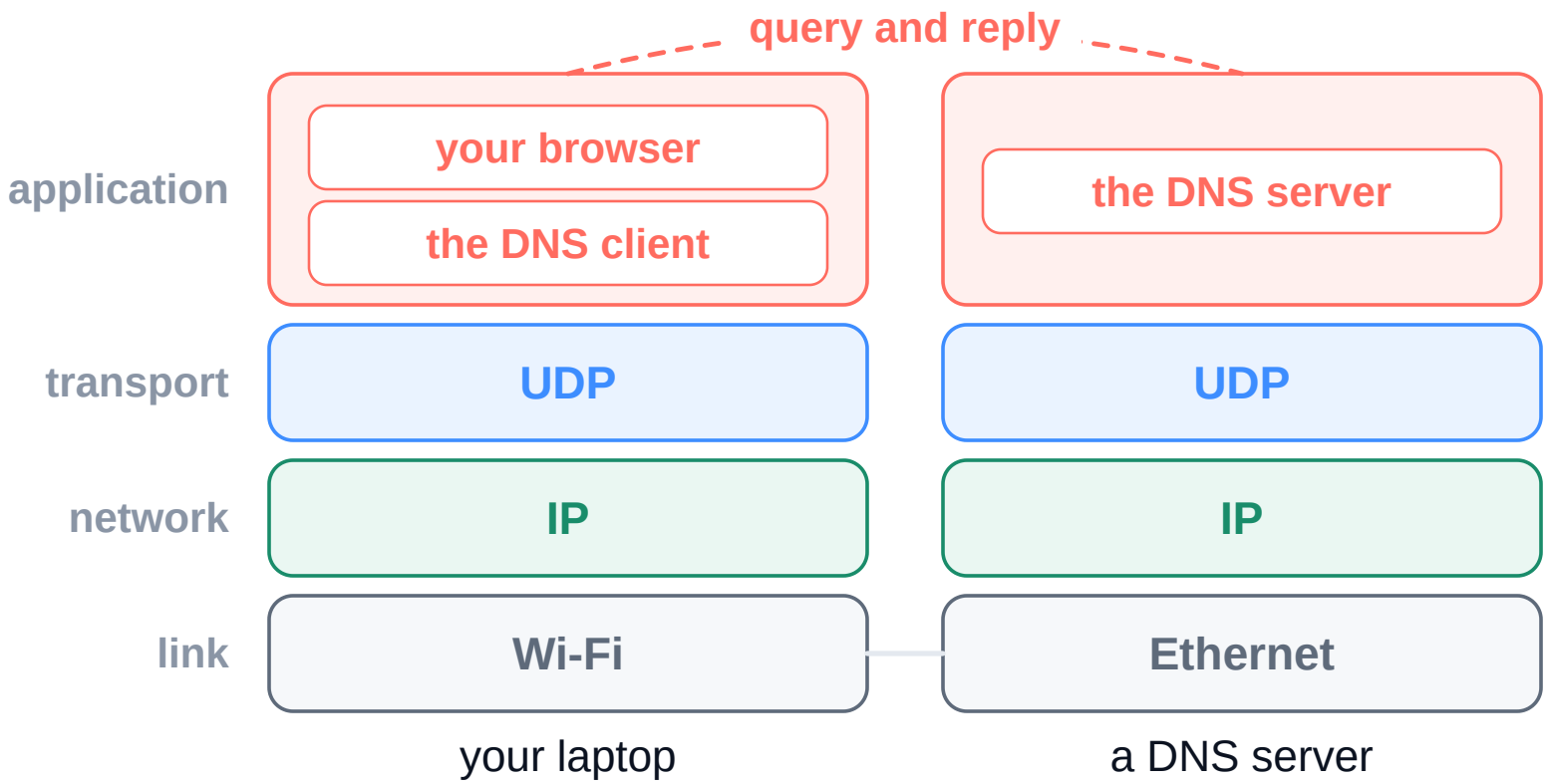
Only the end points run the application layer



a router in between reads the IP header and forwards; nothing above it

Above IP, only the two ends do anything. The network in between does not.

DNS is an application-layer protocol, like HTTP



WHY THIS IS AN APPLICATION

a user-space process

sends and receives UDP

no kernel support

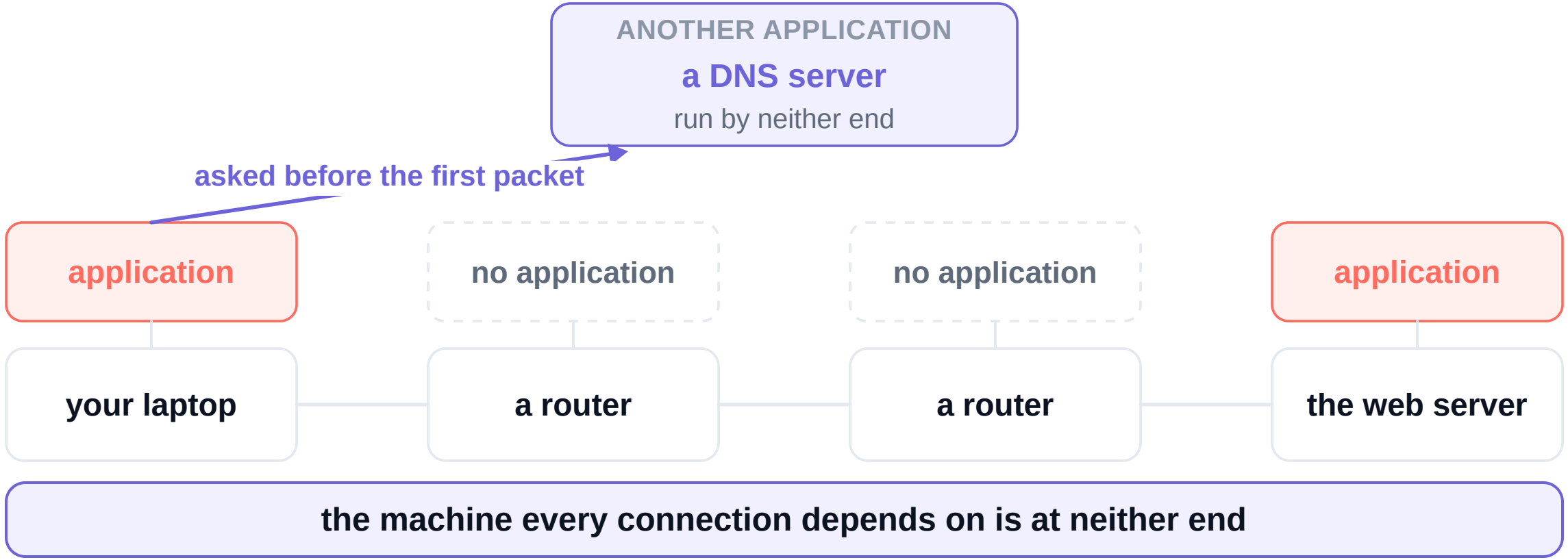
no new layer in the stack

you could write one yourself

an ordinary application

Both ends are ordinary hosts. A DNS server is just another end point.

An application that became network infrastructure



The model puts applications at the edge. This one became infrastructure.

You do not see it until it stops

19–20 OCTOBER 2025

one empty DNS record

a race condition in one provider’s DNS automation left a regional endpoint resolving to nothing

EC2	Lambda	ECS	EKS
Fargate	STS	IAM	Redshift

15 hours of cascading failure

WHAT MAKES IT INFRASTRUCTURE

it runs before every connection

no application shows it to you

no single company owns it

thousands of independent

operators keep it running

and you notice none of it

An ordinary application, and the network cannot run without it.

Where this goes

What the rest of this lecture answers

02 What does the simplest name service do?

03 Why can one server not hold all of it?

04 How is a single name resolved?

05 Why is it fast?

06 How do I publish a name of my own?

Part 2 begins with the simplest thing that works.

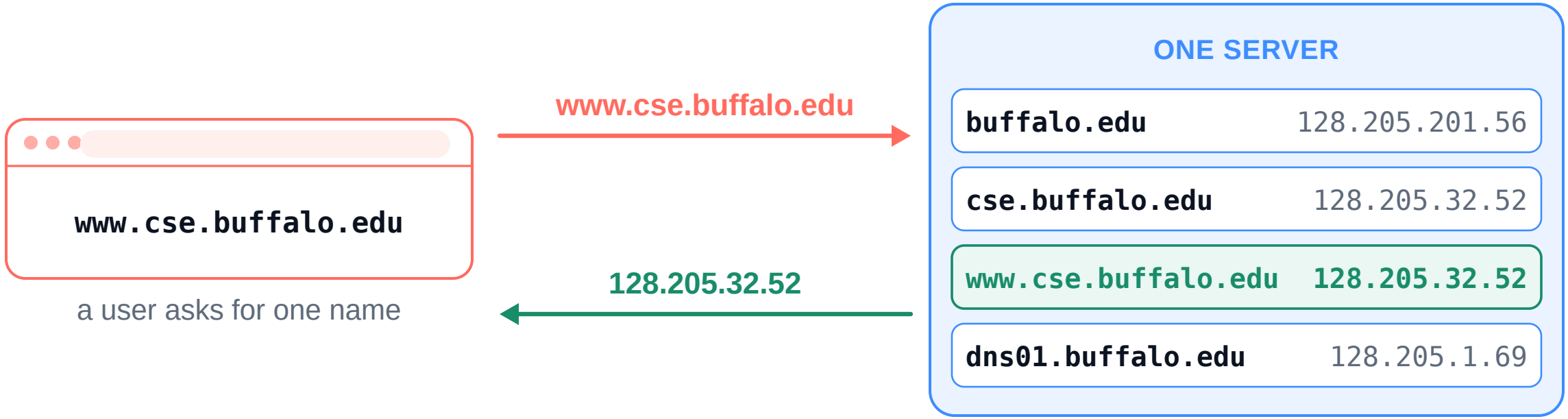
What does a name service actually have to do?

02

One server, one file

Build the smallest design that works, with nothing in it that is not needed yet. One server, one set of names, one kind of record.

One server can answer for every name it is given



This is the whole exchange: the user gives a name, the server returns the address that goes with it. This really was the design of Internet naming until 1983.

Everything that follows is added to this, one problem at a time.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

A query names one thing and asks one question

A = what is the IPv4 address of this name?

One name plus one question. That is the entire request.

The answer is one resource record

NAME	TYPE	VALUE	TTL
www.cse.buffalo.edu	A	128.205.32.52	21600

NAME	the name this record is about
TYPE	which question it answers
VALUE	the answer itself
TTL	how long you may keep it — Part 5

A record is the unit DNS stores and the unit it returns.

The pairs live in one file on that server

```
; every name this server answers for
buffalo.edu.              IN  A  128.205.201.56
cse.buffalo.edu.          IN  A  128.205.32.52
www.cse.buffalo.edu.      IN  A  128.205.32.52
dns01.buffalo.edu.        IN  A  128.205.1.69
```

One record per line, one line per name. The server answers for these four names and for nothing else — a query for any other name gets no answer from it.

The server’s knowledge is exactly the contents of this file.

One datagram each way, over UDP, to port 53

WHY UDP AND NOT TCP

opening a TCP connection would cost more round trips
than the two-packet exchange it would carry

No connection, and no state kept on either side.

The client cannot tell why no answer came back

THREE THINGS THAT LOOK IDENTICAL TO THE CLIENT

the query was lost — **the reply was lost** — **the server is slow**

and it never finds out which one it was

the client starts a timer when it sends

if nothing arrives before it expires, it sends the query again

A timer and a retransmission. That is the whole recovery mechanism.

Evidence

dig shows the question and the record that answers it

```
$ dig www.cse.buffalo.edu A +noall +answer +stats
www.cse.buffalo.edu.  21600  IN  A  128.205.32.52
;; Query time: 31 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; MSG SIZE  rcvd: 93
```

The name, the TTL, the type and the value — the four fields from two slides ago, in that order. 31 ms, 93 bytes, two packets.

You can now explain every byte of a complete name lookup.

This is a complete name service. It has one problem.

WHAT WE HAVE

a query: a name and a type

an answer: a record

a file: the names one server holds

a transport: UDP, port 53

WHAT IS WRONG WITH IT

one server

one file

one administrator

for every name on the Internet

Part 3 is about splitting that one server into many, and who is allowed to.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

Why can one server not hold all of it?

03

The tree

Three of the four problems are about performance and are fixed by copying. The fourth is about who is allowed to write, and it is the one that decided the shape of DNS.

The Internet now has 401 million registered domains

1983
~300 hosts
one printed page

401,600,000

registered domain names · Q2 2026

each one holds as many names below it as its owner defines

no file, and no machine, is going to hold this

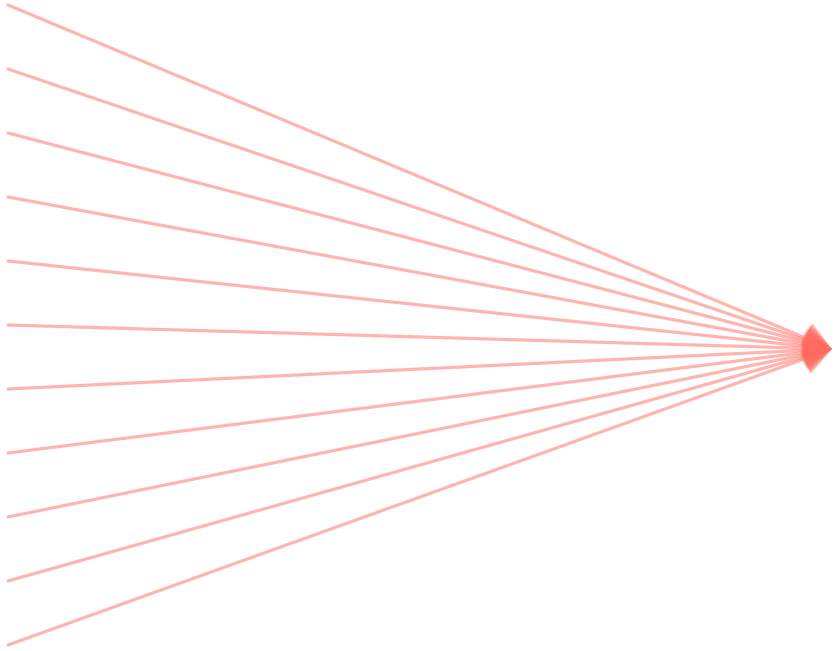
The design from Part 2 is adequate at 300 rows. At four hundred million it is not, and it fails in four separate ways.

Four problems. The fourth is the decisive one.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

Problem 1

One server would receive every lookup on the Internet



one server
answering all of it

every device, every application, every page load

and it must answer

before any other connection can be made

a lookup happens before every connection, not occasionally

Query load. Replication solves it.

Problem 2

One server is a single point of failure

no other component has failed

every link, every router and every server is still running

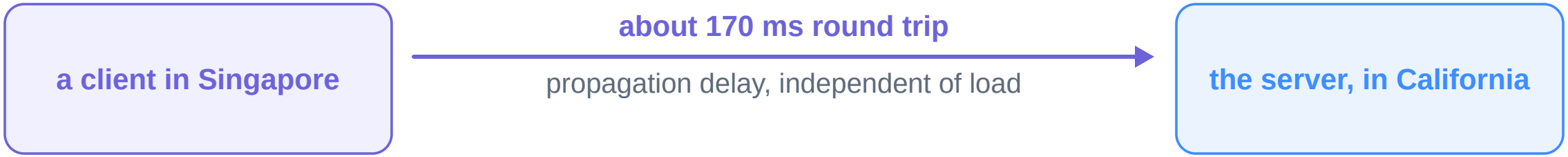
and no client can reach any service

This is a different problem from the previous one. Additional capacity does not help if all of it is in one location.

Availability. Geographically separated replicas solve it.

Problem 3

A distant server adds latency to every lookup



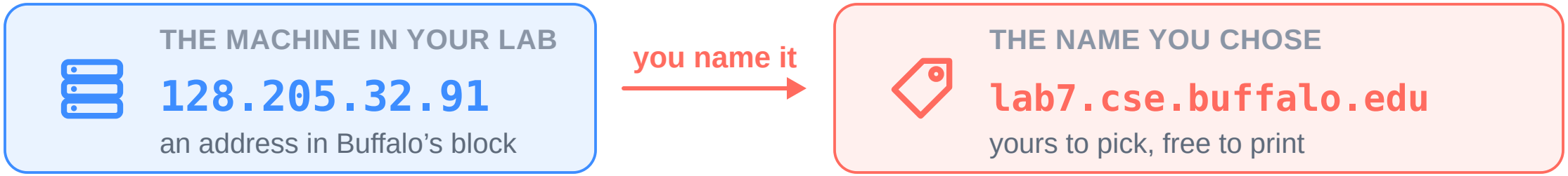
the application request begins only after this

no page content has been requested yet

Latency. Replicas placed near clients solve it.

Problem 4

You want to publish lab7.cse.buffalo.edu



someone you told types it

how does their machine learn 128.205.32.91?

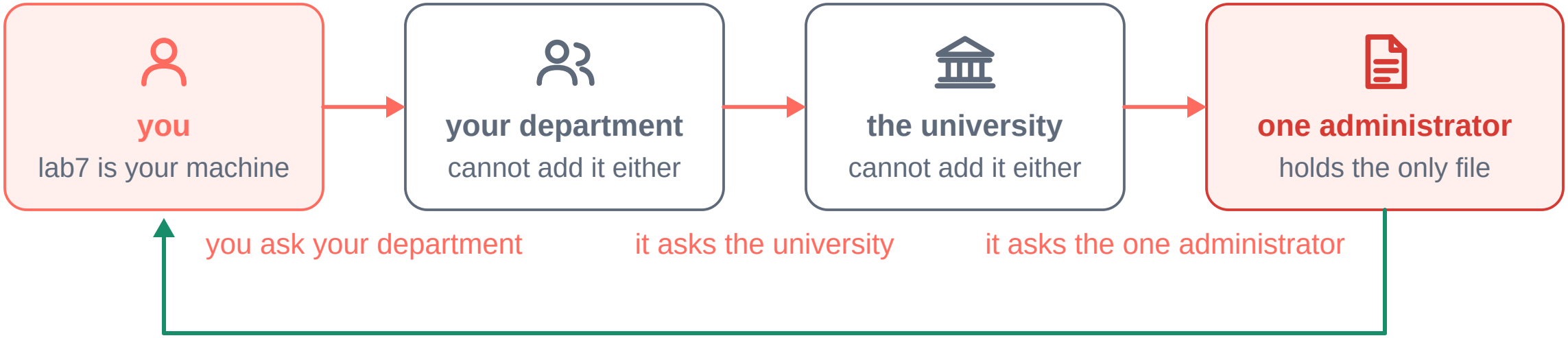
only if that line is in the file

A name is free to choose and useless until someone writes it down.

Problem 4

Adding one name means asking all the way up

`add lab7.cse.buffalo.edu → 128.205.32.91`



the line is written, and only then does the name resolve

every name on the Internet has to pass through that last box

Nobody in the chain may add the name. Only the last box may.

Problem 4

How many of these requests arrive every day

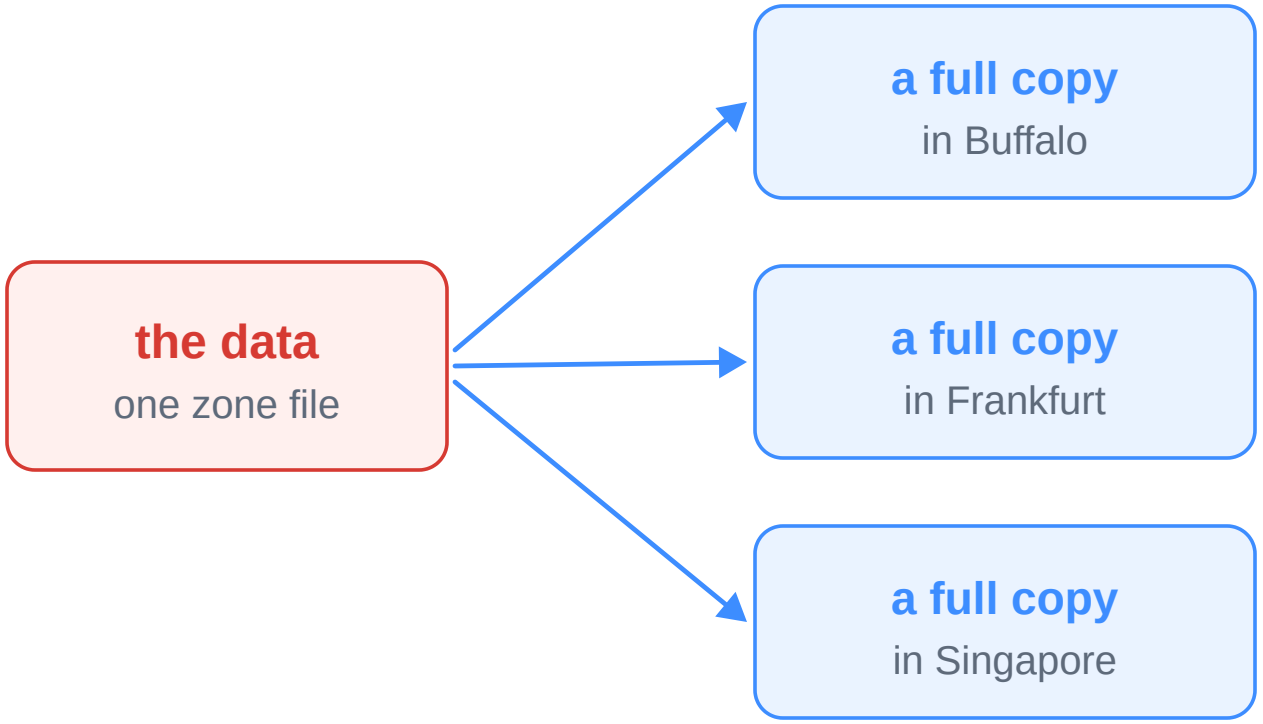
and this counts only .com and .net — 179 of the 401 million domains

a name like lab7 inside buffalo.edu is not even counted here

Verisign reported 12.7 million new .com and .net registrations during the second quarter of 2026. The per-day and per-second figures are that total divided by the length of the quarter.

Administrative control. Replication cannot solve this one.

Copies of the data solve three of the four

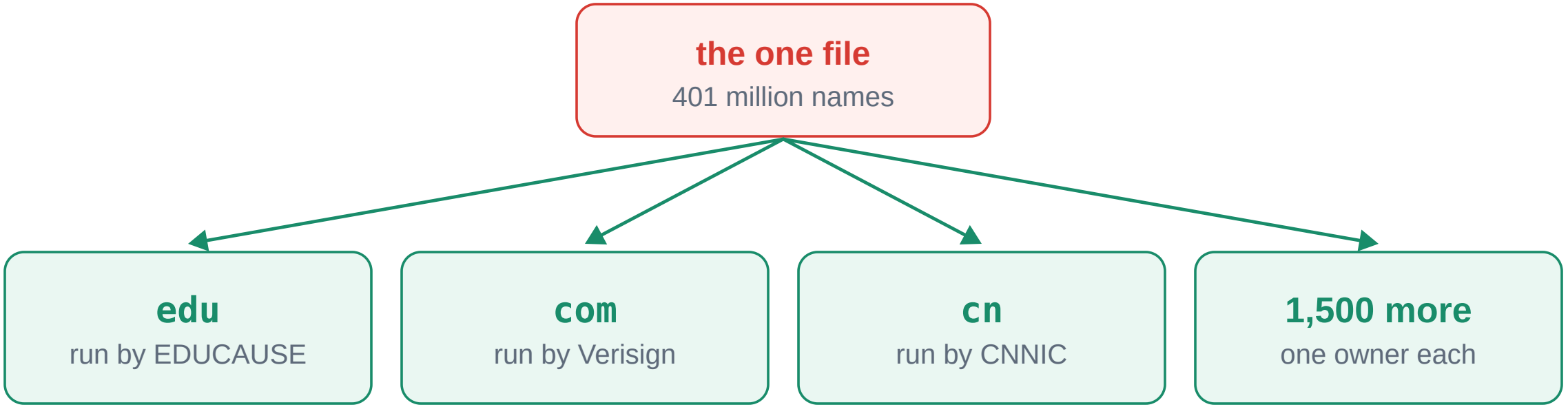


WHAT IT SOLVES

- ✓ problem 1 · query load
- ✓ problem 2 · availability
- ✓ problem 3 · latency
- ✗ problem 4 · who may edit

Three problems gone. Every copy is still edited by the same one owner.

Splitting the data is what solves the fourth

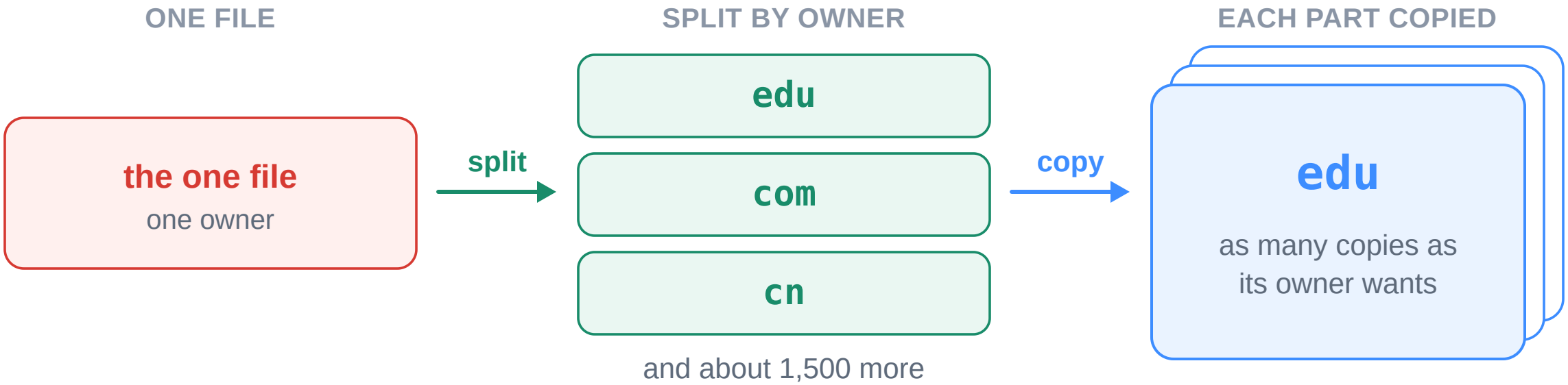


each part has one owner, and nobody waits for the others

✓ **problem 4 is solved — authority is what got split**

But each part is now a single copy again, so problems 1 to 3 come back.

DNS splits first, then copies each part

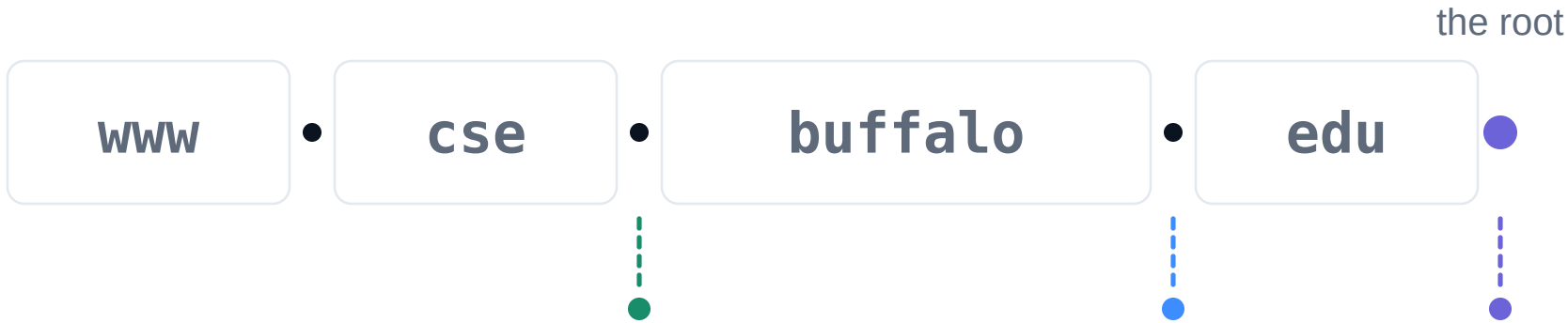


IN THE OTHER ORDER

copy the whole file first, and every copy still has one owner

Partition solves problem 4. Replication then solves 1 to 3 inside each part.

The dots in the name are where it can be split



the root hands edu to a registry

edu hands buffalo.edu to the university

buffalo.edu hands cse.buffalo.edu to the department

every dot is a place where authority may change hands

Nothing new had to be invented. The boundaries were already in the name.

The namespace is a tree, and every level has an owner

	LEVEL	EXAMPLE	WHO RUNS IT
0	the root	`.`	ICANN, through IANA
1	top-level domain	`edu`	a registry, one per TLD
2	registered domain	`buffalo.edu`	whoever registered it
3	everything below	`www.cse.buffalo.edu`	the owner of that domain

Each level only knows how to reach the level below it.

How many names there are at each level

the root	1	one zone, one file
top-level domains	1,500	approximately — the root zone lists them all
registered domains	401,600,000	Q2 2026, across every TLD
names below those	?	nobody counts them, and nobody needs to

The top of the tree is tiny. Almost all of the data is at level 2 and below.

Top-level domains come in two main kinds

COUNTRY-CODE · ccTLD

two letters, one per country

`cn` `de` `uk` `nl`

about 250 in Latin letters,
and about 60 more in other scripts

GENERIC · gTLD

anyone may apply for one

`com` `org` `xyz` `app`

seven of them until 2013,
about 1,200 of them now

and a dozen special ones with an eligibility rule

edu · gov · mil · int · museum · coop and arpa, used only by the protocol itself

The kind of TLD decides who may register under it, not how it is resolved.

Some country codes are used for what the letters spell

.tv Tuvalu **video and streaming**	**.ai** Anguilla **artificial intelligence**	**.io** British Indian Ocean Terr. **software and startups**
.co Colombia **company**	**.me** Montenegro **personal sites**	**.fm** Micronesia **radio and podcasts**
.ly Libya **word endings, bit.ly**	**.gg** Guernsey **gaming**	**.sh** Saint Helena **shell, command line**

nothing in the protocol knows what the letters mean

What differs is the registry’s rules, not the resolution.

What a two-letter code is worth, and what it depends on

.ai — Anguilla

1,277,727 names registered by June 2026

$32 million in fees in 2023, over 10% of GDP

.tv — Tuvalu

GoDaddy pays the government $10 million a year

8.4% of all government revenue in 2019

.su — the Soviet Union

the country dissolved in December 1991

112,000 names still registered in 2026

.io — British Indian Ocean Territory

sovereignty transfer signed in May 2025

losing the ISO code starts a 5-year retirement

every one of these names works because a government’s delegation still exists

A delegation is a political object as much as a technical one.

The ten largest top-level domains

1	2	3	4	5	6	7	8	9	10
com	cn	de	net	org	uk	xyz	ru	top	nl

.com alone

166,600,000

more than a third of every domain on Earth

all country codes together

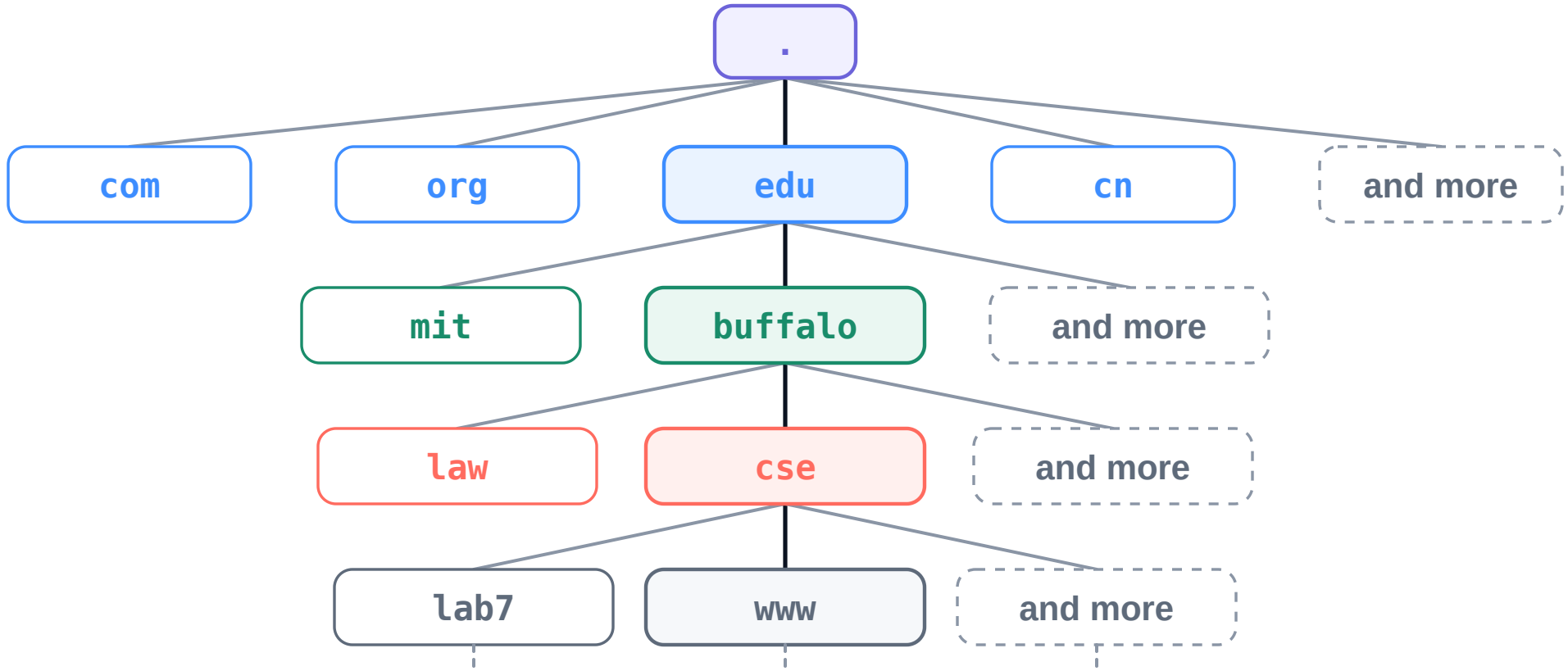
148,600,000

spread over about 250 registries

Five of the ten are country codes and five are generic. The two kinds are not separate systems; they are neighbours in the same root zone.

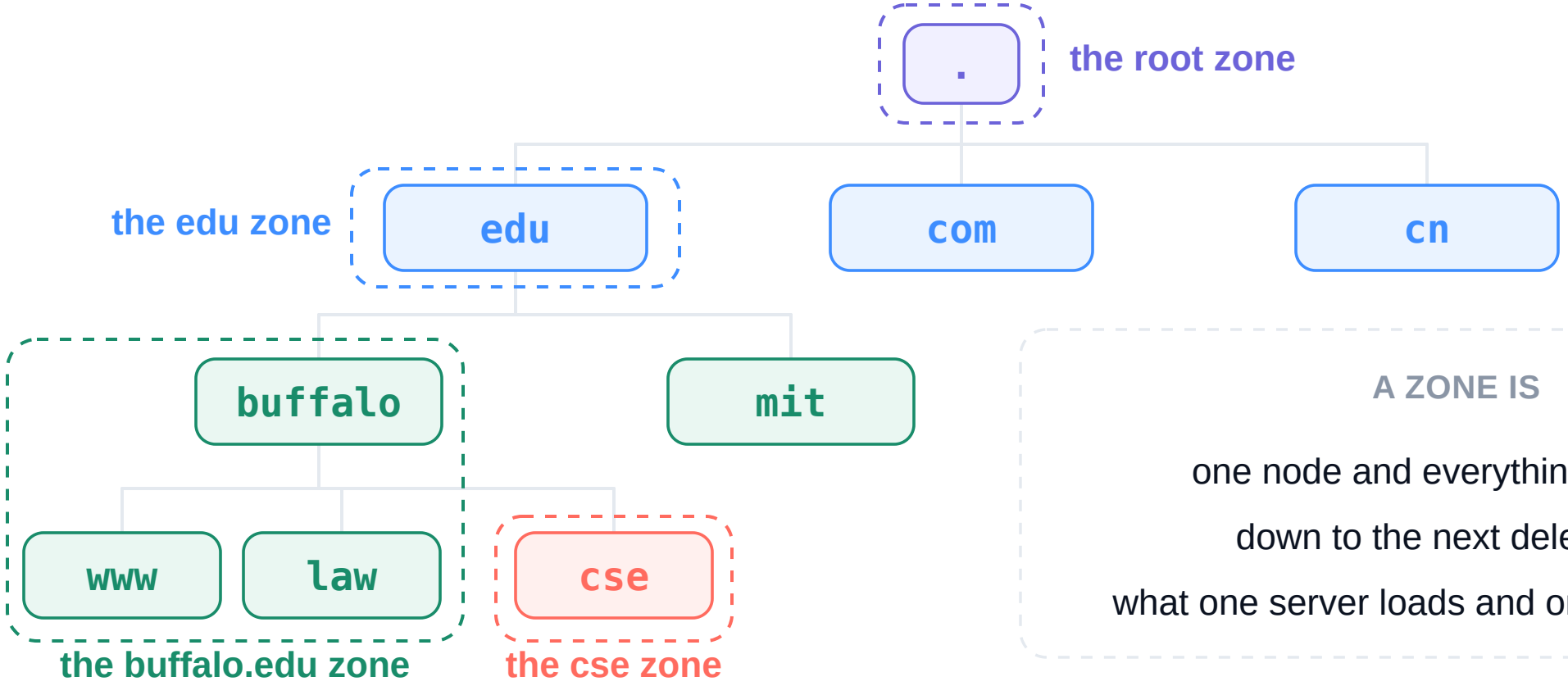
Size is very uneven, and the design has to work at both ends of it.

One tree, from the root down to a single host



Only the top three levels are counted. Below them, nobody is keeping track.

A zone is the part of the tree one server answers for



A ZONE IS

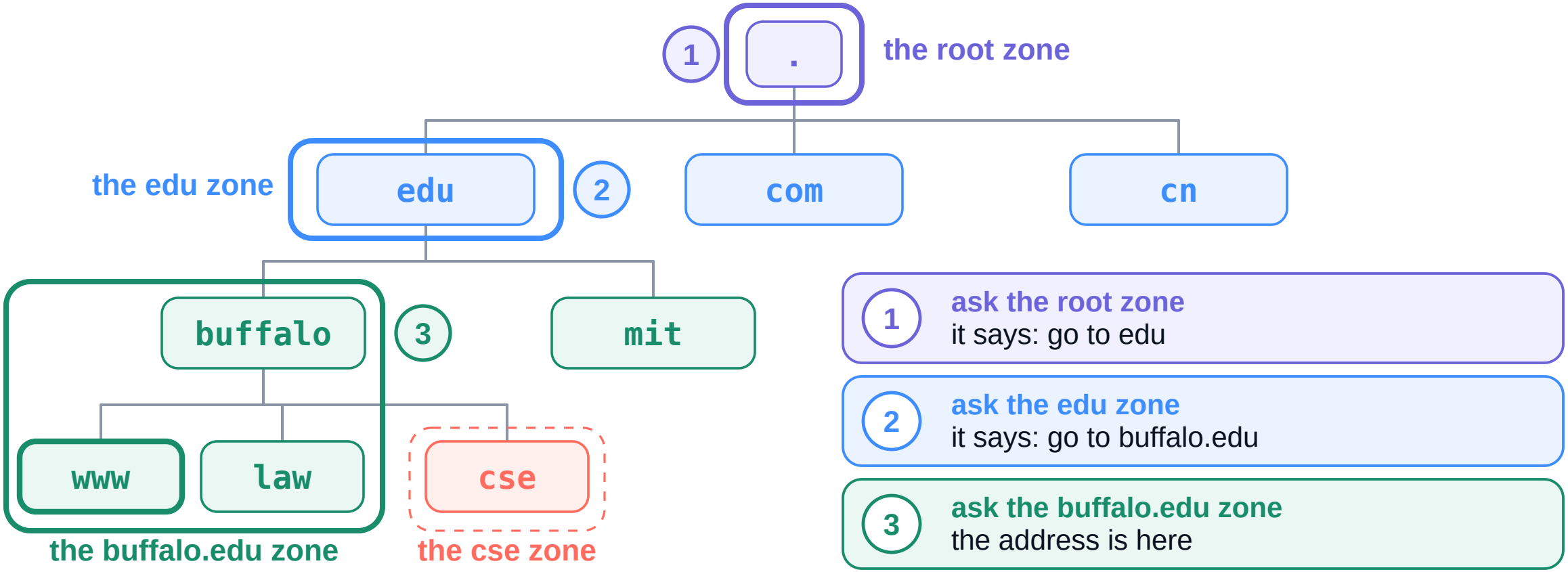
one node and everything below it

down to the next delegation

what one server loads and one owner edits

The single file in Part 2 was one zone. The tree is 400 million of them.

How the query moves from one zone to the next



Each zone knows its own names, and the name of who to ask next.

The root has no record for www.buffalo.edu

the root server

its zone lists only top-level names:

`com` `org` `edu` `cn` + 1,500 more

you ask the root for **`www.buffalo.edu`** — **no such record**

yet on the last page it said **go to edu**

so how can the root name a server for a zone it does not hold?

Nothing in the root zone mentions buffalo. Something still points there.

The root zone holds one record about edu

the root zone contains this one line about edu

edu.	172800	IN	NS	a.edu-servers.net.
the zone	cache time	class	type	the server to ask

NS = name server

the root is handing this zone
to somebody else

the name of the server
that holds it

READ IT AS

for any name ending in edu, ask a.edu-servers.net

✓ **that is how the root could say: go to edu**

A parent zone stores a pointer, and nothing behind it.

How do you reach a.edu-servers.net?

the root told you to ask

`a.edu-servers.net`

but that is a name

a packet needs an address

you cannot send anything to a name

SO YOU WOULD HAVE TO LOOK IT UP

`ask a.edu-servers.net` → **what is its address?** ←

and round again, forever

You cannot follow a referral you have no address for.

The reply carries two records, not one

The root does not send one record. It sends two.

NS

`edu. NS a.edu-servers.net.`

the name of the server to ask

a name

A

`a.edu-servers.net. A 192.5.6.30`

and its address

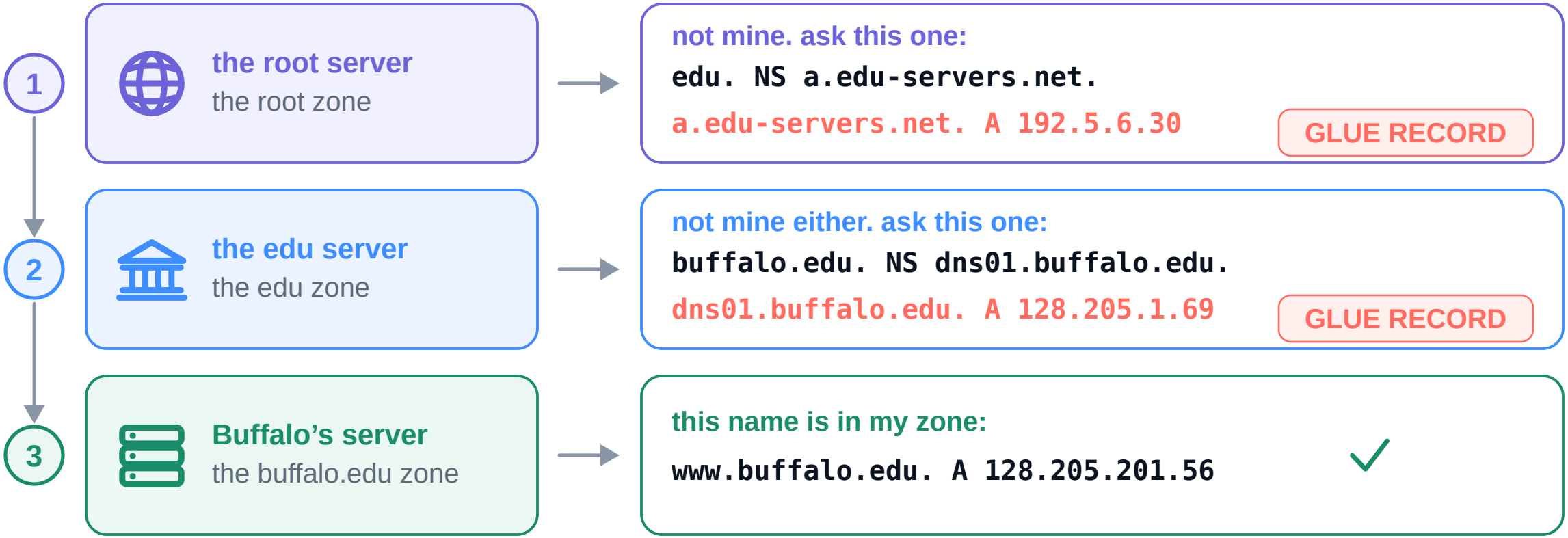
an address

so you already have the IP — no second lookup

that second record is called glue

Glue is the address a parent publishes for a server it does not own.

The same walk, with the records the servers send



Every referral is two records: the name to ask, and its address.

The resolver is the program that does the walking

WHAT IT IS

a program on a server
run by your ISP or campus
or a public one: 8.8.8.8
it holds no zone of its own

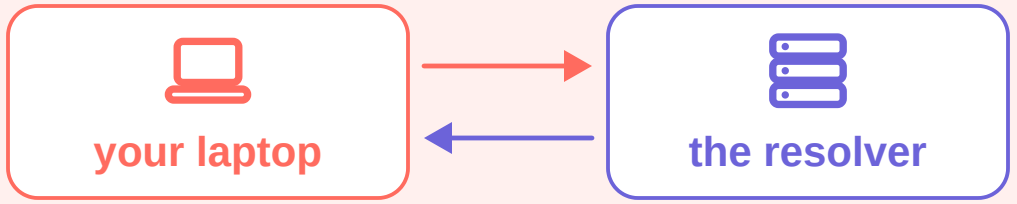
WHAT IT DOES

takes your one question
walks the tree for you
sends back one answer
and remembers it for next time

Your laptop asks it one question and waits. It does everything else.

One recursive query becomes three iterative ones

RECURSIVE · WHAT YOUR LAPTOP DOES



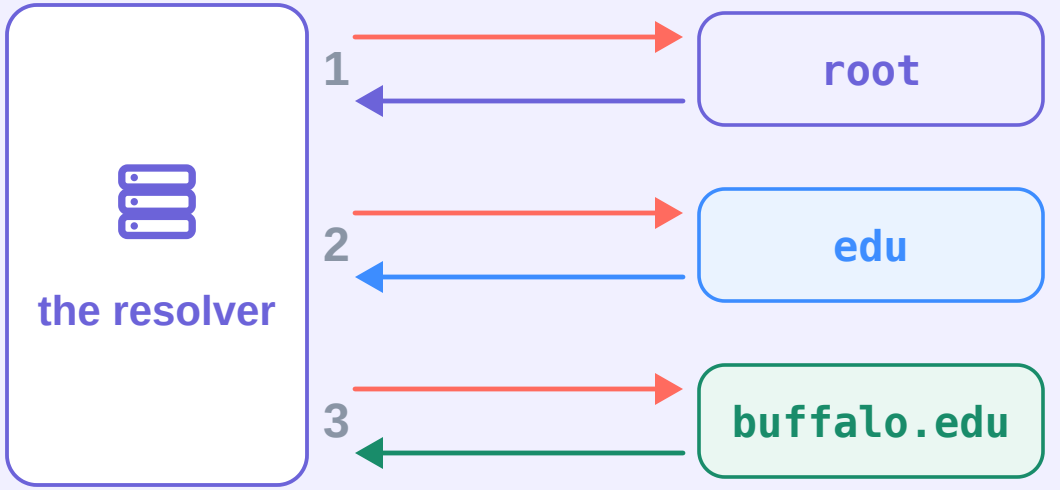
RD=1 · resolve this for me

`128.205.201.56`

one query, one answer

the laptop never talks to root,
edu or buffalo.edu

ITERATIVE · WHAT THE RESOLVER DOES



RD=0 · three queries, two referrals

Recursive: ask once, get the answer. Iterative: ask, get pointed onward.

13 names, 12 operators, 2,045 machines

a b c d e f g h i j k l m

a.root-servers.net through m.root-servers.net

13
names
one letter each, a to m

12
operators
no single organisation

2,045
machines
all serving one zone

one IP address per name, announced from every site at once
you reach whichever machine is nearest

This is the replication half of the fix, applied to the busiest zone.

What the tree achieves

no server holds everything

the root zone is about 2 MB

no server is asked everything

only about the level below

each owner edits its own zone

with no central approval

problem 4 is solved: administrative control is what got split

Part 4: none of this yet explains how one client gets one answer.

The tree is built. How is one name resolved?

04

The lookup

Four components take part, and your program talks to only one of them. Every field of the DNS message you have not met yet appears in this part, at the moment it first matters.

Four components take part in a lookup

your program
calls getaddrinfo()

→

the stub resolver
a library in your OS

→

the recursive resolver
your ISP, or 8.8.8.8

→

root, TLD, authoritative
the tree from Part 3

on the client machine, no packets sent yet

remote servers, where the lookup happens

The recursive resolver does the work. Your program does not.

The application makes one call and waits

```
sockaddr = getaddrinfo("www.cse.buffalo.edu", 80)

# ... 31 ms pass ...

s.connect(("128.205.32.52", 80))
```

one library call

one query out,
one answer back

no retries in your code

one bit in the query asks for all of this: RD = 1

recursion desired: do the lookup and come back with an answer

The application sends one query and receives one answer. Everything else in this part happens because one bit asked somebody else to do it.

RD=1 is a request for work, not for data.

A resolver starts with only the root server addresses

a fresh resolver
empty cache

except →

named.root, the root hints file

```
a.root-servers.net. 198.41.0.4
b.root-servers.net. 170.247.170.2
c.root-servers.net. 192.33.4.12
…
```

→ **everything else**
learned from NS records

these thirteen addresses are the only data a resolver is configured with

Every other server address is learned on the way.

Step 1

The root cannot answer, so it refers you instead

TWO SECTIONS YOU HAVE NOT SEEN BEFORE

```
AUTHORITY: edu. 172800 IN NS a.edu-servers.net.
ADDITIONAL: a.edu-servers.net. IN A 192.5.6.30
```

The answer section is empty. The root zone holds no record for buffalo or cse. It returns the NS records for edu, and their addresses.

A referral moves the lookup one level down the tree.

Step 2

The .edu server refers you to buffalo.edu

THE SAME SHAPE, ONE LEVEL DOWN

```
AUTHORITY: buffalo.edu. 172800 IN NS dns01.buffalo.edu.
ADDITIONAL: dns01.buffalo.edu. IN A 128.205.1.69
```

This server holds the edu zone: about 8,000 delegations and almost nothing else. It knows which servers speak for buffalo.edu, and none of that zone's data.

Every level applies the same rule: answer, or refer.

Step 3

The authoritative server owns the name and answers

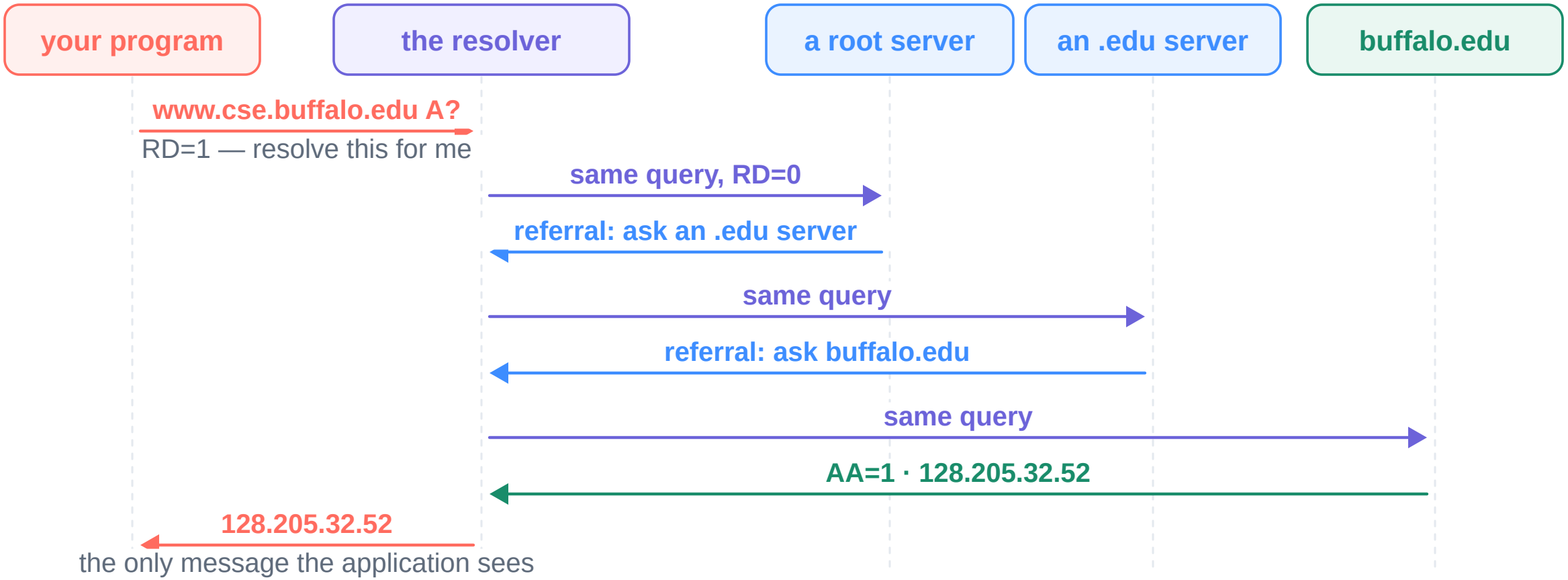
ANSWER SECTION

```
www.cse.buffalo.edu. 21600 IN CNAME jeeves.cse.buffalo.edu.
jeeves.cse.buffalo.edu. 21600 IN A 128.205.32.52
```

AA=1 is the bit that says so. Buffalo serves cse.buffalo.edu from the same machines, so there was no fourth hop. Ignore the CNAME line for now — Part 6 explains it.

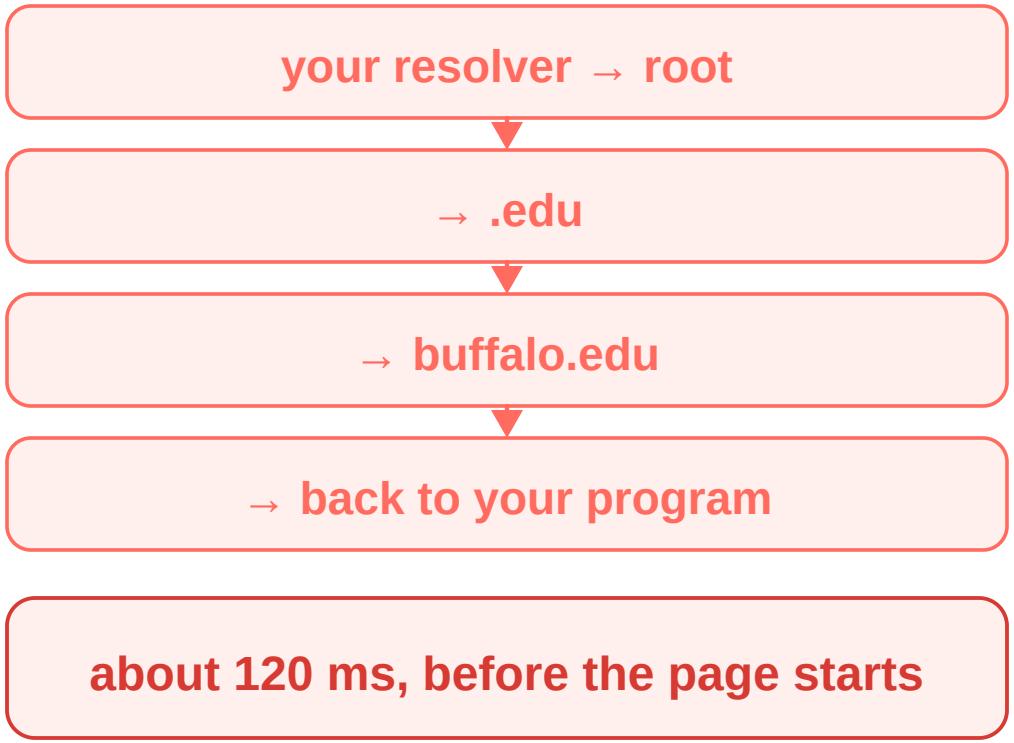
The number of queries depends on delegations, not on labels.

The complete lookup takes eight messages



Your program was in two of them. The other six are the resolver walking.

A full lookup costs four round trips



Nobody could afford this per page. Part 5 is why almost nobody pays it.

Four round trips per name. Why is the web not slow?

05

Caching

The resolver keeps what it learned, and every record carries an expiry set by whoever owns it. That one field decides how fast DNS is and how fast it can change.

The resolver keeps every record it learns

the five records the resolver learned on that one walk

`. → a.root-servers.net`	518400	6 days
`edu. → a.edu-servers.net`	172800	2 days
`buffalo.edu. → dns01.buffalo.edu`	172800	2 days
`www.cse.buffalo.edu → jeeves…`	21600	6 hours
`jeeves.cse.buffalo.edu → 128.205.32.52`	21600	6 hours

the TTL field from Part 2 is what makes this possible

The TTL is set by the owner of the zone the record came from.

A cached lookup does not touch the tree at all

4 round trips · about 120 ms

1 round trip · about 5 ms

The referrals are cached too, not only the final answer. One walk pays for every later name under buffalo.edu, and for every later name under edu.

The tree provides correctness; the cache provides speed.

The TTL trades update speed against query load

SHORT TTL

a change takes effect within minutes, and almost every visit causes a full lookup

chosen
per record

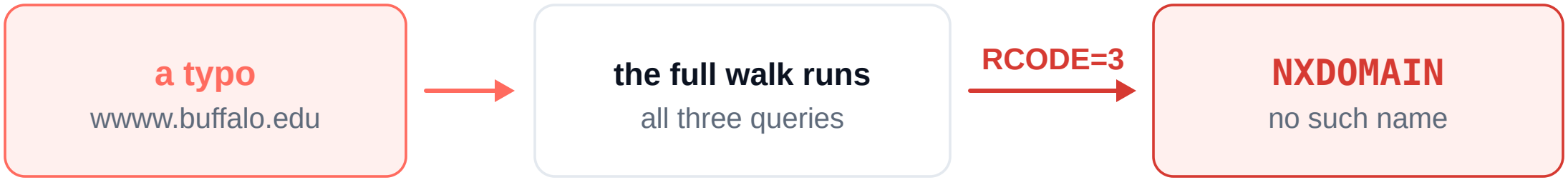
LONG TTL

almost every visit is served from cache, and a change can take days to take effect

Before moving a service, operators lower the TTL to 60 seconds a day in advance, make the change, and then raise it again. The extra query load is only paid during that window.

The TTL is the only control over how quickly a change takes effect.

A name that does not exist is an answer worth keeping



the negative answer is cached too

for 900 seconds, the last field of Buffalo’s SOA record

repeats of the same query are answered from cache

SOA is a per-zone record holding its operational parameters. Only its last field matters here: how long a "no such name" answer may be cached.

A name you just created can stay unreachable for exactly that long.

A change takes effect when cached records expire

A COMMON DESCRIPTION

"DNS is propagating"

as if the change were pushed out

WHAT ACTUALLY HAPPENS

cached records expire

nothing is pushed anywhere

this interval is exactly the old TTL

so the TTL is lowered before the change, not after

Lower the TTL one old-TTL period before making a change.

The root is asked only for what nobody has cached

in normal operation, most queries never leave the resolver

and of the ones that do, almost none reach the root

the root
asked once per TLD, per cache

→

the TLD servers
asked once per domain

→

the resolver
asked for everything else

this is why 13 names and 2,045 machines are enough for the whole Internet

Part 6: everything so far has been reading. Now you write.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

How does the world learn to find your server?

06

Publishing a name

Everything so far has been reading. Publishing is a write, and part of that write happens in a zone you do not control. The record types you have not met yet appear here, one per requirement.

Three things must be true before anyone can find you

1

the parent delegates

an NS record for mysite.com

+

2

servers answer for it

at least two machines

+

3

the zone has the record

an A record with

✕ **if any one is missing, the walk stops before it reaches your server**

Two of the three are somebody else’s records, not yours.

Step 1

Three parties, and only one may edit the zone

registry

one per TLD

the only party that may edit that TLD’s zone

Verisign runs .com, EDUCAUSE runs .edu, CNNIC runs .cn

registrar

a reseller

accredited by the registry to submit changes for you

GoDaddy, Namecheap, Cloudflare — you never talk to the registry

registrant

that is you

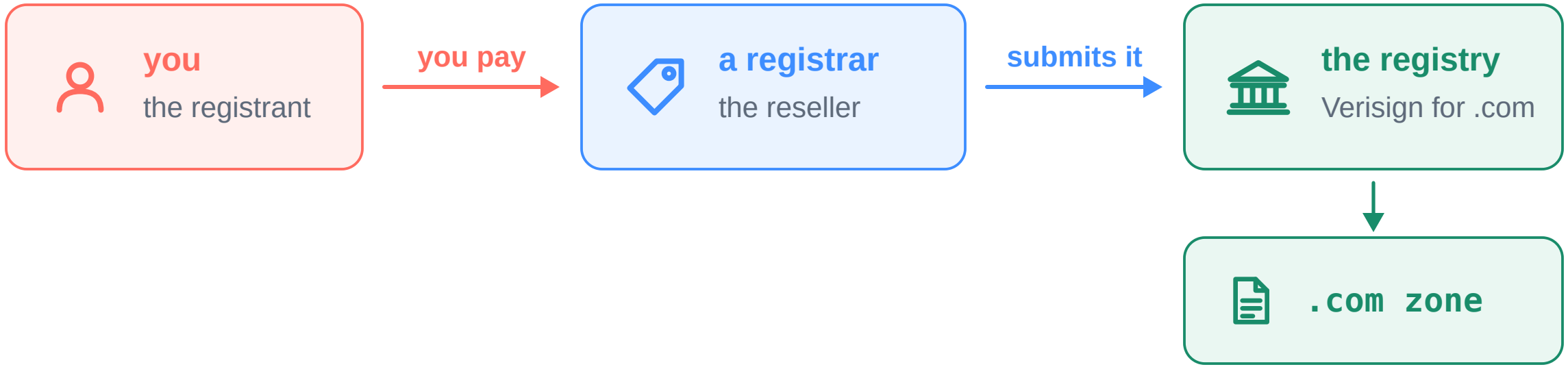
the party the name is registered to

you hold it for a term and renew it; you do not own it

You are a customer of a reseller of the one organisation that runs .com.

Step 1

What you buy, and who actually writes it down



a registration does not buy the name itself

it buys one NS line in the parent zone, renewed every year

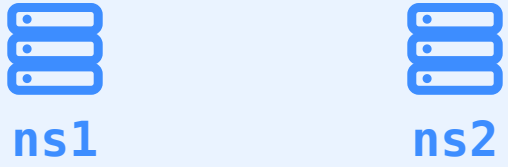
One organisation runs .com. Everyone else is a customer of a customer.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

Step 2

You need at least two authoritative name servers

RUN THEM YOURSELF



two machines, separate power
and separate networks

USE A DNS HOST

Cloudflare, Route 53, and others

their anycast servers,
your zone file

two is the minimum because one server is a single point of failure

that is problem 2 from Part 3

Registering a domain and serving its zone are separate jobs.

Step 3

The registry adds the NS records to the .com zone

you give the registrar two server names; the registry writes these records

THE .COM ZONE, EDITED BY THE REGISTRY

```
mysite.com. 172800 IN NS ns1.dnshost.net.
mysite.com. 172800 IN NS ns2.dnshost.net.
```

no glue needed here

the server names are not in your zone

servers inside mysite.com

would need glue: their addresses too

The domain becomes resolvable the moment this delegation exists.

Step 4

Your zone file starts with SOA, NS and one address

```
$TTL 3600
@     IN SOA  ns1.dnshost.net. you.example.com. (
                2026091101 7200 3600 1209600 300 )
@     IN NS   ns1.dnshost.net.
@     IN NS   ns2.dnshost.net.
@     IN A    203.0.113.10
@     IN AAAA 2001:db8::10
```

AAAA is the same record for IPv6 · **SOA** is zone housekeeping, and only its last field leaves the zone

300 is the negative-cache TTL from Part 5. Keep it low while you are still making mistakes.

Three of the five record types in this lecture are already here.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

CNAME points one name at another name

CNAME **canonical name** — its value is another name, not an address

you also want www.mysite.com to work, and the address may change

TWO ADDRESS RECORDS

```
@ IN A 203.0.113.10
www IN A 203.0.113.10
```

✗ move the server, edit both

ONE ADDRESS, ONE ALIAS

```
@ IN A 203.0.113.10
www IN CNAME mysite.com.
```

✓ move the server, edit one

A CNAME lets one name track another without copying its address.

MX says where mail for the domain is delivered

MX **mail exchanger** — the server that accepts mail for the domain

`mysite.com A?` → `203.0.113.10`

`mysite.com MX?` → `10 mx.mailhost.net.`

✓ **same name, two questions, two answers — the TYPE field is what separates them**

One name can carry a web server and a mail server at the same time.

A real zone

What buffalo.edu actually publishes today

```
; what buffalo.edu really publishes
buffalo.edu.       IN NS    dns01.buffalo.edu.  ; and dns02-04
buffalo.edu.       IN A     128.205.201.56
buffalo.edu.       IN MX    10 buffalo-edu.mail.protection.outlook.com.
www.buffalo.edu.   IN CNAME www.buffalo.edu.edgekey.net.
dns01.buffalo.edu. IN A     128.205.1.69
```

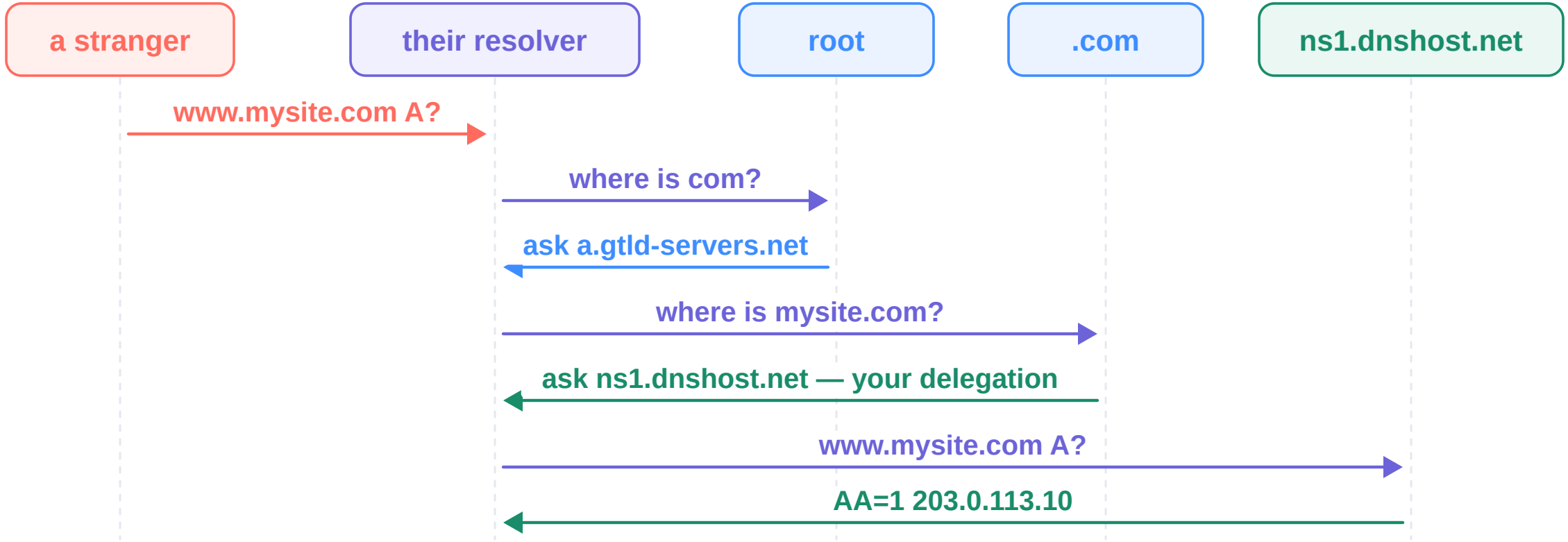
Two of these are worth reading twice. The MX sends every message for @buffalo.edu to Microsoft, and the CNAME hands the home page to Akamai — without changing a single name anybody types.

Four record types, one file, and the whole university reachable from it.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

Step 5

A client’s first lookup of your new name



You did not add a step. You supplied the last one that was missing.

1 Why names | 2 One server | 3 The tree | 4 The lookup | 5 Caching | 6 Publishing

Verifying the delegation and the zone separately

CHECK THE DELEGATION

```
dig +trace www.mysite.com
```

runs the walk hop by hop, bypassing all caches

QUERY YOUR OWN SERVER

```
dig @ns1.dnshost.net www.mysite.com
```

if this fails, the zone is wrong, not the delegation

WHY A NEW NAME CAN SEEM SLOW

you queried the name before it existed

↓

your resolver cached NXDOMAIN

↓

for the parent zone’s negative-cache TTL

a cache entry expiring, not a failure

Test a new domain from a resolver that has not cached the earlier failure.

The complete DNS message, field by field

THE 12-BYTE HEADER

`ID`	matches a reply to its query
`RD`	do the whole lookup for me
`AA`	this answer is authoritative
`TC`	the answer did not fit
`RCODE`	0 = ok, 3 = no such name
`counts`	how many records per section

THE FOUR SECTIONS

`QUESTION`	the name and the type
`ANSWER`	the records that answer it
`AUTHORITY`	who to ask instead
`ADDITIONAL`	the glue for those servers

Not one of these fields was introduced before it had a job.

1 Why names · 2 One server · 3 The tree · 4 The lookup · 5 Caching · 6 Publishing

Four questions to ask of any naming system

1	**Who owns this name?**	which zone is it in, and who edits that zone
2	**Which servers are authoritative?**	which servers answer with AA=1
3	**Who is allowed to delegate it?**	what the parent zone says about it
4	**How long may an answer be cached?**	the TTL, and who set it

Certificate authorities, package registries and service discovery inside a cluster all use the same structure: hierarchical names, delegated authority, and cached mappings with an expiry.

Next: what happens once the address is known.

1 Why names · 2 One server · 3 The tree · 4 The lookup · 5 Caching · 6 Publishing