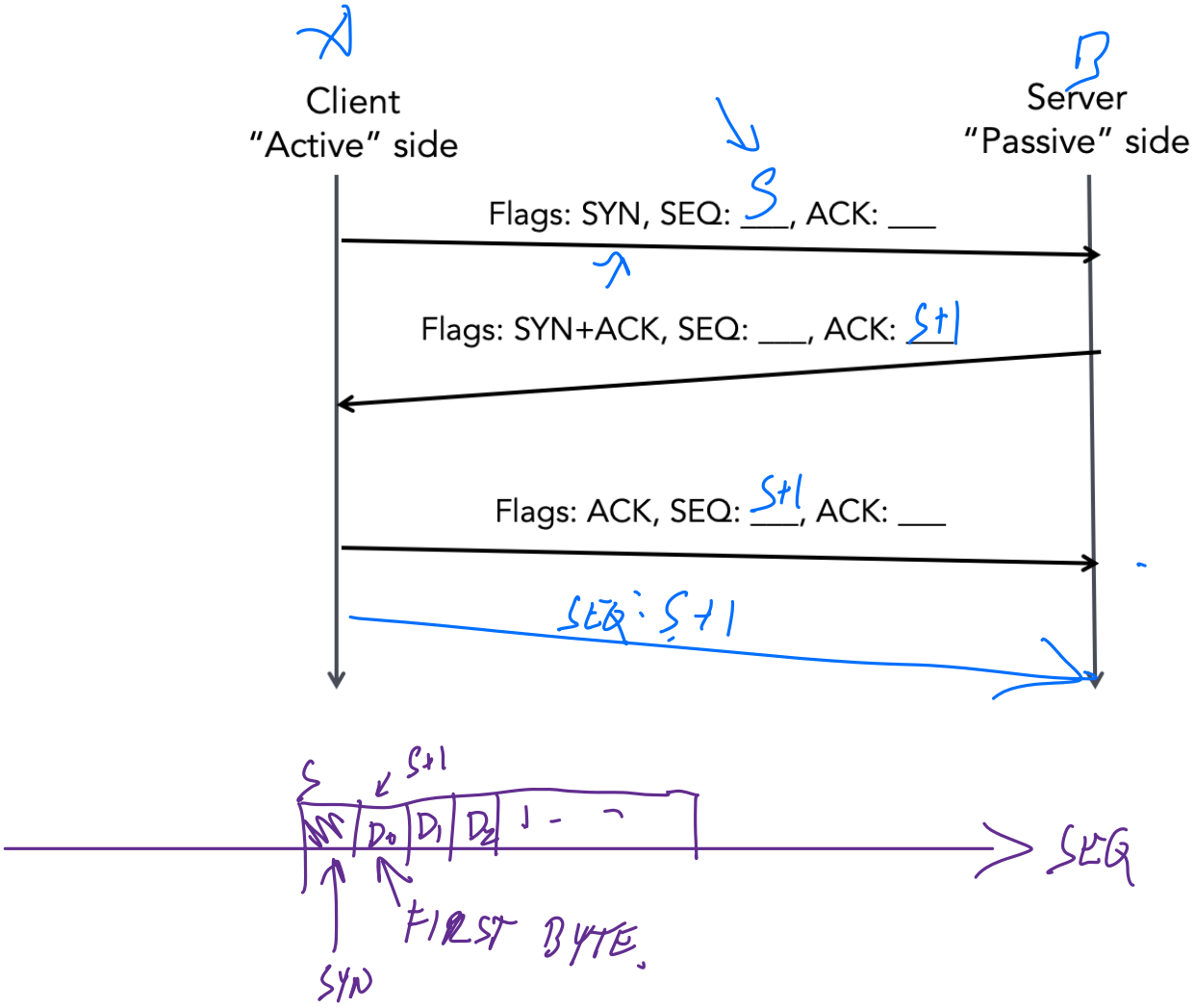
Today
- TCP teardown
- DNS

This week
- TCP milestone 1: should do meeting by today
- TCP gearup II: Wednesday, April 1, 6pm CIT 368
- HW3 (short!): Due Friday
- TCP SRC component (short): due Friday (just push to your repo)
    - (soft deadline)

- Milestone II meetings: Mon-Wed next week

Lecture 17: TCP Teardown, DNS I

Quick Warmup: If a connection for host A starts with sequence number S, what is the sequence number for the first data segment sent from A?

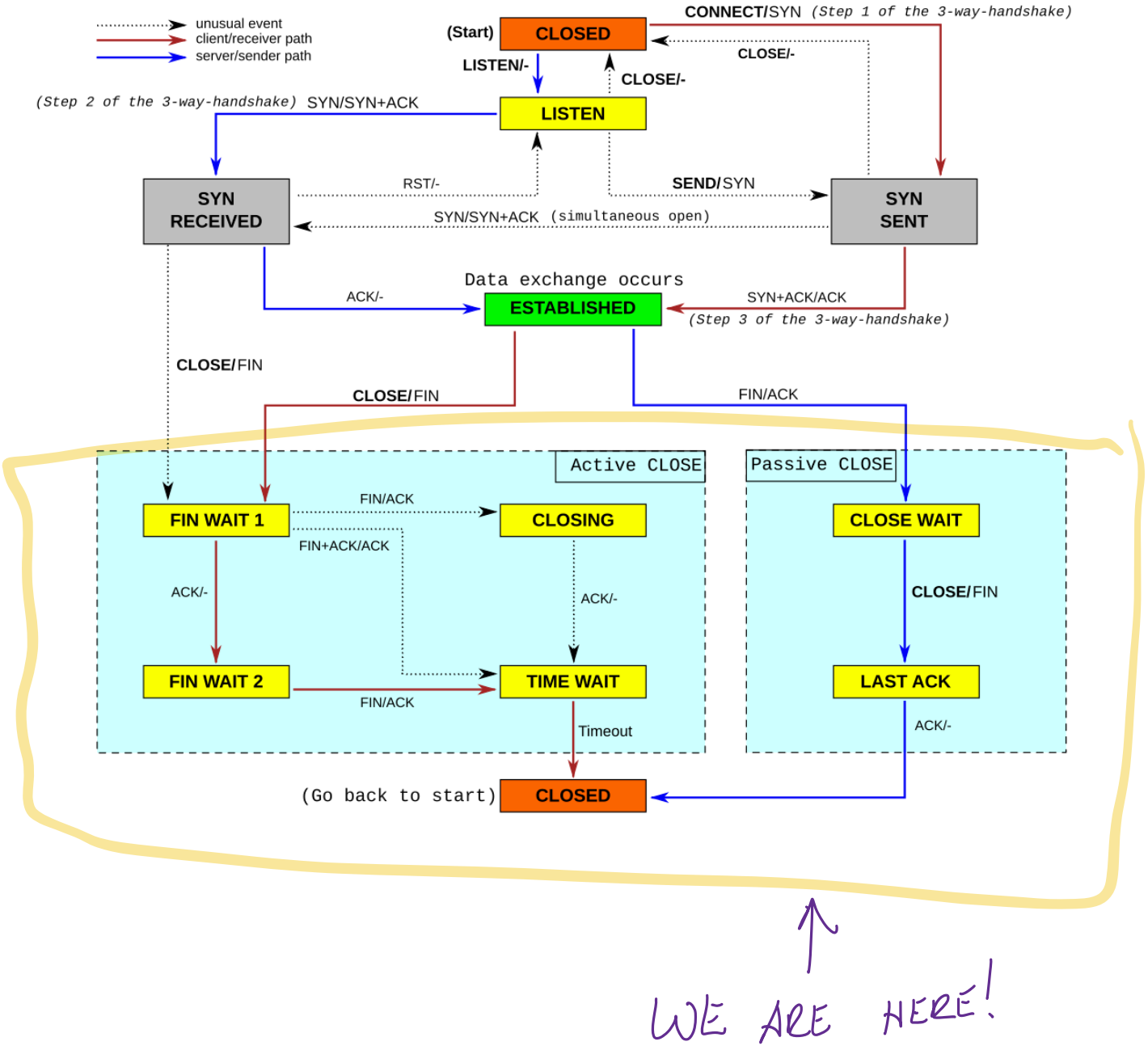


TCP: closing connections

When does a connection end?

- Normal case: call CLOSE in socket API (VClose)
  => Need to make sure all data has arrived before we are "done"
    => When we can delete the TCB (no further retransmissions are necessary)

Error cases:
- Max retransmissions exceeded (some configurable limit)
- RST flag => immediately terminate connection


unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN (Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
(Step 2 of the 3-way-handshake) SYN/SYN+ACK
LISTEN
SYN RECEIVED
RST/-
SEND/SYN
SYN SENT
SYN/SYN+ACK (simultaneous open)
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
ACK/-
CLOSE/FIN
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
Timeout
ACK/-
(Go back to start)
CLOSED

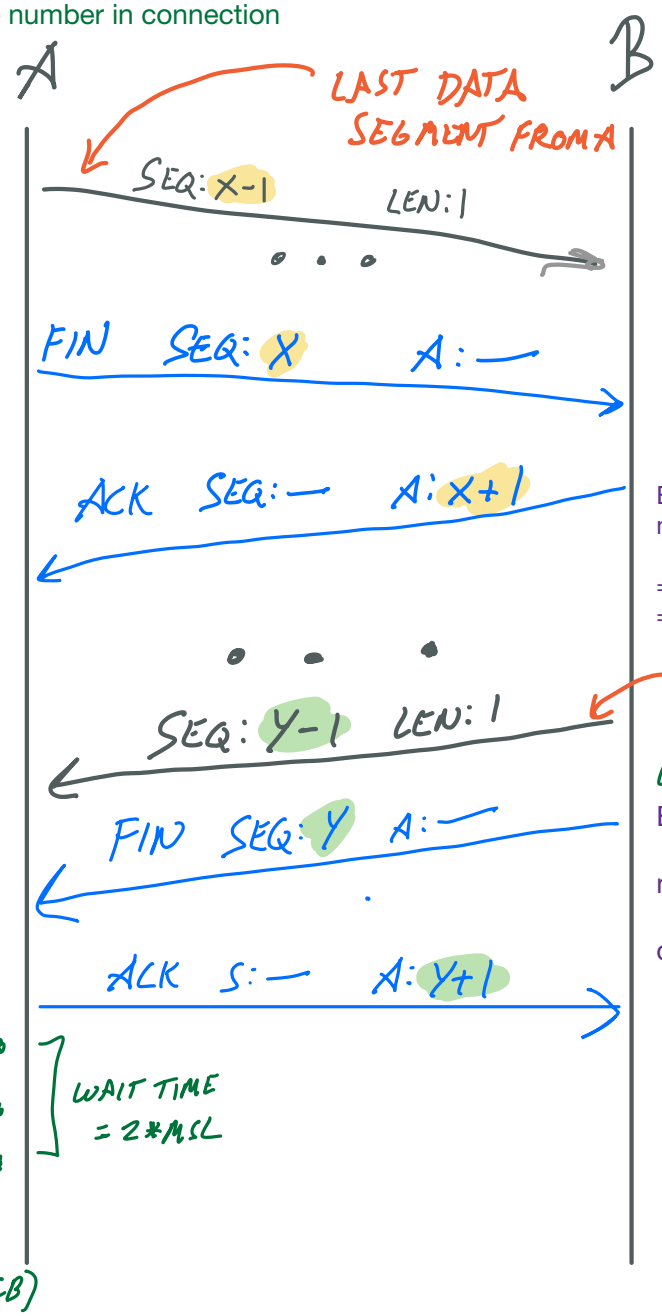
WE ARE HERE!

Closing a connection: 4-way "teardown" process

Either side can decide to CLOSE the connection
=> The endpoint that ends first does "active close"
=> The other endpoint does "passive close"

Basic idea: after sending last segment, send a FIN (set FIN flag)
=> FIN is the last sequence number in connection

App calls CLOSE => send FIN
- FIN MUST be last sequence number in data stream
- Must have sent all other data first (in terms of sending, just like any other normal segment)



FIN_WAIT_2

A can't send any more data, but could still receive data from B

CLOSE_WAIT

B ACK's FIN just like any other sequence number (wait until you have all data before it)

=> B acknowledges that A is done sending
=> However, B could still have data to send

LAST_ACK

Eventually, app on B calls close
=> Sends FIN (as last sequence number in data stream, just like A)
=> Once again, must have sent all data first

TIME_WAIT

Once A receives B's FIN, both sides have confirmed they're done sending, so we're done!

... or are we???

Even after all of this, the initiating side doesn't know the final ACK was received

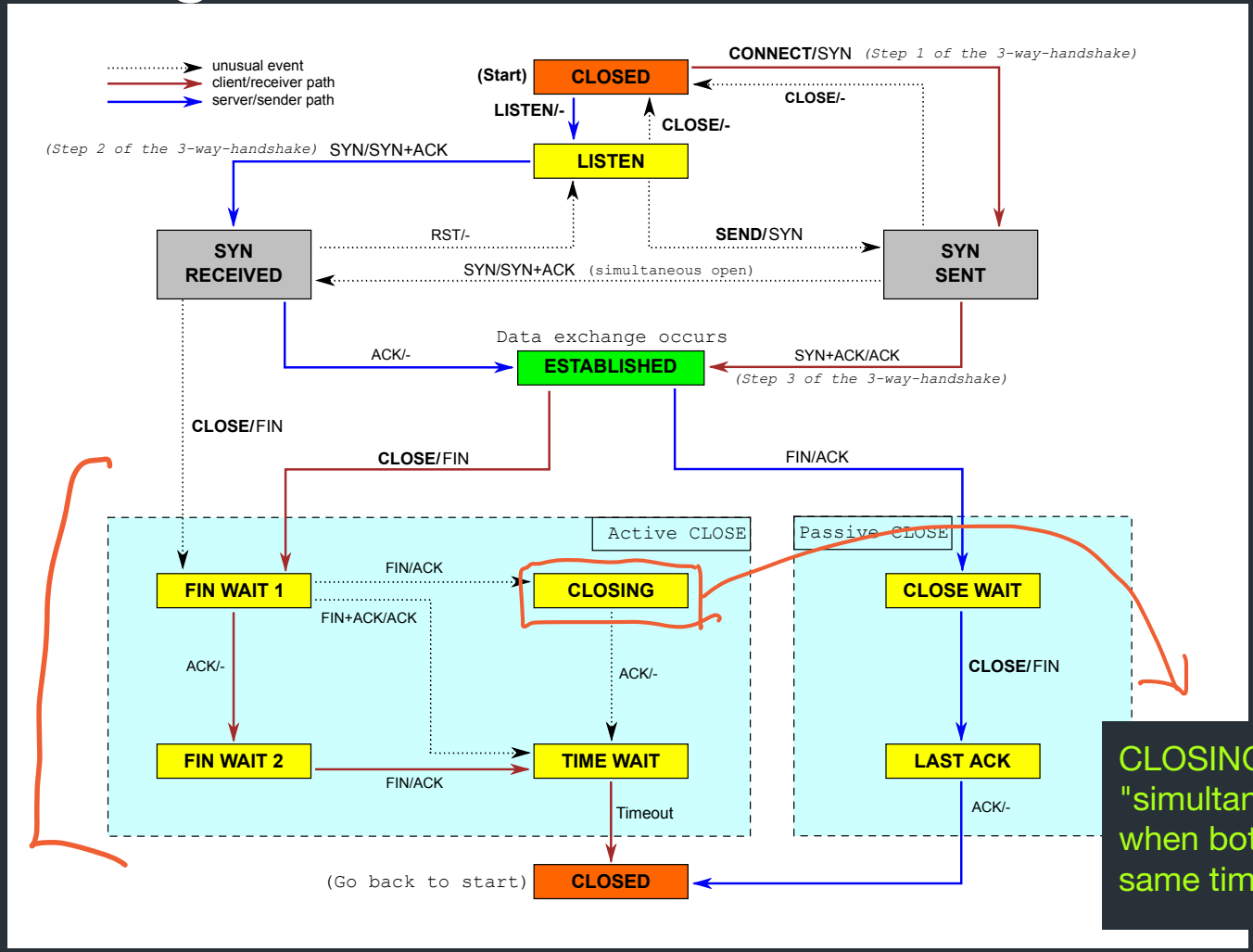
If the ACK was lost, B might need to retransmit its FIN, so A can't delete the TCB yet

Solution: need to wait a while before we can delete the TCB (purges TCP state for this connection)
=> How long to wait? 2*MSL (longest time a segment might be delayed) ~2 minutes, configurable

In practice, when we close a connection, it means we're done reading and writing
=> BUT, TCP allows you to close one side at a time (and this process is what lets us do it)

TCP State Diagram

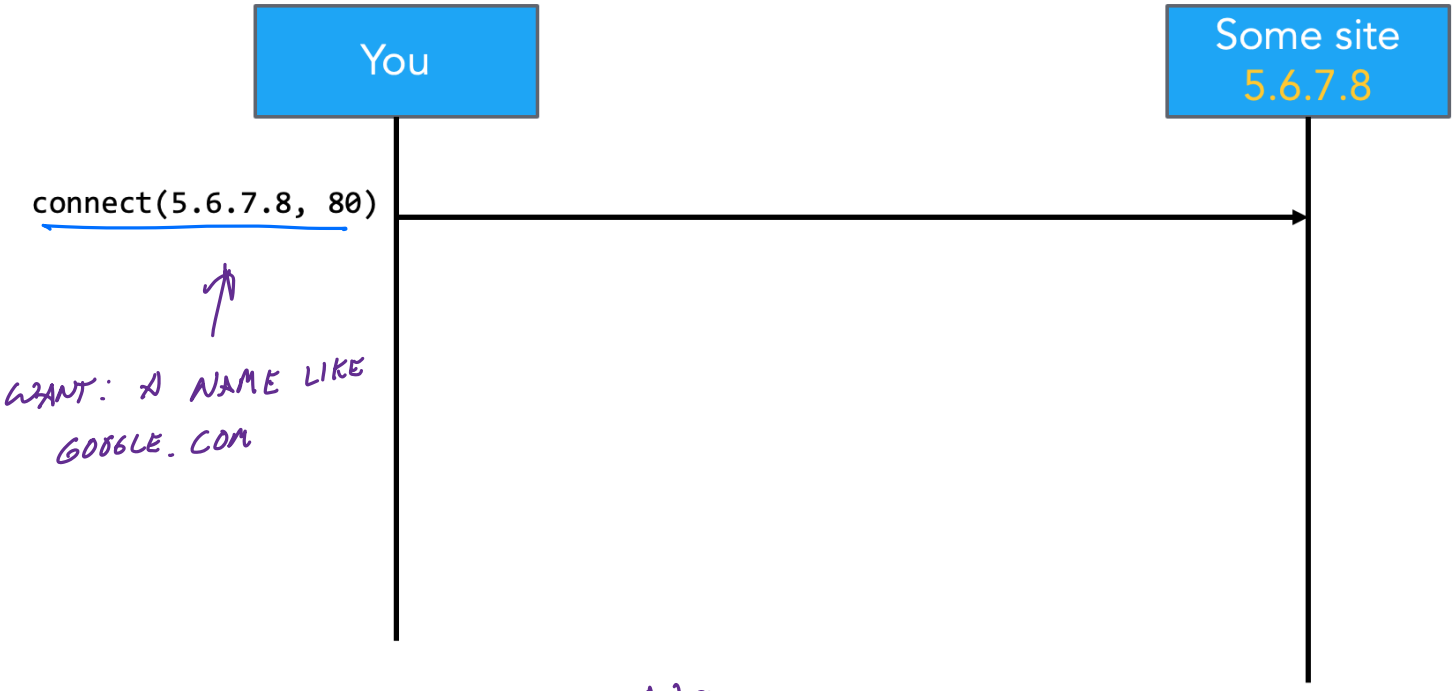
(Extra detail: an edge case we didn't talk about)



CLOSING: used only for "simultaneous close" => when both sides close at the same time (edge case)

Connecting to a server: the story so far

POV: You want to connect to some website



Is this how users interact with the network?? No.

Why DON'T we just want to use IPs?
- Typing them is annoying
- Would like to have names for "services" => multiple IPs
  => IPs usually depend on where you are located on the network
- Easier to identify (for humans, for machines in some contexts)
- Client applications don't know IPs of server

What we have

IP addresses

- Used by routers to forward packets
- Fixed length, binary numbers
- Assigned based on where host is on the network
- Usually refers to one host

Examples

- 5.6.7.8
- 212.58.224.138
- 2620:6e:6000:900:c1d:c9f7:8a1c:2f48

Efficient forwarding: ✅
Human readable: ❌
Scalable for distributed services: ❌

=> Need a new abstraction for “stuff” we are trying to access

What we have so far

IP addresses

- Used by routers to forward packets
- Fixed length, binary numbers
- Assigned based on where a host is on the network
- Usually refers to one host

Examples

- 5.6.7.8
- 212.58.224.138
- 2620:6e:6000:900:c1d:c9f7:8a1c:2f48

(This is an IPv6 address. IPv6 addresses are 128 bits instead of 32 bits, but generally otherwise behave the same as IPv4 addresses.)

In summary:

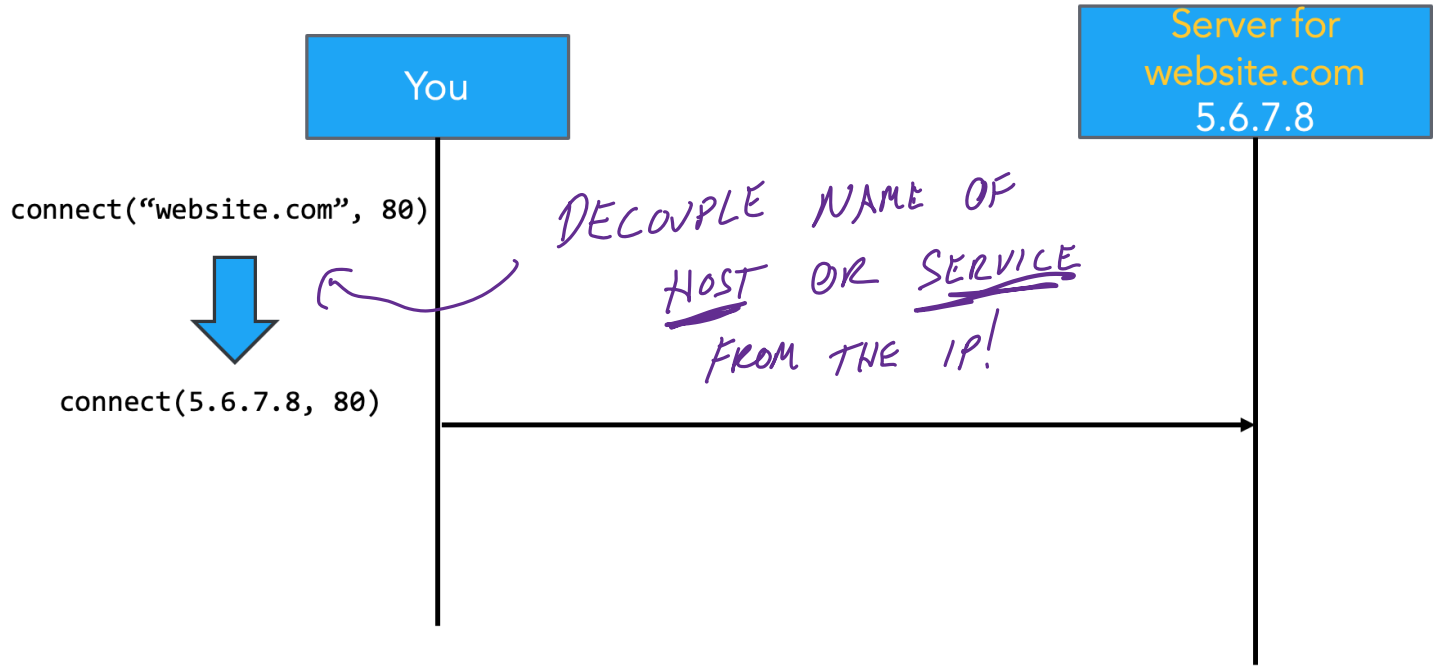
Efficient forwarding: ✅

Human readable: ❌

Scalable for distributed services: ❌

=> Need a new abstraction for "stuff" we are trying to access

What we want: a new abstraction for names



What does this mean?

"SERVICE"/"THING"

`cs.brown.edu => 128.148.32.110`

DNS

Why?

- Names are easier to remember
-
  Addresses can change underneath
  => e.g., renaming when changing providers
  Useful multiplexing sharing
  - One name => multiple addresses
  - Multiple names => one address

Remember ARP?

IP Address => Link layer address

L3 L2

Now: DNS

Names useful to users/apps => IP addresses

"WHO HAS GOOGLE.COM?" → 1.2.3.4

"QUESTION" "ANSWER"

Another change in layers, which enables so much more...

The original way (pre-DNS)

One file: hosts.txt

- Flat namespace
- Central administrator kept primary copy (for the whole Internet)
- To add host, emailed that person
- Download file regularly

Does it scale? lol no.

→ EXAMPLE ON NEXT PAGE

Domain Name System (DNS)

Originally proposed by RFC882, RFC883 (1983)

Distributed protocol to translate names => IP addresses

- Human-readable names ←
- "Distributed control"
- Load-balancing (and more)
- Much much more...

=> Distributed key-value store, before it was cool...

An example hosts.txt file from 1982

```
320 -- ************************
10-Jun-82 17:48:41-PDT,114828;000000000000
Mail-from: ARPANET host SRI-NIC rcvd at 10-Jun-82 1747-PDT
Date: 10 Jun 1982 1742-PDT
From: Dyer
Subject: Hostname table, 10-June-82
To:   dcacode252 at USC-ISI
cc:   nic

                ARPANET HOST NAMES AND LIAISON                10-Jun-82

HOST NAME     HOST ADDRESS      SPONSOR          LIAISON

ACC             10.2.0.54       VDH  ARPA    Lockwood, Gregory (LOCKWOOD@BBNC)
                                              Associated Computer Consultants
                                              414 East Cota Street
                                              Santa Barbara, California 93101
                                              (805) 965-1023

   CPUtype: PDP-11/70(UNIX)
ACCAT-TIP       10.2.0.35         ARPA    McBride, William T.
                                          (MCBRIDE@USC-ISIC)
                                          Naval Ocean Systems Center
                                          Code 8321
                                          271 Catalina Boulevard
                                          San Diego, California 92152
                                          (714) 225-2083 (AV) 933-2083

   CPUtype: H-316
AEROSPACE       10.2.0.65         AFSC    Nelson, Louis C. (LOU@AEROSPACE)
                                          Aerospace Corporation
                                          A2/1013
                                          P.O. Box 92957
                                          Los Angeles, California 90009
                                          (213) 615-4424

   CPUtype: VAX-11/780(UNIX)
AFGL            10.1.0.66         AFSC    Cosentino, Antonio
                                          (COSENTINO@AFSC-HQ)
                                          Air Force Geophysics Laboratory
                                          SUNA
                                          Mail Stop 30
                                          Hanscom Air Force Base,
```

High level DNS goals

Scalability: need to be able to have a huge number of "records" (mappings from names => addrs)
- Lots of queries to look up names
- Lots of updates (#updates << #queries)

Distributed control: need to let people/organizations control their own names

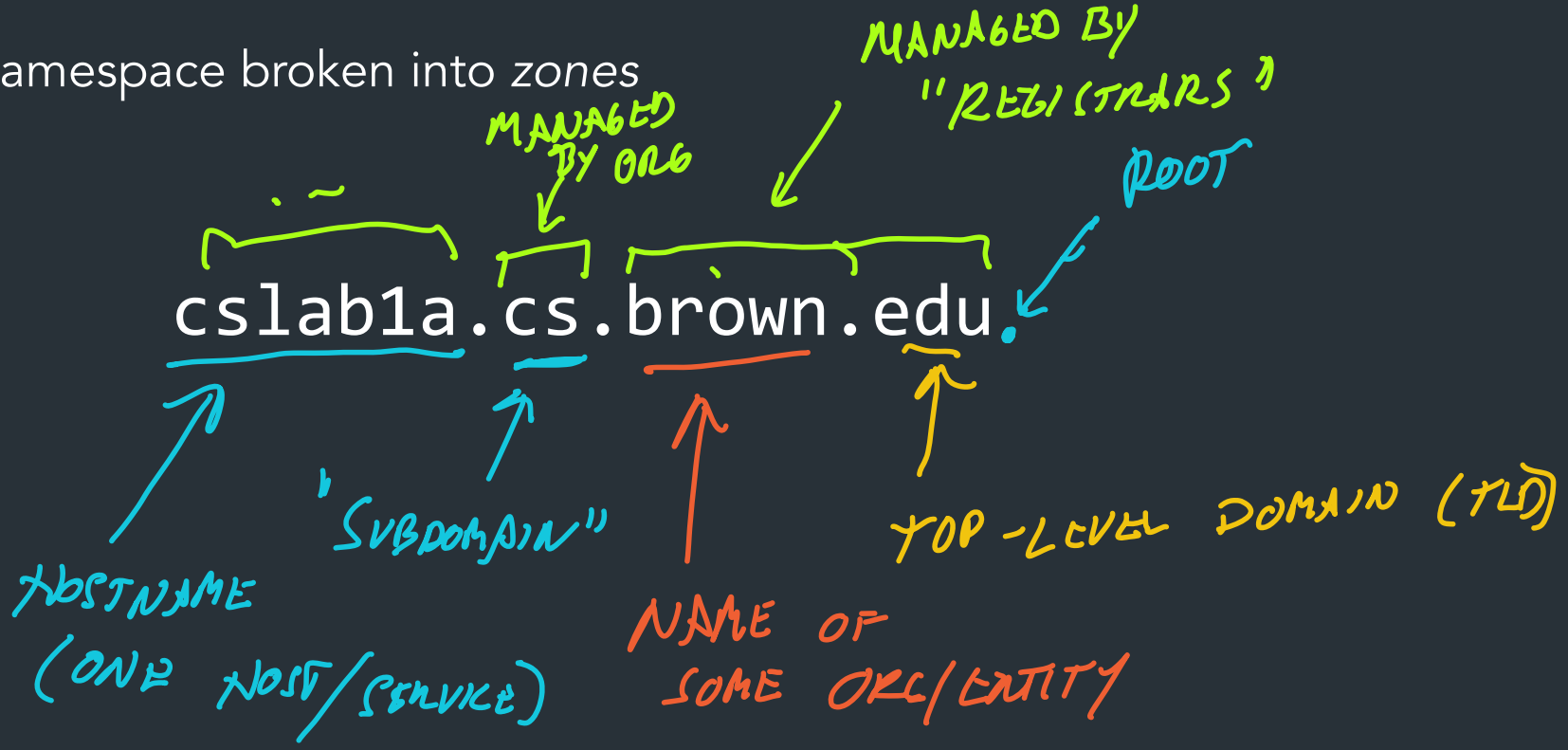
Redundancy/fault tolerance:
Redundant way to do lookups, provide records

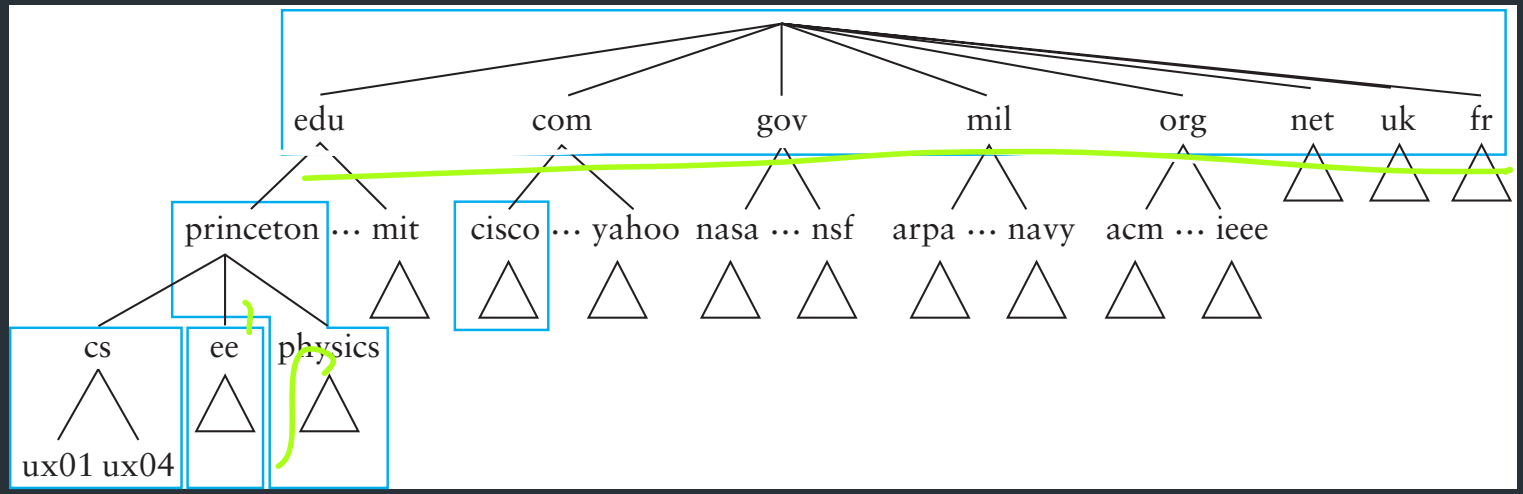
Some properties about the system that make this possible:
- Loose consistency: when changing records, not a huge deal if it takes a while to propagate (several minutes)

- Read-mostly database: writes generally infrequent, we can use lots and lots and lots of caching

How it works

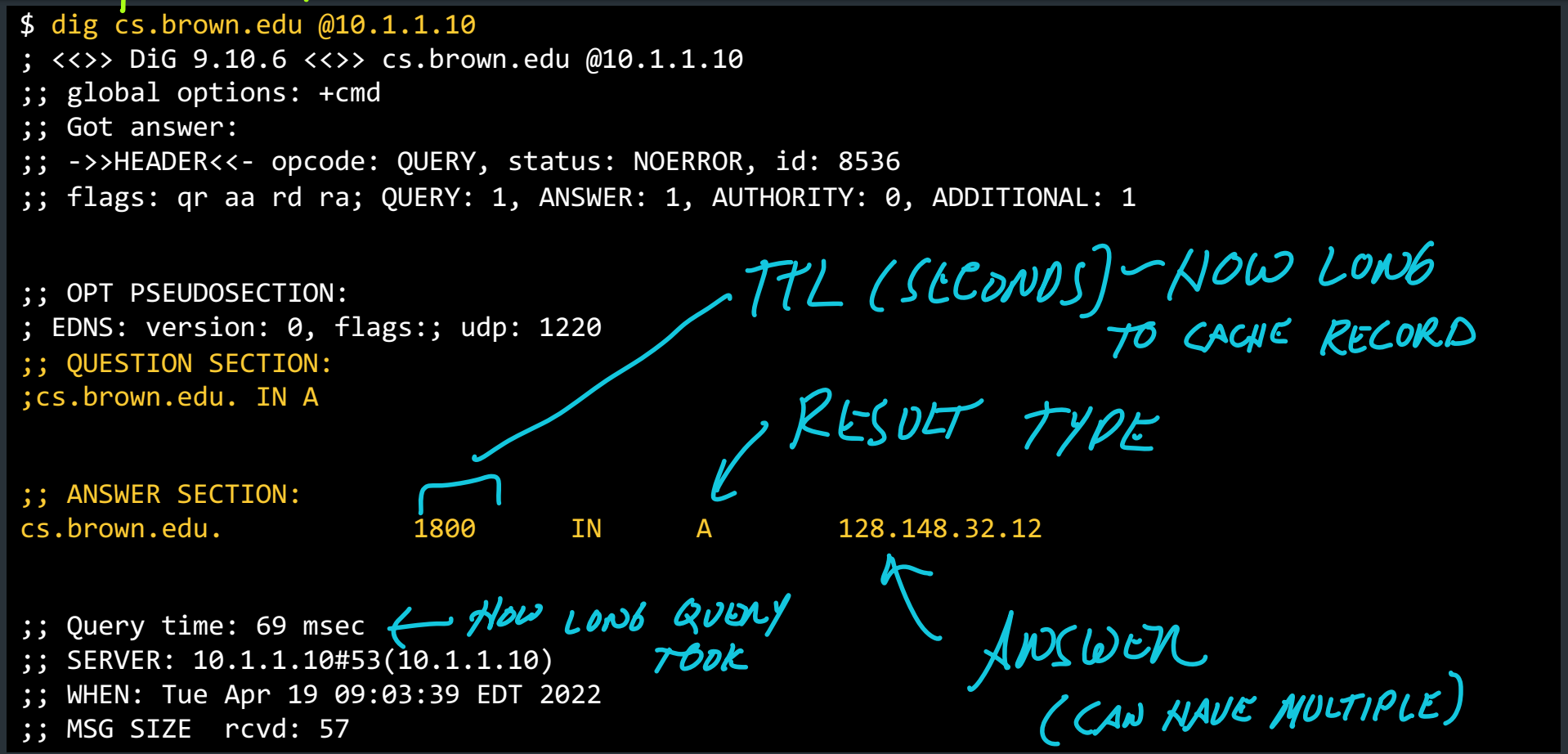
Hierarchical namespace broken into *zones*




edu
com
gov
mil
org
net
uk
fr
princeton
mit
cisco
yahoo
nasa
nsf
arpa
navy
acm
ieee
cs
ee
physics
ux01 ux04

DNS Example

(This is a typical, recursive-style query (more on what this means later))

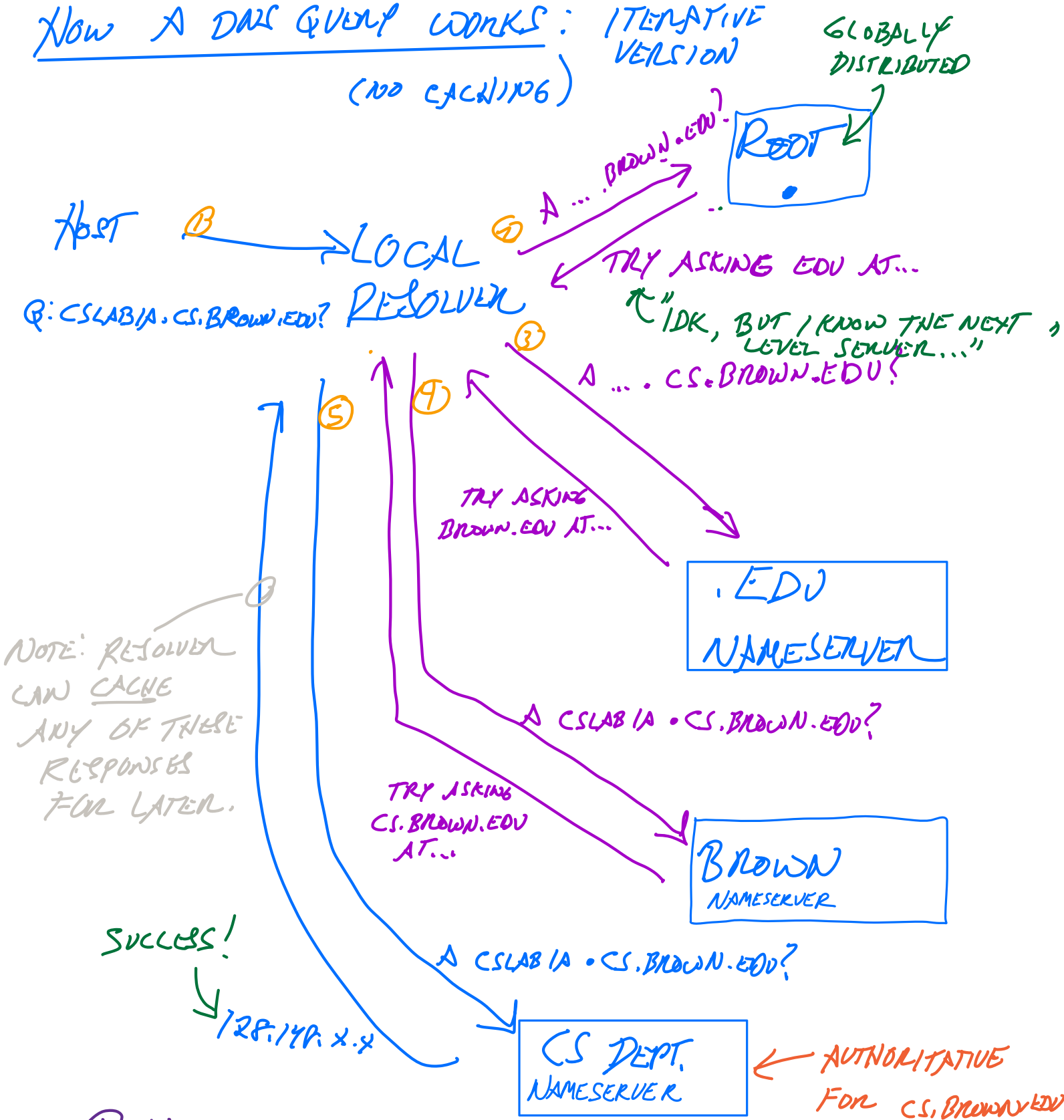


In practice: types of DNS servers

- Authoritative servers
  Servers that "own" records for some domain
  (cs.brown.edu => 128.148.x.x.)

- Resolvers

You (or another server) queries this to look up names, tries to get closer to authoritative server
=> In most cases, you interact with a resolver, it contacts an authoritative server if it doesn't know the answer
=> These are basically caches


HOW A DNS QUERY WORKS: ITERATIVE VERSION
(NO CACHING)
GLOBALLY DISTRIBUTED
ROOT
.
HOST
LOCAL RESOLVER
Q: CSLAB1A.CS.BROWN.EDU?
A ... BROWN.EDU?
TRY ASKING EDU AT...
"IDK, BUT I KNOW THE NEXT LEVEL SERVER..."
A ... CS.BROWN.EDU?
TRY ASKING BROWN.EDU AT...
.EDU NAMESERVER
A CSLAB1A.CS.BROWN.EDU?
TRY ASKING CS.BROWN.EDU AT...
BROWN NAMESERVER
A CSLAB1A.CS.BROWN.EDU?
CS DEPT. NAMESERVER
AUTHORITATIVE FOR CS.BROWN.EDU
SUCCESS!
128.148.X.X
NOTE: RESOLVER CAN CACHE ANY OF THESE RESPONSES FOR LATER.
1 HOST ASKS LOCAL RESOLVER
2 RESOLVER STARTS RECURSIVE QUERY FROM ROOT
⇒ 2 3 4 INTERMEDIATE NAMESERVERS DON'T HAVE ANSWER, BUT RESPOND W/ NEXT SERVER THAT KNOWS MORE
5 FOUND SERVER W/ AUTHORITATIVE ANSWER!

Iterative query: step 1

```
$ dig cs.brown.edu @e.root-servers.net

; <<>> DiG 9.10.6 <<>> cs.brown.edu @e.root-servers.net
[ . . .]
;; QUESTION SECTION:
;cs.brown.edu. IN A

;; AUTHORITY SECTION:
edu. 172800 IN NS b.edu-servers.net.
edu. 172800 IN NS i.edu-servers.net.
edu. 172800 IN NS g.edu-servers.net.
[ . . .]

;; ADDITIONAL SECTION:
[ . . .]
i.edu-servers.net. 172800 IN A 192.43.172.30
g.edu-servers.net. 172800 IN A 192.42.93.30
b.edu-servers.net. 172800 IN A 192.33.14.30


;; Query time: 123 msec
;; SERVER: 2001:500:a8::e#53(2001:500:a8::e)
;; WHEN: Thu Oct 31 08:29:45 EDT 2024
;; MSG SIZE  rcvd: 839
```

ASK ROOT NAMESERVER

QUERY

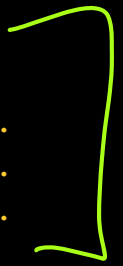
No answer, but try these authoritative servers (for .edu)

Additional records: "BTW, here are the IPs for those other nameservers to try"

=> These are called "glue records" (needed because resolving b.edu-servers.net would otherwise require another DNS query, and possibly have a circular dependency)

Iterative query: step 2

```
$dig cs.brown.edu @192.33.14.30.  [192.33.14.30 was IP returned for b.edu-servers.net]

; <<>> DiG 9.10.6 <<>> cs.brown.edu @192.33.14.30
[ . . . ]
;; QUESTION SECTION:
;cs.brown.edu. IN A


;; AUTHORITY SECTION:
brown.edu. 172800 IN NS ns1.ucsb.edu.
brown.edu. 172800 IN NS bru-ns1.brown.edu.
brown.edu. 172800 IN NS bru-ns2.brown.edu.
brown.edu. 172800 IN NS bru-ns3.brown.edu.


;; ADDITIONAL SECTION:
ns1.ucsb.edu. 172800 IN A 128.111.1.1
ns1.ucsb.edu. 172800 IN AAAA 2607:f378::1
bru-ns1.brown.edu. 172800 IN A 128.148.248.11
bru-ns2.brown.edu. 172800 IN A 128.148.248.12
bru-ns3.brown.edu. 172800 IN A 128.148.2.13
```



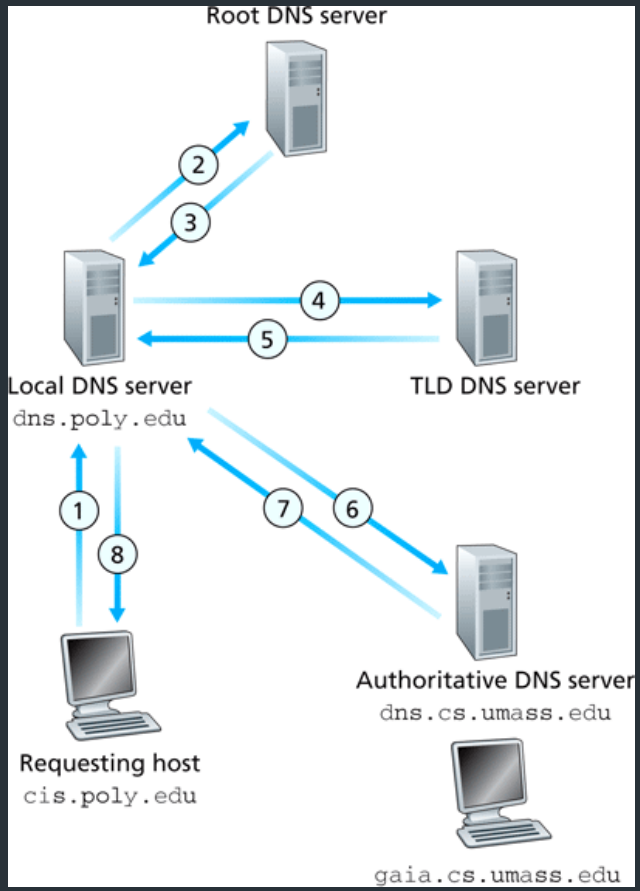
TRY BROWN.EDU NAMESERVERS

```
$ dig cs.brown.edu @128.111.1.1  [128.111.1.1 was IP returned for ns1.ucsb.edu]
; <<>> DiG 9.10.6 <<>> cs.brown.edu @128.111.1.1
[ . . . ]

;; QUESTION SECTION:
;cs.brown.edu. IN A

;; ANSWER SECTION:
cs.brown.edu. 1800 IN A 128.148.32.12


;; Query time: 77 msec
;; SERVER: 128.111.1.1#53(128.111.1.1)
;; WHEN: Thu Oct 31 08:35:11 EDT 2024
;; MSG SIZE  rcvd: 57
```

Resolver operation

Summary of what we just saw

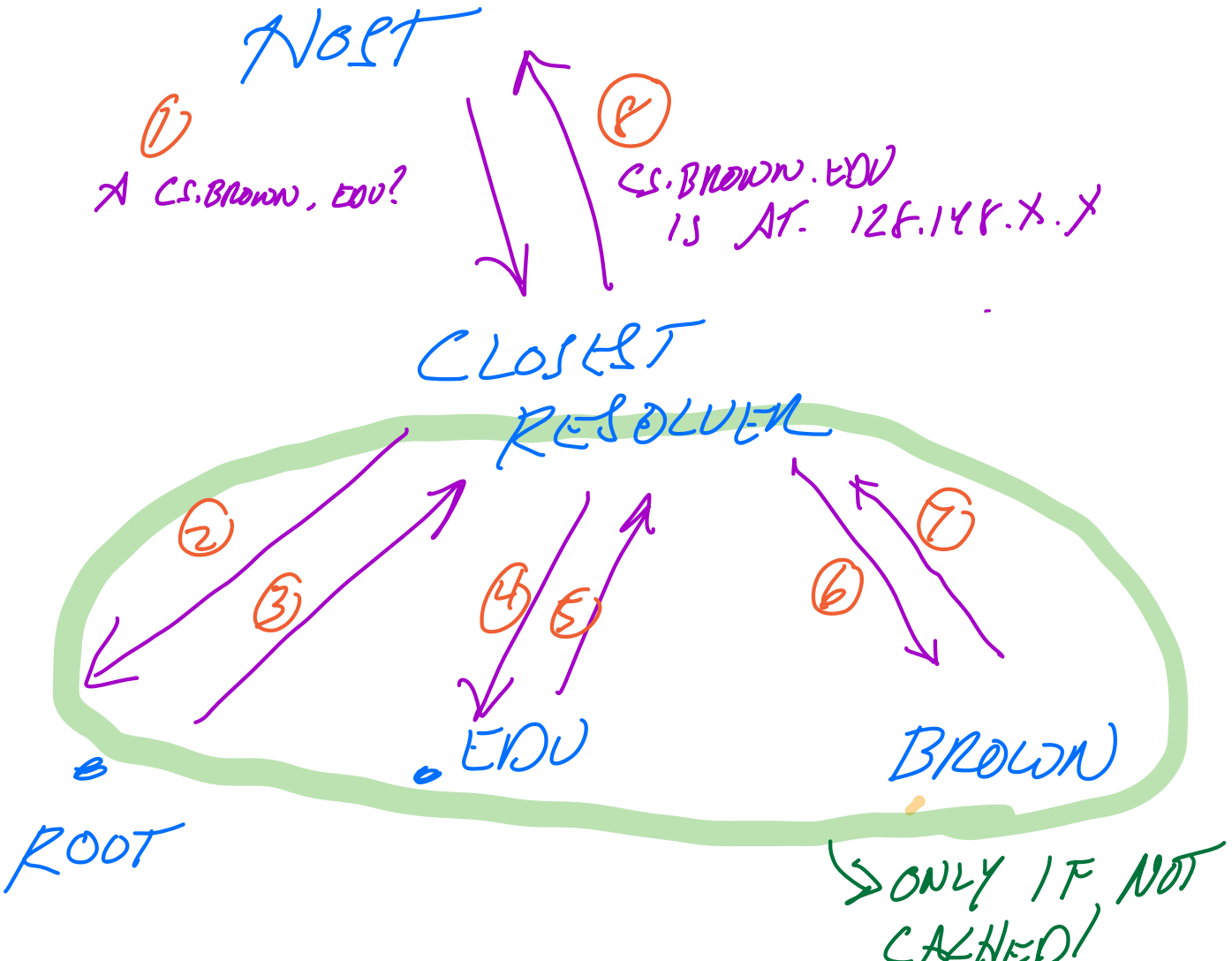
- Apps make recursive queries to local DNS server (1)
  - Ask server to get answer for you
- Server makes iterative queries to remote servers (2,4,6)
  - Ask servers who to ask next
  - Cache results aggressively



Preview notes on the (more common) recursive version of a DNS query after this

Feel free to read ahead!

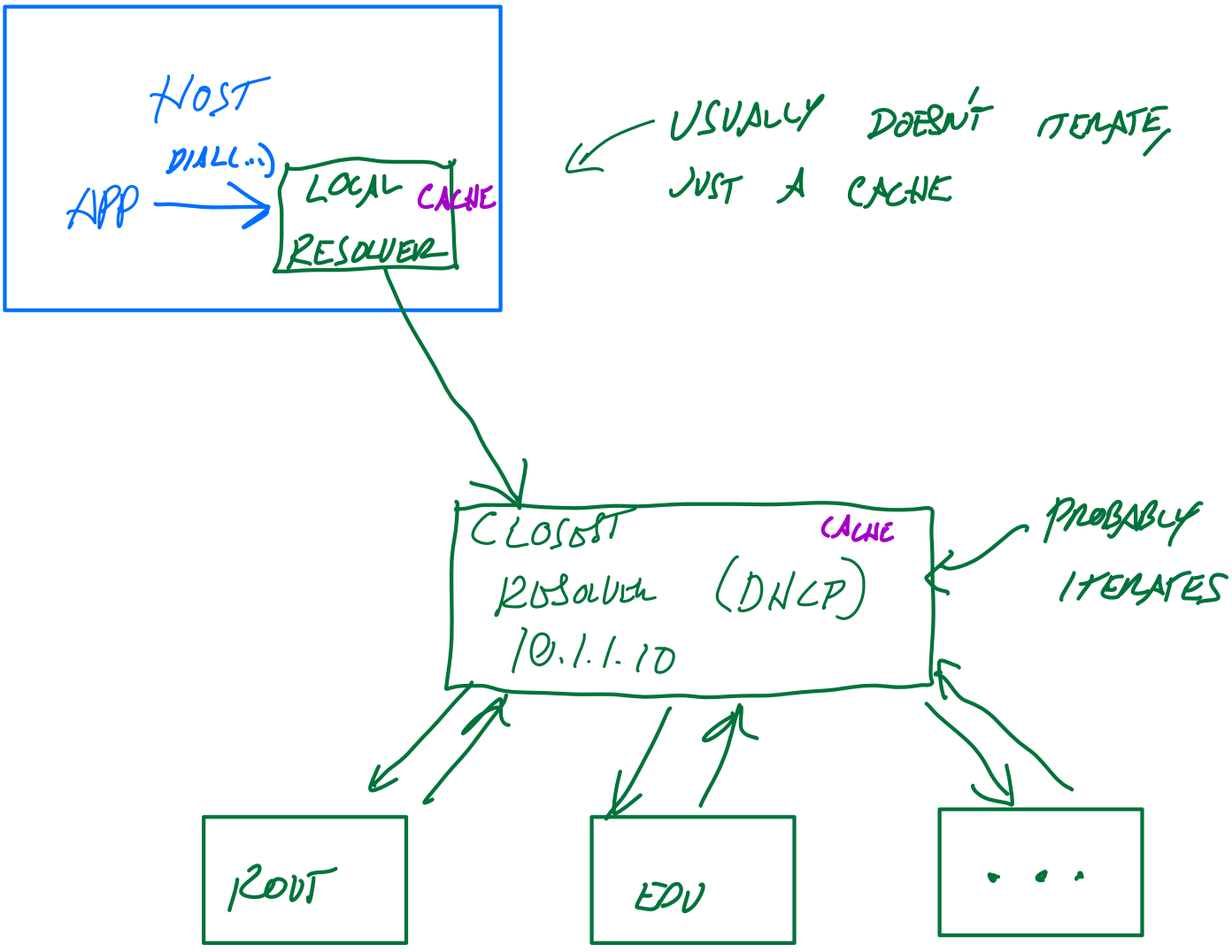
RECURSIVE DNS QUERIES (MORE COMMON)

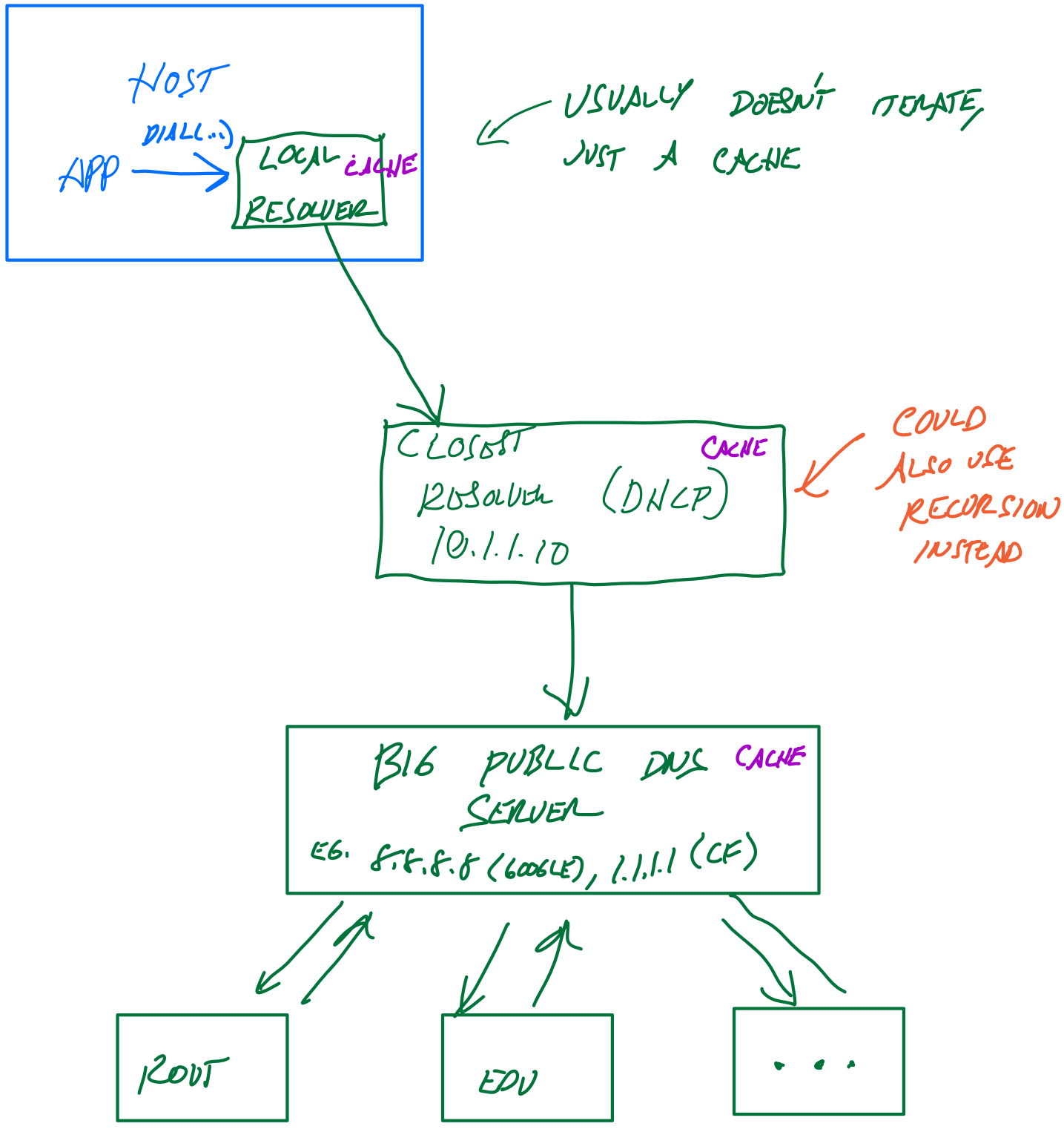


More commonly, hosts perform recursive queries to larger DNS servers, which do the typical iteration process (from the previous page) on the client's behalf.

Why? All resolvers cache responses—a larger resolver is more likely to have these entries in its cache. If the resolver has a valid answer for any of the steps, it can skip it! (For example, if the nameserver for .edu is cached but cs.brown.edu is not, the local resolver can skip skeps 2-3.

Who provides the closest resolver?
- Many OSes have a resolver on the local system, which acts as a local cache
- Usually, every local network has its own resolver (Brown, your home router, etc)
- These local resolvers MIGHT do iterative queries, but often do another recursive step to a big public DNS server (like Google's 8.8.8.8, or Cloudflare's 1.1.1.1)
  => Multiple levels of caching!


HOST
DIALL..)
APP
LOCAL
CACHE
RESOLVER
USUALLY DOESN'T ITERATE, JUST A CACHE
CLOSEST
CACHE
RESOLVER (DHCP)
10.1.1.10
PROBABLY ITERATES
ROOT
EDU
. . .


HOST
APP
DIAL...)
LOCAL CACHE RESOLVER
USUALLY DOESN'T ITERATE, JUST A CACHE
CLOSEST RESOLVER (DHCP) CACHE
10.1.1.10
COULD ALSO USE RECURSION INSTEAD
BIG PUBLIC DNS SERVER CACHE
EG. 8.8.8.8 (GOOGLE), 1.1.1.1 (CF)
ROOT
EDU
...