Computer Networks 3

(Starting at 9:10 PM)

Agenda

- HTTP
- DNS
- IPv4 and IPv6
- NAT
- TCP
  - ↳ Handshake
- TCP v/s UDP → Depth

LLD

Brief

Depth

Web

CN4:

Socket Programming ⇐

CN

1 Unix

→ Mac

→ UBuntu

2 IntelliJ Idea

Ultimate

Spring Boot

VS code

HTTP (Hypertext Transfer Protocol)

https://scaler.com

HTTPS

HTML

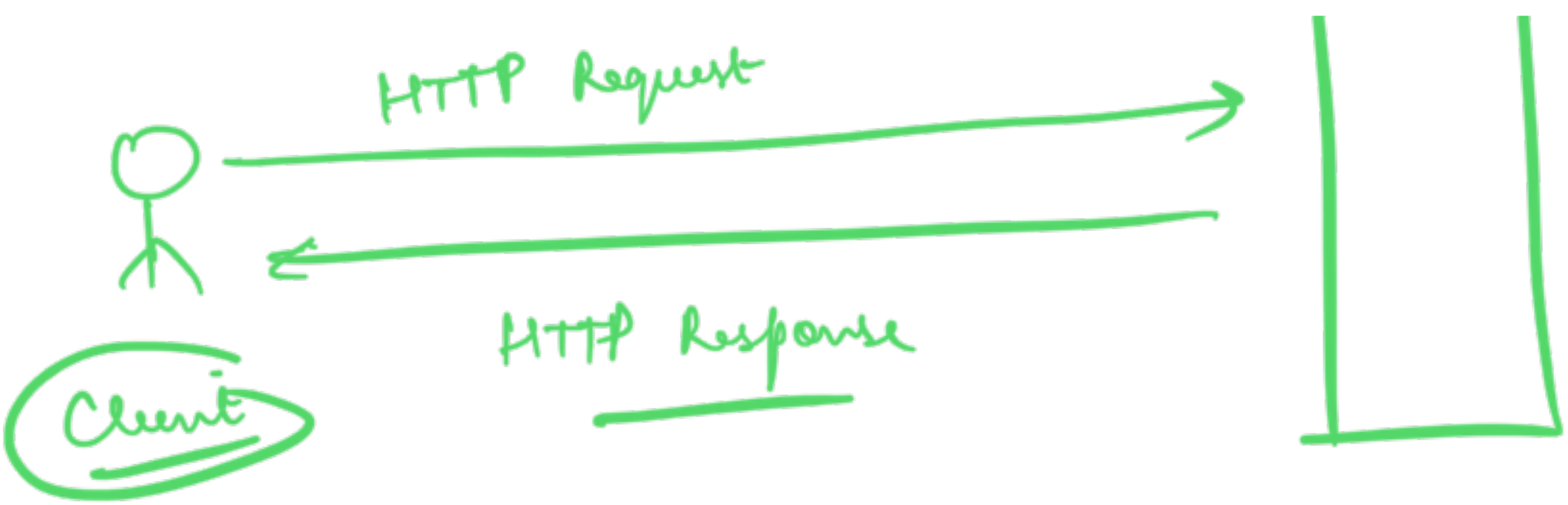
HyperText

Markup

language

 —

→ Client Server Protocol



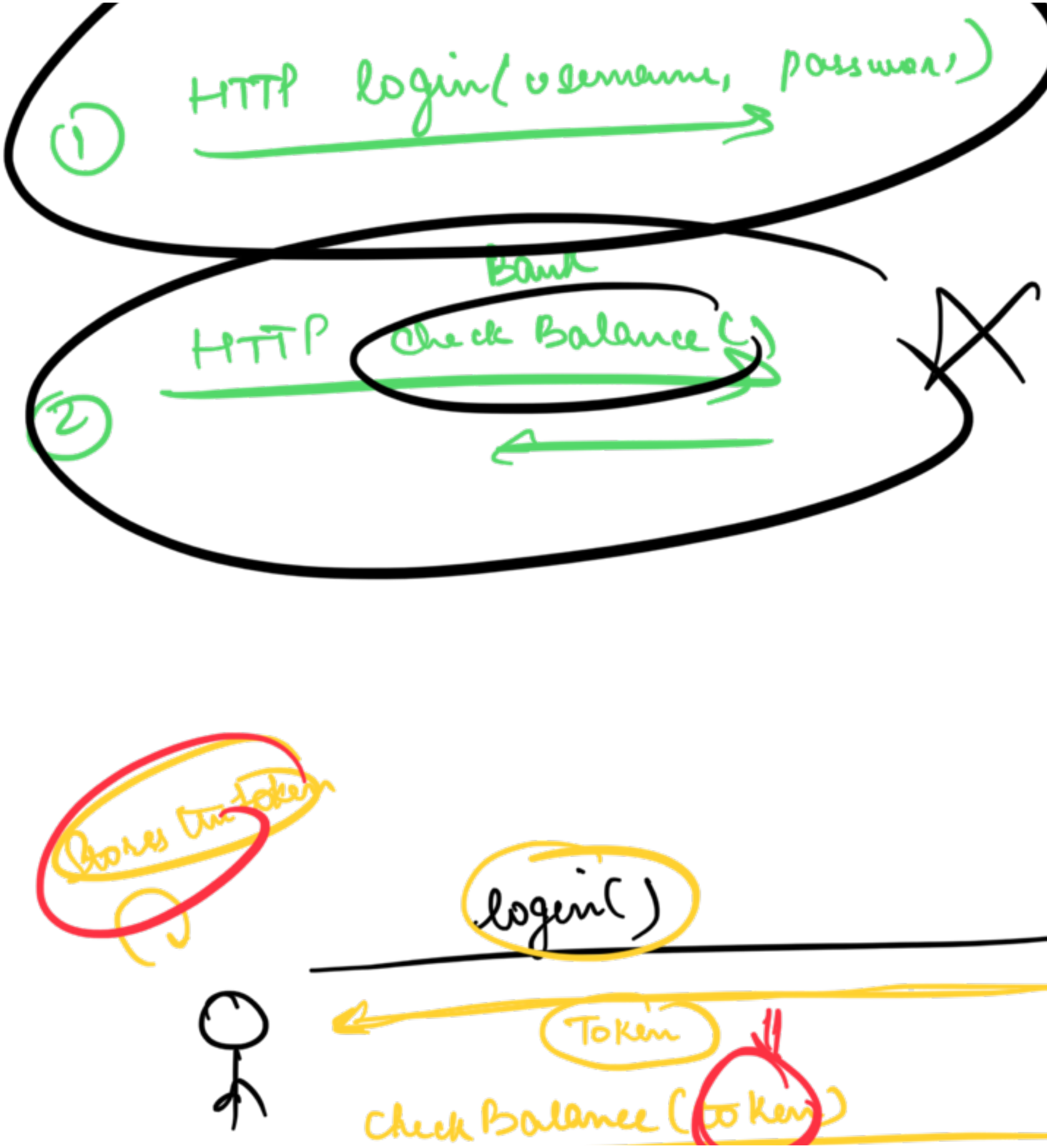
⇒ App^{n} layer protocol

→ HTTP uses TCP

1 Stateless Protocol

⇒ Every request sent via HTTP is independent of any other request

⇒ By default ⇒ no state is maintained on servers.


HTTP login(username, password)
1
Bank
HTTP Check Balance( )
2
Map<User: Token>
Token
01:
Store Token
login( )
Token
Check Balance(token)

Balance

Object → ~~Example~~ Via HTTP we fetch objects

→ Everything is an object / file

→ location?

→ Resource / Obj at a particular location

Scaler.com

→ Scaler.com/index.html


Objects


URL: Uniform Resource Locator
hostname
Port
Arguments
http://
www.google.com
/go
/some-path
?q=id=
&uri=

protocol

domain
IP address

location
Path on the server

& ultra =

Websocket
ws://

ftp://

jdbc://

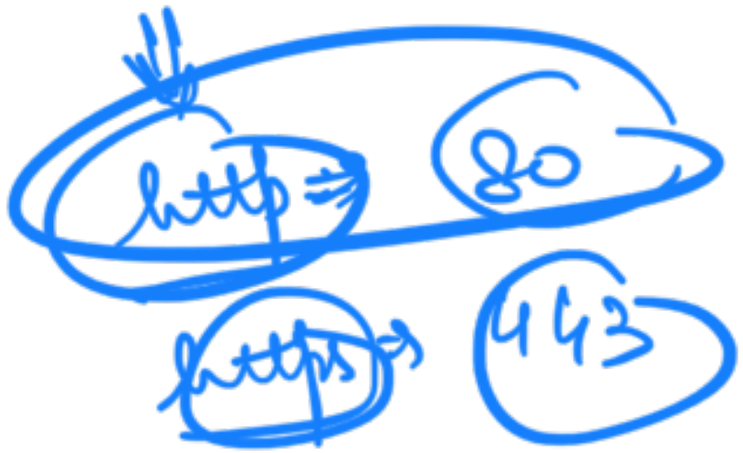
Staging Website | Dev Website

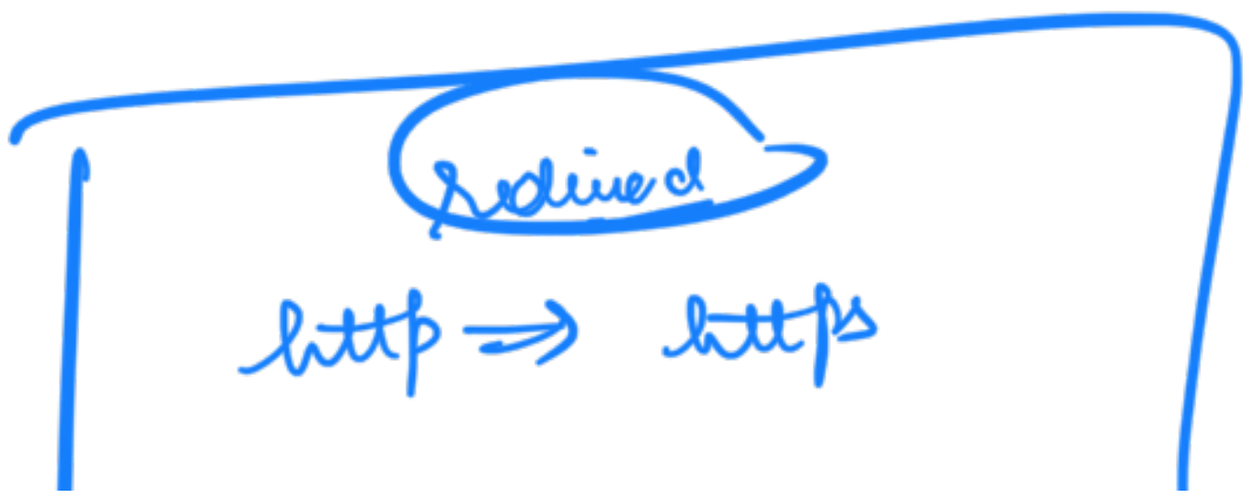
: 5341

444

Scaler.com : 8184 → jenkin / ci/cd

: 4000 → bug fix.





:80 ⇒ :443

:80

:443

Nginx

Nginx

:5151

:80

:443

:80

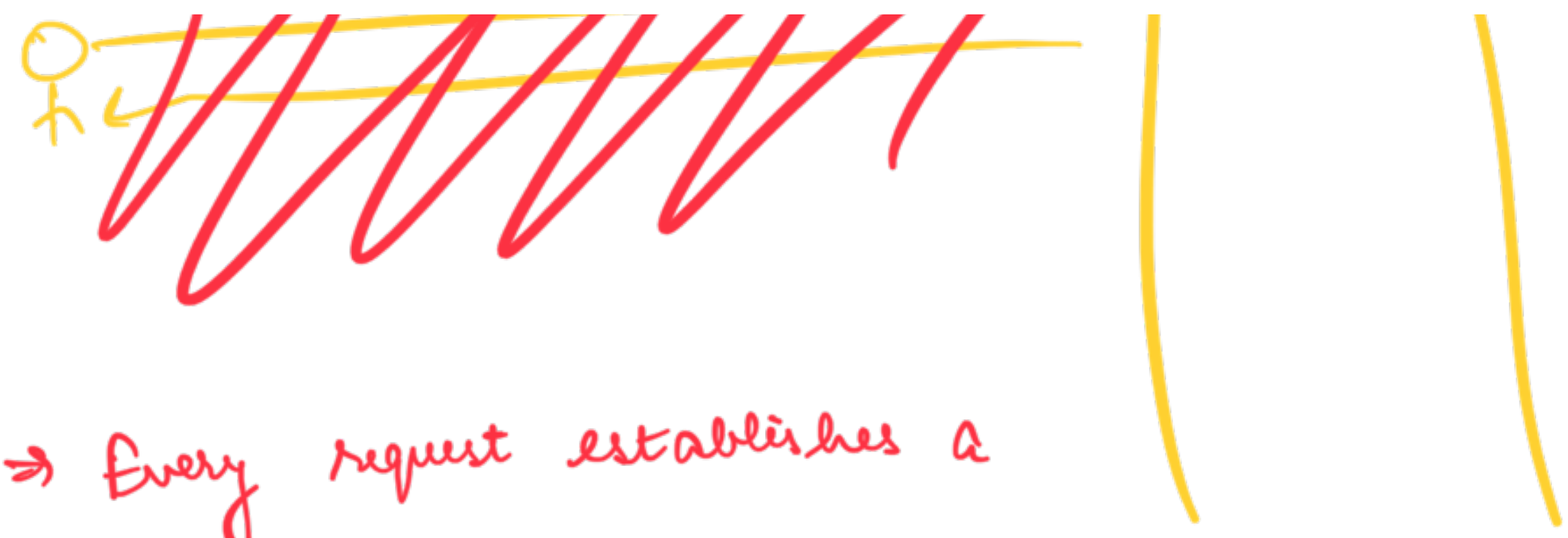
:5151

HTTP

Types of HTTP Requests

Non Persistent

⇒ TCP conn is established per request



→ Every request establishes a new TCP connection

→ Once the response is received connection is thrown away

Cons: Every req will require setting up TCP which is going to slow con

Persistant

Client

Server

Village A

Village B

Con

Maintaing sessions consumes memory

Timeout : 20s

↓

conn will be automatically broken after

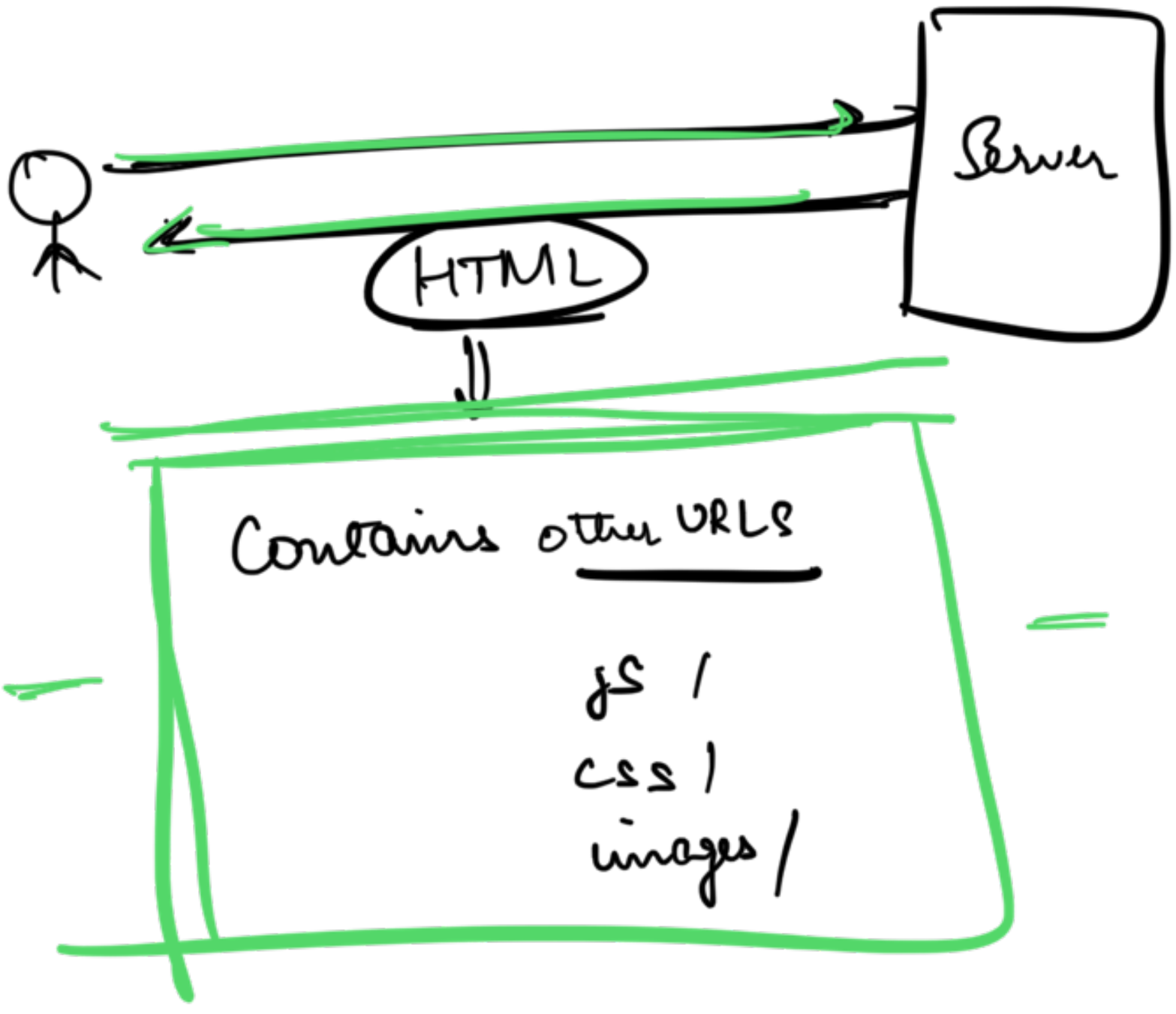
timeout seconds.

→ Non Persistent is used majority of times

* Persistent is used when
frequent data transfers
btw client & server

eg: Streaming

Live Code Editor



When we send an HTTP request

formal

HTTP Method → Request Method

GET, POST, PUT,

HTTP 1.0 → Non persistent

1.1 → Persistent

Scaler.com ⇒ GET

1. GET ⇒ To fetch data from a particular location

2. POST ⇒ Send some data to the server without knowing the final location of data

3. PUT ⇒ upload data at a particular URI

4. HEAD

5. DELETE

6. ~~UPDATE~~

7. PATCH

POST /file

/file/1234

PUT /file/123

GET /file/123

POST

/ users →

{
Name: Naman
email : 123
}

GET /users

POST doesn't know the final loc^{n} of data

POST /users

{
Name: —
email: —

Server returns the final URL of the data

/user/123

GET /users/123

PUT

→ Client knows the final loc^n of the data

If nothing at that loc => it creates a new file

else => updates

GET

@Get

Sign up(email, password)

2

Flipkart

GET

Search ? type = SHIRT & low-price = 1000 &

high-price = 5000 & brand =

PUMA

HTTP Get Request has a size limit

256

POST

GET / POST

When you get a response from Server:

Status Code

1xx ⇒ info

2xx ⇒ Success

3xx ⇒ Redirect

307

4xx ⇒ Client's fault

404 : Nothing exist

403 : Permission Denied

5xx ⇒ Server fault

502 : Bad Gateway

Break till 10:43 PM

HTTP(S) is Stateless

Cookies

⇒ Nothing but a string / token

⇓

⇒ Whenever we get a response

Set-cookie ⇒ " string / token "

Cookies

⇒ used to simulate (state)

⇒ Server sends a cookie (via)

Set-cookie

in response header

⇒ Browser stores all cookies in its memory

↳ whenever any req is sent to that domain
in future all the cookies are sent.
automatically.

why are cookies

1. Auth

(try deleting cookies → logged out)

2) Track You

Youtube.com

← Auto

← Trackers

- Cookie for google analytic
- Cookie for mixpanel
- Cookie for Microsoft clarity

request (Request)
|
if Request.cookie[auth] == dummy fn)
Say login

10 Min

DNS

(How DNS and BGP work)

Why FB was down for multiple hr

Behind XYZ Tree, UP

IP addressing

google.com

19.2.131.41.51

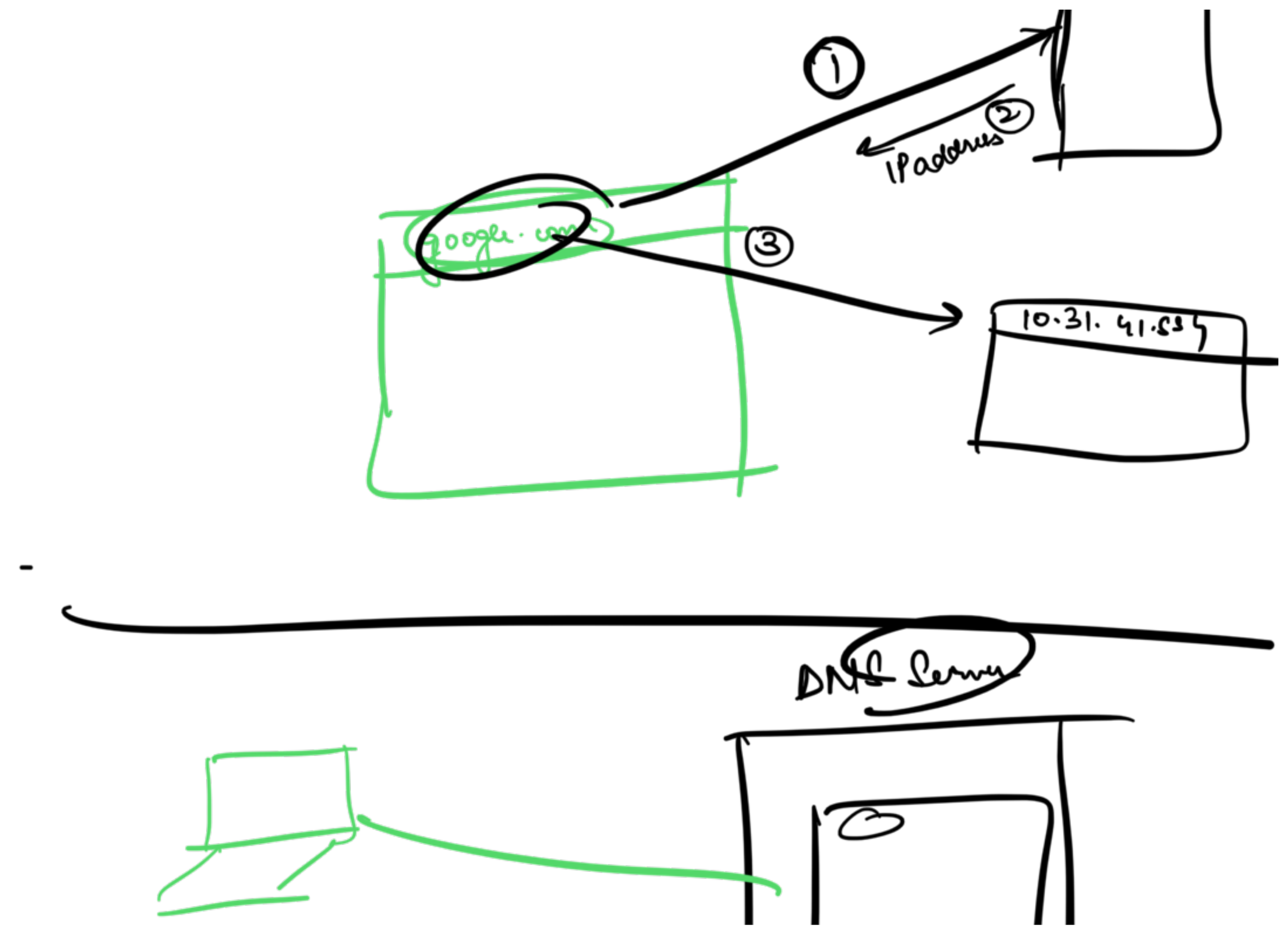
fb.com

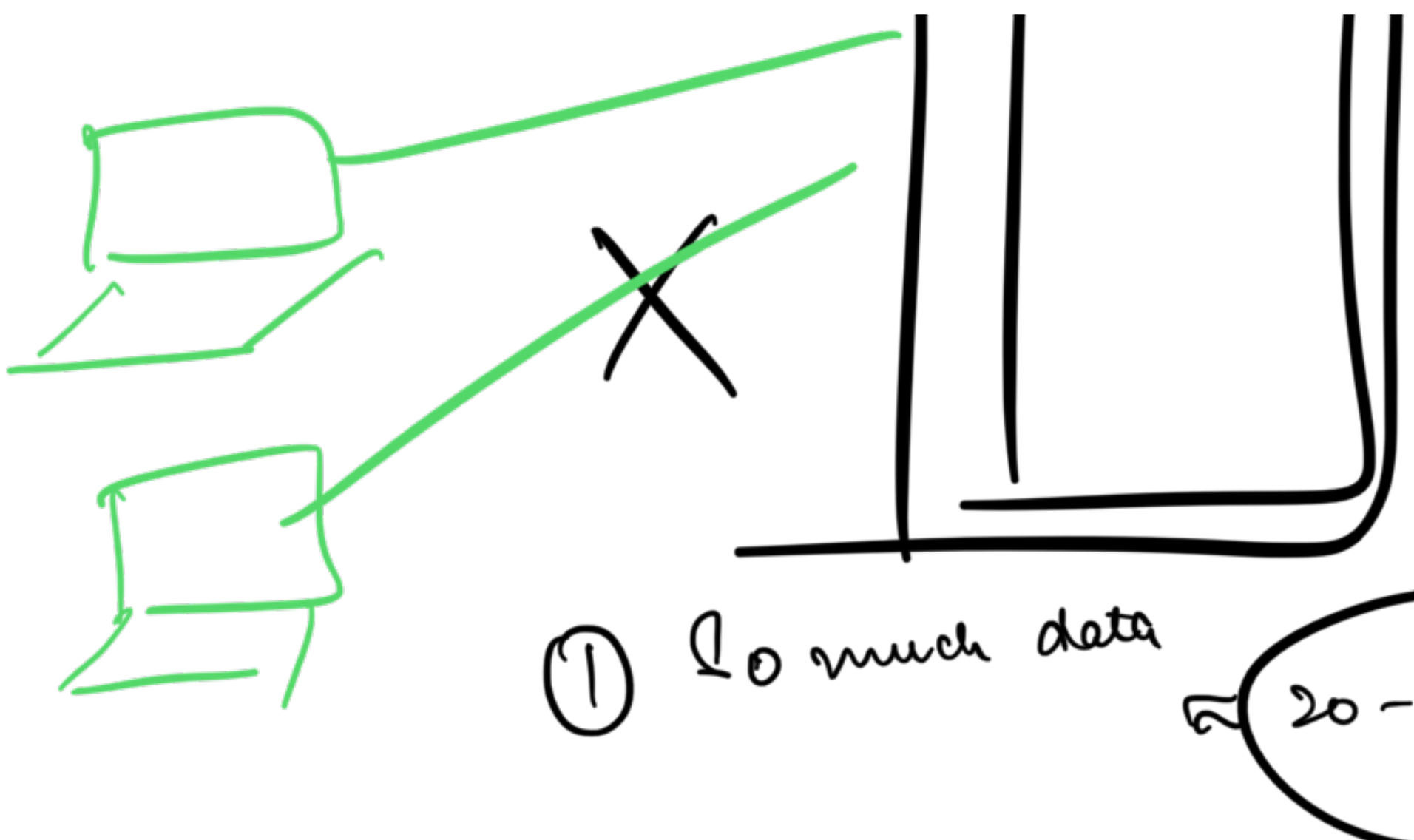
10.41.71.139

→ Humans are good in remembering names

→ But computers work on numbers

→ There is need of a system that converts domain Name to IP address


1
2
IPaddress
google.com
3
10.31.41.53
DNS Server



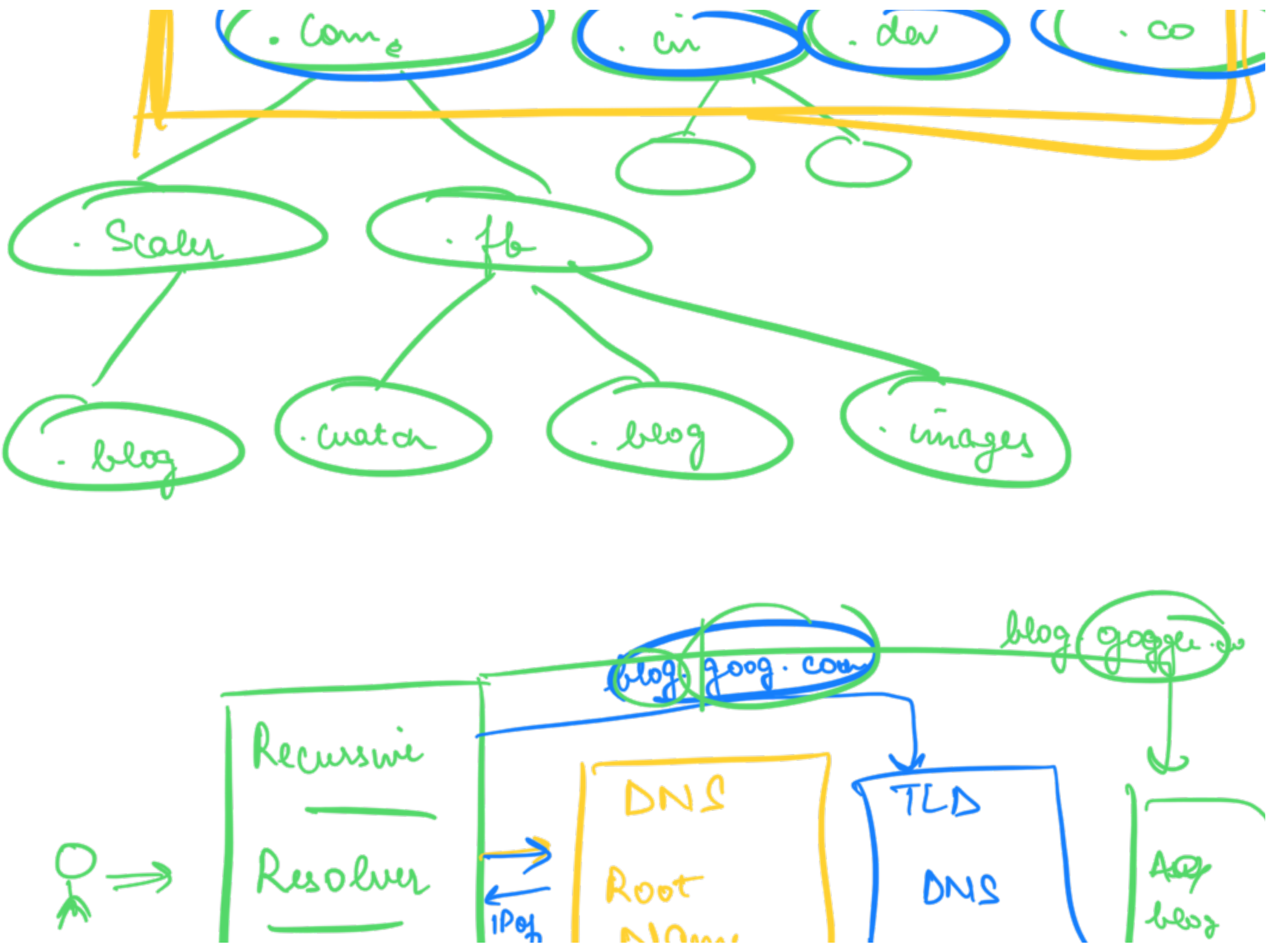
How DNS works

Hierarchical Database

naman.dev

Scaler.com

google.com


.com
.in
.dev
.co
.Scaler
.fb
.blog
.watch
.blog
.images
blog.goog.com
blog.google.in
Recursive
Resolver
IP of
DNS
Root
TLD
DNS
Auth
blog

TLD
NS

Name
Server

blog.google

IP addrss of TLD
NAme Server

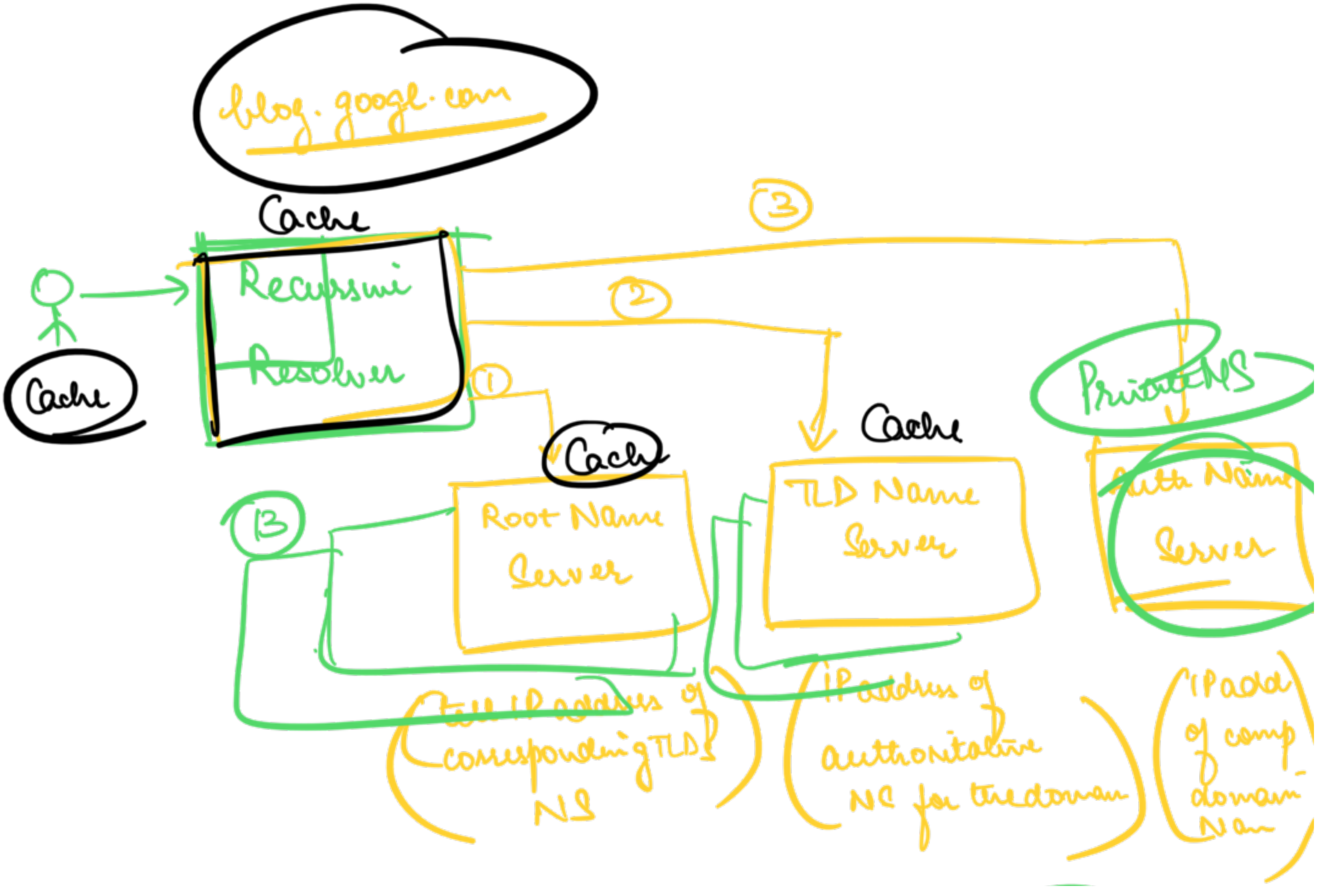
joins
play
wabel

Authoritative
Name
Server

E J O T Y

5 10 15 20 25

TLD ⇒ Top level Domain


blog.google.com
Cache
Recursive
Resolver
Cache
1
Cache
Root Name
Server
B
(IP address of
corresponding TLD
NS
3
2
Cache
TLD Name
Server
IP address of
authoritative
NS for the domain
Authoritative Name
Server
(IP addr
of comp
domain
Nam

DNS involves a lot of caching

TTL

IP addresses of DNS resolvers
Maintained by big comps

8.8.8.8 ⇒ google

0.0.0.0 ⇒ google

1.1.1.1 ⇒ Cloudflare

⇒ ISPs