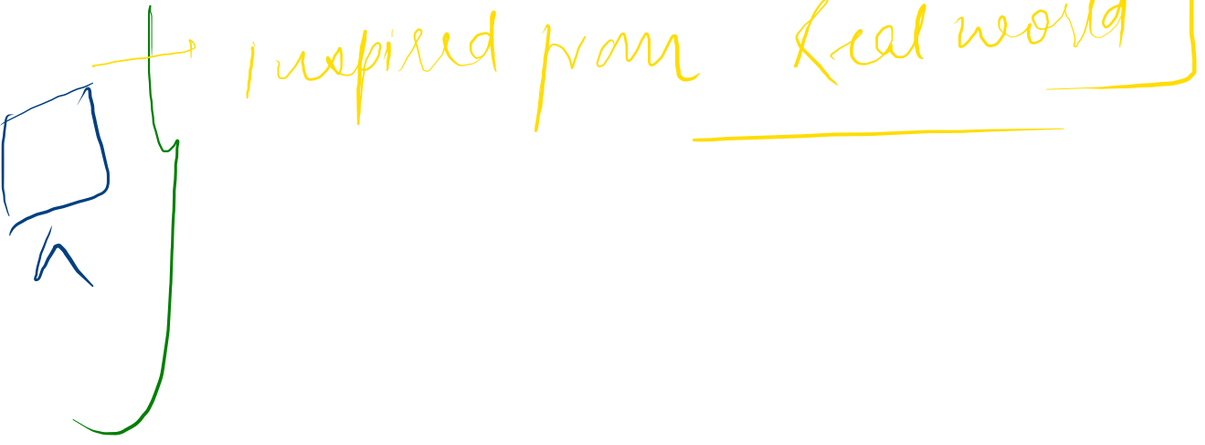
Day 3

1. Architectures
   - Monolithic
   - SOA
   - Microservices
2. Load balancer
3. Latency
4. Throughput

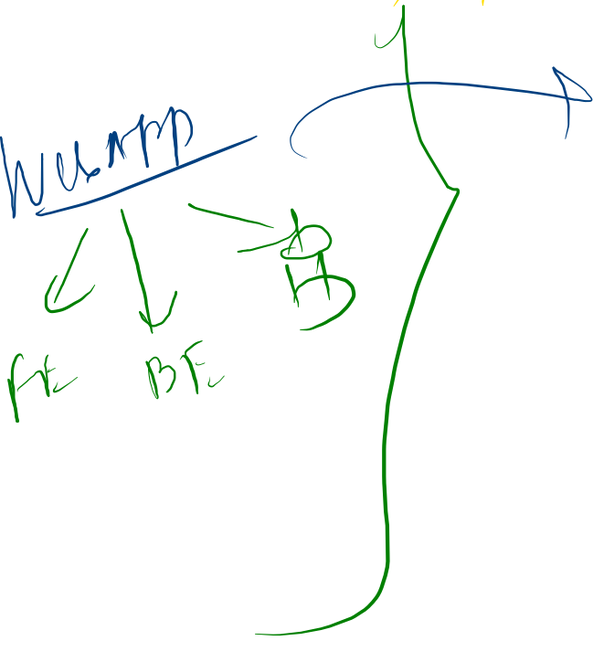
→ server and clients

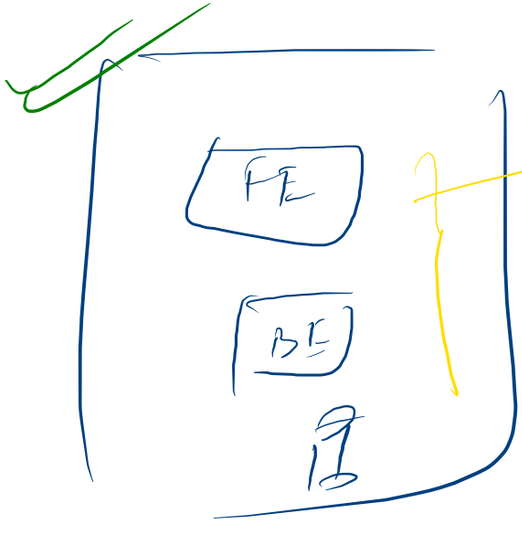

inspired from
real world

Monolithic system

Mono → single

Single Big system





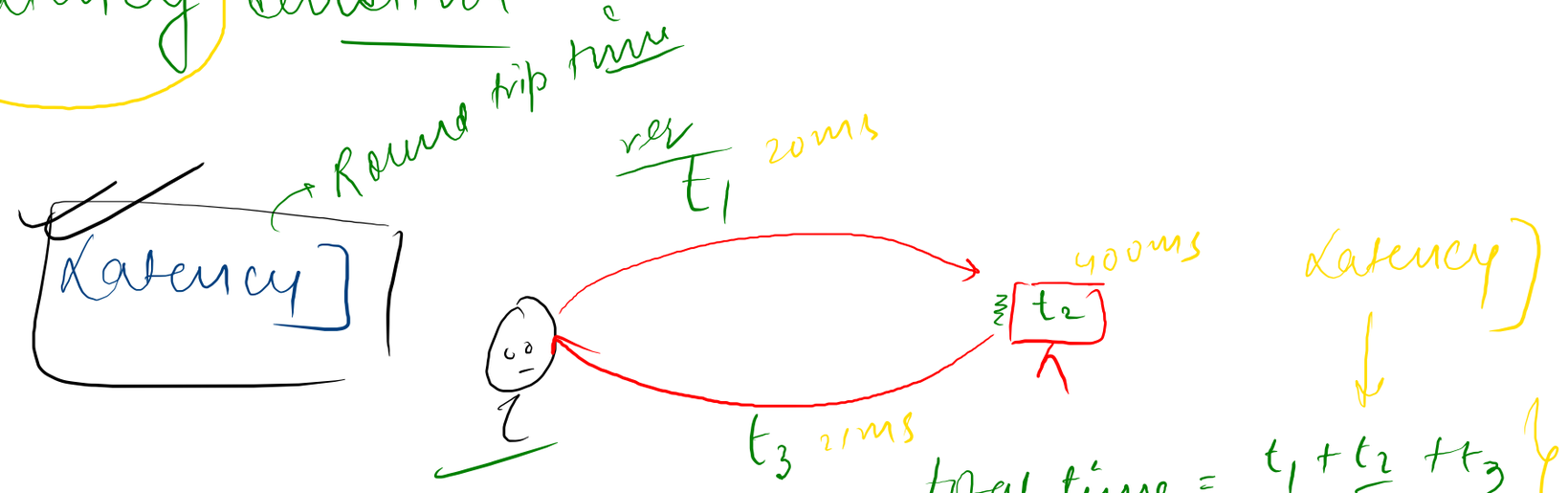
1. same code base
2. Machine
3. Deployed

Scenarios Monolithic / Advantages of monolithic systems

1. Just starting → easy, cost easy

Facebook, Paypal → Monolithic

(2) App → Latency sensitive



Latency

↓

total time = $t_1 + t_2 + t_3$

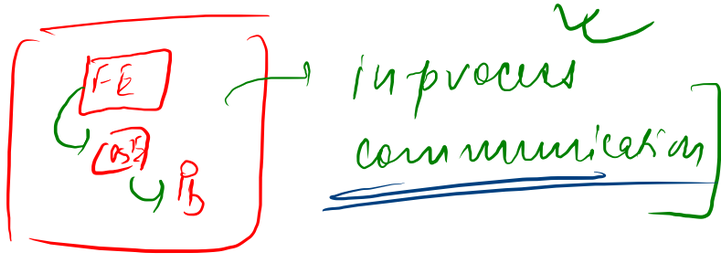
s / ms / MS / ms

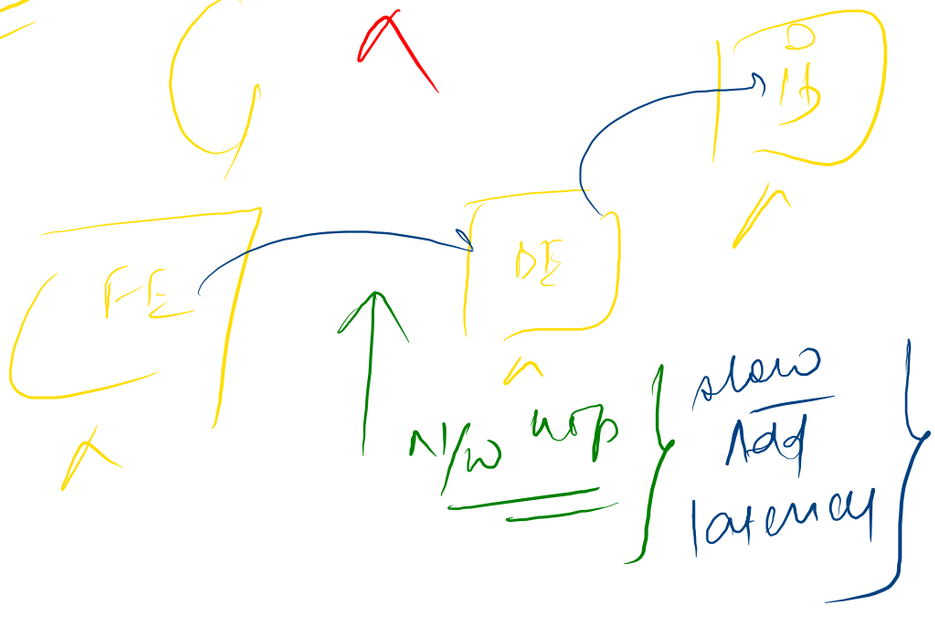
Latency = 20 ms + 400ms + 21ms

= 441 ms

(2) App → latency sensitive → Modularity.

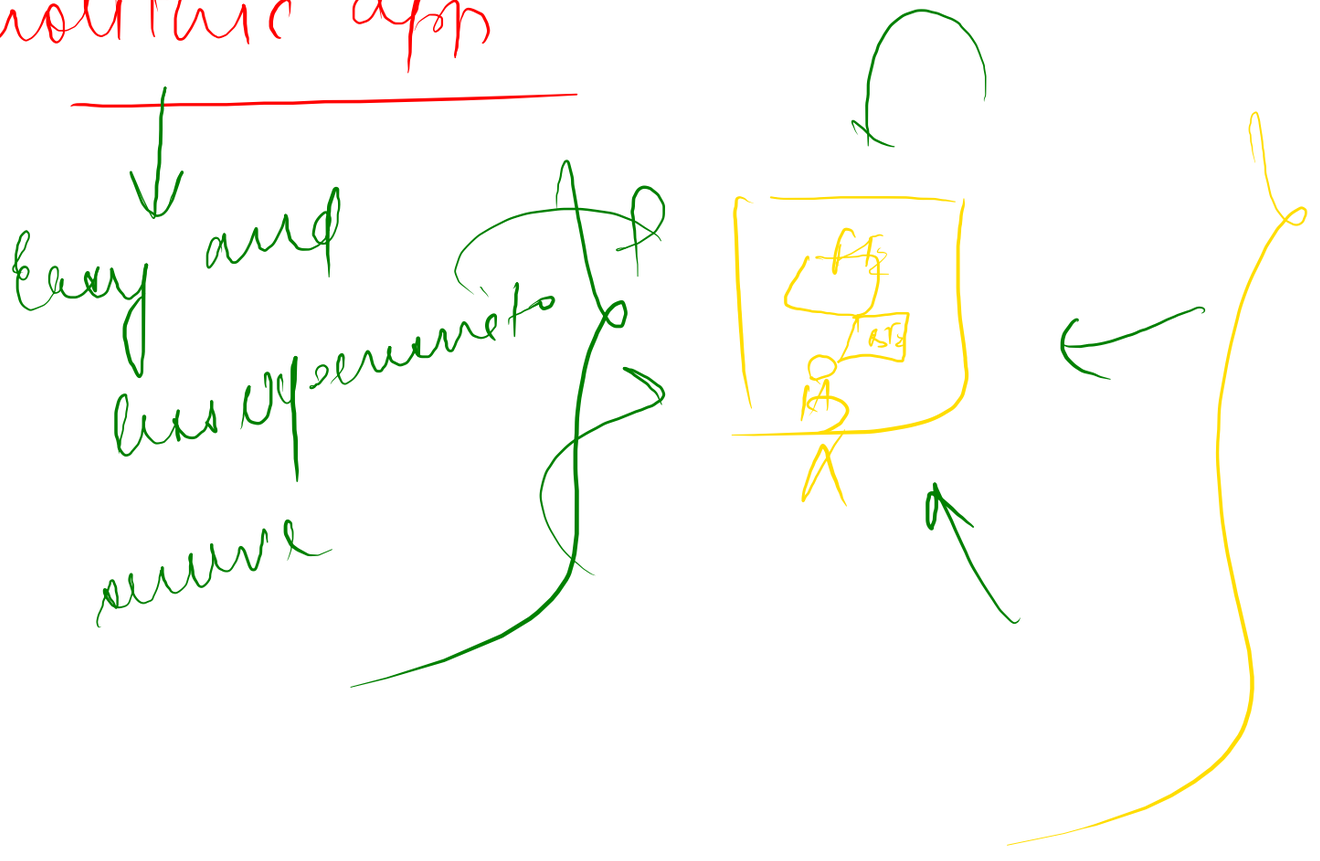
Tesla → 80% software → Modular Architecture

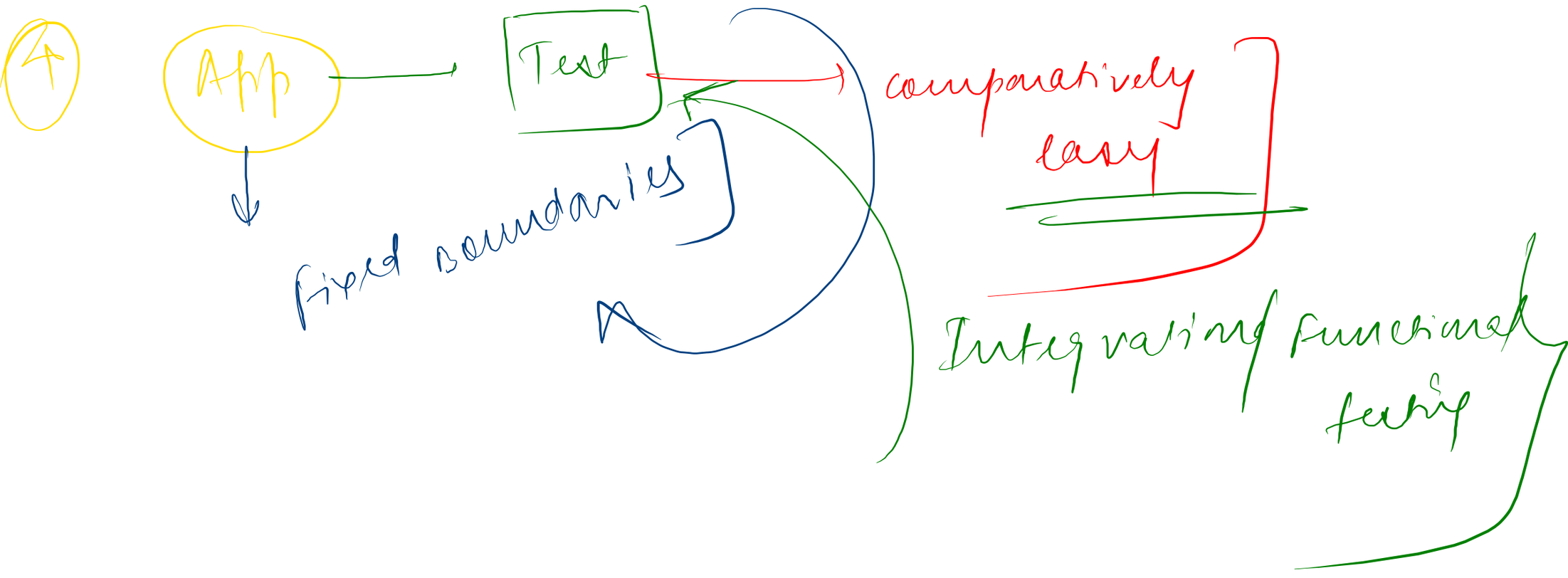


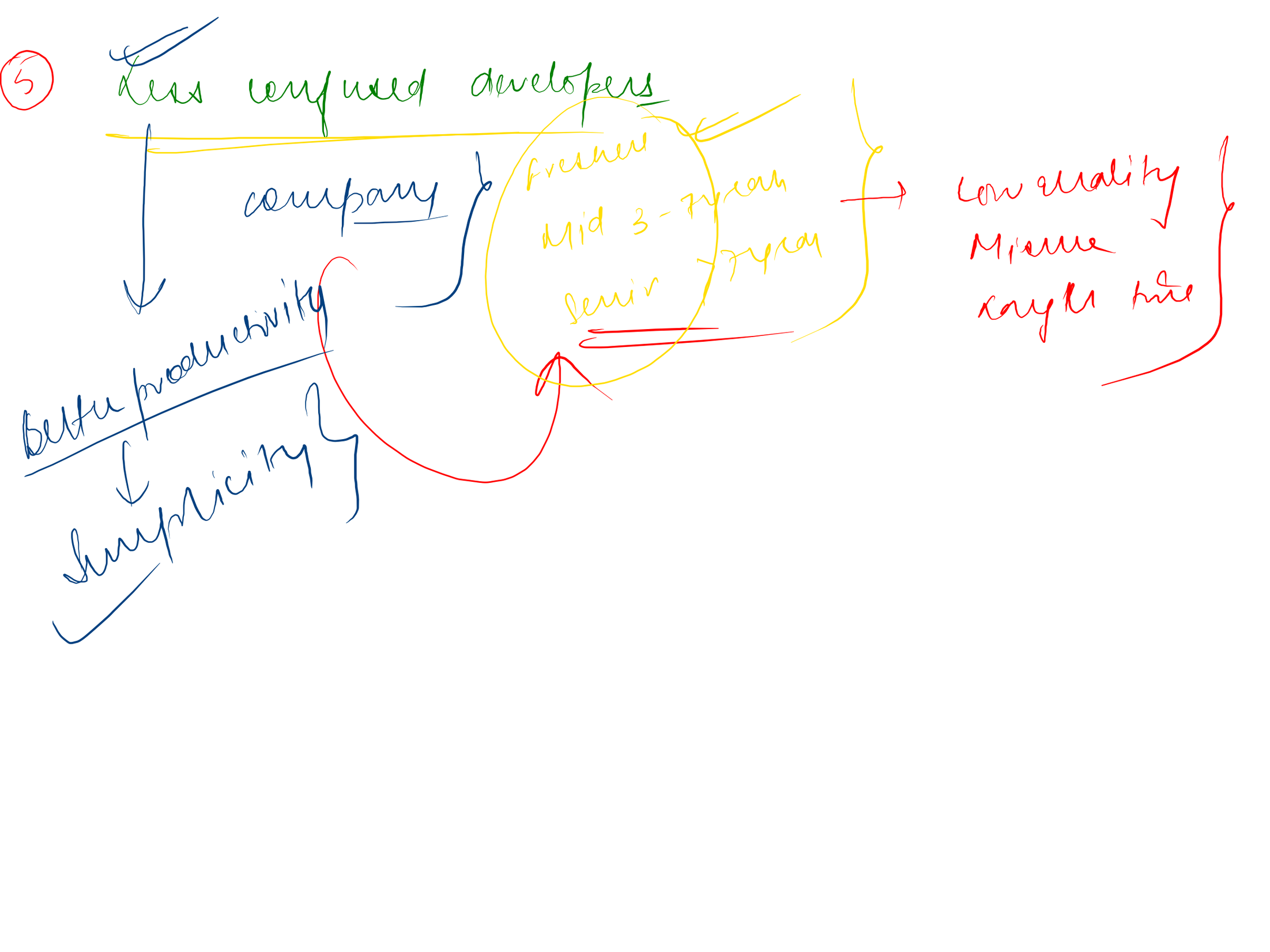


3

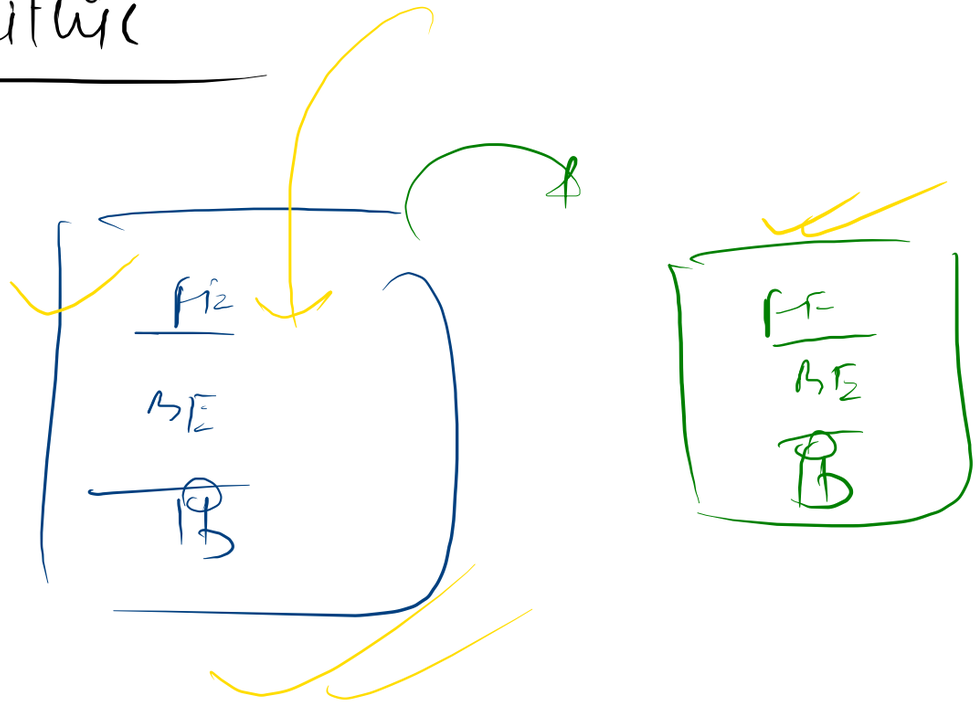
Monolithic app




4
App
Test
comparatively
easy
fixed boundaries
Integrating functionality


5
Less confused developers
company
Better productivity
Simplicity
Fresher
Mid 3-7 years
Senior 7+years
Low quality
Minimum
Long life time

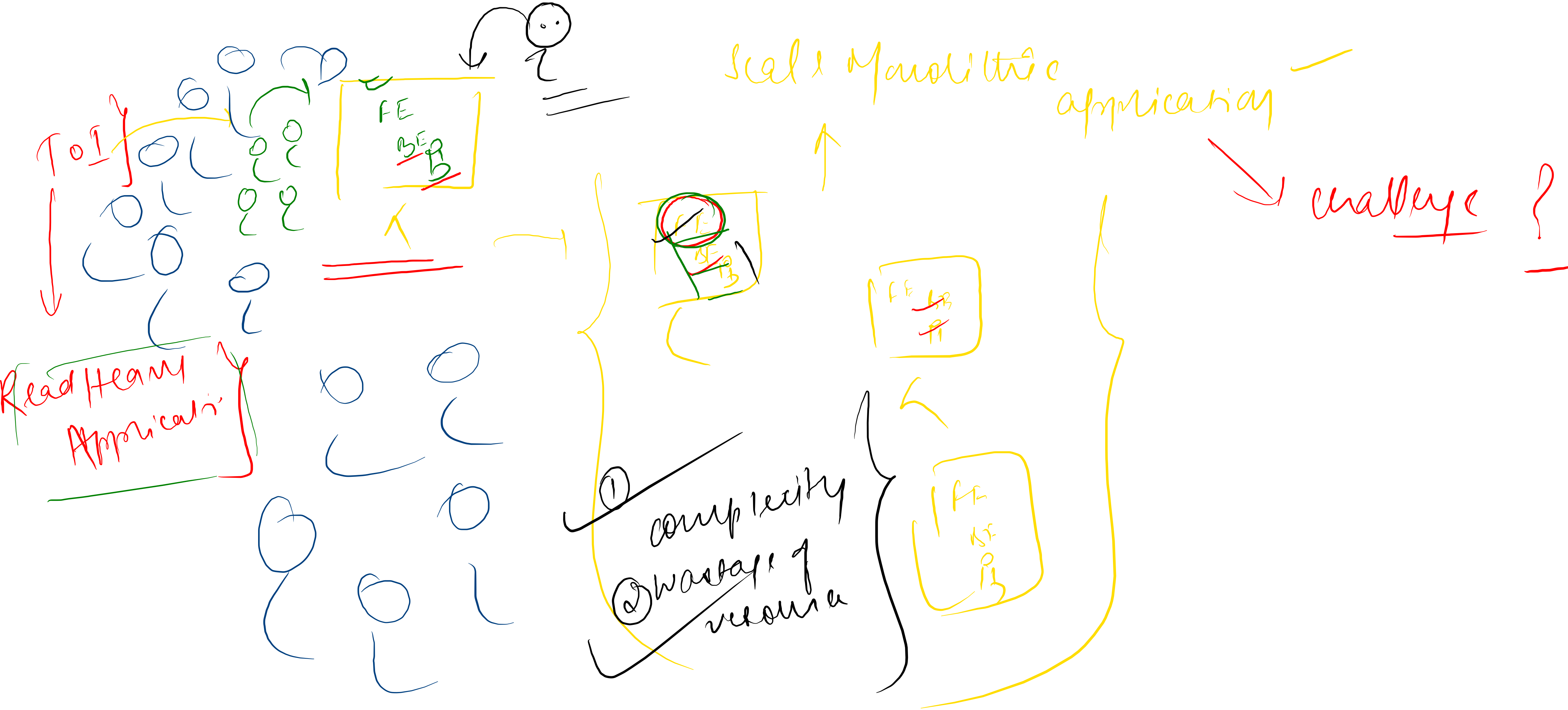
Monolityc

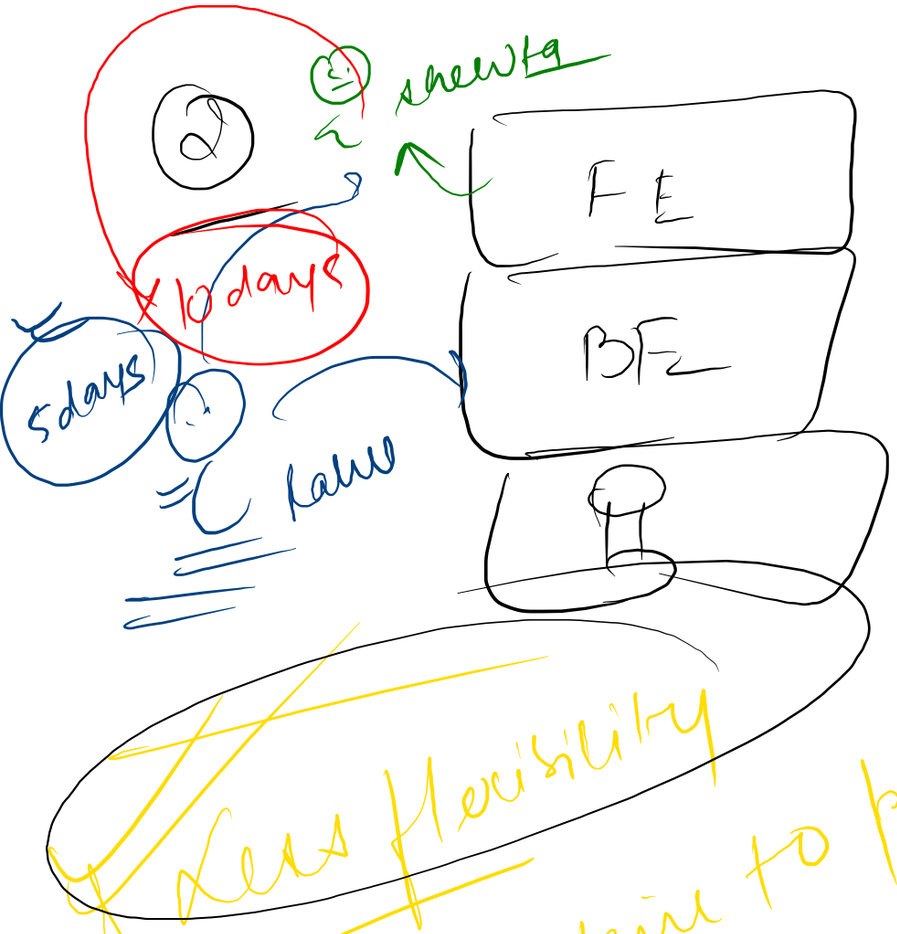


Challenges :-

① Scaling has it's complexity → can't do selective scaling

Can Monolithic applications be scaled ?


TOI
Real Heavy Application
FE
BE
DB
Scale Monolithic application
challenge ?
① complexity
② wastage of resource

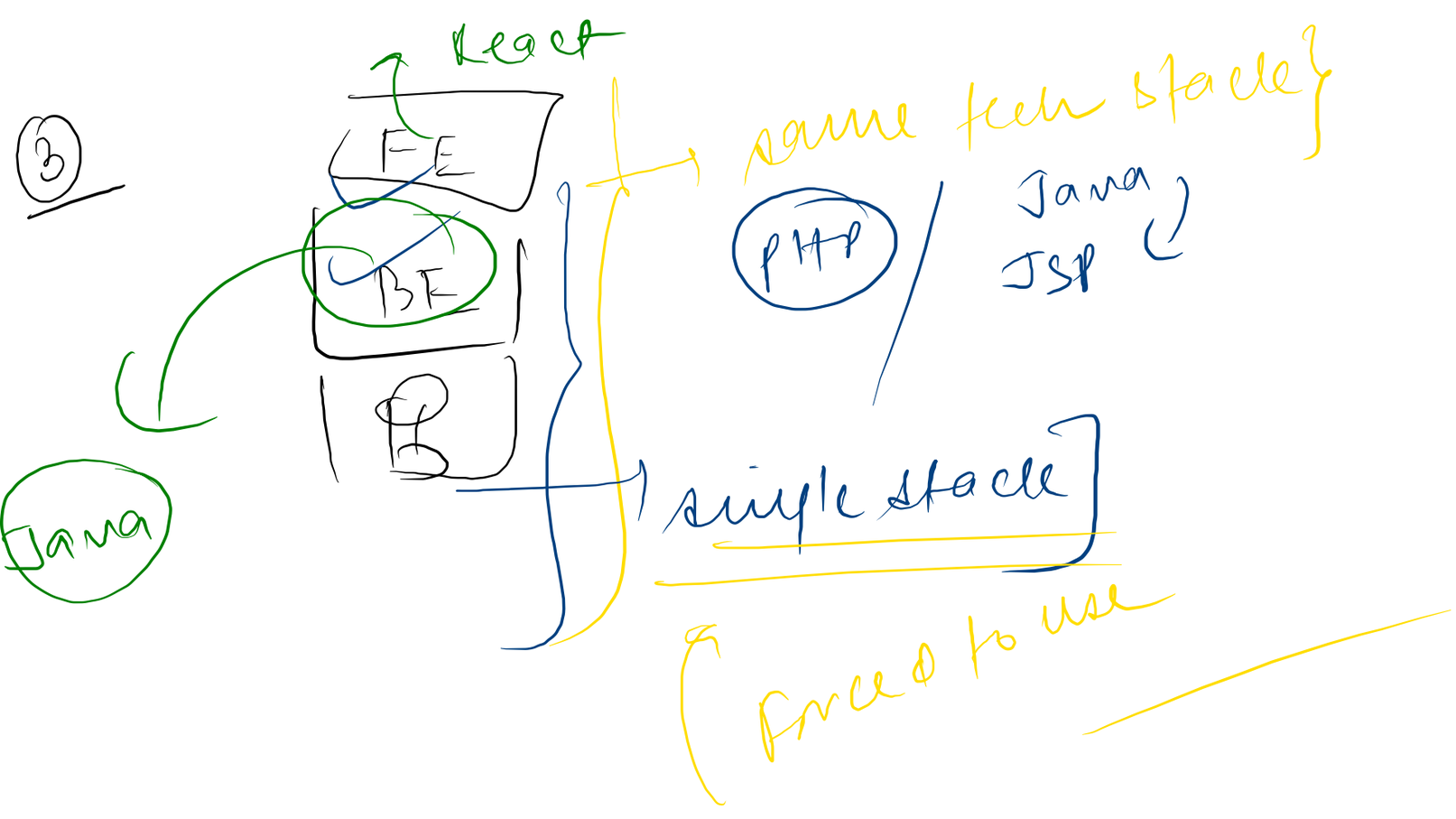


single repo

Less flexibility

Slower time to production

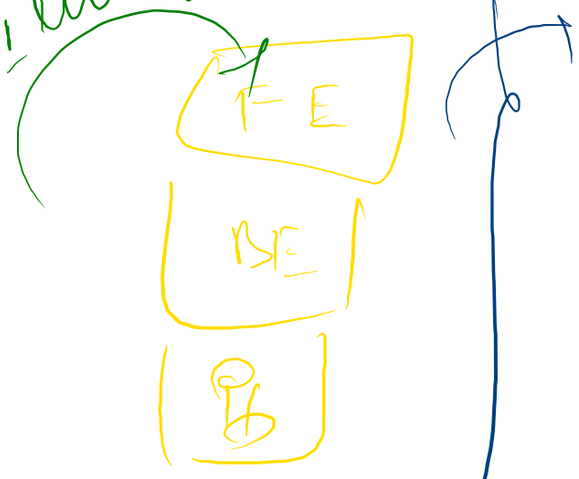
Human conflicts


3
React
FE
BE
DB
Java
same tech stack
PHP
Java
JSP
simple stack
Free to use

4

Deployment time ↑

Live of Process

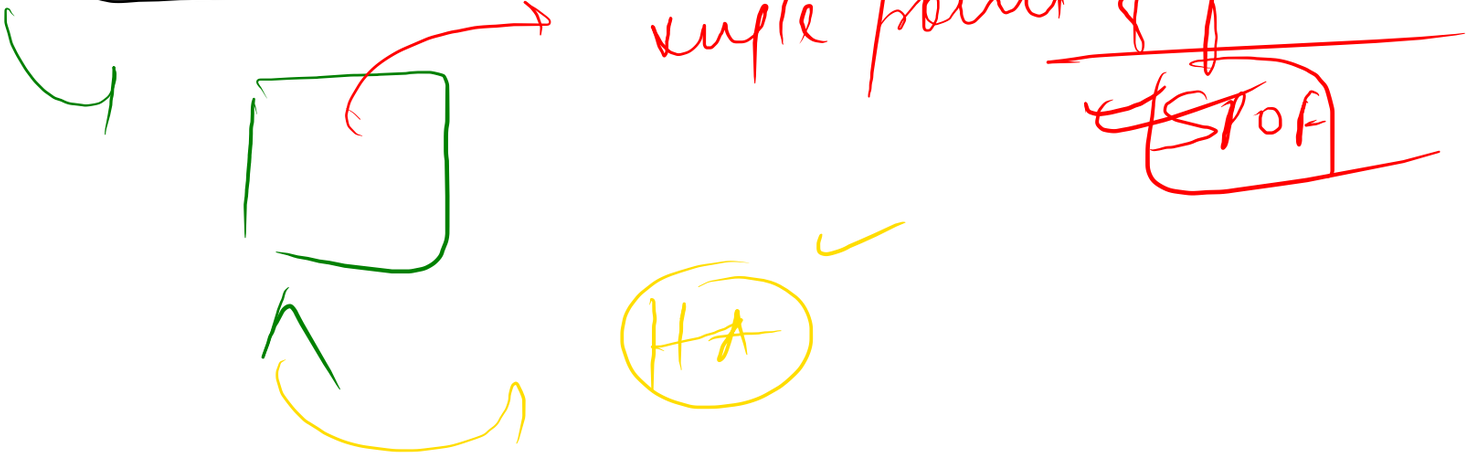


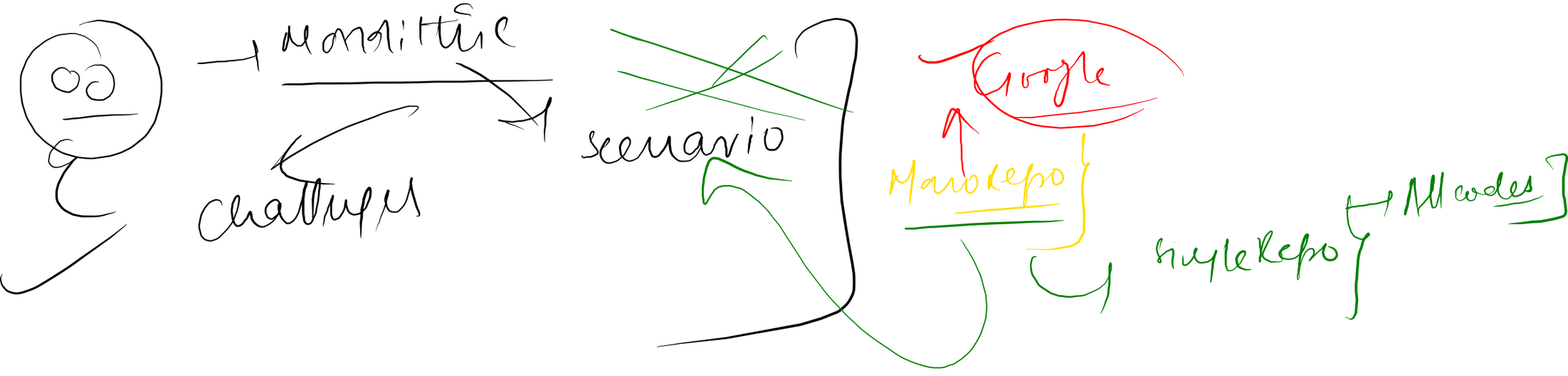
re-deployed

Deployment ↑

↓ Down-time ↑

(5) Everything is at one place




monolithic
scenario
Google
MavoRepo
SimpleRepo
All codes

Monolithic

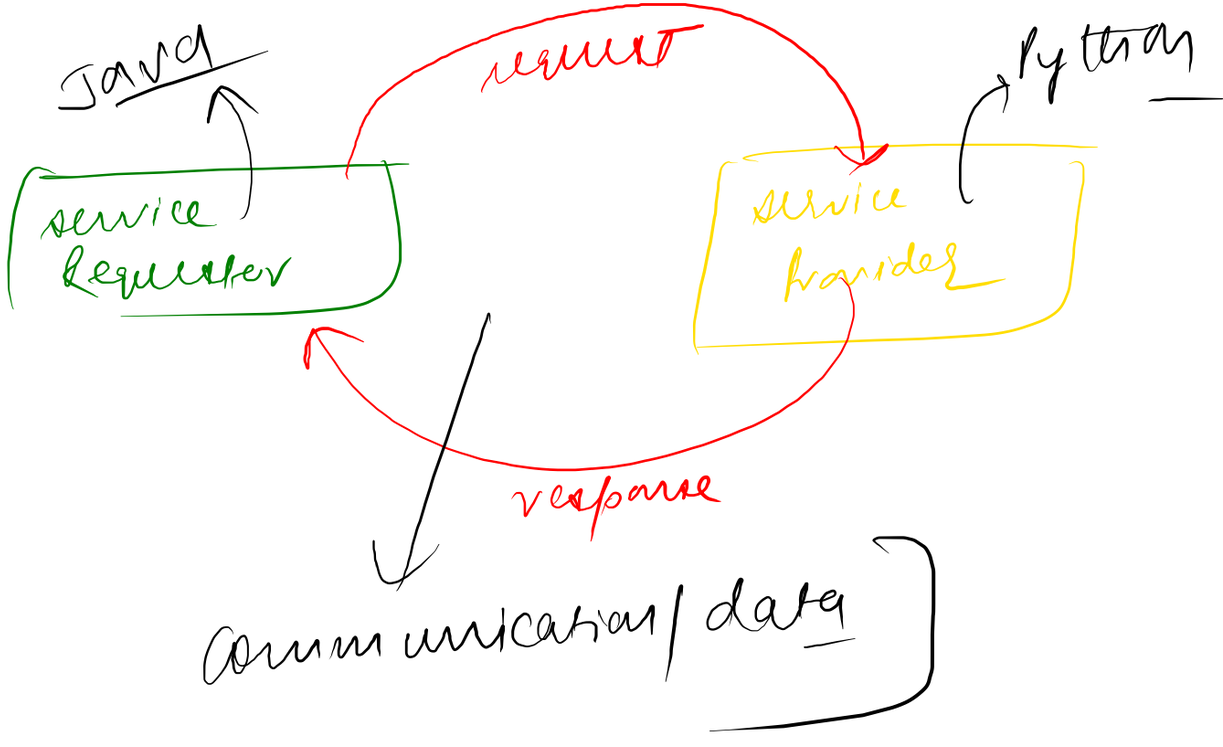
① Vertical scaling

② Fixed space

③ Less flexibility

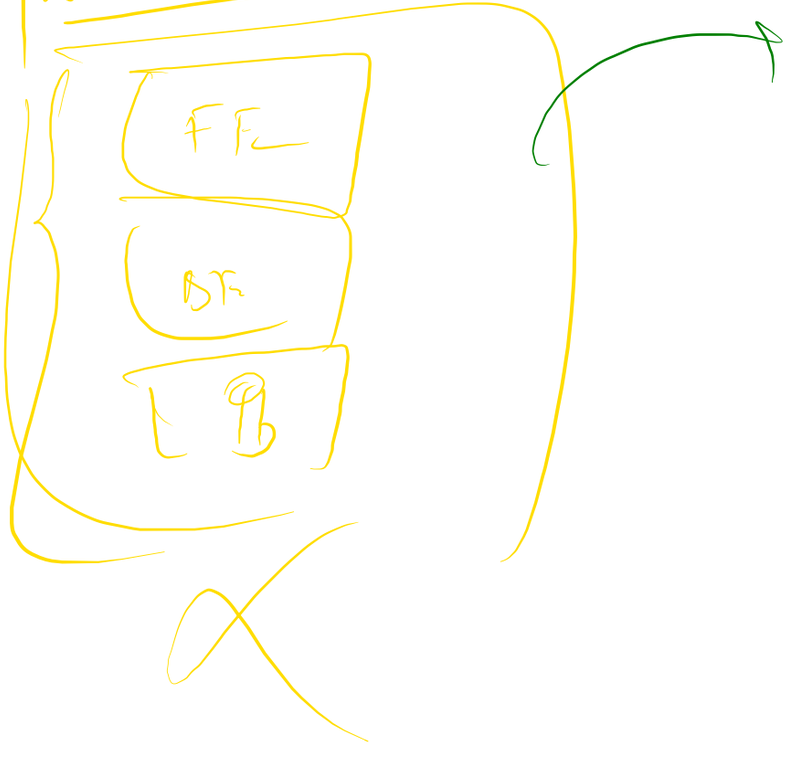
SOA

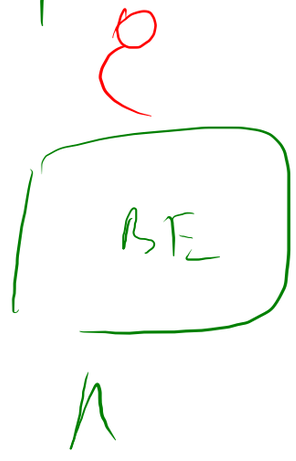
↳ Service Oriented Architecture

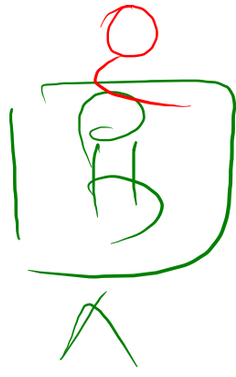


classic SOA → As Granular system as possible

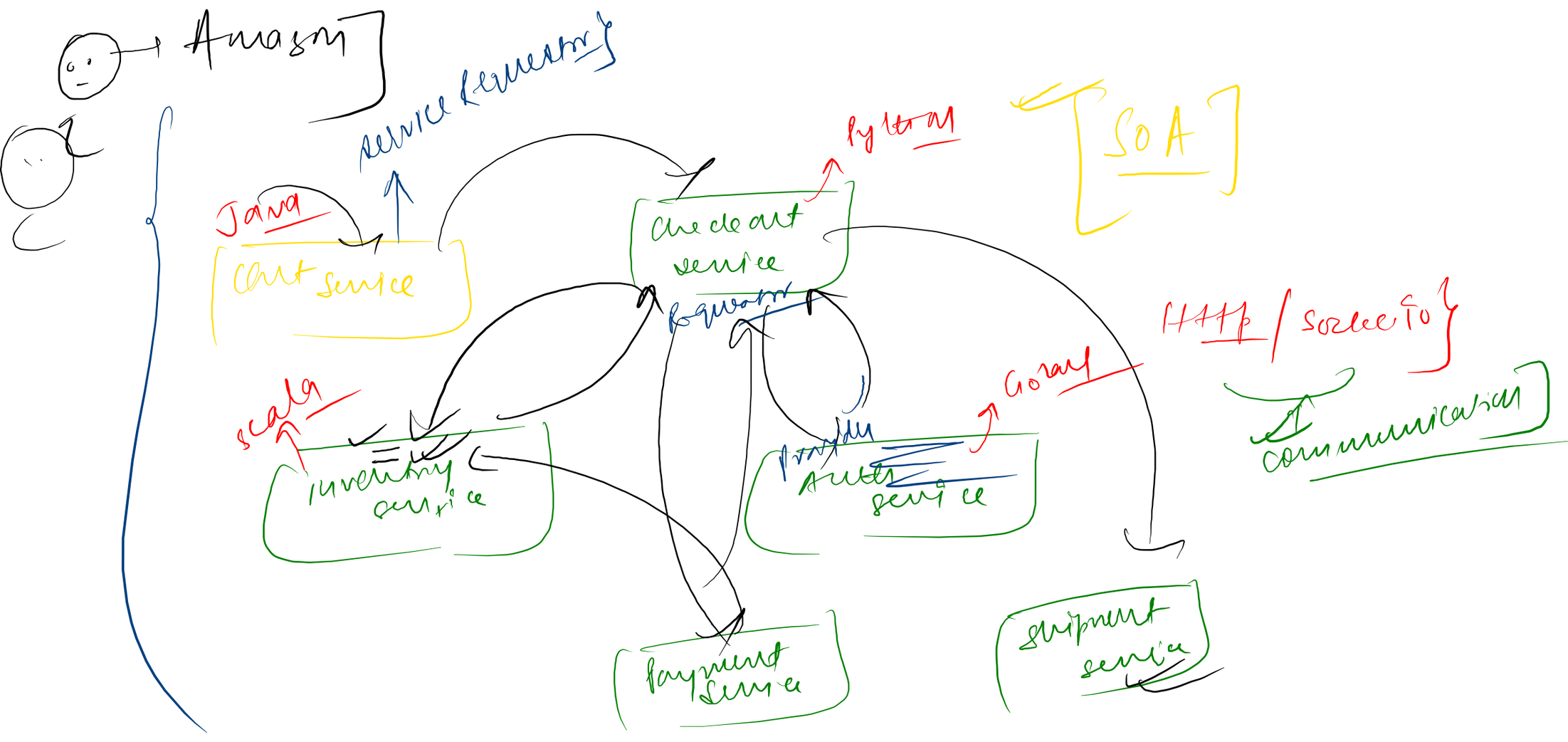
Monolithic







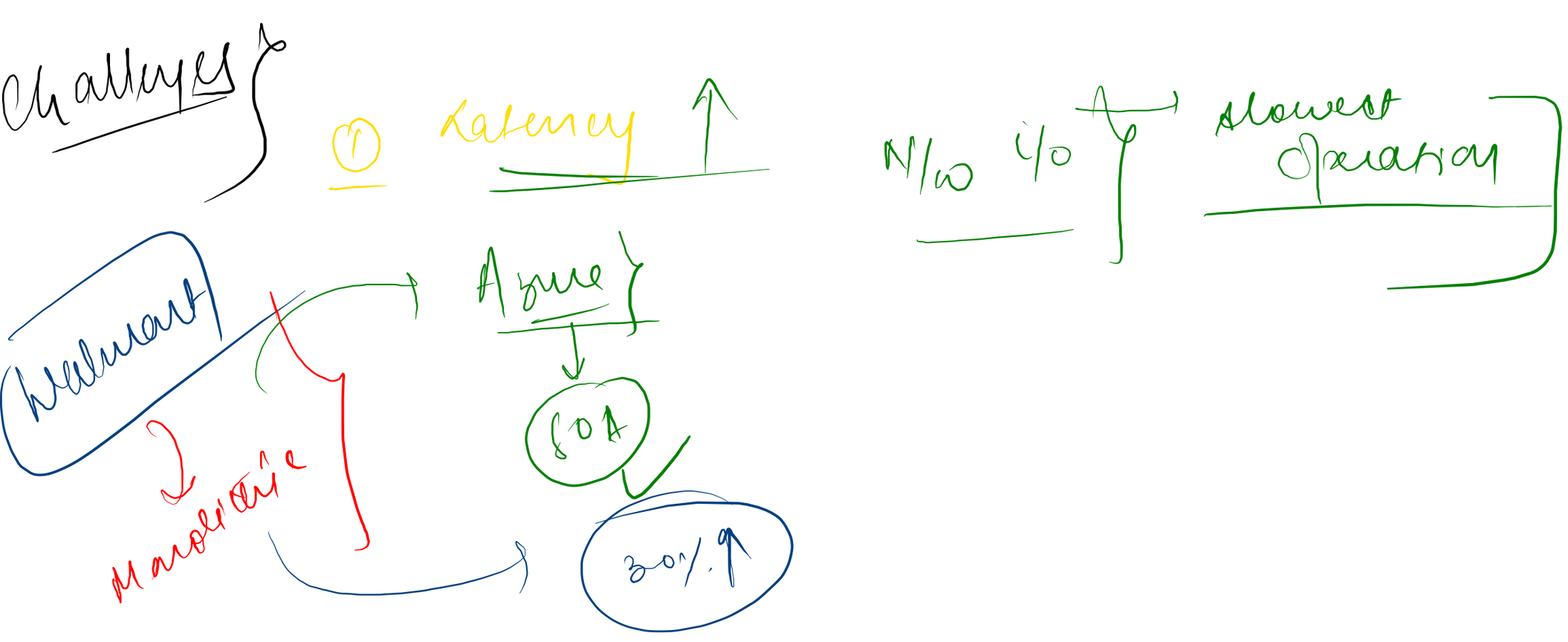
Granular

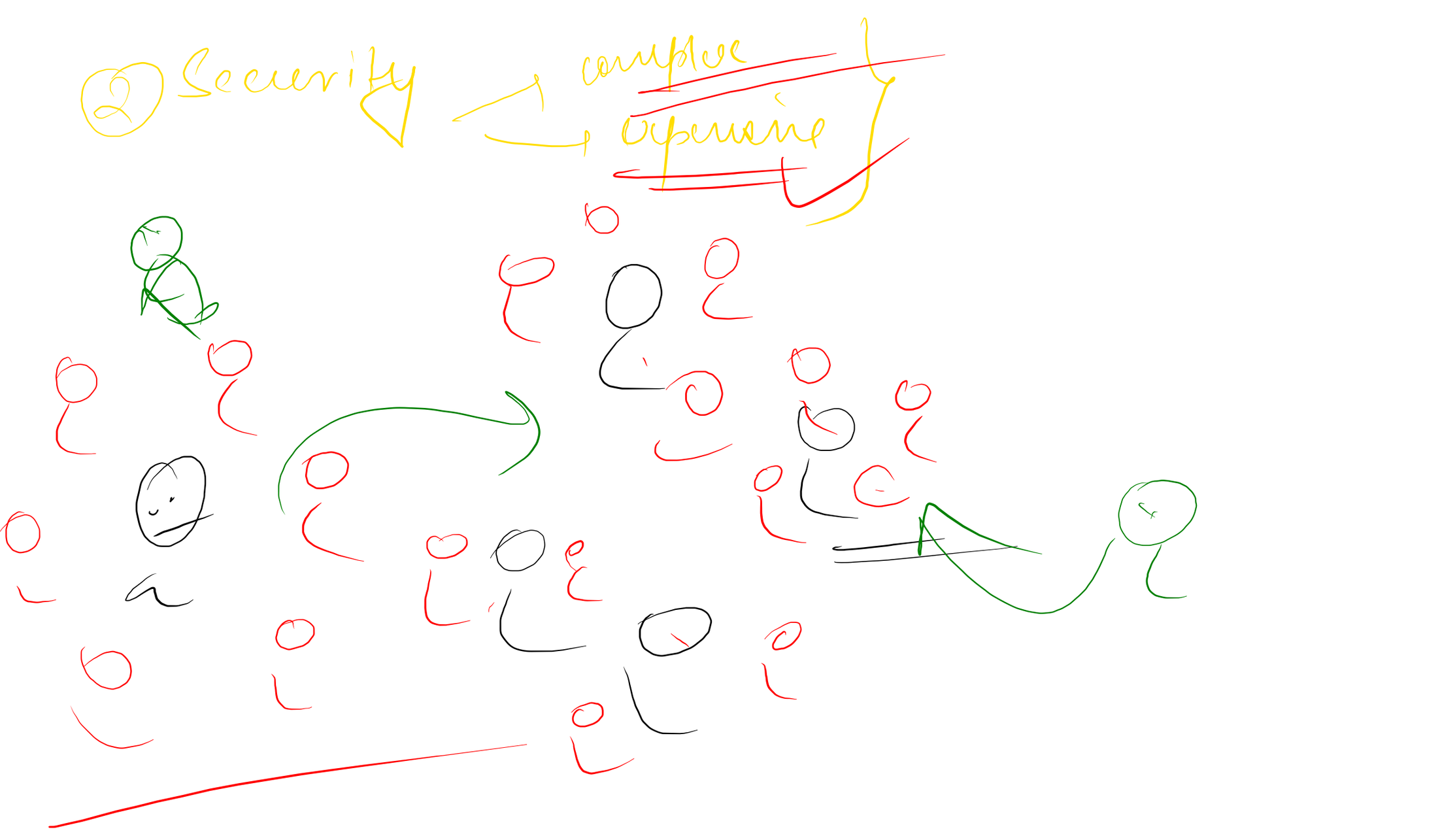

Amazon
service discovery
Java
Cart service
Checkout service
Python
Golang
HTTP / socket
communication
SOA
scala
Inventory service
Payment service
Shipment service

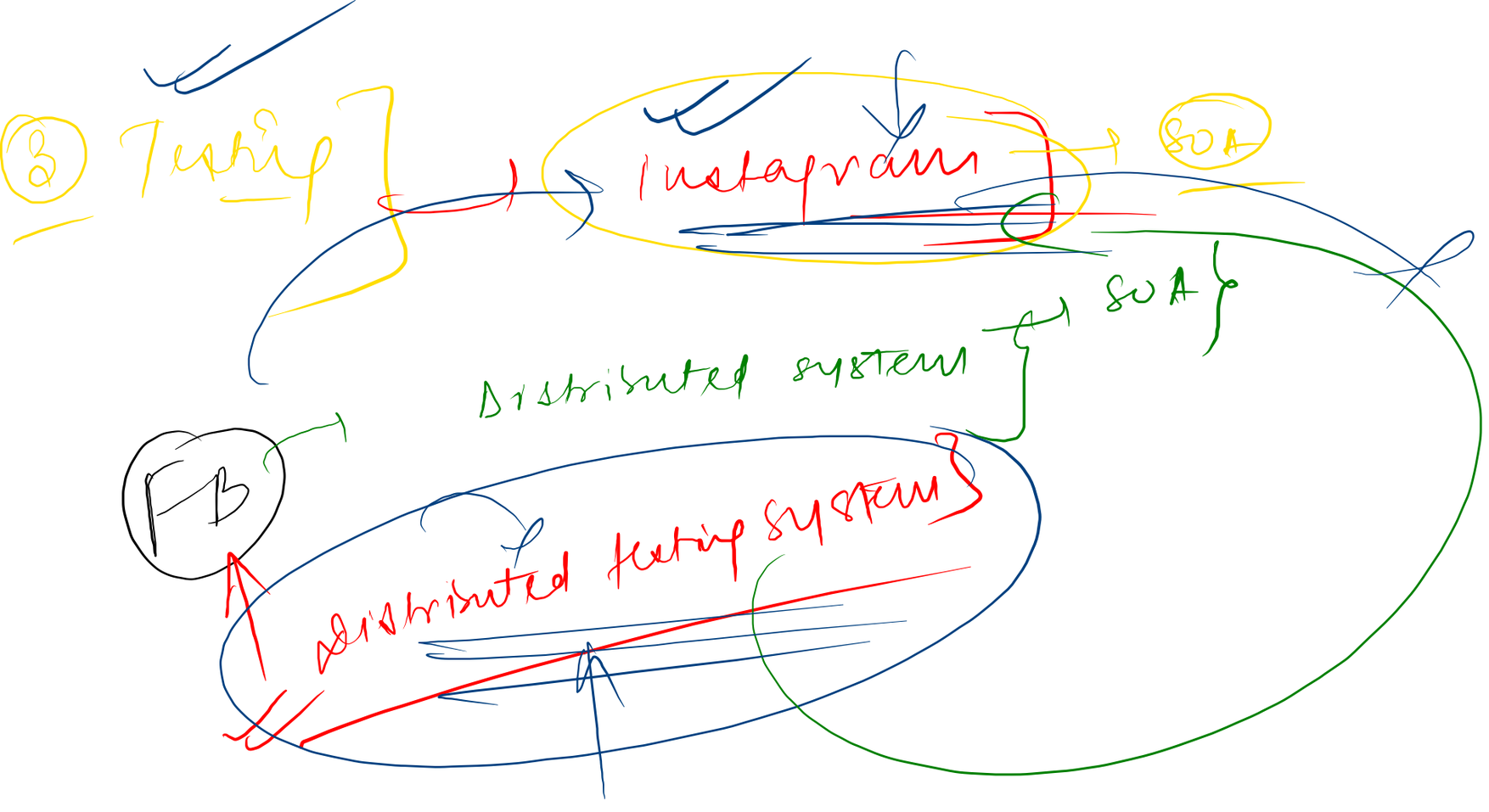
SOA → When to use SOA / Advantages

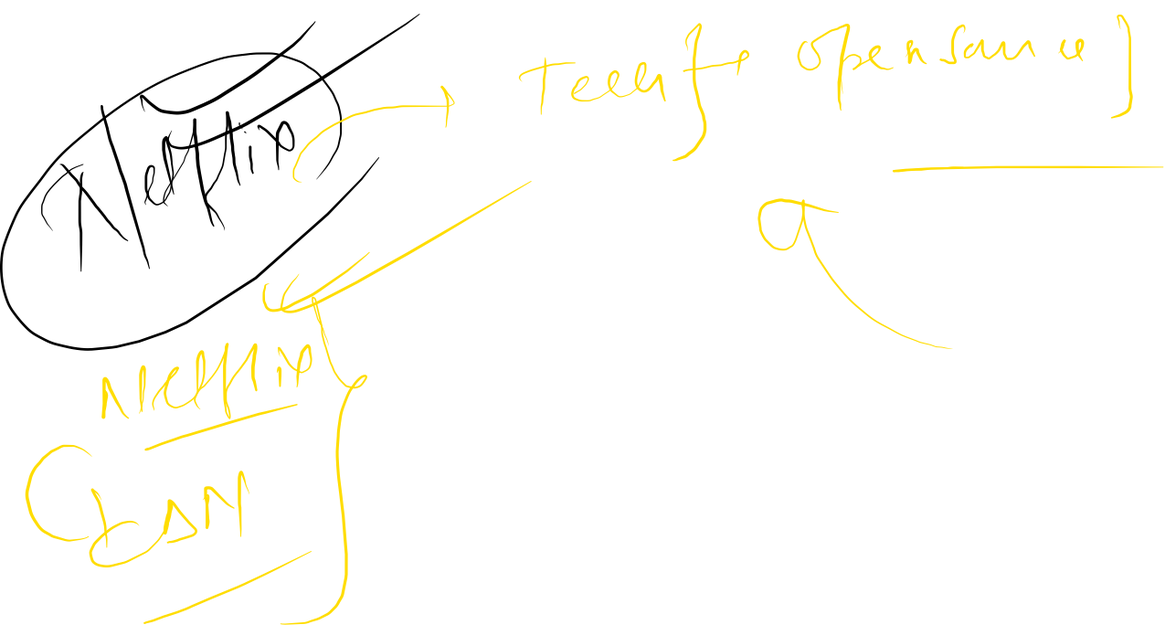
1. Selective scaling
2. Flexible to use multiple tech stacks
3. Loosely coupled
   - → Flexibility ↑
   - → Time to production ↓
4. Change in one component doesn't impact other
5. Deployment ↓
6. No SPOF

Distributed work

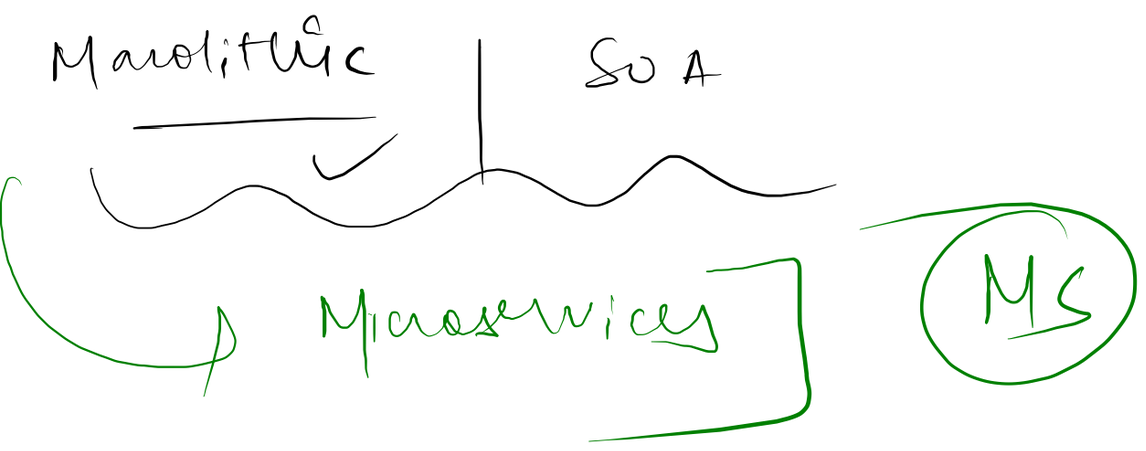

Challenges
1
Latency
N/w i/o
slowest
Operation
Network
Latency
SOA
30%

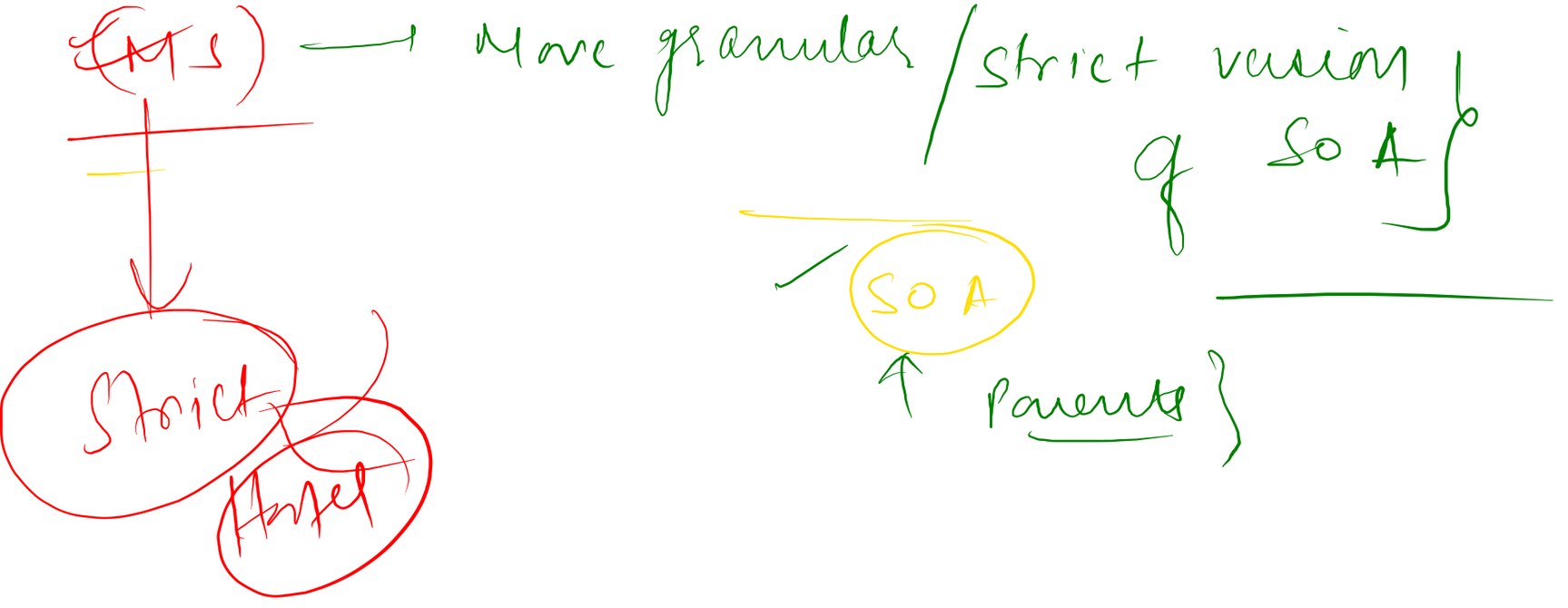

2 Security
complex
operating

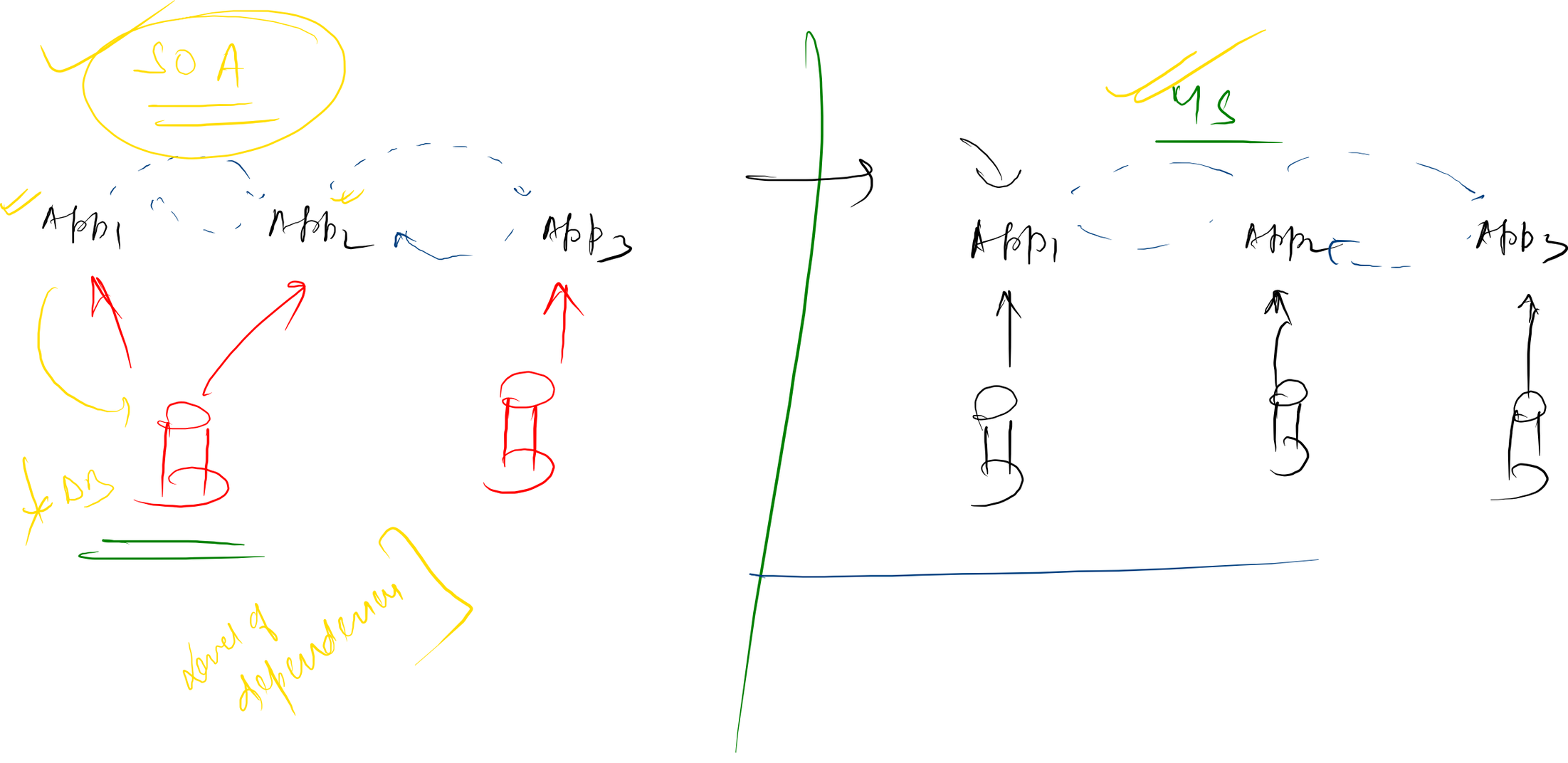

Twitter
Instagram
SOA
SOA
Distributed systems
FB
distributed testing systems

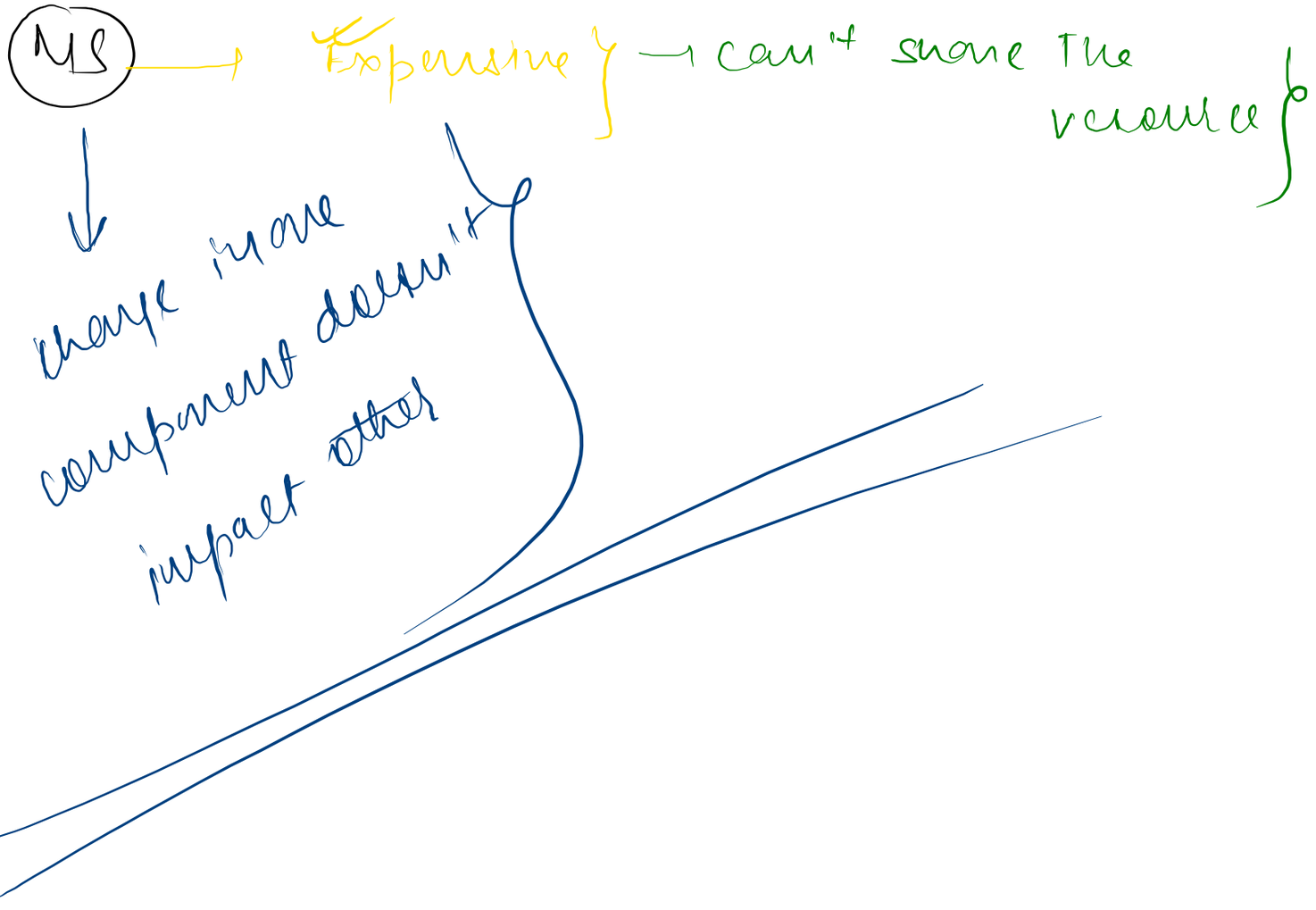


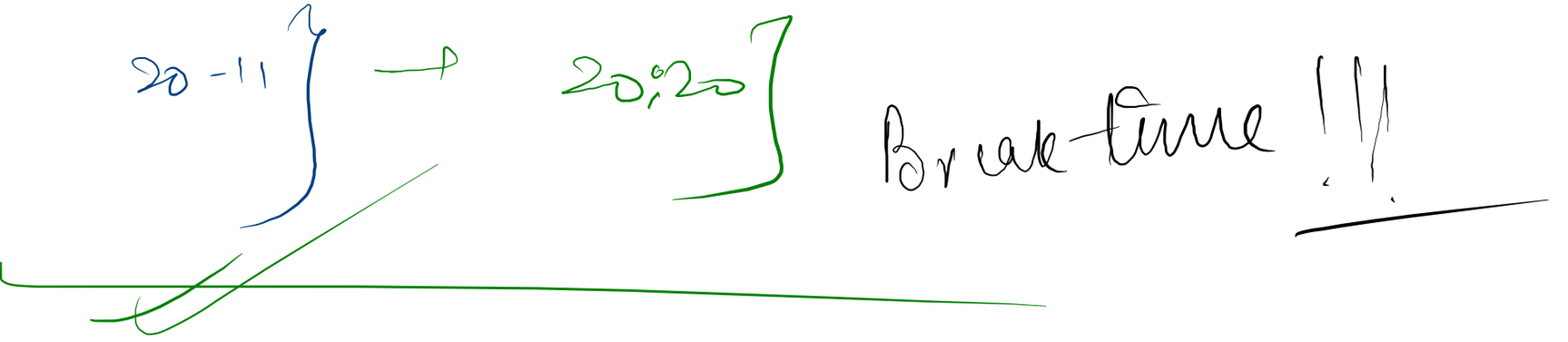
4 confused developers


Monolithic
SOA
Microservices
MS

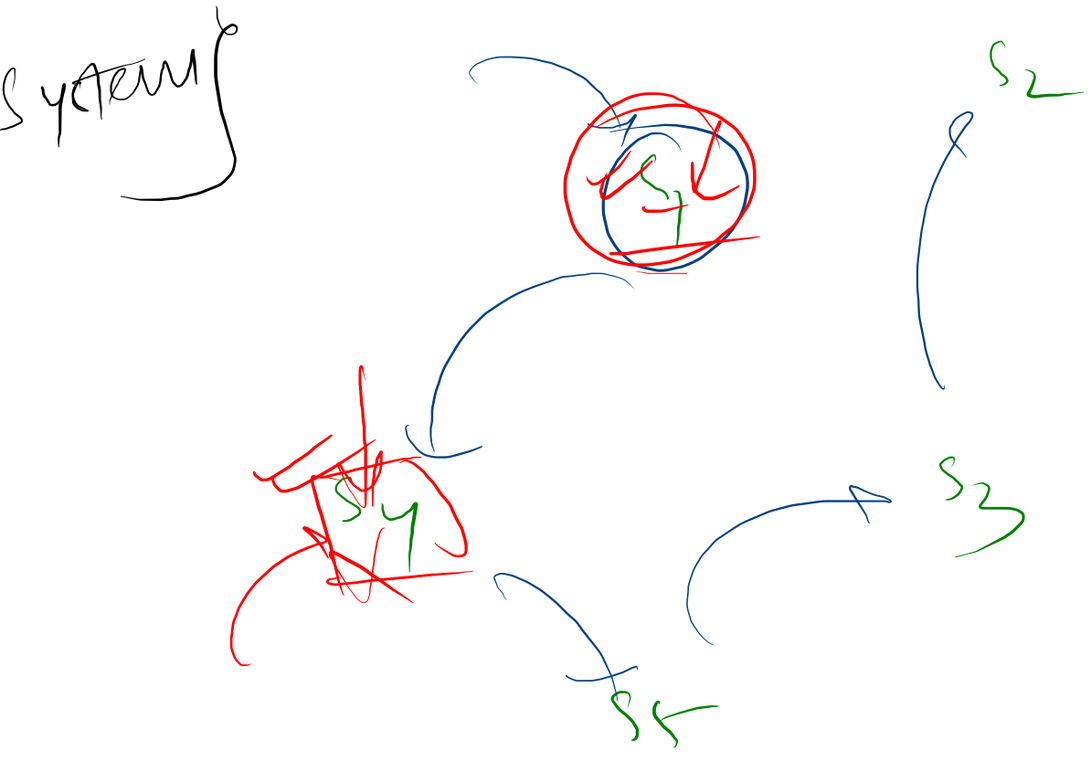

(MS)
More granular / strict version of SOA
SOA
Parents
Strict
Agile

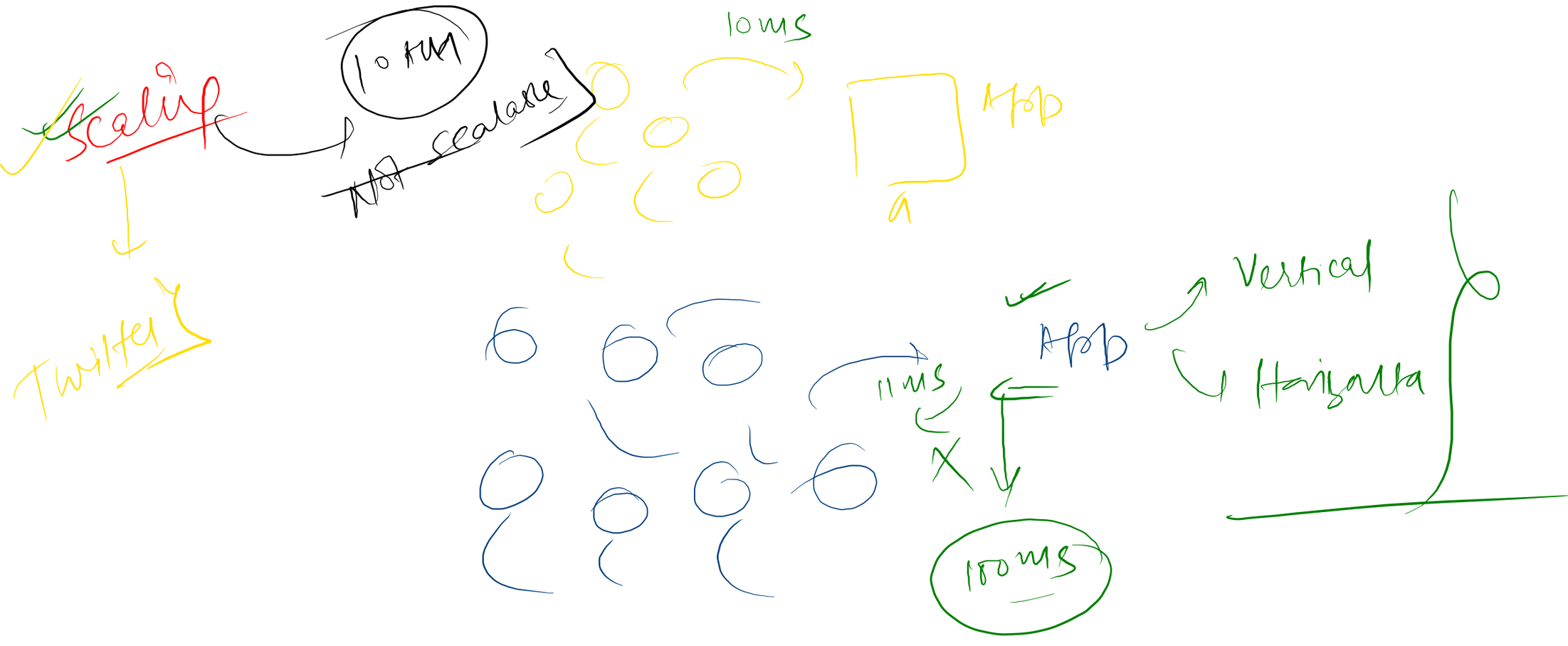

SOA
App1
App2
App3
DB
level of dependency
App1
App2
App3


MS
Expensive
can't share the resource
change in one component doesn't impact other



Break time !!!

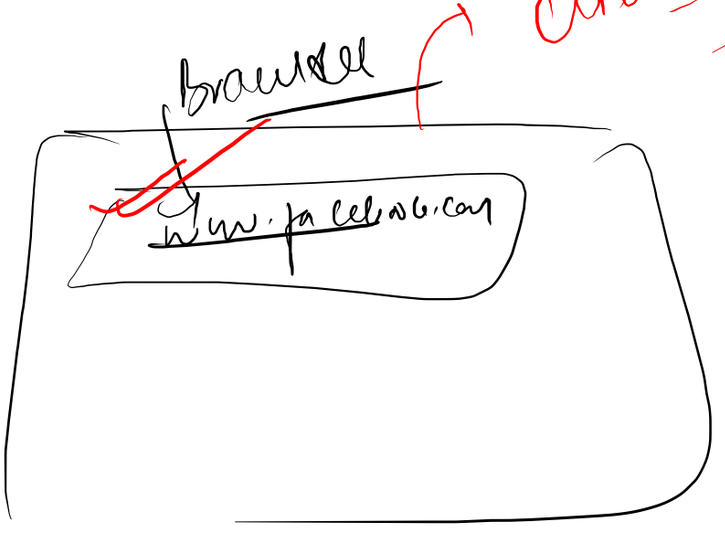

S1
S2
S3
S4

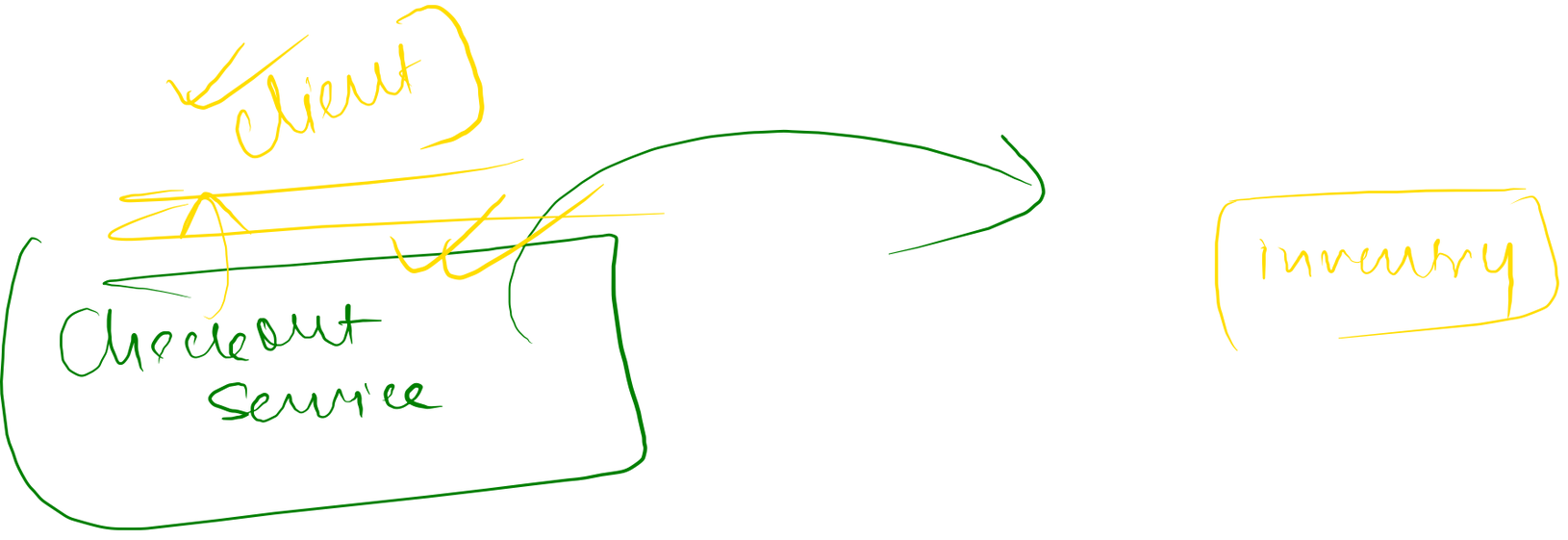

Scaling
10 RPM
Not scalable
10ms
App
Twitter
11ms
App
Vertical
Horizontal
X
180ms

Client and Server

[Client] Remote

[Client]

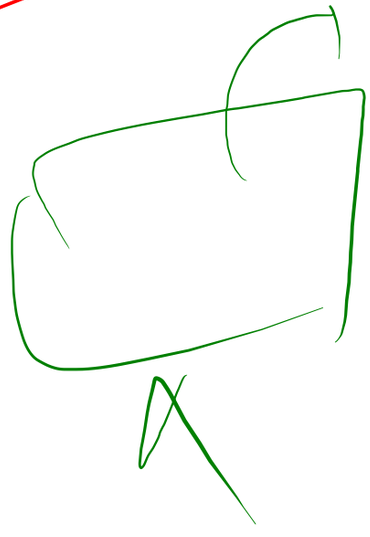


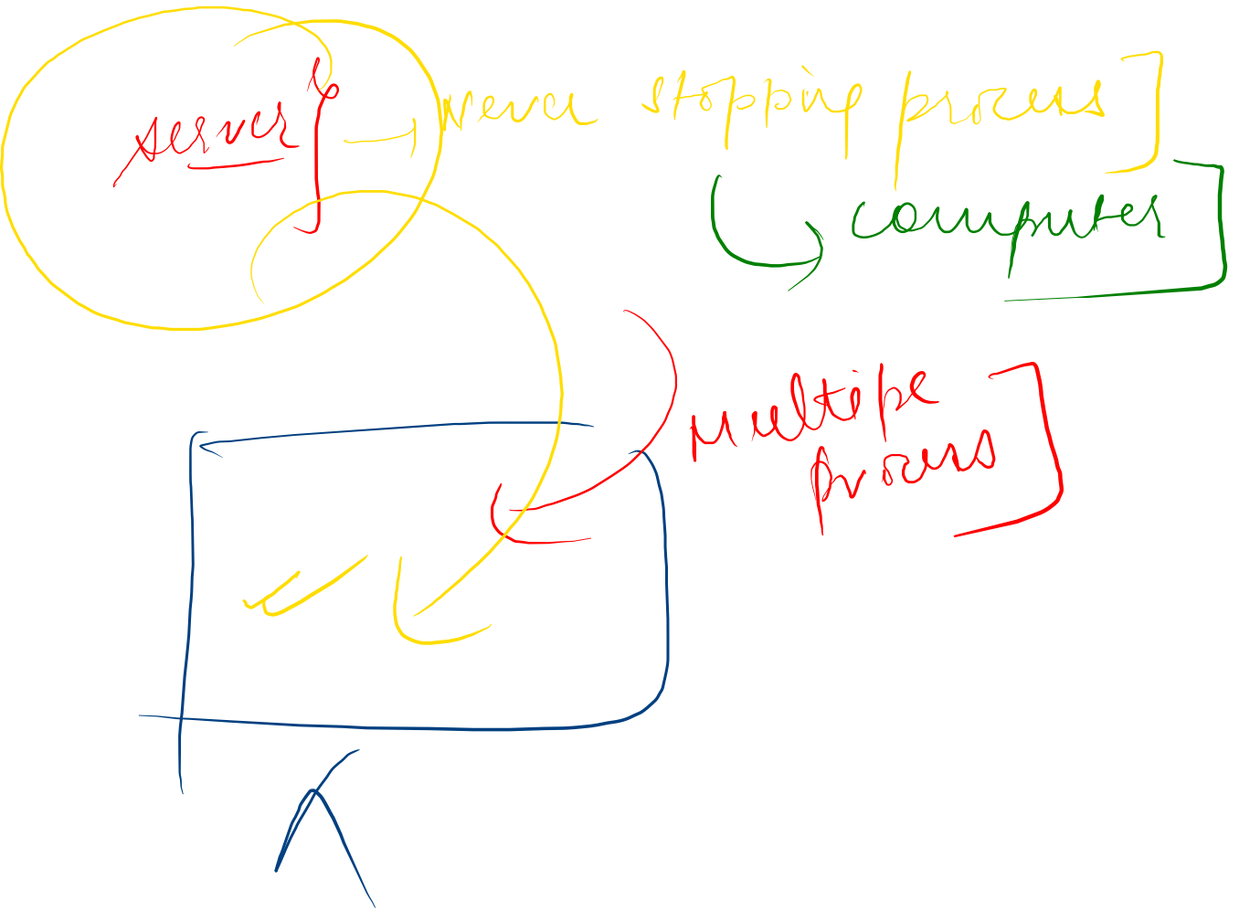

Client
Checkout
Service
Inventory

Server

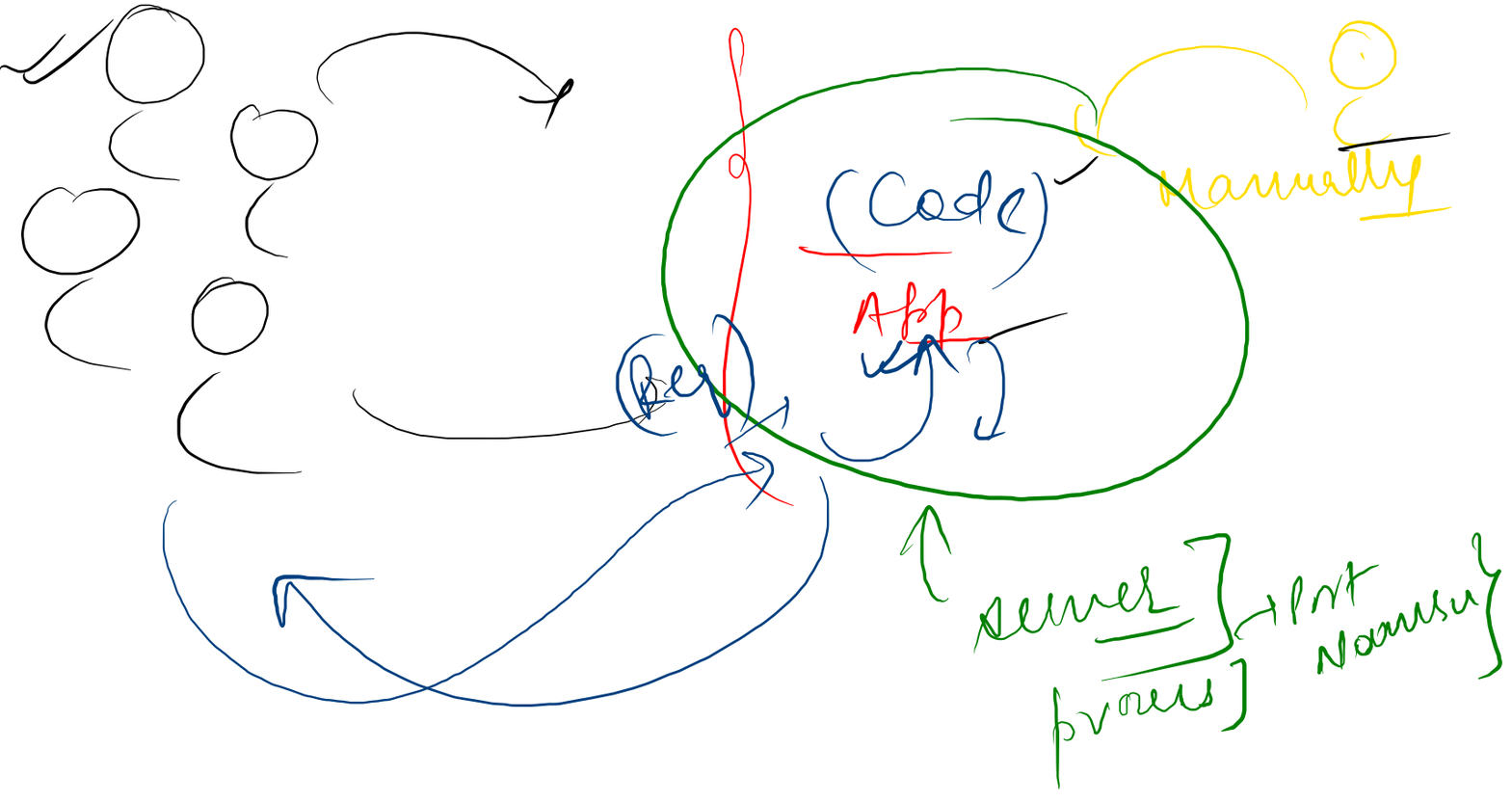
Meaning of Server !

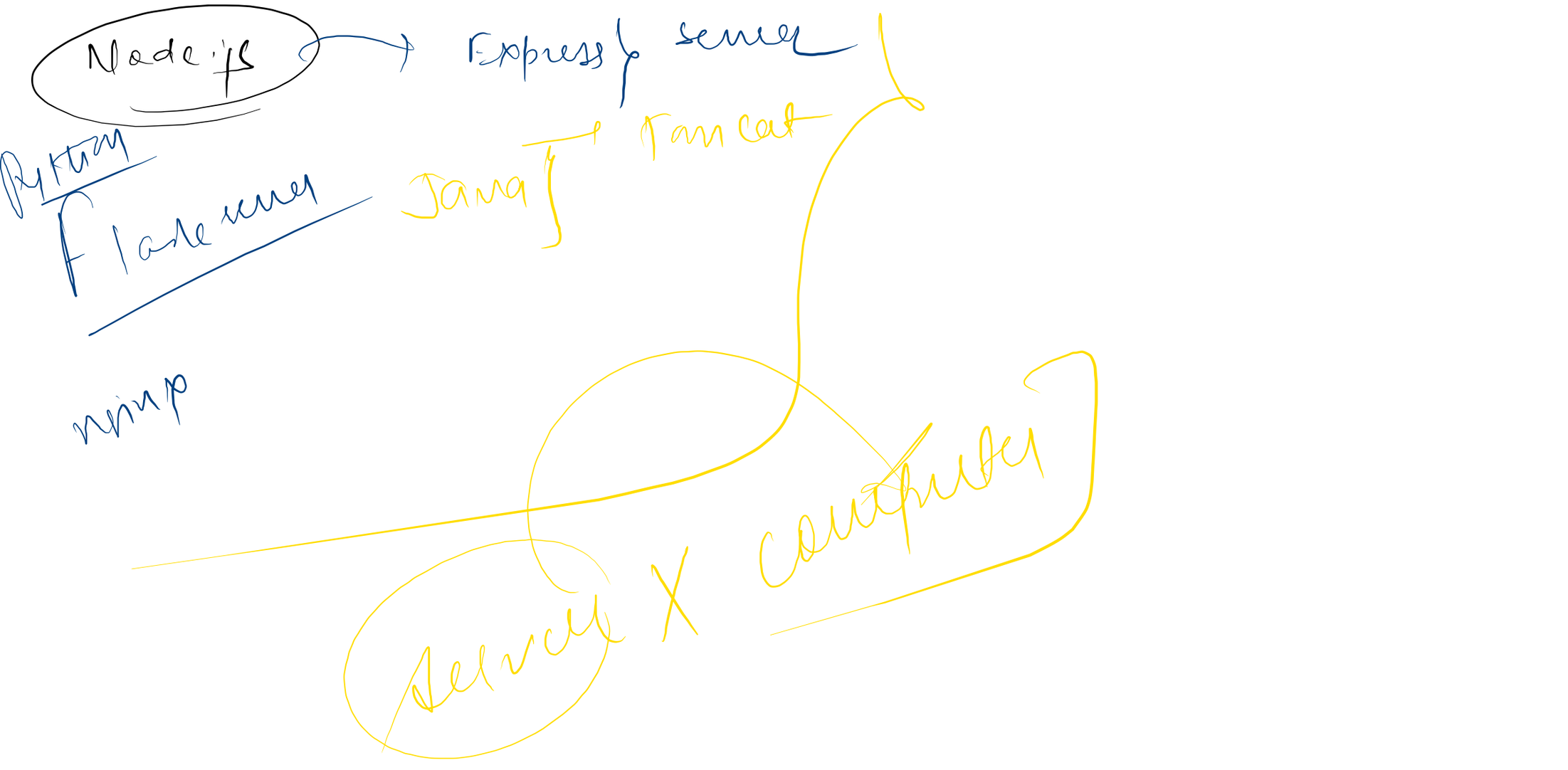
computer is a server ✗

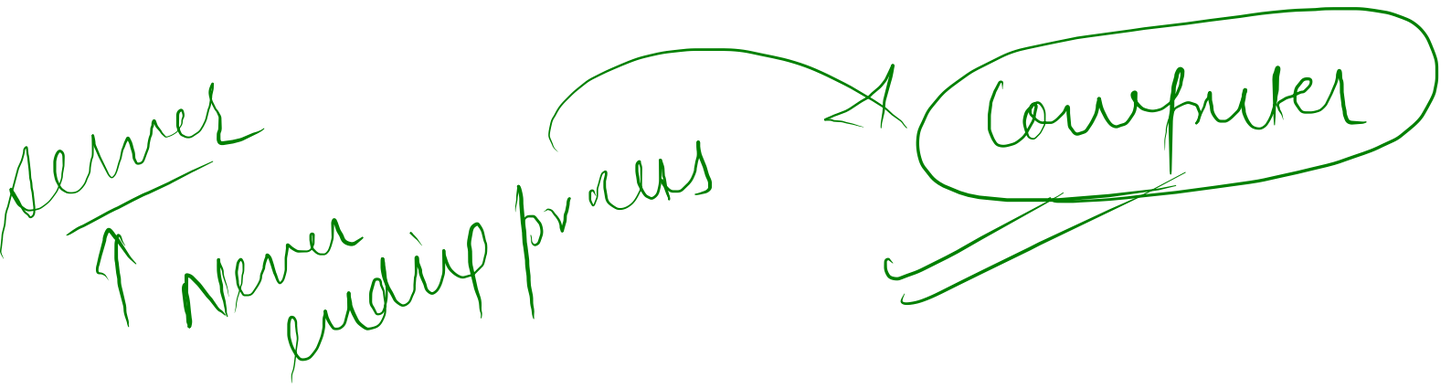


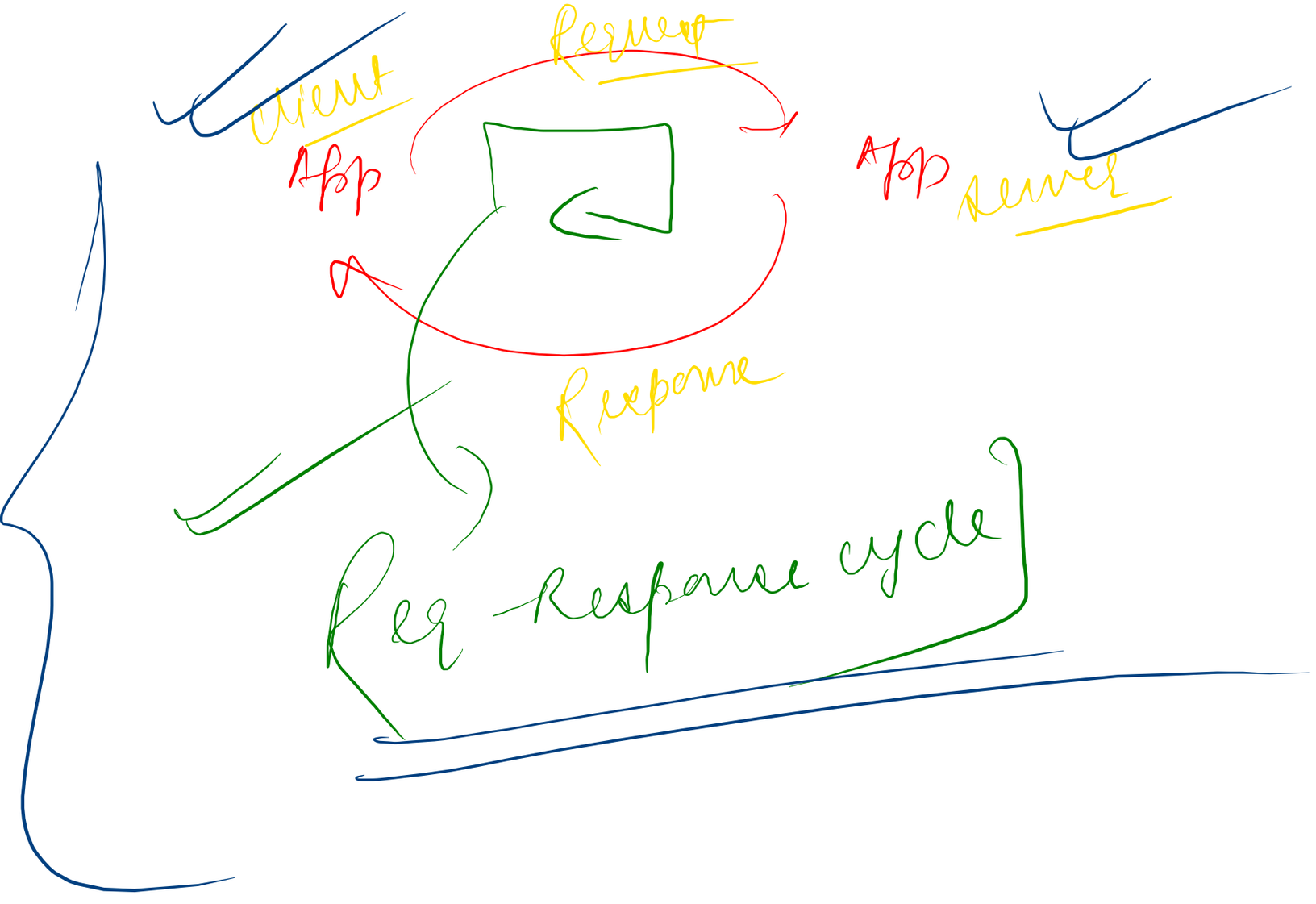

server
never stopping process
computer
multiple process

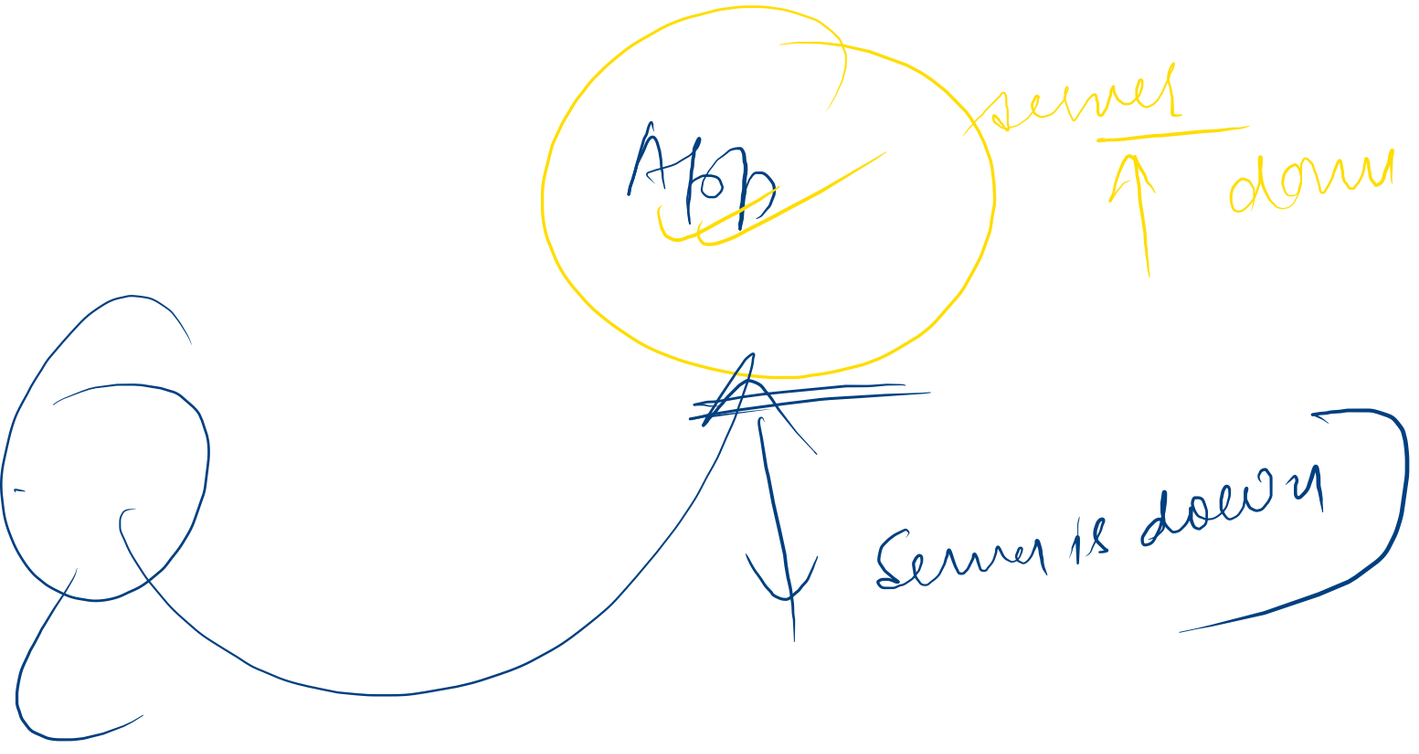

Python


(Code)
manually
App
Req
server
process
Port
Number

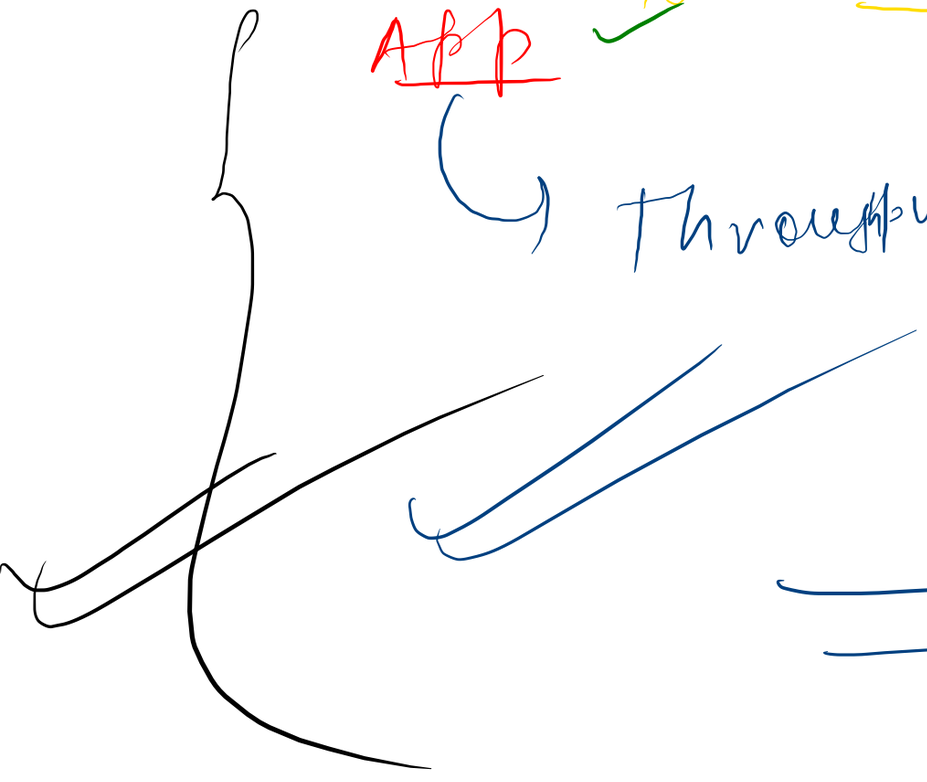

Node.js
Express
server
Python
Flask server
Java
Tomcat
servlet
JSP


server
never ending process
computer


Client
Request
App
App
Server
Response
Req Response cycle


App
server
down
Server is down

Throughput → work done per unit of time


App
10 seconds | 500 Requests
Throughput = 500 Requests / 10 seconds
= 50 R/seconds

Thoughput

Why ?

① Performance evaluation

② Capacity planning

Thoughput

Machine

100 req/s

100 req/s

3 instances

③ Identify bottlenecks

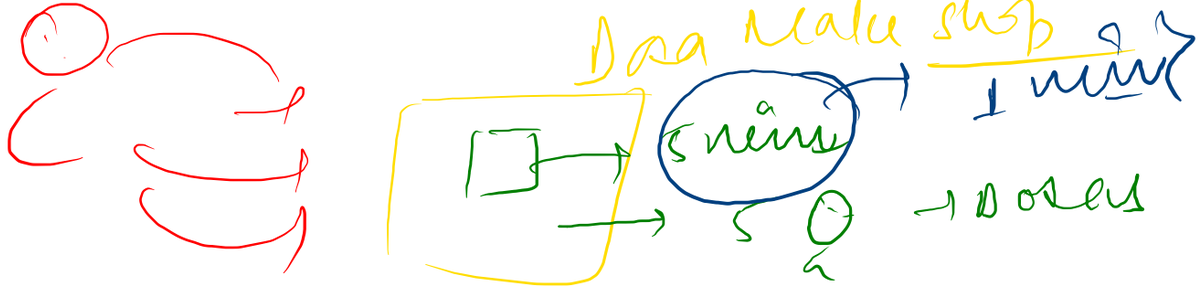
How can I improve throughput ?

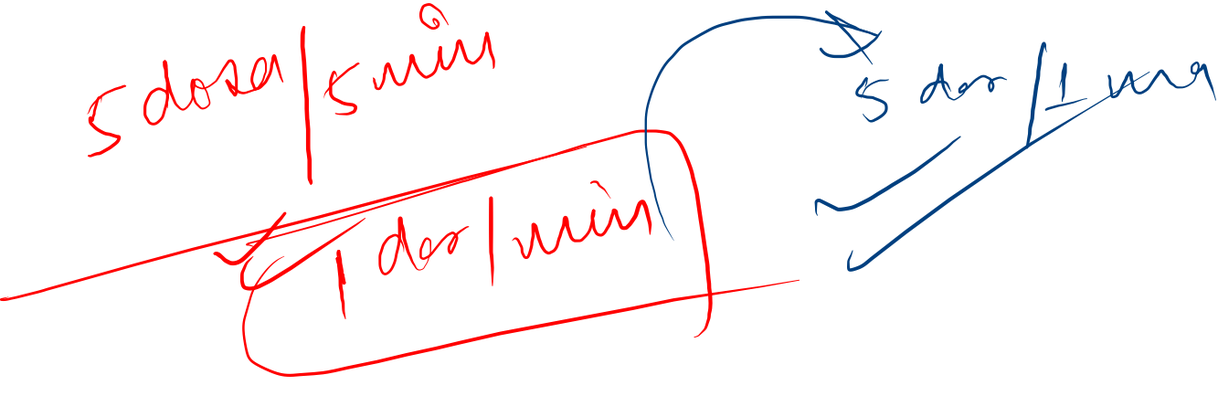
1. Scaling
2. Optimized code — DSA ↑ TC ↓ — code with lower TC
3. Caching / compression
4. DB optimizations
5. N/w optimizations
6. parallel processing

Throughput → related to latency

Latency ↓

Throughput ↑





Throughput

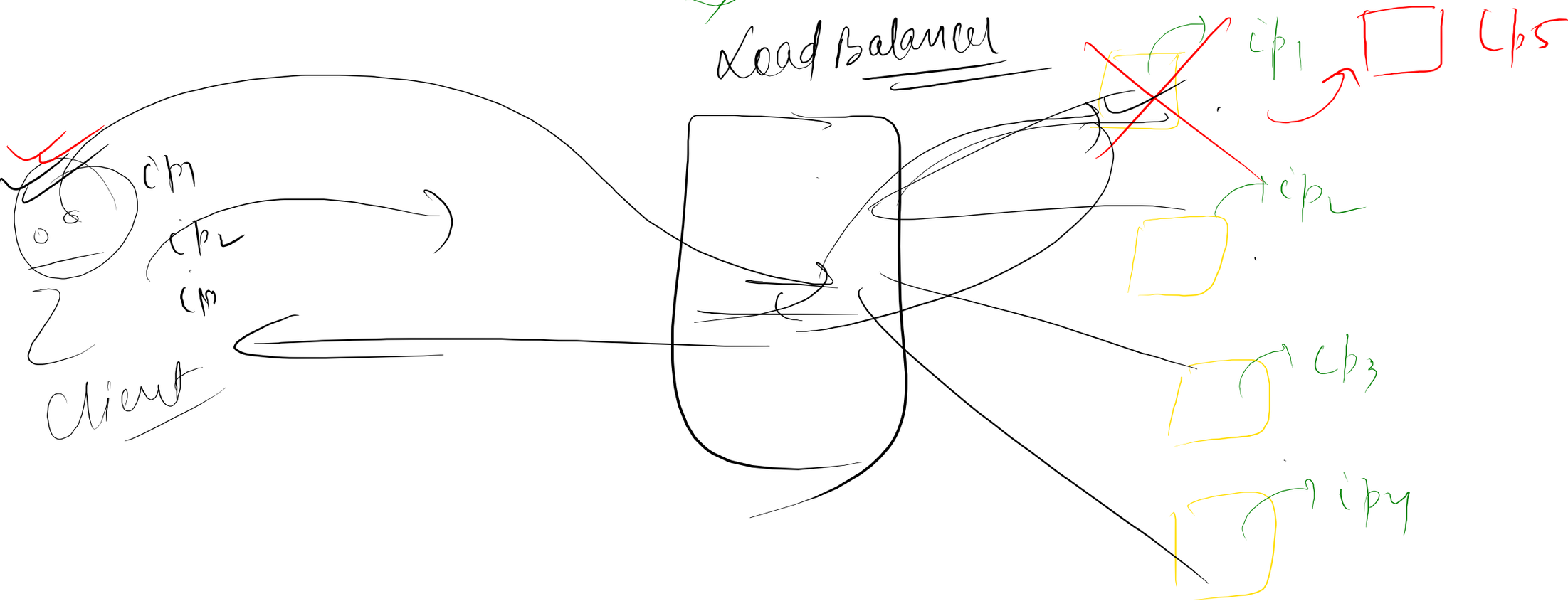
work done / time

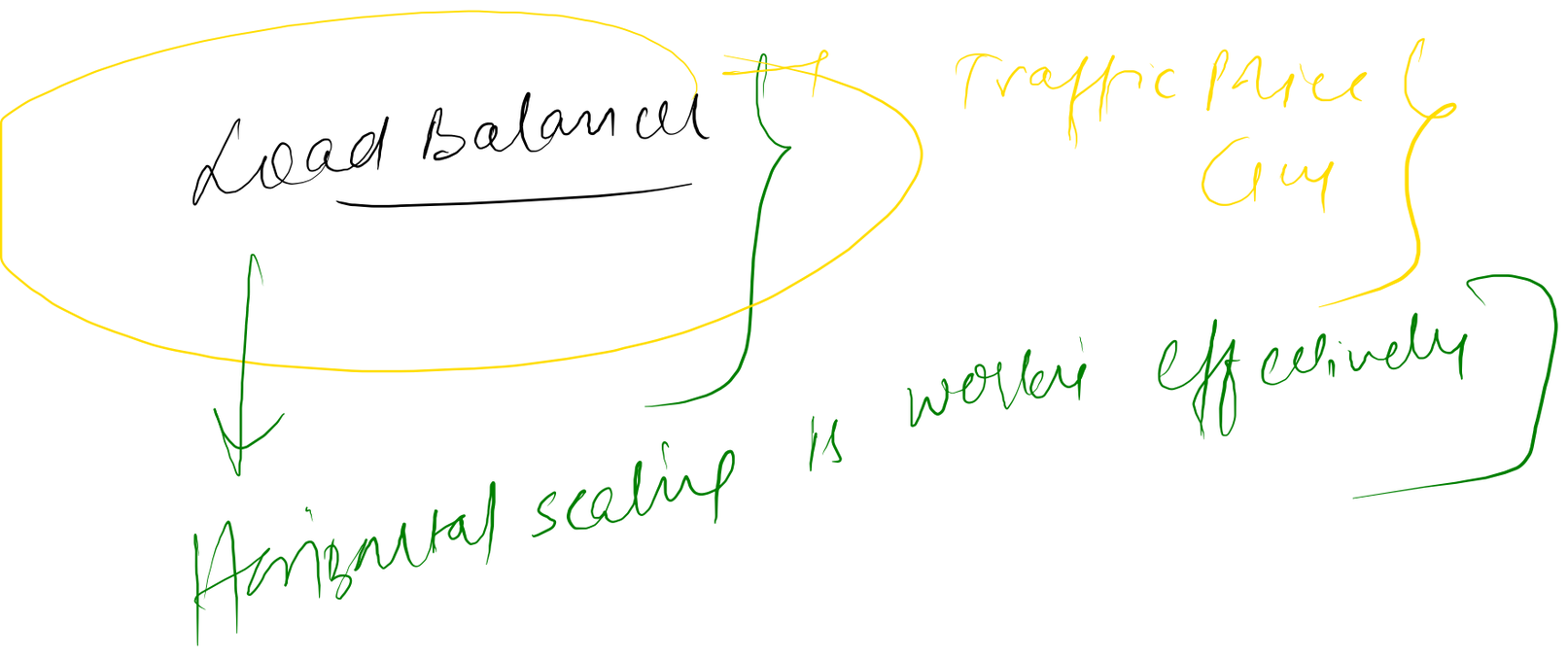
Latency

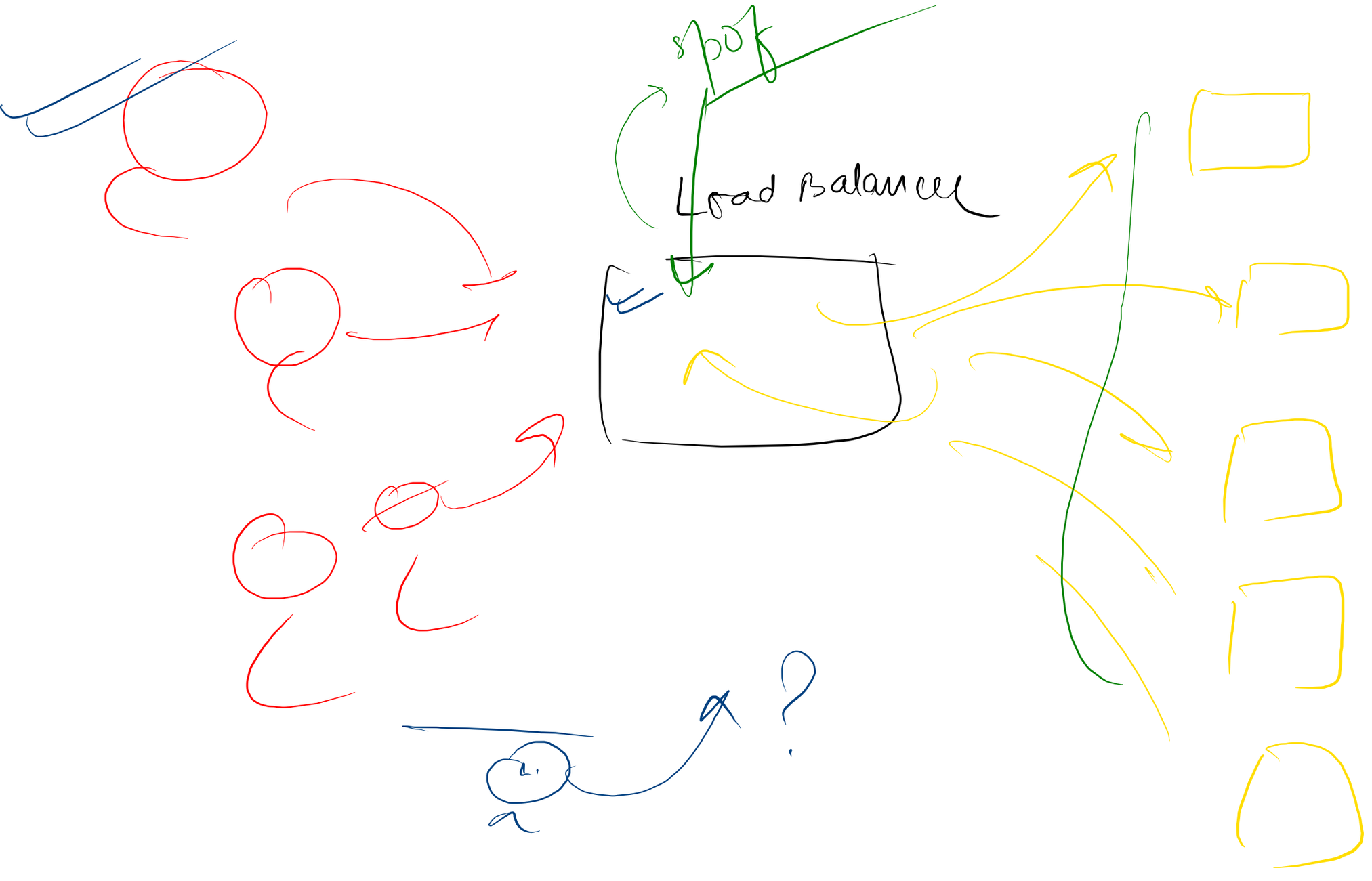
time to run a

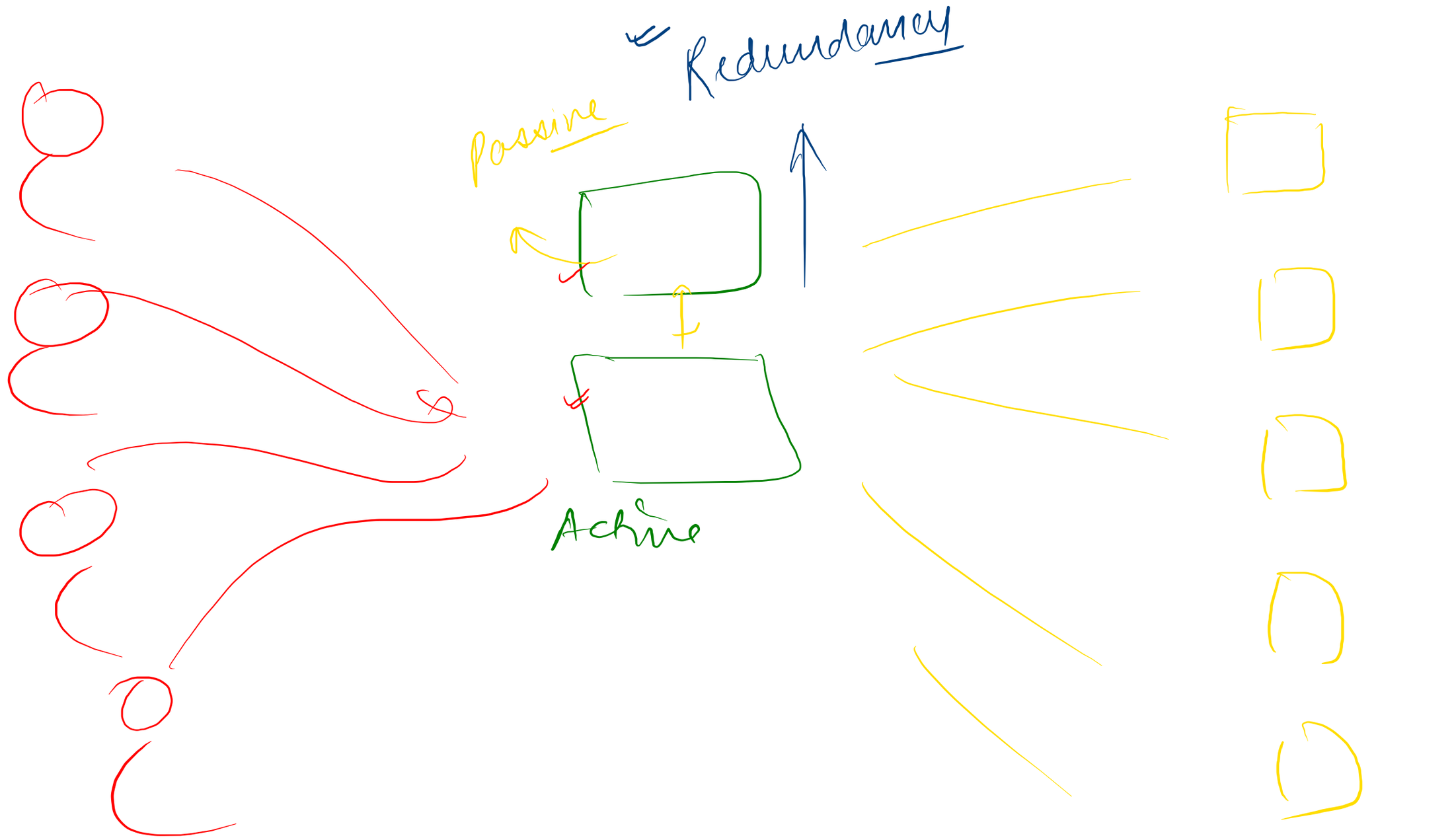
Latency ↓ → serve more

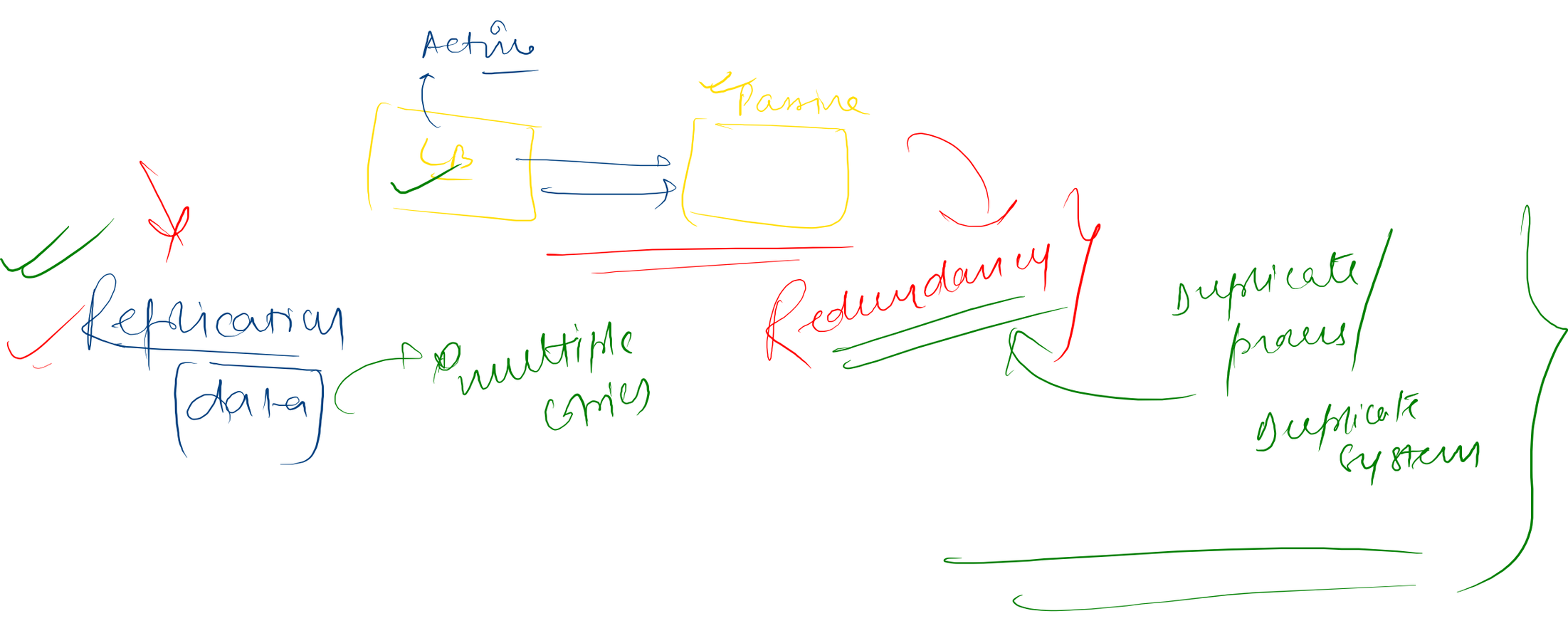
Load Balancers




Load Balancer
Traffic Police Guy
Horizontal scaling is working effectively


spof
Load Balancer
?


Redundancy
Passive
Active


Active
Passive
Replication
data
multiple copies
Redundancy
duplicate process
Duplicate System

Type of Load balancer

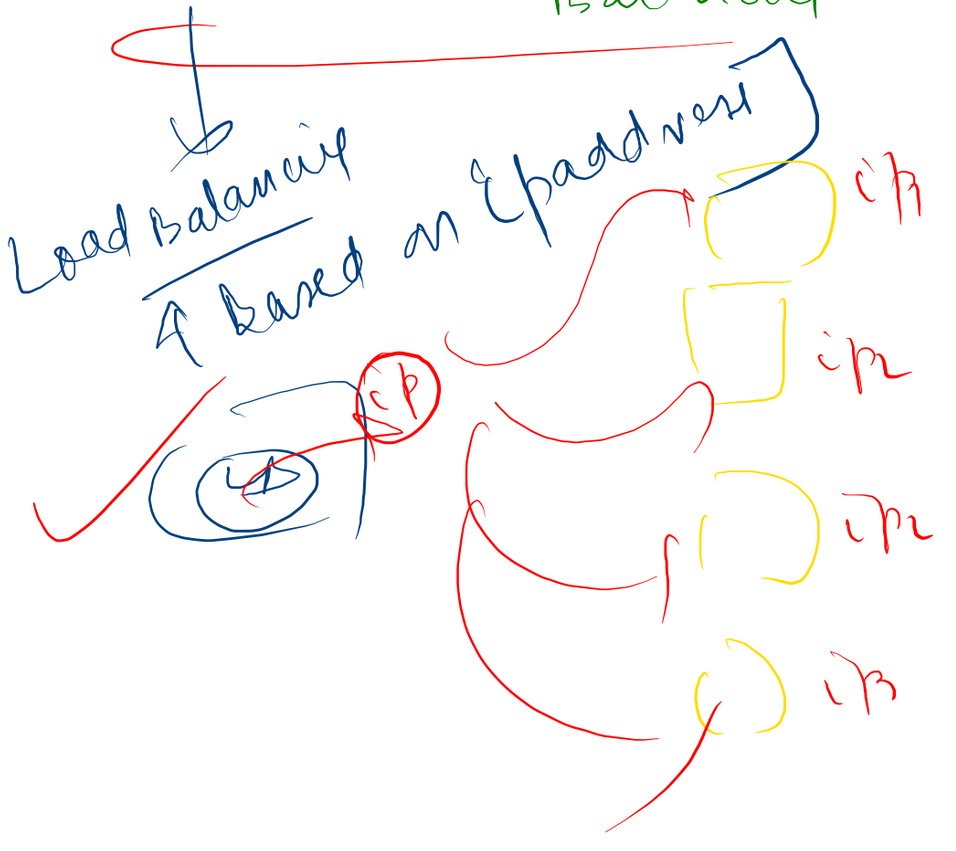
- Physical LB
- Dedicated resources
- And specific features
  - secure/encryption
- More load
- Expensive

Software LB

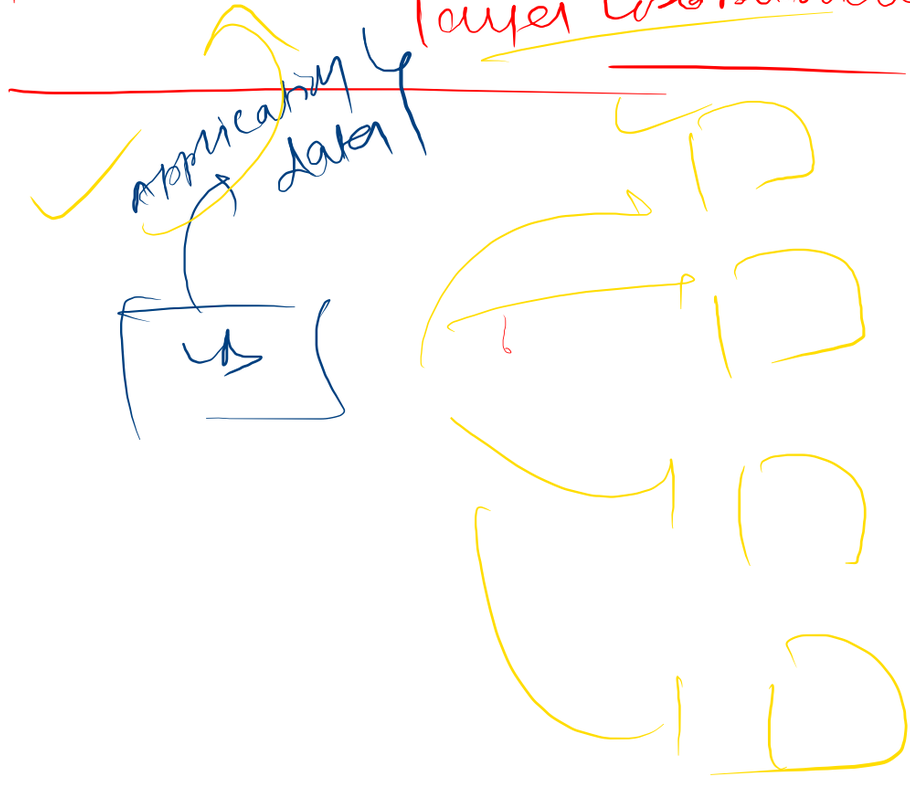
- Flexible
- Shared resources
  - VM / Docker images
- customizable
- Cost effective

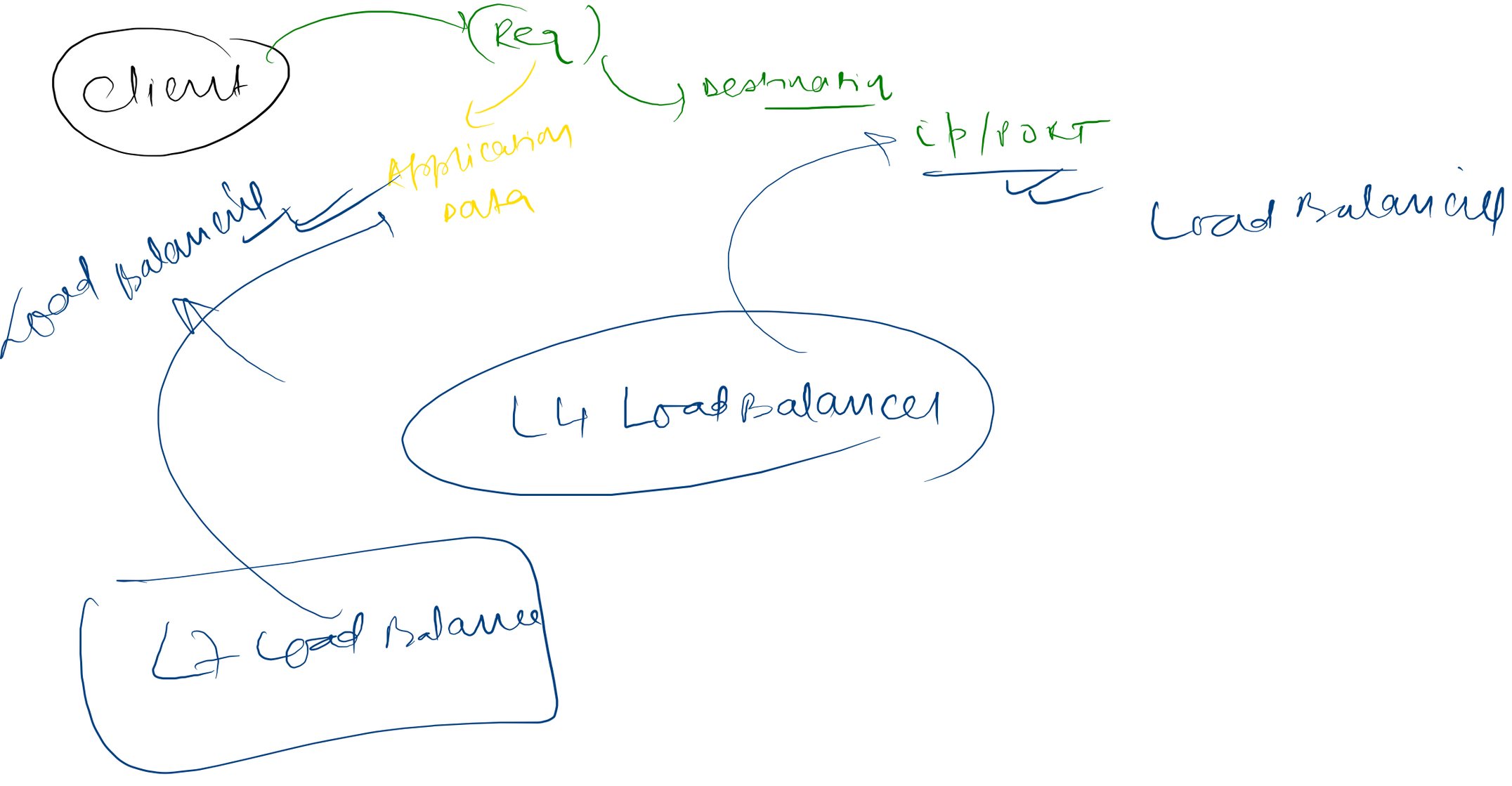
N/w → OSI layer

L4 → Transport Layer | L4 Load balancing



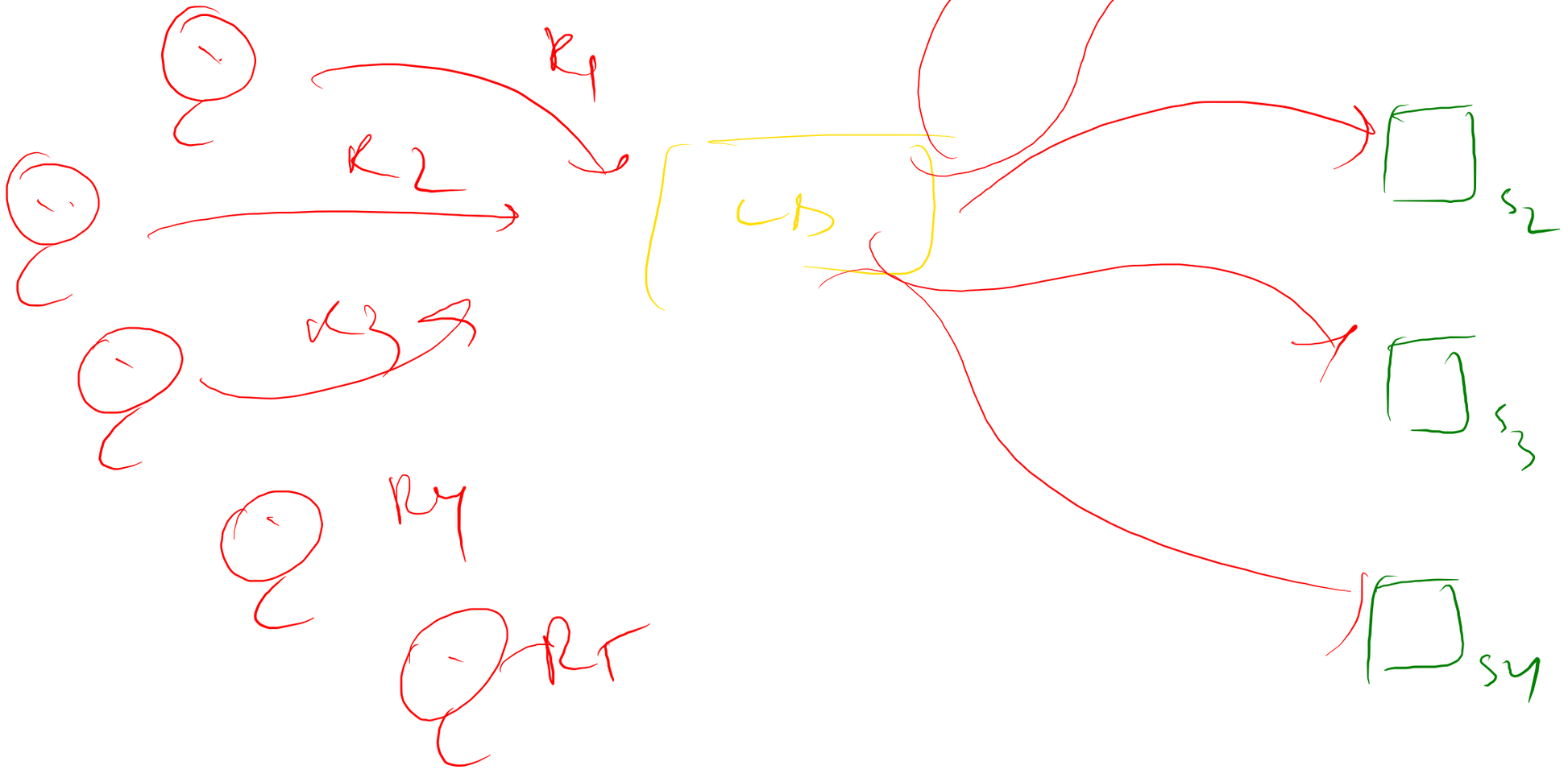
Layer 7 → Application layer | Layer Load balancing



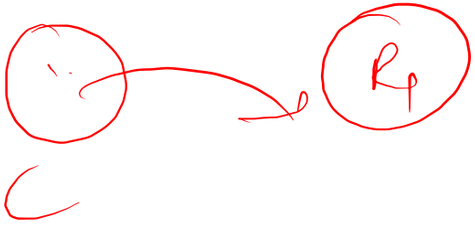

Client
Req
destination
IP/PORT
Application data
Load Balancer
Load Balancer
L4 Load Balancer
L7 Load Balancer

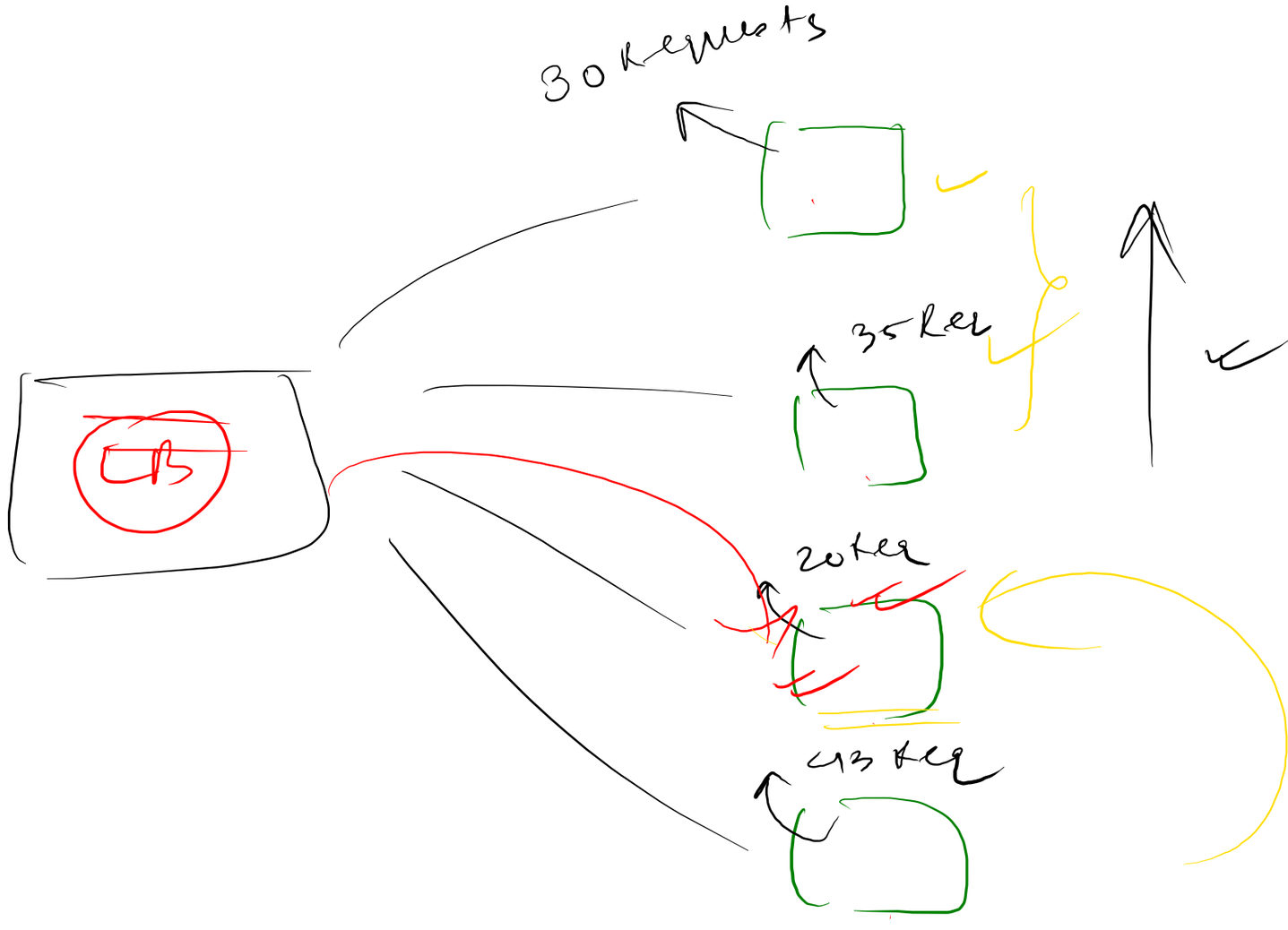
Algorithms of LB

1. Round Robin



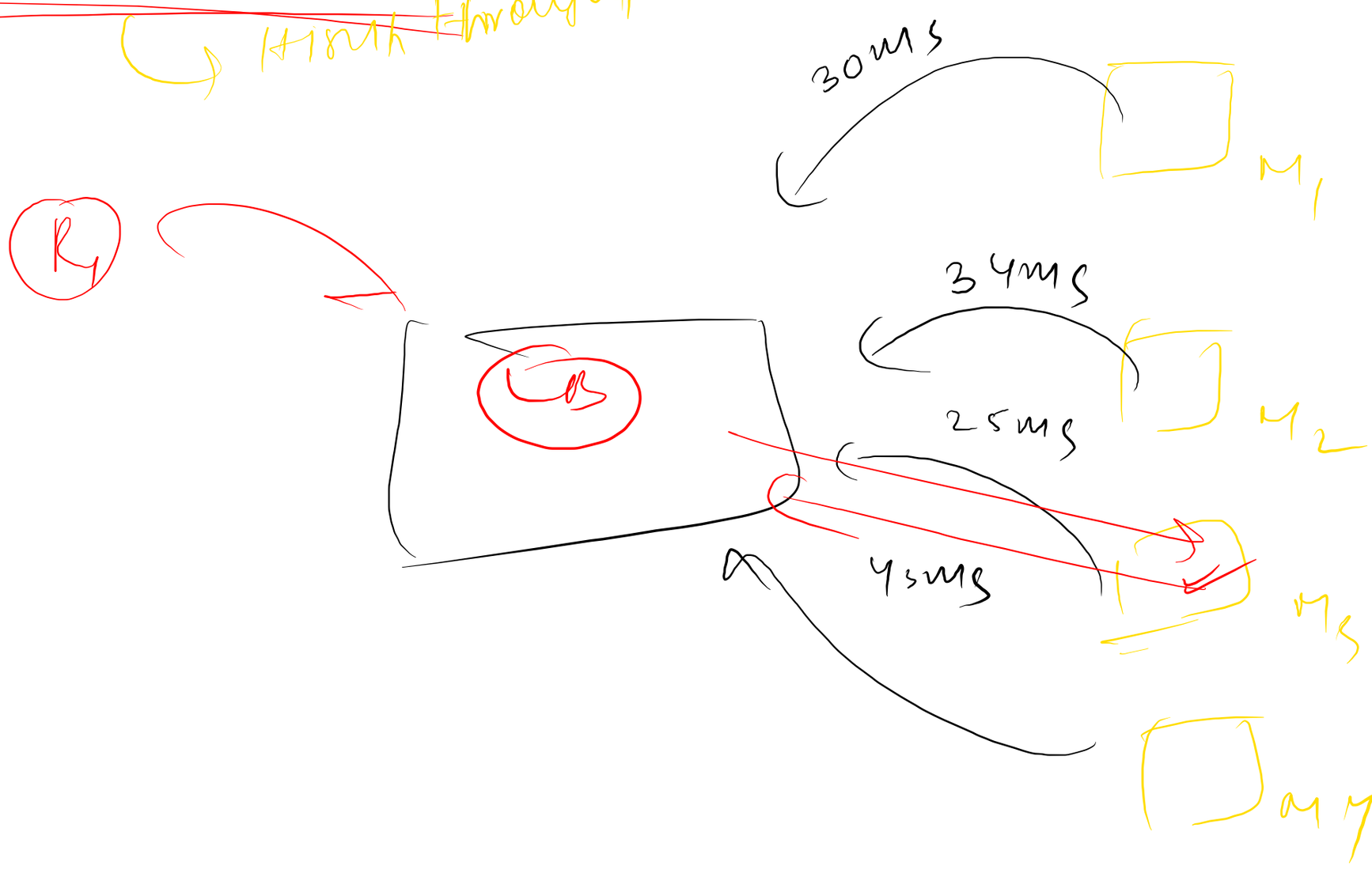
2 Least connection



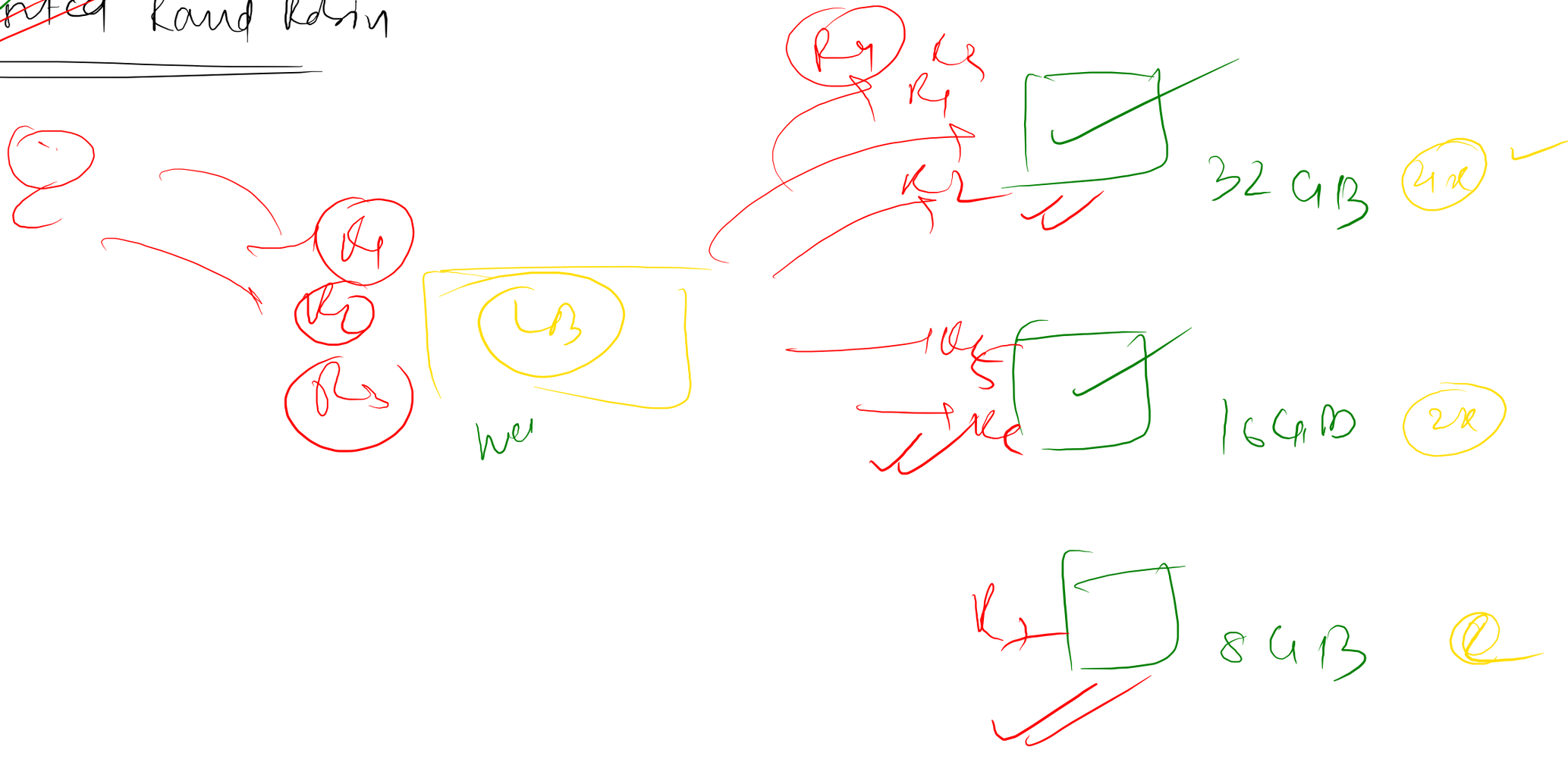


③ Least Response time

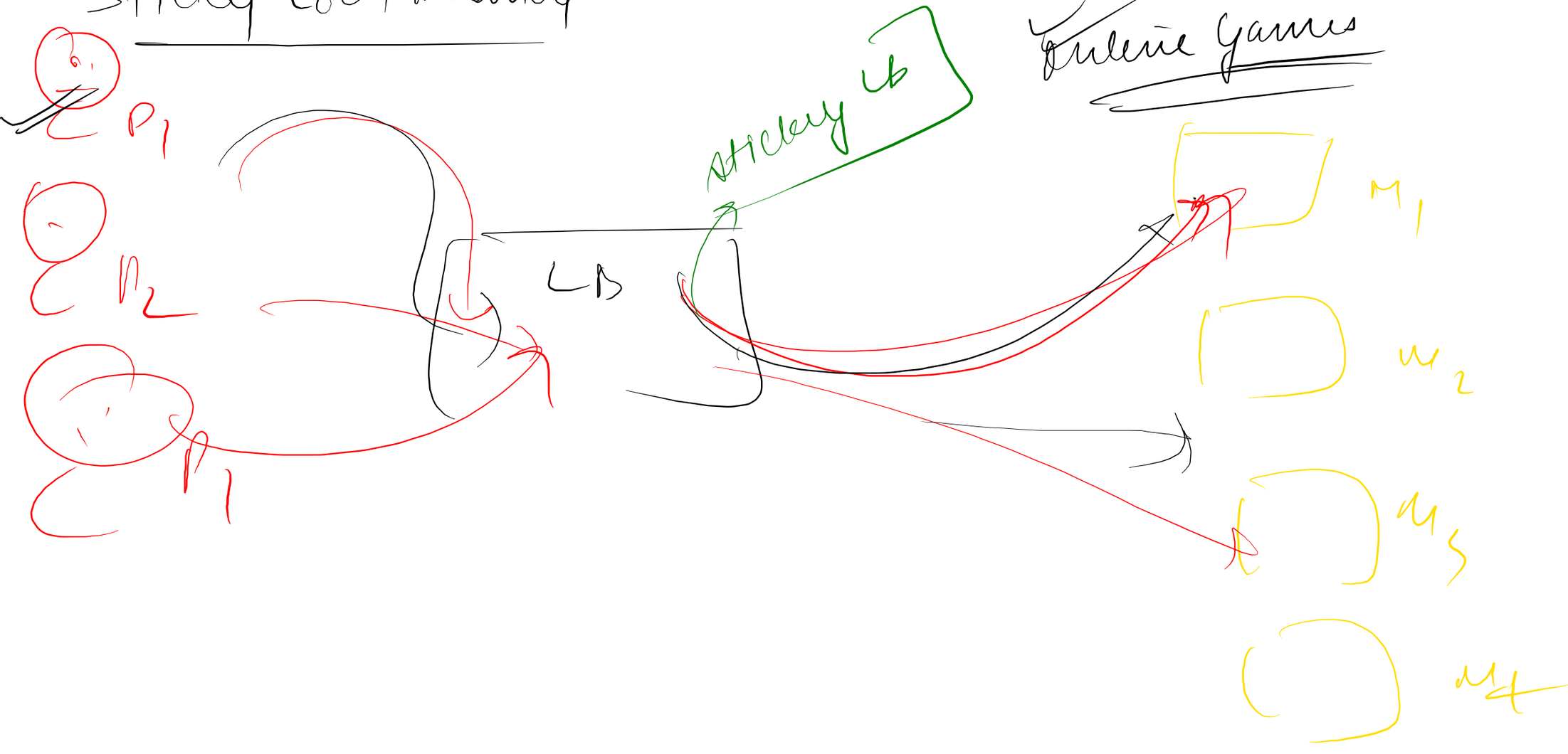
High throughput

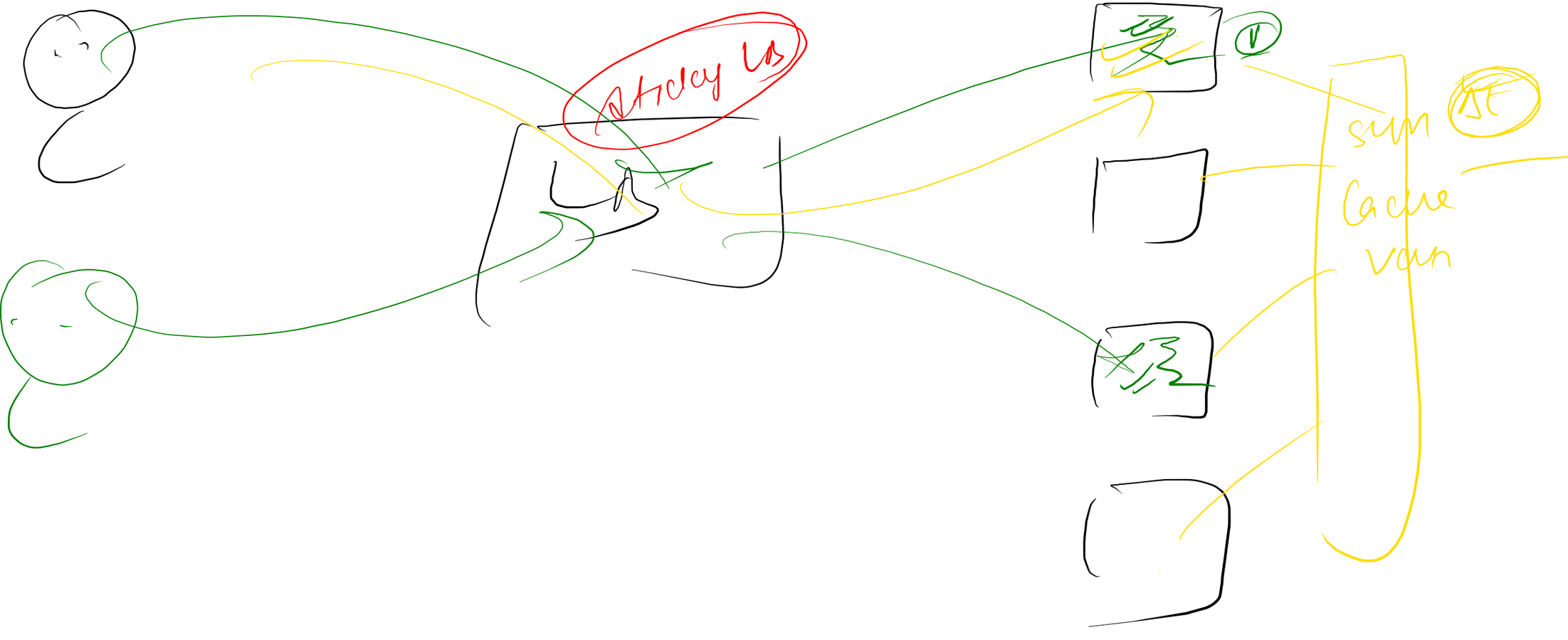


Weighted Round Robin



Sticky Load Balancer




Sticky LB
sun
Cache
van

Monolithic

SOA

SOA

Client Server

Throughput

Latency

Load balancer

caching