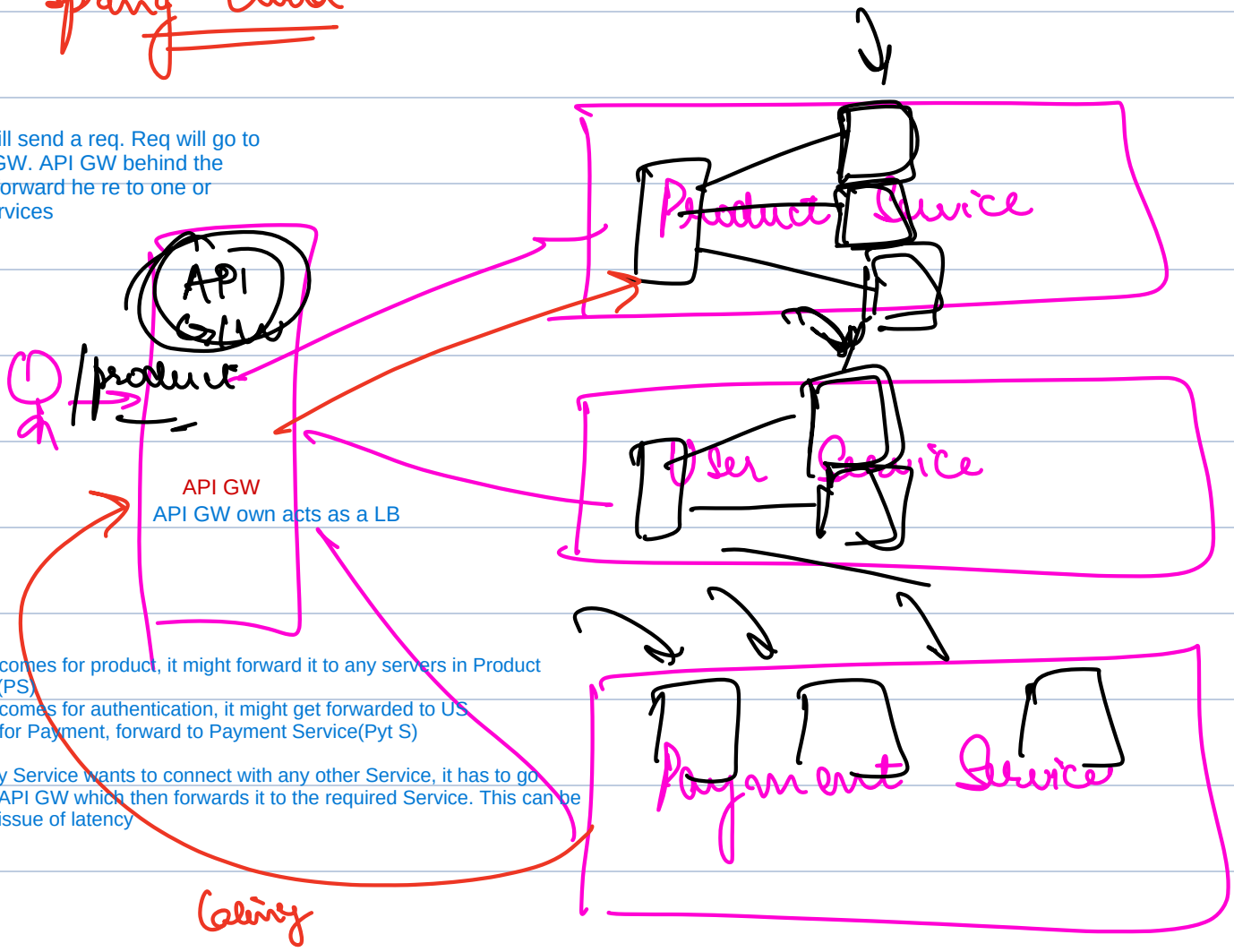
Spring Cloud

Each services will have its LB, Each service will have more servers

Users will send a req. Req will go to an API GW. API GW behind the scenes forward he re to one or more services



API GW
API GW own acts as a LB

Any req comes for product, it might forward it to any servers in Product Service (PS)
Any req comes for authentication, it might get forwarded to US
Any req for Payment, forward to Payment Service(Pyt S)

But if any Service wants to connect with any other Service, it has to go through API GW which then forwards it to the required Service. This can be another issue of latency

Problems with MS

MS -> Microservices

1. Dist Tracing
2. Comm b/w services to be fast

Distance Tracing is issue from the Developer perspective -> since tracing/debugging, logging becomes a problem coz they have their different servers, each server has their own logs; you want to understand whats happening across different servers.

Communication b/w services to be fast

MS

- → Tracing
- → API G/W
- → Comm b/w microservices

Spring → Spring Cloud

↳ Project of Spring

→ makes it easy to implement common things related to microservices based env.

① Spring Boot

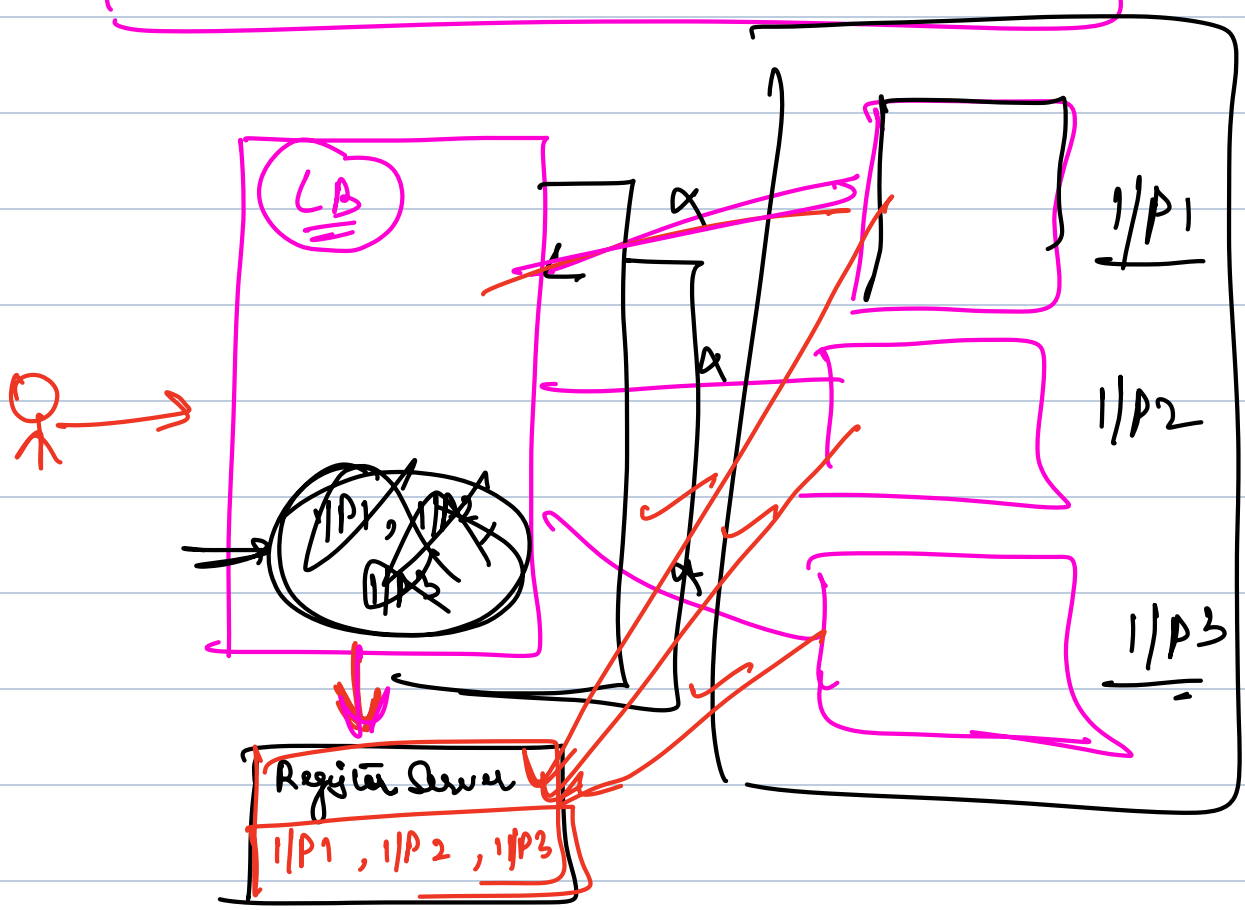
② Spring Data JPA — with this, it is easy to work with DBs

③ Spring Web — with this, it is easy to work with APIs

Netflix's most of the internal structure is hosted on Spring Boot. Then Netflix created a library called Eureka -> this library became so popular that Spring put that within itself as a Spring project under the name of Spring Cloud Netflix (SCN).
SCN provides features:-
1. Service Discovery:-

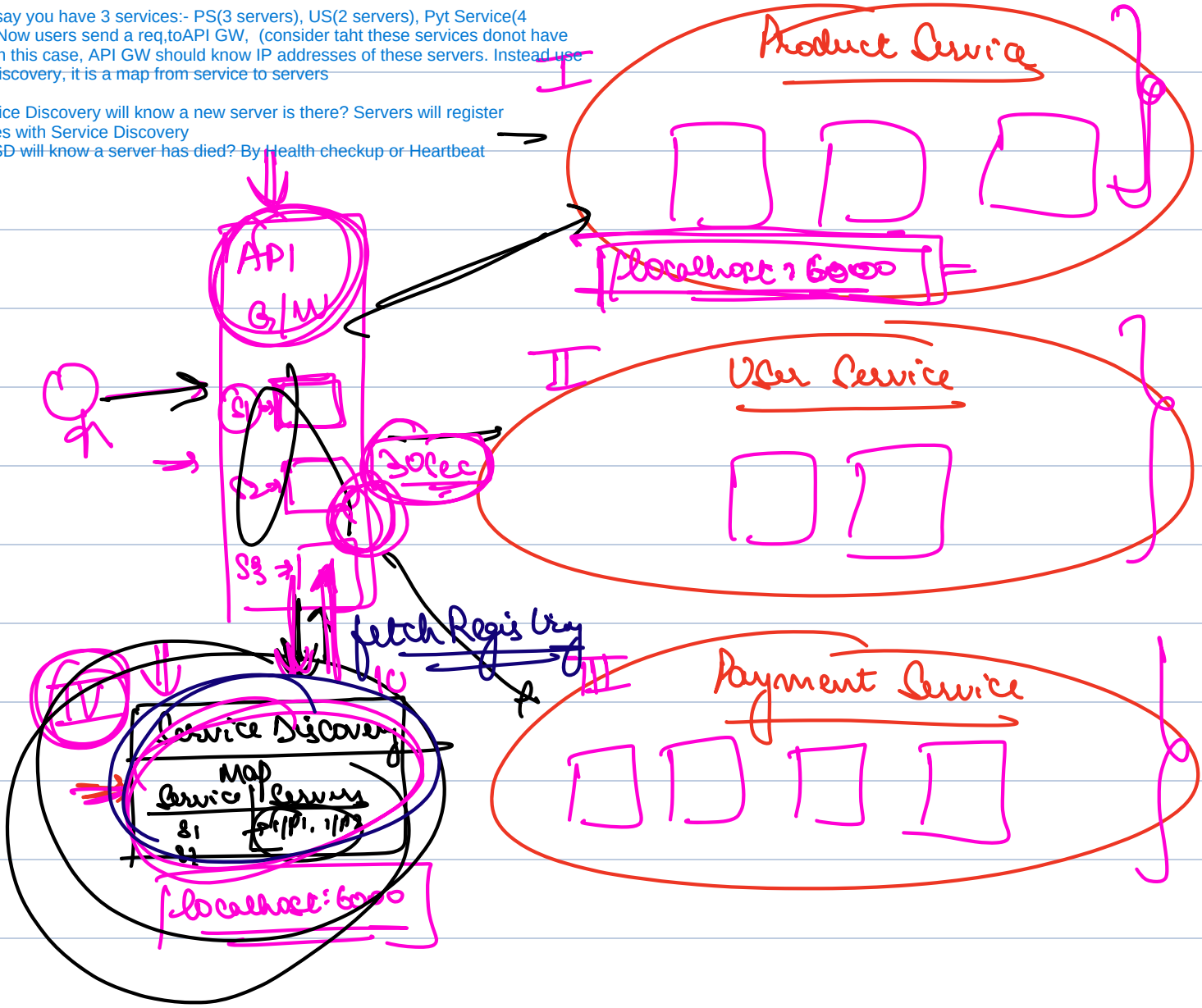
Spring Cloud Netflix

Service Discovery



Now lets say you have 3 services:- PS(3 servers), US(2 servers), Pyt Service(4 servers). Now users send a req,toAPI GW, (consider taht these services donot have their LB, in this case, API GW should know IP addresses of these servers. Instead use Service Discovery, it is a map from service to servers

How Service Discovery will know a new server is there? Servers will register themselves with Service Discovery
How will SD will know a server has died? By Health checkup or Heartbeat



1. Server registers itself with service discovery
2. Health Check / Heart Beat

If I want SD, I have to create a 4th MS. MS -1 is PS, MS2 is US, M3 is PytS. I will create another MS and host at a particular port: localhost:6000. Then to other services, I will have to say there is localhost:6000 where you ahve to go and register yourself, you have to put in the configuration in the services to register themselves in SD localhost and then start yourself.

You have more than one 2 SD in case one SD is down

to avoid latency since, APIGW before any req to any service, goes to SD for ip address and then reach out to that particular service

API GW

Any client of service discovery caches info within itself for some time

When API GW is calling SD, and fetching the whole info to itself -> that is k/a fetching registry

if it is caching for lets say 30 secs, API GW might give stale info but that is a tradeoff

Break till 10:15

User Service → Product Service

Features of Spring Cloud:-
1. Distributed/versioned configuration:- for eg if v2 of US is not compatible with v1 of PS, it will allow configuration in a way such that configuration which are compatible may work

2. Service registration & discovery:- server will know what are the 2 instances of PS or 3 instances of US

3. Routing:- correctly routing the req to correct server. Spring Cloud allows you to create your own API Gw by using a Spring Cloud Project. if you are not using AWS or any other means

4. Service-to-Service calls: One service calling other services. If I go via API GW, it will add latency
5. Circuit Breakers:- Circuit Breakers is one of the patterns of MicroServices(MS)-> how to know if the other MS is down, you donot have to write the code, Spring Cloud has already that particular thing
6. Load Balancer

Spring Cloud is not just one library, it is a set of libraries. Within Spring they are multiple libraries. Spring Azure, Spring AWS -> it will make easy for you to work with AWS functionality. Spring Messaging API implementation -> SQS. ElasticCache(AWS's Redis), if you will host MongoDB, or Redis then use ElasticCache

If you want to deploy your MS envt in Kubernetes -> then use Spring Cloud Kubernetes

Spring Cloud

For Eureka server, you have to do only these 2 things -> 1. set
client: registerWithEureka: false
fetchRegistry: false

and 2. @EnableEurekaServer