Will start at 9:10

Agenda

1. Project Req and Microservices — Why? → APIs
2. Why Frameworks
3. Intro to Spring
4. Dependency Injection and Spring — Most fundamental Concepts of Spring
5. Spring Boot
6. Building our first API

Break till 10:10 PM

Git: Version Control

API

Frameworks: Why Frameworks, Spring Boot, first app using FW

Microservices : HLD Module

→ Authentication
→ Classes.
→ Dashboard
→ Assignments.
→ Career Platform
→ Mailing Portal
→ Resume Builder.

One codebase : All I/M wrt everything are together

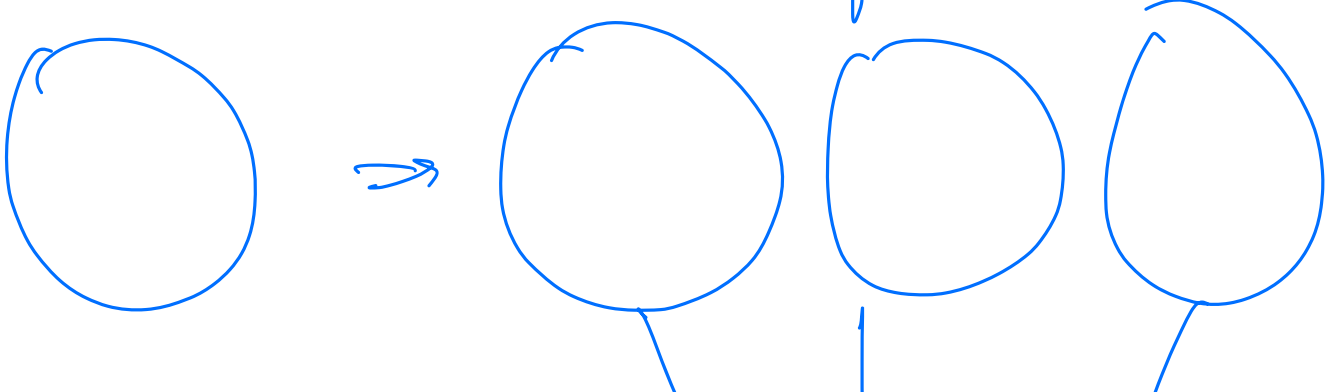
→ Size of codebase will increase

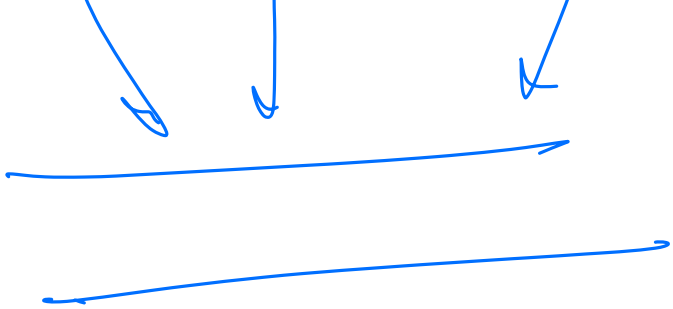
1 Mist

→ (1) Slow compiling and slow application startup

(2) Code will be overwhelming to understand.

(3) Inter team conflicts.

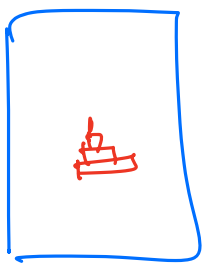


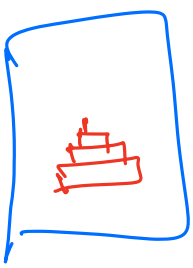


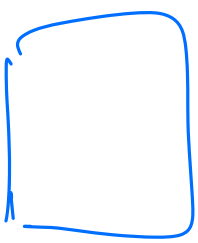
④ Same tech stack ⇒ legacy ⇒ Not easy to try new tech.

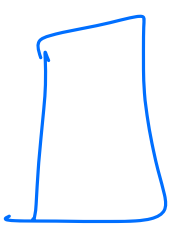
⑤ Selective Scalability is not there
→ Spend more money

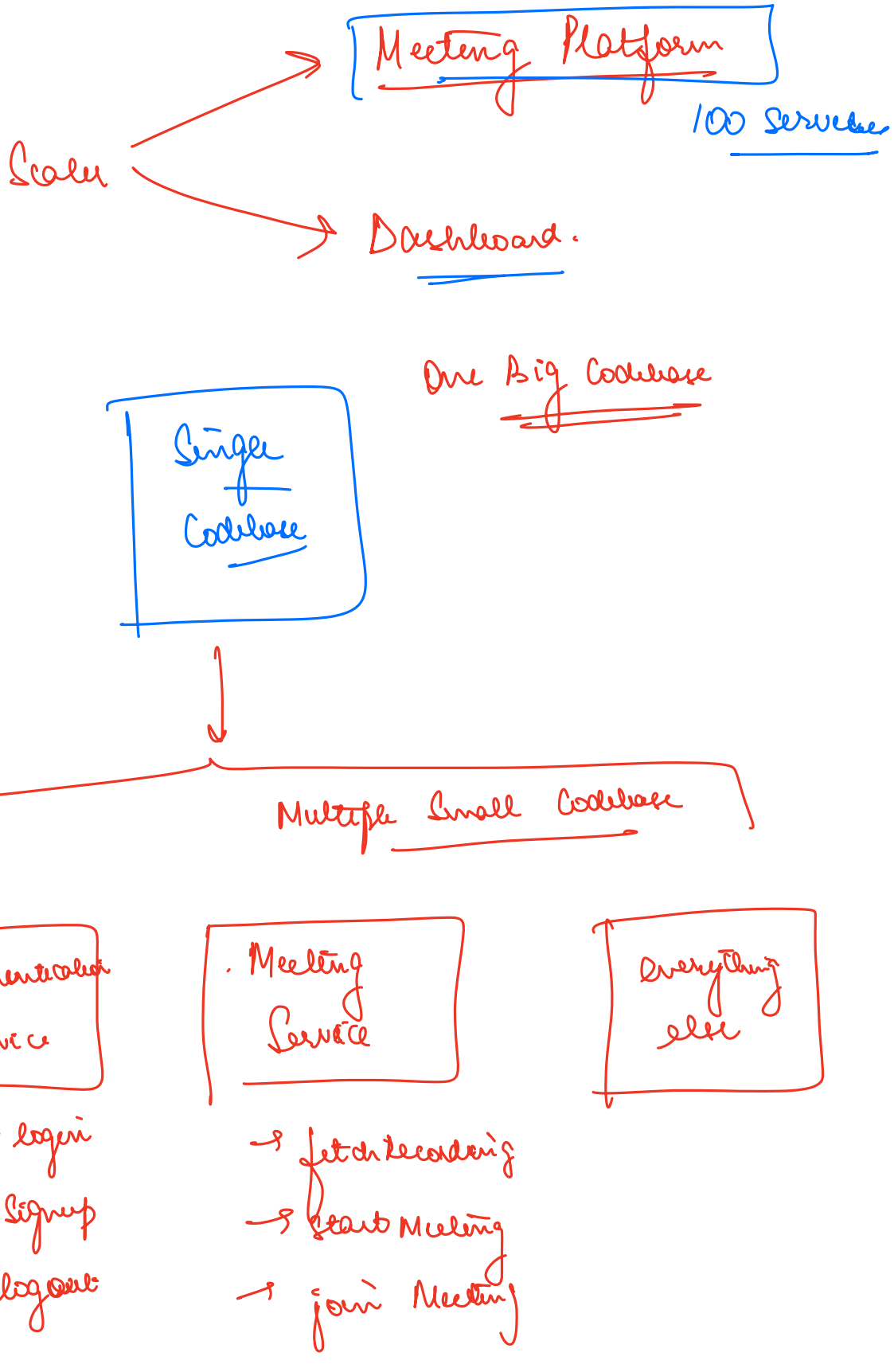
Does every part of appⁿ need same amount of Scale.

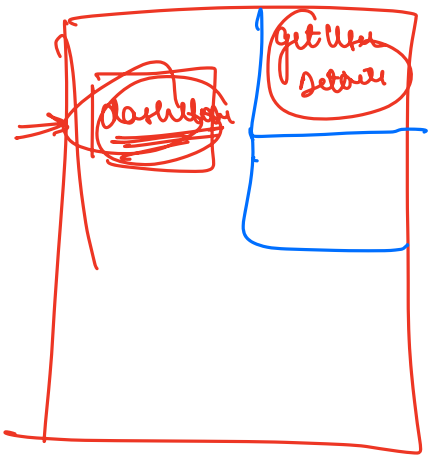




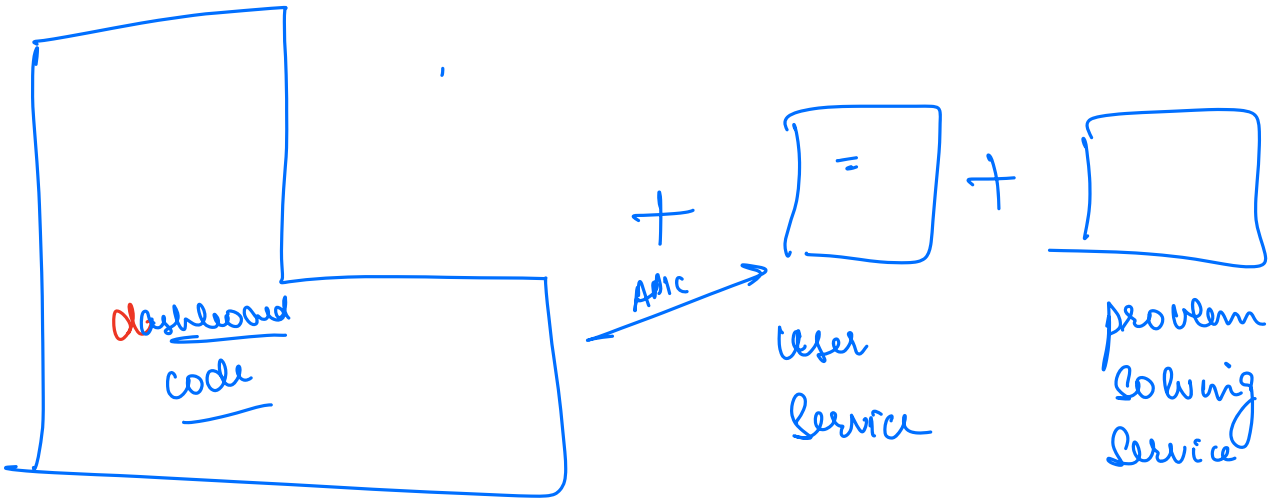


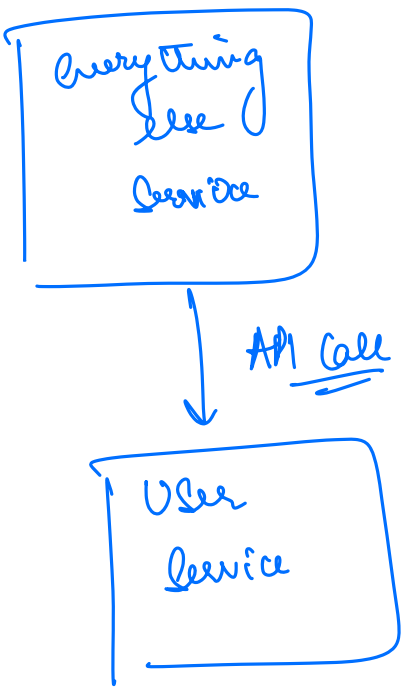



Meeting Platform
100 Servers
Scale
Dashboard.
One Big Codebase
Single Codebase
Multiple Small Codebase
Authentication Service
Meeting Service
Everything else
→ login
→ Signup
→ logout
→ fetch recordings
→ start Meeting
→ join Meeting



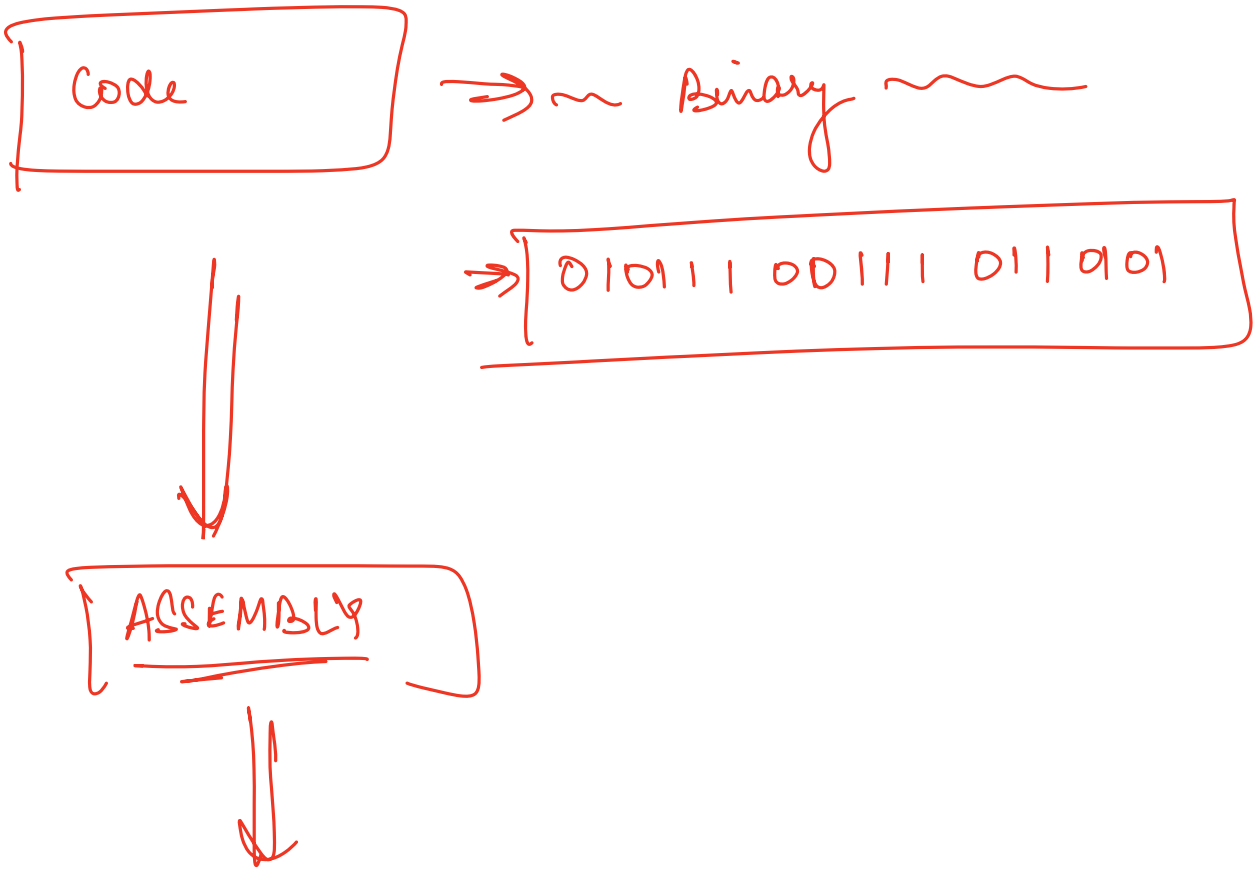
= function Call from one module to other.





⇒ As codebases evolve, they are going to have a lot of f^n ⇒ a lot of similar things.

1. Creating APIs
2. Talking to DB.
3. Authentication
4. Logging



High level Lang

[1, 1, 1, 1, 1, 1]

→ [[1] * 6

⇒ a lot of thing that I may want to do are easy to do.

C ⇒ custom write sort

Java / Python / JS ⇒ inbuilt sorting fn

Arrays.sort( )

Priority Queue <>

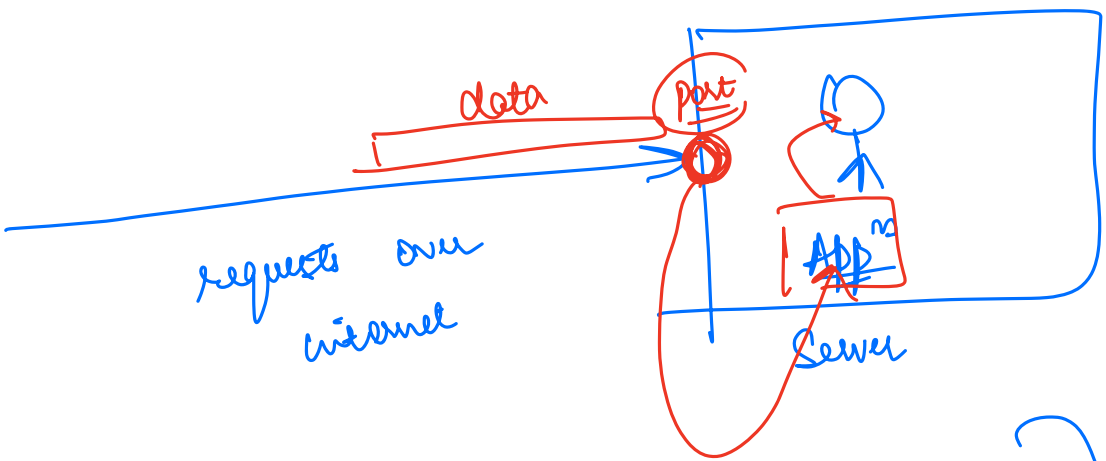
heap ?-

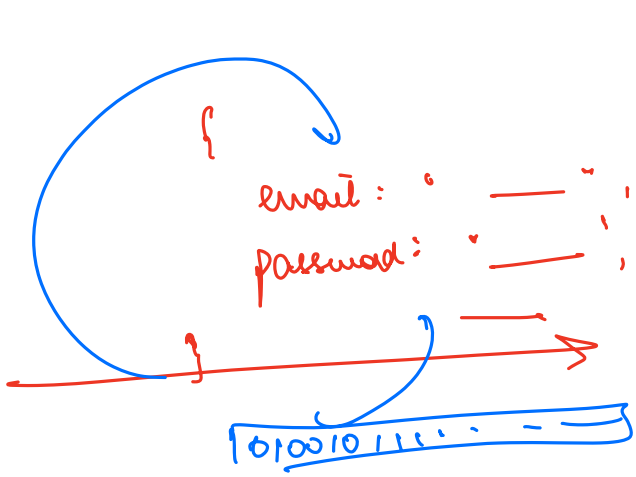
⇒ provide ready to use implementation of common things that many software may be doing.

⇒ a SWE only has to think of logic.

⇒ exist methods for most of those things.

Building APIs





login (String email, String password) {

"STAND ON THE SHOULDERS OF GIANTS"

"DON'T REINVENT THE WHEEL" — Write everything from scratch

⇒ Try to reuse as many things as possible.

Frameworks :⇒ Functionalities provided in a ready to use way to make frequently occurring tasks easy to do.

⇒ efficient/ ready to use way to do common things

⇒ creating APIs

⇒ talking to DB

⇒ wiring auth.

(1) Build in efficient way

(2) Reducing overhead

(3) Ready to use fn

Lib v/s Framework

set of f^n that you can use as per need.

⇒ set of methods.

⇒ use whatever method is req^d.

⇒ ecosystem

⇒ US visa.

⇒ you will have to mould yourself as that f/w expects

⇒ you have to follow practices of f/w

diff f/w exist for diff prog languages

"easy way to do common thing"

→ Auth: OAuth2

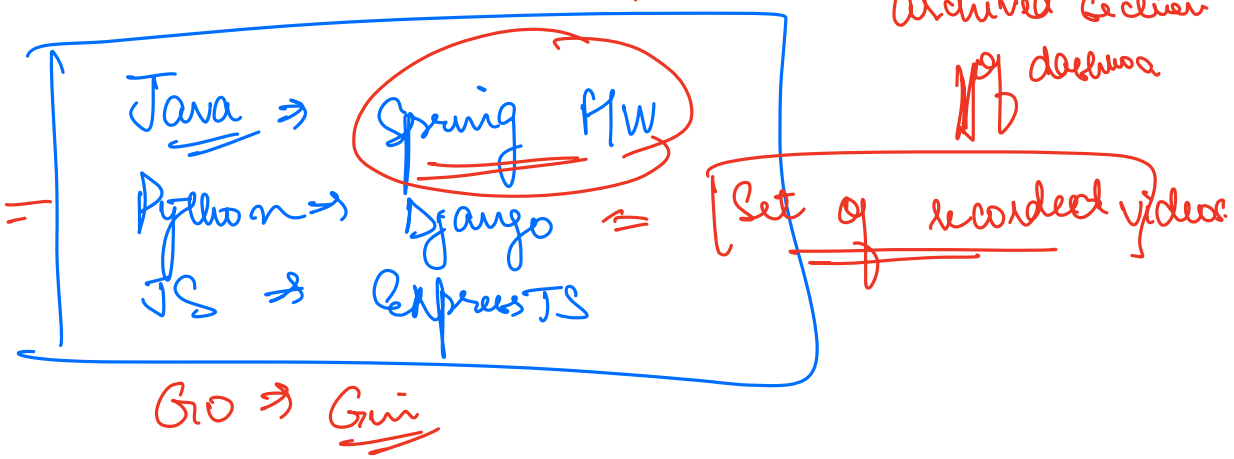
→ API: get, post

→ DB

⇒ easy way to do these concepts

① Learn Concepts

② Implement in Spring



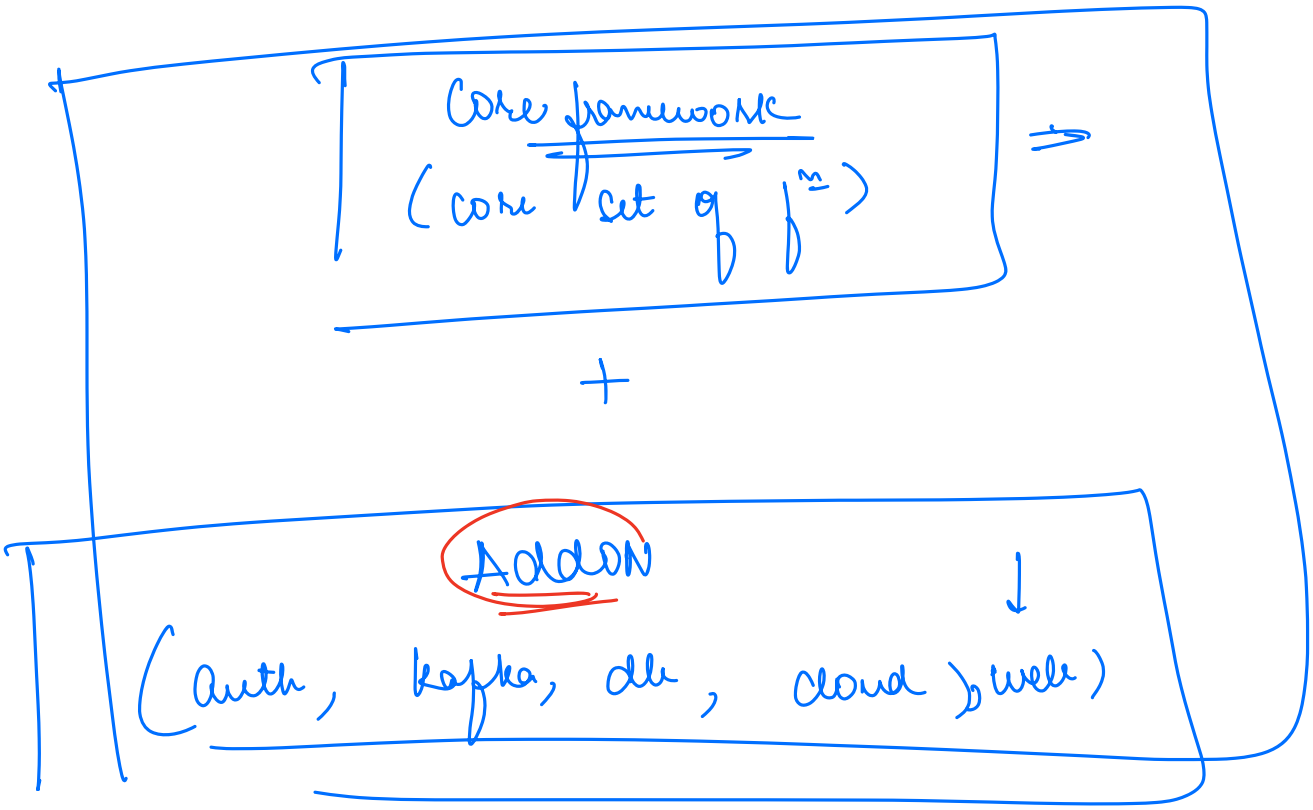
Spring F/W

⇒ Set of Projects that allow easy creation of enterprise level Java App^n

↳ prod^n ready

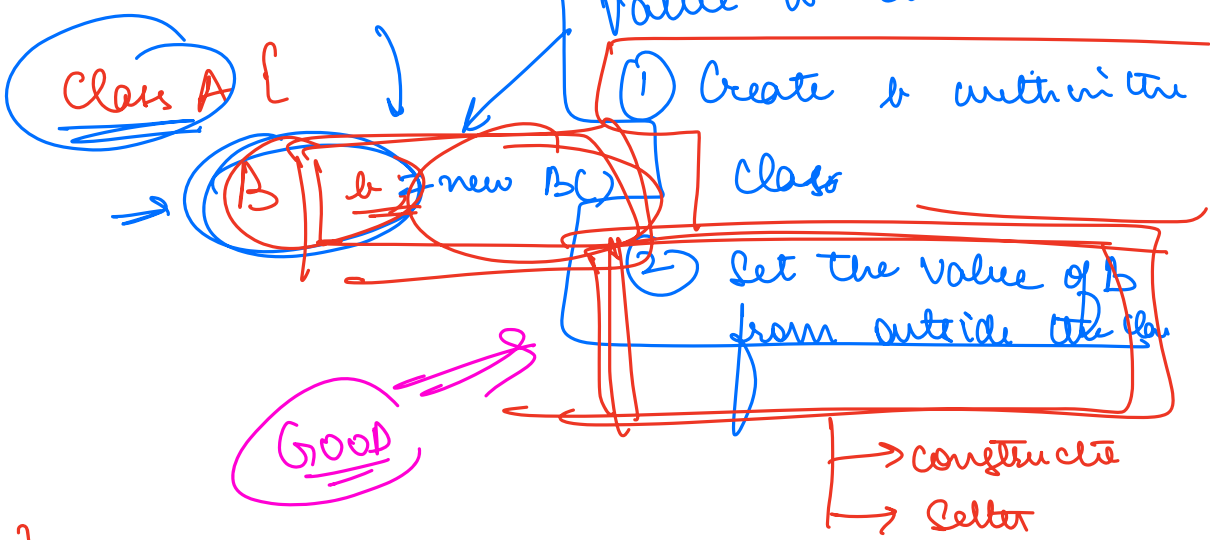
by allowing to do a common thing easily

Spring



Dependency Injection

(How to get value of a to value to this 'b')



}

```
Class A {
    B b;
    A(B b) {
        this.b = b;
    }
    void setB(B b) {
        this.b = b;
    }
}
```

Constructor injection

Method injection

```
Main {
    B b = new B()
    A a = new A(b);
}
```

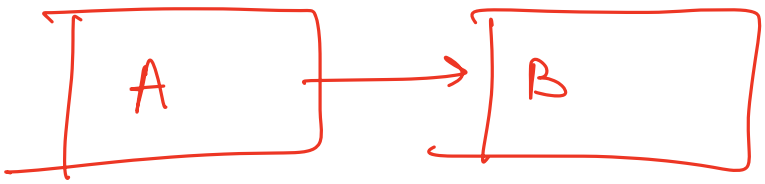
Creating a dependency outside the class is better than creating dependency within class

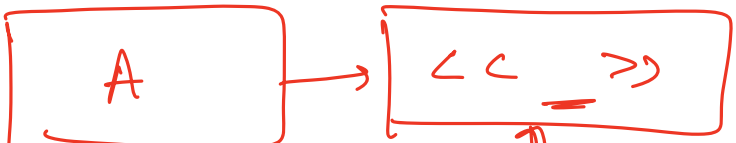
① Same obj

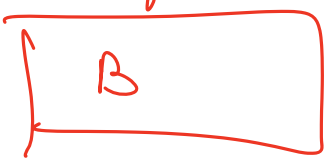
User Service {
db
save()
create()
}

Product Service {
DB db
save()
getes()
}

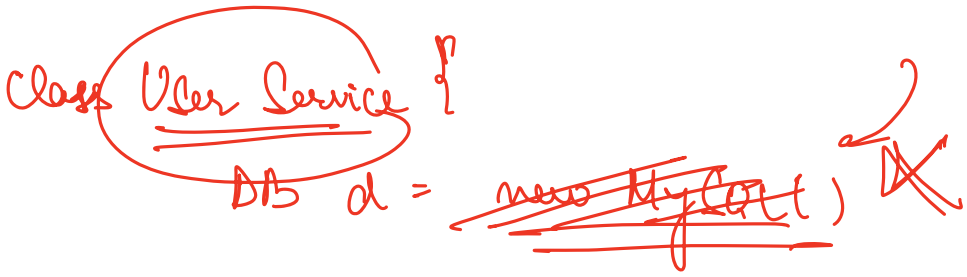
① Create one DB obj and pass to multiple places.







⇒ loosely coupled



dependency injection

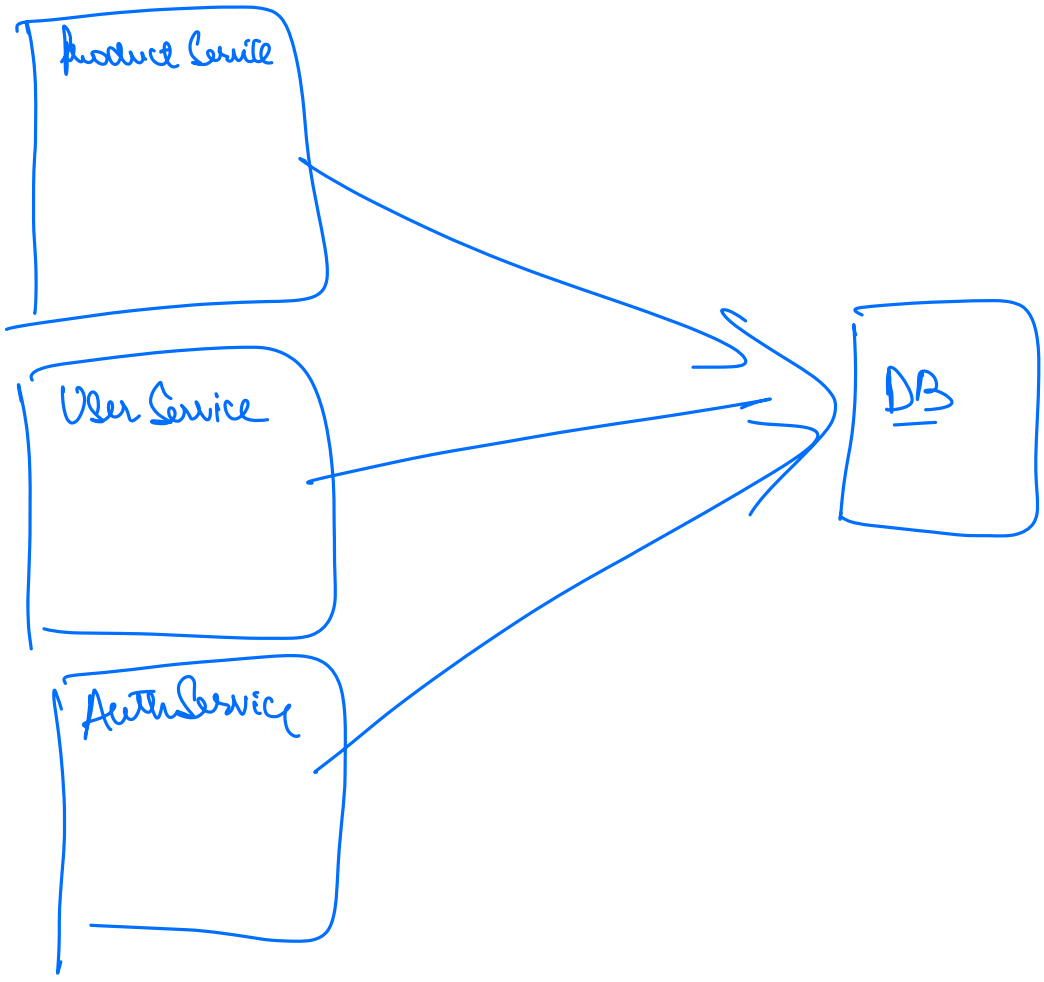
```
User Service (DB db) {
    this.db = db;
}
```

instead of a class creating objects of its dependencies itself passing the dependencies to class

Dependency injection

- → Loosely Coupled
- → Easy to Test

Whenever we create an app^{n}, there are many common dependencies that have to be injected in multiple classes.



⇒ Spring provides a very easy way to do dependency injection.

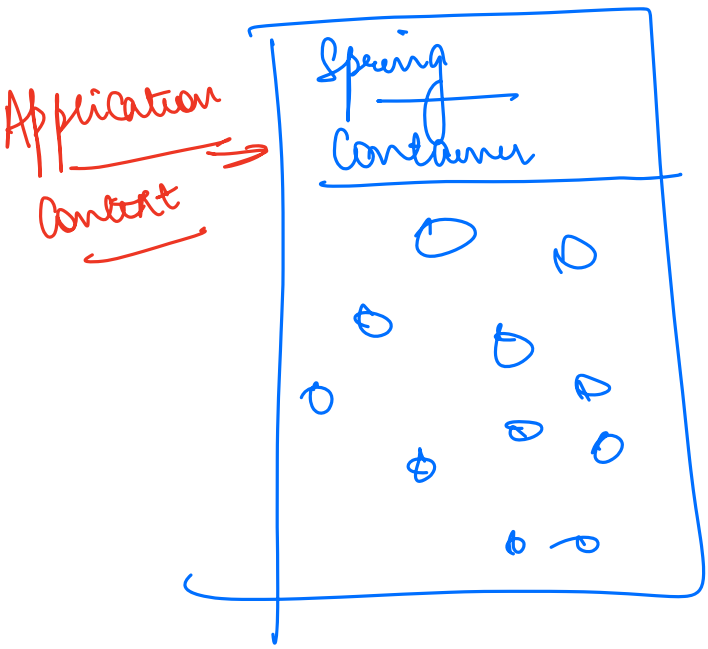
Inversion of Control ⇒ F/W doing dependency injection on your behalf.

Bean ⇒ special objects that you give to Spring so that it can inject them automatically whenever needed.

DBDao = new MySQL();

→ new ProductService (dao)

Bean → an object that Spring will automatically Create as well as automatically inject wherever need.



1. Start Spring App
2. Creates objects of all beans
3. puts in container for future use.

→ Spring F/W provided DI

→ But a lot of I^m that I need are via Spring add-ons

→ EARLIER

Configuration xml

for each addon.

Create an .xml file

define a lot of thing in .xml

Spring-data

Config

username

password

url

package

hair cut + head spa + face spa

Gui + Arm +

⇒ LIIT

Package ⇒ Defined config

Popular

Before SB → manually create config for each add on

After SB → Cocktail

⇒ automatically configuring addons via known best practices while still making it easy for you to override them.

SB → Spring project
- → Spring data.
- → Spring Security

→ no need to write any config.xml

→ Spring boot will automatically config data via known best practices.

Any good fw
1. have an opinionated view. (this is best way to do this)
2. easy for you to change if you want

Now, every Spring Project uses Spring Boot