Python → support @ scaler.com

future req ⇒ 6 months / 9 months

6 projects → 3 → 15

{ Current req / very limited future req. } ⇒ { 5 Open → Java, Kotlin, React, Python, Node }

→ active github profile / ↑ daily

MOST IMPORTANT

* L1 → ~~L2~~

L2 → ~~L3~~

* (10) → 100 → attendance, psp, contest, mock interviews — 100%

→ IntelliJ IDEA Community Edition

Agenda:

i) Pre-talks

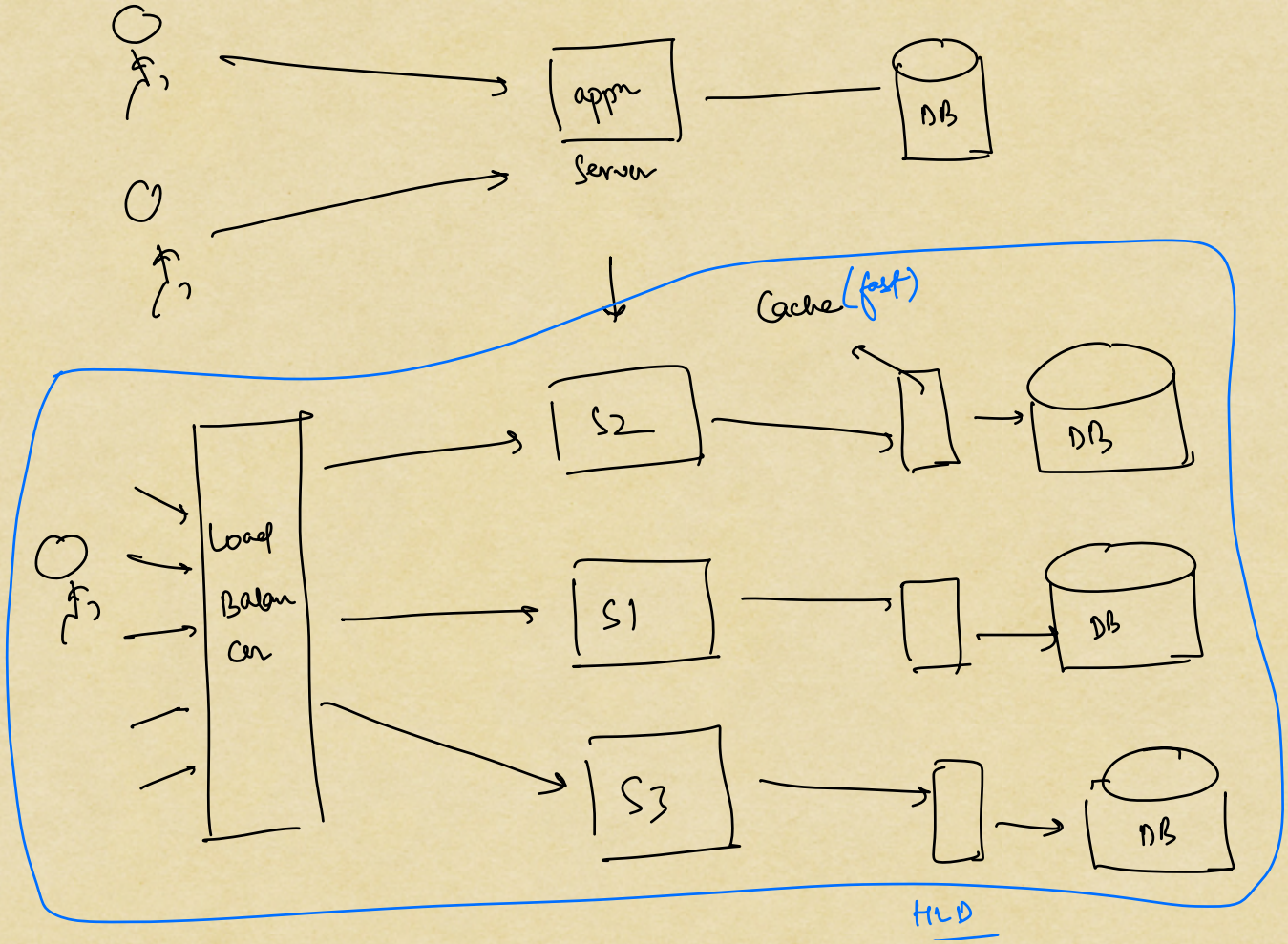
ii) Intro to LLD

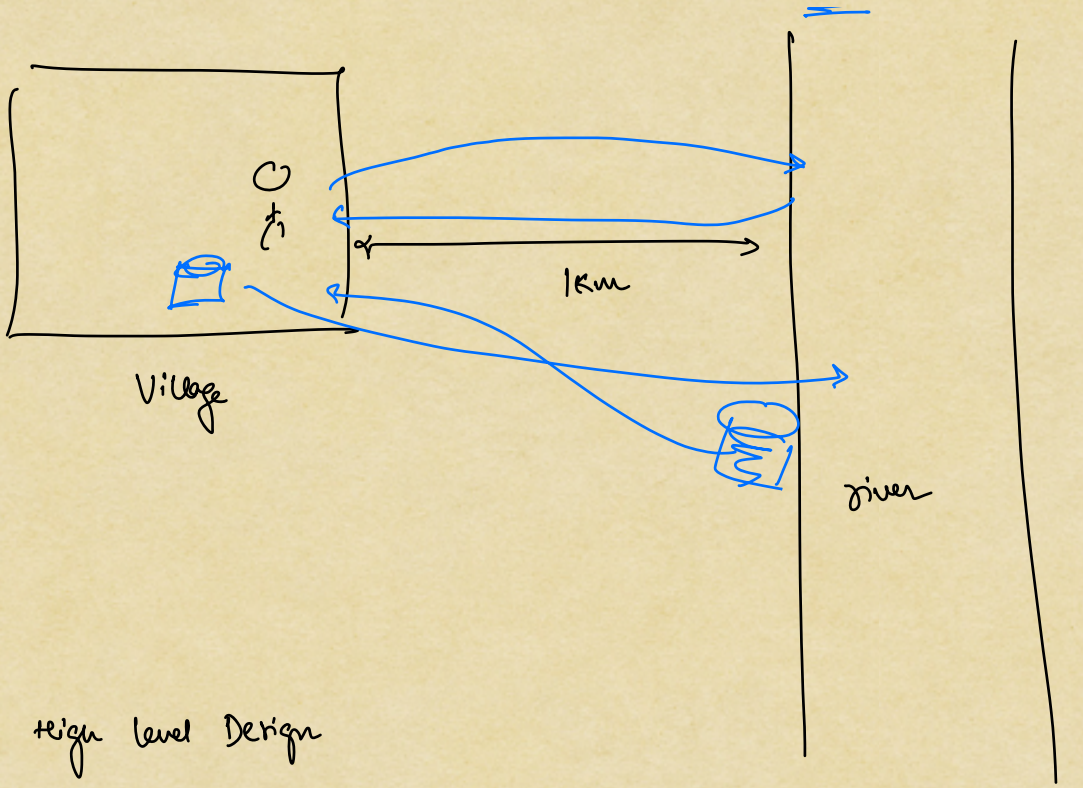
iii) Why LLD is important

iv) Intro to OOPs

⇒ Intro to LLD:

LLD → Low Level Design





HLD ⇒ High level Design

→ Overview

→ Not going into implementation details

→ Bird's eye view

LLD ⇒ Code implementation

↓

Indian term } OOD → Object Oriented Design

global term

⇒ Avg. day for an engineer :-

→ Meetings

→ Code → LLD

→ Code Reviews | Design Reviews → Readable

→ Requirement Analysis → LLD

→ Bug Fixes → Maintainable

→ KT ⇒ Knowledge transfer → Readable

→ Jira updates → Requirement gathering

→ Documentation → Readable

→ Testing - - - - .

→ Youtube | Insta | WA | group | office - .

12% of the time goes into coding.

24 ⇒ 40LPA

12% of 40 ⇒ 4.8 Lakhs

⇒ LLD helps you to make better use of the 88% of your time

⇒ If LLD is good, then,

i) Readable/understandable ⇒ code

ii) Req. gathering

[iii) Extensible

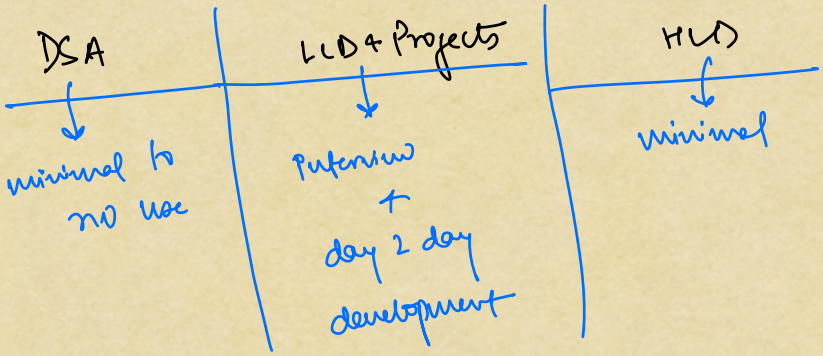
iv) Maintainable

Diff b/w extensible vs maintainable :-

Extensible : easy to add new features

maintainable : easy to keep it running as it is

ex :⇒ bug fixes, version upgrades, patch updates etc.



Theory / Syntax	Design	Machine Coding
TCS, Capgemini, Infosys, Wipro, Bosch, PwC, Deloitte, EY - - - - no design discussion - theory based Qns - syntax based Qns - 45-60 mins	FAANG, Atlassian, Intuit, — - - — — . Single live problem statement Design it end 2 end - Ask Questions - Get clarifications - No coding involved - Class diagram / Use case diagram - Schema Design - Design Patterns - 45-60 mins	Unacademy, Cred, Scaler, Swiggy, Flipkart, Doordash, Juspay. — — . - Detailed problem statement (doc) - Design - Code everything in the doc with tests - 120 mins

→ 11- fundamentals [OOPs, Concurrency, Java Advanced]

9- SOLID + Design Patterns

theory based interviews

i) Refactor

ii) Parsing UPI

iii) BookMyShow

iv) Splitwise

Design
- Use case
- class diagram
- Schema Design

+

Code
- Machine Coding

Java 11

⇒ Intro to OOP:

→ Programming Paradigms:

→ i) Procedural → C

→ ii) OOP → Java, C++, Python, Js, C#

iii) functional → Scala, Haskell

iv) Reactive → Java

: Procedural Programming

Procedure ⇒ Old age term for methods / functions

→ we arrange our code into a bunch of functionality [procedure]
→ each procedure may internally call other procedures.
→ execute starts from a specific procedure

↓

main()

⇒ Procedural does not have anything that hides data and behaviour.

for ex ⇒ C has struct [that only stores data and no behaviour]

```
struct Student {
    String name;
    int age;
    String gender;
}
```

Issues

i) struct has no methods
ii) All the variables of struct are visible to others.

```
printStudent(Student s) {
    sout(s.name)
    sout(s.age)
    sout(s.gender)
}
```

Procedures are acting on Struc.

⇒ Verb is being done on Noun

↓ ↓

printStudent Student

IRL Noun does a verb

↓

Somebody does something

Procedural - Something is being done on somebody

⇒ Object Oriented programming:

Somebody does something

In OOP we have Objects and objects contains [data + behaviour]

Object can perform actions

```
class Student {
        String name;
        int age;
        String gender;
        public void printDetails() {
```

}

}

Student0bj.printDetails()

OOP is very close to real life understanding.

+ principle of OOP

3 pillars of OOP

Principle → Fundamental foundation / concept

Pillars → Support / hold the foundation.

Principle of OOPs => Abstraction

Pillar => Inheritance Encapsulation Polymorphism

⇒ Abstraction :

+ concept of making something abstract

+ Abstract : representing something in terms of ideas

What it does → ✓

How it does → ✗✗

Hiding the implementation details.

: Scaler