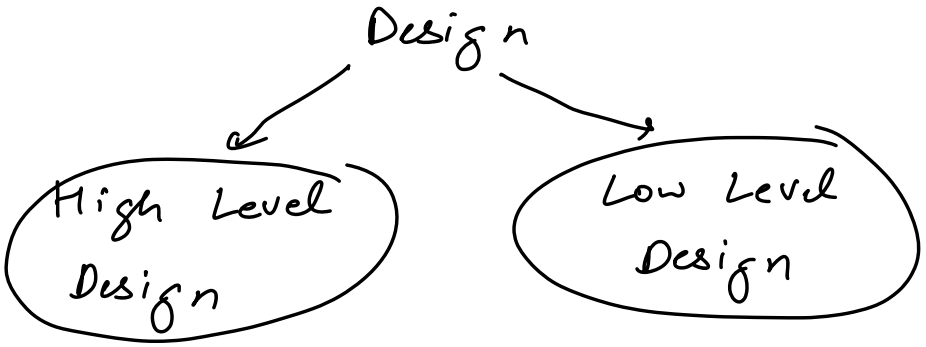
1. Good Evening
2. Lecture begins at 9:08 pm
3. Topic - Intro to LLD

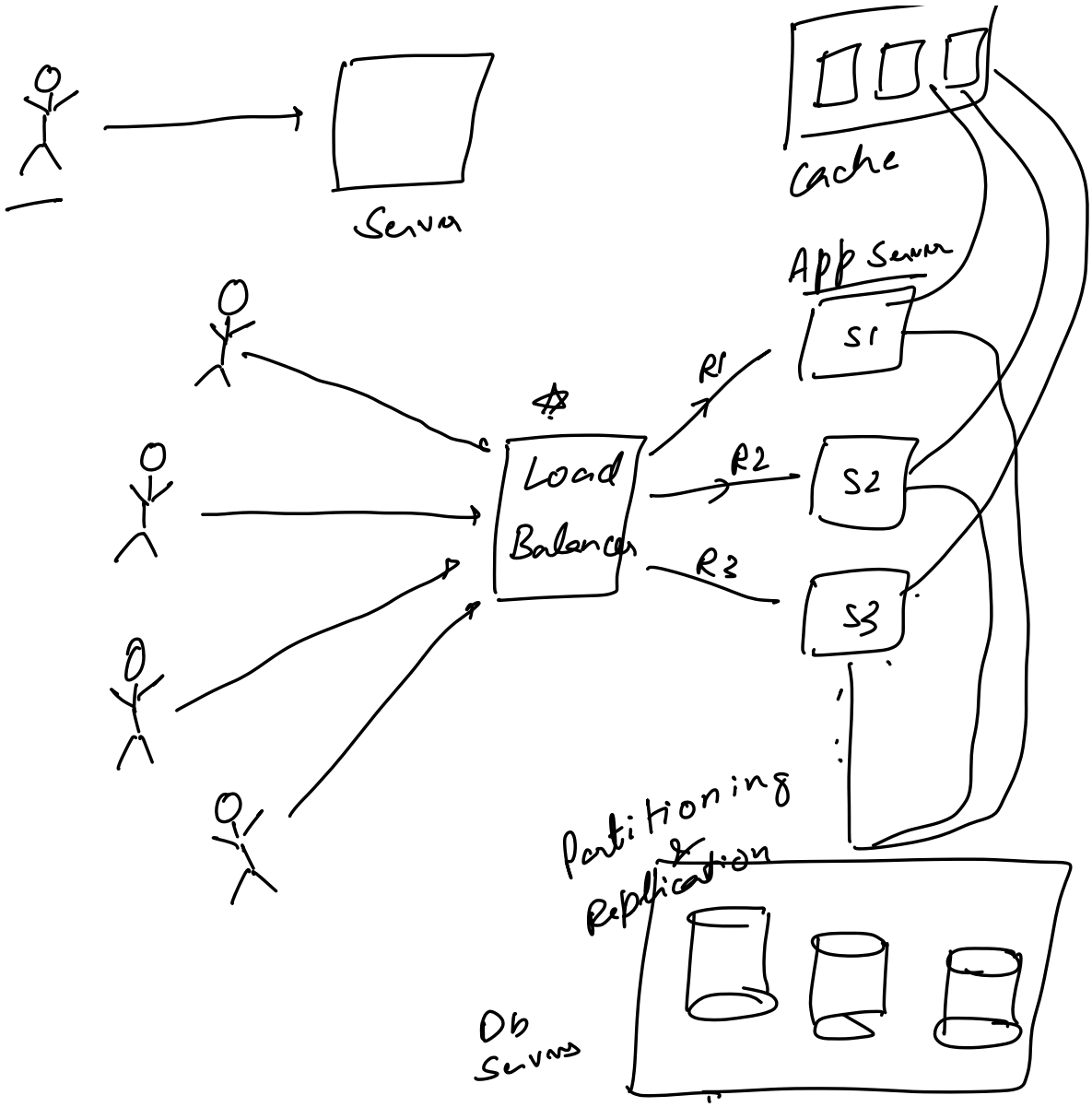
Agenda

1. Introduction to LLD
2. Why LLD is important?
3. [Structure of LLD Module.]
4. Intro to OOPs

Intro to LLD



FaceBook / Instagram



- LB
- App server
- DB server
- Caches.

High level Design

1. Overview / Bird's eye view
2. Ideas / Not implementation

- Different infrastructure layers that work together to achieve

responsibilities of the software system

Low Level Design

* Components → Db, Cache, App Server, Load Balancer are computer are running different software.

LLD * Details of the code that is running.

- → What are the classes in system?
- → What are the attributes of classes?
- → What behaviour / method does the class have?
- → How classes will interact with each other?

LLD = Object Oriented Design

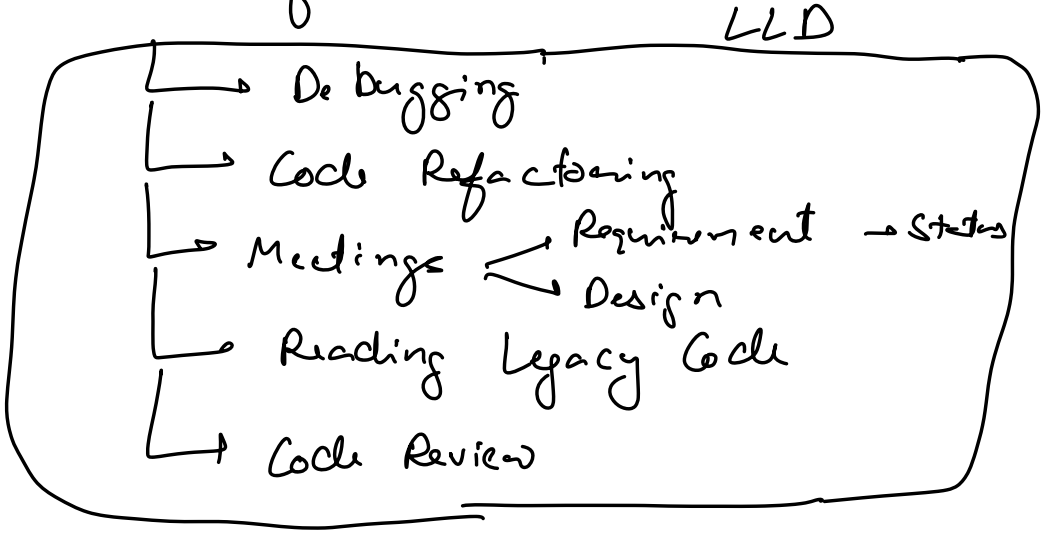
Why is LLD important?

Part 1

* Most startups & product-based companies → 1. Machine Coding
2. Low Level Design Discussions.

Part 2 → 12% of time is actually spent in coding

→ 88% of time



LLD Curriculum

1. OOPs [4 lectures] ✓

2. [SOLID Design Principles] [2 Lectures]

- Extensible : Allows us to add new feature to system
- Maintainable
  - Allow a system to keep running as it is.
    - → Dependencies
    - → Bugs

3. Design Patterns (5 Lectures) ✓

- Creational (5) → 2 Lectures
- Structural (3) → 2 Lecture
- Behavioral (2) → 1 Lecture

4. UML Diagrams [1 Lecture]

5. Practice] → 8-9

Entity [1. How to approach LLD Problems
Design a Pen]

Game [2. Designing Tic-Tac-Toe & Code TTT

Management [3. Designing a Parking Lot (Schema Design) & Code

Sys Design

- 4. Designing Book My Show & Code
- Designing a real life
  - 5. Designing Splitwise & Code
  - 6. Designing a cache & Code
    - Designing engineering problems → e.g. Crawler

Parking lot → MVC

Splitwise → Spring Boot, APIs

BookMyShow → Concurrency.

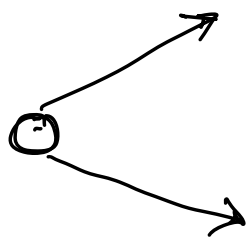
- ☆ DNS Details
- ☆ 4 Leetcode Semaphore

Intro to OOPs

Break → 9:55 - 10:00

Intro to OOP

Programming Paradigms



1. Procedural → C , SQL
2. Declarative → HTML , SQL
3. Object Oriented → Java, C#, Python, C++, JS
4. Functional → Haskell, Scala
5. Reactive → Java

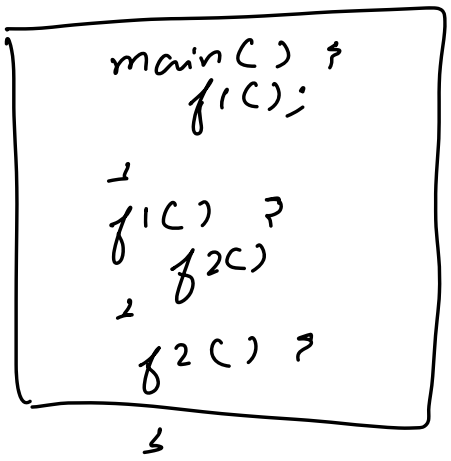
Procedural Programming

- Procedures / Functions / Methods

A set of instructions put together.

* We organize code into a bunch of procedures
* Procedures invoke each other

* Execution begins from a special procedure called main.



[Most prominent problem about procedural programming]

Programming Languages are created to help developers express logic [in machine understandable manner] to solve business problems

What are you doing right now?

→ Sawan is taking class

→ Sunil is learning OOPs

→ Divya is trying to stay awake

→ Shubham is making notes

[Subject + verb]

Active voice is more common way of thinking

↳ Human way of thinking focuses more on the noun/entity (subject & less on the verb

But let us take an example of PPL

```
void printStudent (String name, int age,
                   String gender) ?
{
    Syso (name)
    "    (age)
    "    (gender)
}
```

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

void printStudent (Student s) {
    Sysо (s.name)
    "  (s.age)
    "  (s.gender)
}
```

printStudent + Student = Passive Voice

Printing happened to student =

Verb

Human focus more on nouns.

OOP languages

s printed himself

s.print() ——→ natural { closer to real-life }

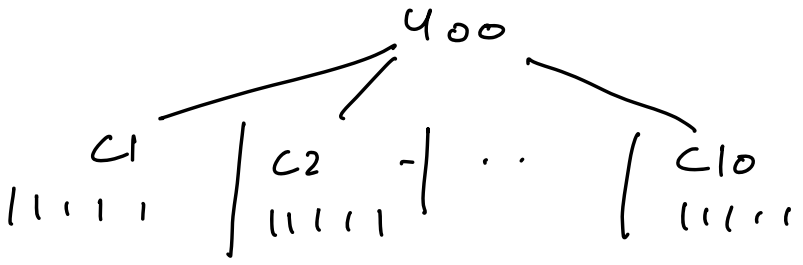
print (s)

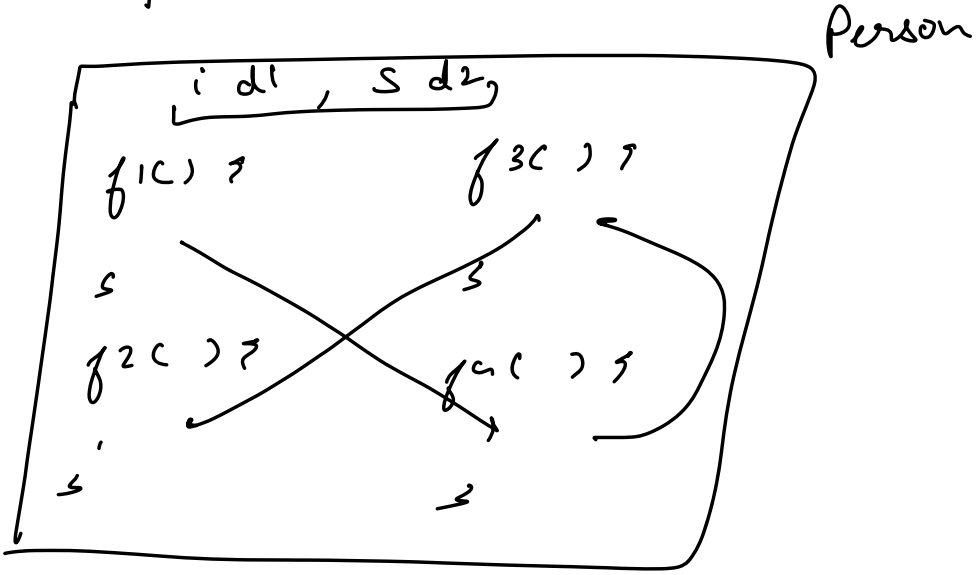
Cons of Procedure Programming Languages

1. Passive voice → not natural
2. Spagethi Code → God Module — Lack of modularity

↙ Difficult to distribute to a team

↘ Difficult to maintain & extend — Readability





☆ Difficult to break up the task for different members of the team → Extensibility } Problems

- ☆ Readability Problems ⟵ Maintainability
  - Lack of Modularity
    - Not close to real-life way of thinking ✓
    - Lack of modularity ✓

Object Oriented Programming (Nouns → Modules)

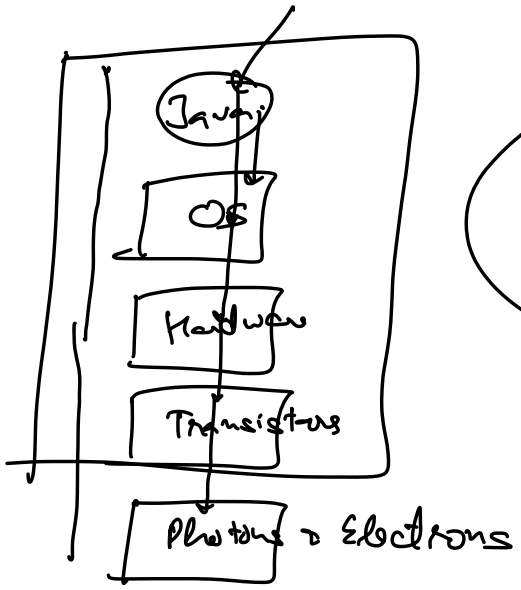
- 1 principle → Abstraction ✓
- Pillars of Object Oriented Programming
  - Encapsulation ✓
  - Polymorphism
  - Inheritance
- 2 keywords, Initial Code

Abstraction

- Summary, TLDR, essence

↳ No focus on details, focus on usage.

→ [To use a thing, we need to know public API, & not implementation details.]



I want to learn in depth

↓

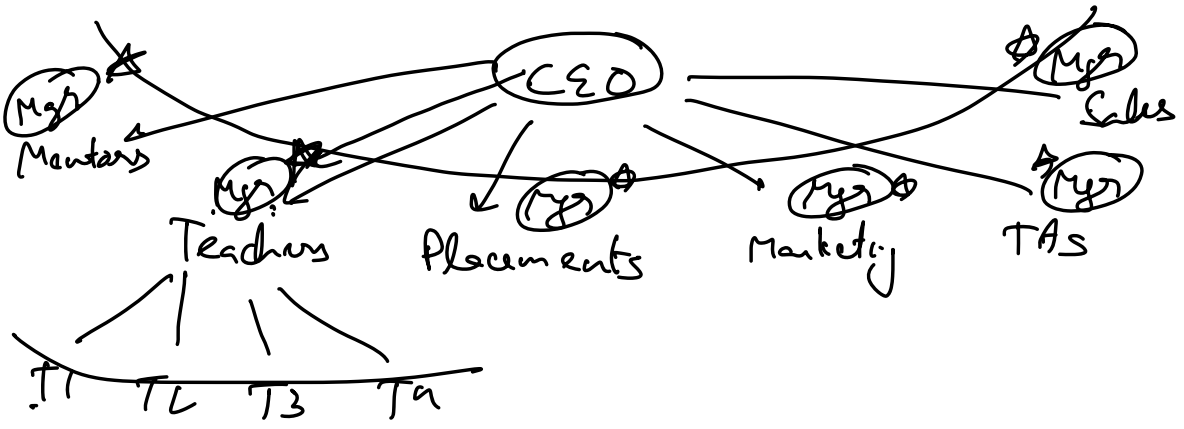
[Learning { Thinking at appropriate layer]

Recursion

↳ fib(n) ?

int fnm1 = fib(n-1)

→ Think about the details that help you use a system [don't think about how system internally works]



Encapsulation → Capsule

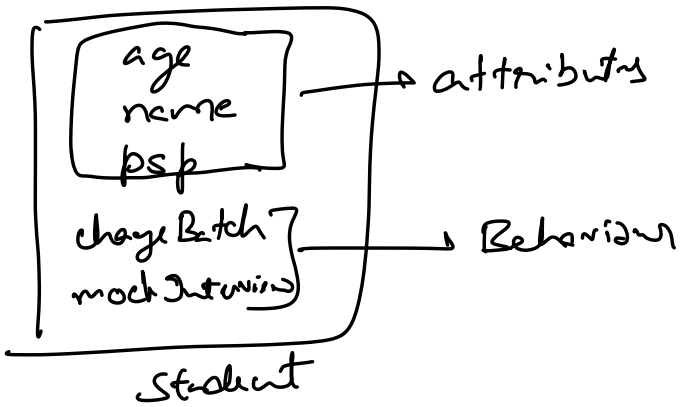
Group together

Protect it from environ-ment

Encapsulation in OOPs [Grouping data & fns. in classes]

1. Group together attributes (adjectives) & behaviours (verbs)

of entities
nouns



2. Attributes & behaviour of entity are protected from other entities
   - access modifiers

Real-life Modularity

CLASS
- Blueprint for an entity
- Expression of concept.

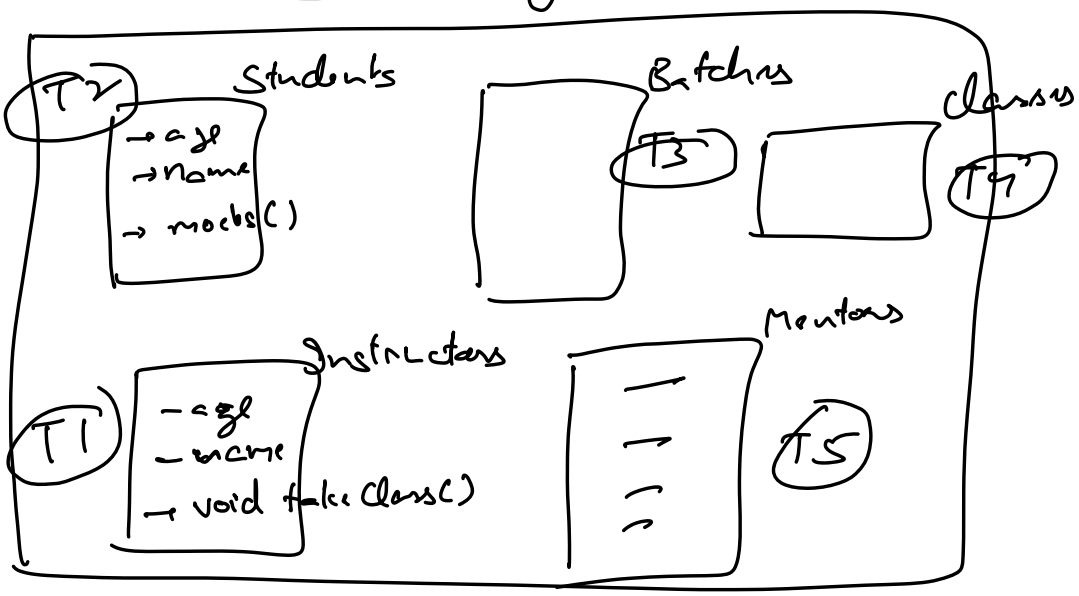
```
class Student {
    int age;
    String name;
    double psp;

    void mockInterview() {
    }
}
```

s

Object → Real Instance of a class
→ Occupies memory
→ Each object is separate from others.

OOPs → Modularity



Team lead

Each class is a separate module

↳ Divisible in a team?

└→ Understandable]

S. printf()

OOPs → Natural way of expressing logic

Classes & Objects → Define the system in terms of modules

Teamwork

Understandable

Abstraction	Encapsulation
To use something, we don't need to know inner details.	putting data & methods together in a class