LLD1 : LLD 101

Intro to OOP

AND

Intro to LLD

OOP Part 2

Agenda

- What is LLD
- Why learn LLD
  - Interviews
  - Career
- LLD at Scaler
  - # of classes
  - curriculum
  - Takeaways
- Intro to OOP

→ Key terms of OOP

- → Classes
- → Objects
- → Inheritance
- → State

What is LLD

LLD: Low level Design

→ High level

→ Bird's Eye View

→ Don't go into details

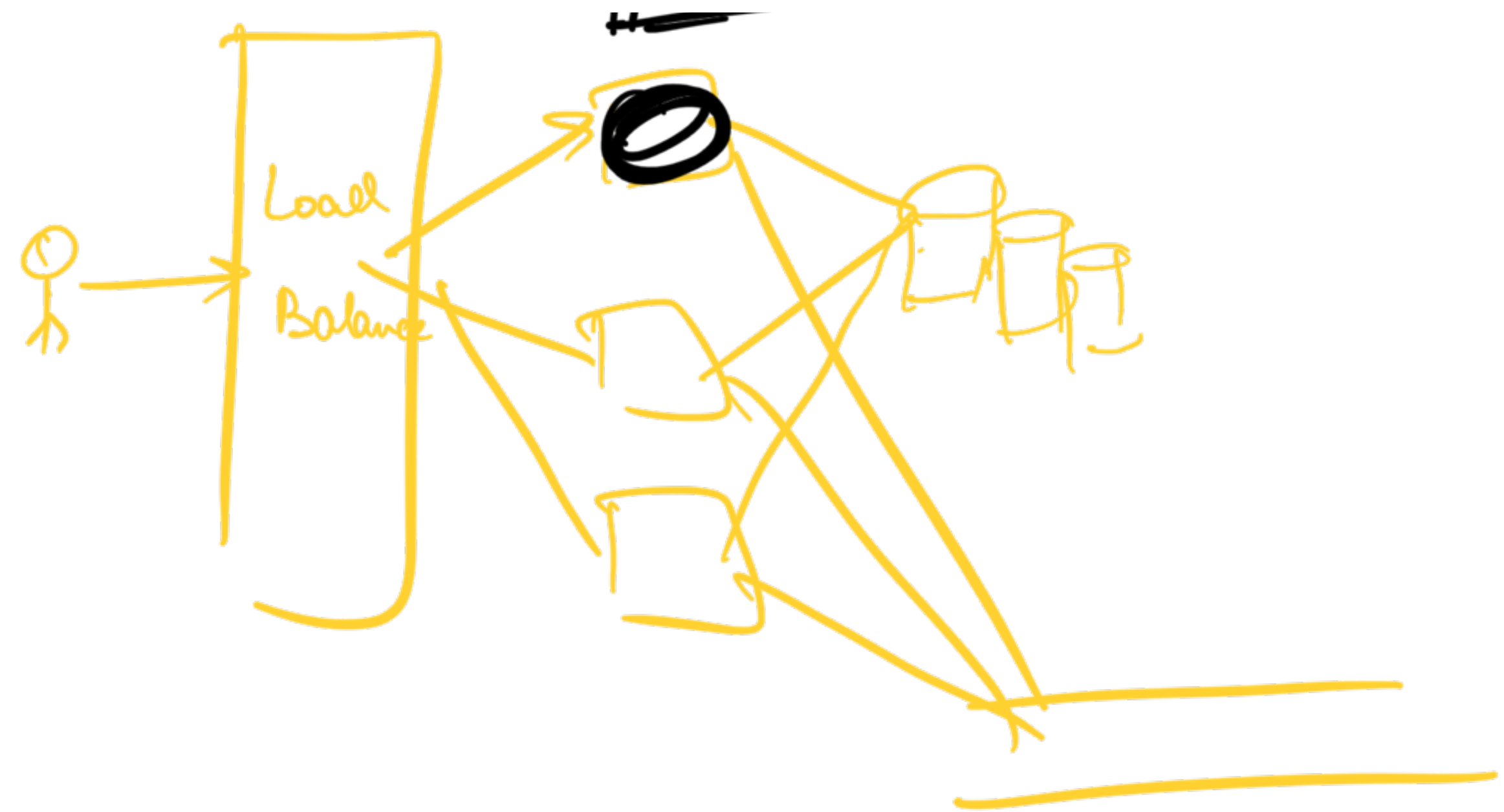
→ Abstract View

HLD

Overview of the working of a complete System

→ Infrastructure Layers

App Servers


Load
Balance

Going into Detail
of the implementation

→ How we Code → LLD

⇒ Details of Code

→ Business logic ←

→ Classes ←

→ f^n ←

→ Packages ←

→ Interaction b/w code

→ Design of implementation of a software System

PM Interviewer

→ PM

→ Business

→ CEO

→ Requirements Gathering

→ Clarifying Gathering

→ Design (Flowcharts, Documentation)

→ Use Case Diagram

→ Class Diagram

→ Schema Design

→ Code

→ Testing

Why learn LLD

≤ 12% writing code

2 - 3 hrs / 8 hrs ≈ 25-30%

Blocker

Meetings → Requirement Gathering
Planning → Design Discuss

⇒ go throughl code ⇐ Fix Bugs ⇒ Debugging

Understand code

Coffee / Chai / Sutta

Debugging

Prevent future

Testing

Reading others' code

Reading ⇐ Code Reviews ⇐

LLD ≈ 80% of your time easier

⇒ Promote faster

0-2 year

SDE1 ≈ Understand how to structure a project-
work with dependencies

2-5yrs

SDE2 ⇒ You will be able to independently design
a full enough feature with minimal

help from seniors

SDE 3

5-8 yrs → You will be able to handle a large/ Cross team project and guide others On their Skills via Code Reviews

Interviews

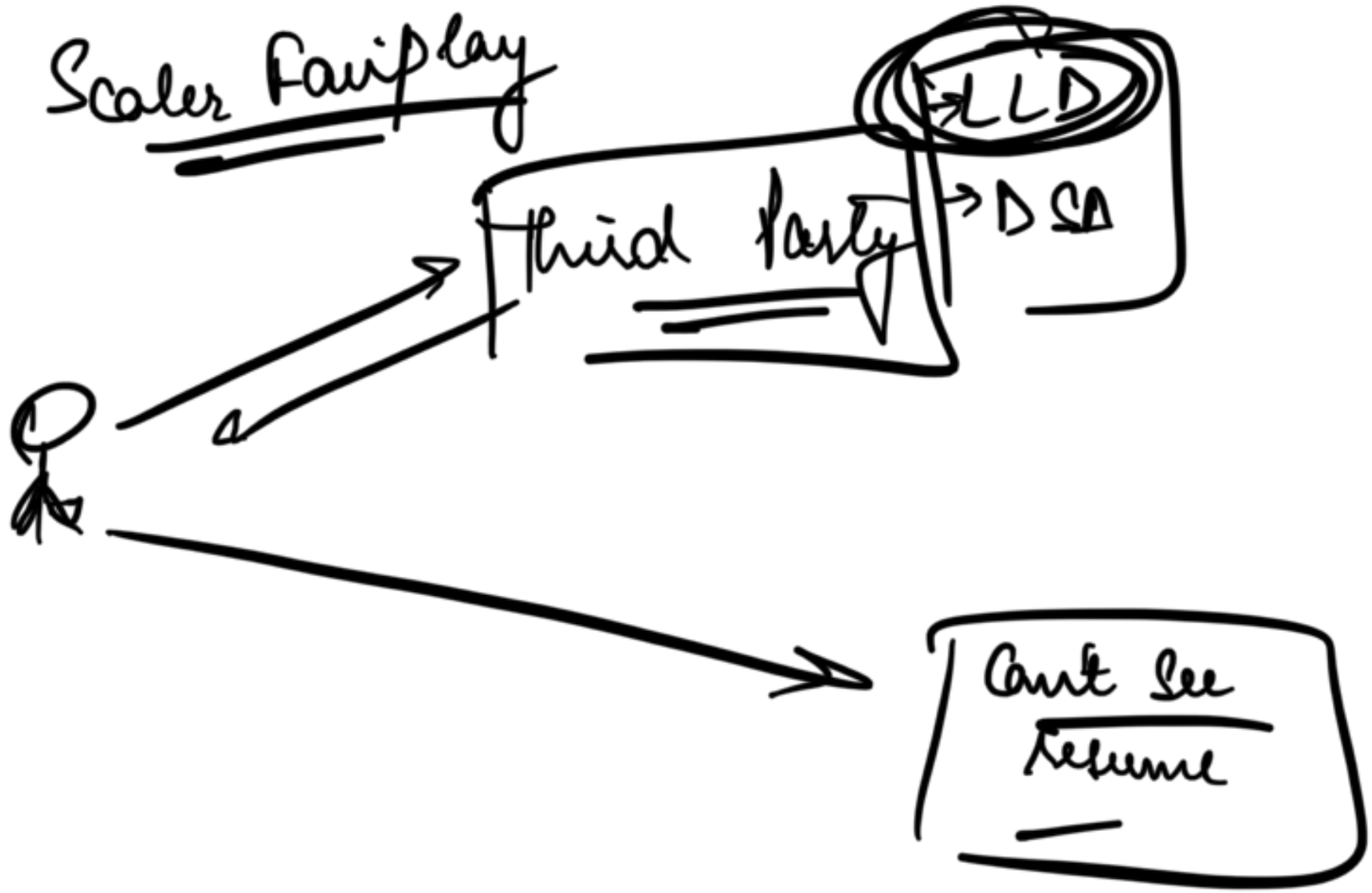
Top Startup

→ FK, Swiggy, Phonepe, Groww

→ LLD → Spring

→ JS

SDE1 : LLD | HLD



MNCs (MAANG, OS Startups)

Class Airbnb

SDE 2+

Types of LLD Interviews

1. Theoretical — 30 - 45 Mins

→ Direct Q^n

↳ Design Pattern

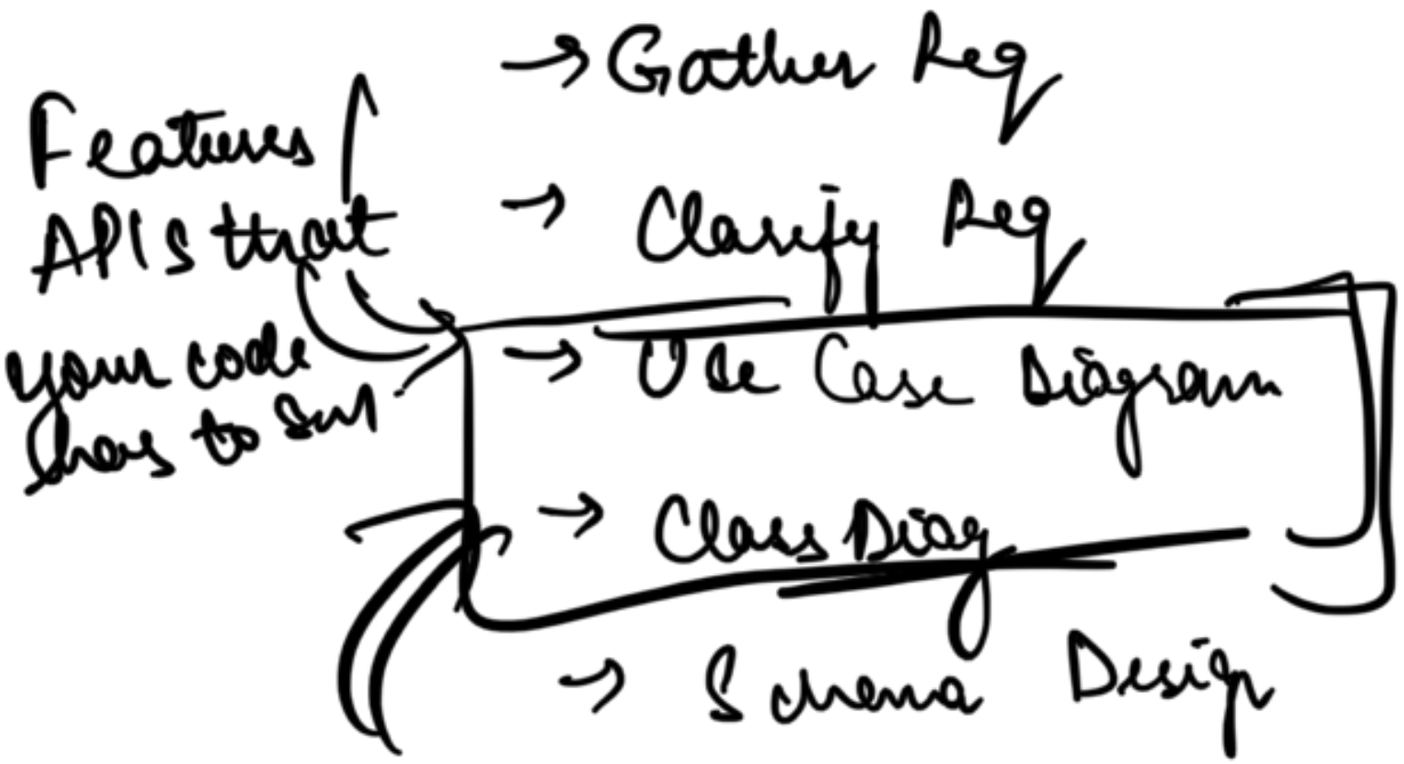
↳ OOP Concepts

2. LLD Round / Design Round — 60 Mins

(2) LLD Round (Design round)

→ Very abstract q^n : Design Pen / Design Zoomcar

Design Snake and Ladders



You might be asked to code a single class

(3) Machine Coding Round (E2E working code is expected)

→ Gather Req X

→ Clarify Req

→ Use Case Diag

→ Class Diag

→ Schema Design

→ Code (Might / Might Not include Test Cases)

2.5 hrs

30 Min

→ Testing Skills ⇒ (✓)

→ Concurrency Skills ⇒ (✓)

1.5 hrs

→ Ask q^n regarding code ⇒ 15 Min

→ Hidden DSA Problems ⇒ Dunzo ⇒ 60 Min

LLD at Scaler

16-18 classes

- 1 → Intro to LLD and OOP → 2 classes
- 2 → OOP
- 3 → Design Principles → SOLID

Head First DP

→ Philosophies

→ Values

→ Code to interface

4
5
6
7
→ Design Patterns → Creational (2)
Structural (1)
Behavioural (1)
→ 10 DP

8 → UML Diagrams & Schema Design

9 → How to approach LLD problems in interviews

+

Case Studies → Entities → Pen

Games → TicTacToe

OOP Language

LLD of BMS

Concurrency:

BMS concurrency

Email Campaign Mgmt Sys

→ Multithreading
→ Sync
→ Locking

Real life System

→ Snakes & Ladder

→ Book My Show

→ Parking lot

→ Matchmap

→ Splitwise

CS Problems
→ Scheduler
→ LRU Cache

DSA + LLD

Assignments

① MCQ → Test your understanding

② Design → Peer + TA

③ Code → dump a few classes / fn

10:05 → 5 min

10:10 min

Break

SEM → Parking lot

→ LLD

TL

6 type

→ HLD

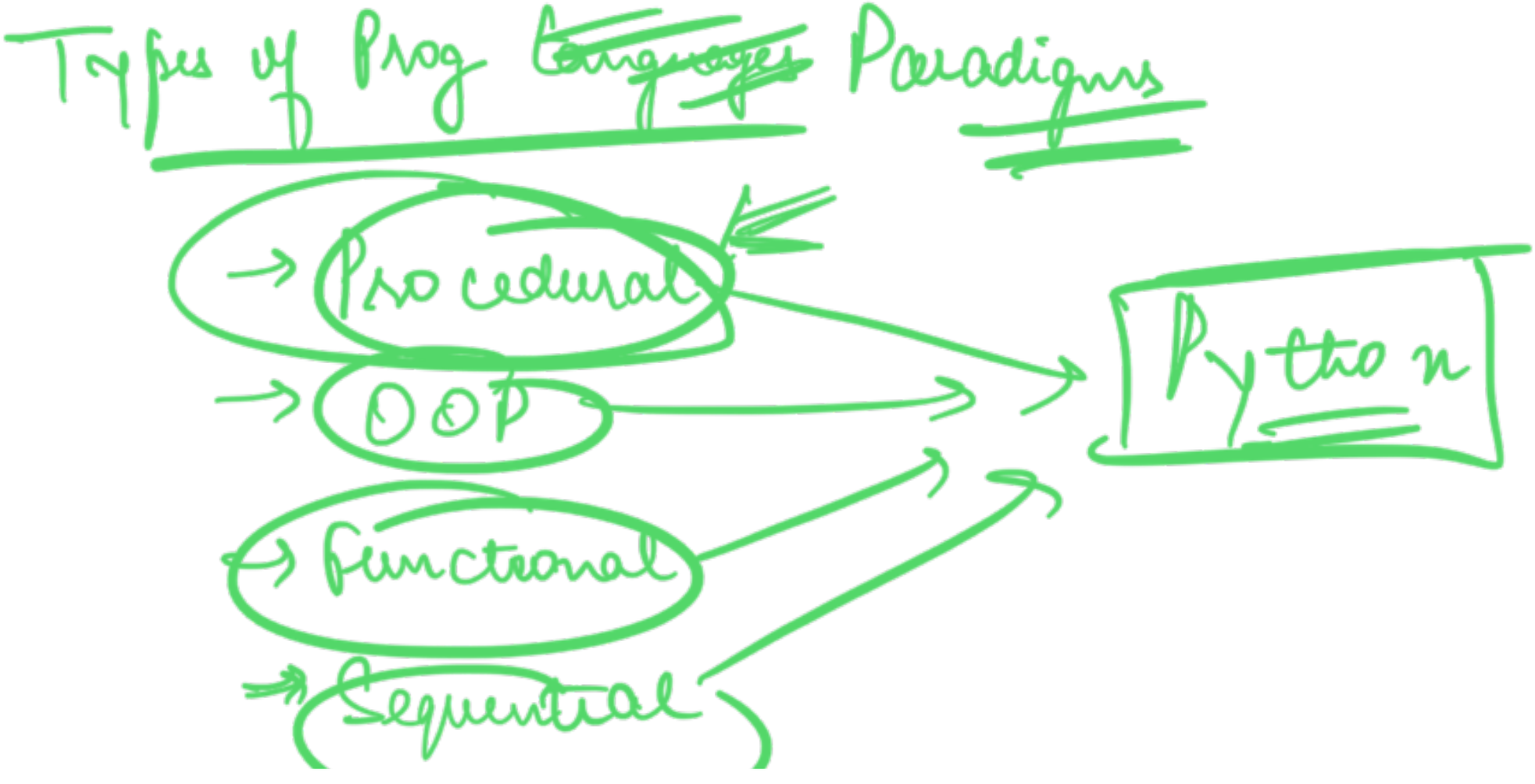
→ 2 Mgmt Flow

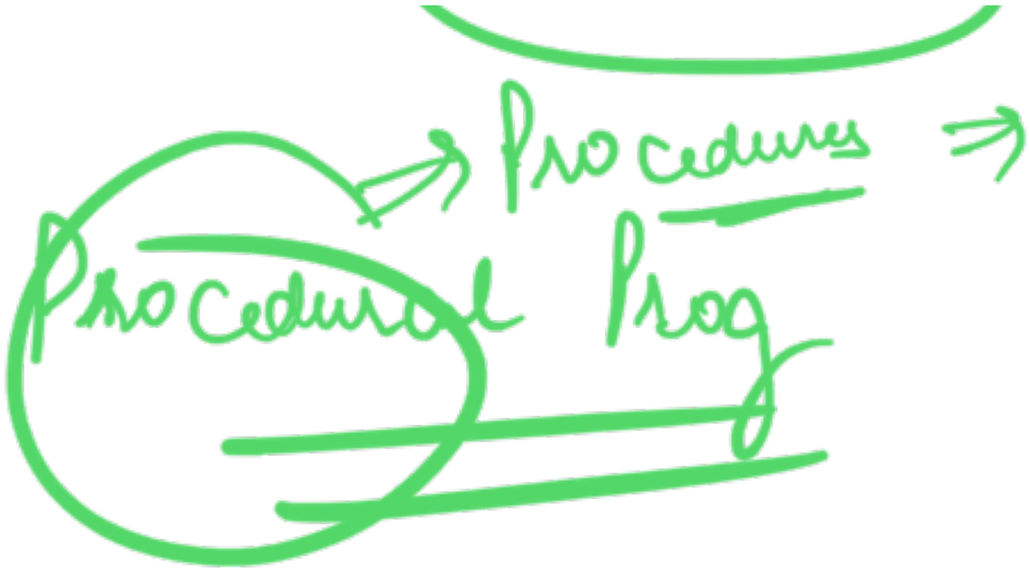
L6

SDE4

Prog Languags

Types of Prog ~~Languages~~ Paradigms



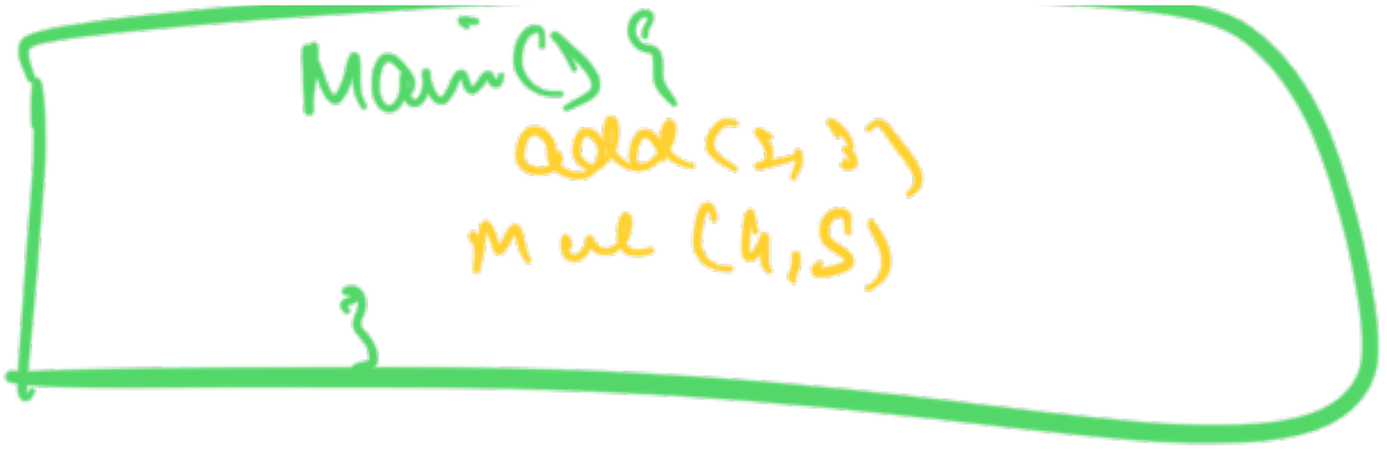


→ Code Works by calling multiple procedures One after the other and the code execution starts from a special procedure (main())

→ procedure: take a set of data items
↳ do something using those data
↳ give a result

add (int a, int b)
return a+b

mul(int a, int b)
return a*b



Real life

Always an entity that Makes a behaviour

Entities doing Something which leads to an action

Sub + Verb

Entity + Action

We think of real world as multiple

doing multiple

In a complex software system as well,

there are multiple entities

can perform multiple actions

whole software system

⇒ sequence of multiple entities perform

System

Humans in their real lang are accustomed to think of the world as entities doing things

even software systems be structured ⟸

⇒ OOP came into existence

Procedural Prog

⇒ procedures

⇒ procedures

⇒ I have to use these procedures in a particular order to get desired results

Con

⇒ C

- for a large enough system it

Can be difficult to manage

OOP

Sub + Verb

Entity

↳ Something in the requirements that can perform multiple action (verbs)

T int, String, boolean

print(int)
print(String)

print(___)
int ⇓
⇑ String

printInt( )
printStr( )

int.print

abs(int)
absL(long)
absF(float

Sankalp → walk

Sankalp. walk( )

Puppet System ← Manage Sankalp

Sankalp

Puppet

(

— walk Sankalp (Sankalp) {

Threads that

control the puppet

}

Sankalp.walk()

Procedural

Entity is nothing but a set of data item

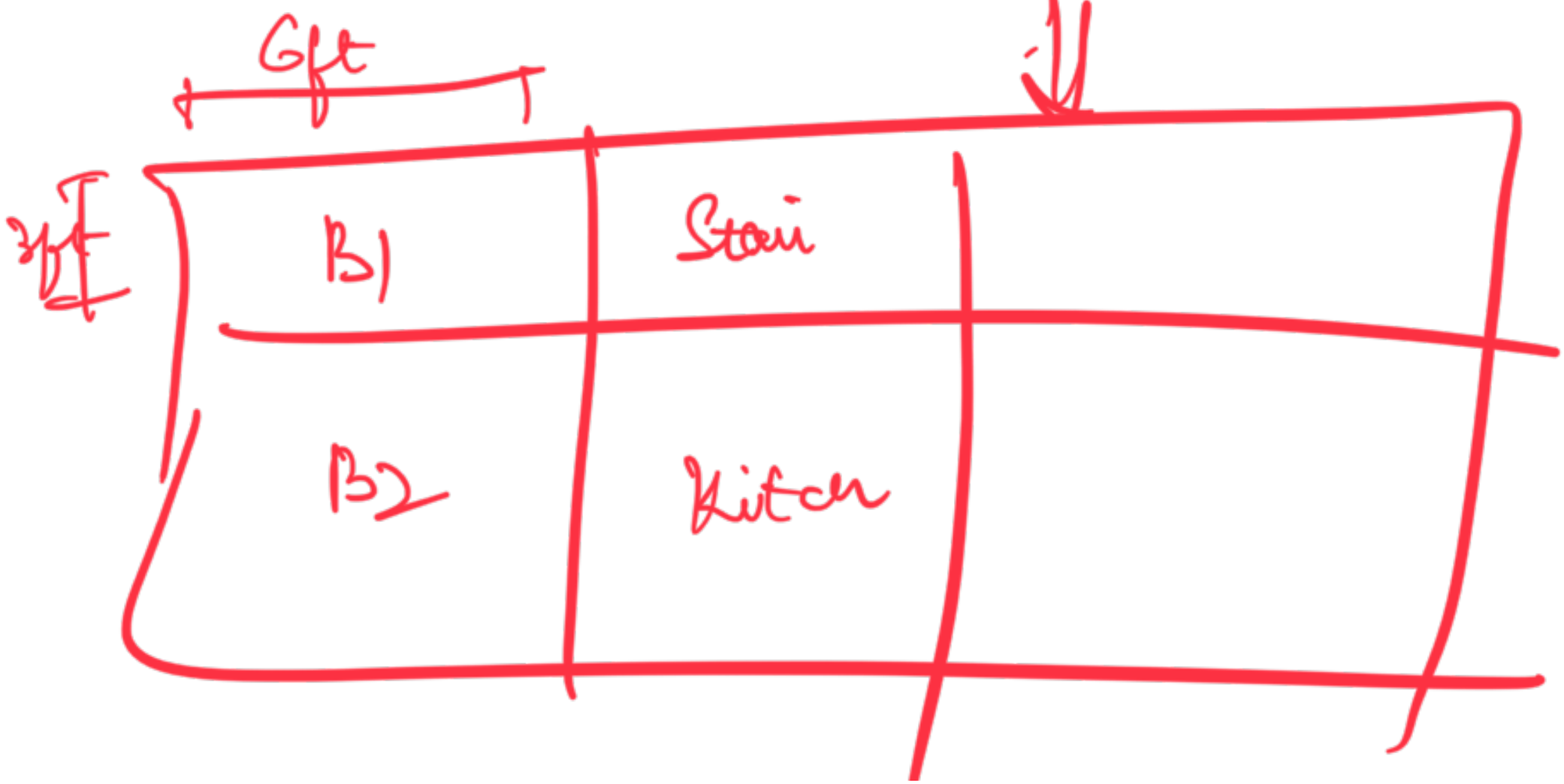
OOP

→ Entities are the central part
→ Entities decide their Behaviours
→ For prog they make it easier to make sense of a big system

Key Terms of OOP

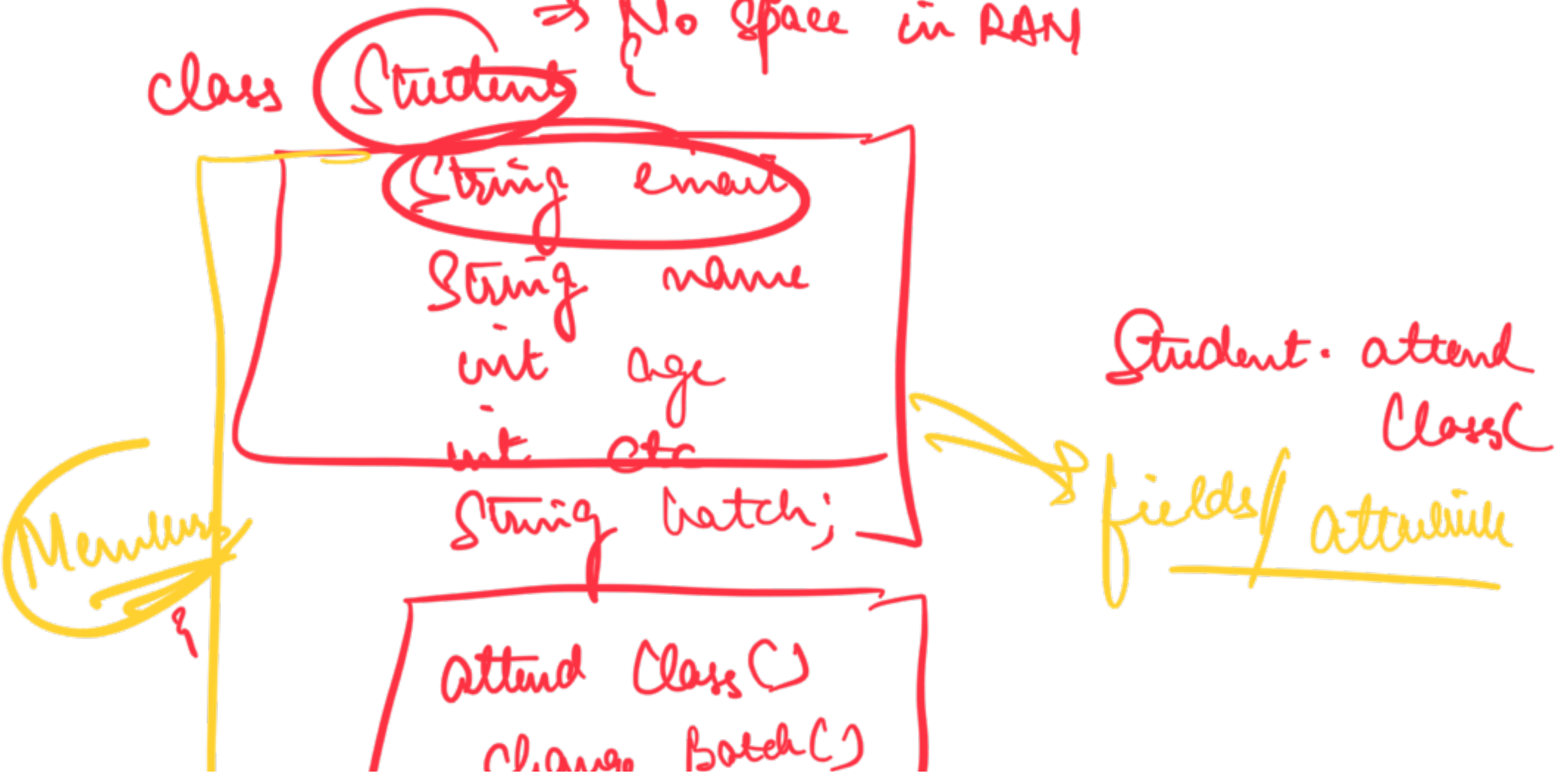
Build a House
Architect → Blueprint
(Design of the House)

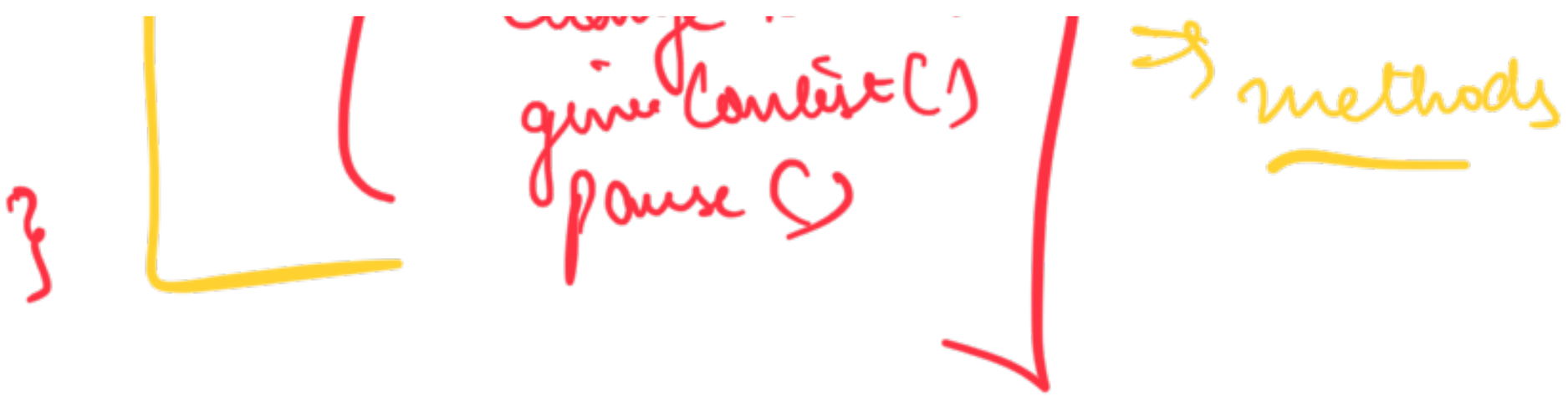
Entities → Classes → Blueprint of an entity



Class diff entities into groups that have
Similar behaviour & attributes

→ Class has no physical existence
→ No space in RAM





Object ⇒ real instance of an entity

gautam.pause()

gautam.attributes

⇒ Object will take space in RAM.

properties

Fields / Attributes : Different Data points that define a particular class

Methods : Diff actions that can be taken by the class objects.

gautam = new Student()

rahul = new Student()

gautam.age = 18

rahul.age = 24

gautam

{18, Rahul, Null}

gautam.Name = Rahul

Rahul

{Null, 24, Null}

Rahul.Email = xyz

{Null, 24, xyz}

At any particular time, the value of every attribute of a particular OBJECT

→ State of the object

July
Intermedia
Advanced

CEO
VP
VP
VP

Even when we think of diff entities in
a software system, we try to categorize
them

Animal

Mammal

Reptiles

Bird

Fish

↳

OOP

→ allows us to create a hierarchy

between classes

⇒ reuse behaviour and attributes