LLD Basics

↳ Low Level Design

support@scaler.com

Agenda

→ Intro

→ When will LLD ?

→ Structure of LLD classes

→ What is LLD ?

→ Why learn LLD ?

↳ Interview point of view

first class of LLD

Module

↳ Job (Career)

6-7w

→ Intro to OOP

→ Procedural V/S OOP

→ How OOP came into picture

→ Basic terminologies

↳ Class

↳ Object

↳ State

↳ Inheritance

→ Pillars of OOP

↳ Abstraction

↳ Encapsulation

↳ Inheritance

↳ Polymorphism

→ Case Study

Java ⇒ Jar ⇒ Every other

↳ C++

↳ C#

↳ Python

↳ TS

↳ TS

↳ Golang

⇒ Bridge written Material

Naman Bhalla

Ex SWE @ Google

Cure fit

Swiggy

2 years

@ Naman

+91-9996203771

1 Month

Beg

3-4 Months

DSA

Months

OS | CN | DB

1.5 Month

LLD

6 – 7 weeks

Structure of LLD curriculum (18 – 20 classes)

2 – 2.5 hrs

- OOP (2 classes)
- Design Principles (1 class)

(SOLID, Code to interface) —

2 classes → Creational

1 class → Structural

1 class → Behavioral

→ Design Patterns (4 classes)

(5 + 2 + 4) → 11 Design Patterns

→ UML Diagrams (1 class)

→ Schema (1 class)

→ Project Structure & Design (1 class)

→ Concurrency (Multithreading, Mutex, Threads, IPCs, Semaphores, 1 class.

→ Unit Testing (1 class) → SyncBlock → TDD → BDD

→ How to approach LLD Problems

→ Case Studies

Java 11

→ Pen & Entity

→ TicTacToe } Games

→ Snakes & Ladders

→ BMS

→ Parking lot } Real Systems

→ Mail client

→ Splitwise

Code a car rental system

IMP

LLD

→ Dist Cache

→ Message Queue

} Eng Problems

① Java

② Python

Singleton

↳ Multithreading

↳ Serialization

→ Head first DP

→ Refactoring. Guru

→ Effective Java

→ Hacker News (News. Y combinator .com)

What is LLD

LLD → Low level Design /
Object Oriented Design

Design → Planning
↳ Share the plan with team mate
↳ Plan what to code
↳
↳ Split the work amongst team member

① Gathering Req

② Clarify Req

③ Code Diagrams (UML)

↳ Class Diag

↳ Activity Diag

↳ Use Case Diag

④ Design Doc

↳ Team Members

↳ Architect

⑤ Code

Design of how to implement a Software System via code

↳ Class

↳ Package → Components / Module

↳ Methods

↳ Interaction amongst Classes

→ Persistence into DB

↳ Schema of DB

MS → HLD

LLD

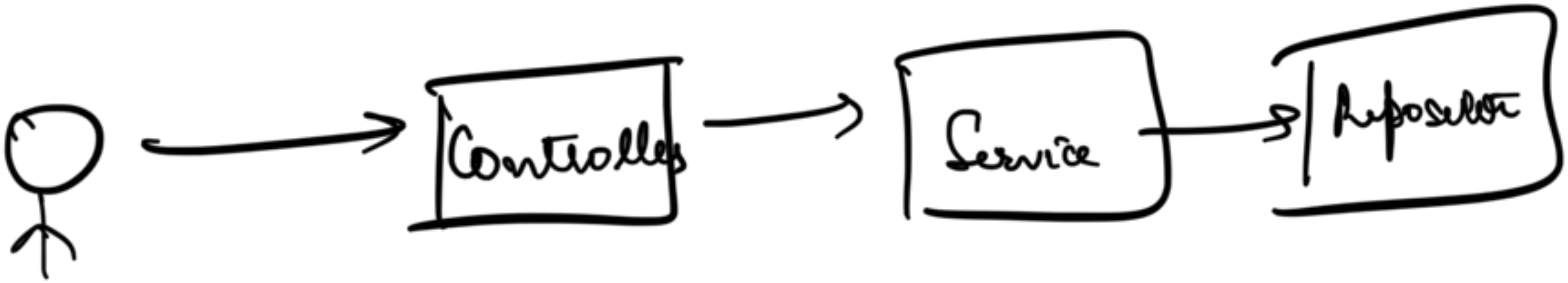
Divide of code

v/s

HLD

← Microservi

diff infras tructur layer



34 + 29 + 13 → 67%

<3 hrs =

100x

75%

Gain
Breaking

12% of their day writing code

→ Meeting ⇒ Gather Req / Understand Req / Done

→ Analysis ⇒ Monitoring ⇒ Refactoring

→ Debugging ⇒ Read Code

→ Error Fixing ⇒

→ Code Review ⇒ Reading Code

→ Maintenance ⇒ Read Prev Code + Adapt

→ [redacted]

→ [redacted]

→ [redacted]

→ Testing ⇒ Read Code + Evaluate it against Cases

→ Designing

→ Changing / Refactoring Code ⇐

→ Documentation

→ Stack Over Flow

SST.

Reading Code as an Engineering Activities

→ Doc

→ Code of others

↳ Ref

↳ Code Review

LLD

↳ Maintenance

↳ Debug

→ Understanding about what code

LCS allows you to write code in such a way that makes what you do 98% of your time easier.

How is LLD asked in interview

FAANG

MNCs

Startups

SDE1

LLD/ Machine Coding Round

0-2 YOE

SDE2

2-5 YOE

SDE3

5-10

1 DSA

2 HLD

LLD

(40-50 mins)

LLD

Machine Coding

2-2.5 hrs

A limited set of requir

Gather

Clarify

UML Diagrams

Schema Design

Code

(Working Code)

Career

Eng Ladder

CTO / HoE

Level

Skills

SDE1

SDE2

SDE3

API Design / Coner

- Only when they demonstrate all skills of other pos

6-12 months

5 yoe → SDE2 → 6 months → SDE3 → 1.5 yr → SDE4

SDE3

Intro to OOP

Paradigms of Prog languages

- Functional
- Procedural
- Object Oriented
- Reactive
- Aspect Oriented

Procedural
OOP
Functional

procedural

Group of instructions to take a particular action

Languages that run a program by running 1 procedure at a time and starting from a main procedure

A program is nothing but diff procedures running in a particular order to perform the desired actions.

CH

PROCEDURES (INPUTS)

CODE THAT TAKE ACTION

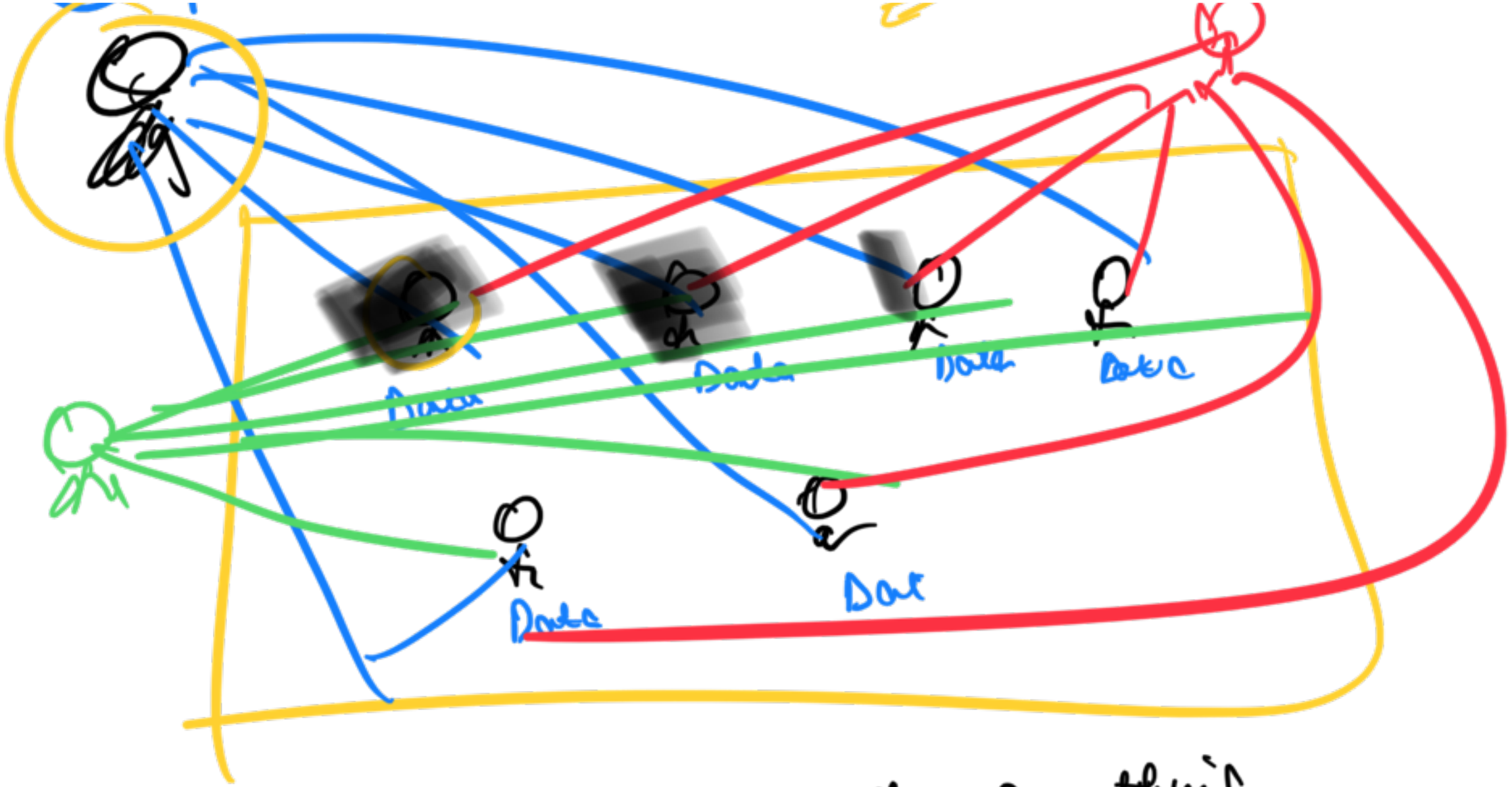
OUTPUT ←

OOP / Non OOP

Depending on how is data supported,

lang is OOP / Non OOP

PH procedu



To get humans to do anything

PH has to take action

Nango

Constructs Human

name
age
# of legs
weight
...

Puppet Master

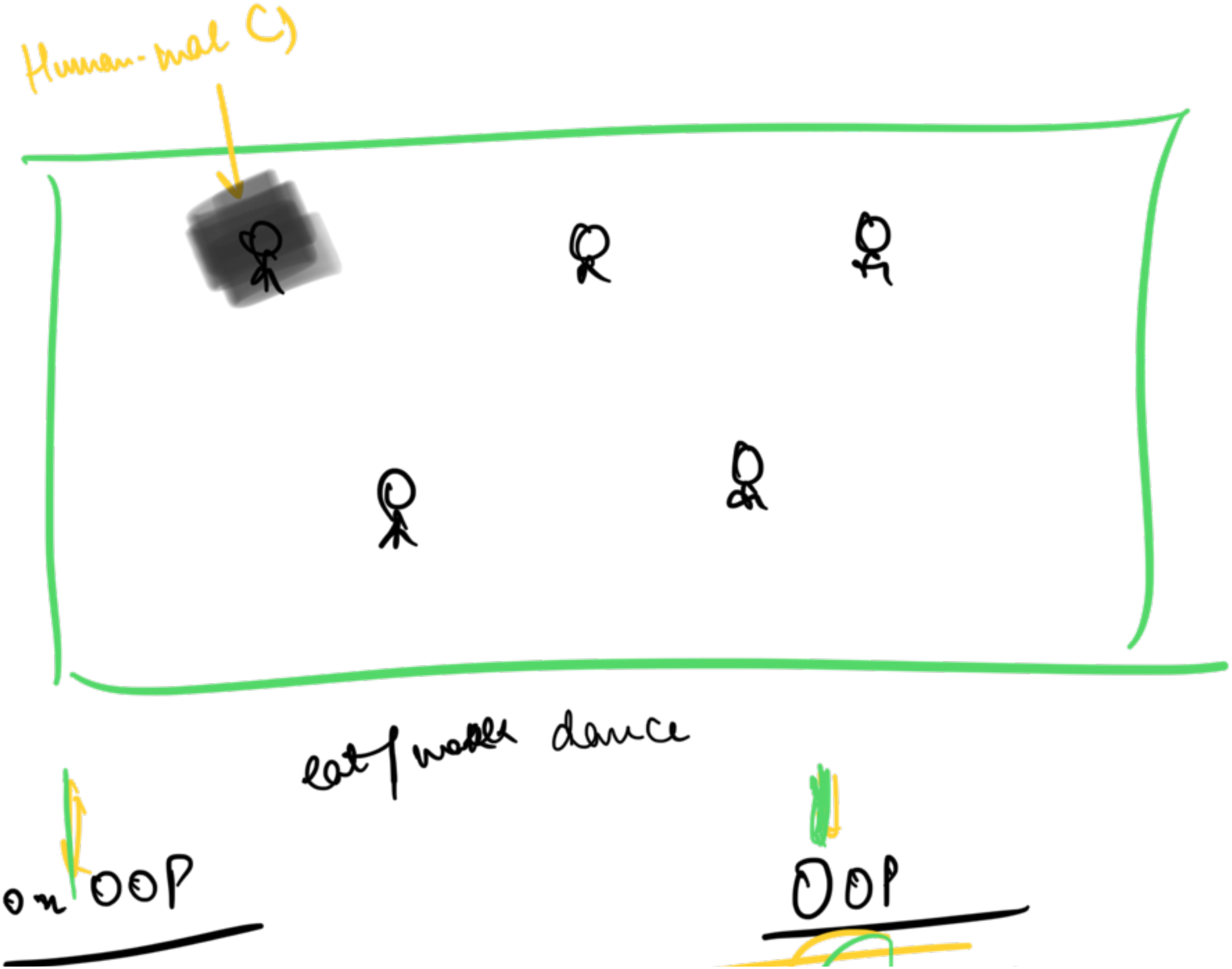
walk Human ('Human')

eat Human ('Human')

)

)

→ In OOP Language → Data Items are fully independent entities


Human-mal C)
eat / walk / dance
Non OOP
OOP

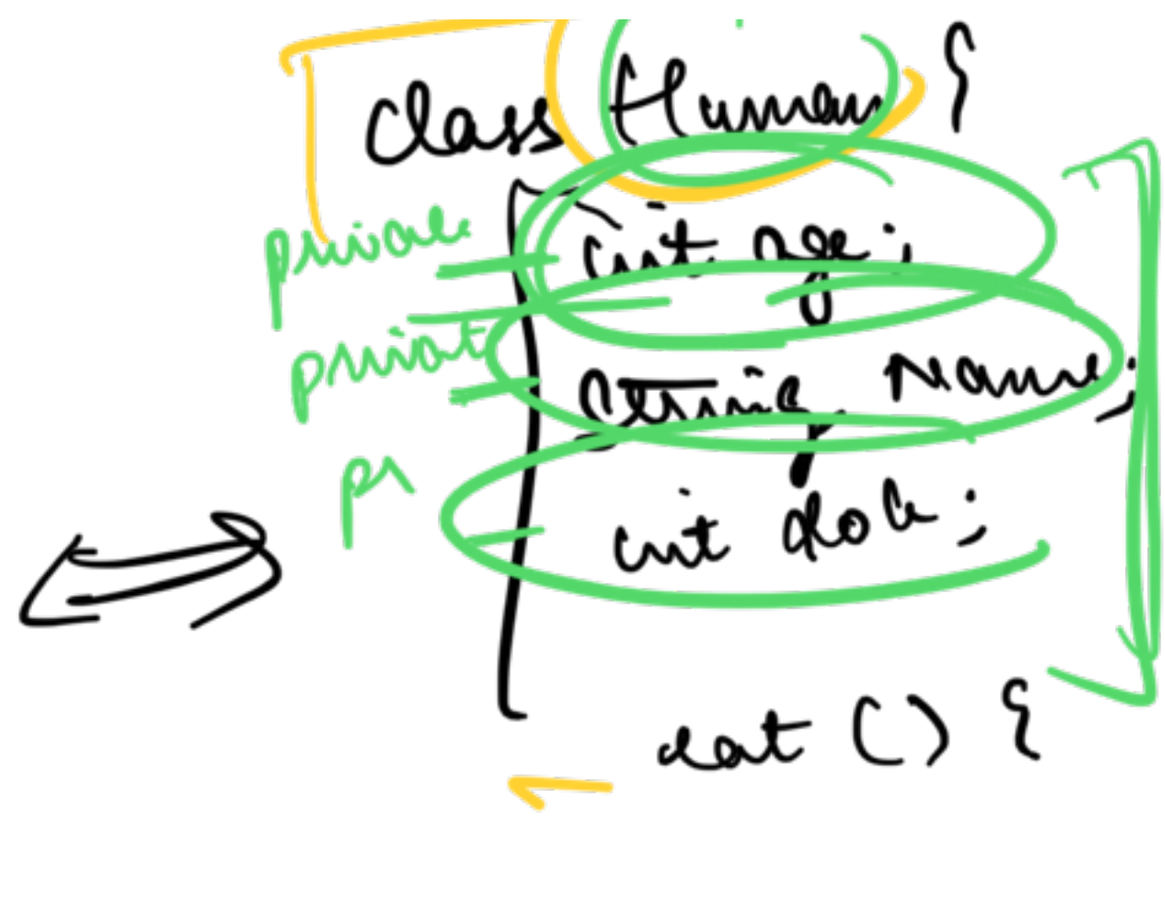
⇒ Ou struct Human {

int age

String Name

int doc

}



human Eat (Human h) {

}

human Dance (Human h) {

}

}

human walk (Human h) {

}

Human deepak = new Human()

walk Human (deepak)

}

Human dheeraj = new Human

dheeraj - walk(

walk Humans (List <Human> humans) {

}

Puppet Master {

List <Human> humans ;

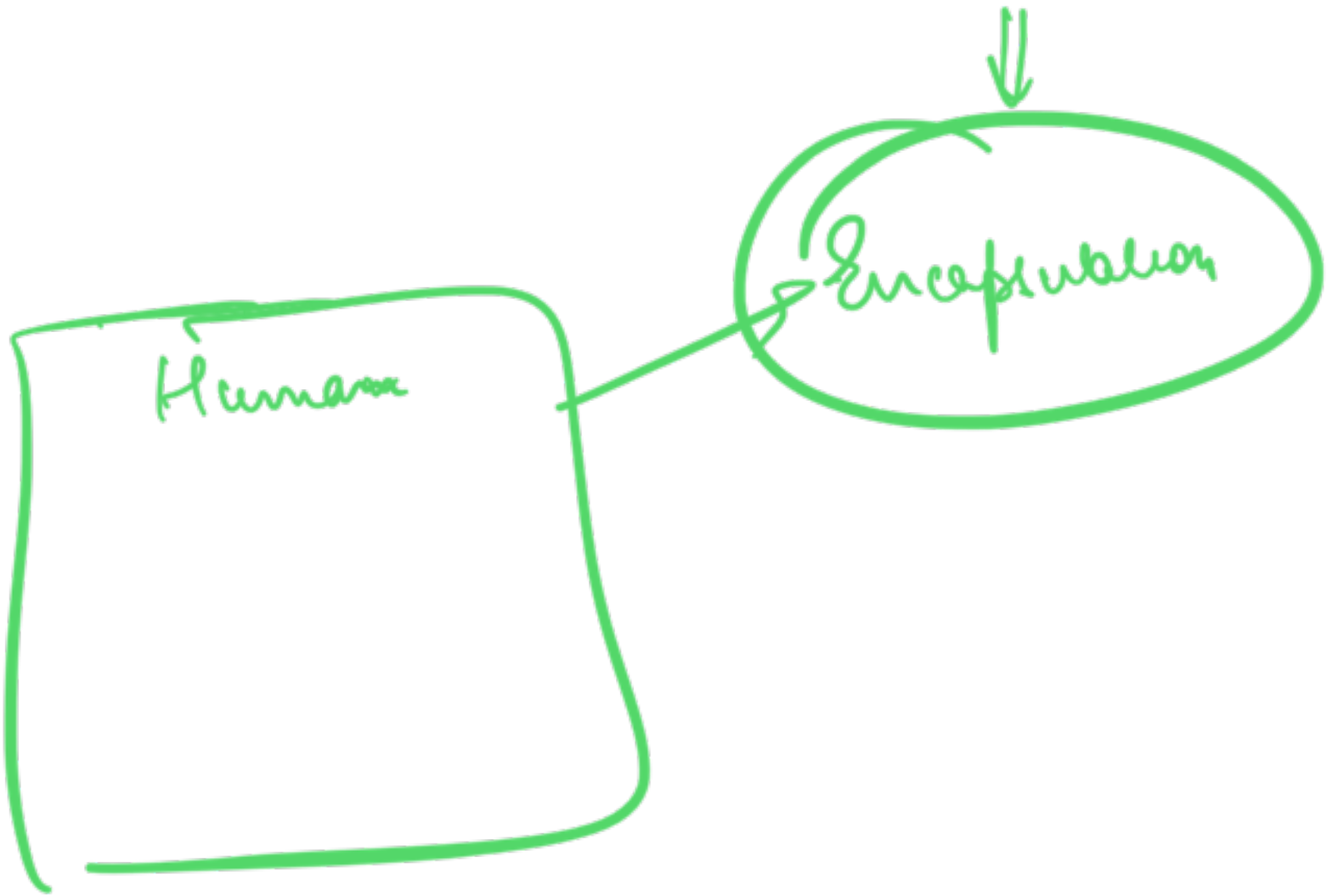
walkHumans ( ) {

for (

}

}

In OOP Languages: Data are full fleged entities



NON OOP

C → Struct

C++

Struct → Class

Util → SRP

SRP

Util

Single Gr

Procedure == Functional

Procedural P == Functional P

In F^{n} Prog language, F^{n} are also full fledged

Entities

1

We can create a variable of 1st type

We can pass f^{n} to f^{n}

Python

Java

JS

Java → Procedu

→ OOP

→ F^{nc}

Golang OOP

JS → OOP

JS

```
feed ( human, pizza, checkHungry ())

doSomething (void func() ) {

    func()
}
```

↳

Key terms related to OOP

Classes

Object

State

Members

Attributes

OOP every entity has data and behaviour

Type of entity → Class

ask constructor to start

building ⇒ grid design

Architect

Software Architect

class: Blueprint of an entity.

class Zoo {

class Animal {

age
Name
weight
height
walk()
sleep()
eat()

}

class Cage {

attributes
fully

Members

methods

}

Object → real instance of a class

Animal tiger = new Animal()

↳ age
↳ name
↳ weight
↳ height

→ taking space in mem

tiger.age = 5

Animal dog = new Animal();

dog.age = 2

class Pencil

)

class Stationery

Stationery pencil = new Stationery();

→ Attributes / Fields → data values of a class

→ Methods → behaviours of a class

→ State → value of all attributes of an object at a particular time

Animal tiger = new Animal

tiger.age = 12
tiger.name = AB
tiger.weight = 123

{ age = 12
name AB
weight 123 }

}

tiger. age = 23

{ age = 23

name = AB

)

Class, Object, fields, Attr, Methods, Members, etc

class Tiger {

}

Animal

Shera = new Tiger

Shera = new Animal

RBI banned 4B from ...

Adapter

PhonePe

PayTM

UPI

1 4000 USD

1. Loosely Coupled Systems //
2. No 2 concrete classes depend on each other
3. Use interfaces properly

Mailchimp

Code → Frontend → Adopte