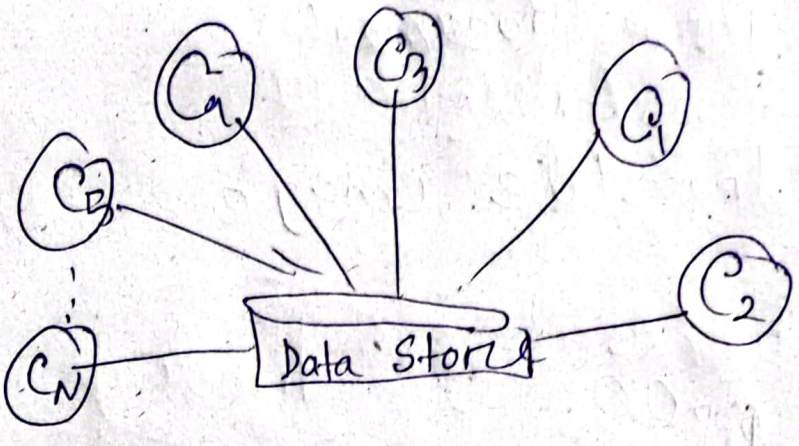
25/7/23

Data Centered Architecture

→ Central point of failure

Benefits

- Message passing
- Mediator
- less coupling



The above structure is similar to a "mediator" pattern so it also shows same deficiency : Central point of failure.

→ Data Flow Architecture

- Filter:
- Pipe:

- Chain of Responsibility
- Decorator

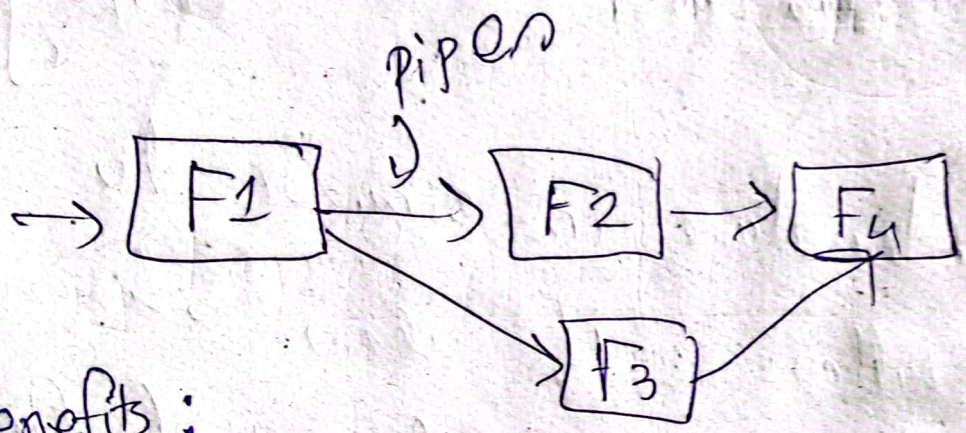
Similar to

Each filter performs diff task.

Translate Document :

F1 : Parse document

F2 : Translate image



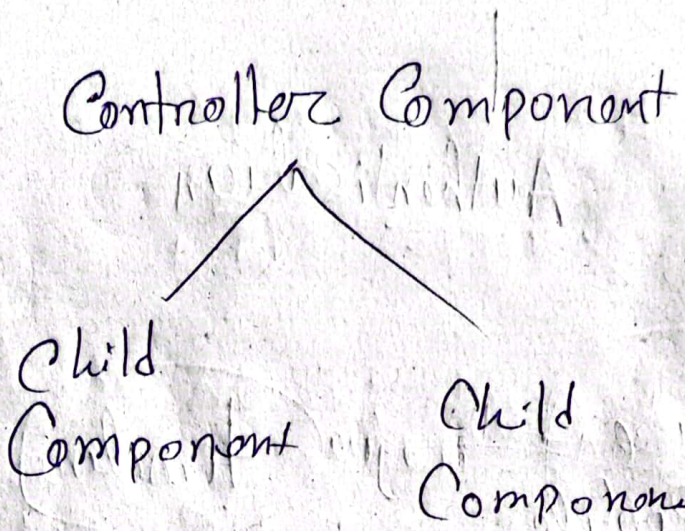
Benefits :

⇒ Filters are easily swappable to less coupling.

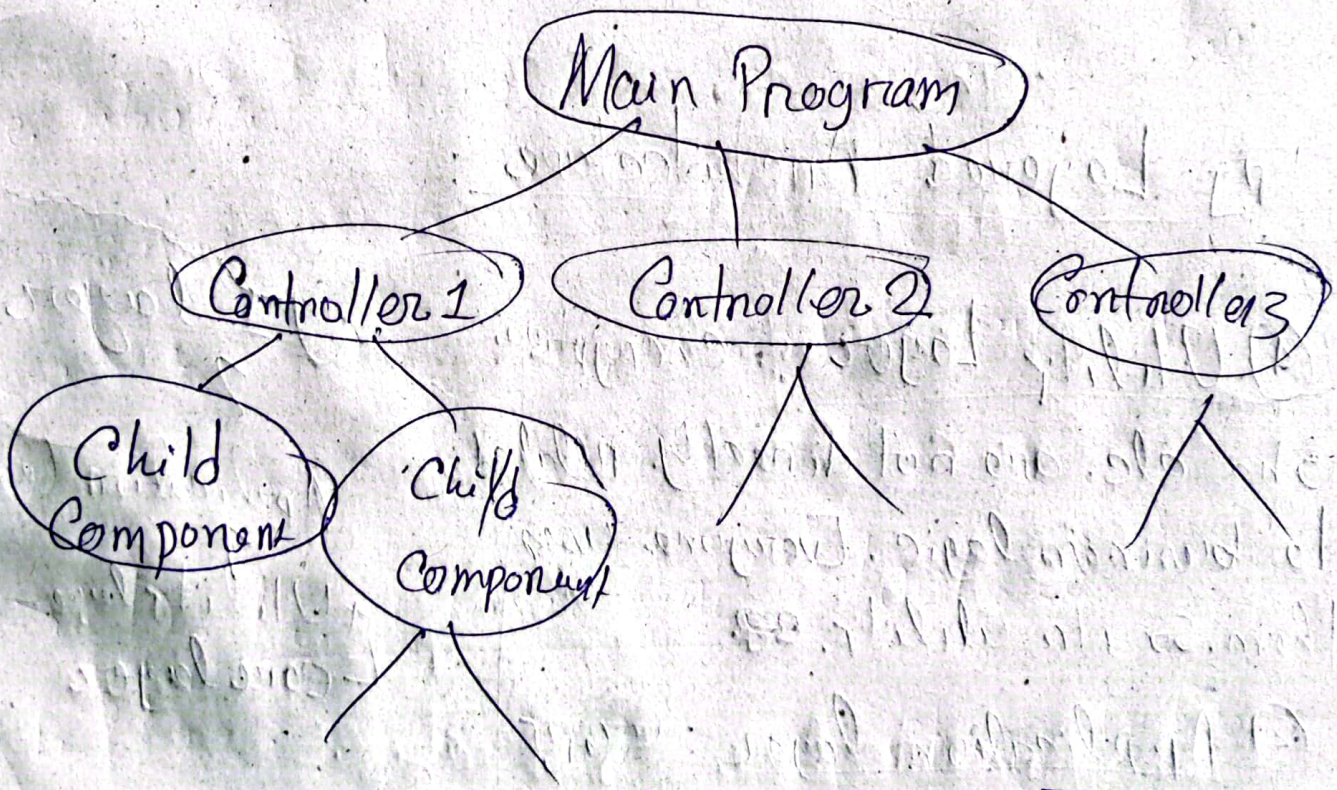
⇒ Much like decorator pattern, the filters can be arranged in many diff orders to make different systems.

Main Program/Subprogram Architecture

Subprograms are essentially components.



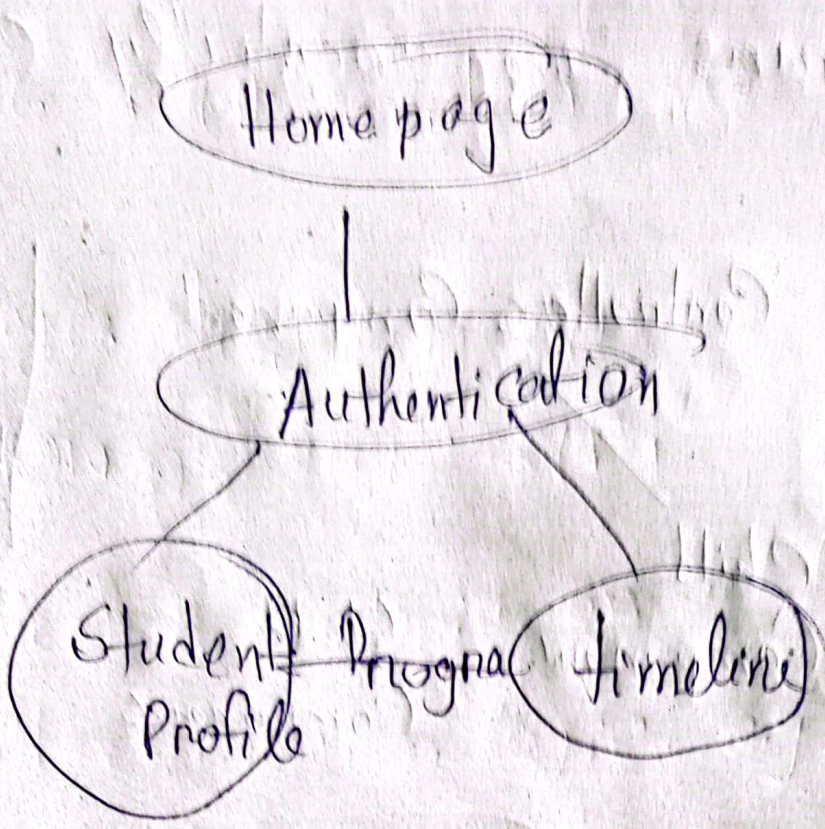
also known as call-and-return architecture



এরকম typical C/C++ code গুলোকে compile

২১।

e.g.



Layered Architecture :

① Utility Layer : encryption, Sha etc. are not directly related to business logic. Everyone uses them. So its utility. eg.

② Application layer : यहाँ business logic रहा,

UI layer
↑
Application layer
↑
Utility layer
core layer

→ Layer independence

→ Utility Layer has the most coupling.

→ Single way communication

Core layer : আমরা আমার core layer কে বারবার হয় না, এটা আরও Infrastructure.

Peer System / Superordinate-subordinate Relationship

Peers : They use each other. [My system use them. They use me.]
e.g. bkash

Superordinate System : এরা আমার system ব্যবহার করে, যেমন : আমি api open করে দিলে অন্যরা আমার system ব্যবহার করবে।

Subordinate System : Always EXTERNAL entity. আমরা বাইরের যেসব api, sensor ব্যবহার করি, তারা, ~~এরা~~ [DB/UI not external of my system]

ACD / Architectural Context Diagram:

- Actor কখনো system হতে পারে না।

→ Single diagram হবে

→ একটাই context হবে।

→ Mention criteria on what you consider business logic. e.g.

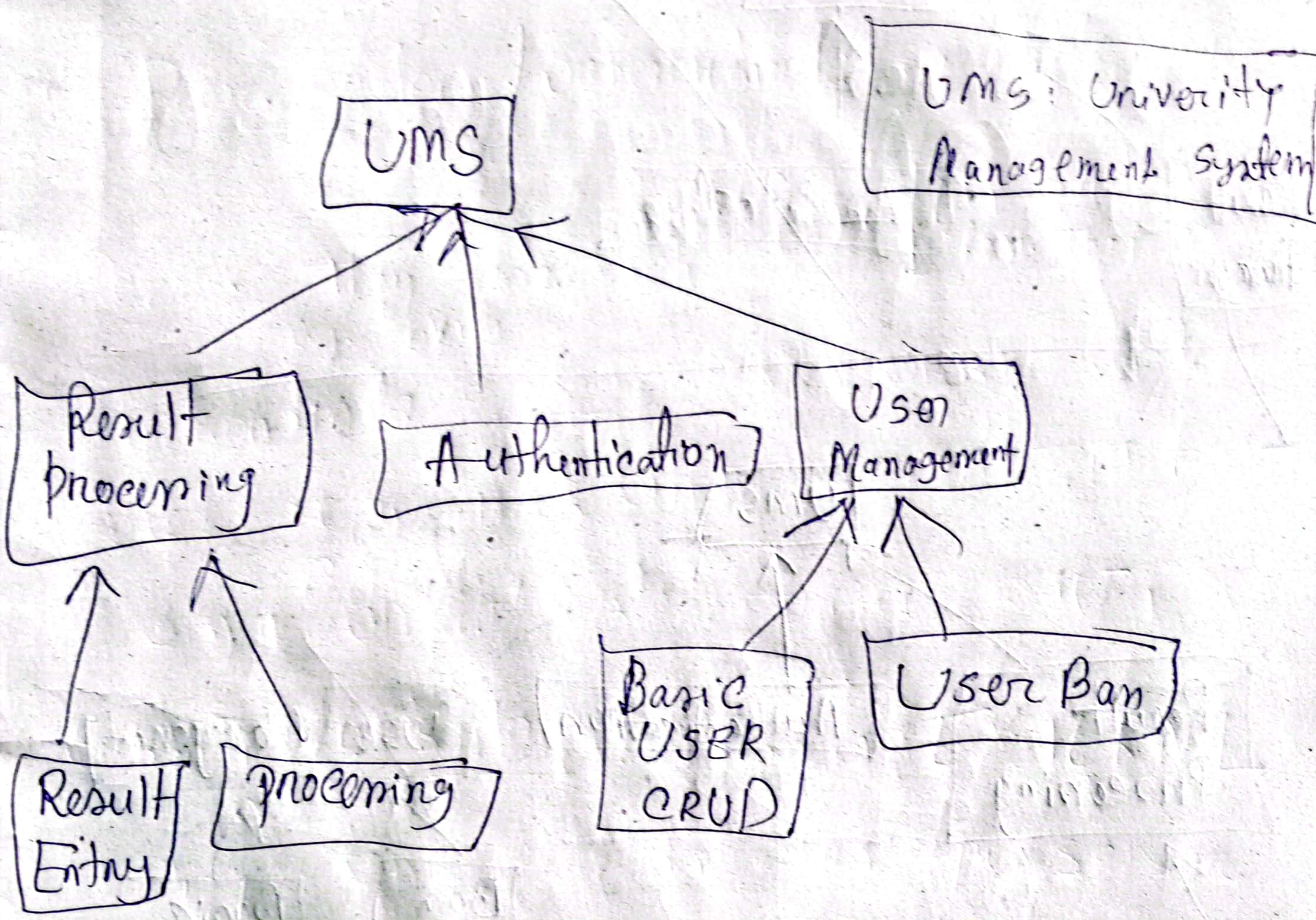
- Technical আর business logic.
- Application - Service layer হলো Application layer থাকবে business logic. Service layer হলো data layer.

Architecture → Architectural

Component → Initials Pattern

- System কে সরাসরি কিছু obvious component এ ভাগ করি, যেমন:

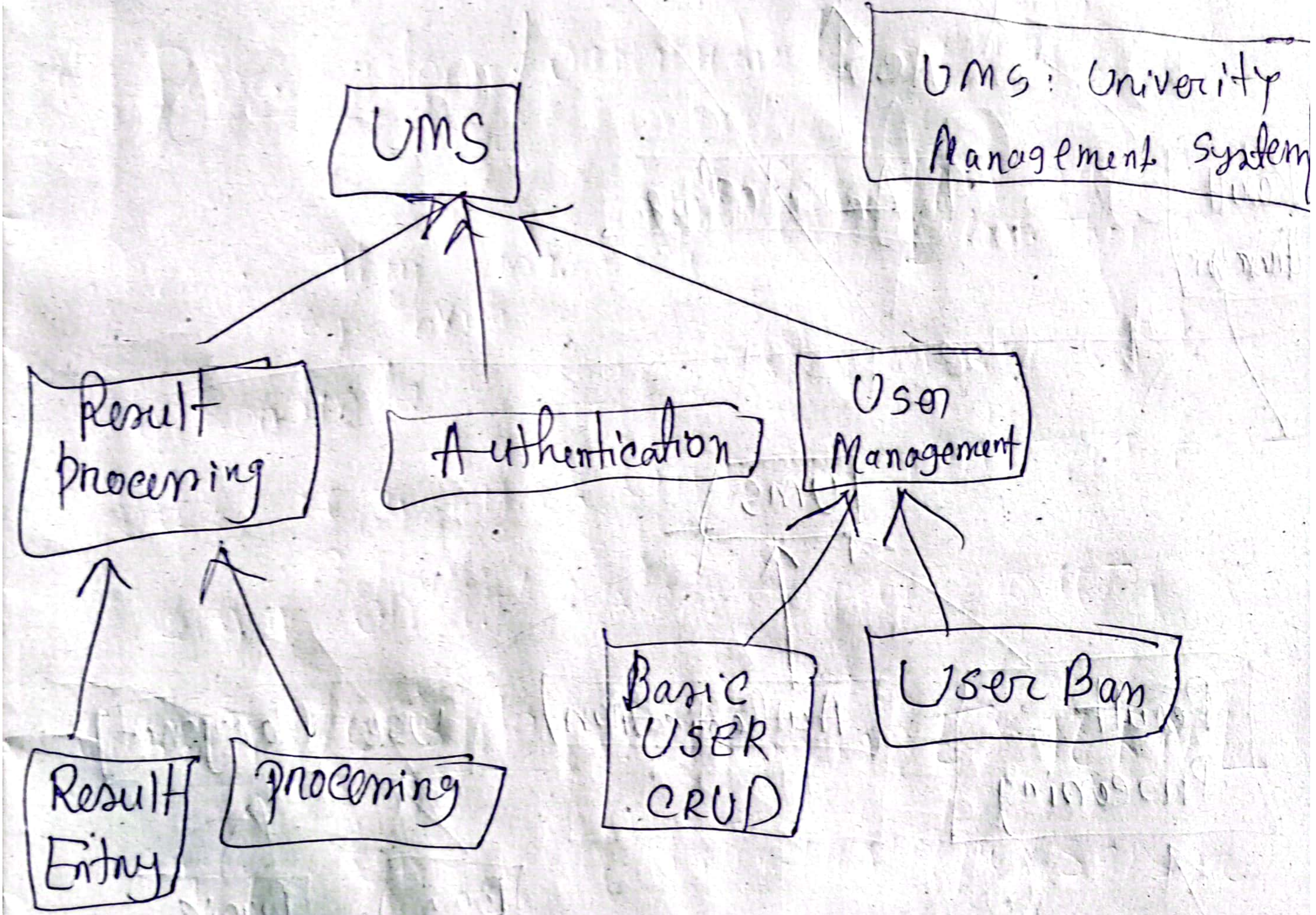
→ External Communication Management comes from ACD.



- The goal of composition decomposition should be generic, not specific. So almost all such design is reusable to other system.

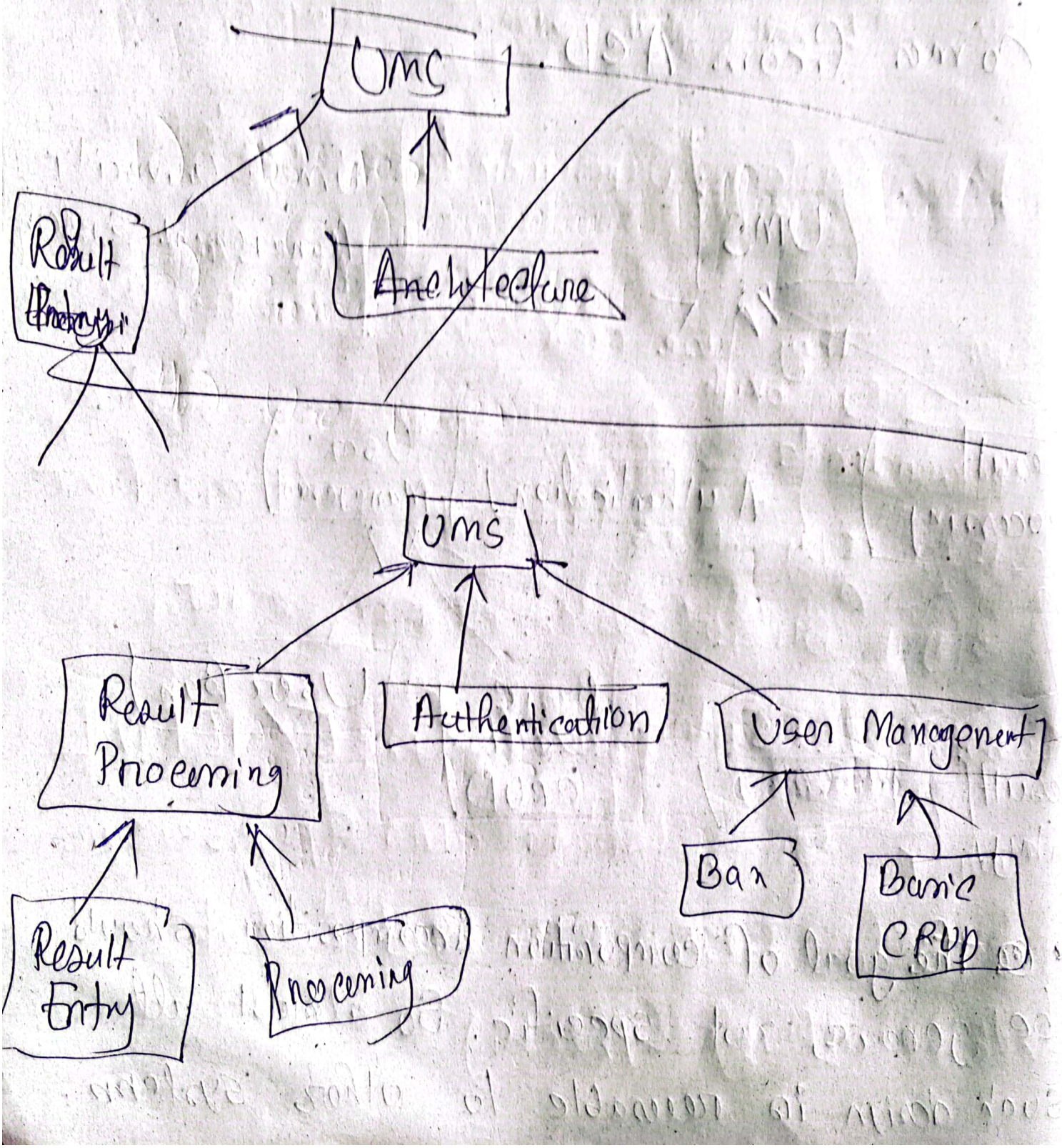
- System কে আপাতত কিছু obvious component এ ভাগ করি, যেমন:

→ External Communication Management comes from ACD.



- The goal of composition decomposition should be generic, not specific. So almost all such design is reusable to other systems.

- We need use pattern on general structure to make it specific.

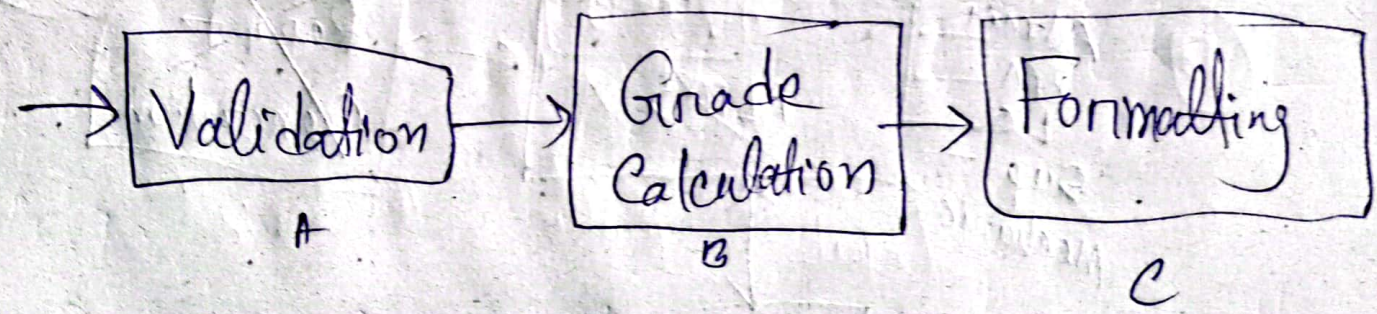


3-layered leaf এ usually pattern প্রয়োগ করা হয়,

Result Entry: We can use layered architecture for this component

UI
Utility
Application

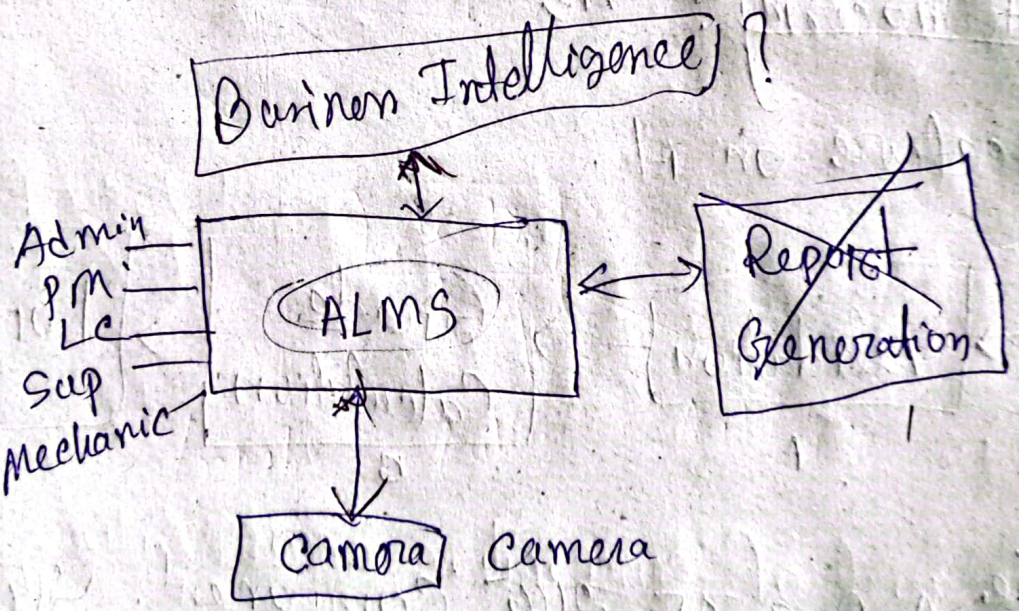
Result Processing: We can use pipe-and-filter architecture on it.



এখানে, A, B, C হল তিনটি Component. এগুলো আবার class/algo design করতে, Result processing is much like controller.

Task: Complete these 4 steps for your project.

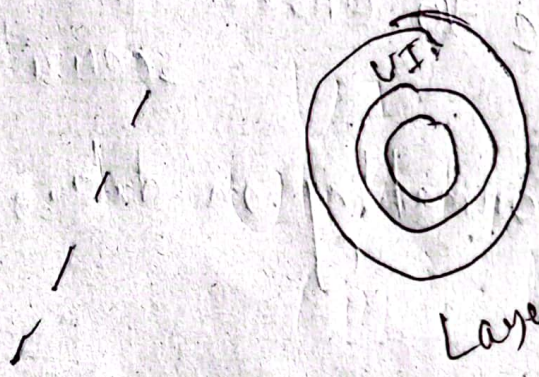
1) Do ACD
2) Architectural Pattern for (general
3) General Component
4) Use pattern



→ Camera once

→ Define pipe/filter.

→ Layout : Layered.



Layered

• Define in ②

• UI empty.

• MVC

• Data Centered.

UI : User interface

~~App~~ DAO : DB

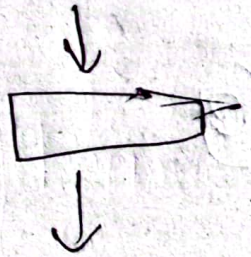
Utility : Utility

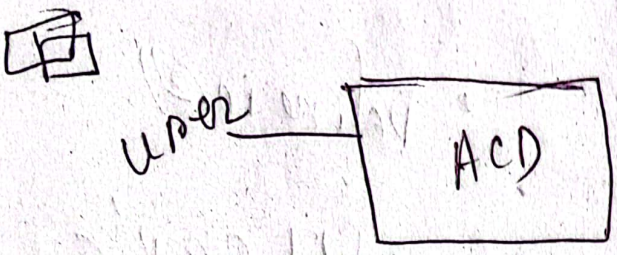
Service : Business Logic

Pipe :

Filter :

ACD



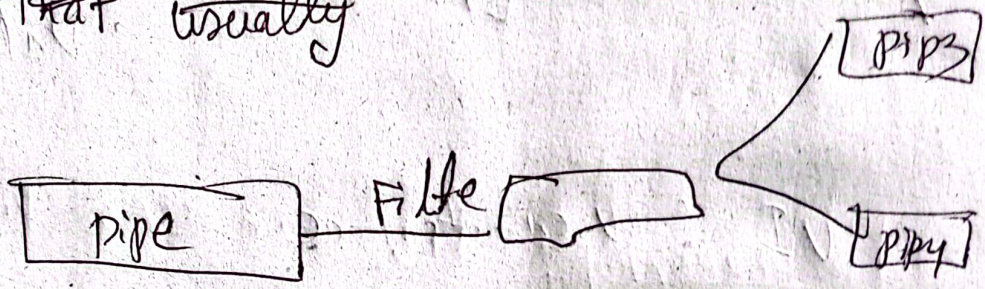


My will use pipe-and-filter architecture.

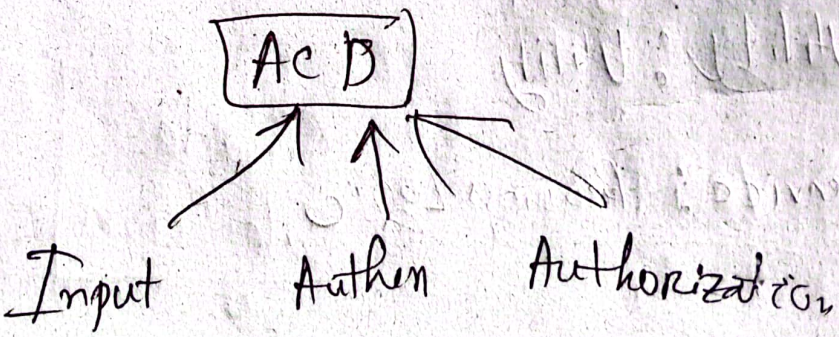
Pipe: All processes done to ensure privacy & security

Filter: Intermediary formatting on data.

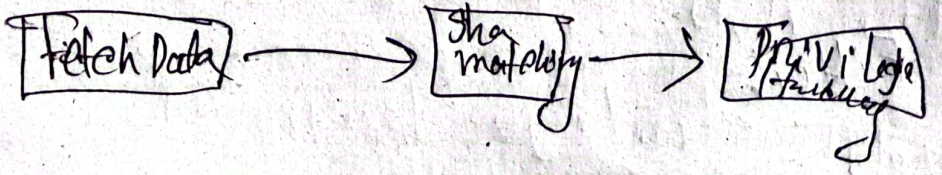
~~that~~ usually



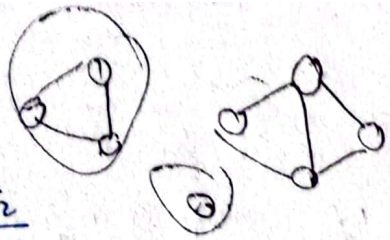
③

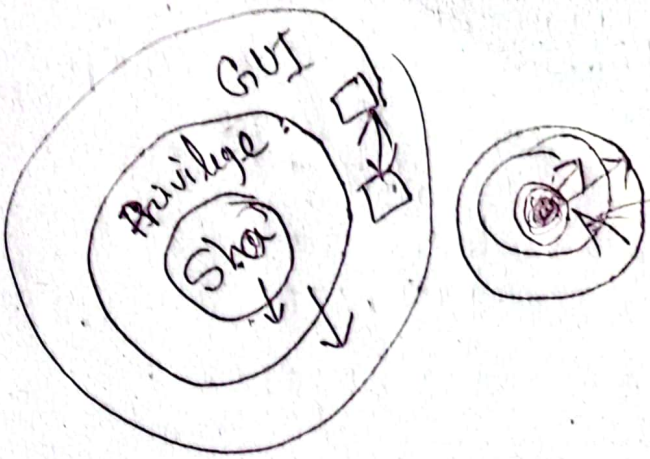


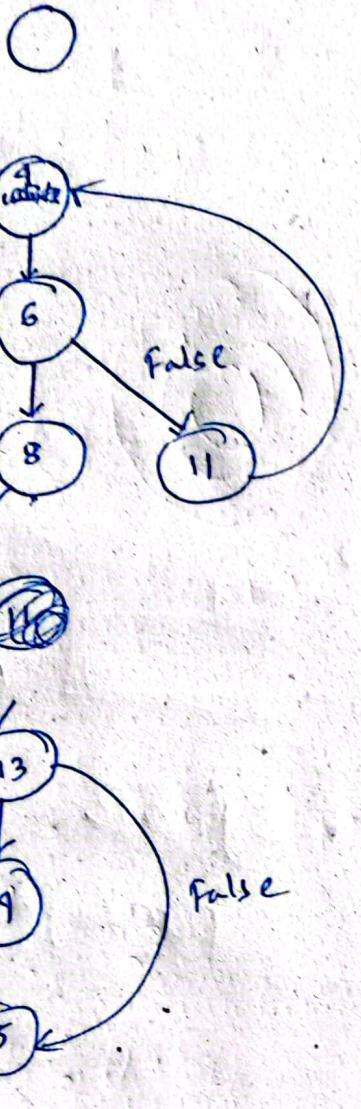
④ Modern:



CFG







4 - while

6 - if

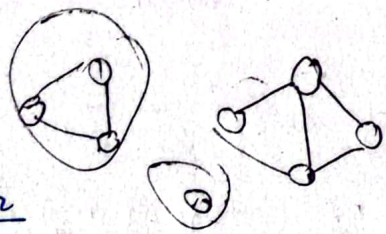
8-9 → if Condition body

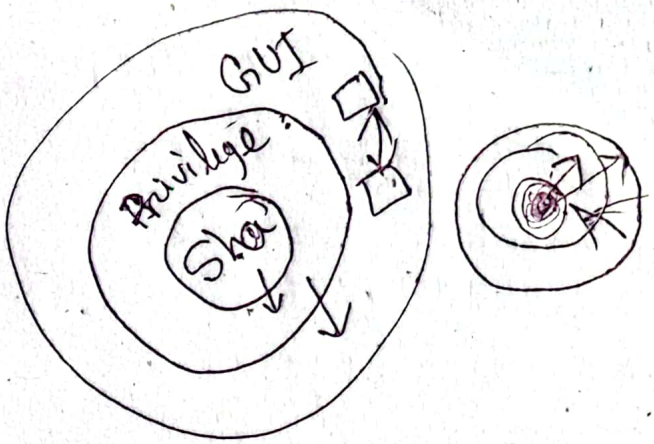
11 → index++

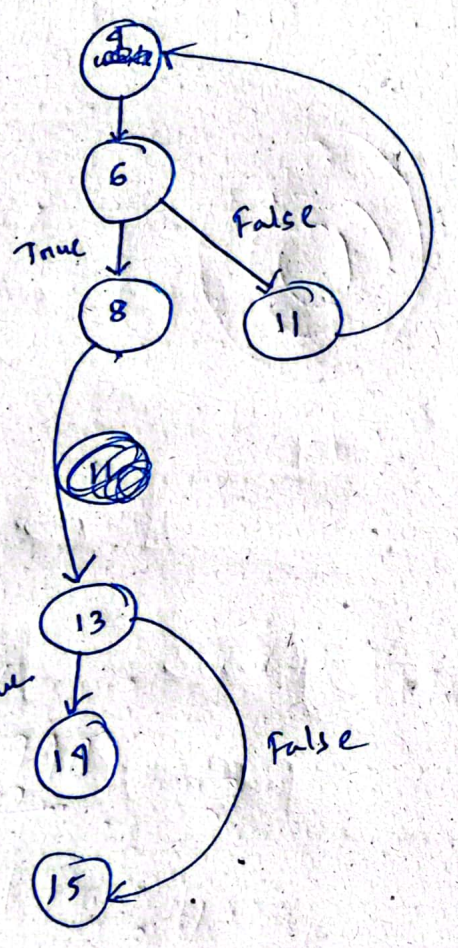
13 → if ~~body~~

14 → if body

CFG







4 - while

6 - if

8-9 → if Condition body

11 → index++

13 → if ~~body~~

14 → if body