Monolith

→ single massive system with single codebase

Adv → Easy development (1 codebase)
→ Easy testing
→ Easy Deployment (1 executable file)
→ Easy Debugging
→ Performance (can perform same funct^n that many APIs with microser)

Disadv → Slower Development Speed
→ Scalability
→ Reliability (error in 1 module affects entire app)

Microservices

→ collect^n of small, self-contained modules based on business functionalities

Adv → Agility → small teams that deploy frequently
→ Flexible Scaling
→ Cont. Deployment
→ Highly Maintainable & testable
→ Independently Deployable
→ Tech flexibility
→ High Reliability

Disadv → Management - multiple servers, multiple teams. If not managed properly →

→ Barrier to Tech Adoption
- any changes in framework/language affects entire app.

→ Lack of flexibility - constrained by tech already used

→ Deployment - small change requires redeployment

slower dev speed & poor op. perform.

→ Infra Costs

→ Add organizational head
→ communicat & collab to coordinate & update

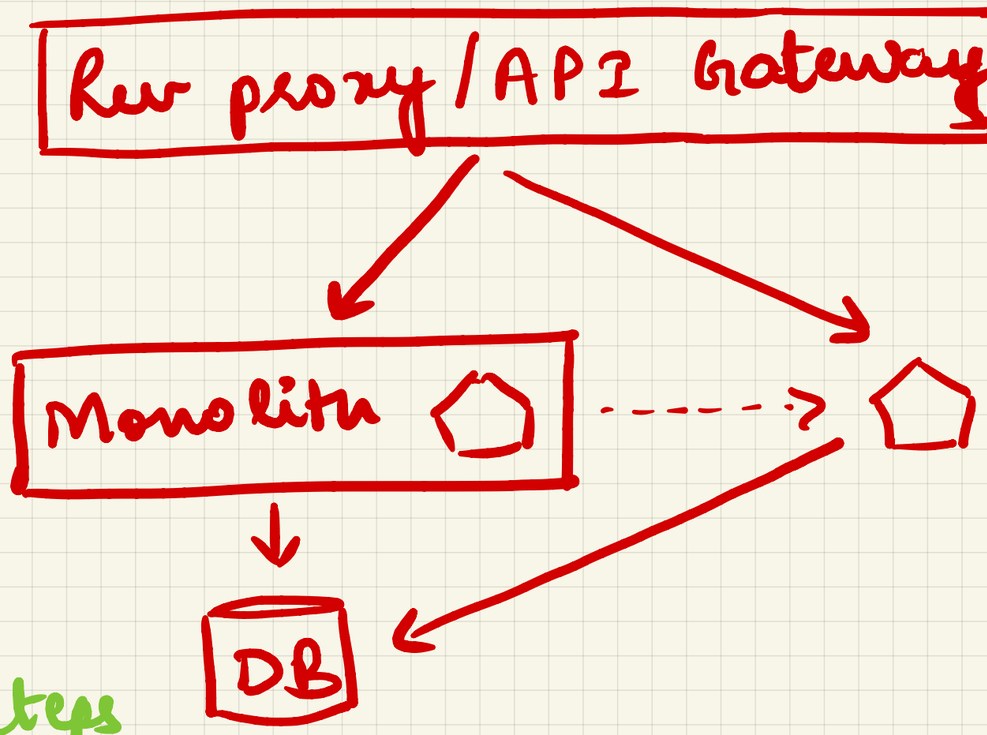
→ Debugging challenges

→ Lack of standardizat

→ Lack of closer ownership

Strangler-Fig Pattern

→ incrementally migrate new functionality to new sys. & rollback easily if required



⇒ handle req. navigation

→ most common
→ minimal impact on existing system

3 Steps

① Identify parts of existing system that we wish to migrate
② Implement this functionality into new microservice
③ Consider "parallel run" to give more functionality to system

Disadv → Need to intercept / divert calls

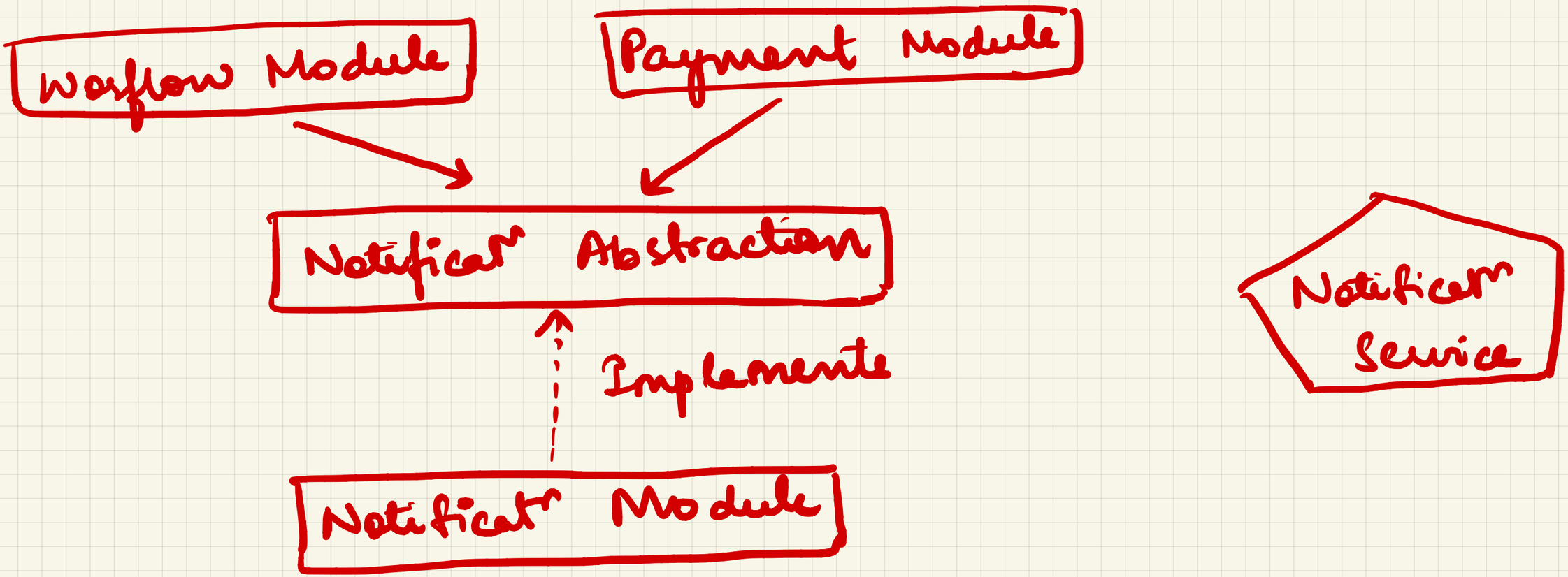
Branch by Abstraction Pattern

→ focuses on improving current codebase, allowing apps to coexist alongside each other without creating too much disruption in same code version.

6 Phases

① Build Abstraction for functionality to be replaced
② Implement newly built abstract in existing module
③ Implement new functionality in a new service

For some time, this implementation would be idle & would not connect to any functional flow.



④ Implement service provider or some wrapper/face to communicate with freshly built service by implementing the abstraction

Workflow Module

Payment Module

Notification Abstraction

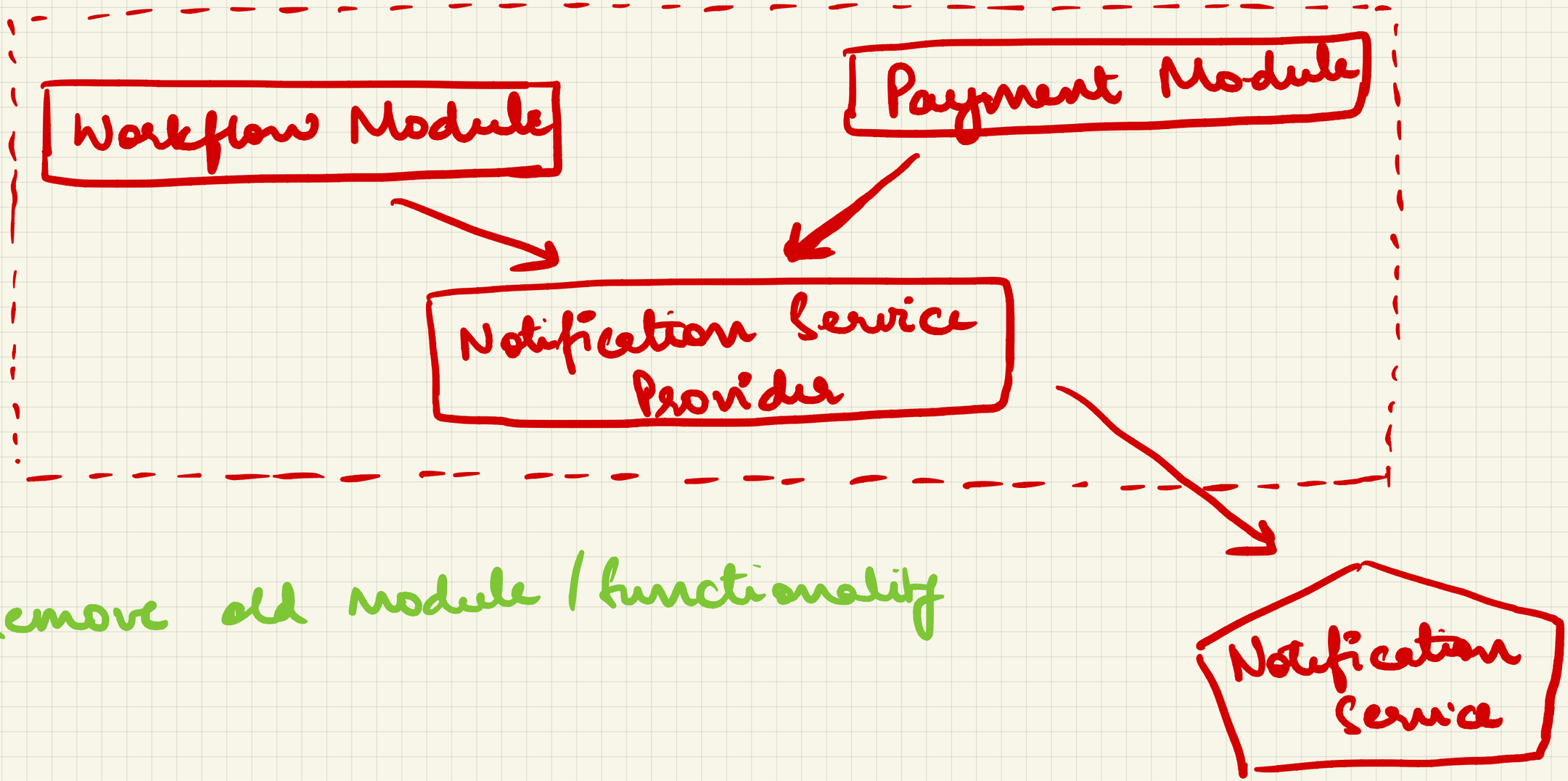
Implements

Notification Module

Notification Service Provider

Notification Service

⑤ Now Data flow is using new service & you can get rid of abstraction



⑥ Remove old module / functionality

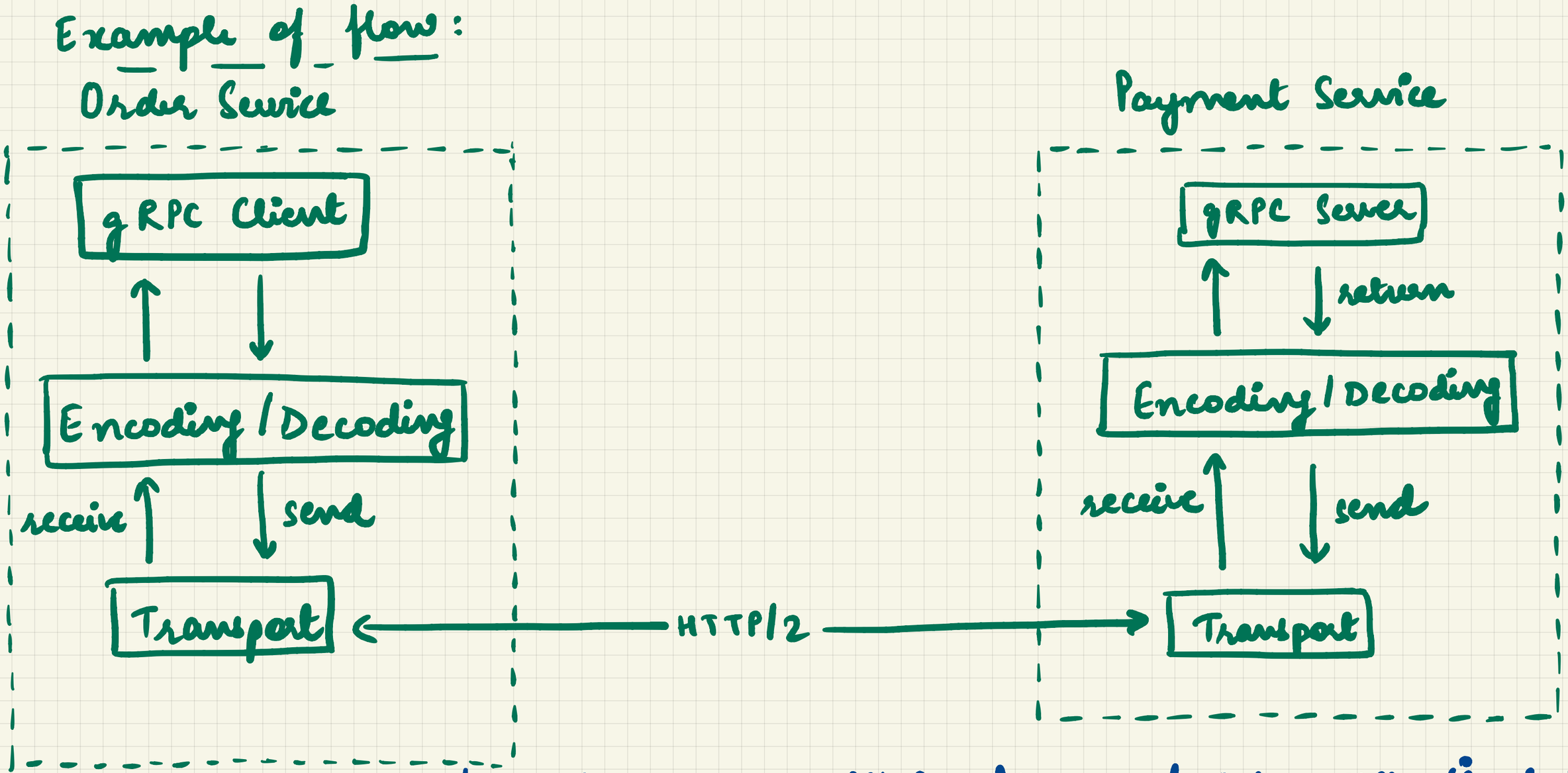
Remote Procedure Call (RPC)

→ allows programs on different machines/systems to call functions / procedures in a remote process as if they were local.
IPC - Inter-process communication

gRPC → open-source
→ built on top of HTTP/2 & uses protocol buffers (protobuf) as IDL - interface definition language

→ **Protocol Buffers** → platform-neutral, extensive mechanism for serializing structured data
→ structure msgs & services in a '.proto' file
→ very efficient, much faster than JSON

Example of flow:



→ gRPC uses HTTP/2 streams → multiple streams of msgs over single TCP connection. Thus, many concurrent RPC calls can be handled over a single TCP connection.

→ gRPC is for communication between microservices, not for frontend to backend because gRPC relies on lower level access

Webhooks

3 step process -

① Webhook Setup → Receiver of webhook provides unique URL where it wants to receive data

② Event Occurrence

③ Webhook Execution → HTTP Post request to receiver to the webhook URL.

eg → CI/CD - automatically trigger build process / deployment actions when changes are pushed to version control repos.

→ payment processing - update order status & trigger actions

→ sending automatic reminders