Design Payment gateway

(*) Contents :- → (TERMINOLOGY & PAYMENT DOMAIN TERMS)

→ How Card Payment works?

→ How 3D secure card payment works?

→ Requirements

→ Design considerations

→ HLA

→ Relevant APIs

→ Payment Processor.

(*) Gather Requirements :- (Payment System of E-Commerce)

(*) Estimations

(*) High Overview of design :-

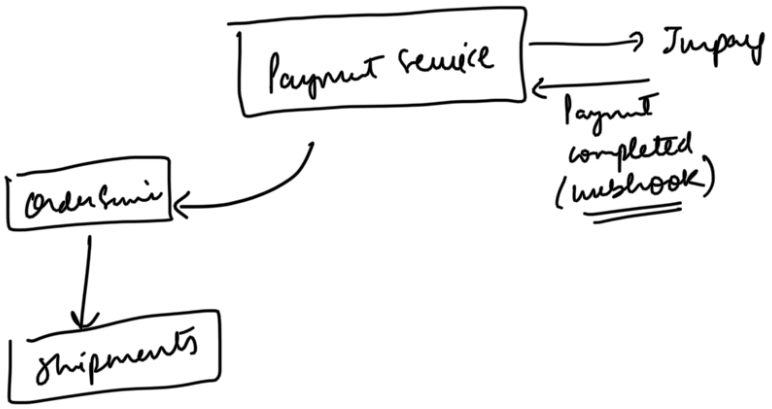
(*) Deep Dive :-

Place Order —click→ Jumpay
(Payment Gateway Provider)

productid/sku
customerid (check which warehouse in pune)

delivery addrs (nearest warehouse mapping)

order —1:n— shipments
order —1:n— items
items —1:n— shipments



(*) Requirements :- (Interviewer)

e.g. E-commerce System (Amazon)

Payment Options (Credit Card, UPI)

Who's handling Payment Process? → PSP (Jupay) (Stripe)

∴ Payment System → PSP → UPI → Bank Settlement

We'll discuss this

Will we store Credit Card of user? X → No
→ Compliance issues.
X → Payment gateway (currency do we)

How many txns per day? → 1 Million txn per day
= 10 lac txn per day

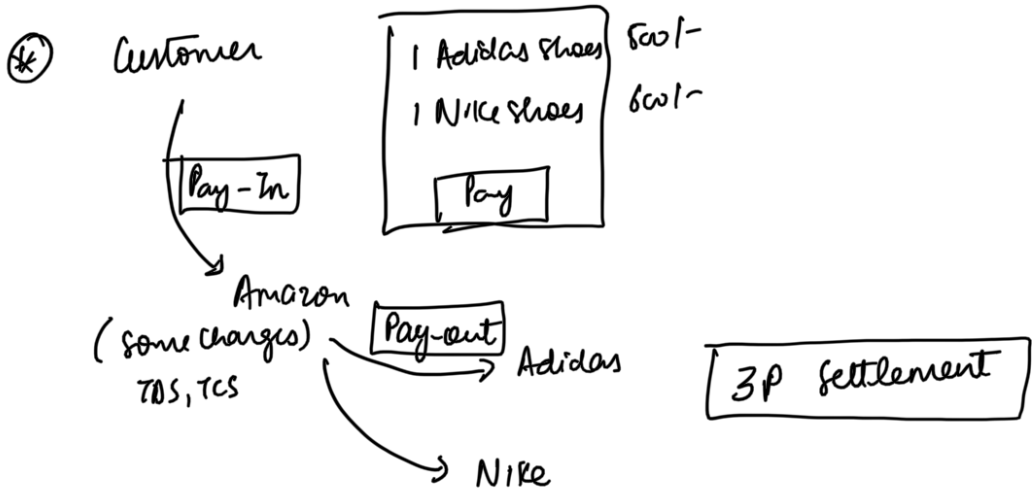
Which Part should I focus more?

↳ More reliability, handling failure

reconciliation

IDIOt → Blue shaming

Functional Requirements :-



- Pay-In
- Pay-Out

Non-Functional Requirement :-

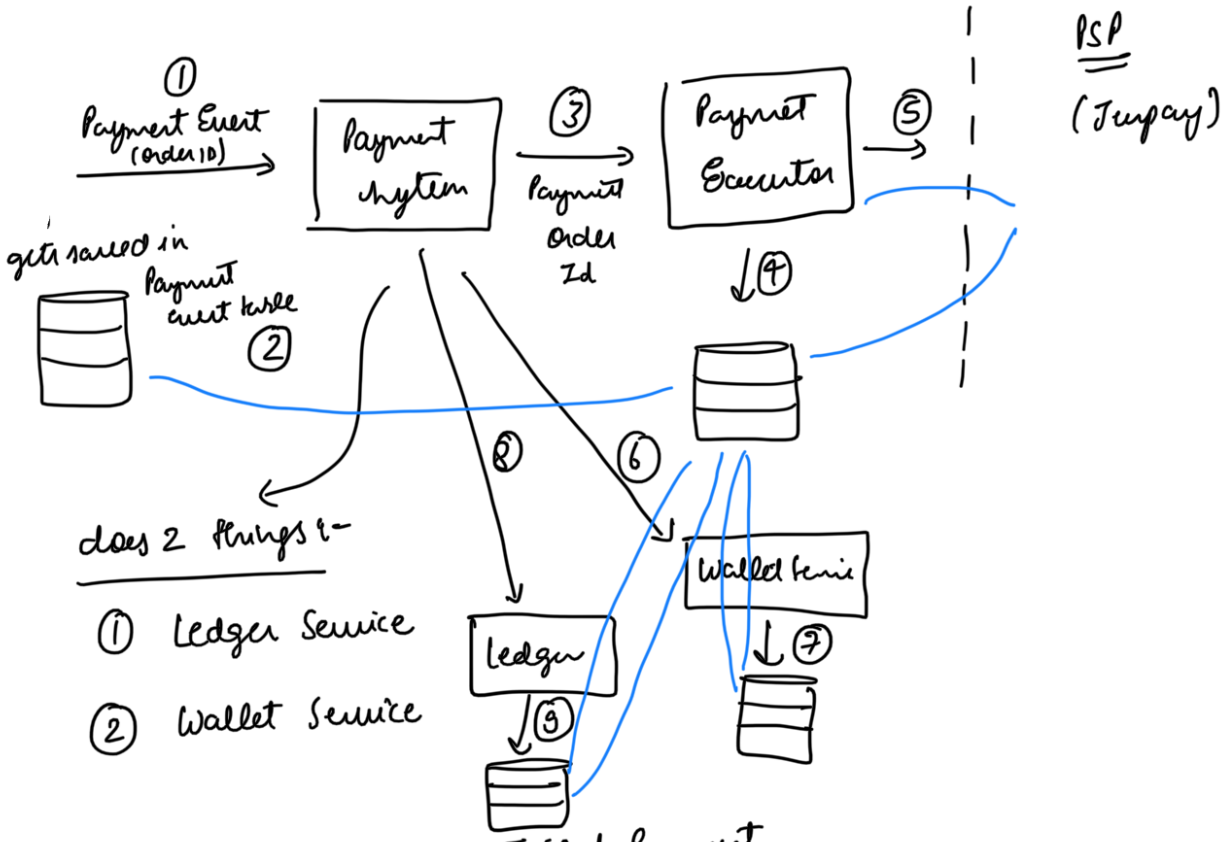
- Reconciliation → handling all failures
  - → under-processing (delayed payments)
    - ↓
    - can be settled once a day at EOD.
- Payment should be done Once.

Estimations :-

- 1 million req / day → 10,00,000 / 84600 → 10 TPS (Low Throughput system)
- Handling Failures | Reliability :-

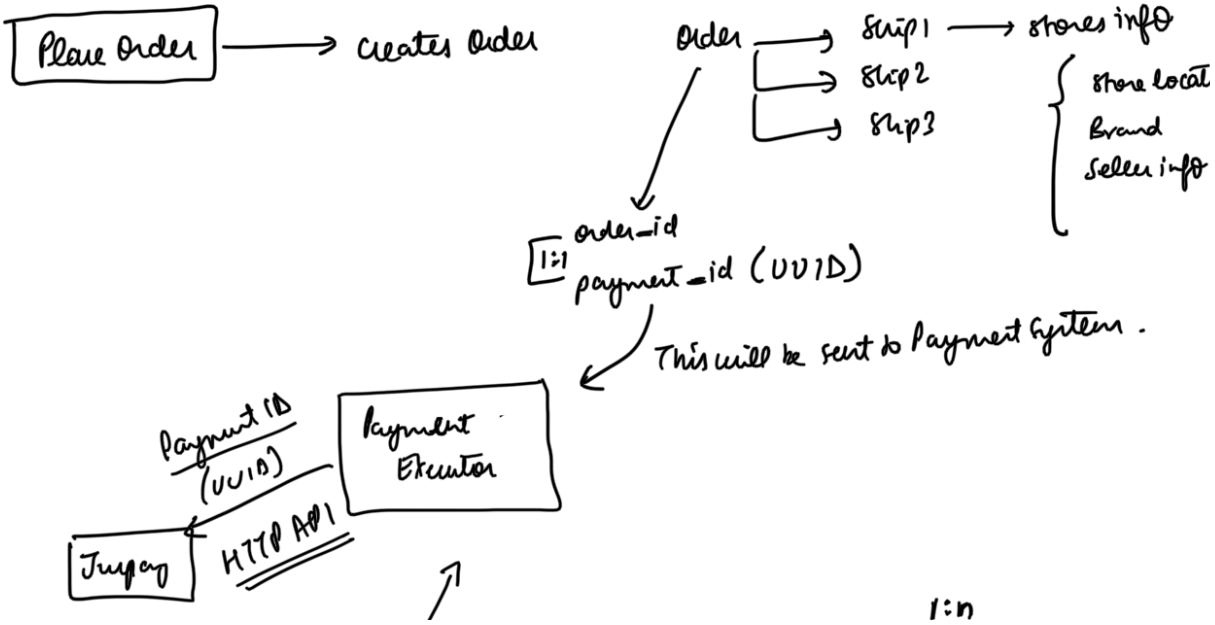
Pay-In Flow :-

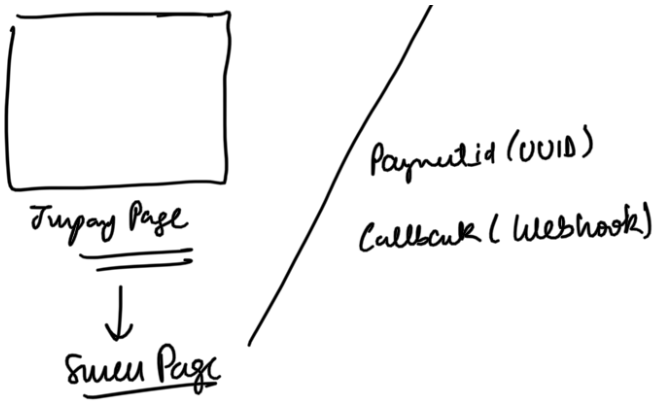
↳ Only talk about Payment System
(not about OMS / Inventory Mgmt systems)



Most Imp. → Handled Failed Payment
(यहाँ store करना पड़ेगा, at reconciliation time)

How Flow Works?



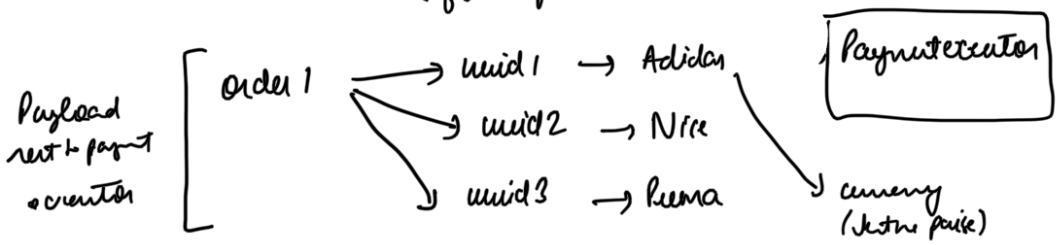


if; Payment success comes in 30 secs (send a redirect URL at time of request)

Payment Service → Jupay
UUID; redirect_url;

else; you open a new page; (order is pending

* We create multiple payment events for a order_id,
(for loop)



OR

- AWS Step Function (Managed) via resiliency.
- * (Chain of responsibility Pattern)
  ① → ② → ③ → ④

OR

- * FSM / State-Machine
  pending → ready → ready to ship

Happy Flow Done !!

Databases :-

→ system is not read heavy OR not write heavy

→ data will be a lot (stability)

→ Any database
- MongoDB (ACID + NoSql)

PaymentEvent
1 ↳ order_id
: n ↳ payment order id
- uuid
- amount
- currency
- (bool) is payment done
- (bool) is ledger_done
- payload
- (bool) is_wallet_upda
- store_loration
- state

Pay-Out Mechanism :-

→ Read to 3P settlement

→ Go to wallet Table

→ Query → - TDS
- Charges

Amount → PayOut to Adidas

PayOut link of RajaPay
- OTP comes to your phone
- Fill out
- PSP does payOut to you.

(After 30 days of order)

(order may be cancelled)

(Remove after order is delivered +7 days)

E-commerce (30 days)
Swiggy/Zomato (1 day)

Deep Dive :-

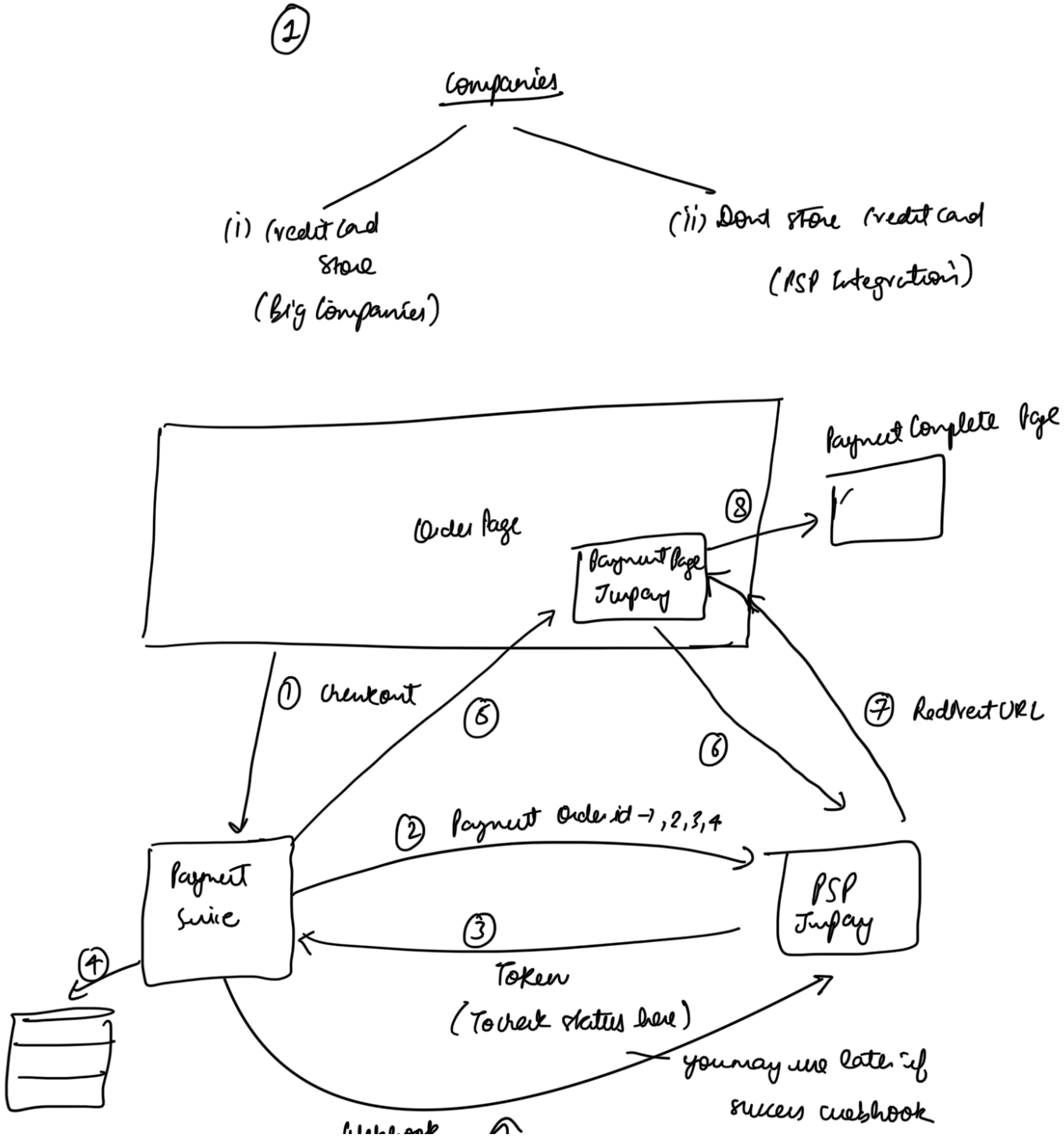
1. PSP Integration
2. Reconciliation

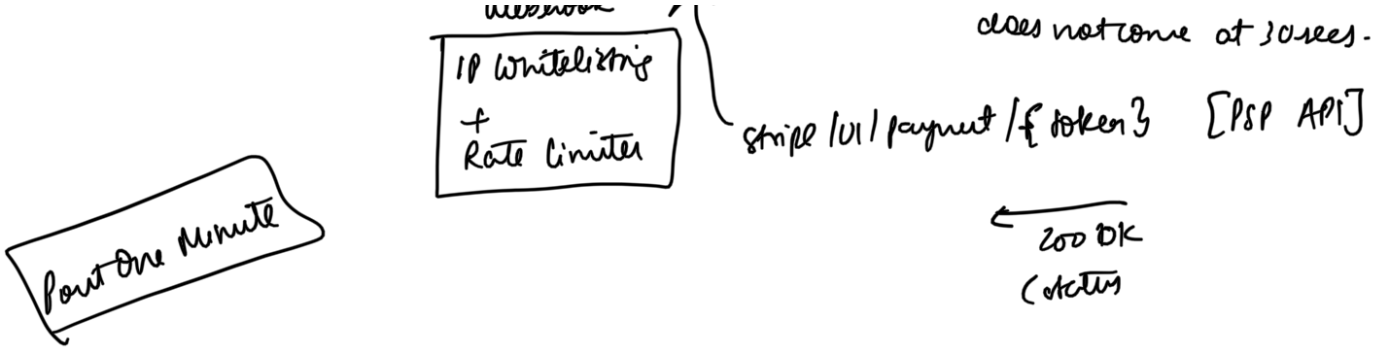
③ Handling Payment Delay

④ Communication b/w ~~Backend~~ External.

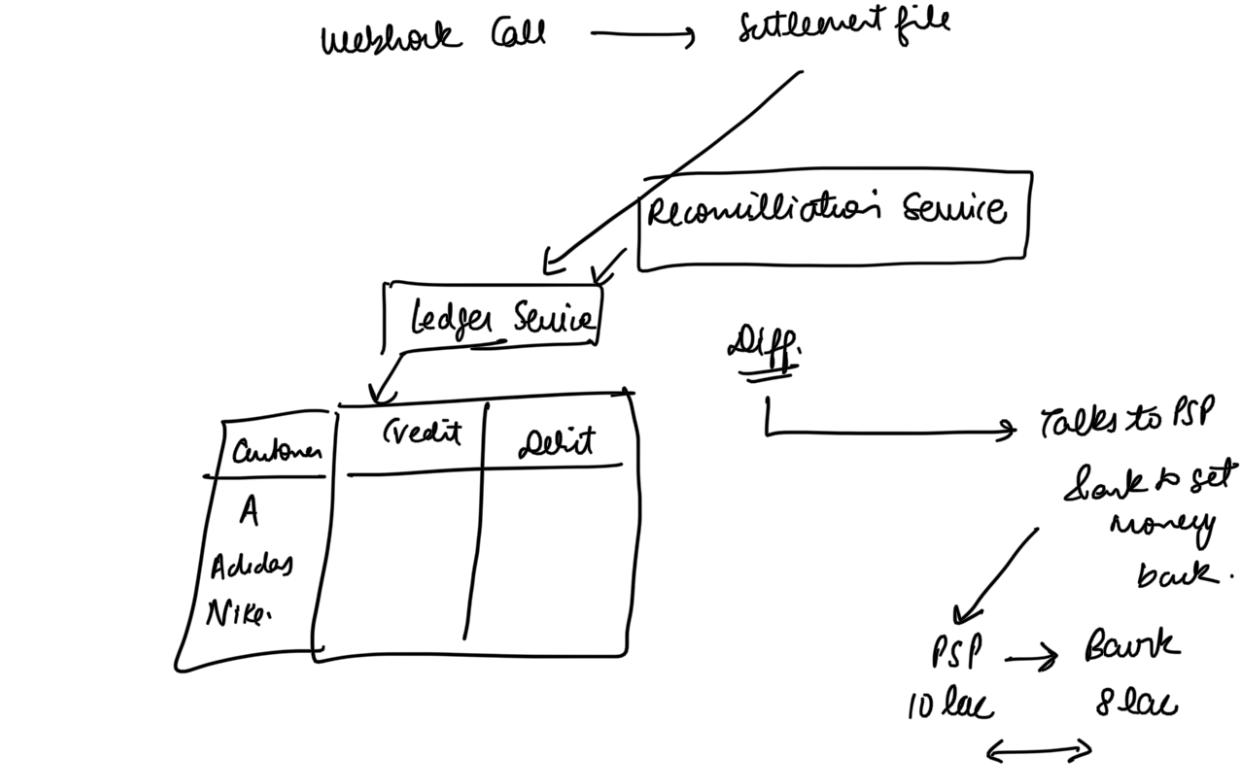
⑤ Idempotency → payment at once (no payment on second click).

⑥ Consistency





② RECONCILIATION :-



(*) if Payment Failure is happening a lot,

set alert system (Alert Charles)

(Grafana, Prometheus, Kora logics)

as payment system size Does

Logs store

Based on logs, we can create relative alerts / absolute
10% more alerts

③ Handling Payment Delay :-

ⓐ Webhook call from PSP & state change.
→ service

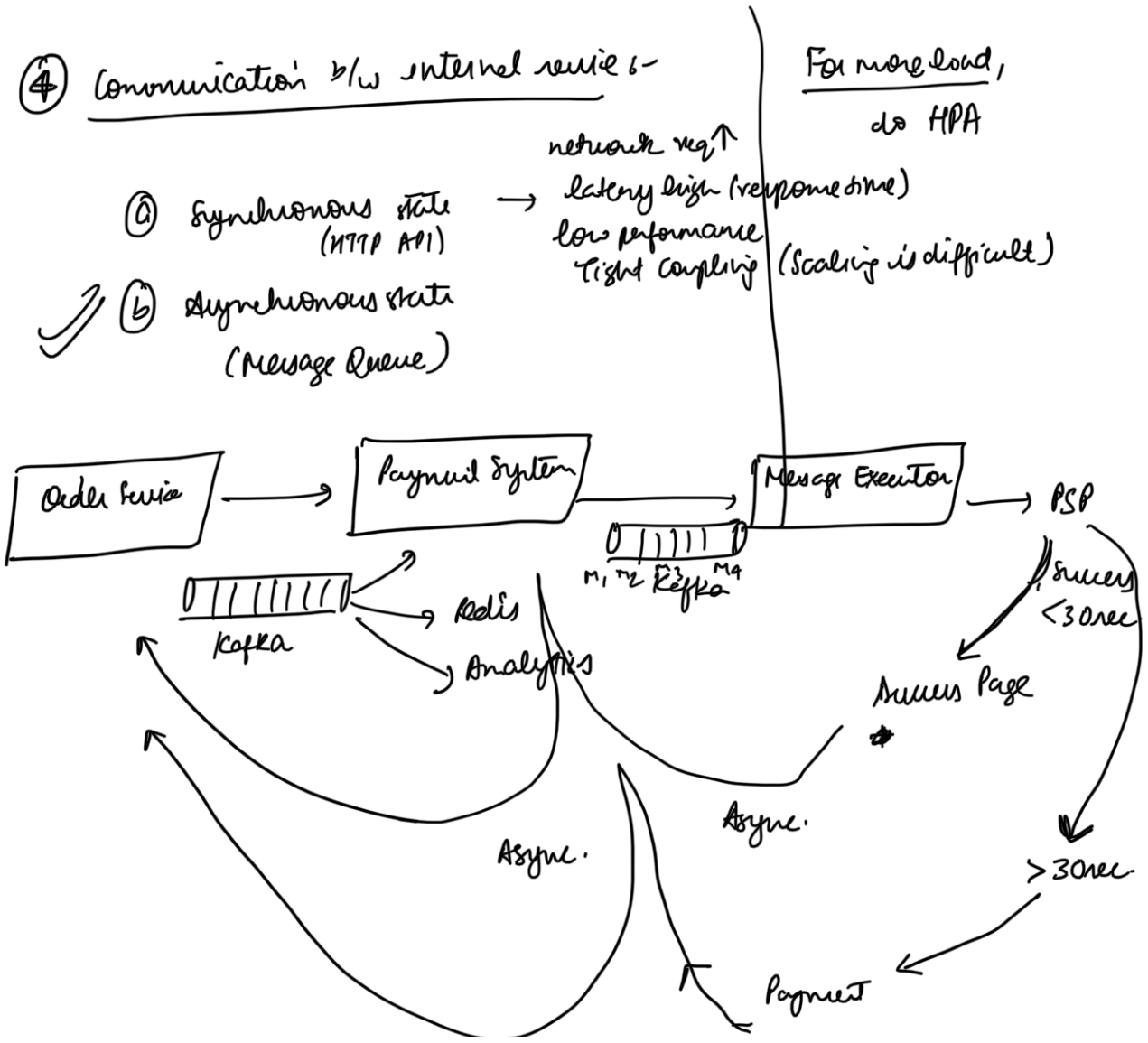
ⓑ Polling to PSP to get states.

④ Communication b/w internal service :-

For more load, do HPA

ⓐ Synchronous state (HTTP API) → network req ↑, latency high (response time), low performance, tight coupling (Scaling is difficult)

ⓑ Asynchronous state (Message Queue)



Pending

+

Order confirmed

later settled via async call to order/payment svc

High Risk Cases

↓

High Risk User ← Intelligence Service / Third Party User

↓

Check if order intent is correct

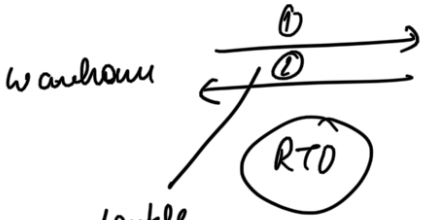
↓

Process Payment / Reject.

COD Blocking — Gokwick (Third Party) → Risk if we can block COD.

By sending event data.

- IP address
- mobile number.
- user id



→ COD Block.

(5) Idempotency :-

↳ maximum one time payment

Payment May Fail Also :-

Scenario1 → Order service → Async → Payment service

Payment
service is down.

Order s/v to Payment s/v — Retry Count with Circuit Breaker
↓
2/3
(exponential backoff implementation)

- First check if retry is possible.
  - if not possible, — store in database
    — alert the system

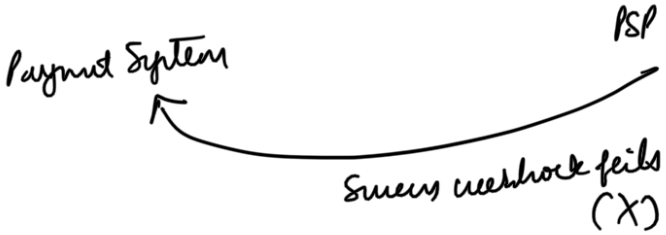
every retry → idempotency
(ಯಾವಾಗಲೂ)

- Retry Yes → Message Queue → Payment System.
  Update Retry Count in DB.
  OR
  Redis Queue

- if then also fails → Dead Letter Queue ಗೆ send
  ↓
  from there you can retry later.

Tool :- Sidekiq library from Ruby

Scenario 2



>30 sec
check status API
& update payment status.

CONSISTENCY

① load Tres
- replicas create (horizontal scaling)
- read from replicas
- write to leader

- CONSENSUS ALGO
-
- QUORUM
  R+W ≥ N

② sharding (for data older than 3 months)
in a dumb database.