Advanced Fintech Guide to Payment Processing (Interview-Grade Depth)

> Purpose: This guide provides a deep dive into payment processing systems for SaaS and fintech companies, suitable for senior engineer interviews. It covers industry-standard architecture, flows, compliance, and edge cases.

---

1. High-Level Payment Ecosystem

Participants: - Customer - Merchant (SaaS / eCommerce) - Payment Gateway / Processor (e.g., Stri - Acquirer (Merchant Bank) - Card Networks (Visa, Mastercard) - Issuer (Customer Bank)

Core Principles: - Asynchronous communication - Event-driven state management - Regulatory complia (PCI DSS, PSD 2) - Multi-provider abstraction - Idempotency and retry handling

---

2. Participant Responsibilities in Depth

2.1 Customer

- Inputs payment information
- Completes authentication (SCA/ 3 DS)
- Can initiate disputes/chargebacks
- Does not interact with internal payment infrastructure

2.2 Merchant

- Maintains payment and order state machines
- Initiates payment requests to gateway
- Handles asynchronous webhooks for success/failure
- Performs reconciliation with settlement reports
- Manages refunds and partial refunds
- Maintains audit logs for compliance and chargebacks
- Implements idempotency for safe retries

Example State Machine:

```
OrderStatus = {
  Pending,
  PaymentAuthorised,
  PaymentFailed,
  PaymentCaptured,
  Refunded,
  PartiallyRefunded,
  ChargebackReceived
}
```

2.3 Payment Gateway / Processor

- PCI DSS compliant card data handling
- Tokenization of card info
- Fraud scoring and risk rules
- Orchestrates SCA/ 3 DS flows
- Supports multiple payment methods (cards, wallets, open banking)
- Sends reliable webhooks
- Optional: Multi-provider routing

2.4 Acquirer

- Merchant account provider
- Receives authorized payments from gateway
- Submits transactions to card networks
- Manages settlement to merchant bank account
- Handles chargebacks
- Maintains liability and risk

2.5 Card Network

- Routes transactions issuer <-> acquirer
- Defines dispute rules and timelines
- Calculates interchange fees
- Monitors fraud and compliance

2.6 Issuer Bank

- Validates transactions and funds
- Executes SCA if required
- Provides authorization/decline responses
- Handles customer disputes
- Reports chargebacks to card network/acquirer

---

3 . Advanced Payment Lifecycle

3 . 1 Initiation

```
function initiatePayment(orderId, amount, paymentMethod):
    payment = new Payment(orderId, amount, paymentMethod)
    save(payment)
    return gateway.authorise(payment)
```

3 . 2 Authorisation & SCA

- Authorisation reserves funds
- SCA (Strong Customer Authentication) required for EU/UK
- 3 D Secure 2 flow triggers if SCA required
- Soft decline: retry or request additional authentication
- Hard decline: cannot retry

Pseudocode:

```
if response.requiresSCA:
    redirectToBankApp(response.authUrl)
    await webhook for payment.authorised
else if response.declined:
    markPaymentFailed(paymentId)
```

3 . 3 Capture

- Confirms transaction
- Can be manual or automatic
- Triggers clearing and settlement

```
function capturePayment(paymentId):
    gateway.capture(paymentId)
    markPaymentCaptured(paymentId)
```

3 . 4 Clearing & Settlement

- Clearing: exchange transaction details between issuer and acquirer
- Settlement: funds move from issuer → acquirer → merchant
- Usually batched (daily or real-time with modern gateways)

Reconciliation logic:

```
for each settlement in acquirerReport:
    match = findInternalPayment(settlement.reference)
    if match.status != 'Captured':
        raiseAlert()
```

3.5 Refunds & Partial Refunds

- Initiated by merchant
- Separate transaction from original payment
- Updates ledger and triggers webhooks

3.6 Chargebacks

- Customer disputes transaction
- Issuer pulls funds from acquirer
- Merchant receives notification, submits evidence
- Arbitration decides outcome
- Requires strong logging for evidence

4. Multi-Provider Routing

Use-case: Reduce failures, improve coverage, optimize fees

```
function routePayment(payment):
    if payment.currency in ['EUR', 'GBP']:
        provider = Adyen
    else:
        provider = Stripe

    try:
        provider.authorise(payment)
    catch TemporaryFailure:
        fallbackProvider.authorise(payment)
```

5. Event-Driven Architecture

Event flow:

```
PaymentInitiated -> PaymentAuthorised -> PaymentCaptured -> PaymentFailed ->
RefundIssued -> ChargebackReceived
```

- Webhooks are critical for asynchronous updates - Use Outbox pattern to avoid missing events - Store event IDs to ensure idempotency

---

6. Regulatory Compliance

6.1 PCI DSS

- Never store full card numbers
- Use tokenization / hosted fields
- Encrypt sensitive data

6.2 PSD2 / SCA

- Required for EU/UK online transactions
- Two-factor authentication (2 of 3): knowledge, possession, inherence
- 3DS2 used to implement SCA

6.3 Open Banking (UK)

- Customer initiates transfer via bank login
- Instant confirmation, low fees, no chargebacks

---

7. Idempotency & Retry Handling

Idempotency ensures no duplicate charges

```
function authorise(request):
    if exists(request.idempotencyKey):
        return storedResponse
    response = gateway.authorise(request)
    store(request.idempotencyKey, response)
    return response
```

- Essential for webhook retries and network failures

---

8. Ledger & Accounting

- Use double-entry ledger
- Each payment has corresponding debit/credit entries
- Chargebacks and refunds update ledger independently
- Keeps financial system auditable

Example:

```
PaymentCaptured:
    debit: MerchantReceivable
    credit: Revenue
RefundIssued:
    debit: Revenue
    credit: MerchantReceivable
```

---

9. Advanced Edge Cases

- Multi-currency conversion
- Partial authorisation and capture
- Delayed settlement
- Fraud review holds
- Subscription proration and automated retries
- SCA exemptions and risk-based authentication

---

10. Interview-Grade Scenarios

Scenario Questions: 1. How would you handle failed SCA attempts asynchronously? 2. How do
reconcile gateway, acquirer, and ledger when amounts differ? 3 How do you design multi-prov
fallback with idempotency? 4. How to handle partial refund on subscription with proration? 5.
do you prevent duplicate payments on webhook retries?

You should be able to draw: - End-to-end payment flow - Event-driven state machine - Ledger update
each payment/refund/chargeback

---

11. Recommended Learning Path

1. Understand Authorisation vs Capture

2. Implement Stripe / Adyen with webhooks in sandbox
3. Simulate SCA flow with 3DS2
4. Add multi-provider abstraction
5. Implement ledger with double-entry
6. Handle chargebacks and refunds
7. Optimize for reconciliation, idempotency, and reliability

Summary

Payments in fintech are **distributed, asynchronous, regulated, and event-driven**. A senior engineer must master: - Payment flows & lifecycle - Multi-provider abstraction - SCA & PSD2 compliance - Eve
state machines - Idempotency & retries - Ledger & reconciliation - Edge cases like chargebacks,
multi-currency

This guide equips engineers to **design, implement, and interview** for payment systems in SaaS/fintech environments.