PAYMENTS SIGNAL REFERENCE ARCHITECTURE
SYNTHETIC / TRAINING ONLY

Payment lifecycle, state, event, queue, and idempotency architecture

A payment is not one database row that changes mysteriously. It is an ordered history of commands, proven outcomes, deadlines, and external facts.

Version: Reference architecture v1
Reviewed: 2026-07-23

## REFERENCE POSTURE

The blueprint assumes one logical payment-state owner and uses one queue and one journal. Real estates may shard by product or region and may use several brokers, stores, and recovery coordinators.

## COMPONENTS

01 Command intake [service]
Owner: Payment platform
Receives a request to do something and assigns it one identity.

02 Payment state machine [application]
Owner: Payment platform
Allows only transitions that make sense for the payment s proven history.

03 Immutable event journal [data-store]
Owner: Payment platform
Keeps the ordered facts from which the current payment state can be explained.

04 Transactional outbox and queue [service]
Owner: Payment platform
Carries commands and events without losing them between the payment record and another system.

05 Timer and deadline service [control]
Owner: Payment platform
Turns a business deadline into a recorded event instead of an untracked sleep.

06 Ledger system [ledger]
Owner: Core banking and finance
Owns the accounting fact independently from the payment engine s processing state.

07 External network [external-network]
Owner: External network or market infrastructure
Owns delivery, acceptance, clearing, or settlement facts outside the bank.

08 Read model and status API [service]
Owner: Payment platform
Presents a useful current view without becoming a second source of truth.

09 Exception and reconciliation desk [operations]
Owner: Payment operations
Resolves cases where internal state and external evidence do not yet tell one consistent story.

## INTERFACES

01 Expected-state command
state-command -> state-machine | api | synchronous

02 Append state event
state-machine -> state-journal | event | asynchronous

03 Committed work
state-journal -> state-outbox | event | asynchronous
04 Posting command
state-outbox -> state-ledger | posting | asynchronous
05 Network command
state-outbox -> state-network | message | asynchronous
06 Network result
state-network -> state-machine | event | asynchronous
07 Deadline event
state-timer -> state-machine | control | asynchronous
08 Projection feed
state-journal -> state-projection | event | asynchronous
09 Case evidence
state-projection -> state-operations | operator | operator

TRACES AND STRESS CASES

TRACE: One durable state journey

1. Receive one command - COMMAND_RECEIVED
2. Accept one transition - TRANSITION_ACCEPTED
3. Write the fact - RECORDED
4. Publish committed work - DISPATCHED
5. Record the accounting result - POSTED
6. Release the network instruction - SENT
7. Apply the external result - NETWORK_CONFIRMED
8. Present an explainable status - VISIBLE

STRESS: Queue delivers twice

Settlement: Depends on the first network delivery; the repeat cannot be assumed new.
Funds: The posting may already exist under the original key.
Owner: Payment-platform operations
Safe action: Consumers return the existing result for the same identity and reject changed reuse.

STRESS: Timeout followed by late success

Settlement: Unknown at timeout; the late status may prove that settlement succeeded.
Funds: Any timeout restoration or release must be reconciled against the late success.
Owner: Payment operations with payment-platform support
Safe action: Treat timeout as uncertainty, block blind retry, and reconcile the late status against the original payment.

STRESS: Operator acts on stale status

Settlement: The current state may already be final.
Funds: No second release or compensating posting is permitted from a stale view.
Owner: Payment operations
Safe action: Use expected-version checks and reject the stale command before it changes anything.