CAPSTONE · SOLUTION ARCHITECTURE

National Instant Payment Rail

A real-time, 24/7 account-to-account transfer switch — RAAST / UPI / Pix / FedNow class

Design targets: **60M txns/day** · **~42k ledger writes/s peak** · **p99 < 1s** · **99.999% uptime**

Prepared by: Your Name · your.email@example.com | System Design Masterclass, Cohort 4 · Umer Farooq (CTO, MRS Technologies)

THE PROBLEM

What a national instant payment rail must do

Traditional interbank transfers clear in batches over hours or days. A real-time rail settles person-to-person and person-to-merchant payments in under a second, every day of the year.

1

Always on

24/7/365 clearing. No batch windows, no bank-holiday delays — funds are usable by the payee within seconds.

2

Interoperable

One switch connects every participant bank and wallet. A payer at Bank A pays a payee at Bank B through a shared directory of aliases.

3

Pay by alias

Send to a mobile number, national ID, or a human-readable handle — no need to exchange raw account numbers.

4

Irrevocable & final

Once cleared, a transfer is final and guaranteed. The money must never be lost, duplicated, or double-spent.

Analogues in production: Pakistan RAAST, India UPI (~500M+ txns/day), Brazil Pix, US FedNow. This design is switch-side (the central clearing rail), not a single bank's app.

FUNCTIONAL REQUIREMENTS

What the system does — user actions, inputs, outputs

1. **Register & resolve alias**
   Link a mobile no. / national ID / handle to a bank account. Resolve alias → (bank, account token, title).

2. **Push payment (credit transfer)**
   Payer initiates a transfer by alias or account. Input: payer, payee, amount. Output: cleared / rejected + reference.

3. **Request-to-pay (collect)**
   Payee raises a request; payer approves. Powers merchant and biller collections.

4. **Real-time authorization & routing**
   Validate, risk-score, route to payee bank, obtain accept/reject — all within the latency budget.

5. **Settlement across banks**
   Post double-entry ledger movements and net/gross settle participant positions with the central bank.

6. **Status, notifications & history**
   Both parties get near-instant confirmation; status is queryable; completed transfers are retained.

7. **Reversals & refunds**
   Failed or timed-out transfers auto-reverse; disputes and refunds are supported end-to-end.

8. **Idempotent retries**
   A retried request with the same key never debits twice — exactly-once money movement.

NON-FUNCTIONAL REQUIREMENTS

How well it must perform — the numbers the design is held to

LATENCY

End-to-end p99 < 1s

Switch processing budget < 300 ms; alias resolve < 50 ms (cached).

CONSISTENCY

Strong where money moves

Per-shard ACID; end-to-end atomic via guaranteed saga. Directory/notify may be eventual.

AVAILABILITY

99.999% (≈ 5 min/yr)

Active-active, multi-AZ + multi-region. No batch downtime.

DURABILITY

RPO = 0 for the ledger

Zero committed-transaction loss. Immutable, auditable trail.

THROUGHPUT

Peak ≈ 7k txn/s

≈ 42k ledger writes/s at peak with headroom for 20% YoY growth.

SECURITY

Bank-grade, regulated

mTLS between participants, signed ISO 20022 messages, HSM keys, tokenized accounts.

SCALE & GROWTH ASSUMPTIONS

Where we start, and how it grows

100M

registered aliases

bankable population

30M

daily active users

~30% of aliases transact/day

60M

transactions / day

2 txns per active user

420K

peak-minute volume

≈10× avg, salary-day evening

Assumptions behind the numbers

- **Average vs peak**
  Retail payments cluster hard: assume ~40% of daily volume lands in the single busiest hour (salary day, festival evening) → 24M/3,600s ≈ 6,700 txn/s ≈ ×10 over the 694 txn/s average. Sized to peak.
- **Postings per transfer**
  Each transfer = 4 ledger postings (payer→suspense, suspense→payee across two shards) → amplifies write load 4×.
- **Growth**
  20% YoY compounding adoption (conservative vs UPI's early curve). Capacity + storage planned on a 5-year horizon.
- **Read : write mix**
  Alias resolves + status polls dominate reads (cacheable). Money movement dominates durable writes (not cacheable).

QPS ESTIMATION

Reads vs writes, step by step, sized to peak

WRITE PATH (durable, not cacheable)

Base transaction rate	`60M / 86,400 s`	**694 txn/s**
Peak transactions (×10)	`694 × 10`	**6,940 txn/s**
Ledger postings (4 / txn)	`6,940 × 4`	**27,760 /s**
Transaction / request row	`6,940 × 1`	**6,940 /s**
Fraud / risk event writes	`6,940 × 1`	**6,940 /s**

Peak durable writes ≈ 42,000 / s

READ PATH (cacheable)

Alias resolves (1.5 / txn)	`6,940 × 1.5`	**10,410 /s**
Payment status polls (2 / txn)	`6,940 × 2`	**13,880 /s**
Directory / limit lookups	`≈ 1 / txn`	**6,940 /s**
History & receipt fetch	`≈ 0.5 / txn`	**3,470 /s**

~85% of reads are served from Redis (alias directory + hot status), so the relational tier only sees the durable write load and cache misses.

Peak reads ≈ 34,700 / s

Settlement writes are netted and batched asynchronously (see Settlement & liquidity) — deliberately excluded from the peak synchronous write path.

STORAGE ESTIMATION

Row sizes → daily volume → 5-year growth

Row sizes

Transaction (instruction)	id+refs 96 · amount/status 24 · times 24 · idem+meta 48	**≈ 192 B**
Ledger posting	entry 8 · txn 16 · acct 8 · amt+bal 16 · type+time 16	**≈ 64 B**
Alias / directory	id 16 · value 32 · bank 4 · token 32 · title 64 · meta 20	**≈ 168 B**

Daily transactional volume

Transactions	`192 B × 60M`	**11.5 GB / day**
Ledger (4 postings/txn)	`64 B × 240M`	**15.4 GB / day**
Directory (mostly static)	`168 B × 100M`	**16.8 GB total**

≈ 27 GB/day new transactional data → ~9.9 TB/year (single copy)

5-year cumulative storage

20% YoY growth · replication factor 3

Year	New (TB)	Cumulative	A =3
1	9.9	9.9 TB	29.7 TB
2	11.9	21.8 TB	65.4 TB
3	14.3	36.1 TB	108.3 TB
4	17.1	53.2 TB	159.6 TB
5	**20.6**	**73.8 TB**	**221.4 TB**

Raw row bytes shown; secondary indexes + the immutable audit log add ~2–3×. Hot 90 days on fast distributed SQL, older partitions tier to cold object storage (7-yr retention).

DATA MODEL

Core entities and how they relate

P Participant
A member bank or wallet on the rail. Owns accounts and issues aliases.

A Alias / Directory
Maps a mobile/ID/handle to (participant, account token, title). The routing lookup.

A Account
A settlement or virtual account at a participant. Holds a balance position.

T Transaction
One transfer instruction: payer, payee, amount, status, idempotency key.

L Ledger Posting
An immutable debit/credit line. Transaction → many postings (double-entry).

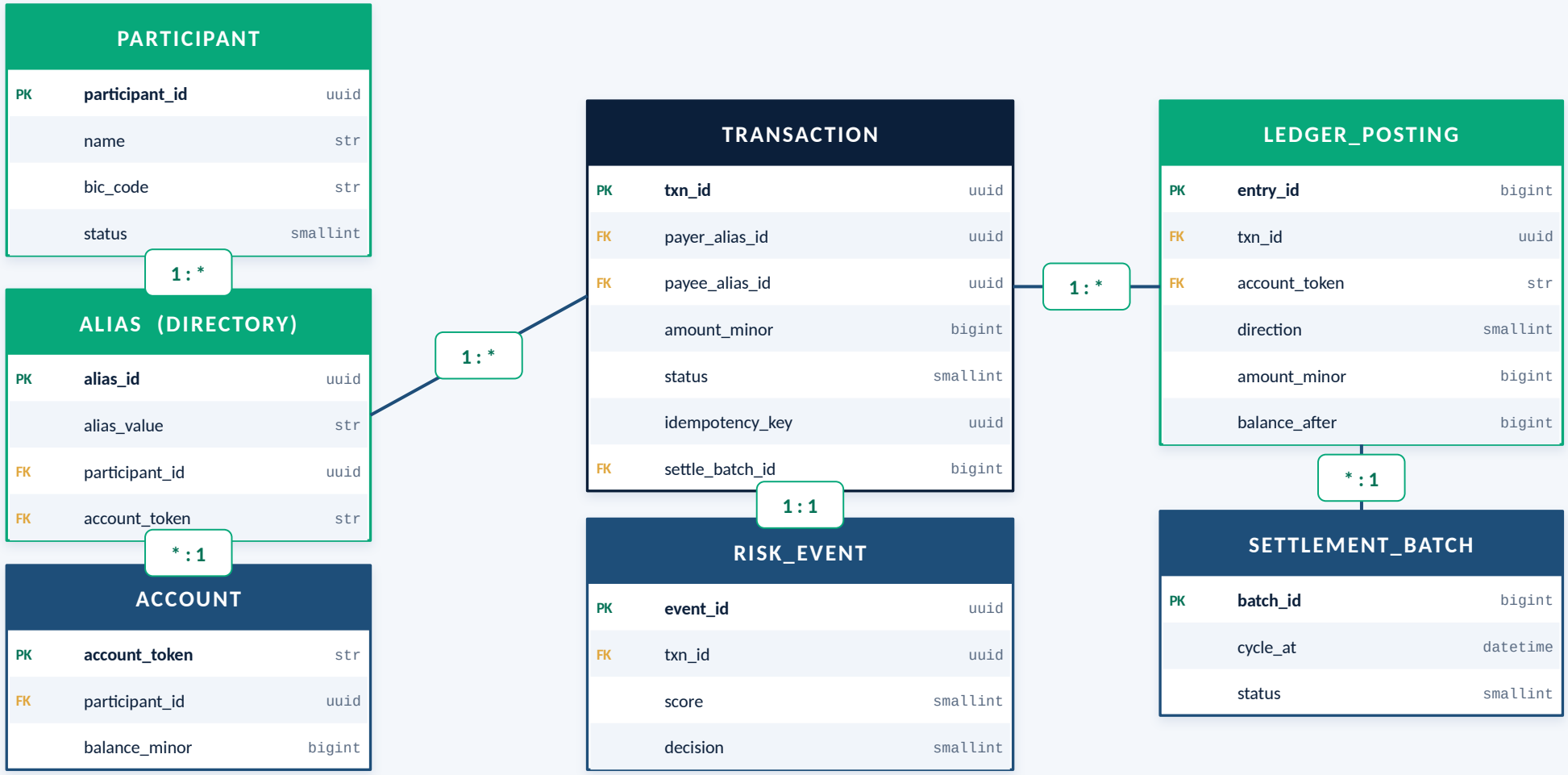
S Settlement Batch
A net-settlement cycle; groups postings into inter-bank positions.

R Risk / Fraud Event
The score & decision attached to a transaction at authorization time.

Key relationships: Participant 1—* Alias · Alias *—1 Account · Transaction 1—* Ledger Posting · Transaction 1—1 Risk Event · Settlement Batch 1—* Postings

ENTITY-RELATIONSHIP DIAGRAM

Double-entry ledger at the center



Transaction → many Ledger postings drives sharding.

PK primary key FK foreign key 1 : * one-to-many 1 : 1 one-to-one

API DESIGN

Versioned REST over signed ISO 20022 messages

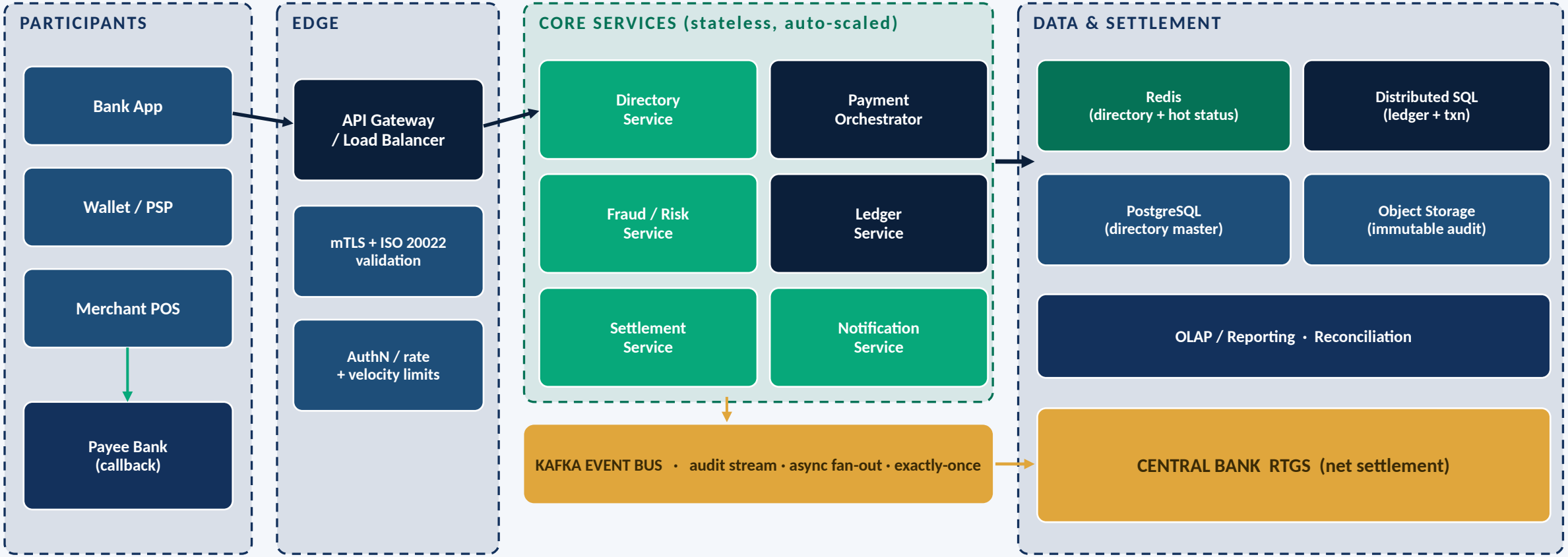
METHOD	ENDPOINT	PURPOSE
POST	/v1/aliases	Register an alias → account
GET	/v1/aliases/resolve	Resolve alias → bank + token
POST	/v1/payments	Initiate a push transfer
POST	/v1/payments/{id}/confirm	Payer authorizes (SCA)
GET	/v1/payments/{id}	Query status
POST	/v1/payments/{id}/reversal	Reverse / refund
POST	/v1/requests	Request-to-pay (collect)
POST	/v1/requests/{id}/approve	Payer approves a request

```
POST /v1/payments

Idempotency-Key: a1b2-c3d4
{
  "payer_alias": "+92300...",
  "payee_alias": "merchant@rail",
  "amount_minor": 250000,
  "currency": "PKR"
}



201 →  response
{
  "payment_id": "uuid",
  "status": "PENDING",
  "e2e_ref": "E2E-8842..."
}
```

Every payment carries an idempotency key (safe retries) and an end-to-end reference that becomes the distributed-trace id across all services.

HIGH-LEVEL ARCHITECTURE

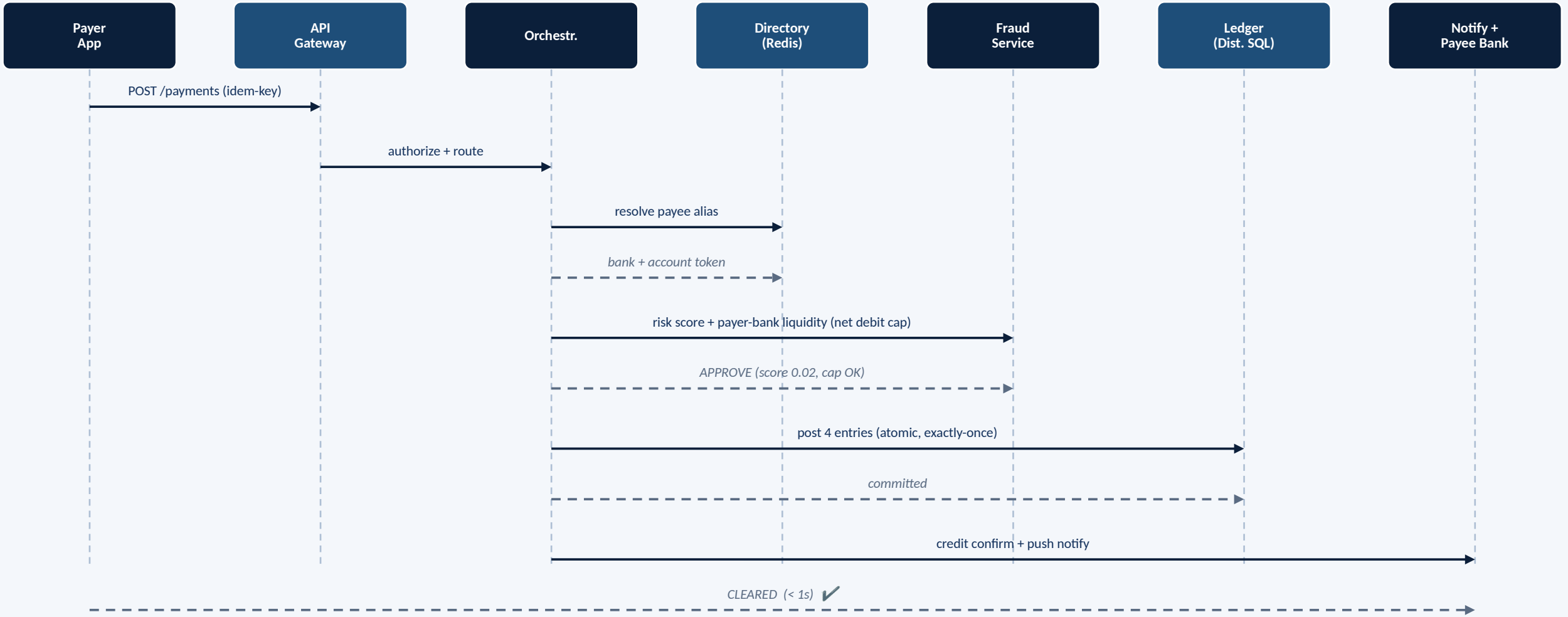
Participants → edge → core services → data



Money path (sync, strongly consistent): resolve alias → risk-score → orchestrate → post double-entry ledger → confirm. **Event path (async):** every service emits to Kafka → immutable audit, reconciliation, OLAP, and net settlement with the central bank.

REQUEST FLOW

Happy-path push transfer, end to end

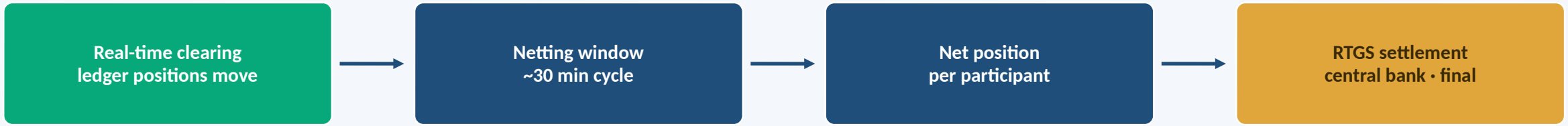


Net settlement across participant positions runs asynchronously on the Kafka event stream — off the sub-second money path.

SETTLEMENT & LIQUIDITY

Clearing is instant — settlement nets on a cycle

End users see an irrevocable transfer in under a second. Actual central-bank money moves between banks on a netted cycle — the split that makes a national rail affordable.



1 Deferred net settlement

60M gross transfers over a window net down to a few hundred inter-bank positions — orders of magnitude fewer, cheaper central-bank settlements than settling every transfer gross.

2 Prefunding & net debit cap

Each participant prefunds a collateral account at the central bank. The switch verifies available liquidity before authorizing, so every cleared transfer is guaranteed to settle — even if a participant defaults.

3 Finality & reconciliation

Cleared = irrevocable to end users; net positions settle with finality each cycle. Σ debits = Σ credits makes every position provable. High-value transfers can settle gross in real time.

Instant clearing + netted settlement is exactly how RAAST, UPI and Pix reconcile real-time UX with central-bank settlement economics.

WRITE-INTENSIVE CORE

The append-only double-entry ledger

1 Append-only, never update

Postings are immutable inserts — no in-place balance row to lock. Removes write contention at the source. Balance = checkpoint + delta.

2 Exactly-once

Idempotency key dedupes retries; Kafka transactional producer + idempotent consumer guarantee each transfer is processed once, even across failures.

3 Double-entry invariant

Σ debits = Σ credits enforced per transaction. A transfer that can't post fully doesn't post at all — money is never created or lost.

4 4 atomic postings / txn

payer → suspense (shard A), suspense → payee (shard B). ~42k writes/s at peak, growing to ~104k/s over 5 yr — met by adding ledger shards linearly.

SCALABILITY — SHARDING & PARTITIONING

Even distribution, and the hot-account problem

Directory

`Shard by alias hash`

Reads dominate and are Redis-cached. Uniform hash → no hotspot. Read replicas absorb misses.

Transactions

`Hash(txn_id) + time partition`

Random txn_id spreads writes evenly; daily partitions make archival & retention a partition drop.

Ledger

`Shard by account_token`

Balanced for normal accounts — but high-volume payees (a biller, a big merchant) concentrate on one shard.

Solving the hot account

1 Balance striping

A hot account's balance is split into N sub-balance buckets. Credits round-robin across buckets on separate shards; reads sum the buckets. Single-row contention disappears.

2 Append-only postings

No balance row is locked on the write path — postings are contention-free inserts. The materialized balance is refreshed from a checkpoint asynchronously.

3 Suspense per shard

Cross-shard debit+credit avoids distributed 2PC on the hot path: debit → local suspense (shard-local txn), then settle payee via saga. Bounded, guaranteed, reversible.

SCALING BOTTLENECKS

Where it breaks first — and the mitigation

Ledger hot account

Risk: Popular payee → single-shard write hotspot & balance-row contention.

Fix: Balance striping + append-only postings + suspense/saga (prev. slide).

Directory read storm

Risk: Peak alias resolves can overwhelm the master.

Fix: Redis cache (~85% hit) + read replicas; directory is read-mostly.

Fraud in the hot path

Risk: Heavy ML scoring inline would blow the latency budget.

Fix: Lightweight inline rules; async feature pre-compute; cached scores.

Settlement contention

Risk: Per-transfer RTGS calls don't scale.

Fix: Net settlement in batched cycles — thousands of transfers → one position.

Kafka partition skew

Risk: Keying by hot account skews partitions.

Fix: Partition by txn_id, not account; consumers scale per partition.

Single points of failure

Risk: A regional switch outage halts payments nationwide.

Fix: Active-active multi-region; stateless services; quorum-replicated ledger.

HORIZONTAL VS VERTICAL SCALING

Add machines, or bigger machines — and why

SCALE HORIZONTALLY (default)

Stateless services
Gateway, orchestrator, directory, fraud, notification — add pods behind the LB; no shared state.

Kafka
More partitions + brokers → linear write throughput and consumer parallelism.

Redis
Sharded cluster; add shards as directory/status traffic grows.

Distributed SQL ledger
Add nodes → more shards & quorum capacity while keeping ACID.

SCALE VERTICALLY (where it's right)

HSM crypto appliances
Signing/key ops have fixed throughput per unit; scale up + a small fixed pool, not elastic pods.

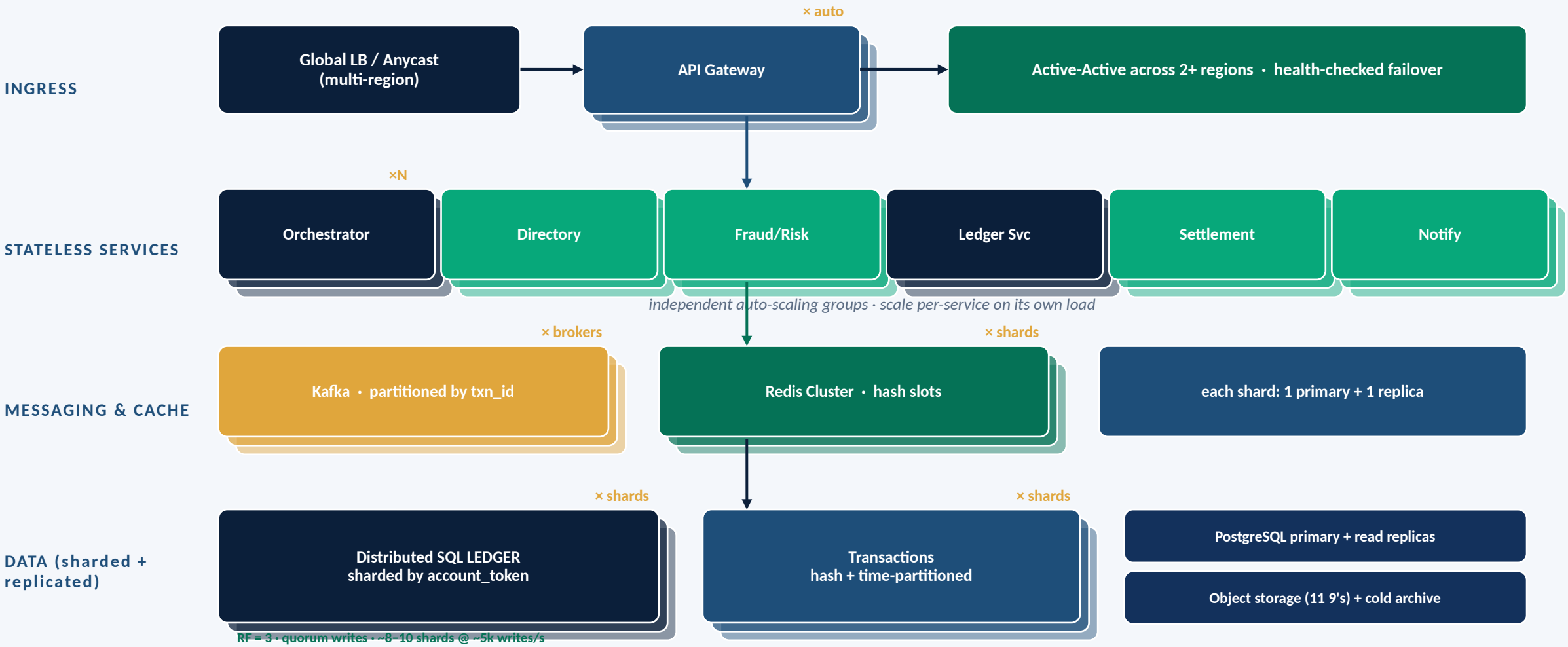
Directory master (early)
A single strong PostgreSQL primary is simplest before read volume forces replicas/sharding.

Settlement coordinator
Ordering & netting is inherently coordinated — a bigger, HA-paired node beats fanning it out.

Trade-off
Vertical has a ceiling & a cost knee; used only where coordination or crypto makes horizontal unsafe.

SCALABILITY DIAGRAM

How every tier scales together



Every tier scales out independently; the only vertical-scaled pieces (HSM, settlement coordinator) sit behind the stateless layer and are HA-paired.

DATA DURABILITY & RECOVERY

Money must survive any single failure

0

RPO = 0 (ledger)

Synchronous quorum writes (RF=3, majority ack). A committed transfer is on ≥2 nodes before the client hears 'cleared'.

↻

RTO < 1 min

Automatic leader election on node loss; stateless services re-register in seconds behind health checks.

Multi-region DR

Synchronous within region, asynchronous cross-region replica. Region loss → promote DR region, active-active absorbs.

✓

Kafka guarantees

RF=3, min.insync.replicas=2, acks=all — no acknowledged event is lost; audit stream is complete.

▤

Immutable audit + PITR

WAL archiving enables point-in-time recovery; append-only postings + object-storage audit give a tamper-evident trail.

⇄

Daily reconciliation

Positions reconciled against every participant and the central bank settlement account; breaks alert immediately.

TECHNOLOGY CHOICES, JUSTIFIED

Why each, and what we rejected

Distributed SQL

for the ledger

Per-shard ACID + horizontal scale. Cross-shard ACID is available; we choose a saga on the hot path for latency.

Rejected: vs sharded PostgreSQL (manual cross-shard txns) / vs eventually-consistent NoSQL (unsafe for money).

Redis

directory + hot status

Sub-ms geo/kv reads; cluster sharding; absorbs ~85% of read load.

Rejected: vs hitting the SQL master for every alias resolve.

Kafka

event & audit backbone

High-throughput, ordered, replayable; transactional producer = exactly-once.

Rejected: vs RabbitMQ (lower throughput, weaker replay) for a streaming audit log.

Object storage

immutable audit + archive

11 nines durability, cheap cold tier for 7-yr retention.

Rejected: vs keeping years of postings on hot SSD (cost).

PostgreSQL

directory master

Simple, strongly-consistent source of truth for a read-mostly dataset.

Rejected: vs premature sharding before read volume demands it.

HSM

key custody & signing

Regulated key storage; hardware-enforced message signing.

Rejected: vs software keys (fails bank-grade / compliance).

BEYOND THE HAPPY PATH

Security · observability · cost

SECURITY

- mTLS between every participant & the switch
- Signed ISO 20022 messages; HSM-held keys
- Tokenized account refs — switch stores no raw PAN/PII
- Velocity + anomaly limits at the edge
- Encryption at rest & in transit; least-privilege

OBSERVABILITY

- Distributed tracing on every e2e reference
- p99 latency & success-rate SLOs per service
- Settlement-lag & reconciliation-break alerts
- Real-time fraud & throughput dashboards
- Structured audit events replayable from Kafka

COST

- Storage tiering: hot 90d → warm → cold archive
- ARM/Graviton compute for stateless services
- Redis offload shrinks the SQL tier (~85% reads)
- Net settlement → far fewer costly RTGS calls
- Autoscale to peak, scale in off-peak

ALIGNMENT WITH REQUIREMENTS

Every requirement → the decision that meets it

REQUIREMENT	MET BY	AT SCALE
Exactly-once money movement	Idempotency keys + Kafka transactions + double-entry invariant	Holds at ~42k ledger writes/s
p99 < 1s, budget < 300 ms	Redis-cached resolve, inline lightweight risk, async settlement off hot path	Peak ≈ 7k txn/s
Strong consistency (money)	Per-shard ACID, RF=3 quorum writes; eventual only for directory/notify	Cross-shard via guaranteed saga
99.999% availability	Active-active multi-region, stateless services, automatic failover	No batch downtime, 24/7
RPO = 0 durability	Synchronous quorum commit + immutable audit + PITR	Reconciled daily vs central bank
20% YoY growth headroom	Independent horizontal scaling per tier; sharded, replicated data	5-yr storage ≈ 221 TB (RF=3)
Hotspot-free distribution	Hash sharding + balance striping for hot payee accounts	Removes single-row contention
Guaranteed settlement finality	Prefunding + net debit cap verified at authorization; deferred net settlement via RTGS	60M transfers → ~100s of RTGS legs

A rail that clears in under a second,
and never loses a rupee.

60M	~42k	p99 < 1s	99.999%	221 TB
txns / day	ledger writes/s peak	end to end	availability	5-yr @ RF=3

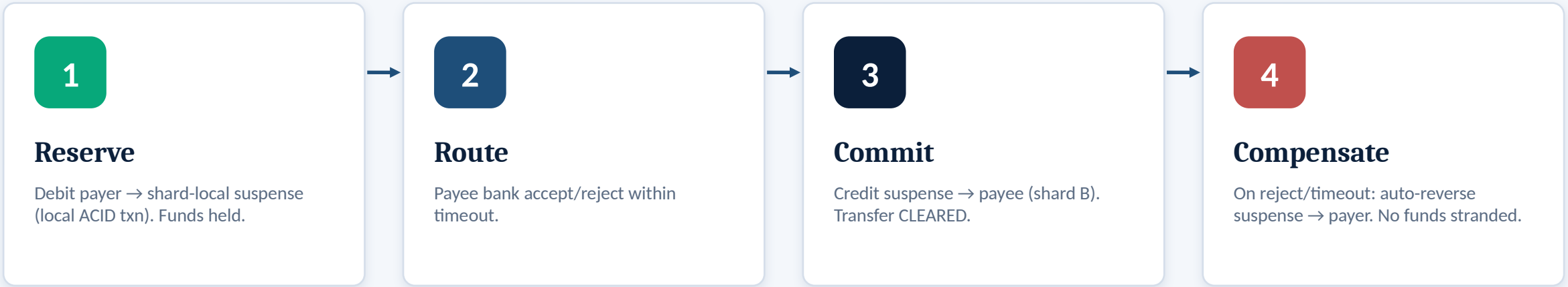
Write-intensive double-entry ledger · hotspot-free sharding · strong consistency where money moves, eventual where it doesn't.

Thank you.

APPENDIX

Cross-shard transfer: saga & compensation

When payer and payee sit on different ledger shards, we avoid distributed 2PC on the hot path and use a durable saga with a compensating reversal.



Trade-off: a sub-second window where funds sit in suspense (payer debited, payee not yet credited). Bounded, guaranteed by the saga, and always reversible — chosen over 2PC's latency & lock cost on the hot path.

Invariant upheld throughout: Σ debits = Σ credits at every step. If a compensation itself fails, the durable saga log retries it to completion; any residual break surfaces in daily reconciliation against the suspense account — which must net to zero.