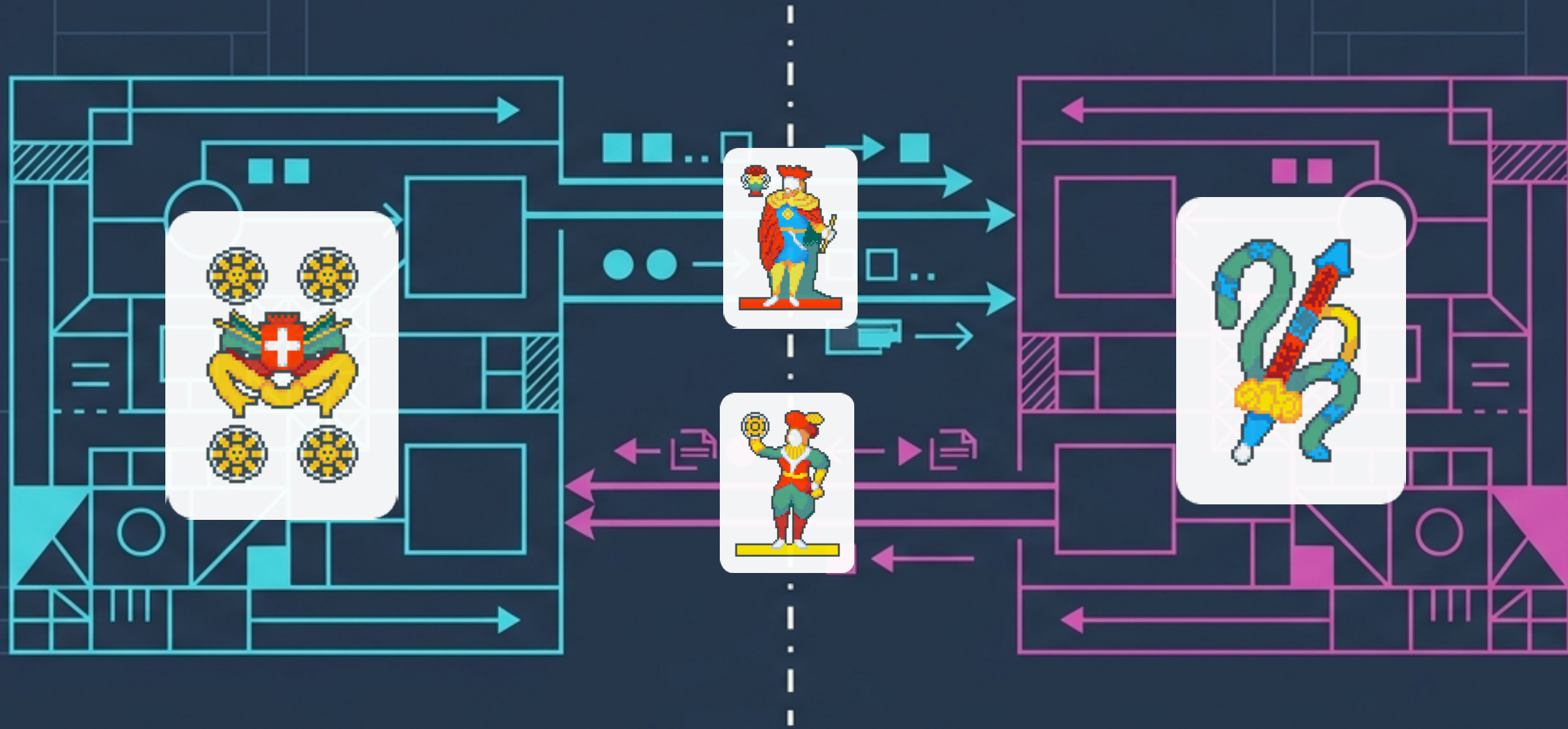
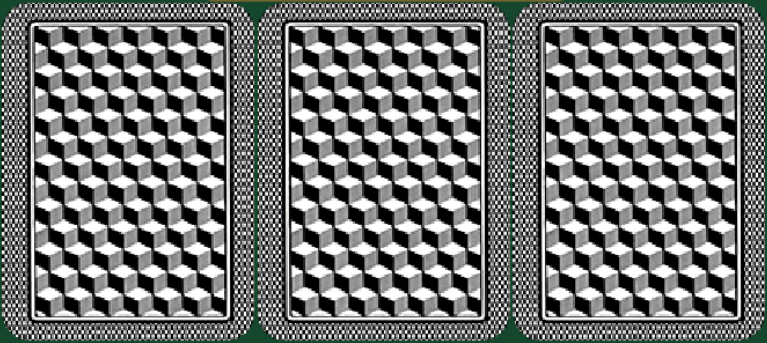




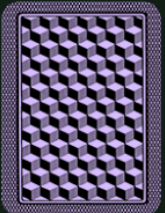
SCOPA2

a distributed, multiplayer reinterpretation of the traditional Italian card game Scopa

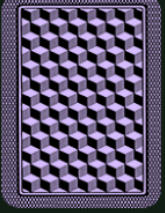
YOUR TURN



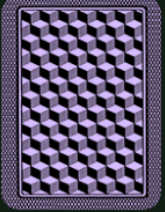
♦ SHOP ♦



San Pantal...



San Pantal...



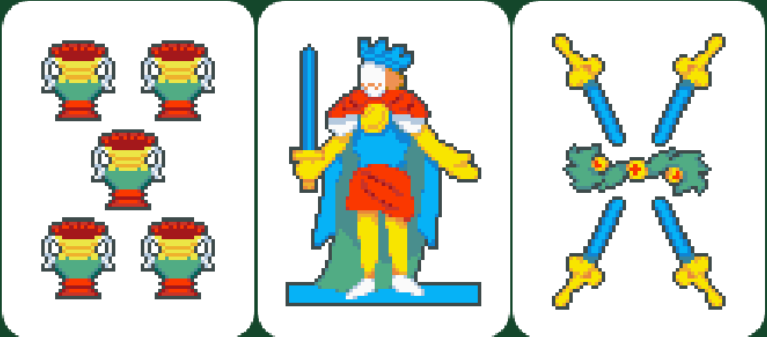
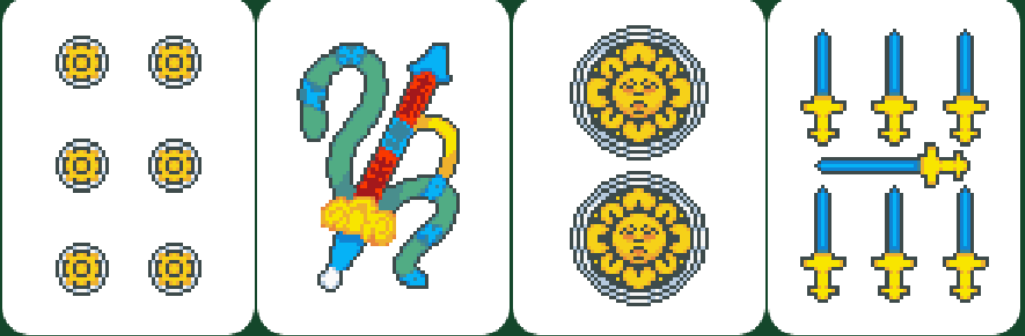
Santa Cate...

DECK



30

cards left



0

OPPONENT

Points: 0

Captured

0

Scope

0

0

YOU

Points: 0

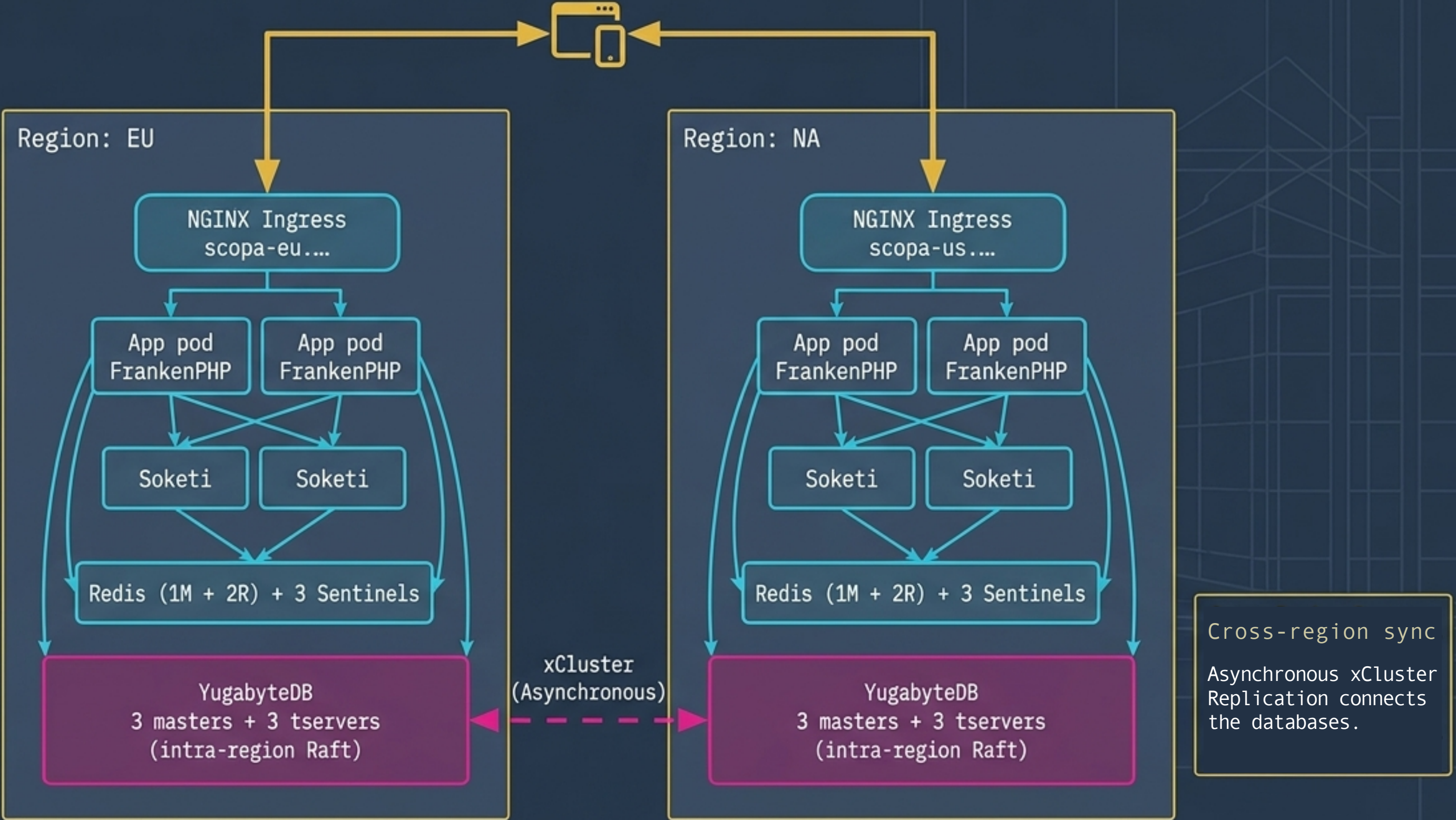
Captured

0

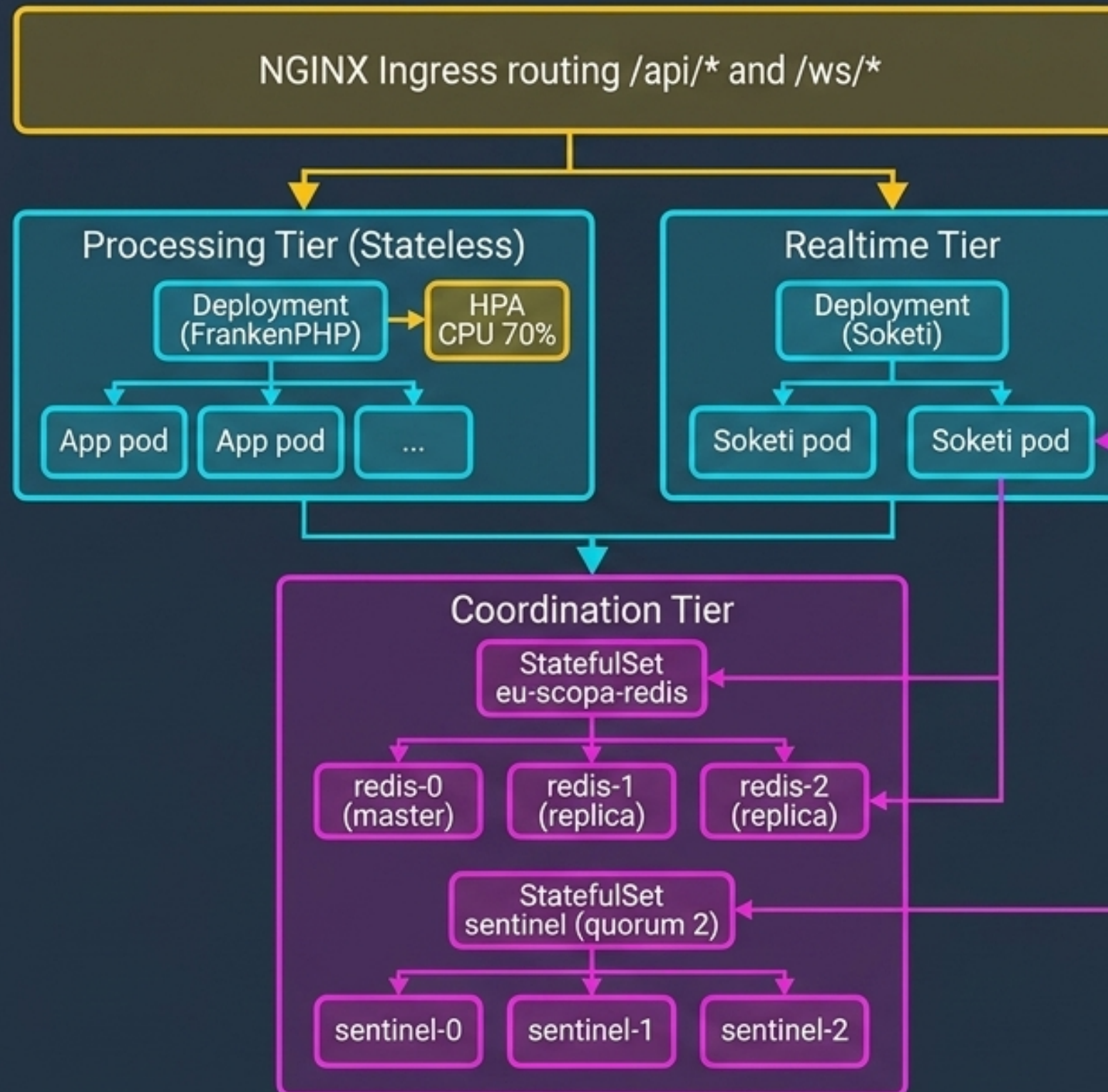
Scope

0

Global Multi-Region Topology



Regional Kubernetes Architecture

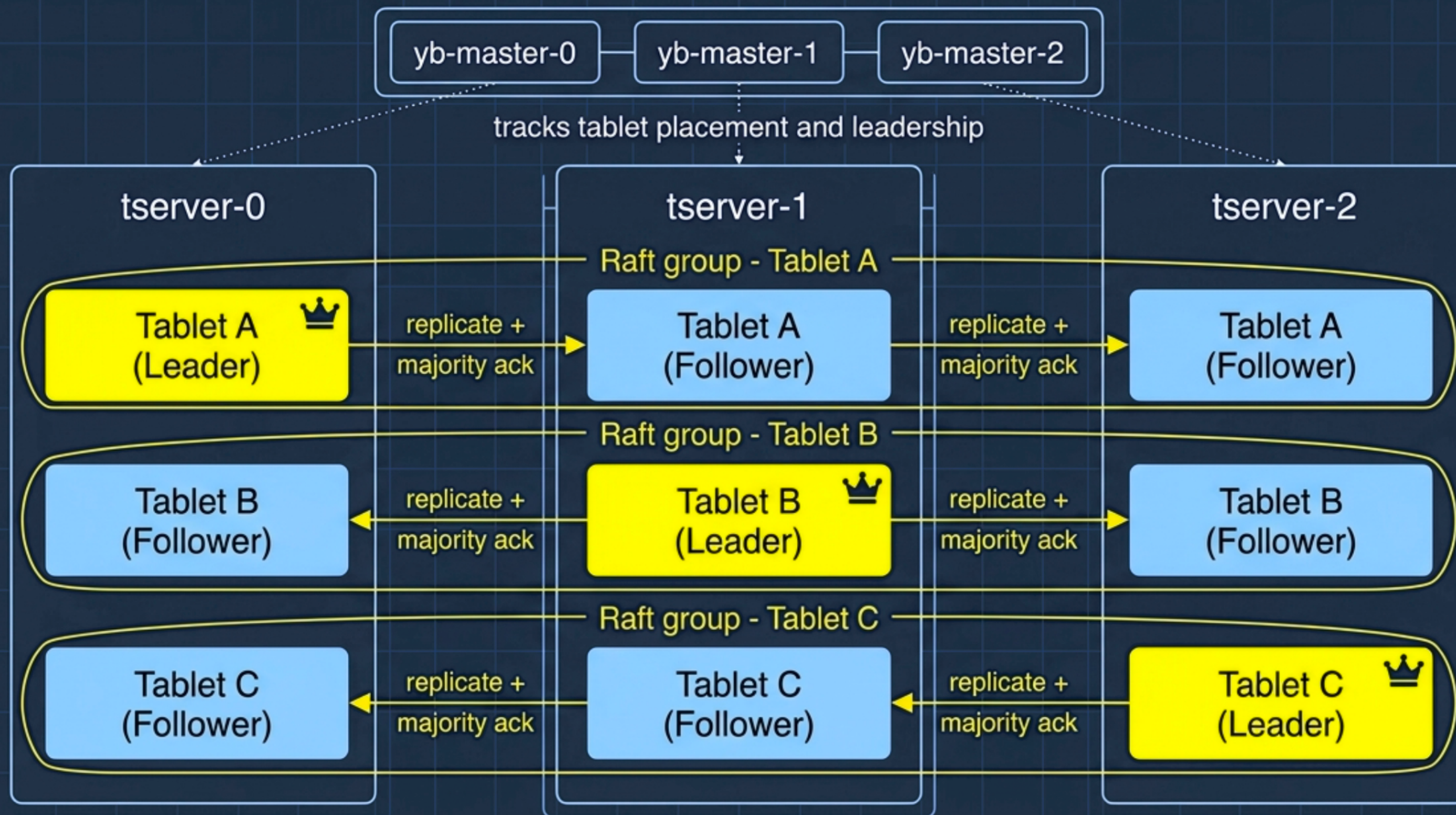


Ingress Routing: NGINX handles TLS termination and path-based routing.

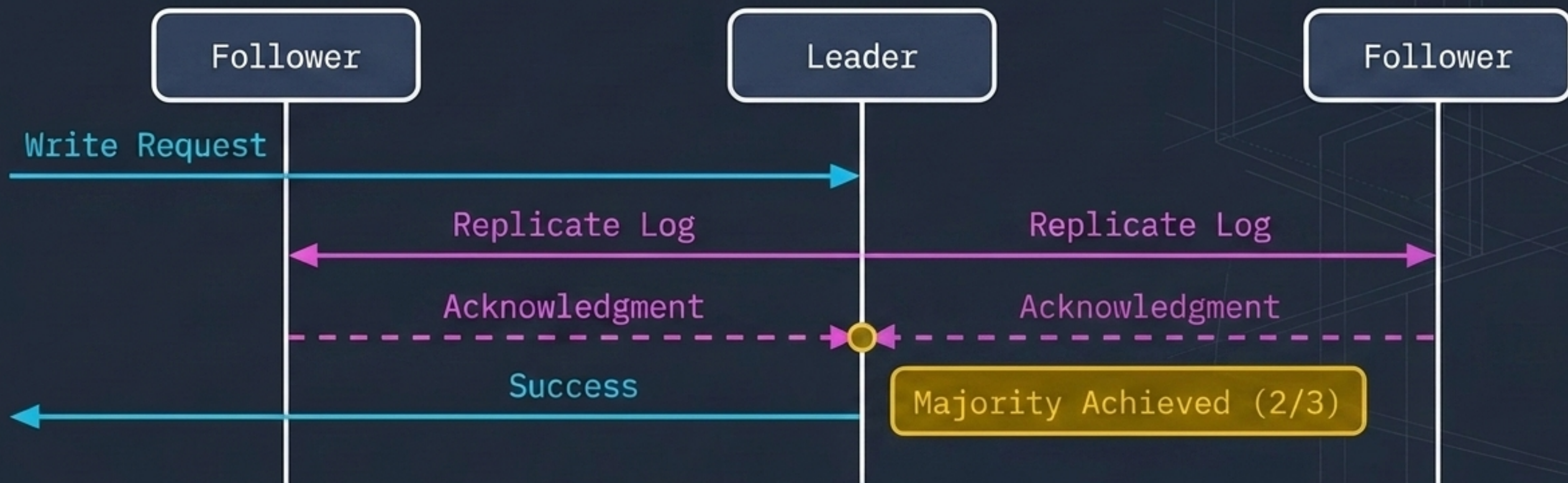
Stateless Processing: FrankenPHP keeps the application resident in memory. Pods hold no session state, allowing instant horizontal scaling via CPU targets (HPA).

Shared Cache State: Soketi pods share state via a Redis adapter to ensure cross-pod fan-out; a message published to Redis is delivered to clients on any Soketi pod.

Deconstructing the YugabyteDB Data Tier



Intra-Region Consensus via Raft

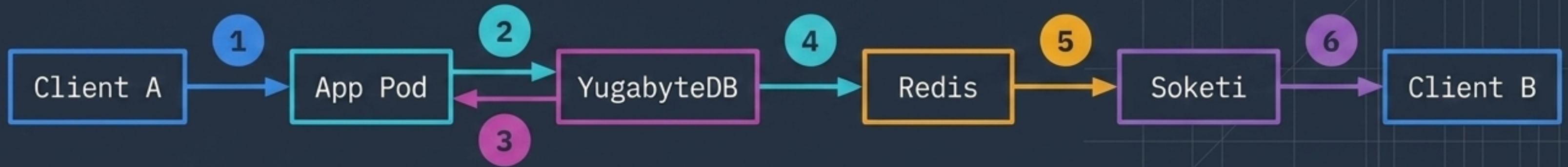


State Machine Replication:
YugabyteDB hides a Raft-replicated log behind a standard PostgreSQL wire protocol.

Synchronous Quorum: A tablet leader requires acknowledgment from a majority (2 out of 3 nodes) before confirming a write.

The Consistency Guarantee:
A write to the event log either commits cluster-wide or fails visibly. No stale reads, no split-brain.

The Life of a Distributed Move



1. Action: Client A POSTs a move payload.

2. Lock & Validate: App pod opens a database transaction, row-locks the game state, and validates the move against a deterministic engine.

3. Append: Move is appended to YugabyteDB (Raft Quorum achieved).

4. Broadcast: App pod publishes an event to the local Redis adapter.

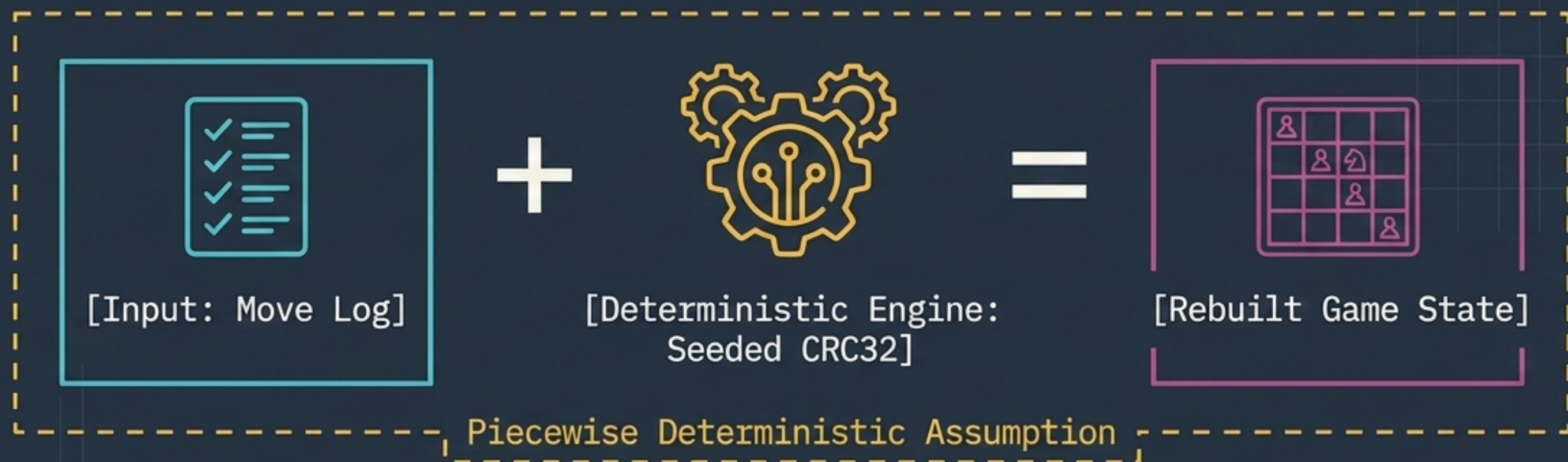
5. Fan-out: Redis pushes the event to all Sockets pods.

6. Delivery: Client B receives the updated state via WebSocket in near-realtime.

Chaos Matrix: Automated Fault Tolerance

Action	Expected outcome	Result
Kill one app pod	LB drops failed pod, retry on peer pod returns 200	PASS
Kill all app pods in a region	Client times out, fails over to other region, state preserved	PASS
Scale Soketi to one replica then back to two	Clients reconnect on the surviving pod	PASS
Kill a Redis follower	No visible effect	PASS
Kill Redis master	Sentinels elect a new master, traffic resumes	PASS
Kill a TServer leader	Raft re-election, query latency spike ~3 s	PASS
Reduce TServer to one replica	Writes fail; reads via follower-reads, manual intervention required	PASS
Delete an entire region namespace	Client failover to surviving region	PASS

The Architecture of Determinism



The system delegates hard distributed problems (consensus, leader election) to infrastructure (YugabyteDB, Redis Sentinel, K3s).

This frees the application layer to act purely as a stateless, event-sourced engine. Pessimistic Message Logging ensures every external input is synchronously persisted to the Raft log before its effect becomes visible.

The Result: Any application pod can rebuild any state instantly by replaying the totally ordered event log. Recovery is reduced to pure deterministic replay.

Multi-Domain PACELC Optimization

Intra-Region (YugabyteDB Raft)

PACELC Stance: PC / EC

If Partition (P): Choose Consistency (C). Raft rejects writes without a majority. No split-brain.

Else (E): Choose Consistency (C). The system gladly pays a few milliseconds of Raft quorum latency to achieve linearizable writes.

Inter-Region (xCluster)

PACELC Stance: PA / EL

If Partition (P): Choose Availability (A). A severed region continues serving its own active sessions without blocking.

Else (E): Choose Latency (L). Writes are replicated asynchronously. An EU write is never blocked waiting for a US mirror acknowledgment.