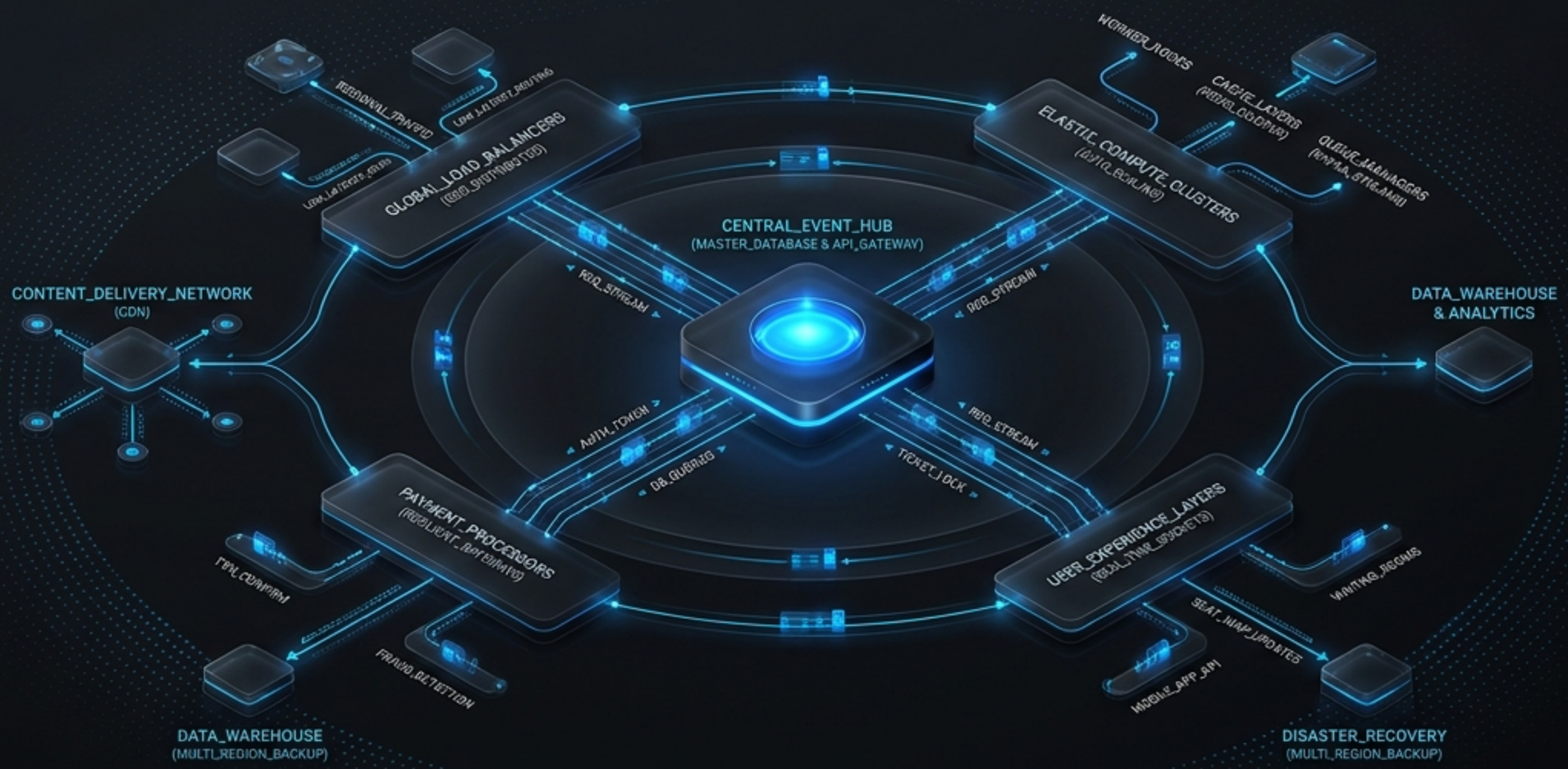
ARCHITECTING FOR THE SURGE

Designing a massive-scale ticket booking system resilient to extreme demand.



Unprecedented demand shatters standard ticketing architectures

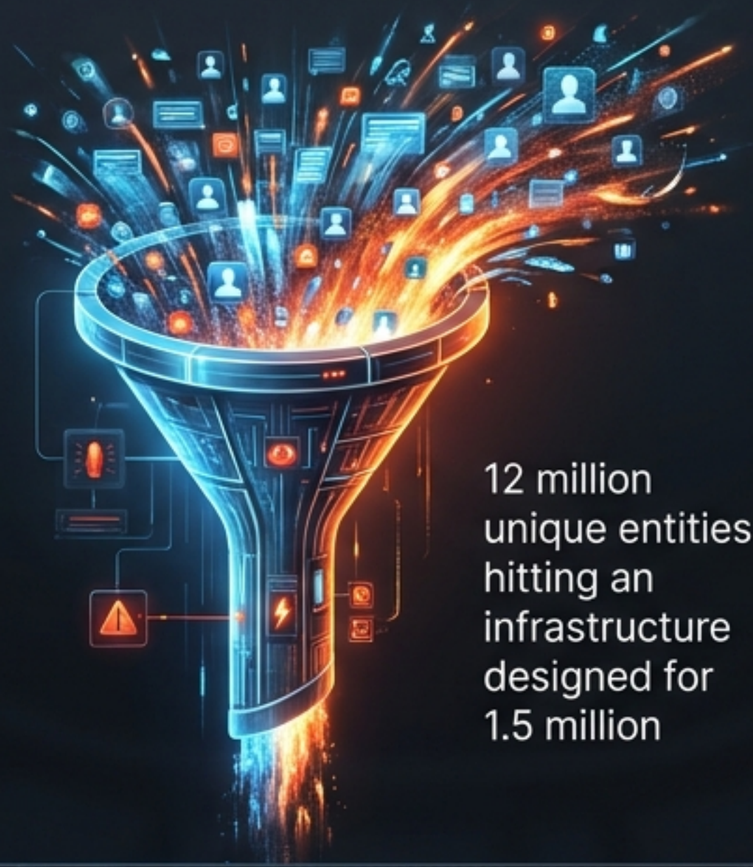
The Metrics

3.5 Million
Registered Verified Fans

14 Million
Total users who actually showed up

3.5 Billion
Total system requests (4x previous peak)

The Bottleneck



12 million unique entities hitting an infrastructure designed for 1.5 million

The Fallout

Frozen Queues ⚠

```
[10:15:23] SYSTEM ALERT: Queue processing halted. High latency detected.
```

Cart Abandonment Failures

```
[10:32:45] TRANSACTION ERROR: Checkout completion rate dropped to <10%. Timeout exceptions.
```

System Outages

```
[10:48:11] CRITICAL FAILURE: Multiple server nodes offline. Load balancer overwhelmed. 502 Bad Gateway.
```

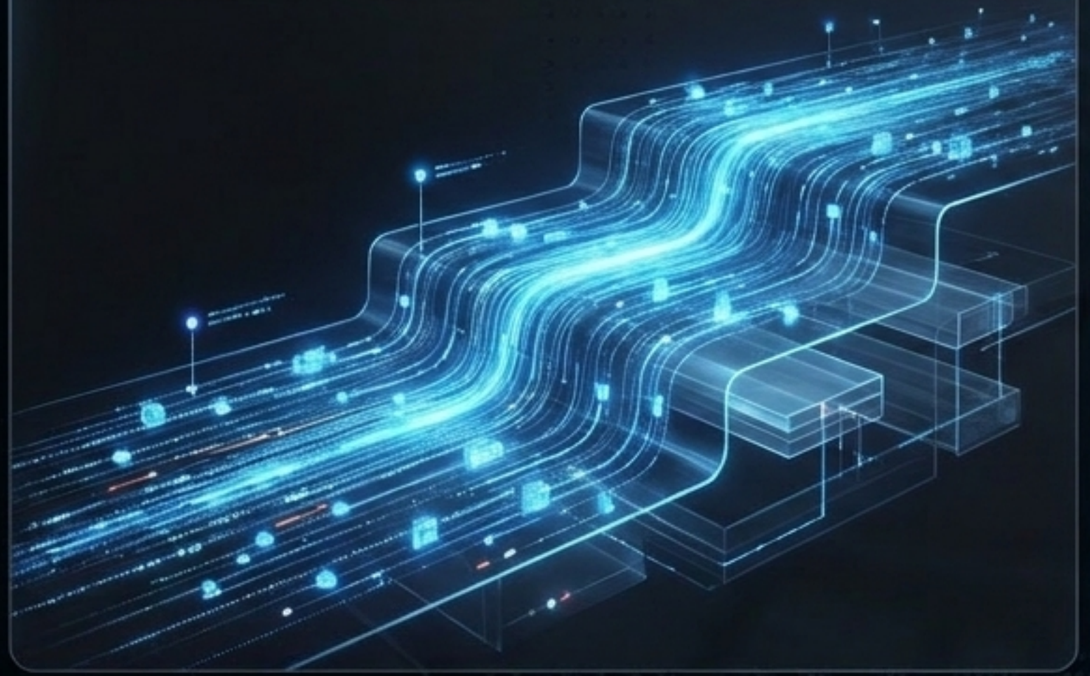
Congressional Hearings

```
[11:00:00] EXTERNAL EVENT: Legal and regulatory inquiry initiated. Public relations impact imminent.
```

Event ticketing demands a delicate balance of availability and consistency

The Read Path

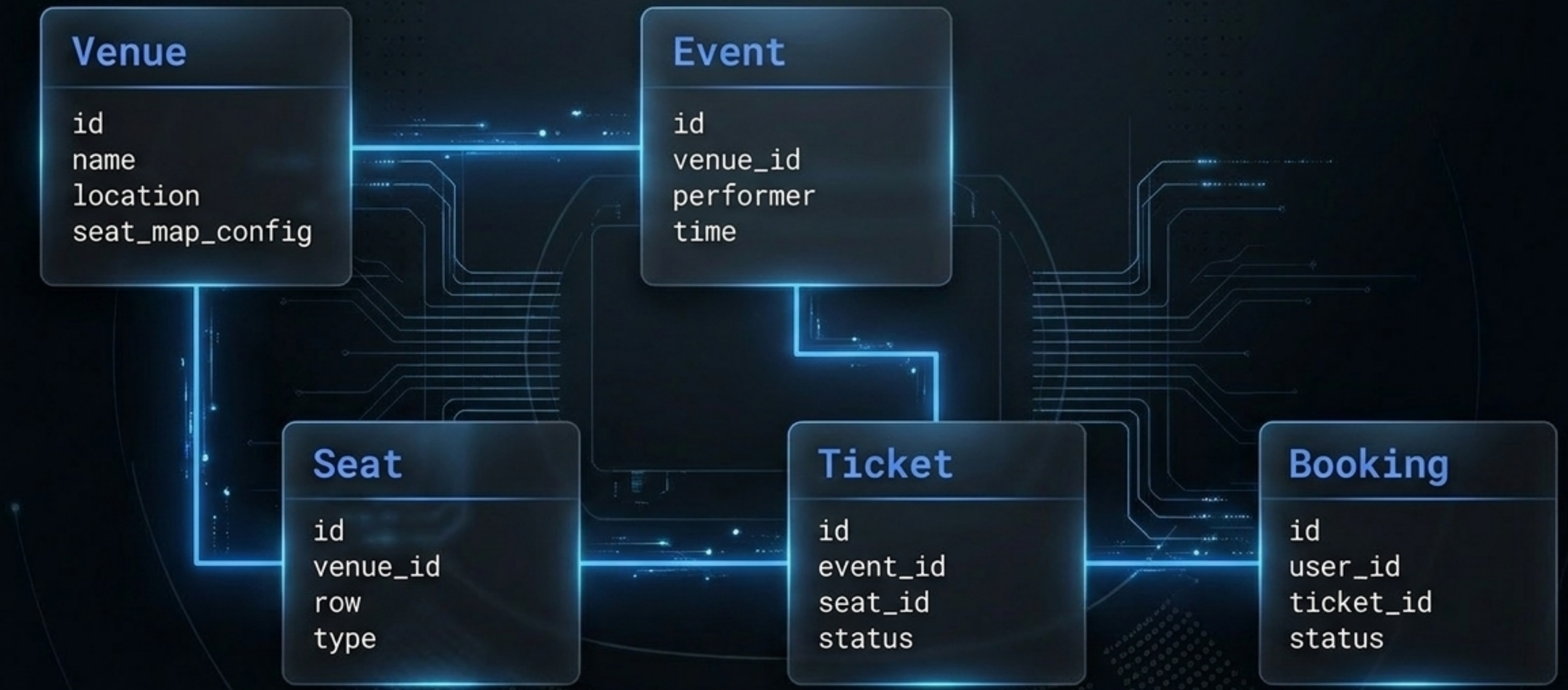
- **Goal:** High Availability (99.99% uptime)
- **Load:** `100,000+` views/sec
- **Ratio:** `100:1` Read-to-Write



The Write Path

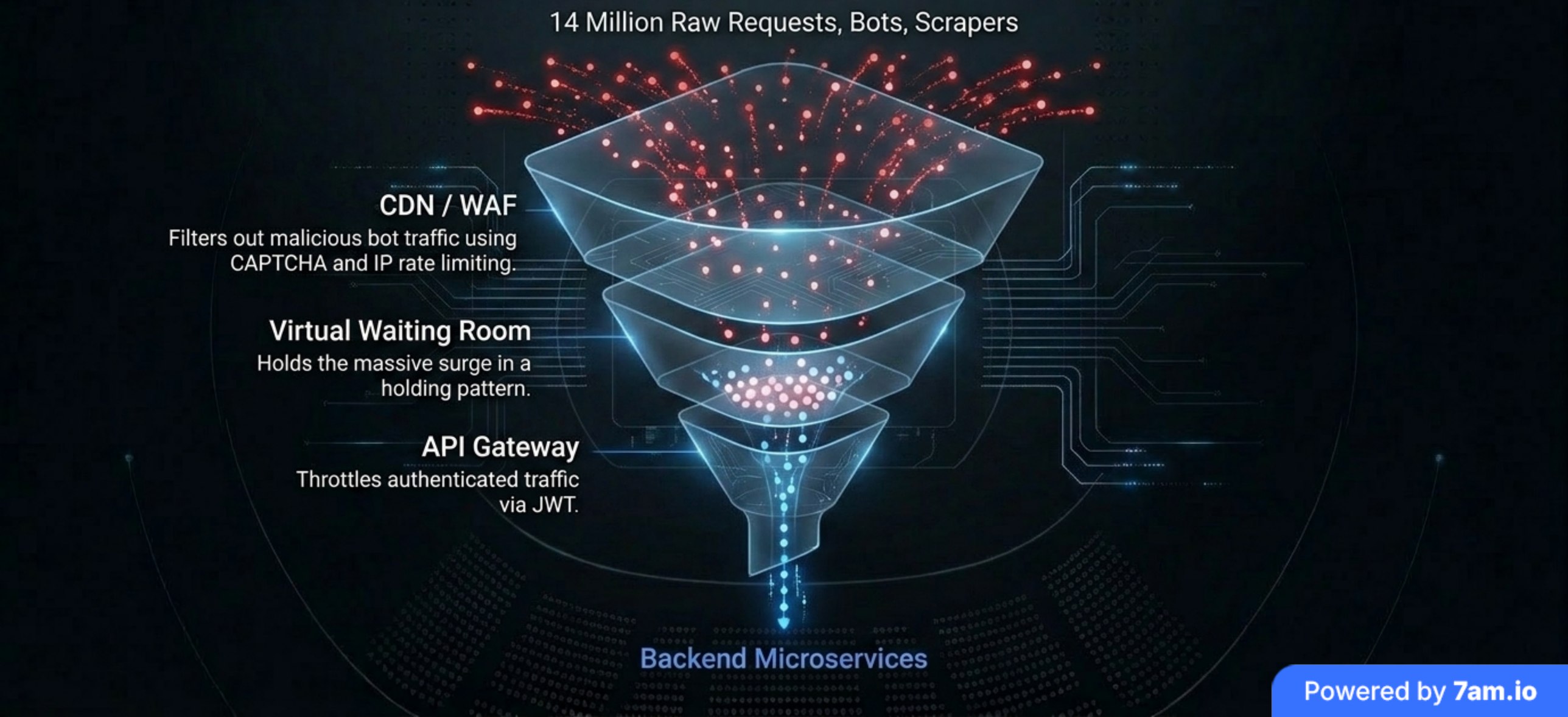
- **Goal:** Strong Consistency (Zero double-bookings)
- **Load:** `10,000` transactions/sec
- **Constraint:** Finite, immovable inventory

Relational data modeling grounds the core ticketing entities



RESTful interfaces decouple the browsing and purchasing pathways

Massive traffic surges require ruthless traffic shaping at the edge

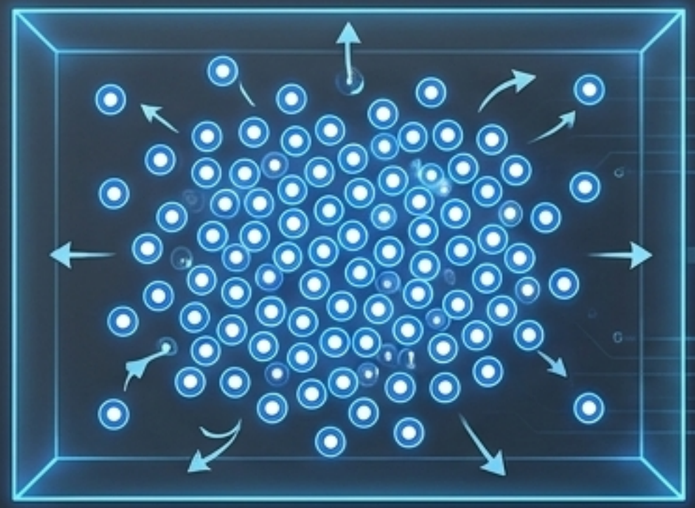


Comparing traffic control strategies reveals the necessity of virtual queues

Strategy	Pros	Cons
SSE Seat Map Updates	Great for moderate traffic.	Fails completely under extreme surges (seat map goes instantly black).
FIFO Queue (First-In-First-Out)	Prevents server crashes.	Rewards speed-bots and the fastest clickers; unfair to legitimate fans.
Randomized Pre-Queue (Queue-it Model)	Pools early arrivals and randomizes them when the sale begins. Absolute fairness.	Requires third-party edge integration.

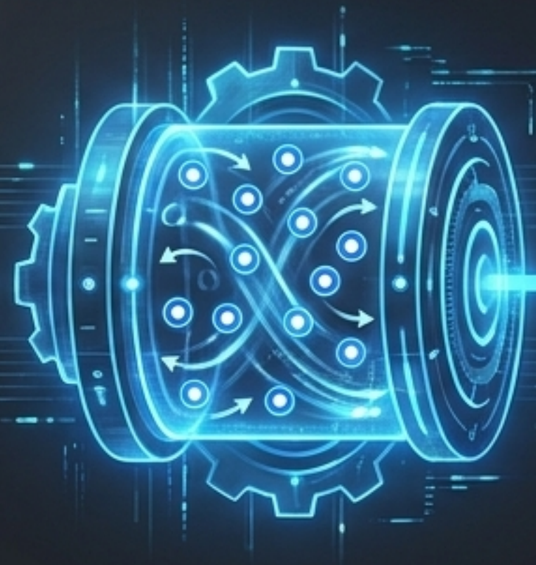
Randomized pre-queues neutralize bot advantages during high-demand drops.

T-Minus 30 Mins: The Waiting Room



A chaotic cloud of user nodes gathering in a holding area.

T-Zero: The Raffle

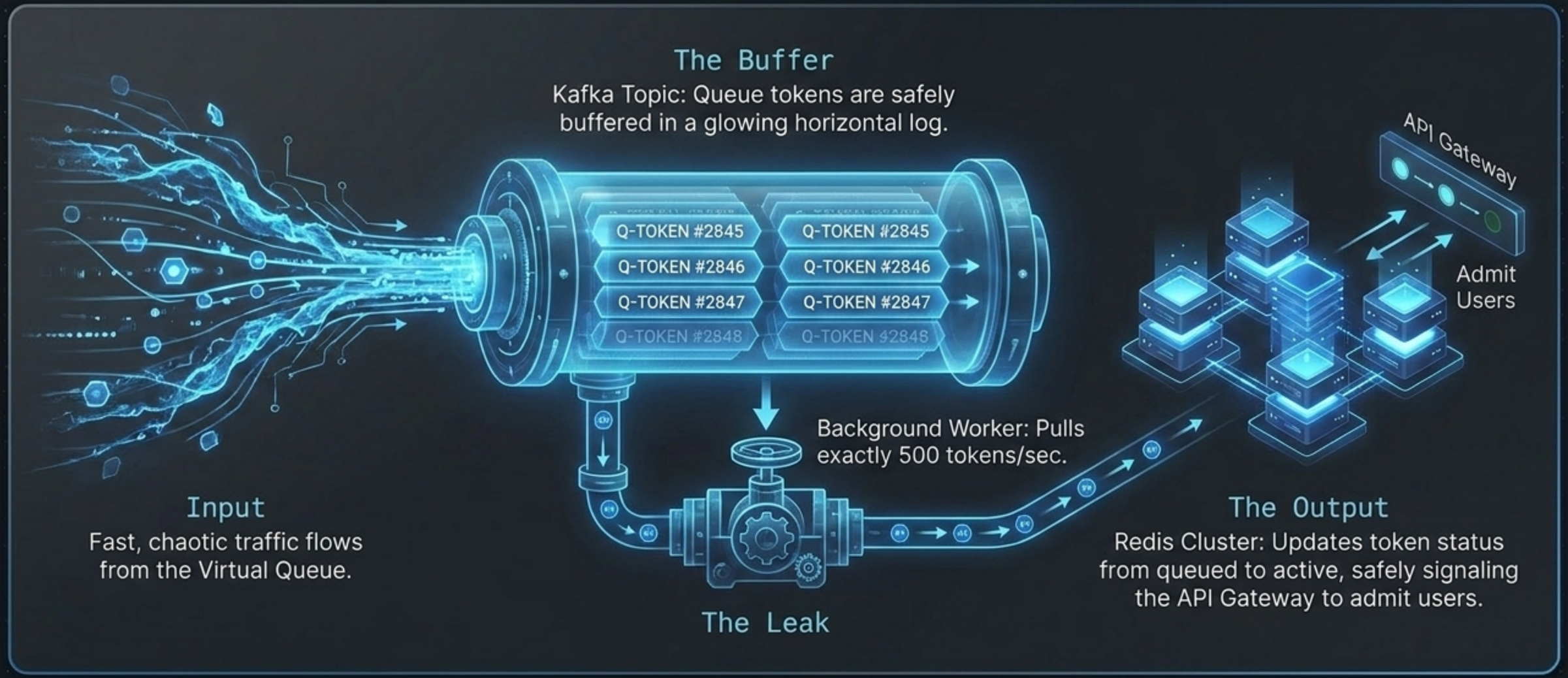


A randomization engine instantly shuffles the pool and assigns sequential queue numbers.

T-Plus 1 Min: The Flow

Late arrivals are placed sequentially at the back of the line (FIFO), ensuring absolute fairness and defeating speed-based scalping bots.

Kafka buffers requests into a controlled leak to protect backend services



MISSION CONTROL

Change Data Capture synchronizes the primary database with Elasticsearch.

Primary DB

PostgreSQL containing source-of-truth Event data

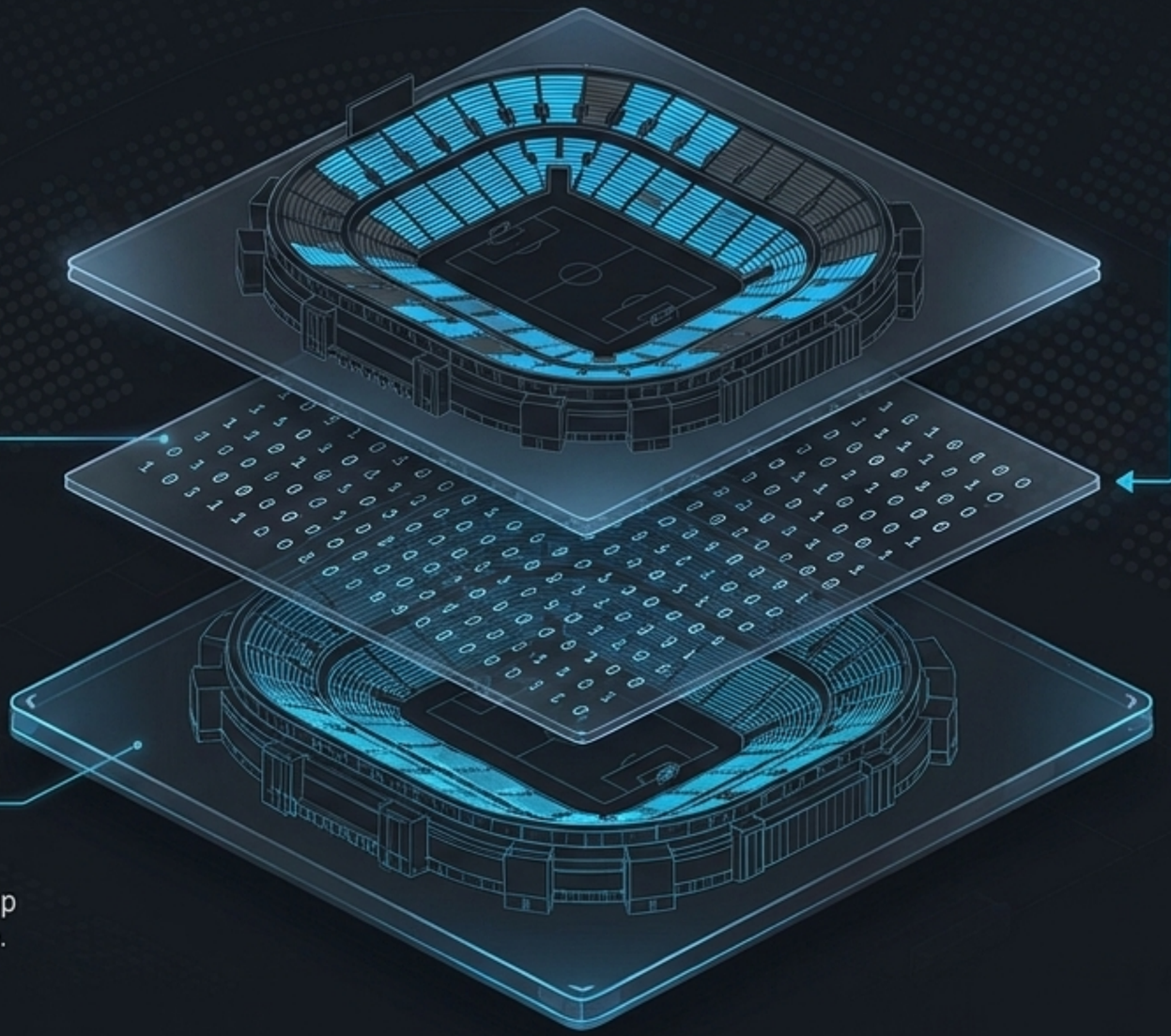
Change Data Capture (CDC)

Extracts changes from the Write-Ahead Log without locking primary database tables

Search DB

Elasticsearch cluster provides sub-500ms full-text search optimized for 100,000+ views/sec

Binary bitfields overlay static seat maps to eliminate database reads



Client Merge

0 paints the SVG seat electric blue (Available).
1 paints it dark grey (Taken).

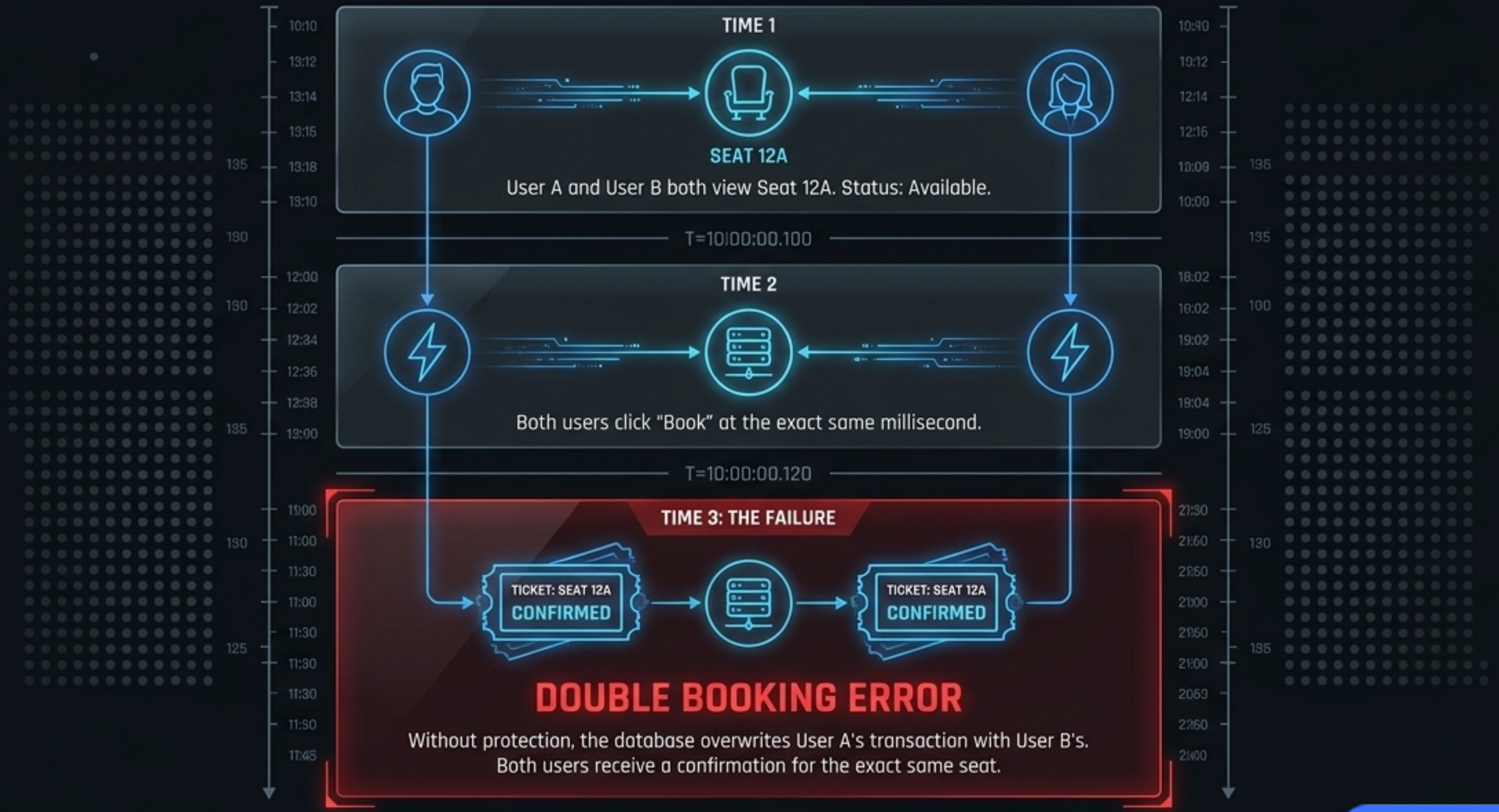
The Dynamic Data

Lightweight binary bitfield array fetched from a Redis in-memory cache.

The Static Asset

Heavy 50,000-seat SVG stadium map cached completely at the CDN edge. Never hits backend servers.

Simultaneous booking requests create highly contested race conditions



Evaluating concurrency strategies against strict double-booking constraints

Cron Job & DB Status

Status: Reserved + Expiration Timestamp

Verdict: Slow unlocking, high DB load, poor UX if cron fails.

Redis Distributed Lock

In-memory lock with TTL

Verdict: Extremely fast, self-cleaning.

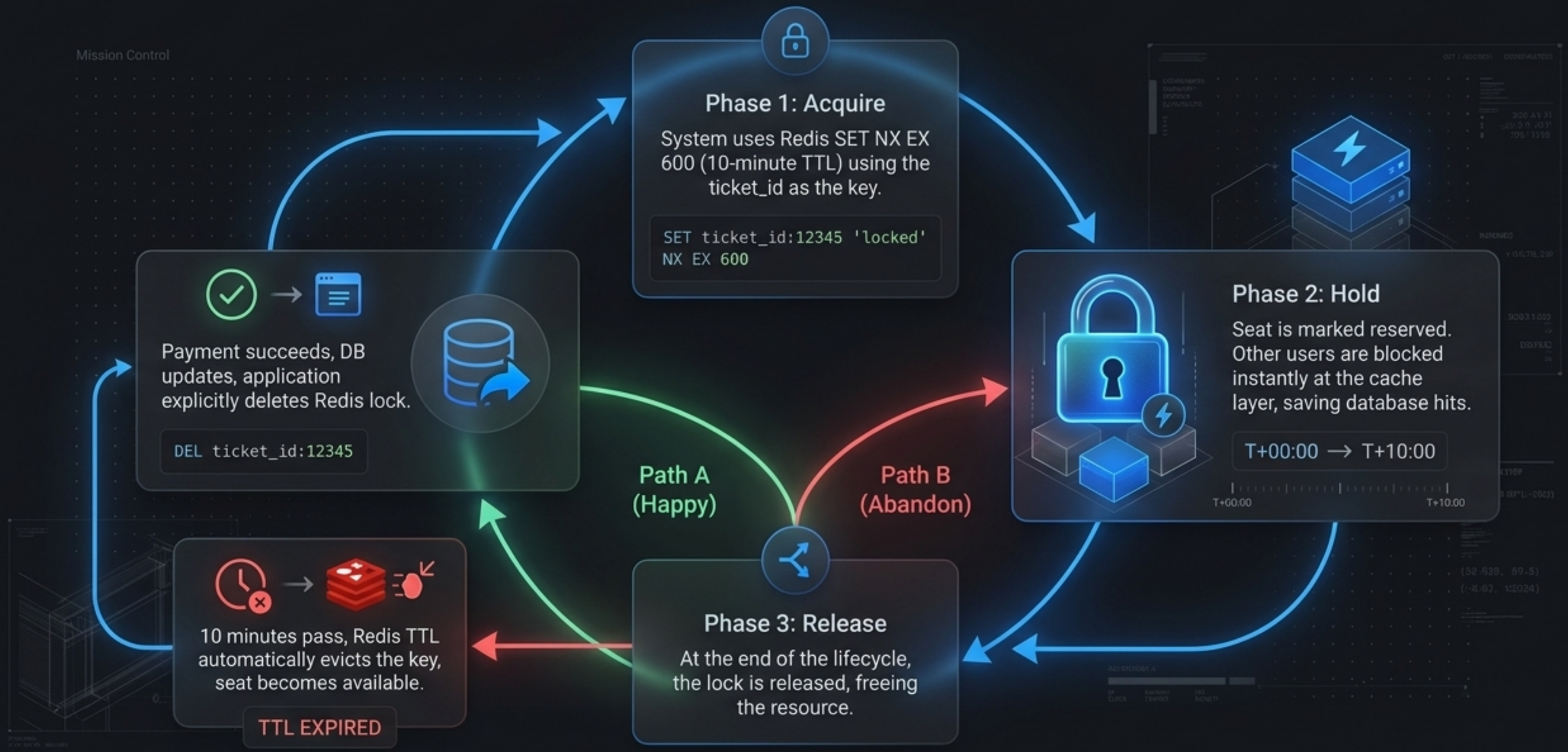
Risk: Data loss of reservations if cache restarts.

MySQL SKIP LOCKED

Database-level row locking

Verdict: Atomic, permanent, avoids contention. **The elite standard.**

Distributed locks guarantee temporary exclusivity via TTL expirations



Shopify's SKIP LOCKED mechanism bypasses row contention entirely

The Setup: One row per unit of inventory, bounded to a pool of 1,000 available units to prevent slow scans.

The Action

Transaction A locks Row 1 (FOR UPDATE).

Transaction B

The Magic

Transaction B jumps over to instantly lock Row 2.

Result: MySQL completely ignores locked rows, eliminating waiting and database contention.

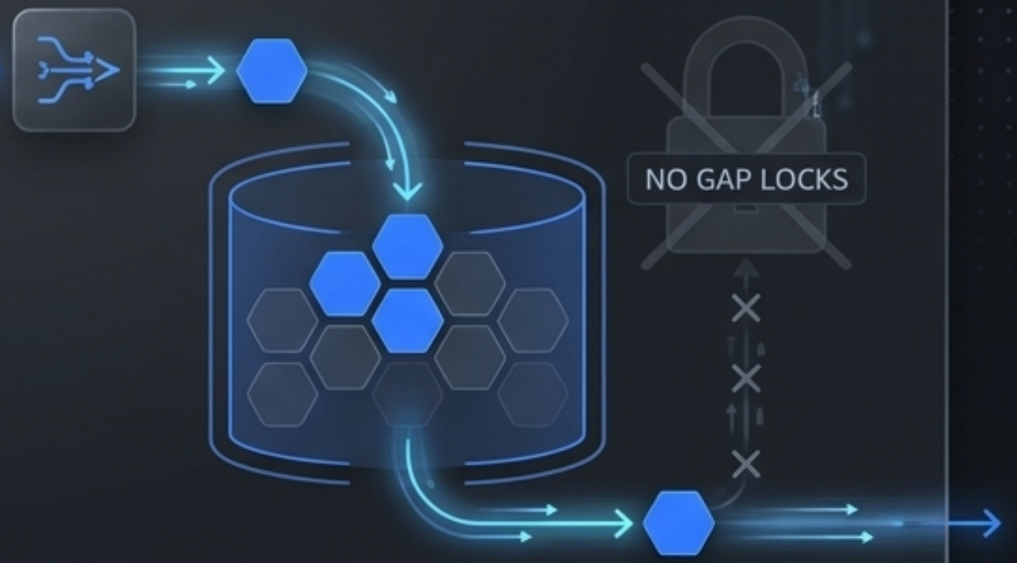
Composite primary keys and READ COMMITTED isolation maximize throughput

Composite Keys

```
shop_id, inventory_item_id, inventory_group_id, id
```

Changing from an auto-increment ID to a composite key ensures a single lock per row.

READ COMMITTED



Changing the default isolation level prevents InnoDB from taking gap (supremum) locks, allowing background replenishment without deadlocks.

The two-step booking flow separates reservations from finalized payments

Idempotency keys prevent duplicate charges during network failures

Asynchronous delay queues automatically sweep abandoned reservations

TRIGGER

Booking is created.

Task pushed to a Delay Queue scheduled for exactly 10 minutes in the future.

LISTENER

10 minutes

After 10 minutes, a Cleanup Worker wakes up and pulls the `booking_id`.

RECONCILIATION

Checks primary database. If status is still pending (user abandoned cart):

```
>_
1. Reverts DB status
   to available.
2. Flips the Redis
   bitfield back to 0.
3. CDN pulls the fresh
   state.
```

The Modern Mission Control: A unified architecture for massive-scale ticketing

Scale is achieved through intentional constraints

Pillar 1: Shape the Traffic

Virtual waiting rooms and Kafka buffers protect the backend.

Pillar 2: Decouple the Reads

CDN edge caching and Elasticsearch eliminate database query load.

Pillar 3: Lock the Writes

Distributed Redis locks and MySQL `SKIP LOCKED` prevent race conditions.