Step 1: The Contract (Requirements)

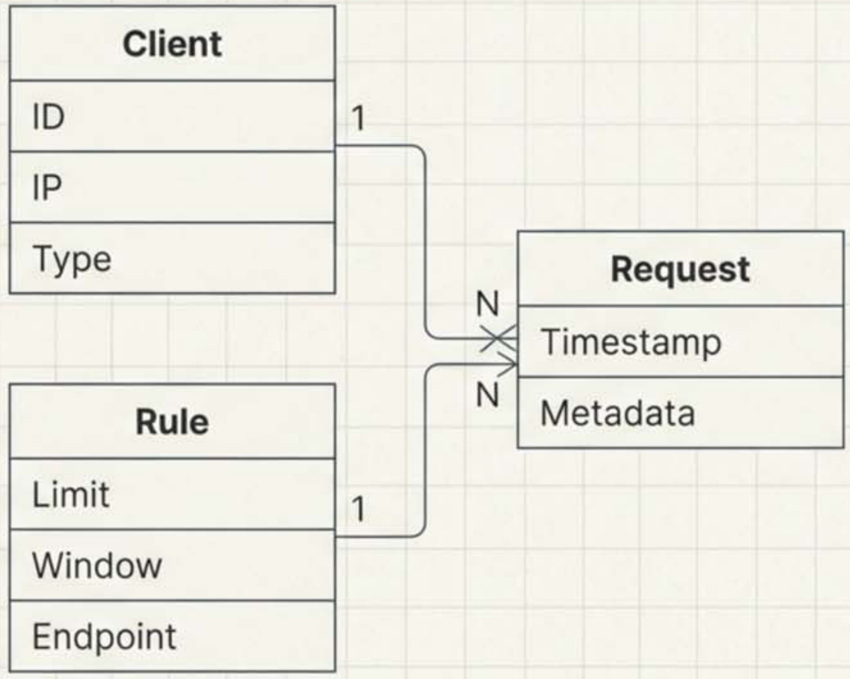
Functional Requirements

- ✓ Identify Clients: `User ID`, `IP Address`, or `API Key`
- ✓ Configurable Rules: `e.g., 100 requests / minute`
- ✓ Feedback: Return `HTTP 429` + Headers (Remaining, Reset Time)

Non-Functional Requirements

- Scale: **1M Requests Per Second (RPS)** / 10M DAU
- Latency: **< 5ms** per check
- Availability: High availability > Strong consistency

Step 2: The Blueprint

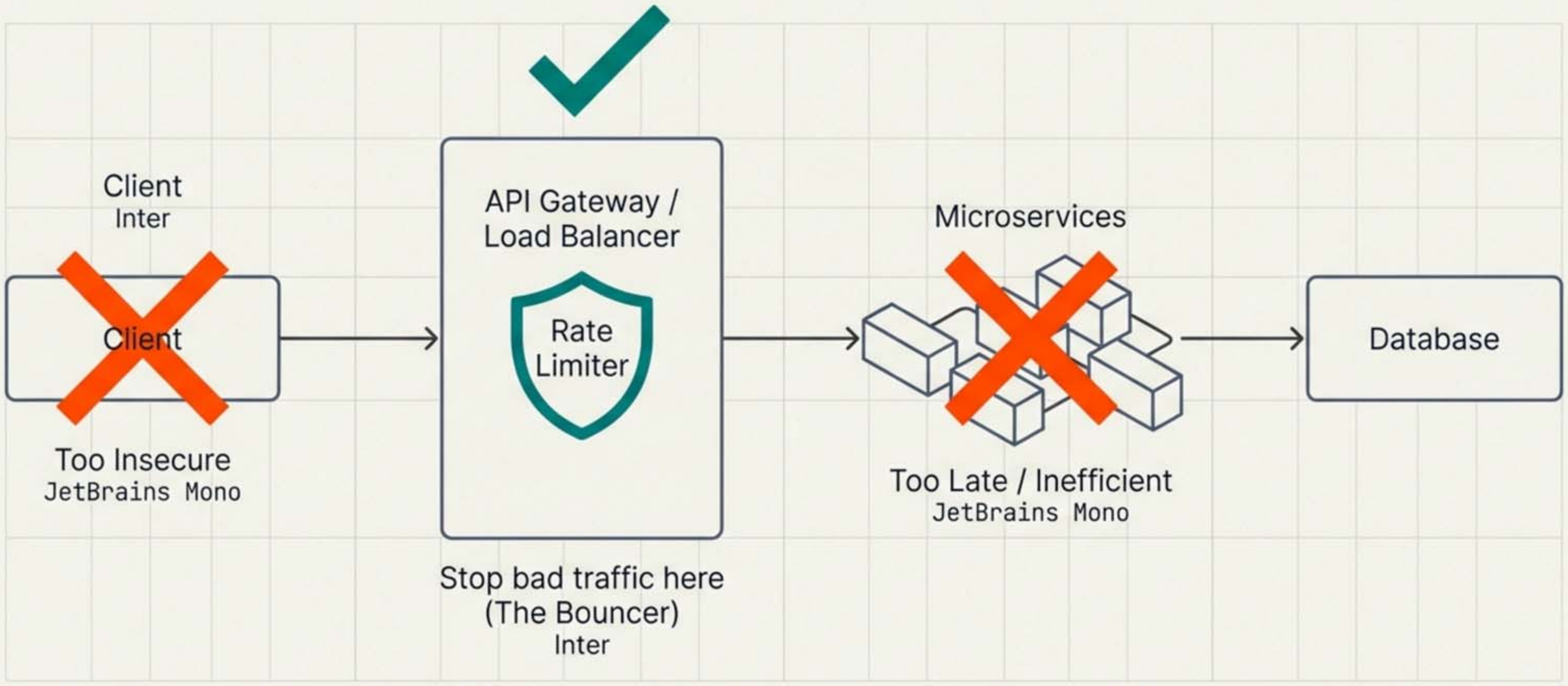


```java
interface RateLimiter {
  boolean isRequestAllowed(String clientId, String ruleId);

  // Returns:
  // - allowed: true/false
  // - remaining: int
  // - resetTime: timestamp
}
```

NotebookLM

Strategic Placement: The ‘Bouncer’ Analogy



NotebookLM

Identification Strategy: Who is Knocking?

User ID

Best for Authenticated Users. Enables specific user limits.

IP Address

Fallback for Anonymous/DDoS. **Risk:** Shared IPs (NAT).

API Key

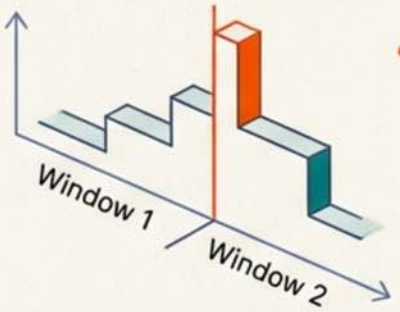
Standard for B2B / Developer Tools.

The Senior Answer

Use a Hybrid Approach. Authenticated users get higher limits; fallback to IP limits for unauthenticated traffic.

Algorithm Selection: Trade-offs

Fixed Window



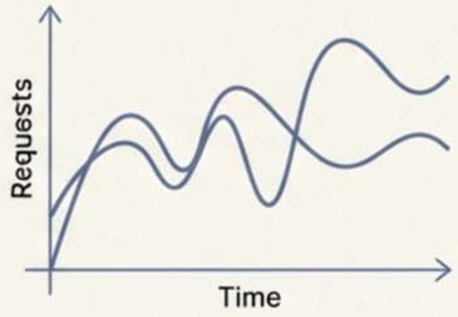
⚠ Boundary Effect (Burst Issue)

Sliding Log

```
10:00:01.123
10:00:01.456
10:00:02.789
10:00:02.901
10:00:03.234
...
```

⚠ High Memory Cost (Not for 1M RPS)

Sliding Window

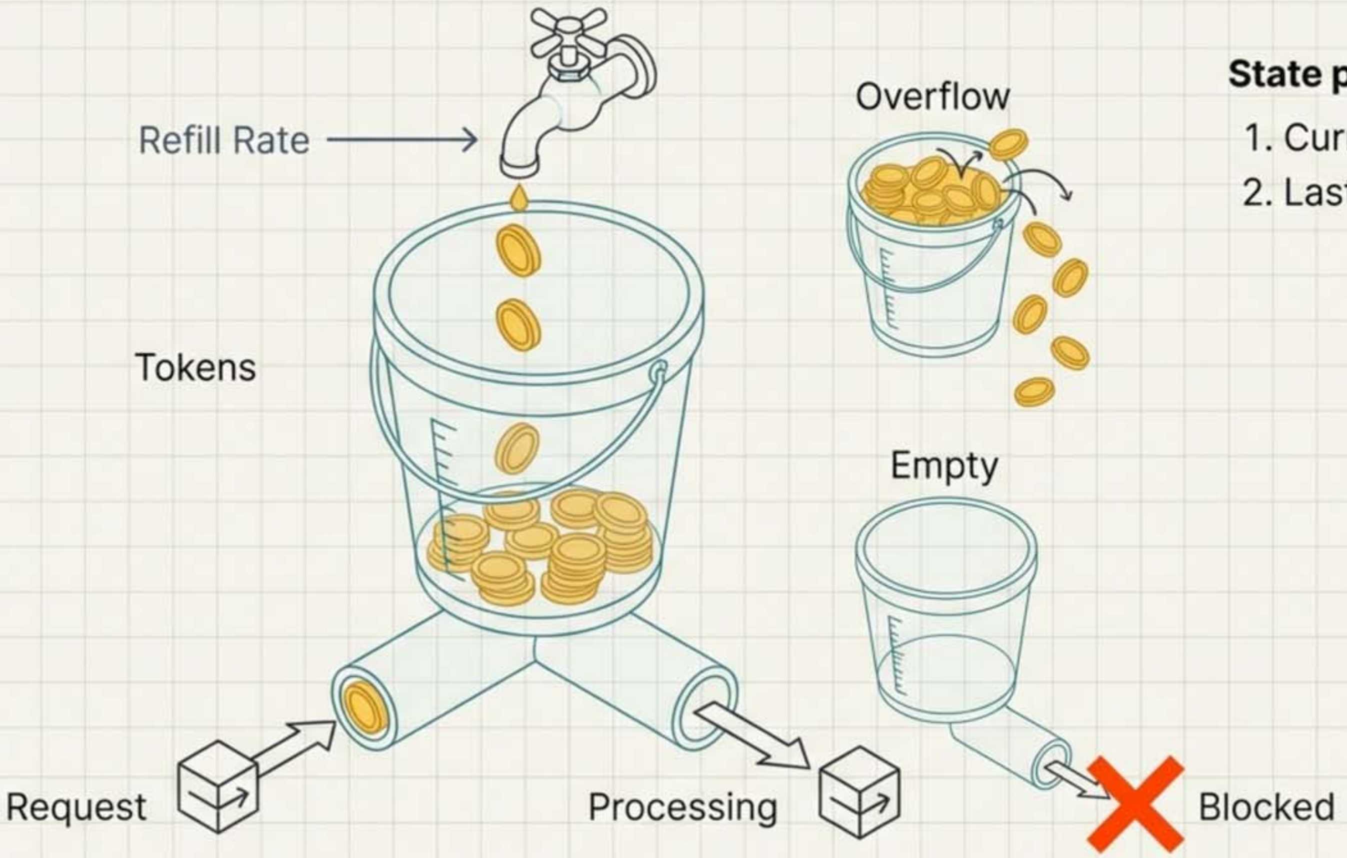


Good Accuracy, Complex Math

Token Bucket

✓ **The Winner:** Memory Efficient & Handles Bursts

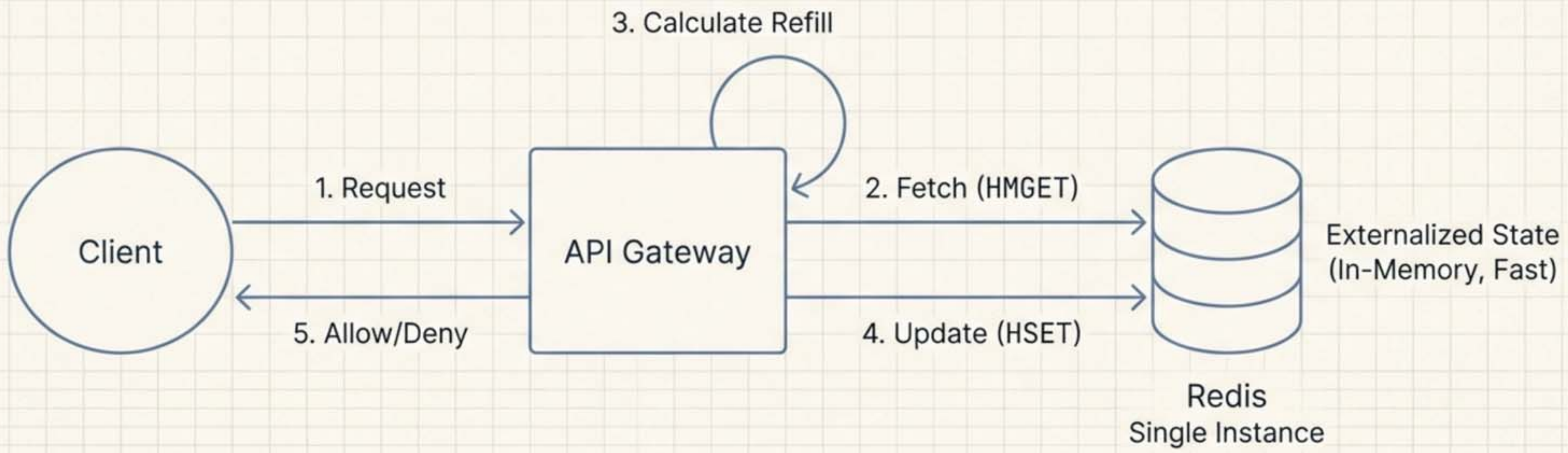
The Chosen Algorithm: Token Bucket



State per User

1. CurrentTokens (Integer)
2. LastRefillTimestamp (Long)

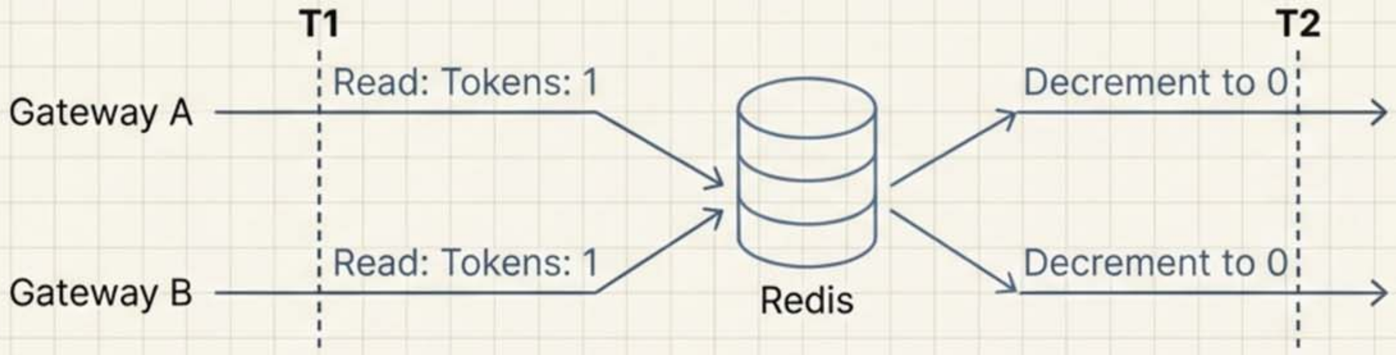
High-Level Architecture (The MVP)



NotebookLM

Critical Deep Dive: The Race Condition

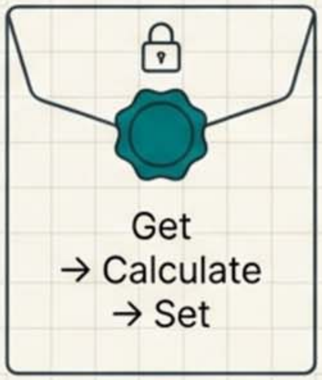
The Problem



Result: Both requests pass. Limit Exceeded.

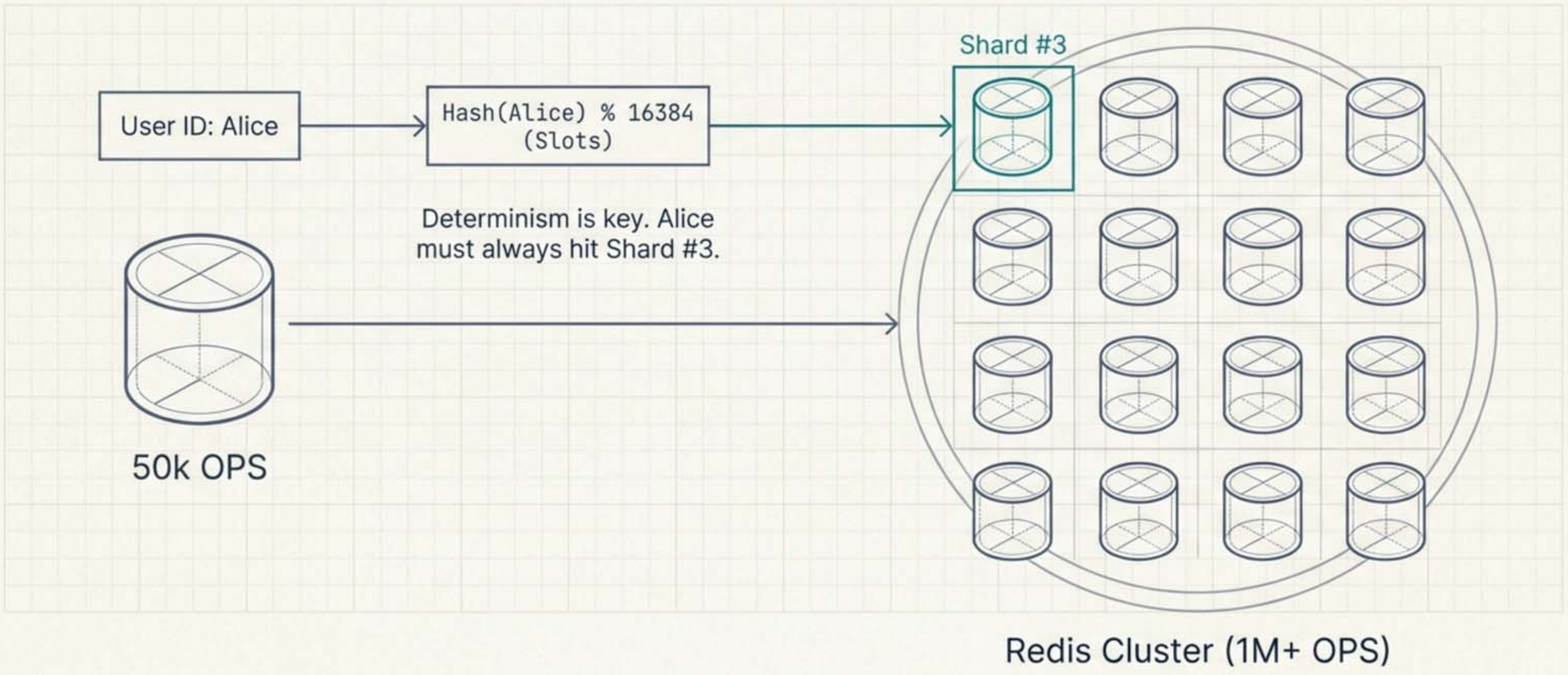
The Solution

Redis Lua Script



Atomic Execution.
No other operation can interrupt the script.

Scaling to 1 Million RPS: Sharding

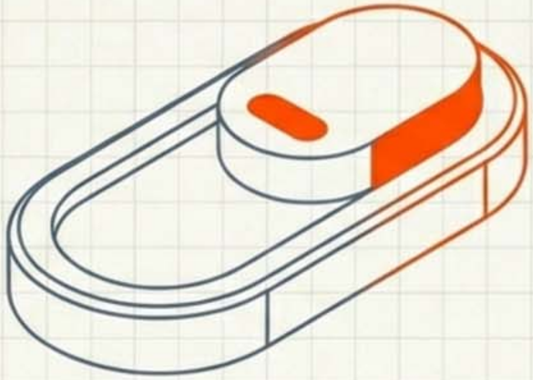


NotebookLM

Availability Strategy: Fail Open vs. Closed

Fail Open

User Happy,
Backend Risks Overload.



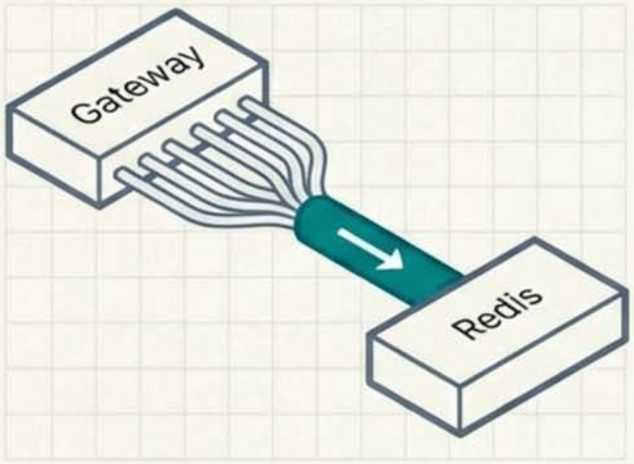
Fail Closed

User Blocked,
Backend Protected.

Staff Level
The Advanced Solution: Degraded Mode / Circuit Breaker. If Redis fails, fall back to local memory (allow 20% traffic) to prevent total outage.

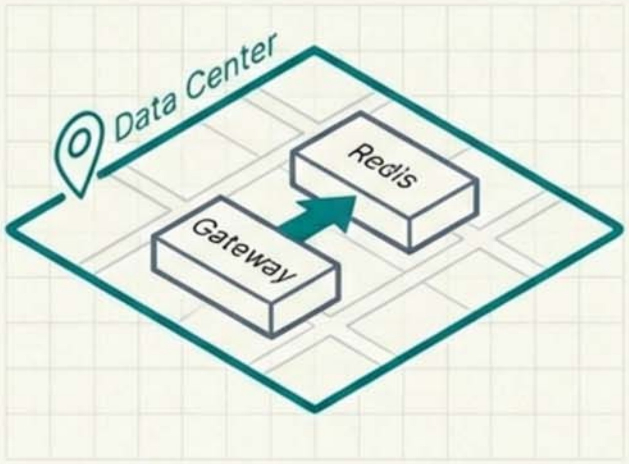
Optimizing for < 5ms Latency

Connection Pooling



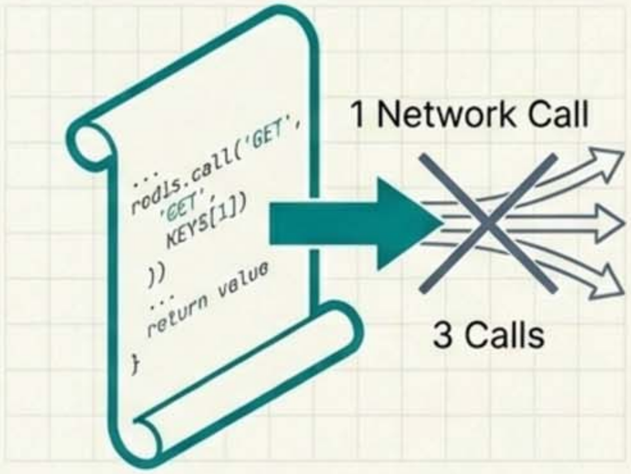
Reuse TCP connections.
Avoid handshake overhead.

Geo-Proximity



Colocate Cache
& Gateway.

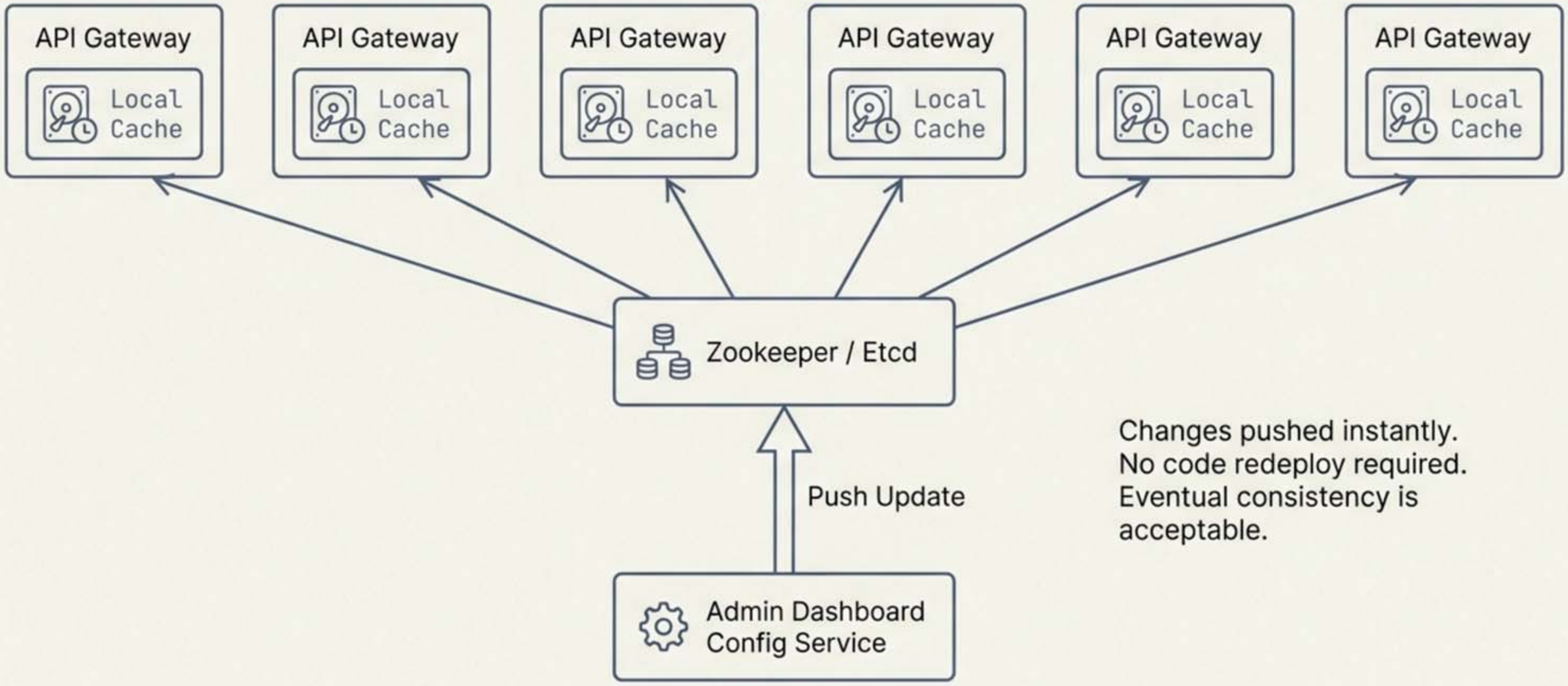
Lua Scripting



```
...
rodis.call('GET',
  'GET',
  KEYS[1])
 ))
 ...
 return value
}
```

1 Network Call
instead of 3.

Dynamic Configuration



Changes pushed instantly.
No code redeploy required.
Eventual consistency is acceptable.