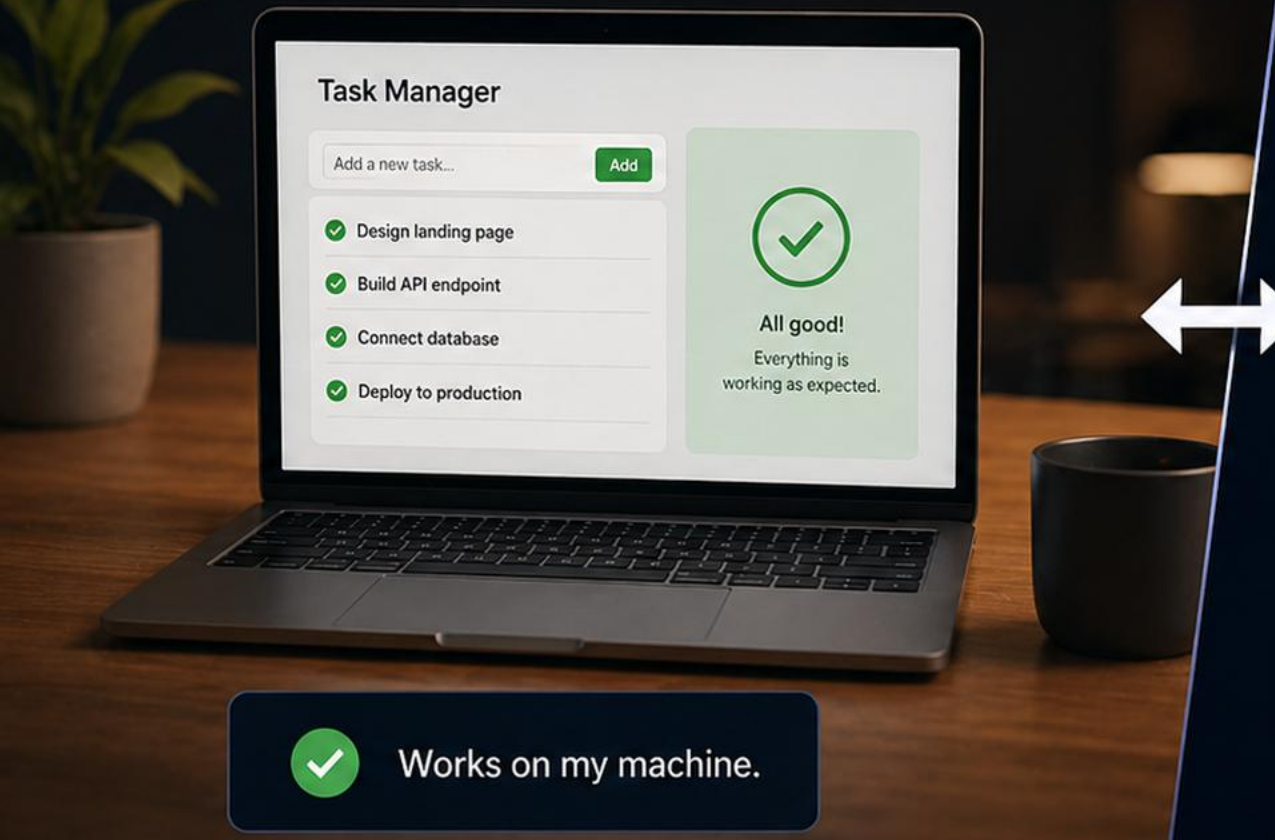
AI Built It. The Server Died.

A practical scaling survival guide for vibe coders

Working Is Not The Same As Surviving

A demo proves behavior. A service proves endurance.

demo



Works on my machine.

traffic

LATENCY (P95)
2.48s

COST (TODAY)
$482.71

ERROR RATE
14.7%

RECENT ERRORS

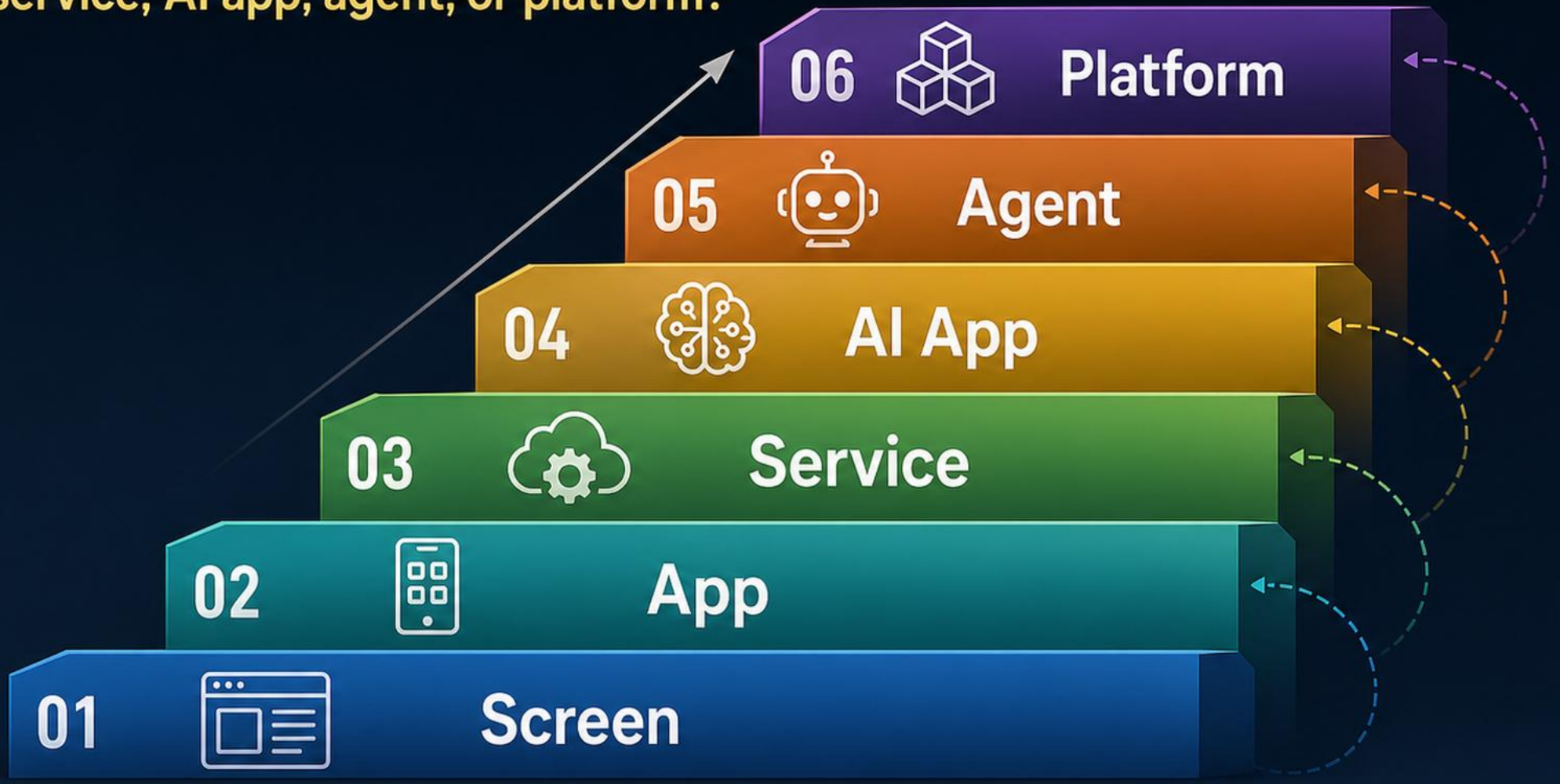
14:23:51	500	Internal Server Error	/api/orders
14:23:48	504	Gateway Timeout	/api/generate
14:23:46	429	Too Many Requests	/api/ai/chat
14:23:42	503	Service Unavailable	/api/payments
14:23:39	500	Internal Server Error	/api/orders

The New Starting Line

AI lowered the barrier to building, not the cost of operating.

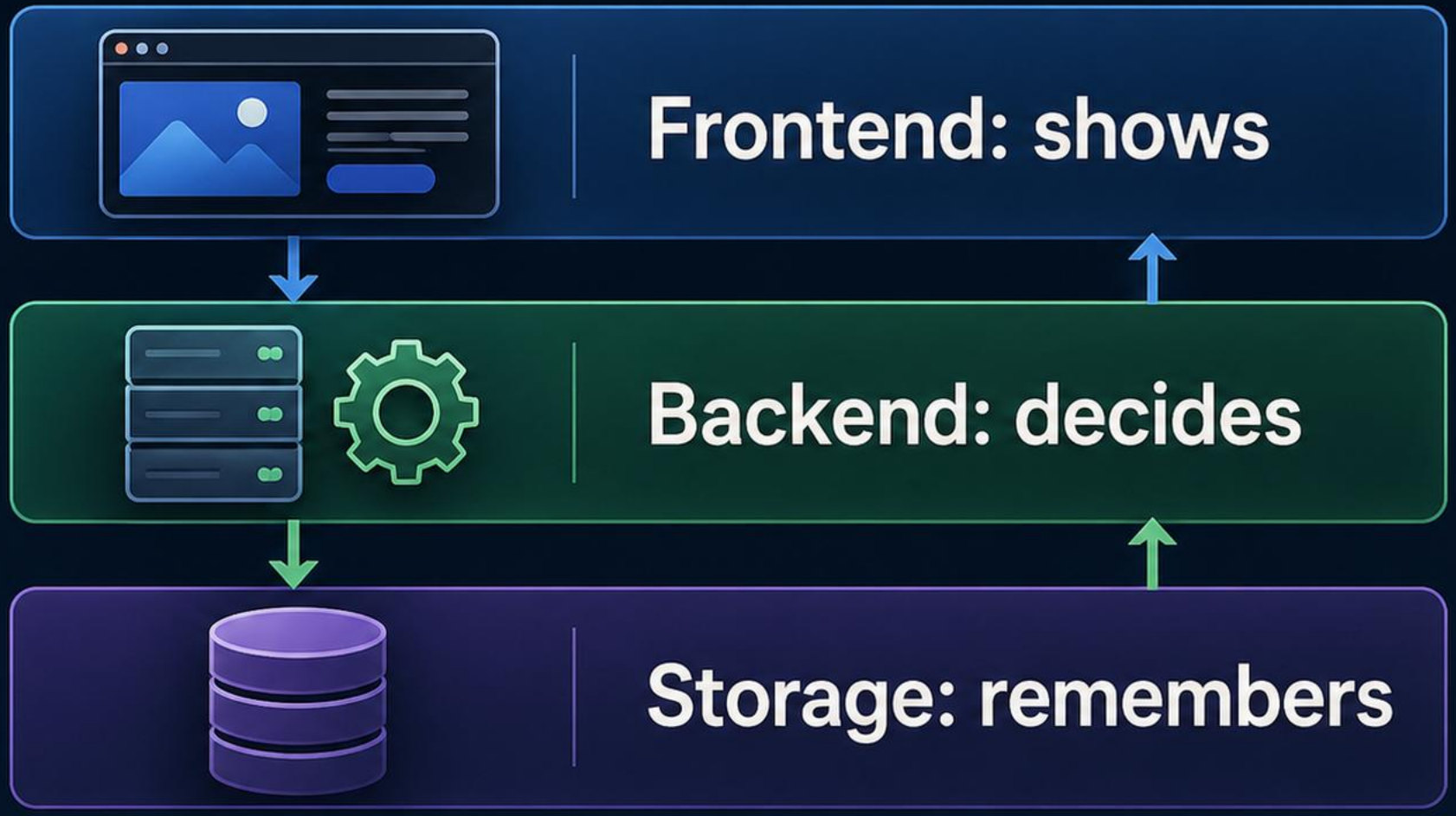
What Did You Actually Build?

Screen, app, service, AI app, agent, or platform?



Three Layers You Must See

Who shows? Who decides? Who remembers?

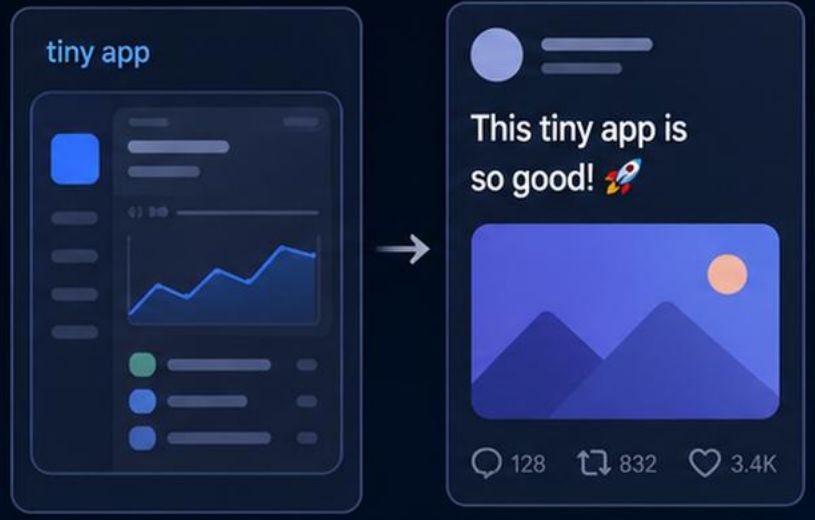


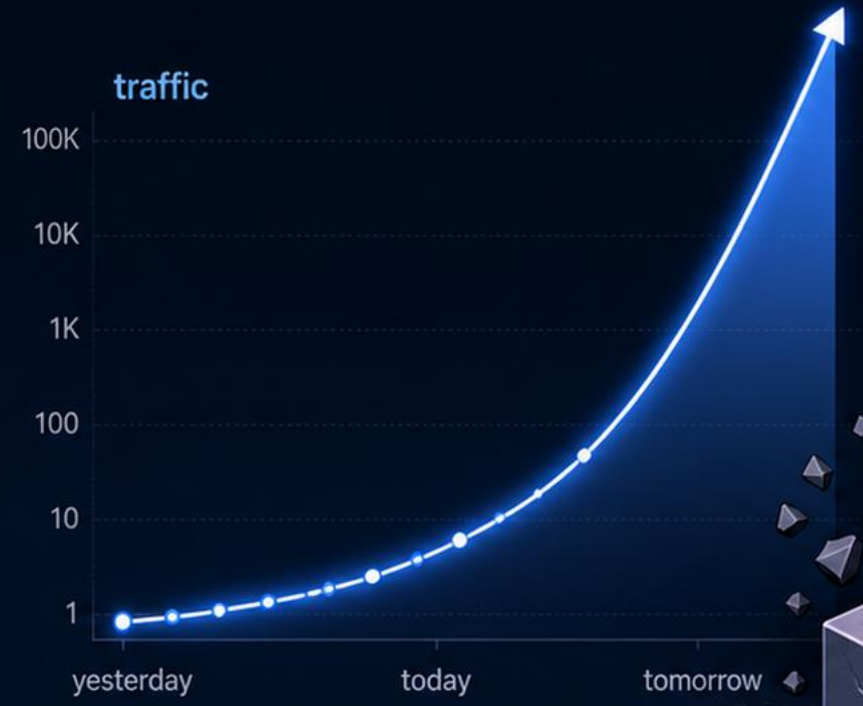
One Click Is Many Events

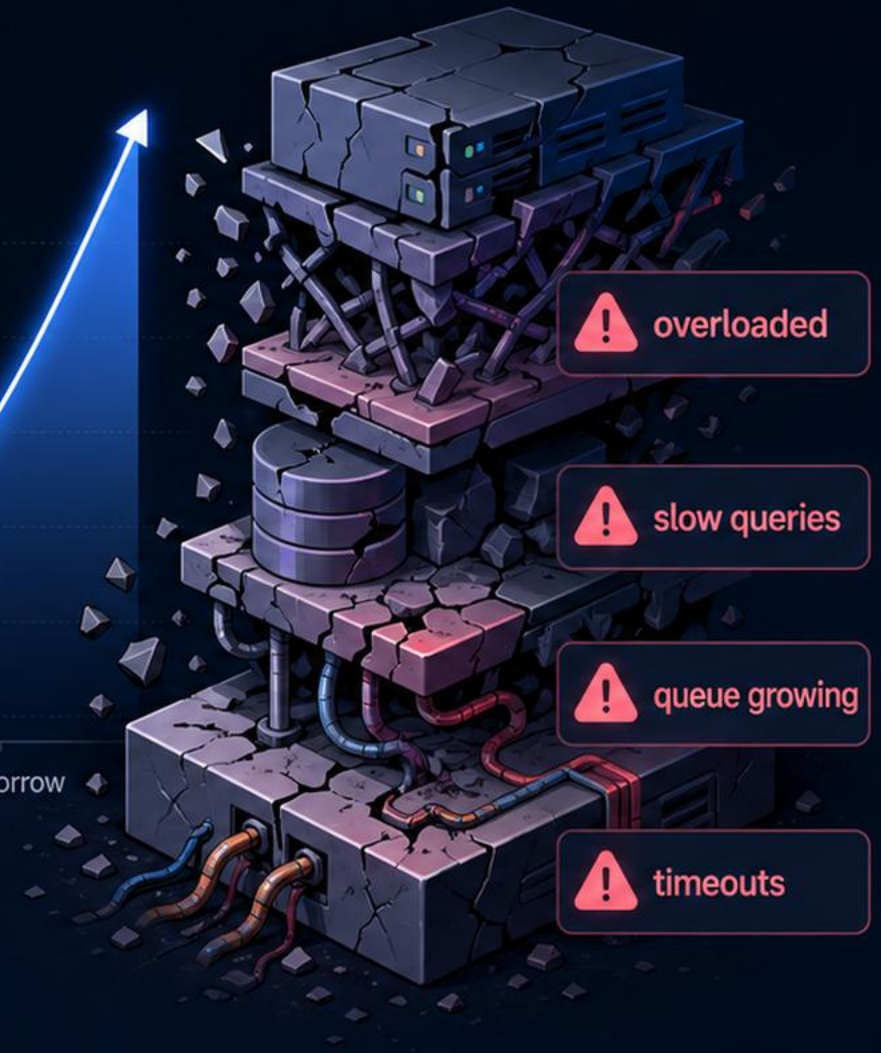
A button press crosses the whole system.

Small Success Reveals Hidden Debt

Yesterday was fine because yesterday was small.







Small load
everything looks fine

Real load
hidden debt appears

Slow Is Different From Down

Latency, limits, outages, and bad deploys need different responses.

Slow

Latency

Database Query

```
SELECT * FROM orders
WHERE user_id = ?
ORDER BY created_at DESC
LIMIT 50;
```

Duration	2.48 s
Rows Returned	50
Scanned Rows	1.2M
Query Plan	Using filesort

Service is slow.
Requests are completing.

Down

Status

500
SERVICE UNAVAILABLE

Error Rate

GET	/api/orders	500
GET	/api/payments	500
GET	/api/users	503
GET	/api/checkout	500
GET	/api/inventory	503

Limits

Connections	100 / 100

Service is down or blocked.
Requests are failing.

Dashboards Are Early Warning Systems

Read metrics before users read failure.

You can't fix what you don't see. Monitor early. Act early.

Healthy Watch Alert Info

The Frontend Is The First Experience

React, Vite, Next.js, CSR, SSR, and SEO are user-facing tradeoffs.

COMPONENTS

- Header
- Hero
- Features
- Pricing
- Contact
- +

SERVER-RENDERED HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Awesome App</title>
  <meta name="description"
   content="Build faster.
   Scale smarter." />
</head>
<body>
  <header>...</header>
  <main>
    <h1>Build faster.</h1>
    <p>Scale smarter.</p>
    <section>...</section>
  </main>
  <footer>...</footer>
</body>
</html>
```

Rendered on the server
Fast first byte, SEO friendly

CLIENT HYDRATION

JavaScript loads
and makes it interactive
CSR or selective hydration

SEARCH INDEX

Awesome App – Build faster. Scale smarter.
https://awesome.app
Build faster. Scale smarter. The platform for ambitious teams.

Crawled and indexed
Visibility drives traffic

Fast to interactive
keeps users happy

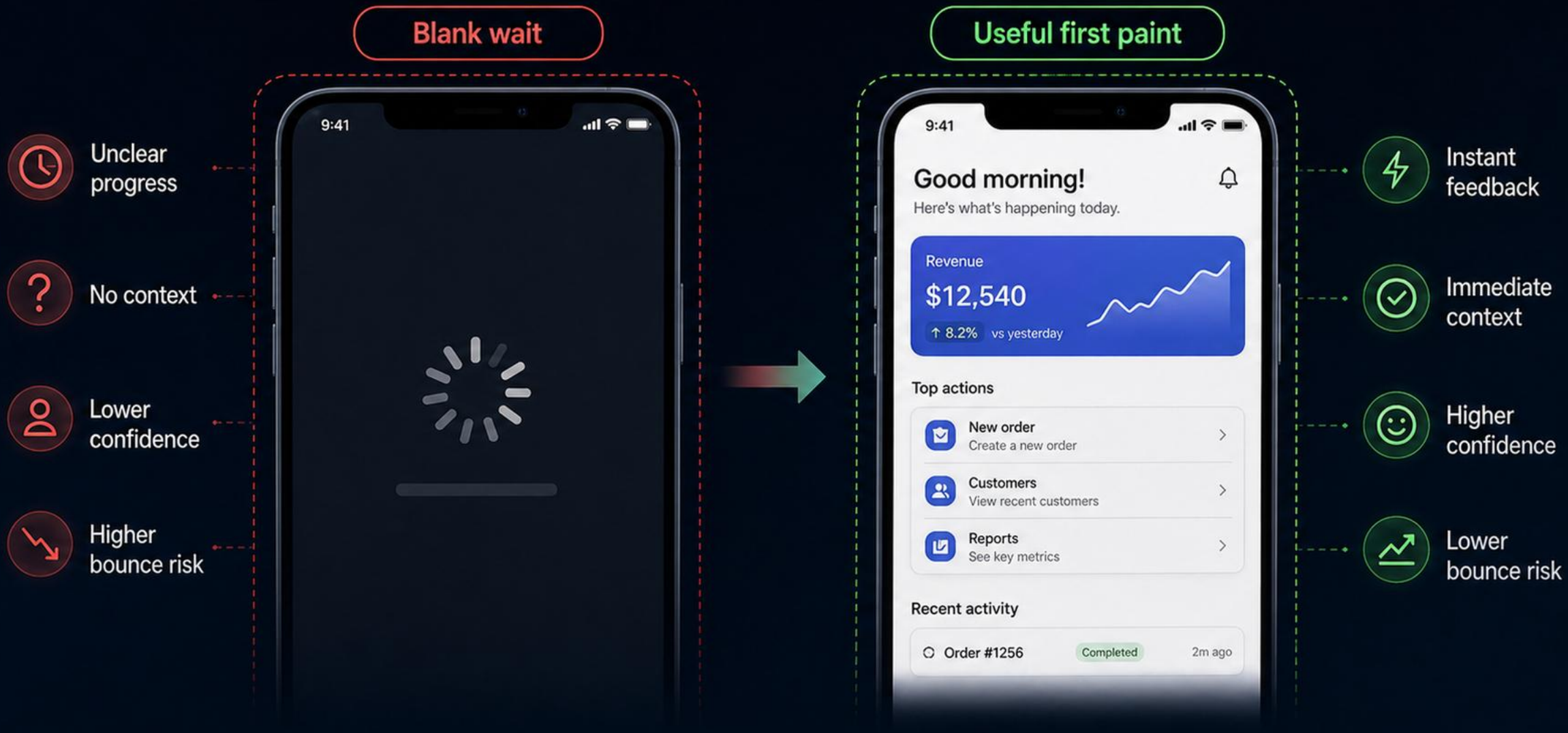
Semantic HTML
gets found

Good performance
earns trust

Every choice is a
user-facing tradeoff

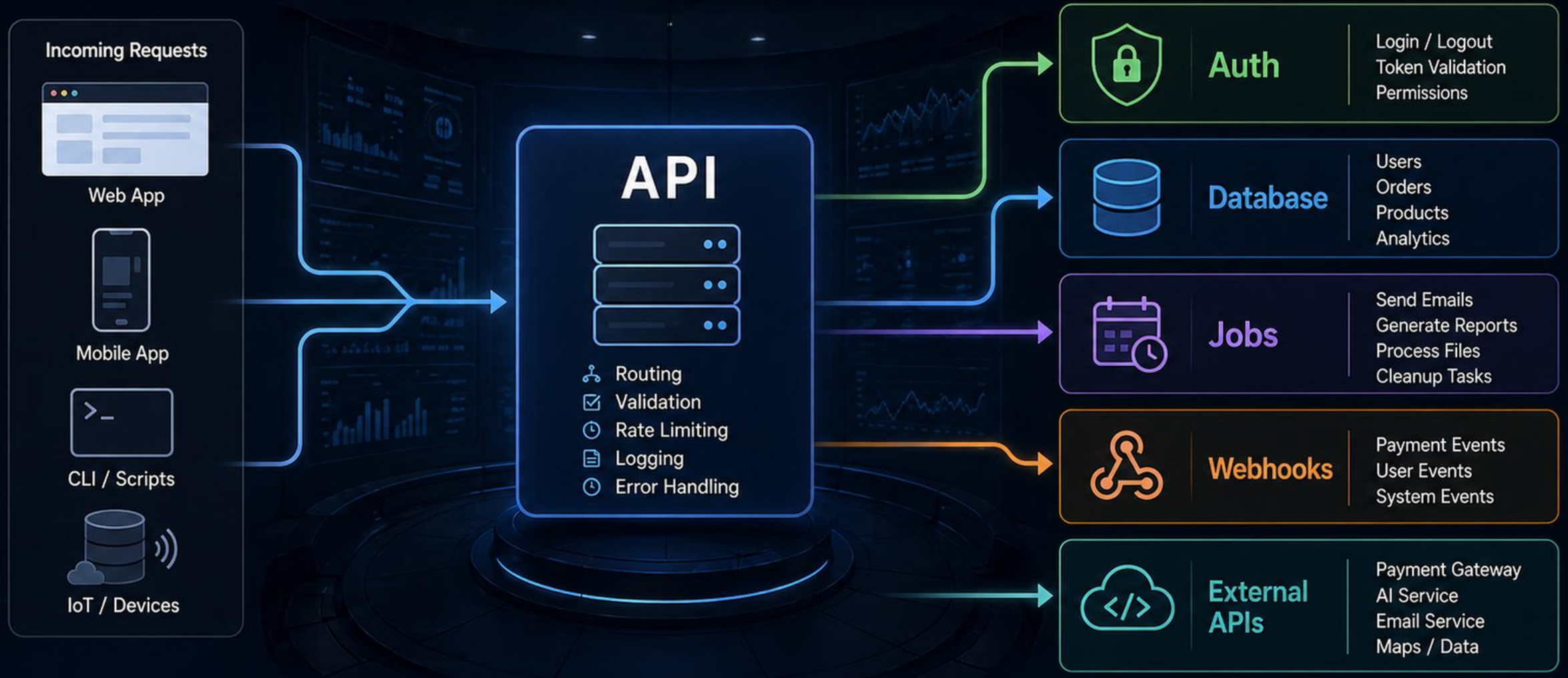
First Screen Speed Is A Product Decision

What appears first shapes trust.



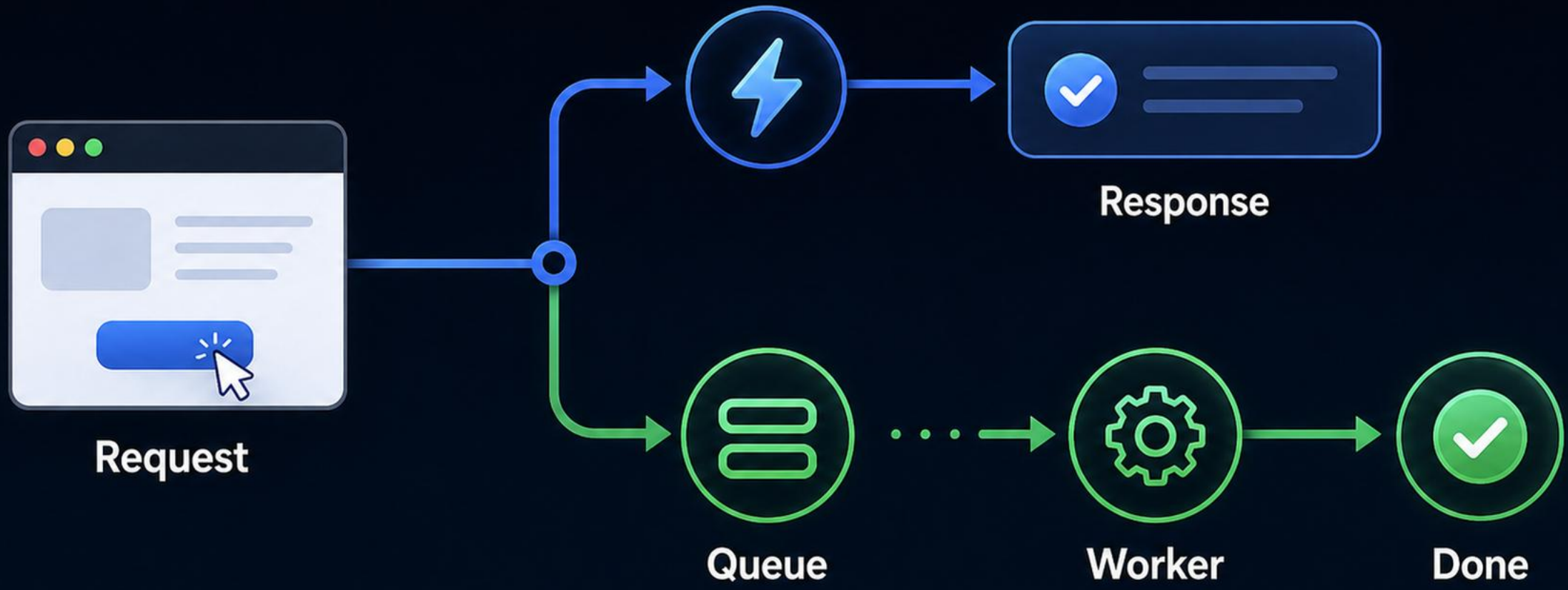
The Backend Is Where Rules Live

APIs, webhooks, auth, jobs, and external services define behavior.



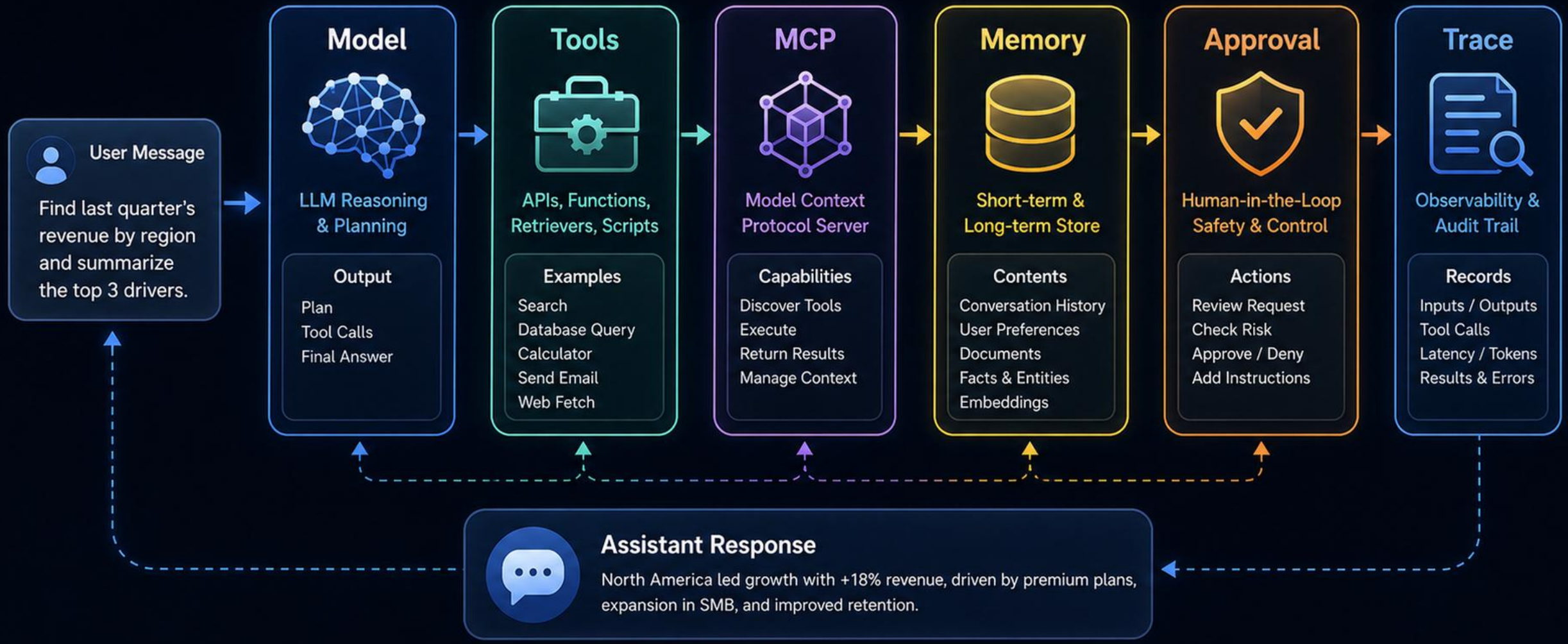
Long Work Should Leave The Request

Fast response now, background job after.



AI Backends Add A New Layer

Model calls, tools, MCP, memory, approvals, and traces.

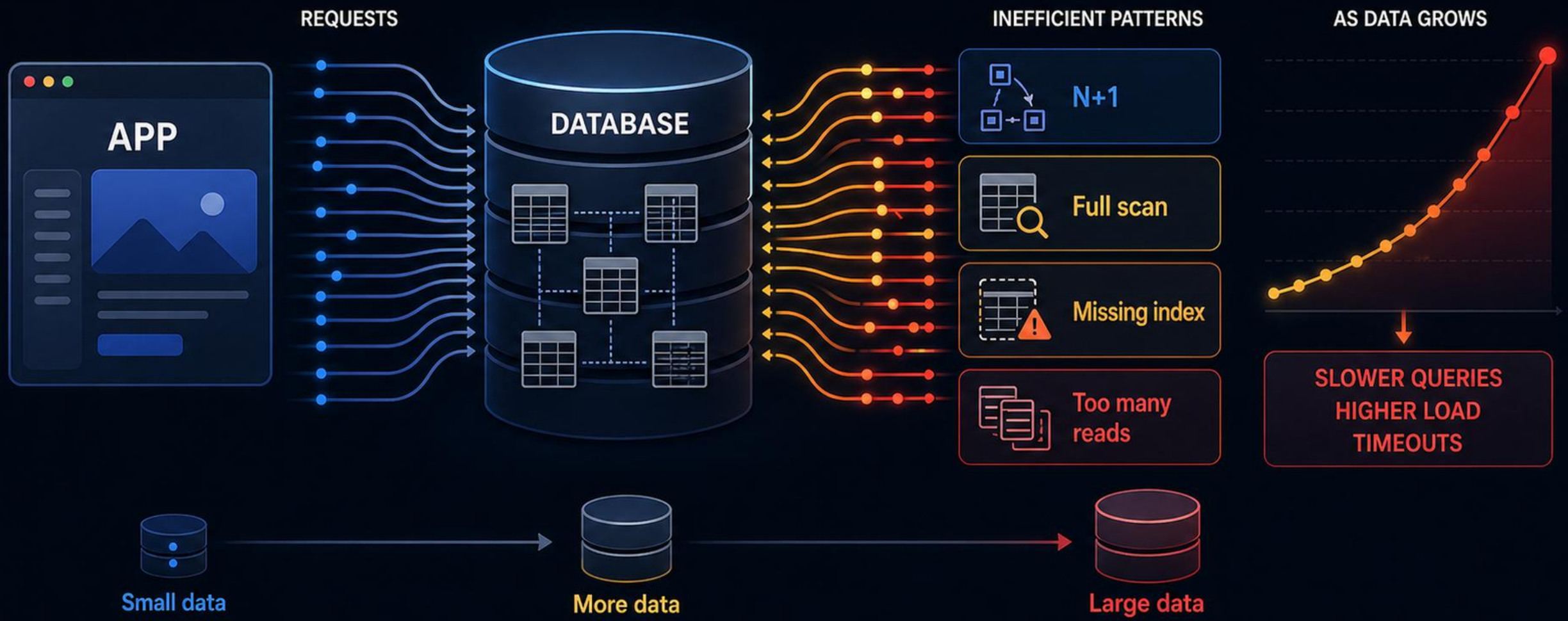


Agents Are Not Just Chatbots

They pursue goals through steps, tools, and state.

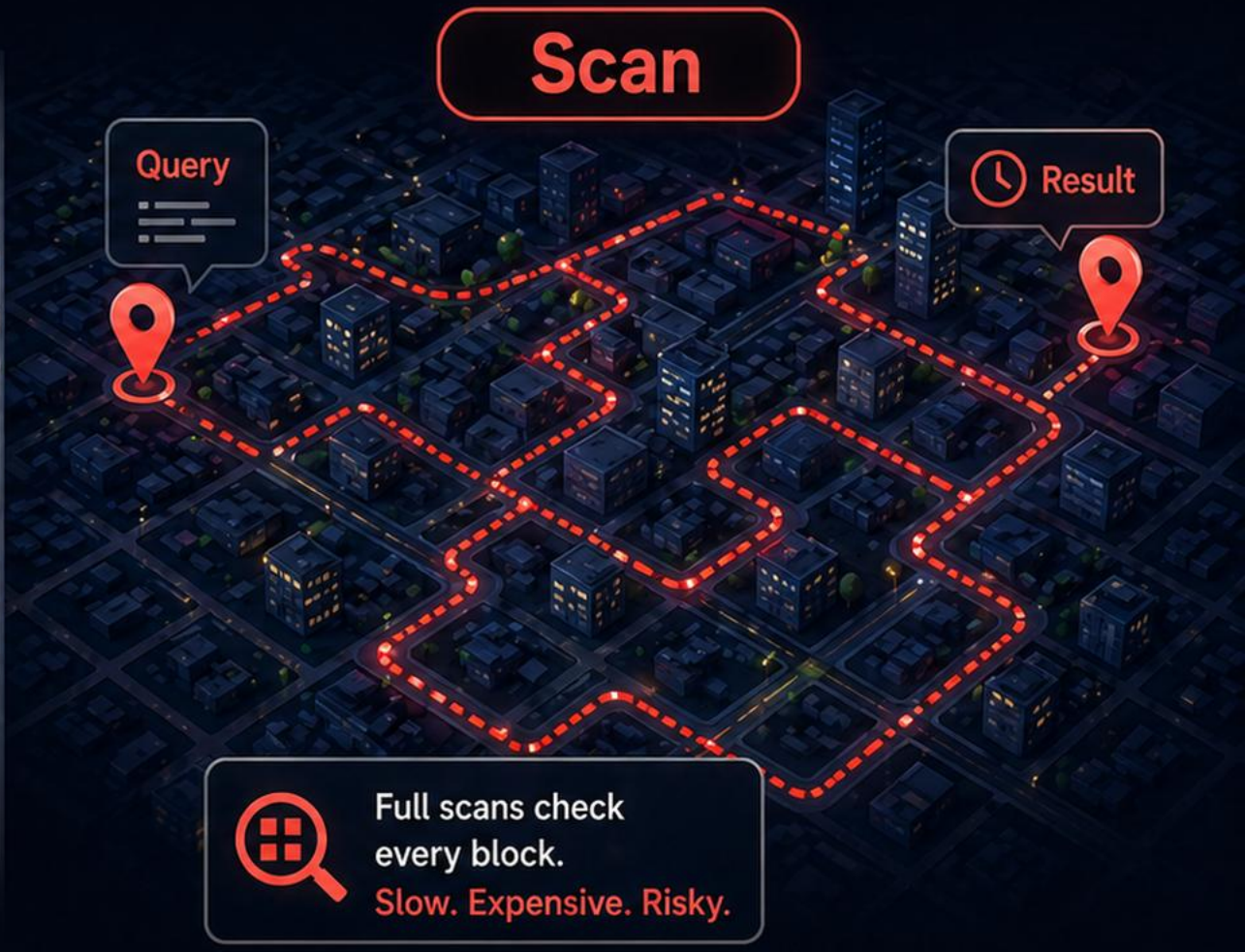
Most Bottlenecks Start In Storage

Small data hides bad access patterns.



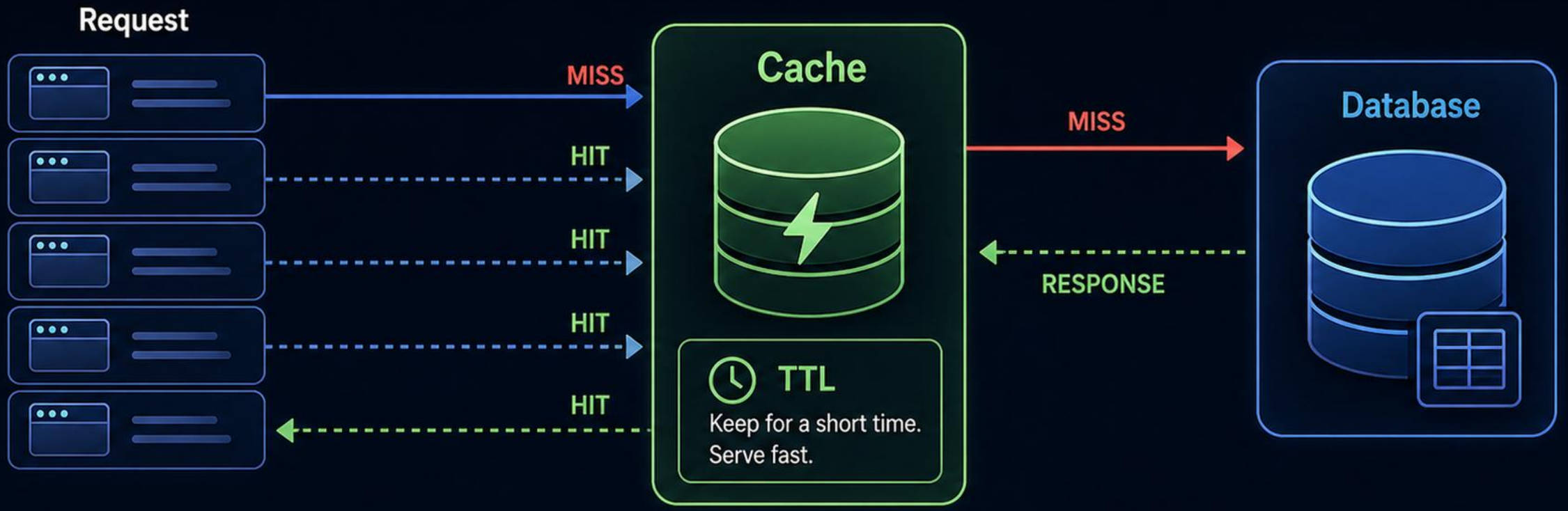
Indexes Are Roads Through Data

Without roads, every query walks the whole city.



Cache Means Ask Less Often

Speed improves because repetition disappears.



MISS Not in cache. Go to database.

HIT In cache. Return immediately.

Writes Break Differently

When many users update one value, correctness is at risk.

Hot value

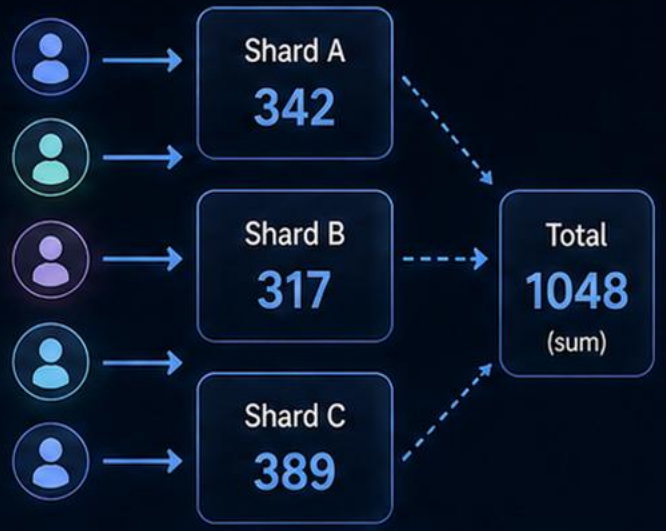
Contention

- Many writes collide on one value.
- Updates get lost or overwritten.
- Retries increase load and latency.

Queue

- Writes are serialized.
- No lost updates.
- Predictable and safe.

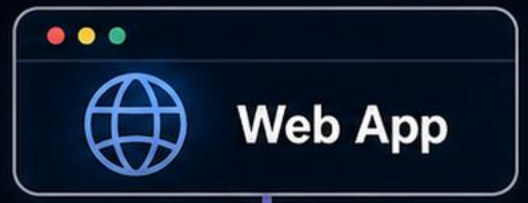
Shard



- Spread writes across shards.
- Lower contention.
- Aggregate when needed.

Infrastructure Is A Set Of Tradeoffs

VPS, serverless, containers, and edge solve different problems.



VPS

- Full control
- Predictable performance
- You manage everything
- Best for steady workloads

Serverless

- Pay per use
- Auto scale
- No servers to manage
- Best for spiky workloads

Containers

- Portable and consistent
- Fine-grained control
- You operate the platform
- Best for complex services

Edge

- Near users
- Low latency
- Limited compute
- Best for global delivery

Choose VPS for control and predictable costs.

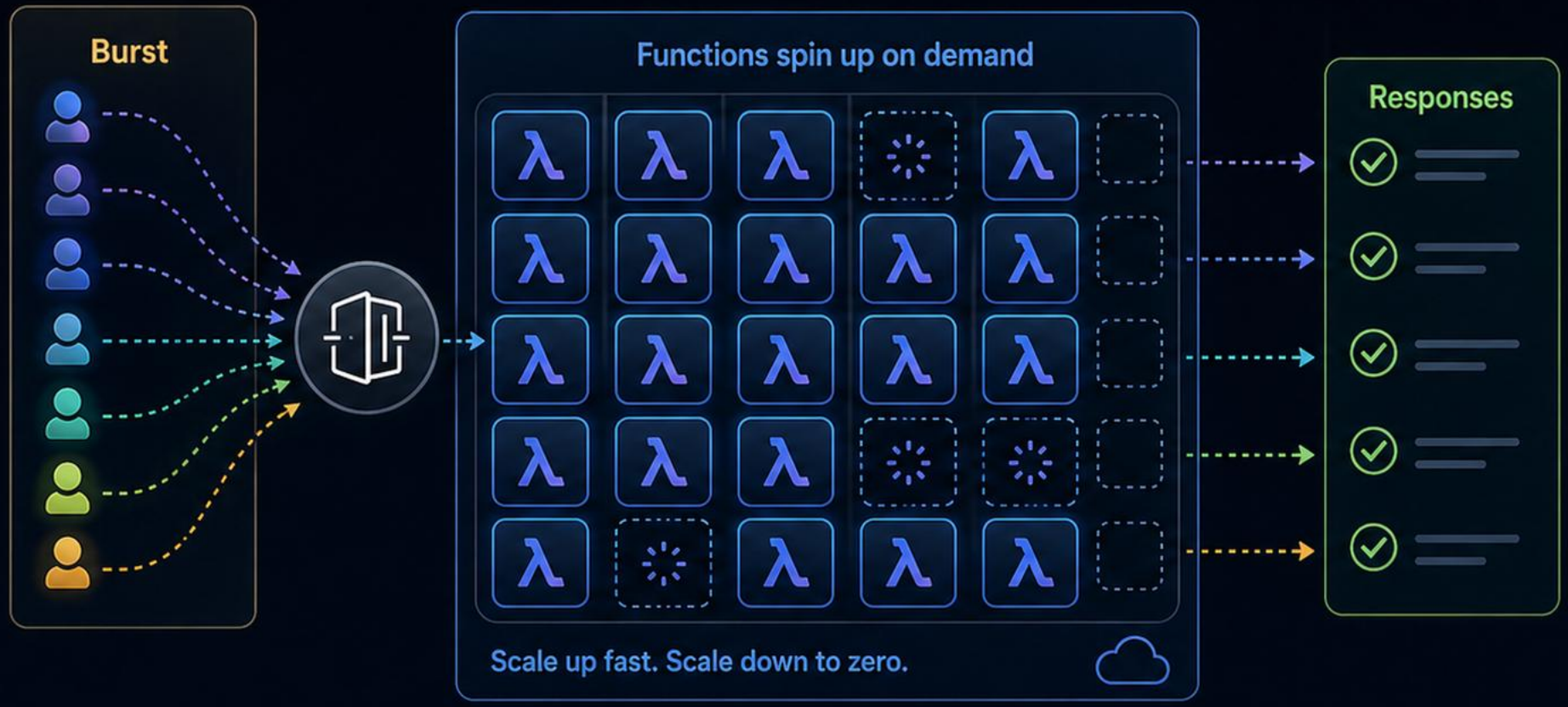
Choose serverless for speed and elasticity.

Choose containers for portability and scale.

Choose edge for proximity and performance.

Serverless Is A Default, Not A Religion

Great for bursts. Risky for long work and hidden limits.



Cold start
First request is slower. Functions need time to initialize.

Timeout
Long work gets cut off. Design for completion within the limit.

Concurrency
Limited parallelism. High traffic can hit the ceiling.

Hidden limits
Quotas, request size, memory, and outbound calls all have caps.

Great for
- Spiky traffic
- Event-driven work
- Short tasks
- Pay only for usage

Be careful with
- Long-running jobs
- Stateful workflows
- Tight latency SLOs

The Bill Is Another Log

Cost tells you what your architecture is repeating.

Reads

High read cost means you're reading too much.

Bandwidth

High transfer cost means you're moving too much data.

Function time

High execution cost means your functions run too often or for too long.

Images

High image cost means you're processing too many or at too large a size.

API calls

High request cost means you're calling too often or not batching.

Read the bill like a log.

Every dollar is a signal about repetition, inefficiency, or scale.

Growth Changes The Question

0-100 users, 100-10k users, and 10k-100k users need different judgment.

0-100 users

Exist

- Ship quickly
- Keep it simple
- Stay reliable
- Learn from real users

100-10k users

Find bottlenecks

- Measure what matters
- Identify constraints
- Optimize the right things
- Improve stability

10k-100k users

Redesign boundaries

- Re-architect for scale
- Decouple and distribute
- Automate and observe
- Prepare for more growth

Ask Better Questions Before It Breaks

Describe symptoms, platform, traffic, metrics, and recent changes.

- **Symptoms** — What's happening and what changed?
- **Platform** — Where is it running and what dependencies?
- **Traffic** — How many users, where, and what patterns?
- **Metrics** — What do graphs, logs, and alerts show?
- **Recent changes** — What was deployed, configured, or released?