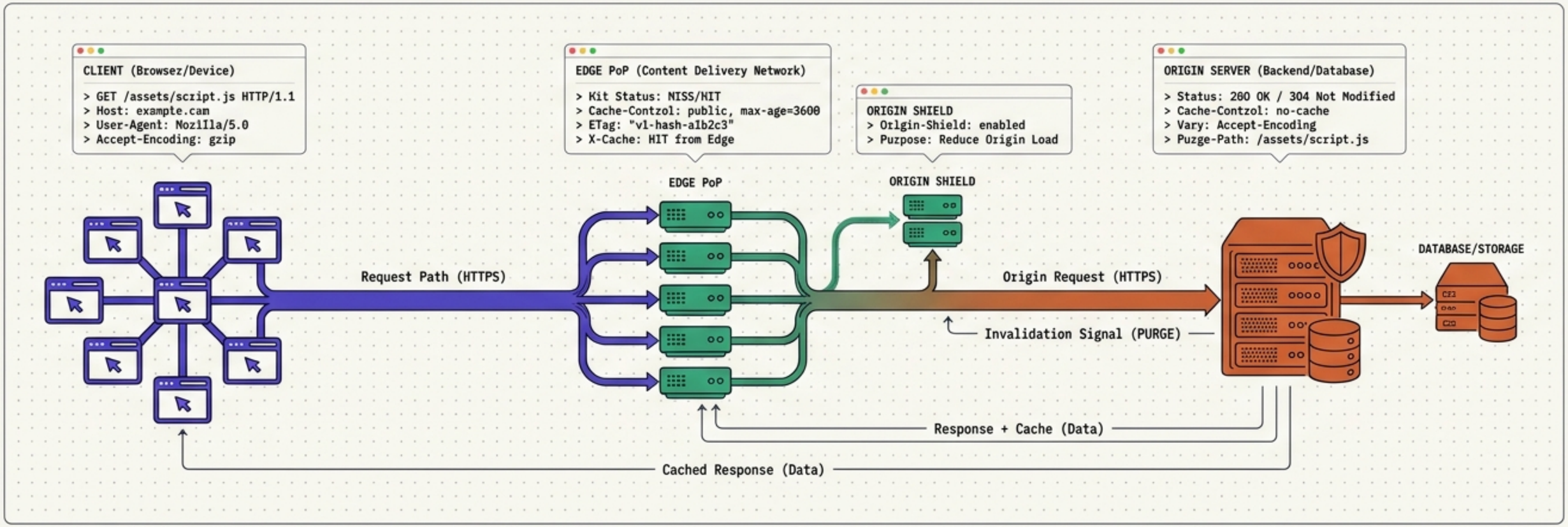
Web Architecture — S02E02: CDN Internals & Caching

RFC 9111, Origin Shielding, Cache Invalidation & Advanced Topology



Gemini Notebook

Data resides across distinct physical tiers, from client memory to the central origin

Tier 1: Private Caches



Browser Memory/Disk

Single-tenant

Stores personalized/ authenticated data

Tier 2: Shared Caches



CDN Edge PoPs / Reverse Proxies

Multi-tenant pooled resources

Shared across users

Tier 3: Origin Server



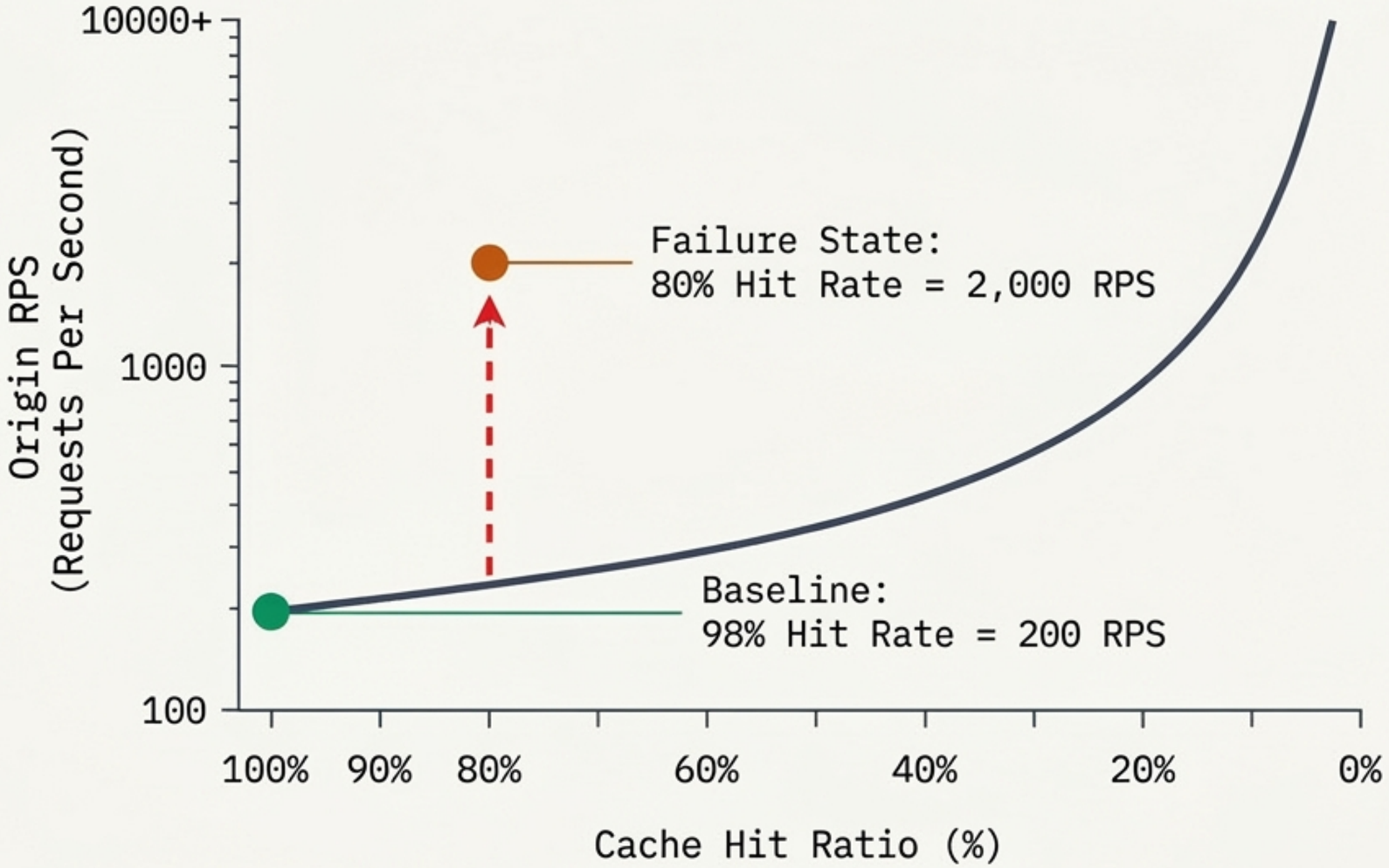
Backend infrastructure

Source of truth

Rule of architecture: Private caches protect the network; Shared caches protect the origin.

Gemini Notebook

A minor drop in cache hit rate causes an exponential surge in origin load



Blast Radius

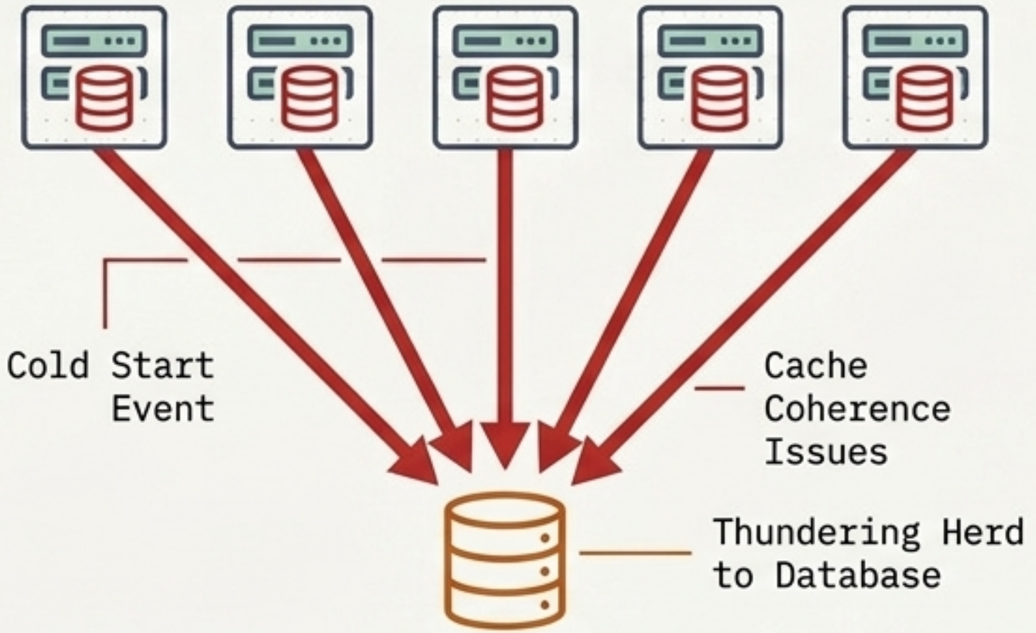
Dropping from 98% to 80% is only an 18% change in hit rate, but it causes a 10x surge in backend load.

Systems addicted to their caches melt when traffic patterns shift.

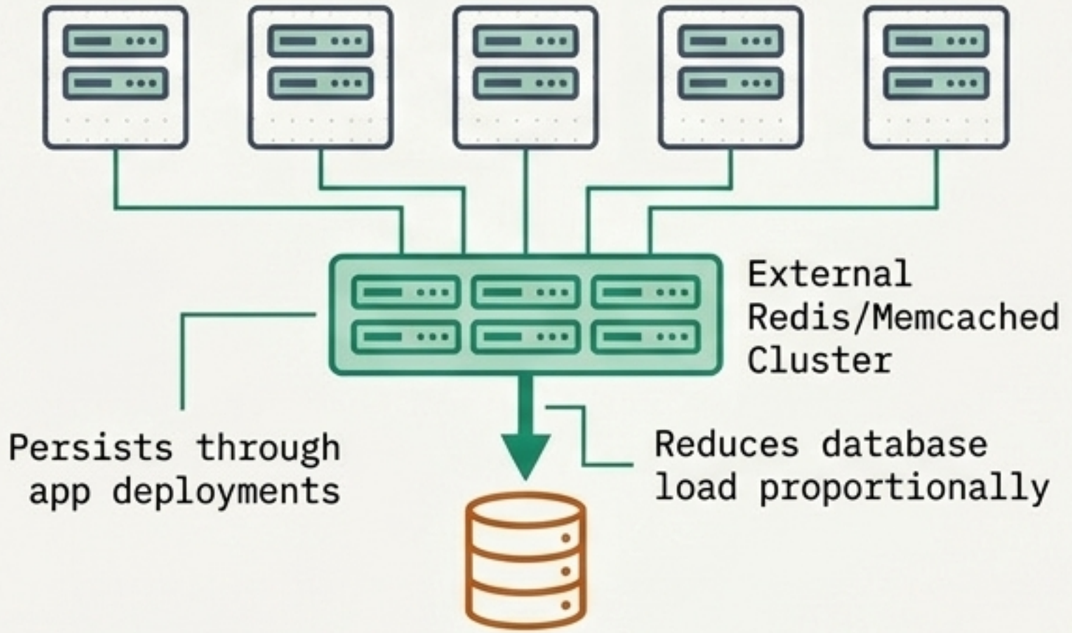
Gemini Notebook

On-box in-memory caches create severe fleet deployment hazards

In-Memory Caches [Danger]



External Side-Cache [Safe]



Rule of thumb: Downstream load with on-box caching scales linearly with fleet size. External caches break this dependency.

Gemini Notebook

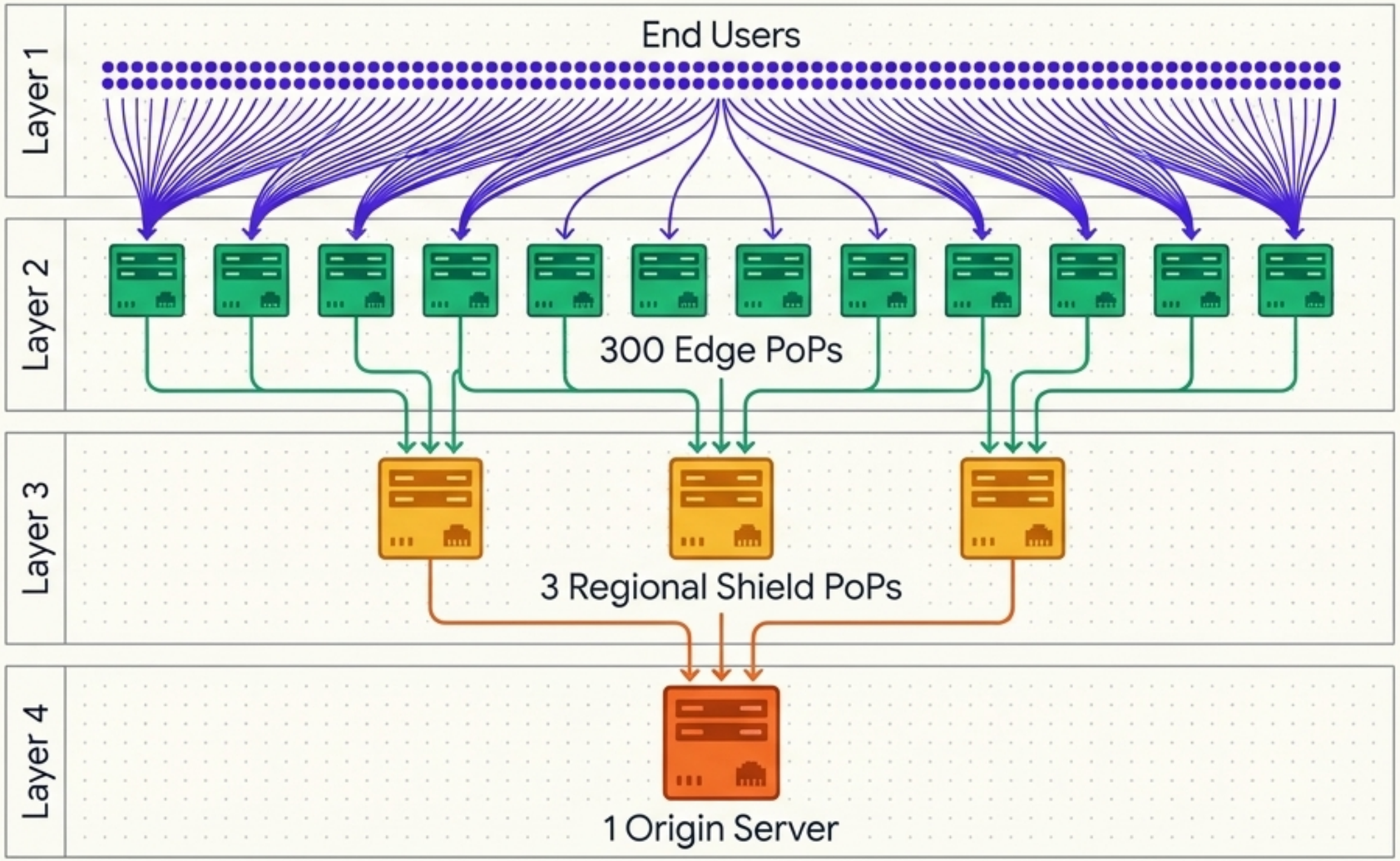
A massive global edge network cannot prevent a thundering herd on cold misses



When a cache is cold, every isolated edge node acts independently. 300 edge PoPs mean 300 independent cache misses forwarded to the origin at the exact same millisecond.

Gemini Notebook

Inserting a regional origin shield tier mathematically caps backend load

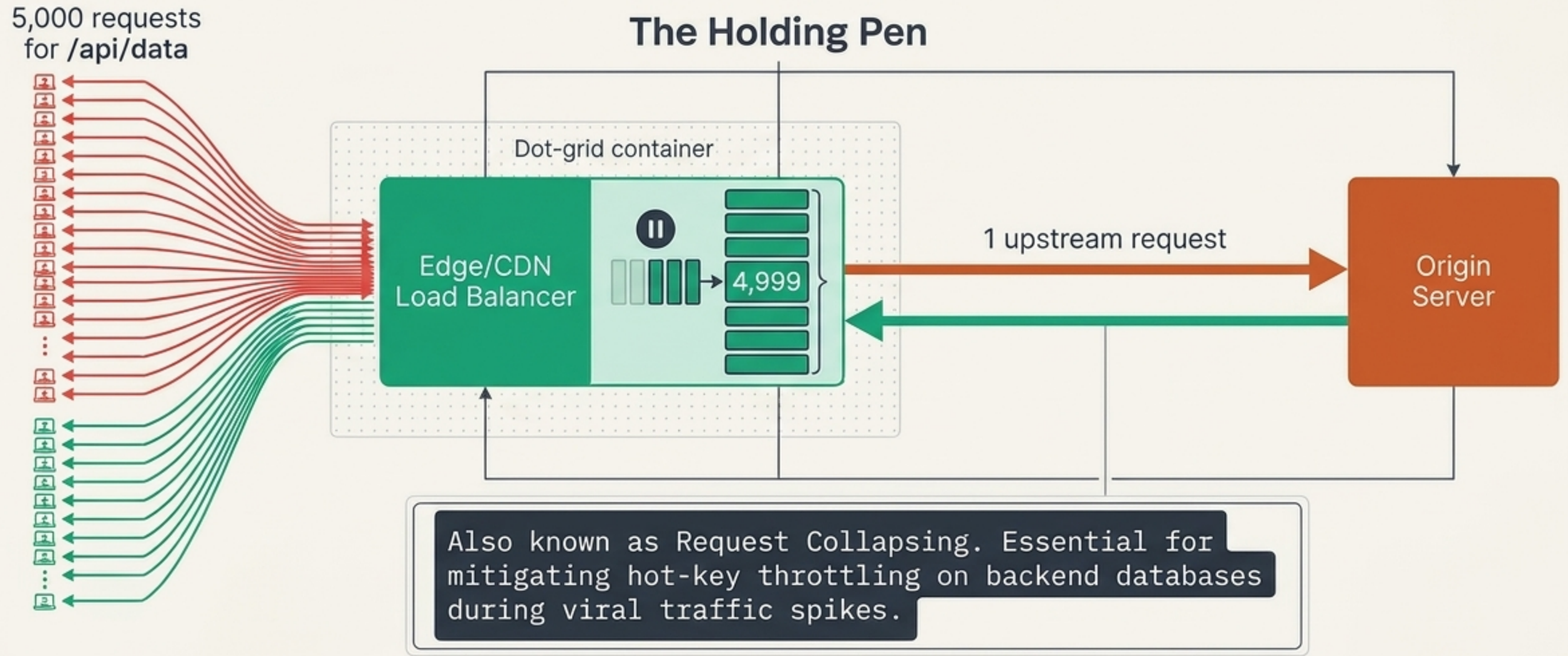


By routing misses through an intermediate layer, `origin` load is reduced from `N (number of edge nodes)` to `1 (number of regional shields)`.

The `shield absorbs the global impact.`

Gemini Notebook

Request coalescing collapses thousands of simultaneous misses into a single fetch



Gemini Notebook

Response Age vs. Freshness Lifetime




```
Cache-Control: max-age=3600
```

Base TTL. Browser/Private cache stores for 1 hour.

```
Cache-Control: s-maxage=86400
```

Shared cache override. CDN stores for 24 hours, ignoring `max-age`.

```
Cache-Control: public
```

Explicitly permits shared caching, even if the request contained an `Authorization` header.

Gemini Notebook

The most dangerous **misconception in caching is conflating no-cache with no-store**

`no-cache`	`no-store`
(The Revalidator)	(The Ghost)
Storage: YES. The payload is written to disk/memory.	**Storage:** NO. Zero persistence.
Rule: Mandates a conditional revalidation (e.g., If-None-Match) with the origin before serving.	**Rule:** Must absolutely not be written to non-volatile storage. Drops the payload immediately after serving.
Use Case: Always-fresh HTML documents.	**Use Case:** Bank balances, PII, sensitive patient data.

Gemini Notebook

Strict enforcement directives override default heuristic browser behaviors

>_ The Disconnected Cache

```
Cache-Control: max-age=3600, must-revalidate
```

If the cache is disconnected from the origin (e.g., network partition), it is strictly forbidden from serving a stale. It MUST return a 504 Gateway Timeout.

>_ The Reload Suppressor

```
Cache-Control: max-age=31536000, immutable
```

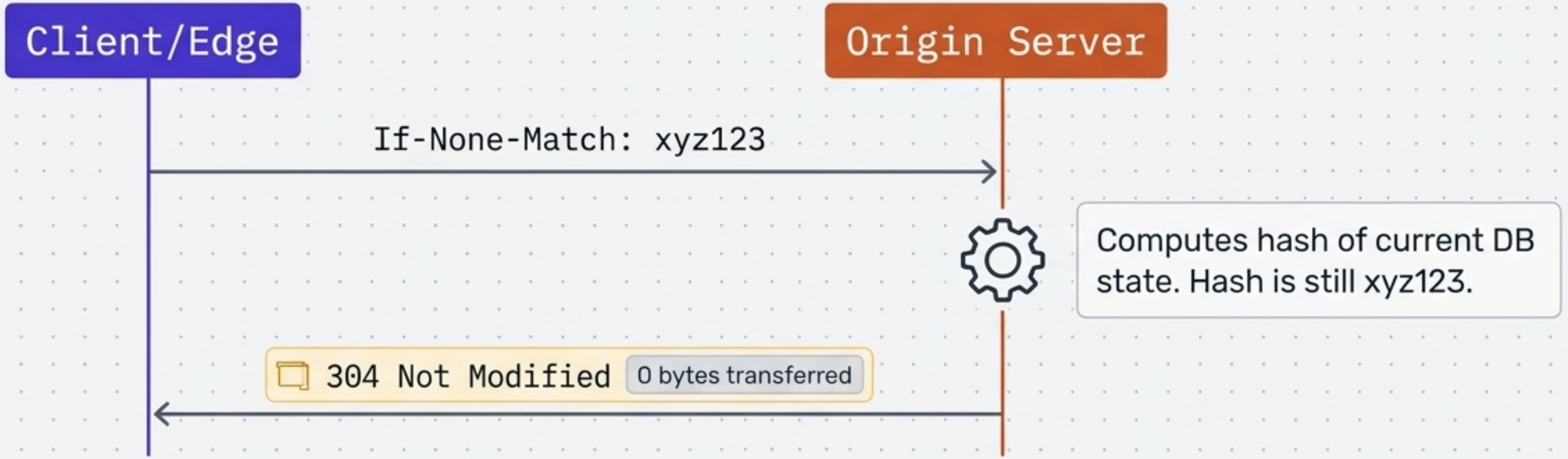
Informs the browser the file literally never changes.

Even if the user aggressively mashes the “Refresh” button, the browser will NOT send an If-None-Match revalidation request.

Saves massive amounts of unnecessary 304 traffic.

Gemini Notebook

Conditional requests leverage ETags to eliminate redundant data transfer



Strong ETag: xyz123 (Byte-for-byte identical representation)

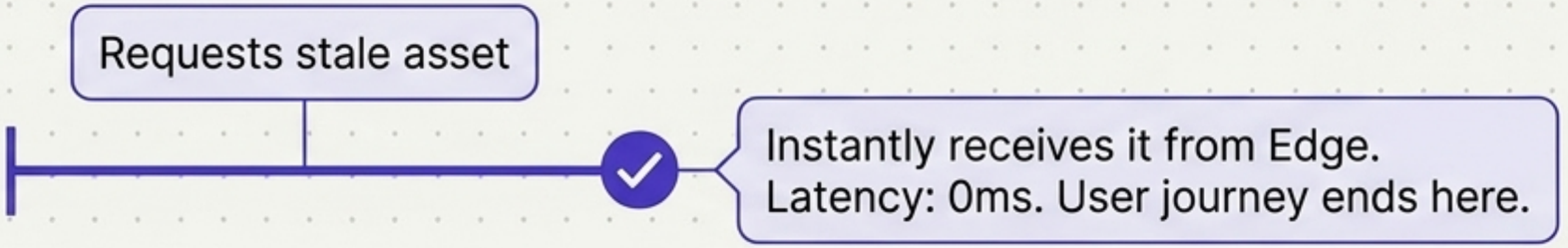
Weak ETag: W/xyz123 (Semantically equivalent, e.g., gzip vs brotli)

RFC Rule: ETag / If-None-Match always takes strict precedence over Last-Modified timestamps.

Gemini Notebook

RFC 5861 async extensions eliminate tail-latency spikes on stale assets

The User Experience



The CDN Background Process



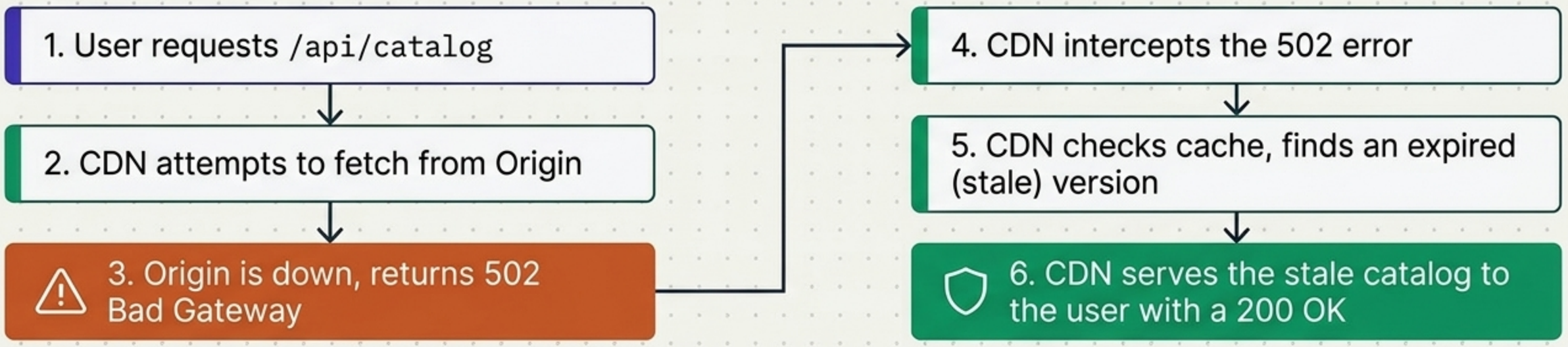

```
Cache-Control: max-age=600, stale-while-revalidate=3600
```

Translation: Serve from cache for 10 mins. For the next hour, serve stale instantly while updating in the background.

Gemini Notebook

Soft TTLs provide graceful degradation during catastrophic origin outages

Graceful Degradation

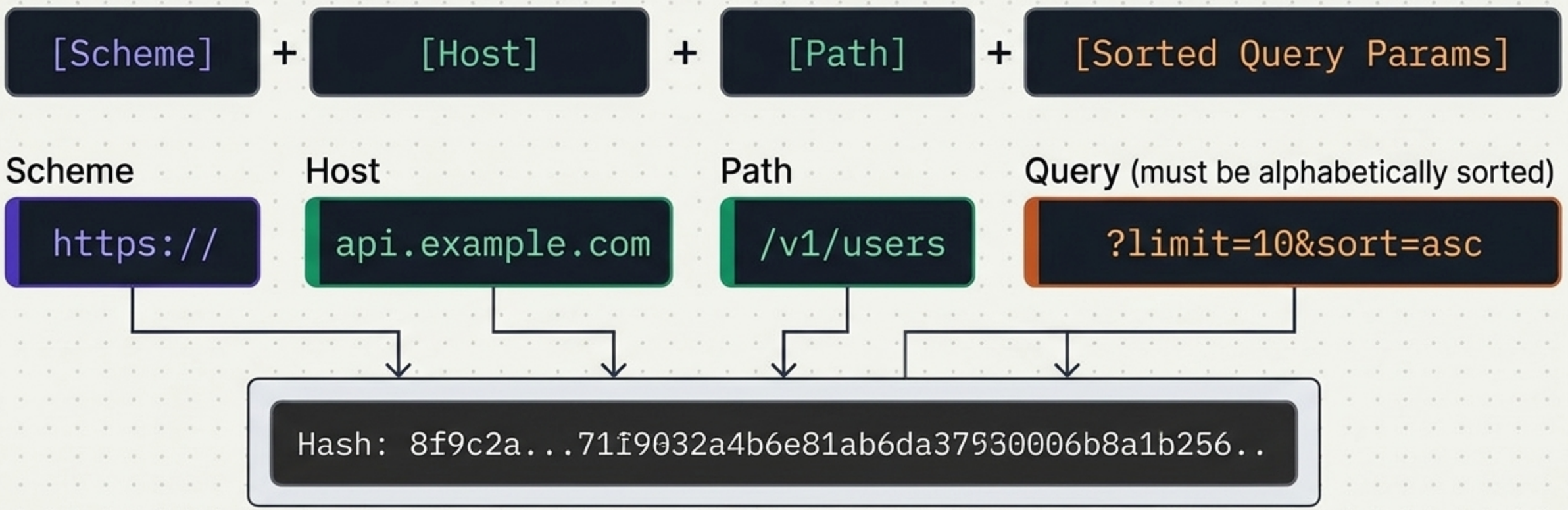



```
Cache-Control: max-age=600, stale-if-error=86400
```

Translation: If the backend dies, you have 24 hours of stale cache buffer to fix the database before users see an error screen.

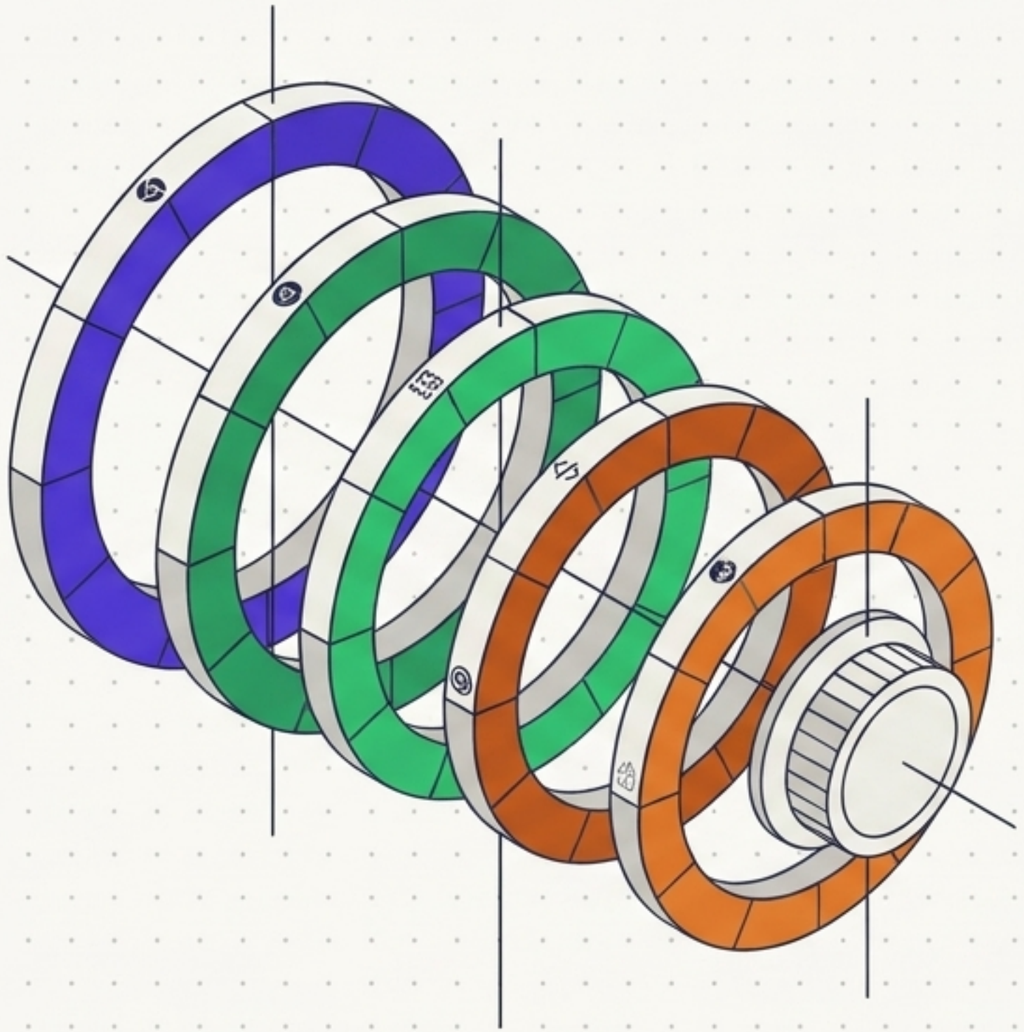
Gemini Notebook

Caches are fundamentally distributed Hash Maps requiring strict key normalization



A cache hit only occurs when the incoming request's constructed key perfectly matches the stored hash. Unnormalized data destroys hit ratios.

Gemini Notebook



Vary header: A combination lock exponentially fragmenting hit rates.

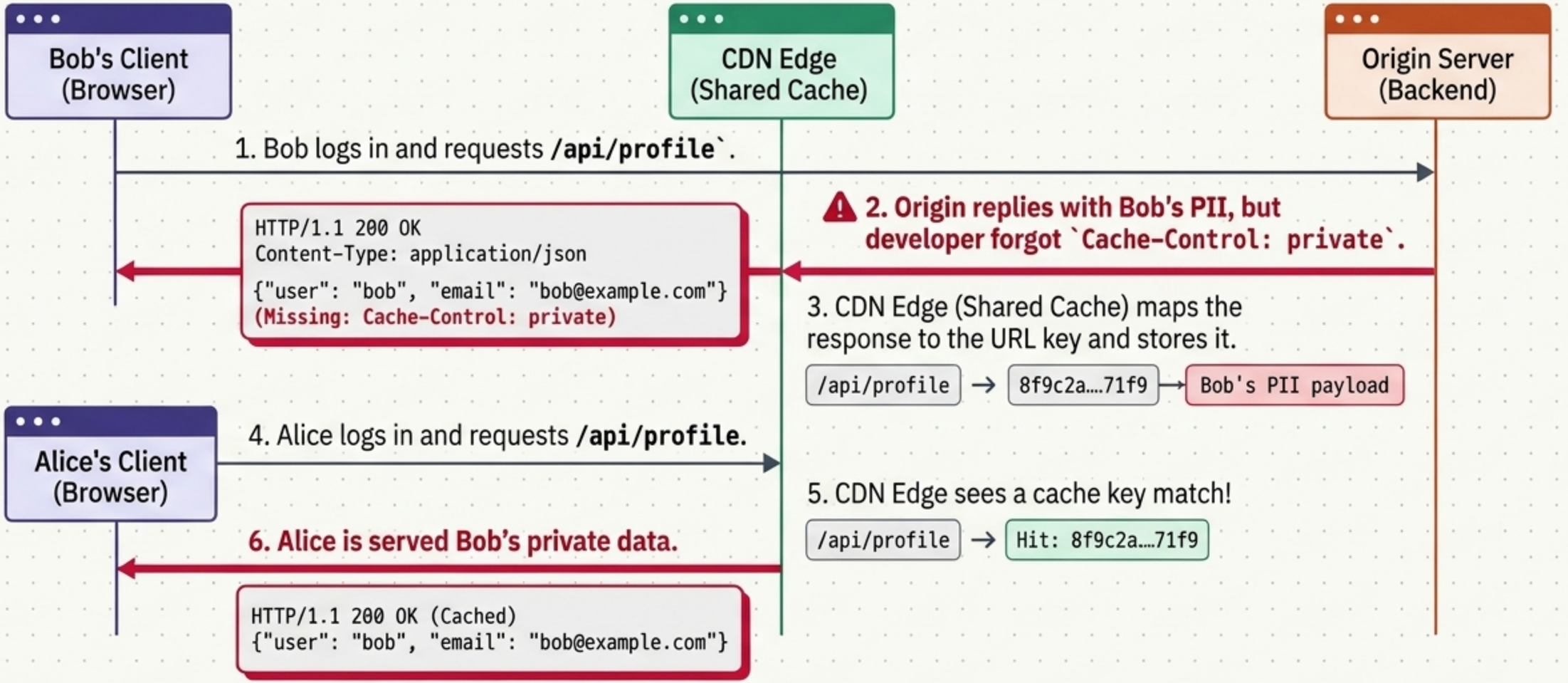
Common Vary Header Traps

1	`Vary: User-Agent`	Creates a unique cache entry for every single browser version. Hit rate approaches 0%.
2	`Vary: Accept-Encoding`	Without edge normalization, creates fragmented entries for **gzip**, **br**, and uncompressed requests.
3	`?utm_source=twitter`	Marketing tracking parameters generate unique cache keys for every single click.

Solution: Strip all tracking parameters and normalize User-Agents at the edge *before* constructing the cache key.

Gemini Notebook

Missing private directives on authenticated routes cause fatal PII leaks



SHARED CACHE WARNING: Shared caches use the URL as the key, not the authentication cookie. Authenticated endpoints MUST return `**Cache-Control: private, no-store**` to prevent cross-tenant data leakage.

Gemini Notebook

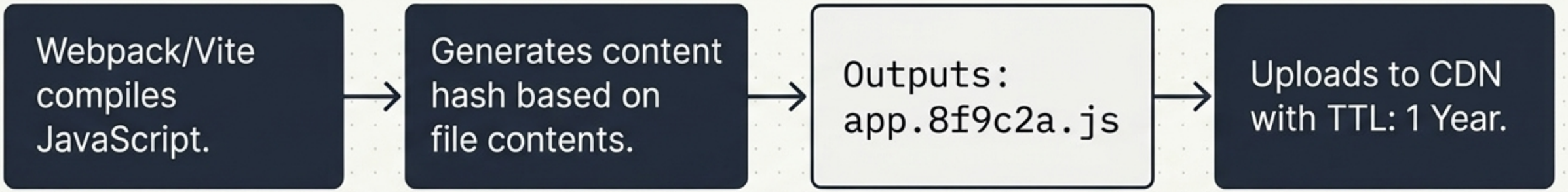
Soft invalidation via Surrogate-Keys prevents origin stampedes during purges

CMS Updates	
Strategy A: Hard Purge (Anti-Pattern)	**Strategy B: Soft Purge via Cache-Tags (Best Practice)**
• **Action** DELETE request sent to cache key. Data is wiped. • **Result** Next user experiences high latency (cache miss). If traffic is high, causes a micro-thundering herd to the origin.	• **Action** API calls edge to invalidate Surrogate-Key: `` `Surrogate-Key: article_123` ``. • **Result** Edge simply marks the tag as stale (does not delete). Next user triggers `` `stale-while-revalidate` ``. New data is fetched seamlessly in the background with zero user latency.

Gemini Notebook

The most reliable cache invalidation strategy is avoiding invalidation entirely

A modern CI/CD build process:




```
Cache-Control: public, max-age=31536000, immutable
```

Because the hash is in the filename, the file will literally never change. You never need to issue a purge request. To update the app, simply deploy a new HTML document pointing to the new hashed filename.

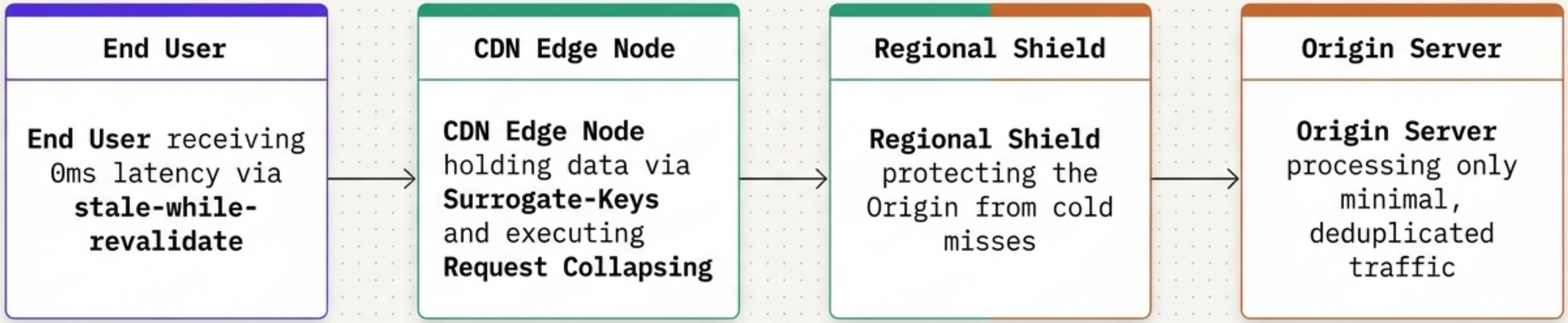
Gemini Notebook

The Production Blueprint: Standardized header policies by asset type

Asset Type	Cache Policy	Behavior
Static Assets (JS/CSS/Images)	`public, max-age=31536000, immutable`	Cache forever, bust via filename
SSR / ISR HTML Documents	`public, max-age=0, s-maxage=60, stale-while-revalidate=86400`	Revalidate in browser, CDN caches briefly, hide latency via async fetch
Public JSON APIs (e.g., Catalog)	`public, max-age=15, stale-if-error=3600`	Micro-caching for spikes, graceful degradation on backend failure
Authenticated Private Routes	`private, no-cache, no-store, must-revalidate`	Zero persistence, mandatory validation, strictly tied to client session

Gemini Notebook

Synthesis: Topology and Protocol operating in unified harmony



Scale is achieved at the edge, but controlled by the origin. Cache aggressively, shield regionally, collapse requests, and leverage RFC 5861 to hide latency. The best cache invalidation is the one you never have to make.

Gemini Notebook