Engineer

Cache

a stored copy a browser may reuse instead of asking

What it is

A cache is a store that keeps copies of data close to whoever reads them, so that a repeated read is answered from the copy instead of from the slower place where the data really lives. The plural is caches; one stored item is a cache entry, and the data in it is a cached copy.

A cache is never the source of truth. The file, row or server response it copies stays authoritative, and every entry can be thrown away and fetched again. That separates a cache from a replica, which replication keeps complete and in step on purpose, and from a buffer, which holds data in transit that may exist nowhere else yet. The Linux page cache is both: it keeps file data that was read and takes writes first, marking those pages dirty until they reach the disk.

Caches stack up between users and data. The kernel's page cache sits under every file a program reads. PostgreSQL keeps its own pool of pages, `shared_buffers`, on top of it. A browser keeps HTTP responses in a private cache for one person; a CDN or reverse proxy keeps a shared cache for many. A page's script can keep responses in Cache Storage through a service worker, apart from the browser's HTTP cache. And an application can keep values it has computed in a key-value store such as Redis.

The vocabulary is the same at every layer. A read that finds its entry is a hit; one that does not is a miss and goes on to the origin. The share of reads that hit is the hit ratio. Each entry is found by its key, which for an HTTP cache is at least the request method and the URL. A full store drops entries to make room, which is eviction, and its eviction policy picks what goes; least recently used, or LRU, is the common one.

Use the concept in a decision

Put “What is a cache?” to work in a real decision. We can help relate the concept to the system you are building or buying. Finding the queries and the missing indexes behind a system that has got heavier every year, and fixing the ones that matter.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. 30 days of electricity data in 180 ms — 180 ms to aggregate a month of meter readings

30 days of electricity data in 180 ms Get in touch →

Why it matters

A cache is usually the cheapest speed a system can buy, cheaper than new hardware or a new schema. The bill arrives as memory, as copies in places that are hard to reach and as copies that stop being true. Three decisions about the store settle most of it.

The first is where it sits. The nearer the copy is to the reader, the more it saves: a copy in the browser skips the network as well as the server. It is also further from the owner. A Redis instance can be emptied with one command; a copy on a visitor’s phone stays there under whatever rules were sent with it.

The second is size, because memory given to a cache is taken from something else. PostgreSQL’s documentation puts the default for `shared_buffers` at typically 128MB. On a dedicated server with 1GB or more of memory it suggests starting at 25%, and because PostgreSQL depends on the operating system’s cache as well, it says more than 40% is unlikely to work better than a smaller amount. Two caches share one machine.

The third is who shares the store. RFC 9111 separates a private cache, for one user, from a shared cache, which reuses the responses it stores for more than one user. A personal response in a shared store is how one person’s page reaches another, so a shared cache must not reuse a response to a request that carried an `Authorization` header unless the response explicitly allows it.

One rule follows from all three: a cache must be safe to empty. If emptying it loses data, it has become a database without a database’s guarantees. How long an entry stays good enough to serve, and what removes it early, belong to caching, the practice; the store’s own job is to hold copies, find them by key and give up space when it runs short.

What it looks like in practice

On a 464 MB host that runs its own forge and web server, the kernel's page cache decides what an alarm should measure. A dead-man's switch that raises the alarm by going quiet runs there every fifteen minutes, with jitter: a script measures available memory and free disk and pings an external service only while both are above their thresholds. Available is the right measure because on a healthy Linux box free memory sits near zero, with the kernel using idle memory as cache. The script read the correct column, but six variable names around it said "free". Someone trusting those names against a ten per cent threshold would see twelve megabytes free on 464 MB, call the check broken and make a one-character change, leaving an alarm that breaches permanently and is muted within a week. The naming has since been corrected.

One layer up, an application cache sits in front of a database. The note on optimising a database before sharding it puts one in front of a sell-out. When every unit of an item is claimed, the database can say when the earliest claim lapses, ten minutes after it was made, and the note caches "fully reserved" with exactly that expiry. Until then visitors are turned away by the cache rather than by the database. A claim released early, by a declined card or an abandoned basket, deletes the entry, so the unit is on sale again at once. The cache may refuse; only the database sells, through a conditional `UPDATE`. A stale entry can cost a sale for a few seconds and can never sell one unit twice.

The outermost layer is the visitor's own device, where this site creates no script-managed store. Its page on cookies says it uses neither IndexedDB nor Cache Storage and installs no service worker. A browser may still keep responses in its ordinary HTTP cache, which is separate from an offline copy managed by the site.

Where it goes wrong

After a restart, the origin takes every read at once. A cold cache misses on everything, so a deploy, a cache restart or a failover sends the whole read load to the database. A system that coped only because most reads were hits can fall over before the cache refills. Size the origin for the miss path, warm the cache before it takes traffic, and bring traffic back in steps. PostgreSQL's `pg_prewarm` does this for the buffer cache, and can record what shared buffers held and read it back after a restart.

The cache grows until the machine runs out of memory. Redis sets no memory limit by default on 64-bit systems, so an instance used as a cache grows with its key space until the operating system steps in. Set a limit and a policy together. Under `noeviction`, a full instance answers commands that would cache new data with an error, which suits a store that must keep everything and surprises one meant to forget. Redis's documentation calls `allkeys-lru` a good default absent a reason to prefer another:

```
maxmemory 256mb
maxmemory-policy allkeys-lru
```

A visitor is served a page in the wrong language. The key leaves out an input the value depends on. A page rendered in Danish is stored under its URL alone, and the next reader gets Danish whatever they asked for. Build the key from every input that changes the output, such as language, currency or role. For HTTP caches, the `Vary` response header names the request headers that belong in the key, and RFC 9111 lets a cache reuse a stored response only when those headers match or after revalidating it.

The installed app keeps running old code. Entries in Cache Storage carry no expiry of their own and outlive an update of the service worker script. Files changed under the same cache name keep being served from the old copies, and a cache renamed each release without cleanup leaves one more on the device every time. Name each cache with a version, as the Service Workers specification recommends, and delete every older one once the new worker takes over.

Further reading

- RFC 9111: HTTP Caching — the IETF standard for private and shared HTTP caches, their keys and what a shared cache must leave alone.
- Linux kernel: memory management concepts — how the page cache holds file data in memory and why its pages count as reclaimable.
- PostgreSQL: Resource Consumption — `shared_buffers`, its usual default and sizing advice that counts the operating system's cache.
- Service Workers — the W3C specification of Cache Storage and of how it differs from the browser's HTTP cache.
- Redis: Key eviction — `maxmemory` and what a full cache does under each eviction policy.

Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a cache?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/cache/