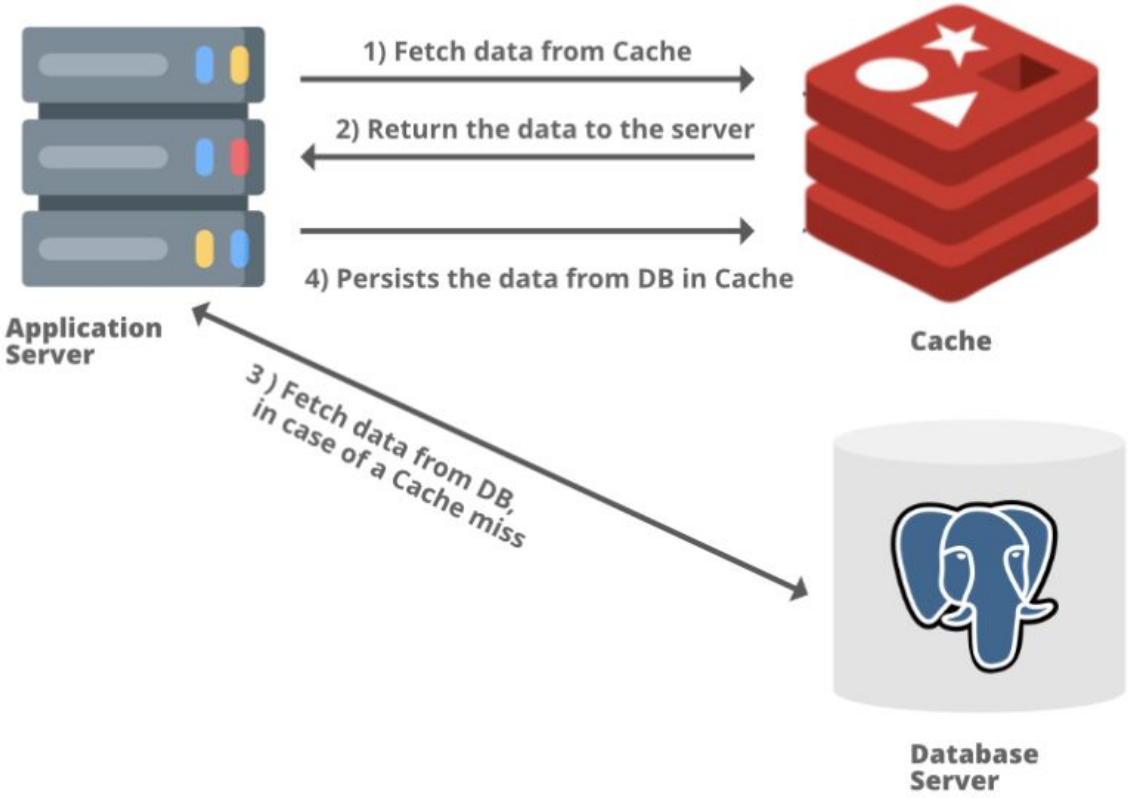
Caching

...

What is Caching

- Caching is the process of storing copies of files in a cache, or temporary storage location, so that they can be accessed more quickly.
- Consider it as a **short-term memory** that has **limited space**
    - but is faster and contains **the most recently accessed items.**
- “Cache Miss”: when something is not in the cache and has to be fetched

What is Caching



Caches

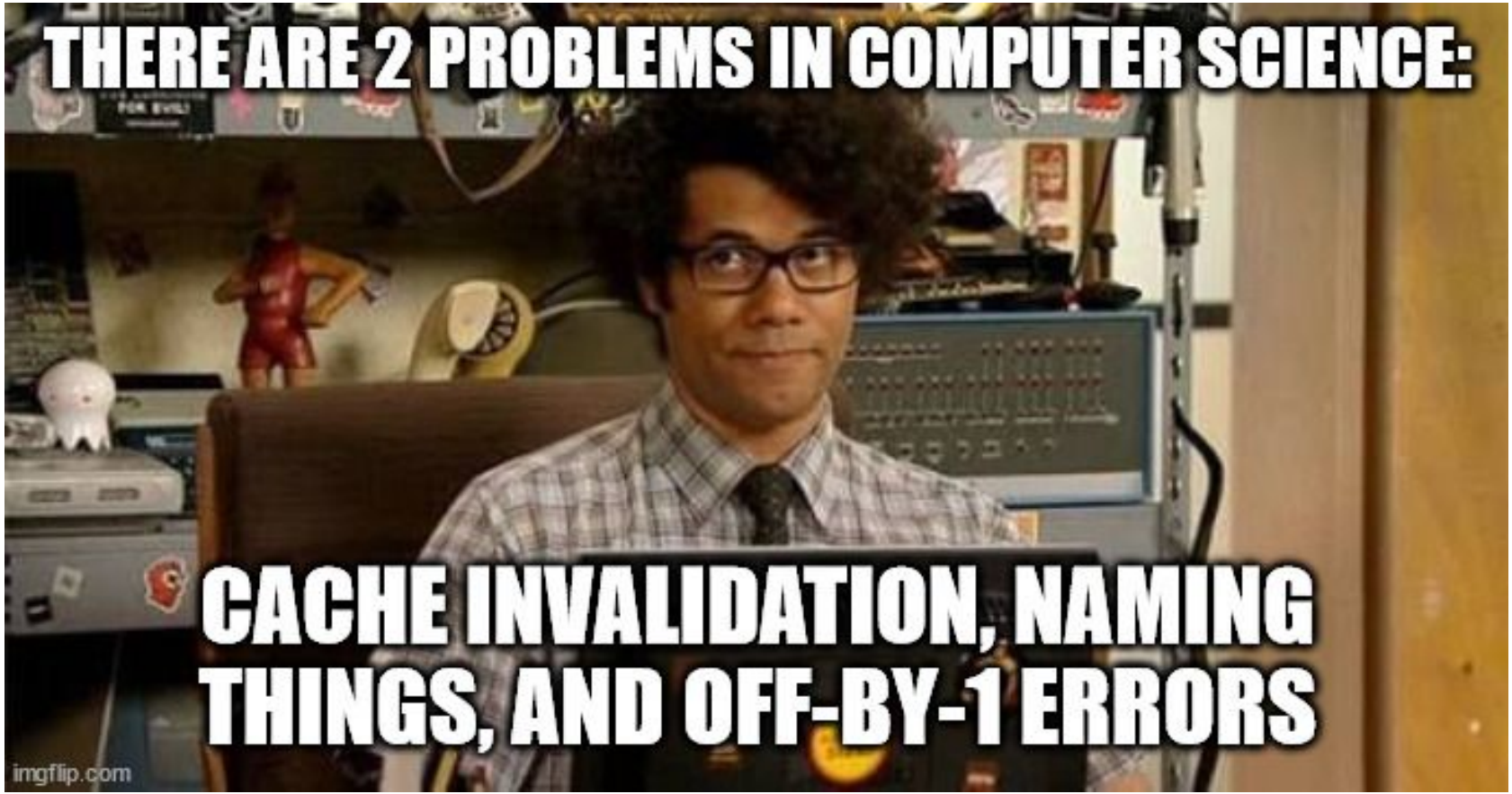
- Are usually a key-value pair (hashmap)
- VERY fast lookup
- When are caches used?
  - Browsers cache HTML files, JavaScript, and images in order to load websites more quickly
  - DNS servers cache DNS records for faster lookups
  - CDN servers cache content to reduce latency.
  - Caches added before databases to minimize DB queries

Browser Caching

- Every time a user loads a webpage, their browser has to download data in order to display that webpage.
- To shorten page load times, browsers cache most of the content that appears on the webpage, saving a copy of the webpage's content on the device’s hard drive
- This way, the next time the user loads the page, most of the content is already stored locally and the page will load much more quickly.

Cache Eviction (Invalidation)

- Cache invalidation (eviction) is deciding when to expire things from cache
  - Time to live (TTL)
  - First in First Out (FIFO)
  - Last in First Out (LIFO)
  - Least Frequently Used (LFU)
  - Random
    - Prevents many worst-case or degenerate times


THERE ARE 2 PROBLEMS IN COMPUTER SCIENCE:
CACHE INVALIDATION, NAMING THINGS, AND OFF-BY-1 ERRORS
imgflip.com

Reducing deployment costs via caching

- Reduce DB reads:
  - Same query over and over -> cache the results
- Almost anything that is:
  - Repetitive
  - Slow and/or expensive
- Should be cached!

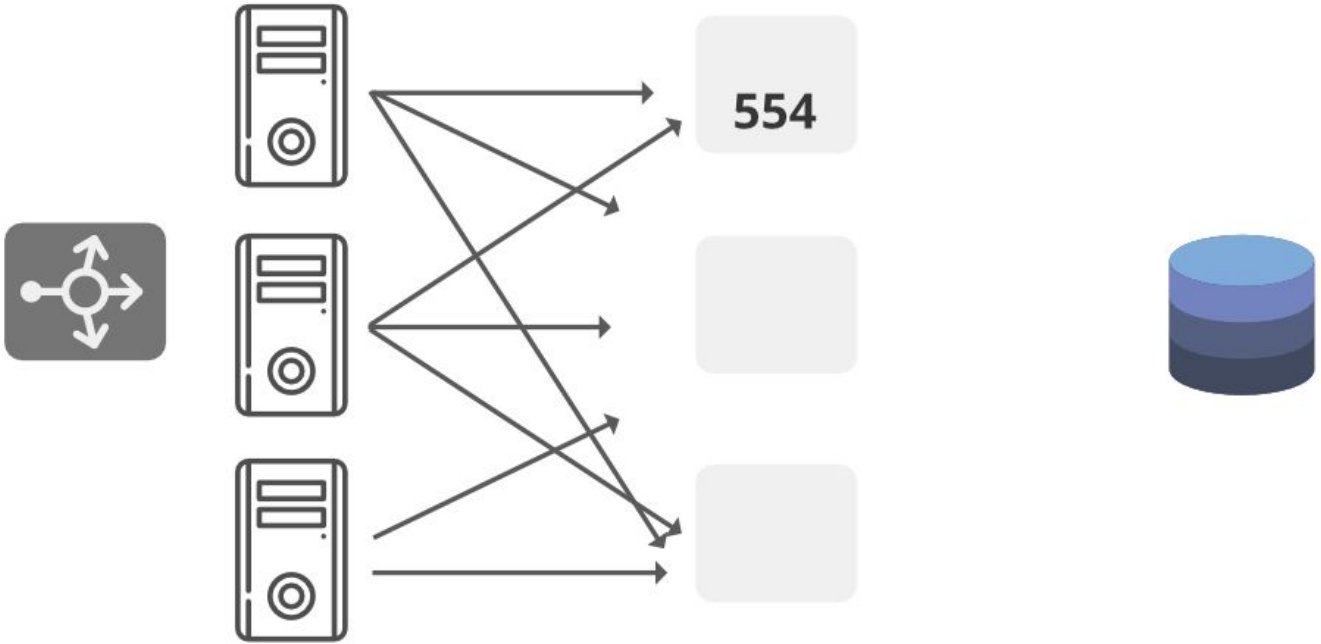
Local Cache

- A cache that is local to (completely contained within) a node, instance, program, or cluster
- Often the fastest (lowest-latency) implementation
- If the cluster, node, etc goes down, so does the cache

Distributed Cache

- Each node will have a part of the whole cache space
- Using a consistent hashing function, each request can be routed to where the cache request could be found.
    - Each of its nodes will have a small part of the cached data.
    - If a requesting node is looking for a certain piece of data, it can quickly know where to look within the distributed cache to check if the data is available.
    - We can easily increase the cache memory by simply adding the new node to the request pool.

Distributed Cache



Local vs Distributed

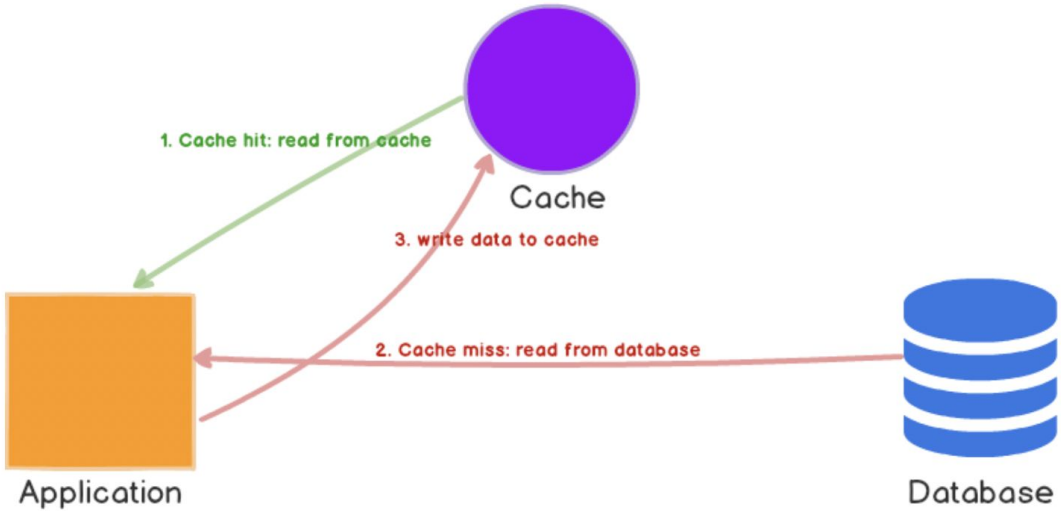
	LOCAL CACHE	DISTRIBUTED CACHE
SHOULD IT SCALE HORIZONTALLY?	✗	✓
WILL IT BE ACCESSED BY MULTIPLE SERVERS?	✗	✓
SHOULD IT HAVE HA CAPABILITIES?	✗	✓
SHOULD IT SHARE DATA BETWEEN SERVERS?	✗	✓
SHOULD IT HAVE THE FASTEST ACCESS?	✓	✗

Caching Strategies

- Refer to how you set up reads, writes to the cache and DB
- Many with different benefits!
- Here are several...

Caching Strategies: Cache Aside

- The cache sits on the side and the application directly talks to both the cache and the database.
- There is no connection between the cache and the primary database.
- All operations to cache and the database are handled by the application.

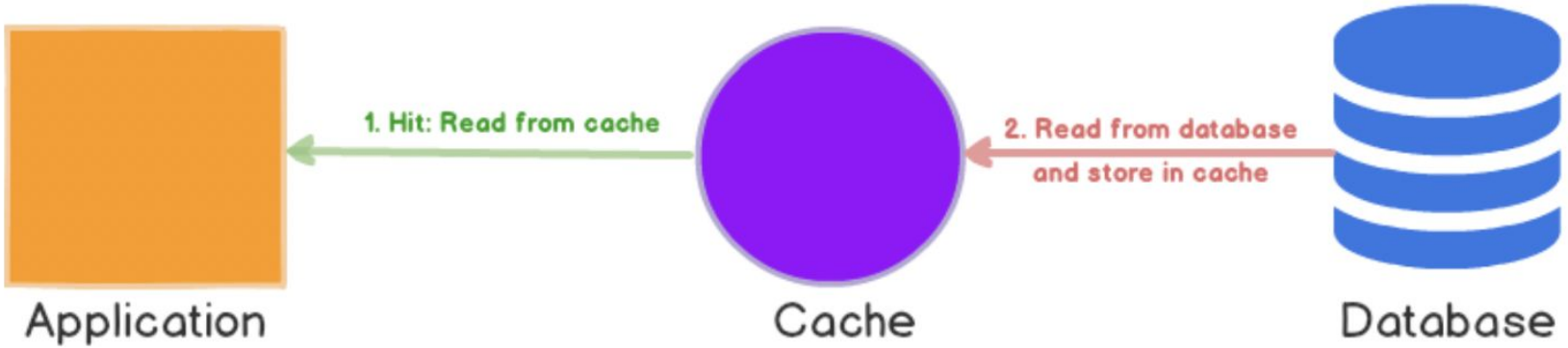


Caching Strategies: Cache Aside

- Work best for read-heavy workloads.
- Systems using cache-aside are resilient to cache failures.
  - If the cache cluster goes down, the system can still operate by going directly to the database.
  - It doesn't help much if cache goes down during peak load. Response times can become terrible and in worst case, the database can stop working.
- Data model in cache can be different than the data model in database.
- When cache-aside is used, the most common write strategy is to write data to the database directly.
  - When this happens, cache may become inconsistent with the database.
  - To deal with this, developers generally use time to live (TTL) and continue serving stale data until TTL expires.
  - If data freshness must be guaranteed, developers either invalidate the cache entry or use an appropriate write strategy, as we'll explore later.

Caching Strategies: Read-Through

- Read-through cache sits in-line with the database.
- When there is a cache miss, it loads missing data from database, populates the cache and returns it to the application.

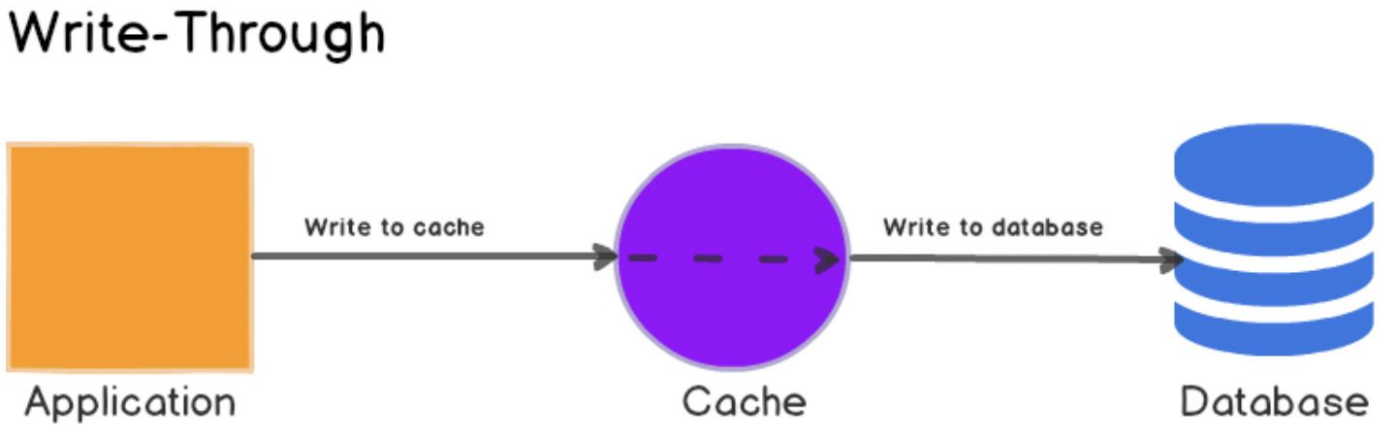


Caching Strategies: Read-Through

- While read-through and cache-aside are very similar, there are differences:
  - In cache-aside, the application is responsible for fetching data from the database and populating the cache.
  - In read-through, this logic is usually supported by the library or stand-alone cache provider.
  - Unlike cache-aside, the data model in read-through cache cannot be different than that of the database.
- Read-through caches work best for read-heavy workloads when the same data is requested many times.
  - The disadvantage is that when the data is requested the first time, it always results in cache miss and incurs the extra penalty of loading data to the cache.
  - Developers deal with this by ‘warming’ or ‘pre-heating’ the cache by issuing queries manually.
- Just like cache-aside, it is also possible for data to become inconsistent between cache and the database, and solution lies in the write strategy, as we’ll see next.

Caching Strategies: Write-Through

- Data is first written to the cache and then to the database.
- The cache sits in-line with the database and writes always go through the cache to the main database.
- This helps cache maintain consistency with the main database.



Caching Strategies: Write-Thru

- Introduce extra write latency
  - data is written to the cache first and then to the main database (two write operations.)
- Data consistency guarantee!
  - freeing us from using cache invalidation
  - assuming ALL writes to the database go through the cache.
- Can have readthrough/writethrough caches

Caching on cloud providers

- Cloud providers usually provide semi-automated ways to implement caches
- Some caching built-in (e.g. GCP storage)
  - AWS
  - GCP

Caching Cons

- Adds complexity to your app
- Caching can make debugging WAY harder
    - result stored in cache; think you have the right answer, hard to tell what's real