### ****1. What is a Cache?****

* A **cache** is a temporary storage layer that stores frequently accessed data to reduce access time and avoid repeated computations.
* It speeds up response times by minimizing calls to the database.

### ****2. Use Cases for Caching****

#### ****1. Avoiding Network Calls****

* Example: User profile information
  + If the same user repeatedly requests their profile, storing the profile in a cache avoids fetching it multiple times from the database.
  + **Key-Value Store**: Key = user\_id, Value = user\_profile.

#### ****2. Avoiding Expensive Computations****

* Example: Calculating the **average age** of all users.
  + Instead of querying the database and computing every time, calculate once and cache the result:
    - Key = average\_age, Value = 28.

#### ****3. Reducing Load on the Database****

* Cache stores commonly requested data.
* Servers query the **cache** instead of overloading the database.

### ****3. Why Not Store Everything in a Cache?****

1. **Cost**:
   * Cache typically uses faster but **expensive hardware** like SSDs.
   * Databases use cheaper, commodity hardware.
2. **Performance Degradation**:
   * Storing too much data increases **search times** in the cache.
   * Beyond a point, querying the cache becomes less efficient.

### ****4. Cache Policies****

**Cache policies** determine **when to add** or **evict** data from the cache.

#### ****1. Least Recently Used (LRU)****

* Keeps recently accessed entries at the top of the cache.
* Kicks out the **least recently used** entries from the bottom when the cache is full.
  + Example:
    - If a celebrity comment is frequently accessed, it stays at the top.
    - As its popularity declines, it is evicted.

#### ****2. Least Frequently Used (LFU)****

* Evicts data that is **accessed least often**.
* Less common in real-world systems.

#### ****3. Sliding Window Policies****

* Advanced policies used in systems like **Caffeine** by Google.
* Improves upon LRU for better performance.

### ****5. Problems with Caching****

#### ****1. Cache Eviction Issues****

* **Poor Eviction Policy**: If the cache constantly returns "cache misses" (no data), it becomes useless and increases overhead.
* **Thrashing**:
  + Occurs when the cache size is too small.
  + Example:
    - If user X and Y request profiles, and the cache stores only one entry, frequent switching leads to inefficiency.

#### ****2. Consistency Issues****

* **Data Inconsistency**:
  + Example: If a user changes their password, but the cache has outdated data, it could cause errors.
  + Critical for financial or security-sensitive data.

### ****6. Cache Placement****

Where to place the cache depends on the use case:

#### ****1. Local Cache (Close to Servers)****

* **In-Memory Cache**: Stored within the server's memory.
* **Advantages**:
  + Extremely fast access.
  + Reduces network calls.
* **Disadvantages**:
  + If the server fails, the cache is lost.
  + Inconsistency between different server caches.

#### ****2. Global Cache (Separate System)****

* A centralized cache, e.g., **Redis**.
* **Advantages**:
  + Shared by multiple servers.
  + Independent scaling of the cache layer.
  + More resilient to server failures.
* **Disadvantages**:
  + Slightly slower than in-memory caches.

**Preferred Choice**:

* **Global Cache** for higher accuracy and scalability.
* **Local Cache** for faster response times and simplicity.

### ****7. Cache Write Policies****

Cache performance depends on how data updates are handled:

#### ****1. Write-Through Cache****

* Data is **written to the cache first** and then propagated to the database.
* **Pros**: Ensures consistency between cache and database.
* **Cons**: Slower writes. May cause inconsistencies in in-memory caches across servers.

#### ****2. Write-Back Cache****

* Data is written directly to the **database** and then updated in the cache.
* **Pros**: Faster writes.
* **Cons**: Can lead to data inconsistency if the cache is not properly updated.

#### ****3. Hybrid Write Policy****

* A combination of Write-Through and Write-Back:
  + Write updates to the cache first.
  + Periodically **flush bulk updates** to the database.
* Suitable for **non-critical data** where small inconsistencies are acceptable for faster performance.

### ****8. Summary****

* **Caching** helps speed up responses, avoid redundant computations, and reduce database load.
* It relies on efficient **eviction policies** (e.g., LRU) and placement strategies (local vs. global cache).
* **Write policies** (Write-Through, Write-Back) determine how updates are synchronized with the database.
* Choose a caching strategy based on your system's needs (performance, consistency, and cost).

### ****Real-World Use Case: Redis****

* Redis is a popular in-memory data store used as a cache.
* Benefits include high speed, distributed caching, and resilience.