💻 Cache Memory

🔹 What is Cache Memory?

Cache memory is a **very high-speed memory** placed between the CPU and the main memory (RAM).

- It stores **frequently accessed instructions and data** so the CPU doesn't always need to fetch them from slower RAM.
- Usually made of **SRAM (Static RAM)** → faster but more expensive than **DRAM (Dynamic RAM)** used in RAM.

🔹 Why do we use Cache Memory?

- The **CPU works much faster than RAM.**
- If the CPU waits every time for RAM → performance drops (called the **memory bottleneck**).
- Cache solves this by **keeping a copy of frequently used data close to the CPU**, reducing access time.

🔹 How it Works (Simply)

1. **CPU requests data.**
2. Cache checks if the data is present (**cache hit**).
   - If **yes** → data is given instantly.
   - If **no (cache miss)** → data is fetched from RAM and stored in cache for future use.

⚠️ Problem: Sometimes cache may still hold **old (stale) data** if the original value in RAM has changed.
✅ Solution: This is handled using **cache coherence protocols** (write-through, write-back, MESI), which ensure cache and RAM always stay updated.

- **Write-through** → Every update in cache is immediately written to RAM.
- **Write-back** → Update happens in cache first, then RAM is updated later when necessary.
- **MESI Protocol** → Used in multi-core systems to keep caches synchronized across cores.

🔹 Levels of Cache

- **L1 Cache** → Inside CPU core, very small (KBs), extremely fast.
- **L2 Cache** → Larger (MBs), slower than L1, may be shared between cores.
- **L3 Cache** → Even larger (tens of MBs), shared by all cores, slower than L1/L2 but still faster than RAM.

🔹 Advantages of Cache Memory

1. **High Speed** → Much faster than RAM, reduces CPU waiting time.
2. **Improves Performance** → Programs run faster since frequently used data is quickly available.
3. **Bridges the Gap** → Balances speed difference between CPU and RAM.
4. **Automatic Management** → Modern CPUs handle cache operations automatically.

🔹 Disadvantages of Cache Memory

1. **Expensive** → Costlier than RAM.
2. **Limited Size** → Very small compared to RAM (only MBs).
3. **Complex Design** → Managing multiple cache levels increases CPU complexity.
4. **Stale Data Problem** → Cache may contain outdated data if RAM is updated but cache isn't (solved using cache coherence protocols).
5. **Not Always Useful** → If workload doesn't reuse data (low locality), cache hit rate is low, so benefits reduce.

✅ **In short:** Cache memory is a small, high-speed memory that **bridges the gap between fast CPU and slower RAM**, improving performance.
Its main drawbacks are **cost, limited size, and stale data issues**, which are managed using **coherence protocols**.

🖥️ Cache in Server-Side / Load Balancing

🔹 What is Cache in Server-Side / Load Balancing?

In server systems, **cache** refers to a **temporary storage layer** that keeps copies of frequently used data (like database queries, API responses, images, or static files).

Instead of recomputing or fetching from the database every time, servers fetch from cache → much faster.

Common tools: Redis, Memcached, Varnish, CDN caches.

🔹 Why do we use Cache in Load Balancing?

- **Reduce Server Load** → Less database and backend queries.
- **Faster Response Time** → Users get data quickly from cache instead of waiting for computation.
- **Scalability** → Handle more users without crashing.
- **Reliability** → Cached copies still serve users even if backend is temporarily slow.

🔹 Advantages

- **Speed** → Much faster than database queries.
- **Efficiency** → Reduces load on backend servers.
- **Cost Saving** → Fewer DB calls = reduced infrastructure cost.
- **Better User Experience** → Faster page load and API response.
- **Helps Load Balancers** → Instead of routing every request to the backend, the load balancer can serve from cache.

🔹 Disadvantages

- **Extra Lookup Time** → If the data is not present in cache, the system first checks cache, then server/database → increases response time.
- **Stale Data** → Cache might return outdated data if the database has changed.
- **Consistency Issues** → Hard to keep cache and database perfectly synchronized.
- **Extra Layer of Management** → Need cache eviction policies (LRU, TTL, etc.).
- **Memory Cost** → Requires dedicated memory (RAM), which is expensive.
- **Complexity in Distributed Systems** → Cache coherence and invalidation become tricky when many servers are involved.

✅ That's the **server-side cache in load balancing**: a powerful way to speed up systems, but with trade-offs like stale data and management complexity.
