SYSTEM DESIGN

ST-2

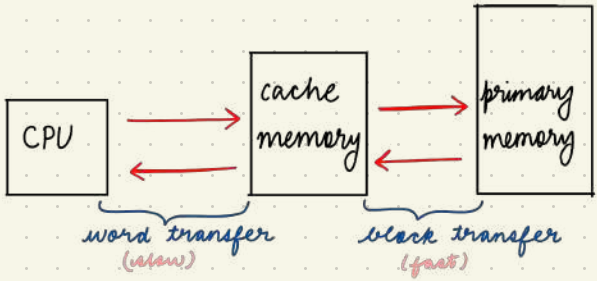
- Cache Memory
- Cache Replacement Policies
- Access control + Authentication
- Secure OS design + implementation
- Malware + defense mechanisms
- DBMS, RDBMS
- SQL vs NoSQL
- SQL Queries

CACHE MEMORY

small, high-speed storage layer close to the CPU

stores frequently accessed data & instructions to reduce the CPU spends accessing main memory.

importance
- speed: cache memory is faster than main memory (RAM)
- efficiency: reduces the average time to access data from the memory hierarchy



Miss Penalty :- Additional time to fetch data from main memory.

Levels of cache
- L1 cache: closest to the CPU, smallest in size but fastest
- L2 cache: larger than L1, slightly slower
- L3 cache:
  - shared among CPU cores
  - larger
  - slower than L2

Average Memory Access Time (AMAT) = Hit Time + (Miss Rate x Miss Penalty)

Cache Hit → occurs when data requested by the CPU is found in the cache memory.

allows the CPU to access the data quickly without needing to fetch it from slower RAM.

Hit Rate = $\frac{\text{Cache Hits}}{\text{Total Memory Accesses}}$

proportion of memory accesses that results in cache hits.

Cache Miss → occurs when the data requested by the CPU is **not found** in the cache memory.

↓

data must be fetched from the main memory

Miss Rate = 1 − Hit Rate

↳ the proportion of memory accesses that result in cache miss

Types of Cache Miss

- **Cold miss** (compulsory miss)
  - ↓ occurs when data is accessed for the **first time** & has not been loaded to the cache yet.
  - inevitable for new data accesses
- **Conflict miss**
  - ↓ happens when multiple memory blocks **compete** for the **same cache line** due to cache mapping
  - can be reduced by increasing associativity
- **Capacity miss**
  - ↳ occurs when the cache **can't hold** all the data needed for a workload, **forcing** older data to be **evicted**.
  - can be reduced by increasing cache size.

Cache Miss Penalty: Time reqd. to bring a missed block from main memory to cache

Types of Cache Access

1. **Parallel Access (Simultaneous Access)**

→ All cache lines are checked simultaneously to find a match (associative caches typically use this method).

→ common in high-performance cache architectures.

→ Faster lookups due to parallel comparison.

2. **Hierarchical Access (Multilevel Access)**

→ The CPU accesses data in a hierarchical manner,

starting from the smallest & fastest memory (L1) & moving to larger, slower memories (L2, L3 & main memory) only if necessary.

when locality of reference is used t_{mm} is replaced by block time (t_{block})

Locality of Reference

→ Principle in computer science that predicts patterns of memory access during program execution.

→ It states that programs tend to access the same set of memory locations repeatedly over a short period, leading to efficient caching & faster execution.

Types of locality →

- Temporal locality
  - if a memory location is accessed, it is likely to be accessed again in the near future.
  - Recently accessed data is stored in fast cache memory, reducing the need to fetch it repeatedly from main memory.

```
int sum=0   → accessed repeatedly
for(int i=0; i<10; i++){
    sum += arr[i]
}
```

- spatial locality
  - if a memory location is accessed, nearby locations are likely to be accessed soon.
  - Prefetching nearby data into cache can improve performance

```
for(int i=0; i<10; i++){
    sum += arr[i]
}
```

(elements are stored in contiguous memory)

- sequential locality → * special case of spatial locality where memory is accessed sequentially.

Types of Mapping

determines how data from main memory is placed into the cache.

it addresses where a particular memory block can reside in the cache & is crucial for efficient memory access.

- Direct Mapping
- Fully-Associative Mapping
- Set-Associative Mapping

Direct Mapping

simple to implement

each block of main memory maps to exactly one cache line

cache line = (block number) % (number of cache lines)

only one cache line is available for each block

disadvantage

high conflict misses

two or more blocks may map to the same cache line.

Fully Associative Mapping

any block from main memory can be placed in any cache line

working

- the cache is searched fully to check if a block is already present.
- A replacement policy determines which block to evict if the cache is full.

disadvantage

- complex hardware required for searching
- slower compared to direct mapping

reduces conflict misses since any block can go anywhere.

Set-Associative Mapping

combines aspects of direct & fully associative mapping by dividing the cache into sets.

working

- A block maps to a specific set, but within that set it can occupy any line.
- The number of lines per set is determined by degree of associativity

Number of sets = cache lines / lines per set

- A replacement policy manages blocks within the set.

CACHE REPLACEMENT POLICIES

→ Cache Replacement Policies determine which cache block is evicted to make room for new data when the cache is full.

Least Recently Used (LRU)

→ Replaces the block that has not been accessed for the longest time.

ਸਭ ਤੋਂ ਪਹਿਲਾ ਸਾਰੇ blocks check ਹੋਣਗੇ, ਜਿਹੜਾ ਸਭ ਤੋਂ ਪੁਰਾਣਾ ਹੋਵੇਗਾ ਉਸ ਦੀ ਜਗ੍ਹਾ ਤੇ ਨਵਾਂ ਆਵੇਗਾ

→ Working: ① tracks the usage history of cache blocks.
② the least recently accessed block is replaced.

→ Advantage: Effectively uses temporal locality
(recently accessed data is more likely to be reused).

→ Disadvantage: ① Requires additional hardware / software to maintain usage history.
② Higher implementation cost for large caches.

Eg. A, B, C, D ese order ch hi already present hai ite kise nve block ne aana hai ta A will be removed. (until A nu replace kraga nve block vote).

First-in-First Out (FIFO)

→ Replaces the **oldest block** in the cache (regardless of usage).

ਜਿਹੜਾ ਸਭ ਤੋਂ ਪਹਿਲਾ ਆਇਆ ਸੀ, ਉਹ replace ਹੋ ਜਾਏਗਾ

→ Working: ① Maintains a queue of cache blocks.

② The block at the front of the queue (added first) is replaced.

→ Advantages: ① simple to implement

② No need to track user history.

→ Disadvantages: Potential inefficiences (∵ it doesn't consider the frequency or recency of block access)

Eg. A, B, C, D ese order ch hi aaya si saare blocks ta jd nva block aayega ta A evict ho jayega kyuki oh sb to pehla aaya si.

Least Frequently used (LFU)

→ Replaces the block that has been accessed the **least number of times**.

ਜਿਹੜਾ ਸਭ ਤੋਂ ਘੱਟ ਵਾਰ use ਹੋਇਆ ਹੈ, ਉਹ replace ਹੋ ਜਾਏਗਾ

→ Working: ① Tracks the number of accesses for each block.

② Evicts the block with the smallest count.

→ Advantage: uses frequency data to keep popular blocks longer.

→ Disadvantage: **increases complexity** (∵ it requires counter per each block).

Eg. A, B, C, D nve block vich C aayega replace kyuki usda access counts ghat hai.
access counts 10, 20, 5, 15

Random Replacement

→ replaces a randomly selected block.

ਕੋਈ ਵੀ ਚੱਕ ਦੇ replace ਹੋ ਜਾਏਗਾ

→ Working: selects a block at random for eviction without considering usage or age.

→ Advantages → simple to implement

→ requires no tracking of usage or age

→ Disadvantage: can lead to inefficient caching if important blocks are evicted.

Most Recently Used (MRV)

→ Replaces the block that was most recently accessed.

ਜਿਹੜਾ ਸਭ ਤੋਂ latest use ਹੋਇਆ ਹੋਵੇਗਾ ਉਹ replace ਹੋ ਜਾਏਗਾ

→ Working: assumes that the most recently accessed block is less likely to be reused soon.

→ Advantage: effective for certain workloads with patterns of one time use.

→ Disadvantage: Poor performance in cases where temporal locality is significant.

ਮੰਨ A, B, C, D ਪੂਰੇ ਪੂਰੇ use ਹੋਇਆ ਹੈ ਤਾਂ ਇਹ replace ਹੋਵੇਗਾ ਨਵੇਂ block ਦਾਸਤੇ

Optimal Replacement Policy (OPT)

→ Replaces the block that will not be used for the longest time in future.

→ Produces the lowest miss rate. advantage

→ disadvantage: not feasible for actual systems, as future access patterns are unknown.

ACCESS CONTROL & AUTHENTICATION

• **Access control** in an operating system refers to the mechanisms used to regulate & restrict access to resources like files, directories, devices & services based on user identity or roles.

Access control lists (ACLs)

→ Each file or resource has a list defining which users or groups can access it & what operations (read, write, execute) they can perform.

Eg. User: Alice - Read, Write
User: Bob - Read

Role-based Access Control (RBAC) simplifies management in environments with many users.

→ Users are assigned roles, and roles have predefined permissions.

ਹਰ ਇੱਕ user ਨੂੰਨ ਉਸਦਾ role ਹੋਏਗੀ ਅਤੇ ਉਸ ਦੀ permission (ਜੋ ਪਹਿਲਾ ਹੀ defined ਹੋਏਗੀ)

Eg. Role: "admin" - full access
Role: "guest" - read only access

Discretionary Access Control (DAC)

→ Owners of the resources define who can access them.

Eg. File permissions in Unix (chmod, chown)

Mandatory access control (MAC)

→ access is controlled by strict policies defined by the system, not user.

common in highly secure systems (eg. military-grade systems)

Capabilities

→ fine grained access rights given to processes instead of users.

eg.
allowing a process to open a network socket without full root privileges

Authentication verifies the identity of a user or entity trying to access the system.

Password-based authentication

→ Users provide a username or password to login.

→ stored as hashed values for security.

Two factor authentication (2FA)

→ combines something you know (password) with something you have (OTP or biometrics).

Biometric authentication

→ uses physical characteristics like fingerprints, facial recognition or iris scan.

Certificate-based authentication

→ relies on digital certificates issued by trusted authorities.

Kerberos authentication

→ a network authentication protocol using secret-key cryptography.

→ Issues tickets to users for accessing resources.

Single Sign-On (SSO)

→ Users authenticate once & gain access to multiple systems or resources.

often used in enterprise environments

SECURE OS DESIGN & IMPLEMENTATION

→ Focuses on building systems that are resilient to malicious attacks, unauthorized access & vulnerabilities.

→ It emphasizes confidentiality, integrity & availability (CIA triad) of data & resources.

Core components

- Authentication Mechanisms
- Access Control Mechanisms
  - DAC: User controls access to resources they own
  - MAC: The system enforces strict access policies
  - RBAC: Permissions are assigned based on user roles.
- Memory Protection
  - prevent processes from accessing each other's memory (eg. virtual memory)
  - implement Address Space Layout Randomization (ASLR) to mitigate buffer overflow attacks.
- Secure Boot & Firmware
  - verify the integrity of the OS during startup using secure boot processes.
  - ensure firmware is signed & verified before execution.
- Process Isolation: use sandboxing techniques to isolate applications & reduce the impact of a compromised process.
- Encryption
  - encrypt data at rest & in transit to protect against unauthorized access.
  - use secure algorithms & manage keys effectively.
- Auditing & Logging
  - maintain logs of system activities for monitoring & forensics.
  - implement tamper-proof logging mechanisms.
- Secure Update Mechanisms
  - use digitally signed updates to prevent malicious code injection.
  - ensure updates are applied automatically or with user consent.
- Kernel Security
  - ↓
  - Harden the kernel against vulnerabilities by reducing attack surfaces. (minimize unnecessary features & modules)

MALWARE & DEFENSE MECHANISMS

Malware (malicious software) refers to any software intentionally designed to harm, exploit or compromise systems, devices or networks.

Defense mechanisms are strategies & tools used to detect, prevent & mitigate the impact of malware

Types of malware

Viruses → attaches to legitimate programs or files & spreads when executed.

Worms → standalone programs that spread automatically across networks.

Trojans	Rootkits
Ransomware	Botnets
Spyware	Fileless malware
Adware	Logic bombs

Defense Mechanisms

1. Prevention mechanisms

Aim to stop malware from infiltrating systems.

1.1 Antivirus software 1.2 Firewalls 1.3 Secure coding Practices

1.4 Access control 1.5 Patch management 1.6 Email filtering

1.7 Endpoint Protection platform (EPP)

2. Detection mechanisms

Identify malware that has bypassed preventive measures.

2.1 Intrusion detection system (IDS) 2.2 Behavioral Analysis

2.3 Heuristic analysis 2.4 Sandboxing

3. Reactive mechanisms

Mitigate damage & remove malware after detection.

3.1 Incident response 3.2 Data backups 1.3 Threat intelligence

3.4 Forensics tools.

4. Advanced Defense Mechanisms

Cutting-edge approaches for sophisticated malware

4.1 AI and machine learning 4.2 Deception Techniques

4.3 Zero trust architecture 4.4 endpoint detection & response (EDR)

4.5 Cloud-based security.

INTRODUCTION TO DBMS & RDBMS

DBMS

(database management system)

→ stores data as files or collections of data.

→ doesnot inherently support relationships b/w data.

→ No structured schema

→ May not uses normalization techniques.

→ limited query mechanisms

→ Higher chances of data redundancy.

→ Basic or no support for transactions.

→ Suitable for smaller datasets.

RDBMS

(relational database management system)

database is a collection of related data.

database
- structured → RDBMS, IRCTC, University
- unstructured → web pages

→ stores data in structured tables with rows and columns

→ Uses primary + foreign keys to define relationships b/w tables.

→ well defined schema.

→ supports normalization to reduce redundancy + improve consistency.

redundancy: state of being no longer useful

→ Uses SQL for data manipulation or queries.

→ Minimizes redundancy through normalization.

→ Provides ACID properties for transactions.

ACID: Atomicity, Consistency, Isolation, Durability

→ Ideal for large-scale, complex datasets.

SQL vs NoSQL

SQL
(structured query language)

- → Relational database management system.
- → fixed or static or predefined schema.
- → not suited for hierarchical data storage.
- → best suited for complex queries.
- → follows ACID property
  atomicity, consistency, Isolation, Durability.
- → Vertically scalable

eg MySQL, PostgreSQL, Oracle, MS-SQL Server

NoSQL
(Non-structured Query language)

- → Non-relational or distributed
- → Dynamic schema
- → suited for hierarchical data storage.
- → not so good for complex queries.
- → Follow CAP
  consistency, availability, partition tolerance

- → Horizontally scalable.

eg. mongoDB, Hbase, Cassandra

SQL QUERIES

→ categorized into different types based on its functionality.

DDL (data definition language)

- used to define, modify, delete the structure of databases objects such as tables, schemas, indexes and views.
- affect the schema of the database
- CREATE
- ALTER
- DROP
- TRUNCATE

DML (data manipulation language)

- to retrieve, insert, update & delete data from database tables.
- deal with the actual data stored in the database
- INSERT
- SELECT
- UPDATE
- DELETE

DCL (data control language)

- control access to the databases by managing user permissions.
- deal with security & access control
- GRANT
- REVOKE

TCL (Transaction control language)

- to manage transactions in a database
- COMMIT
- ROLLBACK
- SAVEPOINT
- SET TRANSACTION

DQL (data query language) → SELECT

↓ focuses exclusively on querying data from database.

→ categorized as a part of DML, but it is specifically used to retrieve data, making it a distinct subset.

SQL DATA TYPES

signed → allows both +ve & -ve numbers.

→ TINYINT SIGNED (-128 to 127)

↓ TINYINT (-128 to 127)

unsigned → only non negative numbers (0 & +ves)

→ increases the maximum range because it doesn't allocate space for -ve nos.

↓ TINYINT UNSIGNED (0 to 255)