Memory Hierarchy and Technologies

Srinivasan Ramachandran

April 8, 2025

Memory Hierarchy

Why Hierarchy?

- Trade-off between speed, cost, and size
- Faster memory is more expensive per byte
- Larger memory is slower to access

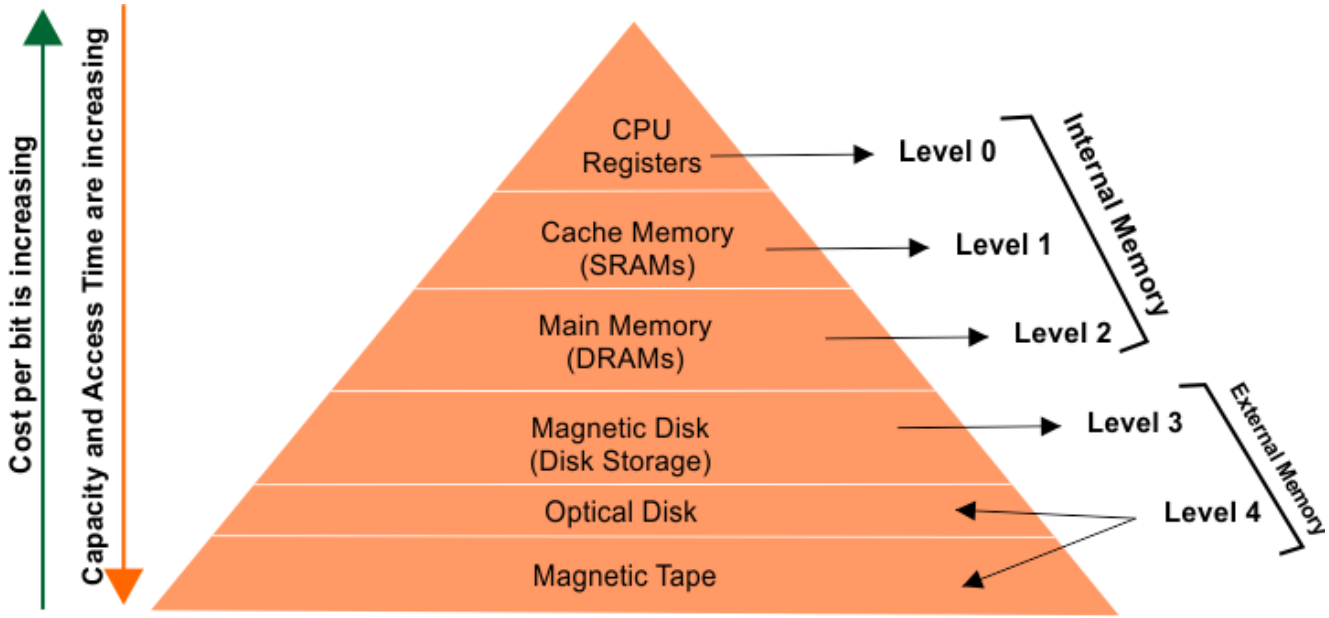
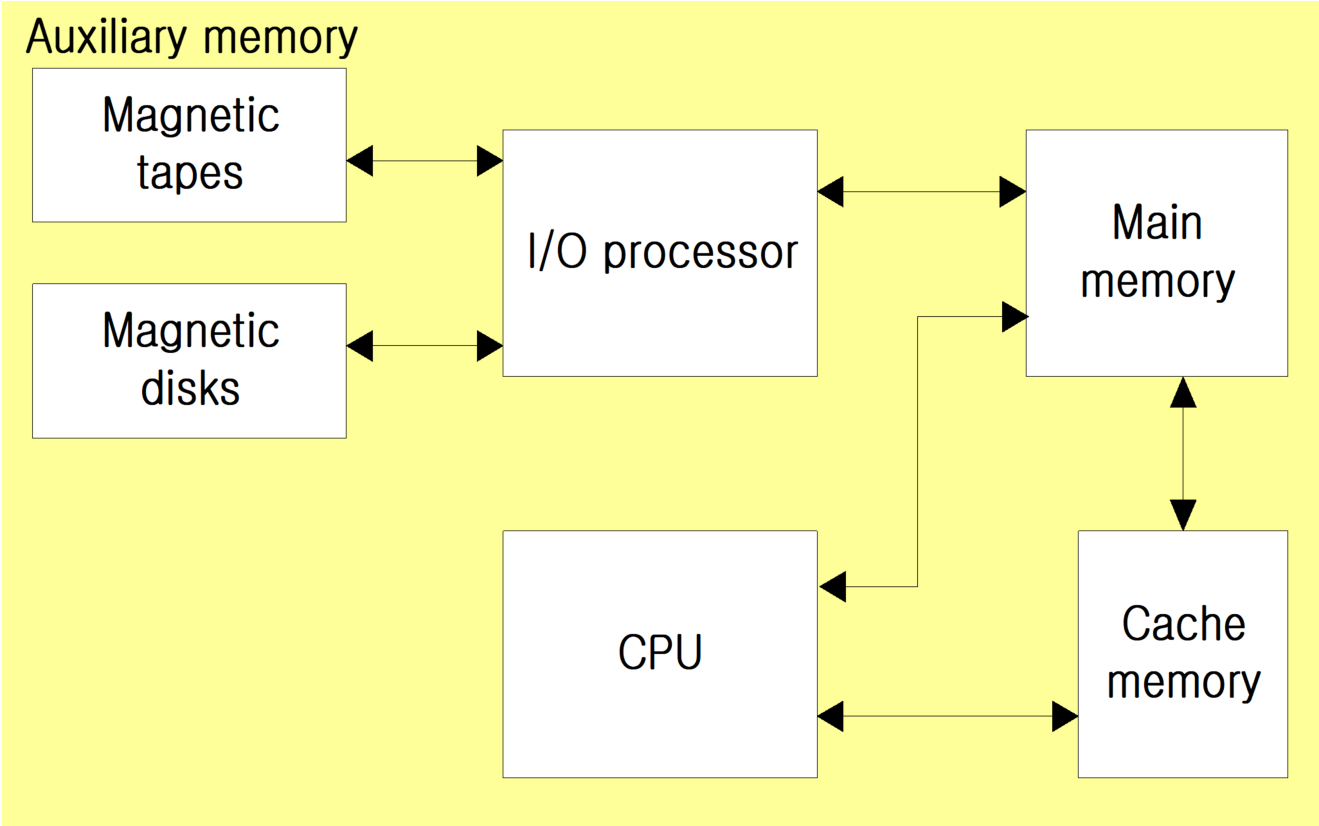


Figure: Typical Memory Hierarchy

Memory Organization

- Categories
  - Main Memory : memory unit that communicates directly with the CPU (RAM)
  - Auxiliary Memory : device that provide backup storage (Disk Drives)
  - Cache Memory : special very-high-speed memory to increase the processing speed (Cache RAM)
- Multiprogramming: Enable the CPU to process a number of independent program concurrently
- Memory Management System: Supervise the flow of information between auxiliary memory and main memory

How Everything Connects



Bootstrap Loader

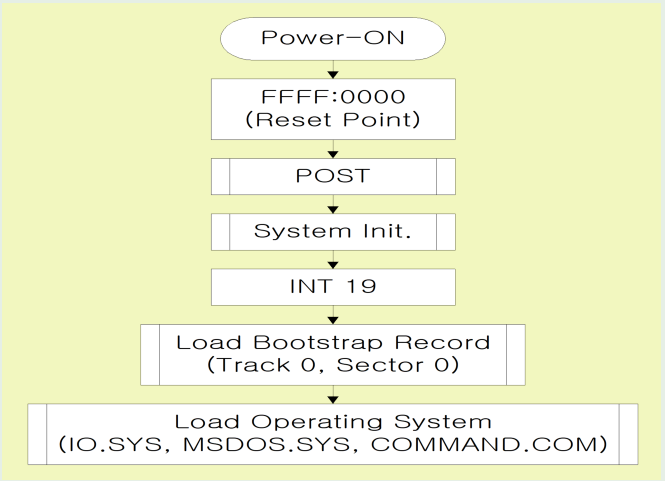
Definition

The **bootstrap loader** is a small program that initializes the operating system when a computer starts up.

- Also known as **boot loader** or **bootstrapper**
- Typically stored in **ROM** or **EEPROM** (BIOS/UEFI)
- First code executed when computer powers on

Boot Process

1. Power-on self-test (POST)
2. Bootstrap loader executes
3. Loads and verifies boot sector
4. Transfers control to OS kernel



RAM & ROM Chip Organization: Address Line Perspective

Fundamental Relationship

$$\text{Memory Locations} = 2^{\text{Number of Address Lines}}$$

- Each address line represents 1 bit in the memory address
- Combination of address lines selects unique memory location

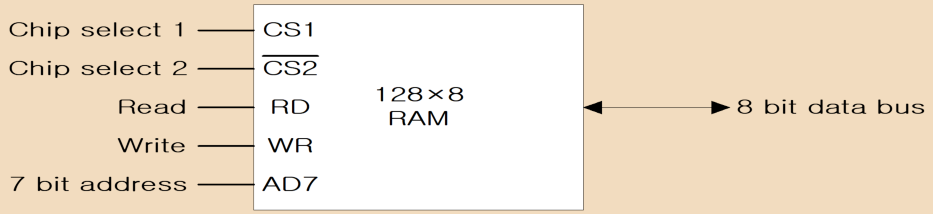
Addressing Fundamentals

$$\text{Memory Capacity} = 2^{\text{Address Lines}} \times \text{Data Bus Width}$$

RAM & ROM Chip Organization: Address Line Perspective

RAM Chip Example

- **Address Lines**: 7 (A0-A6)
- **Addressable Locations**: $2^7 = 128$
- **Typical Data Bus Width**: 8-bit
- **Total Capacity**:
  - $128 \times 8 = 1024$ bits
  - 128 bytes (0.125 KB)
- **Applications**:
  - Small SRAM caches
  - Register banks
  - I/O address spaces



(a) Block diagram

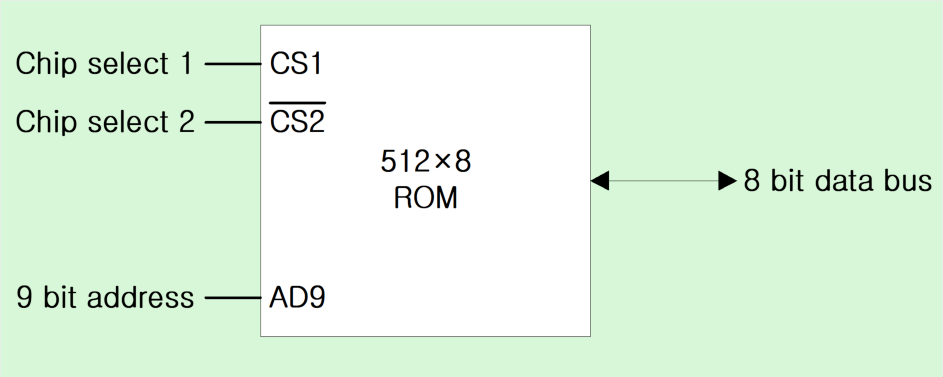
CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

(b) Function table

RAM & ROM Chip Organization: Address Line Perspective

ROM Chip Example

- **Address Lines**: 9 (A0-A8)
- **Addressable Locations**: $2^9 = 512$
- **Typical Data Bus**: 8-bit
- **Total Capacity**:
  - $512 \times 8 = 4096$ bits
  - 512 bytes (0.5 KB)
- **Applications**:
  - Microcontroller firmware
  - Bootloaders
  - Font/character ROMs



Primary Memory Types

RAM (Random Access Memory)

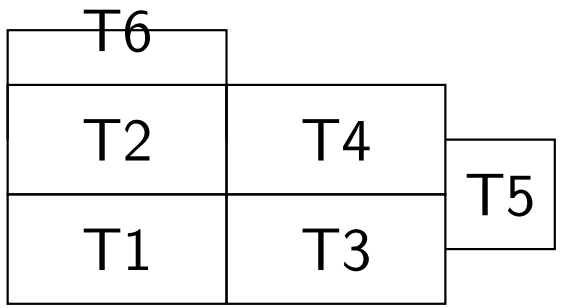
- SRAM (Static RAM)
  - Faster, more expensive
  - Used in cache memory
- DRAM (Dynamic RAM)
  - Needs refresh, higher density
  - Main memory

ROM (Read Only Memory)

- PROM (Programmable)
- EPROM (Erasable)
- EEPROM (Electrically Erasable)
- Flash Memory

Static RAM (SRAM)

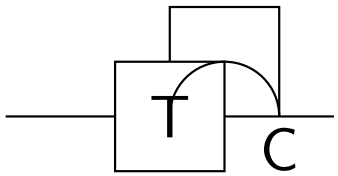
- **Volatile** memory (loses data when power off)
- Uses 6 transistors per bit (flip-flop)
- Faster but more expensive than DRAM
- No refresh needed (static)
- Used for cache memory
- Typical access time: 10-100 ns



6T SRAM Cell

Dynamic RAM (DRAM)

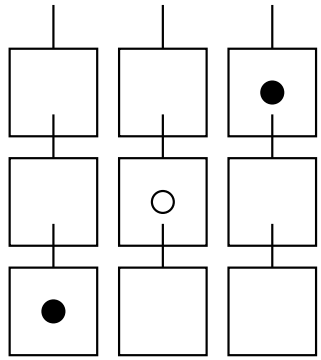
- **Volatile** memory
- Uses 1 transistor + 1 capacitor per bit
- Requires periodic refresh (dynamic)
- Higher density, lower cost than SRAM
- Used for main system memory
- Typical access time: 50-100 ns



1T1C DRAM Cell

Programmable ROM (PROM)

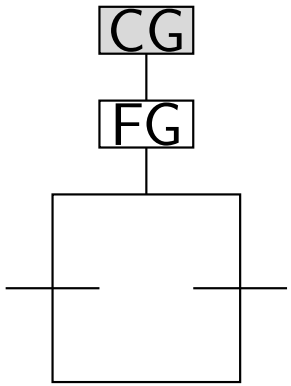
- **Non-volatile** memory
- Programmed once by blowing fuses
- No erase capability
- Cheaper than other ROMs for mass production
- OTP (One-Time Programmable)
- Used for permanent programs



PROM with fusible links

Erasable PROM (EPROM)

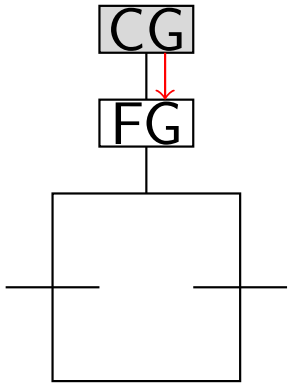
- **Non-volatile**, reprogrammable memory
- Erased by UV light exposure
- Uses floating gate transistors
- Must be removed from circuit for erasure
- Typical endurance: 100-1000 cycles
- Used in older BIOS chips



Floating Gate Transistor

Electrically Erasable PROM (EEPROM)

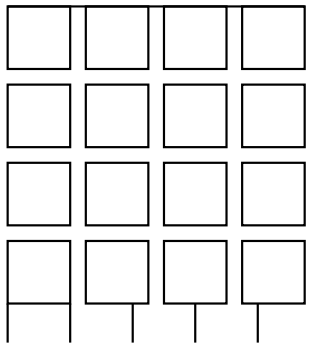
- **Non-volatile**, electrically erasable
- Byte-level erase capability
- Uses Fowler-Nordheim tunneling
- Higher endurance than EPROM
- Slower write than read
- Used for configuration storage



EEPROM with tunneling

Flash Memory

- **Non-volatile**, block-erase memory
- Higher density than EEPROM
- Two main types: NOR and NAND
- Used in SSDs, USB drives, memory cards
- Typical endurance: 10K-100K cycles
- Faster access than hard disks



NAND Flash Memory Array

Practice Problem 1

Problem

A computer system has the following memory specifications:

- L1 cache: 64 KB, 1 ns access time
- L2 cache: 512 KB, 3 ns access time
- Main memory: 8 GB, 60 ns access time

If the hit rates are 90% for L1 and 95% for L2, what is the average memory access time?

Practice Problem 1

Problem

A computer system has the following memory specifications:

- L1 cache: 64 KB, 1 ns access time
- L2 cache: 512 KB, 3 ns access time
- Main memory: 8 GB, 60 ns access time

If the hit rates are 90% for L1 and 95% for L2, what is the average memory access time?

Solution

$\text{AMAT} = \text{HitTime\_L1} + \text{MissRate\_L1} \times (\text{HitTime\_L2} + \text{MissRate\_L2} \times \text{MissPenalty\_Main})$

$\text{AMAT} = 1 + 0.1 \times (3 + 0.05 \times 60) = 1 + 0.1 \times (3 + 3) = 1.6 \text{ ns}$

Cache Memory Principles

Principle of Locality

- Temporal Locality: Recently accessed items likely to be accessed again
- Spatial Locality: Items near referenced items likely to be accessed

Cache Mapping Techniques

- Direct Mapped
- Fully Associative
- Set Associative (n-way)

WORD

Fundamental Data Unit in Computing

Definition

The **WORD** is the native data size a CPU processes in a single operation. Its size is architecture-dependent:

- 16-bit CPU: 2 bytes (e.g., Intel 8086)
- 32-bit CPU: 4 bytes (e.g., ARM Cortex-M)
- 64-bit CPU: 8 bytes (e.g., x86-64)

Key Impact

- Determines **register size** (e.g., EAX in x86)
- Affects **memory addressing limits** (32-bit = 4GB)
- Influences **ALU operation efficiency**

Direct Mapped Cache

Simplest Cache Mapping Technique

Principle

Each memory block maps to **exactly one cache line**:

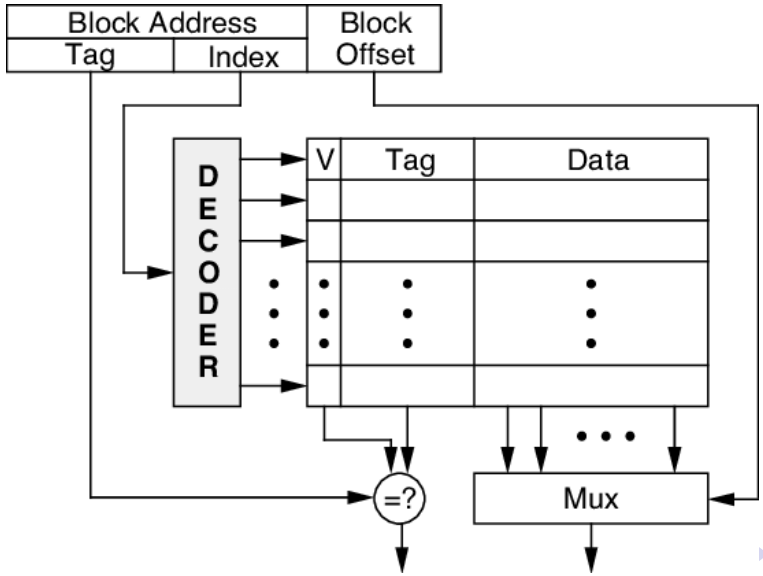
$$\text{Cache Line} = \text{Memory Address} \mod (\text{Number of Cache Lines})$$

Advantages:

- Fast lookup (no search)
- Simple hardware

Disadvantages:

- High conflict misses
- Poor utilization



Fully Associative Cache

Maximum Flexibility, Higher Complexity

Principle

Any memory block can be placed in **any cache line**. Requires:

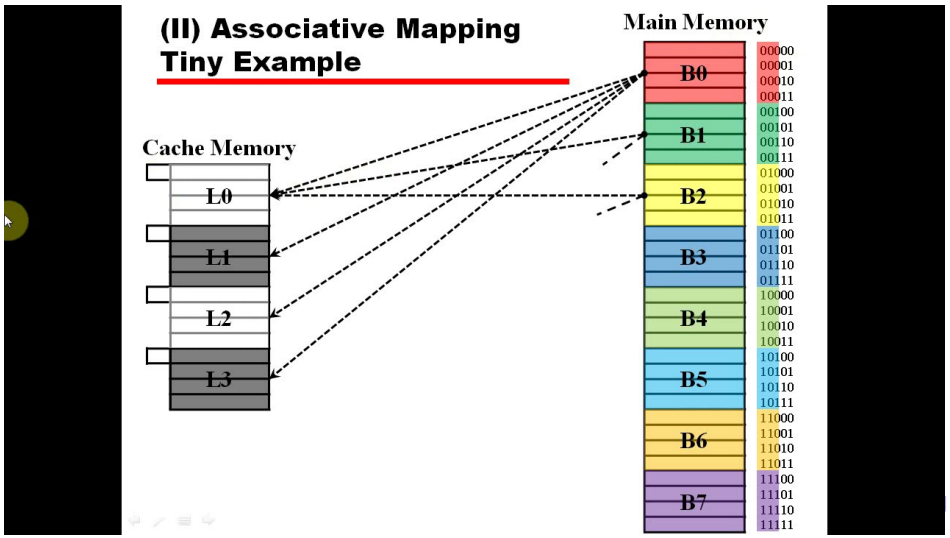
- Parallel search (CAM tags)
- Replacement policies (LRU, FIFO)

Advantages:

- Zero conflict misses
- Optimal utilization

Disadvantages:

- Slow (hardware-intensive)
- Power-hungry



Set-Associative Cache (n-way)

Balancing Speed and Flexibility

Principle

Hybrid of direct-mapped and fully associative:

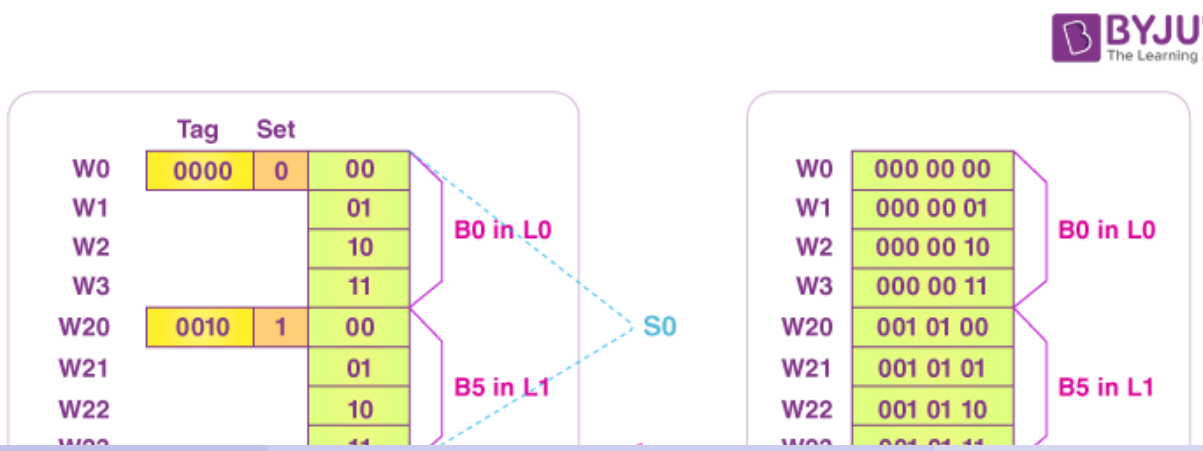
- Cache divided into **sets** (direct-mapped)
- Each set has **n lines** (fully associative within set)

Advantages:

- Fewer conflicts than direct-mapped
- Faster than fully associative

Disadvantages:

- Complex replacement policy
- Higher latency than direct-mapped



Cache Performance

Key Metrics

- Hit Rate = Hits / (Hits + Misses)
- Miss Rate = 1 - Hit Rate
- Miss Penalty: Time to handle a miss

Write Policies

- Write-through
- Write-back
- Write allocate
- No-write allocate

Cache Coherence

Protocols: MESI, MOESI, Directory-based

Practice Problem 2

Problem

Consider a 2-way set associative cache with 4 sets and block size of 2 words. Main memory has 32 words. Show the cache contents after the following word accesses (in order):
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3
Assume LRU replacement policy.

Practice Problem 2

Problem

Consider a 2-way set associative cache with 4 sets and block size of 2 words. Main memory has 32 words. Show the cache contents after the following word accesses (in order):
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3
Assume LRU replacement policy.

Solution Steps

1. Calculate block address = word address / block size
2. Calculate set index = block address % number of sets
3. For each access, determine hit/miss and update cache

Step-by-Step Access Pattern

Access	Address	(Tag,Set)	Cache Action	Set 0 Contents	Replacement
0	(00,00)	Miss	Load (00,00)	[00:0,1]	-
1	(00,00)	Hit	-	[00:0,1]	-
2	(00,01)	Miss	Load (00,01)	[00:0,1]	-
3	(00,01)	Hit	- [00:2,3]	-	-
...	...	...	...	...	...
8	(10,00)	Miss	Replace LRU	[10:8,9], [00:0,1]	Way 1
9	(10,00)	Hit	-	[10:8,9]	-
...	...	...	...	...	...
0	(00,00)	Miss	Replace LRU	[00:0,1], [10:8,9]	Way 0
1	(00,00)	Hit	-	[00:0,1]	-

Replacement Logic

- Initially all sets empty
- When set full, LRU block is replaced
- Final access of 0 replaces newer block (8,9) because Way 0 was LRU

Memory Address Map

- Maps memory addresses to physical devices
- Example for 16-bit address bus:
  - 0000-7FFF: RAM (32KB)
  - 8000-8FFF: ROM (4KB)
  - C000-FFFF: I/O devices

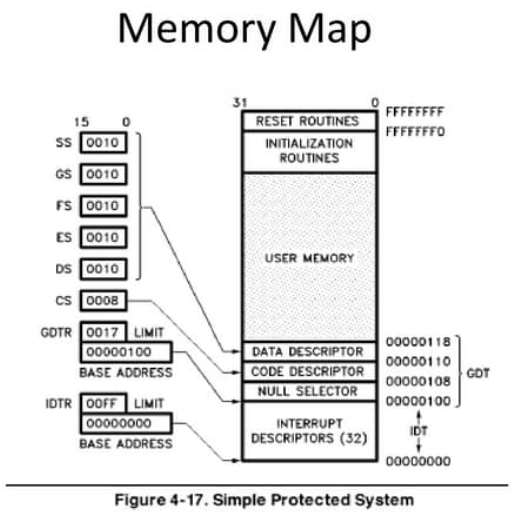
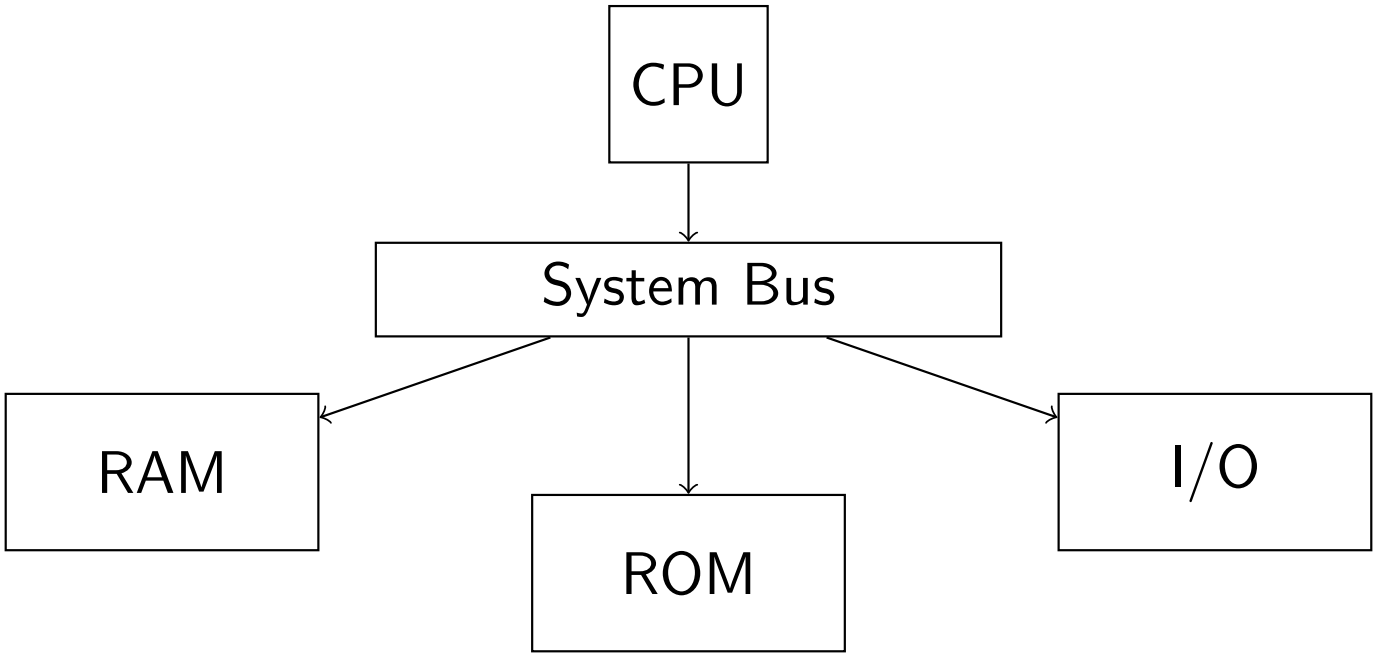


Figure: Memory Address Map Example

Memory Connection to CPU



- Address bus selects memory location
- Data bus transfers data
- Control bus manages operations (read/write)

Emerging Non-Volatile Memories

ReRAM (Resistive RAM)

- Changes resistance to store data
- High density, low power

PCM (Phase Change Memory)

- Uses material phase (crystalline/amorphous)
- Fast, high endurance

STT-RAM (Spin-Transfer Torque RAM)

- Uses electron spin orientation
- High speed, unlimited endurance

Comparison

- Speed: STT-RAM $>$ PCM $>$ ReRAM
- Density: ReRAM $>$ PCM $>$ STT-RAM

Associative Memory

Content-Addressable Memory (CAM)

Key Concept

Associative memory (or **CAM**) retrieves data by content rather than by address (unlike RAM).

- **Search Mechanism:** Parallel comparison of input tag with all stored tags.
- **Output:** Returns matching data (or address) in **constant time** (O(1)).

Analogy

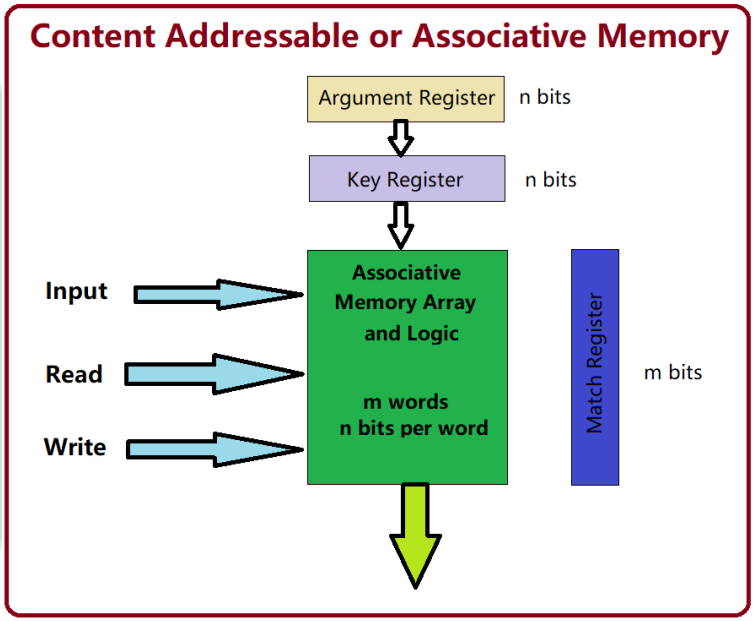
Like a human brain recalling information by association (e.g., "What is the capital of France?" $\rightarrow$ "Paris").

How Associative Memory Works

Hardware Implementation

Components

- **Memory Array**: Stores data + tags.
- **Comparator Circuitry**: Checks input tag against all entries.
- **Match Logic**: Flags matching locations.



Key Feature

Parallel Search: All entries are checked simultaneously (hardware-intensive but fast).

Types of Associative Memory

Classification by Matching Behavior

1. **Exact-Match CAM**:
   - Returns data only if input **exactly matches** a stored tag.
   - Used in CPU caches, network routers.
2. **Ternary CAM (TCAM)**:
   - Supports wildcards (e.g., "10XX" matches "1000", "1011").
   - Critical for **IP routing tables** (longest prefix match).

Advantages & Disadvantages

Why Use (or Avoid) Associative Memory?

Advantages

- Ultra-fast lookups (O(1) latency).
- No address management needed.
- Scalable for databases, AI.

Disadvantages

- High power consumption (parallel circuits).
- Expensive hardware (more transistors than RAM).
- Limited capacity vs. DRAM.

Trade-off

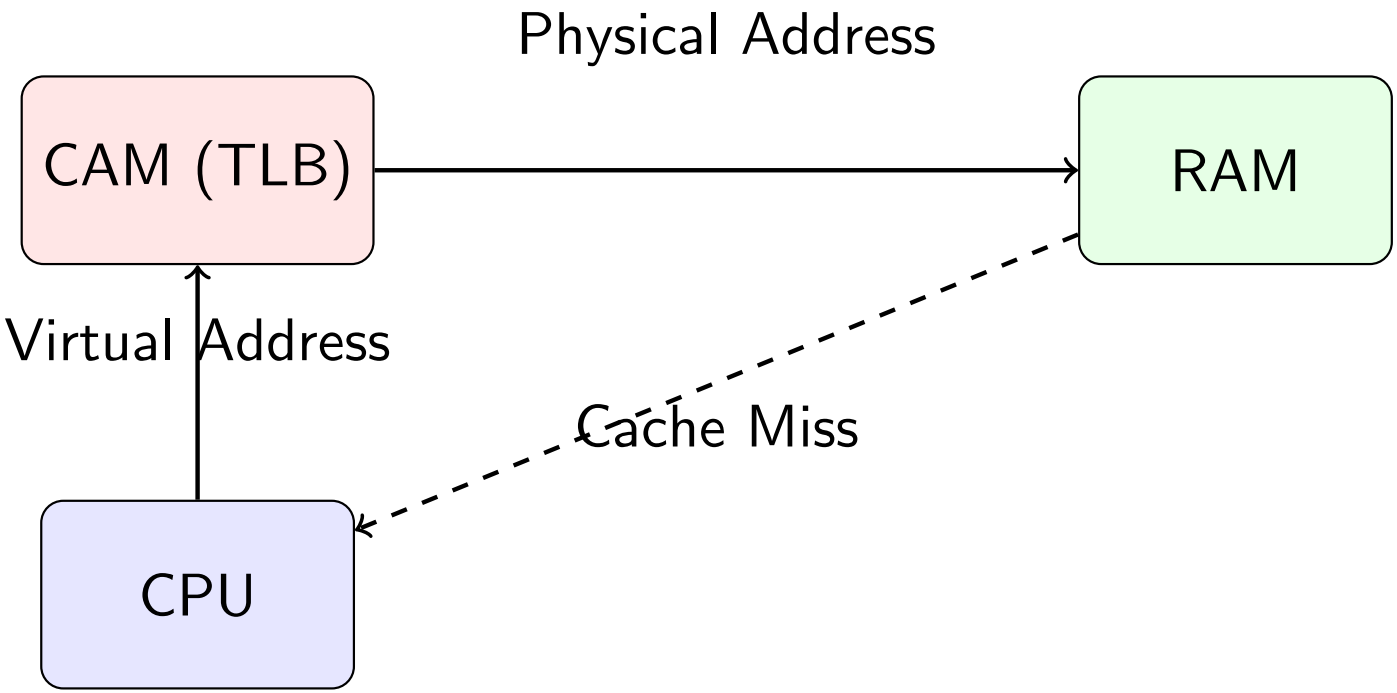
Speed comes at the cost of **area and power**. Used only where latency is critical (e.g., L1 caches).

Applications of Associative Memory

Real-World Use Cases

- **CPU Caches**: TLB (Translation Lookaside Buffer) uses CAM for fast virtual-to-physical address translation.
- **Networking**: TCAMs in routers/switches for packet forwarding.
- **AI/ML**: Nearest-neighbor search in vector databases.
- **Database Engines**: Hash-join acceleration.

How CAM and RAM Work Together



Key Idea

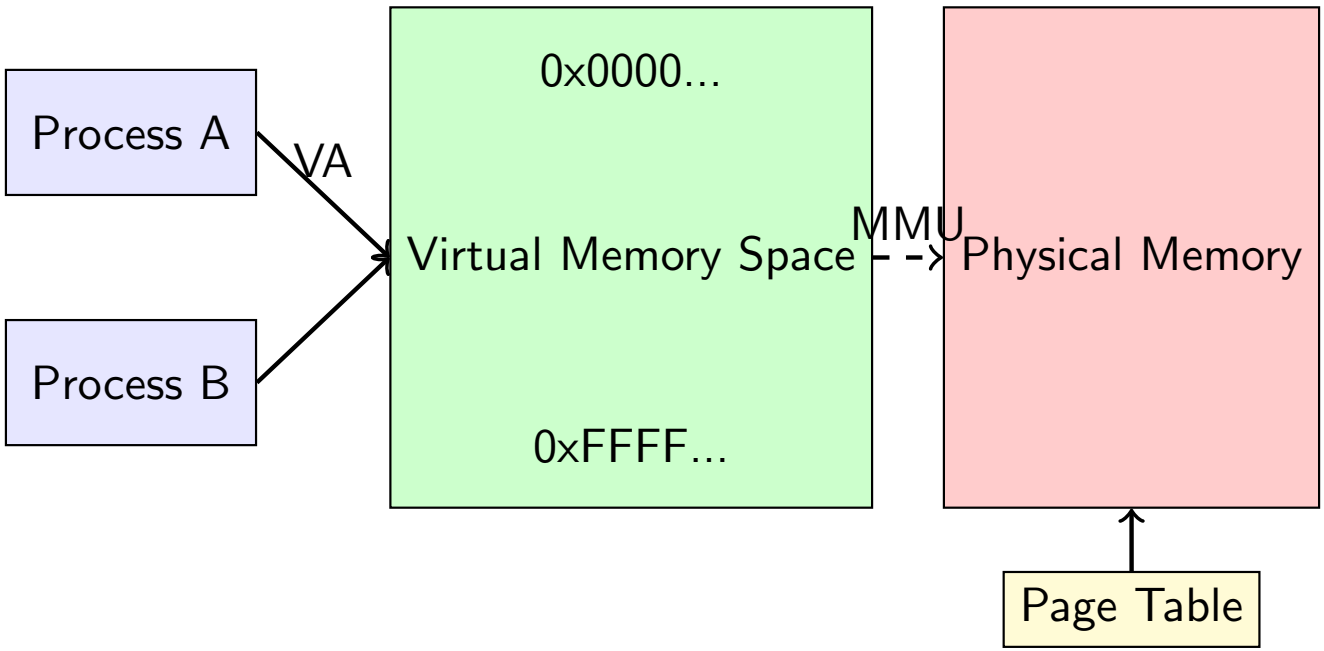
CAM acts as a *gatekeeper* to RAM, reducing access latency for frequent lookups.

Virtual Memory Concept

Abstraction Layer Between Programs and Physical Memory

Key Features

- Each process operates in its own virtual address space
- Memory divided into fixed-size pages (typically 4KB)
- OS manages mapping between virtual and physical memory

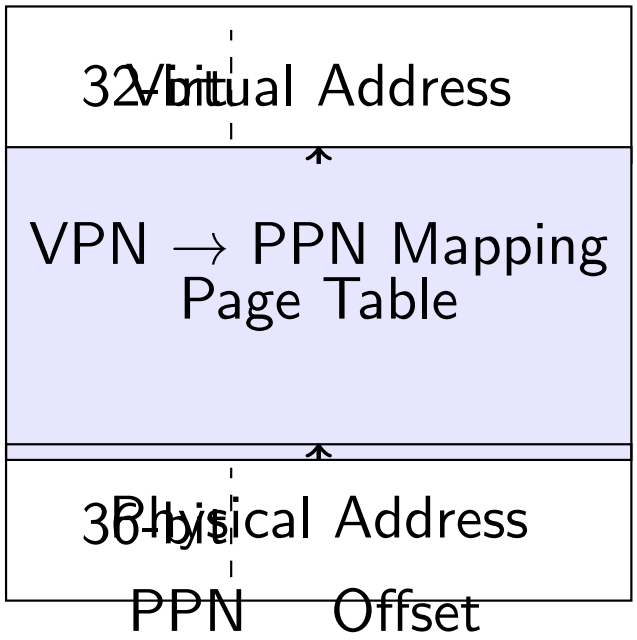


Address Translation

How Virtual Addresses Become Physical Addresses

Translation Steps

1. Split VA into:
   - VPN (Virtual Page Number)
   - Offset
2. Lookup VPN in page table
3. Get PPN (Physical Page Number)
4. Combine PPN + offset



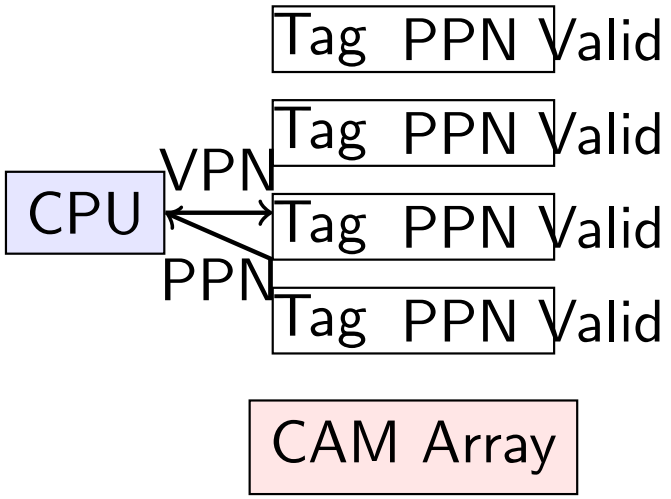
Translation Lookaside Buffer (TLB)

CAM-Based Page Table Cache

- TLB is a special cache for page table entries
- Uses Content-Addressable Memory (CAM)
- Typical sizes: 64-512 entries
- Hit rate ¿ 95% in most systems

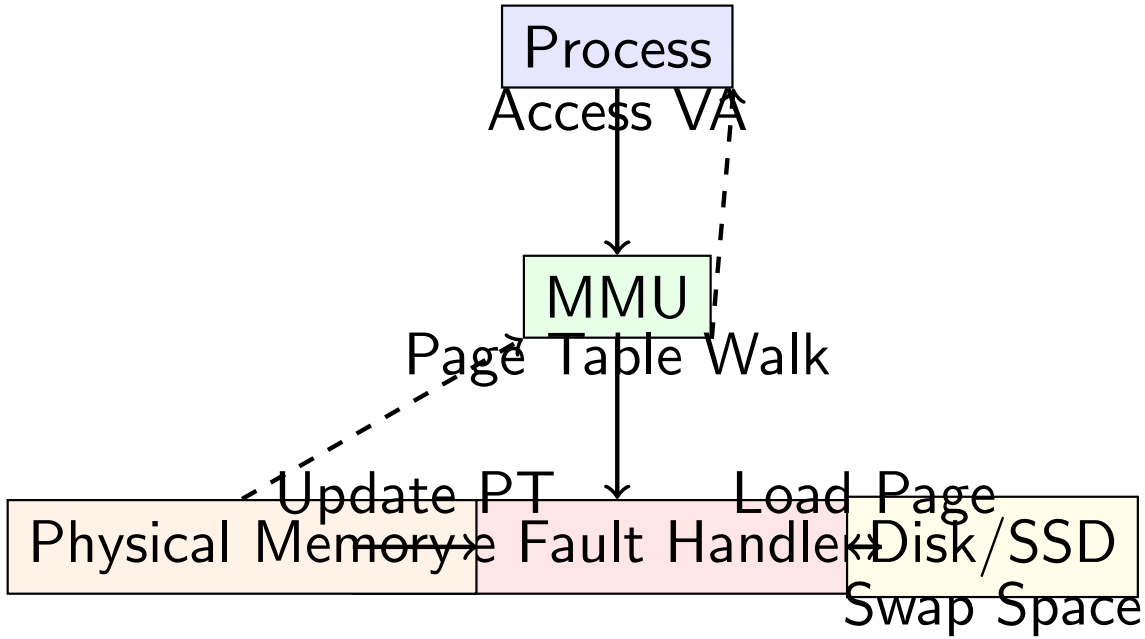
TLB Operation

- Parallel search of all entries
- Returns physical page number in 1 cycle if hit
- On miss: Page table walk (expensive)



Page Fault Handling

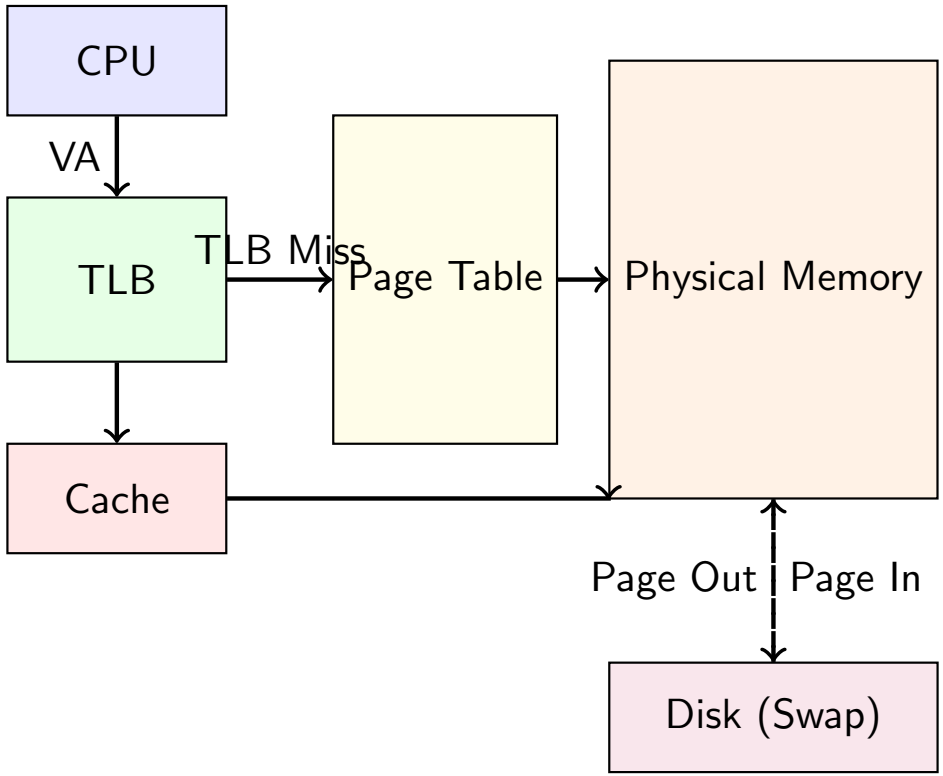
When Data Isn't in Physical Memory



Page Fault Steps

1. OS selects victim page (if memory full)
2. Writes dirty pages to disk
3. Loads requested page from disk
4. Updates page table

Complete Virtual Memory System



Solid: Fast Path Dashed: Slow Path

Key Takeaways

- Memory hierarchy balances speed, cost, and size
- Cache memory exploits locality for performance
- Different mapping techniques affect hit rates
- Emerging memories offer non-volatile alternatives
- Associative memory enables fast searches

Final Practice Problem

A system has a main memory access time of 100ns and a cache access time of 5ns. If the cache hit rate is 95%, what is the effective access time? What hit rate would be needed to achieve an effective access time of 10ns?