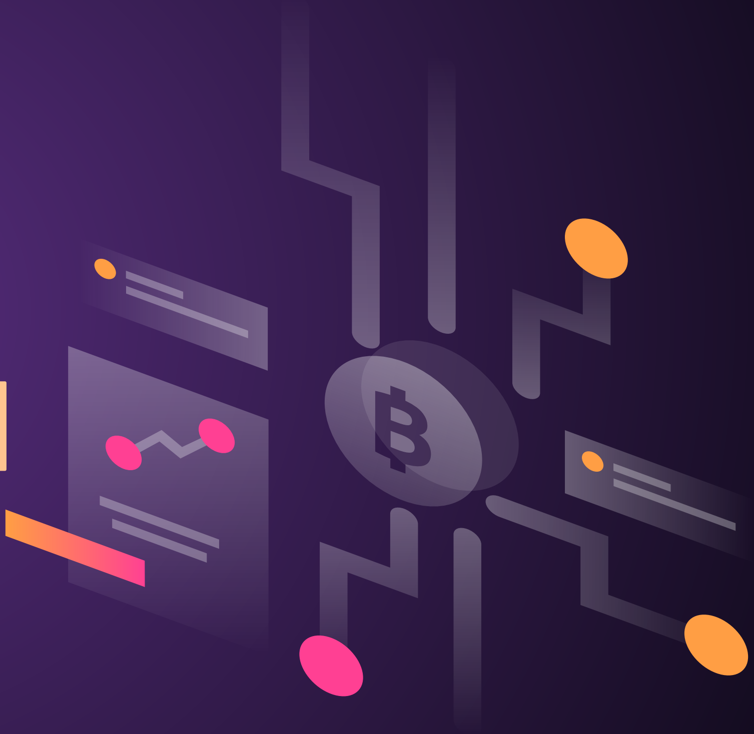
MEMORY ORGANIZATION



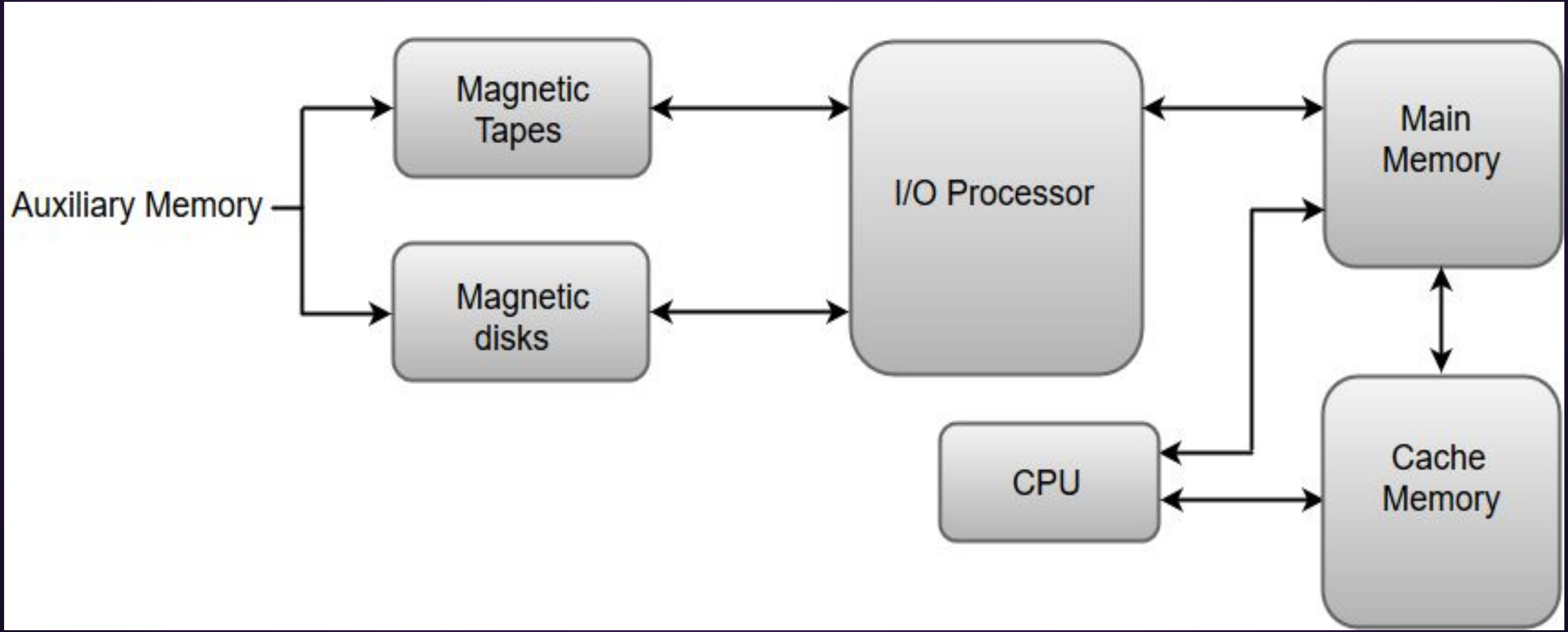
MEMORY HIERARCHY

MEMORY HIERARCHY

- Memory Unit : Essential component in computers
- **Main Memory:** The unit that communicates directly with the CPU
- Programs currently needed by processor reside in main memory
- **Auxiliary Memory:** Devices that provide backup storage.
  Ex: Magnetic disks, tapes

MEMORY HIERARCHY

- Cache Memory: High-speed memory used to increase the processing speed.



MEMORY HIERARCHY

Cache Memory	Auxiliary Memory
Stores data that are currently being used by CPU	**Stores Data which is not currently used by CPU**
Small in size	Large storage capacity
Expensive	Inexpensive
High Access speed	Low access speed
CPU has direct access	CPU does not have direct access

MAIN MEMORY

MAIN MEMORY

- Large and fast memory
- Technology used: Semiconductor Integrated Circuits
- RAM (Random Access Memory) chips can operate in two modes: Static and Dynamic

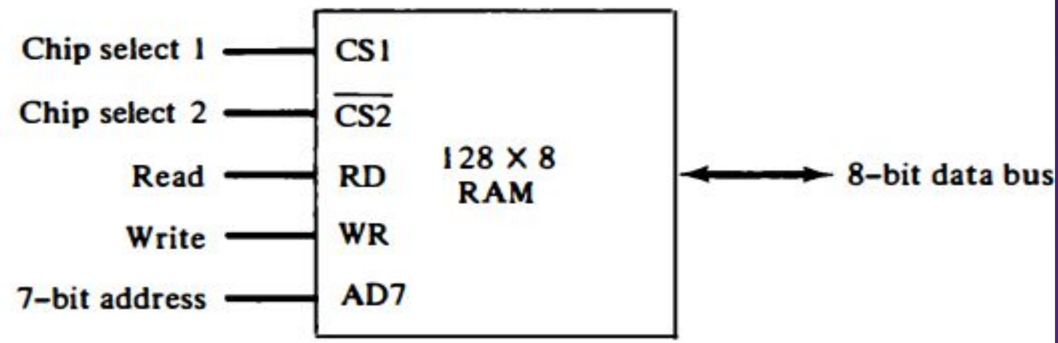
Static RAM	Dynamic RAM
Contains internal flip-flops for storing binary information	Stores binary information in the form of electric charges that are applied to capacitors
Information is valid as long as power is applied	Capacitors must be recharged periodically
Easier to use	Offers reduced power consumption
Has shorter read and write cycles	Larger Storage capacity
Can be used in implementing cache memory	Can be used in implementing main memory

MAIN MEMORY

- Portion of the memory can be constructed with ROM (Read-Only Memory) chips.

RAM	ROM
Temporary storage	Permanent storage
Volatile memory	Non volatile memory
Read/Write Memory	Read only memory
It is working area of computer	Start up memory of the computer which stores the **bootstrap loader**.
Fixed in the memory card	Attached with the motherboard

RAM CHIP



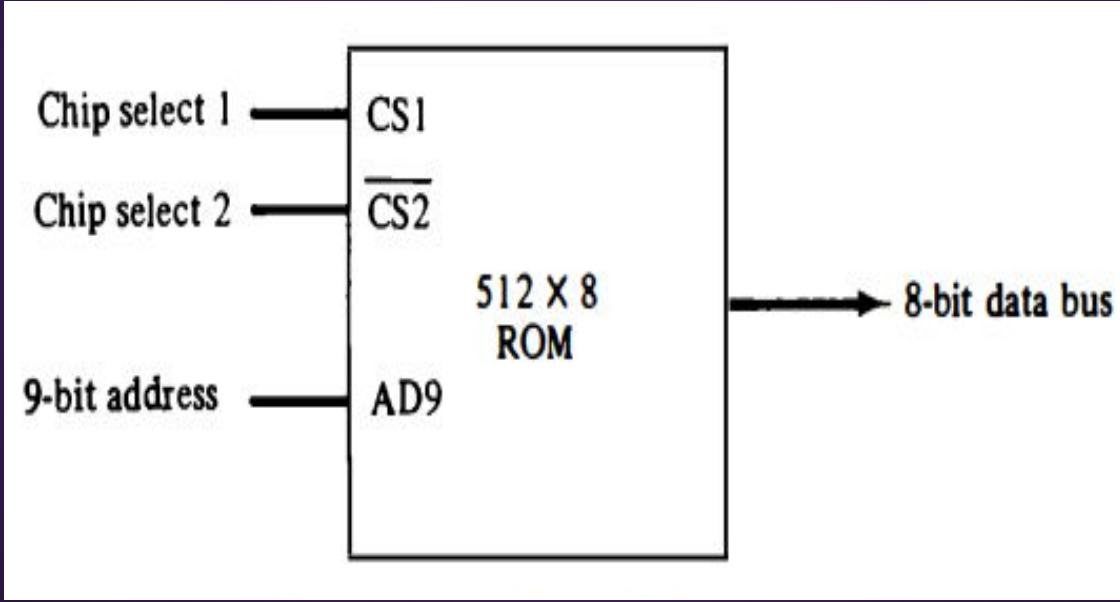
(a) Block diagram

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

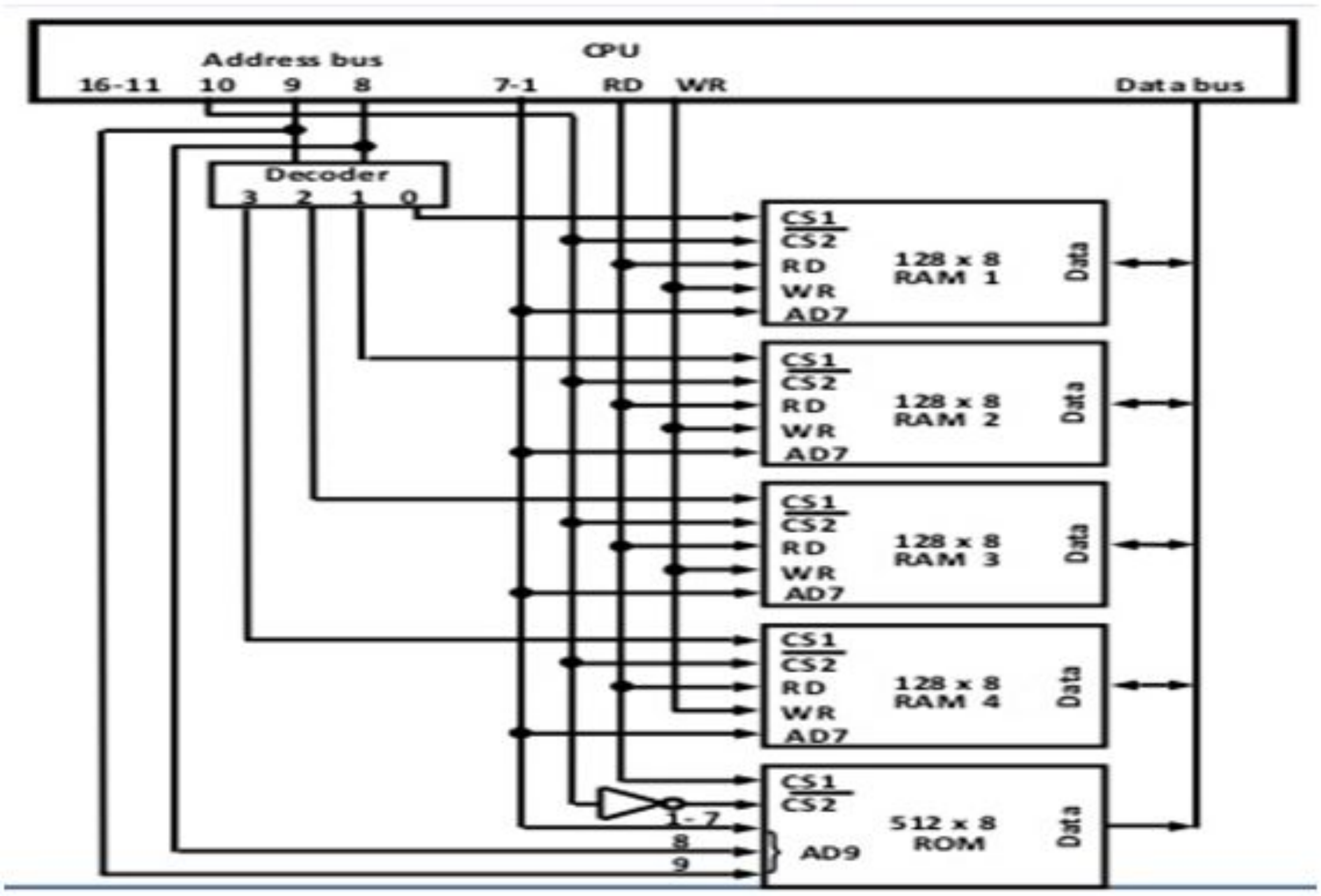
(b) Function table

ROM CHIP

- No need for R/W control



MEMORY CONNECTION TO CPU



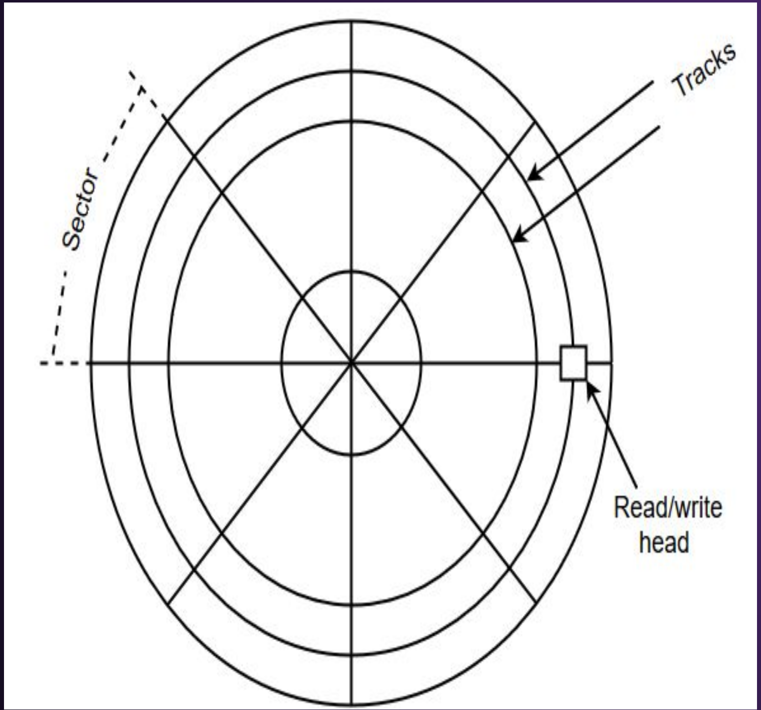
AUXILIARY MEMORY

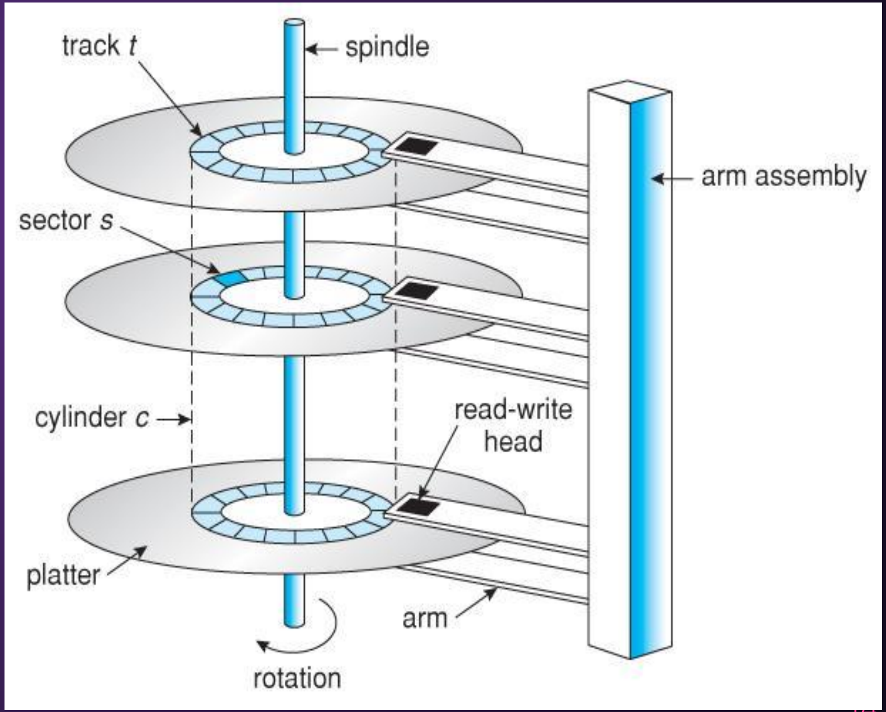
AUXILIARY MEMORY

- Commonly used auxiliary memory devices: Magnetic disks and tapes.

MAGNETIC DISKS

- Circular plate constructed of metal or plastic coated with magnetized material.





MAGNETIC DISKS

- Addressed by address bits that specify the disk no, surface, sector no, and the track within the sector.
- **Hard disks:** Disks that are permanently attached to the unit assembly and cannot be removed by the user
- Floppy disk

MAGNETIC TAPE

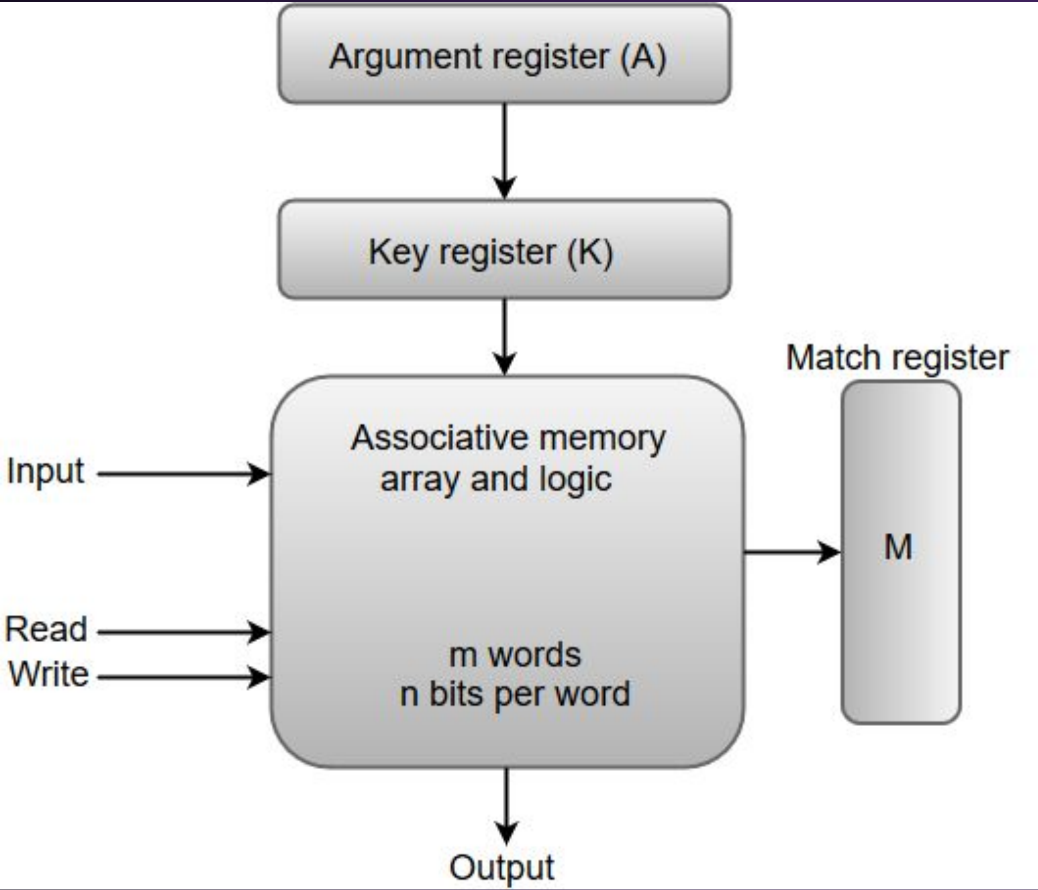
- The tape is a strip of plastic coated with a magnetic recording medium.
  - Can be stopped, started to move forward or in reverse.
  - Addressed by record number and number of characters in the record

ASSOCIATIVE MEMORY

ASSOCIATIVE MEMORY or CAM

- The data stored can be identified for access by the content of the data itself rather than by an address.
- Associative memory or Content Addressable Memory (CAM): A memory unit accessed by content
- No address is specified.
- Suitable for parallel search.
-

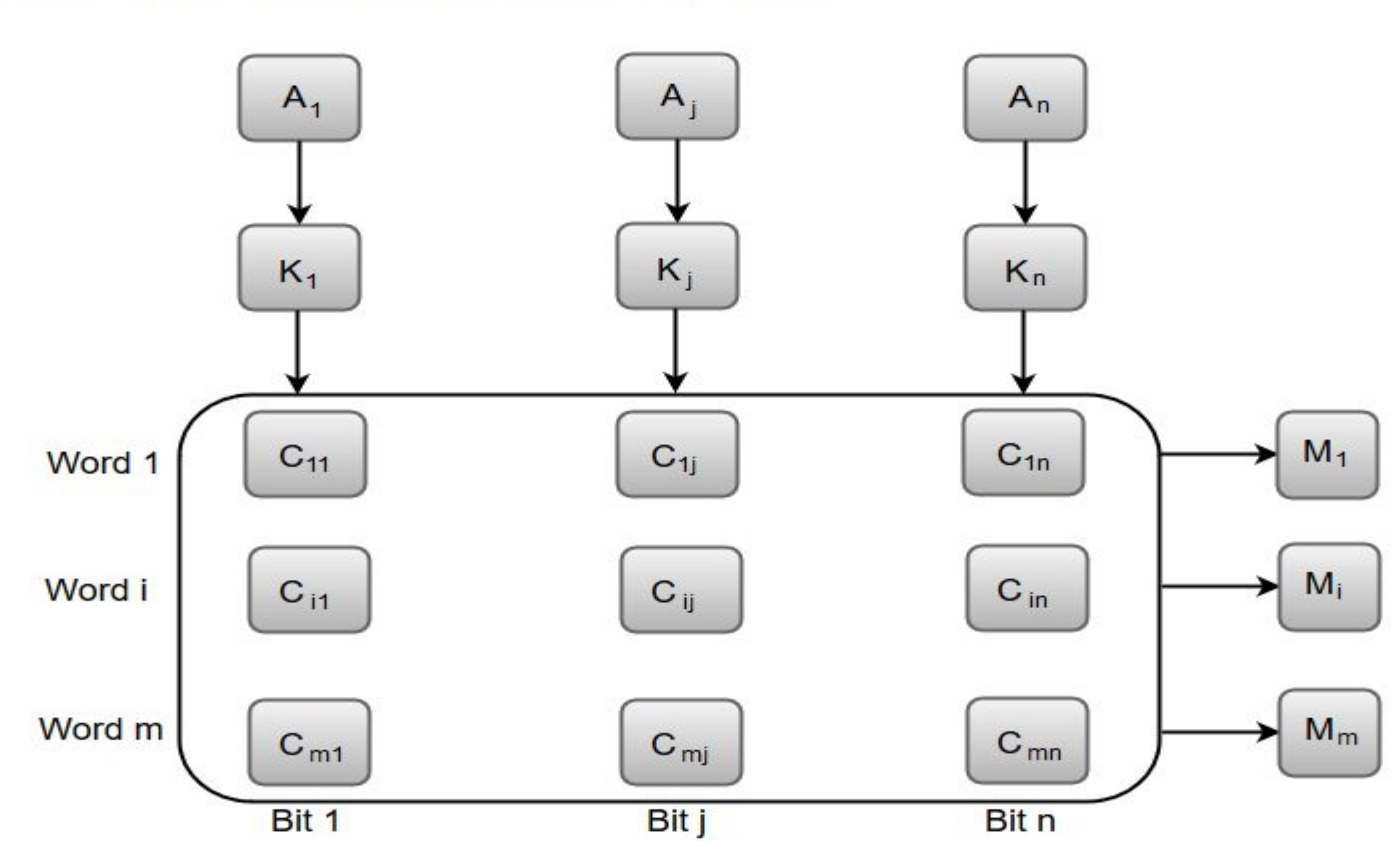
HARDWARE ORGANIZATIION



A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

HARDWARE ORGANIZATIION

Associative memory of m word, n cells per word:

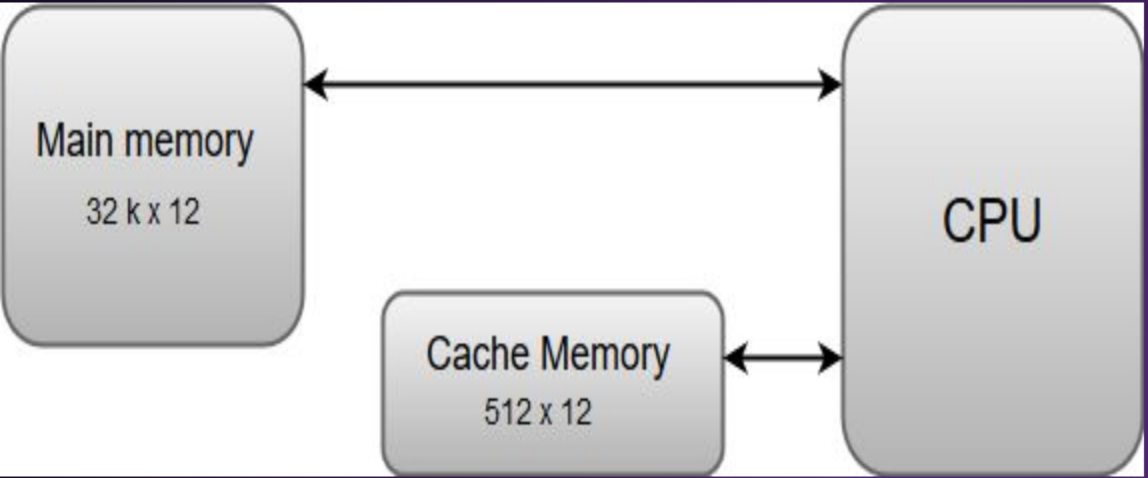


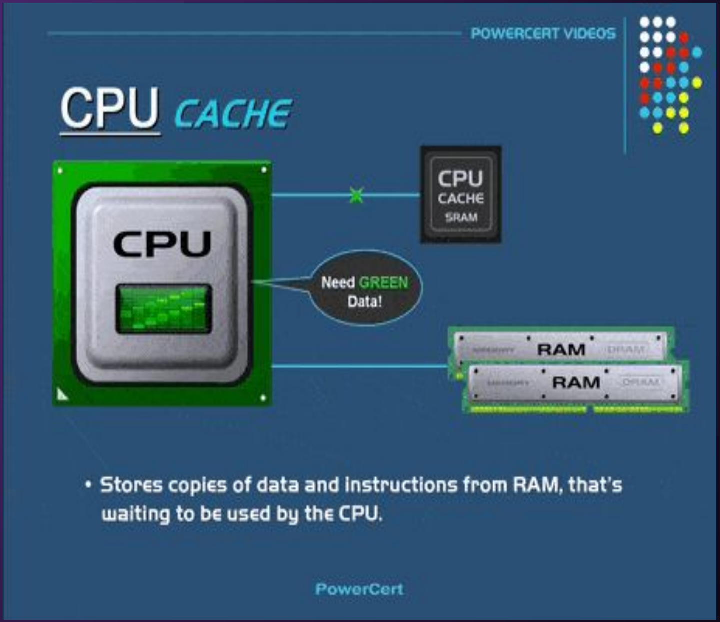
CACHE MEMORY

CACHE MEMORY

- Locality of Reference
- **Cache Memory:** The average memory access time can be reduced with this fast small memory.
- Access time is less by a factor of 5 to 10.
- **Fundamental idea:** The most frequently accessed instructions and data is stored in cache memory.

OPERATION OF CACHE MEMORY





PERFORMANCE OF CACHE MEMORY

- Measured by **Hit Ratio.**
- **Hit:** When CPU refers to memory and finds the word in cache
- **Miss:** If the word is not found in cache, it is in main memory and is referred as count.
- **Hit ratio:** The ratio of the no. of hits divided by the total CPU references (hit plus misses).

MAPPING PROCESS

- The transformation of data from main memory to cache memory.
- Three types of mapping

1. Direct Mapping
2. Associative Mapping
3. Set-Associative Mapping

File Organization and Indexing

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records*. A record is a sequence of fields.

- One approach
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

  This case is easiest to implement.
- We assume that records are smaller than a disk block.

Fixed-Length Records

- Simple approach:
  - Store sequentially
  - Record access is simple but records may cross blocks

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i*: *three* alternatives:
  - **move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1**
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

Record 3 deleted

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i*: alternatives:
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - **move record n to i**
  - do not move records, but link all free records on a *free list*

Record 3 deleted and replaced by record 11

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

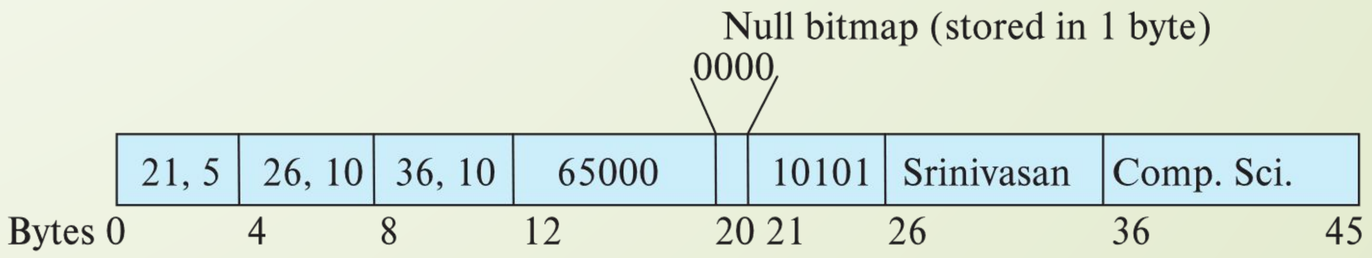
Fixed-Length Records

- Deletion of record *i*: alternatives:
  - move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1
  - move record *n* to *i*
  - **do not move records, but link all free records on a *free list***

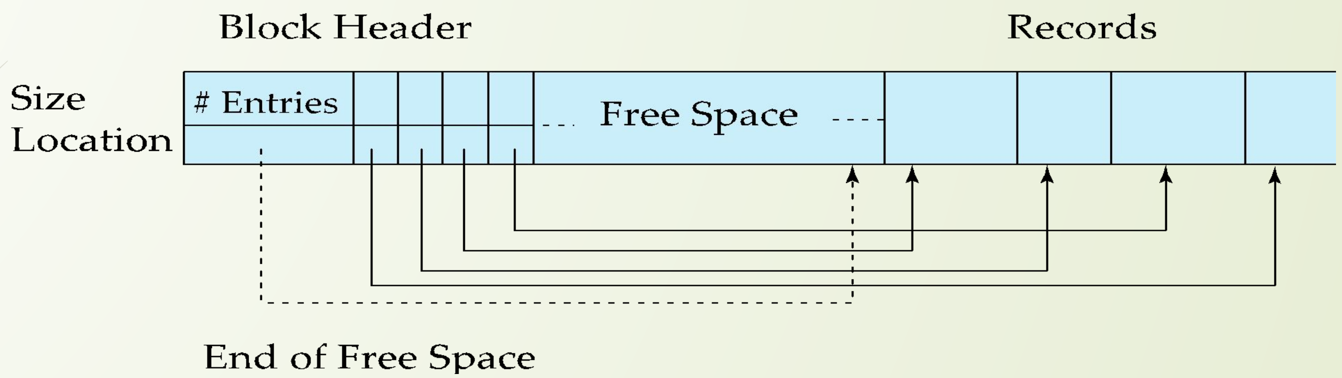
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**).
- Attributes are stored in order.
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes.
- Null values represented by null-value bitmap



Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record

- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.

Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space

- **Sequential** – store records in sequential order, based on the value of the search key of each record

- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed

Heap File Organization

- Records can be placed anywhere in the file where there is free space.
- Records usually do not move once allocated.
- There is no ordering of records based on search key.

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file.
- The records in the file are ordered by a search-key.

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization (Cont.)

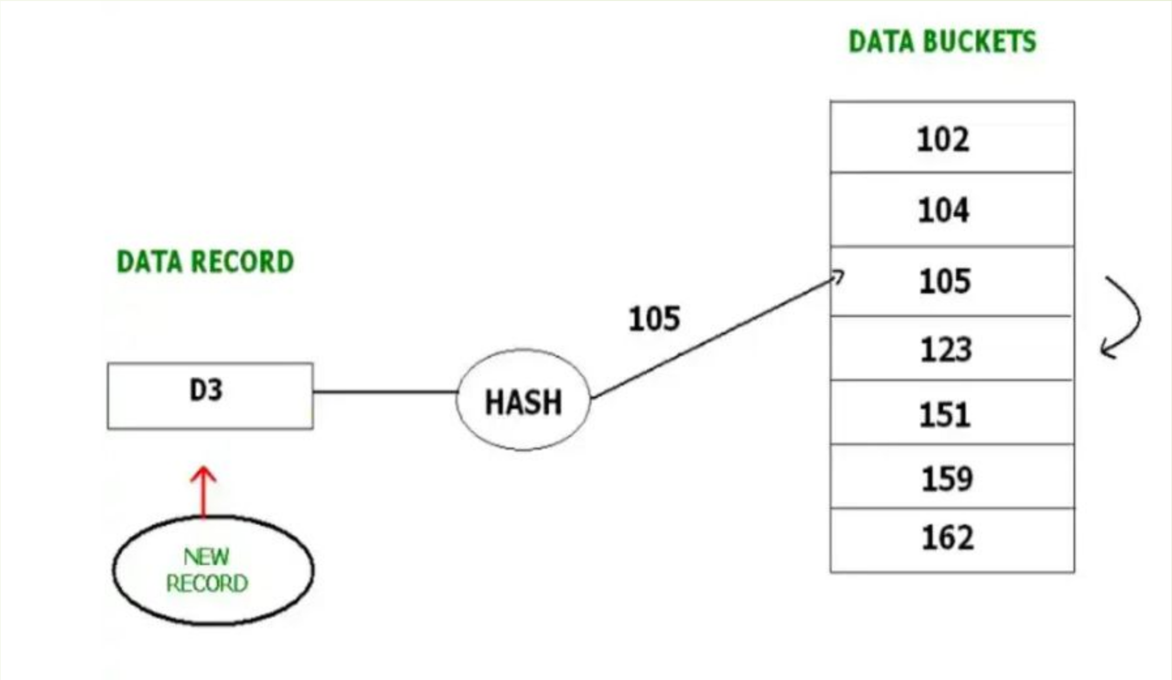
- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there.
  - if no free space, insert the record in an overflow block.
  - In either case, pointer chain must be updated.

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Hashing file oraganization

- **Hashing** is an efficient technique to directly search the location of desired data on the disk.



Indexing in database

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library.
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer

- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order.
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

Index Evaluation Metrics

- Access time
- Insertion time
- Deletion time
- Space overhead

Ordered Indices

- In an **ordered index**, index entries are stored sorted on the search key value.

- **Clustering index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
  - Also called **primary index**
  - The search key of a primary index is usually but not necessarily the primary key.

- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called **non-clustering index.**

Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation

10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Dense Index

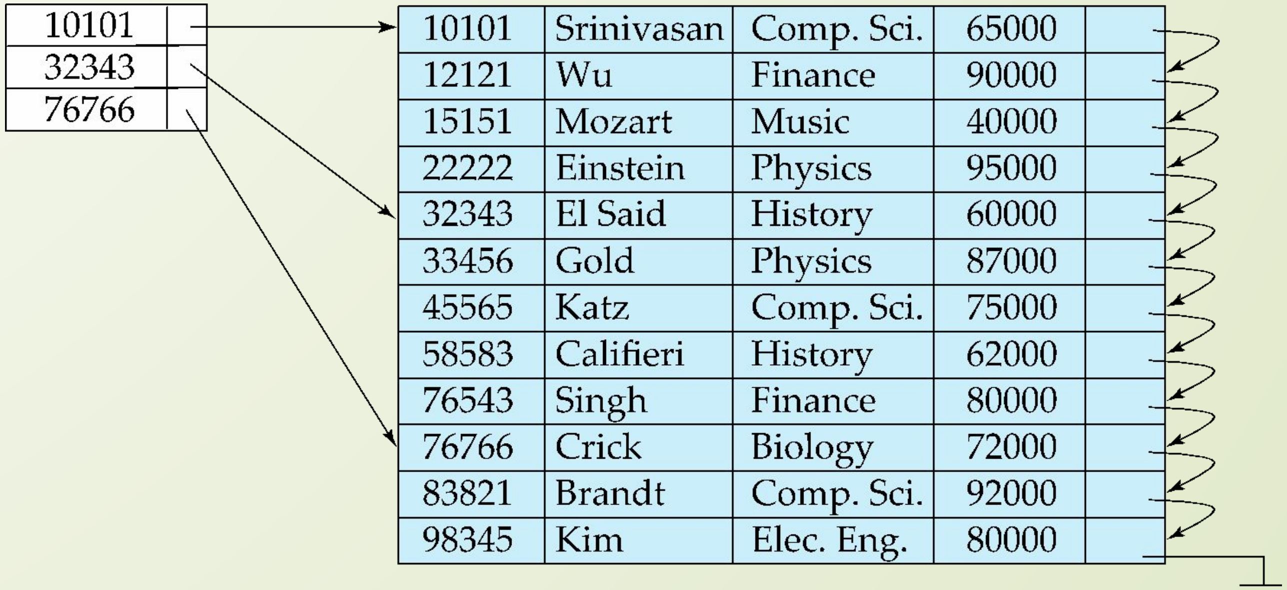
Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

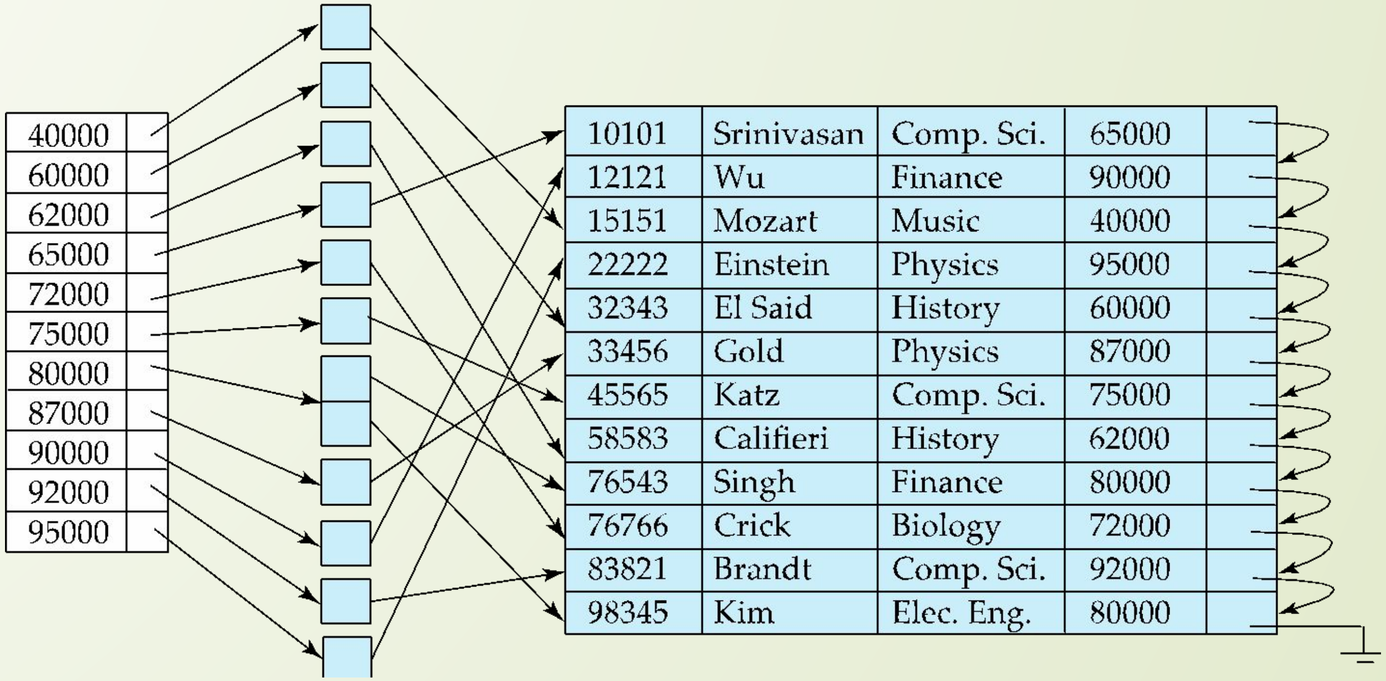
Sparse Index Files

- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points



Secondary Indices Example

- Secondary index on salary field of instructor



- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value.
- Secondary indices have to be dense.

Multilevel Index

- If index does not fit in memory, access becomes expensive.
- Solution: treat index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of the basic index
  - inner index – the basic index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

Multilevel Index