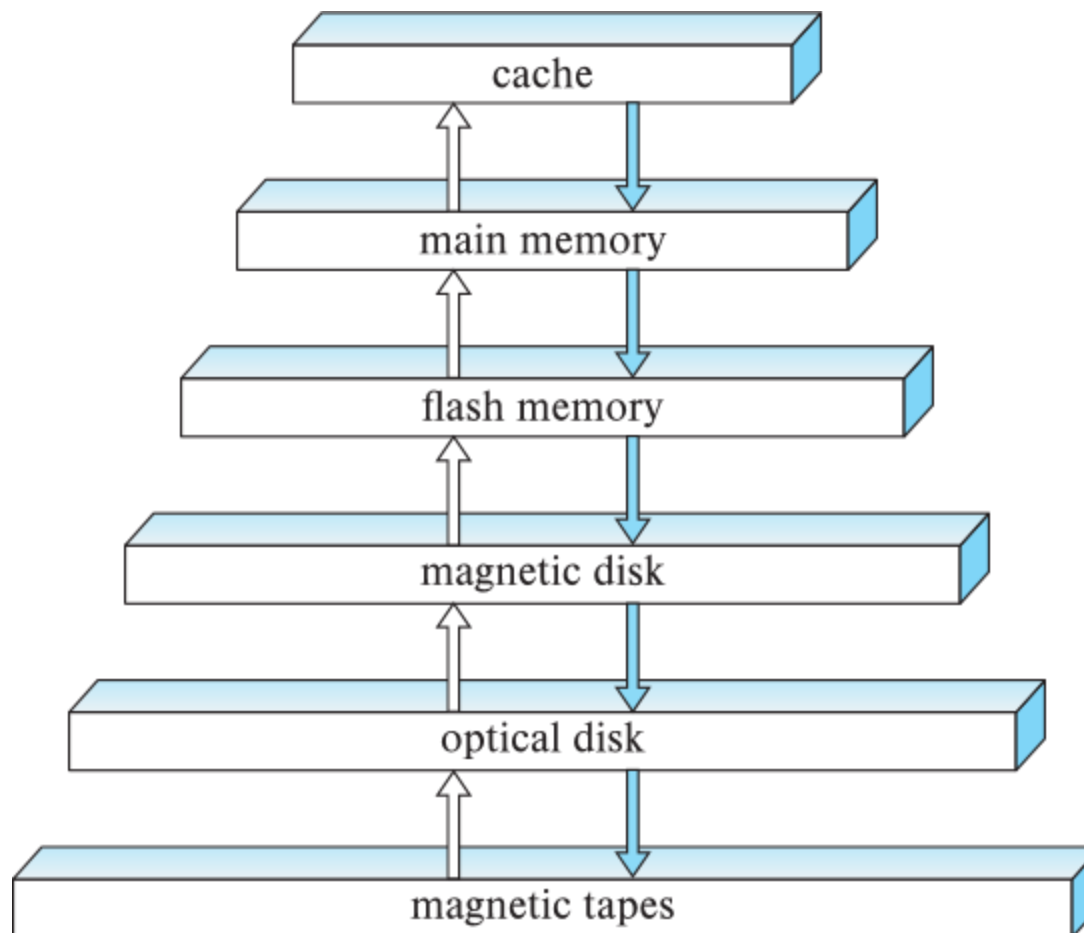


Storage

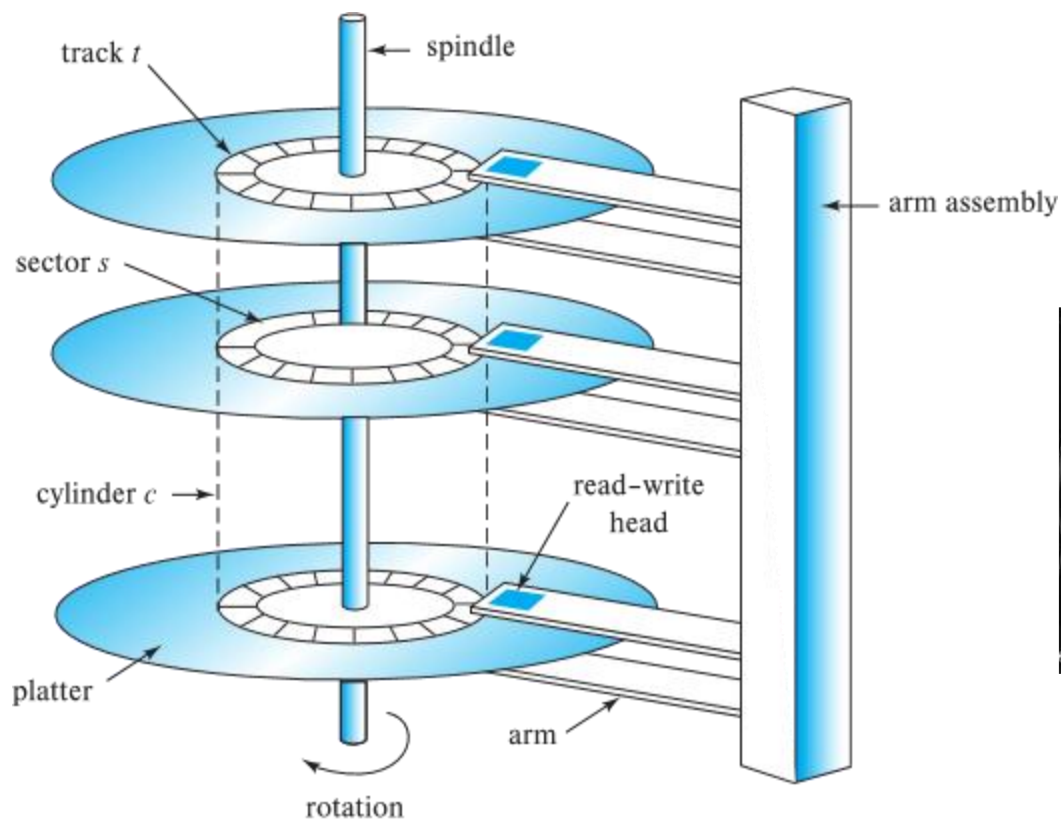
Miao Qiao
The University of Auckland

Storage Hierarchy



[Google Data Center](#)

Magnetic Hard Disk Mechanism



Schematic diagram of magnetic disk drive

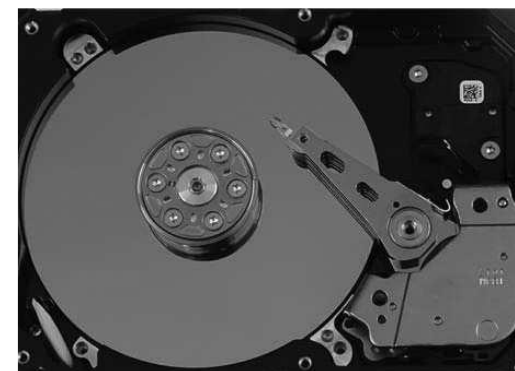


Photo of magnetic disk drive

Performance Measures

- **Disk block**
- **Sequential access pattern**
 - Magnetic disks: 25 to 200 MB per second max rate on
 - SSD: 400 MB/sec for SATA3, 2 to 3 GB/sec using NVMe PCIe
- **Random access pattern**
- **I/O operations per second (IOPS)**
 - Number of random block reads that a disk can support per second
 - Magnetic disks (4K block): 50 to 200 IOPS
 - SSD (4K block):
 - 10,000 reads per second (10,000 IOPS)
 - 40,000 writes per second
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
 - Typically 3 to 5 years
- **Hybrid disks**
 - combine small amount of flash cache with larger magnetic disk

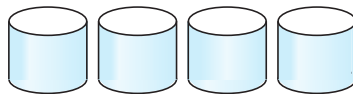
RAID

- **RAID: Redundant Arrays of Independent Disks**

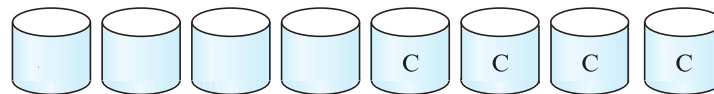
- manage a large numbers of disks, providing a view of a single disk of
 - **high capacity,**
 - **high speed,**
 - **high reliability**
- $\text{Prob}(\text{a disk out of a set of } N \text{ disks fails}) \gg \text{Prob}(\text{a single disk fails})$.
 - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)

Improvement of Reliability via Redundancy

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or **shadowing**)
- **Mean time to data loss** depends on mean time to failure, and **mean time to repair**
 - E.g., MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of 500×10^6 hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)



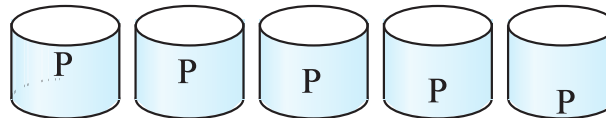
(a) RAID 0: nonredundant striping



(b) RAID 1: mirrored disks

Improvement in Performance via Parallelism

- Two main goals of parallelism in a disk system:
 1. Load balance multiple small accesses to increase **throughput**
 2. Parallelize large accesses to reduce **response time**.
- Improve transfer rate by striping data across multiple disks.
- **Bit-level striping**
- **Block-level striping**
- **Parity blocks**



(c) RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

Choice of RAID Level

- Factors in choosing RAID level
 - Monetary cost
 - Performance: Number of I/O operations per second, and bandwidth during normal operation
 - Performance during failure
 - Performance during rebuild of failed disk
 - Including time taken to rebuild failed disk
- Level 0 is used only when data safety is not important
 - E.g., data can be recovered quickly from other sources
- Level 1 provides much better write performance than level 5
 - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes

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; will consider variable length records later

- We assume that records are smaller than a disk block.

Fixed-Length Records

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

Deletion of record 3:

1. move records $3 + 1, \dots, n$ to $3, \dots, n - 1$, or
2. move record n to 3, or
3. do not move records, but link all free records on a *free list*

Fixed-Length Records

- move records $3 + 1, \dots, n$ to $3, \dots, n - 1$

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

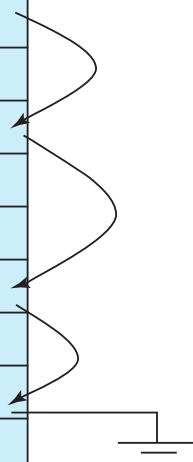
- move record n to i

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

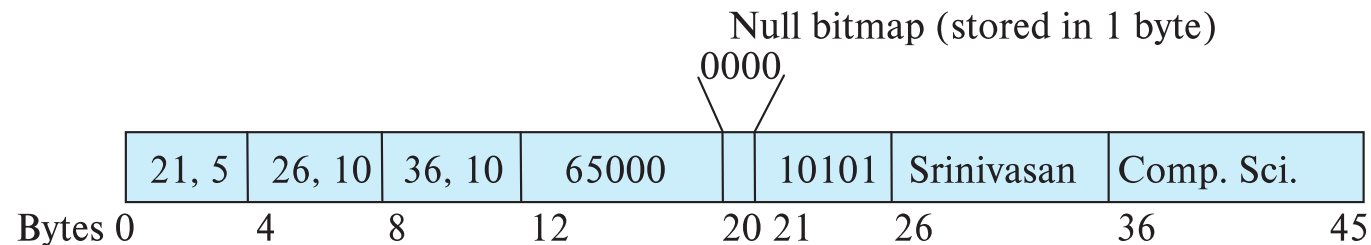
- do not move records, but link all free records on a *free list*

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



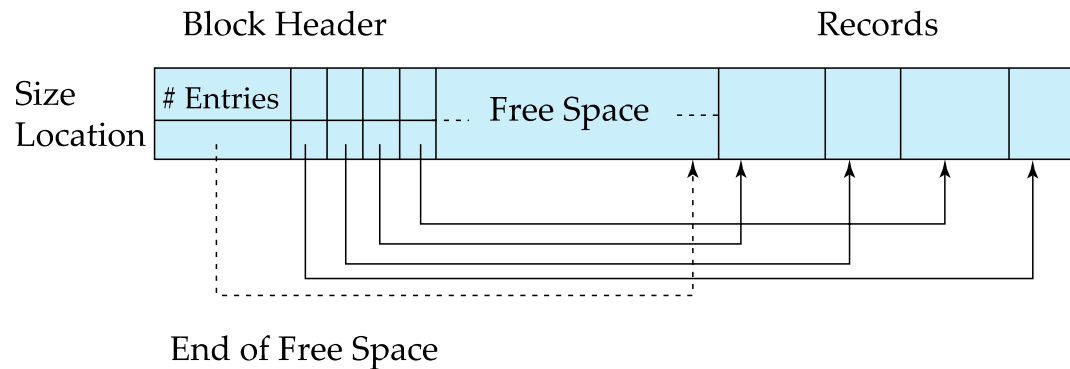
Variable-Length Records

- Variable lengths such as **varchar**



- Attributes in order
- (offset, length): actual data stored after all fixed-length attributes

Variable-Length Records: Slotted Page Structure



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
- **B⁺-tree file organization**
 - Ordered storage even with inserts/deletes
 - More on this in Indexing
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed
 - More on this in Indexing

Sequential File Organization

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	

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	
-------	-------	-------	-------	--

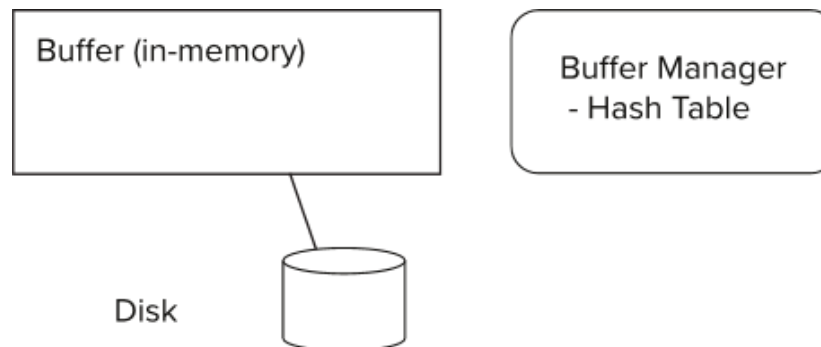
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata** -- data about data, such as

- Relations
 - names
 - names, types and lengths of attributes
 - names and definitions of views
 - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
 - number of tuples in each relation
- Physical file organization
 - How relation is stored (sequential/hash/...)
 - Physical location of relation
- Information about indices...

Storage Access

- Blocks are units of both storage allocation and data transfer.
- **I/O model** - database system seeks to minimize the number of block transfers between the disk and memory.
- We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
- **Buffer** – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory.



Buffer Manager

- **Buffer replacement strategy** (details coming up!)
- **Pinned block**
 - **Pin**
 - **Unpin**
 - Multiple concurrent pin/unpin operations possible
 - Keep a pin count, buffer block can be evicted only if pin count = 0
- **Shared and exclusive locks on buffer**
 - **Locking rules:**
 - Only one process can get exclusive lock at a time
 - Shared lock cannot be concurrently with exclusive lock
 - Multiple processes may be given shared lock concurrently

Buffer-Replacement Policies

- **Least recently used (LRU strategy)**

- Idea behind LRU – use past pattern of block references as a predictor of future references
- LRU can be bad for some queries
 - Example of bad access pattern for LRU: when computing the join of 2 relations r and s by a nested loops

for each tuple tr of r do
 for each tuple ts of s do
 if the tuples tr and ts match ...

- **Most recently used (MRU) strategy**

- System must pin the block currently being processed.
- After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.

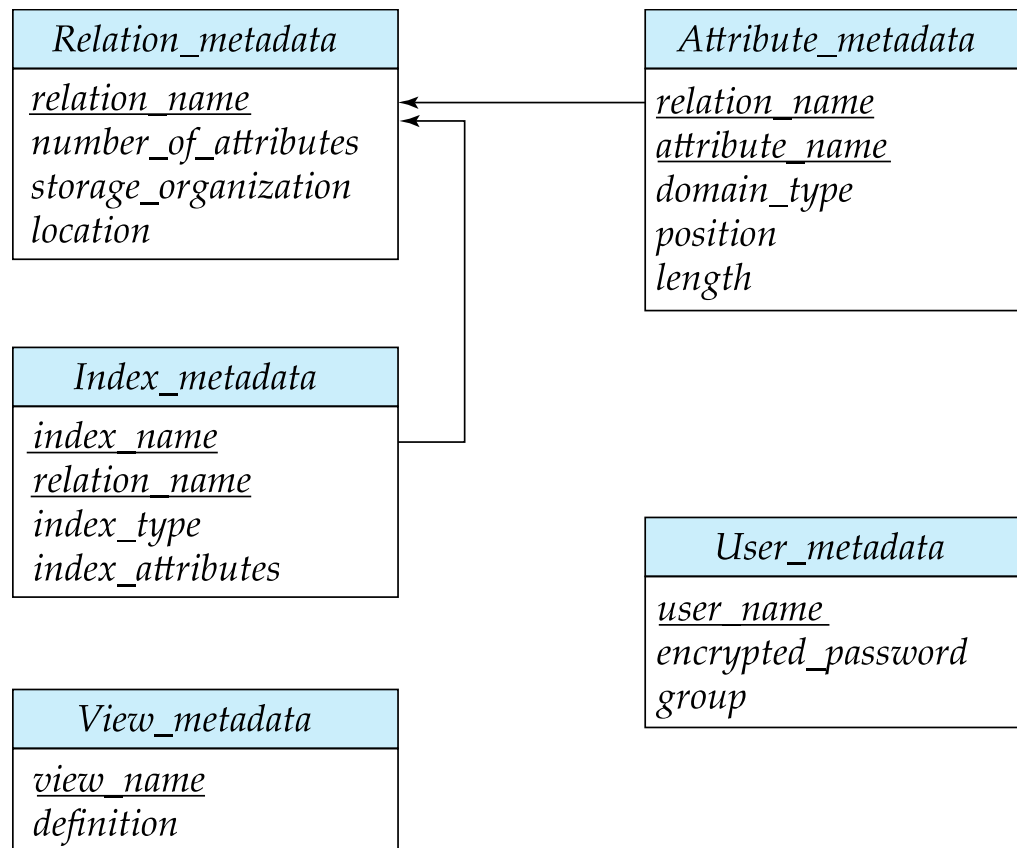
- **Buffer manager** can use statistical information regarding the probability that a request will reference a particular relation

FIN

Any questions?

Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



Optimization of Disk Block Access (Cont.)

- Buffer managers support **forced output** of blocks for the purpose of recovery (more in Chapter 19)
- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM or flash buffer immediately
 - *Writes can be reordered to minimize disk arm movement*
- **Log disk** – a disk devoted to writing a sequential log of block updates
 - Used exactly like nonvolatile RAM
 - Write to log disk is very fast since no seeks are required
- **Journaling file systems** write data in-order to NV-RAM or log disk
 - Reordering without journaling: risk of corruption of file system data

Column-Oriented Storage

- Also known as **columnar representation**

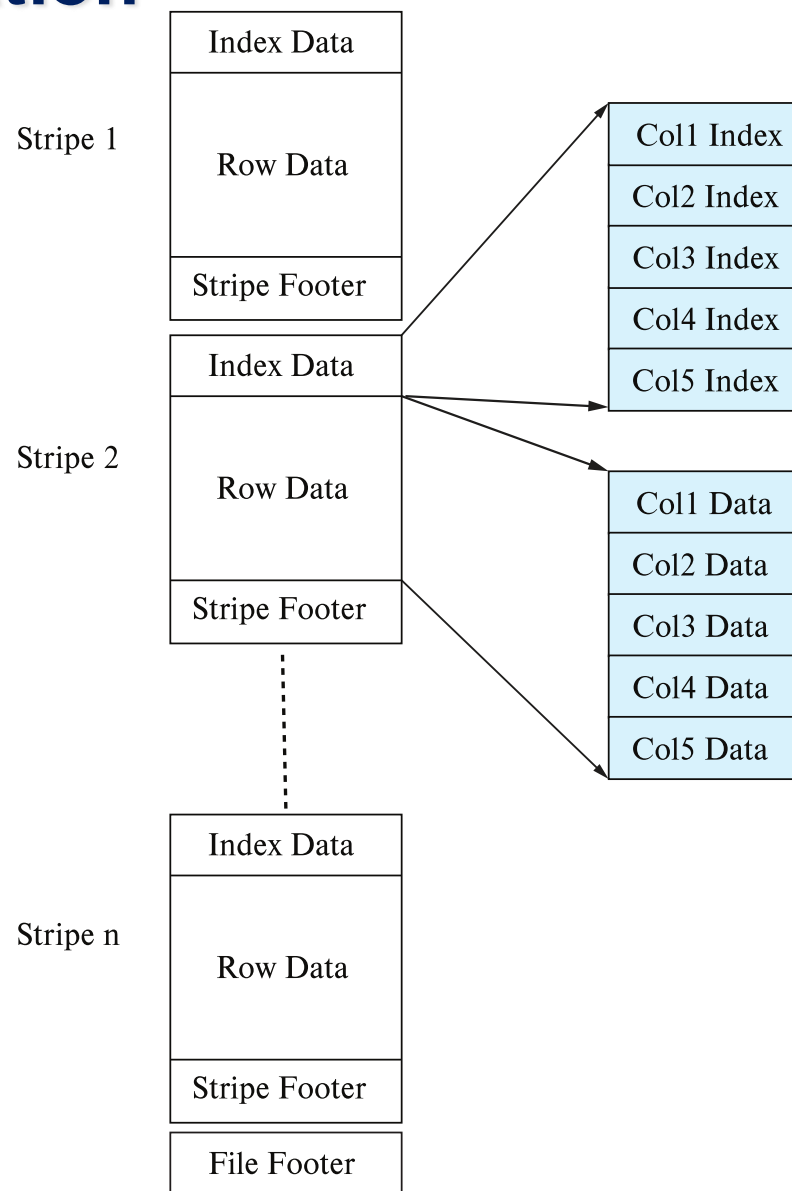
Stor	10101	Srinivasan	Comp. Sci.	65000
	12121	Wu	Finance	90000
Exa	15151	Mozart	Music	40000
	22222	Einstein	Physics	95000
	32343	El Said	History	60000
	33456	Gold	Physics	87000
	45565	Katz	Comp. Sci.	75000
	58583	Califieri	History	62000
	76543	Singh	Finance	80000
	76766	Crick	Biology	72000
	83821	Brandt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

Columnar Representation

- Benefits:
 - Reduced IO if only some attributes are accessed
 - Improved CPU cache performance
 - Improved compression
 - **Vector processing** on modern CPU architectures
- Drawbacks
 - Cost of tuple reconstruction from columnar representation
 - Cost of tuple deletion and update
 - Cost of decompression
- Columnar representation found to be more efficient for decision support than row-oriented representation
- Traditional row-oriented representation preferable for transaction processing
- Some databases support both representations
 - Called **hybrid row/column stores**

Columnar File Representation

- ORC and Parquet: file formats with columnar storage inside file
- Very popular for big-data applications
- Orc file format shown on right:



Storage Organization in Main-Memory Databases

- Can store records directly in memory without a buffer manager
- Column-oriented storage can be used in-memory for decision support applications
 - Compression reduces memory requirement

