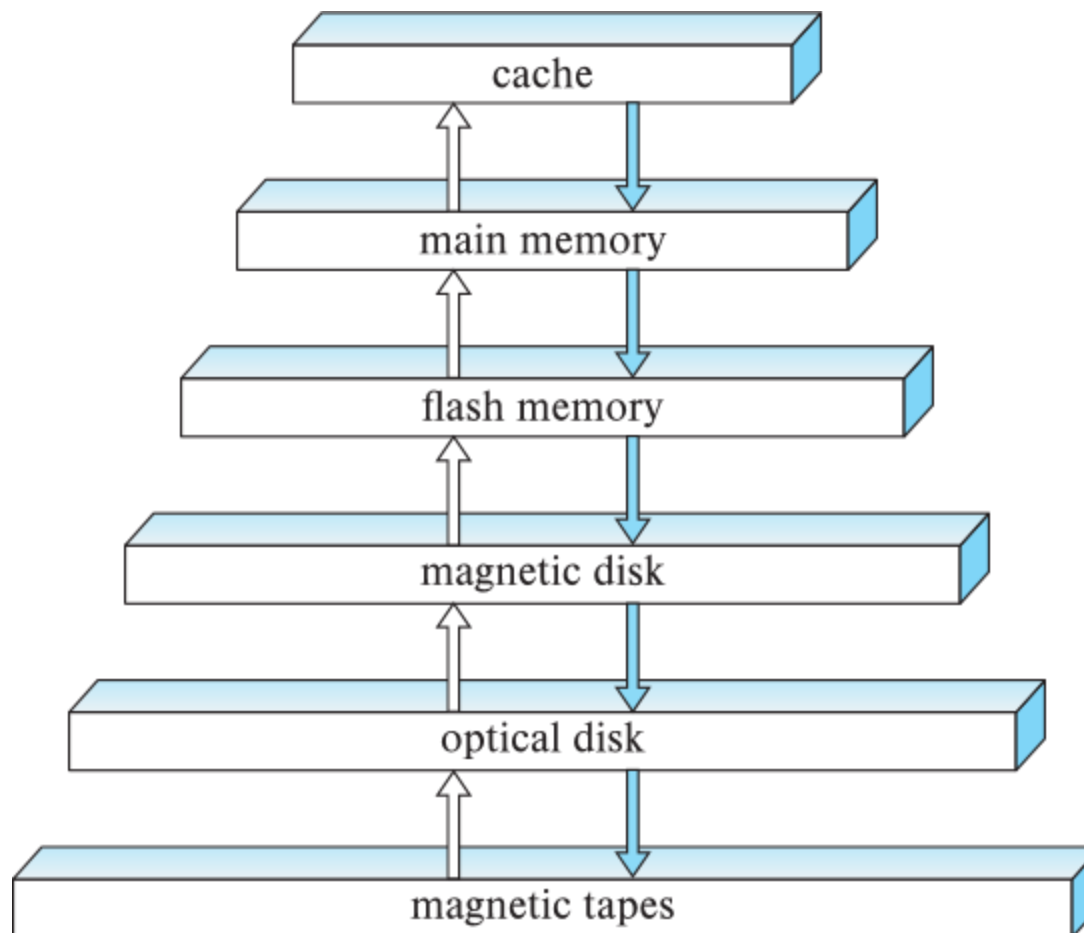


# Storage

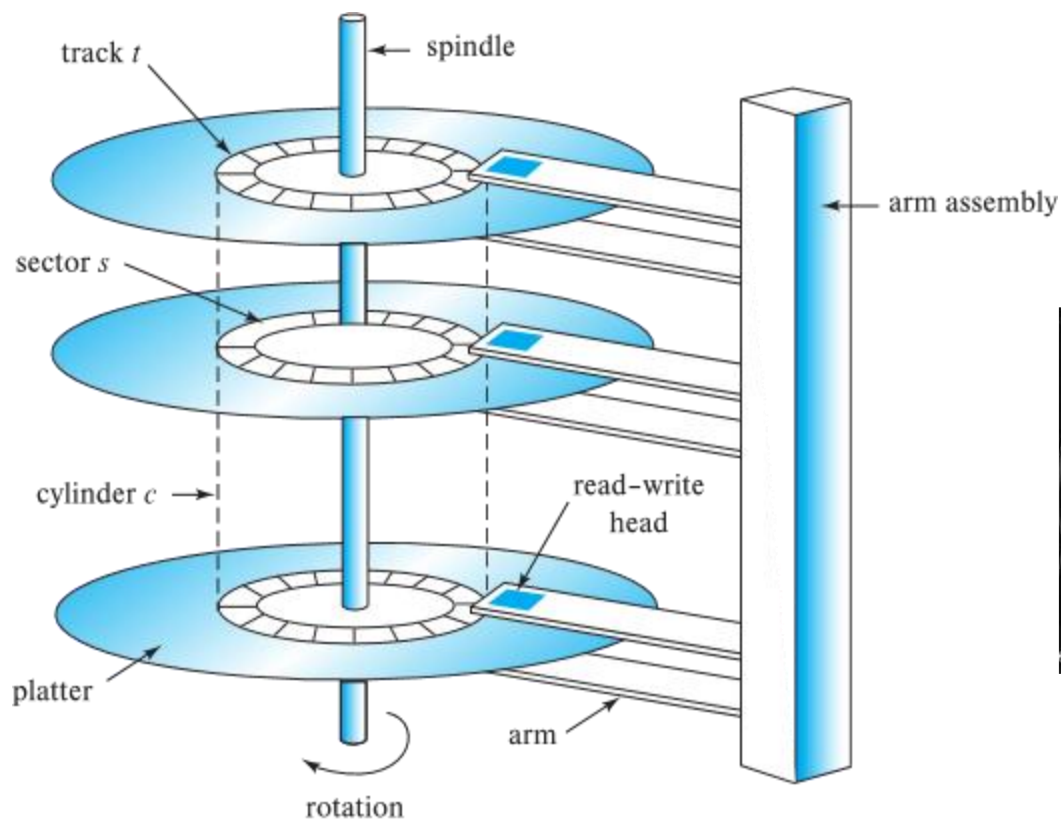
**Miao Qiao**  
The University of Auckland

# Storage Hierarchy

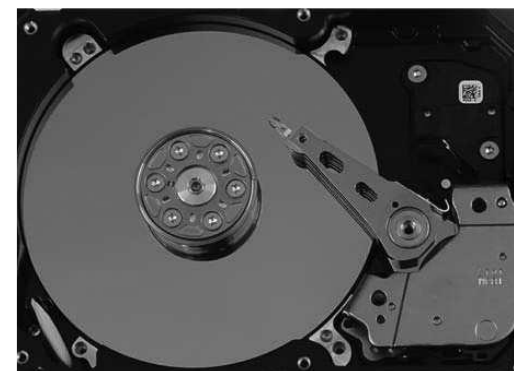


[Google Data Center](#)

# 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

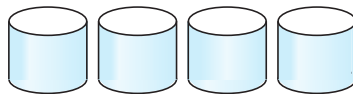
# RAID

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

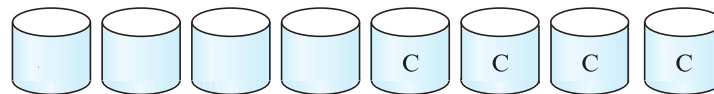
- manage a large numbers of disks, providing a view of a single disk of
  - **high capacity,**
  - **high speed,**
  - **high reliability**
- $\text{Prob}(\text{a disk out of a set of } N \text{ disks fails}) \gg \text{Prob}(\text{a single disk fails})$ .
  - E.g., a system with 100 disks, each with 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  $500 \times 10^6$  hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)



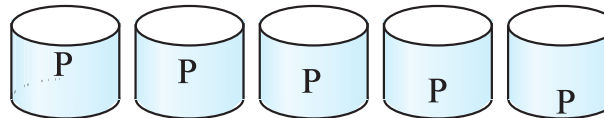
(a) RAID 0: nonredundant striping



(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

# Exercise

A database administrator can choose how many disks are organized into a single RAID 5 array. What are the trade-offs between having fewer disks versus more disks, in terms of

- cost,
- reliability,
- performance during failure, and
- performance during rebuild?



# 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

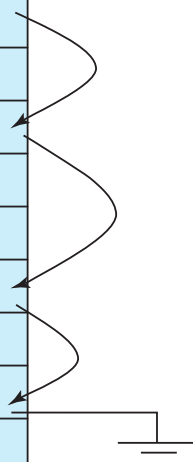
- move record  $n$  to  $i$

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

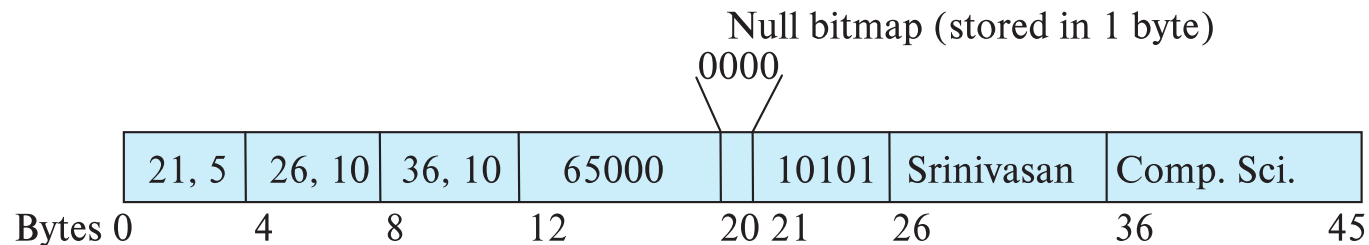
- do not move records, but link all free records on a *free list*

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



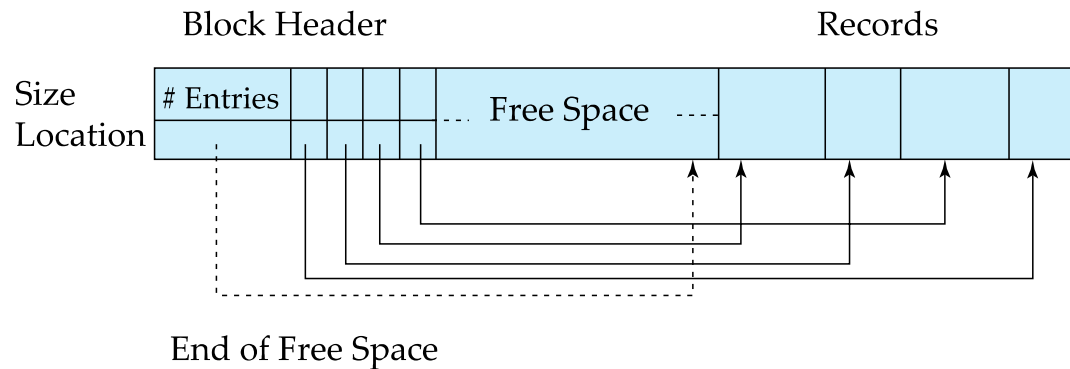
# Variable-Length Records

- 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<sup>+</sup>-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	
-------	-------	-------	-------	--

FIN

Any questions?