

CS 4440 A

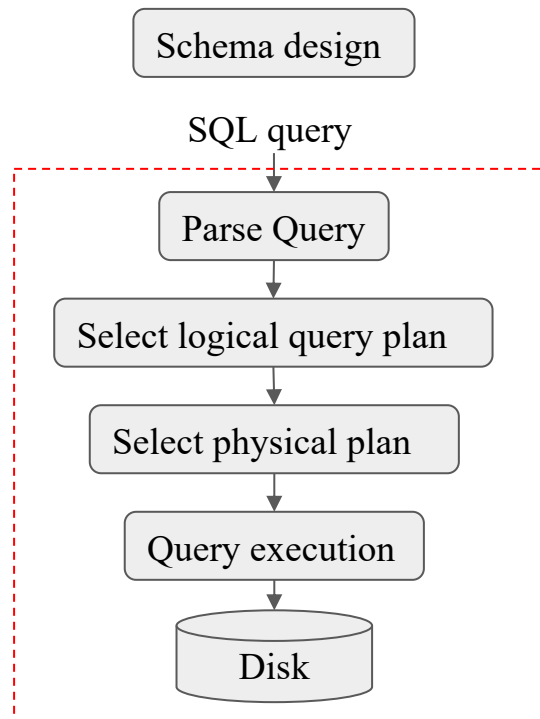
# Emerging Database Technologies

---

Lecture 5  
01/24/24

# Next Part: Database System Implementation

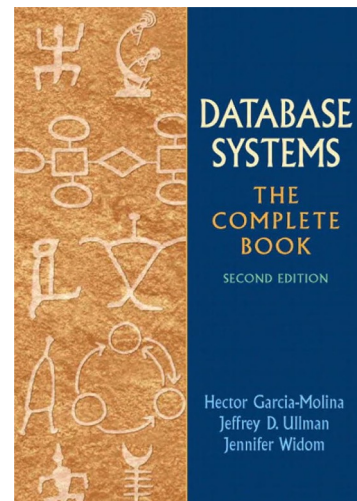
- Hardware and file system structure
- Indexing and hashing
- Query optimization
- Transactions
- Crash recovery
- Concurrency control



# Reading Materials

Database Systems: The Complete Book (2nd edition)

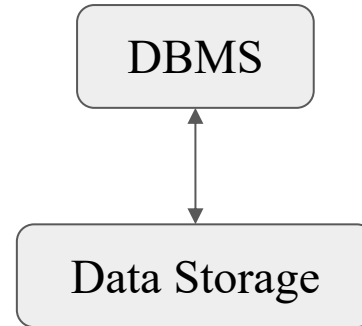
- Chapter 13: Secondary Storage Management



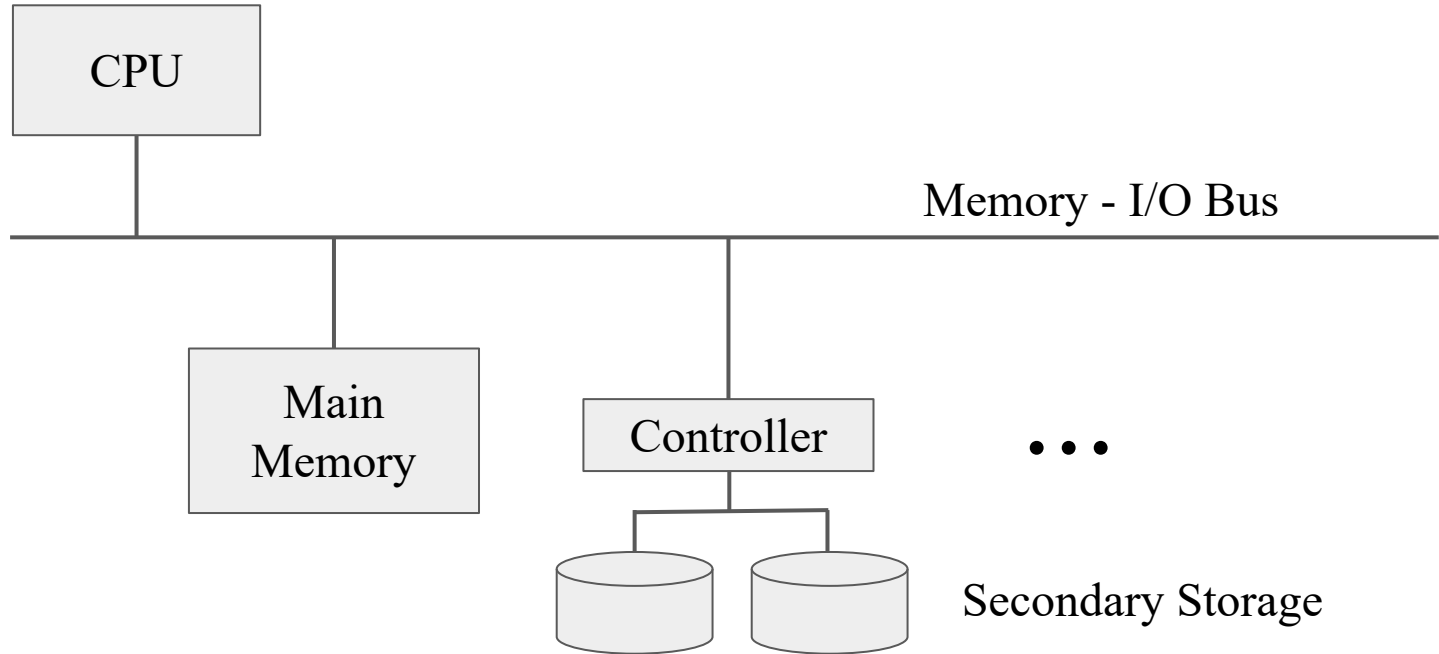
Acknowledgement: The following slides have been adapted from EE477 (Database and Big Data Systems) taught by Steven Whang.

# Data storage

- Database systems always involve secondary storage, e.g., disk
- We will study how a typical computer system manages storage
  - Memory hierarchy
  - Speed of data access
  - Data representation



# Typical computer system (Von Neumann architecture)

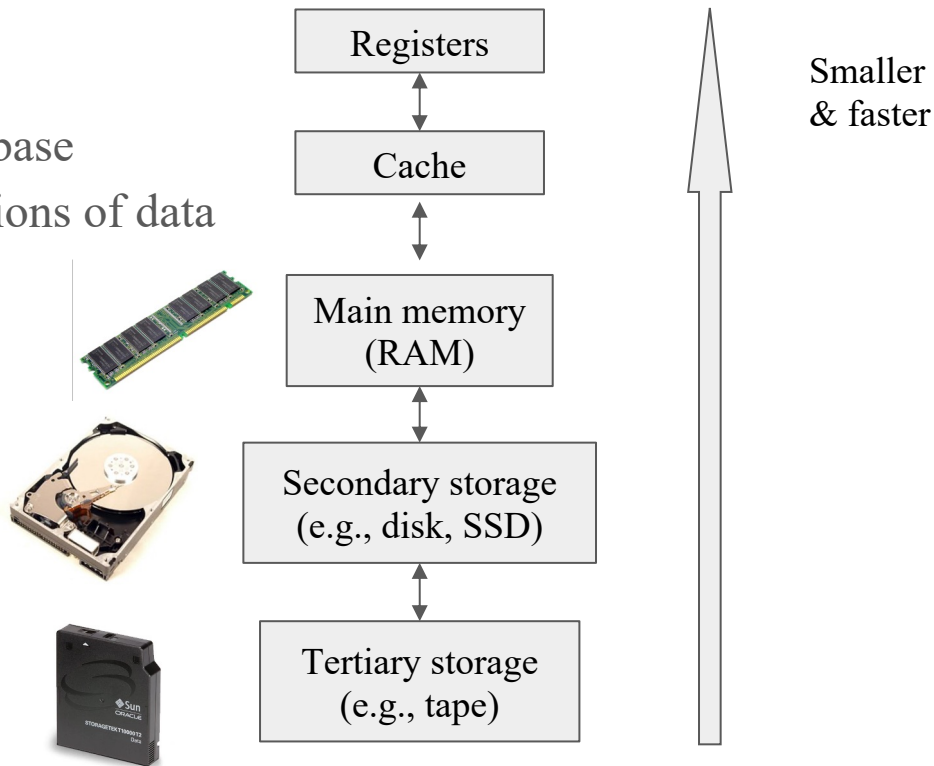


# Why not store everything in main memory?

- Memory is expensive
  - Memory is ~\$10/GB
  - Solid state disk (SSD) is ~\$0.1/GB
  - Magnetic disk is ~\$0.05/GB
  - Large databases can be 10-100 TB
- Memory is volatile
- Some specialized systems do store the entire database in memory

# Storage hierarchy

- Main memory stores current data
- Secondary storage stores main database
- Tertiary storage archives older versions of data



# Numbers everyone should know by Jeff Dean

"Numbers Everyone Should Know" from Jeff Dean. [Slides #1](#), [Slides #2](#)

L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	
Mutex lock/unlock	100 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	10,000 ns	0.01 ms
Send 1K bytes over 1 Gbps network	10,000 ns	0.01 ms
Read 1 MB sequentially from memory	250,000 ns	0.25 ms
Round trip within same datacenter	500,000 ns	0.5 ms
Disk seek	10,000,000 ns	10 ms
Read 1 MB sequentially from network	10,000,000 ns	10 ms
Read 1 MB sequentially from disk	30,000,000 ns	30 ms
Send packet CA->Netherlands->CA	150,000,000 ns	150 ms

Where



- 1 ns =  $10^{-9}$  seconds
- 1 ms =  $10^{-3}$  seconds





Google AI boss



# Jim Gray's storage latency analogy: how far is the data?

$10^9$  Tape   Andromeda 2,000 years

$10^6$  Disk   Pluto 2 years

100 Memory   Columbus, GA 1.5 hours

10 On board cache  This building 10 min

2 On chip cache  This room 1 min

1 Registers  My head



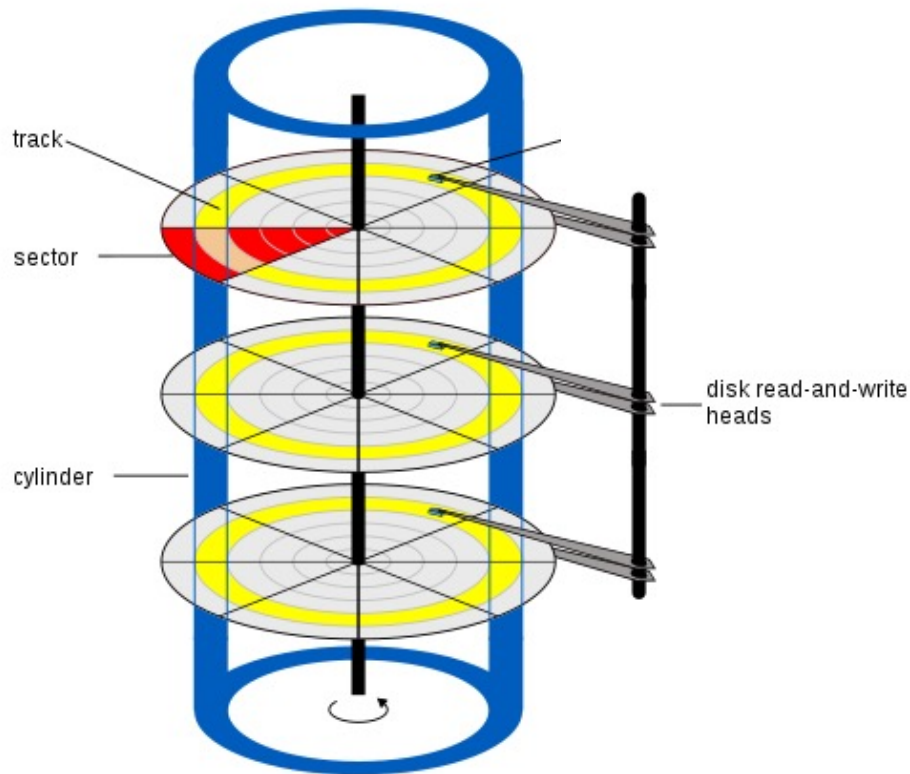
Turing Award, 1998

# Secondary storage

- Storage that is not directly accessible by the CPU
- Retains data even when the computer is powered off
  - Unlike RAM, which is temporary and volatile
- Examples:
  - hard drives
  - SSDs
  - USB flash drives

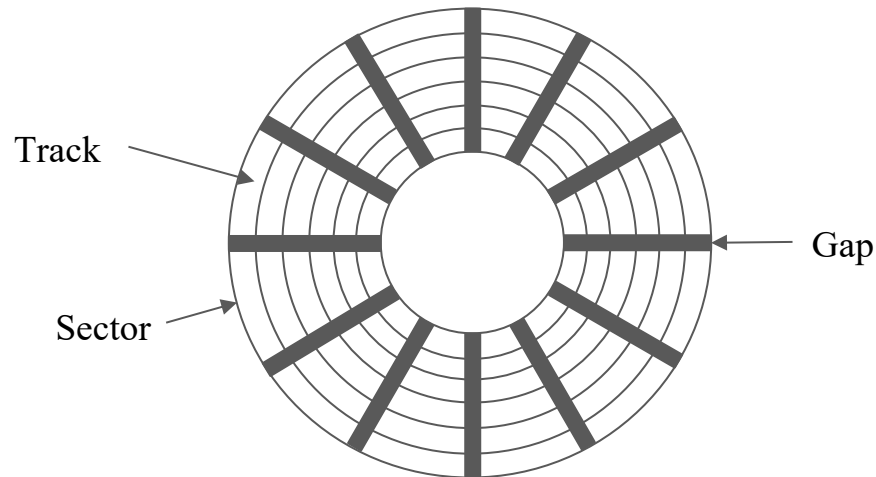
# Disks (HDDs)

- We will focus on the typical magnetic disk
- One or more circular platters rotate around a spindle
- Tracks of the same radius form a cylinder



# Top view of disk surface

- The disk is organized into tracks
- Tracks are organized into sectors, which are indivisible units
- Blocks (unit of transfer to memory) consist of one or more sectors
- Gaps are used to identify the beginnings of sectors



# Disk access time

- Latency = seek time + rotational delay + transfer time + other
  - Seek time: position disk head on a cylinder containing the track to read
  - Rotational latency: time until the first sector of the block moves under the head
  - Transfer time: time to read/write data in sectors

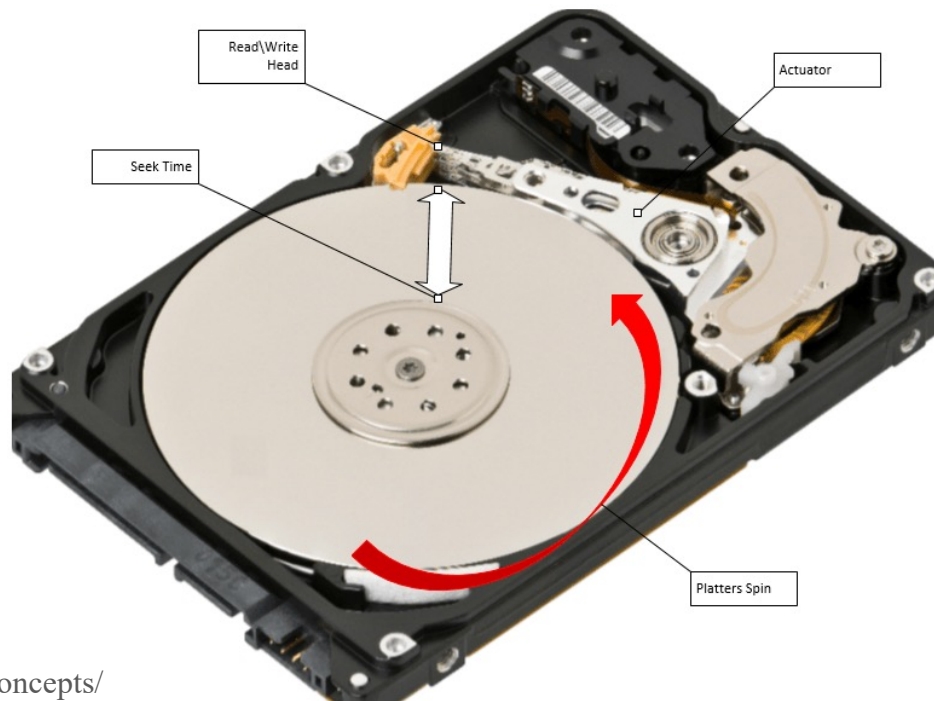
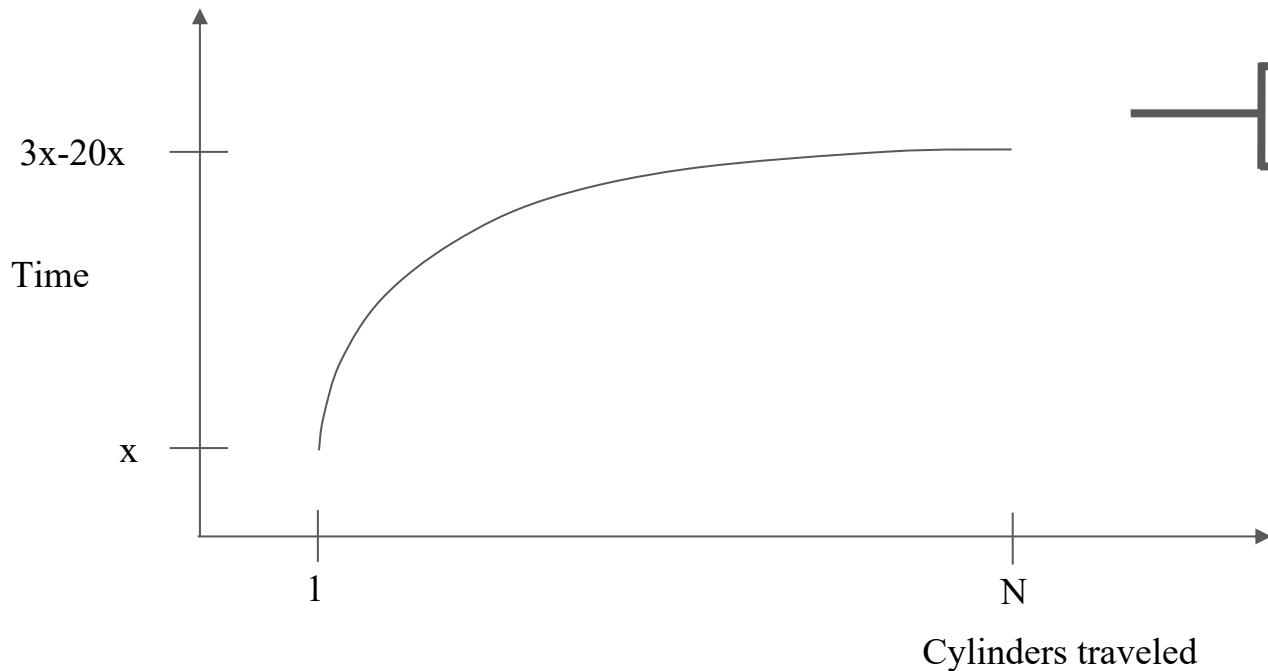


Image source: <https://theithollow.com/2013/11/18/disk-latency-concepts/>

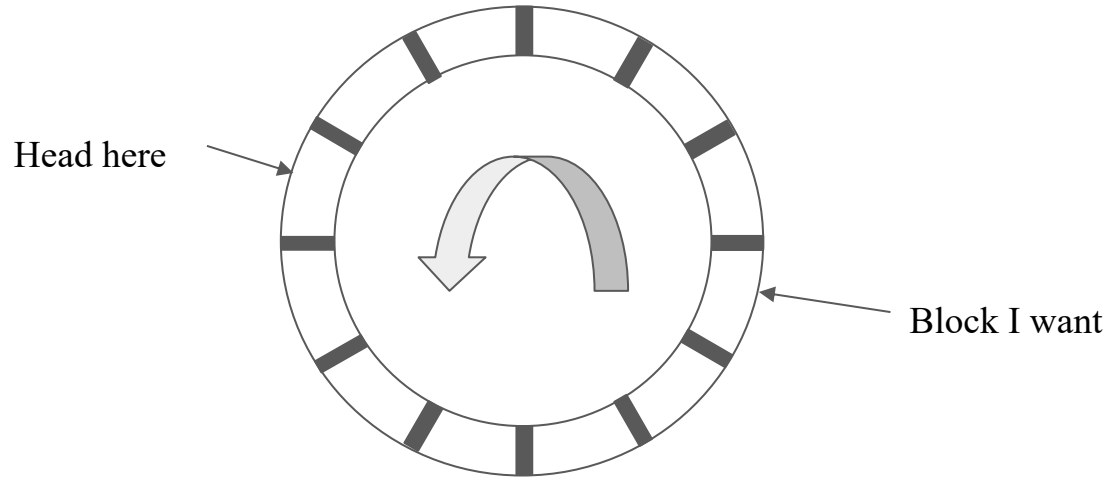
# Seek time

- The seek time depends on the distance the head has to travel to the desired cylinder



# Rotational delay

- The time can range from 0 to the time to rotate the disk once



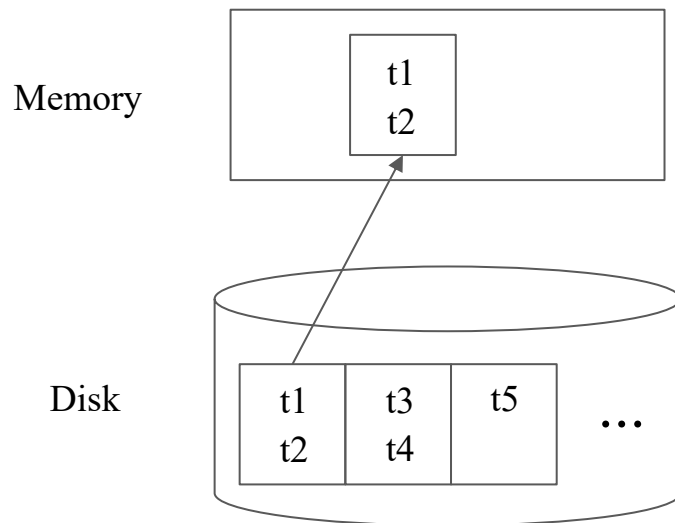
# Relative times

- Seek time
  - Disk: 1~15ms
  - Solid-state drive (SSD): 0.08~0.16ms
- Rotational delay
  - Disk: 0~10ms (on average, 1/2 rotation)
  - SSD: 0ms
- Transfer time
  - Disk: < 1ms for 4KB block
  - SSD: several times faster than disk
- Other delays
  - CPU time, contention for controller/bus/memory
  - Typically 0



# I/O model of computation

- Algorithm time  $\approx$  Number of disk I/Os
- Time to read a block from disk  $\gg$  time to search a record within that block



# Exercise #1

- Consider a 500GB hard disk with the following performance characteristics
  - 5000 revolution-per-minute (RPM) rotation rate
  - 200 cylinders
  - Takes  $1 + (t / 20)$  milliseconds to move heads  $t$  cylinders
  - 100MB/s transfer rate
- What is the average time to read a 1MB block from the hard disk?
  - Assumes that the head travels 100 cylinders on average
  - On average the disk rotates half a circle

# Speeding up disk access

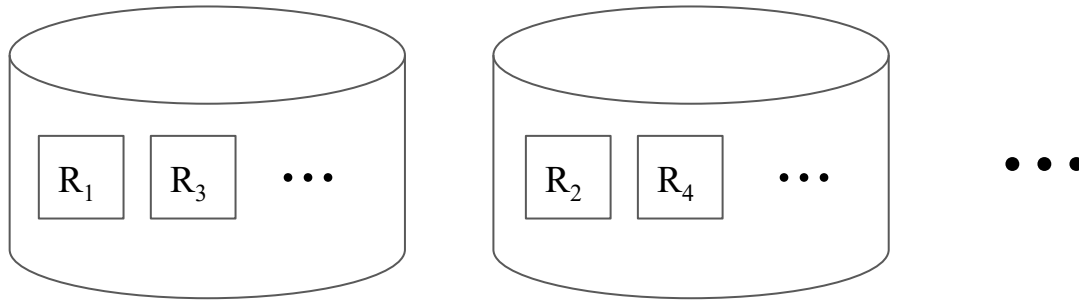
- The previous analysis was on random accesses
- There are several techniques for decreasing average disk access time

# Organize data by cylinder

- Store relation in one or more adjacent cylinders
- Saves seek time and rotational latency

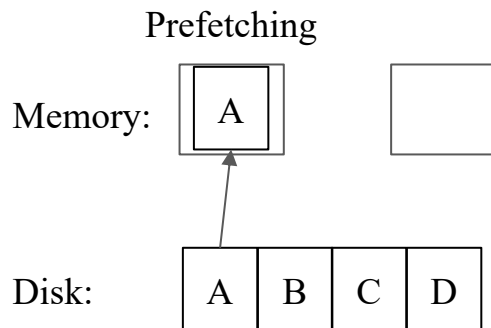
# Use multiple disks

- Stripe the blocks of a relation across multiple disks
- Reduces disk access time



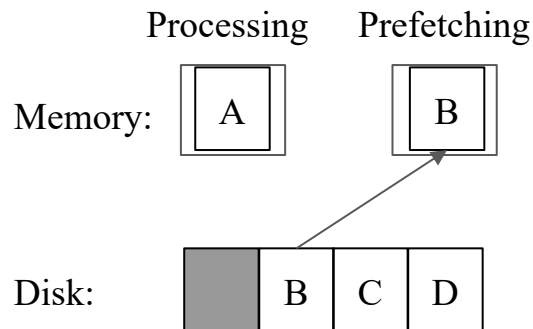
# Prefetching/Double buffering

- Predict block request order and load into memory before needed
- Reduces average block access time



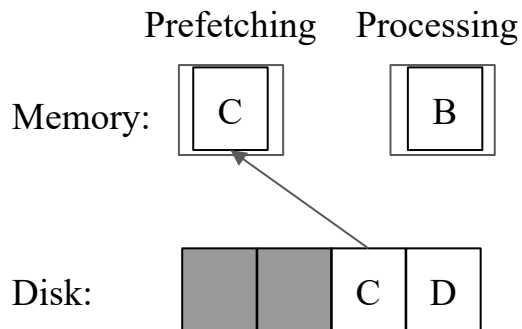
# Prefetching/Double buffering

- Predict block request order and load into memory before needed
- Reduces average block access time



# Prefetching/Double buffering

- Predict block request order and load into memory before needed
- Reduces average block access time



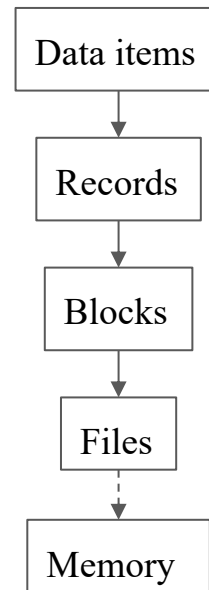


## Exercise #2

- Suppose
  - $P$  = processing time / block
  - $R$  = I/O time / block
  - $N$  = number of blocks
- If  $P \geq R$ , what is the processing time of
  - Single buffering
  - Double buffering

# File system structure

- Now let's look at how disks are used to store databases
- A tuple is represented by a record, which consists of consecutive bytes in a disk block
- Records can be fixed-length or variable-length

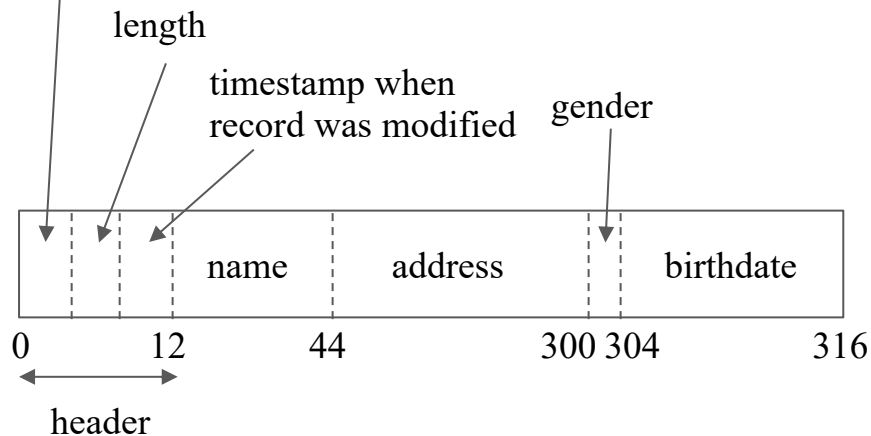


# Fixed-length records

- Each record consists of a header and fixed-length region of record's information
- It is common for field addresses to be multiples of 4 or 8 to align data for efficient reading/writing of main memory (a CPU accesses memory one word at a time)

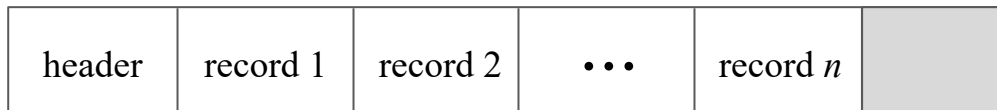
```
CREATE TABLE MovieStar (  
  name          CHAR(30),  
  address       CHAR(255),  
  gender        CHAR(1),  
  birthdate     DATE  
);
```

pointer to schema for finding  
fields of the record



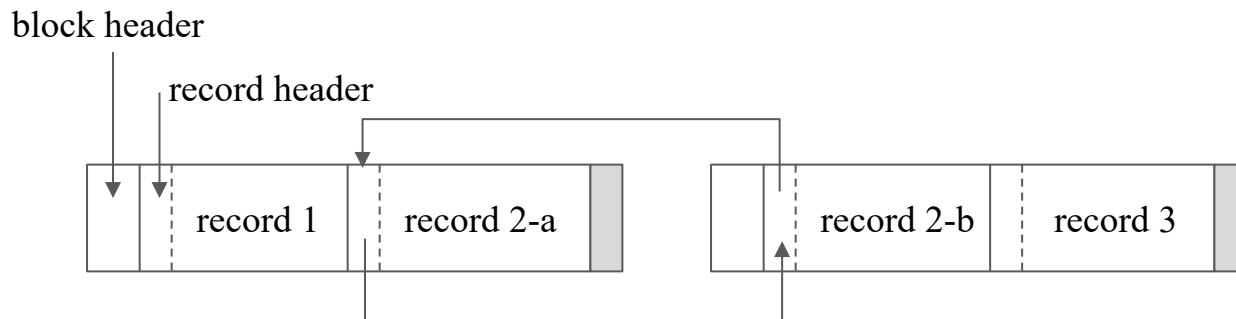
# Packing fixed-length records into blocks

- Records are stored in blocks, which are moved into main memory
- A block header contains:
  - Links to other blocks
  - Relation the tuples belong to
  - A directory of record offsets in block
  - Timestamp of the block's last modification or access



# Records that do not fit in a block

- Use spanned records to store values larger than blocks (e.g., videos)
- Each record fragment header contains information whether it is a fragment, whether it is the first or last fragment, and pointers to next/previous fragments



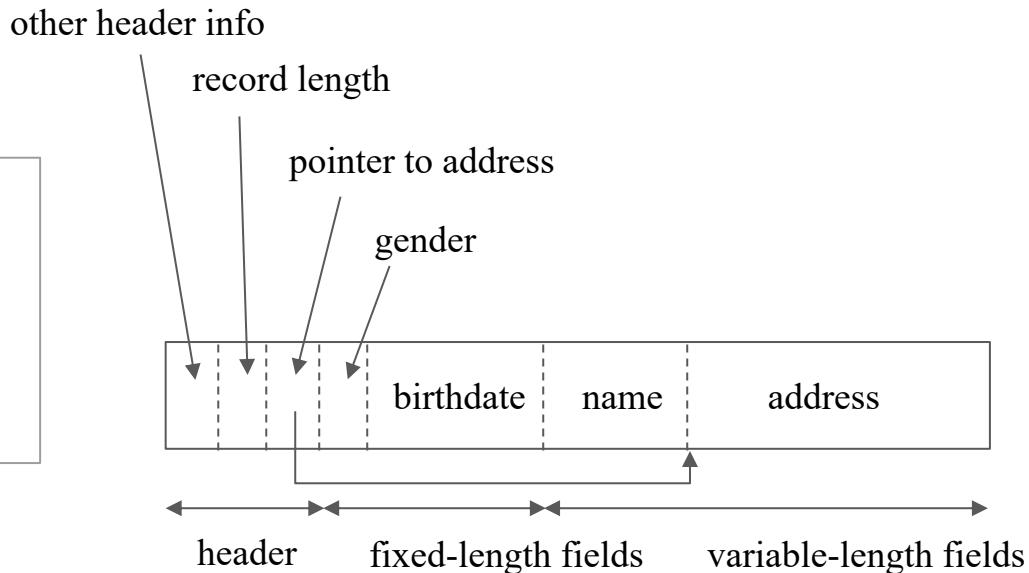
# Variable-length records

- Some records may not have a fixed schema with a list of fixed-length fields
  - e.g., VARCHAR
  - other data models (e.g., semi-structured)

# Records with variable-length fields

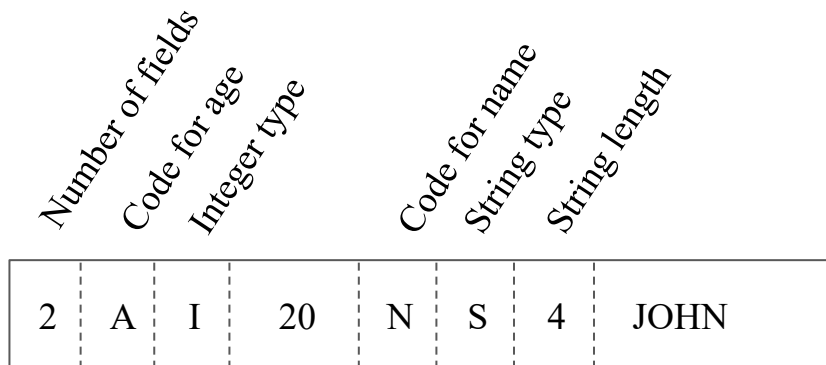
- Put all fixed-length fields ahead of the variable-length fields

```
CREATE TABLE MovieStar (  
  name          VARCHAR(30),  
  address       VARCHAR(100),  
  gender        CHAR(1),  
  birthdate     DATE  
);
```



# Variable-format records

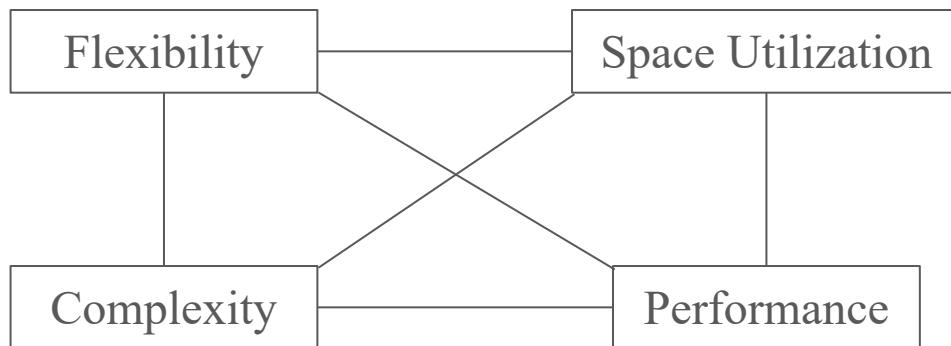
- Records may not have a fixed schema (e.g., JSON)
- Use tagged fields to make record self describing





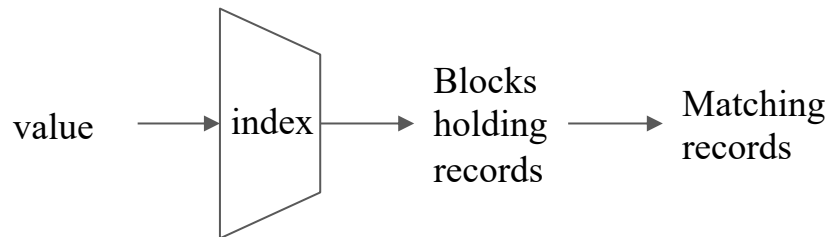
# Summary

- Many ways to store data on disk!



# Index

- A data structure that takes field values and quickly finds records containing them
- Can find tuples of a relation without scanning the entire database



# Reading Materials

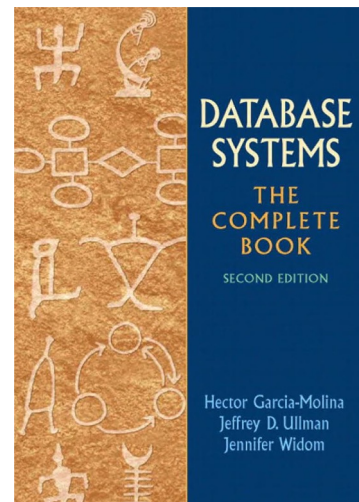
Database Systems: The Complete Book (2nd edition)

- Chapter 14: Index Structures

Supplementary materials

Fundamental of Database Systems (7th Edition)

- Chapter 17 - Indexing Structures for Files and Physical Database Design



Acknowledgement: The following slides have been adapted from EE477 (Database and Big Data Systems) taught by Steven Whang.

# Using Indexes in SQL

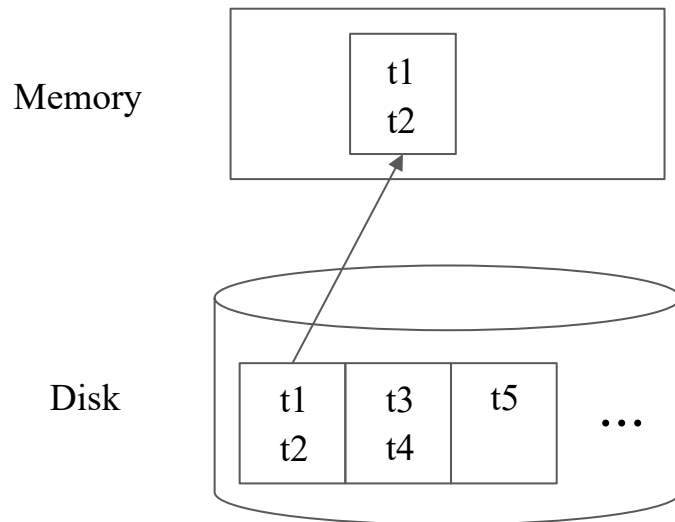
- An index is used to efficiently find tuples with certain values of attributes
- An index may speed up lookups and joins
- However, every built index makes insertions, deletions, and updates to relation more complex and time-consuming

```
CREATE INDEX KeyIndex ON Movies(title, year);
```

```
DROP INDEX KeyIndex;
```

# Simple cost model

- Multiple tuples are stored in blocks on disk
- Every block needed is always retrieved from disk
- Disk I/Os are expensive

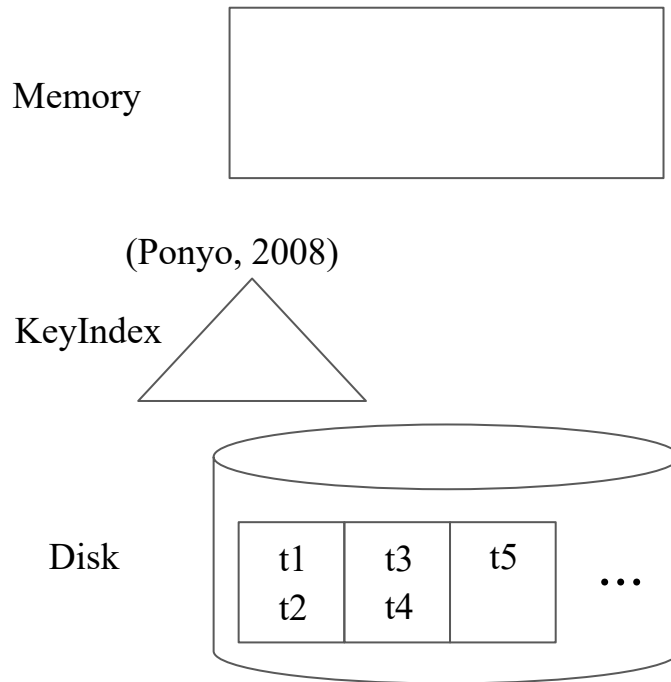


# Index on a key

- An index on a key is often useful
- Retrieve at most one block to memory for tuple
  - Possibly other blocks for the index itself

```
CREATE INDEX KeyIndex ON Movies(title, year);
```

```
SELECT *  
FROM Movies  
WHERE title = 'Ponyo' and year = 2008;
```

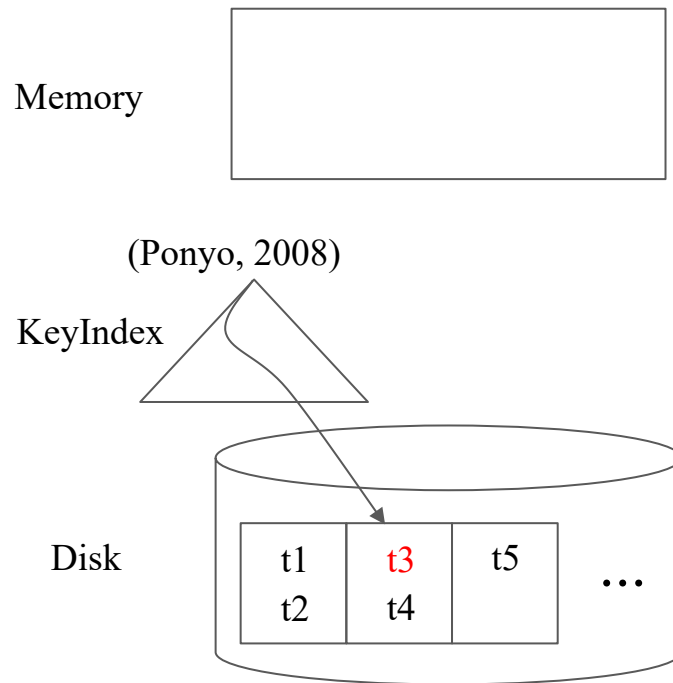


# Index on a key

- An index on a key is often useful
- Retrieve at most one block to memory for tuple
  - Possibly other pages for the index itself

```
CREATE INDEX KeyIndex ON Movies(title, year);
```

```
SELECT *  
FROM Movies  
WHERE title = 'Ponyo' and year = 2008;
```

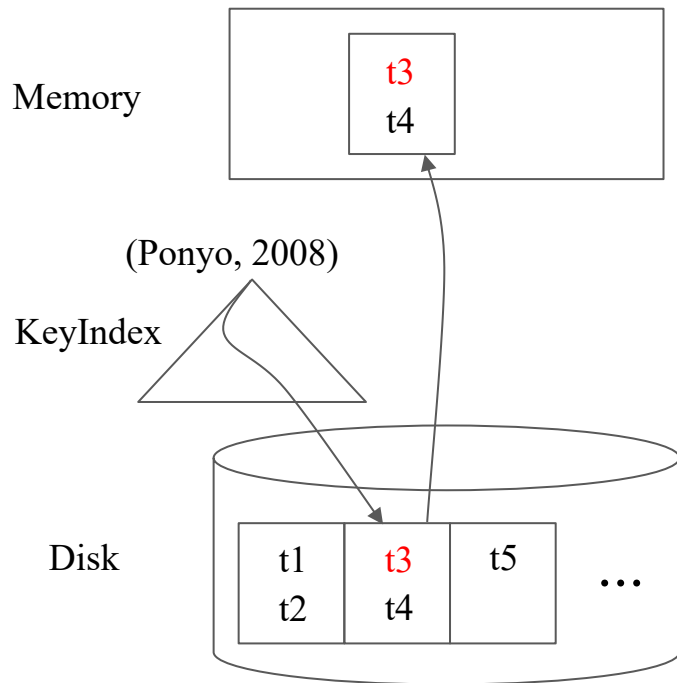


# Index on a key

- An index on a key is often useful
- Retrieve at most one block to memory for tuple
  - Possibly other pages for the index itself

```
CREATE INDEX KeyIndex ON Movies(title, year);
```

```
SELECT *  
FROM Movies  
WHERE title = 'Ponyo' and year = 2008;
```





# Indexes can be used in joins

- With the right indexes, the join below only requires 2 page reads for the tuples
  - And possibly a small number of other pages for accessing the indexes

```
CREATE INDEX MIndex ON Movies(title, year);
```

```
CREATE INDEX MEIndex ON MovieExec(cert#);
```

```
SELECT name  
FROM Movies, MovieExec  
WHERE title = 'Ponyo' and year = 2008  
and producerC# = cert#;
```

Memory

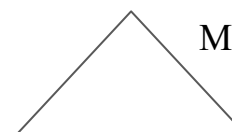


(Ponyo, 2008)

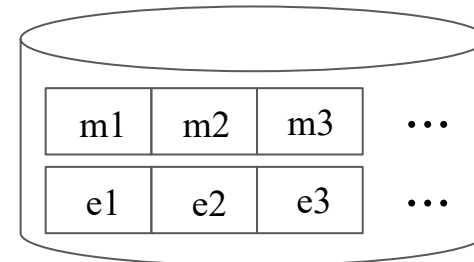
MIndex



MEIndex



Disk



# Indexes can be used in joins

- With the right indexes, the join below only requires 2 page reads for the tuples
  - And possibly a small number of other pages for accessing the indexes

```
CREATE INDEX MIndex ON Movies(title, year);
```

```
CREATE INDEX MEIndex ON MovieExec(cert#);
```

```
SELECT name  
FROM Movies, MovieExec  
WHERE title = 'Ponyo' and year = 2008  
and producerC# = cert#;
```

Memory

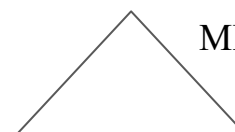


(Ponyo, 2008)

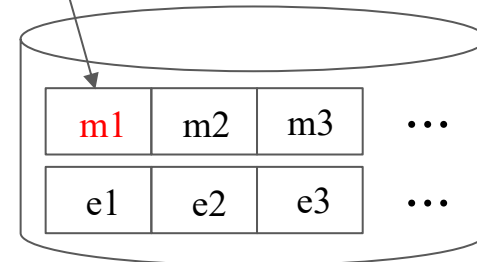
MIndex



MEIndex



Disk



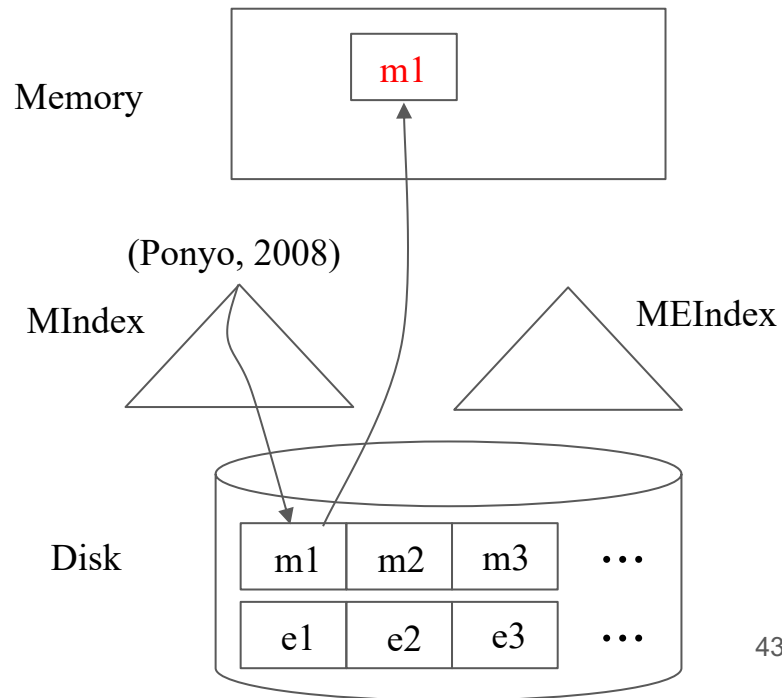
# Indexes can be used in joins

- With the right indexes, the join below only requires 2 page reads for the tuples
  - And possibly a small number of other pages for accessing the indexes

```
CREATE INDEX MIndex ON Movies(title, year);
```

```
CREATE INDEX MEIndex ON MovieExec(cert#);
```

```
SELECT name  
FROM Movies, MovieExec  
WHERE title = 'Ponyo' and year = 2008  
and producerC# = cert#;
```



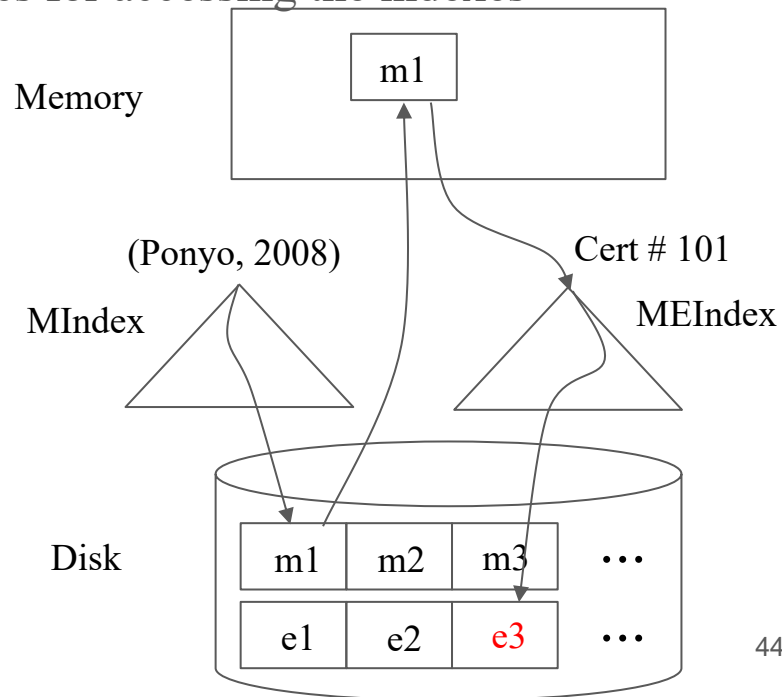
# Indexes can be used in joins

- With the right indexes, the join below only requires 2 page reads for the tuples
  - And possibly a small number of other pages for accessing the indexes

```
CREATE INDEX MIndex ON Movies(title, year);
```

```
CREATE INDEX MEIndex ON MovieExec(cert#);
```

```
SELECT name  
FROM Movies, MovieExec  
WHERE title = 'Ponyo' and year = 2008  
and producerC# = cert#;
```



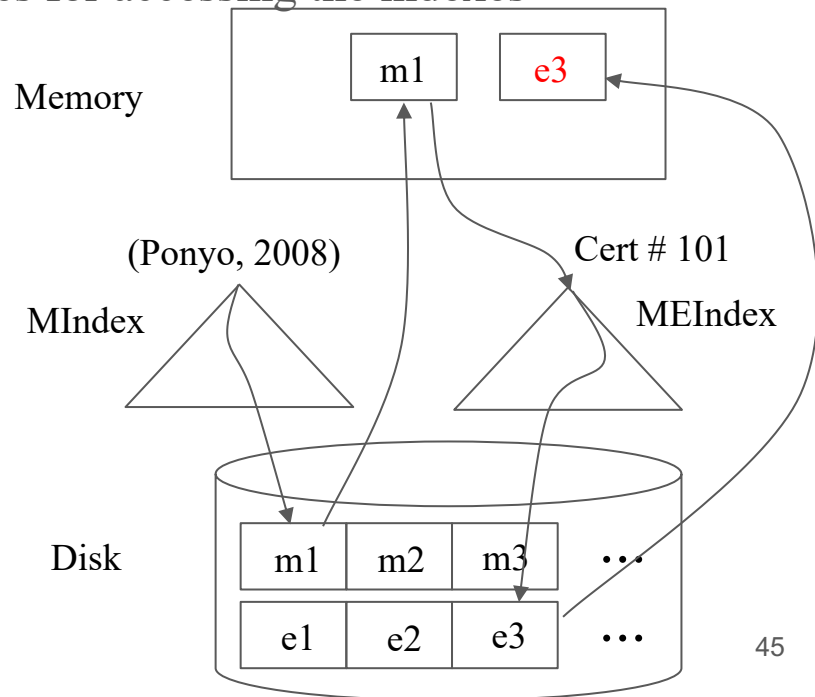
# Indexes can be used in joins

- With the right indexes, the join below only requires 2 page reads for the tuples
  - And possibly a small number of other pages for accessing the indexes

```
CREATE INDEX MIndex ON Movies(title, year);
```

```
CREATE INDEX MEIndex ON MovieExec(cert#);
```

```
SELECT name  
FROM Movies, MovieExec  
WHERE title = 'Ponyo' and year = 2008  
and producerC# = cert#;
```



# So, how are indexes implemented in DBMS?

- Index-structure basics
  - Dense index, sparse index, secondary index
- B-Tree
  - For range lookups
- Hashing
  - For point lookups