

1

## Announcements

1. Project 1 due Friday

2. My Office Hours

- ▷ Today, right after lecture
- ▷ Friday on zoom

# Common Table Expressions (CTEs)

```
SELECT product,  
        SUM(price * quantity) AS TotalSales  
FROM Purchase  
WHERE date > '10/1/2005'  
GROUP BY product
```

(Why is Select on top? Similar to Functions --  $f \Rightarrow$  output on top)

Give it a name

```
WITH ProductSales AS  
(SELECT product,  
        SUM(price * quantity) AS TotalSales  
 FROM Purchase  
 WHERE date > '10/1/2005'  
 GROUP BY product  
)
```

Useful for readability -- e.g., sub-queries, chaining queries

3

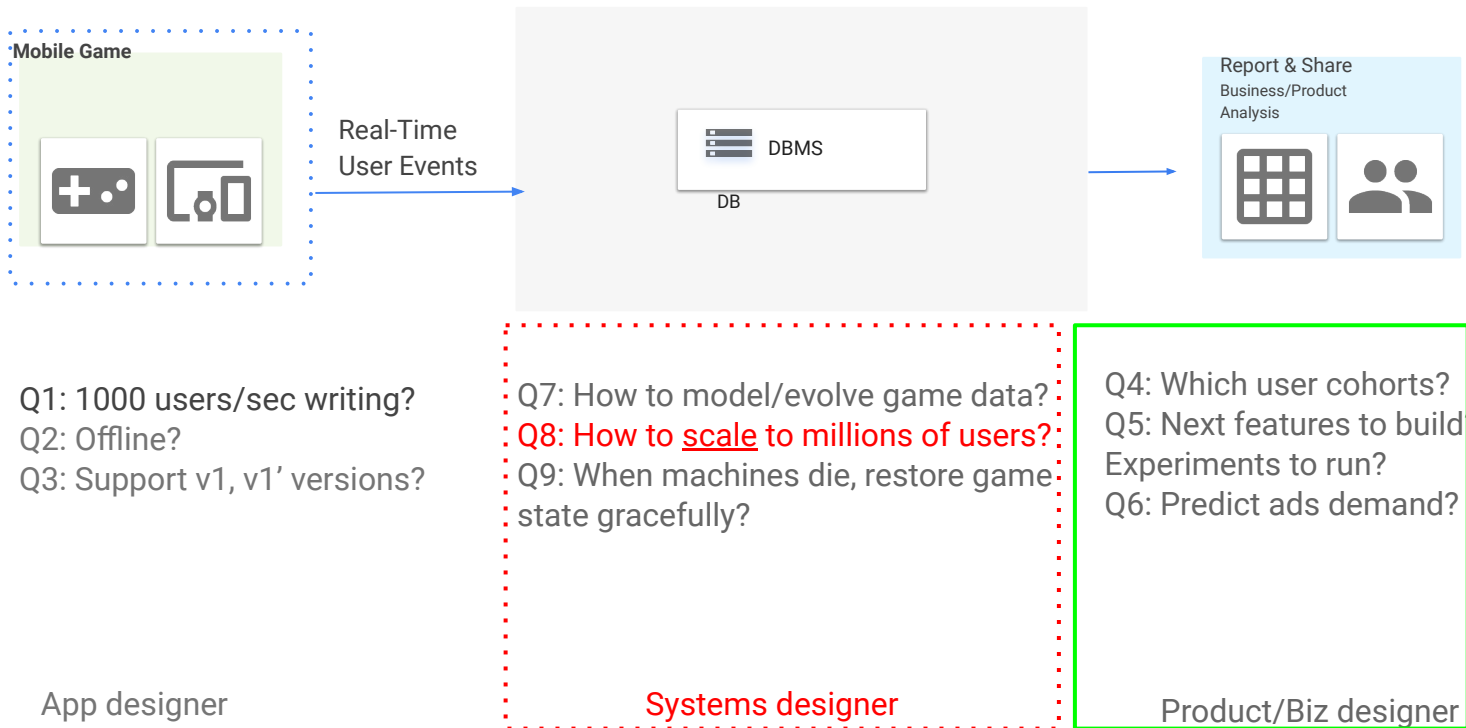


How?

# Example Game App

DB v0

(Recap lectures)





# Scale: Logical → Physical DB

5

Company(CName, StockPrice, Date, Country)

Logical Table

Next: How to store table in physical storage 'files'?  
How to access rows/columns?  
(e.g., disk, RAM, clusters)

Logical →

Physical?

Col3				
Company				
	CName	Date	Price	Country
Row1	AAPL	Oct1	101.23	USA
	AAPL	Oct2	102.25	USA
Row3	AAPL	Oct3	101.6	USA
	GOOG	Oct1	201.8	USA
Row5	GOOG	Oct2	201.61	USA
	GOOG	Oct3	202.13	USA
	Alibaba	Oct1	407.45	China
Row8	Alibaba	Oct2	400.23	China



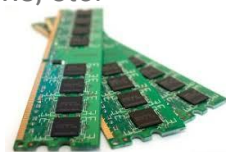
So far... how to run SQL on "logical tables?"

Logical →

Physical?

Small tables (e.g., < 1 GB)? “Easy” with tools you already know

- CS Principles
  - Data structures? Linked lists, arrays, trees, hash tables
  - Algorithms? Sorting, Hashing, Dynamic Programming, Graph algorithms, etc.
- How?
  - I/O model  $\Rightarrow$  Work with data in RAM;
  - Language/library? python/c++, pandas libraries, or build your own

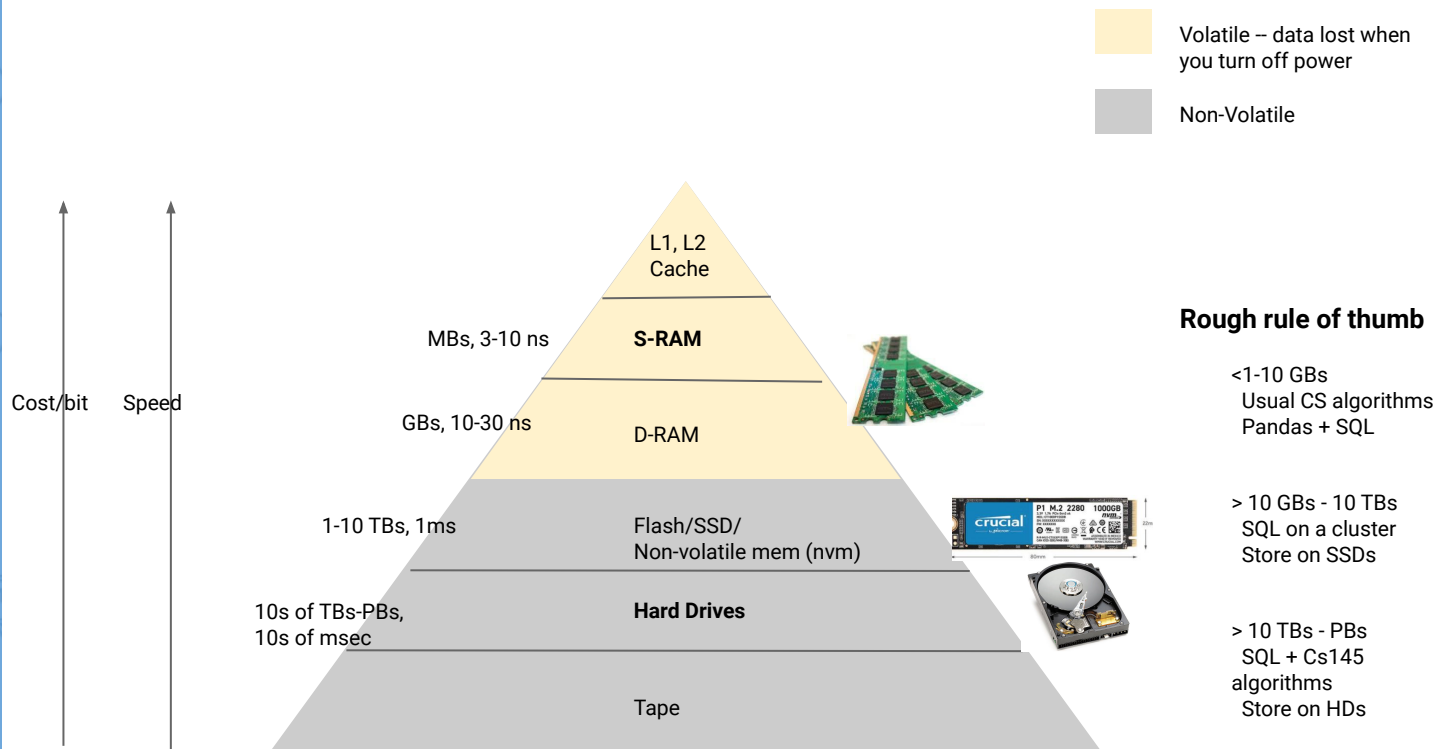


Big tables? (e.g., TBs, PetaBytes?)

- How? Expand our tool set with cs145 + advanced systems classes!
  - I/O model  $\Rightarrow$  RAM + HD/SSDs + Clusters
  - Language/library? “Data language” + “I/O-scaling libraries”

7

# IO Hierarchy



⇒ Rest of cs145: Focus on simplified RAM + Disk model (+Clouds)  
(learn tools for other IO models)

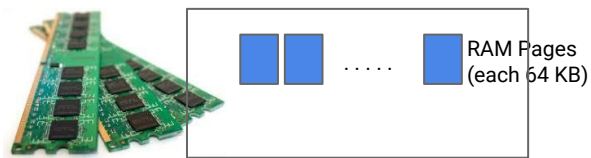
# In this Section

(Next weeks:  
Scale, Scale, Scale)

1. IO Model
2. Data layout
3. Indexing
4. Organizing Data and Indices

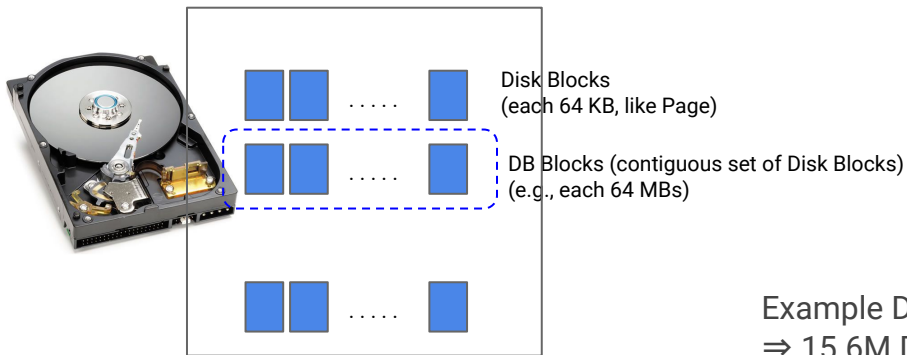


# Basic IO Model for Reads



Example RAM size: 64 GB RAM  
⇒ 1 million 64 KB pages

↑ Read in (Page in from disk)



Example Disk size: 1 TB  
⇒ 15.6M Disk Blocks (= 1 TB/64 KB)  
  
⇒ 15.6K DB Blocks (@ 64 MB/DB block)

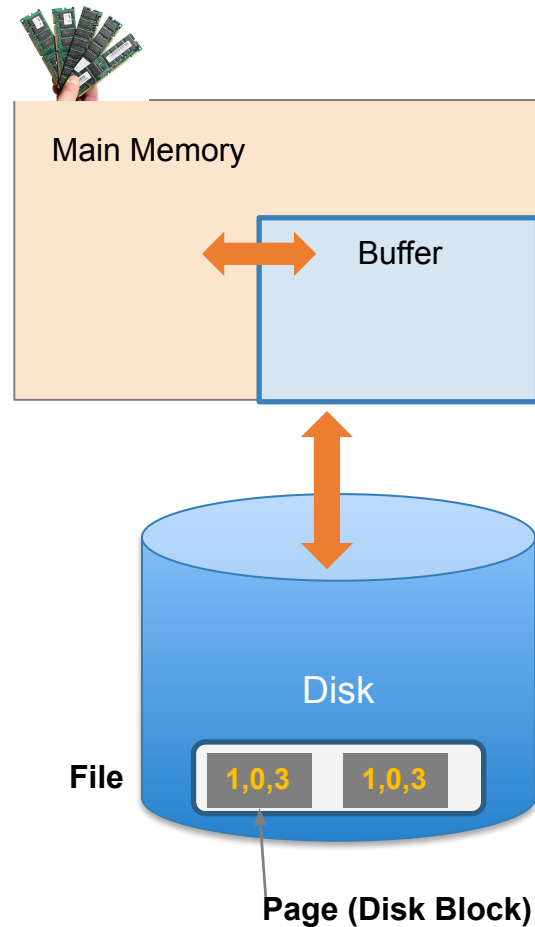
In DBs, Page and Disk Block are usually same size.  
⇒ In this class, we'll use them interchangeably

# DB Buffer in RAM

A buffer is a part of physical memory used to store intermediate data between disk and processes

Database maintains its own buffer (Why? Doesn't the OS already do this?)

- DB knows more about access patterns
- Recovery and logging require ability to flush to disk
- Buffer Manager handles page replacement policies



## In this Section

(Next weeks:  
Scale, Scale, Scale)

1. IO Model
2. Data layout
3. Indexing
4. Organizing Data and Indices

# Data Layout

Company(CName, StockPrice, Date, Country)

Logical Table

How to store table in physical storage 'files'?  
(e.g., disk, RAM)

Col3

Company				
	CName	Date	Price	Country
Row1	AAPL	Oct1	101.23	USA
	AAPL	Oct2	102.25	USA
Row3	AAPL	Oct3	101.6	USA
	GOOG	Oct1	201.8	USA
Row5	GOOG	Oct2	201.61	USA
	GOOG	Oct3	202.13	USA
	Alibaba	Oct1	407.45	China
Row8	Alibaba	Oct2	400.23	China
...				
Row Billion				



# Data Layout

Company(CName, StockPrice, Date, Country)

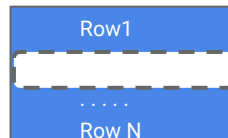
Logical Table

Col3

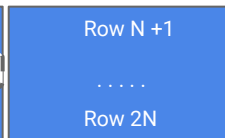
	Company			
	CName	Date	Price	Country
Row1	AAPL	Oct1	101.23	USA
	AAPL	Oct2	102.25	USA
Row3	AAPL	Oct3	101.6	USA
	GOOG	Oct1	201.8	USA
Row5	GOOG	Oct2	201.61	USA
	GOOG	Oct3	202.13	USA
	Alibaba	Oct1	407.45	China
Row8	Alibaba	Oct2	400.23	China

Row  
Billion

Page



Page



Page ... n  
(RAM/Disk)



Row based storage  
(aka Row Store)

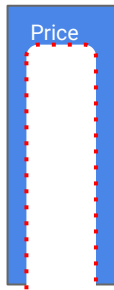
Page



Page



Page



Page



Column based storage  
(aka Column Store)

# Data Layout

Company(CName, StockPrice, Date, Country)

Row based storage  
(aka Row Store)

- Easy to retrieve and modify full tuple/row
- Classic way to organize data

Col3

	Company			
	CName	Date	Price	Country
Row1	AAPL	Oct1	101.23	USA
	AAPL	Oct2	102.25	USA
Row3	AAPL	Oct3	101.6	USA
	GOOG	Oct1	201.8	USA
Row5	GOOG	Oct2	201.61	USA
	GOOG	Oct3	202.13	USA
	Alibaba	Oct1	407.45	China
Row8	Alibaba	Oct2	400.23	China

Column based storage  
(aka Column Store)

- Aggregation queries – e.g., AVG(Price)
- Compression – e.g., see Date column
- Scale to machine clusters – distribute columns to different machines
- Only retrieve columns you need for query
- Cons: Updates are more work

Tradeoffs on 'Workloads:'

1. Analytics: Lots of data, many exploratory queries on few columns, e.g., Youtube analytics
2. Transactions: Good combination of reads and writes, e.g., Delta Airlines

# Example

## Origin Story of BigQuery (Dremel)

WebPage(URL, PageRank, Language, NumVisits, HTML)

Google index of Web Pages (~2005)

URL: 100 bytes

PageRank: 8 bytes

Language: 4 bytes

Number of visitors: 4 bytes

HTML: 2 MBs \* 5 versions ← (the big column)

⇒ Overall size = ~10 MBs/URL, stored in row format

Use case: What's PageRank of popular pages?

- E.g., select AVG(PageRank) ... where NumVisits > 100
- **Hours** to run query over 1 billion URLs. Why?
  - ⇒ Row based layout: Processing 10 MB\*1 billion urls
  - ⇒ Column based layout: Need to process only 12 bytes \* 1 billion urls (~1 million times faster)
- **Core idea:** Exploratory queries usually focus on a few columns

## In this Section

(Next weeks:  
Scale, Scale, Scale)

1. IO Model
2. Data layout
3. Indexing
4. Organizing Data and Indices



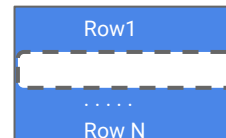
# How to find the right data fast?

Company(CName, StockPrice, Date, Country)

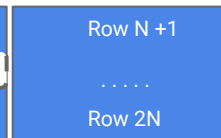
Col3

	Company			
	CName	Date	Price	Country
Row1	AAPL	Oct1	101.23	USA
	AAPL	Oct2	102.25	USA
Row3	AAPL	Oct3	101.6	USA
	GOOG	Oct1	201.8	USA
Row5	GOOG	Oct2	201.61	USA
	GOOG	Oct3	202.13	USA
	Alibaba	Oct1	407.45	China
Row8	Alibaba	Oct2	400.23	China

Page



Page



Page ... n  
(RAM/Disk)



Row based storage  
(aka Row Store)

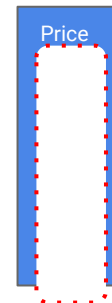
Page



Page



Page



Page



Column based storage  
(aka Column Store)

Next: How to find AAPL Prices?

## Why study Indexes?

1. Fundamental unit for DB performance
2. Core indexing ideas have become **stand-alone systems**
  - E.g., search in google.com
  - Data blobs in noSQL, Key-value stores
  - Embedded join processing

19

Example

Find Book  
in Library



Design choices?

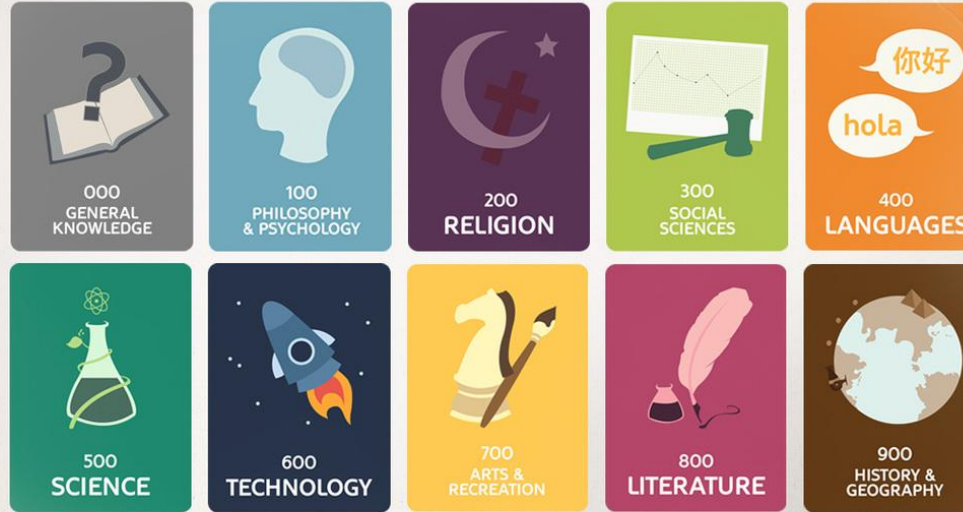
- Scan through each aisle
- Lookup pointer to book location, with librarian's organizing scheme

20

# Example

## Find Book in Library With Index

### the DEWEY DECIMAL SYSTEM



Index Cards



#### Understanding the Dewey Decimal System

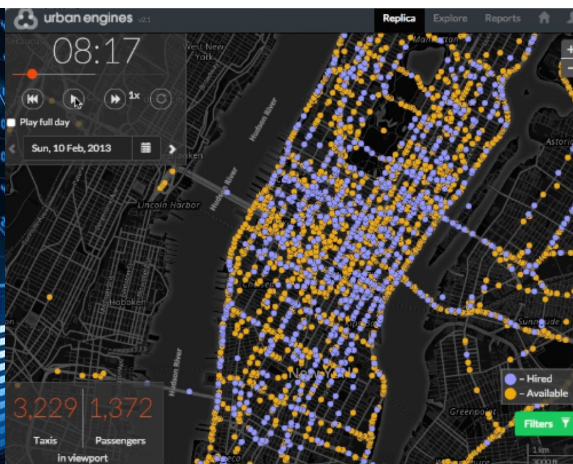
Division  
(Drawing/Decorative  
Arts)  
746.43  
Main Class  
(Art)  
Section  
(Textile Arts)  
Classification  
Within the  
Section  
(Type of Textile)

#### Algorithm for book titles

- Find right category
- Lookup Index, find location
- Walk to aisle. Scan book titles. Faster if books are sorted



# Kinds of Indexes (different data types)



Index for

- Strings, Integers
- Time series, GPS traces, Genomes, Video sequences
- Advanced: Equality vs Similarity, Ranges, Subsequences

Composites of above

# Example: Search on stocks

Company(CName, StockPrice, Date, Country)

Company			
CName	Date	Price	Country
AAPL	Oct1	101.23	USA
AAPL	Oct2	102.25	USA
AAPL	Oct3	101.6	USA
GOOG	Oct1	201.8	USA
GOOG	Oct2	201.61	USA
GOOG	Oct3	202.13	USA
Alibaba	Oct1	407.45	China
Alibaba	Oct2	400.23	China

```
SELECT *  
FROM Company  
WHERE CName like 'AAPL'
```

```
SELECT CName, Date  
FROM Company  
WHERE Price > 200
```

Q: On which attributes would you build indexes?

A: On as many subsets as you'd like. Look at query workloads.

# Example



## CName\_Index

CName	Block #	Company	CName	Date	Price	Country
AAPL	...	AAPL	AAPL	Oct1	101.23	USA
AAPL	...		AAPL	Oct2	102.25	USA
AAPL	...		AAPL	Oct3	101.6	USA
GOOG	...	GOOG	GOOG	Oct1	201.8	USA
GOOG	...		GOOG	Oct2	201.61	USA
GOOG	...		GOOG	Oct3	202.13	USA
Alibaba	...	Alibaba	Alibaba	Oct1	407.45	China
Alibaba	...		Alibaba	Oct2	400.23	China

## PriceDate\_Index

Date	Price	Block #
Oct1	101.23	
Oct2	102.25	
Oct3	101.6	
Oct1	201.8	
Oct2	201.61	
Oct3	202.13	
Oct1	407.45	
Oct2	400.23	

- Index contains search key + Block #: e.g., DB block number.
  - In general, “pointer” to where the record is stored (e.g., RAM page, DB block number or even machine + DB block)
  - Index is conceptually a table. In practice, implemented very efficiently (see how soon)
- Can have multiple indexes to support multiple search keys

# Indexes (definition)

Maps search keys to sets of rows in table

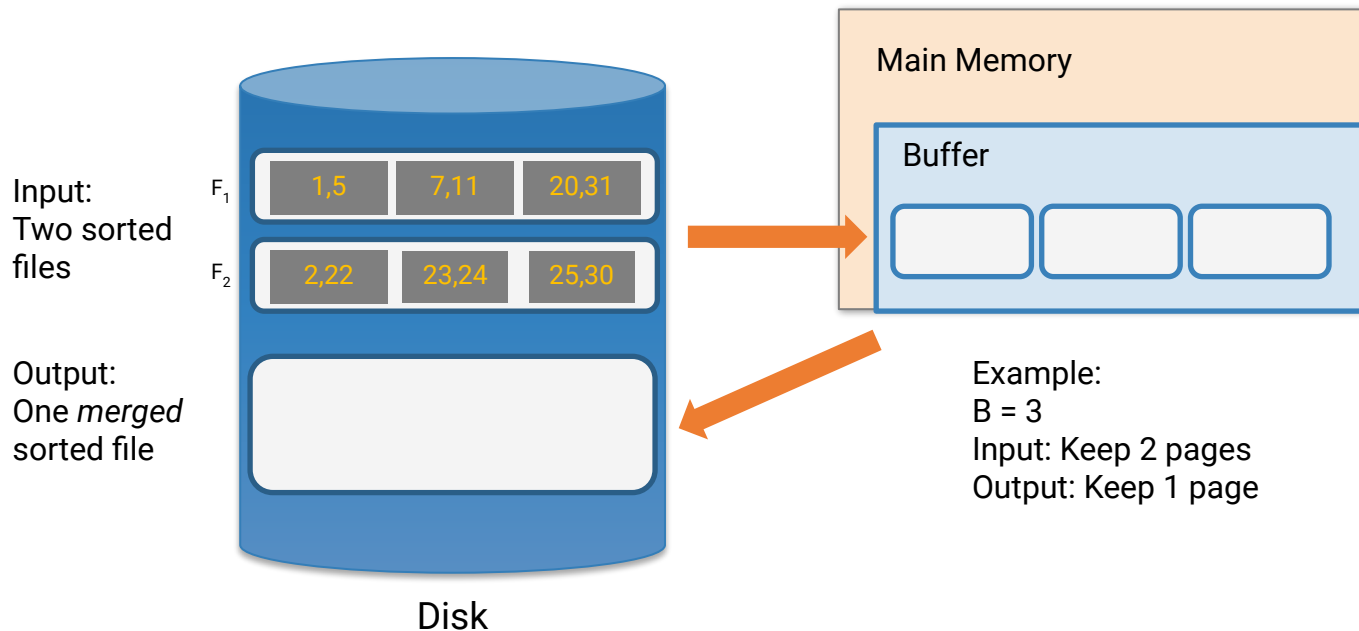
- Provides efficient lookup & retrieval by search key value (much faster than scanning all rows and filtering, usually)
- Key operations: Lookup, Insert, Delete

An index can be

- Primary: Index on a set of columns that includes unique primary key. And no duplicates
- Secondary: Non-primary index (on any set of columns) and may have duplicates [much of our focus – primary is an easier version]
- Advanced: build across rows, across tables



# External Merge Algorithm



# Covering Indexes

## PriceDate\_Index

Date	Price	Block #
Oct1	101.23	
Oct2	102.25	
Oct3	101.6	
Oct1	201.8	
Oct2	201.61	
Oct3	202.13	
Oct1	407.45	
Oct2	400.23	

An index covers for a specific query if the index contains all the needed attributes

*I.e, query can be answered using the index alone!*

The “needed” attributes are the union of those in the SELECT and WHERE clauses...

Example:

```
SELECT Date, Price  
FROM Company  
WHERE Price > 101
```

## Why study Indexes?

1. Fundamental unit for DB performance
2. Core indexing ideas have become **stand-alone systems**
  - E.g., search in google.com
  - Data blobs in noSQL, Key-value stores
  - Embedded join processing


## In this Section

(Next weeks:  
Scale, Scale, Scale)

1. IO Model
2. Data layout
3. Indexing
4. Organizing Data and Indices

## Big Scale

## Roadmap



Hashing

Sorting

Hashing-Sorting solves “all” known data scale problems :=)

+ Boost with a few patterns – Cache, Parallelize, Pre-fetch



**THE BIG IDEA**

Note: Works for Relational, noSQL  
(e.g. mySQL, postgres, BigQuery, BigTable, MapReduce, Spark, Mongo)

30

Big Scale

Roadmap

Primary data structures/algorithms

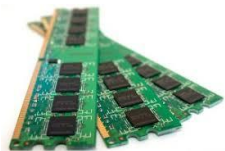
Hashing

HashTables  
( $\text{hash}_i(\text{key}) \rightarrow \text{value}$ )

Sorting

BucketSort, QuickSort  
MergeSort

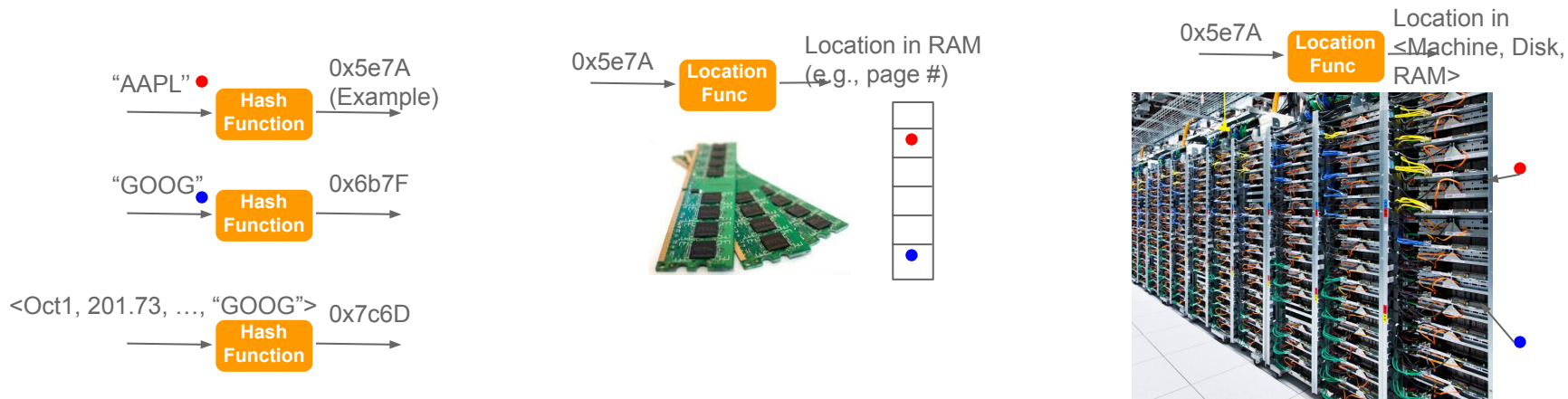
?????





# Hashing

# Recall: Hashing

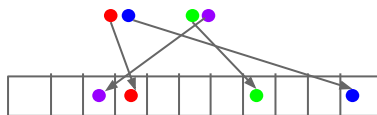


- Magic of hashing:
  - A hash function  $h_B$  maps into  $[0, B-1]$ , nearly uniformly
  - Also called sharding function
- A hash collision is when  $x \neq y$  but  $h_B(x) = h_B(y)$ 
  - Note however that it will never occur that  $x = y$  but  $h_B(x) \neq h_B(y)$

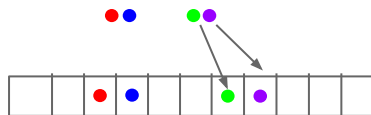


# Hashing ideas for scale

- Idea: Multiple hash functions (uncorrelated to spread data)
  - $h_i(x), h_{i+1}(x), h_{i+2}(x), h_{i+3}(x), \dots$
- Idea: Locality sensitive hash functions (for high dimensional data)
  - Special class of hash functions to keep spread 'local'



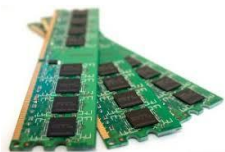
Regular hash functions  
(spread all over)



Locality Sensitive Hash (LSH) functions  
(spread in closer buckets, with high probability)

# Big Scaling (with Indexes)

## Roadmap



### Primary data structures/algorithms

#### Hashing

HashTables  
( $\text{hash}_i(\text{key}) \rightarrow \text{value}$ )

#### Sorting

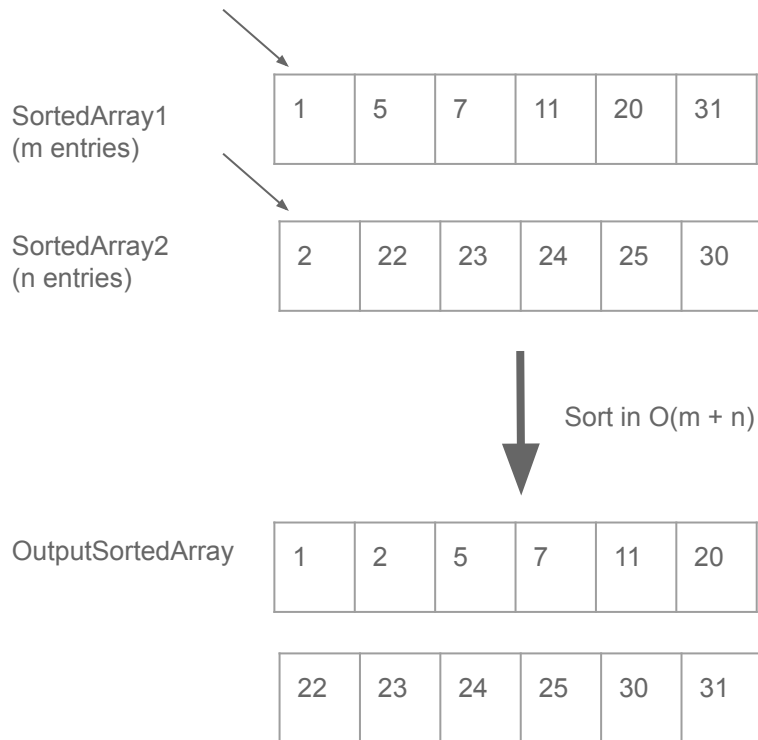
BucketSort, QuickSort  
MergeSort2Lists,  
MergeSort

MergeSortedFiles,  
MergeSort



# Sorting 1: External Merge Algorithm

## MergeSortedFiles (in RAM)



## Key (Simple) Idea

To find an element that is no larger than all elements in two lists, one only needs to compare minimum elements from each list.

If:

$$A_1 \leq A_2 \leq \dots \leq A_N$$

$$B_1 \leq B_2 \leq \dots \leq B_M$$

Then:

$$\text{Min}(A_1, B_1) \leq A_i$$

$$\text{Min}(A_1, B_1) \leq B_j$$

for  $i=1 \dots N$  and  $j=1 \dots M$

## Challenge: Merging Big Files with Small Memory

How do we *efficiently* merge two sorted files when both are much larger than our main memory buffer?

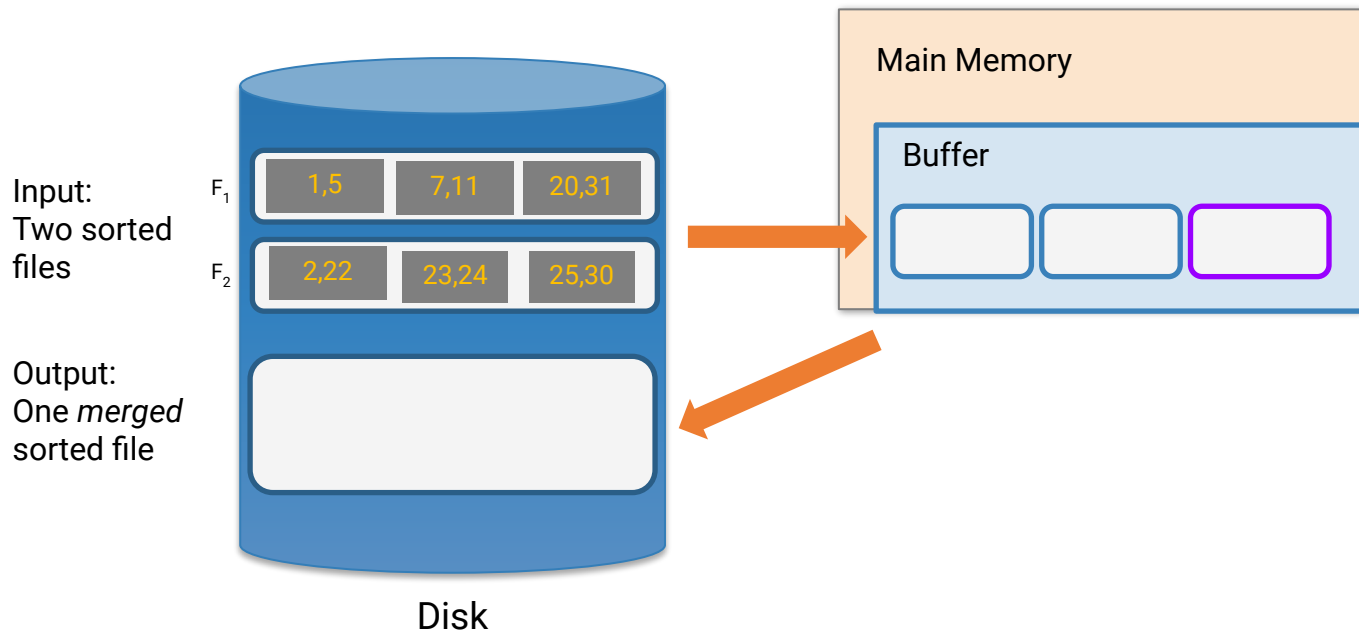
Key point: Disk IO (R/W) dominates the algorithm cost

Our first example of an “**IO aware**” algorithm / cost model

# External Merge Algorithm

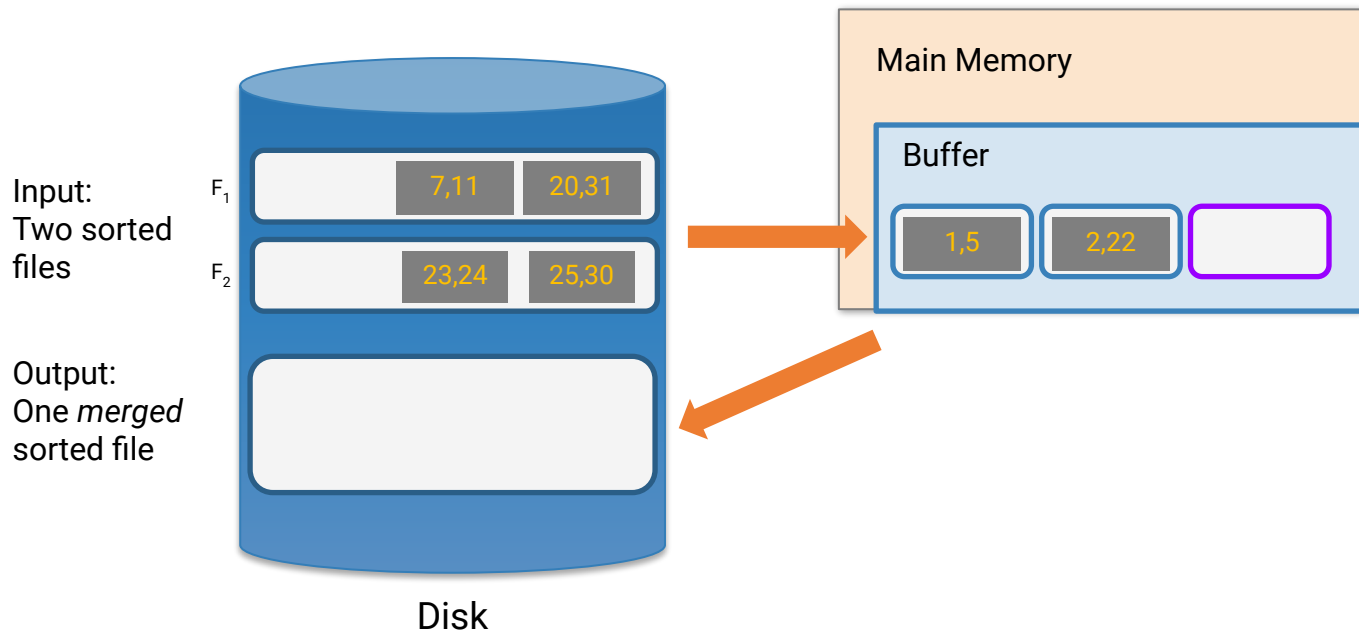
- Input: 2 sorted lists of length  $M$  and  $N$
- Output: 1 sorted list of length  $M + N$
- Required: At least 3 Buffer Pages
- IOs:  $2(M+N)$

# External Merge Algorithm

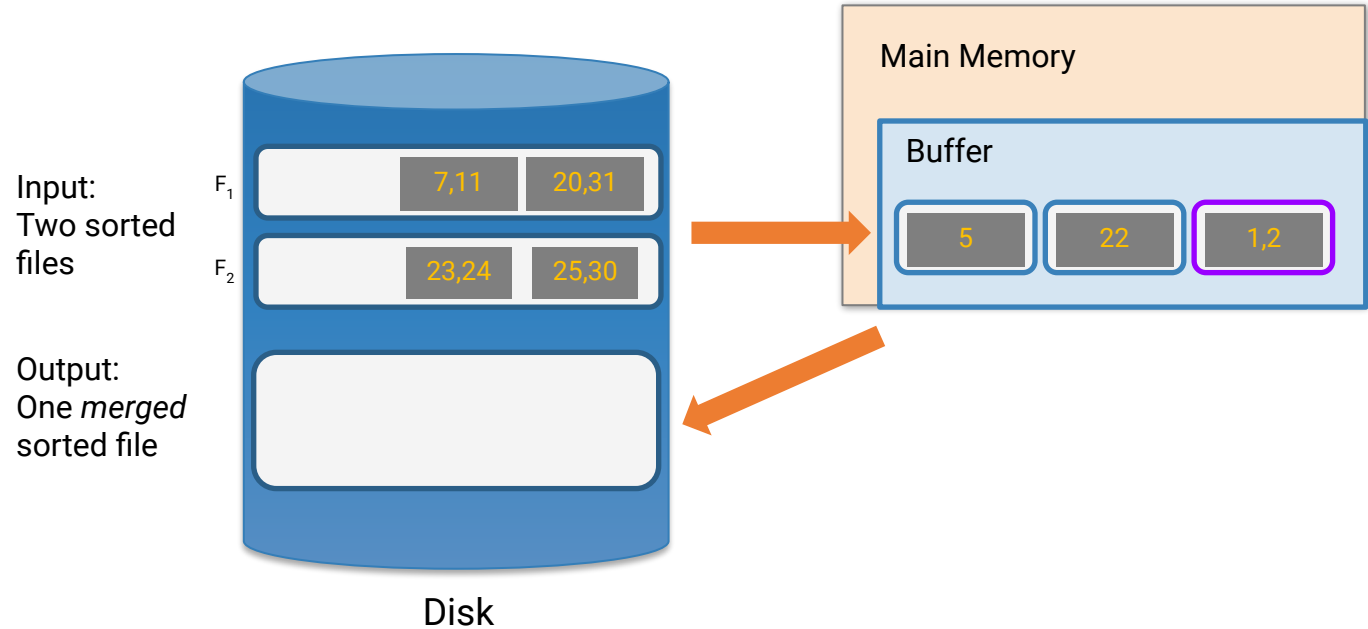




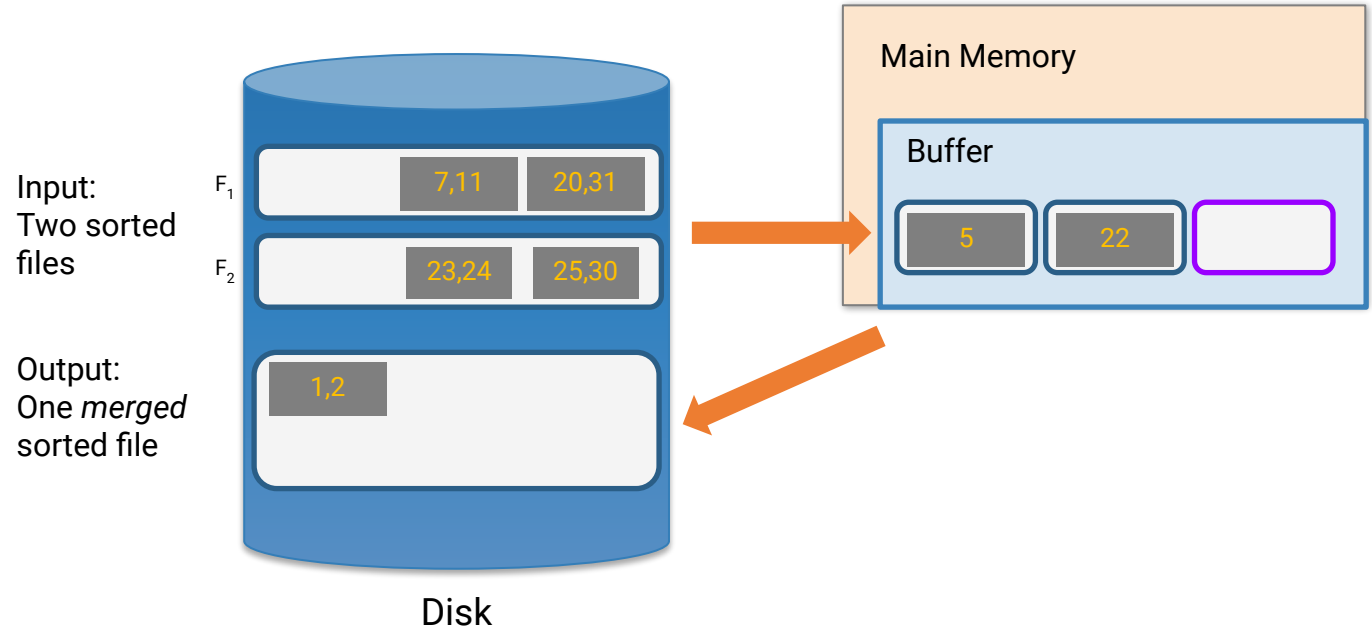
# External Merge Algorithm



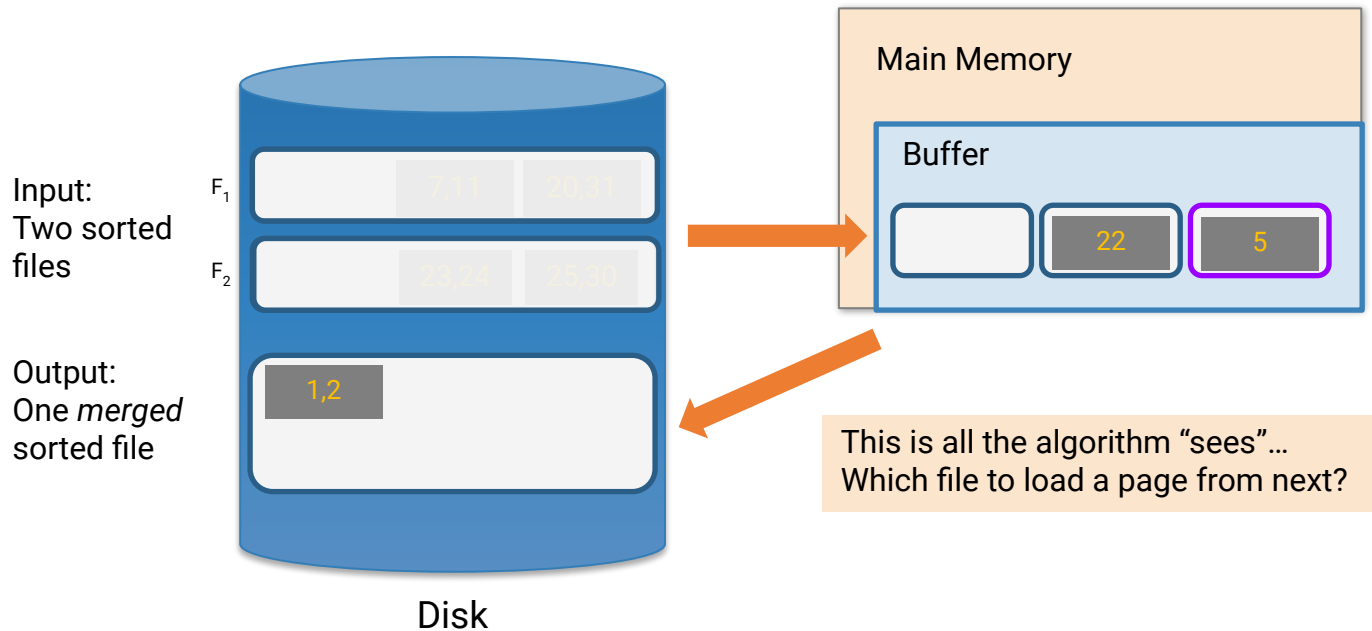
# External Merge Algorithm



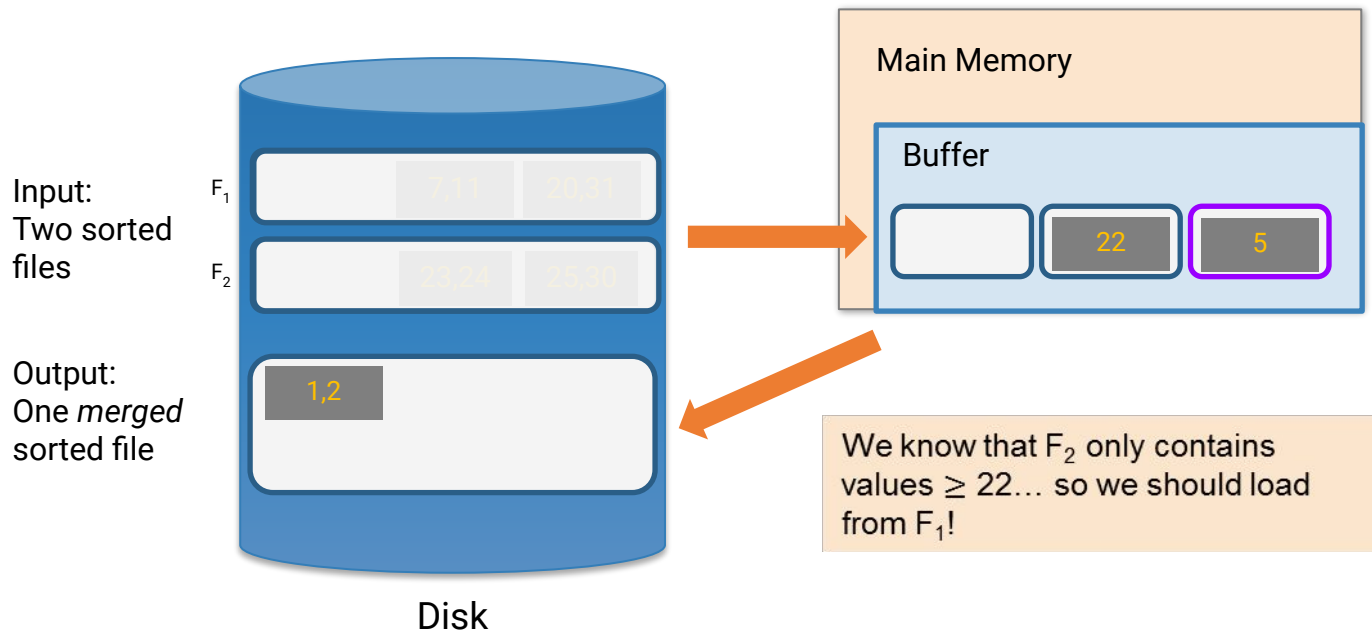
# External Merge Algorithm



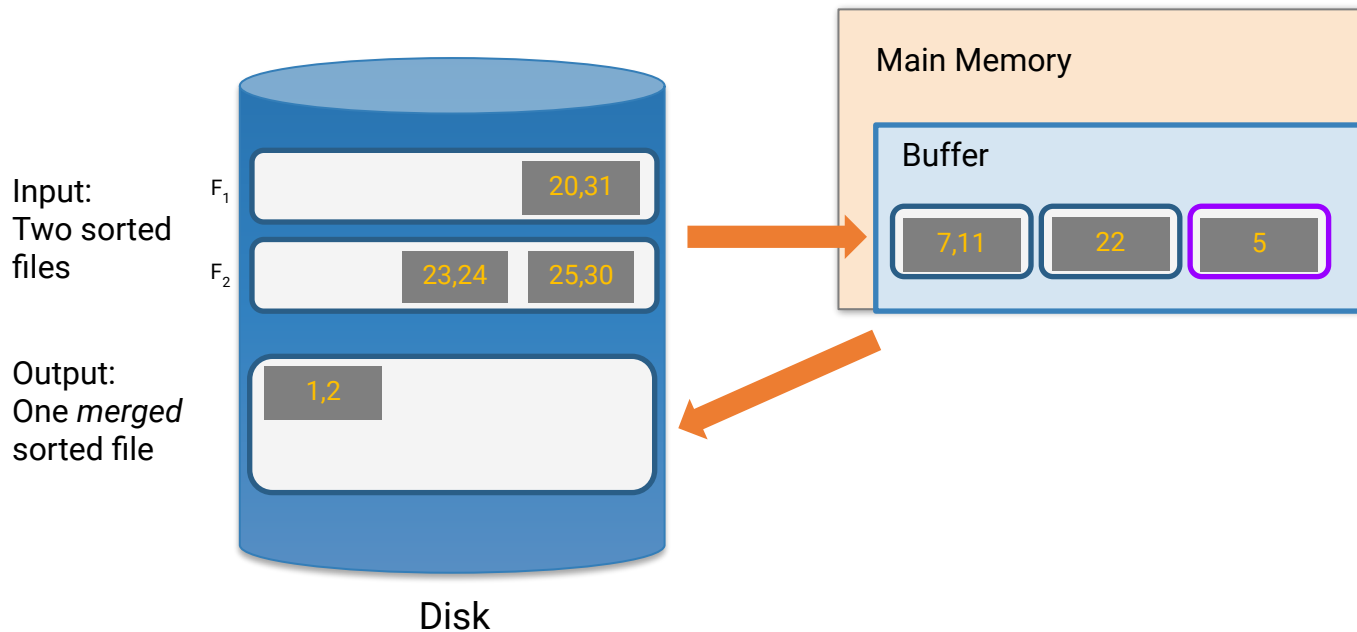
# External Merge Algorithm



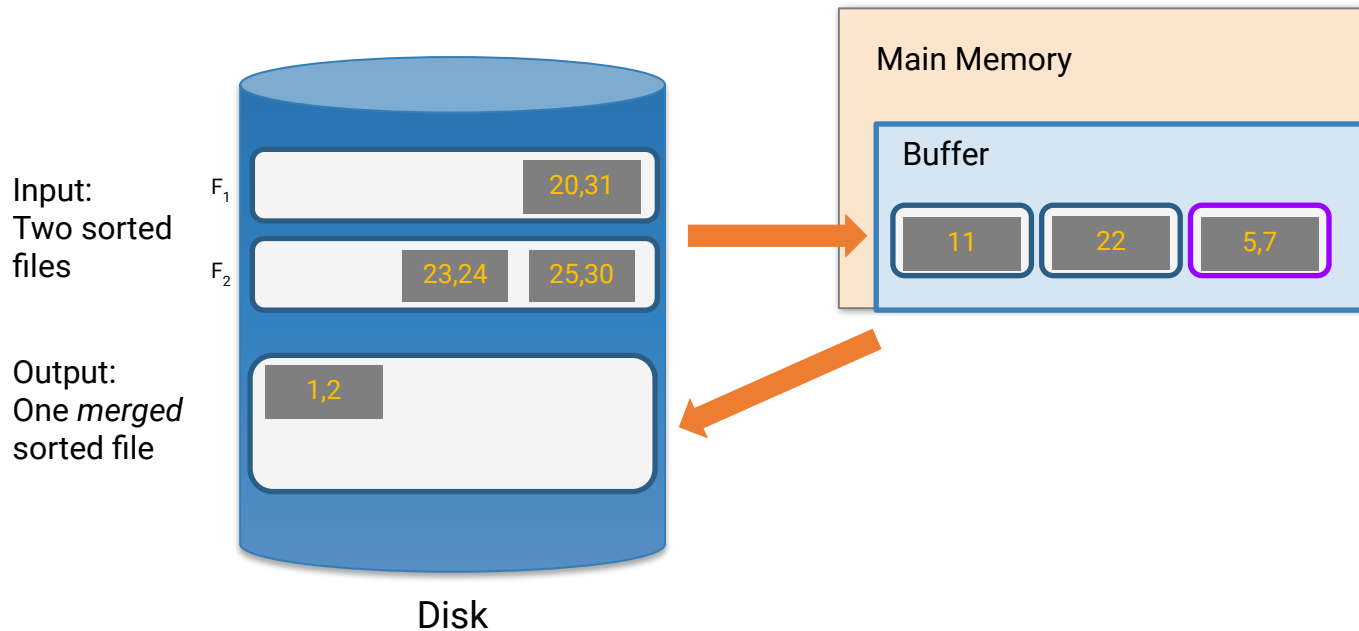
# External Merge Algorithm



# External Merge Algorithm

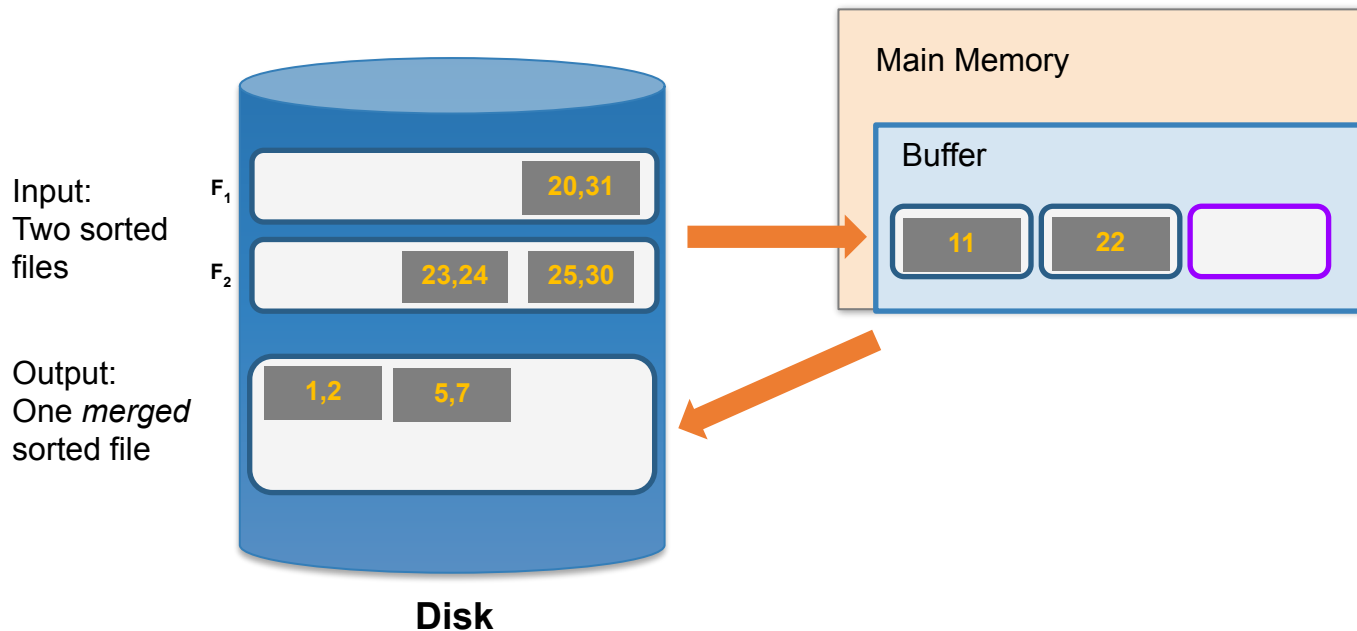


# External Merge Algorithm



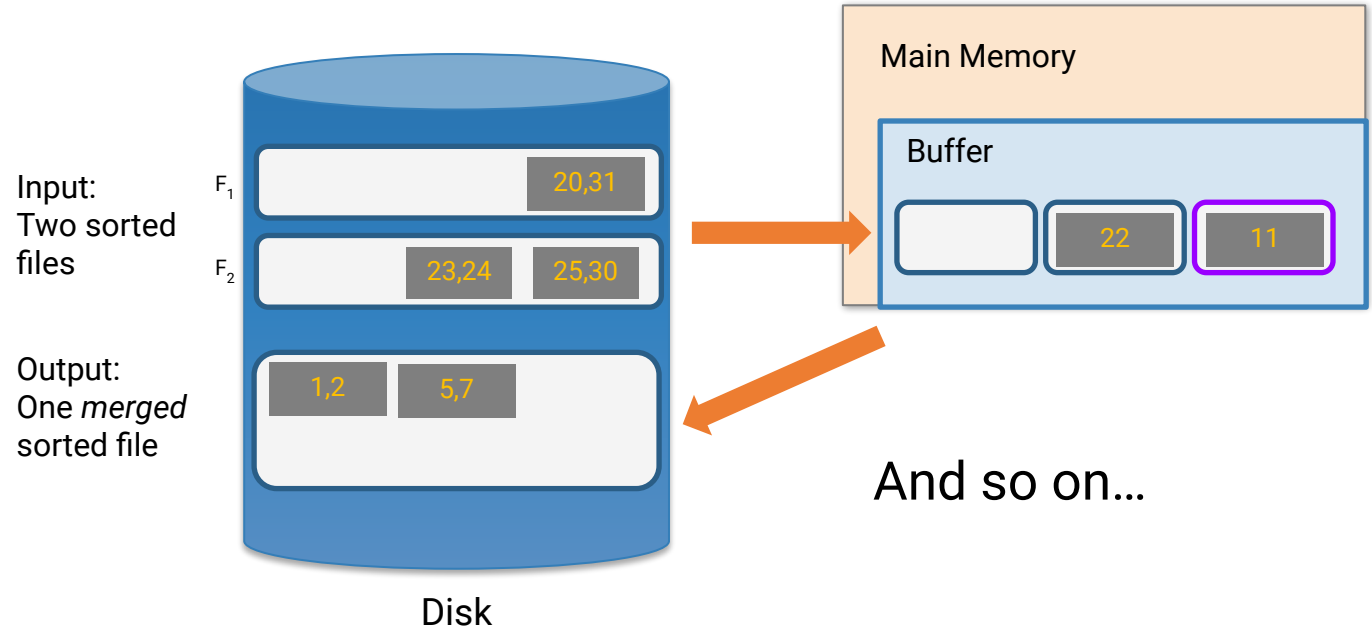


# External Merge Algorithm





# External Merge Algorithm



We can merge lists of arbitrary length with only 3 buffer pages.

If lists of size  $M$  and  $N$ , then  
Cost:  $2(M+N)$  IOs  
Each **page** is read once, written once

1. Recall:  $n \log n$  for sorting in RAM still true. Negligible vs IO costs from disks.
2. With  $B+1$  buffer pages, can merge  $B$  lists. How?

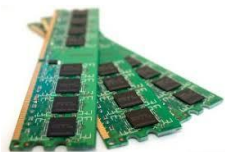
## Recap: External Merge Algorithm

- Suppose we want to merge two sorted files both much larger than main memory (i.e. the buffer)
- We can use the external merge algorithm to merge files of *arbitrary length* in  $2*(N+M)$  IO operations with only 3 buffer pages!

Our first example of an “IO aware”  
algorithm / cost model

# Big Scaling (with Indexes)

## Roadmap



### Primary data structures/algorithms

Hashing

HashTables  
( $\text{hash}_i(\text{key}) \rightarrow \text{value}$ )

Sorting

BucketSort, QuickSort  
MergeSort

MergeSortedFiles

?????

## In this Section

(Next weeks:  
Scale, Scale, Scale)

1. IO Model
2. Data layout
  - ▷ row vs column storage
3. Indexing
4. Organizing Data and Indices
  - ▷ Hashing, Sorting, Counting