Final Review

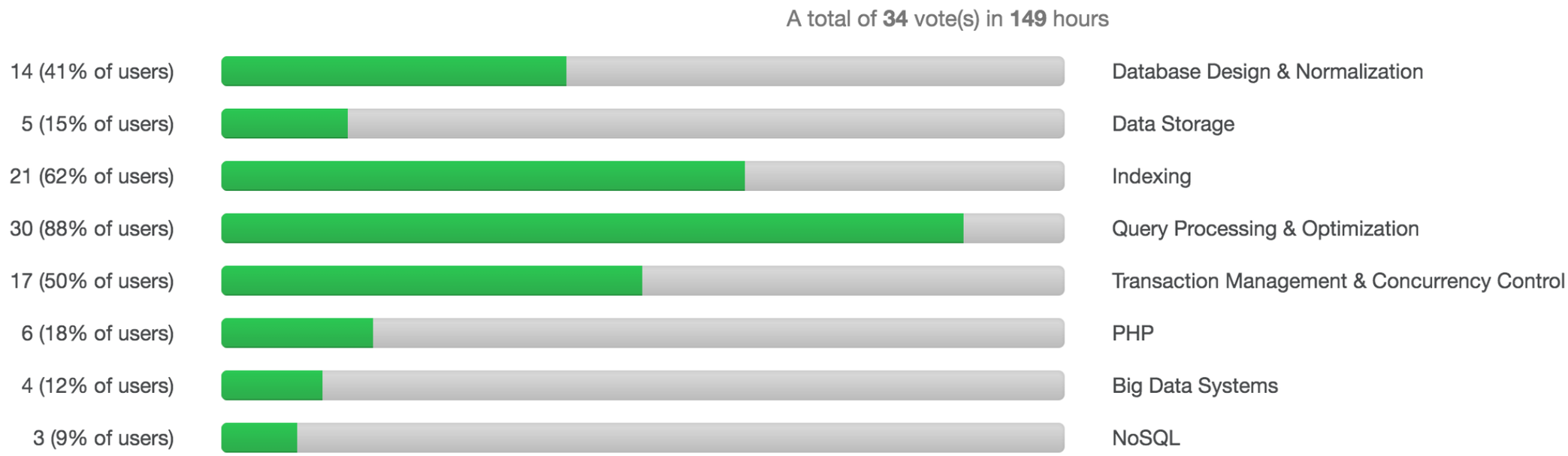
CS 377: Database Systems

Logistics

- April 29th, 8:00 -10:30 AM
- 8 single-sided handwritten cheat sheets
- Comprehensive covering everything up to current class
- Focus slightly more on the latter part of the course
- Should still study the first half material

DISCLAIMER: Concepts/topics not covered in this review does not mean it will not appear on the test!

Piazza Poll Results



Database Design & Normal Forms

Normalization & Functional Dependencies

- Normal form: set of properties that relations must satisfy
  - Relations exhibit less anomalies
  - Successively higher degrees of stringency
- Functional dependencies: X —> Y
  - Constraint between two sets of attributes
  - “Good” FDs are keys

Normalization Steps

- Find all keys of a relation: heuristic #1 or #2
- Find which FDs violate the normal form
    - Break the relation into two or more relations
        - Use closure set of FD
        - Follow lossless decomposition lemmas
        - Repeat FD violation step

Summary of 1NF, 2NF, 3NF

Normal Form	Test	Normalization (Remedy)
1NF	Relation should have no multi-valued attributes or nested relations	Form new relation for each multivalued attribute or nested relation
2NF	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key	Decompose and set up a new relation for each partial key with its dependent attributes using lossless decomposition
3NF	Relation should not have a nonkey attribute functionally determined by another nonkey attribute	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attributes
BCNF	Relation should not have an attribute functionally determined by another nonkey attribute	Decompose and set up a relation that includes the nonkey attribute(s) that functional determine(s) other attributes

Exercise: Normalization

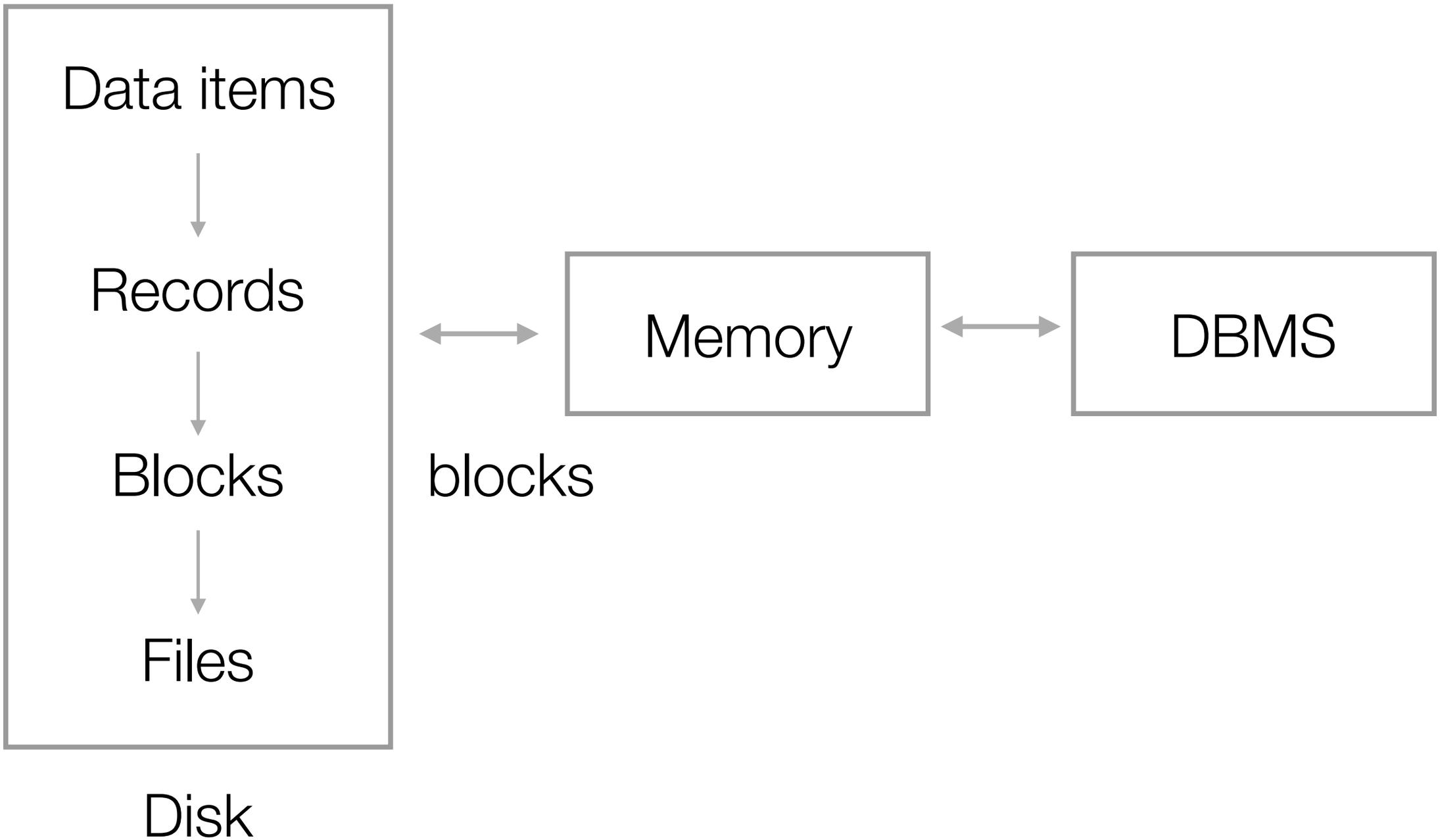
Suppose we have a relation R with four attributes A, B, C, D with the following functional dependencies:

- C —> D
- C —> A
  - B —> C

- What are the candidate key(s)?
- What is the best normal form that R satisfies?
- Decompose it into BCNF it is not BCNF

Data Storage

Data Store Overview



Files

- Disk space is organized into files
- Files consist of blocks (pages)
- Blocks consist of records
- Organization of records in files
  - Heap
  - Ordered (sequential)

Table 17.2 Average Access Times for a File of *b* Blocks under Basic File Organizations

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

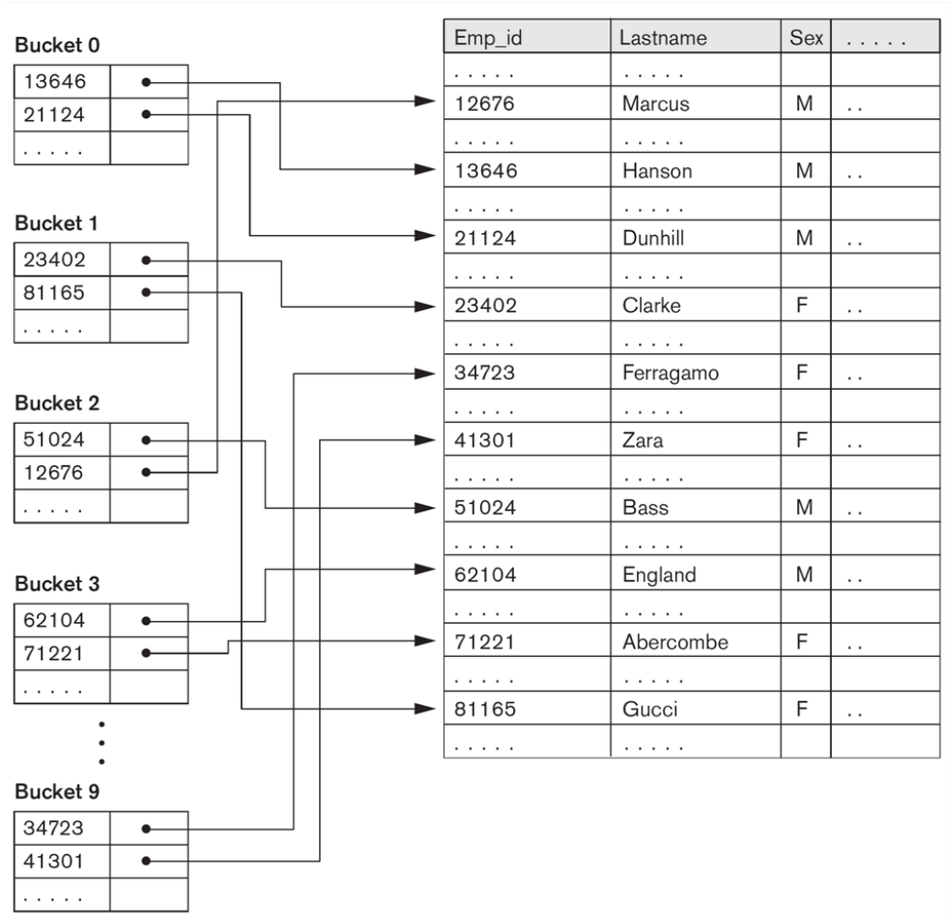
Indexing: Hashing & B+-Tree

Indexes

- Data structures that organize records via trees or hashing
  - Speed up search for a subset of records based on values in a certain field (search key)
    - Any subset of the fields of the relation can be the search field
    - Search key need not be the same as the key!
- Contains a collection of data entries (each entry with sufficient information to locate the records)

Hash Index

- Hash function, h, distributes all search-key values to a collection of buckets
- Each bucket contains a primary page plus overflow pages
- Buckets contain data entries
- Entire bucket has to be searched sequentially to locate a record (since different search-key values may be mapped to same bucket)



Extendible Hashing Structure

Main idea:

- Directory of pointers to the buckets
- Double the number of buckets by splitting just the bucket that overflowed
- Directory is much smaller than file, so doubling it is cheaper

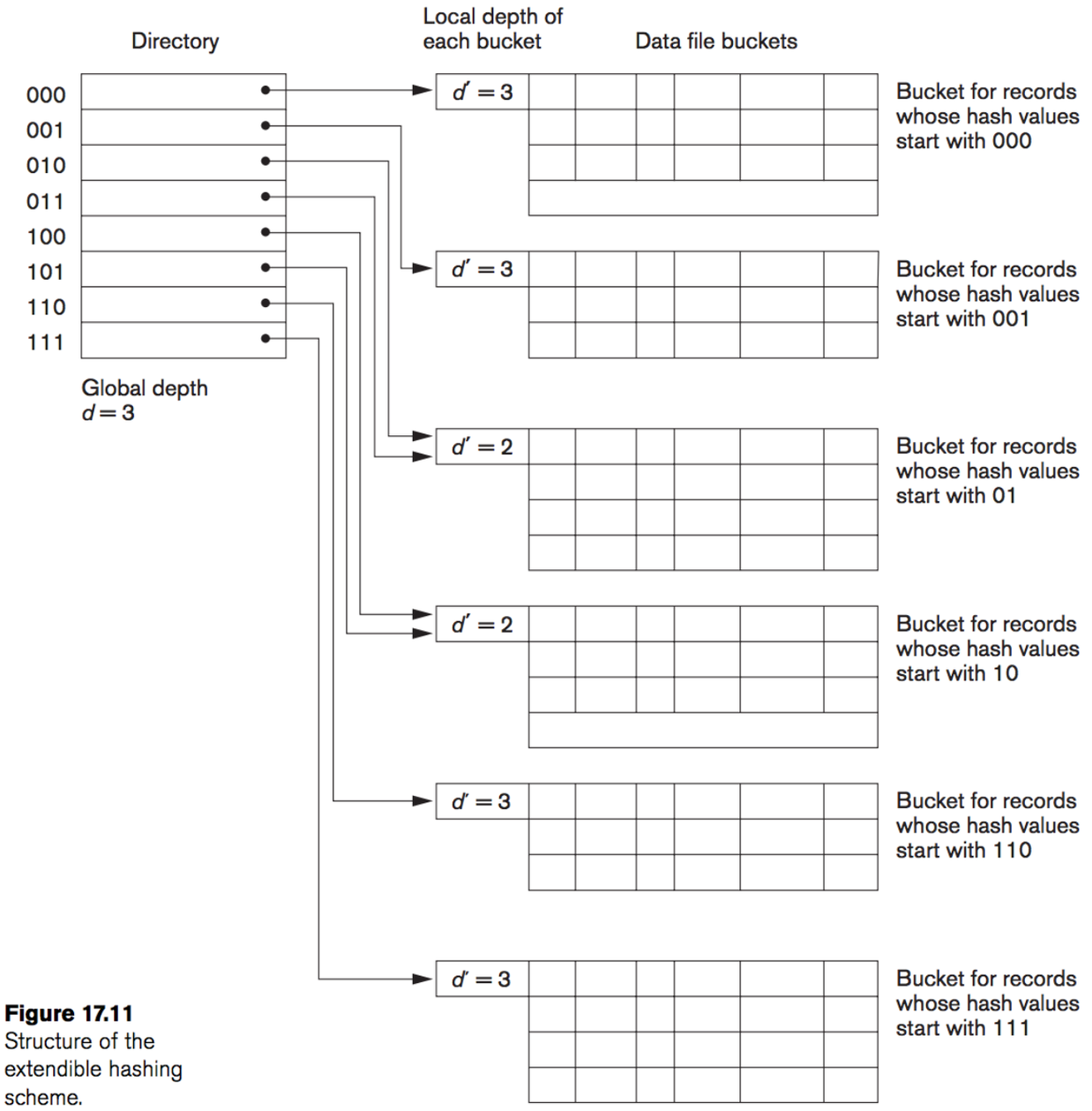


Figure 17.11
Structure of the extendible hashing scheme.

Exercise: Extendible Hashing

Insert the following keys into an empty extendible hashing structure where each bucket can hold up to 2 records and you want to use the highest-bits (leftmost d bits)

- 2 [0010], 10 [1010], 7 [0111], 3 [0011], 5 [0101], 15 [1111]

B+-Tree

- Dynamic, multi-level tree data structure
  - Adjusted to be height-balanced (all leaf nodes are at same depth)
  - Good performance guarantee — supports efficient equality and range search
- Widely used in DBMS

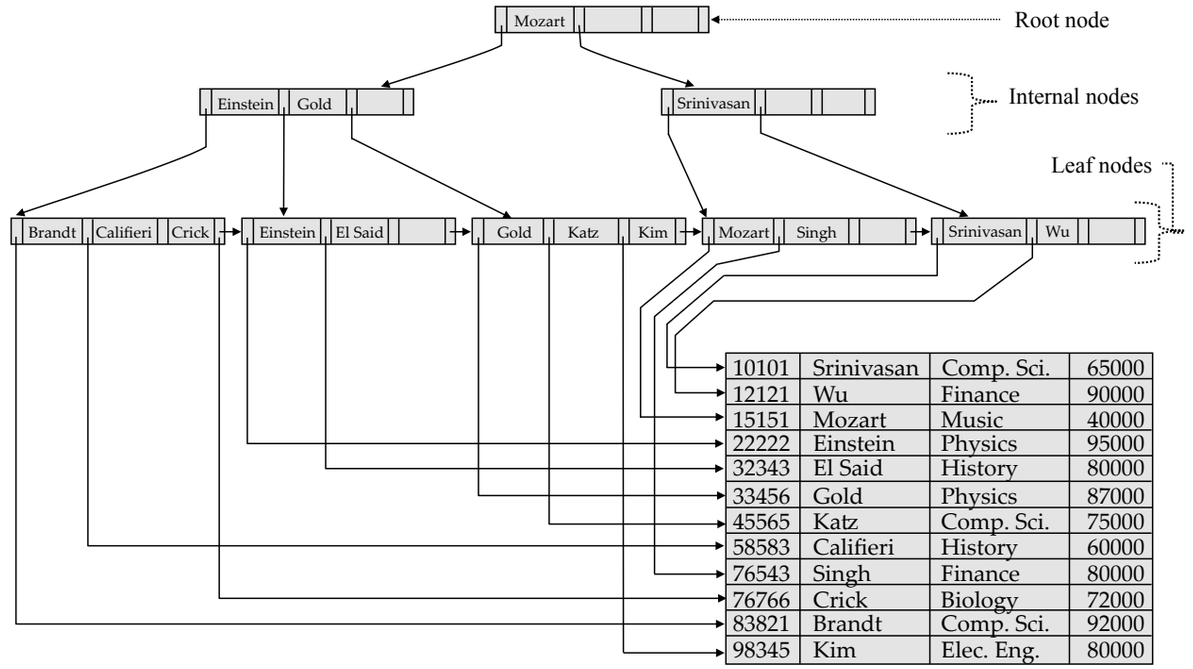
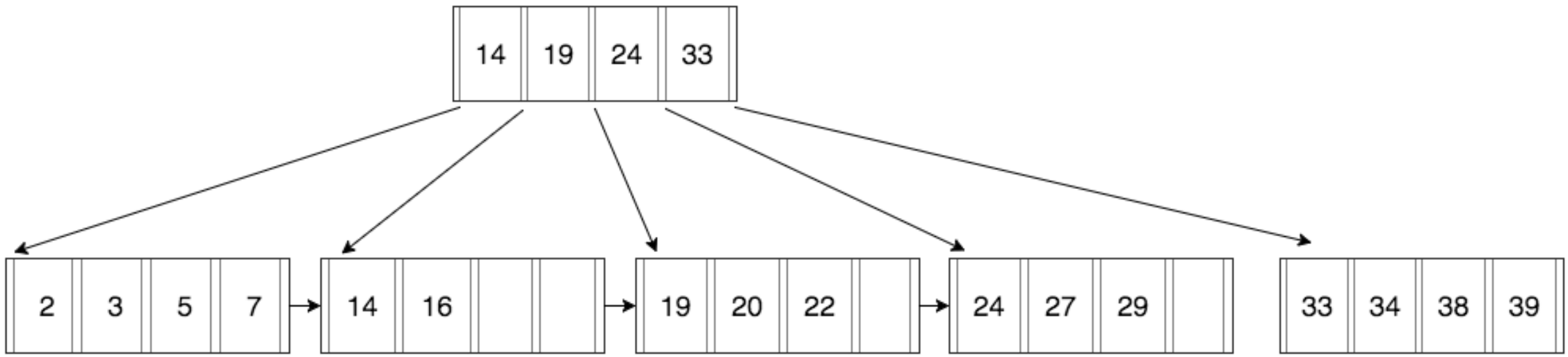


Figure from Database System Concepts book

Exercise: B$^+$-Tree



- Insert 17
- Insert 35
- Delete 7
- Delete 14

Index Structures

- Hash index
  - Good for equality search
  - In expectation: O(1) I/Os and CPU performance for search and insert
- B+ tree index
  - Good for range and equality search
  - I/O cost is height of tree for search, insert, and delete

Query Processing & Optimization

Basic Steps in Query Processing

- Parse and translate: convert to RA query
- Optimize RA query based on the different possible plans
- Evaluate the execution plan to obtain the query results

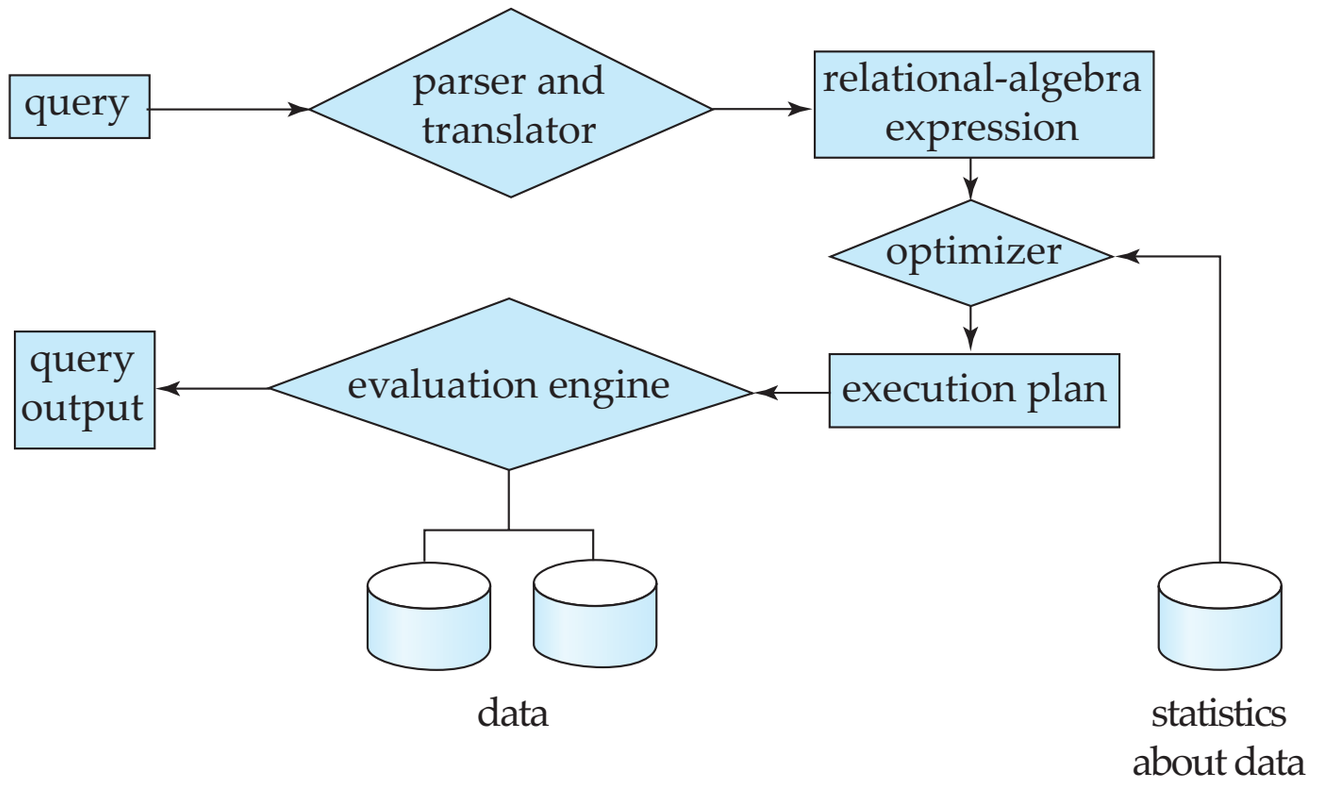


Figure 12.1 from Database System Concepts book

Query Optimization Heuristics (Logical)

Main heuristic: Favor operations that reduce the size of intermediate results first

- Break conjunctive SELECT and move select operations as far down tree as permitted
- Rearrange leaf nodes so leaf nodes with most restrictive select operations are executed first
- Combine cartesian product operation with a subsequent selection operation into join operation
- Break down and move lists of projection attributes down the tree as far as possible

Exercise: RA Query Transformation

Given three relations:
Course(cid, title, dname, credits)
Teaches(iid, cid, sid, semester, year)
Instructor(iid, name, dname, salary)

Query: Find the names of all instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught

- What is the initial RA query?
- Transform the query into an “optimal” RA query

Cost-based Query Optimization

Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate

- Disk I/O cost
- Storage cost
- Computation cost
- Memory usage cost
- Communication cost (distributed databases)

SELECT Algorithms

- Linear search: selection attribute is not ordered and no index on attribute
- Binary search: selection attribute is ordered but no index
- Index search: selection attribute has an index (primary or secondary) that can possibly be used on the query

SELECT Algorithms Cost

Search Type	Details	Cost
Linear		b_r
Binary		$\lceil \log_2 b_r \rceil + \lceil \mathrm{SC}(\mathrm{att},\ \mathrm{r})/f_r \rceil - 1$
Primary index	candidate key	$HT_i + 1$
Primary index	nonkey	$HT_i + \lceil \mathrm{SC}(\mathrm{att},\ \mathrm{r})/f_r \rceil$
Primary index	comparison	$HT_i + \lceil c/f_r \rceil$
Secondary index	candidate key	$HT_i + 1$
Secondary index	nonkey	$HT_i + \mathrm{SC}(\mathrm{att},\ \mathrm{r})$
Secondary index	comparison	$HT_i + (LB_i c)/n_r + c$

Exercise: SELECT

Employee relation with clustering index on salary:

- $n_{employee}$ = 10,000 (10,000 tuples in employee)
- $b_{employee}$ = 2,000 (2,000 blocks)
  - Secondary index (B^+-Tree) on SSN (key attribute)
    - HTi = 4 levels

What algorithm would be used for the following query and why?

$$\sigma_{\mathrm{SSN}=123456789}(\mathrm{Employee})$$

Exercise: SELECT

Same employee relation with clustering index on salary:

- Secondary index (B^+-Tree) on DNO (non-key)
  - HTi = 2
  - LBi = 4 (4 first level index blocks)
  - V(DNO, employee) = 125

What algorithm would be used for the following query and why?

$$\sigma_{\mathrm{DNO}>5}(\mathrm{Employee})$$

External Sort Merge Algorithm

- Sort r records, stored in b file blocks with a total memory space of M blocks (relation is larger than memory)

- Total cost:

$$2b_r(\lceil \log_{M-1}(b_r/M) \rceil + 1)$$

NOTE: that the previous slides (lecture 20) were off by a factor of 2 for the second part!

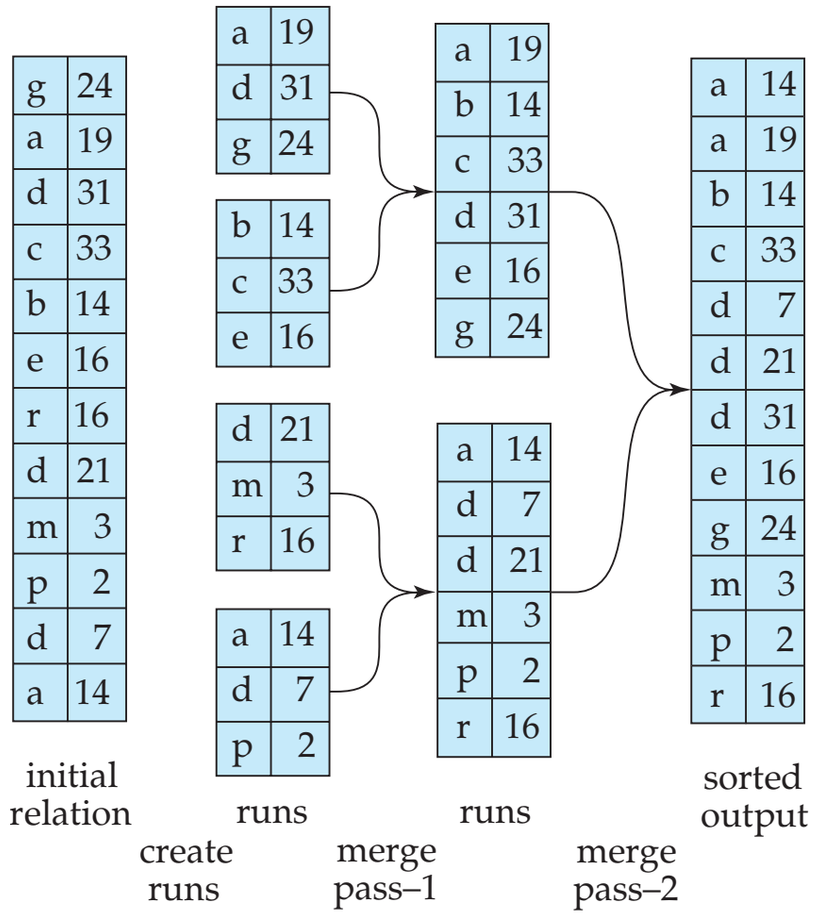


Figure 12.4 from Database System Concepts book

JOIN Algorithms

- Nested loop join: brute force algorithm and can be used with any join condition

- Nested-block join: somewhat brute force except join one block at a time together

- Indexed nested loop join: index is available on inner loop's join attribute to compute the join

- Sort-merge join: sort relations and join, only used for equijoin and natural join

- Hash-join: hash the relations and only join tuples in same bucket, only used for equijoin and natural join

JOIN Algorithms Cost

Type	Details	Cost
Nested loop		$n_R b_S + b_R$
Nested block		$b_R b_S + b_R$
Indexed nested loop		$b_R + n_R c$
Sort merge join		$\text{sort cost} + b_R + b_S$
Hash join	no recursive partitioning	$3b_R + 3b_S$
Hash join	recursive partitioning	$2(b_R + b_S)\lceil \log_{M-1}(b_S) - 1 \rceil + b_R + b_S$

if both fit in memory: $b_R + b_S$

Exercise: JOIN Operation

Employee and Department

- $n_{employee}$ = 10,000 (10,000 tuples in employee)
- $b_{employee}$ = 2,000 (2,000 blocks)
- $n_{department}$ = 125 (125 tuples in department)
- $b_{department}$ = 13 (13 blocks)
- Primary index dnumber in department with HTi = 1
- Secondary index mgr_ssn in department with HTi = 2

Exercise: JOIN Operation (2)

Employee and Department

- What join algorithm makes sense for joining Employee and Department on department number?

- What join algorithm makes sense for joining Employee and Department on manager ssn?

Transaction Management & Concurrency Control

ACID: Transaction Properties

- Atomicity: all actions in transaction happen or none happen

- Consistency: a database in a consistent state will remain in a consistent state after the transaction

- Isolation: the execution of one transaction is isolated from other transactions

- Durability: if a transaction commits, its effects must persist

Recovery via System Logs

Idea: Keep a system log and perform recovering when necessary

- Contains log records that contains information about an operation performed by transaction

- Each transaction is assigned a unique transaction ID to different themselves

Write ahead logging (WAL): all modifications are written to a log before they are applied to database

Logging

- Undo logging: undo operations for uncommitted transactions to go back to original state of database
  - Write data — add [write, T, X, **old_value**], after successful write to log, update X with new value
- Redo logging: save disk I/Os by deferring data changes or do the changes for committed transaction
  - Write data — add [write, T, X, **new_value**], after successful write to log, update X with new value

Conflict

- Pairs of consecutive actions such that if their order is interchanged, the behavior of at least one of the transactions can change
  - Involve the same database element
  - At least one write
- Three types of conflict: read-write conflicts (RW), write-read conflicts (WR), write-write conflicts (WW)

Serializability Definitions

- S1, S2 are **conflict equivalent** schedules if S1 can be transformed into S2 by a series of swaps on non-conflicting actions

- A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule

- Maintains consistency & isolation!

Precedence (Serialization) Graph

- Graph with directed edges
  - Nodes are transactions in S
  - Edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj
- Schedule is serializable if and only if precedence graph has no cycles!

Exercise: Serializability

Consider the schedule given in the table below of three transactions T1, T2, and T3

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
T_1			R(A)		W(A)		R(C)		W(C)		
T_2				R(B)		W(B)					
T_3	R(A)	W(A)						R(B)	W(B)	R(C)	W(C)

- Draw the precedence graph
- Is this schedule serializable?

Strict Two-phase Locking (Strict 2PL)

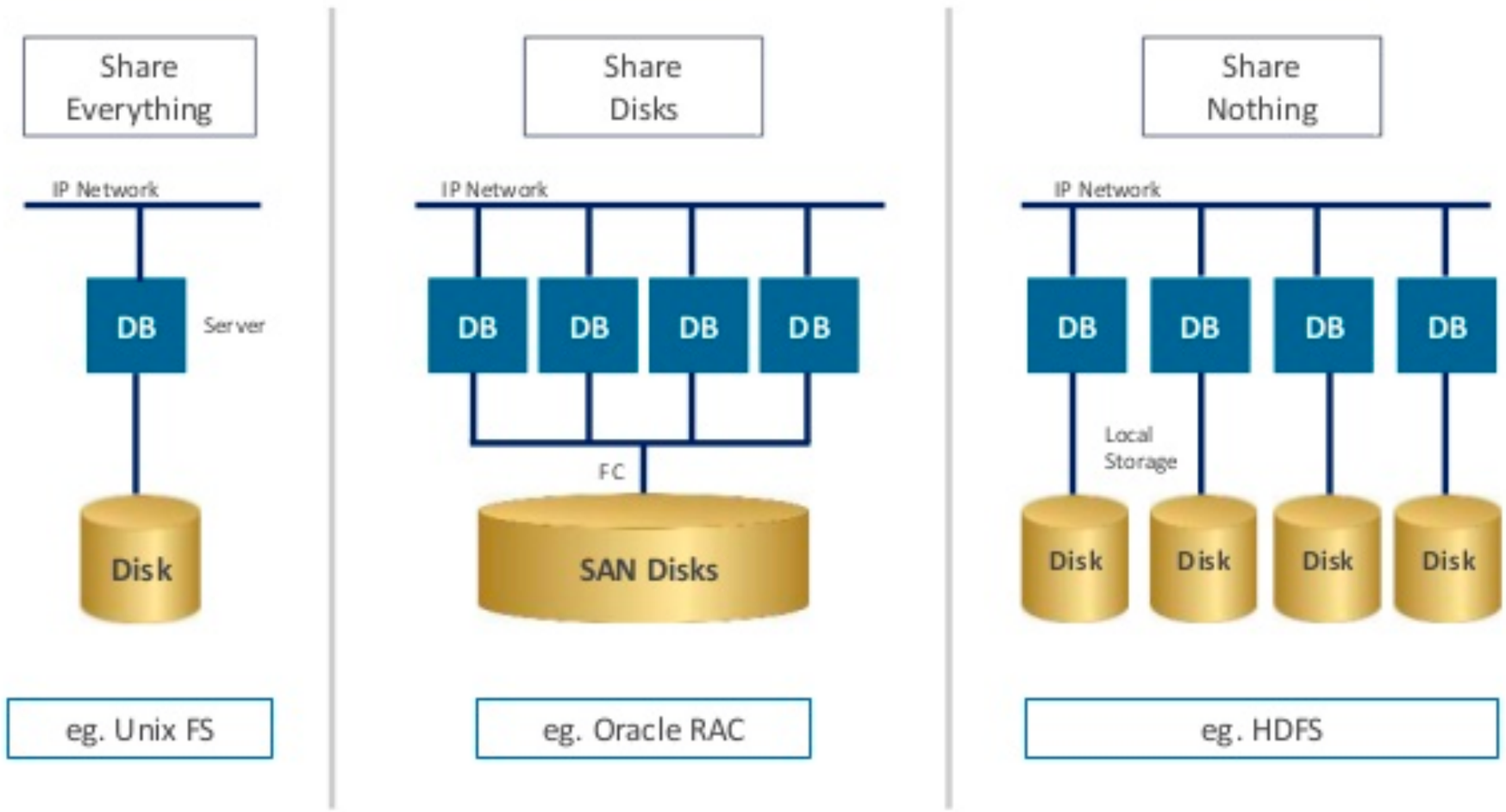
- Each time you want to read/write an object, obtain a lock (shared/exclusive) to secure permission to read/write object
- Only release locks at commit / abort time
    - A transaction that writes will block all other readers until the transaction commits or aborts
- Ensure transactions remain isolated and follow serializable schedules
- Downside: not deadlock free

Big Data Systems

Parallel/Distributed DBMS

- Data partitioned across multiple disks
  - Allows parallel I/O for better speed-up
- Individual relational operations (e.g., sort, join, aggregation) can be executed in parallel
  - Each processor can work independently on its own partition
- Queries can be run in parallel with each other
  - Concurrency control takes care of conflicts

Parallel Architectures



http://image.slidesharecdn.com/hadooparchitecture-091019175427-phpapp01/95/big-data-analytics-with-hadoop-18-638.jpg?cb=1411533606

Parallel/Distributed DBMS Issues

- How to distribute the data
- How to optimize the cost of queries
  - Data transmission + local processing
- How to perform concurrency control
- How to make system resilient to failures and achieve atomicity & durability

MapReduce

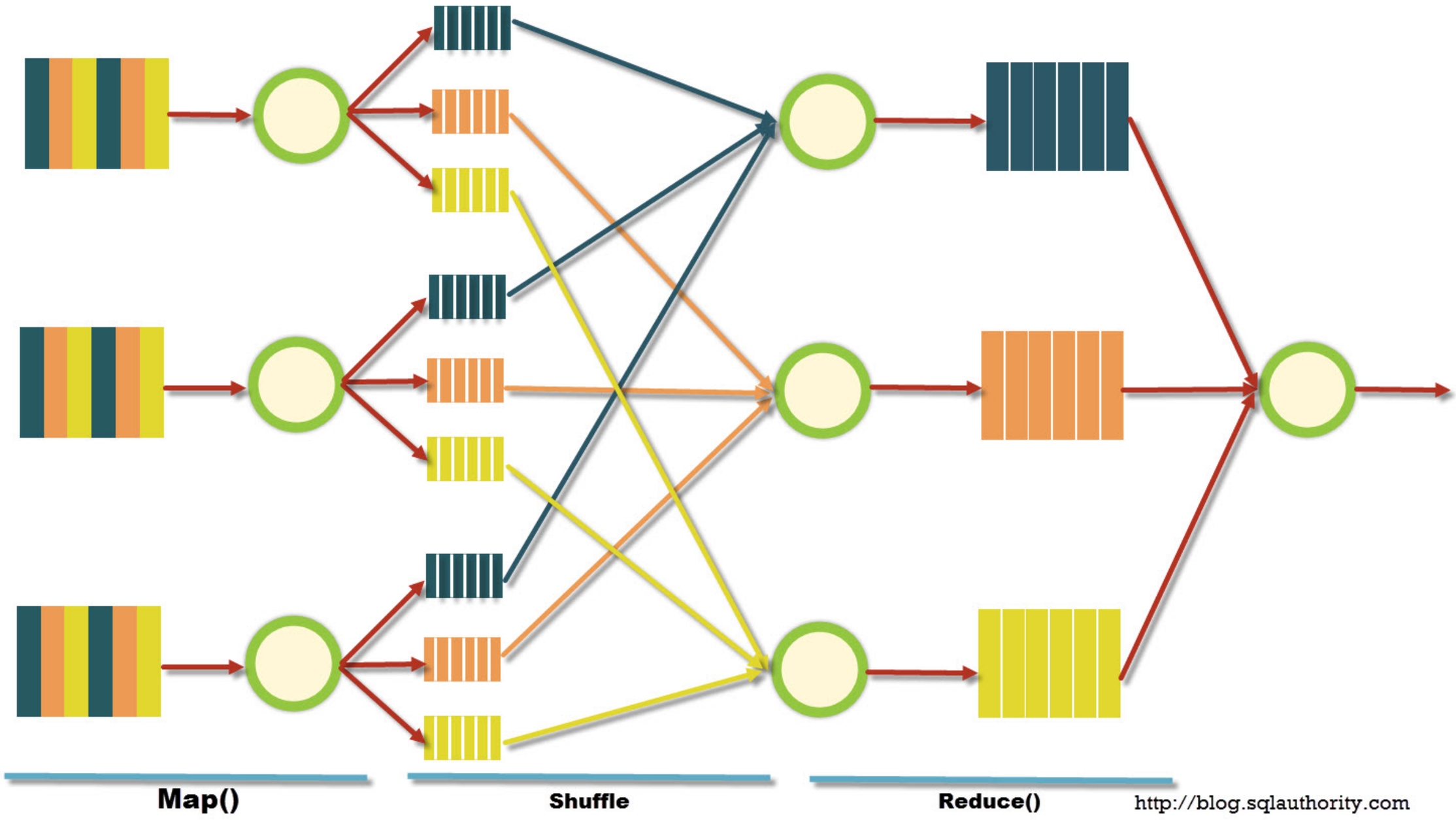
- Initially developed by Jeffrey Dean & Sanjay Ghemawat at Google [2004]

- Open source implementation: Apache Hadoop

- High-level programming model and implementation for large-scale parallel data processing

- Designed to simplify the task of writing parallel programs

MapReduce: Program



https://erlerobotics.gitbooks.io/erle-robotics-python-gitbook-free/content/caches,_message_queues,_and_map-reduce/mapreduce.jpg

NoSQL

CAP Theorem

"Of three properties of shared-data systems — data Consistency, system Availability, and tolerance to network Partitions — only two can be achieved at any given moment in time" — Brewer, 1999

- Consistency: all nodes see the same data at the same time
- Availability: guarantee that every request receives a response about whether it was successful or failed
- Partition tolerance: system continues to operate despite arbitrary message loss or failure of part of the system

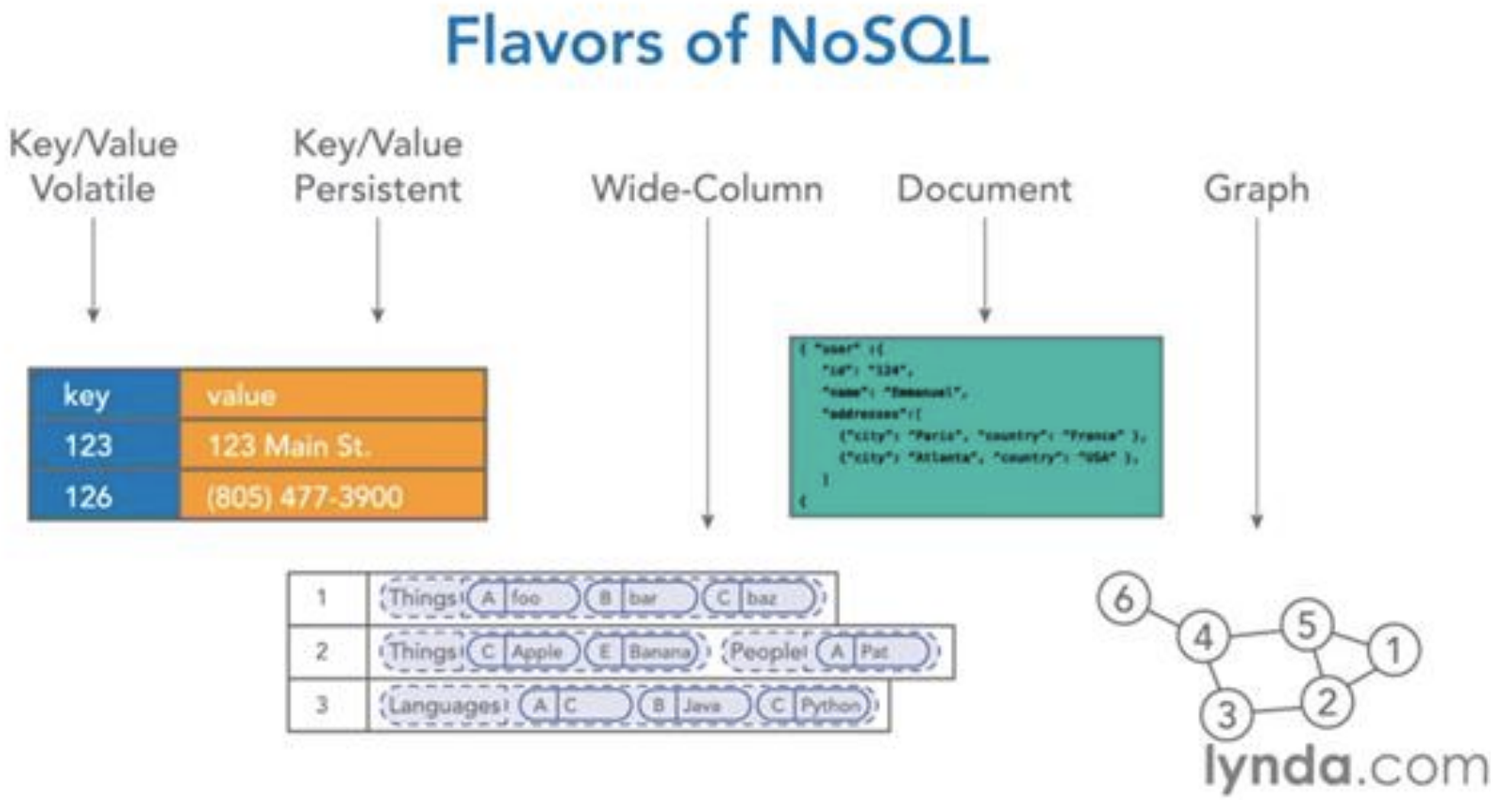
ACID vs BASE

RDMS		NoSQL
ACID		**CRUD**
Atomic	⇔	**B**asically **A**vailable
Consistent		**S**oft state
Isolated		**E**ventually consistent
Durable		

https://www.linkedin.com/pulse/rdbms-follows-acid-property-nosql-databases-base-does

NoSQL Categories

- Key-value stores
- Column-based families or wide column systems
- Document stores
- Graph databases



https://cdn.lynda.com/video/387725-179-635691101939114107_338x600_thumb.jpg

NoSQL Use Cases & Challenges

- When should I think of using NoSQL database?
- What are the advantages of using NoSQL?
- What are the disadvantages of using NoSQL?