Final Review

CS 377: Database Systems

Final Logistics

- May 3rd, 3:00 - 5:30 PM
- 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!

My Office Hours

- 4/24 (Mon): 1:30 PM - 3:30 PM
- 4/26 (Wed): 9:30 AM - 12 PM; 3:30 PM - 5:00 PM
- 4/27 (Thu): 9:30 AM - 12 PM
- 5/1 (Mon): 1:30 PM - 3:30 PM; 4:00 PM - 5:00 PM
- 5/2 (Tue): 9:30 AM - 11:00 AM

Piazza Poll Results

Final Review Topics (Updated) closes in 1 day(s)

A total of **47** vote(s) in **137** hours

17 (36% of users)	Database Normalization	Show Voters
20 (43% of users)	Data Storage	Show Voters
37 (79% of users)	Indexing	Show Voters
32 (68% of users)	Query Processing & Optimization	Show Voters
26 (55% of users)	Transaction Management	Show Voters
25 (53% of users)	Database Application Programming	Show Voters

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

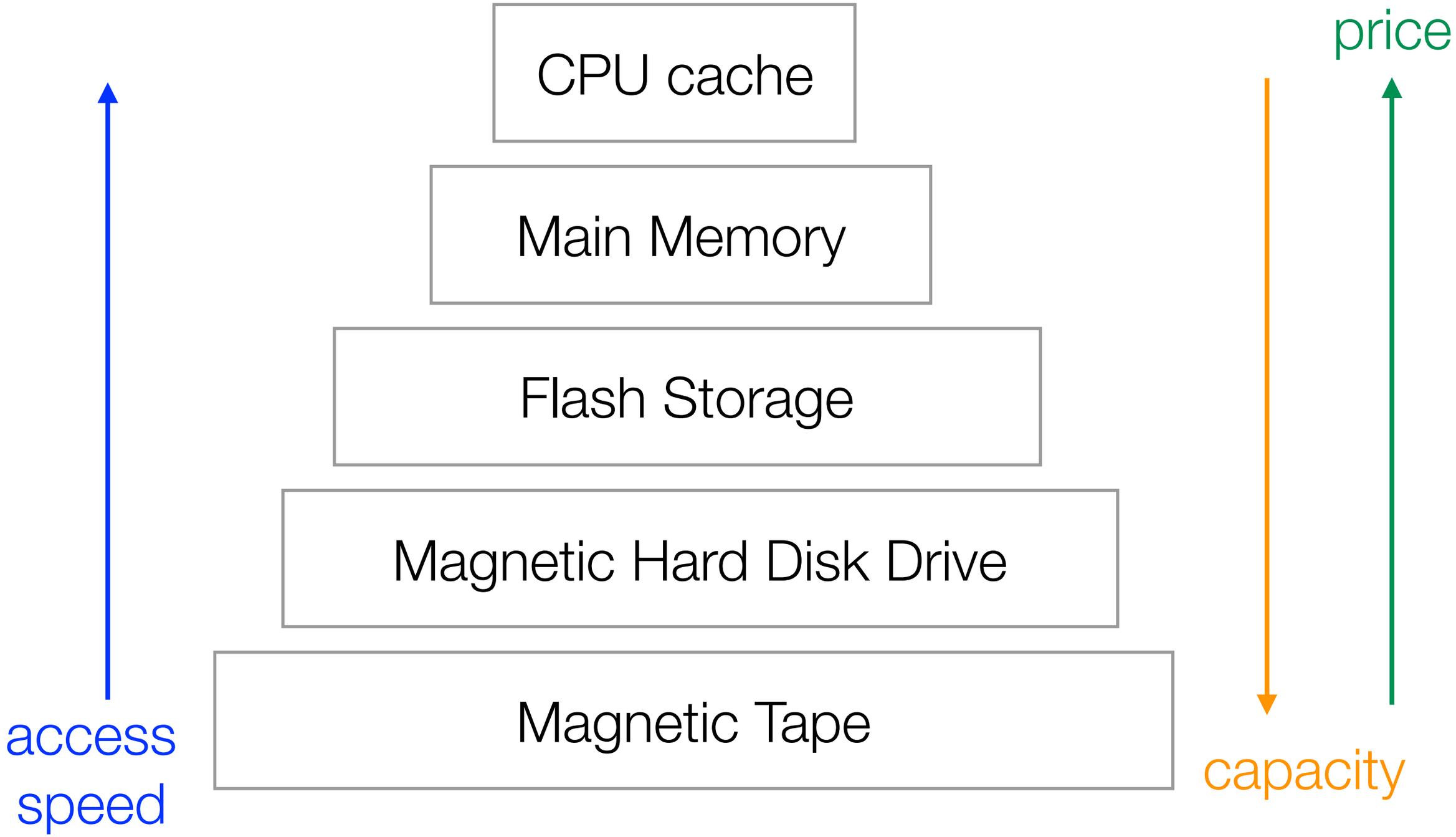
- Find all keys of a relation: heuristic #1 or #2
- Find which FDs violate the normal form
- Break the relation into two or more relations
  - Use closure set of FD
  - Follow lossless decomposition lemmas
  - Repeat FD violation step

Database Normalization: Summary

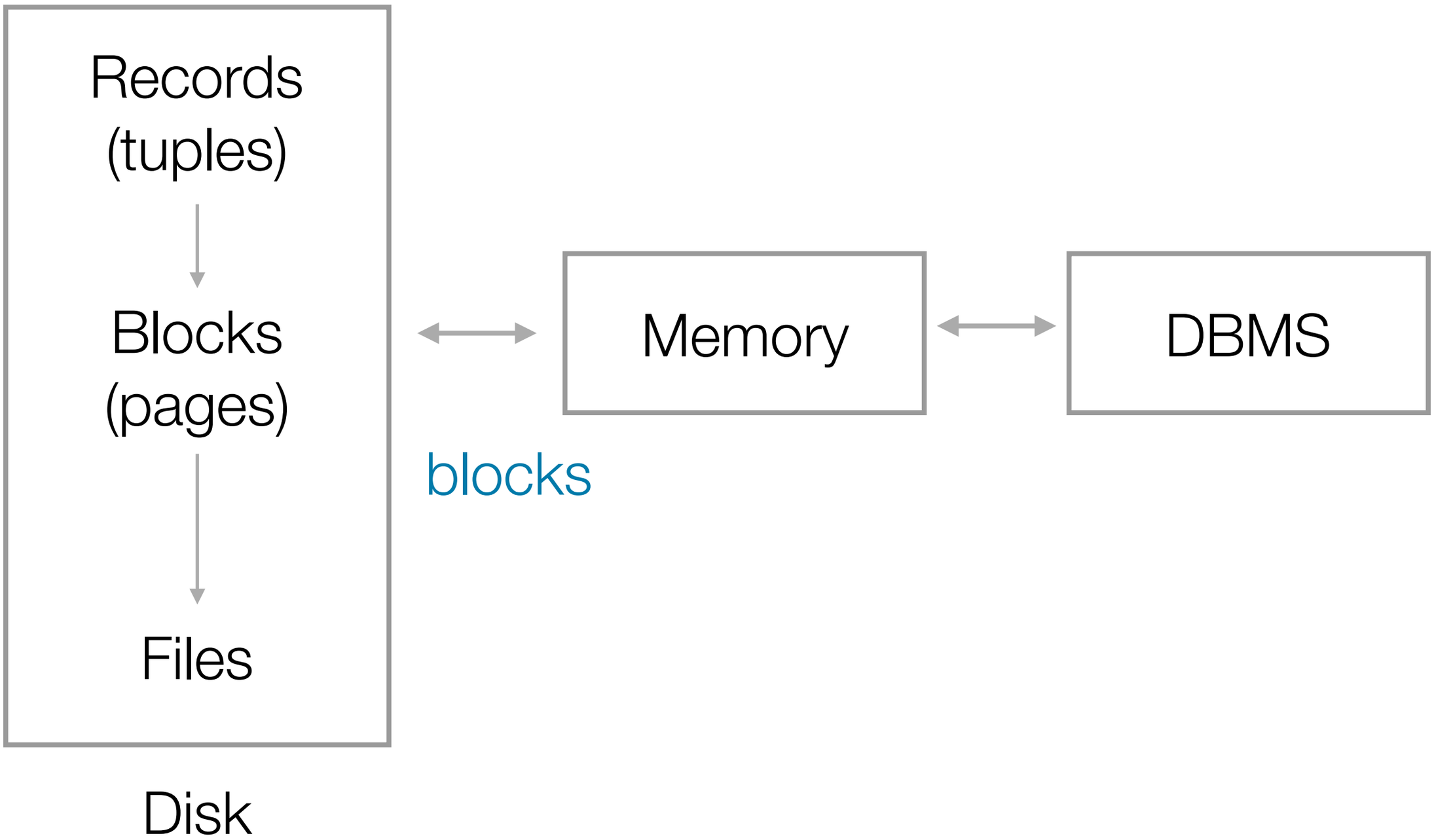
Normal Form	Test	Normalization (Remedy)
1NF	Relation should have no multi-valued attributes or nested relations	Form new relation for each multivalued attribute or nested relation
2NF	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on 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 non-key attribute(s)	Decompose and set up a relation using the closure of the non-key attribute(s)

Data Storage

Memory Hierarchy



Data Store Overview



File: Average Access Times

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	$\log_2 b$

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:

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

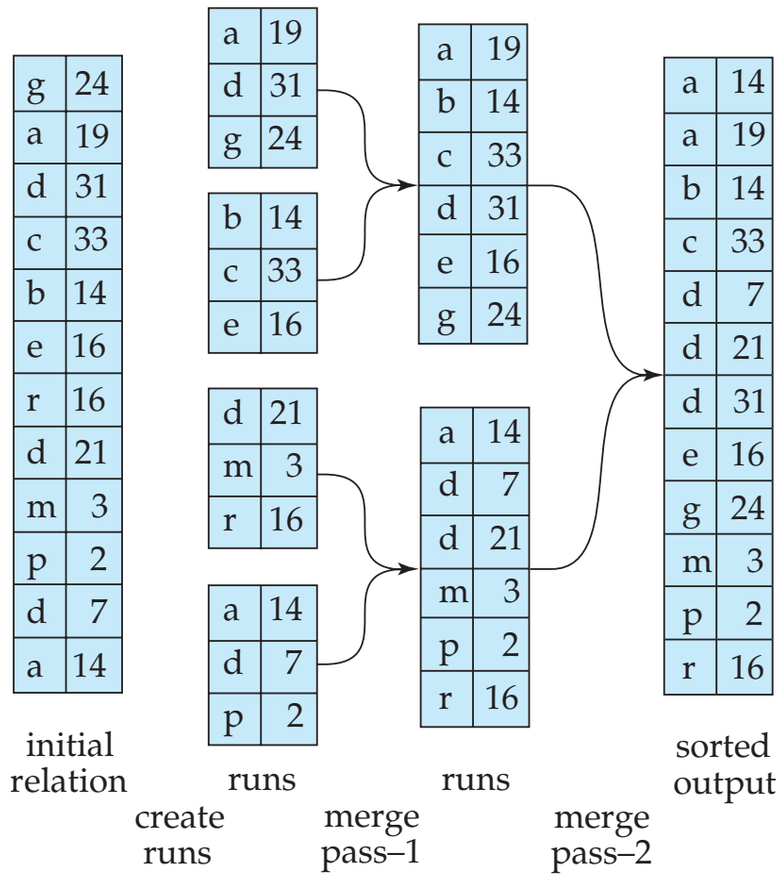


Figure 12.4 from Database System Concepts book

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

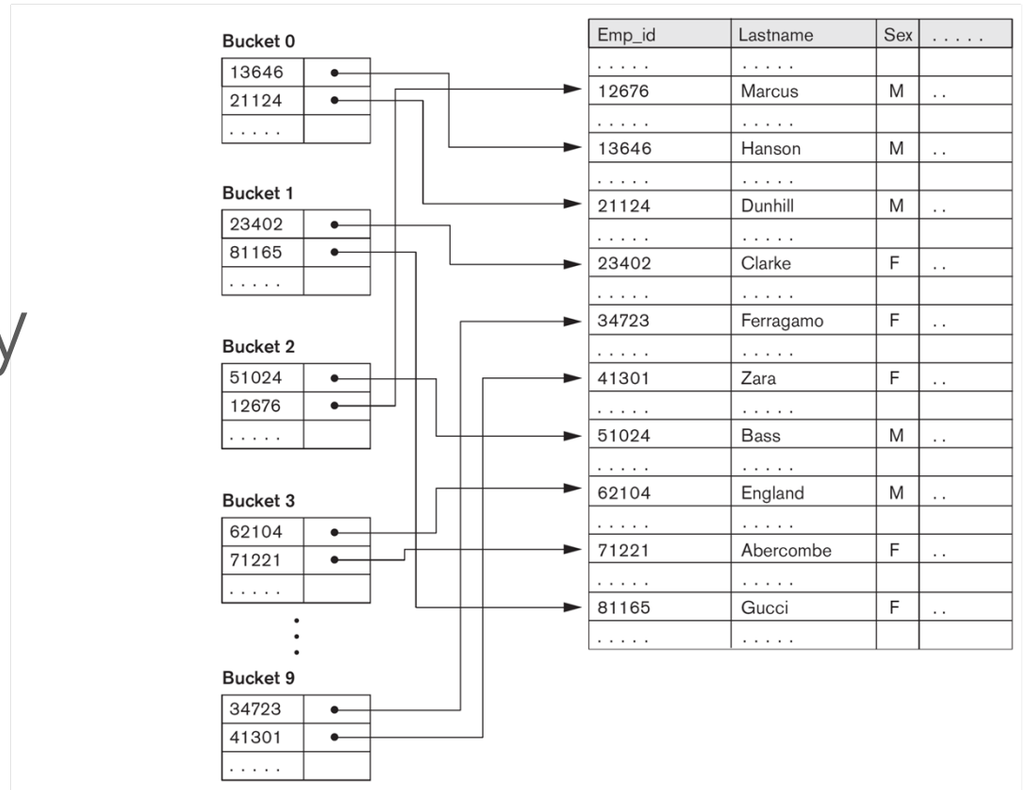


Figure 18.15
Hash-based indexing.

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

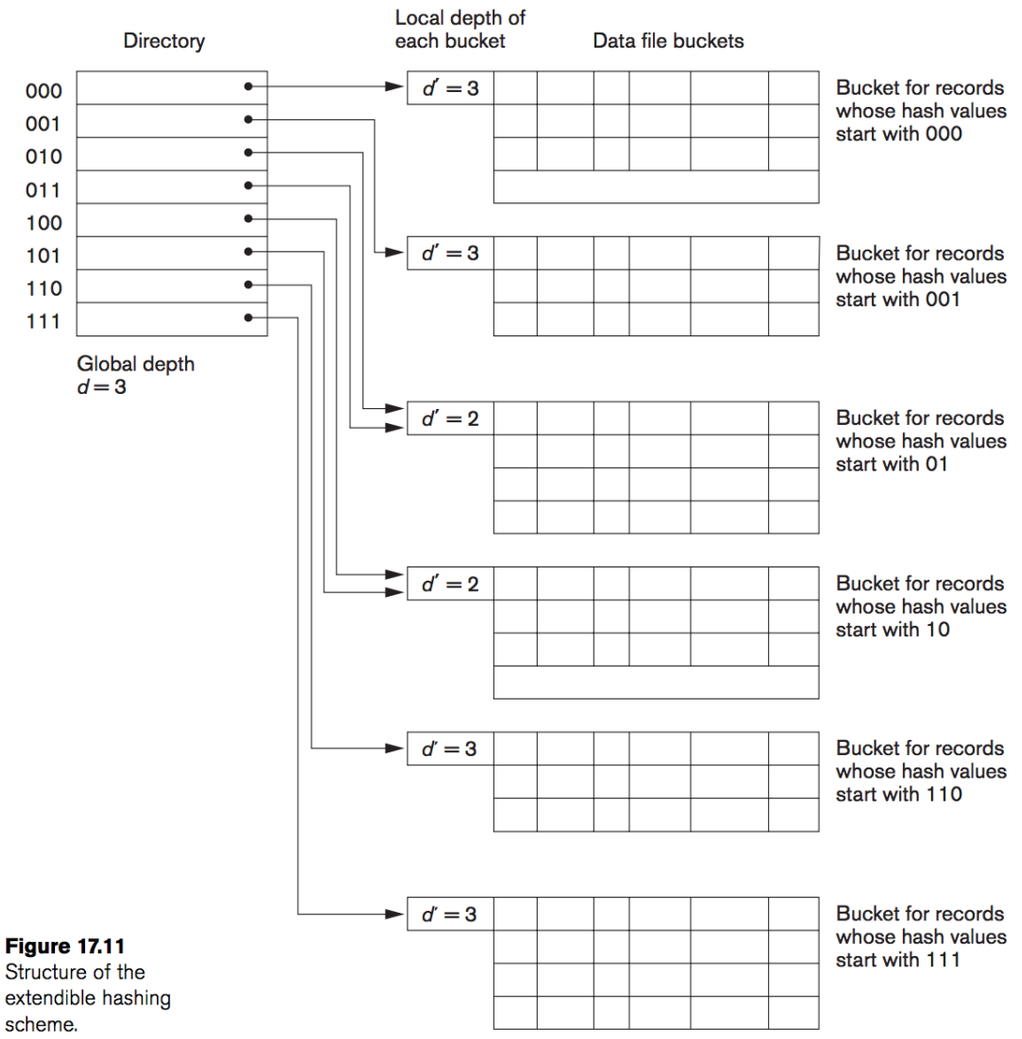


Figure 17.11
Structure of the extendible hashing scheme.

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)
  - Supports efficient equality and range search
- Widely used in DBMS

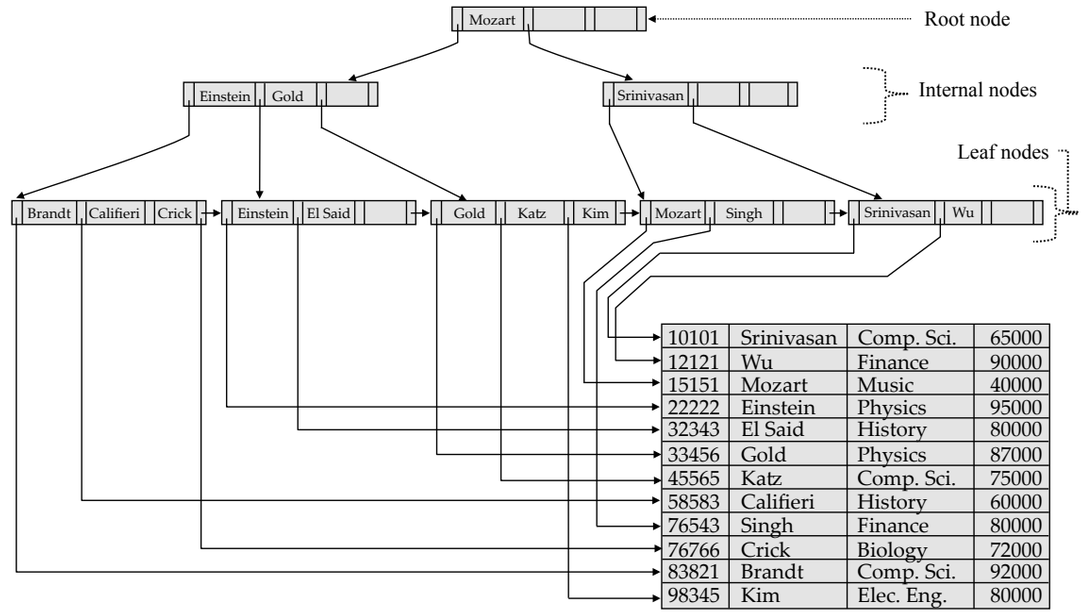
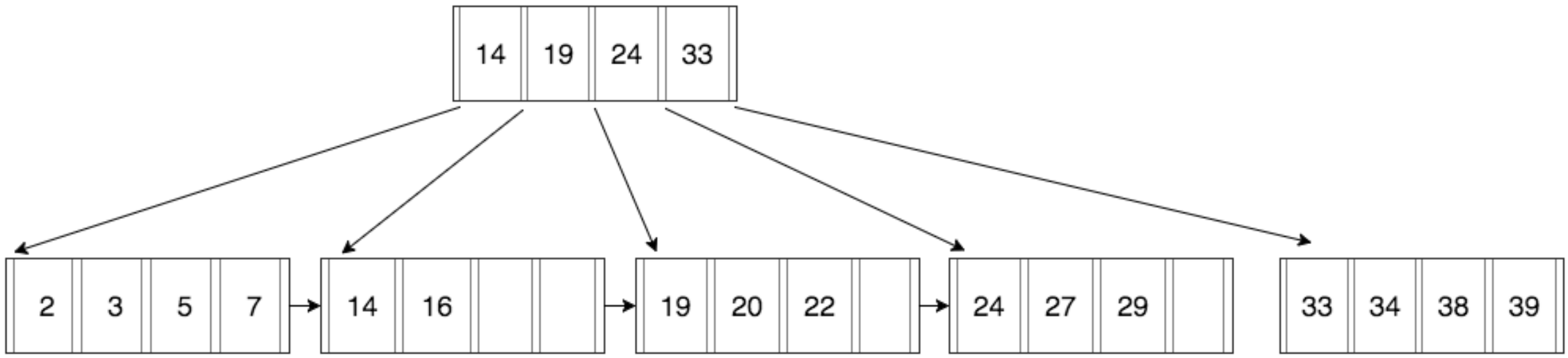


Figure from Database System Concepts book

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

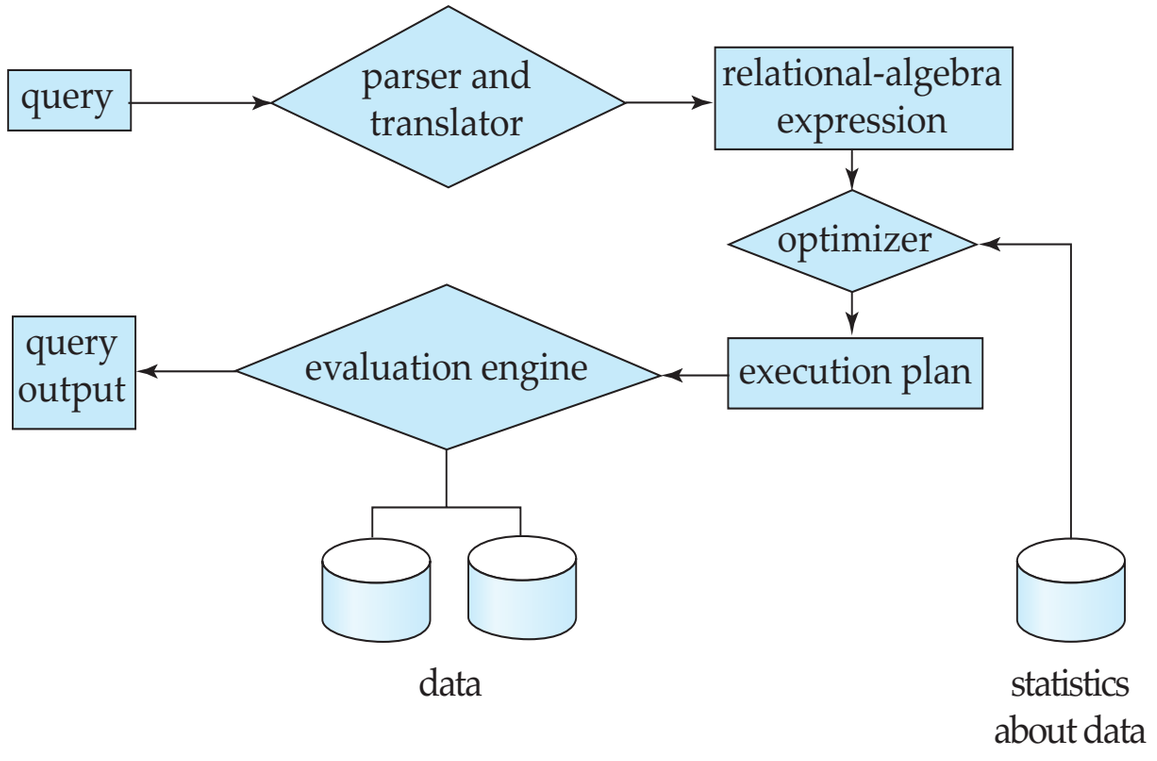


Figure 12.1 from Database System Concepts book

Query Optimization Heuristics

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:
Emp(eid, did, sal, hobby)
Dept(did, dname, floor, phone)
Finance(did, budget, sales, expenses)

Query: Find the name of the department(s) and the associated budget for employees that make at least 59K, work on the first floor, and have a hobby of yodeling

- What is the initial RA query tree?

Exercise: RA Query Transformation

What if the database had: 50K employees, 5K departments, salaries range from 10K-60K, 2 floors, 200 different hobbies

- What is the “optimal” relational algebra 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

Suppose we have the following table:
Emp(eid, sal, age, did)

Number of tuples: 20K (emp)
Number of blocks: 100 (emp)
Height of B+-tree: 4

Consider the query “Find all employees with age > 30”. Let the number of qualifying tuples be N, for what values of N is sequential scan cheaper than using a B+-tree index?

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

Suppose we have the following tables:
Emp(eid, sal, age, did)
Dept(did, pid, budget, status)

Number of tuples: 20K (emp); 5K (dept)
Number of blocks: 100 (emp); 125 (dept)

Consider the query joining emp and department on did, what is the optimal query assuming M = 20 and no indices? Does this change if there is a hash index on did for emp? What if we already have sorted emp by did?

Transaction Management

Transaction: Management

- Recovery (Atomicity & Durability)
  - Ensures database is fault tolerant, and not corrupted by software, system or media
  - 24x7 access to critical data
- Concurrency control (Isolation)
  - Provide correct and highly available data access in the presence of access by many users
- Rely on application program for consistency

Transaction: ACID

Recovery: System Log

Idea: Keep a system log and perform recovering when necessary

- Separate and non-volatile (stable) storage that is periodically backed up

- 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

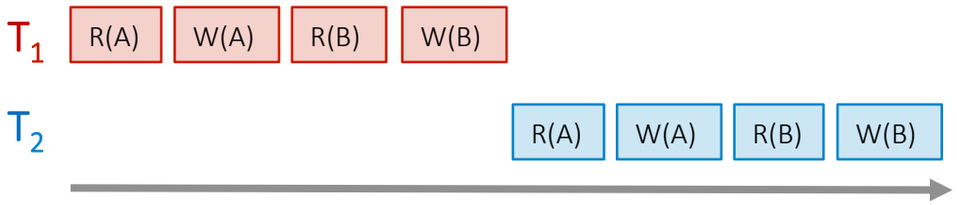
- Undo logging: undo operations for uncommitted transactions to go back to original state of database
  - Write data — add [write, T, X, **old_value**], after successful write to log, update X with new value
- Redo logging: save disk I/Os by deferring data changes or do the changes for committed transaction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

Schedule

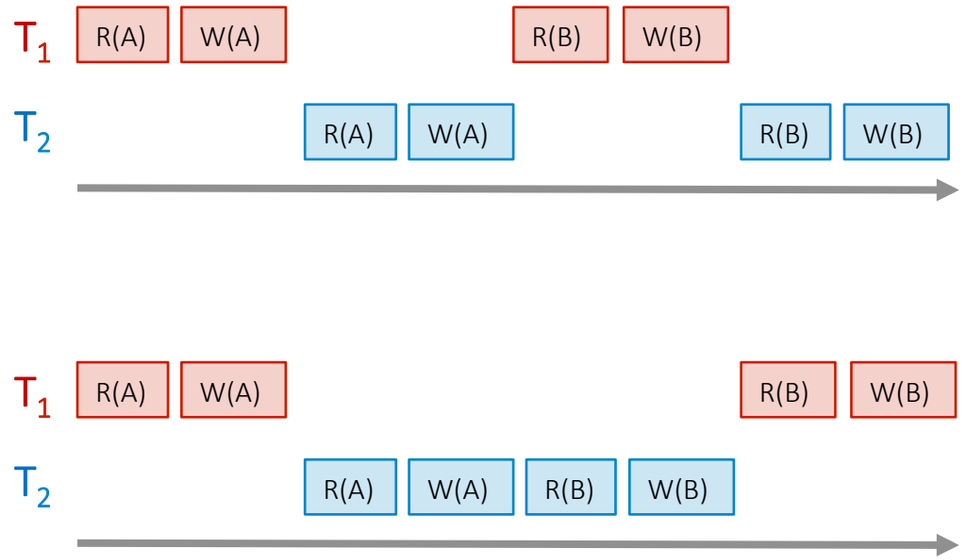
- A schedule S of n transactions T_1, T_2, …, T_n is an ordering of the operations of the transactions
  - For each transaction T_i, the operations in T_i in S must appear in the same order in which they occur in T_i
  - Operations from other transactions T_j can be interleaved with operations of T_i in S
- Want serializable schedules — equivalent to serial schedule

Schedules: “Good” vs “Bad”

Serial Schedule:



Interleaved Schedules:



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

- S_1, S_2 are **conflict equivalent** schedules if S_1 can be transformed into S_2 by a series of swaps on non-conflicting actions
  - Every pair of conflicting actions of two TXNs are ordered the same way
- 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 T_i to T_j if one of the operations in T_i appears before a conflicting operation in T_j
- Schedule is serializable if and only if precedence graph has no cycles!

Exercise: Serializability

- Draw the precedence graph
- Is this schedule serializable?

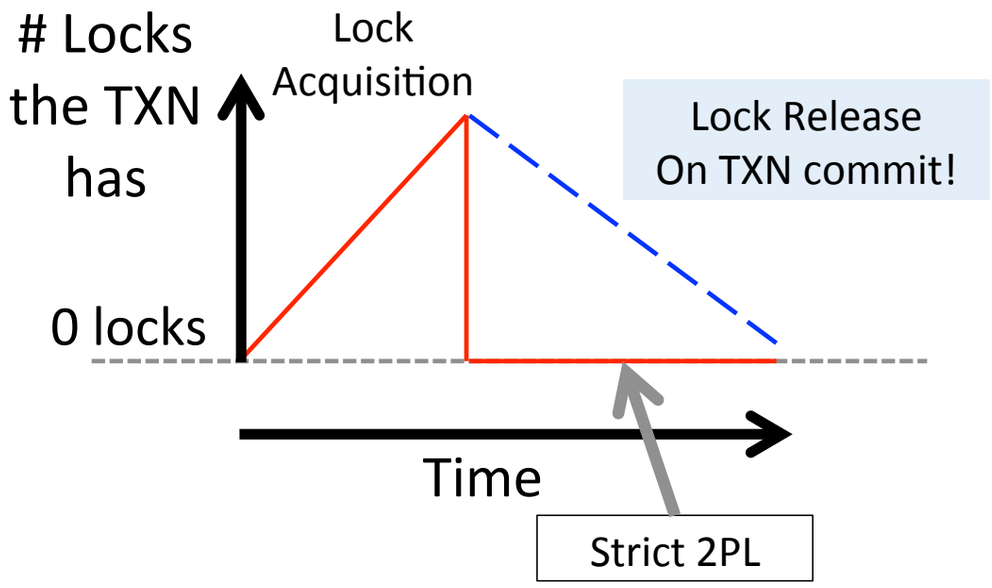
Schedule A					
T1	T2	T3	T4	T5	T6
-	-	-	read(G)	-	-
write(H)	-	-	-	-	-
read(G)	-	-	-	-	-
-	-	-	write(H)	-	-
-	-	-	read(C)	-	-
-	-	-	-	-	read(G)
-	-	-	-	read(H)	-
-	-	-	-	write(B)	-
-	-	-	-	-	read(A)
-	write(C)	-	-	-	-
-	-	write(D)	-	-	-
-	-	read(B)	-	-	-
write(E)	-	-	-	-	-
-	-	-	-	-	read(D)
-	-	-	read(E)	-	-
-	-	read(H)	-	-	-
-	-	write(C)	-	-	-
-	-	-	-	-	read(B)

Strict 2PL

- Each time you want to read/write an object, obtain a lock to secure permission to read/write object

- Only release locks at commit / abort time —> transaction that writes will block all other readers until the transaction commits or aborts

- Transactions remain isolated and follow serializable schedules

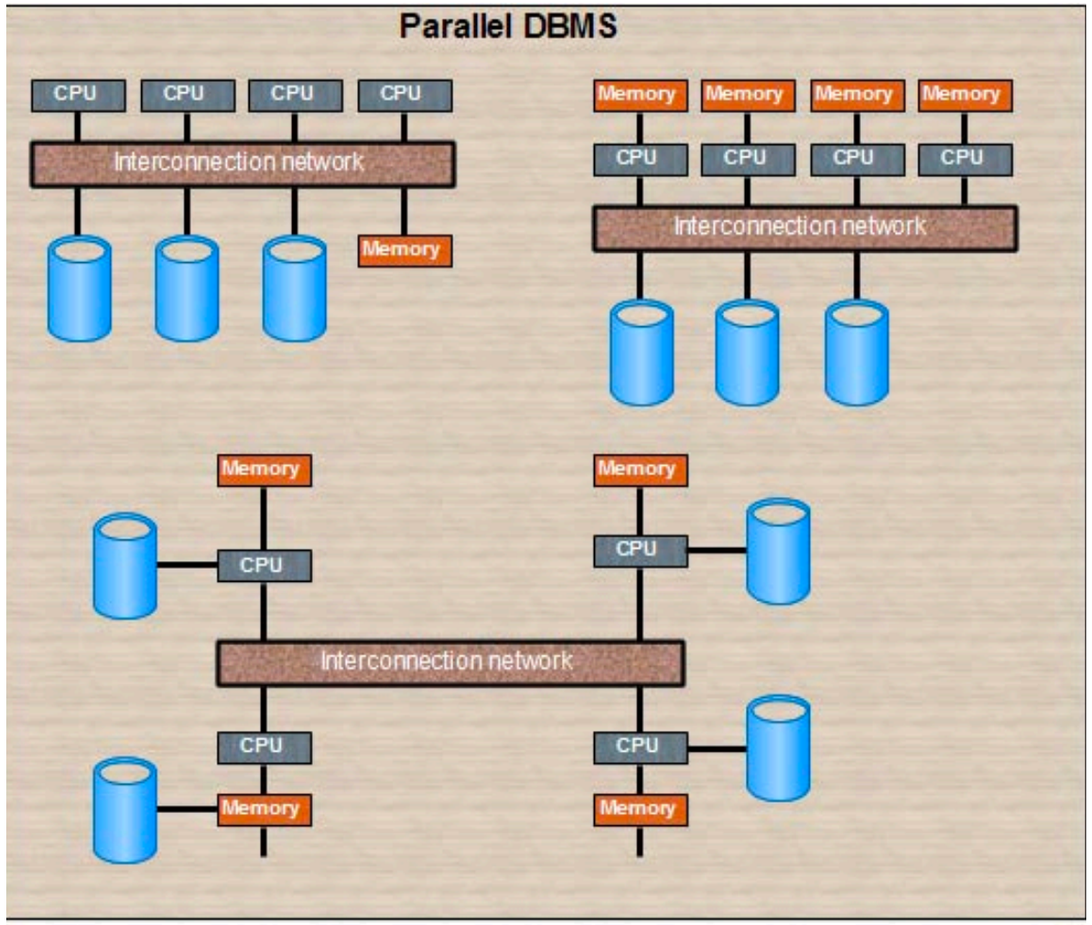


Deadlock may still occur

Big Data Systems

Parallel & Distributed DBs

- Data partitioned across multiple disks
- Allows parallel I/O for better speed-up
- Queries can be run in parallel with each other



Parallel Architectures

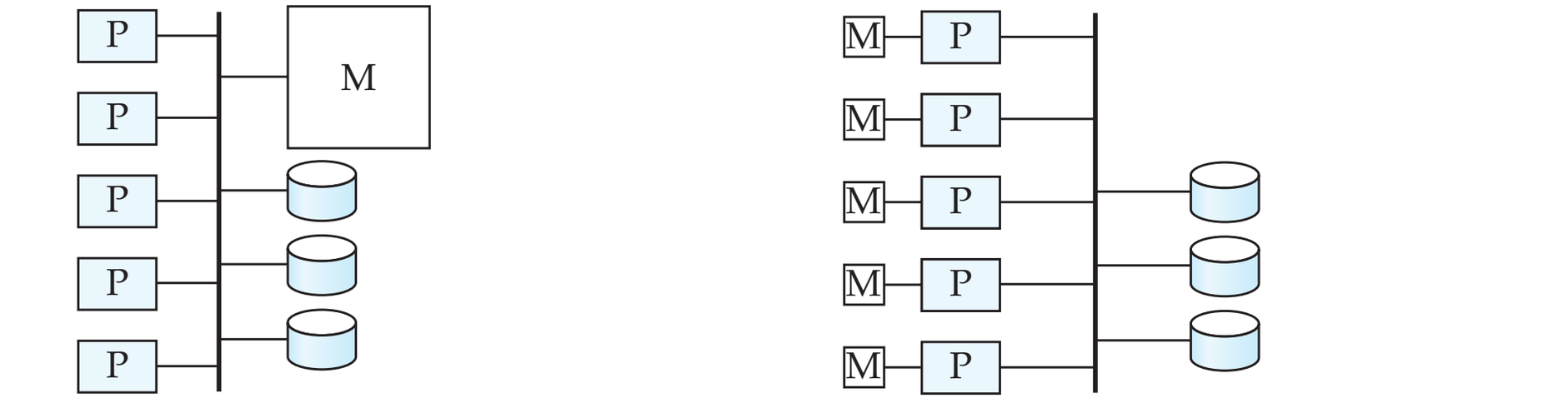


Figure 17.8 (Database System Concepts)

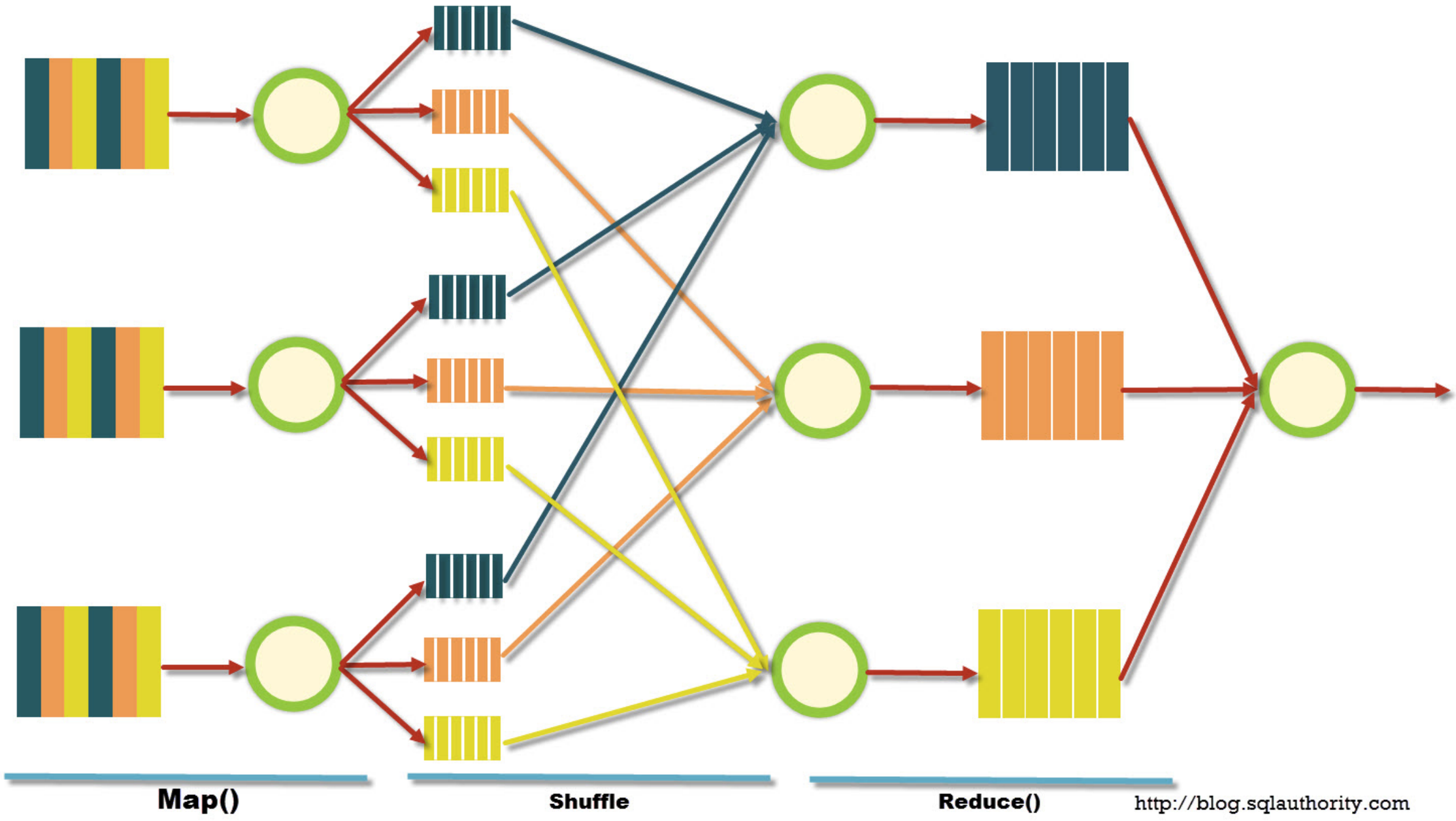
Parallel/Distributed DBMS Issues

- 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

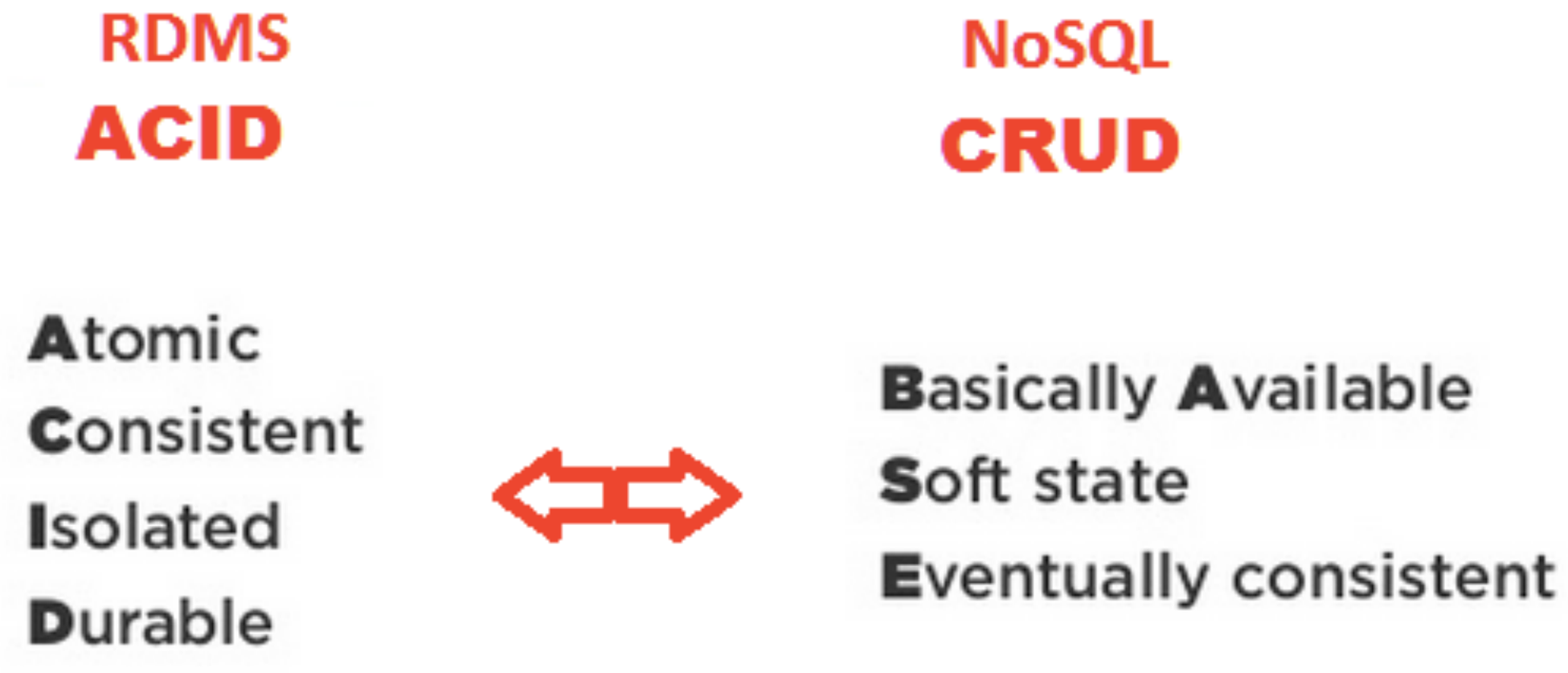
NoSQL

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

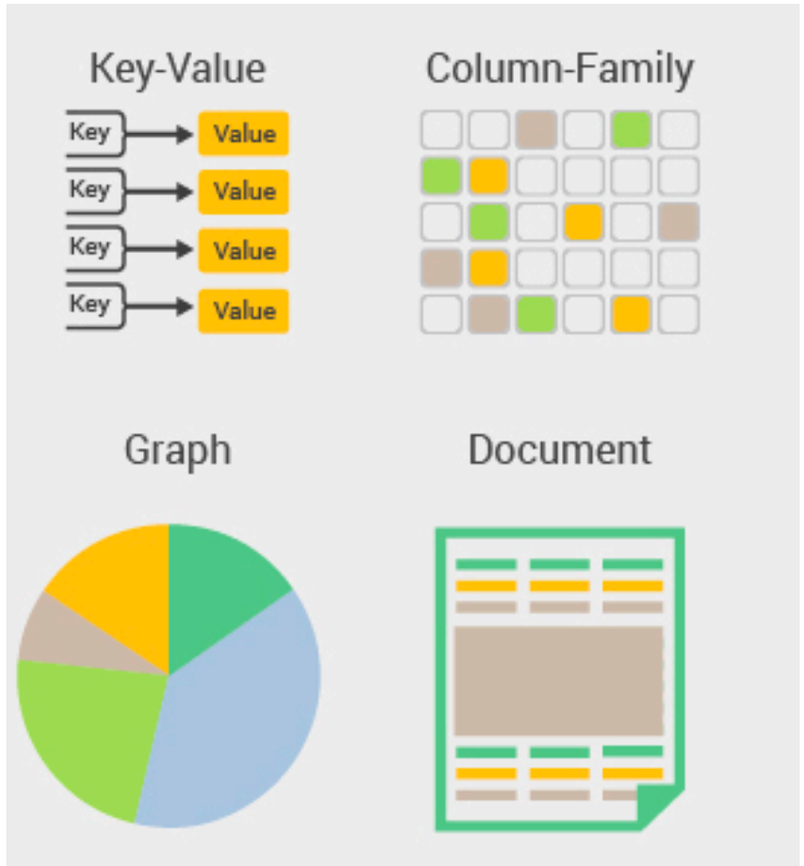
ACID vs BASE



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://www.netsolutionsindia.com/blog/5-things-you-must-consider-adopt-nosql-databases-mongodb/

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?

Quiz Questions

Quiz Questions

- What is metadata and why is it important?
  - Ans: Information about the data — allows us to achieve physical data independence
- What is the result of a relational algebra operation?
  - Ans: Relation

Quiz Questions

- What is a correlated nested query?
  - Ans: Inner query (query in WHERE clause) which uses one or more attributes from relation(s) specified in the outer query
- Why should we care about normal forms?
  - Ans: Normal forms help us avoid bad properties called anomalies (insert, update, delete)

Quiz Questions

- Give an example query on companyDB where you would prefer hash index over B^+-tree and the reverse
- Ans: Prefer hash index for equality searches (WHERE attribute = constant) while B^+-tree for range searches (WHERE attribute < constant1 AND attribute > constant2)