General

- Trade-offs exist between **consistency, availability, and performance.**
- **Indexing strategies** significantly impact query performance.
- **NoSQL databases** are great for unstructured data but may sacrifice strong consistency.
- **Graph databases** are optimized for relationships.

Large Scale Storage & Retrieval: Foundations

- **Foundations of Large-Scale Storage & Retrieval**
  Introduction to Information Storage
  - Information retrieval (IR) is the process of **storing, organizing, and retrieving data efficiently.**
  - IR systems range from **search engines (Google)** to **databases storing medical records.**
  - Key Challenges:
  - **Scalability:** Handling petabytes of data.
  - **Efficiency:** Fast retrieval times (e.g., Google in milliseconds).
  - **Consistency & Availability:** Ensuring fault tolerance and reliability.
- **Searching Strategies**
  - **Linear Search:** Simple but inefficient (**O(n)** time complexity).
  - **Binary Search:** Only works on sorted data (**O(log n)** complexity).
  - **Indexing:** Key to making search efficient; uses external data structures like **B-Trees and Hash Indexes.**
- **Indexing Techniques**
  - **Hash Indexing:** Best for unique lookups (O(1) average case).
  - **B-Trees & B+ Trees:** Used in databases, optimized for disk access.
  - **Inverted Index:** Used in search engines, maps words to documents.

Record – A collection of values for attribues of a single entity instance; a row of a table
Collection – a set of records of the same entity type; a table

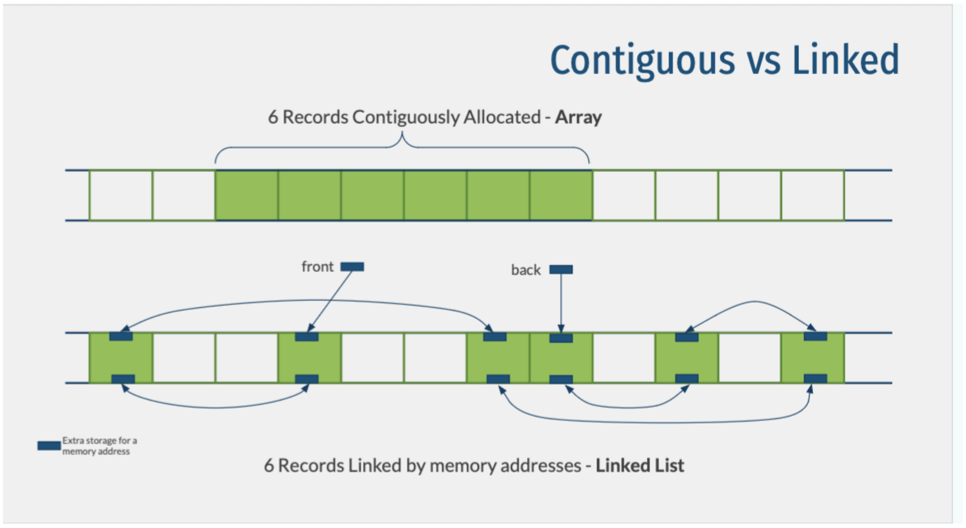
- Trvially, stored in some sequuential order like a list

Search Key – A value for an attribute from the entity type

- Could be >= 1 attribute

n*x bytes of memory = If each record takes up x bytes of memory, then for n records we need
Contiguously Allocated List = All n*x byte are allocated as single "chunk" of memory
Linked List

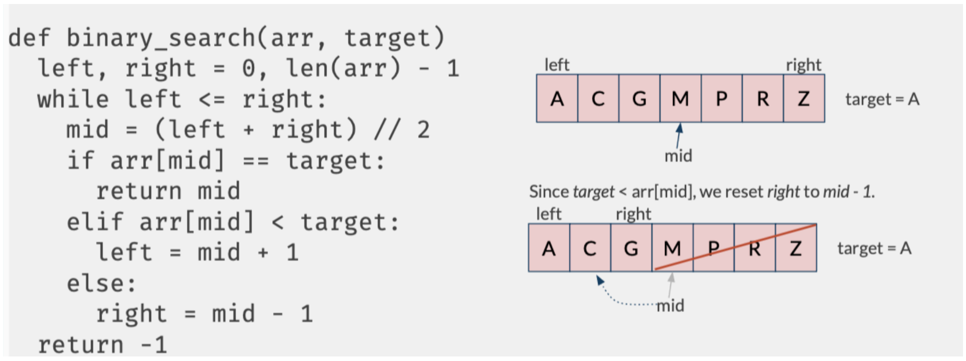
- Each record needs x bytes + additional space for 1 or 2 memory addresses
- Individual records are linked together in a type of chain using memory addresses


Contiguous vs Linked
6 Records Contiguously Allocated - **Array**
front
back
Extra storage for a memory address
6 Records Linked by memory addresses - **Linked List**

Arrays are faster for random access but inserting but the end is slow
Linked Lists are faster for inserting anywhere in the list but random access is slow

Linear Search
Best Case = 1
Worst Case

Binary Search
Best Case = 1
Worst Case= log2n
Read through this code it will explain what binary search is/does:



```
def binary_search(arr, target)
  left, right = 0, len(arr) - 1
  while left <= right:
    mid = (left + right) // 2
    if arr[mid] == target:
      return mid
    elif arr[mid] < target:
      left = mid + 1
    else:
      right = mid - 1
  return -1
```

Since *target* < arr[mid], we reset *right* to *mid - 1*.

Binary Search Tree – a binary tree where every node in left subtree is < parent; every node in right is > parent

AVL Tree – Approx balanced binary search tree

- Maintains balance faster
- AVL Property h(LST) - h(RST) | < or = 1
- Self-balancing

Inserting Imbalance

- LL

- LR
- RL
- RR

Trees

. B-Trees and Indexing

Why B-Trees?

- Used in **database indexing** and **filesystems** (e.g., NTFS, PostgreSQL indexes).
- Keeps tree height small, reducing disk reads.

B+ Trees vs. B-Trees

- **B-Trees:** Store keys and values in internal and leaf nodes.
- **B+ Trees:** Store values only in leaf nodes, making range queries more efficient.

BST Properties

- Left subtree values < Root < Right subtree values.
- Search time complexity: **O(log n)** (balanced), **O(n)** (unbalanced).

AVL Trees

- Self-balancing BST; ensures **O(log n)** search time.
- Rotations used for rebalancing:
- **LL Rotation** (Single Right Rotation)
- **RR Rotation** (Single Left Rotation)
- **LR Rotation** (Left-Right Rotation)
- **RL Rotation** (Right-Left Rotation)

SQL vs No SQL

SQL

Structured Query Langage

- Relational: easy querying on relationships on data between multiple datas

ACID compliant: Atomicity, Consistency, Isolation, Durability

Not good for storing and querying unstructured data where format is unknown
Difficult to scale horizontally

NoSQL

- Document, keyvalue graph, wide column stores

More flexible and simpler to set up
Can shard data across different data stores allowing for distributed databases, making horizontal-scaling much easier
Very large data can be stored

Designed for distributed purposes

Optimistic Concurrency

- Assumes conflicts are unlikely to occur
- Add last updated timestamp and version number columns to every table that can be modified independently of transactions.

Relational Databases vs. NoSQL

ACID vs. BASE

- **ACID:** Ensures strong consistency.
- **Atomicity** - Transactions are all-or-nothing.
- **Consistency** - Data remains valid.
- **Isolation** - Transactions do not interfere.
- **Durability** - Once committed, stays committed.
- **BASE:** Prioritizes availability over consistency.
- **Basically Available, Soft-state, Eventually Consistent.**

SQL vs. NoSQL

Feature	SQL (Relational)	NoSQL (Non-relational)
Schema	Fixed, structured	Flexible, dynamic
Scalability	Vertical (scale-up)	Horizontal (scale-out)
Consistency	Strong ACID Compliance	Eventual Consistency
Examples	MySQL, PostgreSQL	MongoDB, Cassandra, Neo4j

NoSQL

- **CAP Theorem**
  - **Consistency:** All nodes see the same data simultaneously.
  - **Availability:** Every request gets a response (even if stale).
  - **Partition Tolerance:** System functions despite network failures.
- **NoSQL Categories**

Type	Example	Use Case
Key-Value Stores	Redis, DynamoDB	Caching, real-time analytics
Document Stores	MongoDB, CouchDB	JSON-like flexible data models

Column Stores	Apache Cassandra	Time-series & analytics
Graph Databases	Neo4j	Relationship-heavy data

Graph Data Models & Neo4j

Graph Theory Basics

- **Nodes** = Entities (e.g., users, products).
- **Edges** = Relationships between nodes.
- **Traversal Algorithms:** BFS, DFS, Dijkstra’s Shortest Path.

Neo4j & Cypher

- Neo4j is an **ACID-compliant** graph database.
- **Cypher Query Examples:**

```
MATCH (a)-[:FRIEND]->(b) RETURN a, b;
```

Data Replication & Distributed Systems

- Improves **fault tolerance** (if a node crashes, others still serve data).
- Enables **load balancing** (distribute requests across multiple servers).

Replication Strategies

- **Master-Slave Replication:** One node handles writes; others replicate for reads.
- **Multi-Master Replication:** Multiple nodes accept writes; requires conflict resolution.
- **Eventual Consistency:** Updates propagate over time (**used in NoSQL systems** like DynamoDB)