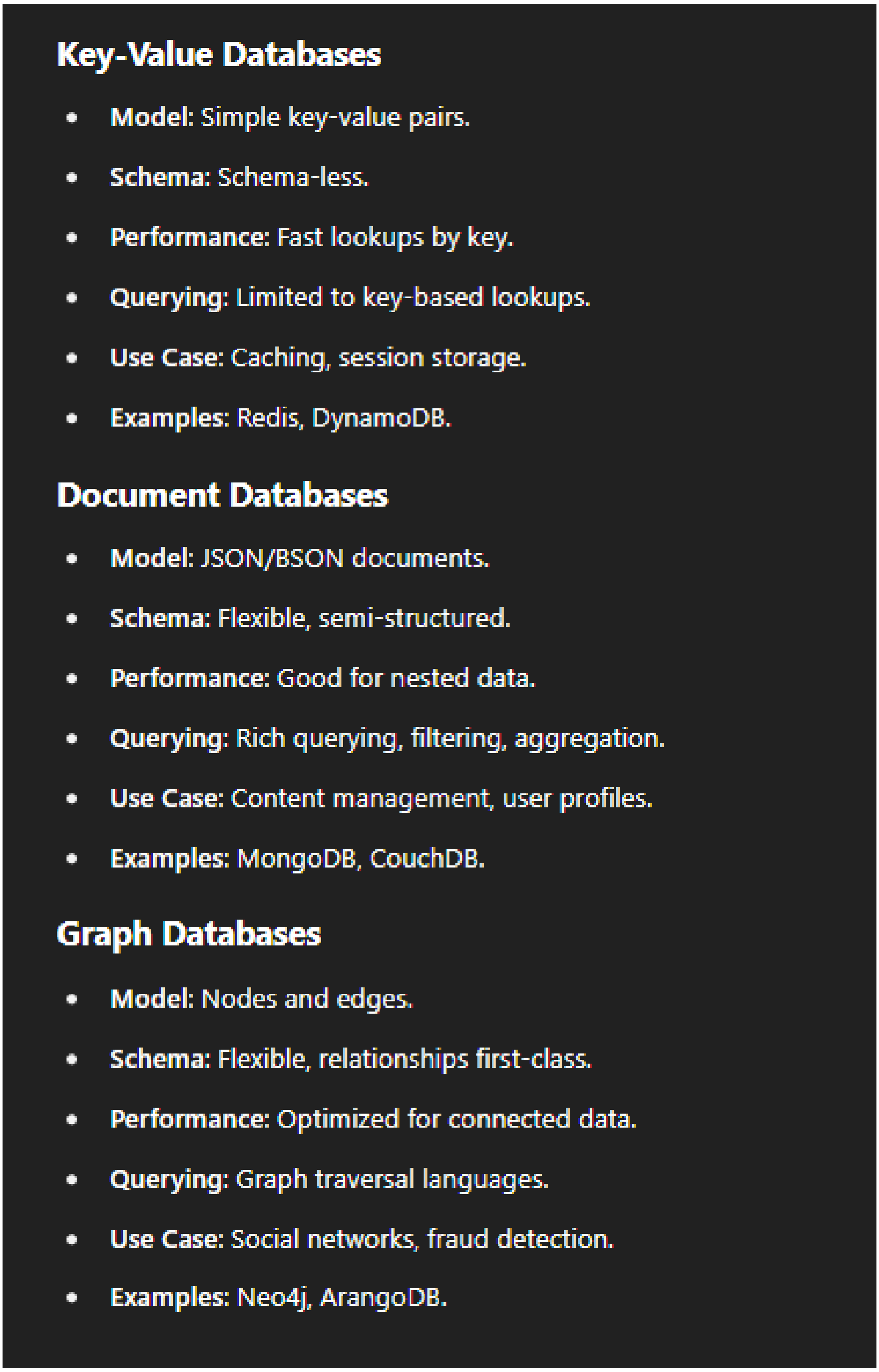
CO4 -HOME ASSIGNMENT

1. Compare and contrast the characteristics of key-value, document, and graph NoSQL databases.



Key-Value Databases

- **Model:** Simple key-value pairs.
- **Schema:** Schema-less.
- **Performance:** Fast lookups by key.
- **Querying:** Limited to key-based lookups.
- **Use Case:** Caching, session storage.
- **Examples:** Redis, DynamoDB.

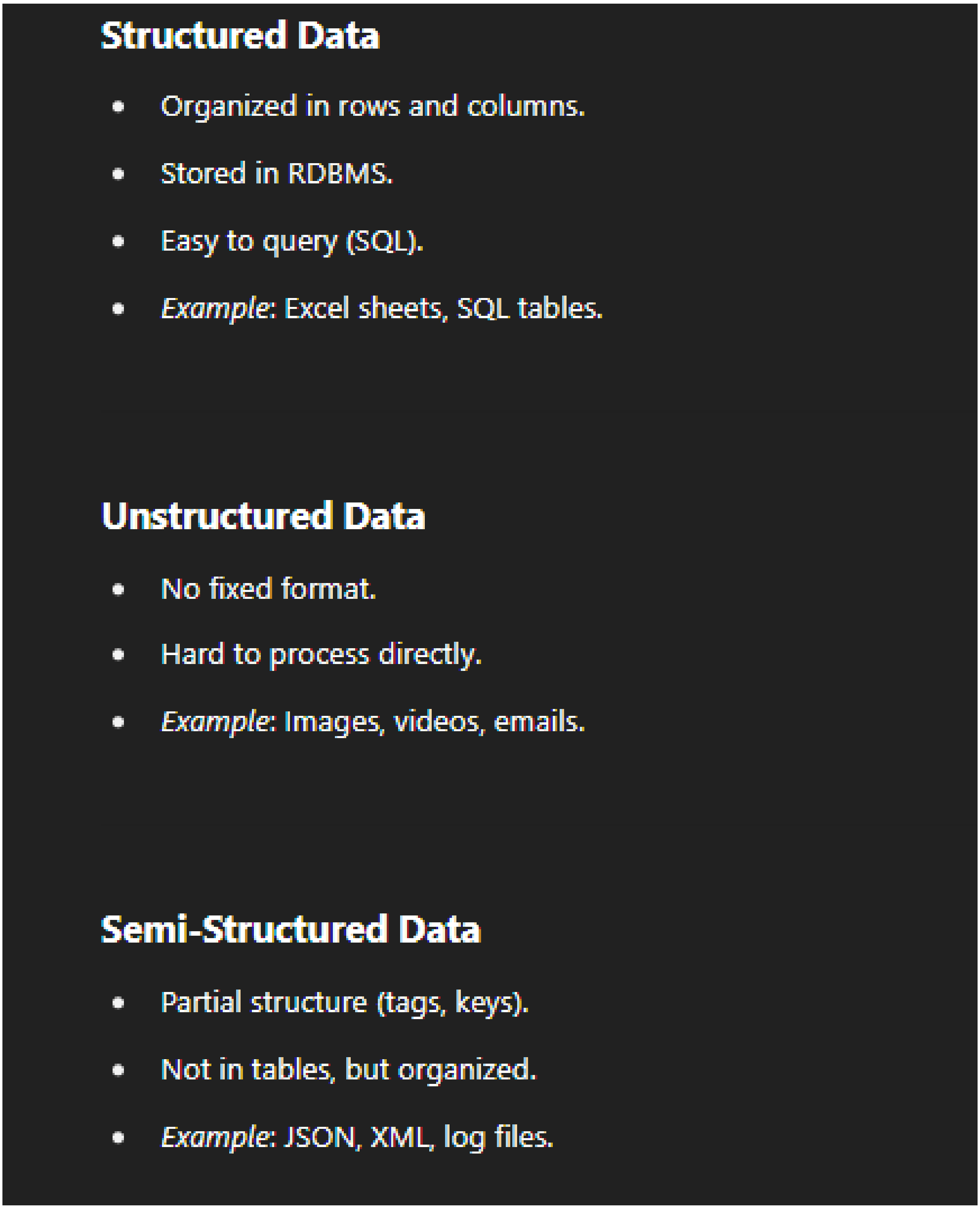
Document Databases

- **Model:** JSON/BSON documents.
- **Schema:** Flexible, semi-structured.
- **Performance:** Good for nested data.
- **Querying:** Rich querying, filtering, aggregation.
- **Use Case:** Content management, user profiles.
- **Examples:** MongoDB, CouchDB.

Graph Databases

- **Model:** Nodes and edges.
- **Schema:** Flexible, relationships first-class.
- **Performance:** Optimized for connected data.
- **Querying:** Graph traversal languages.
- **Use Case:** Social networks, fraud detection.
- **Examples:** Neo4j, ArangoDB.

2. Describe structured, unstructured, and semi-structured data types in bigdata.



Structured Data

- Organized in rows and columns.
- Stored in RDBMS.
- Easy to query (SQL).
- *Example*: Excel sheets, SQL tables.

Unstructured Data

- No fixed format.
- Hard to process directly.
- *Example*: Images, videos, emails.

Semi-Structured Data

- Partial structure (tags, keys).
- Not in tables, but organized.
- *Example*: JSON, XML, log files.

3. Consider the following simplified schema for an airline reservation system: { "_id": ObjectId, "flightNumber": String, "departure": { "airport": String, "city": String, "time": Date }, "arrival": { "airport": String, "city": String, "time": Date }, "seatsAvailable": Number } i. Create a new reservation for a passenger on a specific flight. ii. Find all flights departing from a specific airport. iii. Find all passengers with a specific passport number. iv. Find all reservations for a specific flight. v. Update the contact details of a specific passenger.

i. Create a new reservation for a passenger on a specific flight

```
db.reservations.insertOne({
 flightNumber: "AI101",
 passengerId: ObjectId("PASSENGER_OBJECT_ID"),
 reservationDate: new Date(),
 seatNumber: "12A"
});
```

---

ii. Find all flights departing from a specific airport

```
db.flights.find({ "departure.airport": "JFK" });
```

---

iii. Find all passengers with a specific passport number

```
db.passengers.find({ passportNumber: "A1234567" });
```

---

iv. Find all reservations for a specific flight

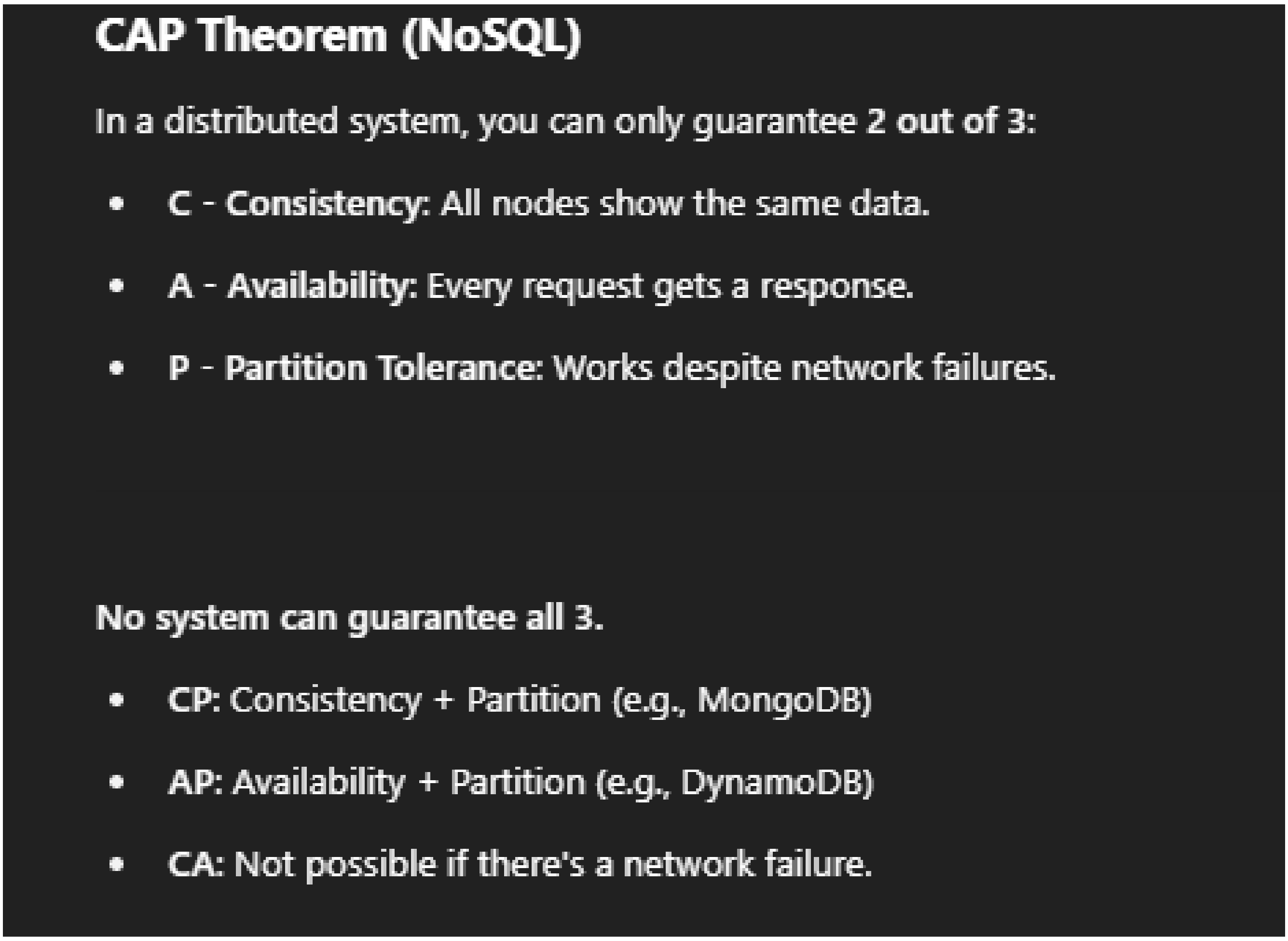
```
db.reservations.find({ flightNumber: "AI101" });
```

---

v. Update the contact details of a specific passenger

```
db.passengers.updateOne(
 { _id: ObjectId("PASSENGER_OBJECT_ID") },
 { $set: { contact: { phone: "9876543210", email: "newemail@example.com" } } }
);
```

4. Explain the CAP theorem in NOSQL databases.



CAP Theorem (NoSQL)

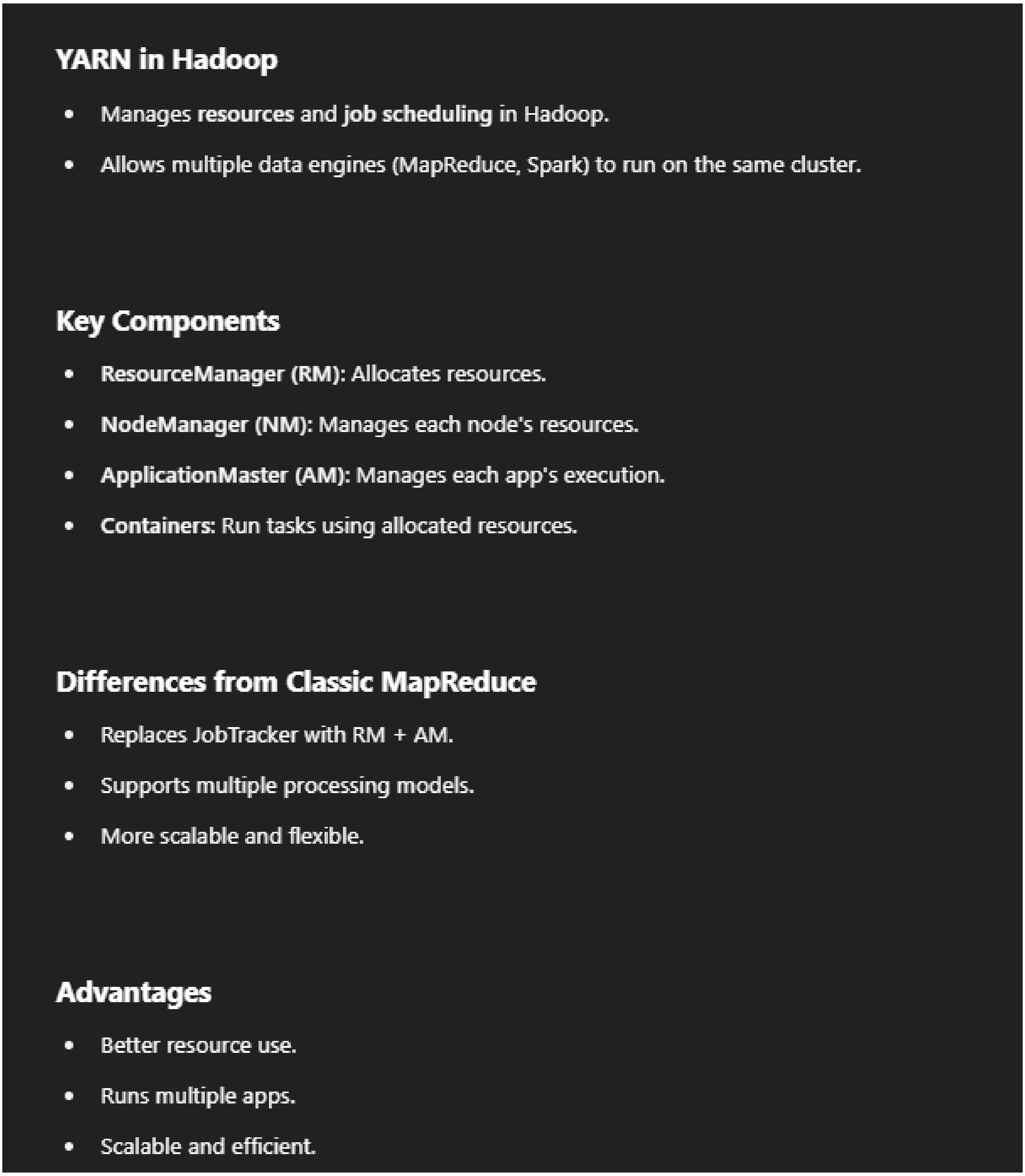
In a distributed system, you can only guarantee **2 out of 3:**

- **C - Consistency:** All nodes show the same data.
- **A - Availability:** Every request gets a response.
- **P - Partition Tolerance:** Works despite network failures.

No system can guarantee all 3.

- CP: Consistency + Partition (e.g., MongoDB)
- AP: Availability + Partition (e.g., DynamoDB)
- CA: Not possible if there's a network failure.

5. Describe the role of YARN (Yet Another Resource Negotiator) in Hadoop architecture. How does it manage resources and schedule jobs in a cluster? What are the architectural differences and advantages of each?



YARN in Hadoop

- Manages **resources** and **job scheduling** in Hadoop.
- Allows multiple data engines (MapReduce, Spark) to run on the same cluster.

Key Components

- **ResourceManager (RM):** Allocates resources.
- **NodeManager (NM):** Manages each node's resources.
- **ApplicationMaster (AM):** Manages each app's execution.
- **Containers:** Run tasks using allocated resources.

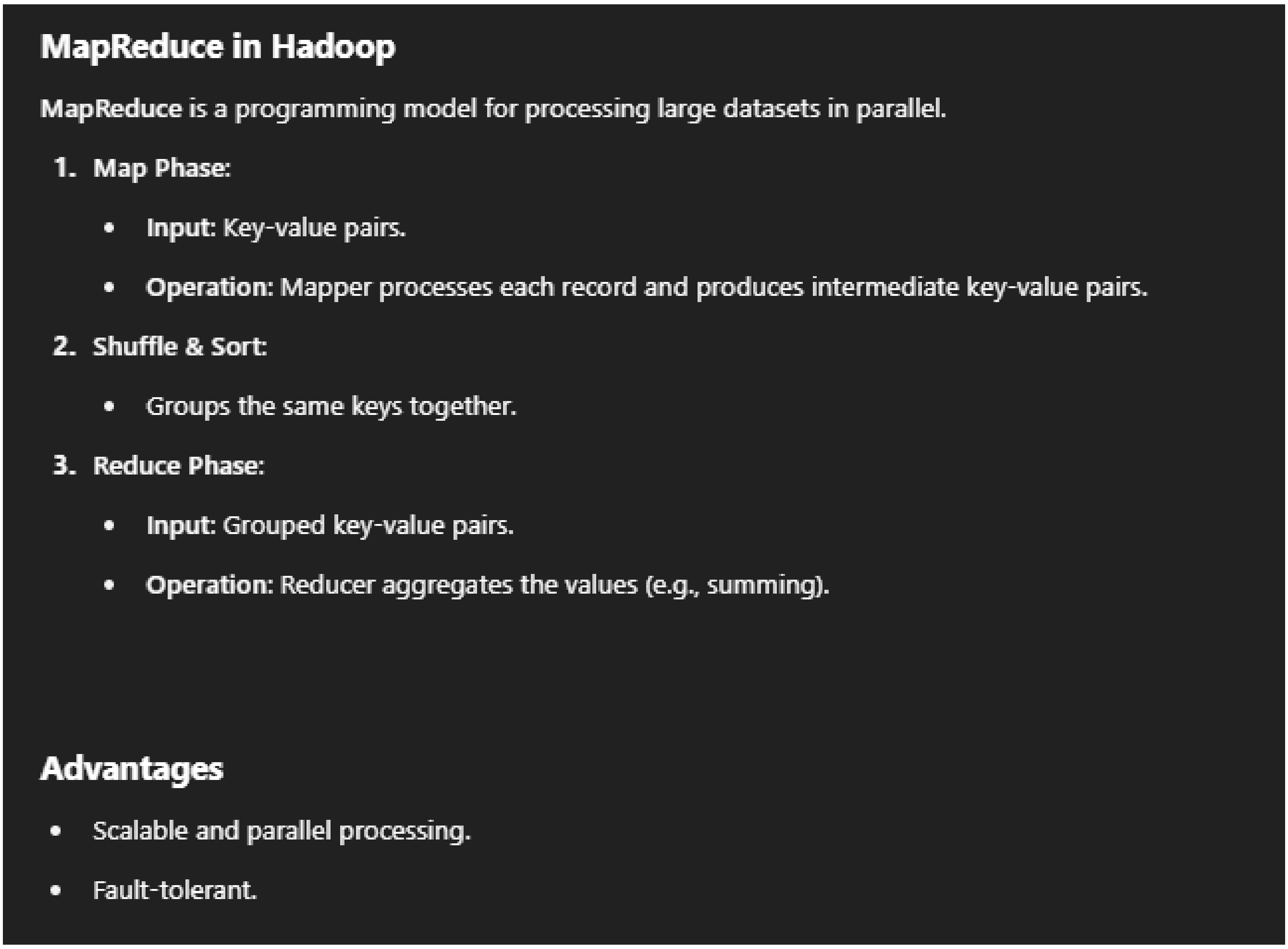
Differences from Classic MapReduce

- Replaces JobTracker with RM + AM.
- Supports multiple processing models.
- More scalable and flexible.

Advantages

- Better resource use.
- Runs multiple apps.
- Scalable and efficient.

6. Explain mapreduce programming in Hadoop.



MapReduce in Hadoop

MapReduce is a programming model for processing large datasets in parallel.

1. **Map Phase:**
   - **Input:** Key-value pairs.
   - **Operation:** Mapper processes each record and produces intermediate key-value pairs.
2. **Shuffle & Sort:**
   - Groups the same keys together.
3. **Reduce Phase:**
   - **Input:** Grouped key-value pairs.
   - **Operation:** Reducer aggregates the values (e.g., summing).

Advantages

- Scalable and parallel processing.
- Fault-tolerant.