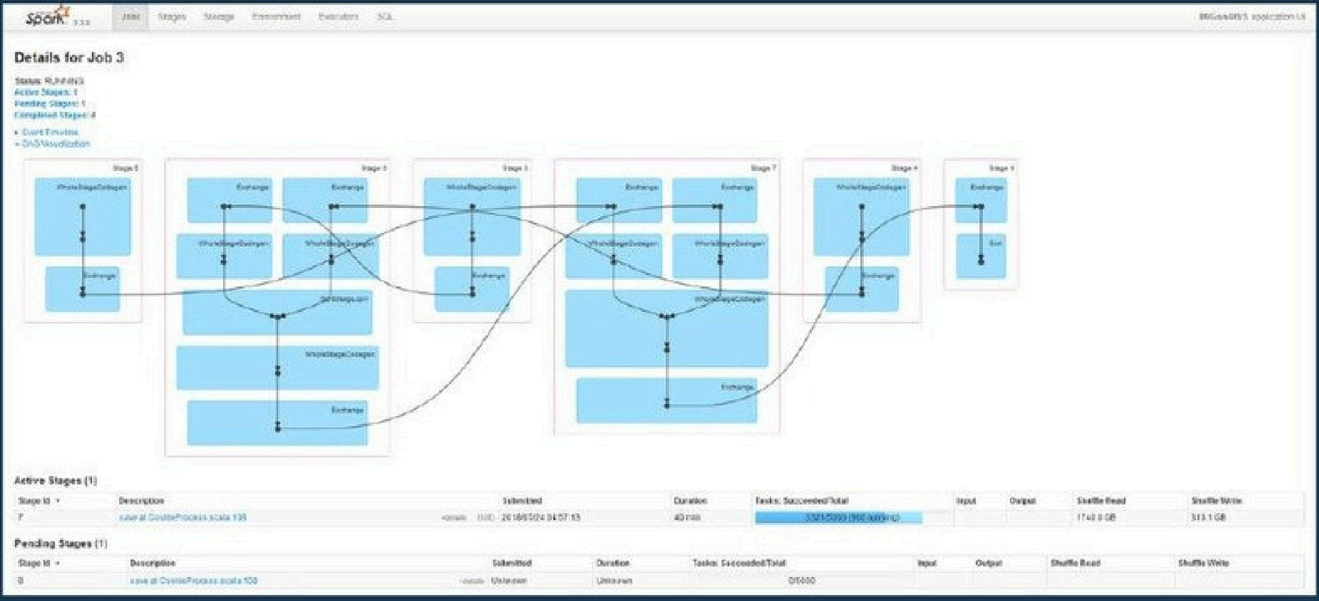
Apache Spark DAG Features

Directed Acyclic Graph - Brain of Spark Job



Details for Job 3

Status: RUNNING
Active Stages: 1
Pending Stages: 1
Completed Stages: 4

- Event Timeline
- DAG Visualization

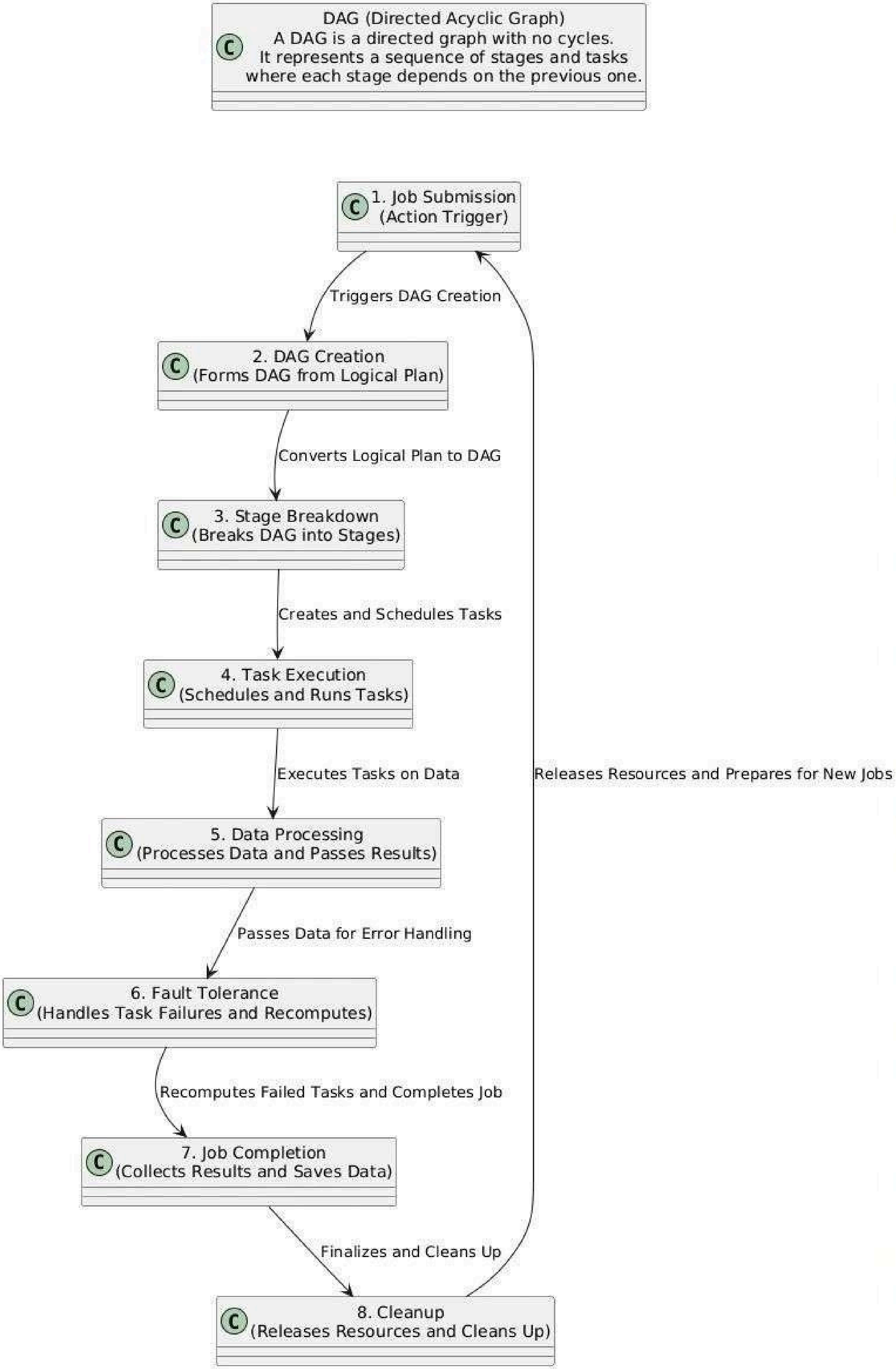
Active Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	save at CombleProcess.scala 135 +details (kill)	2018/05/24 04:57:13	40 min	3321/5000 (960 running)			1740.0 GB	313.1 GB

Pending Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	save at CombleProcess.scala 100 +details	Unknown	Unknown	0/5000				

Spark DAG Operation



DAG (Directed Acyclic Graph)
A DAG is a directed graph with no cycles.
It represents a sequence of stages and tasks
where each stage depends on the previous one.

1. Job Submission
(Action Trigger)

Triggers DAG Creation

2. DAG Creation
(Forms DAG from Logical Plan)

Converts Logical Plan to DAG

3. Stage Breakdown
(Breaks DAG into Stages)

Creates and Schedules Tasks

4. Task Execution
(Schedules and Runs Tasks)

Executes Tasks on Data

5. Data Processing
(Processes Data and Passes Results)

Passes Data for Error Handling

6. Fault Tolerance
(Handles Task Failures and Recomputes)

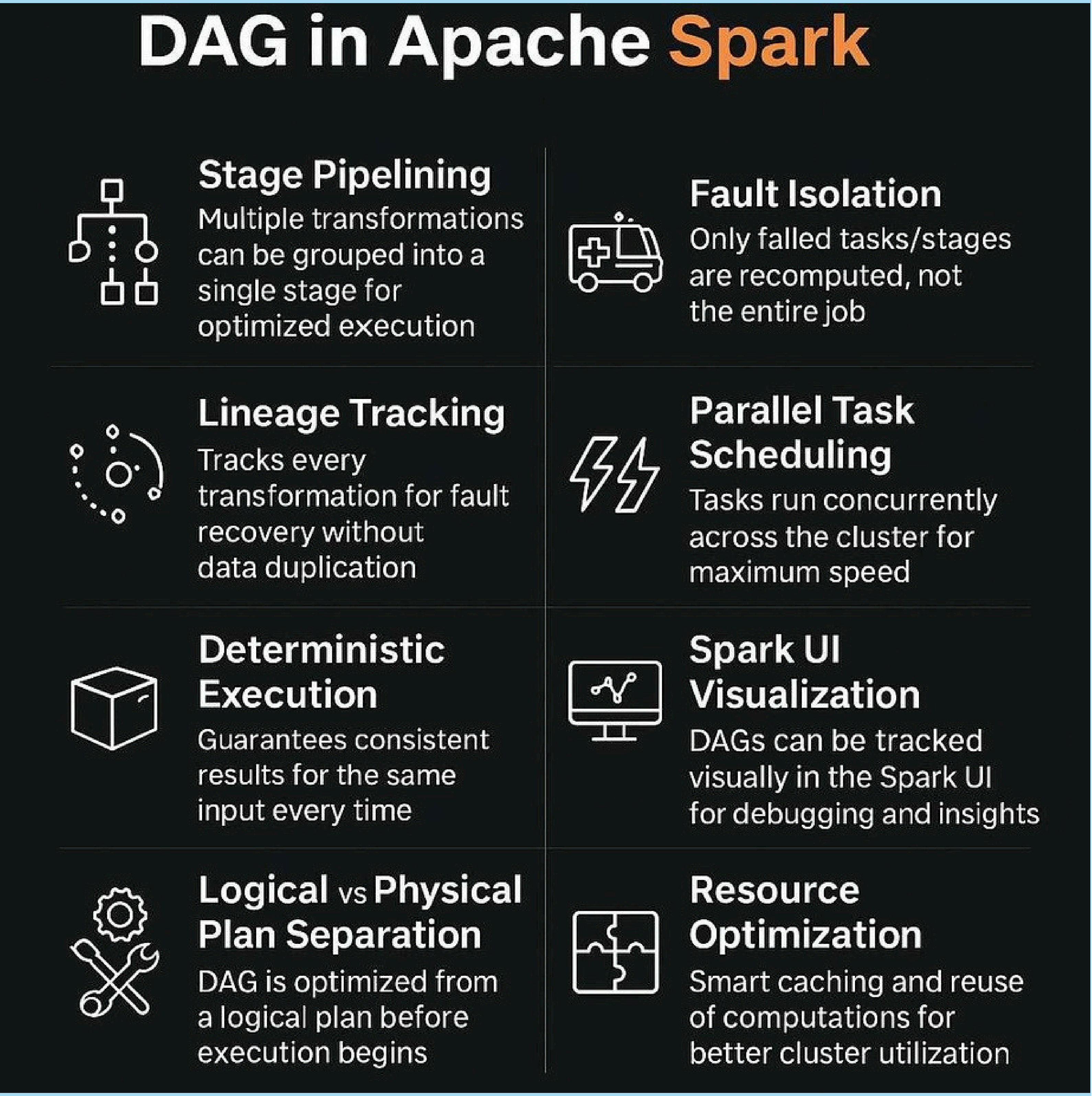
Recomputes Failed Tasks and Completes Job

7. Job Completion
(Collects Results and Saves Data)

Finalizes and Cleans Up

8. Cleanup
(Releases Resources and Cleans Up)

Releases Resources and Prepares for New Jobs



DAG in Apache Spark

Stage Pipelining

Multiple transformations can be grouped into a single stage for optimized execution

Fault Isolation

Only falled tasks/stages are recomputed, not the entire job

Lineage Tracking

Tracks every transformation for fault recovery without data duplication

Parallel Task Scheduling

Tasks run concurrently across the cluster for maximum speed

Deterministic Execution

Guarantees consistent results for the same input every time

Spark UI Visualization

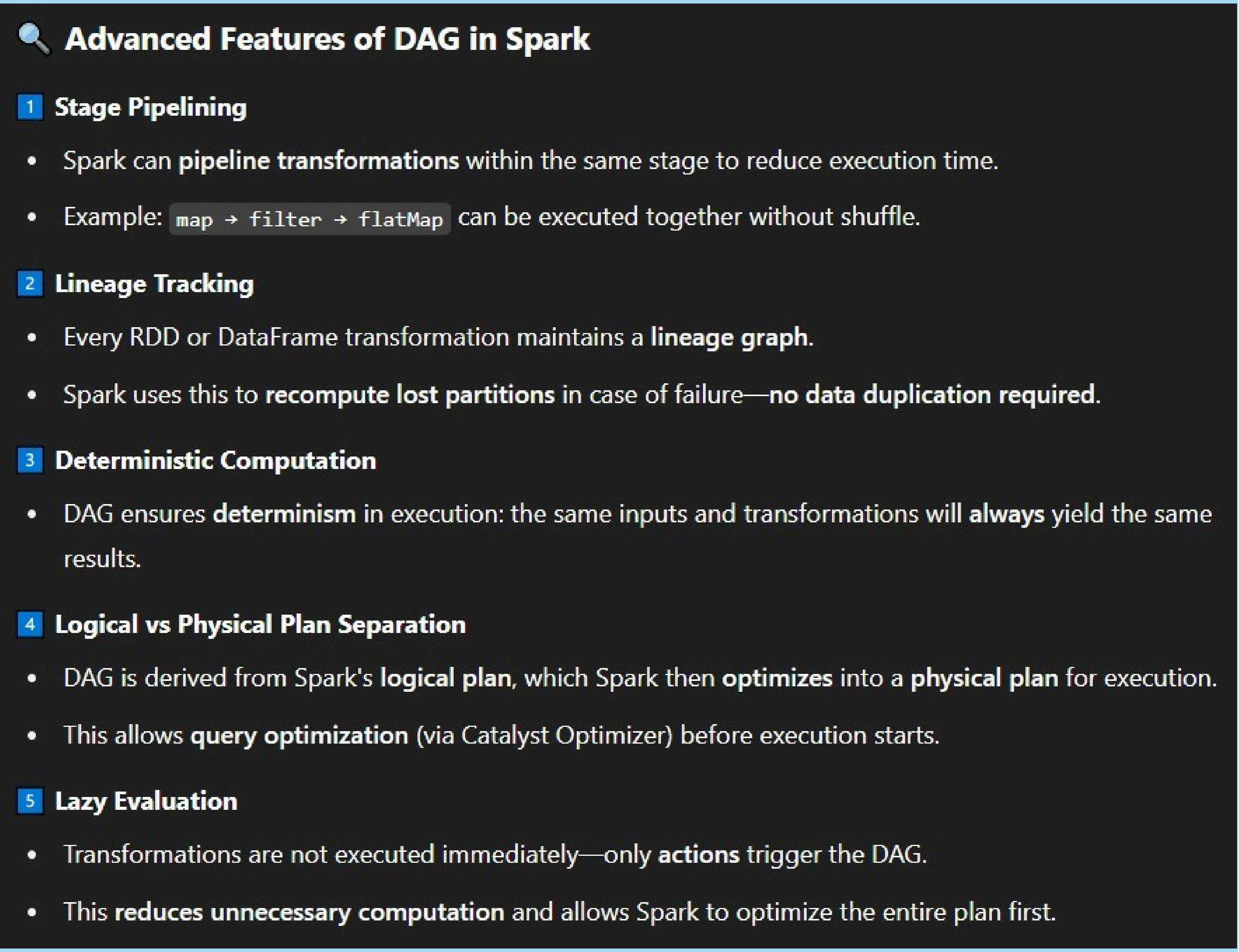
DAGs can be tracked visually in the Spark UI for debugging and insights

Logical vs Physical Plan Separation

DAG is optimized from a logical plan before execution begins

Resource Optimization

Smart caching and reuse of computations for better cluster utilization



🔍 Advanced Features of DAG in Spark

1️⃣ Stage Pipelining

- Spark can **pipeline transformations** within the same stage to reduce execution time.
- Example: `map → filter → flatMap` can be executed together without shuffle.

2️⃣ Lineage Tracking

- Every RDD or DataFrame transformation maintains a **lineage graph**.
- Spark uses this to **recompute lost partitions** in case of failure—**no data duplication required**.

3️⃣ Deterministic Computation

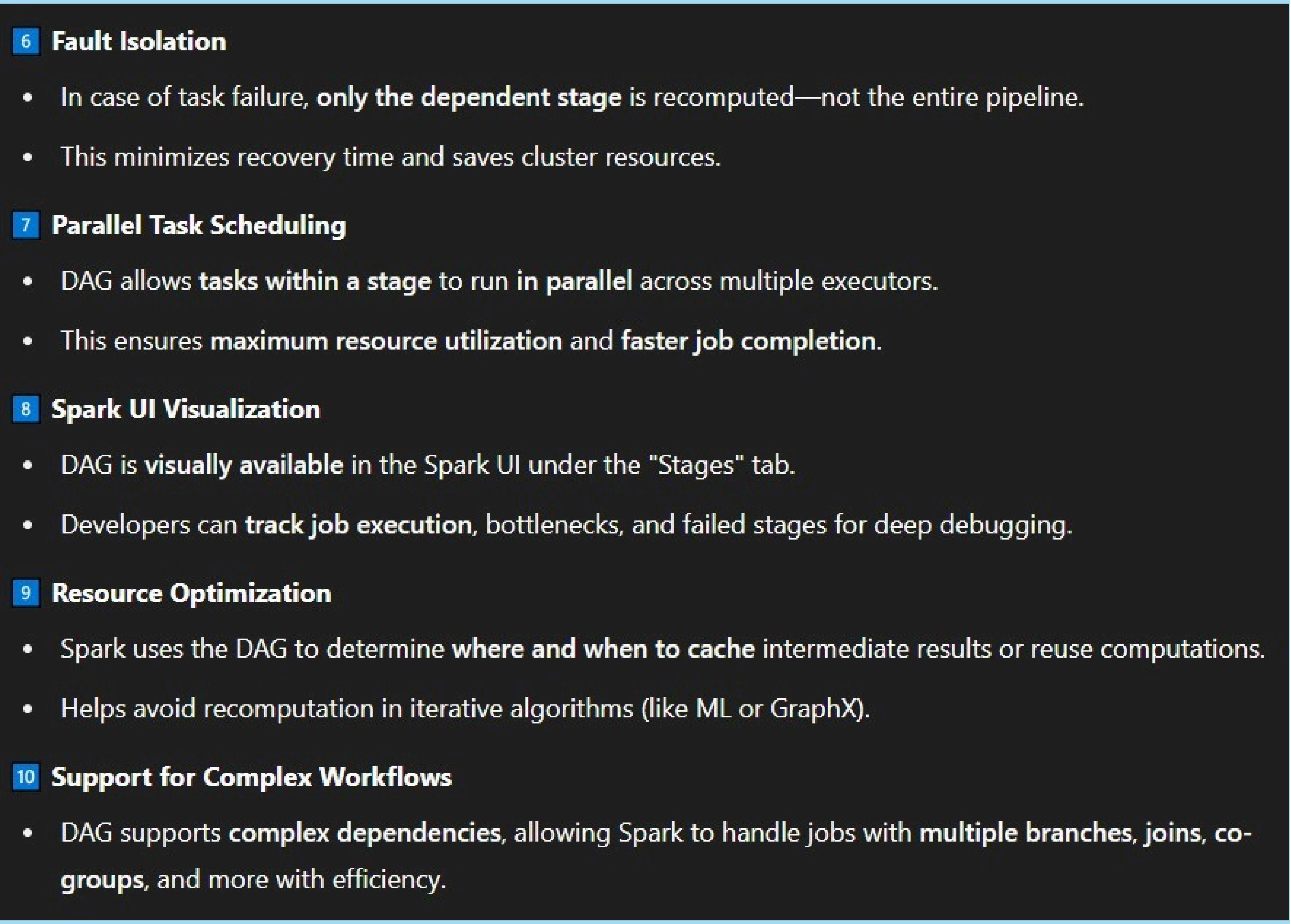
- DAG ensures **determinism** in execution: the same inputs and transformations will **always** yield the same results.

4️⃣ Logical vs Physical Plan Separation

- DAG is derived from Spark's **logical plan**, which Spark then **optimizes** into a **physical plan** for execution.
- This allows **query optimization** (via Catalyst Optimizer) before execution starts.

5️⃣ Lazy Evaluation

- Transformations are not executed immediately—only **actions** trigger the DAG.
- This **reduces unnecessary computation** and allows Spark to optimize the entire plan first.



6️⃣ Fault Isolation

- In case of task failure, **only the dependent stage** is recomputed—not the entire pipeline.
- This minimizes recovery time and saves cluster resources.

7️⃣ Parallel Task Scheduling

- DAG allows **tasks within a stage** to run **in parallel** across multiple executors.
- This ensures **maximum resource utilization** and **faster job completion**.

8️⃣ Spark UI Visualization

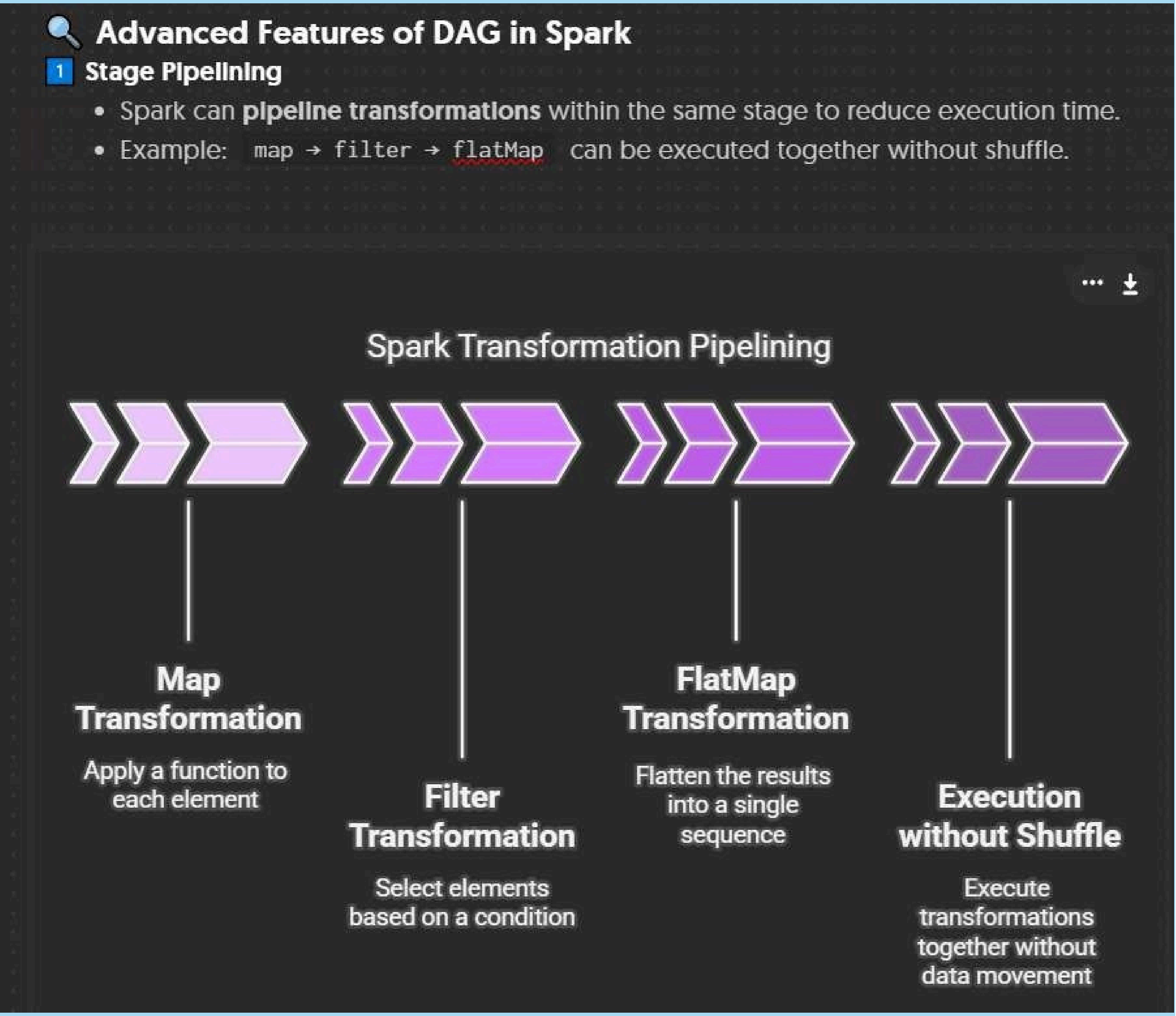
- DAG is **visually available** in the Spark UI under the "Stages" tab.
- Developers can **track job execution**, bottlenecks, and failed stages for deep debugging.

9️⃣ Resource Optimization

- Spark uses the DAG to determine **where and when to cache** intermediate results or reuse computations.
- Helps avoid recomputation in iterative algorithms (like ML or GraphX).

🔟 Support for Complex Workflows

- DAG supports **complex dependencies**, allowing Spark to handle jobs with **multiple branches**, **joins**, **co-groups**, and more with efficiency.



🔍 Advanced Features of DAG in Spark

1️⃣ Stage Pipelining

- Spark can **pipeline transformations** within the same stage to reduce execution time.
- Example: `map → filter → flatMap` can be executed together without shuffle.

Spark Transformation Pipelining

Map Transformation

Apply a function to each element

Filter Transformation

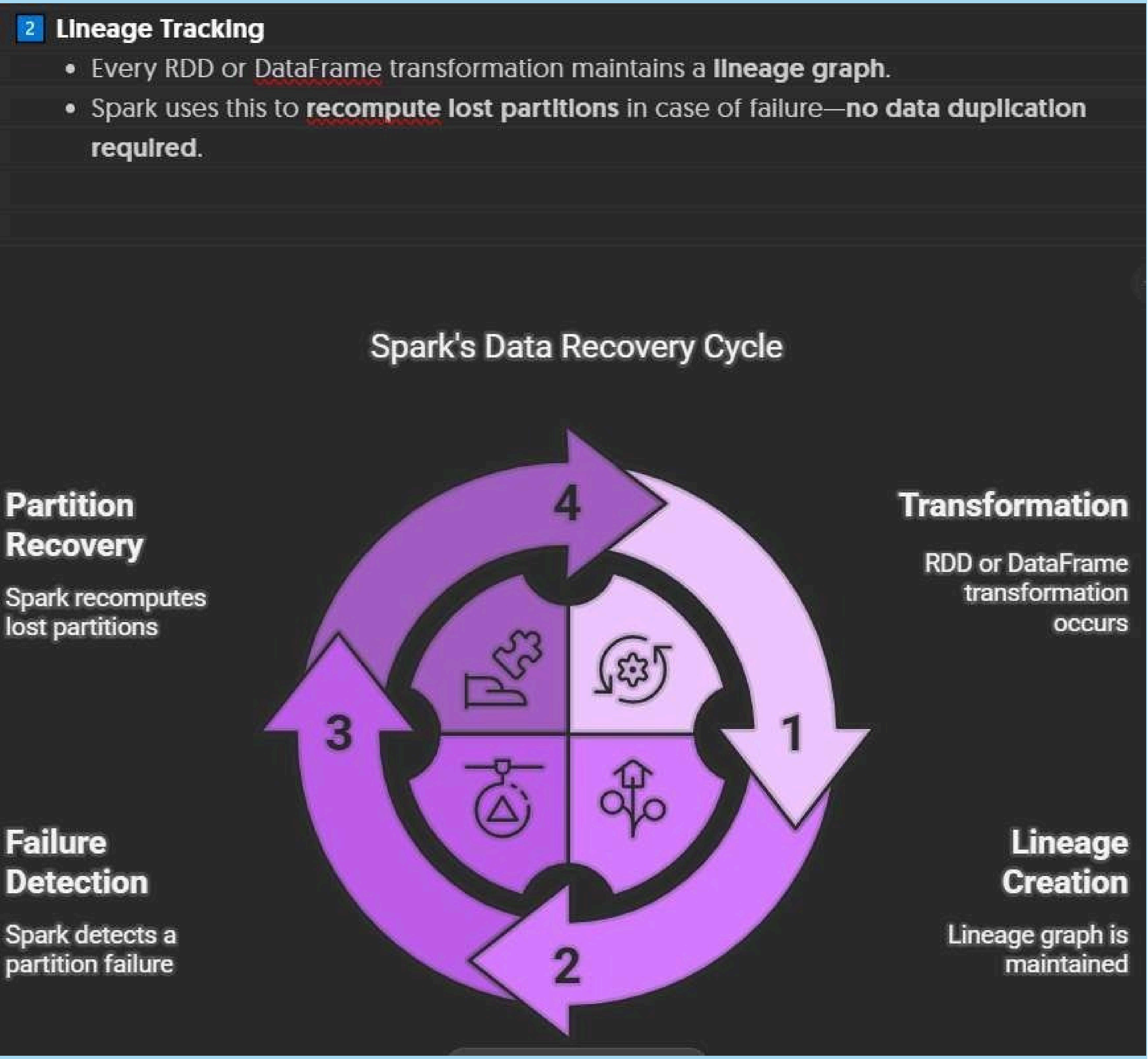
Select elements based on a condition

FlatMap Transformation

Flatten the results into a single sequence

Execution without Shuffle

Execute transformations together without data movement



2️⃣ Lineage Tracking

- Every RDD or DataFrame transformation maintains a **lineage graph**.
- Spark uses this to **recompute lost partitions** in case of failure—**no data duplication required**.

Spark's Data Recovery Cycle

Partition Recovery

Spark recomputes lost partitions

Transformation

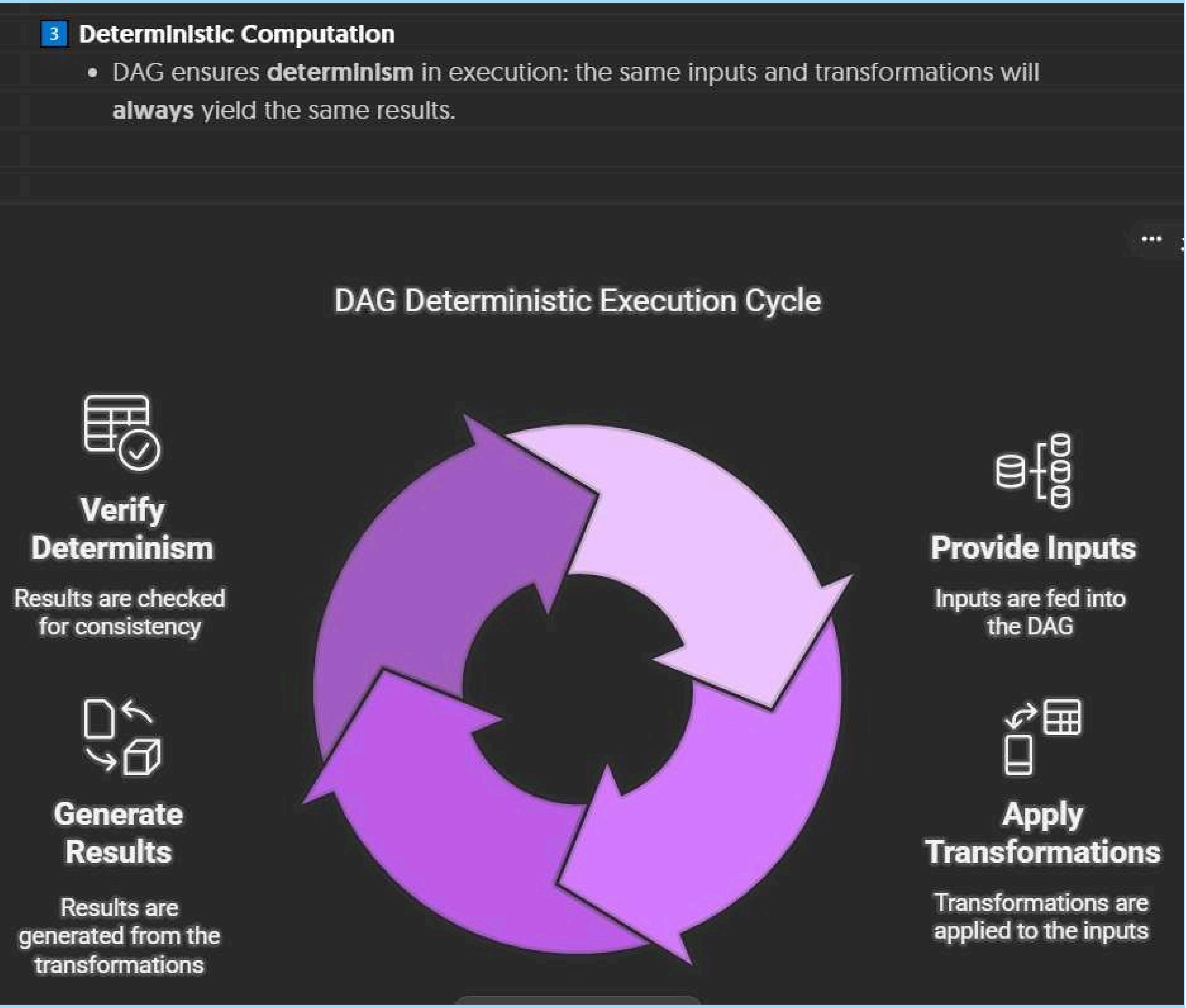
RDD or DataFrame transformation occurs

Failure Detection

Spark detects a partition failure

Lineage Creation

Lineage graph is maintained



3 Deterministic Computation

- DAG ensures **determinism** in execution: the same inputs and transformations will **always** yield the same results.

DAG Deterministic Execution Cycle

Provide Inputs

Inputs are fed into the DAG

Apply Transformations

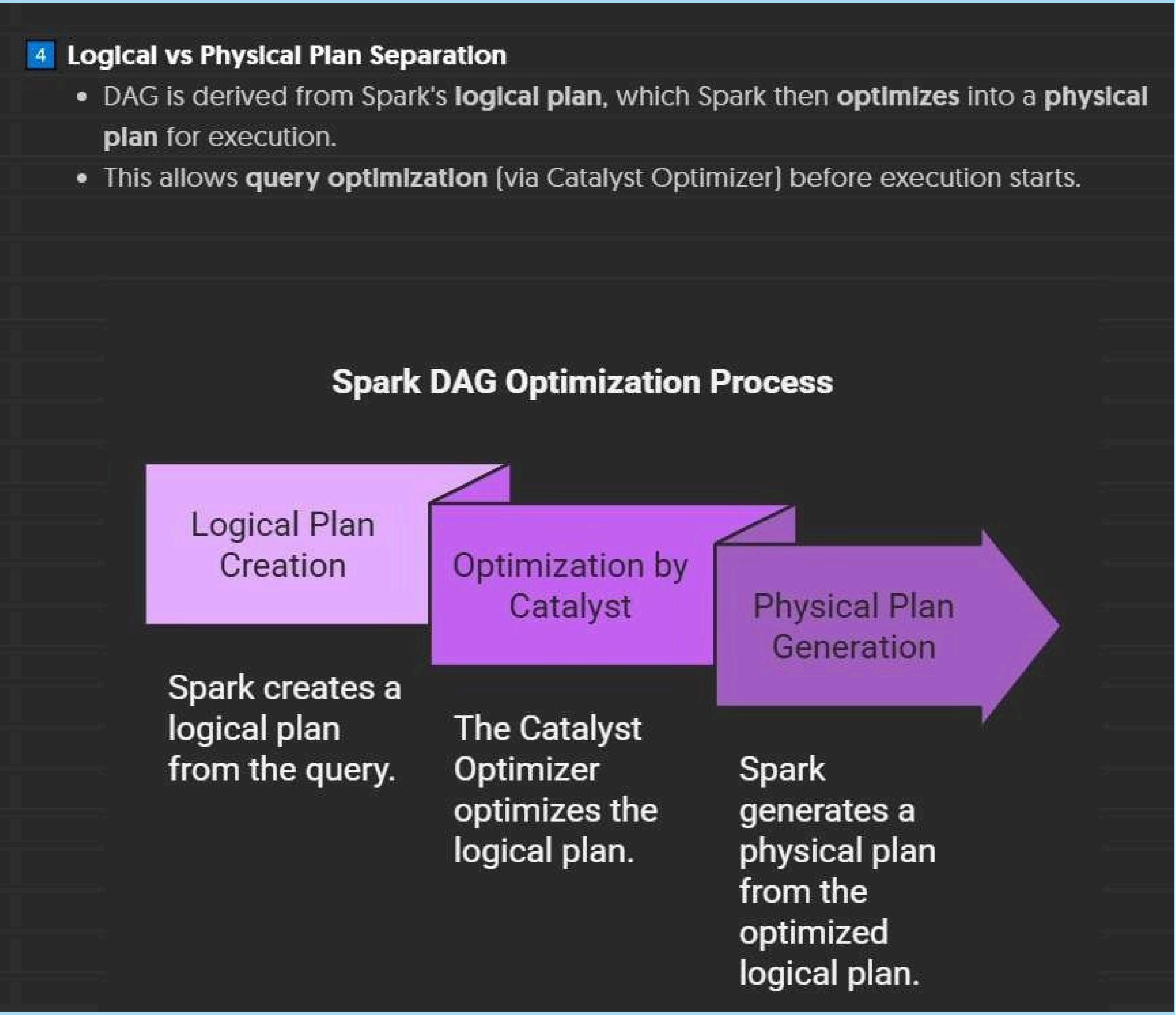
Transformations are applied to the inputs

Generate Results

Results are generated from the transformations

Verify Determinism

Results are checked for consistency



4 Logical vs Physical Plan Separation

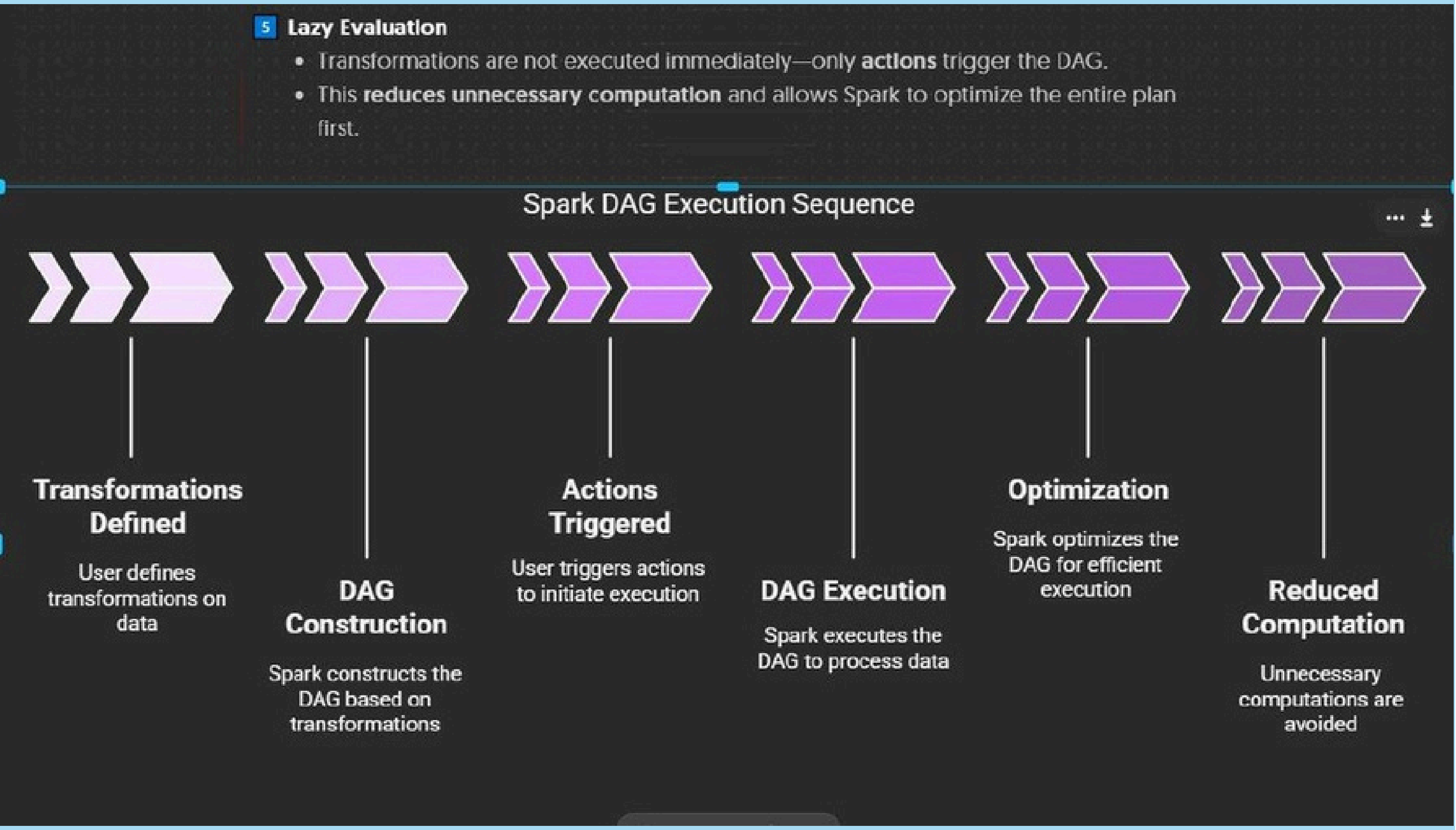
- DAG is derived from Spark's **logical plan**, which Spark then **optimizes** into a **physical plan** for execution.
- This allows **query optimization** (via Catalyst Optimizer) before execution starts.

Spark DAG Optimization Process

Spark creates a logical plan from the query.

The Catalyst Optimizer optimizes the logical plan.

Spark generates a physical plan from the optimized logical plan.



5 **Lazy Evaluation**

- Transformations are not executed immediately—only **actions** trigger the DAG.
- This **reduces unnecessary computation** and allows Spark to optimize the entire plan first.

Spark DAG Execution Sequence

Transformations Defined

User defines transformations on data

DAG Construction

Spark constructs the DAG based on transformations

Actions Triggered

User triggers actions to initiate execution

DAG Execution

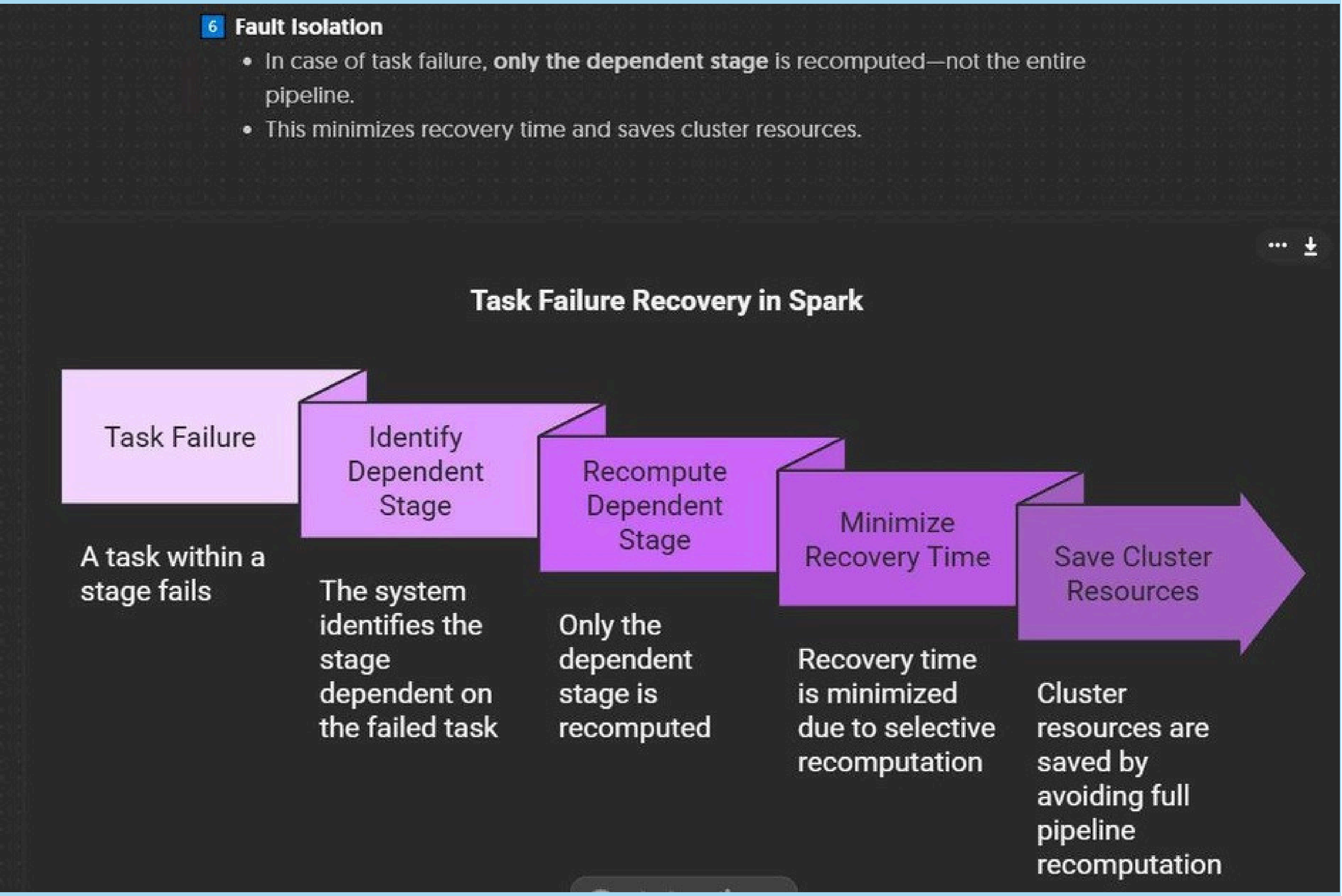
Spark executes the DAG to process data

Optimization

Spark optimizes the DAG for efficient execution

Reduced Computation

Unnecessary computations are avoided



6 Fault Isolation

- In case of task failure, **only the dependent stage** is recomputed—not the entire pipeline.
- This minimizes recovery time and saves cluster resources.

Task Failure Recovery in Spark

Task Failure

A task within a stage fails

Identify Dependent Stage

The system identifies the stage dependent on the failed task

Recompute Dependent Stage

Only the dependent stage is recomputed

Minimize Recovery Time

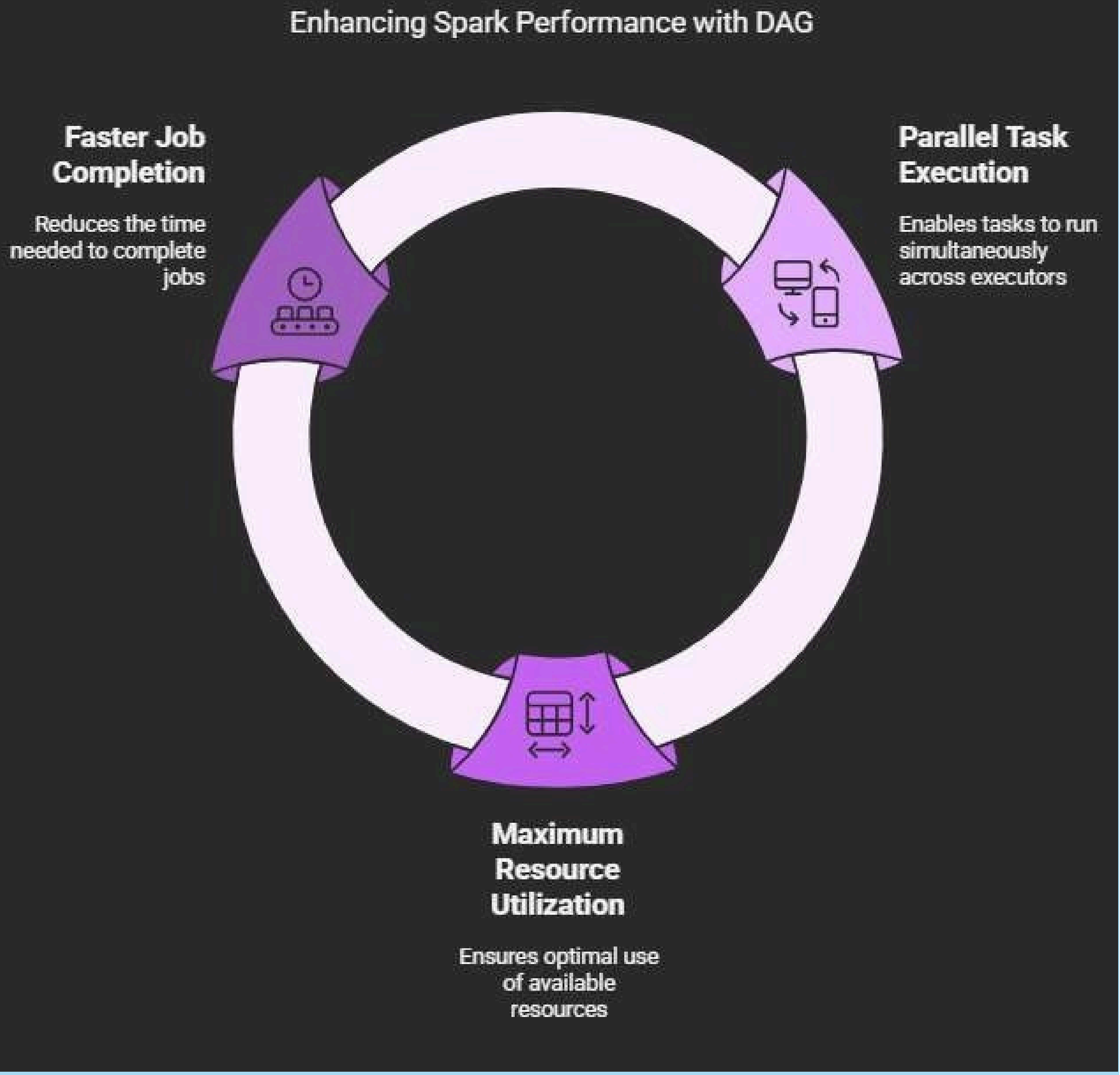
Recovery time is minimized due to selective recomputation

Save Cluster Resources

Cluster resources are saved by avoiding full pipeline recomputation

7️⃣ Parallel Task Scheduling

- DAG allows **tasks within a stage** to run **in parallel** across multiple executors.
- This ensures **maximum resource utilization** and **faster job completion**.



Enhancing Spark Performance with DAG

Faster Job Completion

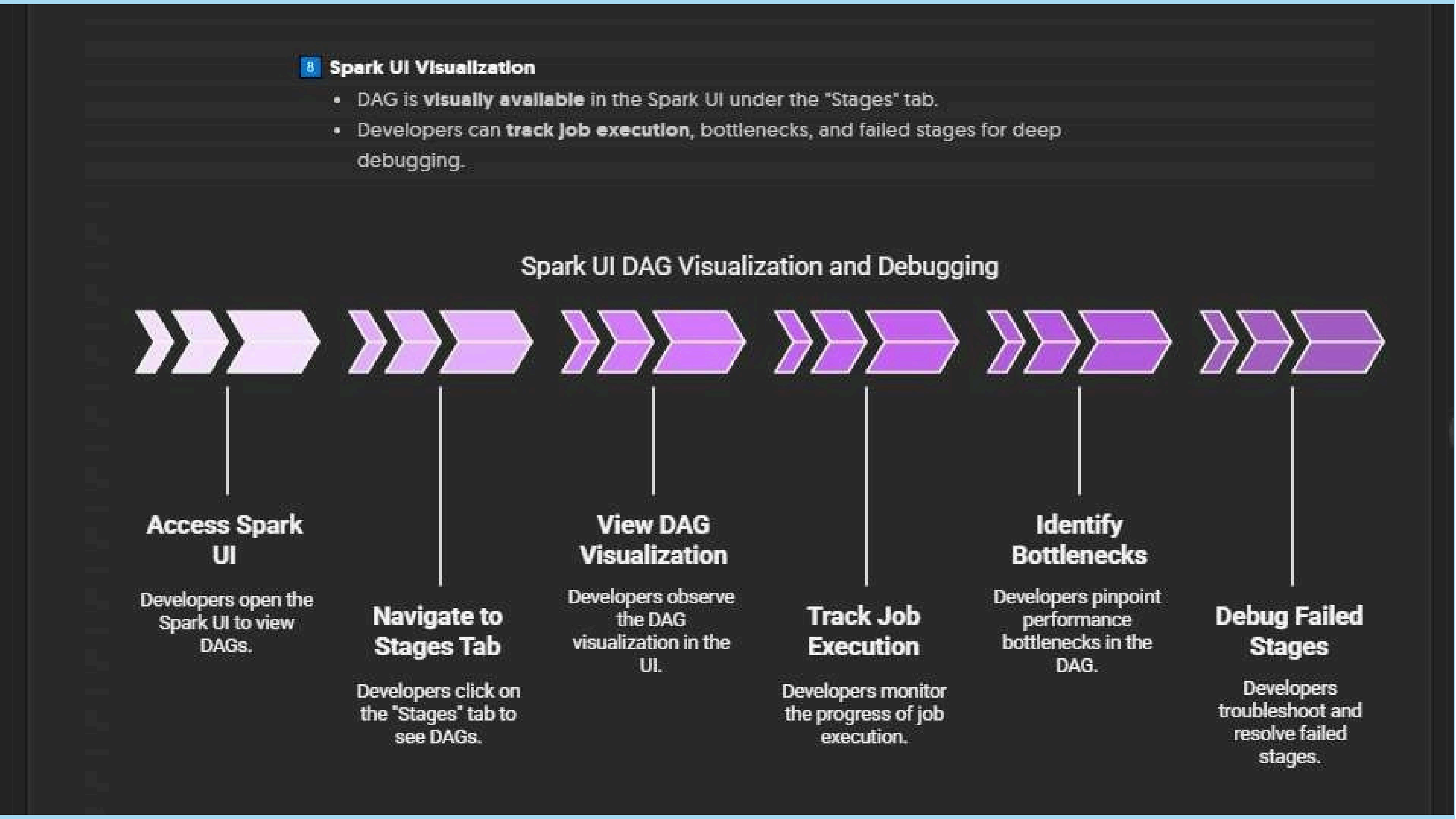
Reduces the time needed to complete jobs

Parallel Task Execution

Enables tasks to run simultaneously across executors

Maximum Resource Utilization

Ensures optimal use of available resources



8 **Spark UI Visualization**

- DAG is **visually available** in the Spark UI under the "Stages" tab.
- Developers can **track job execution**, bottlenecks, and failed stages for deep debugging.

Spark UI DAG Visualization and Debugging

Access Spark UI

Developers open the Spark UI to view DAGs.

Navigate to Stages Tab

Developers click on the "Stages" tab to see DAGs.

View DAG Visualization

Developers observe the DAG visualization in the UI.

Track Job Execution

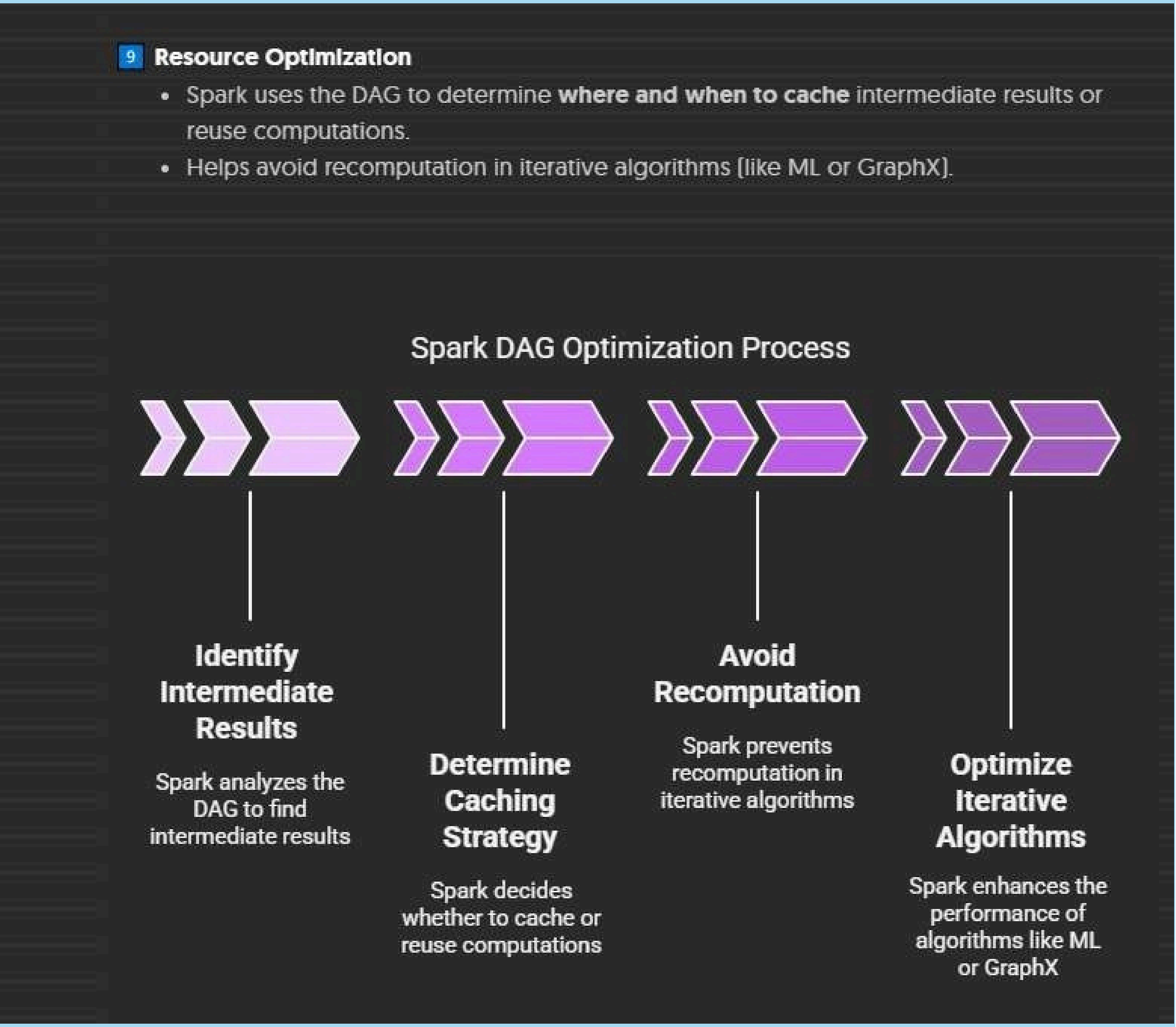
Developers monitor the progress of job execution.

Identify Bottlenecks

Developers pinpoint performance bottlenecks in the DAG.

Debug Failed Stages

Developers troubleshoot and resolve failed stages.



9 Resource Optimization

- Spark uses the DAG to determine **where and when to cache** intermediate results or reuse computations.
- Helps avoid recomputation in iterative algorithms (like ML or GraphX).

Spark DAG Optimization Process

Identify Intermediate Results

Spark analyzes the DAG to find intermediate results

Determine Caching Strategy

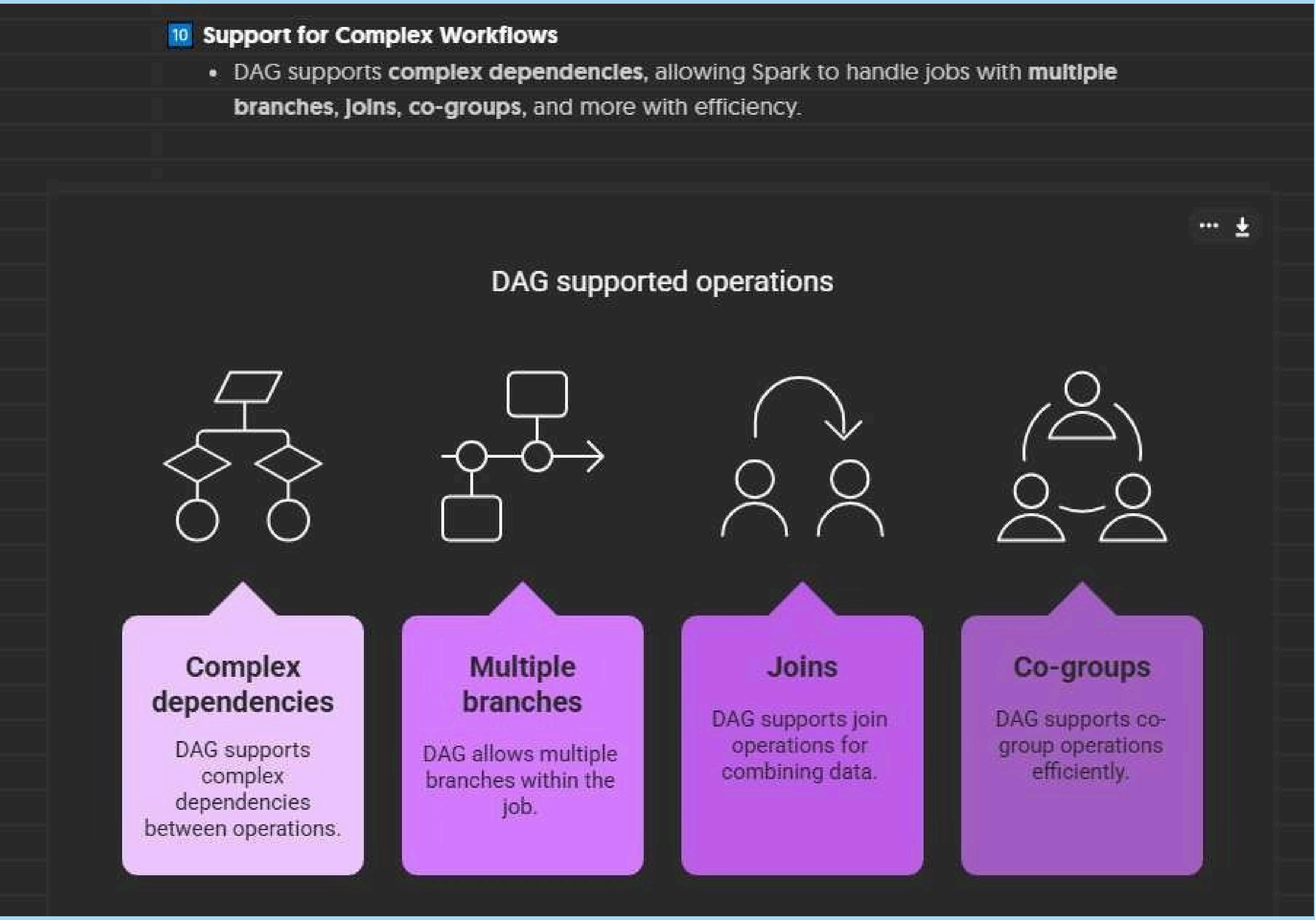
Spark decides whether to cache or reuse computations

Avoid Recomputation

Spark prevents recomputation in iterative algorithms

Optimize Iterative Algorithms

Spark enhances the performance of algorithms like ML or GraphX



🔟 **Support for Complex Workflows**

- DAG supports **complex dependencies,** allowing Spark to handle jobs with **multiple branches, joins, co-groups,** and more with efficiency.

DAG supported operations

Complex dependencies

DAG supports complex dependencies between operations.

Multiple branches

DAG allows multiple branches within the job.

Joins

DAG supports join operations for combining data.

Co-groups

DAG supports co-group operations efficiently.