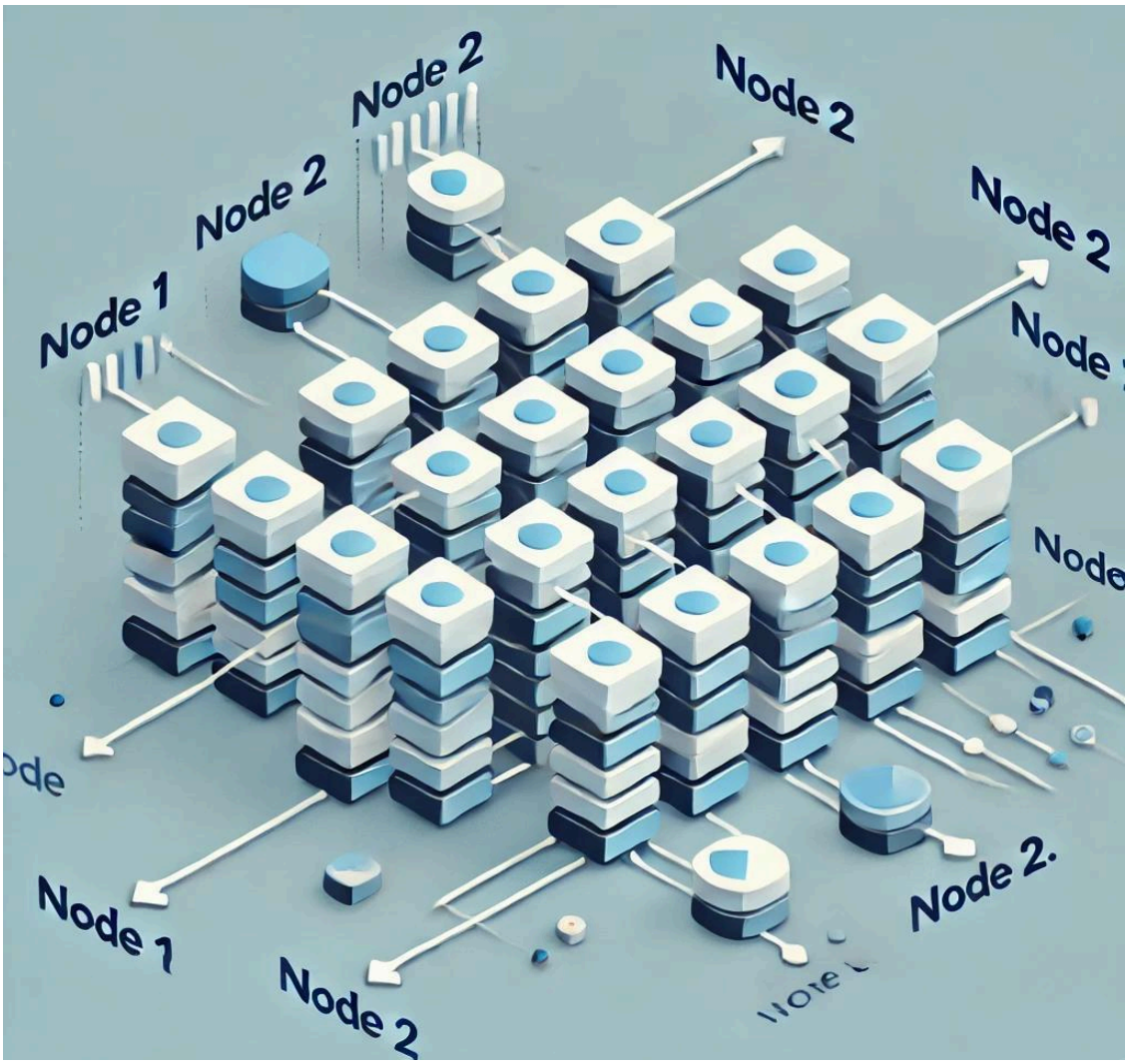
PySpark Intermediate Interview Questions and Experience-Based Questions

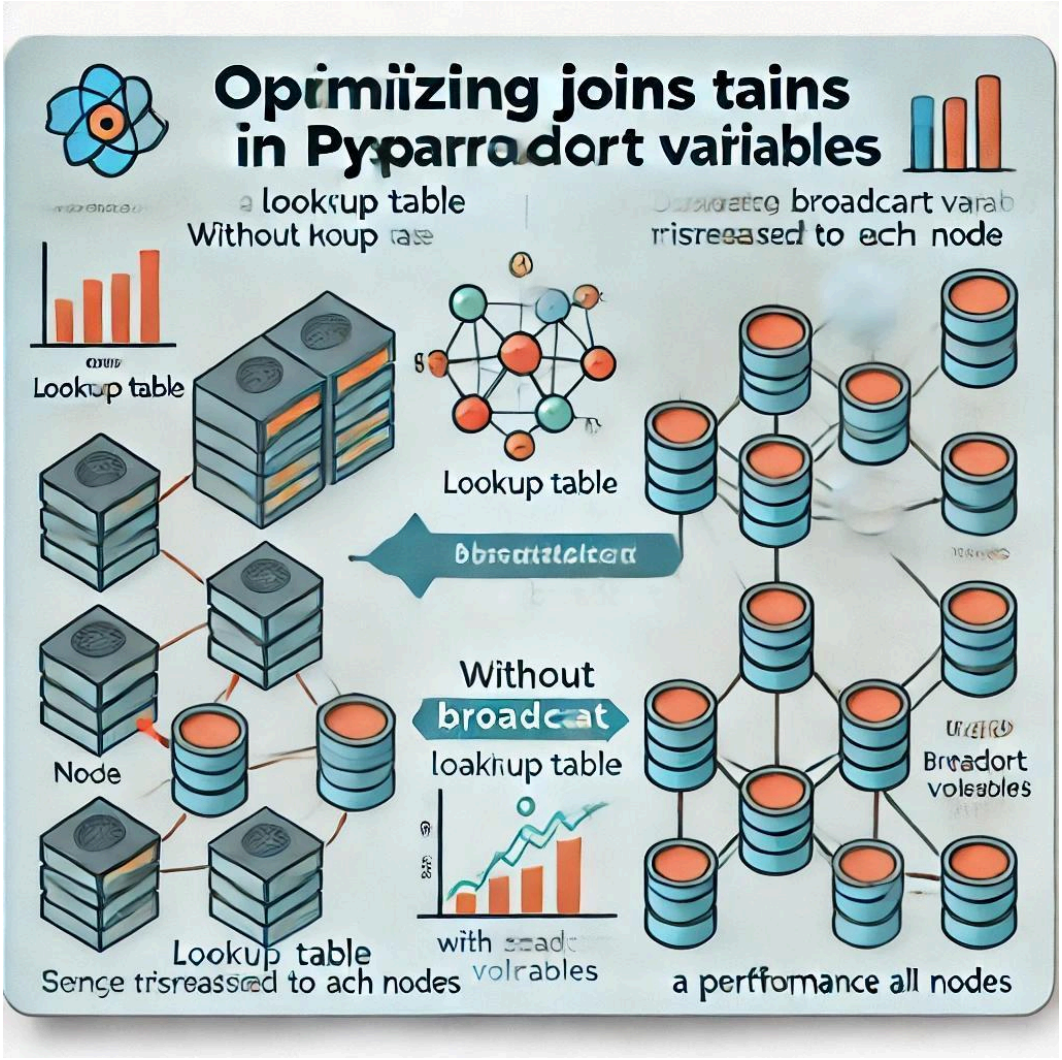
1. How does PySpark handle data distribution across a cluster?

Answer: PySpark distributes data across a cluster using RDDs or DataFrames. The data is split into partitions, each processed by a separate node, ensuring parallel processing. Here is a visual representation of how data distribution works in PySpark:



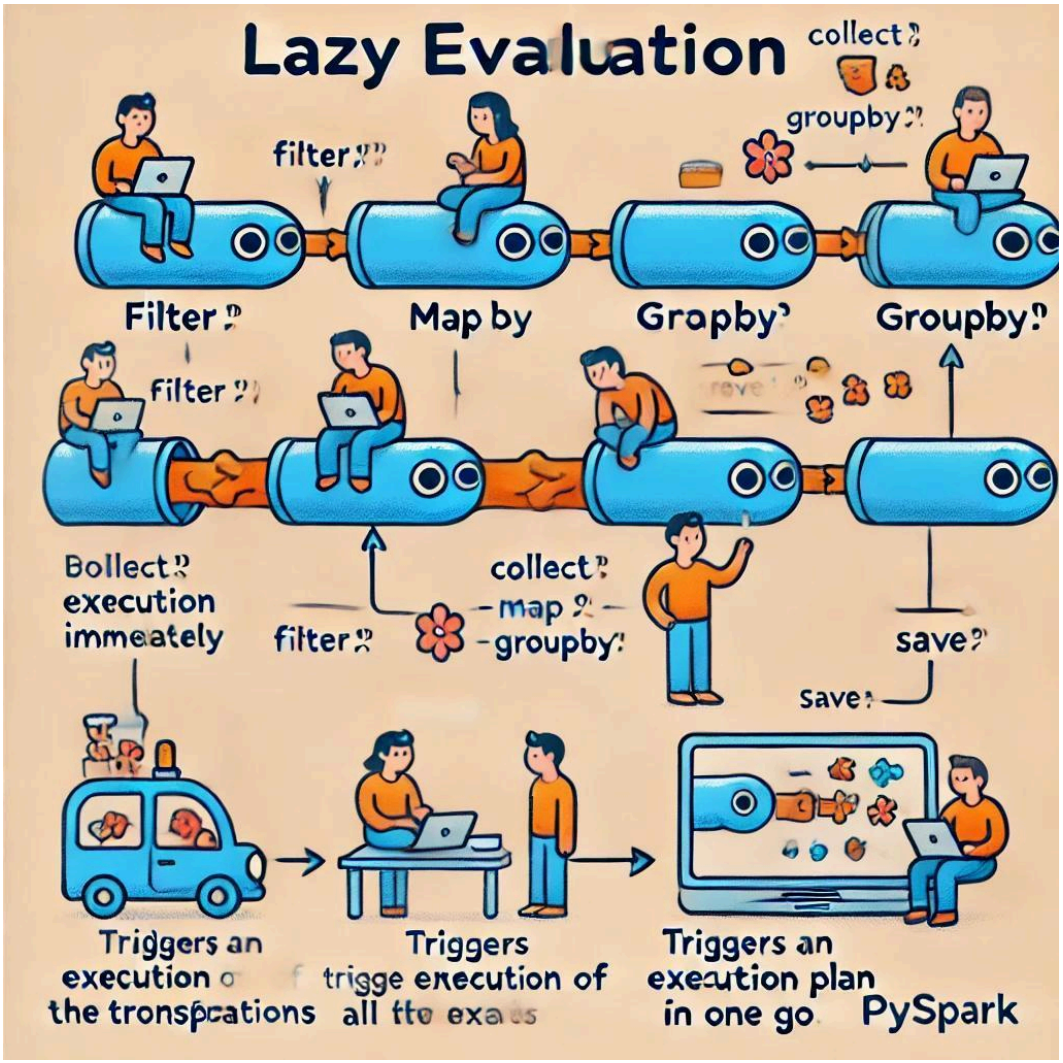
2. How do Broadcast Variables optimize performance in PySpark?

Answer: Broadcast variables distribute a copy of a lookup table to all nodes, reducing network traffic during join operations. The diagram below shows the difference between using a broadcast variable and not using one:



3. How does Lazy Evaluation work in PySpark, and why is it important?

Answer: PySpark uses lazy evaluation, where transformations are not executed until an action like collect() is called. The diagram below shows how transformations are delayed and executed only when an action is triggered:

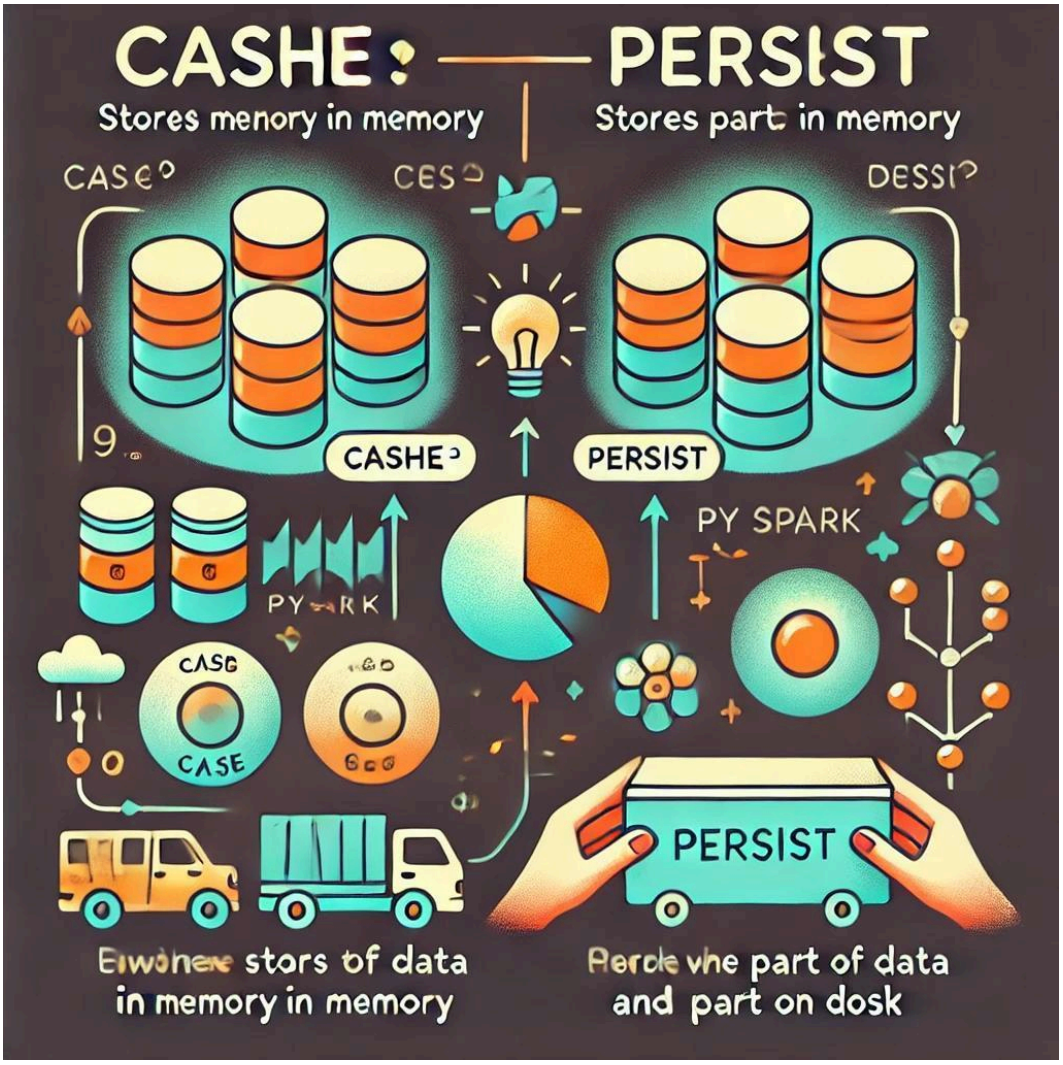


4. How does PySpark ensure Fault Tolerance?

Answer: PySpark provides fault tolerance through RDD lineage. Each RDD tracks the transformations used to build it. If a node fails during a job, PySpark recomputes only the lost partitions using the lineage, avoiding the need to rerun the entire job.

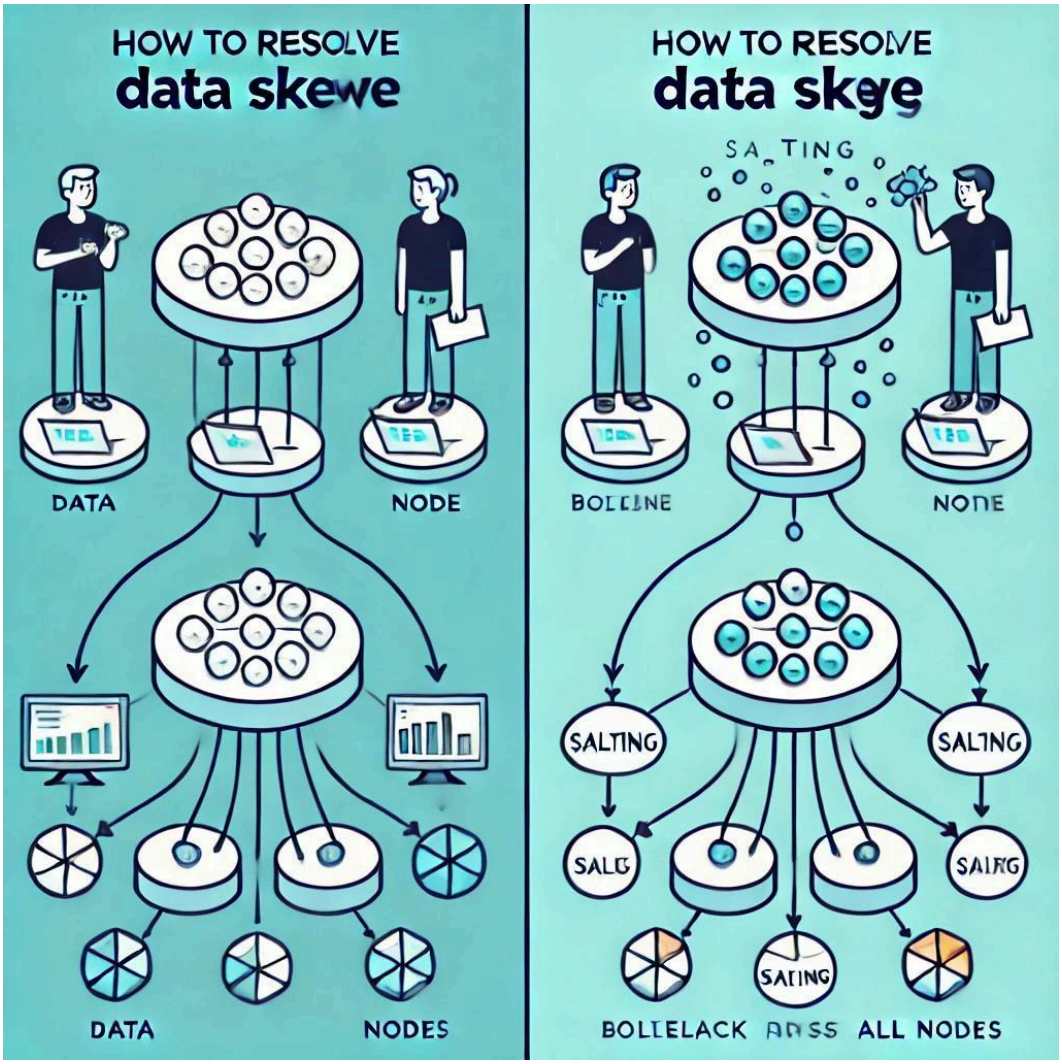
5. What is the difference between cache() and persist() in PySpark, and when would you use each?

Answer: cache() stores the RDD/DataFrame in memory for repeated use. It is best when the dataset fits in memory. persist() provides more flexibility, allowing the dataset to be stored both in memory and on disk (e.g., MEMORY_AND_DISK). Use persist() when the dataset is too large for memory.



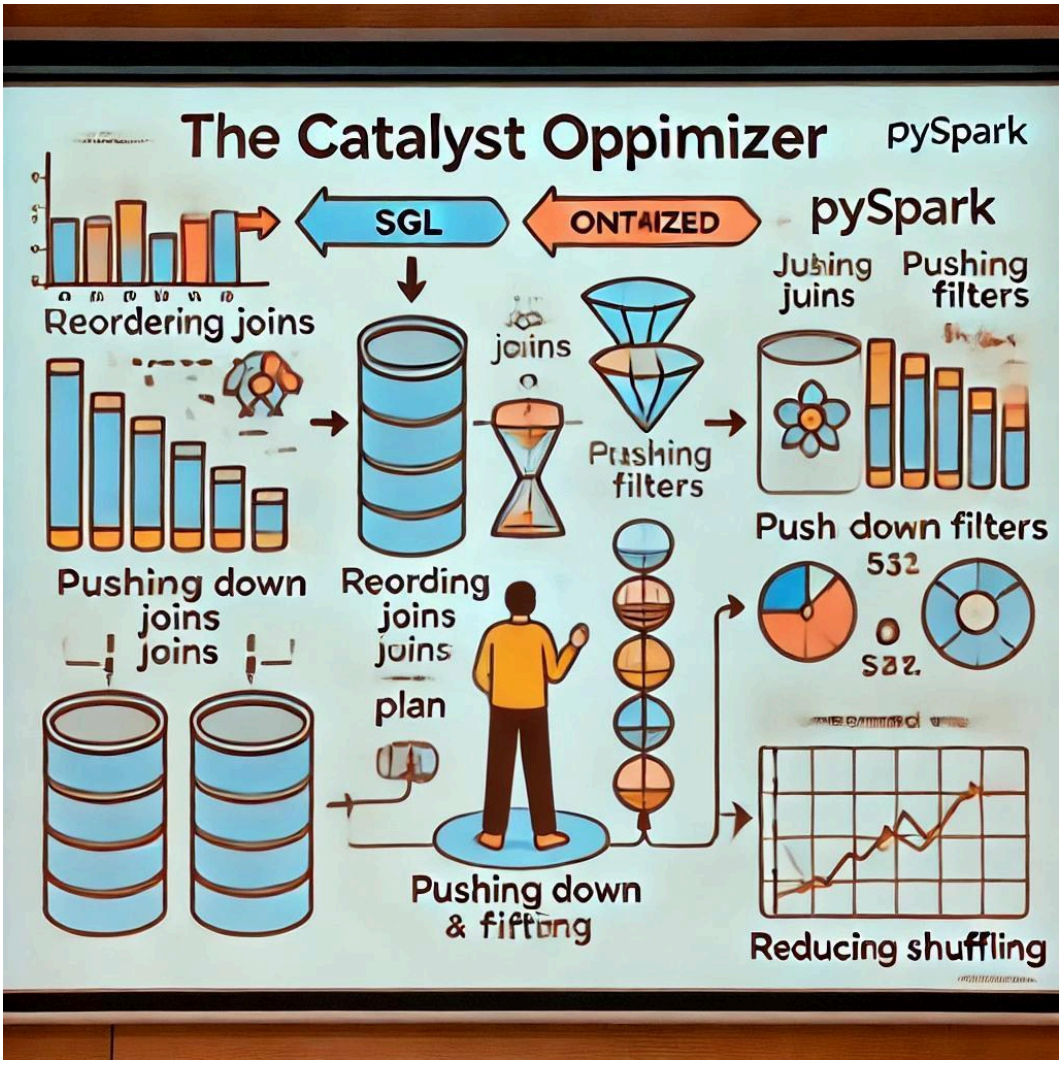
6. How do you handle Data Skew in PySpark?

Answer: Data skew happens when certain partitions contain more data than others, leading to bottlenecks. Salting is a technique where random values are added to skewed keys, distributing the data more evenly across nodes. The diagram below shows how salting helps resolve data skew:



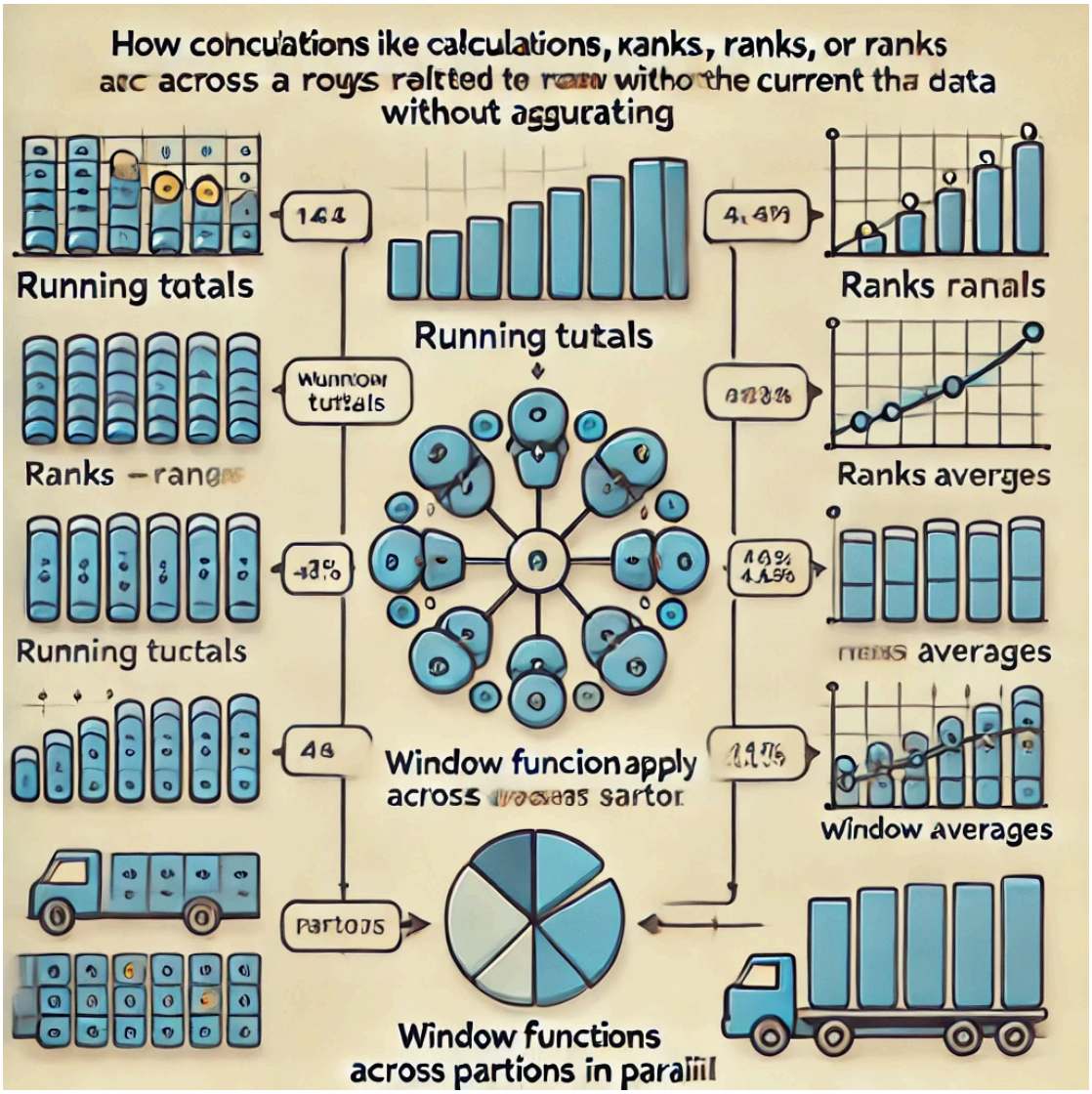
7. Explain how the Catalyst Optimizer improves performance in PySpark.

Answer: The Catalyst Optimizer analyzes the logical plan of a DataFrame or SQL query and applies optimization techniques such as join reordering and filter pushdown. This improves performance by reducing the amount of data shuffled across the cluster. The diagram below illustrates how an unoptimized query plan is transformed into an optimized one using the Catalyst Optimizer:



8. What are Window Functions in PySpark, and how are they used?

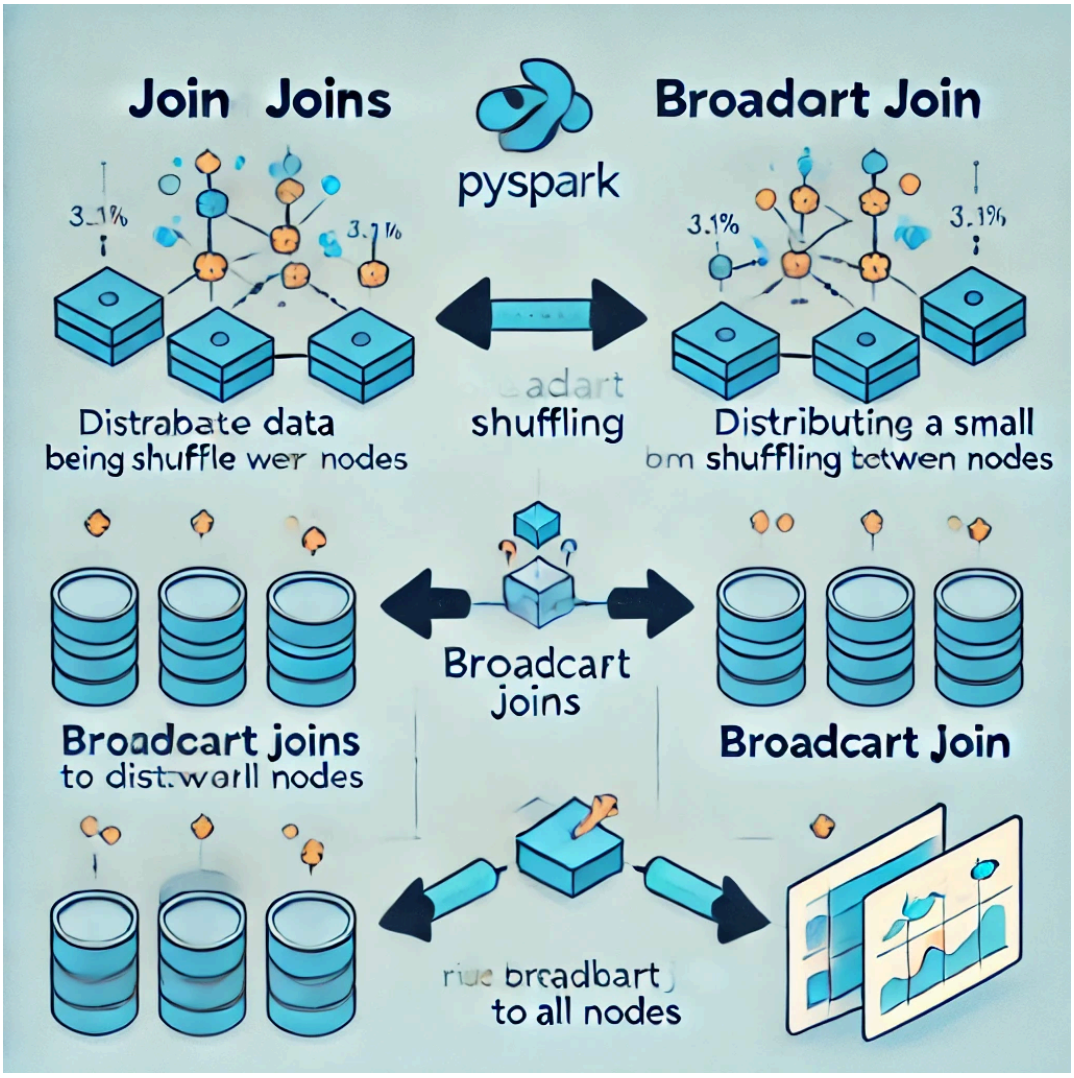
Answer: Window functions allow you to perform calculations across a set of rows related to the current row without aggregating the result. They are useful for tasks like calculating running totals, ranks, and moving averages.



9. How does PySpark handle Join operations, and how do you optimize them?

Answer: PySpark performs joins by redistributing data between partitions. Broadcast joins are useful when one of the datasets is small enough to be broadcasted to all worker nodes, reducing the amount of data shuffled across the

network.



10. Explain the concept of partitioning in PySpark and its significance.

Answer: Partitioning is the process of dividing data into smaller, more manageable chunks, allowing for parallel processing in PySpark. The number of

partitions should be configured based on the size of the dataset and the cluster resources to ensure efficient execution.