Repartition and Coalesce

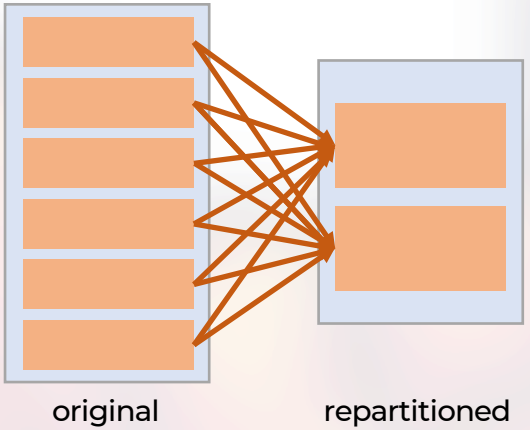
Objective

Change parallelism level of data processing

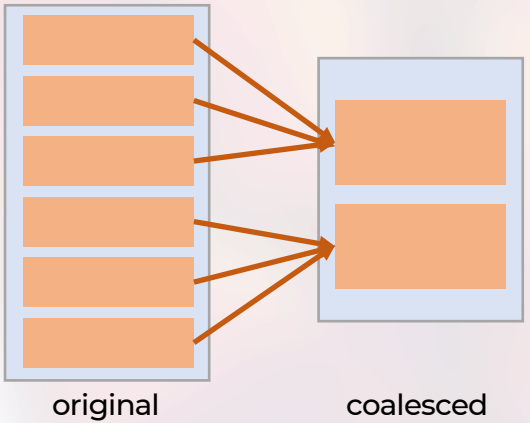
Repartition vs coalesce comparison

Repartition vs Coalesce

Repartition redistributes the data evenly across partitions



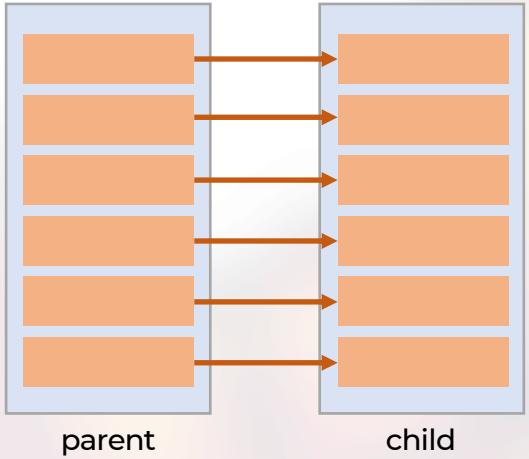
Coalesce "stitches" existing partitions



Dependencies

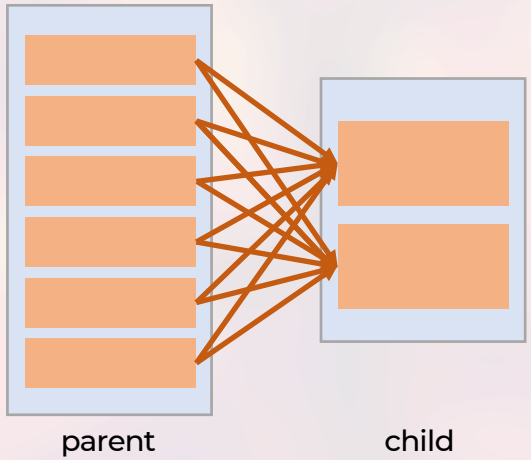
Narrow dependencies

- one input (parent) partition influences a single output (child) partition
- fast to compute
- examples: map, flatMap, filter, projections



Wide dependencies

- one input partition influences more than one output partitions
- involve a shuffle = data transfer between Spark executors
- are costly to compute
- examples: grouping, joining, sorting



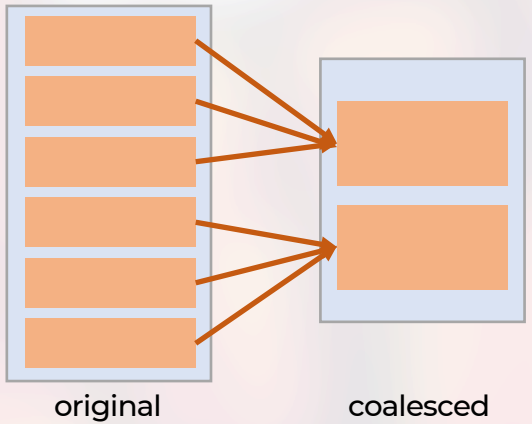
Coalesce

Coalesce is a narrow dependency

- one input (parent) partition influences a single output (child) partition

Coalesce will still move some data

- not a full shuffle
- almost always faster than a shuffle



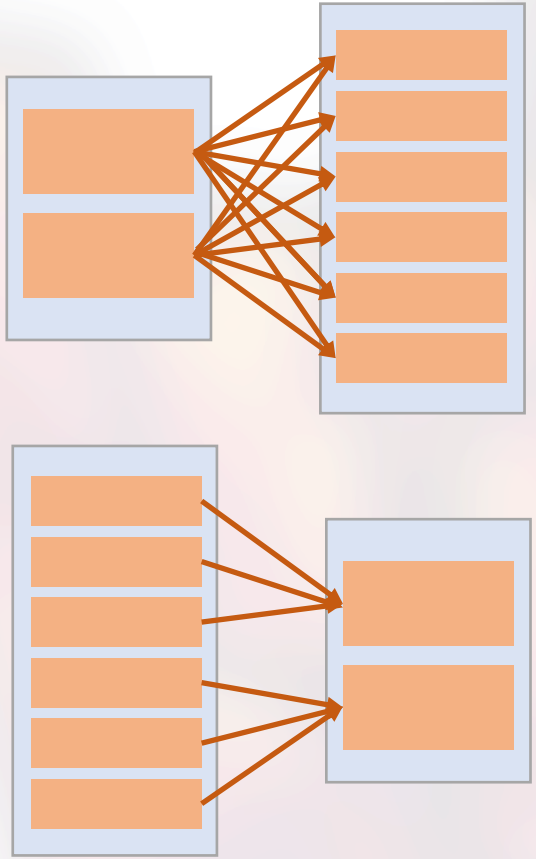
Repartition & Coalesce

Repartition

- returns a new RDD with the specified number of partitions
- will always involve a shuffle
- prioritizes even distribution of data
- necessary to control the number of partitions & partition size

Coalesce

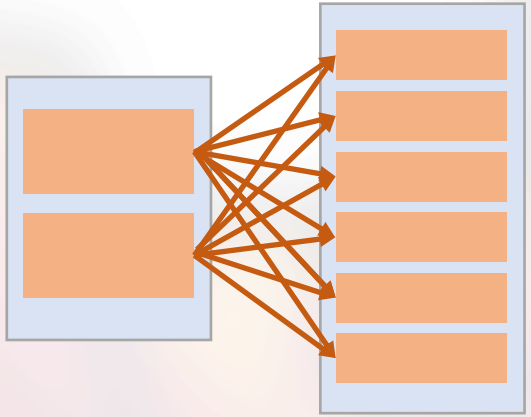
- returns a new RDD with the specified number of partitions
- used to decrease the number of partitions
- in this case
  - coalesce is a narrow transformation
  - cannot guarantee uniform data distribution
- can also be used to increase number of partitions
  - essentially a repartition



When to Use What

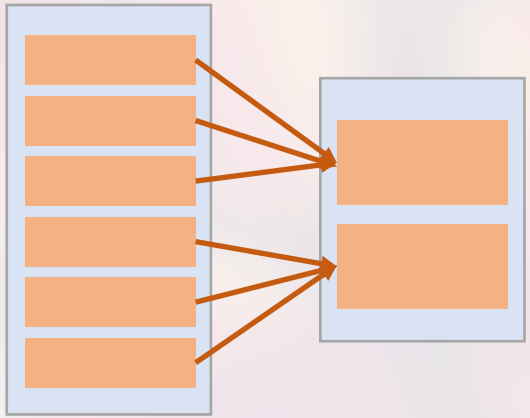
Use repartition when

- you want to increase parallelism/number of partitions
- you want to control partition size
- you want to redistribute the data evenly



Use coalesce when

- you want to reduce the number of partitions & improve perf
- you don't care how data is distributed



Spark rocks