Optimizing Apache Spark

The Five Most Common Performance Problems

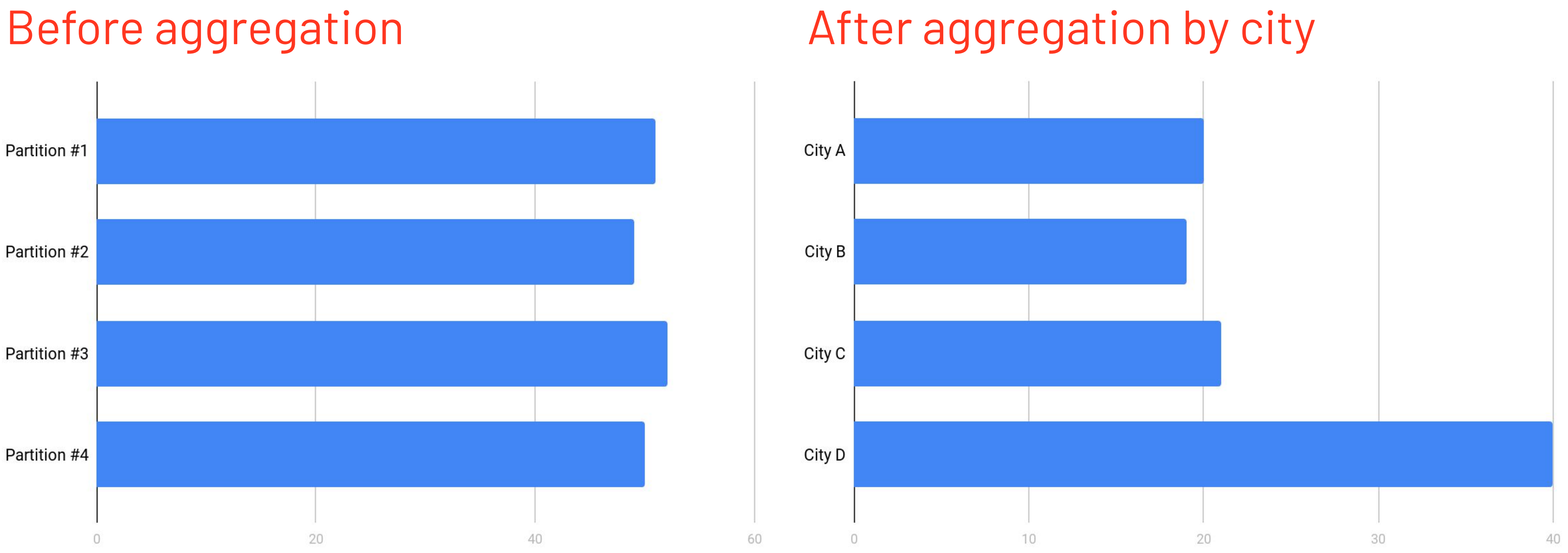
Skew

The 5 Most Common Performance Problems (The 5 Ss)

Skew

- Data is typically read in as 128 MB partitions and evenly distributed
- As the data is transformed (e.g. aggregated), it's possible to have significantly more records in one partition than another
- A small amount of skew is ignorable
- But large skews can result in spill or worse, hard to diagnose OOM Errors

The 5 Most Common Performance Problems (The 5 Ss)
Skew - Before & After

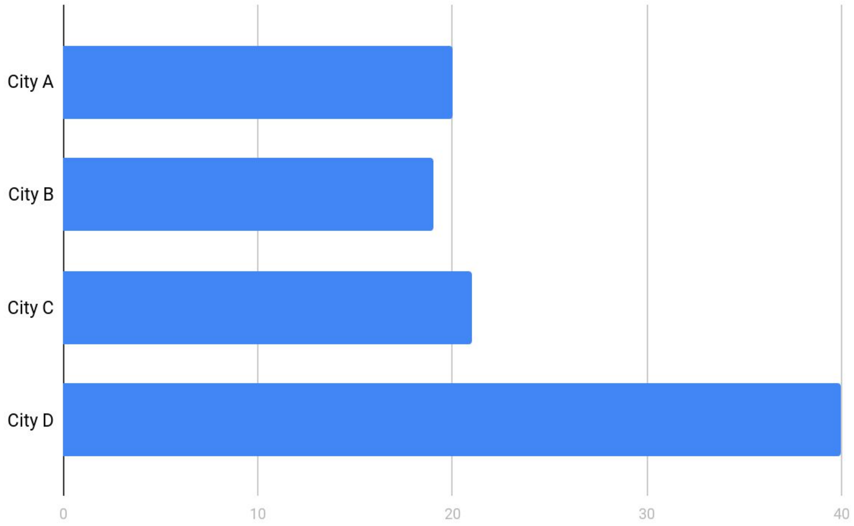


The 5 Most Common Performance Problems (The 5 Ss)
Skew - Ramifications

If **City D** is 2x larger than A, B or C...
- It takes 2x as long to process
- It requires 2x as much RAM

The ramifications of that is...
- The entire stage will take as long as the longest running task
- We may not have enough RAM for these skewed partitions

After aggregation by city



What can we do to mitigate skew?

The 5 Most Common Performance Problems (The 5 Ss)

Skew - Time vs RAM

We need to first ask which problem are we solving for?

- Solving for the RAM problem is only treating the symptoms and not the root cause.
- The RAM problem manifests itself as **Spill** and/or OOM Errors and should **not** be the first thing we solve for...*more on spill later*
- The first problem to solve for is the uneven distribution of records across all partitions which manifests itself as proportionally slower tasks

The 5 Most Common Performance Problems (The 5 Ss)

Skew - Mitigation

There are several strategies for fixing skew:

- Employ a Databricks-specific skew hint
  (*see [Skew Join optimization]*)

- Enable Adaptive Query Execution in Spark 3

- Salt the skewed column with a random number creating better distribution across each partition at the cost of extra processing

The 5 Most Common Performance Problems (The 5 Ss)

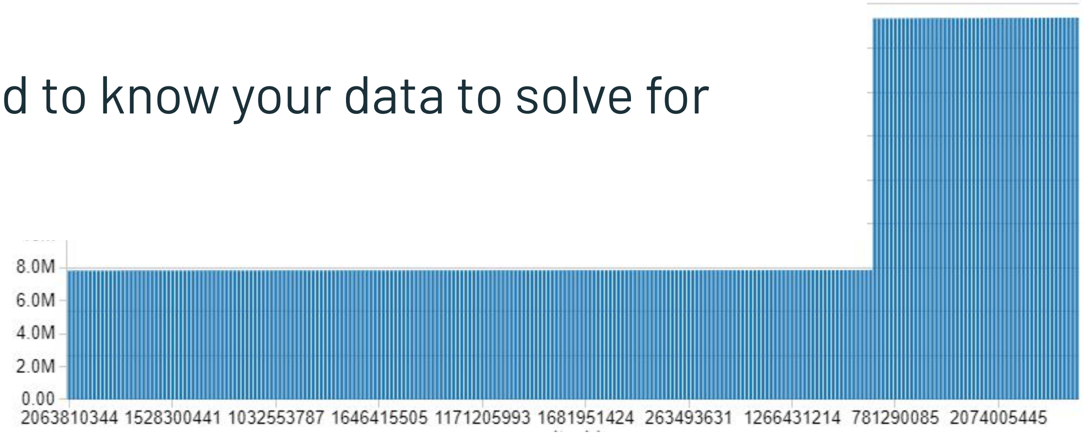
Skew - How Skewed?

See Experiment #1596, **Step B**

- Our perfectly engineered data has a skew in US cities that is ~3x larger than all other countries

- Counts come in at 23 million for skewed cities vs 8 million for other cities

- As we will see, you really need to know your data to solve for this...*maybe not with AQE*



The 5 Most Common Performance Problems (The 5 Ss)
Skew - Baseline vs Hint

See Experiment #1596, **Step C** and **Step D**

- Contrast the **last stage** of the **last job** for the two commands
  - Note the key code differences
  - Note the total execution time of the corresponding commands
  - Note the total number of tasks
  - In the **Spark UI**, **Stage Details**
    - Note the “health” of the stage as seen in the **Event Timeline**
    - Note the min/median/max **Shuffle Read Size** under **Summary Metrics**
    - Note the total amount of spill under **Aggregated Metrics by Executor**

The 5 Most Common Performance Problems (The 5 Ss)

Skew - Baseline vs Hint, Review

Step	Code	Duration	Tasks	Health	Shuffle	Spill
C	Standard	~30 min	832	Bad	0 / 0 / ~100 KB / ~400 MB / ~3 GB	~50 GB
D	Skew Hint	~35 min	832	Mostly OK	134 MB / 174 MB / 184 MB / 195 MB / 1.1 GB	~4 GB

- This scenario introduces the Databricks-specific skew hint (see Skew Join optimization)
- Note the call `.hint("skew", "city_id")`

Skew - Baseline vs Hint vs w/AQE

See Experiment #1596, **Step E** with **Step C** and **Step D**

- Contrast the **last stage** of the **last job** for the two commands
  - Note the key code differences
  - Note the total execution time of the corresponding commands
  - Note the total number of tasks
  - In the **Spark UI**, **Stage Details**
    - Note the "health" of the stage as seen in the **Event Timeline**
    - Note the min/median/max **Shuffle Read Size** under **Summary Metrics**
    - Note the total amount of spill under **Aggregated Metrics by Executor**

Skew - Baseline vs Hint, Review

Step	Code	Duration	Tasks	Health	Shuffle	Spill
C	Standard	~30 min	832	Bad	0 / 0 / ~100 KB / ~400 MB / ~3 GB	~50 GB
D	Skew Hint	~35 min	832	Mostly OK	~135 MB / ~175 MB / ~180 MB / ~200 MB / ~1 GB	~4 GB
E	w/AQE	~25 min	1489	Excellent	0 / ~115 MB / ~115 MB / ~125 MB / ~130 MB	0

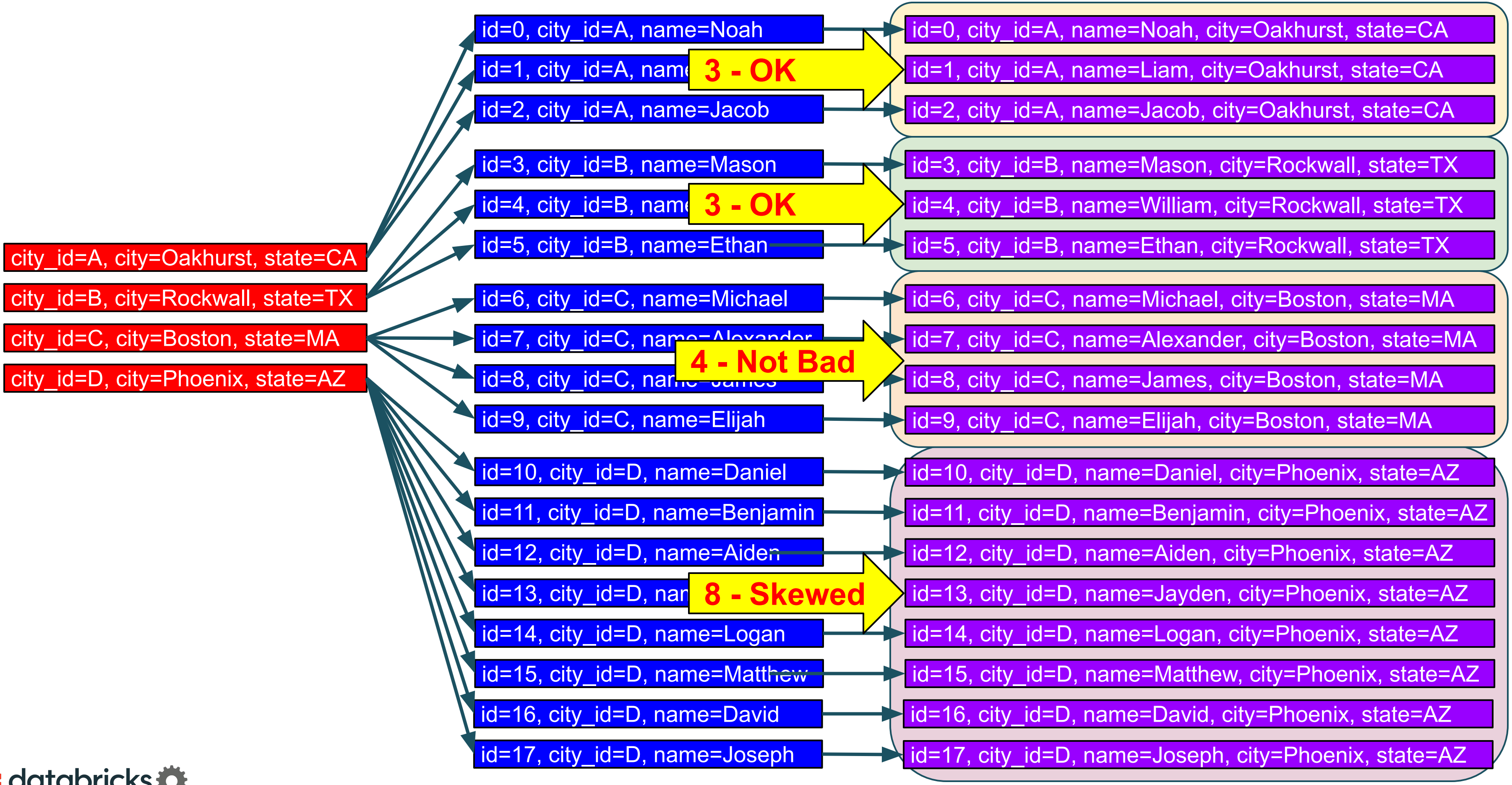
- **Step E** uses **Spark 3's** new feature **Adaptive Skewed Join**
  - See **spark.sql.adaptive.skewJoin.enabled**
  - See **spark.sql.adaptive.advisoryPartitionSizeInBytes**
- The first two **jobs** are read in parallel

Skew - Salted Join

- This approach is by far the most complicated to implement
- It can actually take longer to execute in some cases
- Remains a viable option when other solutions are not available
- The idea is to split large partitions into smaller ones using a “salt”
- Has the side effect of splitting small partitions into even smaller ones
- It’s more about guaranteeing execution of all tasks
  And **not** a uniform duration for each task

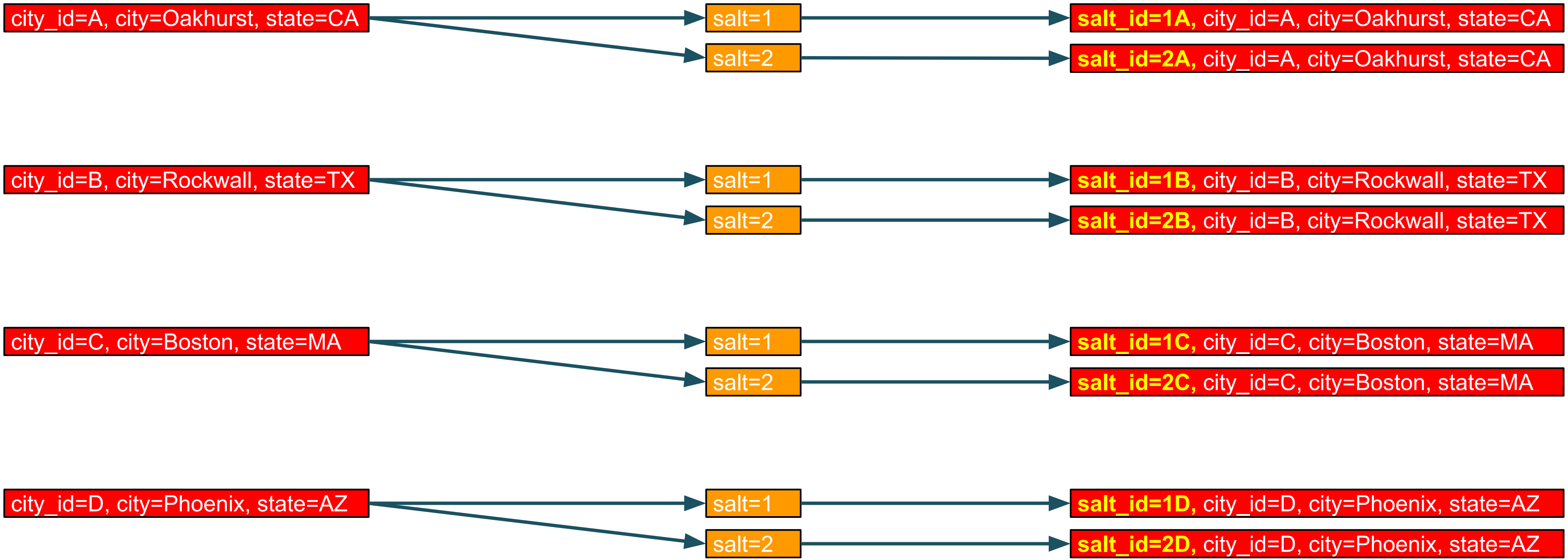
Let’s review how a “standard” join works...

…4 distinct partitions



Let’s review how a “salted” join works...

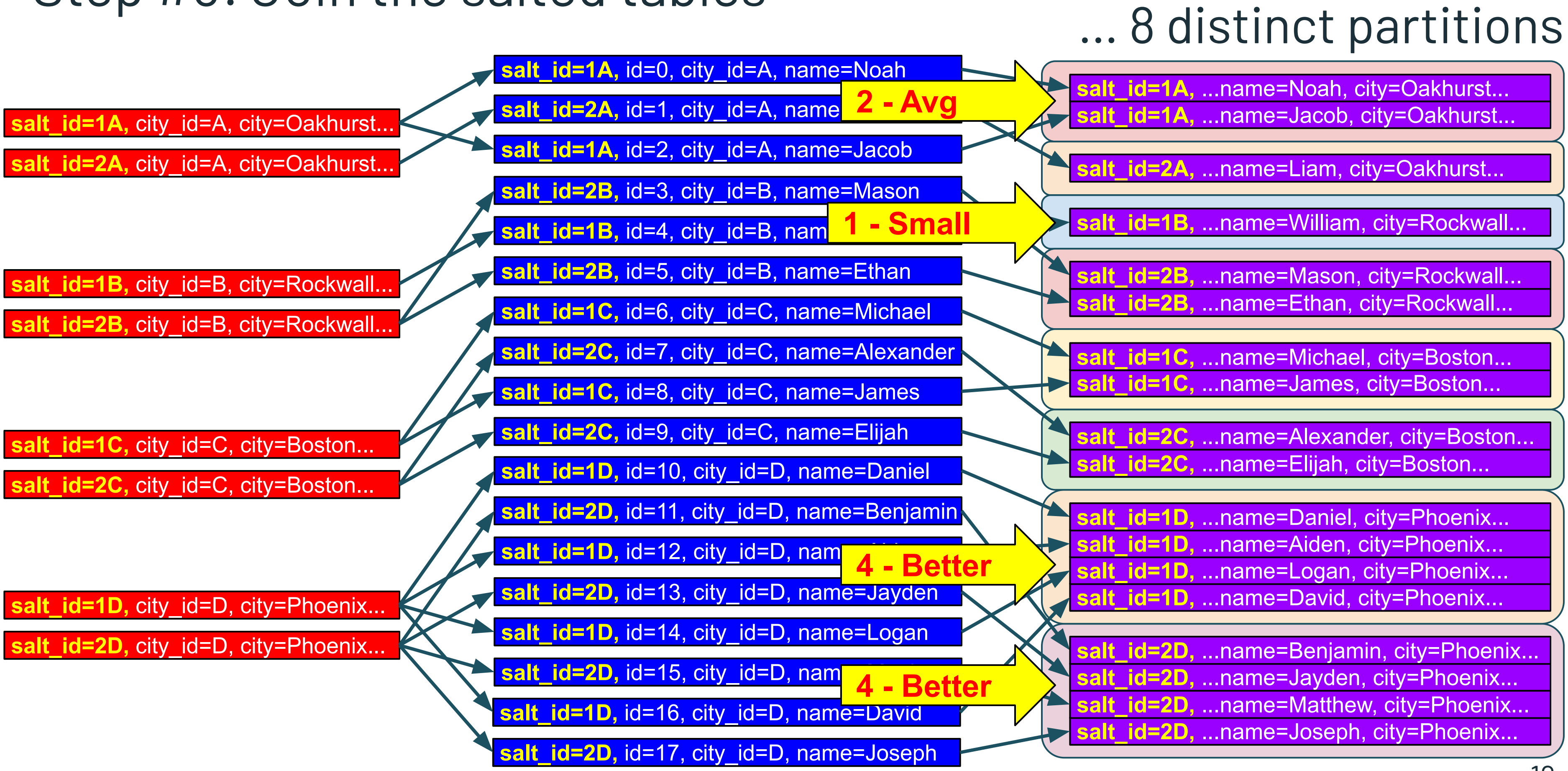
Step #1: Cross join the dimensions to the salt value



Step #2: Randomly salt the fact table

id=0, city_id=A, name=Noah	salt=rand(2)	1	**salt_id=1A,** id=0, city_id=A, name=Noah
id=1, city_id=A, name=Liam	salt=rand(2)	2	**salt_id=2A,** id=1, city_id=A, name=Liam
id=2, city_id=A, name=Jacob	salt=rand(2)	1	**salt_id=1A,** id=2, city_id=A, name=Jacob
id=3, city_id=B, name=Mason	salt=rand(2)	2	**salt_id=2B,** id=3, city_id=B, name=Mason
id=4, city_id=B, name=William	salt=rand(2)	1	**salt_id=1B,** id=4, city_id=B, name=William
id=5, city_id=B, name=Ethan	salt=rand(2)	2	**salt_id=2B,** id=5, city_id=B, name=Ethan
id=6, city_id=C, name=Michael	salt=rand(2)	1	**salt_id=1C,** id=6, city_id=C, name=Michael
id=7, city_id=C, name=Alexander	salt=rand(2)	2	**salt_id=2C,** id=7, city_id=C, name=Alexander
id=8, city_id=C, name=James	salt=rand(2)	1	**salt_id=1C,** id=8, city_id=C, name=James
id=9, city_id=C, name=Elijah	salt=rand(2)	2	**salt_id=2C,** id=9, city_id=C, name=Elijah
id=10, city_id=D, name=Daniel	salt=rand(2)	1	**salt_id=1D,** id=10, city_id=D, name=Daniel
id=11, city_id=D, name=Benjamin	salt=rand(2)	2	**salt_id=2D,** id=11, city_id=D, name=Benjamin
id=12, city_id=D, name=Aiden	salt=rand(2)	1	**salt_id=1D,** id=12, city_id=D, name=Aiden
id=13, city_id=D, name=Jayden	salt=rand(2)	2	**salt_id=2D,** id=13, city_id=D, name=Jayden
id=14, city_id=D, name=Logan	salt=rand(2)	1	**salt_id=1D,** id=14, city_id=D, name=Logan
id=15, city_id=D, name=Matthew	salt=rand(2)	2	**salt_id=2D,** id=15, city_id=D, name=Matthew
id=16, city_id=D, name=David	salt=rand(2)	1	**salt_id=1D,** id=16, city_id=D, name=David
id=17, city_id=D, name=Joseph	salt=rand(2)	2	**salt_id=2D,** id=17, city_id=D, name=Joseph

Step #3: Join the salted tables



Skew - Skew Join, in Action

- Step F-1: Create a DataFrame based on the range of our "skew factor"
  - In the visual example, we used "2"
  - In this code example, we are using "7"
  - You can estimate this based on how many times larger the maximum partition is compared to the median partition size

- Step F-2: For the dimension table, cross join the salts with the city table (repartitioning can help mitigate spills and evenly redistributes the new dimension table across all partitions)

- Step F-3: For the fact table, randomly assign a salt to each record

- Step F-4: Join the two tables based on the **`salted_city_id`**

The 5 Most Common Performance Problems (The 5 Ss)

Skew - Baseline vs Hint vs w/AQE vs Salted

See Experiment #1596, **Step F-4** with **Step C** through **Step E**

- Contrast the **last stage** of the **last job** for the two commands
  - Note the key code differences
  - Note the total execution time of the corresponding commands
  - Note the total number of tasks
  - In the **Spark UI**, **Stage Details**
    - Note the "health" of the stage as seen in the **Event Timeline**
    - Note the min/median/max **Shuffle Read Size** under **Summary Metrics**
    - Note the total amount of spill under **Aggregated Metrics by Executor**

The 5 Most Common Performance Problems (The 5 Ss)

Skew - Baseline vs Hint, Review

Step	Code	Duration	Tasks	Health	Shuffle	Spill
C	Standard	~30 min	832	Bad	0 / 0 / ~100 KB / ~400 MB / ~3 GB	~50 GB
D	Skew Hint	~35 min	832	Mostly OK	~135 MB / ~175 MB / ~180 MB / ~200 MB / ~1 GB	~4 GB
E	w/AQE	~25 min	1489	Excellent	0 / ~115 MB / ~115 MB / ~125 MB / ~130 MB	0
F	Salted	>35 min	832	Better/Bad	~400 KB / ~75 MB / ~150 MB / ~300 MB / ~800 MB	0

- Salting a skewed dataset has a number of problems
- You don't want to salt on the fly - it should be a persisted view of the data
- Consider instead, invest the energy to salt only the skewed keys
- In our example, that would mean salting US cities only