TWO SIGMA

www.twosigma.com

Huohua 火花
Distributed Time Series Analysis Framework For Spark

Wenbo Zhao

Spark Summit 2016

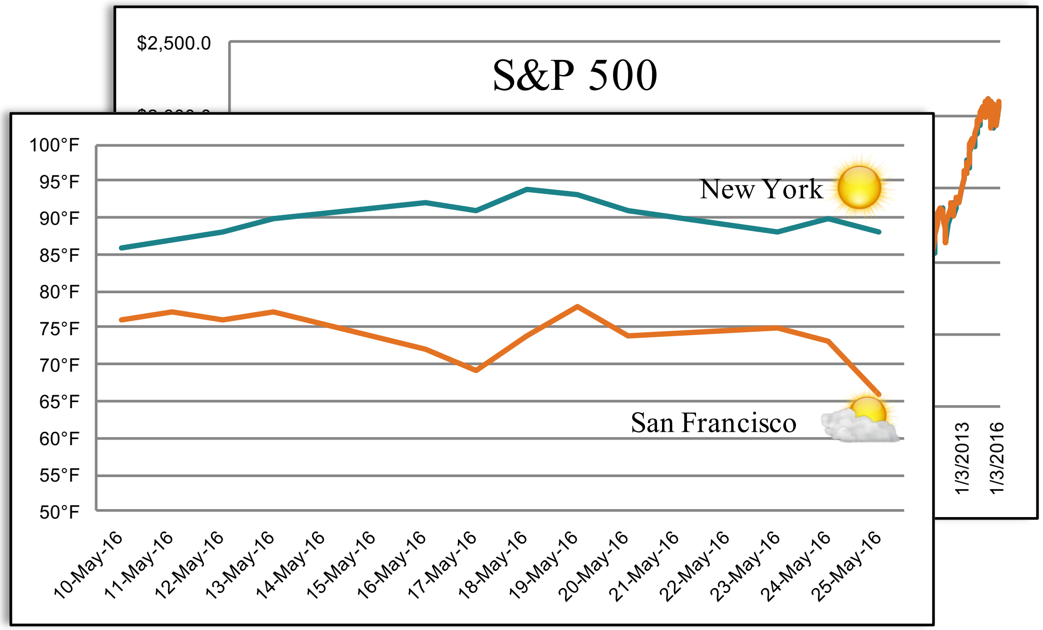
June 10, 2016

About Me

- Software Engineer @ TWO SIGMA
- Focus on analytics related tools, libraries and Systems

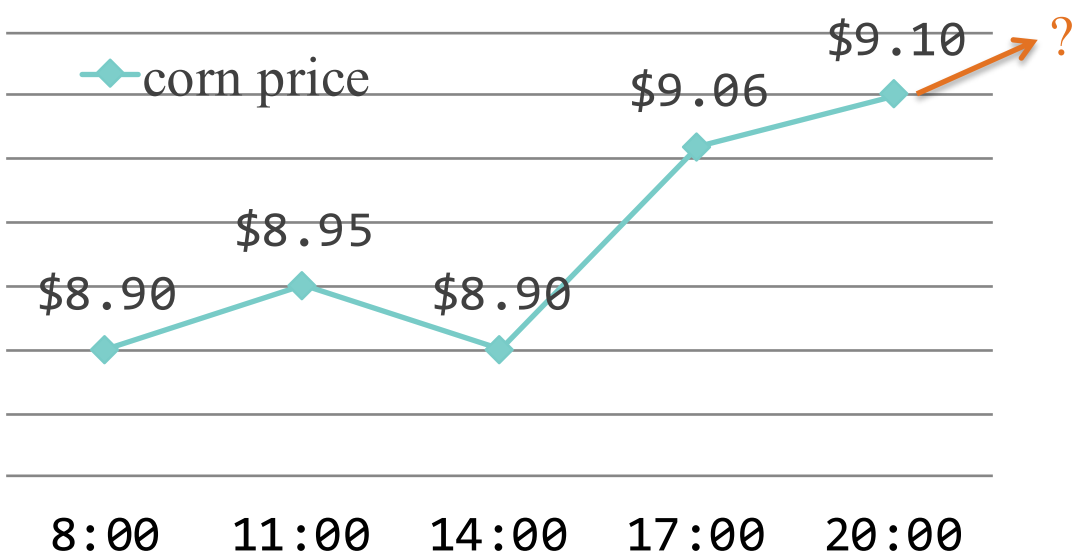
We view everything as a time series

- Stock market prices
- Temperatures
- Sensor logs
- Presidential polls
- …



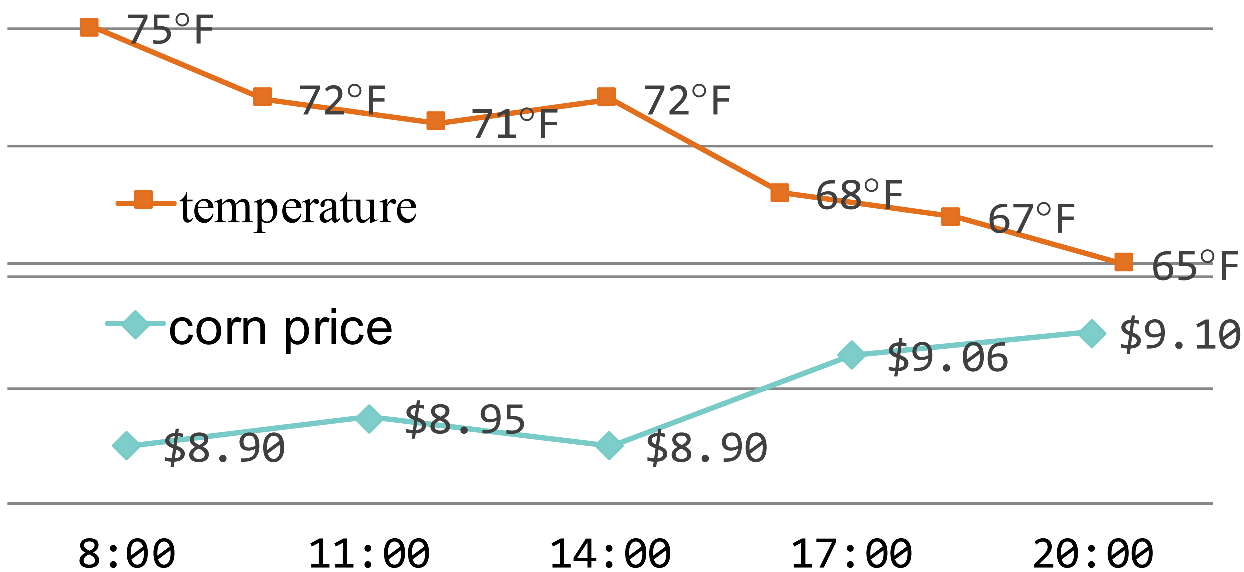
What is a time series?

- A sequence of observations obtained in successive time order
- Our goal is to forecast future values given past observations



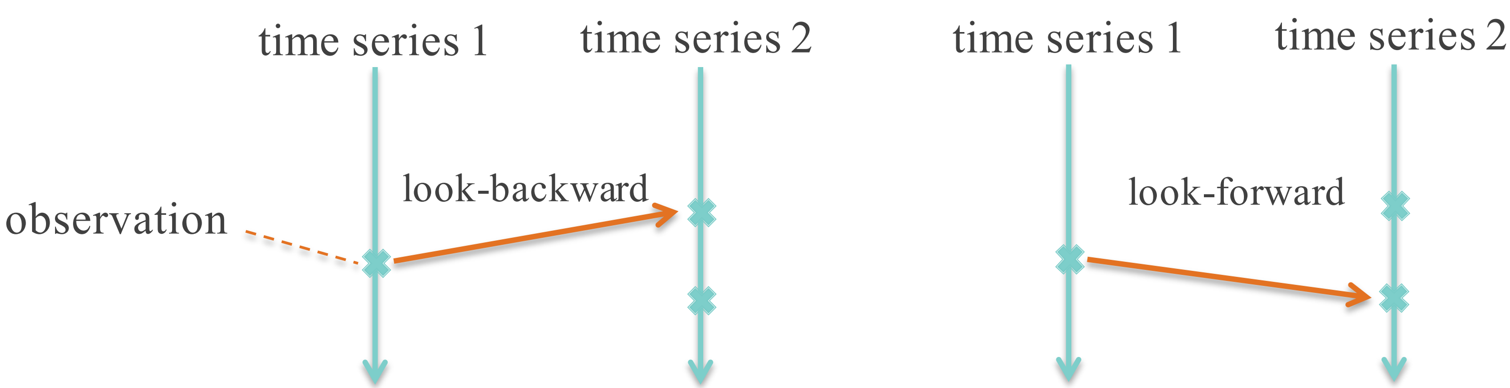
Multivariate time series

- We can forecast better by joining multiple time series
- *Temporal join* is a fundamental operation for time series analysis
- *Huohua* enables fast distributed temporal join of large scale *unaligned* time series



What is temporal join?

- A particular *join* function defined by a matching criteria over *time*
- Examples of criteria
  - *look-backward* – find the most recent observation in the past
  - *look-forward* – find the closest observation in the future



Temporal join with look-backward criteria

time	weather
08:00 AM	60 °F
10:00 AM	70 °F
12:00 AM	80 °F

time	corn price
08:00 AM	
11:00 AM	

time	weather	corn price
08:00 AM		
10:00 AM		
12:00 AM		

Temporal join with look-backward criteria

time	weather
08:00 AM	60 °F
10:00 AM	70 °F
12:00 AM	80 °F

time	corn price
08:00 AM	
11:00 AM	

time	weather	corn price
08:00 AM	60 °F	
10:00 AM		
12:00 AM		

Temporal join with look-backward criteria

time	weather
08:00 AM	60 °F
10:00 AM	70 °F
12:00 AM	80 °F

time	corn price
08:00 AM	$ $
11:00 AM	$ $ $

time	weather	corn price
08:00 AM	60 °F	$ $
10:00 AM	70 °F	$ $
12:00 AM		

Temporal join with look-backward criteria

time	weather
08:00 AM	60 °F
10:00 AM	70 °F
12:00 AM	80 °F

time	corn price
08:00 AM	$ $
11:00 AM	$ $ $

time	weather	corn price
08:00 AM	60 °F	$ $
10:00 AM	70 °F	$ $
12:00 AM	80 °F	$ $ $

Temporal join with look-backward criteria

time	weather
08:00 AM	60 °F
10:00 AM	70 °F
12:00 AM	80 °F

...

time	corn price
08:00 AM	
11:00 AM	

...

Hundreds of thousands of data sources
with unaligned timestamps

time	weather	corn price
08:00 AM	60 °F	
10:00 AM	70 °F	
12:00 AM	80 °F	

Thousands of market data sets

We need fast and scalable distributed temporal join

Issues with existing solutions

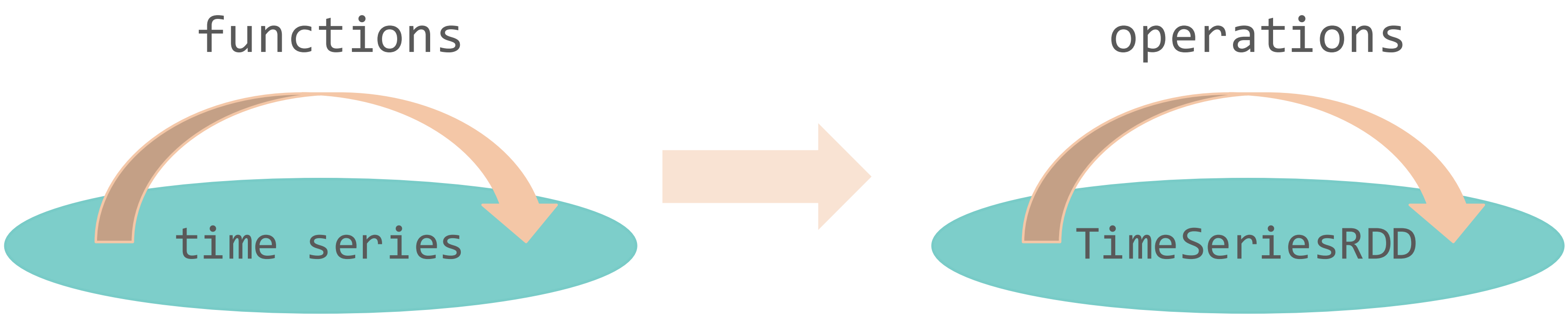
- A single time series may not fit into a single machine
- Forecasting may involve hundreds of time series
- Existing packages don’t support temporal join or can’t handle large time series
  - MatLab, R, SAS, Pandas
  - Even Spark based solutions fall short
    - PairRDDFunctions, DataFrame/Dataset, spark-ts

Huohua – a new time series library for Spark

- Goal
  - provide a collection of functions to *manipulate* and *analyze* time series at scale
    - group, temporal join, summarize, aggregate …
- How
  - build a time series aware data structure
    - extending `RDD` to `TimeSeriesRDD`
  - optimize using temporal locality
    - reduce shuffling
    - reduce memory pressure by streaming

What is a TimeSeriesRDD in Huohua?

- `TimeSeriesRDD` extends `RDD` to represent time series data
  - associates a time range to each partition
  - tracks partitions' time-ranges through operations
  - preserves the temporal order



TimeSeriesRDD– an RDD representing time series

time	temperature
6:00 AM	60°F
6:01 AM	61°F
…	…
7:00 AM	70°F
7:01 AM	71°F
…	…
8:00 AM	80°F
8:01 AM	81°F
…	…

RDD

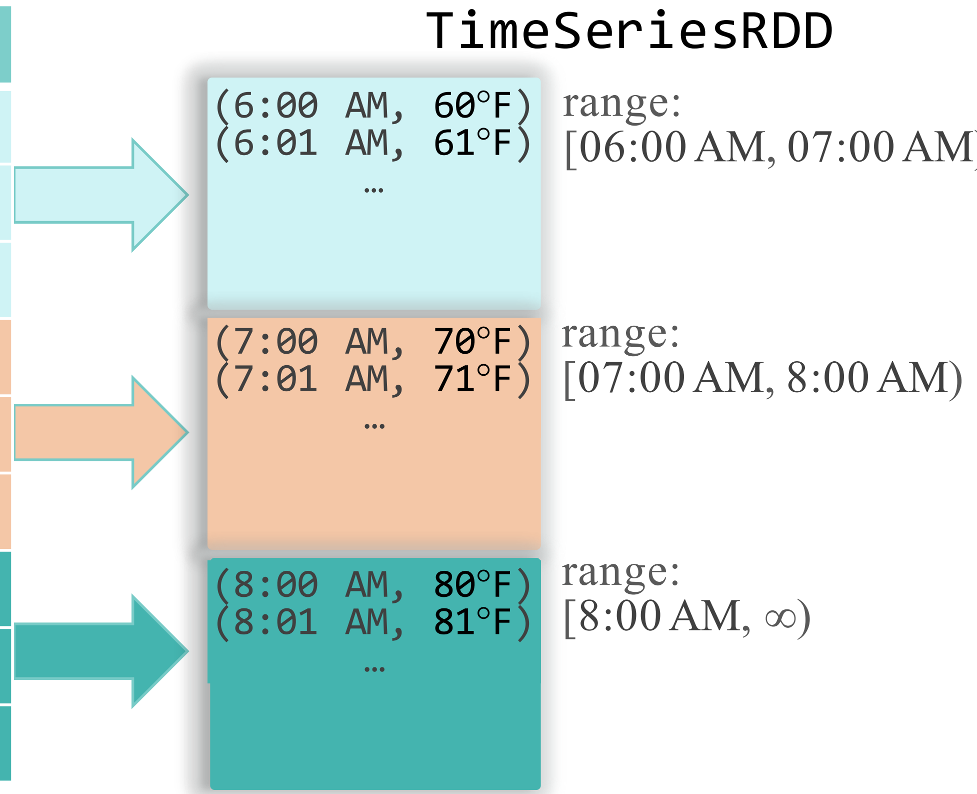
```
(6:00 AM, 60°F)
(6:01 AM, 61°F)
…
```

```
(7:00 AM, 70°F)
(7:01 AM, 71°F)
…
```

```
(8:00 AM, 80°F)
(8:01 AM, 81°F)
…
```

TimeSeriesRDD– an RDD representing time series

time	temperature
6:00 AM	60°F
6:01 AM	61°F
…	…
7:00 AM	70°F
7:01 AM	71°F
…	…
8:00 AM	80°F
8:01 AM	81°F
…	…



Group function

- A *group* function groups rows with exactly the same timestamps

time	city	temperature
1:00 PM	New York	70°F
1:00 PM	San Francisco	60°F
2:00 PM	New York	71°F
2:00 PM	San Francisco	61°F
3:00 PM	New York	72°F
3:00 PM	San Francisco	62°F
4:00 PM	New York	73°F
4:00 PM	San Francisco	63°F

group 1

group 2

group 3

group 4

Group function

- A *group* function groups rows with nearby timestamps

time	city	temperature
1:00 PM	New York	70°F
1:00 PM	San Francisco	60°F
2:00 PM	New York	71°F
2:00 PM	San Francisco	61°F
3:00 PM	New York	72°F
3:00 PM	San Francisco	62°F
4:00 PM	New York	73°F
4:00 PM	San Francisco	63°F

group 1

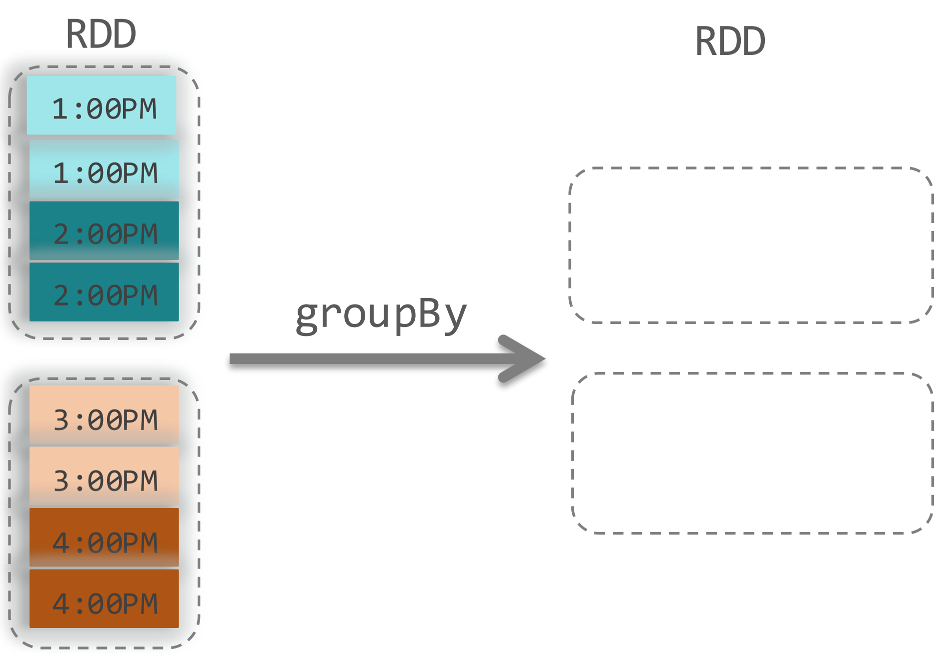
group 2

Group in Spark

- Groups rows with exactly the same timestamps

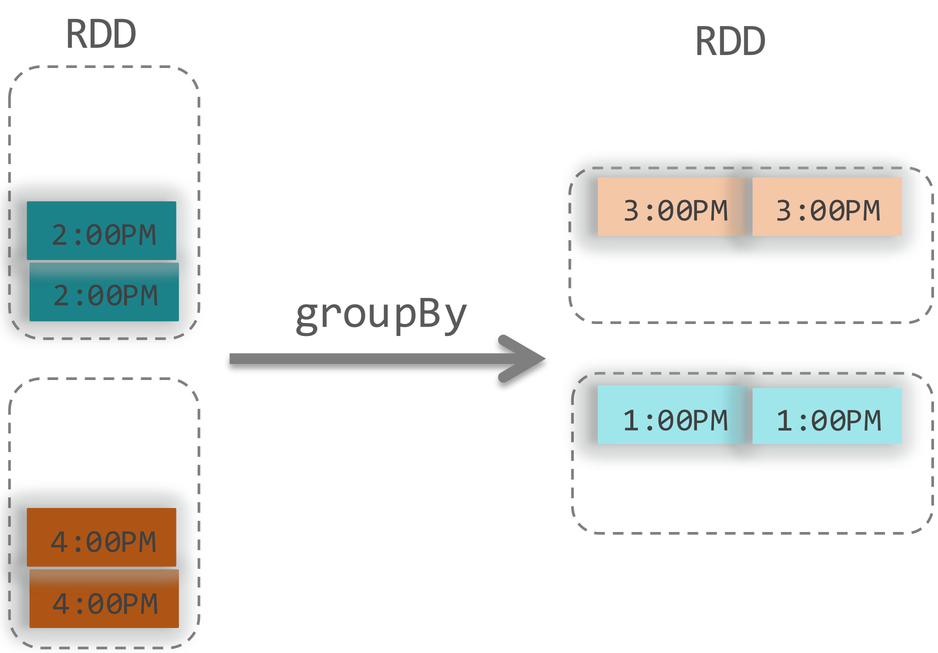
Group in Spark

- Data is shuffled and materialized



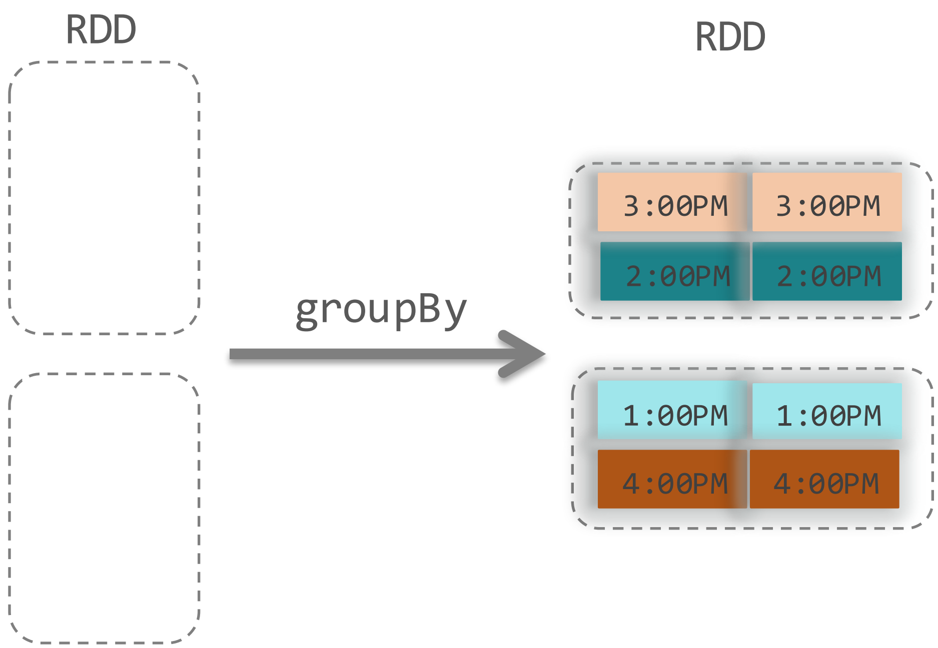
Group in Spark

- Data is shuffled and materialized



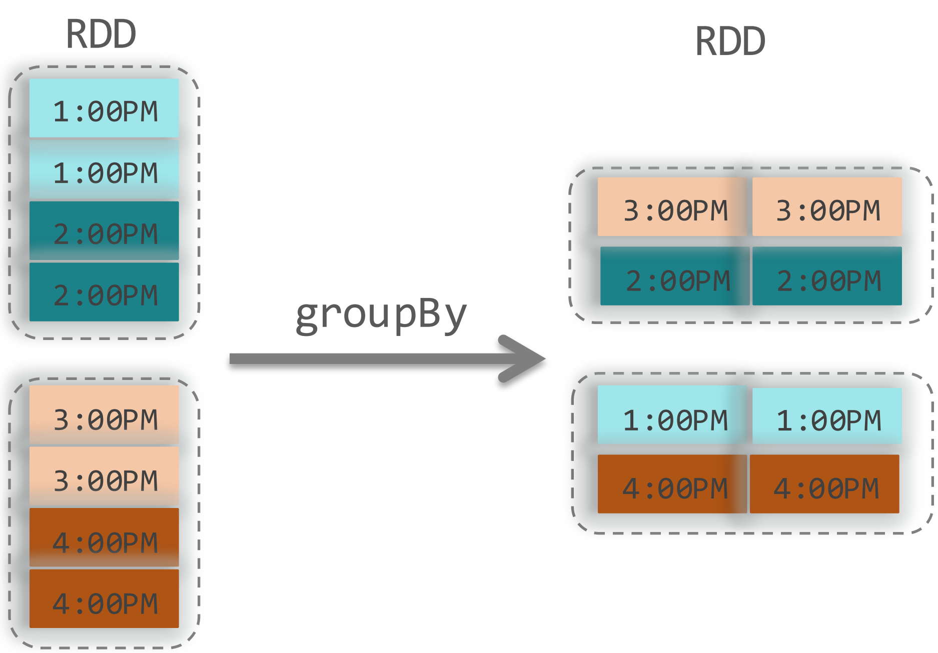
Group in Spark

- Data is shuffled and materialized



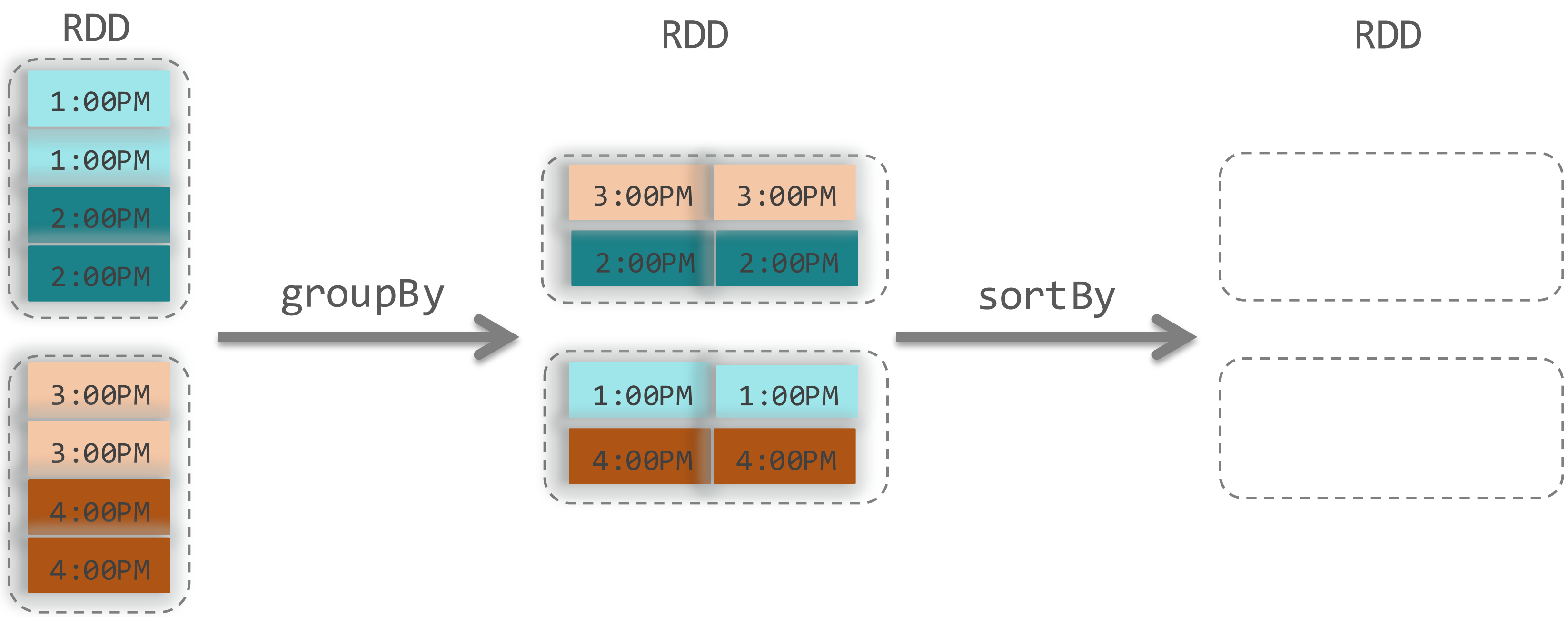
Group in Spark

- Temporal order is not preserved



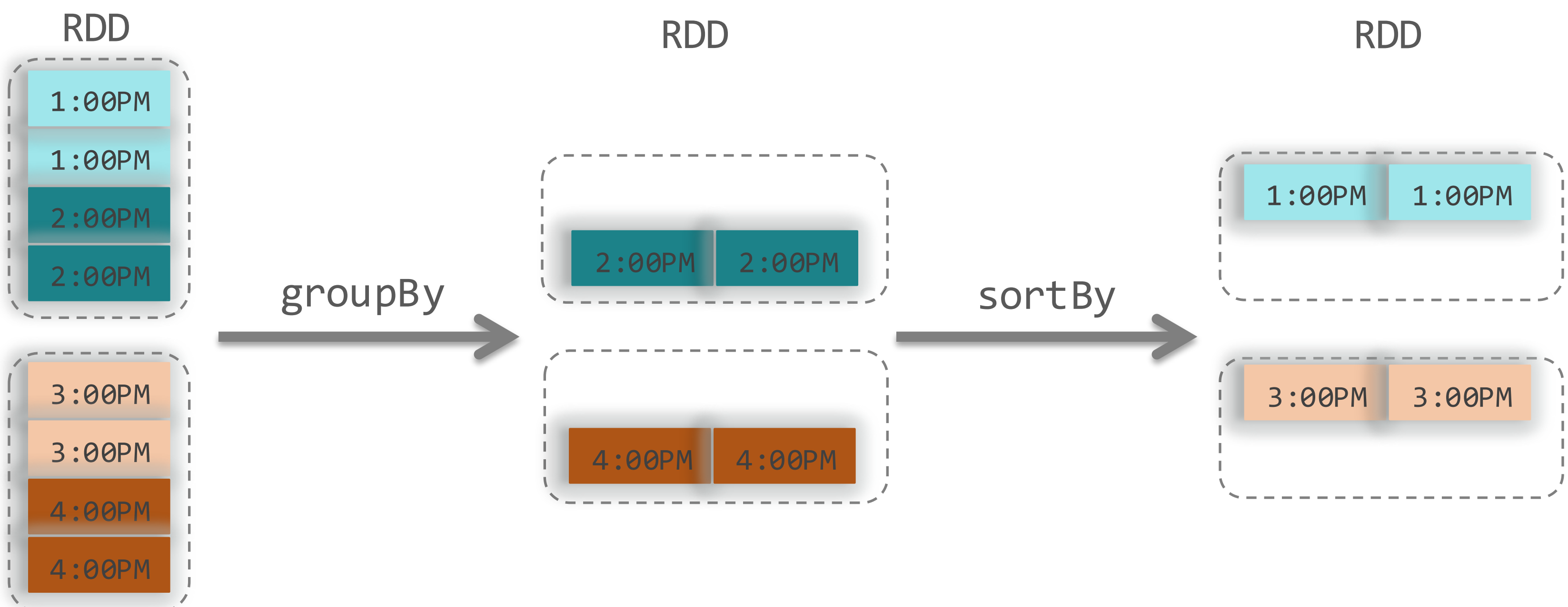
Group in Spark

- Another sort is required



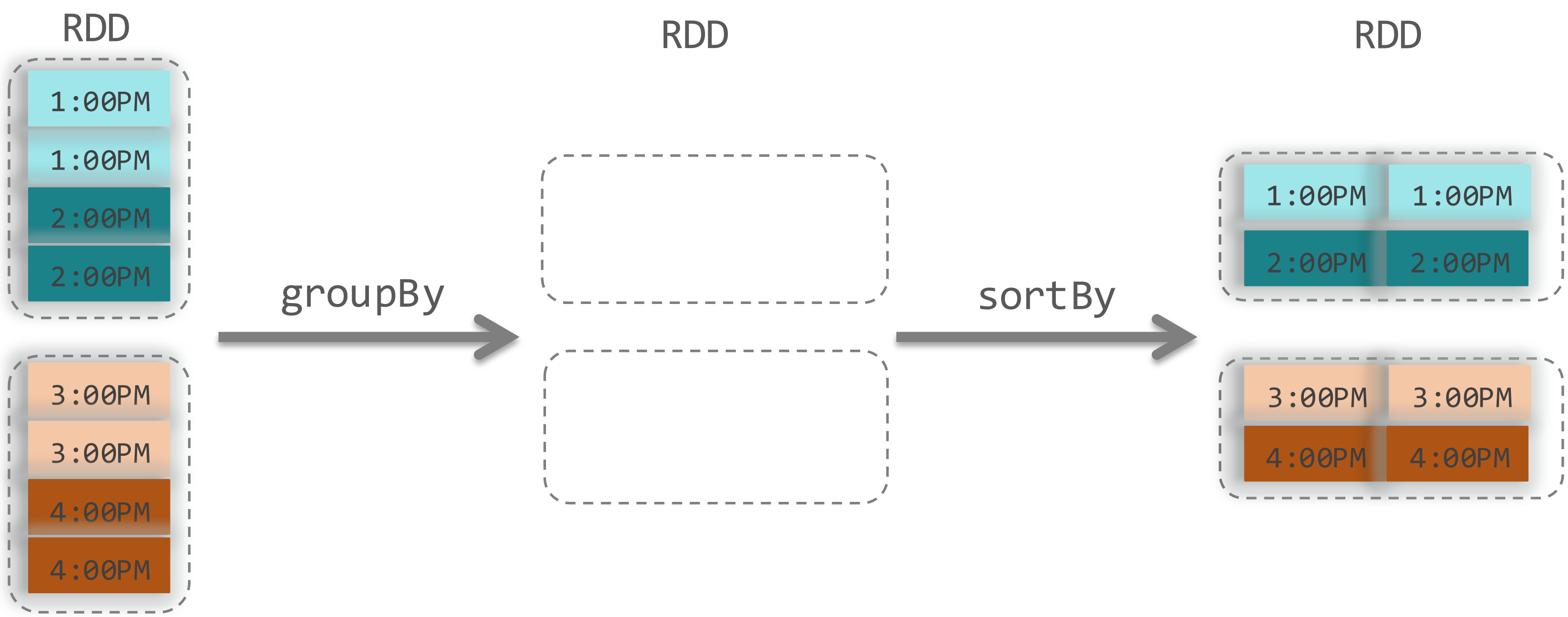
Group in Spark

- Another sort is required



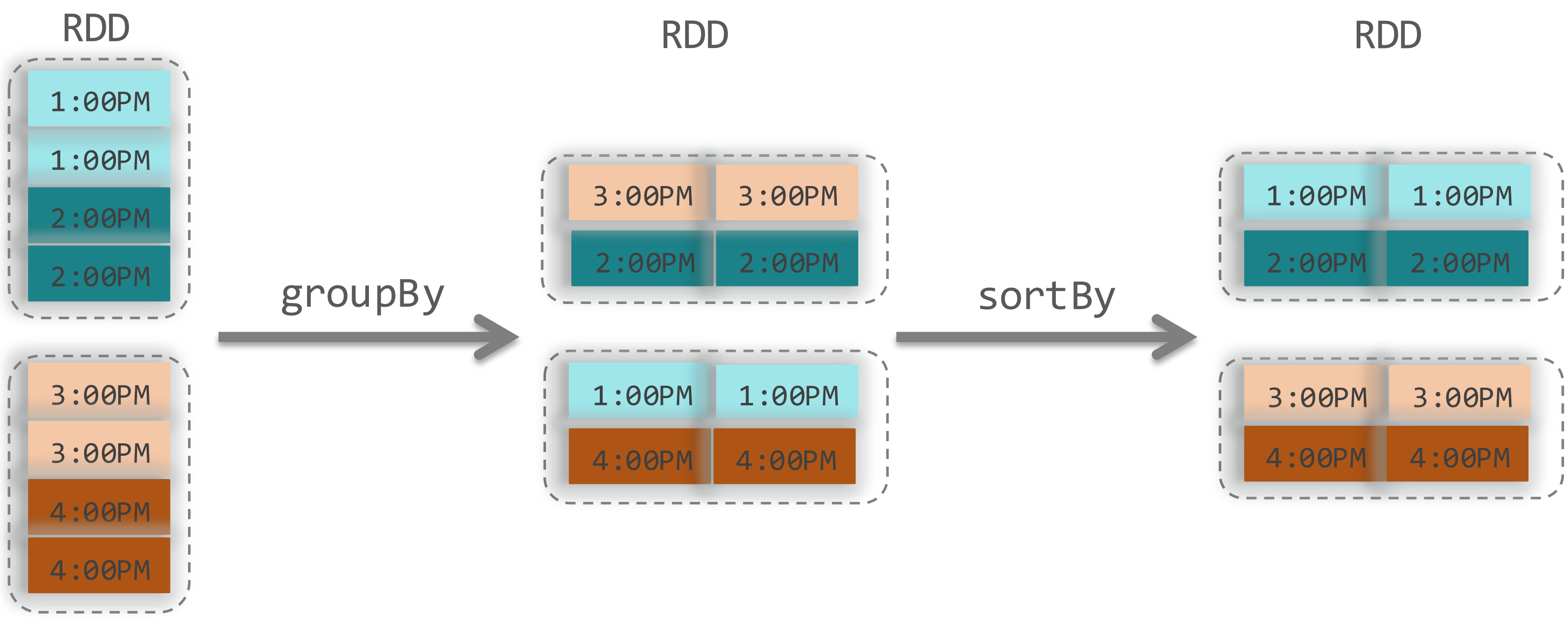
Group in Spark

- Back to correct temporal order



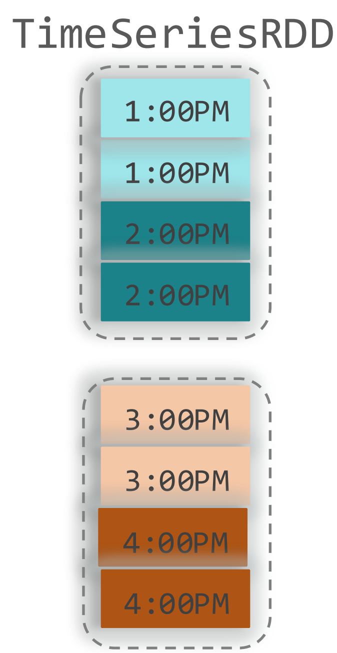
Group in Spark

- Back to temporal order



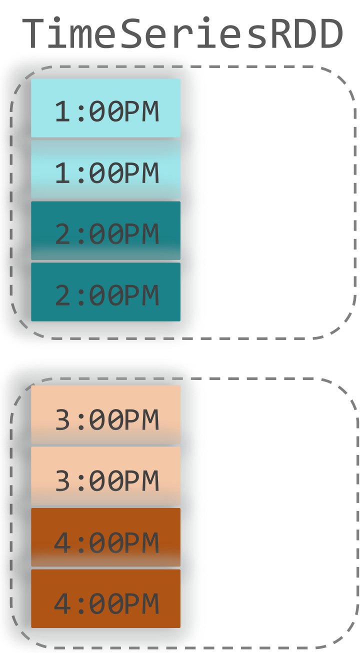
Group in Huohua

- Data is grouped locally as streams



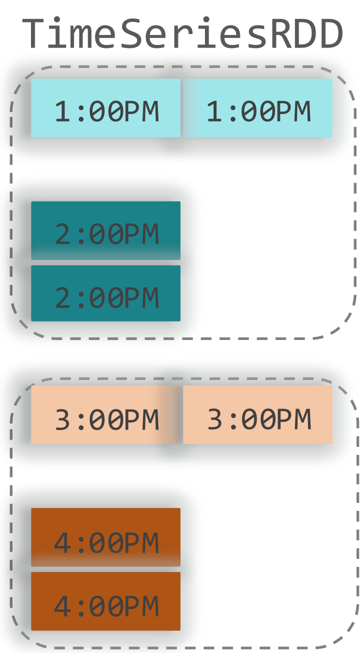
Group in Huohua

- Data is grouped locally as streams



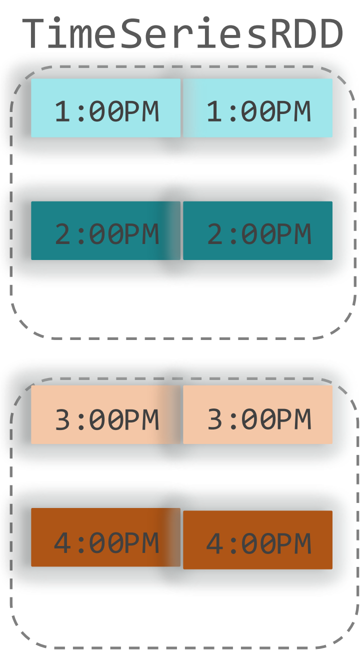
Group in Huohua

- Data is grouped locally as streams



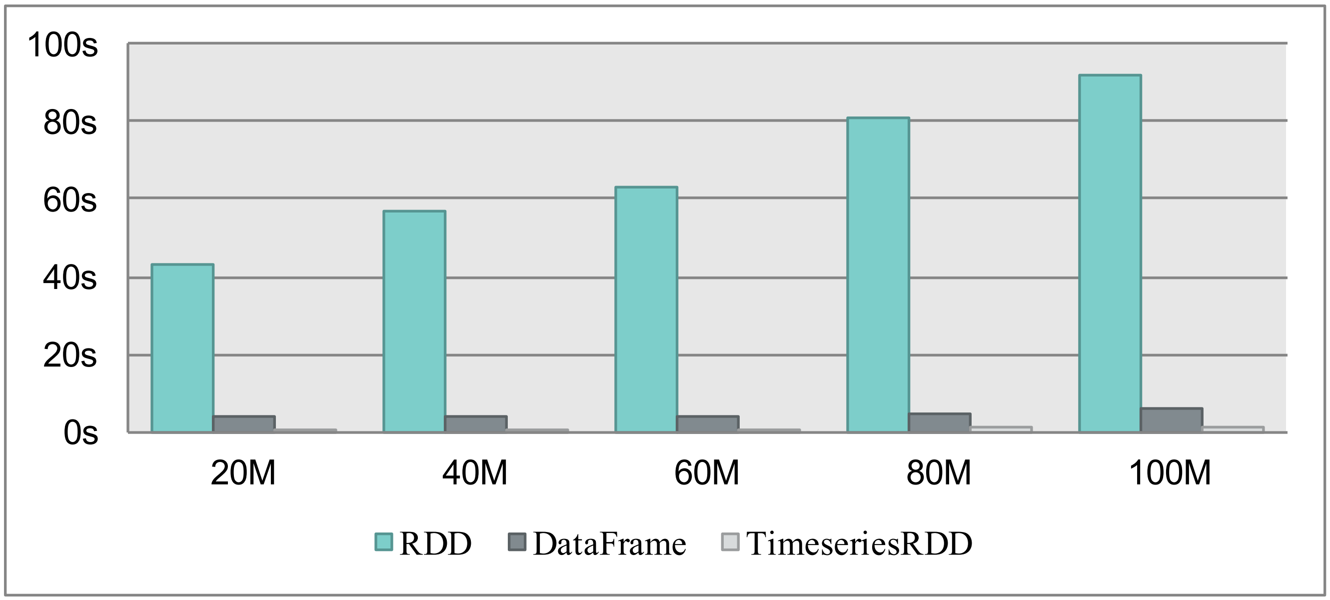
Group in Huohua

- Data is grouped locally as streams



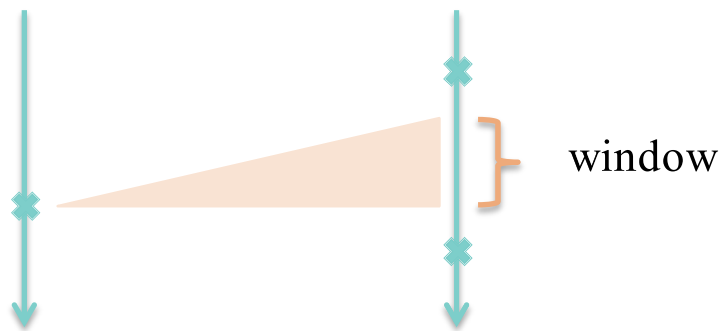
Benchmark for group

- Running time of *count* after *group*
  - 16 executors (10G memory and 4 cores per executor)
  - data is read from HDFS



Temporal join

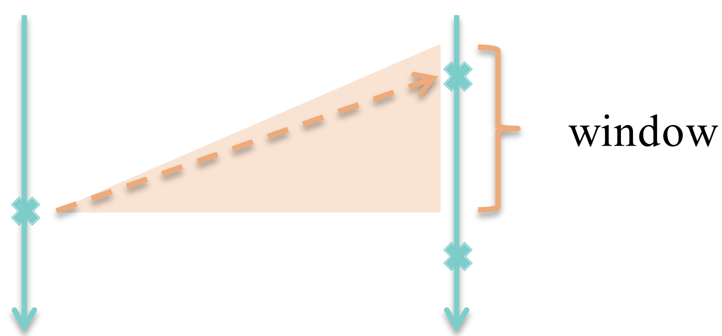
- A *temporal join* function is defined by a matching criteria over *time*
- A typical matching criteria has two parameters
  - *direction* – whether it should look-backward or look-forward
  - *window* - how much it should look-backward or look-forward



look-backward temporal join

Temporal join

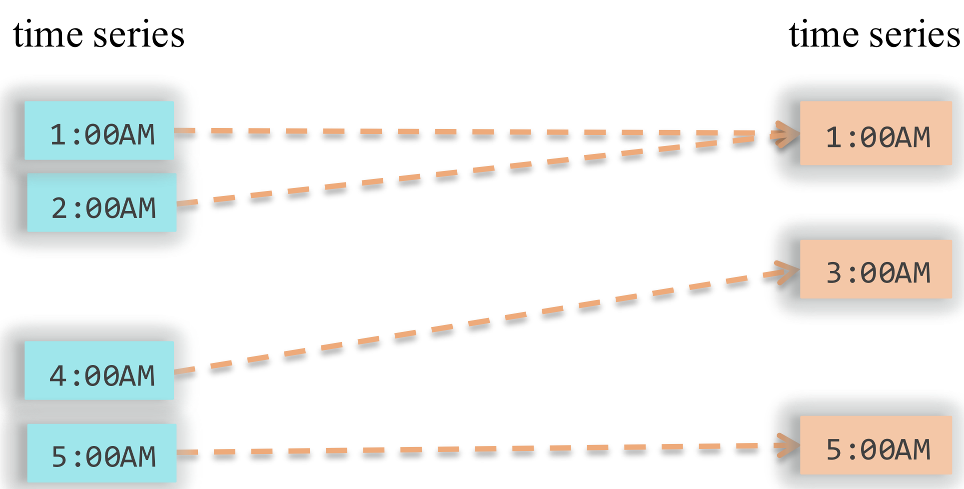
- A *temporal join* function is defined by a matching criteria over *time*
- A typical matching criteria has two parameters
  - *direction* – whether it should look-backward or look-forward
  - *window* - how much it should look-backward or look-forward



look-backward temporal join

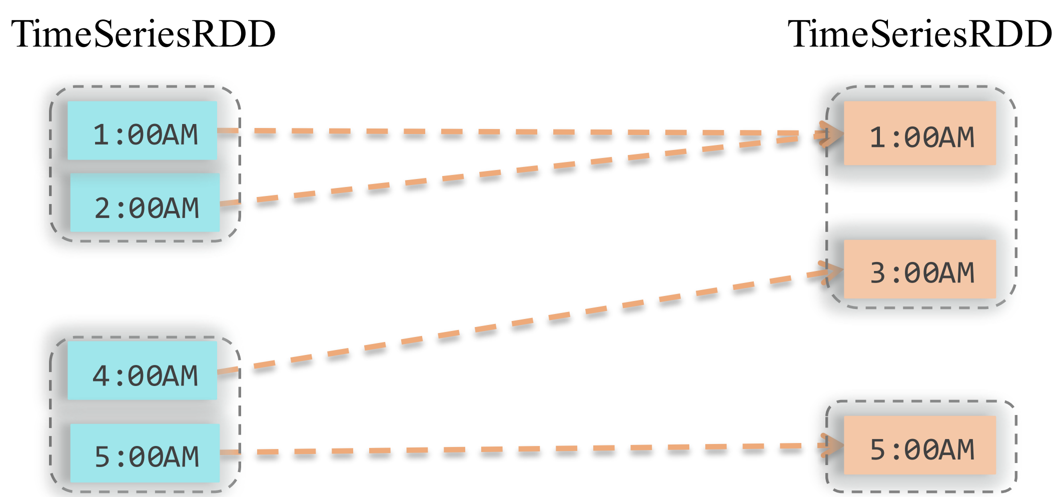
Temporal join

- Temporal join with criteria look-back and window of length 1



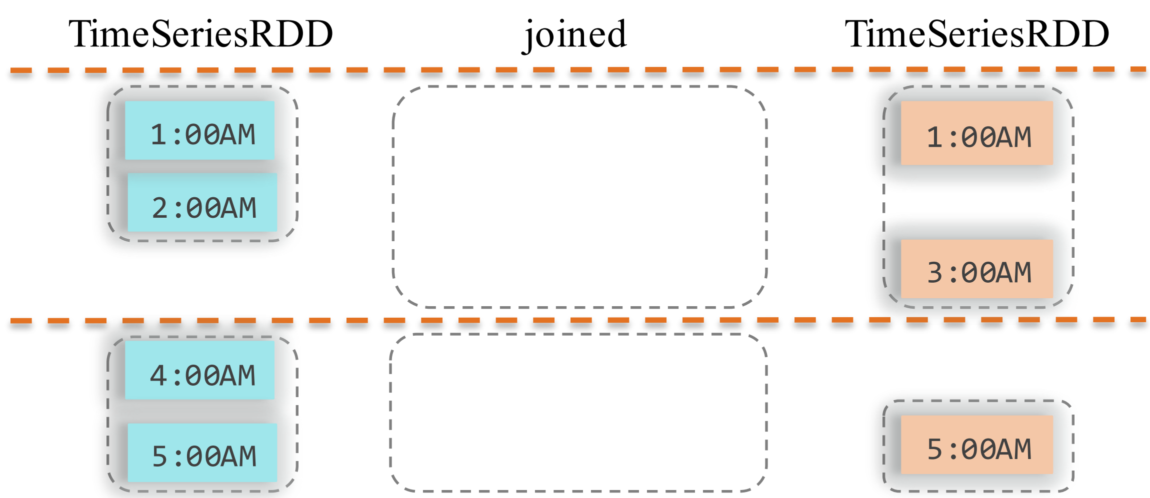
Temporal join

- Temporal join with criteria look-back and window of length 1
  - How do we do temporal join in TimeSeriesRDD?



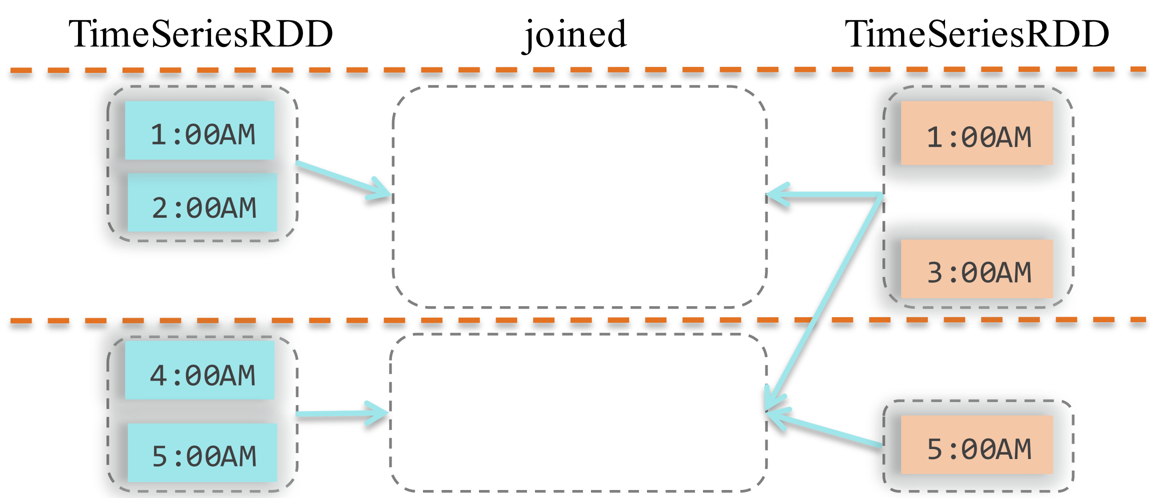
Temporal join in Huohua

- Temporal join with criteria look-back and window of length 1
  - partition *time* space into disjoint intervals



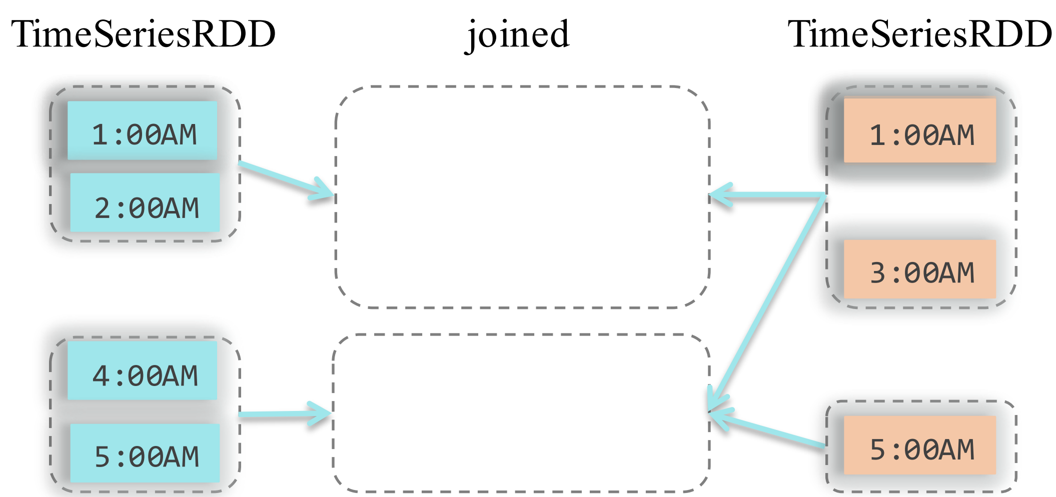
Temporal join in Huohua

- Temporal join with criteria look-back and window of length 1
  - Build dependency graph for the joined TimeSeriesRDD



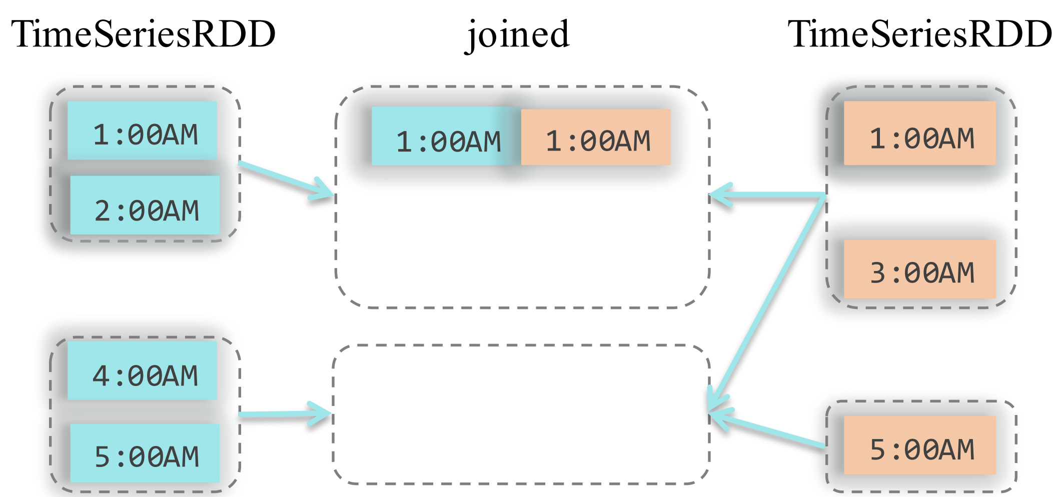
Temporal join in Huohua

- Temporal join with criteria look-back and window 1
  - Join data as streams per partition



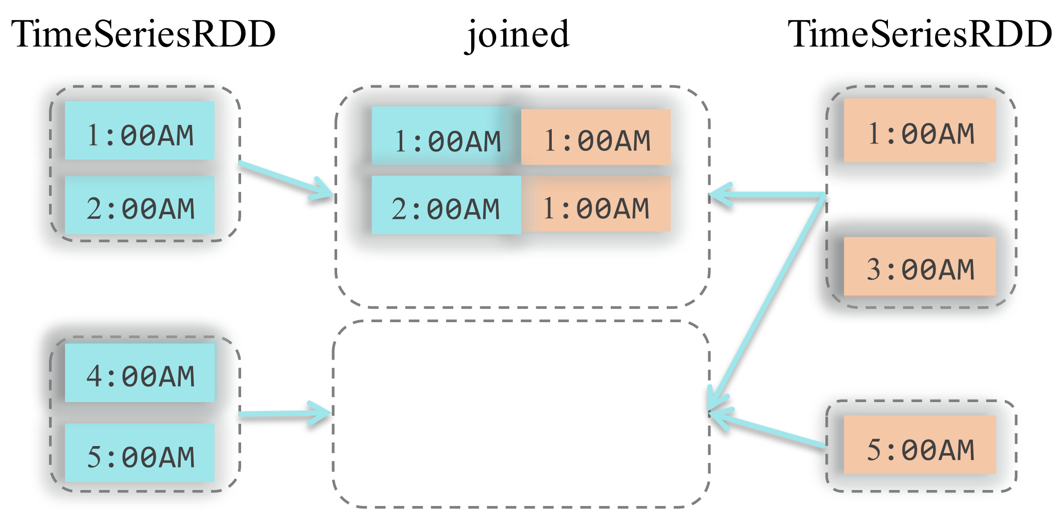
Temporal join in Huohua

- Temporal join with criteria look-back and window 1
  - Join data as streams



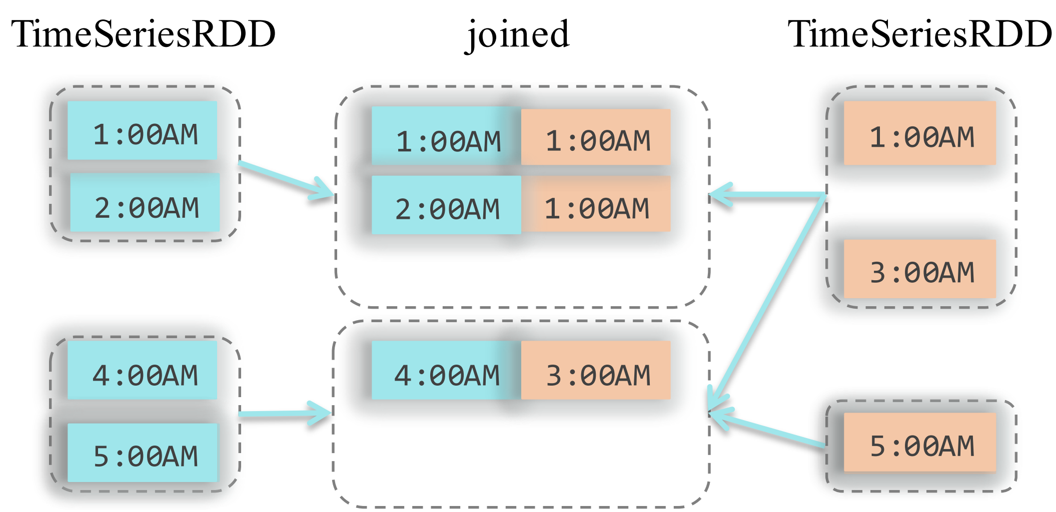
Temporal join in Huohua

- Temporal join with criteria look-back and window 1
  - Join data as streams



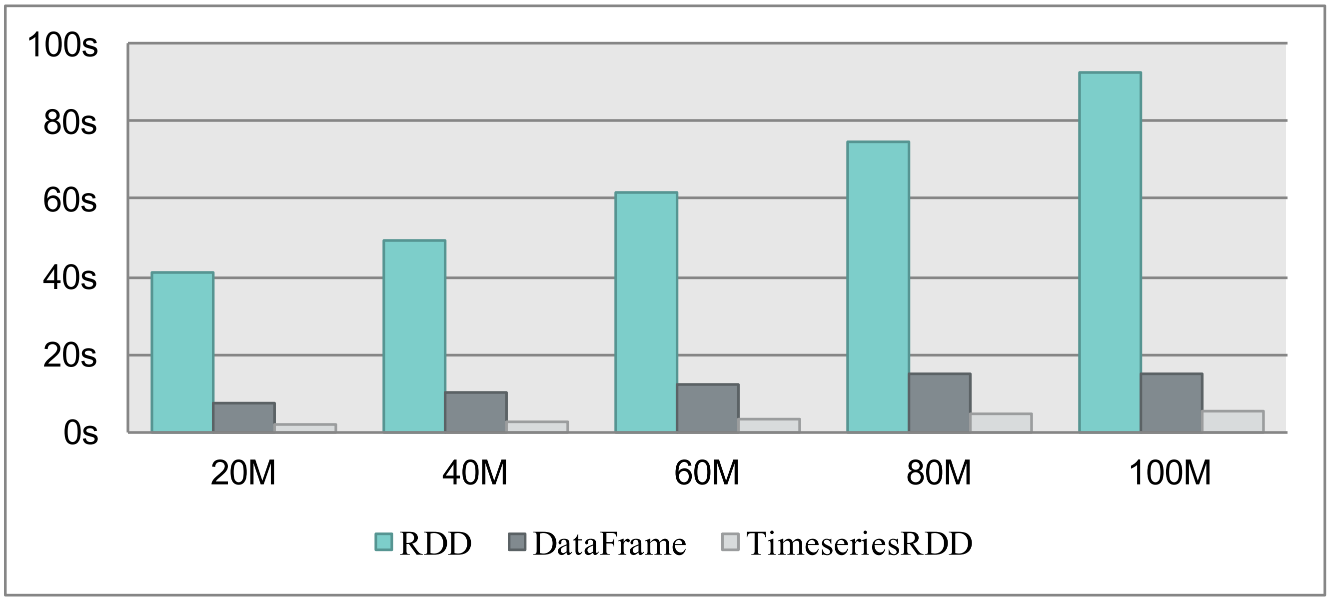
Temporal join in Huohua

- Temporal join with criteria look-back and window 1
  - Join data as streams



Benchmark for temporal join

- Running time of *count* after *temporal join*
  - 16 executors (10G memory and 4 cores per executor)
  - data is read from HDFS



Functions over TimeSeriesRDD

- group functions such as window, intervalization etc.
- temporal joins such as look-forward, look-backward etc.
- summarizers such as average, variance, z-score etc. over
  - windows
  - Intervals
  - cycles

Open Source

- Not quite yet …
- https://github.com/twosigma

Future work

- Dataframe / Dataset integration
  - Speed up
  - Richer APIs
- Python bindings
- More summarizers

Key contributors

- Christopher Aycock
- Jonathan Coveney
- Jin Li
- David Medina
- David Palaitis
- Ris Sawyer
- Leif Walsh
- Wenbo Zhao

Thank you

- QA