



influx/days

Advanced Data Analysis

Emanuele Della Valle

Prof. @ Politecnico di Milano

Founder & Partner @ Quantia Consulting

Marco Balduini

Founder & CEO @ Quantia Consulting

Riccardo Tommasini

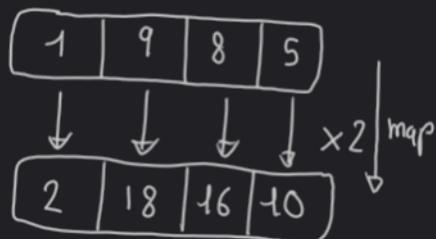
Prof. @ University of Tartu



Exploring flux map()

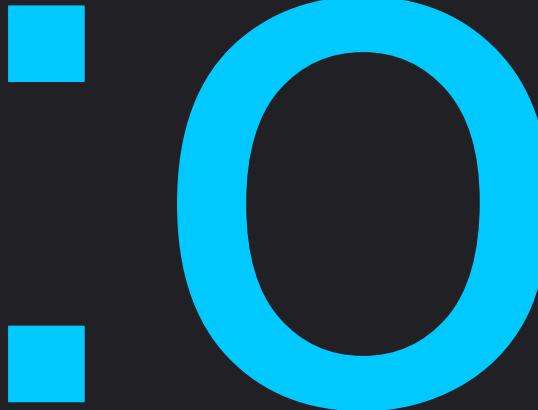
Map()

- How does it work?
 - It applies a function to each element of a collection
 - e.g., double each number
- What's for?
 - Combine data
 - Derive data
 - Enrich data
 - Clean data
 - ...
 - Turn data into information!

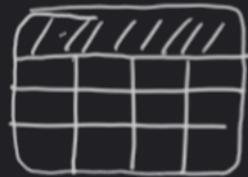
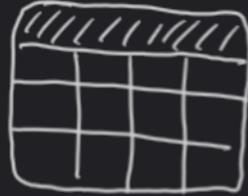


Map() in flux

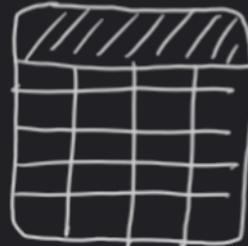
- The `map()` function applies a function to each record in the input tables
- The modified records are assigned to new tables based on the group key of the input table



Combine/derive data with map()



`map()`



_time	_m	_field	_value
10:45	cpu	system	1000
11:45	cpu	system	3000
GroupKey[cpu, system]			

hour
10
11

_time	_m	_field	_value
11:00	cpu	system	2000
11:30	cpu	system	2000
GroupKey[cpu, system]			

```
map(fn: (r) =>
  ({ hour:
    date.hour(t: r._time)
  })
)
```

hour
11
11

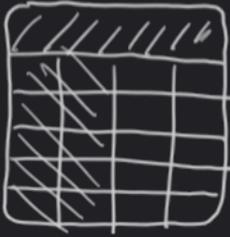
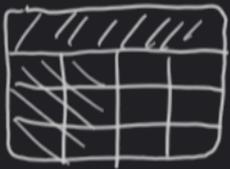
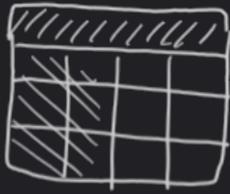
_time	_m	_field	_value
10:35	cpu	idle	100
GroupKey[cpu, idle]			

hour
10

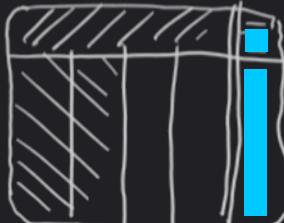
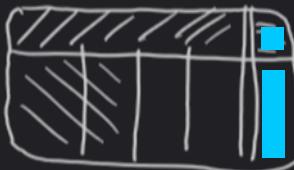
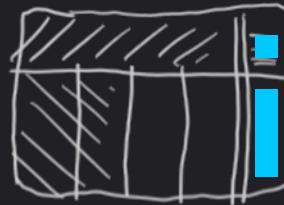
_time	_m	_field	_value
11:15	cpu	idle	200
11:50	cpu	idle	400
GroupKey[cpu, idle]			

hour
11
11

Enrich data using `map(r with ...)`



`map(r with ...)`



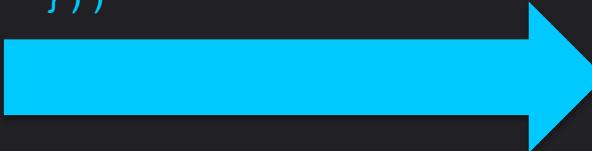
_time	_m	_field	_value
10:45	cpu	system	1000
11:45	cpu	system	3000
GroupKey[cpu, system]			

_time	_m	_field	_value
11:00	cpu	system	2000
11:30	cpu	system	2000
GroupKey[cpu, system]			

_time	_m	_field	_value
10:35	cpu	idle	100
GroupKey[cpu, idle]			

_time	_m	_field	_value
11:15	cpu	idle	200
11:50	cpu	idle	400
GroupKey[cpu, idle]			

```
map(fn: (r) =>
  ({ r with hour:
    date.hour(t: r._time)
  })
)
```



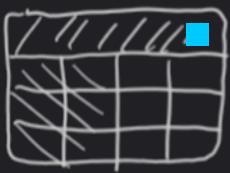
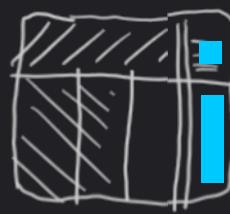
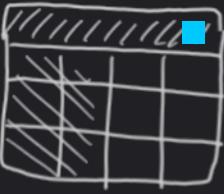
_time	_m	_field	_value	hour
10:45	cpu	system	1000	10
11:45	cpu	system	3000	11
GroupKey[cpu, system]				

_time	_m	_field	_value	hour
11:00	cpu	system	2000	11
11:30	cpu	system	2000	11
GroupKey[cpu, system]				

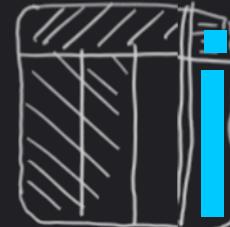
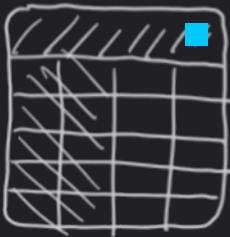
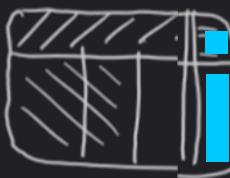
_time	_m	_field	_value	hour
10:35	cpu	idle	100	10
GroupKey[cpu, idle]				

_time	_m	_field	_value	hour
11:15	cpu	idle	200	11
11:50	cpu	idle	400	11
GroupKey[cpu, idle]				

Clean data using map(r with ...)



map(r with ■)



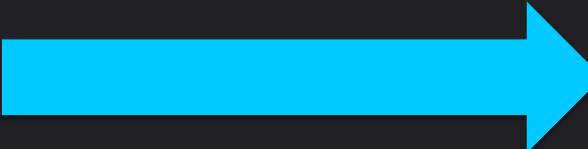
_time	_m	_field	_value
10:45	cpu	system	1000
11:45	cpu	system	3000
GroupKey[cpu, system]			

_time	_m	_field	_value
11:00	cpu	system	2000
11:30	cpu	system	2000
GroupKey[cpu, system]			

_time	_m	_field	_value
10:35	cpu	idle	100
GroupKey[cpu, idle]			

_time	_m	_field	_value
11:15	cpu	idle	200
11:50	cpu	idle	400
GroupKey[cpu, , idle]			

```
map(fn: (r) =>
  ({ r with _value :
    r._value/3000.0 }))
```



_time	_m	_field	_value
10:45	cpu	system	33.3%
11:45	cpu	system	100%
GroupKey[cpu, system]			

_time	_m	_field	_value
11:00	cpu	system	66.6%
11:30	cpu	system	66.6%
GroupKey[cpu, system]			

_time	_m	_field	_value
10:35	cpu	idle	3.3%
GroupKey[cpu, idle]			

_time	_m	_field	_value
11:15	cpu	idle	6.6%
11:50	cpu	idle	13.3%
GroupKey[cpu, , idle]			

Exploring flux Custom Functions

Recall: What Is Flux?

- Flux is a functional data scripting and query language
- Written to be:
 - Useable: easy to learn
 - Readable: developers read more code than we write
 - **Composable: developers can build onto the language**
 - Testable: queries are code
 - Contributable: open source contributions matter

Defining and using a custom function

- Syntax

```
<<function name>> = (<<variable>>*) => <<implementation>>
```

- Example

```
squared = (r) => r*r
```

```
from(bucket:"foo")  
|> range(start: -1h)  
|> filter(fn: (r) => r._measurement == "samples")  
|> map(fn: (r) => ({ _value: squared(r._value)}))  
|> filter(fn: (r) => r._value > 23.2)
```

Defining a custom pipe forwardable function

- Syntax

```
<<function name>> = (<<table>>=<-, <<variable>>*) =>  
    <<table>> |> << implementation >>
```

- Example

```
allSquared = (tables=<-) =>  
    tables |> map(fn: (r) => squared(r: r._value))  
  
from(bucket:"foo")  
    |> range(start: -1h)  
    |> filter(fn: (r) => r._measurement == "samples")  
    |> allSquared()  
    |> filter(fn: (r) => r._value > 23.2)
```

Let's get dirty!

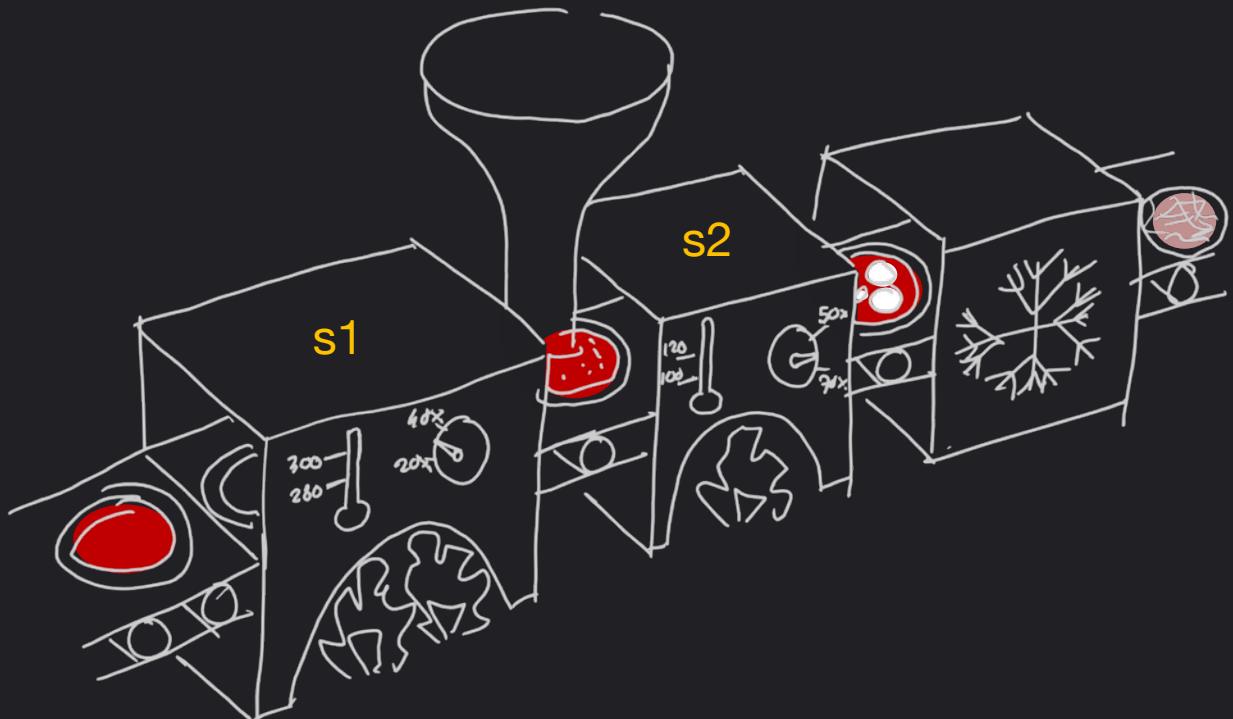


7

Continuous Linear Pizza Oven

Learning goals:

- Map functions
- Custom functions



Task

Correct the temperature observations of the cooking base area by subtracting a delta of 5°C to each value

- Use an inline map
- Create a custom function to be used in the inline map
- Create a custom pipe forwardable function that contains a map

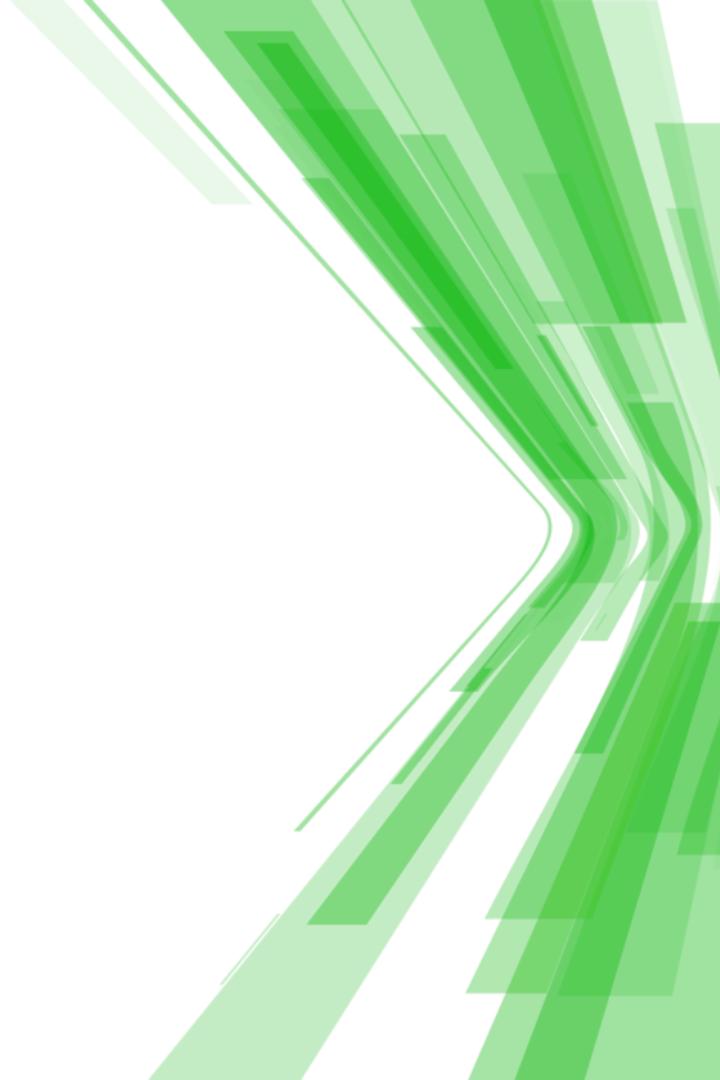
Quiz

- True or false
 - in a functional programming language, map() applies a function to each element of a collection – ?
 - in Flux, map() always adds a column to each table – ?
 - map() using the **with** clause can add a column to each table – ?
 - you can pass a custom functions only to a map() – ?
 - in a pipe-forwardable custom function with a parameter **(t=<-)**, **t** represents input tables that the function applies to – ?

Quiz answers

- True or false
 - in a functional programming language, map() applies a function to each element of a collection – T
 - in Flux, map() always adds a column to each table – F
 - map() using the **with** clause can add a column to each table – T
 - you can pass a custom functions only to a map() – F
 - in a pipe-forwardable custom function with a parameter **(t=<-)**, t represents input tables that the function applies to – T

Exploring flux Joining Time Series



Joining two time series



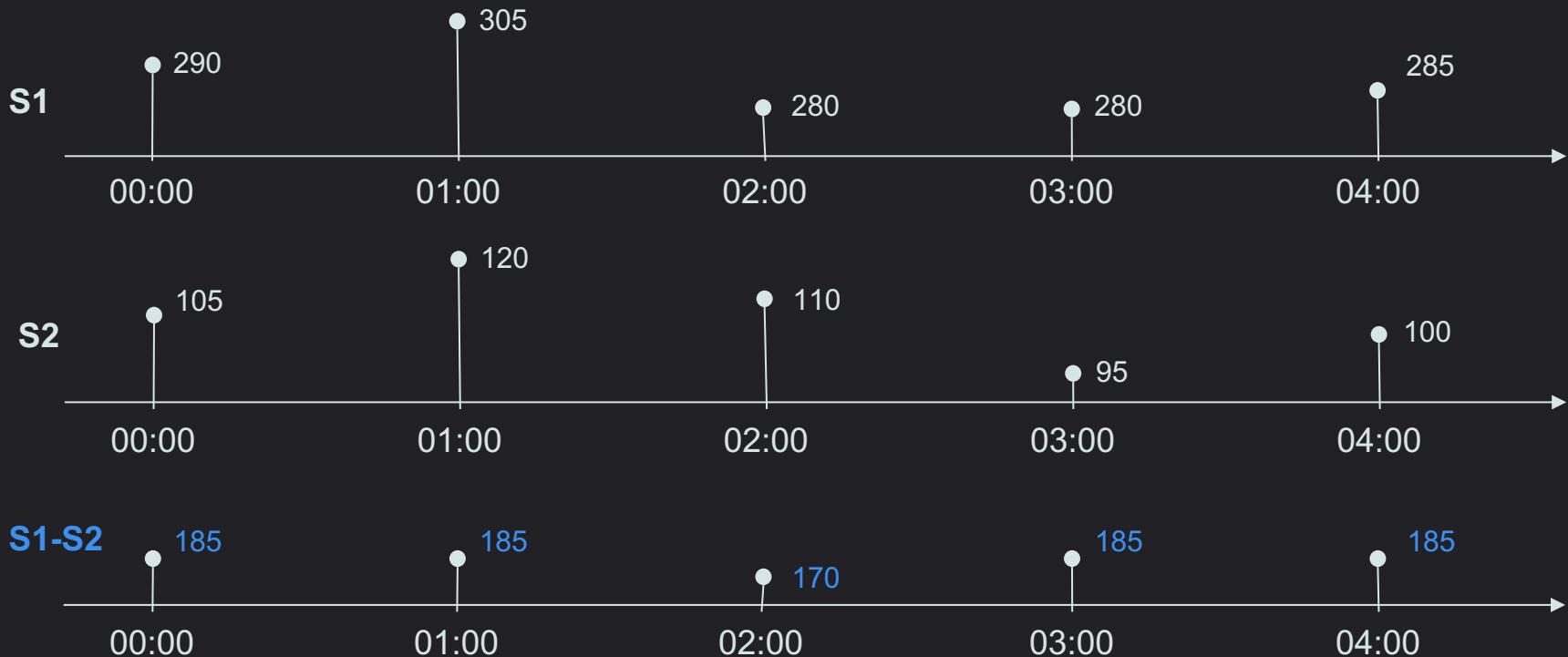
The ideal case



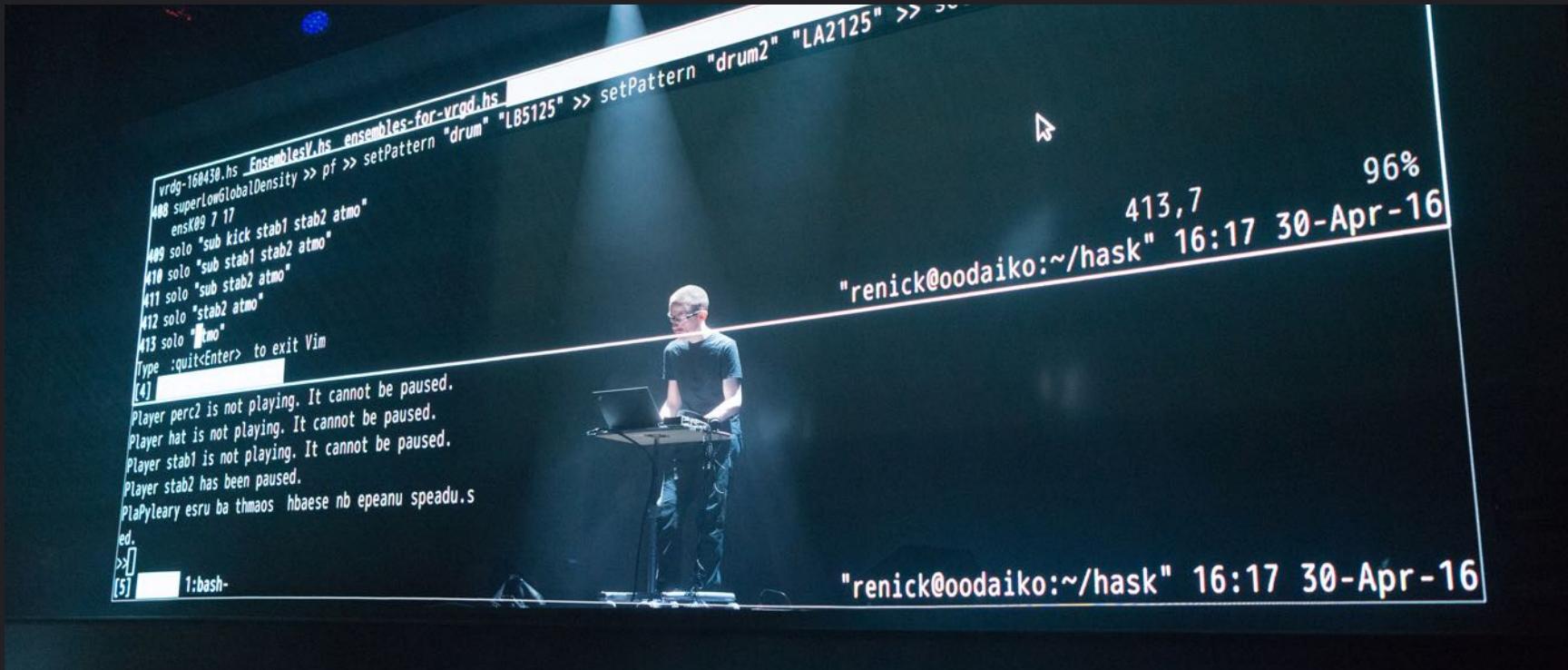
Running example on pizza quality issues

- If the temperature difference between the base cooking area and the mozzarella melting area is lower than 180° or greater than 190° , the mozzarella cheese will not melt properly or it will burn.

Ok ... sounds easy



Let's do some live coding



Here we go

```
ts12 = from(bucket: "training")
|> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
|> filter(fn: (r) => r._measurement == "iot-oven")
|> filter(fn: (r) => r._field == "temperature")

ts1 = ts12 |> filter(fn: (r) => r.sensor == "S1")
ts2 = ts12 |> filter(fn: (r) => r.sensor == "S2")

join(tables: {key1: ts1, key2: ts2}, on: ["_time"], method: "inner")

join |> map(fn: (r) => ({ _time: r._time,
                           _value: r._value_s1 - r._value_s2 }))
```

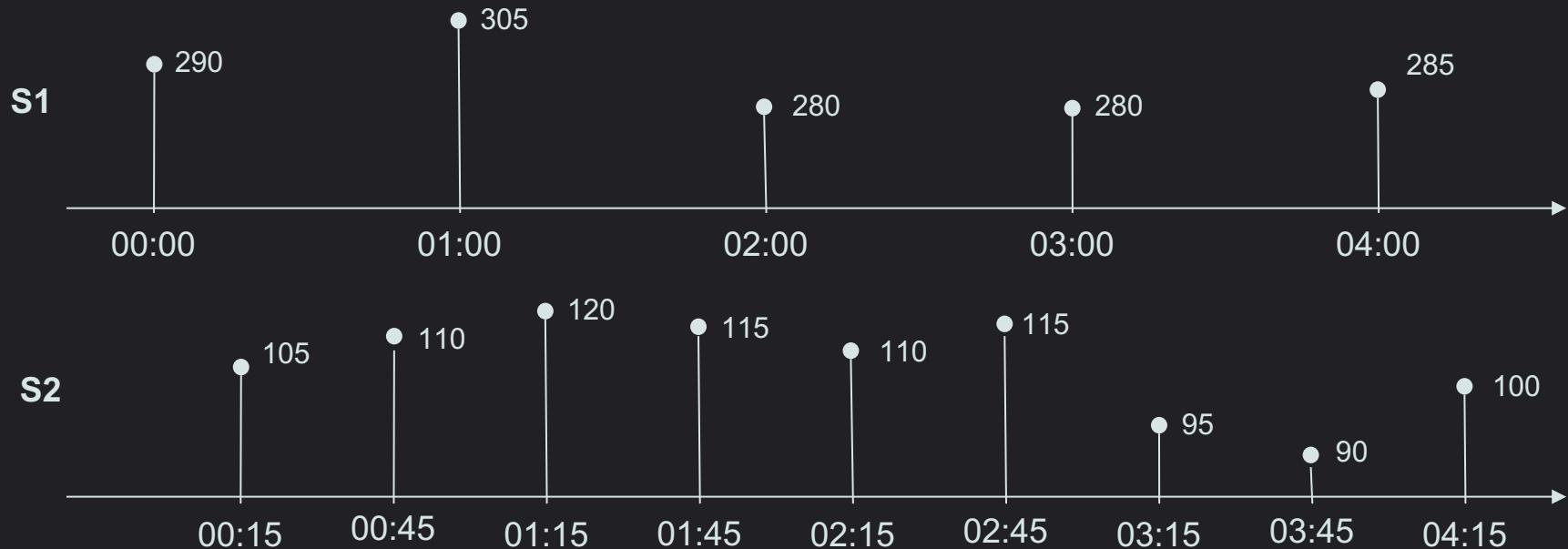
It gives no results ... why?



The reality is always more difficult than expected



A close look to the distribution of our data over time

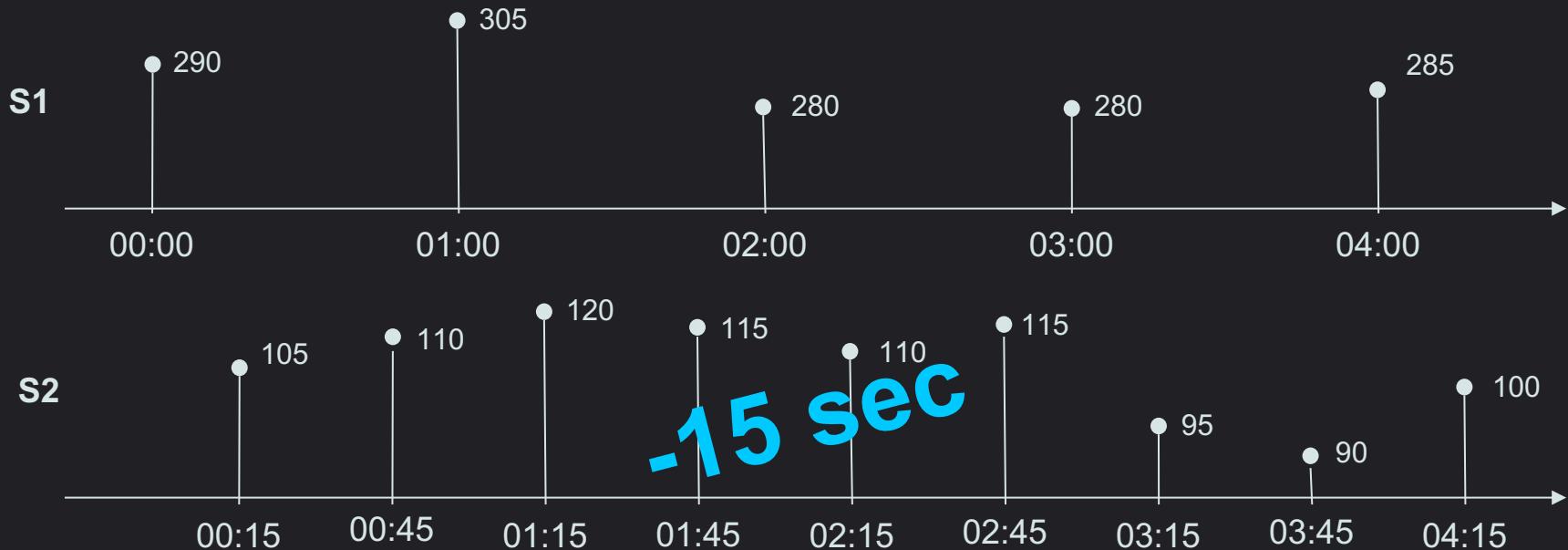


NOTE: S1 and S2 are not synchronized: metrics don't occur at the same time and rate

Can we synchronize them? Let's do a time shift!



Can we synchronize them with a time shift?

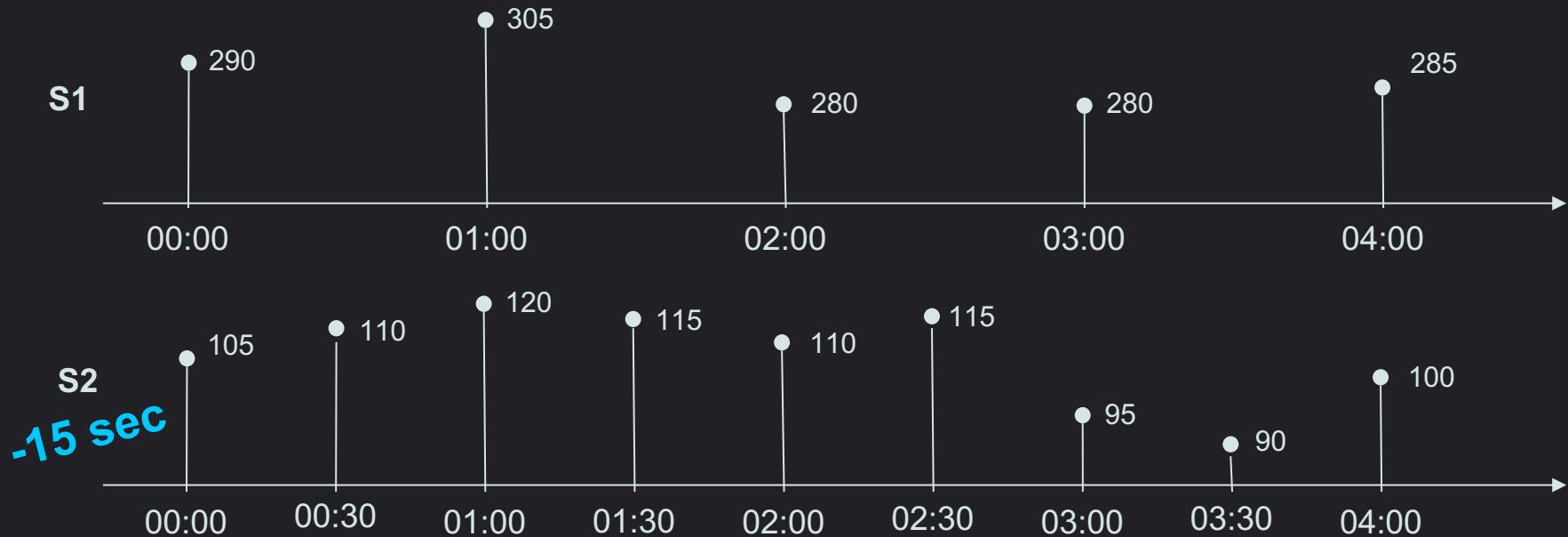


Here we go

```
temp = from(bucket: "training")
|> range(start: 2020-01-01T00:00:00Z, stop: 2020-01-01T00:05:00Z)
|> filter(fn: (r) => r._measurement == "iot-oven")
|> filter(fn: (r) => r._field == "temperature")
tempS1 = temp |> filter(fn: (r) => r.sensor == "S1")
tempS2 = temp |> filter(fn: (r) => r.sensor == "S2")
|> timeShift(duration: 15s, columns: ["_time"] )

join = join(  tables: {s1:tempS1, s2: tempS2},  on: ["_time"] )
join |> map(fn: (r) => ({ _time: r._time,
                           _value: r._value_s1 - r._value_s2 }))
```

Can we synchronize them with a time shift?



NOTE: some metrics now occur at the same time, but the rate is still different

Moreover, is -15 sec correct? Shall I do -30 sec, instead?



Shall we tumble?

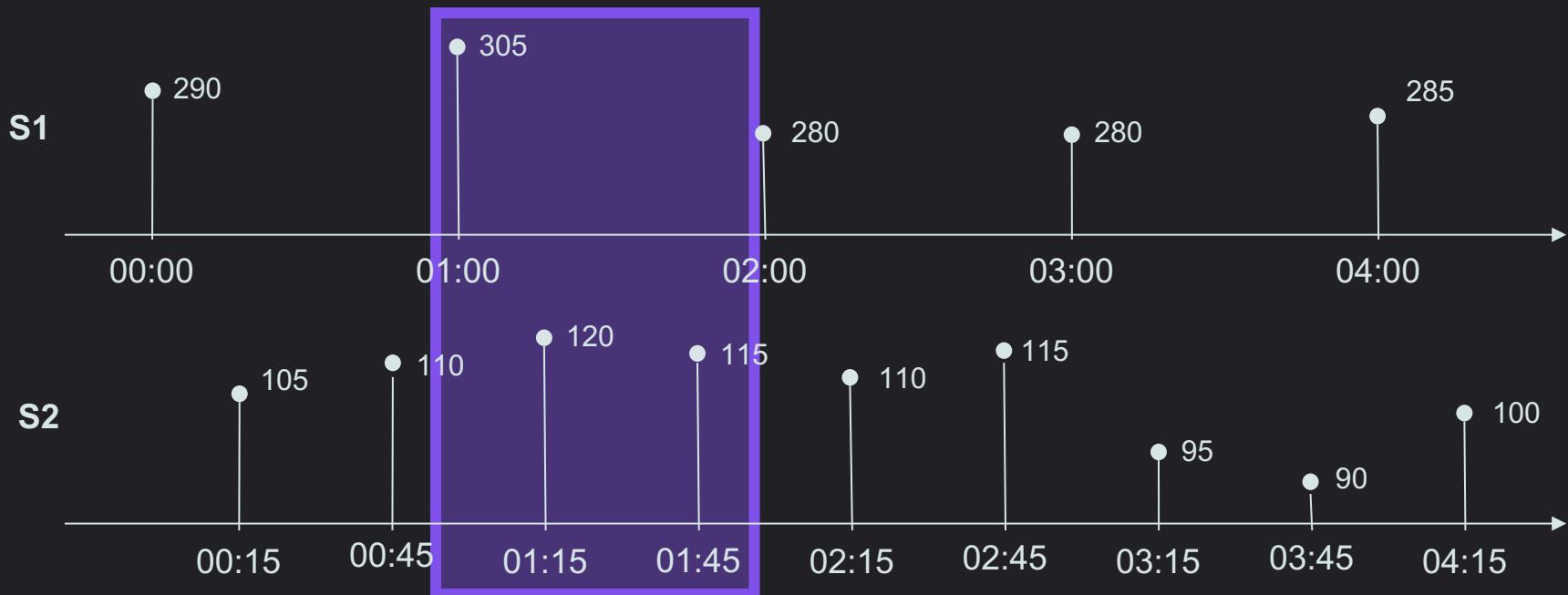


Can we synchronize them with a tumbling window?



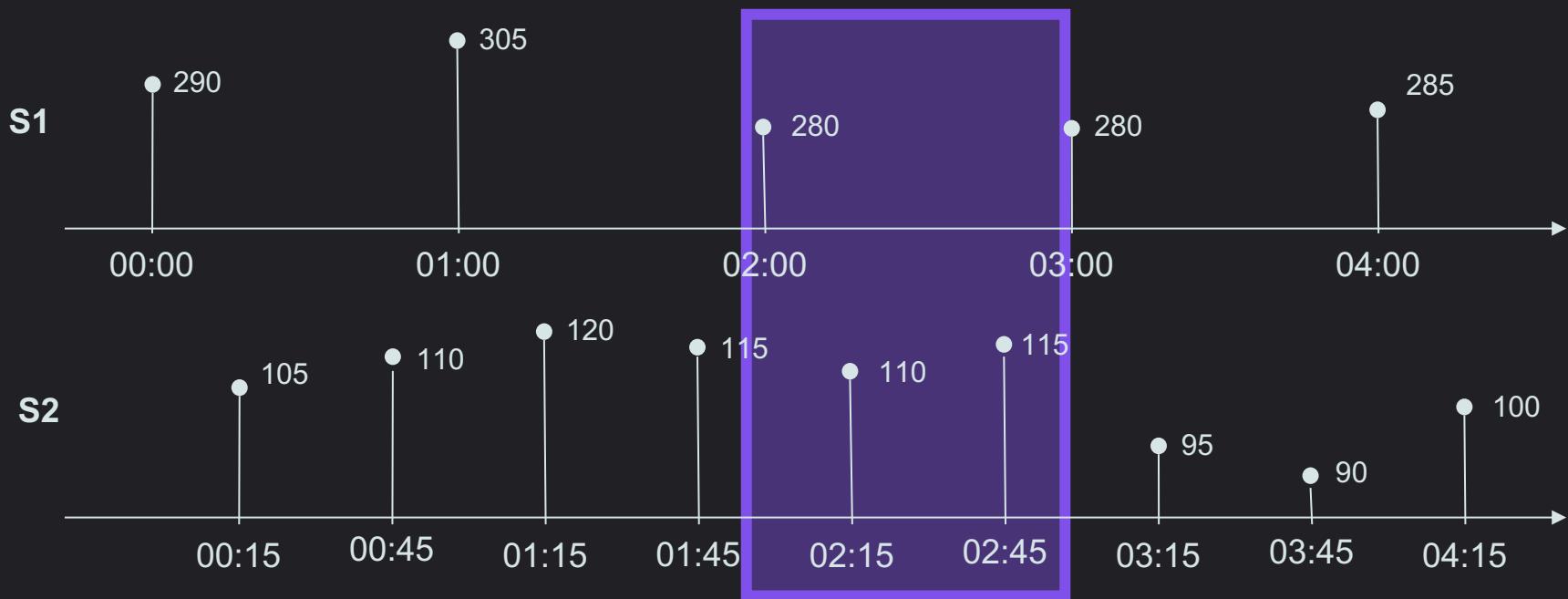
NOTE: S1 and S2 are not synchronized: metrics do not occur nor at the same time nor at the same rate

Can we synchronize them with a tumbling window?



NOTE: S1 and S2 are not synchronized: metrics do not occur nor at the same time nor at the same rate

Can we synchronize them with a tumbling window?



NOTE: S1 and S2 are not synchronized: metrics do not occur nor at the same time nor at the same rate

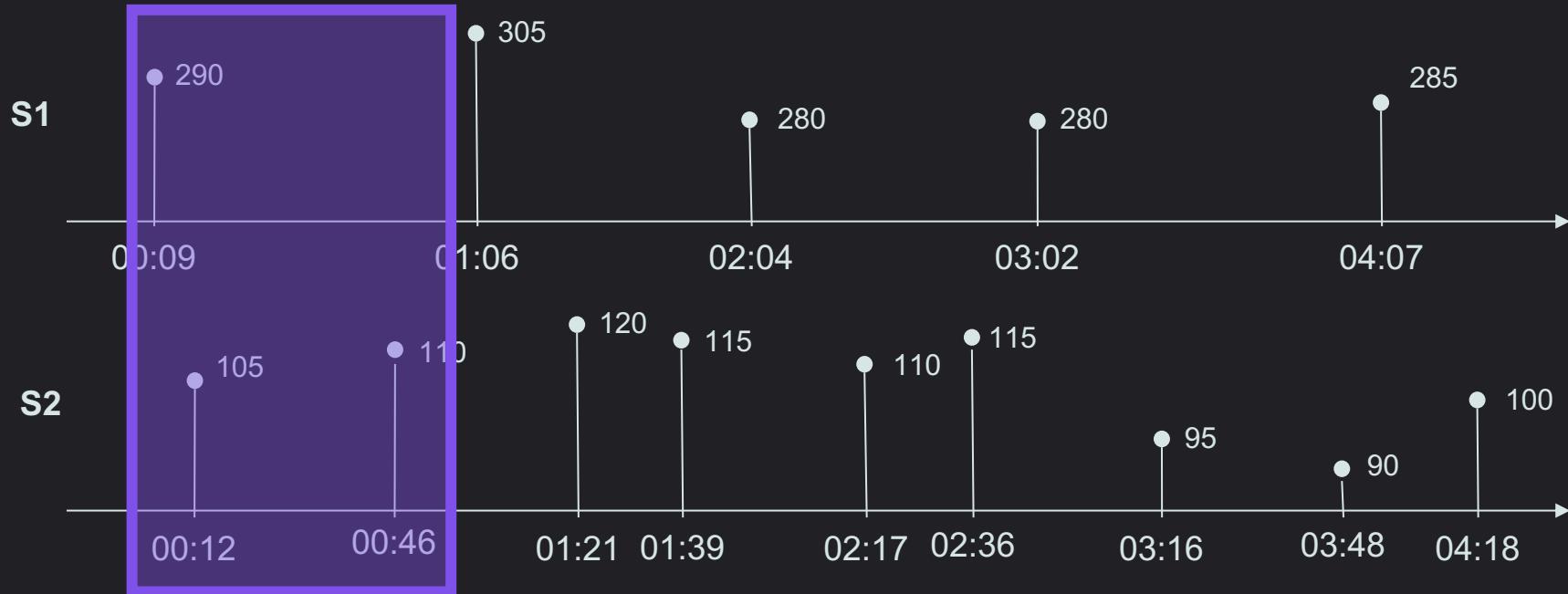
Here we go

```
temp = from(bucket: "training")
|> range(start: 2020-04-30T10:00:00Z, stop: 2020-04-30T10:05:00Z)
|> filter(fn: (r) => r._measurement == "iot-oven")
|> filter(fn: (r) => r._field == "temperature")
tempS1 = temp |> filter(fn: (r) => r.sensor == "S1")
                |> aggregateWindow(every: 60s, fn: mean)
tempS2 = temp |> filter(fn: (r) => r.sensor == "S2")
                |> aggregateWindow(every: 60s, fn: mean)
join = join(  tables: {s1:tempS1, s2: tempS2},  on: ["_time"] )
join |> map(fn: (r) => ({ _time: r._time,
                            _value: r._value_s1 - r._value_s2 }))
```

It works!

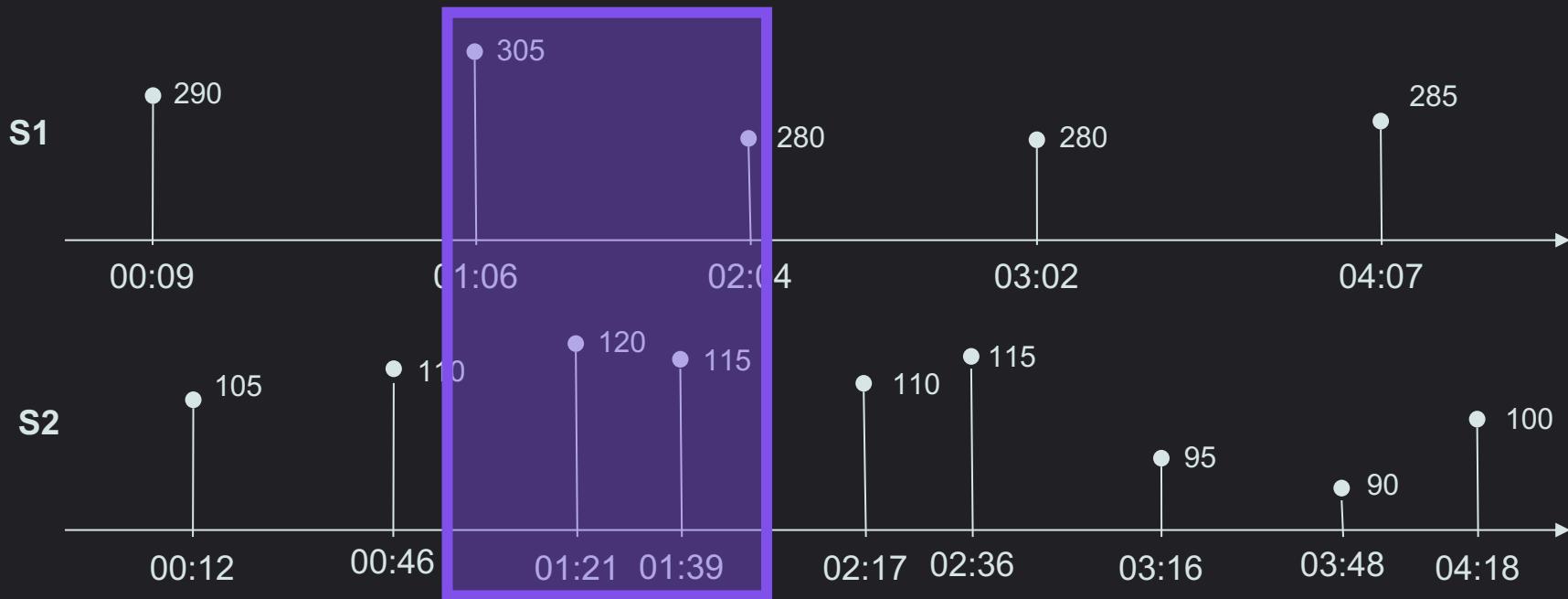


Bonus point: tumbling windows also synchronize events!



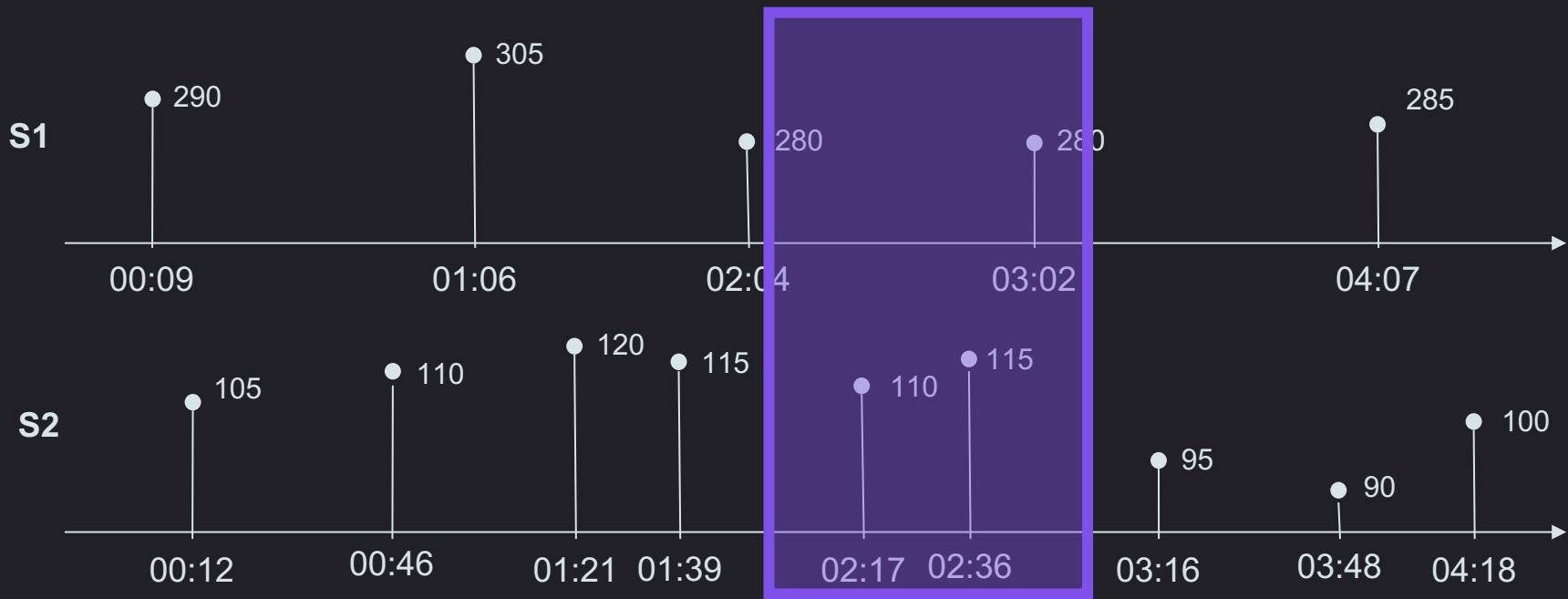
MENO: events are irregular time series

Bonus point: tumbling windows also synchronize events!



MENO: events are irregular time series

Bonus point: tumbling windows also synchronize events!



MENO: events are irregular time series

Quiz

- Which synchronization method shall you use before joining?
none | timeShift | aggregateWindow
 - one timeseries is regular, but the other is not - ?
 - two timeseries are regular, have the same rate, but metrics occur at a different time - ?
 - two timeseries are regular, have the same rate, and metrics occur at the same time - ?
 - two timeseries are irregular - ?
 - two timeseries are regular, but metrics don't occur at the same time and rate - ?

Quiz answers

- Which synchronization method shall you use before joining?
none | timeShift | aggregateWindow
 - one timeseries is regular, but the other is not - aggregateWindow
 - two timeseries are regular, have the same rate, but metrics occur at a different time - timeShift
 - two timeseries are regular, have the same rate, and metrics occur at the same time - none
 - two timeseries are irregular - aggregateWindow
 - two timeseries are regular, but metrics don't occur at the same time and rate - aggregateWindow

Let's get dirty!

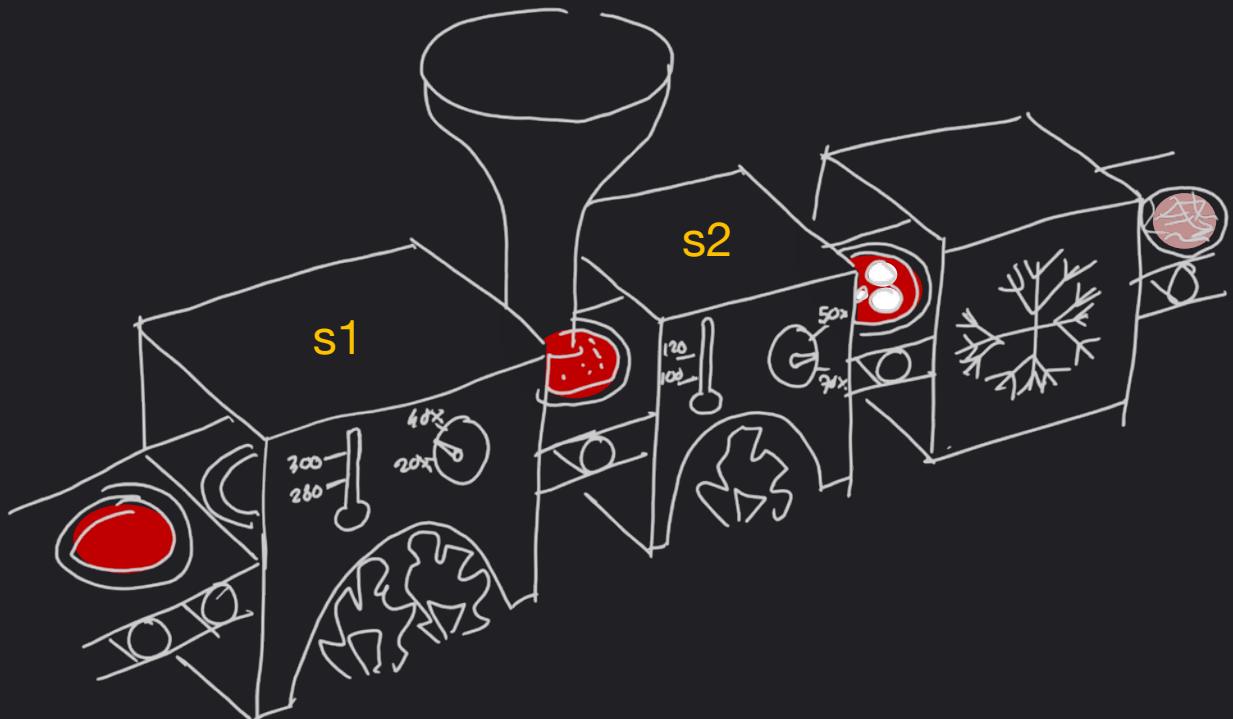


8

Continuous Linear Pizza Oven

Learning goals:

- Join data



Task

- Extract the difference between the humidity levels of the base cooking area and the mozzarella melting area. Find if the differences are lower than 20% or greater than 30%



influx/days



Advanced Data Analysis

Emanuele Della Valle Prof. @ Politecnico di Milano & Partner @ Quantia Consulting

Marco Balduini Founder & CEO @ Quantia Consulting

Riccardo Tommasini Prof. @ University of Tartu