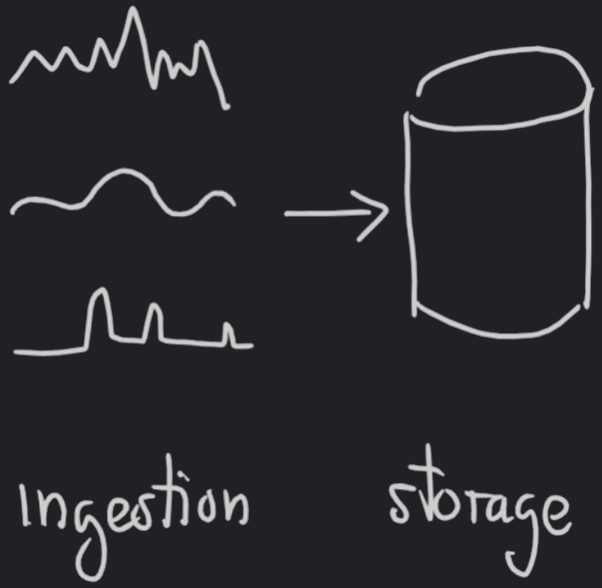
Basic Data Analysis

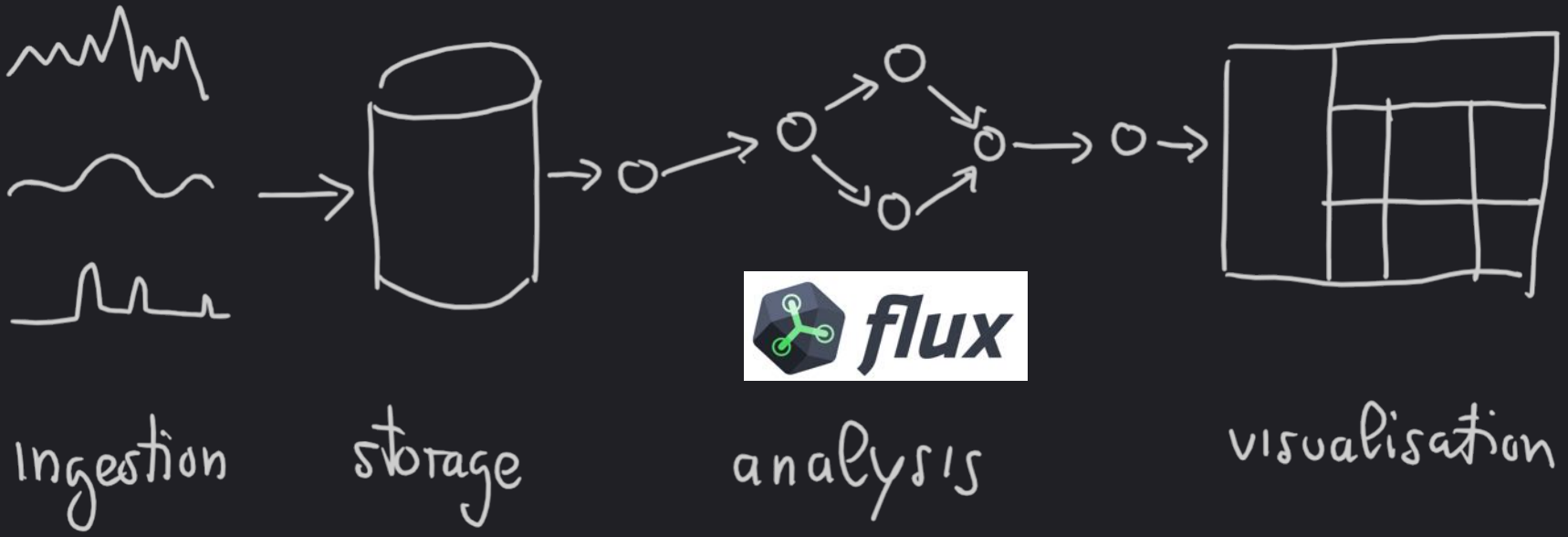
Emanuele Della Valle
Prof. @ Politecnico di Milano
Founder & Partner @ Quantia Consulting
Marco Balduini
Founder & CEO @ Quantia Consulting
Riccardo Tommasini
Prof. @ University of Tartu

Introduction

Data Lifecycle



Data Lifecycle




flux

Why Flux?

I don't want to have to write query code in one language and processing code in another language.

– Paul Dix - CTO InfluxData

What Is Flux?

~~Flux Query Language?~~

Query + Process + Act

Flux Data Scripting Language

What Is Flux?

- Written to be:
  - Useable: easy to learn
  - Readable: developers read more code than we write
  - Extensible: developers can extend the language
  - Composable: developers can build onto the language
  - Testable: queries are code
  - Contributable: open source contributions matter

What is Flux? (cont.)

- Open Source (MIT)
- Data-first mentality
- Streaming Data
- Planner-optimized
- Compatible with multiple sources and sinks

Flux vs. InfluxQL

- Flux is an alternative to InfluxQL with additional functionality like:
  - Joins
  - Math across measurements
  - Sort by any column
  - Pivot
  - Histograms
  - Covariance

- Reference: https://v2.docs.influxdata.com/v2.0/query-data/

Exploring flux
Basics

Basic Flux Language Elements

- Anatomy of a Flux query:
  - data source
  - time range
  - data filters

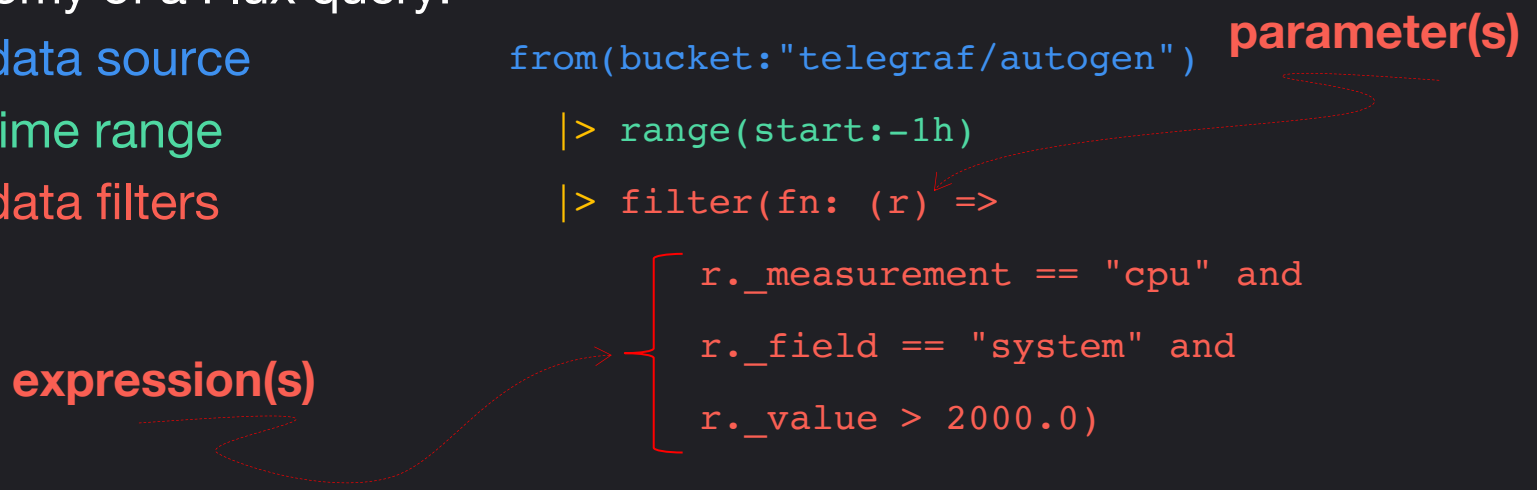
Anonymous function

```
from(bucket:"telegraf/autogen")
  |> range(start:-1h)
  |> filter(fn: (r) =>
       r._measurement == "cpu" and
       r._field == "system" and
       r._value > 2000.0)
```

Pipe-forward operator: chain operations together

Basic Flux Language Elements

- Anatomy of a Flux query:
  - data source
  - time range
  - data filters



Flux Data Model

An infinite stream…

of finite tables…

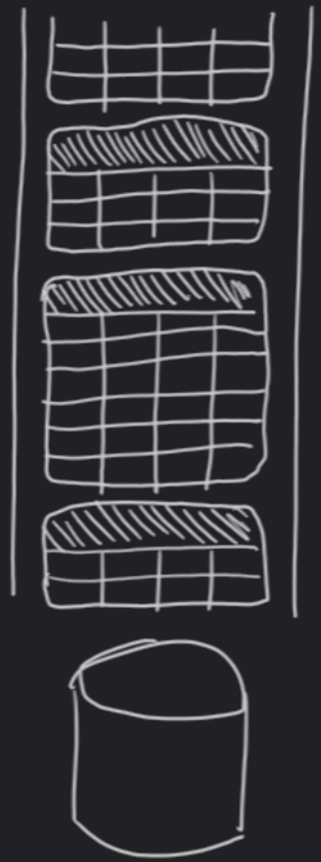
identified by the GroupKey

Flux Data Model

An infinite stream…

of finite tables…

identified by the GroupKey



From bucket to tables

_time	_m	host	_field	_value
14…0	cpu	serverA	system	2342
14…2	cpu	serverA	system	477
GroupKey[cpu, serverA, system]				

_time	_m	host	_field	_value
14…1	cpu	serverB	system	2779
14…3	cpu	serverB	system	1274
GroupKey[cpu, serverB, system]				

[…]

_time	_m	host	_field	_value
14…1	cpu	serverB	idle	1789
14…2	cpu	serverB	idle	2475
GroupKey[cpu, serverB, idle]				

```
from(bucket:"telegraf/autogen")
  |> range(start:-1s)
```

_time	_m	host	idle	system
14…0 … 14…1	cpu … cpu	serverA … serverB	1667 … 1789	2342 … 2779

_m = _measurement

Query Process

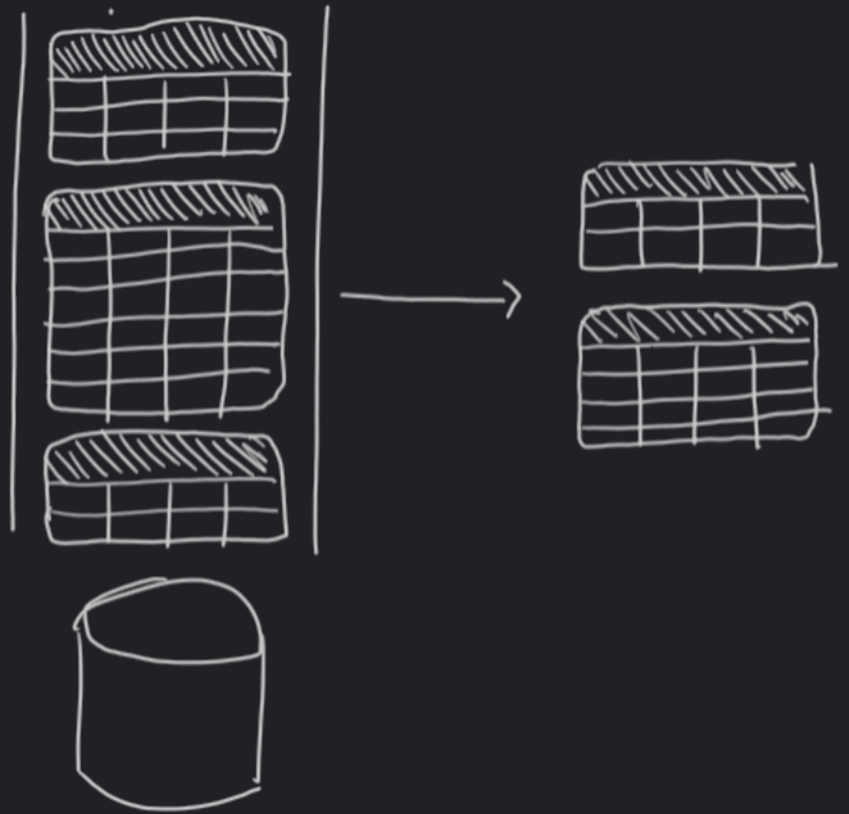
For Each Table:

1. Convert incoming rows into 0 or more outgoing rows
2. Map outgoing rows to tables

Query Process

For Each Table:

1. Convert incoming rows into 0 or more outgoing rows
2. Map outgoing rows to tables



_time	_m	host	_field	_value
14…1	cpu	serverB	idle	1789
14…2	cpu	serverB	idle	2475
GroupKey[cpu, serverB, idle]				

_time	_m	host	_field	_value
14…1	cpu	serverB	system	2779
14…3	cpu	serverB	system	1274
GroupKey[cpu, serverB, system]				

[…]

_time	_m	host	_field	_value
14…0	cpu	serverA	system	2342
14…2	cpu	serverA	system	477
GroupKey[cpu, serverA, system]				

```
|> filter(fn: (r) =>
    r._measurement == "cpu" and
    r._field == "system" and
    r._value > 2000.0)
```

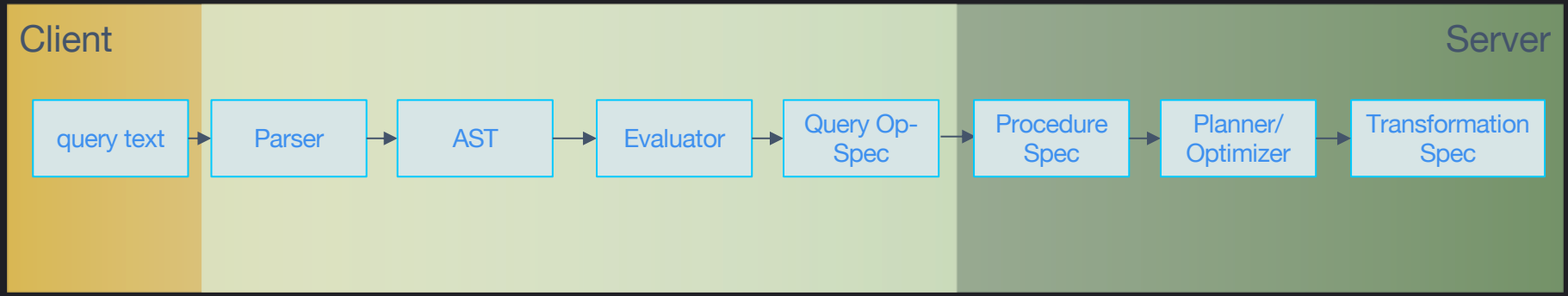
Ø

_time	_m	host	_field	_value
14…1	cpu	serverB	system	2779
GroupKey[cpu, serverB, system]				

[…]

_time	_m	host	_field	_value
14…0	cpu	serverA	system	2342
GroupKey[cpu, serverA, system]				

The Life of a Flux Query



:)

Quiz

1. What's a Bucket?
2. What's the Pipe-forward operator |> for?
3. What's a Table in Flux?
4. What's a Group Key in Flux?

A. List of columns for which every row in a table has the same value

B. A named data source with retention policy

C. Chains operations

D. A part of a result

Quiz answer

1. What's a Bucket?
2. What's the Pipe-forward operator |> for?
3. What's a Table in Flux?
4. What's a Group Key in Flux?

A. List of columns for which every row in a table has the same value

B. A named data source with retention policy

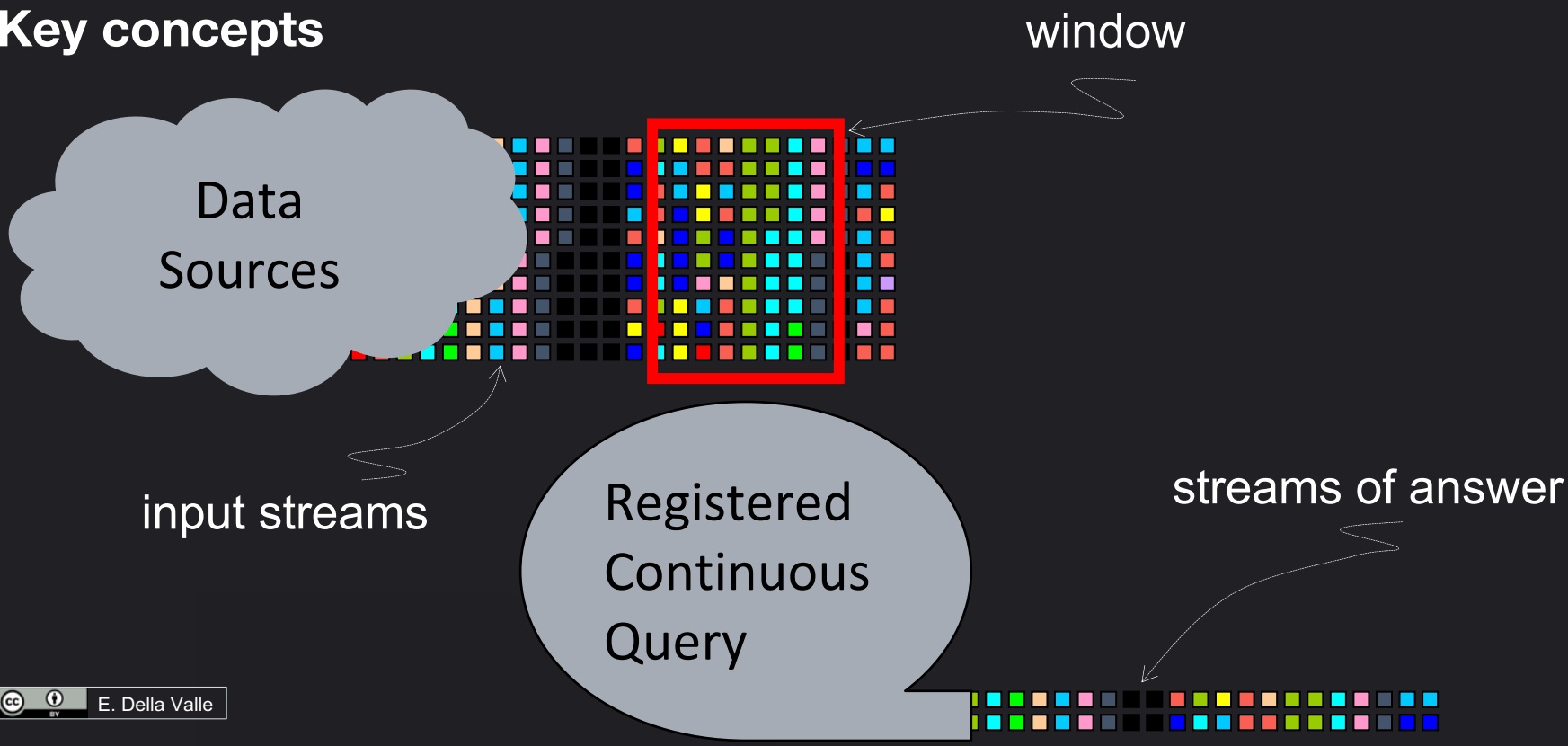
C. Chains operations

D. A part of a result

[1:B, 2:C, 3:D, 4:A]

Exploring flux
Windowing

Key concepts



Windows

- Streams are infinite in nature
- Processing can end only if the input is finite
- A **window operator** selects a finite portion of a stream
  - **Time range** is a window operator whose input is a time interval specified either stating an absolute start and end time

```
range(start: 2019-10-01T00:00:00Z,
      stop: 2019-10-01T00:05:00Z)
```

  - or stating a relative time period relative

```
range(start:-2h, stop: -1h)
range(start:-1h)
```

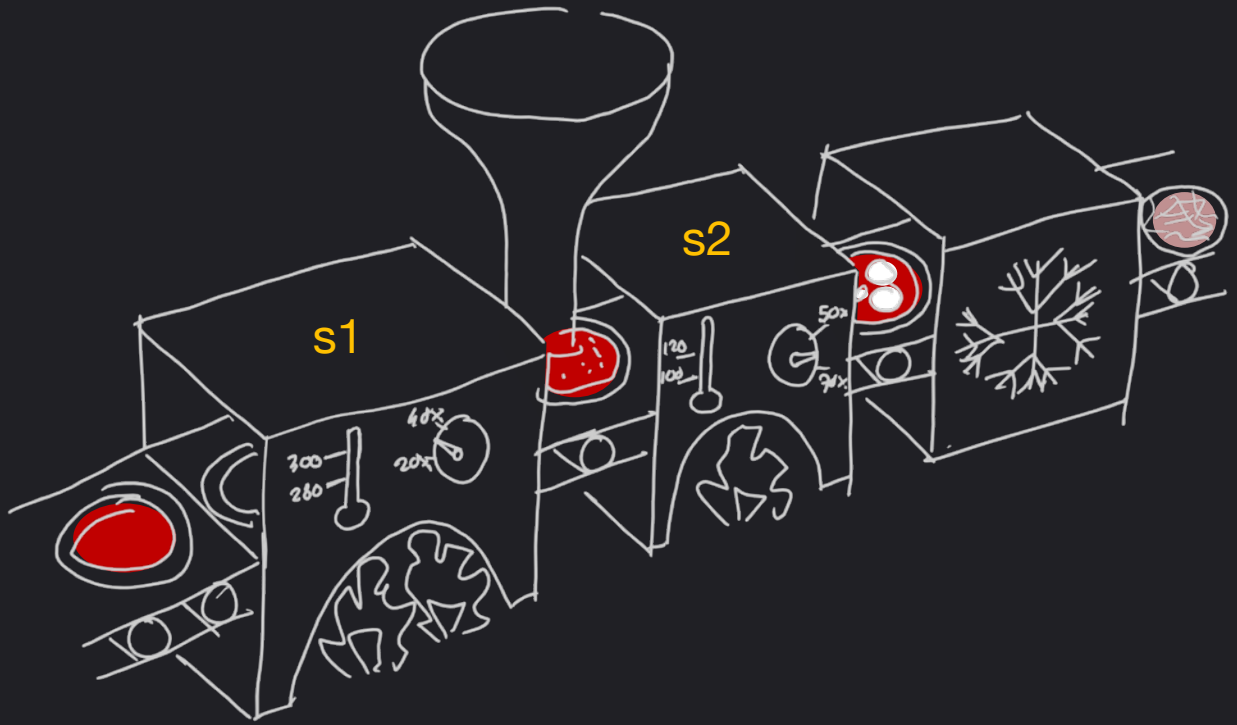
Let’s get dirty!

2

Continuous Linear Pizza Oven

Learning goals:

- Data source
- Window
- Tables



Task

Extract all the measurements in a given range

- Absolute: from 2020-04-30T10:00:00Z to 2020-04-30T10:05:00Z
- Relative: in the last 24 hours

Take home message

The range() function **extracts** from the named bucket all the possible **tables in the given time range**

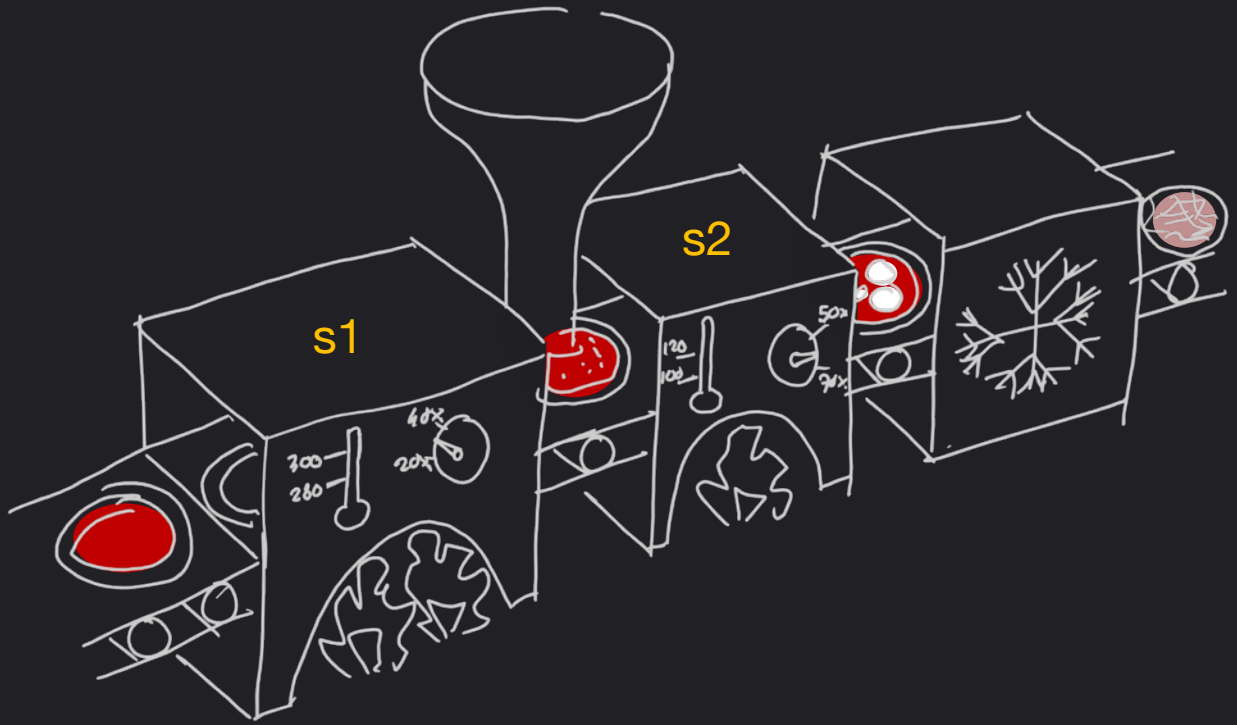
Let’s get dirty!

3

Continuous Linear Pizza Oven

Learning goals:

- Filter by tag



Task

Extract the temperature data from the cooking base area (sensor S1)

Take home message

The filter() function **applied to tags selects** only the **table(s)** of your interest from all the possible tables

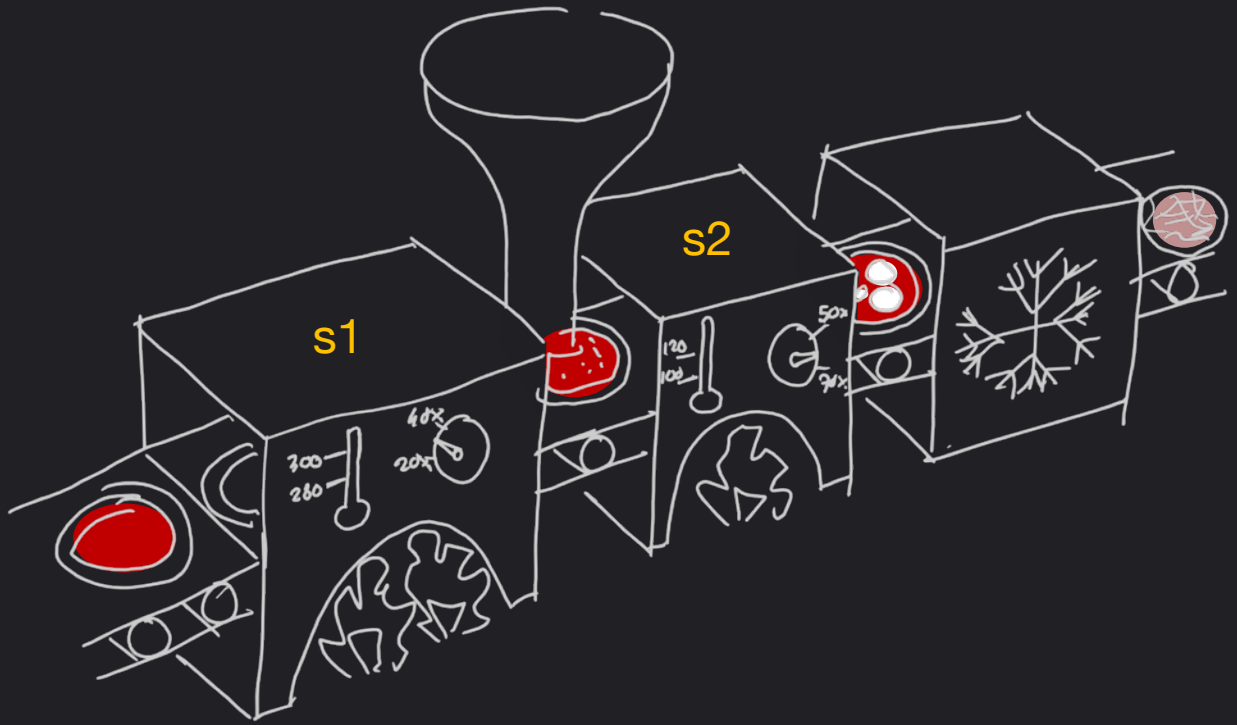
Let’s get dirty!

4

Continuous Linear Pizza Oven

Learning goals:

- Filter by value



Task

Extract the measurements from the cooking base area (sensor S1) with a temperature under 300°

Take home message

The filter() function **applied to values selects** only the **row(s)** of your interest from all the possible tables

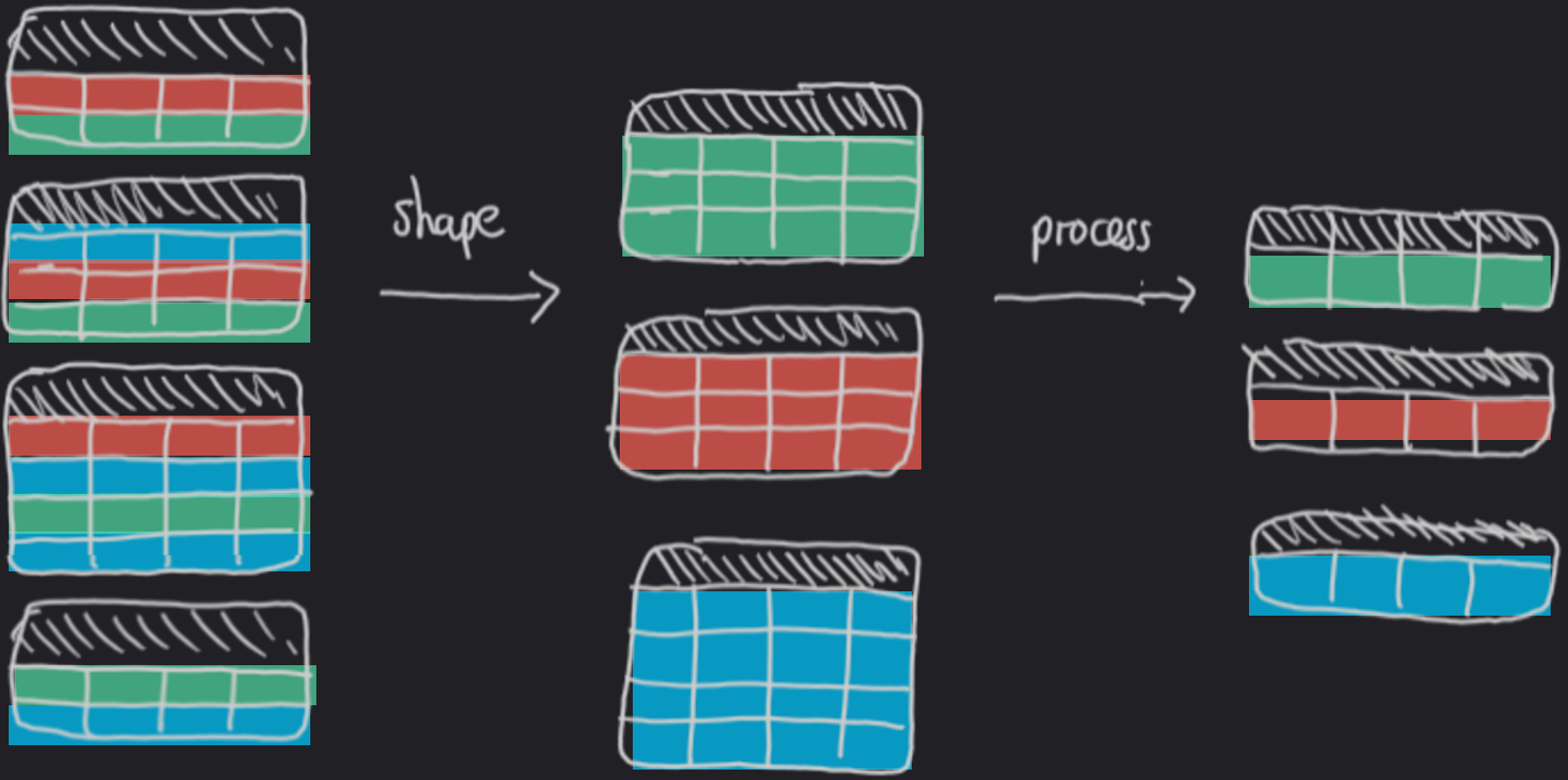
Exploring flux

Shaping and Processing

All Flux Language Elements

- Anatomy of a Flux query:
  - data source
  - time range
  - data filters
  - shaping
  - processing

```
from(bucket:"telegraf/autogen")
  |> range(start:-1h)
  |> filter(fn: (r) =>
        r._measurement == "cpu" and
        r._field == "system" and
        r._value > 2000.0)
  |> group(columns: ["_field"])
  |> mean()
```


shape
process

Exploring flux

Shaping with group Processing with aggregators and selectors

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

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

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

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

```
group(columns: ["_field"])
```

_time	_m	host	_field	_value
10:45	cpu	serverA	system	1000
11:00	cpu	serverB	system	2000
11:30	cpu	serverB	system	2000
11:45	cpu	serverA	system	3000
GroupKey[cpu, system]				

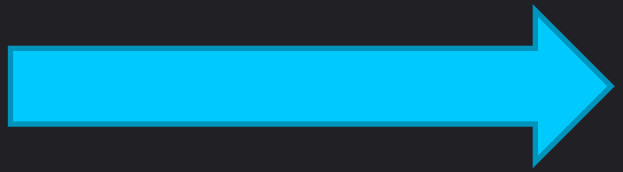
_time	_m	host	_field	_value
10:35	cpu	serverA	idle	100
11:15	cpu	serverB	idle	200
11:50	cpu	serverB	idle	400
GroupKey[cpu, idle]				

Example of aggregator

_time	_m	host	_field	_value
10:45	cpu	serverA	system	1000
11:00	cpu	serverB	system	2000
11:30	cpu	serverB	system	2000
11:45	cpu	serverA	system	3000
GroupKey[cpu, system]				

_time	_m	host	_field	_value
10:35	cpu	serverA	idle	100
11:15	cpu	serverB	idle	200
11:50	cpu	serverB	idle	400
GroupKey[cpu, idle]				

```
|> mean()
```



_m	_field	_value
cpu	system	2000
GroupKey[cpu, system]		

_m	_field	_value
cpu	idle	233,33
GroupKey[cpu, idle]		

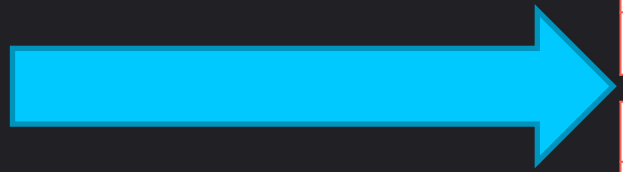
NOTE: time and host columns get deleted

Example of selector

_time	_m	host	_field	_value
10:45	cpu	serverA	system	1000
11:00	cpu	serverB	system	2000
11:30	cpu	serverB	system	2000
11:45	cpu	serverA	system	3000
GroupKey[cpu, system]				

_time	_m	host	_field	_value
10:35	cpu	serverA	idle	100
11:15	cpu	serverB	idle	200
11:50	cpu	serverB	idle	400
GroupKey[cpu, idle]				

```
|> last()
```



_time	_m	host	_field	_value
11:45	cpu	serverA	system	3000
GroupKey[cpu, system]				

_time	_m	host	_field	_value
11:50	cpu	serverB	idle	400
GroupKey[cpu, idle]				

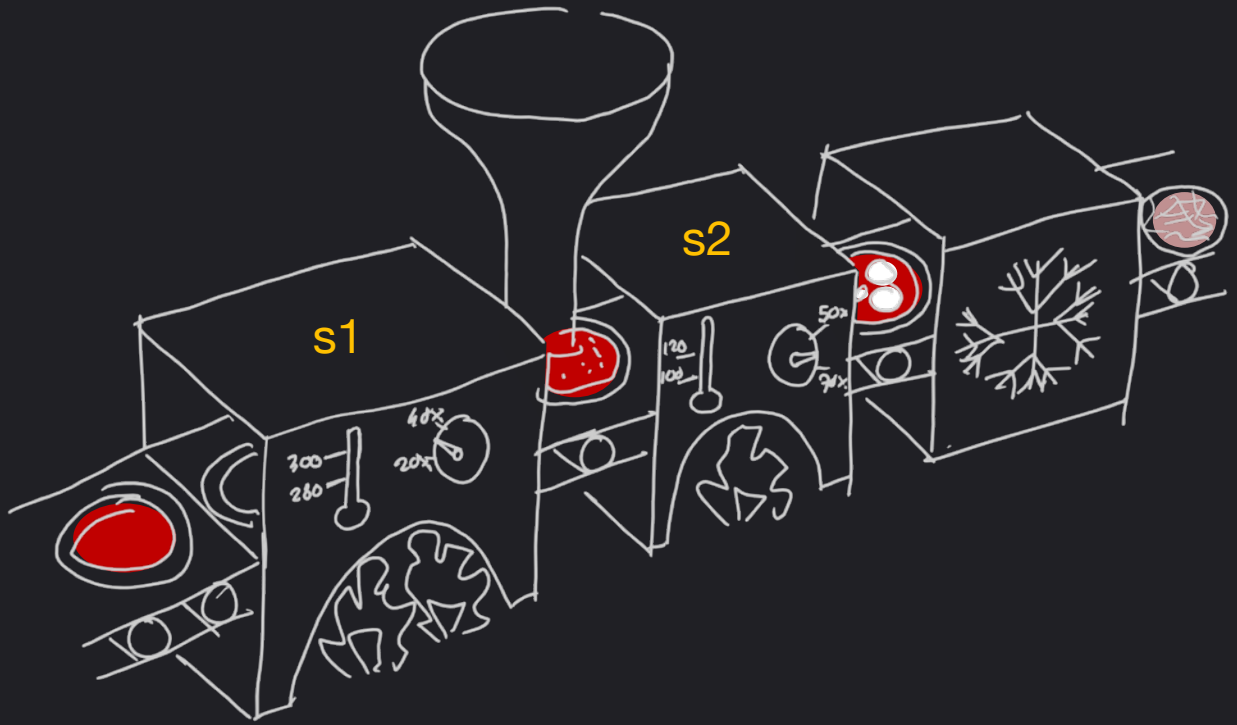
Let’s get dirty!

5

Continuous Linear Pizza Oven

Learning goals:

- Function
  - group()
  - Aggregates
  - Selectors



Task

- **Grouping + Aggregator** - Extract the average temperature and the average humidity along the different stages of the linear pizza oven
- **Selector** - Extract the last humidity observation from the cooking base area

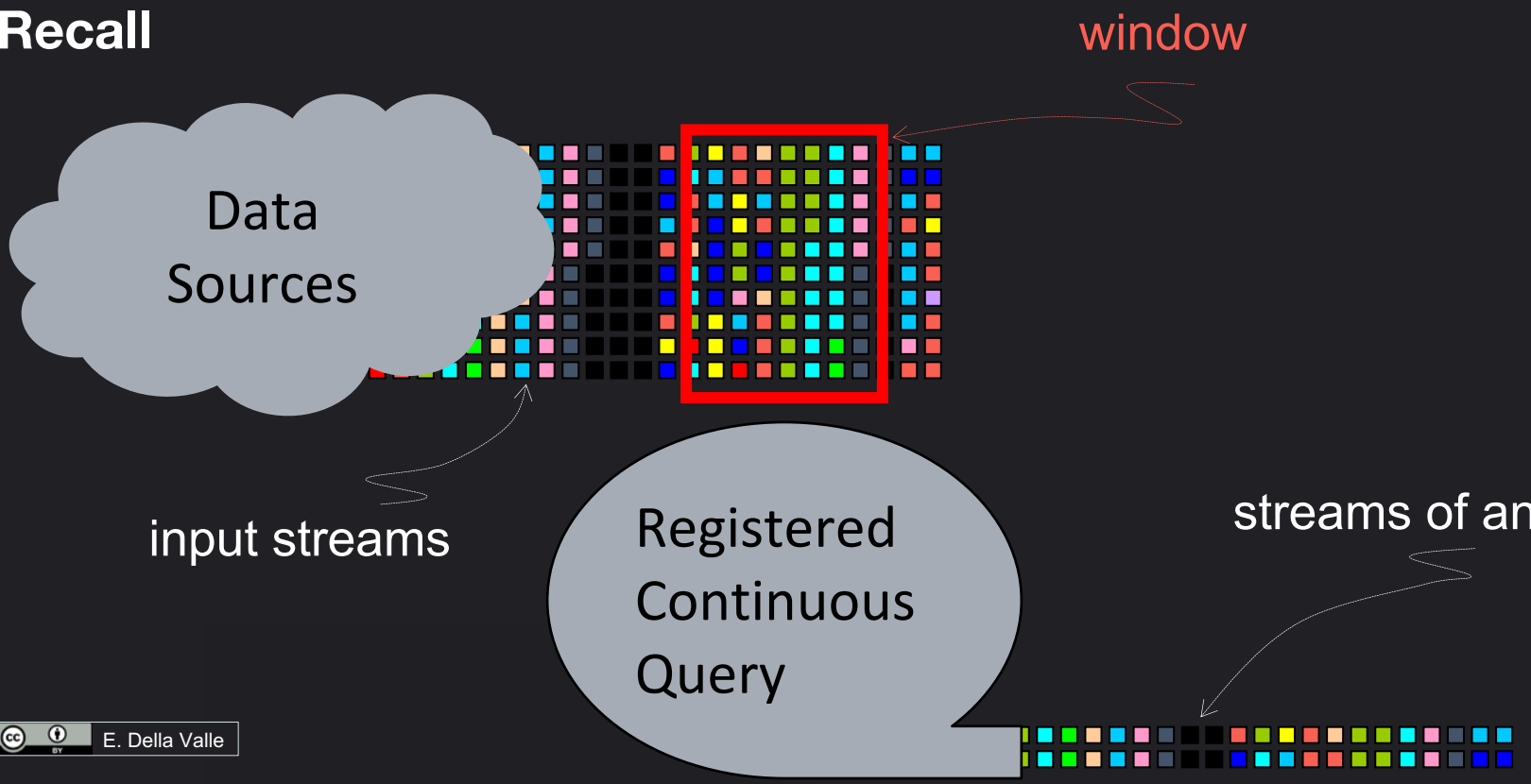
Take home message

- **Grouping** **shape** tables
- **Aggregators** (e.g. mean) **compress** each table **in a row**
- **Selectors** (e.g. last) select **specific row(s)** from each table

Exploring flux

Advanced Windowing

Recall



Windows

- Streams are infinite in nature
- Processing can end only if the input is finite
- A **window operator** selects a finite portion of a stream
- We have seen the time range operator
- Flux includes a window operator that builds table **tumbling** across the results of the time range operator
  - e.g. `aggregateWindow(every: 60m, fn: mean)`

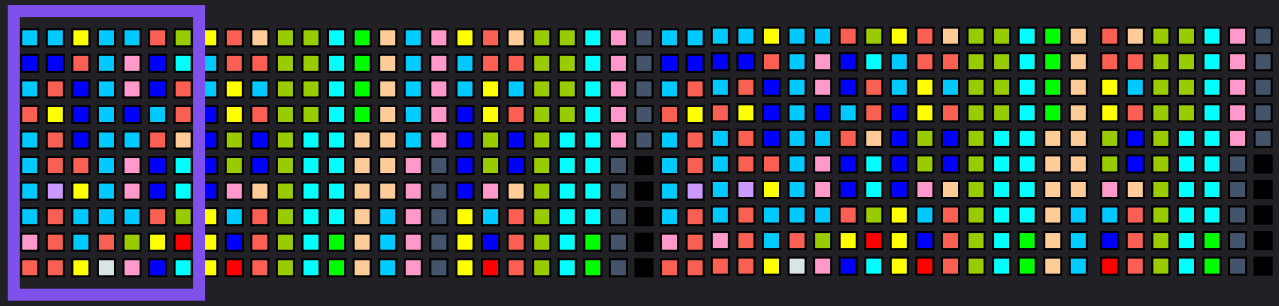
Example of tumbling window

- Anatomy of a Flux query:
  - data source
  - time range
  - data filters
  - shaping
  - processing

```
from(bucket:"telegraf/autogen")
  |> range(start:-1h)
  |> filter(fn: (r) =>
        r._measurement == "cpu" and
        r._field == "system" and
        r._value > 2000.0)
  |> aggregateWindow(every: 60m, fn: mean)
```

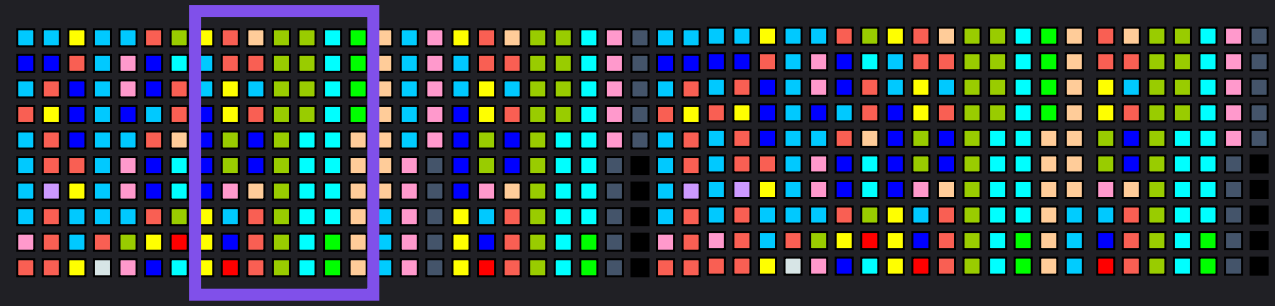
Tumbling?

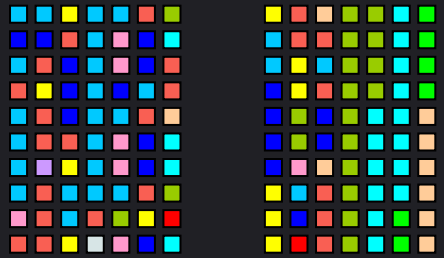
Tumbling windows



[1]

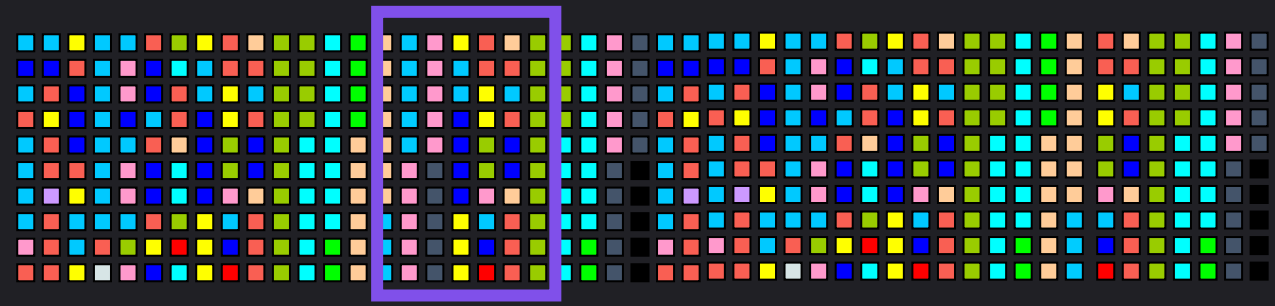
Tumbling windows





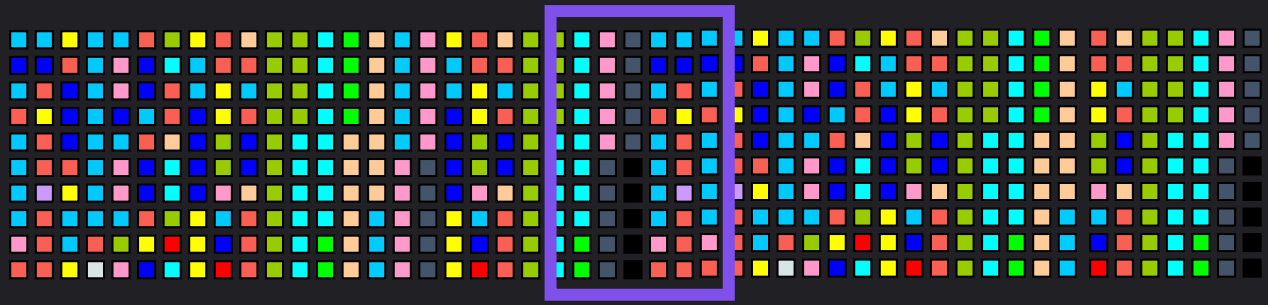
[1] **[2]**

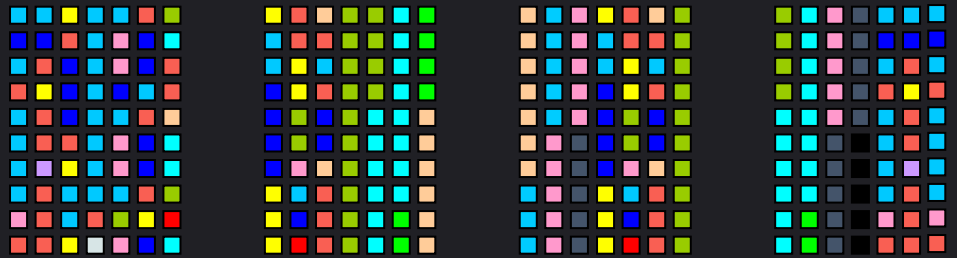
Tumbling windows



[1] [2] **[3]**

Tumbling windows





[1] [2] [3] **[4]**

NOTE: host column gets deleted

_time	_m	host	_field	_value
10:45	cpu	serverA	system	1000
11:00	cpu	serverB	system	2000
11:30	cpu	serverB	system	2000
11:45	cpu	serverA	system	3000
GroupKey[cpu, system]				

```
|> aggregateWindow(
     every: 60m,
     fn: mean)
```

_time	_m	_field	_value
11:00	cpu	system	1000
GroupKey[cpu, system]			

_time	_m	host	_field	_value
10:35	cpu	serverA	idle	100
11:15	cpu	serverB	idle	200
11:50	cpu	serverB	idle	400
GroupKey[cpu, idle]				

NOTE: The start time is not included, the stop time is

_time	_m	_field	_value
11:00	cpu	idle	100
GroupKey[cpu, idle]			

NOTE: host column gets deleted

_time	_m	host	_field	_value
10:45	cpu	serverA	system	1000
11:00	cpu	serverB	system	2000
11:30	cpu	serverB	system	2000
11:45	cpu	serverA	system	3000

GroupKey[cpu, system]

_time	_m	host	_field	_value
10:35	cpu	serverA	idle	100
11:15	cpu	serverB	idle	200
11:50	cpu	serverB	idle	400

GroupKey[cpu, idle]

```
|> aggregateWindow(
    every: 60m,
    fn: mean)
```

NOTE: The start time is not included, the stop time is

_time	_m	_field	_value
11:00	cpu	system	1000

GroupKey[cpu, system]

_time	_m	_field	_value
12:00	cpu	system	2333.3

GroupKey[cpu, system]

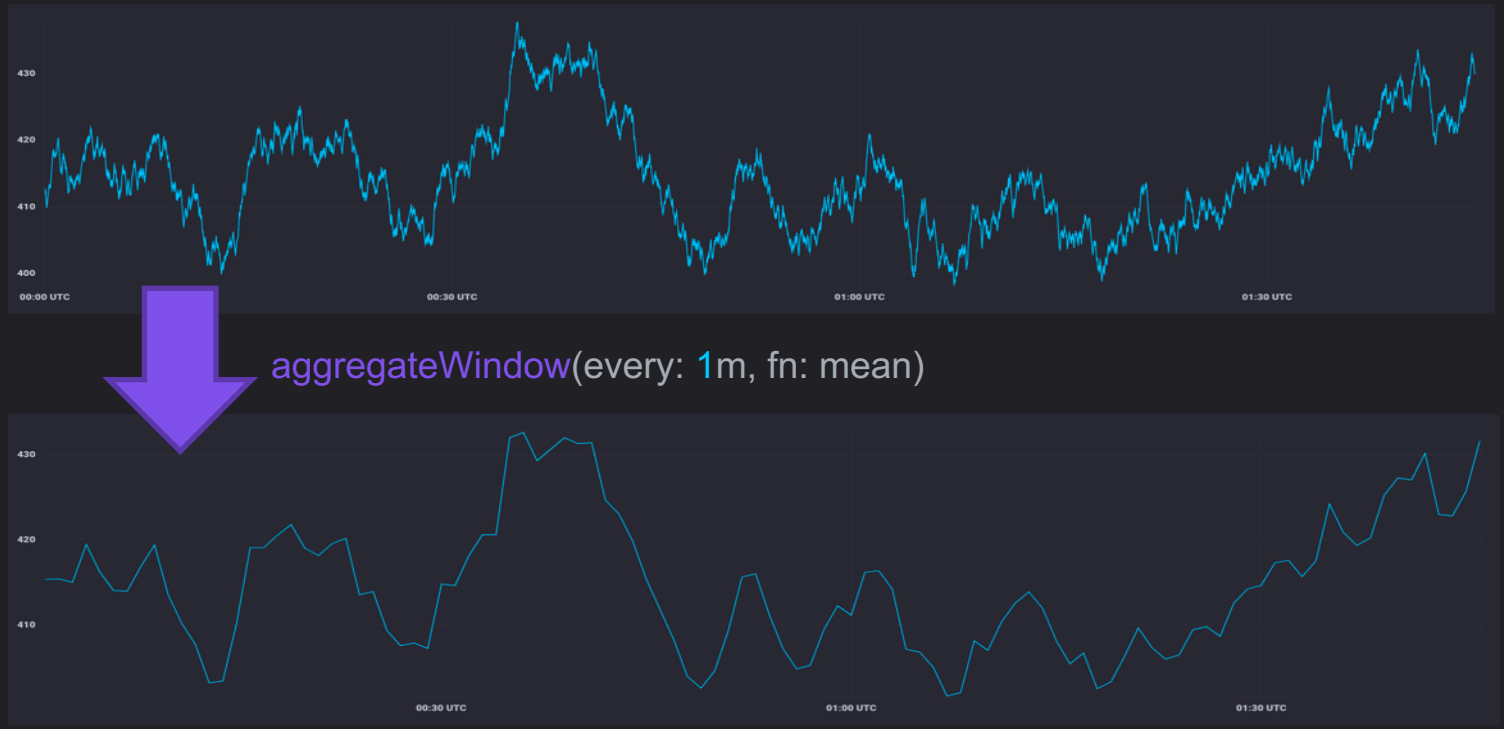
_time	_m	_field	_value
11:00	cpu	idle	100

GroupKey[cpu, idle]

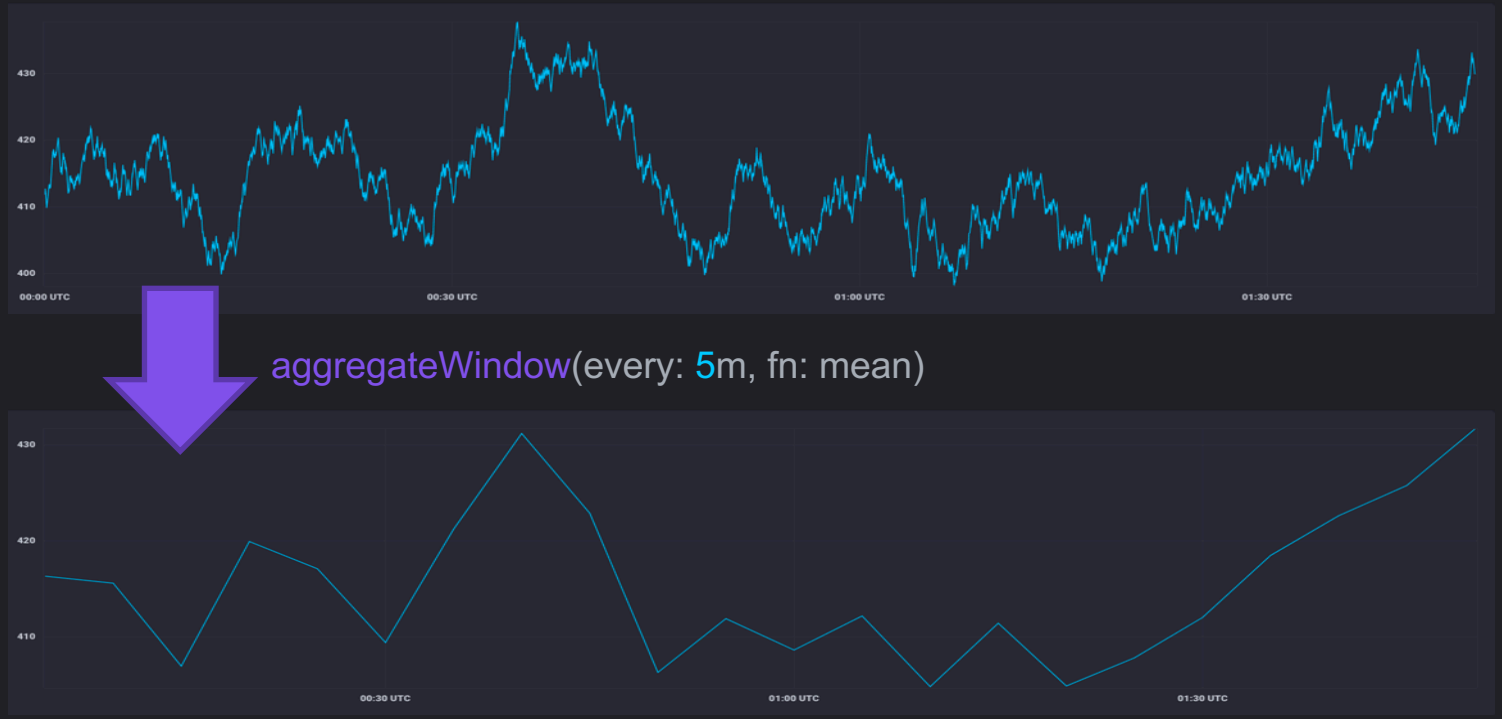
_time	_m	_field	_value
12:00	cpu	idle	300

GroupKey[cpu, idle]

Typical use case: downsampling data



Typical use case: downsampling data (cont.)



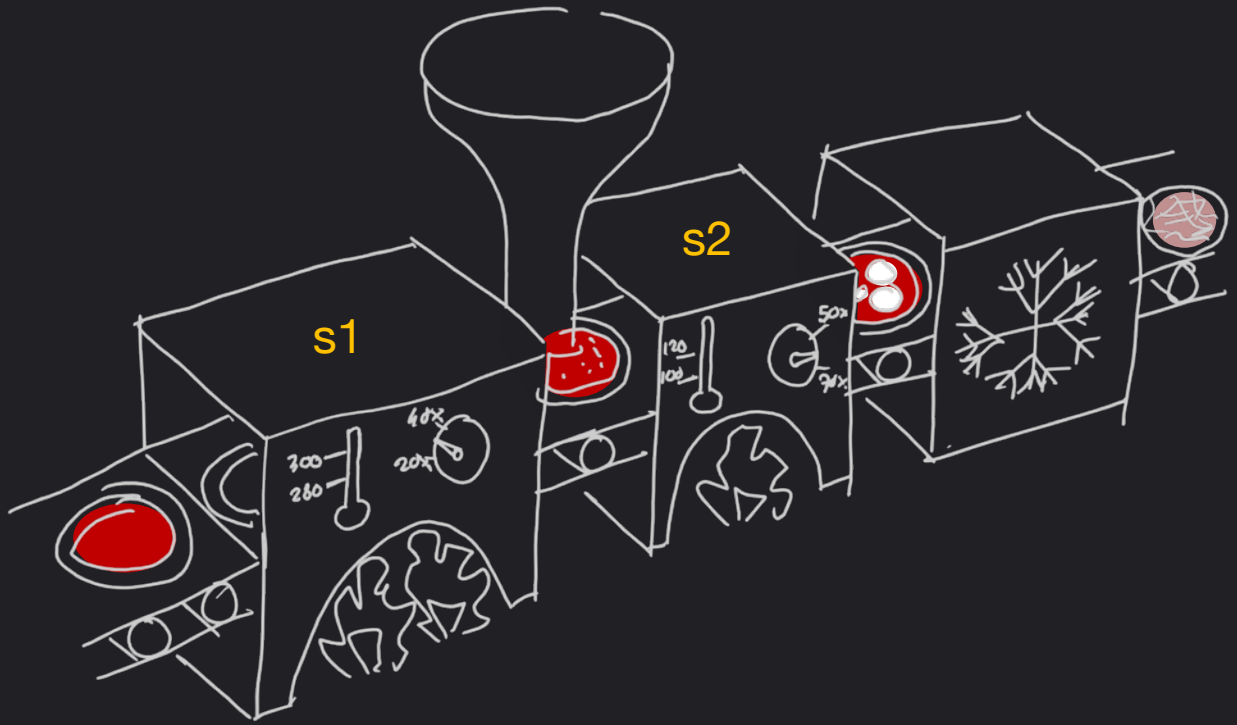
Let’s get dirty!

6

Continuous Linear Pizza Oven

Learning goals:

- aggregateWindow



Task

- Extract the moving average of the temperatures observed in the cooking base area over a window of 2 minutes
- Extract the moving average of the temperatures observed by S2 over a window of 3 minutes

Quiz

- True or false
  - you can omit the range clause – ?
  - filtering can change the number of tables in the result – ?
  - shaping always changes the number of tables in the result – ?
  - processing can change the number of tables in the result – ?
  - aggregateWindow function is just a process step – ?
  - aggregateWindow function eases the downsampling task – ?
  - the parameter createEmpty of the aggregateWindow function can change the number of tables in the result – ?

Quiz answer

- True or false
  - you can omit the range clause – F
  - filtering can change the number of tables in the result – T (empty tables)
  - shaping always changes the number of tables in the result – F
  - processing can change the number of tables in the result – F
  - aggregateWindow function is just a process step – F
  - aggregateWindow function eases the downsampling task – T
  - the parameter createEmpty of the aggregateWindow function can change the number of tables in the result – T

Basic Data Analysis

Emanuele Della Valle Prof. @ Politecnico di Milano & Partner @ Quantia Consulting
Marco Balduini Founder & CEO @ Quantia Consulting
Riccardo Tommasini Prof. @ University of Tartu