Handling late data
How to make right choice?

Bulanova Darya

Data Engineer

- 10+ in IT
- .Net developer
- Data Engineer @ Dodo Engineering
- SmartData conf ПК
- DE or DIE meetup

linkedin.com/daryabulanova

fb.com/darya.bulanova.5

t.me/daryabulanova

Roadmap

- **Chapter 1: Late Data**
  - Late data problem
  - Data processing patterns
  - Conclusion
- **Chapter 2: Completeness**
  - Tools comparison (Spark Streaming, Apache Beam)
  - Tradeoff: completeness, latency, cost
  - Conclusion

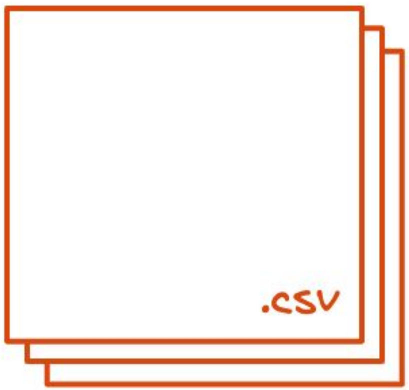
Preface

- **Completeness** - data is considered “complete” when it fulfills expectations of comprehensiveness.
- **Correctness (accuracy)** - the degree to which information accurately reflects an event or object described

Chapter 1: Late Data

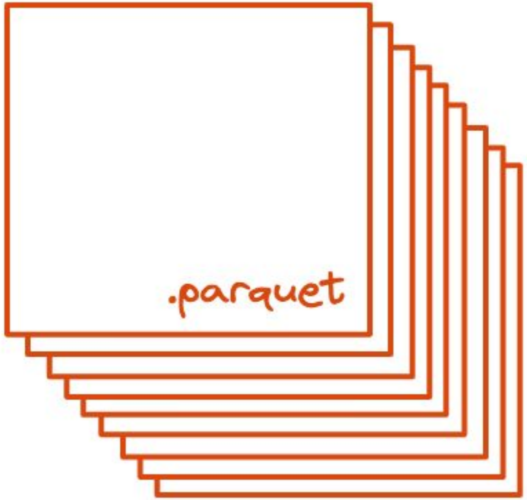
Example: small business

```
+----------+----------+---------------------------+
| Field    | Type     | Description
+----------+----------+---------------------------+
| Id       | int      | номер заказа
| PizzaId  | int      | идентификатор пиццы
| SaleDate | datetime | дата+время продажи заказа
+----------+----------+---------------------------+
```

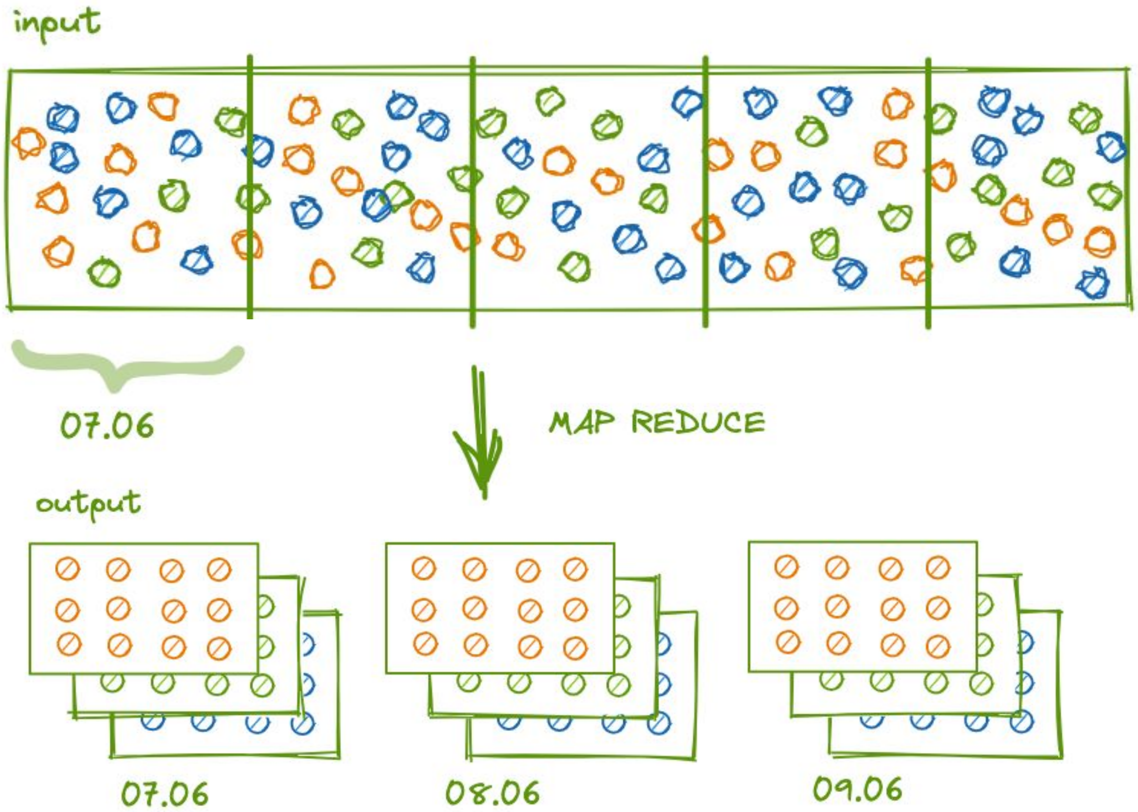


Example: not so small business

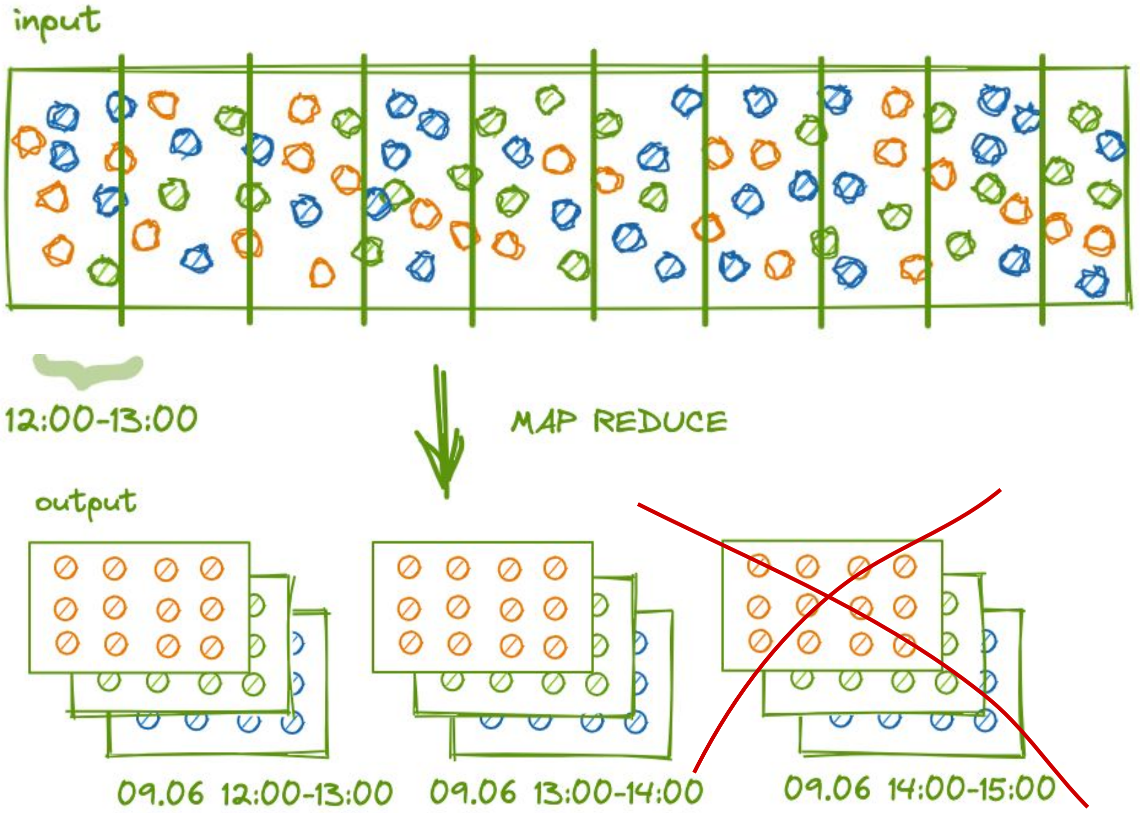
```
+----------+----------+-------------------------+
| Field    | Type     | Description
+----------+----------+-------------------------+
| Id       | int      | номер заказа
| PizzaId  | int      | идентификатор пиццы
| SaleDate | datetime | дата+время продажи заказа
+----------+----------+-------------------------+
```



Example: big business



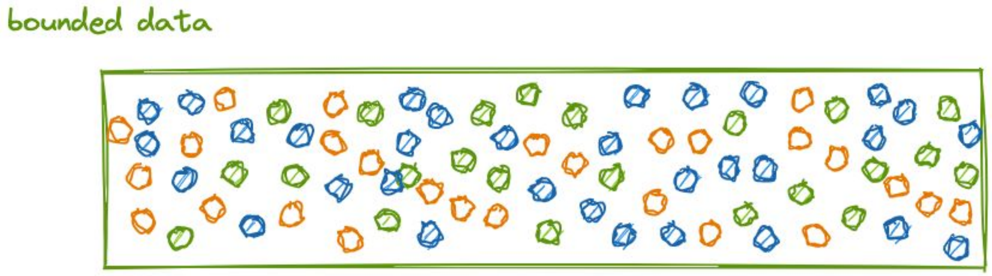
Example: big business



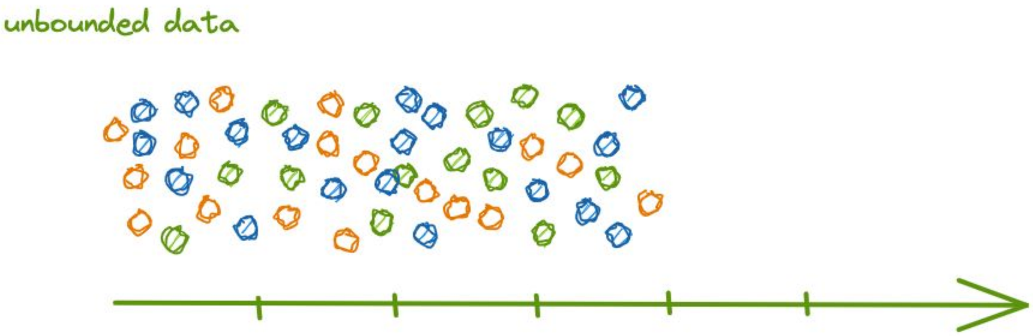
Data is wrong

Fallacy Nº1: Cardinality

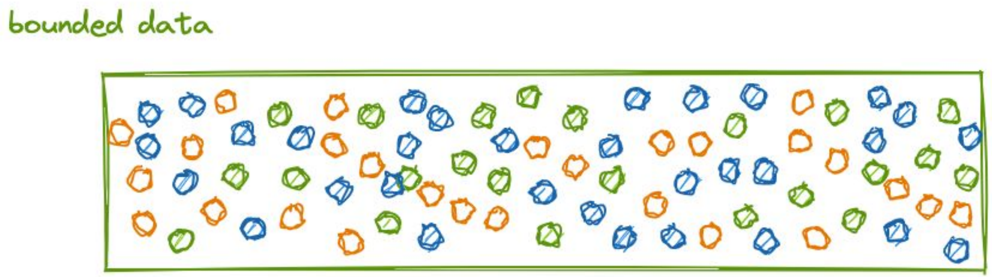
- **Bounded data** - a type of dataset that is **finite** in size



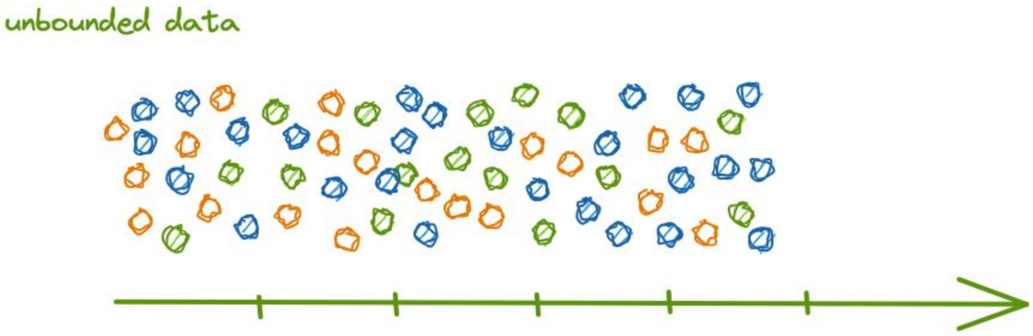
- **Unbounded data** - a type of dataset that is **infinity** in size



Fallacy Nº1: Cardinality



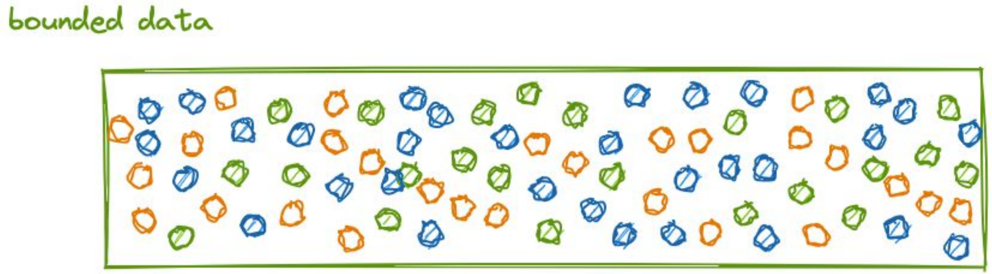
- **Bounded data** - a type of dataset that is **finite** in size



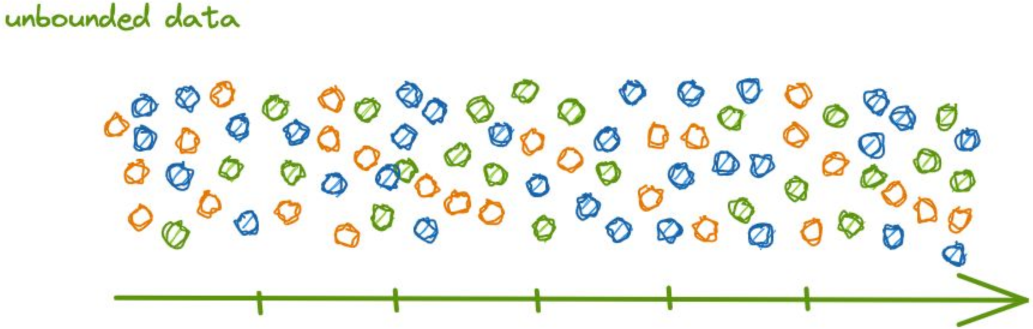
- **Unbounded data** - a type of dataset that is **infinity** in size

Fallacy Nº1: Cardinality

- **Bounded data** - a type of dataset that is **finite** in size

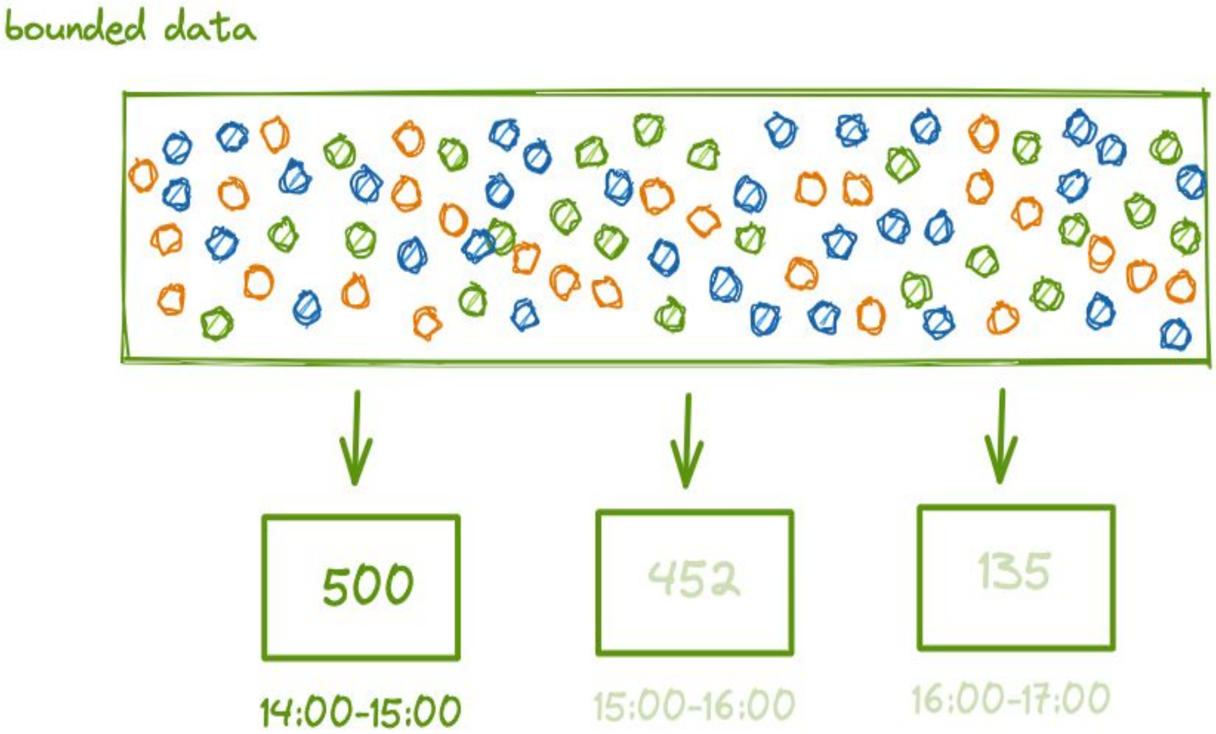


- **Unbounded data** - a type of dataset that is **infinity** in size

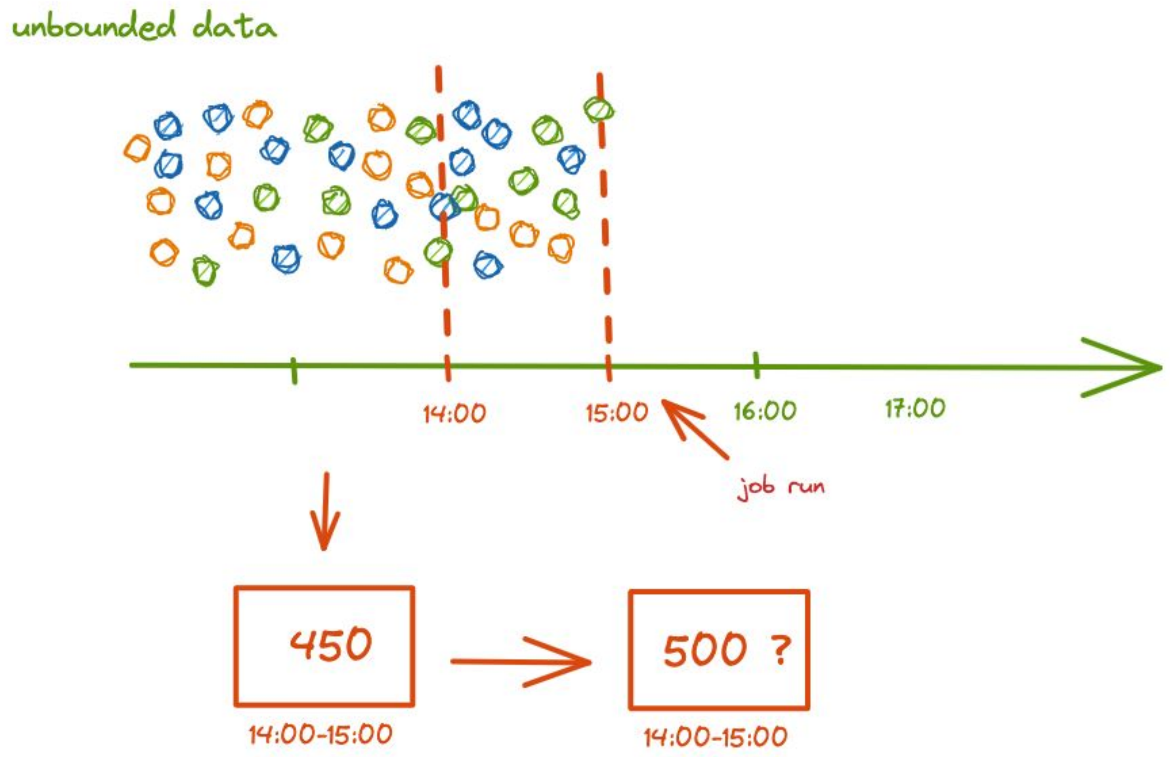


Fallacy Nº1: Cardinality

- **Deterministic**

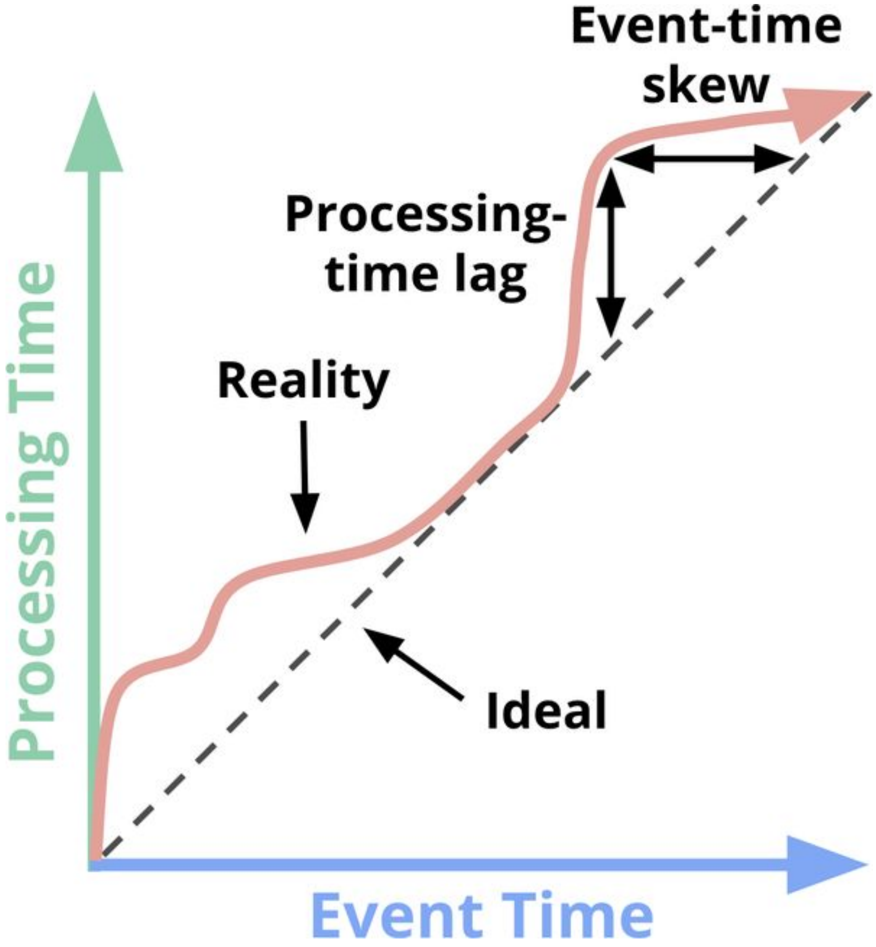


Fallacy №1: Cardinality

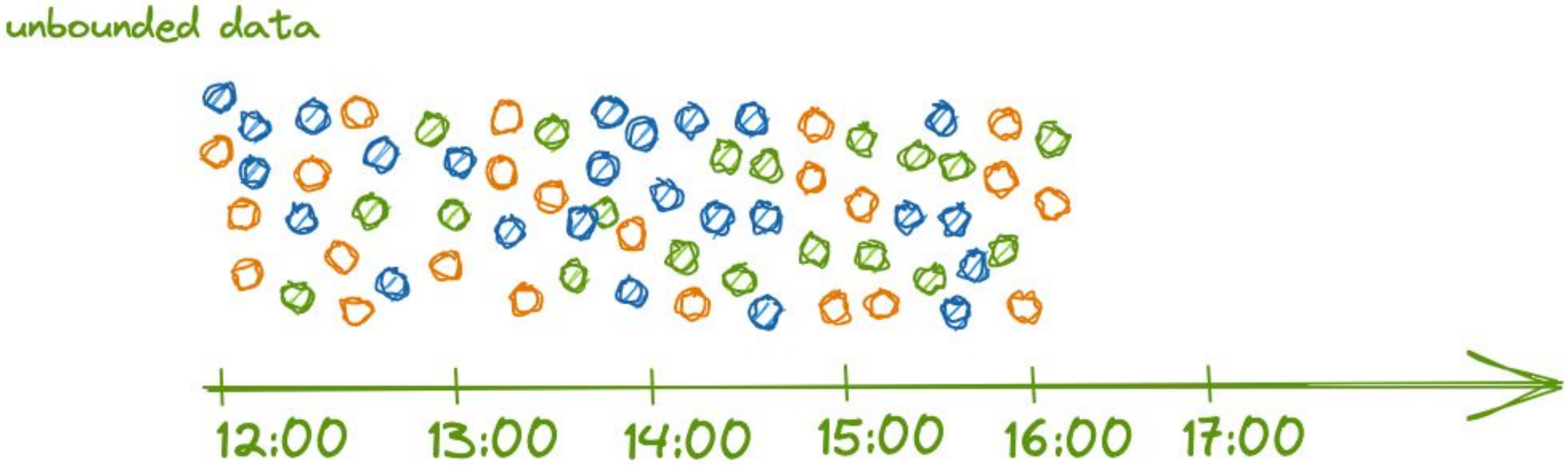


Fallacy №2: Time Domain

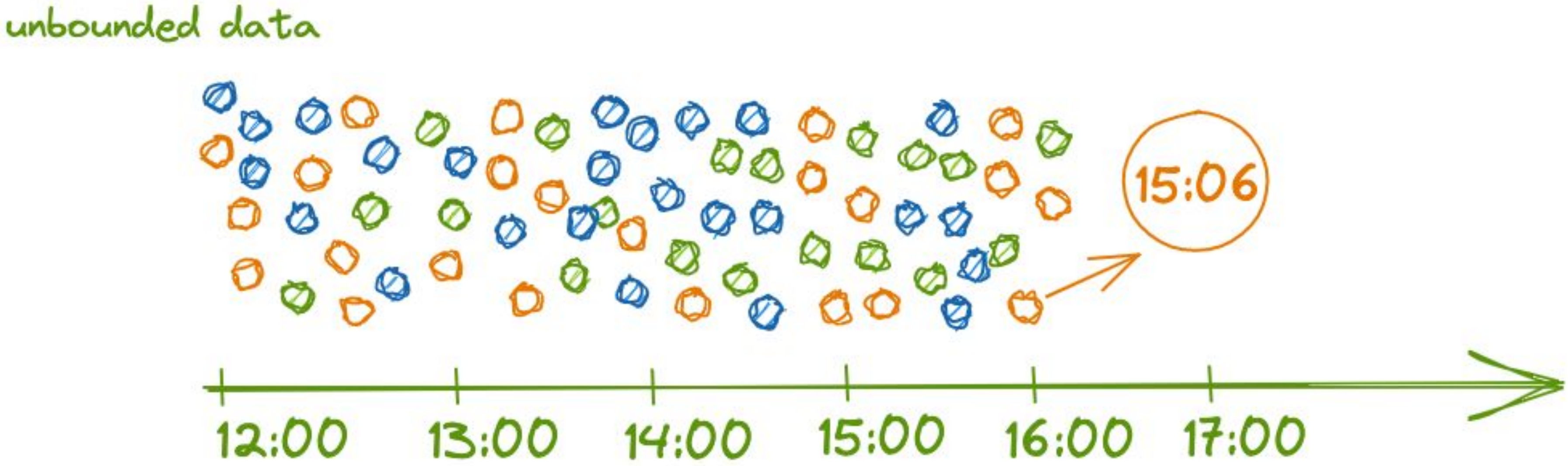
- **Event Time** - the local timestamp assigned to the event by the producer at the time the event occurred.
- **Processing Time** - the **wall-clock** time at which the event has been processed by the consumer.



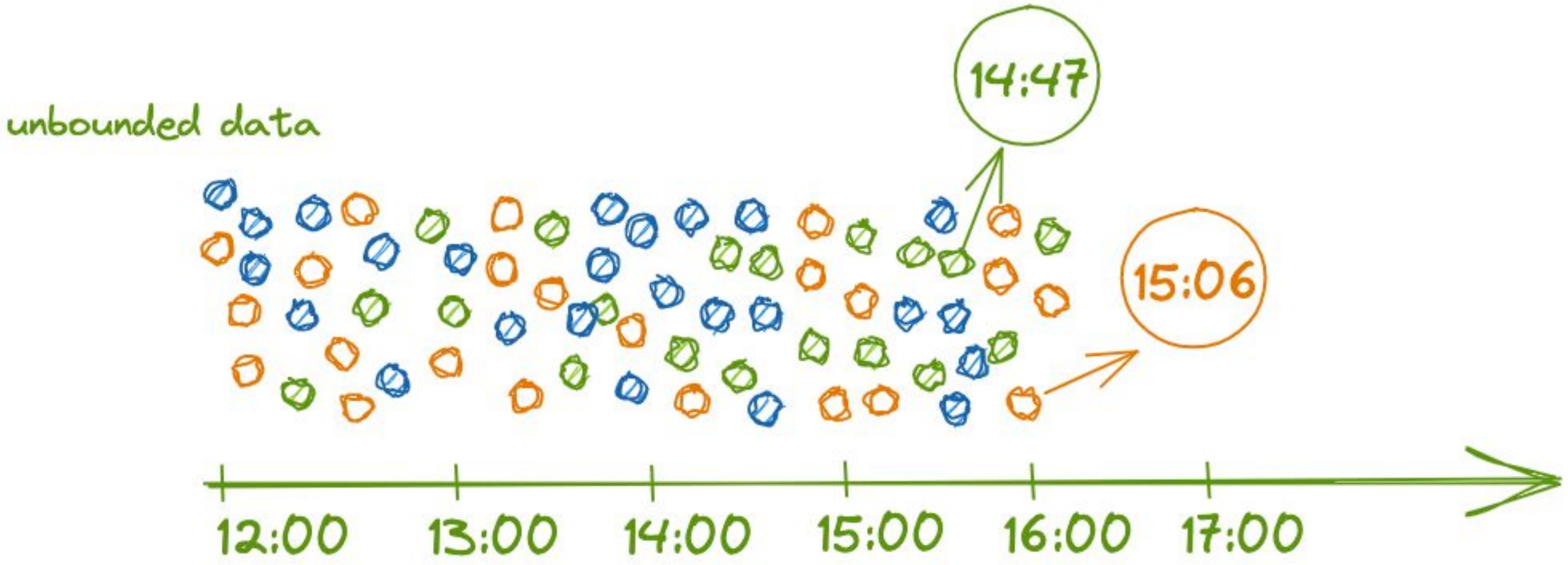
Distrubuted System:
Out-of-order data



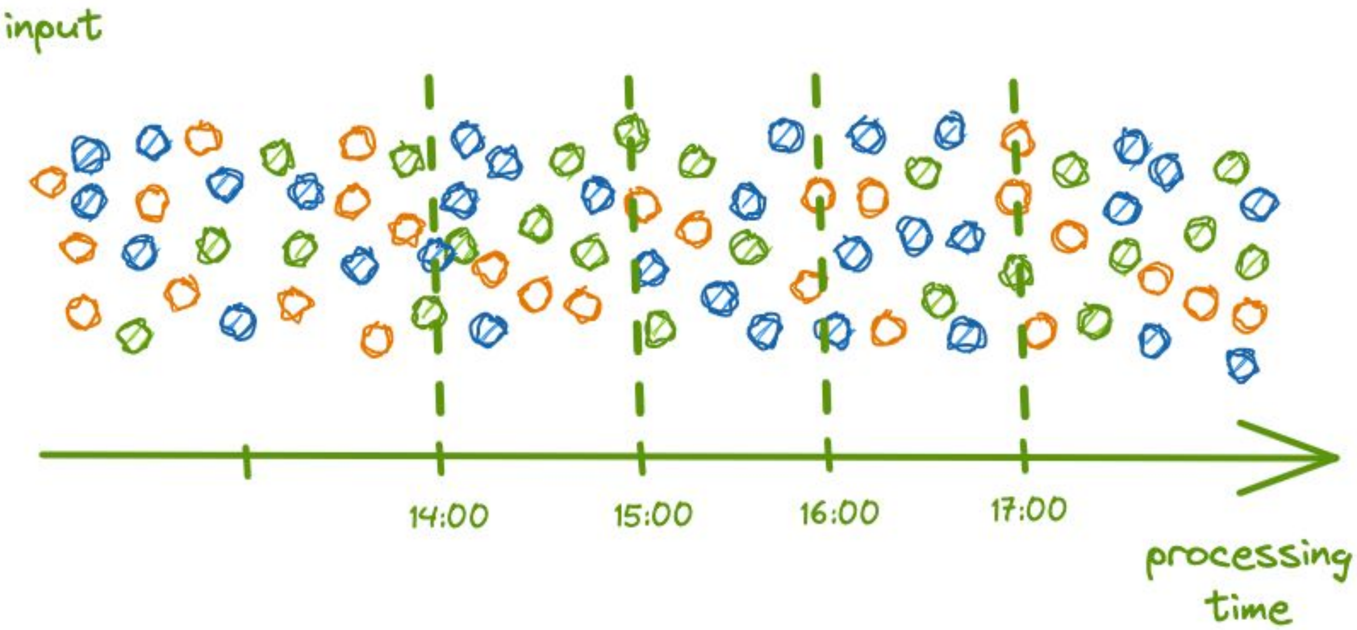
Distrubuted System:
Out-of-order data



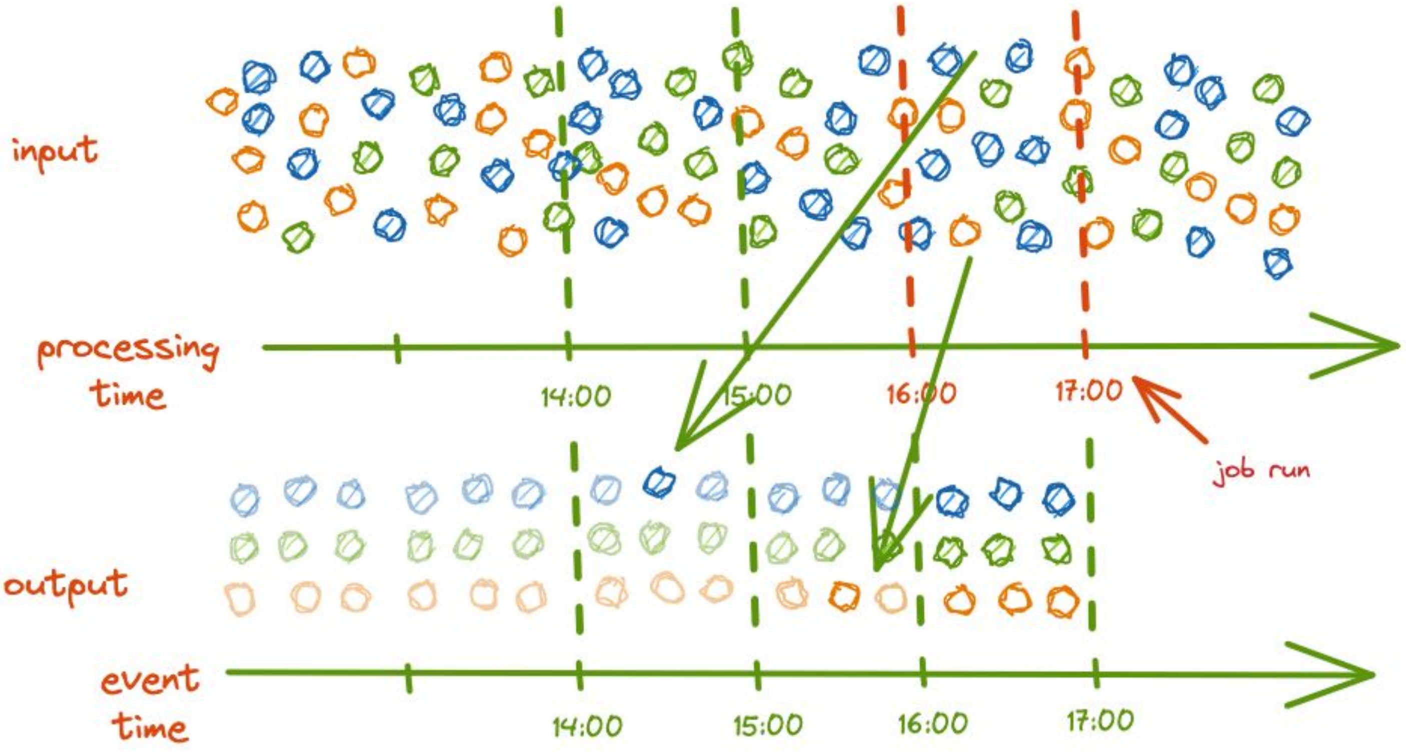
Distrubuted System:
Out-of-order data



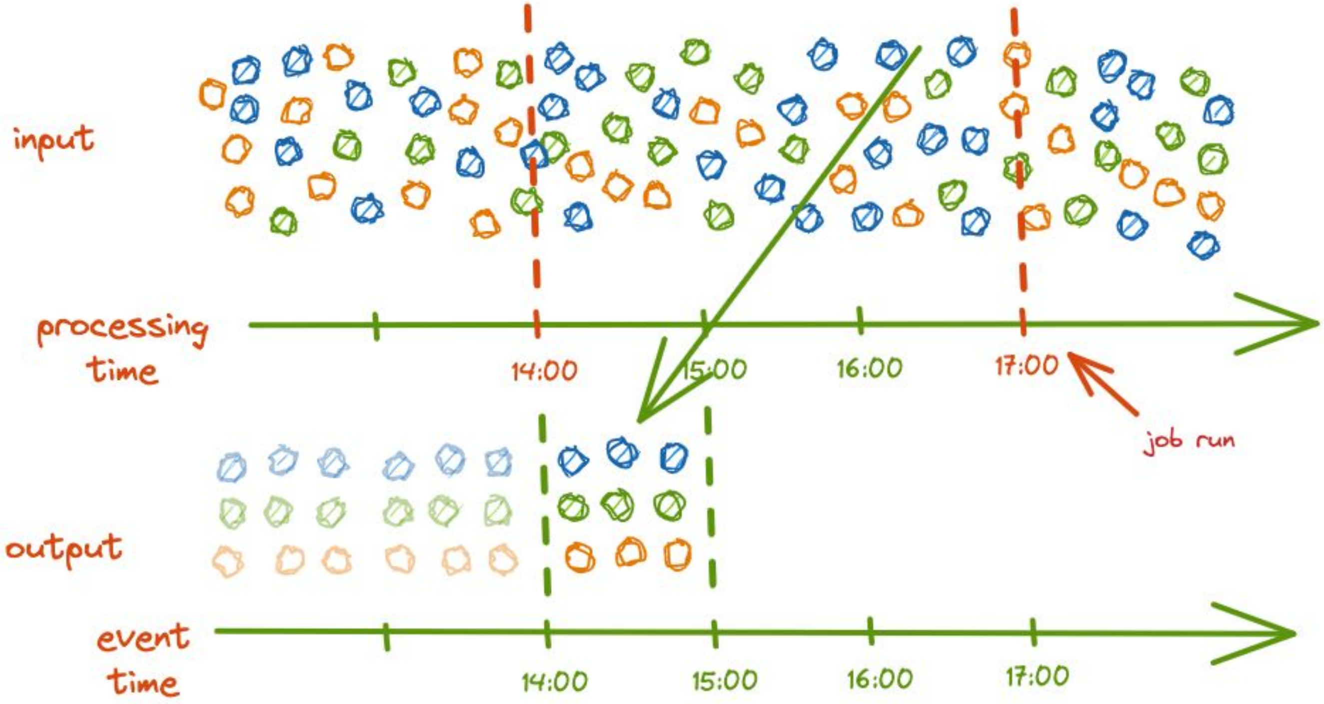
Batch Processing: fixed window by processing time



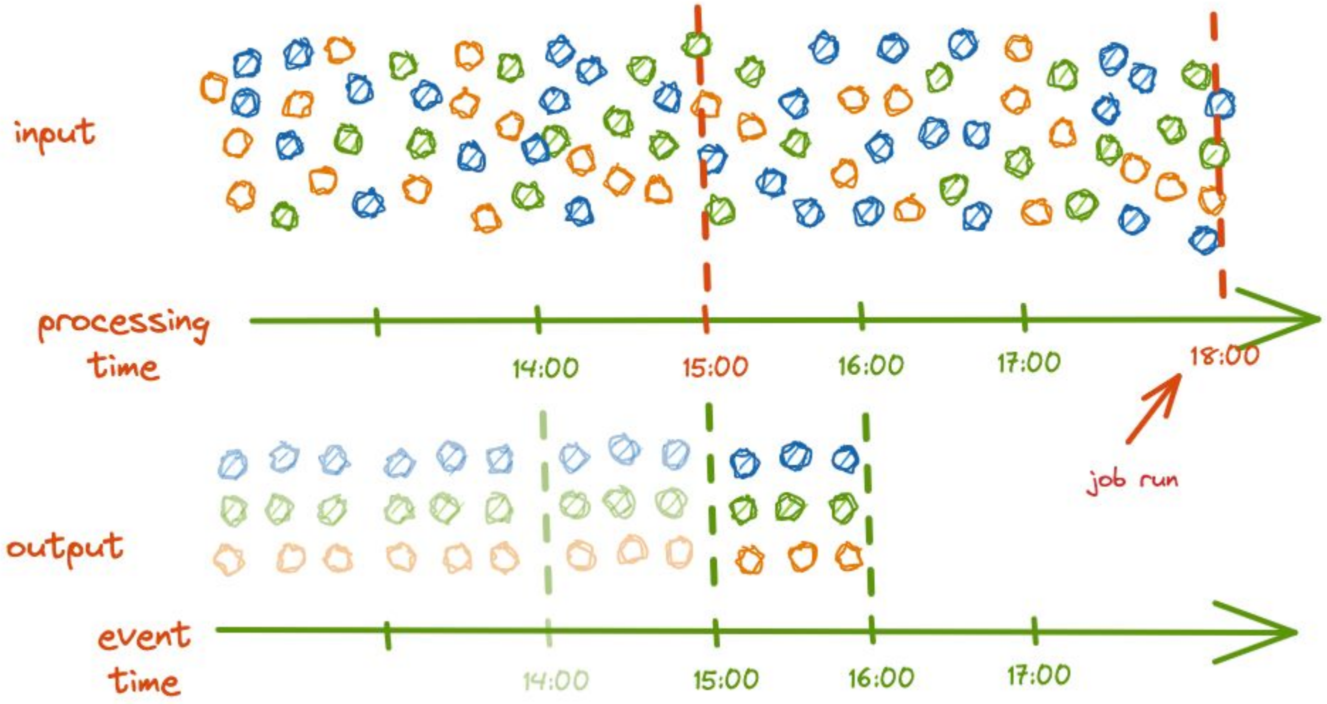
Batch Processing: fixed window by event time



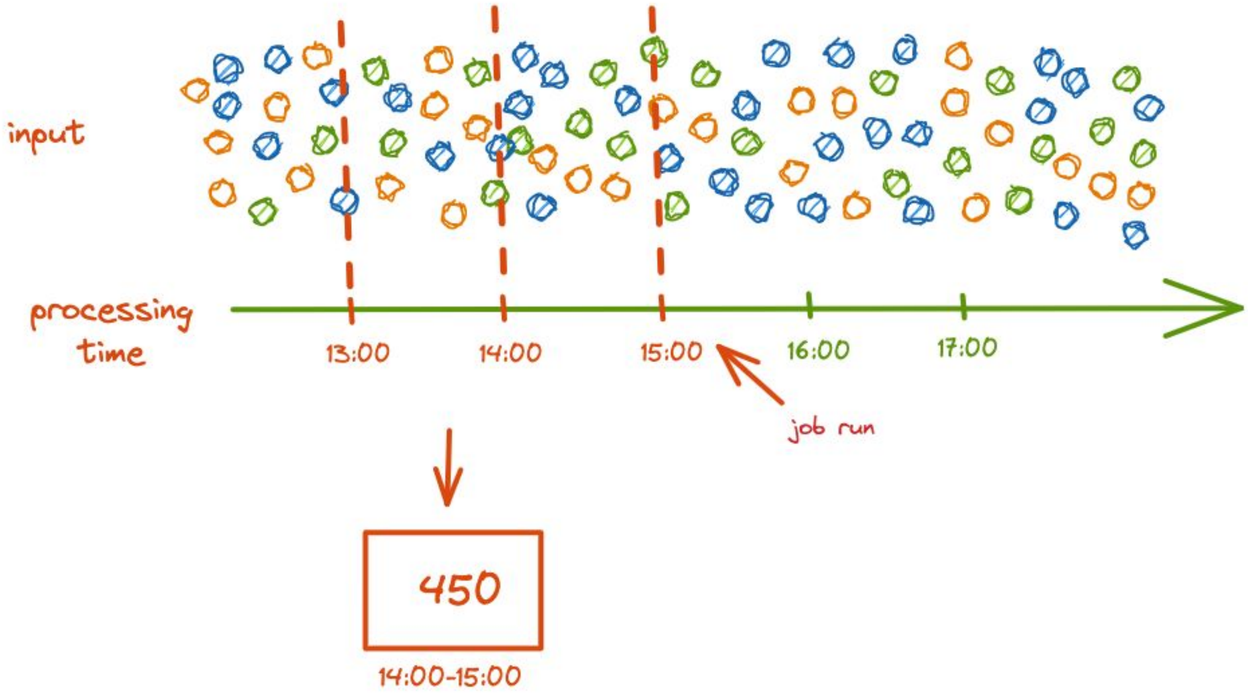
Batch Processing: Delay strategy



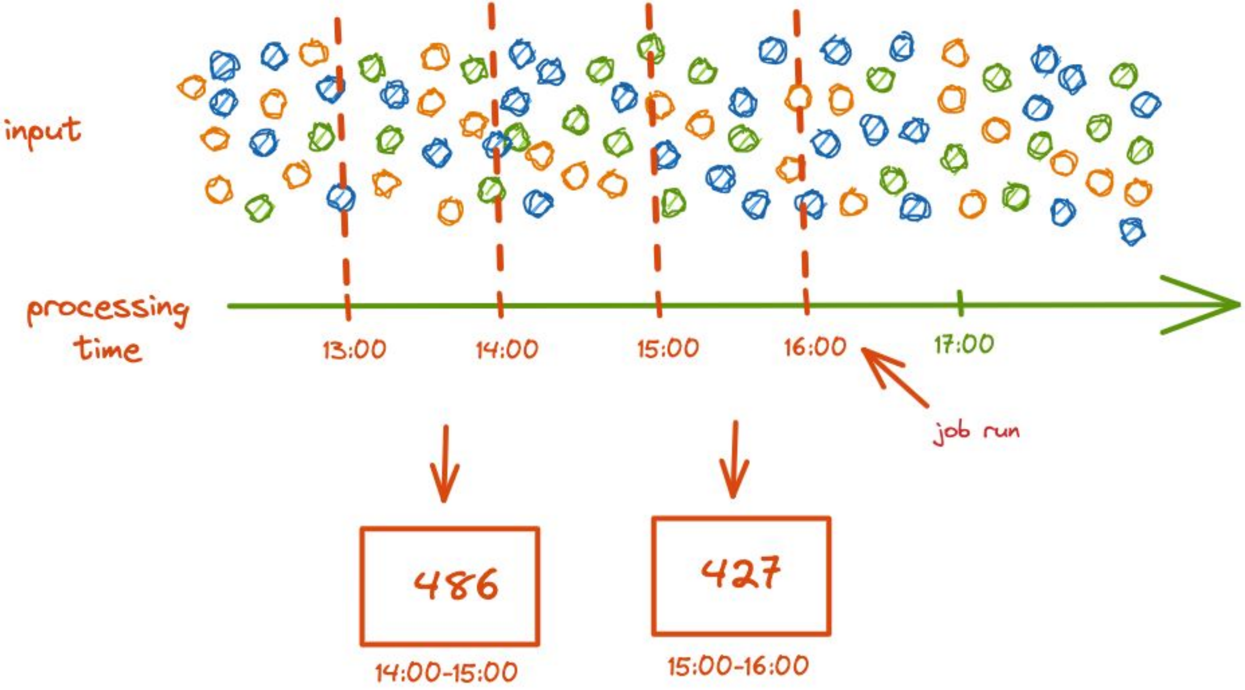
Batch Processing: Delay strategy



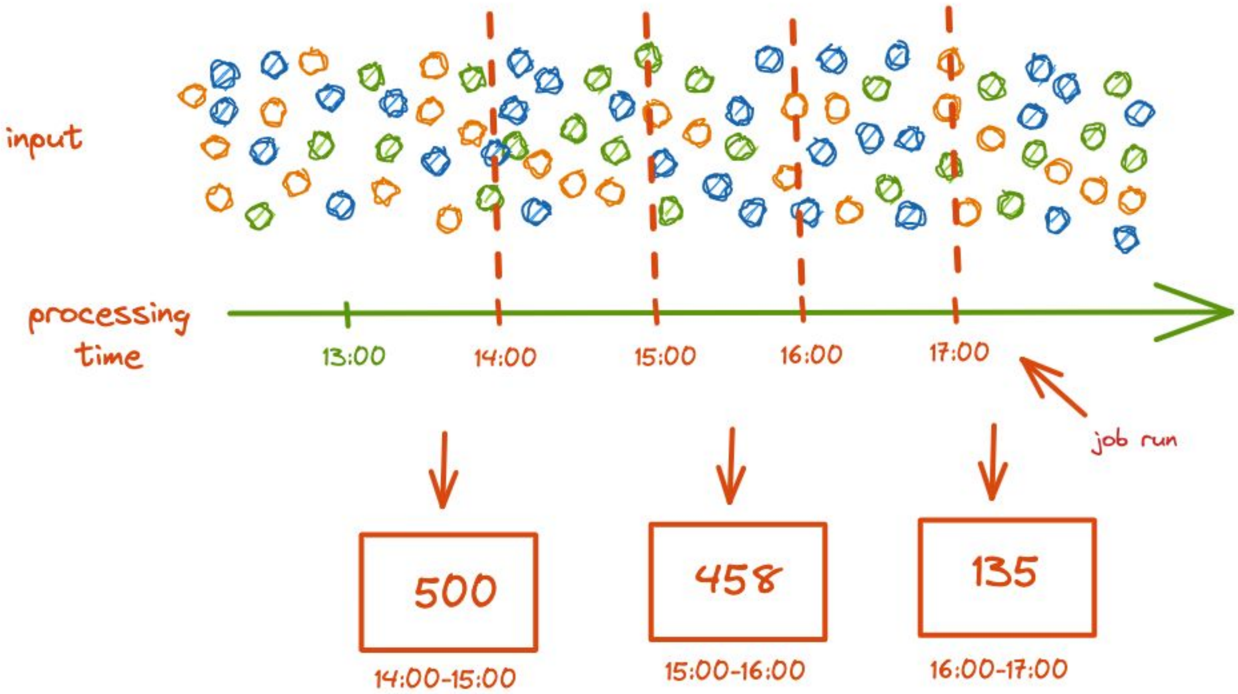
Batch Processing: Repetitive Runs Strategy



Batch Processing: Repetitive Runs Strategy



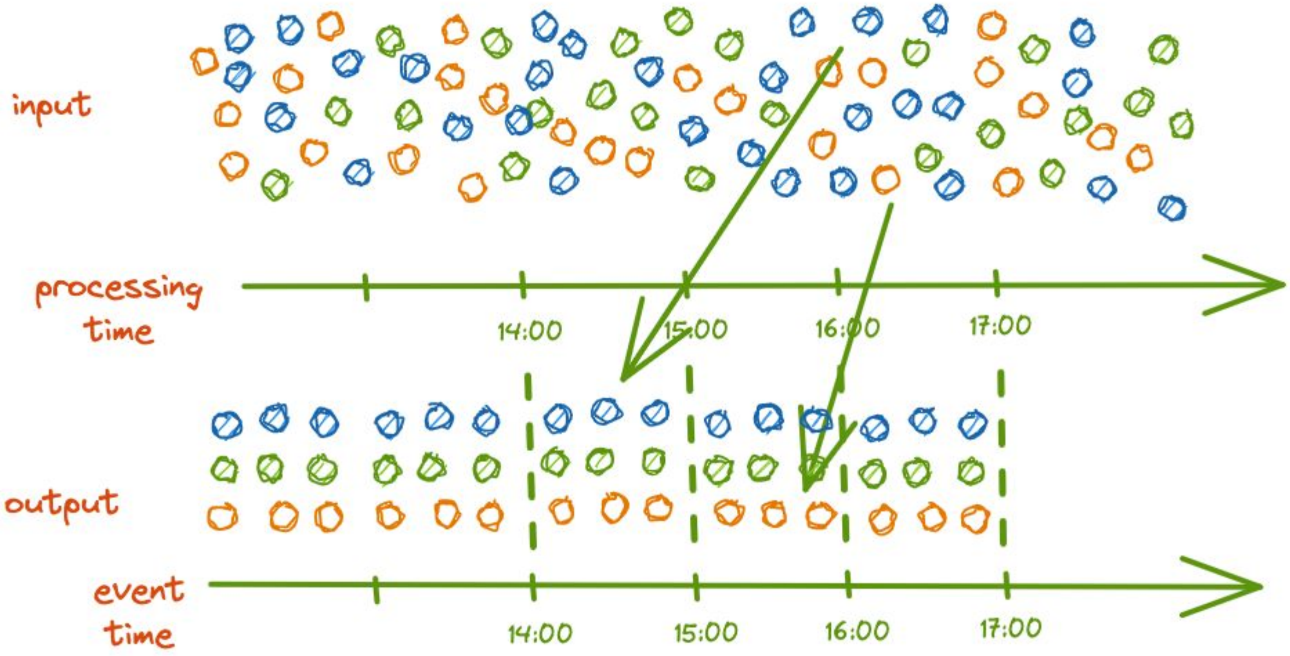
Batch Processing: Repetitive Runs Strategy



Stream processing: primitives

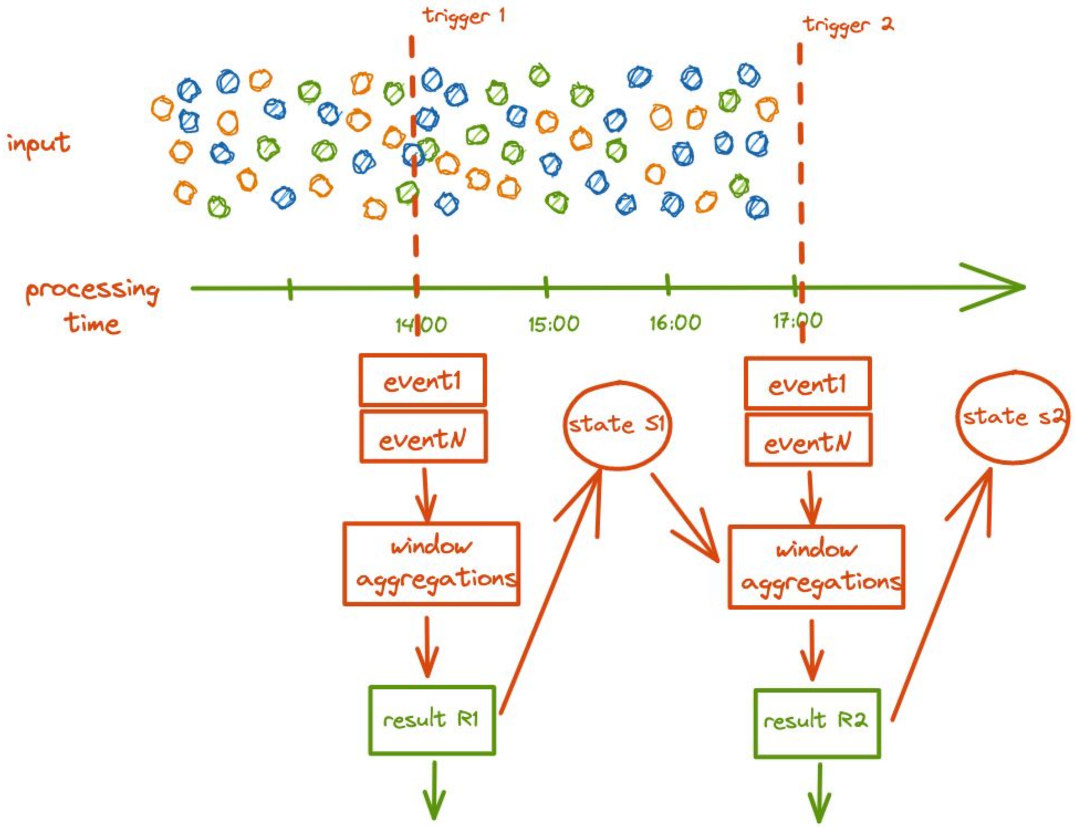
- **Windows** - responsible for correctness
- **Trigger** - when result is materialized
- **Watermarks -** completeness threshold

Stream processing: windows by event time



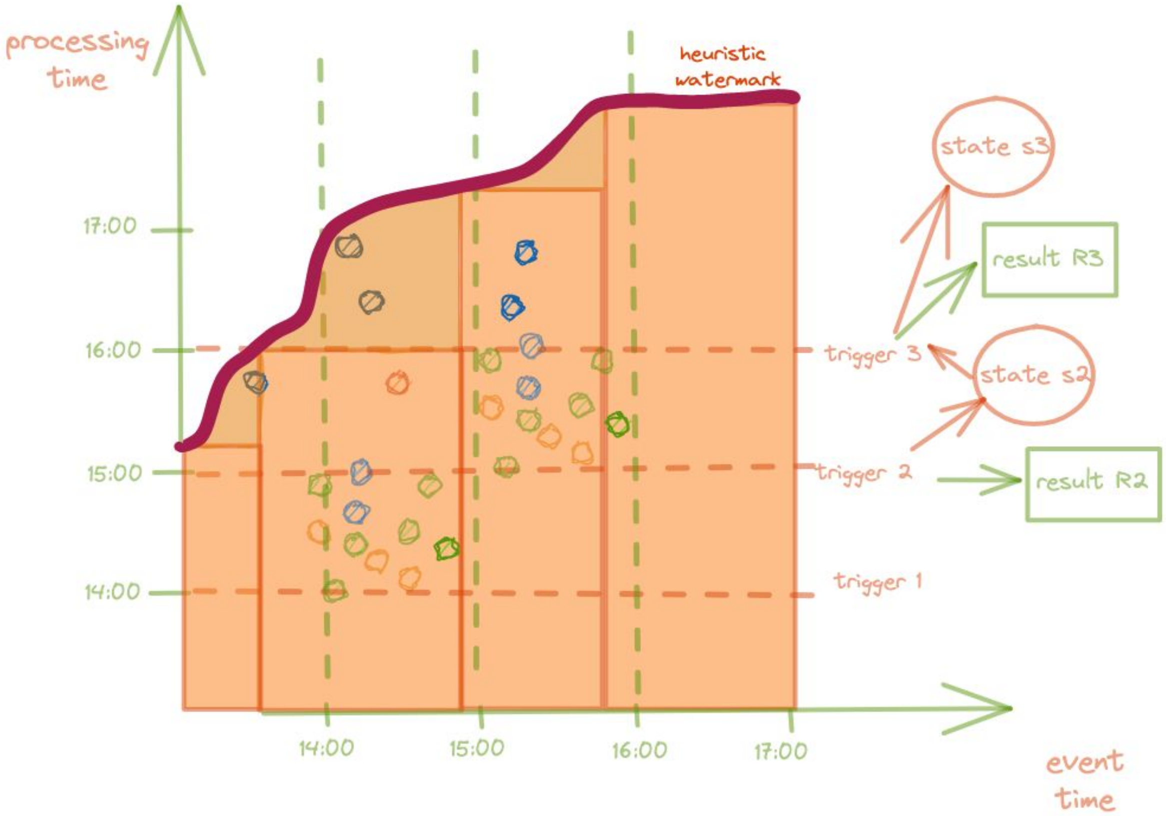
Stream processing: triggers

- **Repetative run**
- **Completeness**



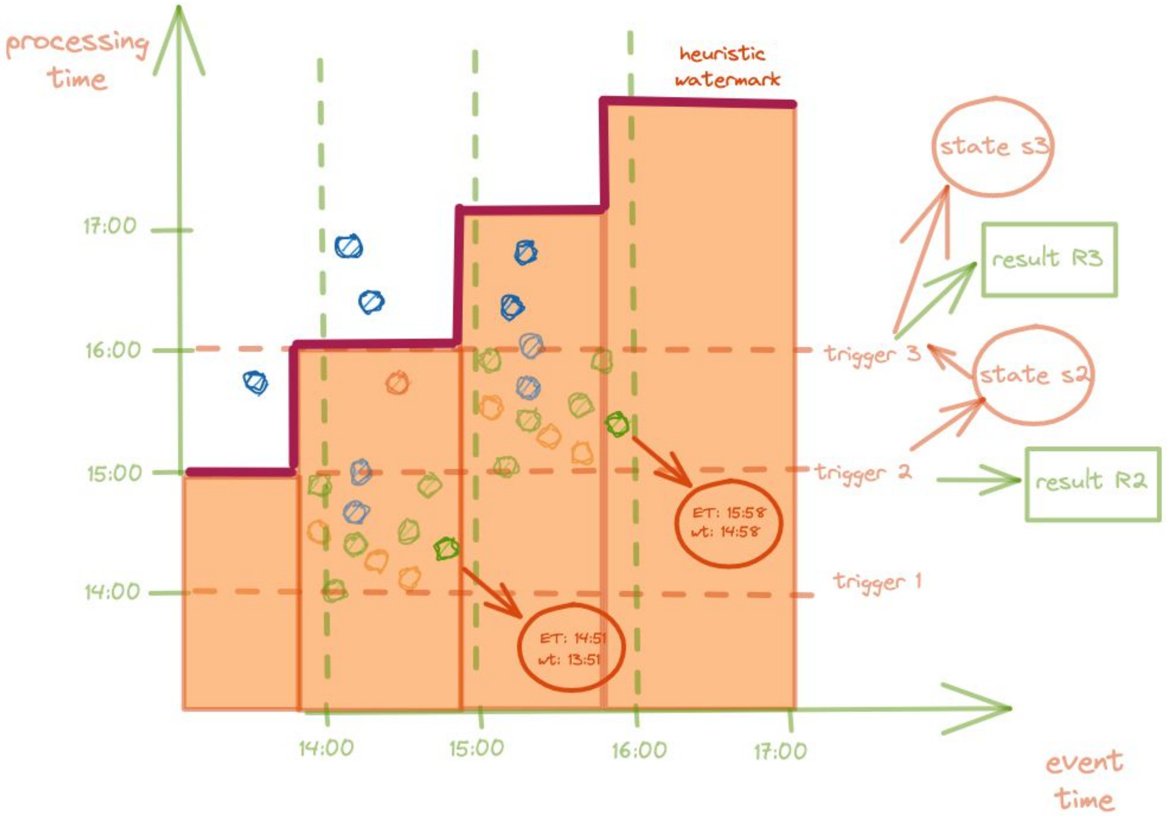
Stream processing: watermarks

- **Perfect** - no such thing as late data; all data are early or on time
- **Heuristic** - an estimate of progress that is as accurate as possible

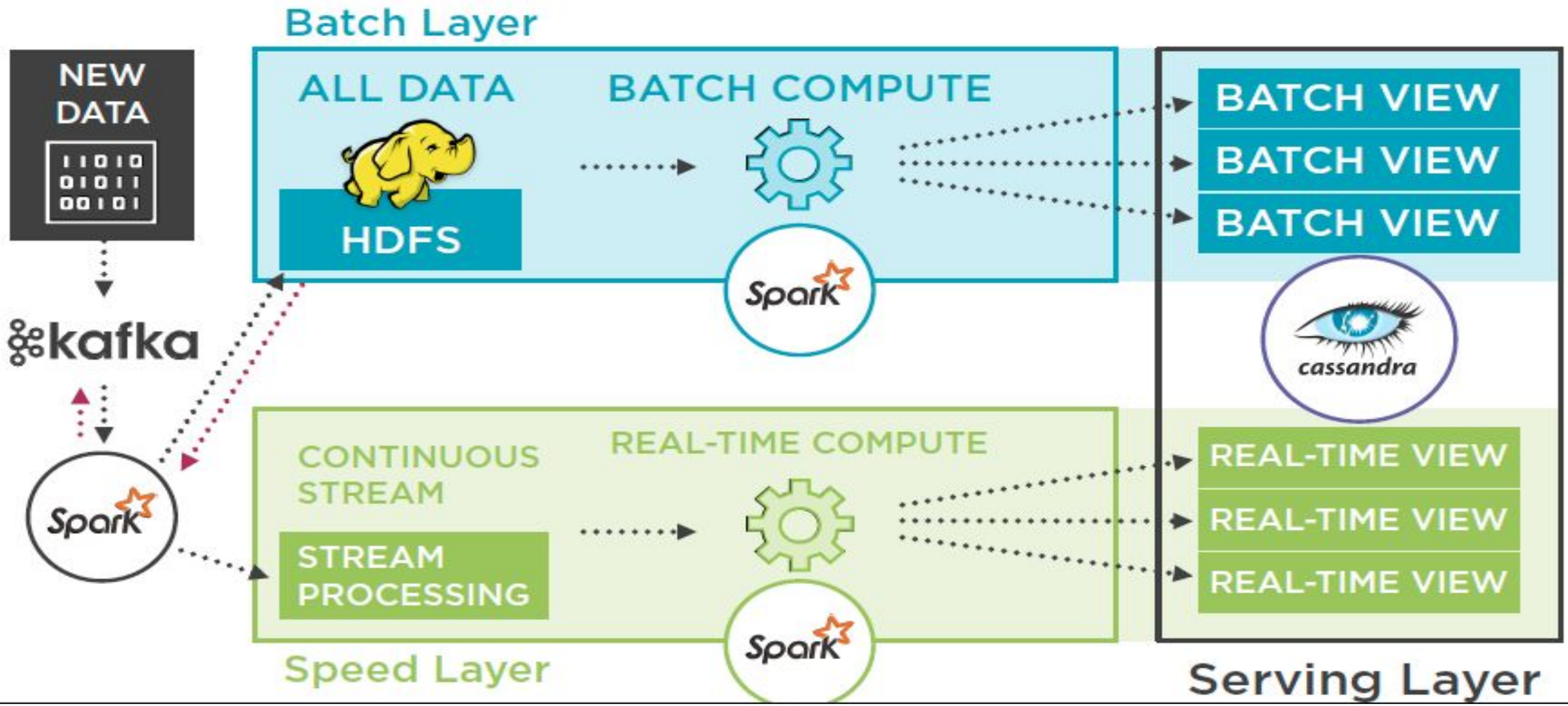


Stream processing: watermarks

- **Perfect** - no such thing as late data; all data are early or on time
- **Heuristic** - an estimate of progress that is as accurate as possible



Lamdba architecture



Conclusion

- **Correctness** - windows by **event-time**
- In **distributed system** you should care about **time skew**
- An **event** is **late** only when it has missed a **deadline** specific to the consumer.
- You should care about that **deadline** for **any data processing patterns**

Data processing patterns

Completeness summary

	early panes	full data	Completeness threshold
Batch processing	no	late	need
Stream processing	yes	earlier	need
Lambda arch	yes	late	need

Chapter 2: Completeness

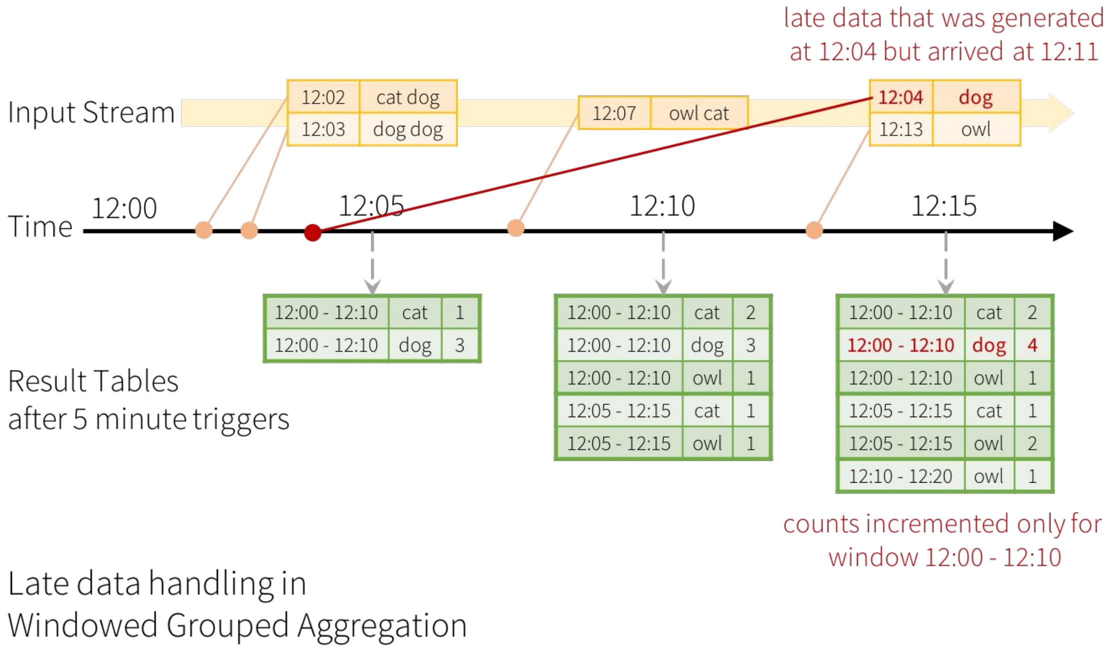
Spark Streaming

Trigger:

- Repeated update triggers

Watermark:

- Garbage collection



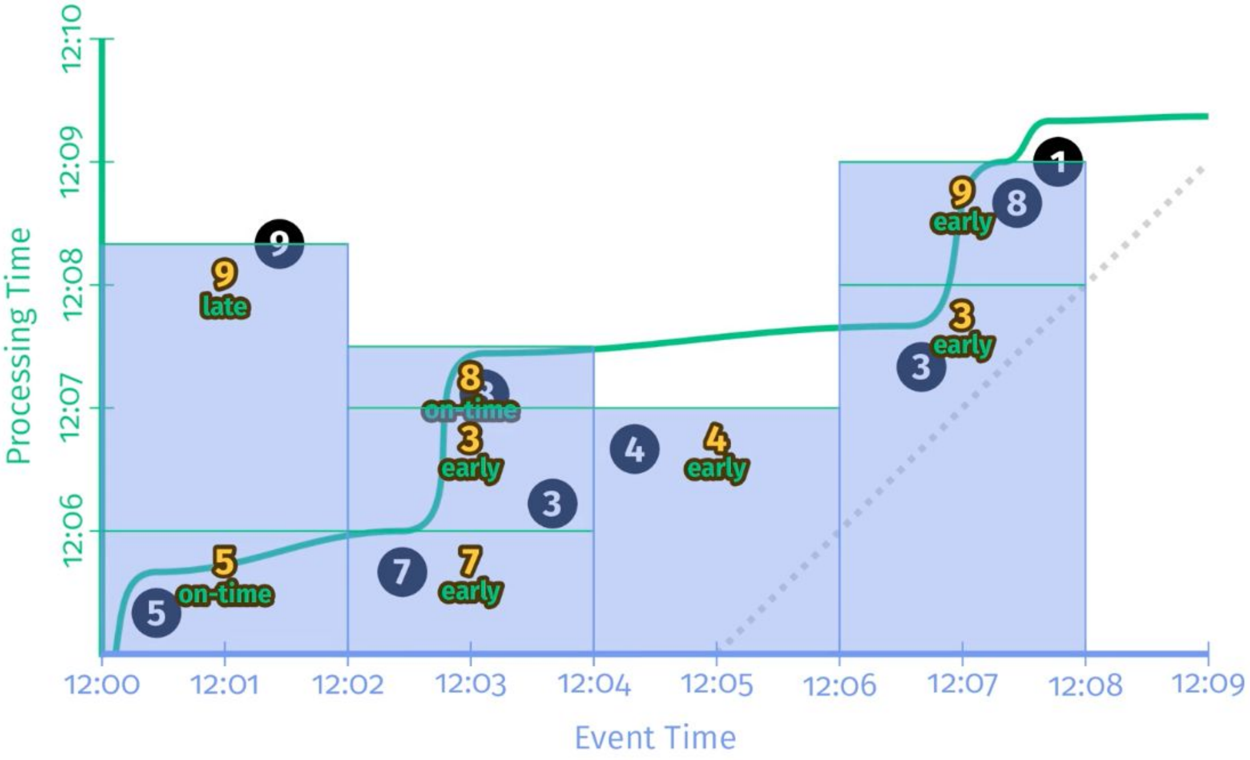
Apache Beam

Trigger:

- Repeated update triggers
  - Unaliged
  - Aliged
- Completeness trigger

Watermark:

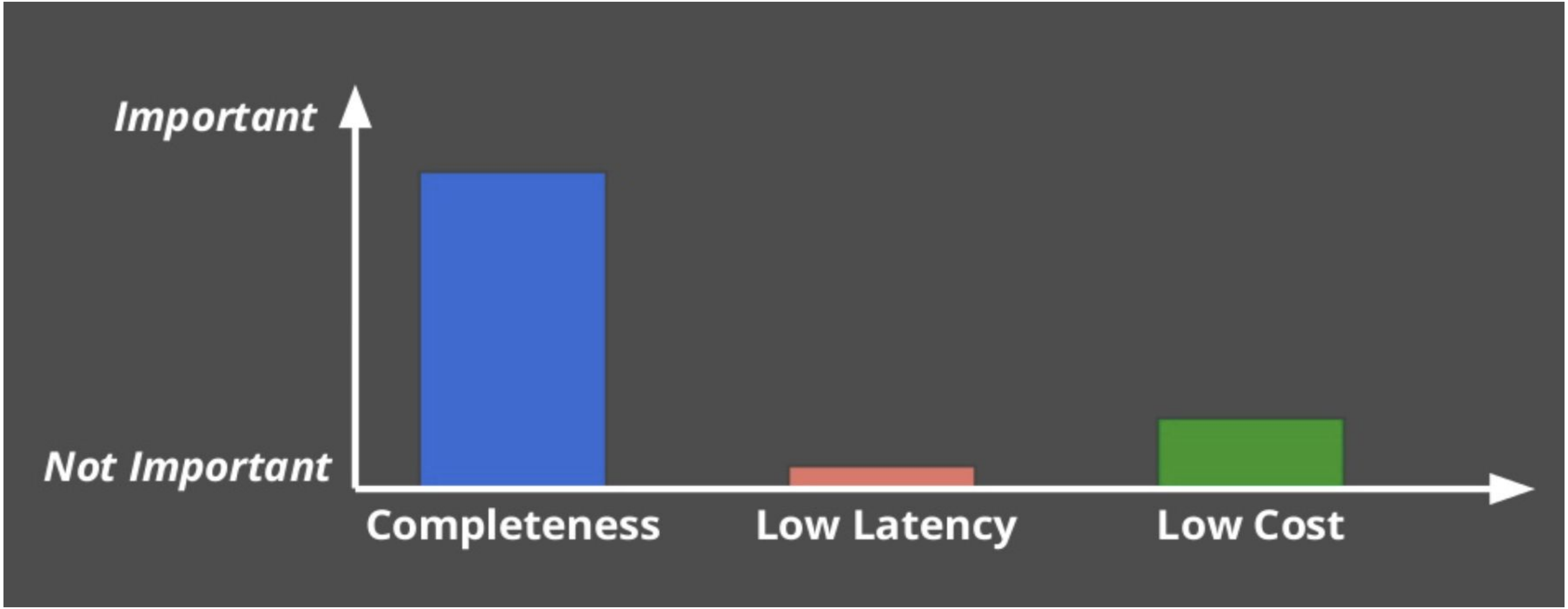
- Early pane
- On-time pane
- Late pane



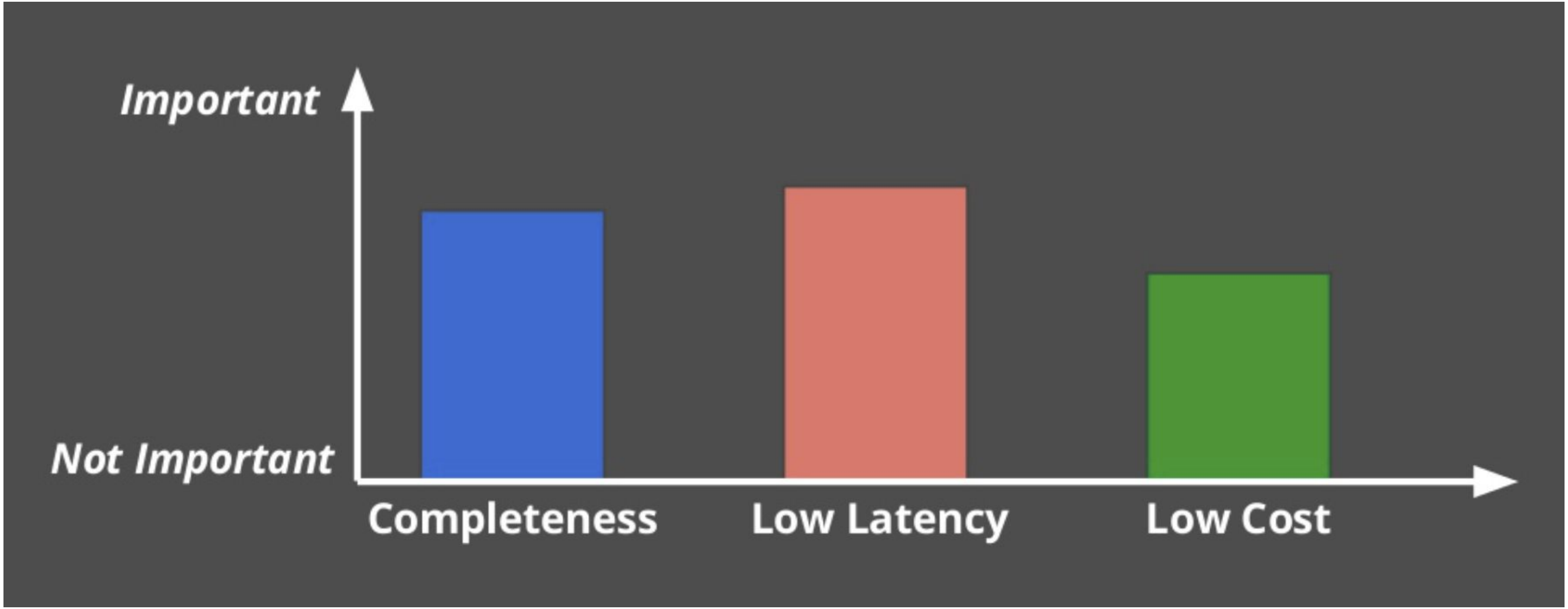
Tradeoffs

- **Completeness:** How important is it to have all of your data before you compute your result?
- **Latency:** How long do you want to wait for data? For example, do you wait until you think you have all data? Do you process data as it arrives?
- **Cost:** How much compute power/money are you willing to spend to lower the latency?

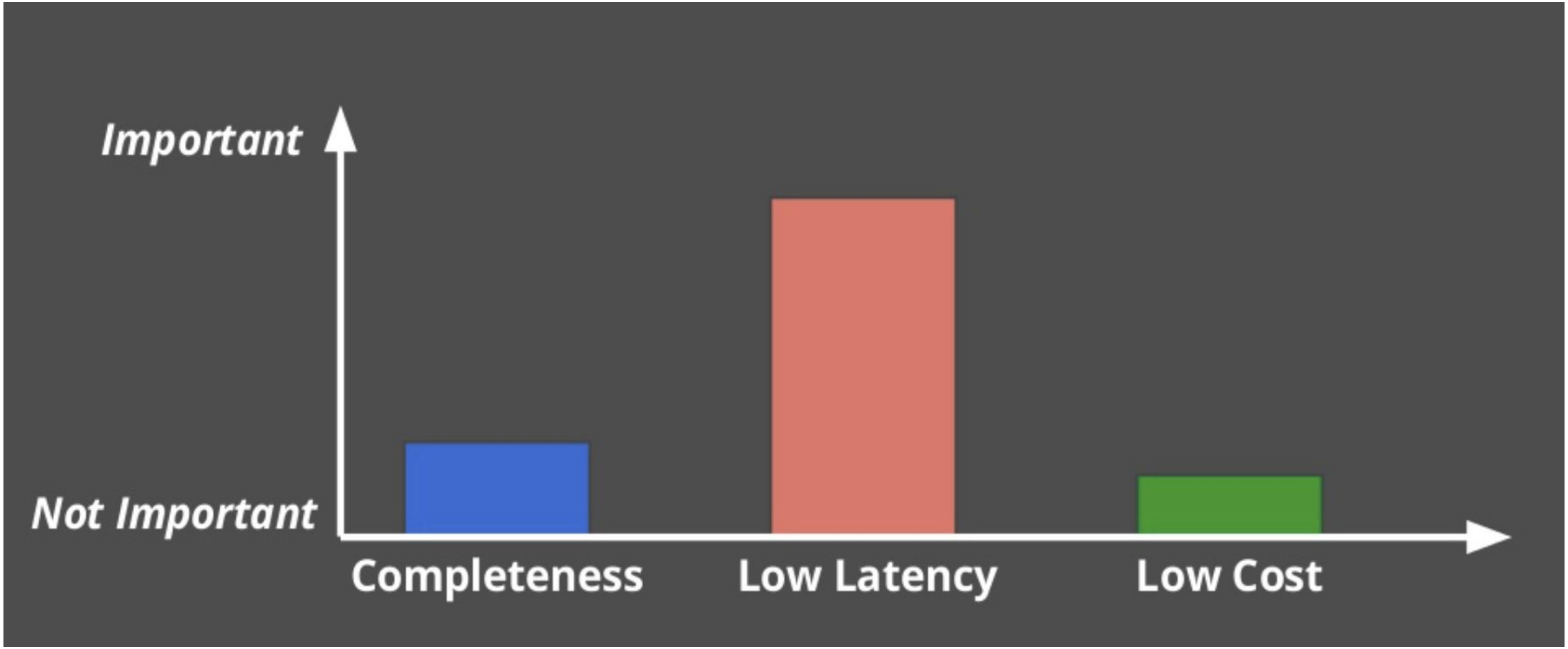
Example: Billing Pipeline



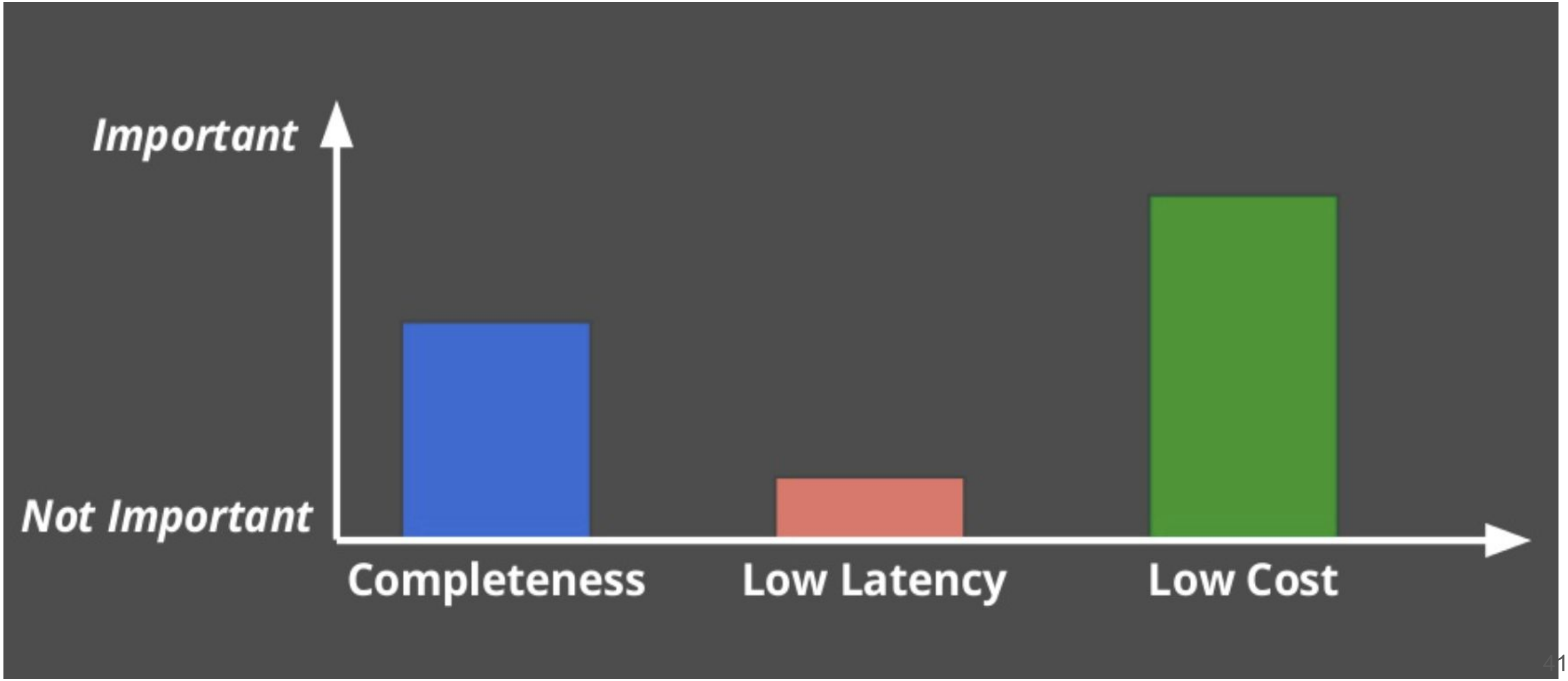
Example: Live cost estimation pipeline



Example: Abuse detection pipeline



Example: Abuse detection backfilling pipeline



Conclusion

- **Tradeoff** is the balance between **completeness, latency, cost**
- **Different frameworks** can offer different ability to managed this balance

To deep dive

- Streaming Systems - the second best book after designing data-intensive application :)
- Building Event-Driven Microservices: Leveragin Organization Data at Scale
- The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing
- Jay Kreps' "Questioning the Lambda Architecture"
- Out-of-Order Processing a New Architecture for High-Performance Stream Systems
- An optimistic approach to handle out-of-order events within analytical stream processing

Drafts

Природа данных

- Batch & Streaming
- Near-real time, microbatch

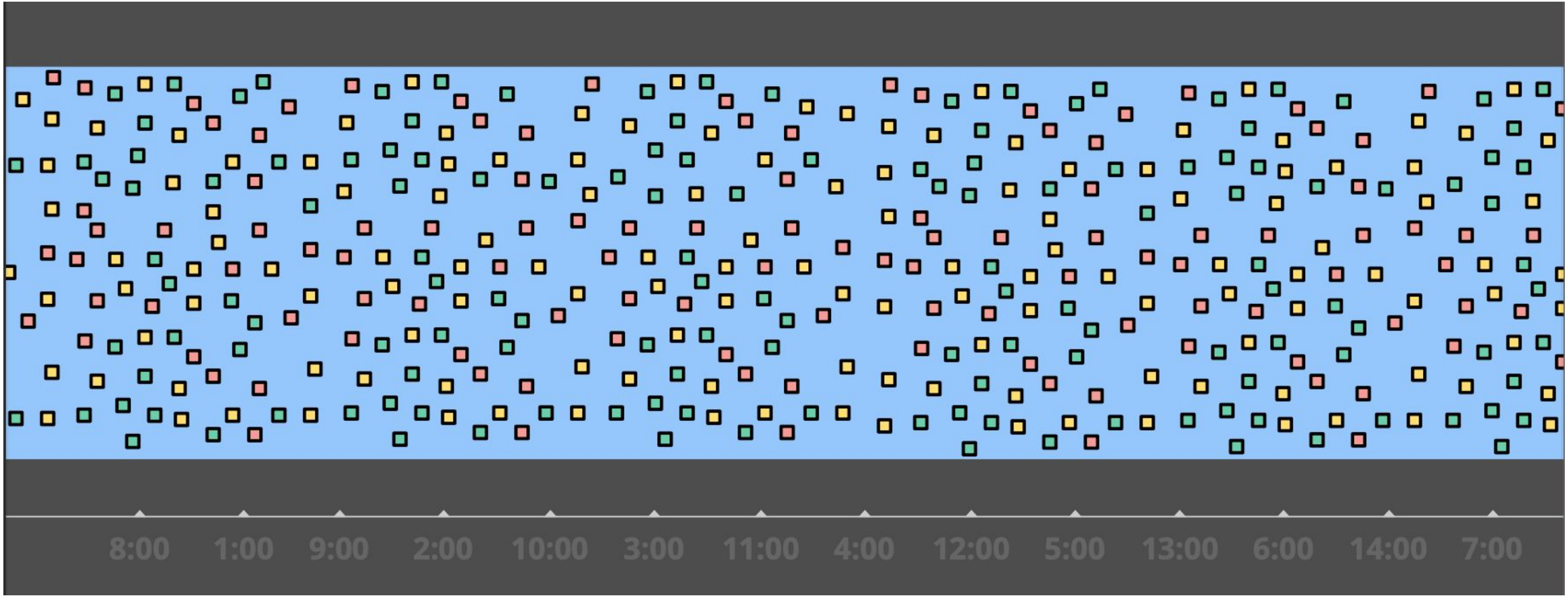
Cardinality

- ***Bounded data*** A type of dataset that is finite in size.
- ***Unbounded data*** A type of dataset that is infinite in size (at least theoretically).

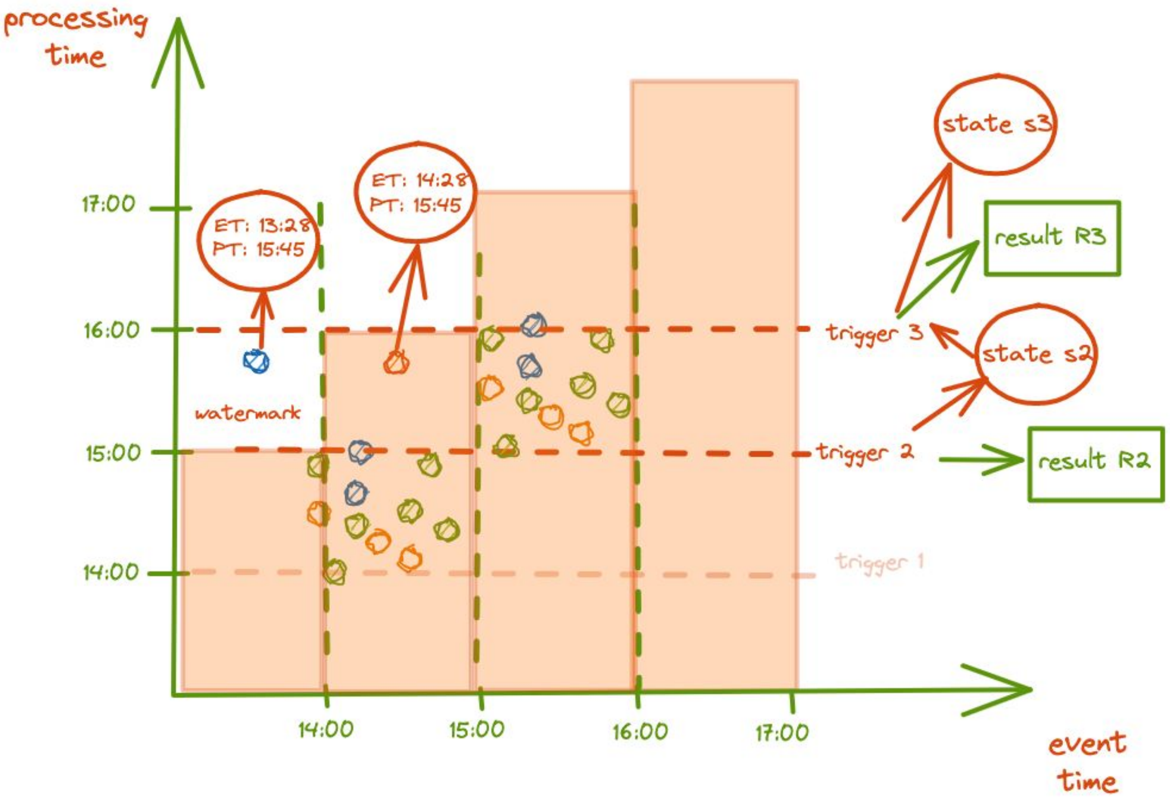
Conclusion

- **Unbounded vs bounded data** is a better characteristic than stream or batch processing for data itself
- Batch processing and streaming **aren't** two **incompatible things**; they are a function of different windowing options.
- **Event time** and **processing time** are two different concepts, and may be out of step with each other.
- **Event-time skew** is a big problem
- **Completeness** is knowing that you have processed all the events for a particular window.
- It's very important to know about input source
- Late data is always a business requirements

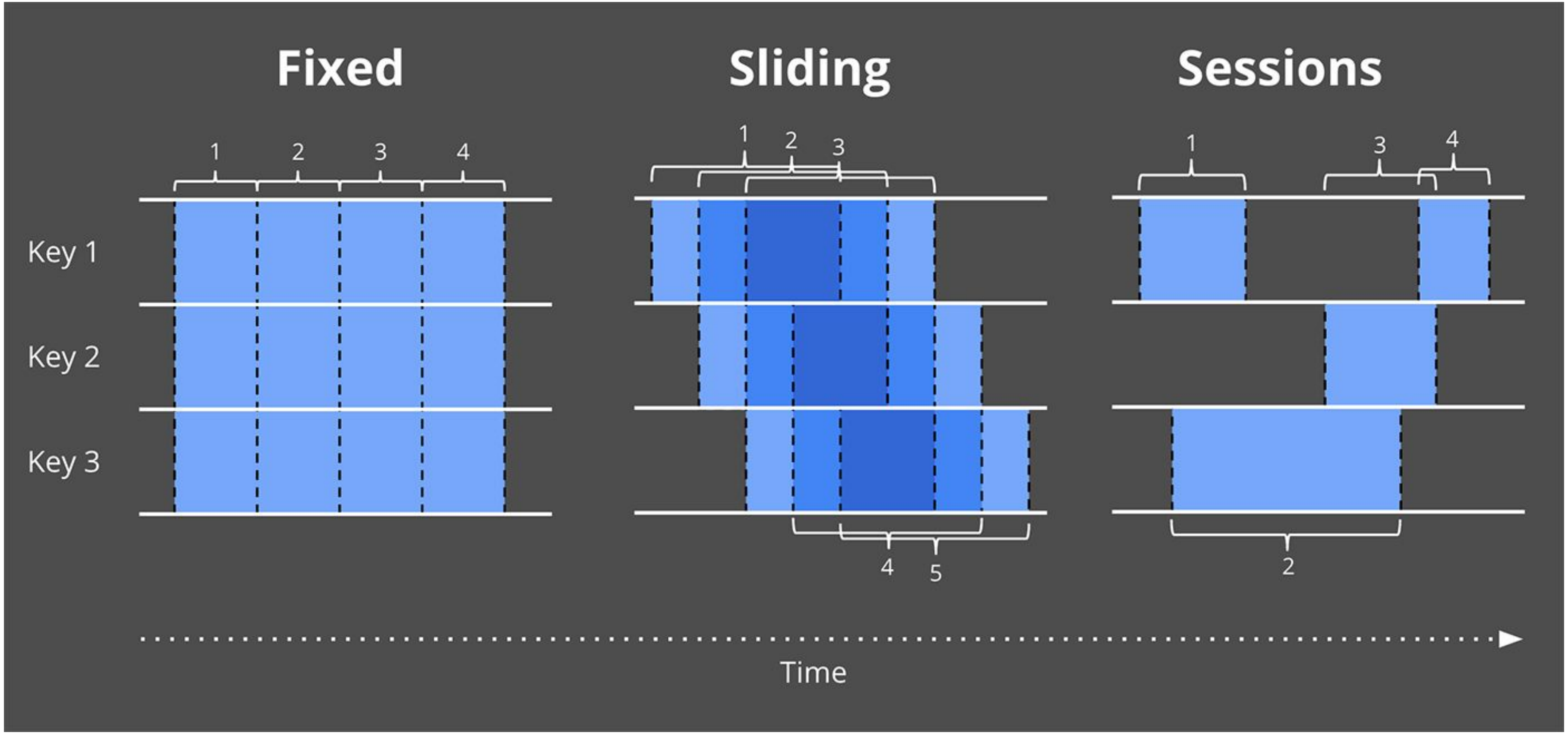
Continuously growing data



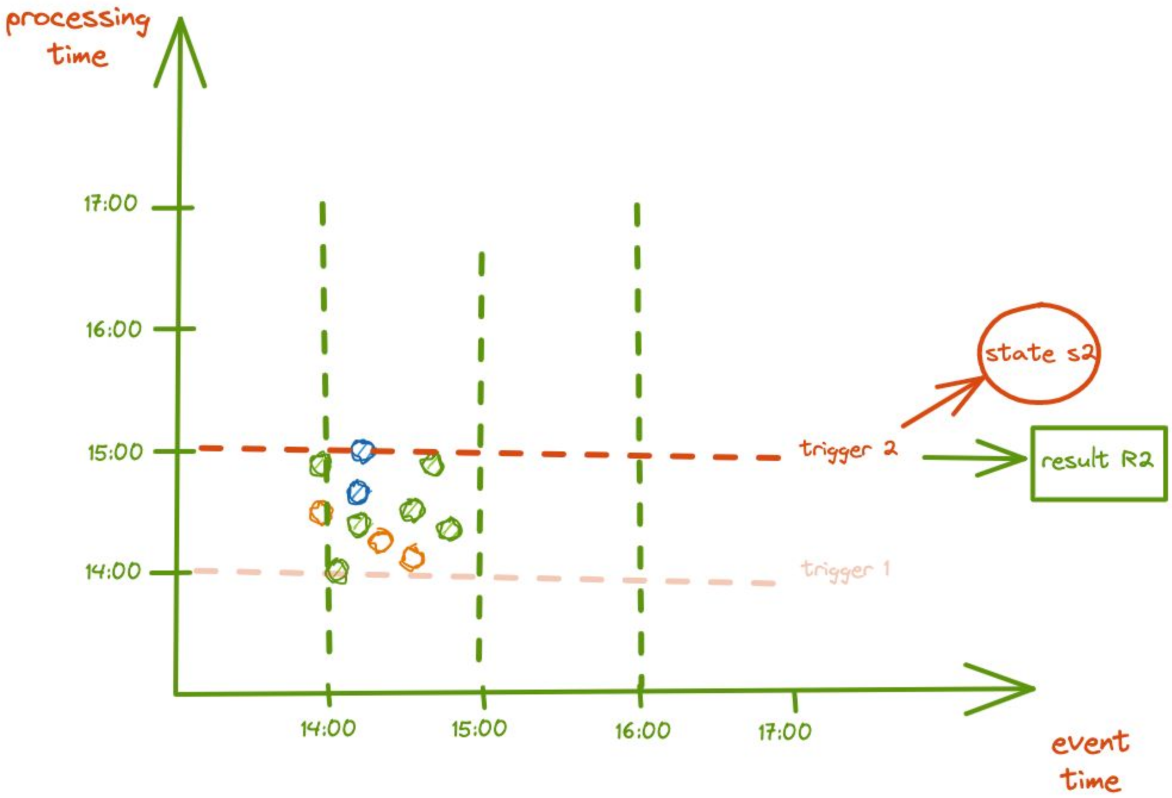
Stream processing: watermarks



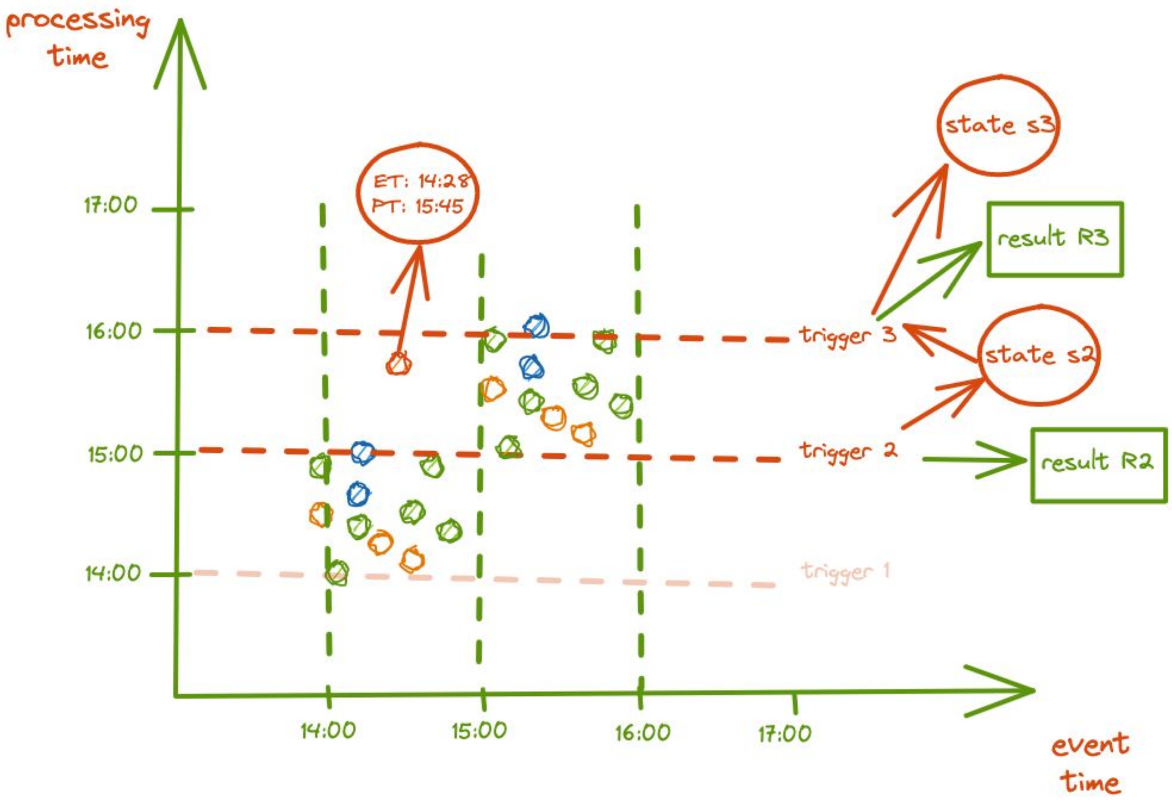
Note: Windows Strategy



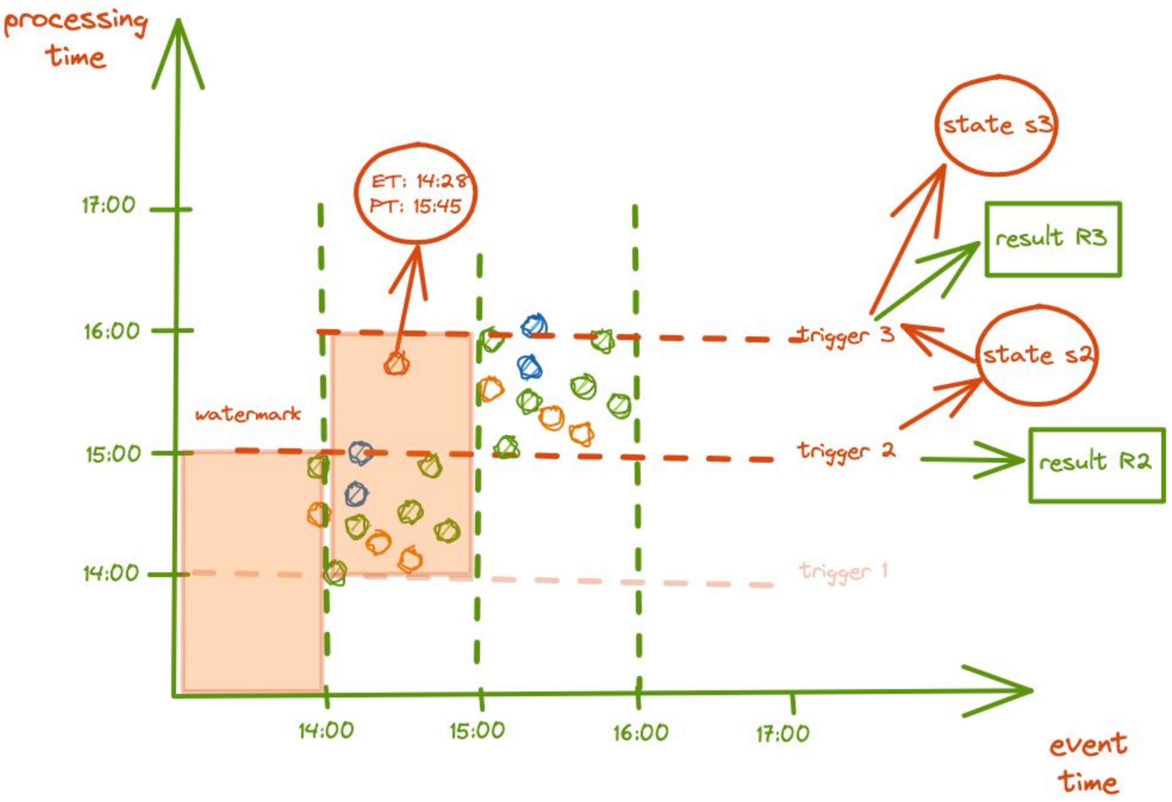
Stream processing: watermarks



Stream processing: watermarks



Stream processing: watermarks



Note: Time Domain

Batch Processing: Fixed Window by event time

- Delay
- Repetitive Runs

Note: Completeness

Streaming: Core Concepts (Spark Streaming)

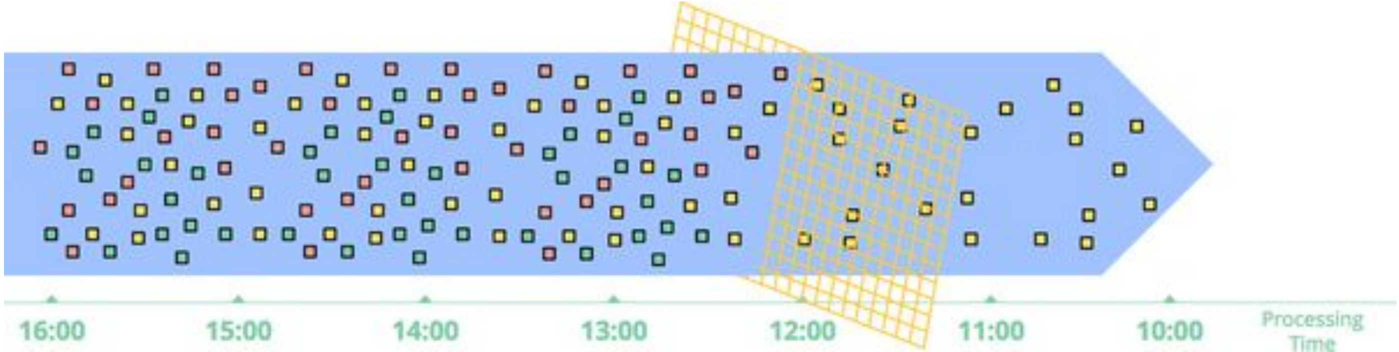
Streaming

- **Highly unordered** with respect to event times, meaning that you need some sort of time-based shuffle in your pipeline if you want to analyze the data in the context in which they occurred.
- **Of varying event-time skew**, meaning that you can't just assume you'll always see most of the data for a given event time *X* within some constant epsilon of time *Y*.

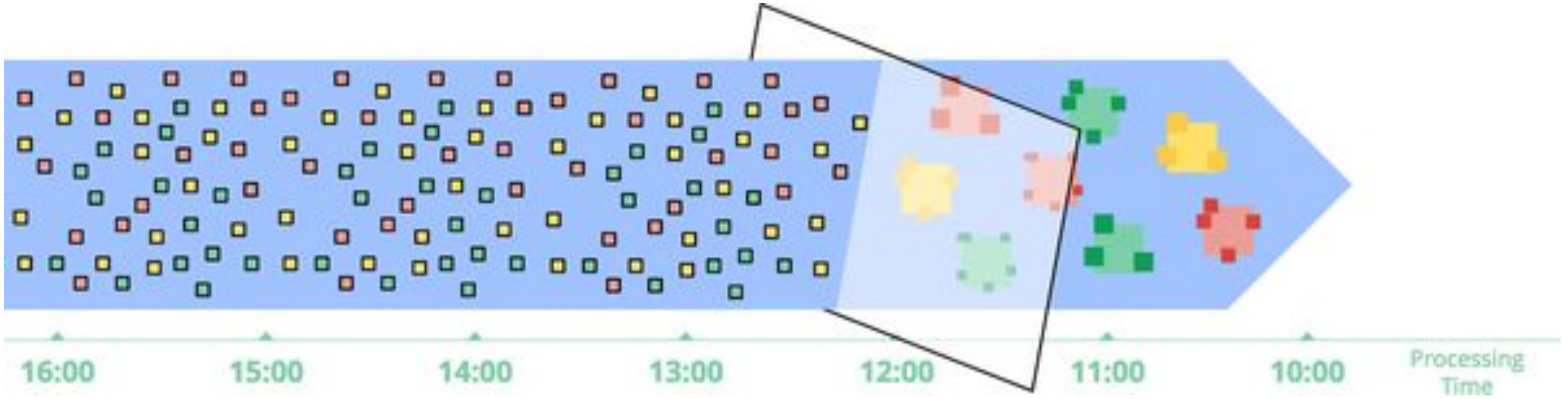
Streaming

- **Time-agnostic**
- **Approximation algorithms**
- **Windowing by processing time**
- **Windowing by event time**

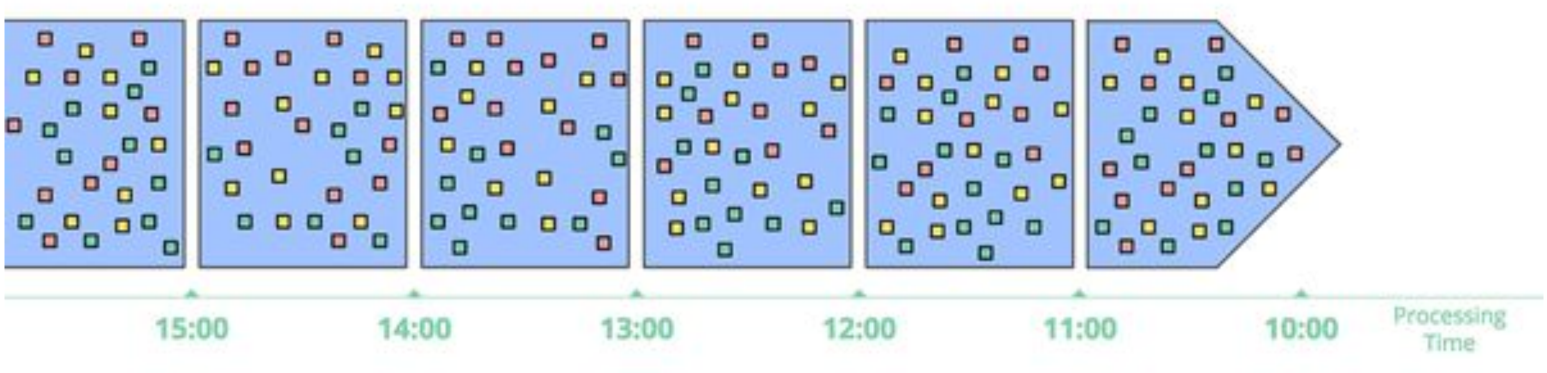
Time Agnostic



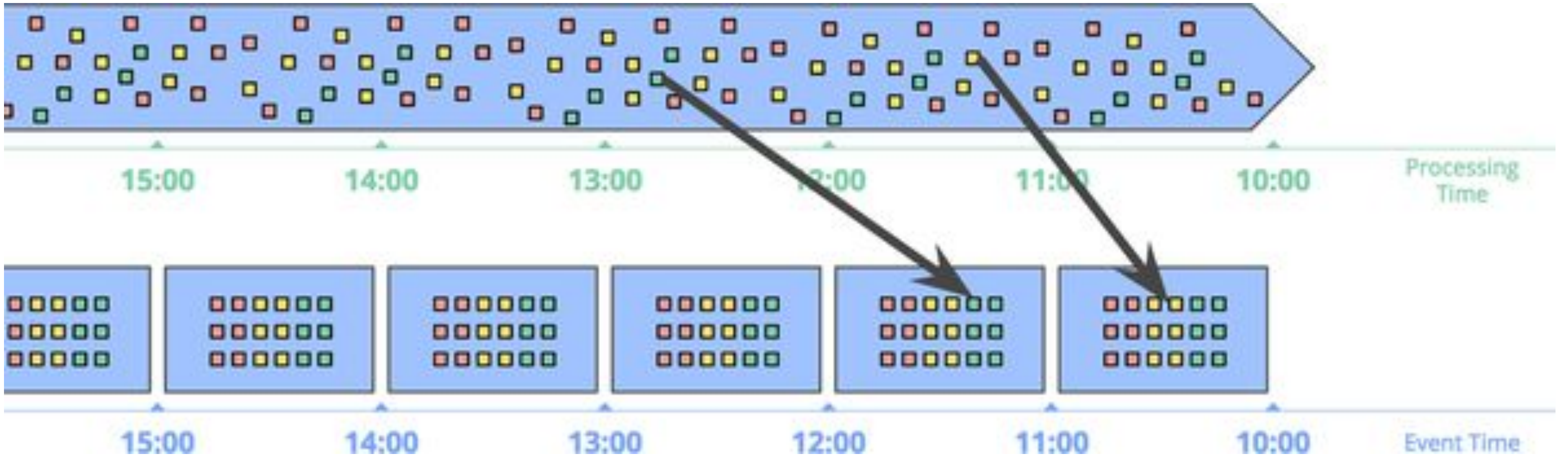
Approximation Algorithms



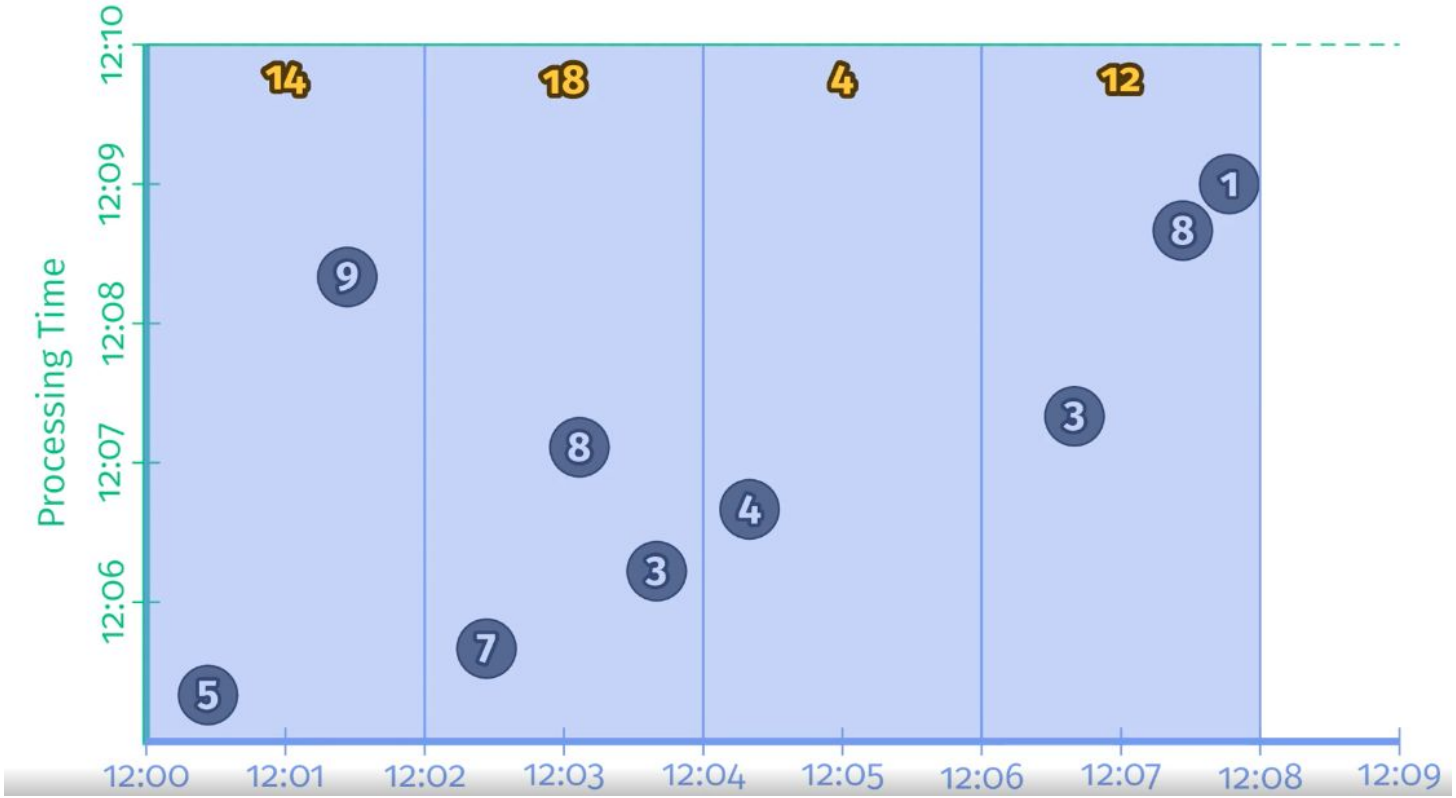
Windowing by procesing time



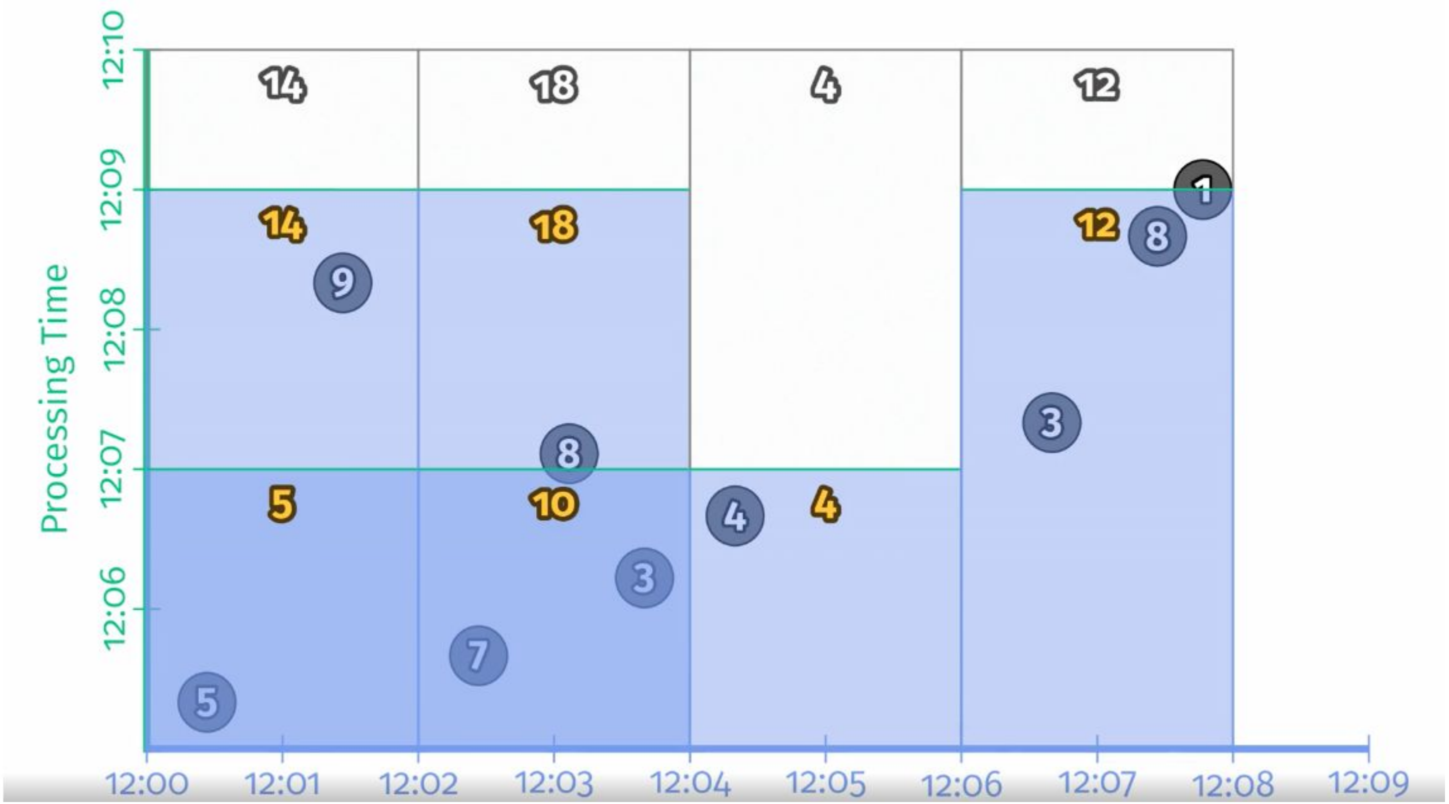
Windowing by event time



Windowing by event time



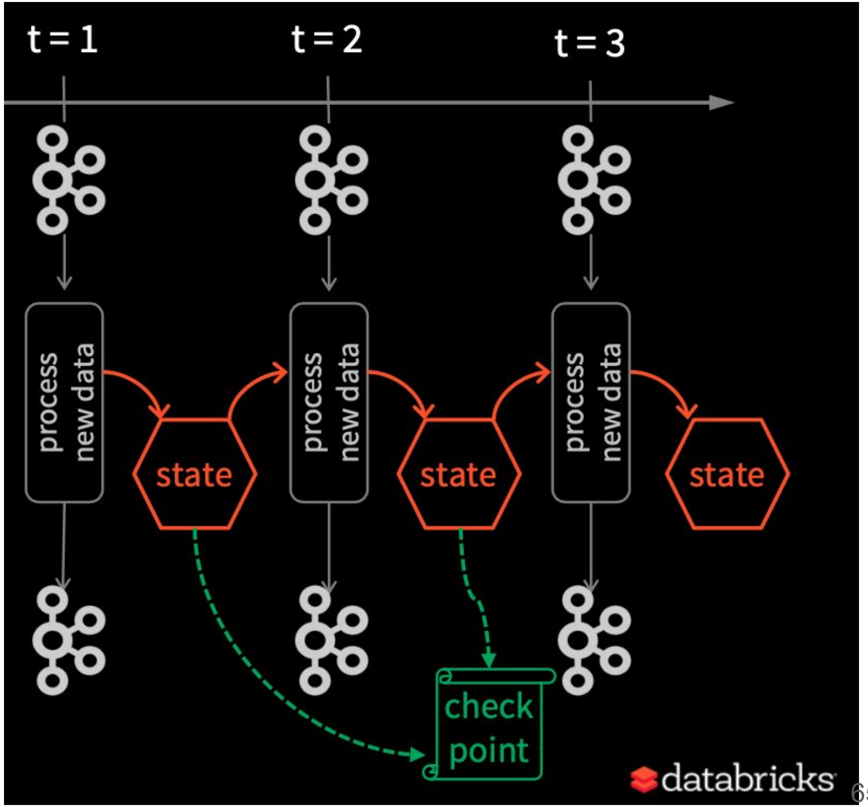
Triggers



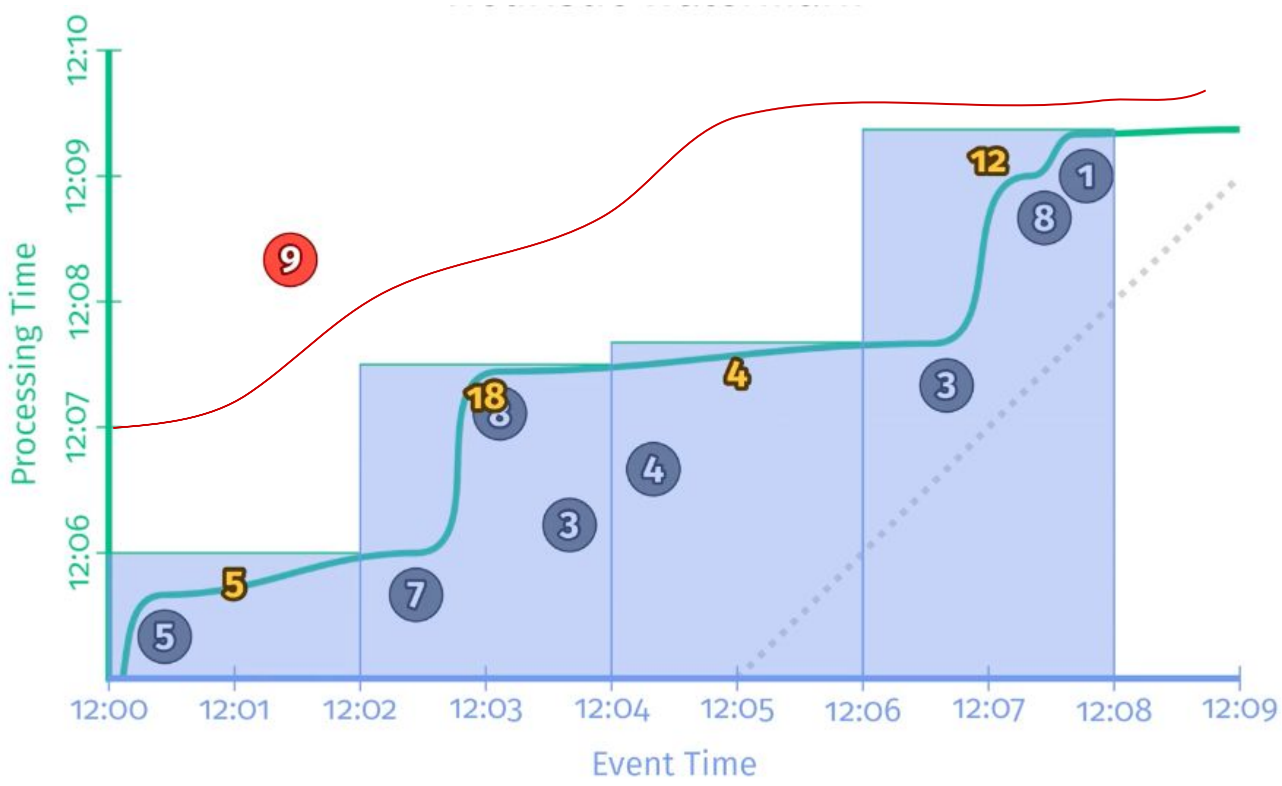
Processing Time
12:10
12:09
12:08
12:07
12:06
14
18
4
12
14
18
12
5
10
4
9
8
3
7
5
4
3
8
1
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09

Stateful streaming

- Each execution reads previous state and writes out updated state
- State stored in executor memory (hashmap in Apache, RocksDB in Databricks Runtime), backed by *checkpoints* in HDFS/S3

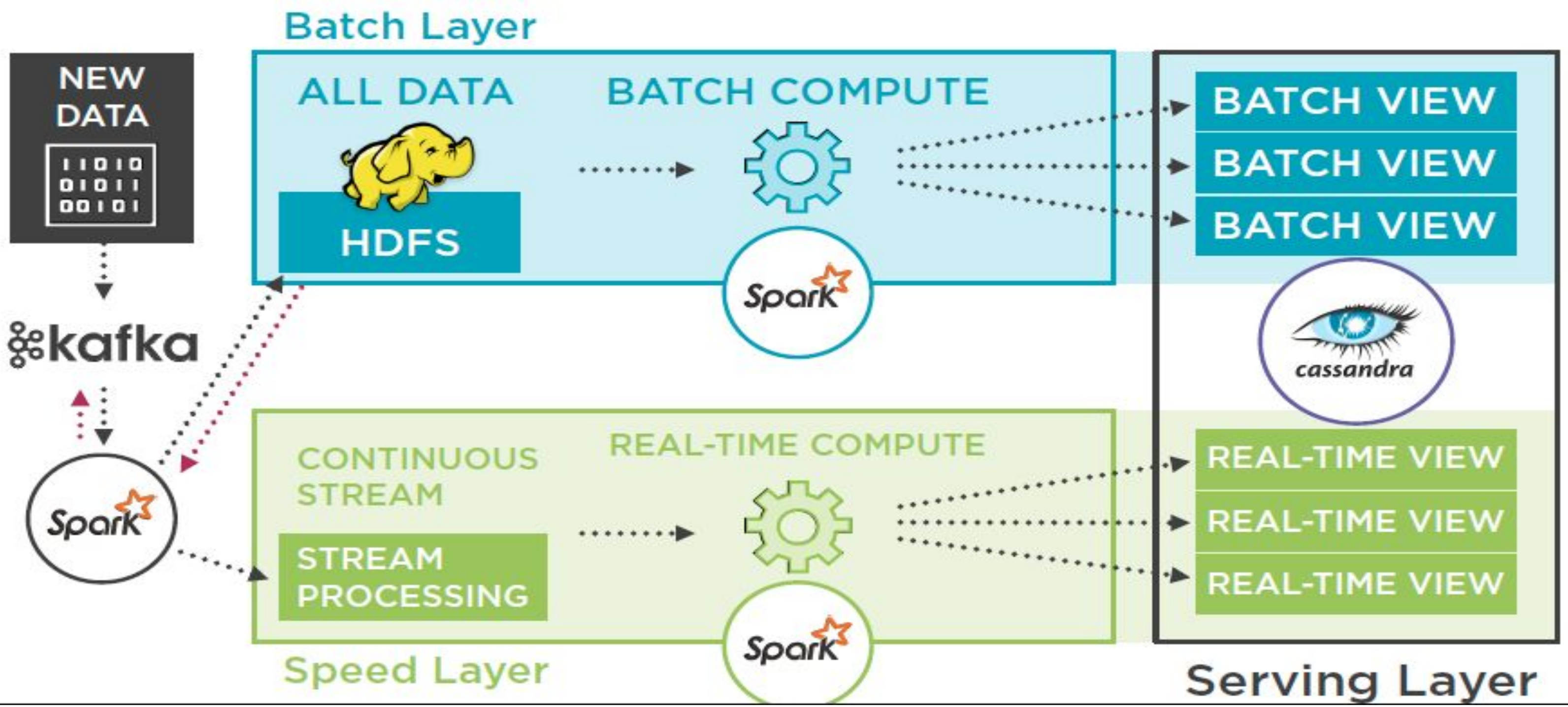


Watermarks



Processing Time
12:10
12:09
12:08
12:07
12:06
9
5
5
7
18
8
3
4
4
3
12
8
1
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09
Event Time

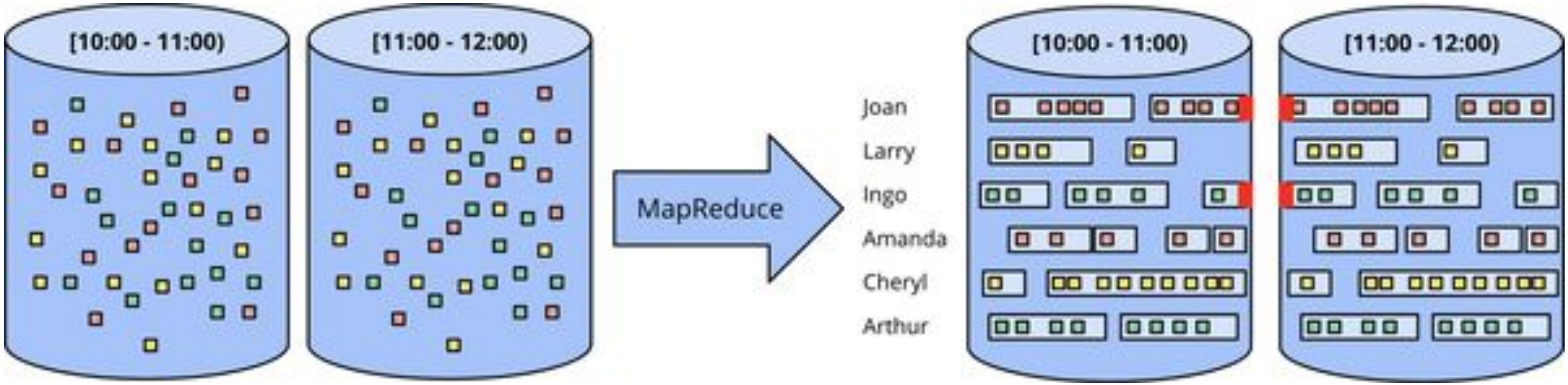
Unbounded data:
Lambda Architecture



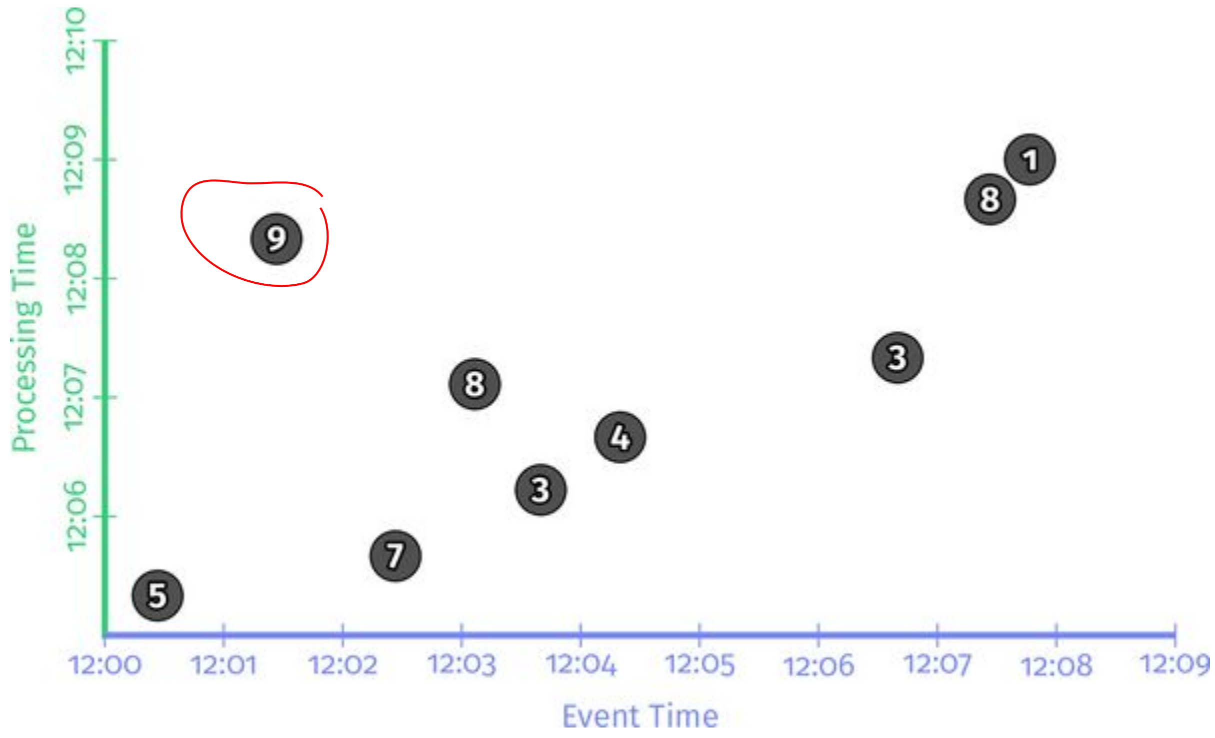
Пример

```
mysql> describe orders;
+----------+----------+-----+-----+----------------------------------+
| Field    | Type     | Null| Key | Description
+----------+----------+-----+-----+----------------------------------+
| Id       | int      | NO  | PRI | номер заказа
| UnitId   | int      | NO  |     | номер пиццерии
| Source   | int      | NO  |     | источник заказа
| State    | int      | NO  |     | статус
| SaleDate | datetime | NO  |     | дата+время продажи заказа
+----------+----------+-----+-----+----------------------------------+
5 rows in set (0.00 sec)
```

Unbounded data:
Batch - Session



Event-time skew



Fallacy №1: Cardinality