# Performing Windowing Operations on Streams

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

# Overview

Windowing operations on streams

Sliding and tumbling windows

Event time vs. processing time

Aggregation operations on windows

UDFs on streaming data

# Stateless and Stateful Transformations

# Transformations

**Stateless**

Transformations which are applied on a single stream entity

**Stateful**

Transformations which accumulate across multiple stream entities

# Stateless Transformations

94 86 32 55 83 22 19 65 78 30

**Each entity is operated on standalone**

Speed exceeded? Alert triggered

1

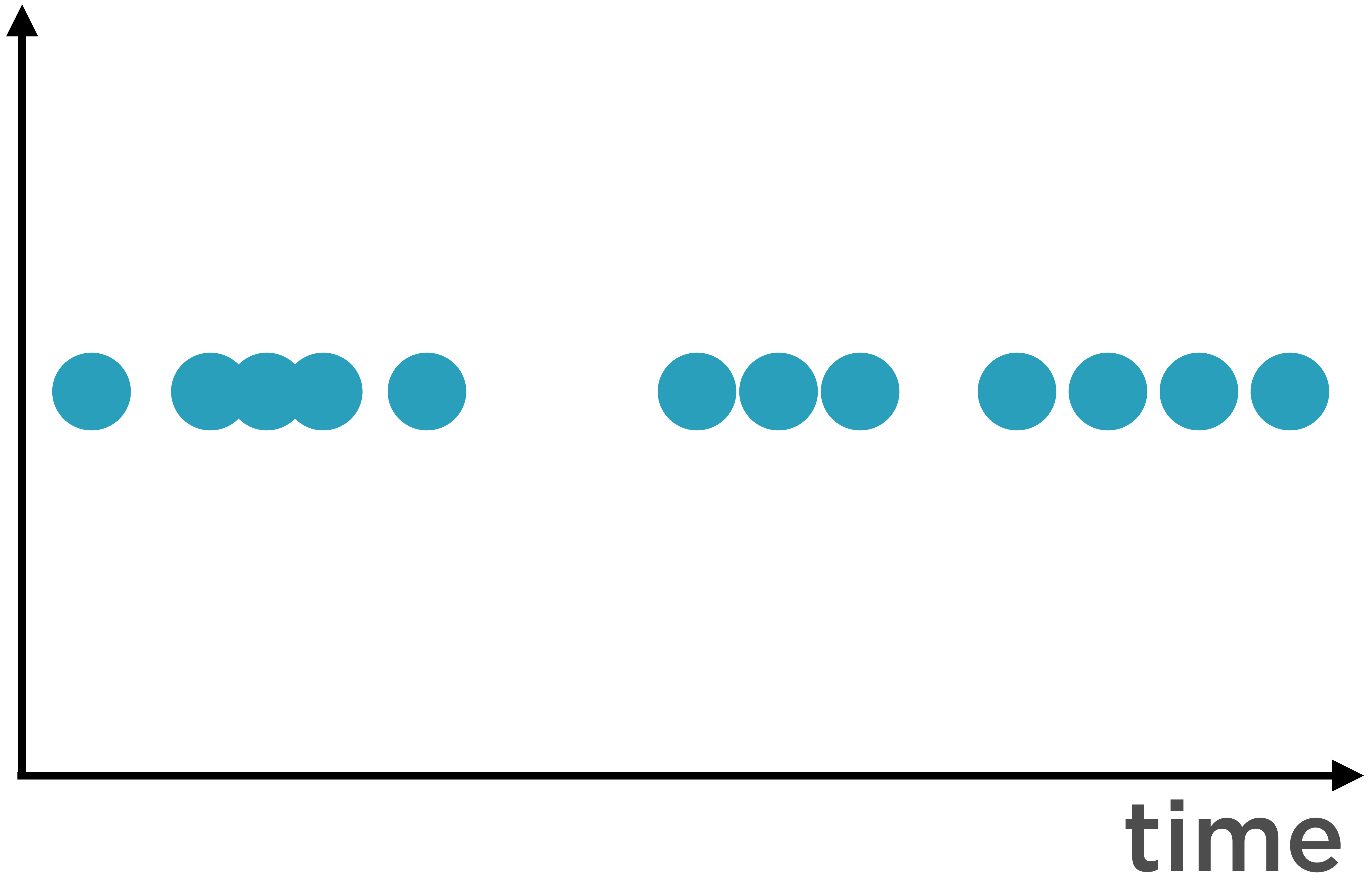
2

4

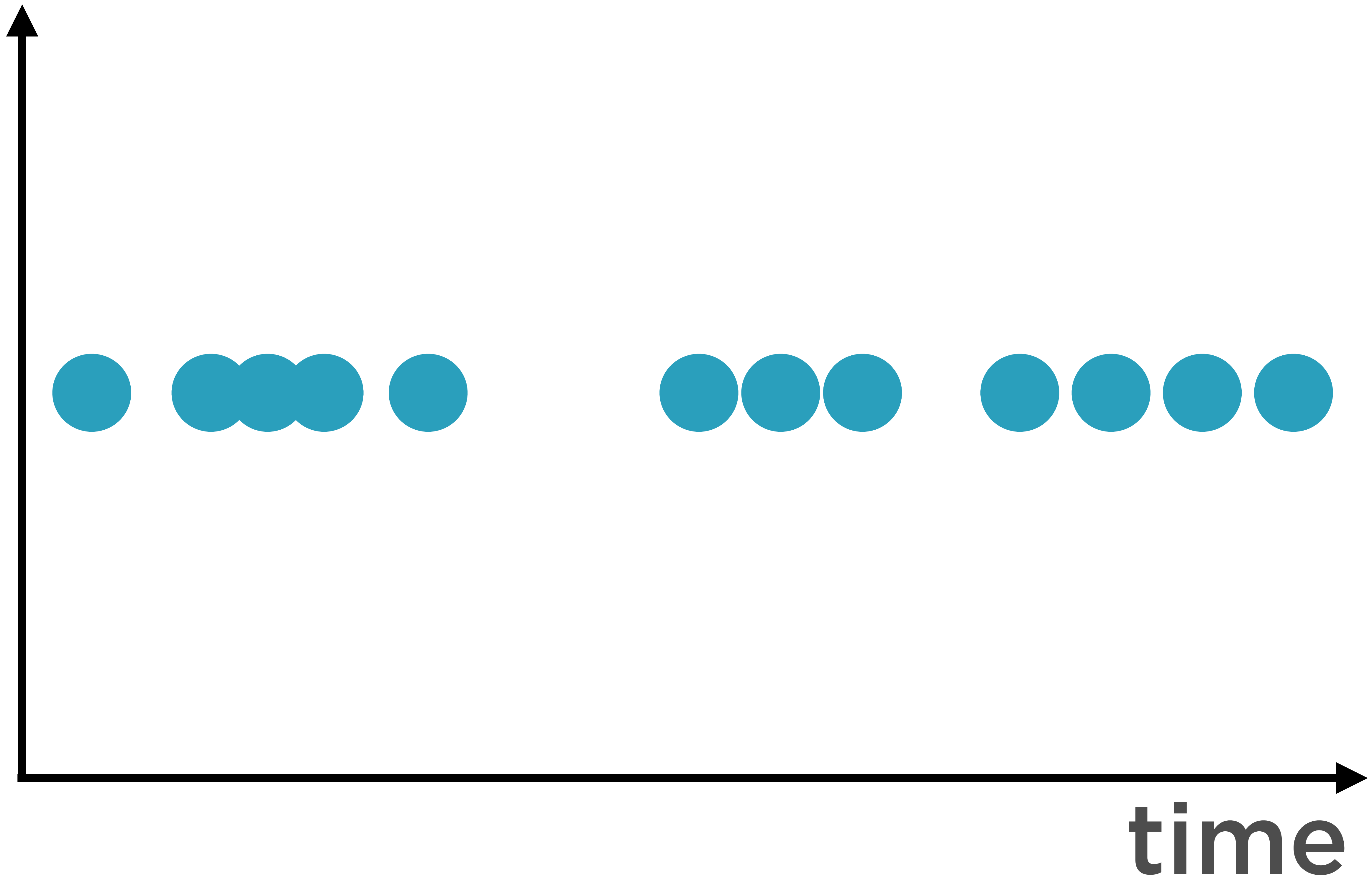
# Window Transformations

Accumulate information across a window in a stream

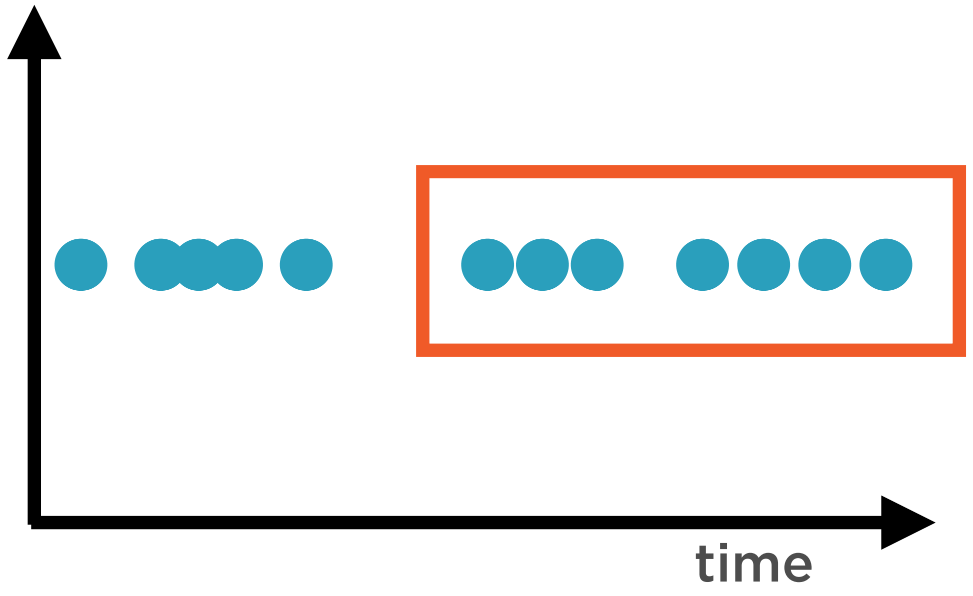
# Streaming Data



Streaming Data



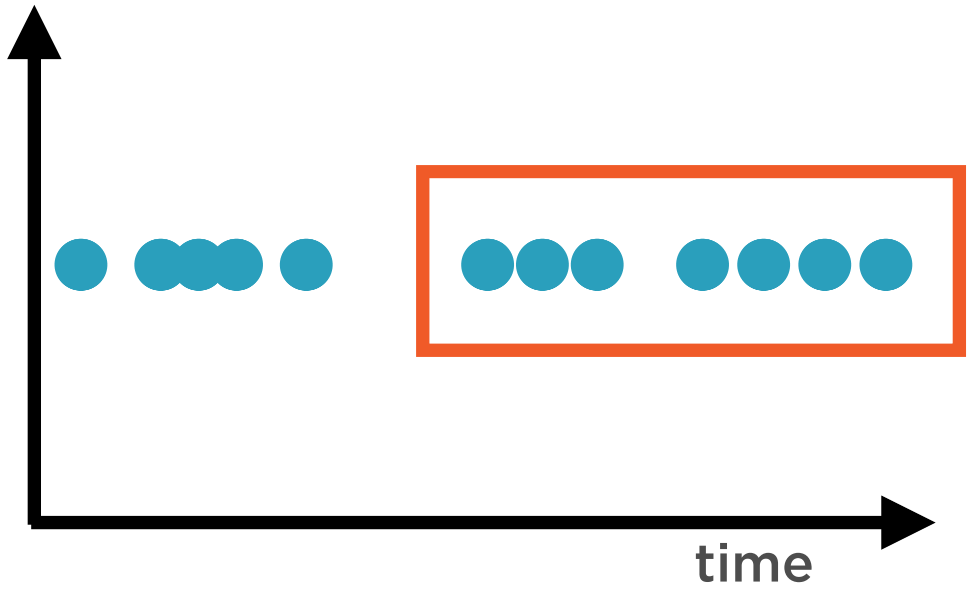
# Streaming Data




**A window is a subset of a stream based on**

- Time interval
- Count of entities
- Interval between entities

# Streaming Data




**Transformations can be applied on all entities within a window**

- sum, min, max, average

# Tumbling, Sliding, and Global Windows

# Types of Windows

Tumbling Window

Sliding Window

Count Window

Session Window

Global Window

# Types of Windows

Tumbling Window

Sliding Window

Count Window

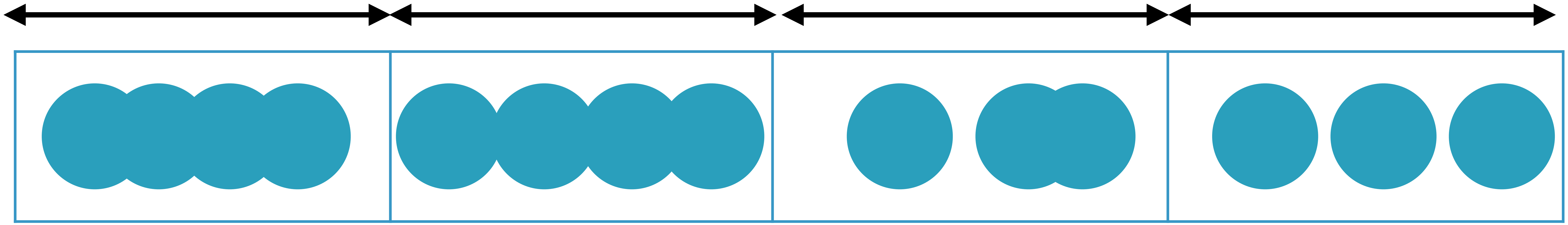
Session Window

Global Window

# Types of Windows

**A stream of data**

# Tumbling Window

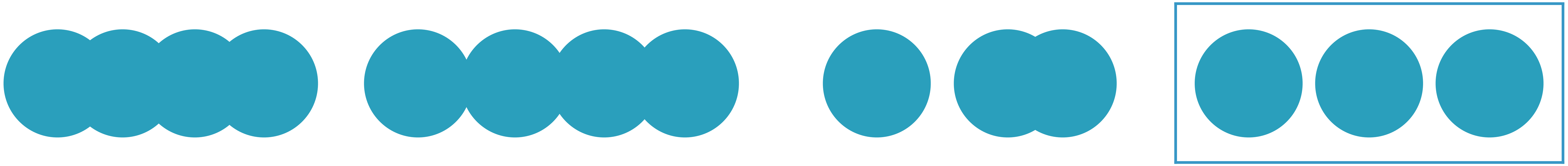


Fixed window size

Non-overlapping time

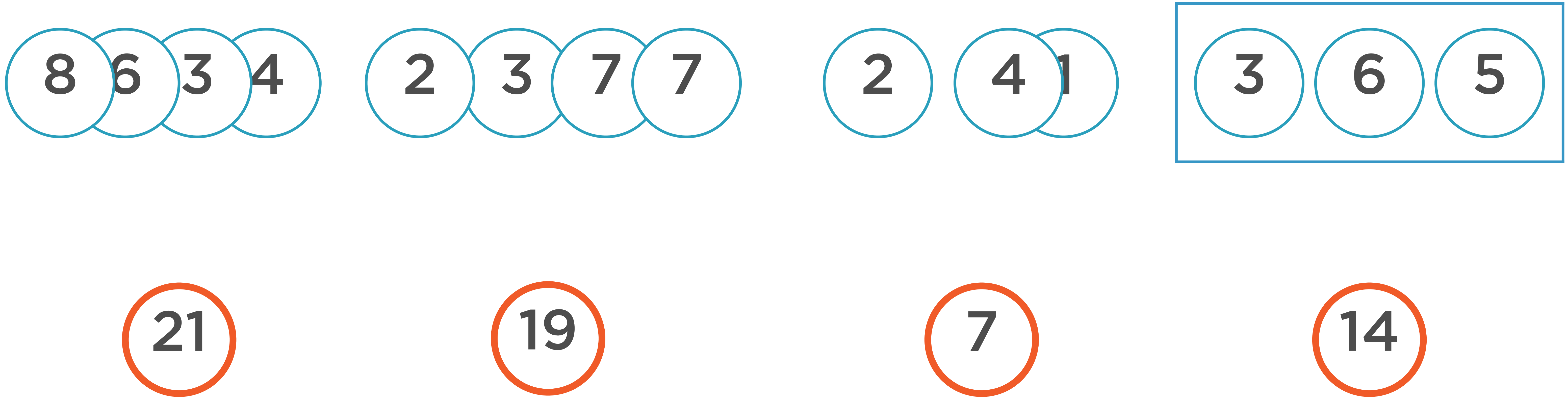
Number of entities differ within a window

# Tumbling Window



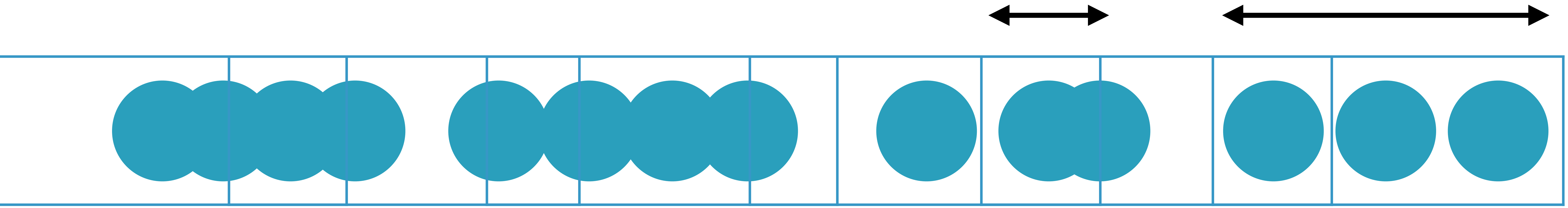
**The window tumbles over the data, in a non-overlapping manner**

# Tumbling Window




**Apply the sum() operation on each window**

# Sliding Window

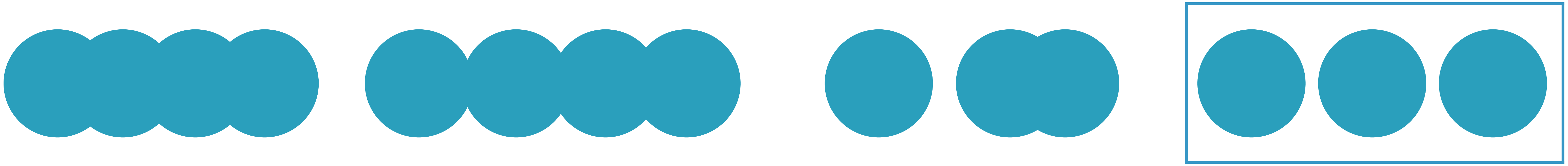


Fixed window size

Overlapping time - sliding interval

Number of entities differ within a window

# Sliding Window

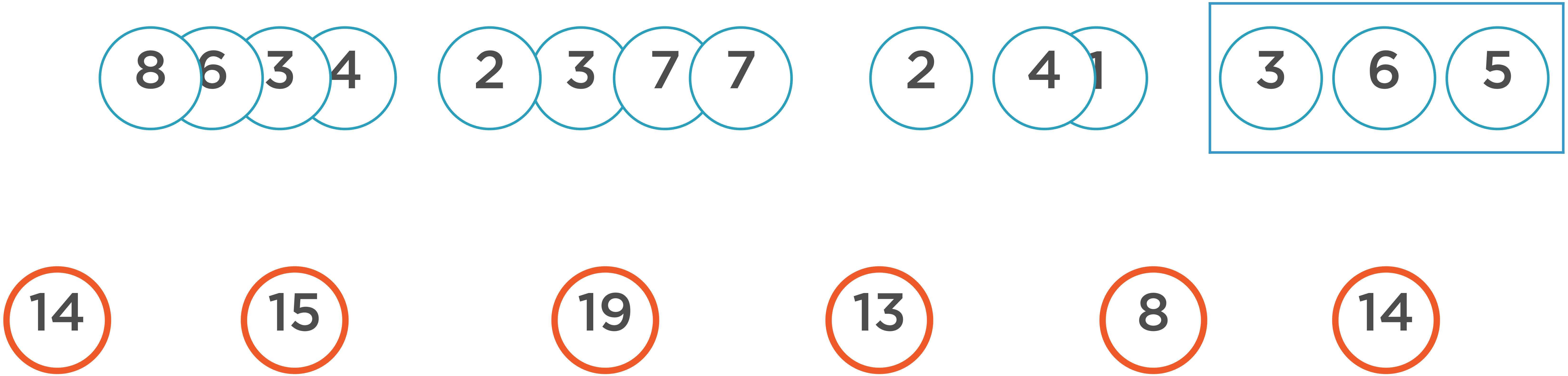


**Fixed window size**

**Overlapping time - sliding interval**

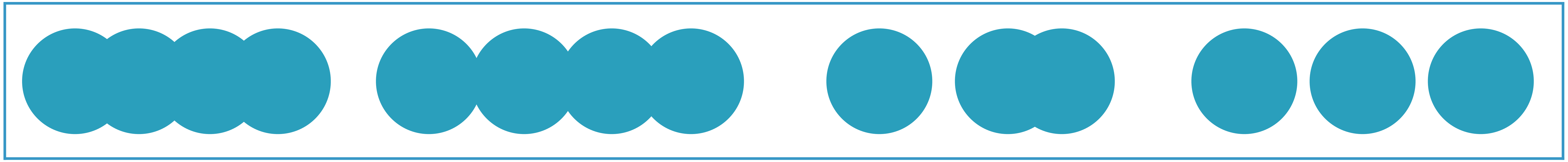
**Number of entities differ within a window**

# Sliding Window




**Apply the sum() operation on each window**

# Global Window

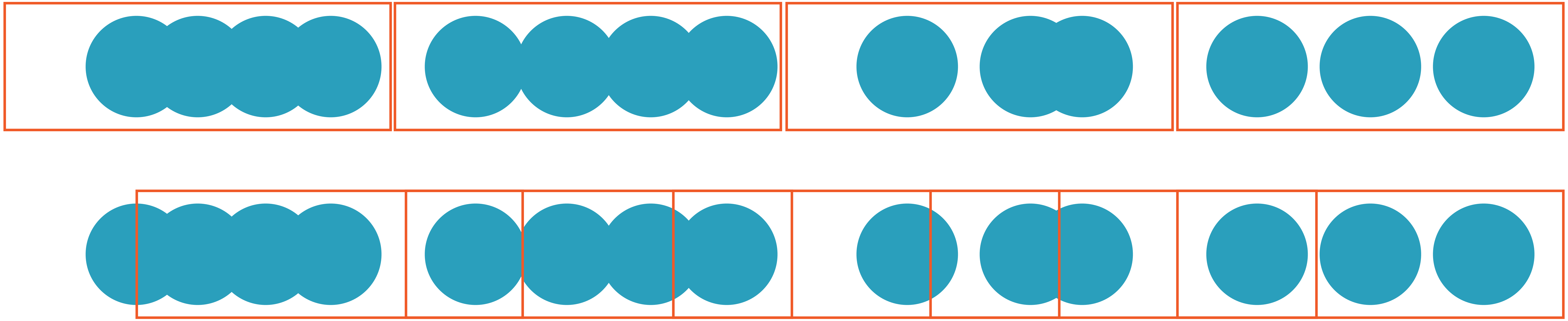


**All data in the stream in one window**

# Event Time and Processing Time

# Time-based Windows

**Tumbling and sliding windows consider entities in a fixed interval of time**



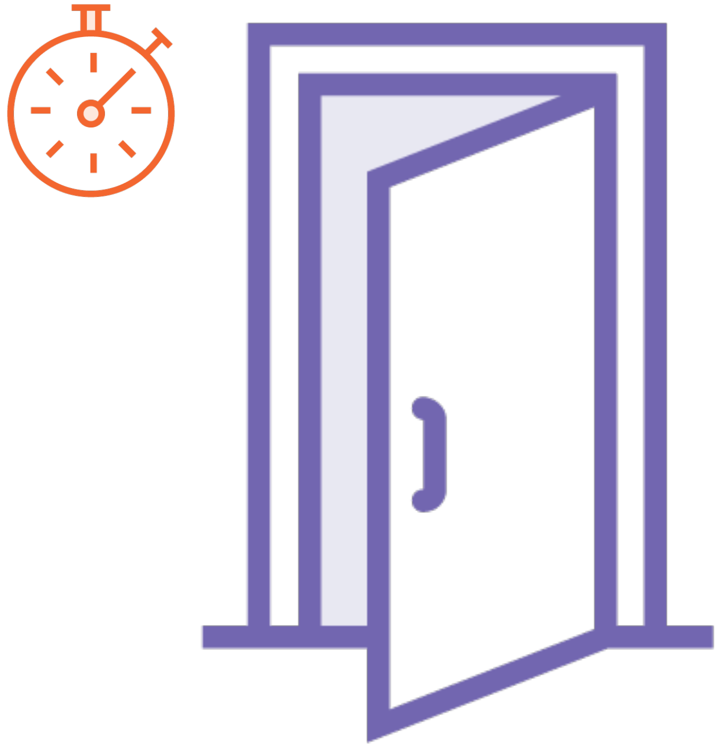
# Time-based Windows

**Tumbling and sliding windows consider entities in a fixed interval of time**

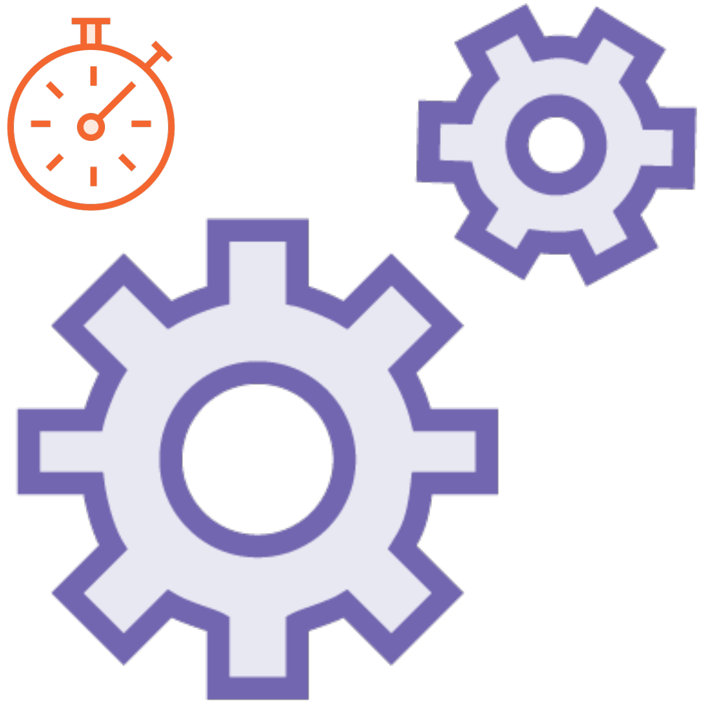
**There are different notions of time that can apply to entities in a stream**

# Time

Event Time



Ingestion Time



Processing Time

# Event Time

**The time at which the event occurred at its original source**

- Mobile phone, sensor, website

**Usually embedded within records**

**Gives correct results in case of out of order or late events**

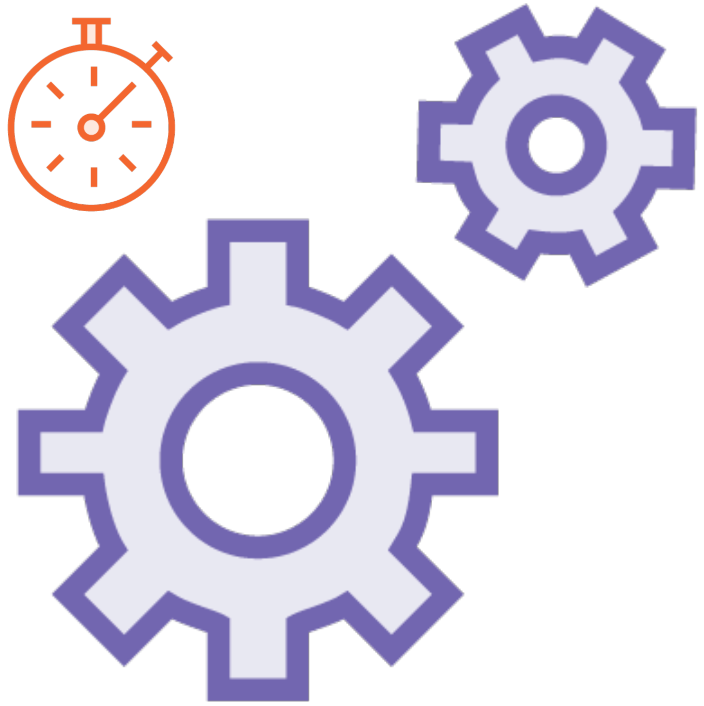
# Ingestion Time

The time at which the event enters the system via a source

Timestamp given by system chronologically after the event time

Cannot handle out of order events

# Processing Time



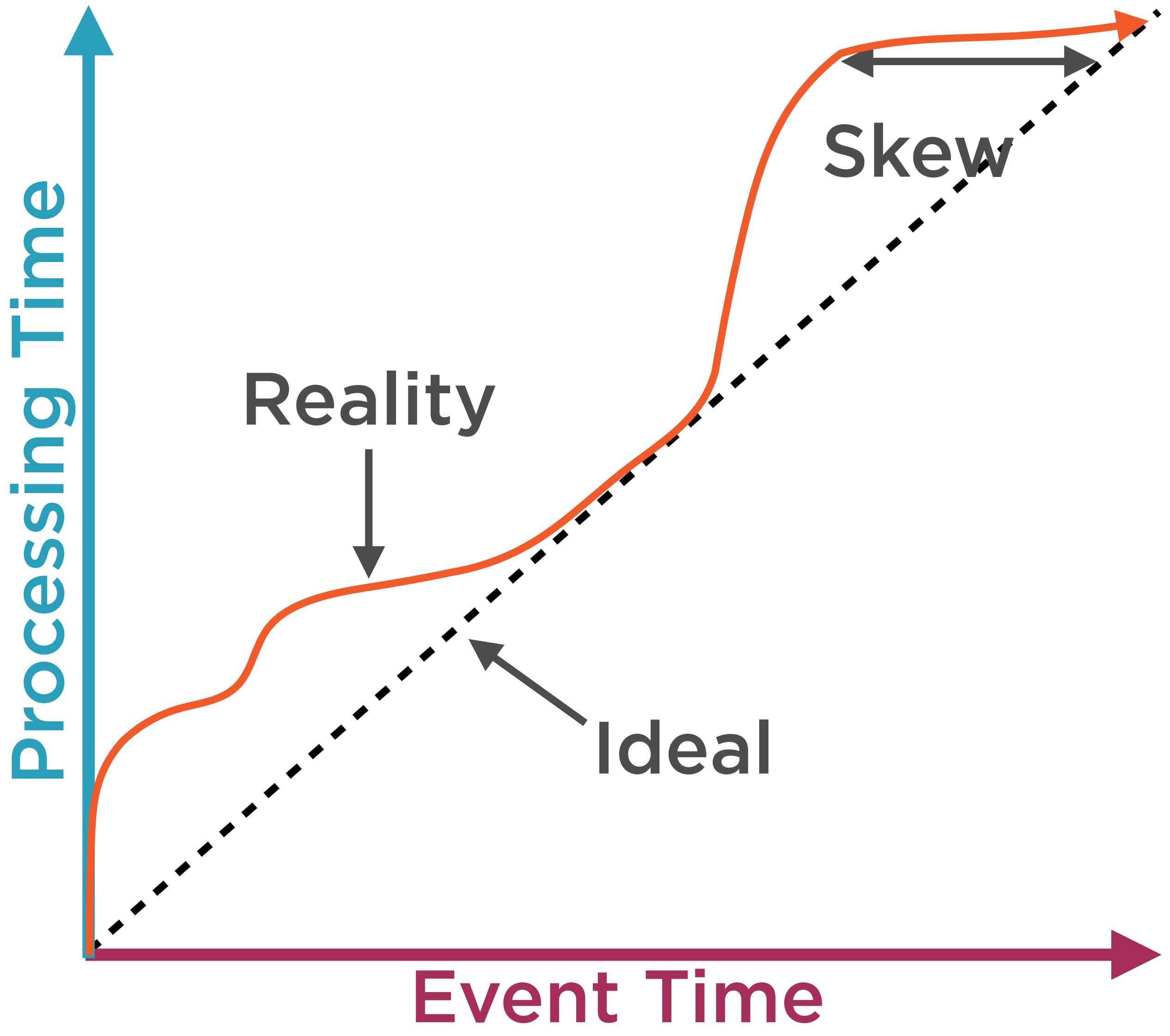
The system time of the machine **processing** entities

Chronologically after event time and ingestion time

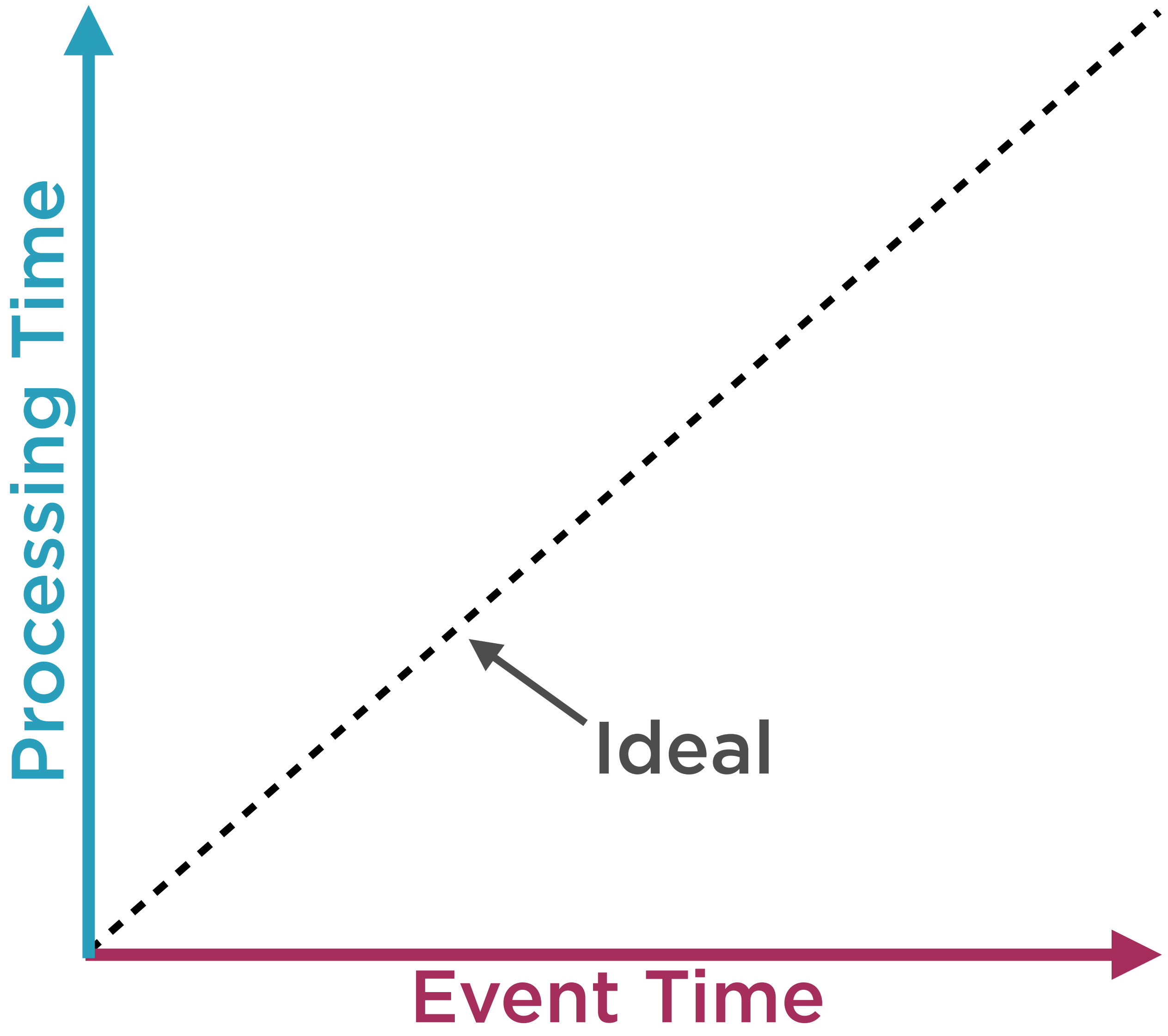
Non-deterministic, depends on when data arrives, how long operations take

Simple, no coordination between streams and processors

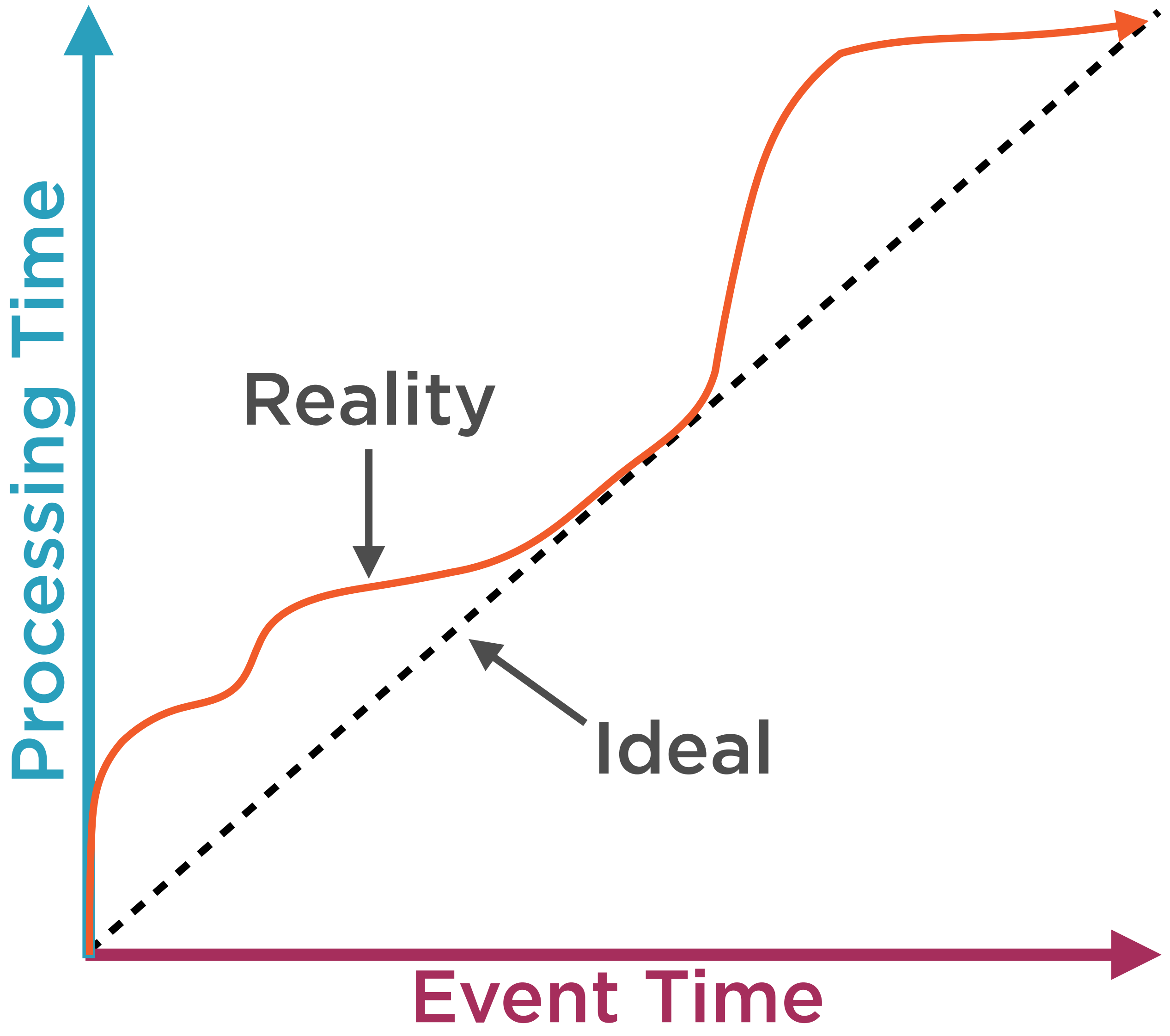
# Event Time vs. Processing Time



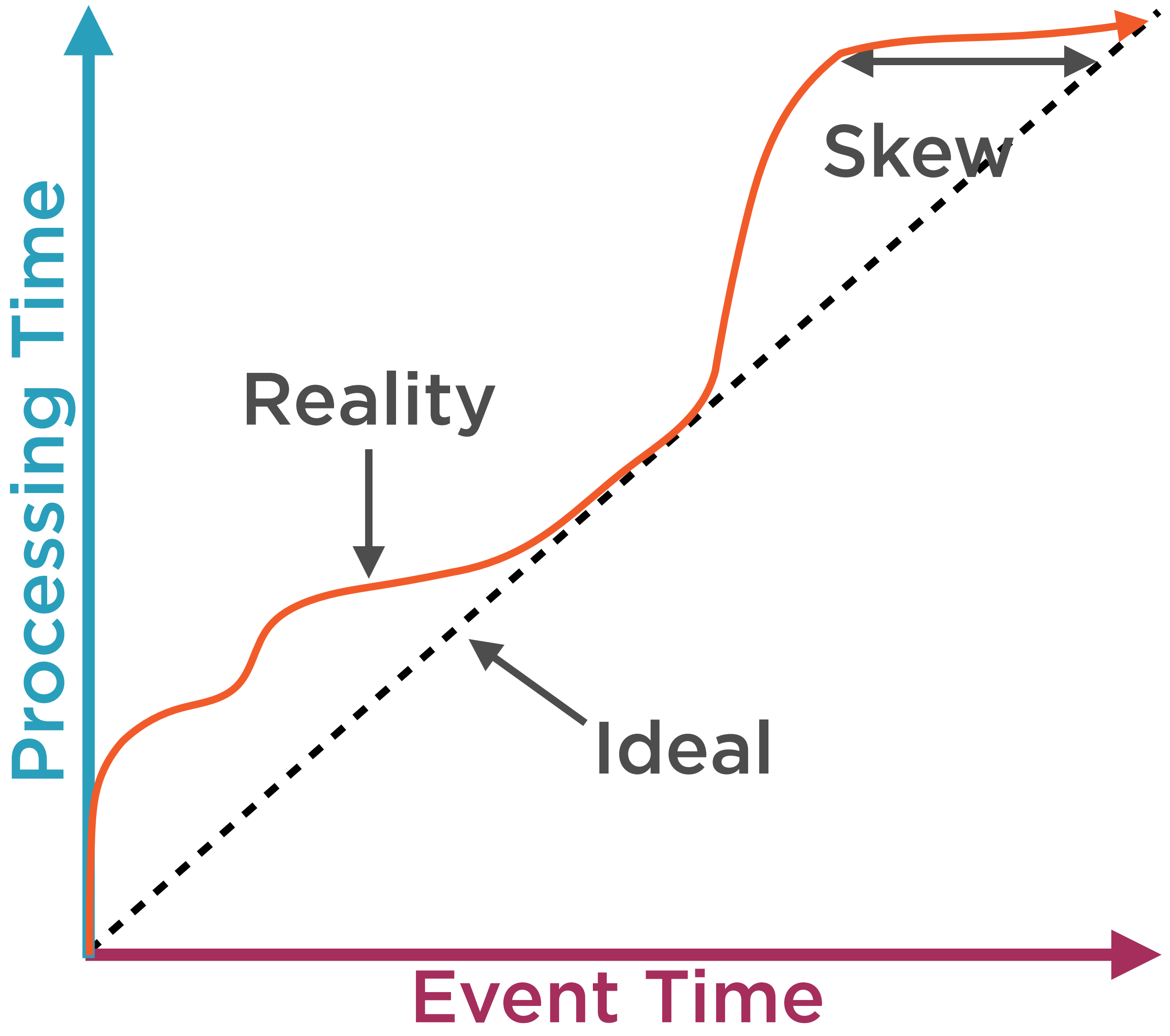
# Event Time vs. Processing Time



# Event Time vs. Processing Time



# Event Time vs. Processing Time

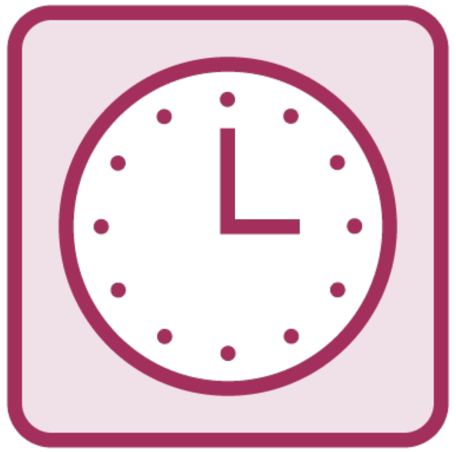


# Demo

**Exploring global windows, tumbling (fixed) windows, and sliding windows**

# Watermarks and Late Data

# How Late Is Late?



**Class At 9 am**

Class starts when clock strikes 9

**Is 9:01 Late?**

Realistically, at least some folks are going to be a minute late

**Is 10:10 late?**

A student is an hour late - allow in or send back?

# How Late Is Late?

The professor “knows” what lateness is reasonable

Students entering within this reasonable lateness are late but OK

Students entering after this reasonable lateness are too late

“Allowed Lateness”

# How Late Is Late?

**Dealing with excessive lateness**

**A student is too late**

- Option 1: Send back home
- Option 2: Allow in, continue class
- Option 3: Allow in, restart class(!)

# Watermarks and Late Data

The system "knows" what lateness is reasonable

Data entering within this reasonable lateness is late but OK

Data entering after this reasonable lateness is too late

# Watermarks and Late Data

**Watermark**

Threshold of allowed lateness (event time)

Data entering within this reasonable lateness is late but OK

Data entering after this reasonable lateness is too late

# Watermarks and Late Data

**Watermark**

Threshold of allowed lateness (event time)

**Late Data**

Data within watermark is aggregated

**Data entering after this reasonable lateness is too late**

# Watermarks and Late Data

## Watermark

Threshold of allowed lateness (event time)

## Late Data

Data within watermark is aggregated

## Dropped Data

Data outside watermark is dropped

```
windowedCounts = words.groupBy(
    window(words.timestamp, "10 minutes", "5 minutes"),
    words.word
).count()
```

# Simple Group-by Without Watermark

**Count words in each sliding window of width 10 minutes, sliding by 5 minutes**

```python
windowedCounts = words \
    .withWatermark("timestamp", "12 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```

# Simple Group-by With Watermark

**We define the watermark i.e. lateness threshold to be 12 minutes**

```
windowedCounts = words \
    .withWatermark("timestamp", "12 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```

# Simple Group-by With Watermark

**Now window triggering will be delayed by 12 minutes**

# Watermark

System generated or user specified

If, say network speed drops, watermark can become more lenient

Lateness = Processing Time - Event time

# Watermarks and Output Modes

**Append mode: Window not triggered at all until watermark elapses**

- No partial updates

**Update mode: Window will trigger even before watermark elapses**

- Engine will keep partial counts

**Complete mode: Can not be used with watermarks**

# Watermarks and Output Modes

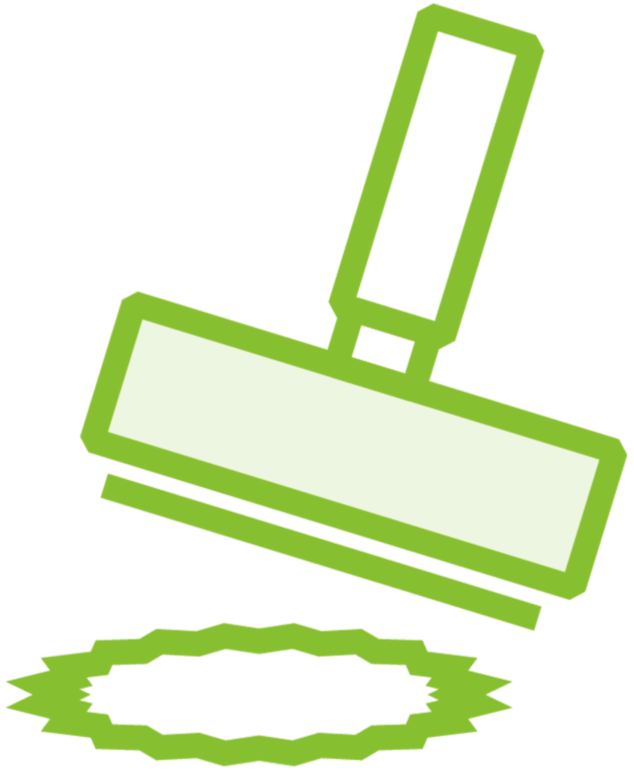
No complete-mode queries

Aggregation must be event-time, or event-time window

`.withWatermark` must be called on same timestamp column as aggregate

`.withWatermark` must be called before the aggregation

# One-way Guarantee



All data before watermark will definitely not be dropped

All data after watermark may or may not be dropped

More delayed the data, less likely the engine is to process the data

# Demo

Using watermarks to allow late data

# Demo

Using UDFs (user-defined functions) to transform data

# Summary

Windowing operations on streams

Sliding and tumbling windows

Event time vs. processing time

Aggregation operations on windows

UDFs on streaming data

**Up Next:**

Working with Streaming Joins