# Streaming Systems

# Streaming Systems: Objectives

1. Theory
   - Define:
     - Data latency
     - Windowing
     - Watermarking
     - Triggers
     - Bounded data
   - Understand exactly-once vs. at-least-once semantics
   - Understand how streaming systems handle late data
   - Understand which windowing strategy should be used based on use case

# Streaming Systems: Objectives

2. Practice
   - Lambda/kappa architecture
   - Articulate when a use case requires streaming architectures

# Streaming Systems: Table of Contents

# The End

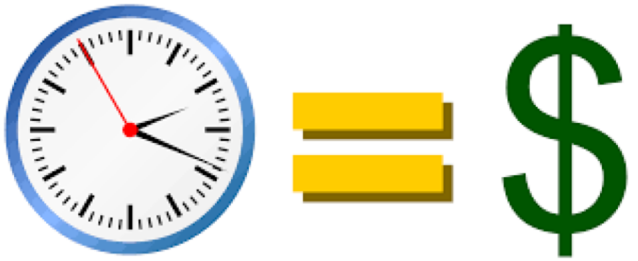
# Data Latency

# Real-Time, Near-Time, Some-Time

- Real-time
  - We need the answer as part of the process to process an event.
- Near-time
  - We need the answer shortly after an event (less than 10 seconds or less than five minutes, based on use case).
- Some-time
  - We need the answer in the future (hourly, daily, monthly, yearly).

# Real-Time, Near-Time, Some-Time Concerns

- Real-time systems are much more expensive than some-time systems.
- Determine the criteria for when to use a real-time system to justify the expense.
- Real-time systems also carry additional complexity, as we will discuss.



# Why Do We Care About Streaming?

- Having the answers earlier can provide an advantage in many ways.
  - Efficiency
  - Strategically
  - Responsiveness
- When we have a need for streaming systems, it becomes painful to deal with the batch systems.
- Did that update affect my users?
  - If so, roll it back
- Is my brand under attack on social media?
  - Get PR involved earlier
- Is there a good deal on the stock market?
  - Buy/sell

# The End

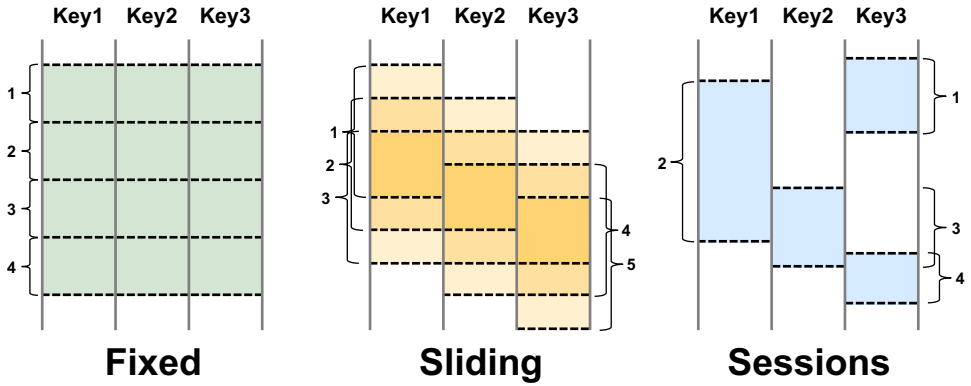
# Streaming vs. Batch

# Bounded vs. Unbounded Data

- Bounded data have a fixed volume to process
- Theoretically, all batch workload
  - Data might have to be split to be able to be processed
  - Directly in line with MapReduce
- Unbounded data do not have a fixed volume
- Practically all data are bounded
  - Can’t have infinite data
- Focus on the volume of data that the system needs to handle

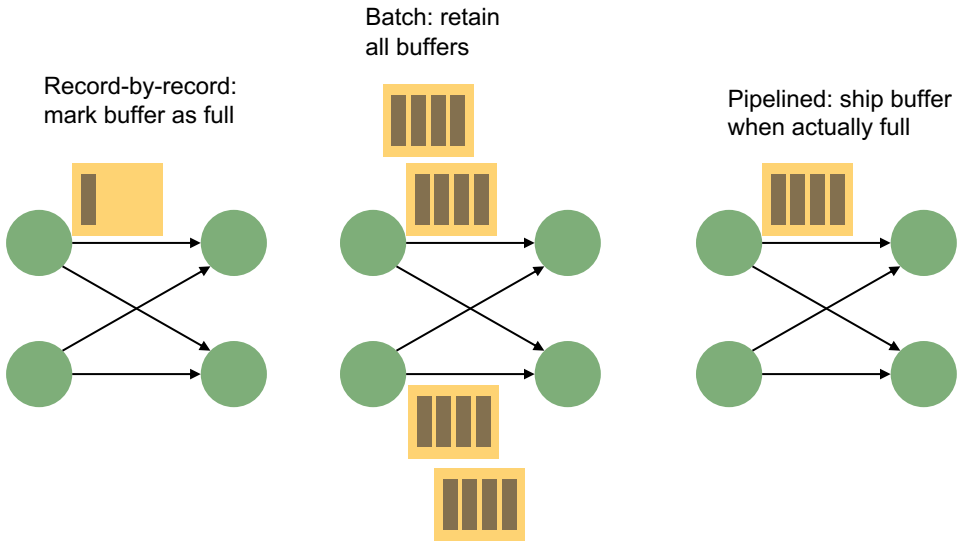
# Windows

- Chop up the data based on a time interval such as event or processing time
  - More on this later
- Streaming systems are designed to deal with complexities that arise in windowing when it comes to completeness of data
- A few ways to window data
  - Fixed
  - Sliding
  - Sessions



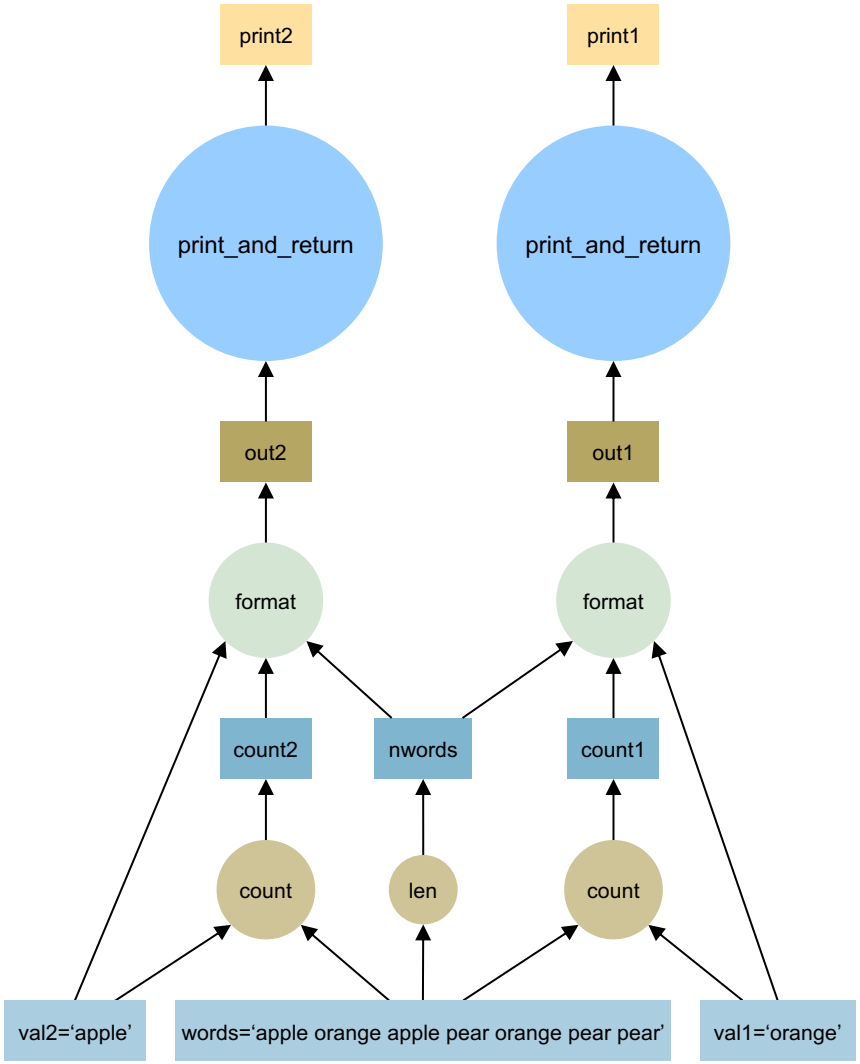
# Runtimes

- When do we ship data for processing?
- There are several options
  - Record-by-record
  - Pipelined
  - Batch
- Each has their own advantages, and different systems implement management of these runtimes differently



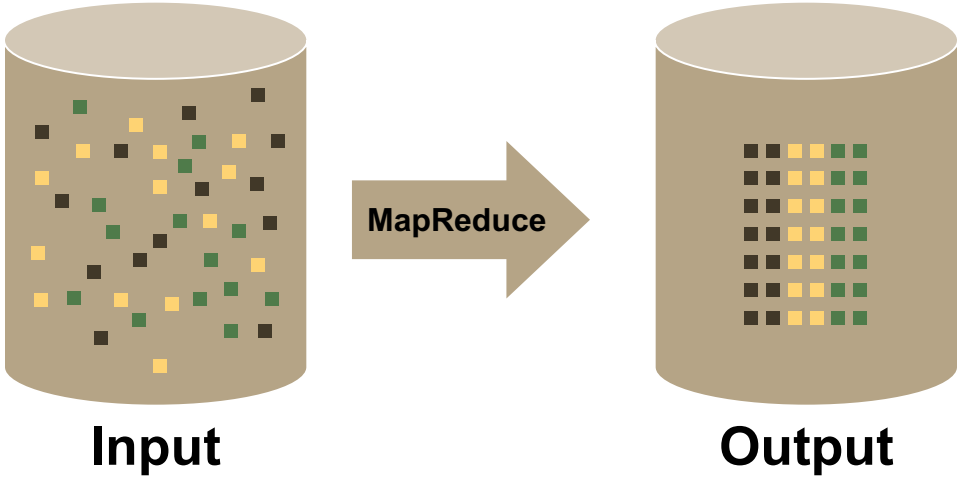
# DAGs

- Directed acyclic graphs (DAGs) are sequences of events that need to follow each other that allow for splitting of independent work as we've seen before
- Batch DAGs run each step that's not blocked as it finishes processing
- Streaming DAGs takes this one step further and pushes a given data buffer through the DAG
- Compliments the runtimes we just discussed
- Every batch process starts by streaming in a file



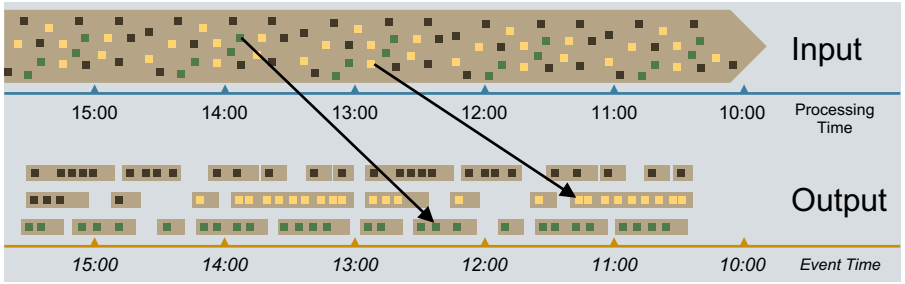
# What Is Batch Processing?

- Bounded data
- Global window
  - Can be broken up into fixed windows
- “Batch” runtime
  - Retain all buffers of data
- Blocking DAGs



# What Is Stream Processing?

- Unbounded data
- Nonglobal windows
  - Fixed
  - Sliding
  - Session
- “Record-by-record” runtime
  - Can also do micro-batching with “pipelined” runtimes
- Streaming DAGs
  - Batch does this in disguise while reading inputs



# The End

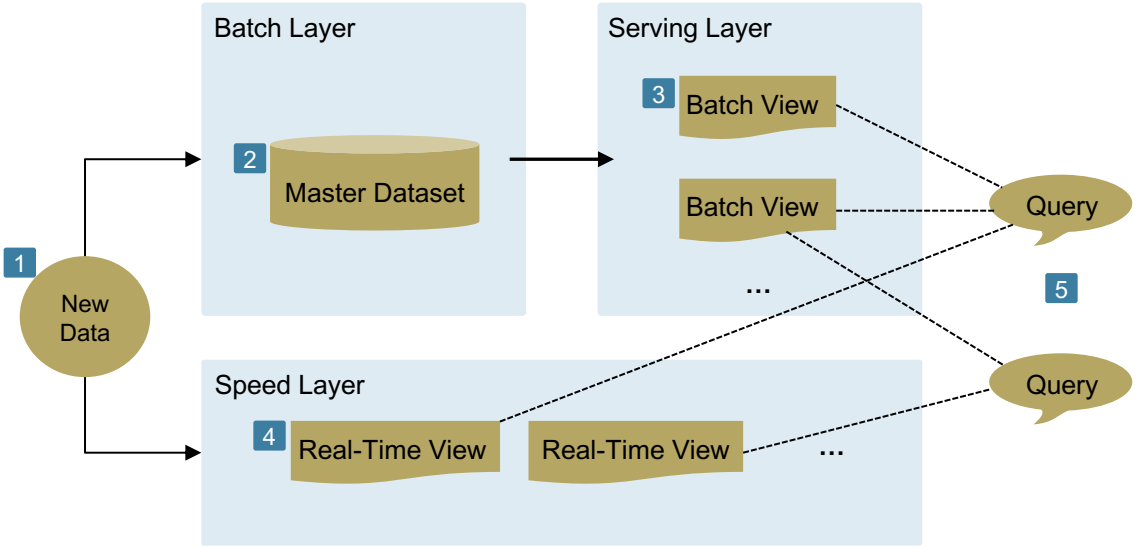
# Architectures

# Why Architecture?

- To make a system work, we need to think about how we should put it in place
- Streaming systems typically follow one of two approaches:
  1. Lambda architecture
  2. Kappa architecture
- These approaches have different trade-offs and have different advantages associated with them
- A good system is built on a solid foundation

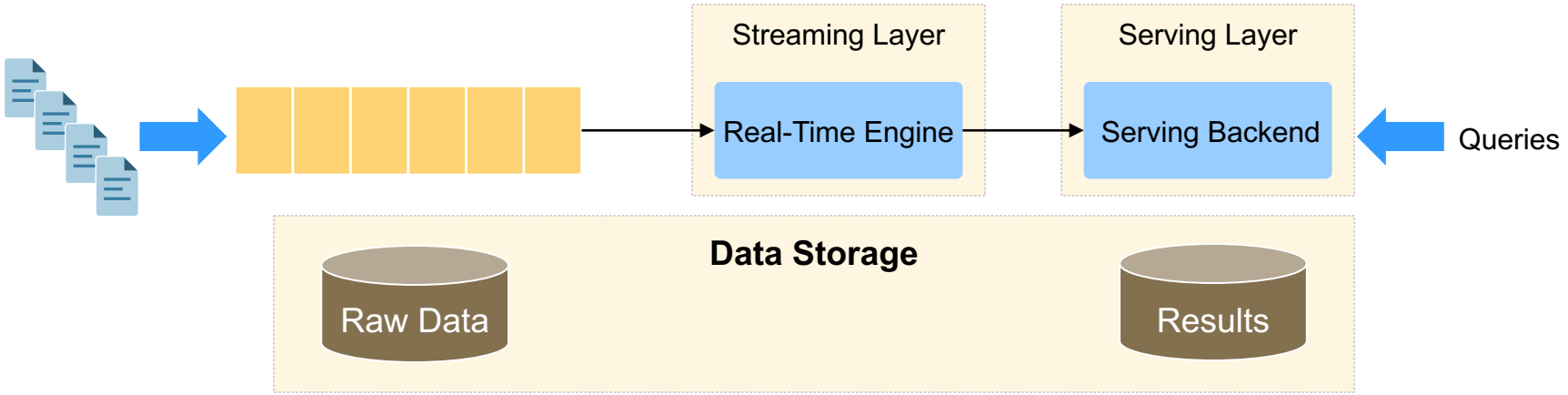
# Lambda Architecture

- Manage two systems—one batch and one streaming.
  - A lot of code to manage
- Streaming systems historically did not handle missing or late data very effectively.



# Kappa Architecture

- The world is a stream of data
- Offload data to batch for when needed
- Serve out of the streaming layer
- Much simpler to manage
- How do you deal with the gotcha?
  - Reconciliation process for late/missing data



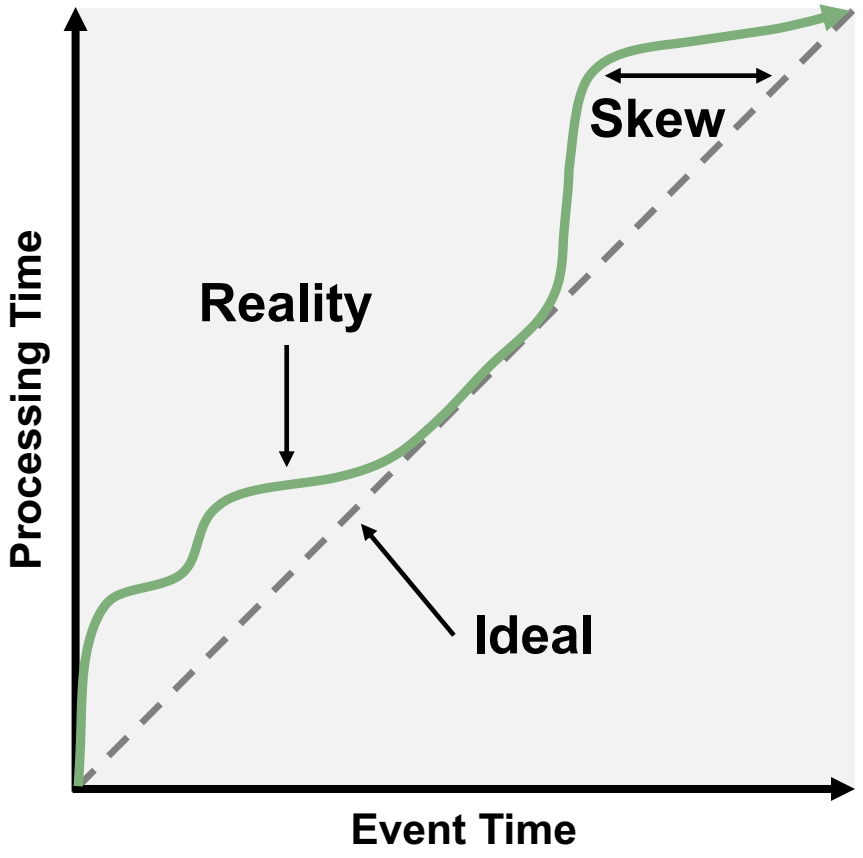
# Exactly-Once vs. At-Least-Once Processing

## Understanding Streaming Semantics


| At most once | At least once | Exactly once |
|---|---|---|
| Message pulled once | Message pulled one or more times; processed each time | Message pulled one or more times; processed once |
| May or may not be received | Receipt guaranteed | Receipt guaranteed |
| No duplicates | Likely duplicates | No duplicates |
| Possible missing data | No missing data | No missing data |

# Real-World Performance Considerations

- No system has perfect communication.
- Latency will always be a factor.
- What if a user walks into an elevator?
- How do we handle missing data?

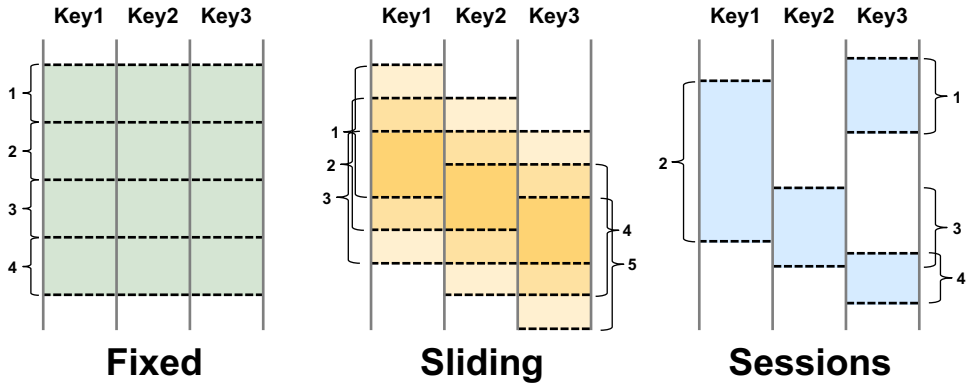


# The End

# Considerations

# Windowing

- Fixed
  - GroupBy
- Sliding
  - Look back five minutes from end of each window
  - Tracking a metric over time such as moving average of financial transactions
- Sessions
  - Dynamic window example
  - Typically used for user behavior
  - Trigger to end session when no messages for some fixed period

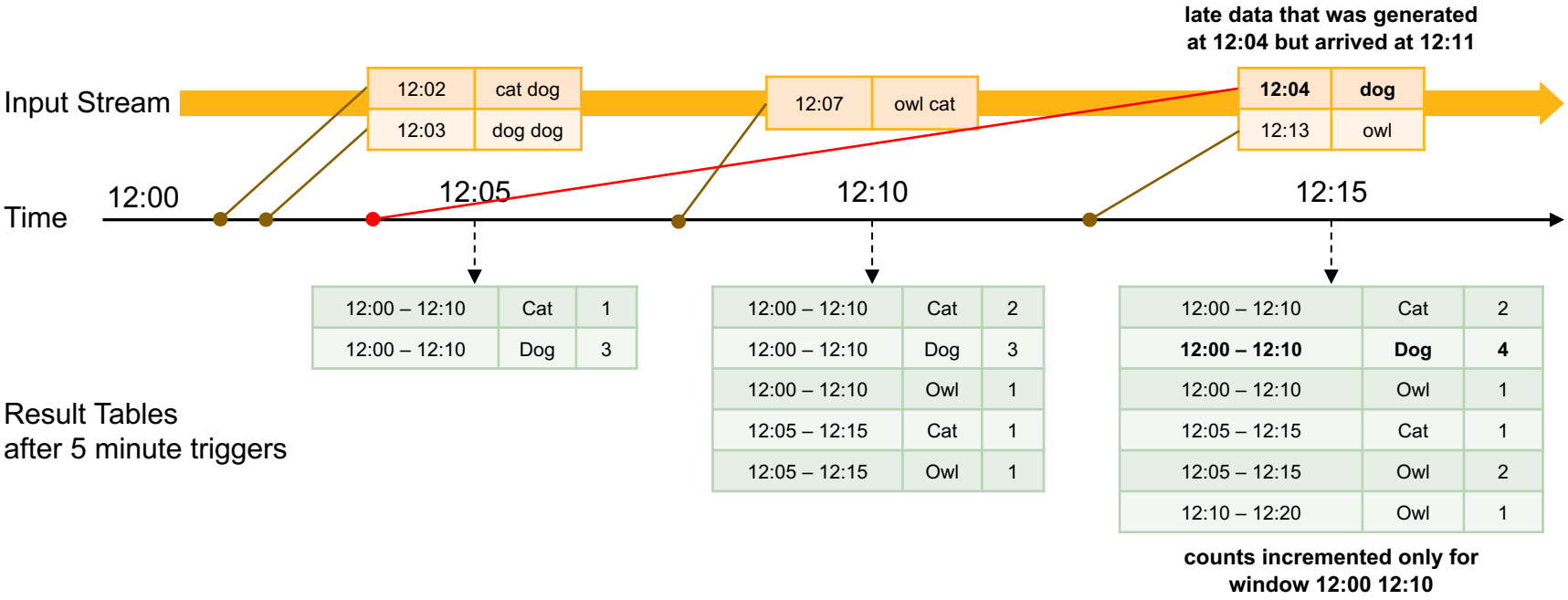


# Triggers

- When to do the action
- Allows for us to view the output of a window again
- Typical built-in triggers include:
  - Count above N
  - Event time
  - Processing time
  - Run one buffer
- Different systems have different semantics, and the triggers will be based on those

# Late Data

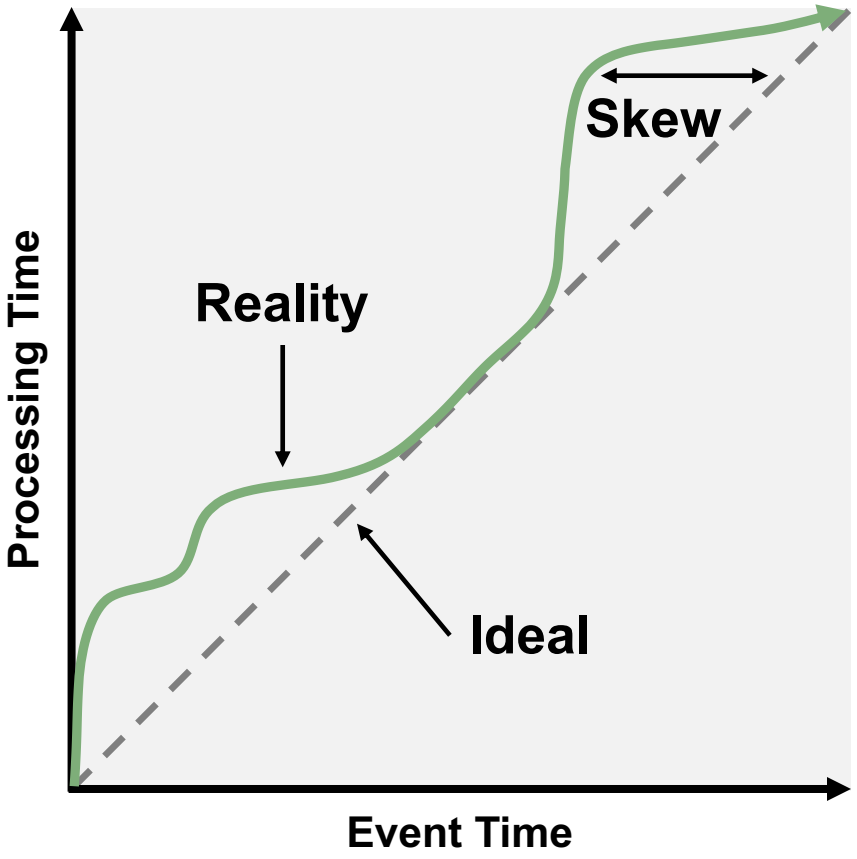
- Sometimes data can show up late
- The system needs to be able to handle this
- Example of a fixed window aggregation and five-minute triggers with late data:



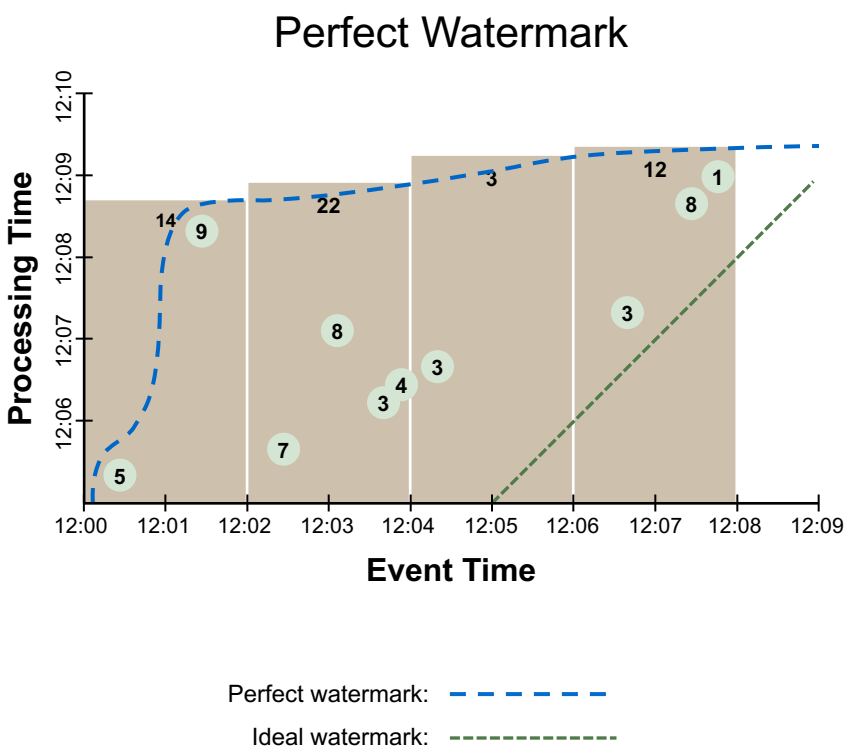

Late data handling in Windowed Grouped Aggregation

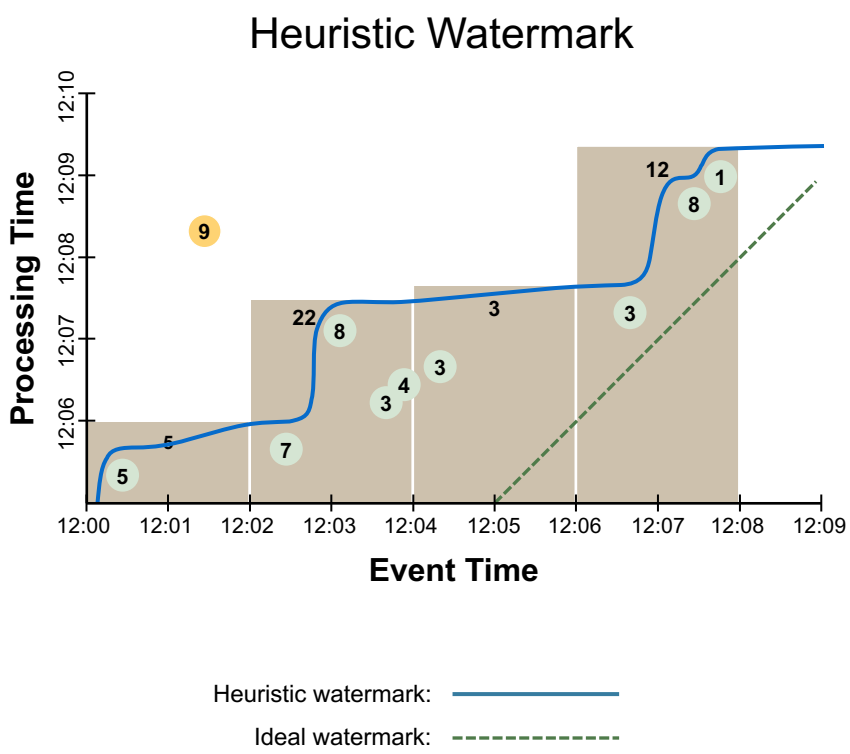
# Watermarking

- We need to be able to keep track of where we are in the processing.
- The world is cruel, and we never have a perfect system.
- Watermarking takes a point in processing time and maps it to an event time.
  - It would be the red line in the figure.
- A perfect watermark would be the diagonal where processing time is equal to event time.
- Data that come in late would likely fall above the red line.



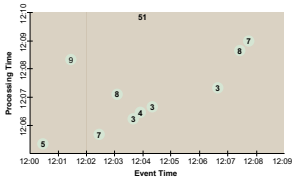
# Watermarking



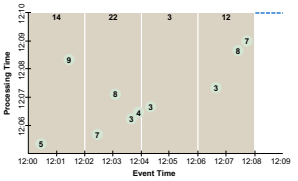



# Going Forward

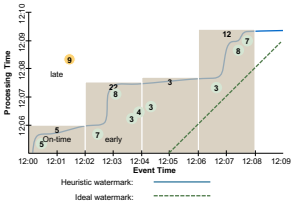
- This class is meant for an introduction to streaming.
- If you want to get deeper into these ideas, review the following as a first step:
  - Event time vs. processing time for watermarking and windowing
  - Perfect vs. heuristic watermarking



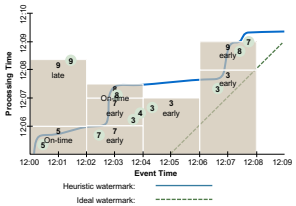
**Classic batch**
Listing 1/Figure 2



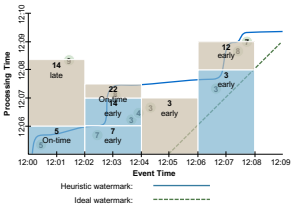
**Fixed windows batch**
Listing 2/Figure 4



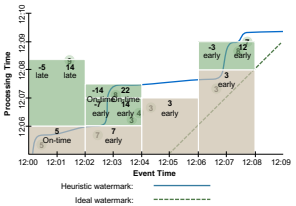
**Fixed windows streaming watermark**
Listing 2/Figure 6



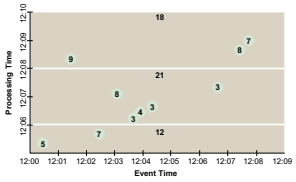
**Early/late discarding**
Listing 7 / Figure 9



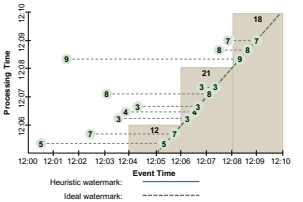
**Early/late accumulating**
Listing 4 and 5 / Figure 7



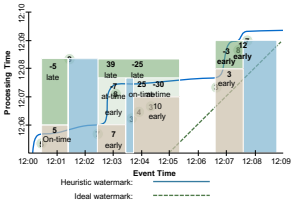
**Early/late retracting**
Listing 8 / Figure 10



**Processing – time (triggers)**
Listing 9 / Figure 14



**Processing – time (ingress time)**
Listing 10 / Figure 15



**Sessions**
Listing 11 / Figure 17

# The End

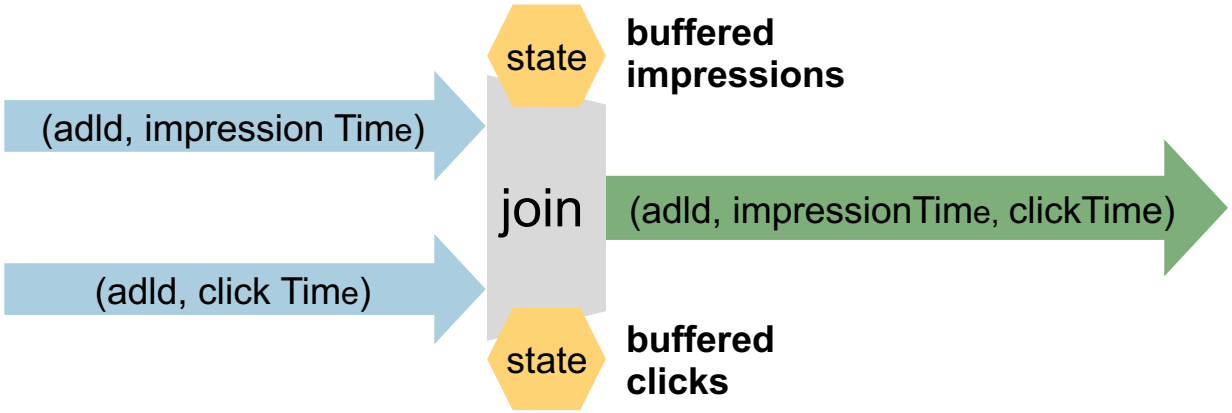
# Use Cases

# Ad Monetization

- Advertisers want information now to ensure that their campaigns are attracting customers and generating revenue
- Two streams of data:
  1. Ads
  2. Clicks
- Need to join the streams, but there are several considerations

The Case for Stream-Stream Joins:

Ad-Monetization



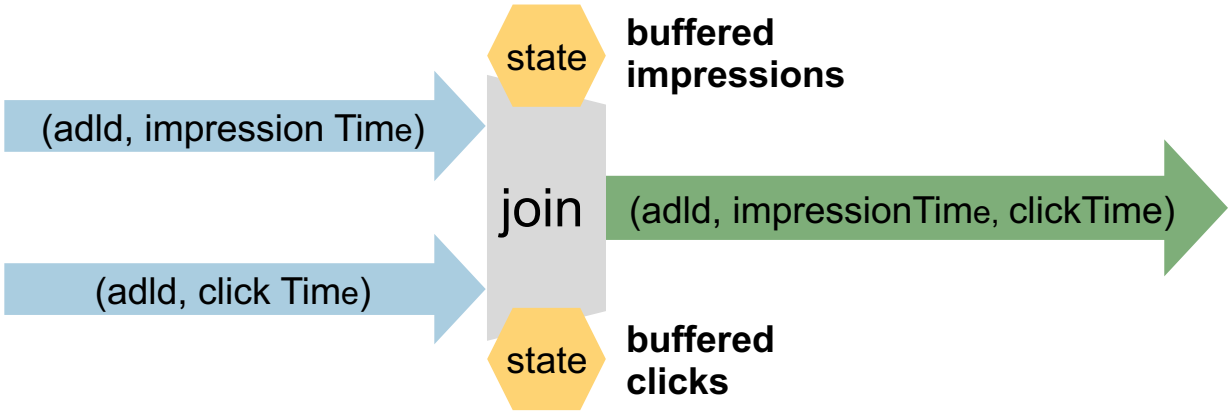

Stream-stream join use case: Ad-Monetization (joining ad clicks to impressions)

# Ad Monetization

- What if there never was a click?
- How long should I wait for a click?
- What are the infrastructure timings?

The Case for Stream-Stream Joins:

Ad-Monetization

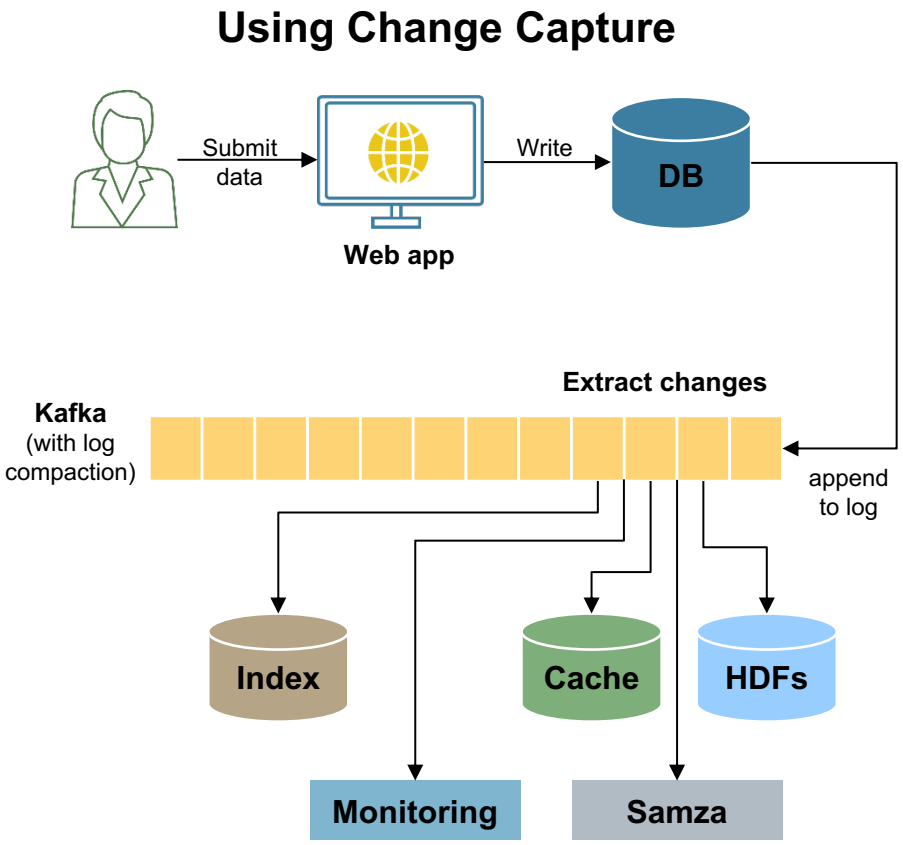



Stream-stream join use case: Ad-Monetization (joining ad clicks to impressions)

# Change Data Capture

- You have a database on premises that is very sensitive to nonapplication workloads running against it
- To enable analytics against the application data, you need to replicate these data somewhere else
- Creating a stream of transactions committed to the database allows you to replicate the database somewhere else, and it is called change data capture (CDC)
- Single stream and one of the oldest use cases for streaming data

**How to Integrate Your Databases**



# The End

# Summary

# Conclusion

- We won't be experts in a day.
- Streaming can be complicated, so limit its use to where it makes sense.
  - Additionally, streaming systems are more expensive than batch systems.
- Understand the basics of watermarking and how it relates to event time vs. processing time.
- Understand the semantics to consider for exactly-once vs. at-least-once processing.
- Speak to example use cases that involve streaming systems.

# Streaming Systems: Outcomes

1. Theory
   - Define
     - Data latency
     - Windowing
     - Watermarking
     - Triggers
     - Bounded data
   - Understand exactly-once vs. at-least-once semantics
   - Understand how streaming systems handle late data
   - Understand which windowing strategy should be used based on use case

# Streaming Systems: Outcomes

2. Practice
   - Lambda/kappa architecture
   - Articulate when a use case requires streaming architectures

# The End