CS 744: DATAFLOW

Shivaram Venkataraman

Fall 2022

ADMINISTRIVIA

Grading In Progress

- Assignment 2
- Course project proposal

About the Midterm

MID-SEMESTER FEEDBACK

More frequent change in paper reading group could be beneficial.
It was hard to find time for paper discussions, given the random assignments and fixed teammates
I would have liked to have chosen my reading group. I find it somewhat difficult to organize
…

I would like if more time was given for exam
Exams should be longer in time duration, very less time to actually think deep
…

It would've been nice if we had to … list any questions we had about the paper in the paper review form.

More time for in-class discussions.
Concentrating for 1.25 hours continuously is hard. A break in the middle for discussion?

maybe in-class quizzes??

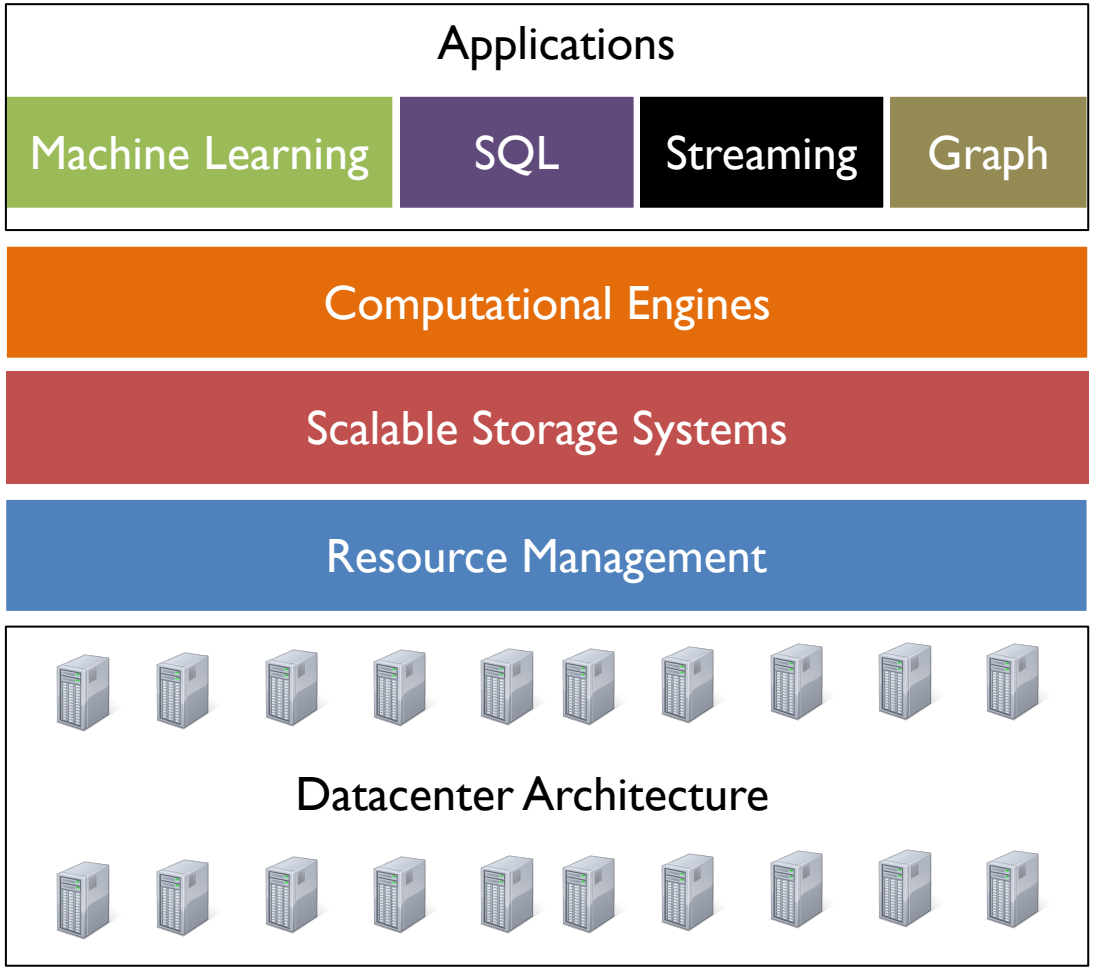
PAPER REVIEW GROUPS

Optional from next class (Nov 3rd)

Re-shuffle groups to have new "suggested" groups

You can do any of

- Use (part of) suggested group
- Discuss with your own group
- Read on your own!


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture

DATAFLOW MODEL (?)

MOTIVATION

Streaming Video Provider

- How much to bill each advertiser ?
- Need per-user, per-video viewing sessions
- Handle out of order data

Goals

- Easy to program
- Balance correctness, latency and cost

APPROACH

Separate user API from execution

Decompose queries into

- What is being computed
- Where in time is it computed
- When is it materialized
- How does it relate to earlier results

STREAMING VS. BATCH

Streaming

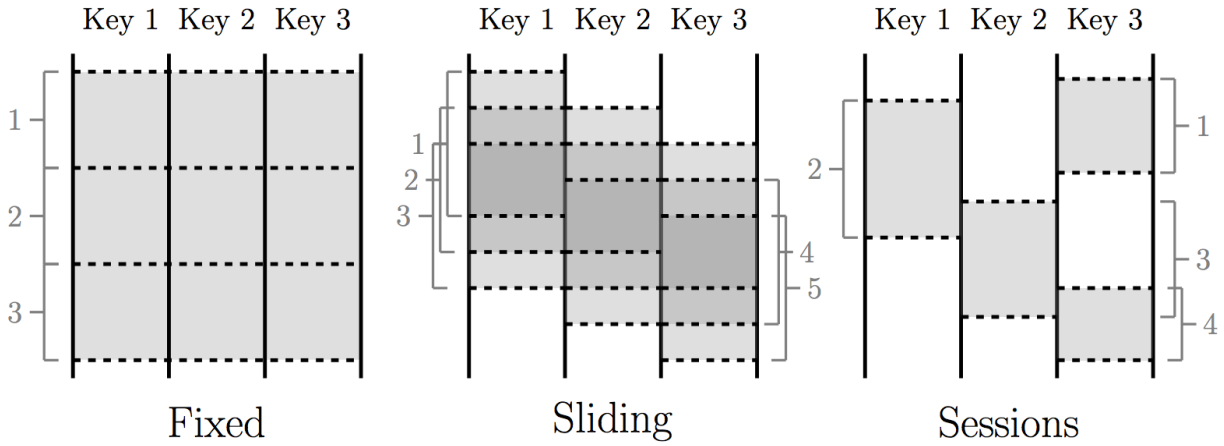
Batch

TIMESTAMPS

Event time:

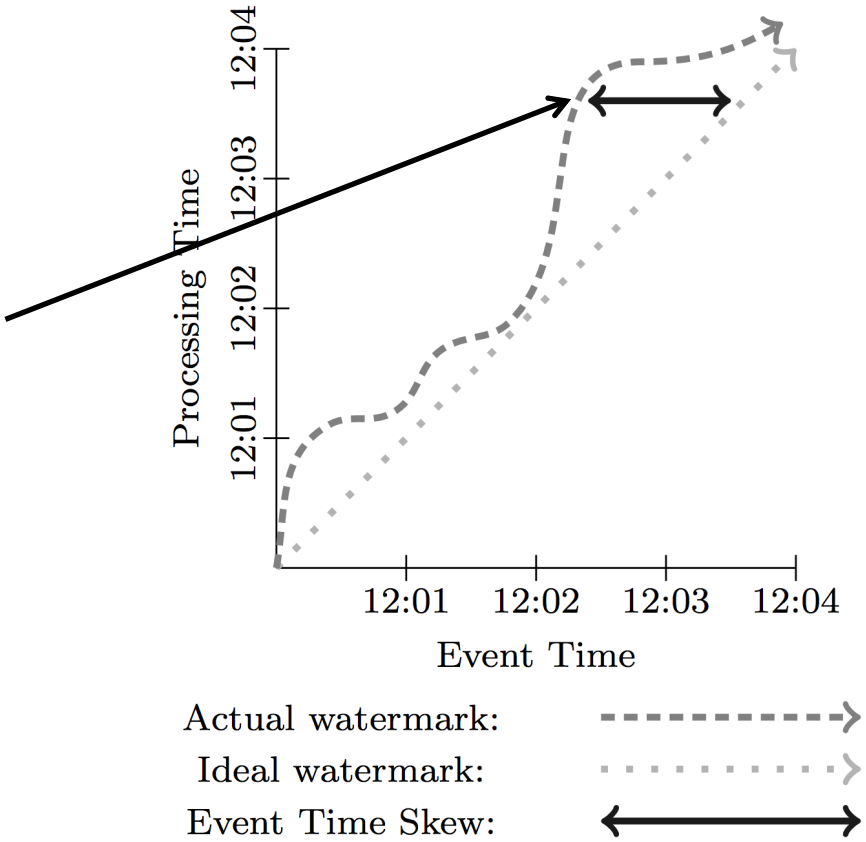
Processing time:

WINDOWING


Key 1
Key 2
Key 3
1
2
3
Fixed
Key 1
Key 2
Key 3
1
2
3
4
5
Sliding
Key 1
Key 2
Key 3
2
1
3
4
Sessions

WATERMARK OR SKEW

System has processed all events up to 12:02:30



API

ParDo:

GroupByKey:

Windowing

AssignWindow

MergeWindow

EXAMPLE

$(k_1, v_1, 13{:}02, [0, \infty))$,
$(k_2, v_2, 13{:}14, [0, \infty))$,
$(k_1, v_3, 13{:}57, [0, \infty))$,
$(k_1, v_4, 13{:}20, [0, \infty))$

↓ *AssignWindows(Sessions(30m))*

$(k_1, v_1, 13{:}02, [13{:}02, 13{:}32))$,
$(k_2, v_2, 13{:}14, [13{:}14, 13{:}44))$,
$(k_1, v_3, 13{:}57, [13{:}57, 14{:}27))$,
$(k_1, v_4, 13{:}20, [13{:}20, 13{:}50))$

↓ *DropTimestamps*

$(k_1, v_1, [13{:}02, 13{:}32))$,
$(k_2, v_2, [13{:}14, 13{:}44))$,
$(k_1, v_3, [13{:}57, 14{:}27))$,
$(k_1, v_4, [13{:}20, 13{:}50))$

GroupByKey

$(k_1, [(v_1, [13{:}02, 13{:}32)),$
$(v_3, [13{:}57, 14{:}27)),$
$(v_4, [13{:}20, 13{:}50))]),$
$(k_2, [(v_2, [13{:}14, 13{:}44))])$

↓ *MergeWindows(Sessions(30m))*

$(k_1, [(v_1, [\mathbf{13{:}02}, \mathbf{13{:}50})),$
$(v_3, [13{:}57, 14{:}27)),$
$(v_4, [\mathbf{13{:}02}, \mathbf{13{:}50}))]),$
$(k_2, [(v_2, [13{:}14, 13{:}44))])$

↓ *GroupAlsoByWindow*

$(k_1, [([\mathbf{v_1}, \mathbf{v_4}], [13{:}02, 13{:}50)),$
$([\mathbf{v_3}], [13{:}57, 14{:}27))]),$
$(k_2, [([\mathbf{v_2}], [13{:}14, 13{:}44))])$

↓ *ExpandToElements*

$(k_1, [v_1, v_4], \mathbf{13{:}50}, [13{:}02, 13{:}50))$,
$(k_1, [v_3], \mathbf{14{:}27}, [13{:}57, 14{:}27))$,
$(k_2, [v_2], \mathbf{13{:}44}, [13{:}14, 13{:}44))$

TRIGGERS AND INCREMENTAL PROCESSING

Windowing: where in event time are data grouped

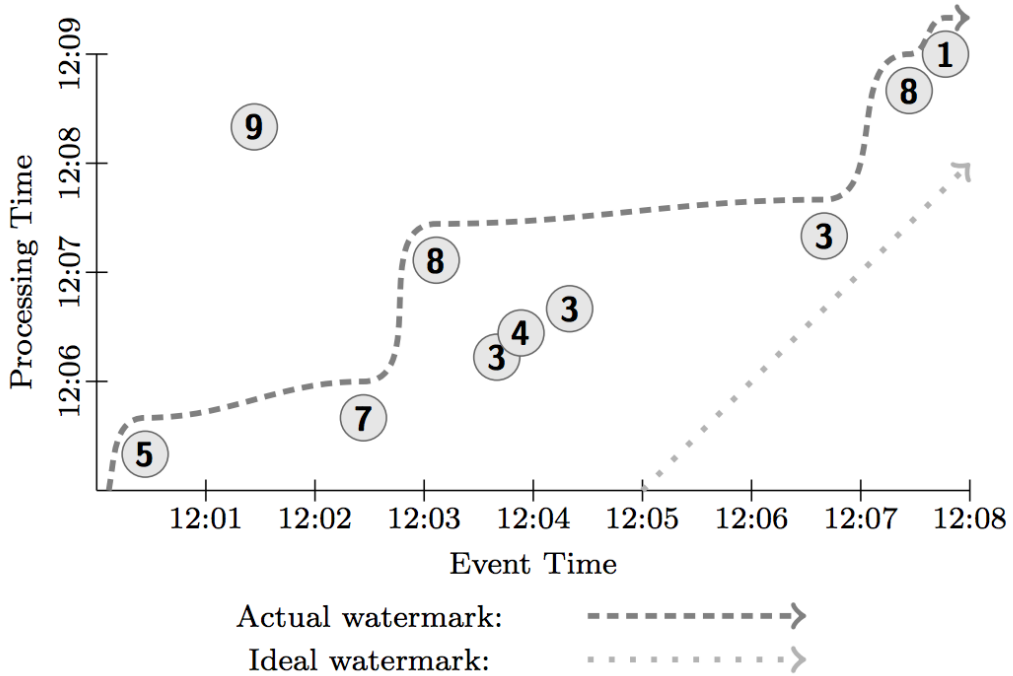
Triggering: when in processing time are groups emitted

Strategies

- Discarding
- Accumulating
- Accumulating & Retracting

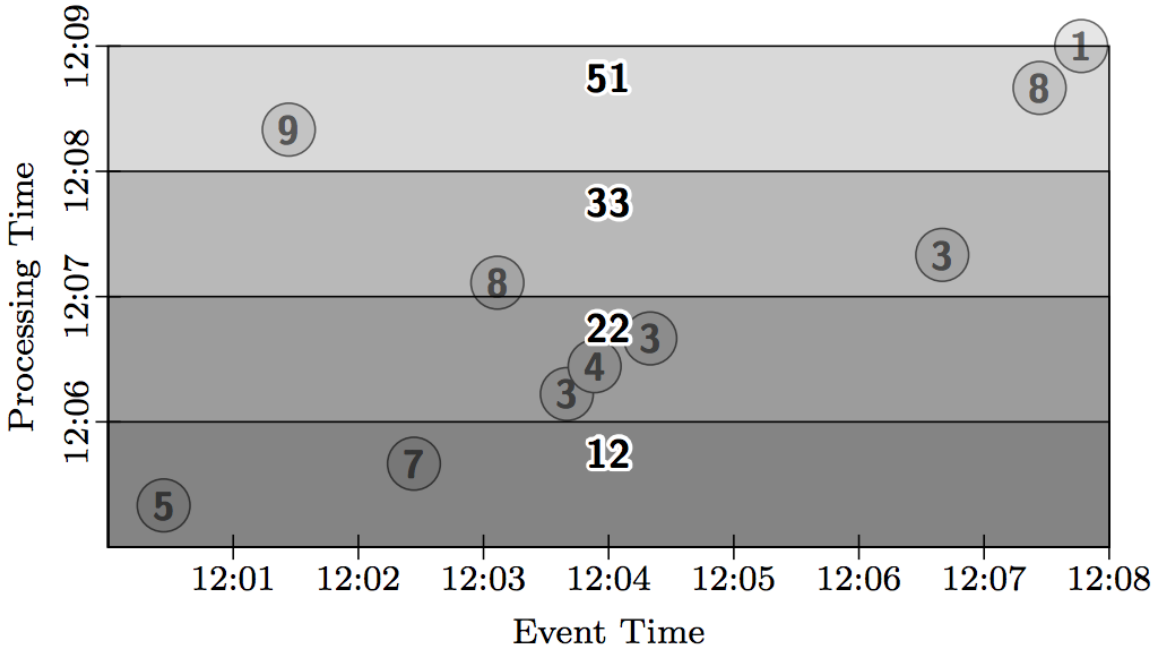
RUNNING EXAMPLE

```
PCollection<KV<String, Integer>> input = IO.read(...);
PCollection<KV<String, Integer>> output =
        input.apply(Sum.integersPerKey());
```



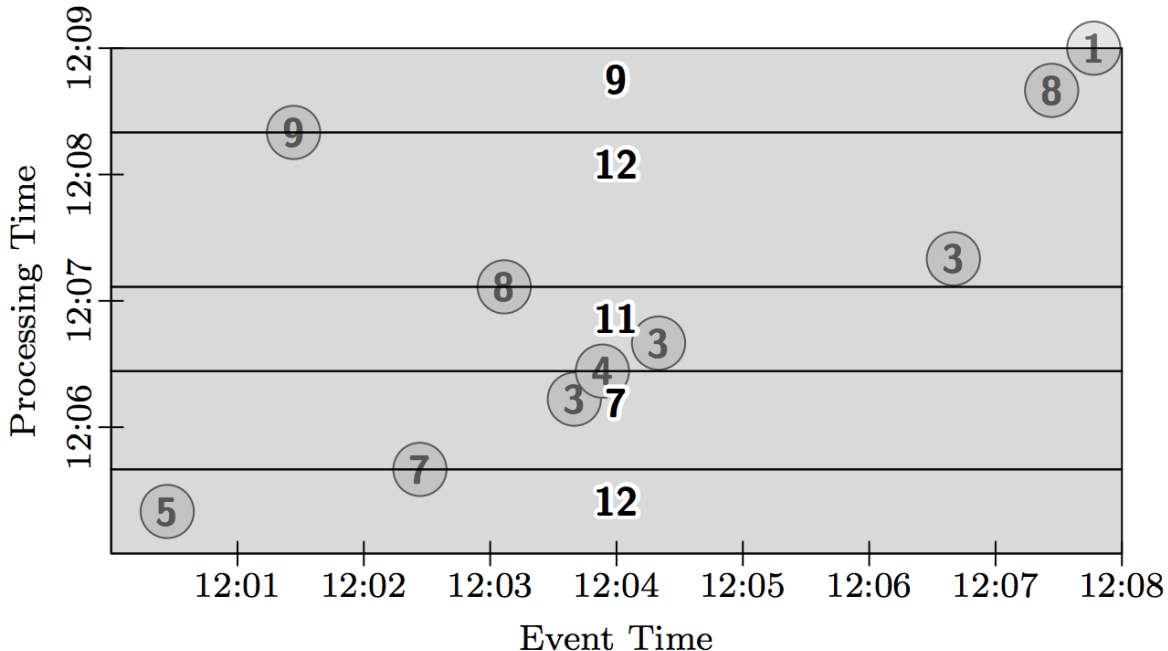
GLOBAL WINDOWS, ACCUMULATE

```
PCollection<KV<String, Integer>> output = input
    .apply(Window.trigger(Repeat(AtPeriod(1, MINUTE))
                .accumulating())
    .apply(Sum.integersPerKey());
```



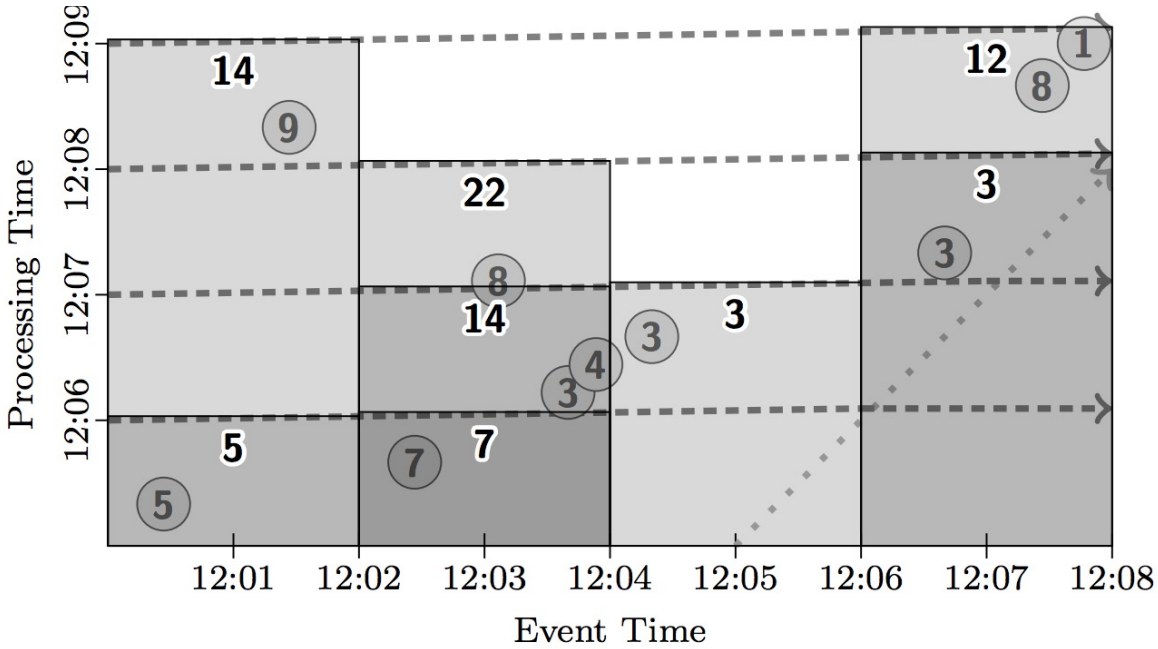
GLOBAL WINDOWS, COUNT, DISCARDING

```
PCollection<KV<String, Integer>> output = input
    .apply(Window.trigger(Repeat(AtCount(2)))
                  .discarding())
    .apply(Sum.integersPerKey());
```



FIXED WINDOWS, MICRO BATCH

```
PCollection<KV<String, Integer>> output = input
    .apply(Window.into(FixedWindows.of(2, MINUTES))
                .trigger(Repeat(AtWatermark())))
                .accumulating())
```



SUMMARY/LESSONS

Design for unbounded data: Don't rely on completeness

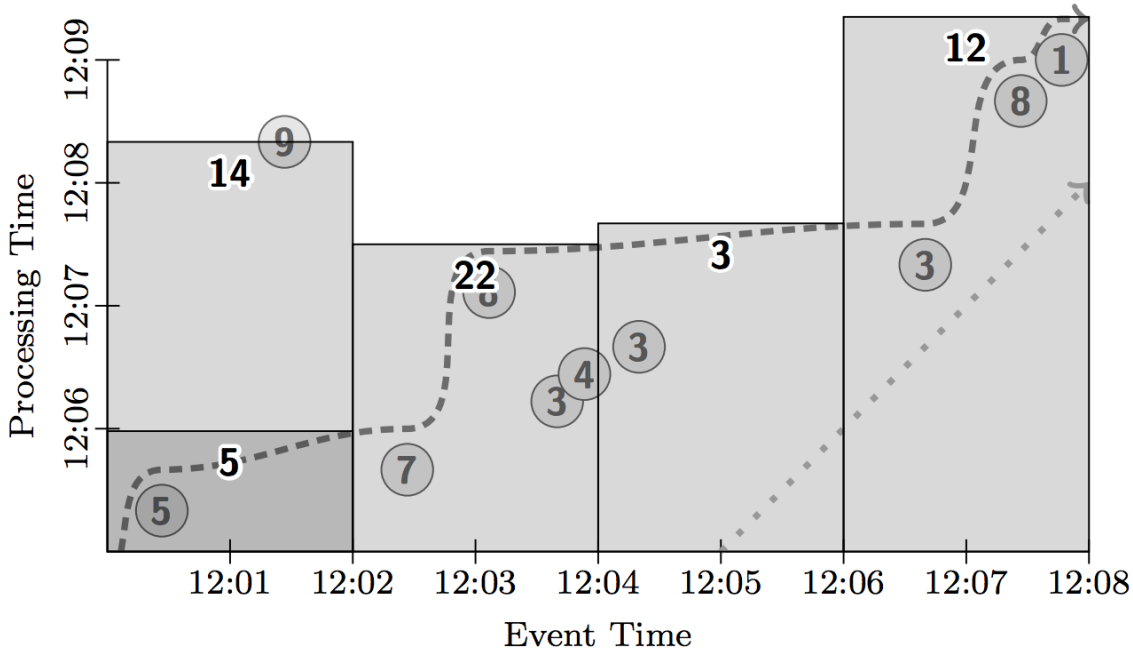
Be flexible, diverse use cases

- Billing
- Recommendation
- Anomaly detection

Windowing, Trigger API to simplify programming on unbounded data

DISCUSSION

https://forms.gle/gUKw3ZP36JjBciABA


Processing Time
12:06
12:07
12:08
12:09
14
9
5
5
7
22
6
3
4
3
3
3
12
1
8
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
Event Time

Consider you are implementing a micro-batch streaming API on top of Apache Spark. What are some of the bottlenecks/challenges you might have in building such a system?

NEXT STEPS

Next class: Flink