Good morning!

CS 744: DATAFLOW

Shivaram Venkataraman

Spring 2024

ADMINISTRIVIA

Grading In Progress

- Course project proposal → steer your project
- Assignment 2 ~ 90% graded
- Midterm

MID-SEMESTER FEEDBACK

"…instead of having everyone submit full discussion answers regardless of what group they're in, maybe just have a single check box on the form…."

"I hope there can be more details about the diagrams in discussion and exams…."

….

checkbox

"… I found midterm exercises were more tricky/challenging …"

"… hard time in the midterm exam…"

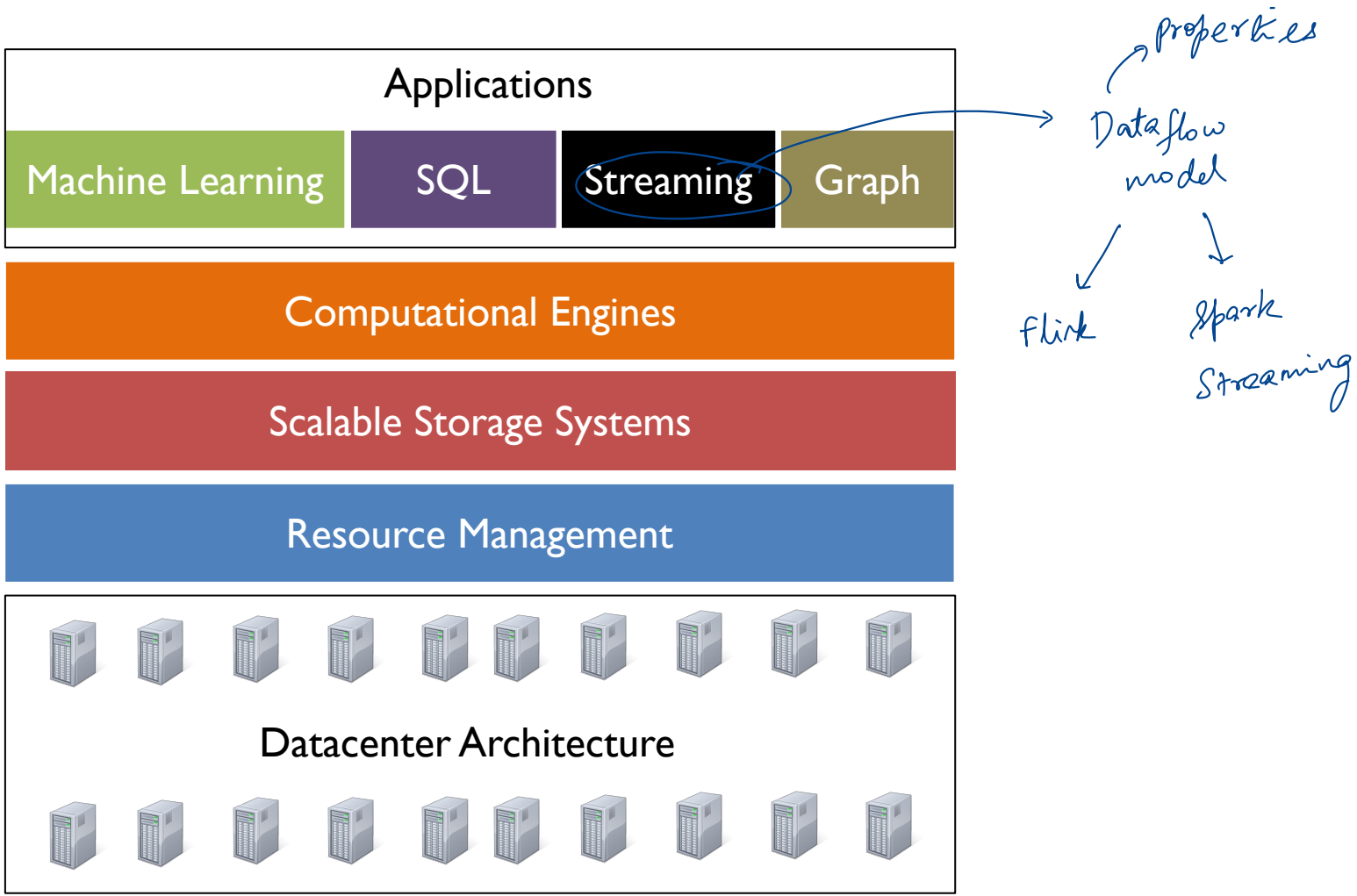
"Its an early morning class"

"The morning timings of the class"

"I want reading groups to talk and understand the paper better."

"… If we had paper reading groups like Distributed Systems course…"

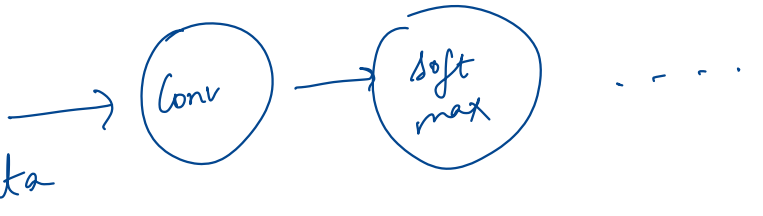
After Spring break


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture
properties
Dataflow model
flink
Spark Streaming

Spark

DATAFLOW MODEL (?)

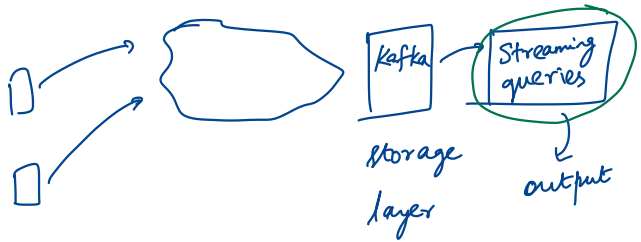
PyTorch



"Unbounded data"

- App logs or logs in Data centers
- → Click logs on web pages
- → IoT sensors
  - ↳ continuous operation
- → Metrics from mobile phone apps

MOTIVATION



Streaming Video Provider

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

→ unbounded

→ Data might arrive in a different than what was generated

Goals

- Easy to program
- Balance correctness, latency and cost

→ similar to PyTorch, Spark

→ Big challenge in system design

APPROACH

Separate user API from execution

Count views / video every minute, every 5 mins dashboard

Decompose queries into

Count / video - What is being computed

group every minute - Where in time is it computed

every 5 mins - When is it materialized

overwrite dashboard - How does it relate to earlier results

API for streaming queries

System to execute them

data has some ts associated with it

Triggers execution

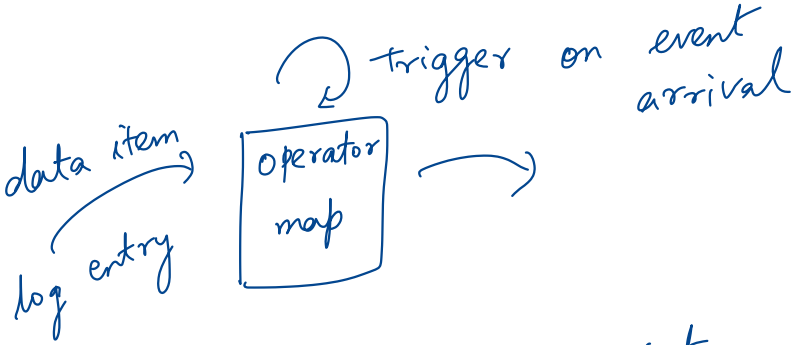
Diff / overlap between results

Select * from users / (map / reduce)

STREAMING VS. BATCH

Streaming

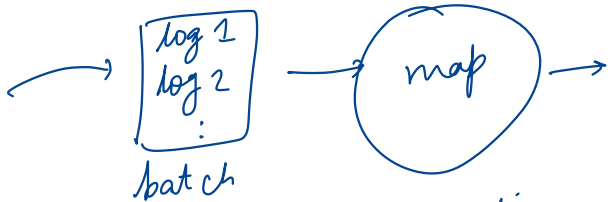
→ Flink, Timely Dataflow



- lower latency of outputs are produced quickly
- more "real time"

Batch

→ MapReduce, Spark, SCOPE



trigger at fixed time interval (5 mins)

- less real time
- more efficient → better tput

TIMESTAMPS

Event time:

↳ Time at which event happened

associated by the sensor / device

Processing time:

↳ Time at which event was processed

by stream processing system

Processing time ≥ event time

WINDOWING → streaming late 90s

query

aligned

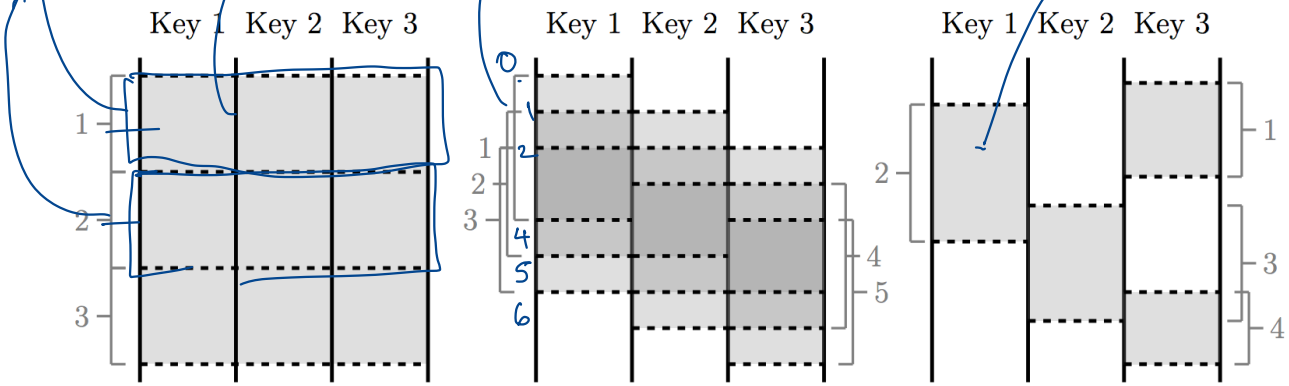
window = 5 mins
slide = 1 min

user

actions within one session

- event time stamp

1 min window



Fixed

or

Tumbling windows

Sliding windows

Sessions → any activity within a time window

WATERMARK OR SKEW

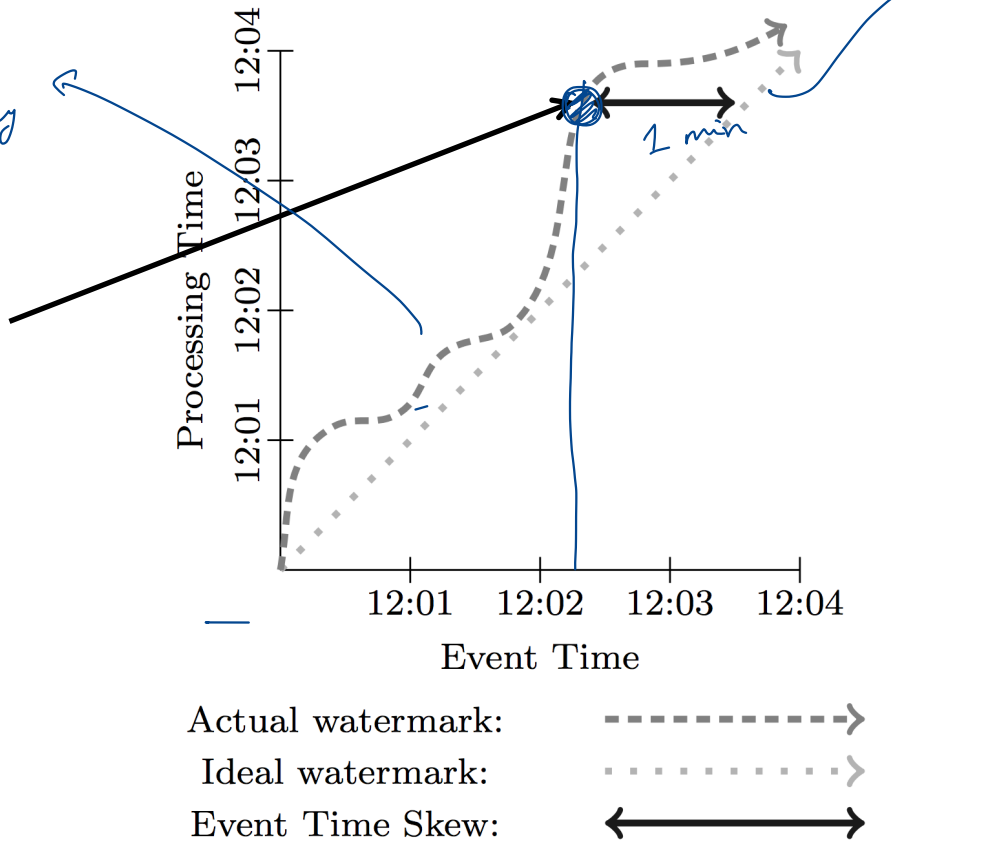
App specific heuristics.

network latency ~ 1 min

Proc = event time

gap between processing time between event time

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



API

ParDo:

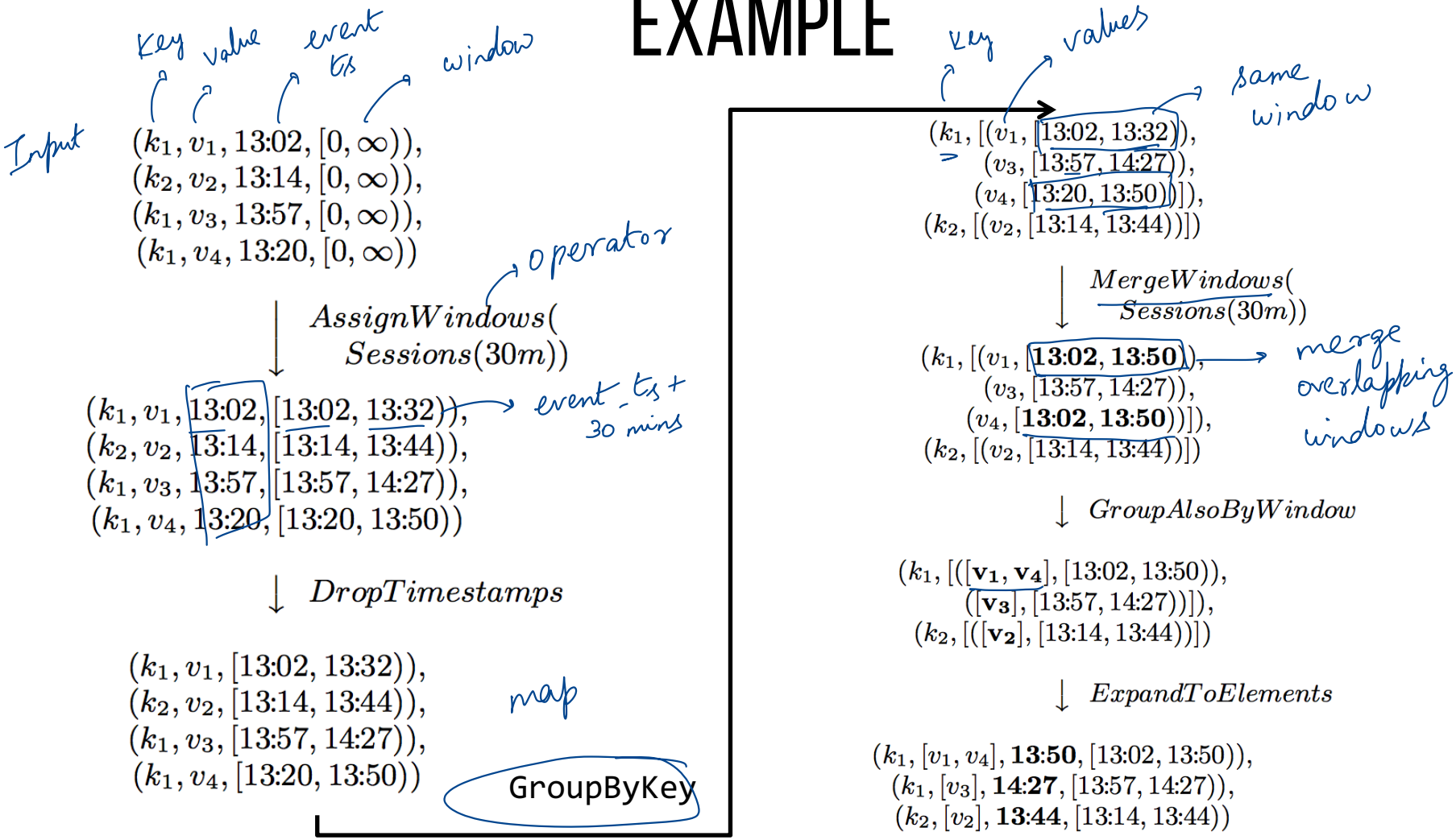
GroupByKey:

Windowing

AssignWindow

MergeWindow

EXAMPLE


key
value
event ts
window
Input
$(k_1, v_1, 13{:}02, [0, \infty))$,
$(k_2, v_2, 13{:}14, [0, \infty))$,
$(k_1, v_3, 13{:}57, [0, \infty))$,
$(k_1, v_4, 13{:}20, [0, \infty))$
operator
$AssignWindows($
$Sessions(30m))$
$(k_1, v_1, 13{:}02, [13{:}02, 13{:}32))$,
$(k_2, v_2, 13{:}14, [13{:}14, 13{:}44))$,
$(k_1, v_3, 13{:}57, [13{:}57, 14{:}27))$,
$(k_1, v_4, 13{:}20, [13{:}20, 13{:}50))$
event_ts + 30 mins
$DropTimestamps$
$(k_1, v_1, [13{:}02, 13{:}32))$,
$(k_2, v_2, [13{:}14, 13{:}44))$,
$(k_1, v_3, [13{:}57, 14{:}27))$,
$(k_1, v_4, [13{:}20, 13{:}50))$
map
GroupByKey
key
values
same window
$(k_1, [(v_1, [13{:}02, 13{:}32)),$
$(v_3, [13{:}57, 14{:}27)),$
$(v_4, [13{:}20, 13{:}50))])$,
$(k_2, [(v_2, [13{:}14, 13{:}44))])$
$MergeWindows($
$Sessions(30m))$
$(k_1, [(v_1, [\mathbf{13{:}02}, \mathbf{13{:}50})),$
$(v_3, [13{:}57, 14{:}27)),$
$(v_4, [\mathbf{13{:}02}, \mathbf{13{:}50}))])$,
$(k_2, [(v_2, [13{:}14, 13{:}44))])$
merge overlapping windows
$GroupAlsoByWindow$
$(k_1, [([\mathbf{v_1}, \mathbf{v_4}], [13{:}02, 13{:}50)),$
$([\mathbf{v_3}], [13{:}57, 14{:}27))])$,
$(k_2, [([\mathbf{v_2}], [13{:}14, 13{:}44))])$
$ExpandToElements$
$(k_1, [v_1, v_4], \mathbf{13{:}50}, [13{:}02, 13{:}50))$,
$(k_1, [v_3], \mathbf{14{:}27}, [13{:}57, 14{:}27))$,
$(k_2, [v_2], \mathbf{13{:}44}, [13{:}14, 13{:}44))$

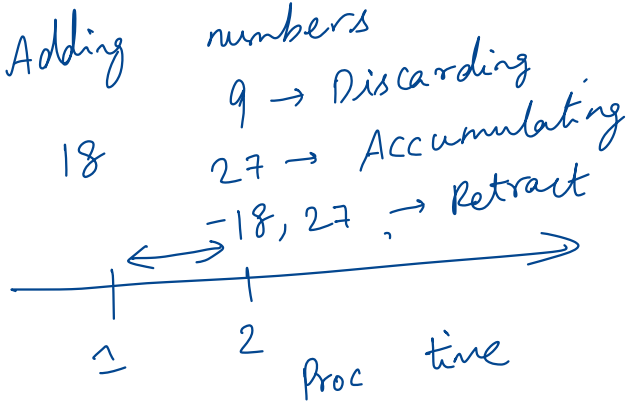
TRIGGERS AND INCREMENTAL PROCESSING

Windowing: where in event time are data grouped

Triggering: when in processing time are groups emitted

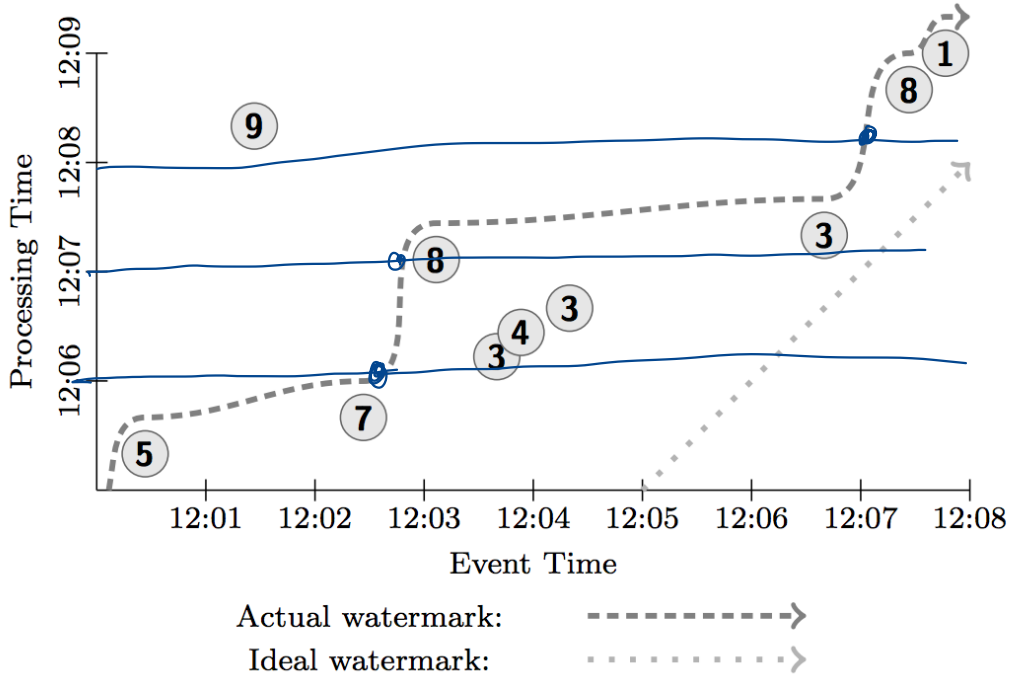
Strategies

- Discarding
- Accumulating
- Accumulating & Retracting



RUNNING EXAMPLE

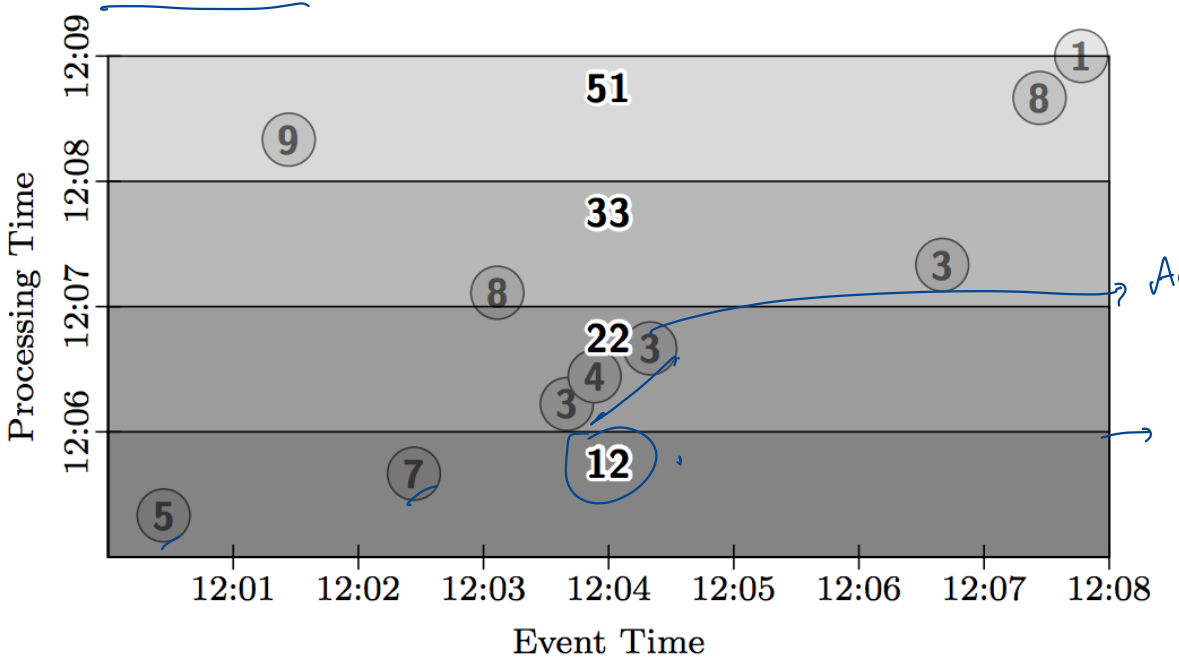
```
PCollection<KV<String, Integer>> input = IO.read(...);
PCollection<KV<String, Integer>> output =
        input.apply(Sum.integersPerKey());
```



GLOBAL WINDOWS, ACCUMULATE

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

tumbling window every 1 min



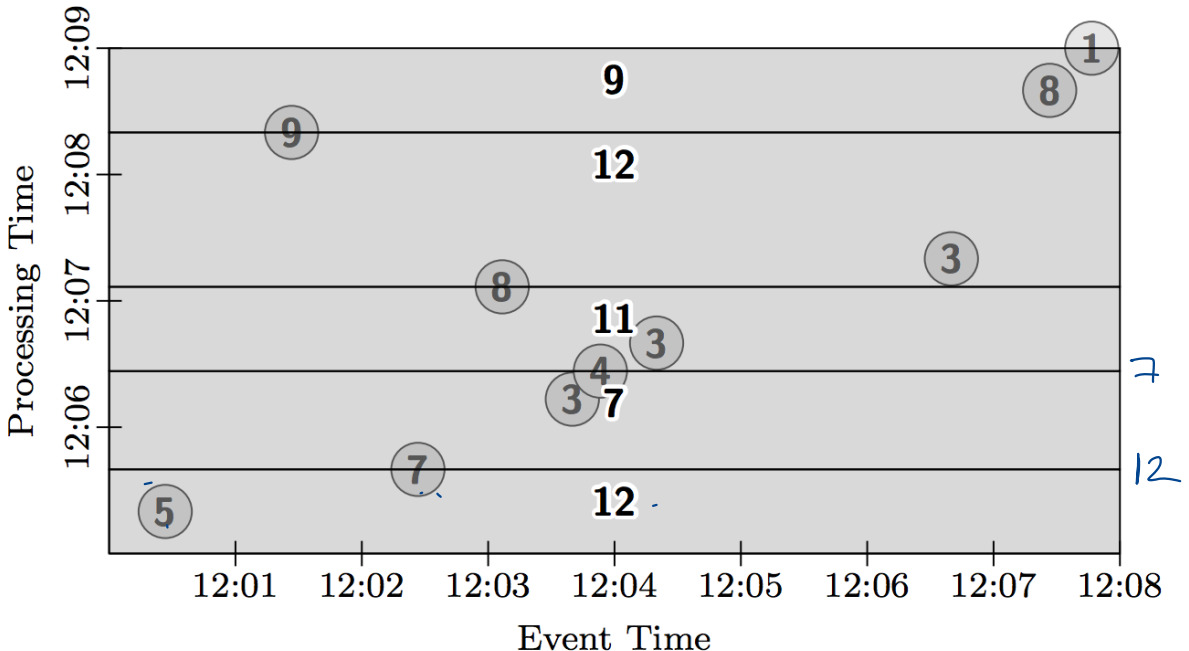
Accumulating

every 1 min trigger

GLOBAL WINDOWS, COUNT, DISCARDING

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

two keys

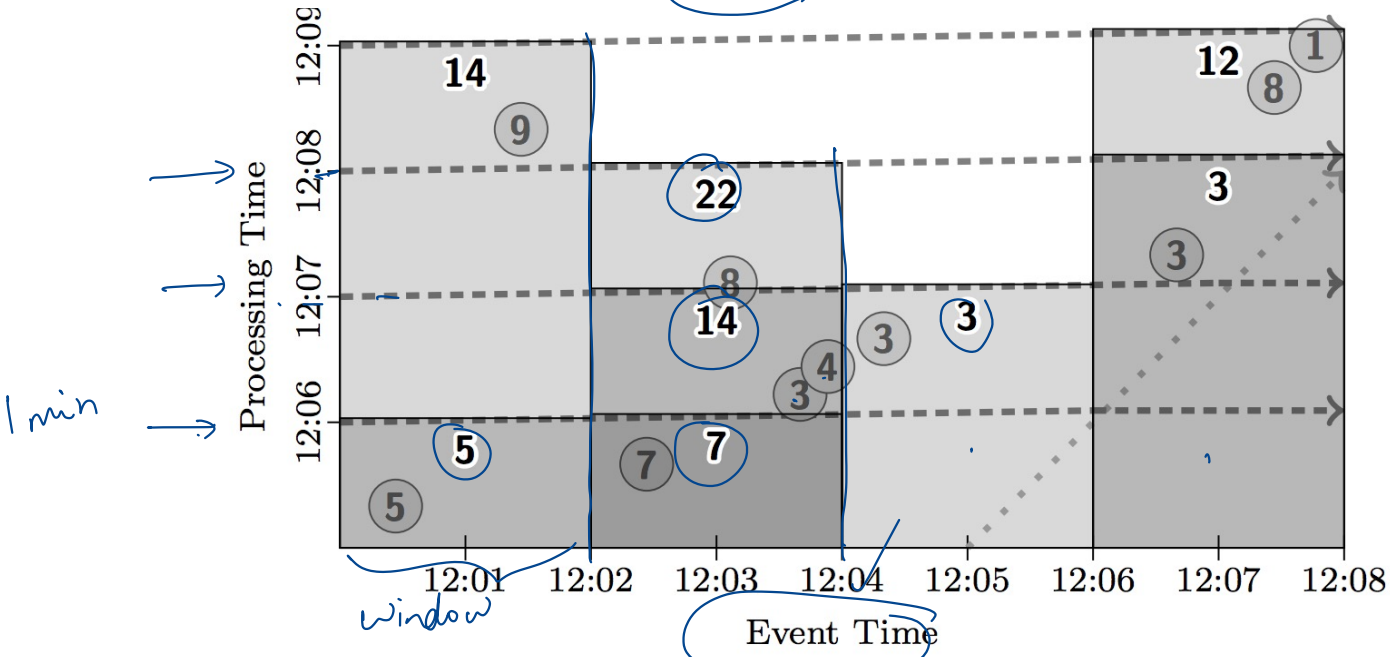


FIXED WINDOWS, MICRO BATCH

triggered every 1 min

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

watermark



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

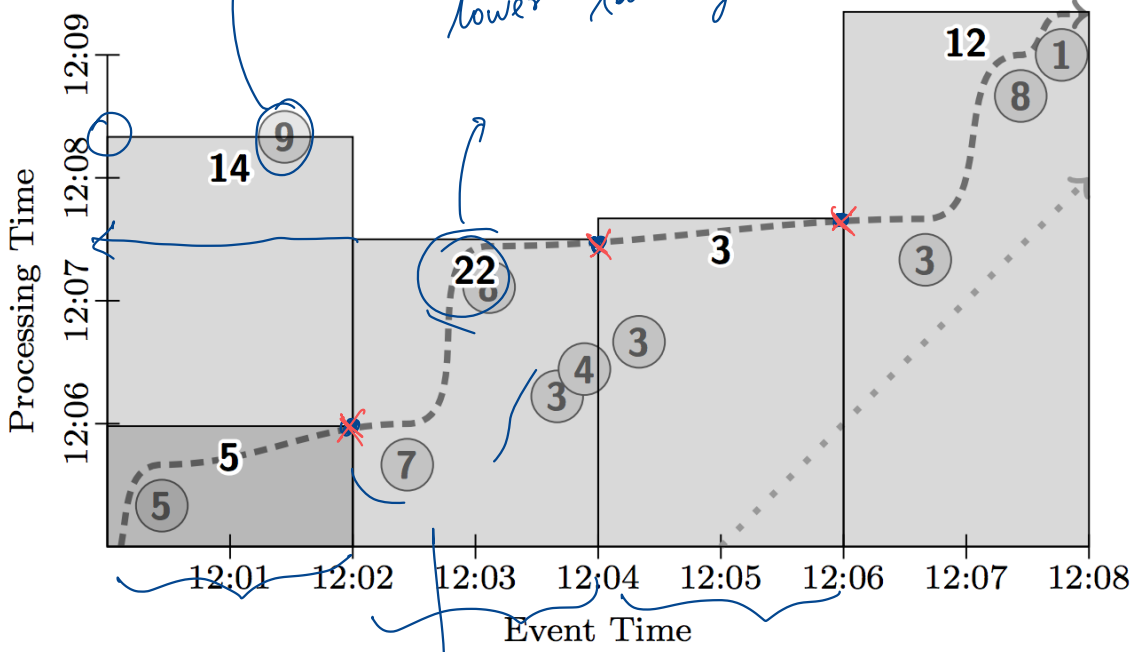
DISCUSSION

https://forms.gle/TB5kz2cH3uYc6rjv6

Trigger a window when watermark crosses the window

outside the watermark → heuristic

lower latency



Value 22 appear at 12:07:30

At 12:08 with mini batch

with micro-batch, we get partial results at 1 min interval

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: Apache Flink