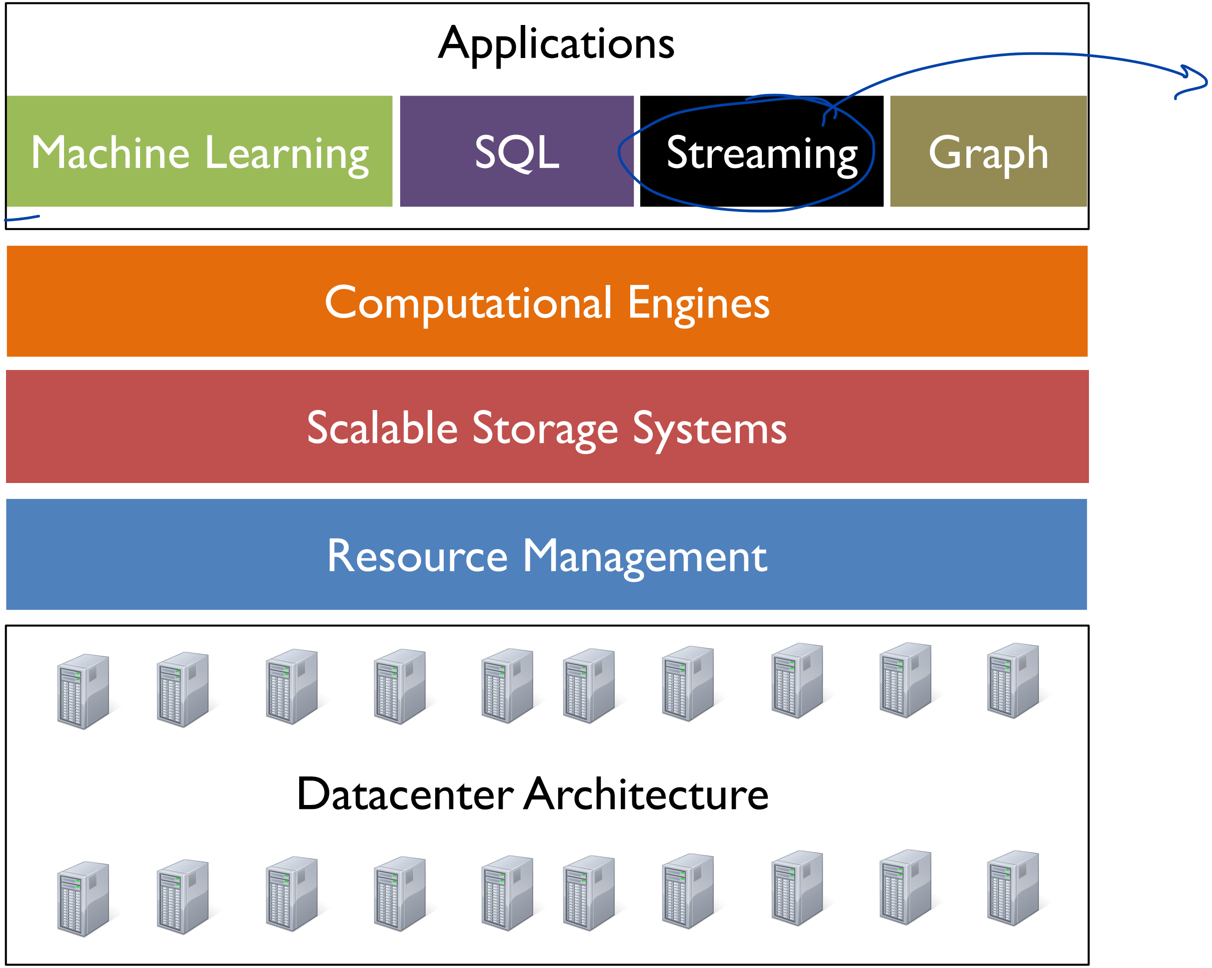
CS 744: DATAFLOW

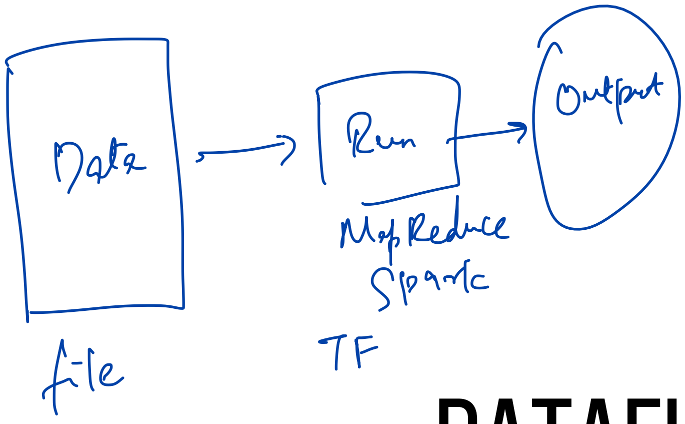
Shivaram Venkataraman

Fall 2019

ADMINISTRIVIA

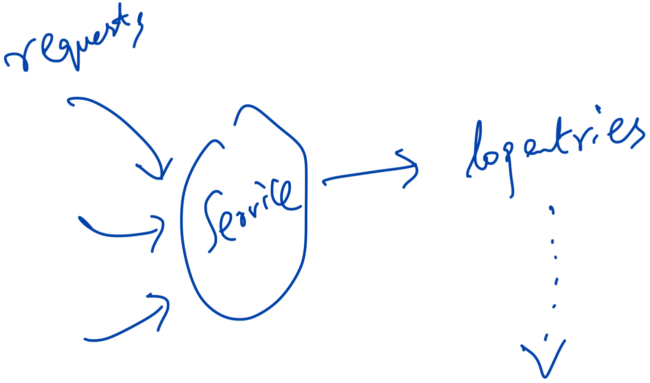
- Assignment 2 grades up
- Midterm grading
- Course project proposal comments
- AEFIS feedback
  - Slides
  - Discussion
- No Class next Tuesday?


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



Static dataset

DATAFLOW MODEL (?)



stream dataset

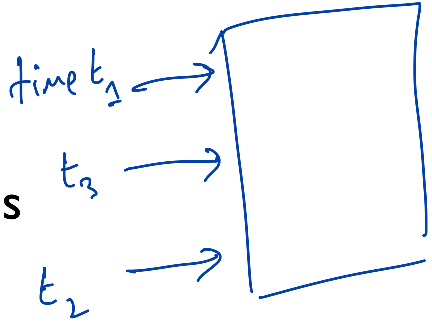
user1 : < videos user has watched >

Correctness vs. latency

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

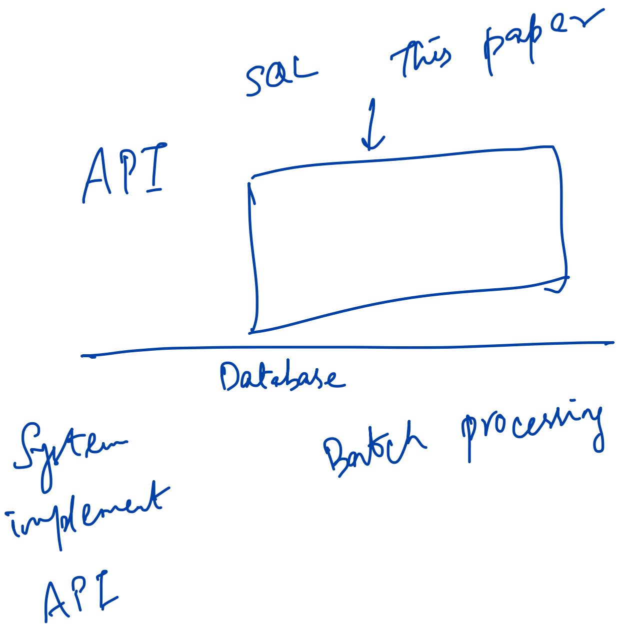
APPROACH

API Design

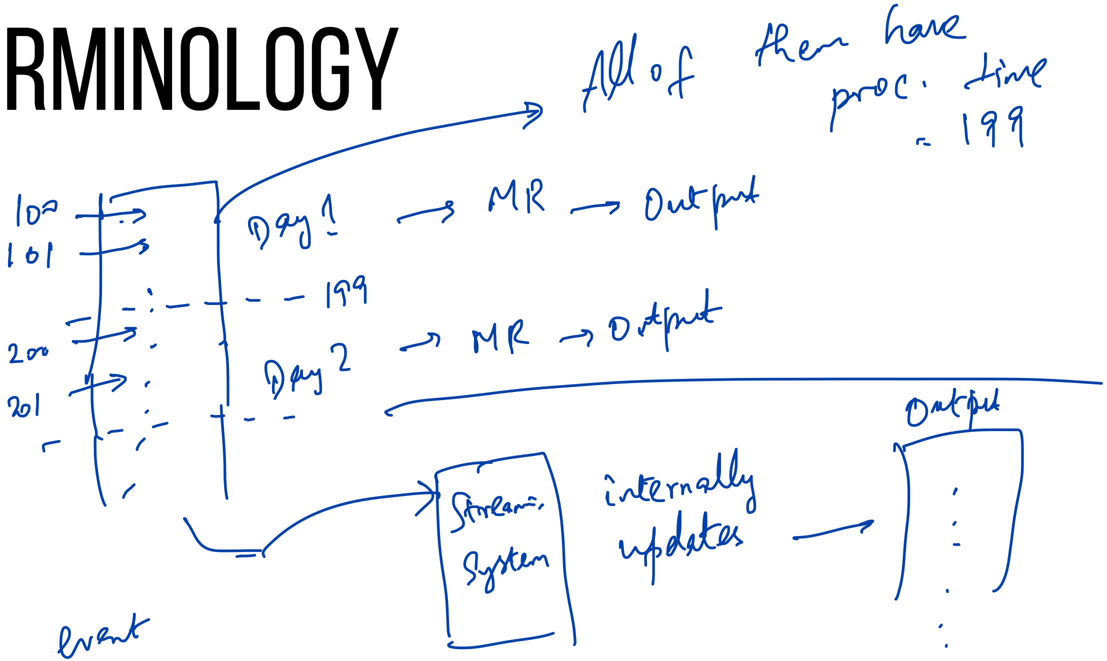
Separate user-facing model from execution

Decompose queries into

- What is being computed — Data
- Where in time is it computed — Scheduling
- When is it materialized — Output
- How does it relate to earlier results — whether you are incrementing or accumulating



TERMINOLOGY



Unbounded/bounded data

Streaming/Batch execution

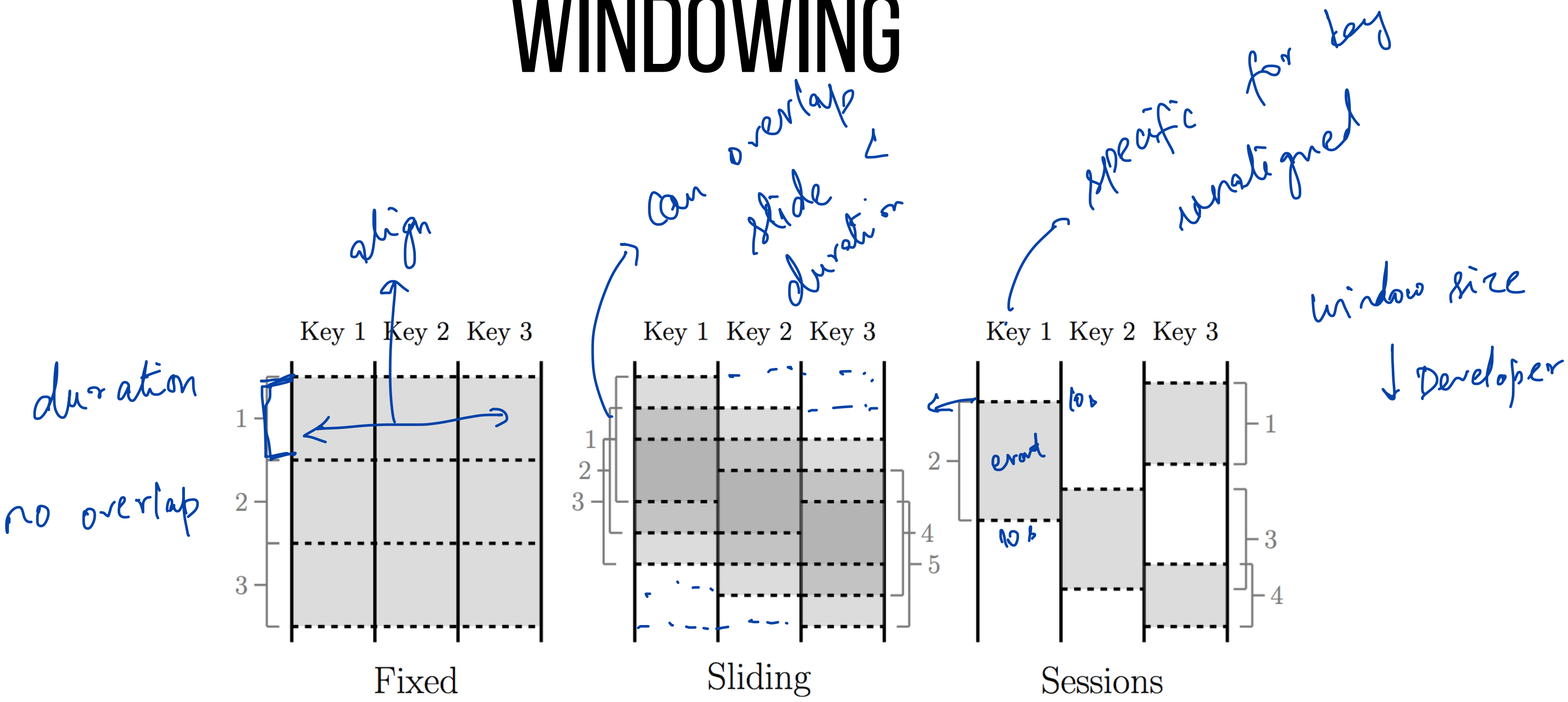
Timestamps

Event time: Time when event occured wrt world input

Processing time:

Time at which event is processed

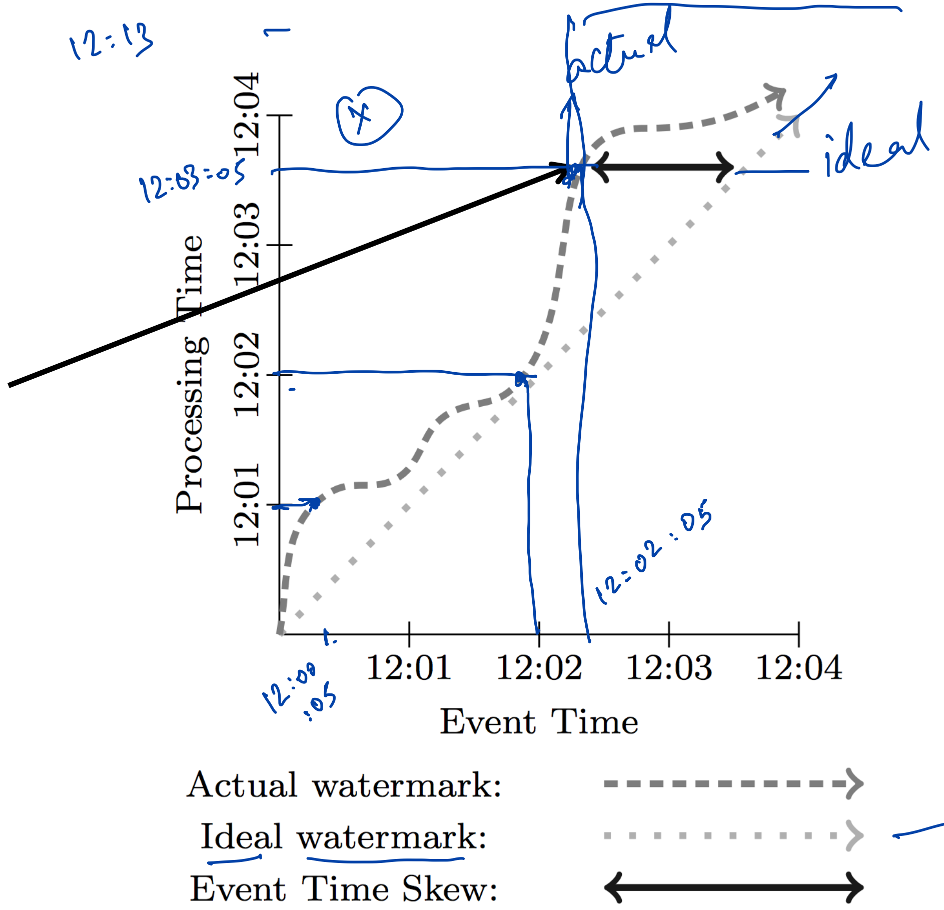
WINDOWING



Key 1
Key 2
Key 3
1
2
3
Fixed
Sliding
4
5
Sessions
duration
no overlap
align
can overlap
slide
duration
specific for key
unaligned
window size
Developer
event
10s
10s

WATERMARK OR SKEW

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



Processing time lags event time

→ Time out

→ Count

→ event = proc. time

API

ParDo: map | flat Map

Input $k, v \rightarrow$ (k_1, v_1) (k_2, v_2) ...

GroupByKey: reduce

$k_1 \rightarrow (v_1, v_2, v_3 \ldots)$

Windowing

AssignWindow → Bucket tuples into window

MergeWindow → Merge buckets based on strategy

EXAMPLE

Event time   Window

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

$\downarrow$ *AssignWindows(Sessions(30m))*

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

$\downarrow$ *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

start   overlap   end   normal group By Key

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

$\downarrow$ *MergeWindows(Sessions(30m))*

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

$\downarrow$ *GroupAlsoByWindow*

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

$\downarrow$ *ExpandToElements*

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

timestamp

query

TRIGGERS AND INCREMENTAL PROCESSING

Windowing: where in event time data are grouped

Triggering: when in processing time groups are emitted

Timestamp

10 min window

every 1 min

Strategies

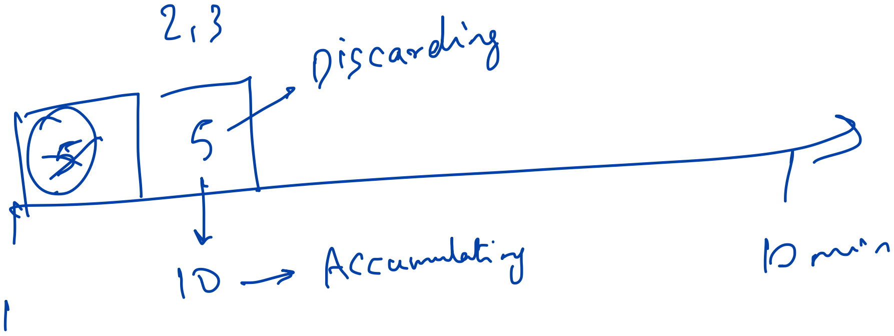
Discarding

Accumulating

Accumulating & Retracting

State: 2,3

5



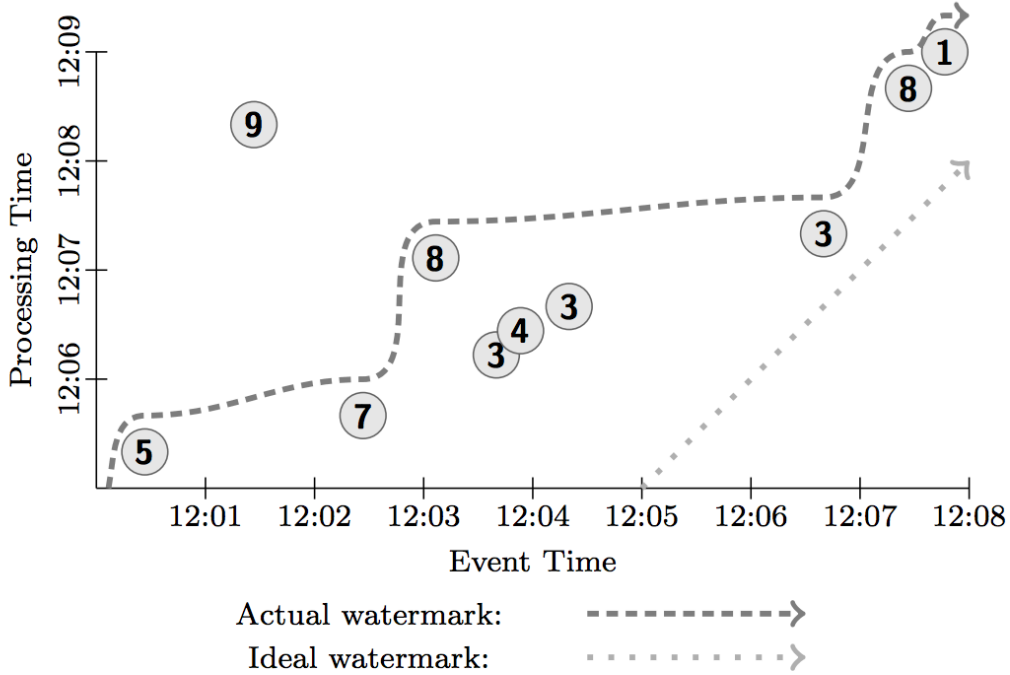
old state, updates

new state

-5, 10

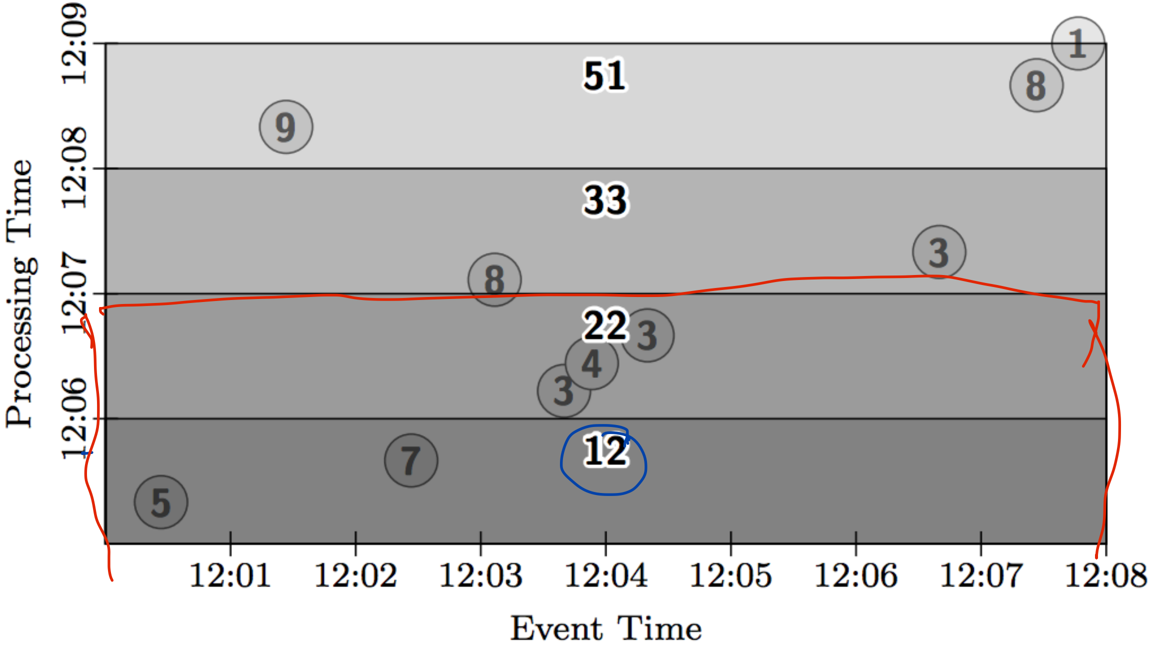
RUNNING EXAMPLE

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



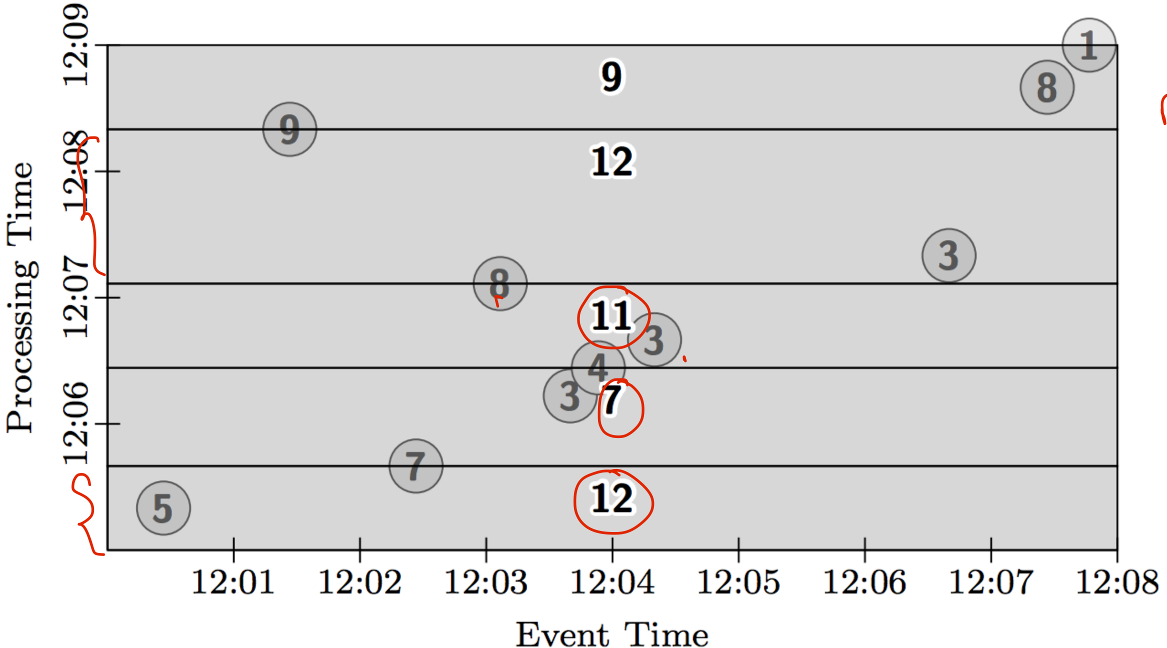
GLOBAL WINDOWS, ACCUMULATE

```
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());
```



not uniform in time

FIXED WINDOWS, MICRO BATCH

Output

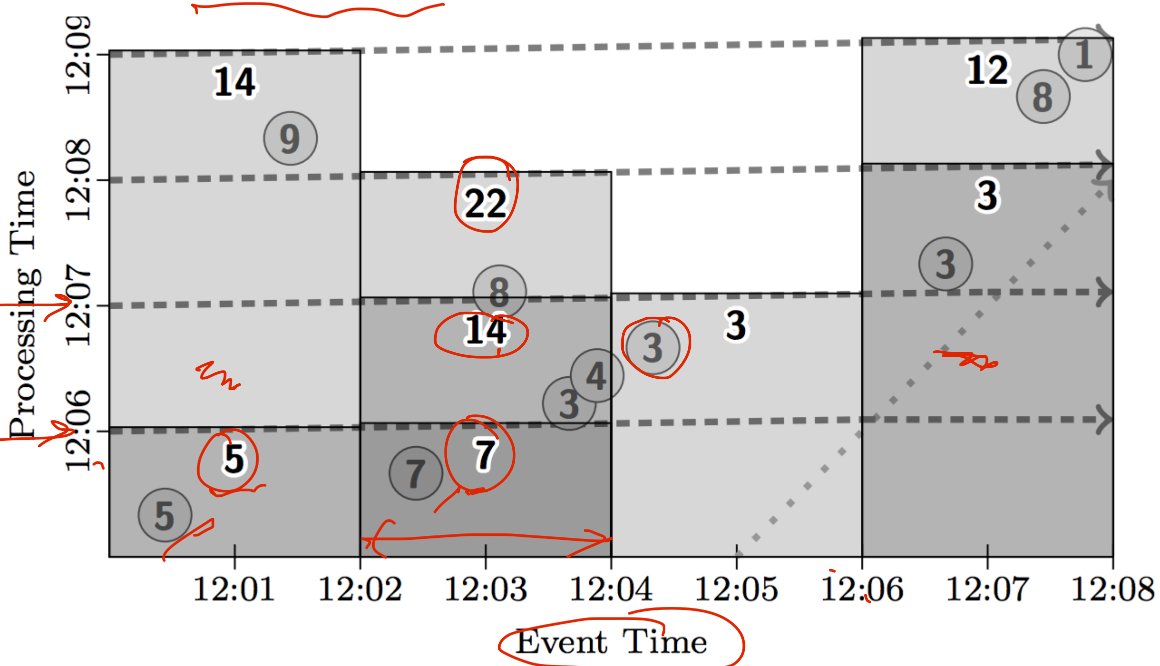
```
PCollection<KV<String, Integer>> output = input
    .apply(Window.into(FixedWindows.of(2, MINUTES))
                 .trigger(Repeat(AtWatermark()))
                 .accumulating())
```

12:00 - 12:02 5

12:02 - 12:04 ~~14~~ 22

12:04 - 12:06 3

2



LESSONS / EXPERIENCES

Don’t rely on completeness

Be flexible, diverse use cases

- Billing
- Recommendation
- Anomaly detection

Support analysis in context of events

DISCUSSION

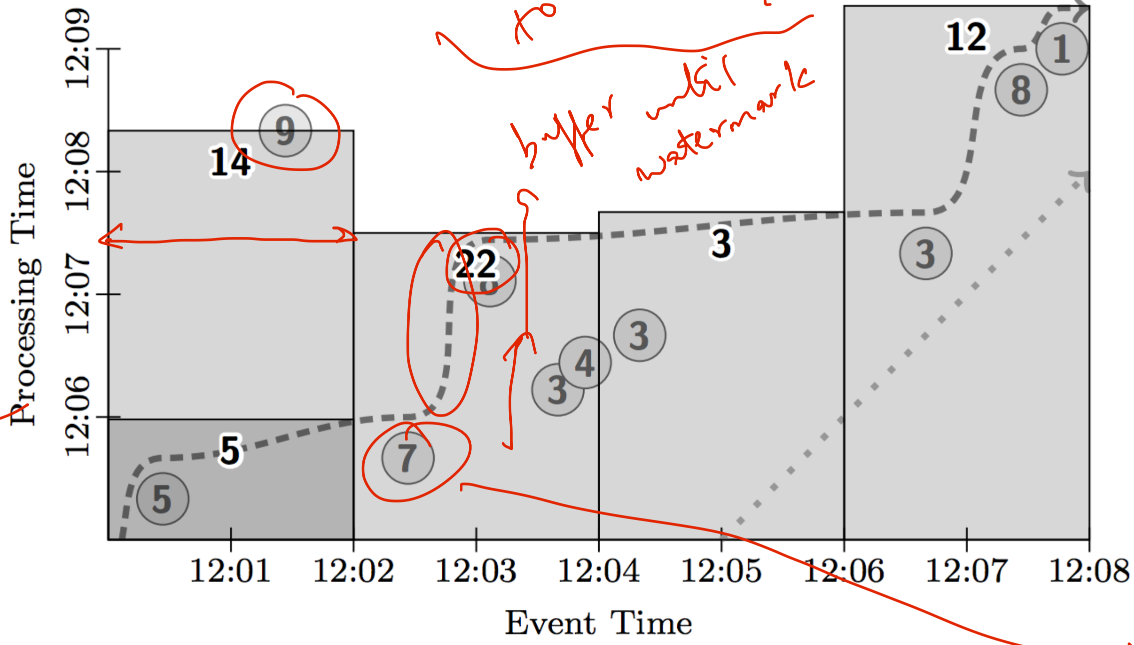
https://forms.gle/s7T2r67BDvkGQhmN9

Computation overheads?

Number of windows to ~~upd~~ update lower for streaming

Batch
22
→ 12:08

Number of outputs is more



buffer until watermark

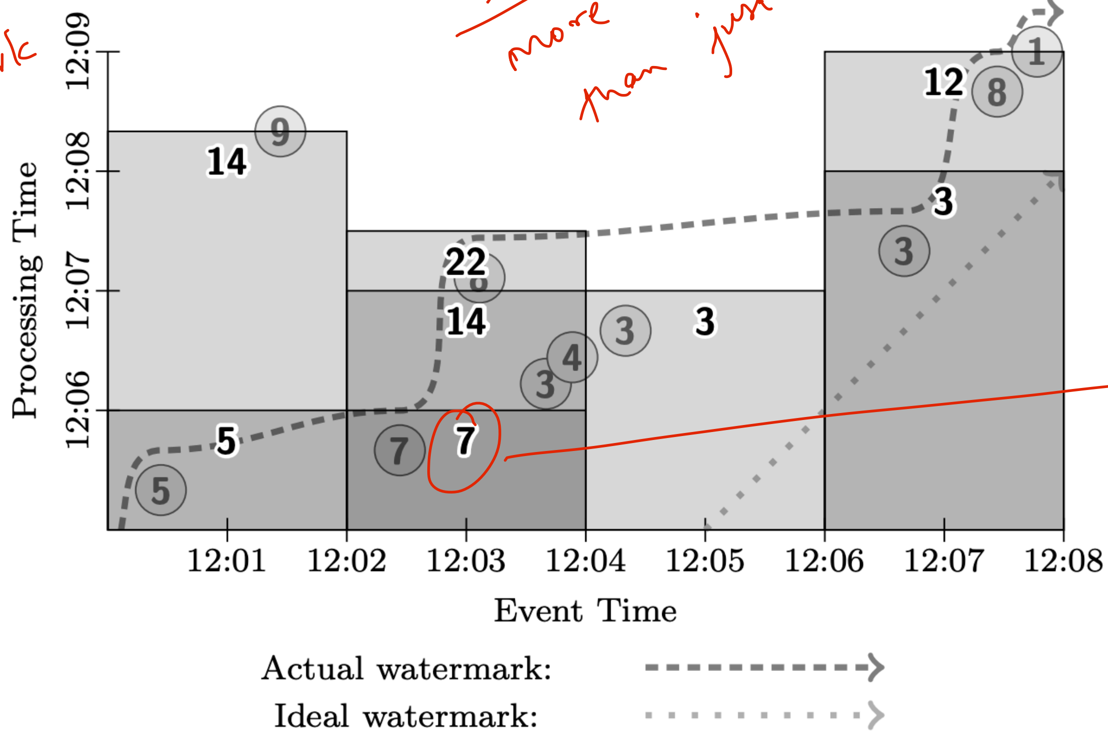
lower latency watermark
22 → 12:07:30

no partial results until watermark

is delayed till 12:07:30

Trigger is composite
→ Time + watermark

Both worlds
more computation than just watermark



Latency?

Partial results

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?