

Hello!

CS 744: DATAFLOW

Shivaram Venkataraman

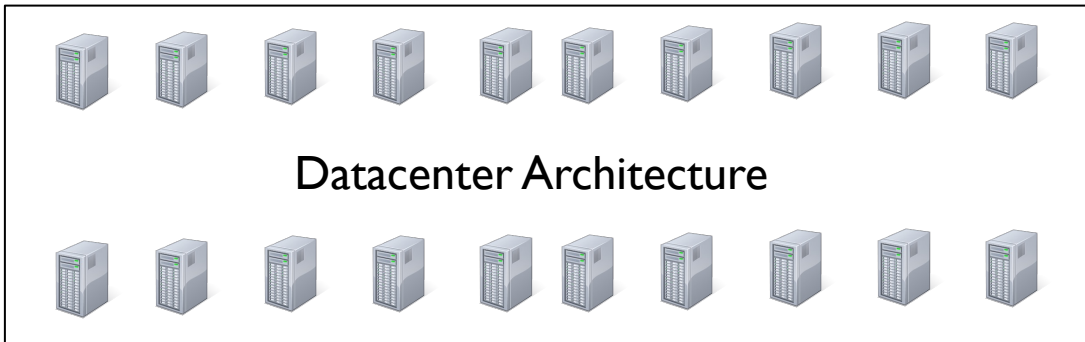
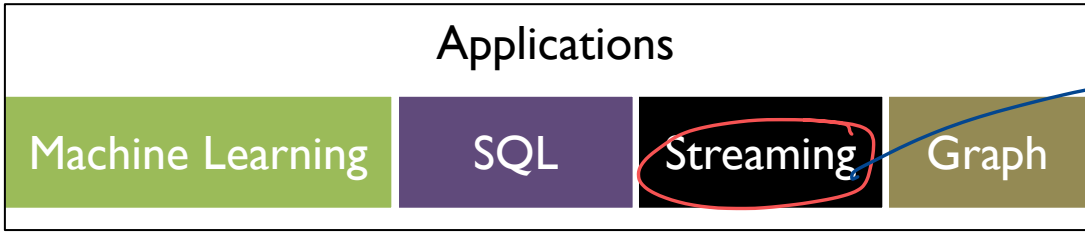
Spring 2025

ADMINISTRIVIA

Grading In Progress

- Course project proposal → prioritize
- Assignment 2 ~ 90% done
- Midterm
↳ after spring break

MID-SEMESTER FEEDBACK



Streaming query
(Dataflow model)
↓ ↓
Flink Spark
Streaming

Handwritten notes in blue ink. An arrow points from the 'Streaming' block in the Applications layer to the text 'Streaming query'. Below it, '(Dataflow model)' is written. Two arrows point down from '(Dataflow model)' to 'Flink' and 'Spark Streaming'.

Spark

Dataflow



Data from natural sources

→ weather, satellite data

→ sensors → temperature

DATAFLOW MODEL (?)

Unbounded data

→ logs and log processing → System logs

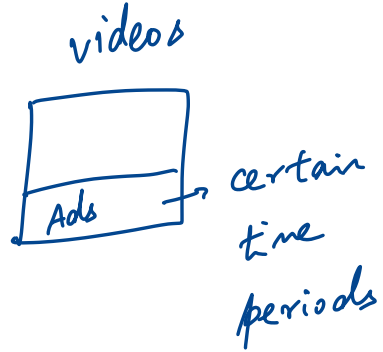
→ clickstream → website / ad-supported

→ videos, gathering data on viewership

MOTIVATION

Streaming Video Provider

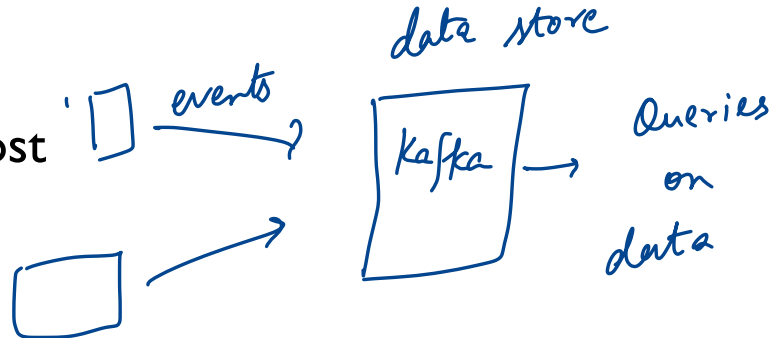
- How much to bill each advertiser ?
- Need per-user, per-video viewing sessions
- Handle out of order data



timestamp when event happens

Goals

- Easy to program
- Balance correctness, latency and cost

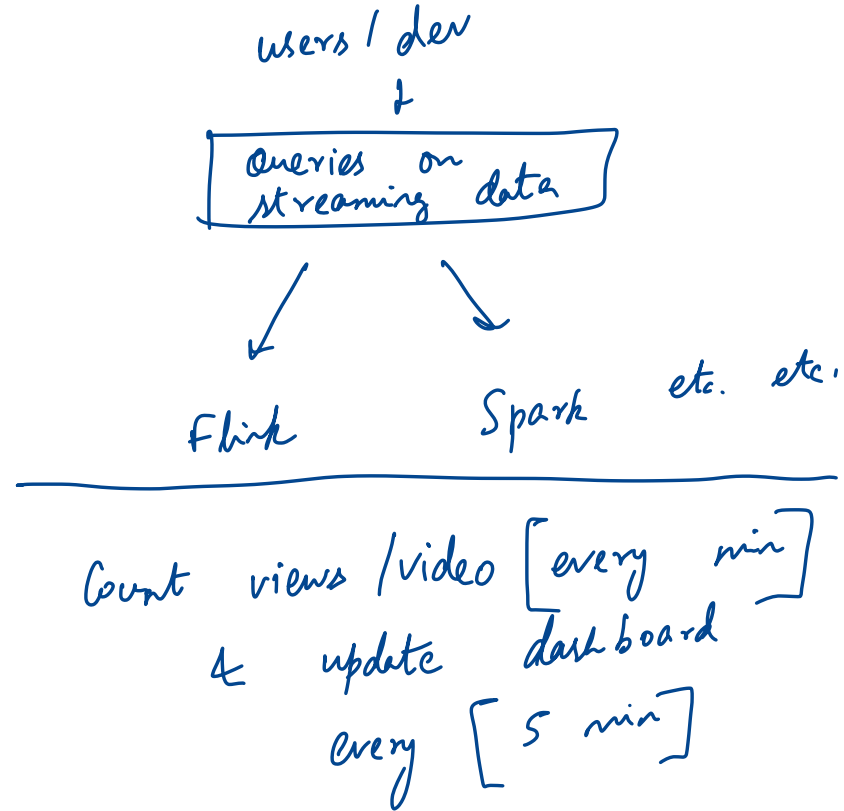


APPROACH

Separate user API from execution

Decompose queries into

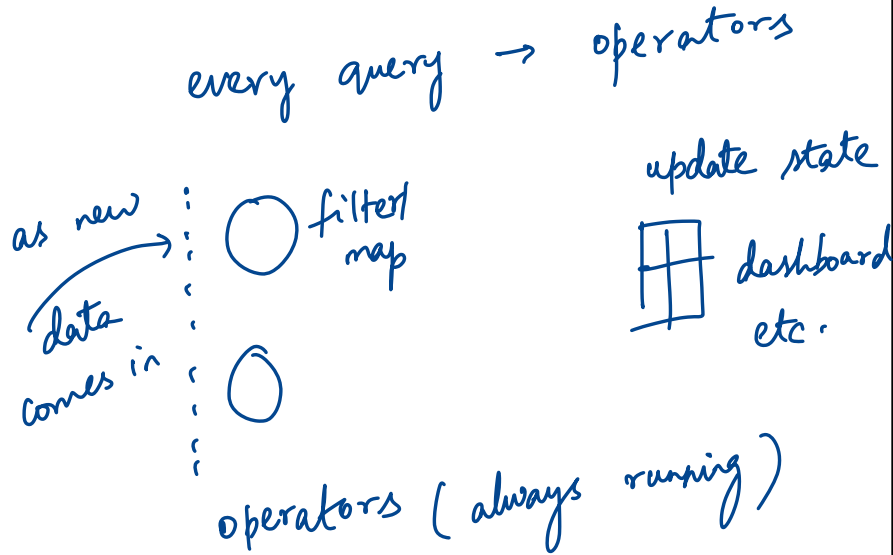
- What is being computed
 - Where in time is it computed
 - When is it materialized
 - How does it relate to earlier results
- Count / video
every min
every 5 min
- update dashboard



STREAMING VS. BATCH

Streaming

→ Flink, Naiad



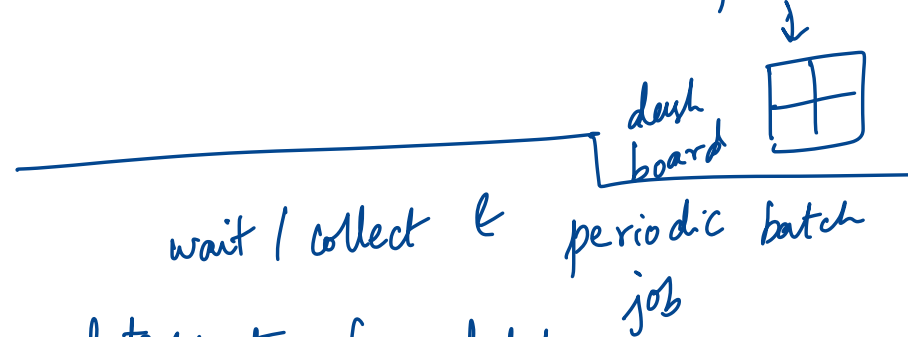
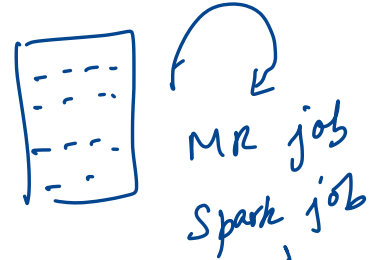
→ low latency

→ lower kput

Batch

→ Spark, Flume Java

→ batch or collect data



→ latency to form batch

→ vectorized / more parallelism

TIMESTAMPS

Event time:

↳ time at which event happens

→ e.g. ad was clicked

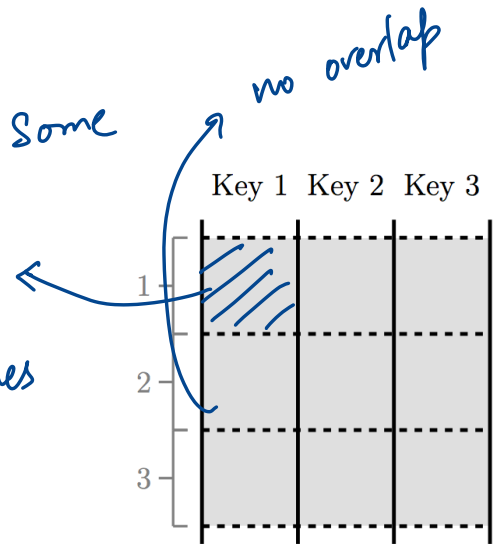
Processing time:

↳ time at which event was processed

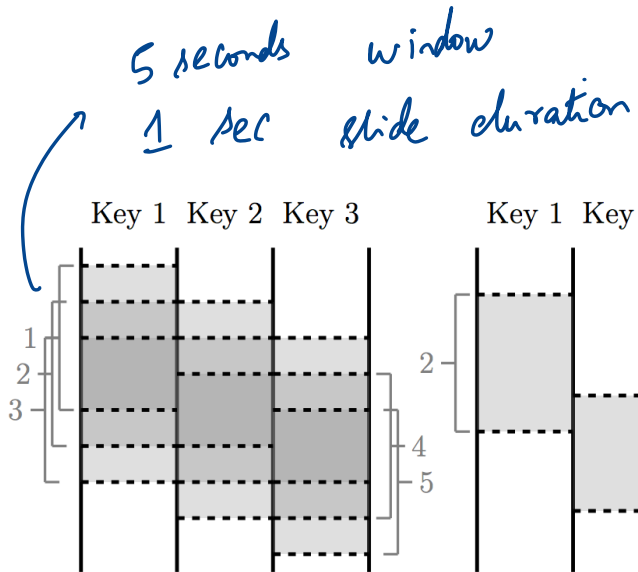
Processing time > event time

WINDOWING

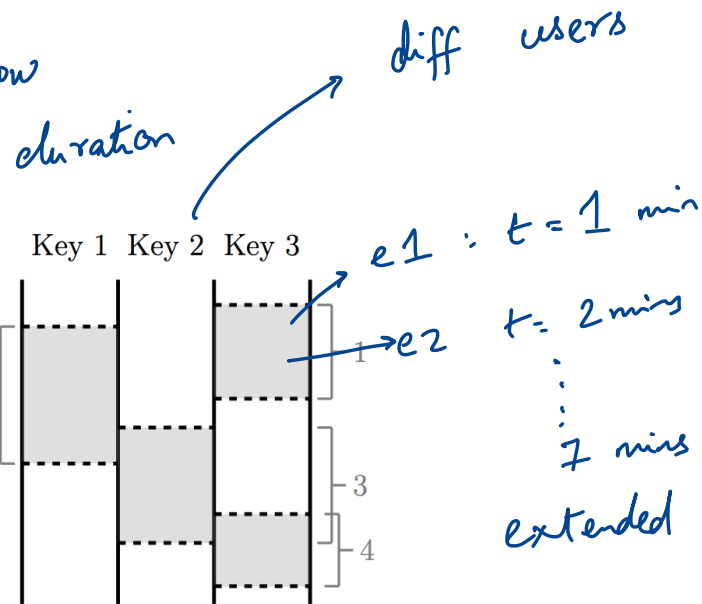
apply function to all tuples inside time range



Fixed
or
Tumbling
window



$[0, 5)$
 $[1, 6)$
.....
overlap



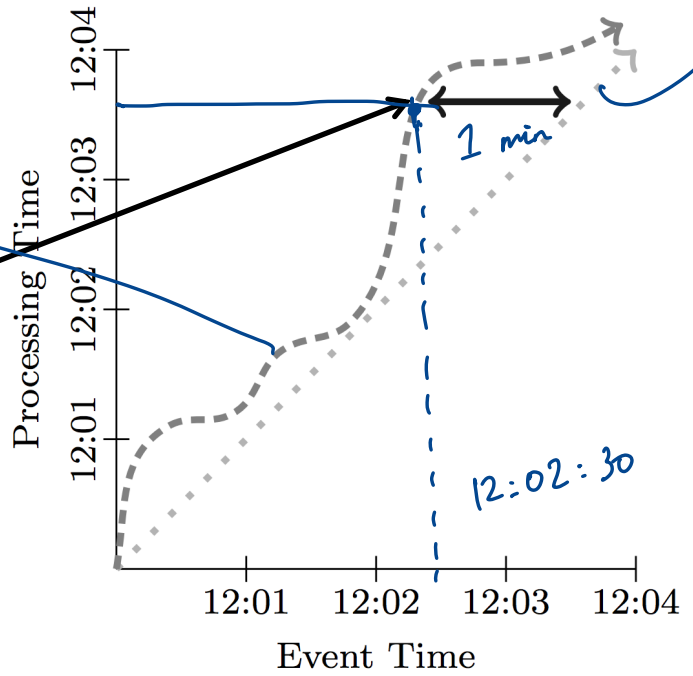
Sessions
any event within 5 mins are in same session

WATERMARK OR SKEW

heuristics for
estimating
events
arrival

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

12:03:30



proc = event
time

gap between
event time
& processing
time

- Actual watermark: ----->
- Ideal watermark:>
- Event Time Skew: <----->

API

ParDo:

GroupByKey:

Windowing

AssignWindow

MergeWindow

EXAMPLE

input $\begin{matrix} \nearrow & \nearrow & \nearrow & \nearrow \\ \text{key} & \text{value} & \text{event ts} & \text{window} \end{matrix}$

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

\downarrow AssignWindows
Sessions(30m)

$(k_1, v_1, \del{13:02}, [13:02, 13:32)),$ \rightarrow 30mins from
 $(k_2, v_2, \del{13:14}, [13:14, 13:44)),$ when
 $(k_1, v_3, \del{13:57}, [13:57, 14:27)),$ event
 $(k_1, v_4, 13:20, [13:20, 13:50))$ happening

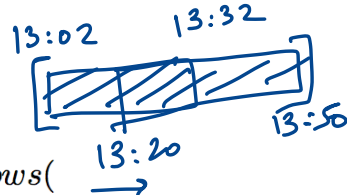
\downarrow DropTimestamps

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

Spark/
MR
 \nearrow
GroupByKey

key : Array values

$(k_1, [(v_1, [13:02, 13:32)),$
 $(v_3, [13:57, 14:27)),$ 1
 $(v_4, [13:20, 13:50))]),$
 $(k_2, [(v_2, [13:14, 13:44))])$



\downarrow MergeWindows(
Sessions(30m))

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

\downarrow GroupAlsoByWindow

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

\downarrow ExpandToElements

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

TRIGGERS AND INCREMENTAL PROCESSING

Windowing: **where** in event time are data grouped

Triggering: **when** in processing time are groups emitted

Strategies

Discarding

Accumulating

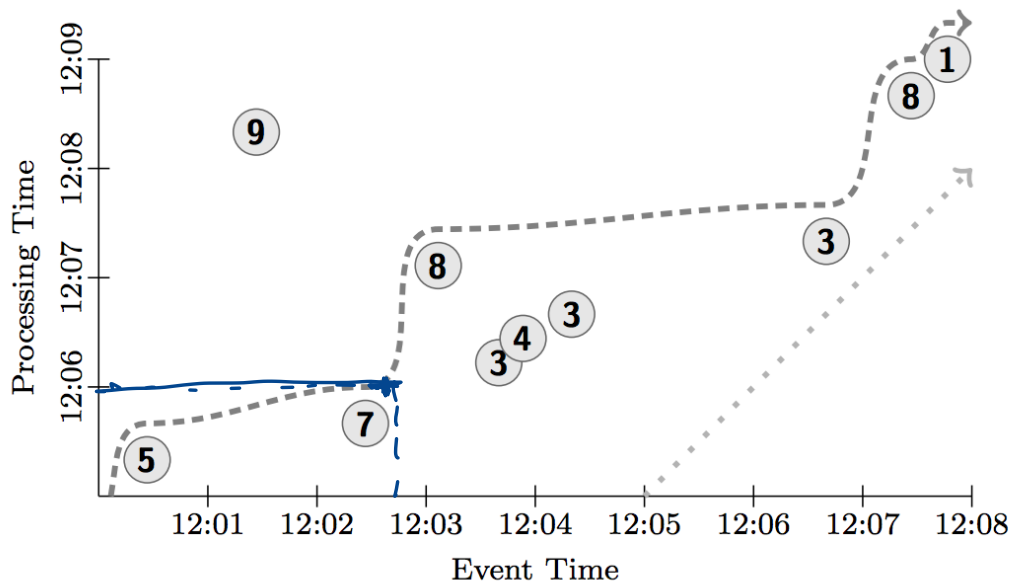
Accumulating & Retracting

when should you compute

how to relate prev output

RUNNING EXAMPLE

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



Actual watermark: ----->
Ideal watermark: >

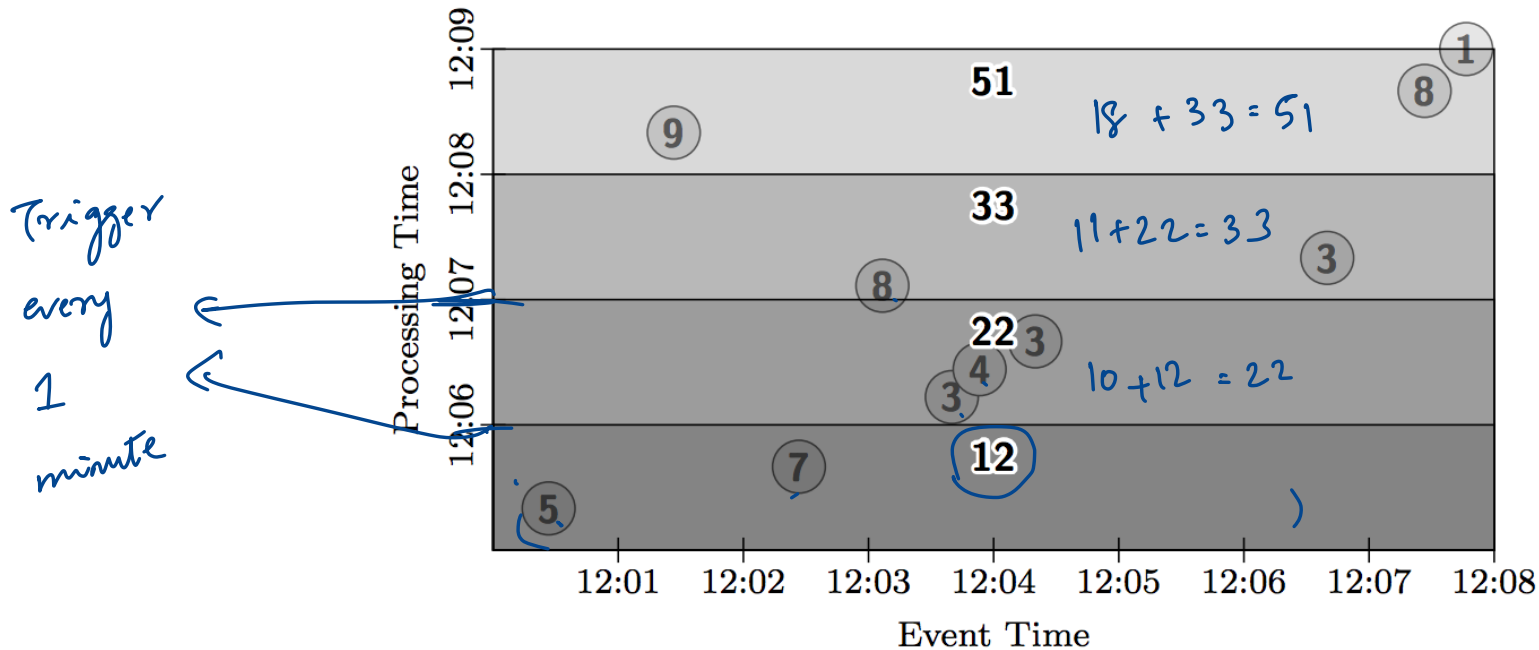
GLOBAL WINDOWS, ACCUMULATE

```
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
```

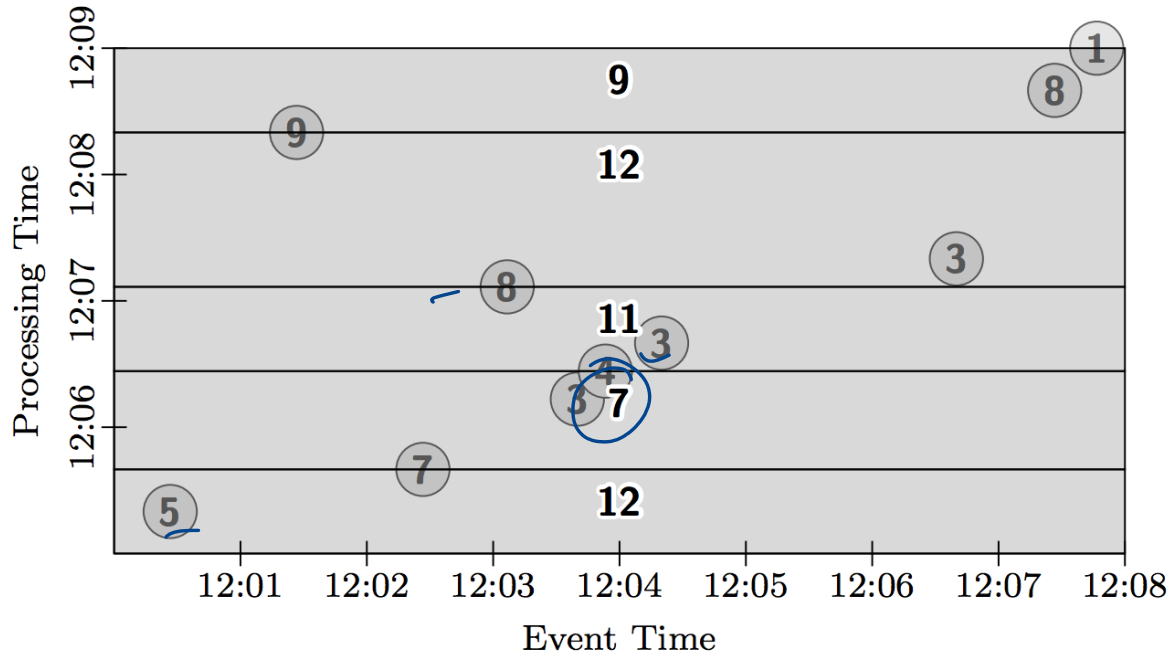
```
.apply(Window.trigger(Repeat(AtCount(2))))
```

```
.discarding();
```

```
.apply(Sum.integersPerKey());
```

*every two events
→ trigger*

*don't care
about prev
output*



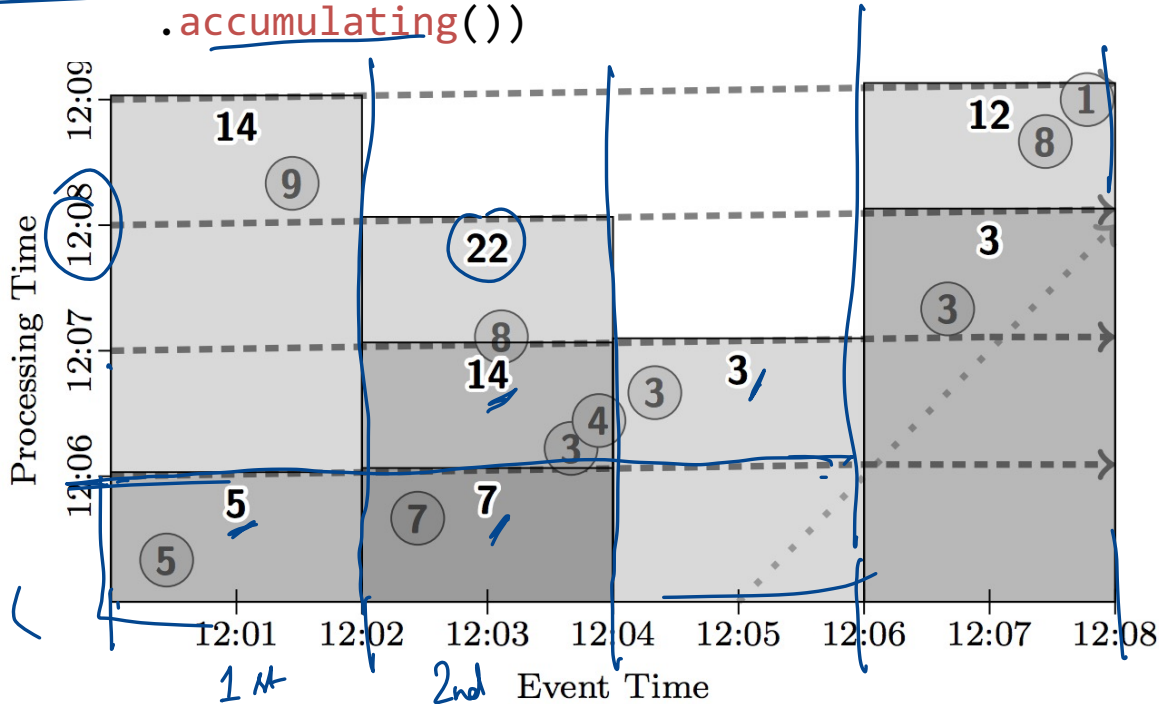
FIXED WINDOWS, MICRO BATCH

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

Tumbling

event time

*whenever
event time
crosses
processing
time*



SUMMARY/LESSONS

Design for unbounded data: Don't rely on completeness

Be flexible, diverse use cases

- Billing
- Recommendation
- Anomaly detection

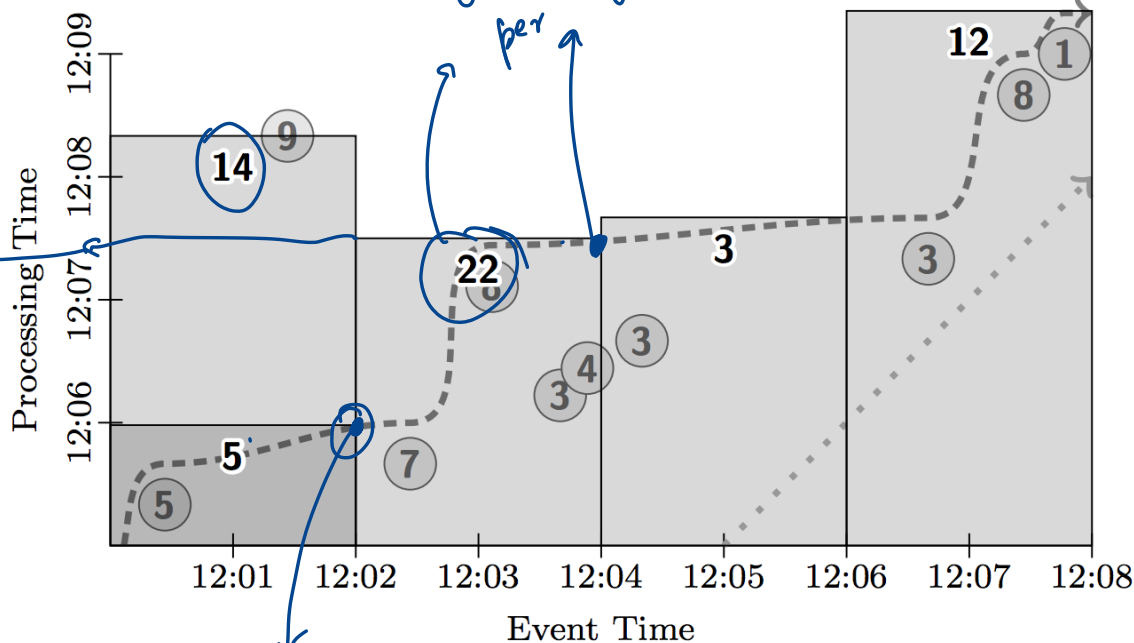
Windowing, Trigger API to simplify programming on unbounded data



DISCUSSION

<https://forms.gle/xzz2XjqENTdrbZ5F6>

Streaming



22 appears at 12:07:30

Window boundaries

mostly one output window per window →

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